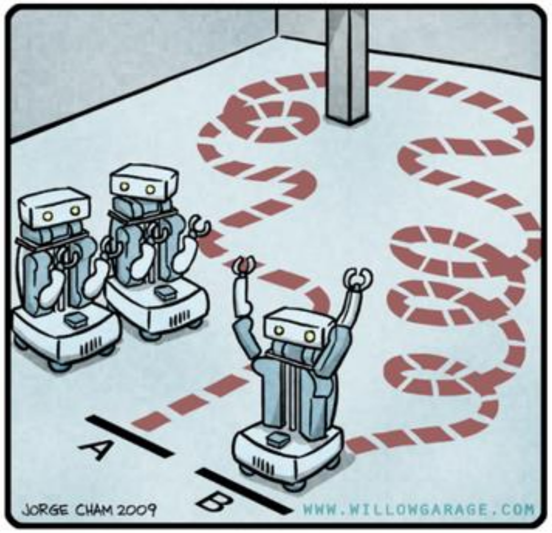
POLITECNICO DI MILANO

Robot Motion Control

Matteo Matteucci – matteo.matteucci@polimi.it

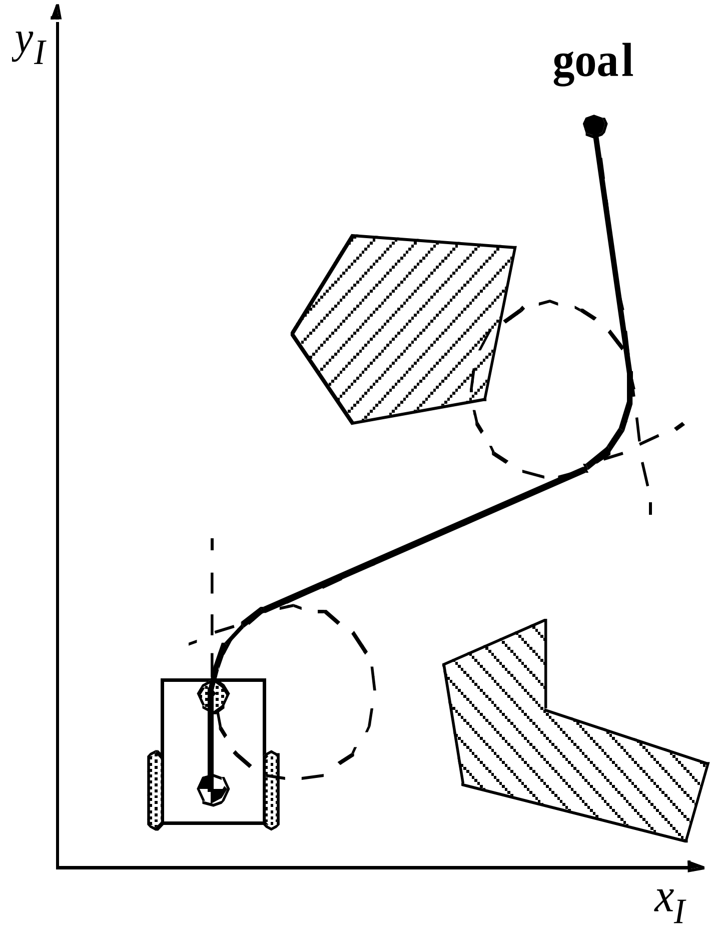


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Open loop control

A mobile robot is meant to move from one place to another

- Pre-compute a smooth trajectory based on motion segments (e.g., line and circle segments) from start to goal
- Execute the planned trajectory along the way till the goal

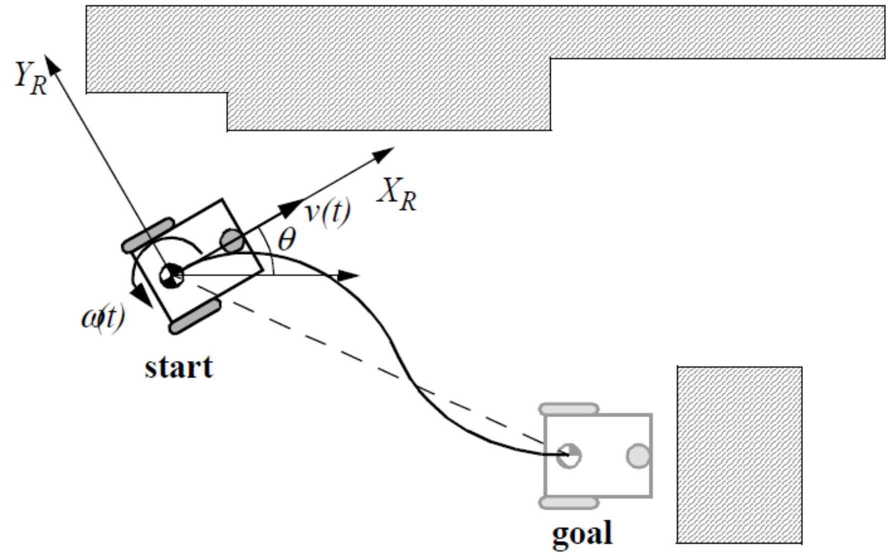


Disadvantages:

- Not an easy task to pre-compute a feasible trajectory
- Limitations and constraints of the robots velocities and accelerations
- Does not handle dynamical changes in the environment
- Resulting trajectories are usually not smooth

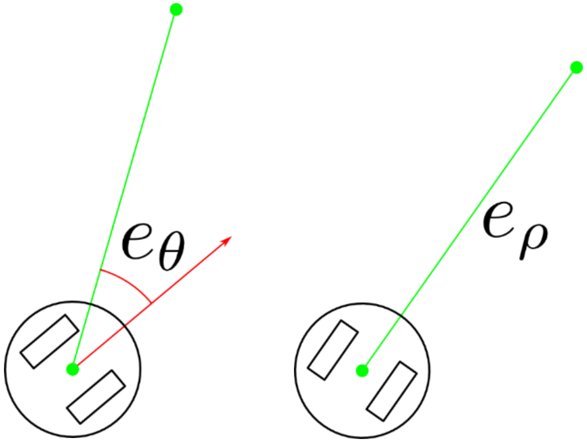
Feedback control (diff drive example)

With feedback control the trajectory is recomputed and adapted online



We can design a simple control schema for path following:

- First we close a speed control loop on the wheels
- Then divide the problem in:
  - Control of the orientation
  - Control of the distance



Control orientation acting on angular velocity

Control distance acting on linear velocity

Feedback control (diff drive example)

With feedback control the trajectory is recomputed and adapted online

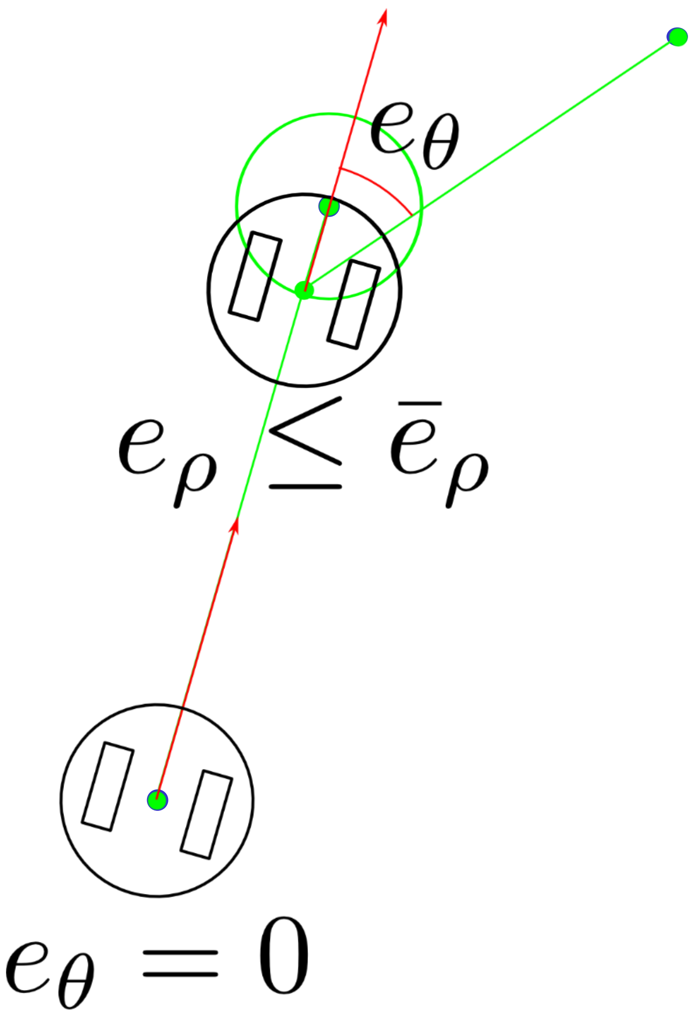
We can design a simple control schema for path following:

- First we close a speed control loop on the wheels
- Then divide the problem in:
    - Control of the orientation
    - Control of the distance

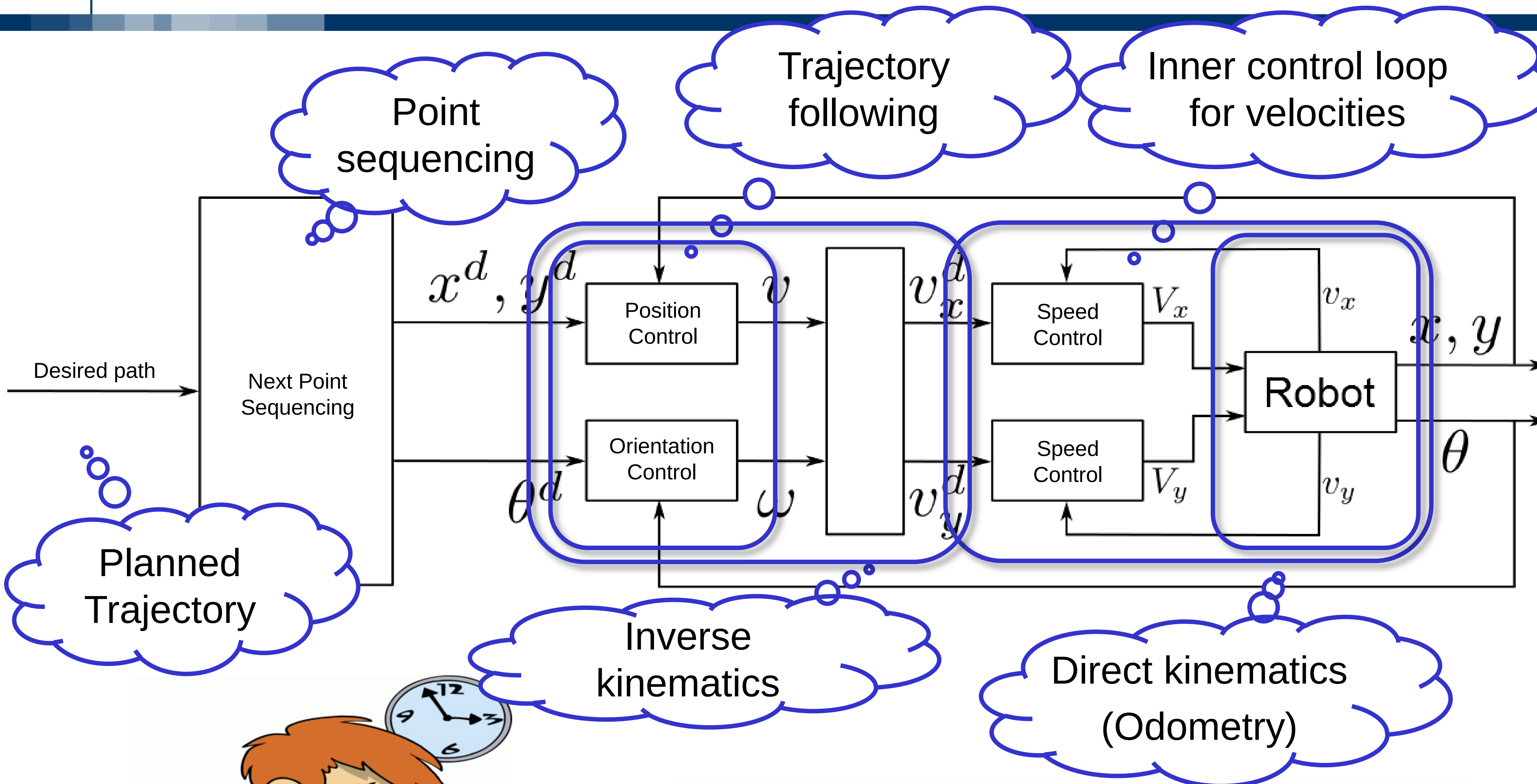
Control orientation acting on angular velocity

Control distance acting on linear velocity

A simple logic handles the next point



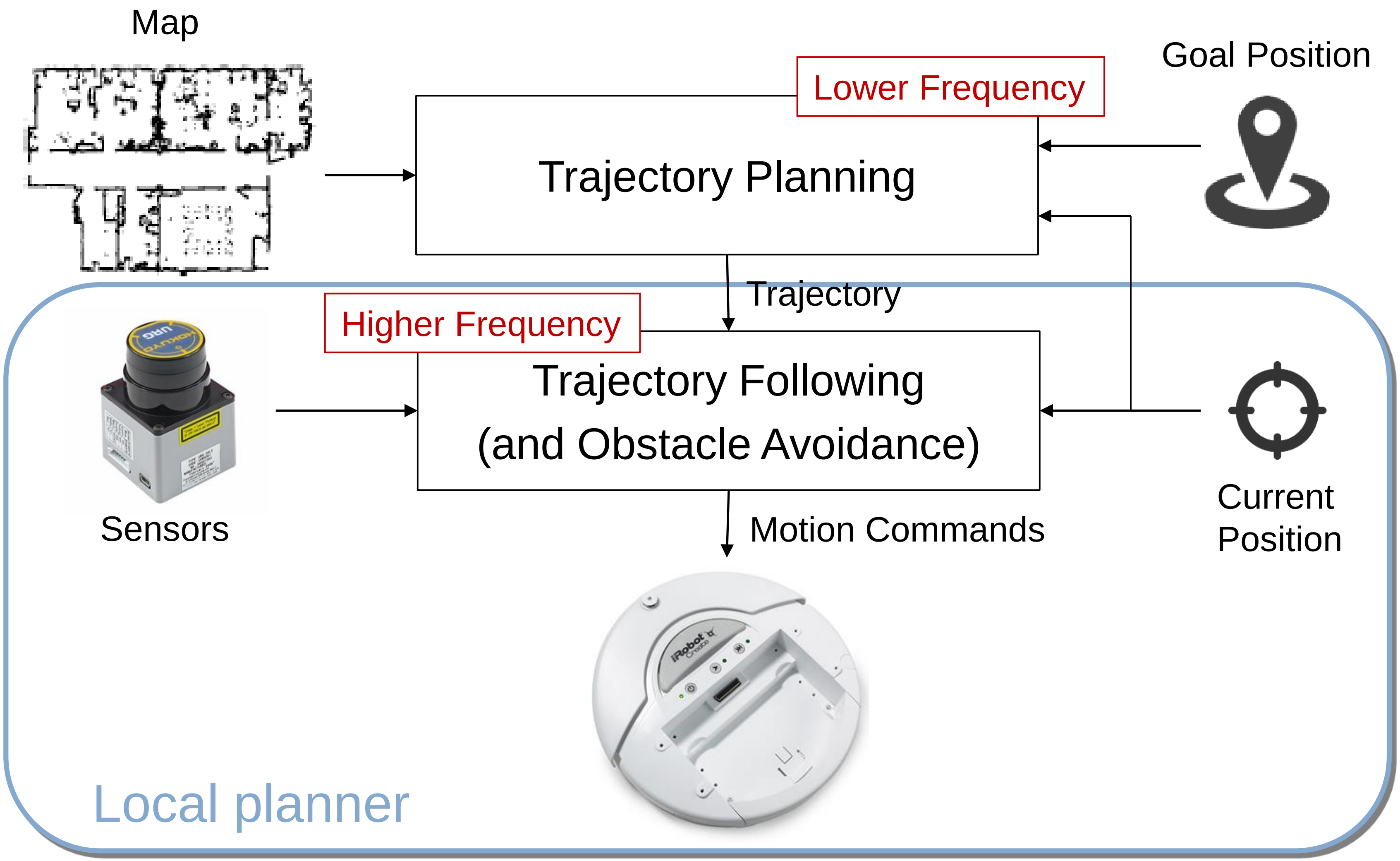
Feedback control (diff drive example)



What about obstacles?
Who does provide the trajectory?
Do you trust direct kinematics?

Matteo Matteucci – matteo.matteucci@polimi.it

A Two Layered Approach



Obstacle Avoidance (Local Path Planning)

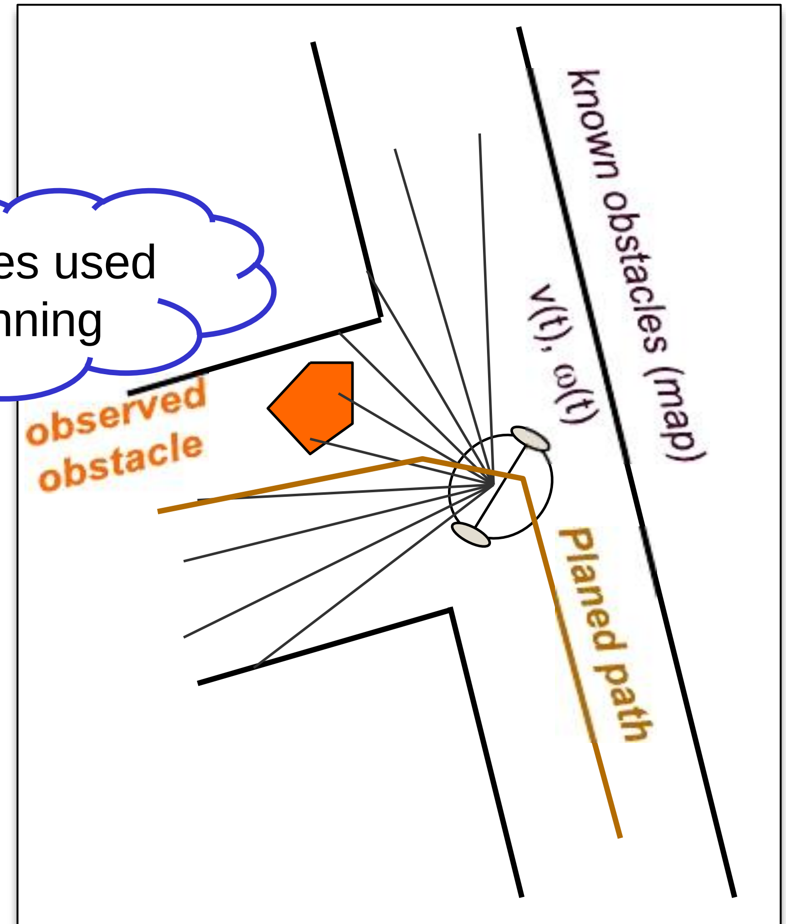
Obstacle avoidance should:

- Follow the planned path
- Avoid unexpected obstacle,
  i.e., those that were not in the map

Several proposed methods in the l

Sometimes used for planning

- Potential field methods
  [Borenstein, 1989]
- Vector field histogram
  [Borenstein, 1991, 1998, 2000]
- Nearness diagram
  [Minguez & Montano, 2000]
- Curvature-Velocity [Simmons, 1996]
- Dynamic Window Approach
  [Fox, Burgard, Thrun, 1997]
- …



Matteo Matteucci – matteo.matteucci@polimi.it

The Simplest One …

"Bugs" have little if any knowledge …

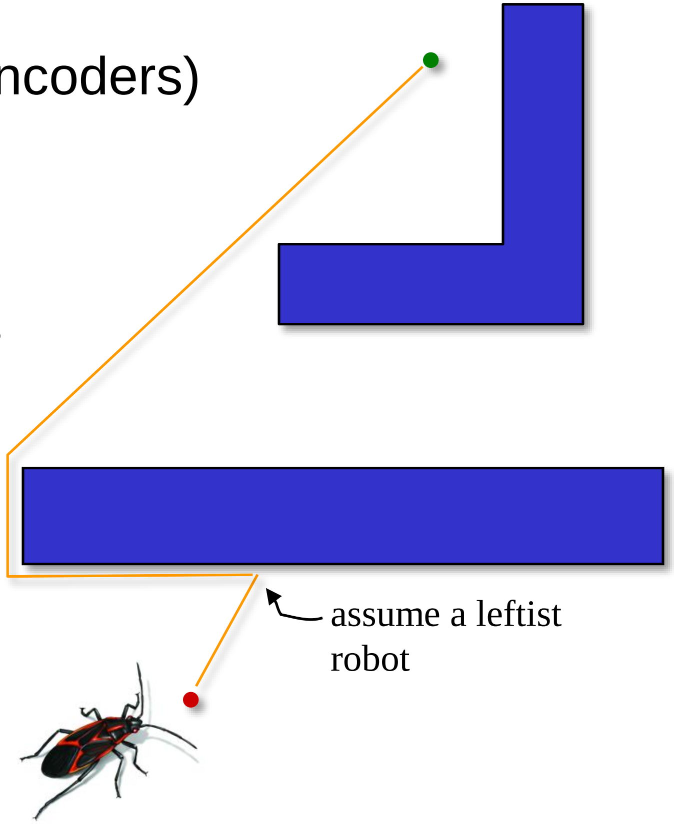
- known direction to the goal
- only local sensing (walls/obstacles + encoders)

… and their world is reasonable!

- finite obstacles in any finite range
- a line intersects an obstacle finite times

Switch between two basic behaviors

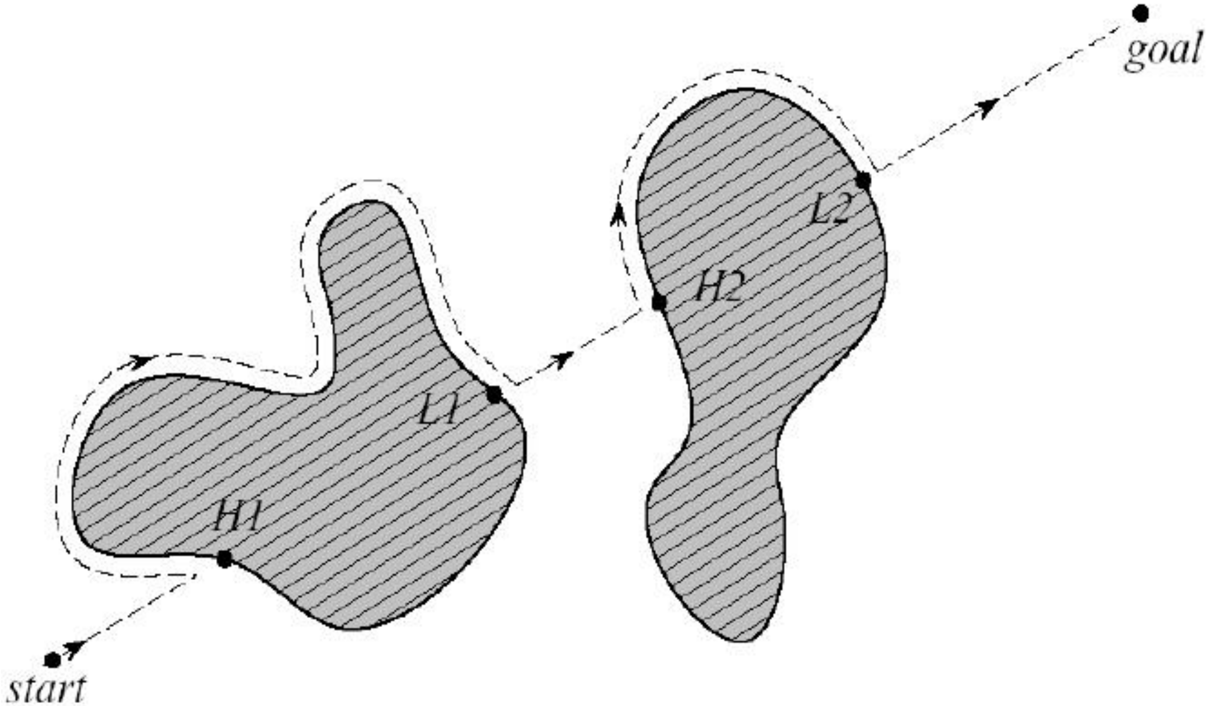
1. head toward goal
2. follow obstacles until you can head toward the goal again

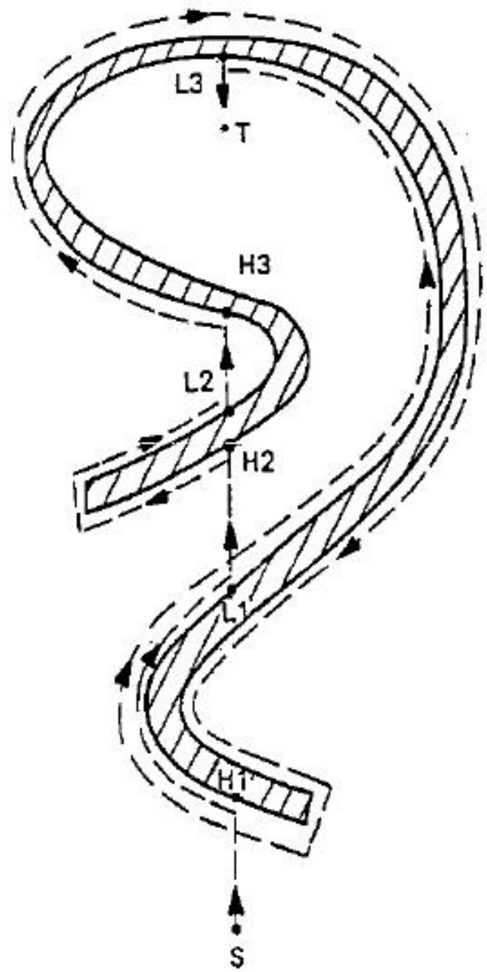


Bugs and Features …

Each obstacle is fully circled before it is left at the point closest to the goals

- Advantages
  - No global map required
  - Completeness guaranteed
- Disadvantages
  - Solution are often highly suboptimal





Matteo Matteucci – matteo.matteucci@polimi.it

Vector Field Histograms (VHF) *[Borenstein et al. 1991]*

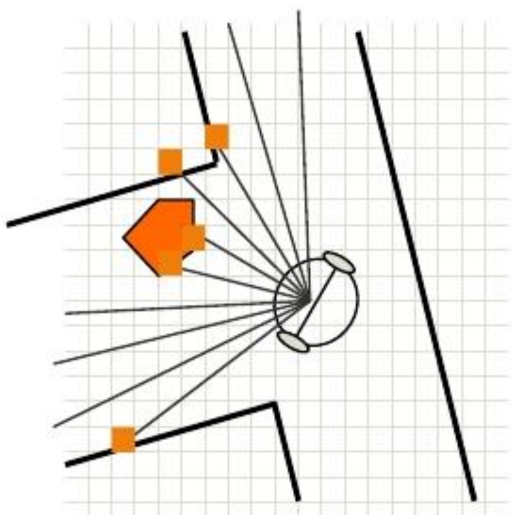
Use a local map of the environment and evaluate the angle to drive towards

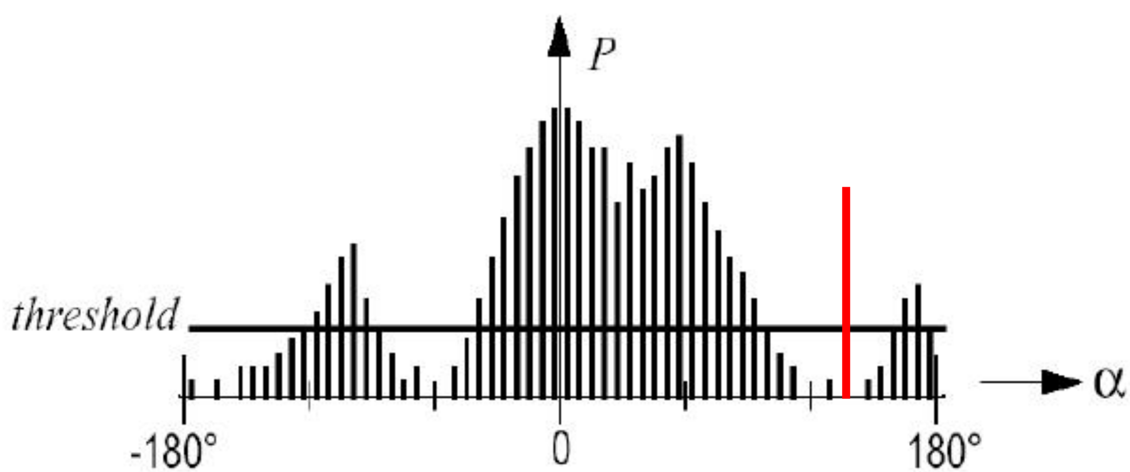
- Environment represented in a grid (2 DOF) with
- The steering direction is computed in two steps:
  - all openings for the robot to pass are found
  - the one with lowest cost function *G* is selected

$$G = a \cdot \text{target\_direction} + b \cdot \text{wheel\_orientation} + c \cdot \text{previous\_direction}$$

target_direction = alignment of the robot path with the goal
wheel_orientation = difference between the new direction and the currrent wheel orientation
previous_direction = difference between the previously selected direction and the new direction

Borenstein et al.

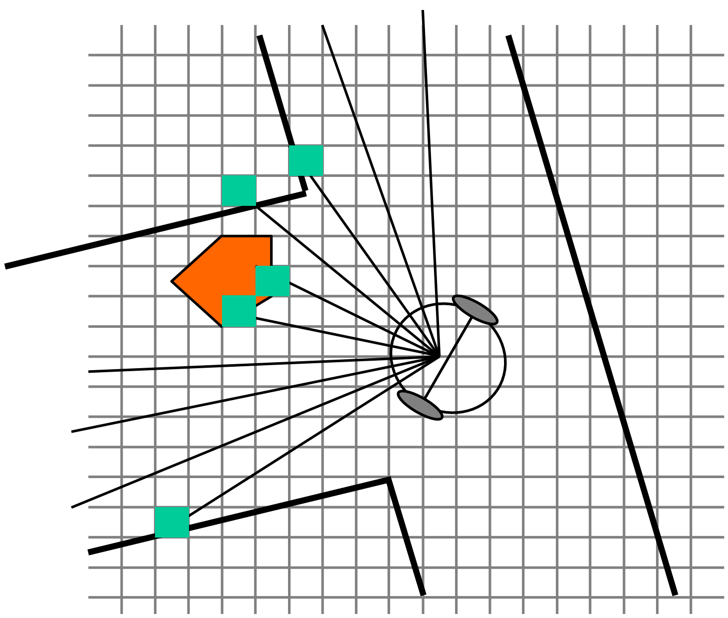


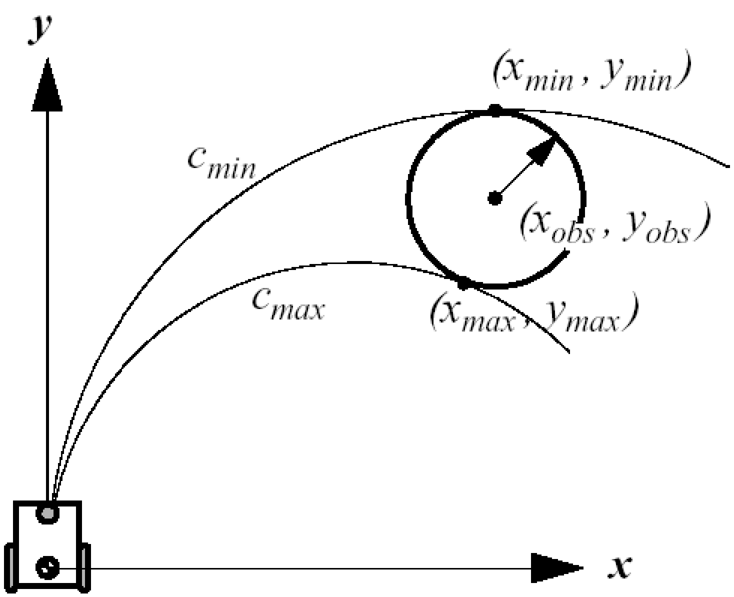


Curvature Velocity Methods (CVM) *[Simmons et al. 1996]*

CVMs add physical constraints from the robot and the environment on (v, w)

- Assumption that robot is traveling on arcs (c= w / v) with acceleration constraints
- Obstacles are transformed in velocity space
- An objective function to select the optimal speed



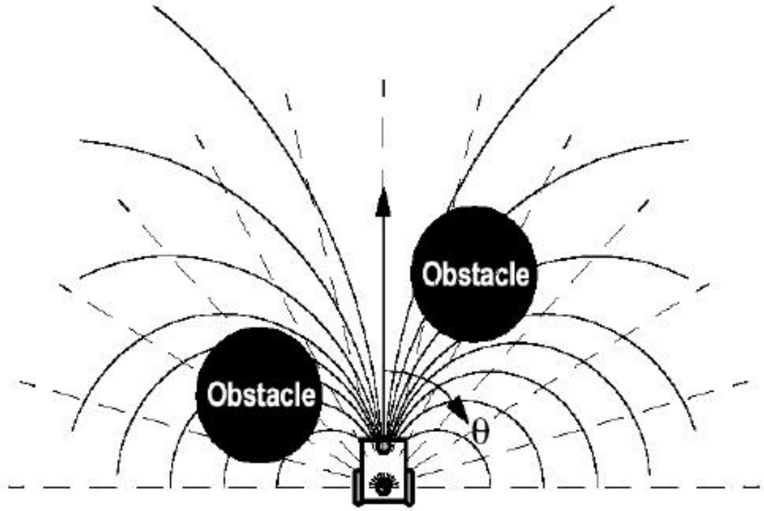


Simmons et al.

Vector Field Histogram+ (VFH+) *[Borenstein et al. 1998]*

VHF+ accounts also in a very simplified way for vehicle kinematics

- robot moving on arcs or straight lines
- obstacles blocking a given direction also blocks all the trajectories (arcs) going through this direction like in an Ackerman vehicle
- obstacles are enlarged so that all kinematically blocked trajectories are properly taken into account



Borenstein et al.

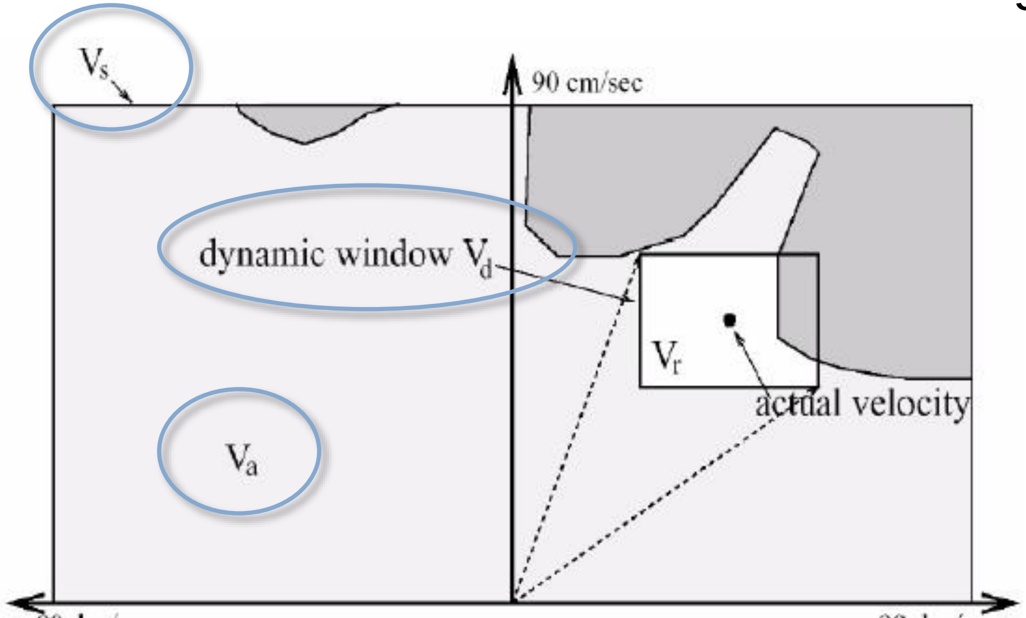
However VHF+ as VHF suffers

- Limitation if narrow areas (e.g. doors) have to be passed
- Local minima might not be avoided
- Reaching of the goal can not be guaranteed
- Dynamics of the robot not really considered

Dynamic Window Approach (DWA) *[Fox et al. 1997]*

The kinematics of the robot are considered via local search in velocity space:

- Consider only circular trajectories determined by pairs $V_s=(v,\omega)$ of translational and rotational speeds
- A pair $V_a=(v,\omega)$ is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.
- A dynamic window restricts the reachable velocities V_d to those that can be reached within a short time given limited robot accelerations



Fox et al.

$$V_d = \begin{cases} v \in [v - a_{tr} \cdot t, v + a_{tr} \cdot t] \\ \omega \in [\omega - a_{rot} \cdot t, \omega + a_{rot} \cdot t] \end{cases}$$

DWA Search Space

$$V_r = V_s \cap V_a \cap V_d$$

Matteo Matteucci – matteo.matteucci@polimi.it

How to choose (v,ω)?

Steering commands are chosen maximizing a heuristic navigation function:

- Minimize the travel time by "driving fast in the right direction"
- Planning restricted to V_r space [Fox, Burgard, Thrun '97]

$$G(v,\omega) = \sigma\left(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot velocity(v,\omega)\right)$$

- *heading*: Alignment with target direction
- *dist*: Distance to closest obstacle intersecting with curvature
- *velocity*: Forward velocity of the robot

- Global approach [Brock & Khatib 99] in <x,y>-space uses

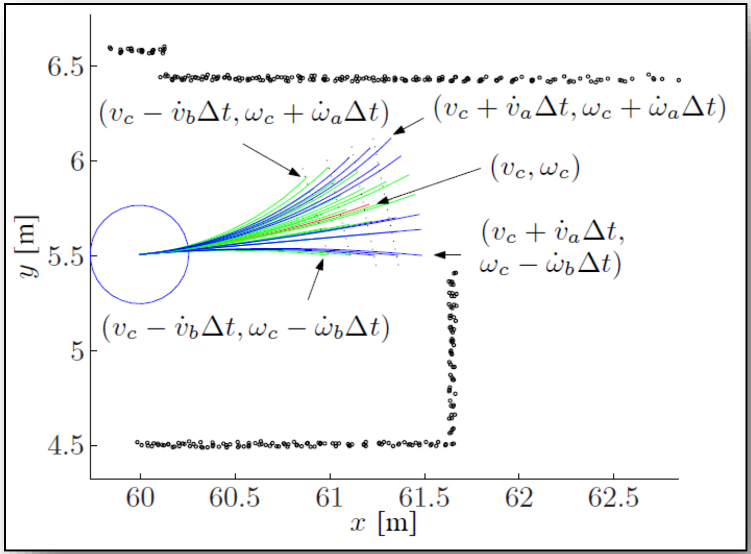
$$NF = \alpha \cdot vel + \beta \cdot nf + \gamma \Delta nf + \delta goal$$

- *vel*: Forward robot velocity
- *nf*: Cost to reach the goal
- Δnf: Follows global path
- *goal*: Goal nearness

DWA Algorithm (as implemented in ROS movebase)

The basic idea of the Dynamic Window Approach (DWA) algorithm follows …

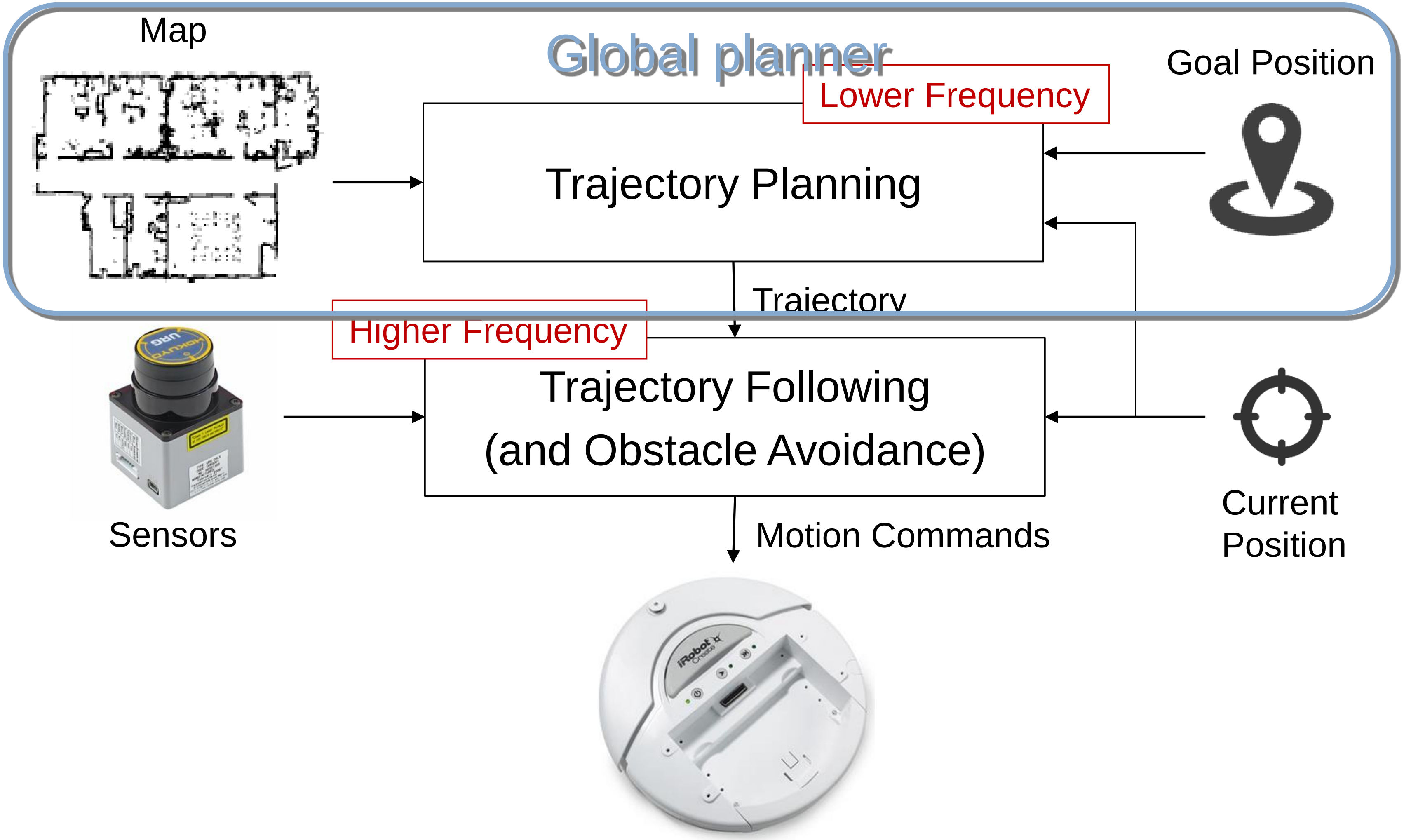
1. Discretely sample robot control space
2. For each sampled velocity, perform forward simulation from current state to predict what would happen if applied for some (short) time.
3. Evaluate (score) each trajectory resulting from the forward simulation
4. Discard illegal trajectories, i.e., those that collide with obstacles, and pick the highest-scoring trajectory



What about non circular kinematics?

Clothoid: $S(x) = \int_0^x \sin(t^2)\,dt, \quad C(x) = \int_0^x \cos(t^2)\,dt.$

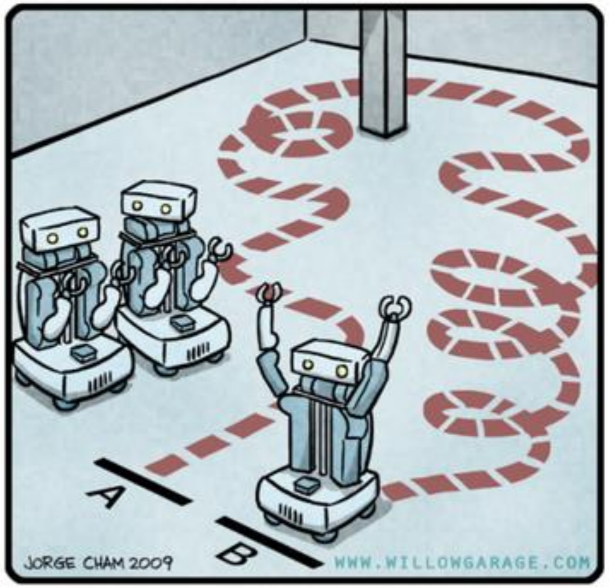
A Two Layered Approach



Matteo Matteucci – matteo.matteucci@polimi.it

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

J.-C. Latombe (1991)



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

Robot Motion Planning Goals

- Collision-free trajectories
- Robot should reach the goal location as fast as possible

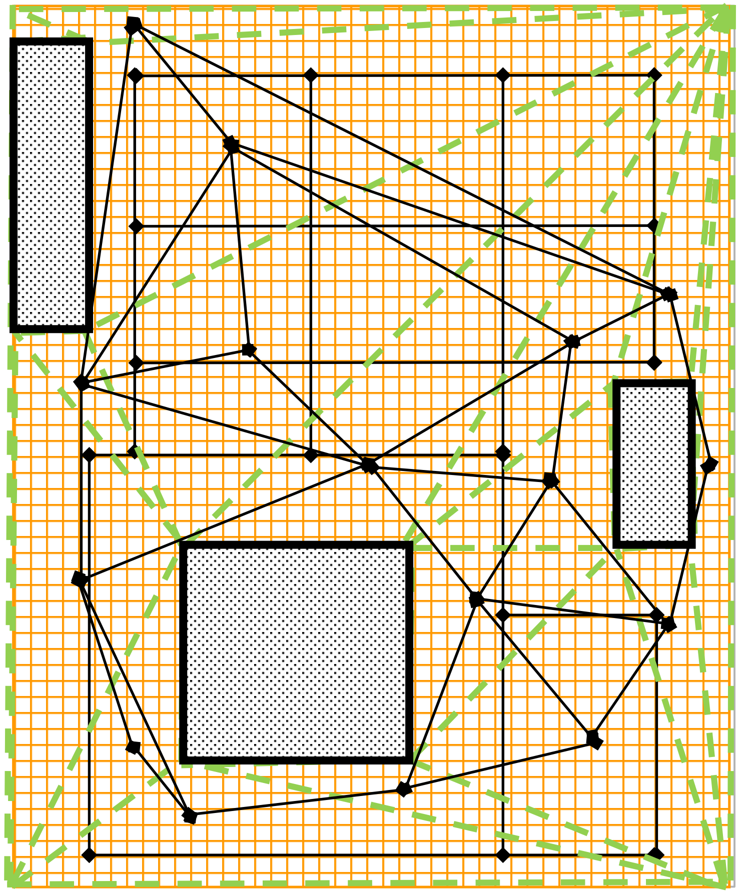
Information available

- Map with obstacles
- Robot shape and kinematics

Planning on grid maps

Different possible maps representations exist for path planning

- Paths (e.g., probabilistic road maps)
- Free space (e.g., Voronoi diagrams)
- Obstacles (e.g., geometric obstacles)
- Composite (e.g., grid maps)



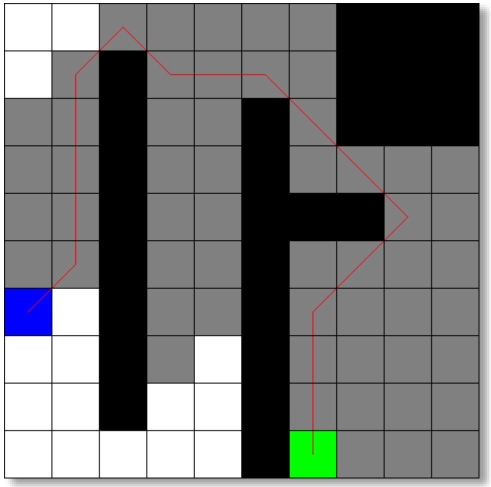
Matteo Matteucci – matteo.matteucci@polimi.it

What a Planner?

Search Based Planning Algorithms

- A*
- ARA*
- ANA*
- *AD**
- D*
- …

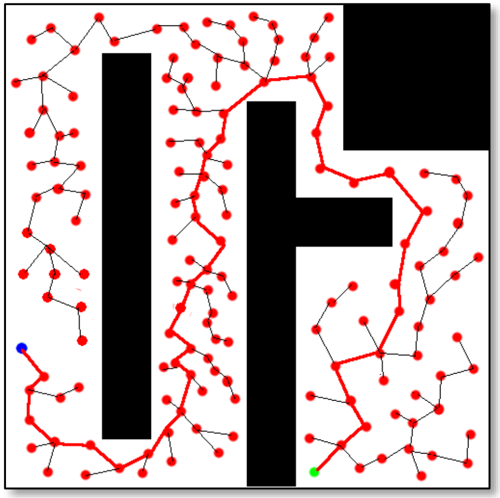
Search Based Planning Library



Random Sampling

- PRMs
- RRT
- T-RRT
- SBL
- …

Open Motion Planning Library



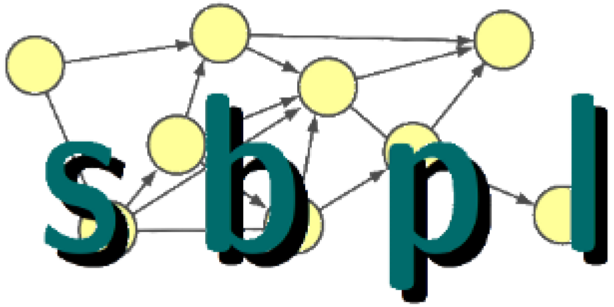
Matteo Matteucci – matteo.matteucci@polimi.it

Pros and Cons

	PROS	CONS
Search Based Planning	• Finds the optimal solution • Possible to assign costs • Use of Heuristics • Can state if a solution exists (complete)	• High computational cost
Random Sampling Planning	• Fast in finding a feasible solution	• Hard to assign costs • Only probably complete (cannot be used to test for existance)

Lets have a look at Search Based Methods (SBPL) because of

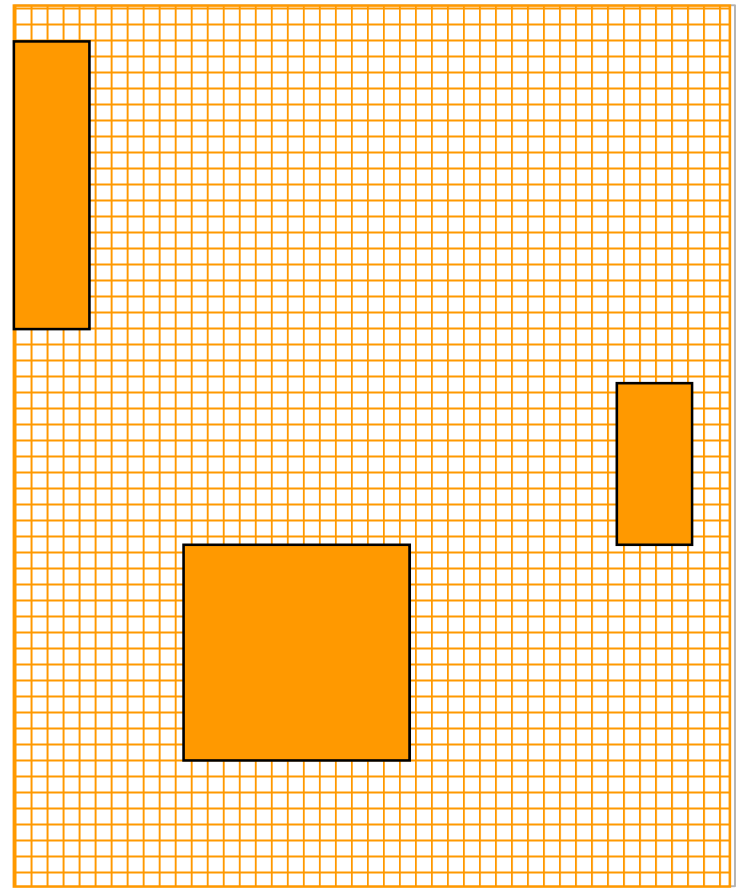
- Their simplicity (at least in description)
- The generality of approaches
- Their theoretical guarantees
  (if connectivity assumptions hold)



Planning on grid maps

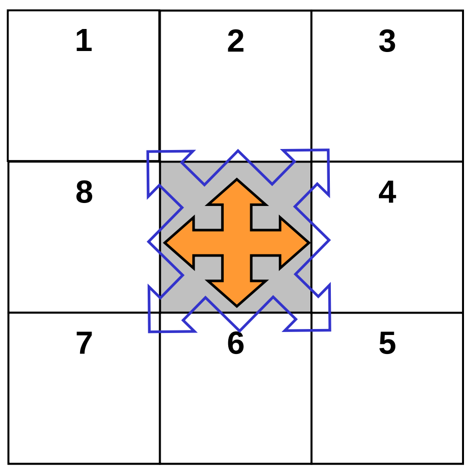
Different possible maps representations exist for path planning

- Paths (e.g., probabilistic road maps)
- Free space (e.g., Voronoi diagrams)
- Obstacles (e.g., geometric obstacles)
- Composite (e.g., grid maps)



Kinematics approximation in grid maps

- 4-orthogonal connectivity
- 4-diagonal connectivity
- 8-connectivity

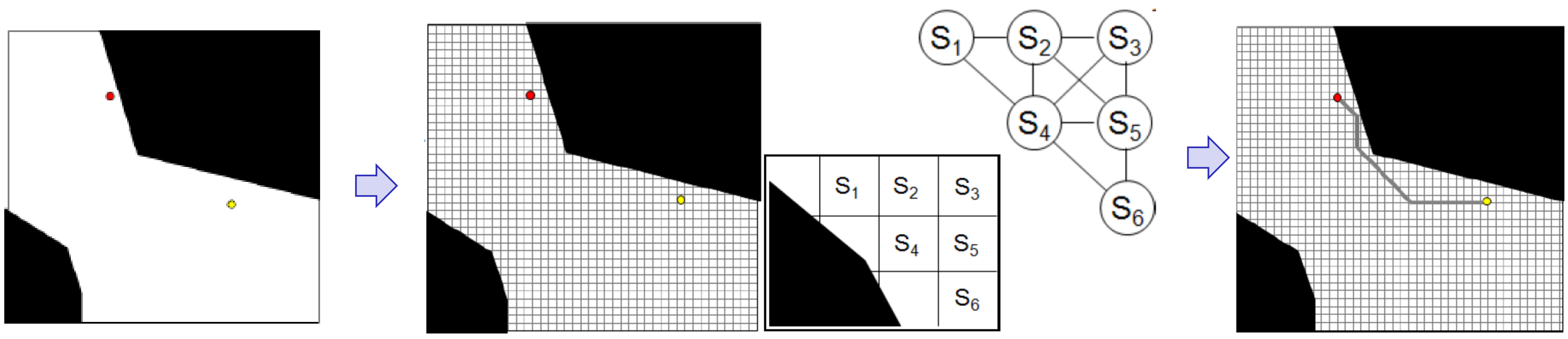


Matteo Matteucci – matteo.matteucci@polimi.it

Graph (Search) Based Planning Basics

The overall idea:

- Generate a discretized representation of the planning problem
- Build a graph out of this discretized representation (e.g., through 4 neighbors or 8 neighbors connectivity)
- Search the graph for the optimal solution

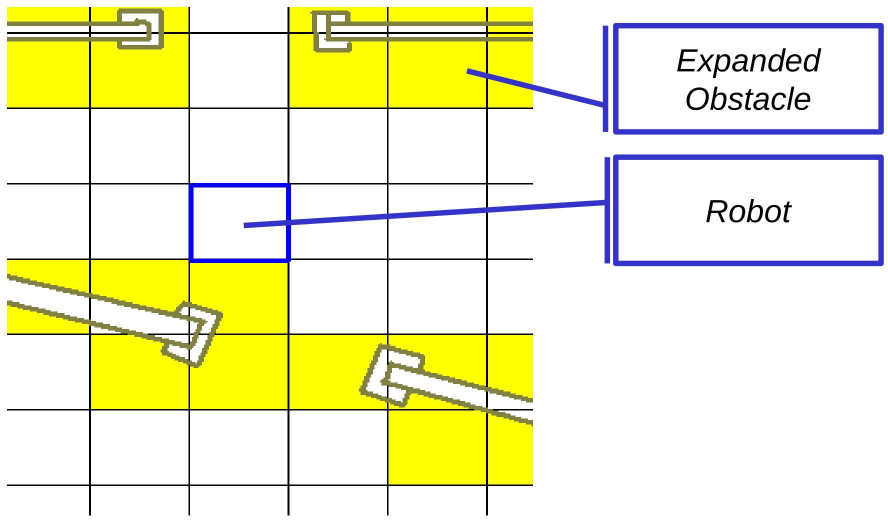


- Can interleave the construction of the representation with the search (i.e., construct only what is necessary)

Robot shape

A real mobile robot should not be modeled as a point; to take into account its shape obstacles are enlarged

This might generate some issues and a trade-off is between memory requirements and performance

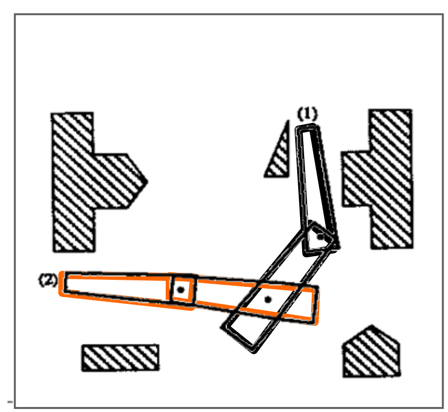


Configuration Space (C-Space)

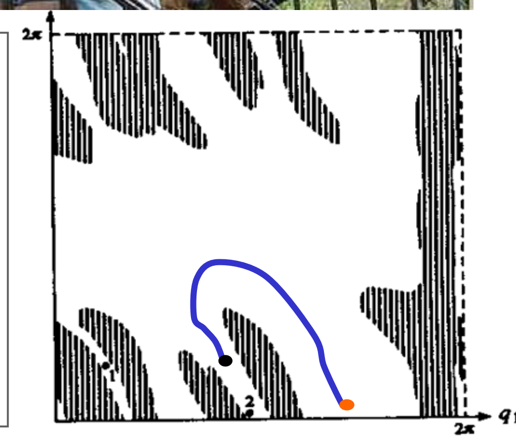
For non circular robots, collisions might depend on the orientation.

The C-Space is used to speed up collision detection

- A configuration of an object is a point q = (q1, q2,…,qn)
- Point q is free if the robot in q does not collide
- C-obstacle = union of all q where the robot collides
- C-free = union of all free q
- Cspace = C-free + C-obstacle



workspace



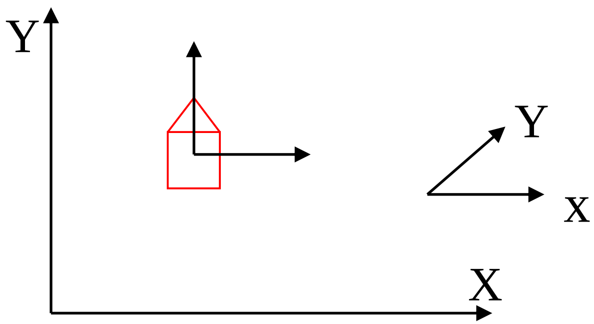
configuration space

Planning can be performed in C-Space

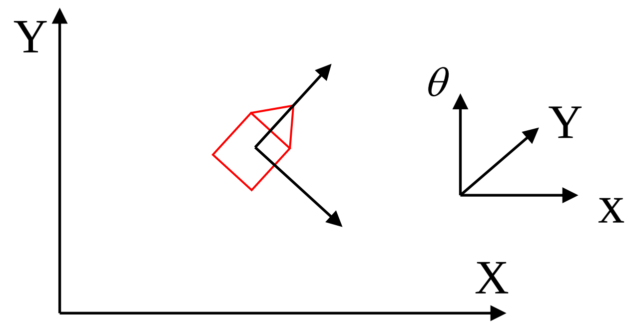
Matteo Matteucci – matteo.matteucci@polimi.it

Mobile robots C-Space

A robot can translate in the plane and/or rotate

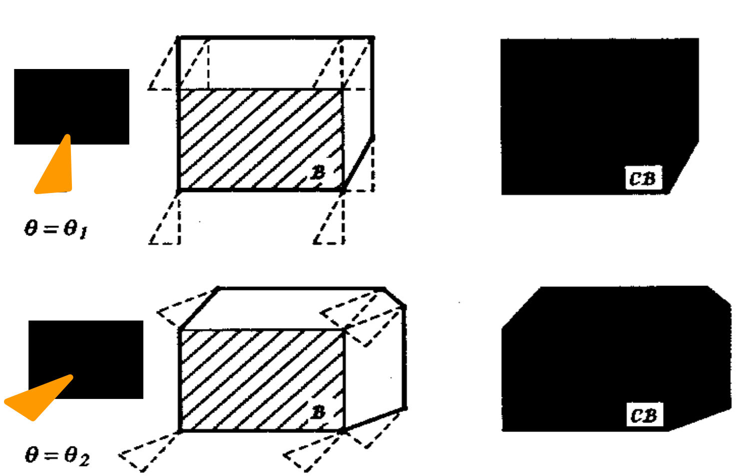


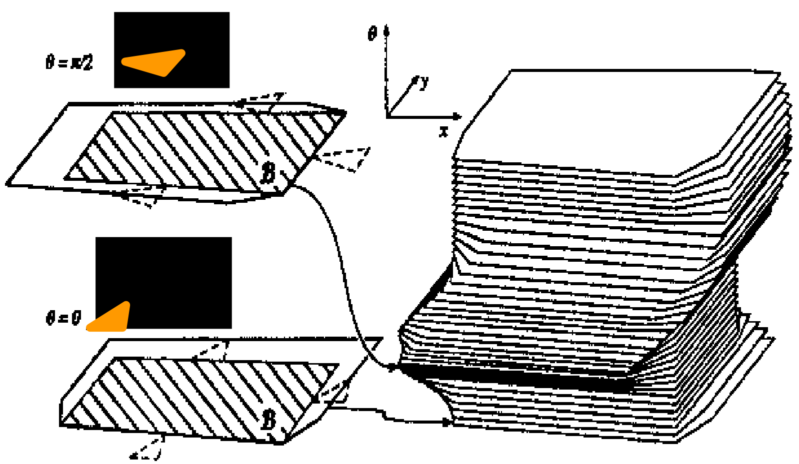
$\mathcal{C}$-space: 2D (x, y)



$\mathcal{C}$-space: 3D (x, y, θ)

Obstacles should be expanded according to the robot orientation



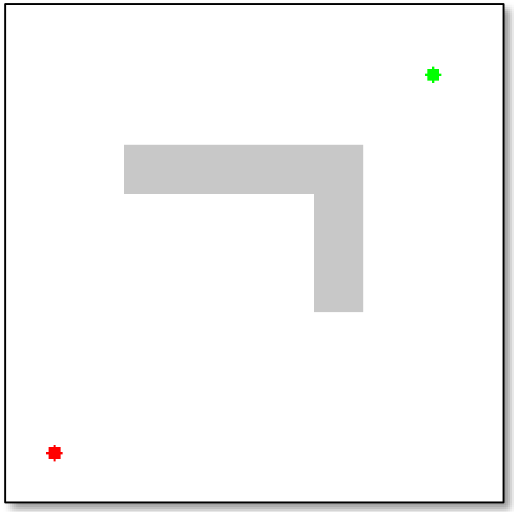


Matteo Matteucci – matteo.matteucci@polimi.it

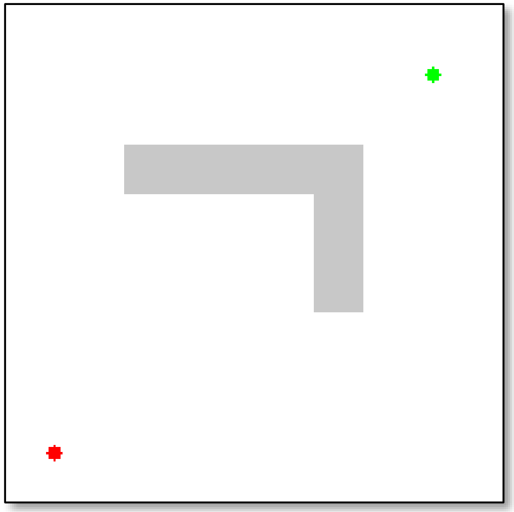
Exact and Approximate Planning (in SBPL)

Different algorithms are available

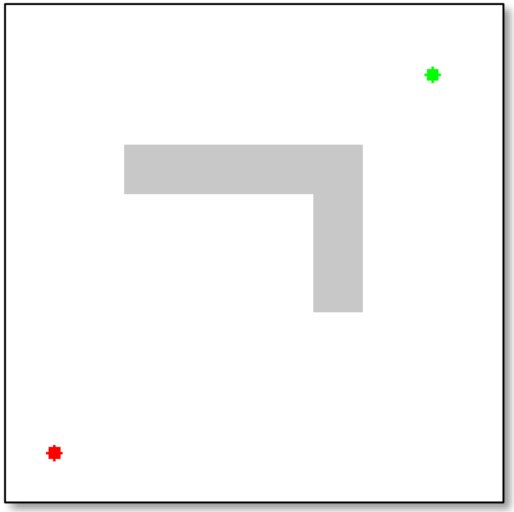
- Returning the optimal path (e.g., Dijstra, A*, …)
- Returning an ε sub-optimal path
  (e.g., weighted A*, ARA*, AD*, R*, D* Lite, ...)



Dijstra



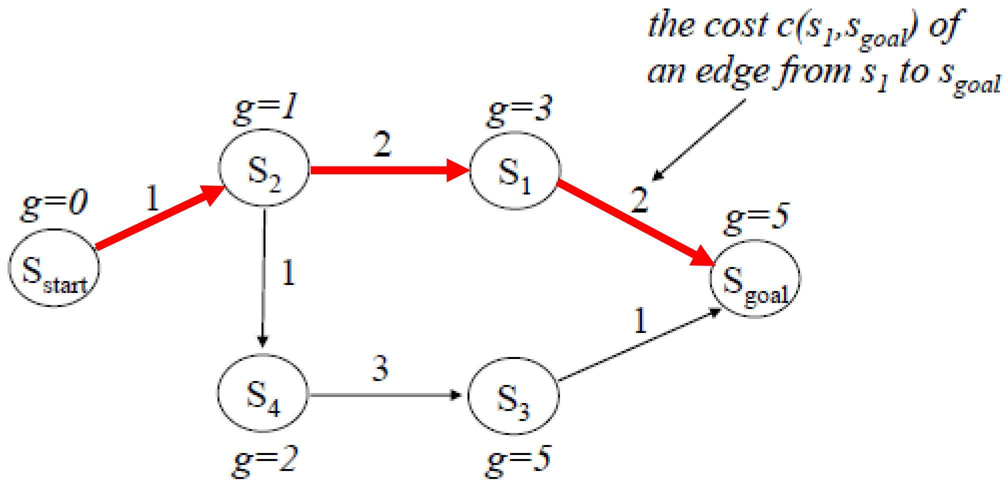
A*



weighted A*

Searching Graphs for Least Cost Path

Given a graph search for the path that minimizes costs as much as possible



Many search algorithms compute optimal <u>g-values</u> for relevant states

- $g(s)$–an estimate of the cost of a least-cost path from s_{start} to s
- optimal values satisfy: $g(s) = \min_{s'' \text{ in } pred(s)} g(s'') + c(s'',s)$

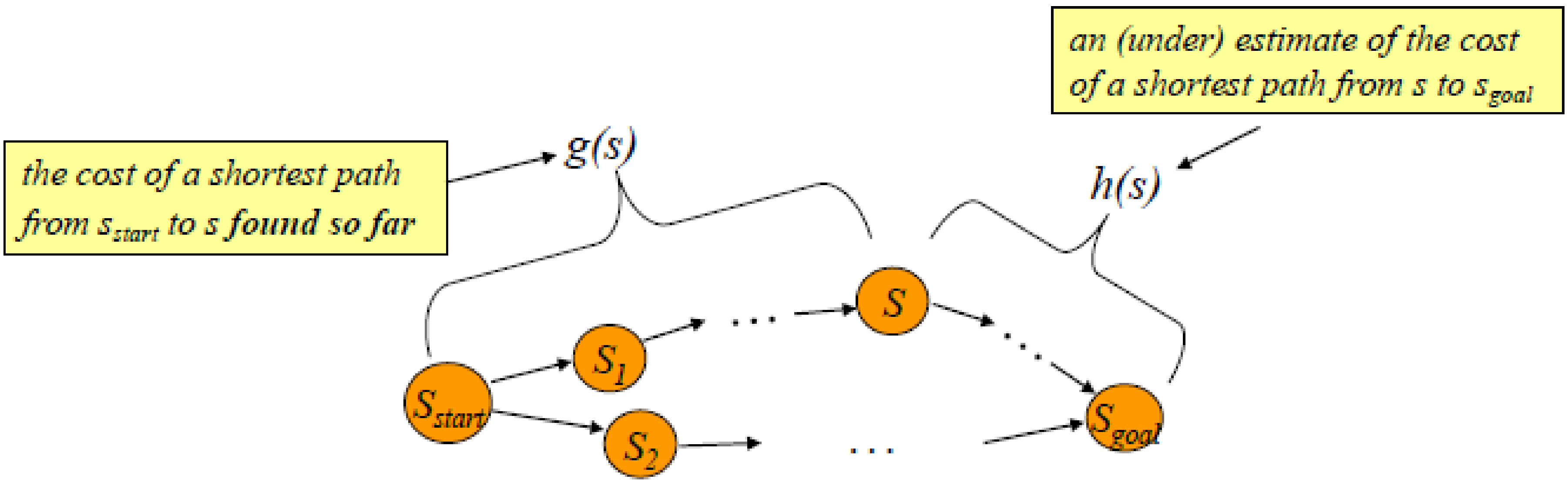
Least-cost path is a greedy path computed by backtracking:

- start with s_{goal} and from any state s move to the predecessor state s' such that

$$s' = \text{argmin}_{s'' \text{ in pred}(s)} (g(s'')+c(s'',s))$$

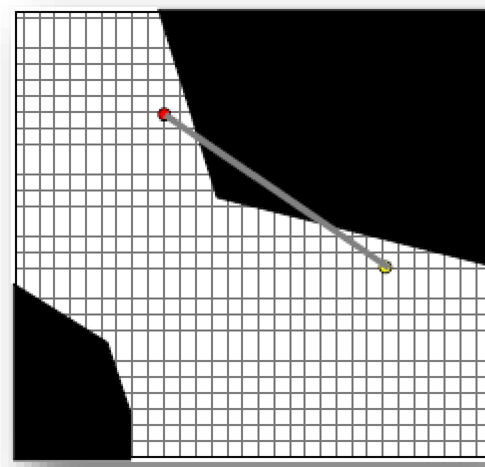
A* Search

A* speeds up search by computing g-values for relevant states as



Heuristic function must be

- admissible: for every state s, $h(s) \leq c^*(s,s_{goal})$
- consistent (satisfy triangle inequality):
  - $h(s_{goal},s_{goal}) = 0$
  - for every $s \neq s_{goal}$, $h(s) \leq c(s,succ(s)) + h(succ(s))$



Admissibility follows from consistency and often consistency follows from admissibility

Matteo Matteucci – matteo.matteucci@polimi.it

A* Search Algorithm

Main function

- $g(s_{start}) = 0;$ all other *g*-values are infinite*;*
- $OPEN = \{s_{start}\};$
- ComputePath();

Set of candidates for expansion

ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  - expand *s*;

For every expanded state g(s) is optimal (if heuristics are consistent)

Matteo Matteucci – matteo.matteucci@polimi.it

A* Search Algorithm

Main function

- $g(s_{start}) = 0$; all other *g*-values are infinite*;*
- $OPEN = \{s_{start}\}$;
- ComputePath();

Set of candidates for expansion

ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest $[f(s) = g(s)+h(s)]$from *OPEN*;
  - insert *s* into *CLOSED*;
  - for every successor *s'*of s_{such} that *s'* not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$;
      - insert *s'* into *OPEN*;

Set of states already expanded

Tries to decrease g(s') using the found path from s_{start} to s

ComputePath function

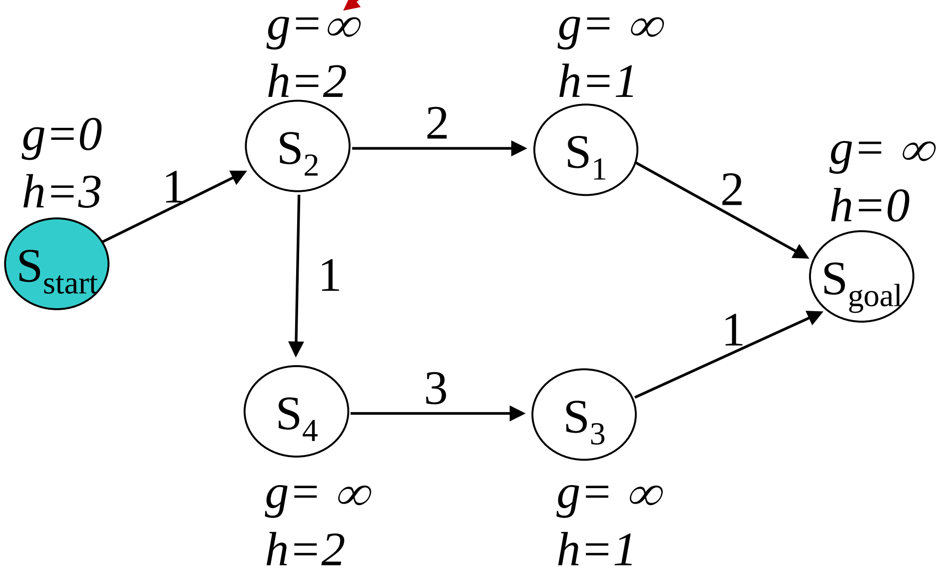
- while(s_{goal} is not expanded)
  - remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  - insert s into *CLOSED*;
  - for every successor s' of s such that s' not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$;
      - insert s' into *OPEN*;

$g(s_2) > g(s_{start}) + c(s_{start}, s2)$

CLOSED = {}

OPEN = $\{s_{start}\}$

next state to expand: s_{start}



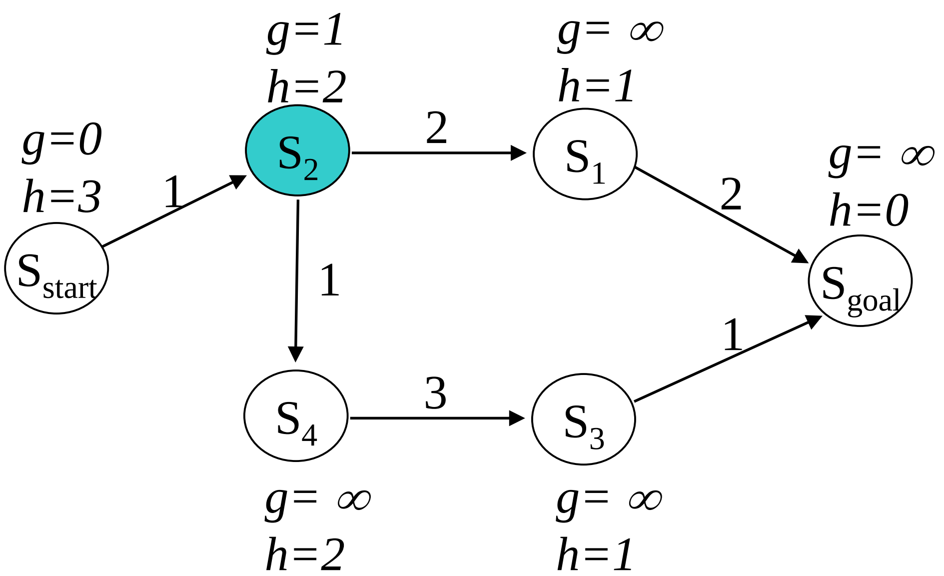
ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]*from *OPEN*;
  - insert *s* into *CLOSED*;
  - for every successor *s'*of *s* such that *s'* not in *CLOSED*
    - if *g(s') > g(s) + c(s,s')*
      - *g(s') = g(s) + c(s,s');*
      - insert *s'* into *OPEN*;

CLOSED = {s_{start}}

OPEN = {s_2}

next state to expand: s_2



A* Search Algorithm

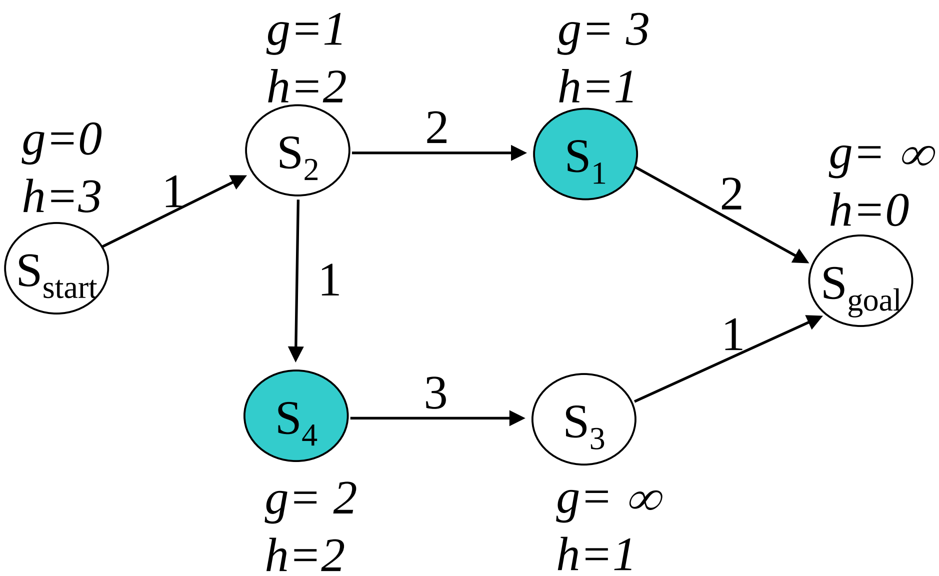
ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]* from *OPEN*;
  - insert *s* into *CLOSED*;
  - for every successor *s'* of *s* such that *s'* not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$;
      - insert *s'* into *OPEN*;

CLOSED = $\{s_{start}, s_2\}$

OPEN = $\{s_1, s_4\}$

next state to expand: s_1



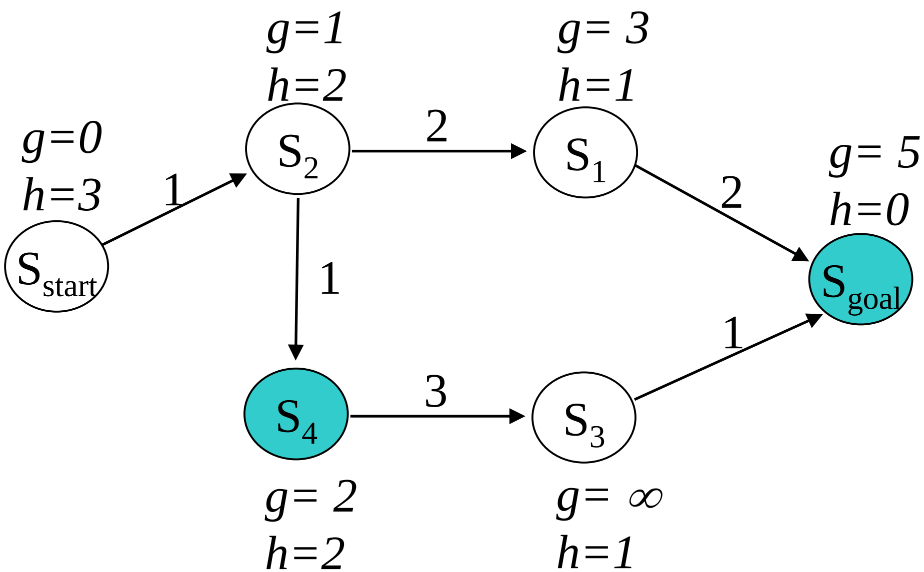
ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]* from *OPEN*;
  - insert *s* into *CLOSED*;
  - for every successor *s'* of *s* such that *s'* not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$;
      - insert *s'* into *OPEN*;

CLOSED = $\{s_{start}, s_2, s_1\}$

OPEN = $\{s_4, s_{goal}\}$

next state to expand: s_4



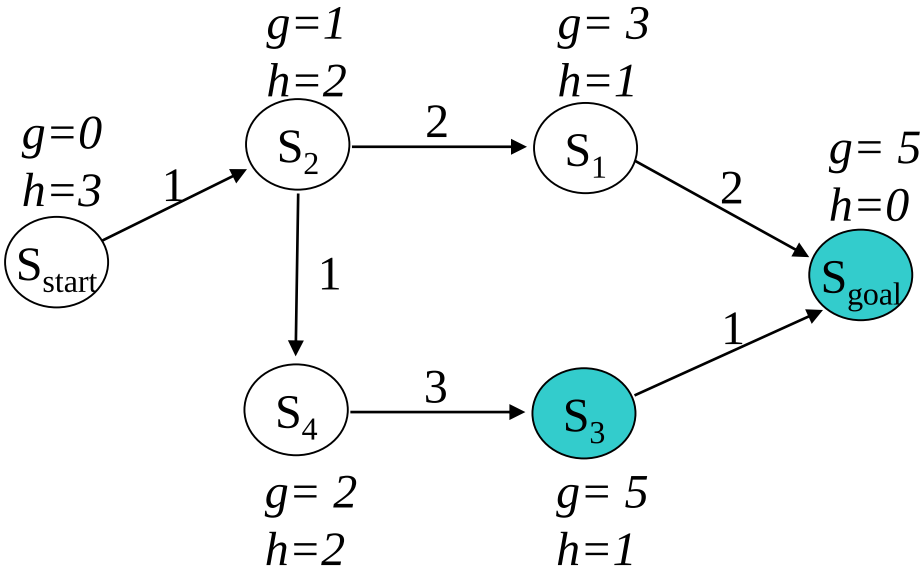
ComputePath function

- while(s_{goal} is not expanded)
  - remove *s* with the smallest *[f(s) = g(s)+h(s)]*from *OPEN*;
  - insert *s* into *CLOSED*;
  - for every successor *s'* of *s* such that *s'* not in *CLOSED*
    - if *g(s') > g(s) + c(s,s')*
      - *g(s') = g(s) + c(s,s');*
      - insert *s'* into *OPEN*;

CLOSED = $\{s_{start}, s_2, s_1, s_4\}$

OPEN = $\{s_3, s_{goal}\}$

next state to expand: s_{goal}



A* Search Algorithm

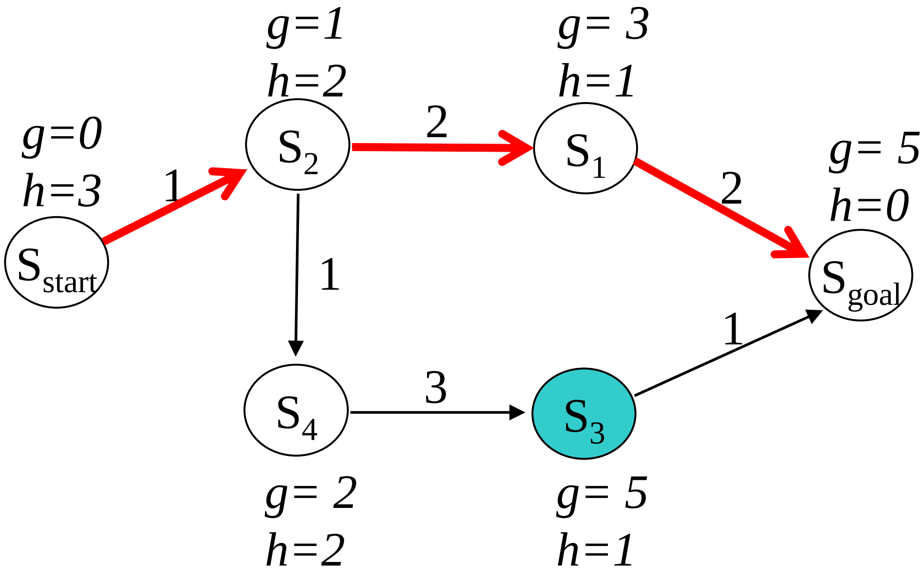
ComputePath function

- while(s_{goal} is not expanded)
  - remove s with the smallest $[f(s) = g(s)+h(s)]$ from *OPEN*;
  - insert s into *CLOSED*;
  - for every successor s' of s such that s' not in *CLOSED*
    - if $g(s') > g(s) + c(s,s')$
      - $g(s') = g(s) + c(s,s')$;
      - insert s' into *OPEN*;

CLOSED = $\{s_{start}, s_2, s_1, s_4, s_{goal}\}$

OPEN = $\{s_3\}$

<u>*DONE!*</u>



A* Properties

A* is guaranteed to

- return an optimal path in terms of the solution
- perform provably minimal number of state expansions

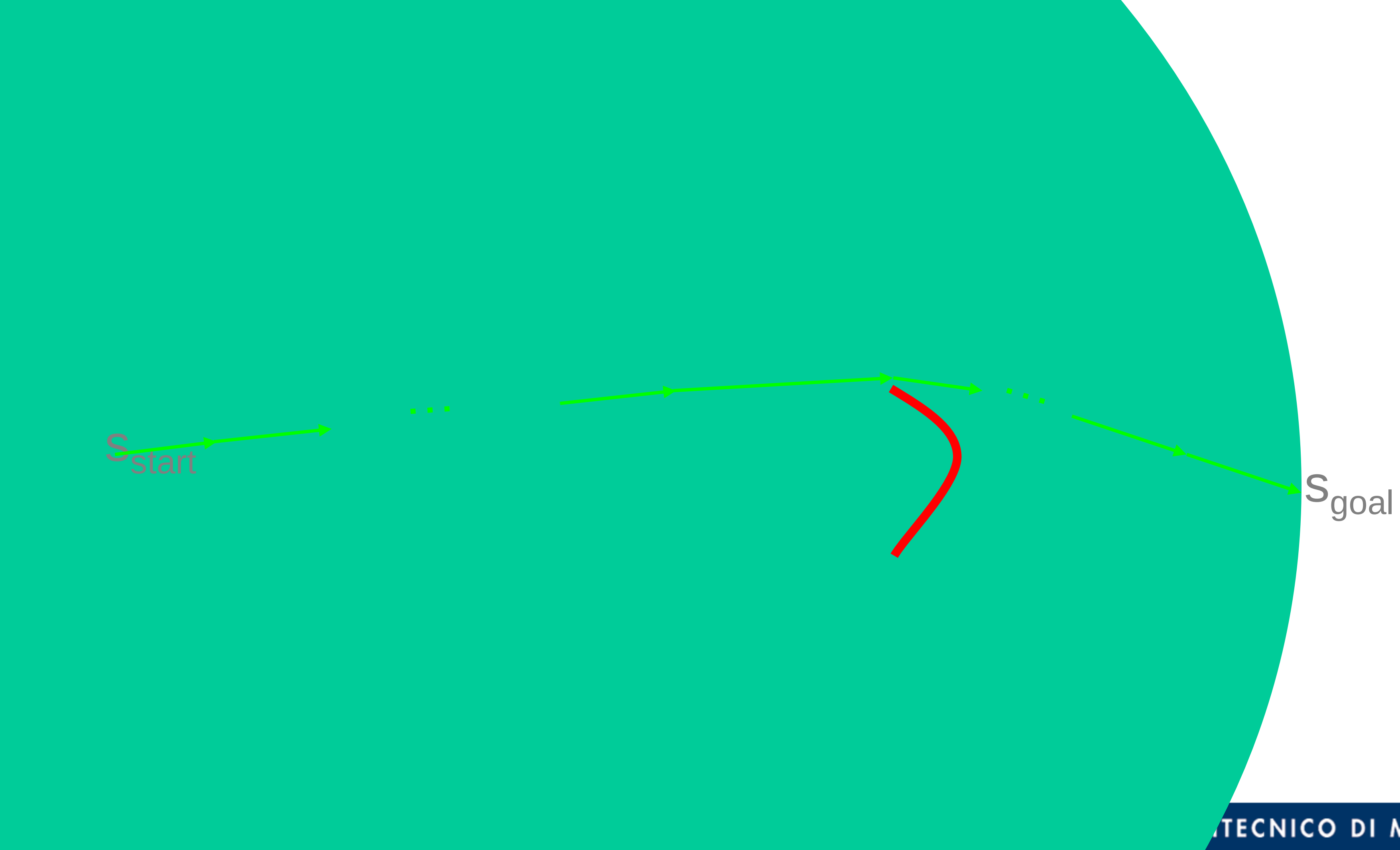
Algorithms state expansion:

- Dijkstra's: expands states in the order of $f = g$ values (roughly)
- A* Search: expands states in the order of $f = g + h$ values
- Weighted A***:**expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$= bias towards states that are closer to goal

Weighted A* Search in many domains, it has been shown to be orders of magnitude faster than A*

Algorithms state expansion:

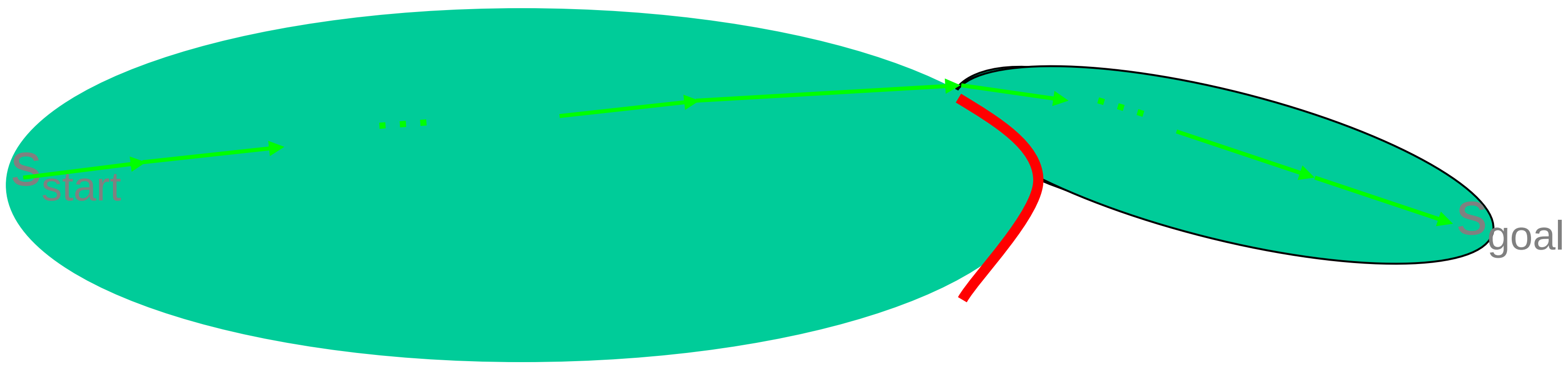
- Dijkstra's: expands states in the order of $f = g$ values (roughly)



A* Properties

Algorithms state expansion:

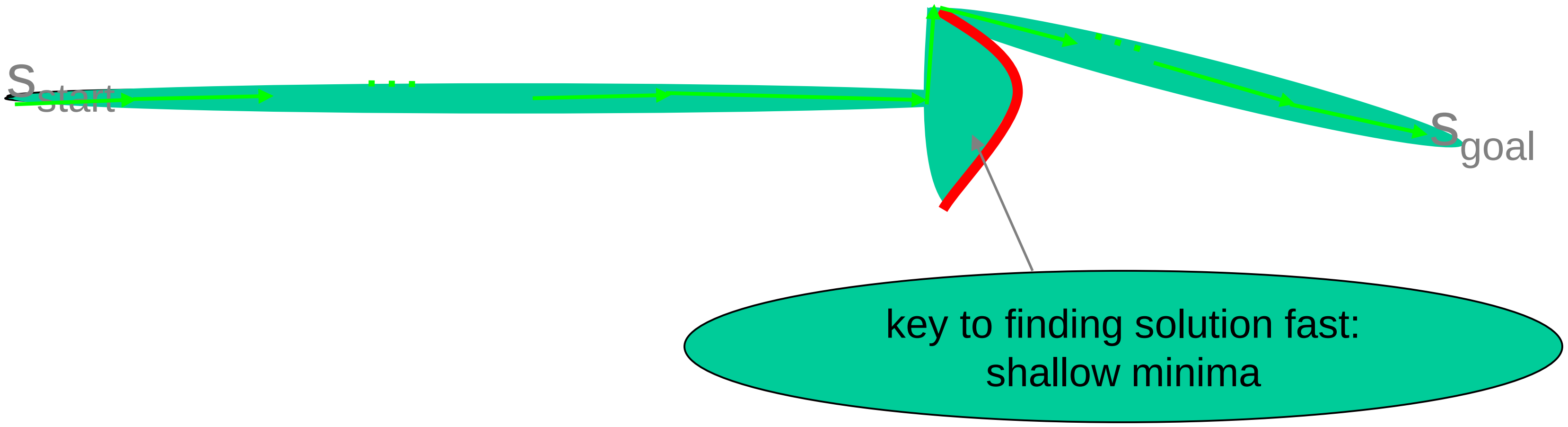
- Dijkstra's: expands states in the order of $f = g$ values (roughly)
- A* Search: expands states in the order of $f = g + h$ values



A* Properties

Algorithms state expansion:

- Dijkstra's: expands states in the order of $f = g$ values (roughly)
- A* Search: expands states in the order of $f = g + h$ values
- Weighted A***:**expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$= bias towards states that are closer to goal



Planning Problem Ingredients

Typical components of a Search-based Planner

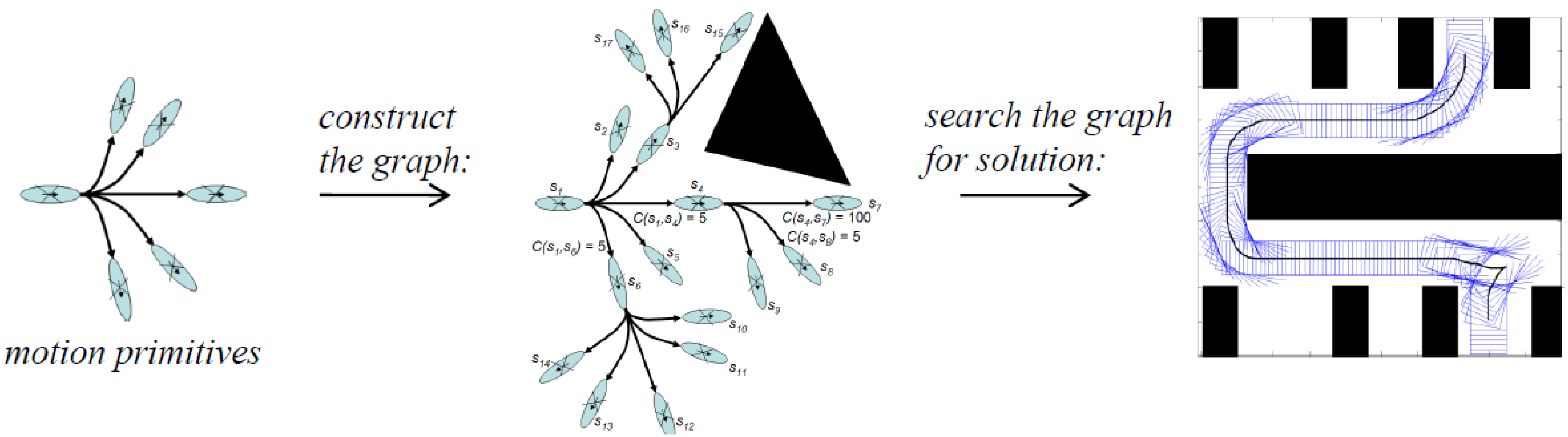
Domain Independent

- Graph search algorithm (for example, A* search)

- Graph construction (given a state what are its successor states)
- Cost function (a cost associated with every transition in the graph)
- Heuristic function (estimates of cost-to-goal)

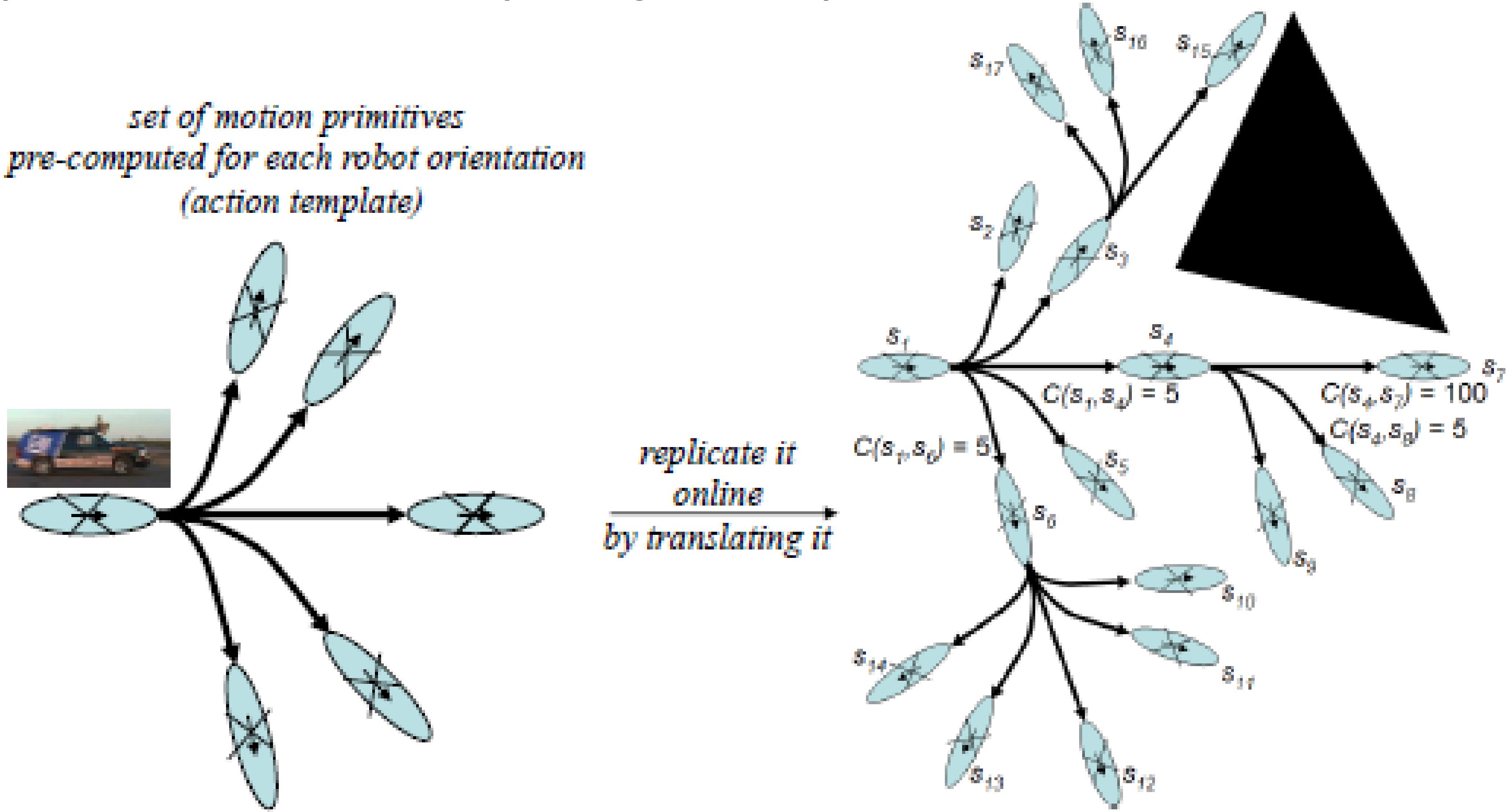
Domain Dependent

The graph can take into account robot dynamics/kinematics constraints



Lattice Based Graphs for Navigation

Graph can be constructed by using motion primitives



- Pros: sparse graph, feasible path, incorporate a variety of constraints
- Cons: possible incompleteness

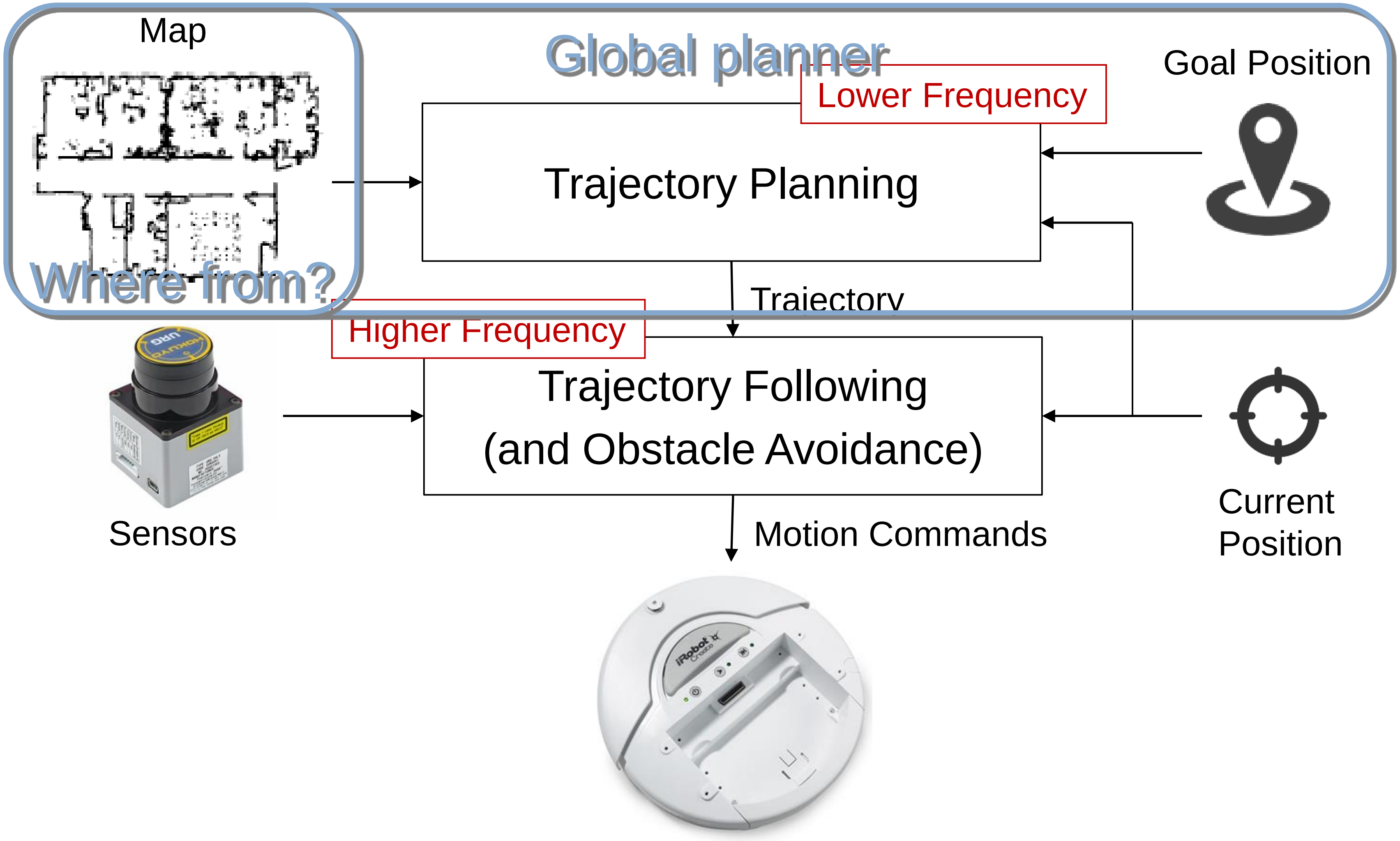
Lattice Based Graphs for Navigation

Graph can be constructed by using motion primitives

- Pros: sparse graph, feasible path, incorporate a variety of constraints
- Cons: possible incompleteness

*planning on 4D (<x,y,orientation,velocity>) multi-resolution lattice using Anytime D**
[Likhachev & Ferguson, '09]

A Two Layered Approach



Matteo Matteucci – matteo.matteucci@polimi.it

POLITECNICO DI MILANO

Cognitive Robotics – Robot Motion Planning

Matteo Matteucci – matteo.matteucci@polimi.it