# Robot Navigation and collision avoidance

# OutLine

- Introduction
- Mapping
- Navigation
- Collision avoidance

# Introduction: What navigation means ?

“ The *Process of directing a vehicle so as to reach the intended destination* ”

IEEE Standard 172-1983

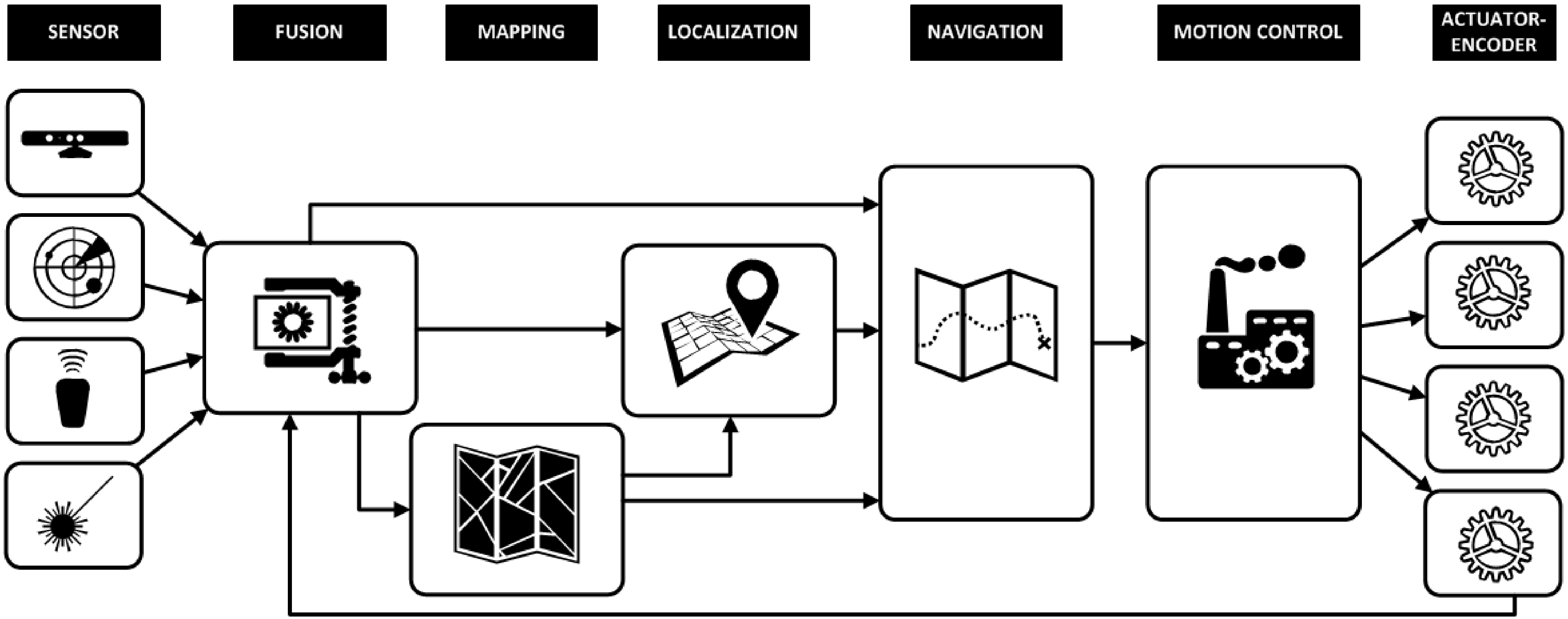
“ *Given partial knowledge about its environment and a goal position or a series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensors values so as to reach its goal positions as efficiently and reliably as possible* ”

Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

“ *Robot navigation is the problem of guiding a robot towards a goal*”

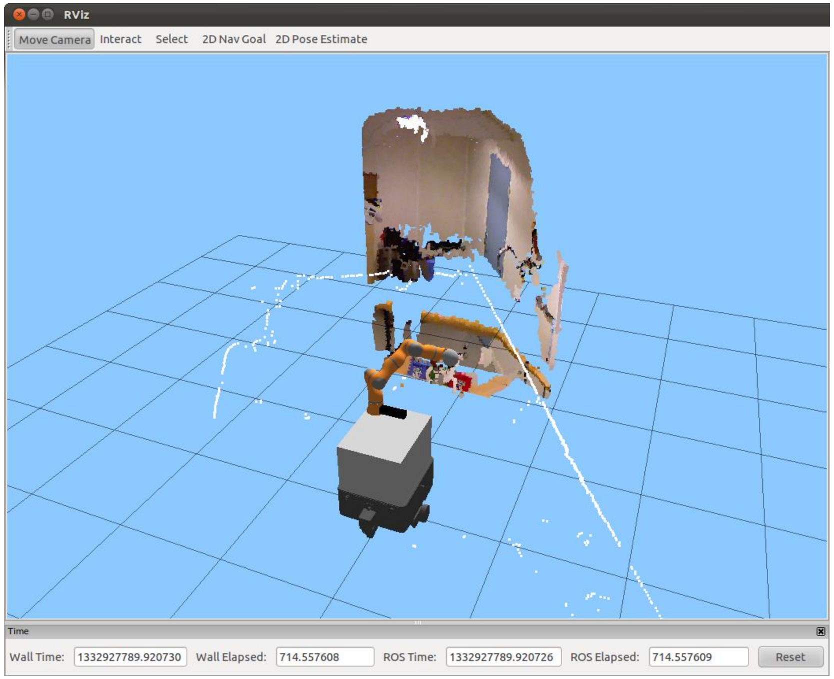
Robotics, Vision and Control, Springer, Peter Corke 2011

# What navigation means ?



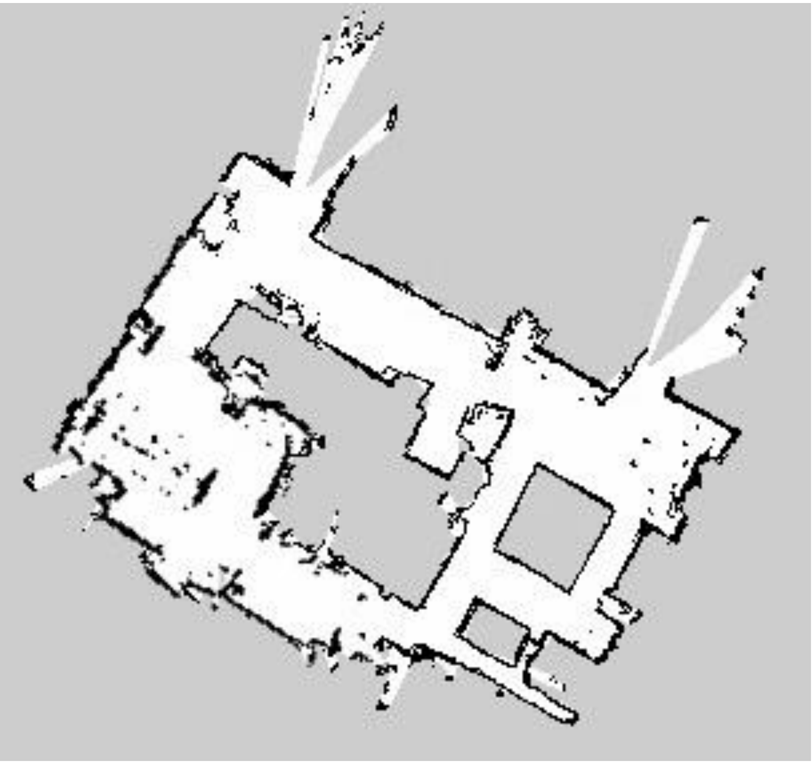
# Data Fusion

- ❑ Collecting all data from sensors
- ❑ Transform data into common languages
- ❑ Merging data
  - Convert into same geometric standard
  - Clean data
  - Merge information into a common representation



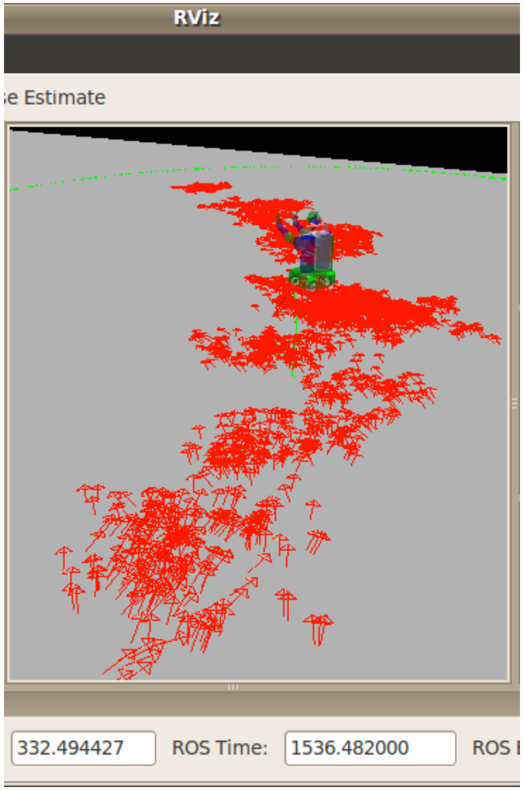
# Mapping

- Collecting all merged data
- Build a cumulative representation of data
- Express the environment obstacle world into a unique robot readable data



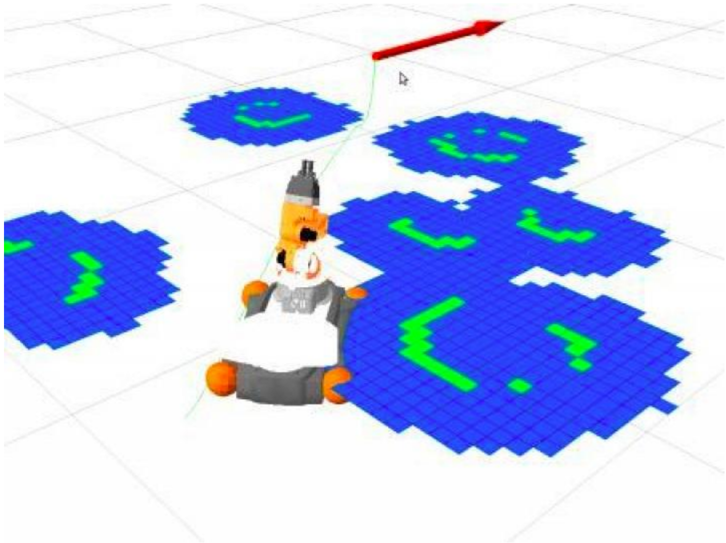
# Localization

- ❑ Collect sensors data
- ❑ Collect encoders data
- ❑ Process all data regarding a given map
- ❑ Express one or many robot position estimations



# Navigation

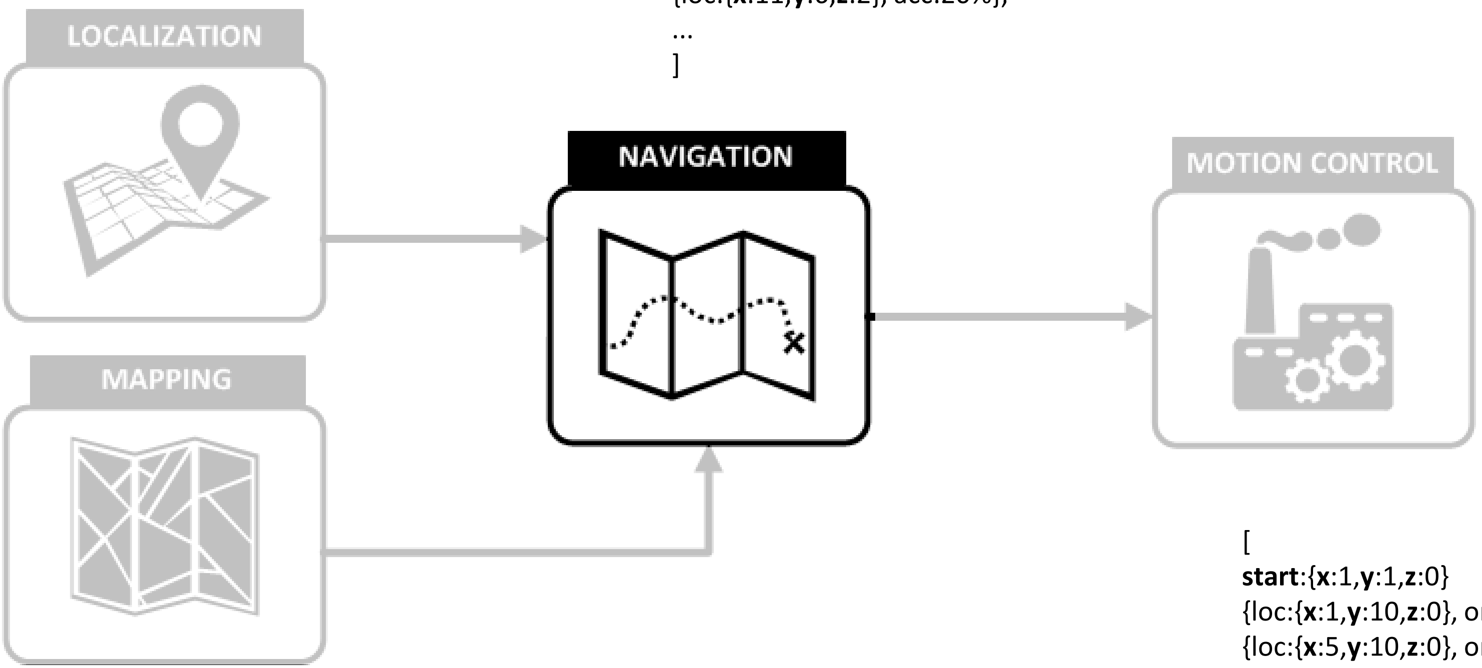
- Collect one or many robot position estimation
- Use the map as obstacle estimator
- Compute path form estimate position to a targeted position
- Re-plan or react in case of new or dynamic obstacles observation



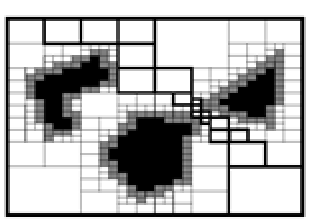
# Navigation – overview –

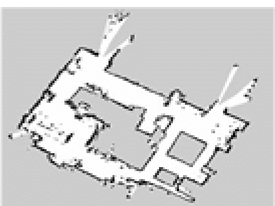
{x:10,y:5,z:2} 80%   OR

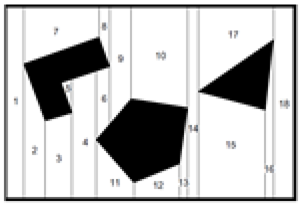
[
{loc:{**x**:10,**y**:5,**z**:2}, acc:60%},
{loc:{**x**:5,**y**:50,**z**:20}, acc:1%},
{loc:{**x**:1,**y**:1,**z**:14}, acc:5%},
{loc:{**x**:11,**y**:6,**z**:2}, acc:20%},
...
]




[
**start**:{**x**:1,**y**:1,**z**:0}
{loc:{**x**:1,**y**:10,**z**:0}, order:1},
{loc:{**x**:5,**y**:10,**z**:0}, order:2},
{loc:{**x**:7,**y**:12,**z**:0}, order:3},
{loc:{**x**:9,**y**:14,**z**:0}, order:4},
**goal**:{**x**:1,**y**:1,**z**:0}
]



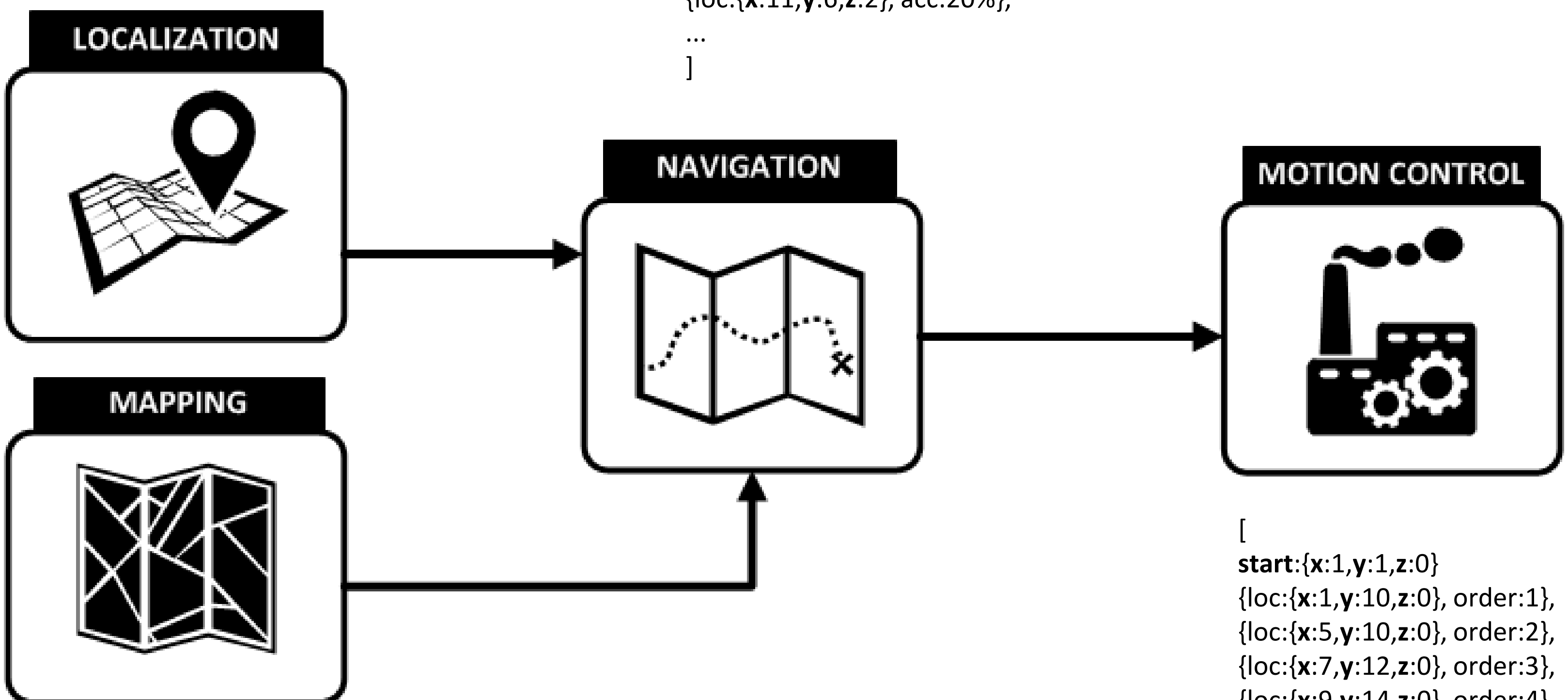




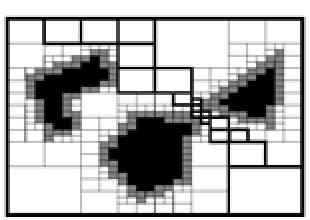
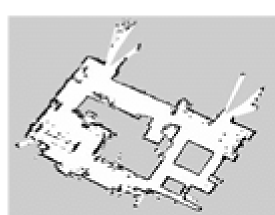
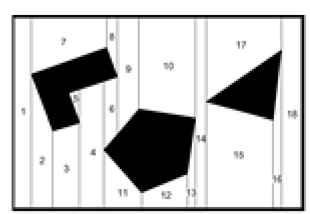
# Navigation – overview –

{x:10,y:5,z:2} 80%  OR

[
{loc:{**x**:10,**y**:5,**z**:2}, acc:60%},
{loc:{**x**:5,**y**:50,**z**:20}, acc:1%},
{loc:{**x**:1,**y**:1,**z**:14}, acc:5%},
{loc:{**x**:11,**y**:6,**z**:2}, acc:20%},
...
]




[
**start**:{**x**:1,**y**:1,**z**:0}
{loc:{**x**:1,**y**:10,**z**:0}, order:1},
{loc:{**x**:5,**y**:10,**z**:0}, order:2},
{loc:{**x**:7,**y**:12,**z**:0}, order:3},
{loc:{**x**:9,**y**:14,**z**:0}, order:4},
**goal**:{**x**:1,**y**:1,**z**:0}
]

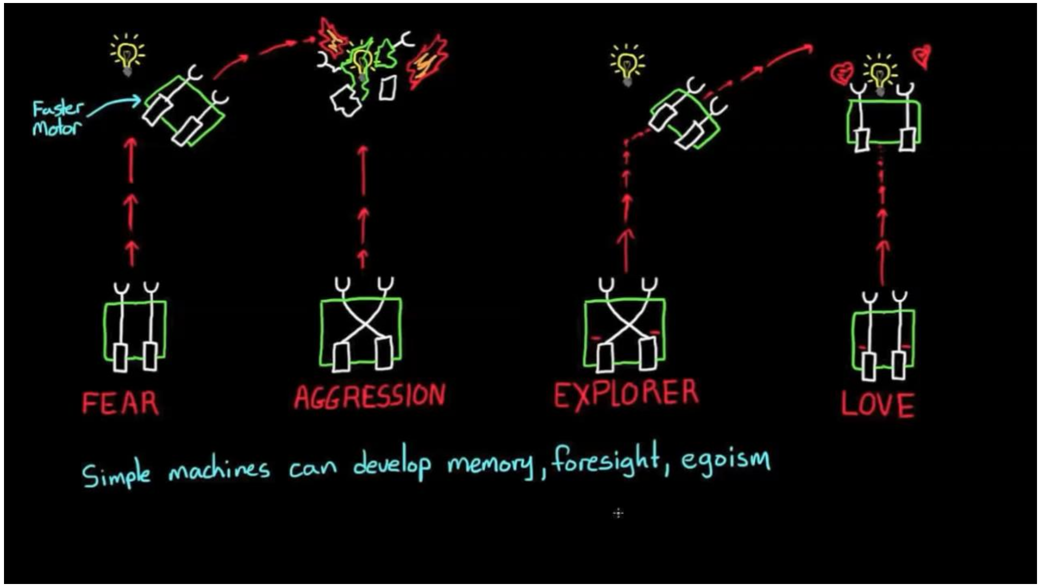




# Navigation – strategies–

- Behavior-Based
  - No Localization
  - External goal
  - e.g : **wall follower**

# Navigation – strategies–

- ❑ Behavior-Based
  - No Localization
  - External goal
- ❑ Reactive-Based
  - No Localization
  - Sensor based goal
  - e.g: **Braitenberg Vehicle**



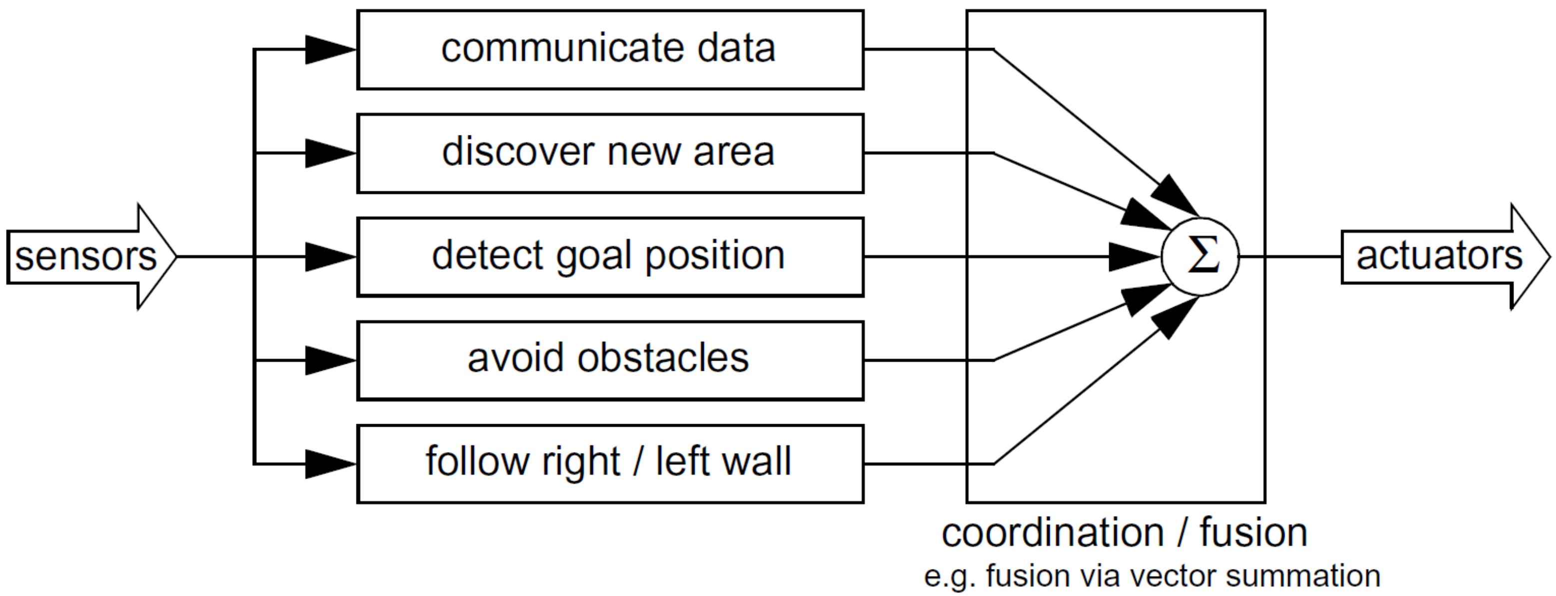

https://www.youtube.com/watch?v=A-fxij3zM7g

# Navigation – strategies–

- Behavior-Based
  - No Localization
  - External goal
- Reactive-Based
  - No Localization
  - Sensor based goal
- Map-Based
  - Localization
  - External goal
  - E.g: **Dynamic A***

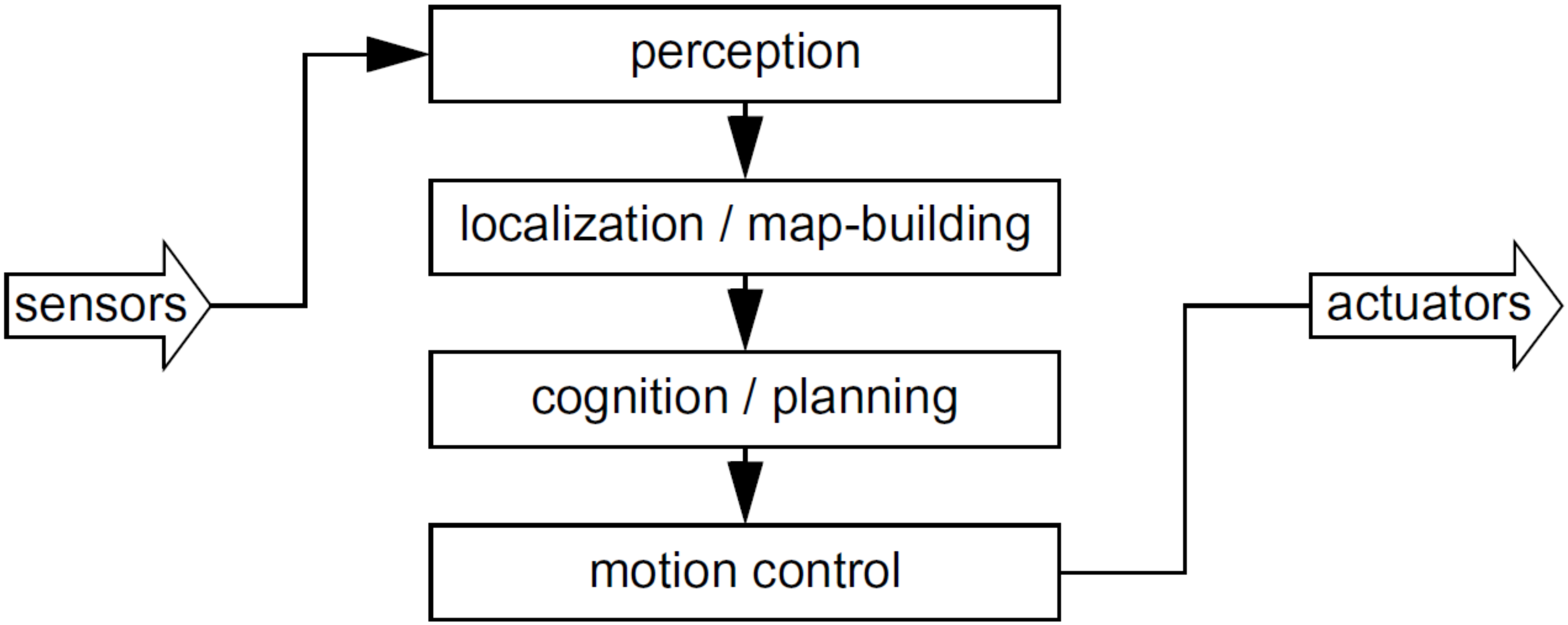
https://www.youtube.com/watch?v=qziUJcUDfBc

# Behavior Based Architecture




Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

# Map-Based Architecture




Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

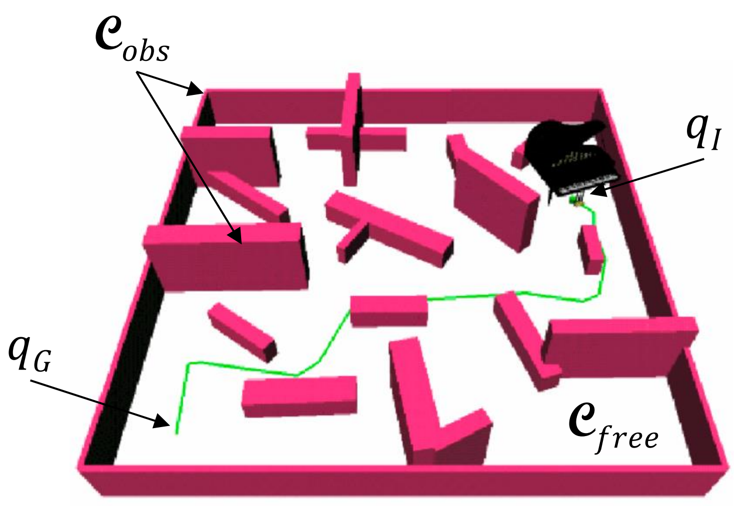
# Mapping

# Overview

- Objectives
  - Put observed data into a standard view (obstacles, objects, robot)
  - Use to estimate the robot position
  - Use to compute a trajectory from a start point to a goal
  - Summarize the collected data

- Map requirement
  - Map accurancy matches  the precision which the **robot needs to achieve a goal**
  - Map accurancy matches the precision of the **precision robot’s sensor**.
  - **Complexity** of the map representation  **has direct** impact on the computational complexity of reasoning about **mapping**, **localization** and **navigation**

# Configuration Space




$\mathcal{C} = \mathcal{C}_{obs} \cup \mathcal{C}_{free}$

$\mathcal{C}$ Set of all possible transformations that may be applied on the robot.

$q_I \in \mathcal{C}_{free}$ Initial configuration

$q_G \in \mathcal{C}_{free}$ Goal configuration

$path\ \tau: [0,1] \rightarrow \mathcal{C}_{free}$

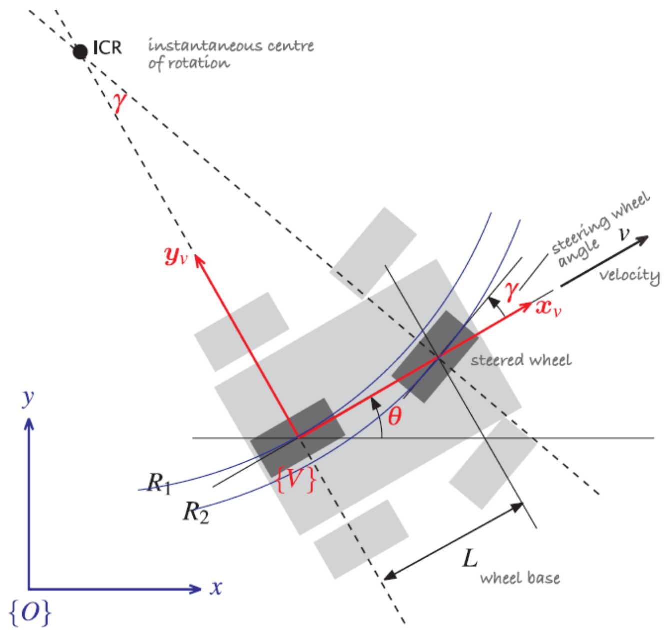
$\tau(0) = q_I$

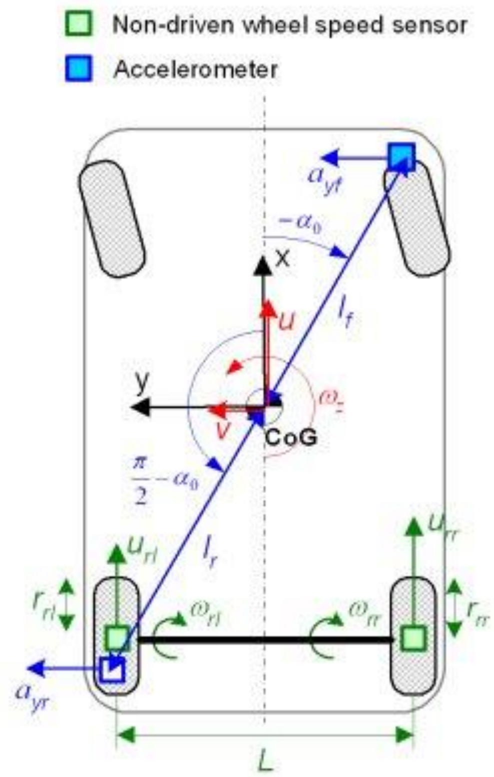
$\tau(1) = q_G$

$\tau = \{q_{I, \dots,} q_{i, \dots,} q_G\}$

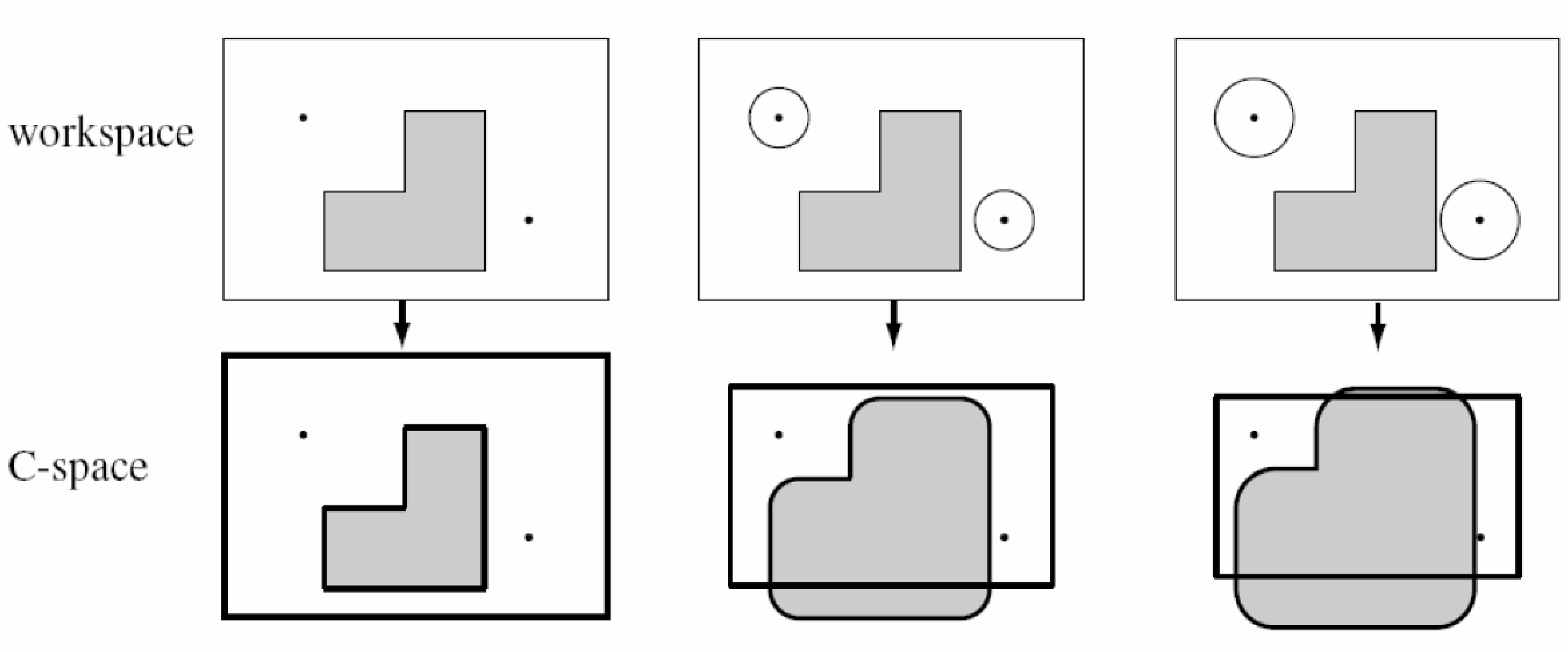
$q = (x, y, \theta)$

# Configuration Space



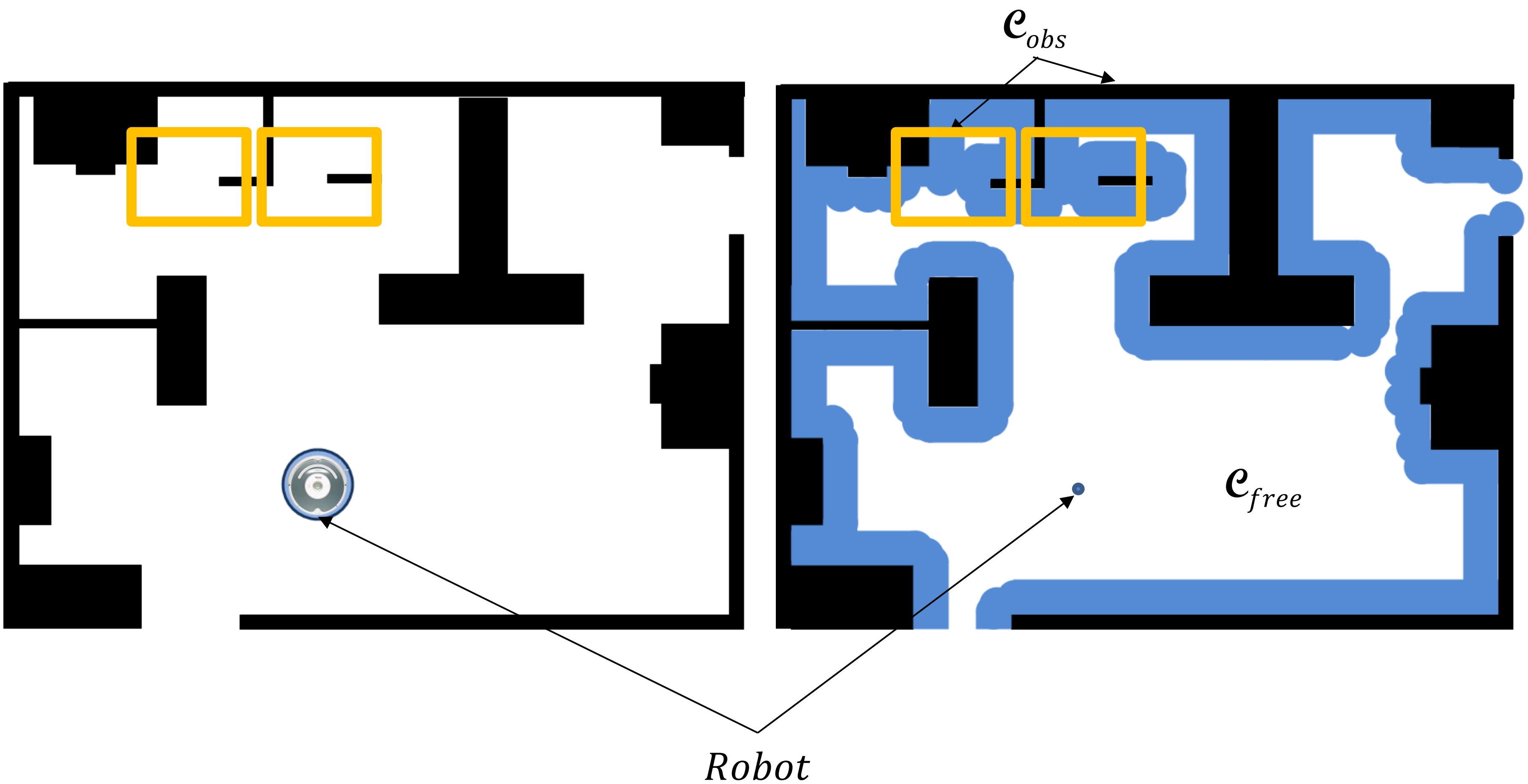



# Configuration Space: Accommodate Robot Size

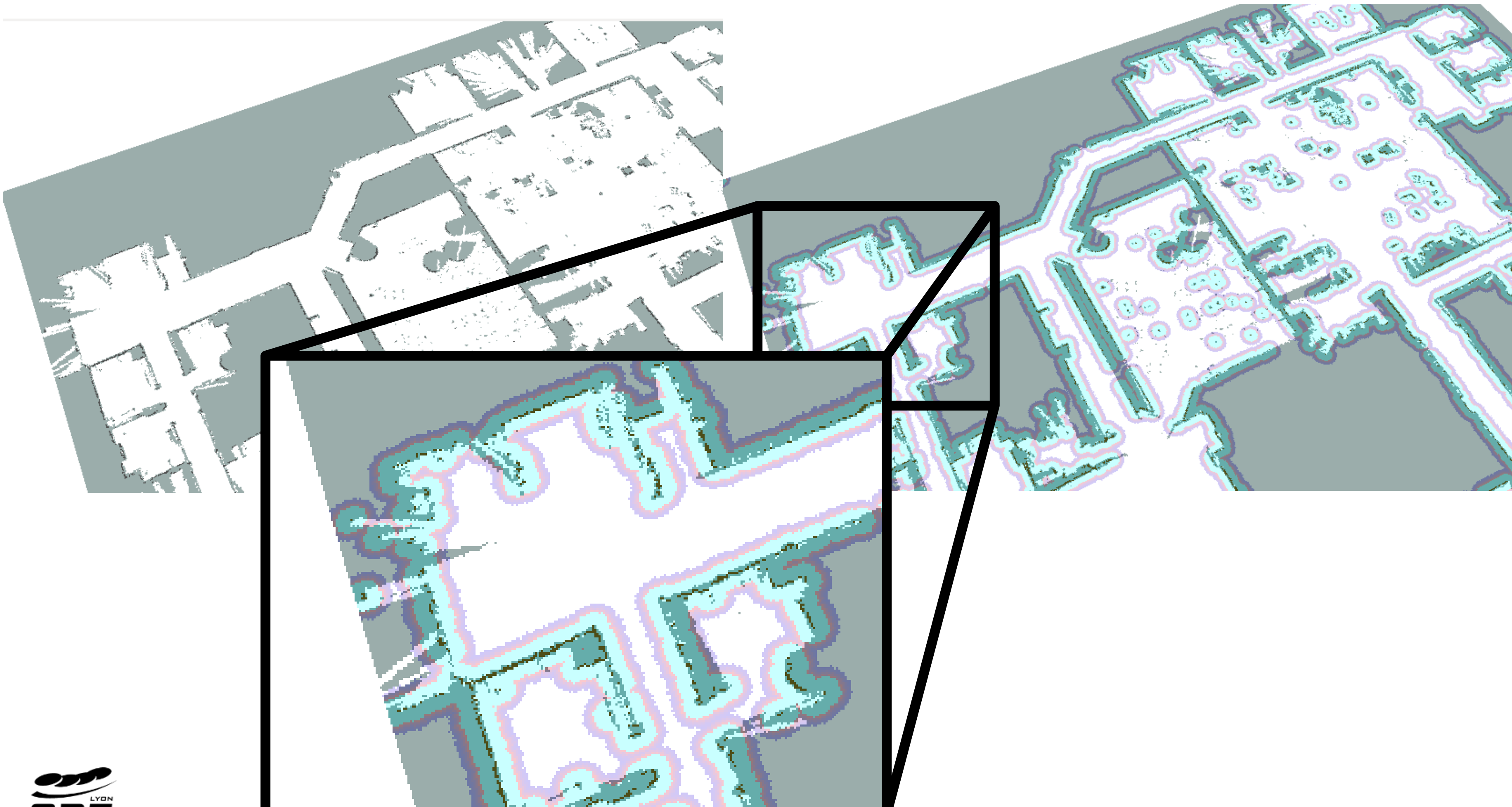



16-735, Howie Choset with slides from G.D. Hager, Z. Dodds, and Dinesh Mocha

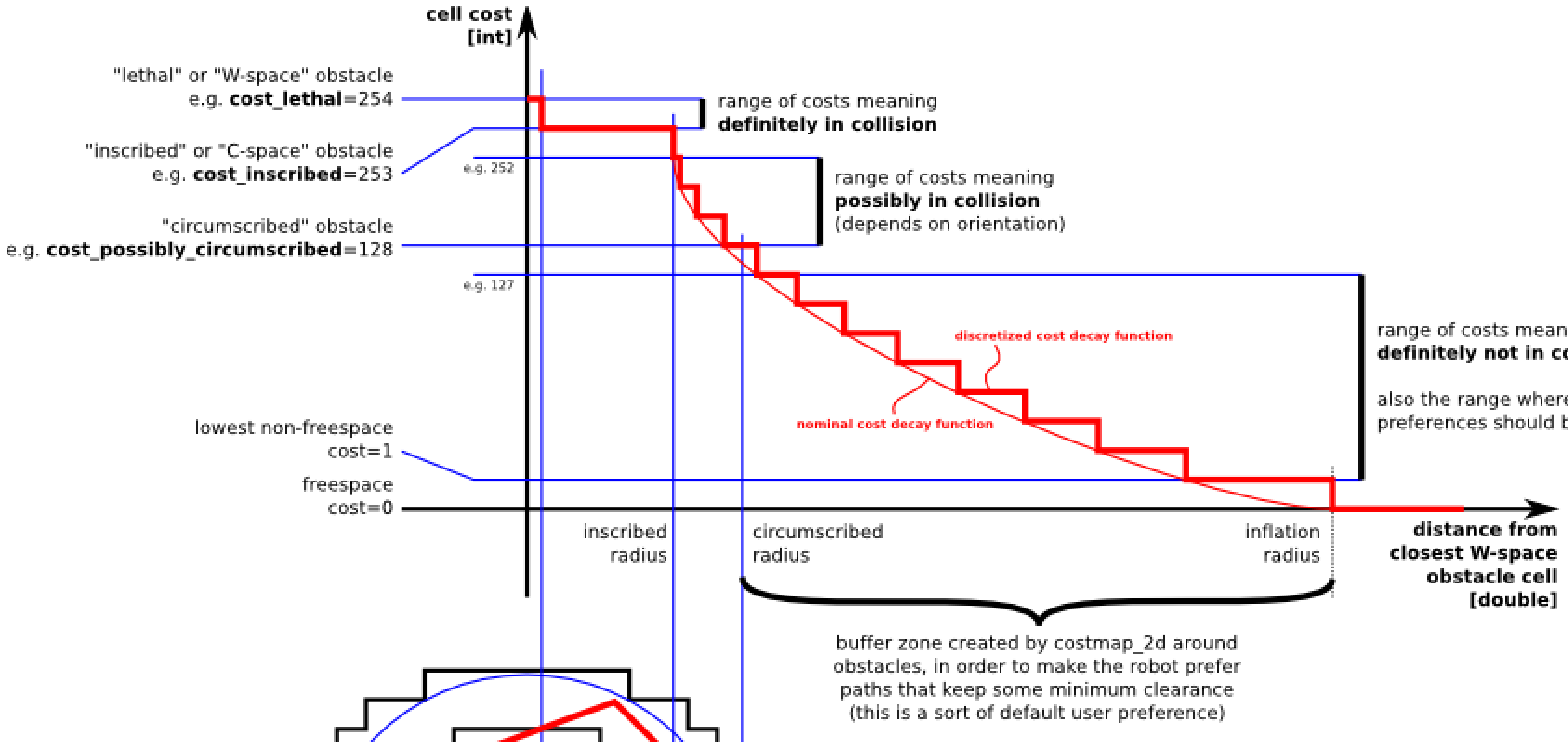
# Configuration Space: Accommodate Robot Size

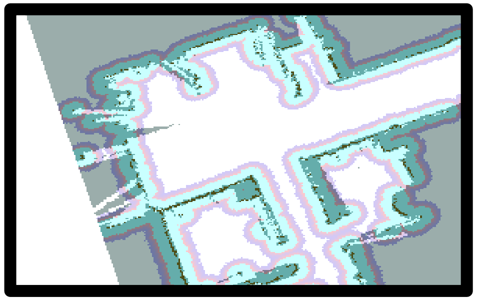


# Configuration Space: Accommodate Robot Size



# Configuration Space: Accommodate Robot Size



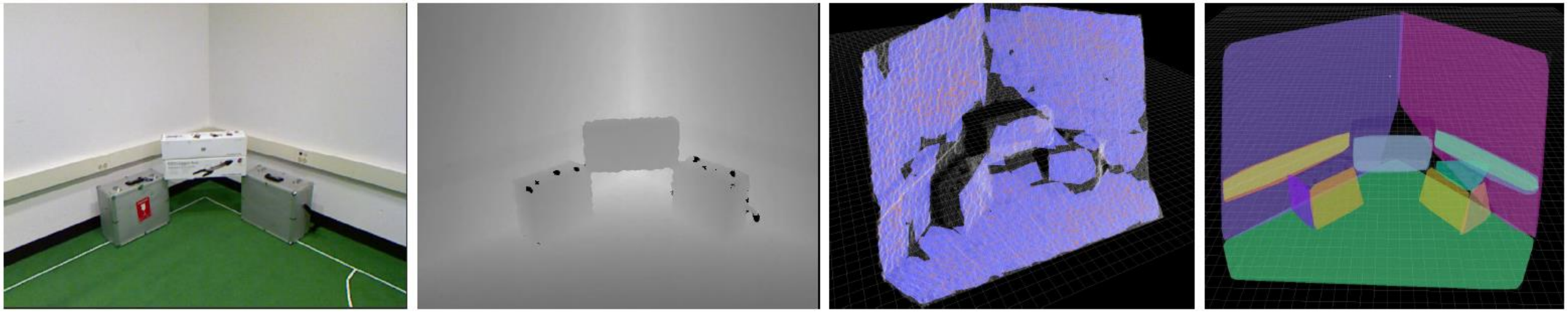
# Map representation

- Continuous
  - All objects in the map are represented
  - Map size depends of the objects density (sparse environment leads to low-memory map)
- Decomposition
  - General decomposition and selection of environment features
  - Loss of fidelity between  map and real environment
  - Capture useful features and discarding other
  - Fixed-decomposition and adaptive decomposition

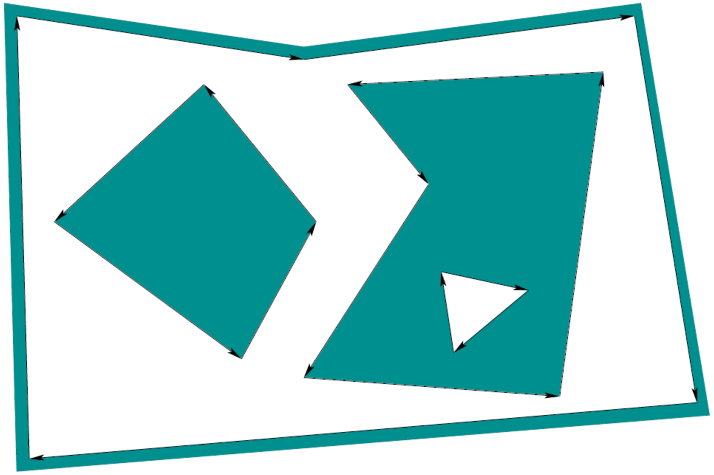
CPE LYON
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

# Continuous representation

- Polygone representation
  - 3D polygone map construction

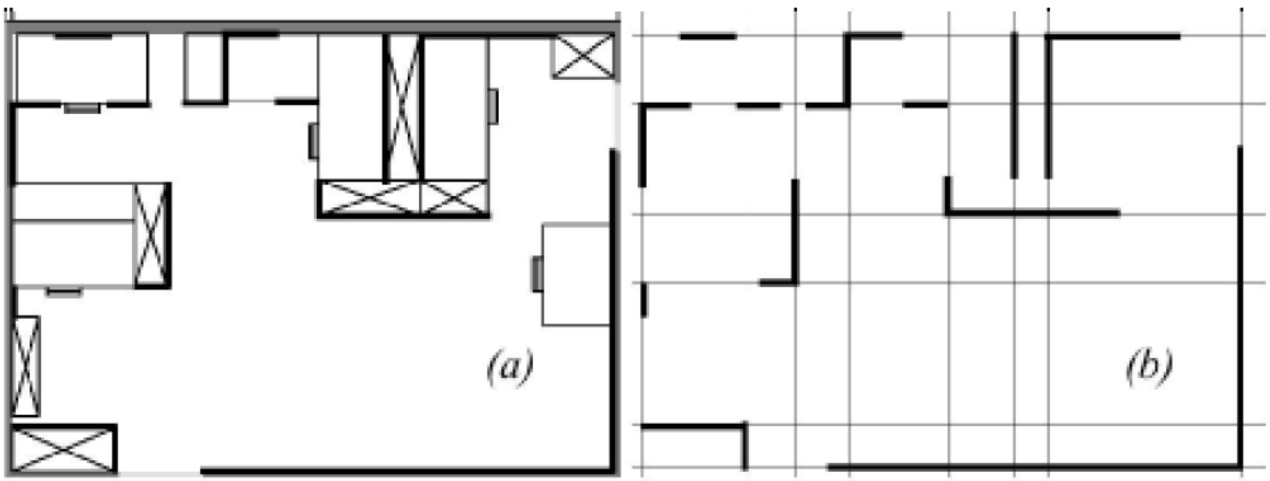


  - 2D polygone map construction



# Continuous representation

- Line representation (EPFL)




  - (a) Real world
  - (b) Representation with a set of infinite lines

Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

# Contineous representation

## + Avantages

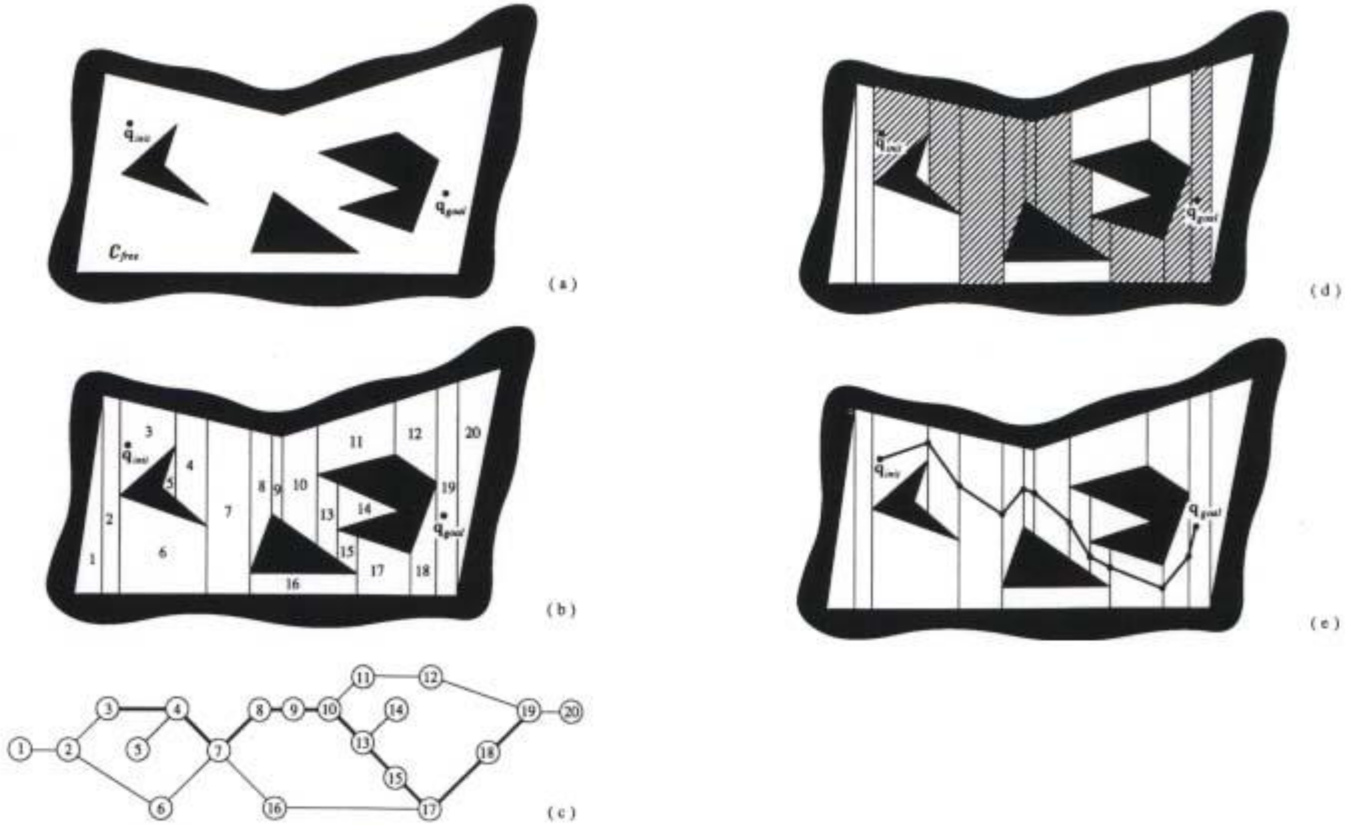
- High robot location precision
- Respect the real world obstacle position and shape
- Low cost memory in case of spare environnent

## - Limitations

- High computation and memory cost in environement with high objects density
- Path planning becomes harder

# Decomposition

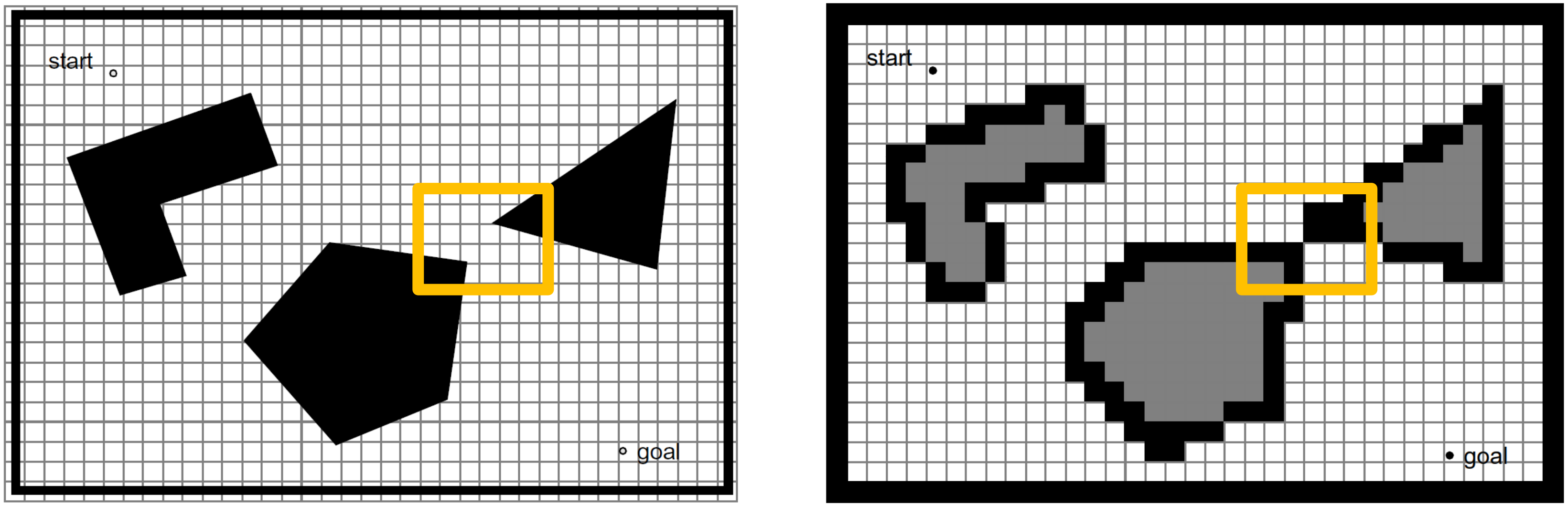
- Exact cell decomposition




http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html

# Decomposition

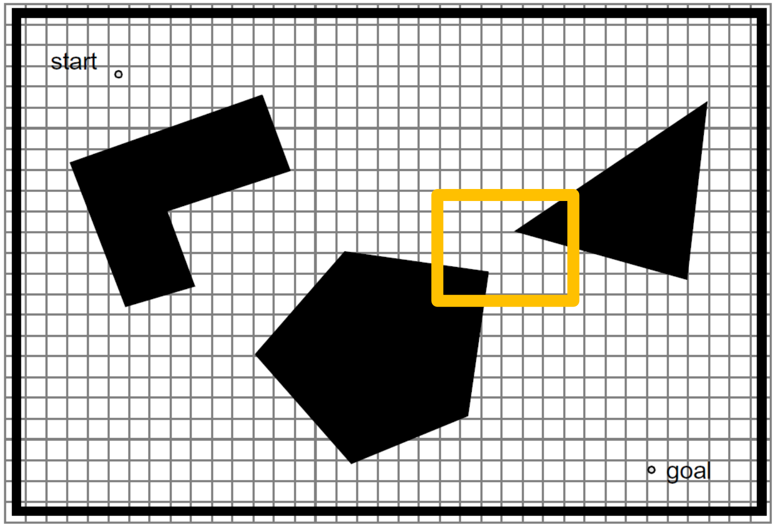
- Fixed decomposition

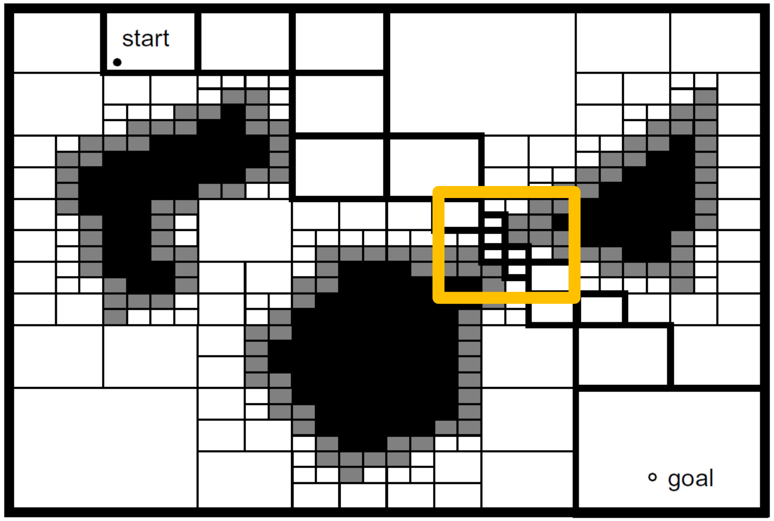



Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

# Decomposition

- Adaptative decomposition

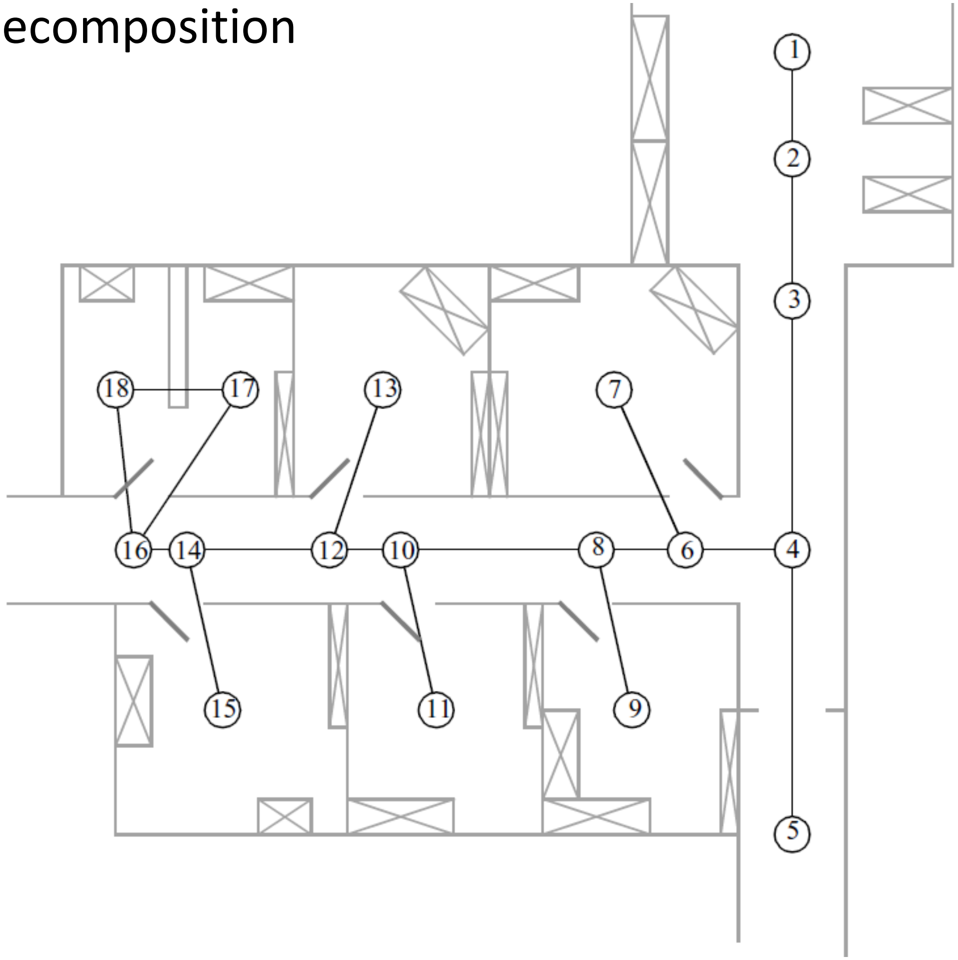



Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

# Decomposition

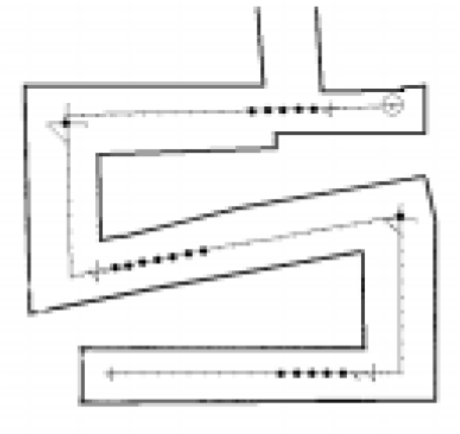
- Topological decomposition




http://www.cim.mcgill.ca/~mrl/pubs/saul/iros98.pdf

# Decomposition

- Topological decomposition



http://www.cim.mcgill.ca/~mrl/pubs/saul/iros98.pdf

# Decomposition

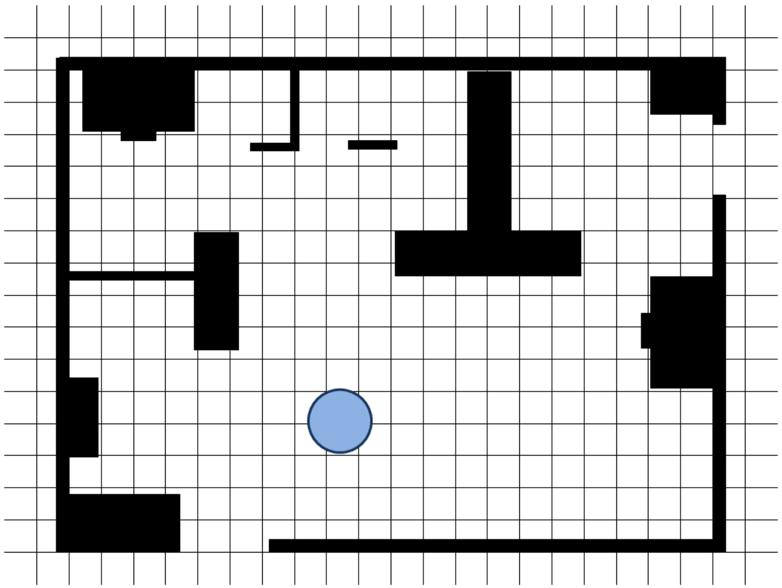
## + Avantages

- Most of the time the map size is predictable
- Ajustable abstraction is possible according to the targeted goal
- Lot's of path planning algorihm exist

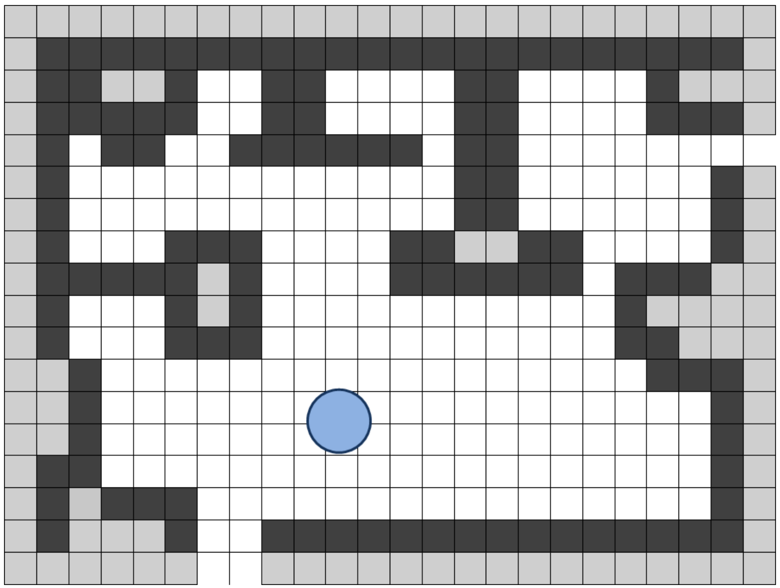
## - Limitations

- Could be far from real environment geometry and representation
- Size of the map could grow with the size of the environment

# Case of study: Occupancy grid



Fixed cell size decomposition



Resulted occupancy grid map

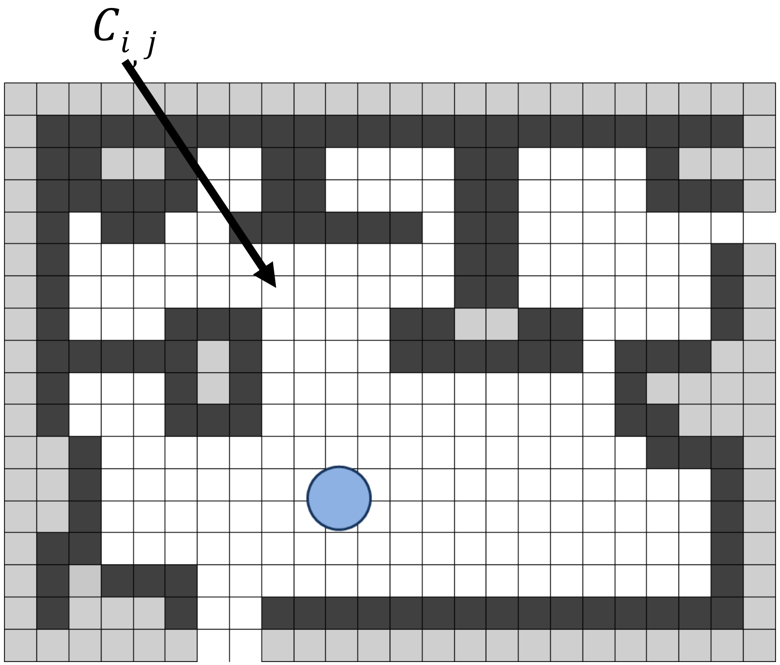
*unknown area*

*cell with obstacle*

*free cell*

# Case of study: Occupancy grid

- Definitions




$occ(i,j)$ — $C_{i,j}$ is occupied

$p(occ(i,j))$ — Probability that $C_{i,j}$ is occupied [0,1]

$o(occ(i,j))$ — Odds function has range $[0,+\infty)$

$$o(A) = \frac{p(A)}{p(\neg A)}$$

$\log(o(occ(i,j)))$ — Log Odds function has range $(-\infty,+\infty)$

→ Each $C_{i,j}$ holds a value $\log(o(occ(i,j)))$

$$\log(o(occ(i,j))) = log\,\frac{p(occ(i,j))}{p(\neg occ(i,j))}$$

# Case of study: Occupancy grid

- Updating grid
  - On each observation by sensor the following assumption is made
  - Reminder : Bayes law

$$p(A|B) = \frac{p(B|A) * p(A)}{p(B)}$$

$A$ is occ(i, j)

$B$ is an observation r giving a value D

$$p(\neg A|B) = \frac{p(B|\neg A) * p(\neg A)}{p(B)}$$

$$o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A) * p(A)}{p(B|\neg A) * p(\neg A)} = \lambda(B|A) * o(A)$$

CPE LYON
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

# Case of study: Occupancy grid

- Updating grid

$$o(A|B) = \lambda(B|A) * o(A)$$

$A$ is occ(i, j)

$B$ is an observation r giving a value D

$$o(A|B) = \frac{P(A|B)}{P(\neg A|B)}$$

$$\frac{\text{Probability that } C_{i,j} \text{ is occupied knowing an observation } r = D}{\text{Probability that } C_{i,j} \text{ is **not** occupied knowing an observation } r = D}$$

$$\lambda(B|A) = \frac{P(B|A)}{P(B|\neg A)}$$

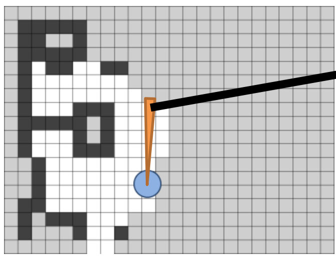
$$\frac{\text{Probability that we made an observation } r = D \text{ knowing } C_{i,j} \text{ is occupied}}{\text{Probability that we made an observation } r = D \text{ knowing } C_{i,j} \text{ is free}}$$

By extension :

$$\log(o(A|B)) = \log(\lambda(B|A)) + \log(o(A))$$

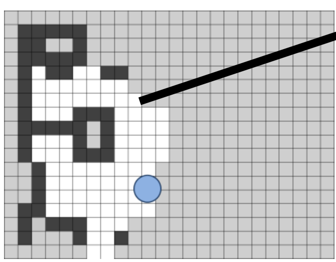
# Case of study: Occupancy grid

- Update Algorithm

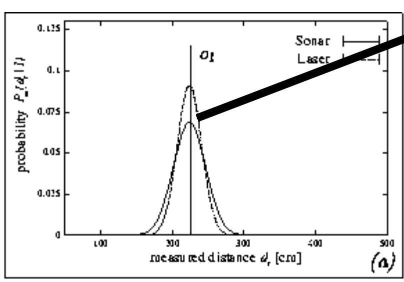


$r = D$

$C_{i,j} = \log(o(occ(i,j)))$



$\mathrm{p}(\mathrm{r} = \mathrm{D}|occ(i,j))$



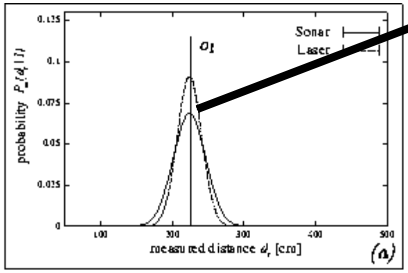

1. Sensor (lazer) get information about the environment r=D on the cell $C_{i,j}$.

2. Information about the map is collected on the targeted cell $C_{i,j} = \log(\mathrm{o}(occ(i,j)))$

3. The new believe on the cell is computed :

$$\log(o(\mathrm{occ(i,j)|r = D)}) = \log\left(\lambda\left(\mathrm{r} = \mathrm{D}|occ\left(i,j\right)\right)\right) + \log(\mathrm{o}(occ\left(i,j\right)))$$

# Case of study: Occupancy grid

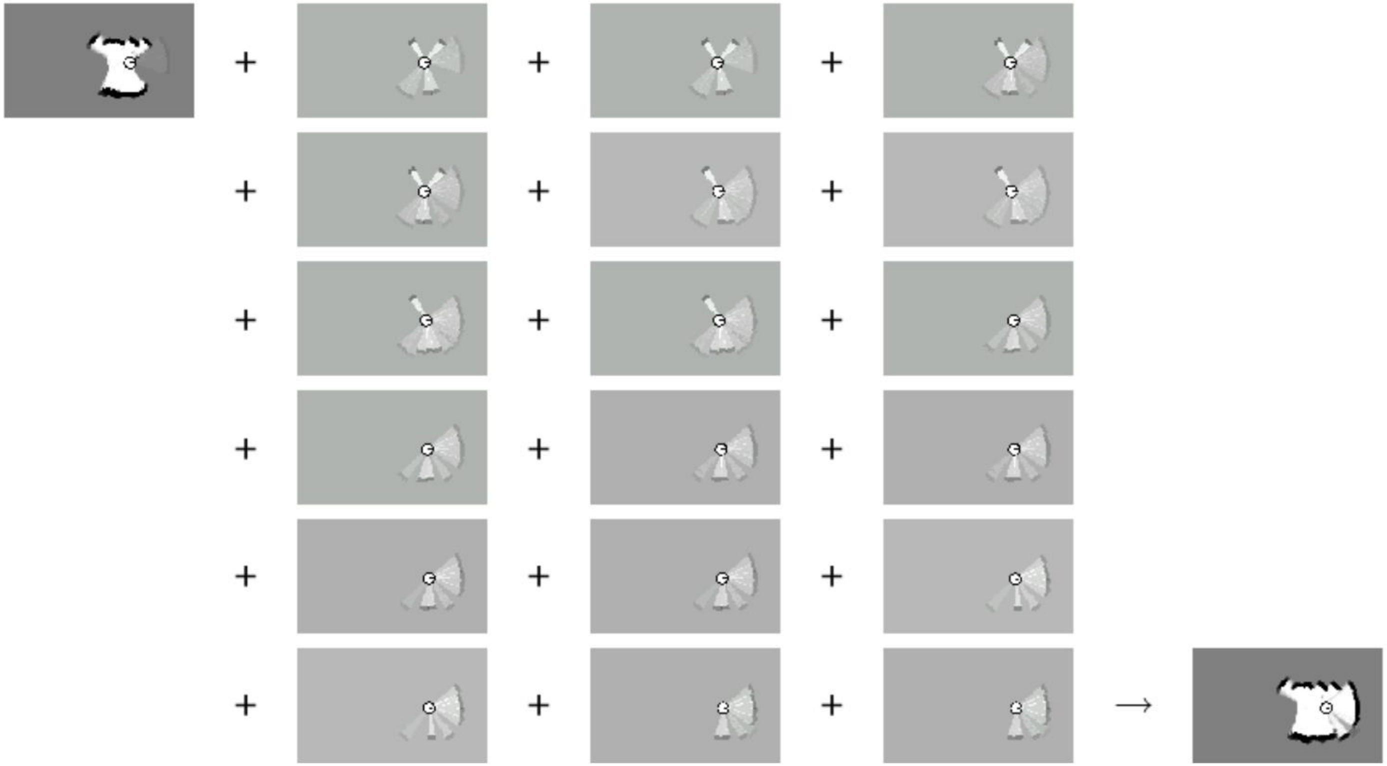



$p(r = D|occ(i,j))$ Lazer detects at a distance D on the cell $C_{i,j}$.

$p(r > D|\neg occ(i,j))$ Lazer passes through the cell $C_{i,j}$

$$\lambda(r = D|occ(i,j)) = \frac{p(r = D|occ(i,j))}{p(r = D|\neg occ(i,j))}$$

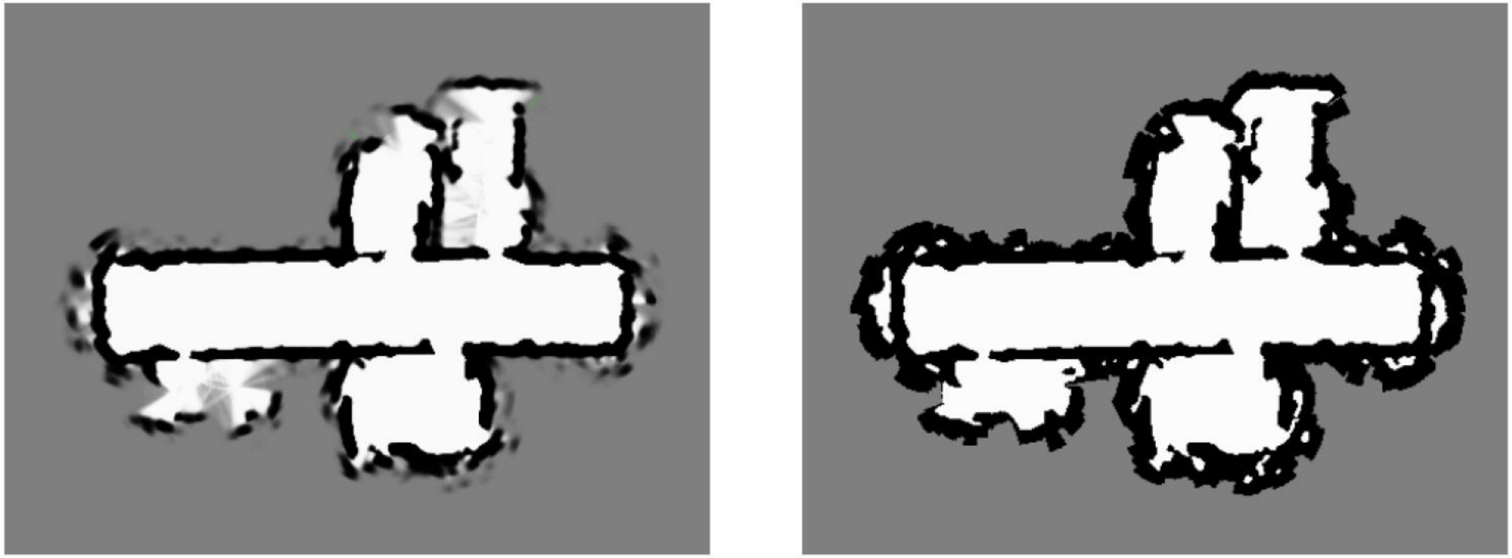
# Case of study: Occupancy grid



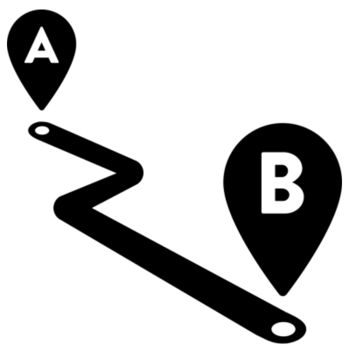
Introduction to Mobile Robotics, Mapping with Known Poses, Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras

# Case of study: Occupancy grid

- Using a given grid map occupancy value (e.g 0.5)



Introduction to Mobile Robotics, Mapping with Known Poses, Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras



# Navigation:
# Path Planning

# Path Planning

- Objective:

find continuous path $\tau$ into $\mathcal{C}_{free}$ from start position $q_I$ to goal position $q_G$.

- 3 main approaches
  - **Road Map path planning**

    Identify a set of routes within $\mathcal{C}_{free}$
  - **Cell Decomposition path planning**

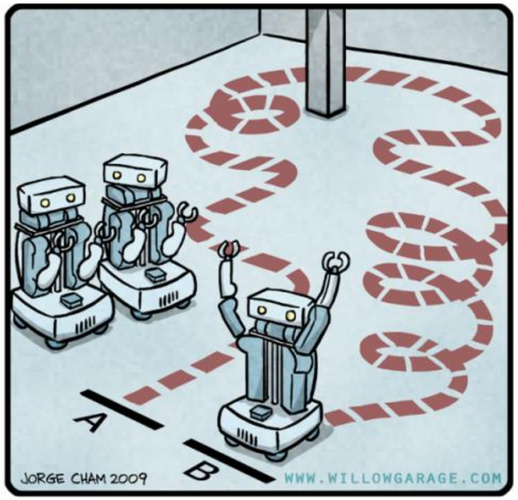
    Discrimintate between free and occupied cells

    (Exact Cell Decomposition, Adaptative Cell Decomposition)
  - **Environmental based path planning**

    Environement information drive the algorithm

    Potential field, ant colony

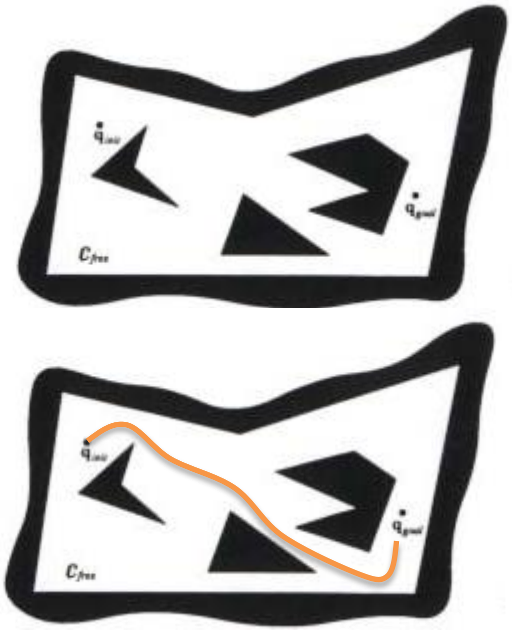
R.O.B.O.T. Comics




"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

$$path\ \tau: [0,1] \rightarrow \mathcal{C}_{free}$$
$$\tau(0) = q_I$$
$$\tau(1) = q_G$$
$$\tau = \{q_I, \ldots, q_i, \ldots, q_G\}$$

# RoadMap Planning



- Methods
  - Visibility Graph
  - Voronoi Diagram
  - Rapid Random Tree

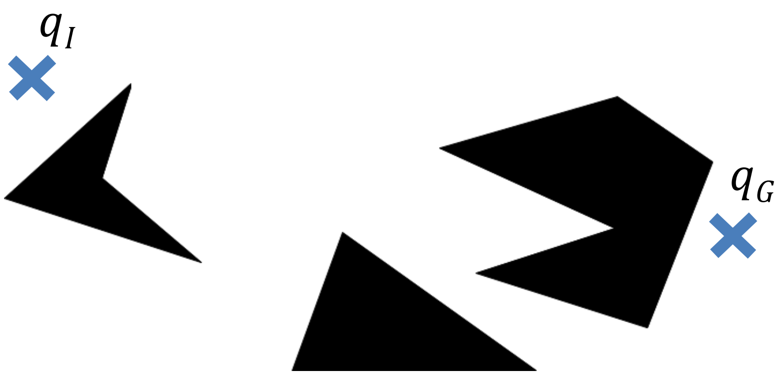
- Properties
  - **Produce a graph in $\mathcal{C}_{free}$ such as vertex is in $\mathcal{C}_{free}$ and edge a collision free path in $\mathcal{C}_{free}$**
  - Mostly based on continuous map (ploygonal representation of the environment)

# Visibility Graph

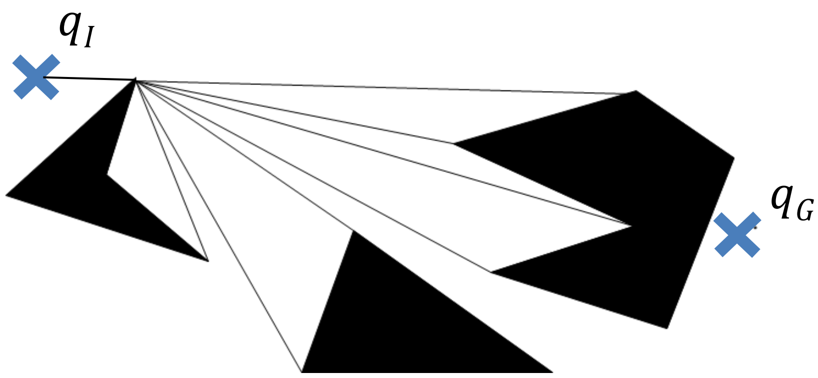
- Objective
  - Create a connectivity graph between obstacles vertices and start/ goal position

- Algorithm
  - Graph computation
    - **vertices** : all vertices of obstacles (polygon) + start point and goal point
    - **edges** : edges joining all pair of vertices that can « see » each other
  - Path selection
    - Short path algorithms (Dijkstra , A*)

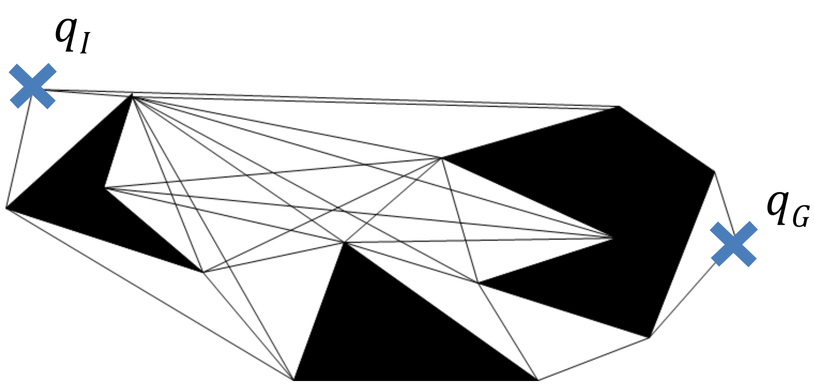
# Visibility Graph



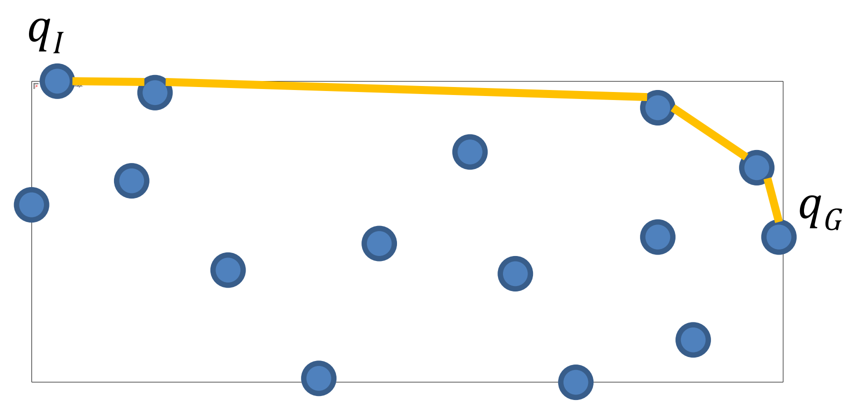

Initial Situation




1 Vertex and associated edges




All Vertices and associated edges




Resulted Graph

# Visibility Graph

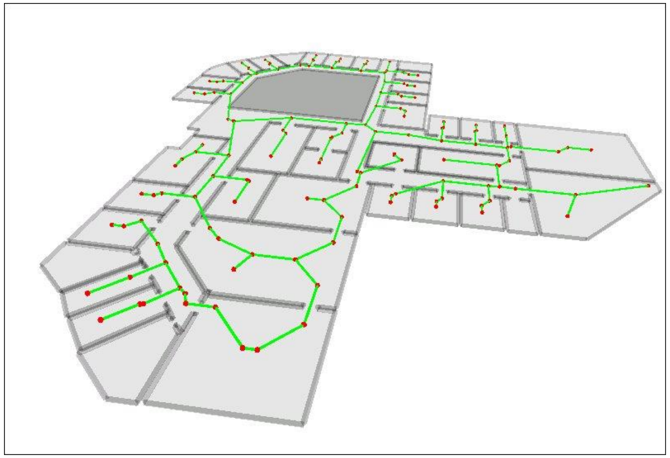
## + Avantages

- Very simple
- Good candidate if continuous representation
- Fast on sparse environement

## - Limitations

- The size depends of number of polygon vertices
- Slow on densely populated environment
- Robot tend to be very close to the obstacles

# Voronoi Diagram



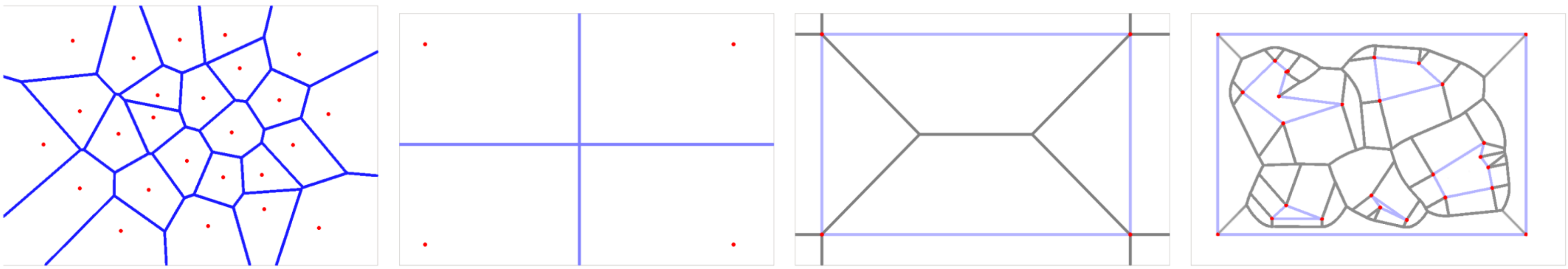
- Objective

construct lines from all points that are equidistant from 2 or more obstacles

- Algorithm
  - Graph Construction
    - Green et Sibson
    - Shamos et Hoey
    - Fortune
    - Randomized incremental construction
  - Path selection
    - Short path algorithms (Dijkstra , A*)

# Voronoi Diagram

Usage sample: http://alexbeutel.com/webgl/voronoi.html



(a) (b) (c) (d)

(**a**) random points, *k* = 25; (**b**) four points forming a rectangle, *k* = 4; (**c**) four walls forming a rectangular environment; (**d**) rectangular environment with fives polygonal obstacles with pruned parts of the Voronoi diagram outside the freespace of the polygonal environment

http://www.mdpi.com/1424-8220/15/6/12736/htm

# Voronoi Diagram

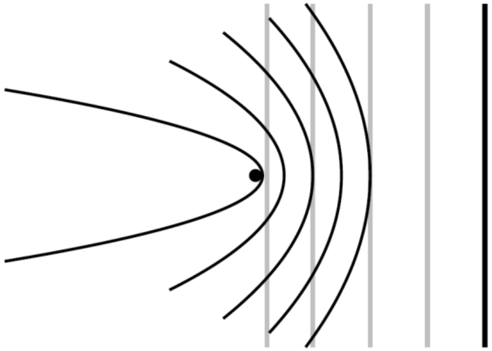
- Fortune Algorithm

**Sweep line** : vertical line moving from the left to the right

**Beach line** : parabolas compositions dividing the portion of the plane on the left side of the sweep line

When obstacle is cross by the sweep line a parabol is added to the beach line such as is point of this parabol is equidistant from the obstacle to the sweep line

**Vertices of the beach line** refers to parabol intersection points



# Voronoi Diagram

## + Avantages

- ❑ Allow « safe » navigation
- ❑ Executability (better for obstacle avoidance)
- ❑ Interesting for autonomous mapping

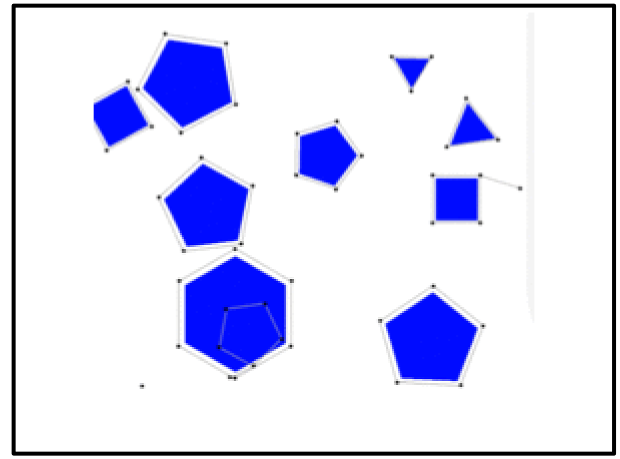
## - Limitations

- ❑ Non optimal navigation path length
- ❑ Localization becomes difficult for short range sensors
- ❑ Unnatural attraction to openspace → suboptimal path

# Probabilistic RoadMap (PRM)

❑ Objective

Determining a path between $q_I$ and $q_G$ without obstacle collision by getting successif random point in $\mathcal{C}_{free}$



❑ Algorithm

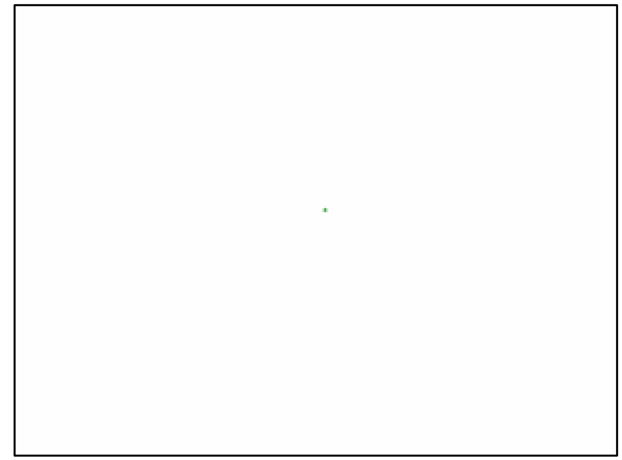
- Graph Construction
  - Take random point
  - Check random point in $\mathcal{C}_{free}$
  - Try to connect this point current graph through « a local planner »
- Path selection
  - Short path algorithms (Dijkstra , A*)

# Rapid Random Tree (RRT)

- Objective

Explore aggressively $\mathcal{C}$ by extending possible locations from initial position $q_I$

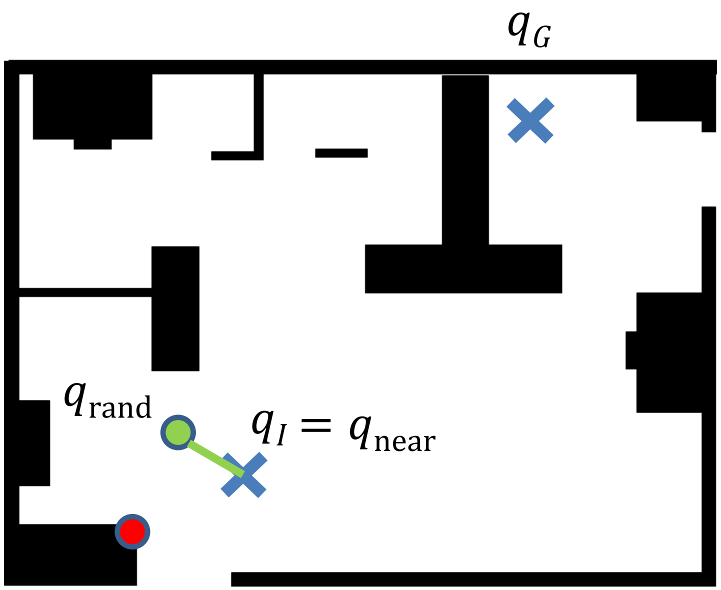
- Algorithm
  - Graph Construction
    - Incremental algorithm
  - Path selection
    - Short path algorithms (Dijkstra , A*)



$G.Init(q_I)$
**Repeat**
$\quad q_{rand} \rightarrow Random\_Config(\mathcal{C})$
$\quad q_{near} \rightarrow Nearest(G, qrand)$
$\quad G.add\_edge\,(q_{near}, qrand)$
**Until** $condition$

# Rapid Random Tree (RRT)

$G.Init(q_I)$
**Repeat**
$q_{rand} \rightarrow Random\_Config(\mathcal{C})$
$q_{near} \rightarrow Nearest(G, qrand)$
$G.add\_edge(q_{near}, qrand)$
**Until** *condition*




1st it.

# Rapid Random Tree (RRT)

$G.Init(q_I)$

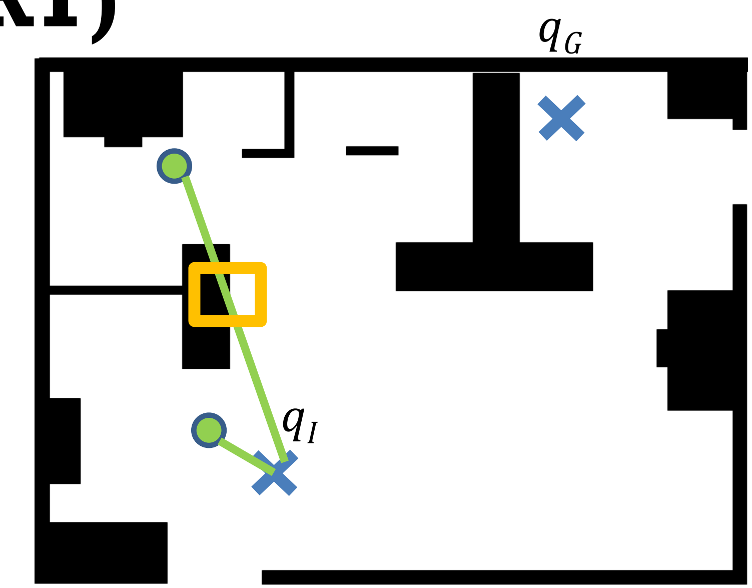
**Repeat**

$q_{rand} \rightarrow Random\_Config(\mathcal{C})$

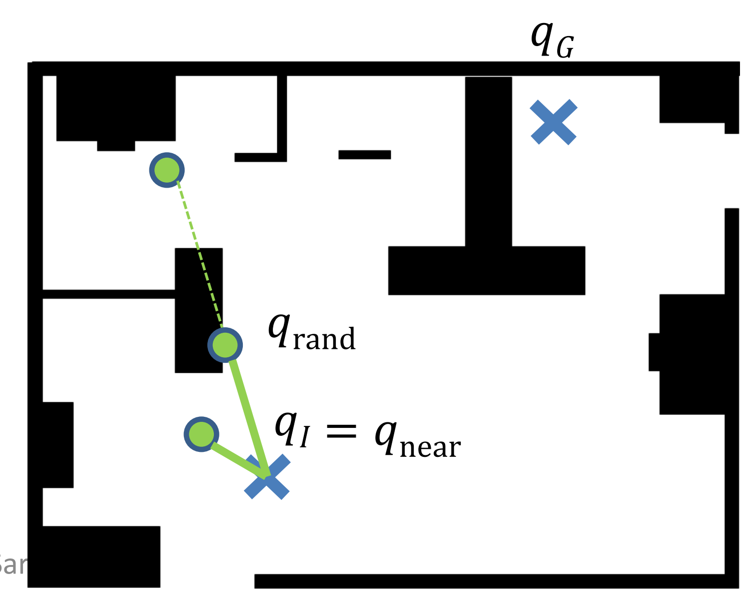
$q_{near} \rightarrow Nearest(G, qrand)$

$G.add\_edge\,(q_{near}, qrand)$

**Until** *condition*




2nd it.




2nd it.

# Rapid Random Tree (RRT)

$G.Init(q_I)$

**Repeat**

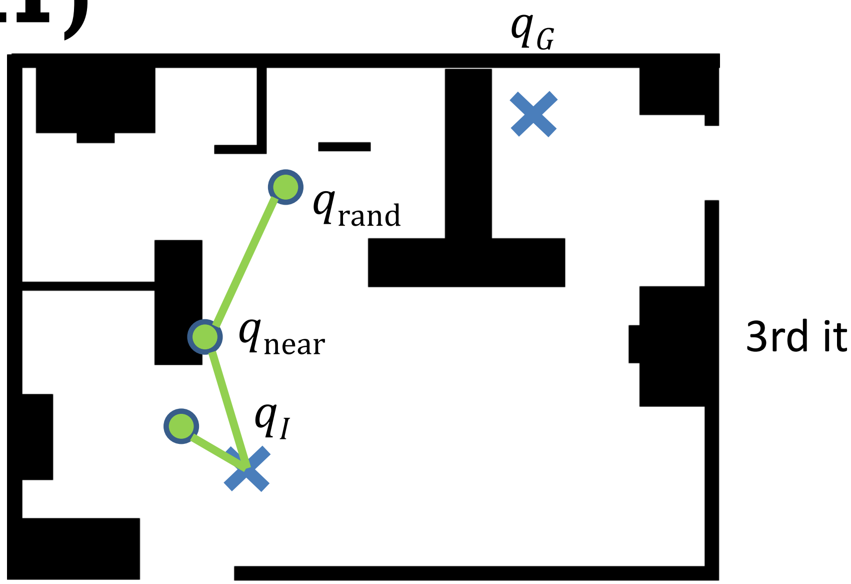
$q_{rand} \rightarrow Random\_Config(\mathcal{C})$

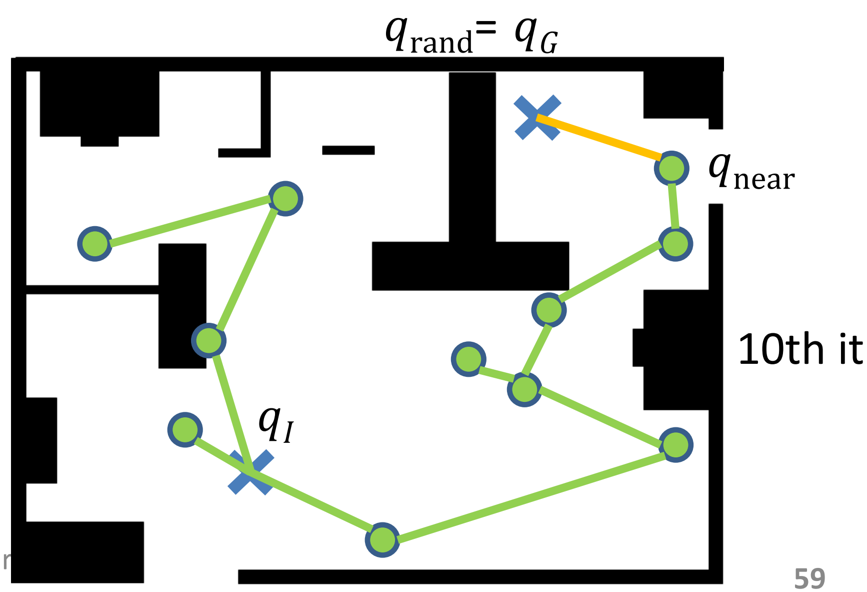
$q_{near} \rightarrow Nearest(G, qrand)$

$G.add\_edge\,(q_{near}, qrand)$

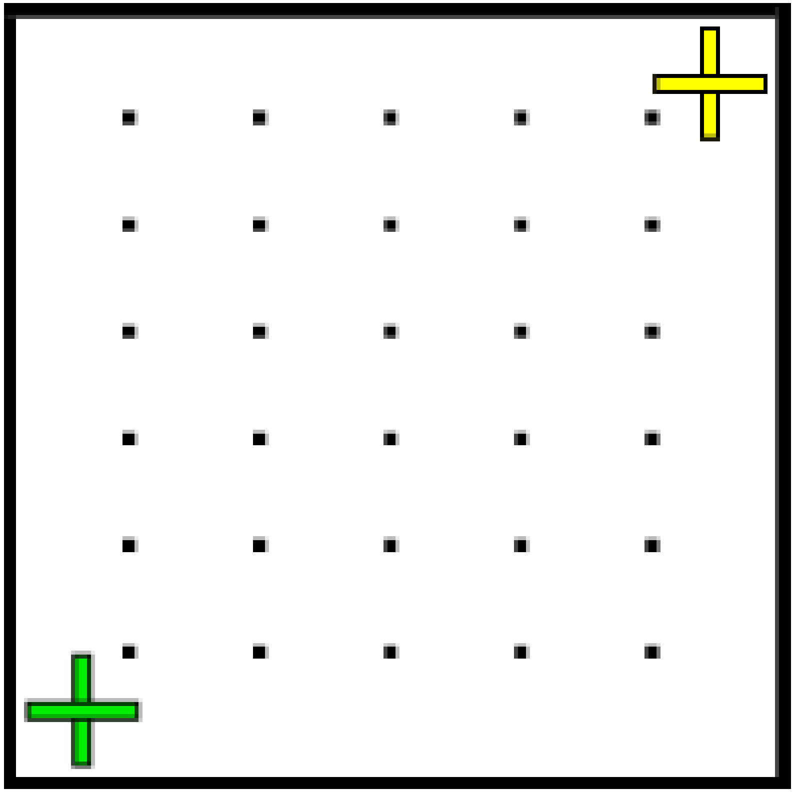
**Until** *condition*

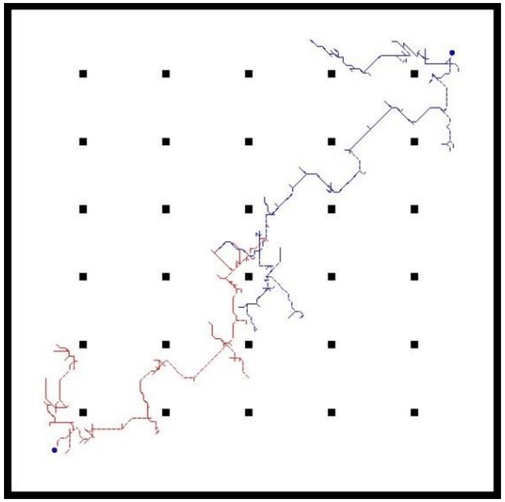
At *n* th iterations force $q_{\text{rand}} = q_G$

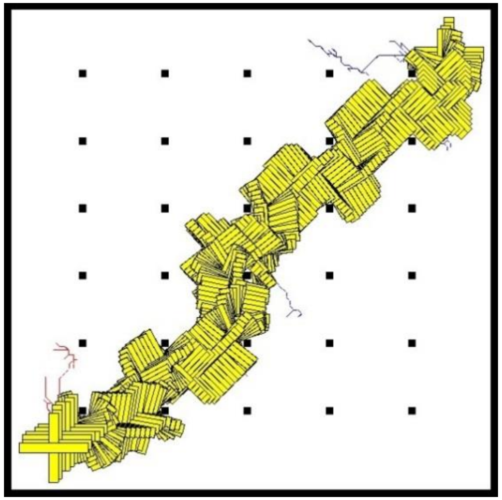


# Rapid Random Tree (RRT)





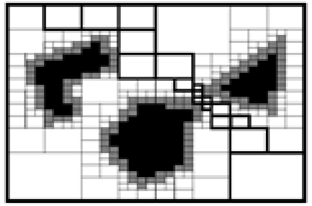


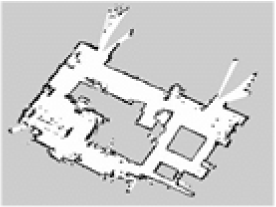
http://msl.cs.uiuc.edu/rrt/index.html
http://msl.cs.uiuc.edu/rrt/gallery_rigid.html

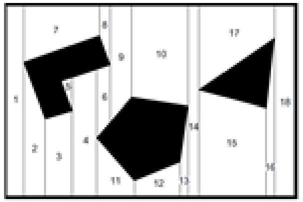
# Decomposition path planning

- Methods
  - Exact cell decomposition
  - Fixed cell decomposition
  - Adaptative cell decomposition

- Properties
  - **Map (Exact / Fixed  / adaptative) gives graph vertices**
  - Cell connectivities gives graph edges

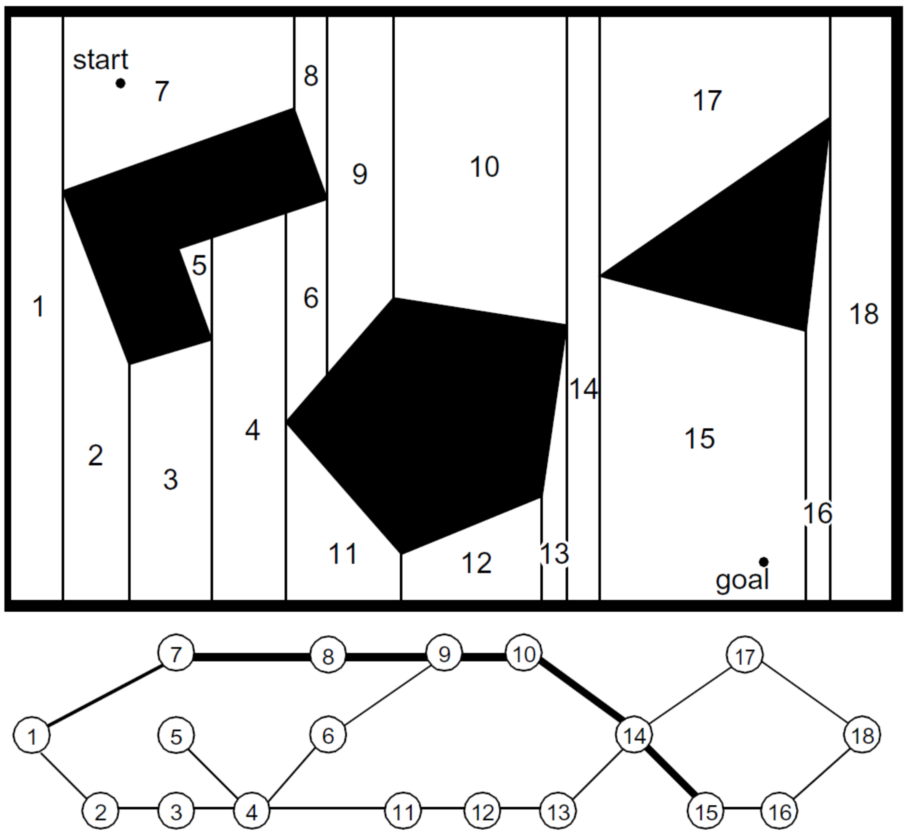






# Cell connectivities

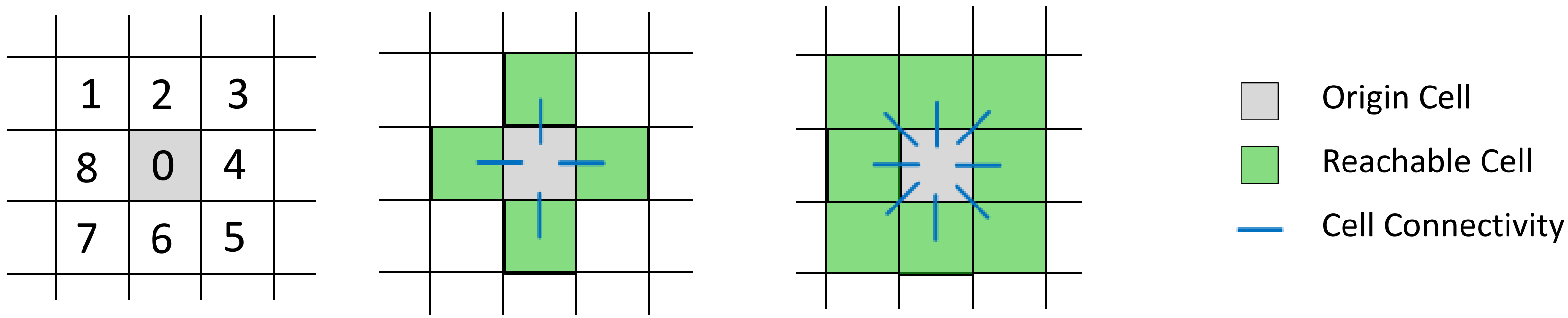
- Exact Cell decomposition
  - Direct Neighbors cell is not an obstacle

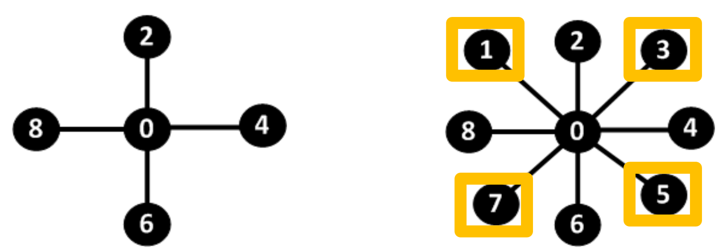



Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004

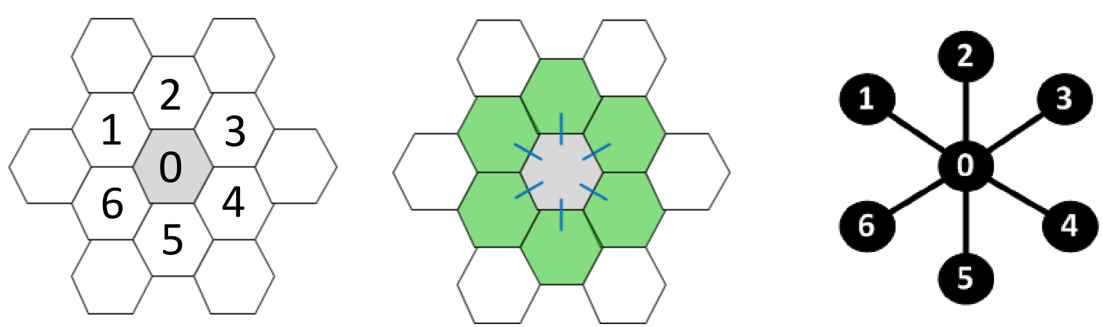
# Cell connectivities

- ❑ Fixed or adaptative Cell decomposing



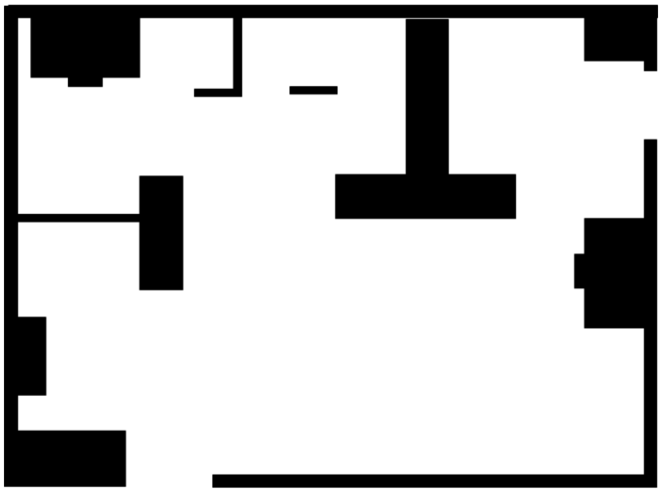




All vertices **are not** equidistant to vertex 0

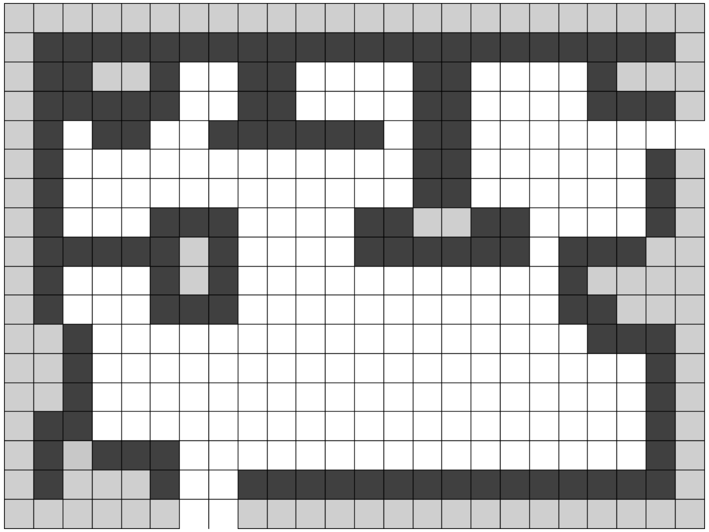



All vertices **ARE** equidistant to vertex 0

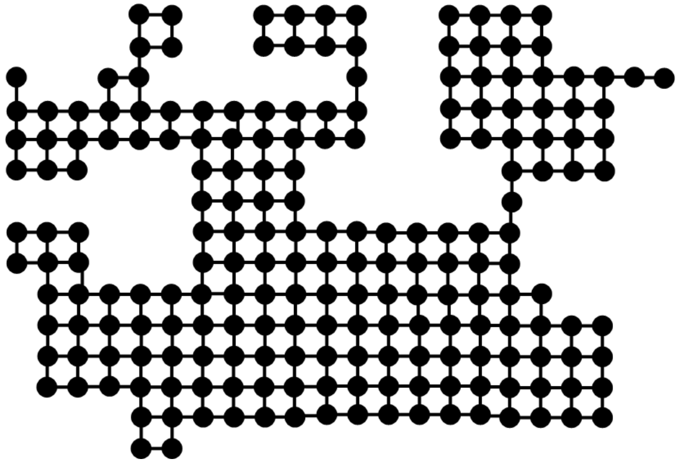
# Cell connectivities



Real environment



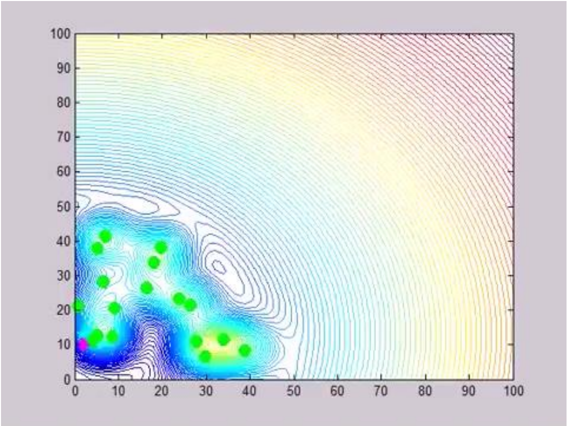
Fixed Cell Decompostion



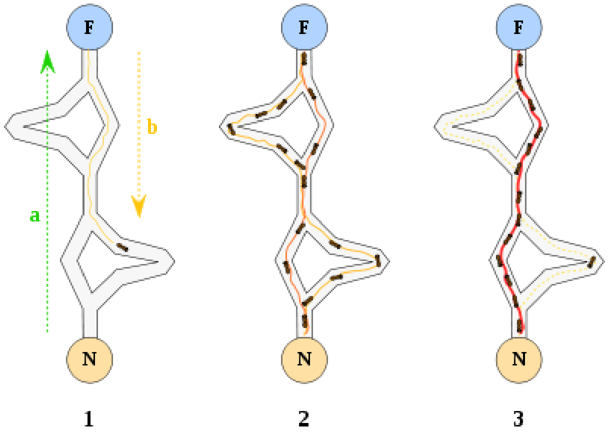
Resulted cell connectivity graph

# Environmental based path planning

- Methods
  - Potential fields
  - Ant colony




- Properties
  - **The environment areas drive the navigation**
  - Robots do not need heavy computation

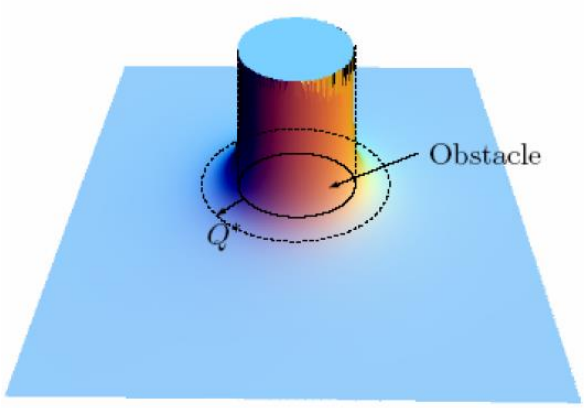


# Potential Fields

- ❑ Objective

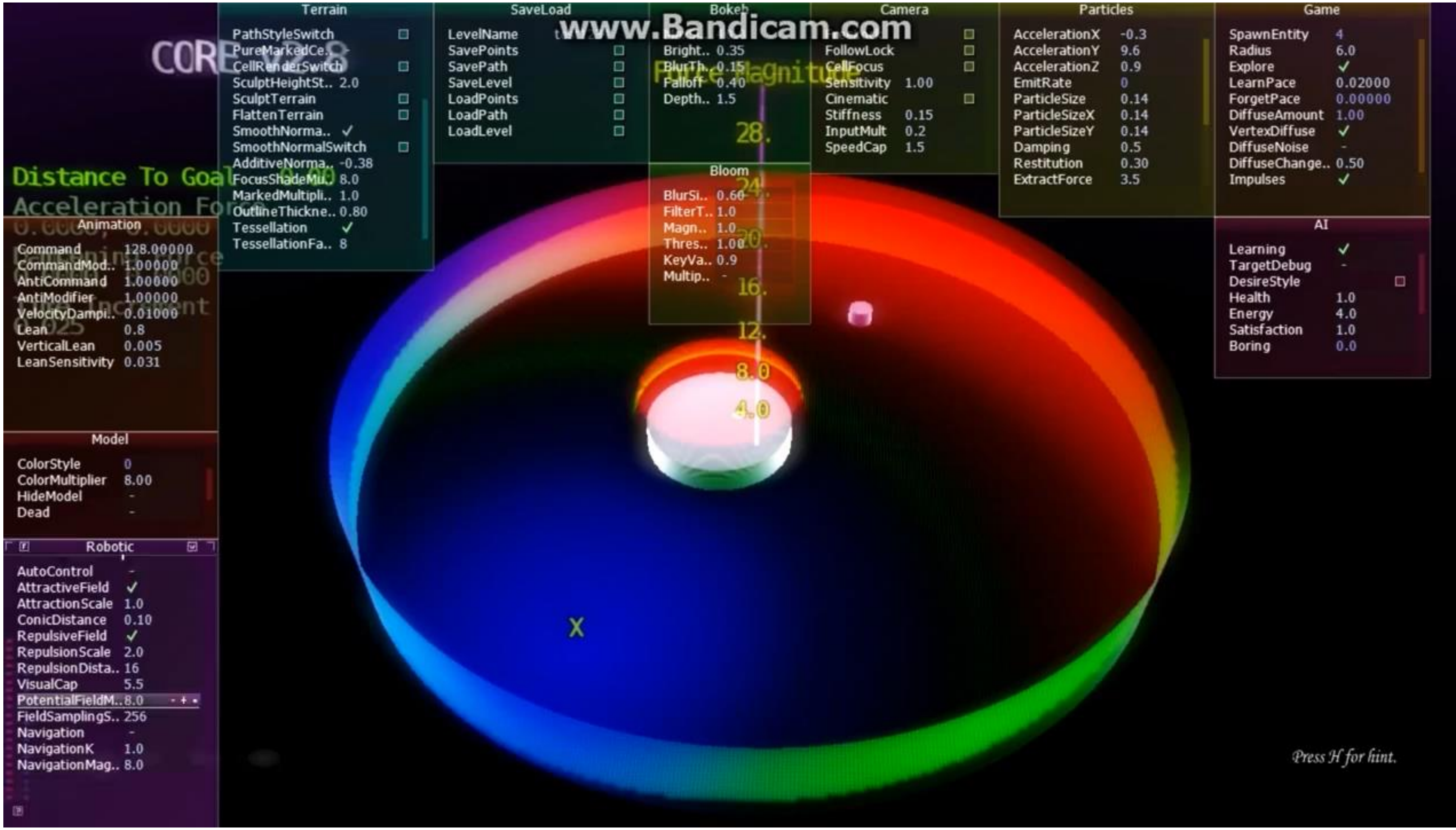
Generate attractive and repulsive potential fieldon the environment to drive the robot until it reaches the goal

- ❑ Algorithm
  - Obstacles generate repulsive potential field. The more the robot is closed to the obstacle, the higher the repusive potential field is,
  - Goal generates attractive potential field



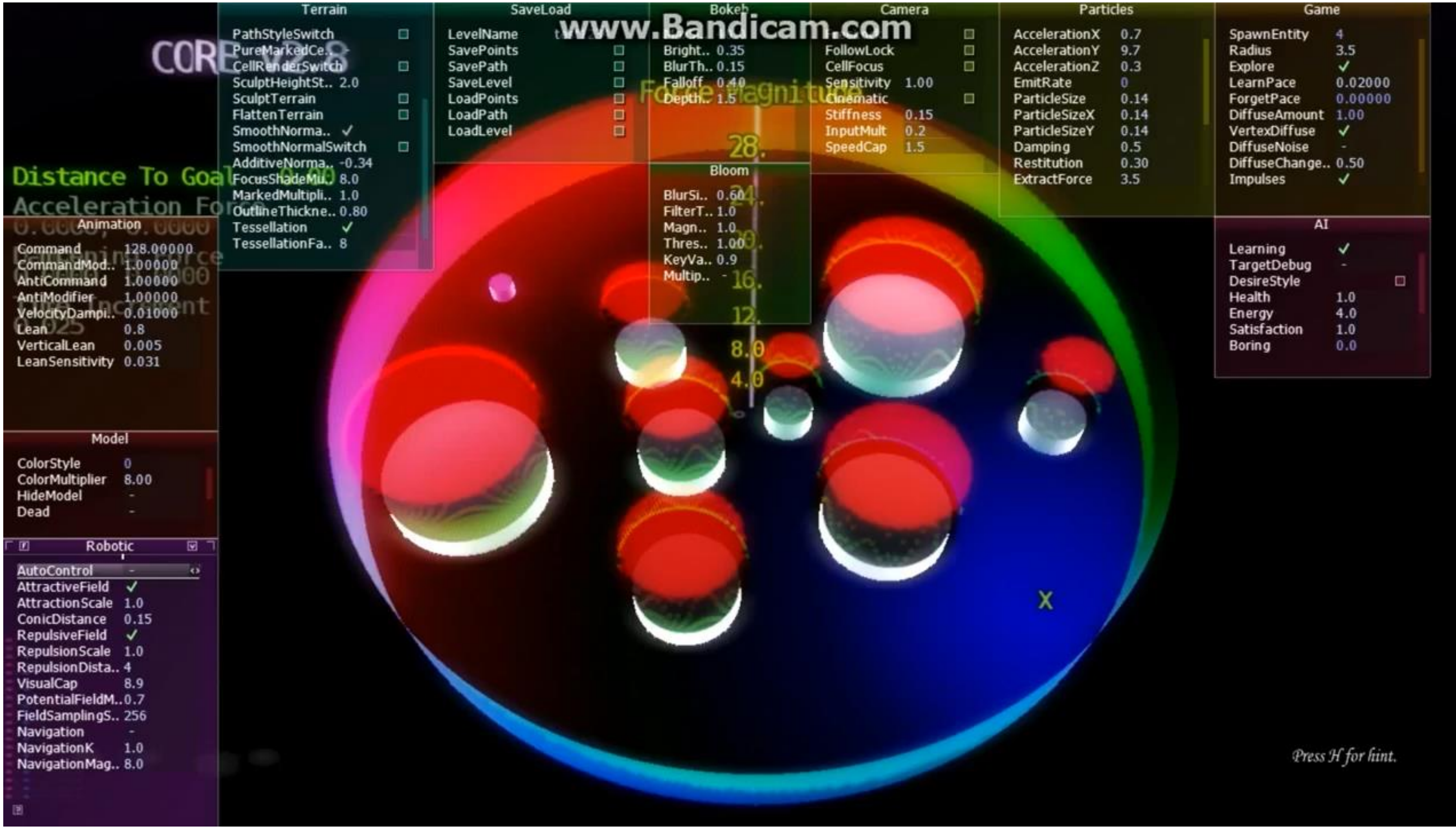

Robotic Motion Planning: Potential Functions,Robotics Institute 16-735, Howie Choset

# Potential Fields




https://www.youtube.com/watch?v=DVnbp9oZZak

# Potential Fields

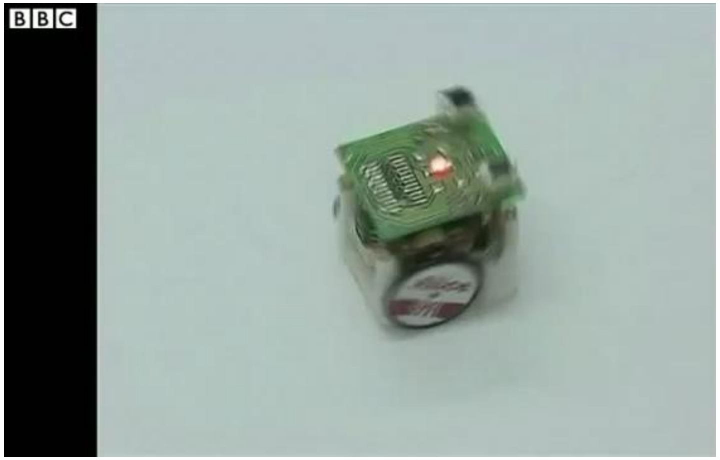



https://www.youtube.com/watch?v=kpWSDyr7sM0



LYON CPE
ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

# Ant colony




https://www.youtube.com/watch?v=vAnN3nZqMqk

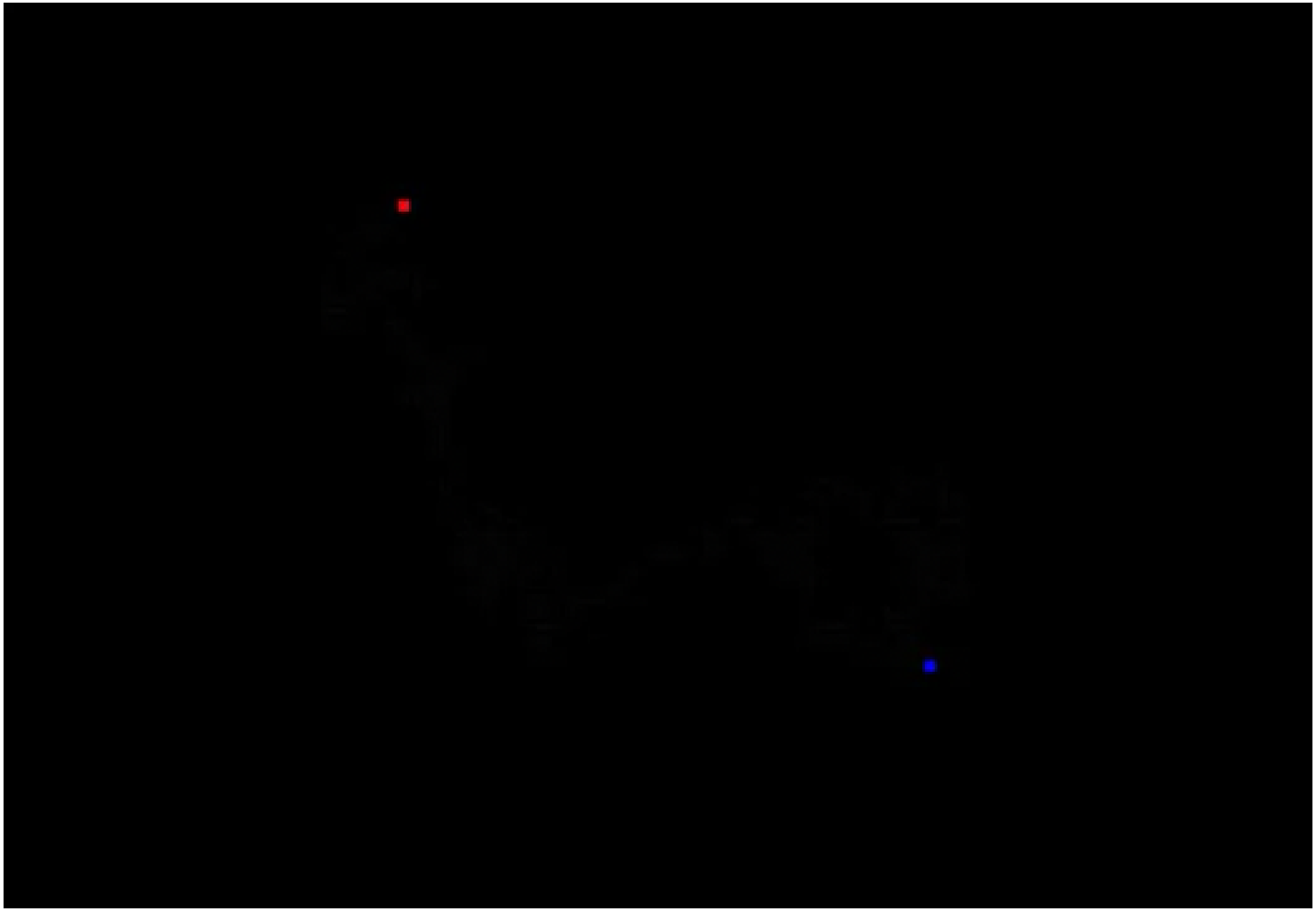
## ❑ Objective

Individues spread on the environment a quantity of pheromone highlighting their path. Large number of individues and evaporation process converge to a solution.

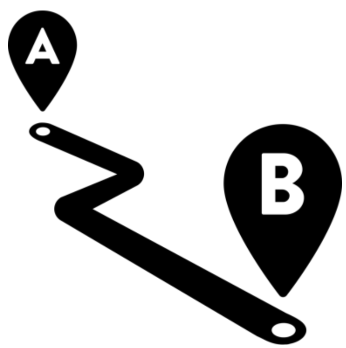
## ❑ Algorithm

- Ants travel on the environment to find food,
- Once 1 ant find food, it comes back to the colony spreading pheromone
- Other ant are attracted by the pheronome and will reinforce the pheromone if they find food
- If several path are possibles, the evaporation process lead to select the shortest path

# Ant colony



https://www.youtube.com/watch?v=vAnN3nZqMqk

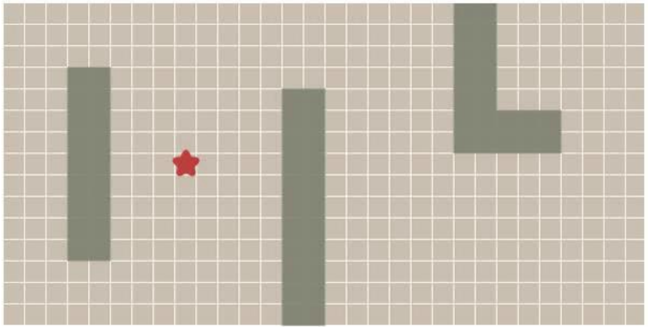



# Navigation:
# Short path samples

# Wavefront: a Breadth-first search

- Principle

Explore the frontier by launching a wavefront that marks each hit cells with a distance to the original point

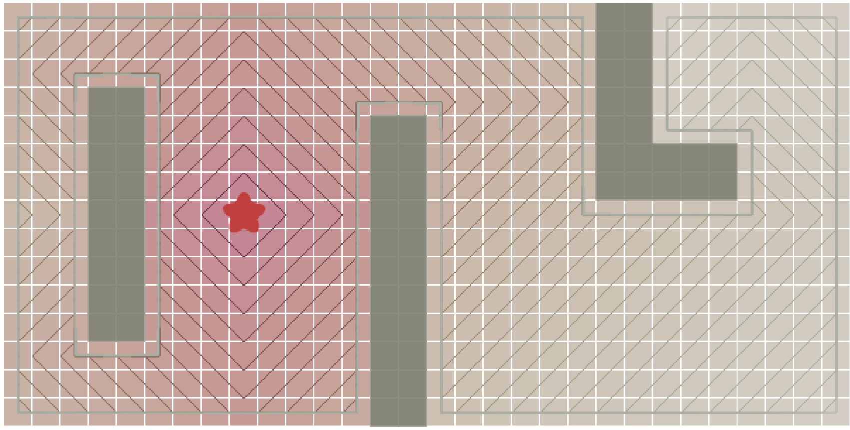


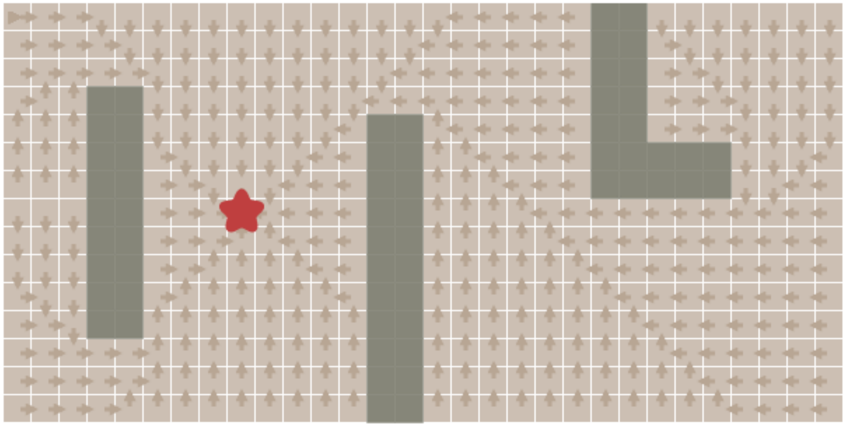
http://www.redblobgames.com/pathfinding/a-star/introduction.html

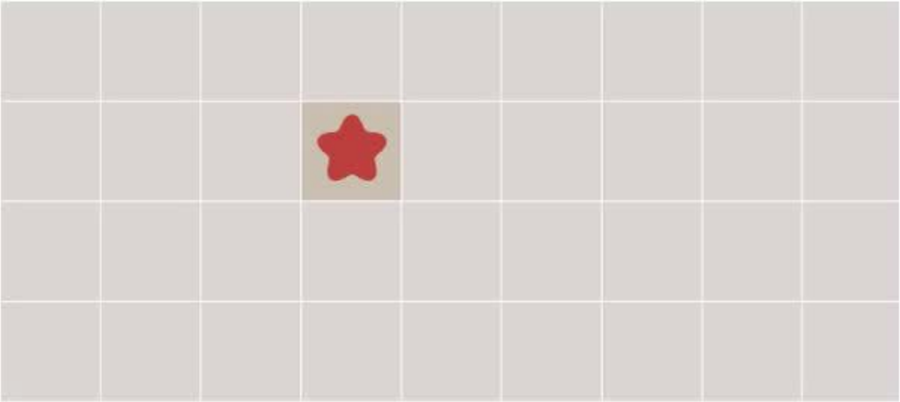
$Unvisited = q_I$
$dist[q_I] = 0$
$prev[q_I] = None$

**For each** $u \in Unvisited$
    $remove\ u\ from\ Unvisited$
    **For each** $v \in Neighbor(u)$
        **If** dist[v]∄
            $add\ v\ to\ Unvisited$
            $dist[v] = dist[u] + 1$
            $prev[u] = v$

# Wavefront: a Breadth-first search

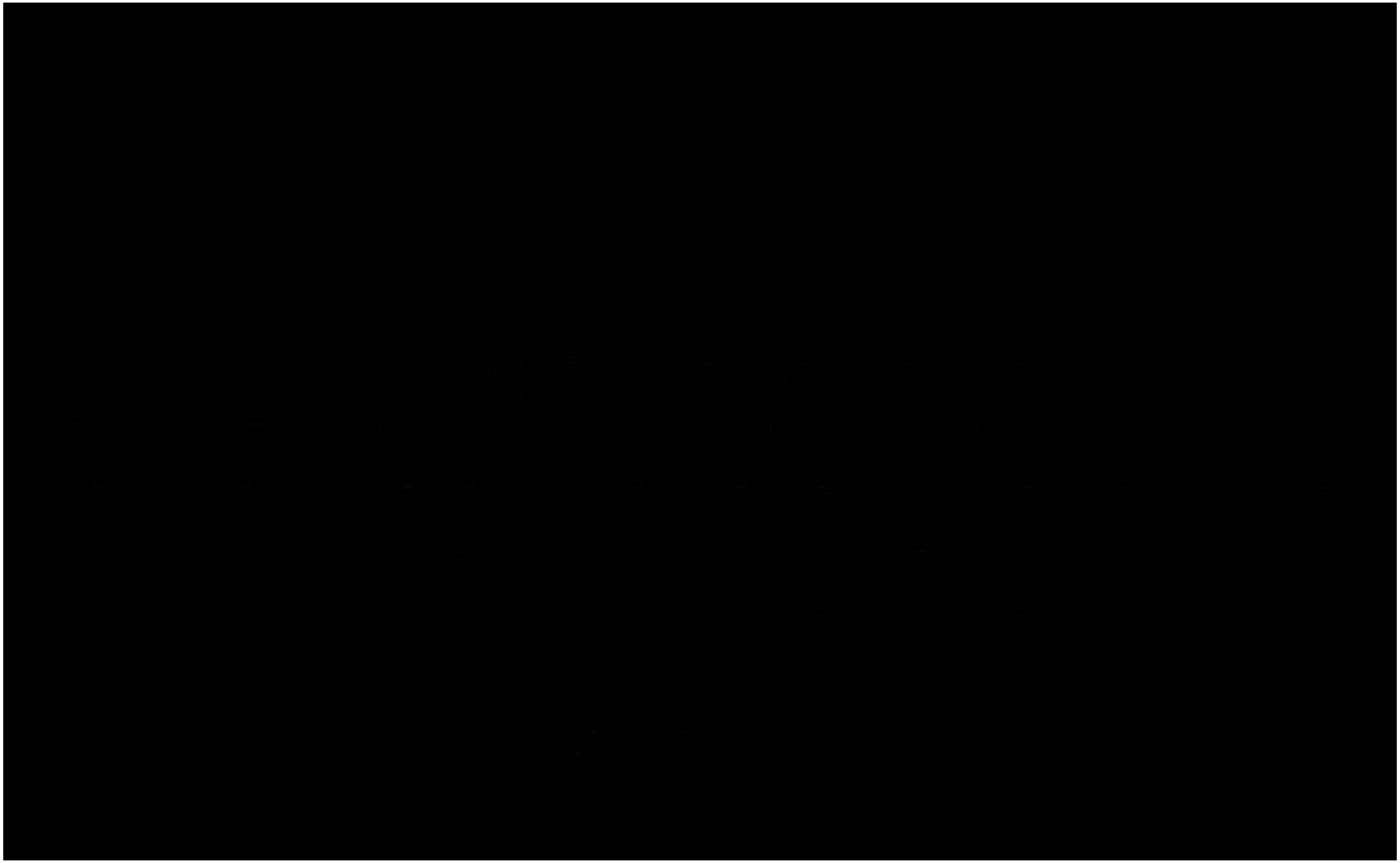






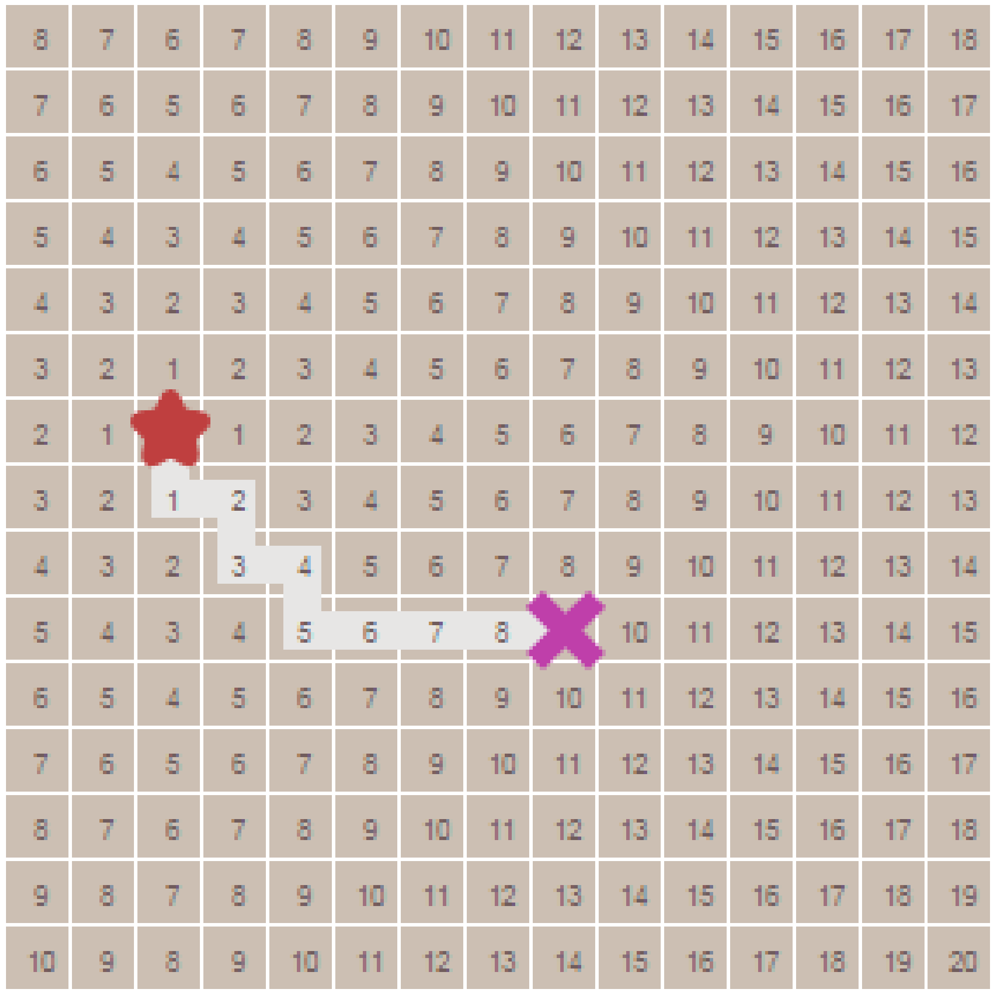
http://www.redblobgames.com/pathfinding/a-star/introduction.html

# Wavefront: a Breadth-first search



https://www.youtube.com/watch?v=ylnH9GctlTA

# Wavefront: a Breadth-first search

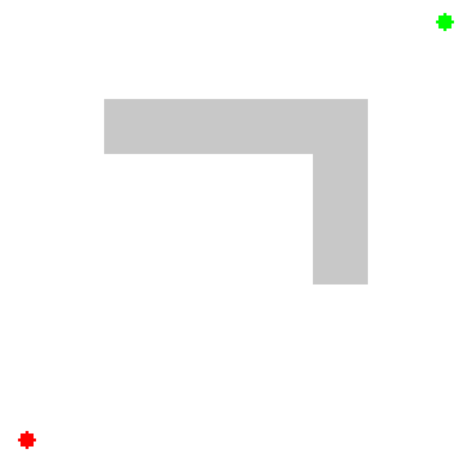



http://www.redblobgames.com/pathfinding/a-star/introduction.html

# Dijkstra's

- Principle

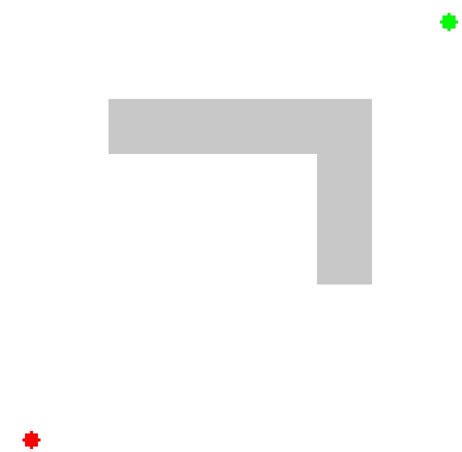
Explore the frontier by selecting candidate points according to their distance to the origine. Cell weight is taken into account in the distance measure



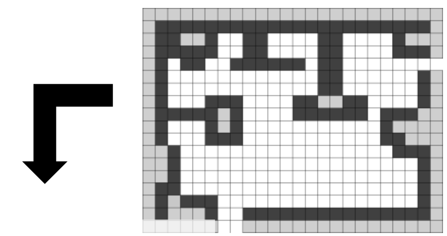
# Dijkstra's

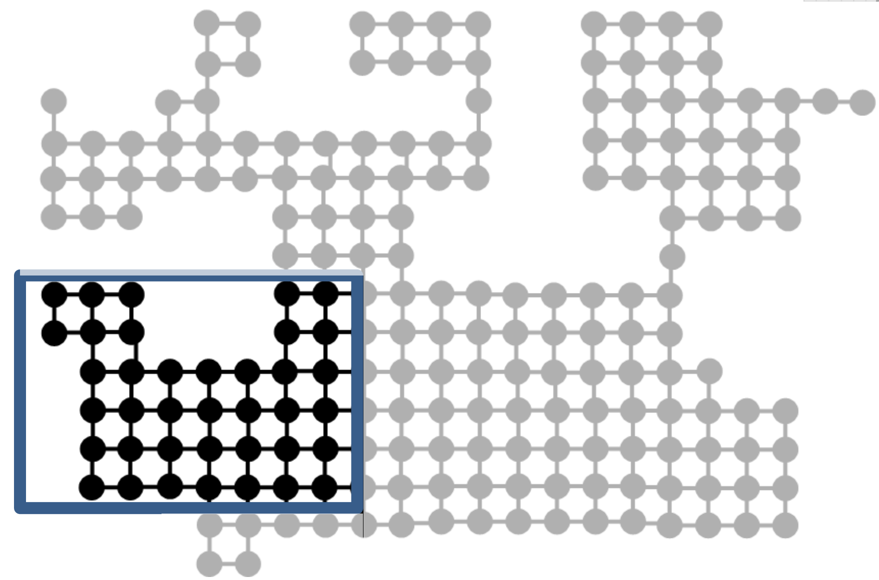
- Algorithm

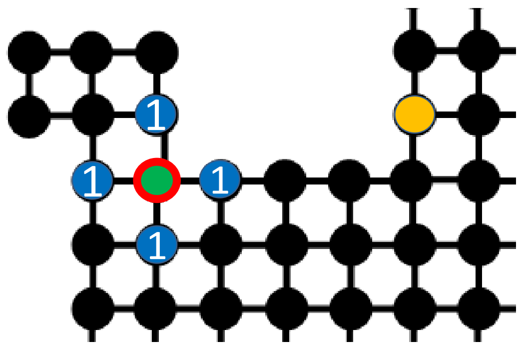
**For each** $C \in \mathcal{C}_{\text{free}}$
$\quad add\ C\ to\ Unvisited$
$\quad \mathrm{f}_{score}[C] = +\infty$
$\quad prev[C] = undefined$
$\mathrm{f}_{score}[q_I] = 0$
**Repeat**
$\quad u \leftarrow MinFscore(Unvisited)$
$\quad remove\ u\ from\ Unvisited$
$\quad$ **For each** $v \in Neighbor(u)$
$\quad\quad current\_score = \mathrm{f}_{score}[u] + length(u, v)$
$\quad\quad$ **If** $current\_score < \mathrm{f}_{score}[\mathrm{v}]$
$\quad\quad\quad \mathrm{f}_{score}[v] = current\_score$
$\quad\quad\quad prev[u] = v$
**Until** $Unvisited\ = \emptyset$

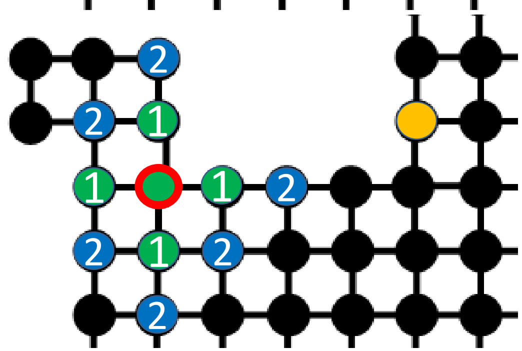


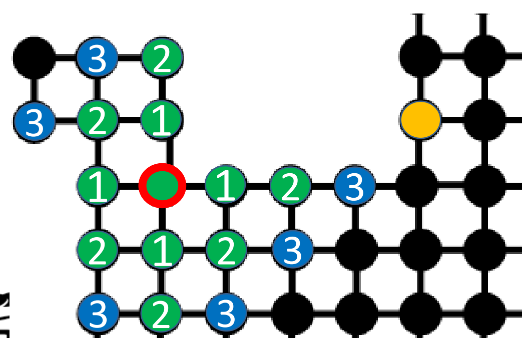
# Dijkstra's

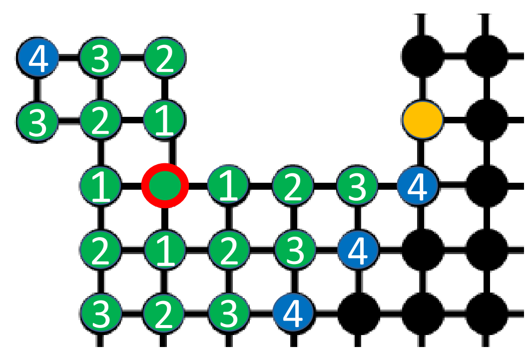


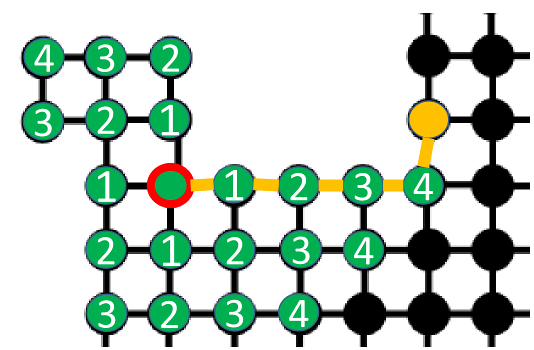




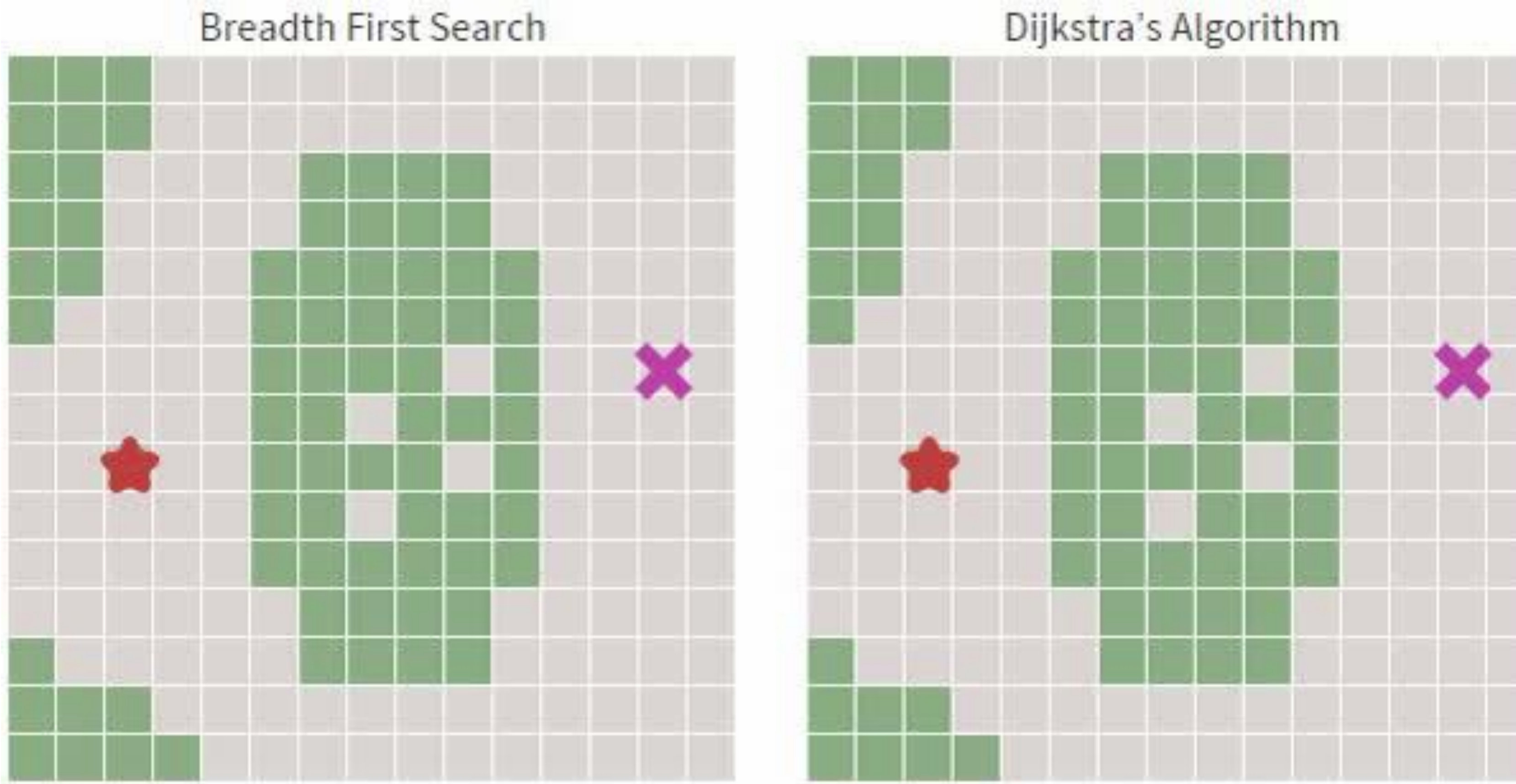








# Dijkstra's

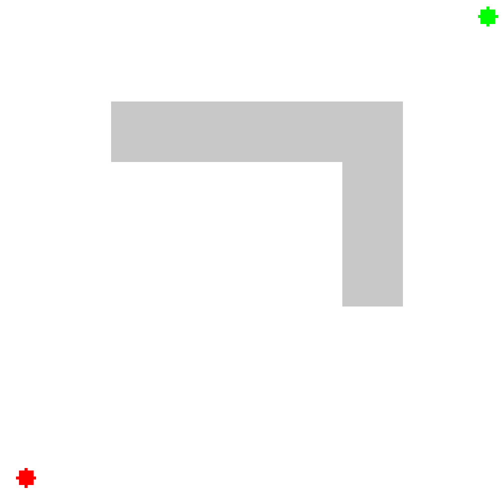



http://www.redblobgames.com/pathfinding/a-star/introduction.html

# Greedy Best First Search

- Principle

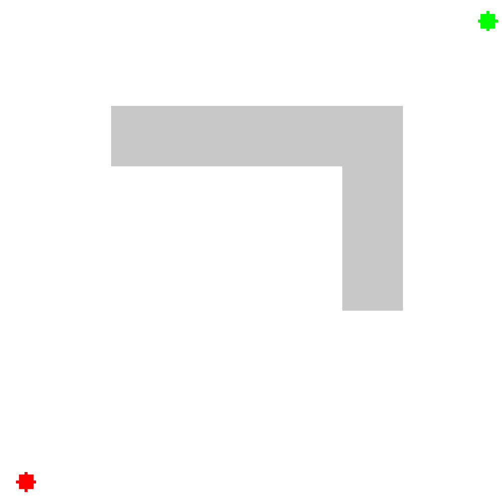
Explore the frontier by selecting candidate points according to **their distance estimate to the goal**.



# Greedy Best First Search

- Algorithm

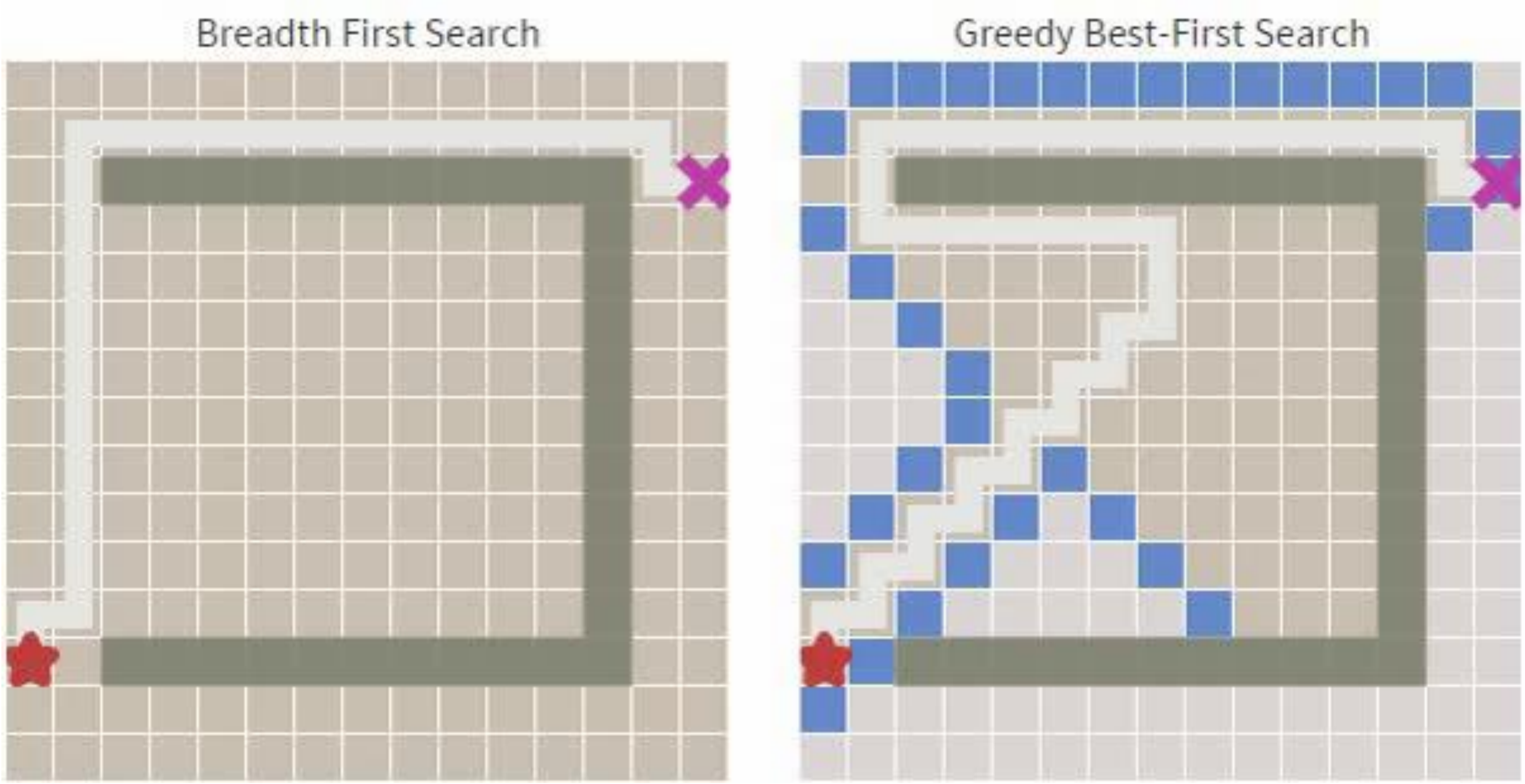
**For each** $C \in \mathcal{C}_{\text{free}}$
  $add\ C\ to\ Unvisited$
  $\mathrm{f}_{score}[C] = +\infty$
  $prev[C] = undefined$
$\mathrm{f}_{score}[q_I] = 0$
**Repeat**
  $u \leftarrow MinFscore(Unvisited)$
  $remove\ u\ from\ Unvisited$
  **For each** $v \in Neighbor(u)$
    $\mathrm{f}_{score}[v] = heuristicCostEstimate(v, q_G)$
    $prev[u] = v$
**Until** $Unvisited = \emptyset$



# Greedy Best First Search


http://www.redblobgames.com/pathfinding/a-star/introduction.html

# Greedy Best First Search




http://www.redblobgames.com/pathfinding/a-star/introduction.html

# Alorithm frontier selection

### Breadth-first search

Unvisited min jump

**j(*u*)= number of jump to reach *u***

**$f_{score}$(*u*)= j(*u*)**

### Dijkstra's

Unvisited min distance to origin

**g(*u*)= cost so far to reach *u***

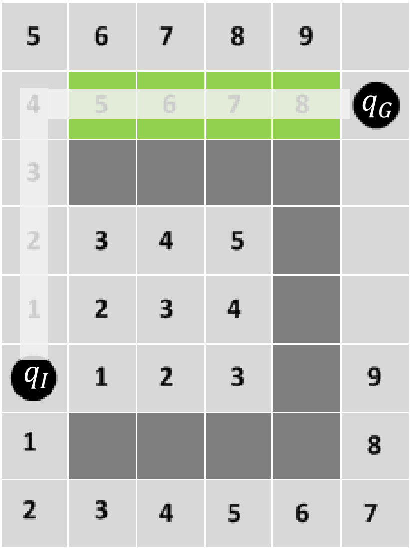
**$f_{score}$(*u*)=g(*u*)**

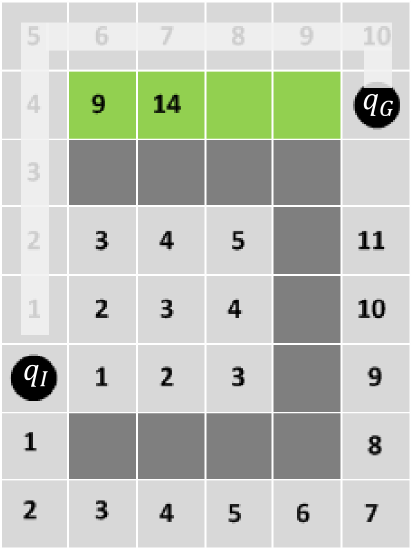
### Greedy Best First Search

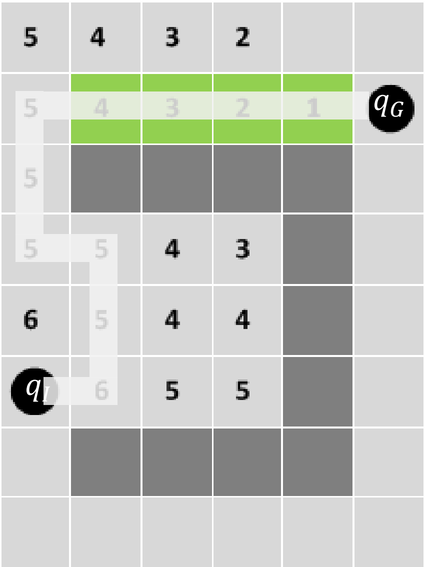
Unvisited min estimate distance distance to goal

**h(*u*)= heuristic estimate distance to the goal**

**$f_{score}$(*u*)=h(*u*)**



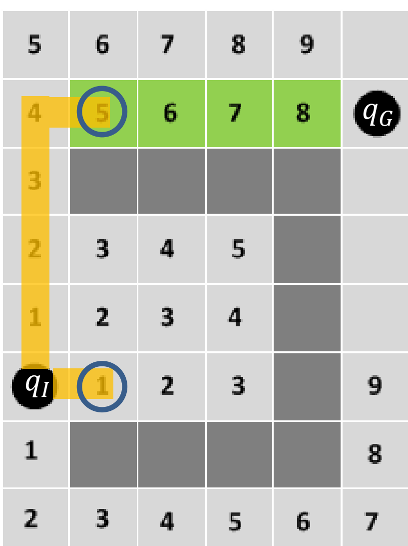



# Alorithm frontier selection

## Breadth-first search

Unvisited min jump

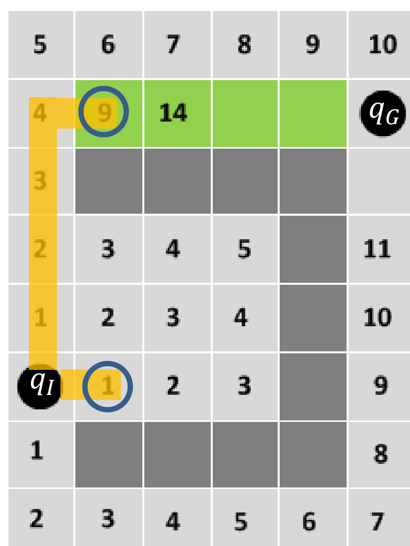
**j(*u*)= number of jump to reach *u***
**$f_{score}$(*u*)= j(*u*)**




## Dijkstra's

Unvisited min distance to origin

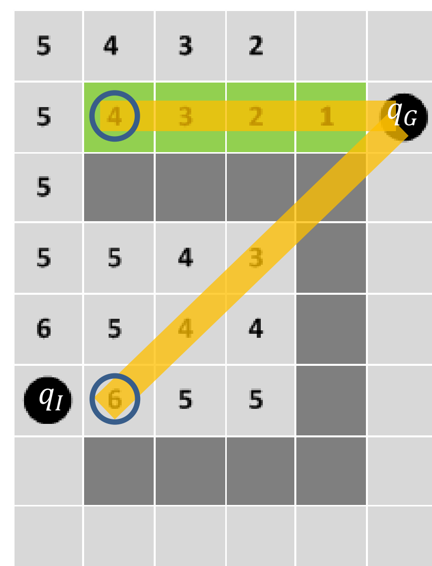
**g(*u*)= cost so far to reach *u***
**$f_{score}$(*u*)=g(*u*)**




## Greedy Best First Search

Unvisited min estimate distance distance to goal

**h(*u*)= heuristic estimate distance to the goal**
**$f_{score}$(*u*)=h(*u*)**



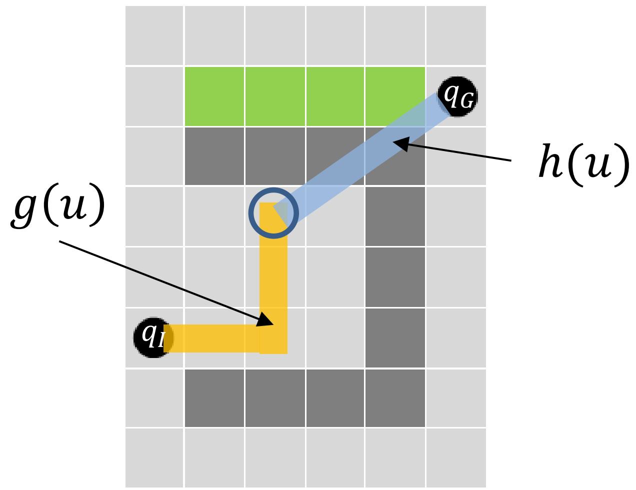
# A*

- ❑ Principle

Combine Dijkstra's ($g(u)$)and Greedy Best First

Search ($h(u)$) frontier selection

$$f_{score}(u) = g(u) + h(u)$$



# A*

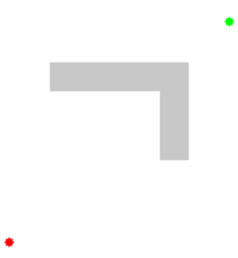
## ❑ Algorithm

$closedList = \{\emptyset\}$
$openList = \{q_I\}$
**For each** $C \in \mathcal{C}_{\text{free}}$
  $g_{score}[C] = +\infty$
  $f_{score}[C] = +\infty$
  $prev_{Node}[C] = \emptyset$
**While** $openList \neq \emptyset$
  $u = \min(f_{score})$
  **If** $u == q_G$
    $reconstructPath(u)$
  $remove\ u\ from\ \ openList$
  $add\ u\ to\ closedList$
  **For each** $v \in Neighbor(u)$
    **If** $v \in closedList$
      $continue$
    $v_{score} = g_{score}[u] + length(u, v)$
    **If** $v \notin openList$
      $add\ v\ to\ openList$
    **ElseIf** $v_{score} \geq g_{score}[v]$
      $continue$
    $prev_{Node}[v] = u$
    $g_{score}[v] = v_{score}$
    $f_{score}[v] = g_{score}[v] + heuristicCostEstimate(v, q_G)$

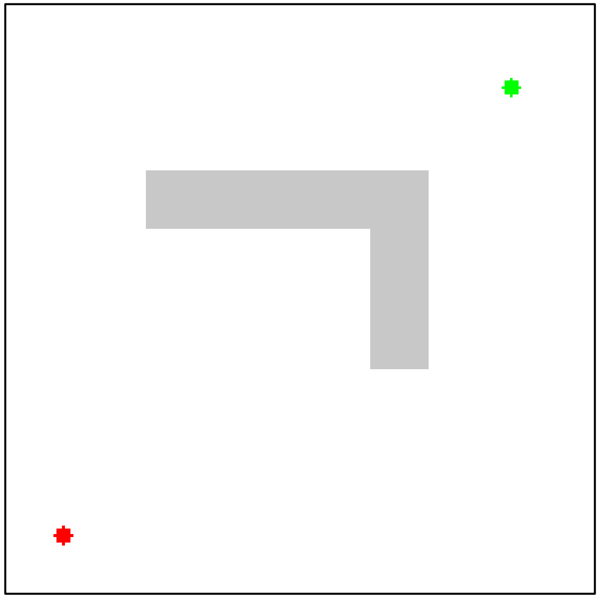
Return Failure

# A*

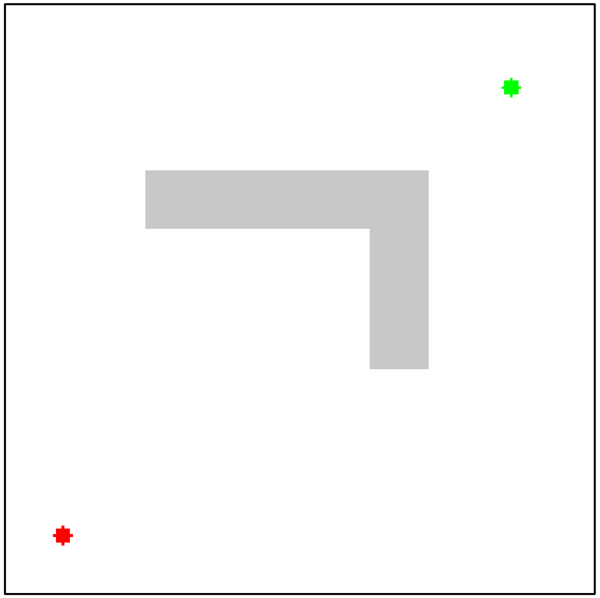
$closedList = \{\emptyset\}$
$openList = \{q_I\}$
**For each** $C \in \mathcal{C}_{\text{free}}$
    $g_{score}[C] = +\infty$
    $f_{score}[C] = +\infty$
    $prev_{Node}[C] = \emptyset$
**While** $openList \neq \emptyset$
    $u = \min(f_{score})$
    **If** $u == q_G$
        $reconstructPath(u)$
    $remove\ u\ from\ \ openList$
    $add\ u\ to\ closedList$
    **For each** $v \in Neighbor(u)$
        **If** $v \in closedList$
            $continue$
        $v_{score} = g_{score}[u] + length(u, v)$
        **If** $v \notin openList$
            $add\ v\ to\ openList$
        **ElseIf** $v_{score} \geq g_{score}[v]$
            $continue$
        $prev_{Node}[v] = u$
        $g_{score}[v] = v_{score}$
        $f_{score}[v] = g_{score}[v] + heuristicCostEstimate(v, q_G)$
**Return Failure**



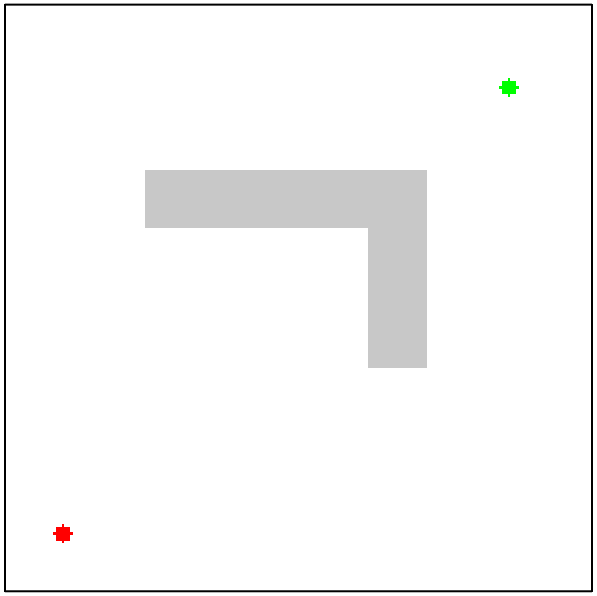
## Dijkstra's



## Greedy Best First Search

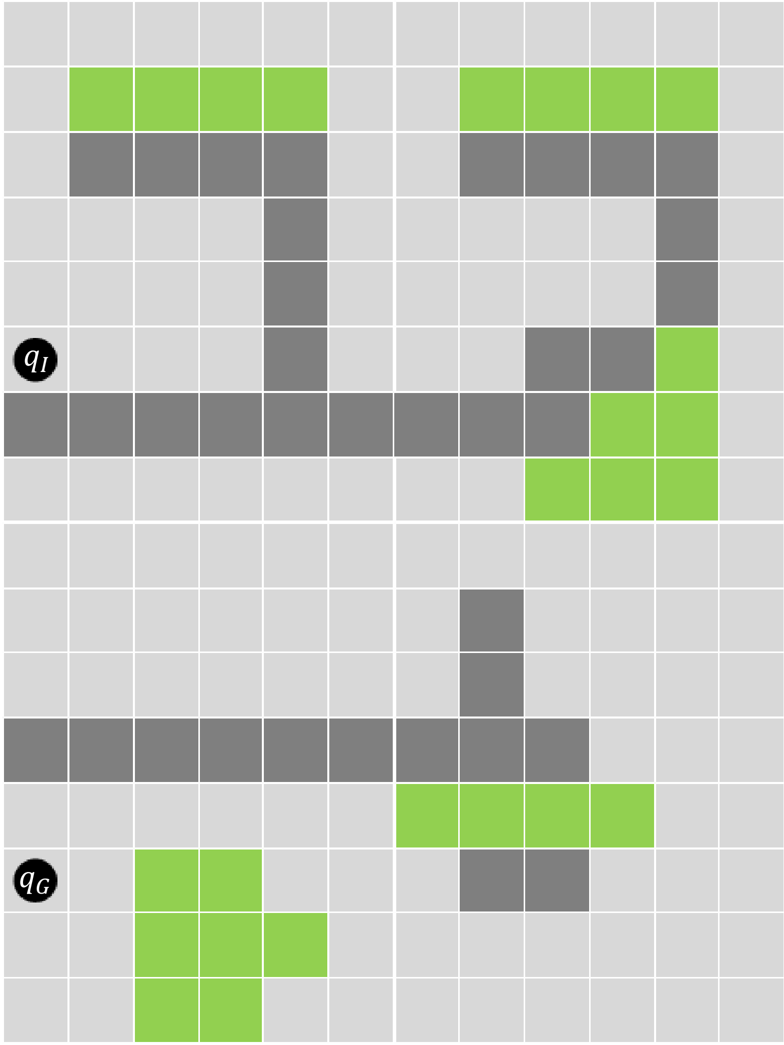


## A*

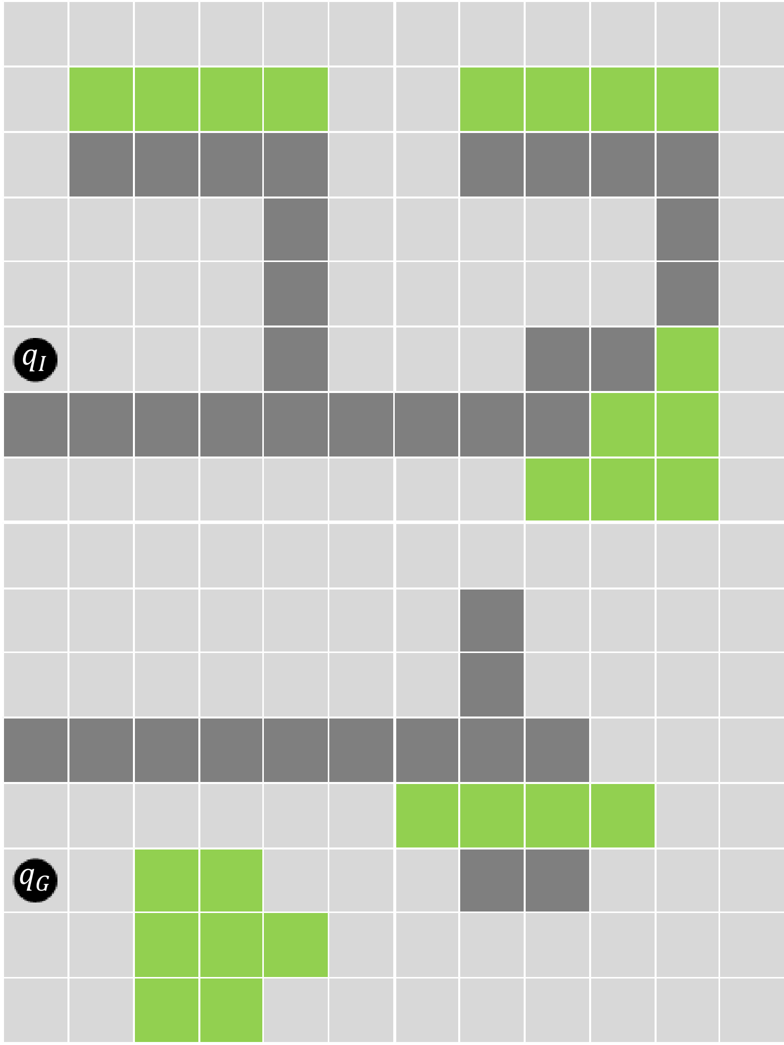


# Exercices

Wavefront

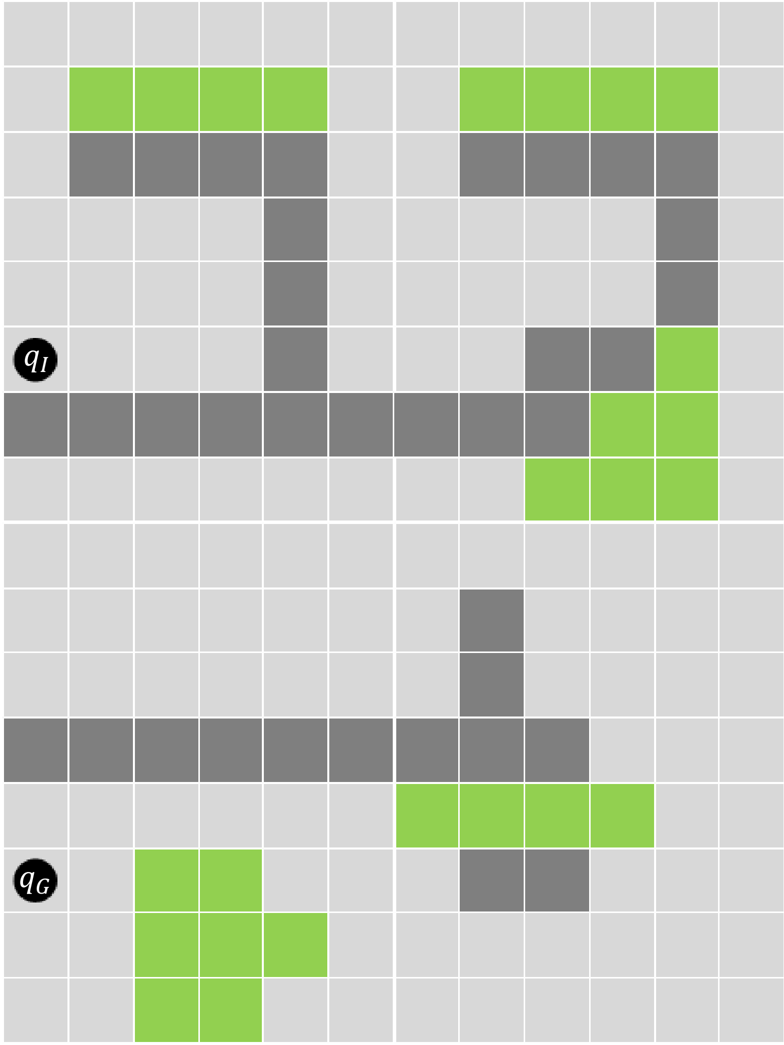



Dijkstra

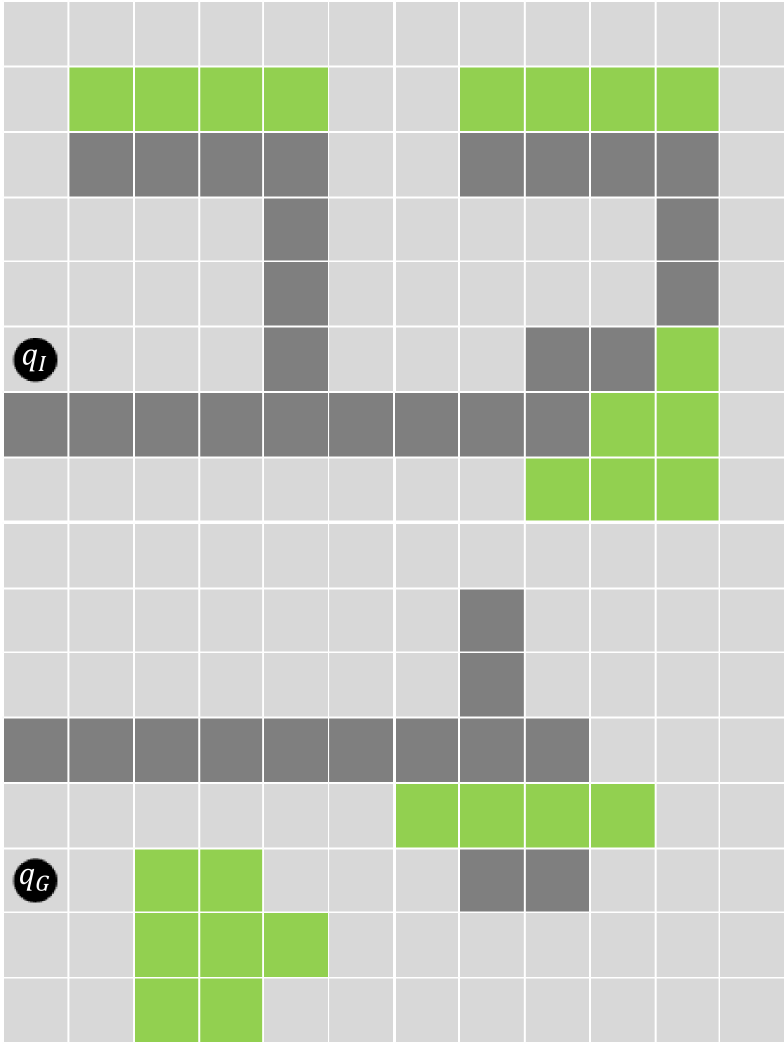


# Exercices

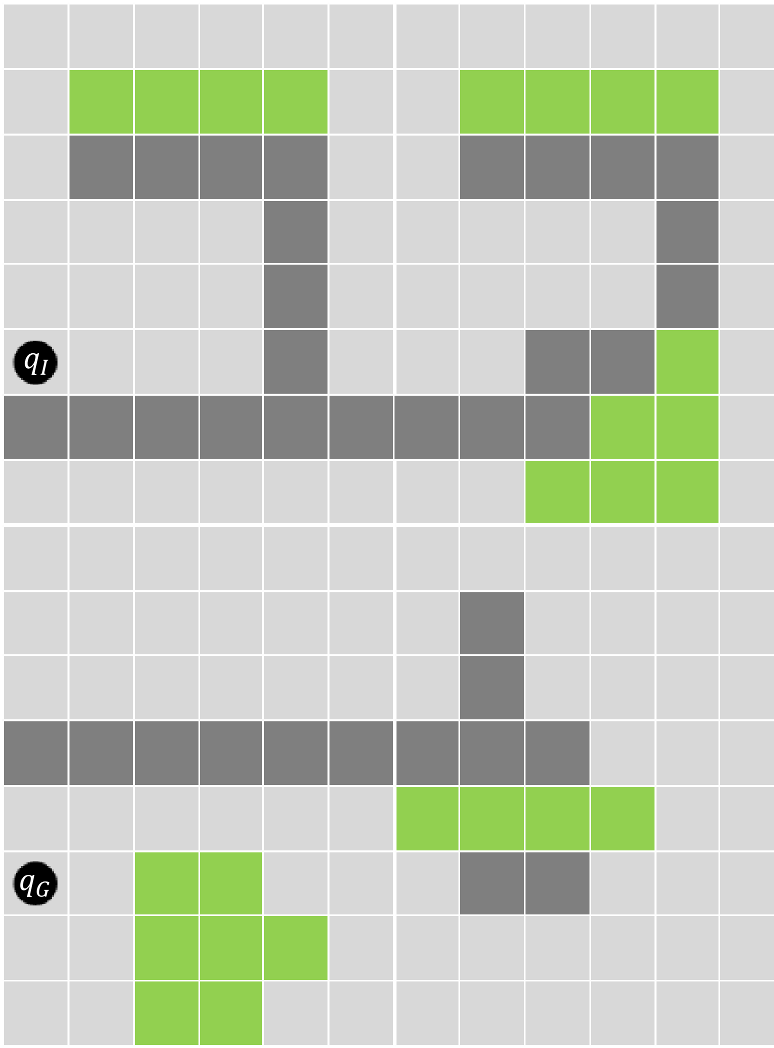
Greedy Best First Search

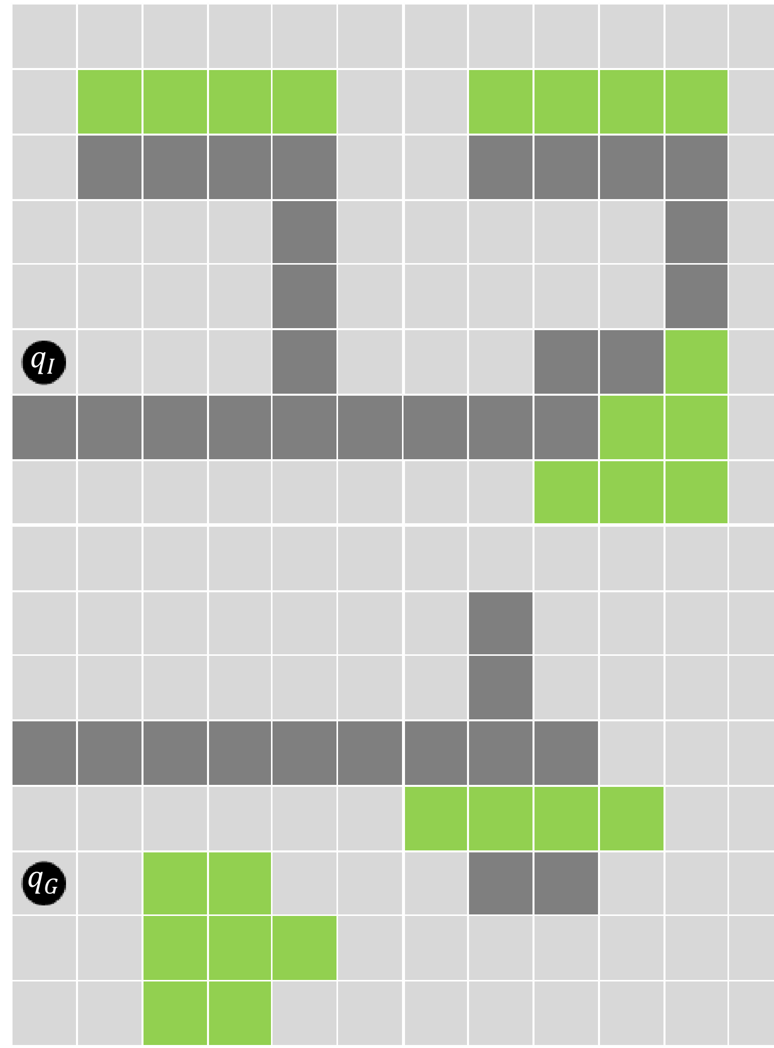



A*

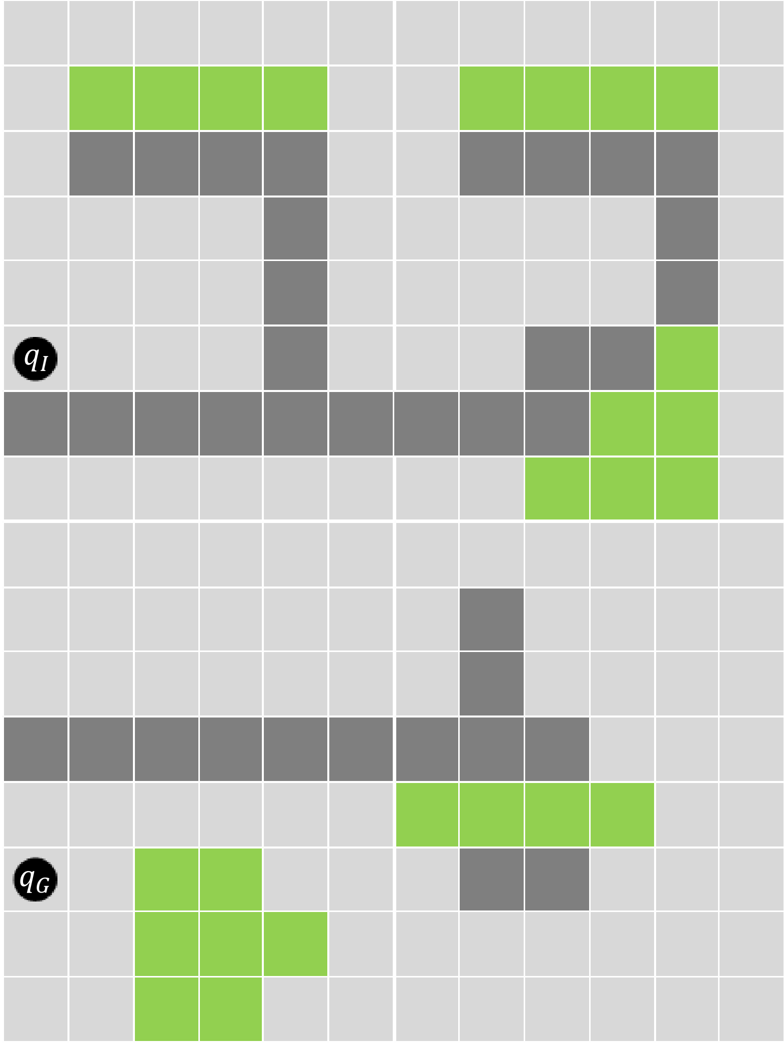


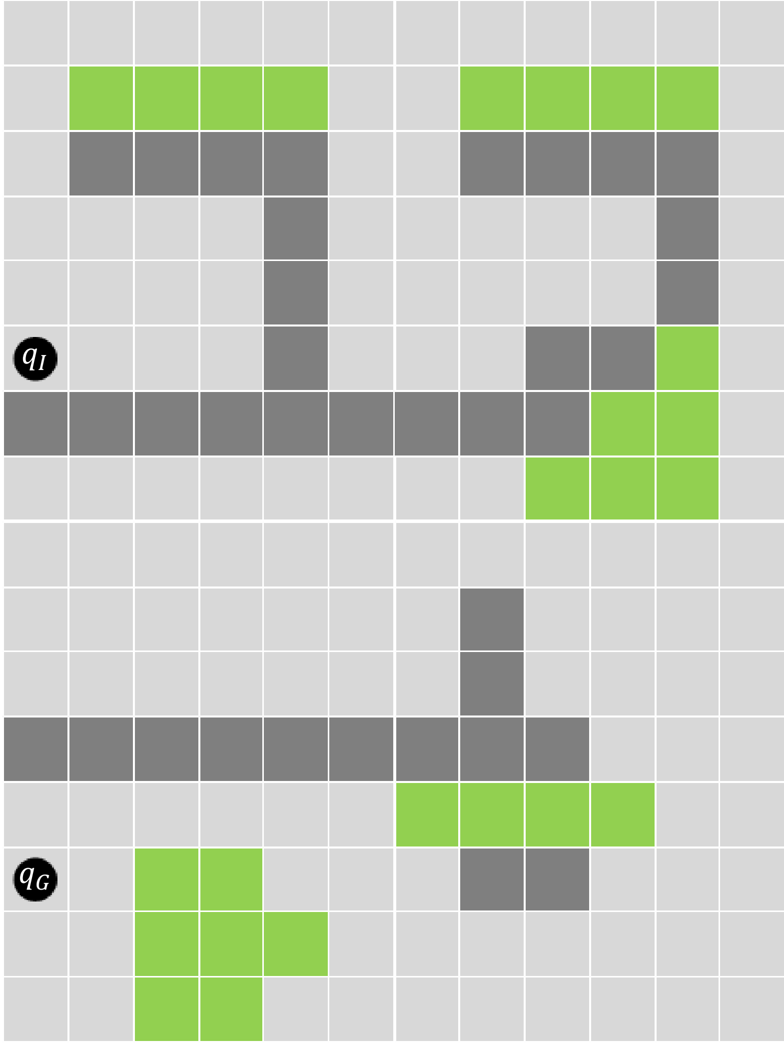
# Exercices



# Exercices



# References

# References (1/2)


- IEEE Standard 172-1983
- **Introduction to Autonomous Mobile Robots, MIT Press, Roland SIEGWART, Illah R. NOURBAKHSH 2004**
- **Robotics, Vision and Control, Springer, Peter Corke 2011**
- **PLANNING ALGORITHMS, Steven M. LaValle,University of Illinois, 2006**
  http://planning.cs.uiuc.edu/
- **Introduction to Mobile Robotics, Mapping with Known Poses, Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras, Uni Freiburg**
  http://ais.informatik.uni-freiburg.de/teaching/ss14/robotics/slides/08-occupancy-mapping-mapping-with-known-poses.pdf
- **Introduction to Mobile Robotics, Robot Motion Planning, Wolfram Burgard, cyrill stachniss, Maren Bennewitz, Kai Arras, Uni Freiburg, 2011**
  http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf
- **Occupancy Grids, Robotics, Benjamin Kuipers**
  https://www.cs.utexas.edu/~kuipers/slides/L13-occupancy-grids.pdf
- http://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html
- http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume11/fox99a-html/node10.html
- http://www.geometrylab.de/applet-30-en
- http://cs.smith.edu/~streinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/visibility.htm
- http://theory.stanford.edu/~amitp/GameProgramming/MapRepresentations.html
- https://fr.wikipedia.org/wiki/Diagramme_de_Vorono%C3%AF
- https://en.wikipedia.org/wiki/Voronoi_diagram
- http://www.barankahyaoglu.com/robotics/voronoirobot/
- https://en.wikipedia.org/wiki/Fortune%27s_algorithm
- http://msl.cs.uiuc.edu/rrt/index.html
- https://en.wikipedia.org/wiki/Probabilistic_roadmap
- http://msl.cs.uiuc.edu/rrt/gallery_rigid.html
- **http://www.redblobgames.com/pathfinding/a-star/introduction.html**
- https://en.wikipedia.org/wiki/A*_search_algorithm
- http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
- http://buildnewgames.com/astar/

# References (2/2)


- https://en.wikipedia.org/wiki/D*
- http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

images
- http://www.ros.org/news/2013/03/
- http://library.isr.ist.utl.pt/docs/roswiki/attachments/pr2_simulator(2f)Tutorials(2f)BasicPR2Controls/rviz_move_base_diverge.png
- http://www.youbot-store.com/developers/software/ros/youbot-ros-navigation-stack
- https://www.fsb.unizg.hr/acg/yaw_rate_estim_fig1.jpg
- http://library.isr.ist.utl.pt/docs/roswiki/costmap_2d.html
- http://joydeepb.com/Publications/biswas-rgbd11-plane-filtering.pdf
- http://www.cim.mcgill.ca/~mrl/pubs/saul/iros98.pdf
- http://www.sfbtr8.spatial-cognition.de/project/r3/HGVG/graphics/3DMzh.jpg
- http://www.mdpi.com/1424-8220/15/6/12736/htm
- http://a4academics.com/images/ProjSeminarImages/Ant-behavior-real-world.png

videos
- https://www.youtube.com/watch?v=A-fxij3zM7g
- https://www.youtube.com/watch?v=qziUJcUDfBc
- https://www.youtube.com/watch?v=zZLQ8Yh2iEE
- https://www.youtube.com/watch?v=DVnbp9oZZak
- https://www.youtube.com/watch?v=vAnN3nZqMqk
- https://www.youtube.com/watch?v=ylnH9GctlTA

Jacques Saraydaryan
Jacques.saraydaryan@cpe.fr