Visual Navigation for Flying Robots

Motion Planning

Dr. Jürgen Sturm

in.tum.summer party & career forum

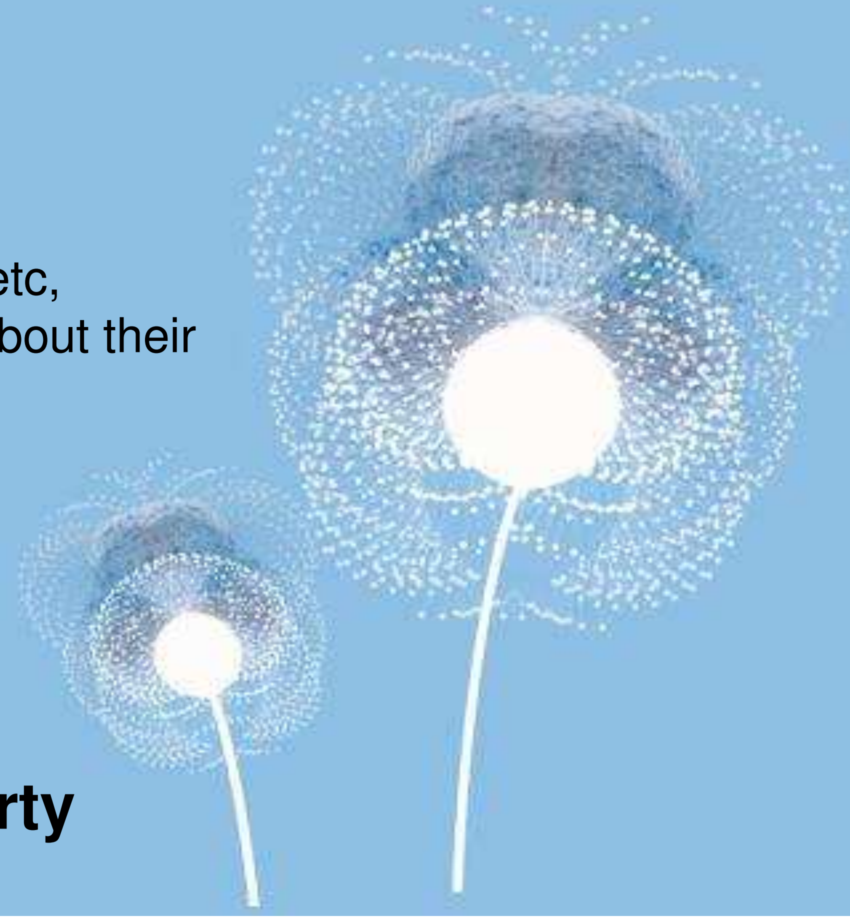
The Department of Informatics would like to invite its students and employees to its summer party and career forum.

July 4, 2012

3 pm – 6 pm **Career Forum**:
Presentations given by Google, Capgemini etc, stands, panel discussion: TUM alumni talk about their career paths in informatics

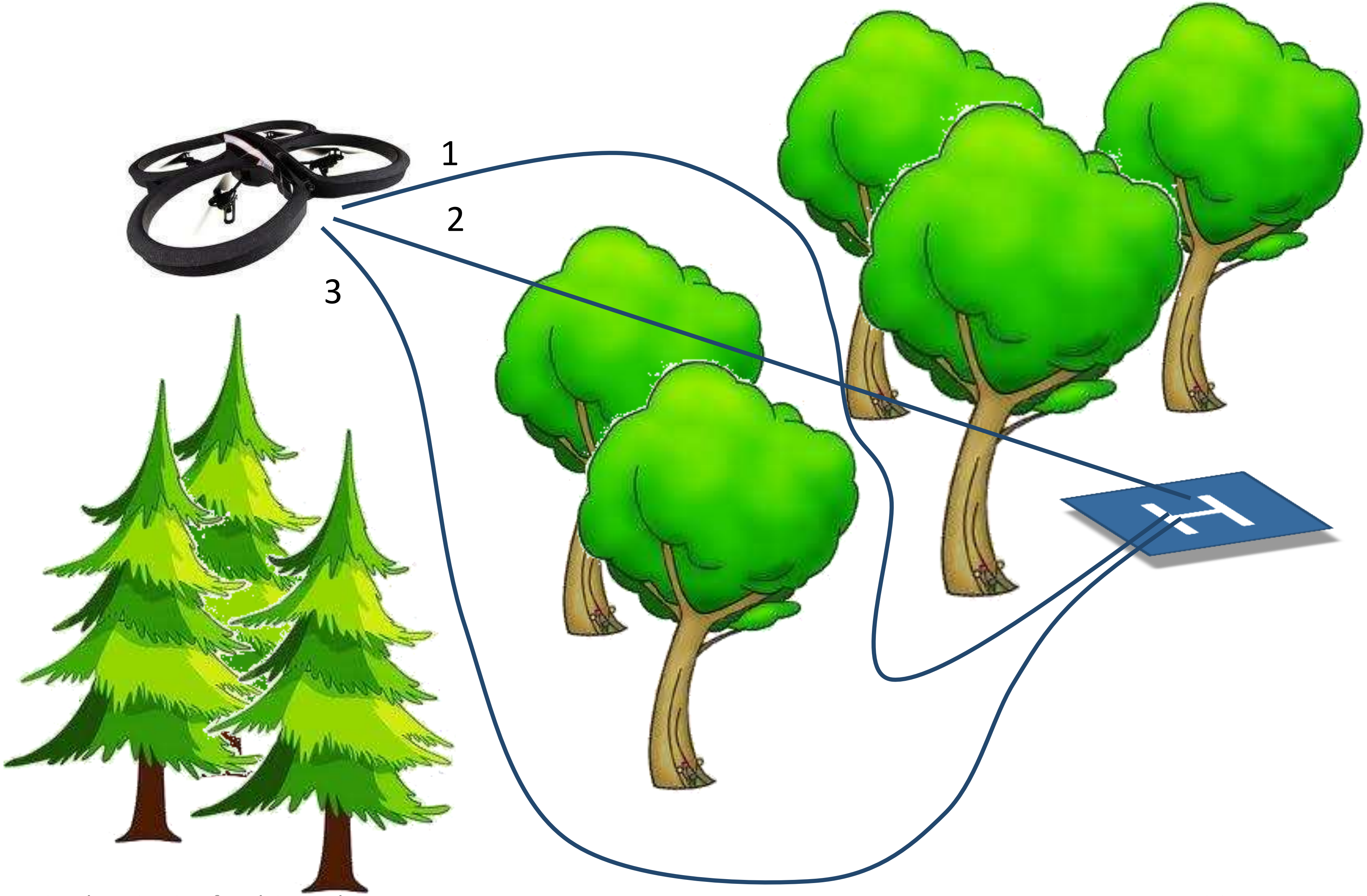
3 pm – 6 pm **Foosball Tournament**

Starting at 5 pm **Summer Party**:
BBQ, live band and lots of fun!



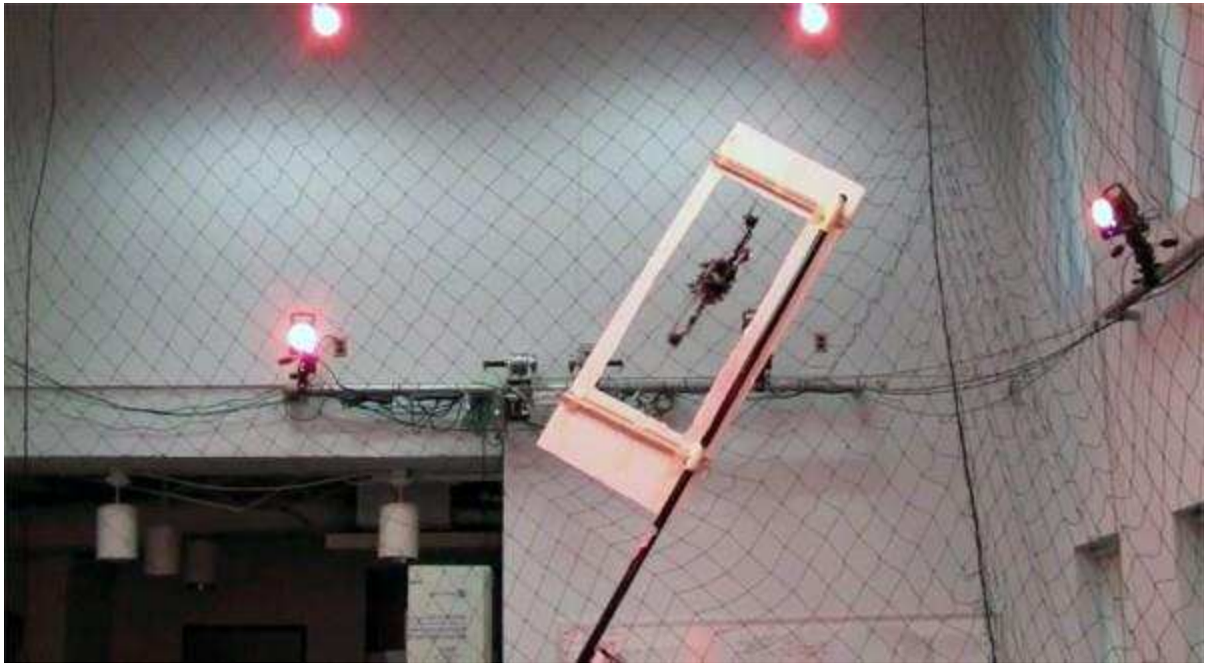
www.in.tum.de/2012summerparty

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?

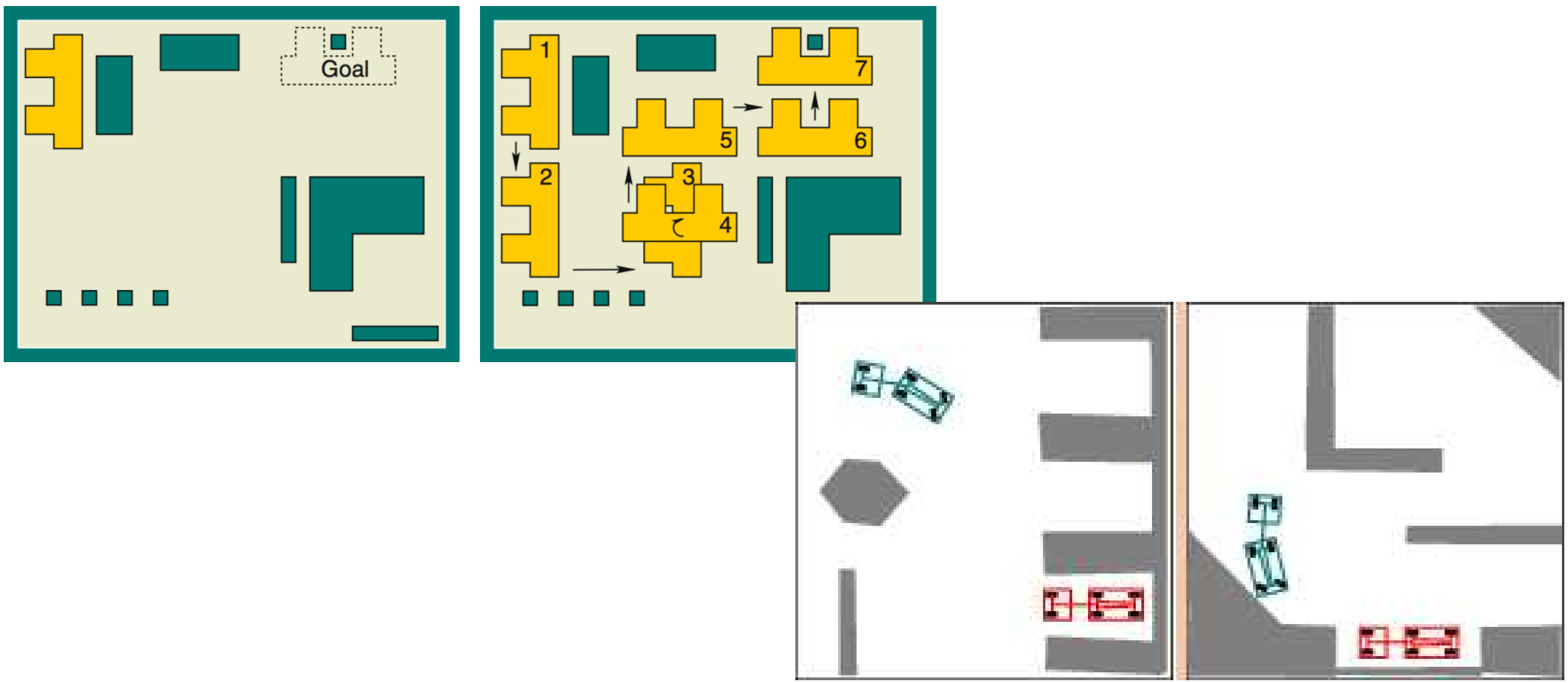
Motion Planning Problem

What are good performance metrics?

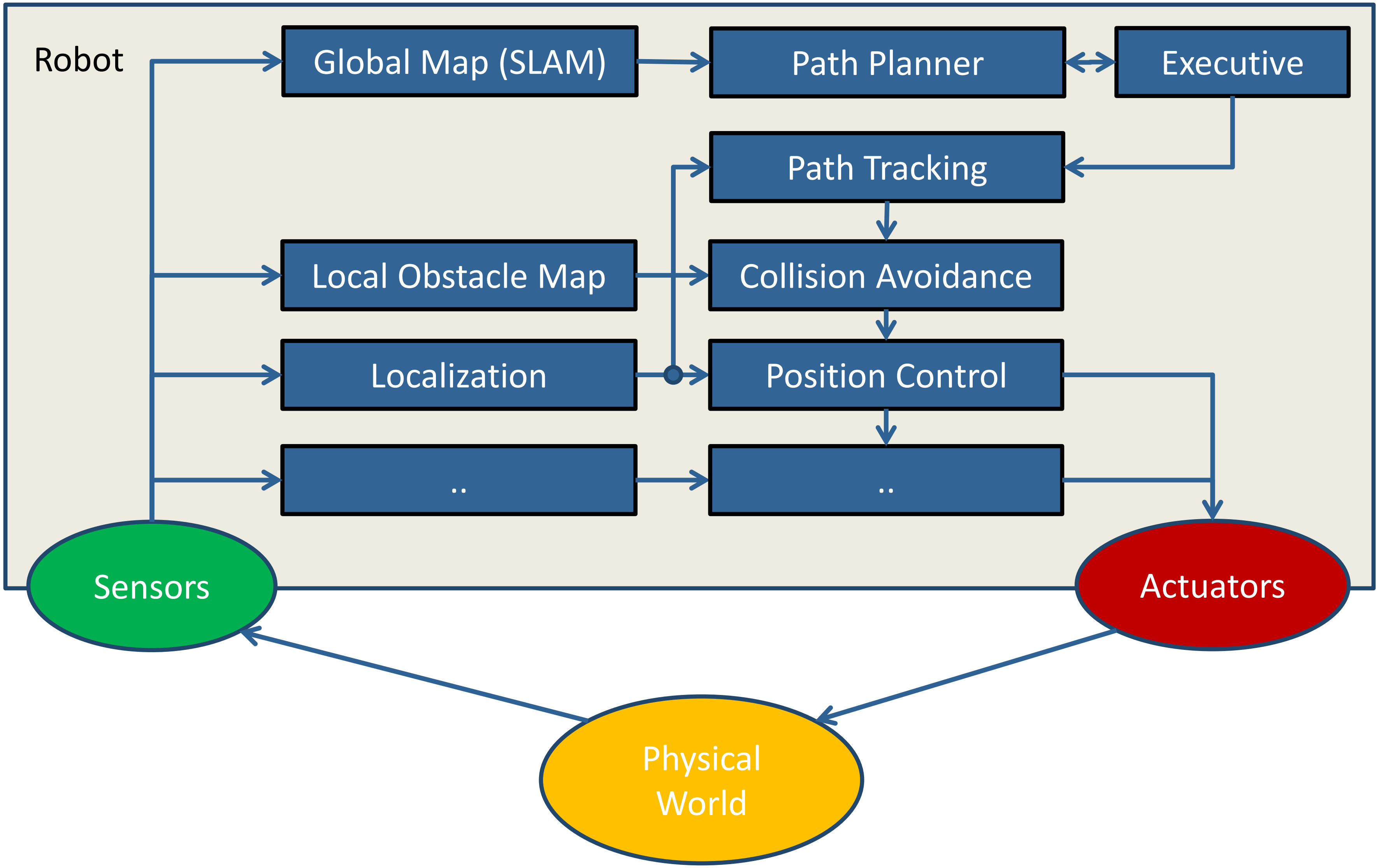
- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- …

Motion Planning Examples

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



Robot Architecture



Agenda for Today

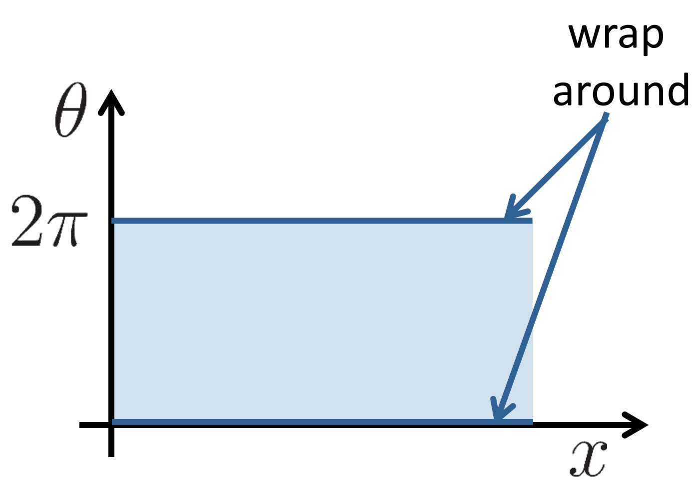
- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

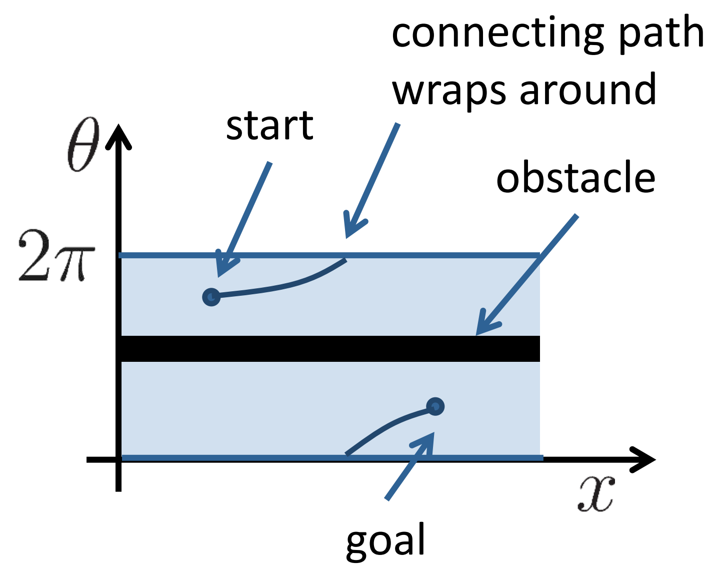
Configuration Space

- Work space
  - Typically 3D pose (position + orientation) → 6 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}

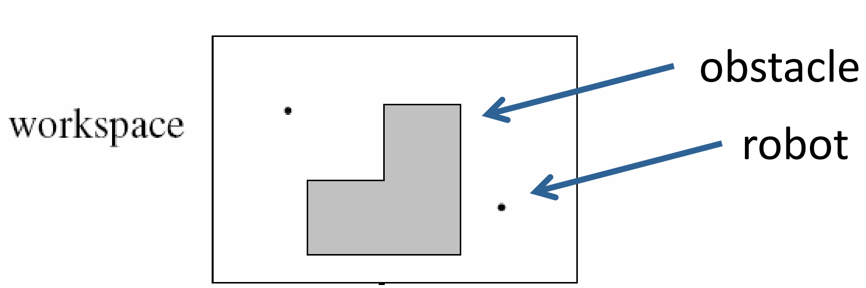
- Properties

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

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

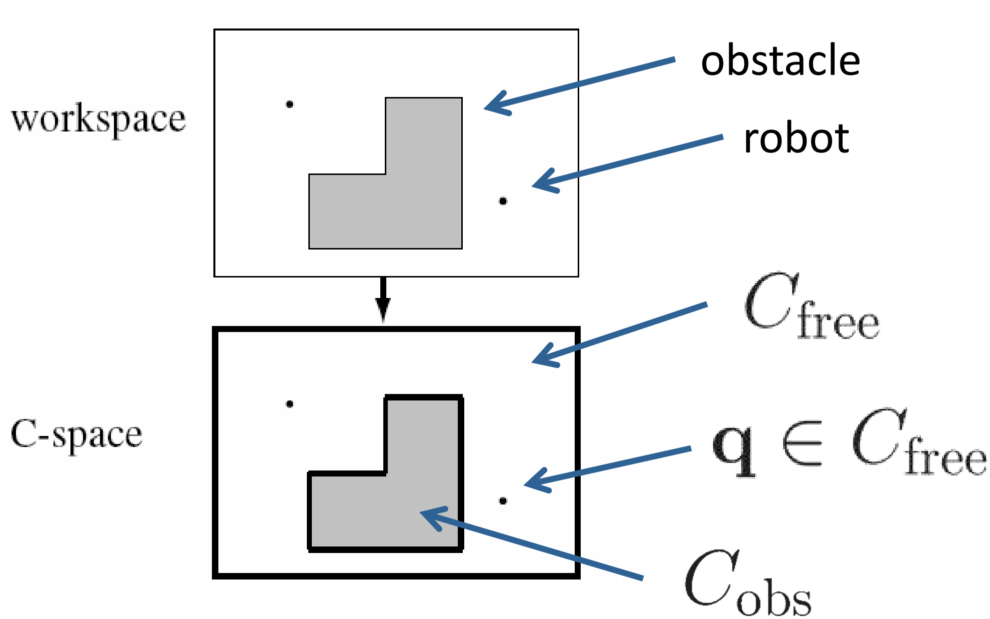
Free Space Example

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



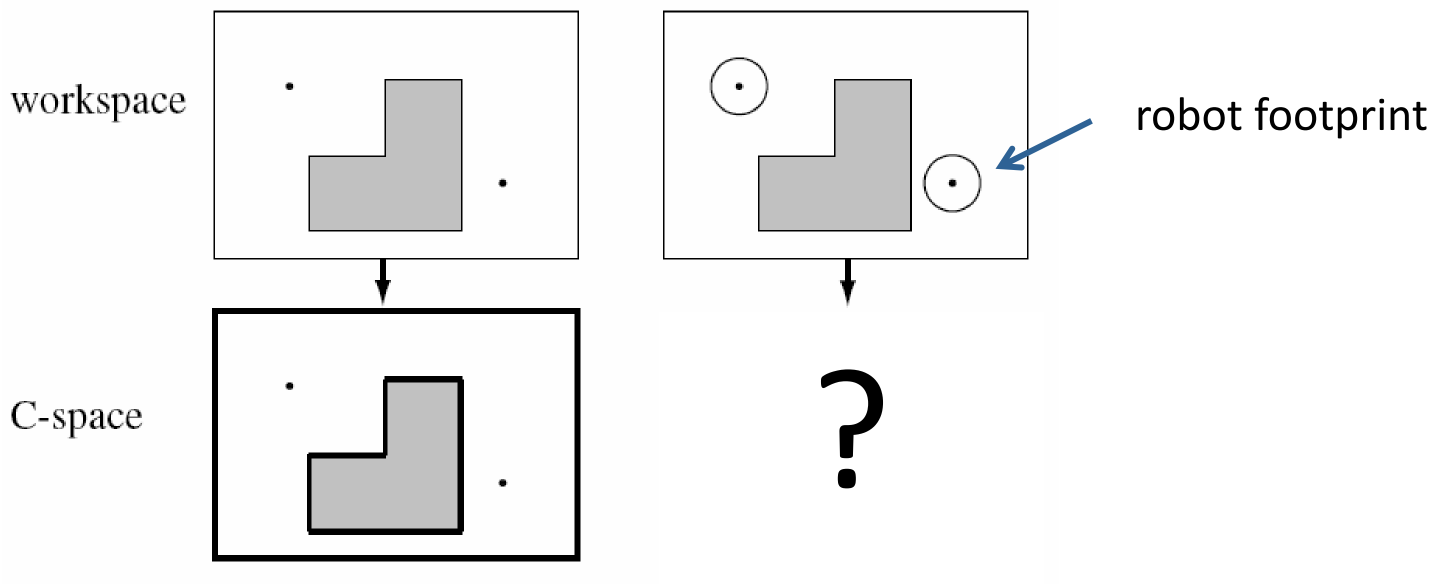
Example

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



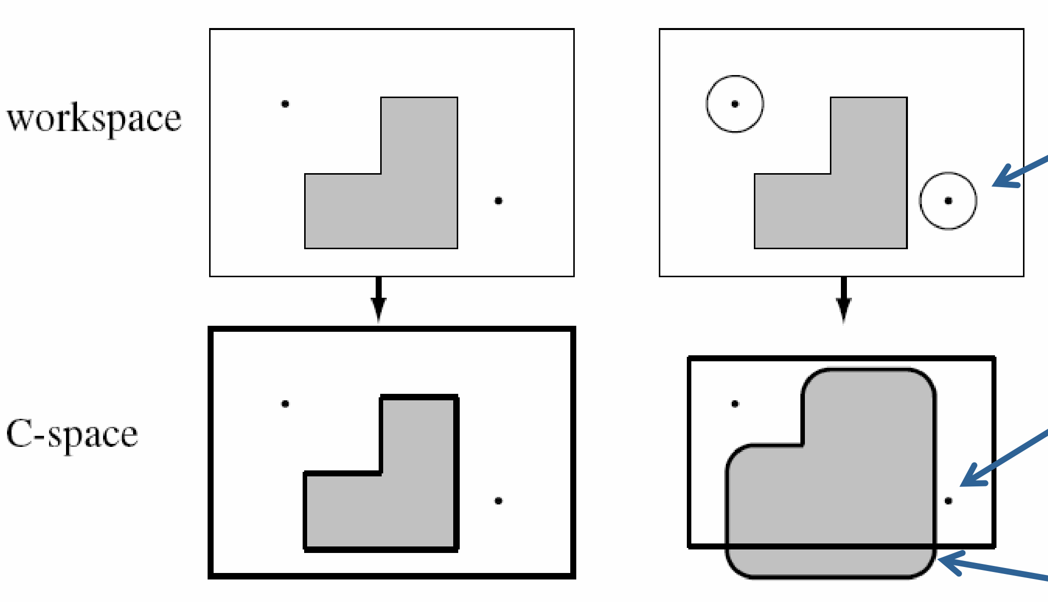
Example

- 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



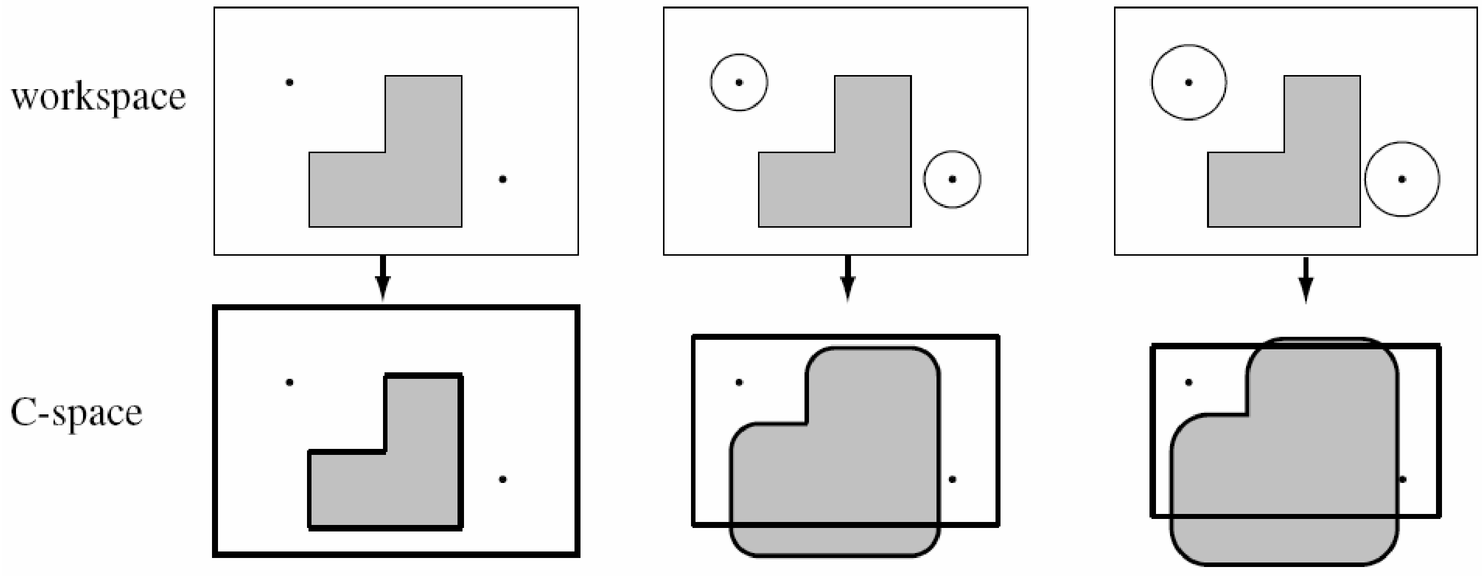
robot footprint in work space (disk)

robot footprint in configuration space (point)

obstacle in configuration space

Example

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



Computing the Free Space

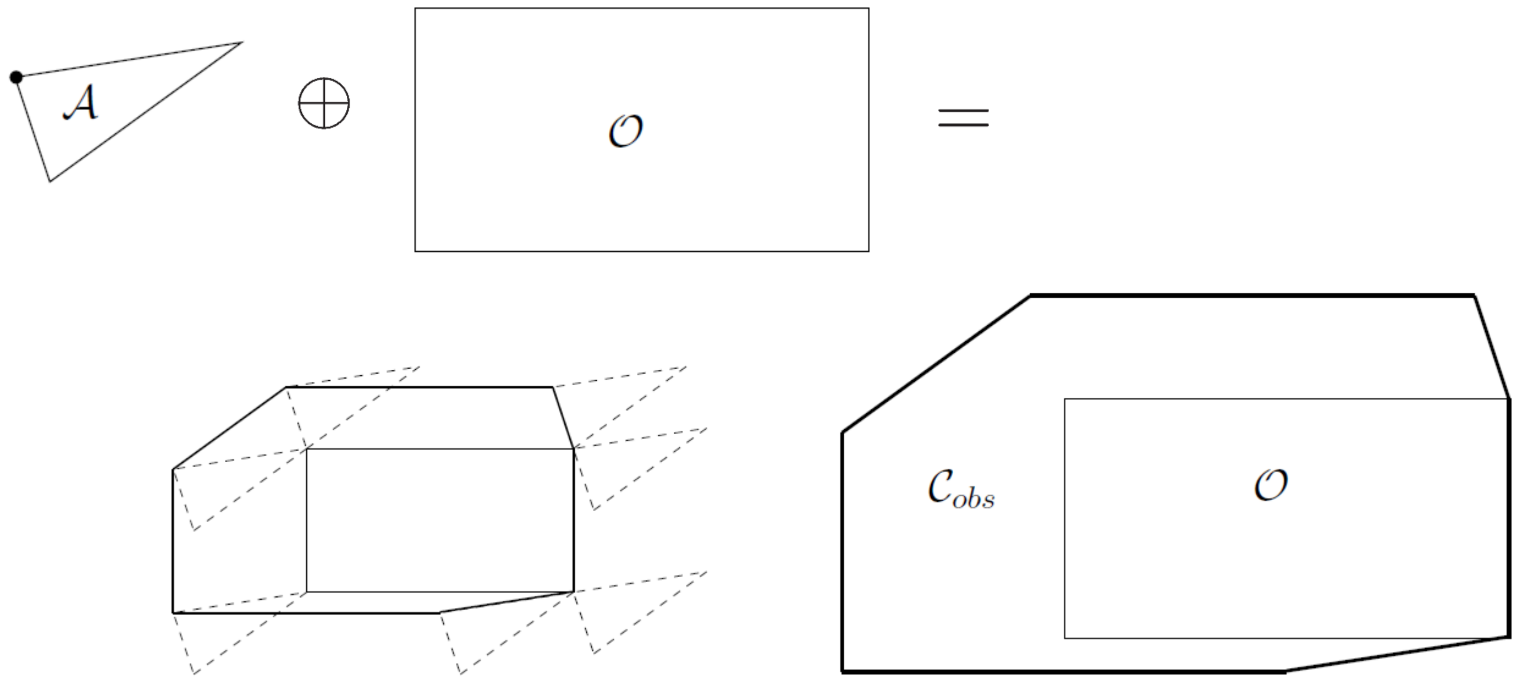
- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius
- This operation is called the **Minowski sum**

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

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

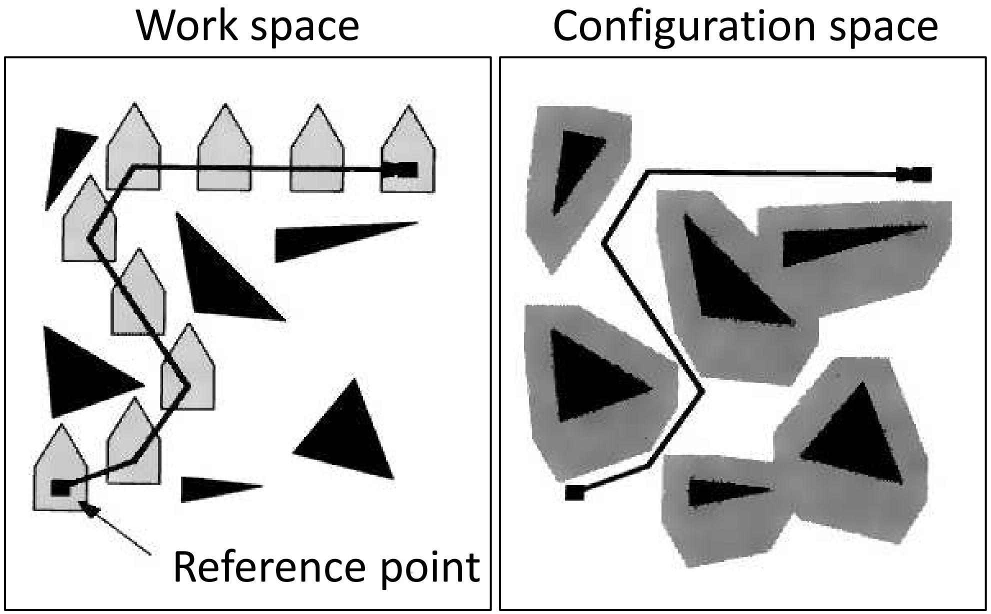
Example: Minowski Sum

- Triangular robot and rectangular obstacle



Example

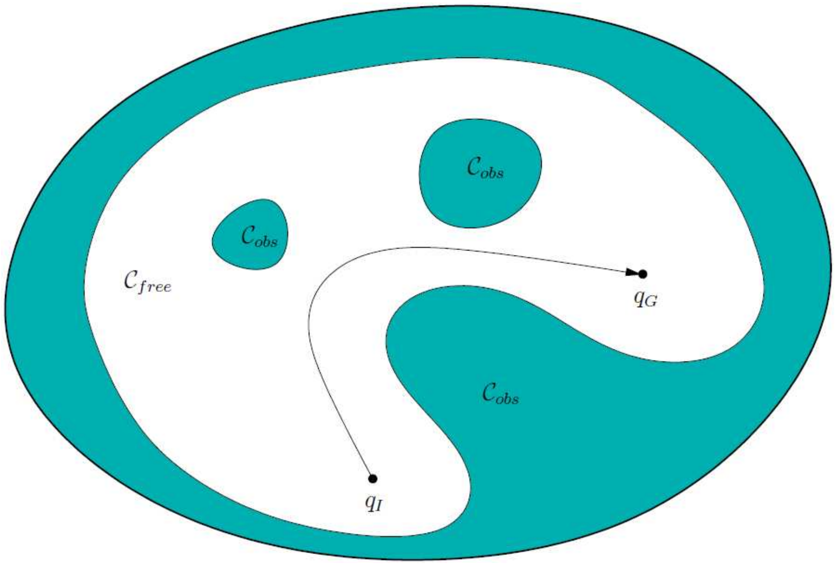
- Polygonal robot, translation only



- 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

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

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

Two competing paradigms

- **Combinatorial planning**
  (exact planning)
- **Sampling-based planning**
  (probabilistic/randomized planning)

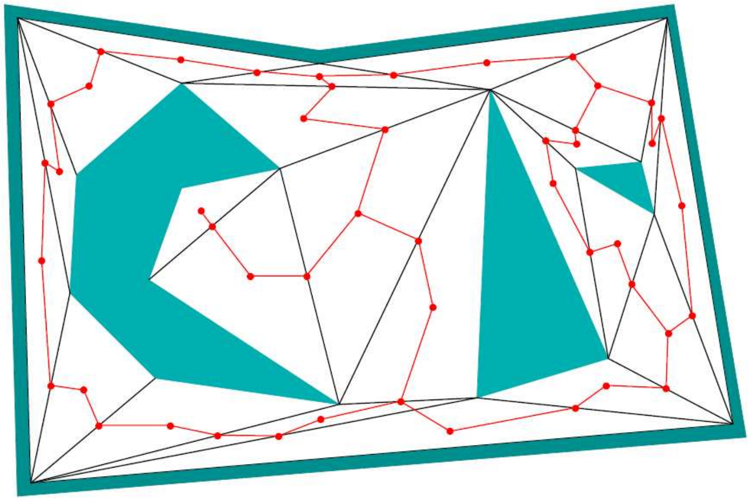
Combinatorial Methods

- Mostly developed in the 1980s
- Extremely efficient for low-dimensional problems
- Sometimes difficult to implement
- Usually produce a road map in C_{free}
- Assume polygonal environments

Roadmaps

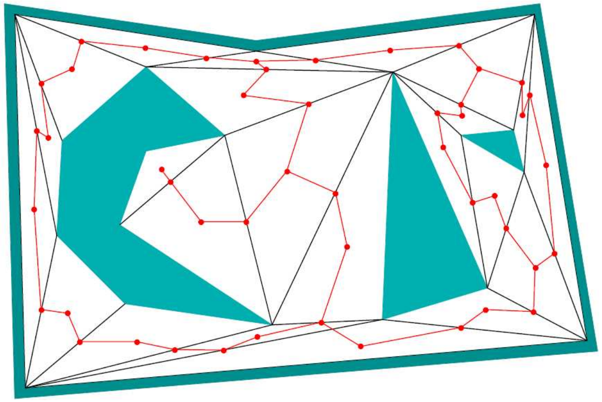
A **roadmap** is a graph in C_{free} where

- Each vertex is a configuration $\mathbf{q} \in C_{\text{free}}$
- Each edge is a path $\tau : [0,1] \to C_{\text{free}}$ for which $\tau(0)$ and $\tau(1)$ are vertices



(Desired) Properties of Roadmaps

- **Accessibility**
  From anywhere in C_{free}, it is easy to compute a path that reaches at least one of the vertices
- **Connectivity-preserving**
  If there exists a path between $\mathbf{q}_I$ and $\mathbf{q}_G$ in C_{free} then there must also exist a path in the road map



Roadmap Construction

We consider here three **combinatorial** methods:

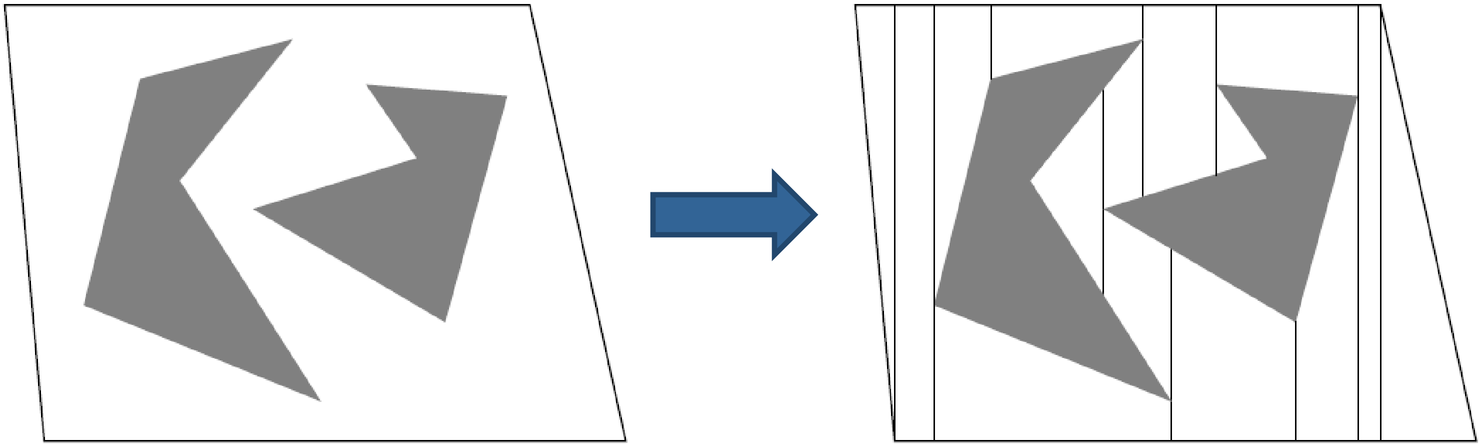
- Trapezoidal decomposition
- Shortest path roadmap
- Regular grid
- … but there are many more!

Afterwards, we consider two **sampling-based** methods:

- Probabilistic roadmaps (PRMs)
- Rapidly exploring random trees (RRTs)

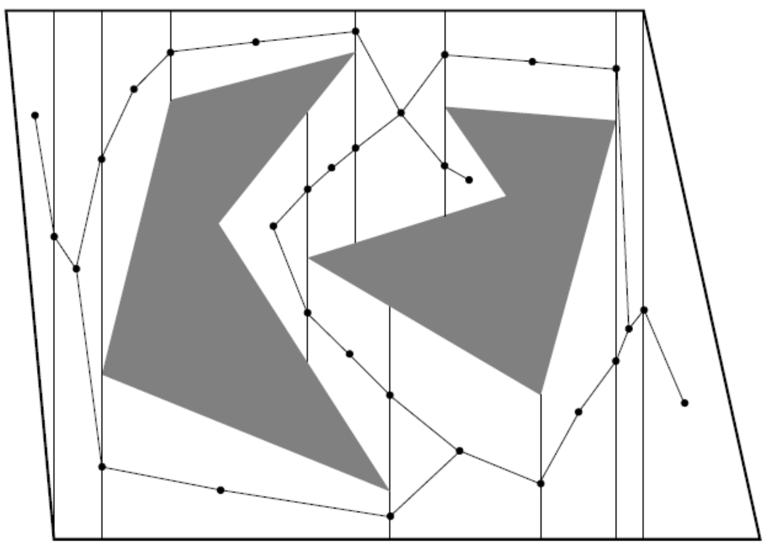
Roadmap Construction

- Decompose horizontally in convex regions using plane sweep
- Sort vertices in x direction. Iterate over vertices while maintaining a vertically sorted list of edges



Roadmap Construction

- Place vertices
  - in the center of each trapezoid
  - on the edge between two neighboring trapezoids
- Resulting road map

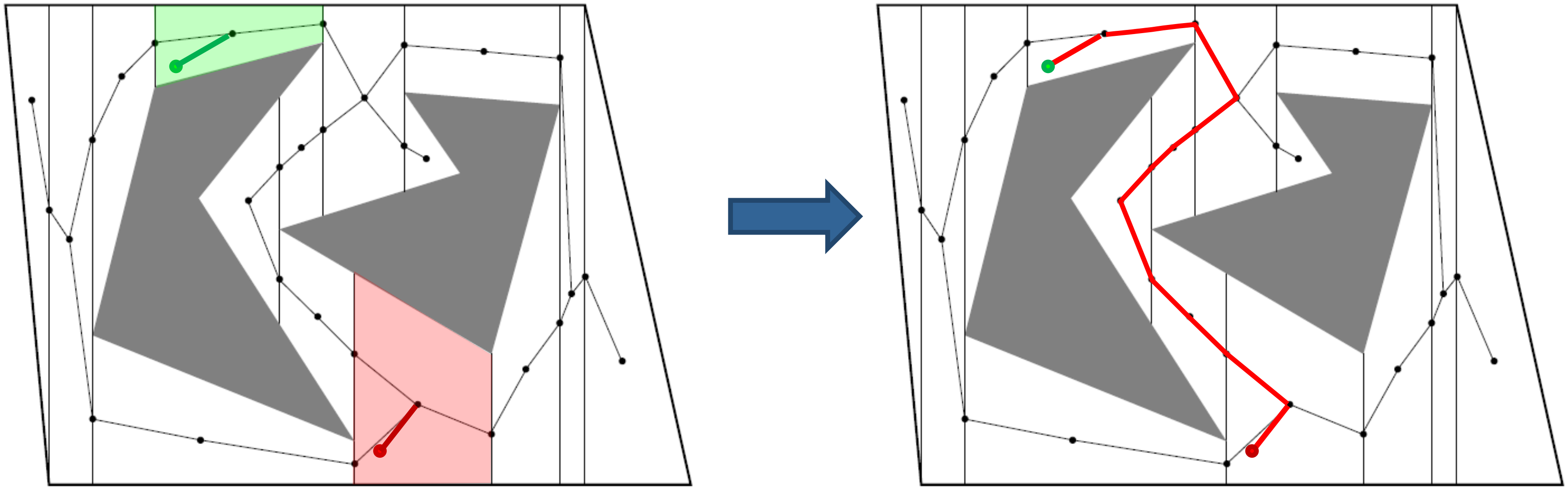


Quick check on properties:
- Accessibility
- Connectivity-preserving?

Example Query

Compute path from $\mathbf{q}_I$ to $\mathbf{q}_G$

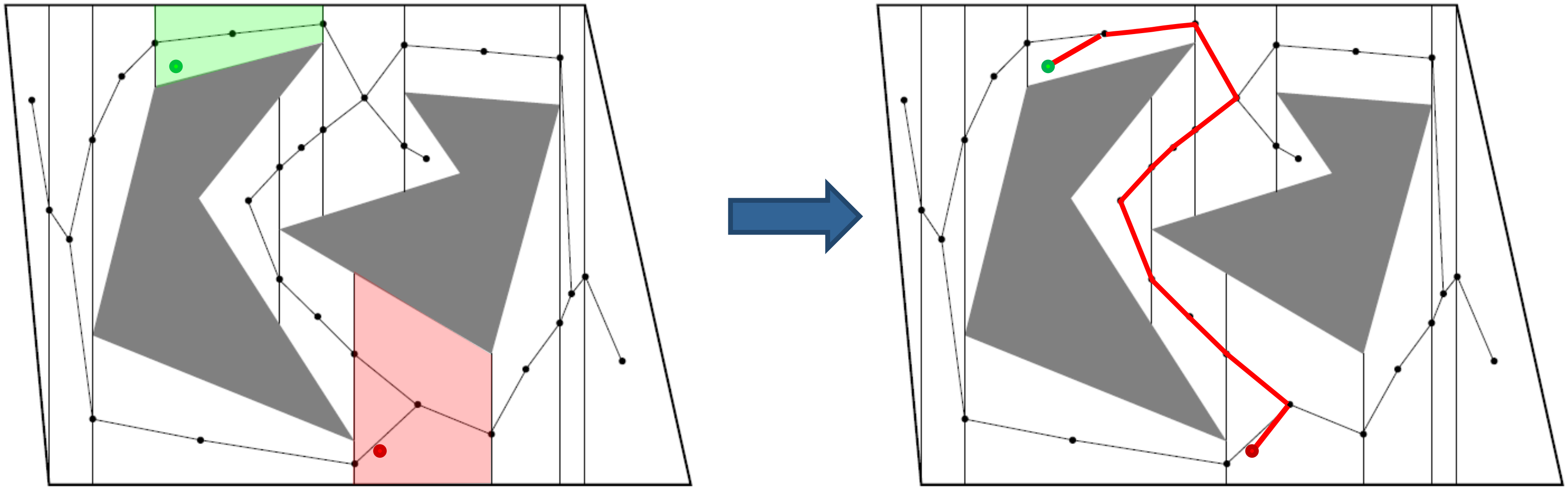
- Identify **start** and **goal** trapezoid
- Connect **start** and **goal** location to center vertex
- Run search algorithm (e.g., Dijkstra)



Properties of Trapezoidal Decomposition

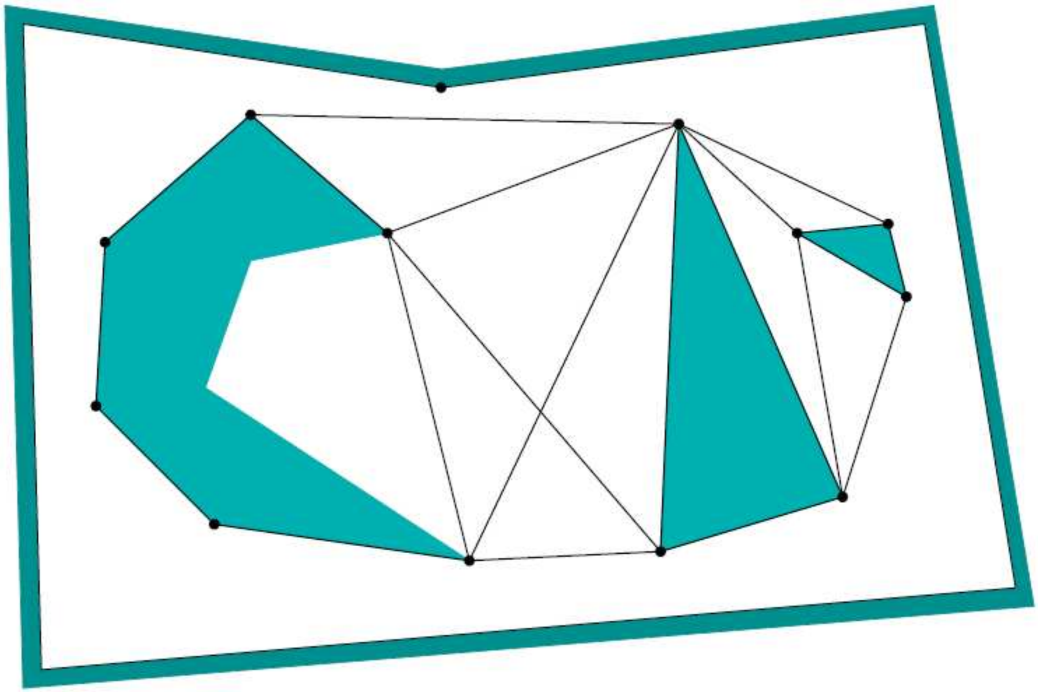
- + Easy to implement
- + Efficient computation
- + Scales to 3D

- \- Does not generate shortest path



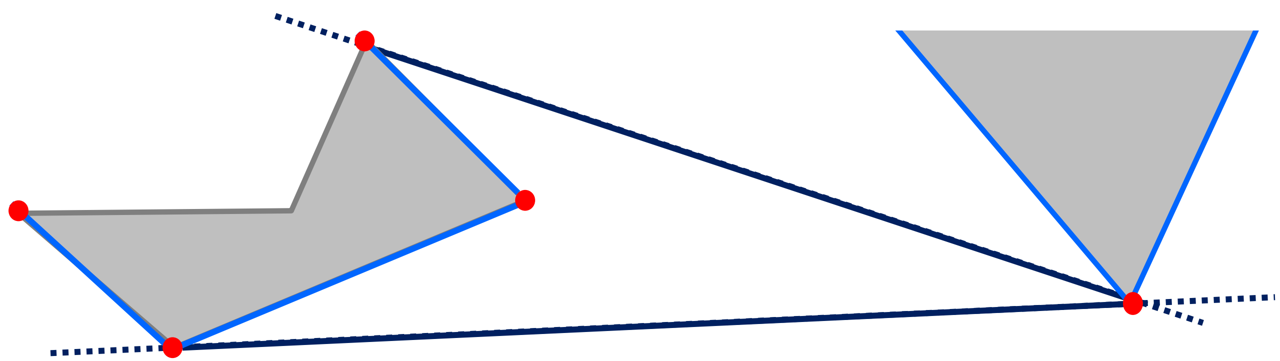
Shortest-Path Roadmap

- Contains all vertices and edges that optimal paths follow when obstructed
- Imagine pulling a tight string between $\mathbf{q}_I$ and $\mathbf{q}_G$



Roadmap Construction

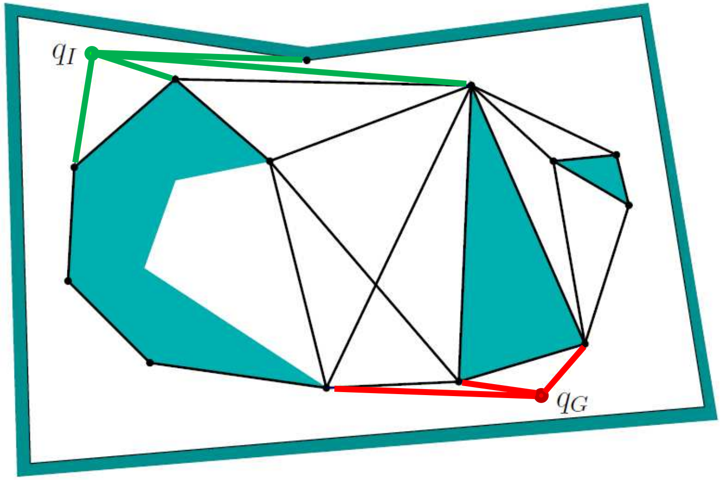
- Vertices = all sharp corners (>180deg, red)
- Edges
  1. Two consecutive sharp corners on the same obstacle (light blue)
  2. Bitangent edges (when line connecting two vertices extends into free space, dark blue)

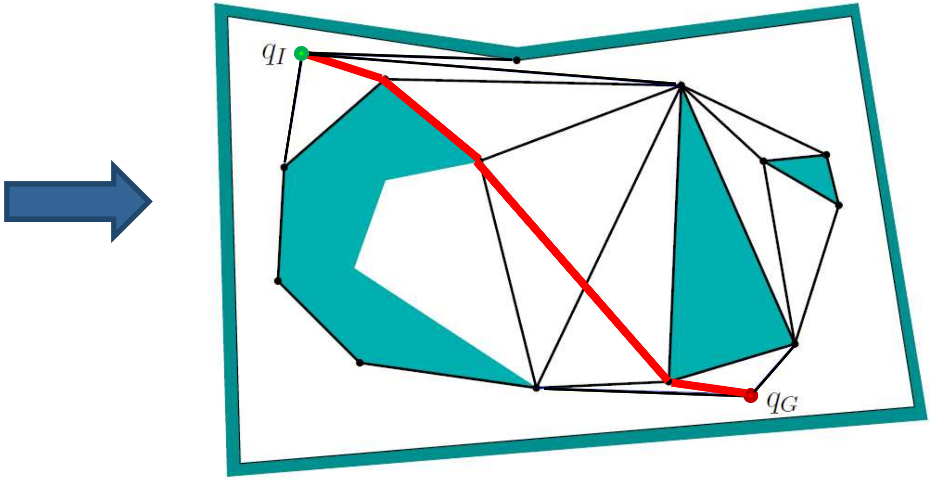


Example Query

Compute path from $\mathbf{q}_I$ to $\mathbf{q}_G$

- Connect **start** and **goal** location to all visible roadmap vertices
- Run search algorithm (e.g., Dijkstra)

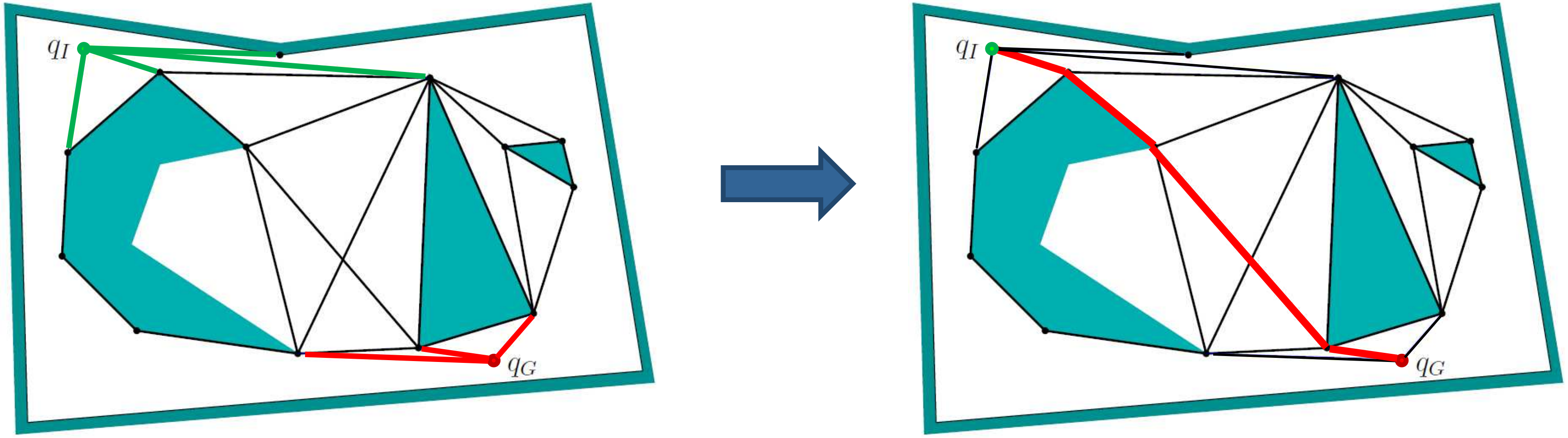




Example Query

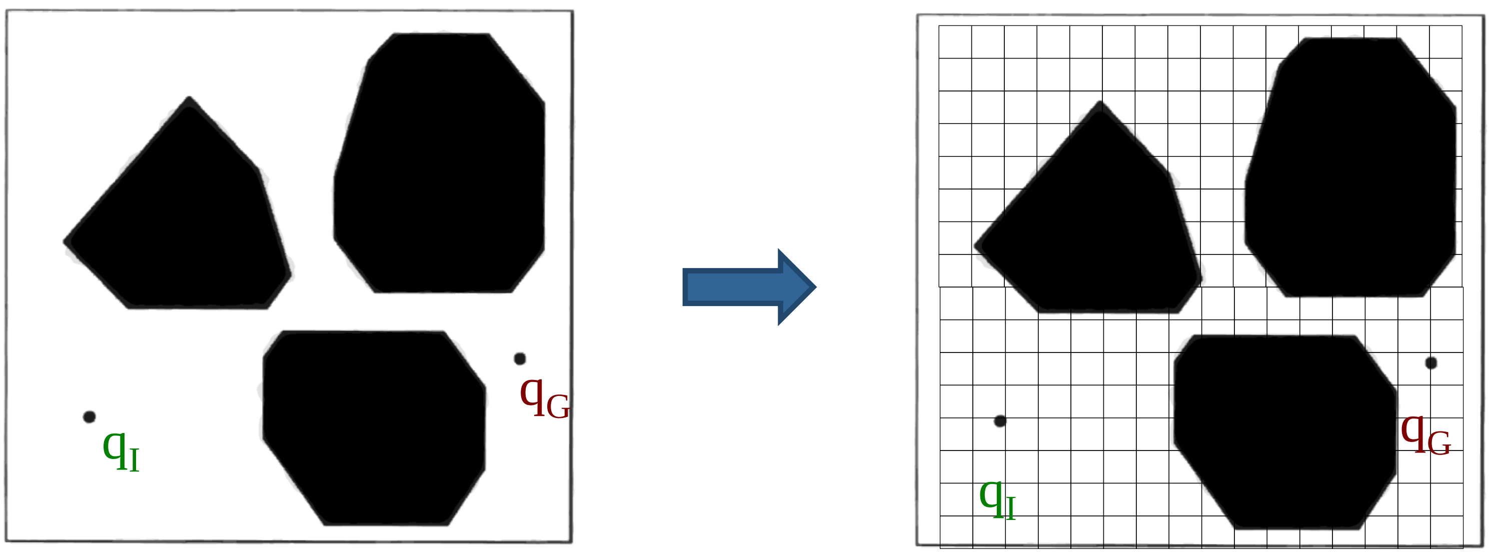
+ Easy to construct in 2D
+ Generates shortest paths

- Optimal planning in 3D or more dim. is NP-hard



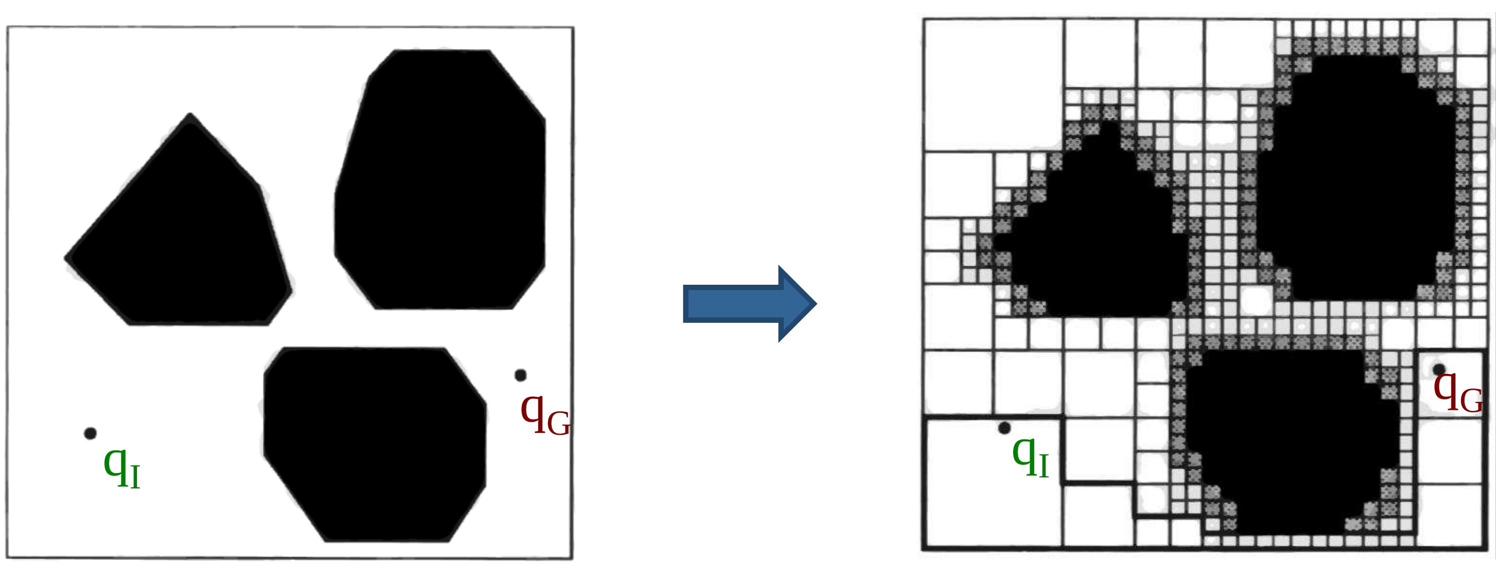
Approximate Decompositions

- Construct a regular grid
- High memory consumption (and number of tests)
- Any ideas?



Approximate Decompositions

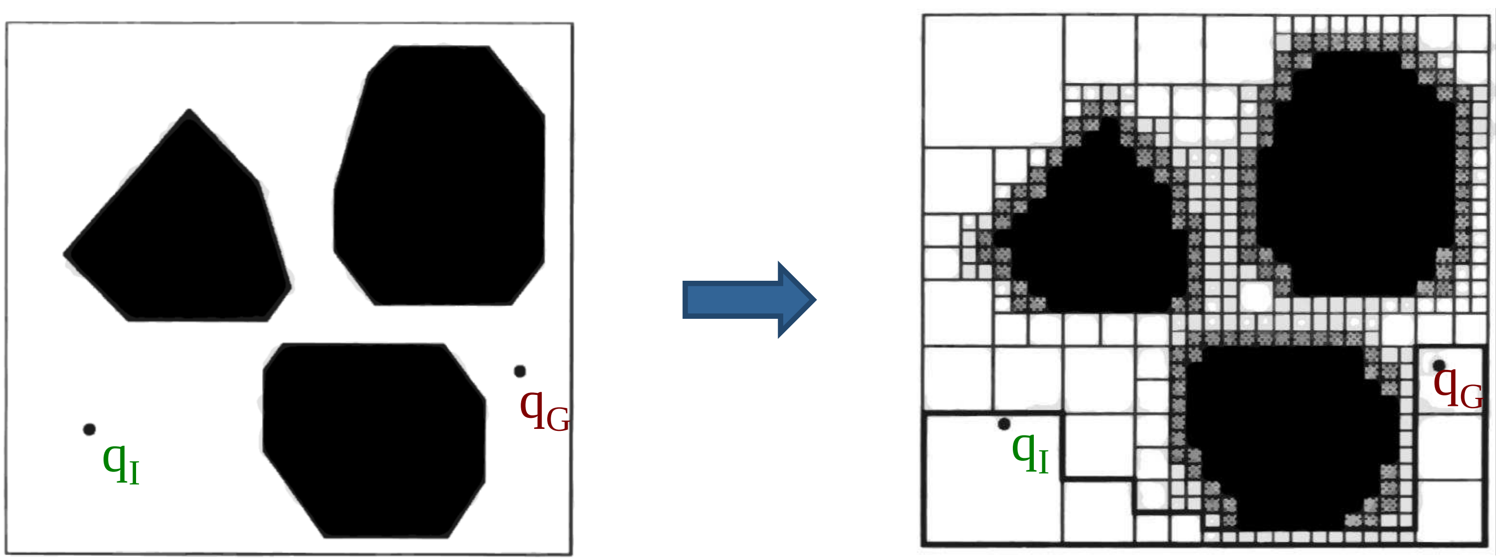
- Construct a regular grid
- Use quadtree/octtree to save memory
- Sometimes difficult to determine status of cell



Approximate Decompositions

+ Easy to construct
+ Most used in practice

- High number of tests



Summary: Combinatorial Planning

- **Pro:** Find a solution when one exists (complete)
- **Con:** Become quickly intractable for higher dimensions

- **Alternative:** Sampling-based planning
  Weaker guarantees but more efficient

Sampling-based Methods

- Abandon the concept of explicitly characterizing C_{free} and C_{obs} and leave the algorithm **in the dark** when exploring C_{free}
- The only light is provided by a **collision-detection algorithm** that probes C to see whether some configuration lies in C_{free}
- We will have a look at
  - Probabilistic road maps (PRMs)
  - Rapidly exploring random trees (RRTs)

Probabilistic Roadmaps (PRMs)

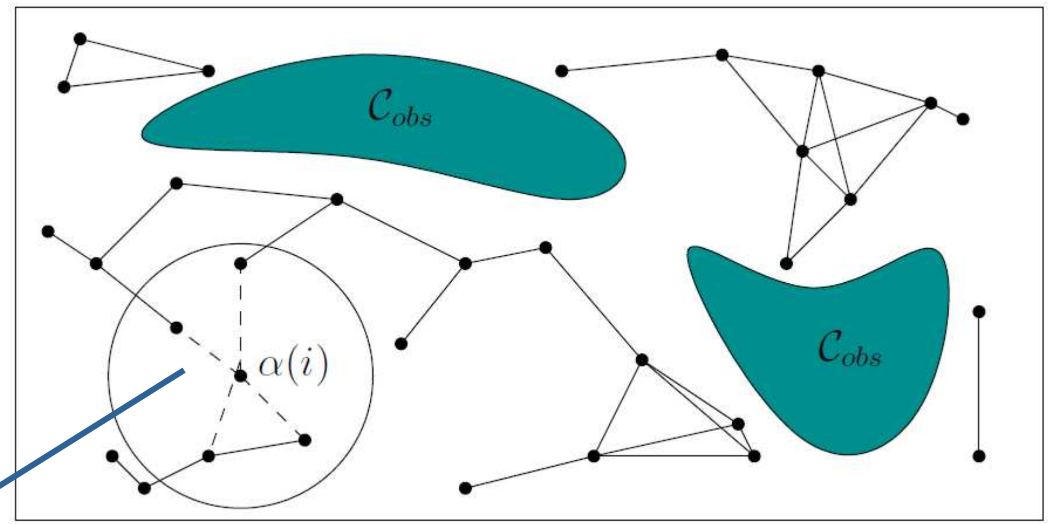
[Kavraki et al., 1992]

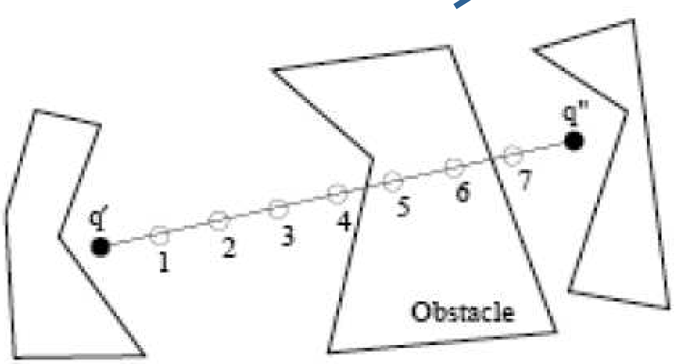
- **Vertex:** Take random sample from C, check whether sample is in C_{free}
- **Edge:** Check whether line-of-sight between two nearby vertices is collision-free

- 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



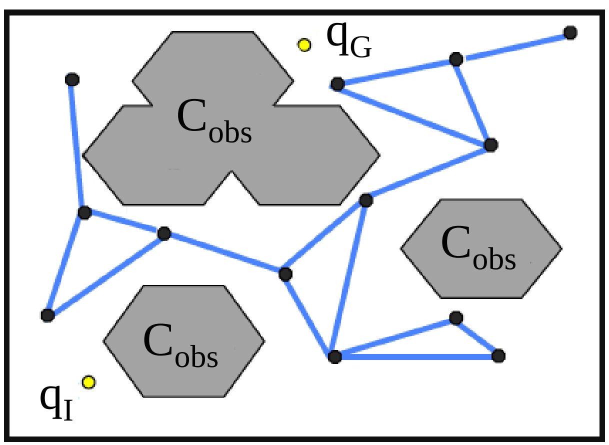


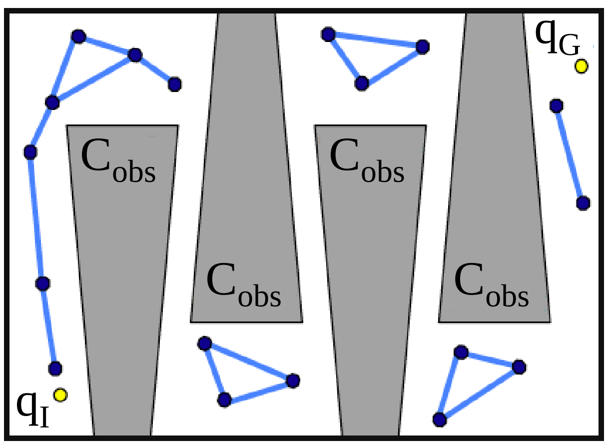
Step 3: Check edges for collisions, e.g., using discretized line search

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

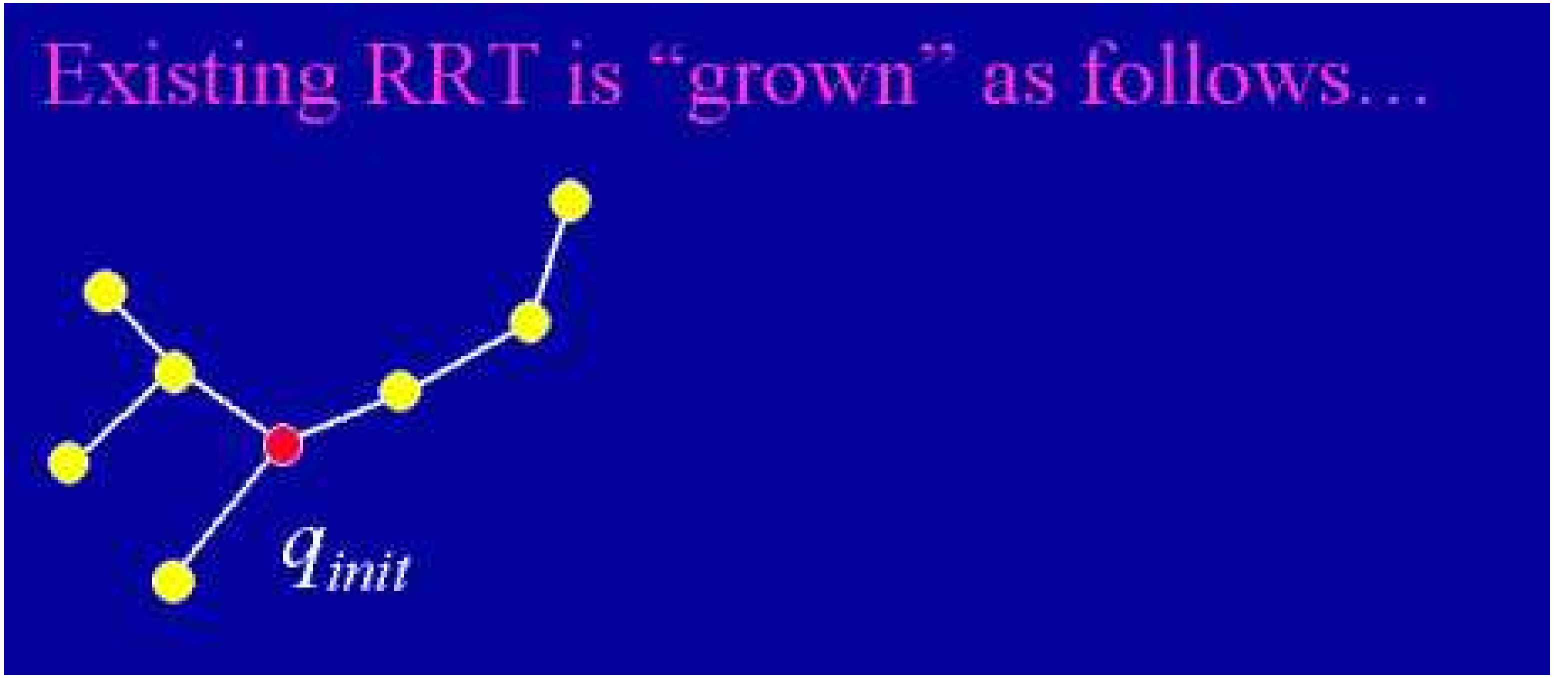




Rapidly Exploring Random Trees

[Lavalle and Kuffner, 1999]

- **Idea:** Grow 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 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?

Rapidly Exploring Random Trees

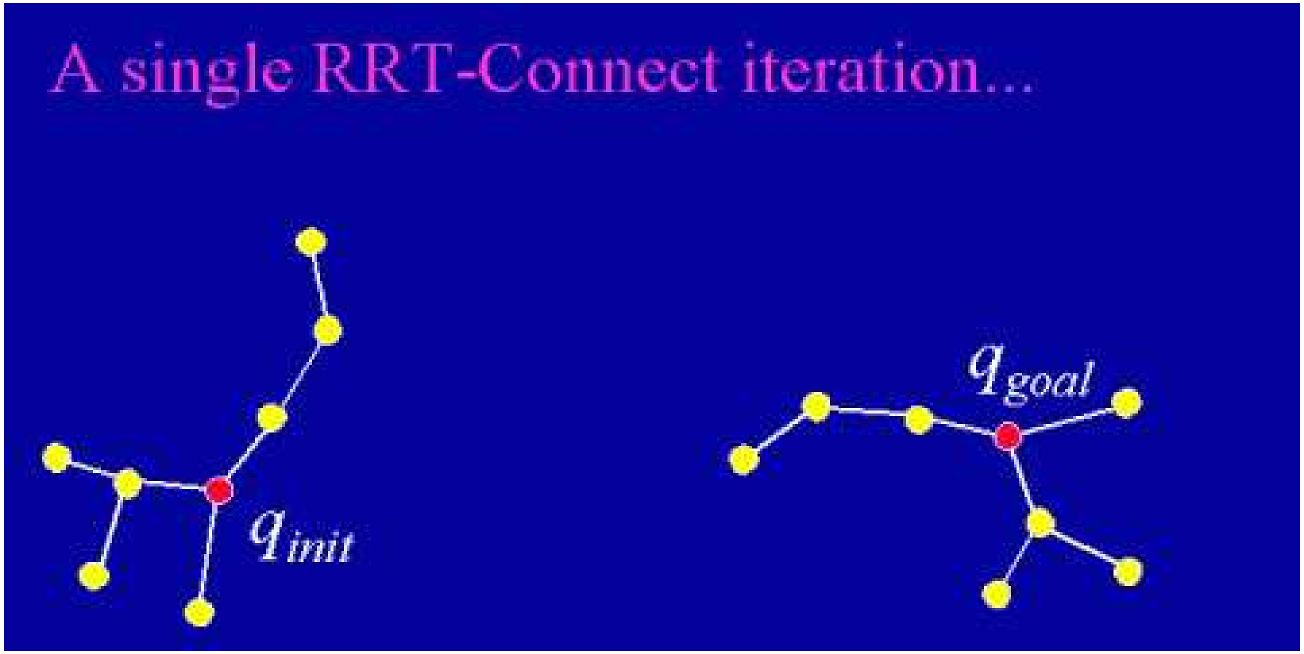
- **Algorithm**
  1. Initialize tree with first node $\mathbf{q}_I$
  2. Pick a random target location (every 100th iteration, choose $\mathbf{q}_G$)
  3. Find closest vertex in roadmap
  4. Extend this vertex towards target location
  5. Repeat steps until goal is reached

- Why not pick $\mathbf{q}_G$ every time?
- This will fail and run into C_{obs} instead of exploring

Rapidly Exploring Random Trees

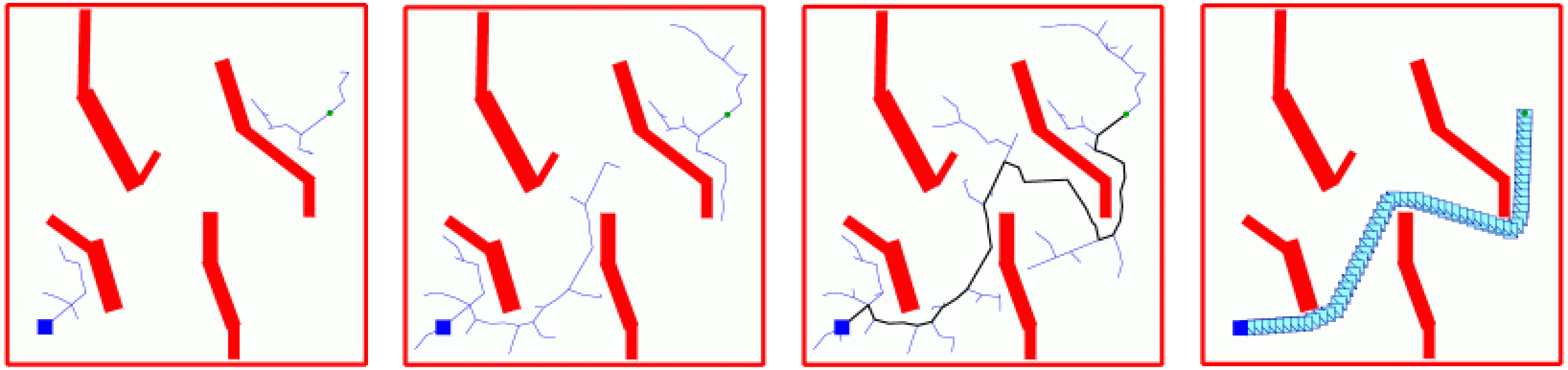
[Lavalle and Kuffner, 1999]

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

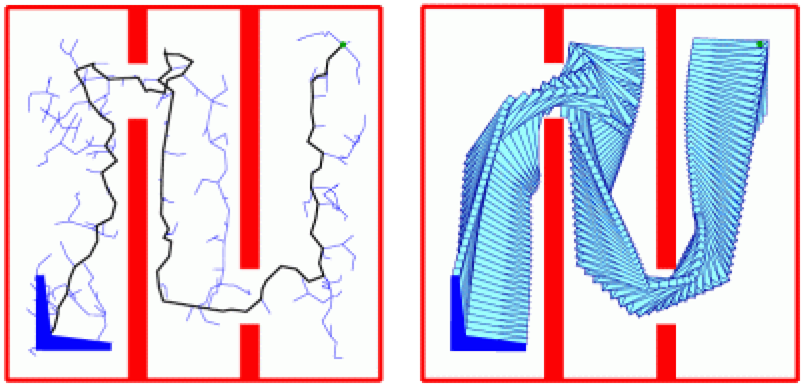


RRT Examples

- 2-DOF example

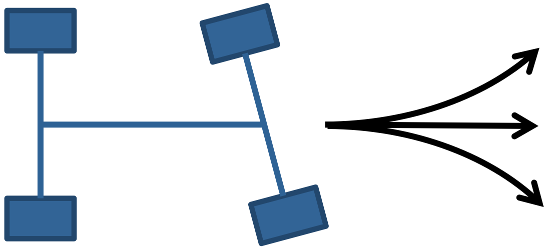


- 3-DOF example (2D translation + rotation)



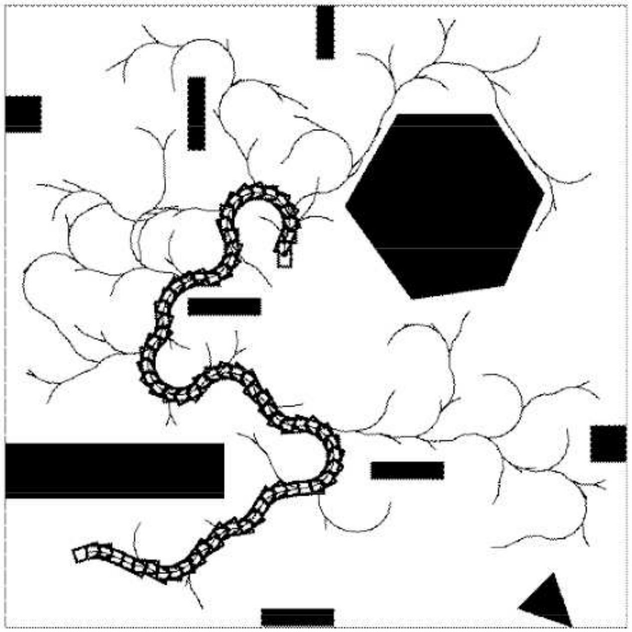
Non-Holonomic Robots

- 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

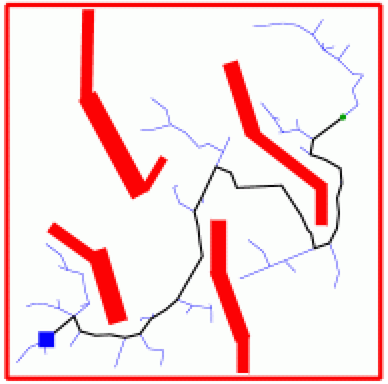
- RRTs can naturally consider such constraints during tree construction
- Example: Car-like robot



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, …)

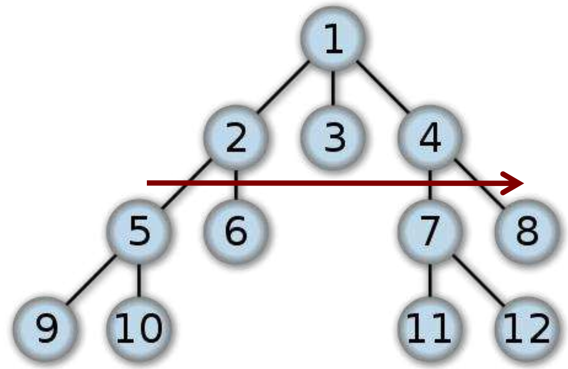
Search Algorithms

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

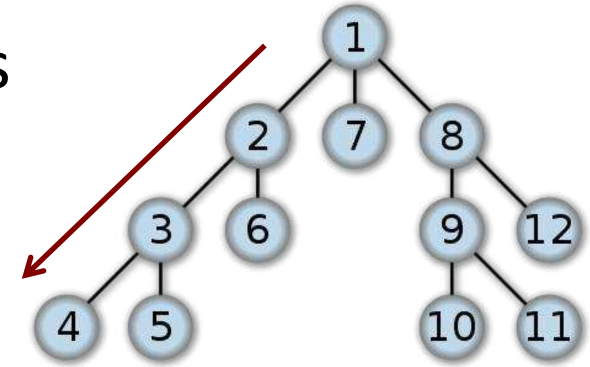
- What search algorithms do you know?

Uninformed Search

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

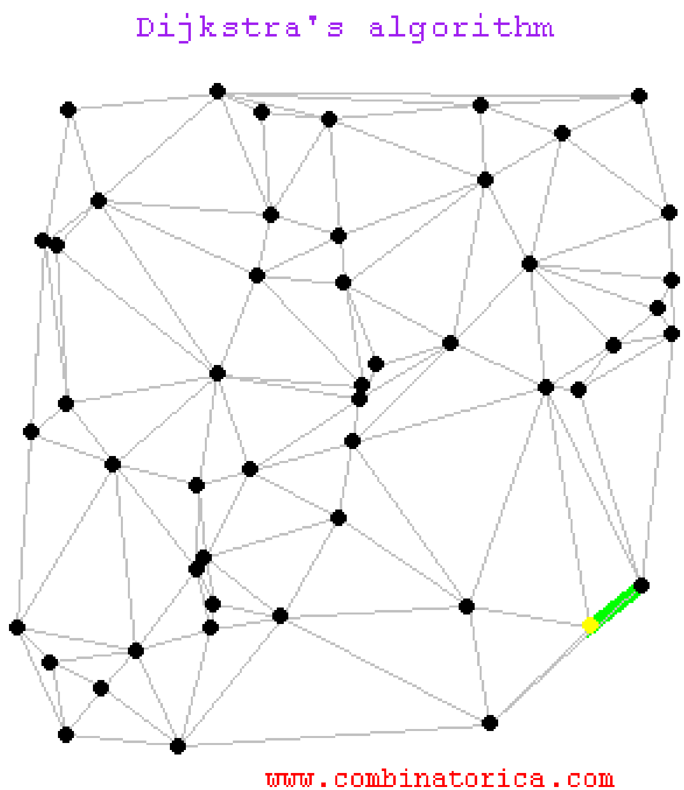


- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$
    (can forget explored subtrees)



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 $f(n)$
  - 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 $f(n)$
  - First explore the node with lowest value
- What is a good evaluation function?
- Often a combination of
  - Path cost so far $g(n)$
  - Heuristic function $h(n)$
    (e.g., estimated distance to goal, but can also encode additional domain knowledge)

Informed Search

- **Greedy best-first search**
  - Simply expand the node closest to the goal

$$f(n) = h(n)$$

  - Not optimal, not complete

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

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

  - **Optimal** and **complete** (if $h(n)$ never overestimates actual cost)

What is a Good Heuristic Function?

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

Comparison Search Algorithms

Problems on A* 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

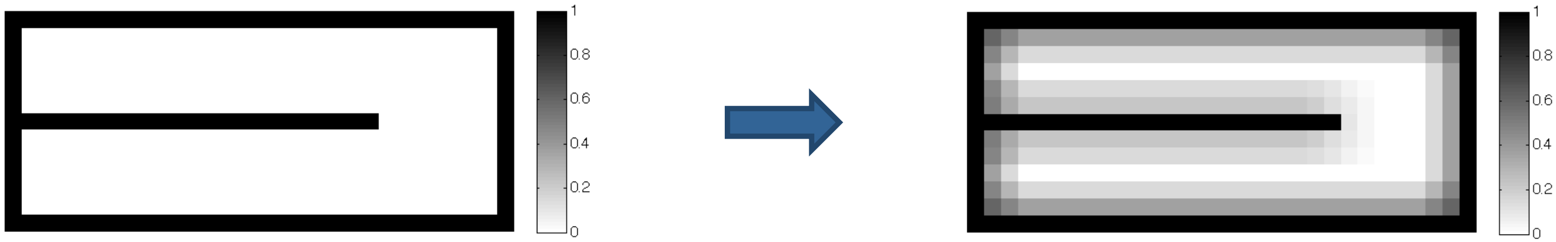
Problems on A* on Grids

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

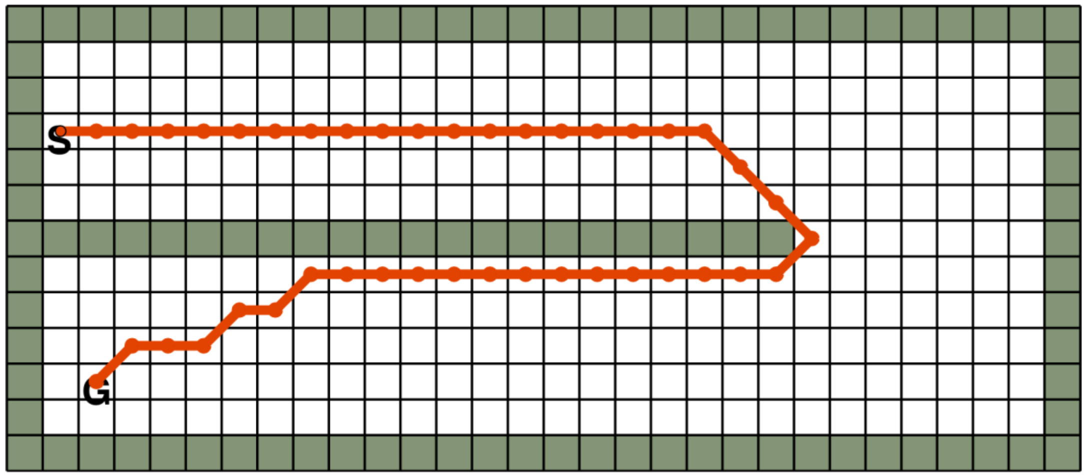
- **Problem:** Path gets close to obstacles
- **Solution:** Convolve the map with a kernel (e.g., Gaussian)

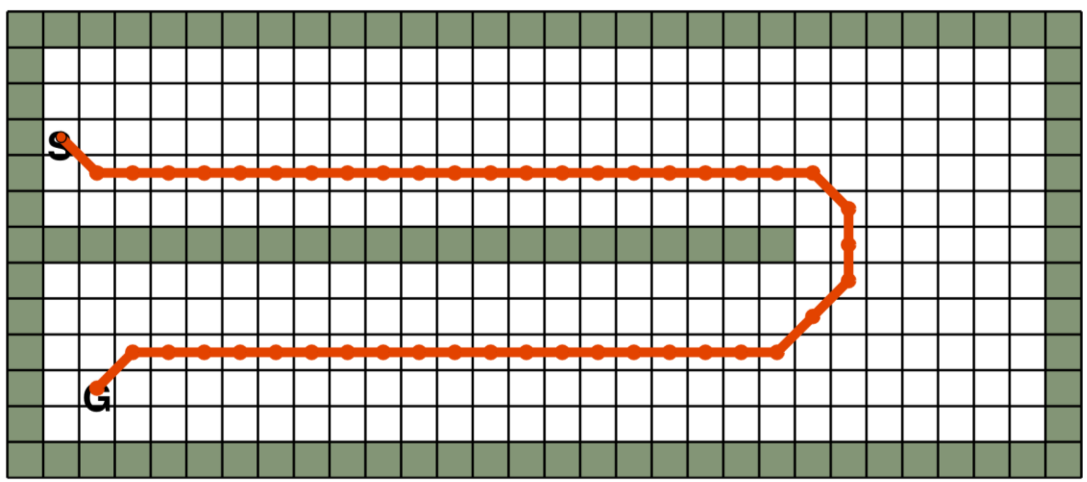


- Leads to non-zero probability around obstacles
- Evaluation function

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

Example: Map Smoothing



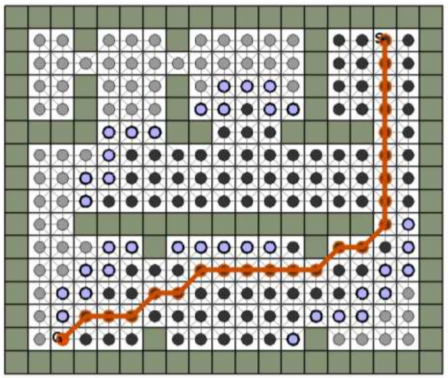


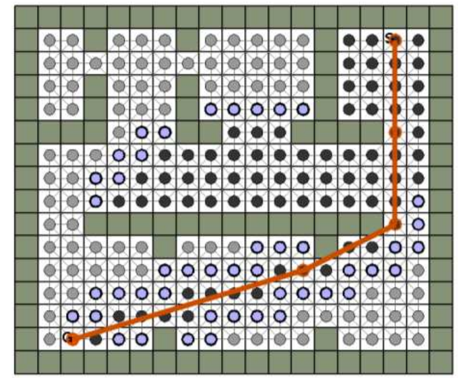
Path Smoothing

- **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)
  - …

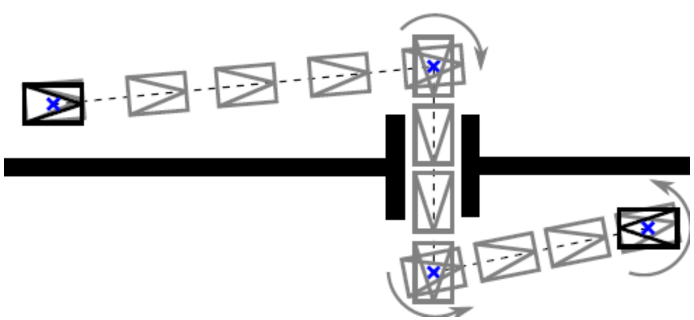
Example: Path Smoothing

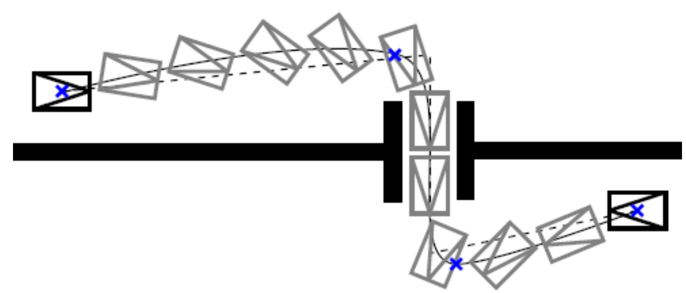
- Replace pairs of nodes by line segments





- Non-linear optimization





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]

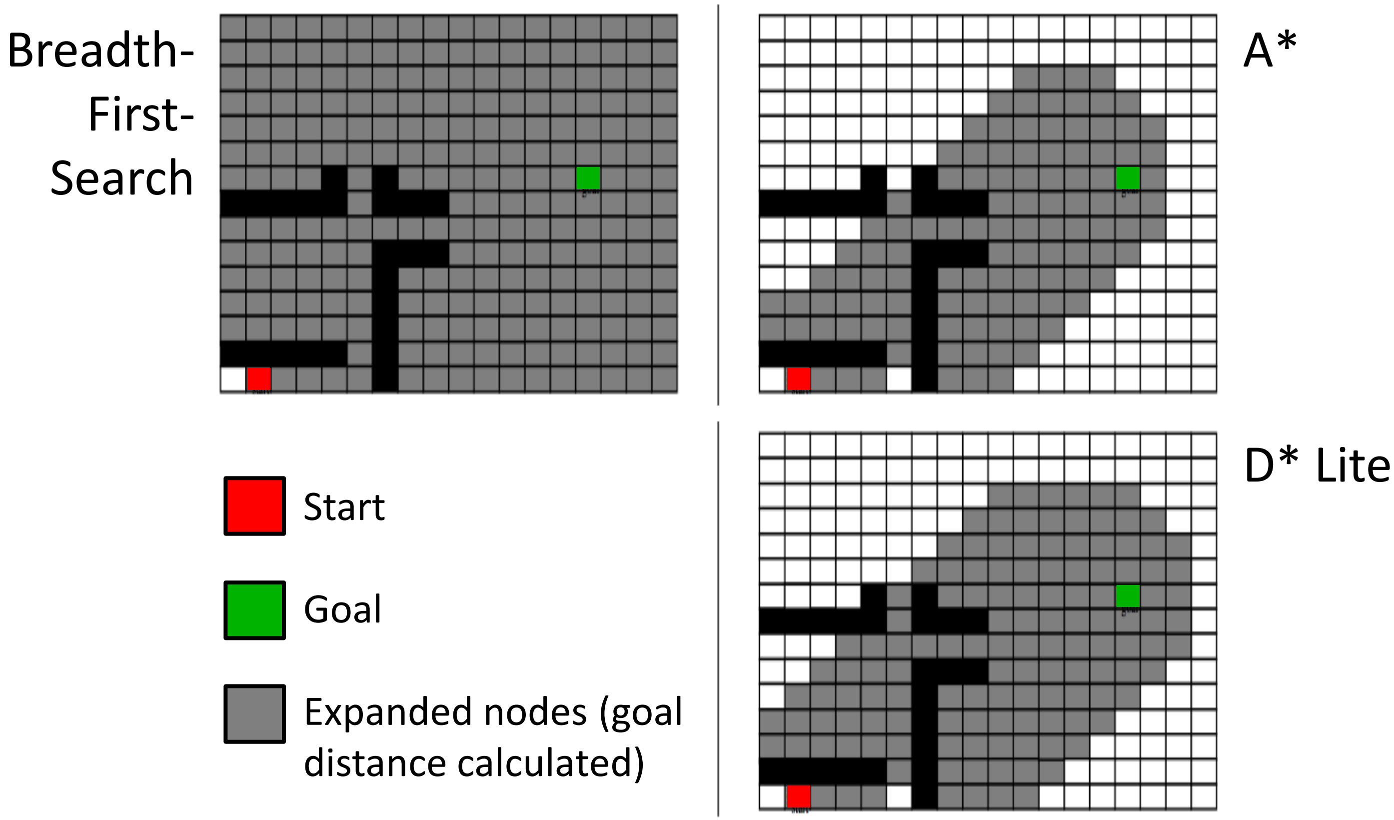
D* Search

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

- Situation at start



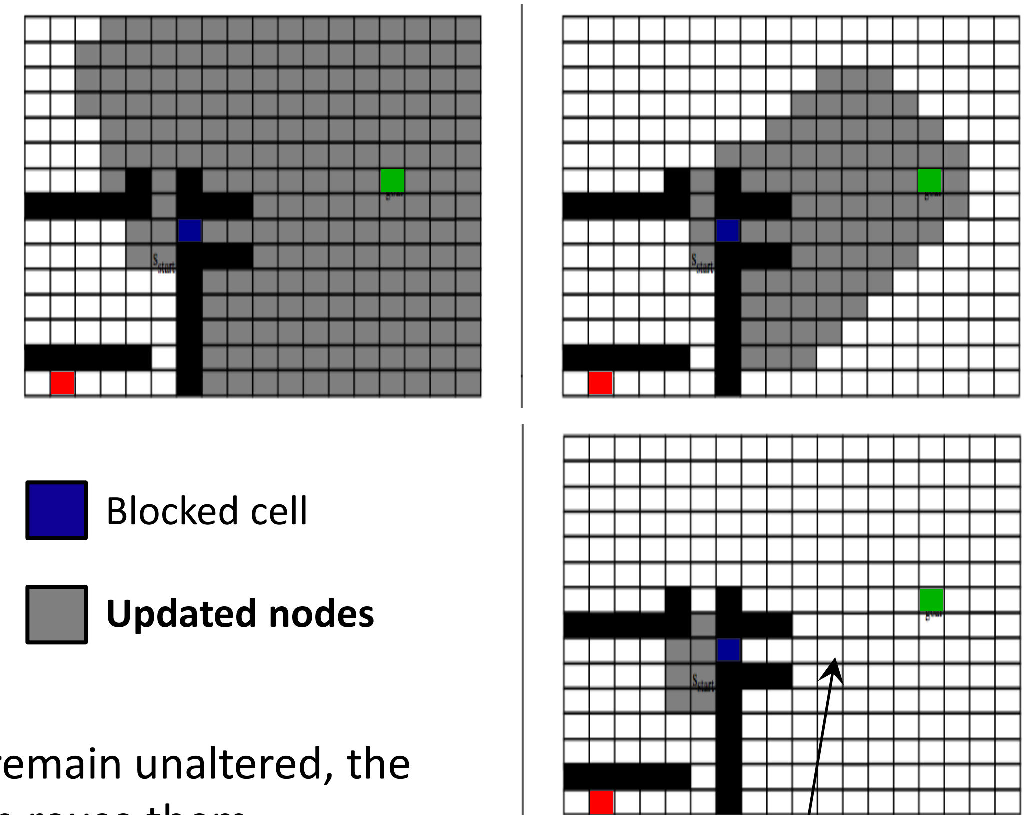
D* Example

- After discovery of blocked cell

Breadth-First-Search

A*

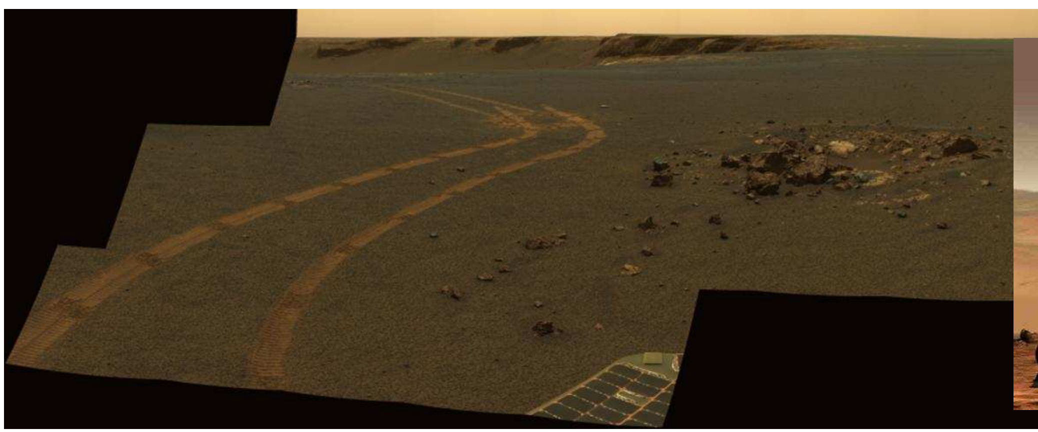
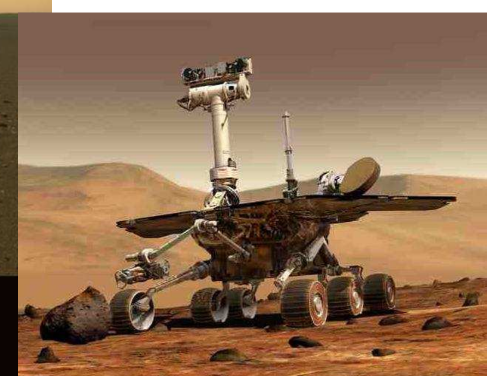
D* Lite



All other nodes remain unaltered, the shortest path can reuse them.

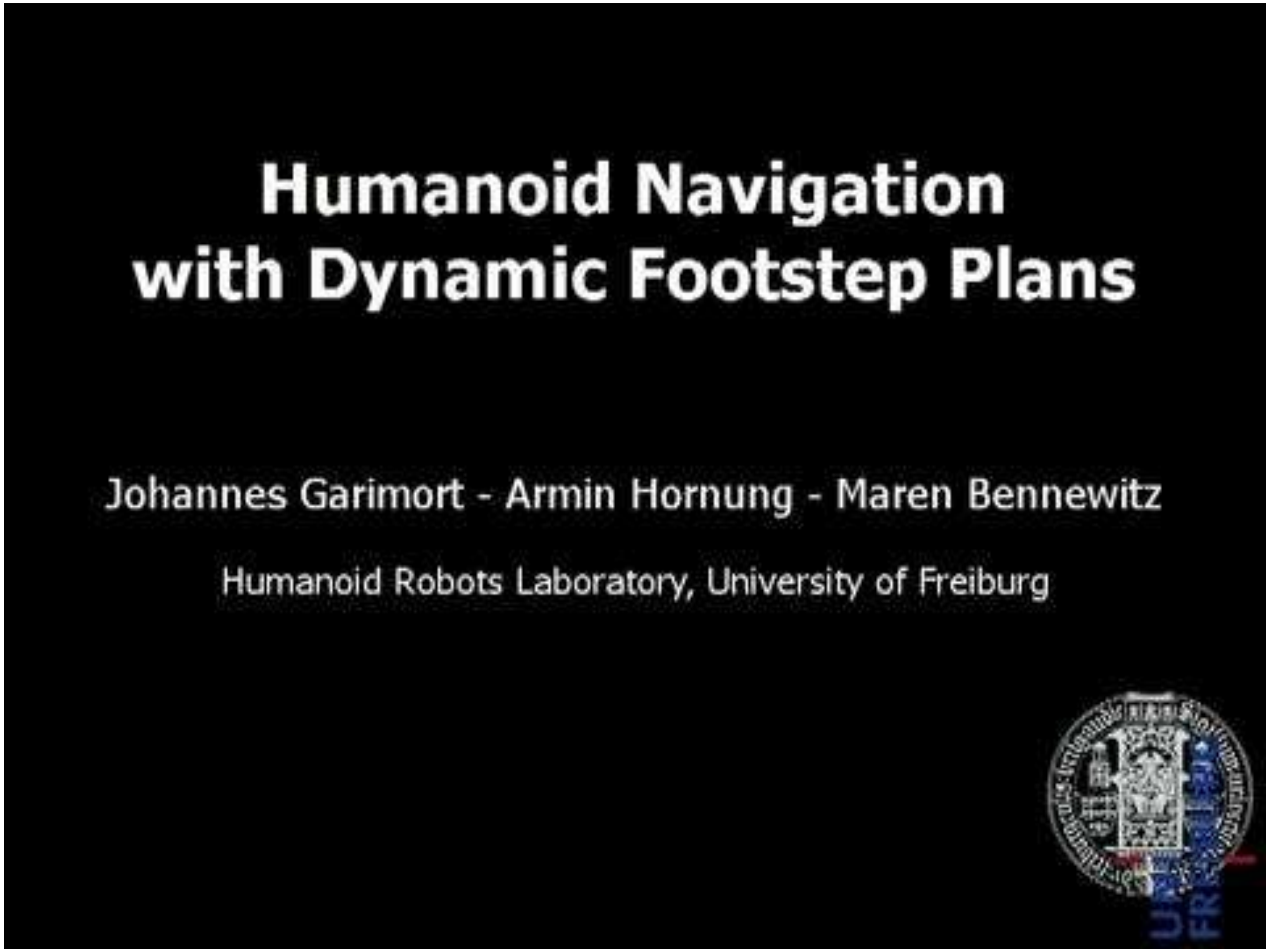
D* Search

- 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]



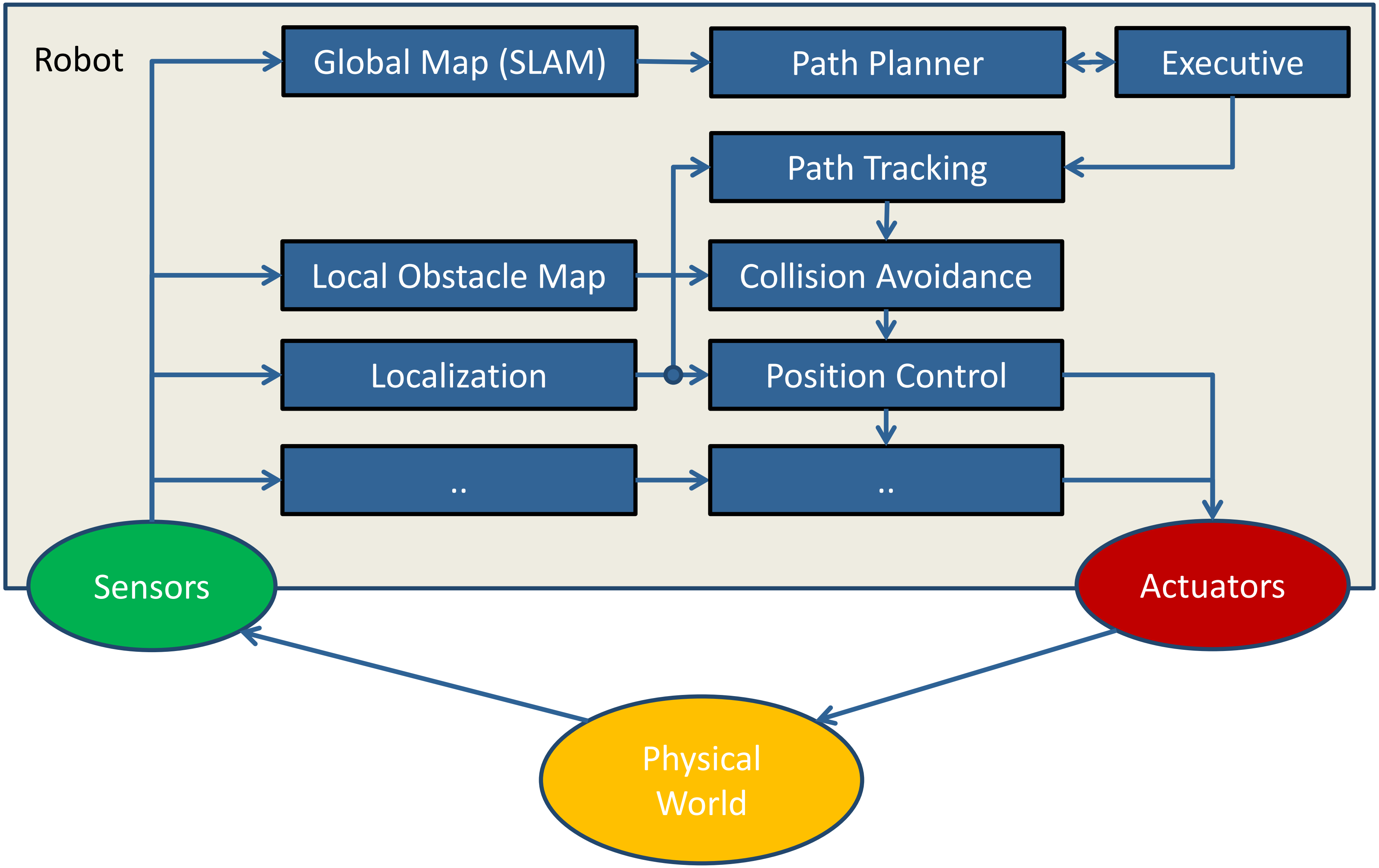
Real-Time Motion Planning

- 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

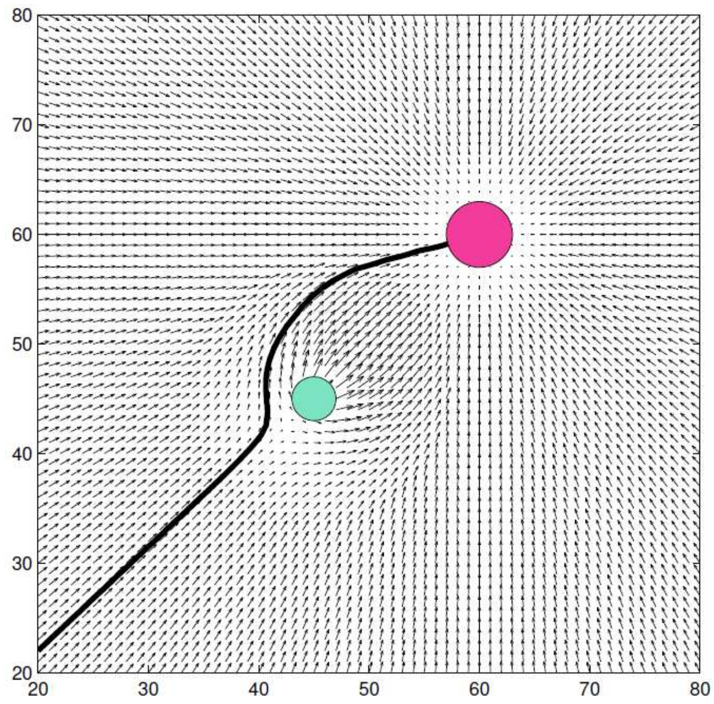
- An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate **local planner** accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)

Local Planner

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

Navigation with Potential Fields

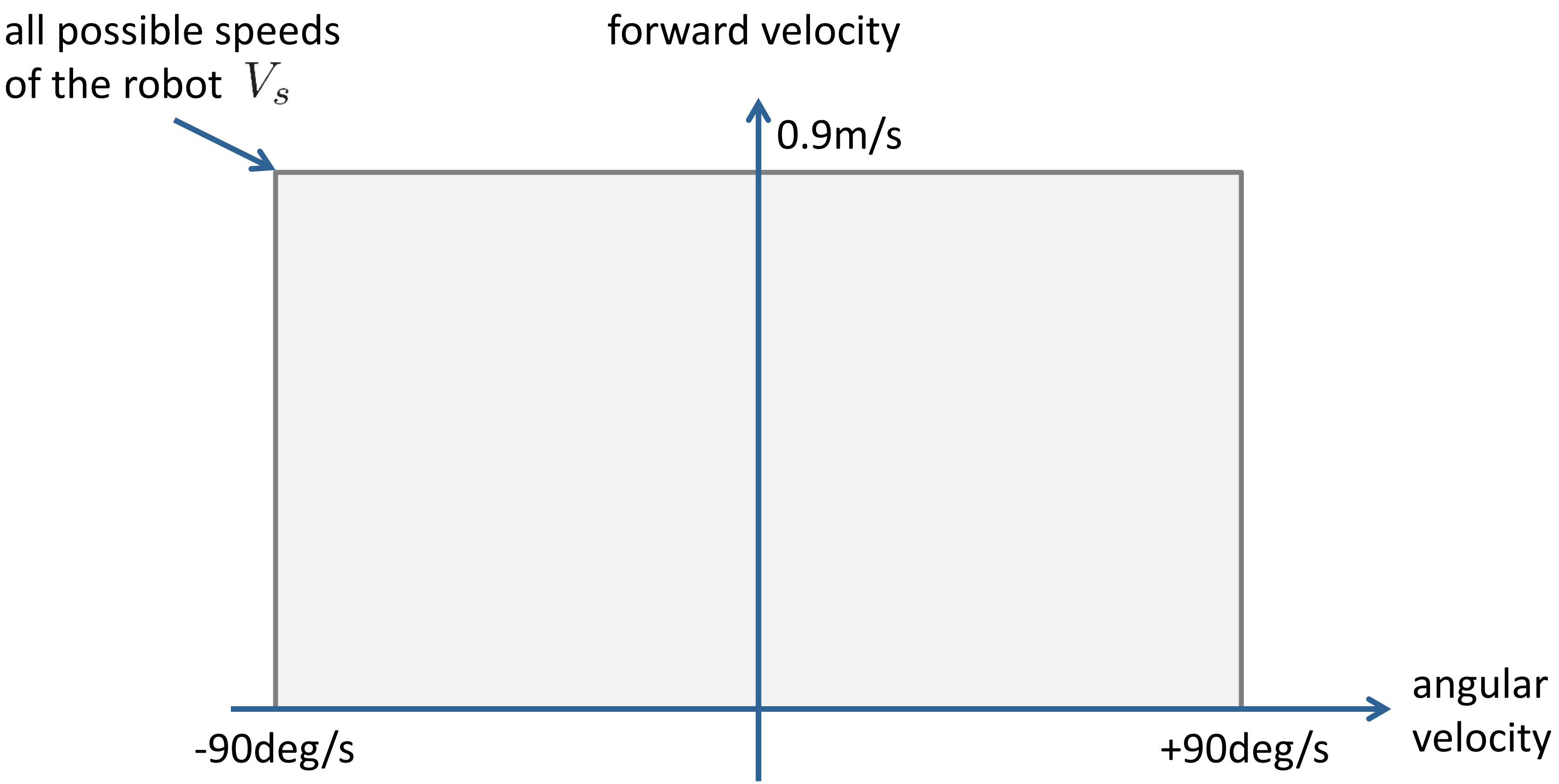
- Treat robot as a particle under the influence of a potential field
- **Pro:**
  - easy to implement
- **Con:**
  - suffers from local minima
  - no consideration of dynamic constraints



Dynamic Window Approach

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

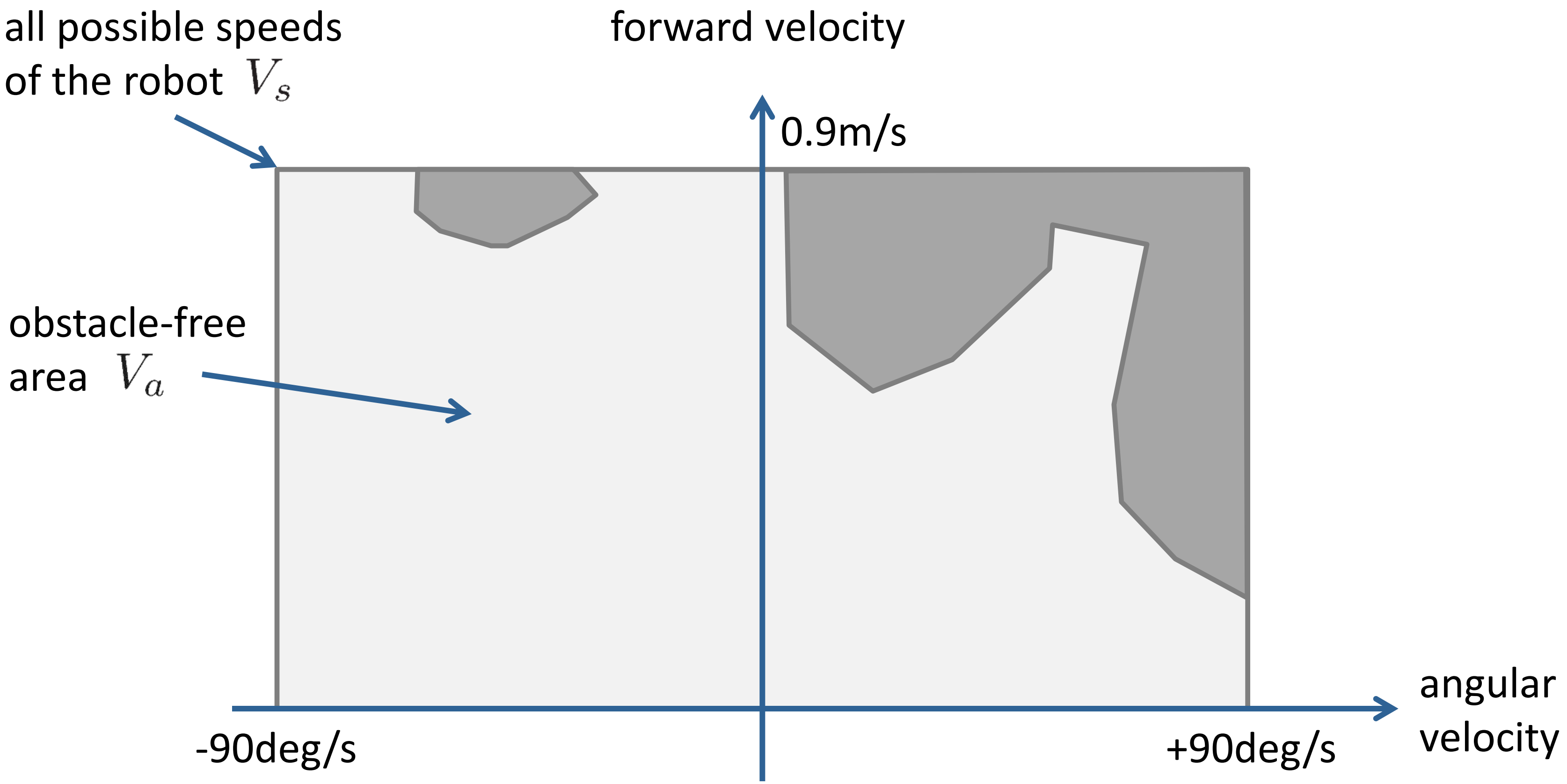
- Consider a 2D planar robot



Dynamic Window Approach

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

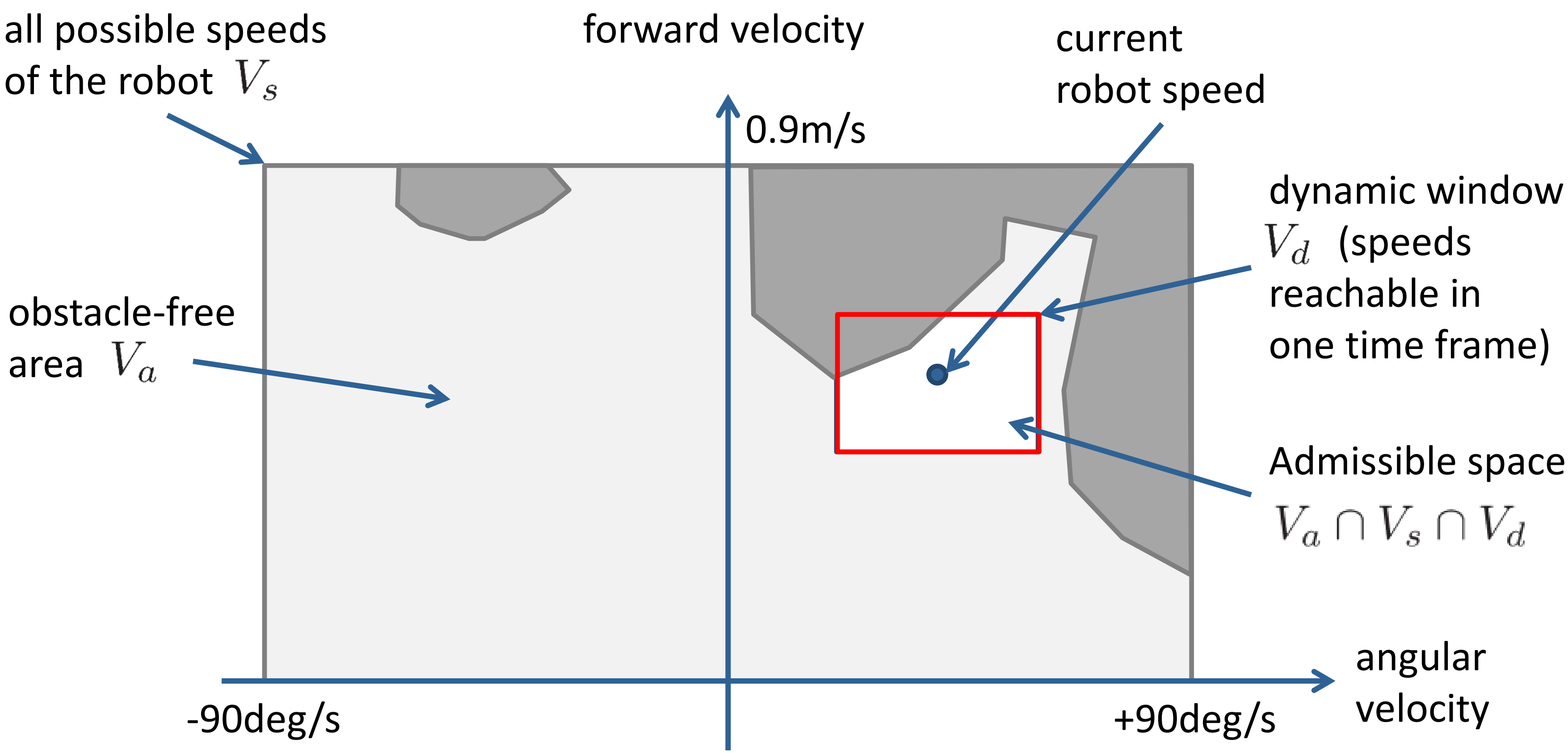
- Consider a 2D planar robot + 2D environment



Dynamic Window Approach

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

- Consider additionally dynamic constraints



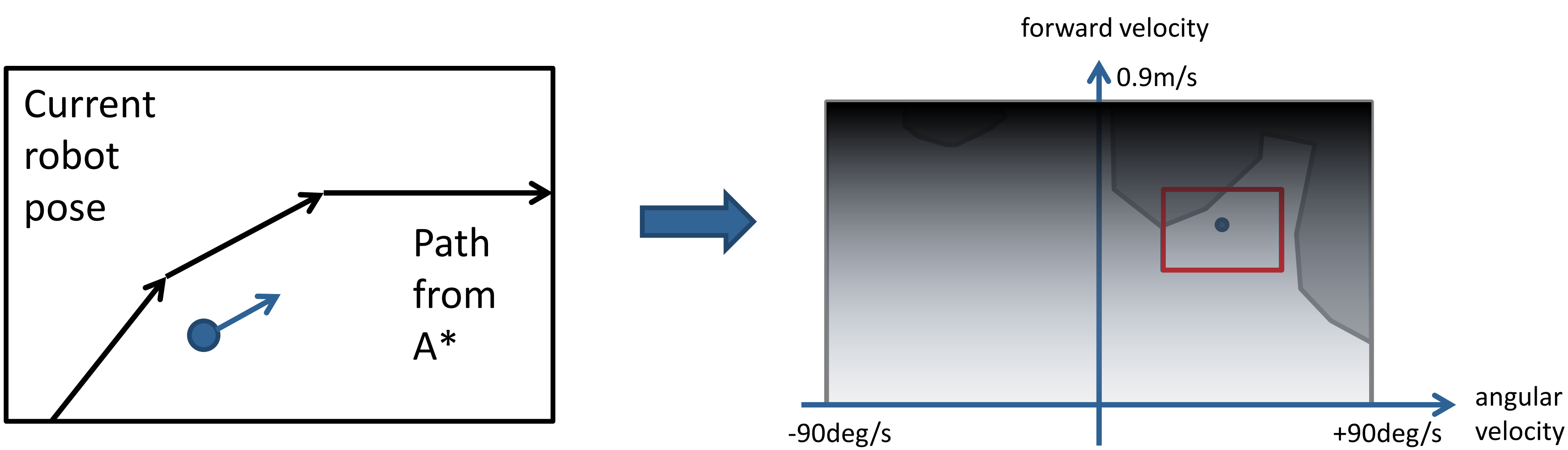
Dynamic Window Approach

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

- Navigation function (potential field)

[illegible]

Maximizes velocity



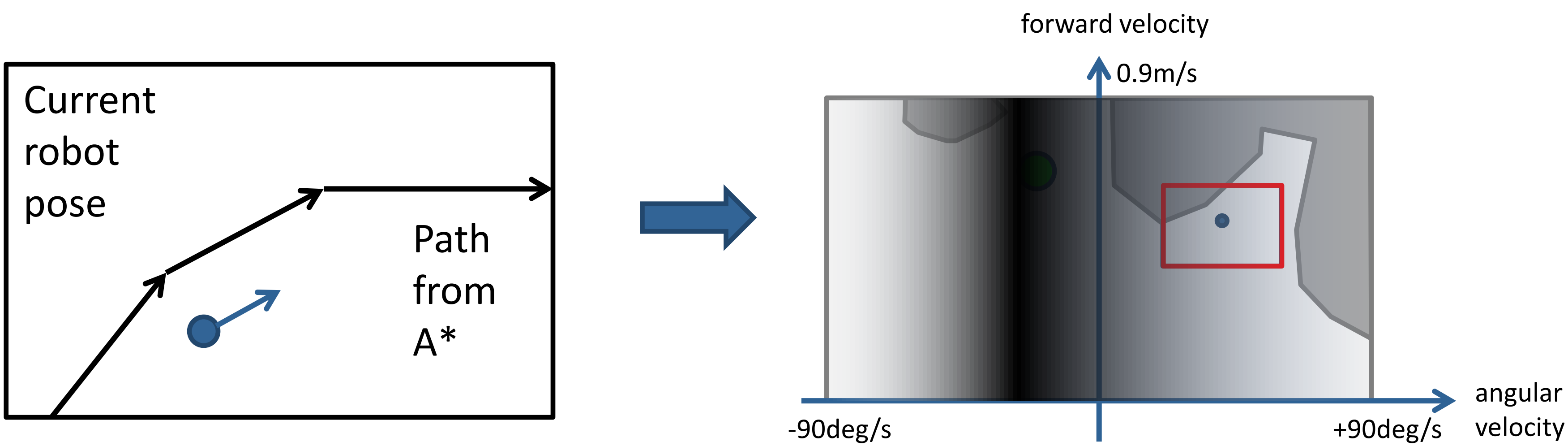
Dynamic Window Approach

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

- Navigation function (potential field)

Maximizes velocity

Rewards alignment to A* path gradient



Dynamic Window Approach

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

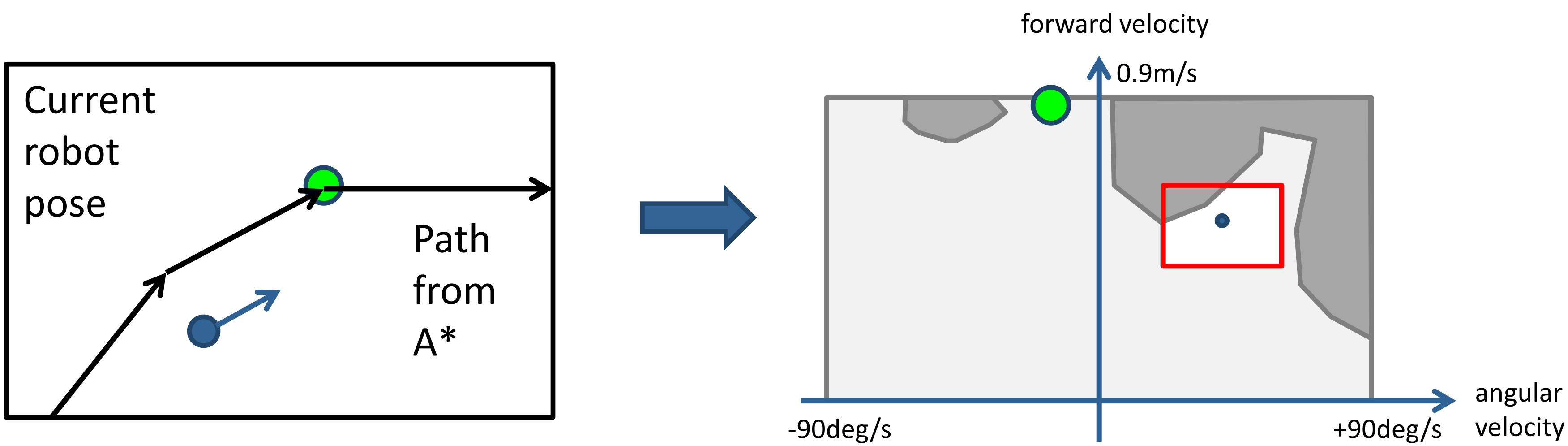
- Navigation function (potential field)

[illegible]

Maximizes velocity

Rewards alignment to A* path gradient

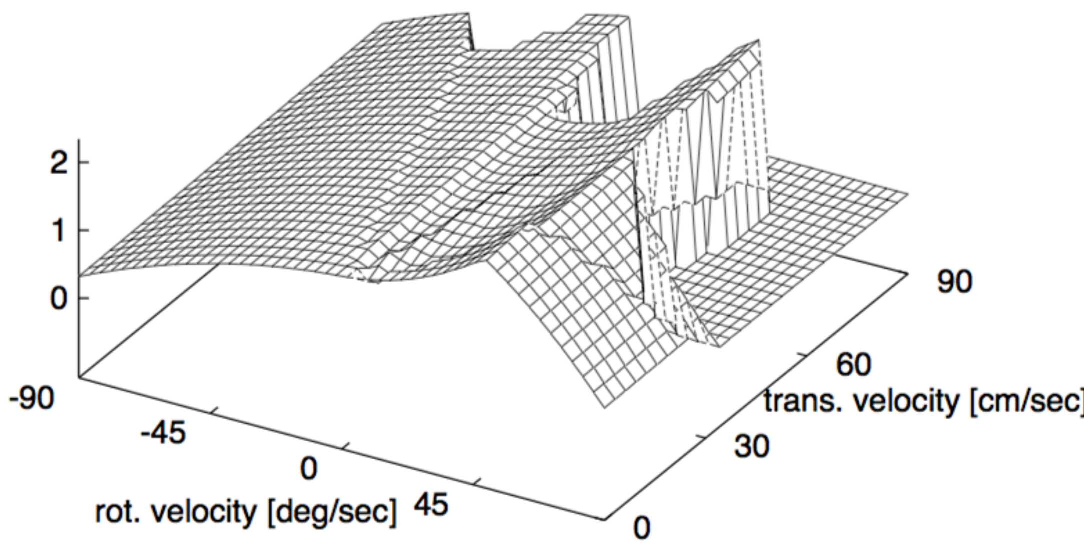
Rewards large advances on A* path



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

Lessons Learned Today

- Motion planning problem and configuration spaces
- Roadmap construction
- Search algorithms and path optimization
- Local planning for path execution