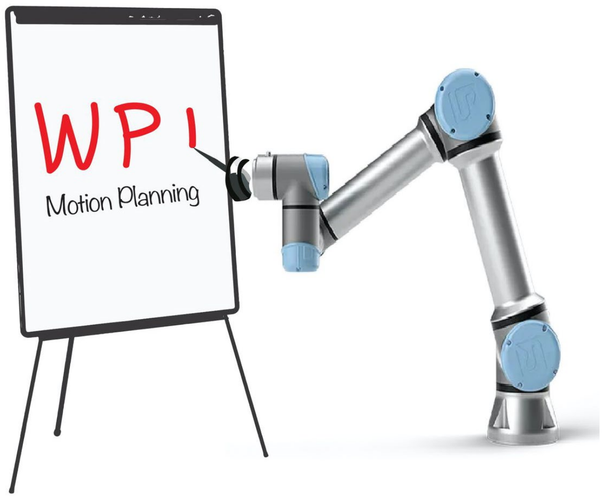
RBE550
Motion Planning
Bugs and Potential Fields



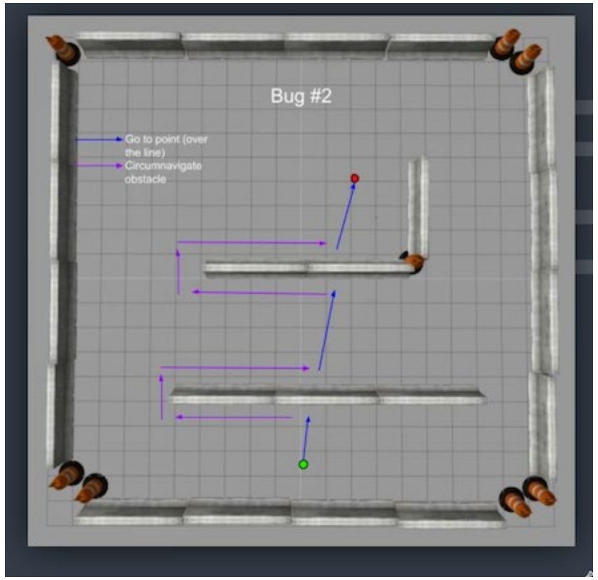
Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

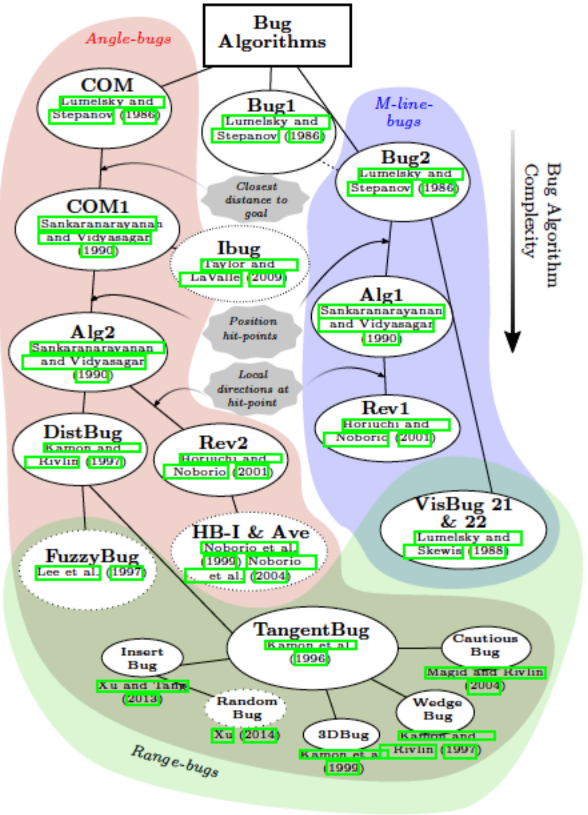
Acknowledgements

The slides are a compilation of work based on notes and slides from Constantinos Chamzas, Morteza Lahijanian, Howie Choset, Lydia Kavraki, Greg Hager, Mark Moll, G. Ayorkor Mills-Tetty, Hyungpil Moon, Zack Dodds, Nancy Amato, Steven Lavalle, Seth Hutchinson, George Kantor, Dieter Fox, Vincent Lee-Shue Jr., Prasad Narendra Atkar, Kevin Tantiseviand, Bernice Ma, David Conner, and other members of previous versions of this class. HC wants to especially thank Greg Hager and Ji Yeong Lee for their help in supplying template slides and figures.

Overview

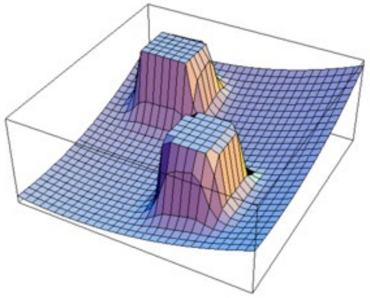
Bug Algorithms

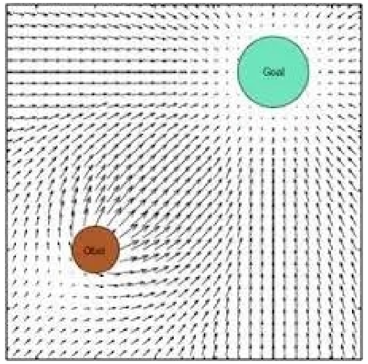


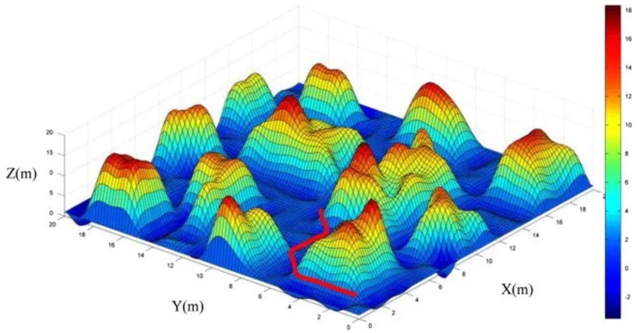


From McGuire at al, 2018

Potential Fields

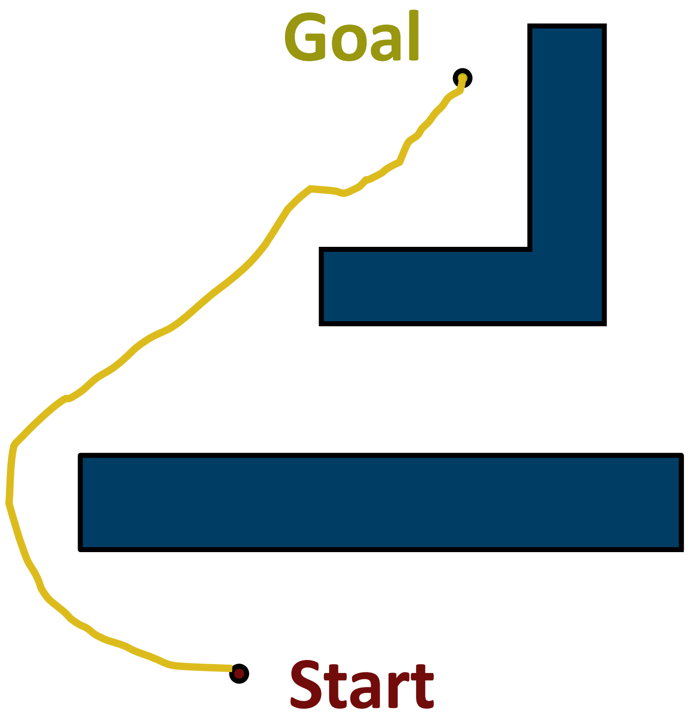






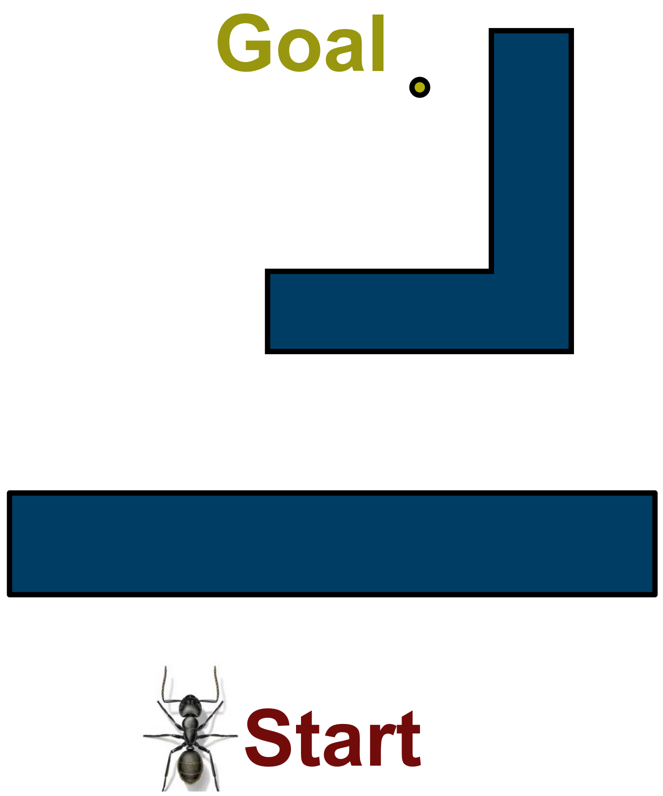
Motion Planning Definition (Point Robot)

- **Workspace**: The environment in which the robot operates
  - Notation: W
  - Generally, either 2- or 3-Dimensional Euclidean space,
  - For 2-D point robot $W = \mathbb{R}^2$
    - Obstacle i in workspace W is denoted by WO_i
    - Free workspace: $W_{\text{free}} = W \setminus \cup_i WO_i$
- **Configuration Space:** a complete specification of the robot's state (informal definition)
  - Notation: Q or X or C
  - Dimension generally depends on the robot
  - For point robot $Q = W = \mathbb{R}^2$ and
  - Collision-free space $Q_{\text{free}} = W_{\text{free}}$, and $q_{start}, q_{goal} \in Q_{free}$
- **Robot path from q_{start} to q_{goal} :** A continuous curve $\sigma(t)$ such that $\{\sigma(t) \in Q_{free} \mid \sigma(0) = \text{q}_{\text{start}}, \sigma(1) = \text{q}_{\text{goal}}\}$



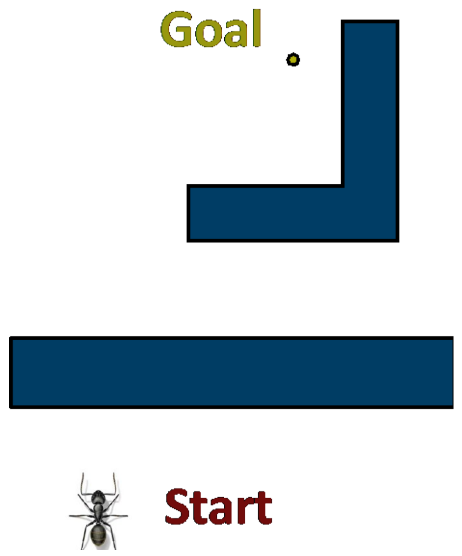
Think Like a Bug

- How would you navigate like a bug?
  - You can smell the food
  - You can't see the walls but you can feel them
- Can you come up with an algorithm?



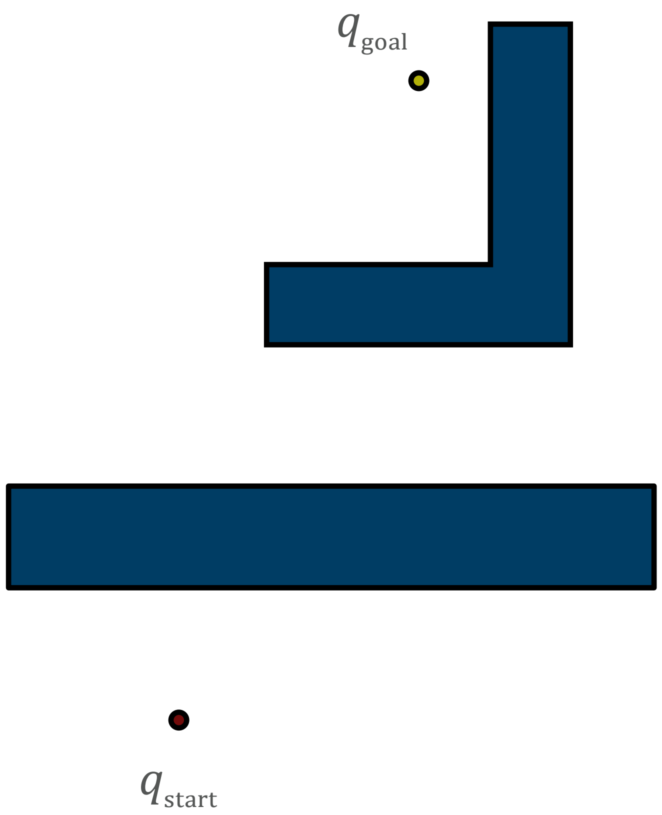
Assumptions

- Known direction to goal:
  - Robot can measure distance $d(x, y)$ between points $x, y \in W$
- Local sensing :
  - Sense walls/obstacles
  - Know position all the time (perfect encoders)
- Reasonable world:
  - Finitely many obstacles in finite area
  - A line will intersect an obstacle finitely many times
  - Workspace is bounded



Bugginer Strategy

“Bug 0” Algorithm

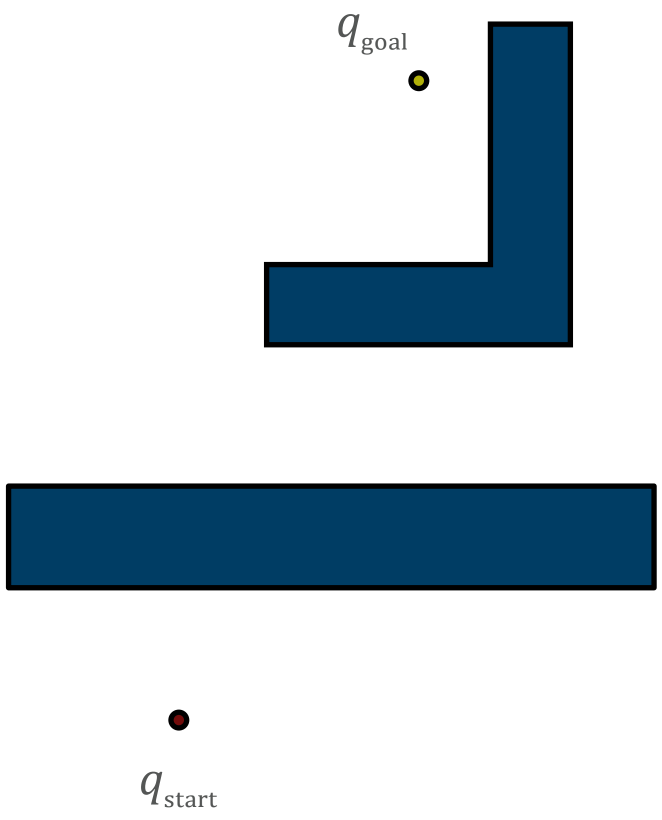


- **Assumptions:**
  - Known direction to goal
  - Local sensing walls/obstacles & encoders
  - Reasonable world

How?

Bugginner Strategy

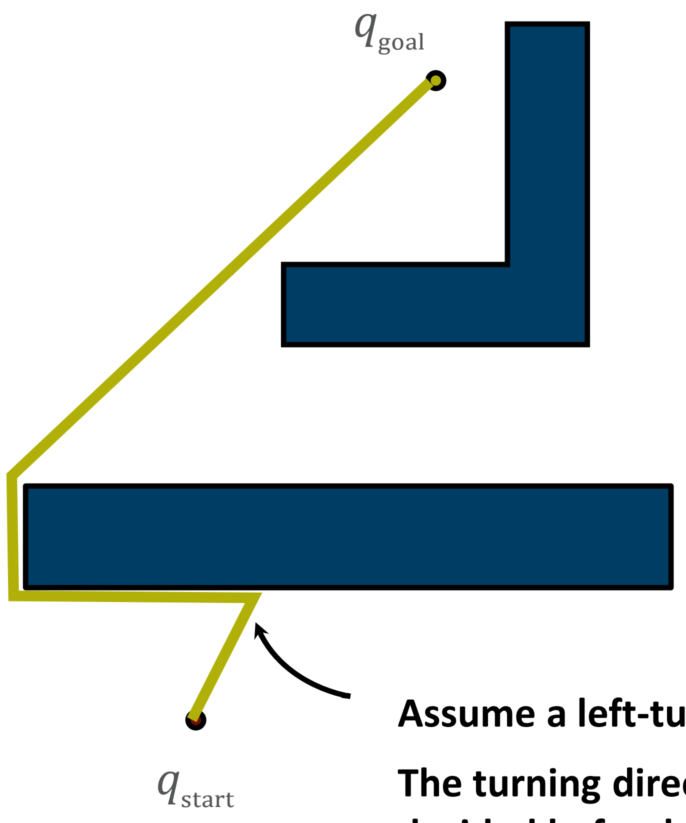
"Bug 0" Algorithm



- **Assumptions:**
  - Known direction to goal
  - Local sensing walls/obstacles & encoders
  - Reasonable world
- **Strategy:**
  1. Head toward goal
  2. Follow obstacles until you can follow the goal again
  3. Continue

Path?

"Bug 0" Algorithm



Assume a left-turning robot.

The turning direction is decided beforehand.

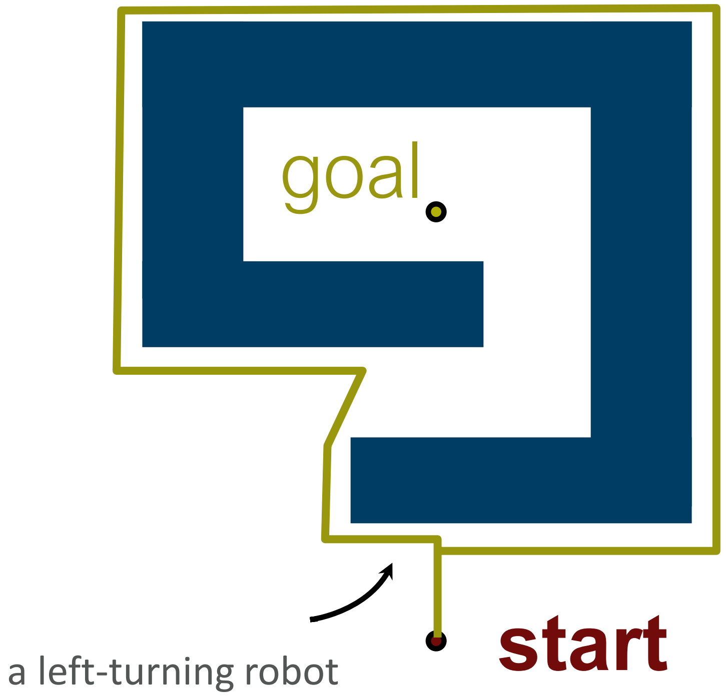
"Bug 0" Strategy:

1. Head toward goal
2. Follow obstacles until you can follow the goal again
3. Continue

Done?

Bug Zapper

Will "Bug 0" Strategy: work here?



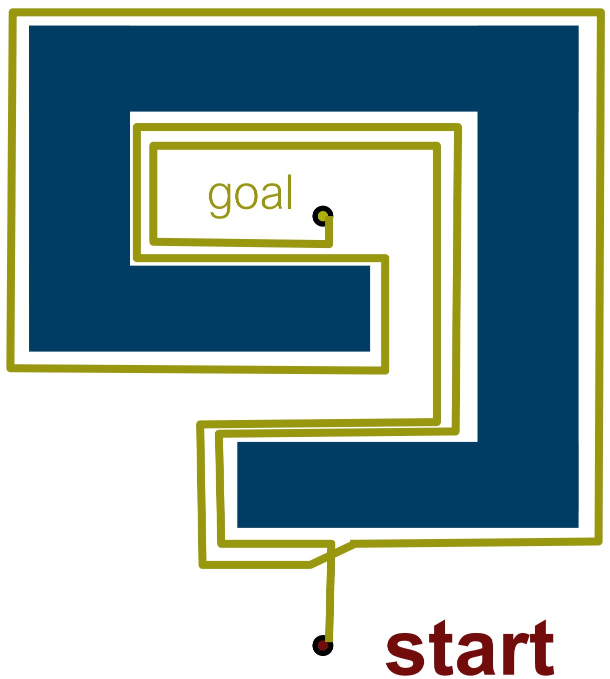
"Bug 0" Strategy:

1. Head toward goal
2. Follow obstacles until you can follow the goal again
3. Continue

How can we make it smarter?

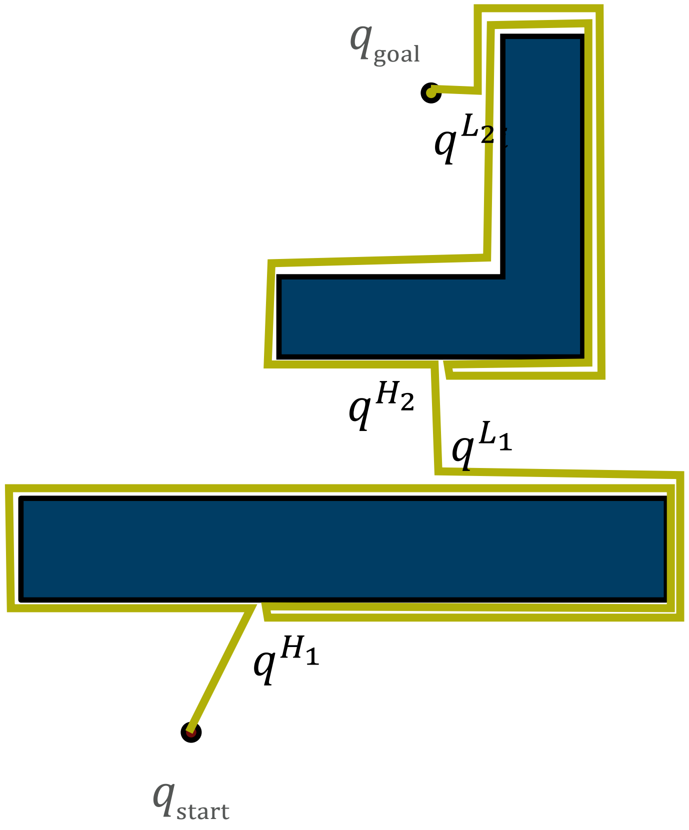
- Add memory!

“Bug 1” Algorithm



- **Assumptions:**
  - Known direction to goal
  - Local sensing walls/obstacles & encoders
  - Reasonable world
  - Has Memory
- **Strategy:**
  1. Head toward goal
  2. If an obstacle is encountered, circumnavigate it and remember how close you get to the goal
  3. Then return to that closest point (by wall-following) and continue

"Bug 1" Algorithm



"Bug 1" Strategy:

1. Head toward goal
2. If an obstacle is encountered, circumnavigate it and remember how close you get to the goal
3. Then return to that closest point (by wall-following) and continue

Some notation:

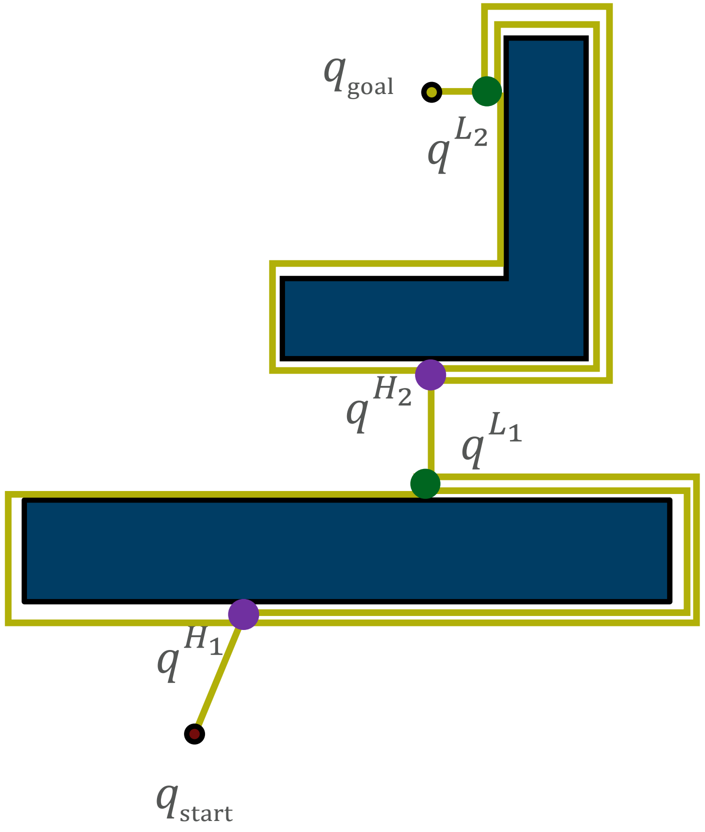
- Start and goal positions: q_{start} and q_{goal}
- "hit point": q^{H_i}
- "leave point": q^{L_i}
- Path: a sequence of hit/leave pairs bounded by q_{start} and q_{goal}

"Bug 1" Algorithm pseudocode

Algorithm: Bug 1

Input: A point robot with a tactile sensor
Output: A path to the q_{goal} or a conclusion no such path exists

- Let $q^{L_0} = q_{\text{start}};\ i = 1$
- **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 point 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



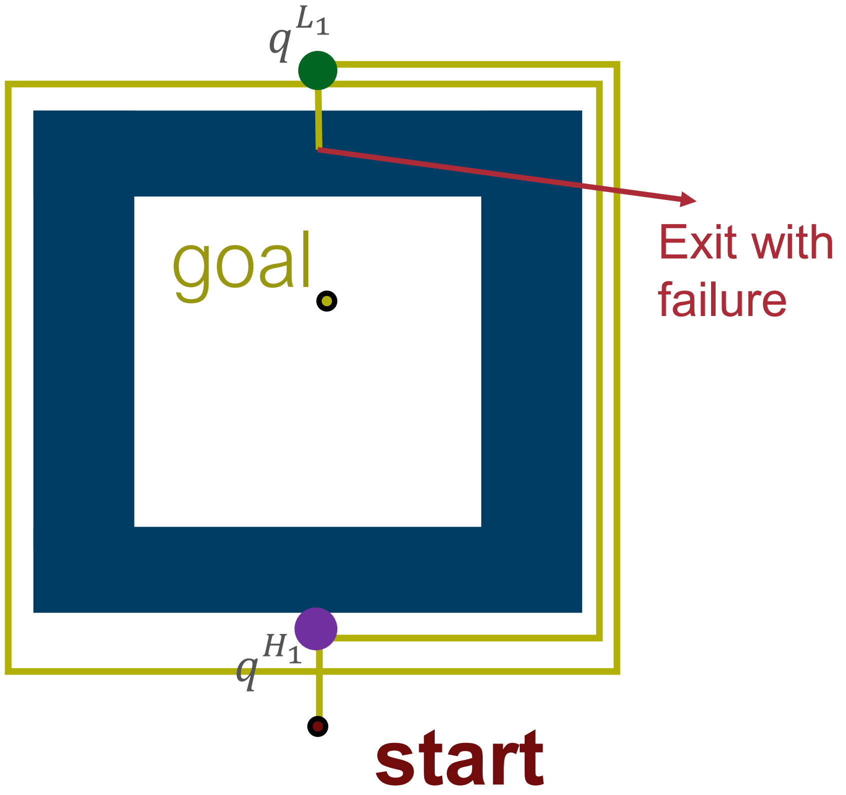
q^{H_i}: hit point $\qquad$ q^{L_i}: leave point

Failure of "Bug 1" Algorithm

Algorithm: Bug 1

Input: A point robot with a tactile sensor
Output: A path to the q_{goal} or a conclusion no such path exists

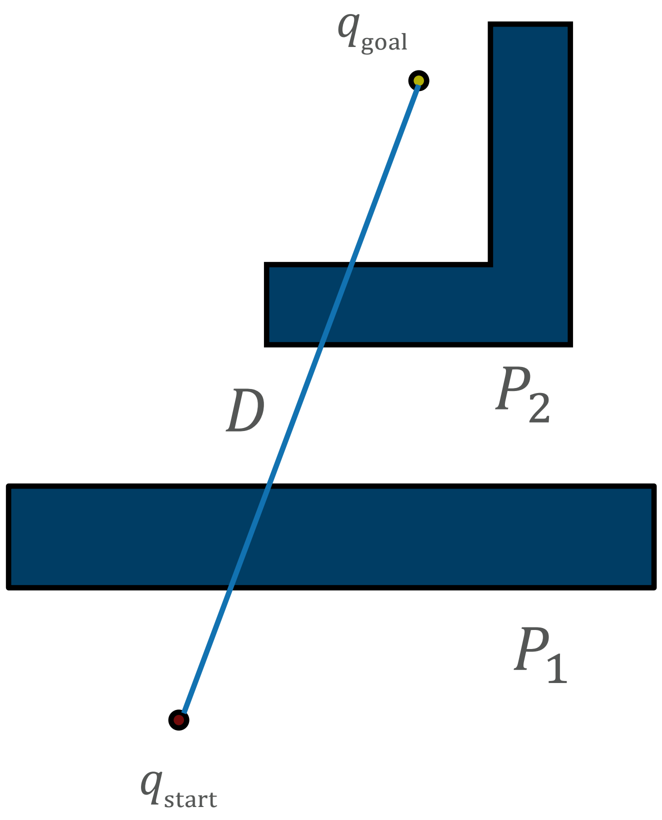
- Let $q^{L_0} = q_{\text{start}};\ i = 1$
- **Repeat**
  - **Repeat**
    - from $q^{L_{i-1}}$ move toward q_{goal}
  - **Until** goal is reached or obstacle encountered at q^{H_i}
  - **If** goal is reached,
    - exit
  - **Repeat**
    - follow boundary recording point q^{L_i} with shortest distance 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



q^{H_i}: hit point q^{L_i}: leave point

"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:

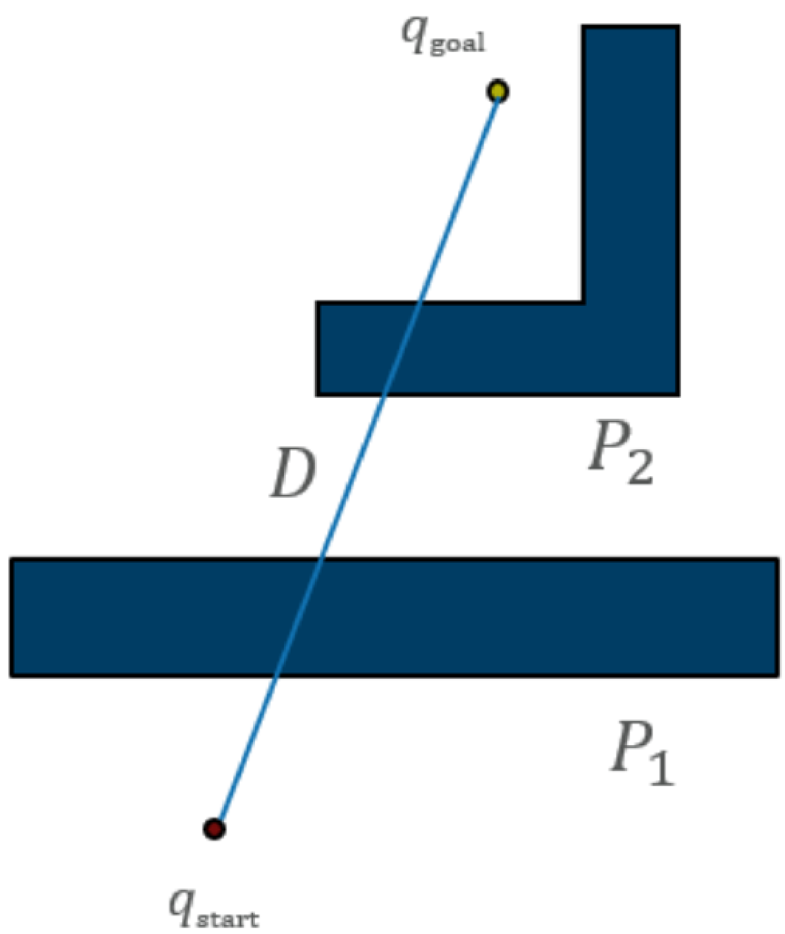
What's the shortest distance it might travel?

Upper bound:

What's the longest distance it might travel?

What is the lower bound for the path that bug1 can take

Bug 1: Path Bounds



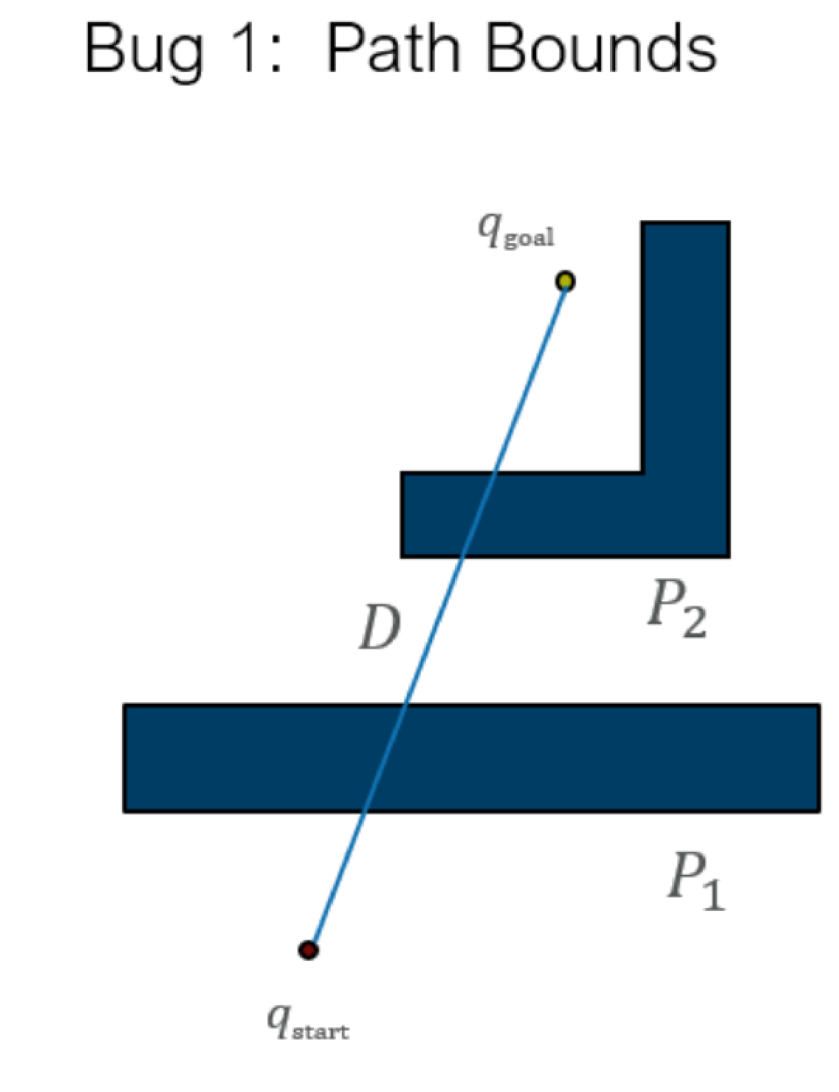
D

$P_1 + P_2$

$2 * D$

$D + P_1 + P_2$

What is the lower bound for the path that bug1 can take



D

0%

$P_1 + P_2$

0%

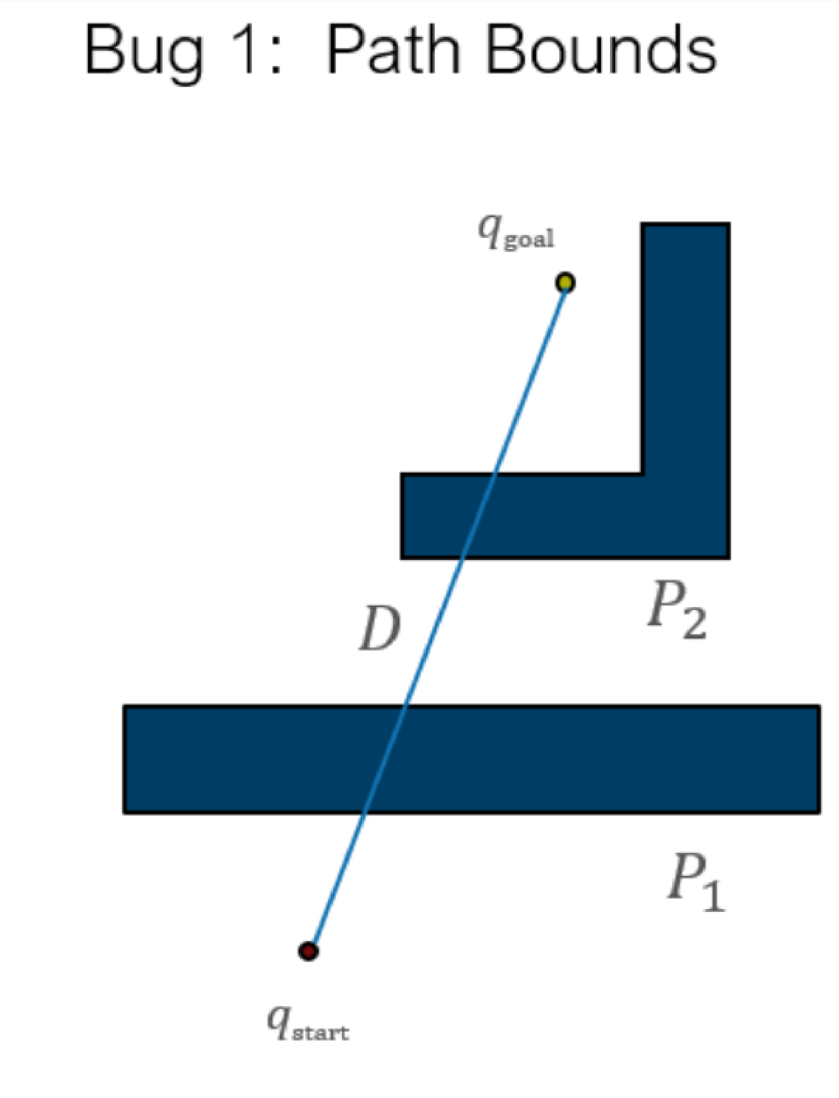
$2 * D$

0%

$D + P_1 + P_2$

0%

What is the lower bound for the path that bug1 can take



D

0%

$P_1 + P_2$

0%

$2 * D$

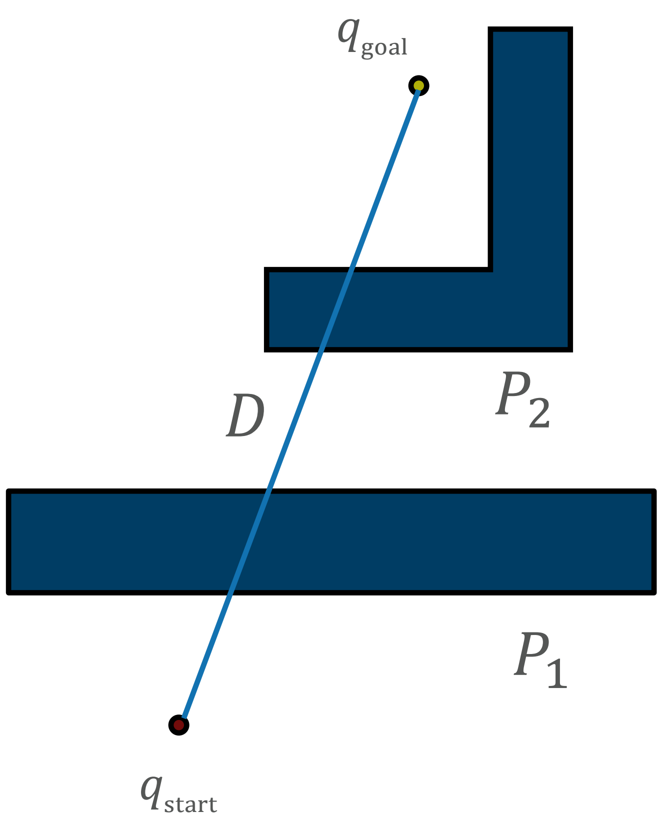
0%

$D + P_1 + P_2$

0%

"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?

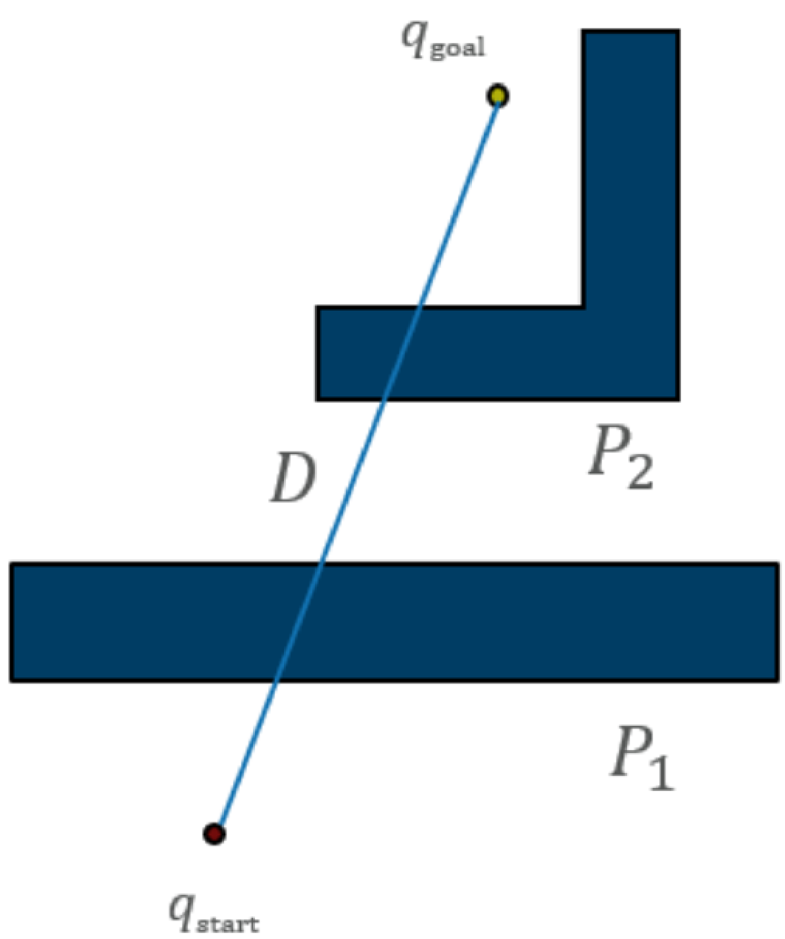
D

Upper bound:

What's the longest distance it might travel?

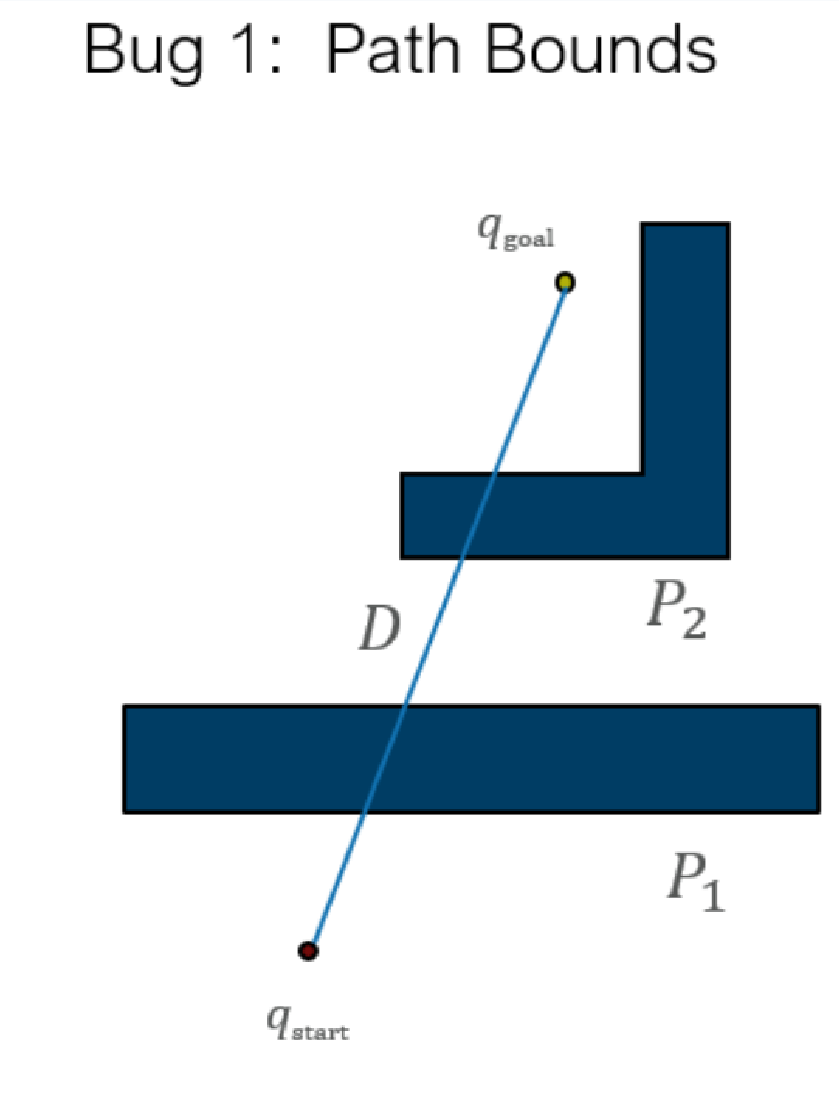
What is the upper bound for the path that bug1 can take

Bug 1: Path Bounds



- $3D$
- $P_1 + P_2$
- $1.5(P_1 + P_2) + D$
- $2.5(P_1 + P_2) + D$

What is the upper bound for the path that bug1 can take



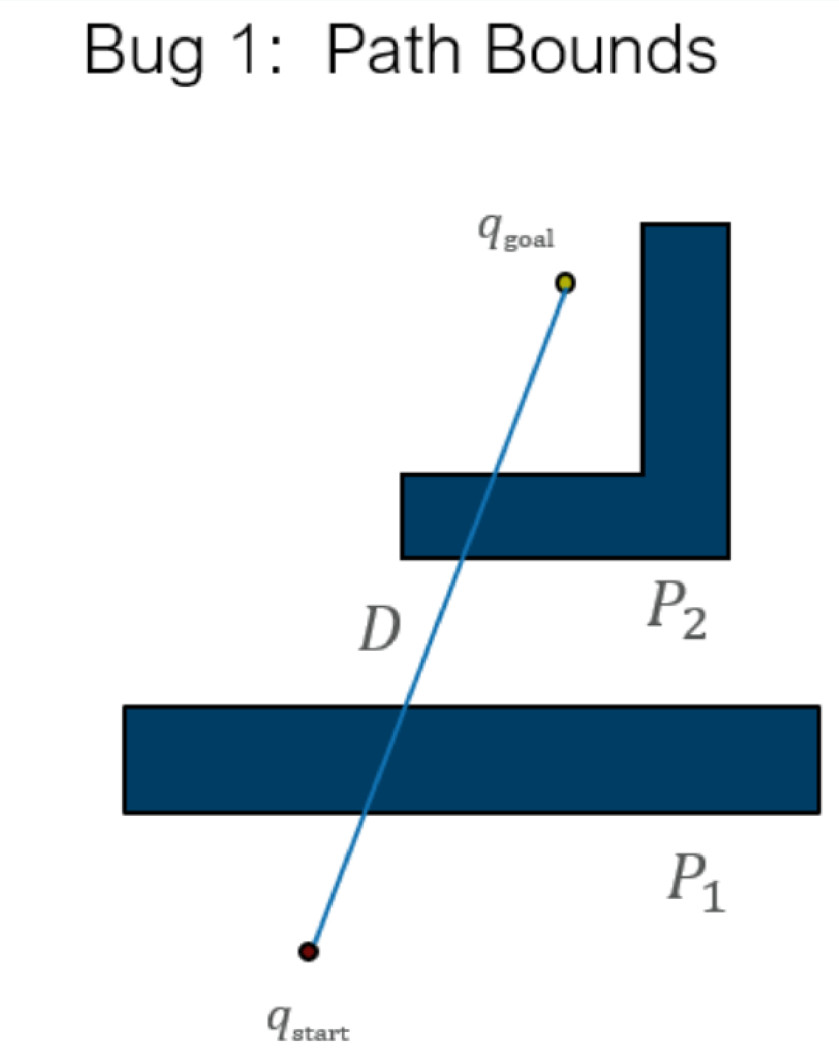
$3D$ 0%

$P_1 + P_2$ 0%

$1.5(P_1 + P_2) + D$ 0%

$2.5(P_1 + P_2) + D$ 0%

What is the upper bound for the path that bug1 can take



$3D$

0%

$P_1 + P_2$

0%

$1.5(P_1 + P_2) + D$

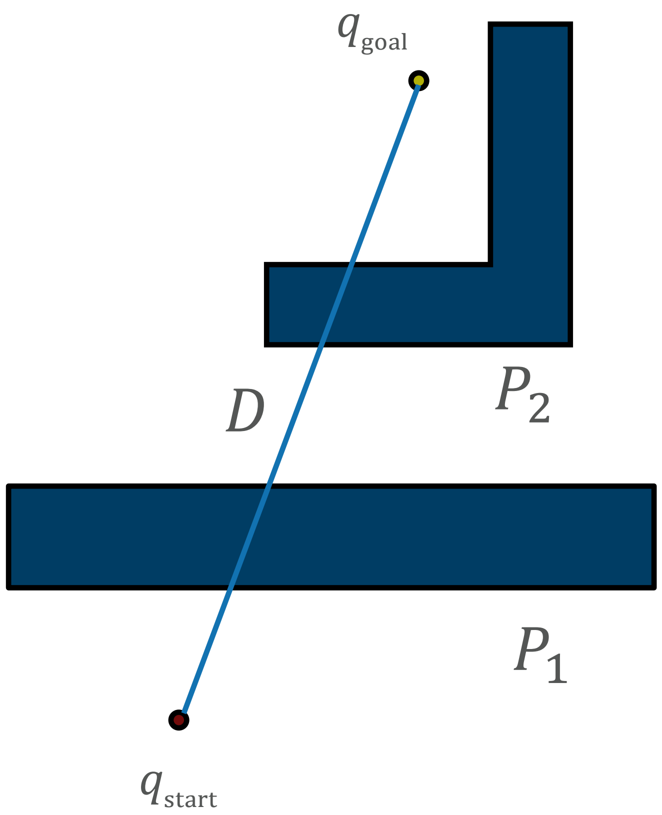
0%

$2.5(P_1 + P_2) + D$

0%

"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:

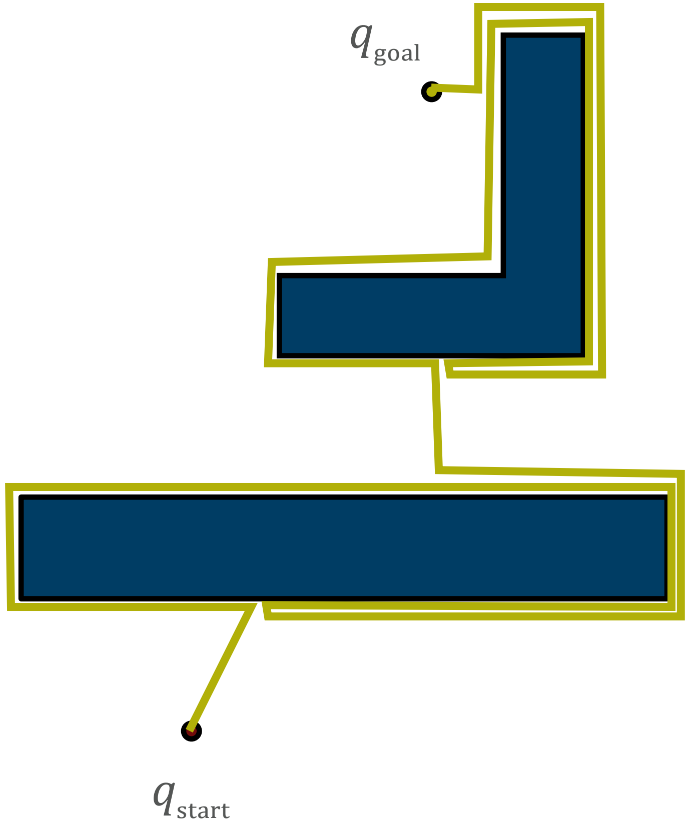
What's the longest distance it might travel?

$$D + \frac{3}{2} \sum P_i$$

Is "Bug 1" Complete?

- **Definition** (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
- **Prove Bug 1 is complete (Proof Sketch):**
  - Suppose BUG1 was incomplete, and there is a path from start to goal
  - BUG1 does not find it
    - Either it never terminates case1 , or, it spends an infinite amount of time
    - Suppose it never terminates
      - But each leave point is closer to the goal 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 point on boundary which contradicts the definition of a leave point.

Can we make something better than ''Bug1''?

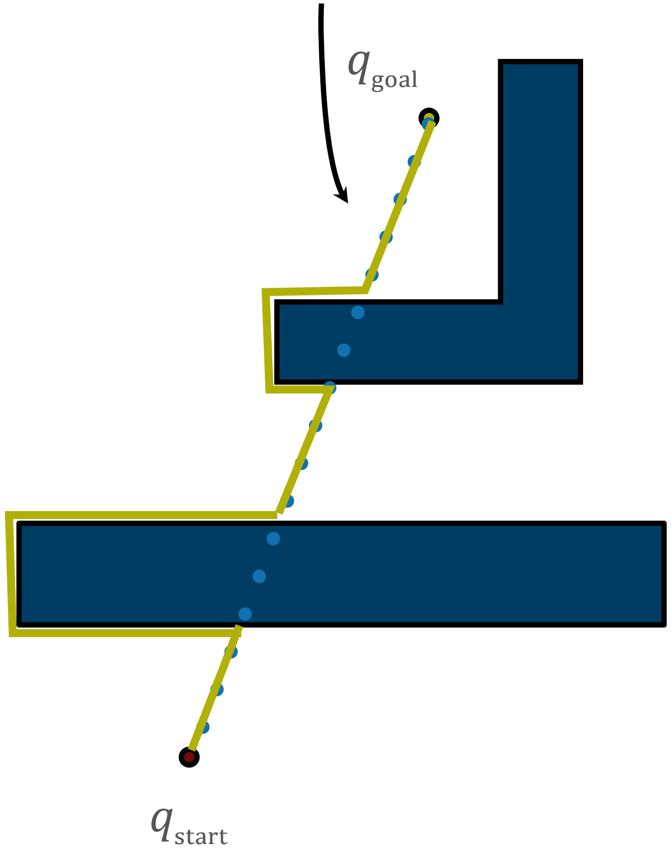


"Bug 1" Strategy:

1. Head toward goal
2. If an obstacle is encountered, circumnavigate it and remember how close you get to the goal
3. Then return to that closest point (by wall-following) and continue

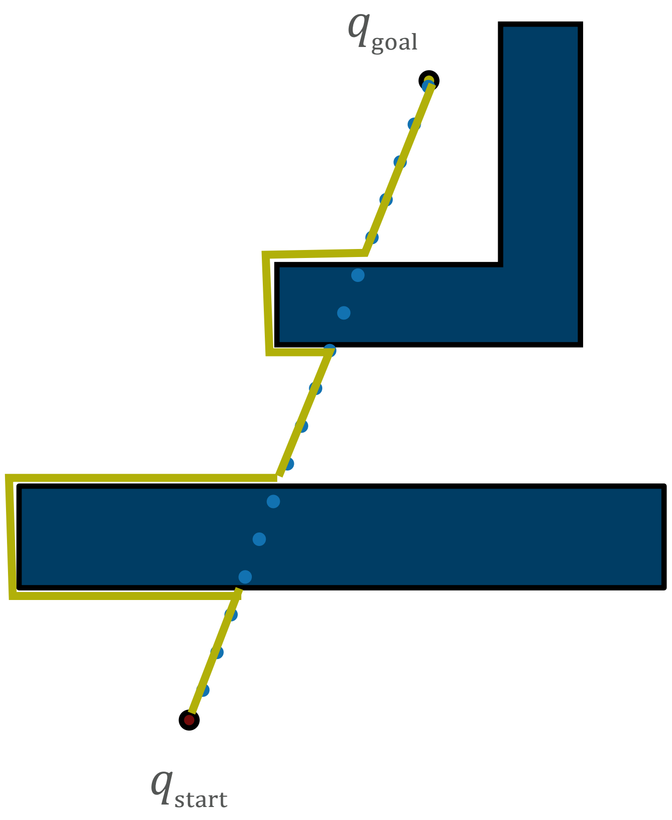
A better bug?

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



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

"Bug 2" Algorithm

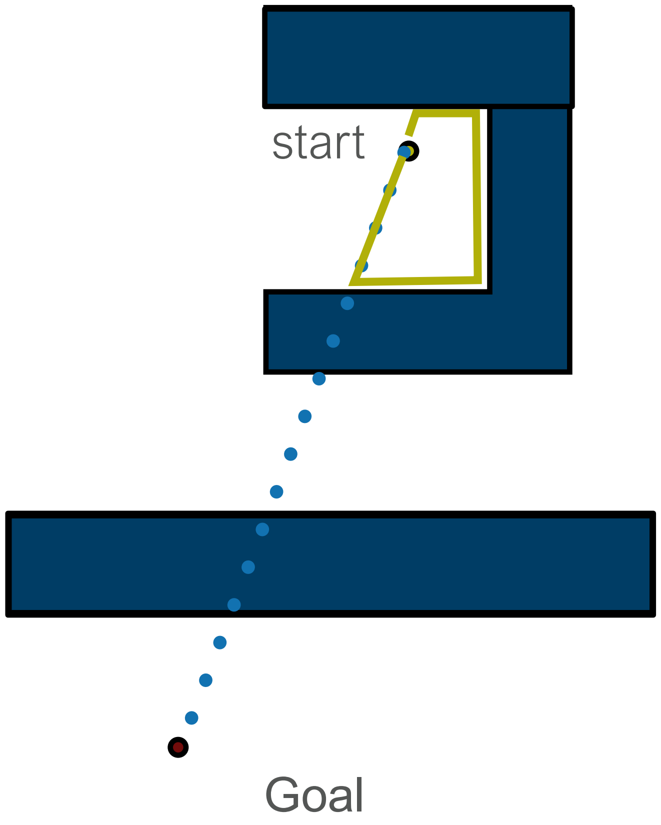


"Bug 2" Strategy:

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

Done?

"Bug 2" Algorithm



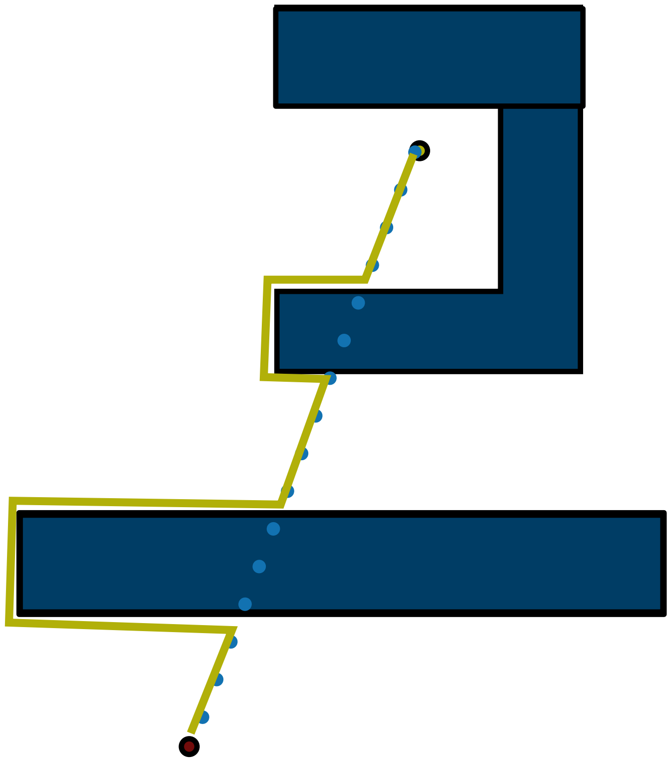
"Bug 2" Strategy:

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

Done?

NO! How do we fix this?

“Bug 2” Algorithm

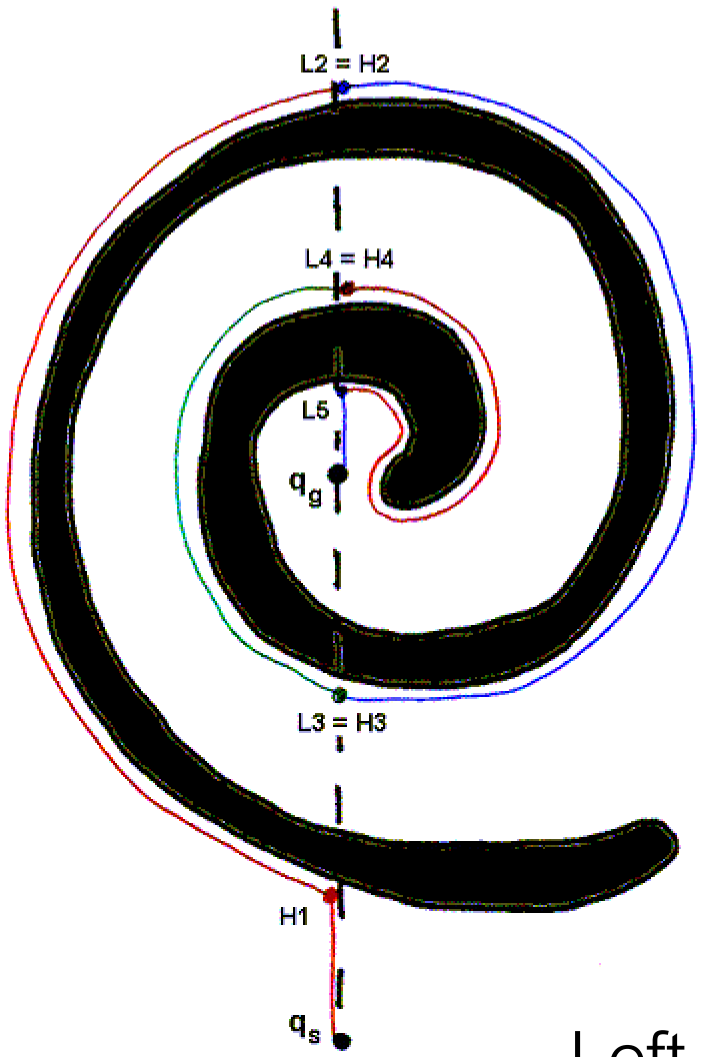


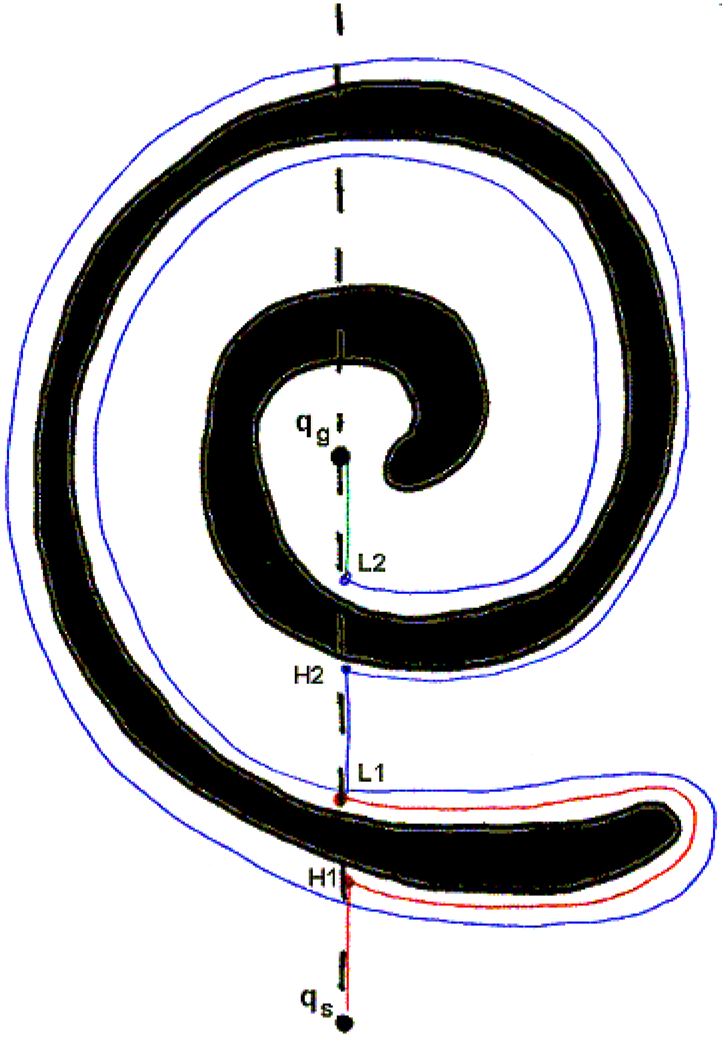
“Bug 2” Strategy:

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?

“Bug 2” Algorithm





Left turn VS Right turn

"Bug 2" Algorithm

Algorithm: Bug 2

Input: A point robot with a tactile sensor
Output: A path to the q_{goal} or a conclusion no such path exists

- Let $q^{L_0} = q_{\text{start}};\ i = 1$
- **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}, $d(x, q_{\text{goal}}) < d(q^{H_i}, q_{\text{goal}})$ and way to goal is unimpeded

  If goal is reached

  exit

  If q^{H_i} is reached

  exit with failure

  Else

  i=i+1

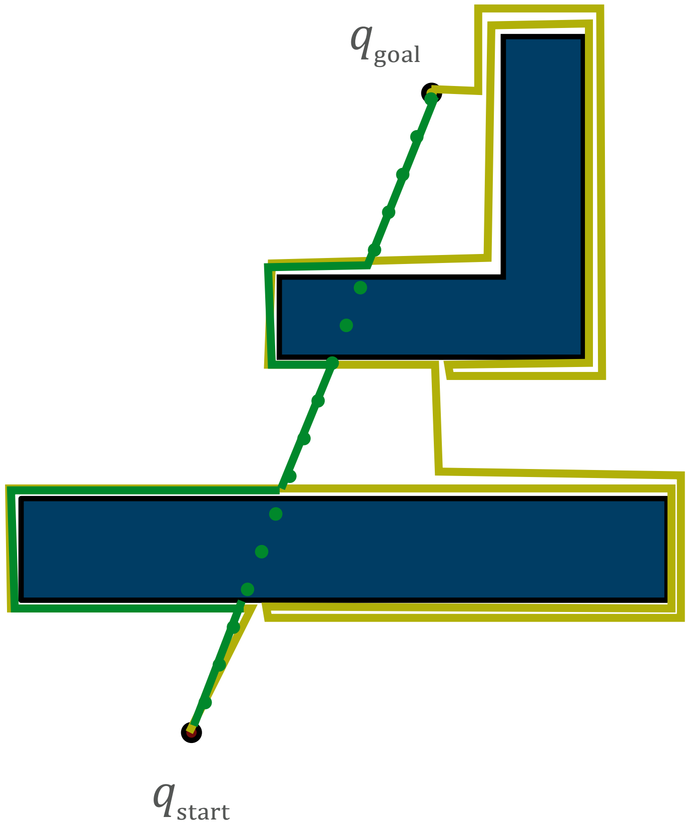
  continue

q^{H_i}: hit point q^{L_i}: leave point

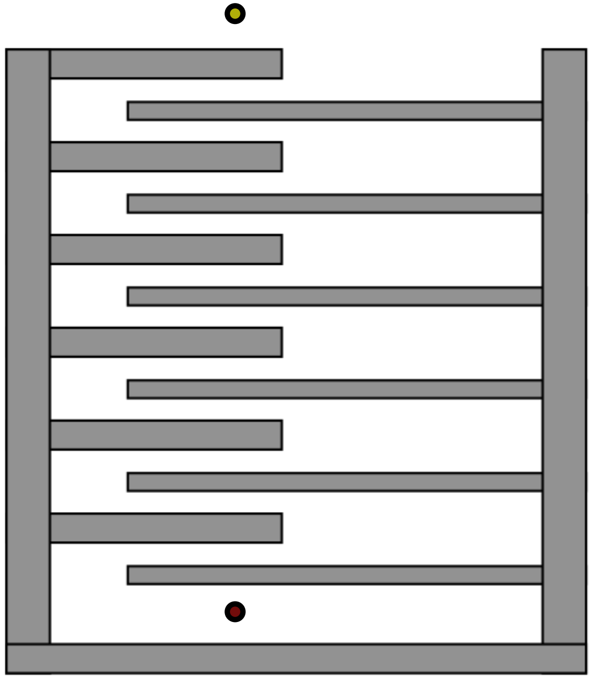
“Bug 1” vs “Bug 2”

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

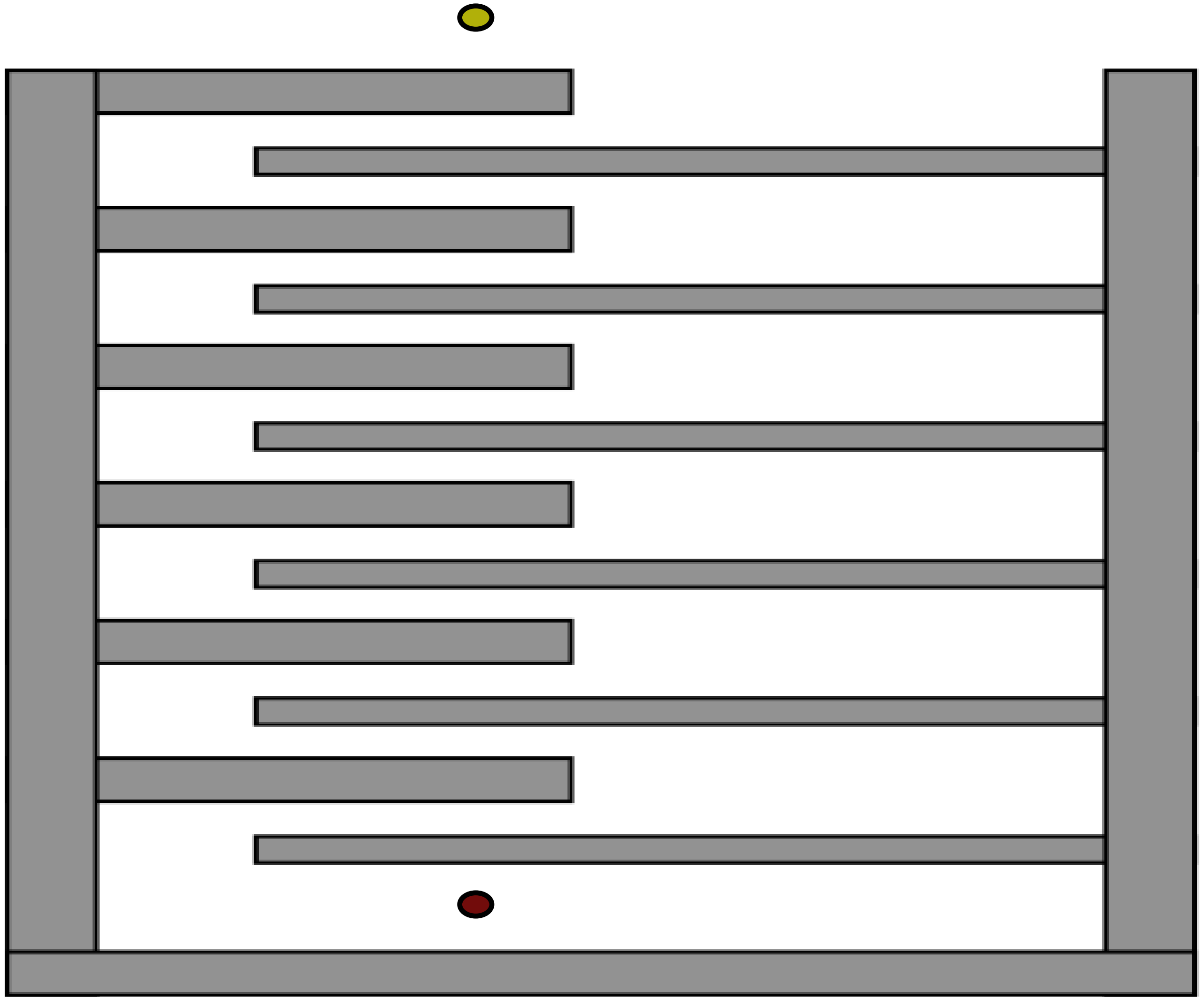
Bug 2 beats Bug 1



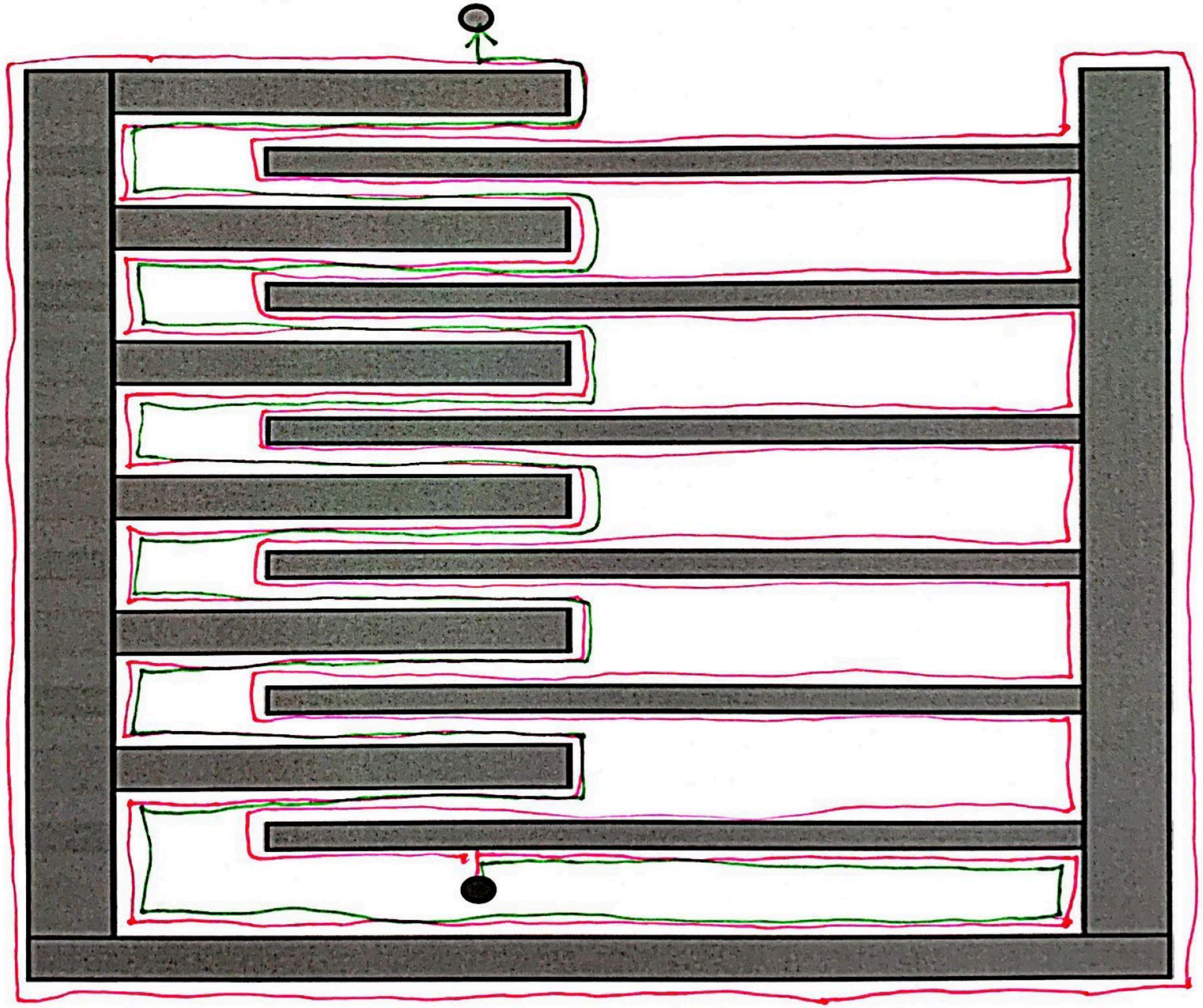
Bug 1 beats Bug 2



Bug1 Vs Bug 2



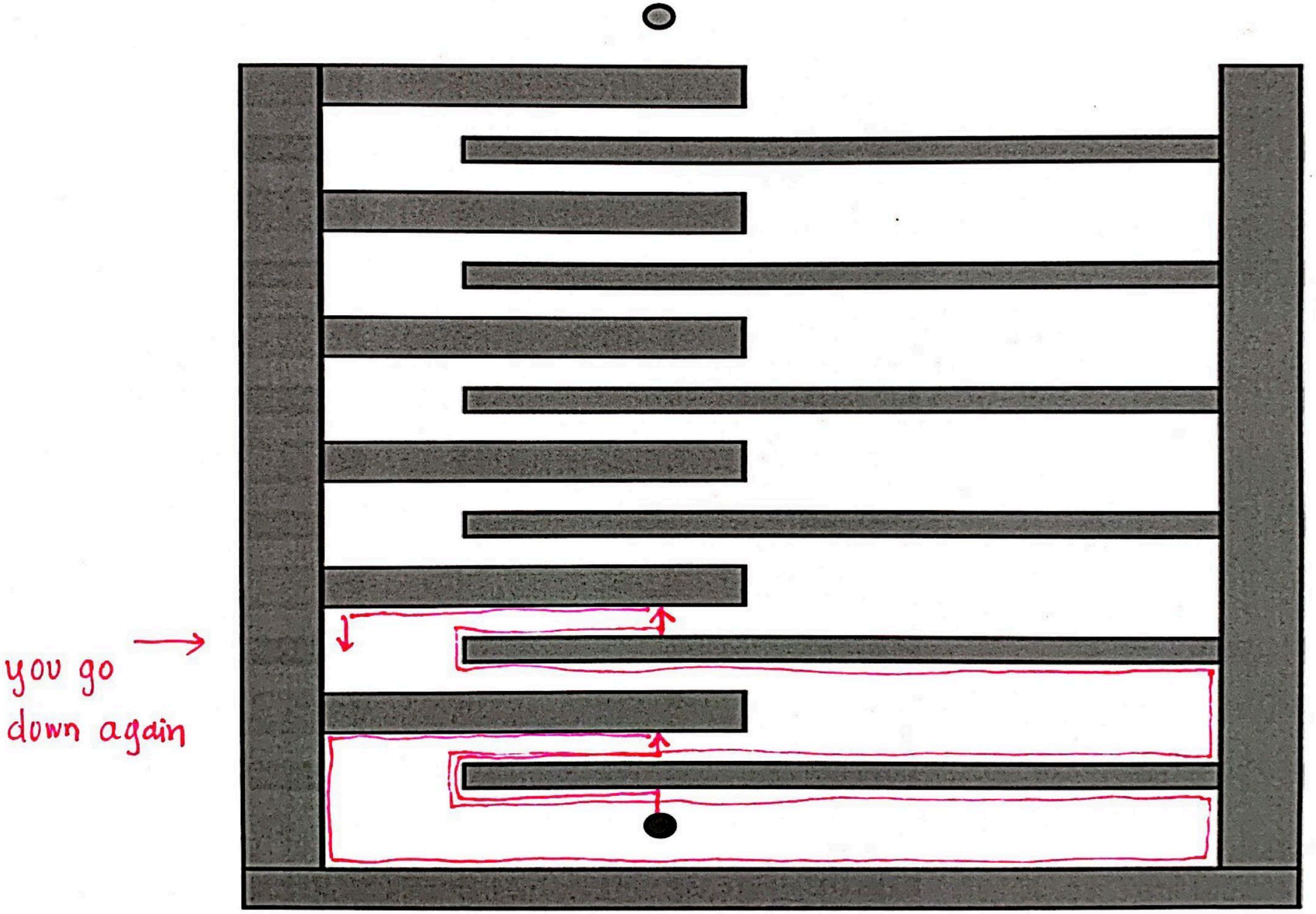
Bug 1



Red Line is the circumnavigation

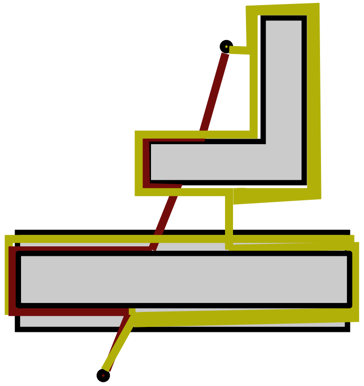
Green Line is going to the closest recorded point

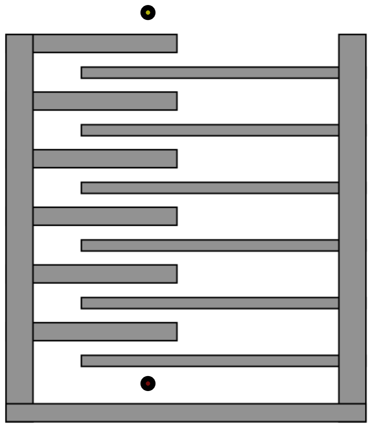
Bug2 – partial strategy



“Bug 1” VS “Bug 2”

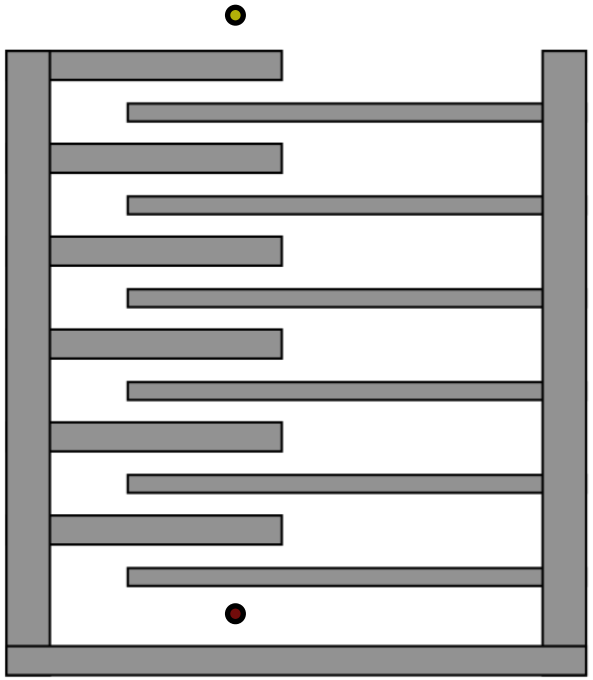
- “Bug 1” is an *exhaustive* search algorithm
  - it looks at *all* choices before committing
- “Bug 2” is a *greedy* algorithm
  - it takes the first thing that looks better
- In many cases, “Bug 2” will outperform “Bug 1”, but
- “Bug 1” has a more predictable performance overall





"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?

$$D$$

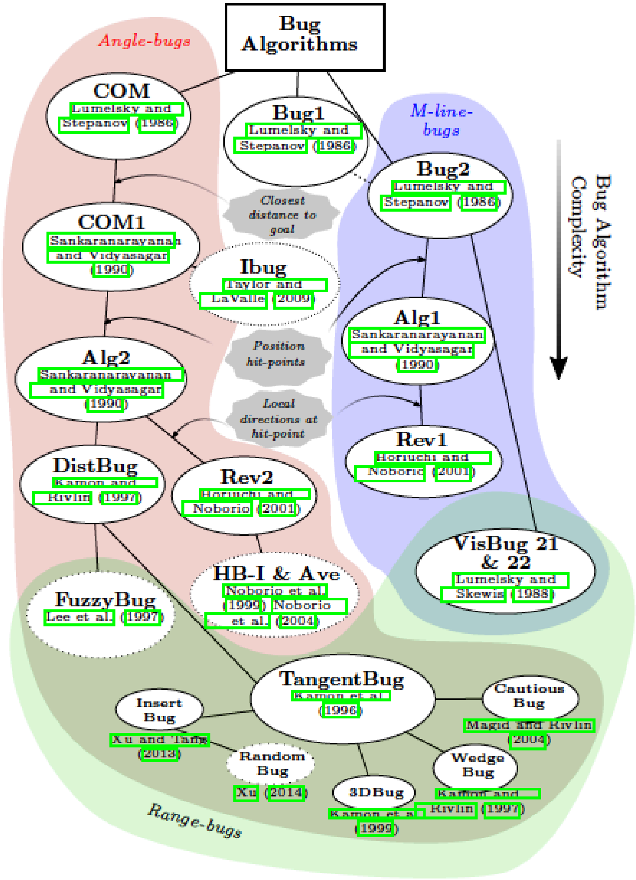
Upper bound:

What's the longest distance it might travel?

$$D + \frac{1}{2}\sum n_i P_i$$

n_i: # of m-line intersections with the i-th obstacle

There is a whole family of Bugs out there

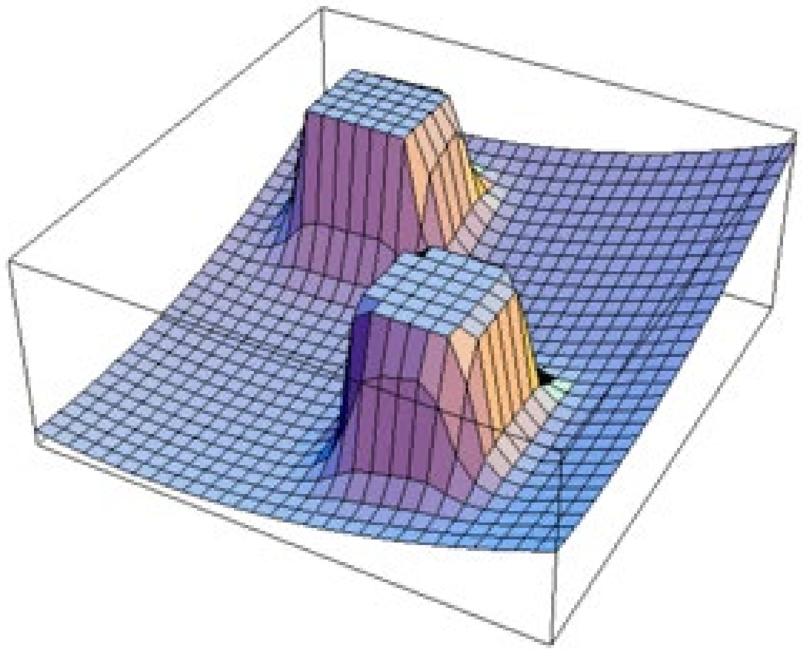


From McGuire at al, 2018

Summary

- Bug 1: safe and reliable
- Bug 2: better in some cases; worse in others
- Overall there are some issues:
    - Knowing exactly where the boundary is
    - Being able to follow it safely
    - Non optimal solutions
    - Applies only to simple robots

Potentials Fields

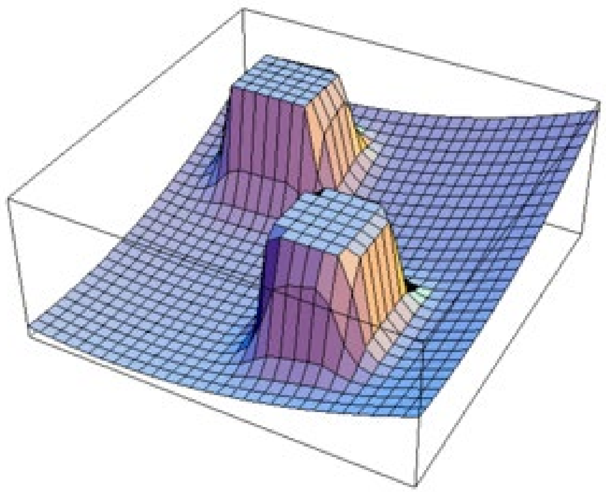


- General Idea:
  - Assume known obstacles
  - Treat robot as a rolling ball
  - Make mountains out of obstacles and a sink hole out of goal destination

Potential Function

- *Potential function* is a differentiable real-valued function

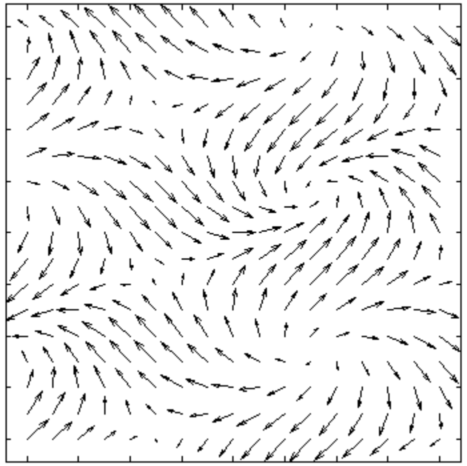
$$U: \mathbb{R}^m \rightarrow \mathbb{R}$$



  - Can be viewed as energy
  - *Gradient* is a vector

$$\nabla U(q) = DU(q)^T = \left[\frac{\partial U(q)}{\partial q_1}, \ldots, \frac{\partial U(q)}{\partial q_m}\right]^T$$

  which points in the direction that locally maximally increases U.

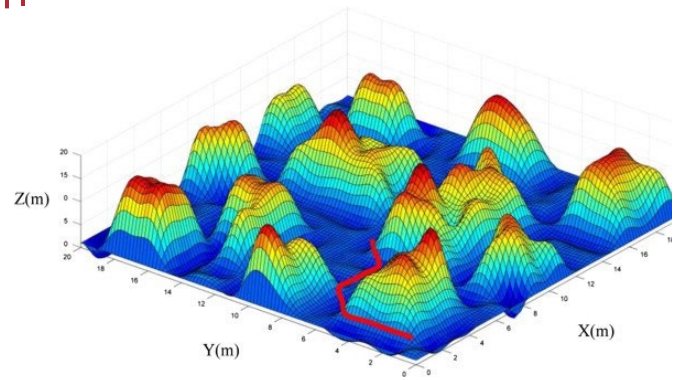


  - Can be viewed as force. Here, assume 1st order dynamics, i.e., $\nabla U(q)$ viewed as velocity.

  - Can be used to define vector field.

Potential Function

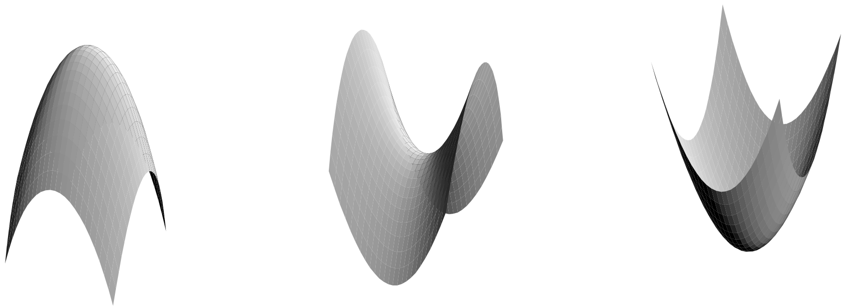
- Potential functions viewed as landscapes moving the robot from high-value state to low-value state



- Robot moves downhill by following the negated gradient of U:
  - i.e., *Gradient descent:* $\dot{c} = -\nabla U(c(t))$

- Robot terminates when it reaches at point q^* where $\nabla U(q^*) = 0$

- Terminating point q^*, called critical point
  - Can be maximum, minimum, or saddle point



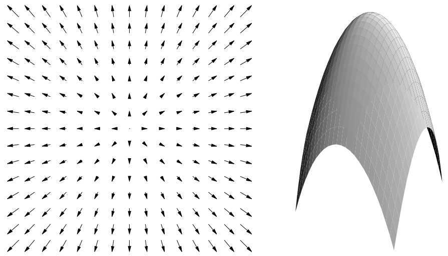
maximum saddle minimum

Potential Function

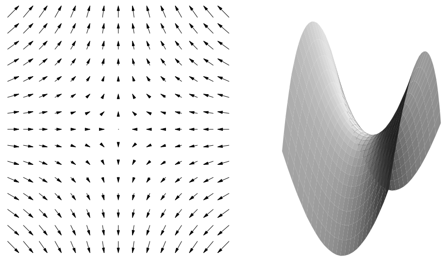
- Second derivative determines the type of critical point
- *Hessian* matrix

$$H = \begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \cdots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_1 \partial q_n} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix}$$

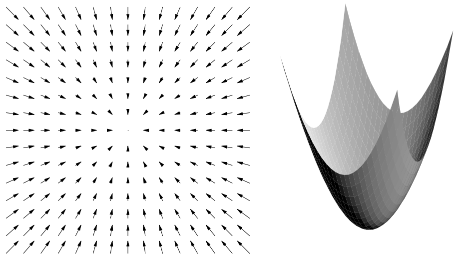
- When $H(q^*)$ is non-singular, q^* is isolated
    - When $H(q^*)$ is positive definite: q^* local minimum
    - When $H(q^*)$ is negative definite: q^* local maximum
- In gradient descent,
    - the robot never terminates in maximum or saddle points because, with a perturbation, it is freed.
    - Local minimum is a problem



maximum



saddle



minimum

Attractive Repulsive Field

- Simplest potential function in Q_{free}

$$U(q) = U_{att}(q) + U_{rep}(q)$$

U_{att} attracts the robot (goal) and U_{rep} repels the robot (obstacles)

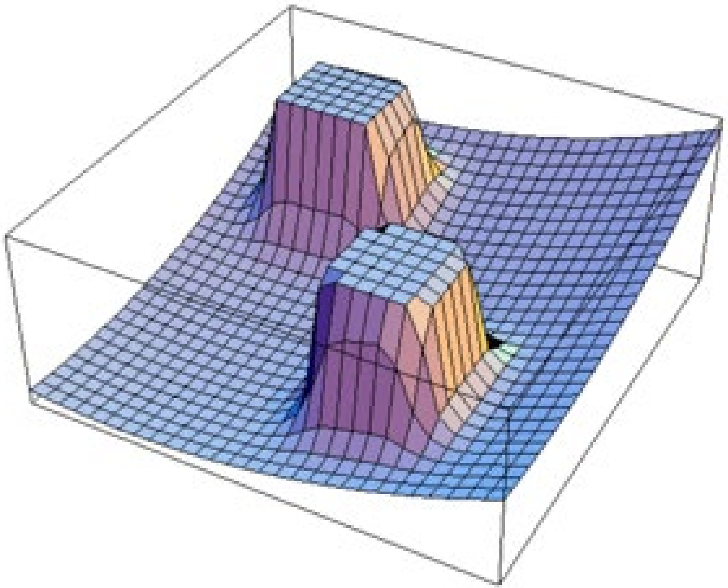
- Designing U_{att}
  - should be monotonically increasing with distance from q_{goal}
  - Simplest: conic potential

$$U_{att}(q) = \xi \, d(q, q_{\text{goal}})$$

parameter to scale the
effect of attractive force

Repulsive Field

- Repulsive potential function U_{rep} keeps the robot from an obstacle
- Strength of U_{rep} depends on robot's proximity to an obstacle
  - The closer to an obstacle, the stronger the repulsive force
  - Define U_{rep} in terms of distance to the closest obstacle: $D(q)$



Gradient Descent Algorithms

- Gradient descent is well-known approach to optimization problems
- Idea:
  - Start at the initial configuration
  - Small step in the direction opposite to the gradient
  - Now at a new (start) configuration
  - Repeat until gradient is zero

Gradient Descent Algorithm

- Potential function $U = U_{att} + U_{rep}$:

Algorithm Gradient Descent

Input: A means to compute the gradient $\nabla U(q)$ at a point q

Output: A sequence of points $\{q(0), q(1), \ldots, q(i)\}$

$q(0) = q_{\text{start}}$

$i = 0$

while $\nabla U(q(i)) \neq 0$ **do**

  $q(i+1) = q(i) + \alpha(i)\nabla U(q(i))$

  $i = i + 1$

end while

  - $q(i)$: value of q at iteration i
  - $\alpha(i)$: step size at the i iteration
    - Needs to be small enough not to allow "jump into obstacles"
    - Needs to be large enough not to require excessive computation time
    - The value is typically chosen ad hoc or empirically
      - e.g., based on distance to the nearest obstacle or to the goal

Gradient Descent Algorithm

- Potential function $U = U_{att} + U_{rep}$:

Algorithm Gradient Descent

Input: A means to compute the gradient $\nabla U(q)$ at a point q

Output: A sequence of points $\{q(0), q(1), \ldots, q(i)\}$

$q(0) = q_{\text{start}}$
$i = 0$
while $\nabla U(q(i)) \neq 0$ **do**
  $q(i+1) = q(i) + \alpha(i)\nabla U(q(i))$
  $i = i + 1$
end while

- Highly unlikely to exactly satisfy $\nabla U\big(q(i)\big) = 0$
- More realistic condition:

$$\left\|\nabla U\big(q(i)\big)\right\| \leq \epsilon$$

Computing Distance

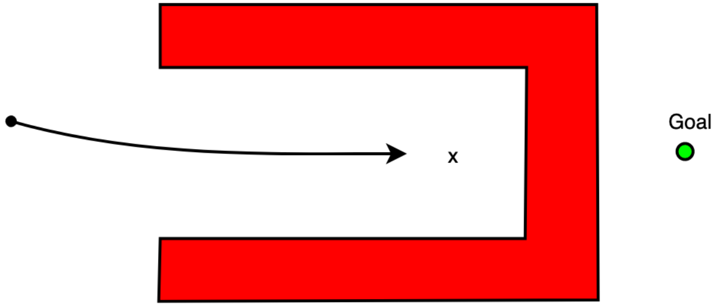
- Potential function $U = U_{att} + U_{rep}$
  - Computing distance in U_{att} is simple because distance to a point
  - Computing distance in U_{rep} challenging because distance to obstacle

- For U_{rep}, can use various distance definitions, e.g.,
  - Range sensor distance
  - Discrete distance through discretization of continuous space

Can we solve all problems now?

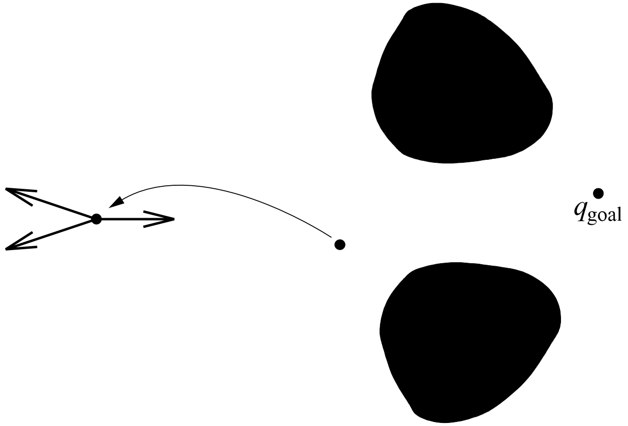
Local Minima Problem

- Problem with all gradient descent algorithms:
    - Getting stuck in local minima



- Gradient descent is guaranteed to to converge to a minimum in the field
    - No guarantee gradient descent will find a path to q_{goal}

- Gradient vanishes when the sum of the attractive gradient and the repulsive gradient is zero



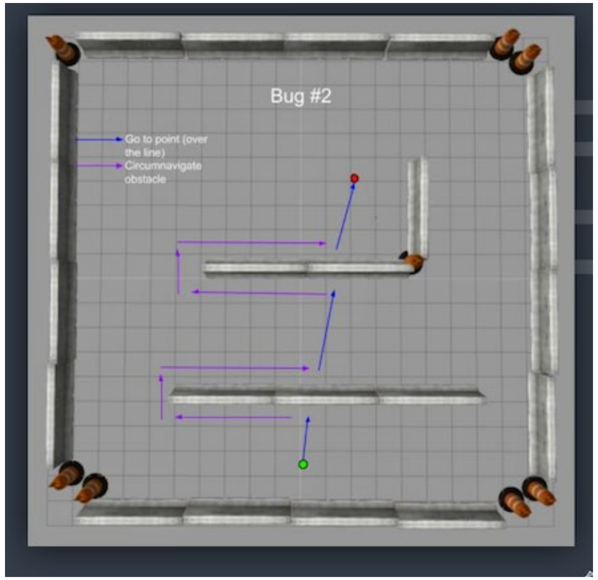
Potential Fields Summary

- Potential functions + gradient descent
  - Need careful design of these functions
    - Simplest form: $U = U_{att} + U_{rep}$
  - Use gradient descent for motion planning
  - **Challenge:** computing distances
  - **Problem:** getting stuck in local minima

Bug Algorithms Potentials Fields Summary

Bug Algorithms

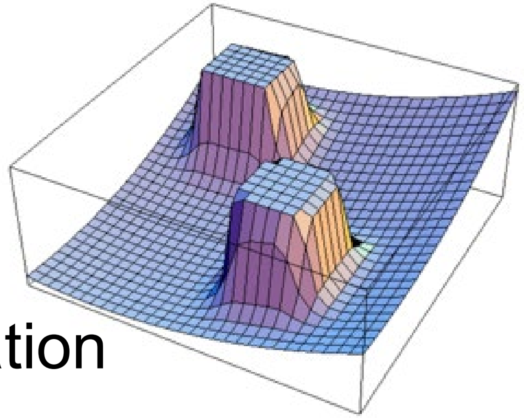
Pros:
- Simple to implement
- Complete
- Continuous



Cons:
- 2D robots
- Non Optimal Motions

Potential Fields

Pros:
- Beyond 2D
- Smooth Motions
- Continuous
- Efficient Computation



Cons:
- Stuck in Local Minima
- Not-Complete
- Non-Optimal Motions
- Challenging distance calculation