

# Robotica Probabilistica

## **Navigazione**

# Navigazione

- Evitare ostacoli (fissi, mobili)
- Pianificare il percorso o la traiettoria
- Esplorare

# Path Planning

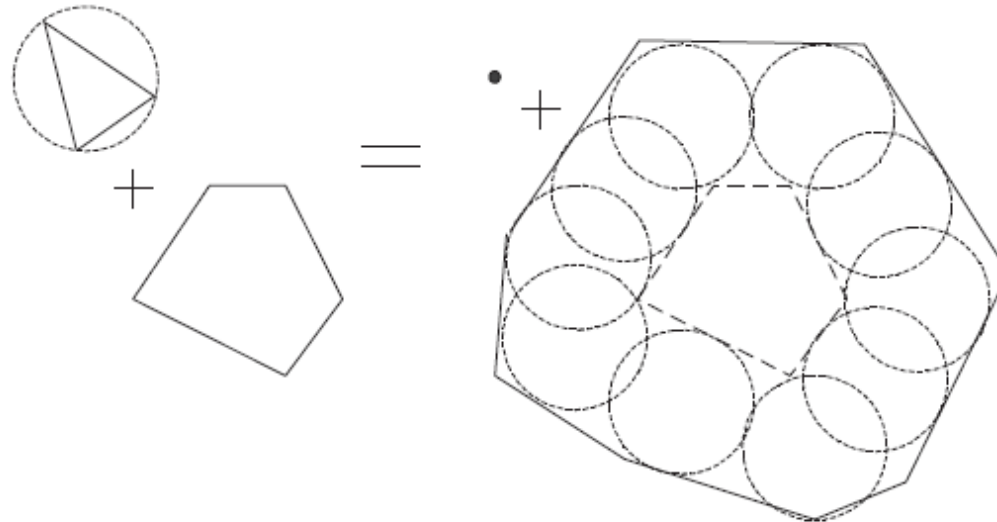
Latombe (1991):

“...eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world.”

- Traiettorie prive di collisioni
- Il robot dovrebbe raggiungere la locazione prima possibile

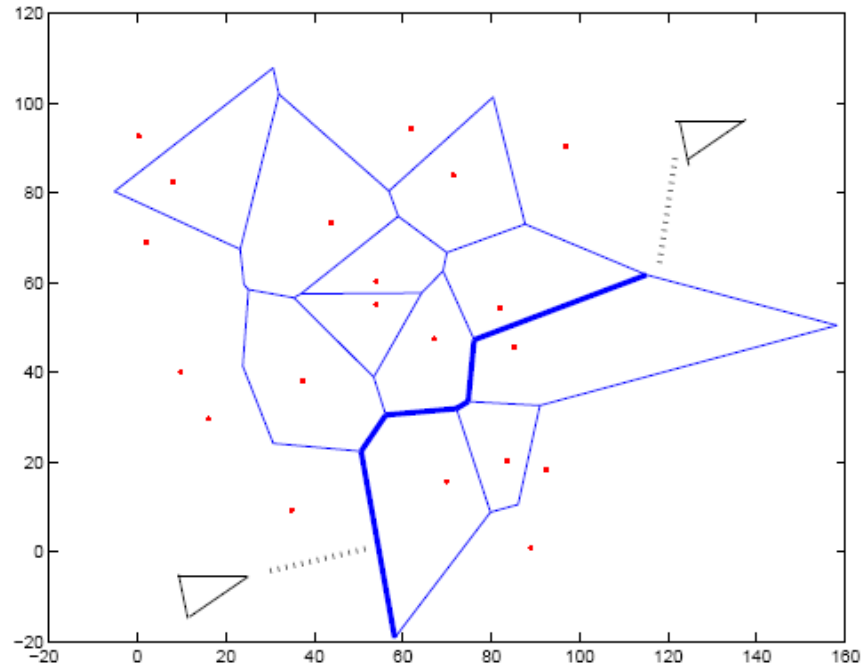
# Ostacoli: Minkowski Sum

- Come rappresentare robot e ostacoli
- Forma ostacolo + forma del robot
- Robot approssimato da punto materiale
- Ostacolo aumentato
  - Somma  $A + B = \{a+b \mid a \text{ in } A, b \text{ in } B\}$  con  $A, B$  spazi vettoriali



# Percorsi tra Ostacoli: Metodi di Voronoi

- Percorsi equidistanti tra i due punti-oggetto più vicini (diagrammi di Voronoi);
- Roadmap a partire dalla mappa:
  - Raggiungere il path vicino,
  - Path verso il goal,
  - Raggiungere il goal.



# Percorsi tra Ostacoli: Metodi Bug

- Metodi Voroni richiedono mappa intera, percorsi equidistanti e lunghi.
- Metodo Bug1:
  - Vai da S a T in linea retta;
  - Se c'è ostacolo circumnavigalo finché non trovi punto di uscita.
- Non sempre un buon piano, ma garantito B se raggiungibile.

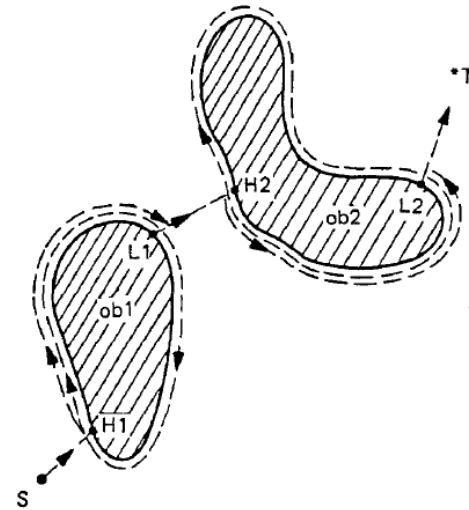


Fig. tratta da [Lumelsky Stepanov, 1987]

# Percorsi tra Ostacoli: Metodi Bug

- Metodi voroni richiedono mappa intera, percorsi equidistanti e lunghi.
- Metodo Bug2:
  - Vai da A a B in linea retta;
  - Se c'è ostacolo circumnavigalo finché non incontri/vedi la linea AB.
  - Continua a seguire AB
- Migliore piano, nei casi semplici



# Percorsi tra Ostacoli: Metodi Potenziali

- Goal basso potenziale, ostacoli alto potenziale;
- Ad ogni punto potenziale:

$$U_{\Sigma}(\mathbf{x}) = \sum_{i=1:k} U_{o,i}(\mathbf{x}) + U_g(\mathbf{x})$$

- Ad ogni punto forza:

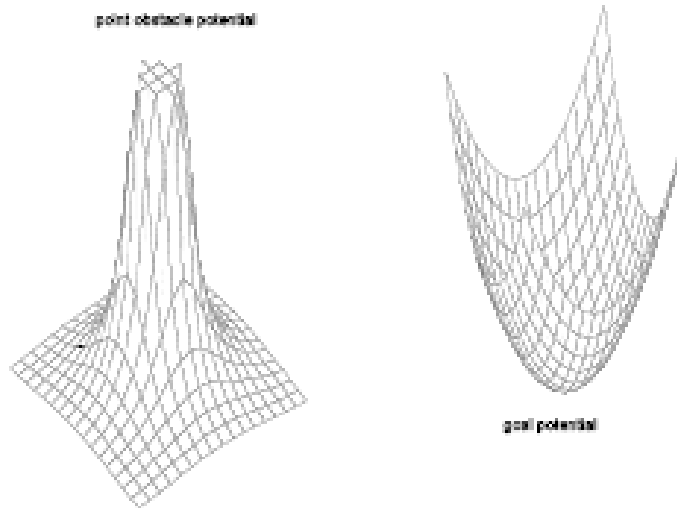
$$\begin{aligned} \mathbf{F}(\mathbf{x}) &= -\nabla U_{\Sigma}(\mathbf{x}) \\ &= -\sum_{i=1:k} \nabla U_{o,i}(\mathbf{x}) - \nabla U_g(\mathbf{x}) \end{aligned}$$

- Quale potenziale?



# Percorsi tra Ostacoli: Metodi Potenziali

- Funzioni tipiche:



$$U_{o,i}(\mathbf{x}) = \eta \begin{cases} \frac{1}{2} \left( \frac{1}{\rho(\mathbf{x})} - \frac{1}{\rho_0} \right)^2 & \forall \rho(\mathbf{x}) \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}$$
$$U_g(\mathbf{x}) = \frac{1}{2} (\mathbf{x} - \mathbf{x}_g)^2$$

- Tra voroni e bug per vicinanza ad ostacoli
- Metodo locale, quindi problema minimi locali

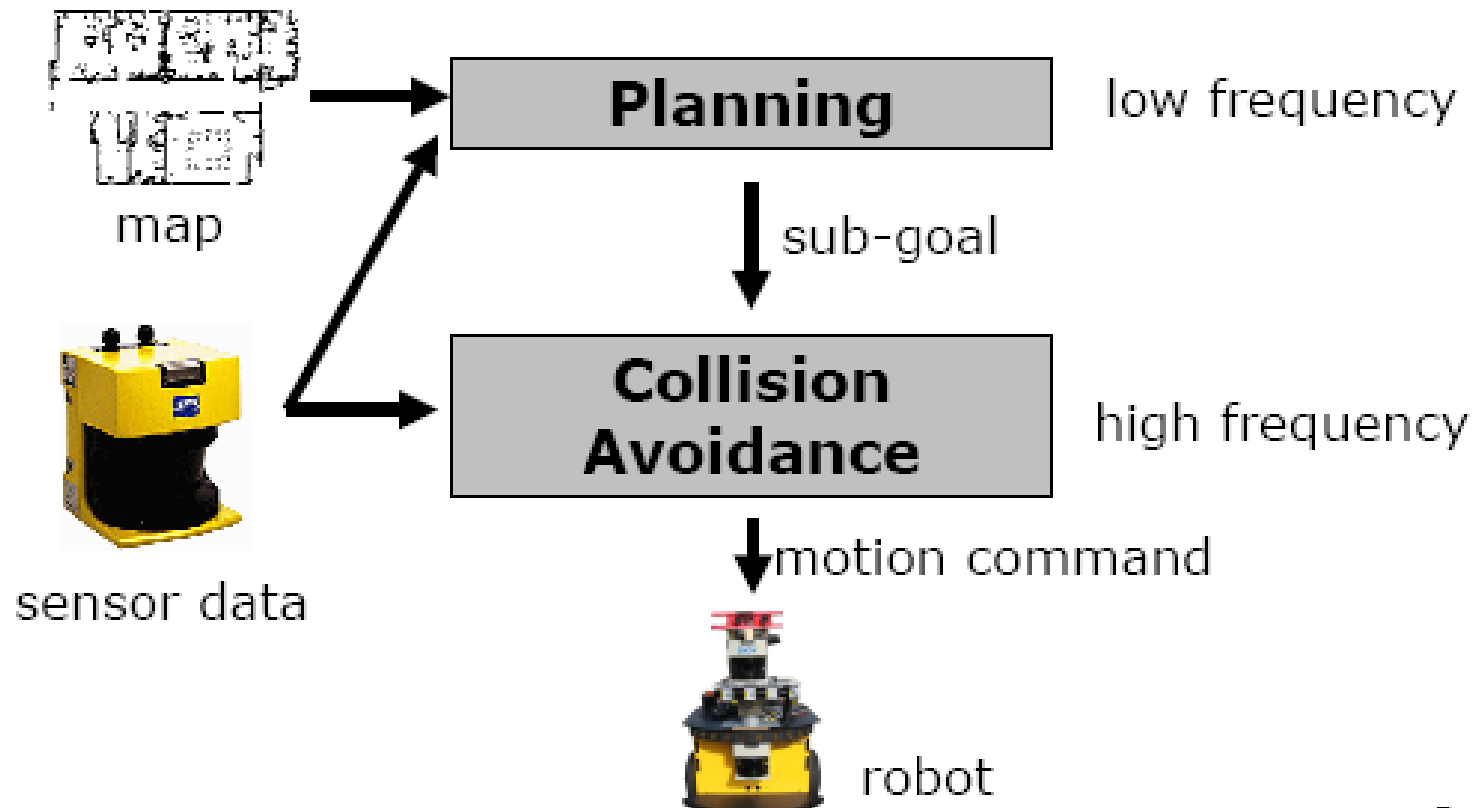
# Path planning e Ambiente Dinamico

- Come reagire ad ostacoli imprevisti?
  - Efficacia
  - Affidabilità
- Approcci:
  - Dynamic Window Approaches  
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]
  - Grid map based Planning  
[Konolige, 00]
  - Nearness Diagram Navigation  
[Minguez et al., 2001, 2002]
  - Vector-Field-Histogram+  
[Ulrich & Borenstein, 98]
  - A\*, D\*, D\* Lite, ARA\*, ...

# Obiettivi in ambiente dinamico

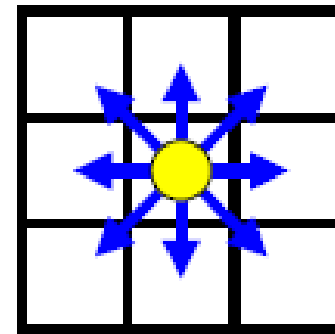
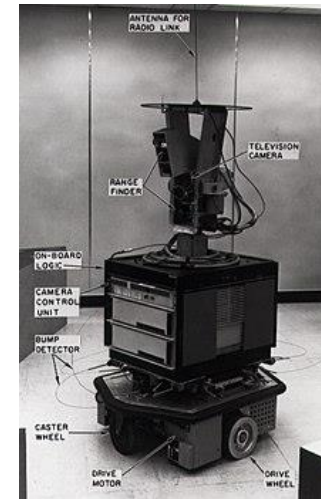
- Calcolo del path ottimo considerando incertezze potenziali nelle azioni
- Generare azioni rapidamente in caso di ostacoli imprevisti

# Approccio Classico



# Path Planning su grid map

- What about using A\* to plan the path of a robot?
- Finds the shortest path
- Requires a graph structure
- Limited number of edges
- In robotics: planning on a 2d occupancy grid map



# Alberi di Ricerca

- Metodo:
  - Esplorazione dello spazi degli stati generando il successor degli stati già esplorati.
  - Ogni stato valutato: *è uno stato goal?*

# Best-first search

- Idea: usare una **funzione di valutazione**  $f(n)$  per ogni nodo:
  - $f(n)$  fornisce una stima del costo totale
  - Espandi il nodo  $n$  con più piccolo  $f(n)$ .

# A\* search

- Idea: Evita di espandere percorsi che sono già molto costosi
- Evaluation function  $f(n) = g(n) + h(n)$
- $g(n)$  = costo fino ad ora per  $n$
- $h(n)$  = stima del costo da  $n$  al goal
- $f(n)$  = stima del costo totale del path per  $n$  al goal
- Best First search ha  $f(n)=h(n)$
- Uniform Cost search ha  $f(n)=g(n)$



# Euristica Ammissibile

- Una euristica  $h(n)$  è **ammissibile** se per ogni nodo  $n$ ,  $h(n) \leq h^*(n)$ , dove  $h^*(n)$  è il costo **reale** per raggiungere il goal da  $n$ .
- Una euristica ammissibile **non sovrastima** il costo per raggiungere il goal, i.e., è **ottimistica**
- Esempio:  $h_{SLD}(n)$  (non sovrastimata la reale distanza)
- Teorema: se  $h(n)$  è ammissibile,  $A^*$  con TREE-SEARCH è ottimale

# Dominanza

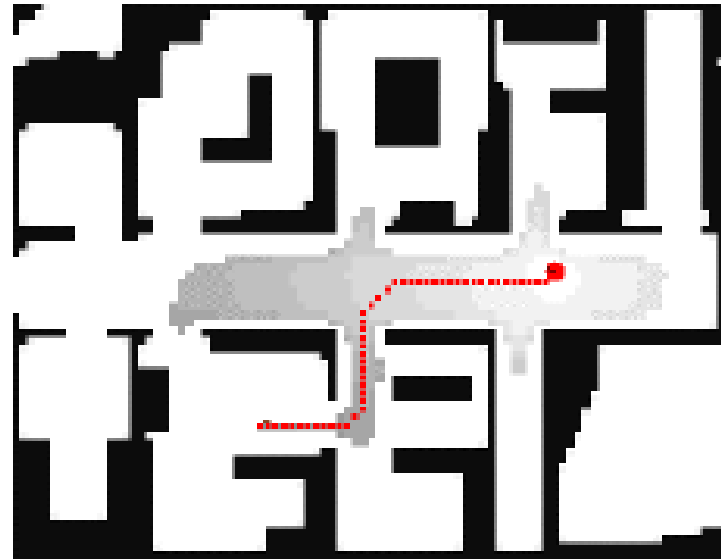
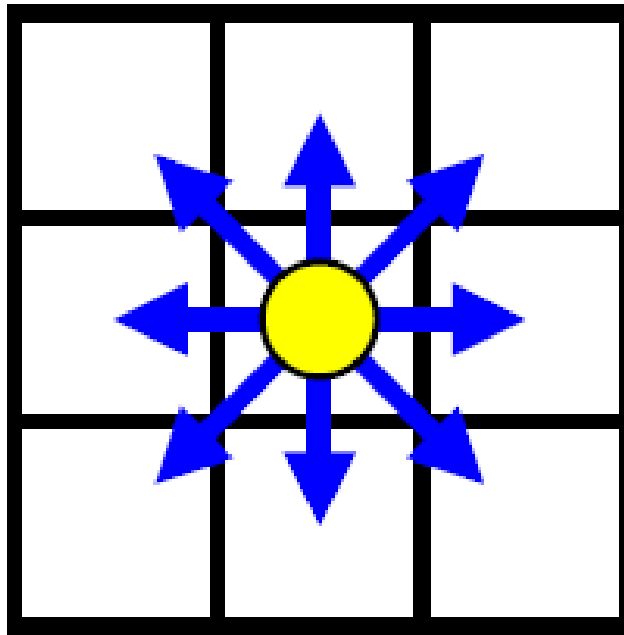
- Se  $h_2(n) \geq h_1(n)$  per ogni  $n$  (entrambe ammissibili)
- allora  $h_2$  **domina**  $h_1$
- $h_2$  è migliore per la ricerca: è garantito che espande un numero di nodi minore o uguale.

# Propert` di A\*

- Terminazione e complessa
  - Su grafi finiti ed archi non negativi
- Tempo:
  - Esponenziale:  $b^d$  dove b ` il branching e d la profondit`
  - Polinomiale se:  $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- Ottimale
- Ottimamente efficiente: (non c'` altro algoritmo che con la stessa euristica pu` espandere meno nodi)

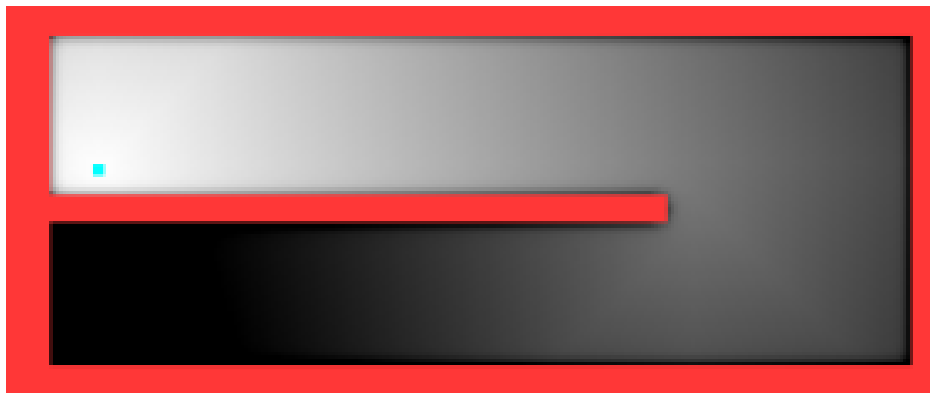
# Path Planning nella grid map

L'algoritmo di A\* può essere utilizzato nella grid map (occupancy grid)



# Value Iteration

- To compute the shortest path from every state to one goal state, use (deterministic) value iteration.
- Very similar to Dijkstra's Algorithm.
- Such a cost distribution is the optimal heuristic for  $A^*$ .



# Assunzioni tipiche in A\* path planning

- A robot is assumed to be localized.
- Often a robot has to compute a path based on an occupancy grid.
- Often the correct motion commands are executed (but no perfect map).

**Is this always true?**

# Problemi

- What if the robot is slightly delocalized?
- Moving on the shortest path guides often the robot on a trajectory close to obstacles.
- Trajectory aligned to the grid structure.

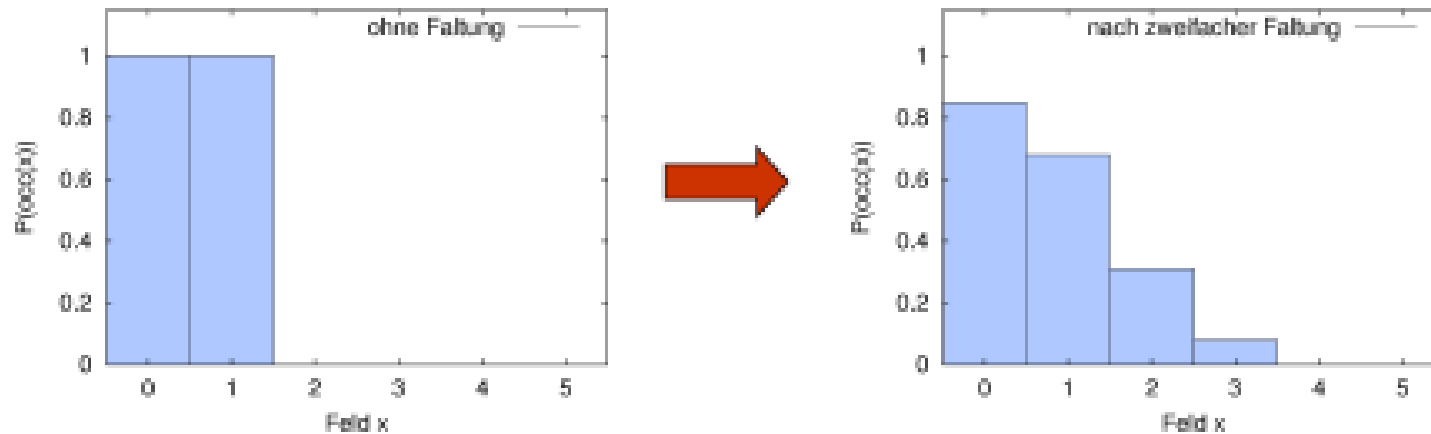
# Possibile Soluzione: convoluzione della grid map

- Convolution blurs the map.
- Obstacles are assumed to be bigger than in reality.
- Perform an A\* search in such a convolved map.
- Robots increases distance to obstacles and moves on a short path!



# Esempio

- 1-d environment, cells  $c_0, \dots, c_5$



- Cells before and after 2 convolution runs.

# Convoluzione

- Consider an occupancy map. Then the convolution is defined as:

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

$$P(occ_{x_0,y}) = \frac{2}{3} \cdot P(occ_{x_0,y}) + \frac{1}{3} \cdot P(occ_{x_1,y})$$

$$P(occ_{x_{n-1},y}) = \frac{1}{3} \cdot P(occ_{x_{n-2},y}) + \frac{2}{3} \cdot P(occ_{x_{n-1},y})$$

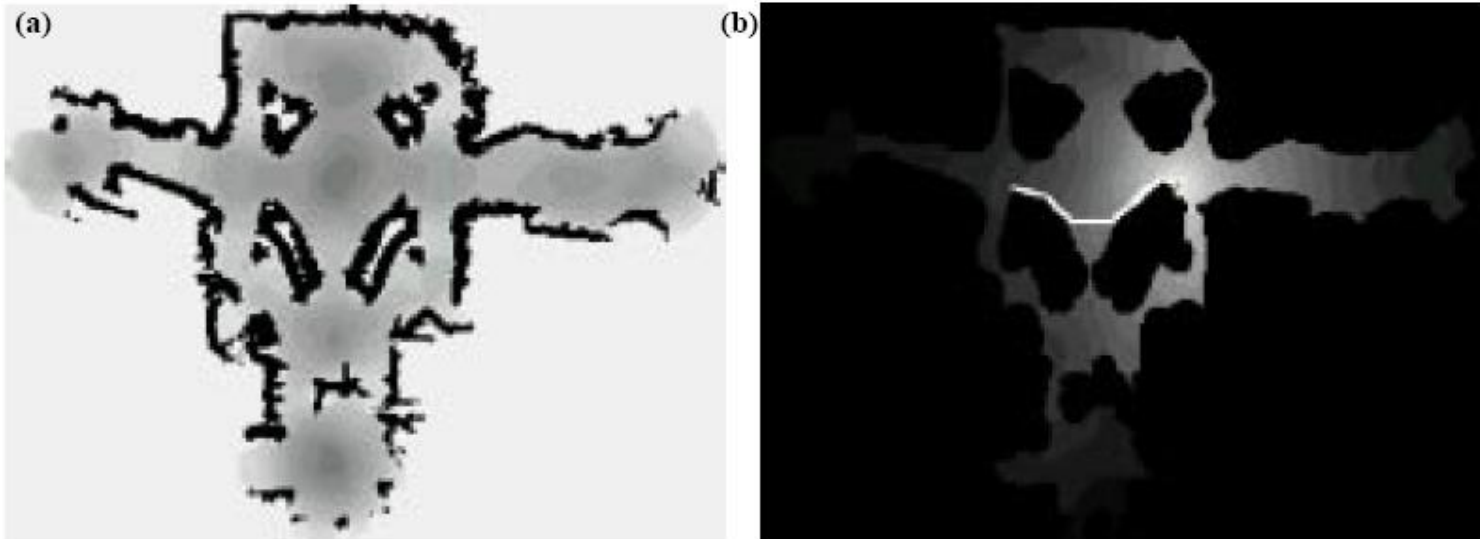
- This is done for each row and each column of the map.
- "Gaussian blur"

# Applicazione di A\*

- The costs are a product of path length and occupancy probability of the cells.
- Cells with higher probability (e.g., caused by convolution) are avoided by the robot.
- Thus, it keeps distance to obstacles.
- This technique is **fast** and quite **reliable**.

# Navigazione Costale

- Non il path più breve, ma quello che minimizza la probabilità di perdersi
- Considerare: lunghezza path e informazione persa (es. Minerva Robot)
- Entropy Map:



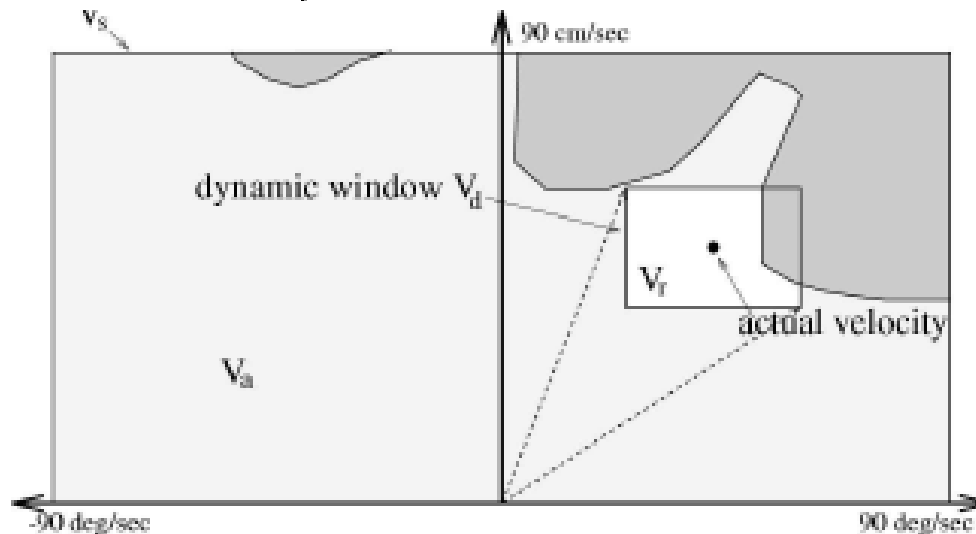
# Dynamic Windows

- **Collision avoidance:** determine collision-free trajectories using geometric operations
- Here: robot moves on circular arcs
- Motion commands  $(v, \omega)$
- Which  $(v, \omega)$  are admissible?

# Dynamic Windows

Regolazione della velocità in funzione degli ostacoli

- Example search-space:



- $V_s$  = all possible speeds of the robot.
- $V_a$  = obstacle free area.
- $V_d$  = speeds reachable within a certain time frame based on possible accelerations.

$$Space = V_s \cap V_a \cap V_d$$

# Dynamic Windows

- How to choose  $\langle v, \omega \rangle$ ?
- Steering commands are chosen by a heuristic navigation function.
- This function tries to minimize the travel-time by:  
“**driving fast** in the **right direction.**”

# Dynamic Windows

- Heuristic navigation function.
- Planning restricted to  $\langle x, y \rangle$ -space.
- No planning in the velocity space.

Navigation Function:

Goal nearness.

$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity.

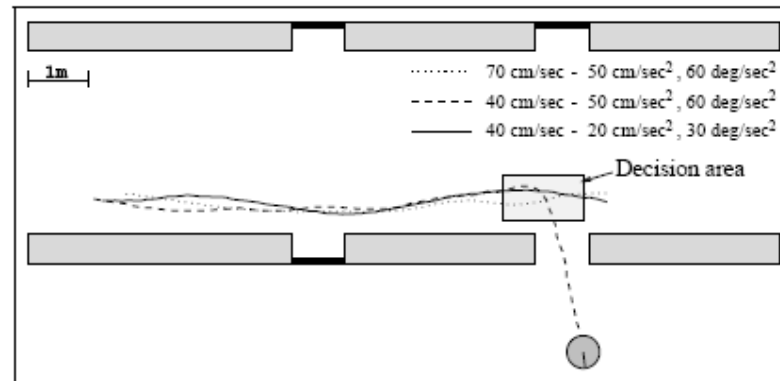
Considers cost to reach the goal.

Follows grid based path computed by A\*.

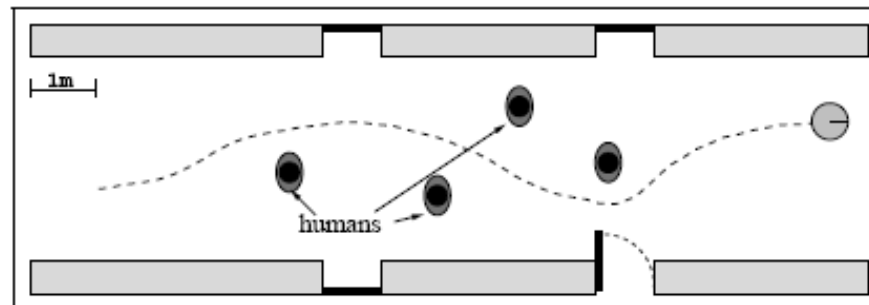


# Dynamic Windows Approach

- Brevi tempi di reazione (dipendenti da parametri)



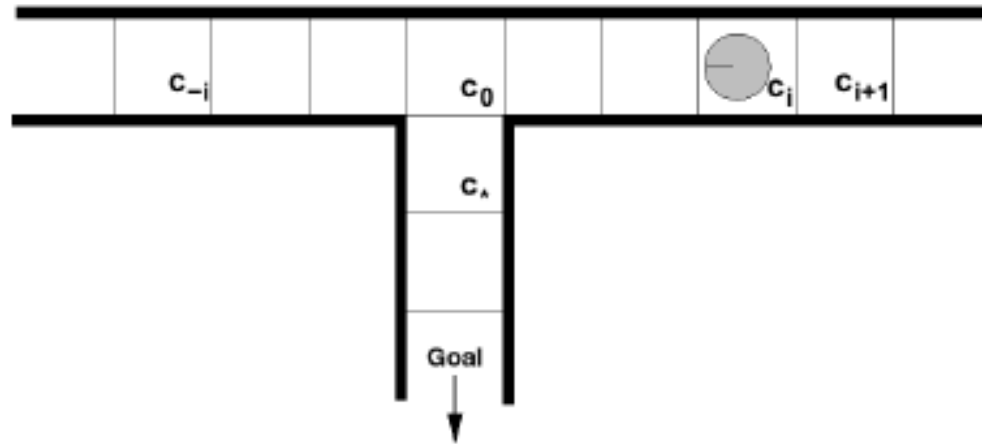
- Veloce in ambienti popolati



# Dynamic Windows

- Reacts quickly.
- Low CPU power requirements.
- Guides a robot on a collision free path.
- Successfully used in a lot of real-world scenarios.
- Resulting trajectories sometimes sub-optimal.
- Local minima might prevent the robot from reaching the goal location.

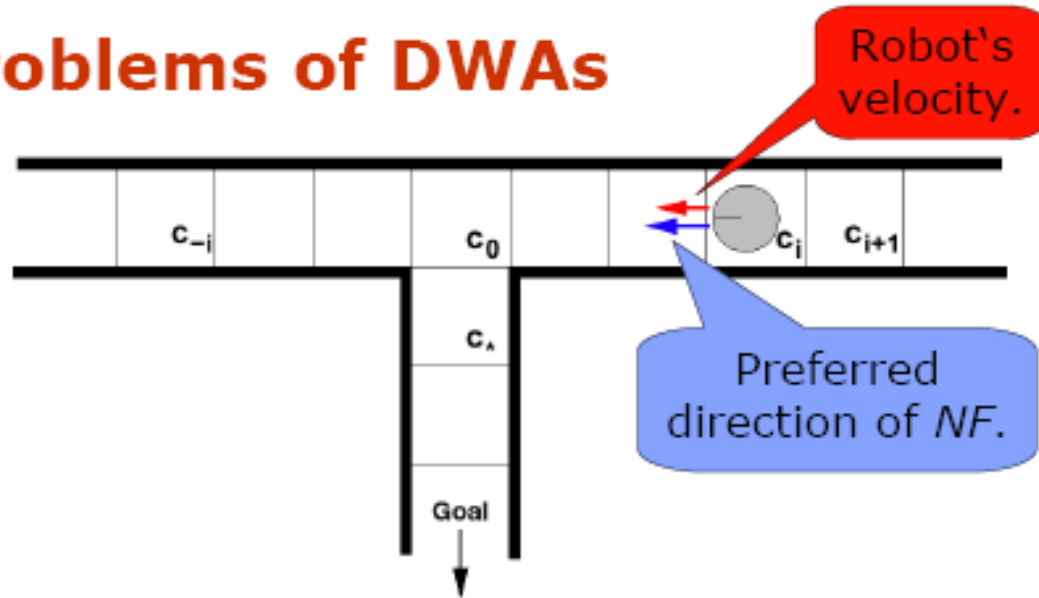
# Problema con DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

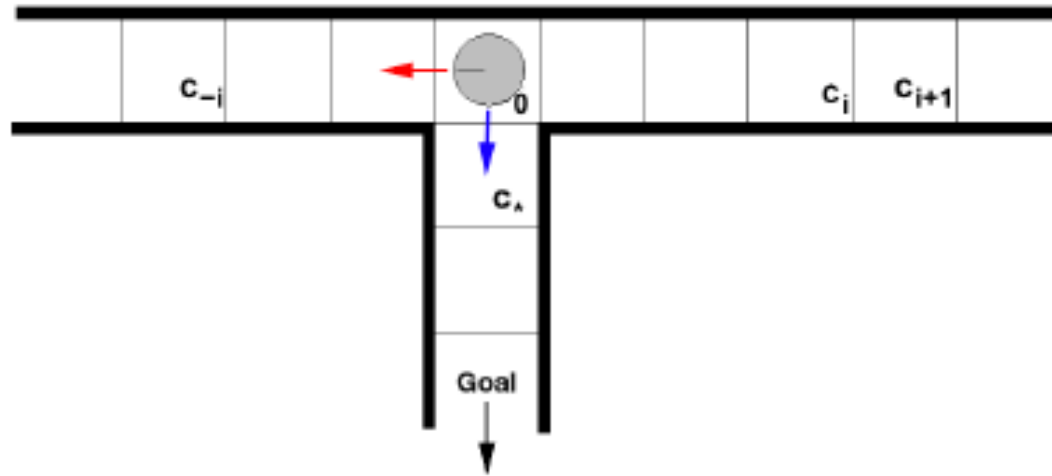
# Problema con DWA

## Problems of DWAs



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

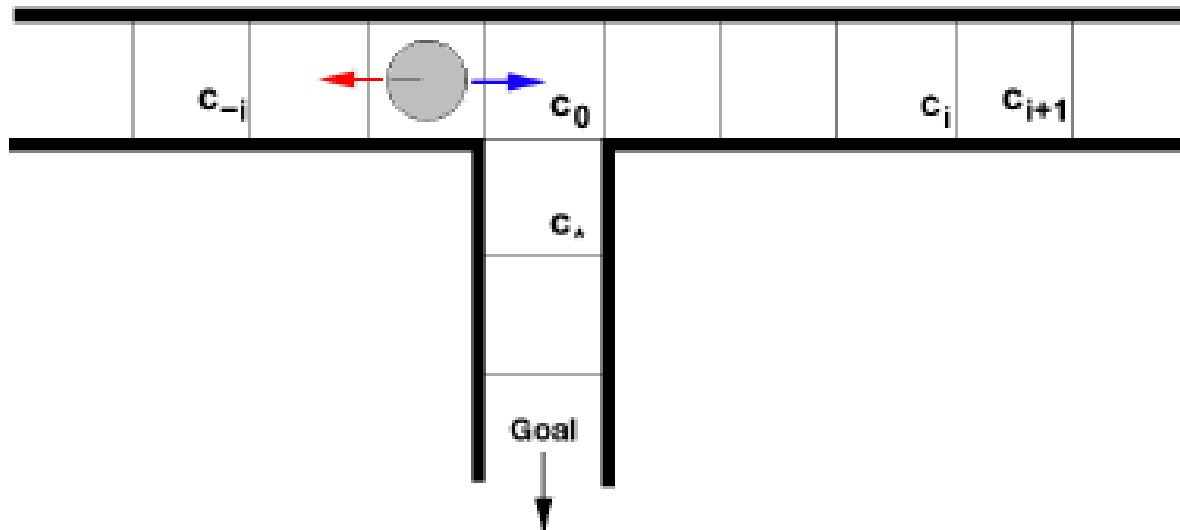
# Problema con DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

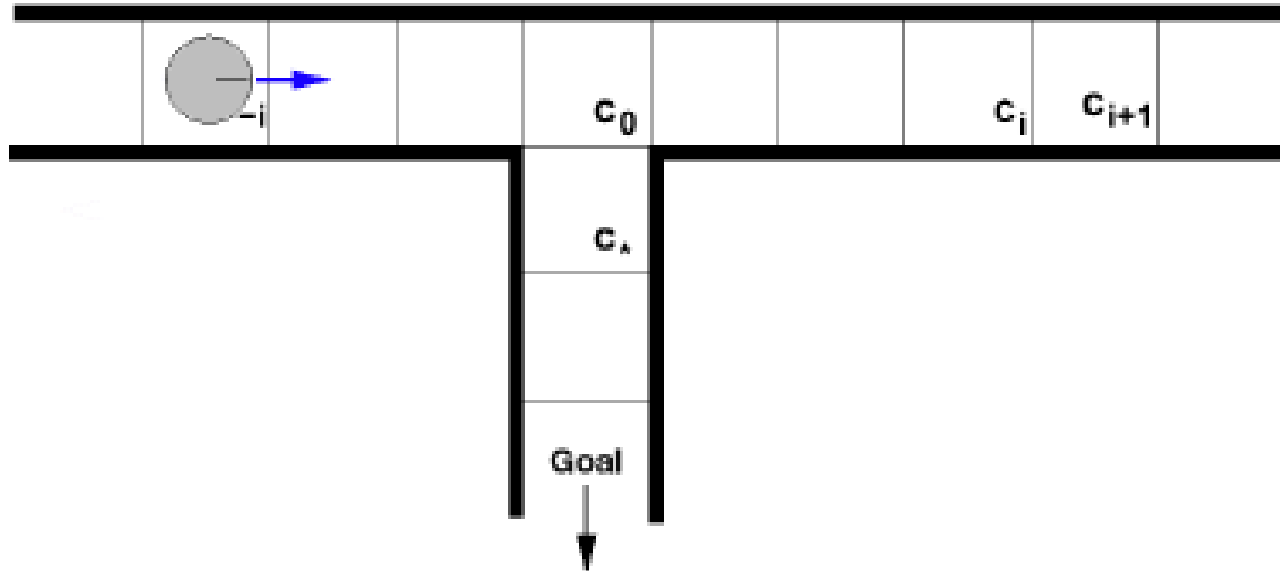
- The robot drives too fast at  $c_0$  to enter corridor facing south.

# Problema con DWA



$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

# Problema con DWA



- Same situation as in the beginning.
  - ➔ DWAs have problems to reach the goal.

# Problema con DWA

- Typical problem in a real world situation:



- Robot does not slow down early enough to enter the doorway.



# 5D Path-planning

- Alternativa all'approccio a due livelli
- Plans in the full  $\langle x, y, \theta, v, \omega \rangle$ -configuration space using  $A^*$ .
  - considers the robot's kinematic constraints.
- Generates a sequence of steering commands to reach the goal location.
- Maximizes trade-off between driving time and distance to obstacles.

# Spazio di Ricerca

- What is a state in this space?  
 $\langle x, y, \theta, v, \omega \rangle =$  current position and speed of the robot
- How does a state transition look like?  
 $\langle x_1, y_1, \theta_1, v_1, \omega_1 \rangle \longrightarrow \langle x_2, y_2, \theta_2, v_2, \omega_2 \rangle$   
with motion command  $(v_2, \omega_2)$  and  
 $|v_1 - v_2| < a_v, |\omega_1 - \omega_2| < a_\omega$ . Pose of the Robot is a result of the motion equations.

# Spazio di Ricerca

**Idea:** search in the discretized  $\langle x, y, \theta, v, \omega \rangle$ -space.

**Problem:** the search space is too huge to be explored within the time constraints (.25 secs for online control).

**Solution:** restrict the full search space.

# Passi dell'algoritmo

1. Update (static) grid map based on sensory input.
2. Use  $A^*$  to find a trajectory in the  $\langle x, y \rangle$ -space using the updated grid map.
3. Determine a restricted 5d-configuration space based on step 2.
4. Find a trajectory by planning in the restricted  $\langle x, y, \theta, v, \omega \rangle$ -space.

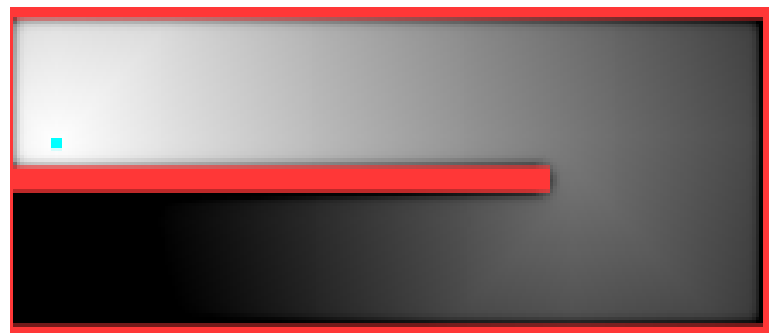
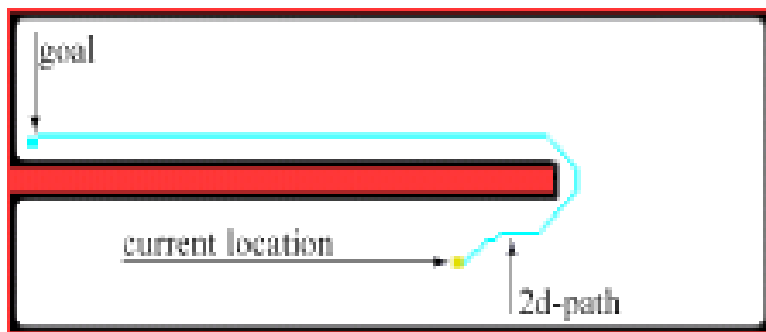
# Aggiornamento della grid-map

- The environment is represented as a 2d-occupancy grid map.
- Convolution of the map increases security distance.
- Detected obstacles are added.
- Cells discovered free are cleared.



# Percorso nella mappa 2D

- Use  $A^*$  to search for the optimal path in the 2d-grid map.
- Use heuristic based on a deterministic value iteration within the static map.



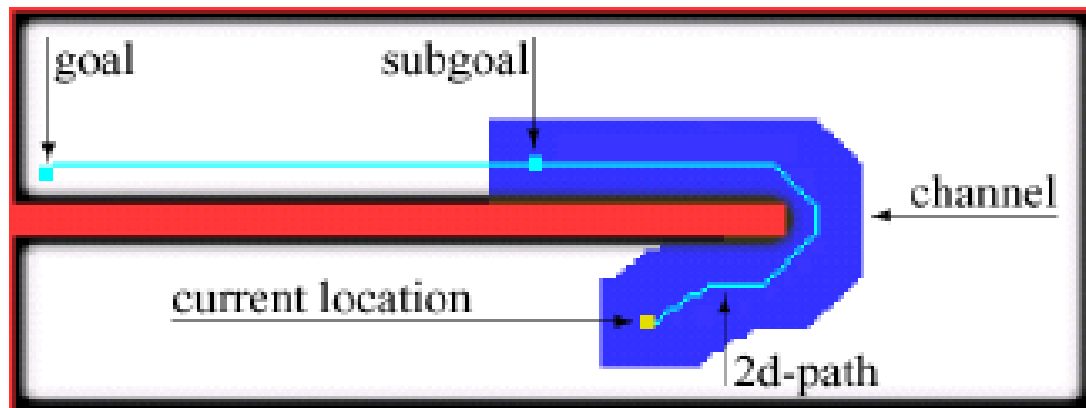
# Restringi lo spazio di ricerca

**Assumption:** the projection of the 5d-path onto the  $\langle x, y \rangle$ -space lies close to the optimal 2d-path.

**Therefore:** construct a restricted search space (channel) based on the 2d-path.

# Spazio di Ricerca

- Resulting search space =  $\langle x, y, \theta, v, \omega \rangle$  with  $(x, y) \in \text{channel}$ .
- Choose a sub-goal lying on the 2d-path within the channel.

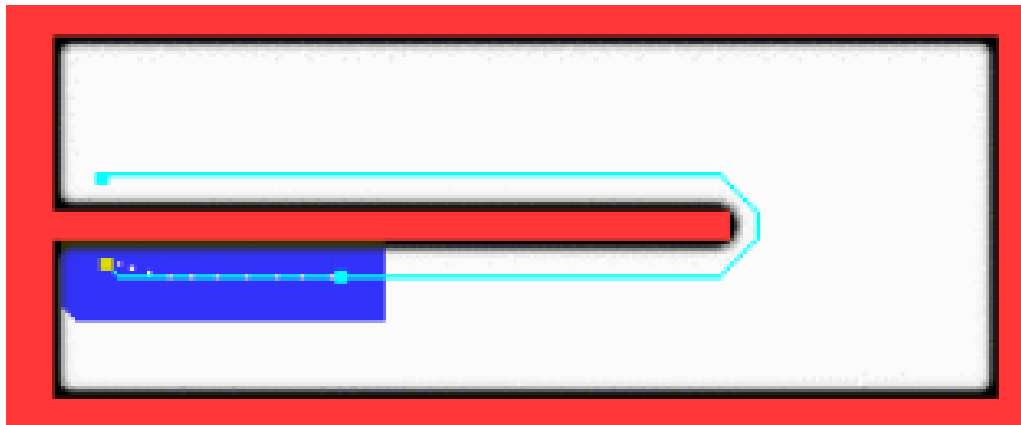
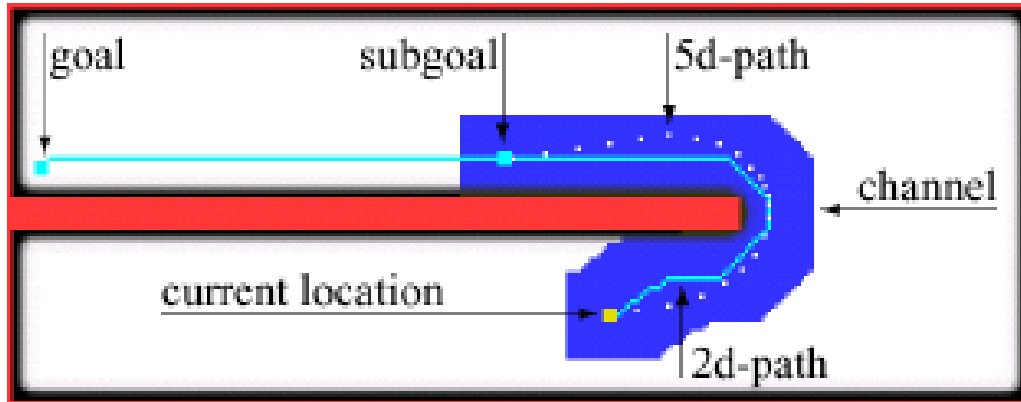




# Trova il percorso in 5D

- Use  $A^*$  in the restricted 5d-space to find a sequence of steering commands to reach the sub-goal.
- To estimate cell costs: perform a deterministic 2d-value iteration within the channel.

# Esempio



# Timeout

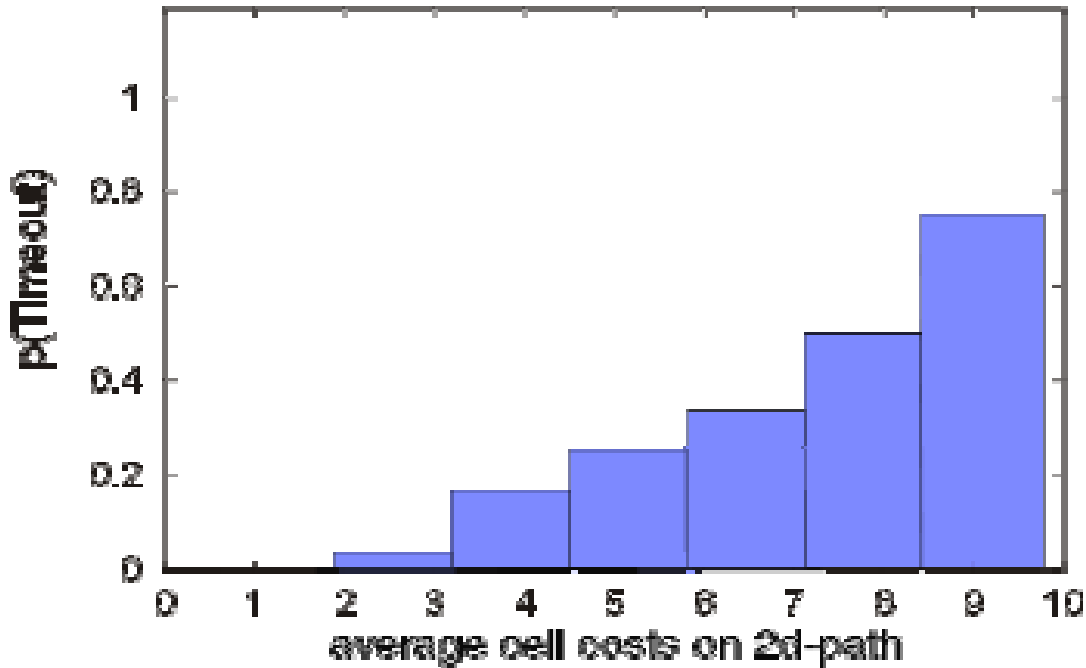
- Online robot control:  
new steering command every .25 secs.
- Abort search after .25 secs.

How to find an admissible steering command?

# Generazione di Comandi

- Previous trajectory still admissible? → OK
- If not, drive on the 2d-path or use DWA to find new command.

# Timeout avoidance

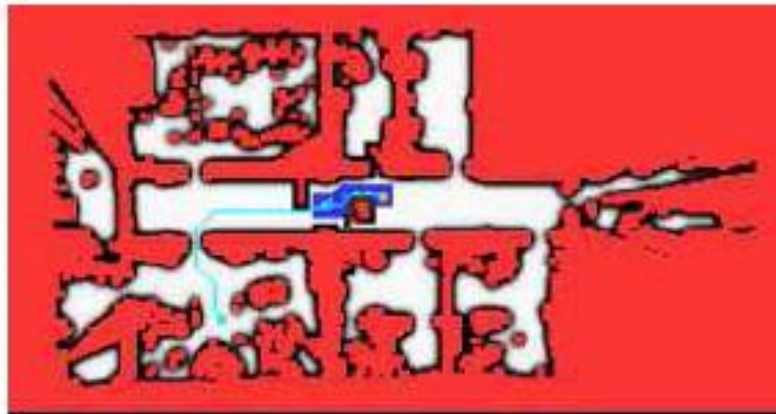


➔ Reduce the size of the channel if the 2d-path has high cost.

# Esempio



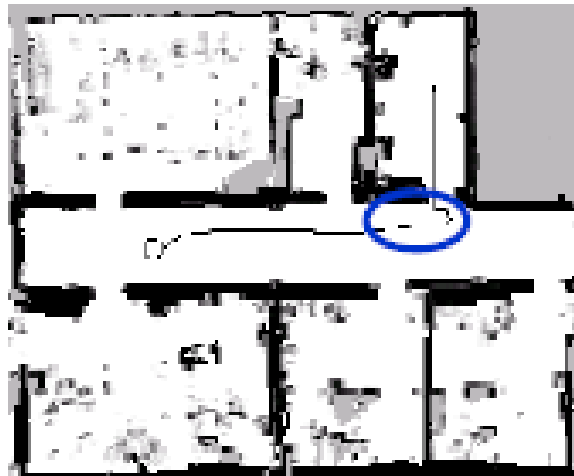
Robot Albert



Planning state

# Confronto con DWA

- DWAs often have problems entering narrow passages.

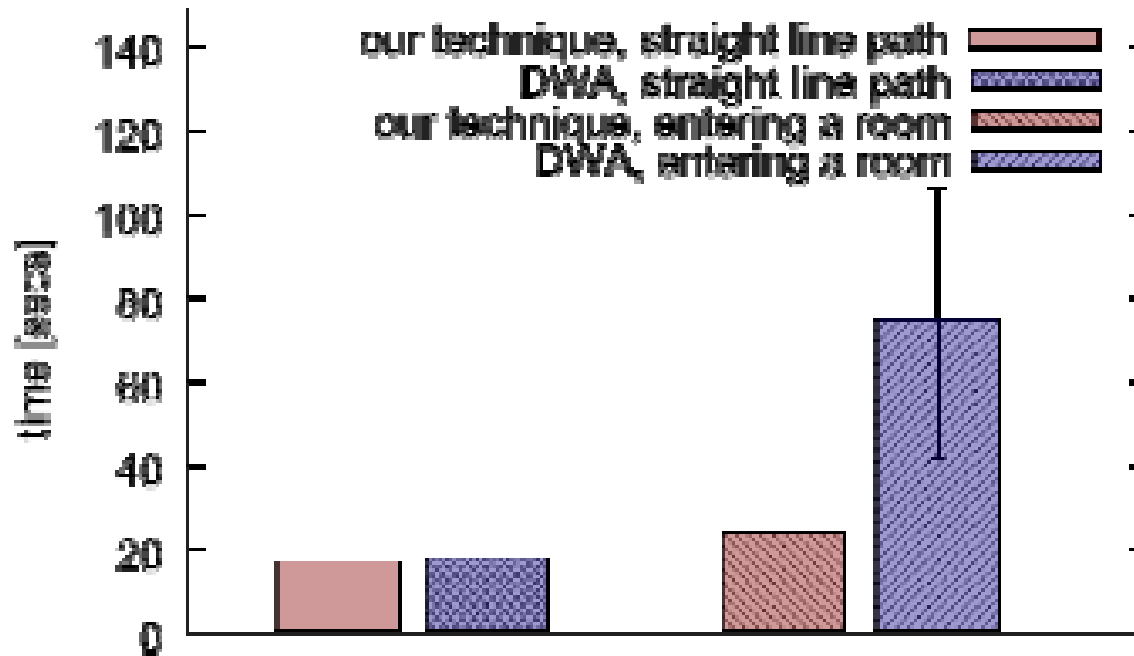


DWA planned path.



5D approach.

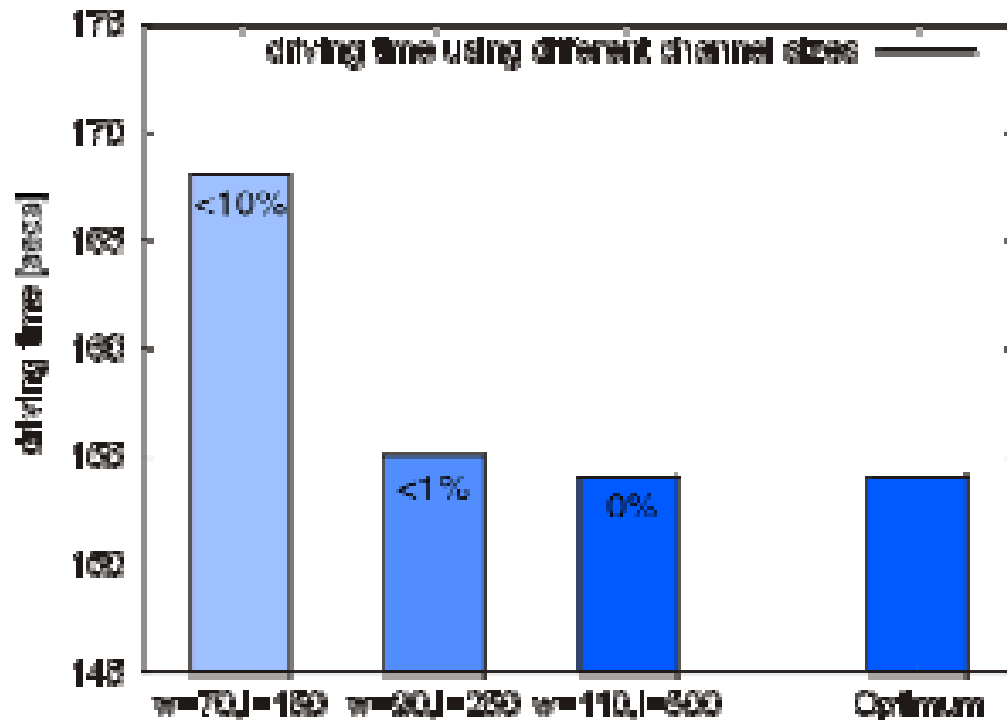
# Confronto con DWA



→ The presented approach results in significantly faster motion when driving through narrow passages!



# Confronto con l'ottimo



Channel: with length=5m, width=1.1m

Resulting actions are close to the optimal solution.

# Riassunto

- Robust navigation requires combined path planning & collision avoidance
- Approaches need to consider robot's kinematic constraints and plans in the velocity space.
- Combination of search and reactive techniques show better results than the pure DWA in a variety of situations.
- Using the 5D-approach the quality of the trajectory scales with the performance of the underlying hardware.
- The resulting paths are often close to the optimal ones.