



UNIVERSITY OF
SOUTH CAROLINA

CSCE 574 ROBOTICS

Path Planning



Ioannis Rekle

Outline

- Path Planning
 - Visibility Graph
 - Potential Fields
 - Bug Algorithms
 - Skeletons/Voronoi Graphs
 - C-Space

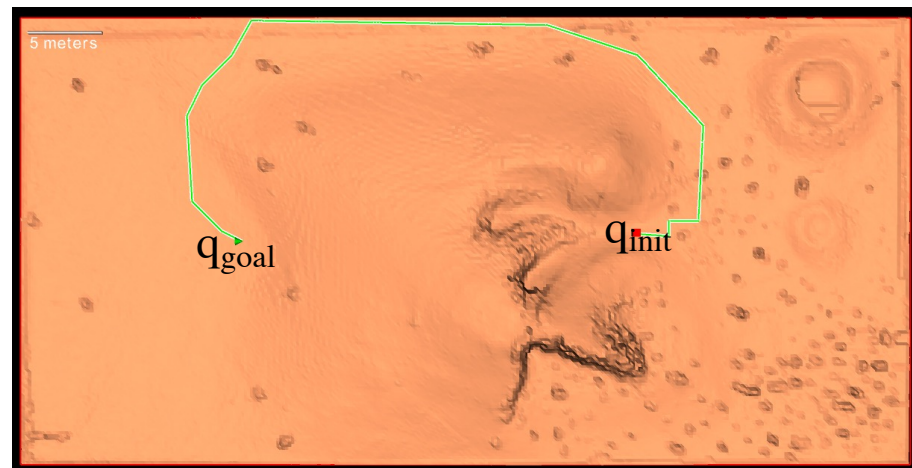
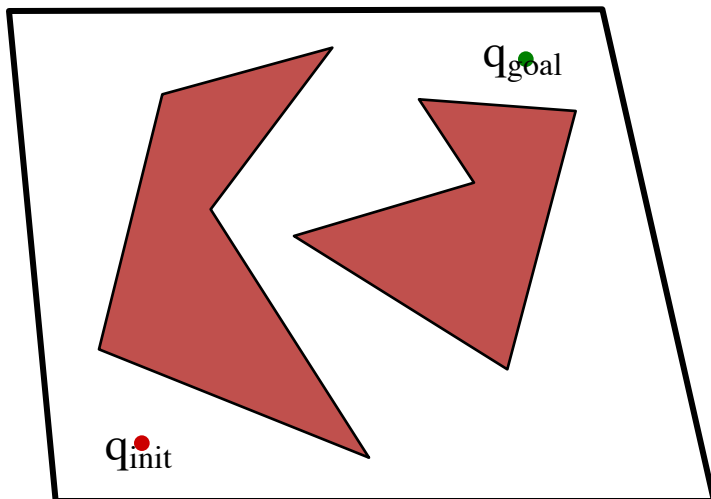


Motion Planning

- The ability to go from **A** to **B**
 - Known map – Off-line planning
 - Unknown Environment – Online planning
 - Static/Dynamic Environment

• q_{init}

• q_{goal}



Path Planning

World

Robot

Map



Path Planning

World

- Indoor/Outdoor
- 2D/2.5D/3D
- Static/Dynamic
- Known/Unknown
- Abstract (web)

Robot

Map



Path Planning

World

Robot

- Mobile
 - Indoor/Outdoor
 - Walking/Flying/Swimming
- Manipulator
- Humanoid
- Abstract

Map



Path Planning

World

Robot

Map

- Topological
- Metric
- Feature Based
- 1D, 2D, 2.5D, 3D



Path Planning

World

- Indoor/Outdoor
- 2D/2.5D/3D
- Static/Dynamic
- Known/Unknown
- Abstract (web)

Robot

- Mobile
 - Indoor/Outdoor
 - Walking/Flying/Swimming
- Manipulator
- Humanoid
- Abstract

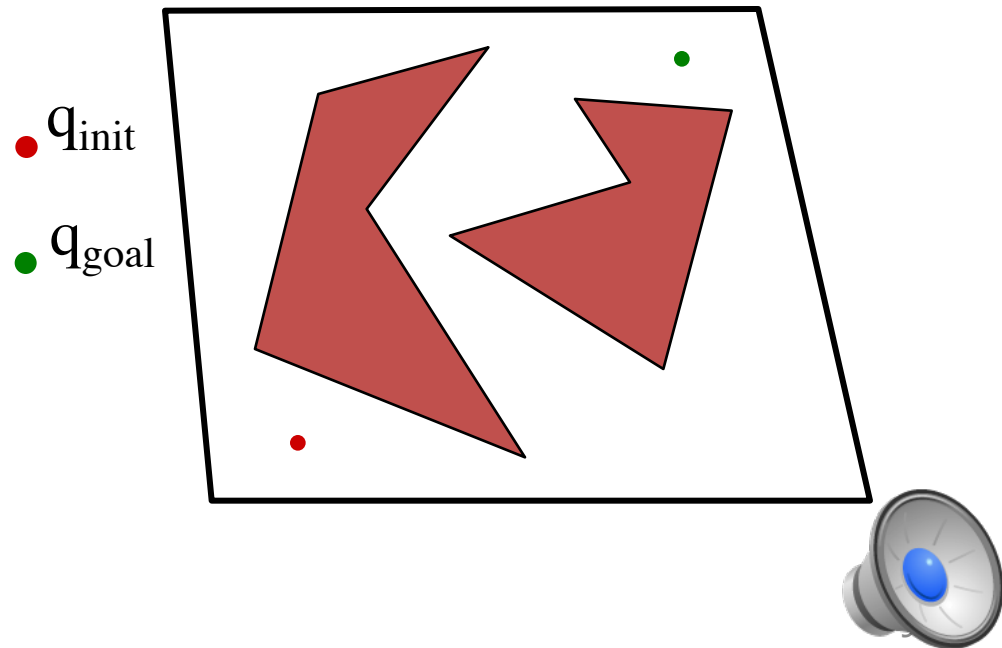
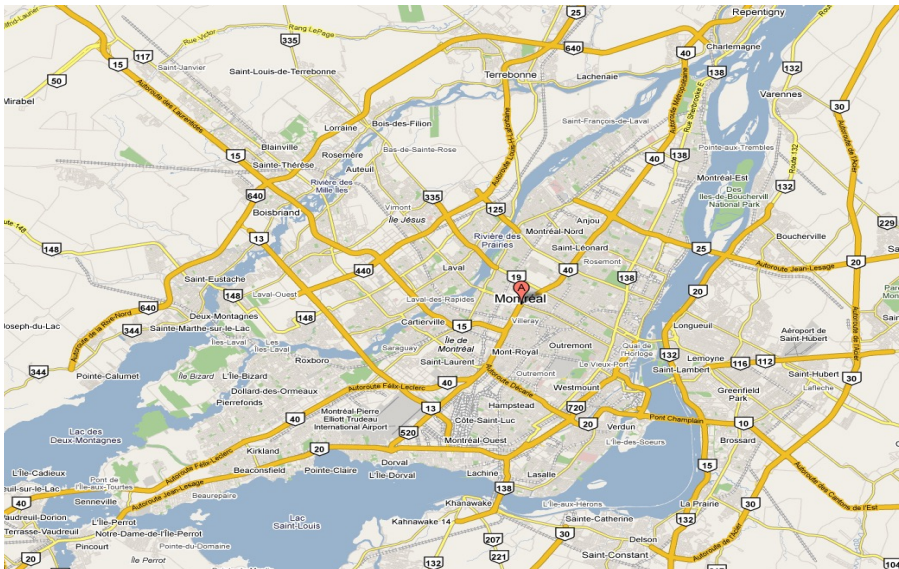
Map

- Topological
- Metric
- Feature Based
- 1D,2D,2.5D,3D



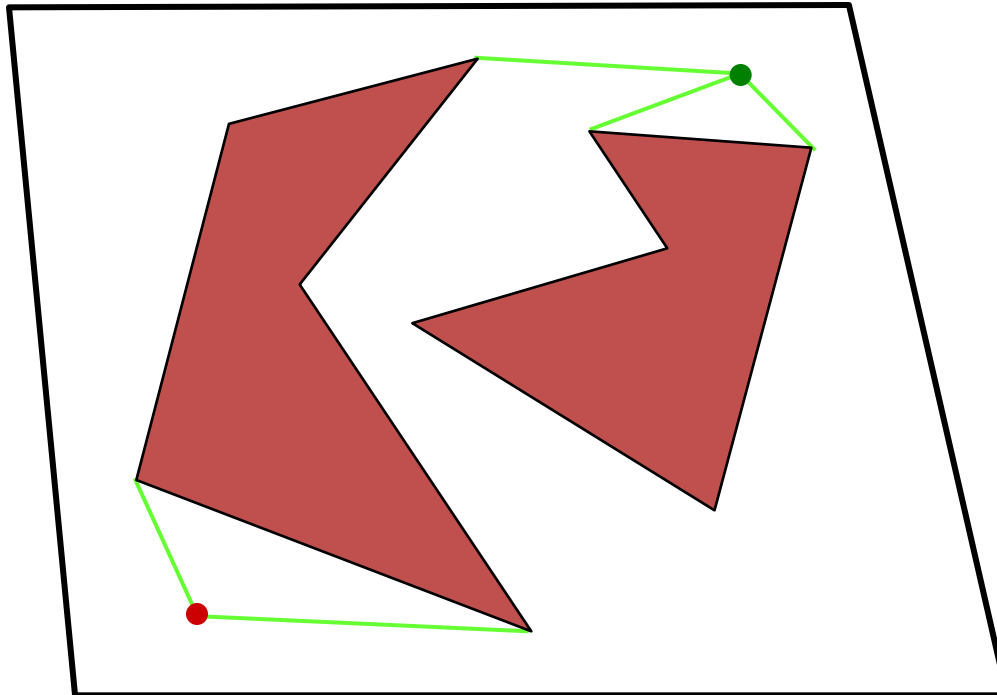
Path Planning: Assumptions

- Known Map
- Roadmaps (Graph representations)
- Polygonal Representation



Visibility Graph

- Connect Initial and goal locations with all the visible vertices



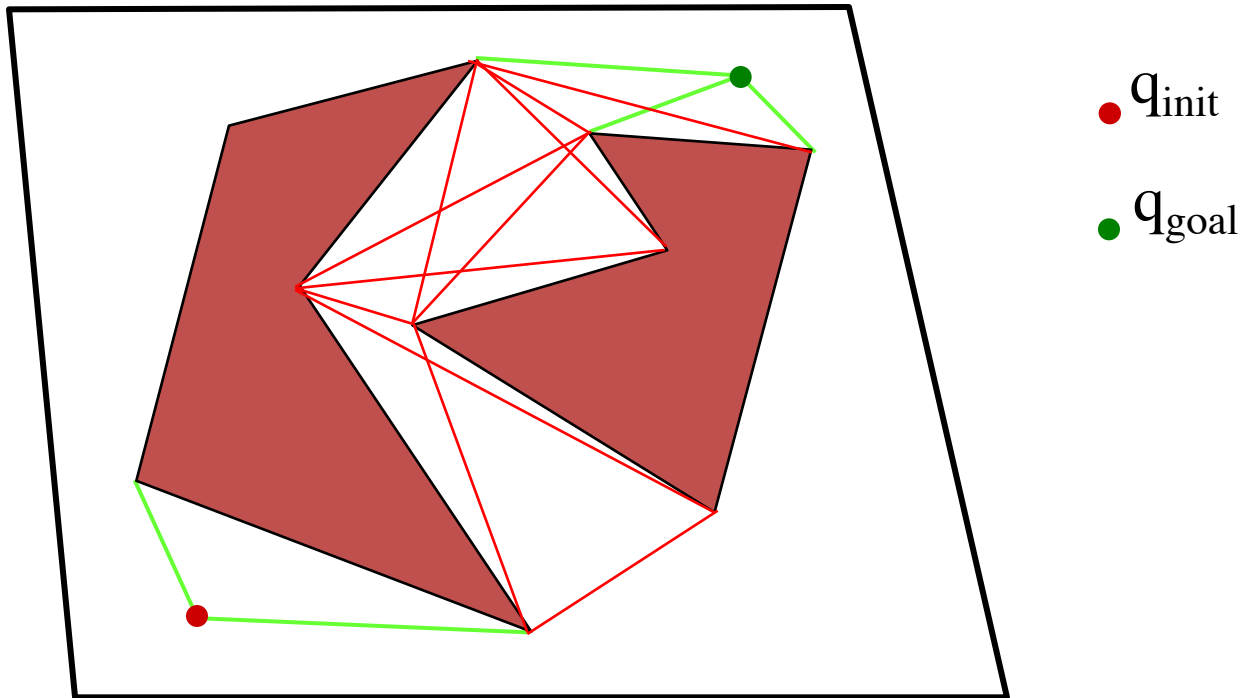
● q_{init}

● q_{goal}



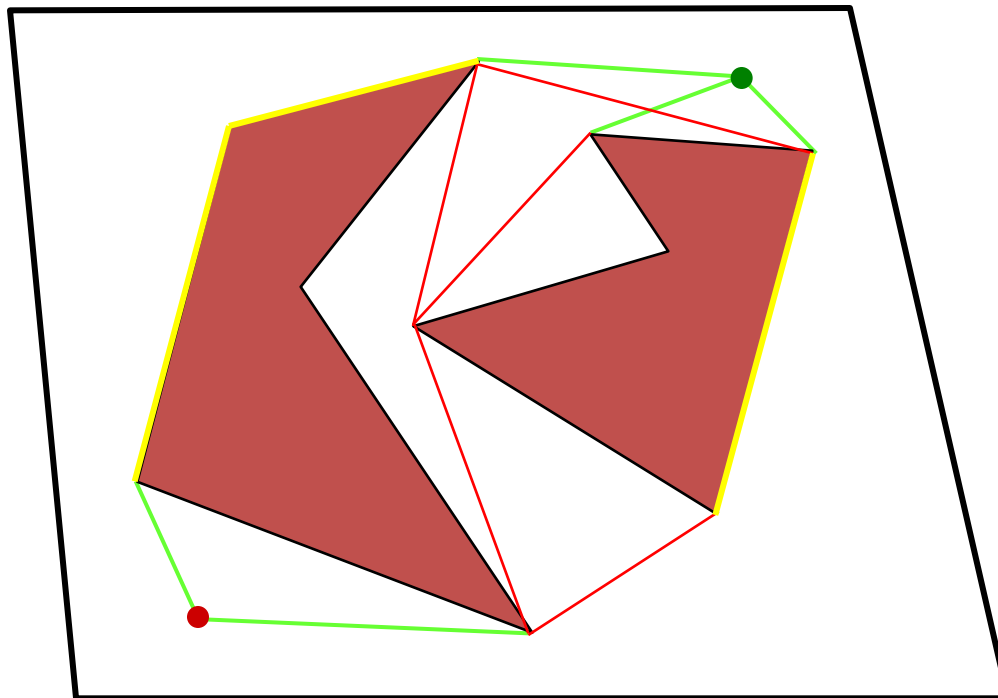
Visibility Graph

- Connect initial and goal locations with all the visible vertices
- Connect each obstacle vertex to every visible obstacle vertex



Visibility Graph

- Connect initial and goal locations with all the visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle



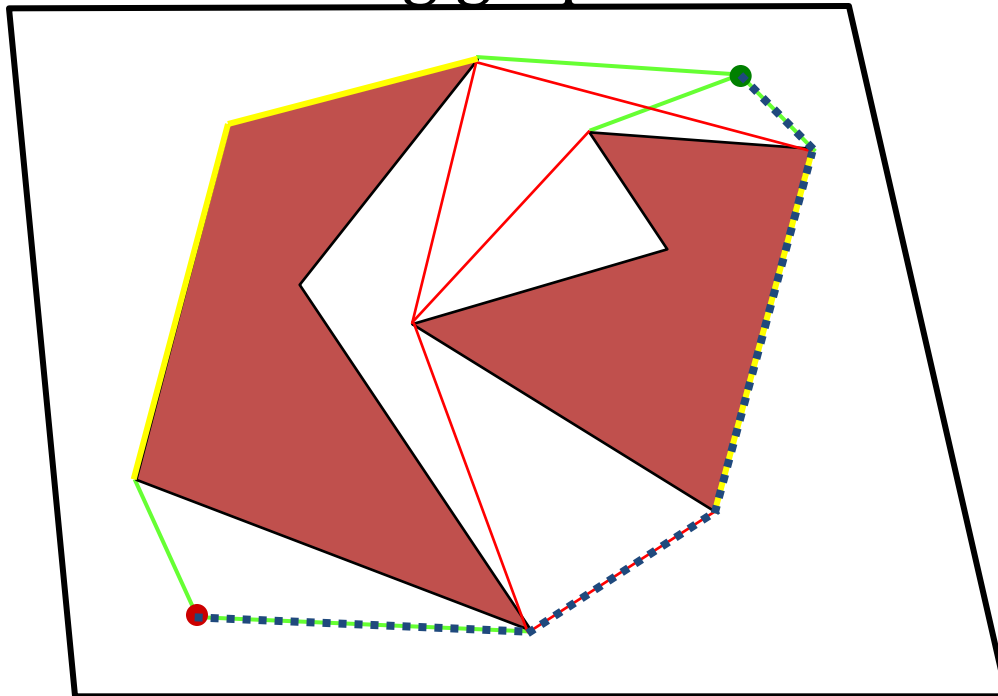
● q_{init}

● q_{goal}



Visibility Graph

- Connect initial and goal locations with all the visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



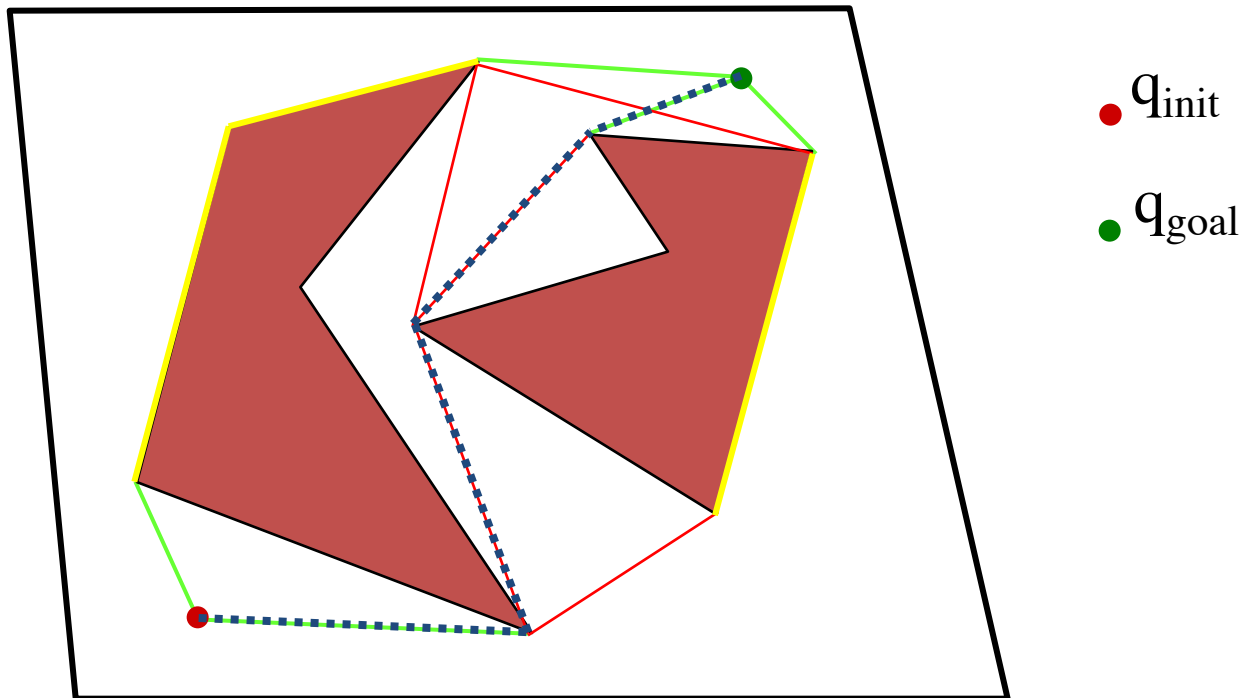
● q_{init}

● q_{goal}



Visibility Graph

- An alternative path
- Alternative name: “Rubber band algorithm”



Major Fault

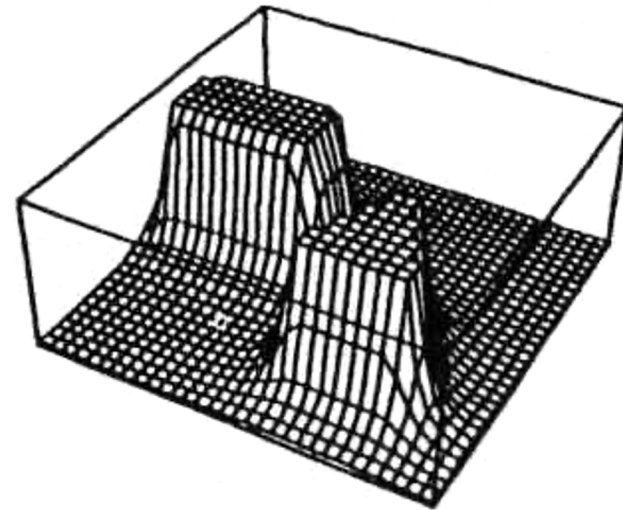
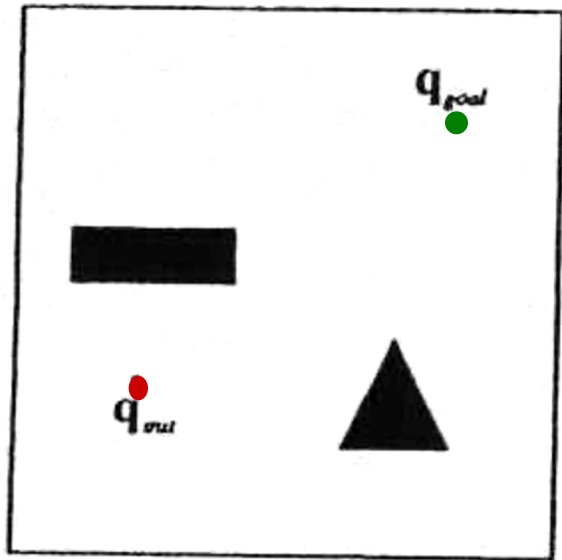
- Point robot
- Path planning like that guarantees to hit the obstacles



Path Planning

Potential Field methods

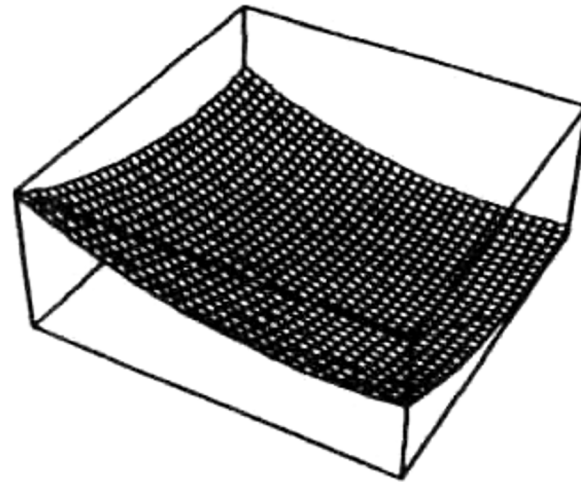
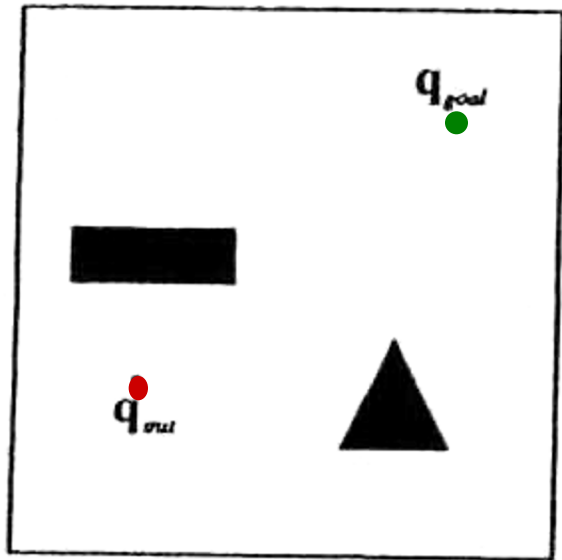
- compute a repulsive force away from obstacles



Local techniques

Potential Field methods

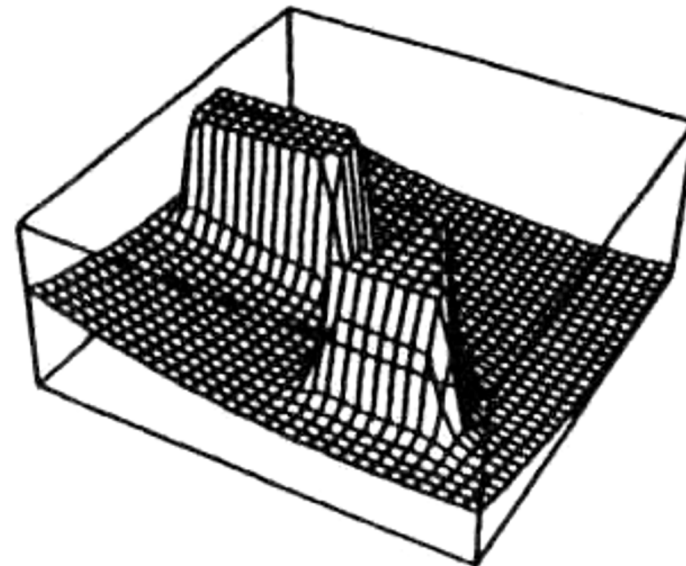
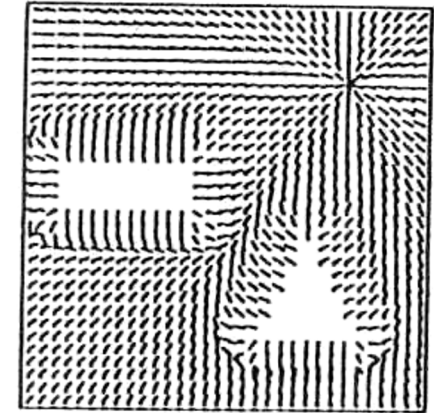
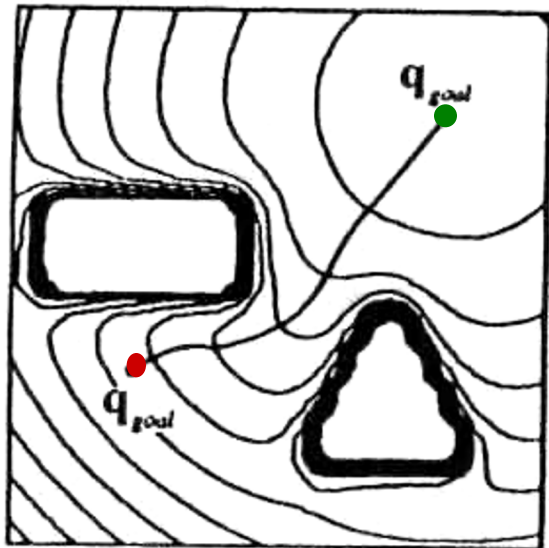
- compute a repulsive force away from obstacles
- compute an attractive force toward the goal



Local techniques

Potential Field methods

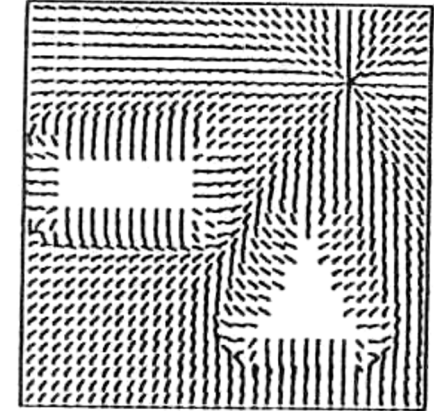
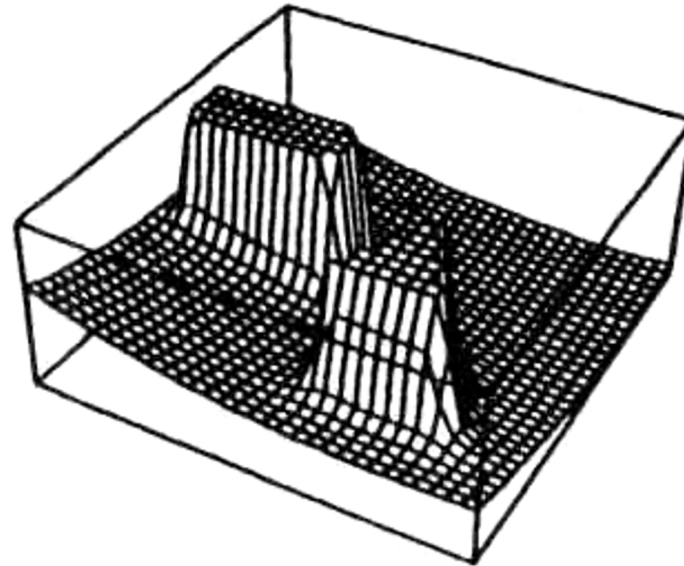
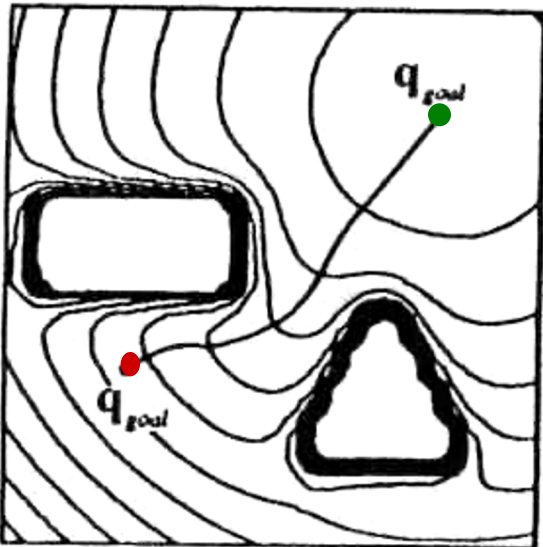
- compute a repulsive force away from obstacles
 - compute an attractive force toward the goal
- let the sum of the forces control the robot



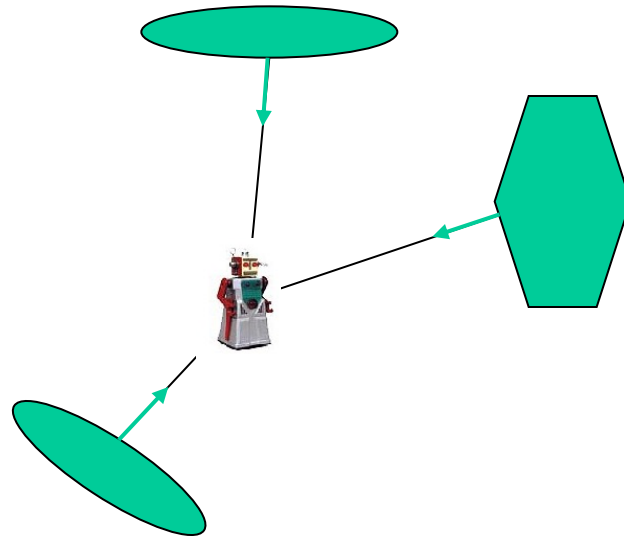
Local techniques

Potential Field methods

- compute a repulsive force away from obstacles
 - compute an attractive force toward the goal
- let the sum of the forces control the robot



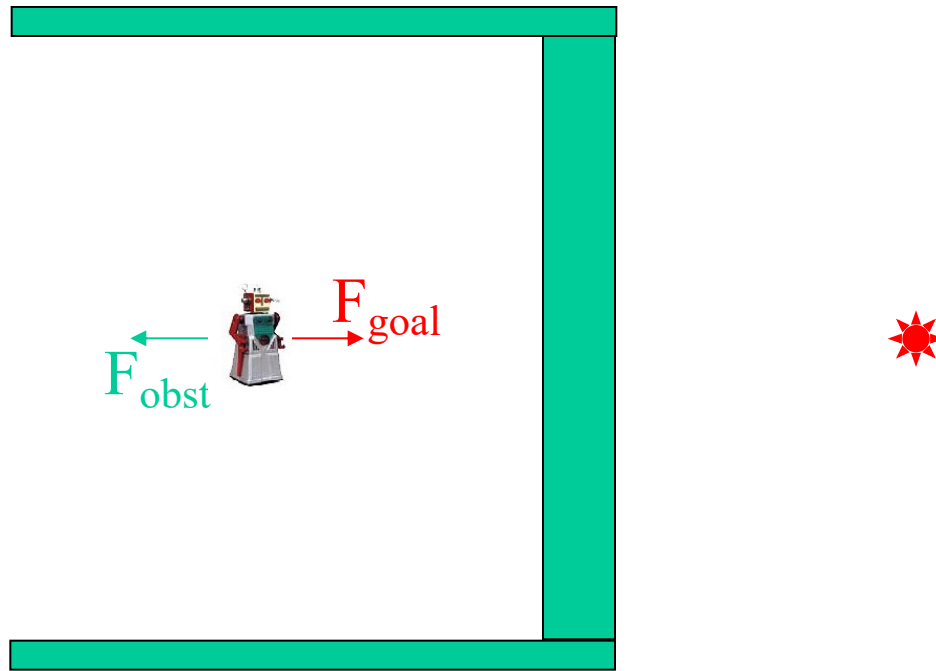
Sensor Based Calculations



Major Problem?



Local Minima!



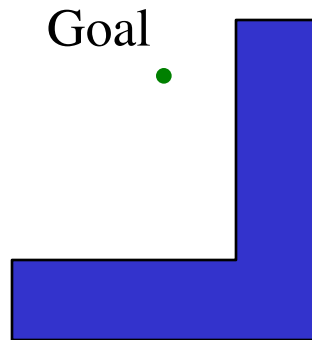
Simulated Annealing

- Every so often add some random force



Limited-knowledge path planning

- Path planning with limited knowledge
 - Insect-inspired “bug” algorithms



- known direction to goal
- otherwise local sensing
 - walls/obstacles encoders
- “reasonable” world
 1. finitely many obstacles in any finite disc
 2. a line will intersect an obstacle finitely many times

• Start



Not truly modeling bugs...

Insects do use several cues for navigation:



they're not ears...

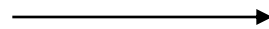
- visual landmarks
- polarized light
- chemical sensing



neither are the current bug-sized robots

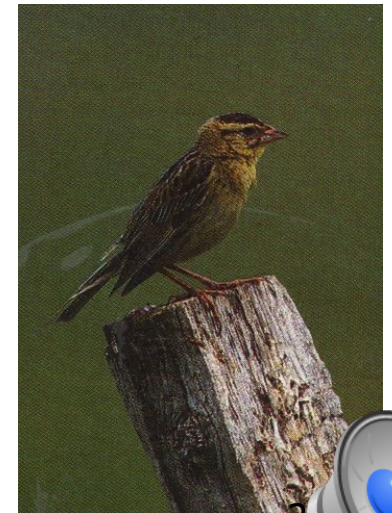
Other animals use information from

- magnetic fields
- electric currents
- temperature



CSCE-574 Robotics

bacteria



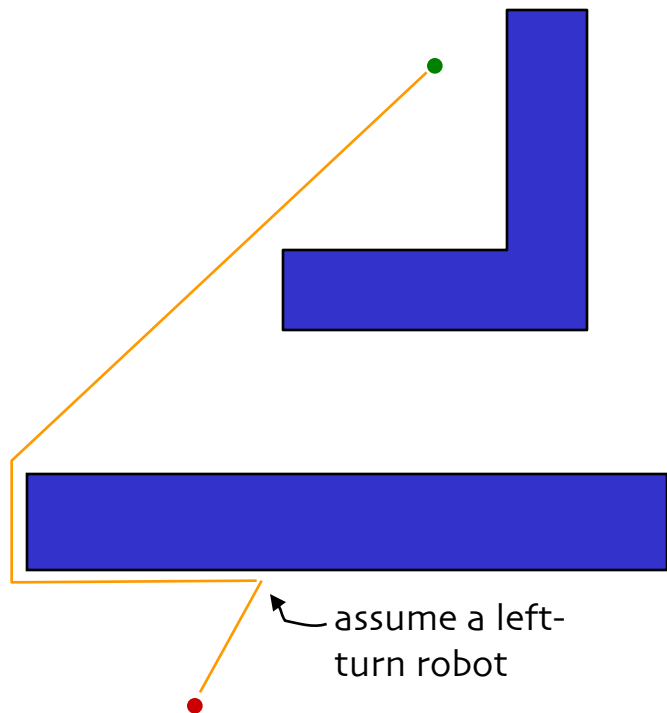
migrating bobolink



Bug Strategy

Insect-inspired “bug” algorithms

- known direction to goal
- otherwise only local sensing
walls/obstacles encoders



“Bug 0” algorithm

- 1) head toward goal
- 2) follow obstacles until you can head toward the goal again
- 3) continue



Does It Work?



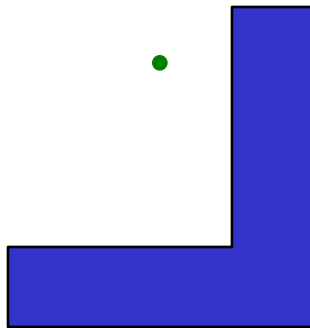
Bug 1

Insect-inspired “bug” algorithms

- known direction to goal
- otherwise only local sensing walls/obstacles encoders

“Bug 1” algorithm

1) head toward goal



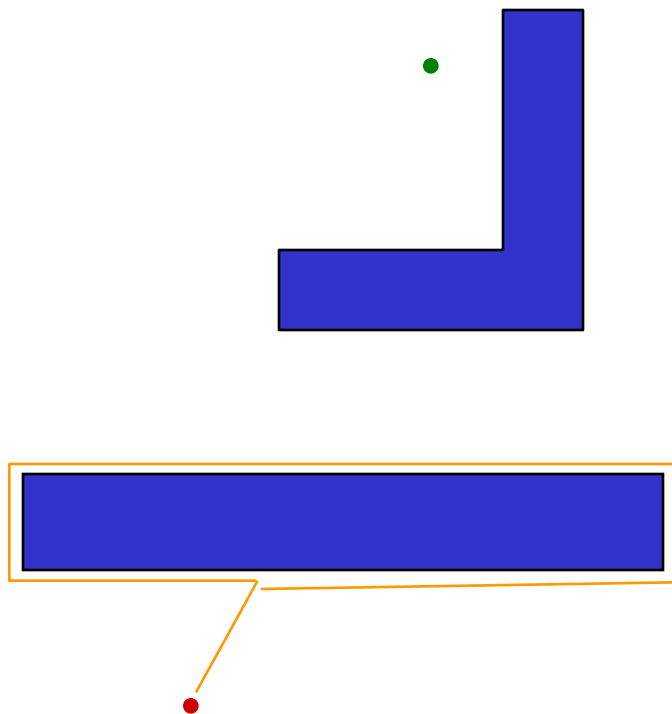
Bug 1

Insect-inspired “bug” algorithms

- known direction to goal
- otherwise only 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



Bug 1 analysis

Distance Traveled

What are bounds on the path length that the robot takes?

Available Information:

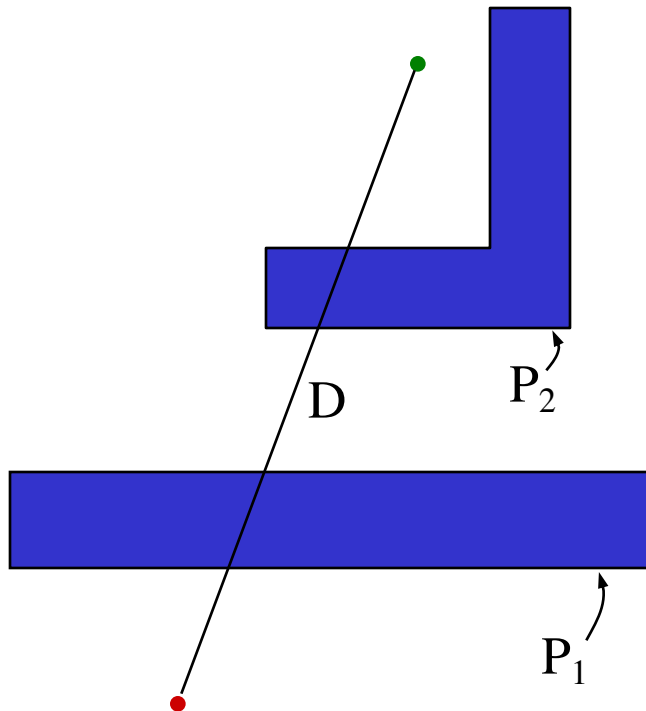
D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

Lower and upper bounds?

Lower bound:

Upper bound:



Bug 1 analysis

Distance Traveled

What are bounds on the path length that the robot takes?

Available Information:

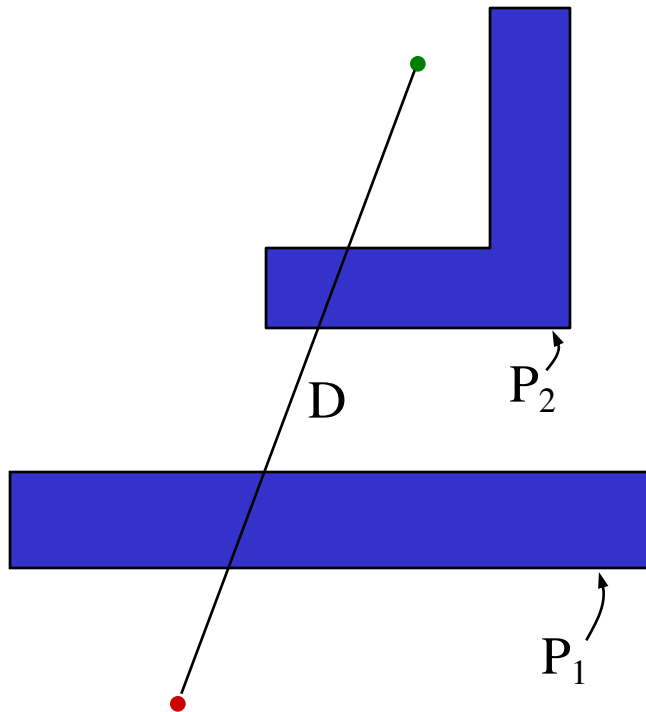
D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

Lower and upper bounds?

Lower bound: D

Upper bound:



Bug 1 analysis

Distance Traveled

What are bounds on the path length that the robot takes?

Available Information:

D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

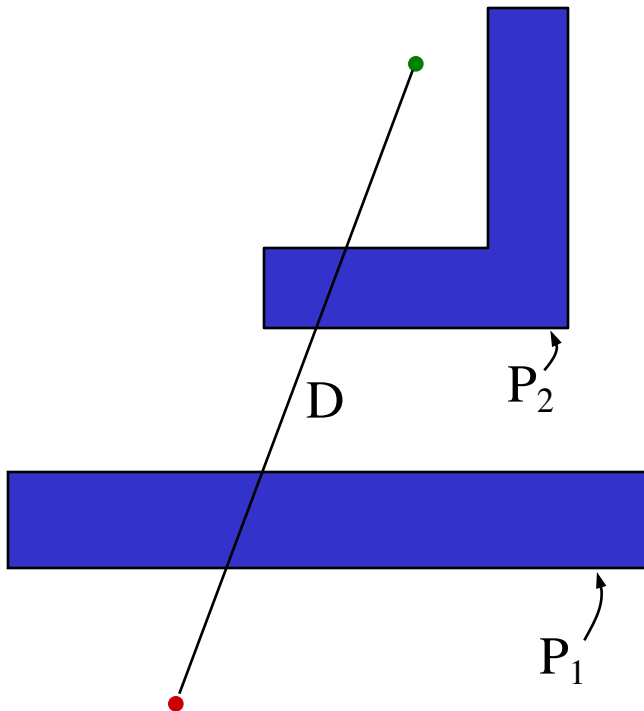
Lower and upper bounds?

Lower bound: D

Upper bound: $D + 1.5 \sum_i P_i$

How good a bound?

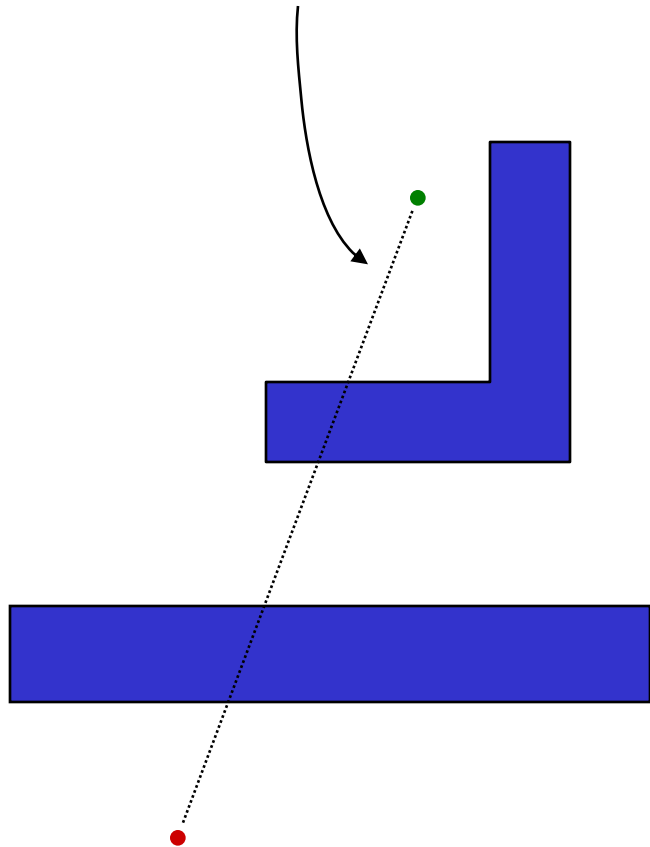
How good an algorithm?



A better bug?

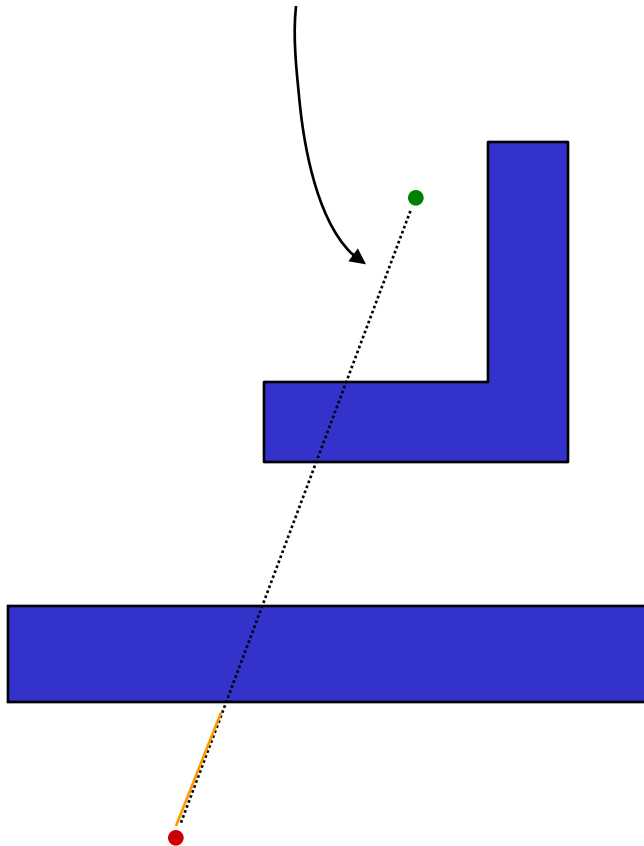
Call the line from the starting point to the goal the *s-line*

“Bug 2” algorithm



A better bug?

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



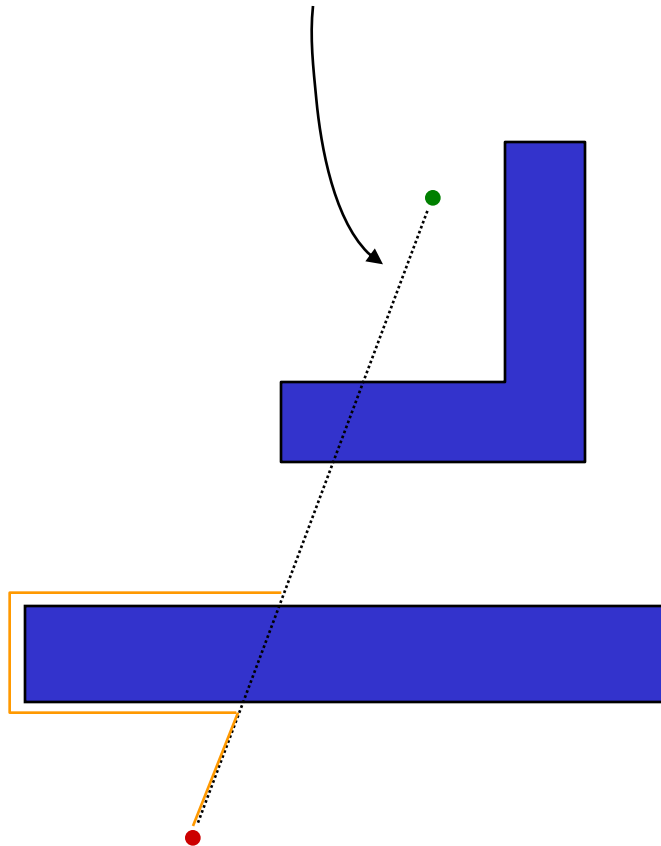
“Bug 2” algorithm

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



A better bug?

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



“Bug 2” algorithm

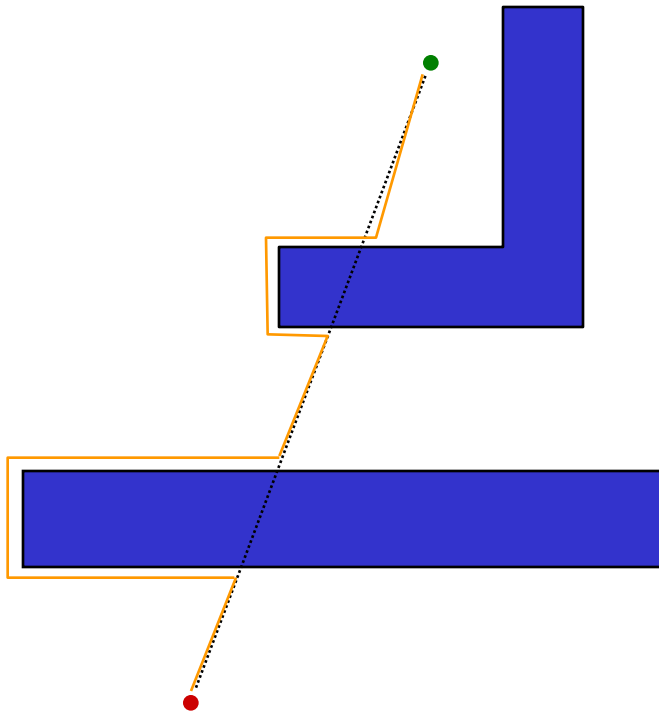
- 1) head toward goal on the *s-line*
- 2) if an obstacle is in the way, follow it until encountering the *s-line* again.



A better bug?

“Bug 2” algorithm

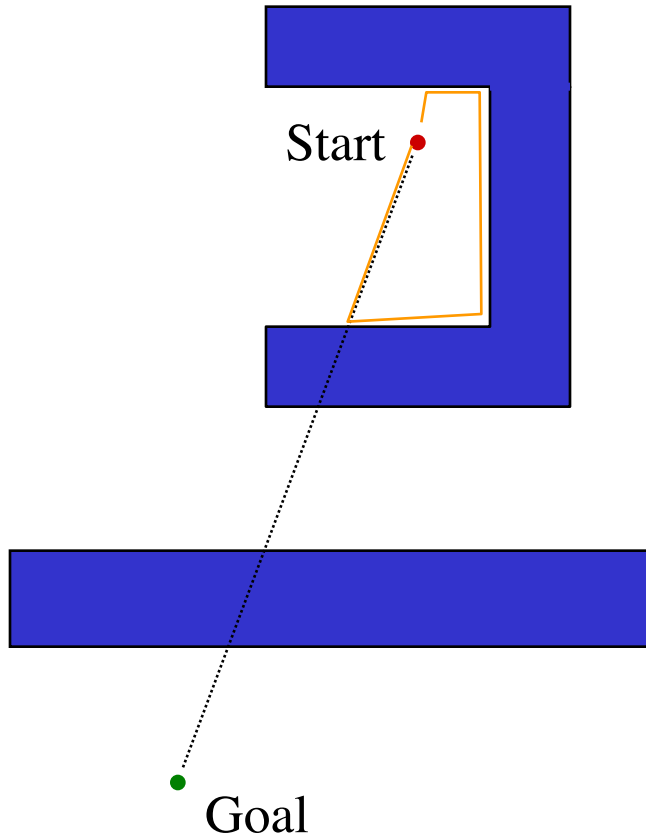
s-line



- 1) head toward goal on the *s-line*
- 2) if an obstacle is in the way, follow it until encountering the *s-line* again.
- 3) Leave the obstacle and continue toward the goal

A better bug?

“Bug 2” algorithm

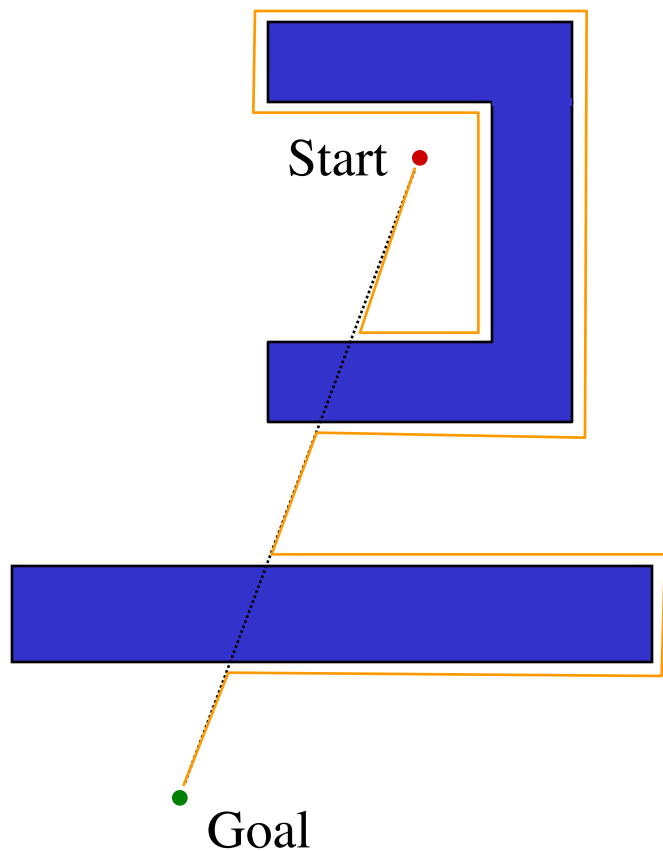


- 1) head toward goal on the *s-line*
- 2) if an obstacle is in the way, follow it until encountering the *s-line* again *closer to the goal*.
- 3) Leave the obstacle and continue toward the goal



Bug 2 analysis

Distance Traveled



What are bounds on the path length that the robot takes?

Available Information:

D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

Lower and upper bounds?

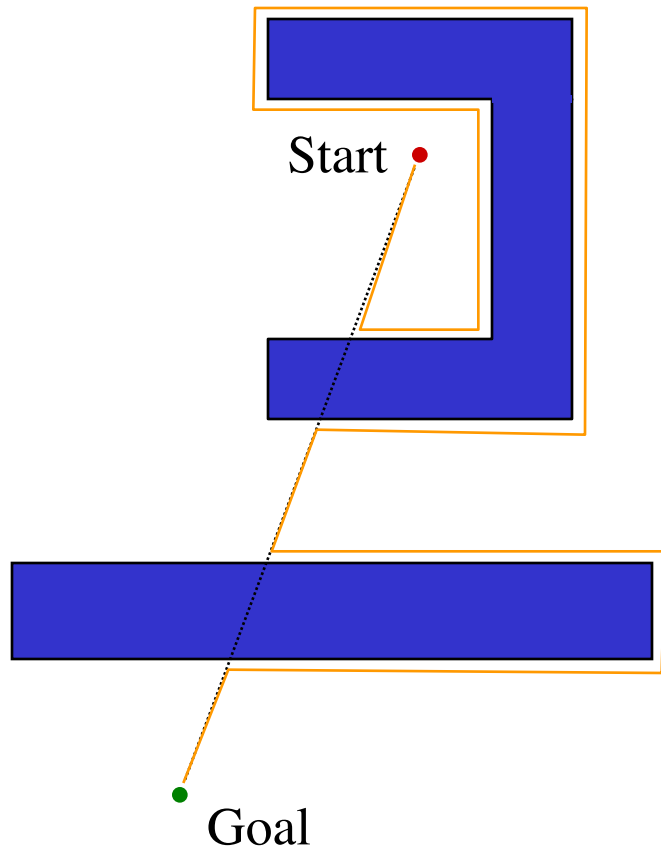
Lower bound:

Upper bound:



Bug 2 analysis

Distance Traveled



What are bounds on the path length that the robot takes?

Available Information:

D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

N_i = number of s-line intersections with the i th obstacle

Lower and upper bounds?

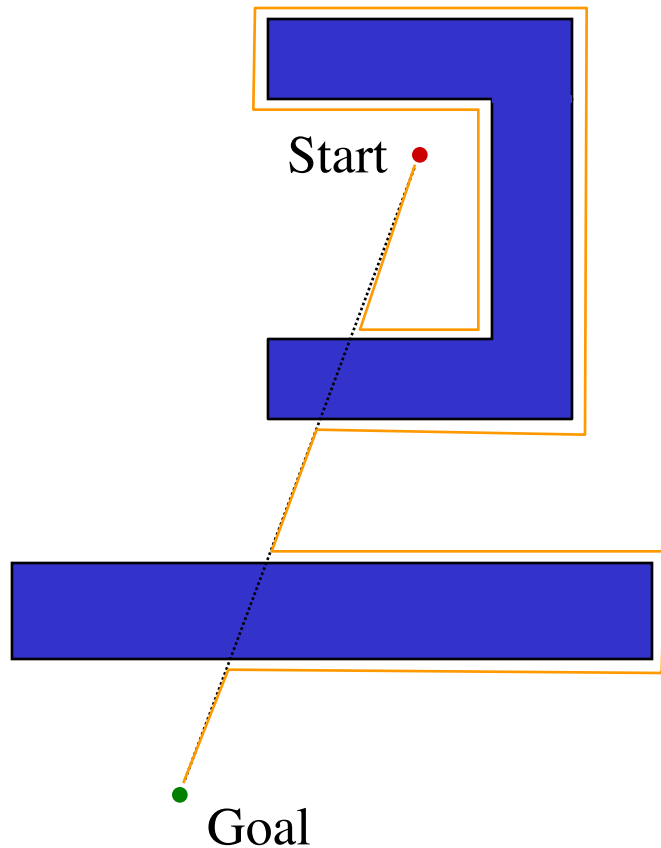
Lower bound:

Upper bound:



Bug 2 analysis

Distance Traveled



What are bounds on the path length that the robot takes?

Available Information:

D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

N_i = number of s-line intersections with the i th obstacle

Lower and upper bounds?

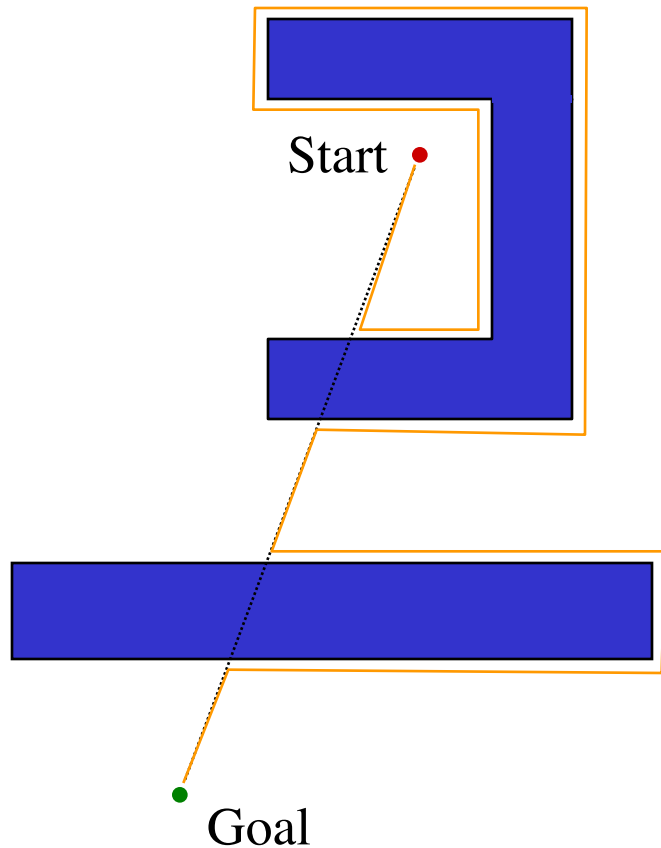
Lower bound: D

Upper bound:



Bug 2 analysis

Distance Traveled



What are bounds on the path length that the robot takes?

Available Information:

D = straight-line distance from start to goal

P_i = perimeter of the i th obstacle

N_i = number of s-line intersections with the i th obstacle

Lower and upper bounds?

Lower bound: D

Upper bound: $D + 0.5 \sum_i N_i P_i$

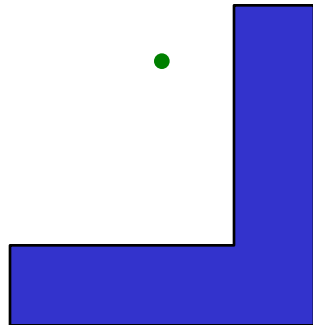


head-to-head comparison

or thorax-to-thorax, perhaps

What are worlds in which Bug 2 does better than Bug 1 (and vice versa) ?

Bug 2 beats Bug 1



Bug 1 beats Bug 2

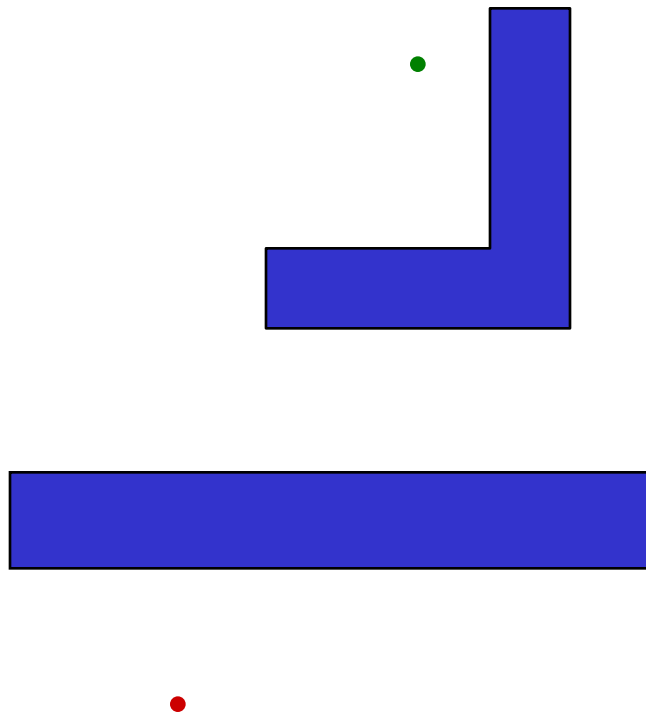


head-to-head comparison

or thorax-to-thorax, perhaps

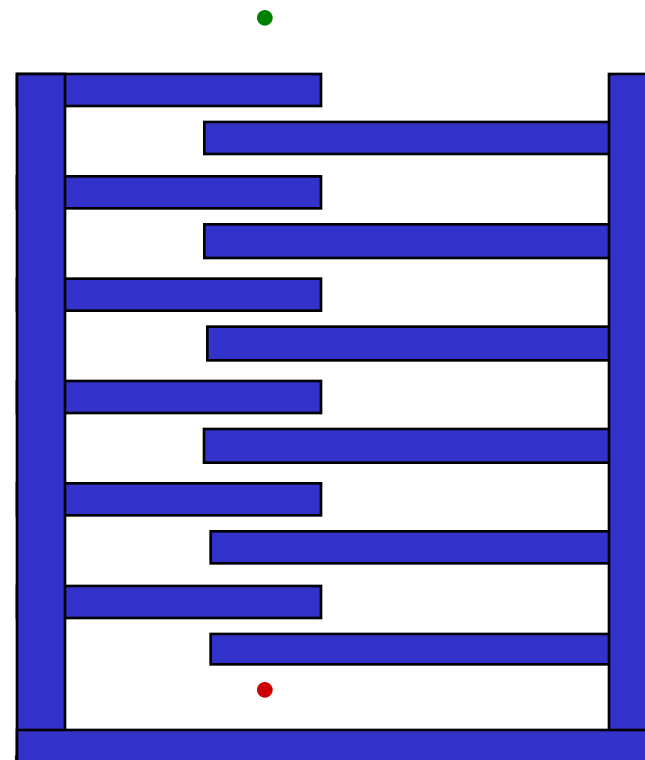
What are worlds in which Bug 2 does better than Bug 1 (and vice versa) ?

Bug 2 beats Bug 1



Bug 1 beats Bug 2

“zipper world”

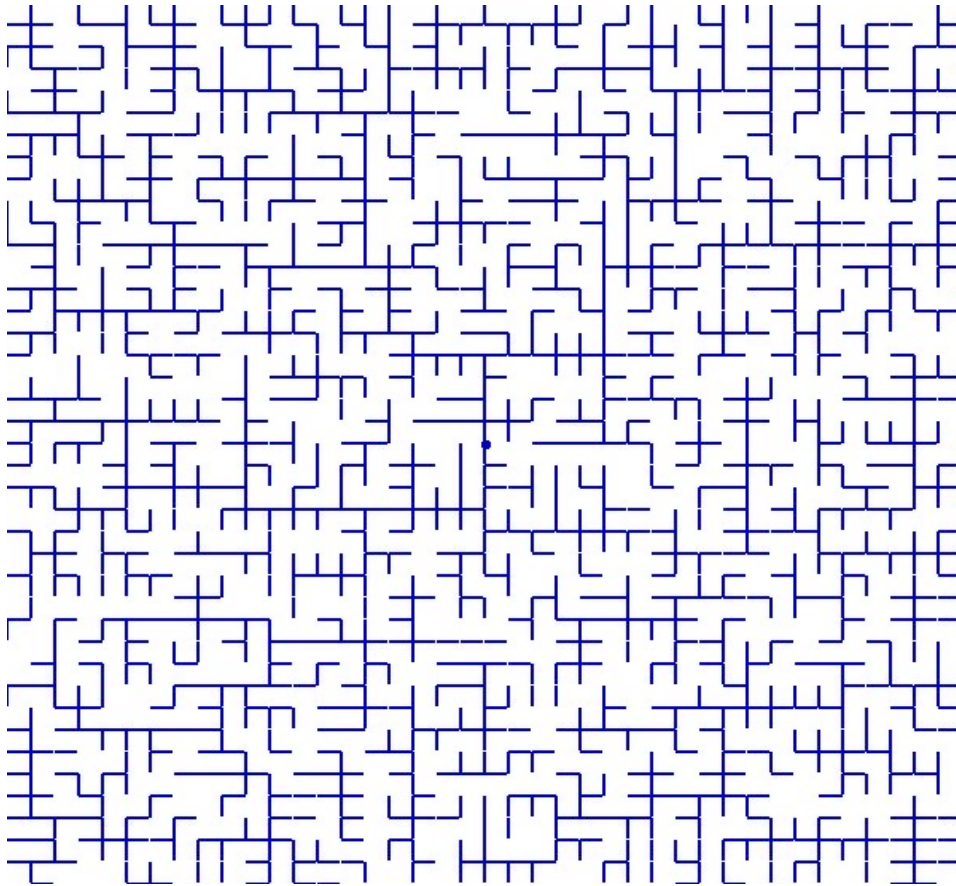


Bug Mapping



Other bug-like algorithms

The Pledge maze-solving algorithm

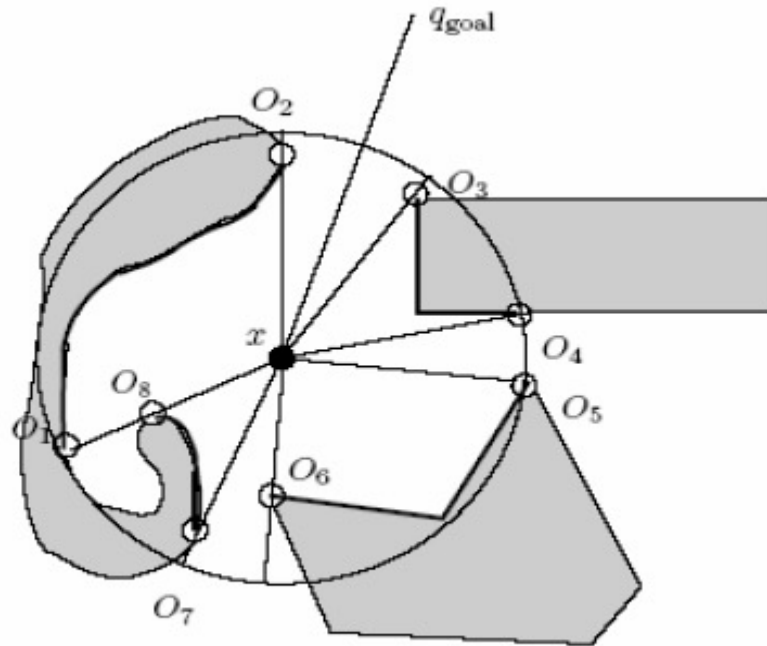


1. Go to a wall
2. Keep the wall on your right
3. Continue until out of the maze

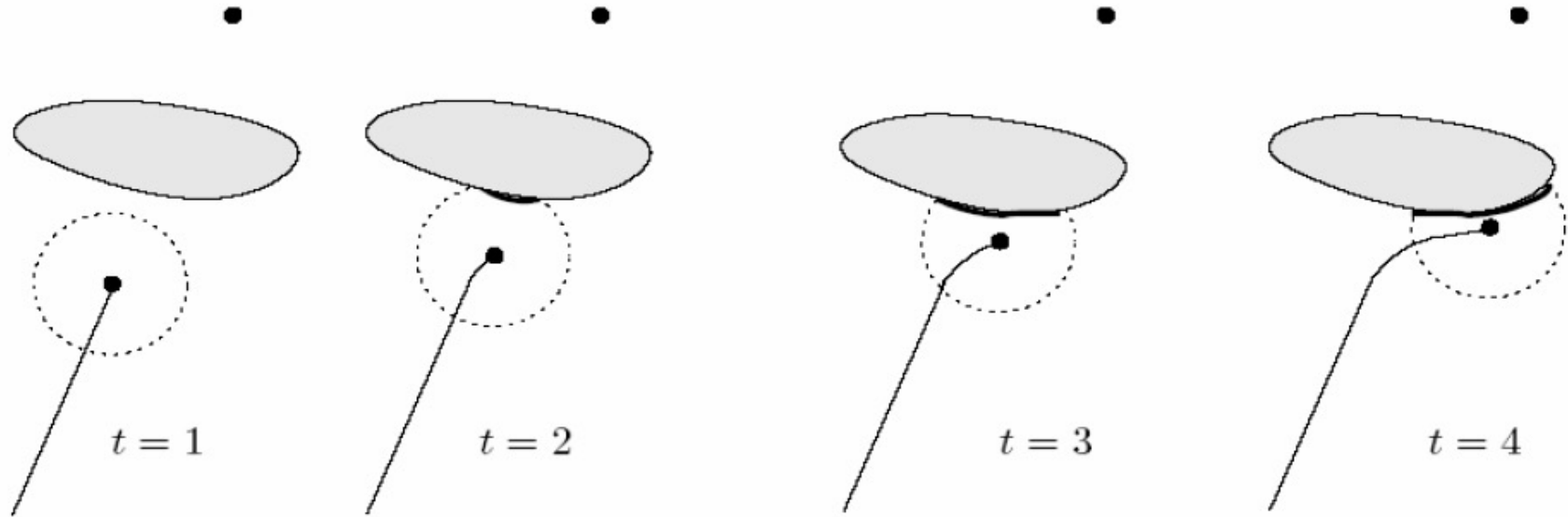


Tangent Bug

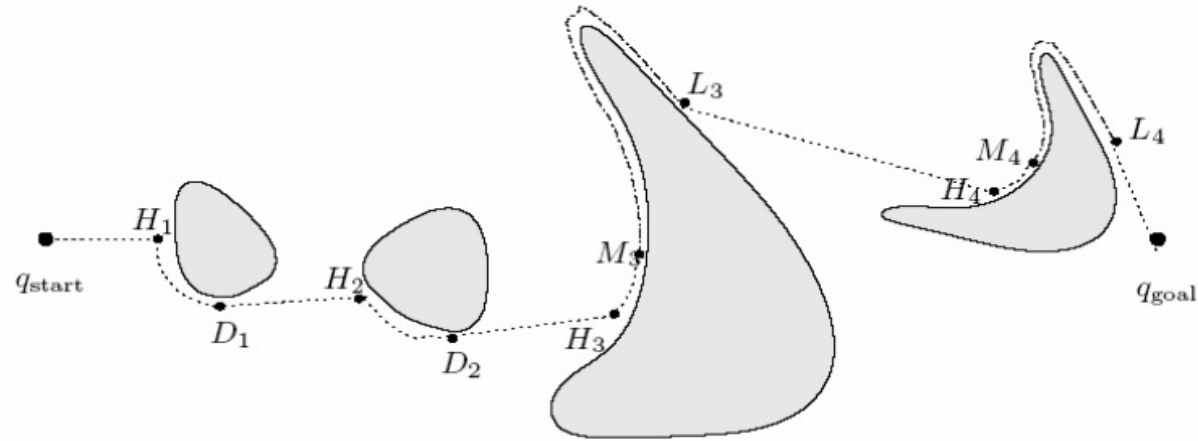
- Limited Range Sensor
- Tangent Bug relies on finding endpoints of finite, continuous segments of the obstacles



Tangent Bug



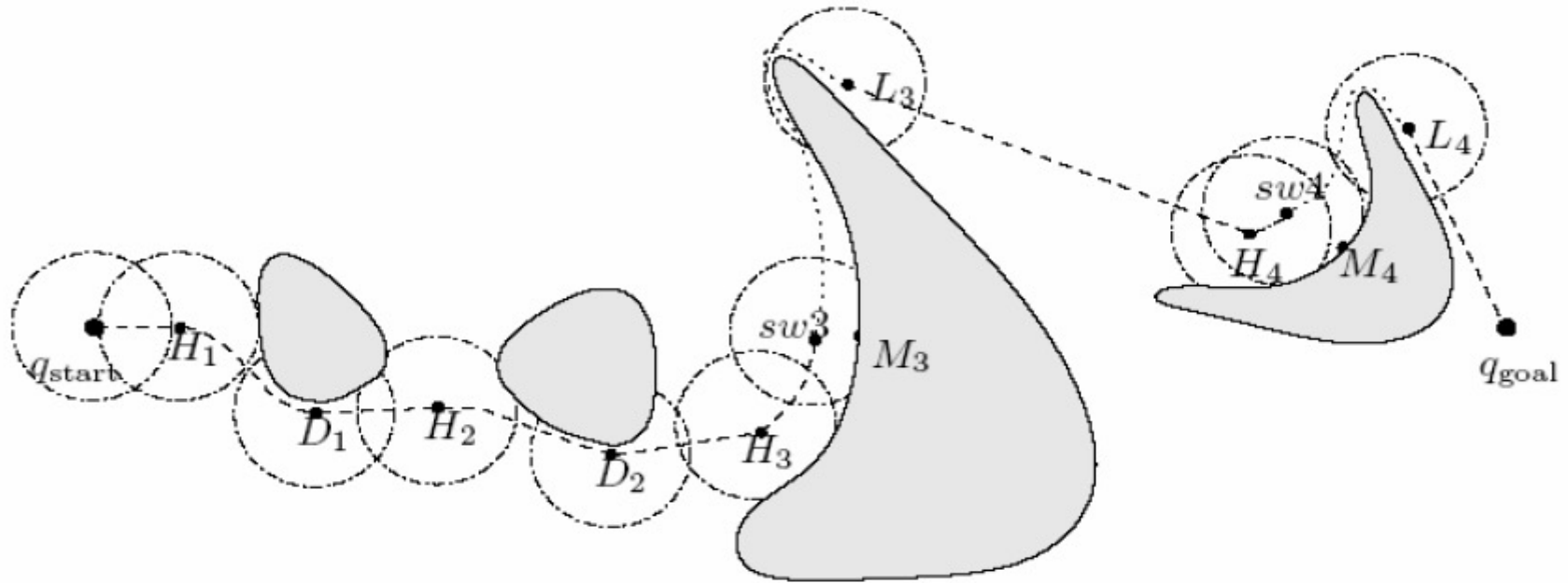
Contact Sensor Tangent Bug



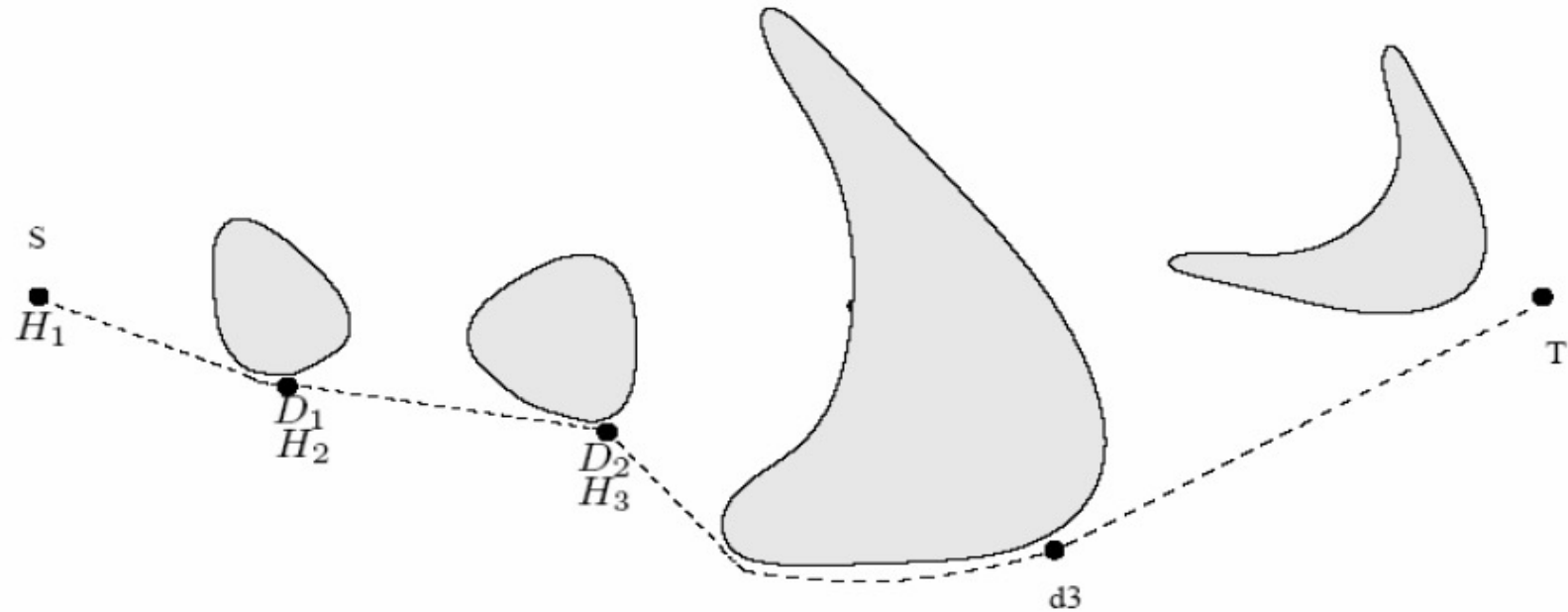
1. Robot moves toward goal until it hits obstacle 1 at H_1
2. Pretend there is an infinitely small sensor range and the direction which minimizes the heuristic is to the right
3. Keep following obstacle until robot can go toward obstacle again
4. Same situation with second obstacle
5. At third obstacle, the robot turned left until it could not increase heuristic
6. $D_{followed}$ is distance between M_3 and goal, d_{reach} is distance between robot and goal because sensing distance is zero



Limited Sensor Range Tangent-Bug

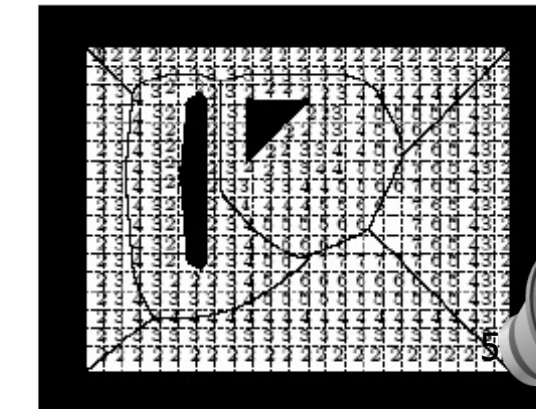
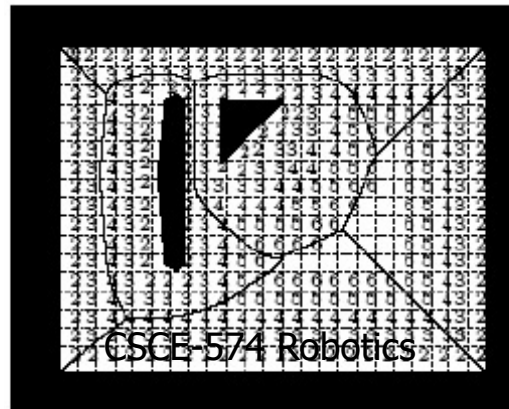
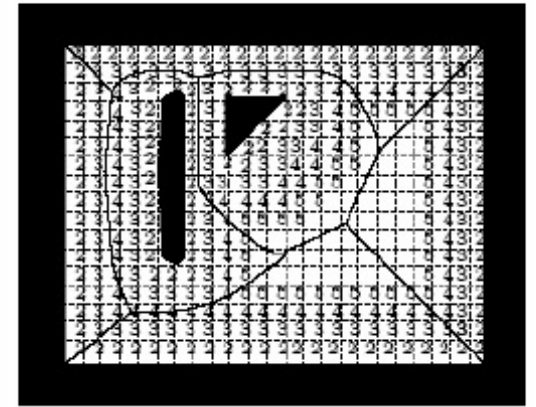
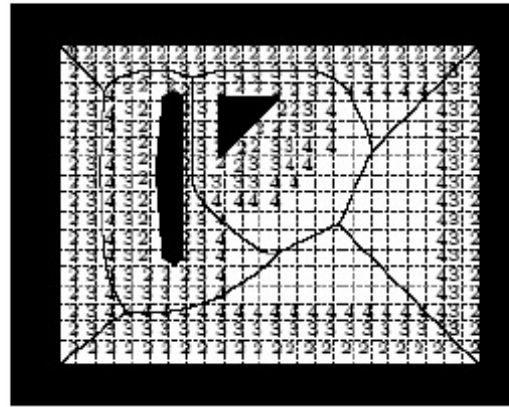
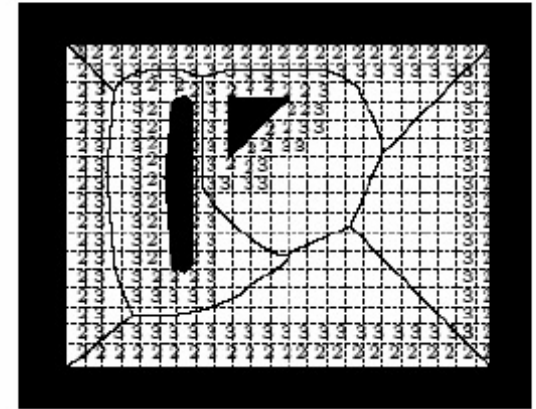
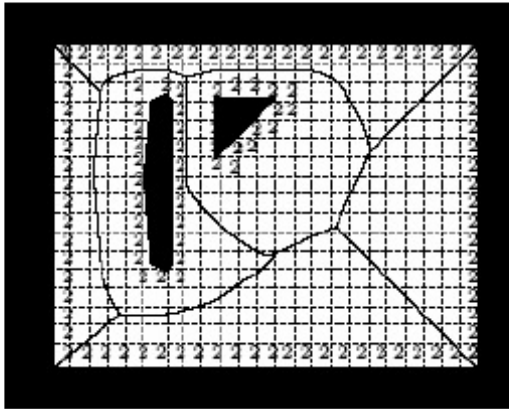


Infinite Sensor Range Tangent Bug



Known Map

Brushfire Transform



The Wavefront Planner: Setup

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with “0” to the current cell + 1
 - 4-Point Connectivity or 8-Point Connectivity?
 - Your Choice. We’ll use 8-Point Connectivity in our example

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action (Part 2)

- Now repeat with the modified cells
 - This will be repeated until no 0's are adjacent to cells with values ≥ 2
- 0's will only remain when regions are unreachable

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	
1	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3	
0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	2	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action (Part 3)

- Repeat

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	
3	0	0	0	0	1	1	1	1	1	1	1	1	5	5	5	
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action (Part 3)

- Repeat

7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	1	1	1	1	1	1	6	6	6	6	
3	0	0	0	0	1	1	1	1	1	1	1	5	5	5	5	
2	0	0	0	0	0	0	0	0	0	0	6	5	4	4	4	
1	0	0	0	0	0	0	0	0	0	0	6	5	4	3	3	
0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	2	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action (Part 3)

- Until Done
 - 0's would only remain in the unreachable areas

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9	
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6	
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5	
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



The Wavefront in Action

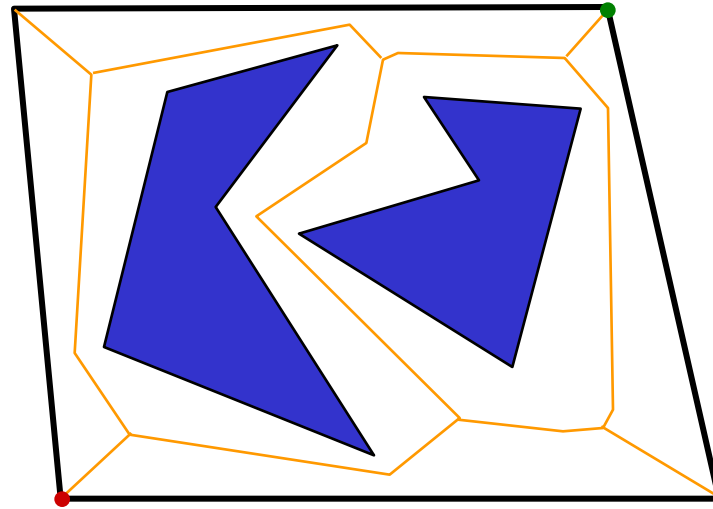
- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
 - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal



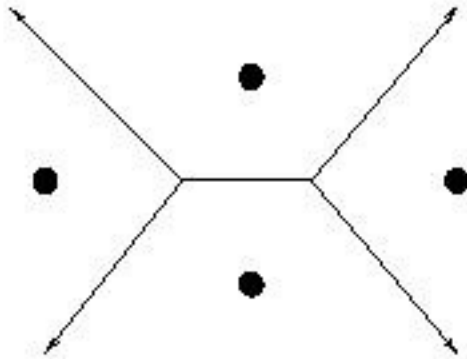
Two possible shortest paths shown



An alternative roadmap



Voronoi diagrams

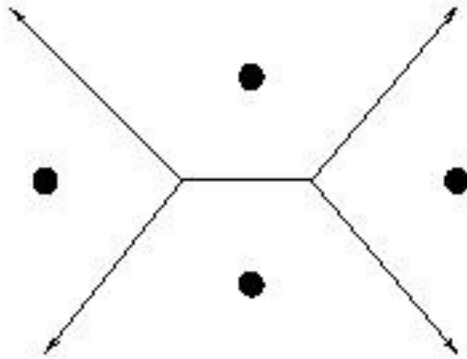


These line segments make up the **Voronoi diagram** for the four points shown here.

Solves the “Post Office Problem”

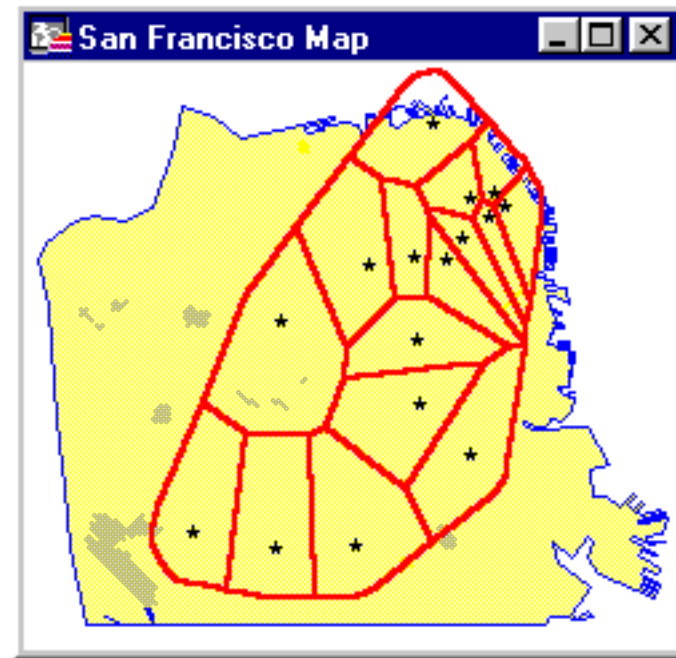


Voronoi diagrams



These line segments make up the **Voronoi diagram** for the four points shown here.

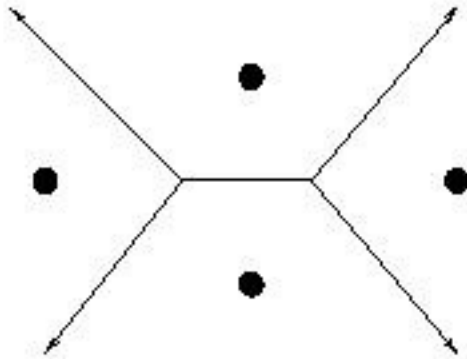
Solves the “Post Office Problem”



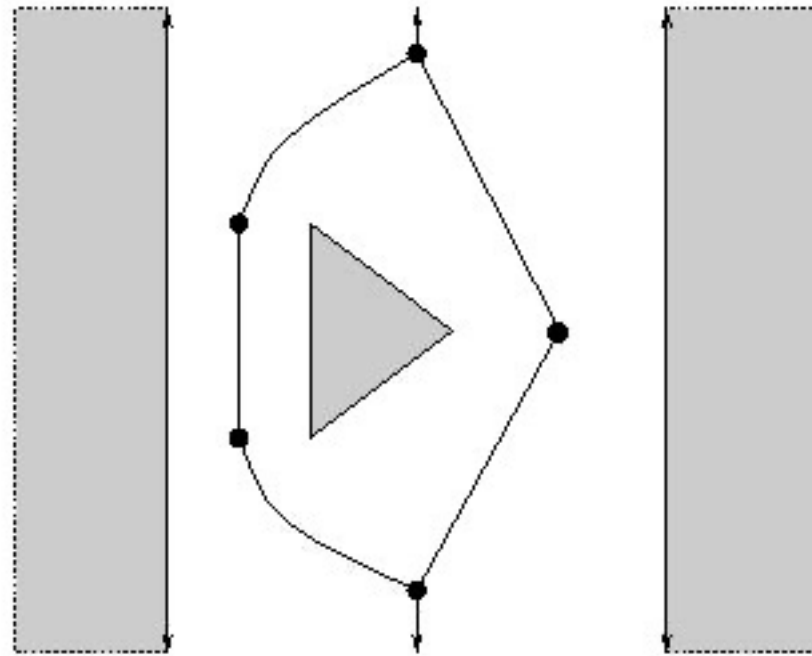
or, perhaps, more important problems...



Voronoi diagrams



“true” Voronoi diagram
(isolates a set of points)



generalized Voronoi diagram

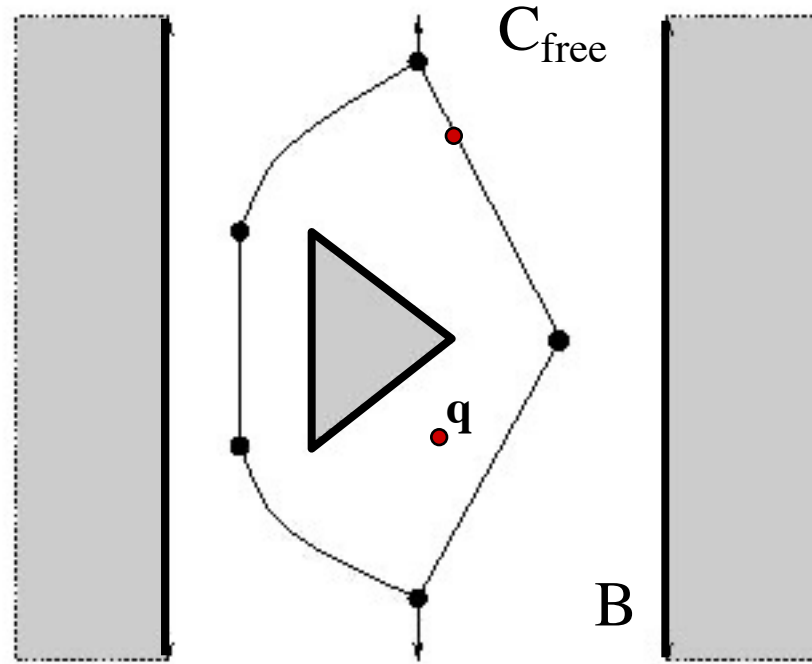
What is it?



Voronoi diagrams

Let B = the boundary of C_{free} .

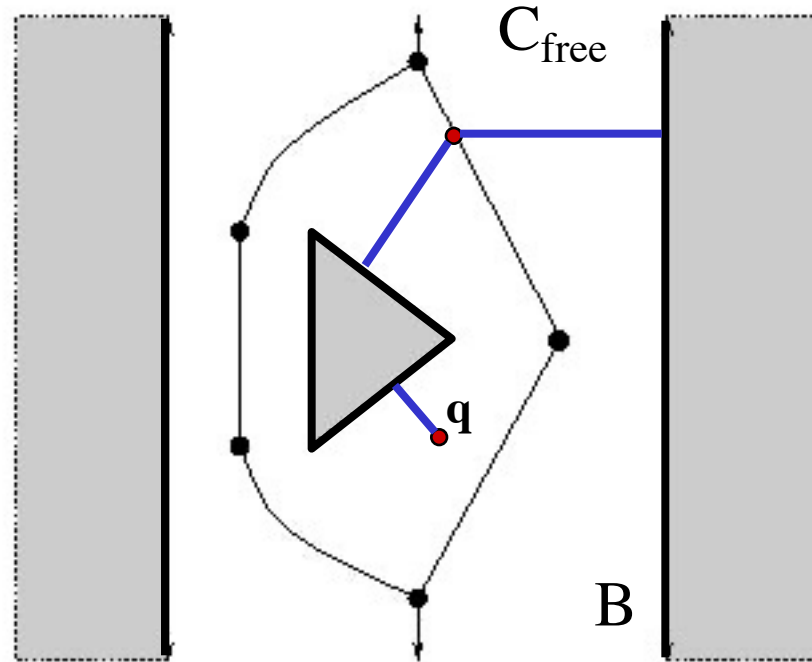
Let q be a point in C_{free} . (\bullet)



Voronoi diagrams

Let B = the boundary of C_{free} .

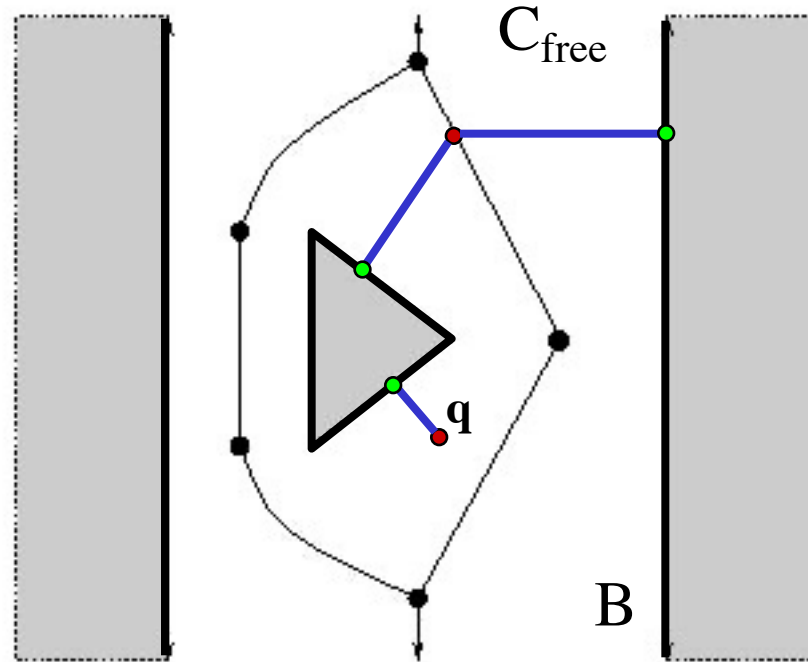
Let q be a point in C_{free} .



Define *clearance*(q) = $\min \{ |q - p| \}$, for all $p \in B$



Voronoi diagrams



Let B = the boundary of C_{free} .

Let q be a point in C_{free} .

Define *clearance*(q) = $\min \{ |q - p| \}$, for all $p \in B$

Define *near*(q) = $\{ p \in B \text{ such that } |q - p| = \textit{clearance}(q) \}$



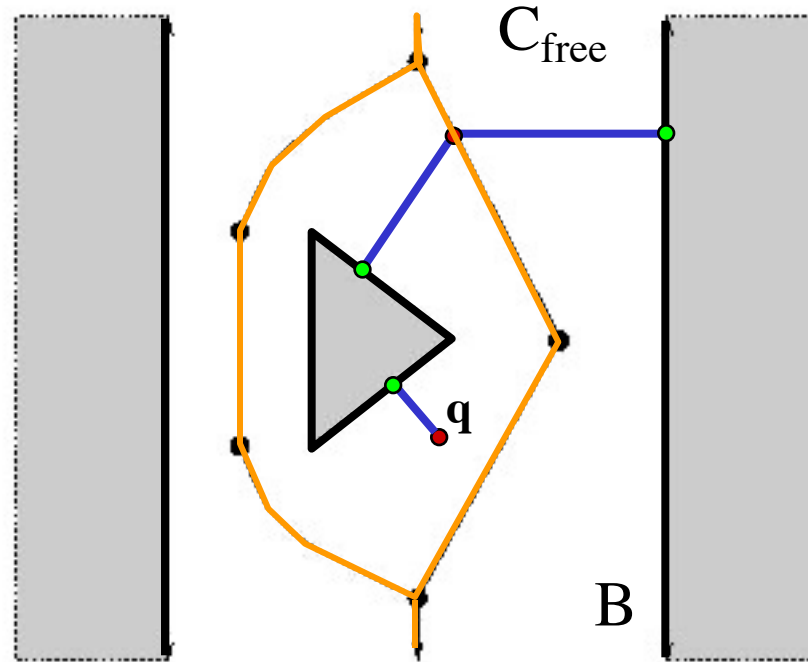
Voronoi diagrams

Evaluation

- + maximizes distance from obstacles
- + reduces to graph search
- + can be used in higher-dimensions
- nonoptimal
- real diagrams tend to be noisy

Let B = the boundary of C_{free} .

Let q be a point in C_{free} .



Define *clearance*(q) = $\min \{ |q - p| \}$, for all $p \in B$

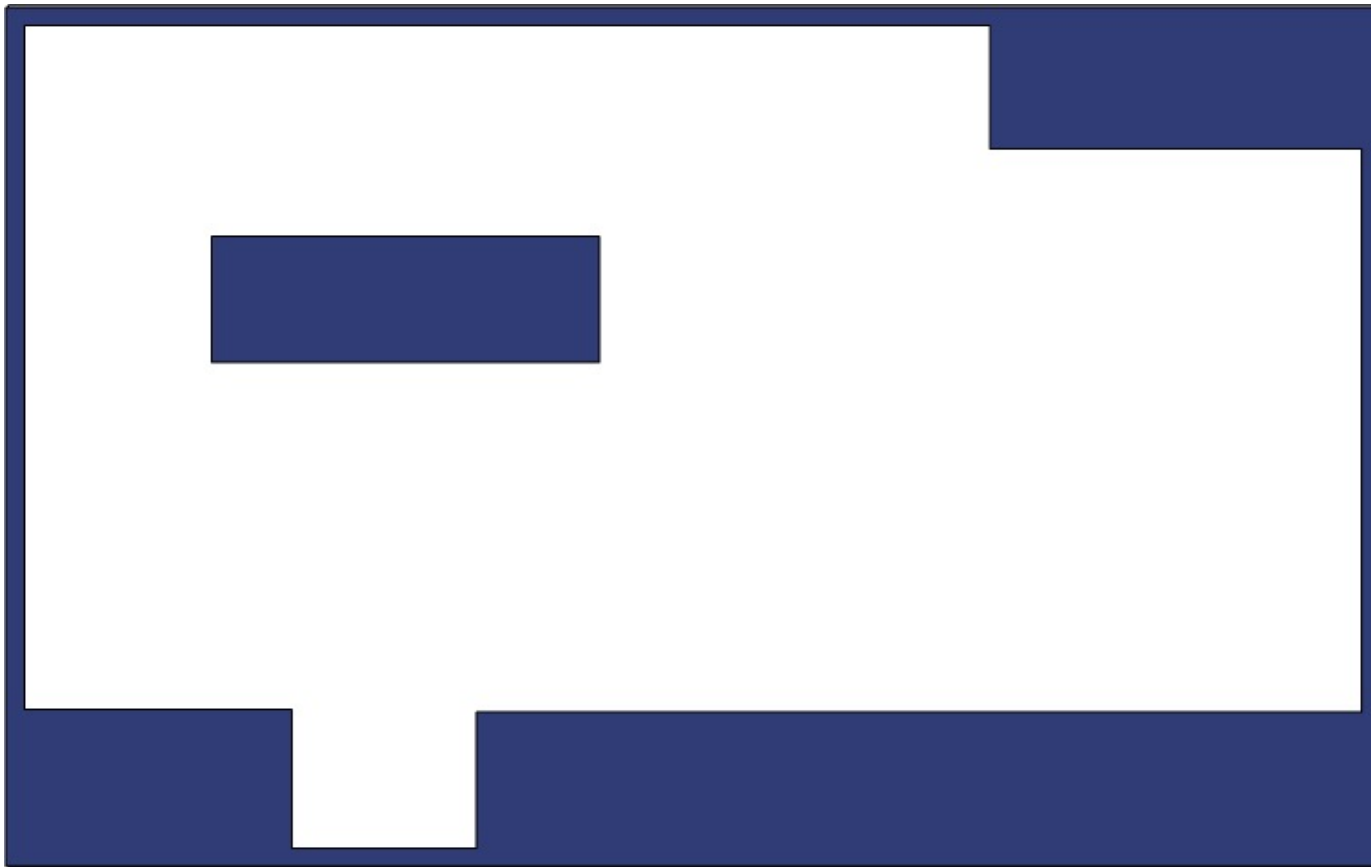
Define *near*(q) = $\{ p \in B \text{ such that } |q - p| = \textit{clearance}(q) \}$

q is in the *Voronoi diagram* of C_{free} if $| \textit{near}(q) | > 1$

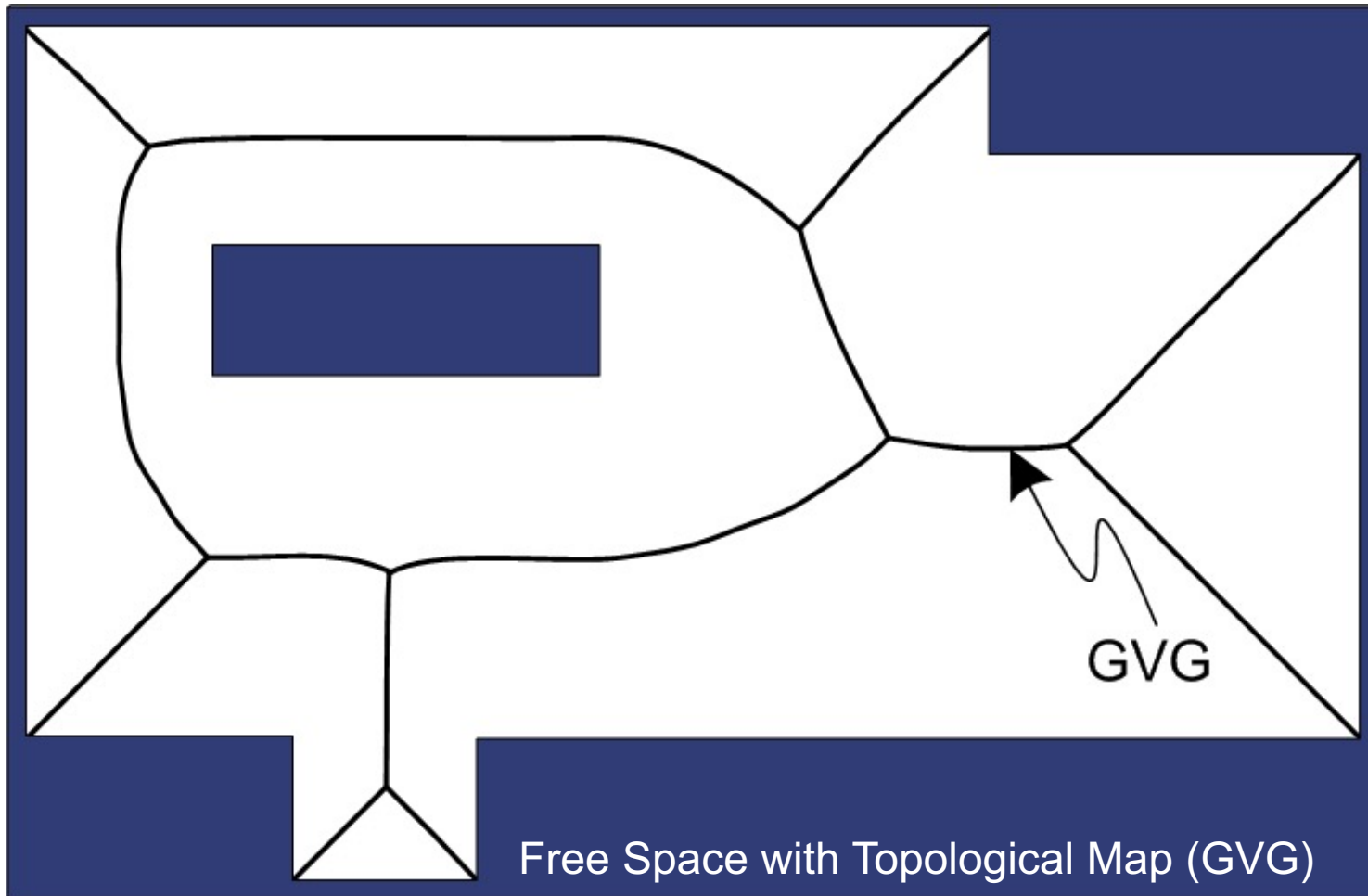
number of
set elements



Generalized Voronoi Graph (GVG)

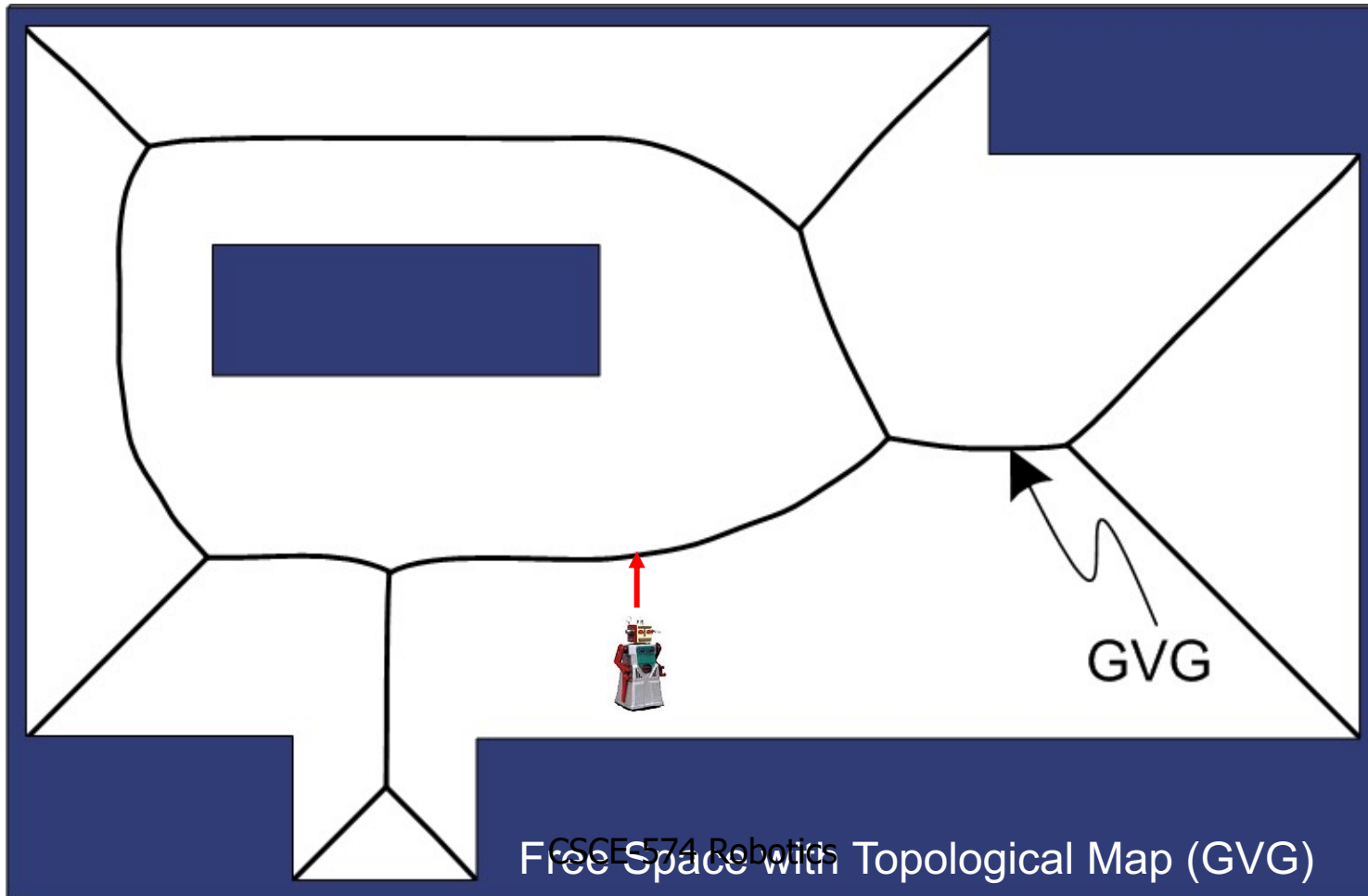


Generalized Voronoi Graph (GVG)



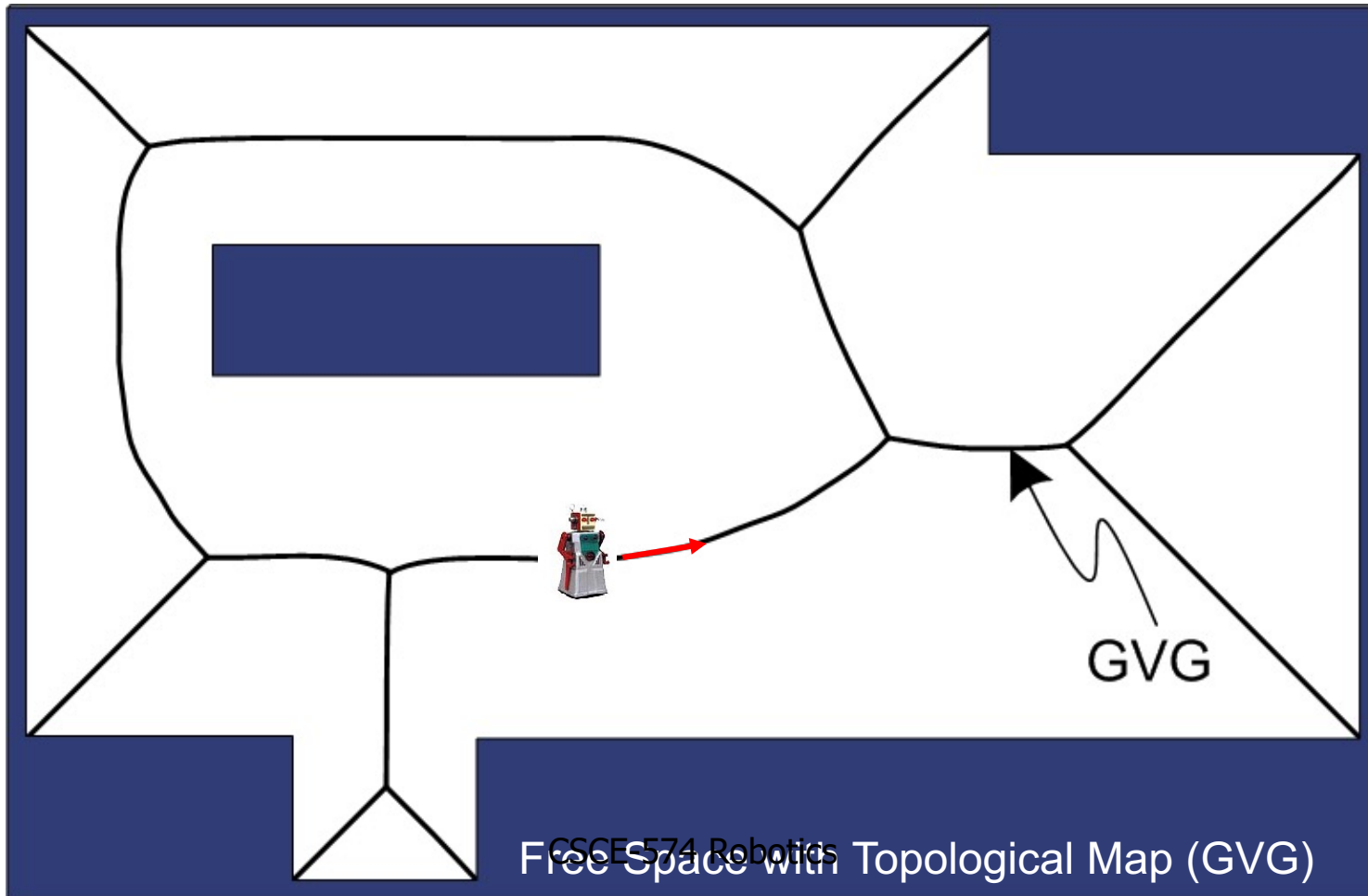
Generalized Voronoi Graph (GVG)

- Access GVG



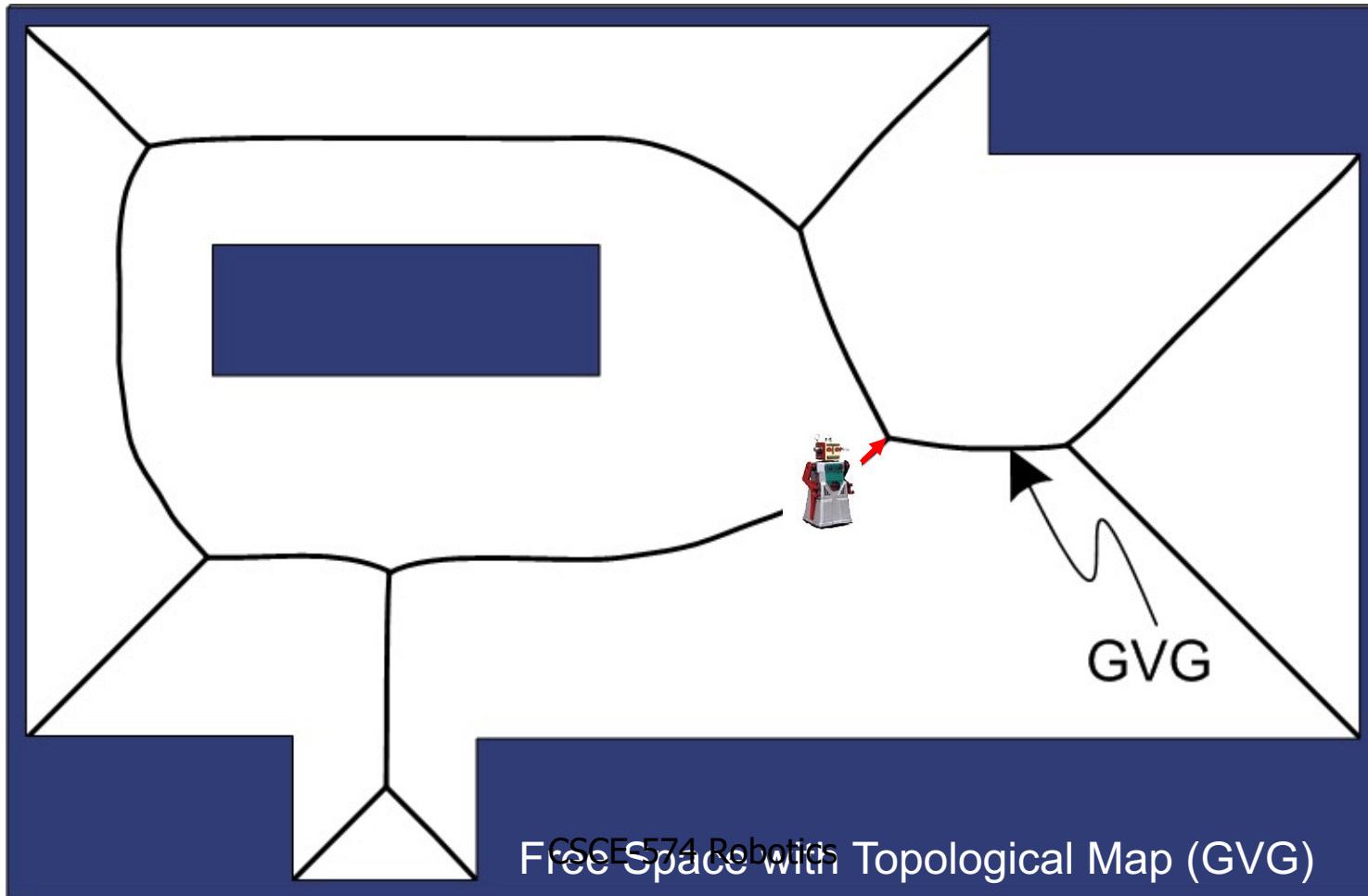
Generalized Voronoi Graph (GVG)

- Access GVG
- Follow Edge



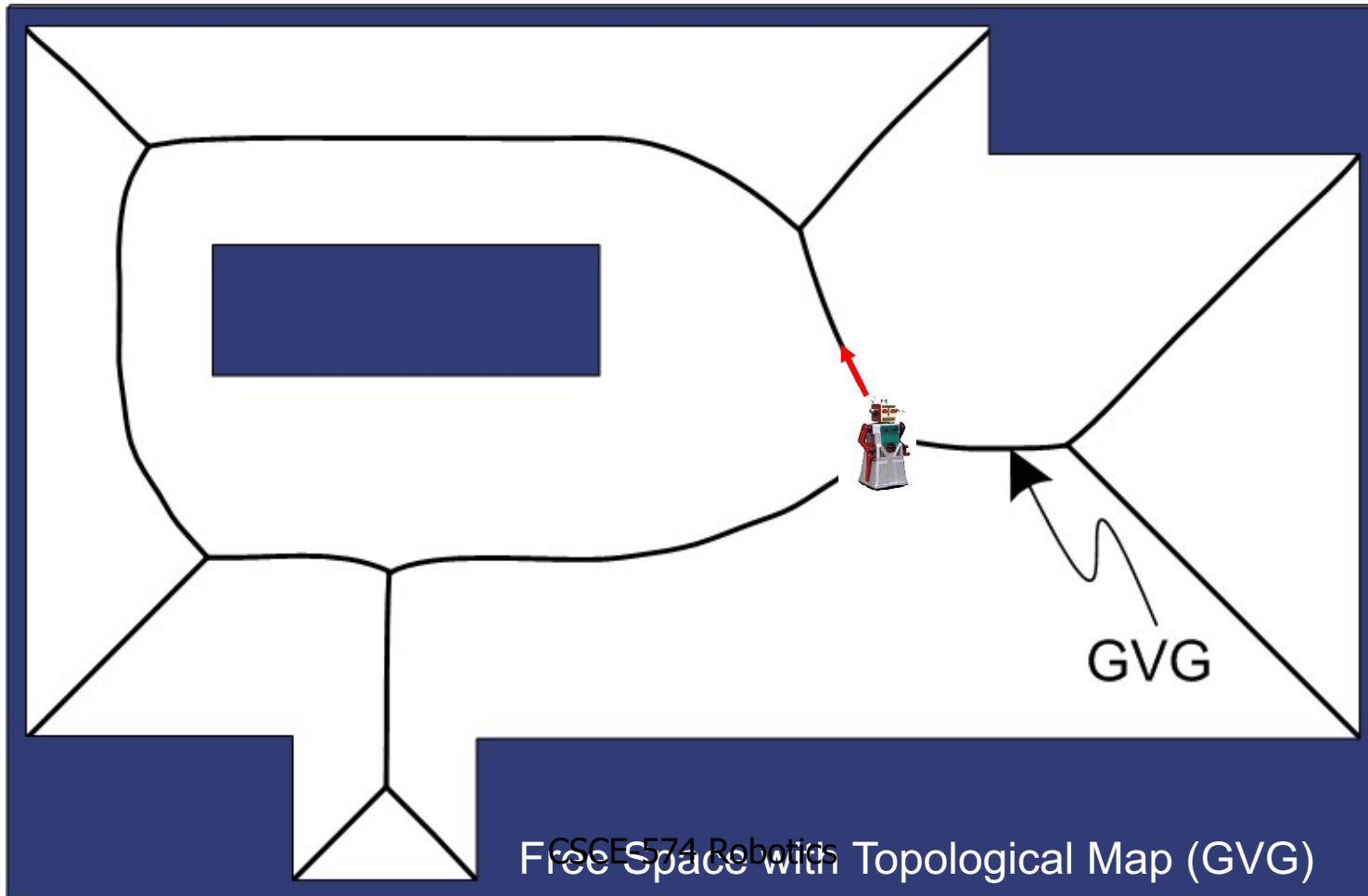
Generalized Voronoi Graph (GVG)

- Access GVG
- Home to the MeetPoint
- Follow Edge



Generalized Voronoi Graph (GVG)

- Access GVG
- Home to the MeetPoint
- Follow Edge
- Select Edge

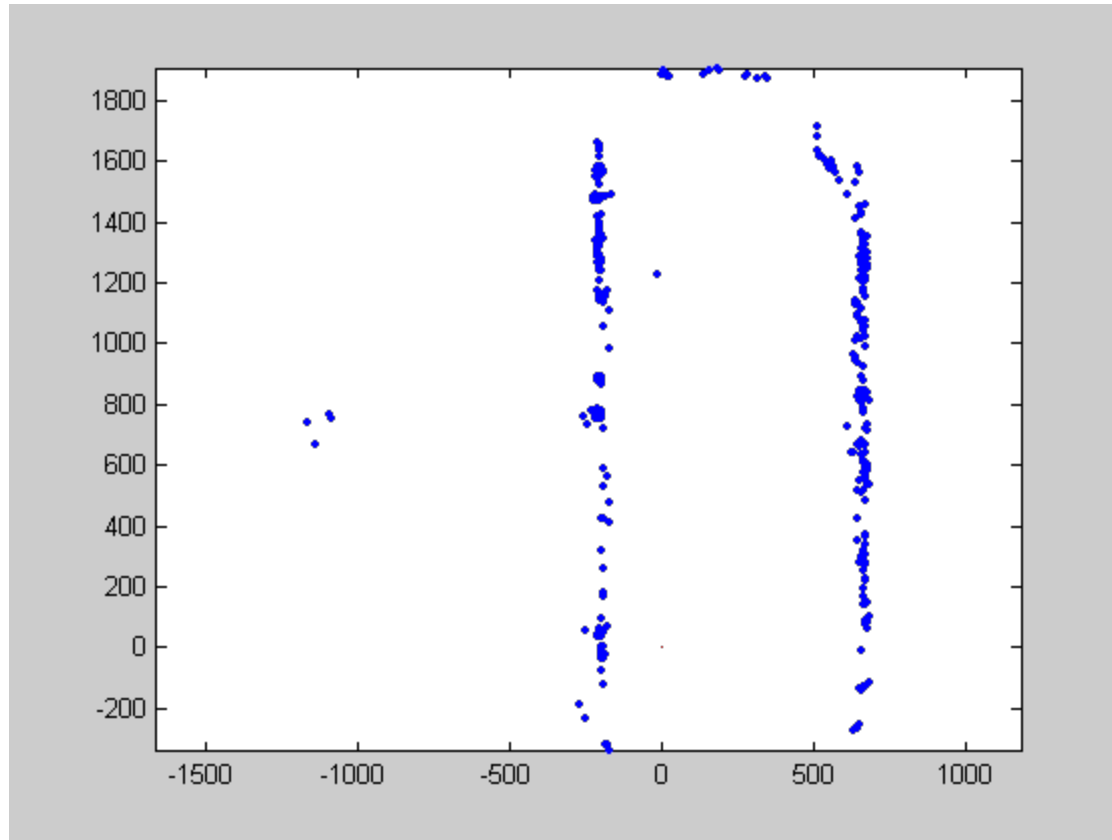


GVG construction using sonar

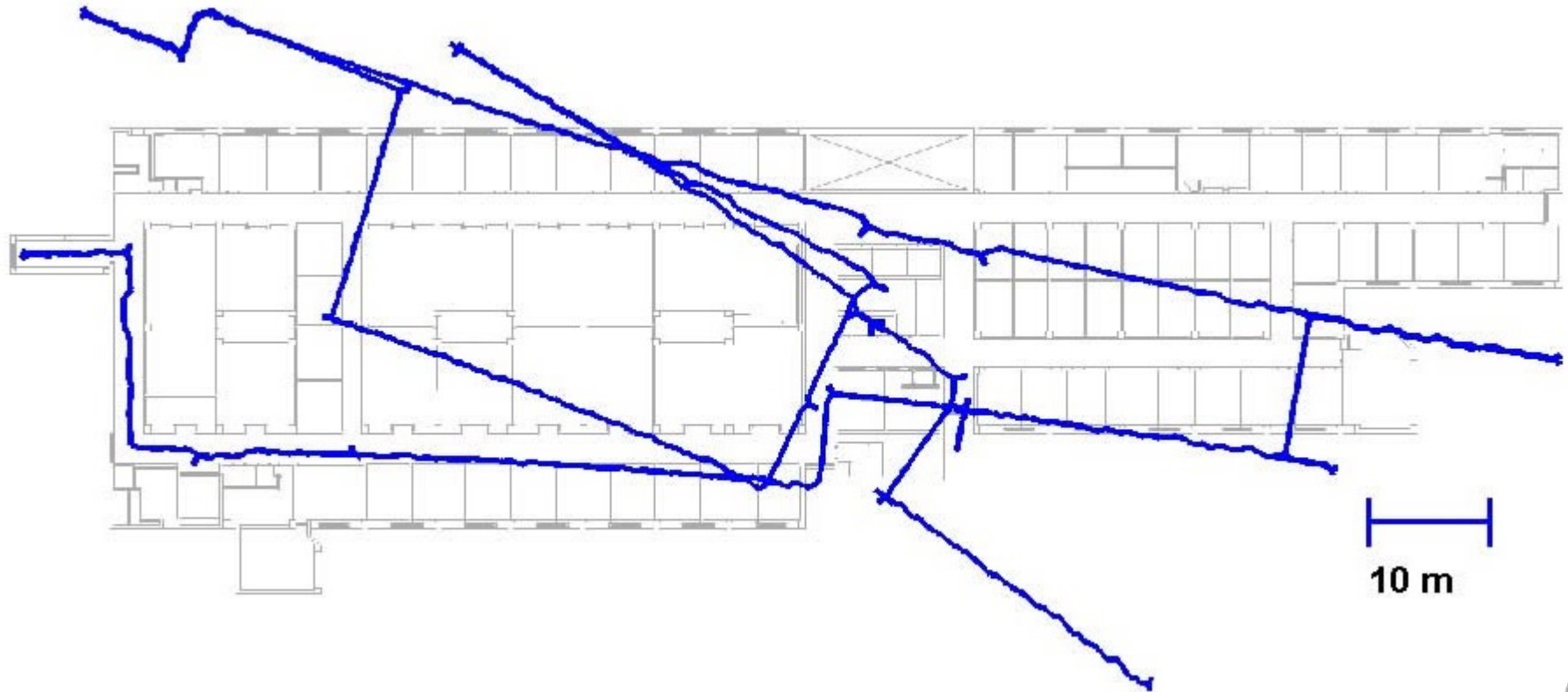


- Nomadic Scout
- Sonar (GVG navigation)
- Camera with omni-directional mirror (feature detection)
- Onboard 1.2 GHz processor

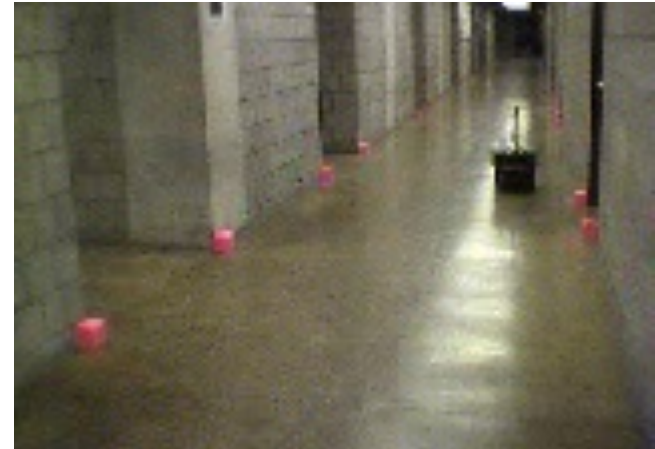
GVG construction using sonar



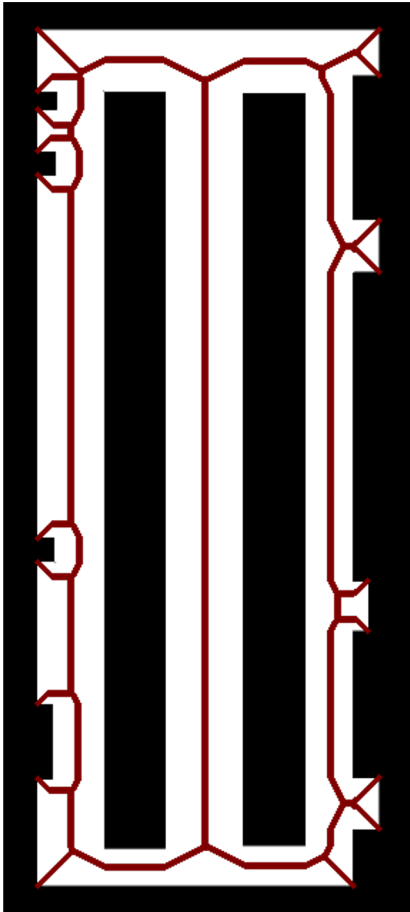
GVG construction using sonar



Slammer in Action



Removing Edges

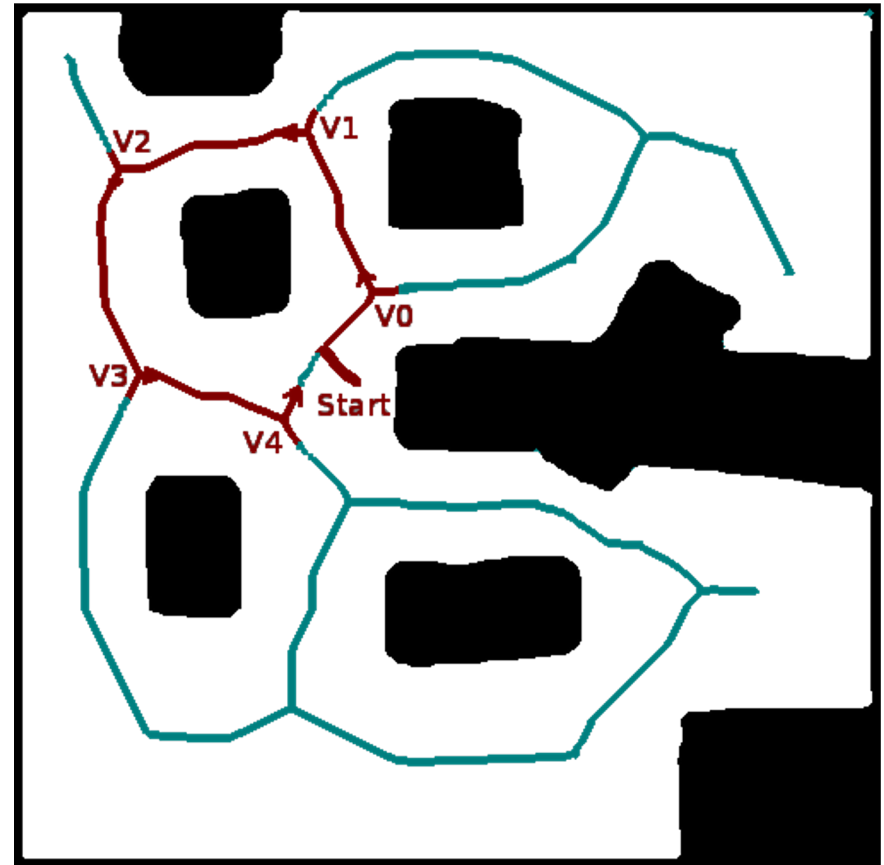
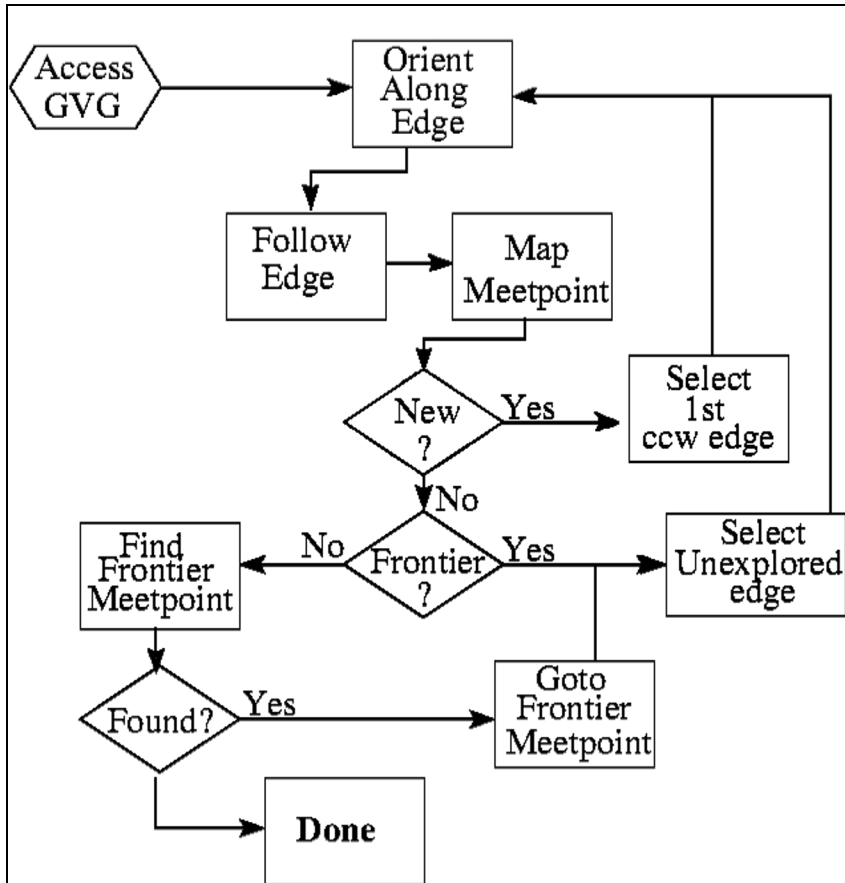


Meetpoint Detection

- 3σ uncertainty ellipse of explored meetpoints
- Meetpoint degree (branching factor)
- Distances to local obstacles
- Relative angle bearings
- Edge signature
 - Edge length
 - Edge Curvature
- Vertex signal

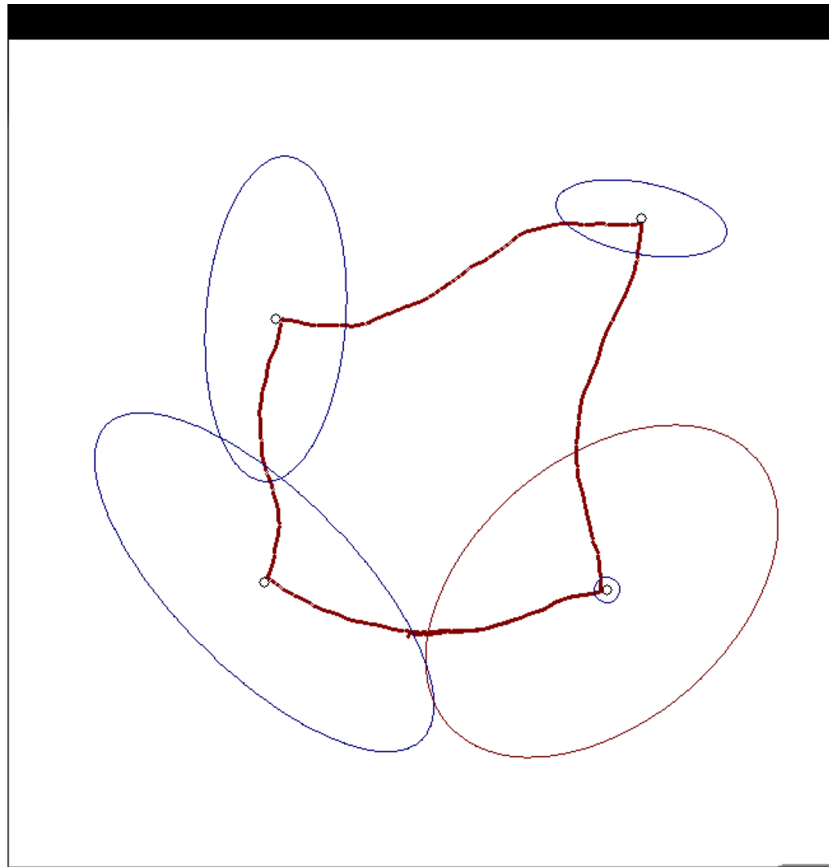


Ear-based Exploration

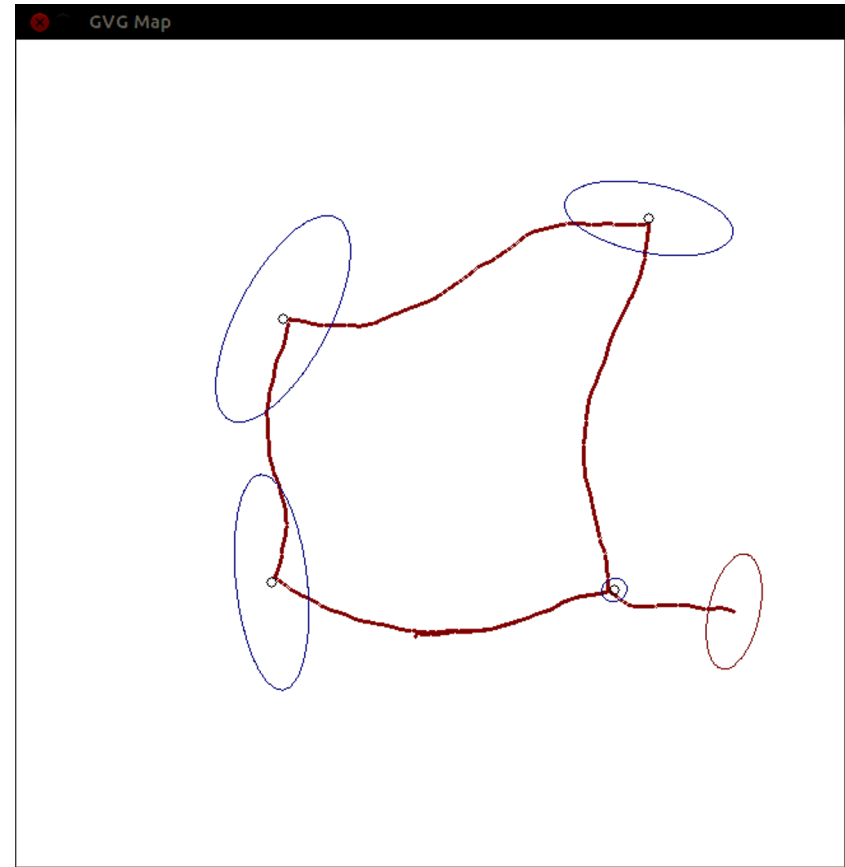


Uncertainty Reduction

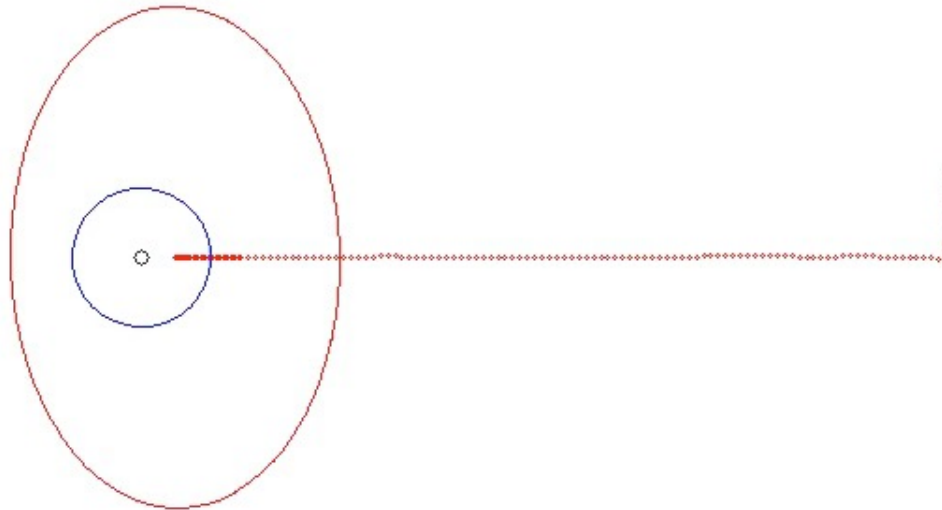
Before Loop-closure



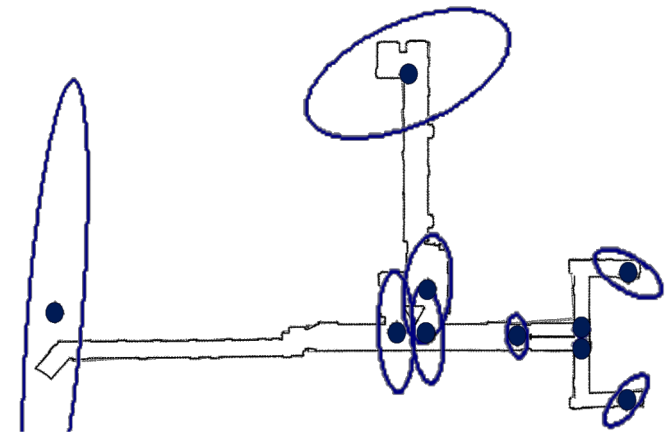
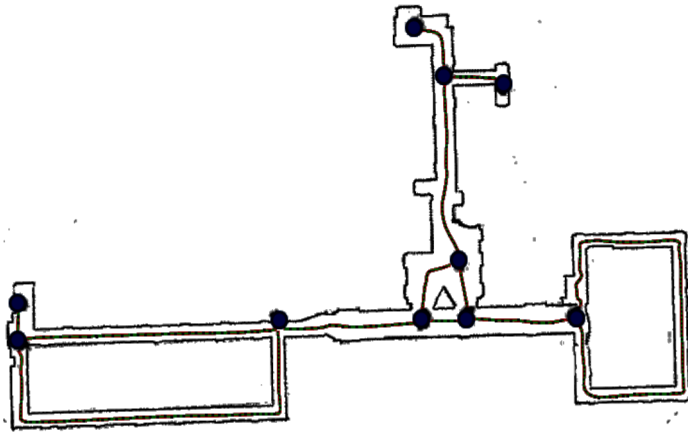
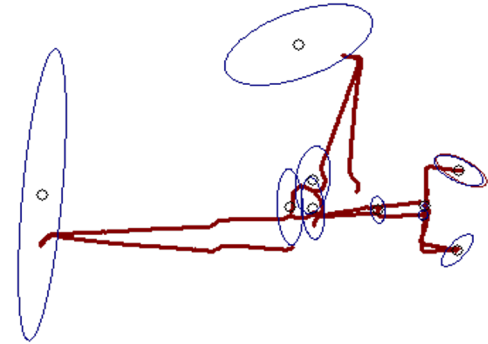
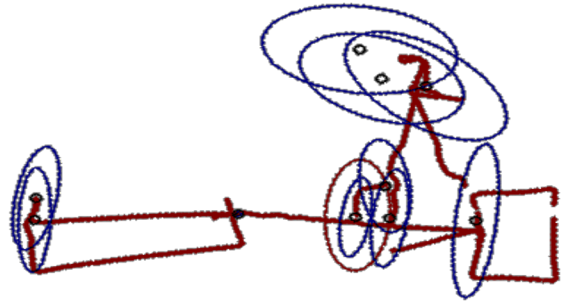
After Loop-closure



Simulated Environment



Real Environment



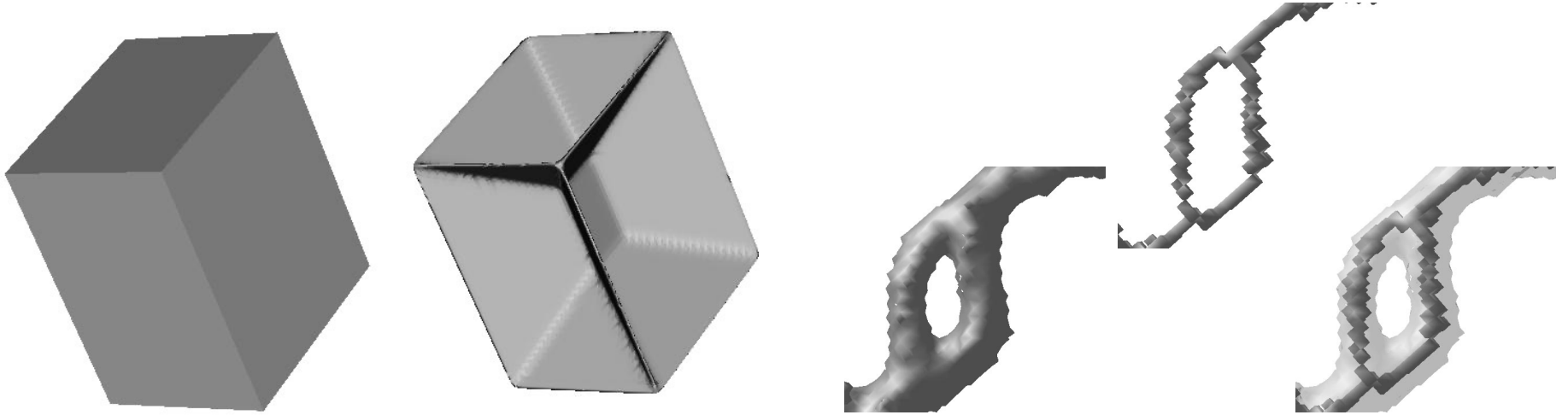
Work Presented at IROS 2014

Ear-based Exploration on Hybrid Metric/Topological Maps

Q. Zhang, D. Whitney, F. Shkurti, and I. Rekleitis
School of Computer Science, McGill University

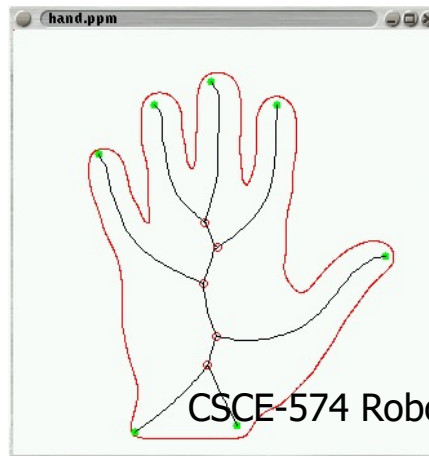


Voronoi applications



A retraction of a 3d object
== “medial surface”

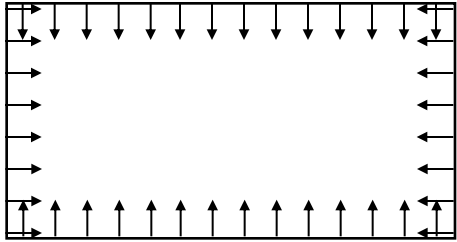
what? → Skeletonizations resulting from
constant-speed curve evolution



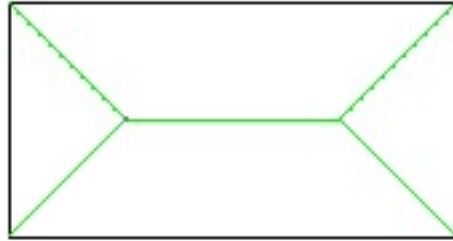
in 2d, it's called
a *medial axis*



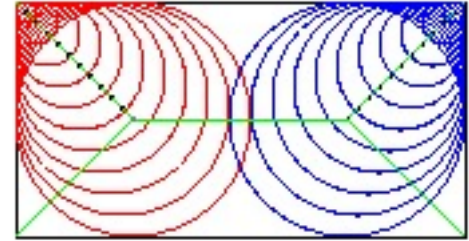
skeleton \leftrightarrow shape



curve evolution



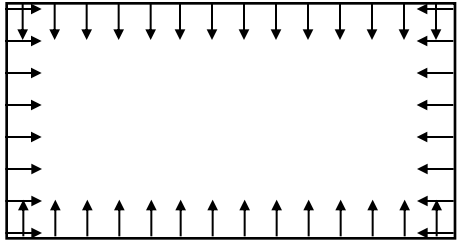
where wavefronts collide



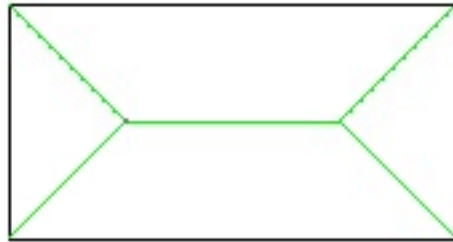
centers of maximal disks

again reduces a 2d (or higher) problem to a question about graphs...

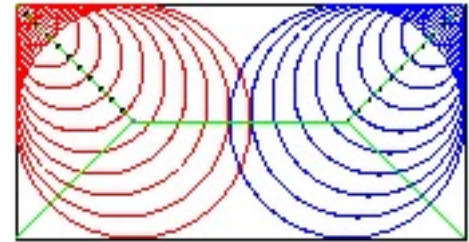
skeleton \leftrightarrow shape



curve evolution

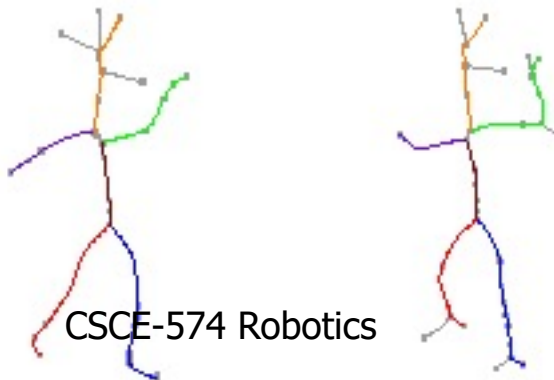


where wavefronts collide



centers of maximal disks

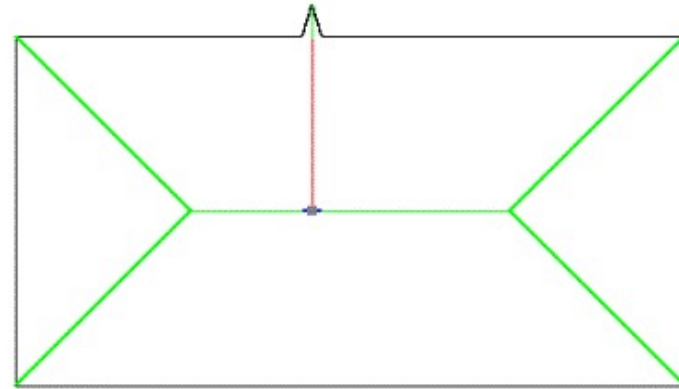
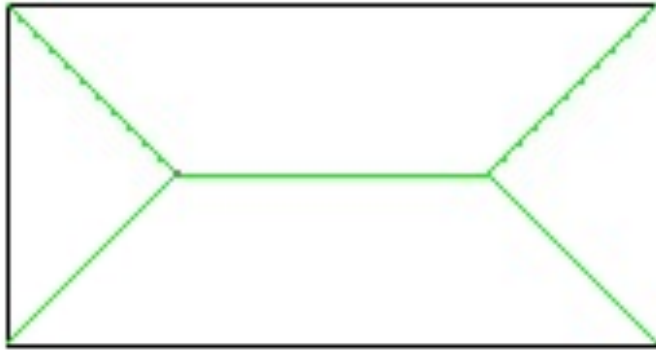
again reduces a 2d (or higher) problem to a question about graphs...



CSCE-574 Robotics



Problems



The skeleton is sensitive to small changes in the object's boundary.

- graph isomorphism (and lots of other graph questions) : NP-complete



Roadmap problems

If an obstacle decides to roll away... (or wasn't there to begin with)

