COMSM4111 Robotic Systems

Motion and Navigation

Walterio Mayol-Cuevas

Navigation

- As simple as

  "To go from one place to another"

- Underpinned by knowledge of world, sensors and drive mechanics.

- For simplicity we will see examples of 2D robots moving on a plane. Most algorithms can be generalized to 3D.

Motion planning

From robot's POV, world can be seen as the union of Free Space and Obstacles

Formally:

Robot's workspace W

– 2D or 3D depending on the robot

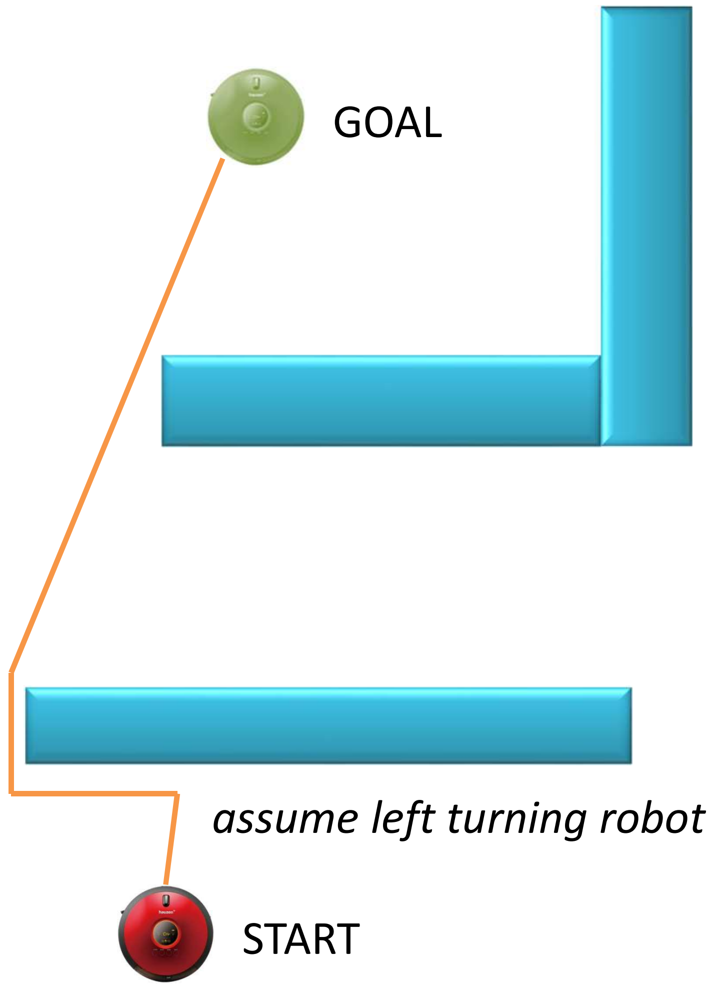
– could be infinite (open) or bounded (closed/compact)

- Obstacle WO_i → eg spacecraft.
- Free workspace $W_{free} = W \setminus \bigcup_i WO_i$

union

Simple "Bug" motion algorithms

Vladimir Lumelsky & Alexander Stepanov: Algorithmica 1987



Bug "0"

•No need to know whole map in advance, only position of goal relative to start.

•Uses local sensing to negotiate obstacles encountered.

STEPS:

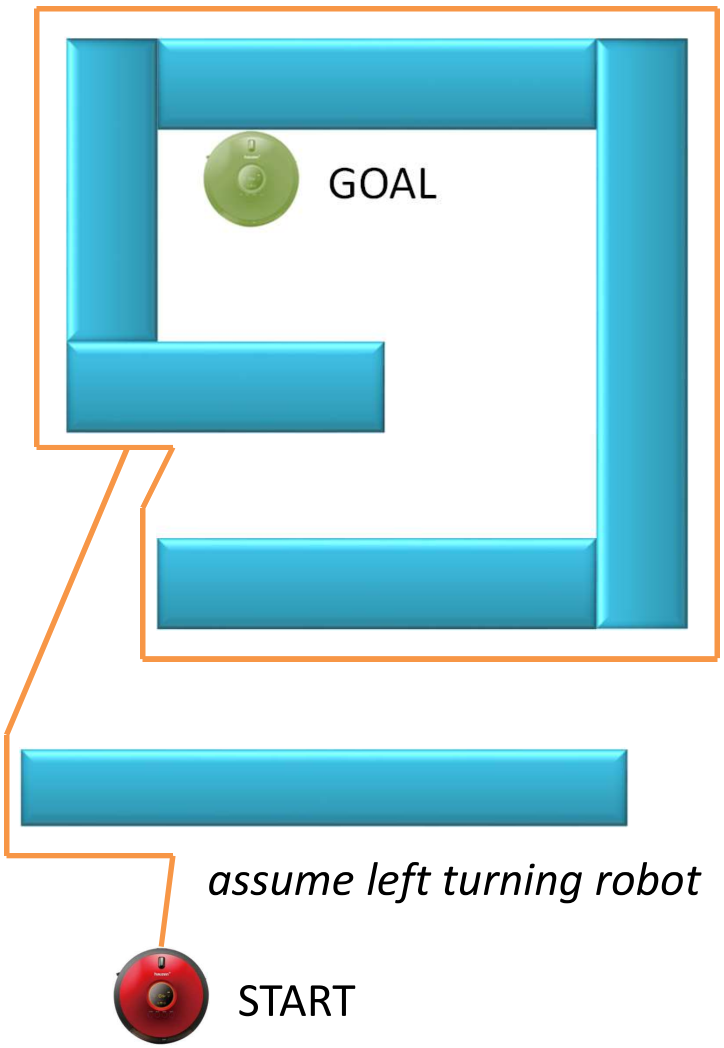
1) head toward goal

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

3) Repeat till goal reached

Adapted from slides from G.D. Hager jhu.edu

Simple "Bug" motion algorithms

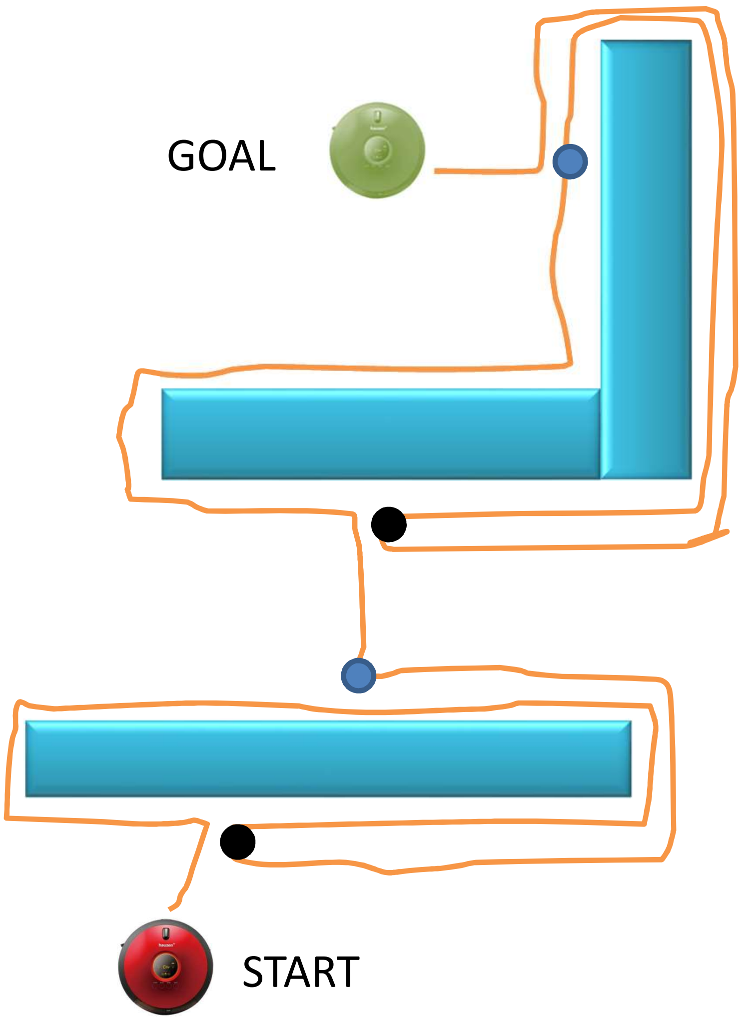


Bug "0"

- Breaks in some cases

Adapted from slides from G.D. Hager jhu.edu

Slightly better



Bug "1"

STEPS:

1) head toward goal

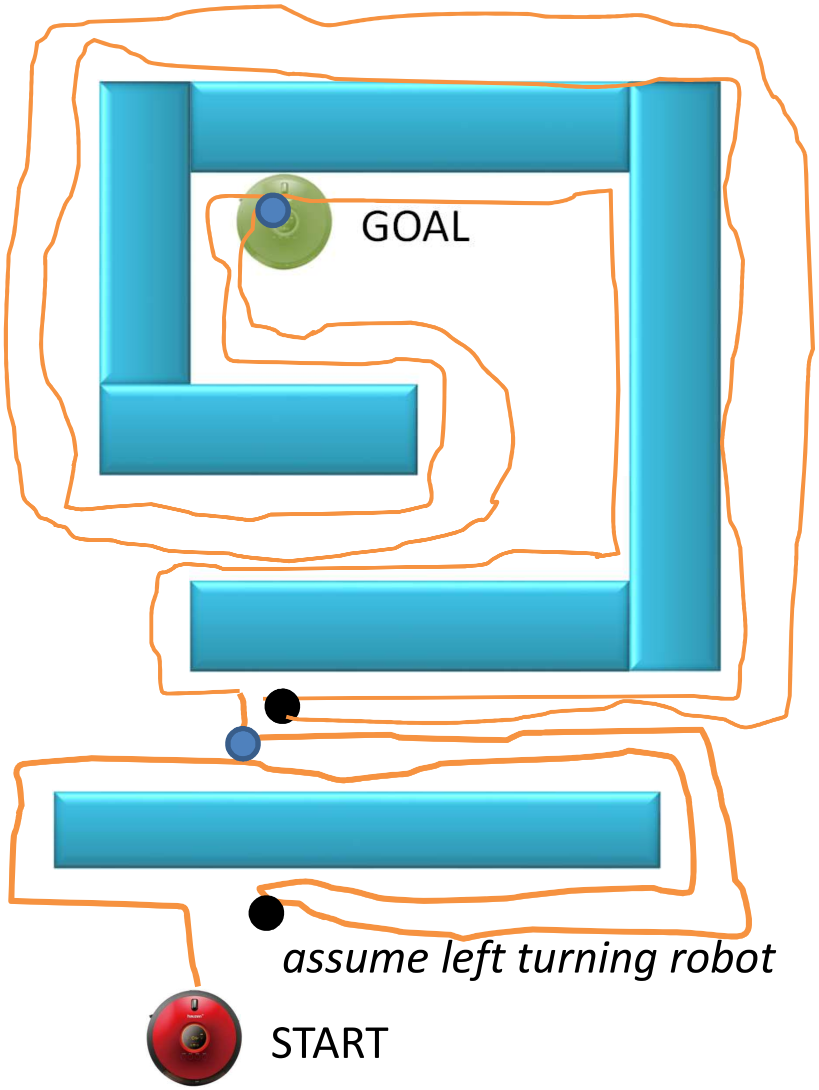
2) if an obstacle is encountered, circumnavigate it *and remember* Where you got closer to the goal and turn where obstacle was encountered

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

This is an exhaustive search
(looks at all options before committing)

Adapted from slides from G.D. Hager jhu.edu

Is it better ?

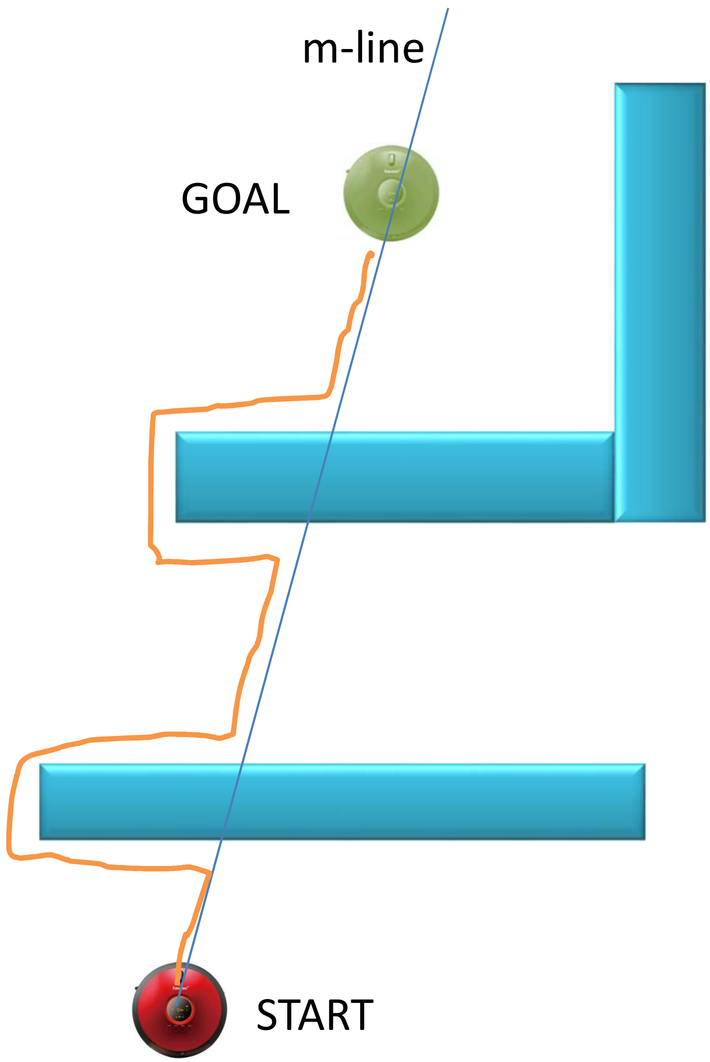


Bug "1"

- Solve this map compared to Bug "0"
- Lots of traversing

Adapted from slides from G.D. Hager jhu.edu

Improved(?) variation



Bug “2”

STEPS:

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

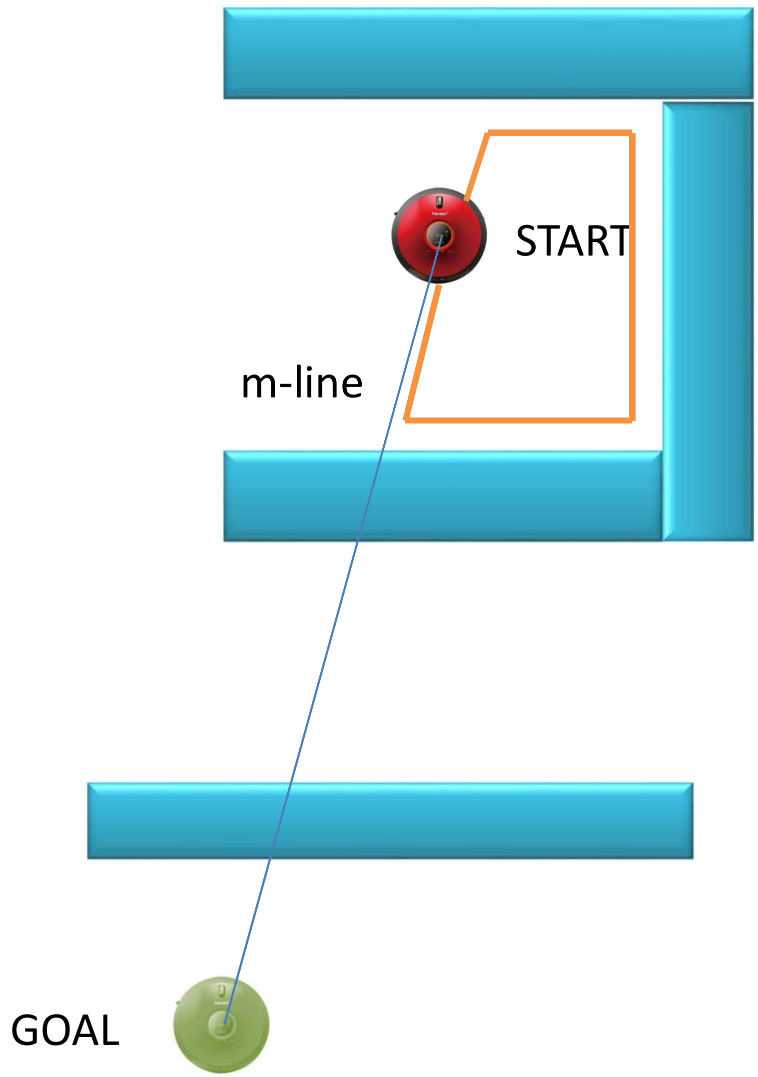
2) if an obstacle is in the way, follow it until you encounter the m-line again ***closer to the goal.***

3) Leave the obstacle and continue toward the goal

This is a greedy algorithm
(takes first thing that looks good)

Adapted from slides from G.D. Hager jhu.edu

A dumber Bug "2"



Bug "2"

STEPS:

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

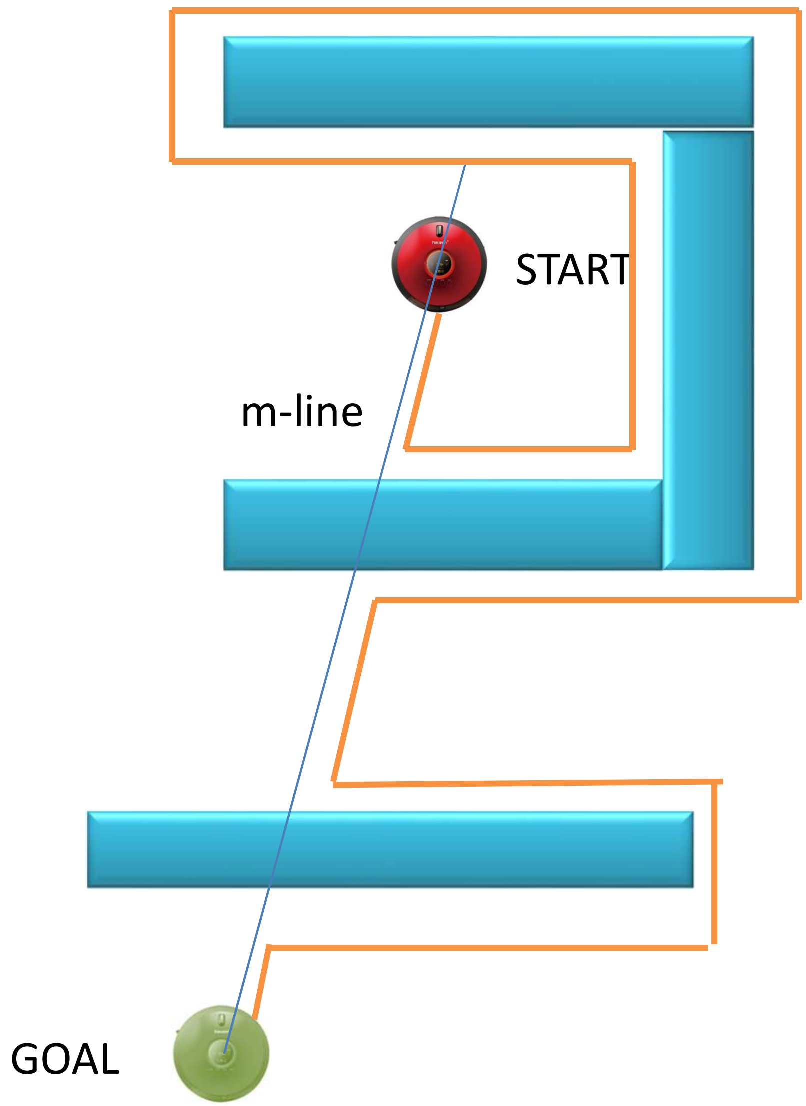
2) if an obstacle is in the way, follow it until you encounter the m-line again.

3) Leave the obstacle and continue toward the goal

This is a greedy algorithm (takes first thing that looks good)

Adapted from slides from G.D. Hager jhu.edu

Bug "2"



Bug "2"

STEPS:

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

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

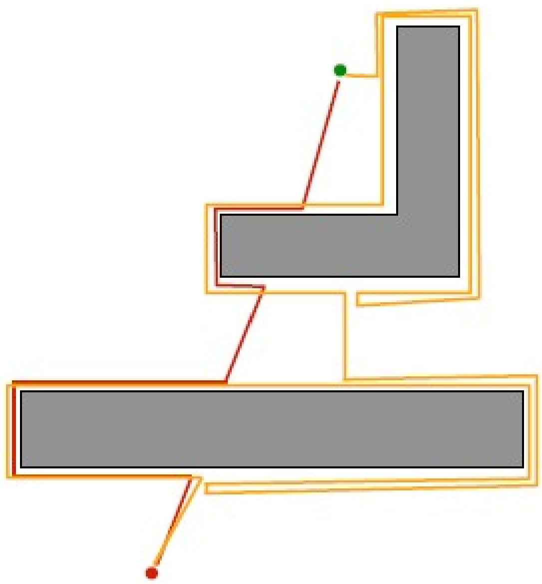
3) Leave the obstacle and continue toward the goal

This is a greedy algorithm
(takes first thing that looks good)

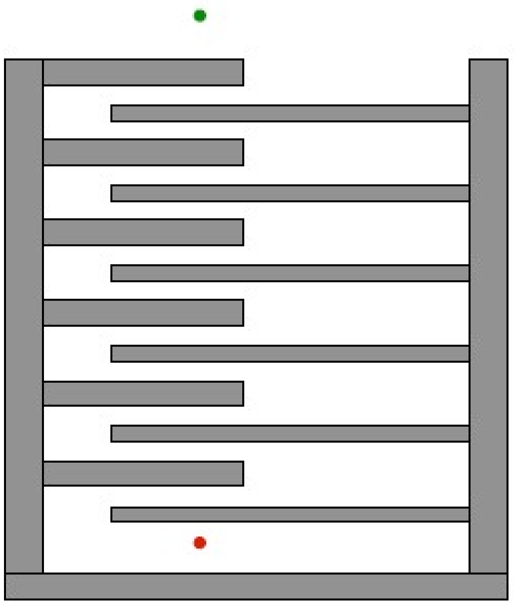
Adapted from slides from G.D. Hager jhu.edu

“insect-inspired” have issues

Bug 2 better than Bug 1



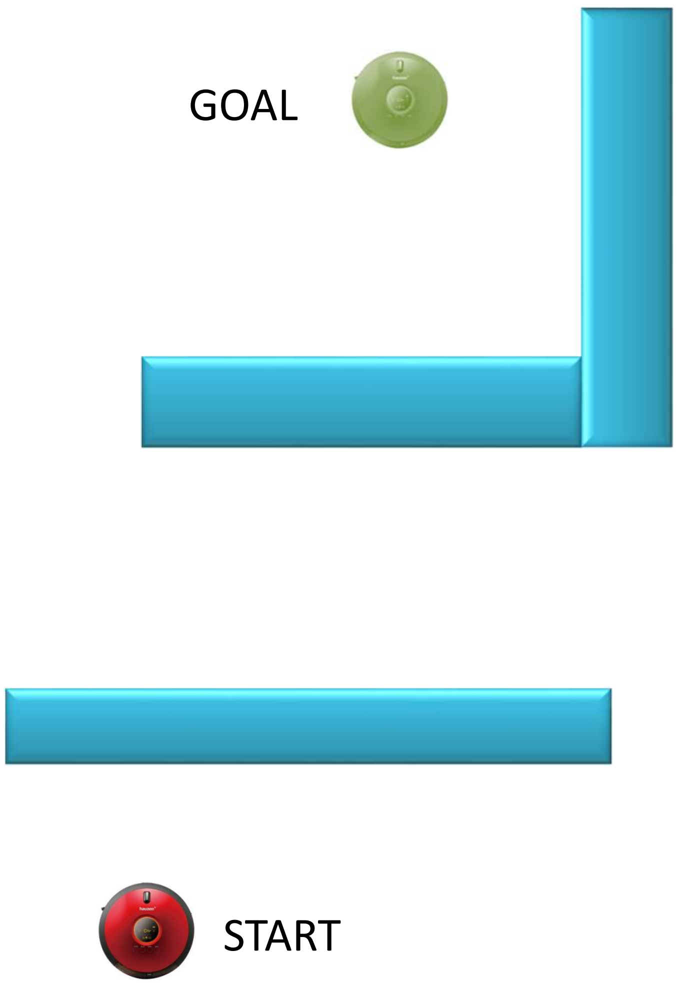
Bug 1 better than Bug 2



Adapted from slides from G.D. Hager jhu.edu

Configuration space

↳ like workspace.



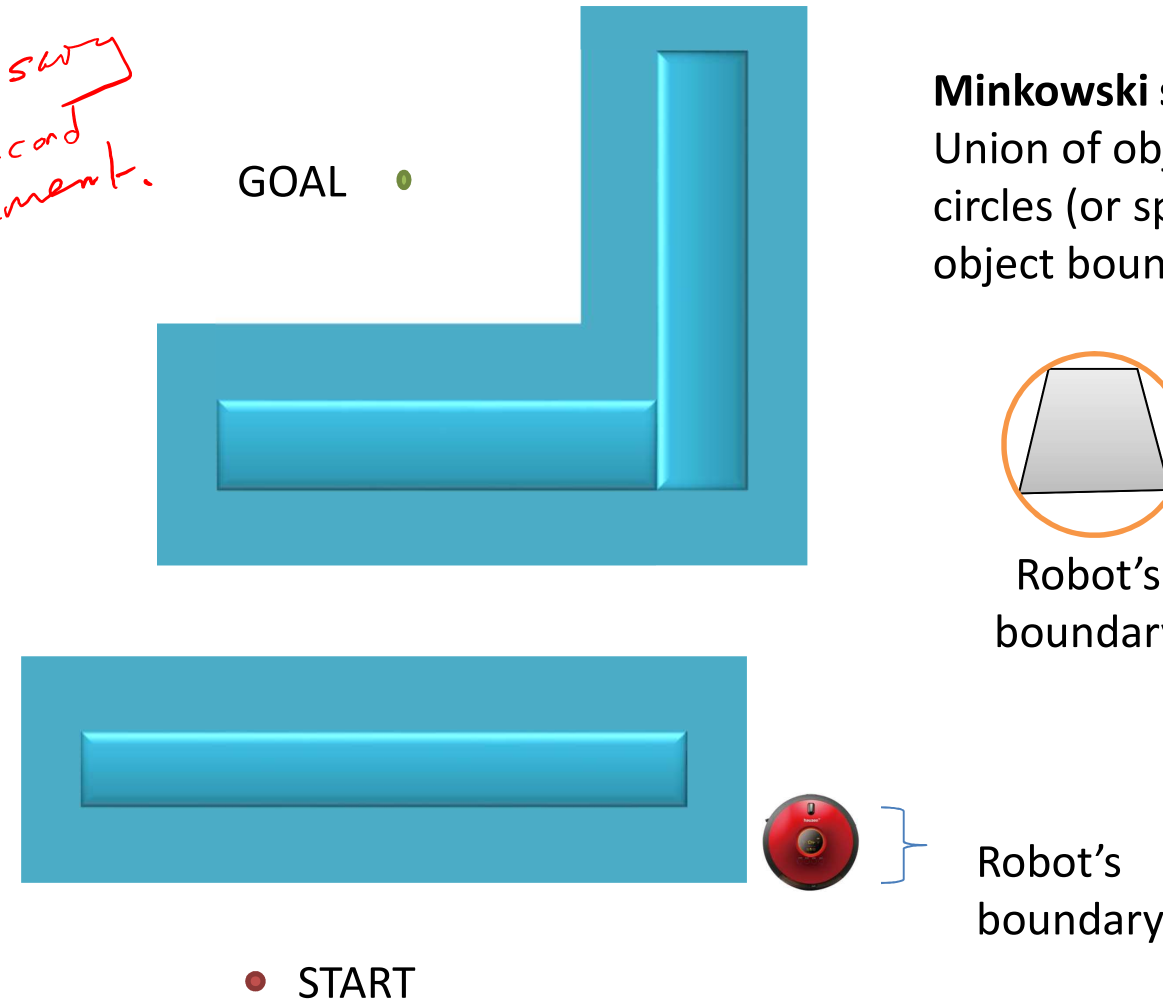
An alternative representation:

- Represent all the allowed possible configurations of the robots.
- Simplifying the robot geometry to a single point in space.
- Simplify calculations and make path planning easier.

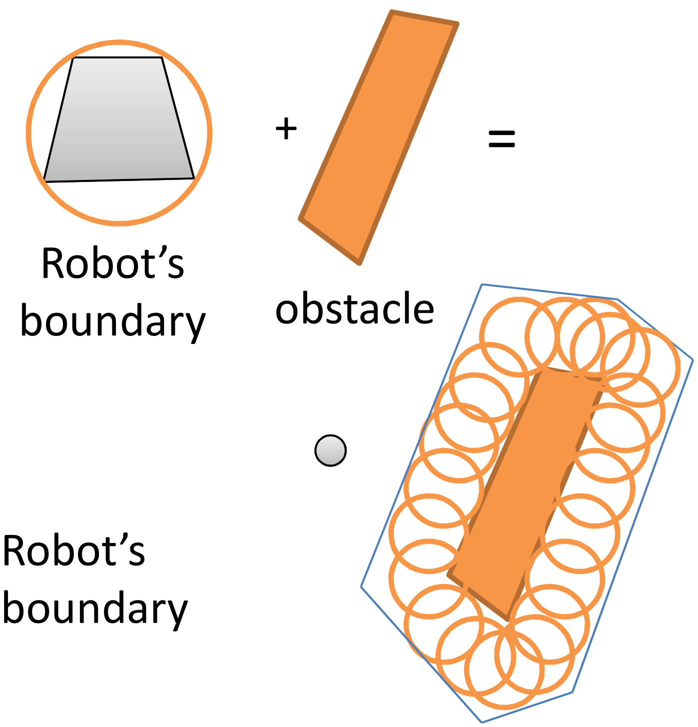
Configuration space

Some planners use configuration space to treat robot as a point to simplify calculations:

Necessary for second assignment.



Minkowski sum:
Union of object shape with set of circles (or spheres) at every object boundary:



Wavefront planner

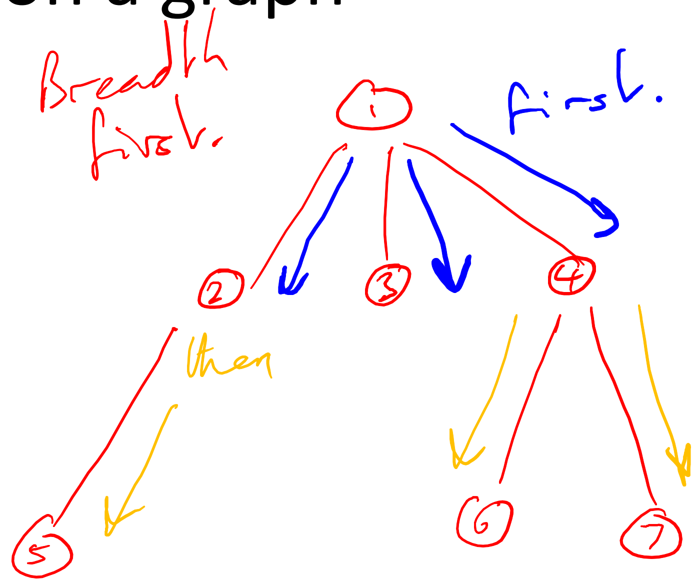
- Assumes we now know the whole map including free-space and obstacles.
- Use inflated configuration space.
- Essentially a breadth-first search on a graph algorithm.
- Setup:

  (Using a grid to simplify explanation)

  – Label Goal cell with a 2

  – Label free space cells with 0

  – Obstacle cells are not labelled



Wavefront: setup

S	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						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	0	0	0	2

- One can use distances without cells
- Here we assume distance to 8-neighbouring cells is equal

Wavefront in action

Starting from goal, set 8 non-touched neighbouring cells to n+1

S	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						6	6
0	0	0						5	5
0	0	0		0	6	5	4	4	4
0	0	0	0	0	6	5	4	3	3
0	0	0	0	0	6	5	4	3	2

Wavefront in action

Starting from goal, set 8 non-touched neighbouring cells to n+1

S	0	0	0	0	9	9	9	9	9
0	0	0	0	0	9	8	8	8	8
0	0	0	0	0	9	8	7	7	7
0	0	0						6	6
0	0	0						5	5
0	0	9		7	6	5	4	4	4
0	0	9	8	7	6	5	4	3	3
0	0	9	8	7	6	5	4	3	2

Wavefront in action

Starting from goal, set 8 non-touched neighbouring cells to n+1

S	13	12	11	10	9	9	9	9	9
13	13	12	11	10	9	8	8	8	8
12	12	12	11	10	9	8	7	7	7
11	11	11						6	6
11	10	10						5	5
11	10	9		7	6	5	4	4	4
11	10	9	8	7	6	5	4	3	3
11	10	9	8	7	6	5	4	3	2

Zeros will remain if a location is unreachable

Wavefront in action

From start simply move to cell with smaller value till the goal is reached

S	13	12	11	10	9	9	9	9	9
13	13	12	11	10	9	8	8	8	8
12	12	12	11	10	9	8	7	7	7
11	11	11						6	6
11	10	10						5	5
11	10	9		7	6	5	4	4	4
11	10	9	8	7	6	5	4	3	3
11	10	9	8	7	6	5	4	3	2

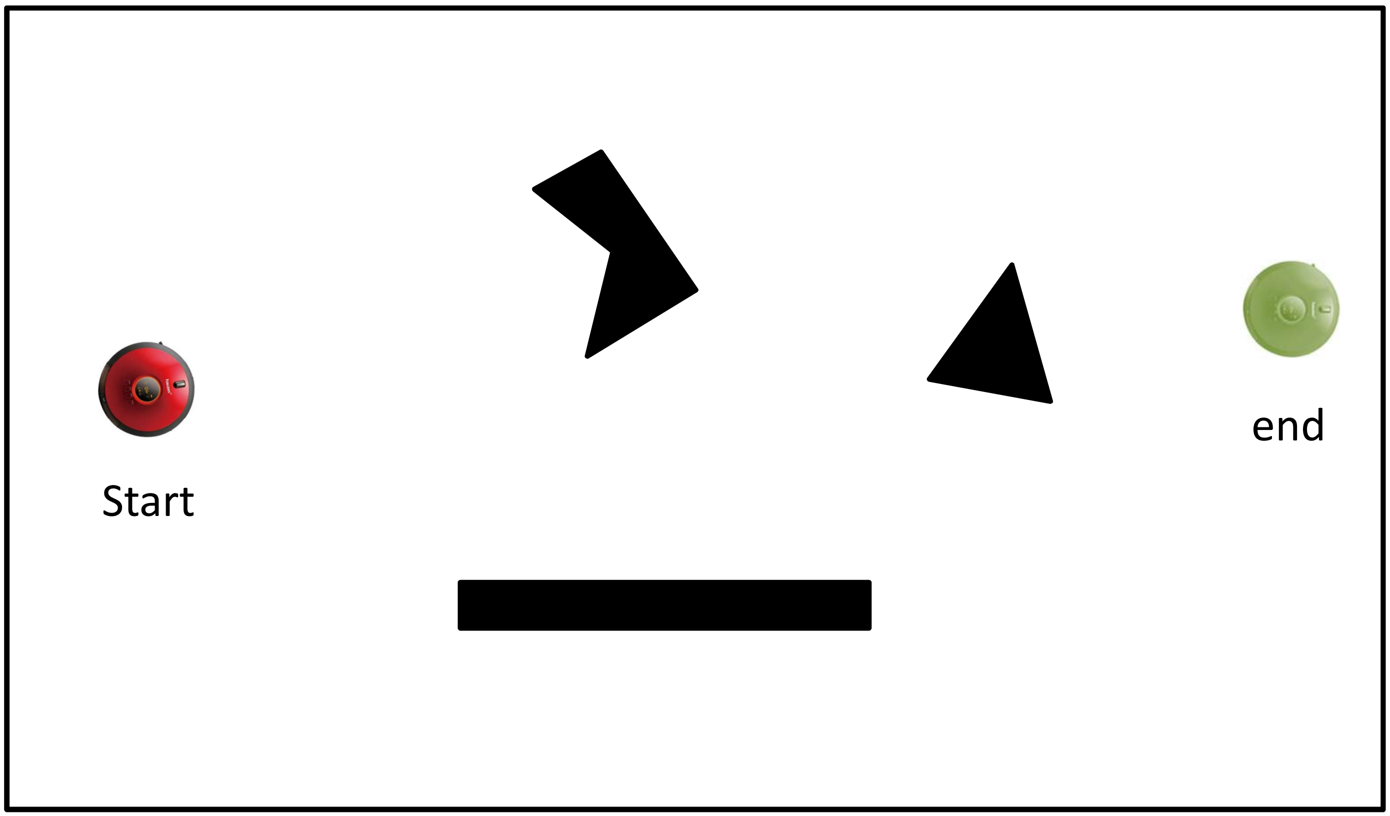
There might be several shortest paths, here we show 2 possible ones

Visibility graphs

- Formed by connecting all "visible" vertices, the start and the goal points.
- For two vertices to be "visible" no obstacle can exist between them. Paths exist on the perimeter of obstacles.
- Use inflated configuration space.
- May be used without full knowledge of the map, i.e. Can be computed as the robot explores the scene. Must still know where the goal is relative to the starting point.

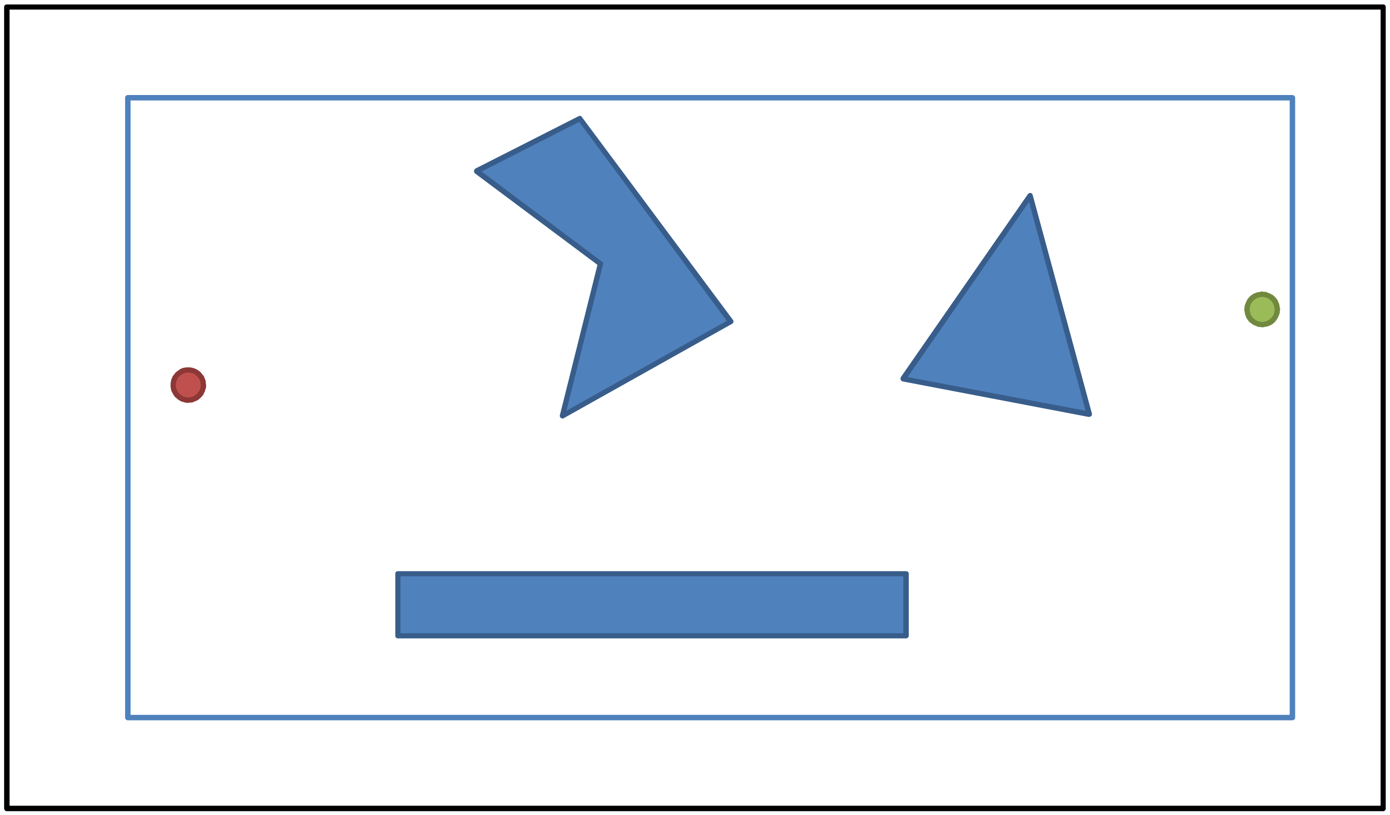
Visibility graphs

Trace visibility between vertices from start and goal



Visibility graphs

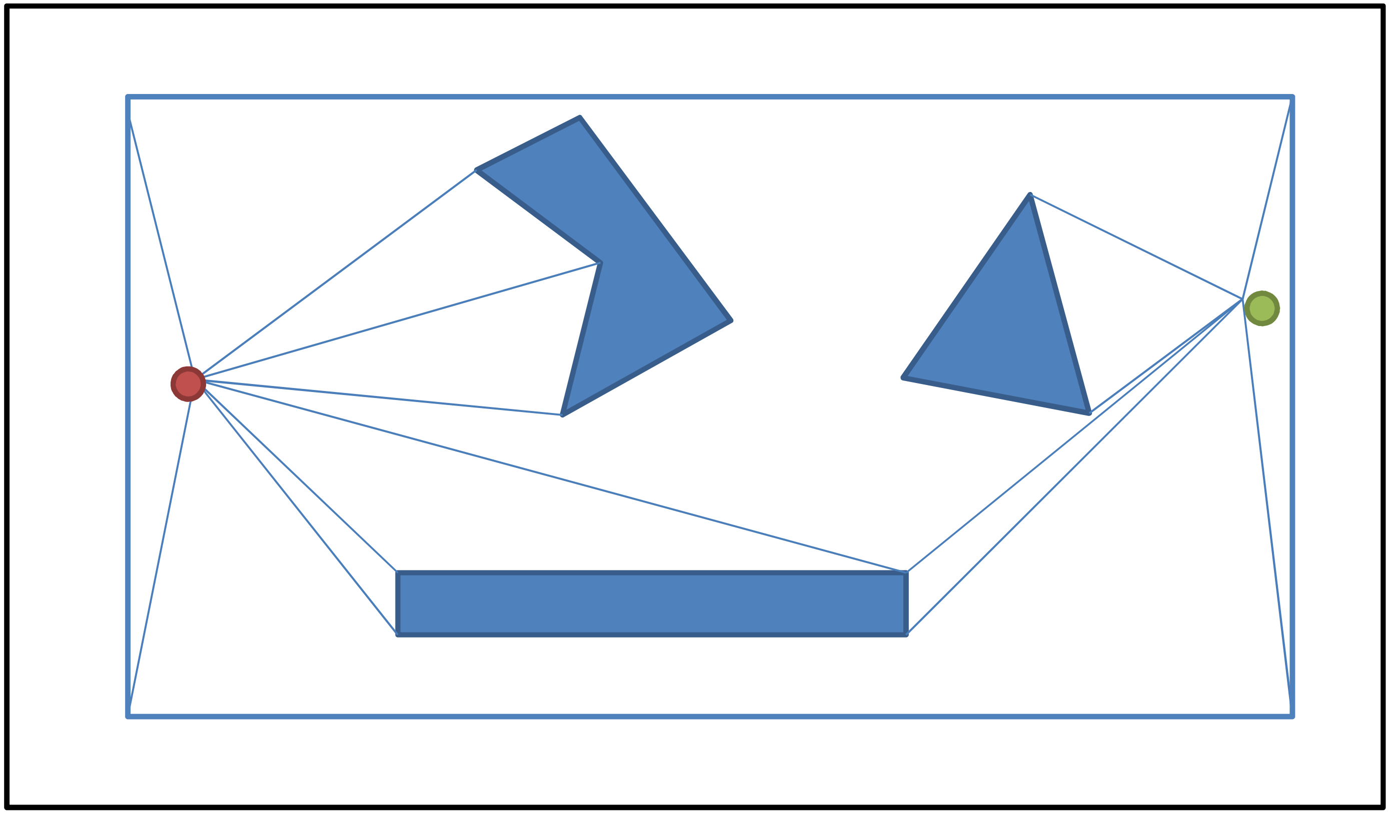
Trace visibility between vertices from start and goal



Use Configuration Space!

Visibility graphs

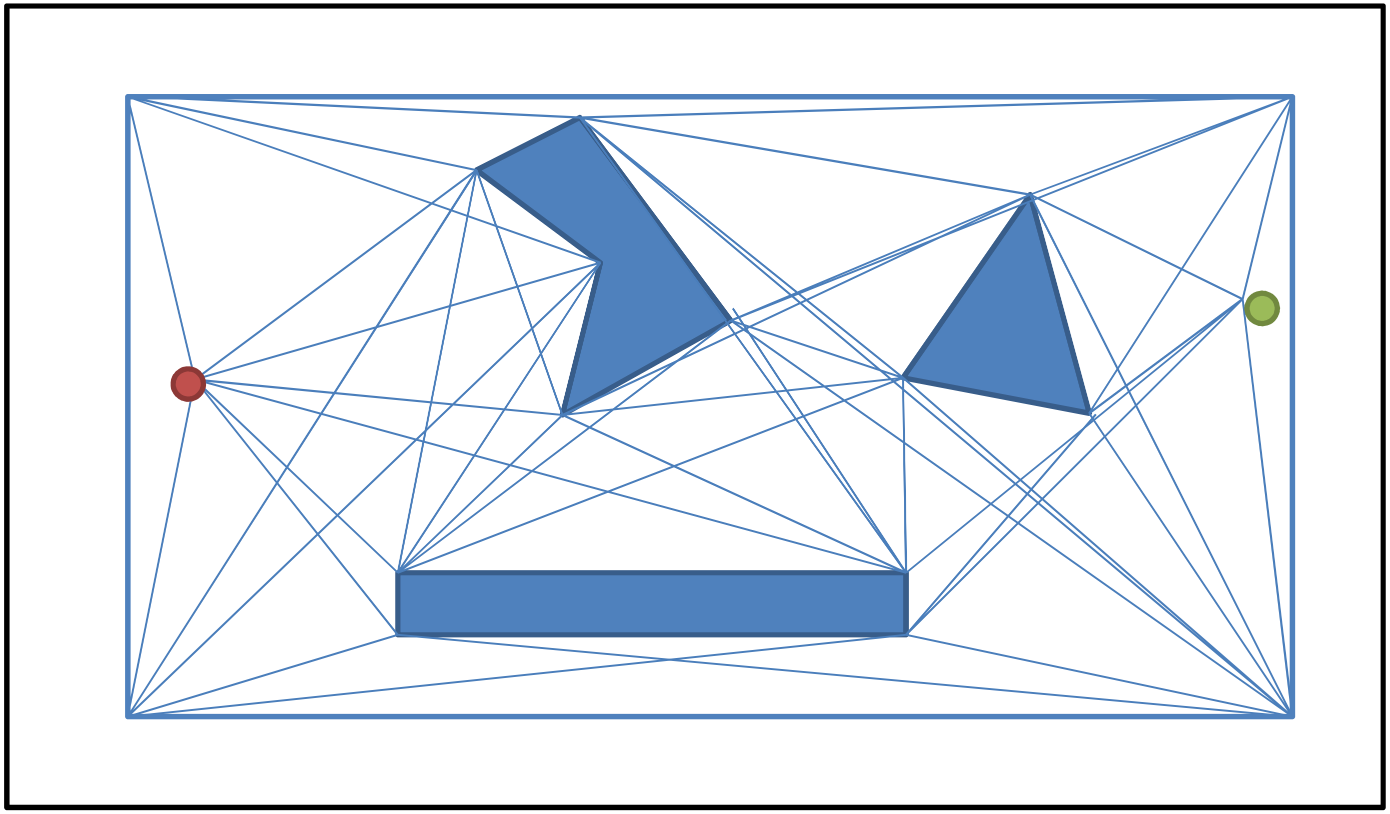
Trace visibility between vertices from start and goal



Draw connections between vertices!

Visibility graphs

Trace visibility between all other vertices



Calculate shortest path. Careful because here the path takes robot close to obstacles
Note: visibility along an edge is also valid.