اصول علم ربات – اسلاید سیزدهم

Fundamentals of Robotics – Slide 13

Navigation in Presence of Obstacles
Maps, Local Maps

دکتر مهدی جوانمردی

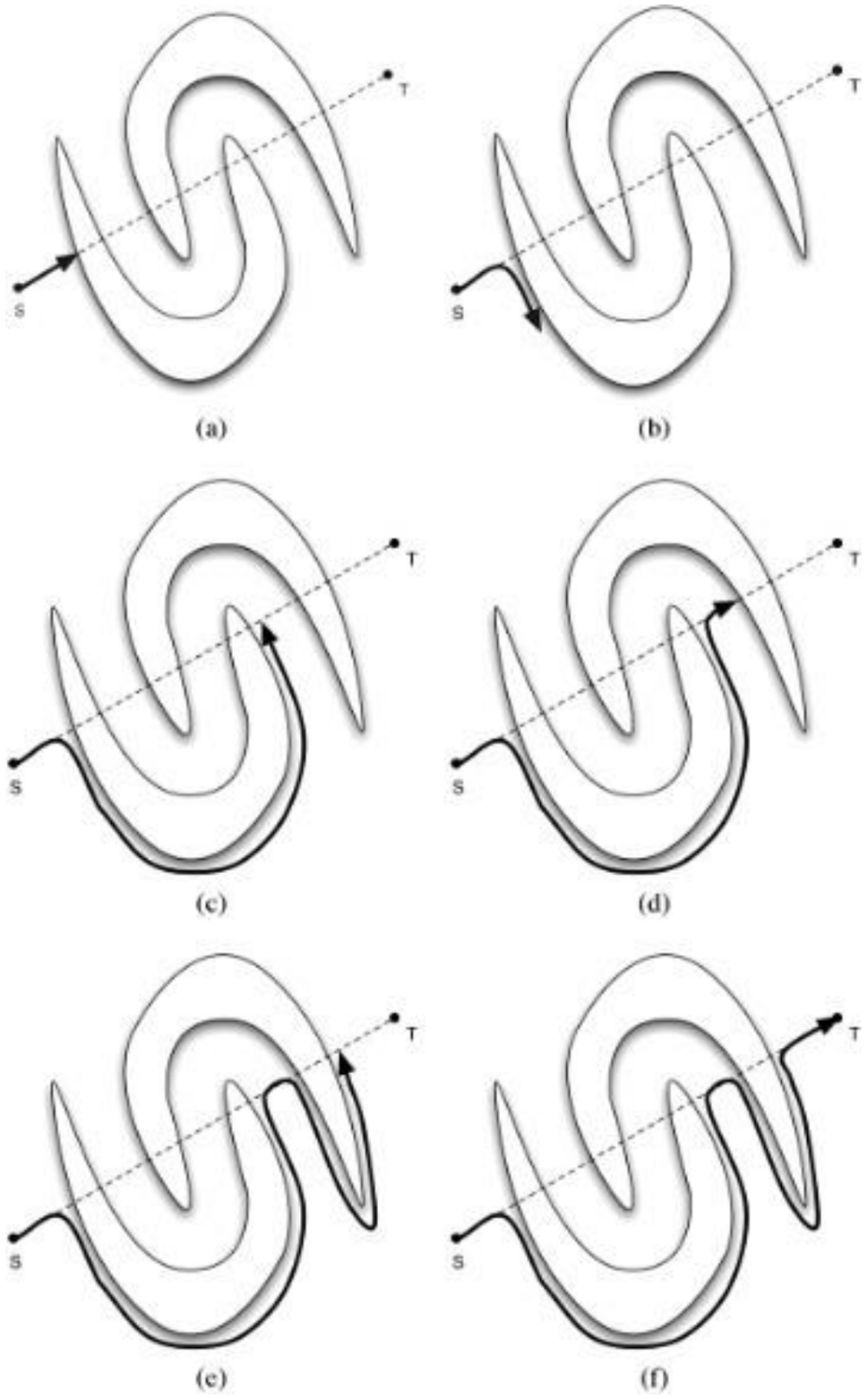
زمستان–بهار ۱۴۰۱

[slides adapted from Gianni Di Caro, @CMU with permission]

Behaviors for moving to a point in presence of obstacles

Bug 1 and Bug 2
Insect-inspired Algorithms (1987)

- **Local decisions, map-free!**
- **Complete algorithms**: find a path if it exists, otherwise returns with a failure.
- **No guarantees** in terms of finding a good (short) path!
- "Nasty" obstacle structure and placement or "wrong" left/ right turning choices can make things going arbitrarily bad.



What's Special About Bugs

- Many planning algorithms assume global knowledge
- Bug algorithms assume only *local* knowledge of the environment and a global goal
- Bug behaviors are simple:
  - 1) Follow a wall (right or left)
  - 2) Move in a straight line toward goal
- Bug 1 and Bug 2 assume essentially tactile sensing
- Tangent Bug deals with finite distance sensing

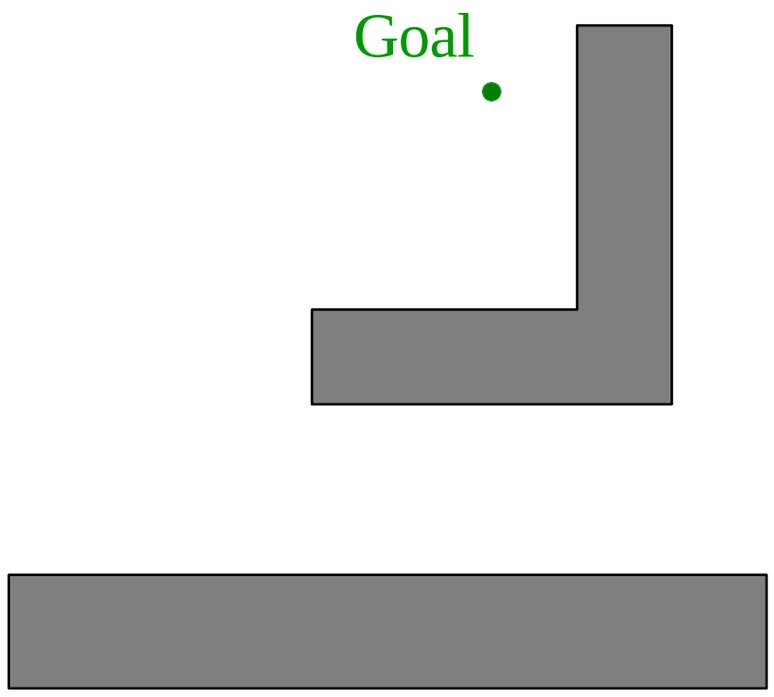
A Few General Concepts

- Workspace W
  - $\Re(2)$ or $\Re(3)$ depending on the robot
  - could be infinite (open) or bounded (closed/compact)
- Obstacle WO_i
- Free workspace $W_{free} = W \setminus \cup_i WO_i$

The ***Bug*** Algorithms

provable results...

Insect-inspired



- **known direction to goal**
  - **robot can measure distance d(x,y) between pts x and y**
- **otherwise local sensing**

  walls/obstacles & encoders
- ***reasonable* world**

  1)finitely many obstacles in any finite area

  2)a line will intersect an obstacle finitely many times

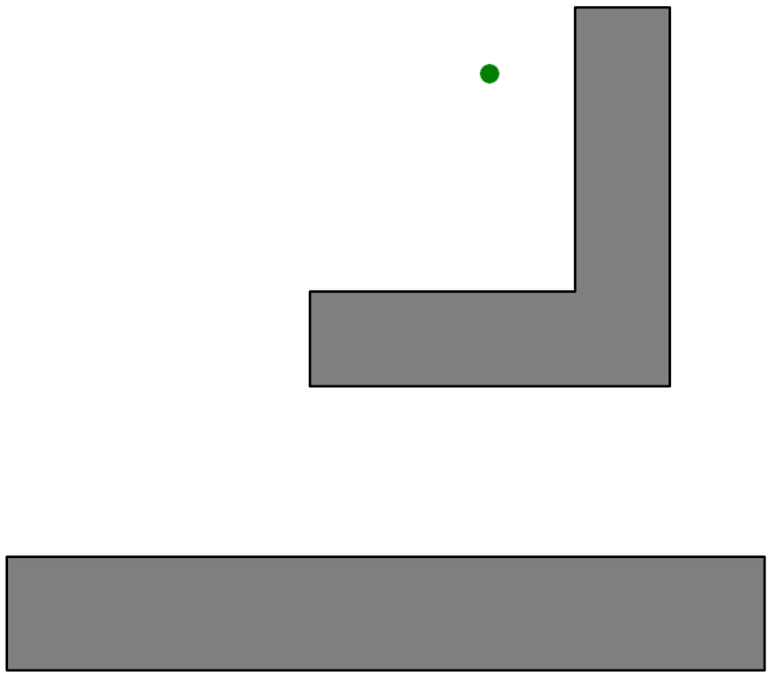
  3) Workspace is bounded

Buginner Strategy

"Bug 0" algorithm

- **known direction to goal**
- **otherwise local sensing**

  walls/obstacles & encoders



Some notation:

q_{start} and q_{goal}

"hit point" q^H_i

"leave point q^L_i

A *path* is a sequence of hit/leave pairs bounded by q_{start} and q_{goal}

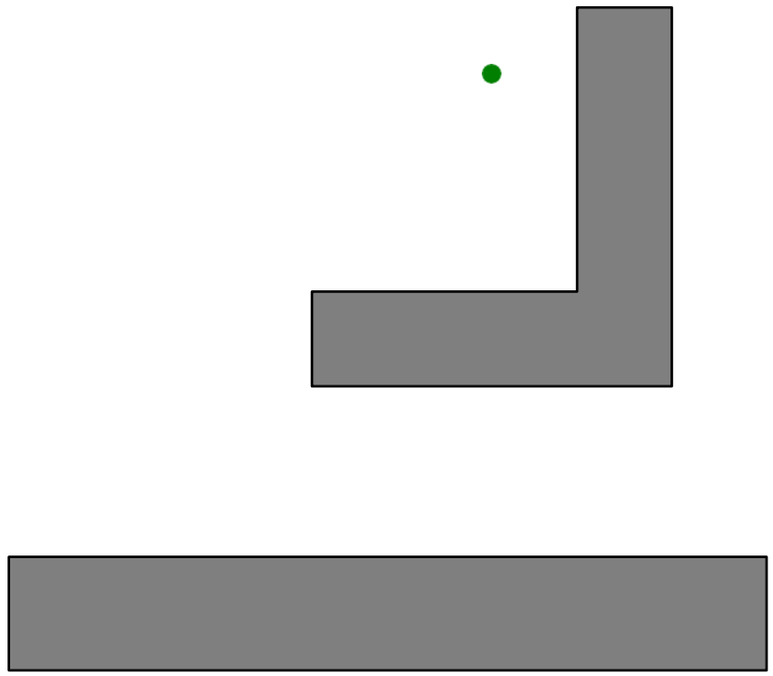
how ?

Buginner Strategy

"Bug 0" algorithm

- **known direction to goal**
- **otherwise local sensing**

  walls/obstacles & encoders



1) head toward goal

2)follow obstacles until you can head toward the goal again

3) continue

path ?

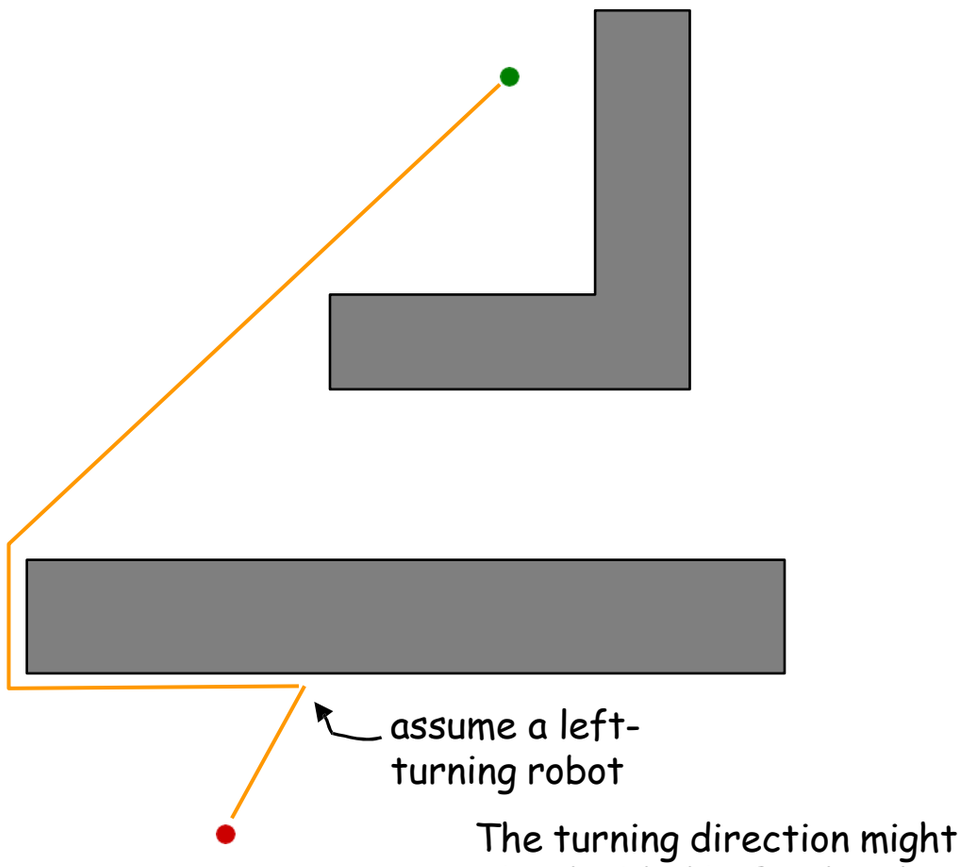
Buginner Strategy

"Bug 0" algorithm

1) head toward goal

2)follow obstacles until you can head toward the goal again

3) continue



OK ?

Bug Zapper

What map will foil Bug 0 ?

"Bug 0" algorithm

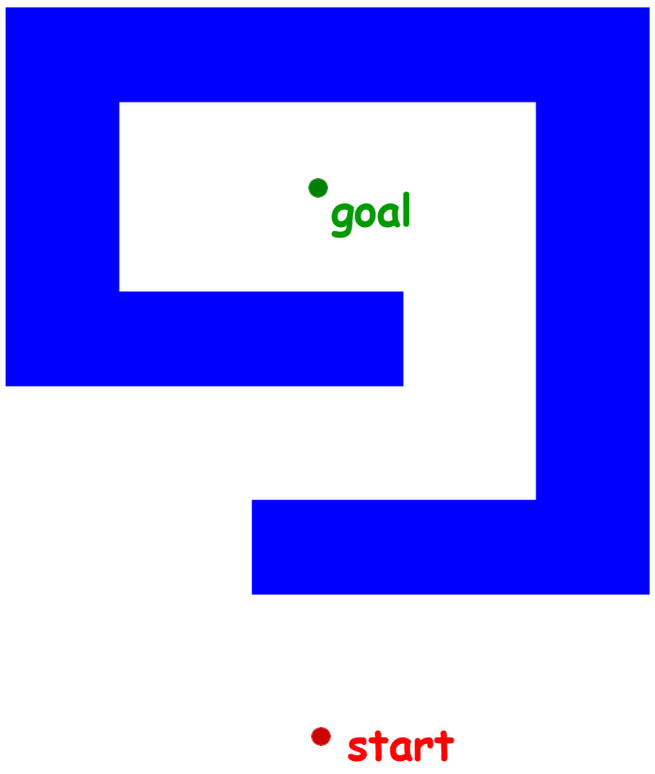
1) head toward goal

2)follow obstacles until you can head toward the goal again

3) continue

Bug Zapper

What map will foil Bug 0 ?



"Bug 0" algorithm

1) head toward goal

2)follow obstacles until you can head toward the goal again

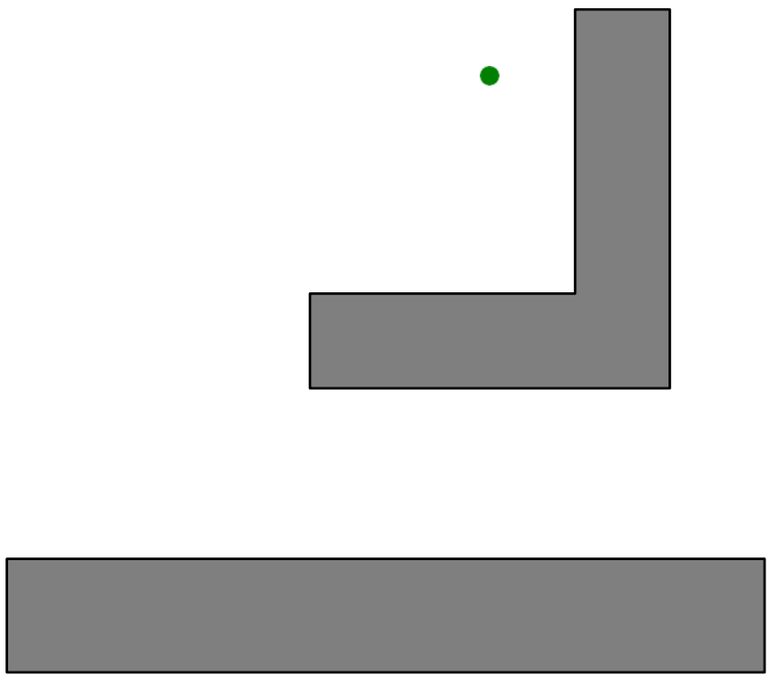
3) continue

A better bug?

But add <u>some</u> memory!

- **known direction to goal**
- **otherwise local sensing**

walls/obstacles & **encoders**



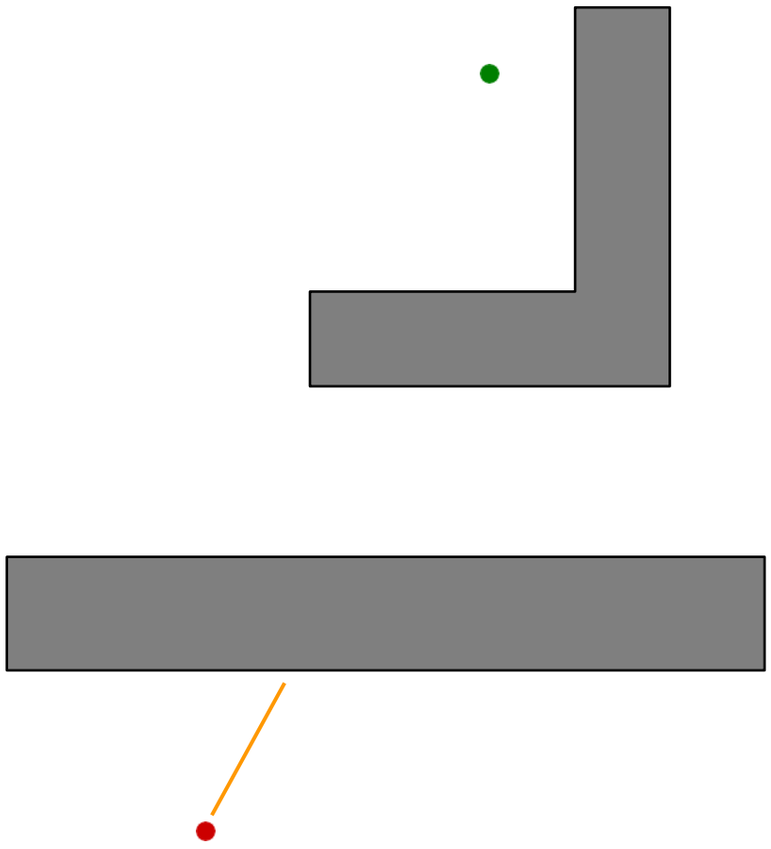
improvement ideas?

Bug 1

But <u>some</u> computing power!

- **known direction to goal**
- **otherwise local sensing**

walls/obstacles & **encoders**



"Bug 1" algorithm

1) head toward goal

2)if an obstacle is encountered, circumnavigate it *and* remember how close you get to the goal

3)return to that closest point (by wall-following) and continue

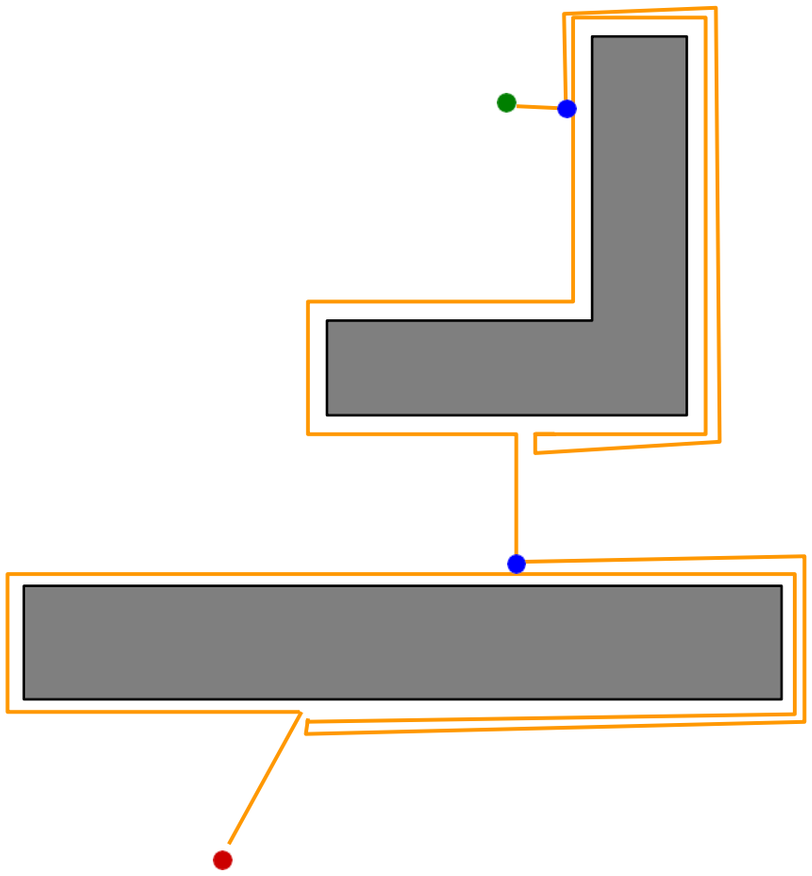
Vladimir Lumelsky & Alexander Stepanov: <u>Algorithmica</u> 1987

Bug 1

But some computing power!

- **known direction to goal**
- **otherwise local sensing**

walls/obstacles & **encoders**



"Bug 1" algorithm

1) head toward goal

2)if an obstacle is encountered, circumnavigate it *and* remember how close you get to the goal

3)return to that closest point (by wall-following) and continue

Vladimir Lumelsky & Alexander Stepanov: Algorithmica 1987

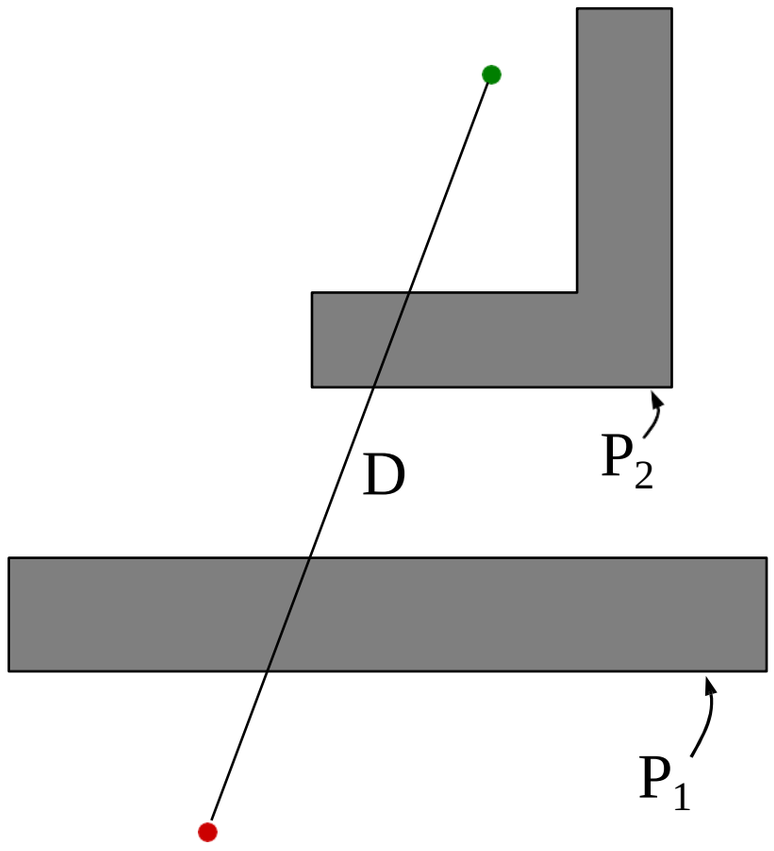
BUG 1 More formally

- Let $q^L_0 = q_{start}$; i = 1
- repeat
  - repeat
    - from q^L_{i-1} move toward q_{goal}
  - until goal is reached or obstacle encountered at q^H_i
  - if goal is reached, exit
  - repeat
    - follow boundary recording pt q^L_i with shortest distance to goal
  - until q_{goal} is reached or q^H_i is re-encountered
  - if goal is reached, exit
  - Go to q^L_i
  - if move toward q_{goal} moves into obstacle
    - exit with failure
  - else
    - i=i+1
    - continue

“Quiz” Bug 1 analysis

Bug 1: Path Bounds

What are upper/lower bounds on the path length that the robot takes?



D = straight-line distance from start to goal

P_i = perimeter of the *i* th obstacle

Lower bound:

What’s the shortest distance it might travel?

Upper bound:

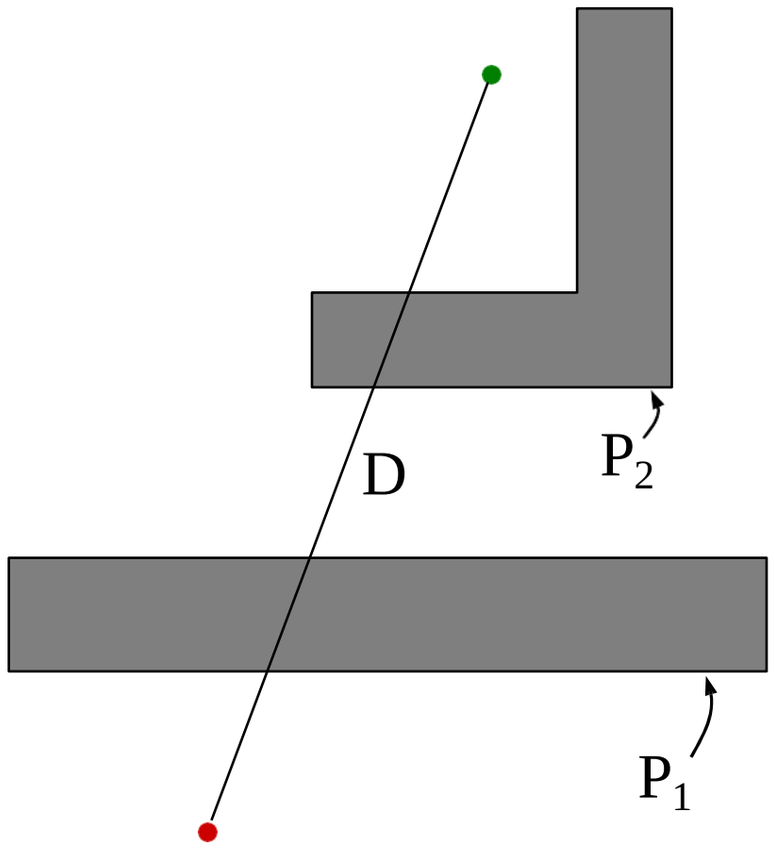
What’s the longest distance it might travel?

What is an environment where your upper bound is required?

“Quiz” Bug 1 analysis

Bug 1: Path Bounds

What are upper/lower bounds on the path length that the robot takes?



D = straight-line distance from start to goal

P_i = perimeter of the *i* th obstacle

Lower bound:

What’s the shortest distance it might travel?

$$D$$

Upper bound:

What’s the longest distance it might travel?

$$D + 1.5 \sum_i P_i$$

What is an environment where your upper bound is required?

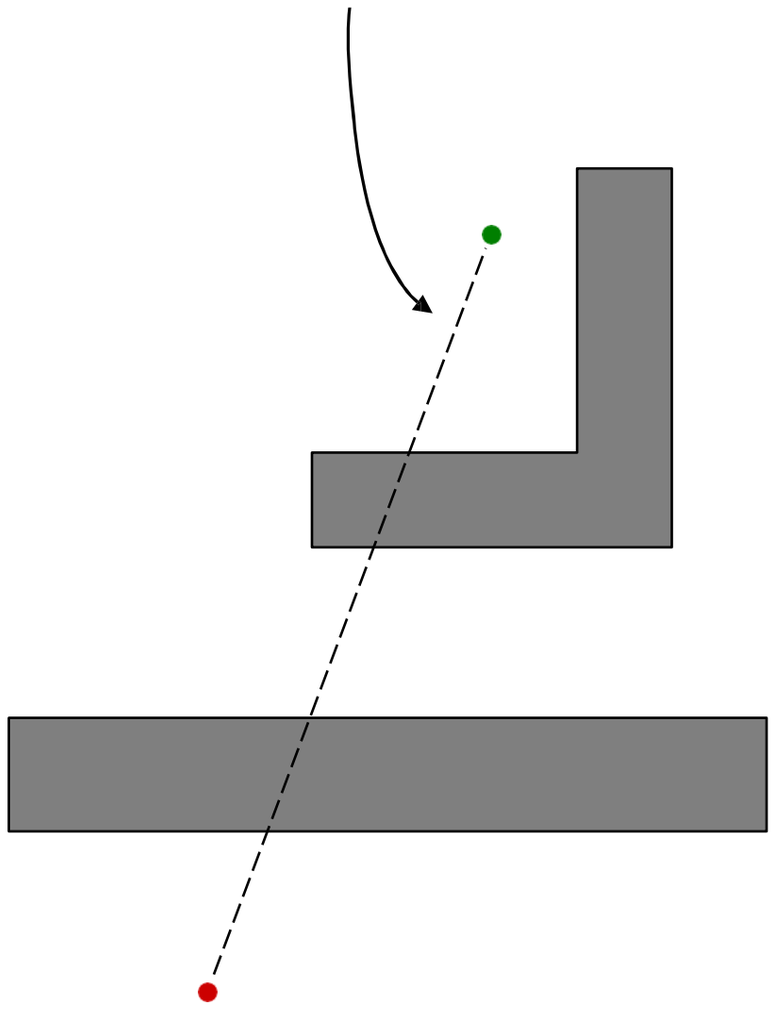
How Can We Show Completeness?

- An algorithm is *complete* if, in finite time, it finds a path if such a path exists or terminates with failure if it does not.

- Suppose BUG1 were incomplete
  - Therefore, there is a path from start to goal
    - By assumption, it is finite length, and intersects obstacles a finite number of times.
  - BUG1 does not find it
    - Either it terminates incorrectly, or, it spends an infinite amount of time
    - Suppose it never terminates
      - but each leave point is closer to the obstacle than corresponding hit point
      - Each hit point is closer than the last leave point
      - Thus, there are a finite number of hit/leave pairs; after exhausting them, the robot will proceed to the goal and terminate
    - Suppose it terminates (incorrectly)
    - Then, the closest point after a hit must be a leave where it would have to move into the obstacle
      - But, then line from robot to goal must intersect object even number of times (Jordan curve theorem)
      - But then there is another intersection point on the boundary closer to object. Since we assumed there is a path, we must have crossed this pt on boundary which contradicts the definition of a leave point.

Another step forward?

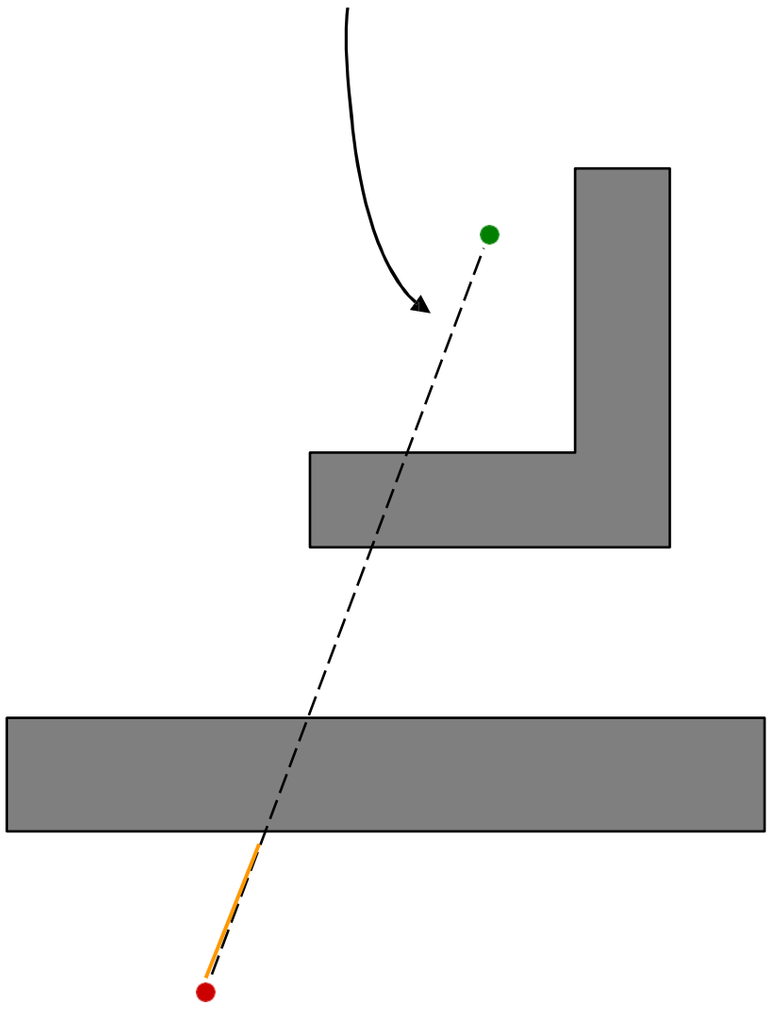
Call the line from the starting point to the goal the ***m-line***



"Bug 2" Algorithm

A better bug?

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

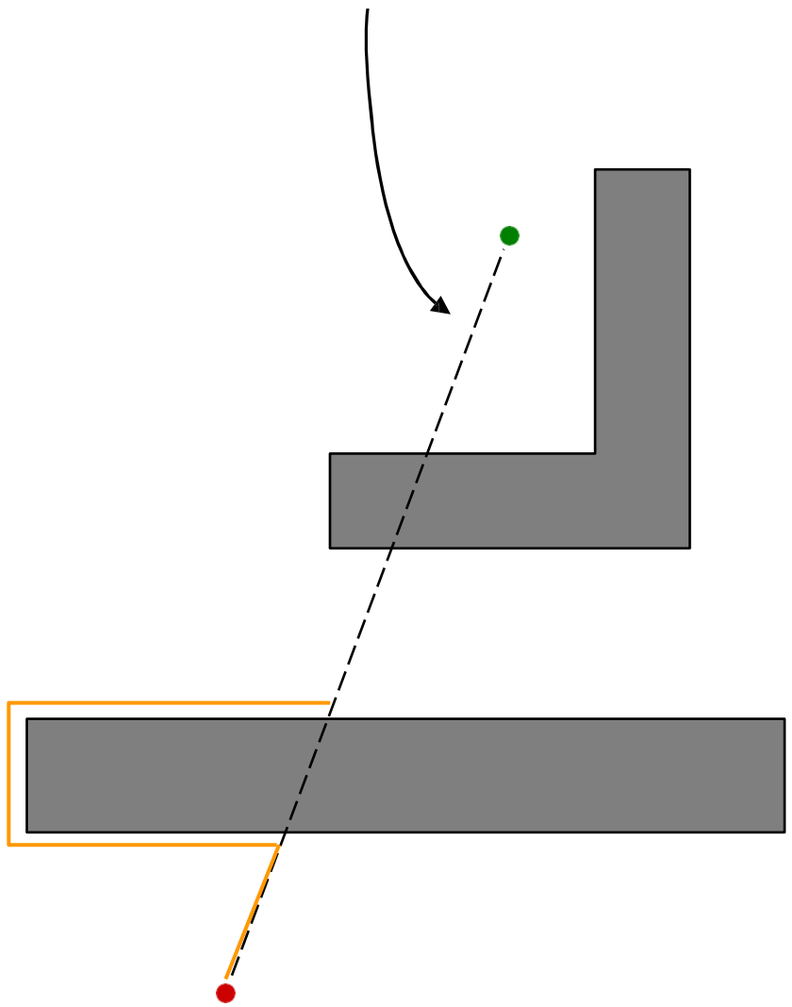


"Bug 2" Algorithm

1) head toward goal on the *m-line*

A better bug?

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

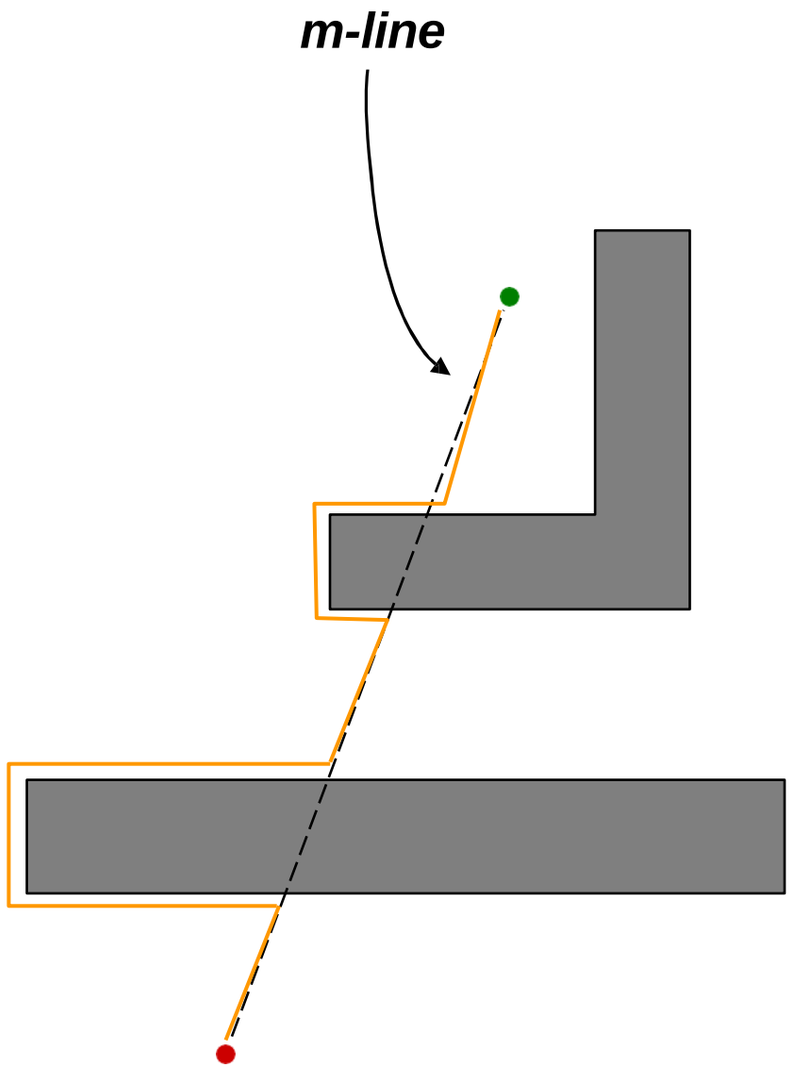


"Bug 2" Algorithm

1) head toward goal on the *m-line*

2)if an obstacle is in the way, follow it until you encounter the m-line again.

A better bug?



"Bug 2" Algorithm

1) head toward goal on the *m-line*

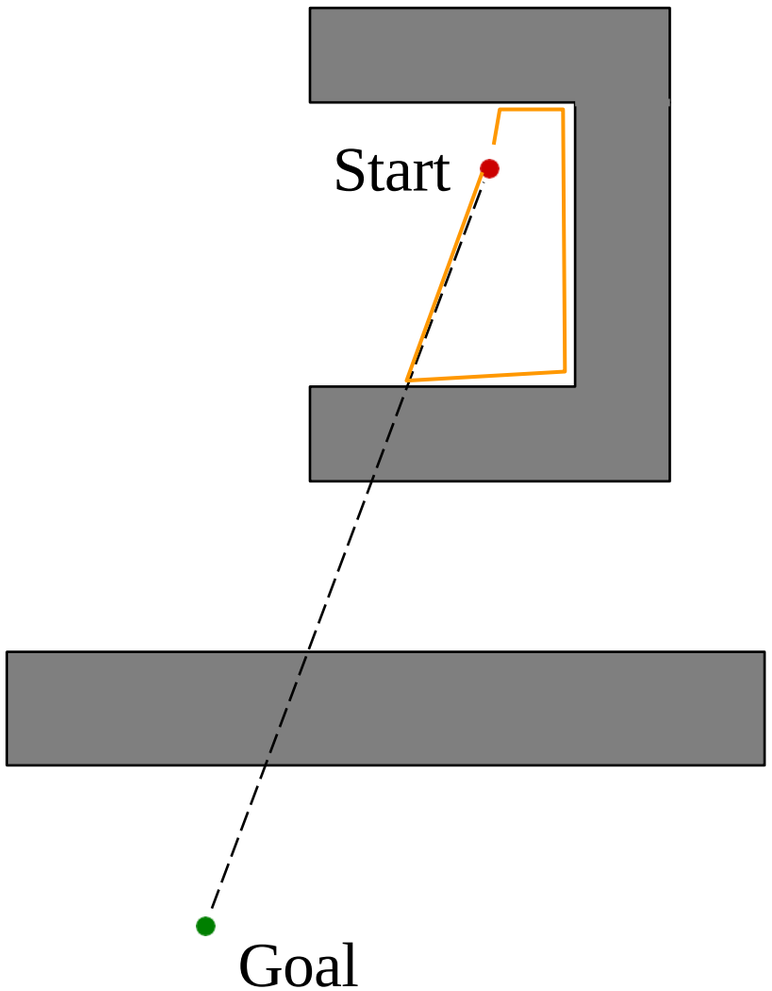
2)if an obstacle is in the way, follow it until you encounter the m-line again.

3)Leave the obstacle and continue toward the goal

OK ?

A better bug?

"Bug 2" Algorithm



1) head toward goal on the *m-line*

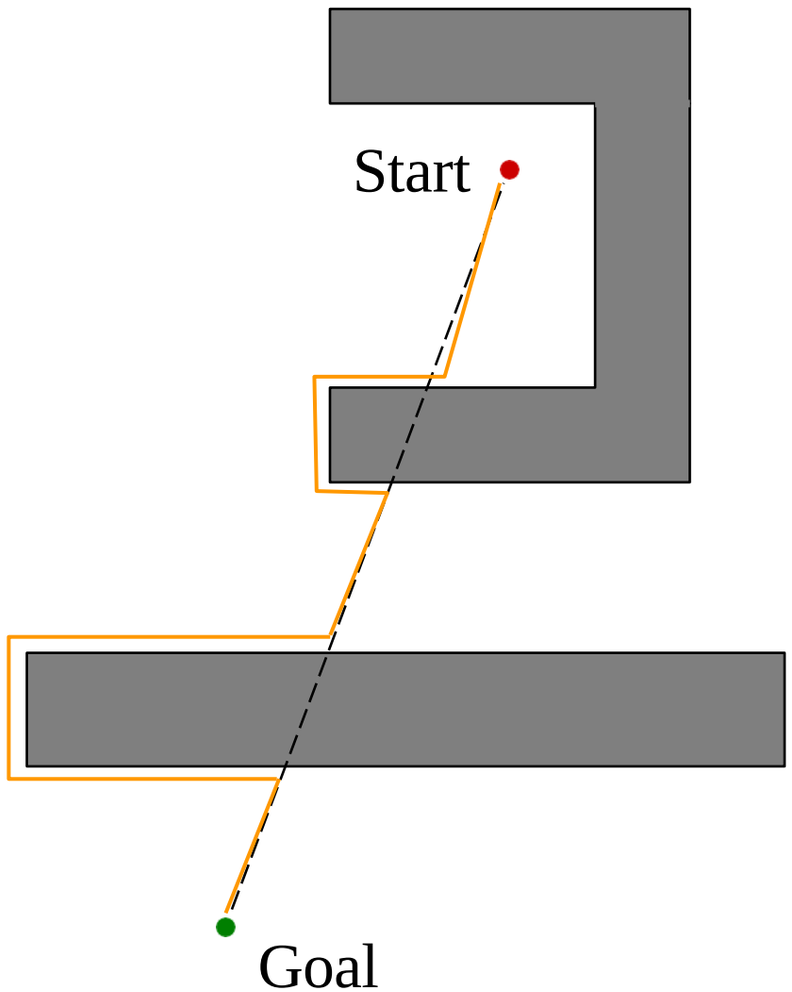
2)if an obstacle is in the way, follow it until you encounter the m-line again.

3)Leave the obstacle and continue toward the goal

NO! How do we fix this?

A better bug?

"Bug 2" Algorithm



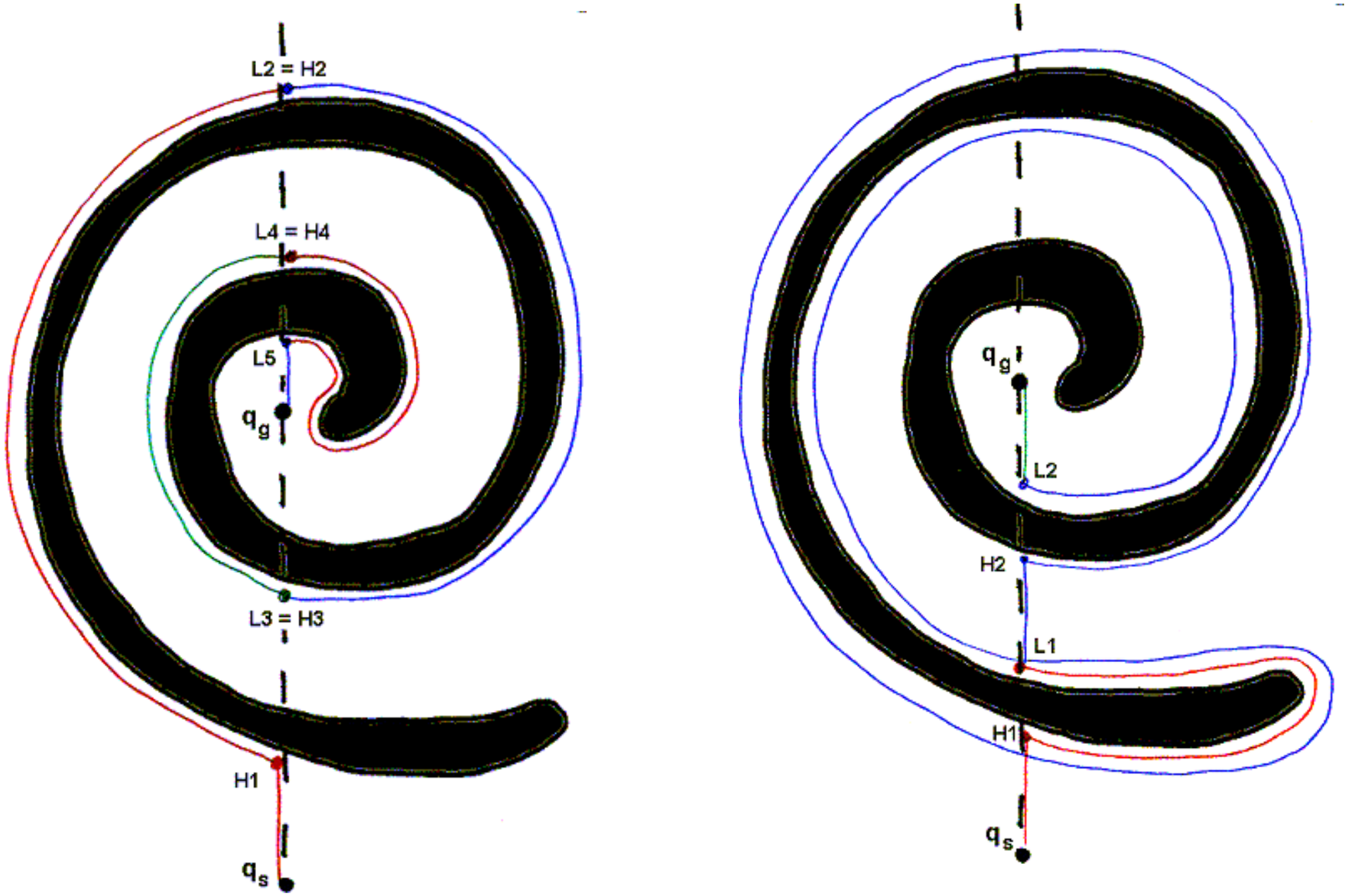
1) head toward goal on the *m-line*

2)if an obstacle is in the way, follow it until you encounter the m-line again ***closer to the goal***.

3)Leave the obstacle and continue toward the goal

Better or worse than Bug1?

The Spiral



16-735, Howie Choset with slides from G.D. Hager and Z. Dodds

BUG 2 More formally

- Let $q^L_0 = q_{start;}$ i = 1
- repeat
  - repeat
    - from q^L_{i-1} move toward q_{goal} along the m-line
  - until goal is reached or obstacle encountered at q^H_i
  - if goal is reached, exit
  - repeat
    - follow boundary
  - until q_{goal} is reached or q^H_i is re-encountered or m-line is re-encountered, x is not q^H_i, $d(x,q_{goal}) < d(q^H_i,q_{goal})$ and way to goal is unimpeded
  - if goal is reached, exit
  - if q^H_i is reached, return failure
  - else
    - $q^L_i = m$
    - i=i+1
    - continue

head-to-head comparison

or thorax-to-thorax, perhaps

Draw worlds in which Bug 2 does better than Bug 1 (and vice versa).

Bug 2 beats Bug 1

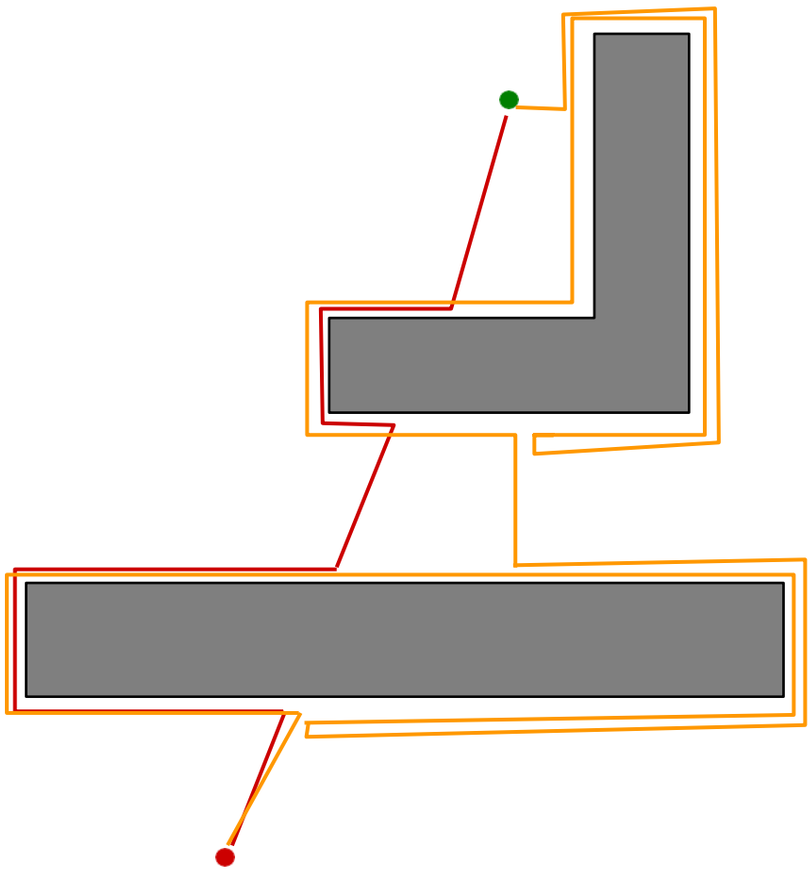
Bug 1 beats Bug 2

head-to-head comparison

or thorax-to-thorax, perhaps

Draw worlds in which Bug 2 does better than Bug 1 (and vice versa).

Bug 2 beats Bug 1



Bug 1 beats Bug 2

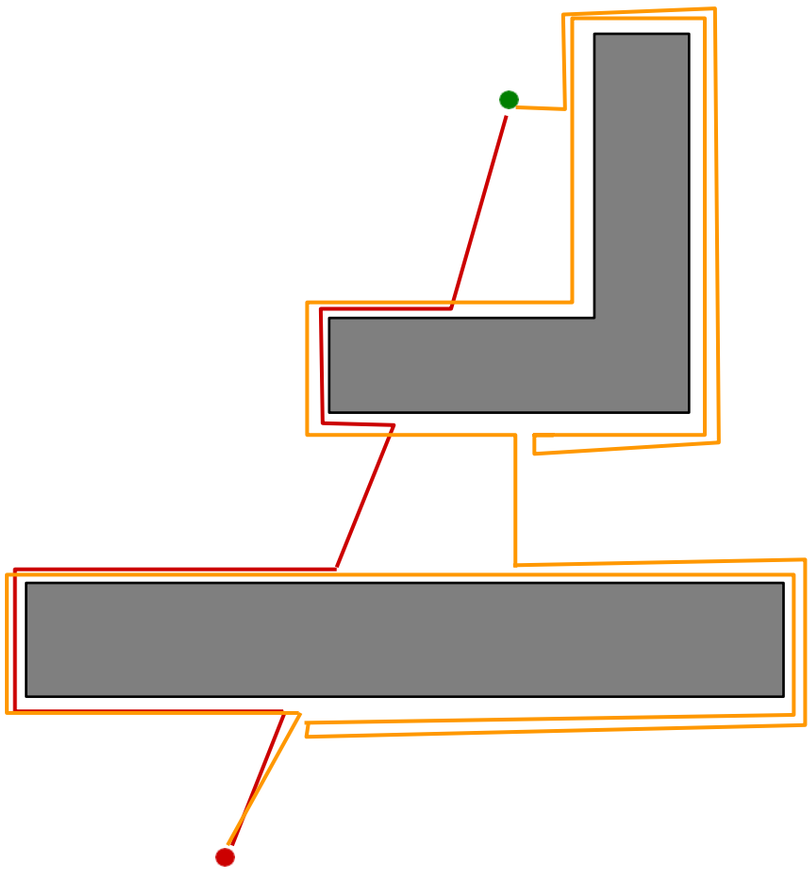
?

head-to-head comparison

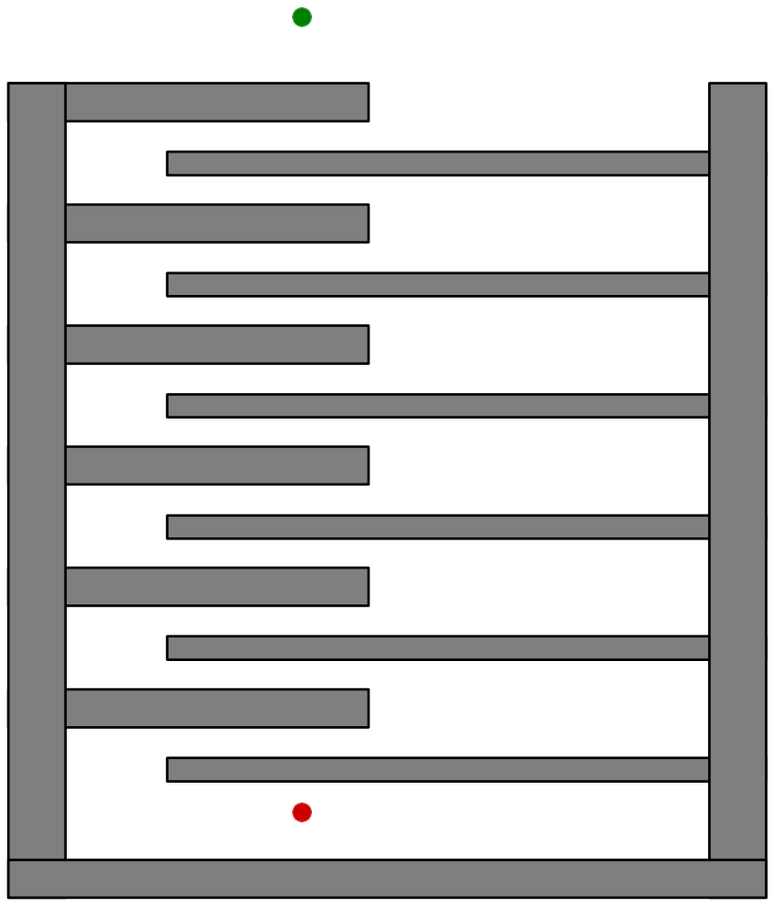
or thorax-to-thorax, perhaps

Draw worlds in which Bug 2 does better than Bug 1 (and vice versa).

Bug 2 beats Bug 1



Bug 1 beats Bug 2



BUG 1 vs. BUG 2

- BUG 1 is an *exhaustive search algorithm*
  - it looks at all choices before commiting

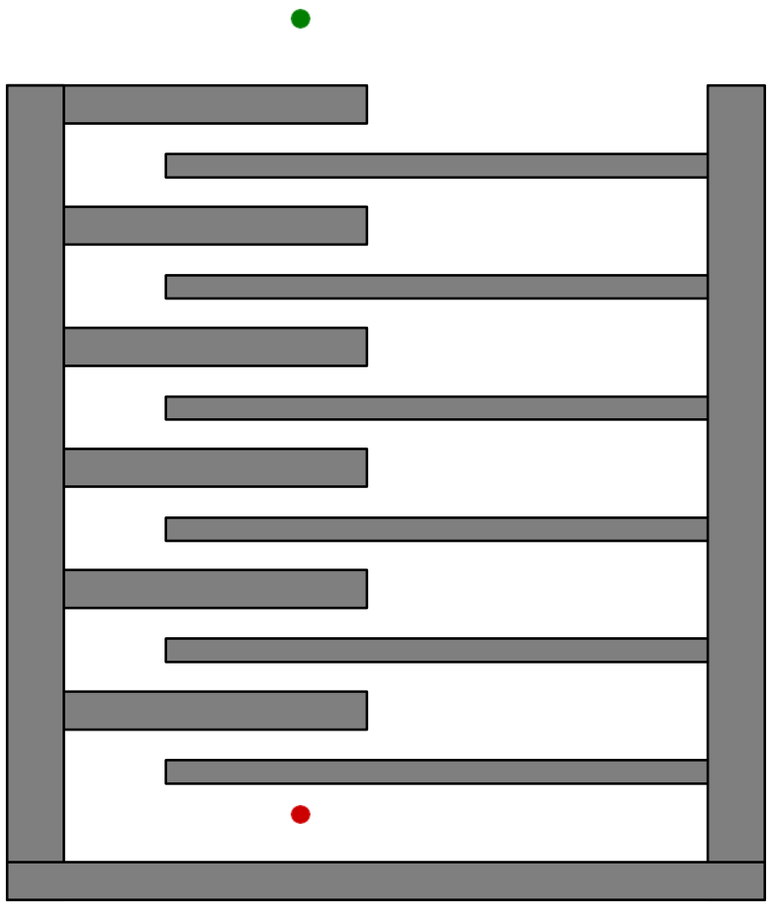
- BUG 2 is a *greedy* algorithm
  - it takes the first thing that looks better

- In many cases, BUG 2 will outperform BUG 1, but

- BUG 1 has a more predictable performance overall

“Quiz” Bug 2 analysis

Bug 2: Path Bounds



What are upper/lower bounds on the path length that the robot takes?

D = straight-line distance from start to goal

P_i = perimeter of the *i* th obstacle

Lower bound:

What’s the shortest distance it might travel?

D

Upper bound:

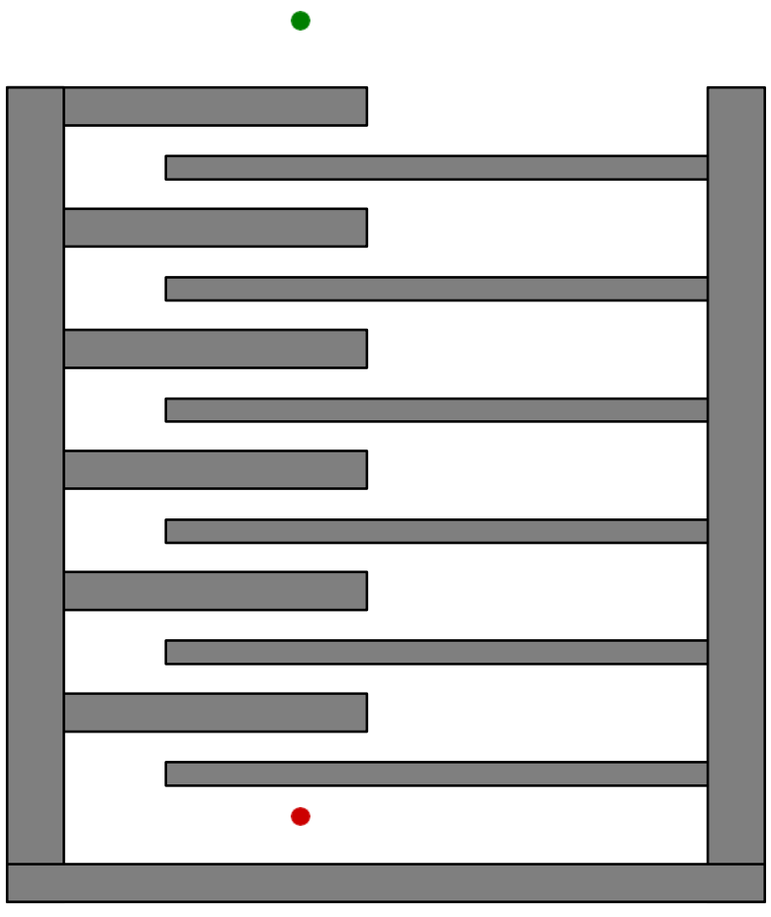
What’s the longest distance it might travel?

What is an environment where your upper bound is required?

“Quiz”

Bug 2 analysis

Bug 2: Path Bounds



What are upper/lower bounds on the path length that the robot takes?

D = straight-line distance from start to goal

P_i = perimeter of the *i* th obstacle

Lower bound:

What’s the shortest distance it might travel?

$$D$$

Upper bound:

What’s the longest distance it might travel?

$$D + \sum_i \frac{n_i}{2} P_i$$

n_i = # of s-line intersections of the *i* th obstacle

What is an environment where your upper bound is required?

A More Realistic Bug

- As presented: global beacons plus contact-based wall following
- The reality: we typically use some sort of range sensing device that lets us look ahead (but has finite resolution and is noisy).
- Let us assume we have a range sensor

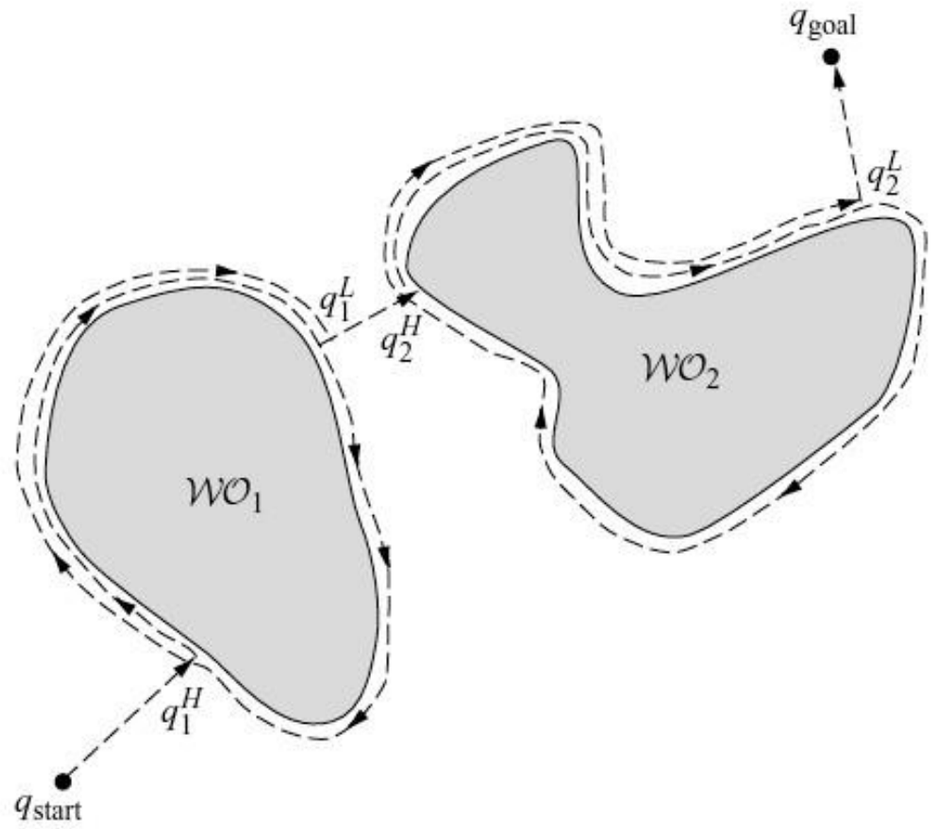
Bug 1: ~Exhaustive Search

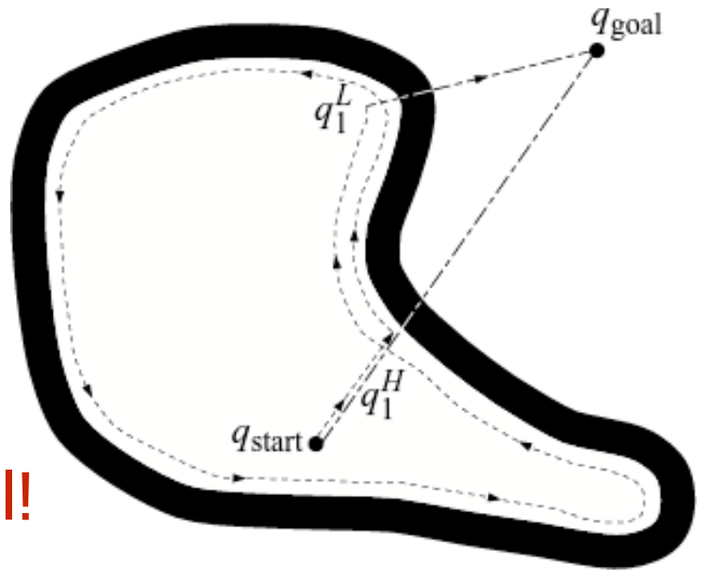
Algorithm 1 Bug1 Algorithm

Input: A point robot with a tactile sensor

Output: A path to the q_{goal} or a conclusion no such path exists

1: **while** Forever **do**
2: **repeat**
3: From q_{i-1}^L, move toward q_{goal}.
4: **until** q_{goal} is reached **or** an obstacle is encountered at q_i^H.
5: **if** Goal is reached **then**
6: Exit.
7: **end if**
8: **repeat**
9: Follow the obstacle boundary.
10: **until** q_{goal} is reached **or** q_i^H is re-encountered.
11: Determine the point q_i^L on the perimeter that has the shortest distance to the goal.
12: Go to q_i^L.
13: **if** the robot cannot move toward the goal **then**
14: Conclude q_{goal} is not reachable and exit.
15: **end if**
16: **end while**





Cannot move to goal!

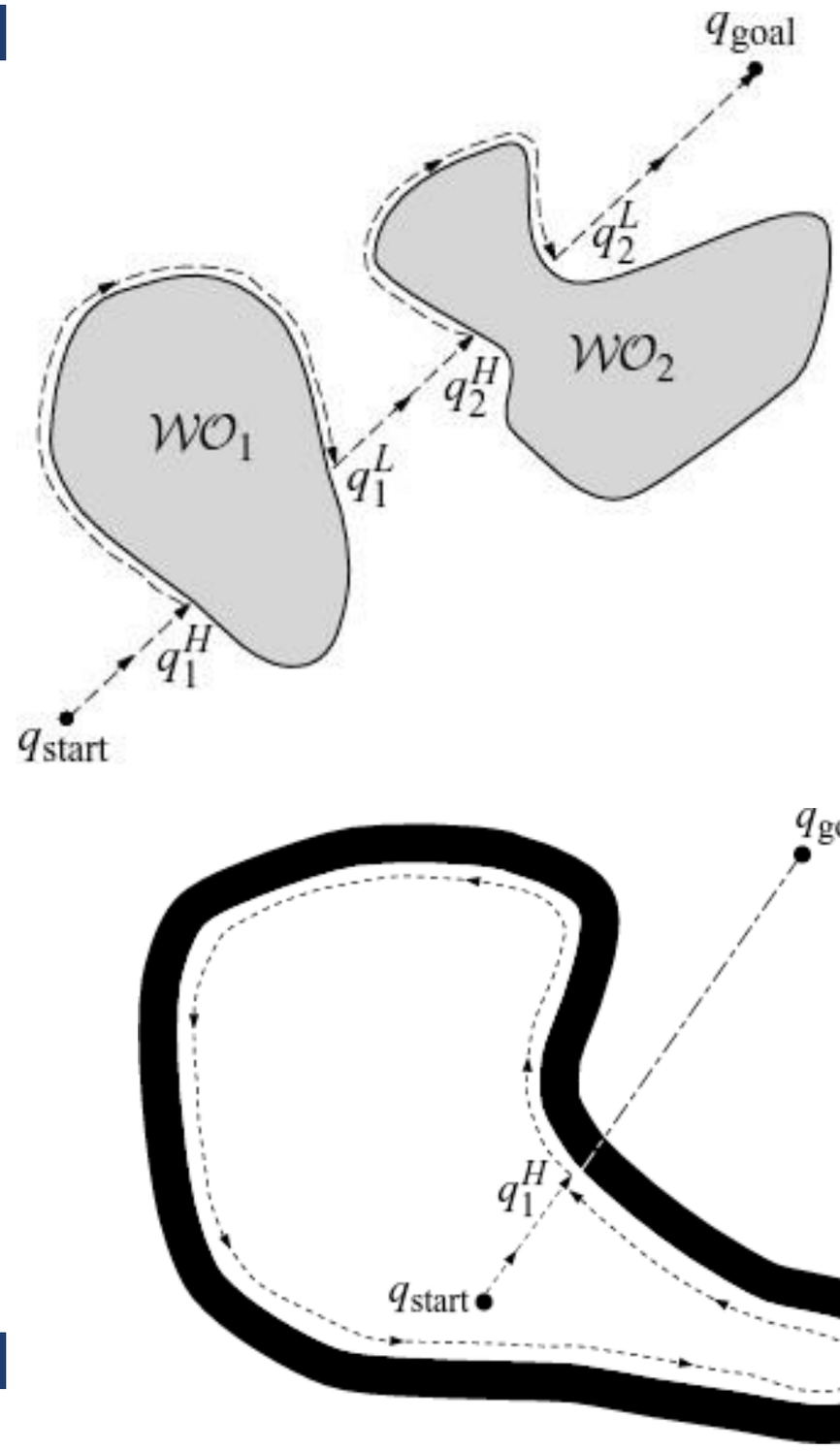
Bug 2: Greedy search

Algorithm 2 Bug2 Algorithm

Input: A point robot with a tactile sensor

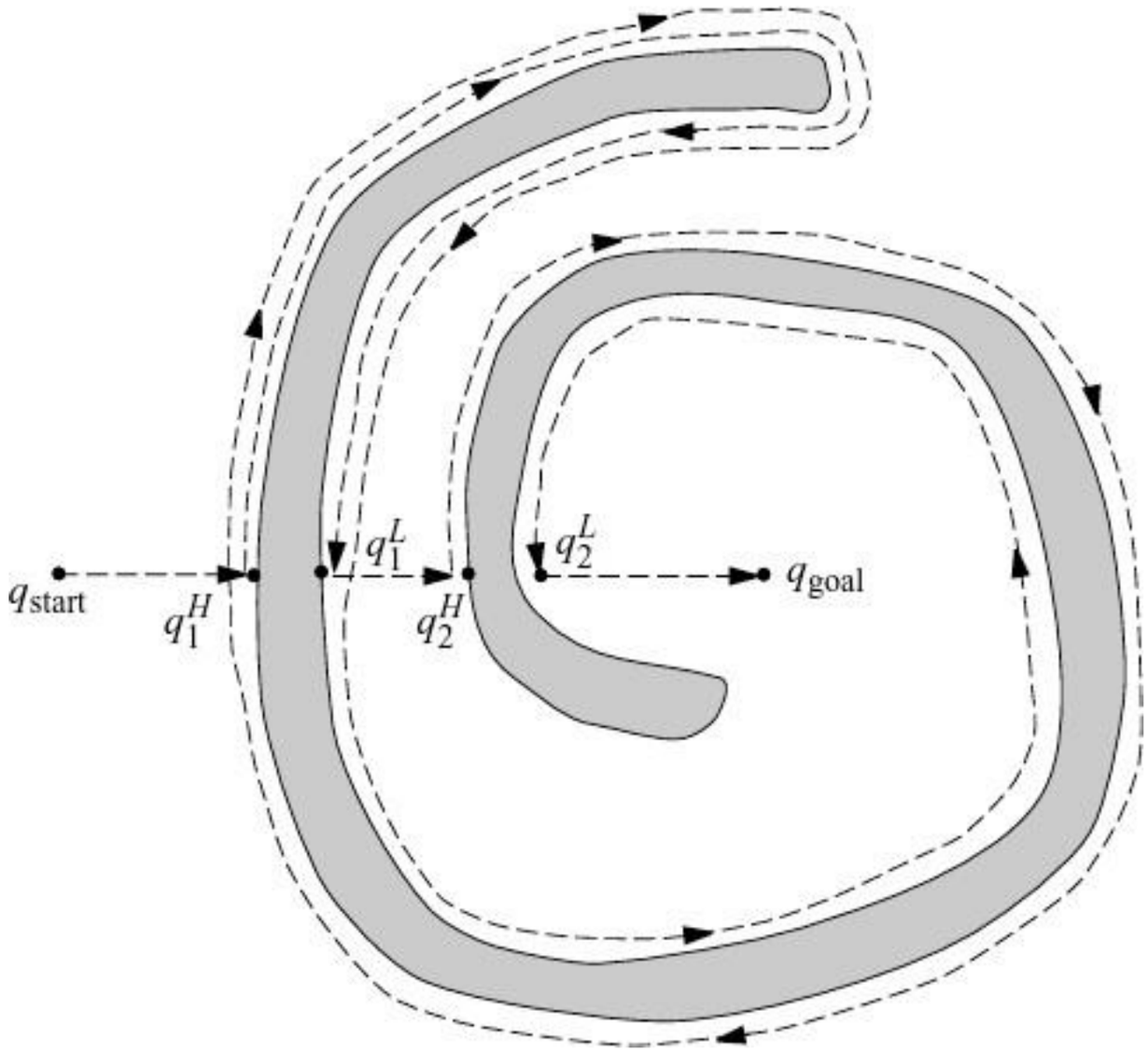
Output: A path to q_{goal} or a conclusion no such path exists

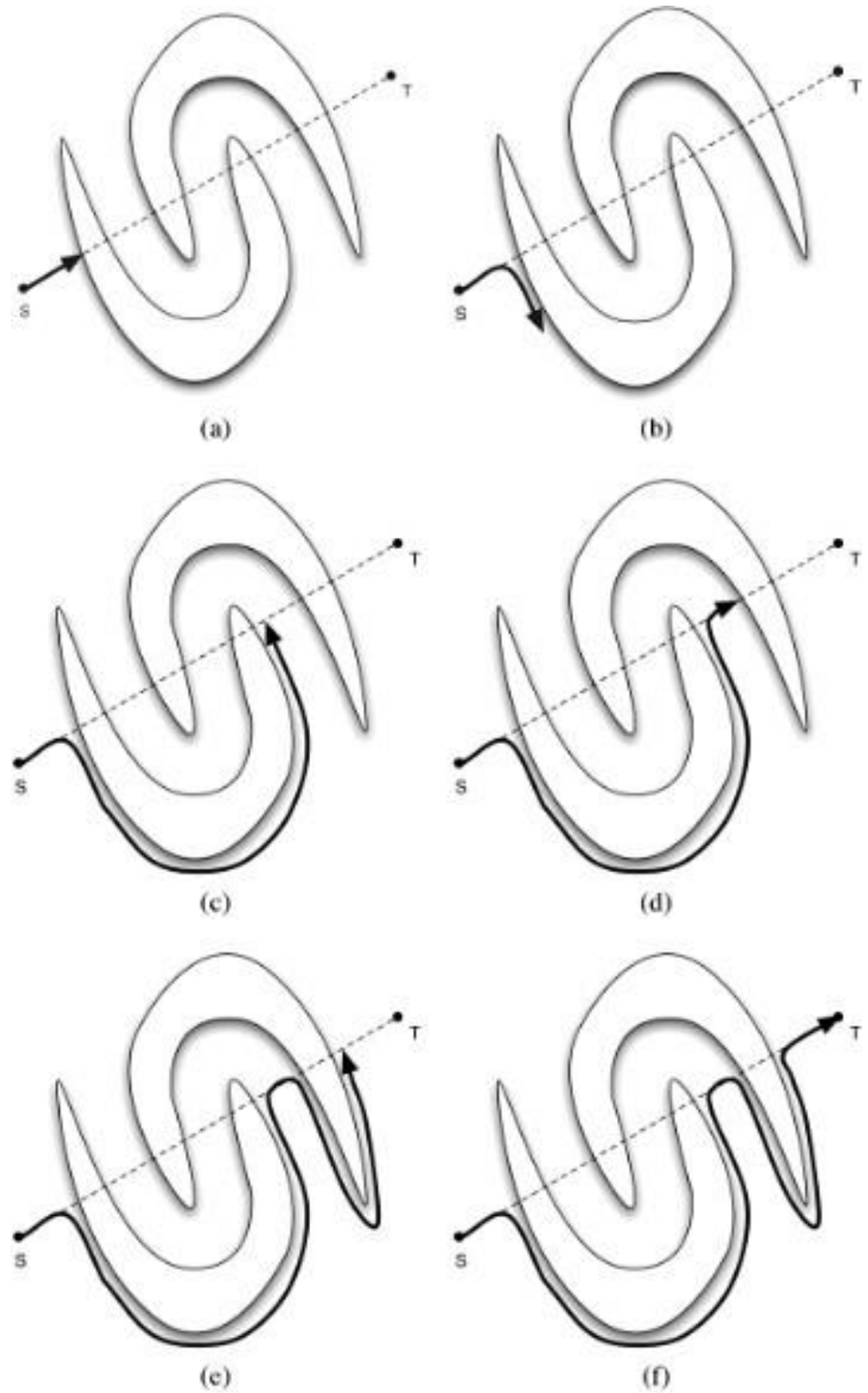
1: **while** True **do**
2: **repeat**
3: From q_{i-1}^L, move toward q_{goal} along m-line.
4: **until**
q_{goal} is reached **or**
an obstacle is encountered at *hit point* q_i^H.
5: Turn left (or right).
6: **repeat**
7: Follow boundary
8: **until**
9: q_{goal} is reached **or**
10: q_i^H is re-encountered **or**
11: m-line is re-encountered at a point m such that
12: $m \neq q_i^H$ (robot did not reach the hit point),
13: $d(m, q_{\text{goal}}) < d(m, q_i^H)$ (robot is closer), and
14: If robot moves toward goal, it would not hit the obstacle
15: Let $q_{i+1}^L = m$
16: Increment i
17: **end while**



Bug 2: going to kitchen!

Bug 2: issues

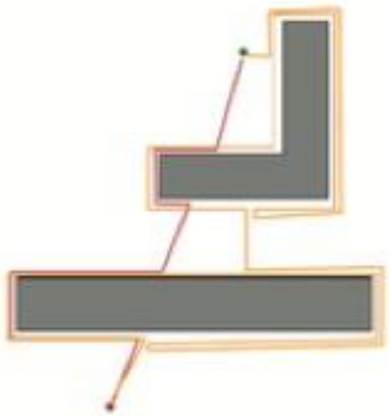




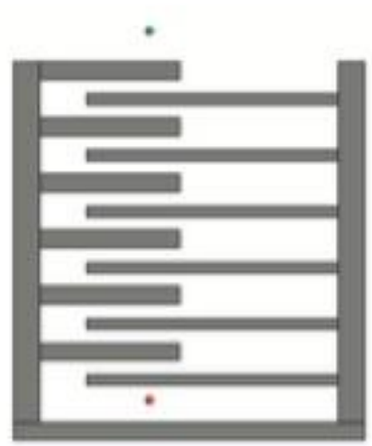
Bug 1 or Bug 2 ?

- At first sight it seems that Bug2 is better than Bug1 (does not involve a complete circumnavigation of obstacles)
- This is not always true, depends on the type of obstacles

Bug2 better than Bug1



Bug1 better than Bug2



Bug 1

- Exhaustive *leave-point* search
- Better on complex obstacles

Bug 2

- Opportunistic *leave-point* search
- Better on simple obstacles

(strong) Assumptions

- **Localization**: The robot must know where it is in the world, and must be able to maintain an (robust) estimate of this while it moves

- **Mapping**: Location of the goal is known within the robot's representation of the world.

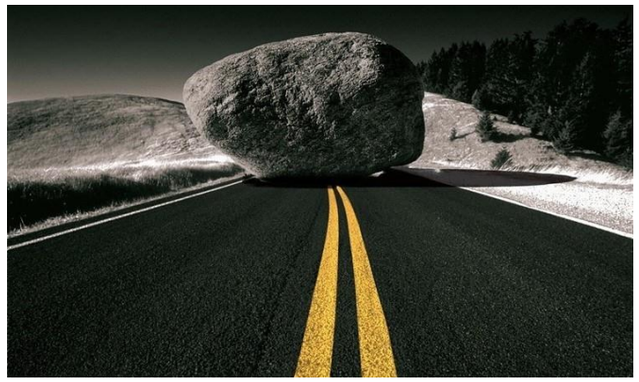
- **Mapping/Localization**: The robot must be able to mark a location in its representation of the world and be able to determine that it has returned to it.

- **Sensing:** Obstacles need to be sensed with precision, in order to circumnavigate them.
  - Obstacle detection and avoidance

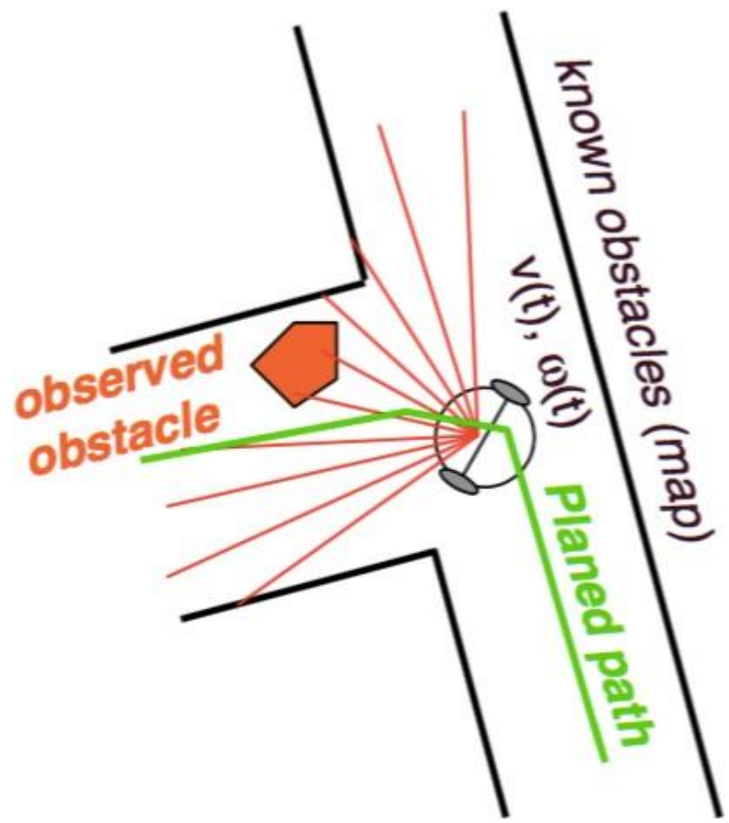
- **Actuation/Control:** The robot needs to be able to smoothly change and adapt its trajectory … velocities?

Obstacle avoidance

The goal of an obstacle avoidance algorithm is to **avoid collisions** with obstacles



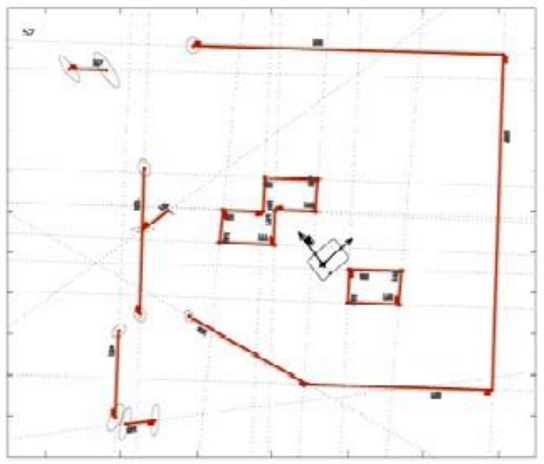
- It is usually based on a **local map**
- Implemented as an independent task / ROS node
- Efficient OA should (try to) be optimal wrt:
  - ✓ the overall goal
  - ✓ the actual speed and kinematics of the robot
  - ✓ the on board sensors
  - ✓ the actual and future risk of collision



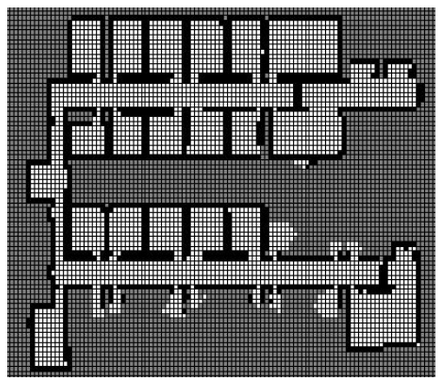
- Given map *and/or*
- Map built using sensor (range sensors)

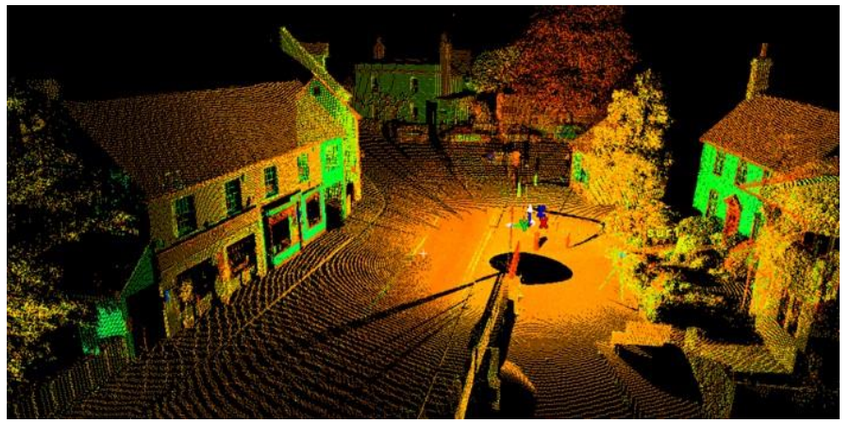
How to build a (local) map (for OA)?

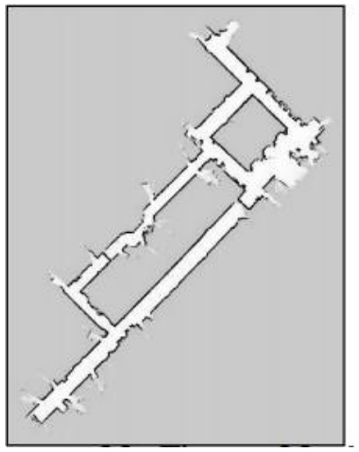
- Metric and/or topological representations of the environment



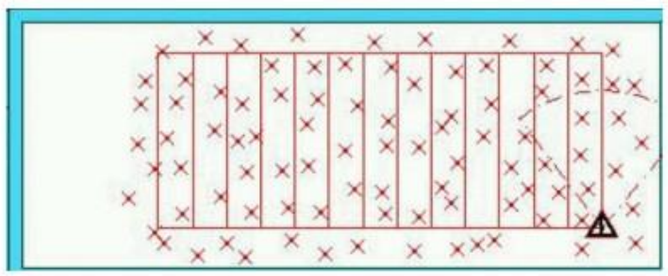
- Grid-based, 2D-3D scan

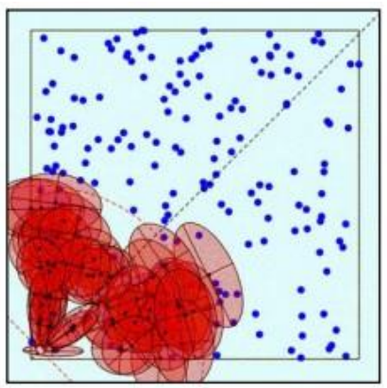




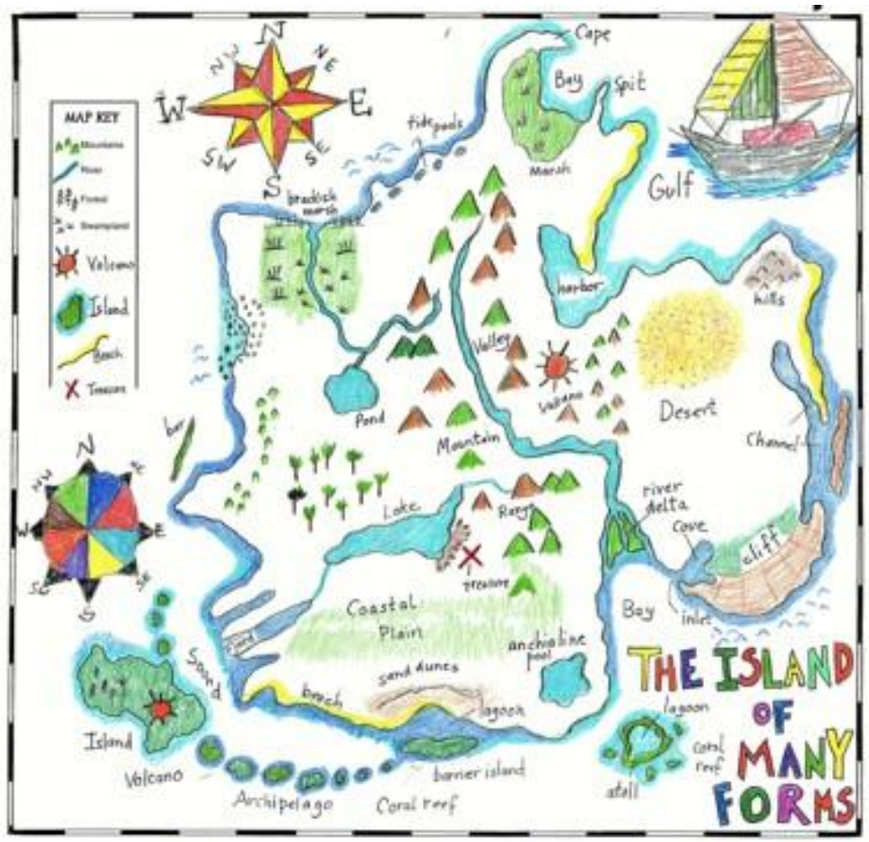


- Landmark-based

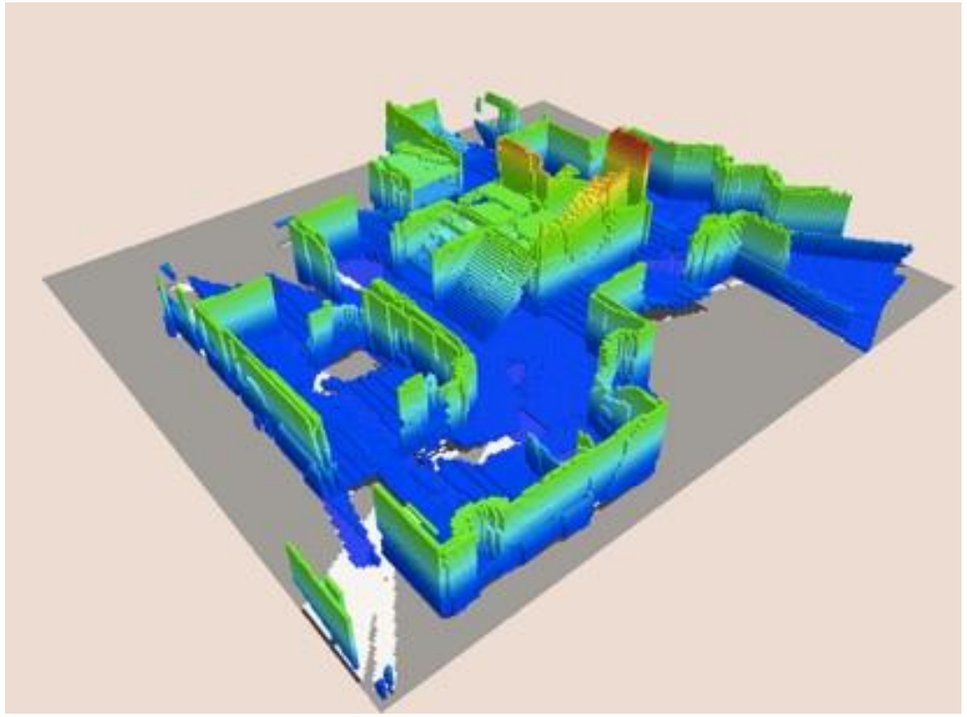




MAP: given or constructed?



Hand-made / Made using external tools



Made by the robot using exteroceptive sensors

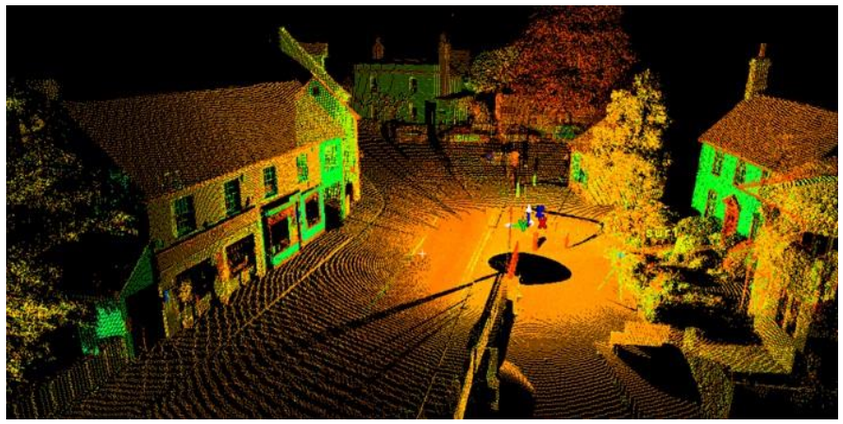
A map classification

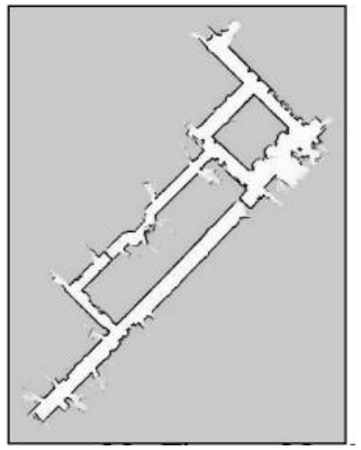
- A map M of the environment is a list of objects in the environment and along with their properties:

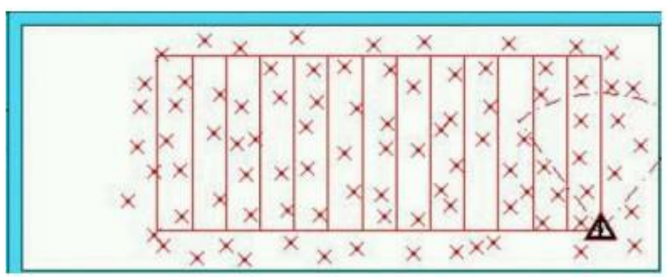
$$M = \{m_1, m_2, \ldots, m_N\}$$

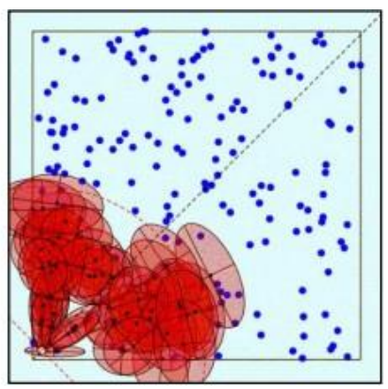
where N is total number of objects used to represent the environment, and each $m_i, i = , 1 \ldots , N,$ specifies a *property* that characterizes the i-th map object.

- Maps are usually *indexed* in one of two ways:







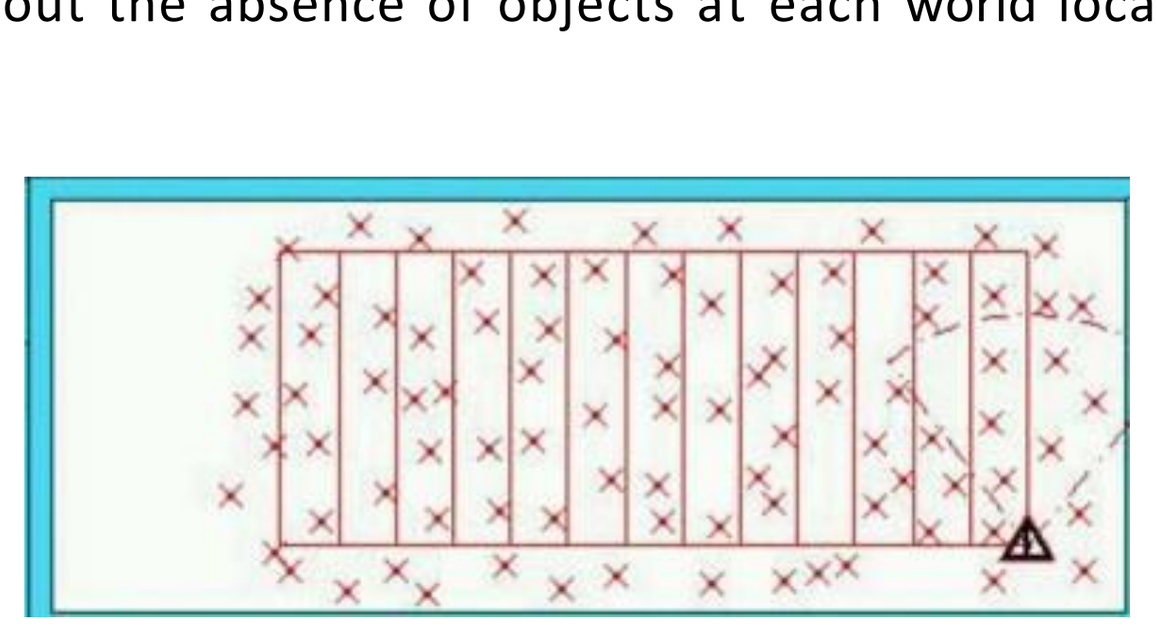


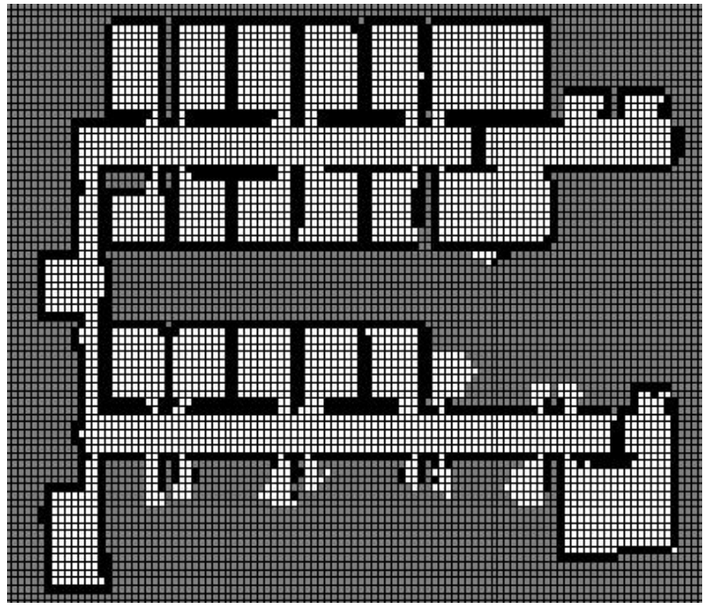
Map indexing

$$M = \{m_1, m_2, \ldots, m_N\}$$

- Location-based, where the index n corresponds to a specific location, such that it is common in planar maps to write $m_{x,y}$ to make explicit that m is the property of a specific world coordinate (x, y)

  Location-based maps are *volumetric*, in that they offer a label for any location in the world: they contain information not only about objects in the environment, but also about the absence of objects at each world location





- Feature-based, where i is a generic feature index, such that the value of m_i contains, next to the properties of a feature (e.g., the color), the Cartesian location of the feature.

  Feature-based maps only specify the characteristics of the environment at the specific locations that are indexed in the map.

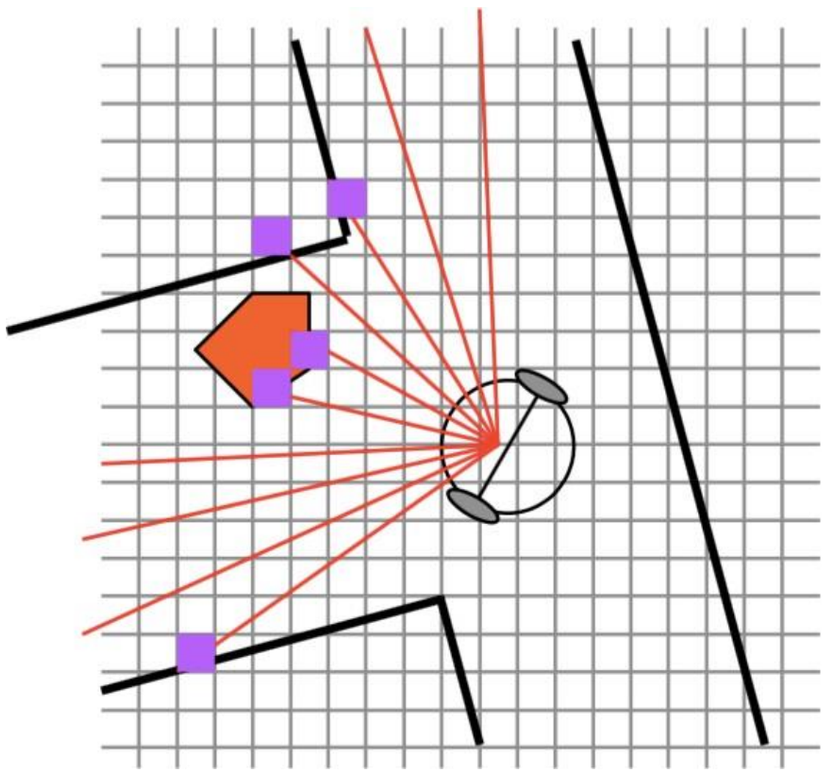
Occupancy grids

- The more classical (and used, in robotics) *location-based* map representation is known as occupancy grid map: the (continuous) space is partitioned into finitely many grid cells m_i, whose union covers the considered space:

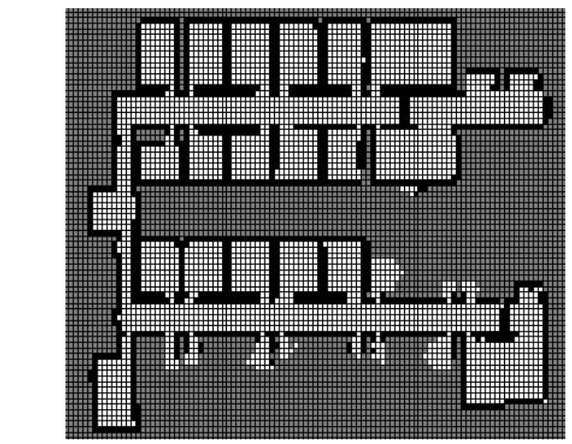
$$M = \sum_{i=1}^{N} m_i$$

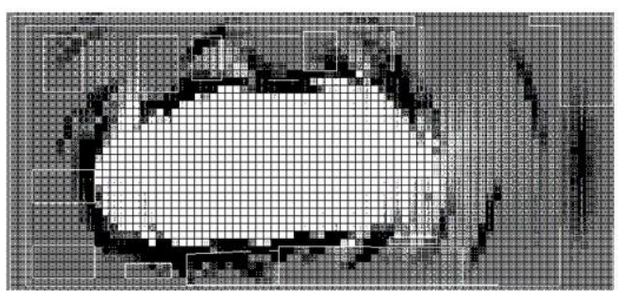
- Each m_i corresponds to a grid cell and are equivalent to the probability that there is an obstacle in that cell.
- After thresholding, each m_i has attached a binary of occupancy value which specifies whether the cell is occupied or free.

Sensor



Local





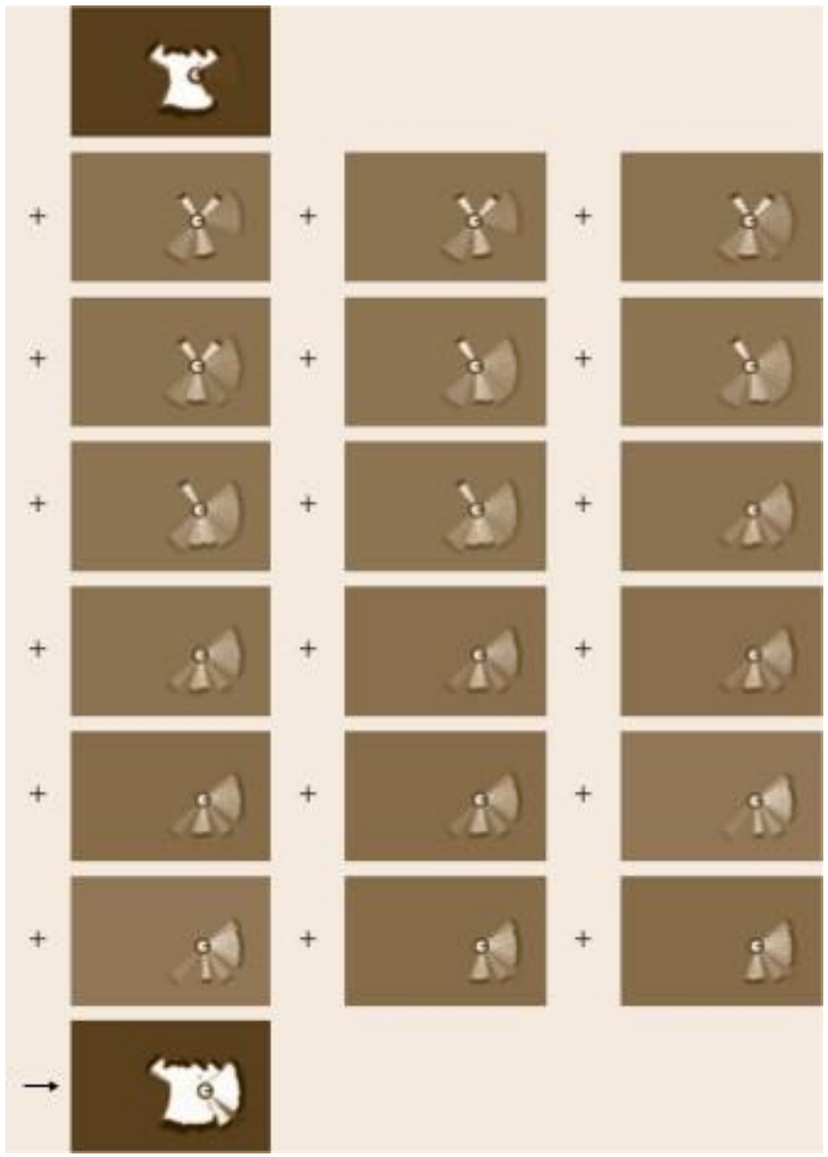
Global

Occupancy grids: probabilistic updates

The gold standard of any *occupancy grid mapping algorithm* is to calculate, at any time k, the posterior over the map given the observation data:

$$p(M \mid z_{1:k}, \boldsymbol{\xi}_{1:k})$$

which means estimating the joint probability that each cell is occupied or free.



How to calculate the **joint probability** *p (over multiple observations)* that a cell is occupied?

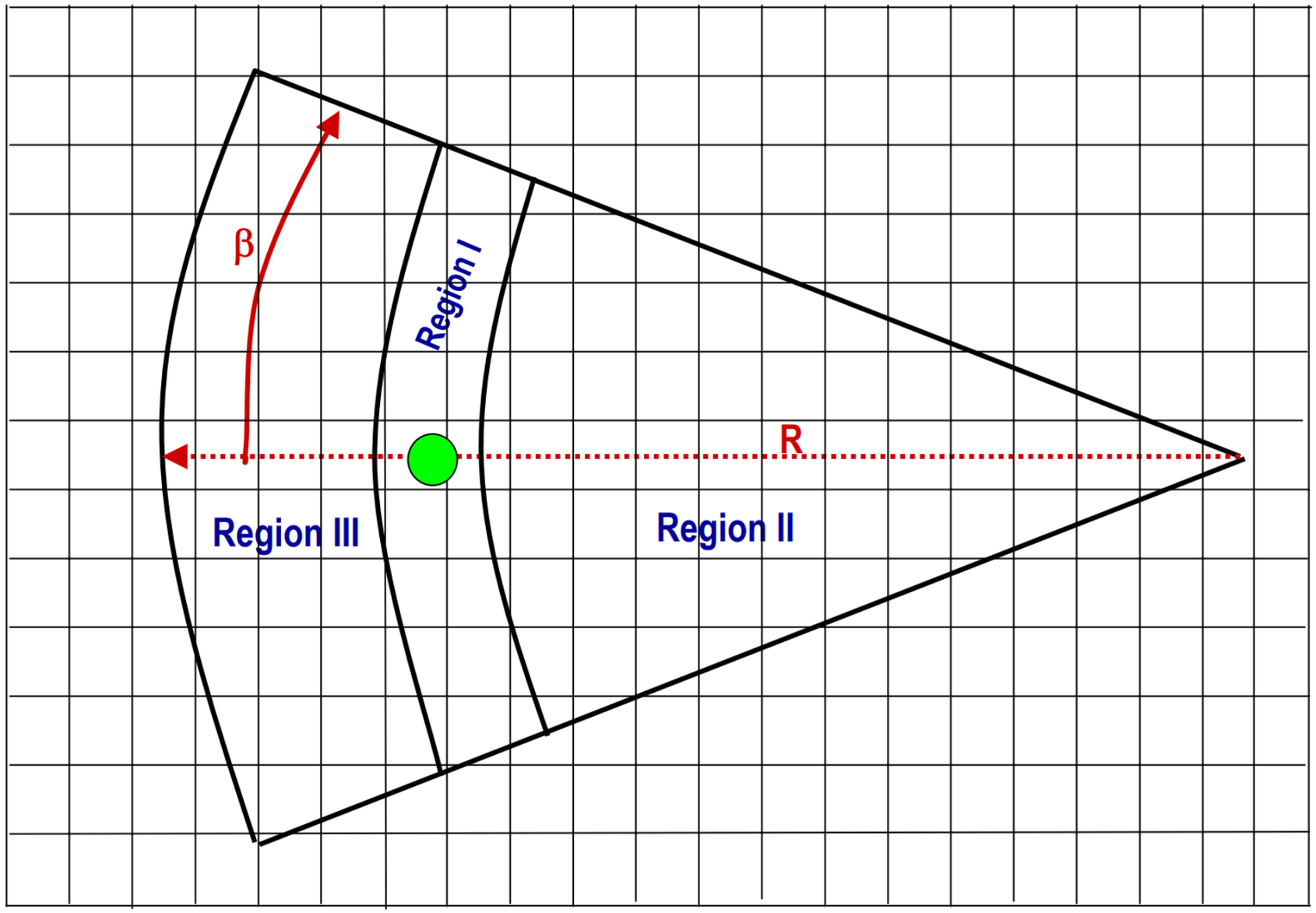
- Need **sensor model** to deal with uncertainty
- Let's look at the approach for *sonars* ...

Sensor models

- Need sensor model to deal with uncertainty

- Methods for generating sensor models:
  - Empirical (i.e., through testing)
  - Analytical (i.e, through an understanding of physical properties)
  - Subjective (i.e., through experience / learning)

Sonar sensor model



Region I: Probably occupied Region II: Probably empty Region III: Unknown

- If a sensor reading returns an obstacle at a distance *d*, what does it mean?
- We need to use the sensor model to define a probability value for each cell being either ***occupied*** or ***empty***

How to convert to numerical values?

- Need to translate model (previous slide) to specific numerical values for each occupancy grid cell

- Three methods

    1. Bayesian updates
    2. Dempster-Shafer approach
    3. HIMM (Histogrammic in Motion Mapping)

We will cover

- Bayesian
- HIMM (Maybe, depending on time)

Bayesian approach (most popular evidential method)

- Goal: Convert sensor readings into probabilities

  - Combine probabilities using ***Bayes' rule***:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

That is:

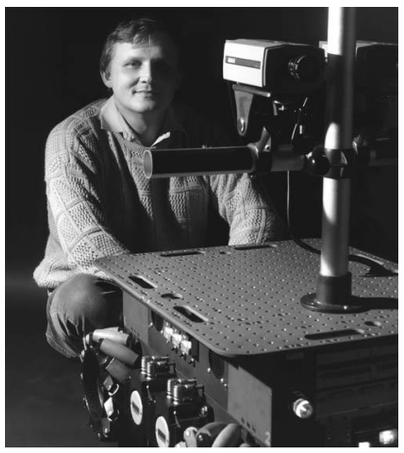
$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalizing constant}}$$

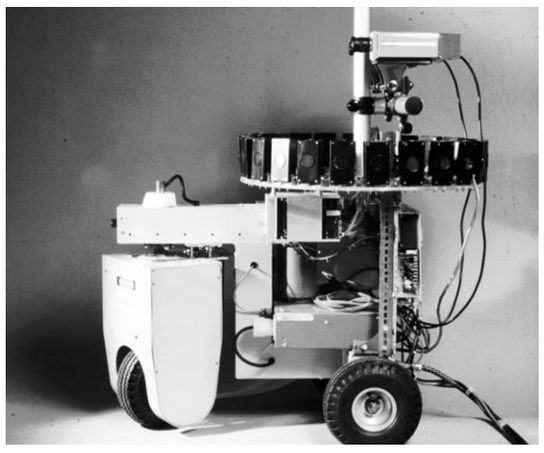
- Pioneers of approach:

  Elfes and Moravec at CMU in 1980s

  https://frc.ri.cmu.edu/~hpm/talks

Neptune





Basic probability theory

- Probability function:
  - *Gives values from 0 to 1 indicating whether a particular event, H (Hypothesis), has occurred*

- For sonar sensing:
  - *Experiment: Sending out acoustic wave and measuring time of flight*
  - *Outcome: Range reading reporting whether the region being sensed is Occupied or Empty*

- Hypotheses (H) = {Occupied, Empty)

- Probability that H has really occurred:

  $0 < P(H) < 1$

- Probability that H has not occurred:

  $1 - P(H)$

Basic probability theory

Probability function:

– Gives values from 0 to 1 indicating whether a particular event, H (Hypothesis), has occurred

- For sonar sensing:
  - – Experiment: Sensing out acoustic waves and measuring the time of flight
  - – Outcome: Range reading reporting whether the region being sensed is Occupied or Empty

- H = {Occupied, Empty)

- Probability that H has really occurred:

$0 < P(H) < 1$

- Probability that H has not occurred:

$1 – P(H)$

Unconditional and conditional probabilities

- Unconditional probability: $P(H)$
  - *"Probability of* H"
  - *Only provides* a priori *information*
  - *For example, could give the known distribution of rocks in the environment, e.g., "x% of environment is covered by rocks"*
  - *For robotics, unconditional probabilities are not based on sensor readings*

- For robotics, we want: Conditional probability: $P(H \mid s)$
  - *"Probability of* H, *given* s" *(e.g., P(Occupied | s), or P(Empty | s))*
  - *These are based on sensor readings,* s

- Note: $P(H \mid s) + P(\text{not } H \mid s) = 1.0$

$$P(H \mid s) = \frac{P(s \mid H)P(H)}{P(s \mid H)P(H) + P(s \mid notH)P(notH)}$$

$P(s)$

Probabilities for occupancy grids

- For each grid element, grid[i][j], store tuple of the two probabilities:

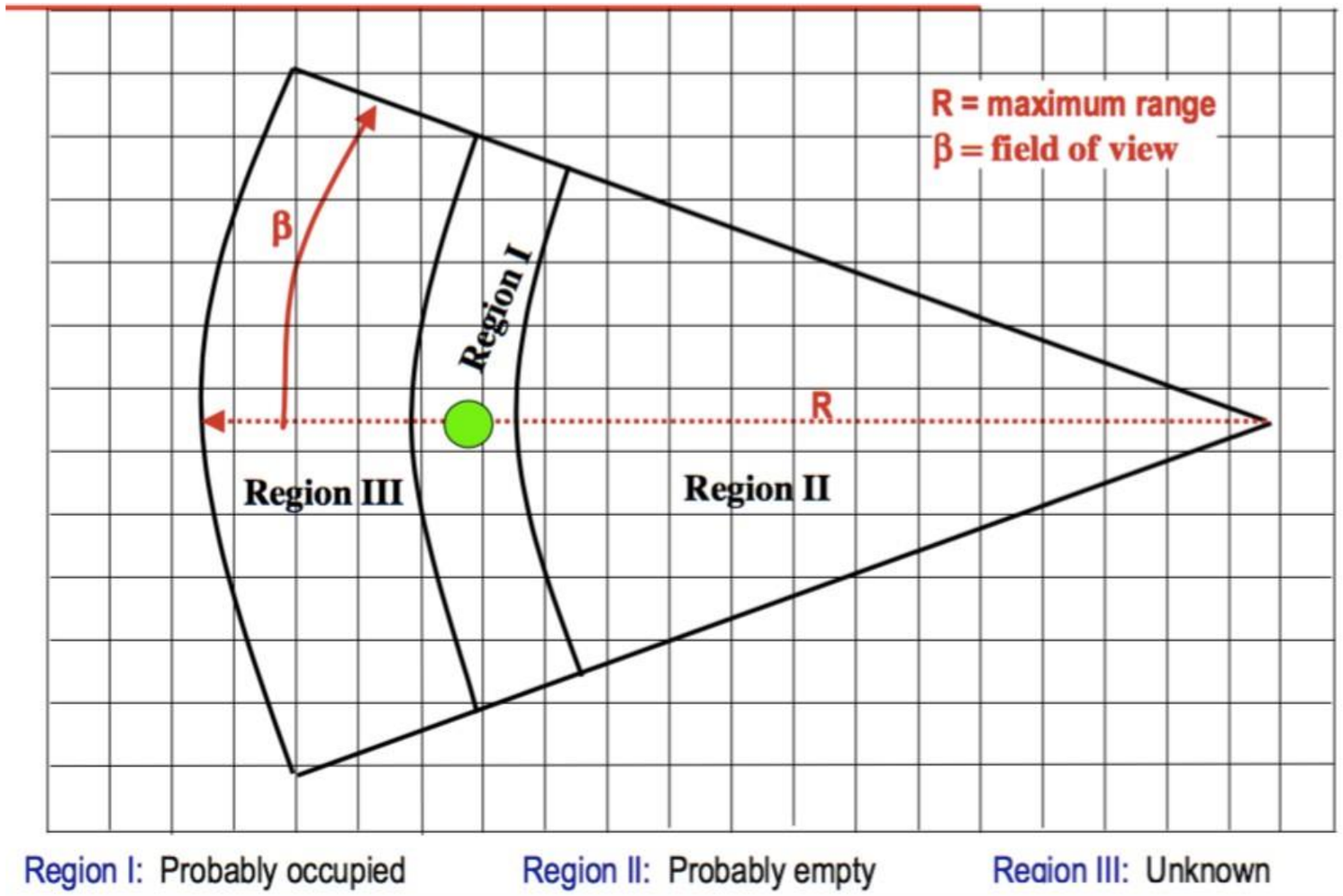
  Compute: P(Occupied | s) and P(Empty | s)

- For each grid element, grid[i][j], store tuple of the two probabilities:

```
typedef struct {
    double occupied;  // i.e., P(occupied | s)
    double empty;     // i.e., P(empty | s)
  } P;

P occupancy_grid[ROWS][COLUMNS];
```

Using sensor model to get P(s|H)

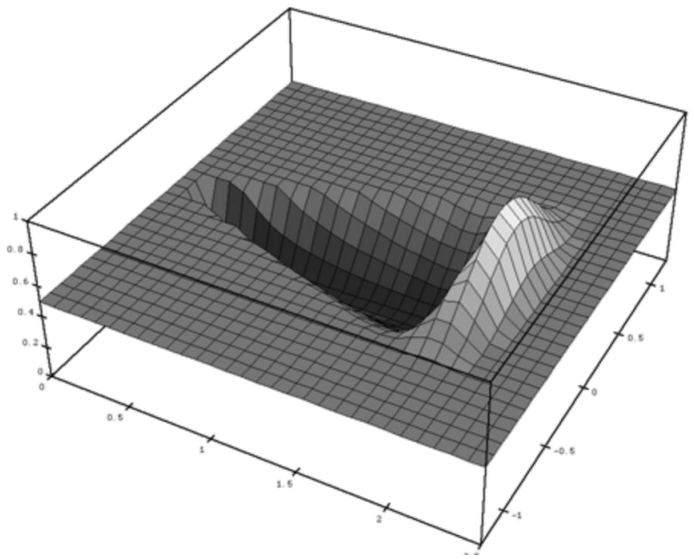


P(*s*|H): given the sensor model, what is the probability of getting reading *s* ?

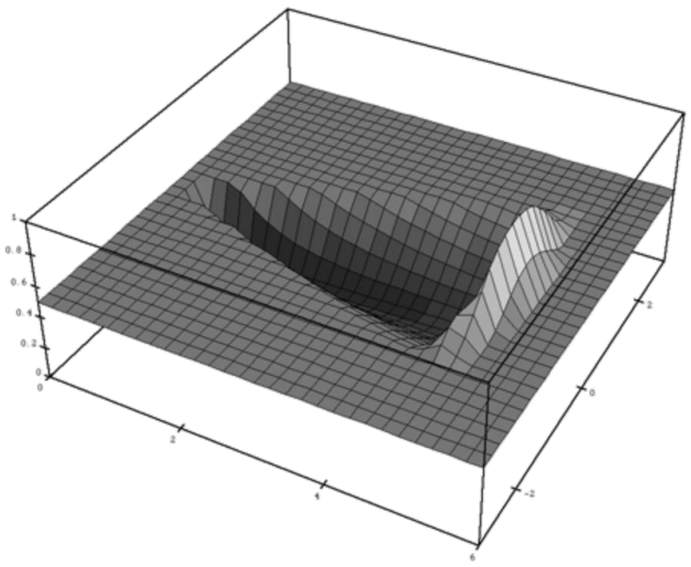
- Given the evidence that cell (x,y) belongs to Region I (probably occupied), what is the probability of a sensor reading *s*?
- Given the evidence that cell (x,y) belongs to Region II (probably empty), what is the probability of a sensor reading s?

Naive sensor model

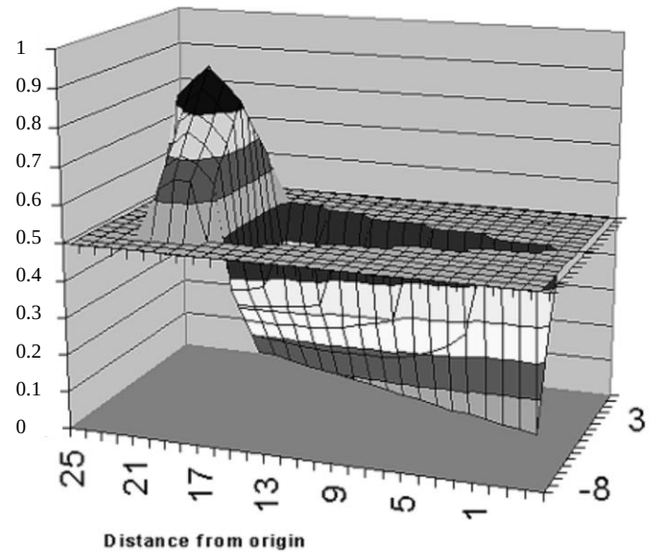
2 ft



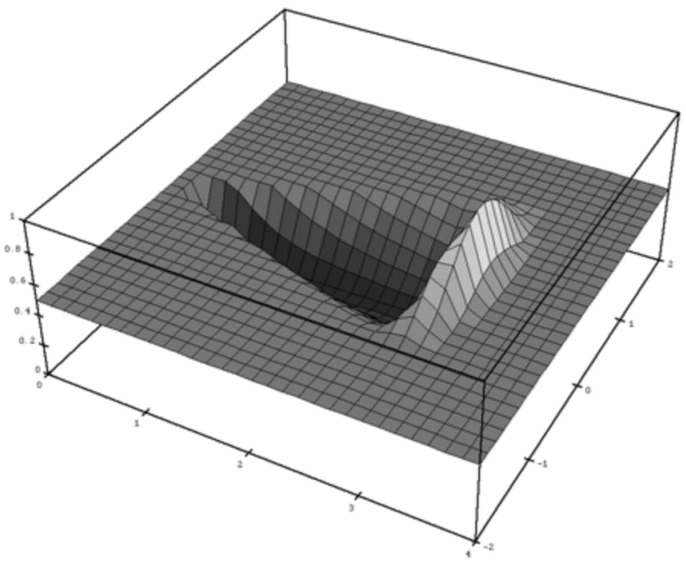
5 ft



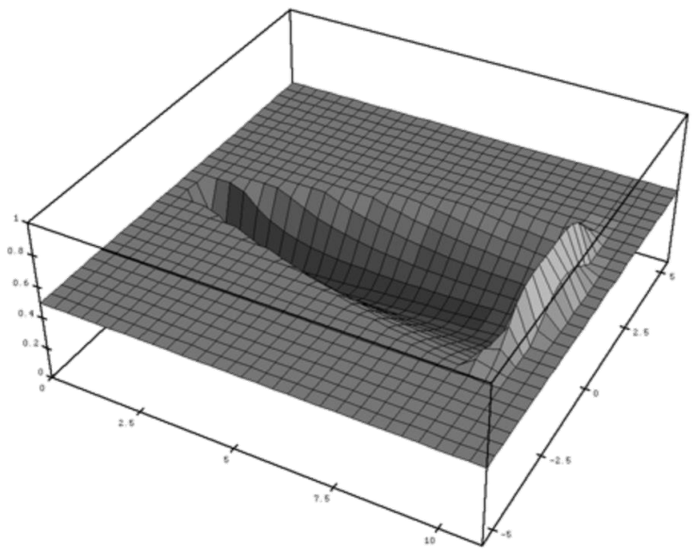
Certainty



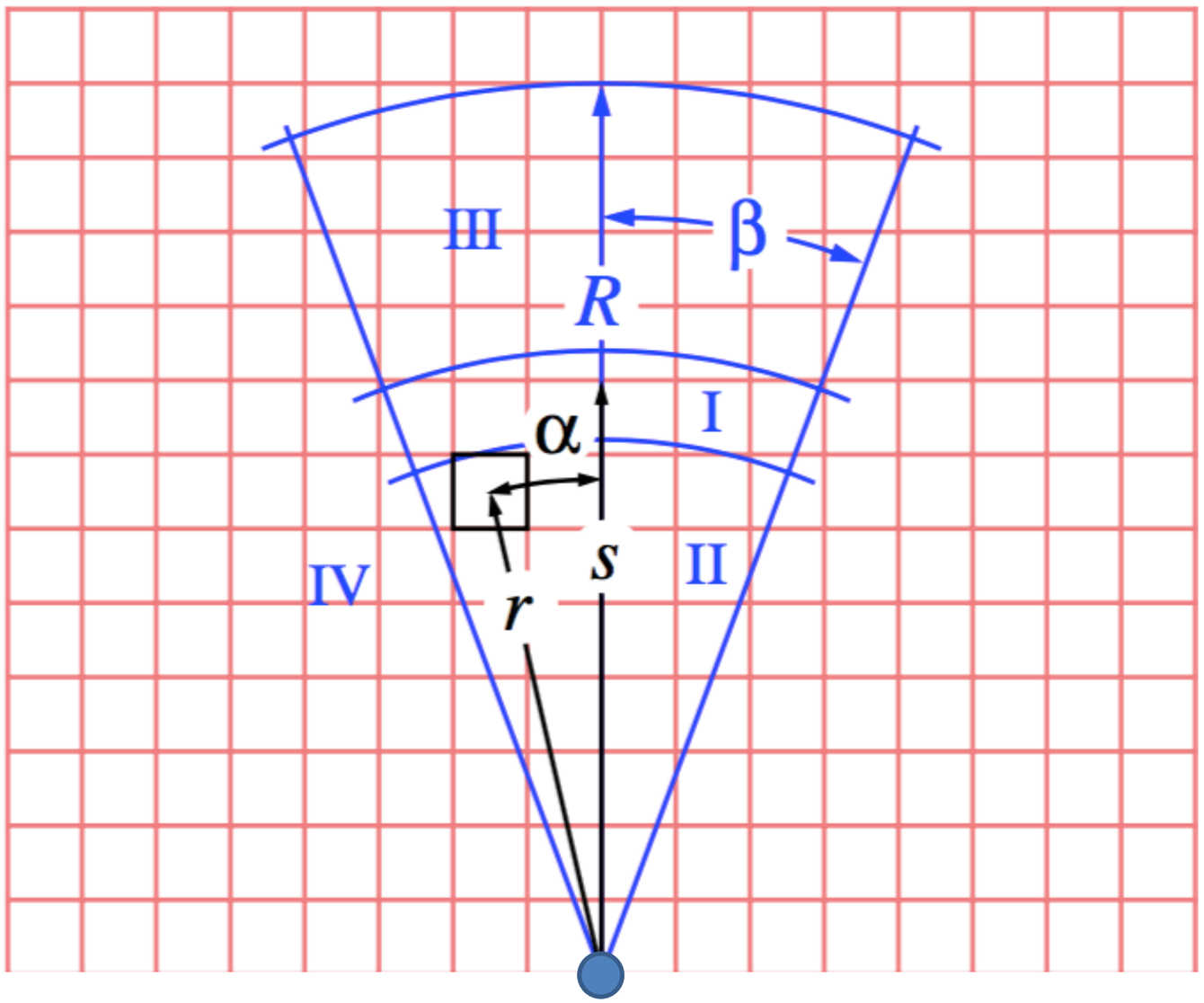
3 ft



10 ft



Parameters for reasoning about a cell



Model for Region I

$$P(s|H) = \frac{\left(\frac{R-r}{R}\right)+\left(\frac{\beta-\alpha}{\beta}\right)}{2} \times M$$

Occupied

$$P(s|\neg H) = 1 - \Pr(C)$$

Empty

Model for Region II

$$P(s|H) = 1 - \Pr(\bar{C})$$

Occupied

$$P(s|\neg H) = \frac{\left(\frac{R-r}{R}\right)+\left(\frac{\beta-\alpha}{\beta}\right)}{2}$$

Empty

Converting sonar readings to probabilities: Region I

Based on the sensor model, if a cell C belongs to Region I, *P*(Occupied) is the probability that the cell is occupied, that depends on its range *r* and bearing α

- Region I:

The nearer the grid element to the origin of the sonar beam, the higher the belief

The closer to the acoustic axis, the higher the belief

$$P(\text{Occupied}) = \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \times Max_{occupied}$$

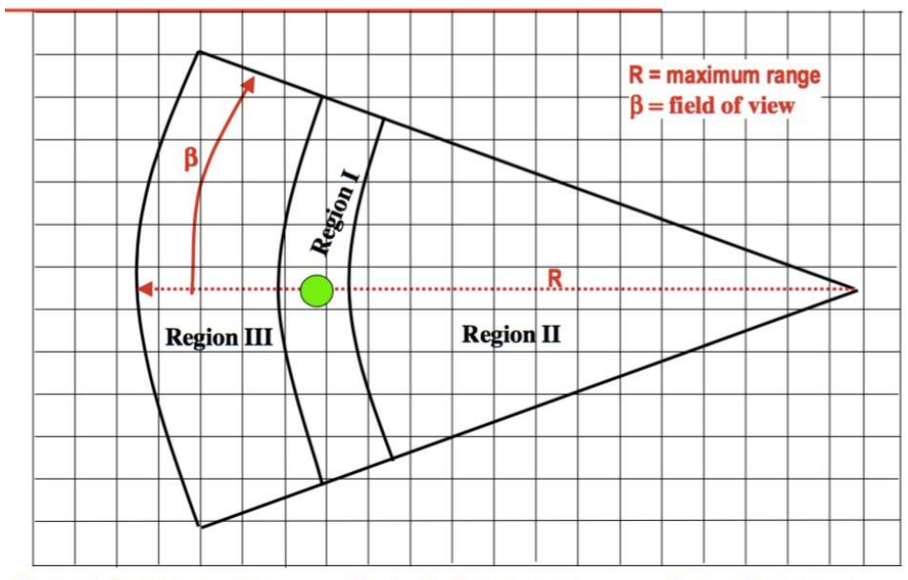
We never know with certainty

where r is distance to grid element that is being updated

α is angle to grid element that is being updated

$Max_{occupied}$ *= highest probability possible (e.g., 0.98)*

$$P(\text{Empty}) = 1.0 - P(\text{Occupied})$$



Converting sonar readings to probabilities: Region II

Based on the sensor model, if a cell C belongs to Region II, *P*(Empty) is the probability that the cell is empty, that depends on its range *r* and bearing α

- Region II:

The nearer the grid element to the origin of the sonar beam, the higher the belief

The closer to the acoustic axis, the higher the belief

$$P(\text{Empty}) = \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2}$$

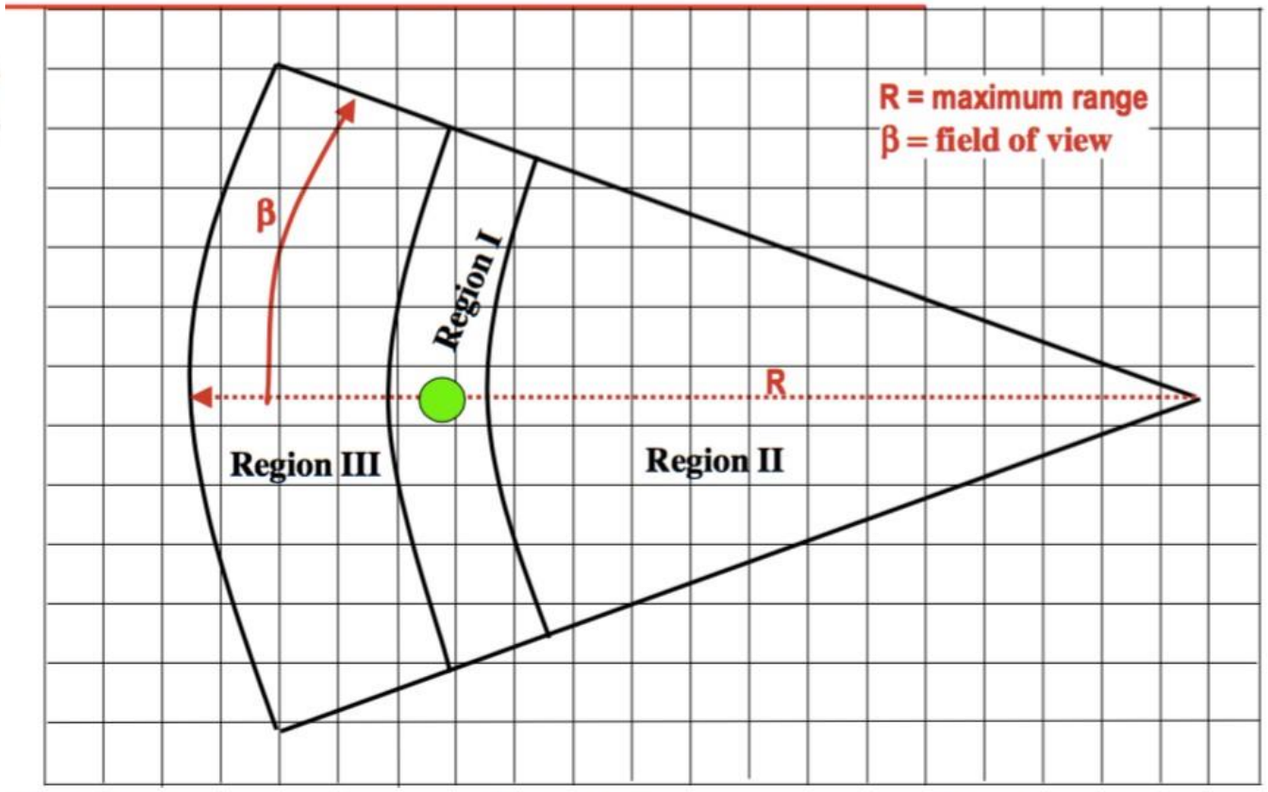
$$P(\text{Occupied}) = 1.0 - P(\text{Empty})$$

where r *is distance to grid element being updated,*
α is angle to grid element being updated

Note that here, we allow probability of being empty to equal 1.0

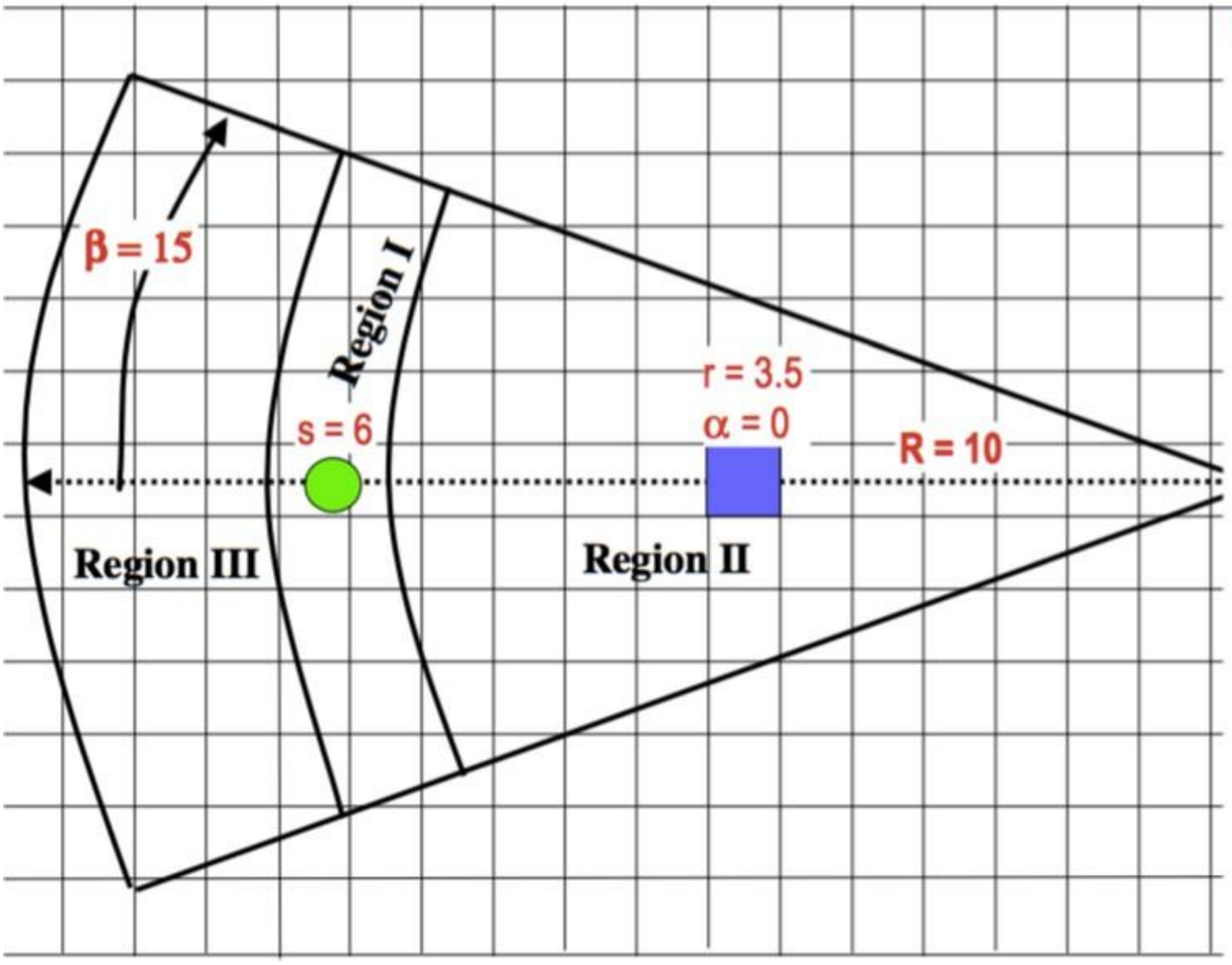
Sensor tolerance

- Sonar range readings have resolution error
- Thus, specific reading might actually indicate range of possible values
- E.g., reading of 0.87 meters actually means within (0.82, 0.92) meters
  - *Therefore, tolerance in this case is 0.05 meters.*
- Tolerance gives width of Region I



Value of a grid cell

Example: What is value of grid cell ■ ? (assume tolerance = 0.5)



Which region?

3.5 < (6.0 – 0.5) ➔ Region II

$$P(\text{Empty}) = \frac{\frac{10-3.5}{10} + \frac{15-0}{15}}{2}$$

$$= 0.83$$

P(Occupied) = (1 – 0.83) = 0.17

We have P(s | H) but we need P(H | s)

- Note that previous calculations were only based on the sensor model: the probability that a cell C is occupied or empty based on the sensor model, that is, based on its position in the model regions
- These probabilities provide $P(s \mid H = \{\text{C Occupied, C Empty}\})$
- In the model a cell C is identified by (r,α), therefore, given the evidence of a cell at $s = (r,\alpha)$ being either empty or occupied, previous probabilities correspond to the conditional probability of getting such a reading s: $P(s \mid H)$
- For instance, if $s = (r,\alpha) = (6,5)$ and we have the evidence that the cell is in Region I, the conditional probability $P(s|H)$ of obtaining a reading s is:

$$P(s \mid H = \text{Occupied}) = \frac{\left(\frac{10-6}{10}\right) + \left(\frac{15-5}{15}\right)}{2} \cdot 0.98 = 0.52$$

We want to compute P(H | s)

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

$$P(H \mid s) = \frac{P(s \mid H)P(H)}{P(s \mid H)P(H) + P(s \mid notH)P(notH)}$$

$P(s)$

$$P(H \mid s) = \frac{P(s \mid Empty)P(Empty)}{P(s \mid Empty)P(Empty) + P(s \mid Occupied)P(Occupied)}$$

- $P(s \mid Occupied)$ and $P(s \mid Empty)$ are known from sensor model
- $P(Occupied)$ and $P(Empty)$ are unconditional, prior probabilities (which may or may not be known)
  - *If not known, okay to assume* P(Occupied) = P(Empty) = *0.5*

Back to the example

- Let's assume we're on Mars, and we know that $P(Occupied) = 0.75$
- Continuing same example for cell  …

- $$P(Empty \mid s{=}6) = \frac{P(s \mid Empty)\, P(Empty)}{P(S \mid Empty)\, P(Empty) + P(s \mid Occupied)\, P(Occupied)}$$

$$= \frac{0.83 \times 0.25}{0.83 \times 0.25 + 0.17 \times 0.75}$$

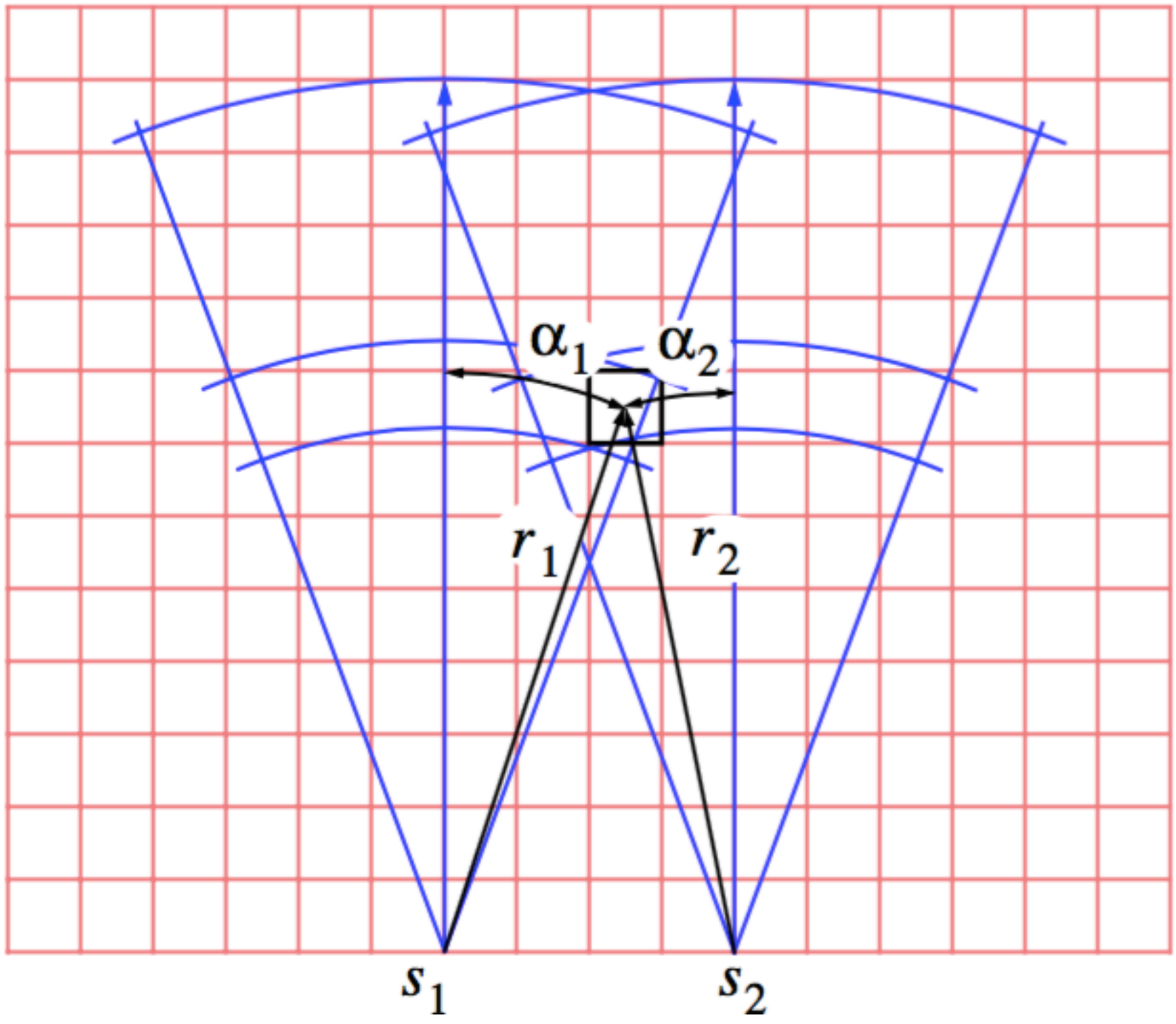
$$= 0.62$$

- $P(Occupied \mid s{=}6) = 1 - P(Empty \mid s{=}6) = 0.38$

- *These are the values we store in our grid cell representation*

Updating with bayes' rule: information fusion

- How to fuse multiple readings obtained over time?

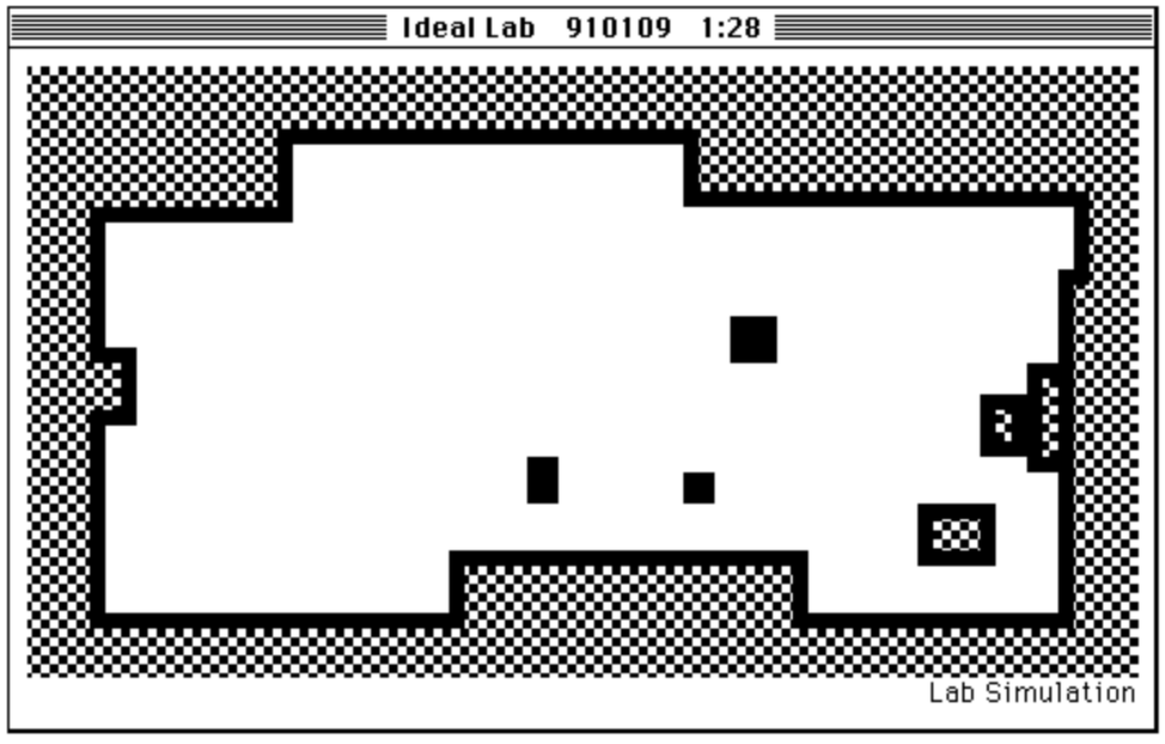


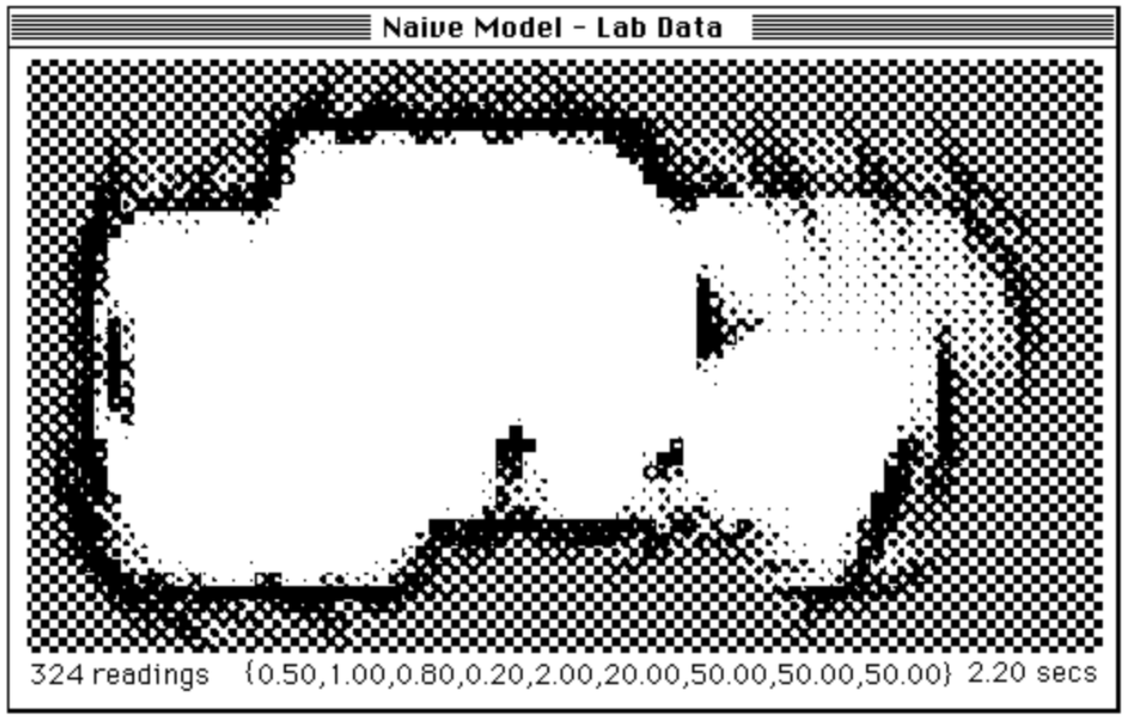
Updating with Bayes' rule: Information fusion

- How to fuse multiple readings obtained over time?
- First time:
  - *Each element of grid initialized with* a priori *probability of being occupied or empty*
- Subsequently:
  - *Use Bayes' rule iteratively*
    - *Probability at time* t_{n-1} *becomes prior and is combined with current observation at* t_n *using recursive version of Bayes rule:*

$$P(H \mid s_n) = \frac{P(s_n \mid H)P(H \mid s_{n-1})}{P(s_n \mid H)P(H \mid s_{n-1}) + P(s_n \mid \neg H)P(\neg H \mid s_{n-1})}$$

Updating with Bayes' rule: Information fusion





https://frc.ri.cmu.edu/~hpm/talks/Sonar.figs/Learn.html

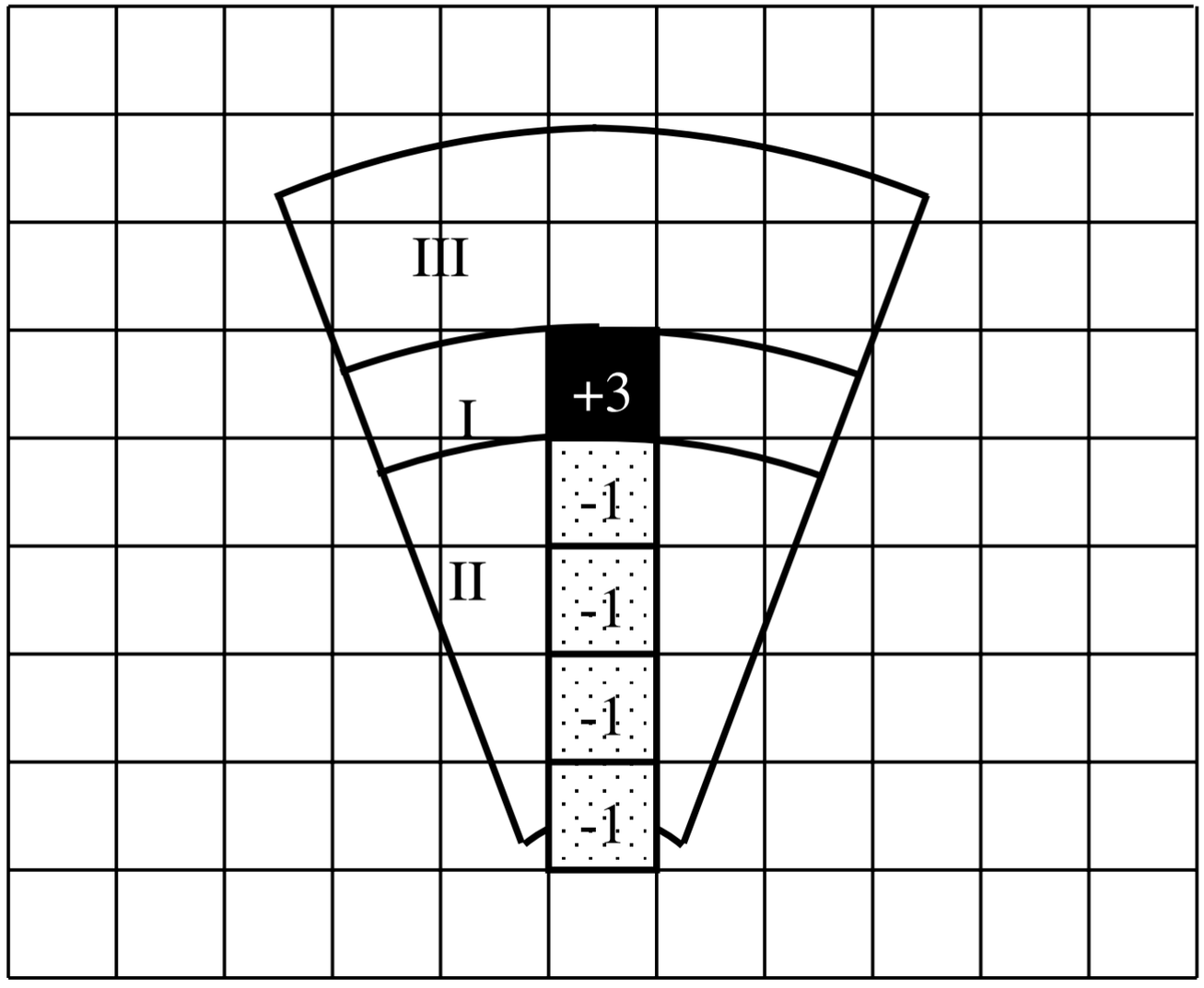
HIMM (Histogrammic in Motion Mapping)

Histogrammic in Motion Mapping (HIMM) algorithm
- developed by Borenstein and Koren at the University of Michigan
- Scoring instead of probability

Motivation: Improve obstacle avoidance
- Safely run their robot CARMEL
- Top speed: 0.8 meters/sec
- Needed to update an occupancy grid on the order of 160ms
- Processor: 20MHz 80386 (1992)
- Bayesian model was slow
- They won 1992 AAAI Mobile Robot Competition
- They used HIMM + VFH
- After that HIMM became standard on many…

HIMM (Histogrammic in Motion Mapping)

- Only elements along the acoustic axis are updated.
- Eliminates up to 90% of the grid elements that are updated by Bayesian
- Scores: 0 to 15 (in one byte rather that floating point)



HIMM Sonar model

HIMM (Histogrammic in Motion Mapping)

- Robot should move fast (Robot sees too little of the world)
- Velocity and grid update rate should match

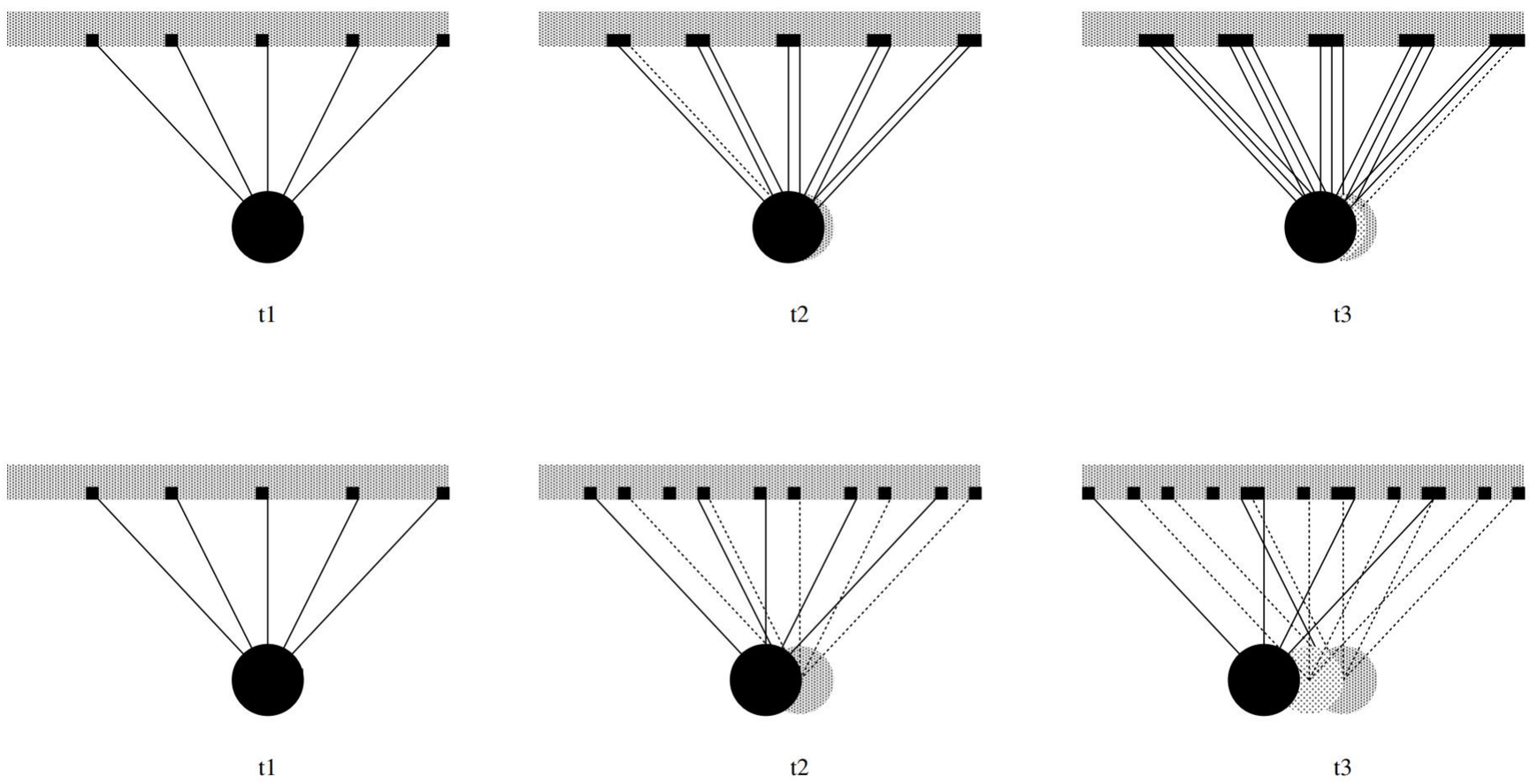


Figure 11.10 HIMM updating example for a wall a.) when velocity and update rates are well matched, and b.) when update rate is slower than velocity, leading to holes.

HIMM (Histogrammic in Motion Mapping)