Marcello Restelli

Dipartimento di Elettronica e Informazione
Politecnico di Milano
email: restelli@elet.polimi.it
tel: 02-2399-3470

Path Planning

Robotica
for Computer Engineering students

A.A. 2006/2007

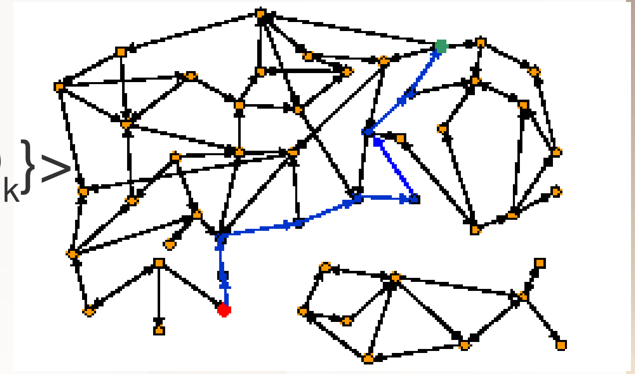
What is Required to Navigate?

- Cognition/Reasoning
  - the ability to decide what **actions** are required to achieve a certain **goal** in a given **situation**
  - decisions ranging from what **path** to take to what **information** on the environment to use
- Industrial robots operate **without cognition** since they work in **static** and very **structured** environments
- In mobile robotics cognition is mainly concerned to which path is better to reach a given location
  - knowledge is **partially known** and **uncertain**
  - requires hybrid solutions running in parallel
    - global path planning
    - local obstacle avoidance

Global Path Planning

- Assumption
  - there exists a good enough map
- Goal
  - find the optimal geometrical path
    - optimal = minimum distance, minimum rotations
- First step
  - use an environment **representation** that allows to apply standard planning algorithms
    - Road-Map
      - visibility graphs
      - Voronoi graphs
    - Cell decomposition
    - Potential Field
    - ...

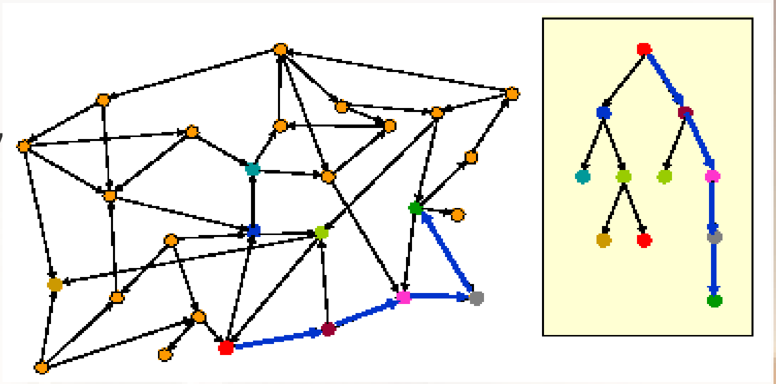
Path Planning as a Graph Search Problem



- For graph-based maps
  - Search problem <S, G, s_0, {O_1, O_2, ..., O_k}>
    - S: set of states
    - G: goal states
    - $s_0 \notin$ G: initial state
    - {O_1, O_2, ..., O_k}: finite set of available operators. If an operator O_j may be applied to a state it returns the state $Succ_j(s)$ with a cost $c_j(s) \geq 0$
  - Solution
    - each sequence that allows to pass from s_0 to a goal state
  - Optimal solution
    - a solution is optimal if its cost is not greater than that of any other solution

Search Solution Methods

- To solve a search problem we need to explore a portion (possibly small) of the state space
- We need to build a search tree



The general algorithm is

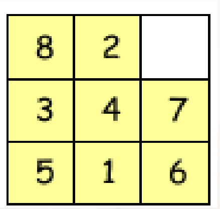
```
Initialize root node with the initial state
REPEAT
    IF there is not any node to be extended THEN
        RETURN failure
    ELSE
        choose a node n
        IF n is a goal state THEN
            RETURN solution
        ELSE
            extend n and add the produced nodes to the search tree
```

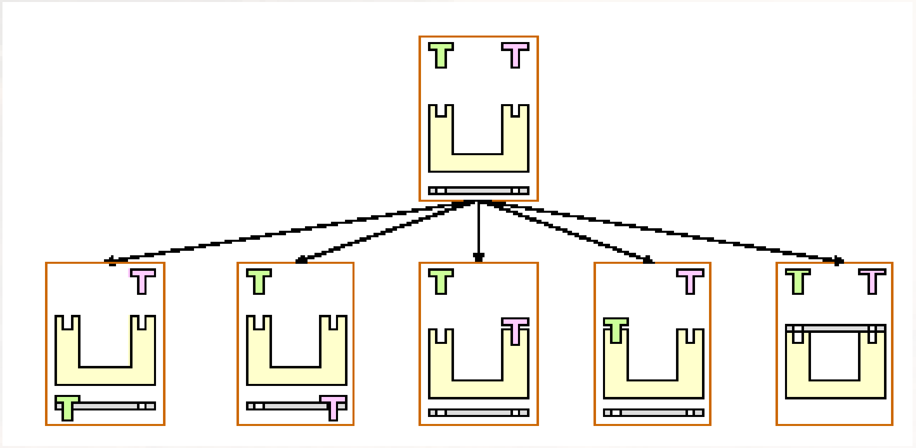
Evaluation of Different Search Strategies

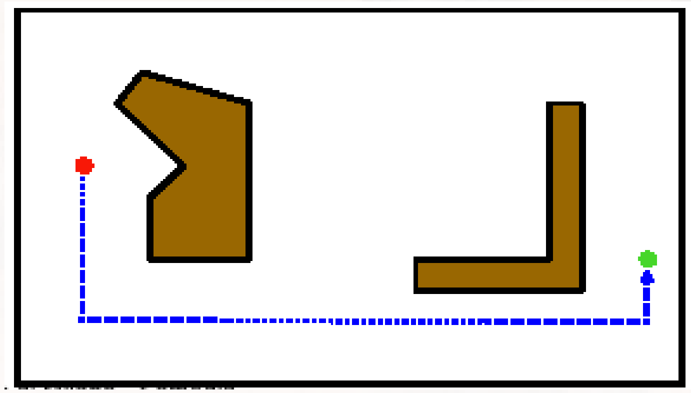
- The main difference between the different search strategies is the way in which they put new nodes into the frontier (the set of nodes that have been generated but still not expanded)
- Properties of a search strategy
  - completeness: it found a solution if at least one exists
  - optimality: the found solution has minimum cost
  - complexity: time and memory requirements
- Parameters of a search strategies
  - b: branch factor, i.e. max number of successor states
  - d: depth of the goal state nearest to the root node

What Search Methods can be Used for?

- Good solution methods exist for problems
  - static
  - full observable
  - discrete
  - deterministic
- Examples
  - Eight game
  - automatic assembly
  - path planning







Search Strategies

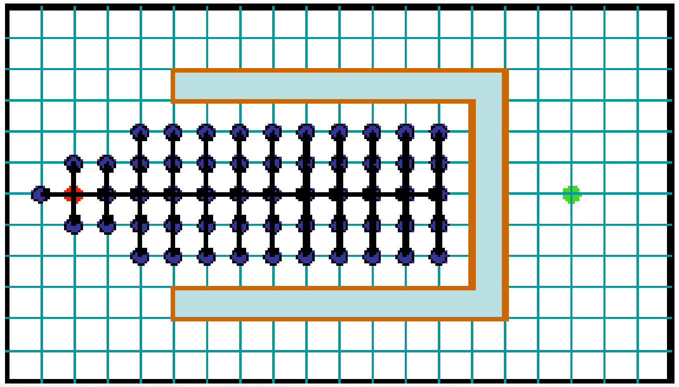
- Uninformed search strategies
  - breadth-first
  - uniform-cost
  - depth-first
  - depth-limited
  - iterative deepening depth first
  - bidirectional
- Informed search strategies
  - best-first greedy
  - A*

Informed Search Strategies

- Informed search strategies exploit specific knowledge that is not contained in the problem formulation
- The next node to be evaluated is chosen according to an evaluation function *f(n)*
- Evaluation function tells how much a node is promising
- According to the way *f(n)* is computed we have different search strategies
- For convention nodes with low values for *f(n)* are better
  - it is typically implemented through a priority queue
  - the node with the lowest value is extracted to be extended

Best-First Greedy Search

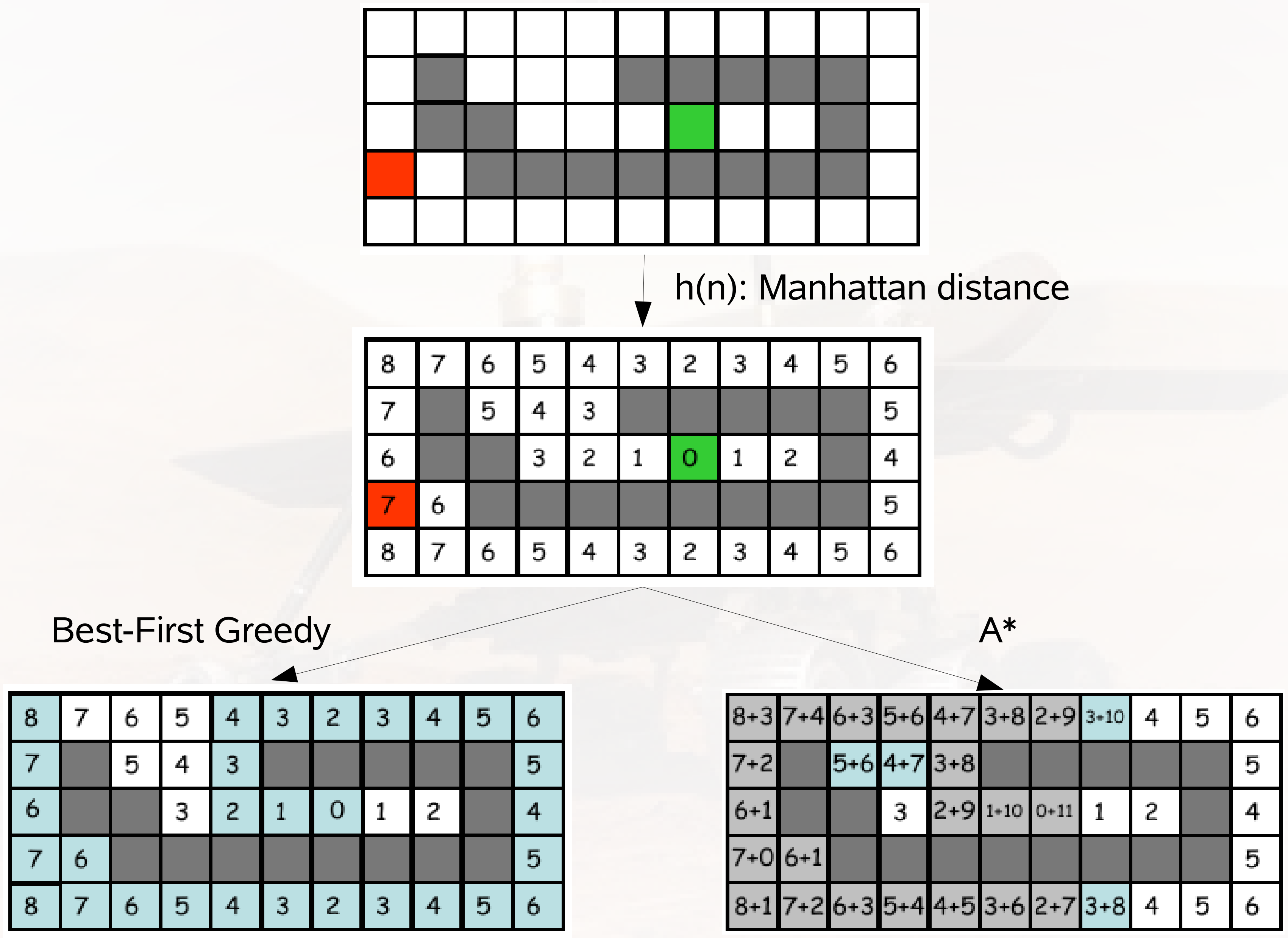
- The evaluation function is equal to a heuristic function *h(n)*
- *f(n) = h(n)* = estimated cost of the shortest path between node *n* and one of the goal nodes
- *h(n)* is an estimate not the real cost (otherwise is trivial)
- Best-first greedy search is
  - not optimal
  - suffers from local minima
  - temporal and spatial complexity
    - $O(b^m)$
      - b is the branch factor
      - m is the maximum search depth (may be ∞)



A* Search

- A* is a kind of best-first search
- The evaluation function is the sum of the cost to reach the node *n* (*g(n)*) and the value of a heuristic function *h(n)*
  - *f(n) = g(n) + h(n)*
- *f(n)* is an estimate of the most convenient solution among those the pass through *n*
  - *g(n)*: cost of the path from the root node to *n*
  - *h(n)*: estimate of the cost of the path from *n* to the nearest goal node
- A* is
  - complete
  - optimal (under some hypotheses on *h(.)*)
  - optimally efficient

Best-First Greedy vs A*: An Example



	▨				▨	▨	▨	▨	▨	
	▨	▨				🟩			▨	
🟥		▨	▨	▨	▨	▨	▨	▨	▨	

8	7	6	5	4	3	2	3	4	5	6
7	▨	5	4	3	▨	▨	▨	▨	▨	5
6	▨	▨	3	2	1	0	1	2	▨	4
7	6	▨	▨	▨	▨	▨	▨	▨	▨	5
8	7	6	5	4	3	2	3	4	5	6

Best-First Greedy

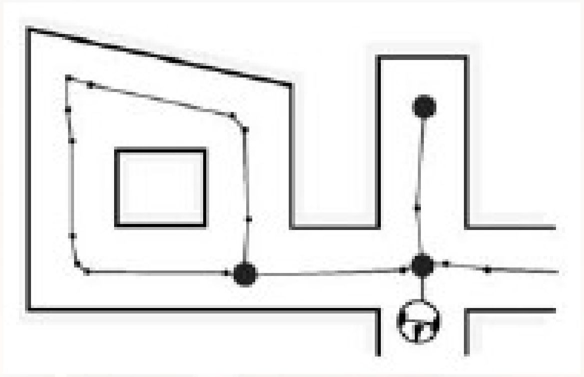
8	7	6	5	4	3	2	3	4	5	6
7	▨	5	4	3	▨	▨	▨	▨	▨	5
6	▨	▨	3	2	1	0	1	2	▨	4
7	6	▨	▨	▨	▨	▨	▨	▨	▨	5
8	7	6	5	4	3	2	3	4	5	6

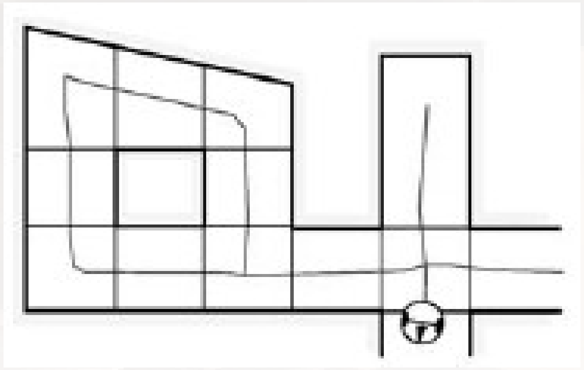
A*

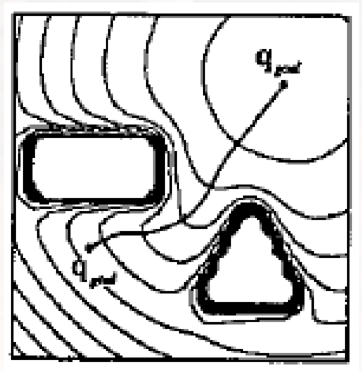
8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2	▨	5+6	4+7	3+8	▨	▨	▨	▨	▨	5
6+1	▨	▨	3	2+9	1+10	0+11	1	2	▨	4
7+0	6+1	▨	▨	▨	▨	▨	▨	▨	▨	5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

Path Planning Overview

- Road Map, Graph construction
  - identify a set of routes within the free space
    - where to put the nodes?
- Cell decomposition
  - discriminate between free and occupied cells
    - where to put the cell boundaries
- Potential field
  - imposing a mathematical function over the space

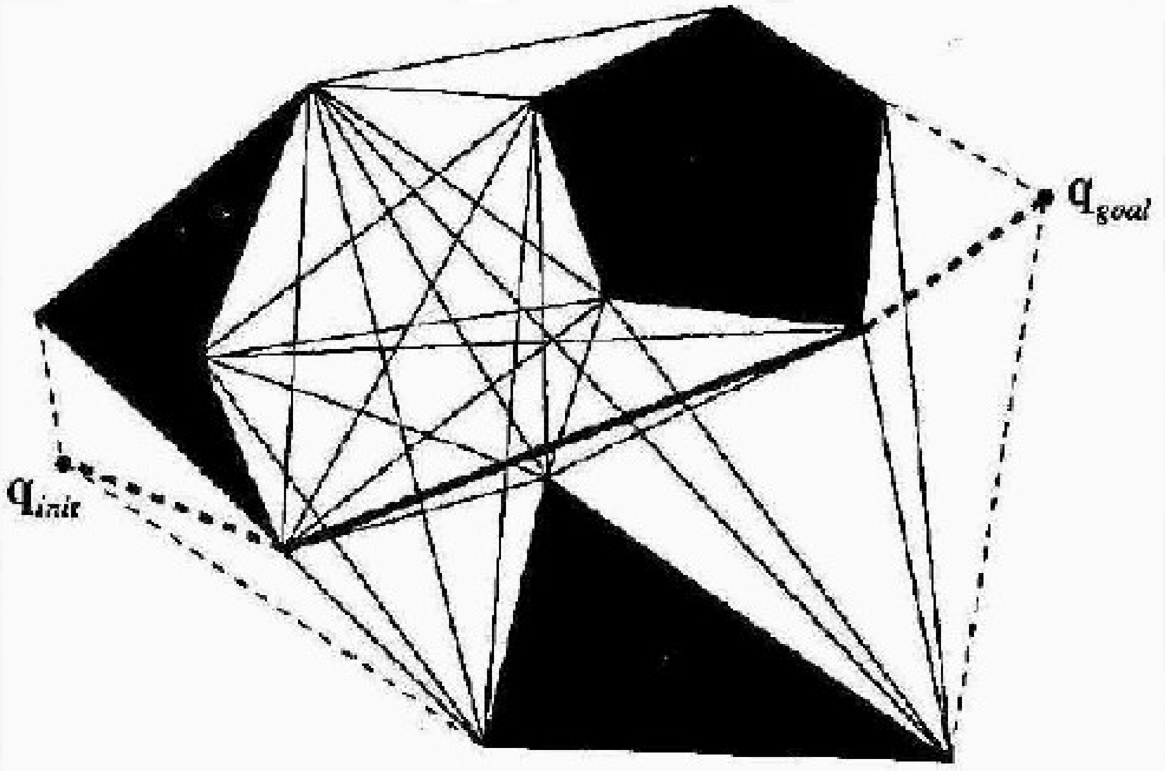






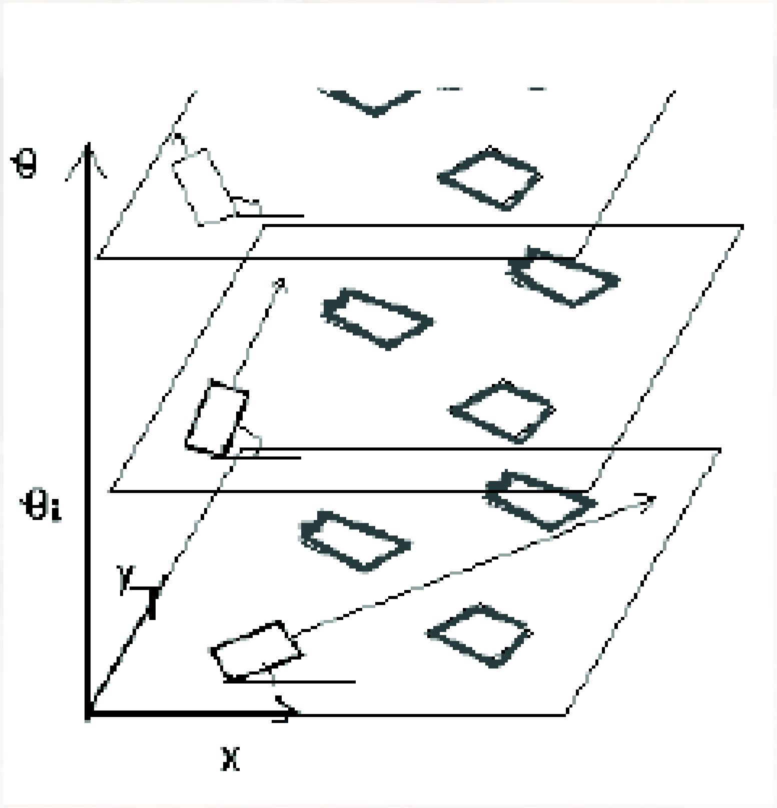
Visibility Graph

- Shortest path length
- Grow obstacles to avoid collisions



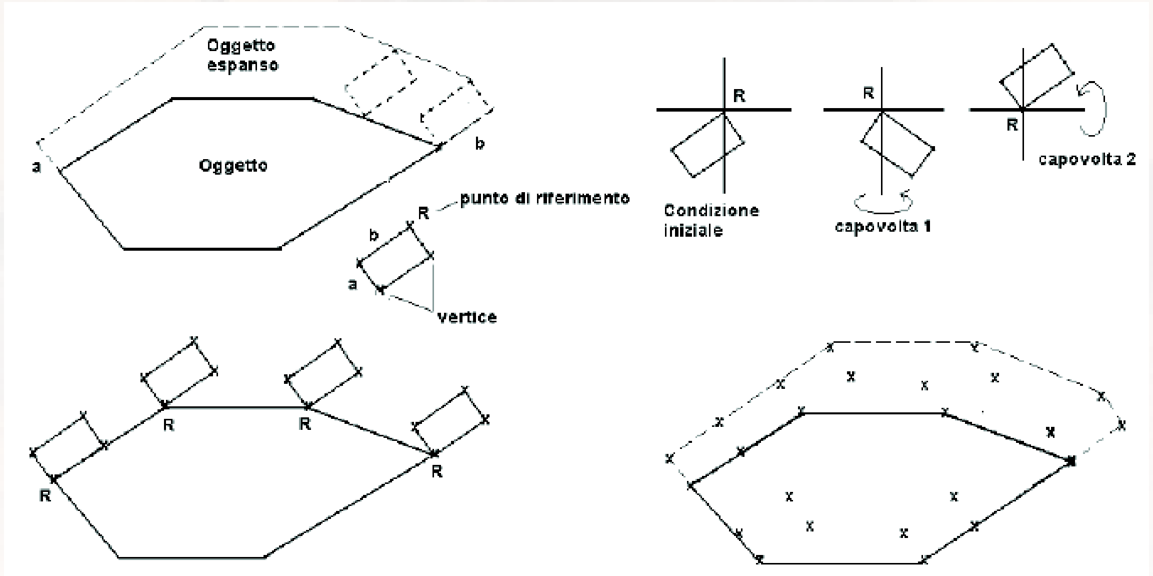
Obstacle Oriented Maps

- Obstacles are stored as a graph of vertexes
- Grow obstacles to avoid them
  - use the radius if the robot is cylindrical
  - use half-diagonal if the robot is rectangular
- This works if there are wide free zones
- Otherwise we need different maps according to the heading of the robot



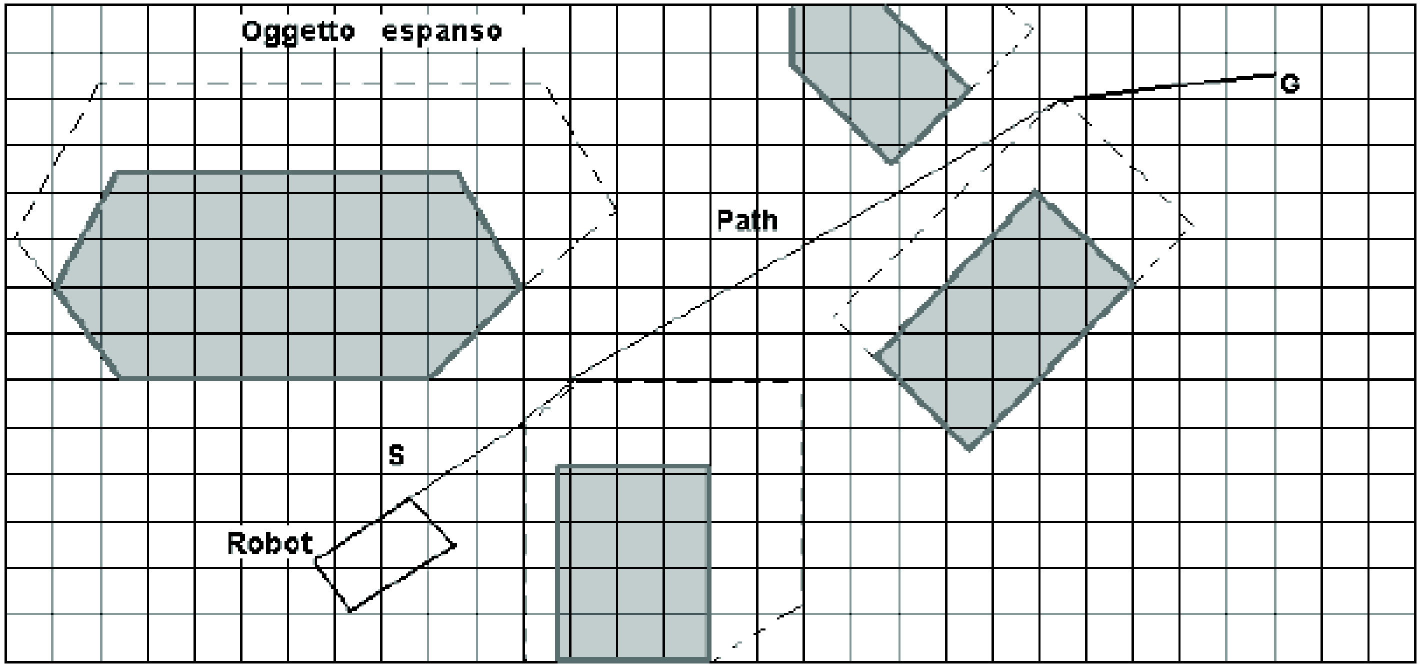
Growing Obstacles

- Consider a point R among the vertexes of the polygon that defines the robot shape
- Consider the polygon obtained through central symmetry with respect to the point R
- Make R overlap with all the vertexes of each obstacle and add the obtained robot vertexes to the list of vertexes
- Join all the external vertexes (obtaining the convex hull)



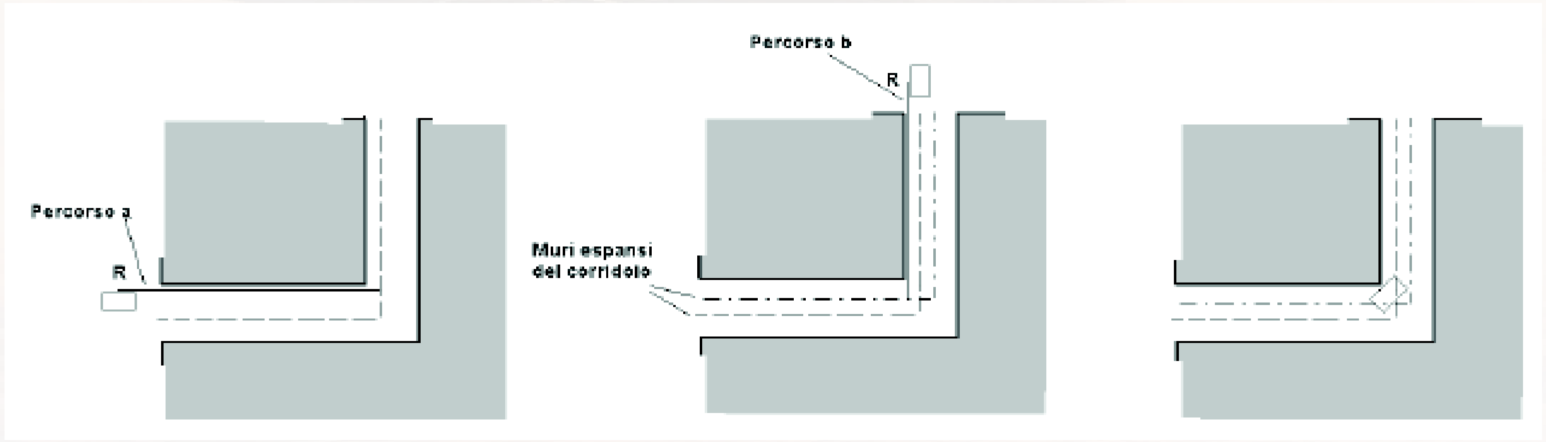
Planning with Visibility Graphs

- Once grown the obstacle for a given heading, we build the visibility graph
- The graph contains all the paths between start and goal
- Planning can be made with the A* algorithm
- The visibility graph is not suitable for dynamic words
- The paths are close to the obstacles



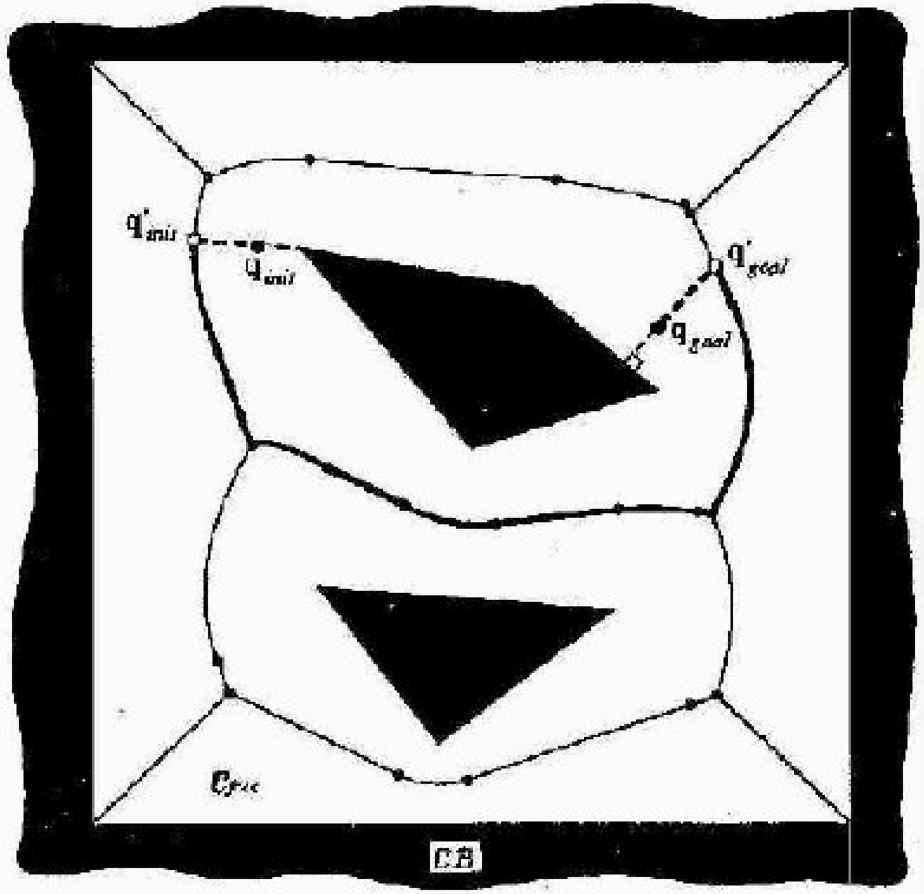
Piano Mover Problem

- Not always the robot can move straight and then perform rotations
- In the example below, the robot can find a path in the first hallway, but without changing its heading there is not a path in the second one



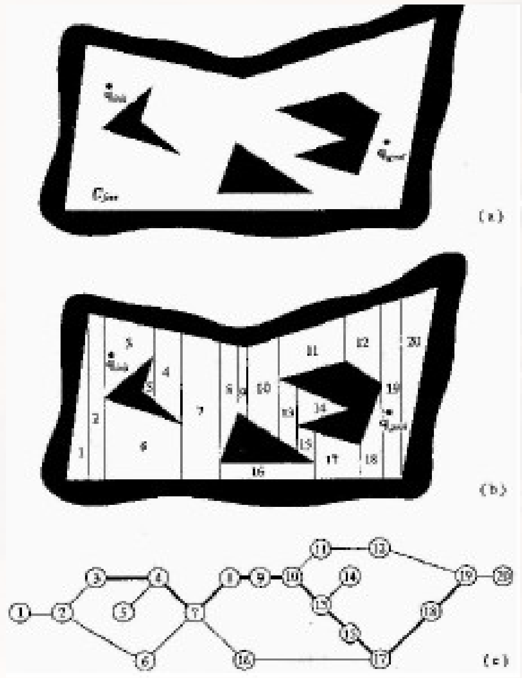
Voronoi Diagram

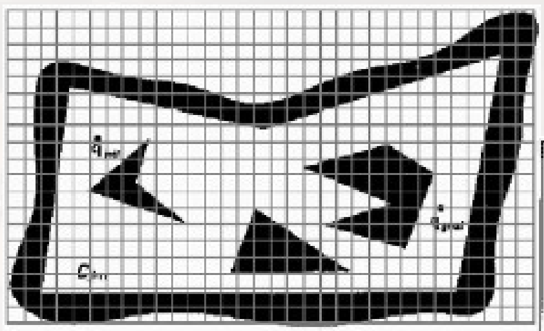
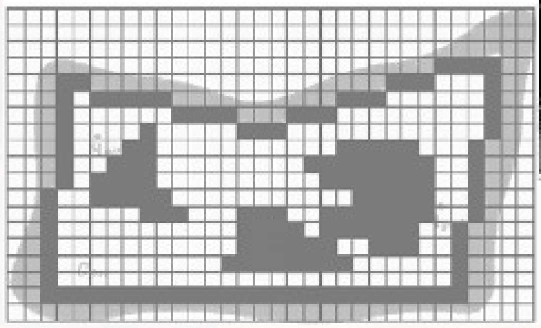
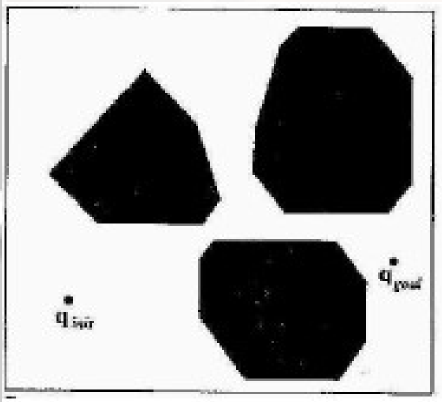
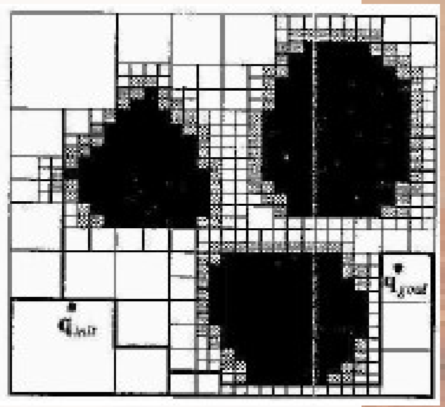
- Easy executable: maximize the sensor readings
- Works also for map-building: Move on the Voronoi edges



Cell Decomposition

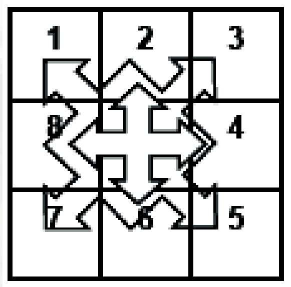
- Divide space in simple, connected regions called cells
- Construct a connectivity graph between adjacent cells
- Find a path the connect the cells containing the initial and goal configuration
- Given the sequence of cells compute a path within each cell

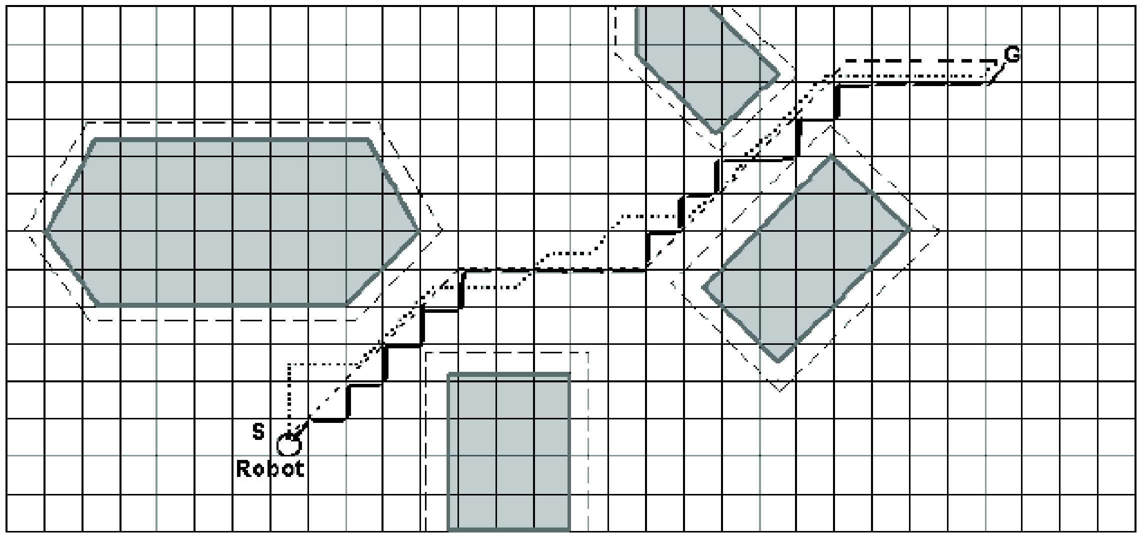


Planning with Regular Grid Maps

- The grid allows to represent both obstacles and free space
- Consider a regular grid
  - its dimension is given by the dimension of the robot
- The robot may be considered either inside the cell or on a vertex
- Planning can be made with A*
  - the successor function can be
    - 8-connected map
    - 4-connected map
      - orthogonal
      - oblique



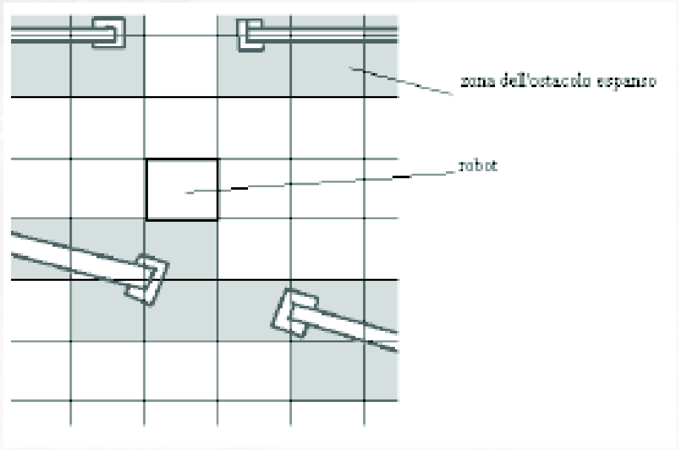


Planning with Regular Grid Maps: Wave Propagation

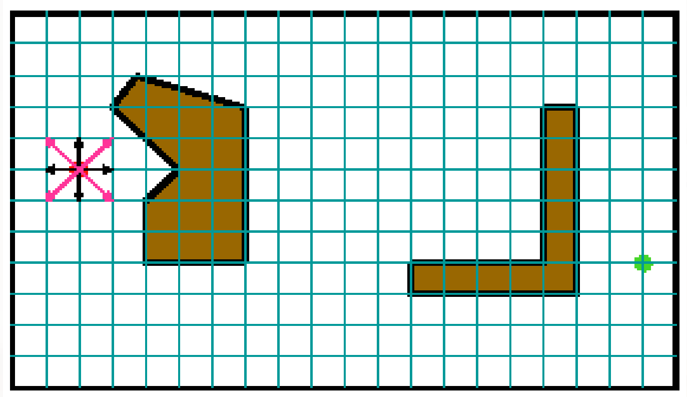
- Instead of A*, we can use an ad hoc method
- The idea is to compute for each cell its distance to the goal
- The algorithm propagate the distance information as a wave that starts from the goal
- If there is a path, the wave will reach the starting cell
- The algorithm (sketch)
  - initialize the cell values (blocked = inf., goal=0, free=row*col)
  - REPEAT
    - Forward step (from left to right, from up to bottom, if cell c is free)
      - c[x,y]:=min(c[x,y], c[x-1,y-1]+1, c[x-1,y]+1,c[x-1,y+1]+1,c[x,y-1]+1)
    - Beckward step (from right to left, from bottom to up, if cell c is free)
      - c[x,y]:=min(c[x,y],c[x+1,y-1]+1,c[x+1,y]+1,c[x+1,y+1]+1,c[x,y+1]+1)
  - UNTIL no change

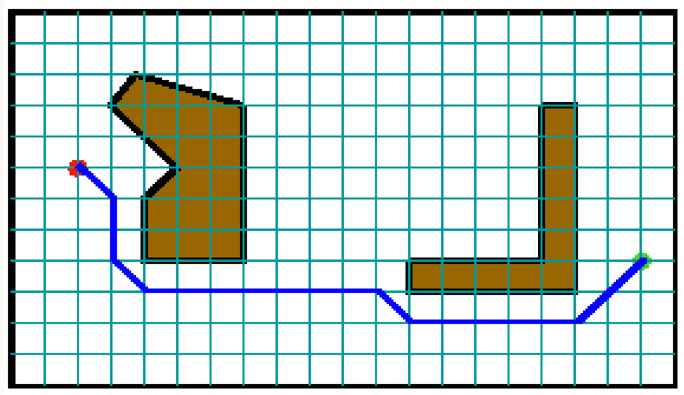
Wave Propagation: Considerations

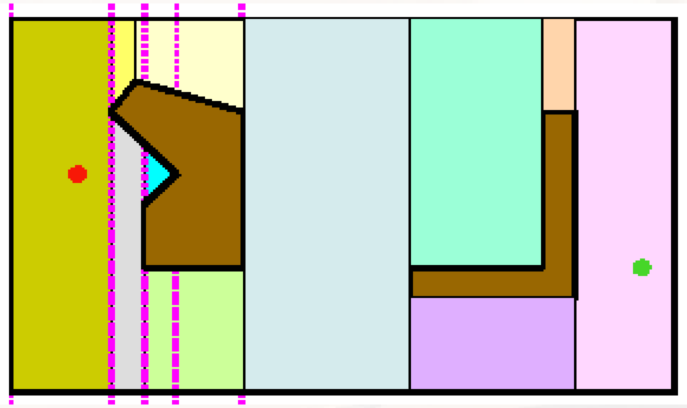
- For the diagonal movement the cost could be 1.4
- The number of iterations depends on the obstacle shape
- The path from start to goal is obtained by following the cells with decreasing value
  - if the path cannot be found, it does not exist
- The accuracy is in trade-off with the memory requirements
- Works in a discretized space
- The path planning algorithm is easy
- it does not suffer from local minima

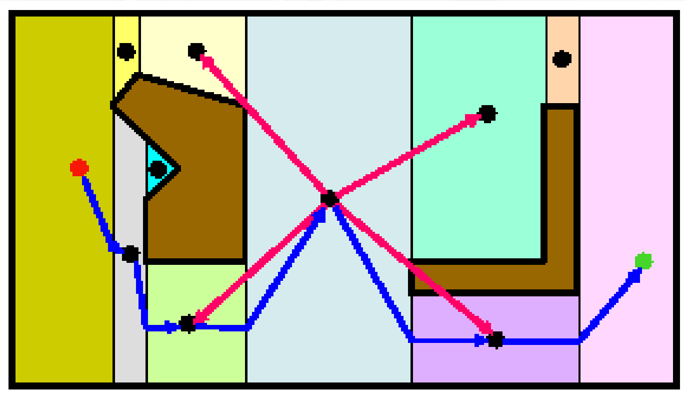


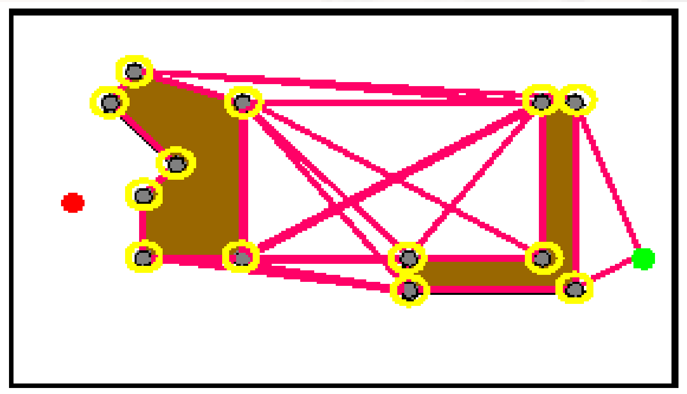
Planning with Different Representations

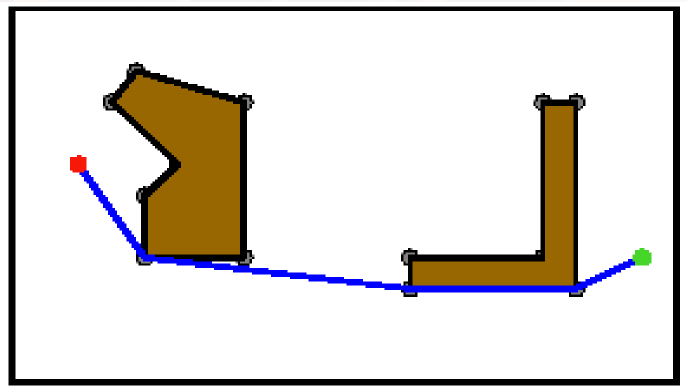








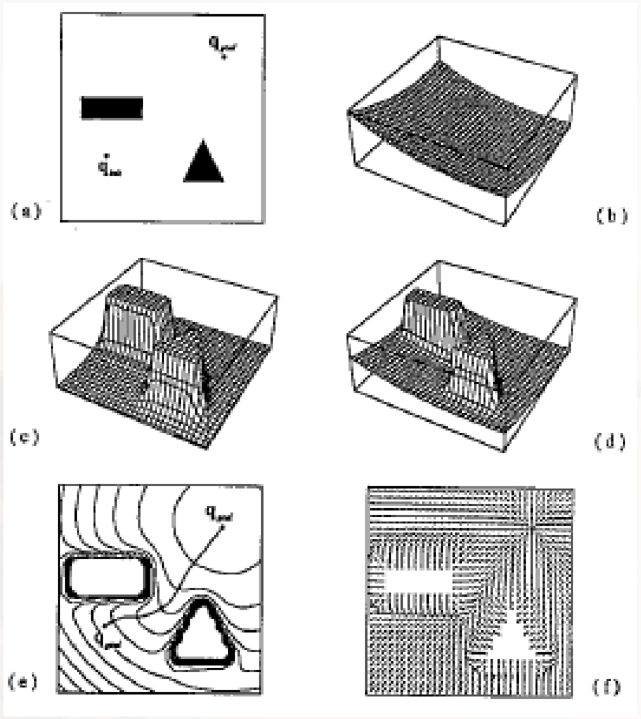




Potential Field

- Robot is treated as a point under the influence of an artificial potential field
  - goal generates attractive force
  - obstacles are repulsive forces
  - summing up the fields: *U(q)*
  - functions must be differentiable
- Generate artificial force field *F(q)*

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$



- Set robot speed (v_x, v_y)
  - the force drives the robot to the goal
  - the robot is considered as a point mass

Potential Field: An Example

- Attractive
  - parabolic function: the distance from the goal

$$U_{att}(q)=\frac{1}{2}k_{att}(q-q_{goal})^2$$

  - the force converges linearly towards 0
- Repulsing
  - strong when close to the obstacles
  - not influence if far from the obstacles

$$U_{rep}(q)=\begin{cases}\frac{1}{2}k_{rep}\left(\frac{1}{q-q_{obs}}-\frac{1}{\rho_0}\right)^2 & if\ q-q_{obs}\leqslant\rho_0\\ 0 & if\ q-q_{obs}\geqslant\rho_0\end{cases}$$

Potential Field: Issues

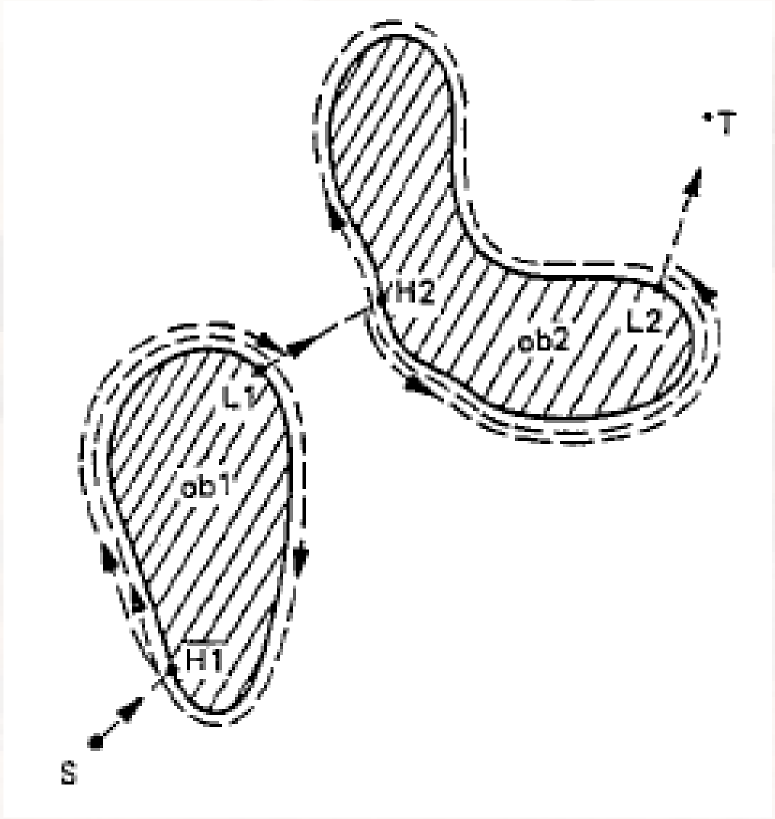
- Suboptimal paths
  - extended potential field method
    - rotational potential field
      - force is also a function of the orientation to the obstacle
    - task potential field
      - filters-out tasks that should not influence the movement
- Local minimum problem exists
  - harmonic potentials
    - robot moving as a fluid particle following its stream
    - no local minima, but complex

Obstacle Avoidance

- The goal is to avoid collisions with obstacles
- It is usually based on local maps
- Often is implemented as an independent task
- Efficient obstacle avoidance should be optimal w.r.t.
  - the overall goal
  - the actual speed and kinematics of the robot
  - the on-board sensors
  - the actual and future risk of collisions

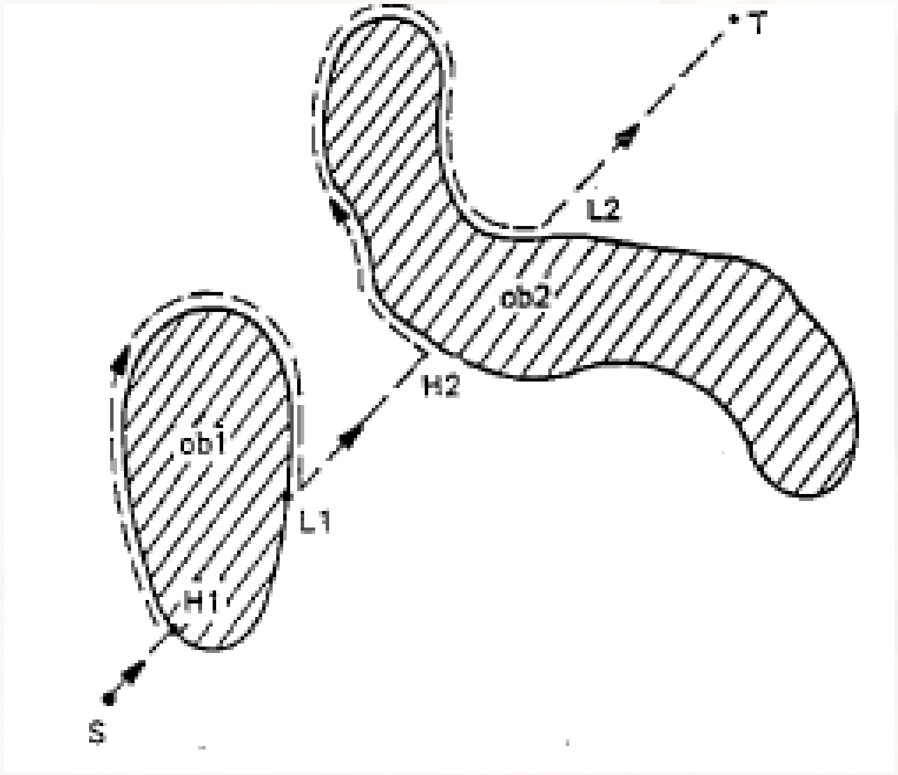
Obstacle Avoidance: Problem 1

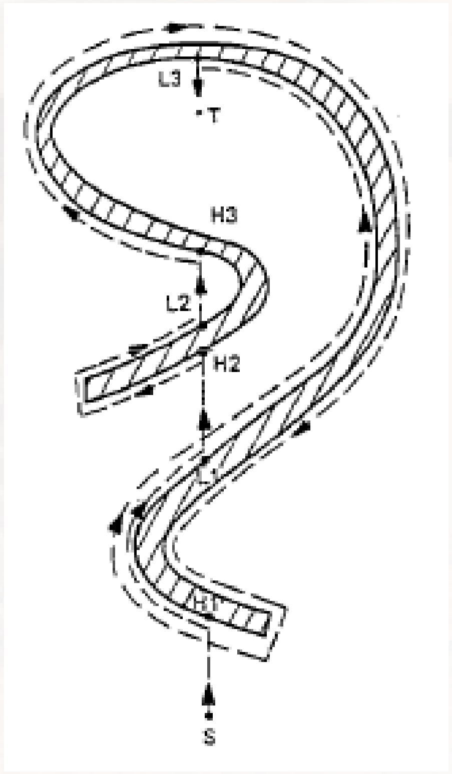
- Following along the obstacle to avoid
- To know which is the point closest to the goal, it is required to fully circle each encountered obstacle



Obstacle Avoidance: Problem 2

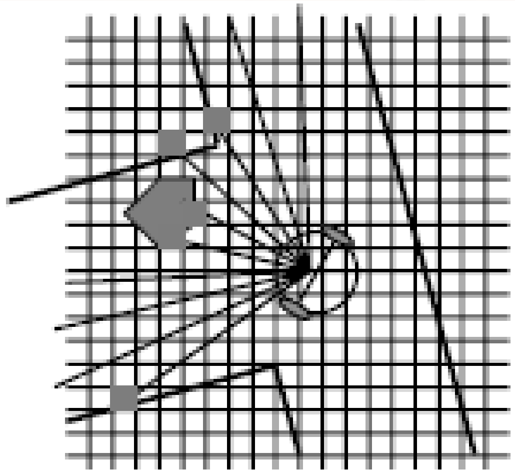
- Following the robot always on the left or right side
- Leaving the obstacle if the direct connection between start and goal is crossed

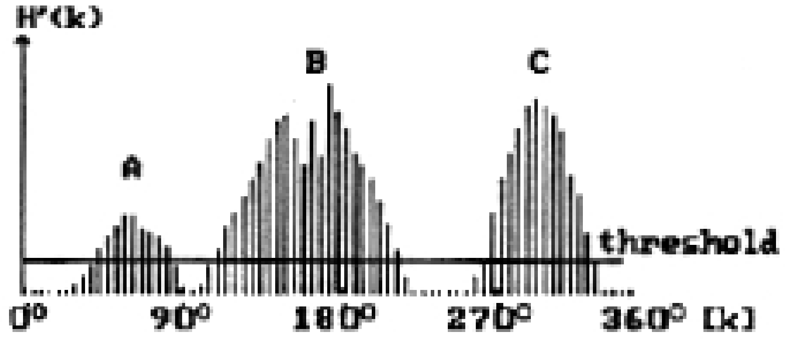




VHF: Vector Field Histogram

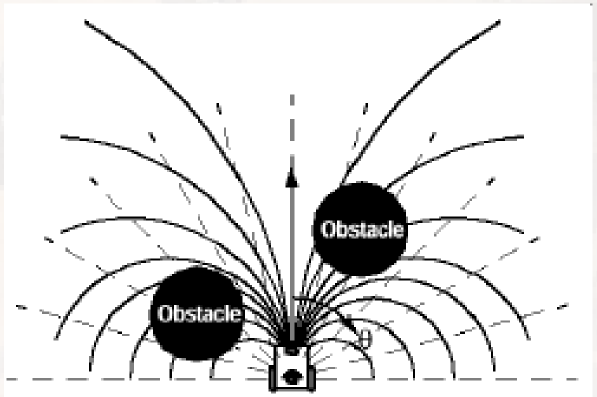
- Environment represented in a grid (2 DOF)
  - cell values equivalent to the probability that there is an obstacle
- Reduction in different steps to a 1 DOF histogram
  - calculation of steering directions
  - all openings for the robot to pass are found
  - the one with the lowest cost function G is selected
    - G = a*target_dir + b*wheel_orientation + c*previous_dir

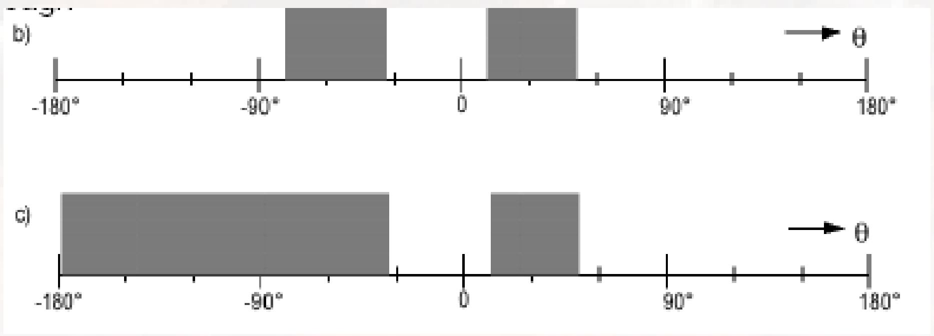




VHF+

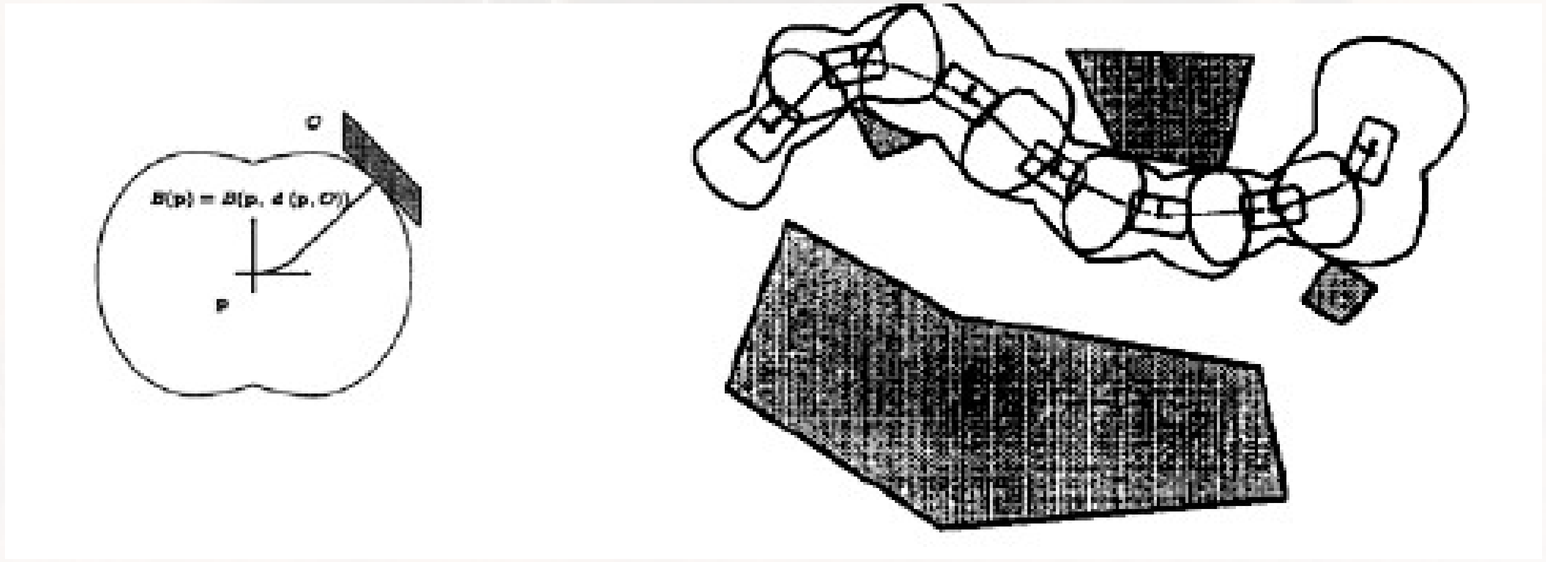
- Improved version of VHF that accounts also (in a very simple way) for the moving trajectories
  - the robot is assumed to move on arcs
  - obstacle blocking a given direction also blocks all the trajectories going through this direction





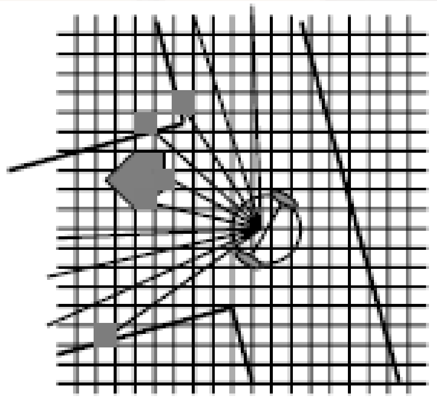
Bubble Band Concept

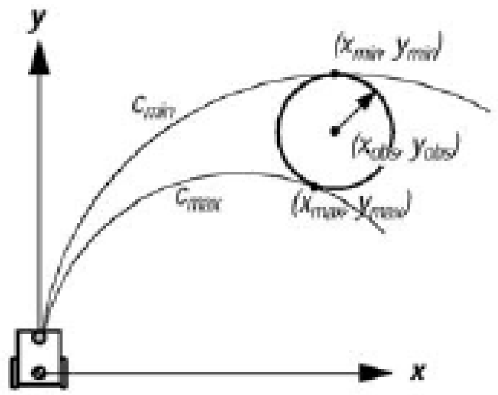
- A **bubble** is the maximum free space which can be reached without any risk of collision
  - generated using the distance to the object and a simplified model of the robot
  - bubbles are used to form a band of bubbles which connects the start point with the goal point



Basic Curvature Velocity Methods

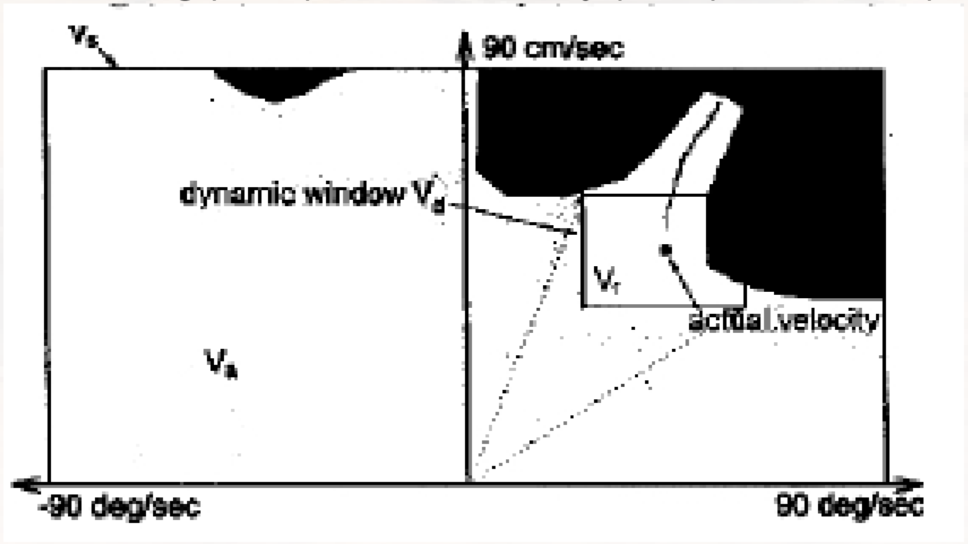
- Adding physical constraints from the robot and the environment on the velocity space (v,ω) of the robot
  - assumption that the robot is traveling on arcs (exist extensions using lanes)
    - $c = \omega/v$
  - acceleration constraints
  - obstacle constraints
    - obstacle are transformed in velocity space
  - objective function to select the optimal speed





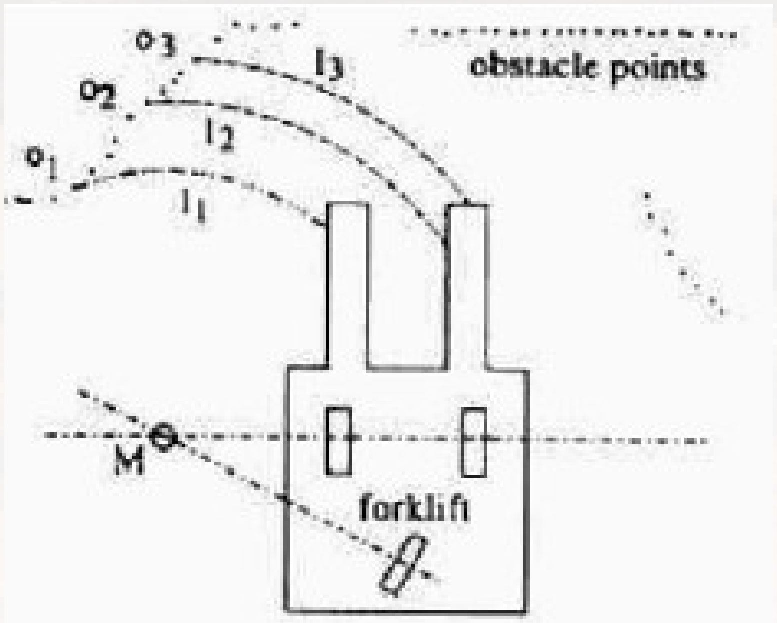
Dynamic Window Approach

- The kinematics of the robot is taken into account by searching a well chosen velocity space
  - velocity space -> some sort of configuration space
  - robot is assumed to move on arcs
  - ensures that the robot comes to stop before hitting an obstacle
  - objective function is chosen to select the optimal velocity
    - O = a*heading(v,ω) + b*velocity(v,ω) + c*dist(v,ω)



The Schlegel Approach

- Variation of the dynamic window approach
  - takes into account the shape of the robot
  - Cartesian grid and motion of circular arcs
  - wave propagation planner
  - real-time performance achieved by use of a precalculated table



Other Approaches

- Behavior based
  - Pros
    - easy to program
    - modular
  - Contra
    - difficult to introduce a precise task
    - reachability of goal not provable
- Fuzzy, Neuro-Fuzzy
  - Pros
    - handling of uncertainty
    - automatic programming
  - Contra
    - learning required
    - difficult to generalize