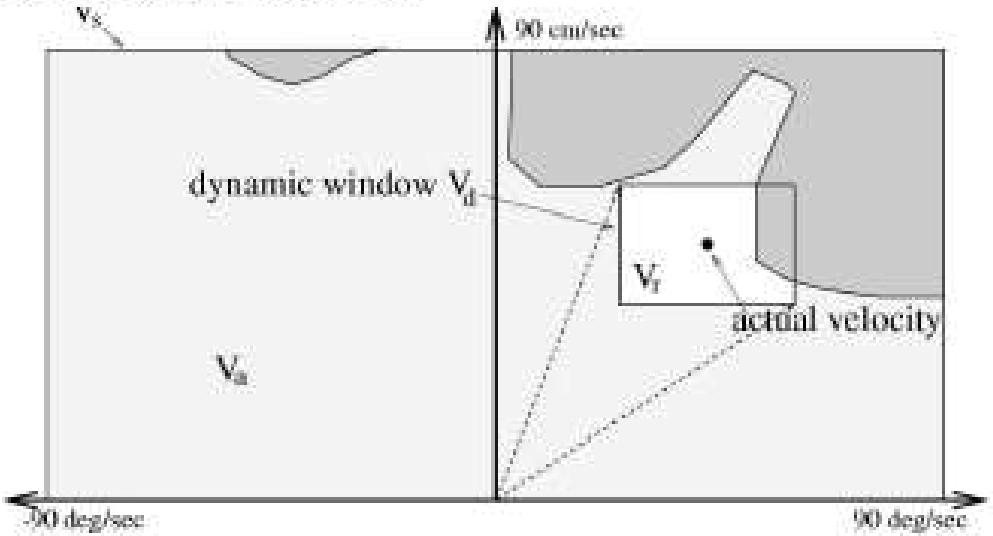
Dynamic Windows

- **Collision avoidance:** determine collision-free trajectories using geometric operations
- Here: robot moves on circular arcs
- Motion commands (v,ω)
- Which (v,ω) 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 <v,ω>?
- 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 <x,y>-space.
- No planning in the velocity space.

Navigation Function:

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

Goal nearness.

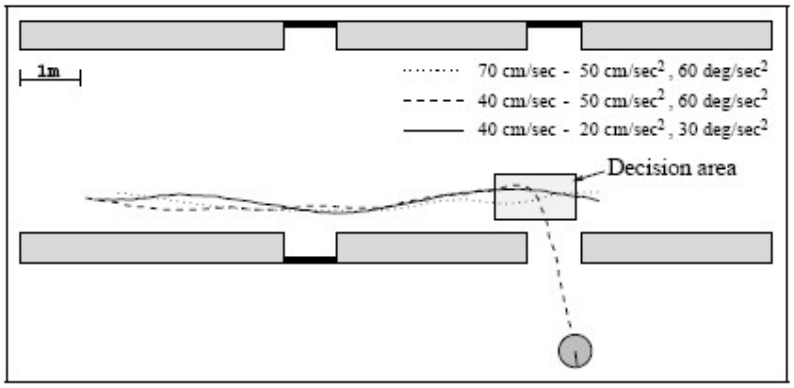
Maximizes velocity.

Considers cost to reach the goal.

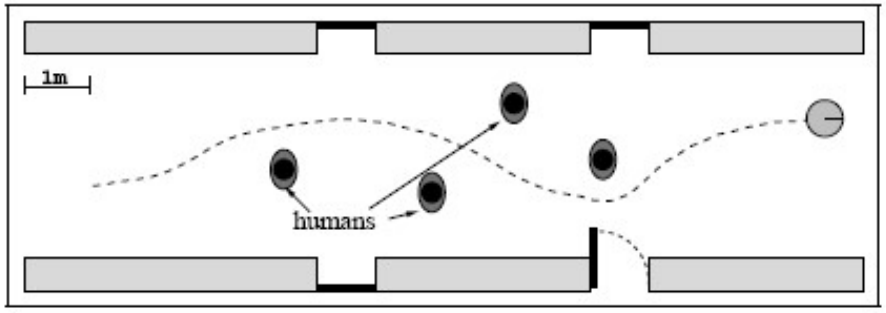
Follows grid based path computed by A*.

Dynamic Windows Approach

- Brevi tempi di reazione (dipendenti da parametri)



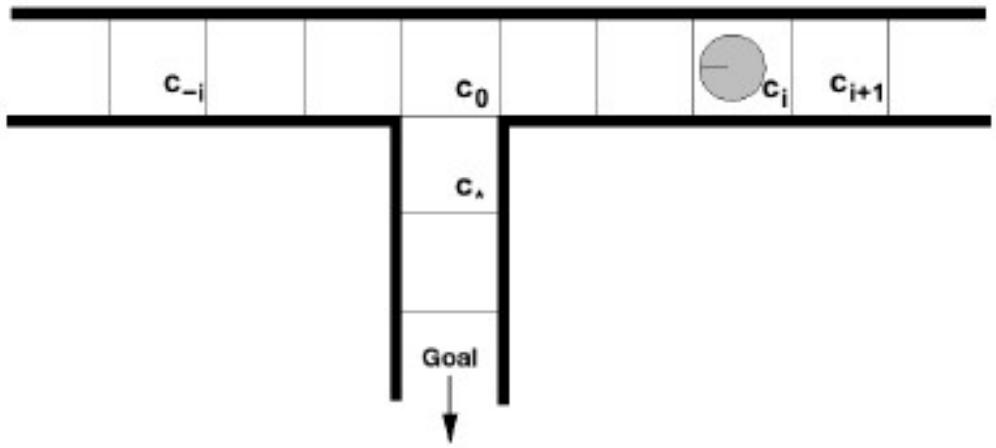
- Veloce in ambienti popolati



Dynamic Windows

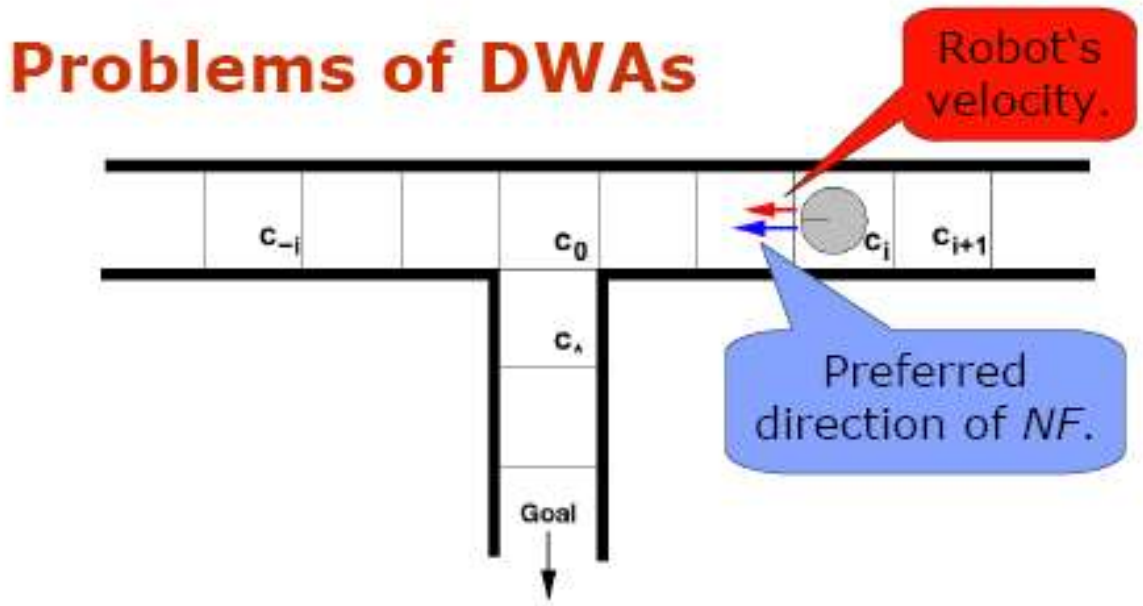
- Reacts quickly.
- Low CPU power requirements.
- Guides a robot on a collision free path.
- Successfully used in a lot of real-world scenarios.
- Resulting trajectories sometimes sub-optimal.
- Local minima might prevent the robot from reaching the goal location.

Problema con DWA



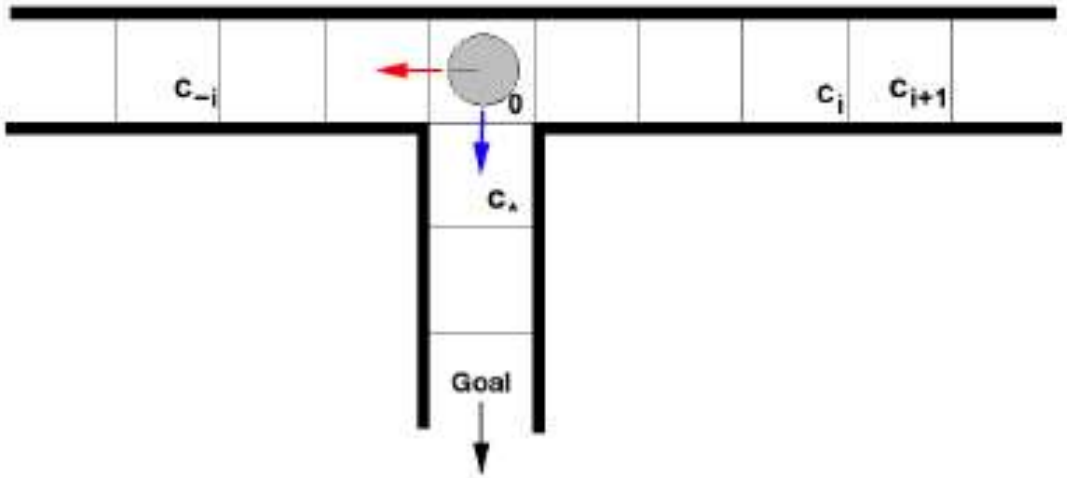
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Problema con DWA



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

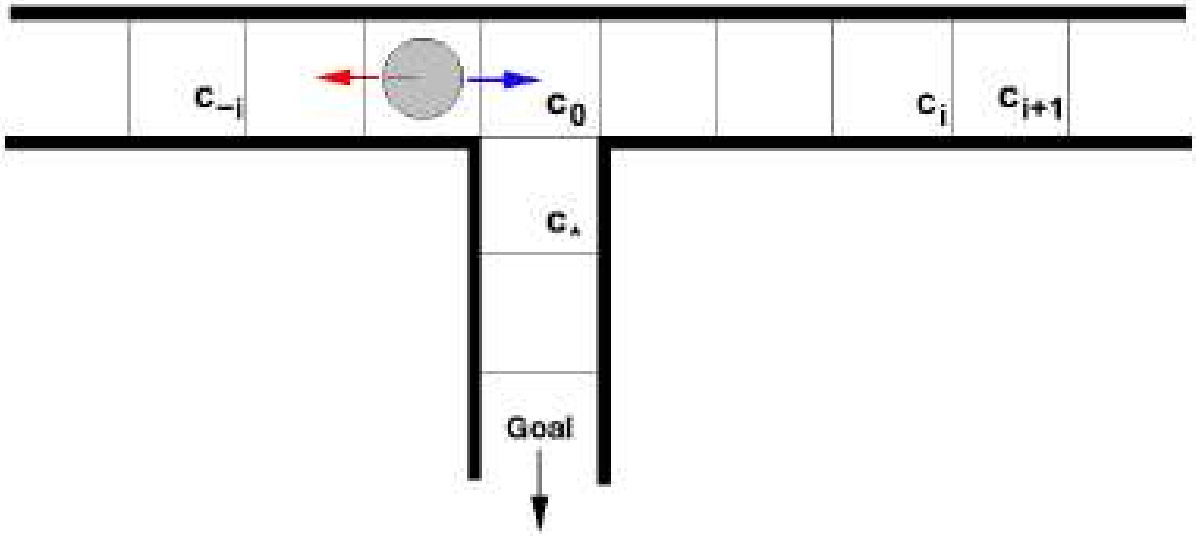
Problema con DWA



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

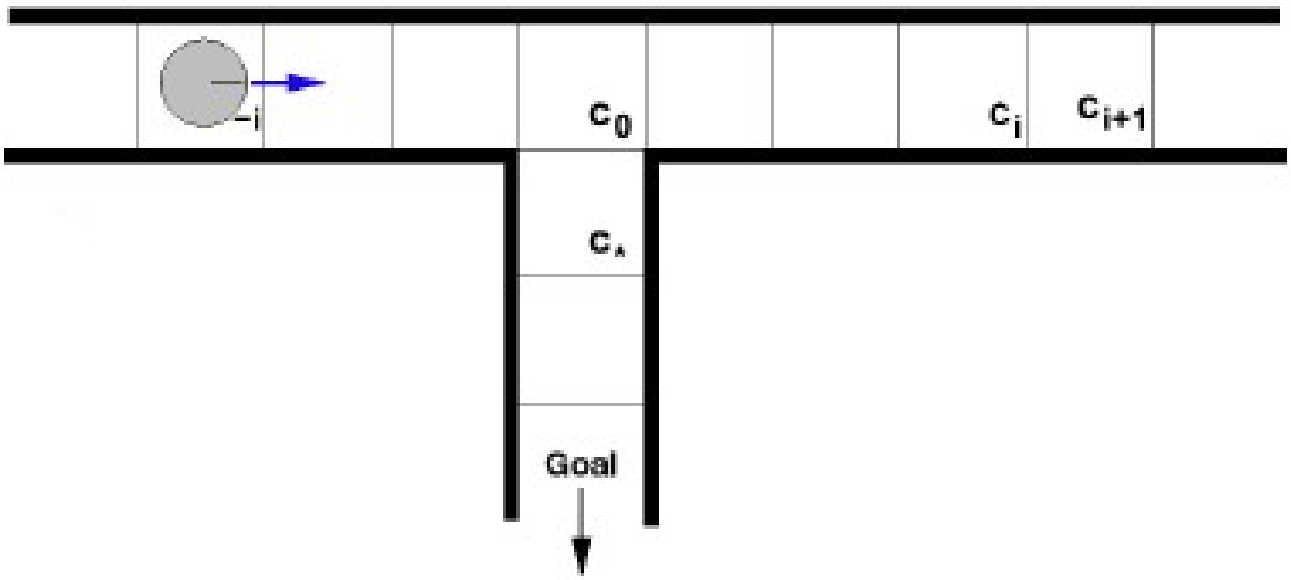
- The robot drives too fast at c_0 to enter corridor facing south.

Problema con DWA



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

Problema con DWA

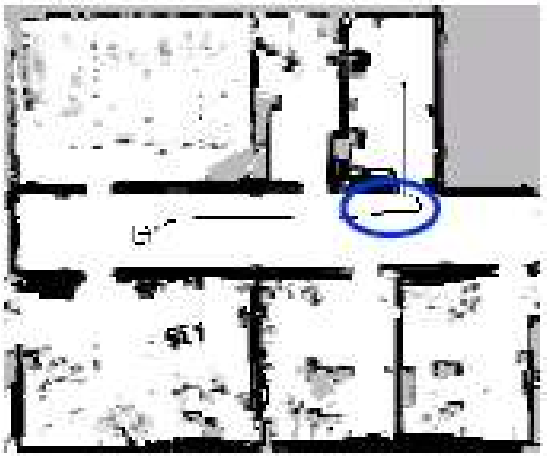


- Same situation as in the beginning.

  ➔ 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 <x,y,θ,v,ω>-configuration space using A^*.
  ➔ considers the robot's kinematic contraints.
- 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?
  $\langle x_1,y_1,\theta_1,v_1,\omega_1 \rangle \rightarrow \langle x_2,y_2,\theta_2,v_2,\omega_2 \rangle$
  with motion command (v_2,ω_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

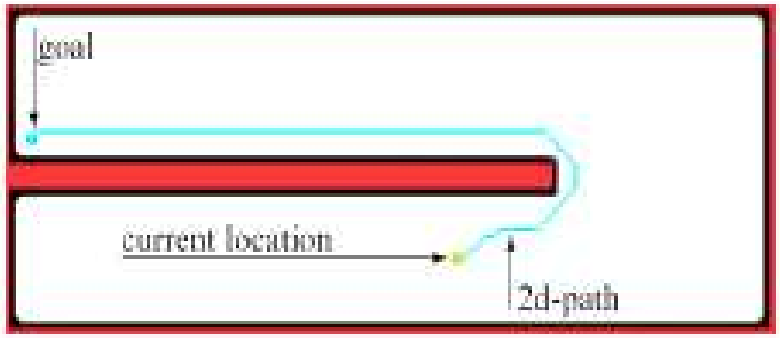
1. Update (static) grid map based on sensory input.
2. Use A^* to find a trajectory in the $\langle x,y \rangle$-space using the updated grid map.
3. Determine a restricted 5d-configuration space based on step 2.
4. Find a trajectory by planning in the restricted $\langle x,y,\theta,v,\omega \rangle$-space.

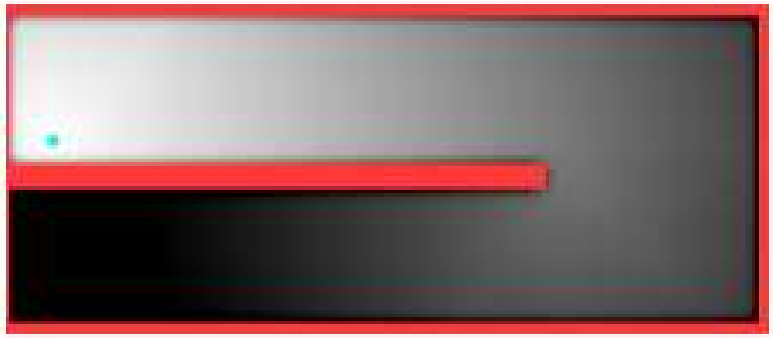
Aggiornamento della grid-map

- The environment is represented as a 2d-occupency 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.





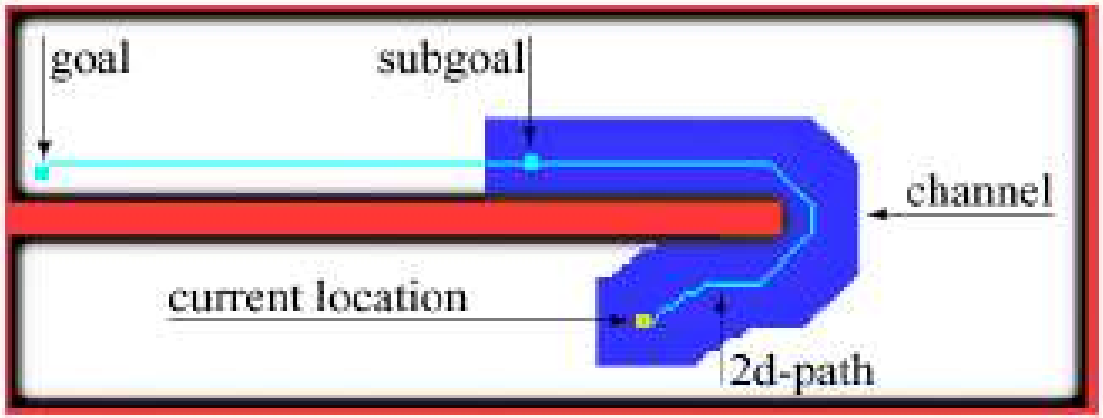
Restringi lo spazio di ricerca

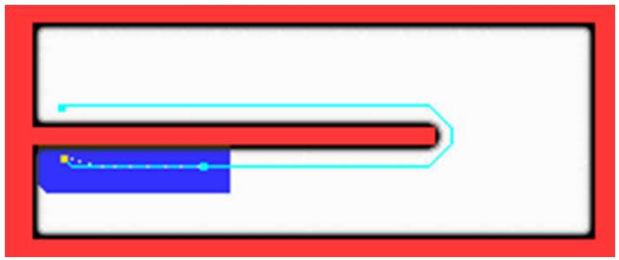
Assumption: the projection of the 5d-path onto the <x,y>-space lies close to the optimal 2d-path.

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 $(x,y) \in$ channel.
- Choose a sub-goal lying on the 2d-path within the channel.



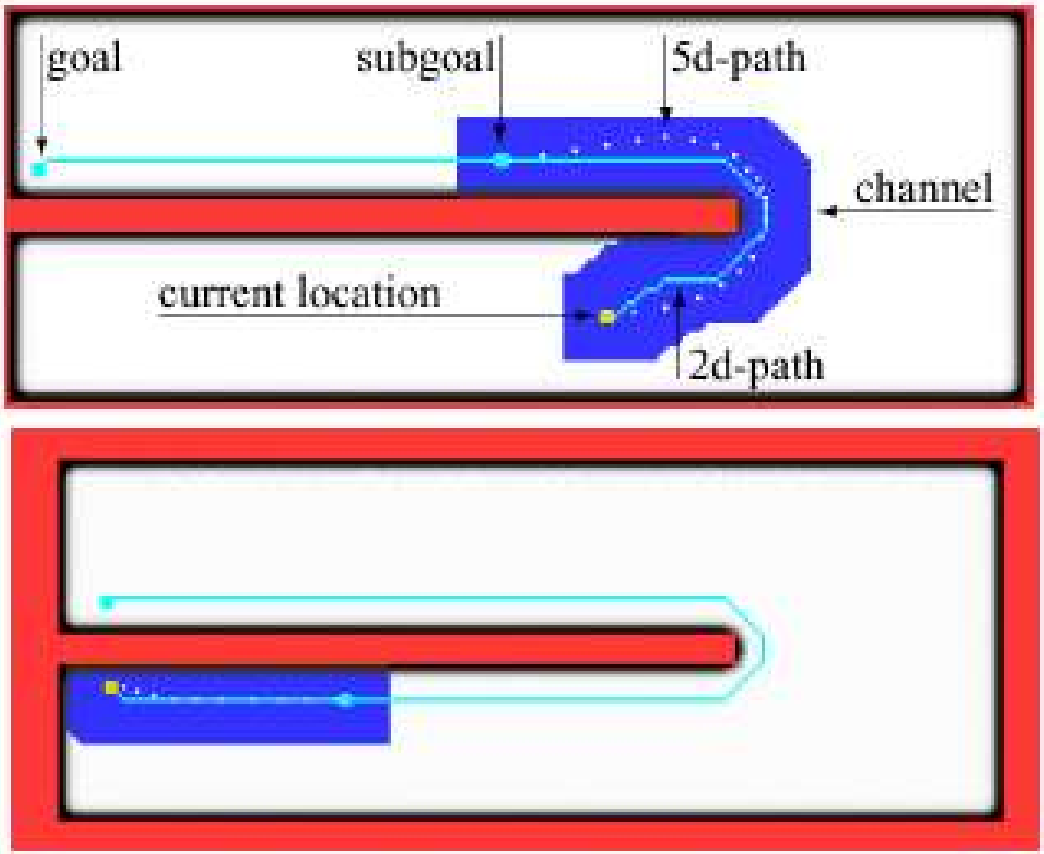


Trova il percorso in 5D

- 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.

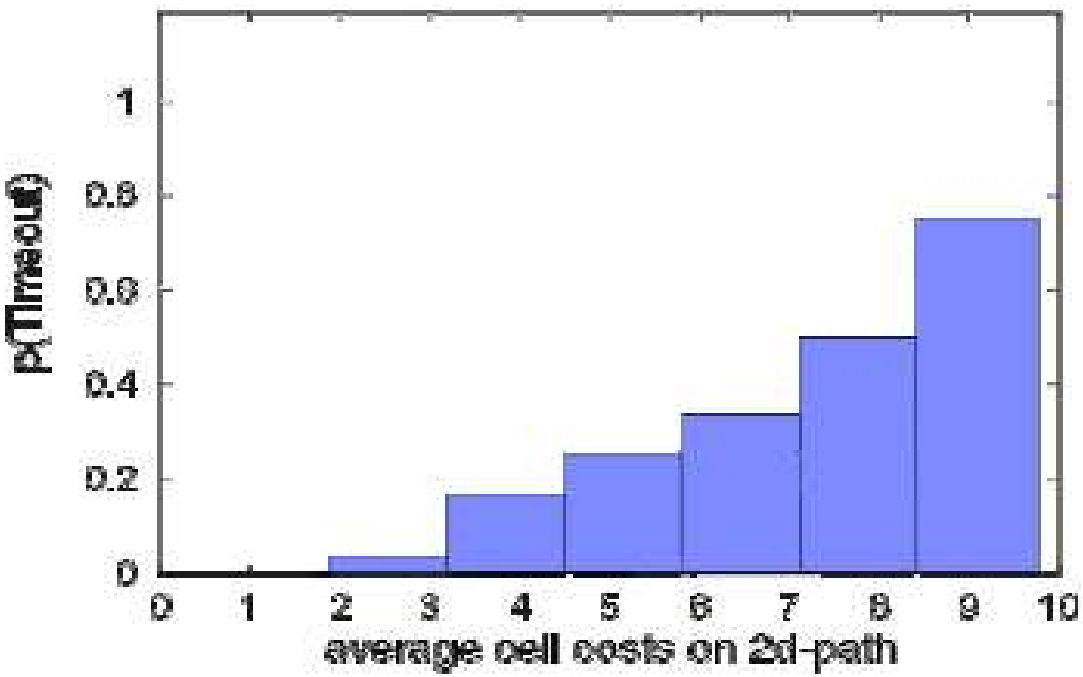
How to find an admissible steering command?

Generazione di Comandi

- Previous trajectory still admissible? → OK

- If not, drive on the 2d-path or use DWA to find new command.

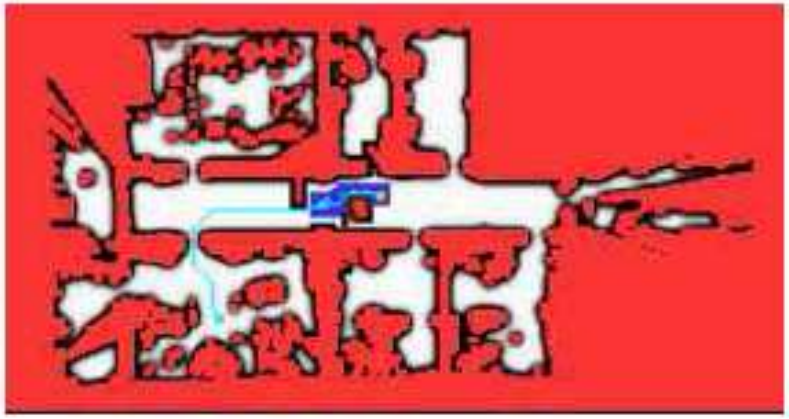
Timeout avoidance



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

Esempio

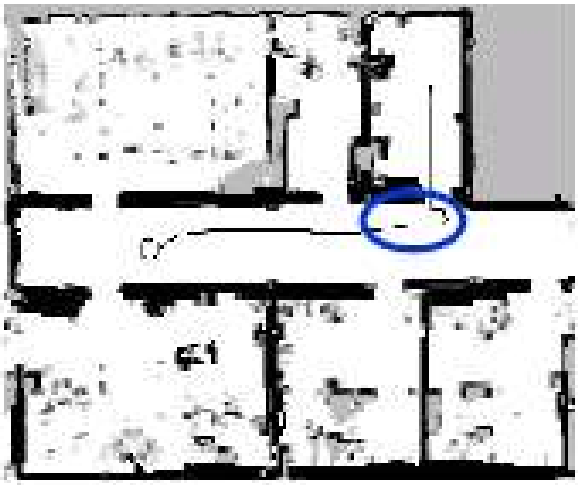
Robot Albert



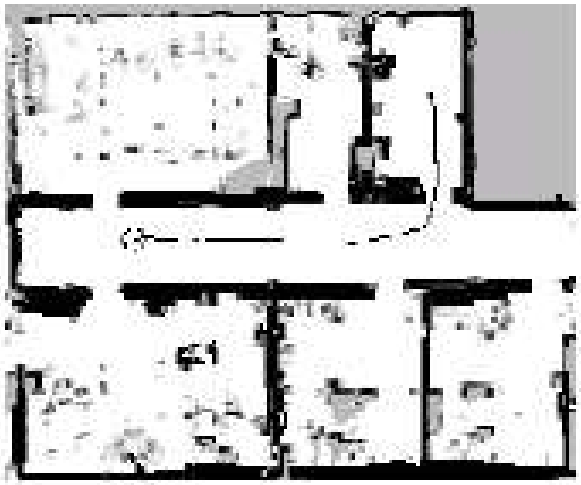
Planning state

Confronto con DWA

- DWAs often have problems entering narrow passages.

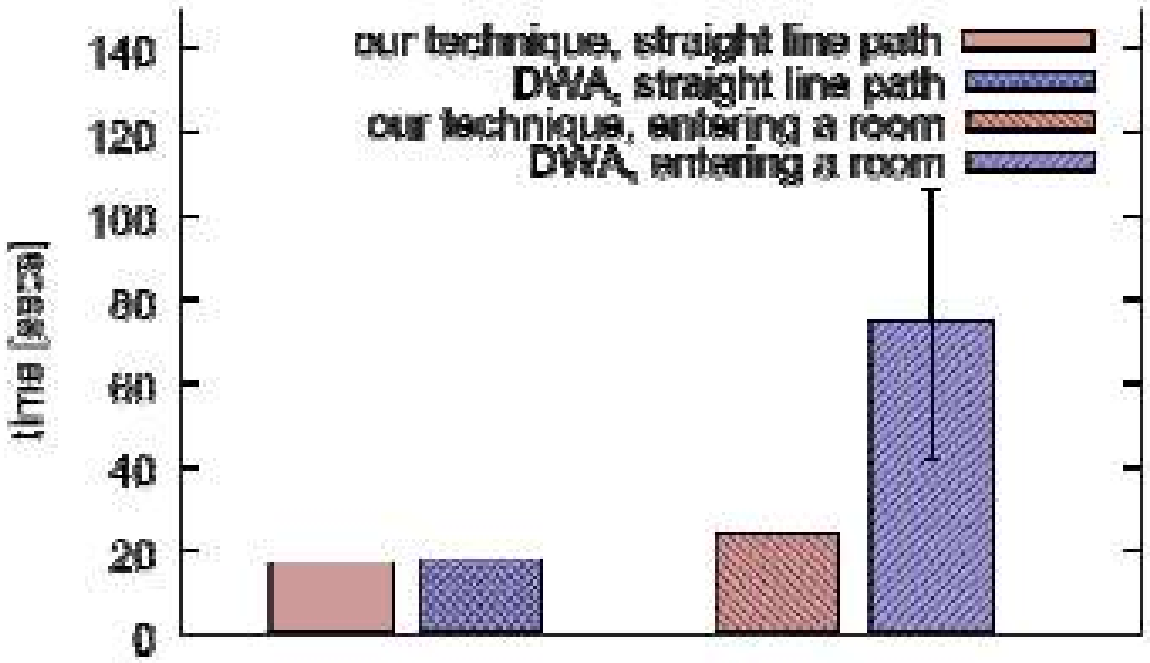


DWA planned path.



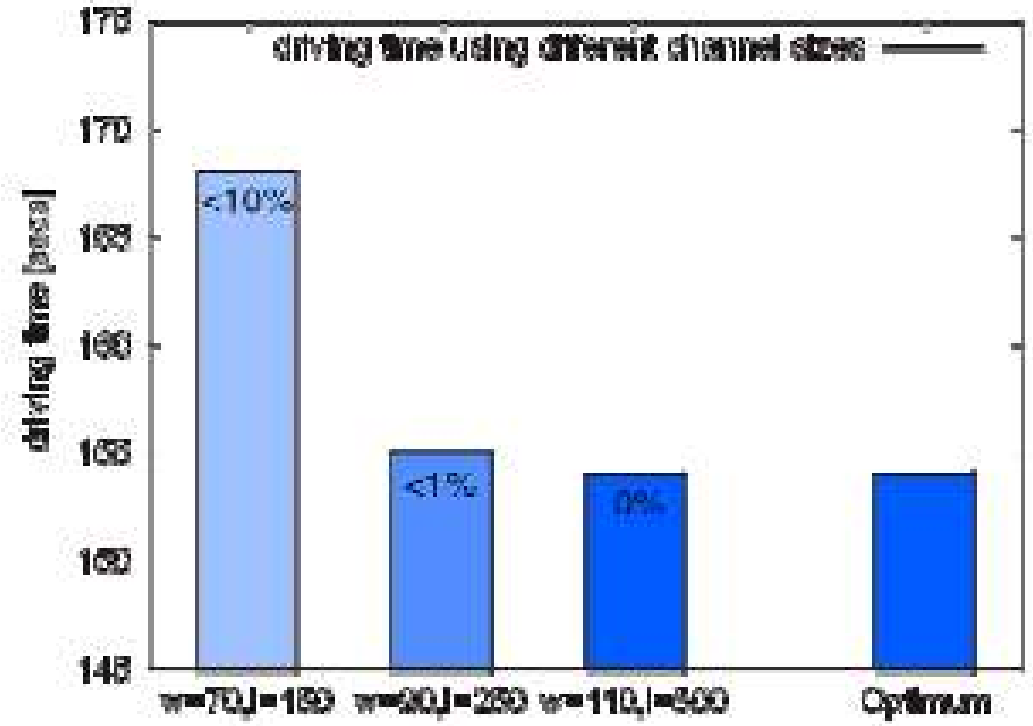
5D approach.

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.

Path Planning Probabilistico

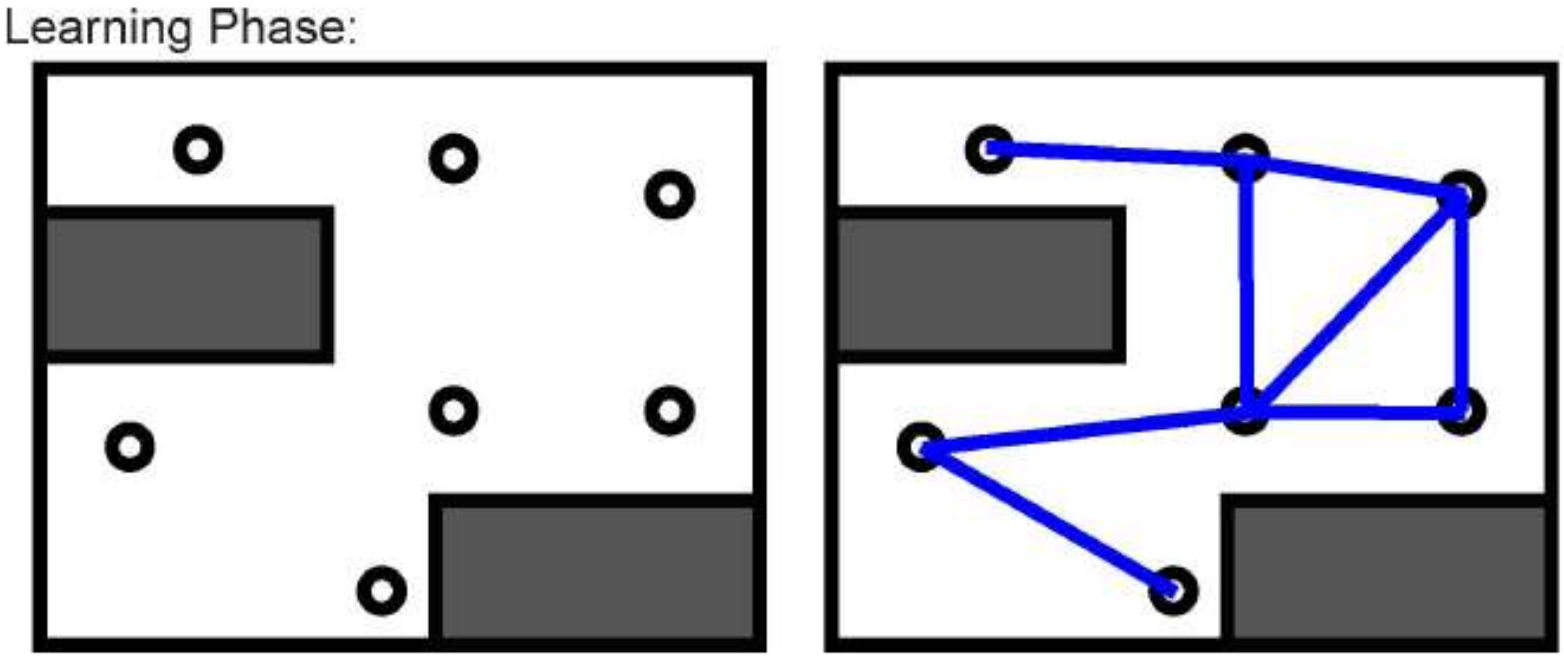
- Continuous state spaces
- Continuous actions

- PRM (Kavraki & many successors)
- RRT (Lavalle & many successors)

Probabilistic Roadmap

- Separate planning into two stages
  - "Learning" Phase
    - random samples of free configurations (vertices)
    - Attempt to connect pairs of nearby vertices with a local planner
    - if a valid plan is found, add an edge to the graph
  - Query Phase
    - find local connections to graph from initial and goal positions

Fase di Learning



Configurazione random nello spazio libero, poi connessa da archi collegando tutti i k vicini a distanza minore di n

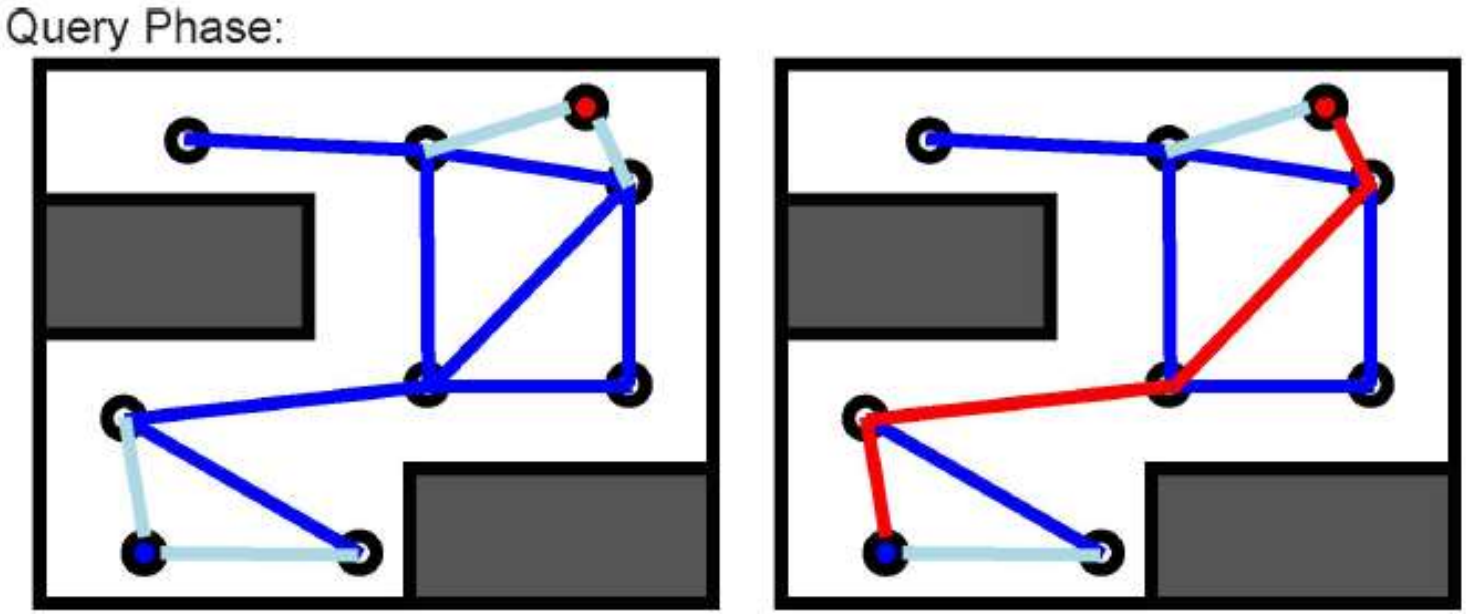
Fase di Learning

Algorithm 1: PRM (preprocessing phase)

```
1 V ← ∅; E ← ∅;
2 for i = 0, ..., n do
3     x_rand ← SampleFree_i;
4     U ← Near(G = (V, E), x_rand, r) ;
5     V ← V ∪ {x_rand};
6     foreach u ∈ U, in order of increasing ‖u − x_rand‖, do
7         if x_rand and u are not in the same connected component of G = (V, E) then
8             if CollisionFree(x_rand, u) then E ← E ∪ {(x_rand, u), (u, x_rand)};
9 return G = (V, E);
```

Configurazione random nello spazio libero, poi connessa da archi collegando tutti i k vicini a distanza minore di n

Fase di Query



Nella fase di Query si connette goal e posizione iniziale e si cerca lo shortest path (A*, Dijkstra)

E’ probabilisticamente completo: all’aumentare dei punti, la probabilità di non trovare il percorso va a zero

Fase di Query

Algorithm 2: sPRM

```
1 V ← {x_init} ∪ {SampleFree_i}_{i=1,...,n}; E ← ∅;
2 foreach v ∈ V do
3     U ← Near(G = (V, E), v, r) \ {v};
4     foreach u ∈ U do
5         if CollisionFree(v, u) then E ← E ∪ {(v, u), (u, v)}
6 return G = (V, E);
```

Nella fase di Query si connette goal e posizione iniziale e si cerca lo shortest path (A*, Dijkstra)

E' probabilisticamente completo: all'aumentare dei punti, la probabilità di non trovare il percorso va a zero

Rapidly Exploring Random Trees

- RRT
  - Explore continuous spaces efficiently
    - No need for an artificial grid
  - Form the basis for probabilistically complete planner
- Complete planners exist, but are slow
- RRT uses random search and approximation for speed

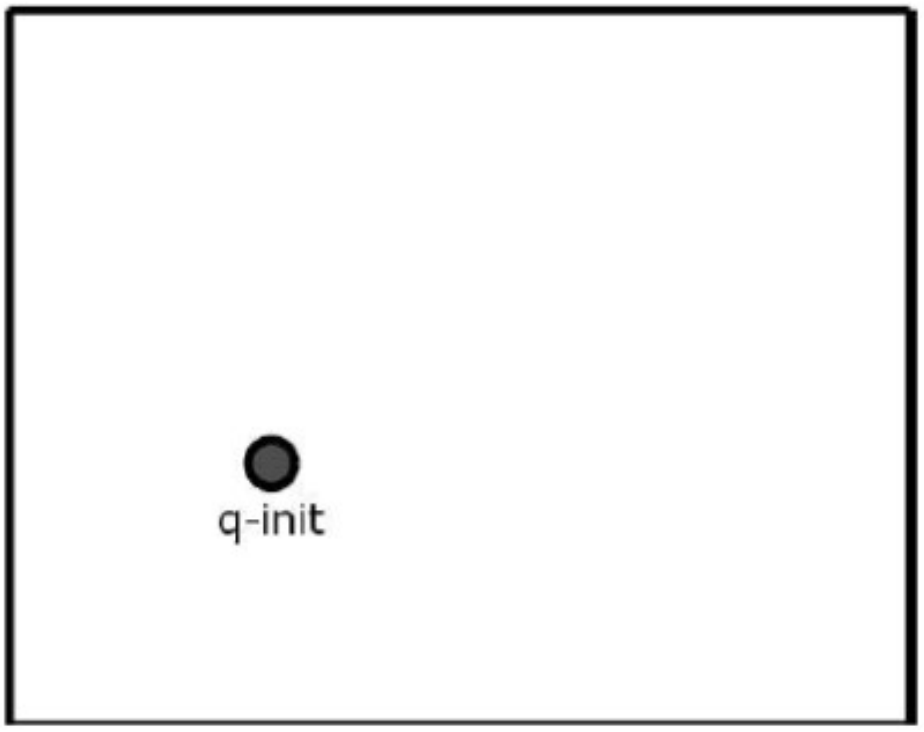
RRT: Algoritmo

Basic RRT Algorithm:

(1) Initially, start with the initial configuration as root of tree
(2) Pick a random state in the configuration space
(3) Find the closest node in the tree
(4) Extend that node toward the state if possible
(5) Goto (2)

RRT: Esempio (inizio)

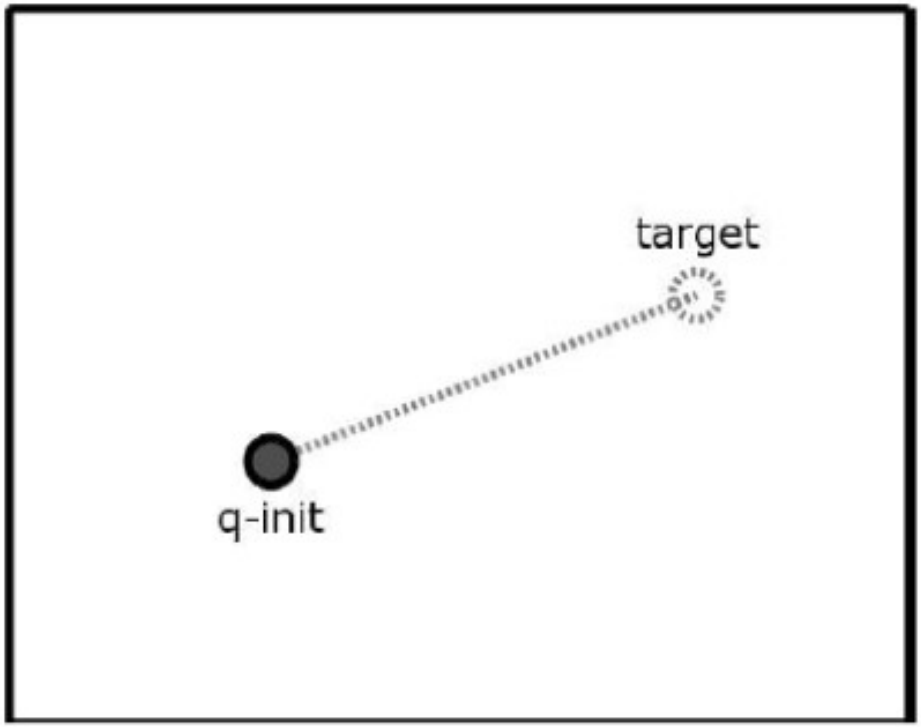
(1) Start with the initial state as the root of a tree



RRT: Esempio (Esplorazione)

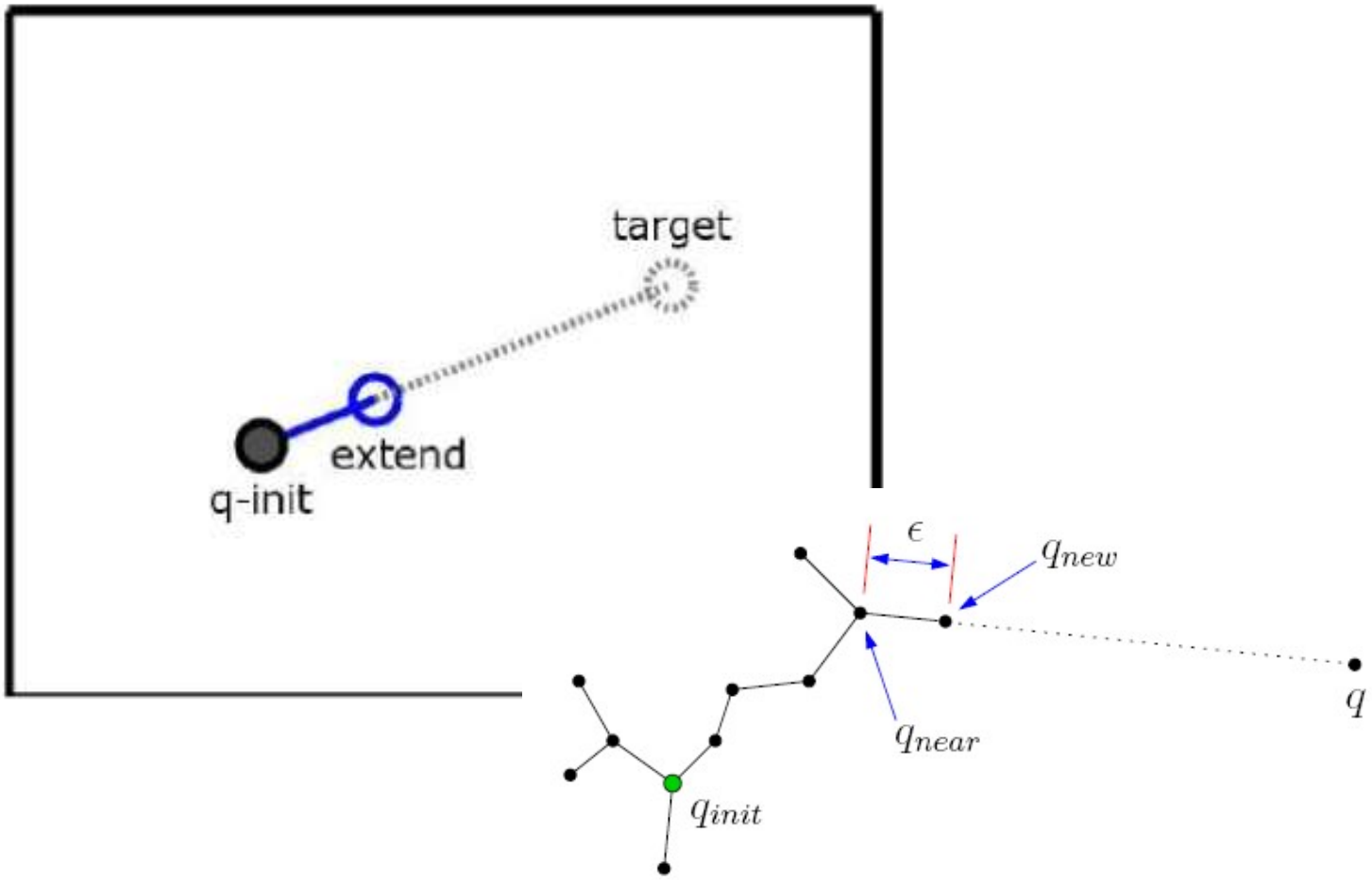
(2) Pick a random state in the environment

(3) Find the closest node in the tree

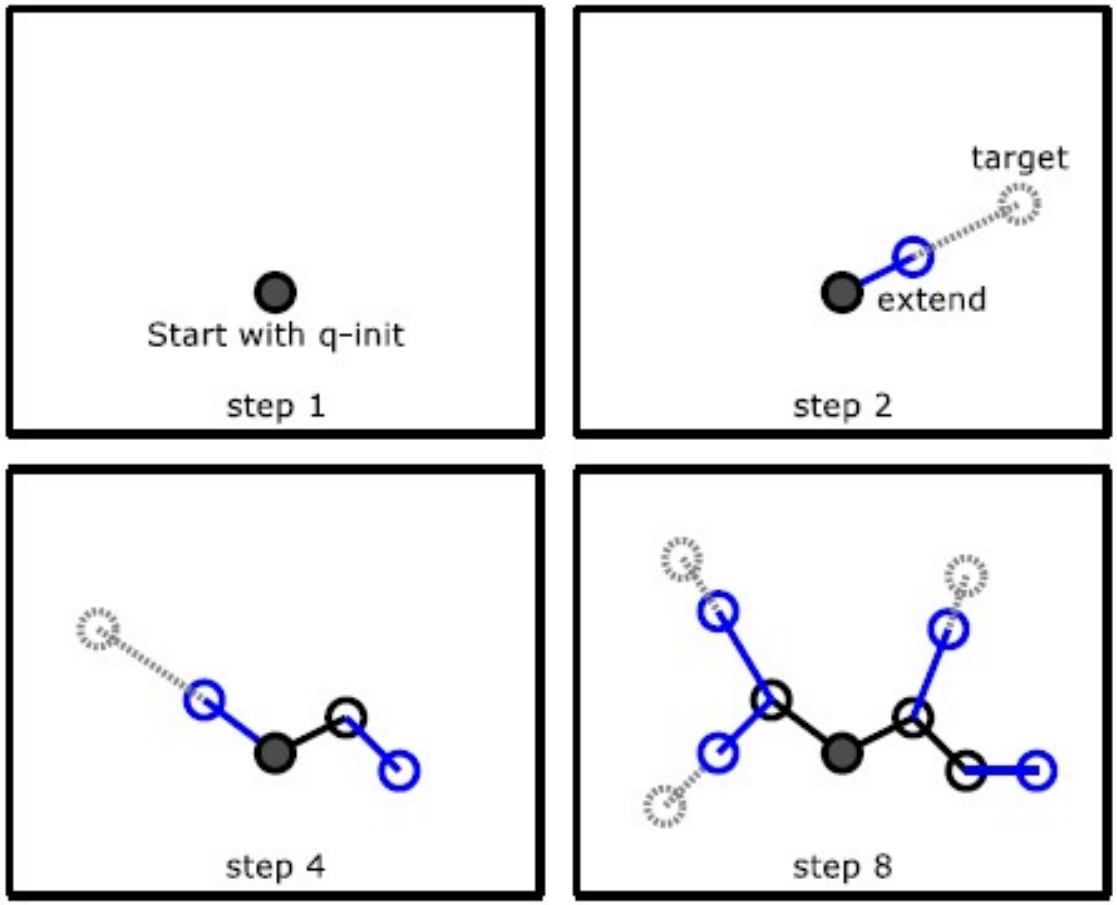


RRT: Esempio (Ricerca)

(4) Extend that node toward the target

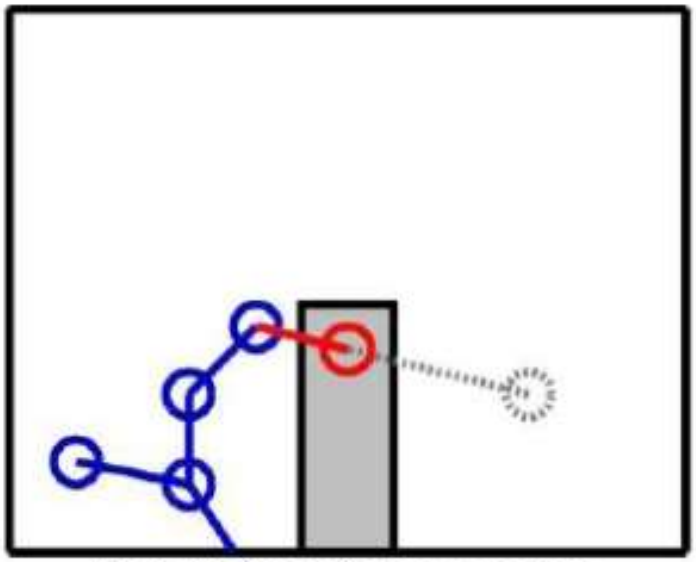


RRT: Esempio (Ricerca)

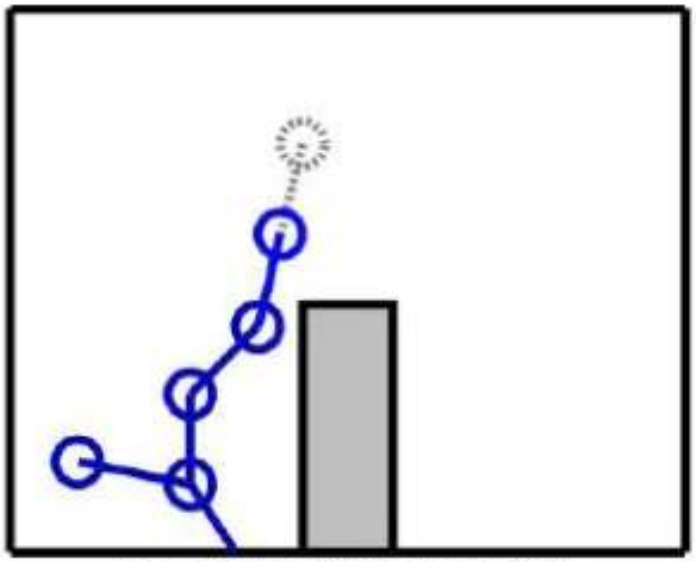


RRT: Ostacoli

- Ignore extensions which hit obstacles
- Resulting tree contains *only* valid paths



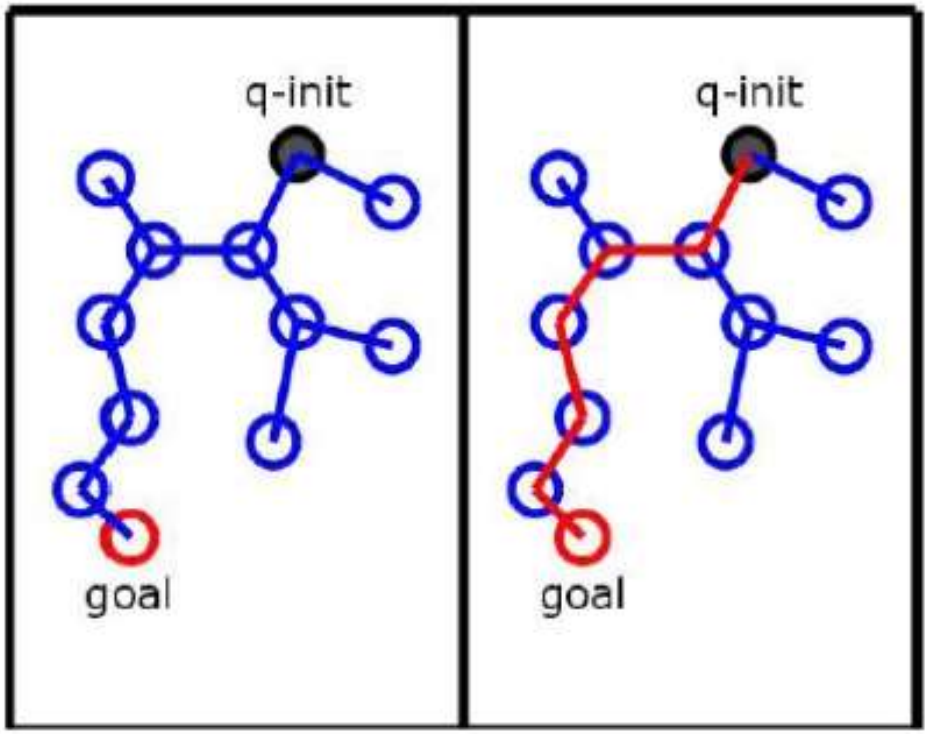
Ignore invalid extension



Record valid extension

RRT per Planning

- Once a node of the tree is a *goal*, the plan is the path back up the tree



RRT: Algoritmo

Algorithm BuildRRT
Input: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq
Output: RRT graph G

G.init(q_{init})

for k = 1 **to** K
  $q_{rand} \leftarrow$ RAND_CONF()
  $q_{near} \leftarrow$ NEAREST_VERTEX(q_{rand}, G)
  $q_{new} \leftarrow$ NEW_CONF(q_{near}, Δq)
  G.add_vertex(q_{new})
  G.add_edge(q_{near}, q_{new})
return G

Algorithm 3: RRT.

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
8 **return** $G = (V, E)$;

RRT Orientato al Gol

1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - Random configuration with probability 1-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

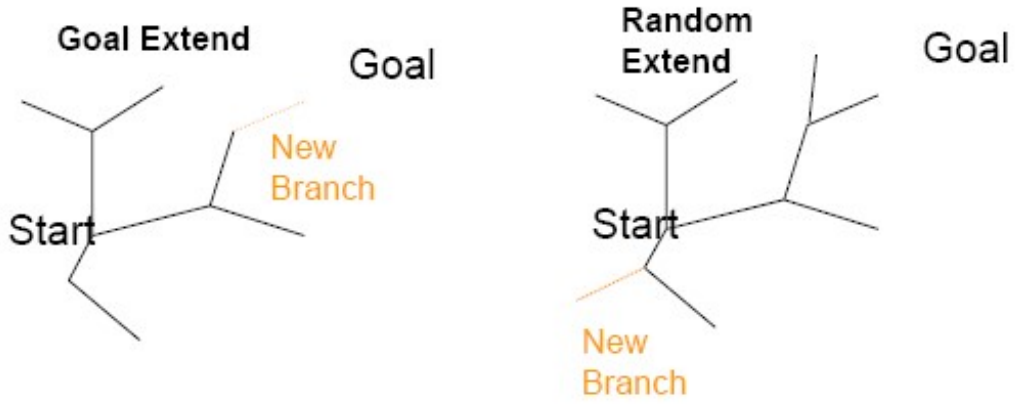
RRT Orientato al Gol

1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - Random configuration with probability 1-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

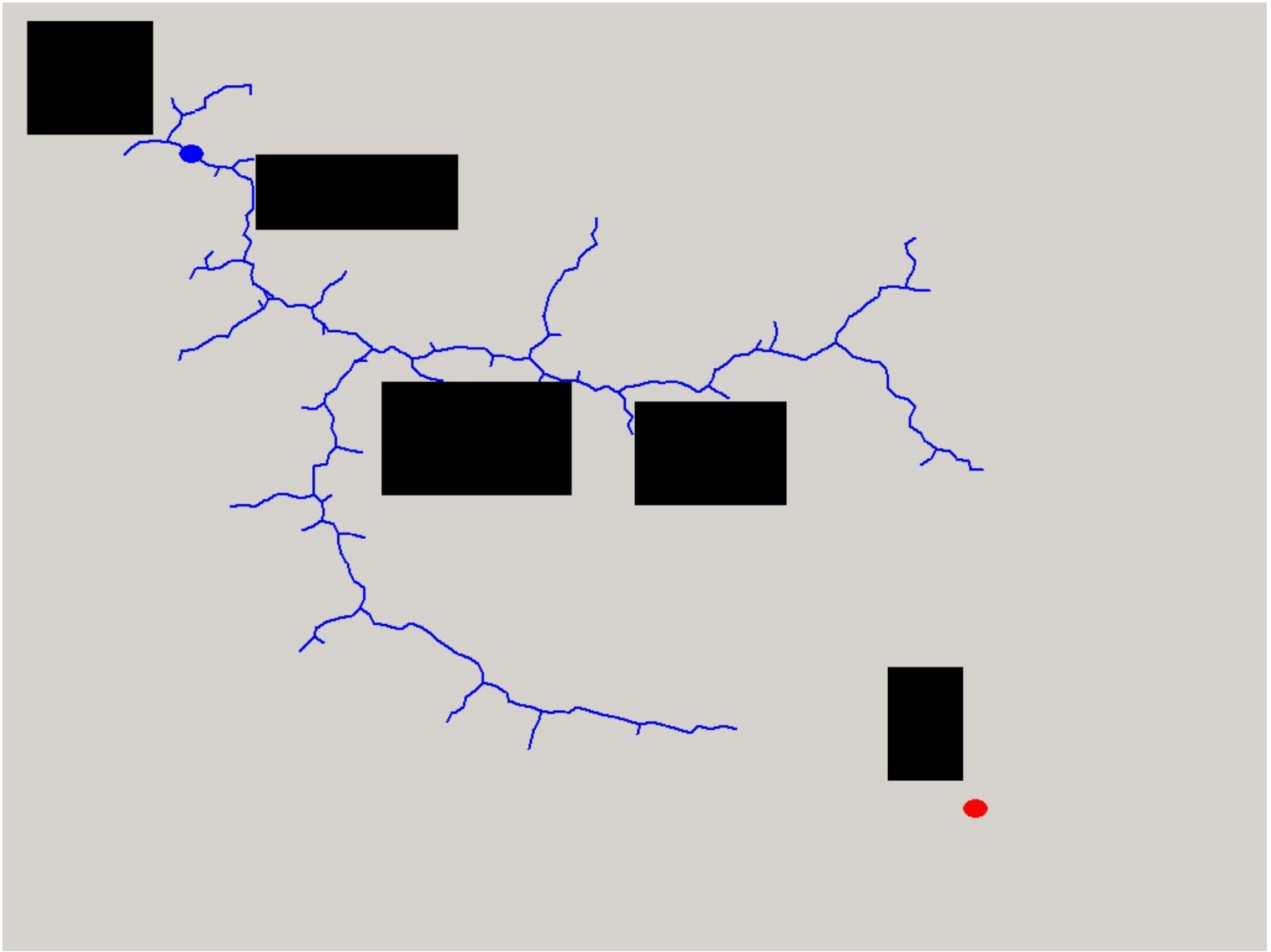
RRT per Planning

Probability *p* : Extend *closest node* in tree towards goal

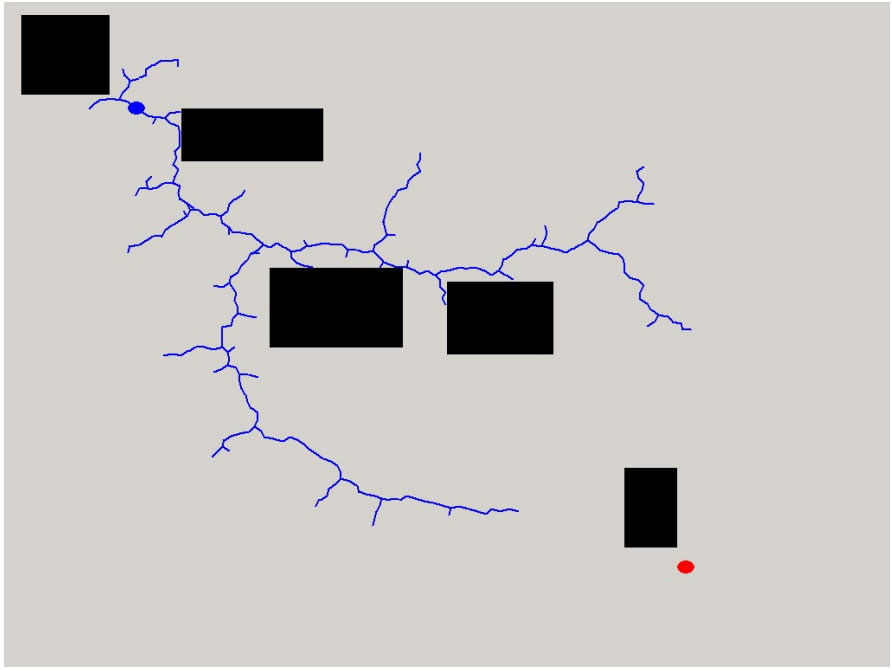
Probability *1-p* : Extend closest node towards a random point



RRT



RRT: Algoritmo



```
function RRTPlan (env:environment,initial:state,
                  goal:state):rrt-tree
    var nearest,extended,target:state;
    var tree:rrt-tree;
    nearest := initial;
    rrt-tree := initial;
    while(Distance (nearest,goal) < threshold)
        target = ChooseTarget (goal);
        nearest = Nearest (tree,target);
        extended = Extend (env,nearest,target);
        if extended ≠ EmptyState then
            AddNode (tree,extended);
    return tree;

function ChooseTarget (goal:state):state
    var p:real;
    p = UniformRandom in [0.0 .. 1.0];
    if 0 < p < GoalProb then
        return goal;
    else if GoalProb < p < 1 then
        return RandomState();

function Nearest (tree:rrt-tree,target:state):state
    var nearest:state;
    nearest := EmptyState;
    foreach state s in current-tree
        if Distance (s,target) <
        Distance (nearest,target) then
            nearest := s;
    return nearest;
```

RRT: Planning/Replanning

- Environments and planning
  - Value of p?
- Dynamic environments

- When failure, what to do?

RRT: Planning/Replanning

Plain RRT planner has little bias toward plan quality, and replanning is naive (all previous information is thrown out before replanning).

Waypoints:

- Previously successful plans can guide new search
- Biases can be encoded by modifying the target point distribution

Waypoint Cache:

- When a plan is found, store nodes into a bin with random replacement
- During target point selection, choose a random item from waypoint cache with probability q

ERRT: Execution extended RRT

Save past path as waypoints

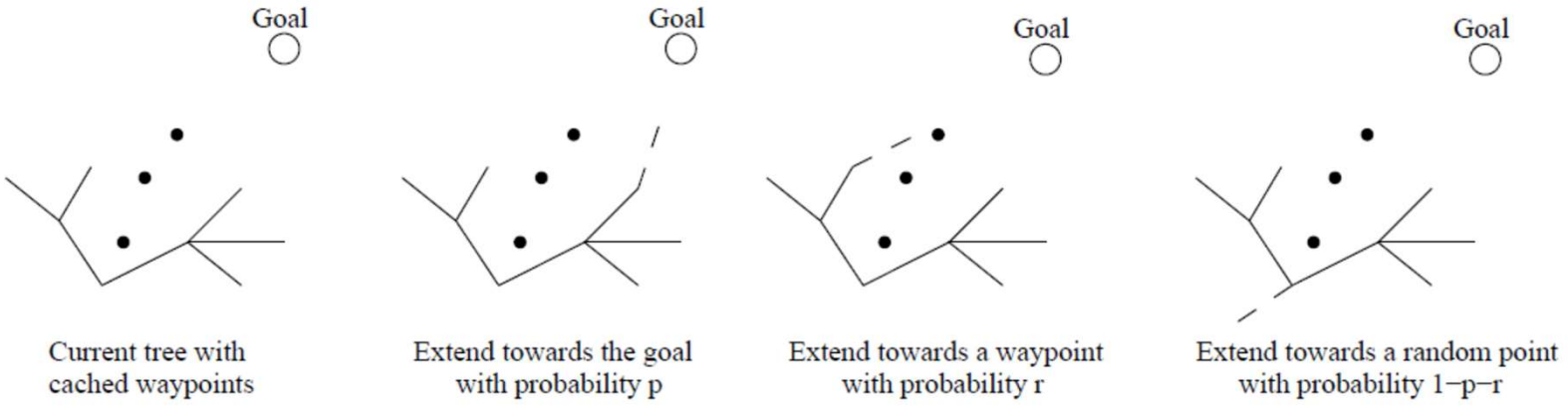
```
function ChooseTarget'(goal:state):state
    var p:real;
    var i:integer;
    p = UniformRandom in [0.0 .. 1.0];
    i = UniformRandom in [0 .. NumWayPoints-1];
    if 0 < p < GoalProb then
        return goal;
    else if GoalProb < p < GoalProb+WayPointProb
    then
        return WayPointCache[i];
    else if GoalProb+WayPointProb < p < 1 then
        return RandomState();
```

1) Start with initial state as root of tree
2) Pick a random target state
   - Goal configuration with probability p
   - **Random item from waypoint cache with probability q**
   - Random configuration with probability 1-q-p
3) Find the closest node in the tree
4) Extend the closest node toward the target
5) Goto step 2

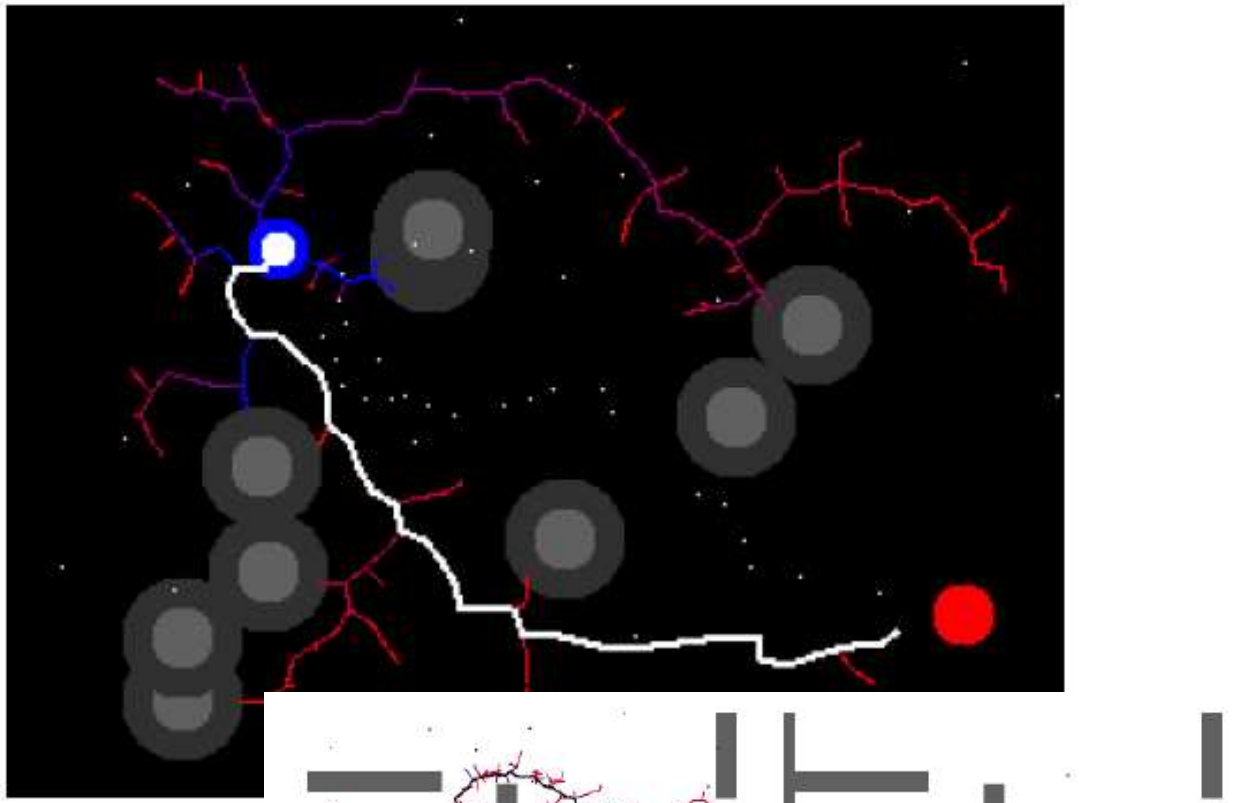
Bruce & Veloso, Real-time randomized path planning for robot navigation, 2003

ERRT: replanning

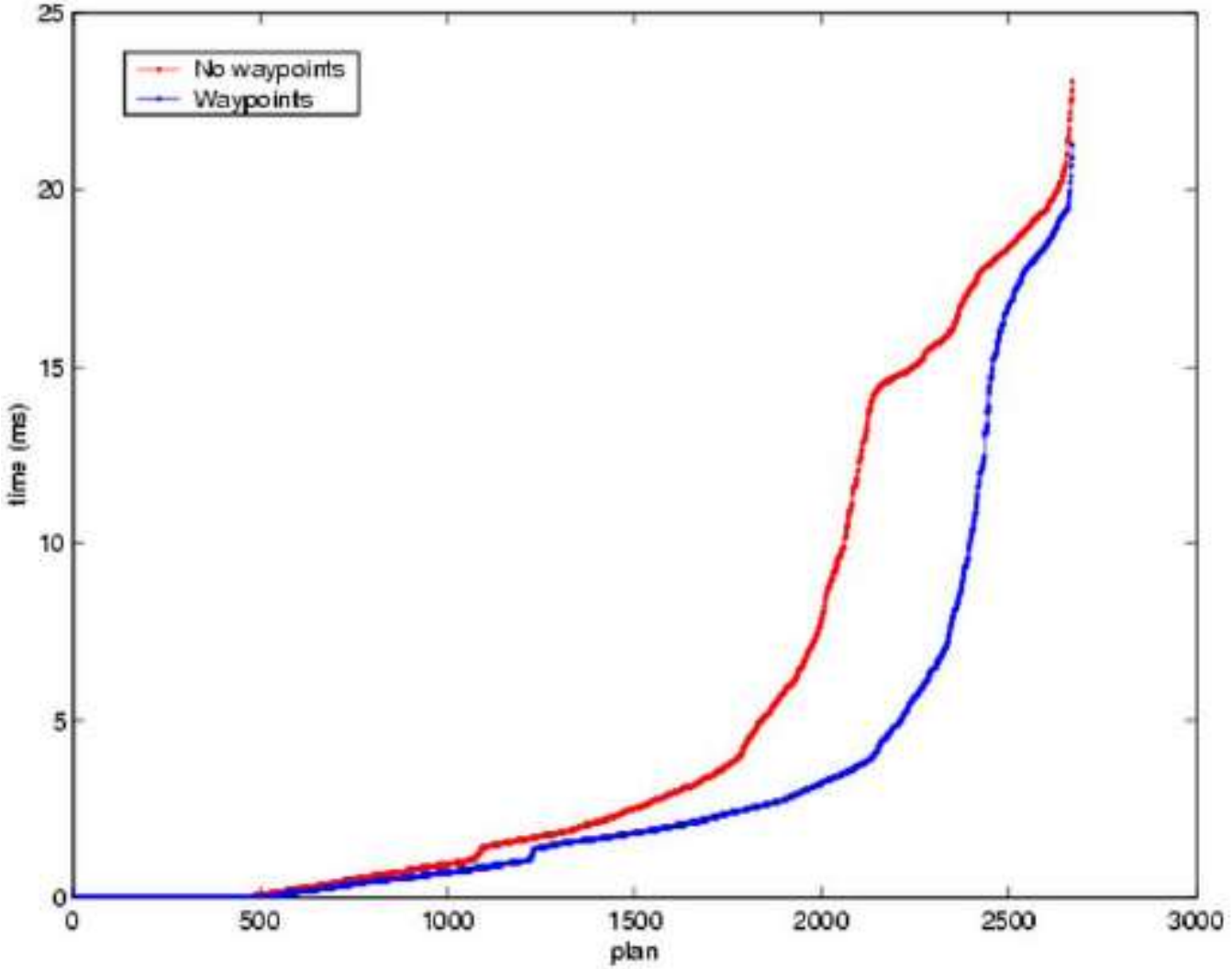
Probability *p* : Extend closest node in tree towards goal
Probability *r* : Extend closest node in tree towards random cache point
Probability *1-p-r* : Extend closest node towards a random point



ERRT: Replanning con waypoint



Prestazioni



Discussione

- Planning with RRT
  - High p – few known obstacles
  - Low p – many known obstacles
- Replanning with ERRT
  - High q – small dynamics (no state change)
  - Low q – high dynamics (lots of state change)
  - ERRT – bias to use previous plan; but could be any other bias
- RRT and ERRT – probabilistic convergence

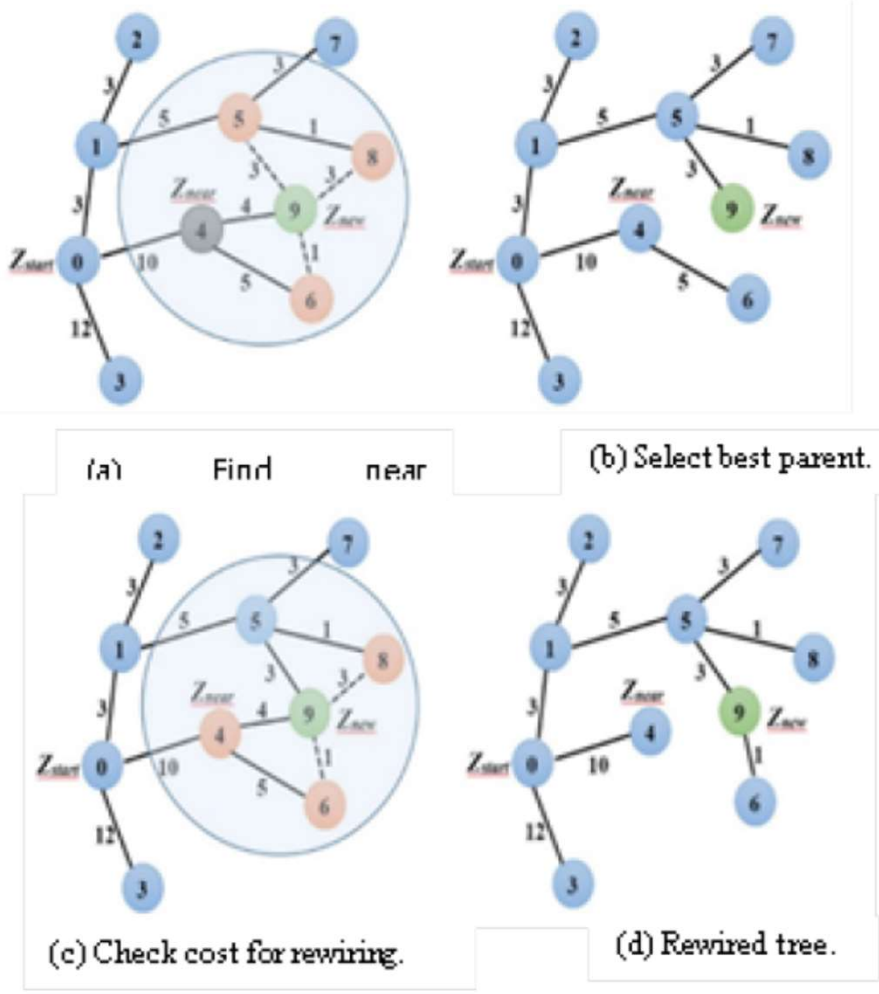
RRT*

- Ricerca dei vicini e riscrittura dell'albero
  - Ricerca:
    - miglior nodo parent per il nuovo nodo prima della sua inserzione nell'albero (nell'area di una sfera di raggio r).
  - Riscrittura:
    - rigenera l'albero nel raggio di area k per maintenere l'albero con costo minimo

S. Karaman and E. Frazzoli, "Sampling-Based Algorithms for Optimal Motion Planning" The International Journal of Robotics Research, 30(7), 2011 pp. 846–894.

RRT*

- Ricerca dei vicini e riscrittura dell'albero



Algorithm 1 The RRT* Algorithm

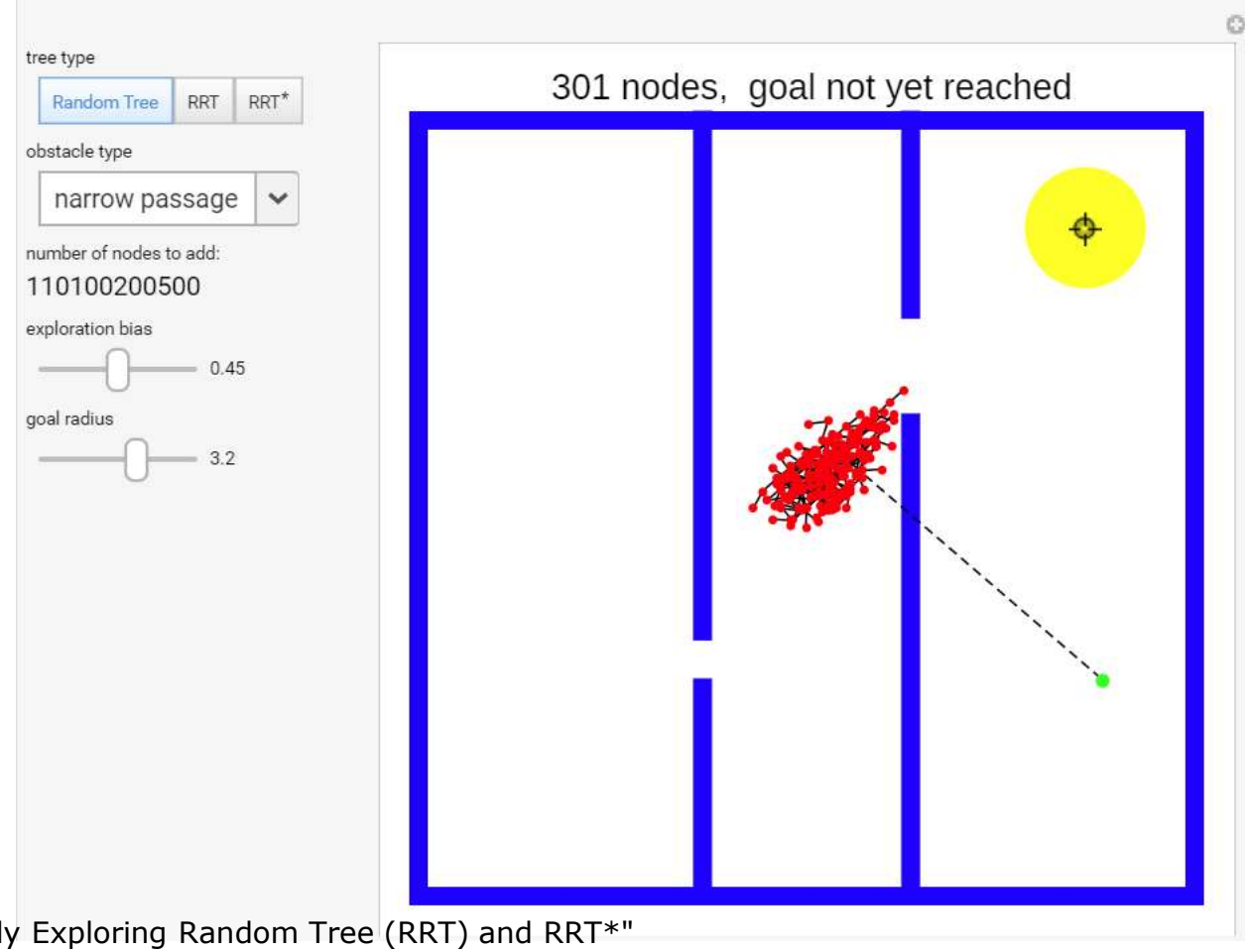
$V \leftarrow \{x_{init}\}$; $E \leftarrow \varnothing$; $T \leftarrow (V, E)$;
for $i = 1 \rightarrow N$ **do**
  $x_{rand} \leftarrow Sample(i)$;
  $X_{near} \leftarrow Near(V, x_{rand})$;
  if $X_{near} = \varnothing$ **then**
    $X_{near} \leftarrow Nearest(V, x_{rand})$;
  end if
  $x_{parent} \leftarrow FindBestParent(X_{near}, x_{rand})$;
  if $x_{parent} \neq$ NULL **then**
    $V \leftarrow V \cup \{x_{rand}\}$; $E \leftarrow E \cup \{(x_{parent}, x_{rand})\}$;
    $E \leftarrow Rewire(E, X_{near}, x_{rand})$;
  end if
end for
return $T = (V, E)$;

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  If $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      If $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      If $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

RT, RRT, RRT*

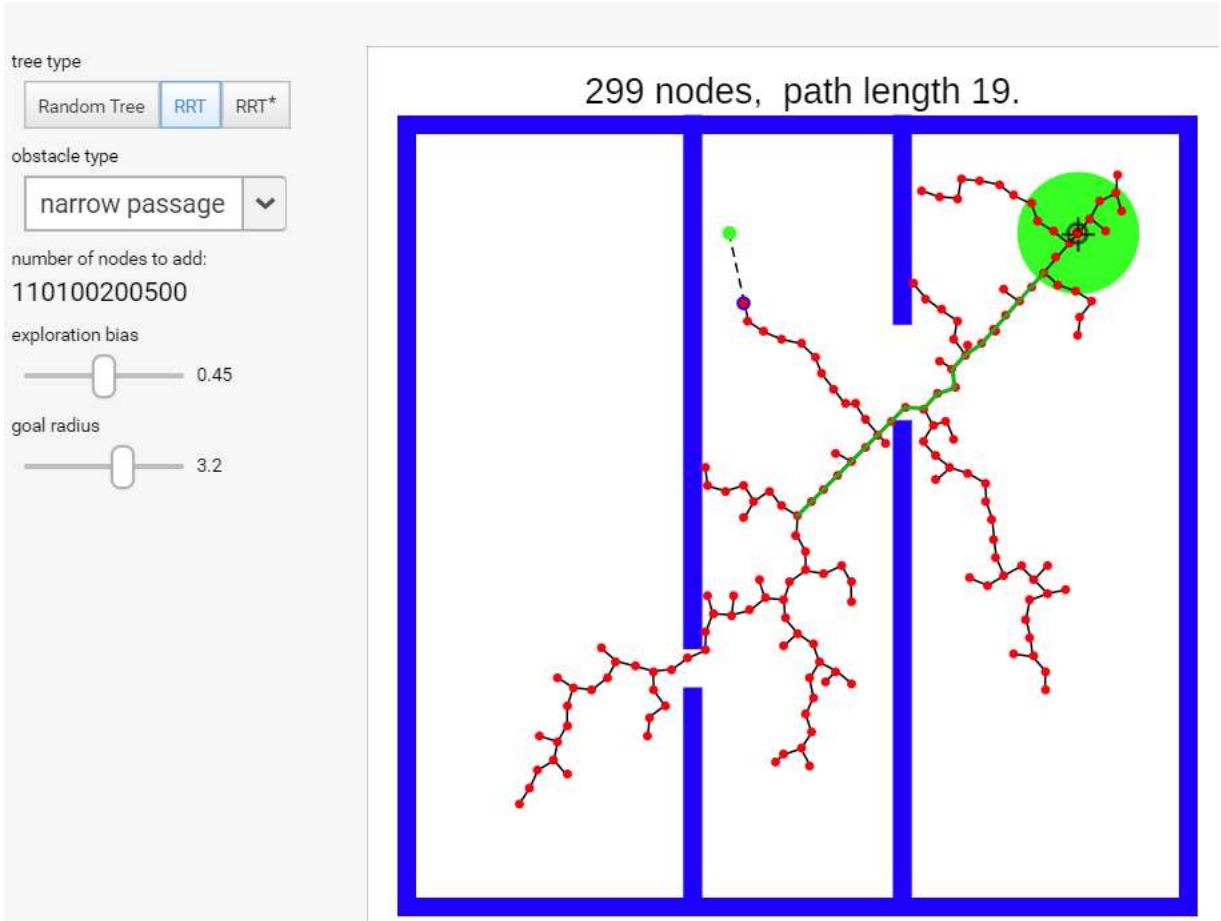
- Simulatore: Ramdom Tree



Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

RT, RRT, RRT*

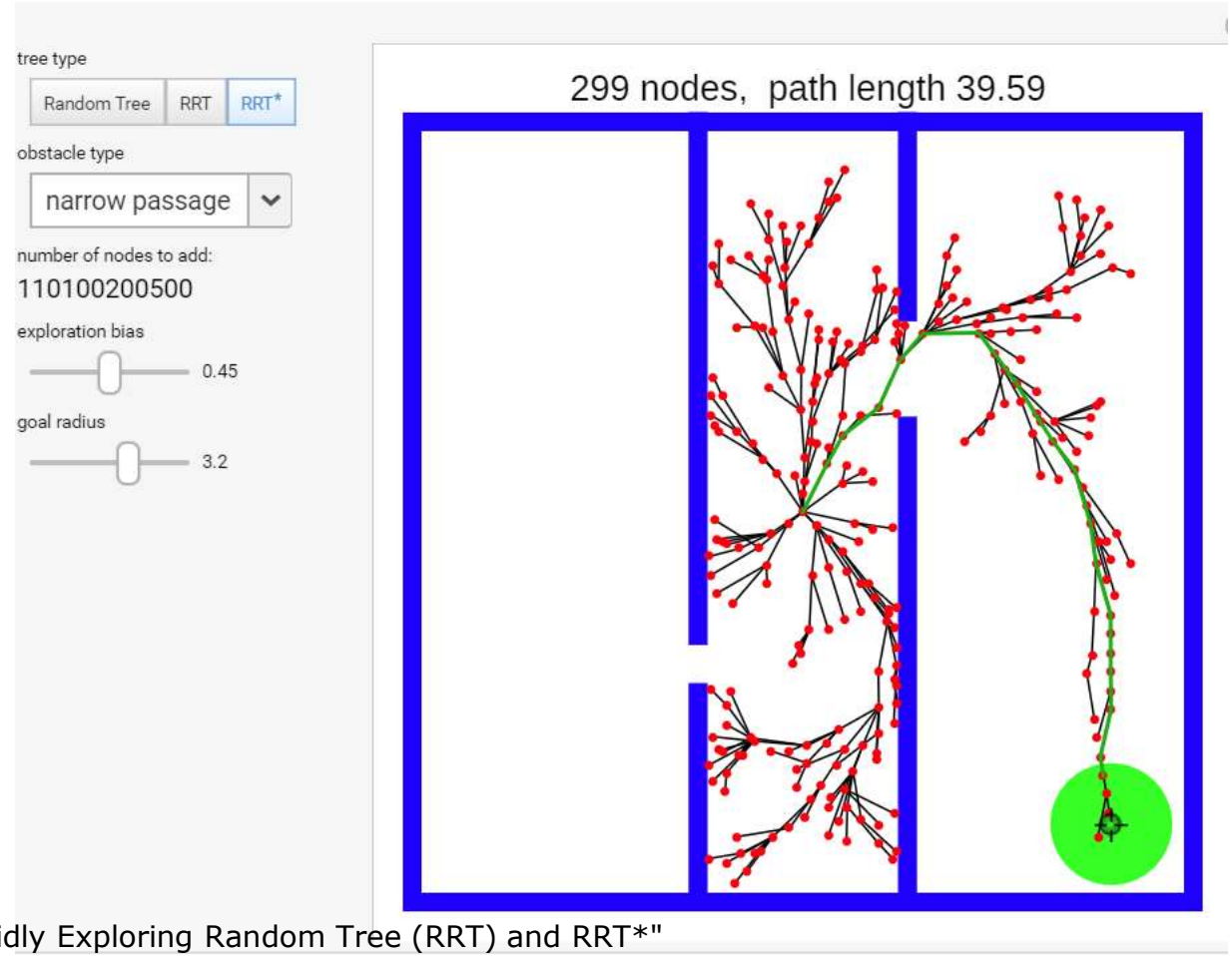
- Simulatore: RRT



Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

RT, RRT, RRT*

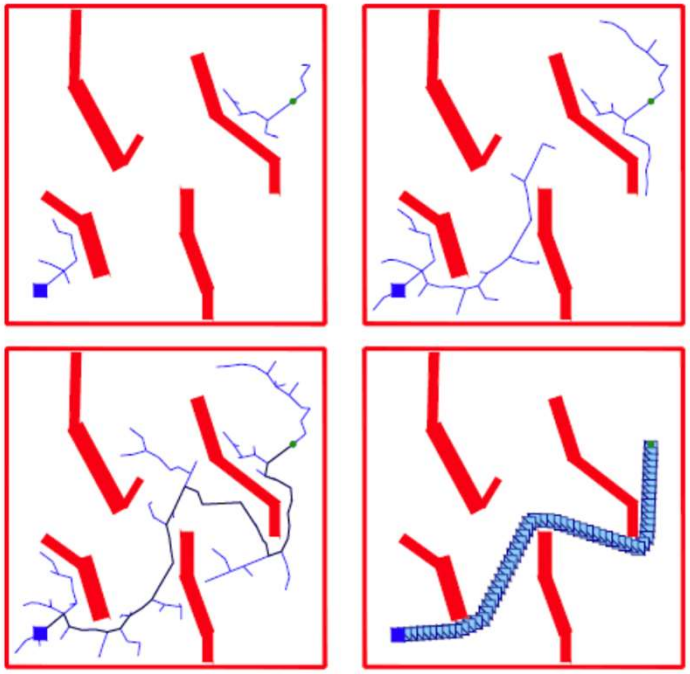
- Simulatore: RRT*



Li Huang "Rapidly Exploring Random Tree (RRT) and RRT*"
http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/
Wolfram Demonstrations Project

RRT-Connect

- Ricerca dal init e goal
- Extend e Connect

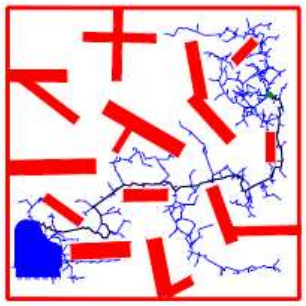


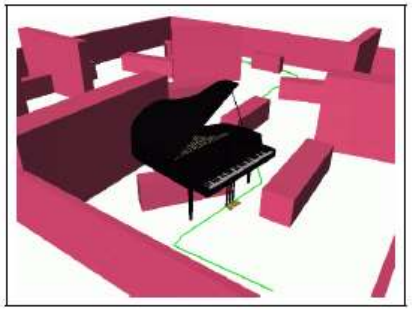
CONNECT($\mathcal{T}, q$)
1 **repeat**
2 $S \leftarrow$ EXTEND($\mathcal{T}, q$);
3 **until not** ($S =$ *Advanced*)
4 Return S;

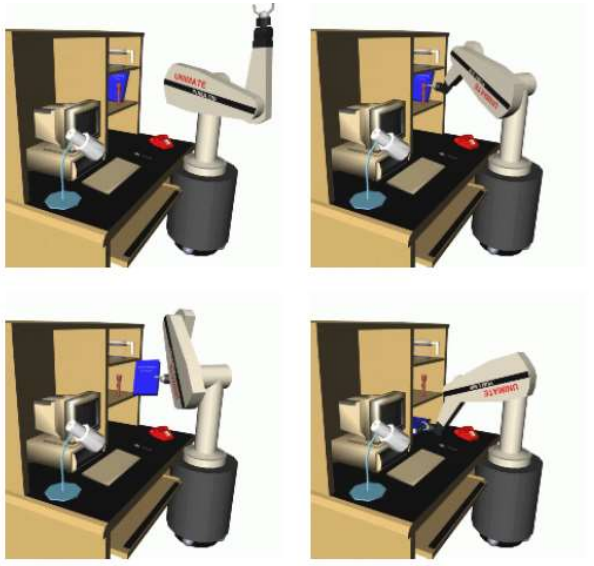
RRT_CONNECT_PLANNER(q_{init}, q_{goal})
1 $\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
2 **for** $k = 1$ **to** K **do**
3 $q_{rand} \leftarrow$ RANDOM_CONFIG();
4 **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
5 **if** (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
6 Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
7 SWAP($\mathcal{T}_a, \mathcal{T}_b$);
8 Return *Failure*

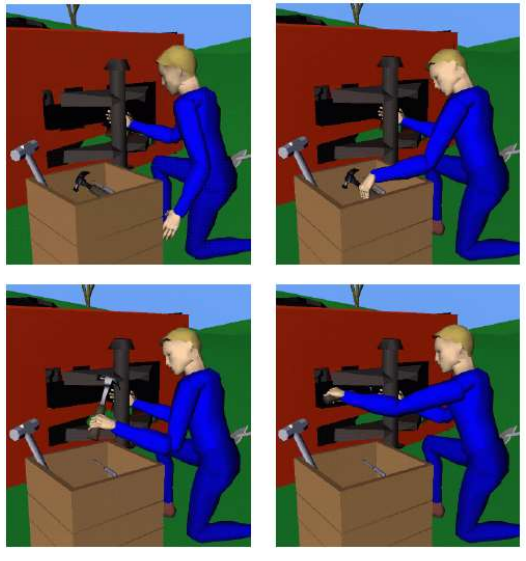
RRT-Connect: An Efficient Approach to Single-Query Path Planning. Kuffner and LaValle 2000.

RRT-Connect: Applicazioni









RRT-Connect: An Efficient Approach to Single-Query Path Planning. Kuffner and LaValle 2000.