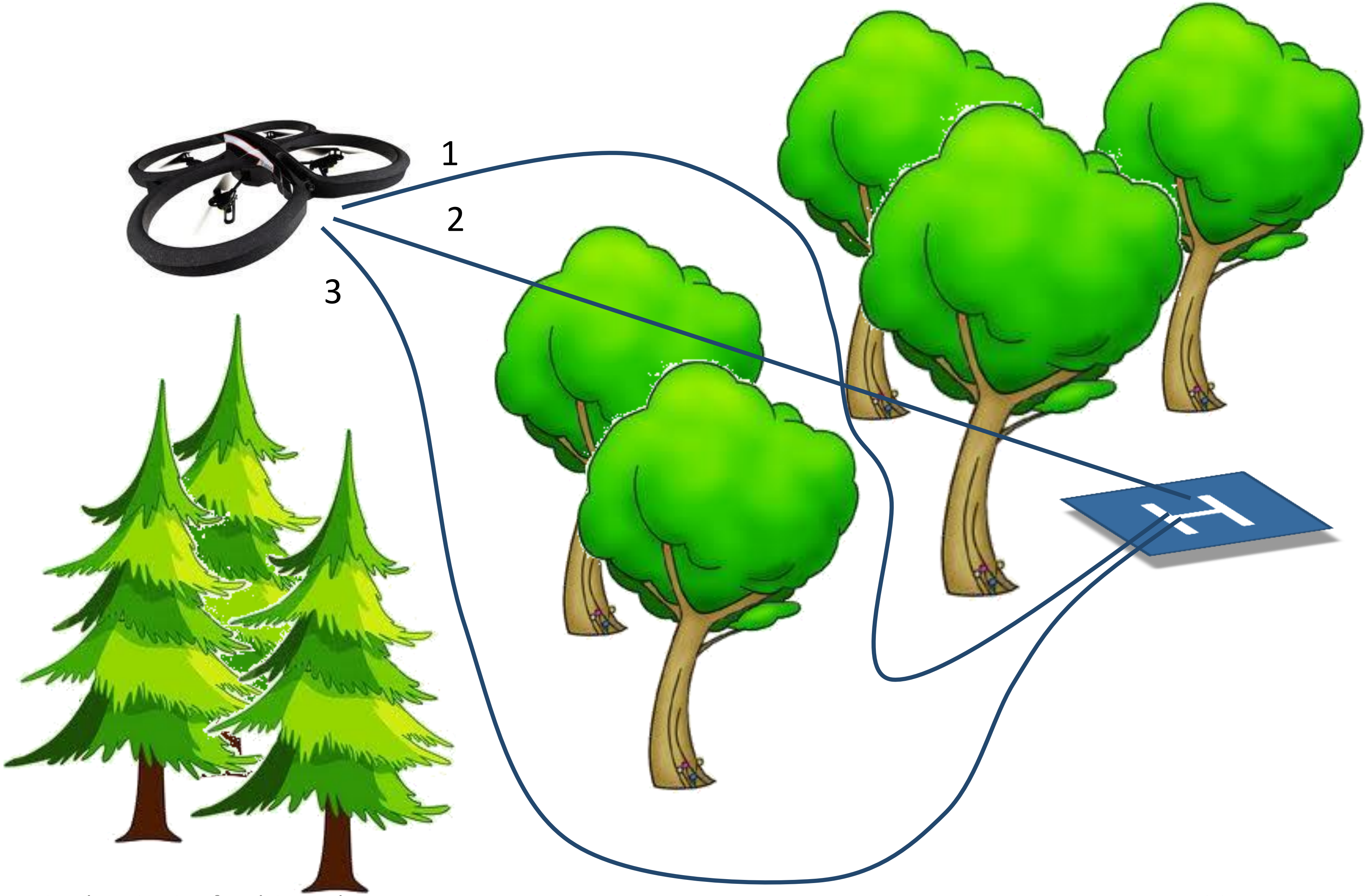
Visual Navigation for Flying Robots

Motion Planning

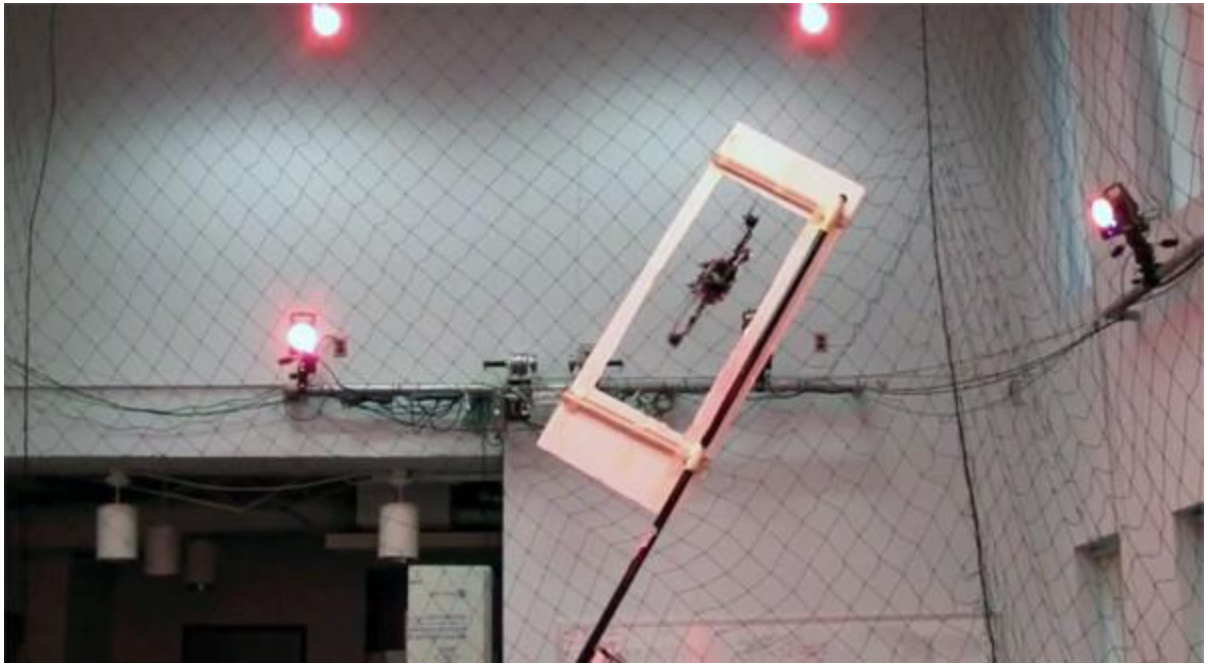
Dr. Jürgen Sturm

Motivation: Flying Through Forests



Motion Planning Problem

- Given obstacles, a robot, and its motion capabilities, compute collision-free robot motions from the start to goal.



Motion Planning Problem

What are good performance metrics?

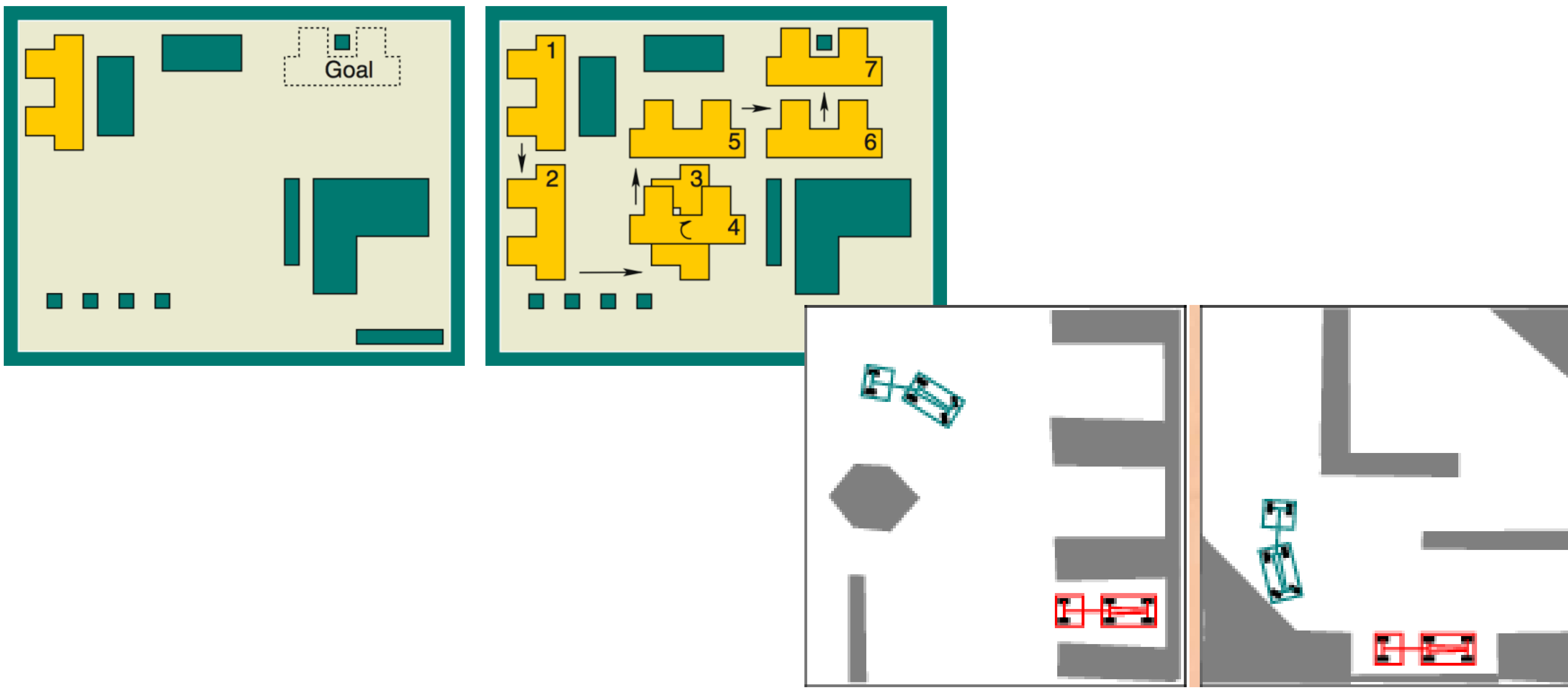
Motion Planning Problem

What are good performance metrics?

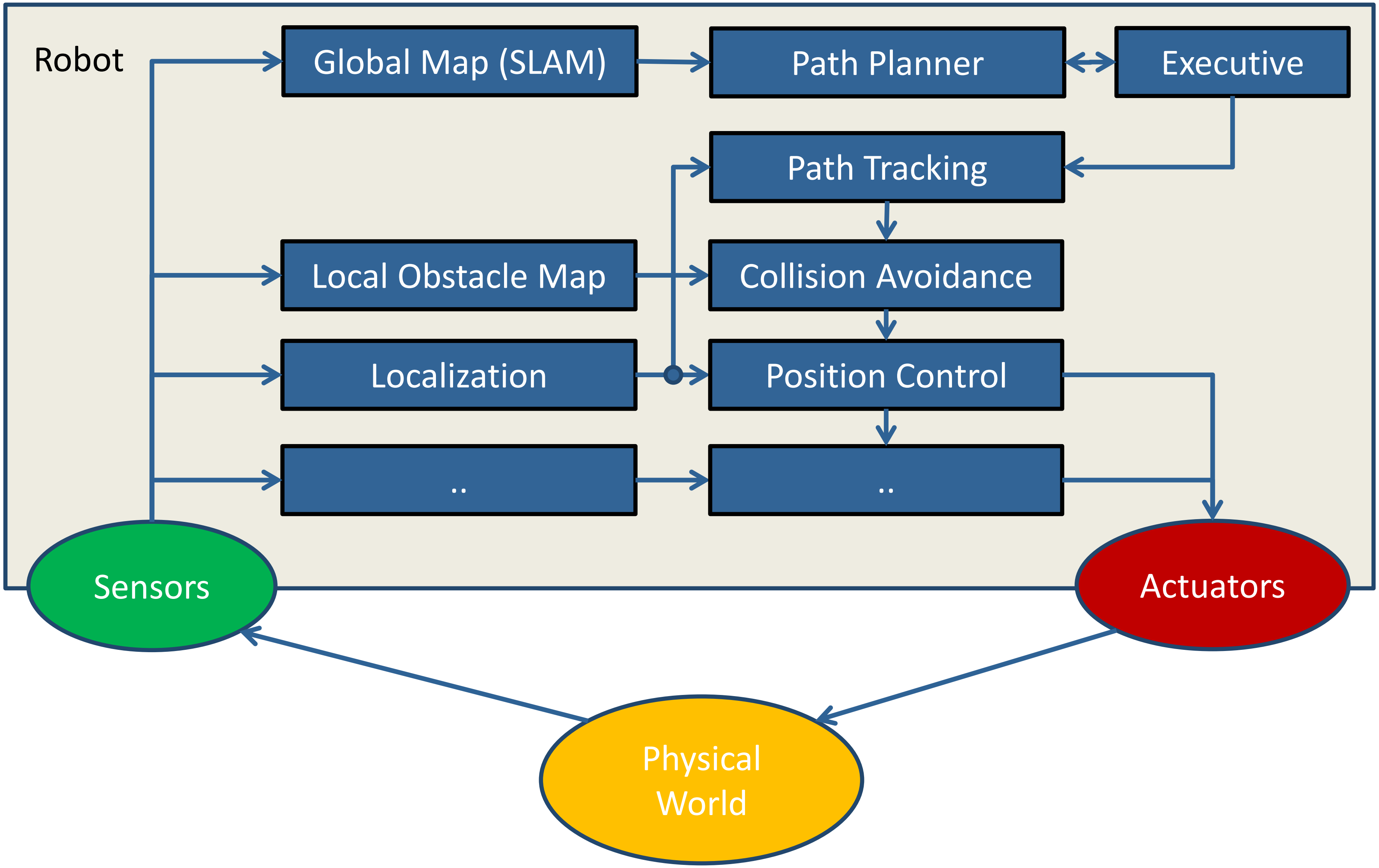
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- …

Motion Planning Examples

Motion planning is sometimes also called the **piano mover's problem**



Robot Architecture



Agenda for Today

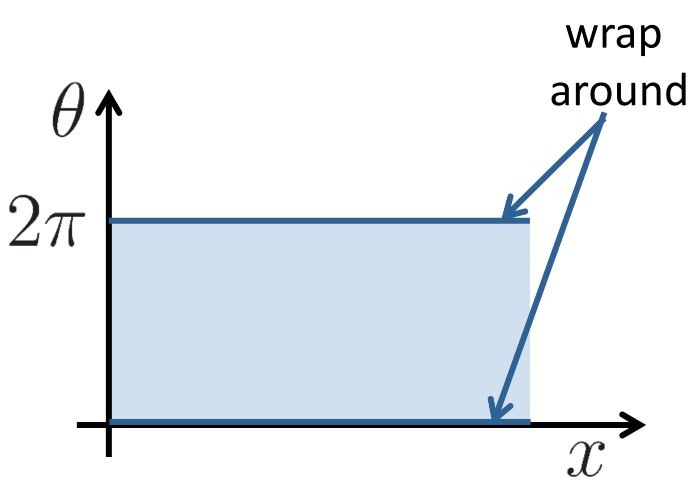
- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

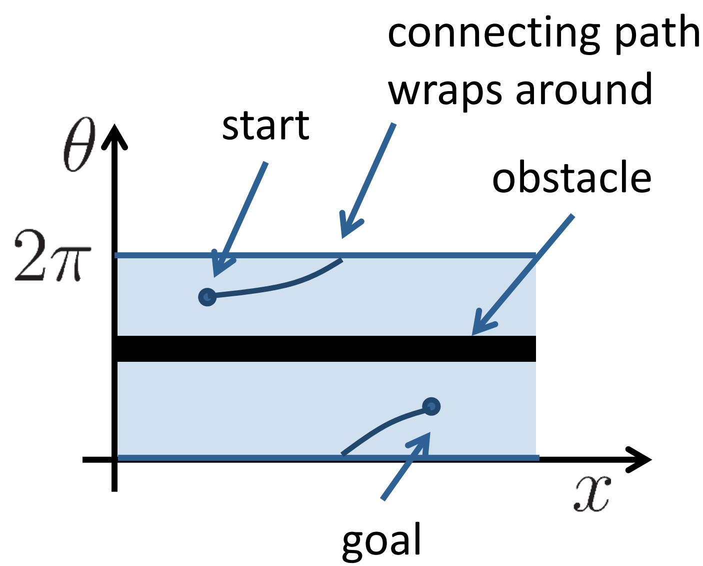
Configuration Space

- Work space
  - Position in 3D → 3 DOF
- Configuration space
  - Reduced pose (position + yaw) → 4 DOF
  - Full pose → 6 DOF
  - Pose + velocity → 12 DOF
  - Joint angles of manipulation robot
  - …
- Planning takes place in **configuration space**

Configuration Space

- The configuration space (C-space) is the **space of all possible configurations**
- C-space topology is usually not Cartesian
- C-space is described as a topological manifold





Notation

- Configuration space $C \subset \mathbb{R}^d$
- Configuration $\mathbf{q} \in C$
- Free space C_{free}
- Obstacle space C_{obs}

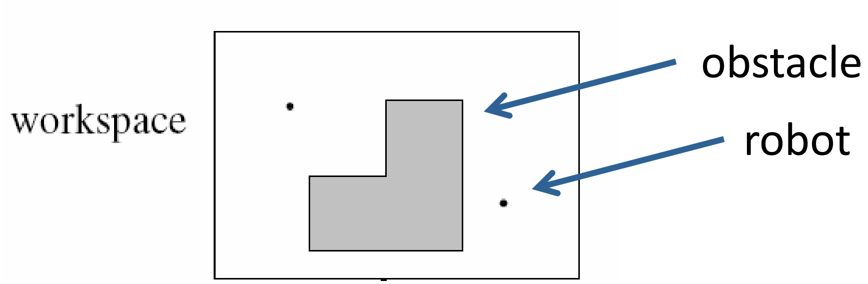
- Properties

$$C_{\text{free}} \cup C_{\text{obs}} = C$$

$$C_{\text{free}} \cap C_{\text{obs}} = \emptyset$$

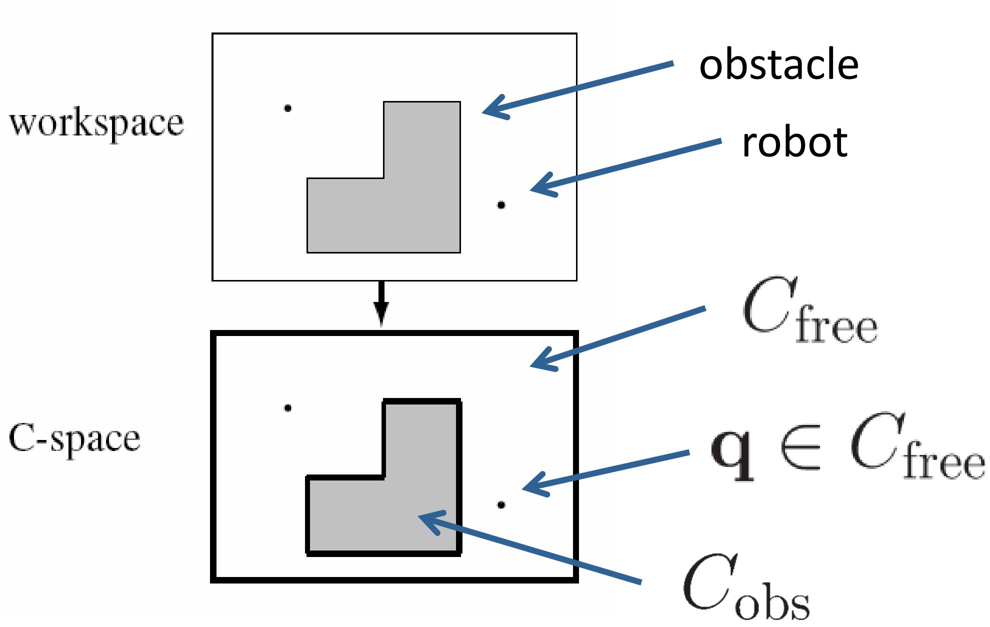
Free Space Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
- “Point” robot



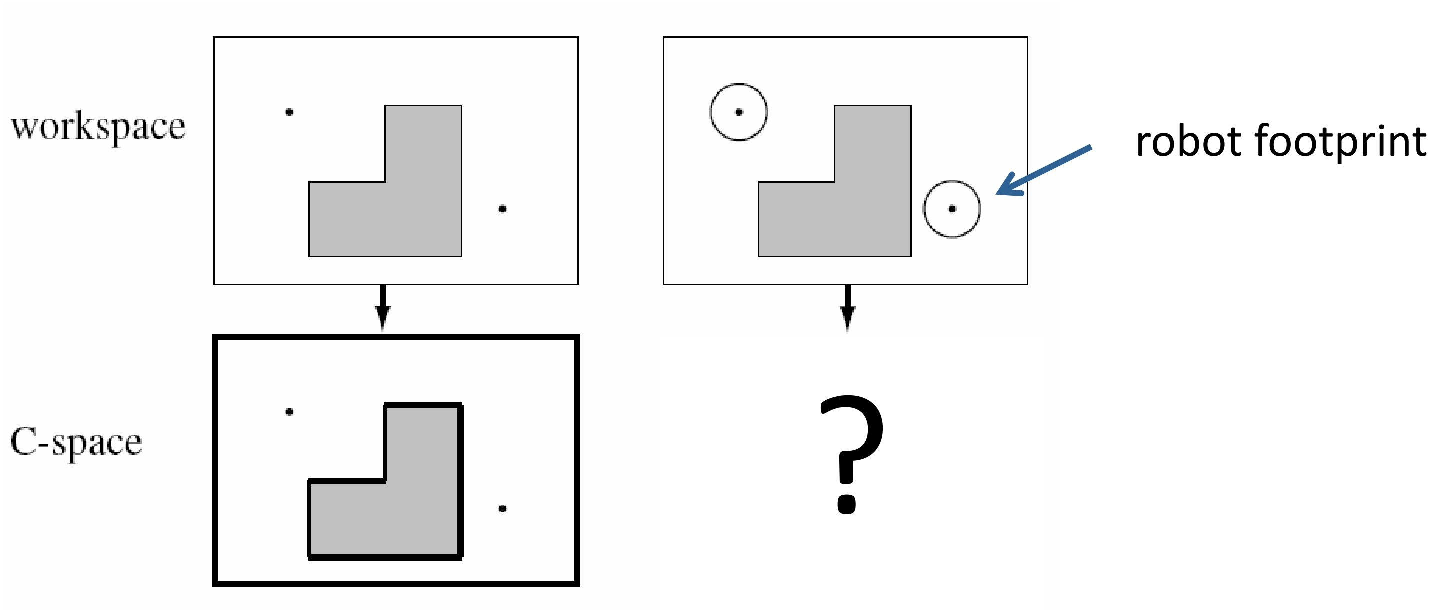
Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
- "Point" robot



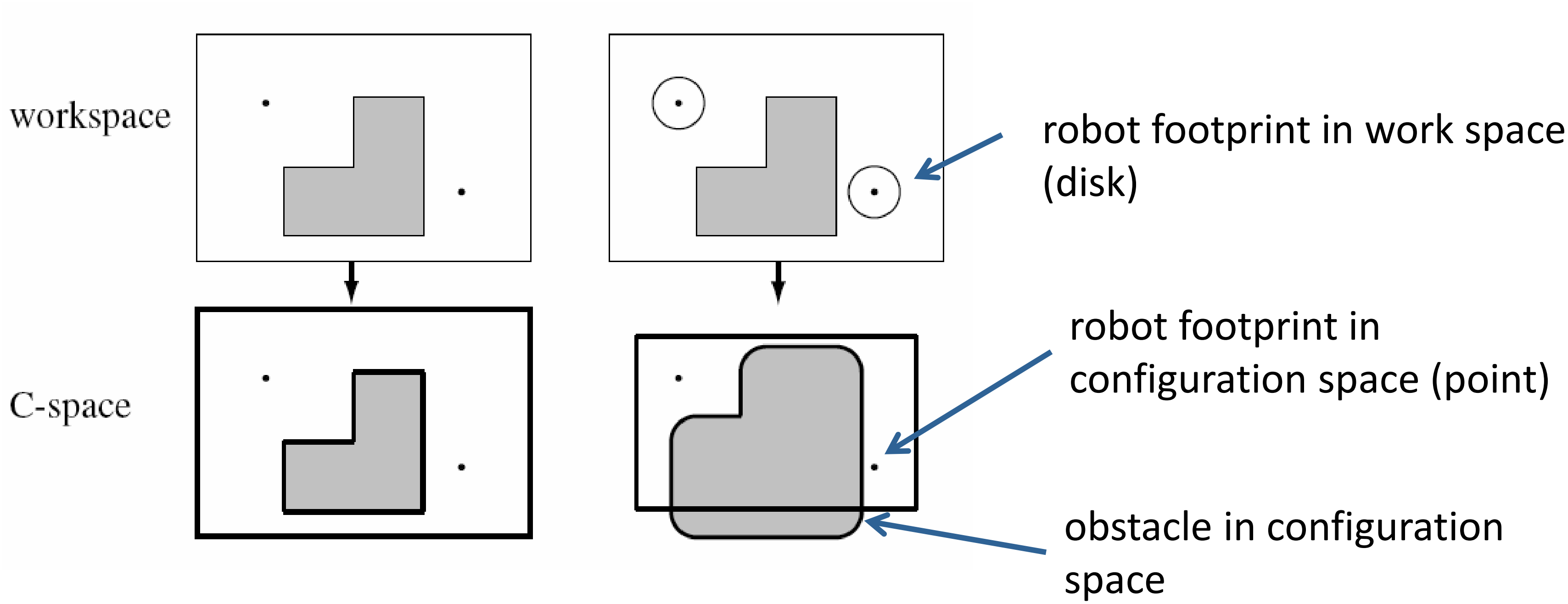
Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
- Circular robot



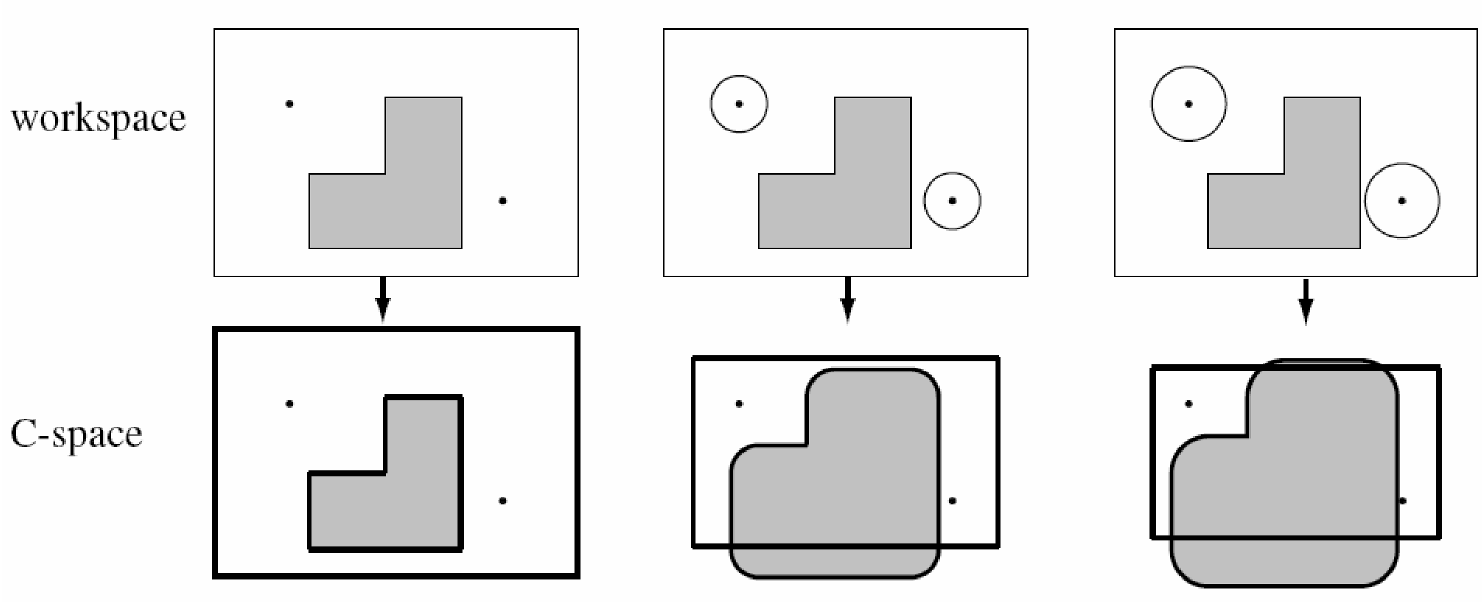
Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
- Circular robot



Example

- What are admissible configurations for the robot? Equiv.: What is the free space?
- Large circular robot



Computing the Free Space

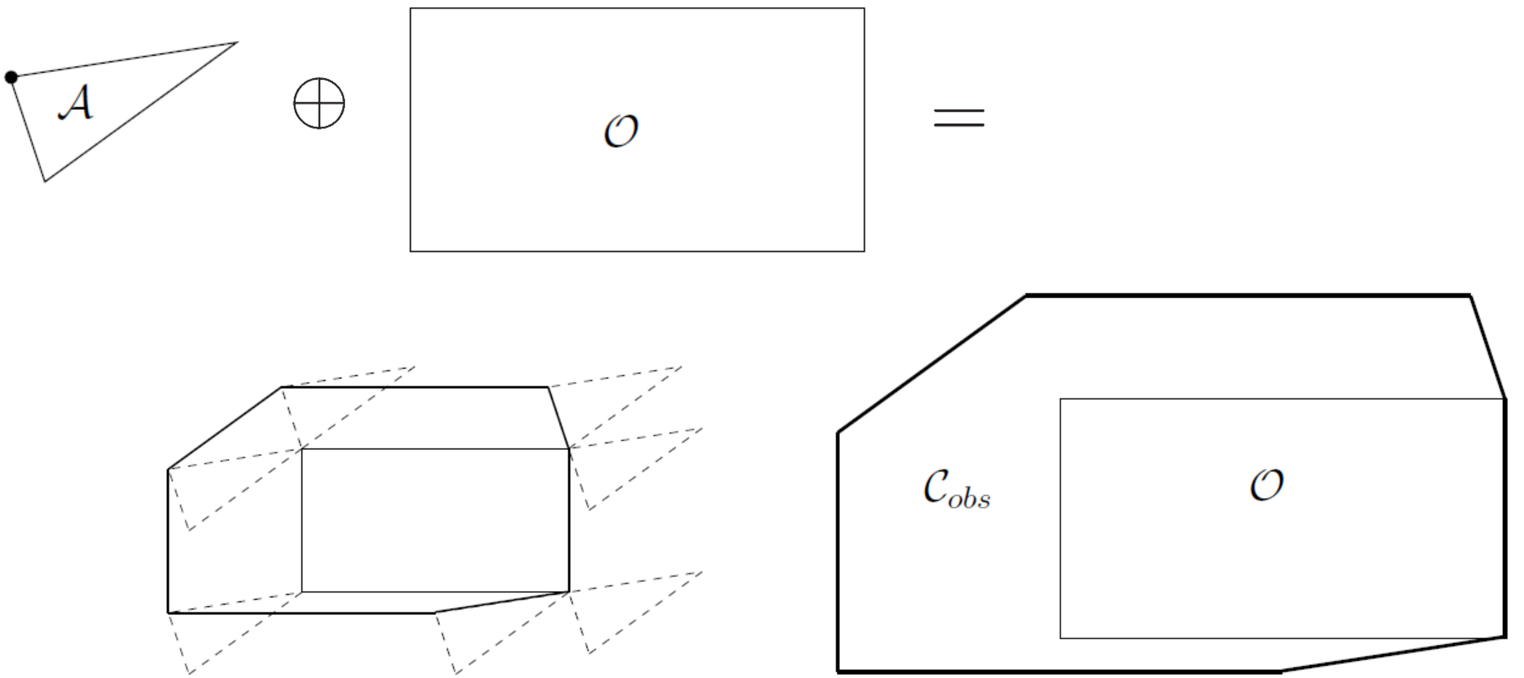
- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius
- This operation is called the **Minowski sum**

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$

where $A, B \subset \mathbb{R}^d$

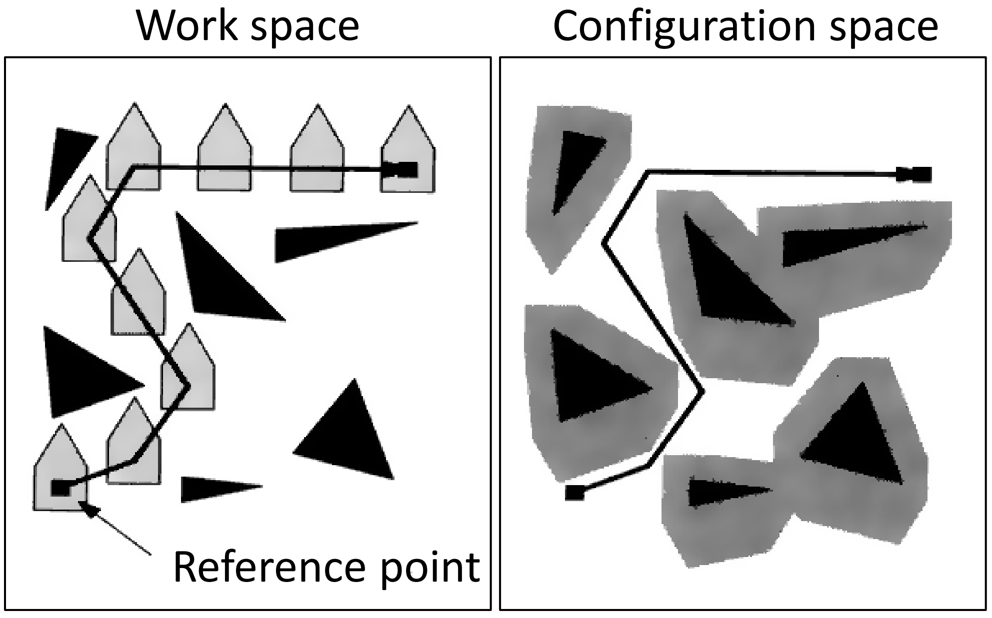
Example: Minowski Sum

- Triangular robot and rectangular obstacle



Example

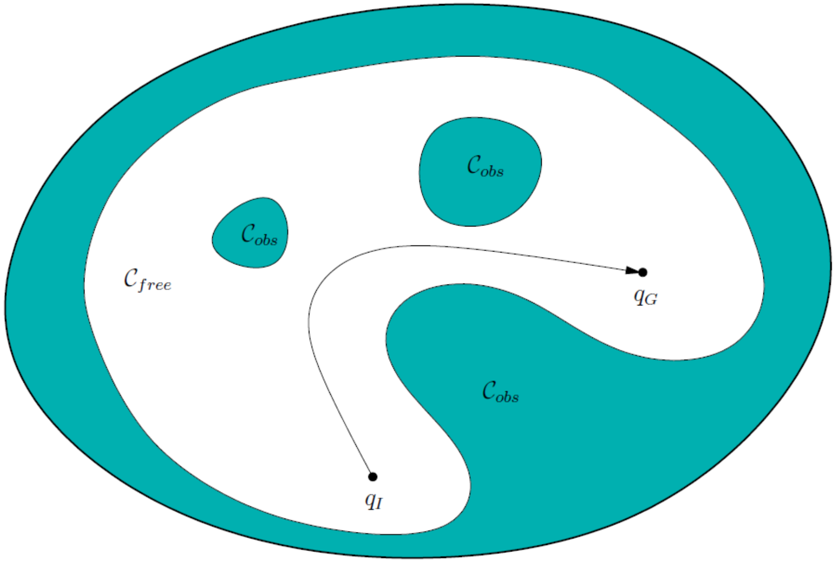
- Polygonal robot, translation only



- C-space is obtained by sliding the robot along the edge of the obstacle regions

Basic Motion Planning Problem

- **Given**
  - Free space C_{free}
  - Initial configuration $\mathbf{q}_I$
  - Goal configuration $\mathbf{q}_G$



- **Goal:** Find a continuous path

$$\tau : [0, 1] \rightarrow C_{\text{free}}$$

with $\tau(0) = \mathbf{q}_I, \; \tau(1) = \mathbf{q}_G$

Motion Planning Sub-Problems

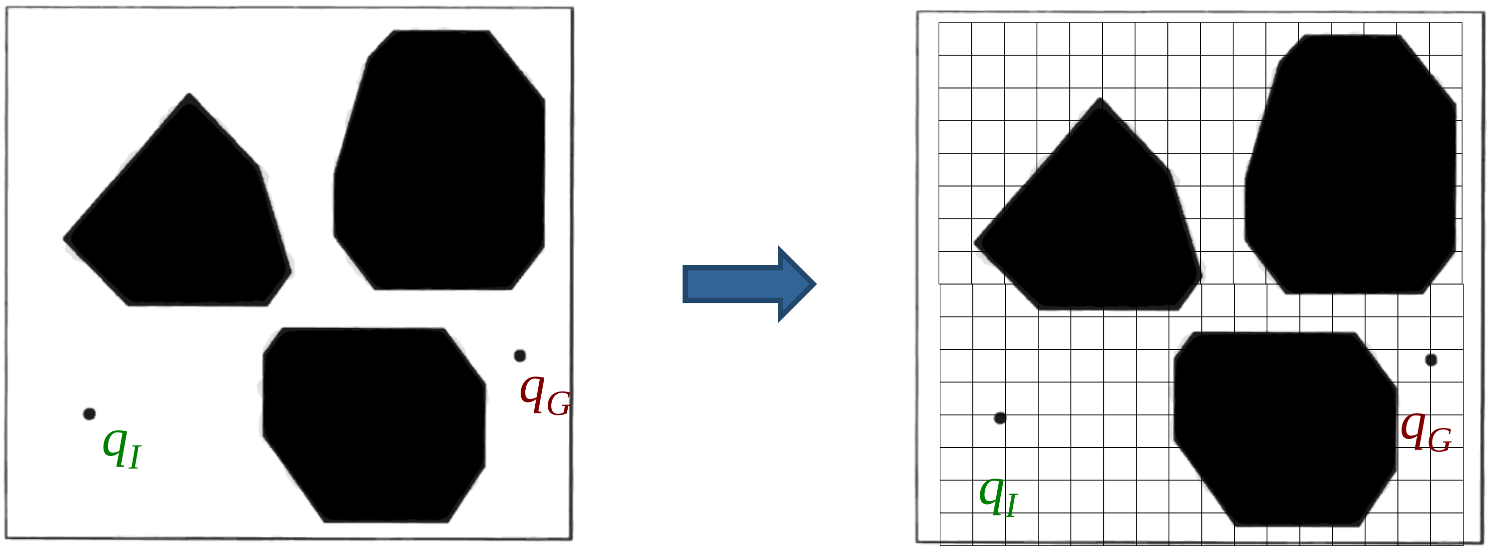
1. **C-Space discretization**
   (generating a graph / roadmap)
2. Search algorithm
   (Dijkstra's algorithm, A*, …)
3. Re-planning
   (D*, …)
4. Path tracking
   (PID control, potential fields, funnels, …)

C-Space Discretizations

- **Combinatorial planning**
  - Find a solution when one exists (complete)
  - Require polygonal decomposition
  - Become quickly intractable for higher dimensions
- **Sampling-based planning**
  - Weaker guarantees but more efficient
  - Need only point-wise evaluations of C_{free}
  - We will have a look at:
    grid decomposition, road maps, random trees

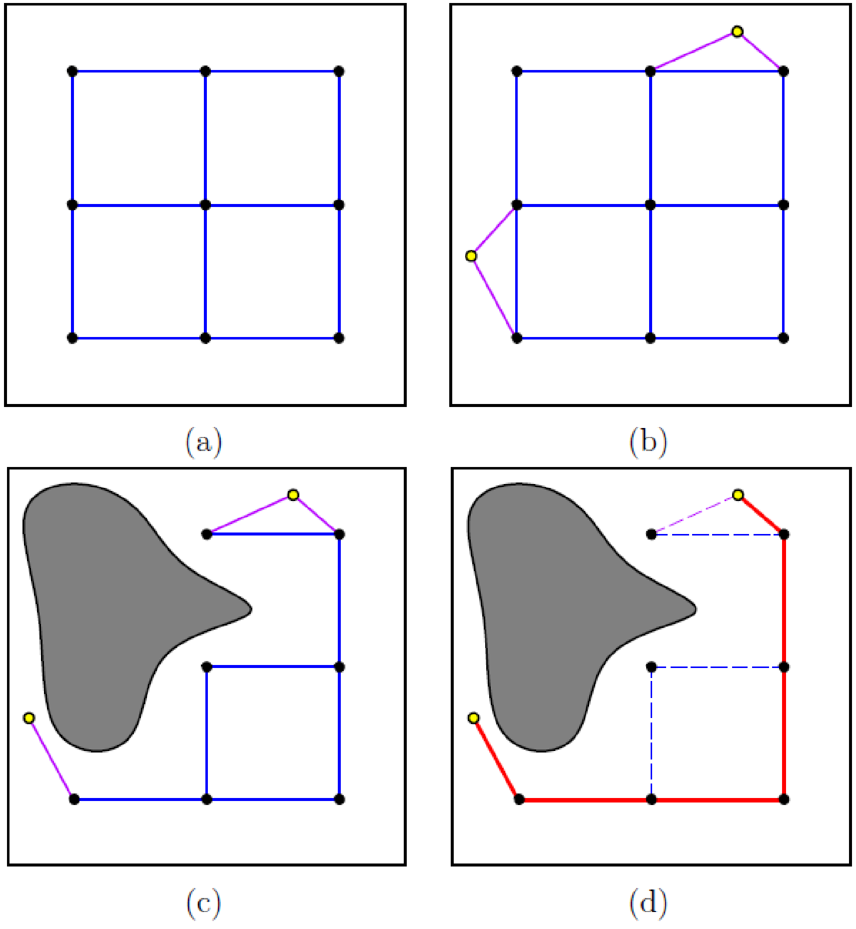
Grid Decomposition

- Construct a regular grid
- Determine status of every cell (free/occ)
- Simple, but not efficient (why?)
- Not exact (why?)



Grid Decomposition

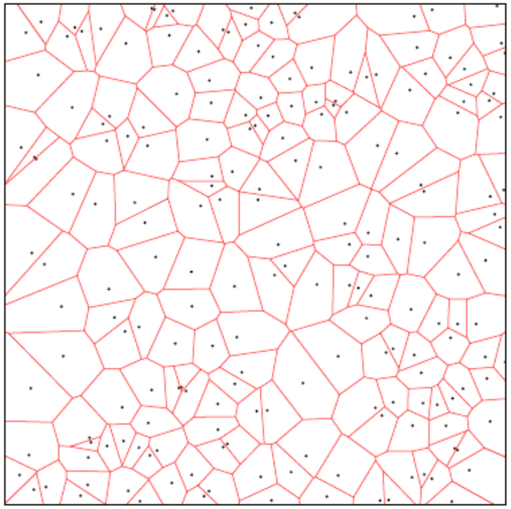
- Regular grid
- Construct graph
  - Grid cells as vertices
  - Edges encode traversability
- Query
  - Add start and goal to graph, connect to nearest neighbors
  - Perform graph search



Probabilistic Roadmaps (PRMs)

[Kavraki et al., 1992]

- Grids do not scale well to high dimensions
- Sampling-based approach
- **Vertex:** Take random sample from C, check whether sample is in C_{free}



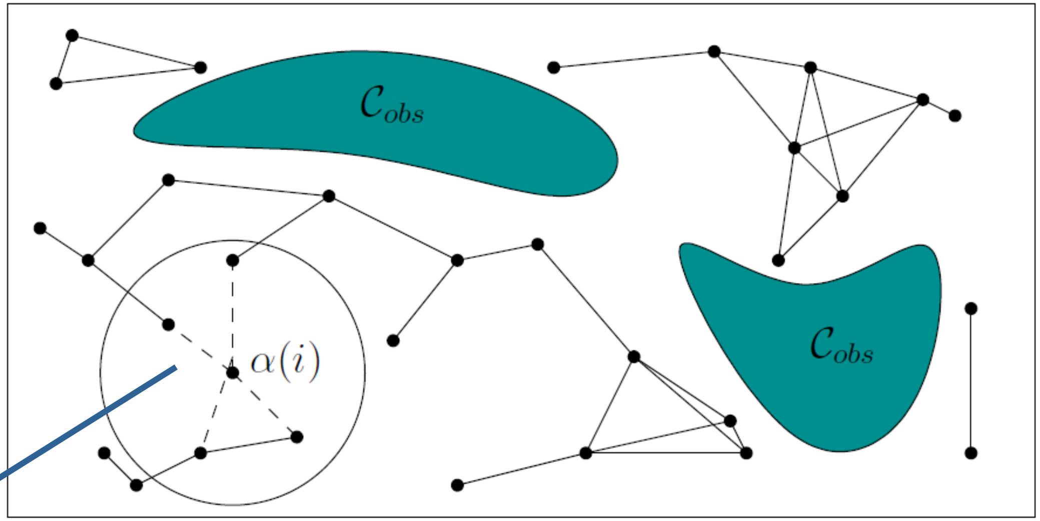
Probabilistic Roadmaps (PRMs)

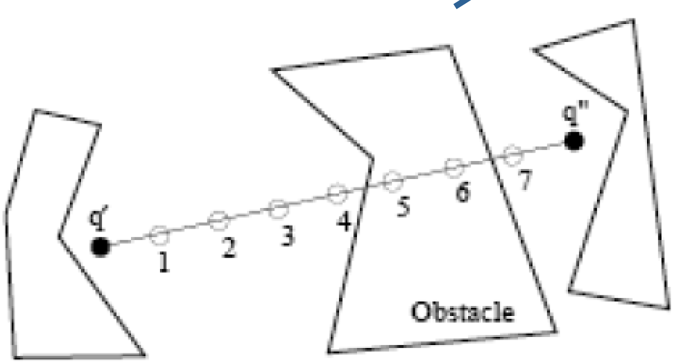
[Kavraki et al., 1992]

- **Vertex:** Take random sample from C, check whether sample is in C_{free}
- **Edge:** Check whether line-of-sight between two nearby vertices is collision-free
- Options for "nearby": k-nearest neighbors or all neighbors within specified radius
- Add vertices and edges until roadmap is dense enough

PRM Example

1. Sample vertex
2. Find neighbors
3. Add edges



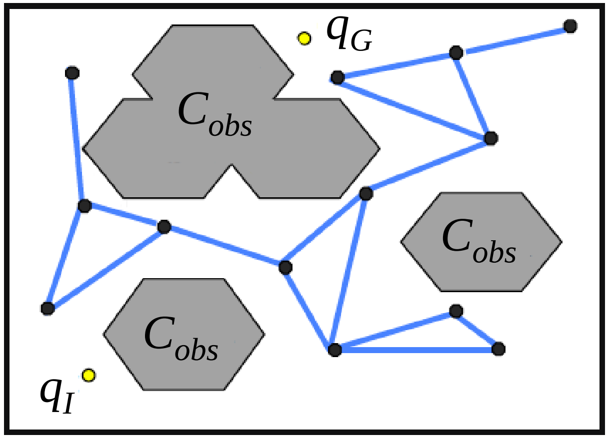


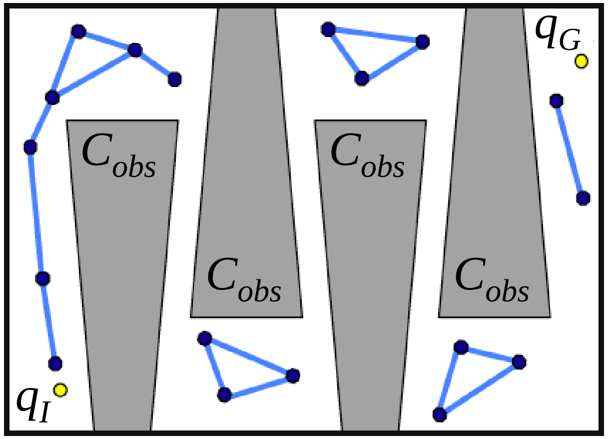
Step 3: Check edges for collisions, e.g., using discretized line search

Probabilistic Roadmaps

+ Probabilistic. complete
+ Scale well to higher dimensional C-spaces
+ Very popular, many extensions

- Do not work well for some problems (e.g., narrow passages)
- Not optimal, not complete

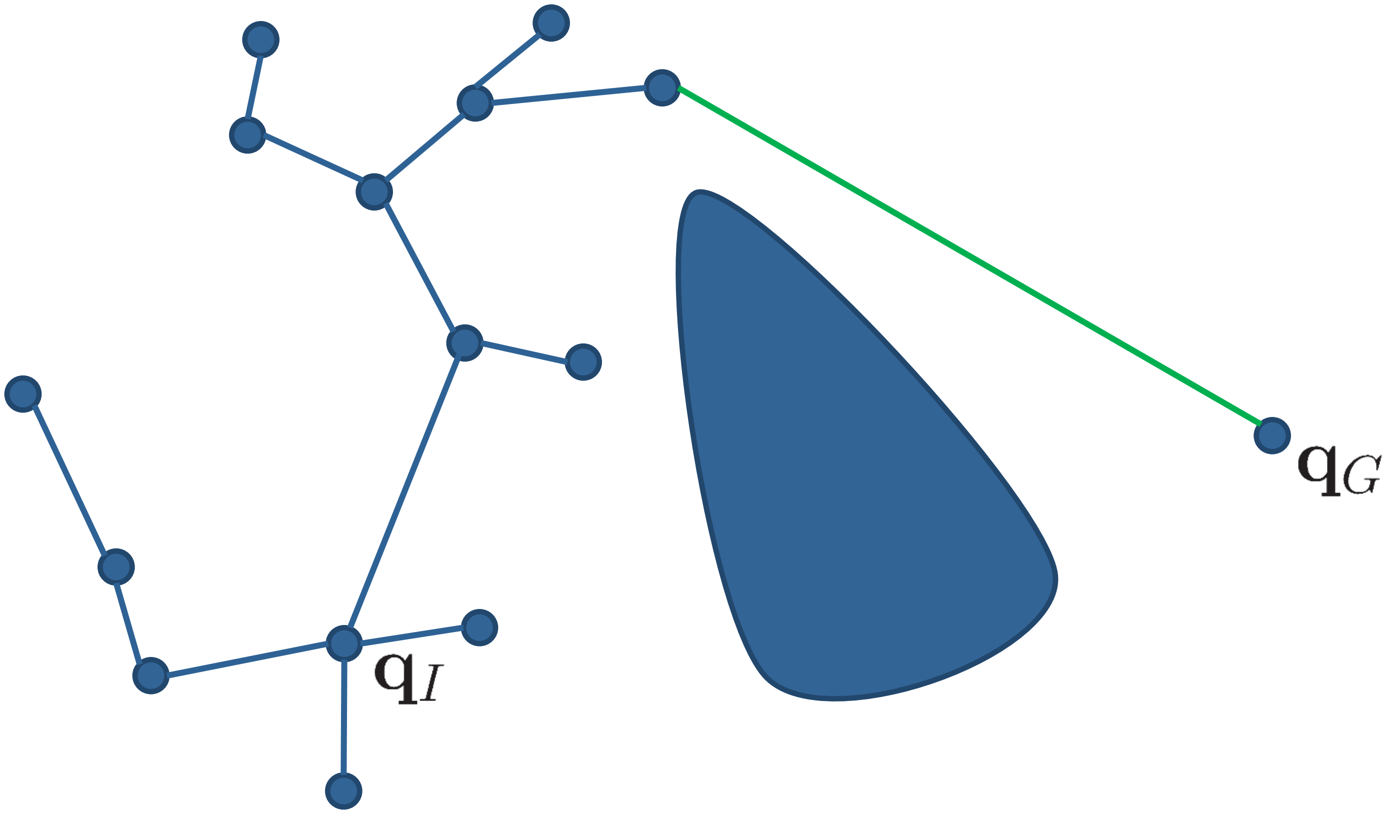




Rapidly Exploring Random Trees

[Lavalle and Kuffner, 1999]

- **Idea:** Grow a tree from start to goal location



Rapidly Exploring Random Trees

- **Algorithm**
  1. Initialize tree with first node $\mathbf{q}_I$
  2. Pick a random target location (every 100$^{\text{th}}$ iteration, choose $\mathbf{q}_G$)
  3. Find closest vertex in roadmap
  4. Extend this vertex towards target location
  5. Repeat steps until goal is reached

- Why not pick $\mathbf{q}_G$ every time?

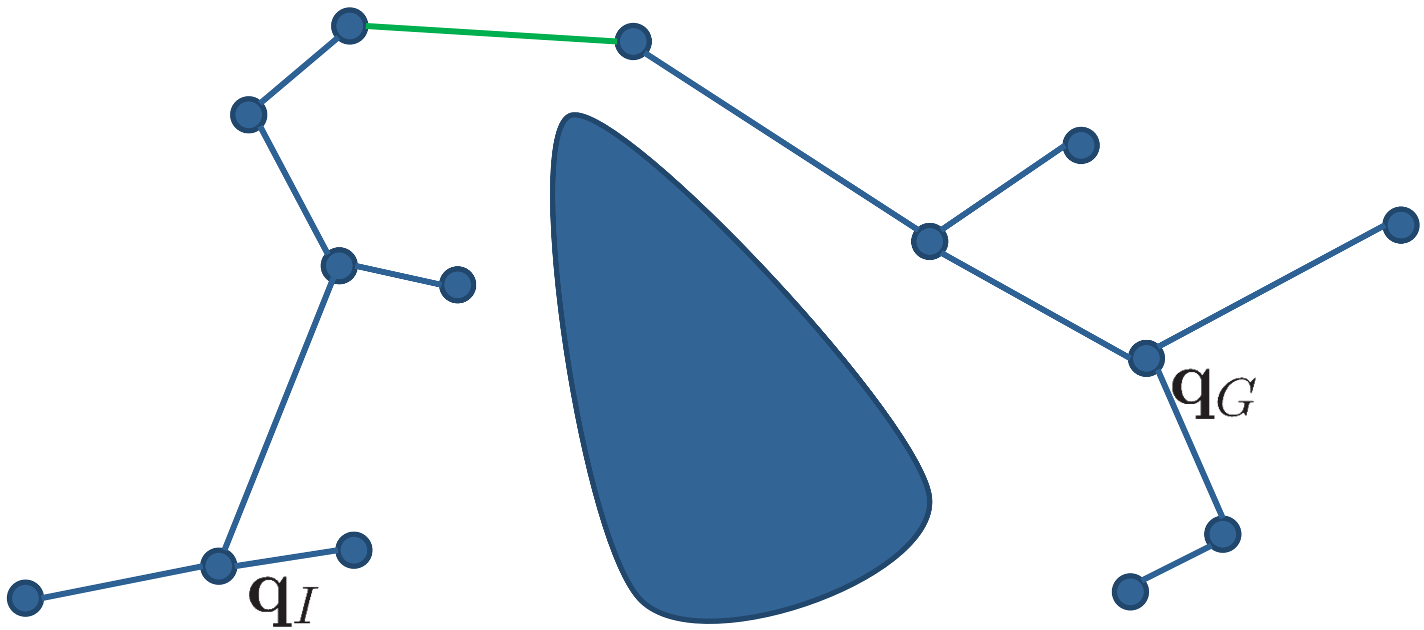
Rapidly Exploring Random Trees

- **Algorithm**
  1. Initialize tree with first node $\mathbf{q}_I$
  2. Pick a random target location (every 100th iteration, choose $\mathbf{q}_G$)
  3. Find closest vertex in roadmap
  4. Extend this vertex towards target location
  5. Repeat steps until goal is reached
- Why not pick $\mathbf{q}_G$ every time?
- This will fail and run into C_{obs} instead of exploring

Rapidly Exploring Random Trees

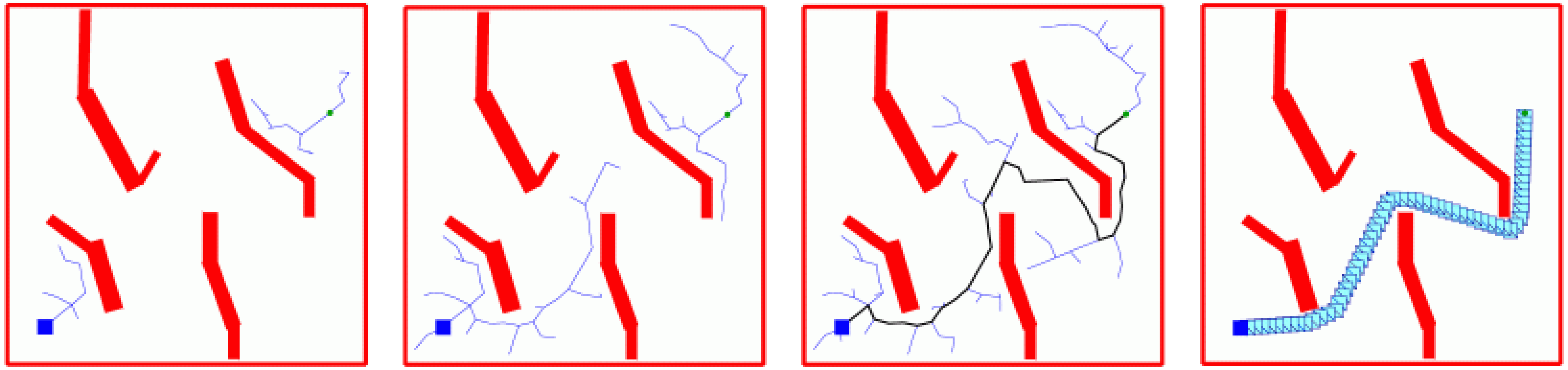
[Lavalle and Kuffner, 1999]

- **RRT:** Grow trees from start and goal location towards each other, stop when they connect

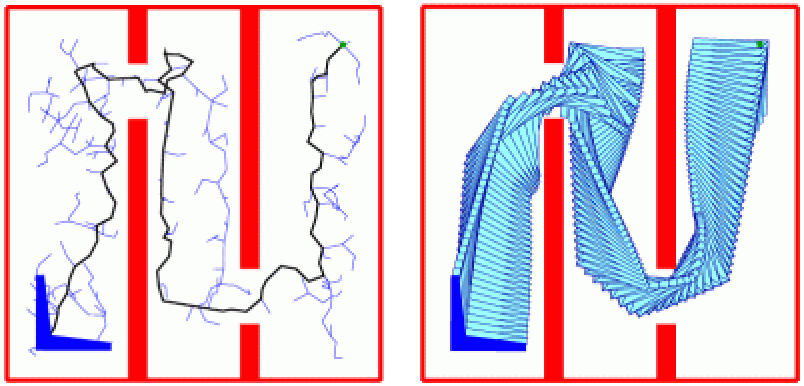


RRT Examples

- 2-DOF example

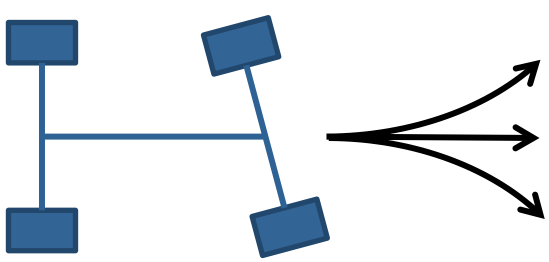


- 3-DOF example (2D translation + rotation)



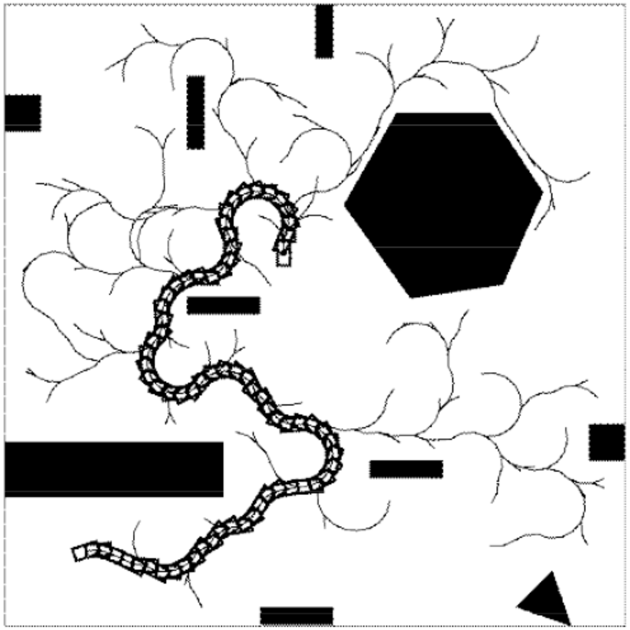
Non-Holonomic Robots

- Some robots cannot move freely on the configuration space manifold
- Example: A car can not move sideways
  - 2-DOF controls (speed and steering)
  - 3-DOF configuration space (2D translation + rotation)

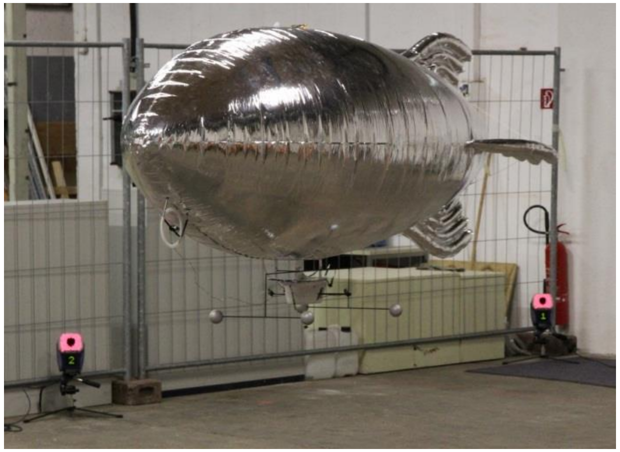


Non-Holonomic Robots

- RRTs can naturally consider such constraints during tree construction
- Example: Car-like robot



Example: Blimp Motion Planning
[Müller et al., IROS 2011]



Advantages

- Low power consumption
- Safe navigation capabilities

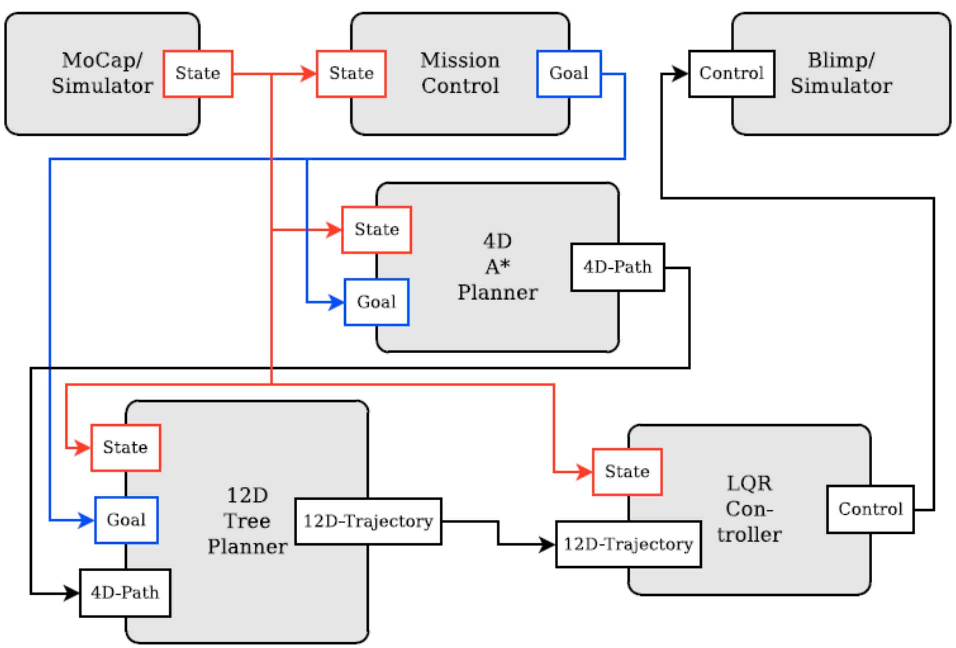
Challenges

- Seriously underactuated (only 3-DOF control)
- Heavily subject to drift
- Requires kinodynamic motion planning

Example: Blimp Motion Planning

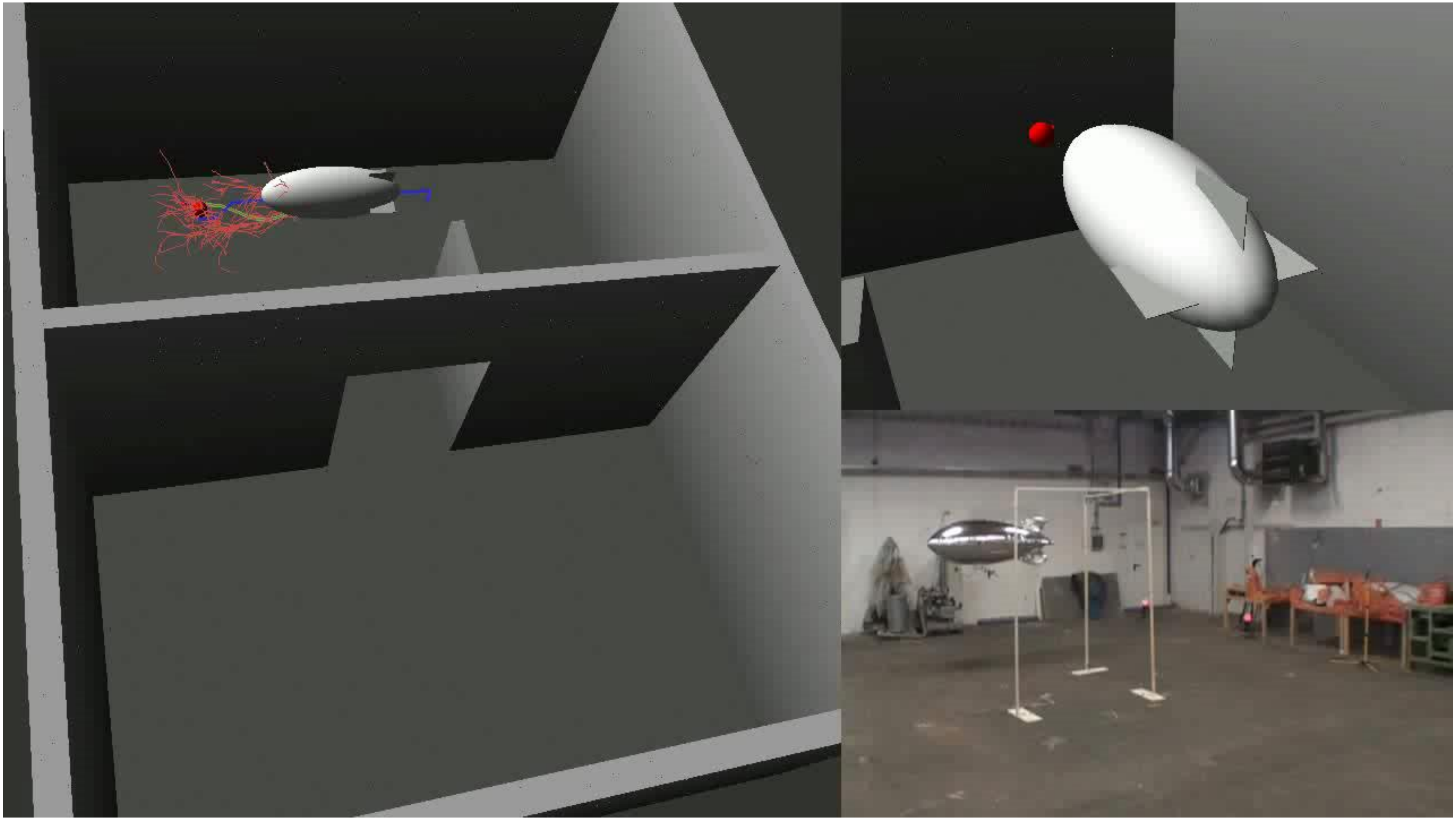
[Müller et al., IROS 2011]

- High-level planner: A* in 4D
- Low-level planner: RRT in 12D considering kinodynamic constraints



Example: Blimp Motion Planning

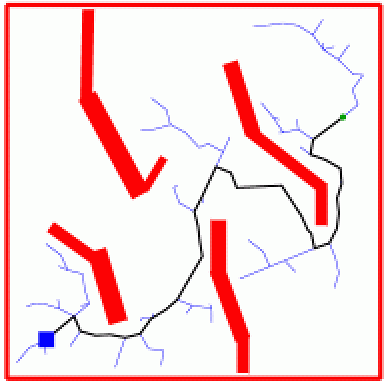
[Müller et al., IROS 2011]



Rapidly Exploring Random Trees

+ Probabilistic. complete
+ Balance between greedy search and exploration
+ Very popular, many extensions

- Metric sensitivity
- Unknown rate of convergence
- Not optimal, not complete



Summary: Sampling-based Planning

- **More efficient** in most **practical problems** but offer weaker guarantees
- **Probabilistically complete** (given enough time it finds a solution if one exists, otherwise, it may run forever)
- Performance degrades in problems with **narrow passages**

Motion Planning Sub-Problems

1. C-Space discretization
   (generating a graph / roadmap)
2. **Search algorithms**
   (Dijkstra's algorithm, A*, …)
3. **Re-planning**
   (D*, …)
4. Path tracking
   (PID control, potential fields, funnels, …)

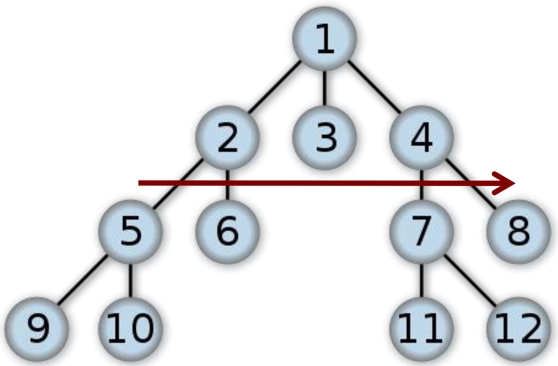
Search Algorithms

- **Given:** Graph G consisting of vertices and edges (with associated costs)
- **Wanted:** Find the best (shortest) path between two vertices

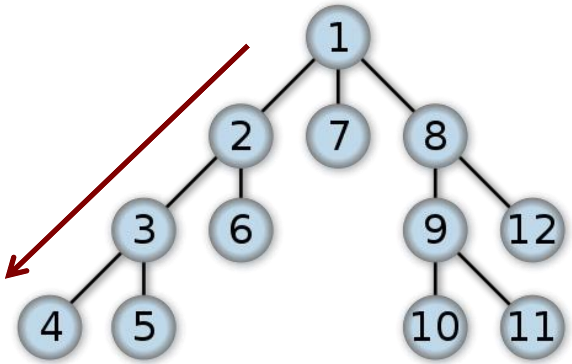
- What search algorithms do you know?

Uninformed Search

- **Breadth-first**
  - Complete
  - Optimal if action costs equal
  - Time and space $O(b^d)$

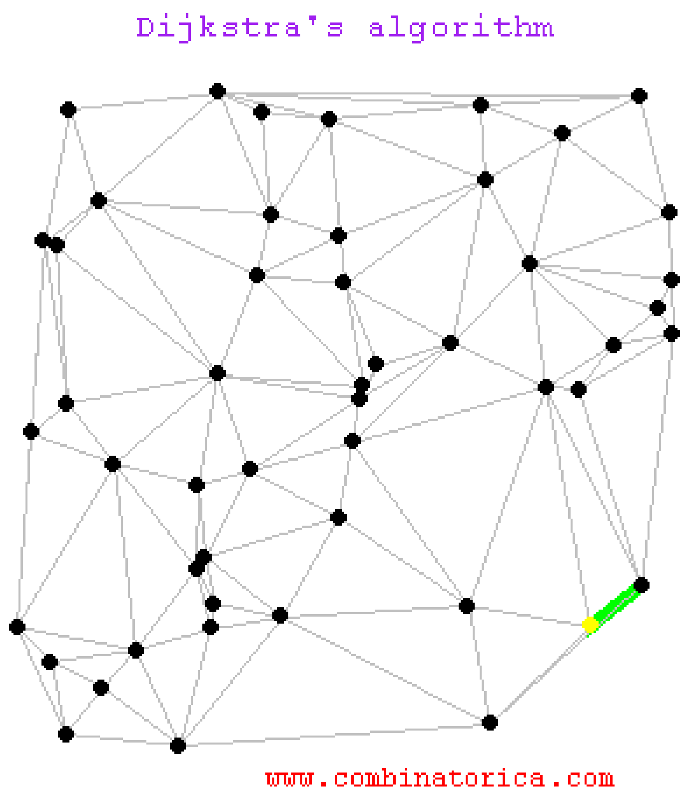


- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$
    (can forget explored subtrees)



Example: Dijkstra’s Algorithm

- Extension of breadth-first with arbitrary (non-negative) costs



Informed Search

- **Idea**
  - Select nodes for further expansion based on an evaluation function $f(s)$
  - First explore the node with lowest value
- What is a good evaluation function?

Informed Search

- **Idea**
  - Select nodes for further expansion based on an evaluation function $f(s)$
  - First explore the node with lowest value
- What is a good evaluation function?
- Often a combination of
  - Path cost so far $g(s)$
  - Heuristic function $h(s)$
    (e.g., estimated distance to goal, but can also encode additional domain knowledge)

What is a Good Heuristic Function?

- Choice is problem/application-specific
- Popular choices
  - Manhattan distance (neglecting obstacles)
  - Euclidean distance (neglecting obstacles)
  - Value iteration / Dijkstra (from the goal backwards)

Informed Search

- **A* search**
  - Combines path cost with estimated goal distance

$$f(s) = g(s) + h(s)$$

  - Heuristic function $h(s)$ has to be
    - Admissible (never over-estimate the true cost)

$$h(s) < c^*(s, s_{\text{goal}})$$

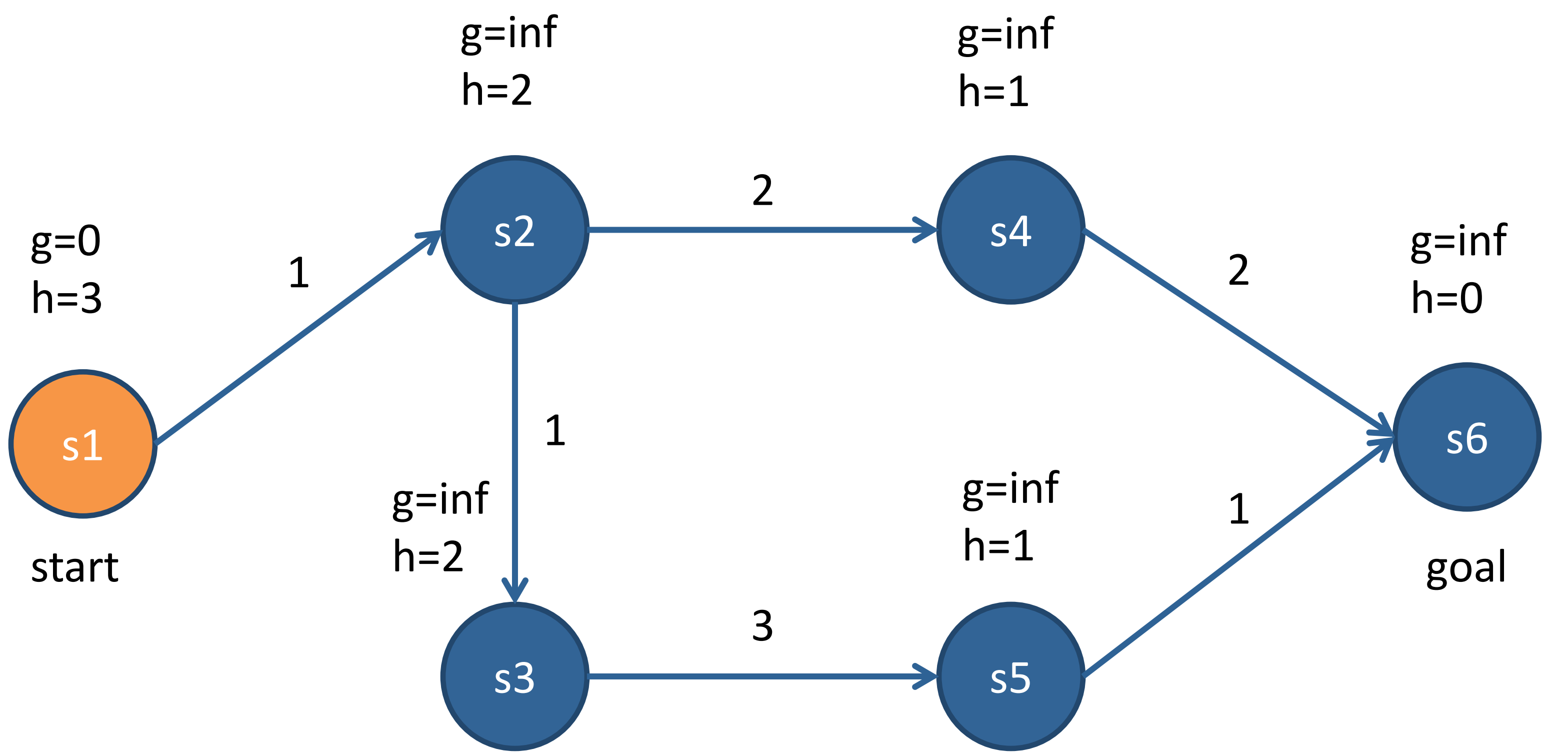
    - Consistent (satisfies triangle inequality)
- **A* is optimal** (in the number of expanded nodes) and **complete** (finds a solution if there is one and fails otherwise)

A* Algorithm

- Initialize
  - OPEN = {start}, CLOSED = {}
  - f(s) = inf
- While goal not in CLOSED
  - Remove vertex s from OPEN with smallest estimated cost f(s)
  - Insert s into CLOSED
  - For every successor s’ of s not yet in CLOSED,
    - Update g(s’) = min(g(s’), g(s) + c(s,s’))
    - Insert s’ into OPEN

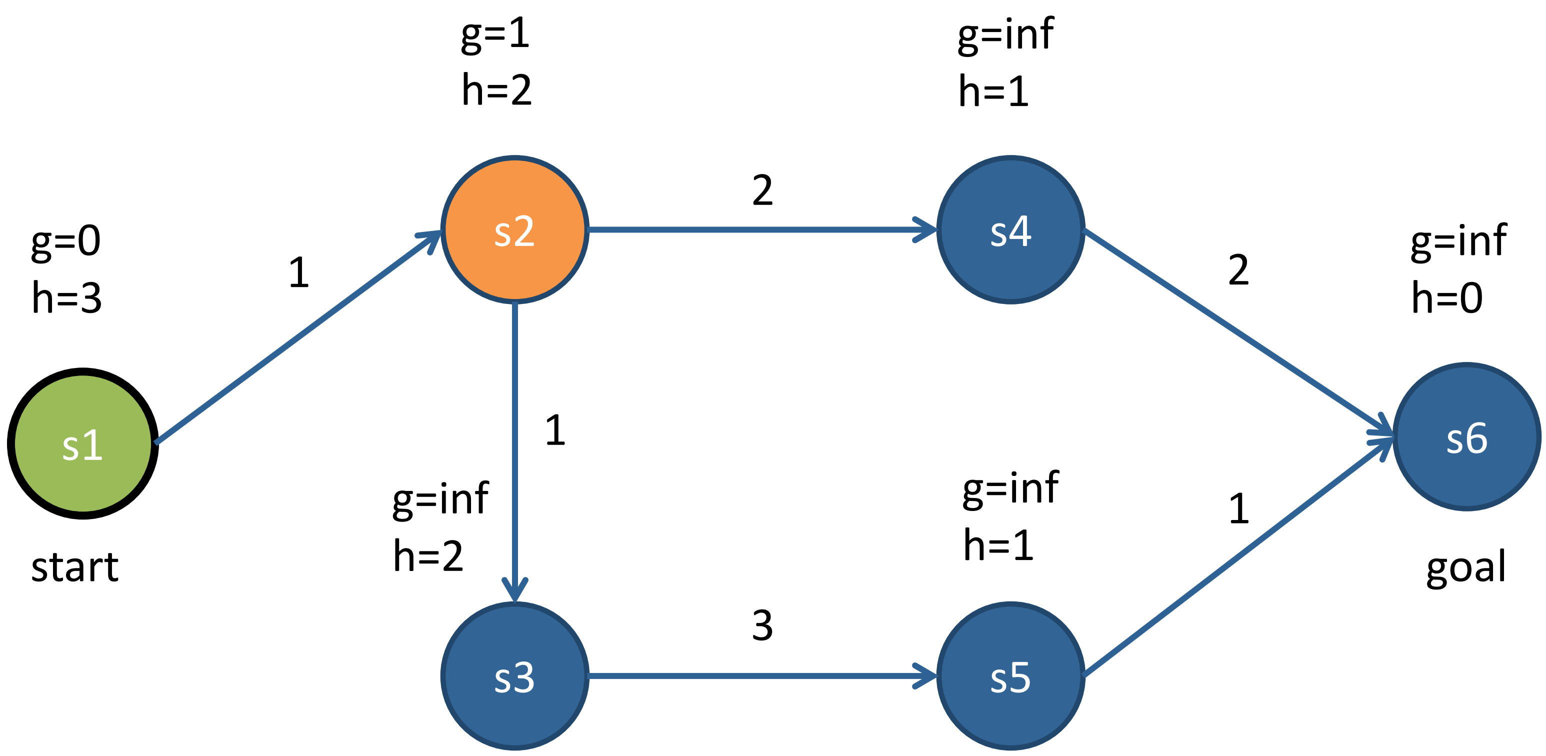
A* Example

- OPEN = {s1}
- CLOSED = {}



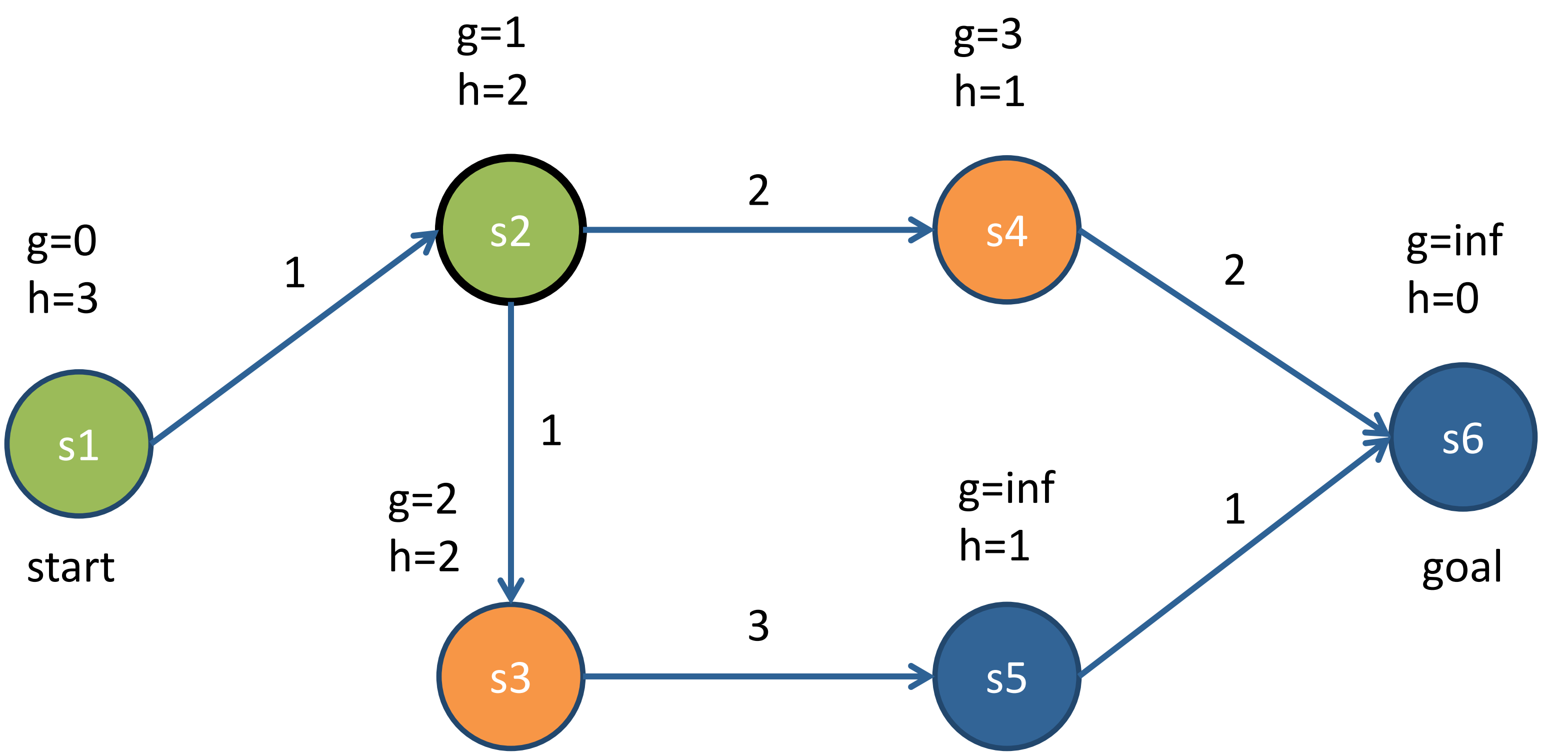
A* Example

- OPEN = {s2}
- CLOSED = {s1}



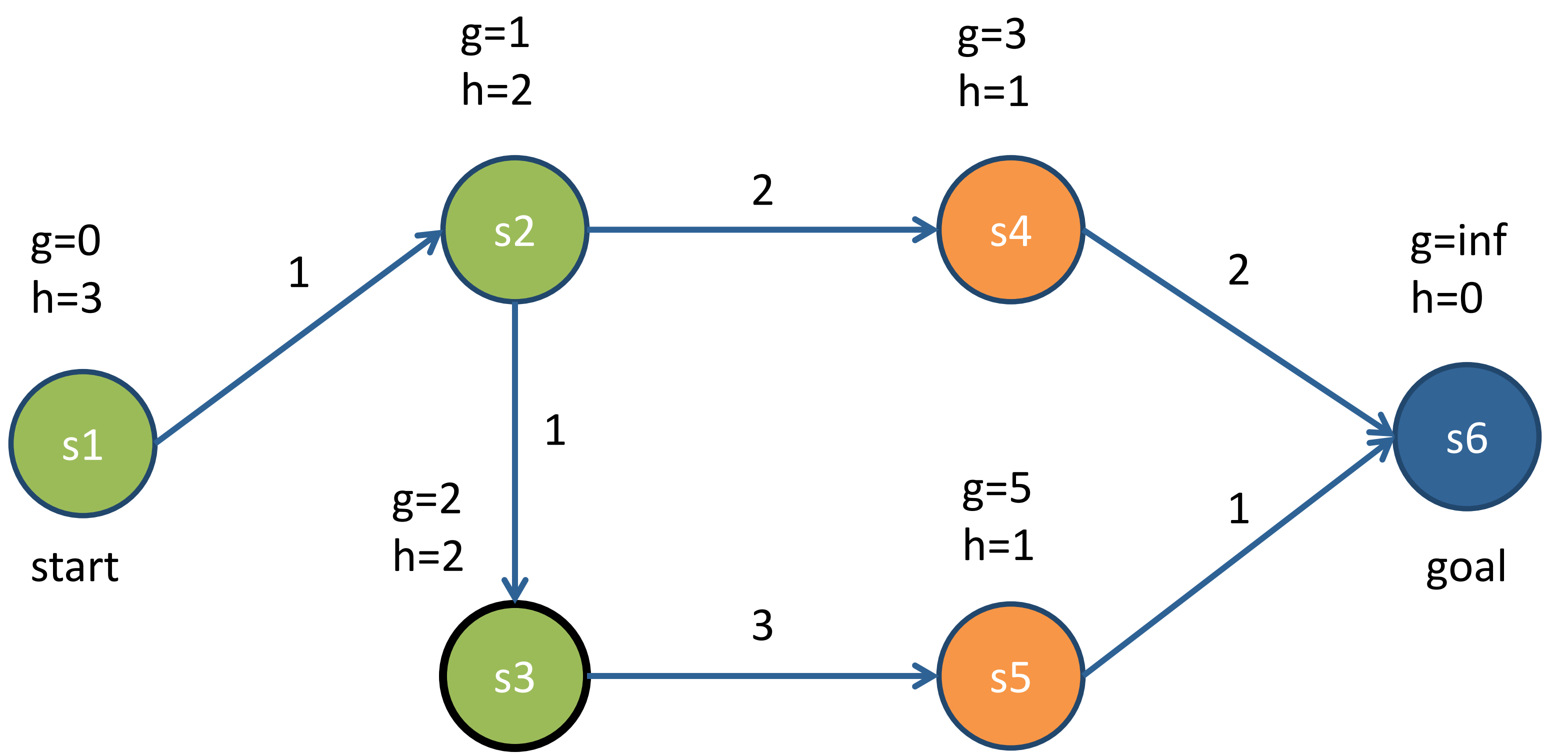
A* Example

- OPEN = {s3,s4}
- CLOSED = {s1,s2}



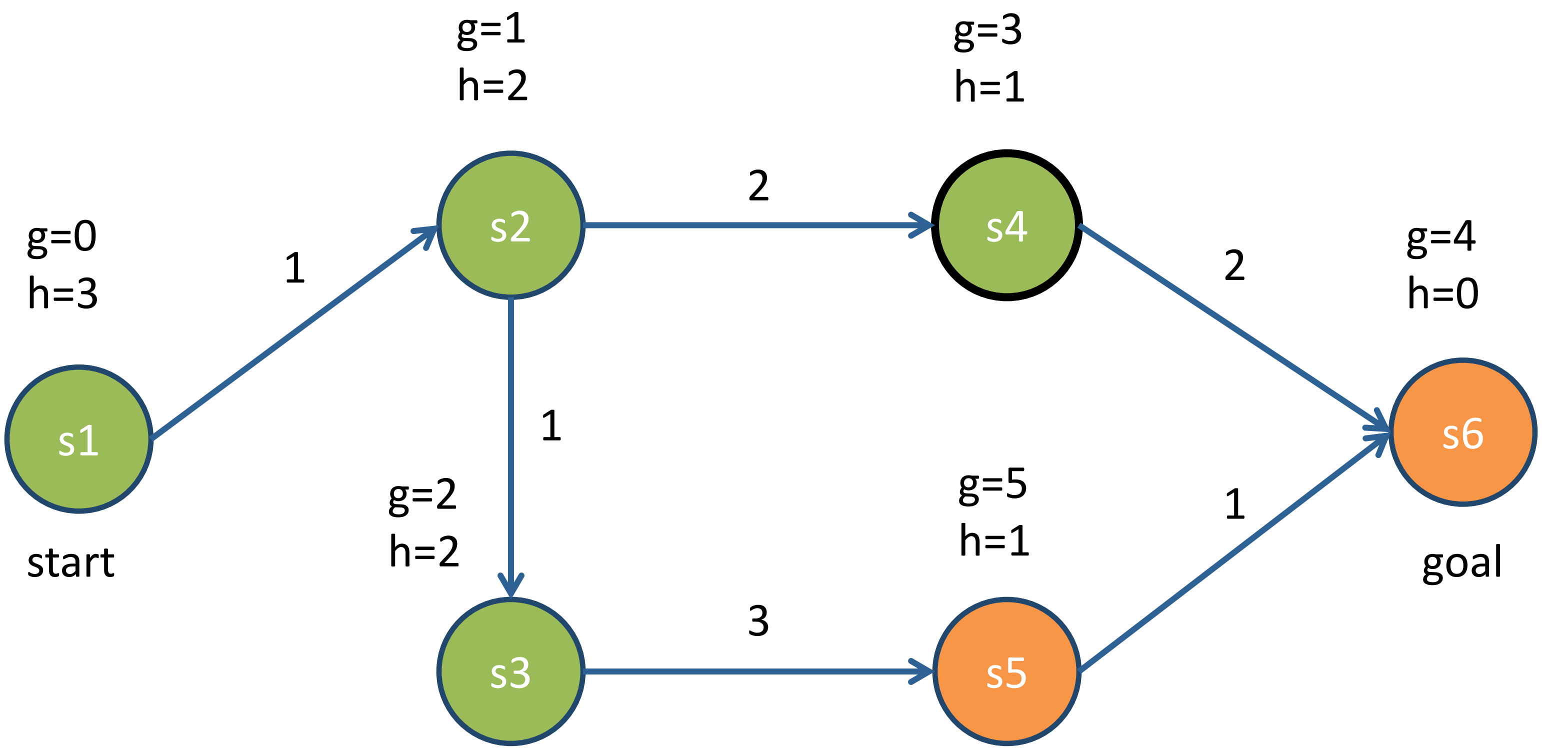
A* Example

- OPEN = {s4}
- CLOSED = {s1,s2,s3}



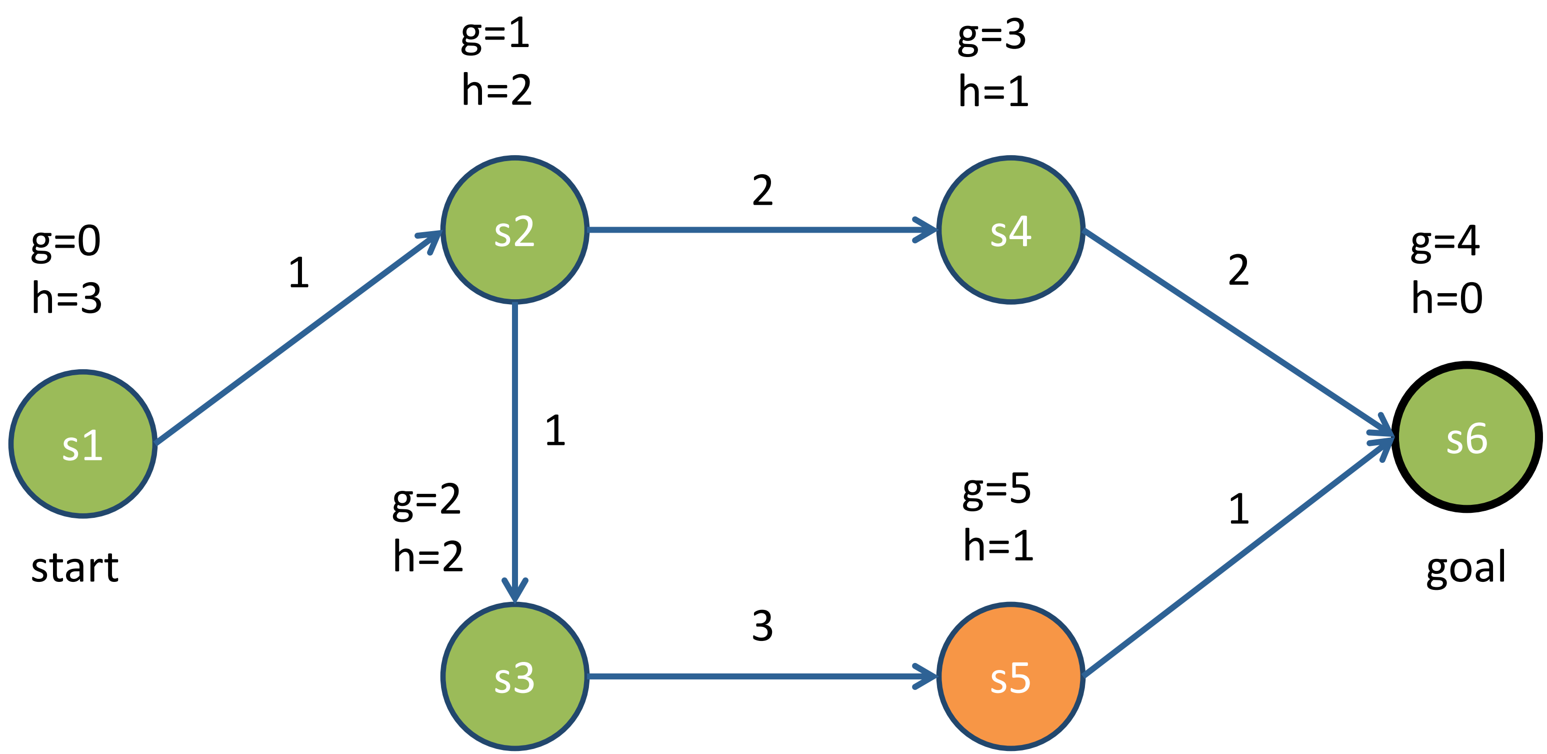
A* Example

- OPEN = {s5,s6}
- CLOSED = {s1,s2,s3,s4}



A* Example

- OPEN = {s5}
- CLOSED = {s1,s2,s3,s4,**s6**}

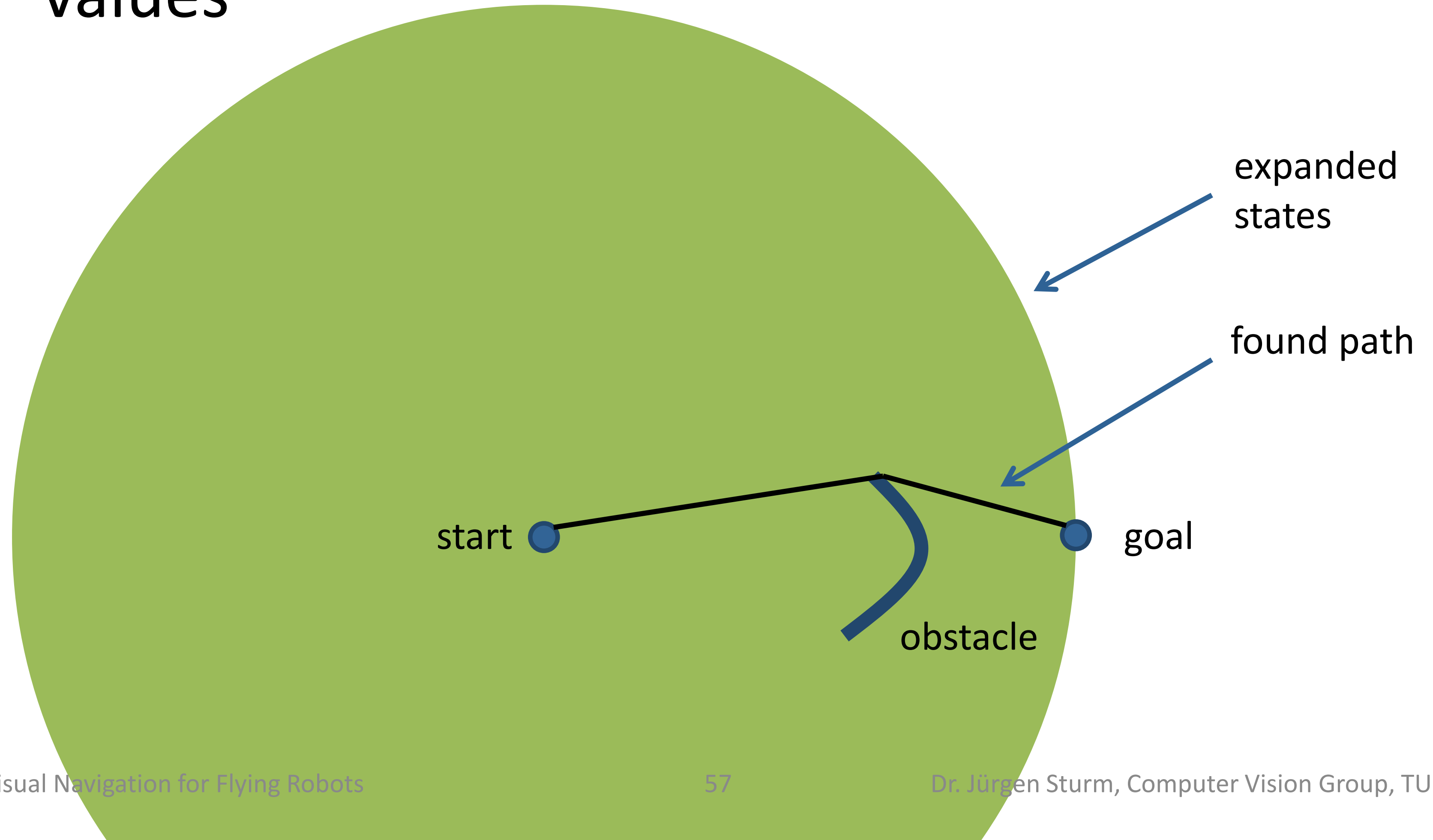


Effect of the Heuristic Function

- Consider the following path planning problem
- How many states will be expanded by the previous search algorithms?

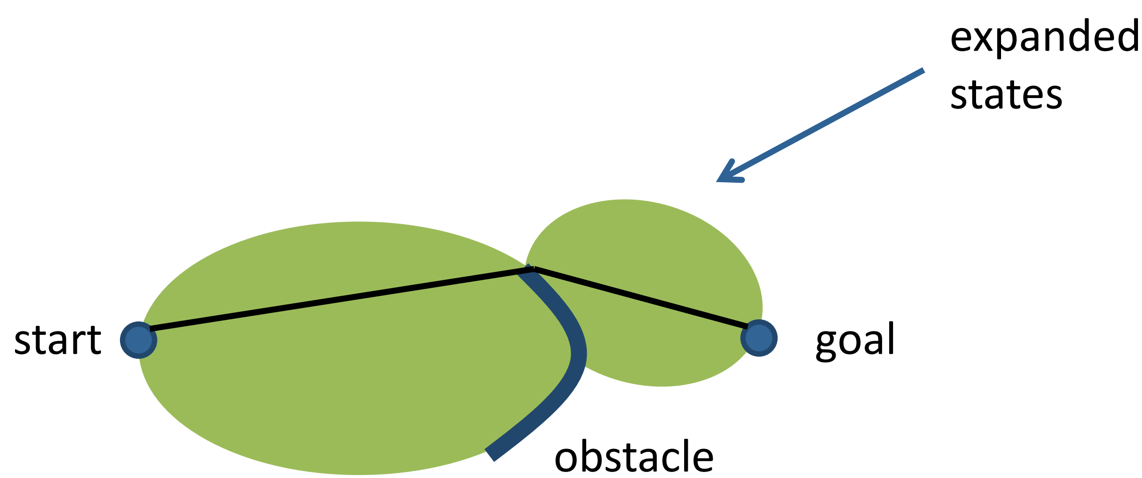
Effect of the Heuristic Function

- Dijkstra expands states in the order of f=g values



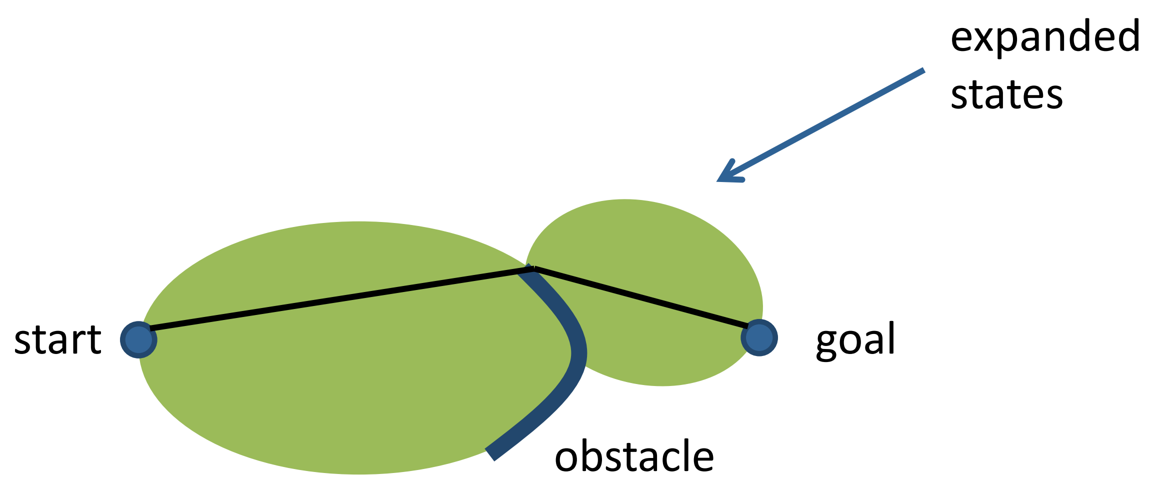
Effect of the Heuristic Function

- A* expands states in the order of f=g+h values



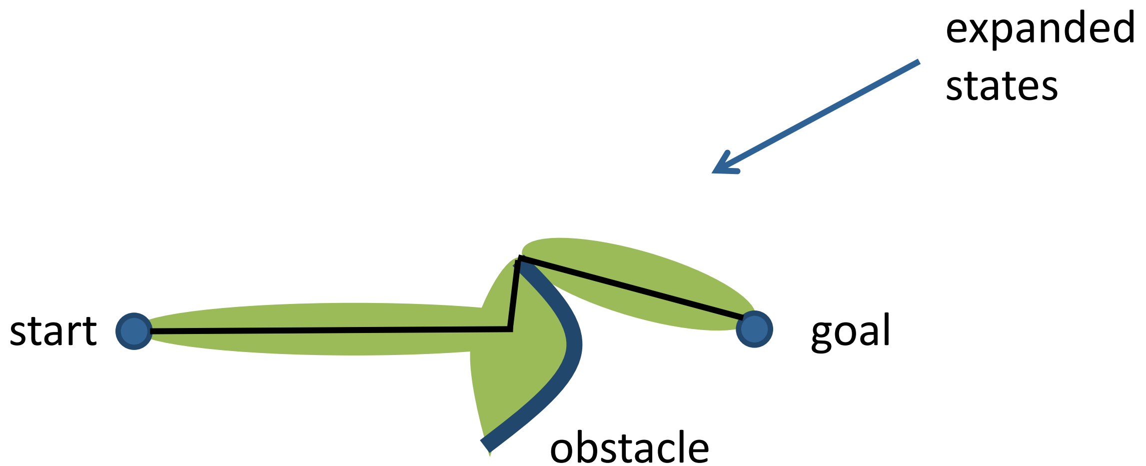
Effect of the Heuristic Function

- A* expands states in the order of f=g+h values
- For large problems, this results in A* quickly running out of memory (many OPEN/CLOSED states)



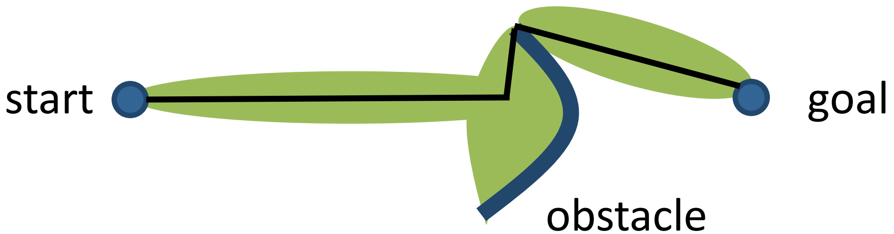
Effect of the Heuristic Function

- Weighted A* search expands states in the order of $f=g+\varepsilon h$
- $\varepsilon>1$ → bias towards states that are closer to the goal



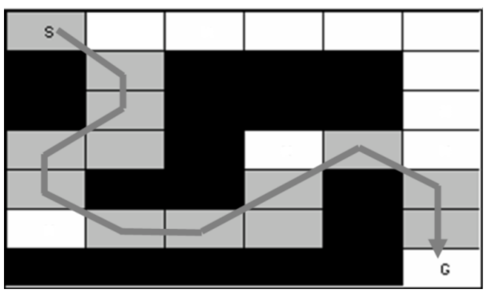
Effect of the Heuristic Function

- Weighted A* search expands states in the order of $f=g+\varepsilon h$
- $\varepsilon>1$ → bias towards states that are closer to the goal
- Search is typically orders of magnitude faster
- Found path may be longer (by a factor of ε)

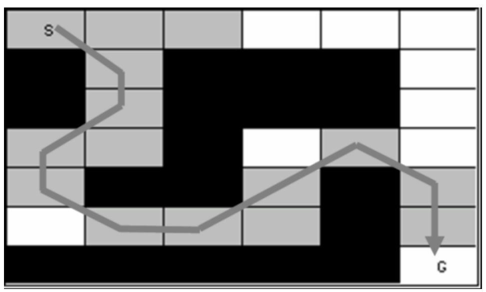


Anytime A*

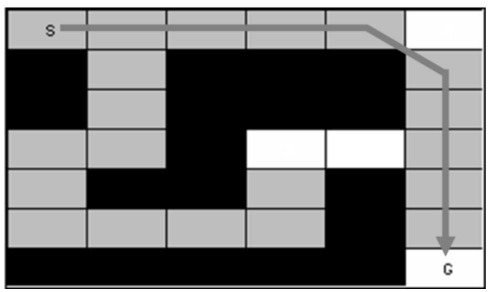
- Constructing anytime search based on A*
  - Find the best possible path for a given ε
  - Reduce ε and re-plan



ε=2.5
expansions: 13
moves: 11



ε=1.5
expansions: 15
moves: 11



ε=1.0
expansions: 20
moves: 10

Comparison Search Algorithms

D* Search

- **Problem:** In unknown, partially known or dynamic environments, the planned path may be blocked and we need to **replan**
- Can this be done efficiently, avoiding to replan the **entire path**?

D* Search

- **Idea:** Incrementally repair path keeping its modifications local around robot pose
- Many variants:
  - D* (Dynamic A*) [Stentz, ICRA ’94] [Stentz, IJCAI ‘95]
  - D* Lite [Koenig and Likhachev, AAAI ‘02]
  - Field D* [Ferguson and Stenz, JFR ‘06]

D* Search

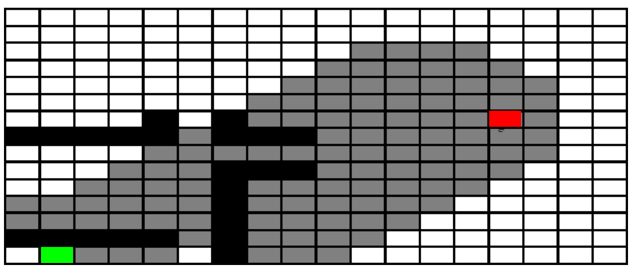
Main concepts

- **Invert search direction** (from goal to start)
  - Goal does not move, but robot does
  - Map changes (new obstacles) have only local influence close to current robot pose
- **Mark** the changed node and all dependent nodes **as unclean** (=to be re-evaluated)
- **Find shortest path** to start (using A*) while **re-using previous solution**

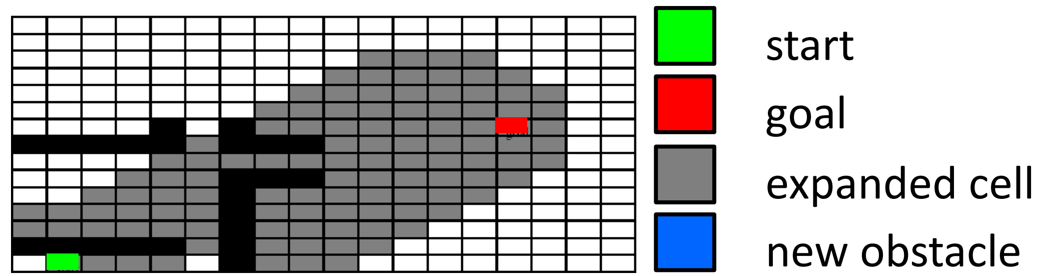
D* Example

- Initial search

Backwards A*

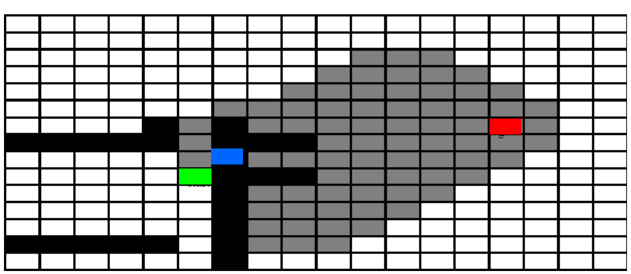


D* Lite

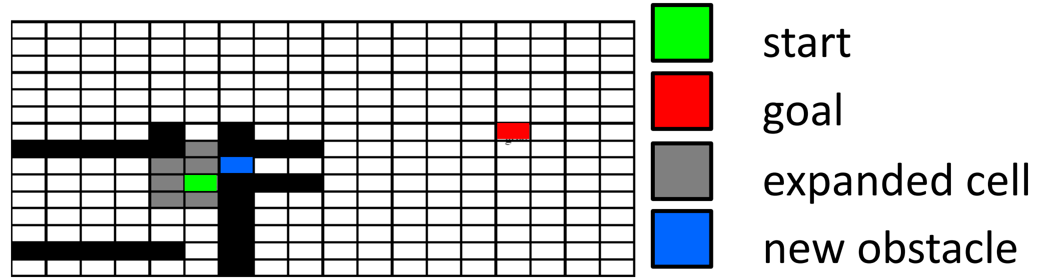


- Second search

Backwards A*

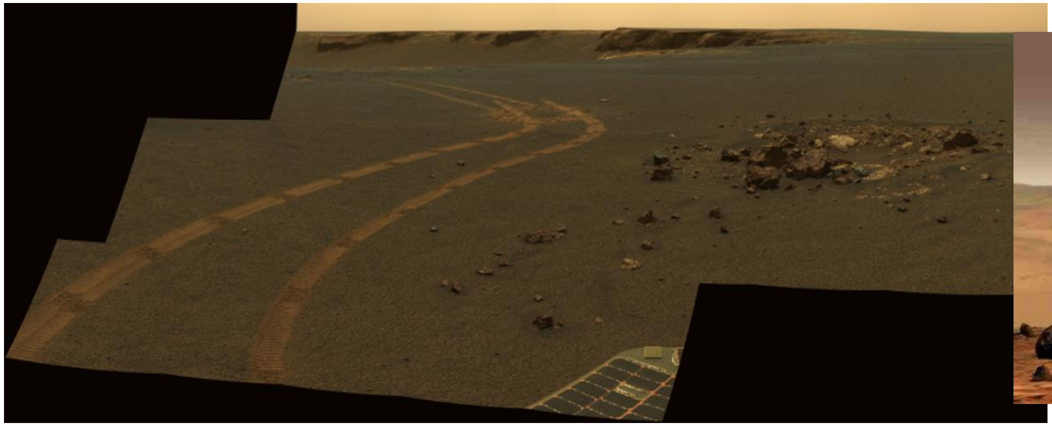


D* Lite



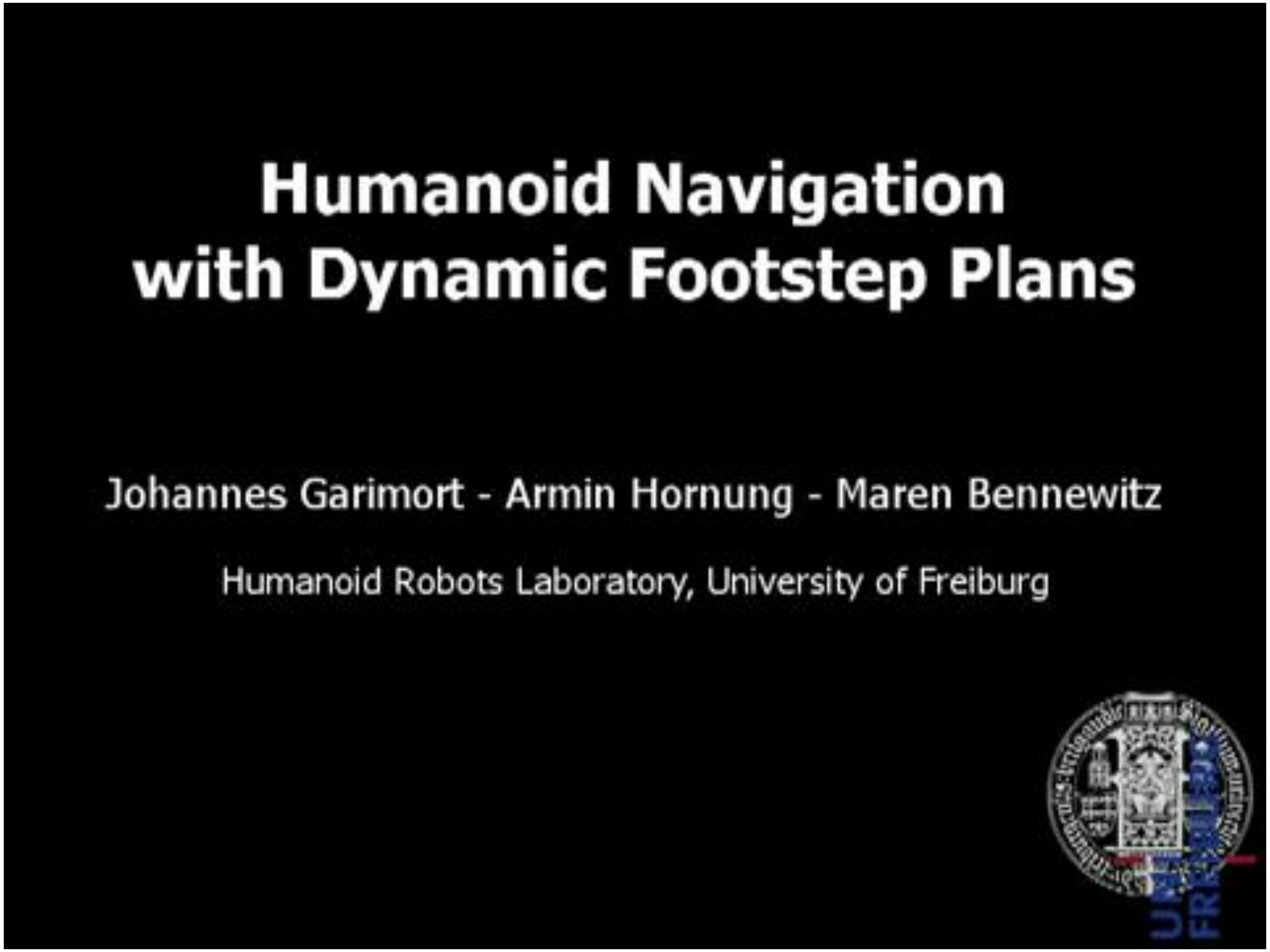
D* Search

- D* is as optimal and complete as A*
- D* and its variants are widely used in practice
- Field D* was running on Mars rovers Spirit and Opportunity



D* Lite for Footstep Planning

[Garimort et al., ICRA ‘11]



Problems on A*/D* on Grids

1. The shortest path is often very **close to obstacles** (cutting corners)
   - Uncertain path execution increases the risk of collisions
   - Uncertainty can come from delocalized robot, imperfect map, or poorly modeled dynamic constraints
2. Trajectories are **aligned to grid** structure
   - Path looks unnatural
   - Paths are longer than the true shortest path in continuous space

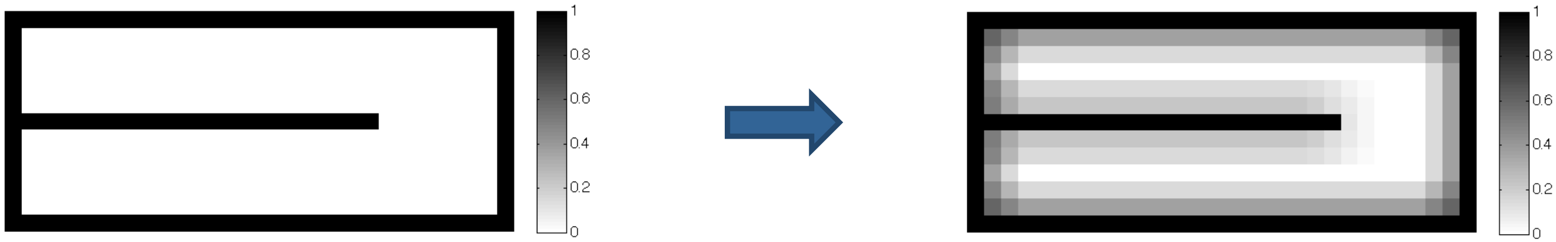
Problems on A*/D* on Grids

3. When the path turns out to be blocked during traversal, it needs to be **re-planned from scratch**
   - In unknown or dynamic environments, this can occur very often
   - Replanning in large state spaces is costly
   - Can we re-use (repair) the initial plan?

Let's look at solutions to these problems…

Map Smoothing

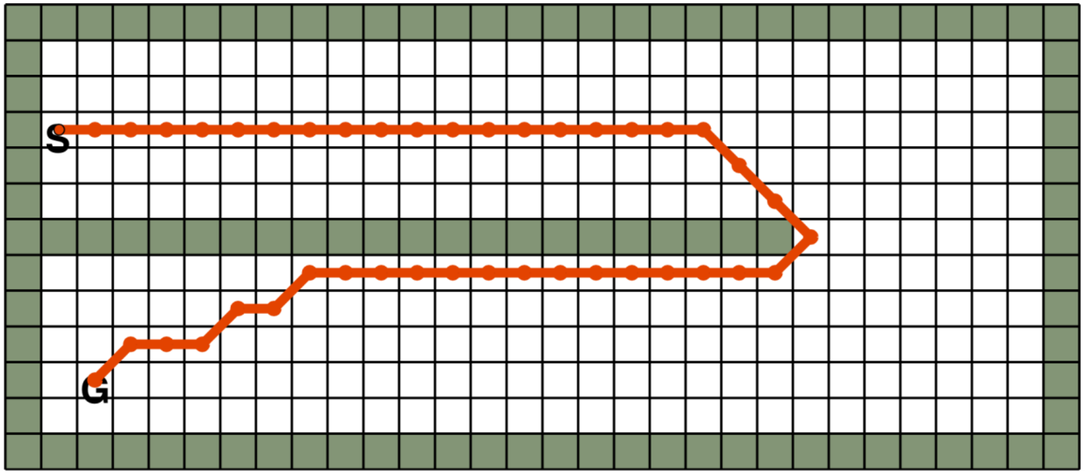
- **Problem:** Path gets close to obstacles
- **Solution:** Convolve the map with a kernel (e.g., Gaussian)

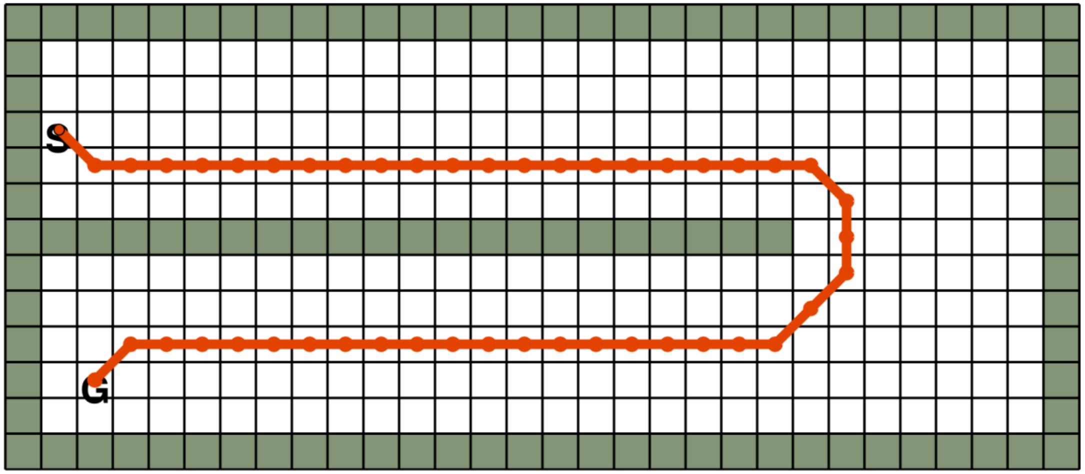


- Leads to non-zero probability around obstacles
- Evaluation function

$$f(n) = g(s) \cdot p_{\text{occ}}(s) + h(s)$$

Example: Map Smoothing





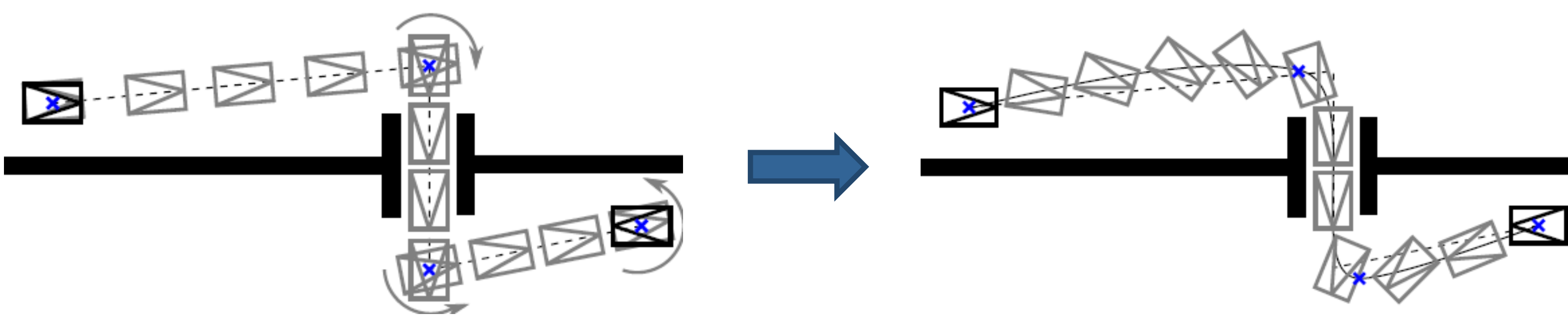
Path Smoothing

- **Problem:** Paths are aligned to grid structure (because they have to lie in the roadmap)
- Paths look unnatural and are sub-optimal
- **Solution:** Smooth the path after generation
  - Traverse path and find pairs of nodes with direct line of sight; replace by line segment
  - Refine initial path using non-linear minimization (e.g., optimize for continuity/energy/execution time)
  - …

Example: Path Smoothing

- Replace pairs of nodes by line segments

- Non-linear optimization



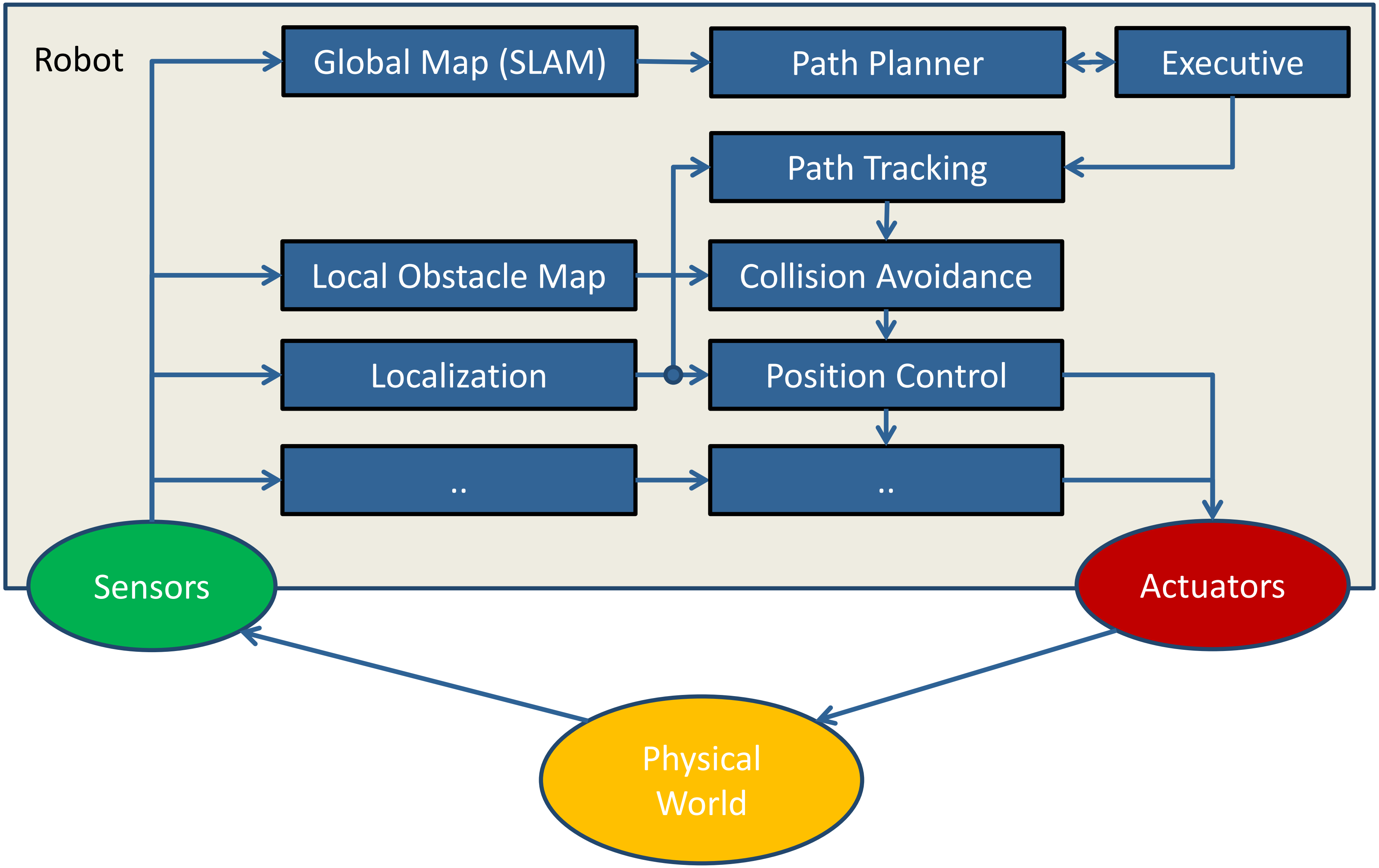
Real-Time Motion Planning

- What is the maximum time needed to re-plan in case of an obstacle detection?
- What if the robot has to react quickly to unforeseen, fast moving objects?
- Do we really need to re-plan for every obstacle on the way?

Real-Time Motion Planning

- What is the maximum time needed to re-plan in case of an obstacle detection?
  In principle, re-planning with D* can take arbitrarily long
- What if the robot has to react quickly to unforeseen, fast moving objects?
  Need a collision avoidance algorithm that runs in constant time!
- Do we really need to re-plan for every obstacle on the way?
  Could trigger re-planning only if path gets obstructed (or robot predicts that re-planning reduces path length by p%)

Robot Architecture



Layered Motion Planning

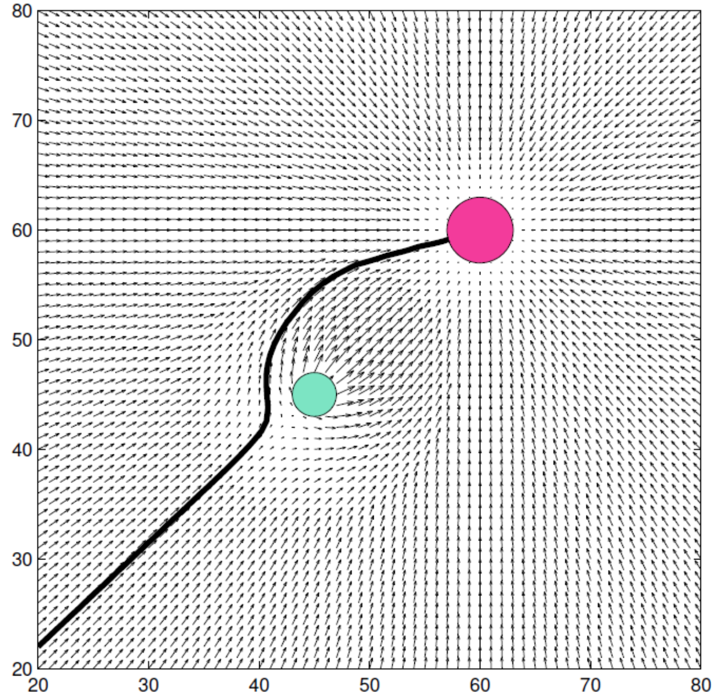
- An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate **local planner** accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)

Local Planner

- **Given:** Path to goal (sequence of via points), range scan of the local vicinity, dynamic constraints
- **Wanted:** Collision-free, safe, dynamically feasible, and fast motion towards the goal (or next via point)
- Typical approaches:
  - Potential fields
  - Dynamic window approach

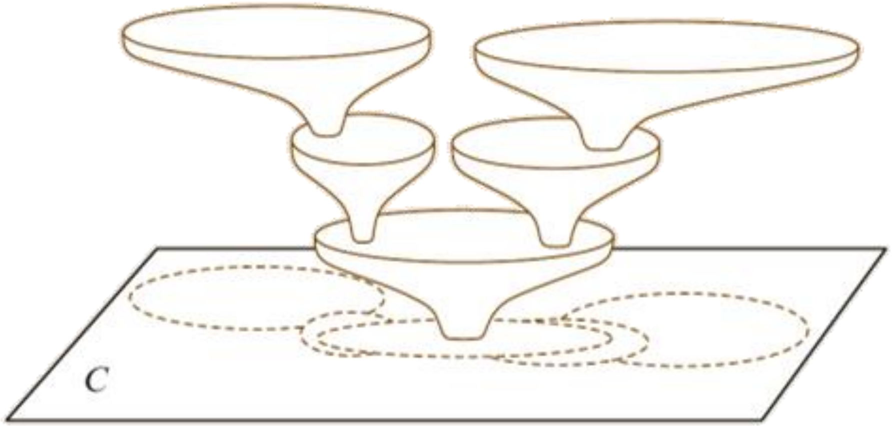
Navigation with Potential Fields

- Treat robot as a particle under the influence of a potential field
- **Pro:**
  - Easy to implement
- **Con:**
  - Suffers from local minima
  - No consideration of dynamic constraints



Navigation with Funnels
[Choi and Latombe, IROS 1991]

- Different regions of the configuration space need different potential fields
- Compose navigation function from overlapping local potential functions (the so-called **funnels**)



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

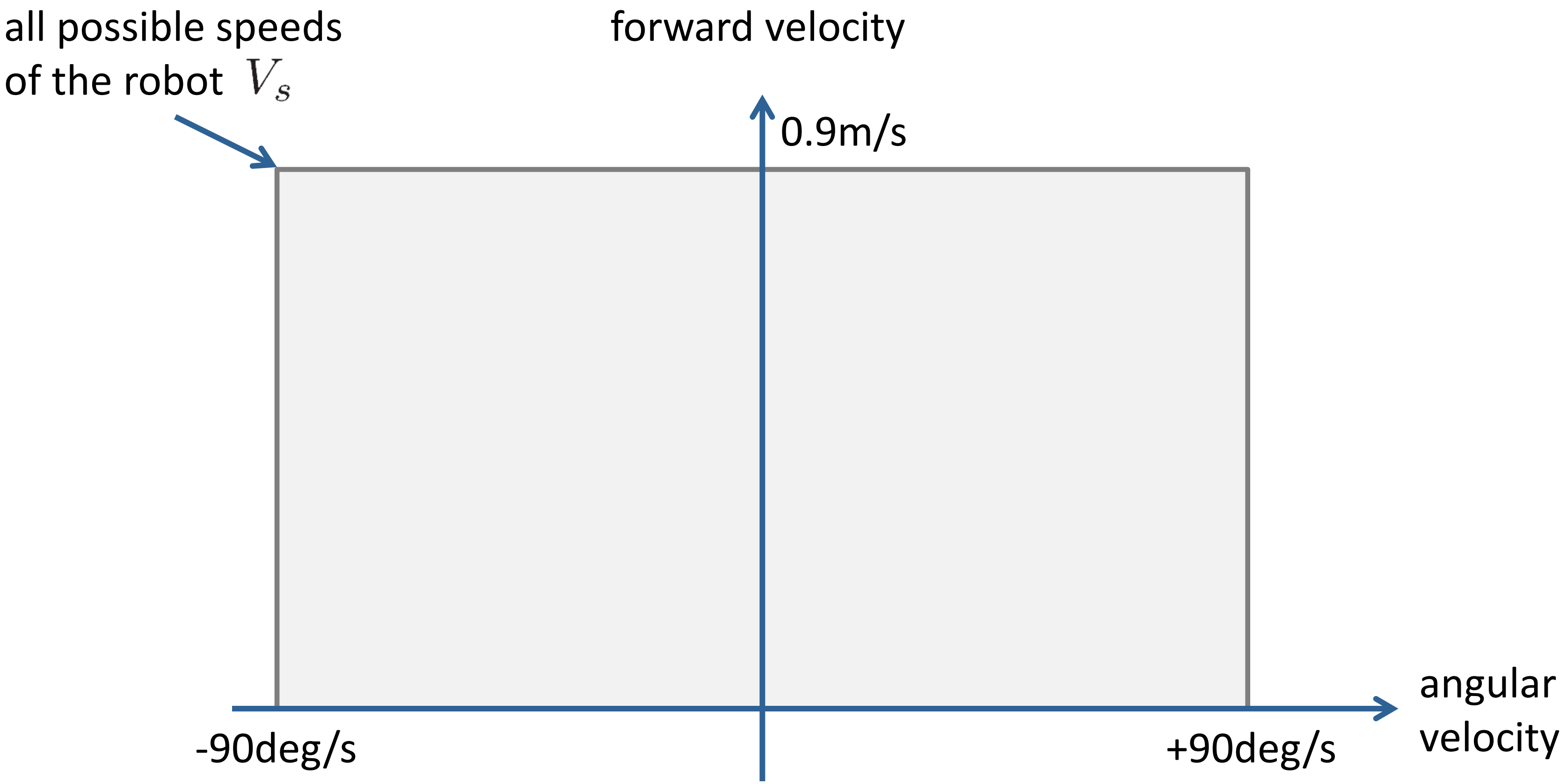
Algorithm:

1. Sample the robot’s control space
2. Simulate each sample for a short period of time
3. Score each sample based on
   - proximity to obstacles
   - proximity to goal
   - proximity to global path
   - speed
4. Pick the highest-scoring control command

Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

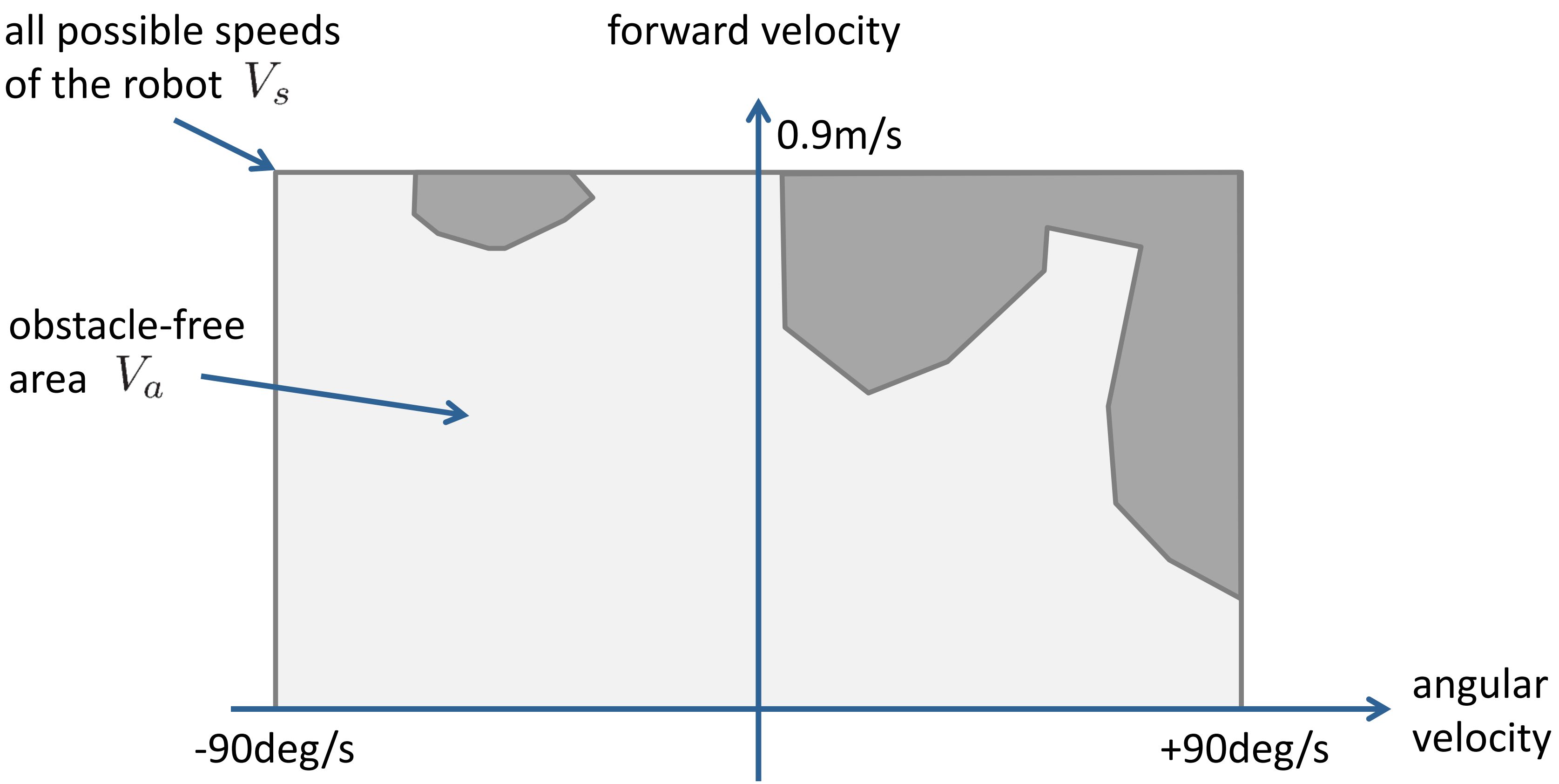
- Consider a 2DOF planar robot



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

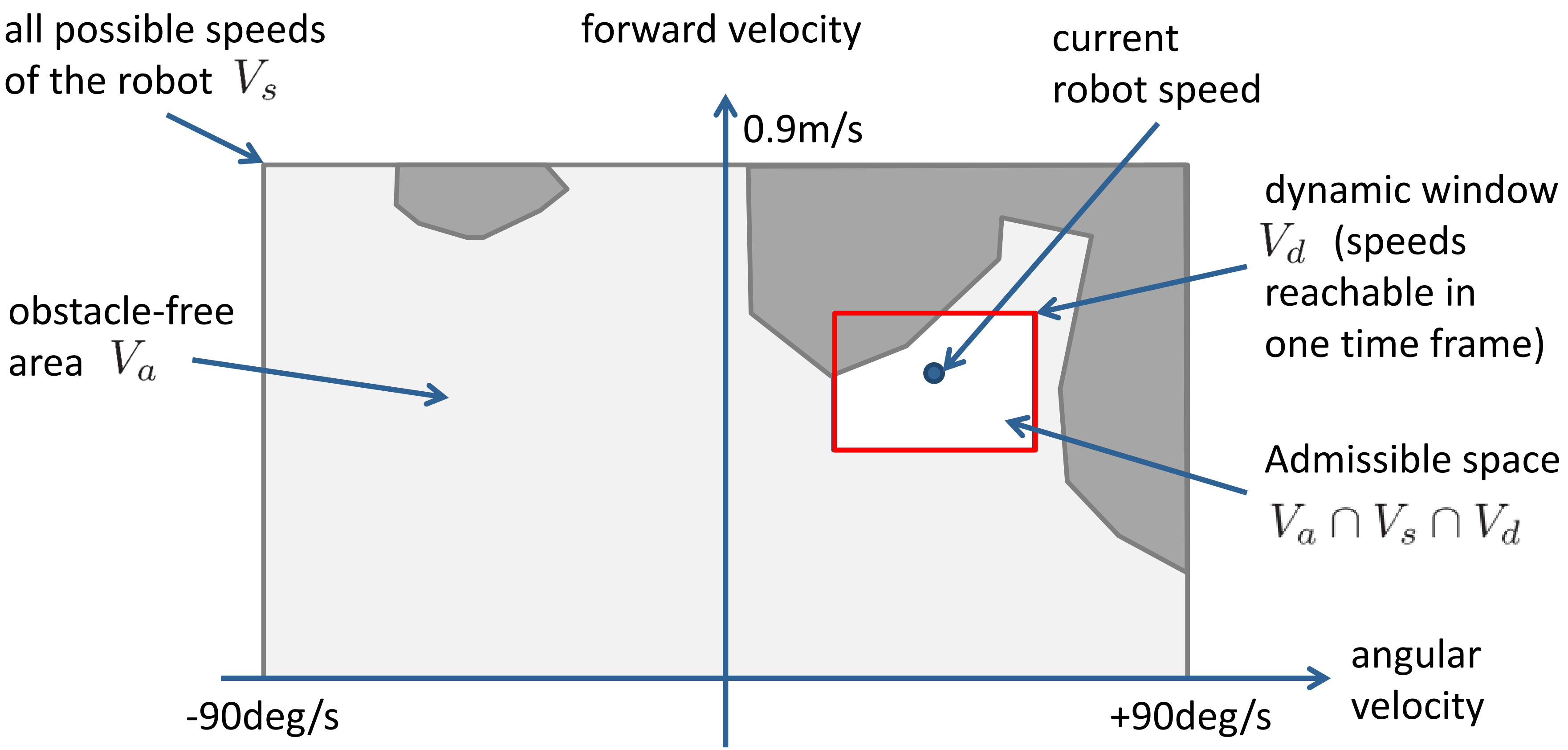
- Consider a 2DOF planar robot + 2D environment



Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Consider additionally dynamic constraints



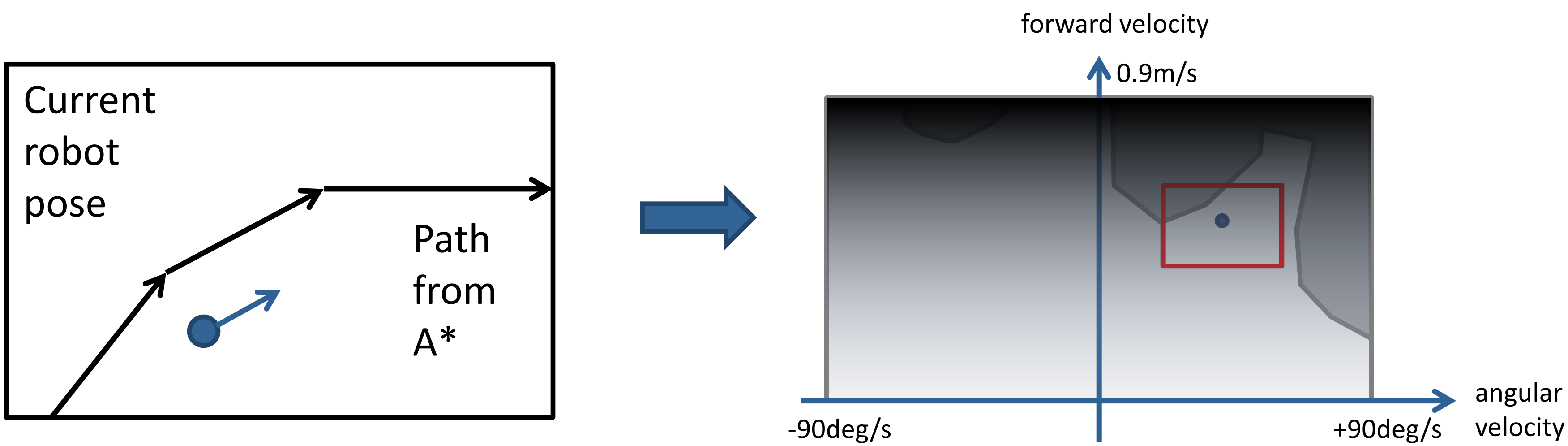
Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Navigation function (potential field)

$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity



Dynamic Window Approach

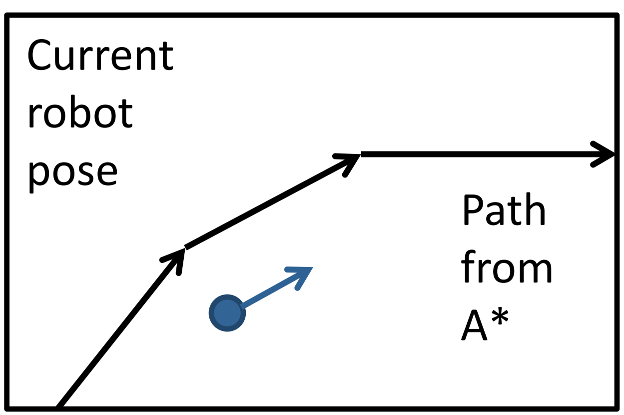
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

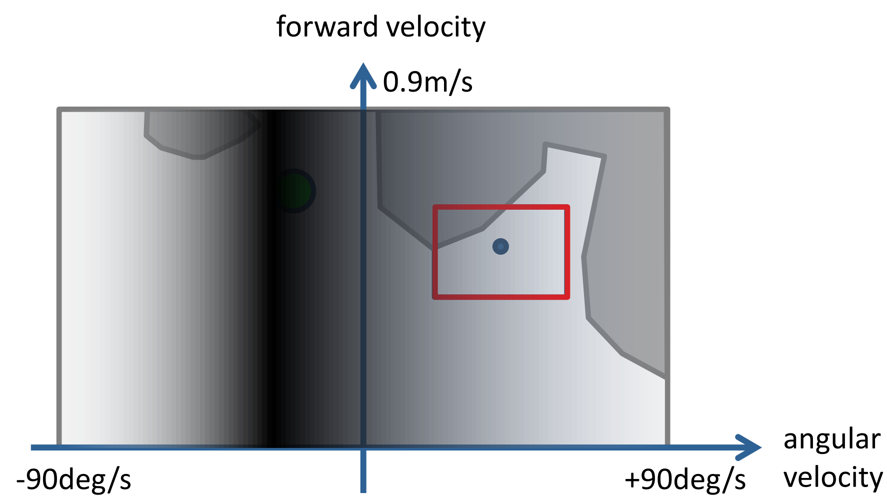
- Navigation function (potential field)

$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity

Rewards alignment to A* path gradient





Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

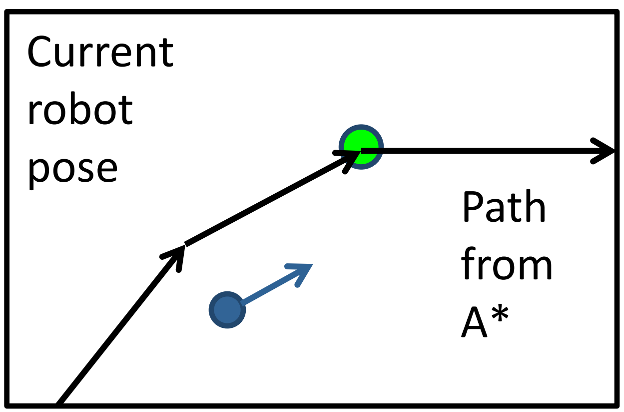
- Navigation function (potential field)

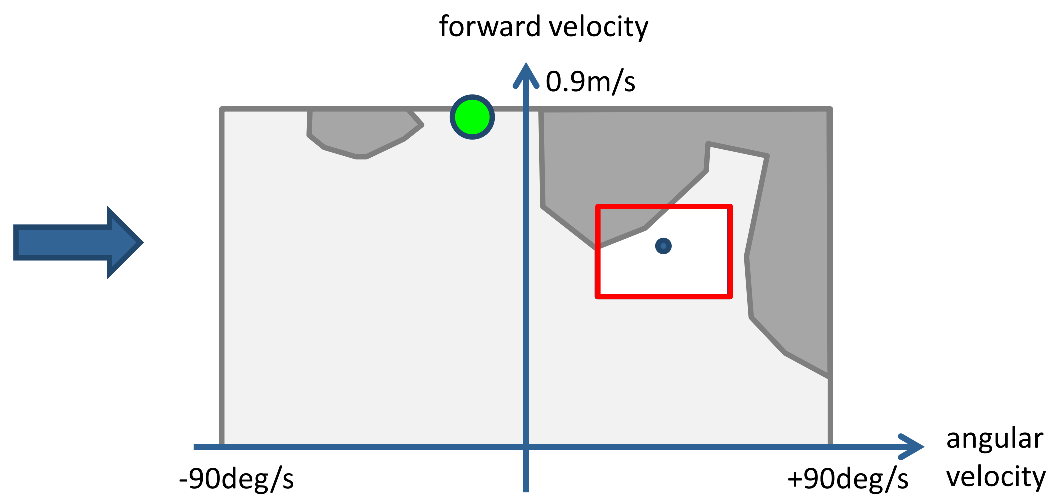
$$f(n) = \alpha \cdot vel + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot goal$$

Maximizes velocity

Rewards alignment to A* path gradient

Rewards large advances on A* path

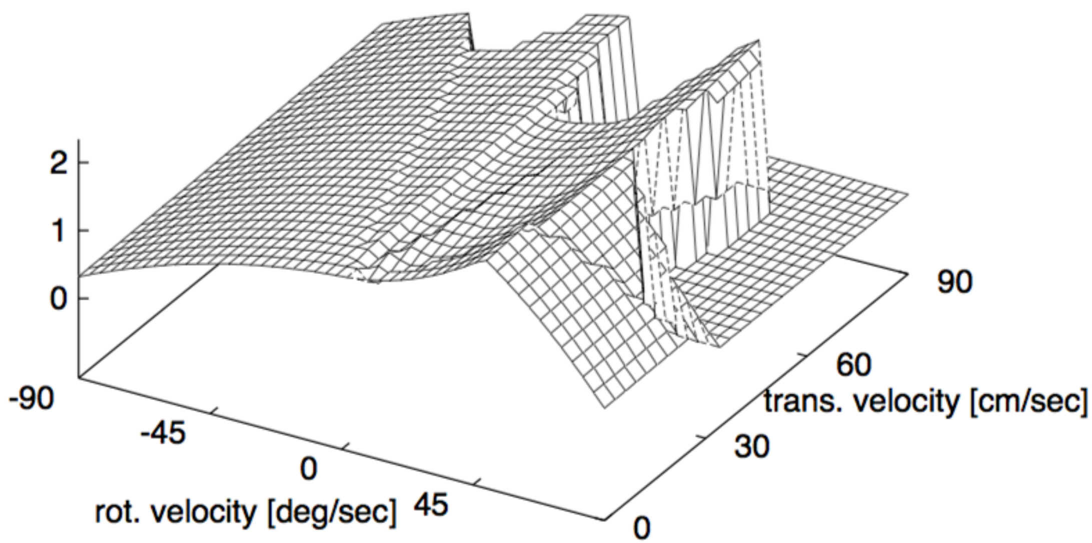




Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Discretize dynamic window and evaluate navigation function (note: window has fixed size = real-time!)
- Find the maximum and execute motion



Example: Dynamic Window Approach

[Brock and Khatib, ICRA ‘99]

Problems of DWAs

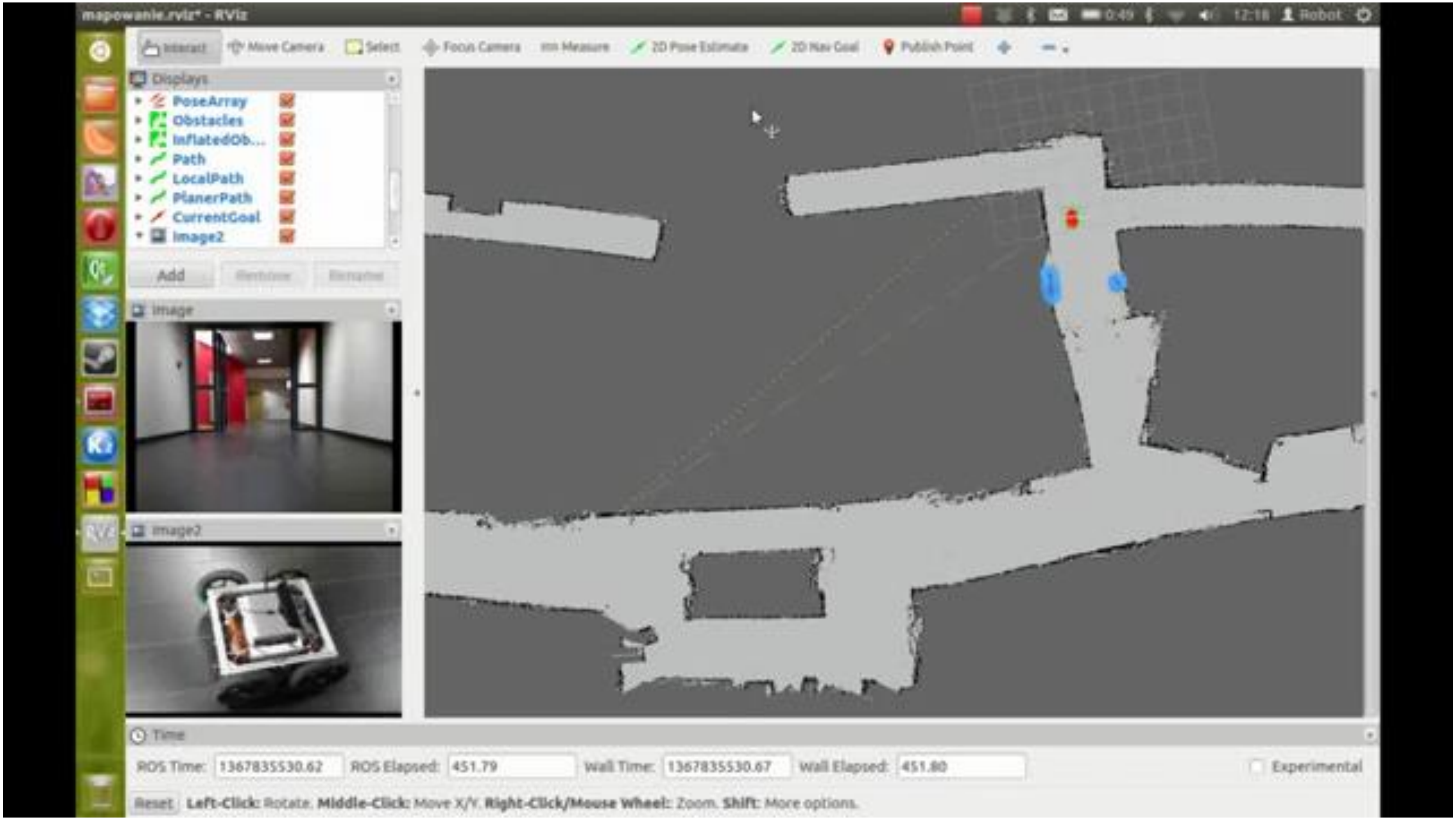
- DWAs suffer from local minima (need tuning), e.g., robot does not slow down early enough to enter doorway:

- Can you think of a solution?
- **Note:** General case requires global planning

Example: Motion Planning in ROS

- Executive: state machine (**`move_base`**)
- Global costmap: grid with inflation (**`costmap_2d`**)
- Global path planner: Dijkstra (Dijkstra, **`navfn`**)
- Local costmap (**`costmap_2d`**)
- Local planner: Dynamic window approach (**`base_local_planner`**)

Example: Motion Planning in ROS



Lessons Learned Today

- How to sample roadmaps and probabilistic random trees
- How to efficiently compute a path between the start and goal node
- How to update plan efficiently
- How to follow and execute a path in real-time