

5 Termine Bahnplanung

22.10. wird getauscht

Course Schedule & Topics

Lecture Dates and Topics

- + 08.10.2024 - 05.11.2024: Path Planning (Yucheng Tang)
 - + Note: 22.10.2024 lecture will be swapped due to a conference
- + 12.11.2024 - 26.11.2024: KUKA Industrial Robot Programming (Hein)
- + 03.12.2024 - 14.01.2025: Policy Learning (Sóti)
 - + 17.12.2024: Project Assignment
 - + Note: no lectures on 24.12.2024 and 31.12.2025
- 21.01.2025: Final Exam - Project Presentations

Important Dates

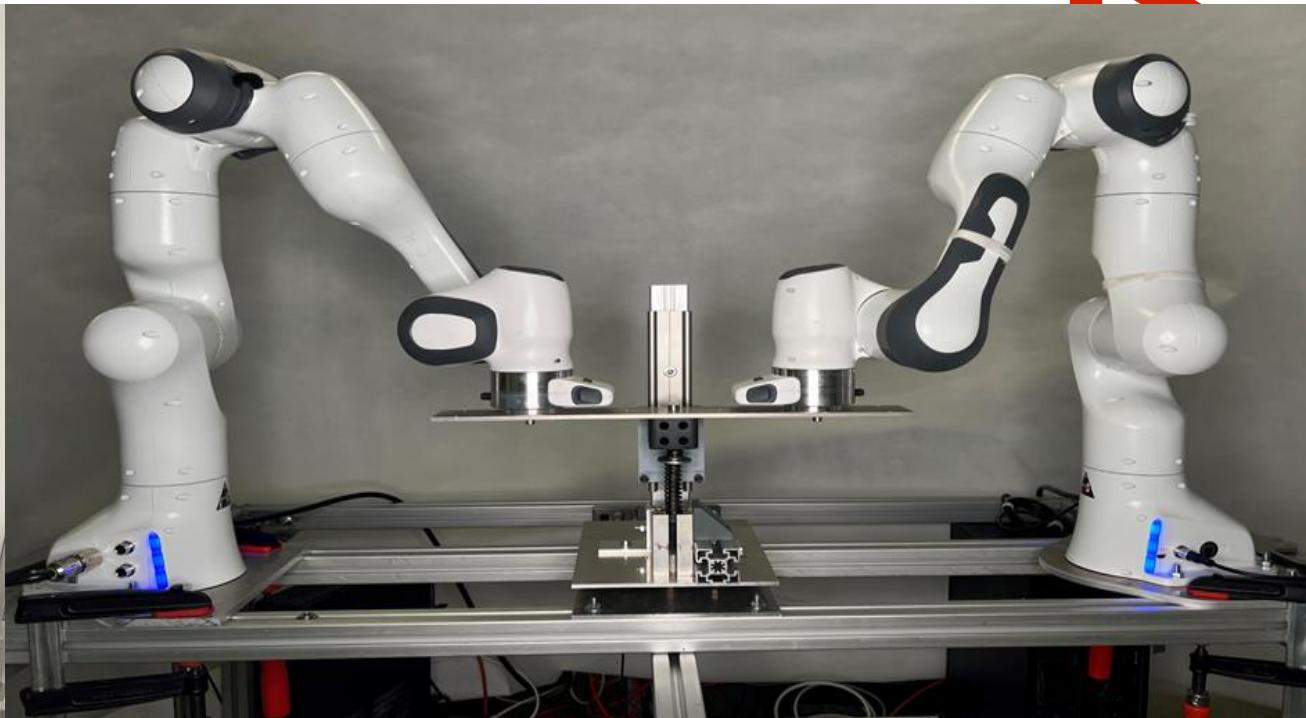
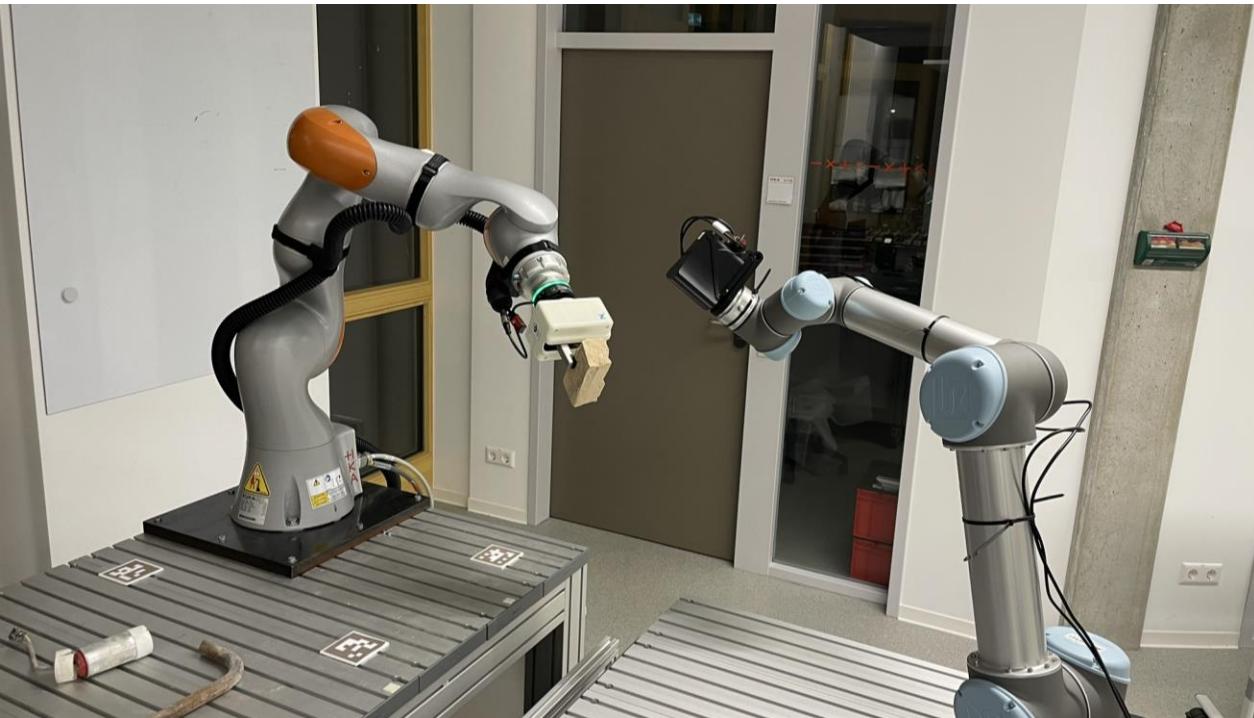
- Project Deliverables Due: 04.02.2025

Präsentation + Code + Dokumentation + Evaluierung

Großer Teil des Skripts befindet sich als
Jupyter Notebook auf ILIAS

Robot Programming

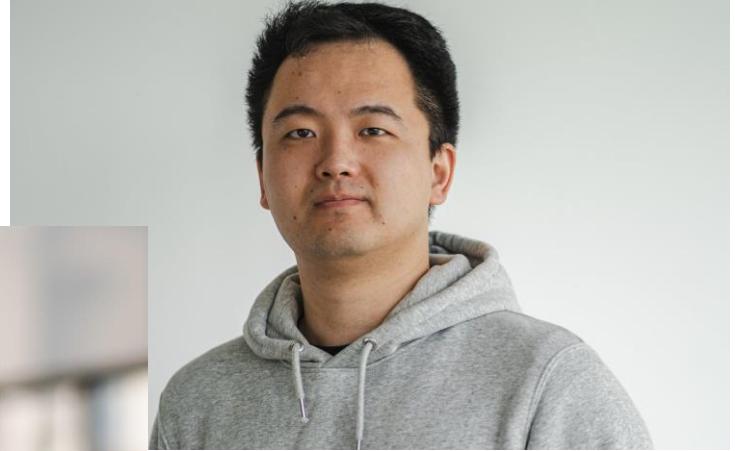
Lecture 1 – Introduction to Motion Planning



Practical Issues



- Lecturer
 - Yucheng Tang
 - Gergely Sóti
 - Prof. Dr.-Ing. habil. Björn Hein
- Office hours
 - TBD
- Exam format
 - 20-minutes oral exam + project work
- Topics
 - Introduction to Motion Planning
 - Python and Math tutorial
 - Discrete Motion Planning
 - Sampling-based Motion Planning
 - Learning-based Motion Planning
 - Introduction to KRL (Prof. Hein)
 - Imitation and Reinforcement Learning in MP (Gergely)



Yucheng Tang

08.10.2024

What you can expect to get from this course



- A comprehensive understanding of path planning algorithms for articulated robots
 - Traditional methods: discrete, sampling and optimization-based planning
 - Machine learning: end-to-end, module replacement
- Programming experience (*Python*)
- Gain a comprehensive overview of the structure, operation and programming of industrial robots.
- Learn and apply AI methods for robot policy learning
 - Reinforcement learning, imitation learning



Introduction to Motion Planning

What is path/motion planning?

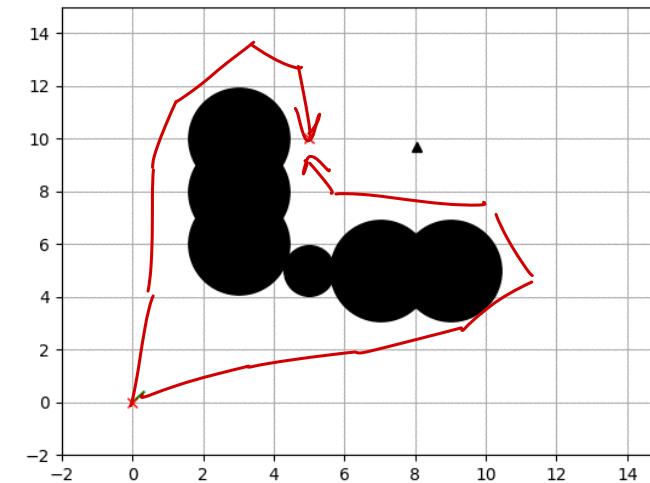
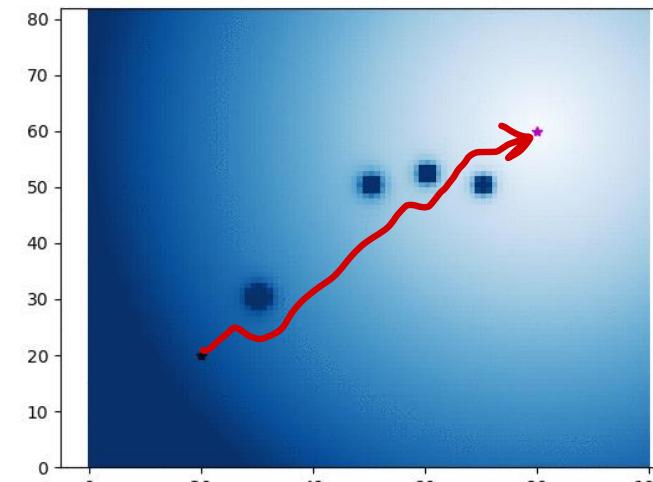
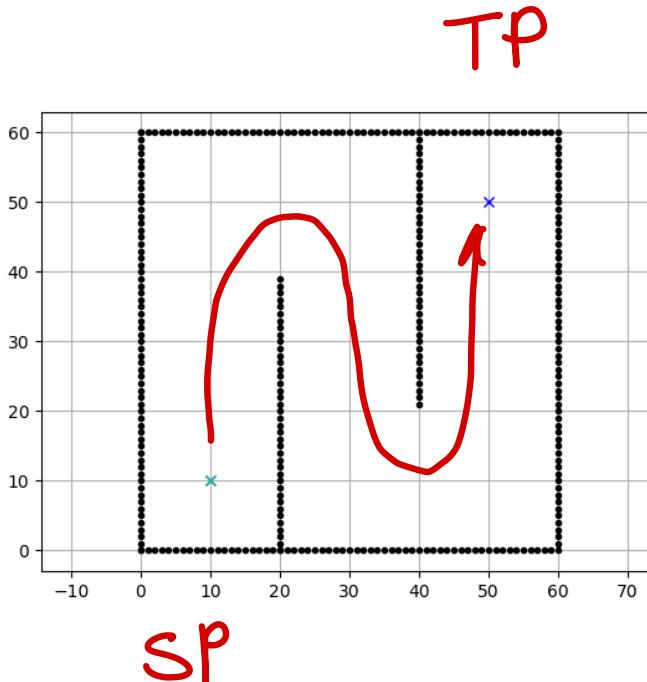
+

- The automatic generation of motion

path → waypoints

motion → waypoints + timestamps

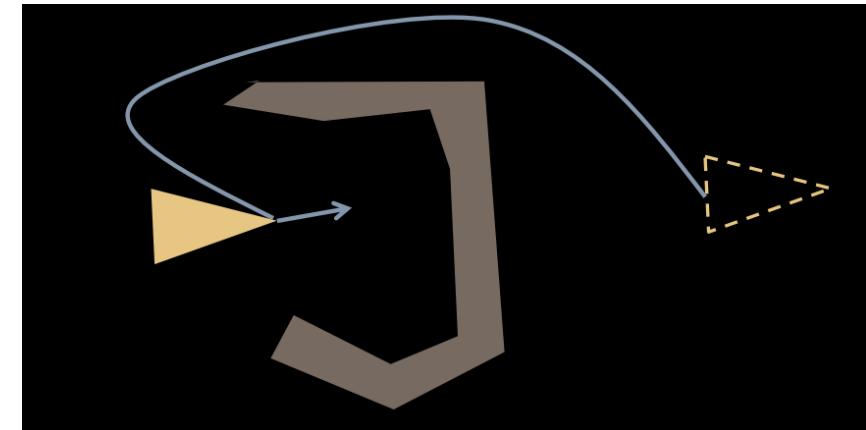
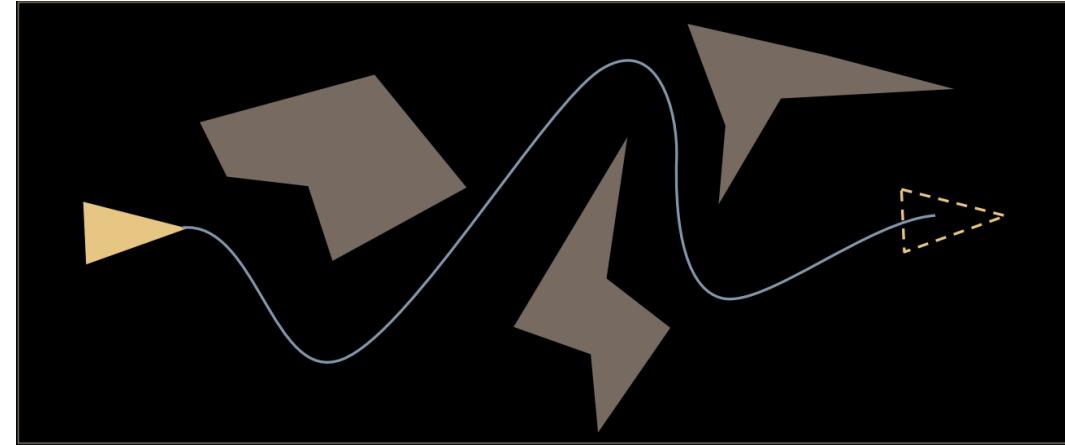
K
A



Why Motion Planning Instead of Obstacle Avoidance



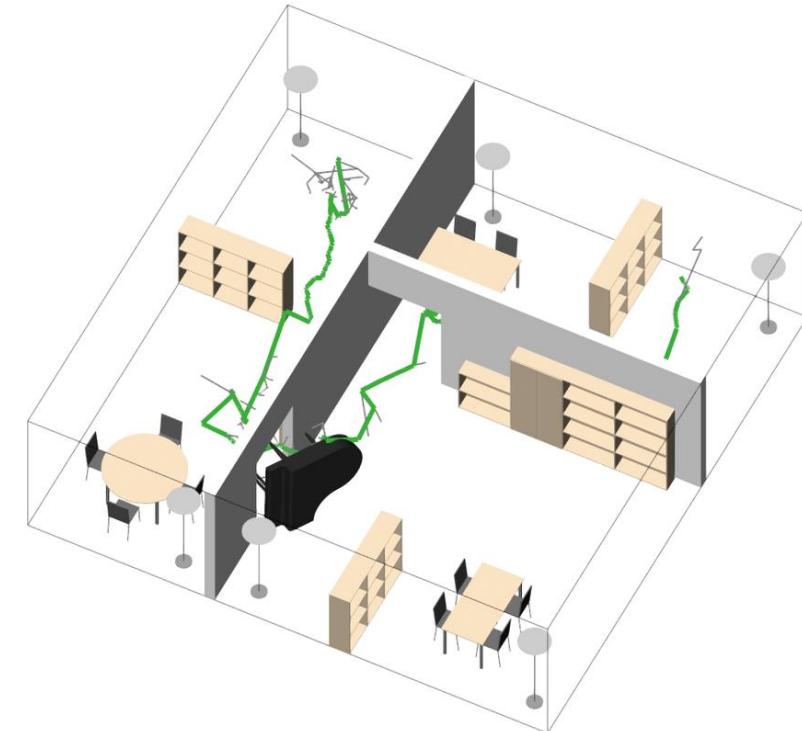
- Motion Planning
 - (Inverse Kinematics)
 - Path Planning respecting weight, velocity and acceleration, ...
 - Trajectory Generation
 - Controller Execution
- Path Planning → einmalige Berechnung
 - Low-frequency, time-intensive search method for global finding of a (optimal) path to a goal
- Obstacle avoidance (aka „Local planner“ or „Controller“) → konstante Korrektur (dynamische Hindernisse)
 - Fast, reactive method with local time (control loop)
- Distinction: Global vs. Local reasoning



The Piano Mover's Problem



- Piano is a “robot” with
 - 3 DoF (Degrees of Freedom) in the 2D case
 - Translation in x-axis, y-axis
 - Rotation around z-axis
 - 6 DoF in the 3D case
 - Translation in x-, y-, z-axis
 - Rotation around x-, y-, z-axis
- The Kinematic is **holonomic** *axes are independent (x, y, z) [next page]*
- Wall, chairs, tables, etc... are fixed, the environment can be assumed to be static



[1] Balancing Exploration and Exploitation in Sampling-Based Motion Planning

Holonomic vs non-holonomic robot



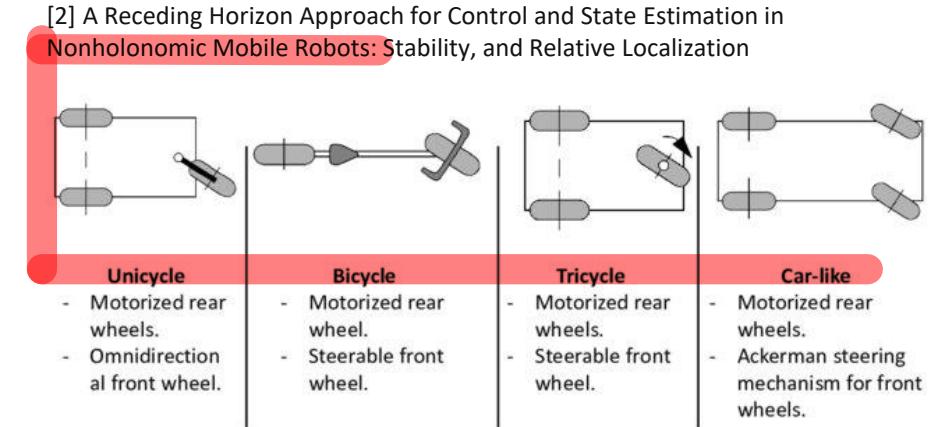
- The ability to move in all direction and rotate independently

- Holonomic**

- Every DoF can be changed separately and independently from the others

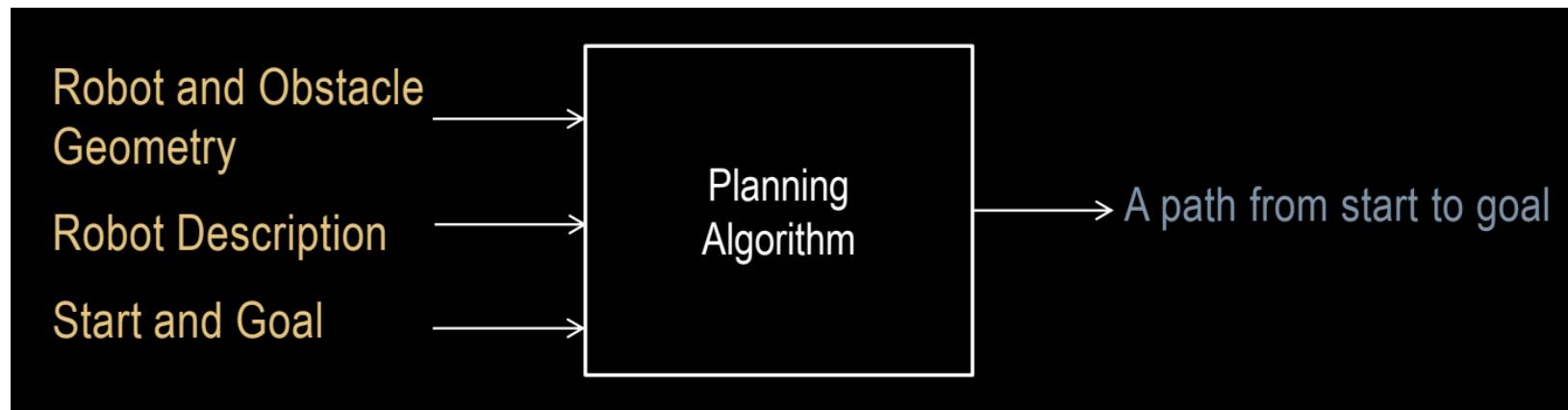
- Non-holonomic**

- DoFs are coupled
- Car-like model: it is not possible to turn without moving forward



Basic Problem Statement

- Getting a path, consisting of collision free movements between starting and goal position
- Information, whether such a path exists or not



What can motion planning do?



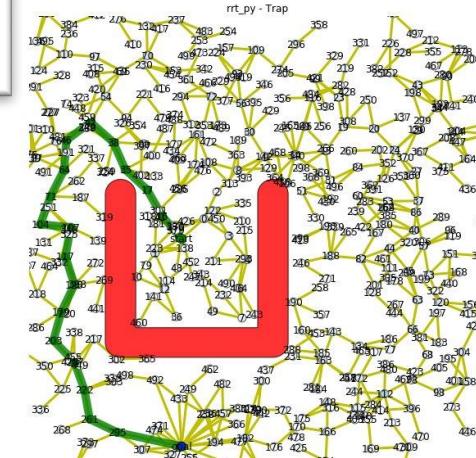
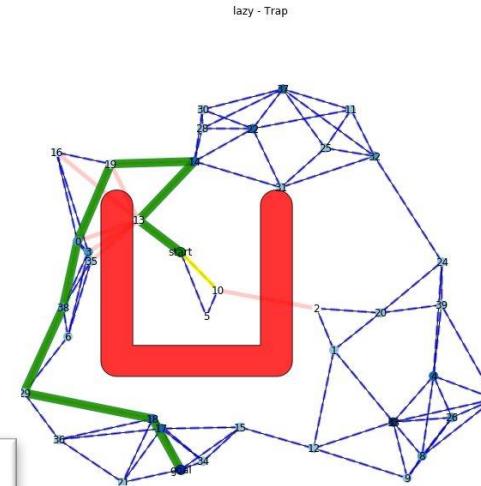
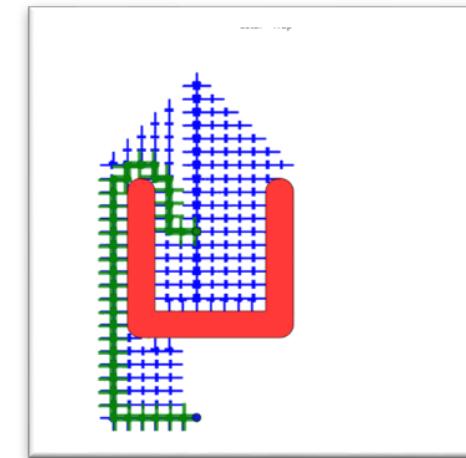
- Automatical motion generation
 - Mobile Robots
 - Robotic Manipulation *industrial*
 - Computer Games *LoL*
- Automatical validation
 - Assembly Planning

Approaches



- Exact Algorithms
 - Either find a solution or prove none exists
 - Very computationally expensive (*standard assembly* \rightarrow time ≈ 7 days)
 - Unsuitable for high-dimensional spaces
- Discrete Search
 - Divide space into a grid, use search based methods (A*) *Diskretisierung*
 - Good for vehicle planning (2D task space)
 - Unsuitable for high-dimensional spaces
- Sampling-based Planning
 - Sample the C-space, construct path from samples
 - Good for high-dimensional spaces
 - Weak completeness and optimality guarantees

C-space \rightarrow configuration space



K
A

What matters?



- Motion planning algorithms are judged on
 - Completeness
 - Optimality
 - Efficiency
 - Generality
- These vary in importance depending on the application

What matters: Completeness



- Will the algorithm solve all solvable problems?
- Will the algorithm return no solution for unsolvable problems?
- What if the algorithm is probabilistic? →

- For what application is completeness very important?

What matters: Optimality



- Will the algorithm generate the shortest path?
- Will the algorithm generate the least-cost path (for an arbitrary cost function)?
- Do we need optimality or is feasibility enough? → *depends on application*
- For what application is optimality very important? → *industrial optimization*

What matters: Efficiency



- How long does it take to generate a path for real-world problems?
- How does the run-time scale with dimensionality of the problem and complexity of the models?
- Is there a quality vs. Computation time trade-off?
- For what application is efficiency very important? → live applications like autonomous driving

What matters: Generality



- What types of problems can it solve?
- What types of problems cannot it solve?
- For what application is generality very important?

Applications



- Industrial Automation & Unmanned Warehouse
 - Completeness, Optimality ++
 - Efficiency, Generality –
- Autonomous Driving
 - Efficiency ++
 - Optimality, Generality +
 - Completeness --
- Robotic Surgery
 - Optimality ++
 - Completeness, Efficiency -
 - Generality --



+|

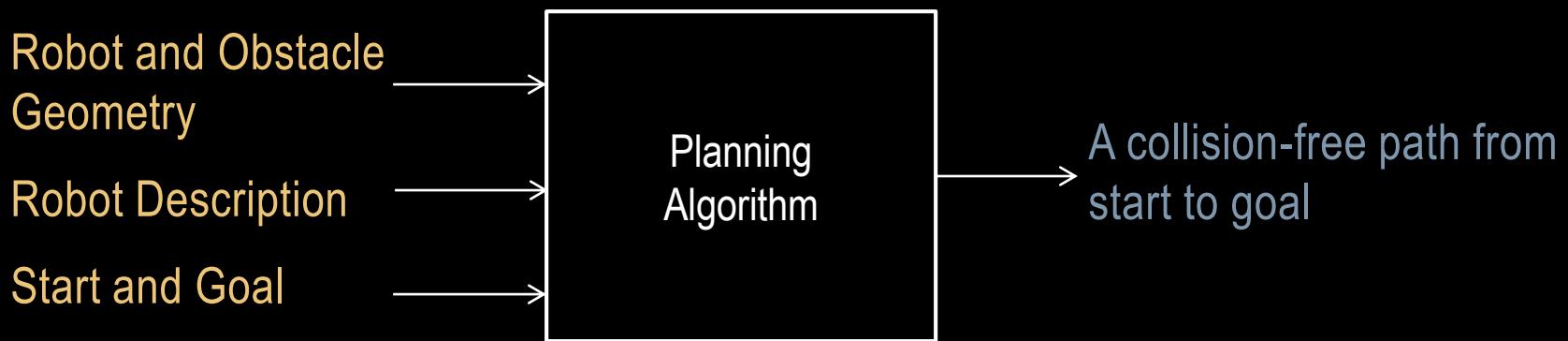
Thank you for your attention!

KA

Path Planning for Point Robots

Basic Problem Statement

- *Automatically compute a path for an object/robot that does not collide with obstacles.*



- Start simple:
 - The robot is a point that can move freely
 - The environment is 2D with polygonal obstacles

Methods

- Visibility graph
- Roadmaps
- Cell decomposition
- Potential fields

Framework

continuous representation

(configuration space formulation)



discretization

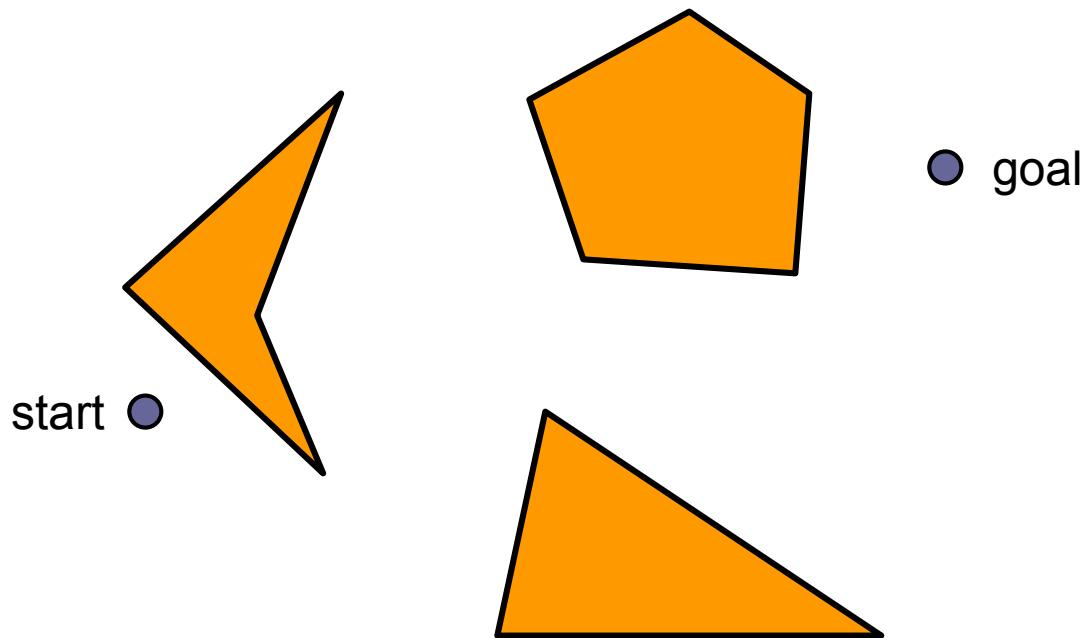
(random sampling, processing critical geometric events)



graph searching

(breadth-first, best-first, A*)

Continuous Representation



Framework

continuous representation



discretization

(random sampling, processing critical geometric events)

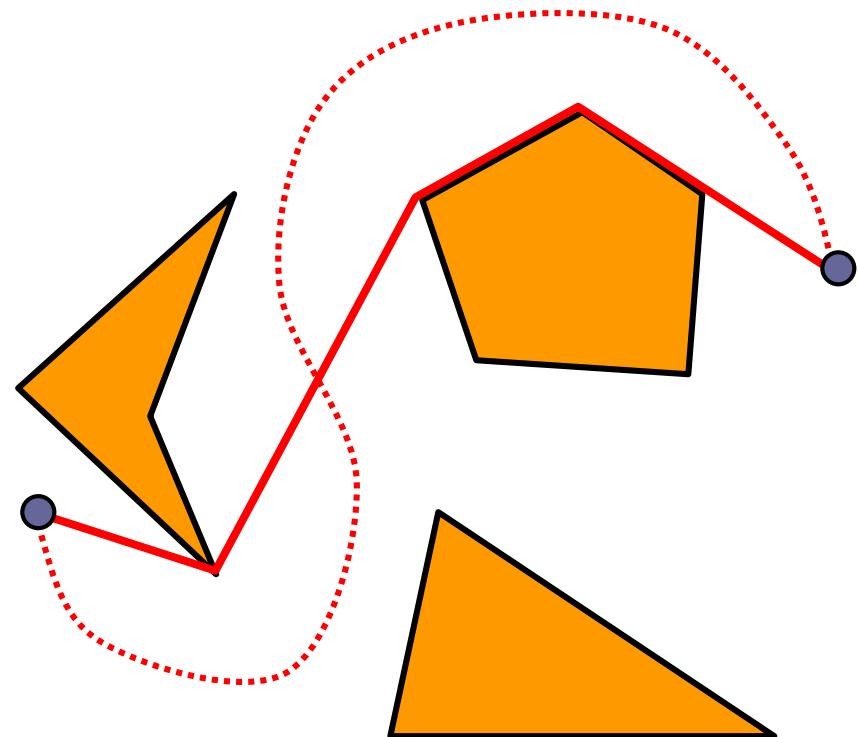


graph searching

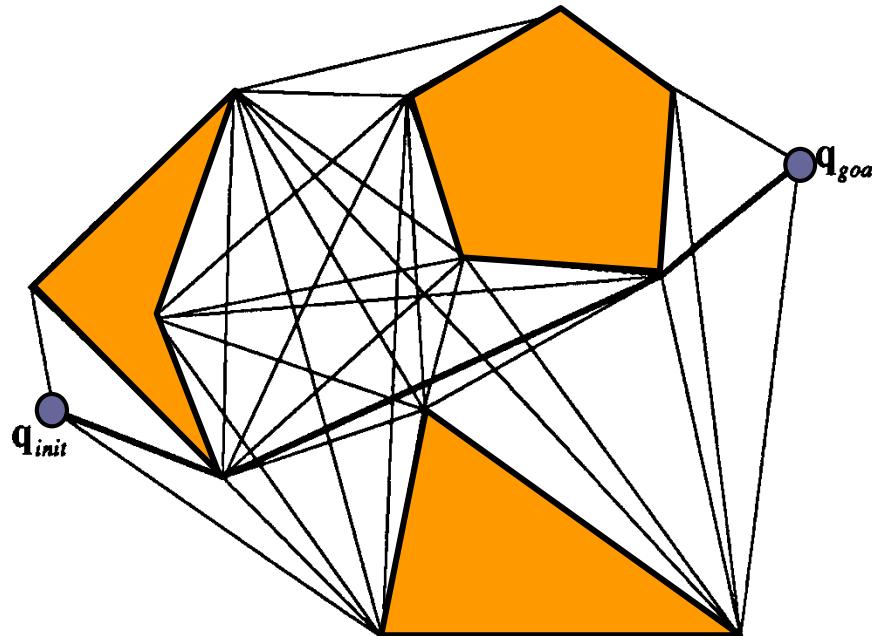
(breadth-first, best-first, A*)

Visibility graph method

- **Observation:** If there is a collision-free path between two points, then there is a piece-wise linear path that bends only at the obstacles vertices.
- **Why?**
Any collision-free path can be transformed into a piece-wise linear path that bends only at the obstacle vertices.



What is a visibility graph?



- A **visibility graph** is a graph such that
 - Nodes: q_{init} , q_{goal} , or an obstacle vertex.
 - Edges: An edge exists between nodes u and v if the line segment between u and v is an obstacle edge or it does not intersect the obstacles.

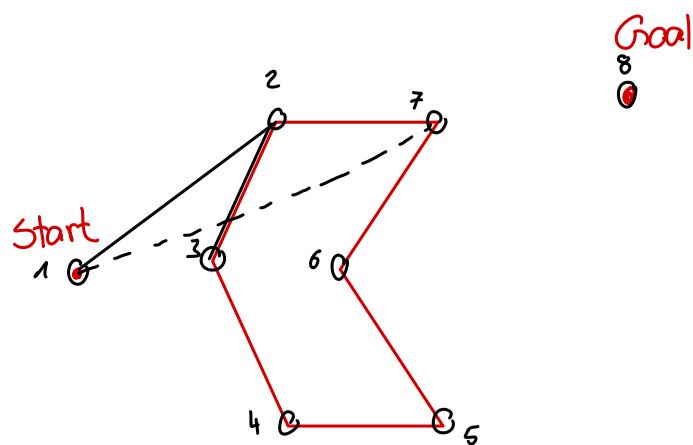
A simple algorithm for building visibility graphs

Input: q_{init} , q_{goal} , polygonal obstacles

Output: visibility graph G

```
1: for every pair of nodes  $u, v$ 
2:   if segment( $u, v$ ) is an obstacle edge then
3:     insert edge( $u, v$ ) into  $G$ ;
4:   else
5:     for every obstacle edge  $e$ 
6:       if segment( $u, v$ ) intersects  $e$ 
7:         go to (1);
8:       insert edge( $u, v$ ) into  $G$ .
```

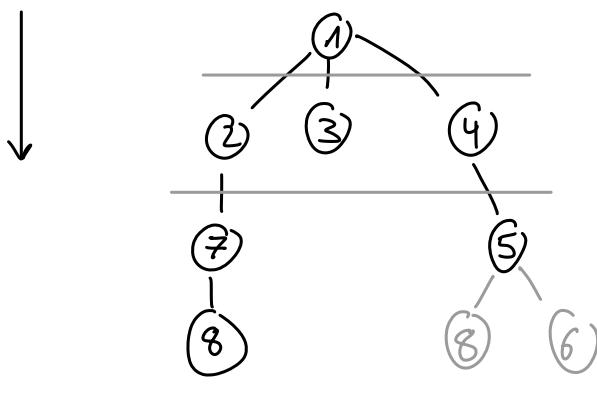
Breadth - first - Search



Q: 1 Startpunkt

→ alle Nachbarn bestimmen → 2; 3; 4

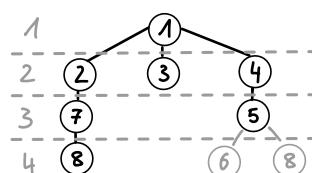
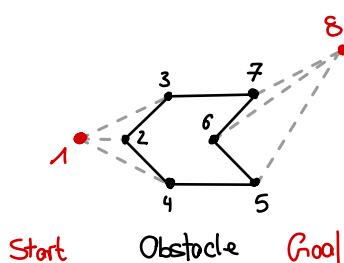
⋮



Verbindungen auf gleicher Ebene sind nicht erlaubt

~~z.B. 3 → 4~~

Reihenfolge relevant, erster gefunderner Pfad zum Zielpunkt ist der gefundene Pfad



Computational efficiency

```
1: for every pair of nodes u,v  $O(n^2)$   
2:   if segment(u,v) is an obstacle edge then  $O(n)$   
3:     insert edge(u,v) into G;  
4:   else  
5:     for every obstacle edge e  $O(n)$   
6:       if segment(u,v) intersects e  
7:         go to (1);  
8:     insert edge(u,v) into G.
```

- Simple algorithm $O(n^3)$ time
- More efficient algorithms
 - Rotational sweep $O(n^2 \log n)$ time
 - Optimal algorithm $O(n^2)$ time
- $O(n^2)$ space

Framework

continuous representation

(configuration space formulation)



discretization

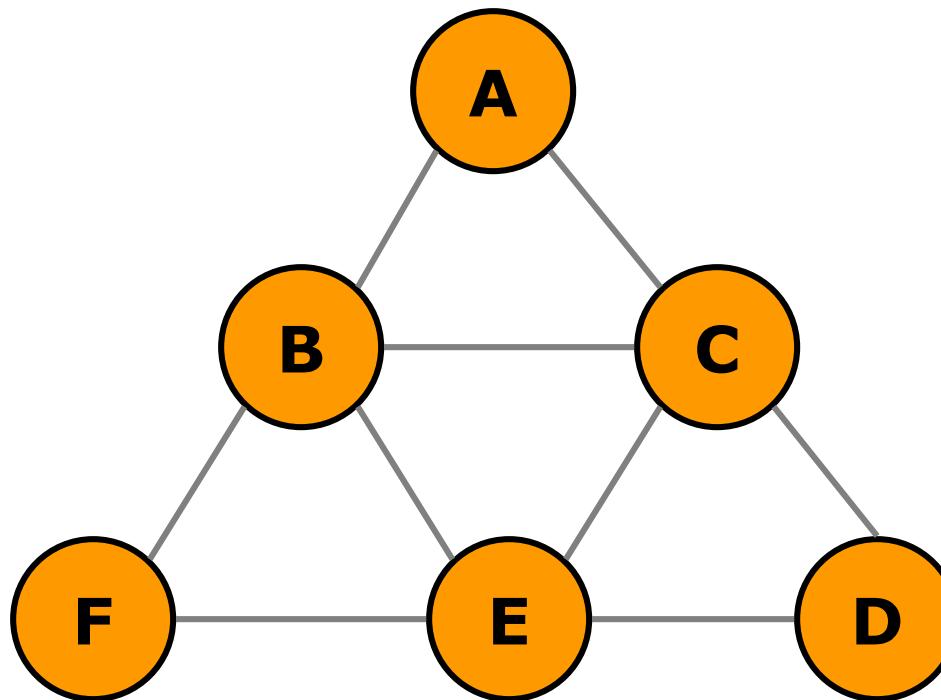
(random sampling, processing critical geometric events)



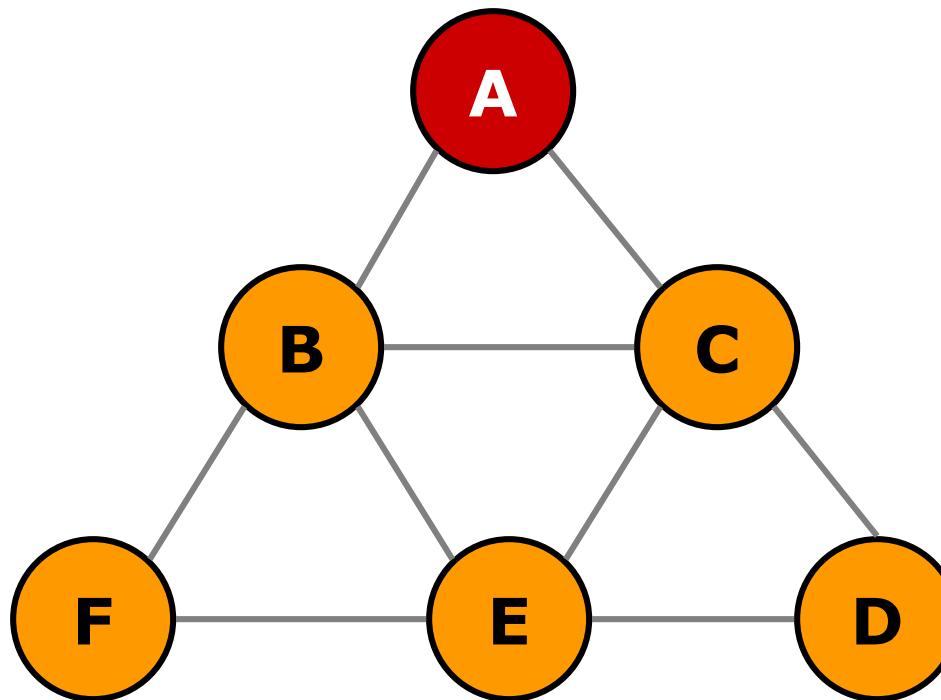
graph searching

(breadth-first, best-first, A^{*})

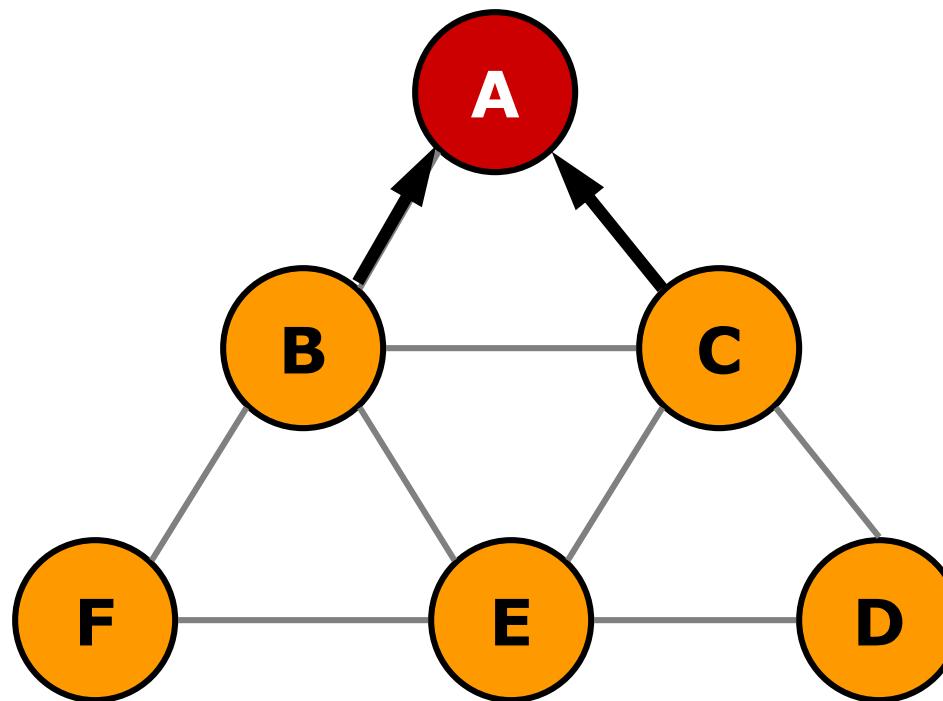
Breadth-first search



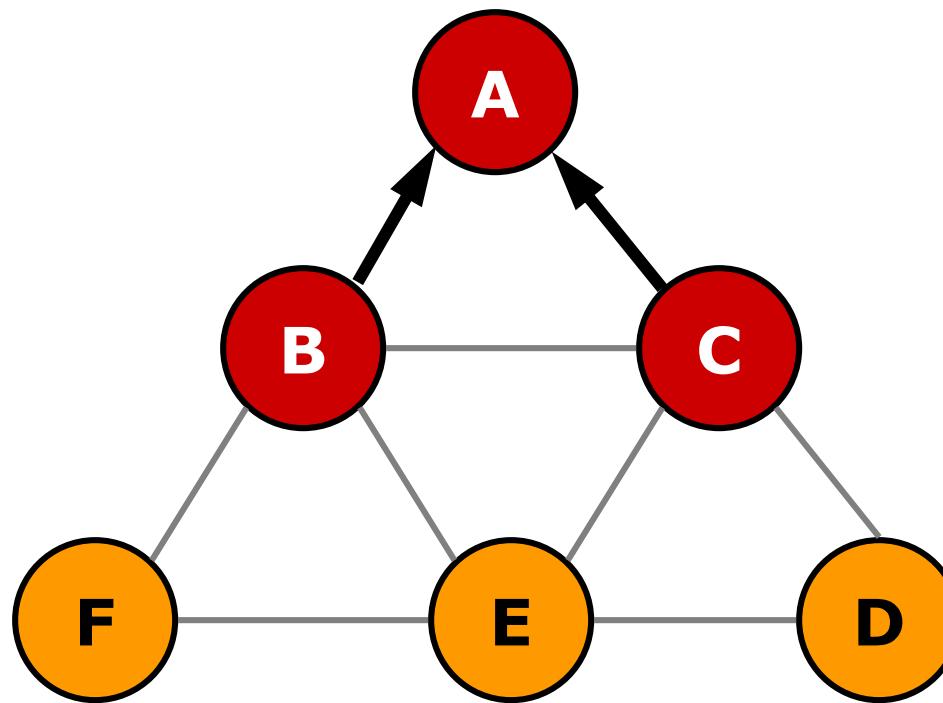
Breadth-first search



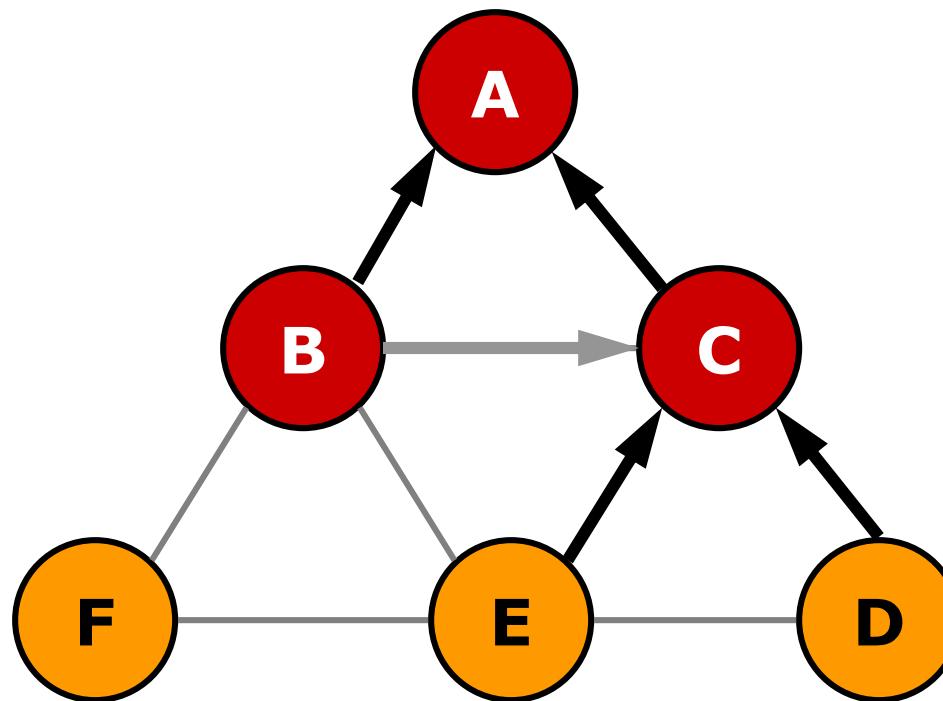
Breadth-first search



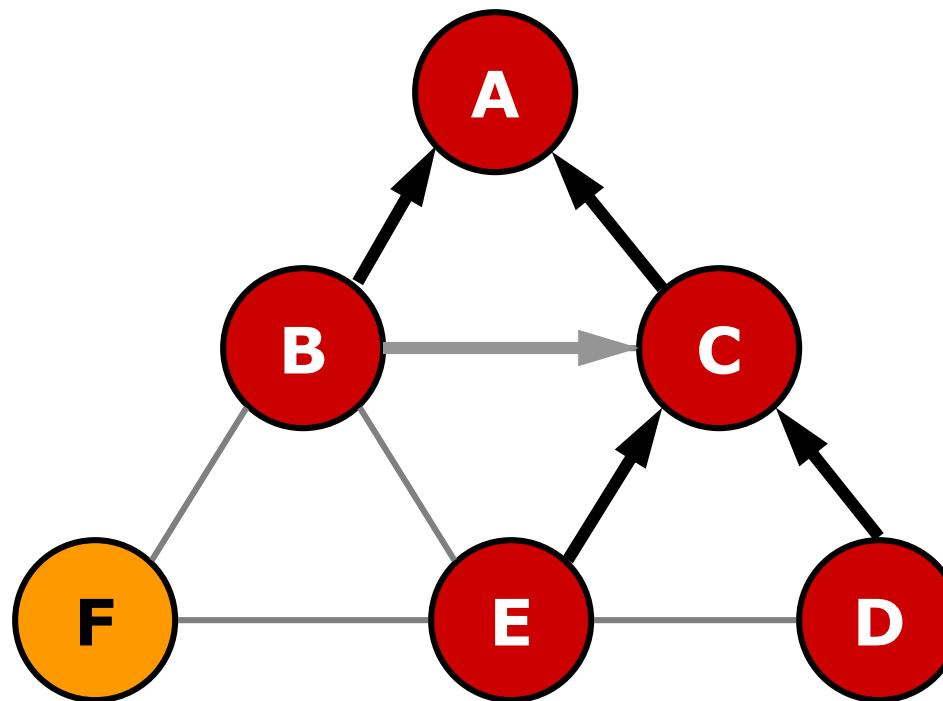
Breadth-first search



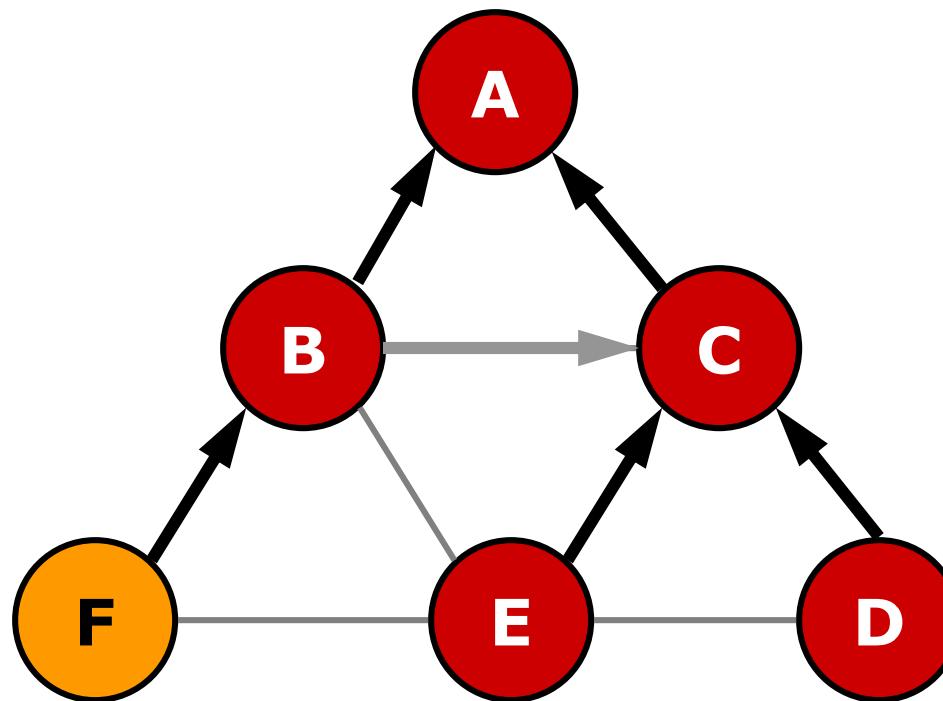
Breadth-first search



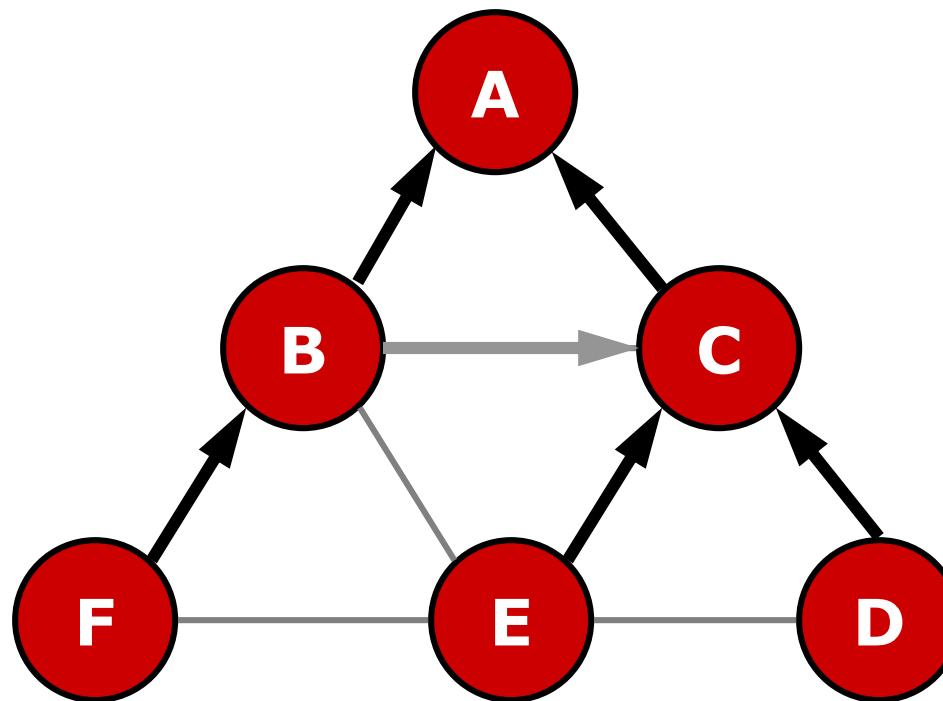
Breadth-first search



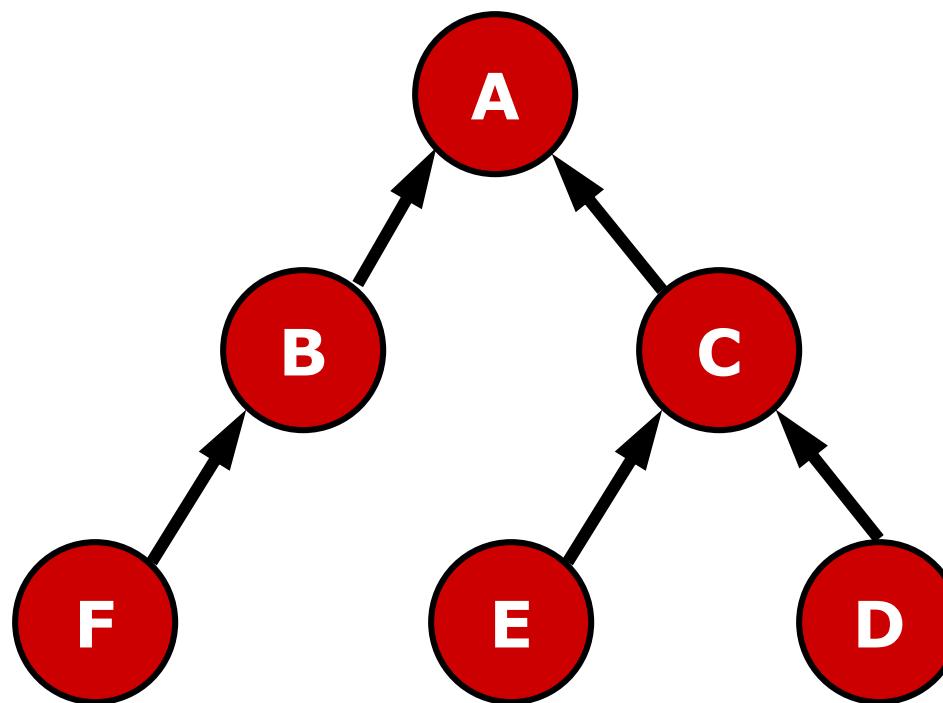
Breadth-first search



Breadth-first search



Breadth-first search



Breadth-first search

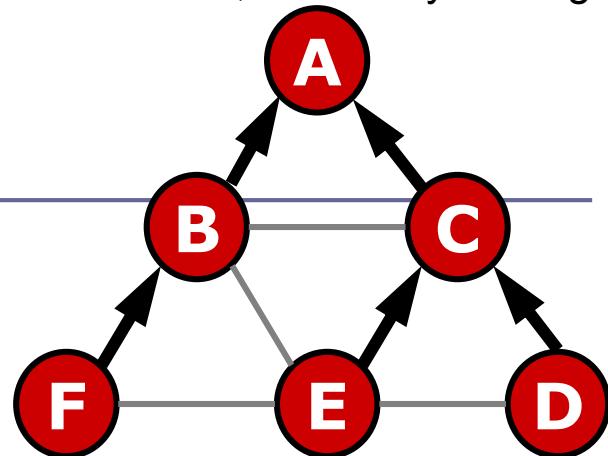
Input: q_{init} , q_{goal} , visibility graph G

Output: a path between q_{init} and q_{goal}

```

1: Q = new queue;
2: Q.enqueue( $q_{\text{init}}$ );
3: mark  $q_{\text{init}}$  as visited;
4: while Q is not empty
5:   curr = Q.dequeue();
6:   if curr ==  $q_{\text{goal}}$  then
7:     return curr;
8:   for each w adjacent to curr
10:    if w is not visited
11:      w.parent = curr;
12:      Q.enqueue(w)
13:      mark w as visited

```



Other graph search algorithms

- Depth-first
 - Explore newly-discovered nodes first
 - Not guaranteed to generate shortest path in the graph
- Dijkstra's Search
 - Find shortest paths to all nodes in the graph from the starts
- A*
 - Heuristically-guided search
 - Guaranteed to find shortest path
- We will learn a lot more about this in a future lecture

Framework

continuous representation



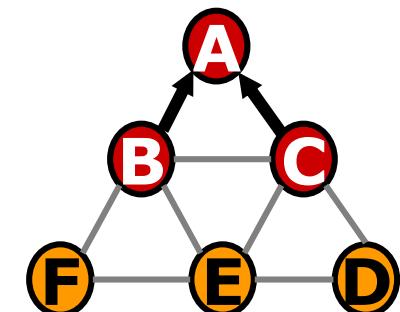
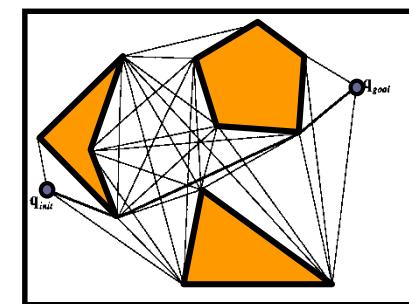
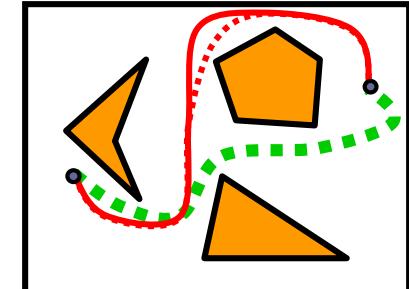
discretization

construct visibility graph



graph searching

breadth-first search



Summary

- Discretize the space by constructing visibility graph
- Search the visibility graph with breadth-first search

- How to perform the intersection test?

Computational efficiency

- Running time $O(n^3)$
 - Compute the visibility graph
 - Search the graph
 - An optimal $O(n^2)$ time algorithm exists.
- Space $O(n^2)$
- Can we do better?

Classic path planning approaches

□ Roadmap

Represent the connectivity of the free space by a network of 1-D curves

□ Cell decomposition

Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells

□ Potential field (not really path planning)

Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

Classic path planning approaches

□ **Roadmap**

Represent the connectivity of the free space by a network of 1-D curves

□ **Cell decomposition**

Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells

□ **Potential field**

Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

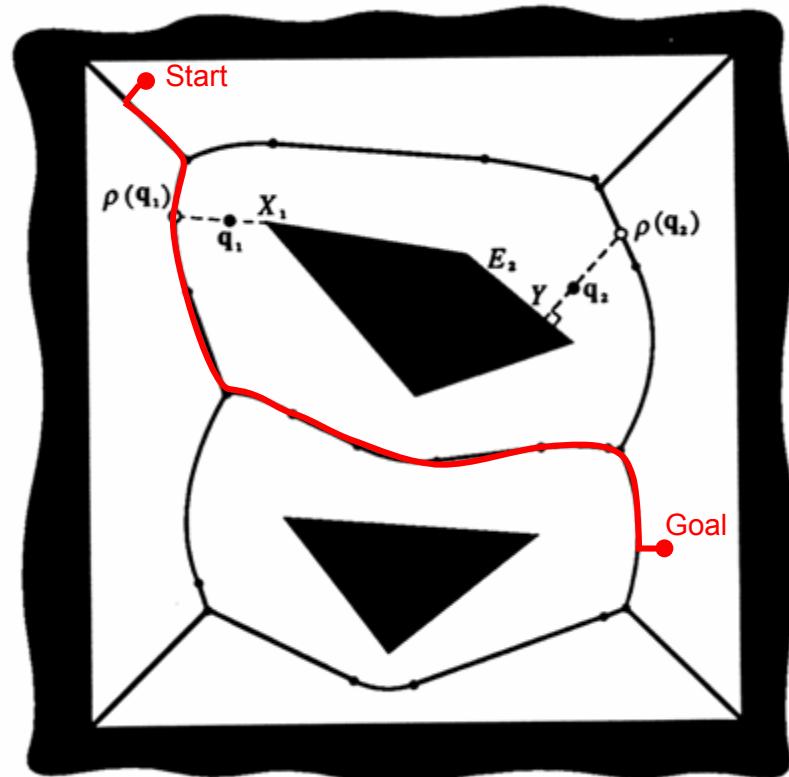
Roadmap

□ Visibility graph

Shakey Project, SRI [Nilsson, 1969]

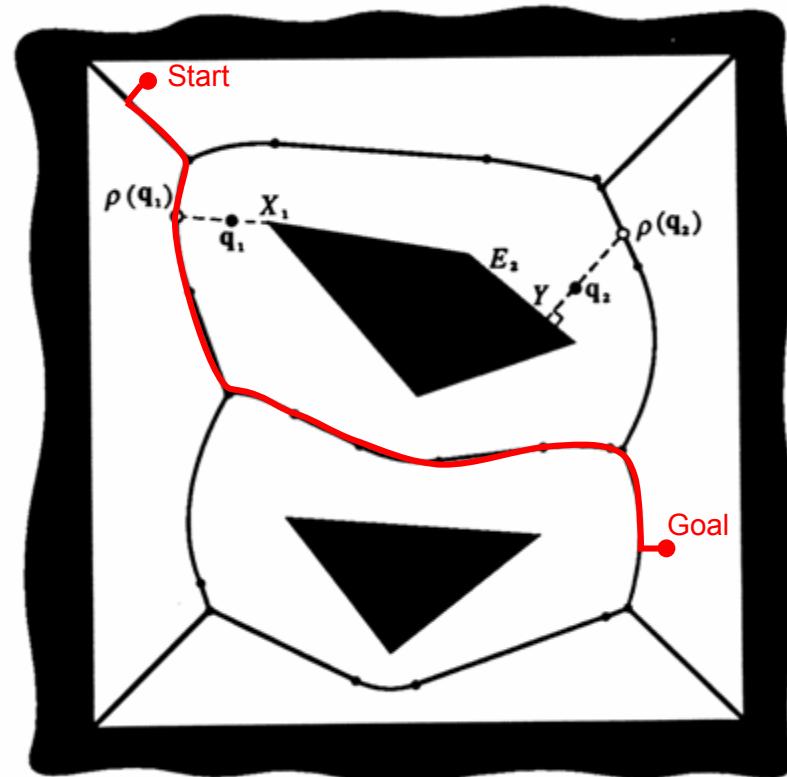
□ Voronoi diagram

Introduced by computational geometry researchers. Generate paths that maximizes clearance.



Voronoi diagram method

- Space $O(n)$
- Running time $O(n \log n)$
- Applicable mostly to 2-D configuration spaces



Classic path planning approaches

□ Roadmap

Represent the connectivity of the free space by a network of 1-D curves

□ Cell decomposition

Decompose the free space into **simple** cells and represent the connectivity of the free space by the adjacency graph of these cells

□ Potential field

Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function

Cell-decomposition methods

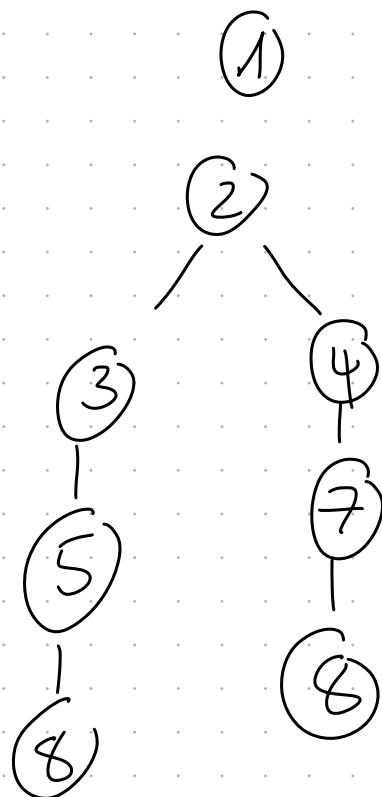
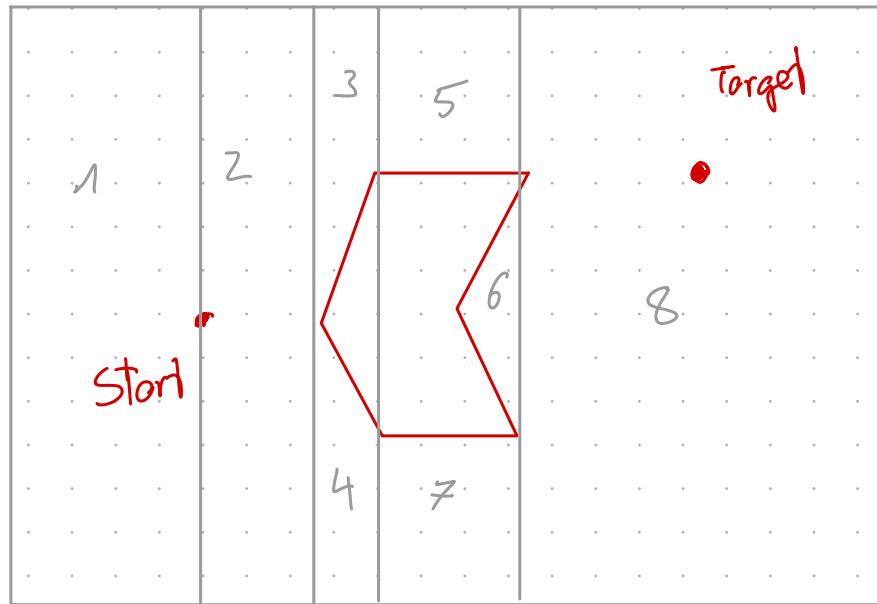
□ Exact cell decomposition

The free space F is represented by a collection of non-overlapping simple cells whose union **is exactly** F .

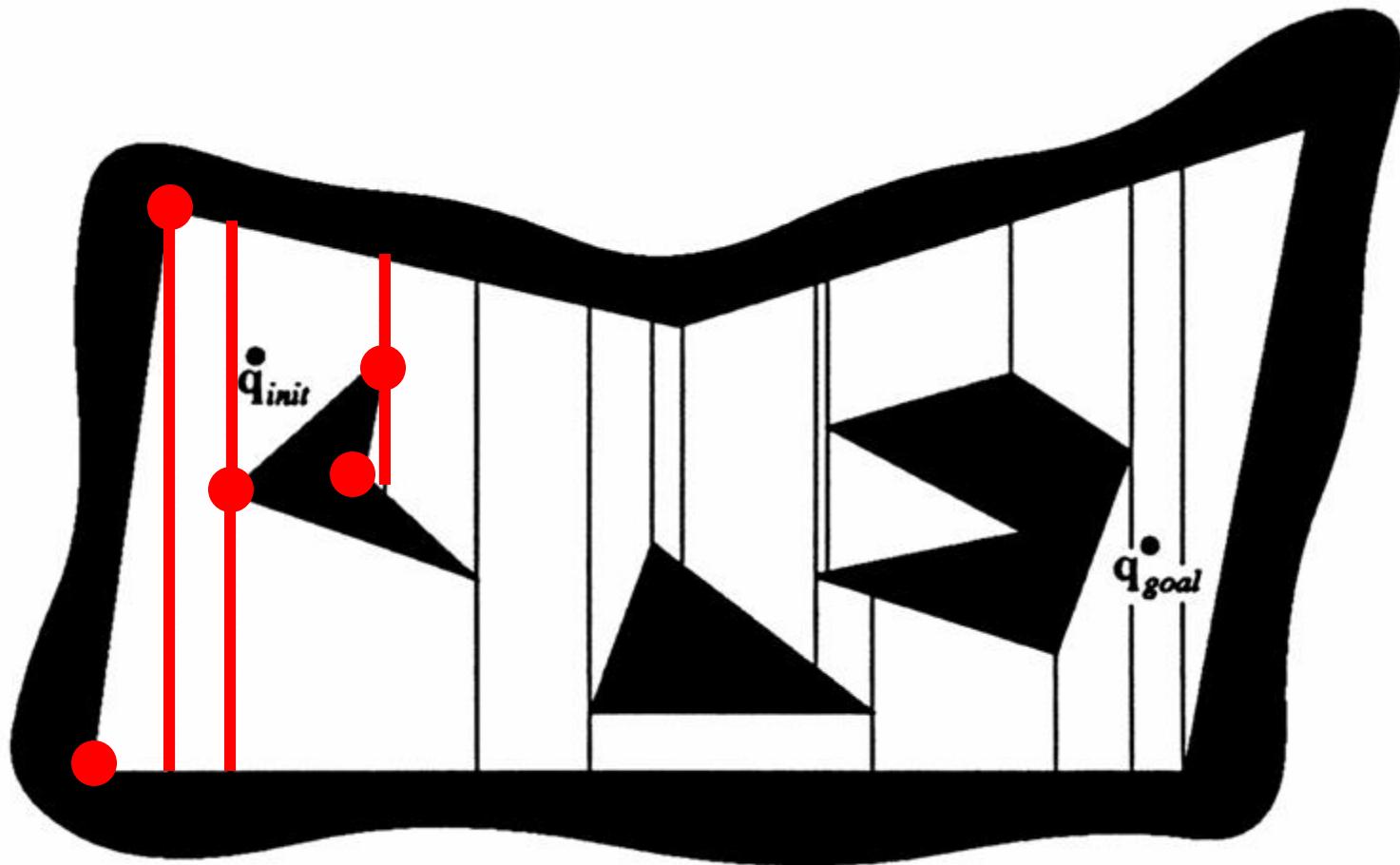
- Examples of cells: trapezoids, triangles

Cell-decomposition method

Exact cell decomposition



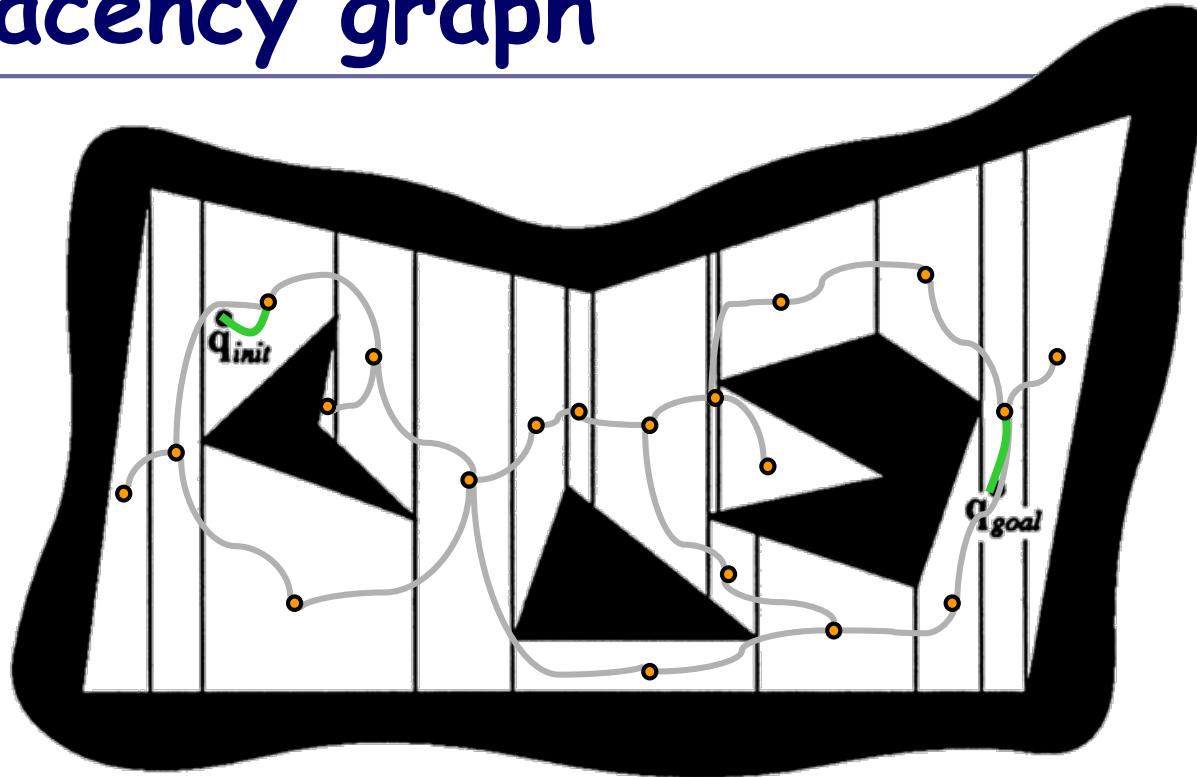
Trapezoidal decomposition



Computational efficiency

- Running time $O(n \log n)$ by planar sweep
- Space $O(n)$
- Mostly for 2-D configuration spaces

Adjacency graph



- **Nodes**: cells
- **Edges**: There is an edge between every pair of nodes whose corresponding cells are adjacent.
- A sequence of edges can be converted into a continuous path
 - This is easy to do when cells are convex. Why?

Framework

continuous representation



discretization

construct an adjacency graph of the cells



graph searching

search the adjacency graph

Break

Cell-decomposition methods

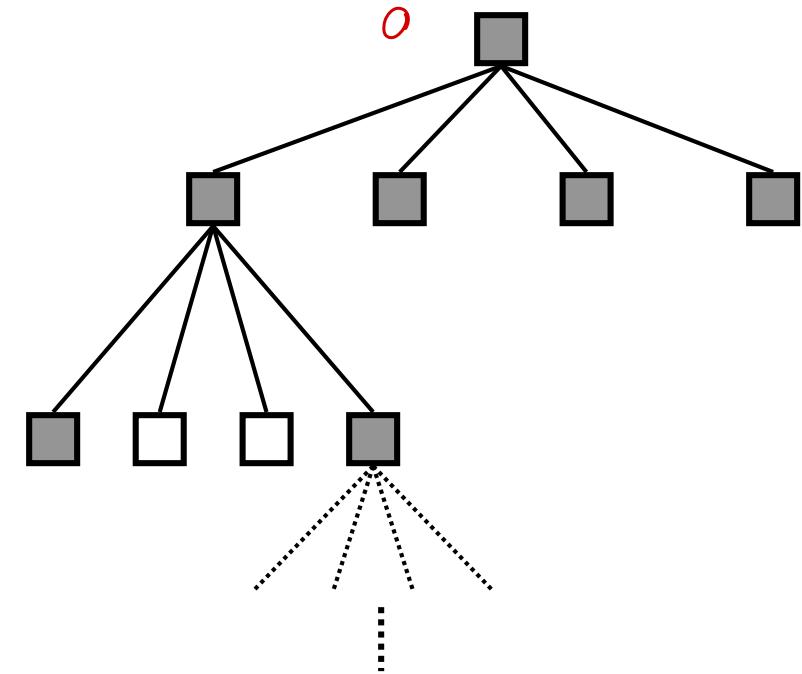
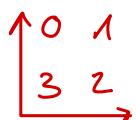
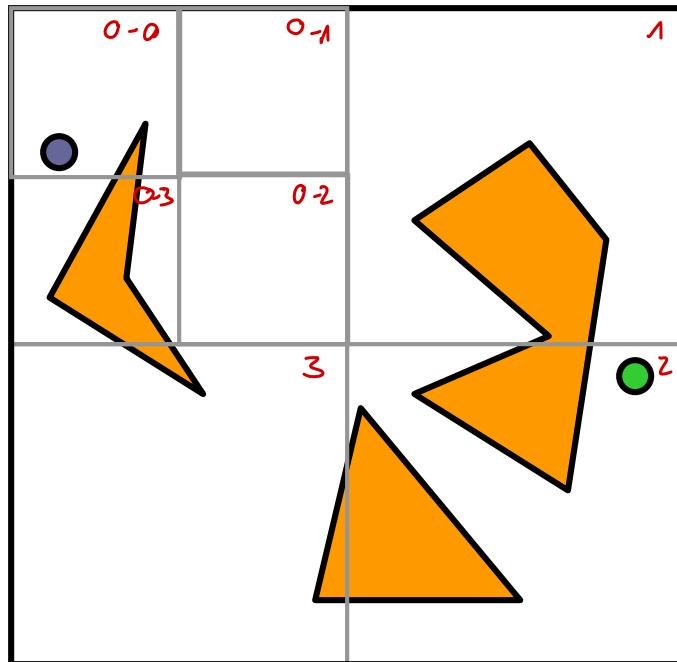
- Exact cell decomposition
- **Approximate cell decomposition**

The free space F is represented by a collection of non-overlapping cells whose union is **contained** in F .

- Cells usually have simple, regular shapes, e.g., rectangles, squares.
- Facilitate hierarchical space decomposition

keine Optimalität

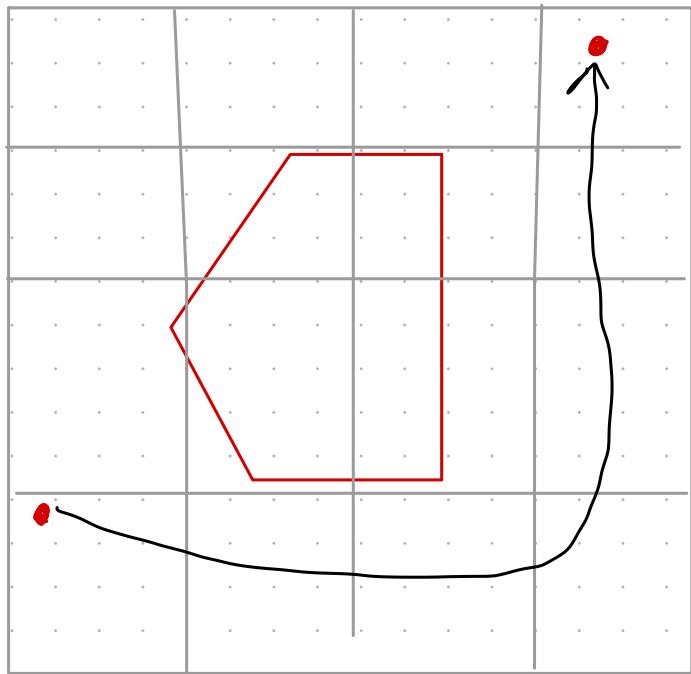
Quadtree decomposition



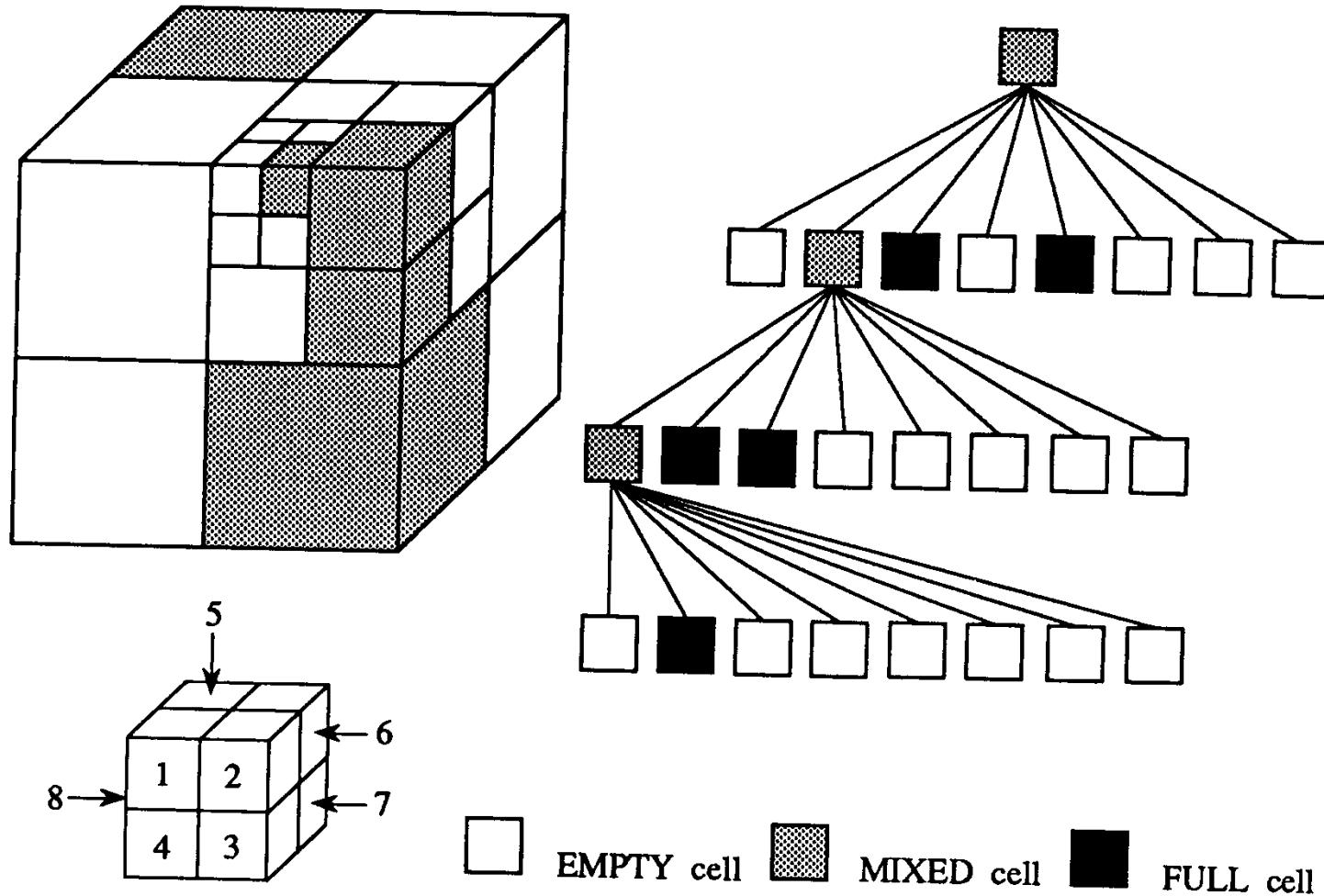
empty

mixed

full



Octree decomposition



Sketch of the algorithm

1. Decompose the free space F into cells.
2. Search for a sequence of **mixed** or **free** cells that connect the initial and goal positions.
3. Further decompose the mixed.
4. Repeat (2) and (3) until a sequence of **free** cells is found.

Classic path planning approaches

□ Roadmap

Represent the connectivity of the free space by a network of 1-D curves

□ Cell decomposition

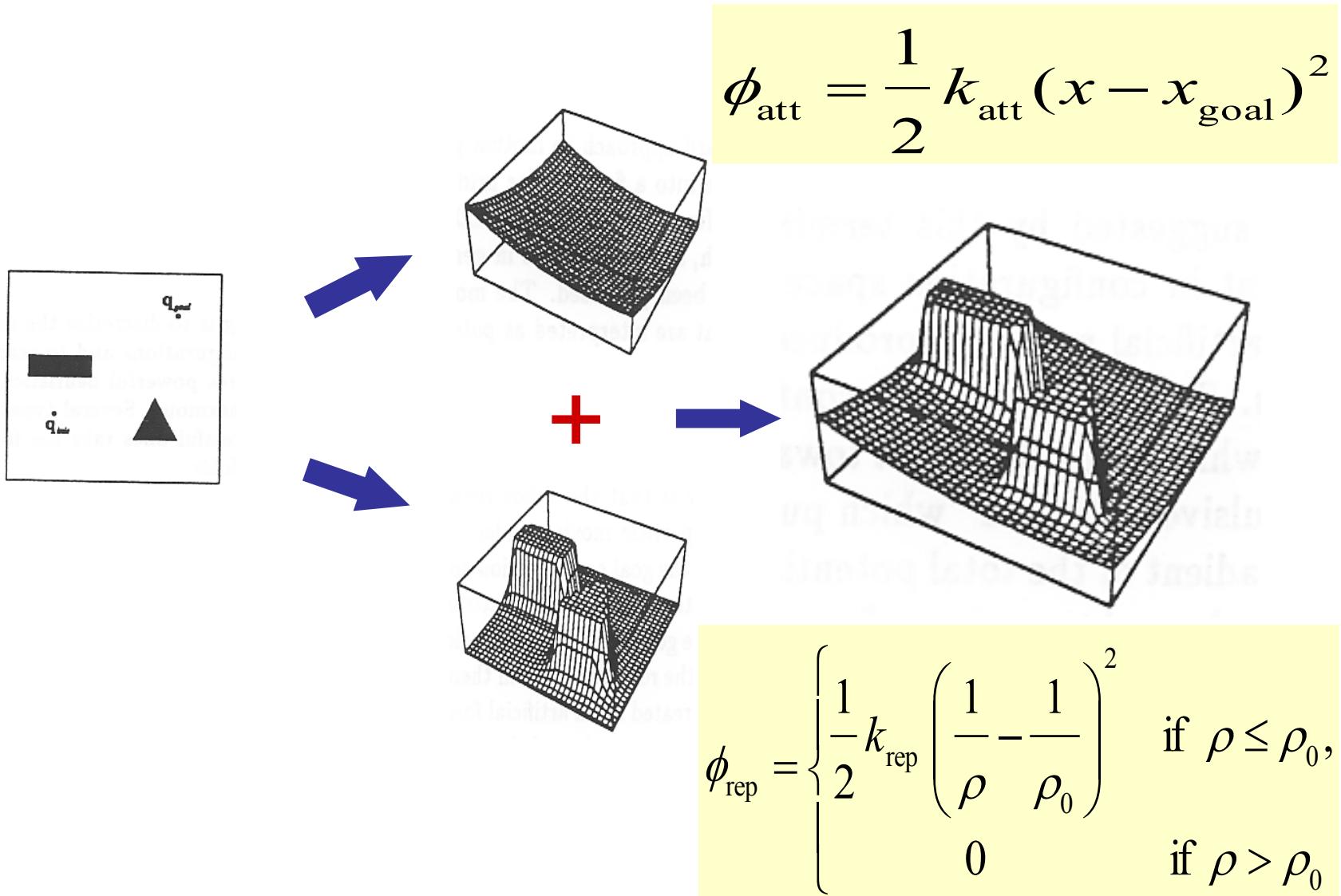
Decompose the free space into **simple** cells and represent the connectivity of the free space by the adjacency graph of these cells

□ Potential field

Define a potential function over the free space that has a global minimum at the goal and follow the steepest descent of the potential function → to global minimum

problem → Local minima

Algorithm in pictures



Attractive & repulsive fields

$$F_{\text{att}} = -\nabla \phi_{\text{att}} = -k_{\text{att}}(x - x_{\text{goal}})$$

$$F_{\text{rep}} = -\nabla \phi_{\text{rep}} = \begin{cases} k_{\text{rep}} \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{if } \rho \leq \rho_0, \\ 0 & \text{if } \rho > \rho_0 \end{cases}$$

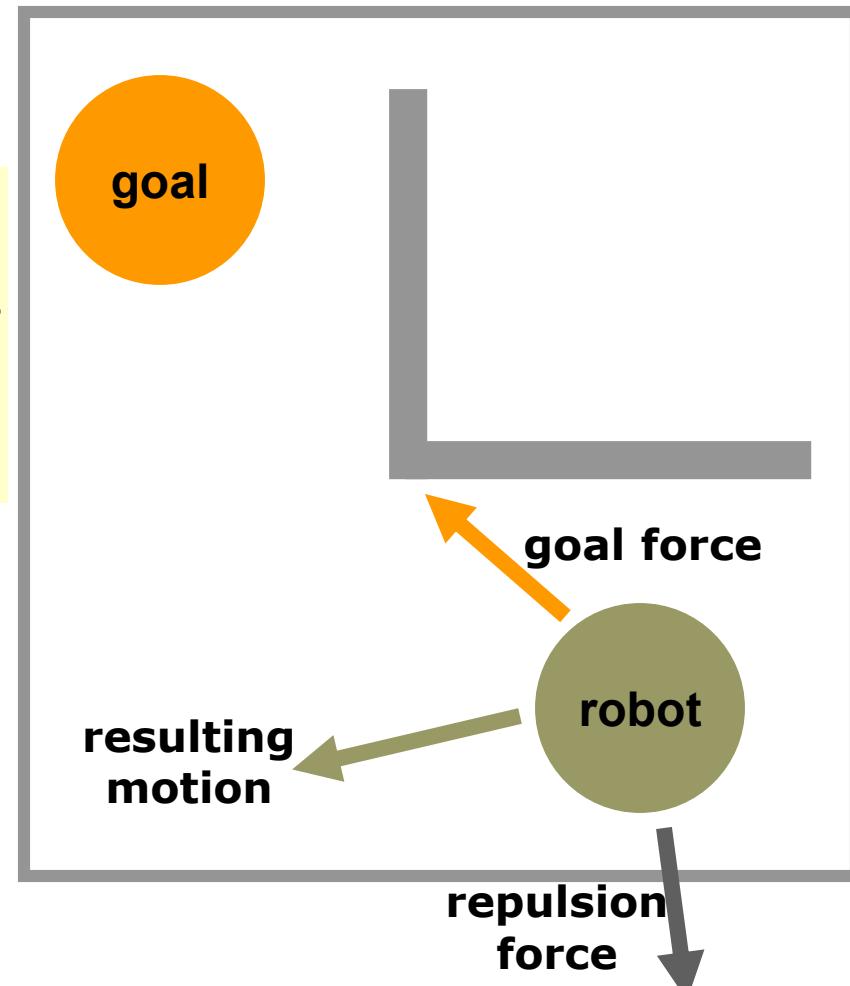
$k_{\text{att}}, k_{\text{rep}}$: positive scaling factors

x : position of the robot

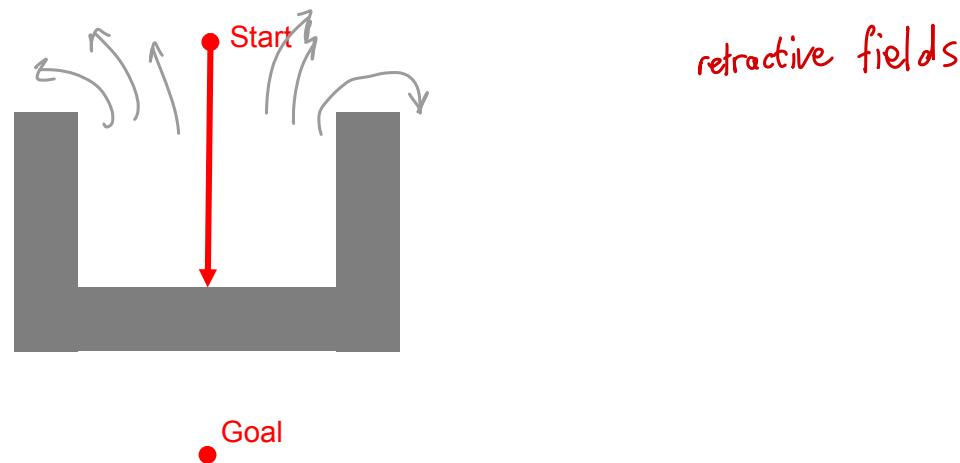
ρ : distance to the obstacle

ρ_0 : distance of influence

[Khatib, 1986]



Local minima



□ What can we do?

- Escape from local minima by taking ~~random walks~~
- Build an ideal potential field – navigation function – that does not have local minima

Potential field vs. Navigation function

- A **potential field** is a scalar function over the free space.
 - Initially proposed for real-time collision avoidance [Khatib, 1986].
 - To navigate, the robot applies a force proportional to the negated gradient of the potential field.
- A **navigation function** is an ideal potential field that
 - has global minimum at the goal
 - has no local minima
 - grows to infinity near obstacles
 - is smooth

How to make a Navigation Function?

- Place a regular grid G over the configuration space
- Compute the potential field over G
- Search G using a best-first algorithm with potential field as the heuristic function

Question

- Can such an ideal potential field be constructed efficiently in general?

Completeness

- A **complete motion planner** always returns a solution when one exists and indicates that no such solution exists otherwise.
 - Is the visibility graph algorithm complete? ✓
 - Is the exact cell decomposition algorithm complete? ✓
 - Is the potential field algorithm complete? ✗

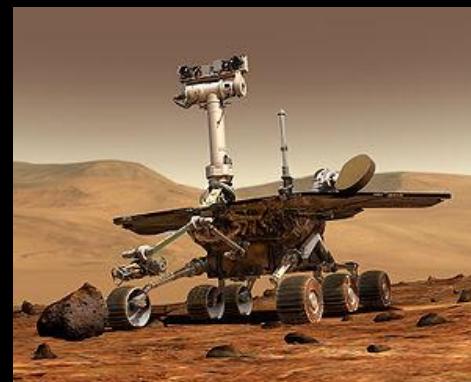
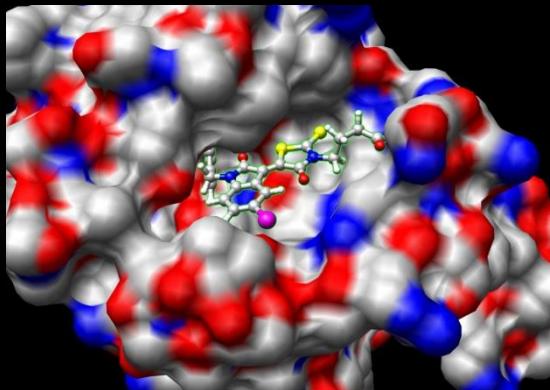
Homework

- Read Chapter 4.0-4.3

Configuration Space

Last time...

- We learned about how to plan paths for a point



- Real-world robots are complex, often articulated bodies
- What if we invented a space where the robots could be treated as points?

Outline

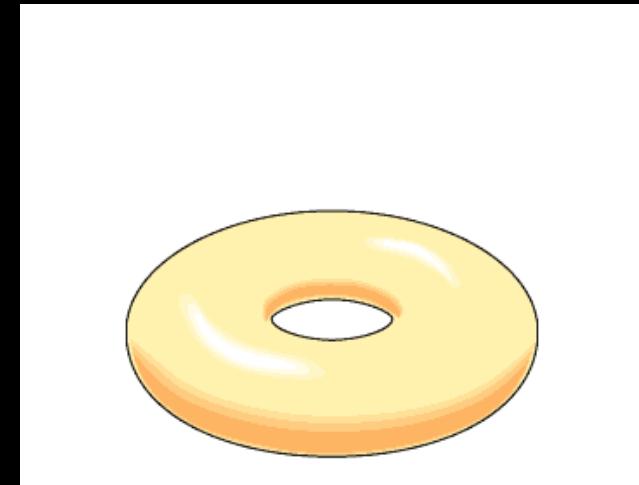
- Topology basics
- Configuration Space
- Obstacles
- Metrics

Basic Sets

- **Open set** – A set with no boundary. Every point in the set has an open neighborhood which is also in the set.
 - In \mathbf{R}^n , this open neighborhood is called an open ball:
$$B(x, \rho) = \{x' \in \mathbf{R}^n \mid \|x' - x\| < \rho\}$$
 - The set $X \subseteq \mathbf{R}^n$ is open if
$$\exists B(x, \rho) \subseteq X, \rho > 0 \quad \forall x \in X$$
 - Example: $X = \{x \in \mathbf{R} \mid 1 < x < 5\}$
- **Closed Set** – A set with a boundary. A closed set is the complement of some open set and vice versa.
 - Example: $X = \{x \in \mathbf{R} \mid 1 \leq x \leq 5\}$
 - What is the complement of this set?

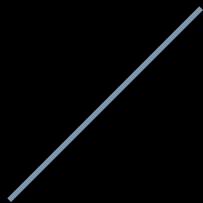
Topological Spaces

- A set X is called a **topological space** if there is a collection of open subsets of X for which the following hold:
 1. The union of any number of open sets is an open set.
 2. The intersection of a finite number of open sets is an open set.
 3. Both X and \emptyset are open sets.
- Two topological spaces X and Y are **homeomorphic** if there is a bijective (one-to-one and onto) function $f: X \rightarrow Y$ and both f and f^{-1} are continuous.
 - Intuitively, you can think of f as a continuous function that warps X into Y
 - f is called a **homeomorphism**

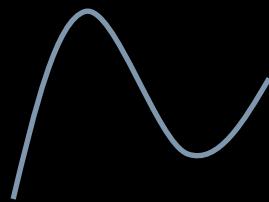


Homeomorphisms

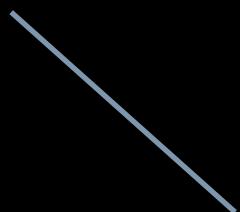
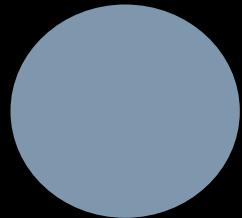
- Which are homeomorphic?



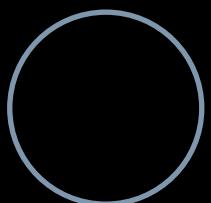
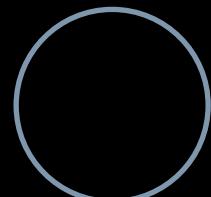
and



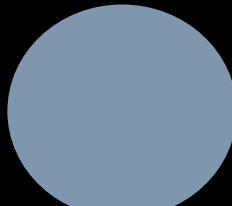
and



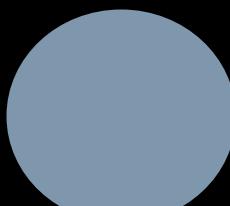
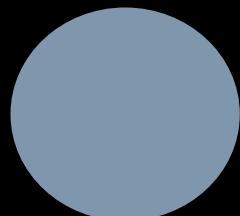
and



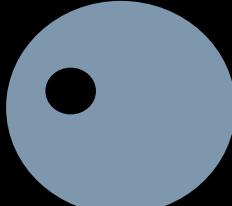
and



and



and



Homeomorphisms can not add or remove holes!

Common Topological Spaces

- The real numbers: \mathbf{R}^1
 - Number of dimensions
 - Symbolic name for the space
- The unit circle: $S^1 = \{(x, y) \in \mathbf{R}^2 \mid x^2 + y^2 = 1\}$
 - This is NOT the same as a *disc* $\{(x, y) \in \mathbf{R}^2 \mid x^2 + y^2 \leq 1\}$

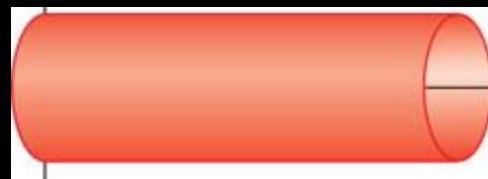
Cartesian Product

- Can make more complex spaces using the **Cartesian product**: Every $x \in X, y \in Y$ makes an $(x, y) \in X \times Y$. For example:

$$\mathbf{R}^1 \times \mathbf{R}^1 = \mathbf{R}^2$$

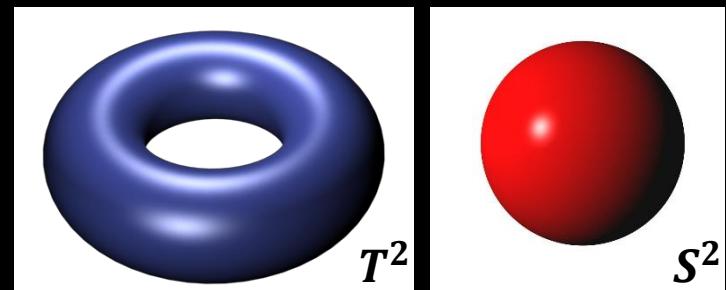


- What is $\mathbf{R}^1 \times \mathbf{S}^1$?
 - A hollow cylinder



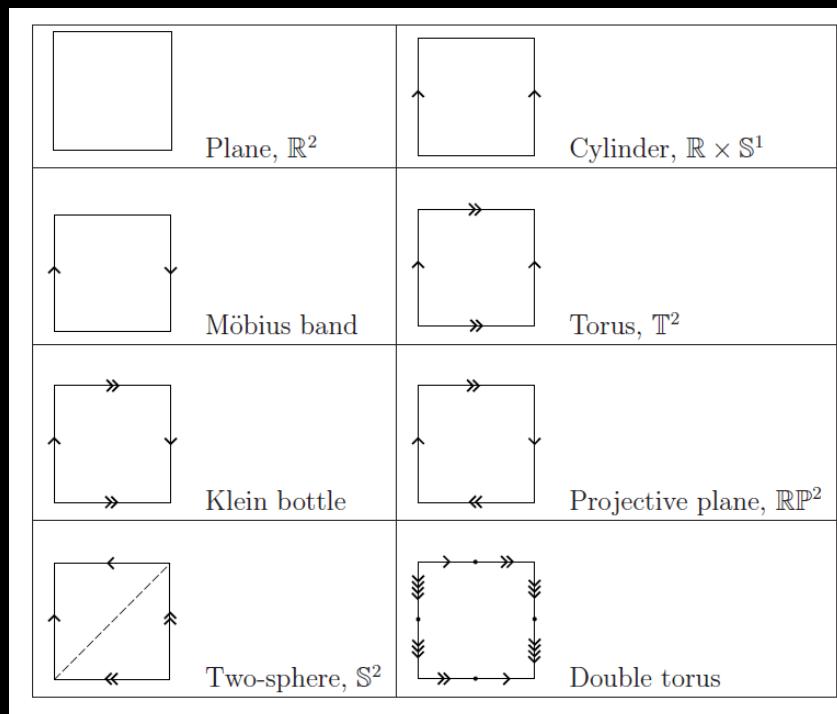
- BE CAREFUL: Results of Cartesian product are not always obvious. This is NOT the same thing as adding up exponents in multiplication.

- Example: What is $\mathbf{S}^1 \times \mathbf{S}^1$?
 - Hint: try to visualize the shape
 - This creates a 2D Taurus, $\mathbf{S}^1 \times \mathbf{S}^1 = \mathbf{T}^2$
 - \mathbf{S}^2 is a sphere
- Also, $\mathbf{S}^1 = \mathbf{T}^1$ but $\mathbf{S}^{n>1} \neq \mathbf{T}^{n>1}$



More complex spaces

- Can create more complex topological spaces by Cartesian products and “gluing” boundaries:



Configuration Space

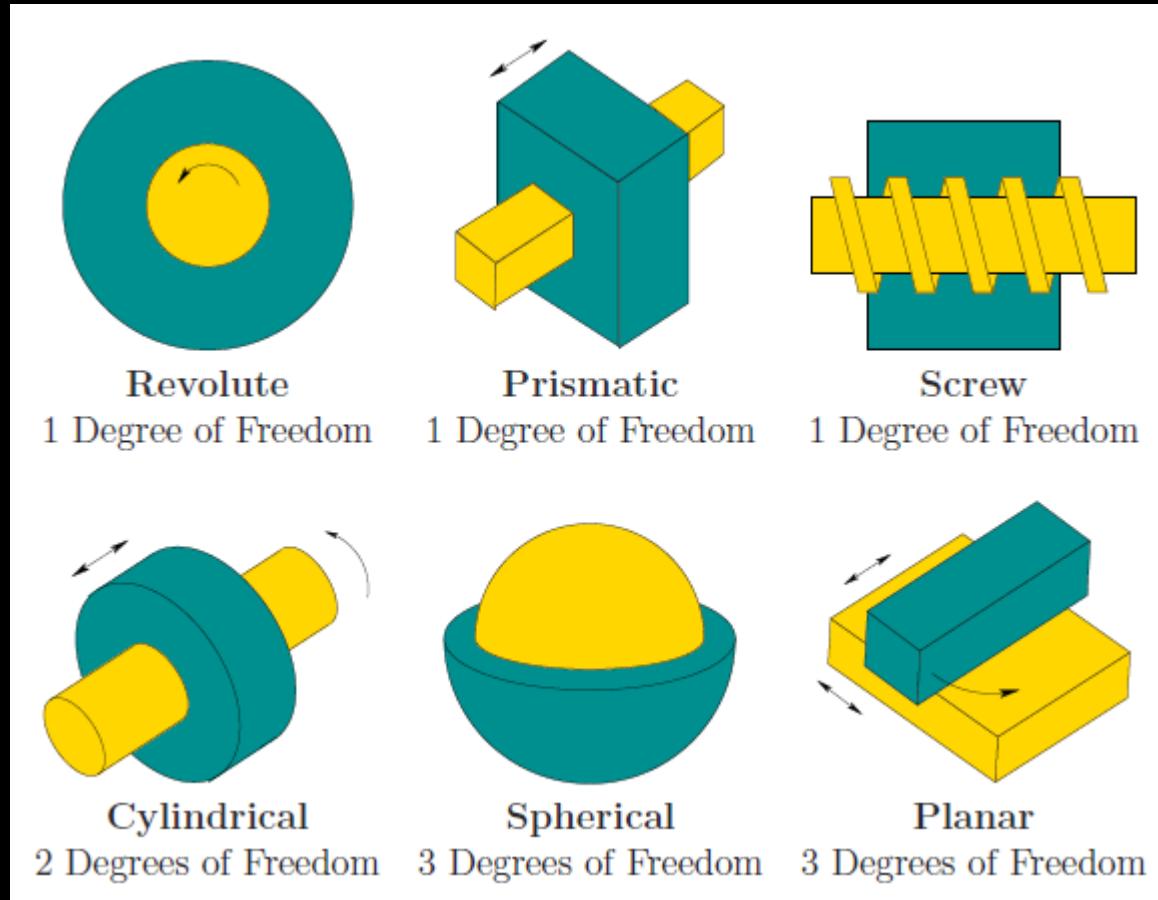
Definitions

- The **configuration** of a moving object is a specification of the position of **every** point on the object.
 - A configuration q is usually expressed as a vector of the Degrees of Freedom (DOF) of the robot

$$q = (q_1, q_2, \dots, q_n)$$

- The **configuration space** C is the set of all possible configurations. Usually this is a topological space.
 - A configuration q is a point in C

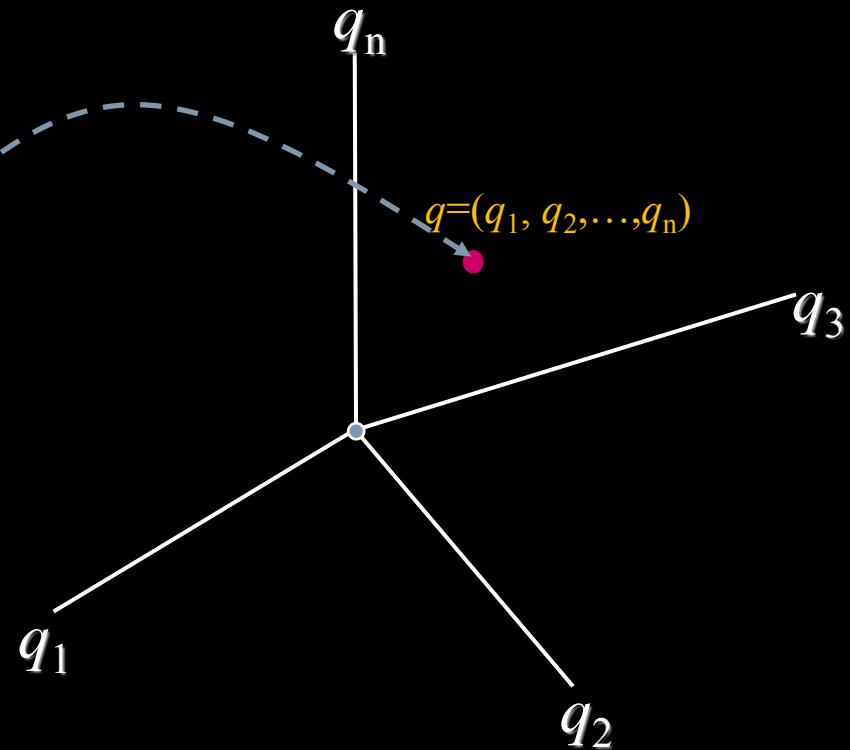
Degrees of Freedom



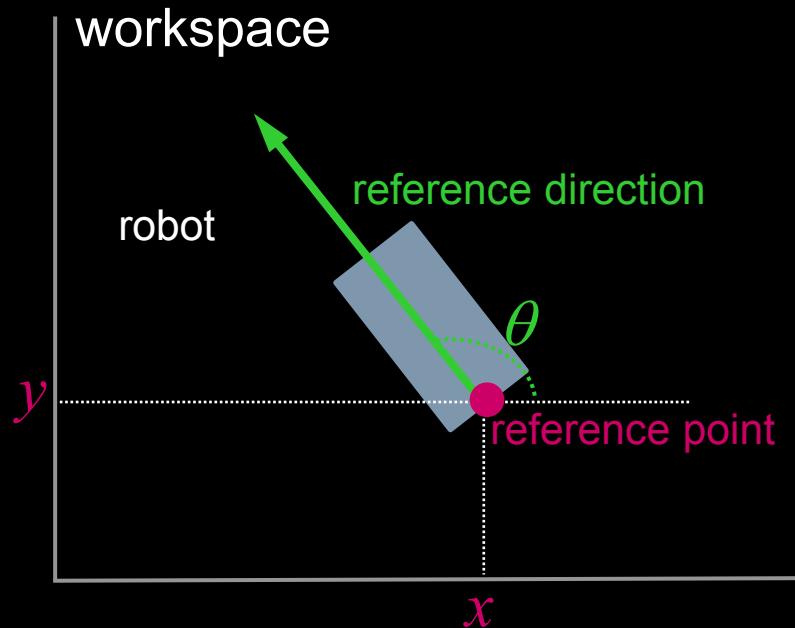
What is the topology of each of these (assume no joint limits)?

Configuration Space

- The **dimension of a configuration space** is the **minimum** number of DOF needed to specify the configuration of the object completely.



Example: A Rigid 2D Mobile Robot



- 3-parameter specification: $q = (x, y, \theta)$ with $\theta \in [0, 2\pi)$.
 - 3D configuration space
 - Topology: $\text{SE}(2) = \mathbb{R}^2 \times S^1$

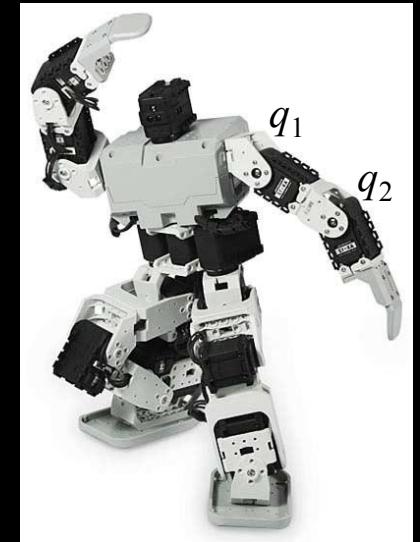
Example: Rigid Robot in 3D workspace



- $q = (\text{position, rotation}) = (x, y, z, \text{????})$
- 3 representations for rotation
 - Euler Angles
 - Transform Matrices
 - Quaternions
- No matter the representation, rotation in 3D is 3 DOF
- C-space dimension: 6
- Topology: $\text{SE}(3) = \mathbb{R}^3 \times \text{SO}(3)$

Configuration Space for Articulated Objects

- An **articulated** object is a set of rigid bodies connected by joints.
- For articulated robots (arms, humanoids, etc.) the DOF are usually the joints of the robot
- Example: For a single revolute joint with no joint limits, what is the topology?
- What is the topology if it does have joint limits?



$$q = (q_1, q_2, \dots, q_n)$$

Number of DOF = n

>30 dof

Paths and Trajectories

- A **path** in C is a continuous curve connecting two configurations q_{start} and q_{goal} :

$$\tau : s \in [0,1] \rightarrow \tau(s) \in C$$

such that $\tau(0) = q_{start}$ and $\tau(1) = q_{goal}$.

- A **trajectory** is a path parameterized by time:

$$\tau : t \in [0, T] \rightarrow \tau(t) \in C$$

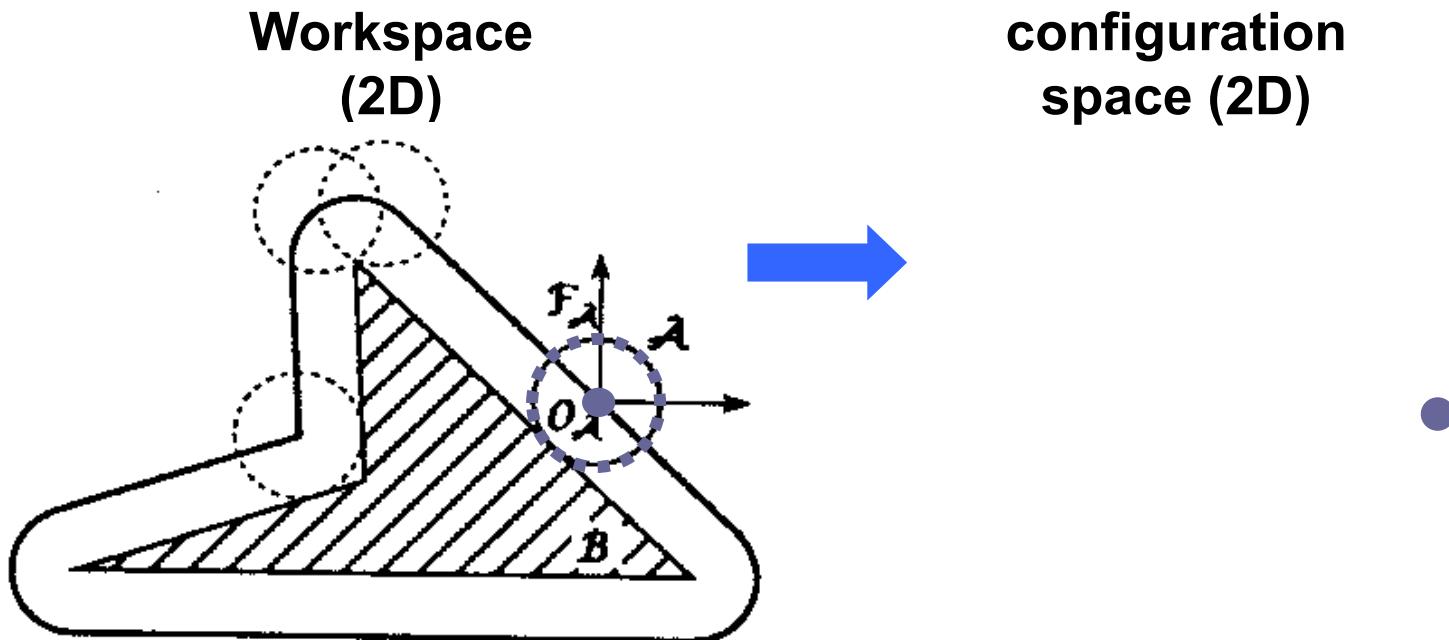
Obstacles in C-space

- A configuration q is collision-free, or **free**, if the robot placed at q does not intersect any obstacles in the workspace.
- The **free space** C_{free} is a subset of C that contains all free configurations.
- A configuration space obstacle (C_{obs}) is a subset of C that contains all configurations where the robot collides with workspace obstacles or with itself (self-collision).

How do we compute C_{obs} ?

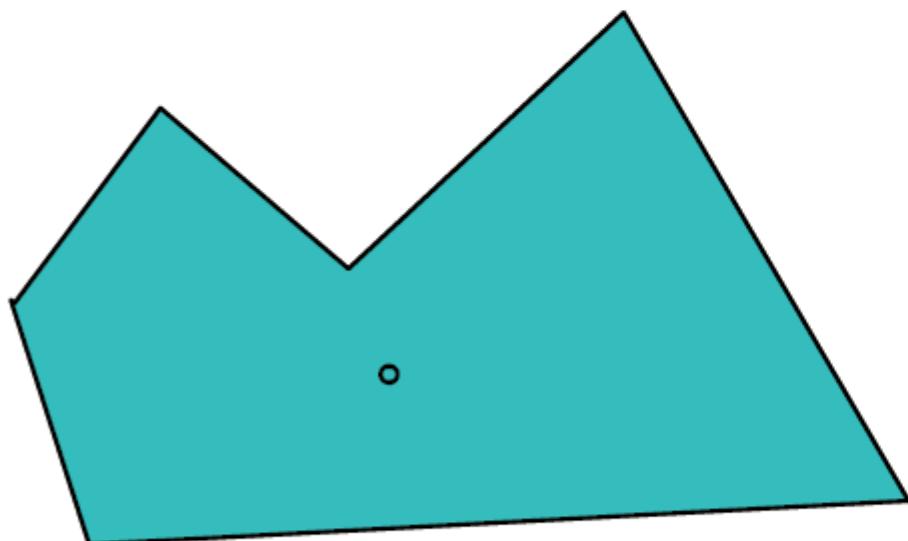
- Start with simple case: 2D translating robot
- Input:
 - Polygonal robot
 - Polygonal obstacle in environment
- Output:
 - Configuration space polygonal obstacle

Example: Disc in 2D workspace

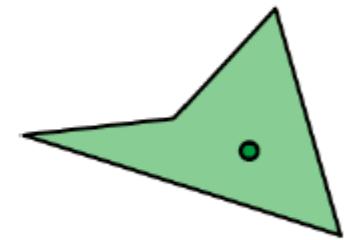


Minkowski Sum

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



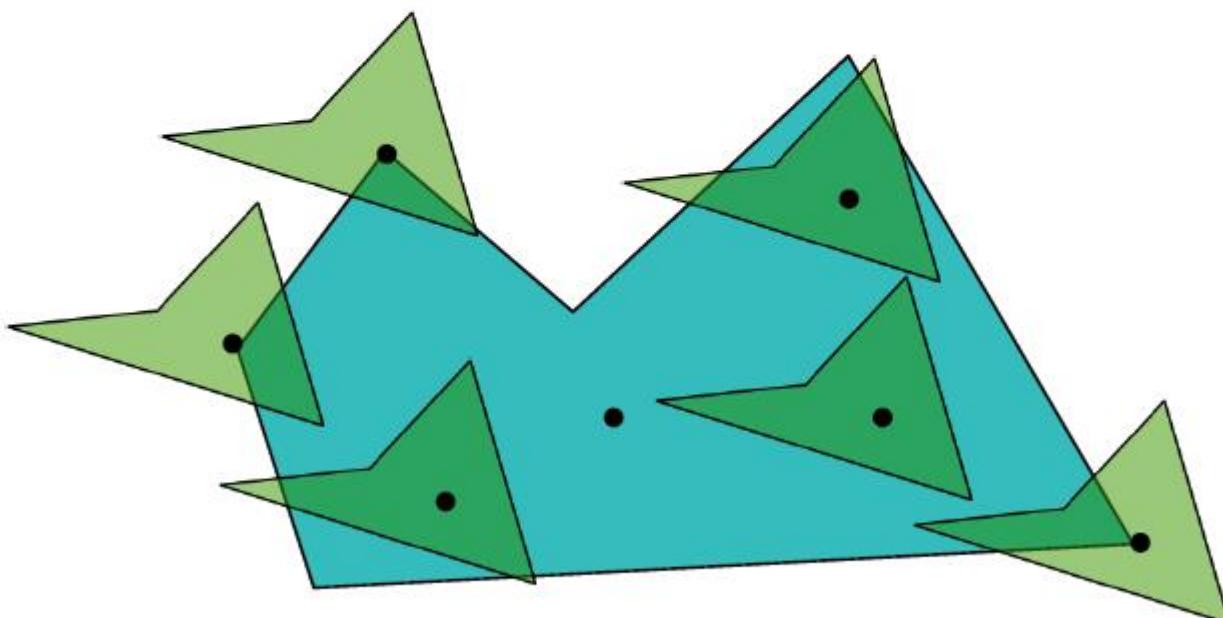
A



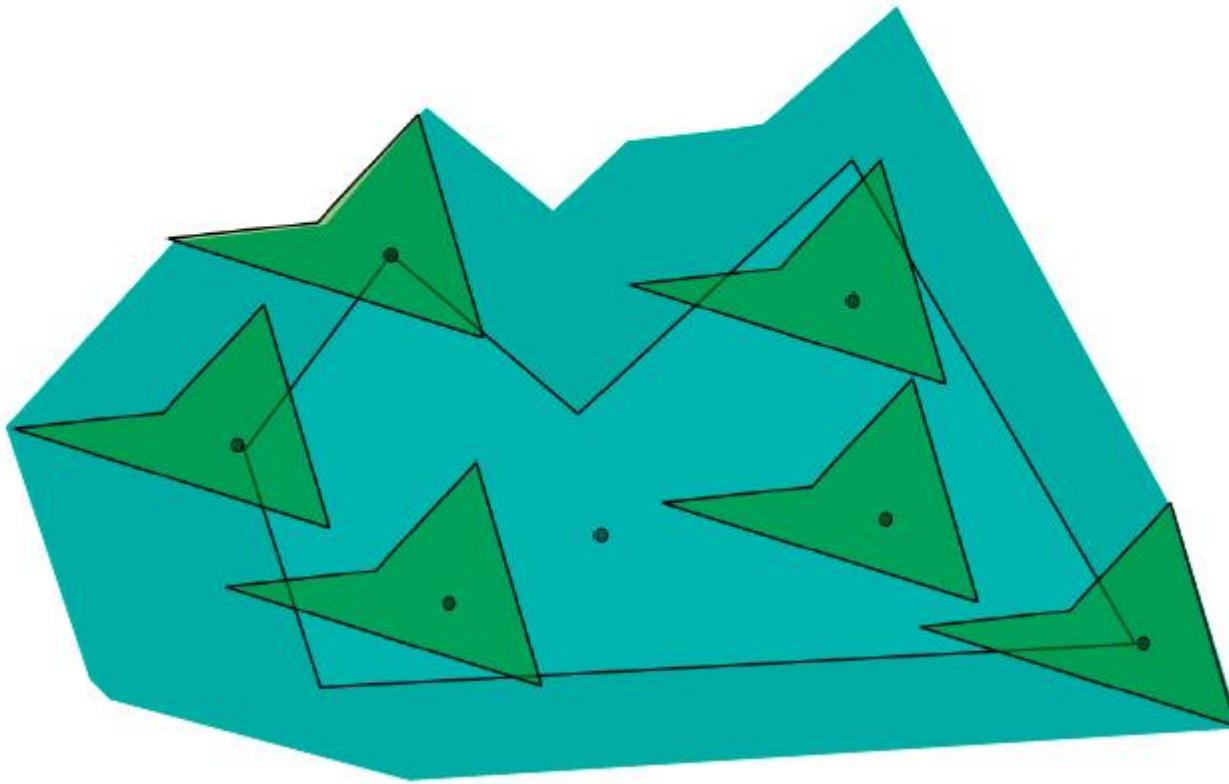
B

Minkowski Sum

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



Minkowski Sum



Minkowski Sum

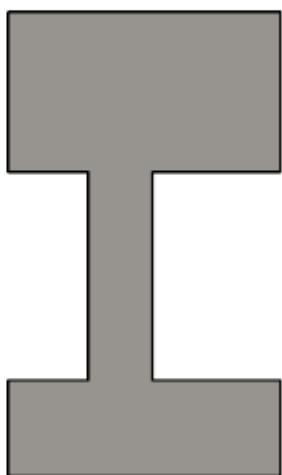
$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$



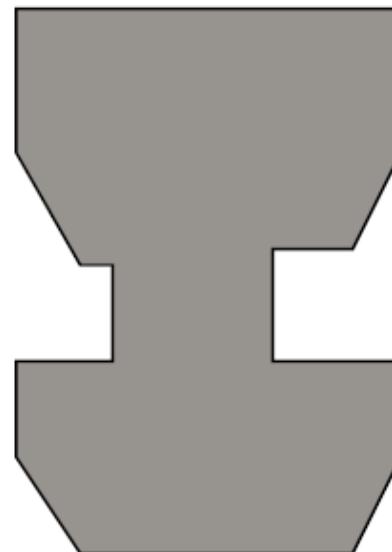
Configuration Space Obstacle

C-obstacle is $O \oplus -\mathcal{R}$

This means use $-b$ instead of b



$$\oplus - \triangle =$$



Obstacle
 O

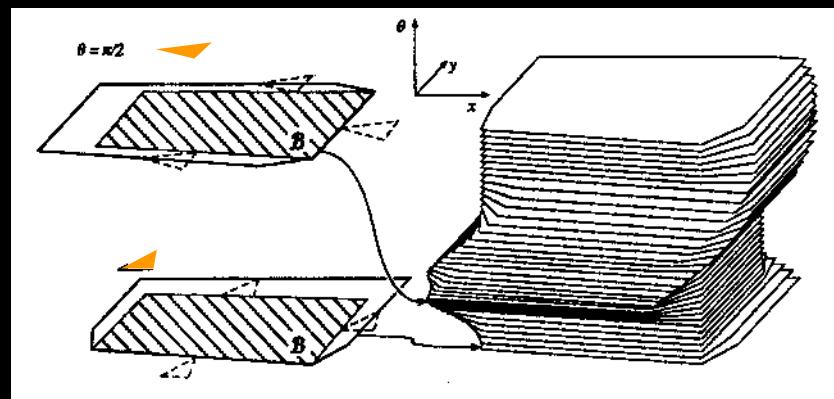
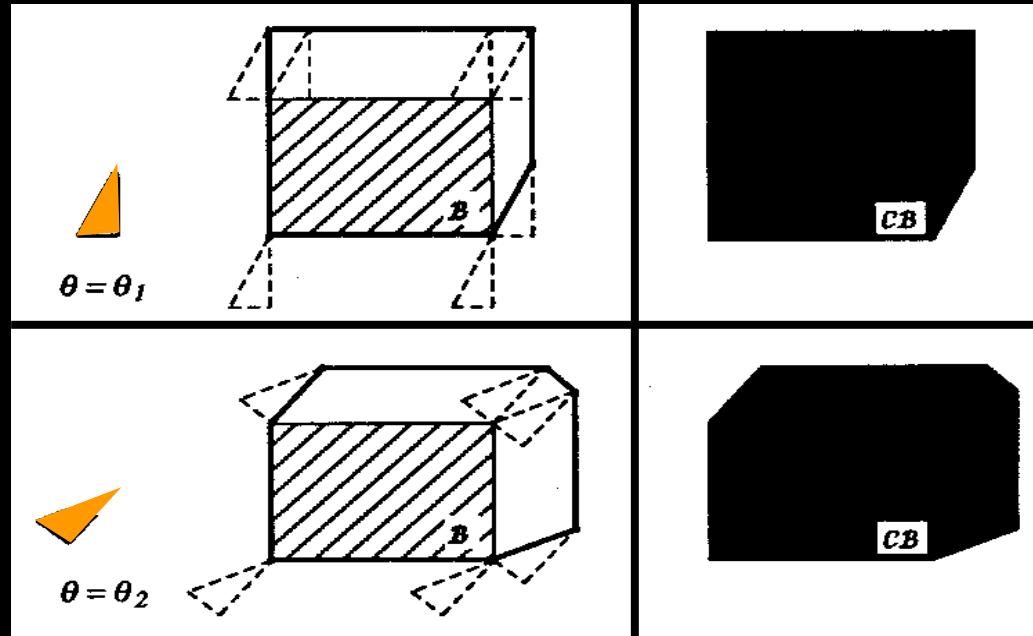
Robot
 \mathcal{R}

C-obstacle
 $O \oplus -\mathcal{R}$

Properties of C_{obs}

- If O and R are *, then C_{obs} is *
 - If convex then C_{obs} is convex
 - If closed then C_{obs} is closed
 - If compact then C_{obs} is compact
 - If algebraic then C_{obs} is algebraic
 - If connected then C_{obs} is connected

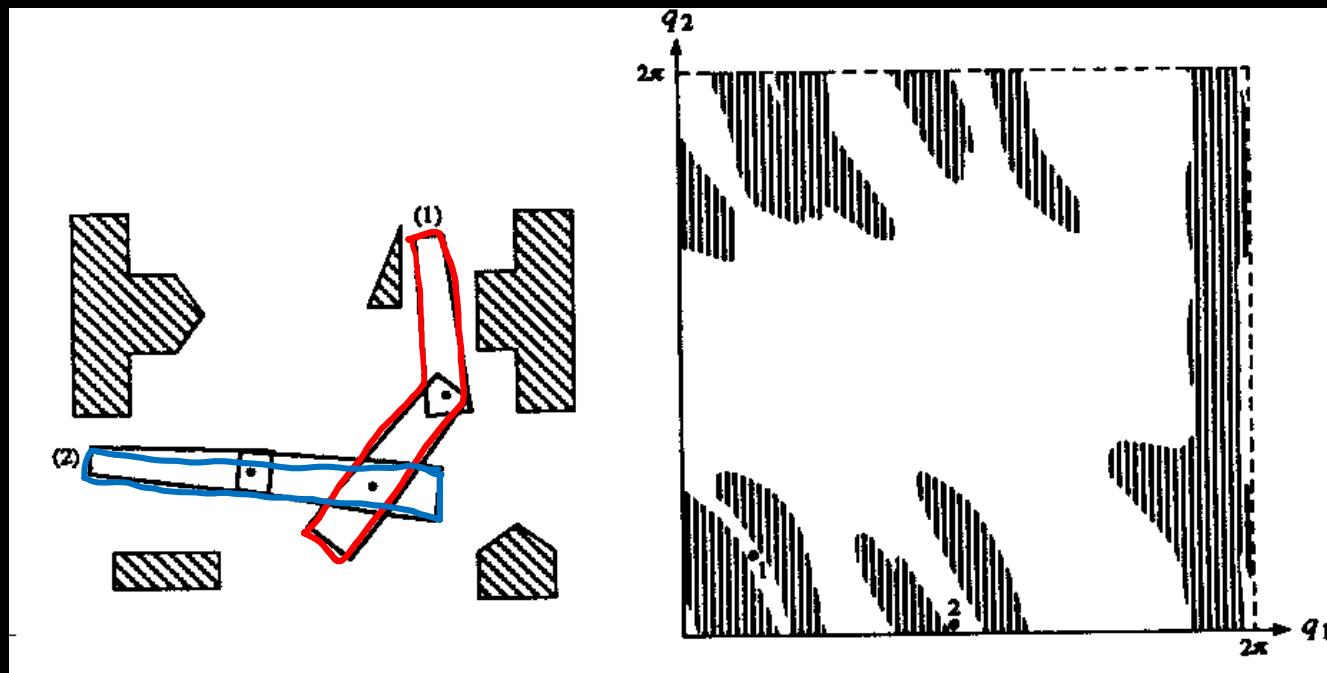
Example: 2D Robot with Rotation



Minkowski Sums

- Can Minkowski Sums be computed in higher dimensions efficiently?

Configuration Space for Articulated Robots



How to compute C_{obs} for articulated bodies?

Break

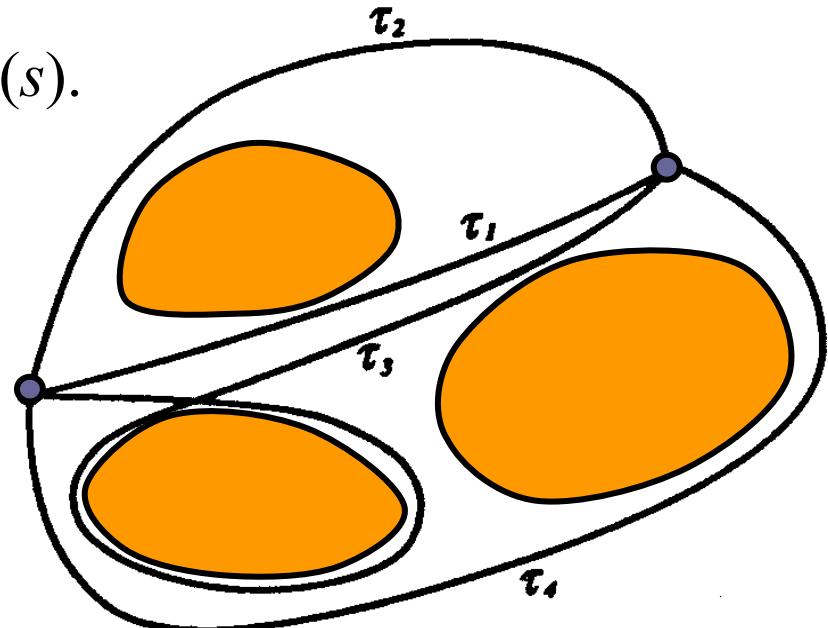
Homotopic paths

- Two paths τ and τ' with the same endpoints are **homotopic** if one can be continuously deformed into the other through the free space F :

$$h : [0,1] \times [0,1] \rightarrow F$$

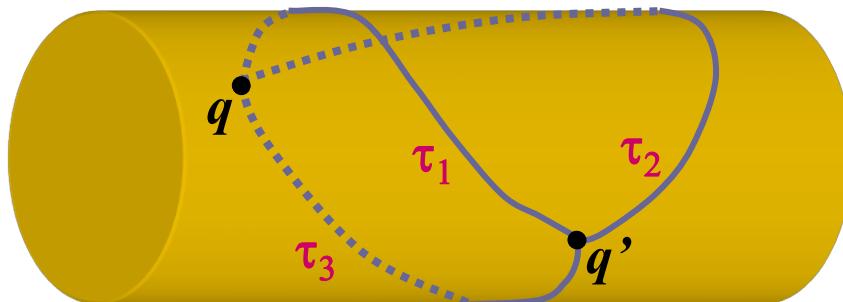
with $h(s,0) = \tau(s)$ and $h(s,1) = \tau'(s)$.

- A homotopic class of paths contains all paths that are homotopic to one another.



Example

- τ_1 and τ_2 are homotopic
- τ_1 and τ_3 are not homotopic
- There are infinity homotopy classes here. Why?



Connectedness of *C*-Space

- C is **connected** if every two configurations can be connected by a path.
- C is **simply-connected** if any two paths connecting the same endpoints are homotopic.
Examples: \mathbb{R}^2 or \mathbb{R}^3
- Otherwise C is multiply-connected.
 - Can you think of an example?

Metrics in configuration space

- A **metric** or **distance** function d in a configuration space C is a function

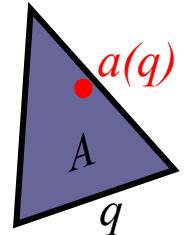
$$d : (q, q') \in C^2 \rightarrow d(q, q') \geq 0$$

such that

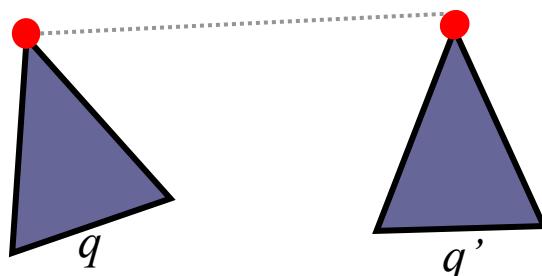
- $d(q, q') = 0$ if and only if $q = q'$,
- $d(q, q') = d(q', q)$,
- $d(q, q') \leq d(q, q'') + d(q'', q')$.

Example

- Consider robot A and a point a on A
- $a(q)$: position of a in the workspace when A is at configuration q
- Example distance metric: d in C is defined by
$$d(q, q') = \max_{a \in A} \|a(q) - a(q')\|$$



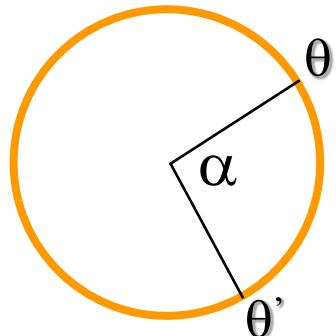
where $\|a - b\|$ denotes the Euclidean distance between points a and b in the workspace.



Examples in $\mathbb{R}^2 \times S^1$

□ Consider $\mathbb{R}^2 \times S^1$

- $q = (x, y, \theta)$, $q' = (x', y', \theta')$ with $\theta, \theta' \in [0, 2\pi)$
- $\alpha = \min \{ |\theta - \theta'|, 2\pi - |\theta - \theta'| \}$

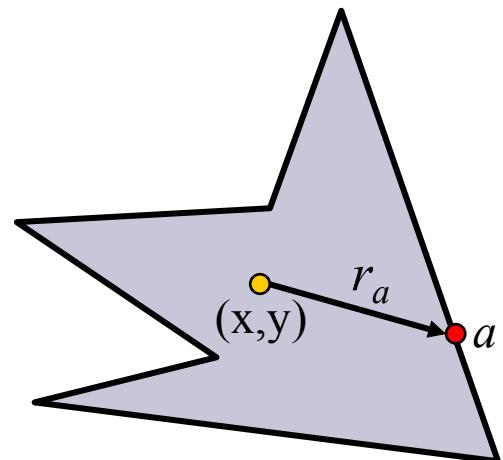


□ $d(q, q') = \max_{a \in A} \|a(q) - a(q')\|$

$$= \max_{a \in A} \sqrt{(x - x')^2 + (y - y')^2 + \alpha r_a}$$

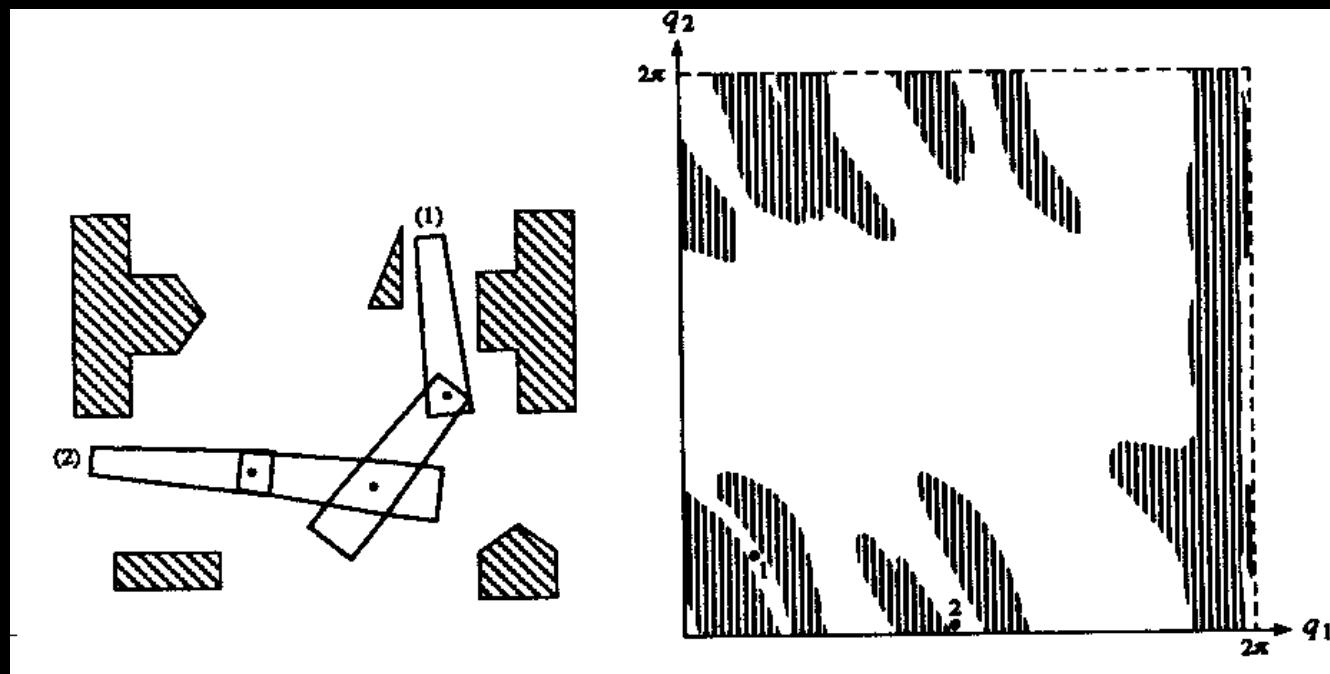
$$= \sqrt{(x - x')^2 + (y - y')^2 + \alpha \max_{a \in A} r_a}$$

$$= \sqrt{(x - x')^2 + (y - y')^2 + \alpha r_{\max}}$$



Distance Metric for Articulated Bodies

- Let's try to think of one



Discussion

- Do we need to have an explicit representation of C-obstacles to do path planning?
- Do we need a specialized distance metric in C-space to do path planning?
 - Can we use Euclidian distance in C-space?

Homework

- Read Chapter 3.2 - 3.3
- HW1 is posted!

Robot policy Learning

policy → steuert das Verhalten des Roboters → Sensordaten und daraus folgende Aktionen

policy

wichtig

Plan

1 Imitation Learning

1.1 Software Framework for Data Generation

1.2 Deep Learning Methods for Imitation Learning

2 Reinforcement Learning

2.1 Q-learning

2.2 Deep RL

Rigid Body Transformations

Rigid Body Transformations

- An understanding of 2D and 3D rigid-body transformations is key to motion planning (and robotics in general)
- There are many representations, none is the “best”
 - Each representation is useful in a different way

Outline

- Homogenous Transforms
- Quaternions
- Euler angles

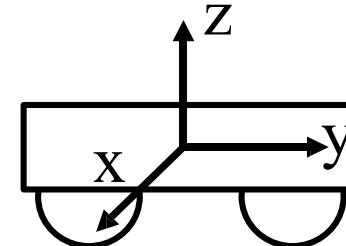
Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary



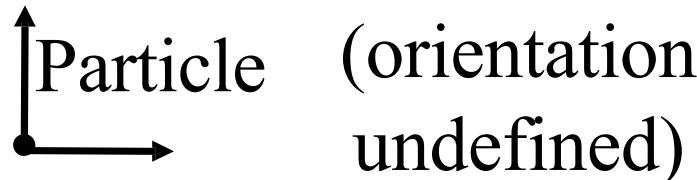
Objects and Embedded Coordinate Frames

- Objects of interest are real:
wheels, sensors,
obstacles.
- Abstract them by sets of axes
fixed to the body.
- These axes:
 - Have a state of motion
 - Can be used to express vectors.
- Call them **coordinate frames**.

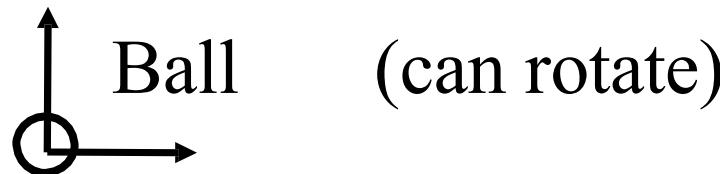


Coordinate Frames

- Points possess position but not orientation:



- Rigid Bodies possess position and orientation:



Relations

- Mechanics is about relations between objects.
- a is “r-related” to b is written:
- Example velocity (v) of robot (r) relative to earth (e):
- Relationship is directional and asymmetric.



$$\begin{matrix} r^b \\ a \end{matrix}$$

$b = \text{base}$

$a = \text{item}$

$$\begin{matrix} e \\ V_r \end{matrix}$$

$e = \text{earth}, w = \text{world}$

$r = \text{robot}$

$$\begin{matrix} r^b \\ a \end{matrix} \neq \begin{matrix} r^a \\ b \end{matrix}$$



Notational Conventions

- Vectors:

$$\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T$$

- Also sometimes as $\underline{\mathbf{p}}$ or as $\overset{\rightharpoonup}{\mathbf{p}}$ to emphasize it is a vector.
- Matrices:

$$\mathbf{T} = \begin{bmatrix} t_{xx} & t_{xy} & t_{xz} \\ t_{yx} & t_{yy} & t_{yz} \\ t_{zx} & t_{zy} & t_{zz} \end{bmatrix}$$

Converting Coordinates

$$p^b = T_a^b p_a$$

- We will see later that T_a^b notation satisfies our conventions where it means the 'T' property of 'object' a w.r.t 'object' b.

Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary



Affine Transformation

- Most general linear transformation

$$\begin{bmatrix} \textcolor{brown}{x}_2 \\ \textcolor{brown}{y}_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} \textcolor{brown}{x}_1 \\ \textcolor{brown}{y}_1 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

- r's and t's are the transform constants
- Can be used to effect translation, rotation, scale, reflections, and shear.
- Preserves linearity but not distance (hence, not areas or angles).



Homogeneous Transformation

- Set $t_1 = t_2 = 0$:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Homogeneous

- r 's are the transform constants
- Can be used to effect rotation, scale, reflections, and shear (not translation).
- Preserves linearity but not distance (hence, not areas or angles).



Orthogonal Transformation

- Looks the same ...
but:

$$\begin{bmatrix} \textcolor{brown}{x}_2 \\ \textcolor{brown}{y}_2 \end{bmatrix} = \begin{bmatrix} \textcolor{brown}{r}_{11} & \textcolor{brown}{r}_{12} \\ \textcolor{brown}{r}_{21} & \textcolor{brown}{r}_{22} \end{bmatrix} \begin{bmatrix} \textcolor{brown}{x}_1 \\ \textcolor{brown}{y}_1 \end{bmatrix} \quad \begin{aligned} \textcolor{brown}{r}_{11}\textcolor{brown}{r}_{12} + \textcolor{brown}{r}_{21}\textcolor{brown}{r}_{22} &= 0 \\ \textcolor{brown}{r}_{11}\textcolor{brown}{r}_{11} + \textcolor{brown}{r}_{21}\textcolor{brown}{r}_{21} &= 1 \\ \textcolor{brown}{r}_{12}\textcolor{brown}{r}_{12} + \textcolor{brown}{r}_{22}\textcolor{brown}{r}_{22} &= 1 \end{aligned}$$

- Can be used to effect rotation, reflections.
- Preserves linearity AND distance (hence, areas and angles).

Rotation Matrix

- Looks the same ...
but:

$$\begin{bmatrix} \textcolor{brown}{x}_2 \\ \textcolor{brown}{y}_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} \textcolor{brown}{x}_1 \\ \textcolor{brown}{y}_1 \end{bmatrix}$$

\mathbf{R}

$$\begin{aligned} r_{11}r_{12} + r_{21}r_{22} &= 0 \\ r_{11}r_{11} + r_{21}r_{21} &= 1 \\ r_{12}r_{12} + r_{22}r_{22} &= 1 \end{aligned}$$

Determinant(\mathbf{R})=1

- Can be used to effect rotation.
- Preserves linearity AND distance (hence, areas and angles).



Definitions

- Heading = angle of path tangent.
- Yaw = rotation about vertical axis
- Pitch = rotation about level sideways axis
- Roll = rotation about “forward” axis.
- Attitude = pitch & roll
- Azimuth = yaw (for a pointing device)
- Elevation = pitch (for a pointing device)



Definitions

- Orientation = attitude & yaw.
- Pose = position & orientation

$$\text{2D: } \begin{bmatrix} x & y & \psi \end{bmatrix}^T \quad \text{3D: } \begin{bmatrix} x & y & z & \theta & \varphi & \psi \end{bmatrix}^T$$

- Posture = Pose plus some configuration
$$\begin{bmatrix} x & y & z & \theta & \varphi & \psi & q \end{bmatrix}^T$$
- Motion = movement of the whole body through space.



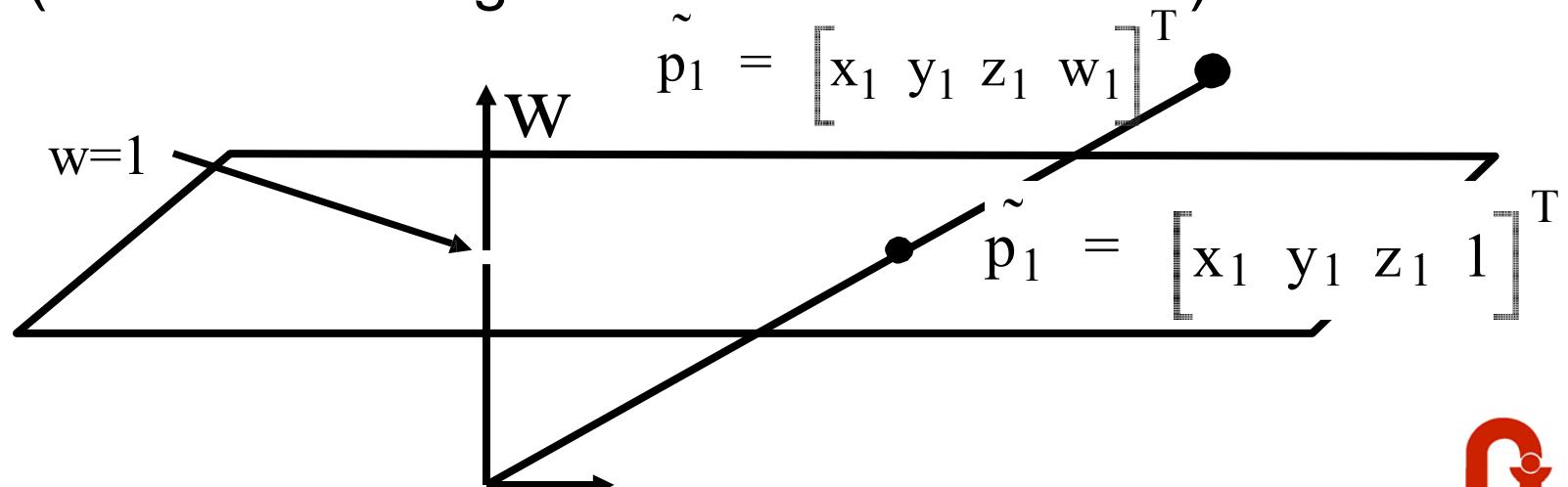
Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary



Homogeneous Coordinates

- Coordinates which are unique up to a scale factor. i.e
 $\underline{x} = 6\underline{x} = -12\underline{x} = 3.14\underline{x}$ = same thing
- The numbers in the vectors are not the same but we interpret them to mean the same thing (in fact. the thing whose scale factor is 1).



Pure Directions

- It's also possible to represent pure directions
 - Pure in the sense they "are everywhere" (i.e. have no position and cannot be moved).
- We use a scale factor of zero to get a pure direction:

$$\mathbf{d}_1 = \begin{bmatrix} \text{x}_1 \\ \text{y}_1 \\ \text{z}_1 \\ 0 \end{bmatrix}$$

- It will shortly be clear why this works.

Why Bother?

- Points in 3D can be rotated, reflected, scaled, and sheared with 3×3 matrices....

$$\mathbf{p}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = T \mathbf{p}_1 = \begin{bmatrix} t_{xx} & t_{xy} & t_{xz} \\ t_{yx} & t_{yy} & t_{yz} \\ t_{zx} & t_{zy} & t_{zz} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} t_{xx}x_1 + t_{xy}y_1 + t_{xz}z_1 \\ t_{yx}x_1 + t_{yy}y_1 + t_{yz}z_1 \\ t_{zx}x_1 + t_{zy}y_1 + t_{zz}z_1 \end{bmatrix}$$

- But not translated.

$$\mathbf{p}_2 = \mathbf{p}_1 + \mathbf{p}_k = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} \neq \text{Trans}(\mathbf{p}_k) \mathbf{p}_1$$

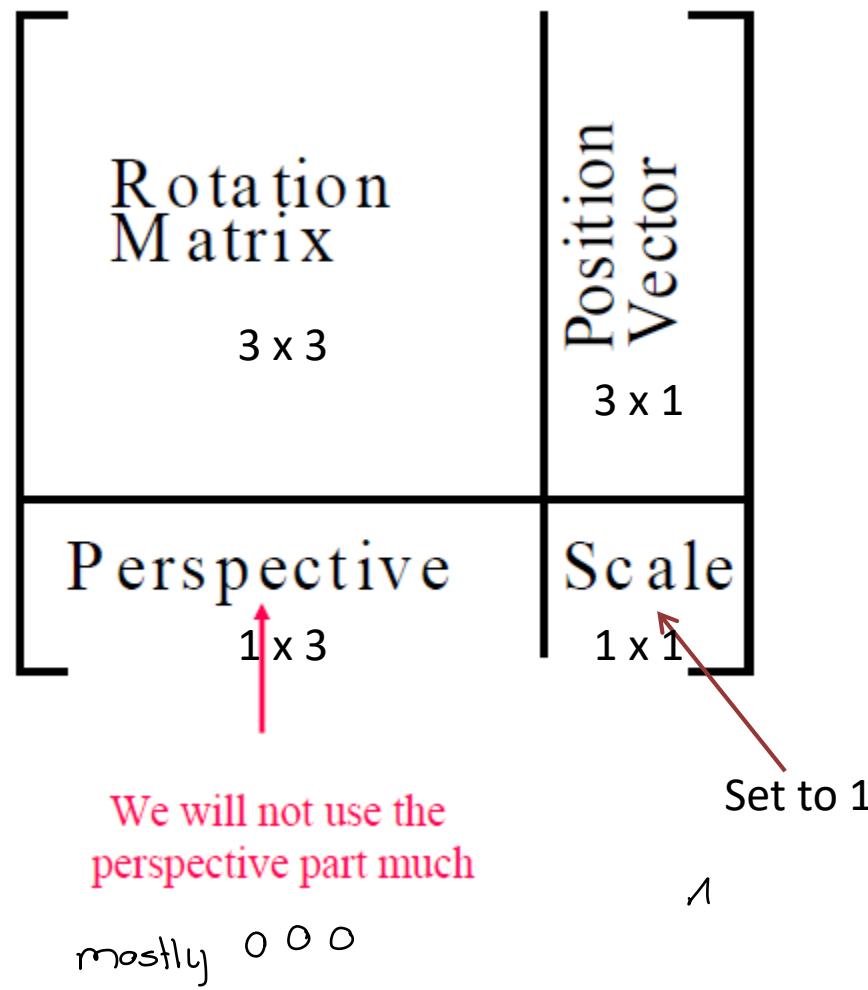
Trick: Move to 4D

$$p_2 = p_1 + p_k = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_k \\ 0 & 1 & 0 & y_k \\ 0 & 0 & 1 & z_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \text{Trans}(p_k)p_1$$

$$\begin{aligned} x_2 &= 1 \times x_1 + x_k \\ y_2 &= 1 \times y_1 + y_k \\ z_2 &= 1 \times z_1 + z_k \end{aligned}$$

- The scale factor in the vector is used to add a scaled amount of the 4th matrix column.

Format of Homogeneous Transforms (HTs)



Inverse of a HT

$$\begin{bmatrix} R & | & p \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & | & -R^T p \\ \hline 0 & 0 & 0 & 1 \end{bmatrix}$$

- Of course standard matrix inverse also works, but this is faster

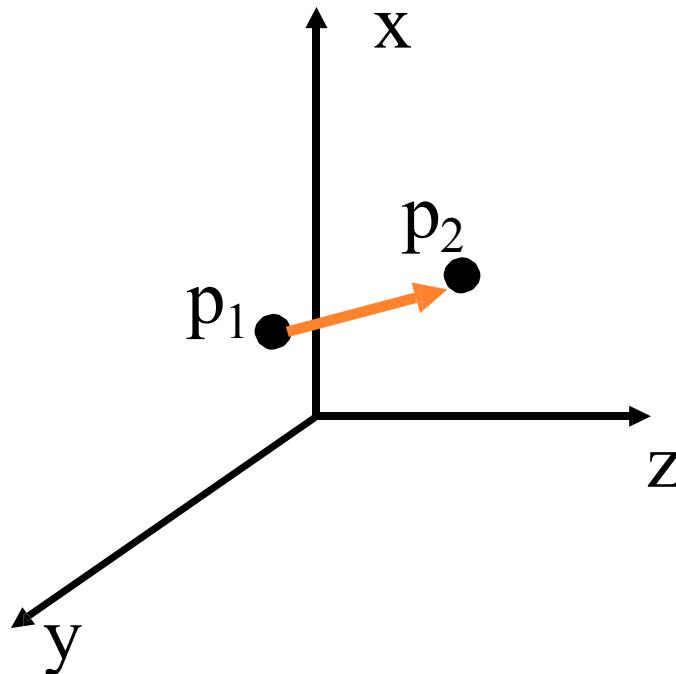
Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary

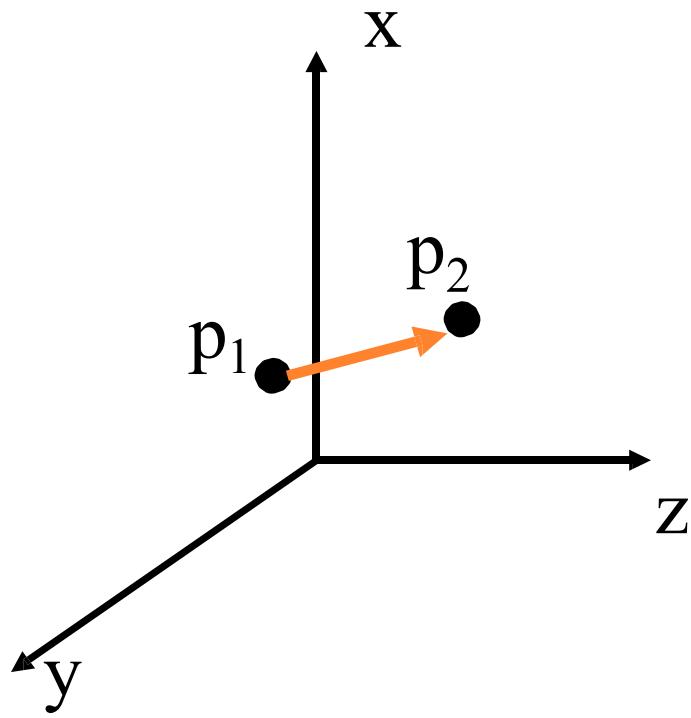


Operators

- Mapping:
 - Point1 -> Point2 (both expressed in same coordinates)



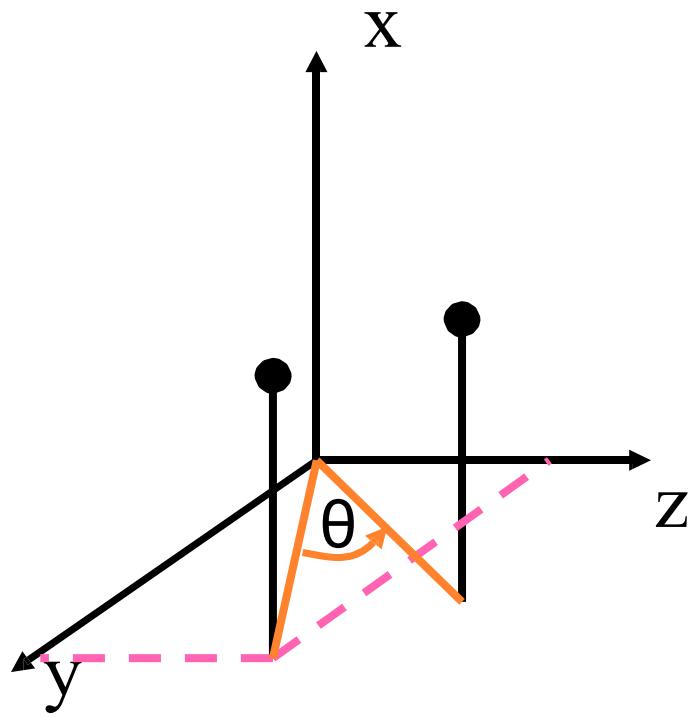
Operators



$$\text{Trans}(u, v, w) = \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Operators

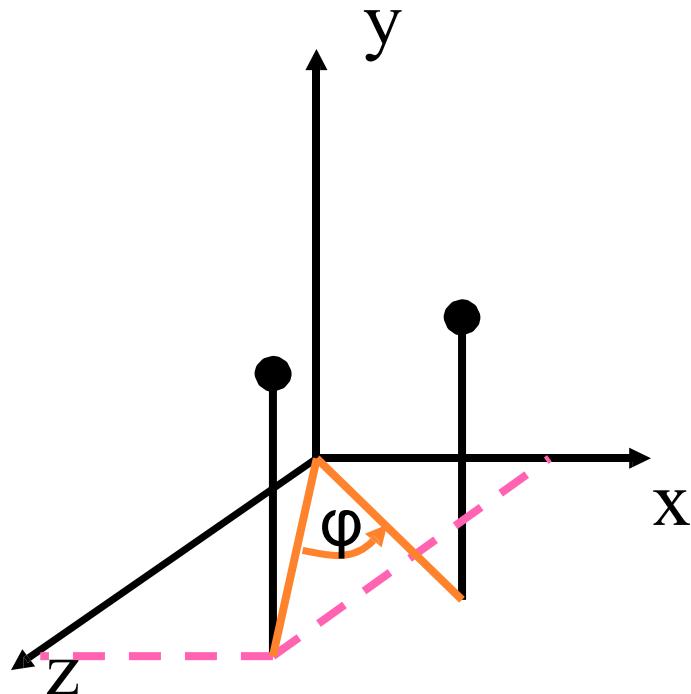
$$\begin{aligned}s\theta &= \sin(\theta) \\ c\theta &= \cos(\theta)\end{aligned}$$



$$\text{Rot}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & 0 \\ 0 & s\theta & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Operators

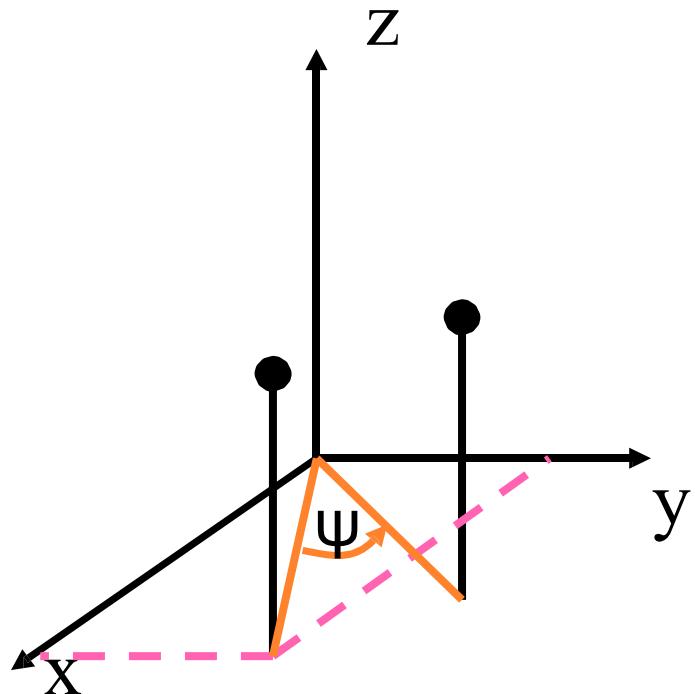
$$\begin{aligned}s\theta &= \sin(\theta) \\ c\theta &= \cos(\theta)\end{aligned}$$



$$\text{Rot}_y(\phi) = \begin{bmatrix} c_\phi & 0 & s_\phi & 0 \\ 0 & 1 & 0 & 0 \\ -s_\phi & 0 & c_\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Operators

$$\begin{aligned}s\theta &= \sin(\theta) \\ c\theta &= \cos(\theta)\end{aligned}$$



$$\text{Rot}_z(\psi) = \begin{bmatrix} c\psi & -s\psi & 0 & 0 \\ s\psi & c\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: Operating on a Point

- A point at the origin is translated along the y axis by 'v' units and then the resulting point is rotated by 90 degrees around the x axis.

$$p = \text{Rot}_x(\pi/2) \text{Trans}(0, v, 0) o$$

$p = [0 \ 0 \ v \ 1]^T$

$$p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ v \\ 1 \end{bmatrix}$$

$$o' = [0 \ v \ 0 \ 1]^T$$

Example: Operating on a Direction

- The y axis unit vector is translated along the y axis by v units and then rotated by 90 degrees around the x axis.

$$\mathbf{j}' = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$$

$$\mathbf{j} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$$

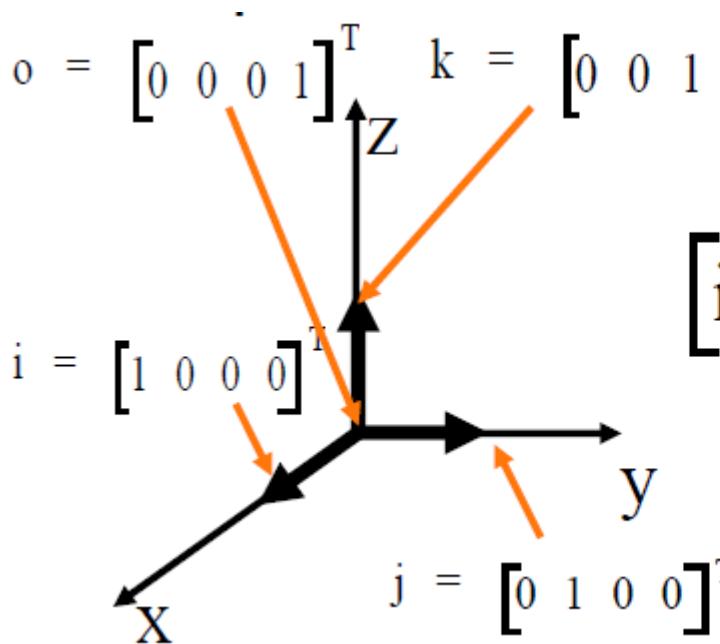
$$\mathbf{j}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The diagram shows the transformation process. The first matrix represents a 90-degree rotation around the x-axis. The second matrix represents a translation along the y-axis by v units. The third matrix represents another 90-degree rotation around the x-axis. The final result is a unit vector along the positive z-axis.

- Having a zero scale factor disables translation.

HTs as Coordinate Frames

- The columns of the identity HT can be considered to represent 3 directions and a point – the coordinate frame itself.



$$\begin{aligned} \mathbf{o} &= [0 \ 0 \ 0 \ 1]^T & \mathbf{k} &= [0 \ 0 \ 1 \ 0]^T \\ \mathbf{i} &= [1 \ 0 \ 0 \ 0]^T & [\mathbf{i} \ \mathbf{j} \ \mathbf{k} \ \mathbf{o}] &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{I} \\ \mathbf{j} &= [0 \ 1 \ 0 \ 0]^T \end{aligned}$$

Example: Operating on a Frame

- Each resulting column of this result is the transformation of the corresponding column in the original identity matrix

The diagram illustrates a coordinate frame transformation. On the left, a 3D Cartesian coordinate system is shown with axes labeled x, y, and z. A second coordinate frame, labeled i' , j' , k' , is attached to the y-axis. An orange arrow points from the origin of the world frame to the origin of the new frame, indicating a translation. The new frame's axes are labeled i' , j' , k' . To the right of the diagram, two matrices are shown. The first matrix, I' , is the result of applying a rotation around the x-axis by $\pi/2$ followed by a translation by vector v . The second matrix, I' , is expanded to show its components: a rotation matrix for Rot_x , a translation matrix for Trans , and a third matrix labeled "in der heutigen Basiskoordinatensystem".

$$\begin{bmatrix} 0 & -1 & 0 & 0 \end{bmatrix}^T \xrightarrow{\text{I}' = \text{Rot}_x(\pi/2) \text{Trans}(0, v, 0) I} \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$$

$$I' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotx Trans in der heutigen Basiskoordinatensystem

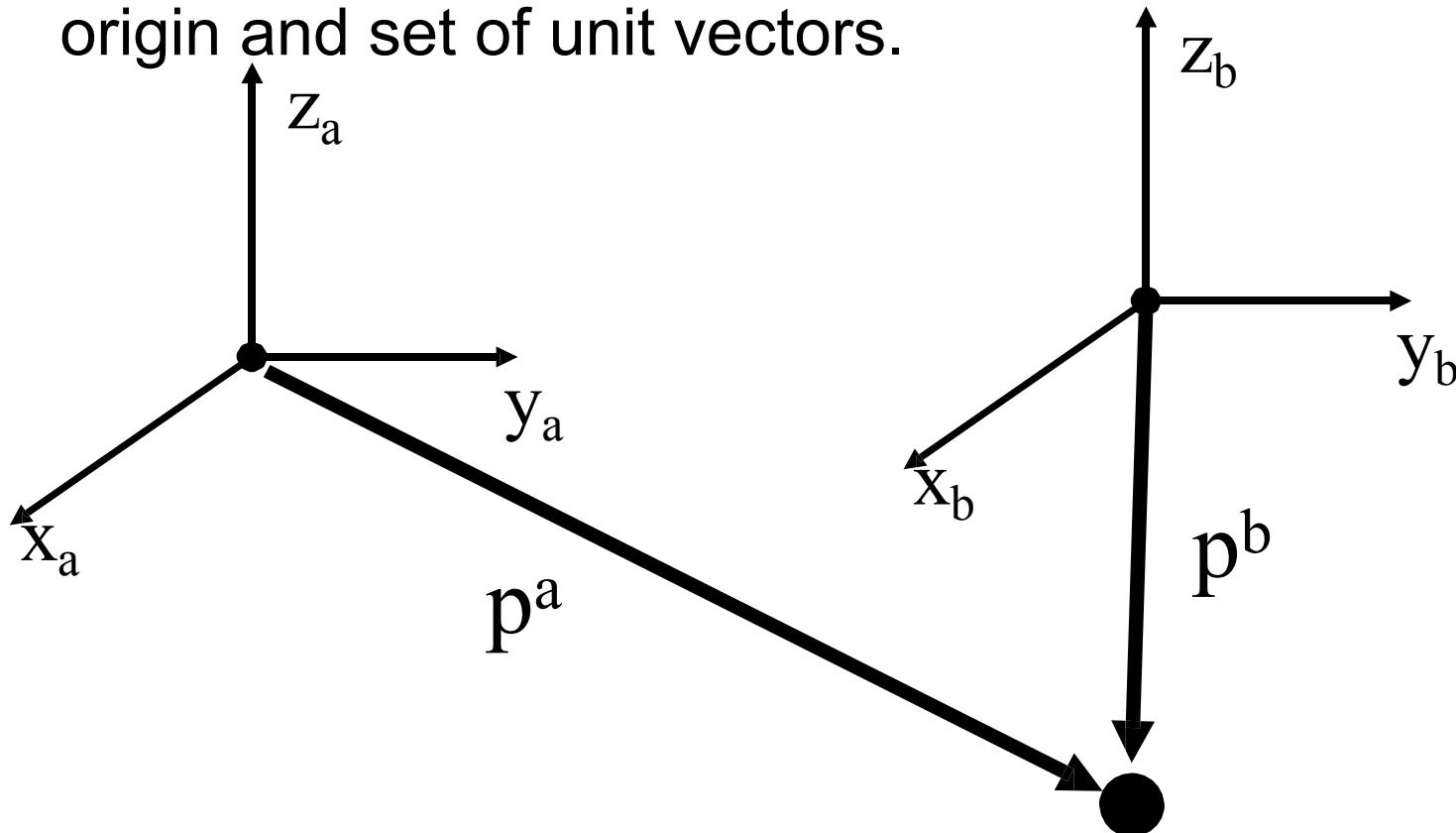
Huh?

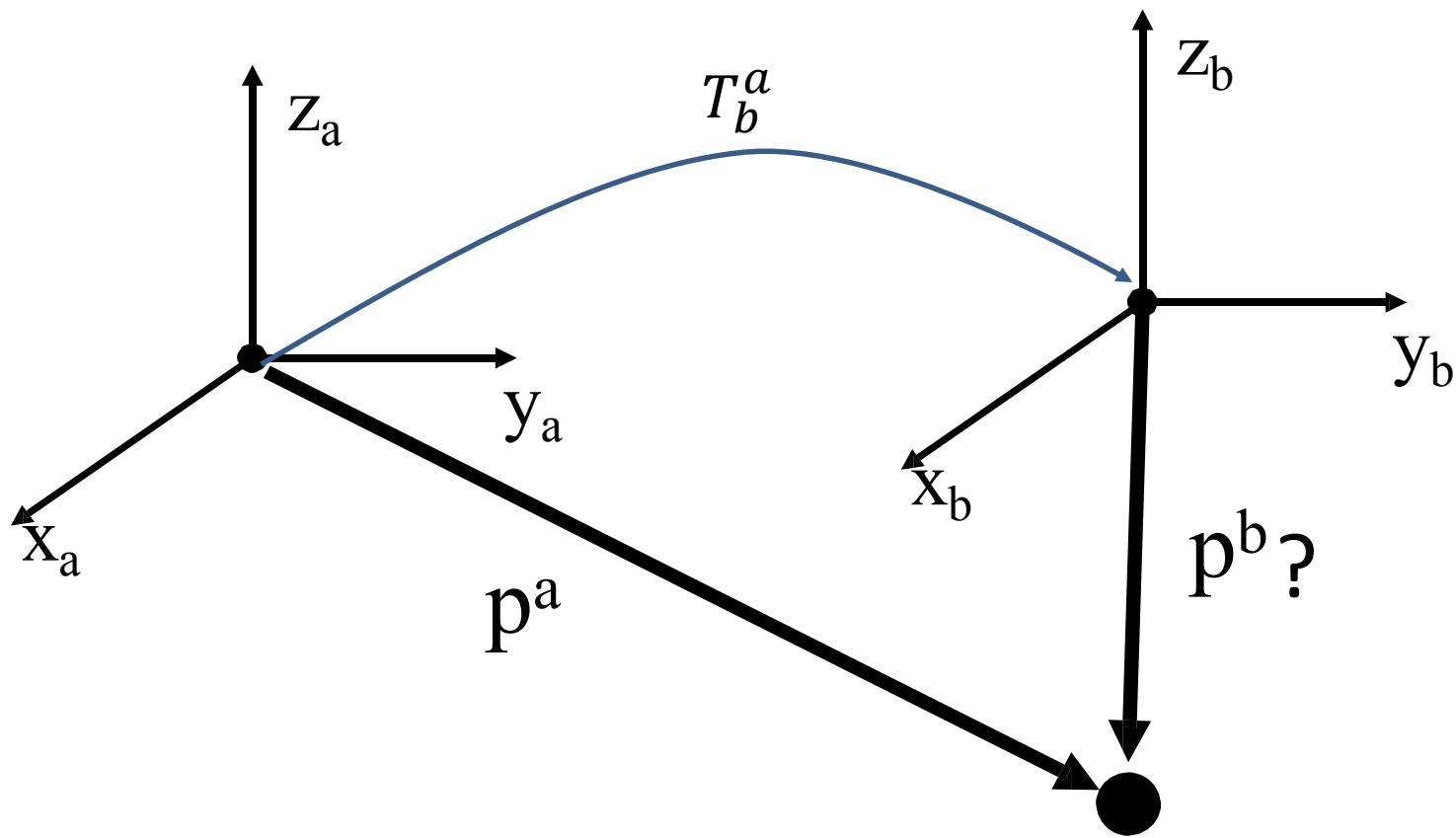
- Columns of an input matrix are treated independently in multiplication.
- Every orthonormal matrix can be viewed as one set of axes located with respect to another set.
 - The “locations” can be read right from the matrix – they’re just the columns.
- We can use this idea to track the position and orientation of rigid bodies....
 - Imagine embedding frames inside them somewhere and track their motions.



Converting Coordinates

- Converting coordinates is about expressing the same physical point with respect to a new origin and set of unit vectors.





- Given T_b^a and p^a , how to compute p^b ?

$$(T_b^a)^{-1} \cdot p^a = p^b$$

Similarity Transforms

- Suppose you have a transform A^0 defined relative to frame 0, and you want to know what it is in frame 1. Assume you know T_1^0 .

$$B = \left((T_1^0)^{-1} \left(A^0 (T_1^0 \cdot v_1) \right) \right)$$

in 1 Sys Transf. in 0 Sys

- B is transform A^0 represented in frame 1

$$T_n^o v_n = v_o$$

$$T_o^1 \cdot A_o v_o = A_o T_n^o v_n$$

$$\boxed{(T_n^o)^{-1} \cdot A_o T_1^0} v_n$$

Sequencing Transforms

- Any sequence of transforms can be represented by a single transform (Euler's rotation theorem)
- How you sequence transforms depends on if you are transforming w.r.t. the *fixed* or the *current* axes

?

- Transforming w.r.t. **current** axes: multiply *on the right*

Beispiel Roboter-Endeffektor

$$T_n^0 = T_1^0 T_2^1 \dots T_n^{n-1}$$

?

- Transforming w.r.t. **fixed** axes of frame 0: multiply *on the left*

$$T_2^0 = T_1^0 [(T_1^0)^{-1} A^0 T_1^0] = A^0 T_1^0$$

kürzen sich

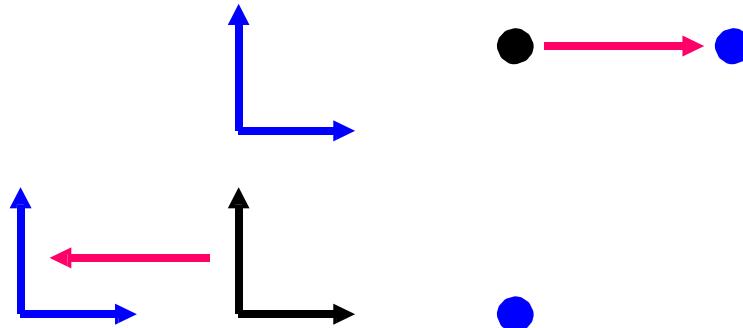
Similarity transform

Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary

Summary

- Everything is relative. There is no way to distinguish moving a point “forward” from moving the coordinate system “backward”.



- In both cases, the resulting (blue) point has the same relationship to the blue frame.

Summary

- Homogeneous Transforms are:
 - Operators
 - Frames
- They can be both the things that operate on other things and the things operated upon.

Outline

- Homogenous Transforms
- Quaternions
- Euler angles

Break

Quaternions



Core Properties

- Quaternions:
 - are generalizations of complex numbers which do not commute (complex numbers do).
 - Use 3 complex numbers:
- $i^2 = j^2 = k^2 = ijk = -1$
- can represent every rotation that a HT can represent.

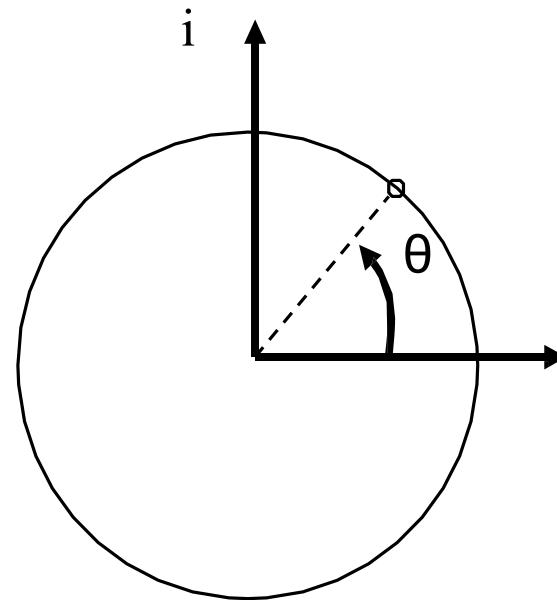
Why Use Em?

- Only way to solve some problems
 - like the problem of generating regularly spaced 3D angles.
- Simplest way to solve some problems.
 - Some problems in registration can be solved in closed form.
- Fastest way to solve some problems.
 - “Quaternion loop” in an inertial navigation system updates vehicle attitude 1000 times a second.



Intuition from Complex Numbers

- Use a second “imaginary” dimension
- Permits manipulation of rotations like a vector



Notations

- 4-tuples
- Hypercomplex numbers
- Sum of real and imaginary parts
- Ordered doublet
- Exponential

$$(q_0, q_1, q_2, q_3) \\ = (w, x, y, z)$$

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$\tilde{q} = q + \hat{\bar{q}}$$

$$(q, \hat{\bar{q}})$$

$$q = e^{\frac{1}{2}\theta w}$$

Manipulate
like
Polynomials

\tilde{q} Yucheng q1

I will use
these two



My Preference

- Mostly use the scalar-vector sum form:

$$\tilde{q} = q + \vec{q}$$

~ means quaternion
→ means 3D normal vector
means scalar

- Occasionally write it out to get hypercomplex form:

$$q = q_0 + q_1 i + q_2 j + q_3 k$$



Multiplication

- Quaternions are elements of a vector space endowed with multiplication.
- The expression:

$$\tilde{p}\tilde{q} = (p_0 + p_1 i + p_2 j + p_3 k)(q_0 + q_1 i + q_2 j + q_3 k)$$

- Gives the sum of all these elements:

	q_0	$q_1 i$	$q_2 j$	$q_3 k$
p_0	$p_0 q_0$	$p_0 q_1 i$	$p_0 q_2 j$	$p_0 q_3 k$
$p_1 i$	$p_1 q_0 i$	$p_1 q_1 i^2$	$p_1 q_2 j i$	$p_1 q_3 k i$
$p_2 j$	$p_2 q_0 j$	$p_2 q_1 j i$	$p_2 q_2 j^2$	$p_2 q_3 j k$
$p_3 k$	$p_3 q_0 k$	$p_3 q_1 k i$	$p_3 q_2 k j$	$p_3 q_3 k^2$

- So, we need to define what i^*i etc mean...



Multiplication Rule

- Two goals:
 - 1) Manipulate like polynomials
 - 2) Product of two quaternions is a quaternion.
- To get things to work as Hamilton intended we need to have:

Diagonals work like complex numbers. Off diagonals work like vector cross product.

- Or, more compactly:

	i	j	k
i	-1	k	-j
j	-k	-1	i
k	j	-i	-1

$$i^2 = j^2 = k^2 = ijk = -1$$

$$\underbrace{k \cdot k}_{k^2} = -1$$

$$\underbrace{i \cdot j}_{k^2} = -1$$

Product

- In hypercomplex (polynomial) form:

$$\begin{aligned} \mathbf{pq} &= (p_0 + p_1 i + p_2 j + p_3 k)(q_0 + q_1 i + q_2 j + q_3 \\ &\quad k) \\ \tilde{\mathbf{pq}} &= (p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3) + (\dots)i + \\ &\quad \dots \end{aligned}$$

- In vector form: $\tilde{\mathbf{pq}} = (\mathbf{p} + \hat{\mathbf{p}})(\mathbf{q} + \hat{\mathbf{q}})$

$$\tilde{\mathbf{pq}} = \mathbf{pq} + \mathbf{p}\hat{\mathbf{q}} + \mathbf{q}\hat{\mathbf{p}} + \hat{\mathbf{p}}\hat{\mathbf{q}}$$

- The last term can be written in terms of familiar vector products.

$$\hat{\mathbf{p}}\hat{\mathbf{q}} = \hat{\mathbf{p}} \times \hat{\mathbf{q}} - \hat{\mathbf{p}} \cdot \hat{\mathbf{q}}$$

- Convenient to summarize like so:

$$\tilde{\mathbf{pq}} = \mathbf{pq} - \hat{\mathbf{p}} \cdot \hat{\mathbf{q}} + \mathbf{p}\hat{\mathbf{q}} + \mathbf{q}\hat{\mathbf{p}} + \hat{\mathbf{p}} \times \hat{\mathbf{q}}$$

Not the same thing



Non-Commutativity

- The vector cross product does not commute.
Therefore:

$$\tilde{\vec{p}}\tilde{\vec{q}} \neq \tilde{\vec{q}}\tilde{\vec{p}}$$

What is the
source of this
property?



Addition

- Works just like vectors, polynomials, and complex numbers....

$$\tilde{p} + \tilde{q} = (p_0 + q_0) + (p_1 + q_1)i + (p_2 + q_2)j + (p_3 + q_3)k$$



Distributivity

- Works just like vectors, polynomials, and complex numbers....

$$(\tilde{p} + \tilde{q})\tilde{r} = \tilde{p}\tilde{r} + \tilde{q}\tilde{r}$$

$$\tilde{p}(\tilde{q} + \tilde{r}) = \tilde{p}\tilde{q} + \tilde{p}\tilde{r}$$

Dot Product and Norm

- Works just like vectors, polynomials, and complex numbers....

$$\tilde{p} \bullet \tilde{q} = pq + \tilde{\vec{p}} \bullet \tilde{\vec{q}}$$

Not the same thing

- Can now define a length (norm):

$$|\tilde{q}| = \sqrt{\tilde{q} \bullet \tilde{q}}$$

- Unit quaternions have a norm of unity.



Conjugate

- Works just like complex numbers....

$$\tilde{q}^* = q - \overline{\tilde{q}}$$

- Product with conjugate equals dot product:

$$\tilde{q}\tilde{q}^* = (qq + \overline{\tilde{q}} \cdot \tilde{q}) = \tilde{q} \bullet \tilde{q}$$

$= q_o^2 - q_n^2 i^2 - q_{l_2}^2 j^2 - q_{l_3}^2 k^2$
 $= q_o^2 - q_n^2 - q_{l_2}^2 - q_{l_3}^2$
 $\approx q \cdot q + \overline{q} \cdot \overline{q}$
 $= \tilde{q} \cdot \tilde{q}$

- Another way to get the norm is then:

$$|\tilde{q}| = \sqrt{\tilde{q}\tilde{q}^*}$$



Quaternion Inverse

- The Big Kahuna. Since we have:

$$\tilde{q}\tilde{q}^* / |\tilde{q}|^2 = 1$$

- By definition of inverse:

$$\tilde{q}^{-1} = \tilde{q}^* / |\tilde{q}|^2$$



Rotations

- The unit quaternion: $\tilde{q} = \cos \frac{\theta}{2} + \hat{w} \sin \frac{\theta}{2}$
For 0 rotation = $1+0i+0j+0k$
- Represents the operator which rotates by the angle θ around the axis whose unit vector is

$$\theta = 2 \arctan 2(\|\tilde{q}\|, q)$$

$$\hat{w} = \tilde{q} / \|\tilde{q}\|$$



Rotation

- Note that \tilde{q} and $-\tilde{q}$ perform the same rotation! Example:

$$\theta = \frac{\pi}{4}$$

$$\hat{w} = [1, 0, 0]$$

$$\tilde{q} = \cos\left(\frac{\pi}{4}/2\right) + [1, 0, 0]\sin\left(\frac{\pi}{4}/2\right) = [0.9239, \quad 0.3827, \quad 0, \quad 0]$$

Now let's get axis and angle from $-\tilde{q}$:

$$\theta = 2\text{atan2}(|[-0.3827, 0, 0]|, -0.9239) = \frac{7\pi}{4} = \frac{7\pi}{4} - 2\pi = -\frac{\pi}{4}$$

$$\hat{w} = \frac{[-0.3827, 0, 0]}{|[-0.3827, 0, 0]|} = [-1, 0, 0]$$

$\theta = \frac{\pi}{4}$ and $\hat{w} = [1, 0, 0]$ perform the same rotation as $\theta = -\frac{\pi}{4}$ and $\hat{w} = [-1, 0, 0]!$

Vectors as Quaternions

- Quaternionize:

$$\tilde{\mathbf{x}} = \mathbf{0} + \overrightarrow{\mathbf{x}}$$



Any 3D vector

$$= \mathbf{0} + (x_i, y_j, z_k)$$

$$= (0, x, y, z)$$

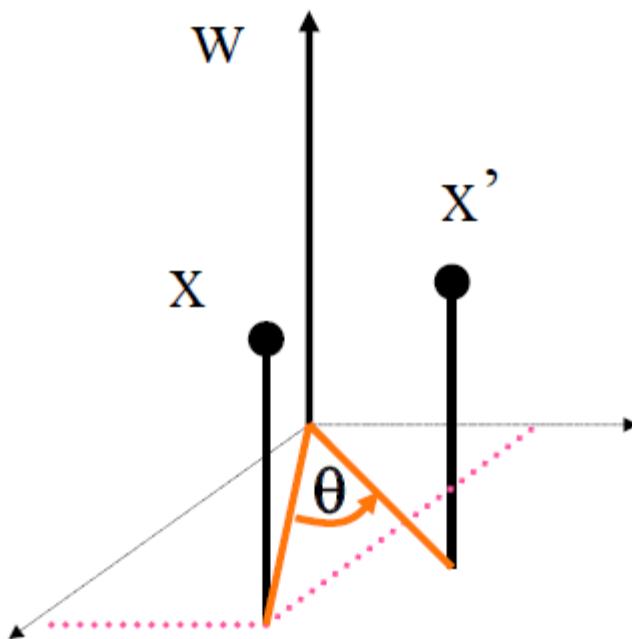
$$q_0 q_1 q_2 q_3$$



Rotating a Vector (Point)

- Use the quaternion sandwich:

$$\tilde{x}' = \tilde{q} \tilde{x} \tilde{q}^*$$



Composite Rotations

- Use the composite quaternion sandwich:

$$\tilde{x}' = \tilde{q}\tilde{x}\tilde{q}^*$$

$$\tilde{x}'' = \tilde{p}\tilde{x}\tilde{p}^* = (\tilde{p}\tilde{q})\tilde{x}(\tilde{q}^*\tilde{p}^*)$$



Quaternion to Rot() Matrix

- For the quaternion:

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

- The equivalent Rot() matrix is:

$$R = \begin{bmatrix} 2[q_0^2 + q_1^2] - 1 & 2[q_1 q_2 - q_0 q_3] & 2[q_1 q_3 + q_0 q_2] \\ 2[q_1 q_2 + q_0 q_3] & 2[q_0^2 + q_2^2] - 1 & 2[q_2 q_3 - q_0 q_1] \\ 2[q_1 q_3 - q_0 q_2] & 2[q_0 q_1 + q_2 q_3] & 2[q_0^2 + q_3^2] - 1 \end{bmatrix}$$

können transformiert
 werden
 (Python etc. Funktion)



Rot() Matrix to Quaternion

- For the Rot() matrix:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

- The equivalent quaternion is determined from:

$$r_{11} + r_{22} + r_{33} = 4q_0^2 - 1$$

$$r_{11} - r_{22} - r_{33} = 4q_1^2 - 1$$

$$-r_{11} + r_{22} - r_{33} = 4q_2^2 - 1$$

$$-r_{11} - r_{22} + r_{33} = 4q_3^2 - 1$$



Summary

- Quaternions are hypercomplex numbers with an i,j, and k that act like the i in complex numbers.
- Notation is half the battle.
- Provide elegant and efficient ways to model 3D transformations of points (and hence 3D coordinate system conversions).
- All rotations are represented as unit quaternions

Goodfellow ?

Quaternion displacement
Dual quaternion
 $q = q_1 + \hat{q}_2$ Translation

3RAB



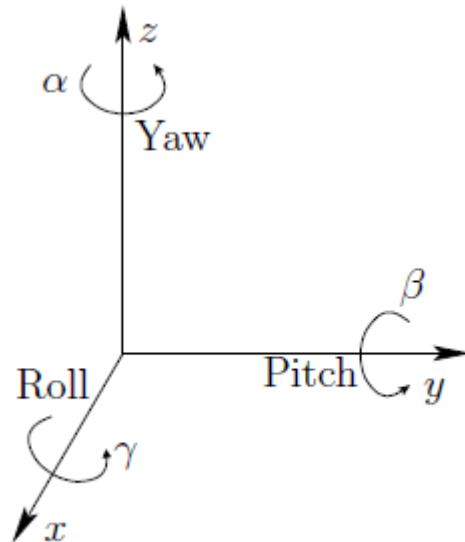
Euler Angles

- Can define rotation relative to the axes of a frame

$$R_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\beta) = \begin{pmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{pmatrix}$$

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{pmatrix}$$



Rotating about more than one axis

- Great for rotating about a single axis
- What if we want to rotate about multiple axes?
 - Need to adopt a *convention*: for example roll-pitch-yaw

$$R(\alpha, \beta, \gamma) = R_z(\alpha) R_y(\beta) R_x(\gamma) =$$
$$\begin{pmatrix} \cos \alpha \cos \beta & \cos \alpha \sin \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \cos \gamma + \sin \alpha \sin \gamma \\ \sin \alpha \cos \beta & \sin \alpha \sin \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \cos \gamma - \cos \alpha \sin \gamma \\ -\sin \beta & \cos \beta \sin \gamma & \cos \beta \cos \gamma \end{pmatrix}$$

- There are 12 variants of ordering and axis selection for Euler angles

Problems with Euler Angles

- Single axis: no problem!
- Two or more axes
 - Results can be counter-intuitive
 - “Small” rotations are better than “large” ones
- Singularities
 - Many Euler Angles map to one rotation (called Gimbal Lock)
 - Where singularities are depends on the convention
 - Small rotations near singularities can have big effects

Singularity Example

- Let's say this is our convention (ZXZ convention):

$$R = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta \\ 0 & \sin \beta & \cos \beta \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Let's set $\beta = 0$

$$R = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Multiplying through, we get:

$$R = \begin{bmatrix} \cos \alpha \cos \gamma - \sin \alpha \sin \gamma & -\cos \alpha \sin \gamma - \sin \alpha \cos \gamma & 0 \\ \sin \alpha \cos \gamma + \cos \alpha \sin \gamma & -\sin \alpha \sin \gamma + \cos \alpha \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Simplify:

$$R = \begin{bmatrix} \cos(\alpha + \gamma) & -\sin(\alpha + \gamma) & 0 \\ \sin(\alpha + \gamma) & \cos(\alpha + \gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

*α and γ do the same thing!
We have lost a degree
of freedom!*

What to use?

- Euler angles are simplest but have singularity problems
 - Besides you usually convert to rotation matrices to actually use them
- Quaternions are beautiful math but can be annoying to manipulate
 - But, quaternions are perfect for some problems. For example: sampling 3D rotations
- My recommendation: Use Homogenous Transforms whenever possible.
 - Easy to compose and manipulate (just linear algebra)
 - Rotation and translation in one consistent package
 - Not super-simple, but can “read” the coordinate axes by looking at columns
- Many software frameworks provide methods to convert between these representations (e.g. ROS)
 - *only when the problem is really easy*

Homework

- Read Chapter 2.0 - 2.3

An dieser Stelle befinden sich chronologischer
Weise die Inhalte aus dem Jupyter Notebook

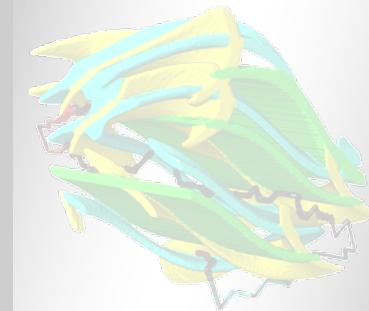
IP-5-0-PRM_Basics

PRM – Overview: Sampling

Probabilistic Roadmap



All PRM-Methods have common issues:
Which strategies for optimal sampling of C-Space?



References

Ideas taken from presentation & papers:

- ▶ „Sampling and Connection Strategies for PRM Planners“: presentation (Jean-Claude Latombe, Computer Science Department, Stanford University)
- ▶ “Sampling Strategies for PRMs”: slides of T.V.N. Sri Ram
- ▶ “A General Framework for Sampling on the Medial Axis of the Free Space”: presentation (Jyh-Ming Lien, Shawna Thomas, Nancy Amato)
- ▶ “The Gaußion Sampling Strategy for Probabilistic Roadmap Planners”: paper (Boor, Overmars, Stappen)
- ▶ “Probabilistic Motion Planning”: presentation (Shai Hirsch)
- ▶ “Rapidly-Exploring Random Trees”: presentation (Steven LaValle)

Sampling strategies

Strategies depending on intention of PRMs

► Multi-Query-PRM:

- ▶ Build up a good roadmap covering C-Space in an efficient way
- ▶ Allow a little bit of time for roadmap building
- ▶ Typical strategies:
 - ▶ Multi-Stage-Sampling
 - ▶ Obstacle-Based-Sampling
 - ▶ Narrow-passage-Sampling

► Single-Query-PRM:

- ▶ Test as few nodes and edges as possible
- ▶ Cover C-Space between start and goal as fast and as good as possible
- ▶ Typical strategies:
 - ▶ Diffusion
 - ▶ Adaptiv-Step
 - ▶ Biased Sampling
 - ▶ Control-Based Sampling

Optimality is not as important in this method

Multi-Query-PRM

- ▶ Typical strategies:
 - ▶ Multi-Stage-Sampling
 - ▶ Obstacle-Based-Sampling
 - ▶ Narrow-passage-Sampling

Multi-Query-PRM: Multi-Stage-Sampling

Idea

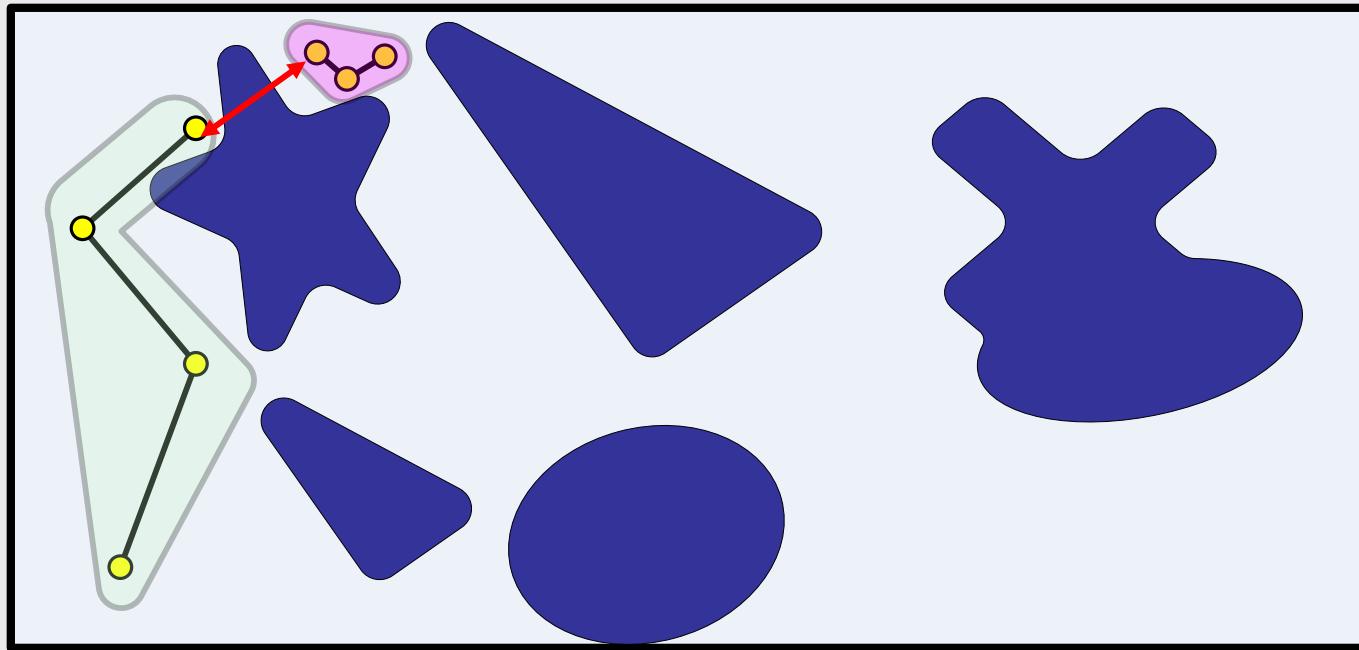
- ▶ Identify regions of the free space whose connectivity is more difficult to capture → force sampling there (= **second stage**).

As already proposed e.g.:

- ▶ Number of neighbors of a node
 - ▶ If the number is low, then the obstacles are probably a large part of the area around → difficult region
- ▶ Distance nearest connected component not containing node c .
 - ▶ If d_c is small, then c is in a region where two components failed to connect. → difficult region
- ▶ Could also use information collected by the local planner.
 - ▶ If the planner often fails to connect a node to others, then this indicates the node is in a difficult area.
- ▶

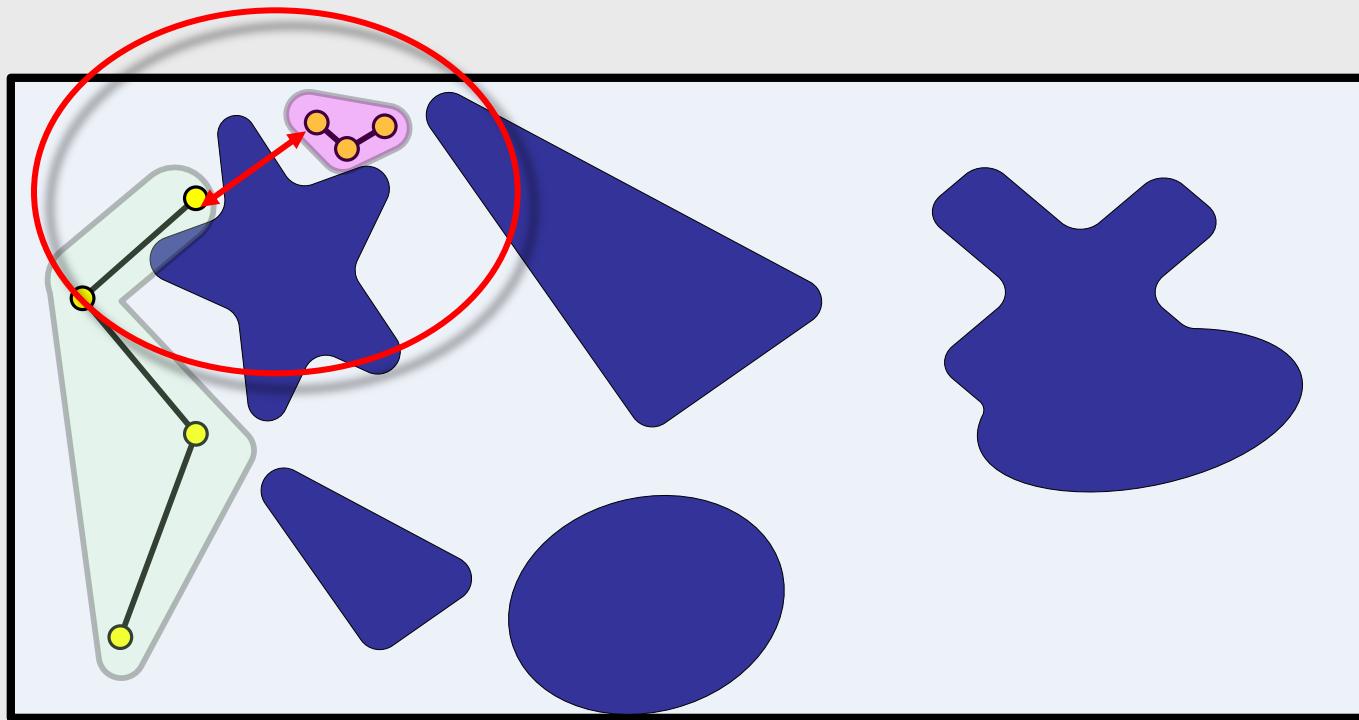
Multi-Stage-Sampling: identify difficult regions

- ▶ Distance nearest connected component not containing node **c**



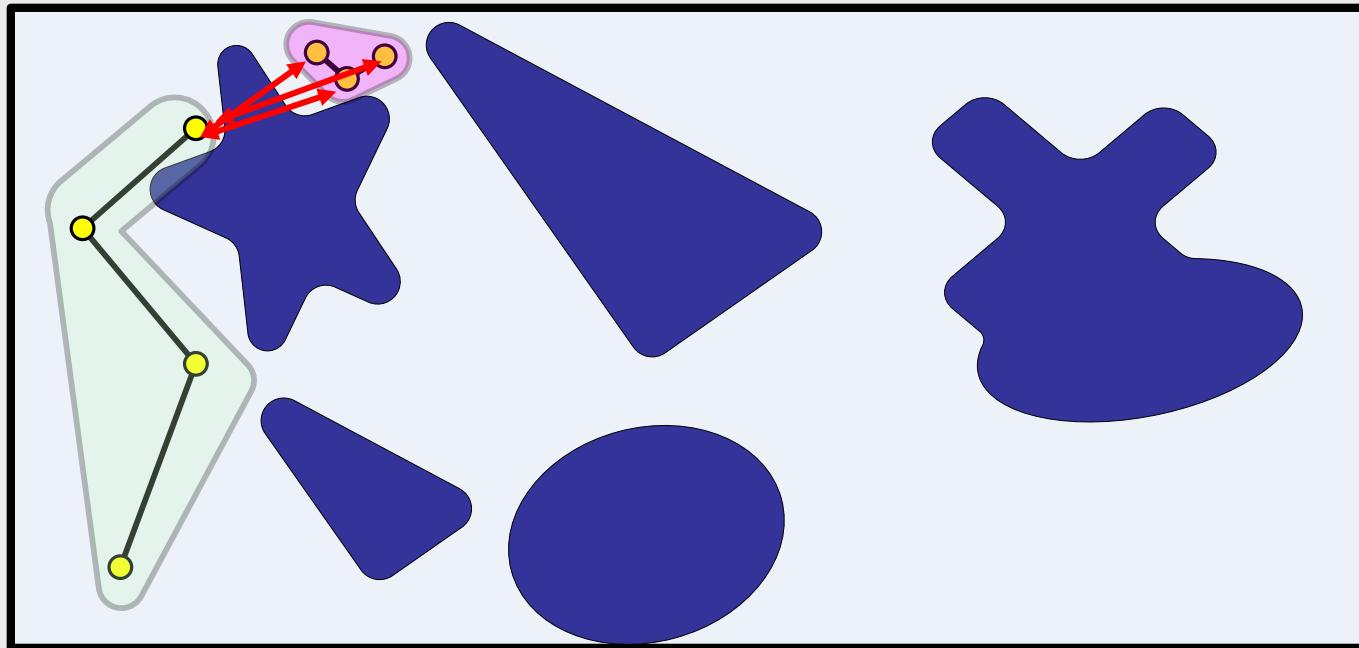
Multi-Stage-Sampling: identify difficult regions

- ▶ Distance nearest connected component not containing node **c**



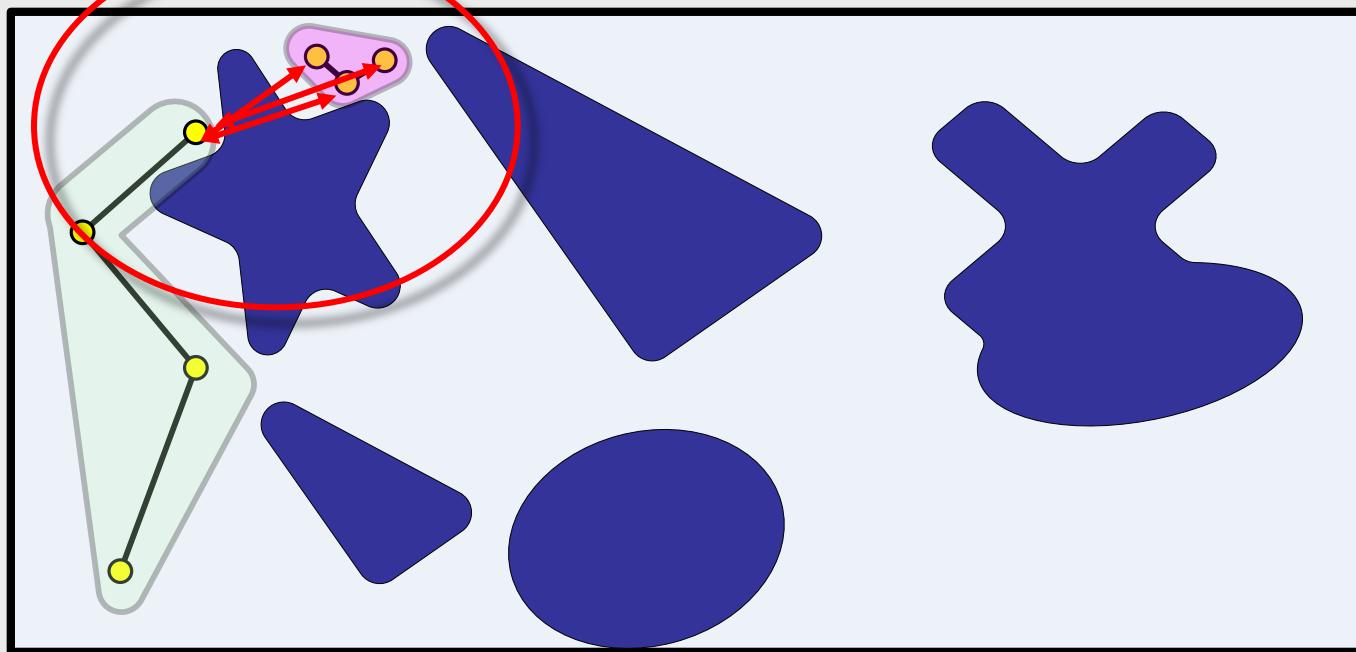
Multi-Stage-Sampling: identify difficult regions

- ▶ Distance nearest connected component not containing node **c**
- ▶ Use information collected by the local planner
 - ▶ Number of failed connections



Multi-Stage-Sampling: identify difficult regions

- ▶ Distance nearest connected component not containing node **c**
- ▶ Use information collected by the local planner
 - ▶ Number of failed connections

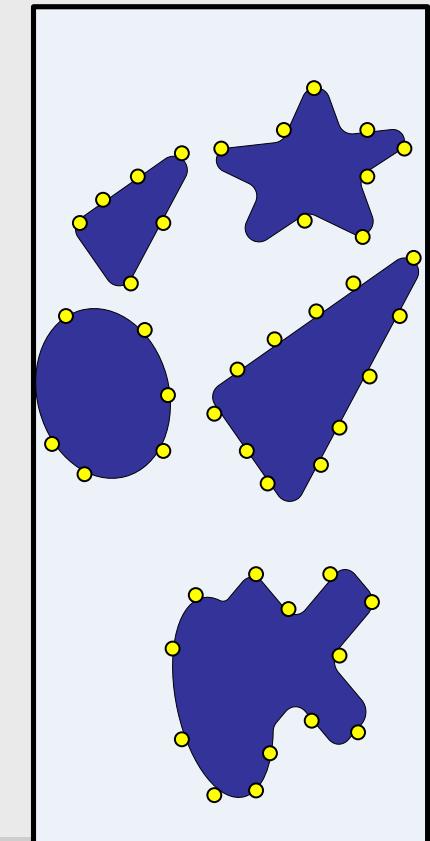


Multi-Query-PRM

- ▶ Typical strategies:
 - ▶ Multi-Stage-Sampling
 - ▶ **Obstacle-Based-Sampling**
 - ▶ Narrow-passage-Sampling

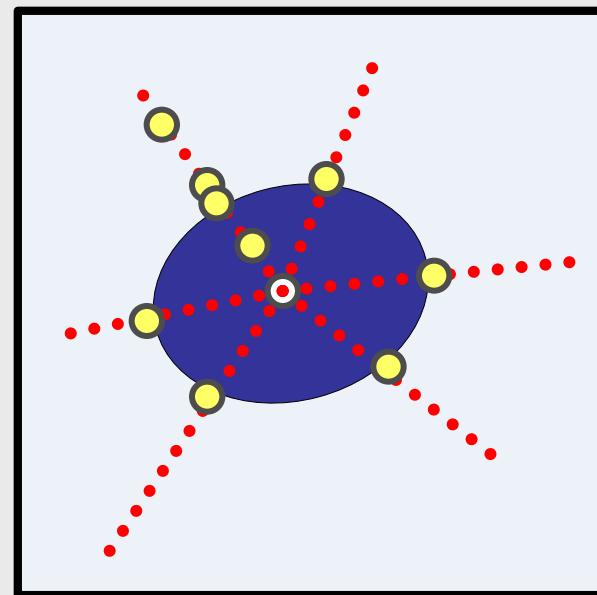
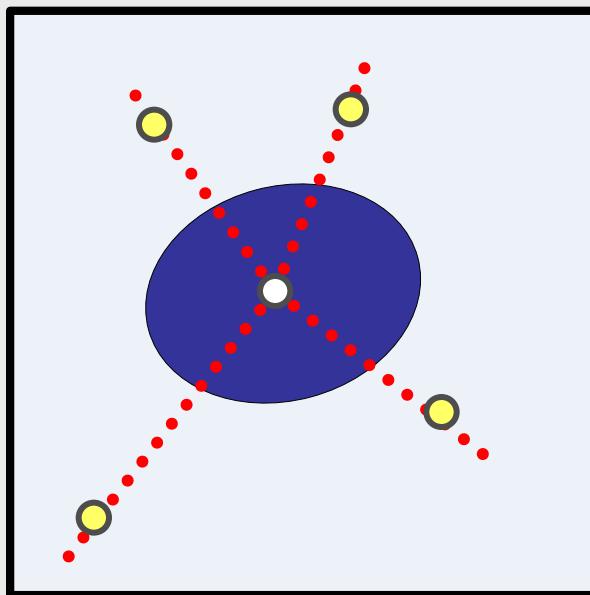
Idea

- ▶ Examination of connectivity of free space is most difficult nearby obstacles → samples needed in these regions
 - ▶ Ray-Cast-Sampling
 - ▶ Amato, Bayazit, Dale, Jones and Vallejo
 - ▶ Overmars
 - ▶ Gaussian Sampling
 - ▶ Boor, Overmars, van der Stappen



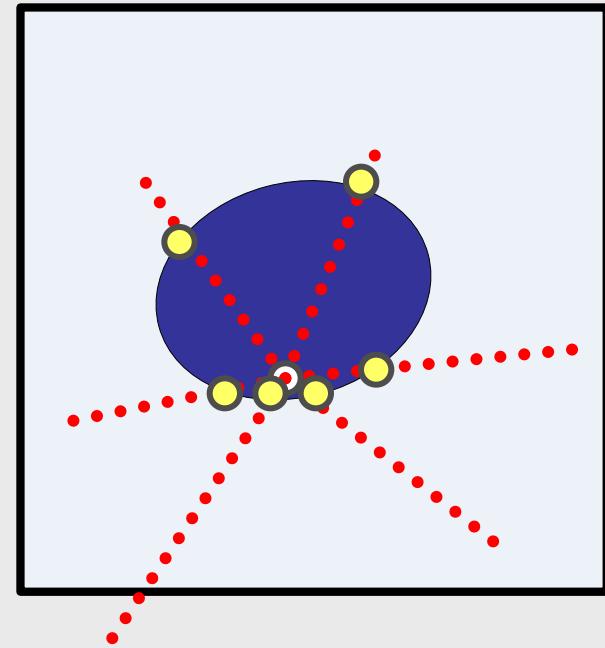
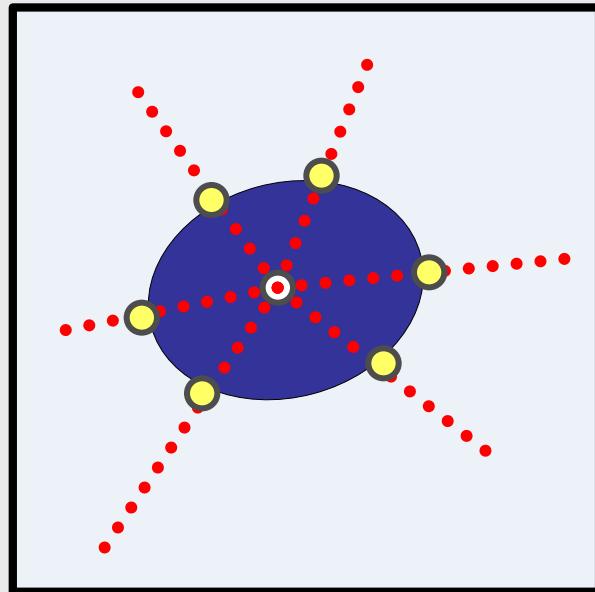
Obstacle-Based-Sampling: Ray-Cast-Sampling

- ▶ Use rays – to find nodes **near** obstacles
- ▶ Use rays – to find nodes **on** obstacles
 - ▶ Compute ray's intersection with obstacle using bisection
 - ▶ Try to uniformly distribute nodes on surface



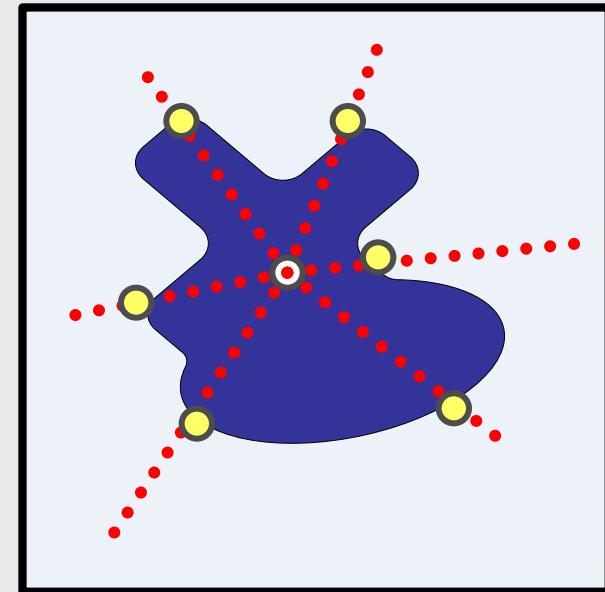
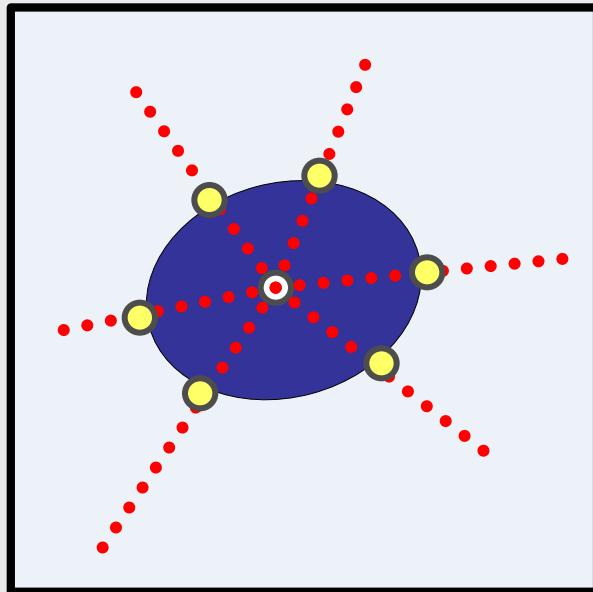
Ray-Cast-Sampling: Issues

- ▶ How to find start-point of rays to achieve uniform distribution?



Ray-Cast-Sampling: Issues

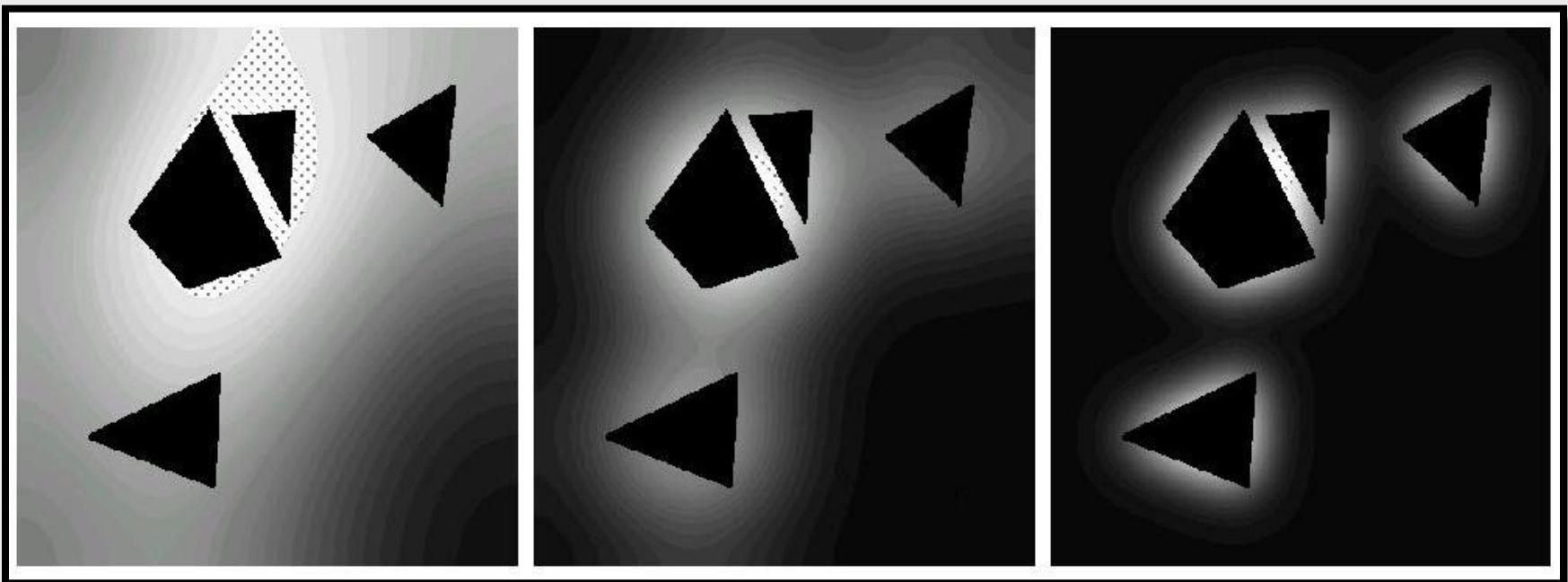
- ▶ How to handle skewed objects to achieve uniform distribution?
 - ▶ Finding right points in obstacles (**begin** of ray)
 - ▶ Finding right points on obstacles (**choosing** of rays)



Not as good in finding the true form of the obstacles

Obstacle-Based-Sampling: Gaussian Sampling

- ▶ Generate nodes nearby obstacles based on a gaussian distribution

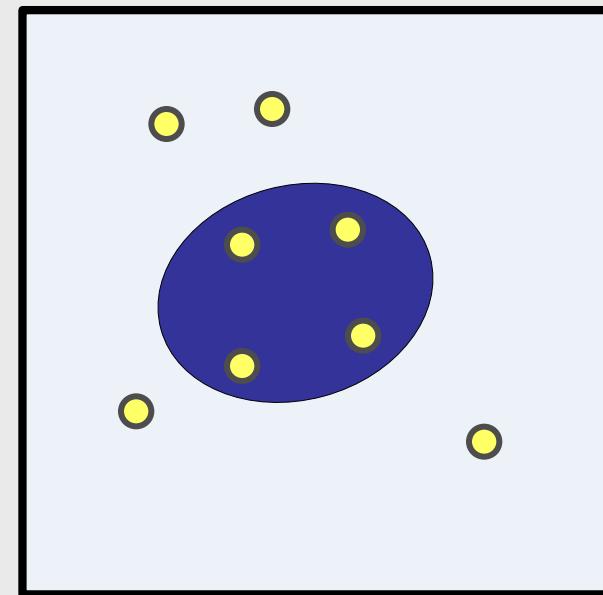


Obstacle-Based-Sampling: Gaussian Sampling

- ▶ How to compute the gaussian distribution ...
Remember goal: **avoid computation of obstacles in C-space**
- ▶ Implementation
 - ▶ **Sample pairs of configuration** such that the second is at distance d from the first.
 - ▶ **Discard pairs where both are free or both are forbidden.**
 - ▶ From remaining pairs, **keep free configuration.**

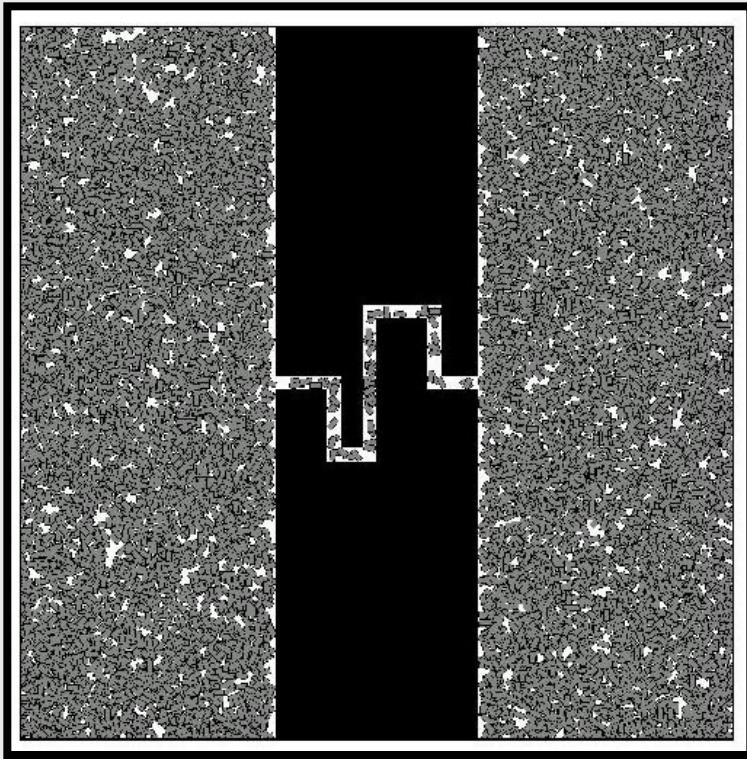
Gaussian Sampling: Algorithm

```
1. loop
2.   c1      ← a random configuration
3.   d        ← a distance chosen according to
              a normal distribution
4.   c2      ← a random conf. at distance d from c1
5.   if c1 in Cfree and c2 not in Cfree then
6.       add c1 to the graph
7.   else if c2 in Cfree and c1 not in Cfree then
8.       add c2 to the graph
9.   else
10.      discard both
```

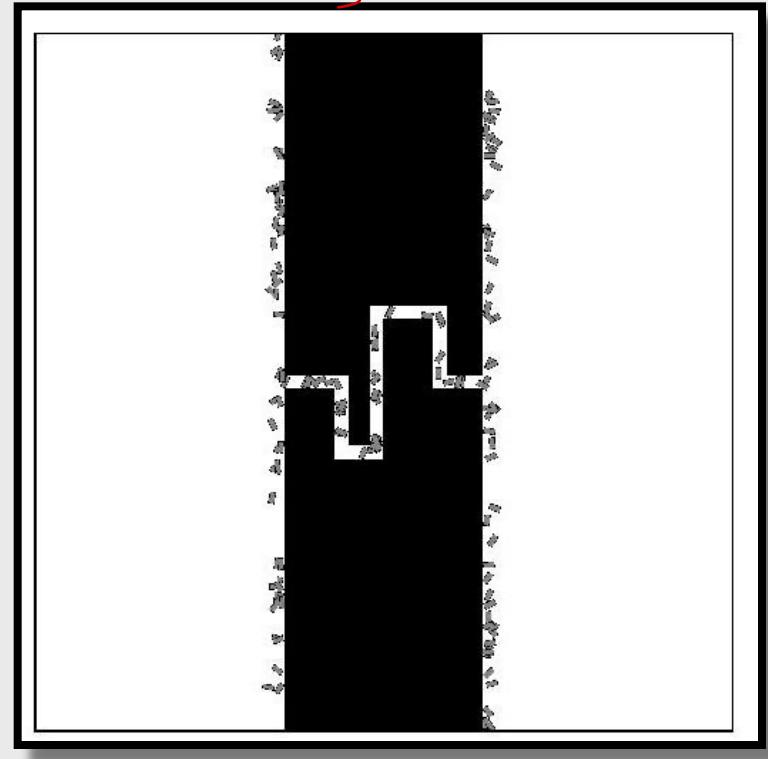


Gaussian Sampling

uniform



Gaussian Sampling



Obstacles-based-Sampling

- ▶ Problems:
 - ▶ Paths touch or are very close to obstacles
- ▶ Advantages:
 - ▶ Manipulation tasks on surfaces („near obstacles“)
 - ▶ Extremely cluttered C-Space can be examined

Sehr gut, lange Rechenzeit

Multi-Query-PRM

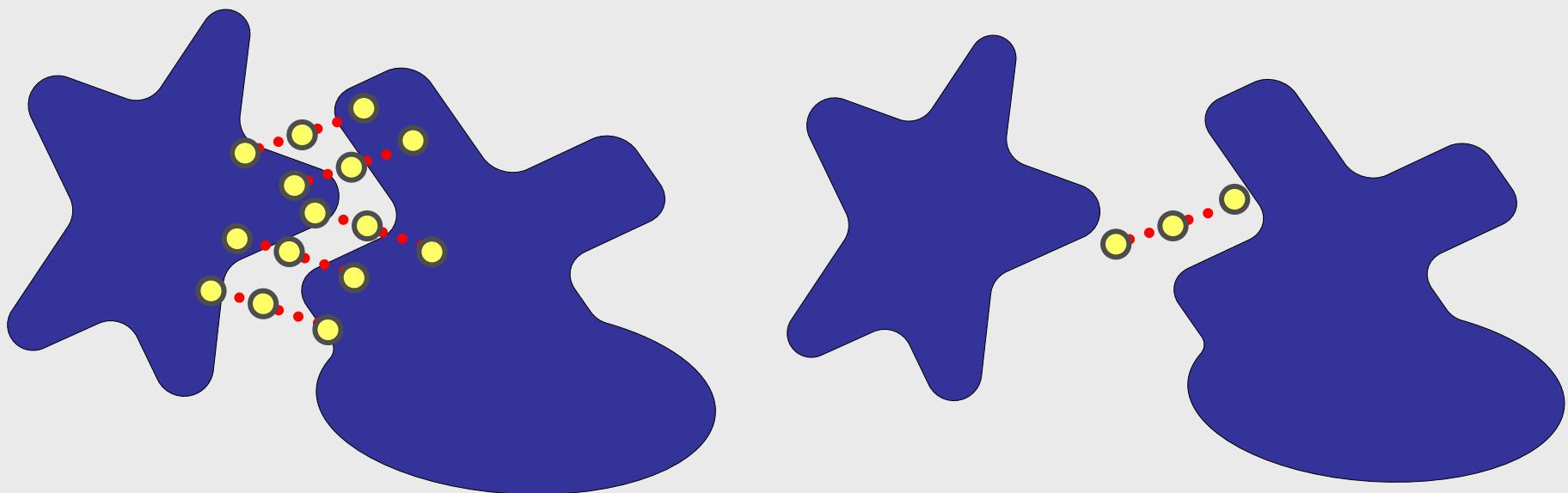
- ▶ Typical strategies:
 - ▶ Multi-Stage-Sampling
 - ▶ Obstacle-Based-Sampling
 - ▶ Narrow-passage-Sampling

Multi-Query-PRM: Narrow-Passage-Sampling

- ▶ Idea
- ▶ Finding ways through narrow passages is main problem, don't need to know whole contour of obstacle
 - ▶ Bridge Test
 - ▶ Hsu, Jiang, Reif, Sun
 - ▶ Medial Axis *(Wissen über Hindernisse bereits vorhanden → immer Mittelpunkt zwischen zwei Hindernissen)*
 - ▶ Amato, Kavraki

Narrow-Passage-Sampling: Bridge Test

- ▶ Add mid-point as a new node if two end-points are in collision and mid-point is collision-free
- ▶ Constrain the length of the bridge: Distinguish between “narrow” passages and normal obstacle distribution



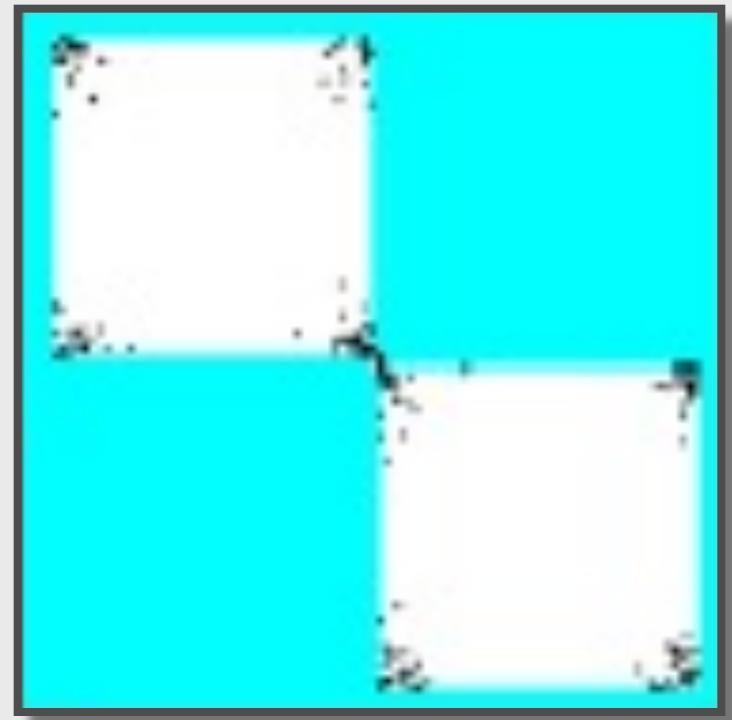
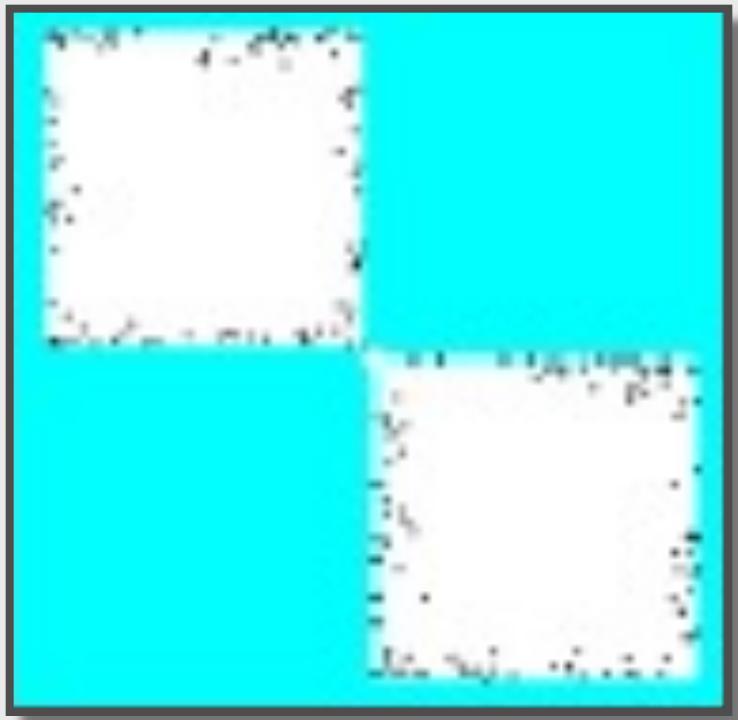
Bridge Test: Algorithm

repeat

```
Pick a point x from C-Space uniformly at random
if ( x is not colliding ) return FALSE else
    Pick a point x' in the neighborhood of x according
    to a suitable probability density  $\lambda_x$ .
    if ( x' is not colliding ) return FALSE else
        Set p to the midpoint of line segment xx'
        if ( p is colliding ) return FALSE else
            Insert p into G as a new node
```

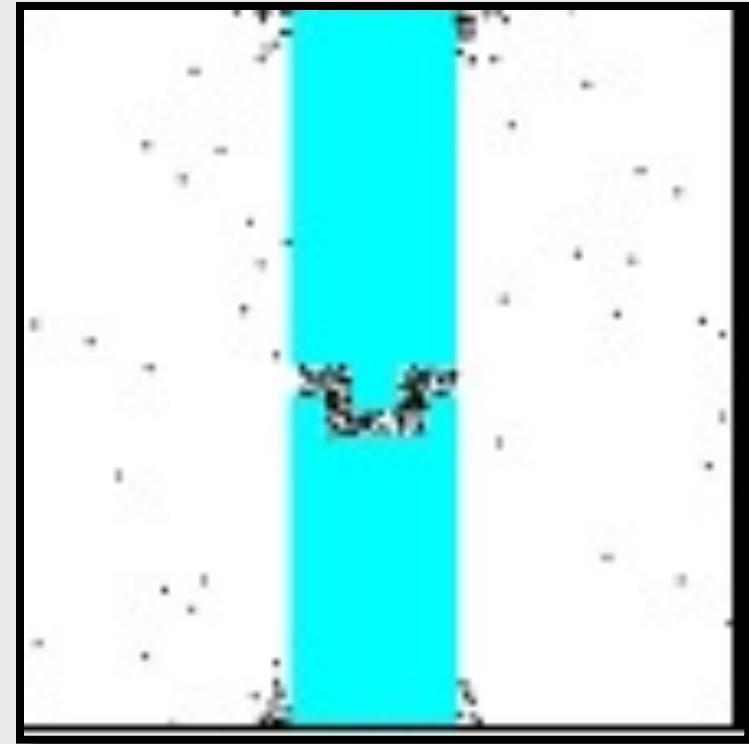
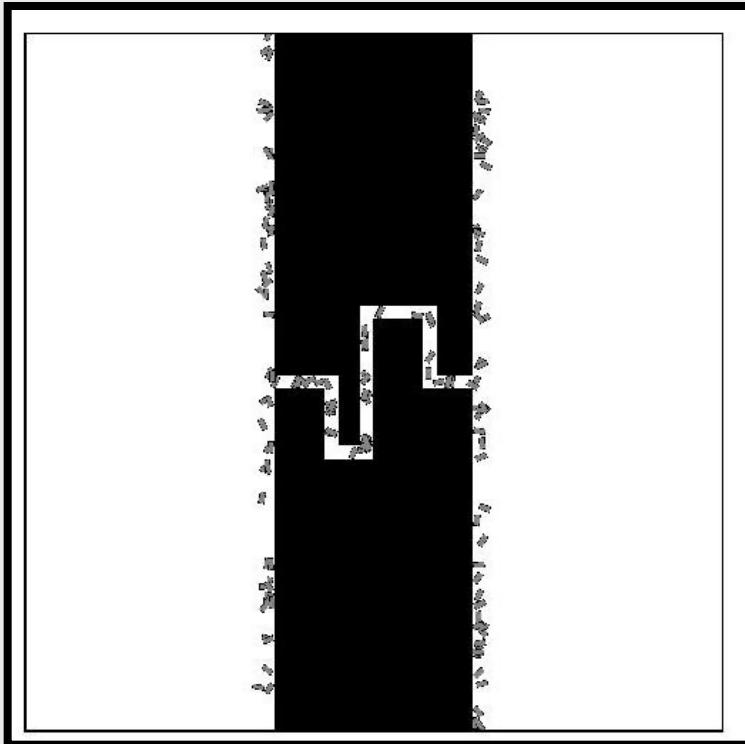
Bridge Test: Compared to Gaussian

- ▶ Left: Gaussian
- ▶ Right: Bridge Test



Bridge Test: Compared to Gaussian

- ▶ Left: Gaussian
- ▶ Right: Bridge Test



Bridge Test: features

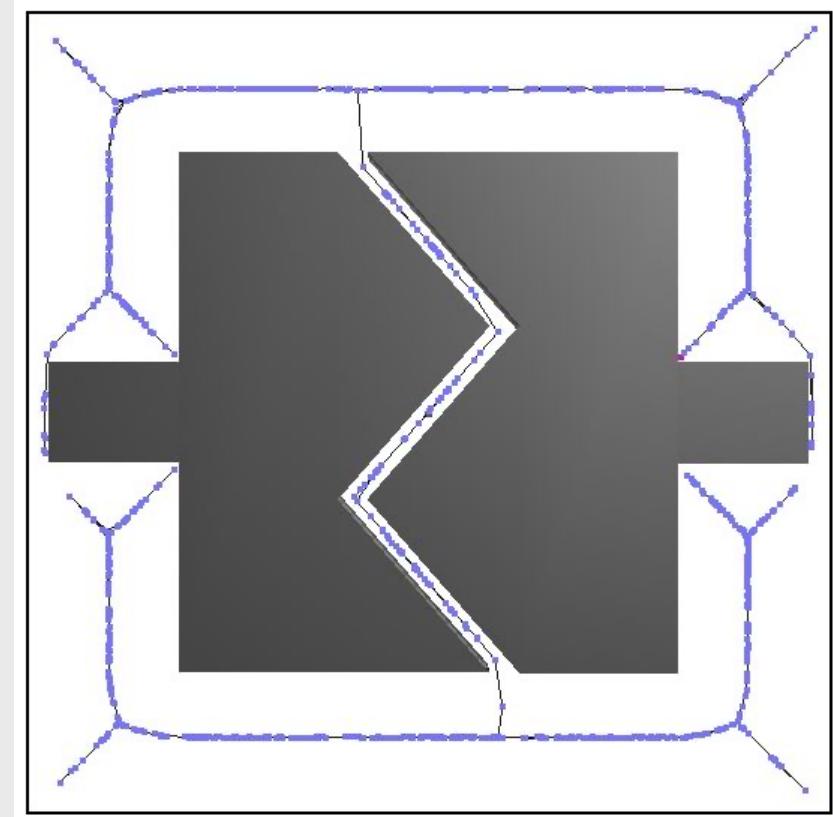
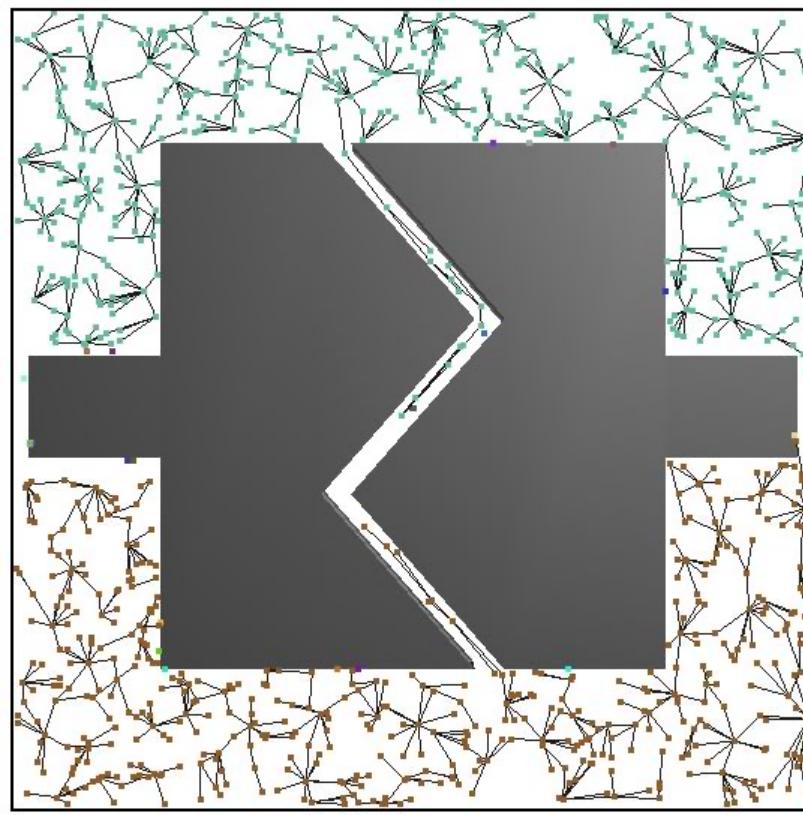
- ▶ Avoids sampling of “uninteresting” obstacle boundaries.
- ▶ Local Approach: Does not need to “capture” the C-obstacle in any sense
- ▶ The bridge test most likely yields a high rejection rate of configurations
 - general it results in a much smaller number of milestones
 - hence much fewer connections to be tested
 - testing connections is costly
 - can be significant computational gain

Bridge Test: Issues

- ▶ Deciding the probability density (π_B) around a point P , which has been chosen as first end-point.
- ▶ Combining Bridge Builder and Uniform Sampling
 - ▶ $\pi = (1-w) * \pi_B + w * \pi_v$ (*remember A* heuristic*)
 - ▶ π_B : probability density induced by the Bridge Builder
 - ▶ π_v : probability density induced by uniform sampling

Narrow-Passage-Sampling: Medial Axis

- ▶ **Advantage** of having samples on the medial axis → security distance from obstacles.
- ▶ **Problem** computation of medial axis itself is costly.



Narrow-Passage-Sampling: Limitations

- ▶ Costly actions in complex problems (number of surfaces & objects): “**Nearest Contact Configuration**”. → significantly increases computation time
- ▶ Extension to higher dimensions difficult. Which direction to search for nearest contact?

Single-Query-PRM

- ▶ Typical strategies:
 - ▶ Diffusion
 - ▶ Adaptiv-Step
 - ▶ Biased Sampling
 - ▶ Control-Based Sampling

Single-Query-PRM: Diffusion

► Density-based strategy

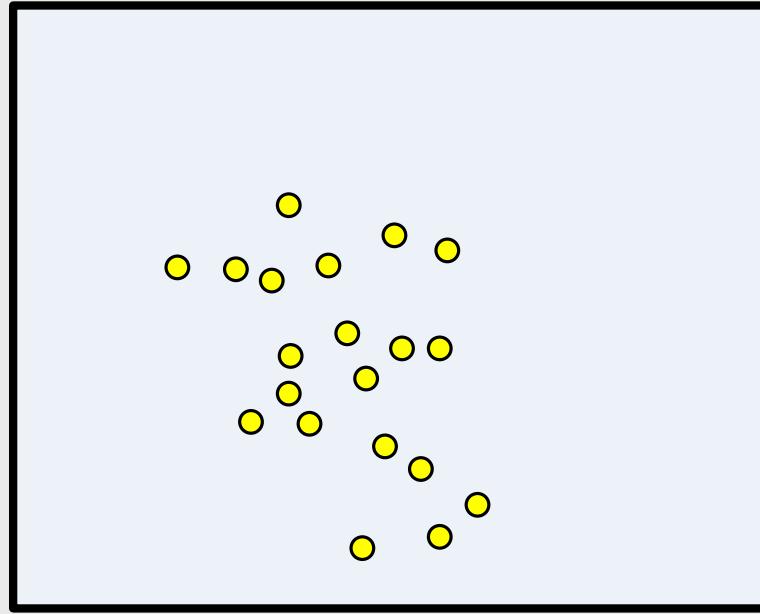
- ▶ Associate a sampling density to each milestone in the trees
- ▶ Pick a milestone m at random with probability inverse to density (Remember cell density SBL-Planner)
- ▶ Expand from m

► RRT strategy

- ▶ Pick a configuration q uniformly at random in c-space
- ▶ Select the milestone m the closest from q
- ▶ Expand from m

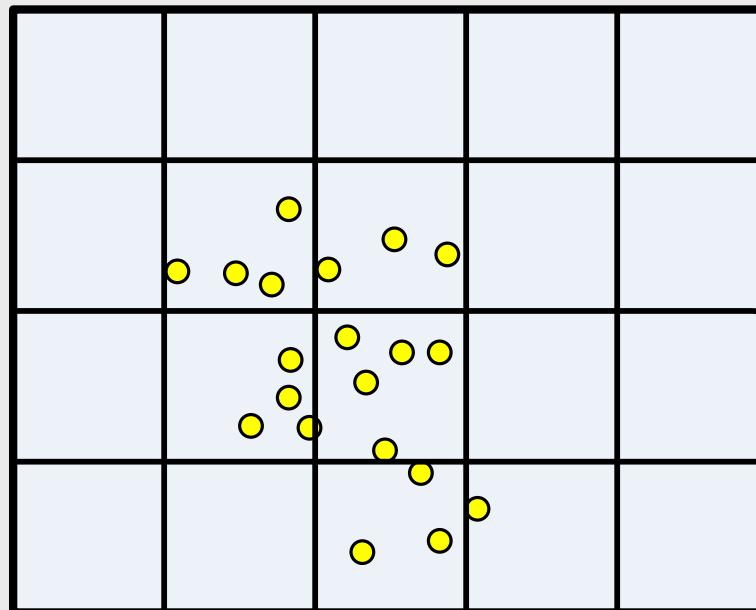
Density-based strategy

- ▶ Don't maintain density around each node → overhead



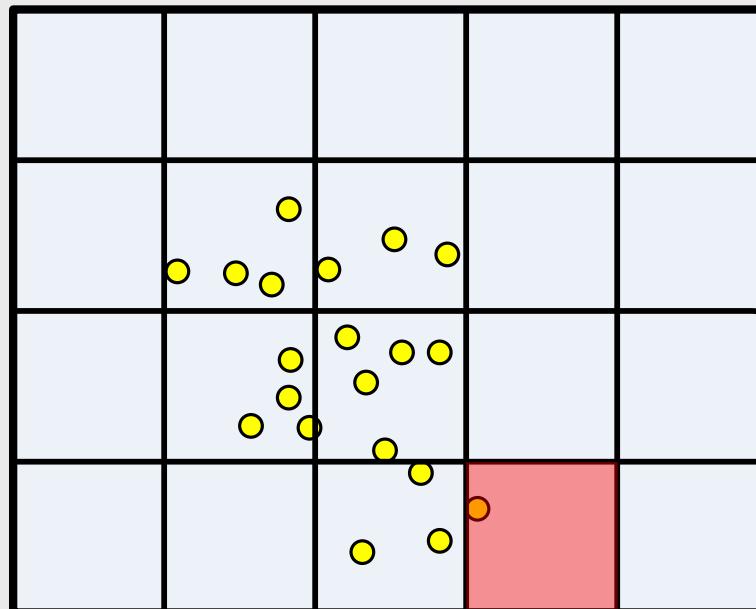
Density-based strategy

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally seized cells and maintain density in these cells



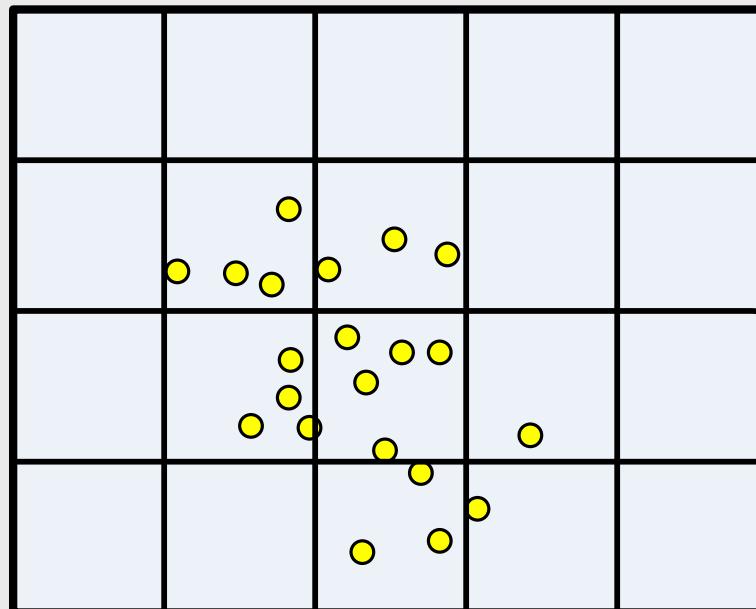
Density-based strategy

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally sized cells and maintain density in these cells
- ▶ Choose cell having lowest density with highest probability



Density-based strategy

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally sized cells and maintain density in these cells
- ▶ Choose cell having lowest density with highest probability
- ▶ Generate new node



RRT-Strategy

- ▶ Pick a configuration q uniformly at random in c-space
- ▶ Select the milestone m the closest from q
- ▶ Expand from m



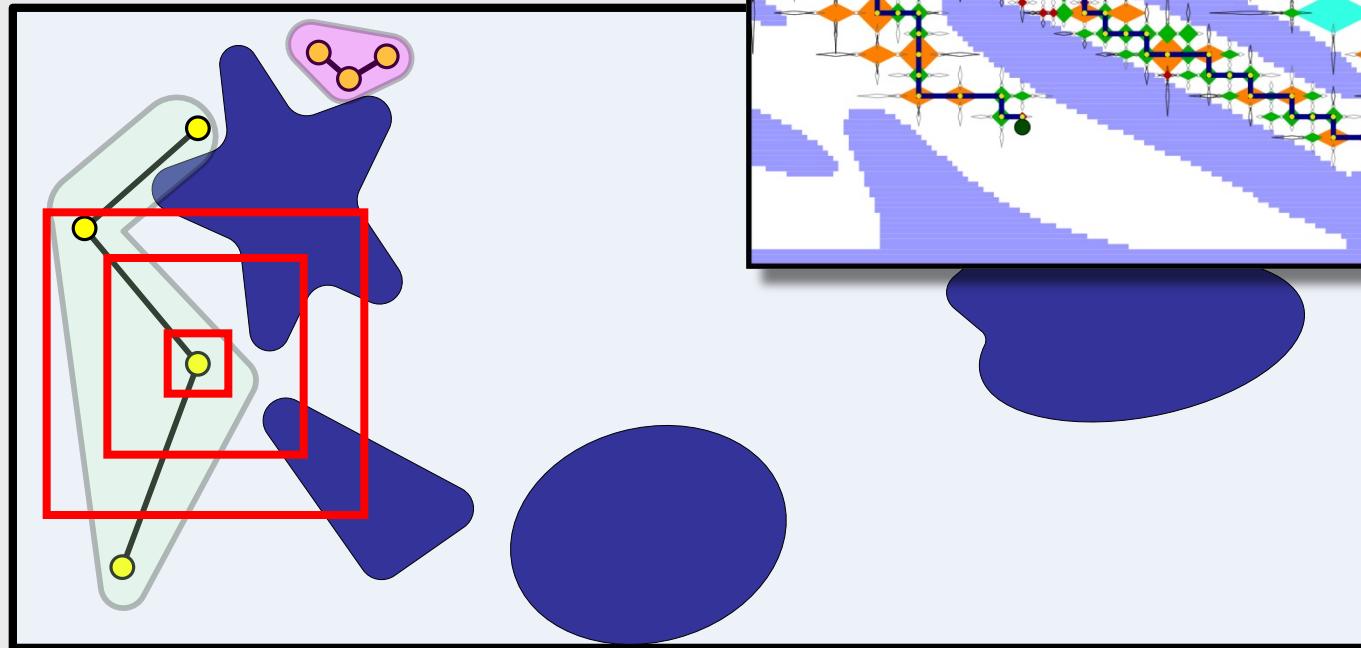
Single-Query-PRM

► Typical strategies:

- ▶ Diffusion
- ▶ Adaptiv-Step
- ▶ Biased Sampling
- ▶ Control-Based Sampling

Single-Query-PRM: Adaptive step size

- ▶ Make big steps in open space and small steps nearby obstacles
 - ▶ Adapt size of sampling window
 - ▶ Remember hierarchical A*



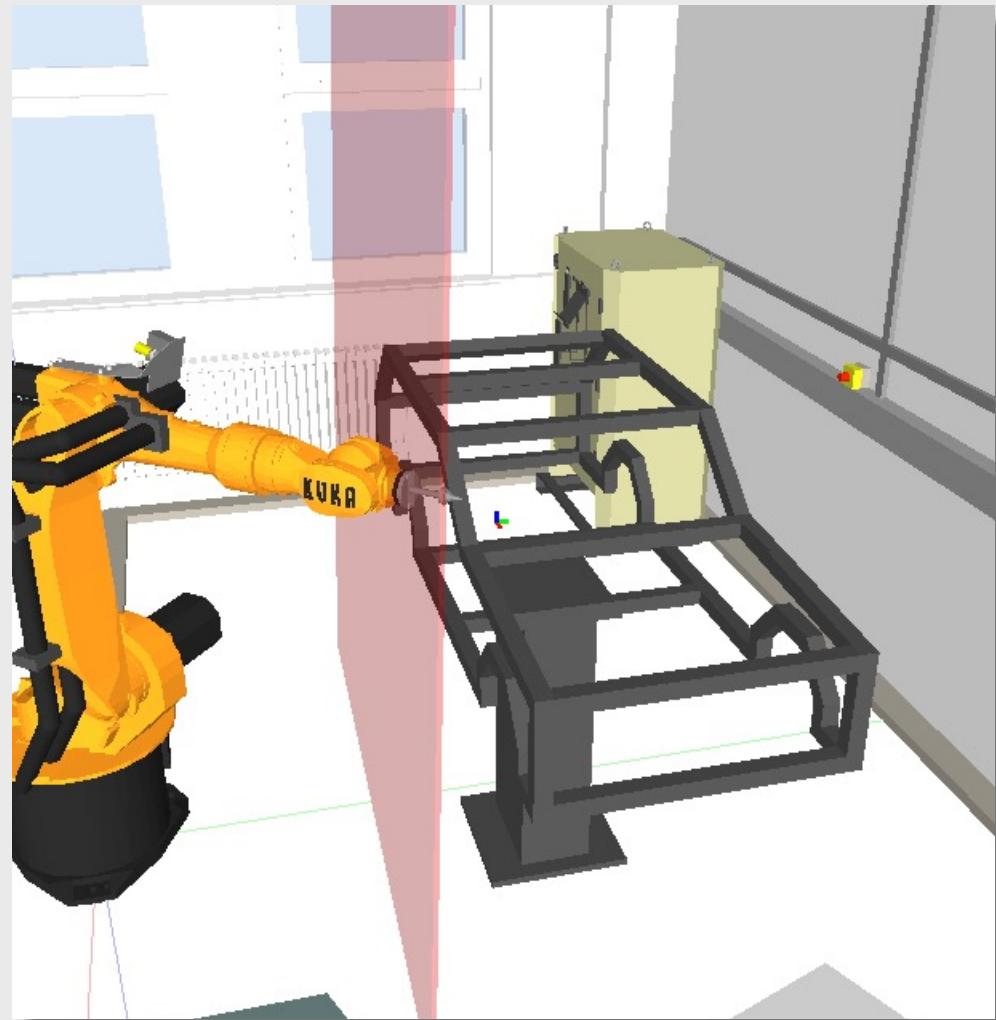
Single-Query-PRM

- ▶ Typical strategies:
 - ▶ Diffusion
 - ▶ Adaptiv-Step
 - ▶ **Biased Sampling**
 - ▶ Control-Based Sampling

Single-Query-PRM: Biased Sampling

- ▶ Heuristic added to search

- ▶ Favor sampling that TCP is near of the virtual red wall

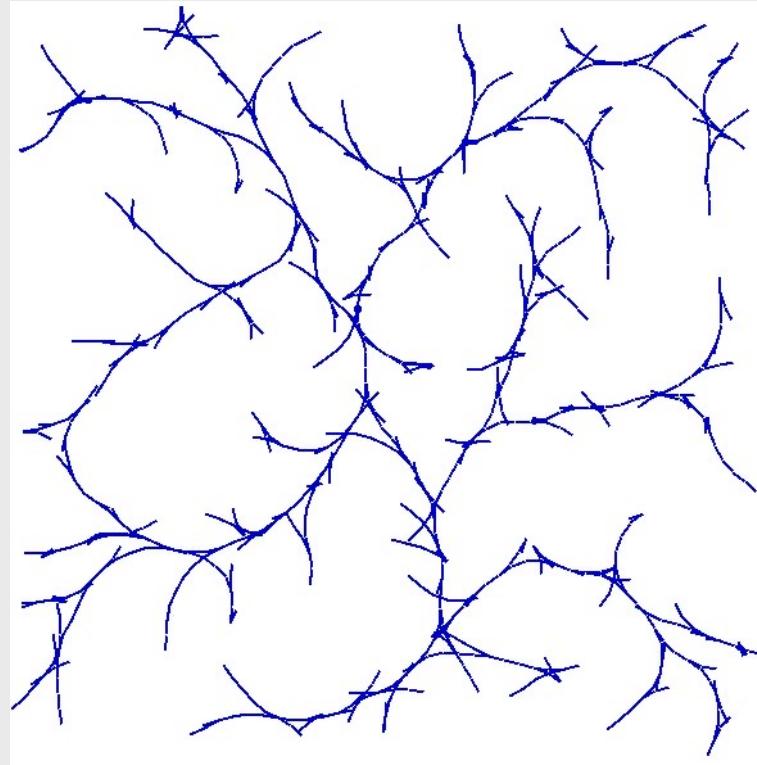


Single-Query-PRM

- ▶ Typical strategies:
 - ▶ Diffusion
 - ▶ Adaptiv-Step
 - ▶ Biased Sampling
 - ▶ Control-Based Sampling

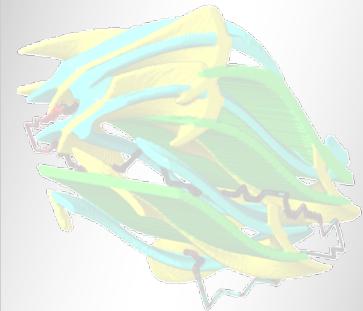
Single-Query-PRM: Control-Based Sampling

- ▶ Satisfy kinodynamik contraints (Typical RRT)



Visibility-PRM

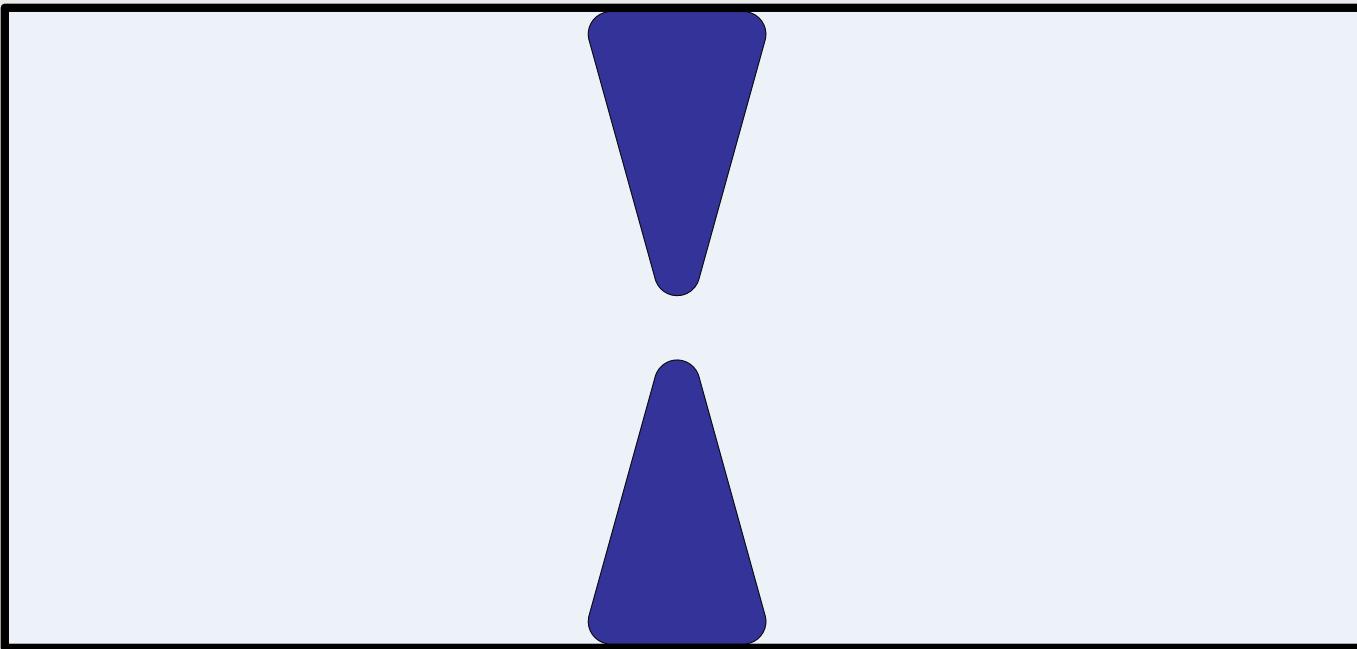
Effective creation of small roadmaps
covering most of search space (C-Space)



Motivation

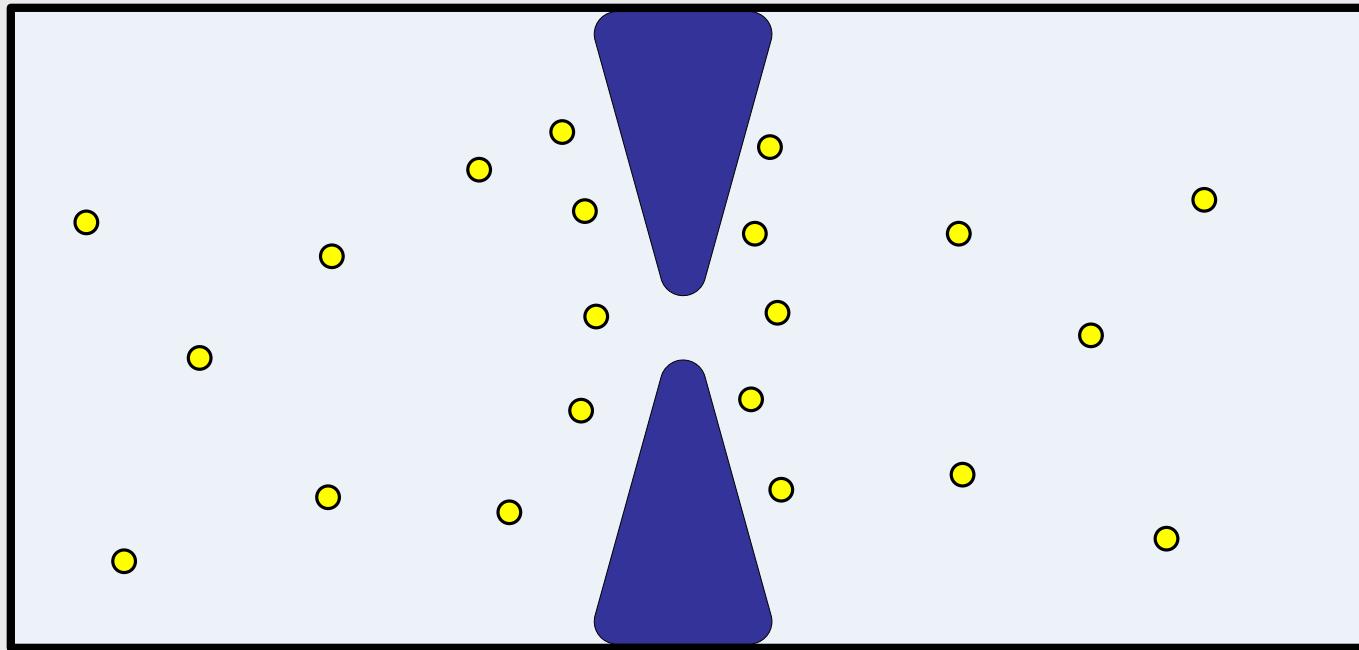
- ▶ Reduce number of nodes in roadmap but without loosing coverage and connectivity
 - ▶ Reducing memory consumption (due to smaller roadmap)
 - ▶ Reducing query-time (again due to smaller roadmap)
- ▶ Visibility-PRM is an optimized version of the classical PRM

Visibility-PRM: the idea behind



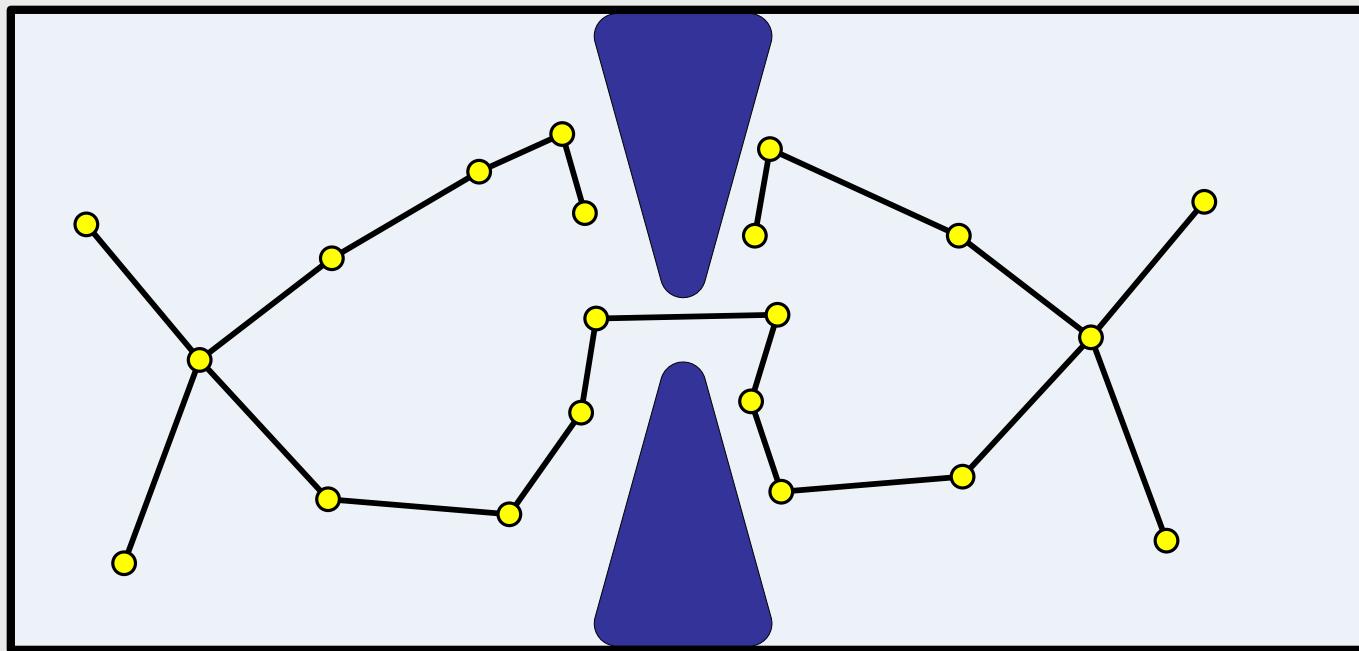
- ▶ Standard-PRM iteratively

Visibility-PRM: the idea behind



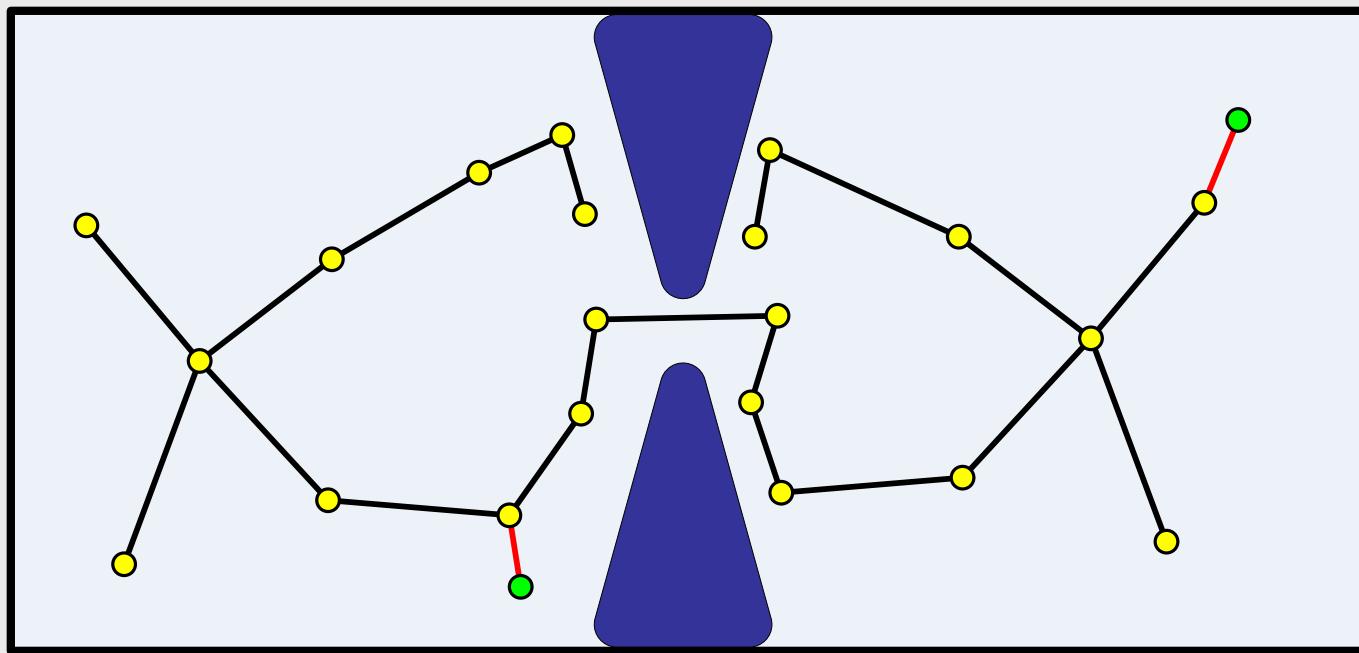
- ▶ Standard-PRM iteratively
 - ▶ fills C-Space with random nodes

Visibility-PRM: the idea behind



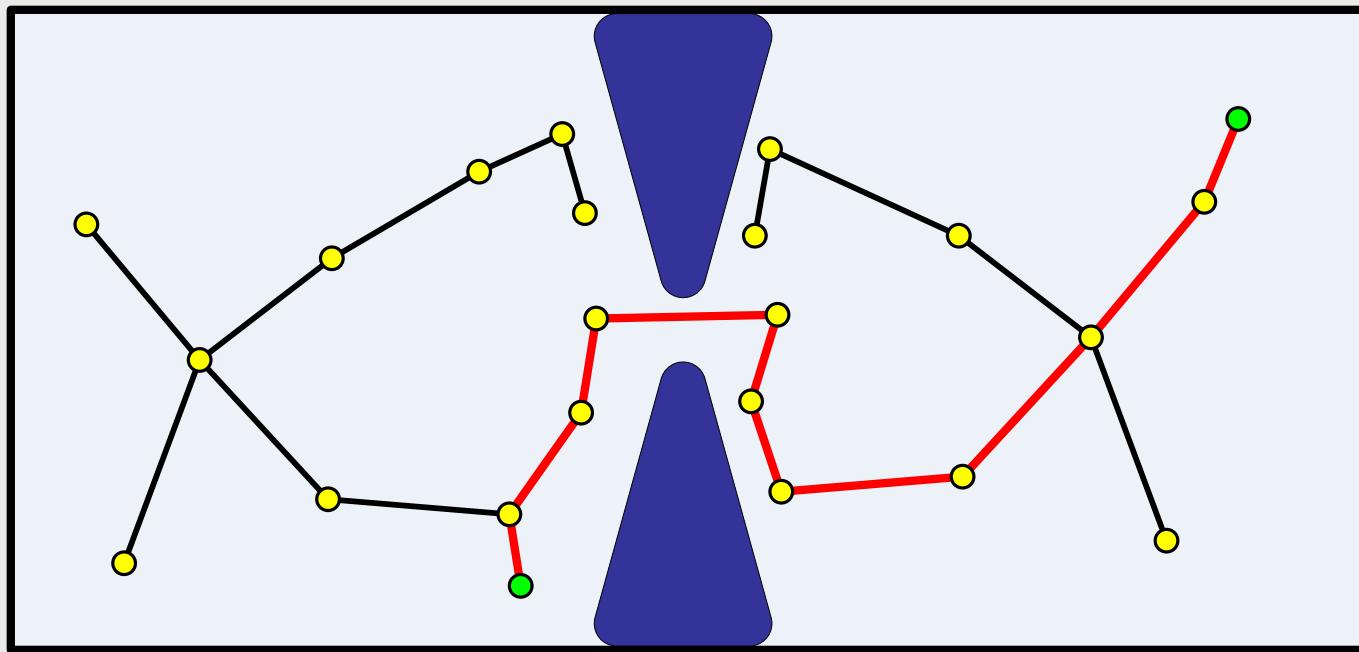
- ▶ Standard-PRM iteratively
 - ▶ fills C-Space with random nodes
 - ▶ builds Roadmap \mathbf{G}

Visibility-PRM: the idea behind



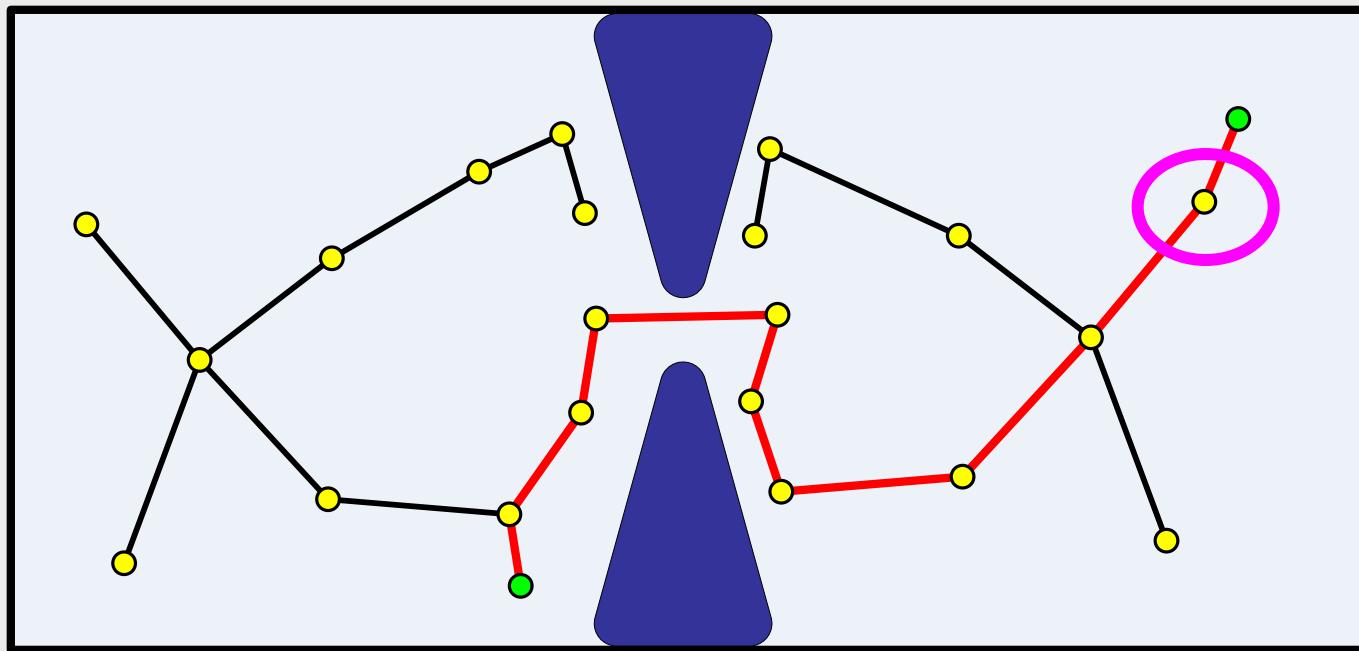
- ▶ Standard-PRM iteratively
 - ▶ fills C-Space with random nodes
 - ▶ builds Roadmap **G**
- ▶ Query: Connect start & goal to **G**

Visibility-PRM: the idea behind



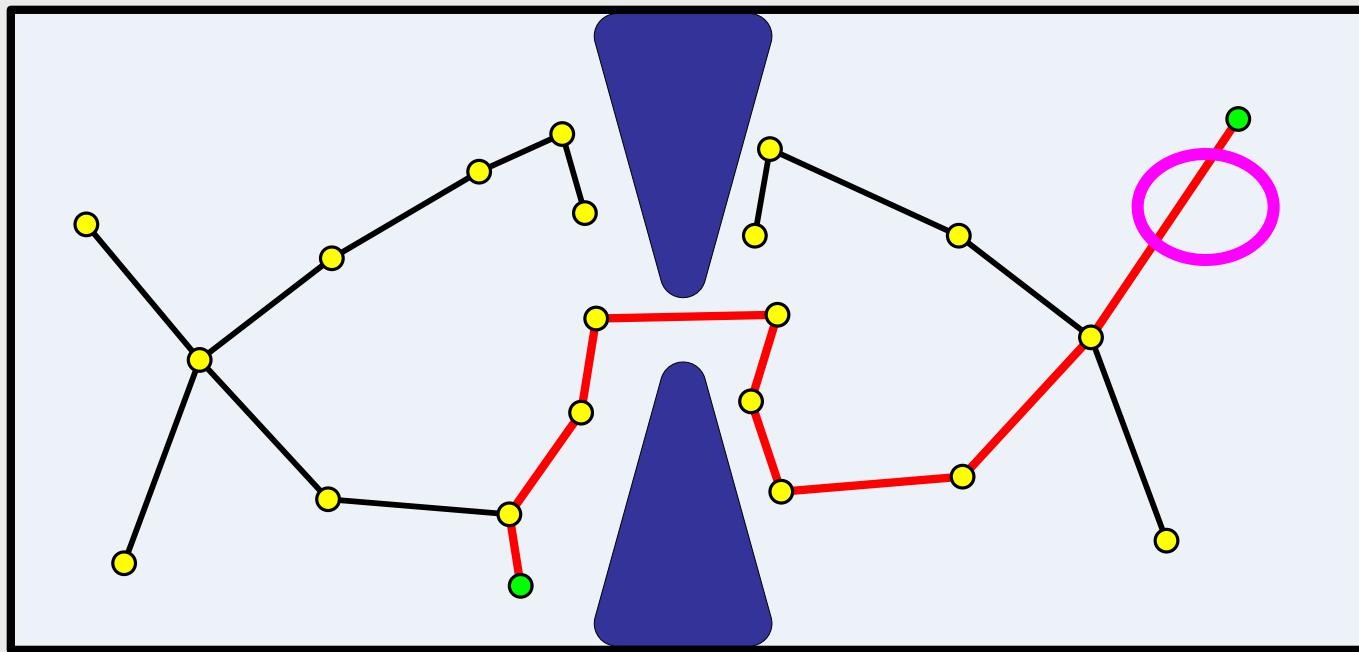
- ▶ Standard-PRM iteratively
 - ▶ fills C-Space with random nodes
 - ▶ builds Roadmap **G**
- ▶ Query: Connect start & goal to **G** and find path

Visibility-PRM: the idea behind



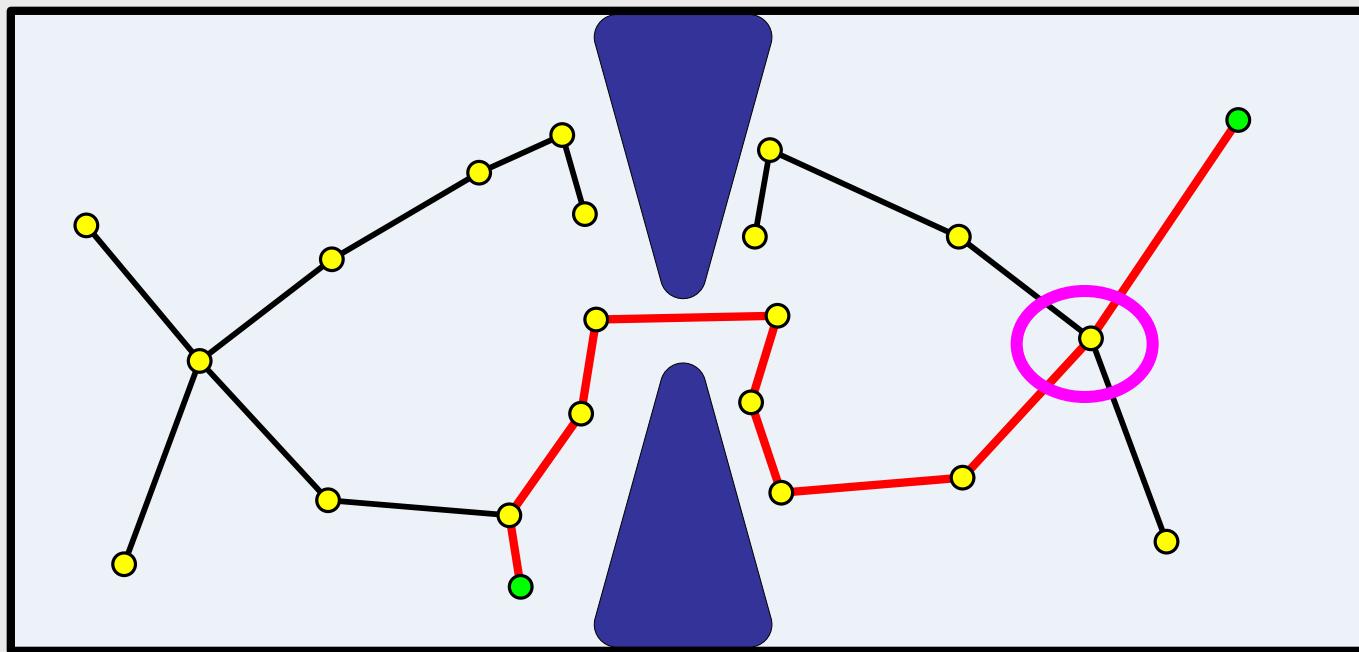
- ▶ Is this point for finding a solution path necessary?

Visibility-PRM: the idea behind



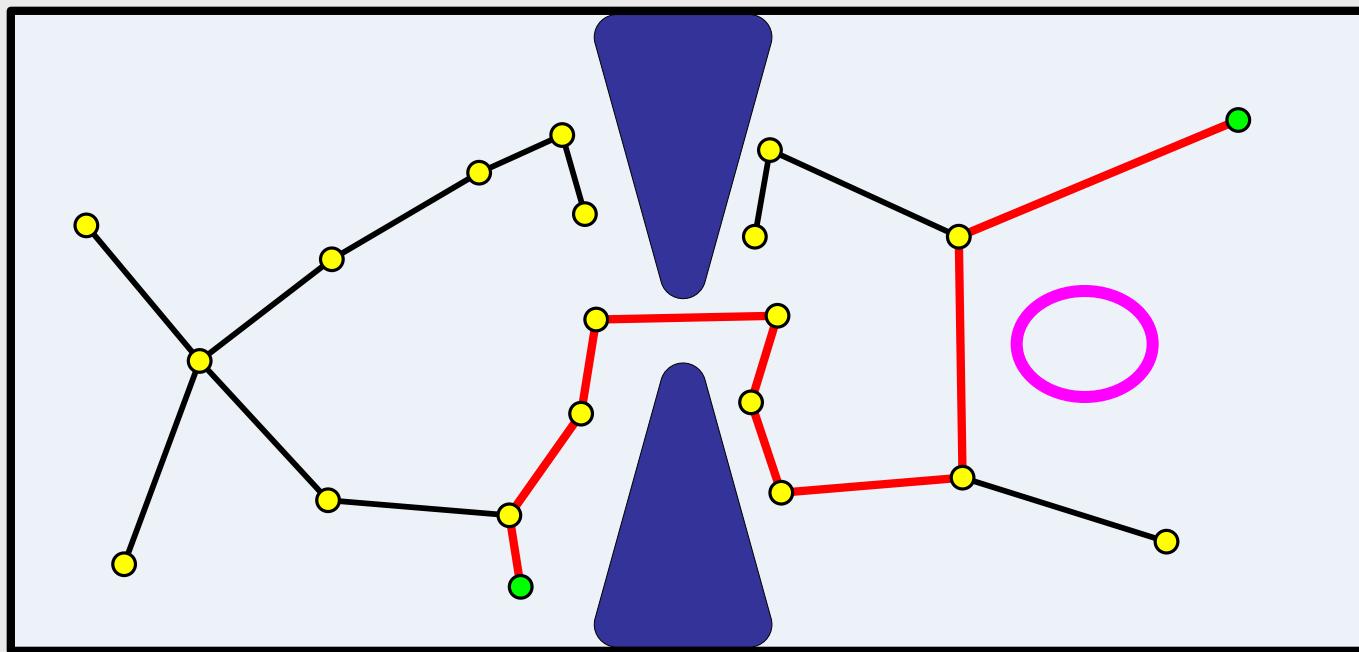
- ▶ Is this point for finding a solution path necessary? No

Visibility-PRM: the idea behind



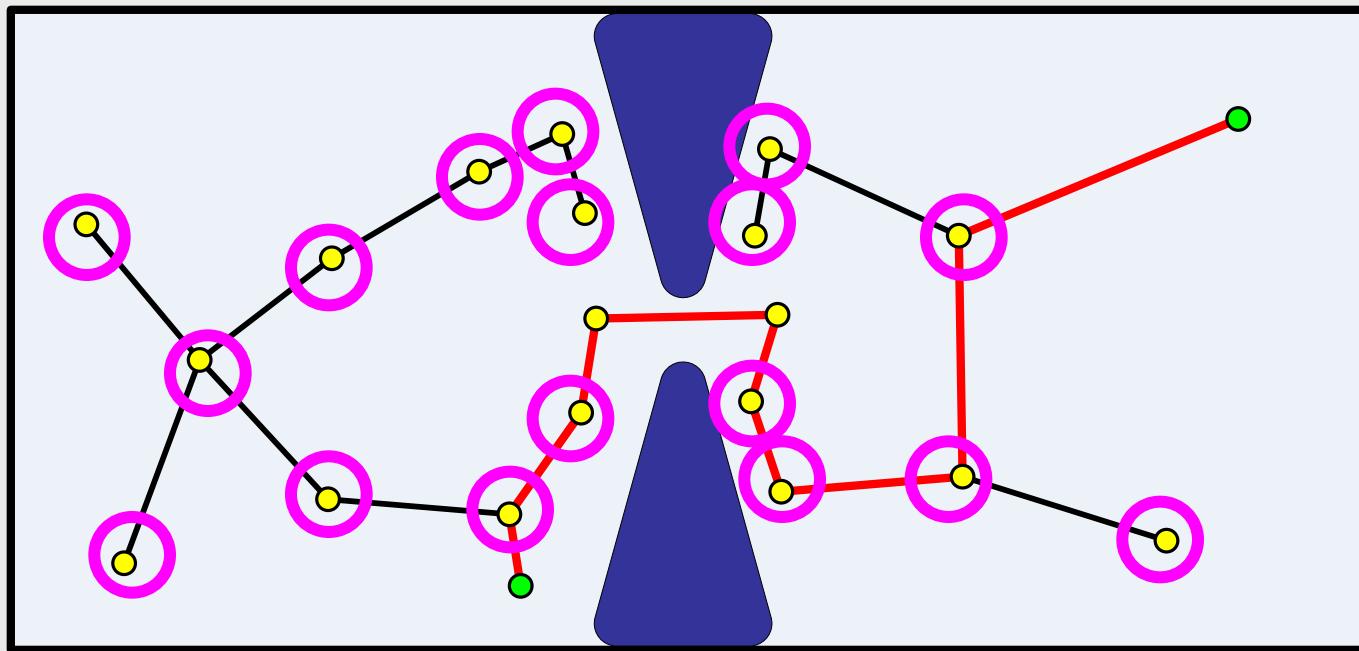
- ▶ Is this **point** for finding a solution path necessary?

Visibility-PRM: the idea behind



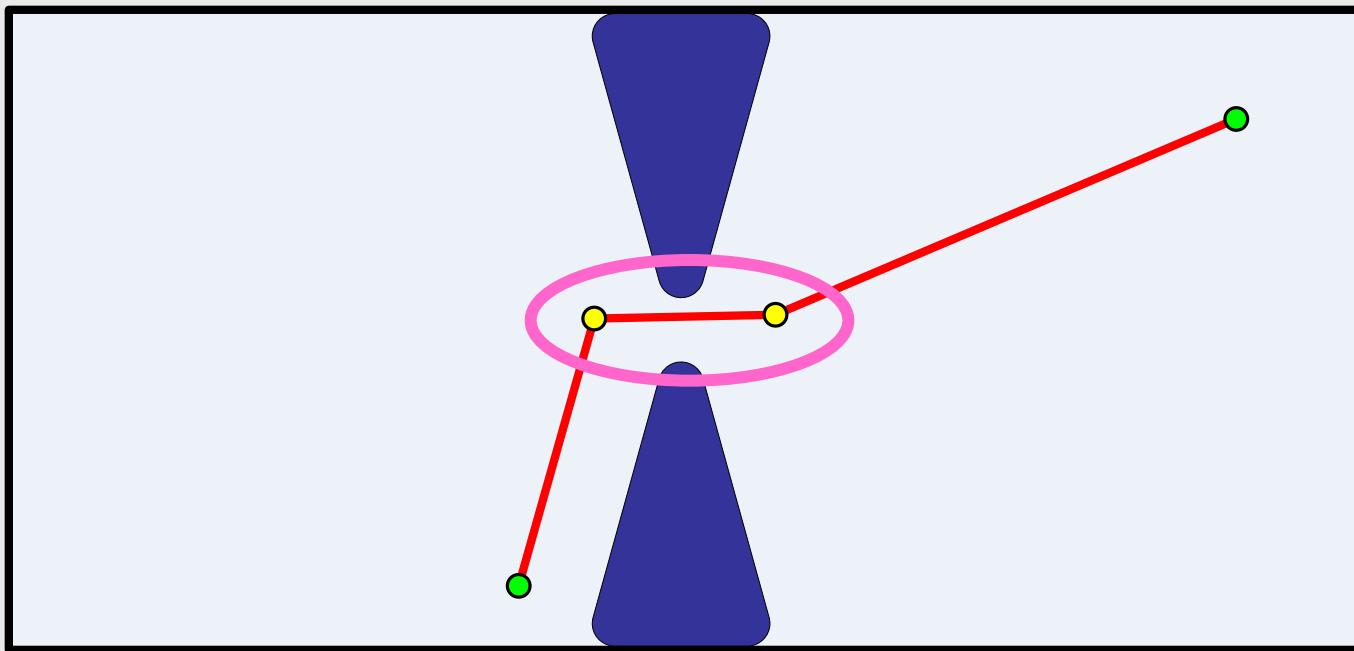
- ▶ Is this **point** for finding a solution path necessary? No

Visibility-PRM: the idea behind



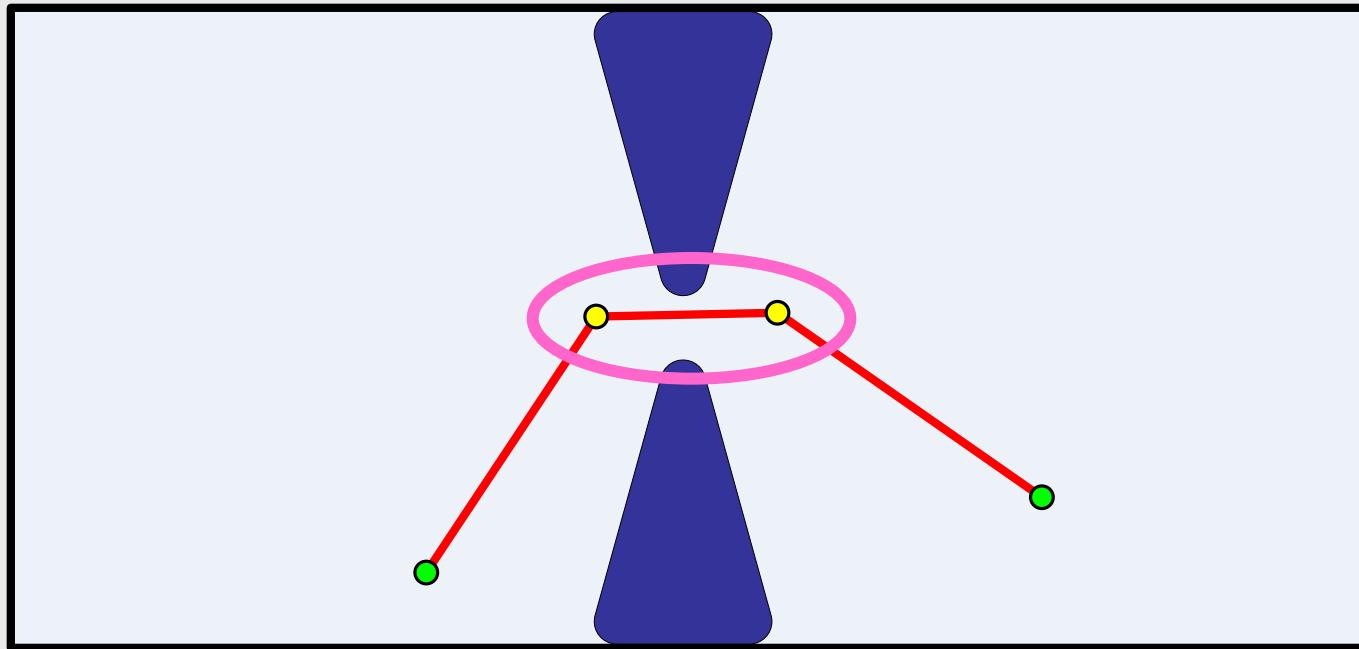
- ▶ All these **points** are not necessary for finding a solution path!!!!

Visibility-PRM: the idea behind



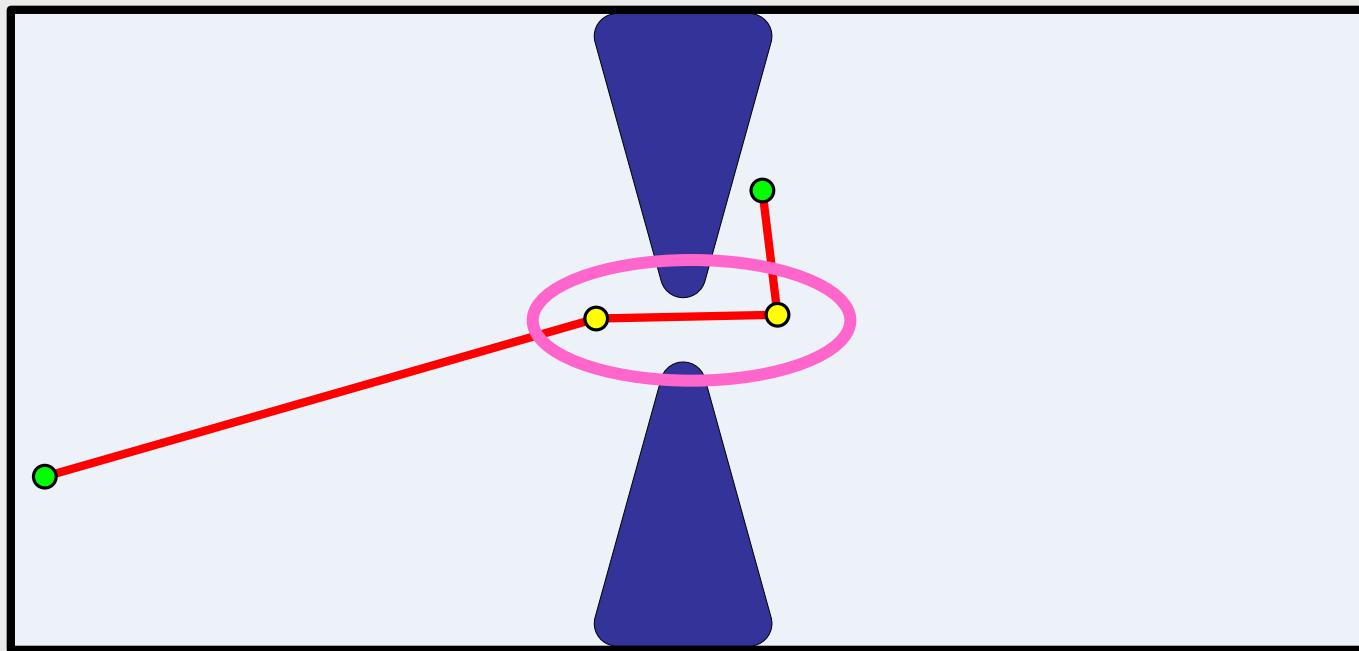
- ▶ This minimal „roadmap“ is valid for every start & goal position

Visibility-PRM: the idea behind



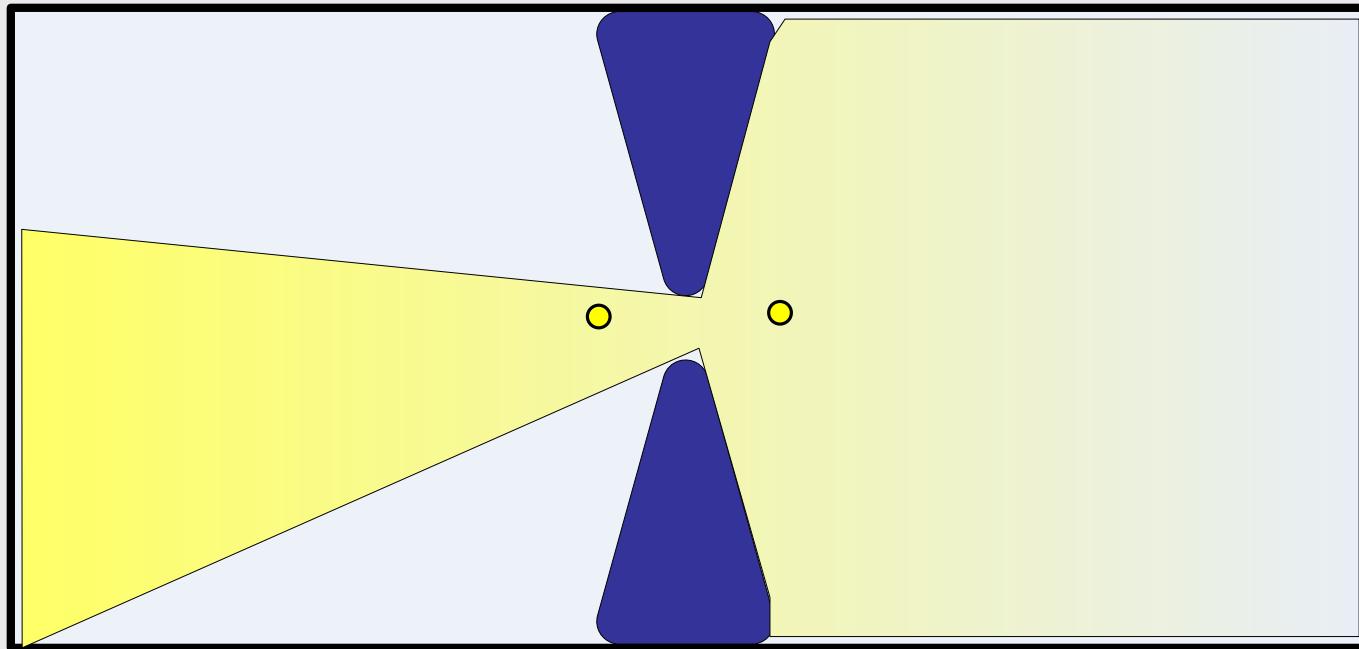
- ▶ This minimal „roadmap“ is valid for every start & goal position

Visibility-PRM: the idea behind



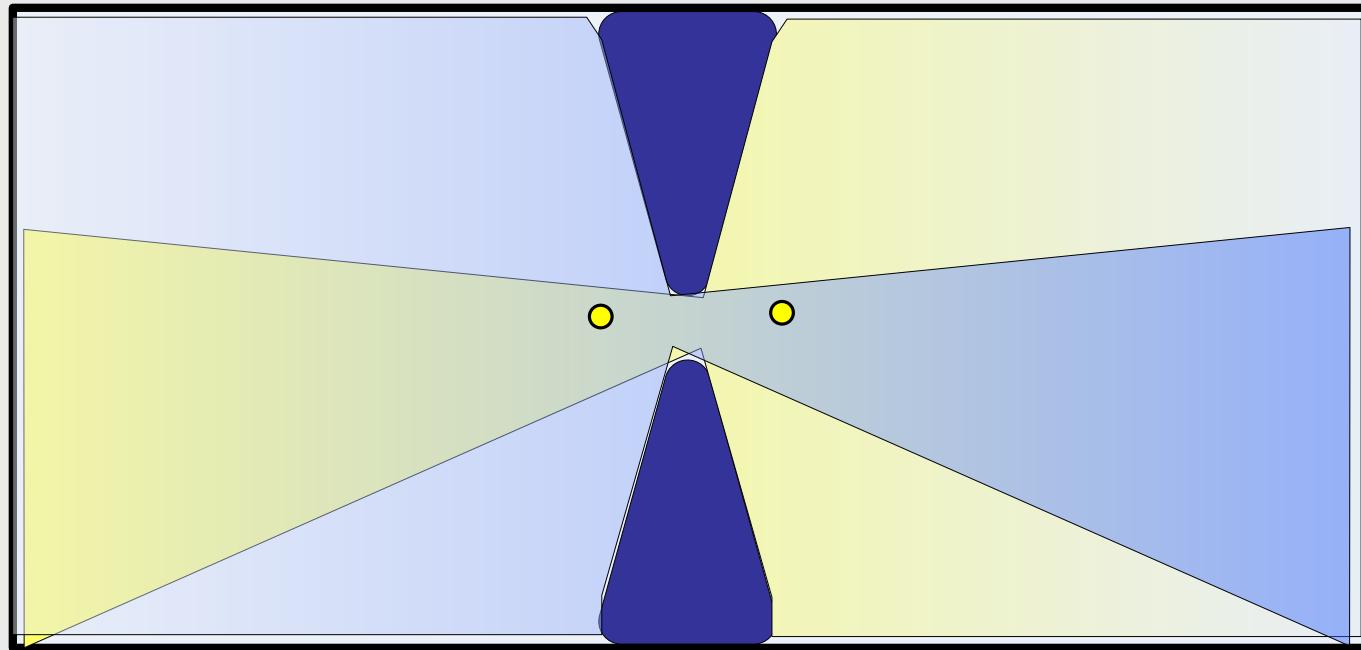
- ▶ This minimal „roadmap“ is valid for every start & goal position
- ▶ Why?

Visibility-PRM: the idea behind



- ▶ This minimal „roadmap“ is valid for every start & goal position
- ▶ Why? All possible points are visible from the two of the roadmap

Visibility-PRM: the idea behind



- ▶ This minimal „**roadmap**“ is valid for every start & goal position
- ▶ Why? All possible points are **visible** from the two of the roadmap

Some Terms

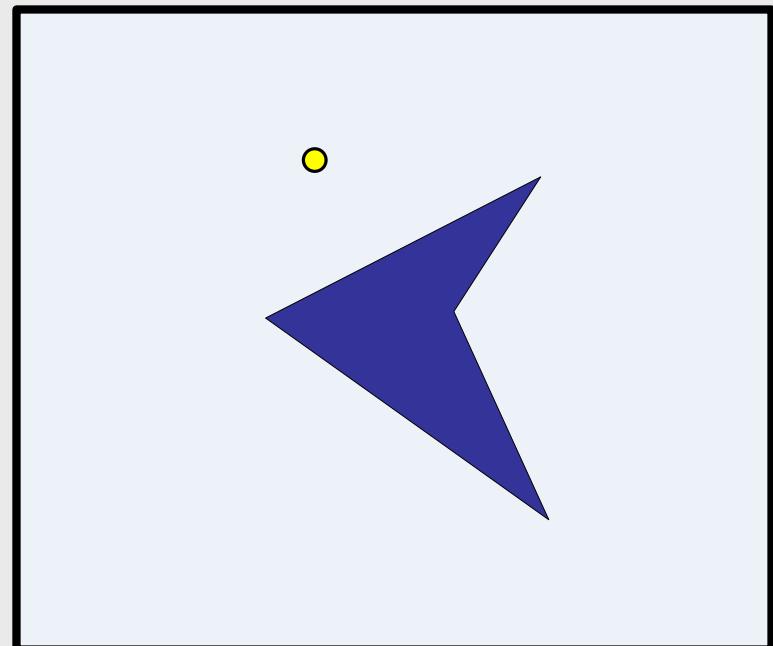
► Visibility domain, Guard and Connection:

- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$. Collision free
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.

Some Terms

► Visibility domain, Guard and Connection:

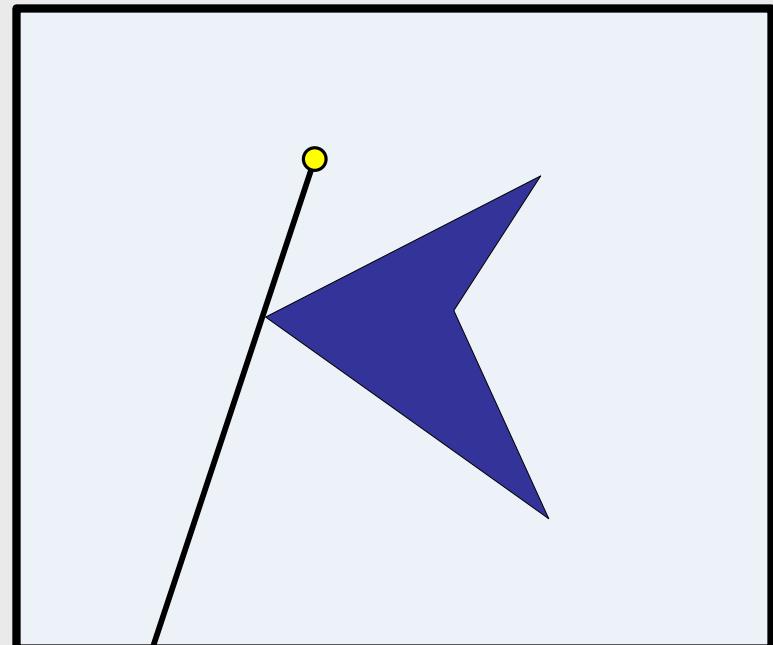
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

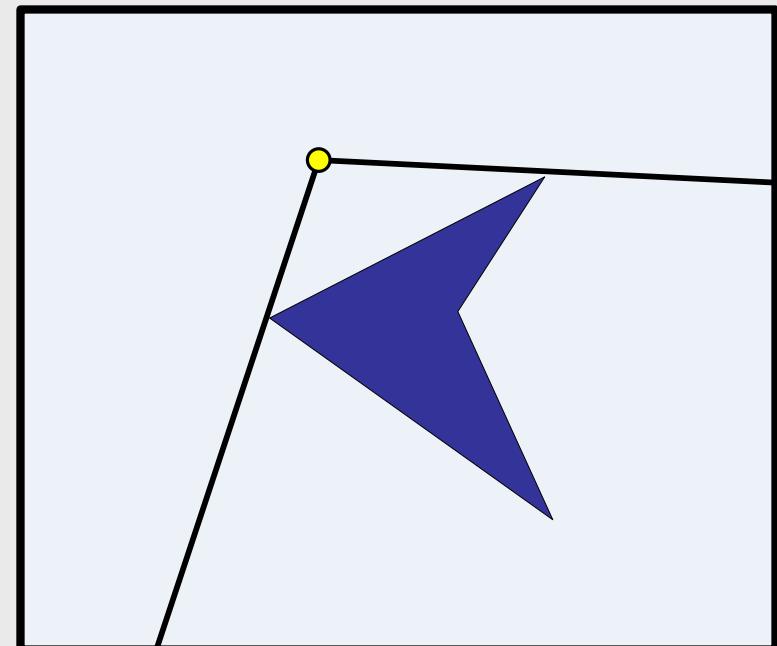
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

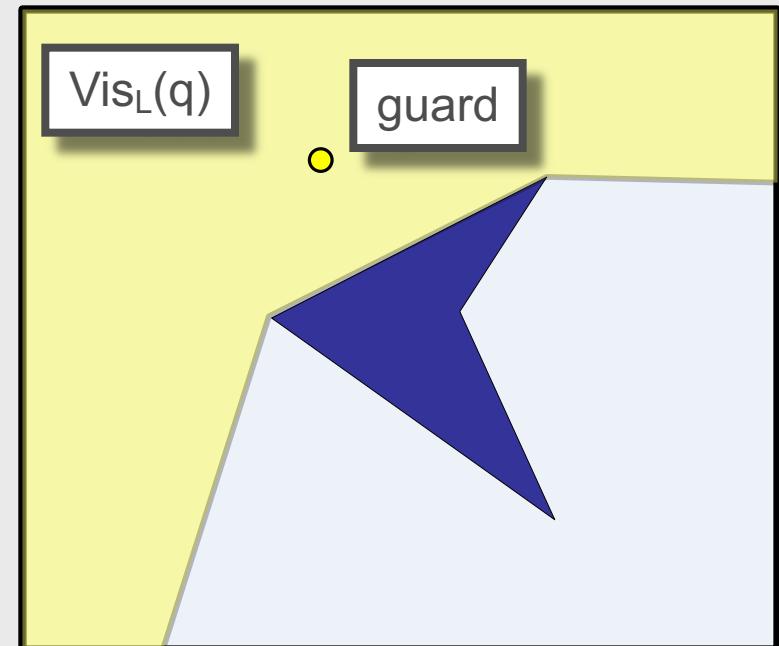
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

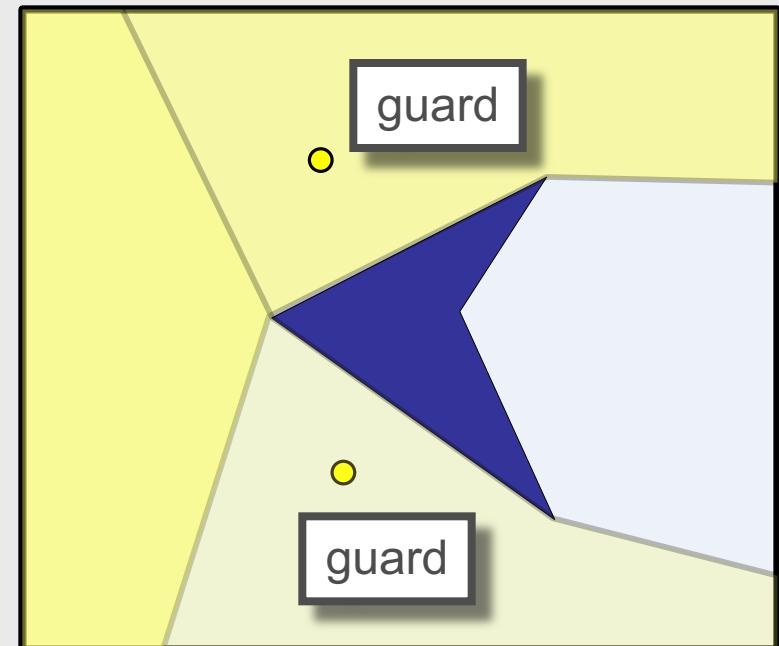
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

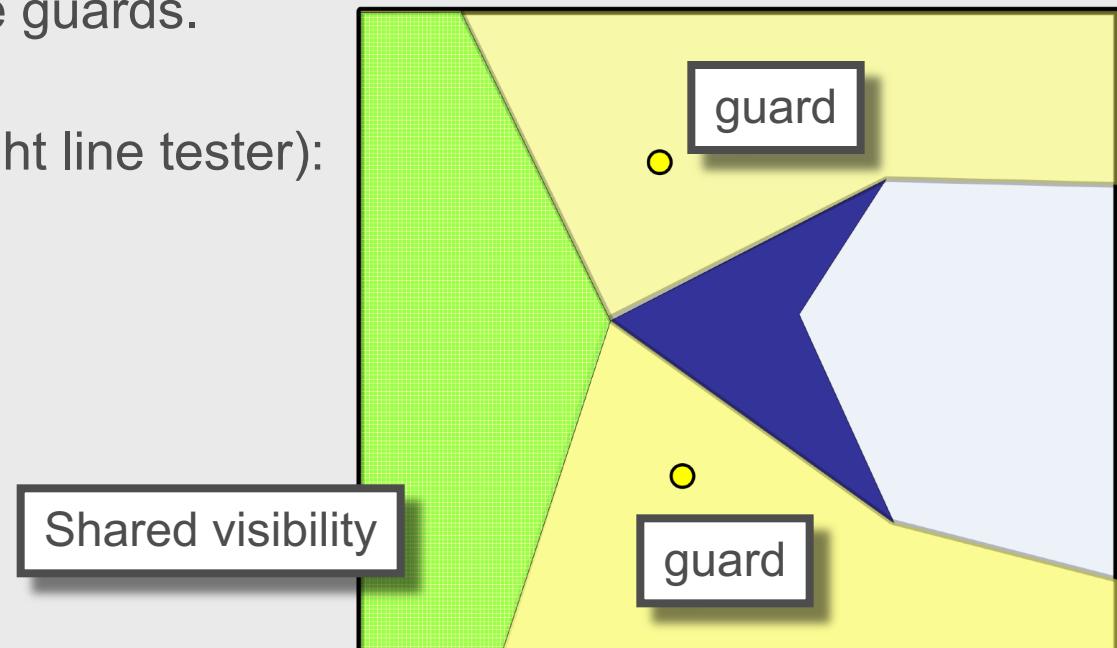
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

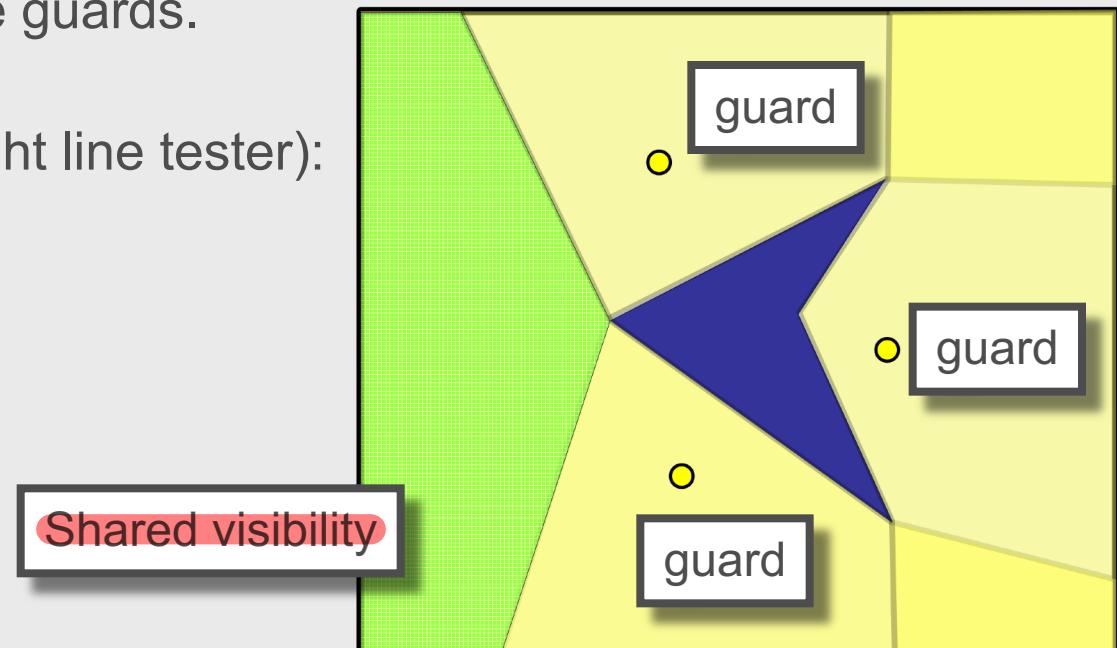
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

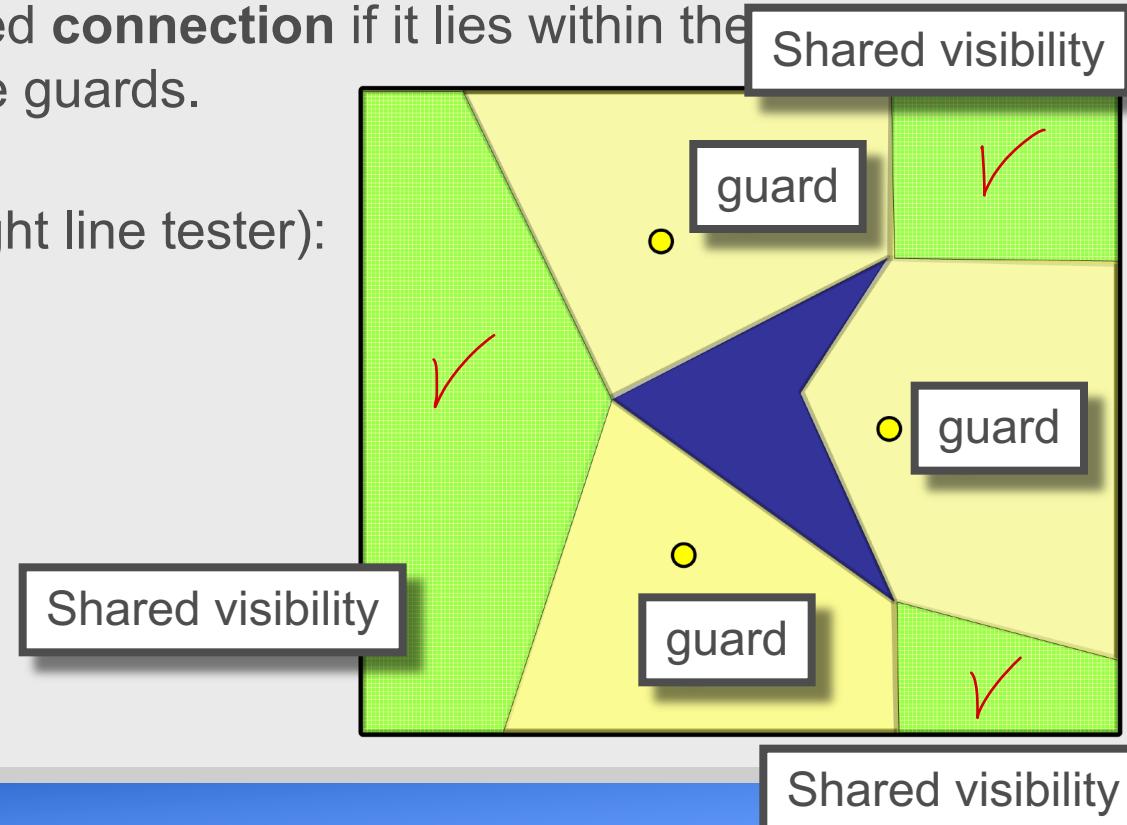
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

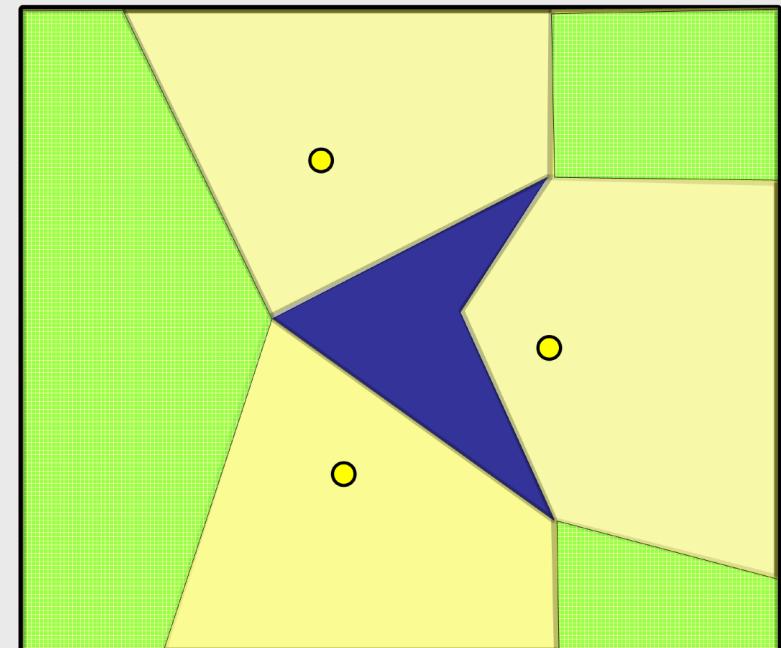
- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):



Some Terms

► Visibility domain, Guard and Connection:

- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$.
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):

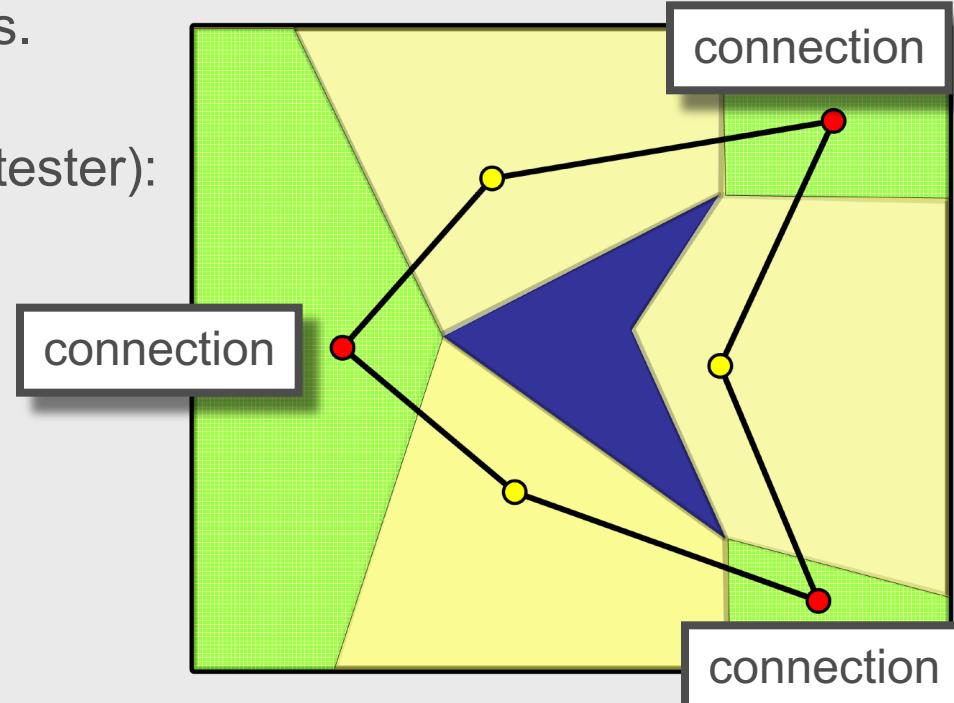


Some Terms

► Visibility domain, Guard and Connection:

- ▶ Let $L(q, q')$ be **any local method** that computes a path between two configurations q and q' .
- ▶ Visibility domain: $\text{Vis}_L(q) = \{ \text{all } q' \text{ in } CS_{\text{free}} \text{ such that } L(q, q') \in CS_{\text{free}} \}$.
- ▶ q is then called **guard** of $\text{Vis}_L(q)$. *Guard ist ein Node, von dem aus ein Viewfield erzeugt wird.*
- ▶ a configuration is called **connection** if it lies within the visibility domain of two or more guards.
- ▶ Example:
(local method = straight line tester):

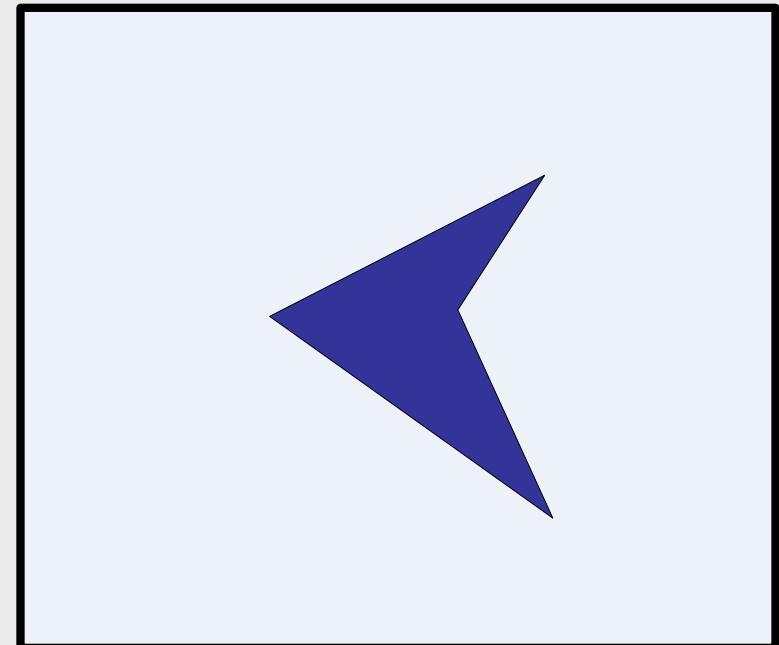
3 nodes are enough to create a complete path



Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

```
Guard = Ø; Connection = Ø; ntry = 0
while (ntry < M)
    Select a random free configuration q
    gvis = Ø; Gvis = Ø
    for all components Gi of Guard do
        found = FALSE
        for all nodes g of Gi do
            if ( q belongs to Vis(g) ) then
                found = TRUE
                if ( !gvis ) then
                    gvis = g; Gvis = Gi
                else /* q is a connection node */
                    add q to Connection
                    Create edges (q,g) and (q, gvis)
                    Merge components Gvis and Gi
            until found = TRUE
            if ( !gvis ) then /* q is a guard node */
                add {q} to Guard; ntry = 0
            else ntry = ntry +1
    end
```



Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node ***/**

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

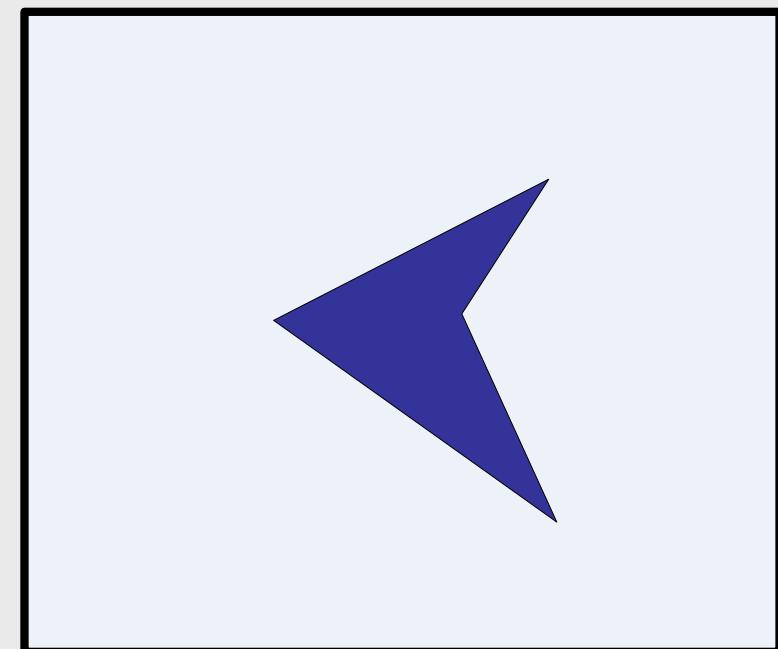
if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0



Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

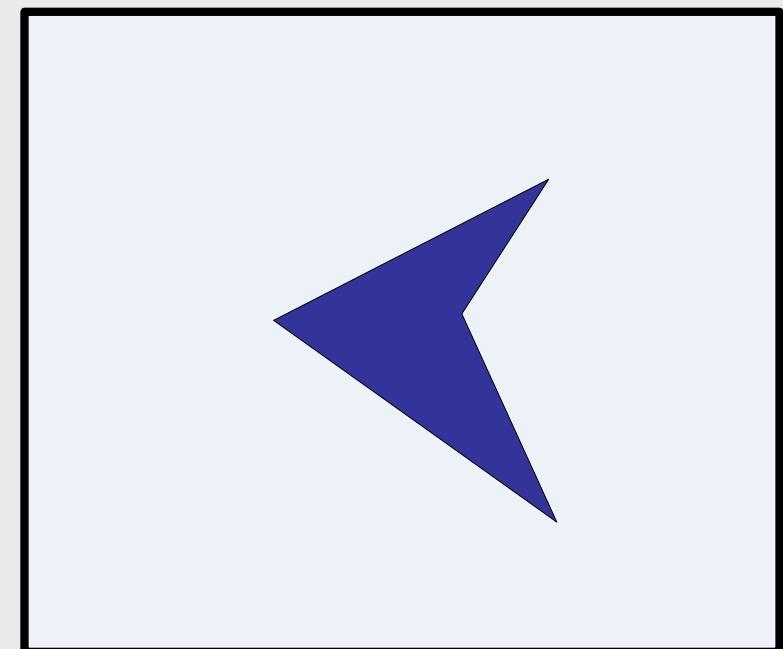
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0



Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

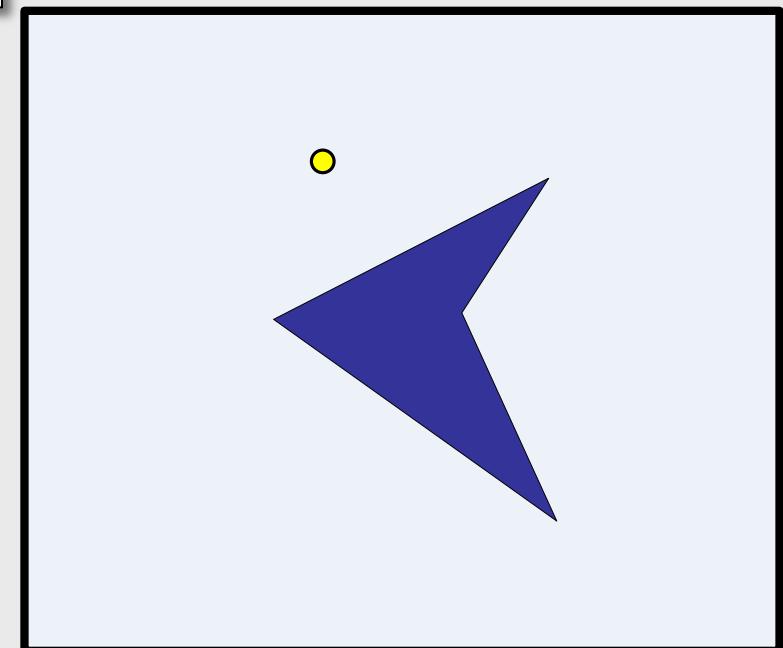
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0



Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node ***/**

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

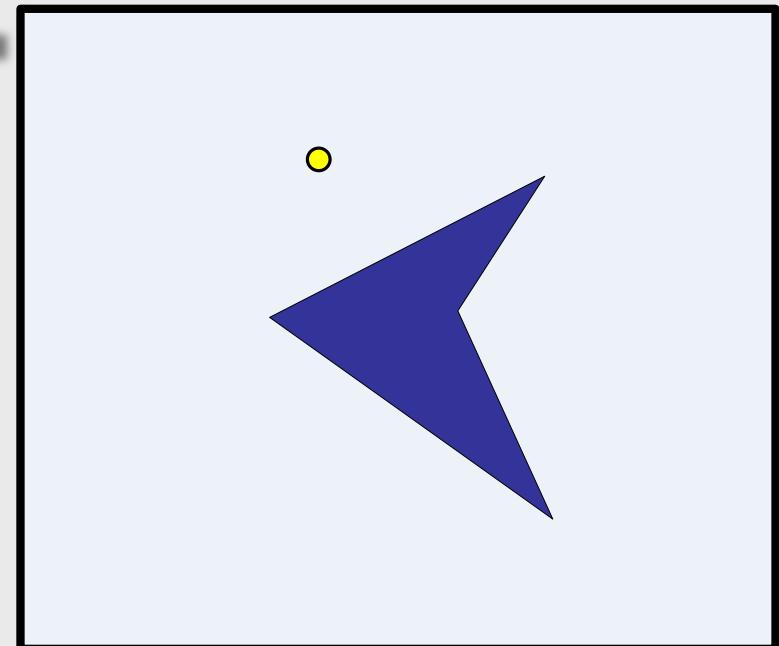
if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

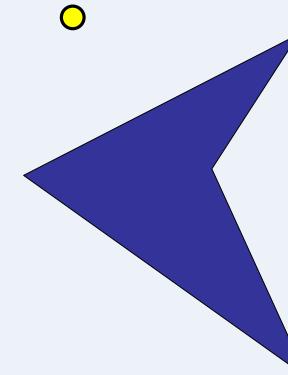
 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0

Guard = \emptyset



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node ***/**

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

$g_{vis} = \emptyset$

else /* q is a guard node ***/**

 found = TRUE

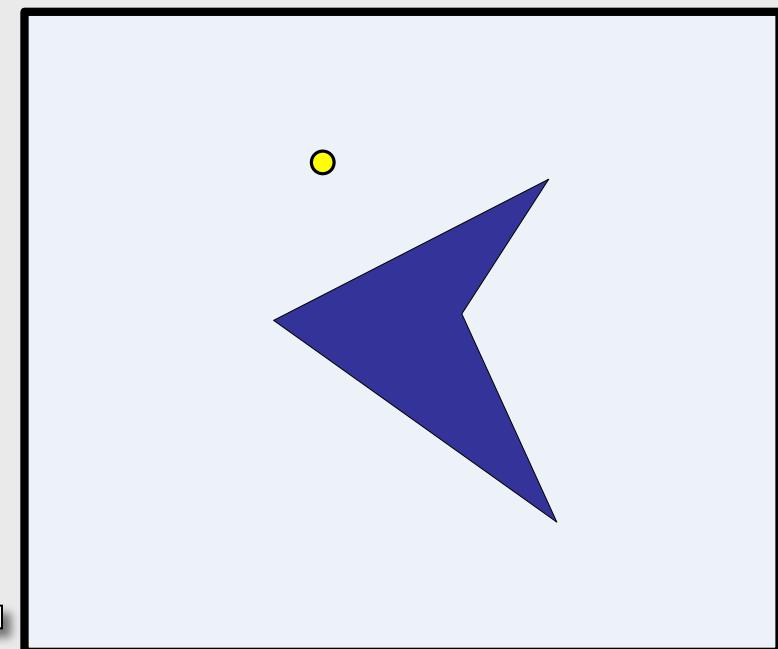
if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = \emptyset
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

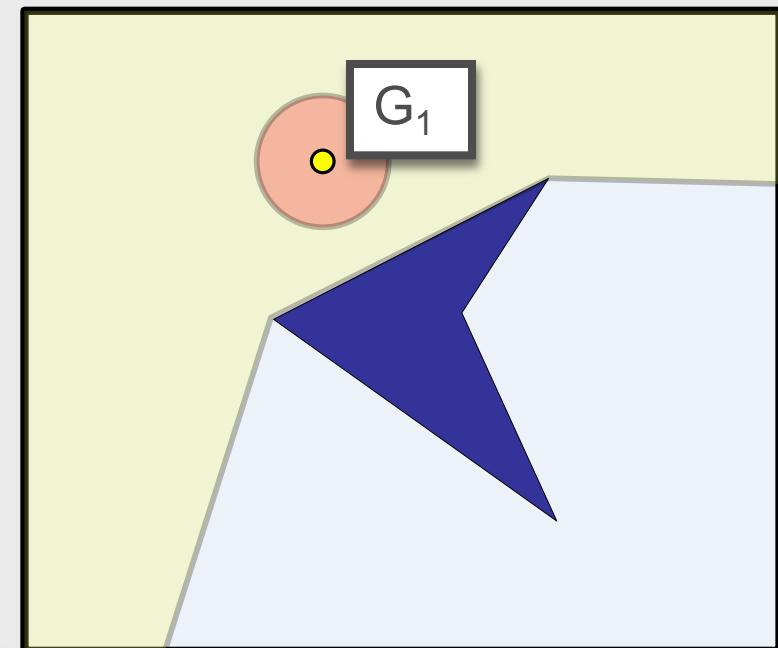
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

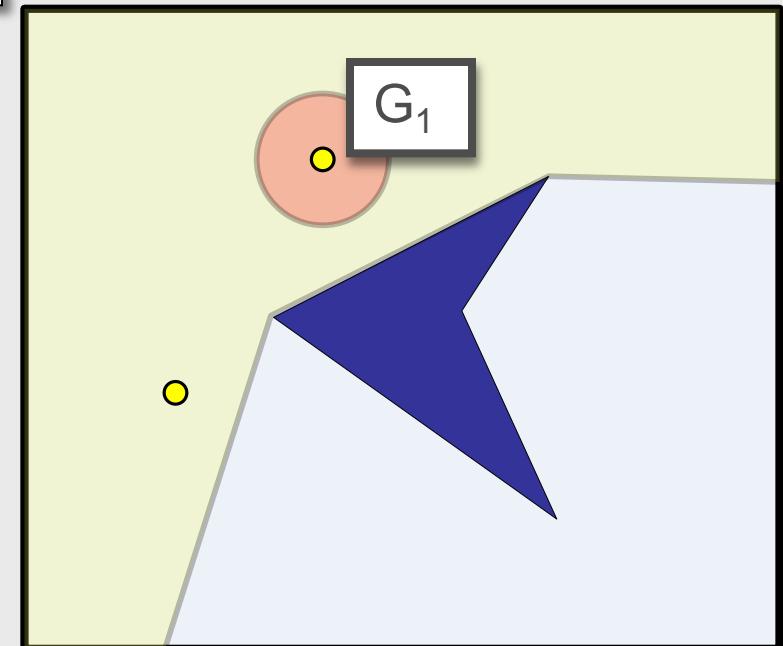
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

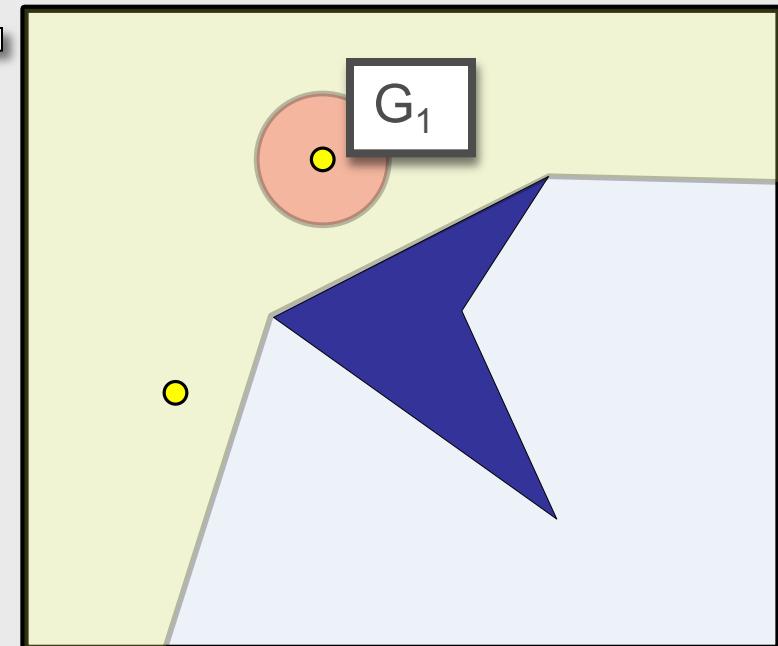
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

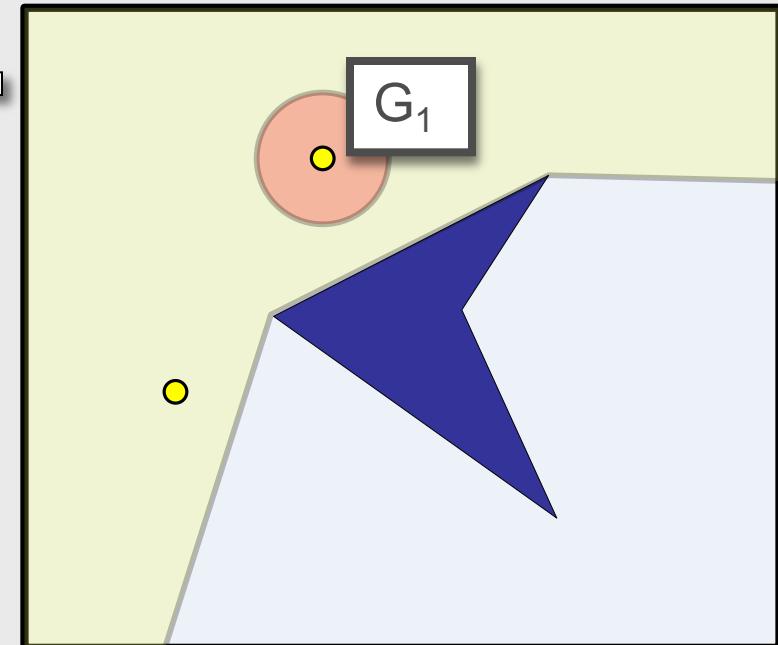
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

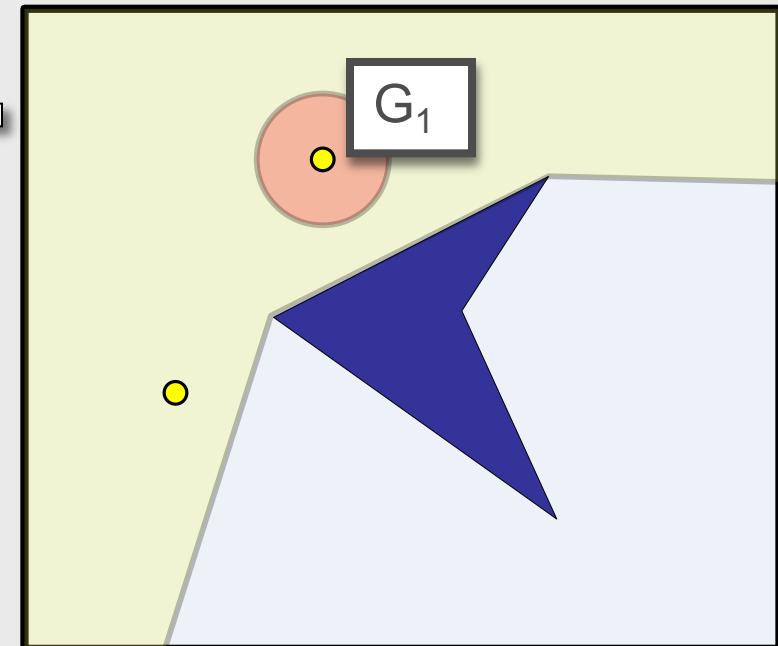
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

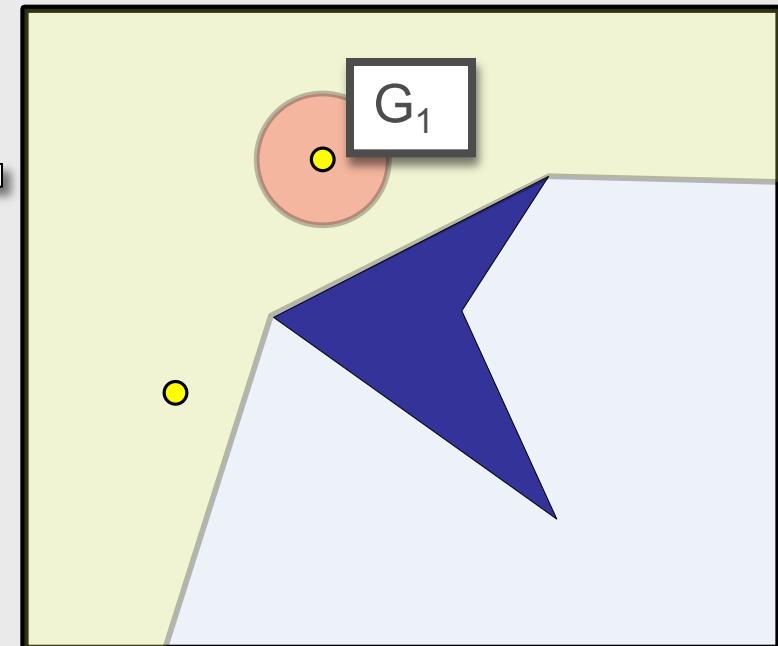
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

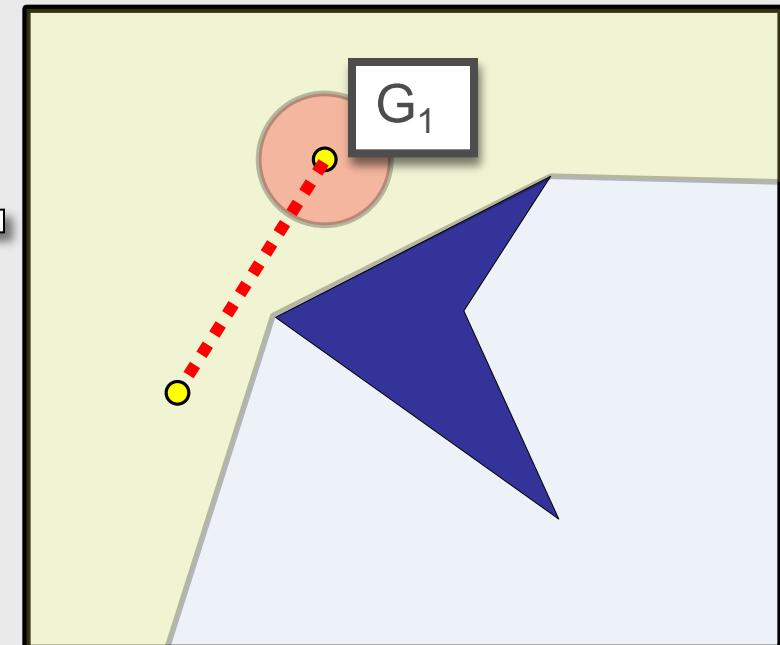
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

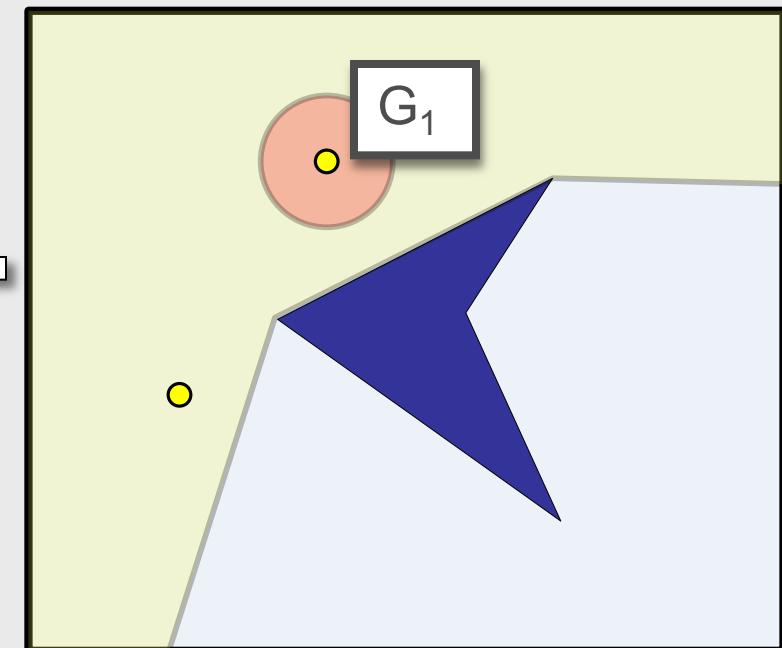
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

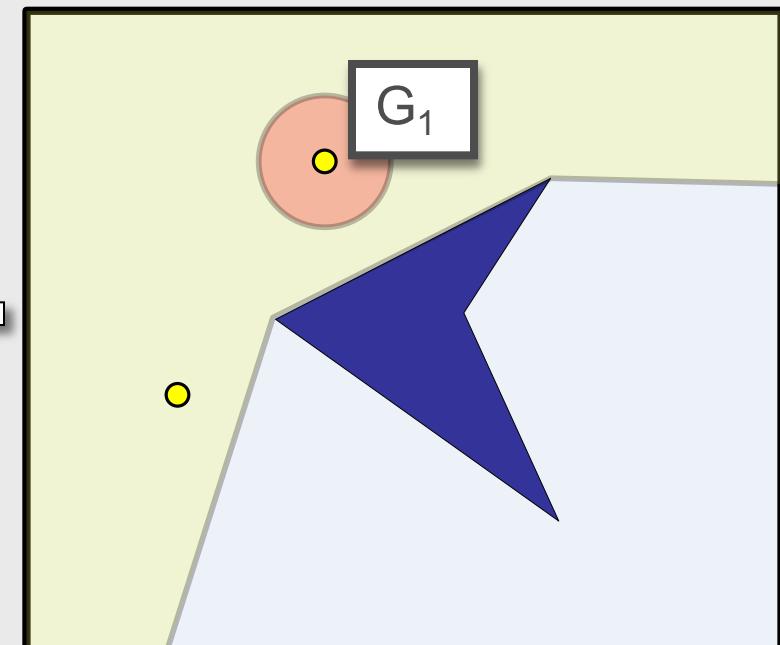
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

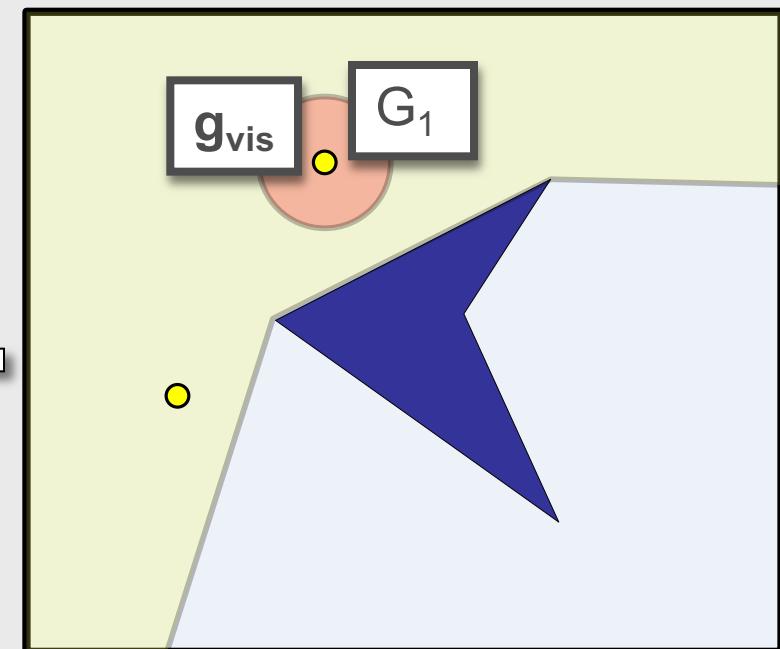
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = g$
 $G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

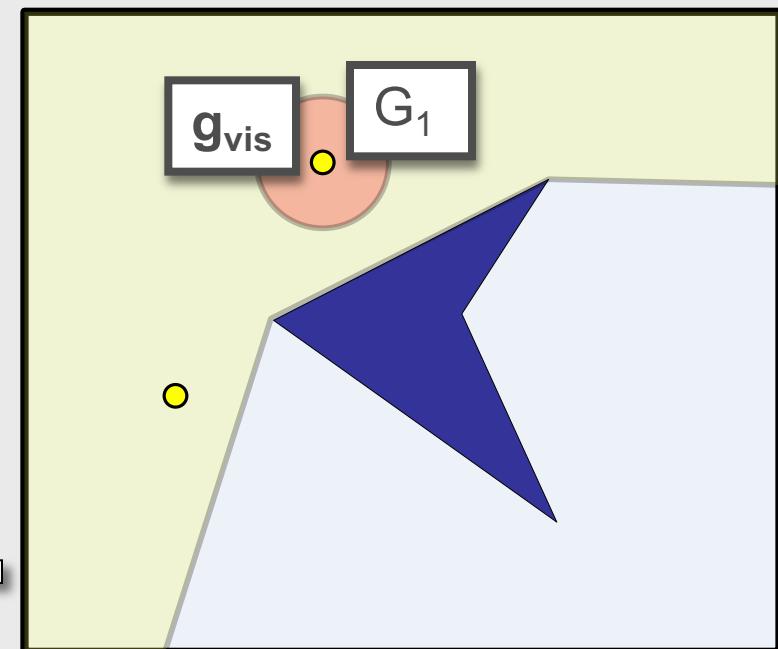
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = g$
 $G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

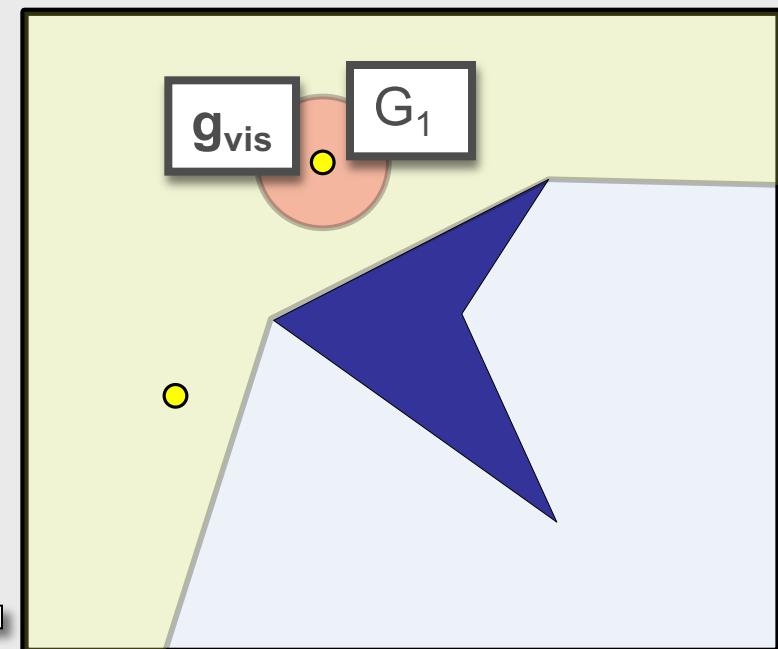
if (! g_{vis}) **then /* q is a guard node */** ←

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 0



$g_{vis} = g$
 $G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

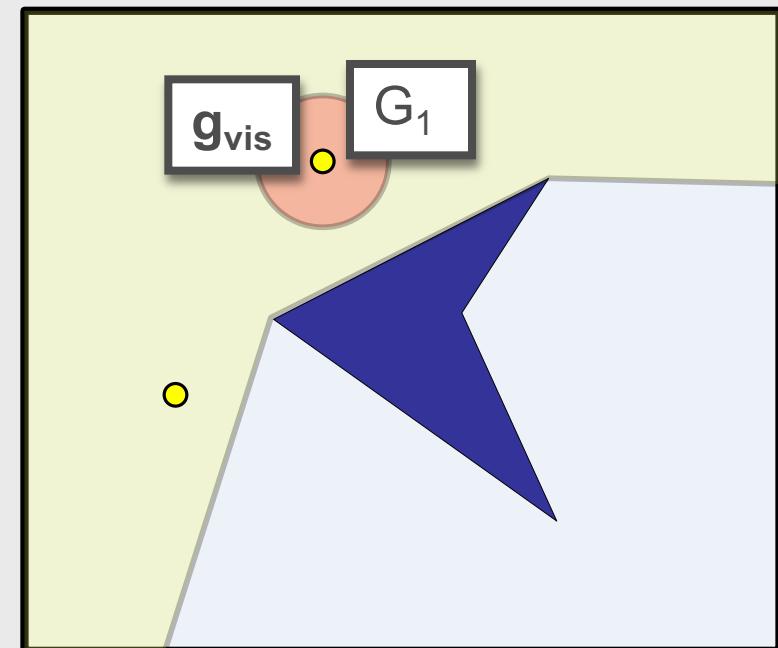
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = g$
 $G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

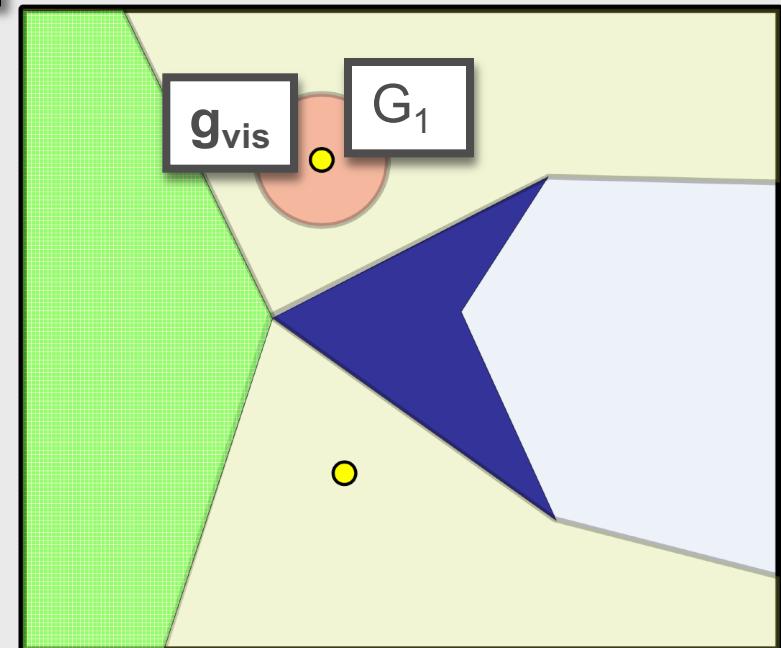
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = g$
 $G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

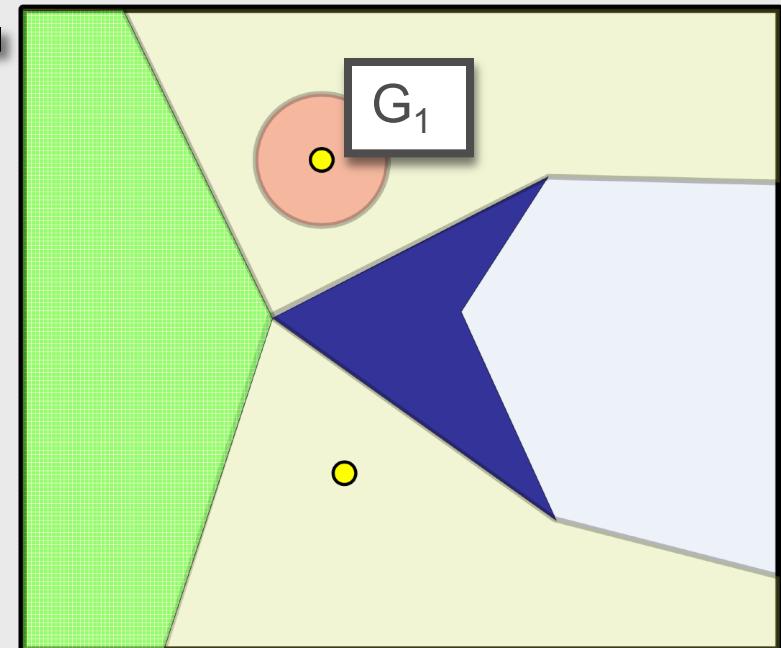
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

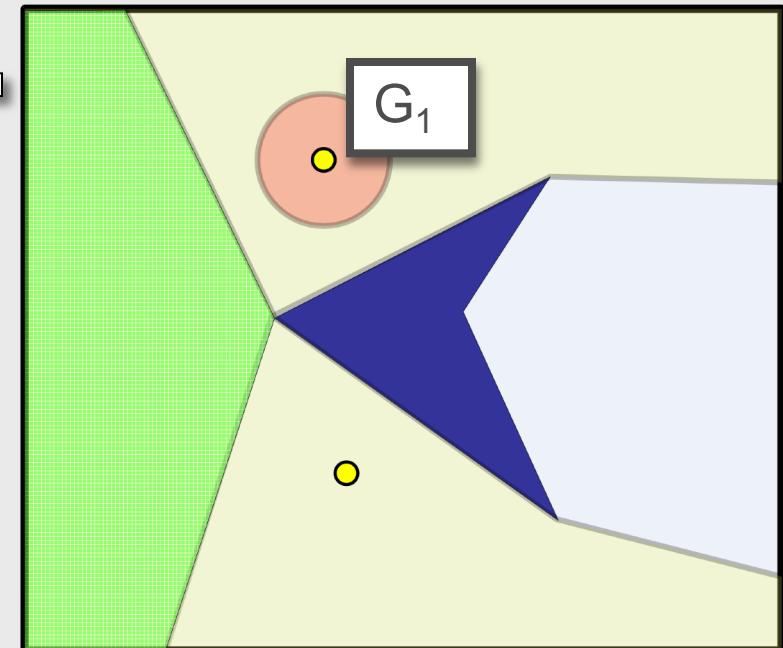
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

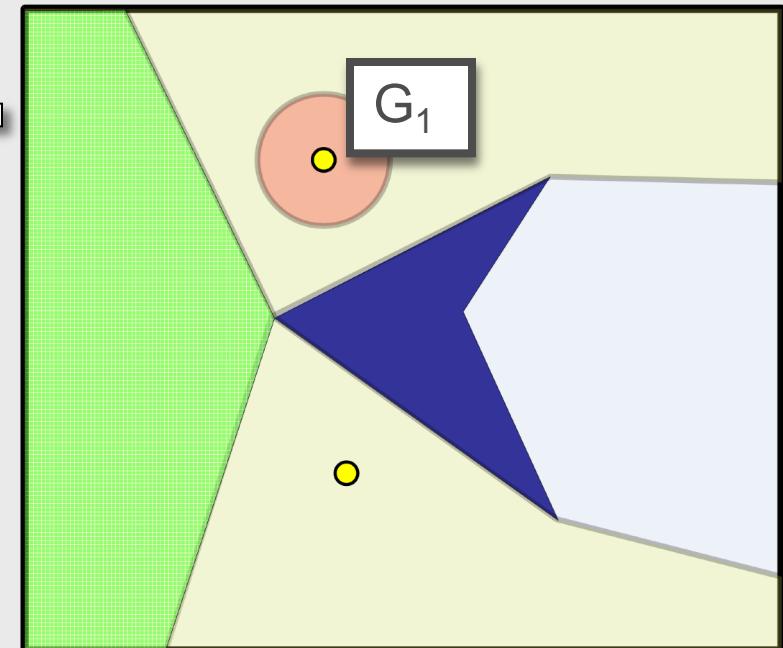
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node ***/**

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

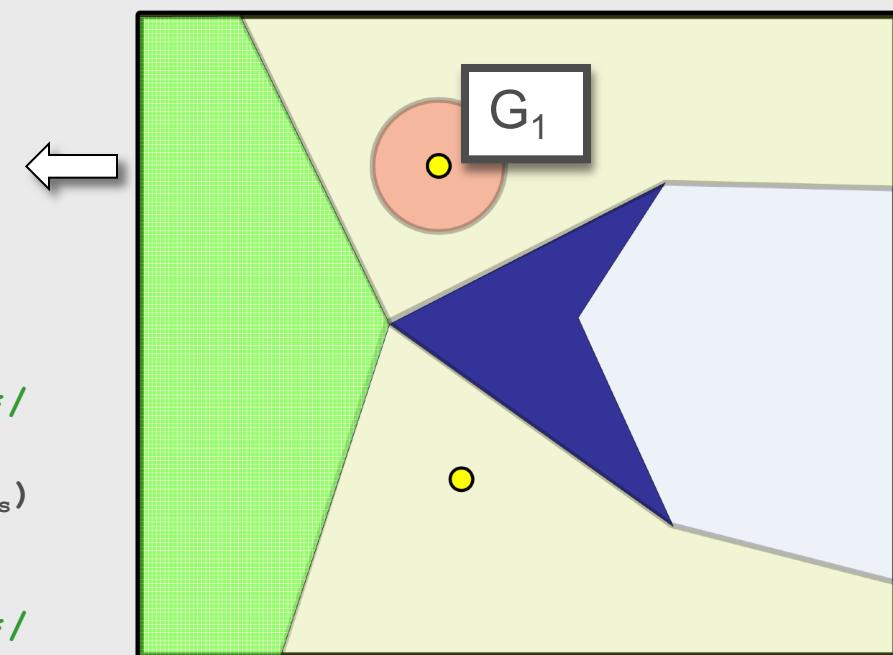
if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

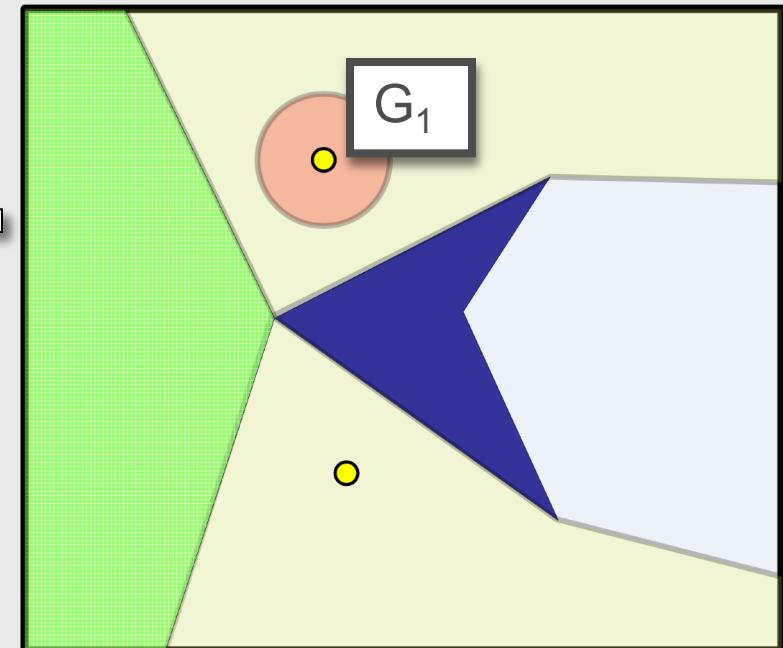
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

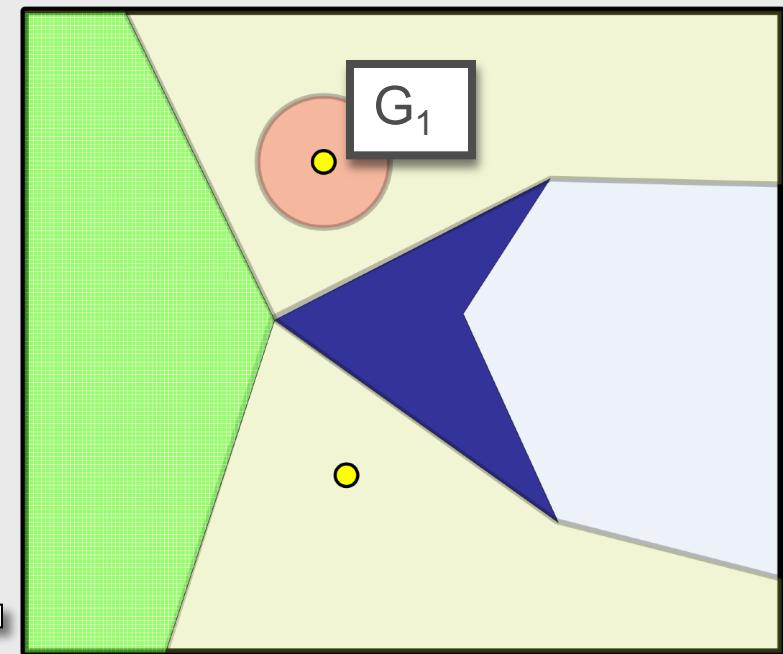
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1\}$
Connection = \emptyset
ntry = 1



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

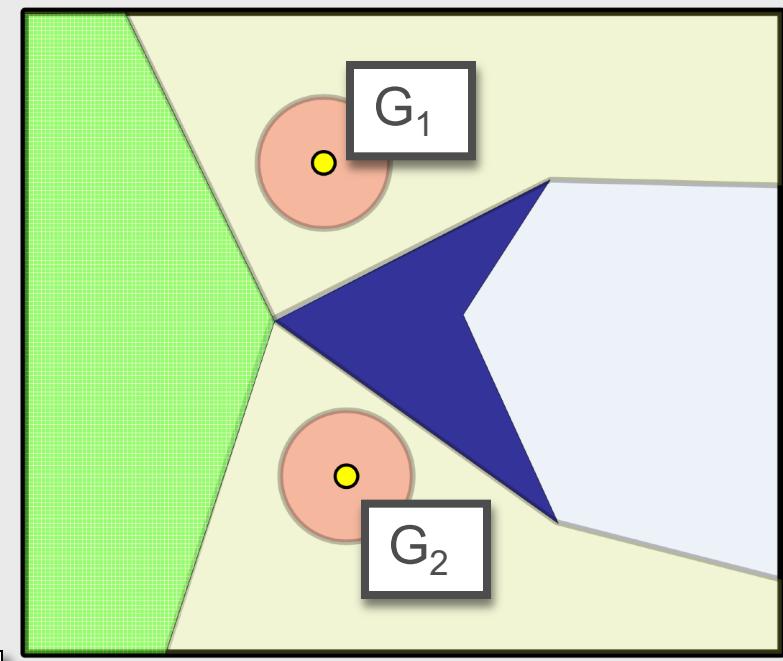
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

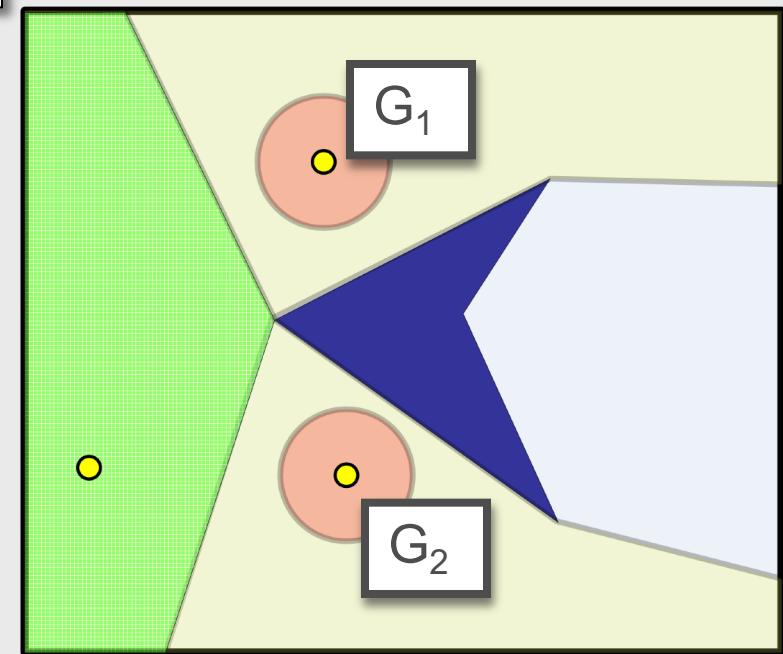
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

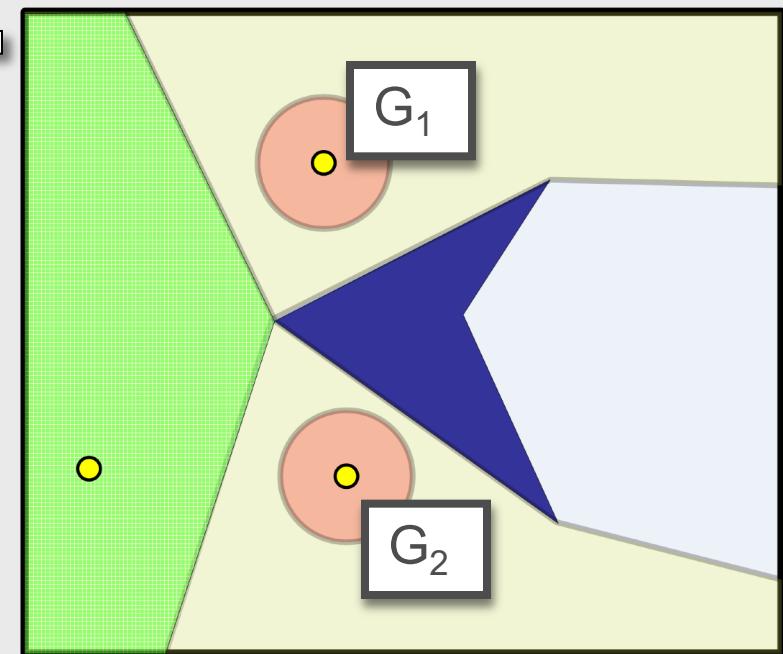
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

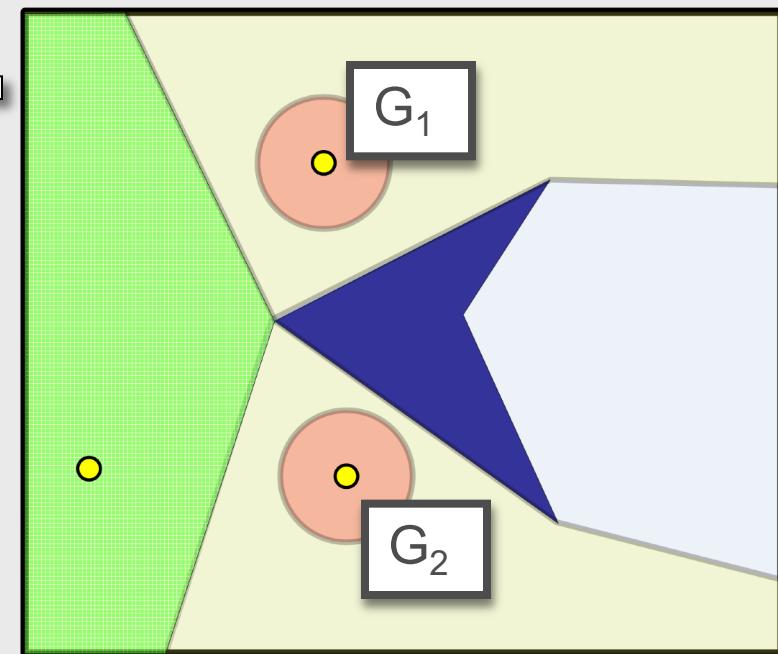
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

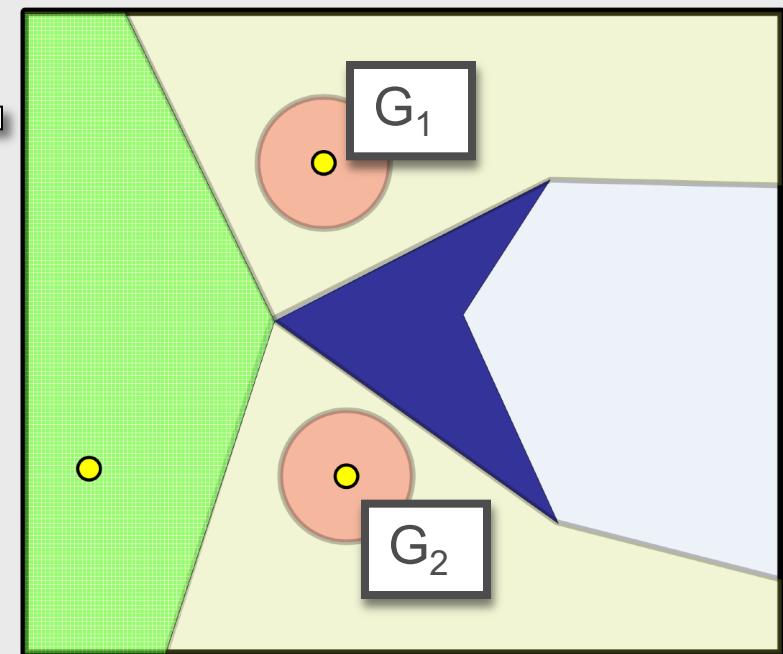
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

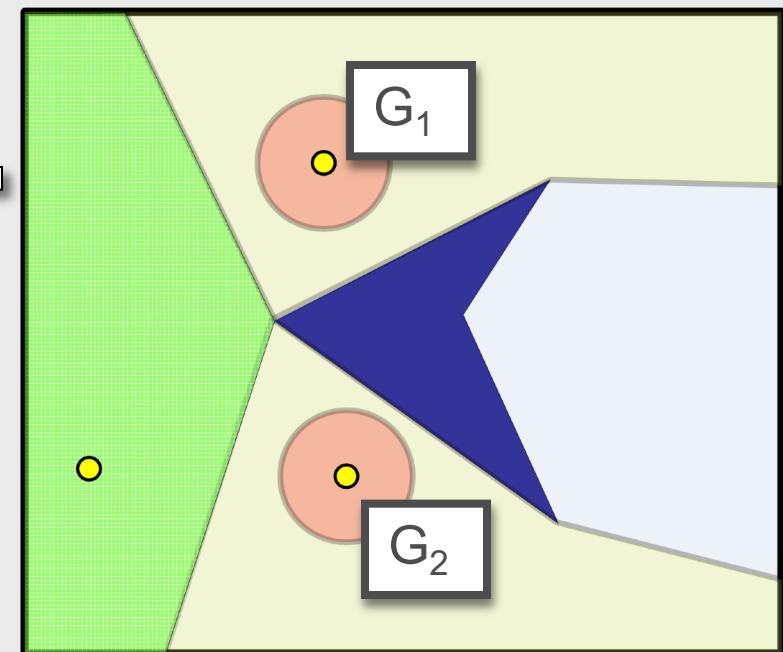
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

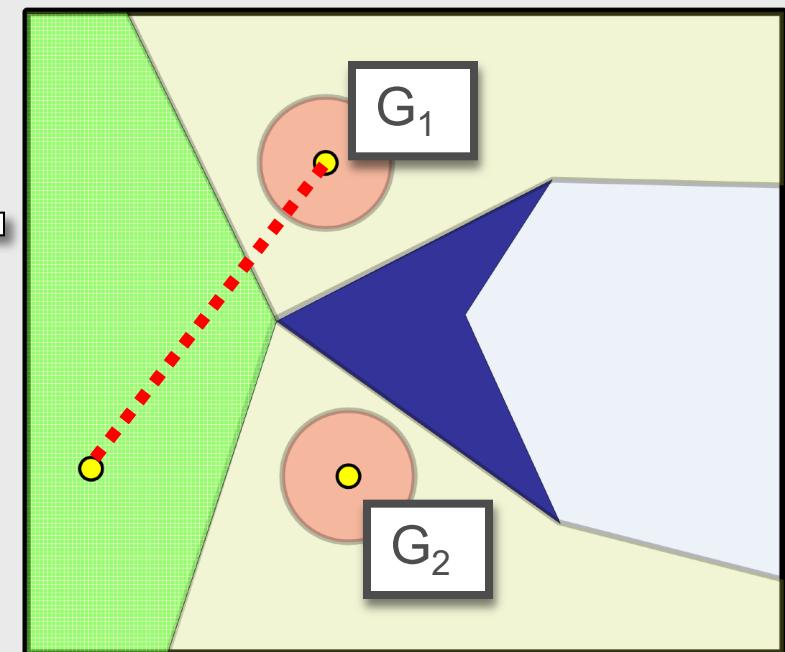
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

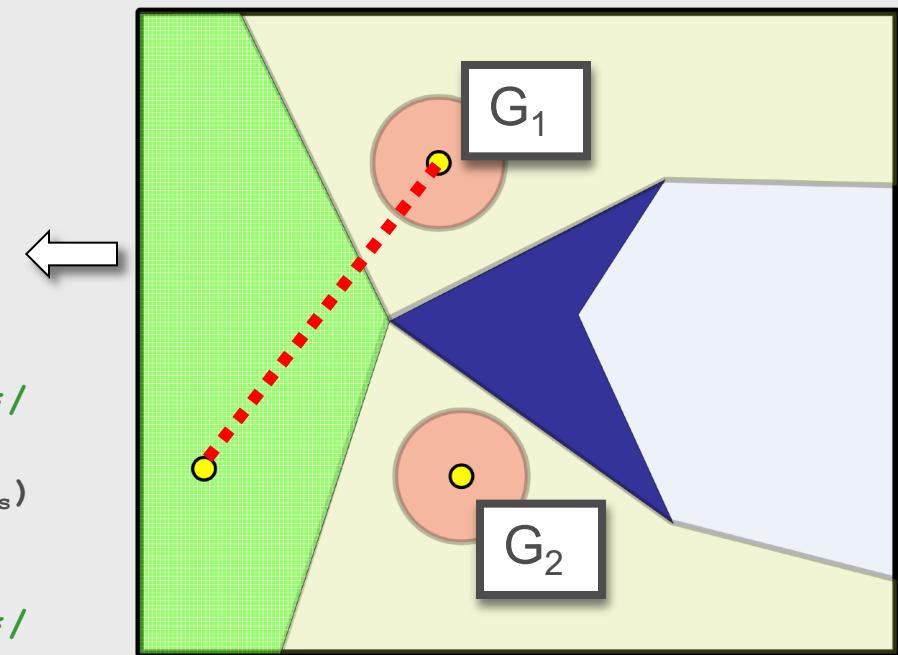
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

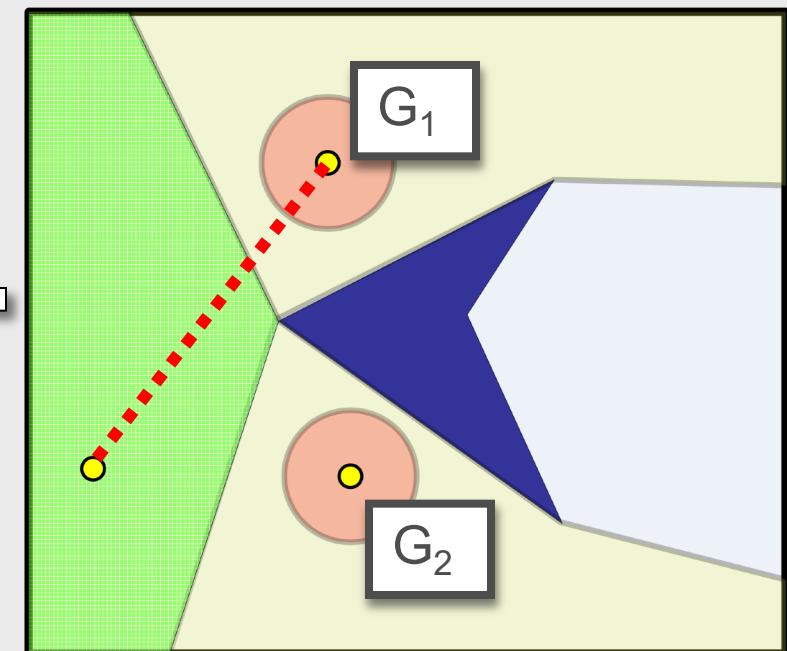
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

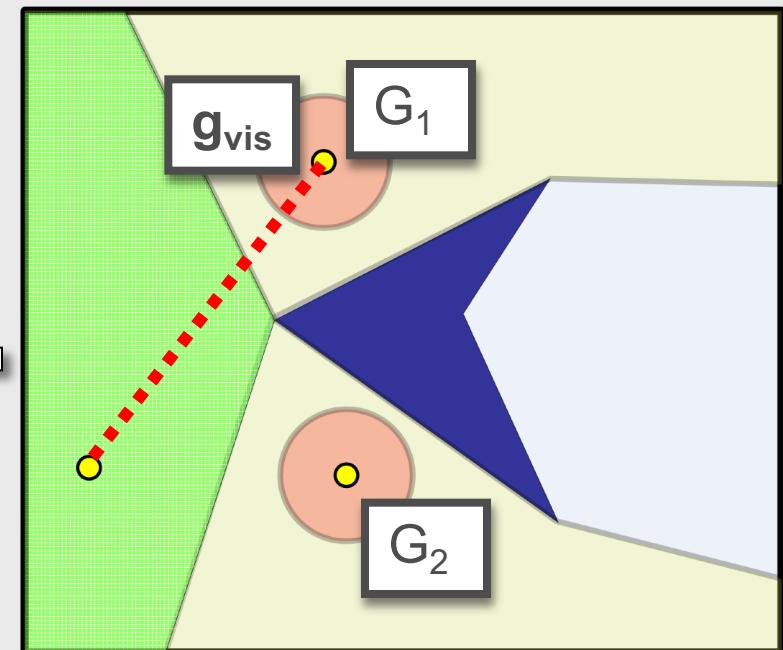
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

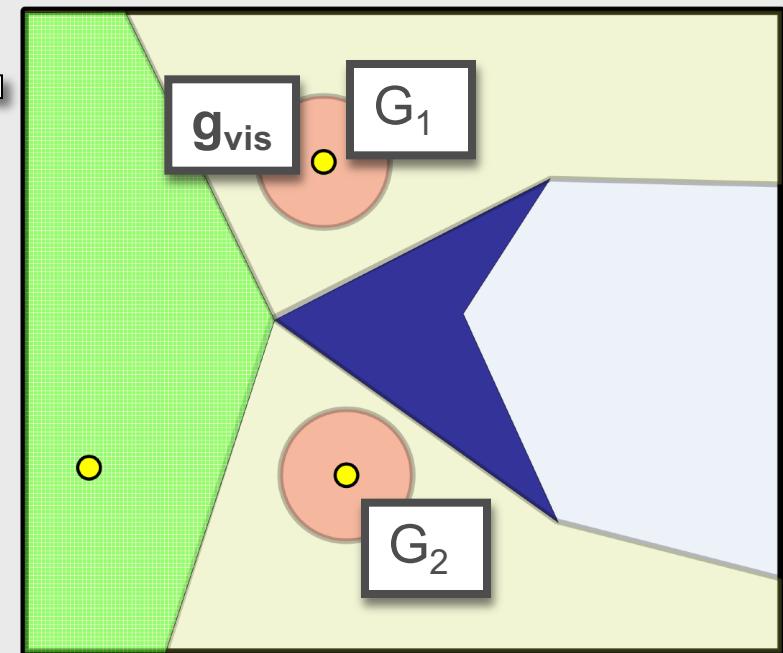
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

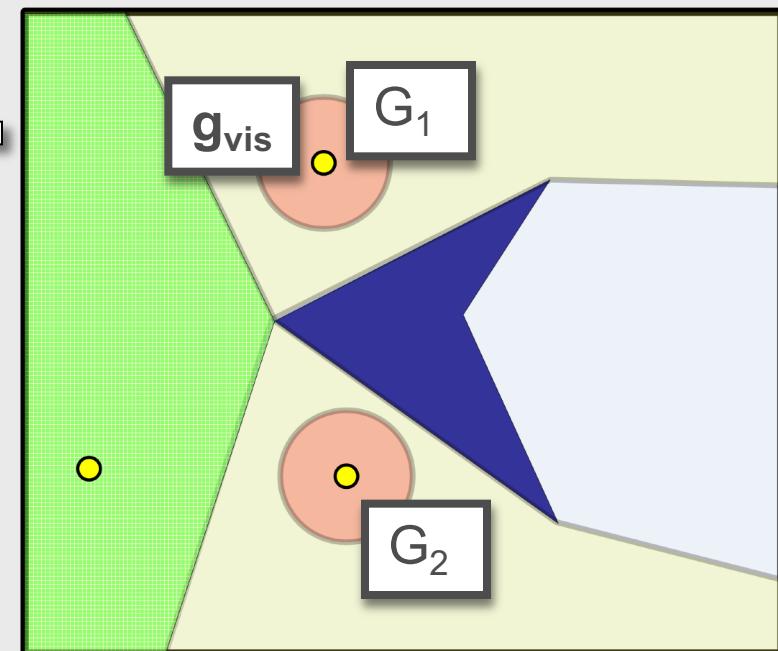
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

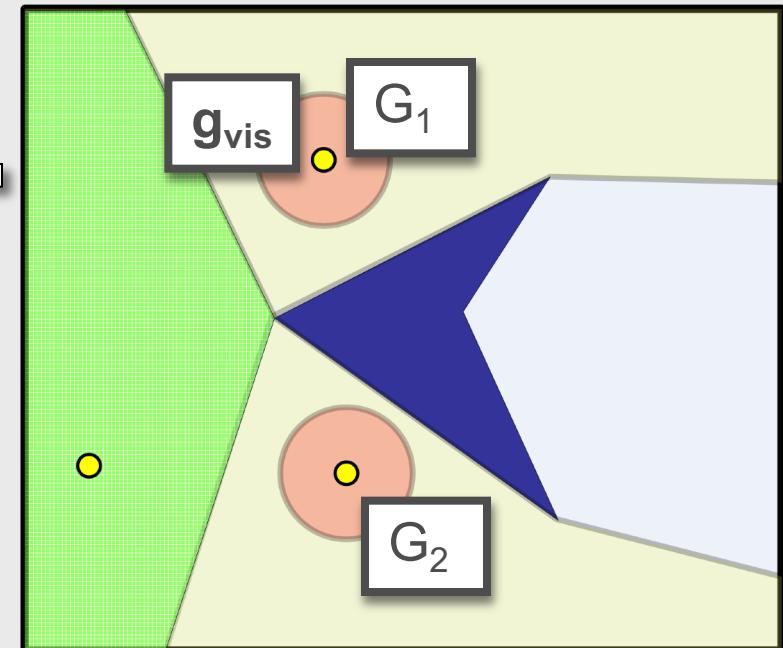
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

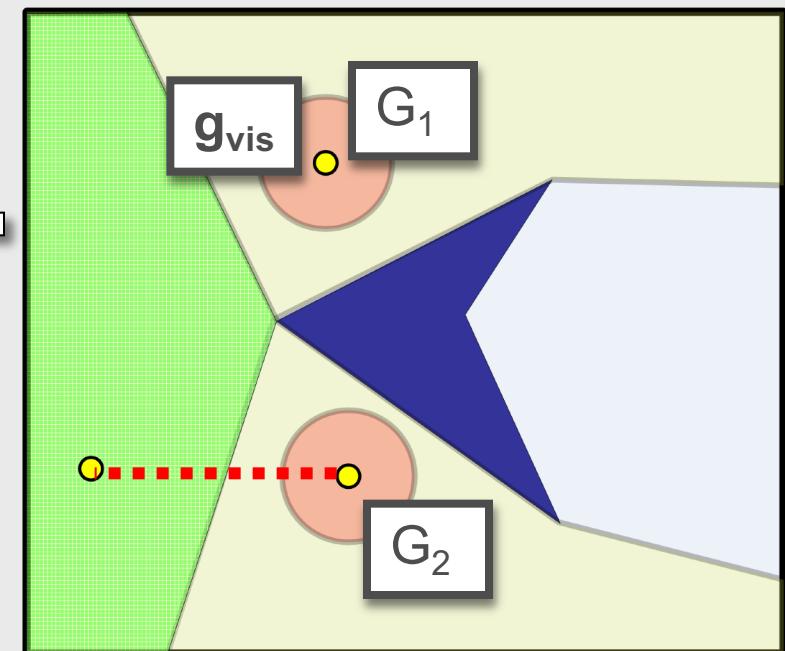
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

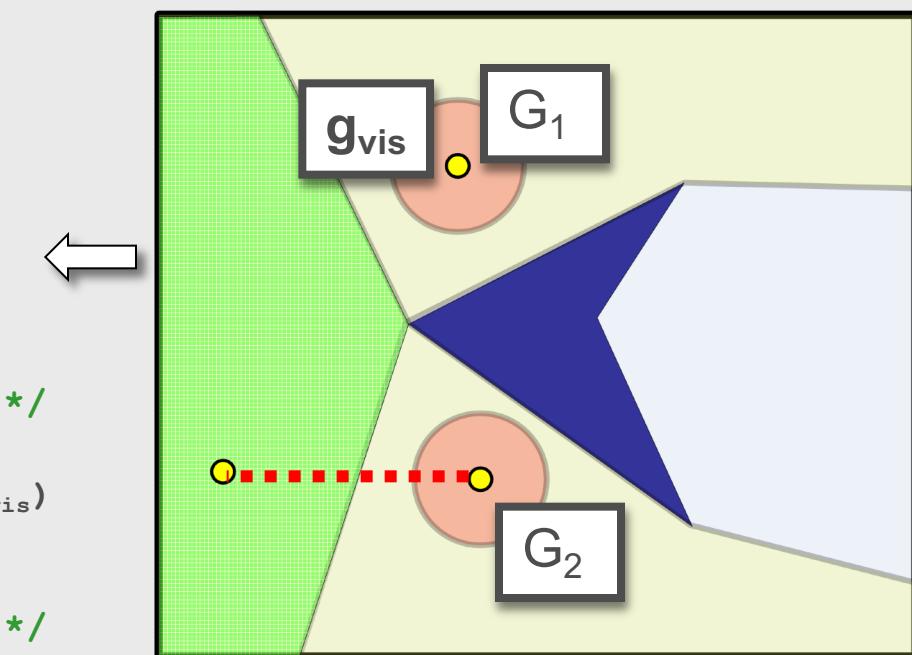
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

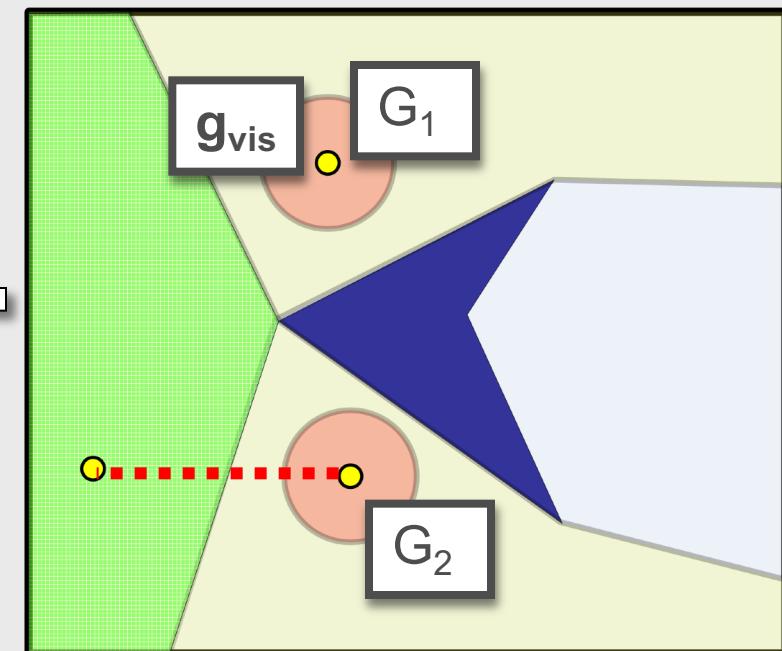
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

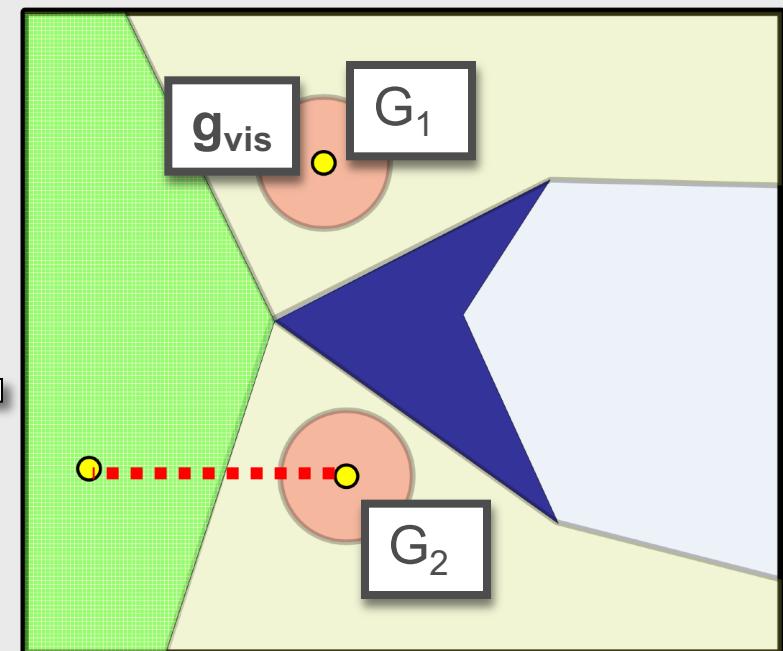
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = \emptyset

ntry = 0



$g_{vis} = g$

$G_{vis} = G_1$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

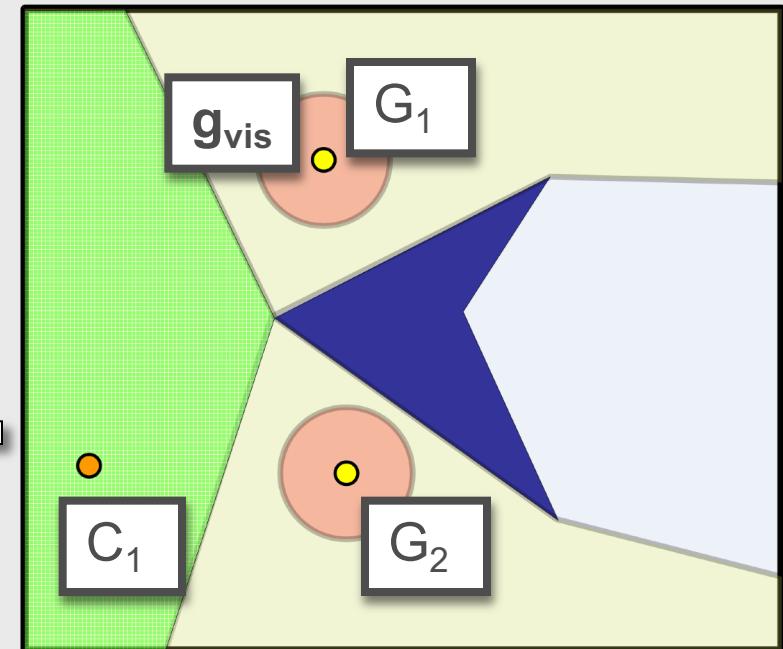
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1, G2}  
Connection = {C1}  
ntry = 0
```



```
 $g_{vis} = g$   
 $G_{vis} = G_1$ 
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

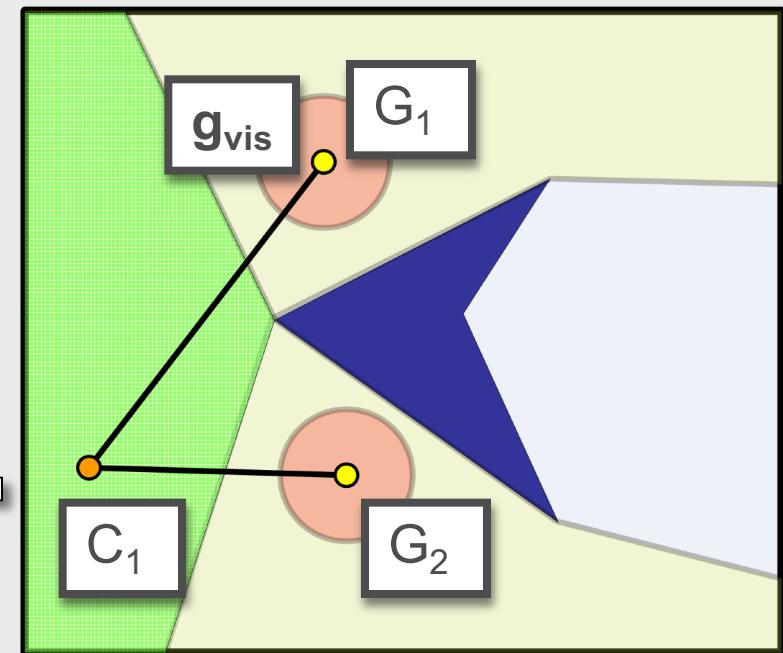
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1, G2}  
Connection = {C1}  
ntry = 0
```



```
gvis = g  
Gvis = G1
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

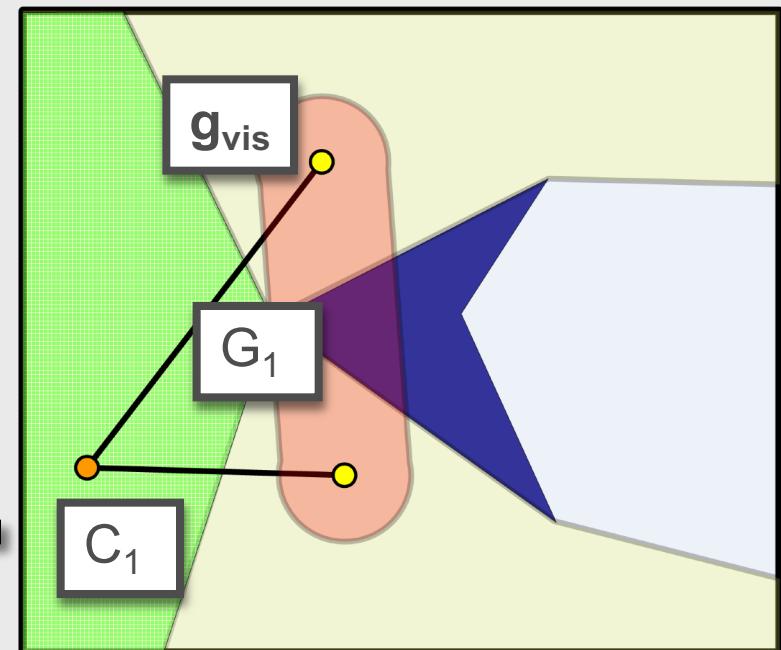
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = g
Gvis = G1
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to $Vis(g)$) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

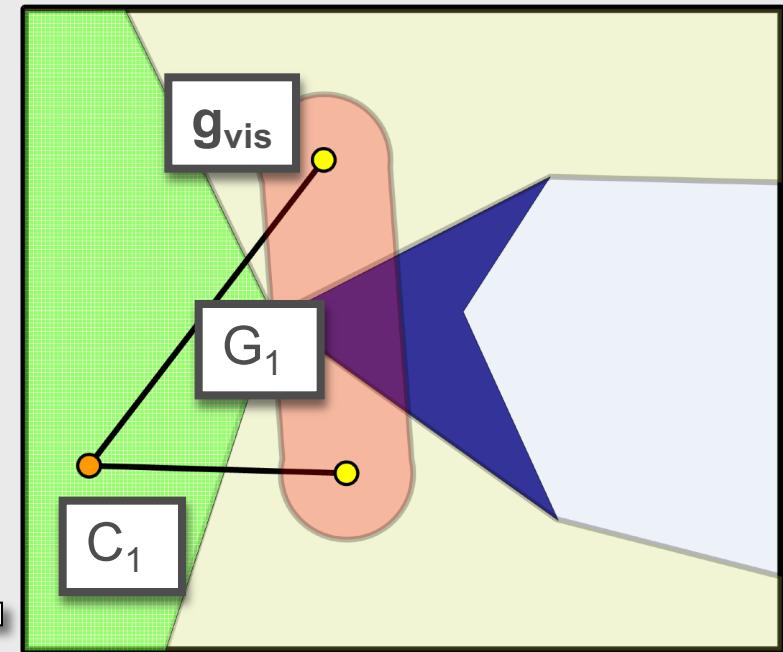
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = g
Gvis = G1
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

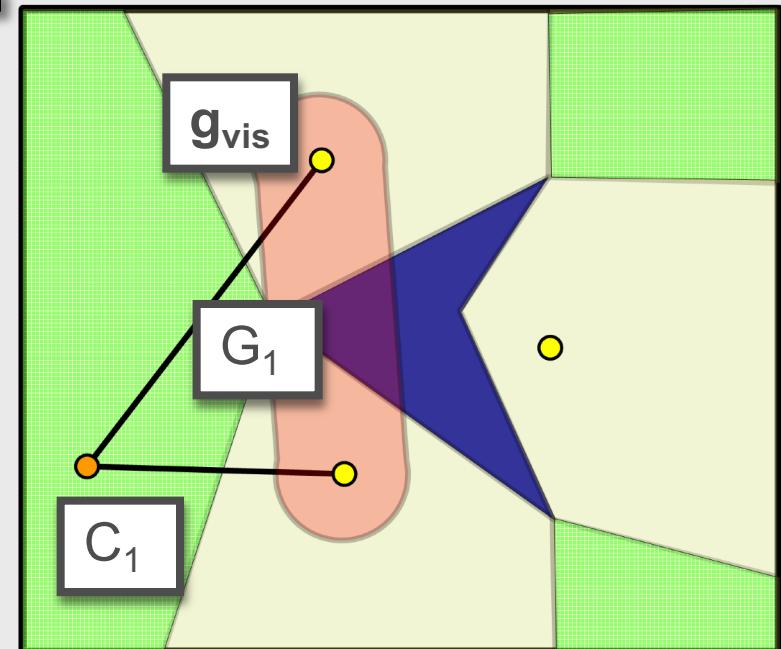
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = g
Gvis = G1
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

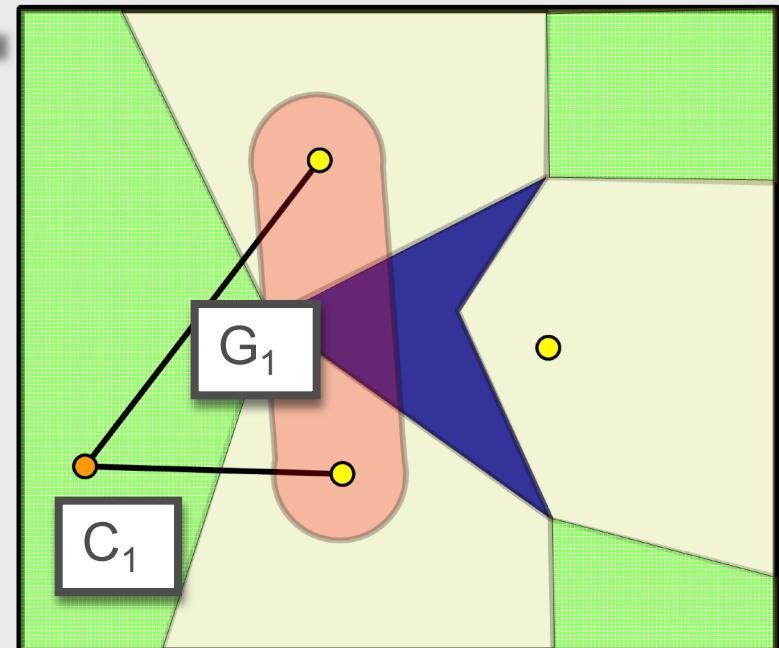
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = { G_1 }
Connection = { C_1 }
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

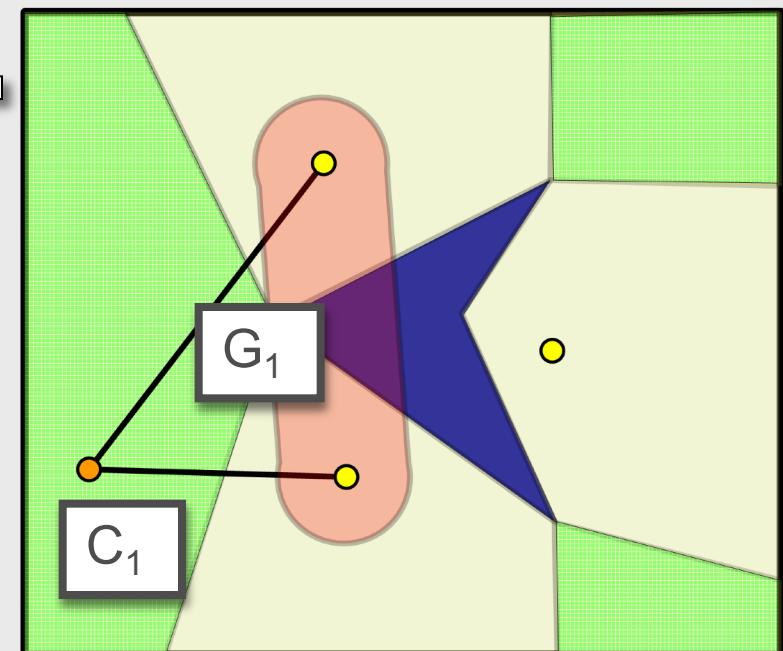
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = ∅
Gvis = ∅
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

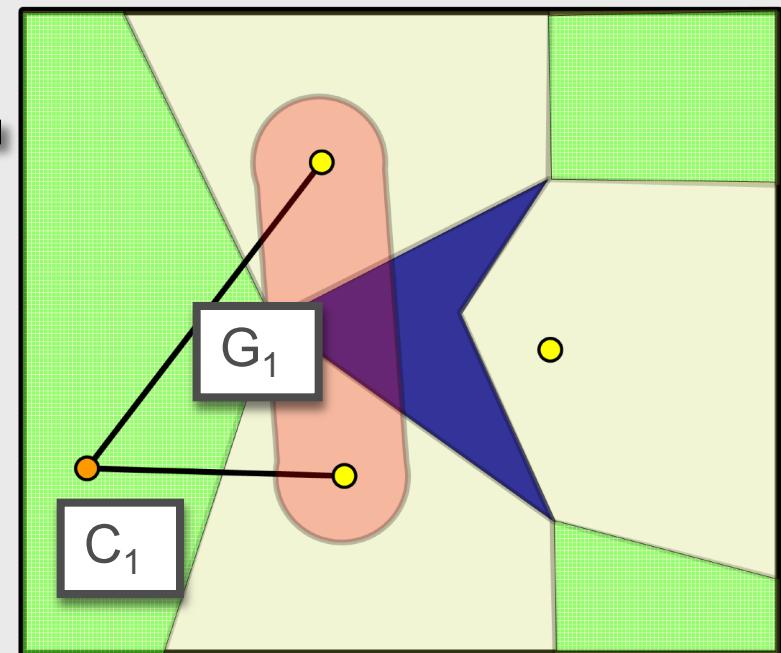
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = ∅
Gvis = ∅
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

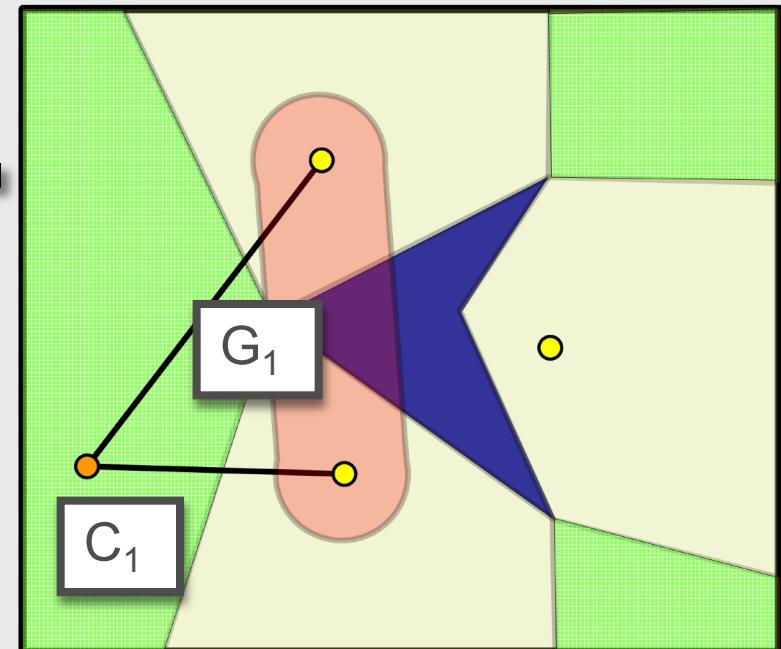
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = ∅
Gvis = ∅
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

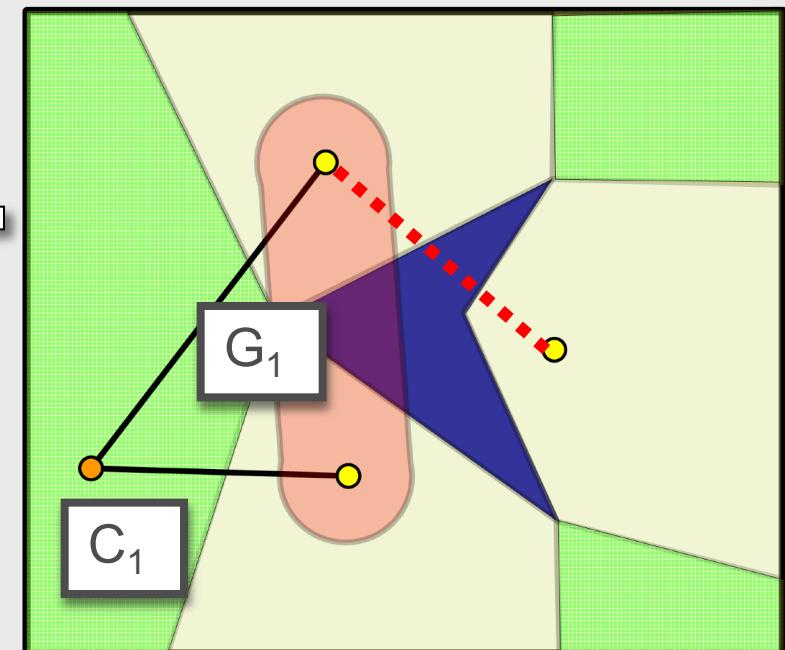
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
 $g_{vis} = \emptyset$ 
 $G_{vis} = \emptyset$ 
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

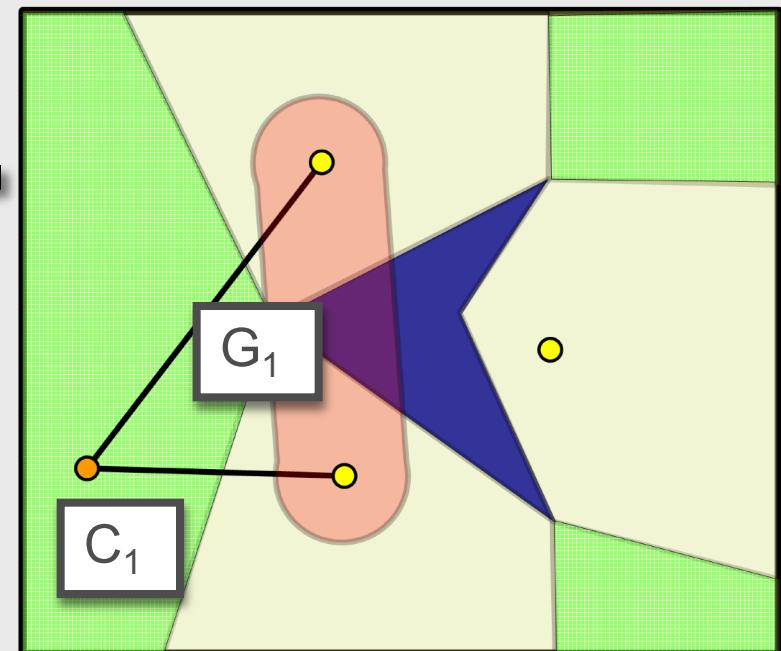
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
gvis = ∅
Gvis = ∅
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

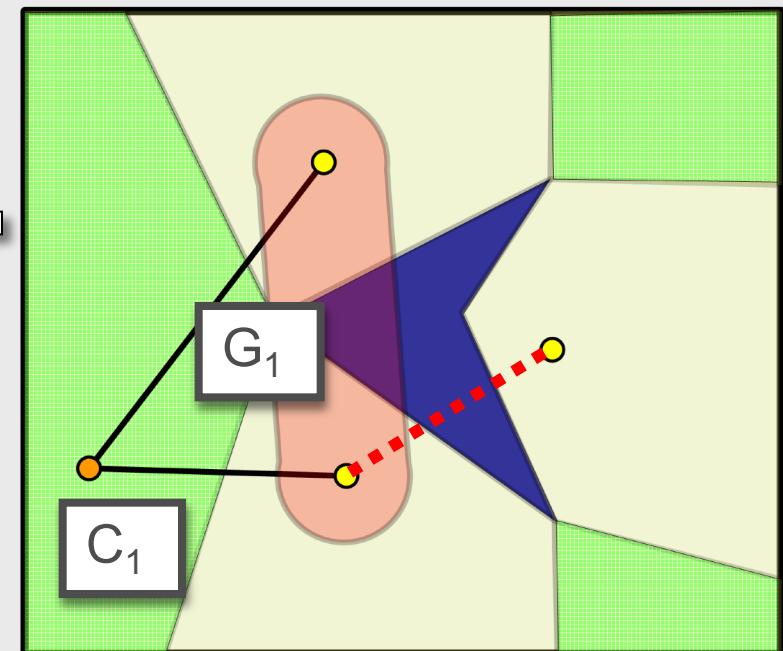
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
 $g_{vis} = \emptyset$ 
 $G_{vis} = \emptyset$ 
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

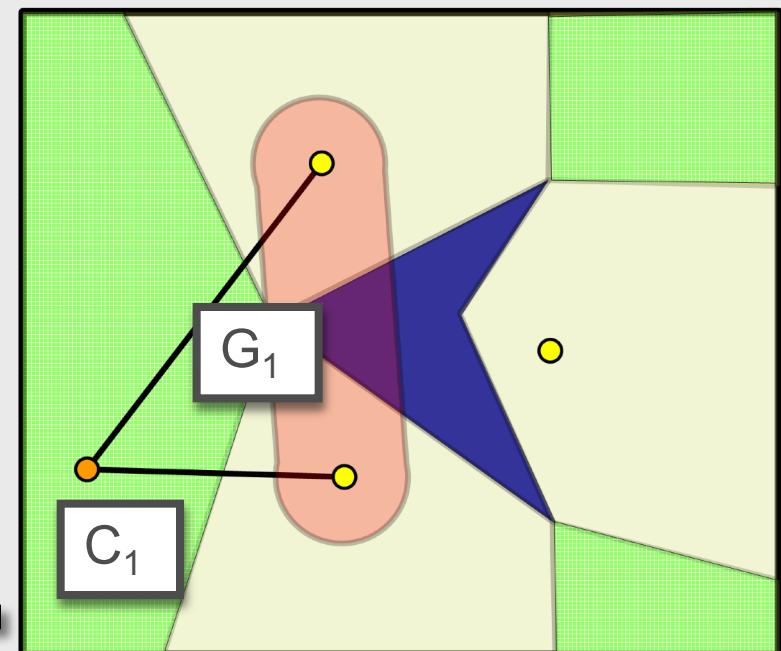
if (! g_{vis}) **then /* q is a guard node */** ←

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

```
Guard = {G1}
Connection = {C1}
ntry = 0
```



```
 $g_{vis} = \emptyset$ 
 $G_{vis} = \emptyset$ 
```

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

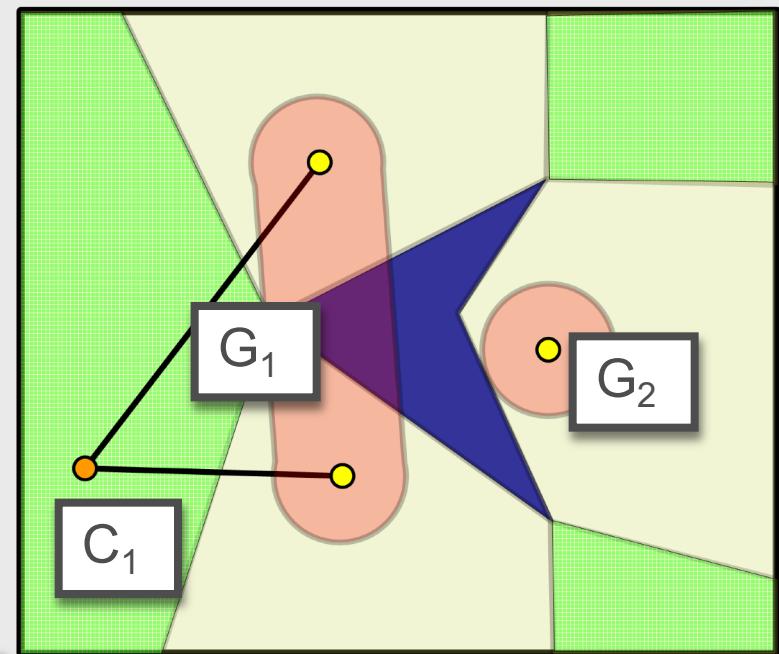
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$
Connection = $\{C_1\}$
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node */

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

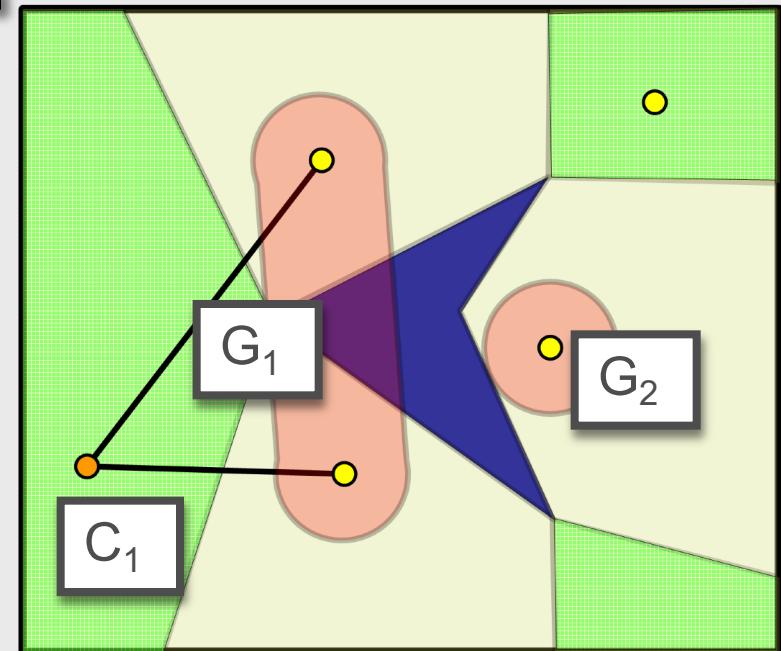
if (! g_{vis}) **then /* q is a guard node */**

 add {q} to Guard; ntry = 0

else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$
Connection = $\{C_1\}$
ntry = 0



$g_{vis} = \emptyset$
 $G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

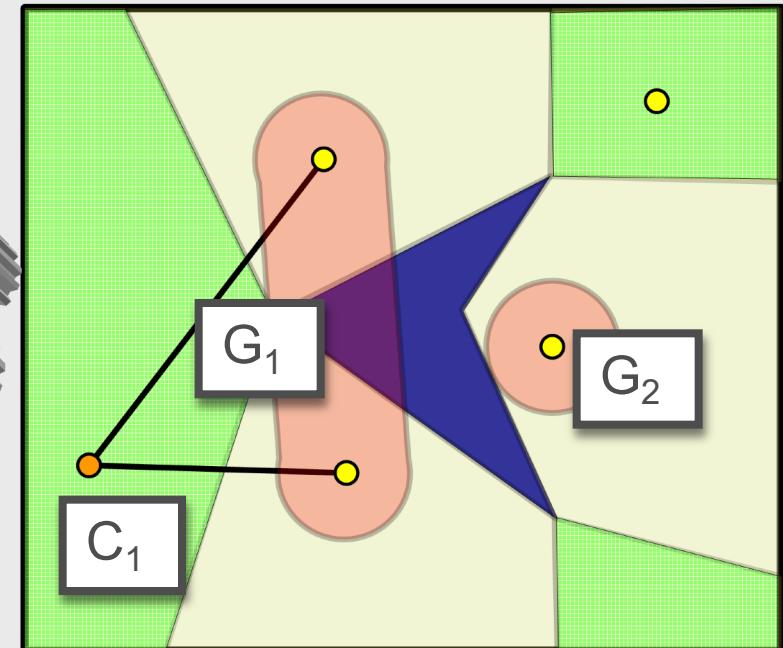
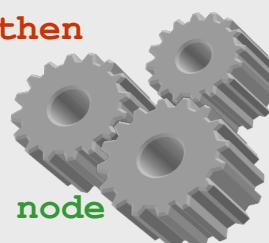
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = $\{C_1\}$

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

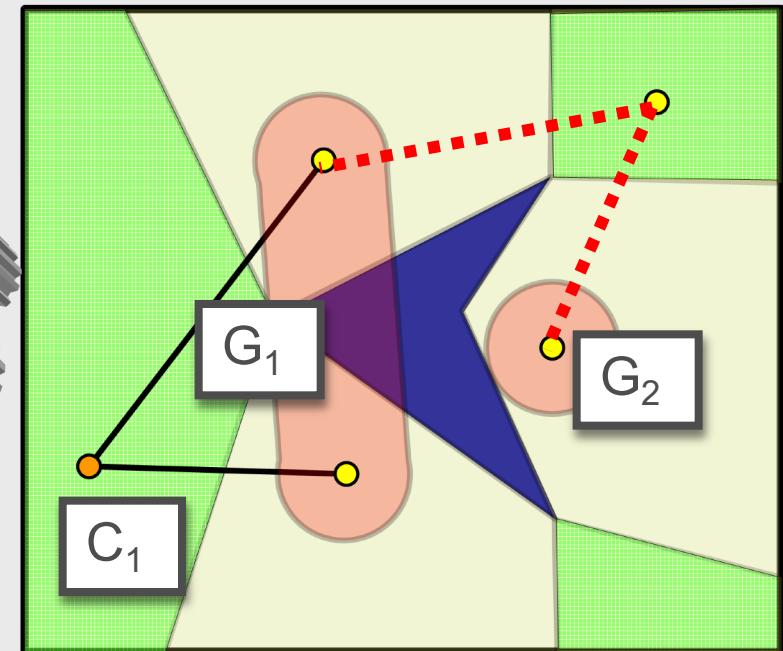
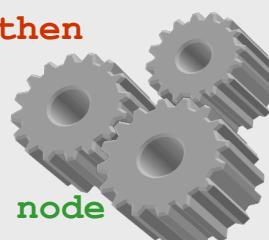
else ntry = ntry +1

end

Guard = $\{G_1, G_2\}$

Connection = $\{C_1\}$

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Build up the roadmap

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

 Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /*** q is a guard node ***/**

 add {q} to Guard; ntry = 0

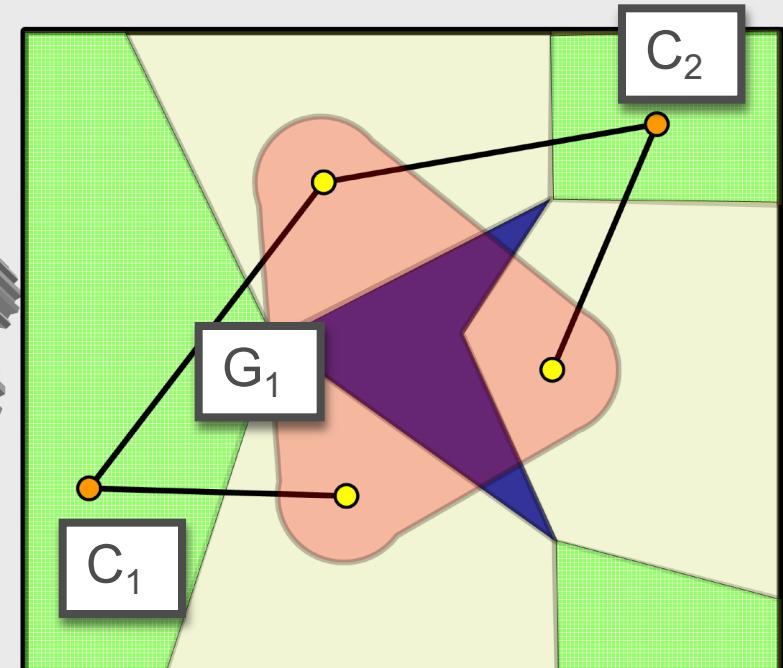
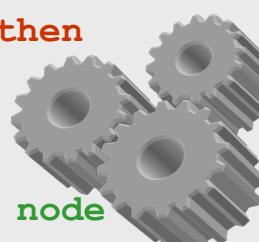
else ntry = ntry +1

end

Guard = { G_1 }

Connection = { C_1, C_2 }

ntry = 0



$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

Visibility-PRM: Termination

Input M = Number of tests to be done

Guard = \emptyset ; Connection = \emptyset ; ntry = 0

while (ntry < M)

Select a random free configuration q

$g_{vis} = \emptyset$; $G_{vis} = \emptyset$

for all components G_i of Guard **do**

 found = FALSE

for all nodes g of G_i **do**

if (q belongs to Vis(g)) **then**

 found = TRUE

if (! g_{vis}) **then**

$g_{vis} = g$; $G_{vis} = G_i$

else /* q is a connection node

 add q to Connection

 Create edges (q,g) and (q, g_{vis})

 Merge components G_{vis} and G_i

until found = TRUE

if (! g_{vis}) **then /*** q is a guard node *****

 add {q} to Guard; ntry = 0

else ntry = ntry +1

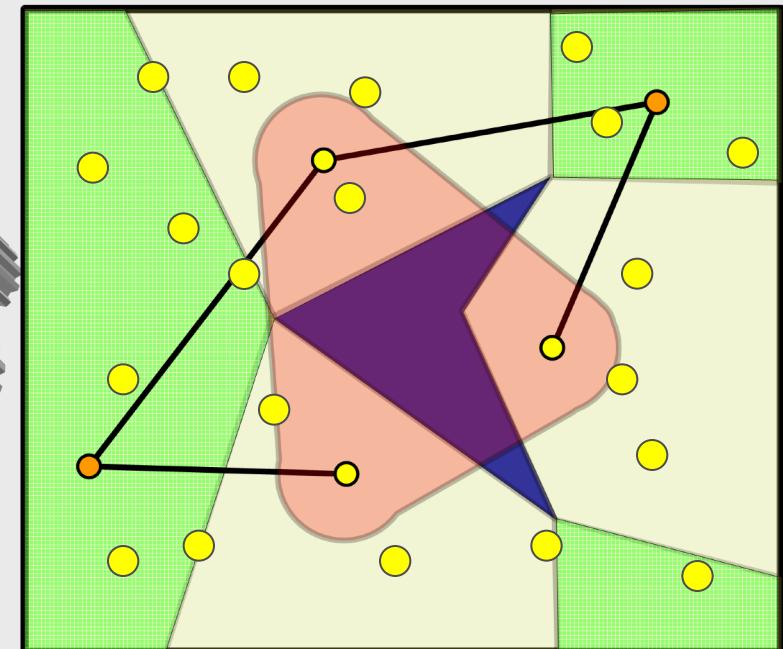
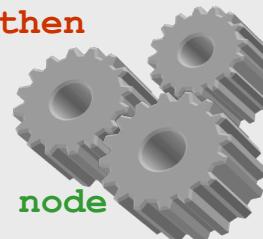
end

If successing tested nodes
are visible from roadmap
ntry increases until it is
higher than M

Guard = $\{G_1\}$

Connection = $\{C_1, C_2\}$

ntry = 0

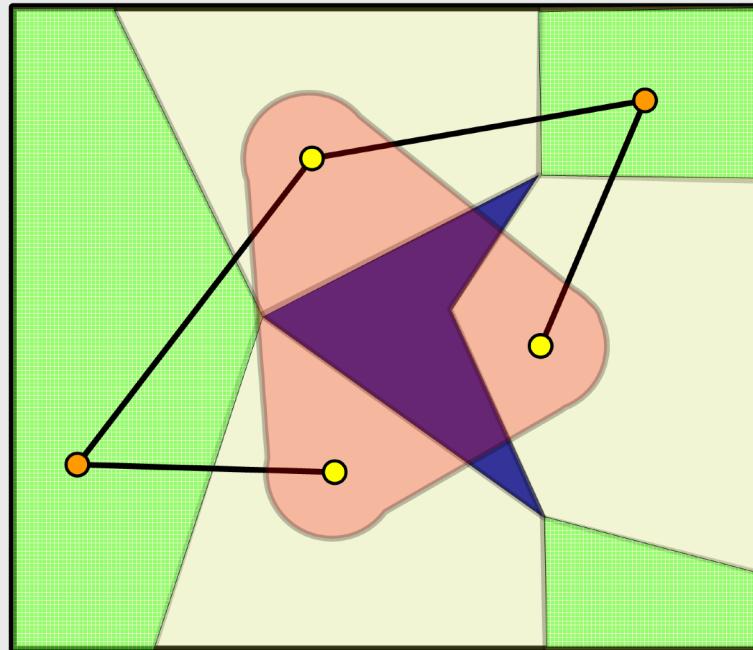


$g_{vis} = \emptyset$

$G_{vis} = \emptyset$

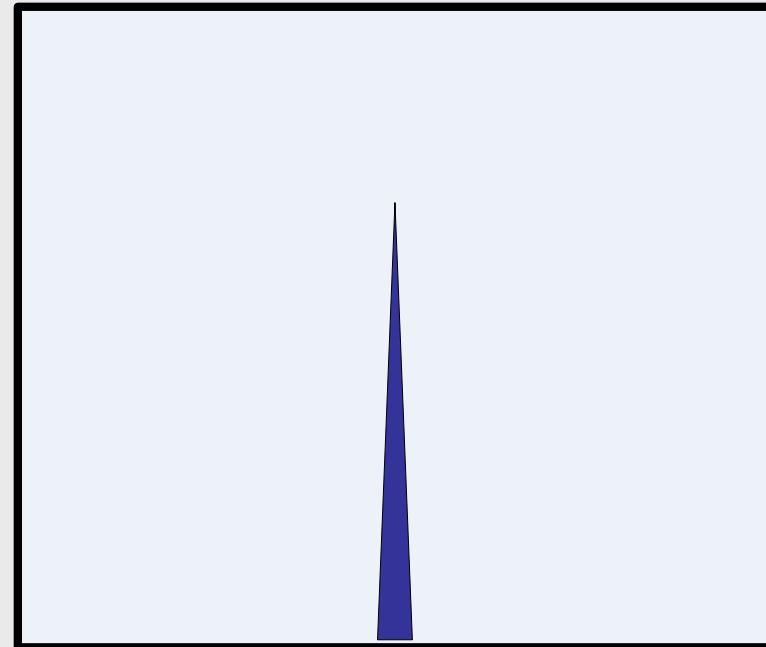
Visibility-PRM: Result

- ▶ All points in C-Space are visible from the roadmap
 - ▶ All start & and goal configuration can be connected to the roadmap



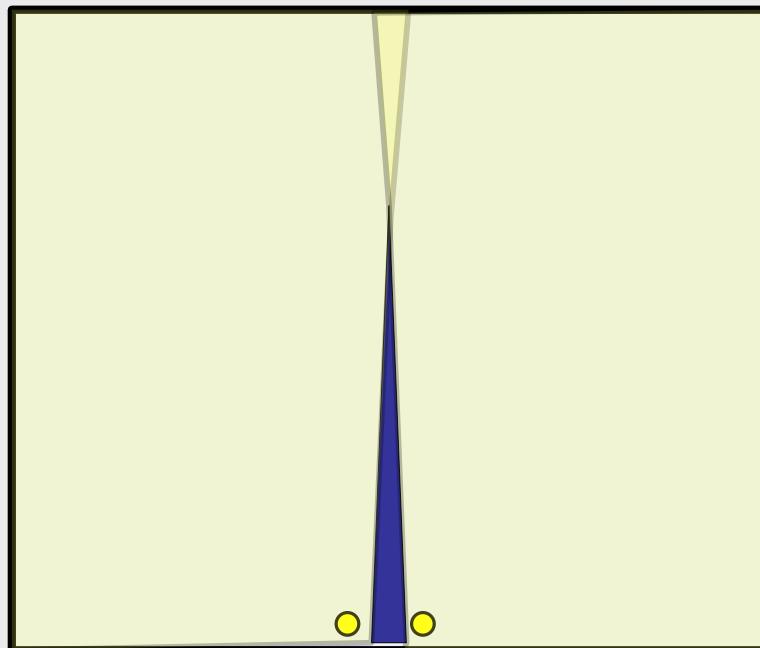
Problematic but rare case

- ▶ Why could be following benchmark be nearly unsolvable?



Problematic but rare case

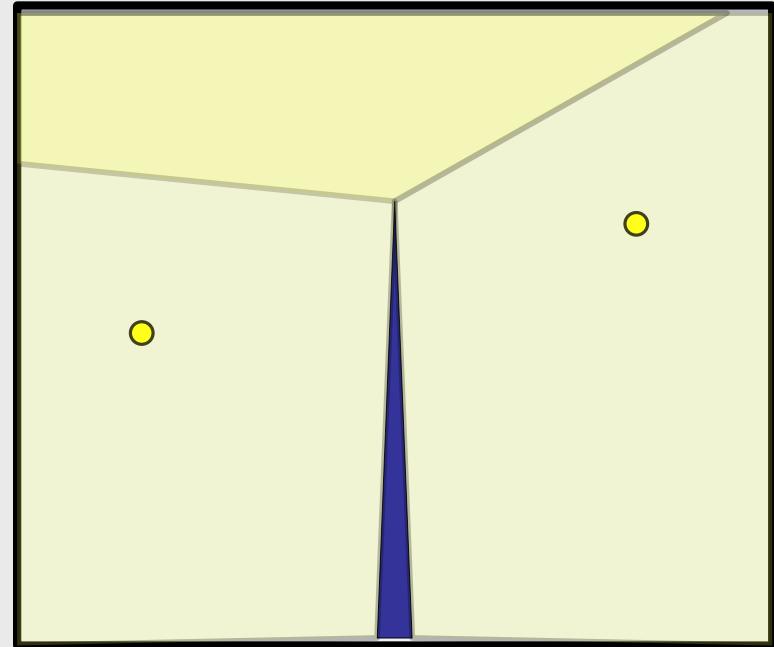
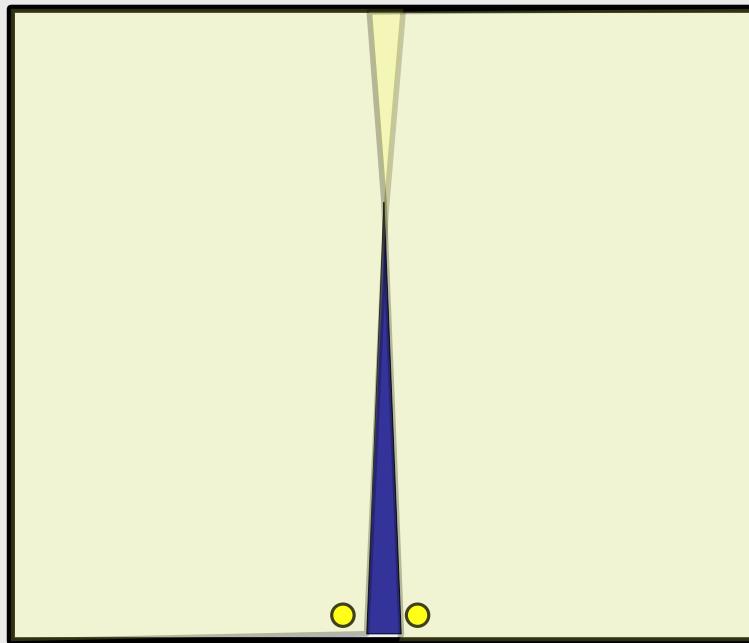
- ▶ Why could the following benchmark be nearly unsolvable?



Very Small Shared Visibility

Problematic but rare case

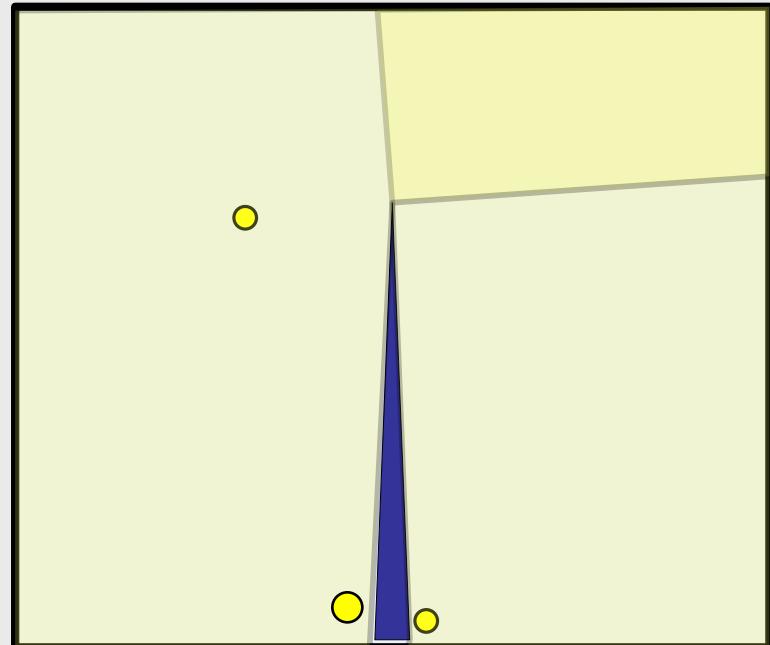
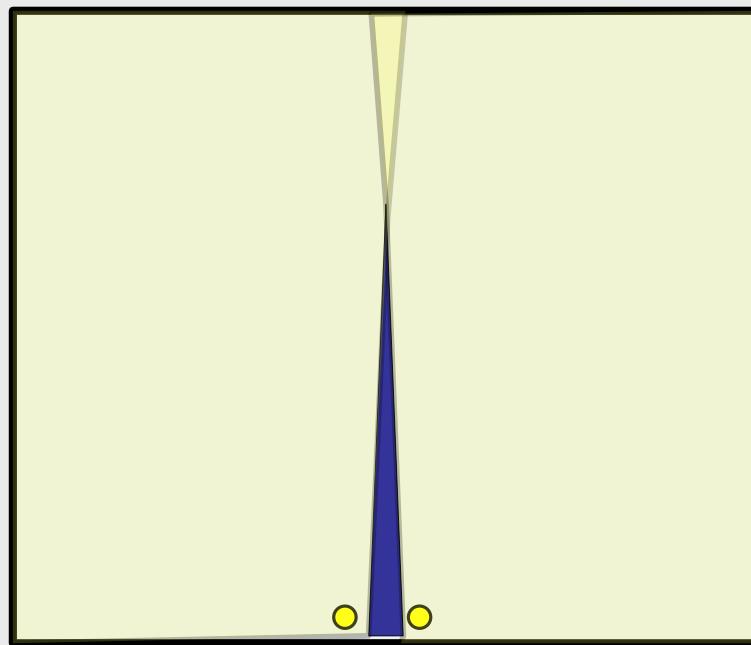
- ▶ Why could the following benchmark be nearly unsolvable?



- ▶ Solution:
 - ▶ Build new roadmap → new seedpoints
 - ▶ Move guards
 - ▶ Better sampling strategies

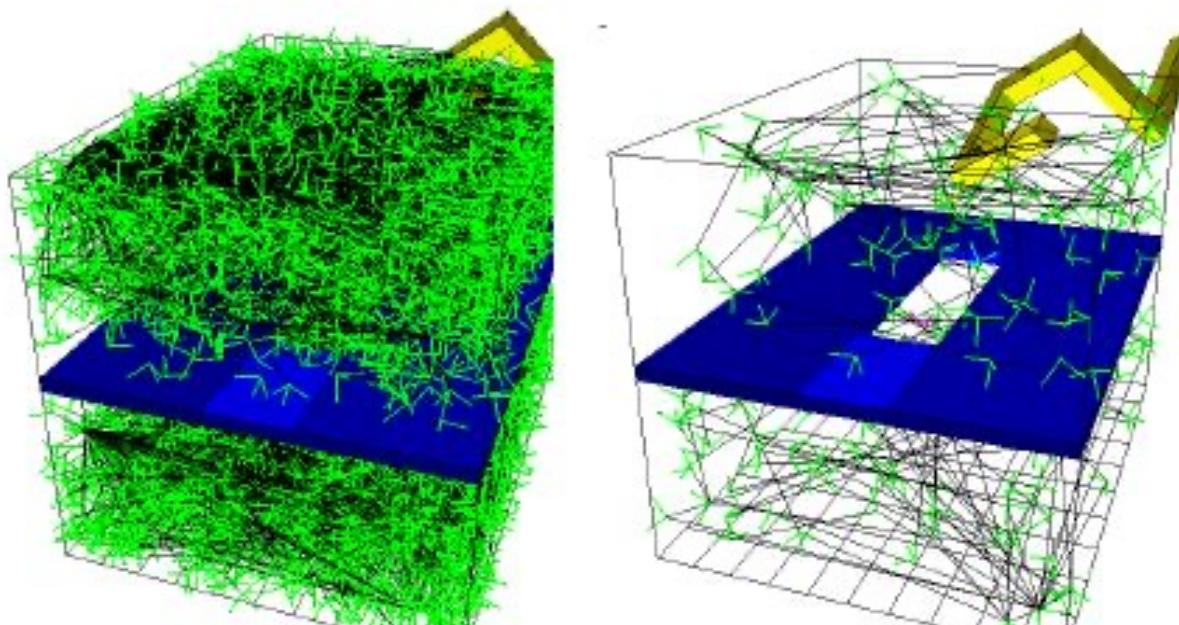
Problematic but rare case

- ▶ Why could be following benchmark be nearly unsolvable?



- ▶ Solution:
 - ▶ Build new roadmap → new seedpoints
 - ▶ Move guards
 - ▶ Better sampling strategies

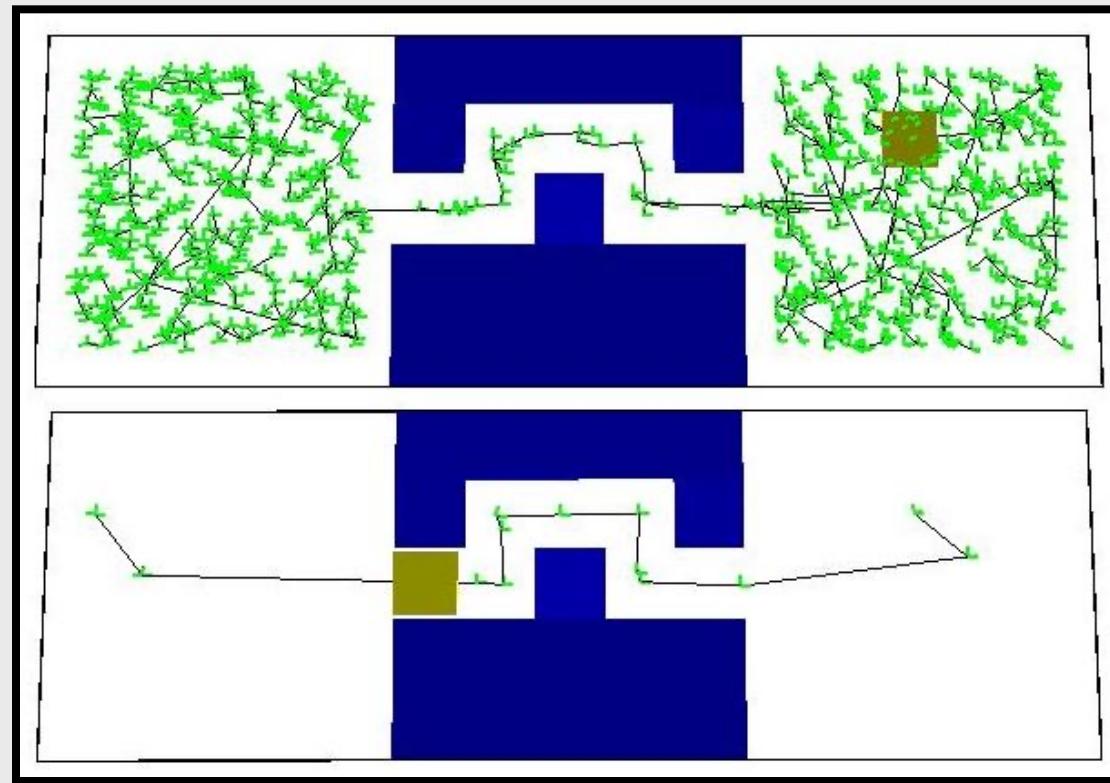
Comparison Visibility / BASIC-PRM



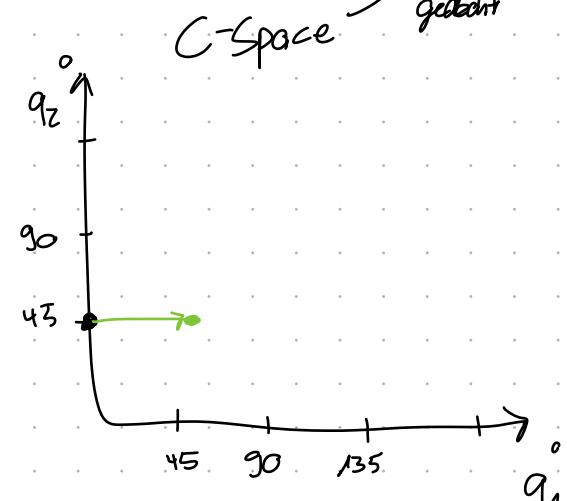
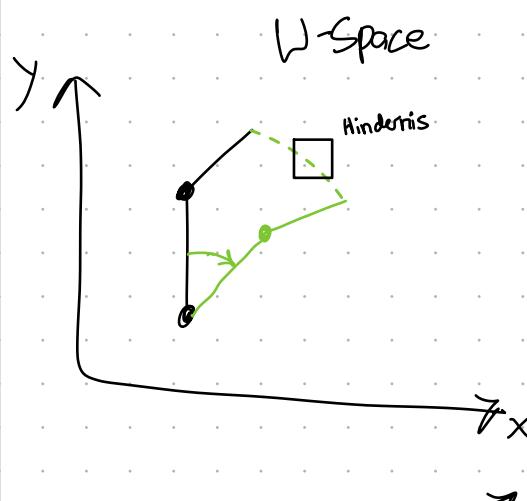
PRM	Basic	Visibility
Roadmap size	4723	103
CS_{free} coverage	99.9%	99.7%
Random confs	14169	14369
Free confs	4723	4753
Local method #calls	700610	57622
Col. checker #calls	8.725985	1.121790
CPU time	3367 sec	281 sec

Conclusion

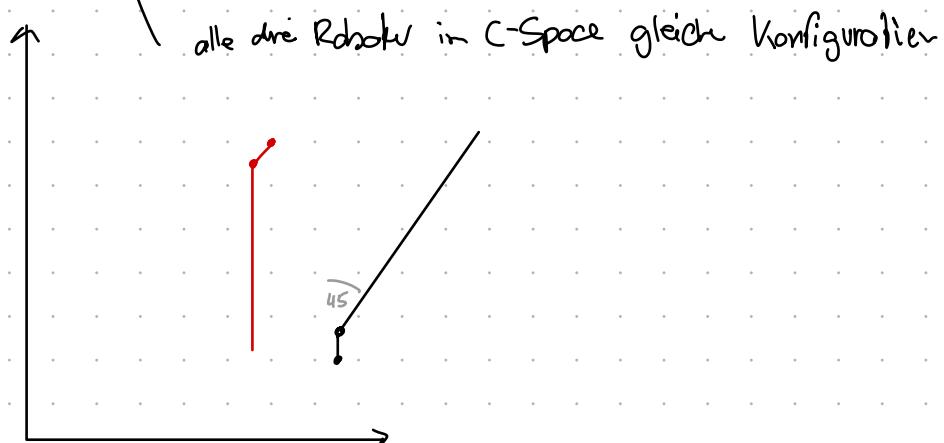
- ▶ Roadmap is drastically reduced
- ▶ Faster Queries
- ▶ Remaining problems randomly chosen configurations



nur für einen Roboter
gedacht



Idee Ein Punkt beschreibt Konfiguration des Roboters



Bahnplanungsweg muss nicht angepasst werden, da es in C-Space stattfindet

Wie übersetzt man reale Hindernisse in C-Space \rightarrow mathematisch nicht gut möglich

\rightarrow Hindernisse im C-Space nicht sichtbar

\hookrightarrow Sampling: Unterteilung C-Space in viele Bereiche und Überprüfung (Ritter) ob Kollision vorliegt

\rightarrow Kollisionsfreier Pfad \rightarrow Überprüfung Pfad C-Space in äquidistantem Ritter ob in Realität kollidiert

\rightarrow hoher Aufwand keine gute Idee

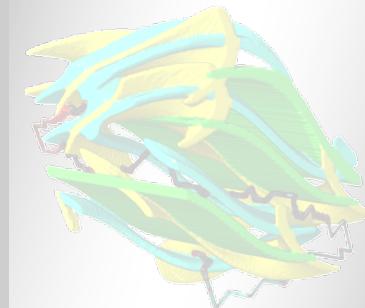
A^* nur für Roboter mit max 6 Achsen

A^* kann optimales Pfad finden, benötigt jedoch sehr viel Zeit

Lazy-PRM



Why do all this extensive collision testing?
(R. Bohlin and L. Kavraki)



Disadvantages so far!

- ▶ Proposed algorithms spend a lot of time planning paths that will never get used.
 - ▶ Especially, when you just want to do single-queries
- ▶ Heavily reliant on fast collision checking / distance computation.
- ▶ An attempt to solve these is made with Lazy PRMs
 - ▶ Tries to minimize collision checks
 - ▶ Tries to reuse information gathered by queries

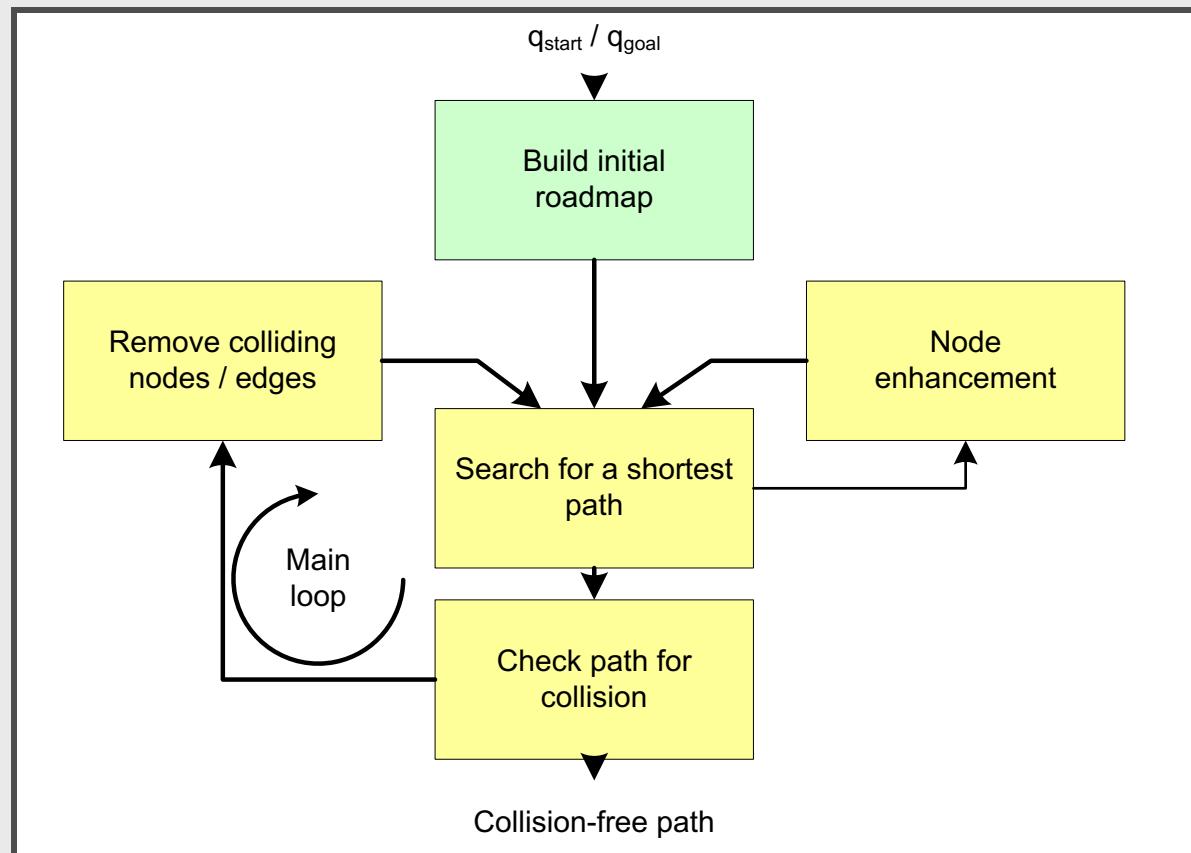
References

Ideas taken from presentation & papers:

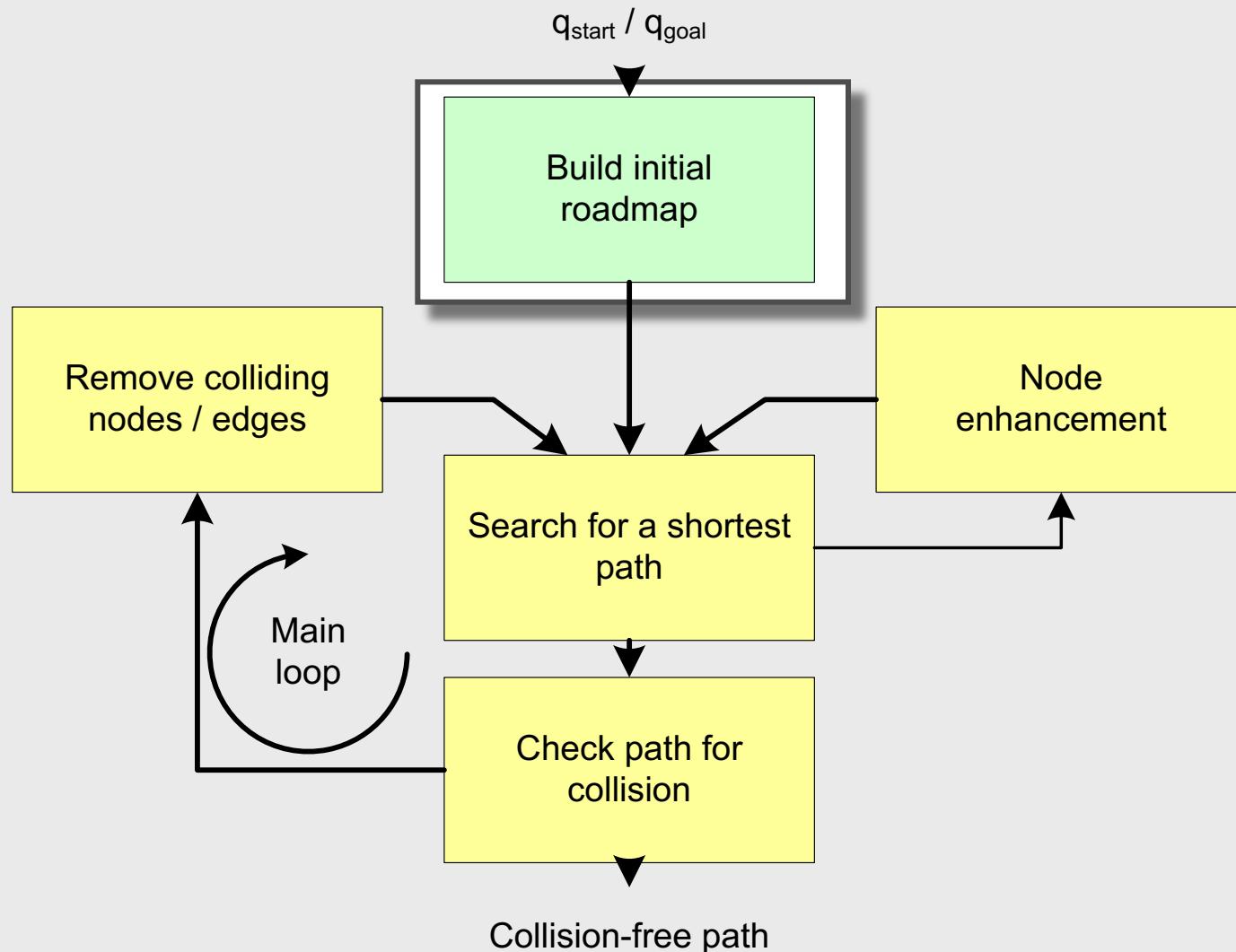
- ▶ “**On Delaying Collision Checking in PRM Planning**”: presentation (G. Sanchez, J. Latombe, Computer Science Department, Stanford University)
- ▶ “**Path Planning using Lazy-PRM**”: paper of T.V.N. Sri Ram
- ▶ “**A General Framework for Sampling on the Medial Axis of the Free Space**”: presentation (Jyh-Ming Lien, Shawna Thomas, Nancy Amato)
- ▶ “**The Gaussian Sampling Strategy for Probabilistic Roadmap Planners**”: paper (Boor, Overmars, Stappen)
- ▶ “**Probabilistic Motion Planning**”: presentation (Shai Hirsch)
- ▶ “**Rapidly-Exploring Random Trees**”: presentation (Steven Lavalle)

Lazy-PRM: the idea behind

- ▶ Instead of building a roadmap of feasible paths,
build a roadmap of paths assumed to be feasible.
- ▶ Test feasibility as late as possible → during query phase

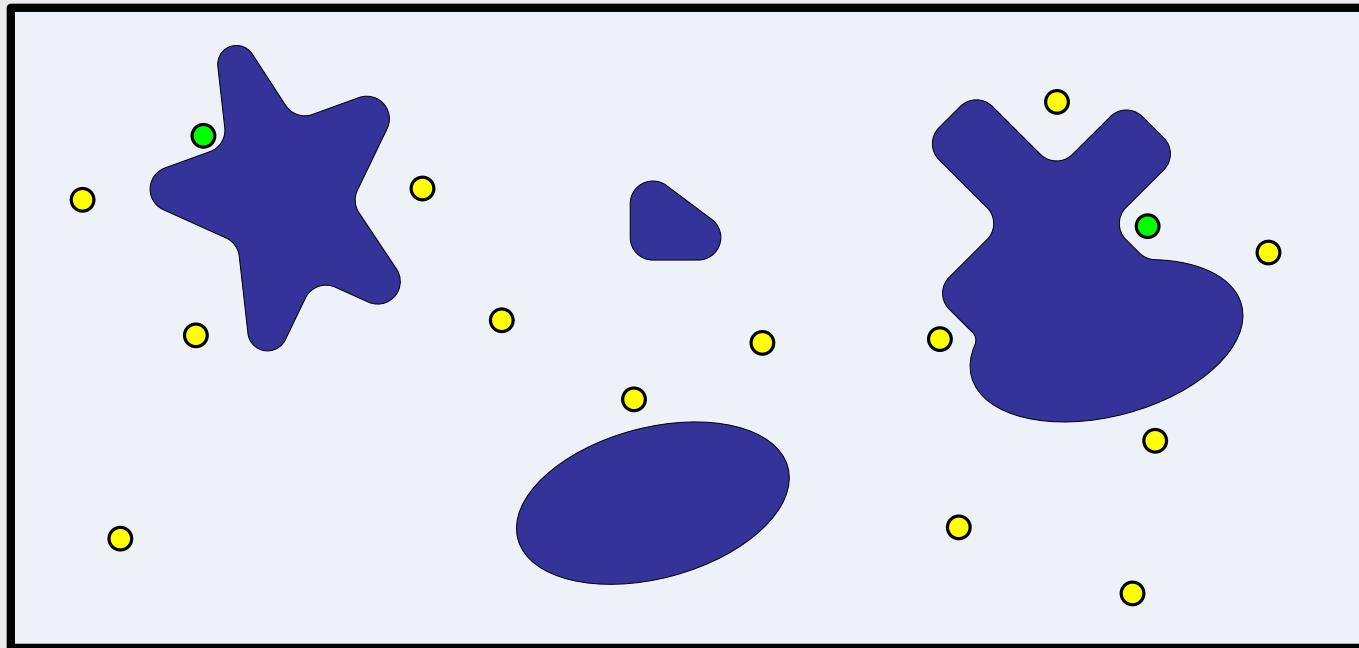


Lazy-PRM: The flow.....



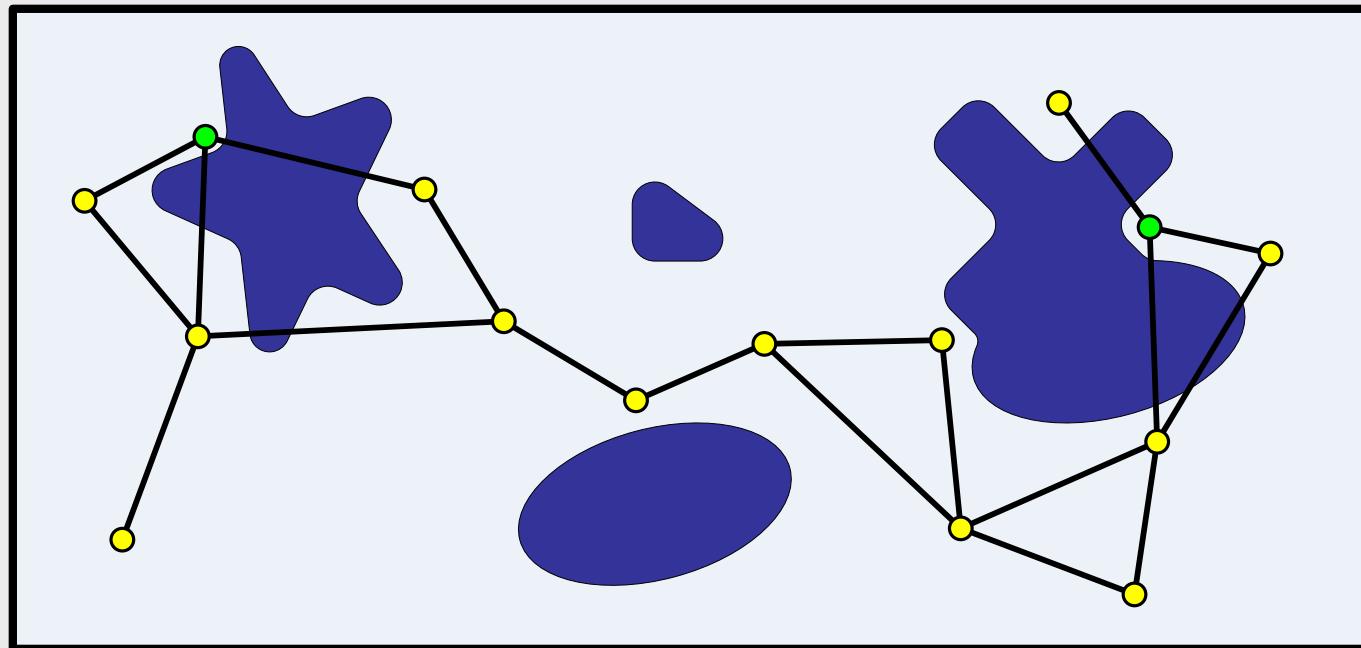
Lazy-PRM: Build initial Roadmap

- ▶ Insert start & goal and N_{init} uniformly distributed nodes, don't care about collisions

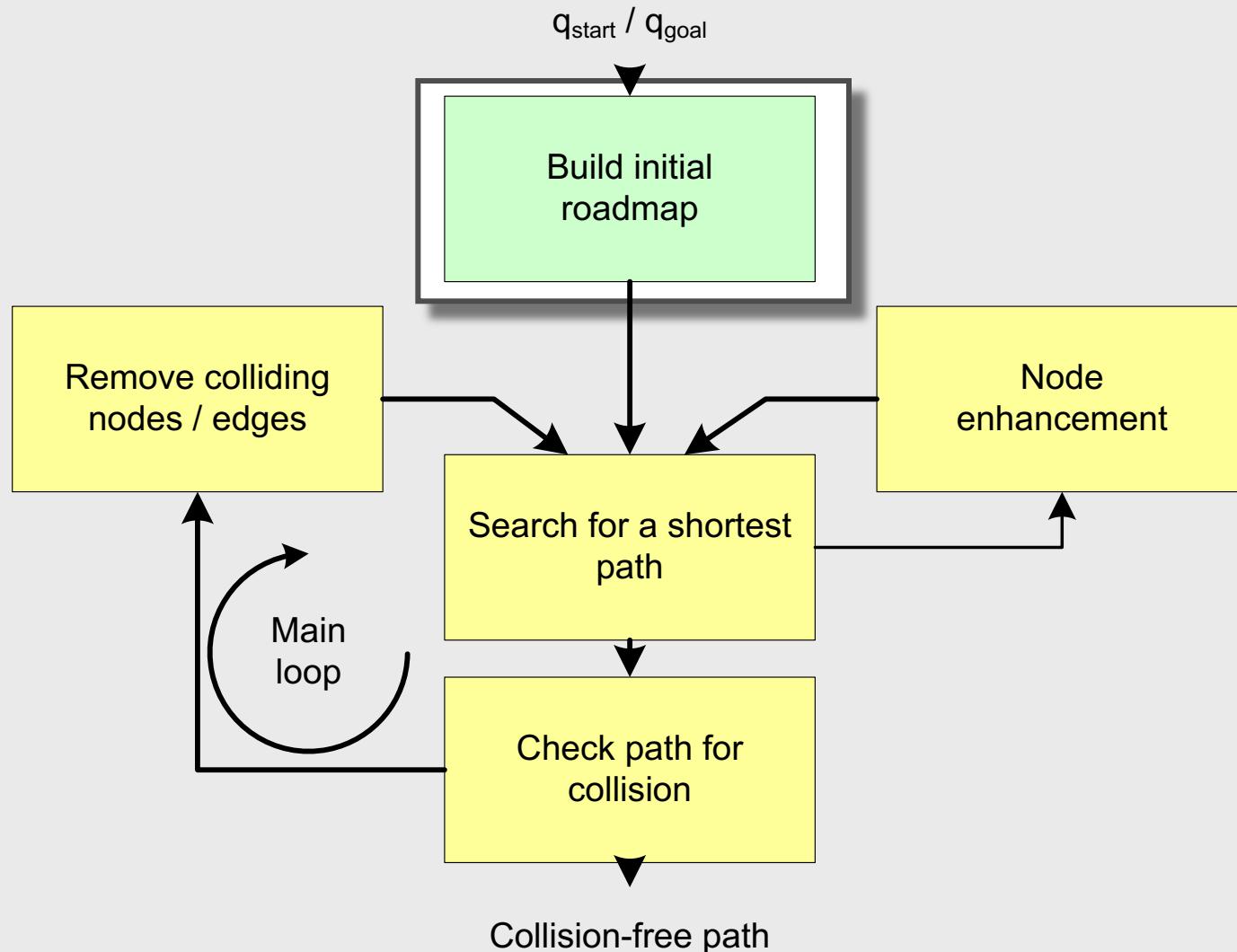


Lazy-PRM: Build initial Roadmap

- ▶ Insert start & goal and N_{init} uniformly distributed nodes, **don't care about collisions**
- ▶ Build roadmap using k-closest strategy or something similar, **don't care about collisions**

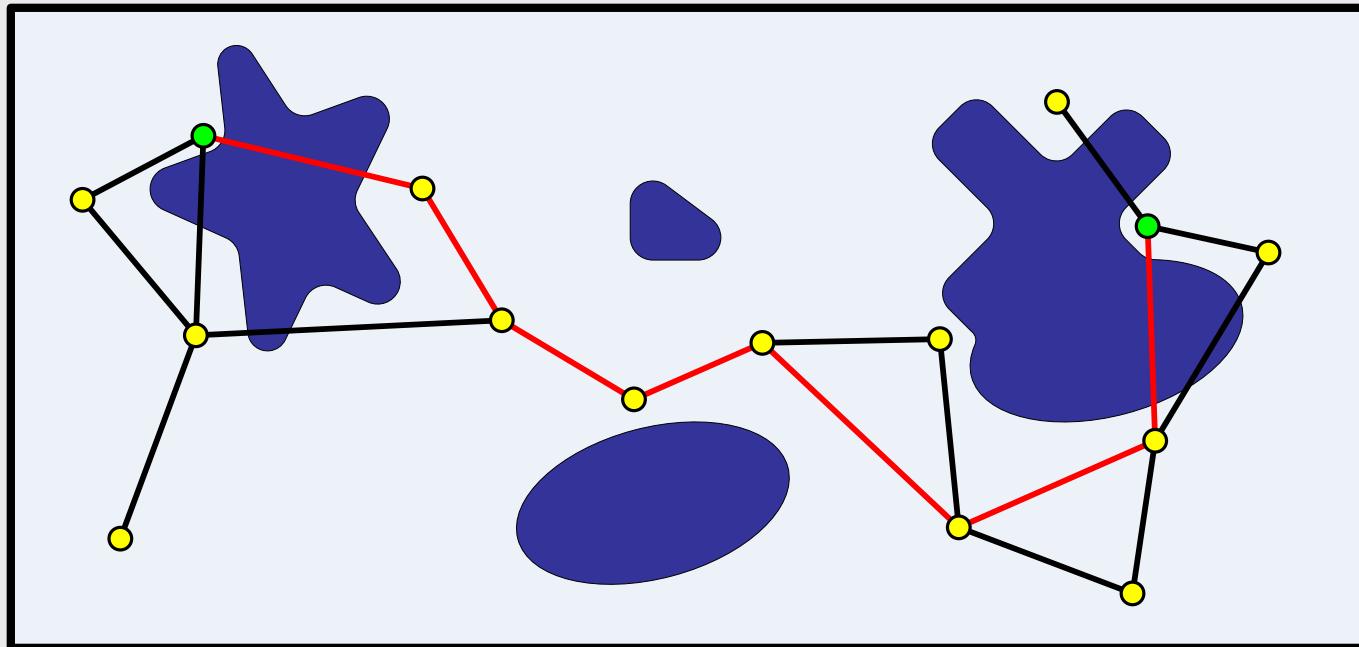


Lazy-PRM: The flow.....

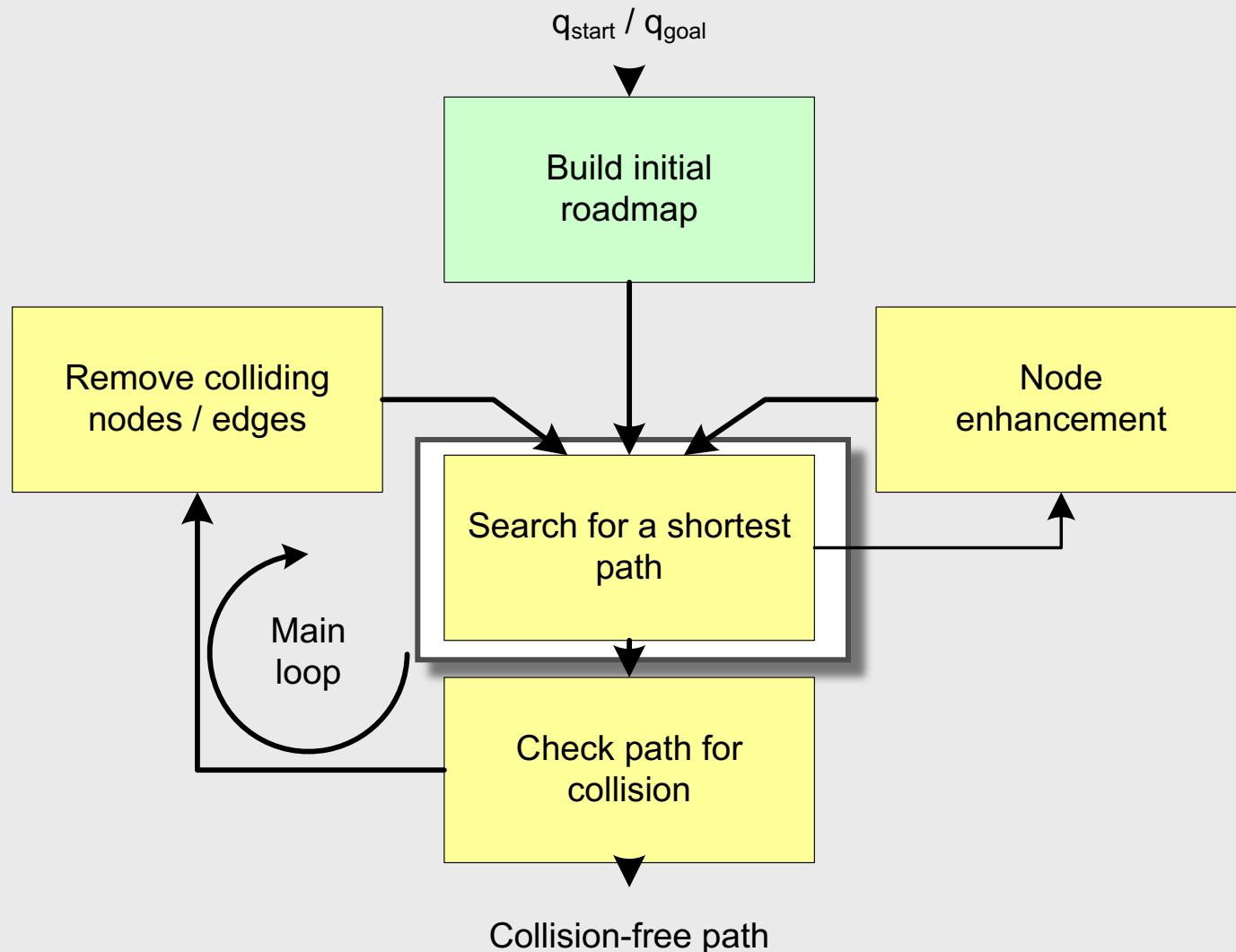


Lazy-PRM: Search shortest path

- ▶ Search shortest path on computed Roadmap, **don't care about collisions**.
 - ▶ Using A* or Dijkstra

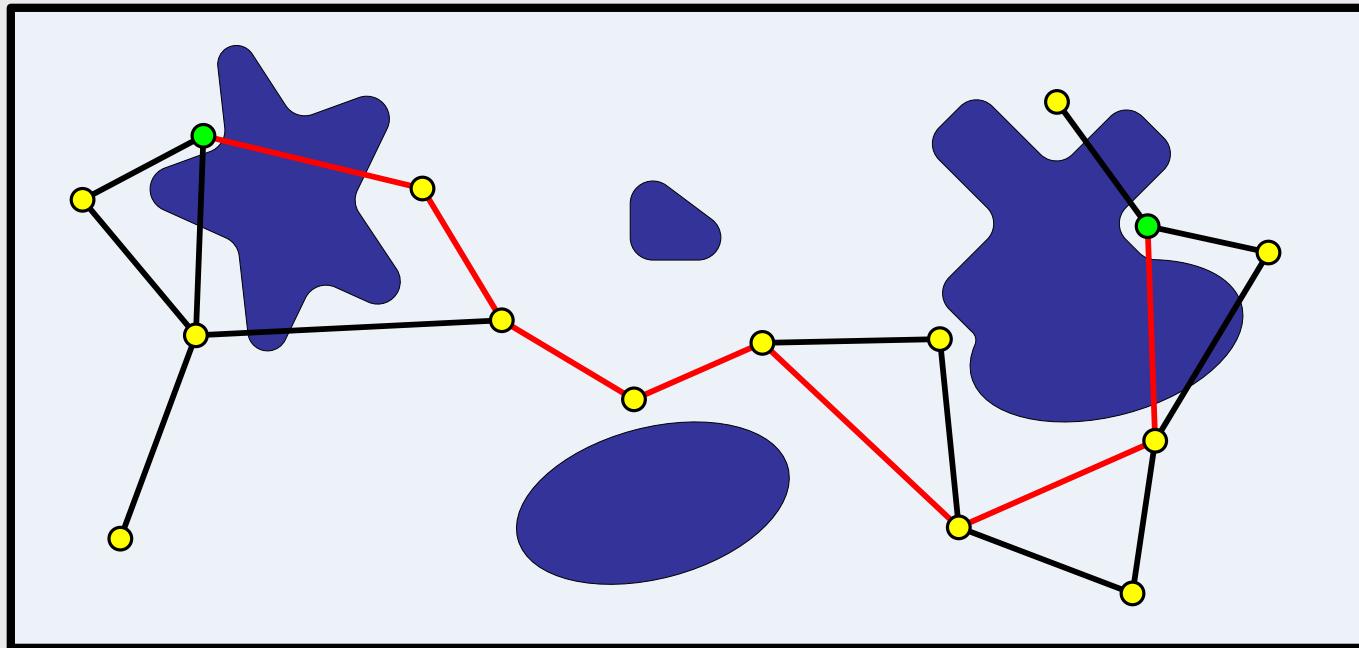


Lazy-PRM: The flow.....



Lazy-PRM: Check nodes /path for collisions

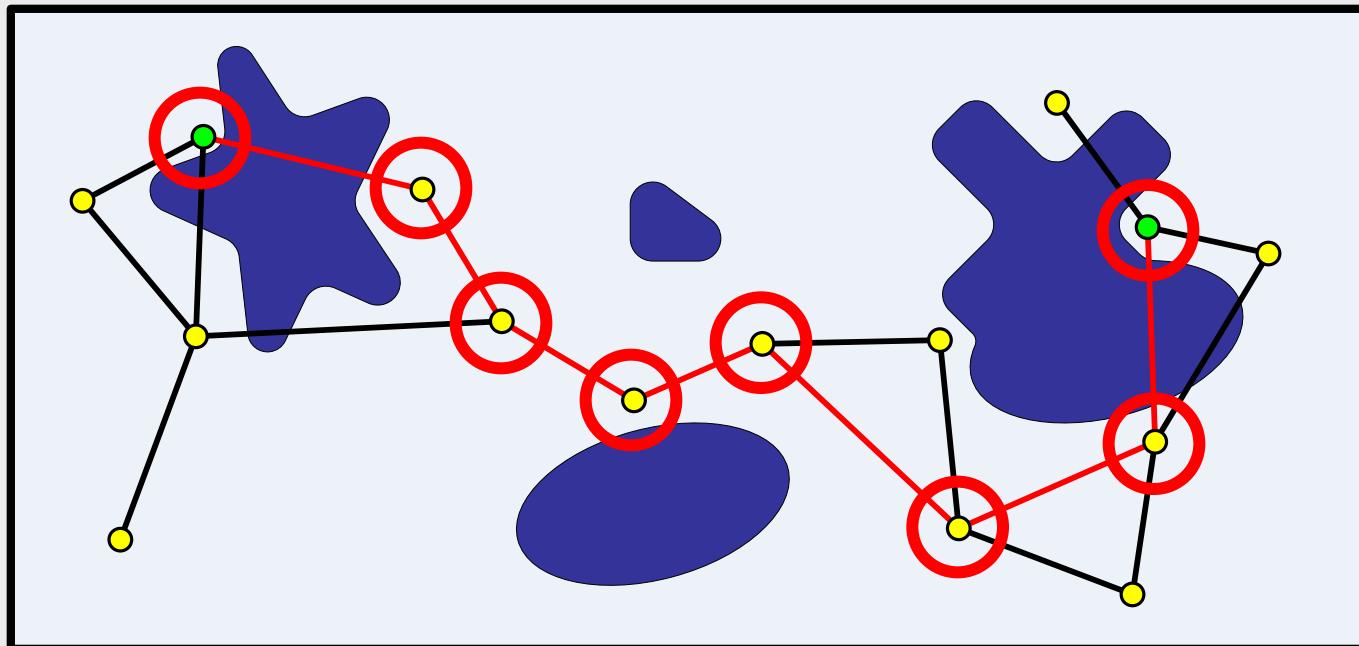
- ▶ Check nodes for collisions



Lazy-PRM: Check nodes /path for collisions

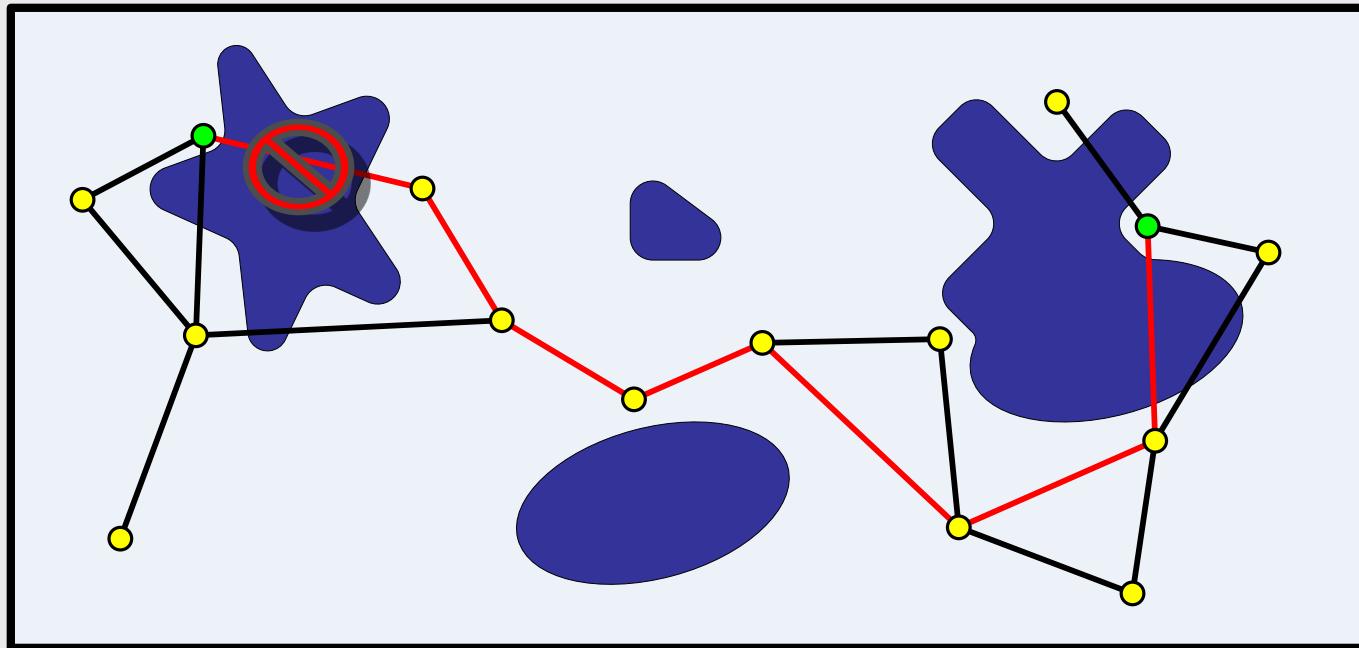
▶ Check nodes for collisions

- ▶ Starting with first and last point of path
- ▶ Check nodes alternatingly
- ▶ If a node is colliding remove node and compute again shortest path

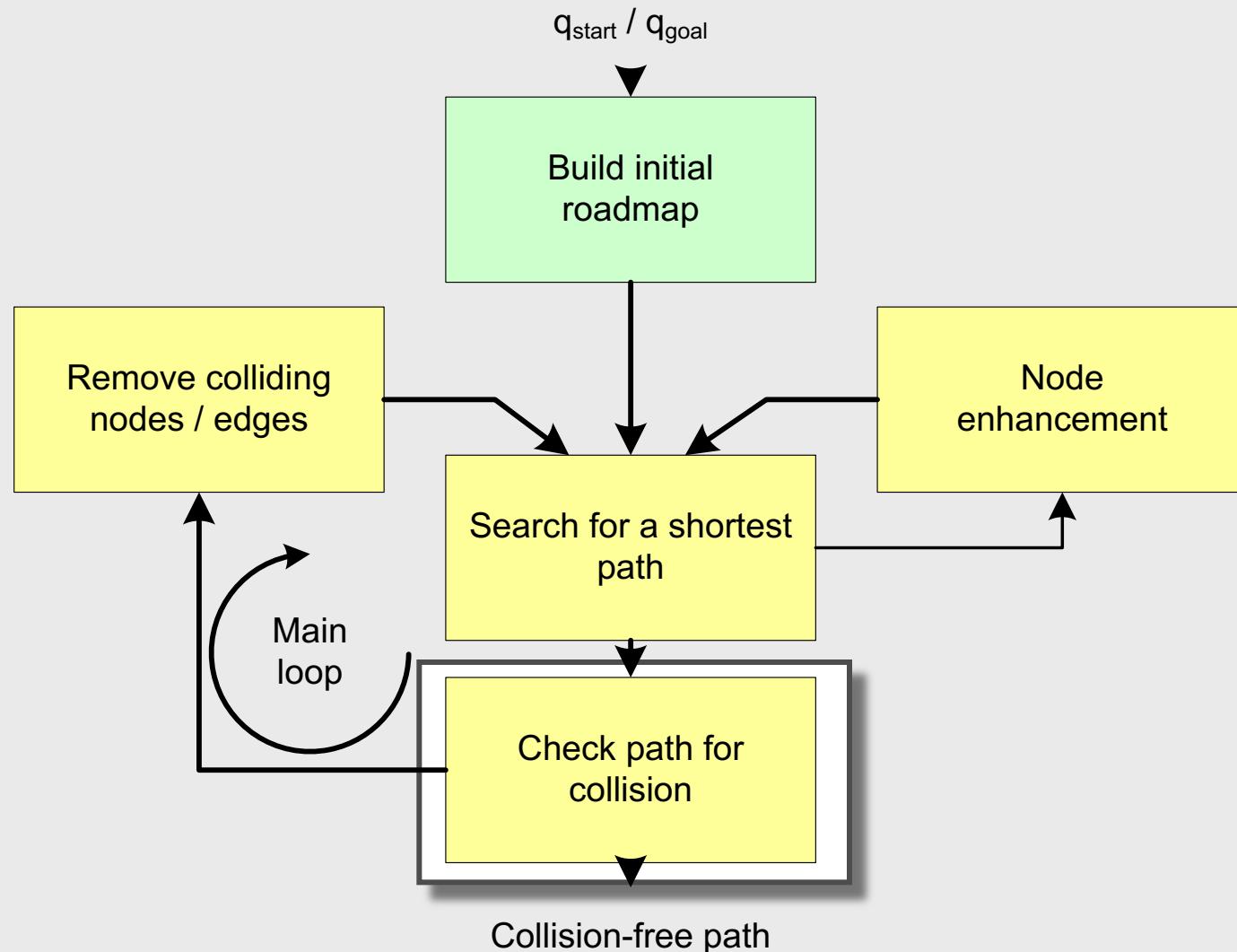


Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions
 - ▶ Alternatingly like nodes

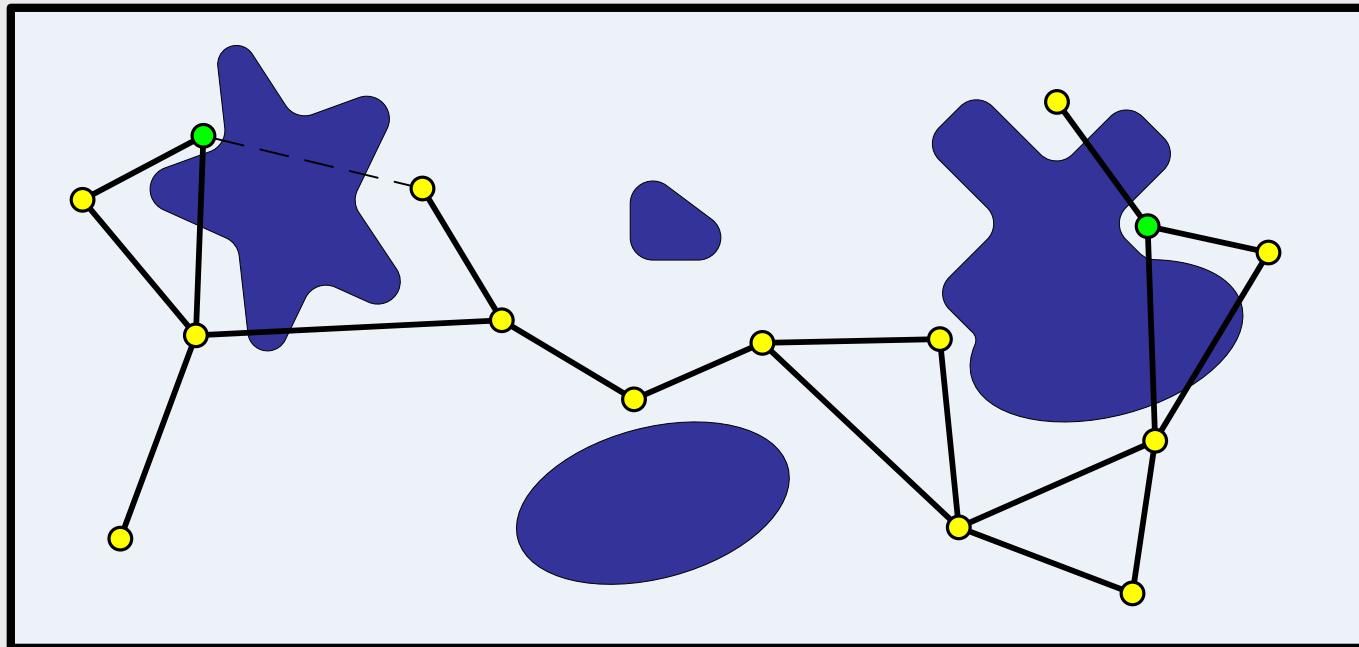


Lazy-PRM: The flow.....

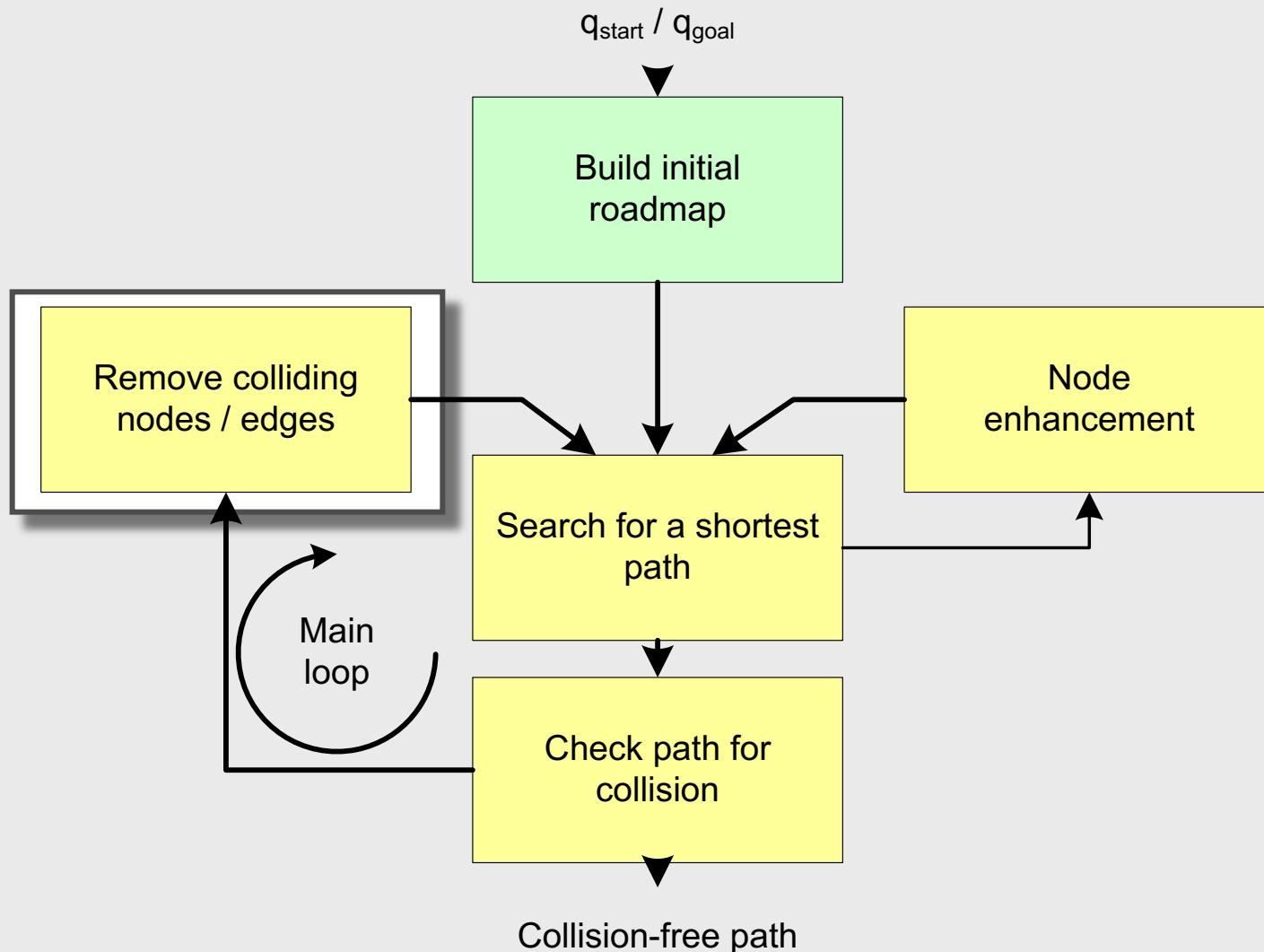


Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions
 - ▶ Remove colliding edges

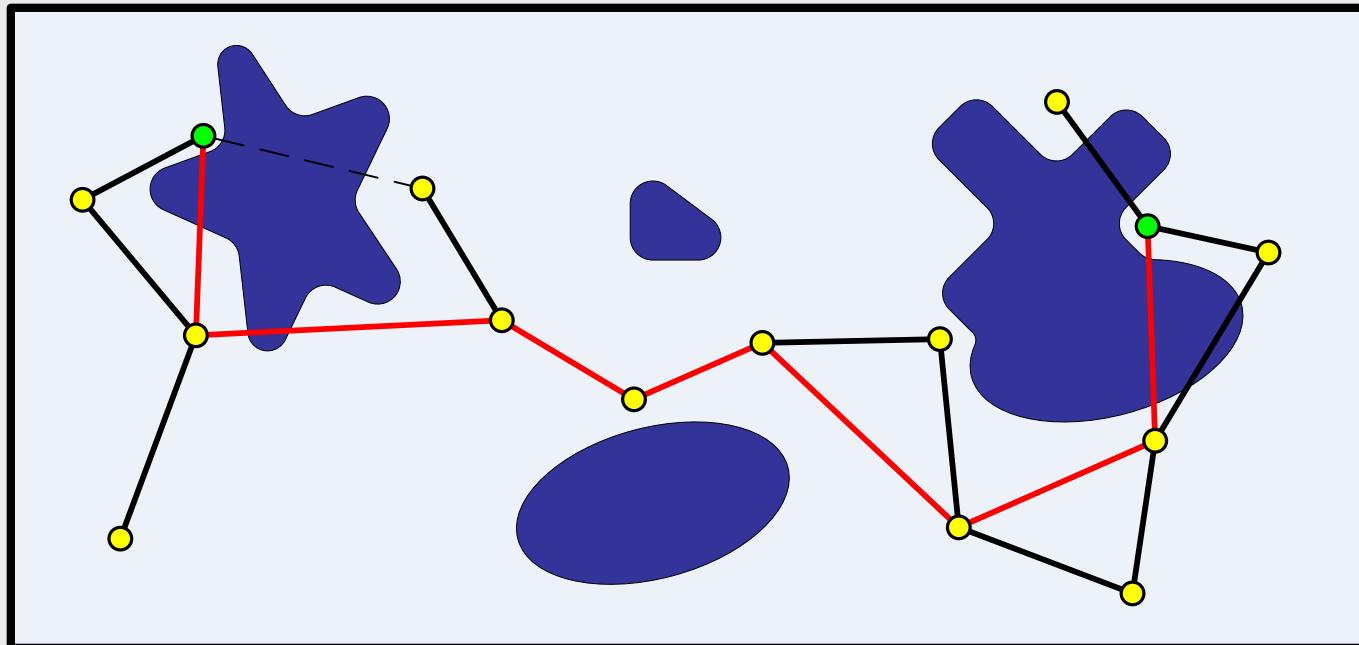


Lazy-PRM: The flow.....



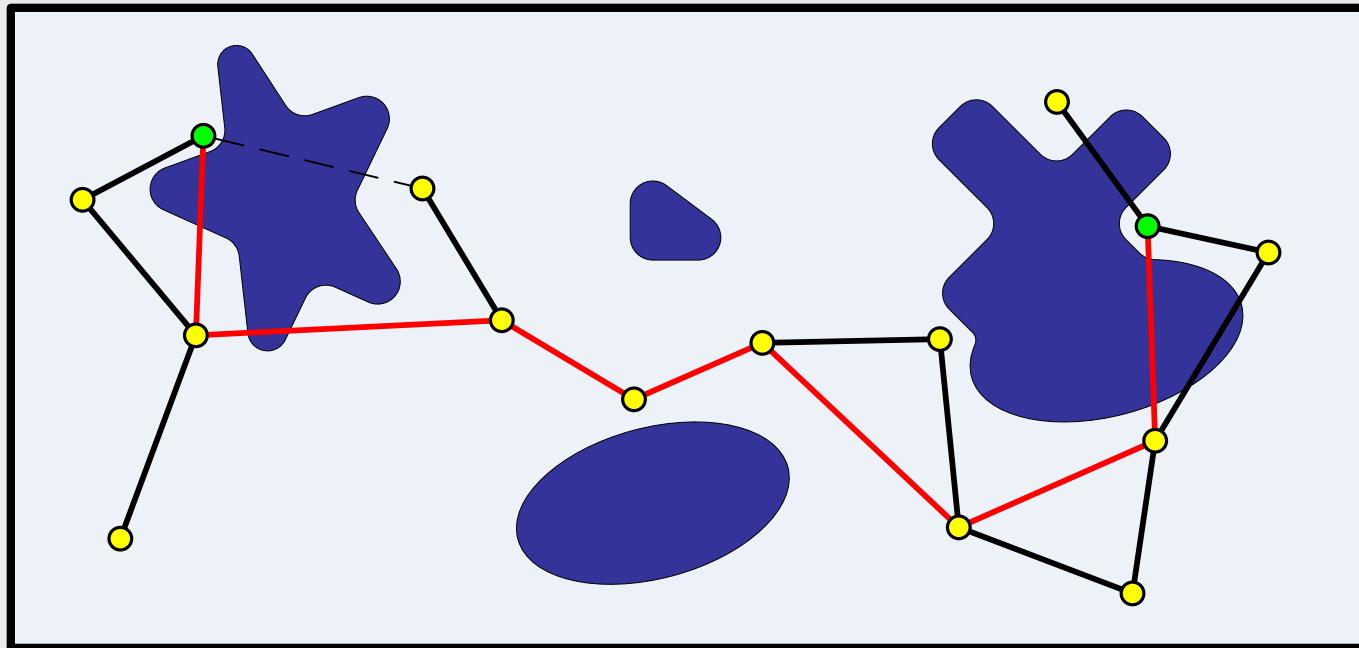
Lazy-PRM: Search again shortest path

- ▶ Search shortest path on computed Roadmap, **don't care about collisions**.
 - ▶ Using A* or Dijkstra



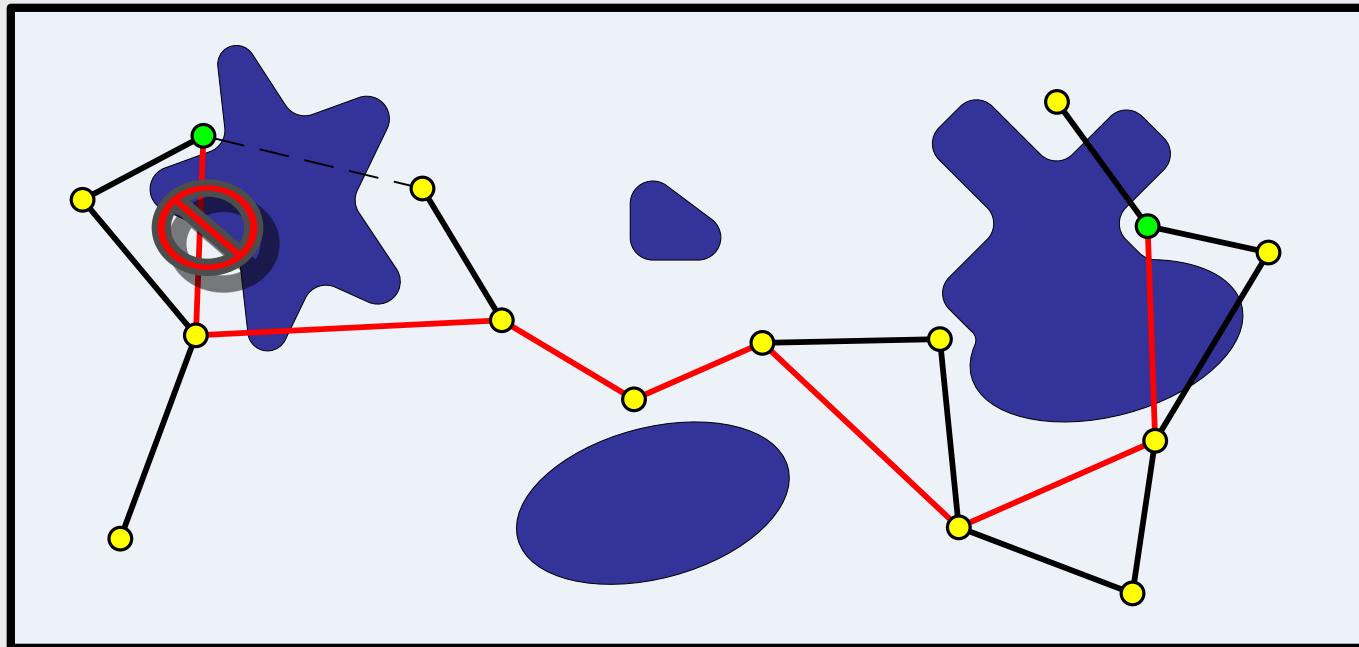
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions



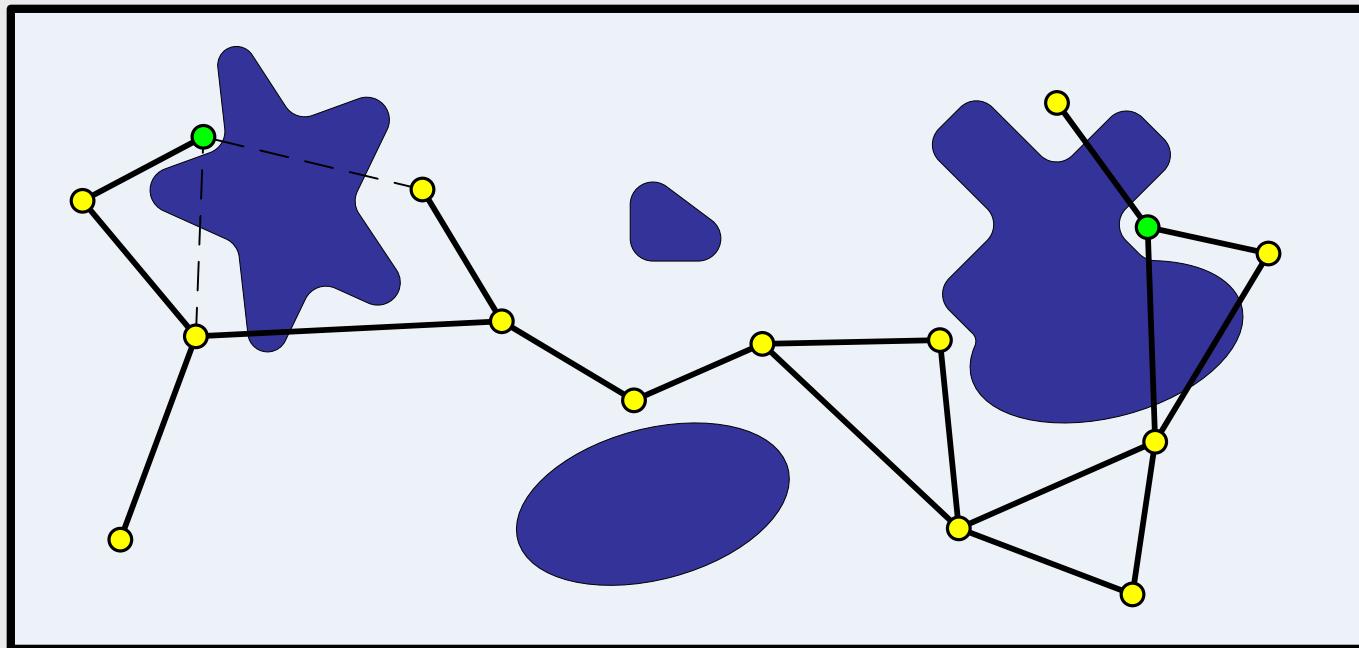
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions



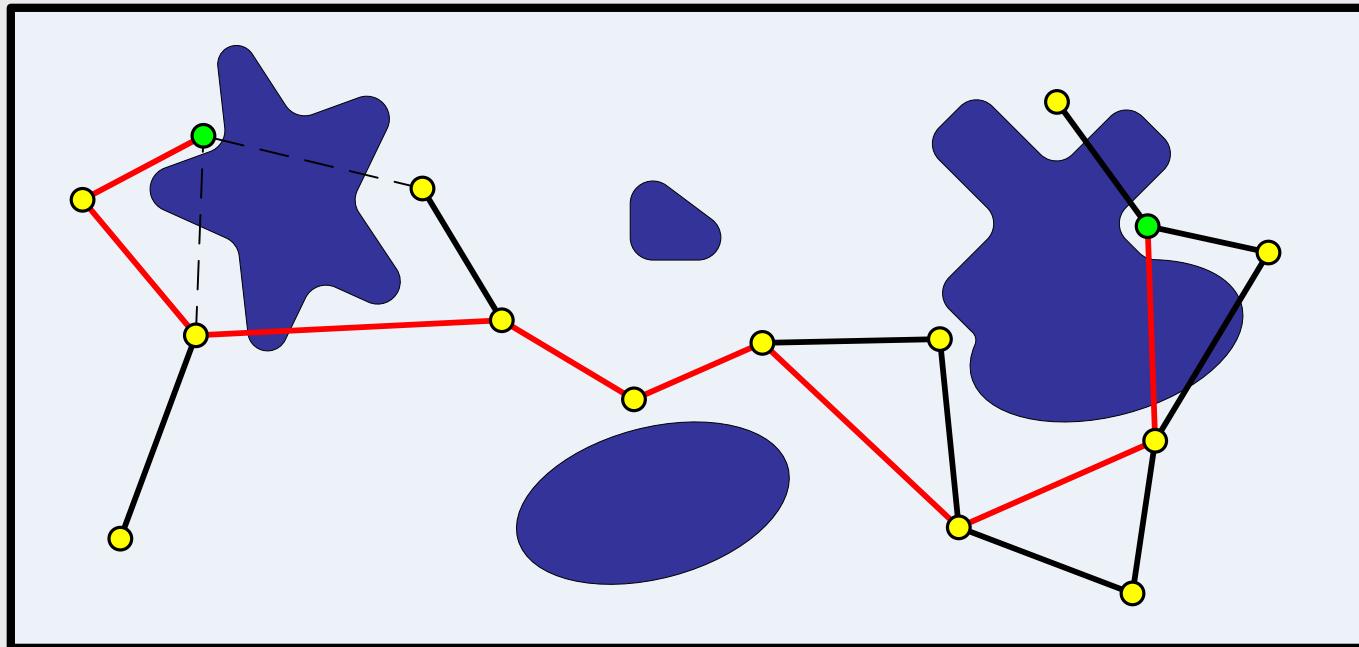
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions
 - ▶ Remove colliding edges



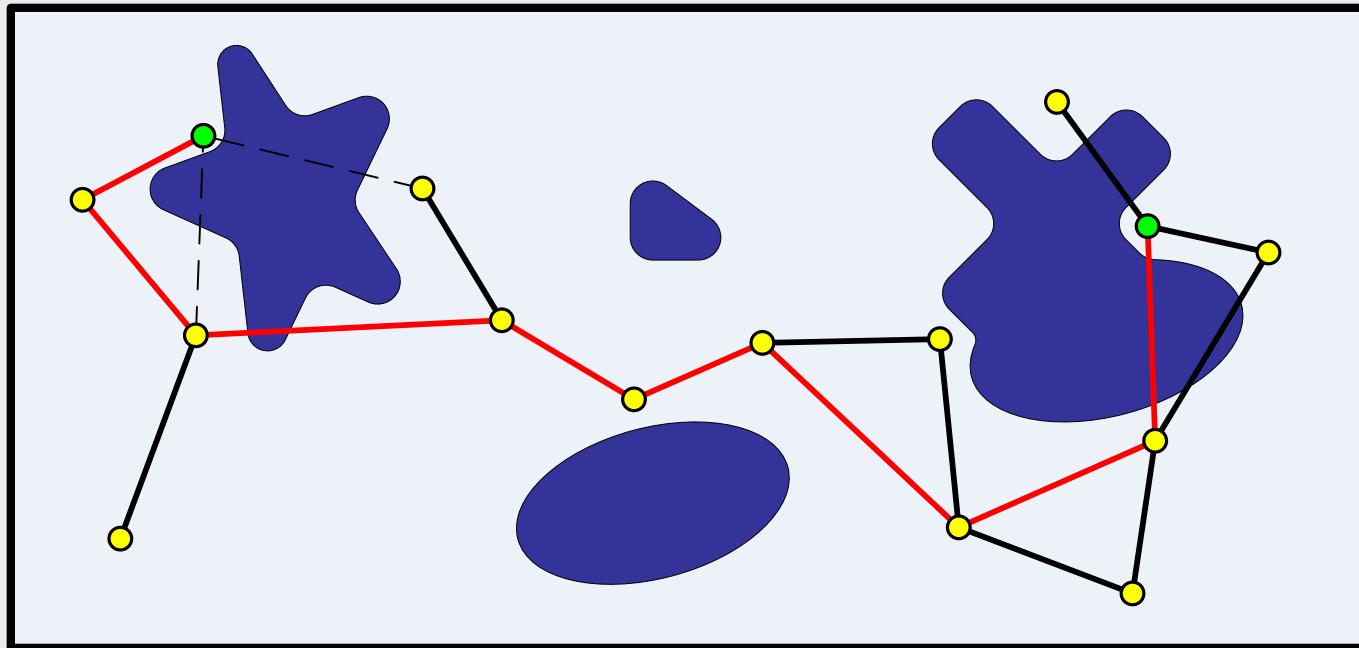
Lazy-PRM: Search again shortest path

- ▶ Search shortest path on computed Roadmap, **don't care about collisions**.
 - ▶ Using A* or Dijkstra



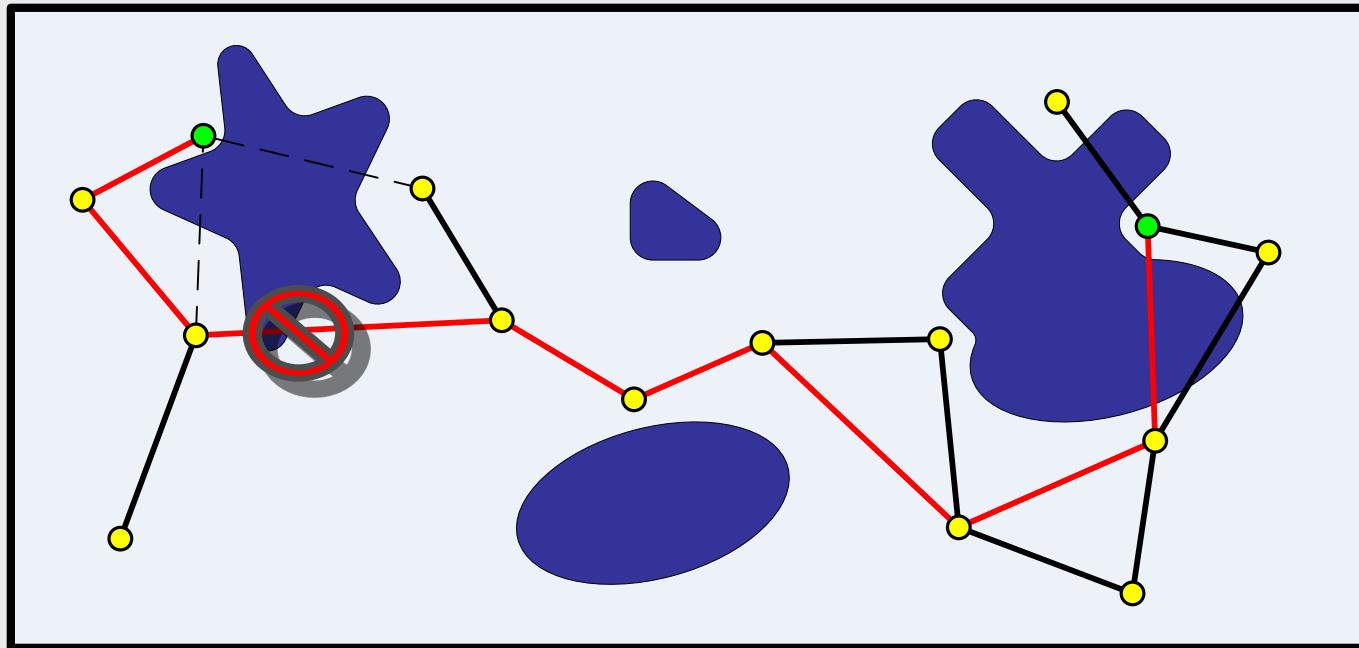
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions



Lazy-PRM: Check nodes /path for collisions

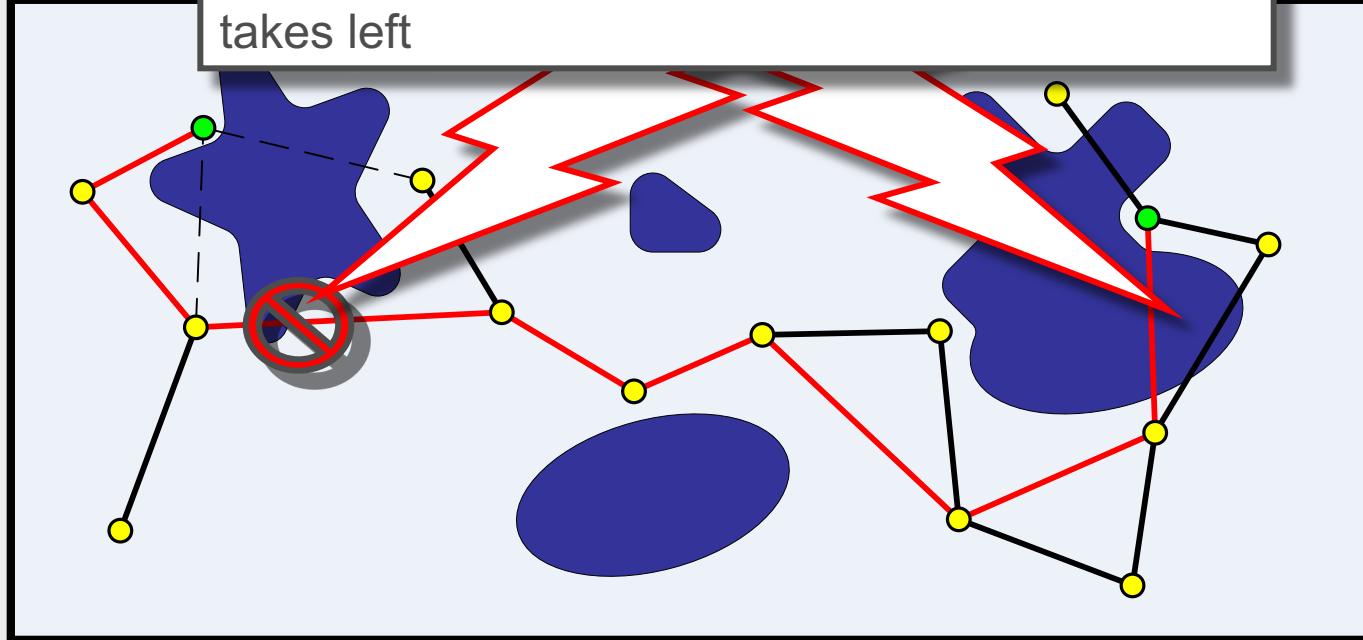
- ▶ Check nodes for collisions
- ▶ Check edges for collisions



Lazy-PRM: Check nodes /path for collisions

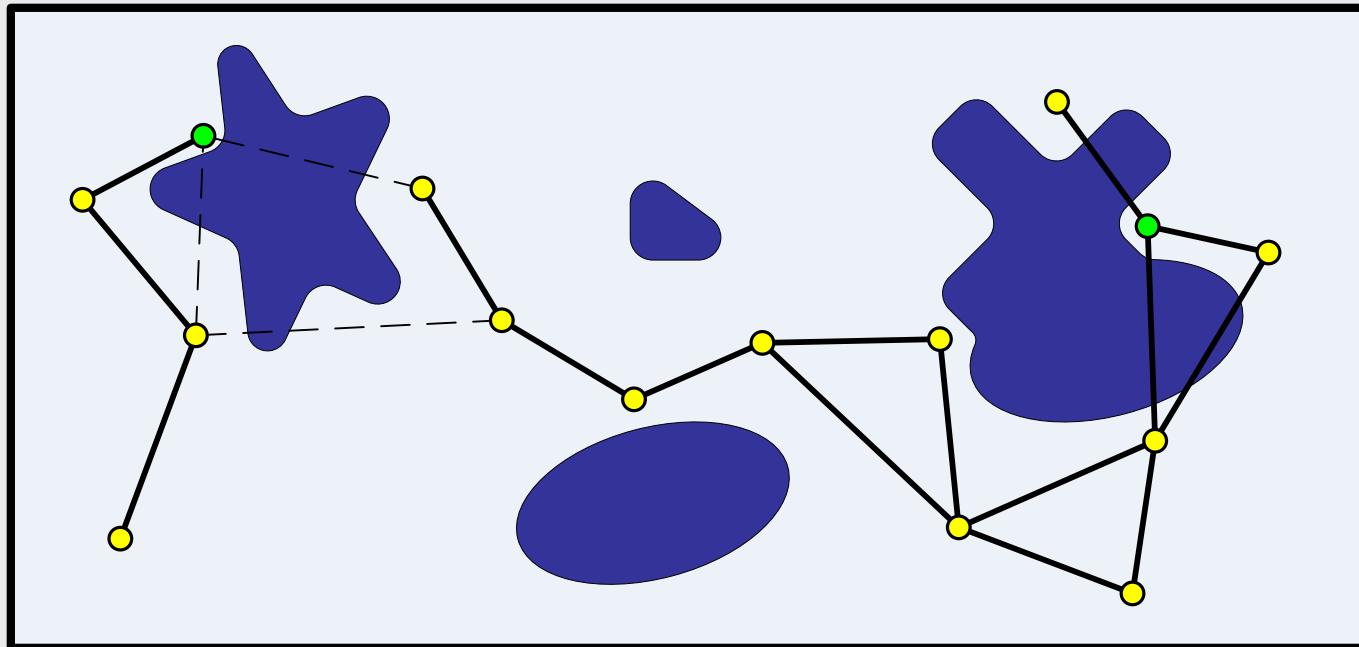
- ▶ Check nodes for collisions
- ▶ Check edges for collisions

Errata! Following correctly the algorithm right edge would be the first to be found colliding, due to alternating testing \leftrightarrow example wrongly takes left



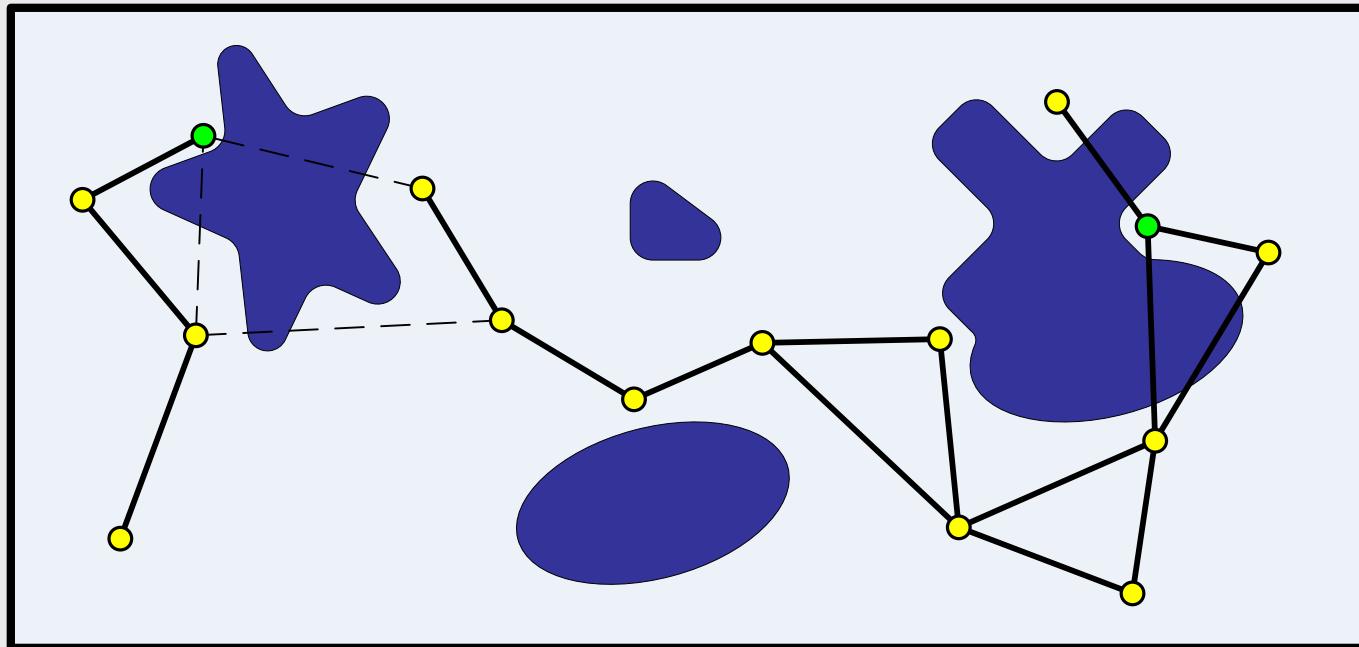
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions
 - ▶ Remove colliding edges

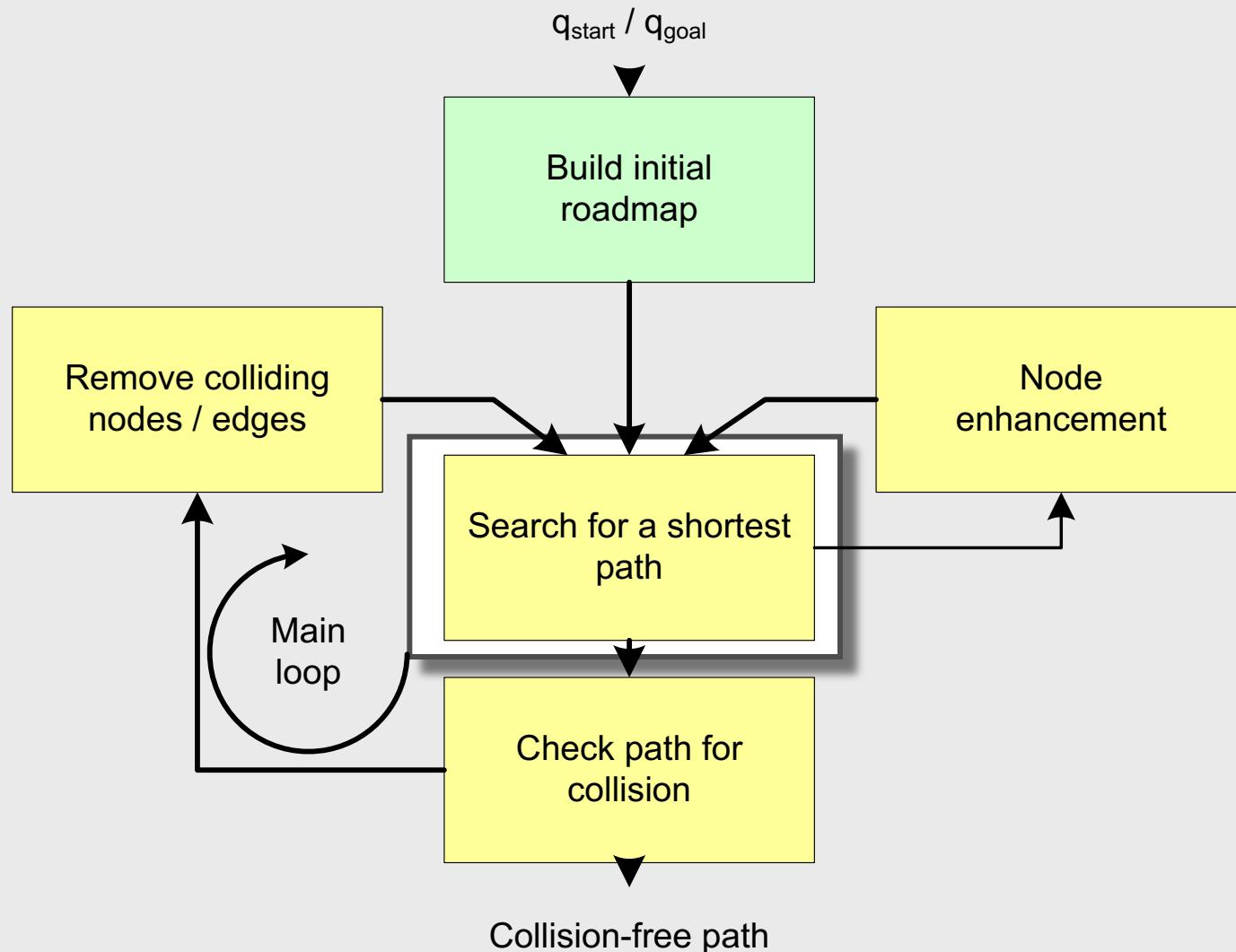


Lazy-PRM: Search again shortest path

- ▶ Search shortest path on computed Roadmap, **don't care about collisions**.
 - ▶ Using A* or Dijkstra
 - ▶ **NO path is found**



Lazy-PRM: The flow.....

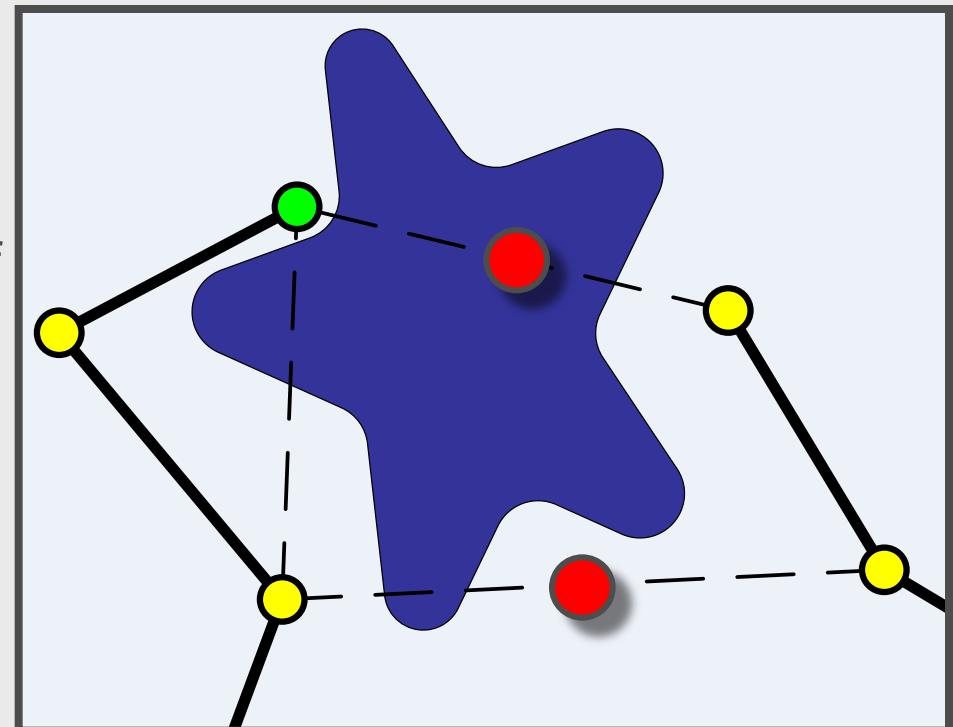


Lazy-PRM: Node enhancement

- ▶ If we don't create new nodes algorithm fails.
- ▶ Strategies:
 - ▶ Create again uniformly distributed nodes, or
 - ▶ Choose special regions to create nodes: **Seed Points, or**
 - ▶ Mix uniformly distributes nodes and nodes generated with seed points.

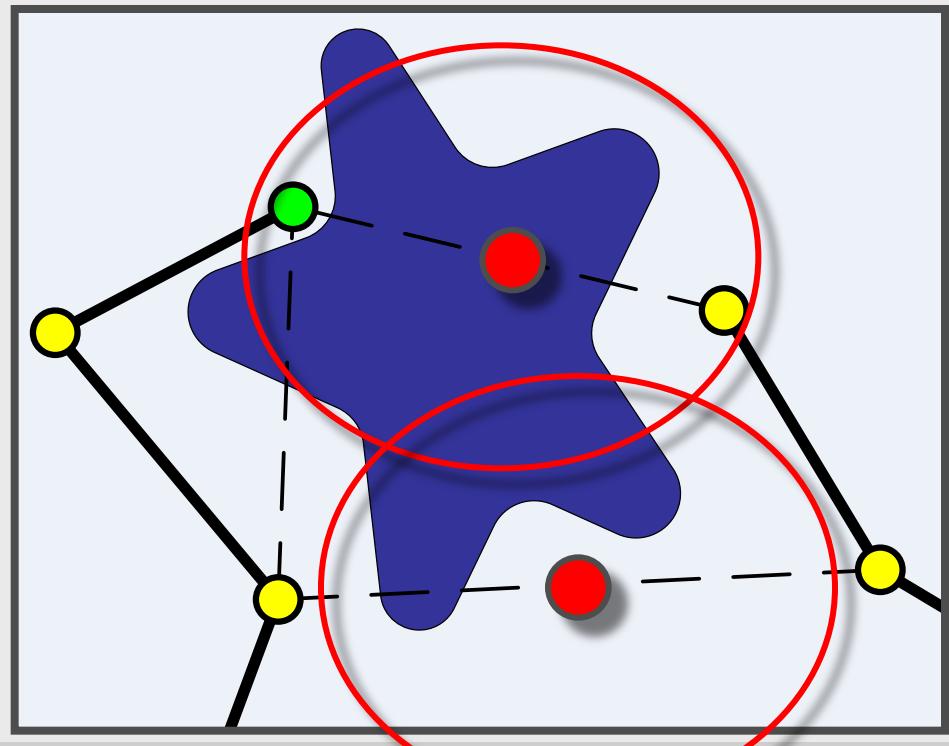
Lazy-PRM: Node enhancement

- ▶ If we don't create new nodes algorithm fails.
- ▶ Strategies:
 - ▶ Create again uniformly distributed nodes, or
 - ▶ Choose special regions to create nodes: **Seed Points**, or
 - ▶ Mix uniformly distributed nodes and nodes generated with seed points.
- ▶ Possible way of generating seed points
 - ▶ Seed points in the middle of colliding nodes



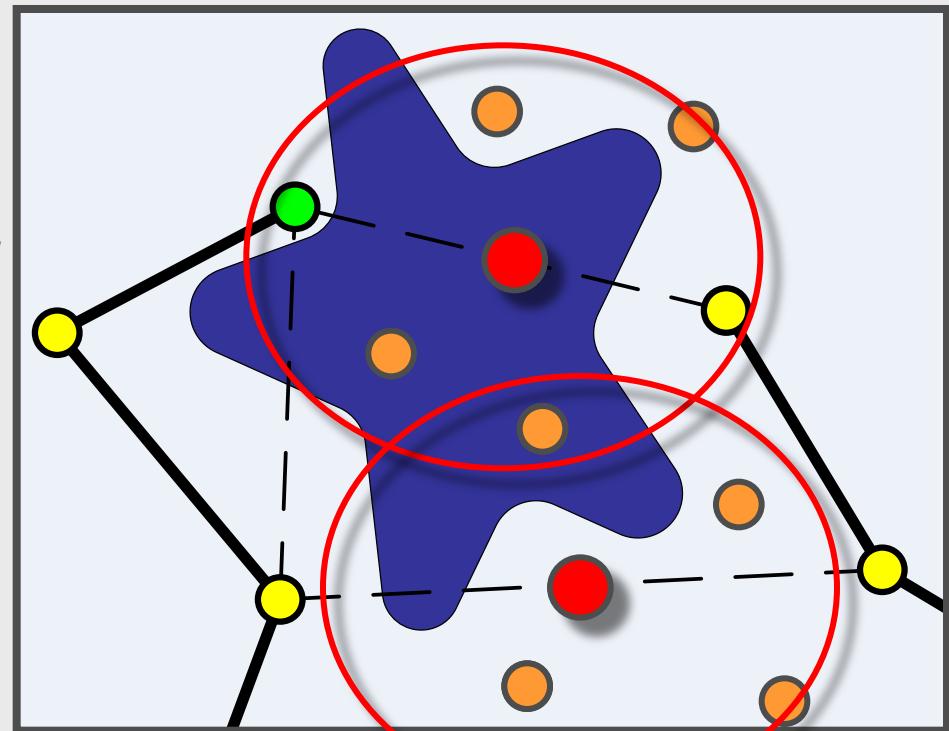
Lazy-PRM: Node enhancement

- ▶ If we don't create new nodes algorithm fails.
- ▶ Strategies:
 - ▶ Create again uniformly distributed nodes, or
 - ▶ Choose special regions to create nodes: **Seed Points**, or
 - ▶ Mix uniformly distributed nodes and nodes generated with seed points.
- ▶ Possible way of generating seed points
 - ▶ Seed points in the middle of colliding nodes
 - ▶ Generate randomly new nodes in a region around seed points



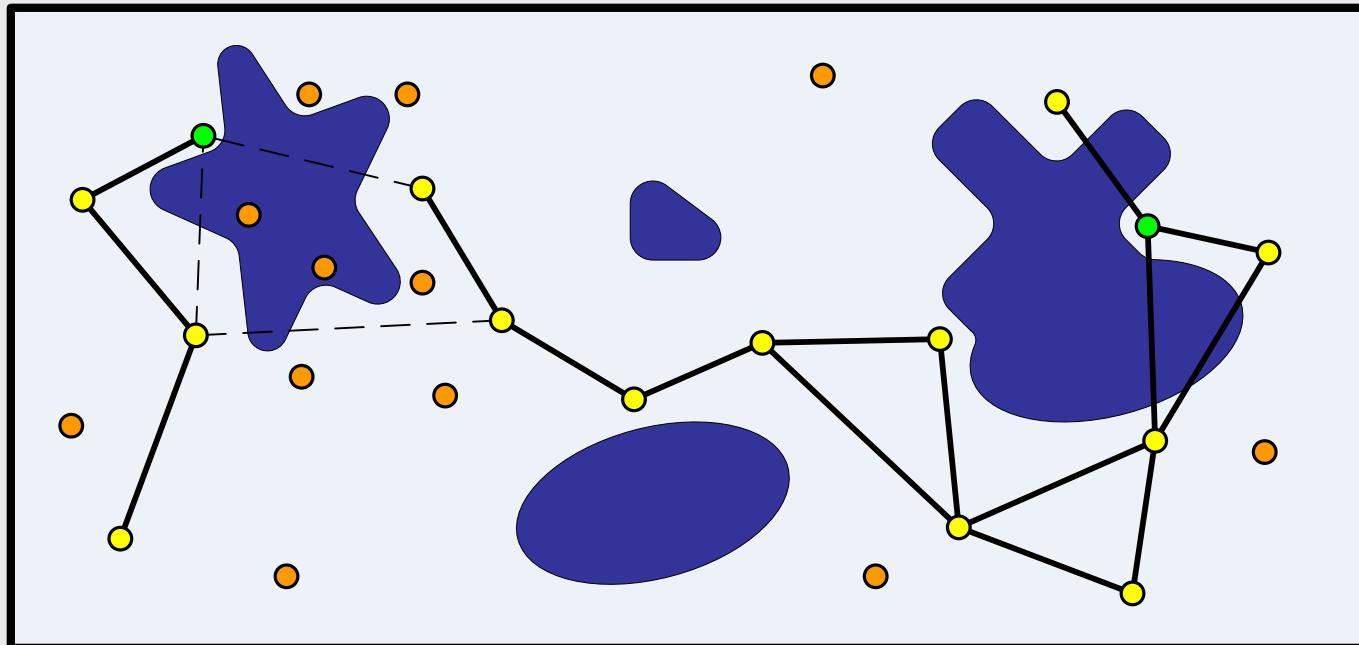
Lazy-PRM: Node enhancement

- ▶ If we don't create new nodes algorithm fails.
- ▶ Strategies:
 - ▶ Create again uniformly distributed nodes, or
 - ▶ Choose special regions to create nodes: **Seed Points**, or
 - ▶ Mix uniformly distributed nodes and nodes generated with seed points.
- ▶ Possible way of generating seed points
 - ▶ Seed points in the middle of colliding nodes
 - ▶ Generate randomly new nodes in a region around seed points



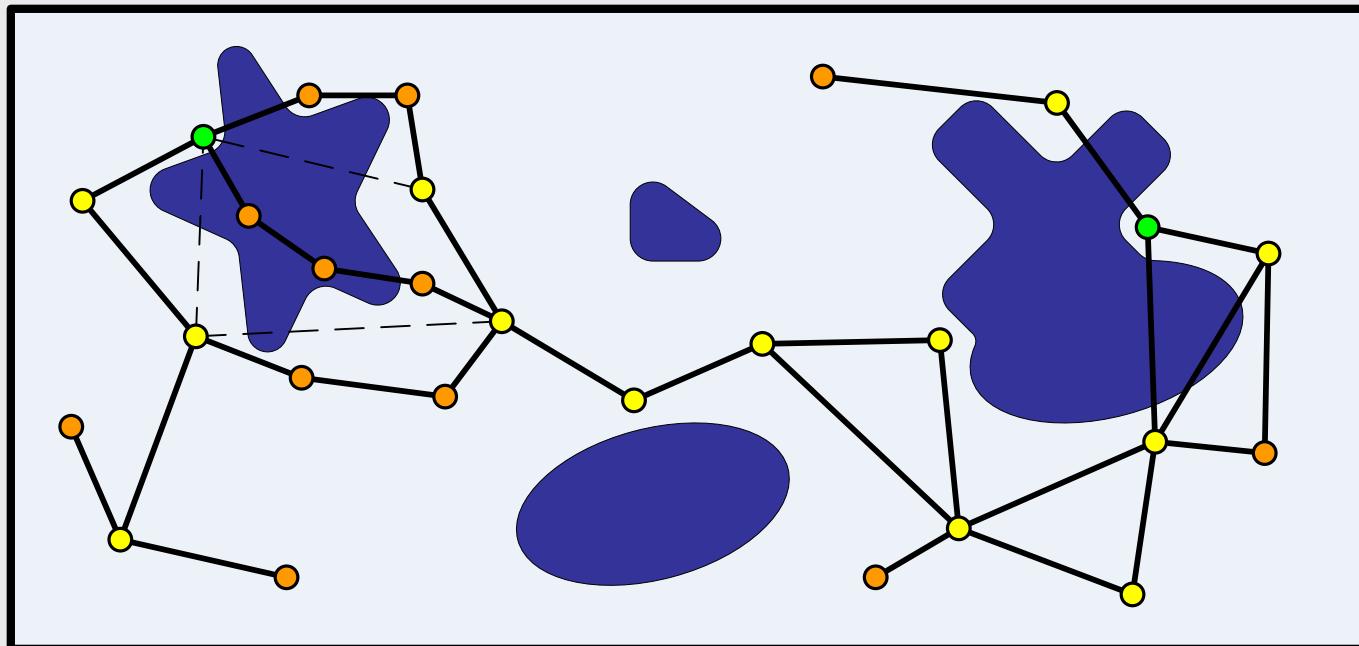
Lazy-PRM: Node enhancement

- ▶ Create nodes based on the combination of seed points and uniformly distributed nodes, **don't care about collisions**

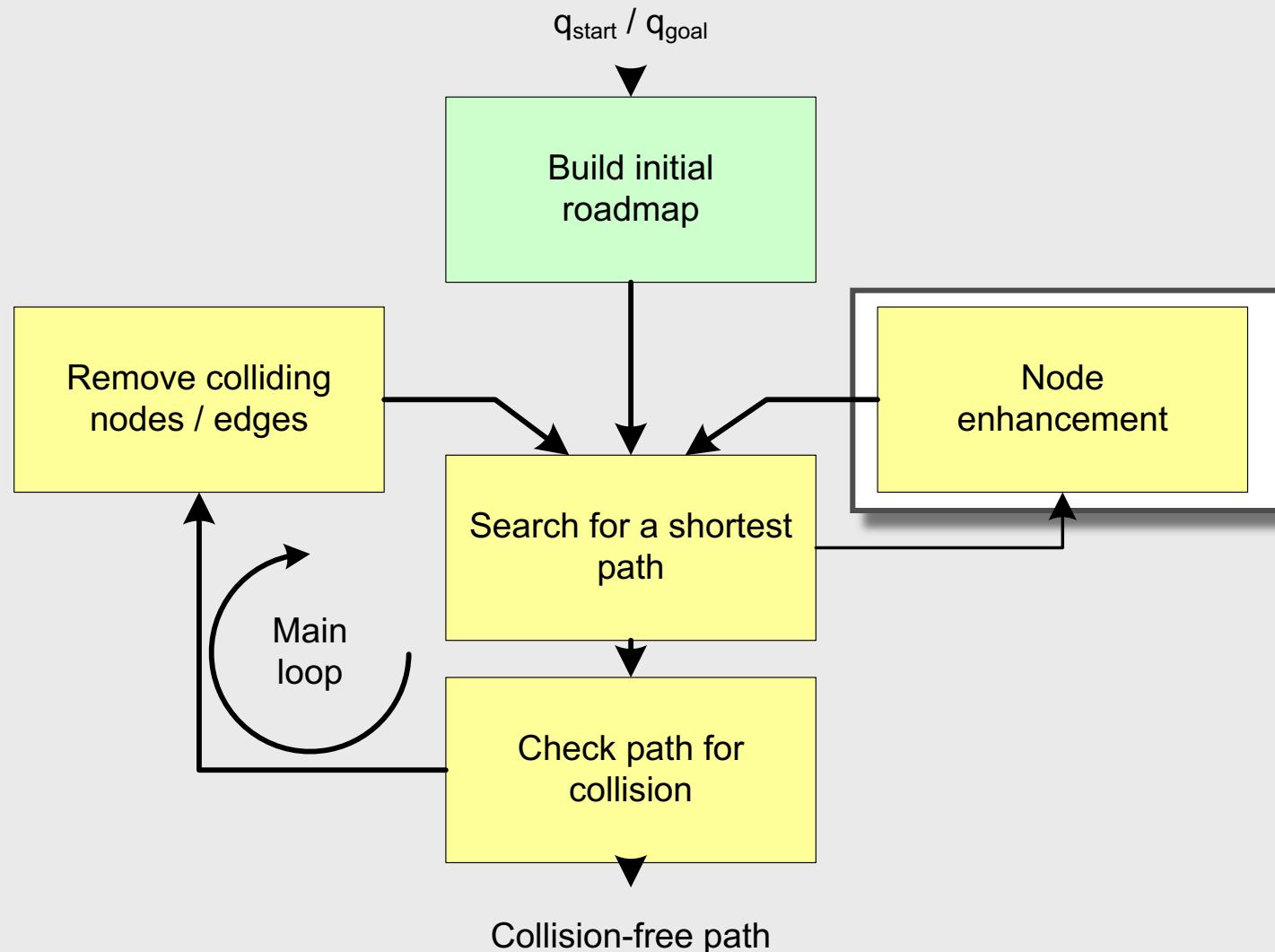


Lazy-PRM: Node enhancement

- ▶ Create nodes based on the combination of seed points and uniformly distributed nodes, **don't care about collisions**
- ▶ Create roadmap (k-closest strategy, in example not all created edges are shown to keep drawing and example simple ☺)

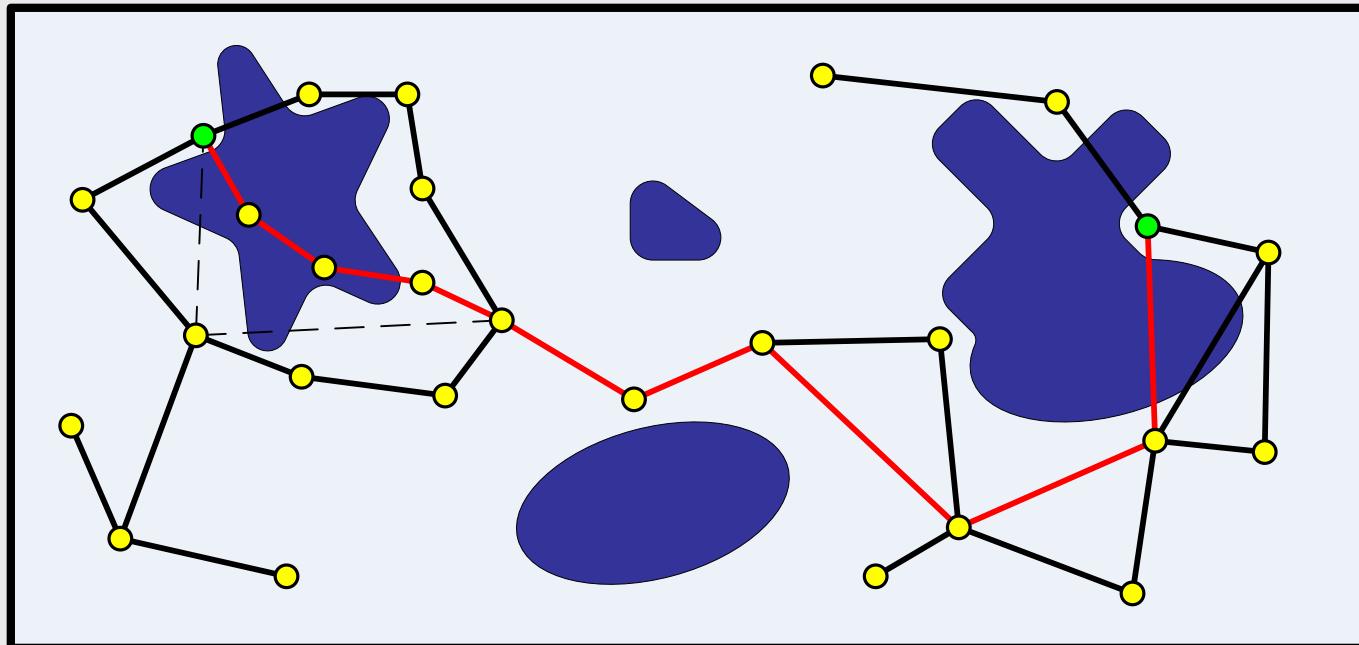


Lazy-PRM: The flow.....



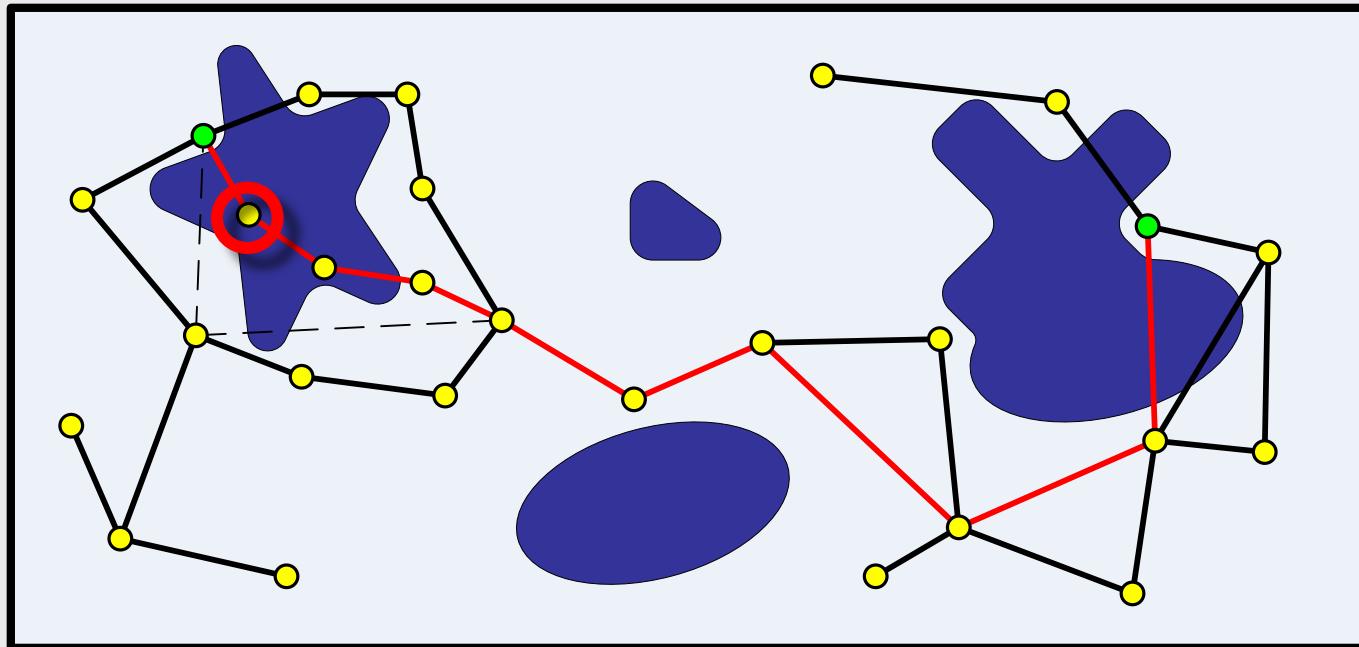
Lazy-PRM: Search again shortest path

- ▶ Search shortest path on computed Roadmap, don't care about collisions.
 - ▶ Using A* or Dijkstra



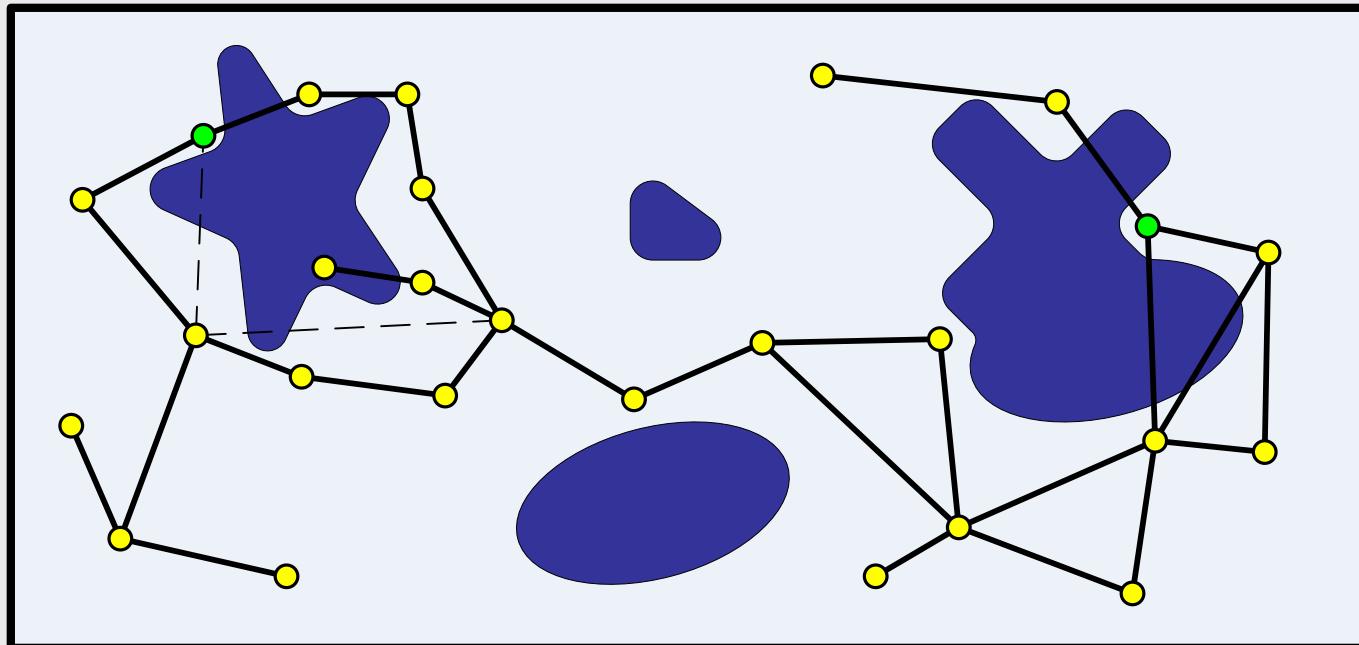
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
 - ▶ Remove colliding node



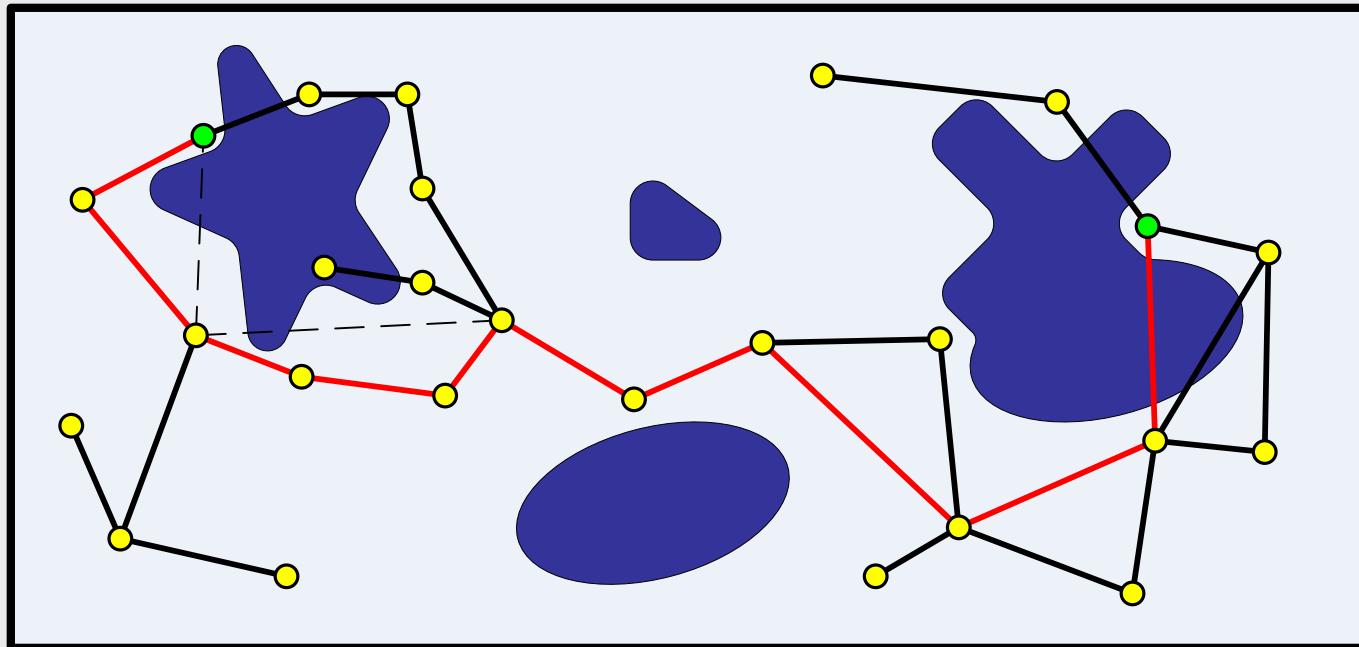
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
 - ▶ Remove colliding node



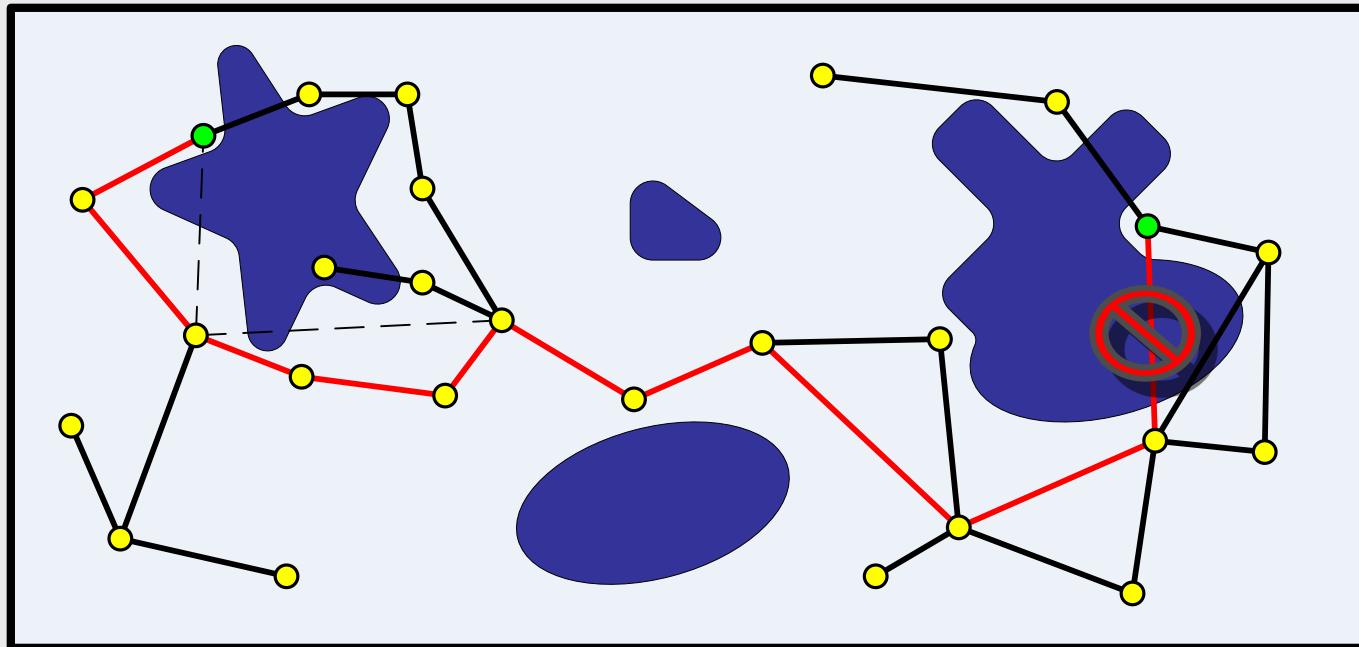
Lazy-PRM: Search again shortest path

- ▶ Search shortest path on computed Roadmap, don't care about collisions.
 - ▶ Using A* or Dijkstra



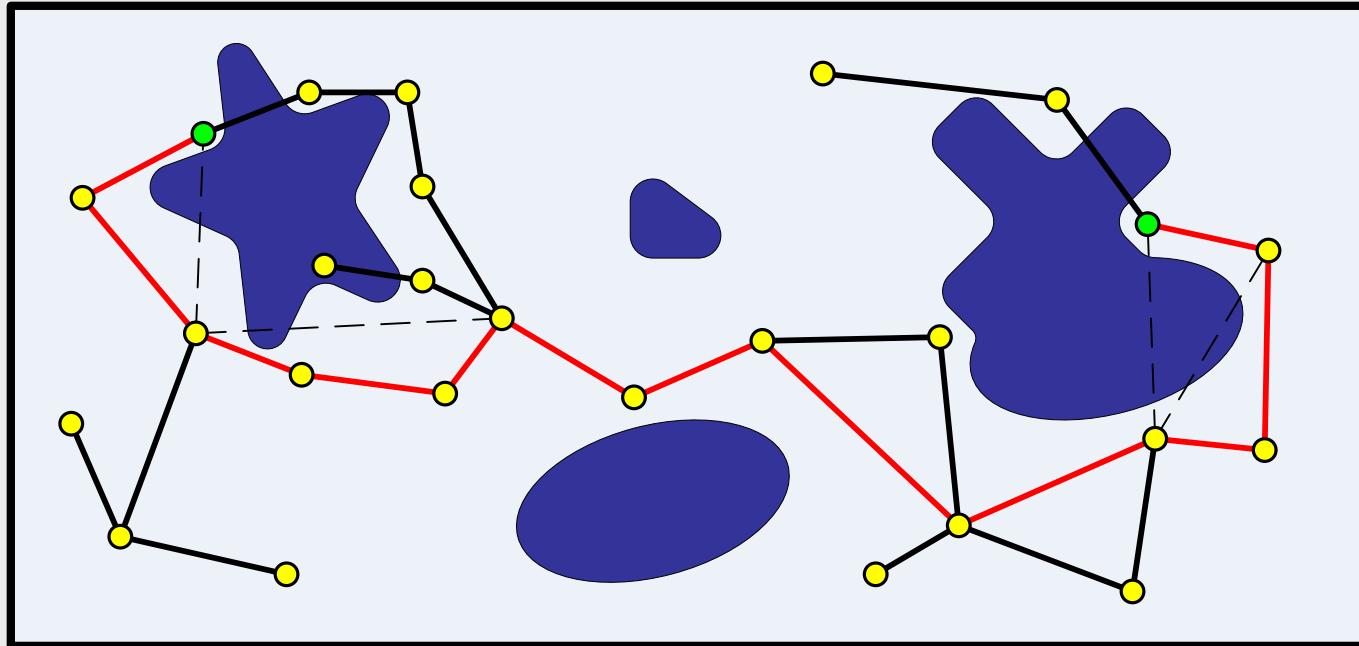
Lazy-PRM: Check nodes /path for collisions

- ▶ Check nodes for collisions
- ▶ Check edges for collisions



Lazy-PRM

► And so on..... 'til path is found!



Lazy-PRM: in short

1. Create roadmap containing start & goal
2. Randomly generate # configurations without testing them for collision and create corresponding nodes in graph
3. Generate (like K-closest-PRM) for every node edges with k-closest neighbors
4. Search a path with A* connecting start and goal
5. Test alternately from start and goal whether **nodes** of found path are collision free
 - ▶ Stop if collision is detected and **remove colliding node**
 - ▶ Stop if alternating search from start and search from goal meets in the middle
6. Test alternately from start and goal whether **edges** of found path are collision free
 - ▶ Use local path planner
 - ▶ If all edges are collision free → path is found
 - ▶ If collision mark edge as colliding and as long as start and goal are connected go back to 4. otherwise go to 7.
7. Generate new points (use some strategy) go to 3.

Lazy-PRM: Generating new nodes

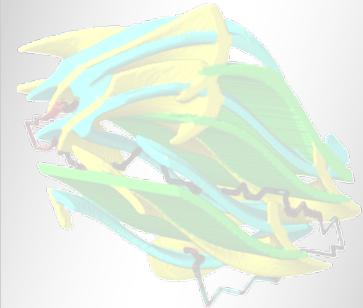
- ▶ If start and goal are not connected anymore due to removal of colliding nodes or edges, new nodes have to be generated.
- ▶ Possible strategies:
 - ▶ Randomly generate nodes (like in 1.) overall in C-space
 - ▶ **More sophisticated way:** using information about colliding regions (**seed points & expansion**)
 - ▶ Regions give information about existing obstacles
 - ▶ New points nearby obstacles can open up new collision free paths around these obstacles
 - ▶ Two steps are to be done
 1. Chosing seed points
 2. Generating new nodes around seed points (**seed expansion**)

Lazy-PRM: Generating nodes during seed expansion

- ▶ Random based expansion
 - ▶ Choose a distribution (e.g. gaussian, normal, ...) and derivation:
 - ▶ Other dependencies (e.g on corresponding link of robot → main axes other derivation than hand axes)
- ▶ Distance based expansion
 - ▶ Generate in free C-space around seed points new nodes
 - ▶ Use free distances as minimal step size for distance of new nodes

Lazy-Variants: SBL

Single-query
Bi-directional
Lazy-collision-checking
(G. Sánchez and J. Latombe)

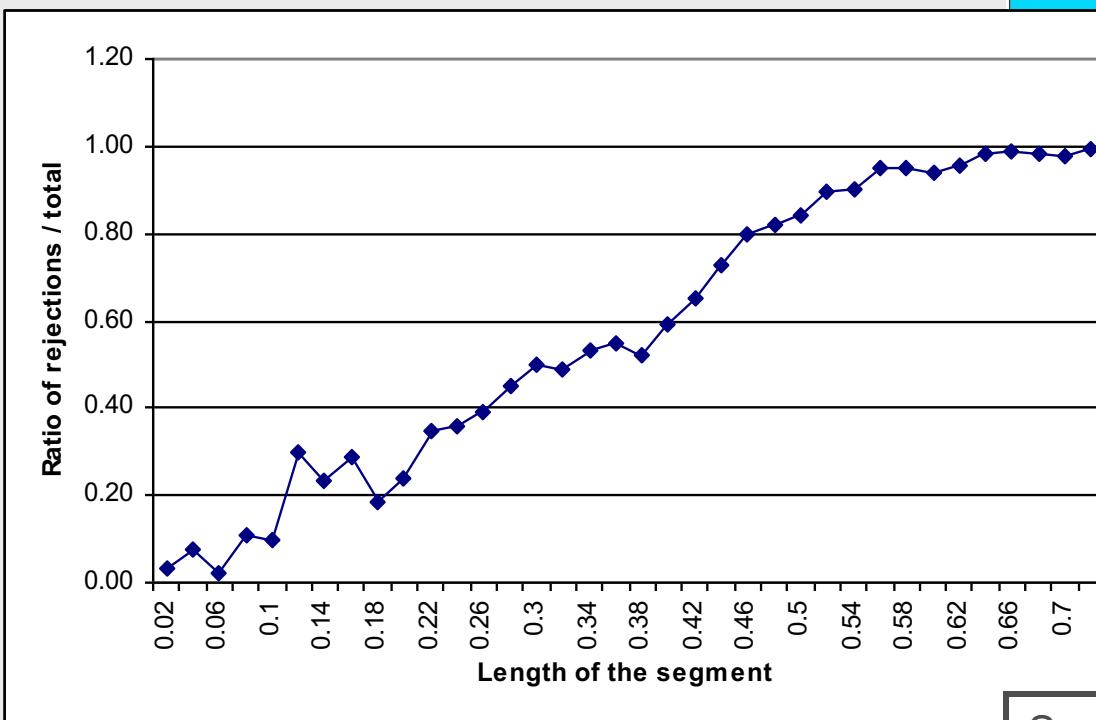
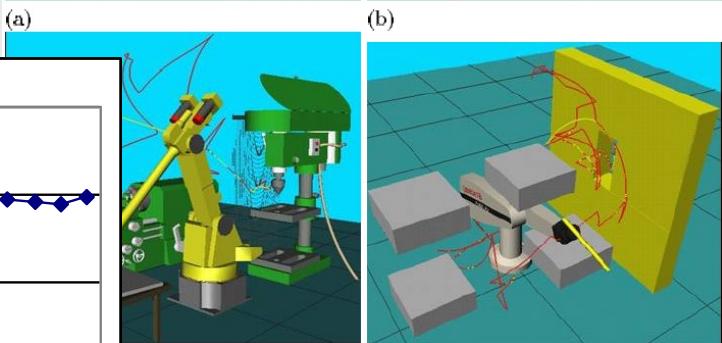
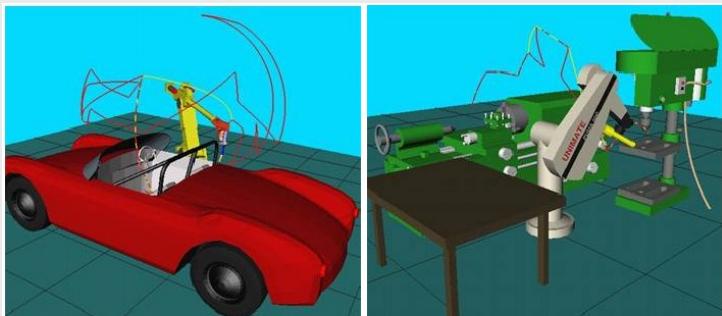


Some issues still open

- ▶ Potential of „lazy collision-checking“ only partially exploited
- ▶ In advance to decide how large network should be
 - ▶ To coarse: it'll fail
 - ▶ To dense: wasting time checking similar paths
- ▶ Focus on shortest paths could be too expensive regarding planning time → it's better to focus on finding fast a collision-free path

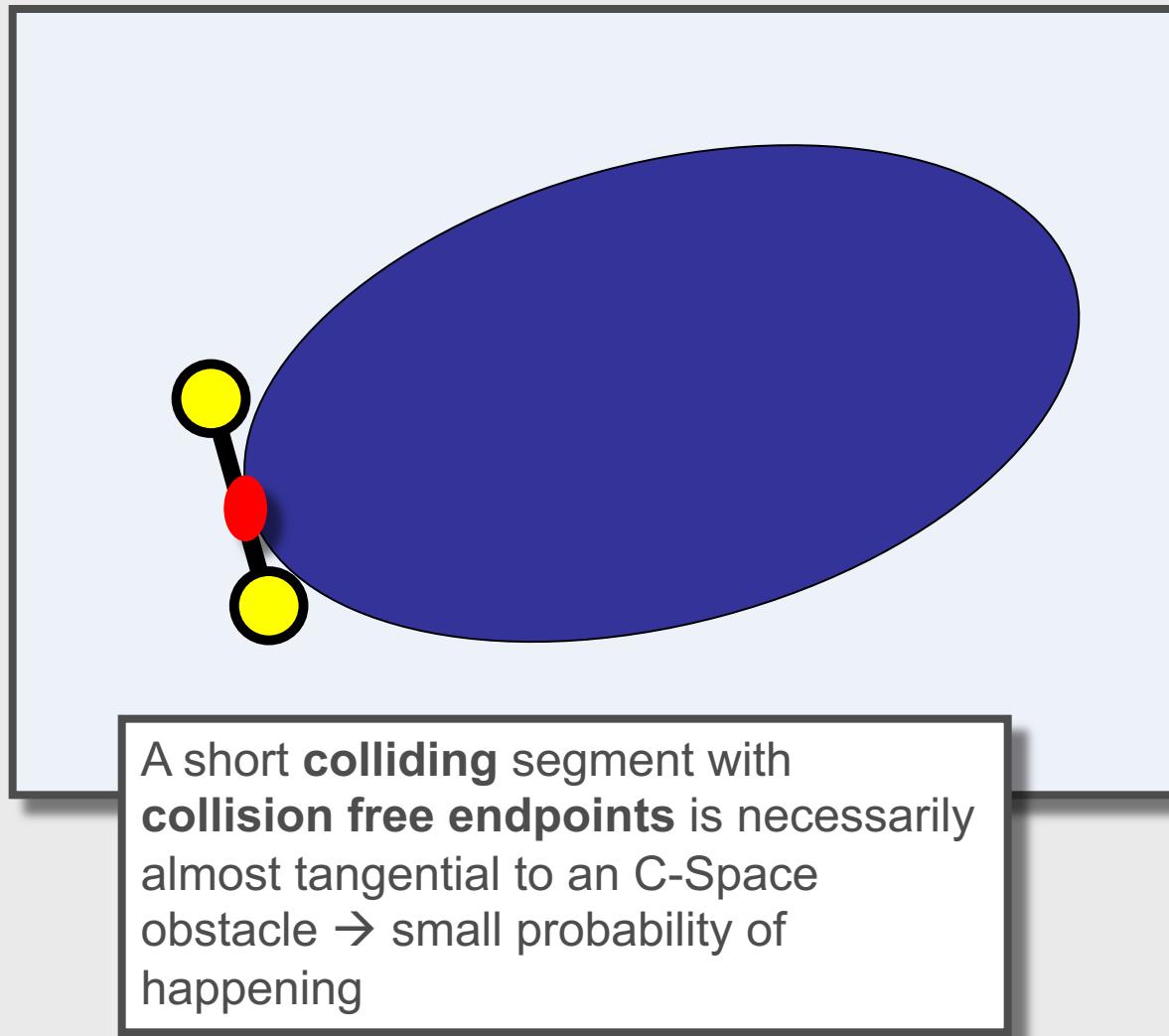
Probability of collision depending on segment length

- ▶ 100 randomly generated **free** configurations
- ▶ Connect these configurations to 100 other randomly generated ones
- ▶ Getting 10000 segments



Source: presentation by Niloy J. Mitra

Why are there few short colliding segments?



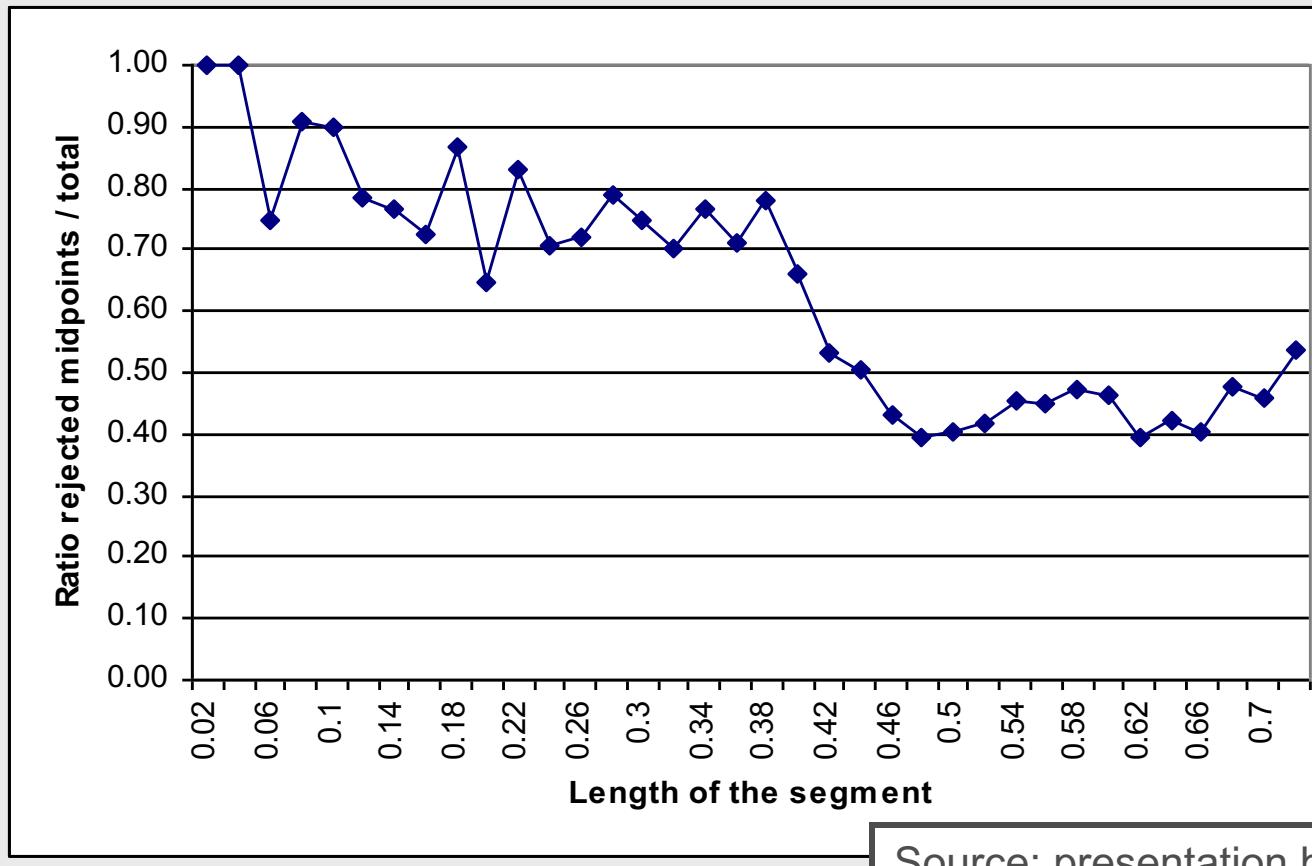
Background: Experimental foundations

Observations when using PRMs:

- ▶ Most local paths are not on the final path
- ▶ Collision-free tests are most expensive
- ▶ Short connections between two milestones have high prior probabilities of being free
- ▶ If a connection is colliding, its midpoint has high probability of being in collision
 - ▶ Next slide...

If a connection is colliding - probability of colliding Midpoint is high

- Adapt therefore line-testing strategy



Source: presentation by Niloy J. Mitra

- ▶ Combining three aspects:
 - ▶ Single-query
 - ▶ Bi-directional
 - ▶ Adaptive sampling strategies
- ▶ Incrementally construct network from **two trees** starting at start & goal
- ▶ Focusing search at regions being reachable from these trees
- ▶ Adapt sampling resolution depending on free space
 - ▶ Big steps in open regions
 - ▶ small steps in narrow passages
- ▶ Connections are not immediately tested for collision
- ▶ Only when a sequence of nodes are joining start and goal the edges are tested for collision

SBL: Overall algorithm

Input:

s - maximal number of milestones that it is allowed to generate
 ρ - distance threshold. Two configurations are considered *close* if their distance is less than ρ .

1. Insert q_{start} and q_{goal} as the roots of T_{start} and T_{goal} respectively
2. Repeat s times
 - a. EXPAND-TREE
 - b. $\tau \leftarrow$ CONNECT-TREE
 - c. If $\tau \neq nil$ then return τ
3. Return *failure*

SBL: Overall algorithm

Input:

s - maximal number of milestones that it is allowed to generate
 ρ - distance threshold. Two configurations are considered *close* if their distance is less than ρ .

1. Insert q_{start} and q_{goal} as the roots of T_{start} and T_{goal} respectively
2. Repeat s times
 - a. EXPAND-TREE
 - b. $\tau \leftarrow \text{CONNECT-TREE}$
 - c. If $\tau \neq \text{nil}$ then return τ
3. Return *failure*

= Add a node to one of the two trees

SBL: Expand Tree

$B(q, r) = \{ q' \in C \mid d(q, q') < r \}$: neighborhood of q of radius r

$\pi(v) \sim 1/\eta(v)$; $\eta(v)$ = Density of nodes around v

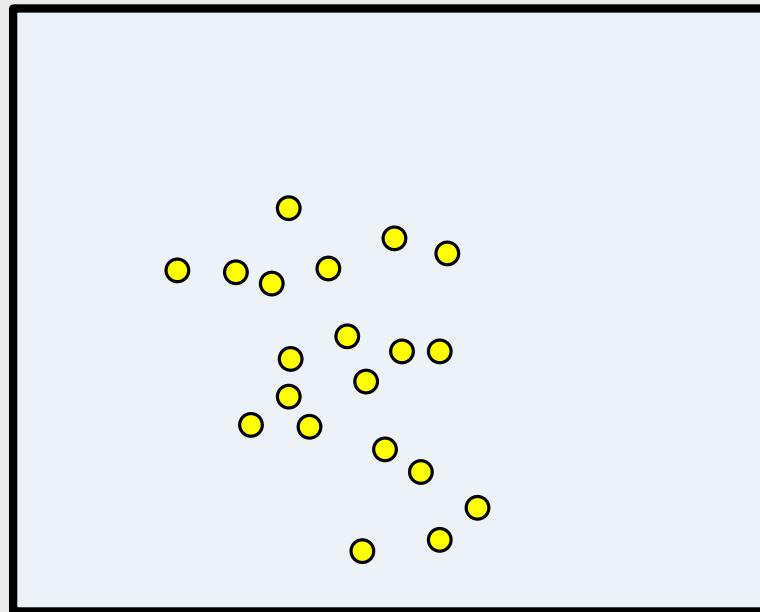
1. Pick T to be either T_{start} or T_{goal} , each with $P=0.5$
2. Repeat until a node is generated
 1. Pick a node v at random, with Probability $\pi(v)$
 2. For $i = 1, 2, \dots$ until a new q been generated
 1. configuration q uniformly at random from $B(v, \rho/i)$
 2. If q is collision-free, then install it as a child of v in T
(Note: collision-test of segment is not done here)

How to compute „density“ in a fast way

- ▶ Create two arrays A_{start} and A_{goal} ($\text{DIM} = 2$ or 3)
- ▶ Both arrays partition space in equally sized cells
- ▶ When a new node is added to a tree T_{start} or T_{goal} , corresponding $A_{\text{start}}/A_{\text{goal}}$ is updated.
- ▶ Choose node depending on density:
 - ▶ Pick a non-empty cell of A_{xxx} with probability depending on number of nodes in cell
 - ▶ Pick a node of this cell

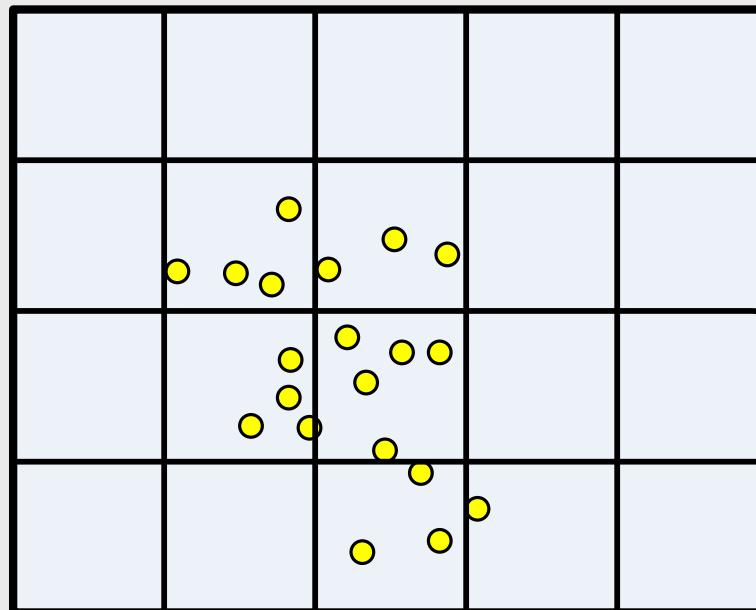
Idea behind cell-density

- ▶ Don't maintain density around each node → overhead



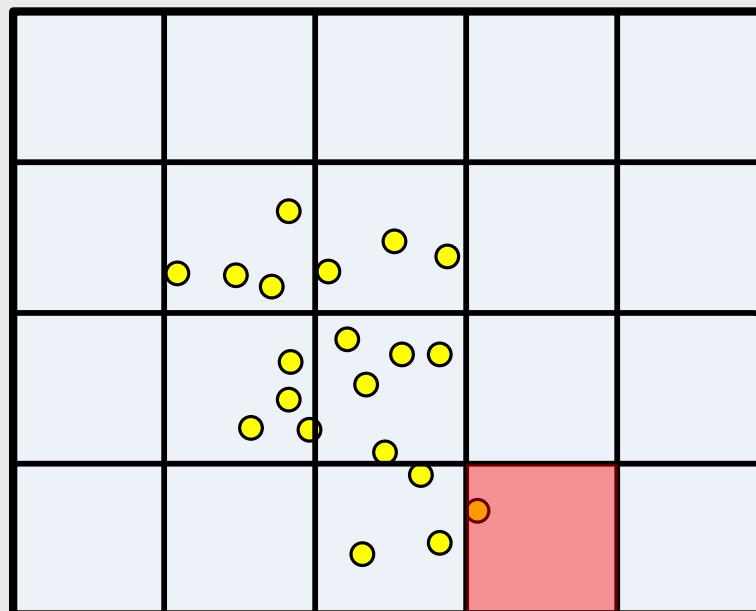
Idea behind cell-density

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally sized cells and maintain density in these cells



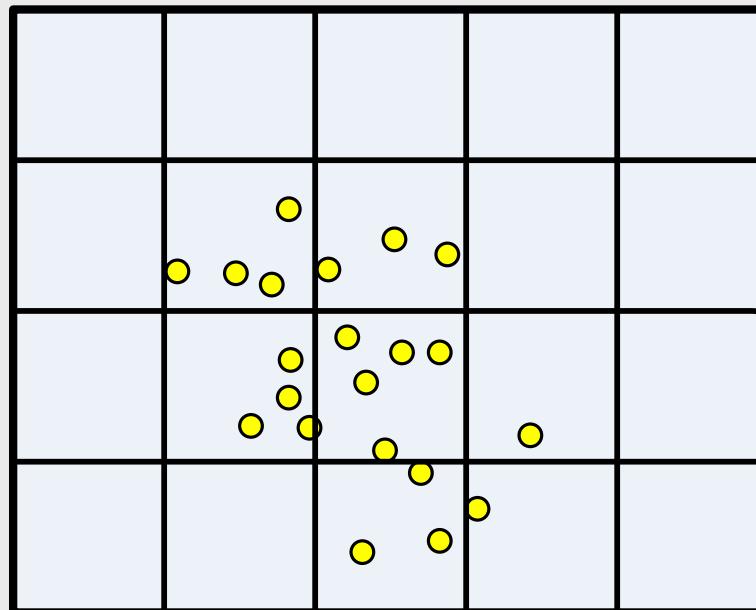
Idea behind cell-density

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally sized cells and maintain density in these cells
- ▶ Choose cell having lowest density with highest probability



Idea behind cell-density

- ▶ Don't maintain density around each node → overhead
- ▶ Use equally sized cells and maintain density in these cells
- ▶ Choose cell having lowest density with highest probability
- ▶ Generate new node



SBL: Overall algorithm

Input:

s - maximal number of milestones that it is allowed to generate
 ρ - distance threshold. Two configurations are considered *close* if their distance is less than ρ .

1. Insert q_{start} and q_{goal} as the roots of T_{start} and T_{goal} respectively
2. Repeat s times
 - a. EXPAND-TREE
 - b. $\tau \leftarrow \text{CONNECT-TREE}$ = Connect the two trees
 - c. If $\tau \neq \text{nil}$ then return τ
3. Return *failure*

SBL: Connect Tree

1. $v \leftarrow$ most recently created node
2. $v' \leftarrow$ closest node to v in the other tree
3. If $d(v, v') < \rho$ then
 1. Connect v and v' by a bridge w
 2. $\tau \leftarrow$ path connecting q_{start} and q_{goal}
 3. Return TEST-PATH
4. Return nil

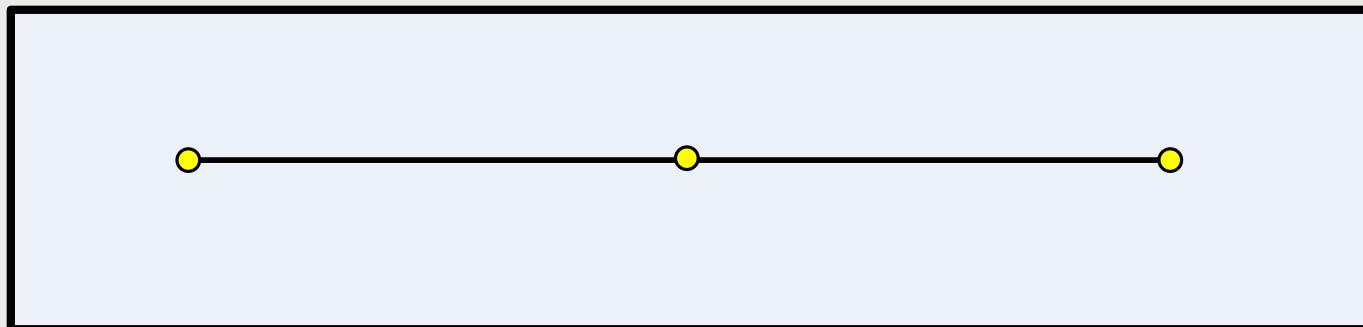
SBL: Path testing

- ▶ SBL uses collision-check index $\kappa(u)$ rating the amount of testing done on an edge u
- ▶ $\kappa(u) = 0 \rightarrow$ only two endpoints of u are tested



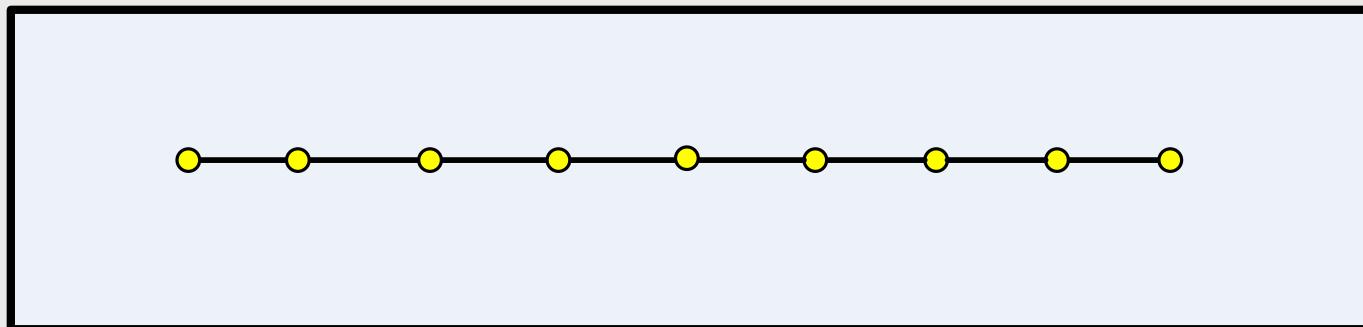
SBL: Path testing

- ▶ SBL uses collision-check index $\kappa(u)$ rating the amount of testing done on an edge u
- ▶ $\kappa(u) = 0 \rightarrow$ only two endpoints of u are tested
- ▶ $\kappa(u) = 1 \rightarrow$ two endpoints and midpoint are tested



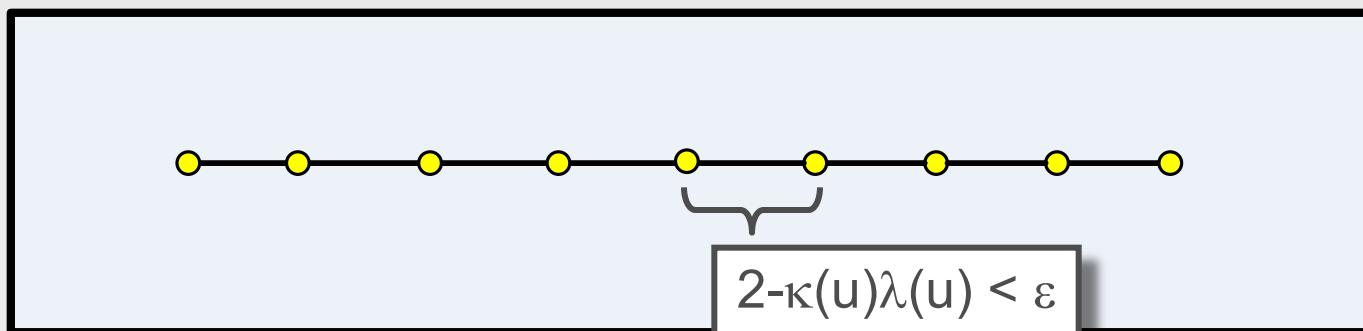
SBL: Path testing

- ▶ SBL uses collision-check index $\kappa(u)$ rating the amount of testing done on an edge u
- ▶ $\kappa(u) = 0 \rightarrow$ only two endpoints of u are tested
- ▶ $\kappa(u) = 1 \rightarrow$ two endpoints and midpoint are tested
- ▶ For a given $\kappa(u)$, $2^{\kappa(u)}+1$ equally distant points are tested and are collision-free



SBL: Path testing

- ▶ SBL uses collision-check index $\kappa(u)$ rating the amount of testing done on an edge u
- ▶ $\kappa(u) = 0 \rightarrow$ only two endpoints of u are tested
- ▶ $\kappa(u) = 1 \rightarrow$ two endpoints and midpoint are tested
- ▶ For a given $\kappa(u)$, $2^{\kappa(u)}+1$ equally distant points are tested and are collision-free
- ▶ If $\lambda(u) = \text{length}(u) \rightarrow u$ is save if $2^{-\kappa(u)}\lambda(u) < \varepsilon$



SBL: Test edge

- ▶ Let $\sigma(u,j)$ be the set of points along $beeing$ already tested collision-free in order for $\kappa(u)$ to have value j
- ▶ Algorithm Test-Segments increase $\kappa(u)$ by one

1. $j \leftarrow \kappa(u)$

SBL: Test edge

- ▶ Let $\sigma(u,j)$ be the set of points along beeing already tested collision-free in order for $\kappa(u)$ to have value j
- ▶ Algorithm Test-Segments increase $\kappa(u)$ by one

1. $j \leftarrow \kappa(u)$
2. **for every** $q \in \sigma(u,j+1) \setminus \sigma(u,j)$ **if** q **is in collision**
return *collision*



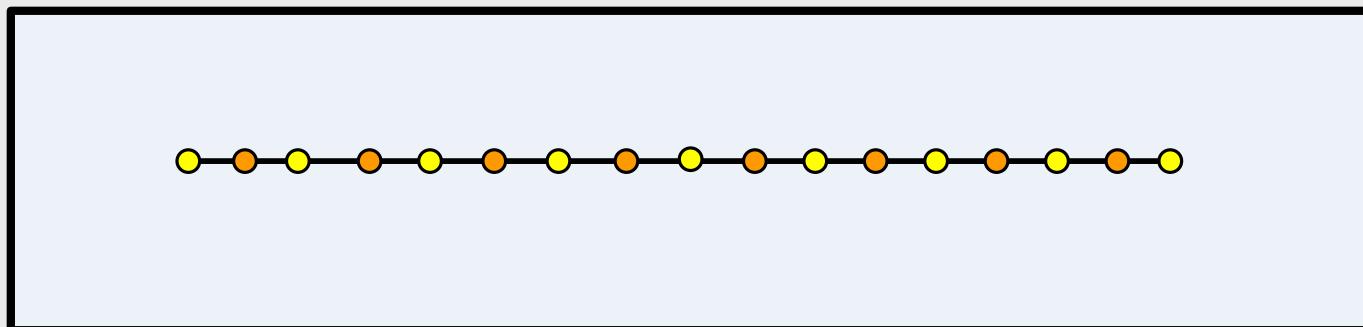
Only orange vertices are tested : $\sigma(u,j+1) \setminus \sigma(u,j)$

SBL: Test edge

- ▶ Let $\sigma(u,j)$ be the set of points along beeing already tested collision-free in order for $\kappa(u)$ to have value j
- ▶ Algorithm Test-Segments increase $\kappa(u)$ by one

1. $j \leftarrow \kappa(u)$
2. **for every** $q \in \sigma(u,j+1) \setminus \sigma(u,j)$ **if** q **is in collision**
return *collision*
3. **if** $2^{-(j+1)} \lambda(u) < \varepsilon$ **mark** u **as safe**
else $\kappa(u) \leftarrow j + 1$

For every edge not beeing safe, the value $2^{-\kappa(u)}\lambda(u)$ is cached.



SBL: Test Path

- ▶ u_1, u_2, \dots, u_p denote all edges in path that are not marked save
- ▶ U is a priority queue of these edges sorting by decreasing order of $2^{-\kappa(u_i)} \lambda(u_i)$ ($i = 1$ to p).

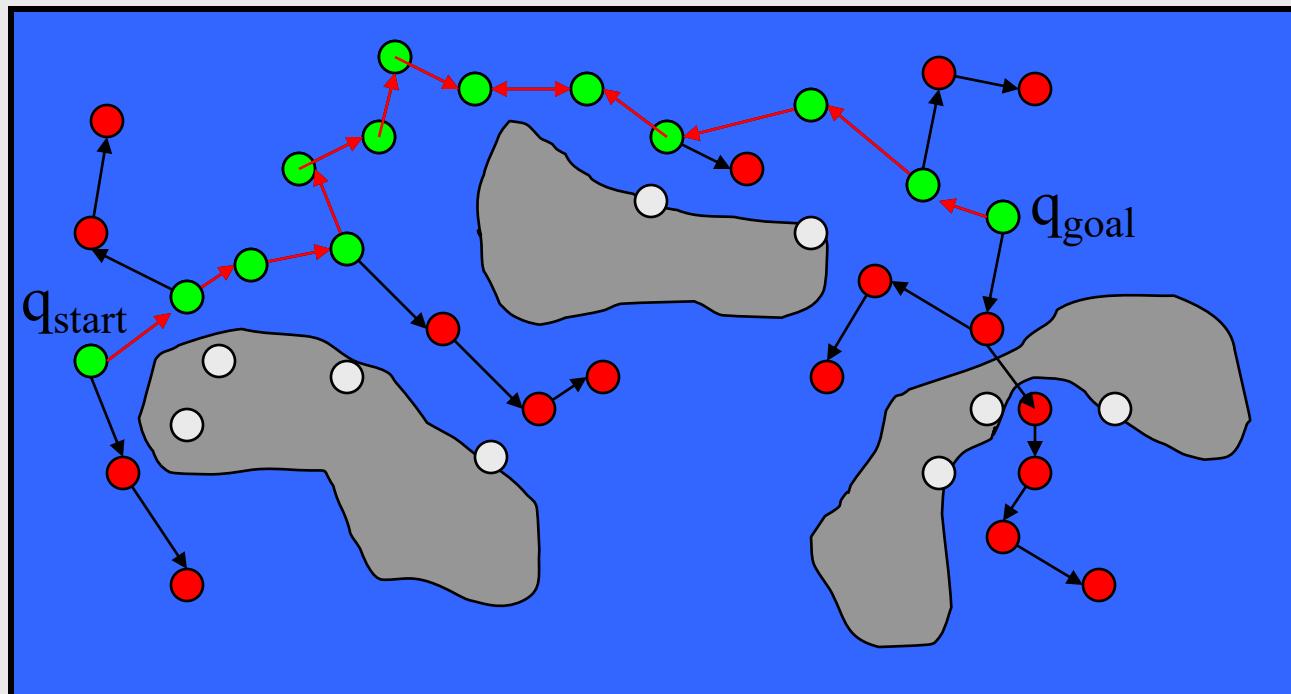
TEST-PATH (τ)

PriorityQueue U ;

1. While U is not empty do
 1. $u \leftarrow \text{extract}(U)$
 2. If TEST-SEGMENT (u) = *collision* then
 1. Remove u from the roadmap
 2. Return *nil*
 3. If u is not marked *safe* then re-insert u into U
2. Return τ

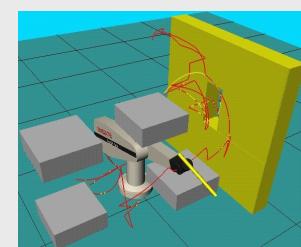
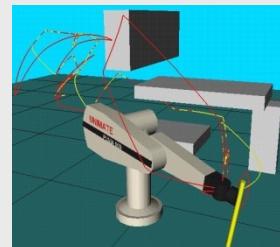
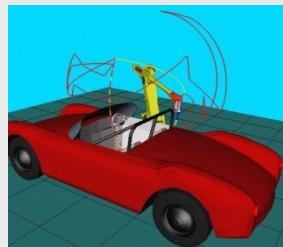
SBL: Example

- ▶ Example search
 - ▶ Two search trees are



Source: presentation by Niloy J. Mitra

Performance – BOOST.....



Average performance with lazy collision checking

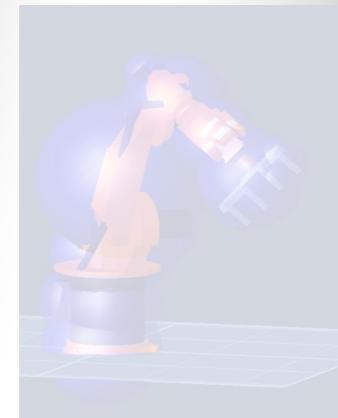
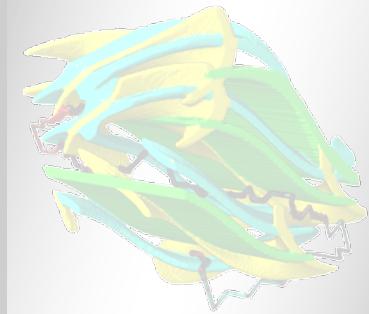
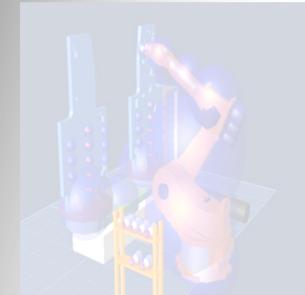
Example	Running Time(secs)	Milestones in Roadmap	Milestones in Path	Total Nr of Collision Checks	Collision Checks on the Path	Sampled Milestones	Comput. Time for Coll-Check (secs)	Std. Deviation for running time
1a	0.60	159	13	1483	342	245	0.58	0.38
1b	4.45	1609	39	11211	411	7832	4.21	2.48
1c	4.42	1405	24	7267	277	3769	4.17	1.86
1d	0.17	33	10	406	124	47	0.17	0.07
1e	6.99	4160	44	12228	447	6990	6.30	3.55

Average performance without lazy collision checking

Example	Running Time(secs)	Milestones in Roadmap	Milestones in Path	Total Nr of Collision Checks	Collision Checks on the Path	Sampled Milestones	Comput. Time for Coll-Check (secs)	Std. Deviation for running time
1a	2.82	22	5	7425	173	83	2.81	3.01
1b	106.20	3388	32	300060	421	9504	105.56	59.30
1c	18.46	771	16	38975	219	3793	18.35	15.34
1d	1.03	29	9	2440	123	46	1.02	0.70
1e	293.77	6737	24	666084	300	11971	292.40	122.75

Source: presentation by Niloy J. Mitra

A* - Algorithm for path planning

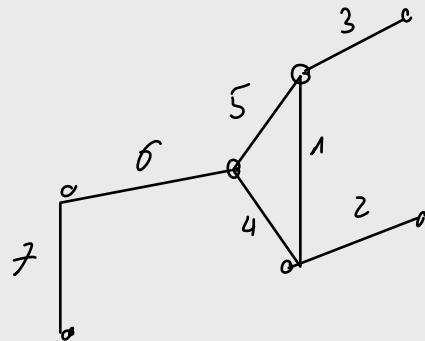


Discretisation of C-space

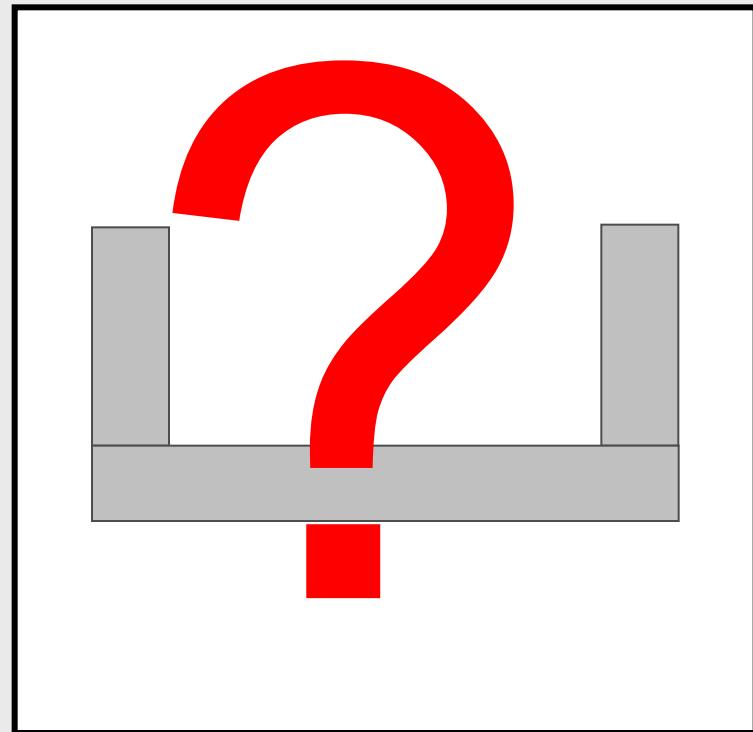
- To use an A*-algorithm as path planning algorithm, C-space must be discretized.

im Grundgedanken auf Graphen

A* um Pfad auf Graphen zu finden



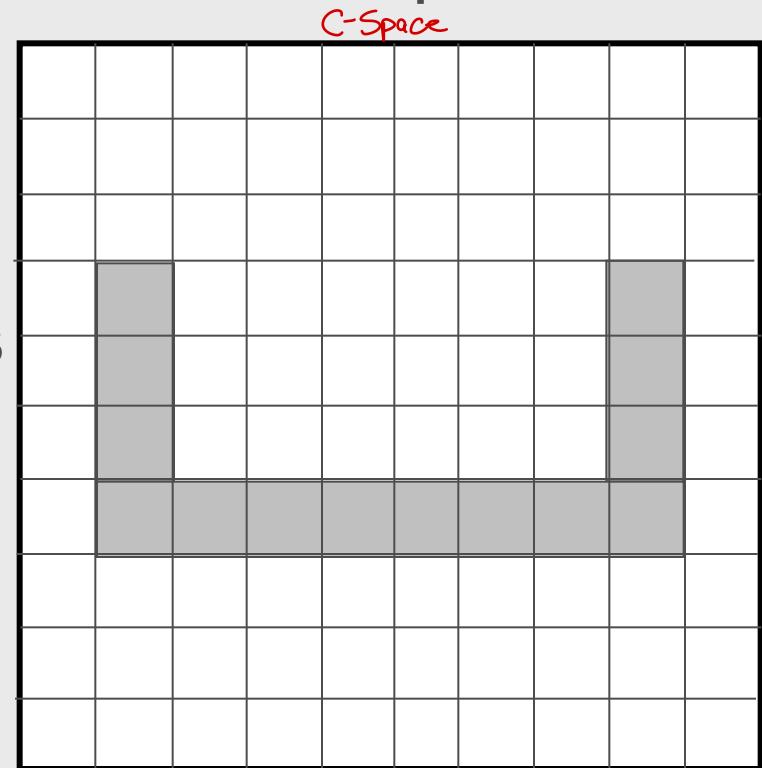
Graph hat keinen geometrischen Bezug → kann nur räuml.
zugeordnet werden
ursprünglich



Discretisation of C-space

- ▶ To use an A*-algorithm as path planning algorithm, C-space must be discretized.
- ▶ After discretisation C-space can be interpreted as a grid on which A* can „take place“
- ▶ The joint values are transformed to **grid coordinates** based on joint limits and numbers of discretisation steps

Auf n-Dimensionen erweiterbar



jedem Knoten exakte Position zugewiesen

Discretisation of C-space: grid coordinates

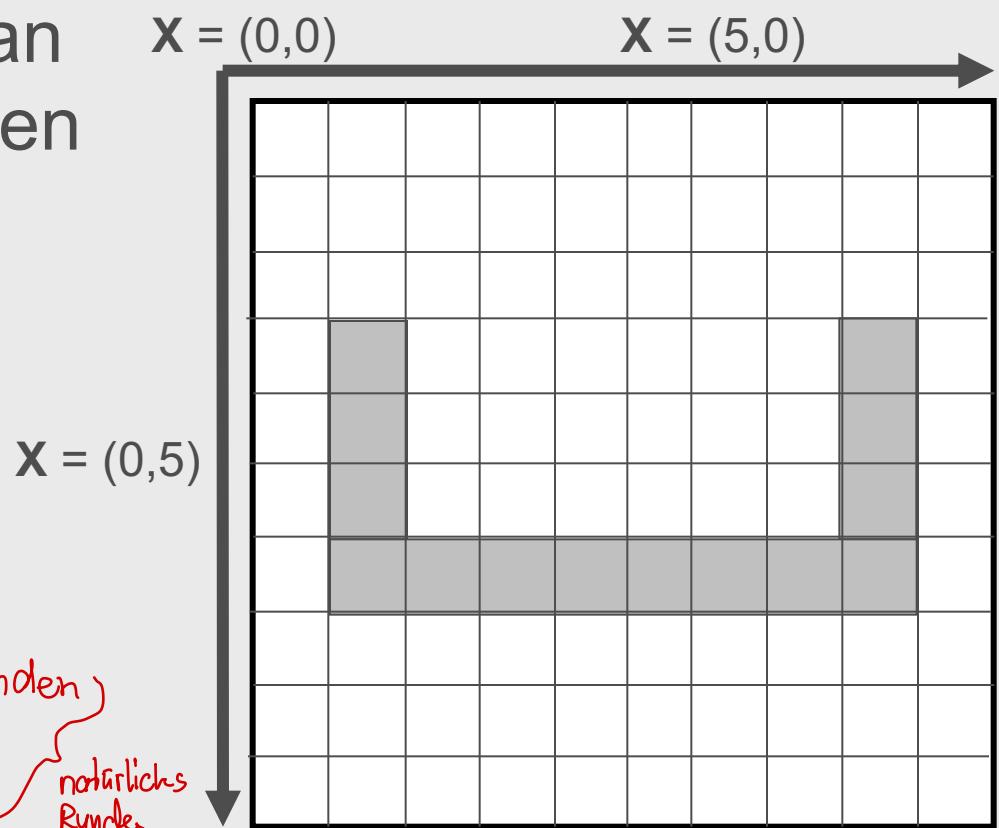
- ▶ Every grid coordinate corresponds exactly to one configuration
- ▶ Let $\Delta\Theta_i$ the chosen discretisation and $\Theta_{\min,i}$ the lower limit of joint i, then φ_i can be determined by a given grid coordinate x_i :

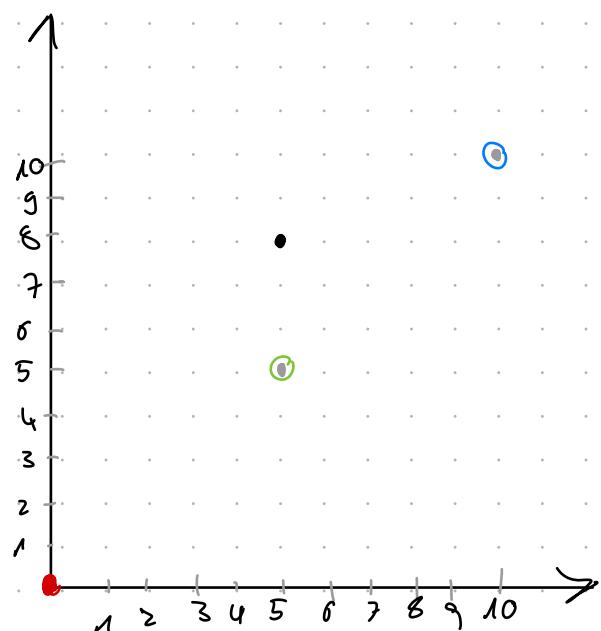
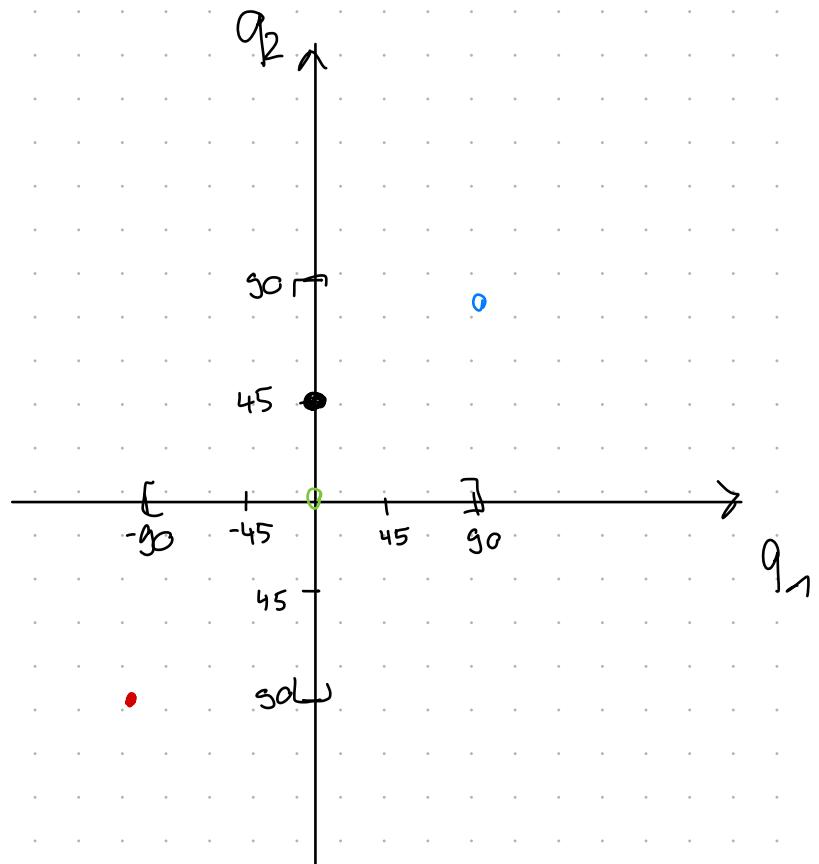
$$\varphi_i = \Theta_{\min,i} + \Delta\Theta_i x_i$$

- ▶ Other way around:

$$x_i = \left\lfloor \frac{\varphi_i - \Theta_{\min,i}}{\Delta\Theta_i} + \frac{1}{2} \right\rfloor$$

Abrunden
min. auf nächsten Knoten
natürliches Runden
(ab ..., 5 wird aufgerundet)





Gitter lässt keine Dezimalwerte im Gitter zu.

A* algorithm in more detail

```
procedure A* (S;G)
tNode : S;                                /* INPUT: Start node */
tNode : G;                                /* INPUT: Goal node */
begin
    tNode :     n;                         /* Temporary node */
    tNodeList : s;                         /* Successor node list */
    tHeap :     OPEN;                      /* Nodes to be expanded */
    tHashtable : CLOSED;                  /* Nodes already expanded */
    tBoolean :   goalFound;                /* TRUE, if Goal is found */
    init (OPEN;CLOSED; S) ;
    goalFound = FALSE;
    while (OPEN ≠ Ø) do begin
        n = best (OPEN) ;
        if (n == G) then
            goalFound = TRUE;
            break;
        endif
        s = expandNode (n) ;
        handleSuccessors (OPEN;CLOSED; s) ;
    end
    if (goalFound == TRUE) then
        printSolution (: : : : ) ;
    endif
    return (goalFound) ;
end;
```

nach zu untersuchen

schon besucht

Heap → nach priorisierte sortierte Liste

↳ nächstw Wert ist idr. der Beste Wert nach bestimmten Kriterium

Hashtable → Feld / Array → sehr schnell auf speziellen Wert zugreif

↳ Python Beispiel Dictionary

A* algorithm in more detail

```
procedure bestfirst (S;G)
tNode : S;                                /* INPUT: Start node */
tNode : G;                                /* INPUT: Goal node */
begin
    tNode :     n;                         /* Temporary node */
    tNodeList : s;                         /* Successor node list */
    tHeap :    OPEN;                        /* Nodes to be expanded */
    tHashtable : CLOSED;                   /* Nodes already expanded */
    tBoolean : goalFound;                 /* TRUE, if Goal is found */
init (OPEN;CLOSED; S) ;
goalFound = FALSE;
while (OPEN ≠ Ø) do begin
    n = best (OPEN) ;
    if (n == G) then
        goalFound = TRUE;
        break;
    endif
    s = expandNode (n) ;
    handleSuccessors (OPEN;CLOSED; s) ;
end
if (goalFound == TRUE) then
    printSolution (: : : : ) ;
endif
return (goalFound) ;
end;
```

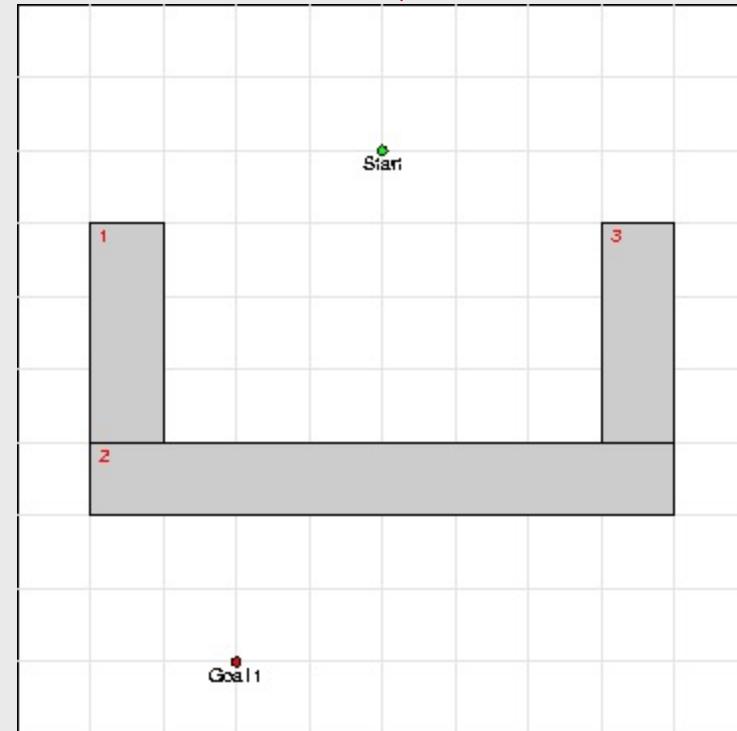
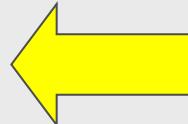
Place to modify
algorithm

A*-search

► Graph search (A*) in implicit C-space

```
init (OPEN;CLOSED; S) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```

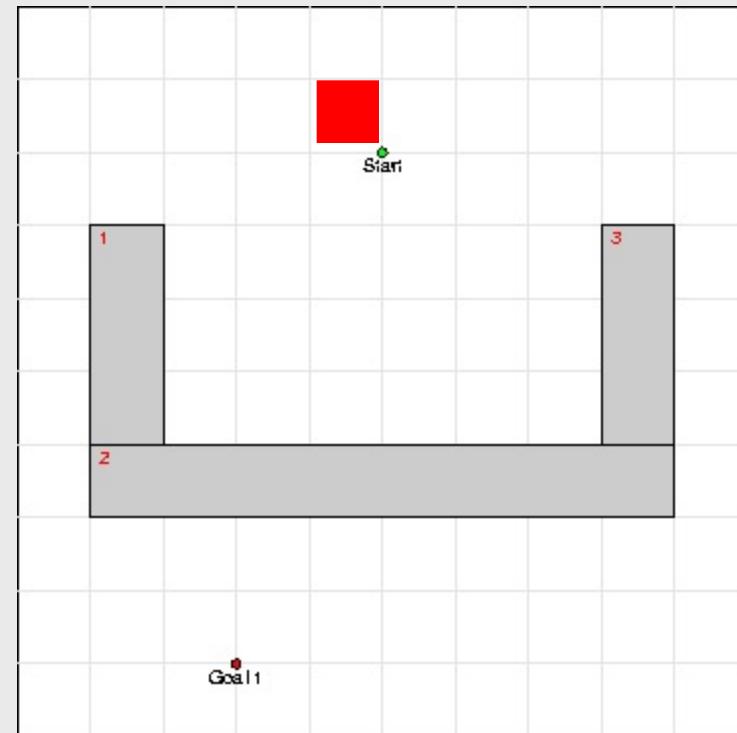
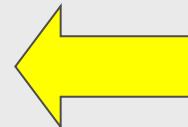
Start in open Liste (eineriger Punkt)
besitzt Knoten < Start \Rightarrow + Ziel \rightarrow Knoten wird expandiert



A*-search

- ▶ Graph search (A*) in implicit C-space

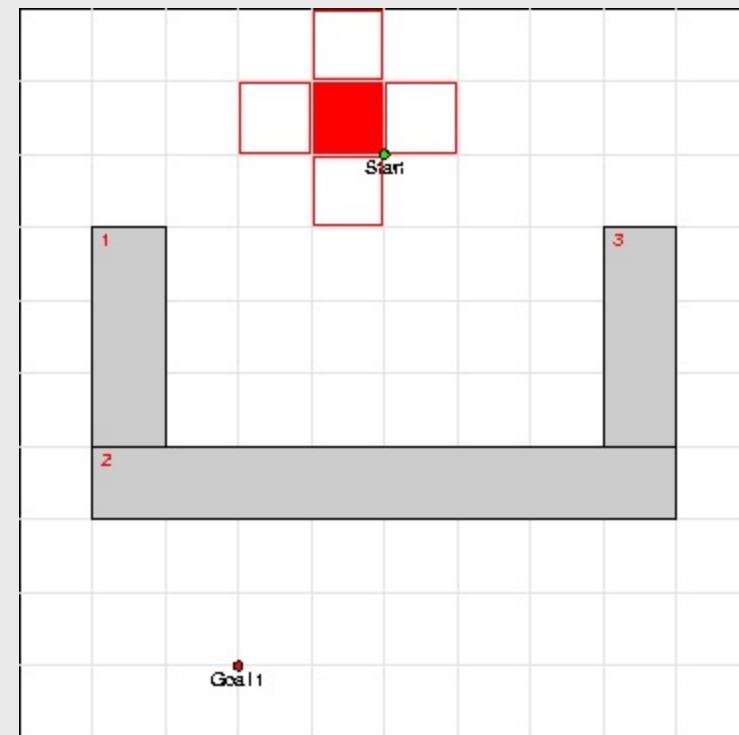
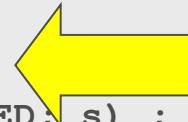
```
init (OPEN;CLOSED; S) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```



A*-search

- ▶ Graph search (A*) in implicit C-space

```
init (OPEN;CLOSED; S) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```



A*-search

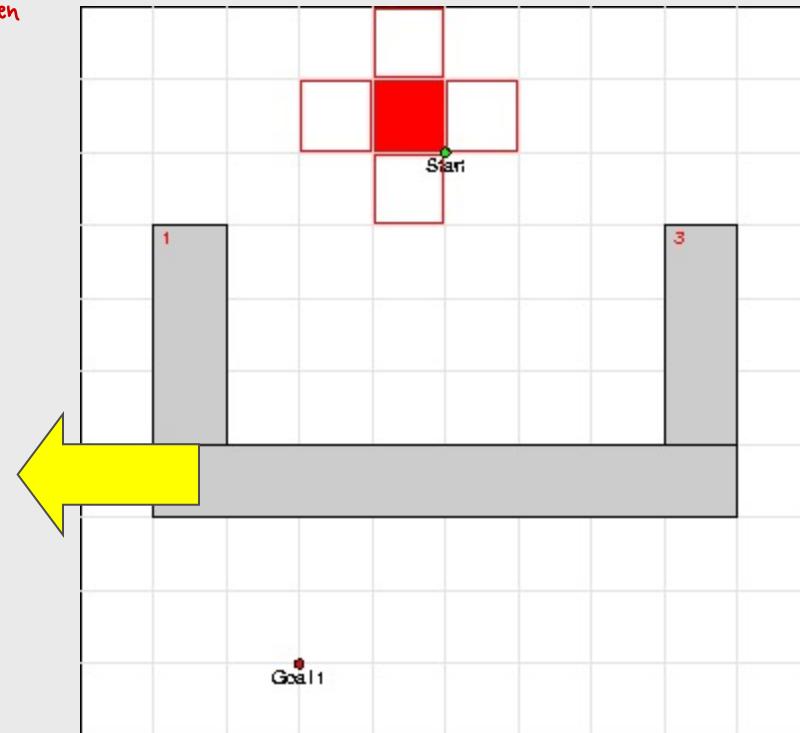
- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function

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

Tatsächliche Kosten vom Start bis zum Knoten n Schätzung der verbleibenden Kosten von n bis zum Ziel

```
init (OPEN;CLOSED; S) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```

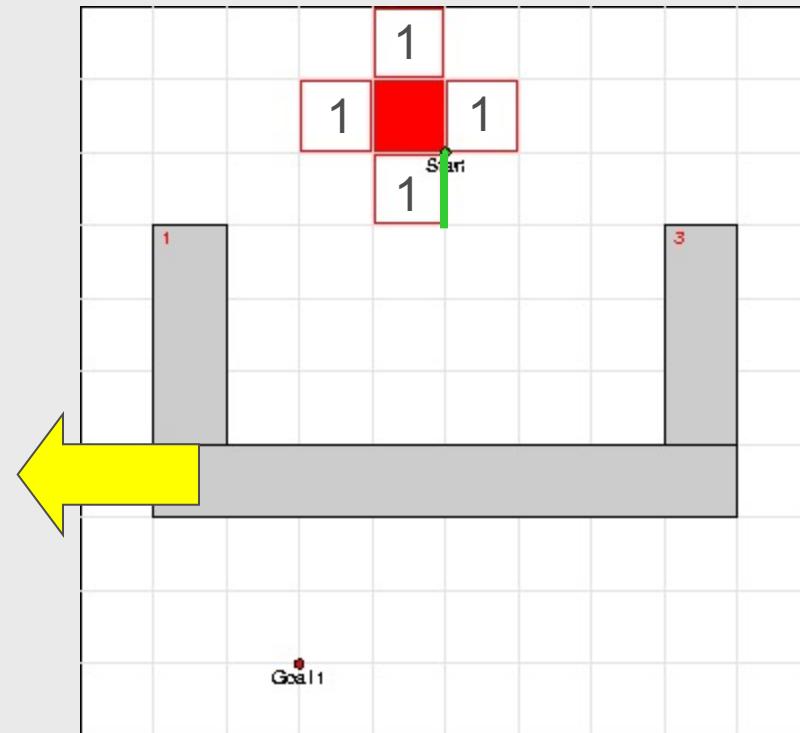
in jede Dimension wird der Nachfolgeknoten hinzugefügt



A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
 $f(n) = g(n) + h(n)$

```
init (OPEN;CLOSED; s) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```

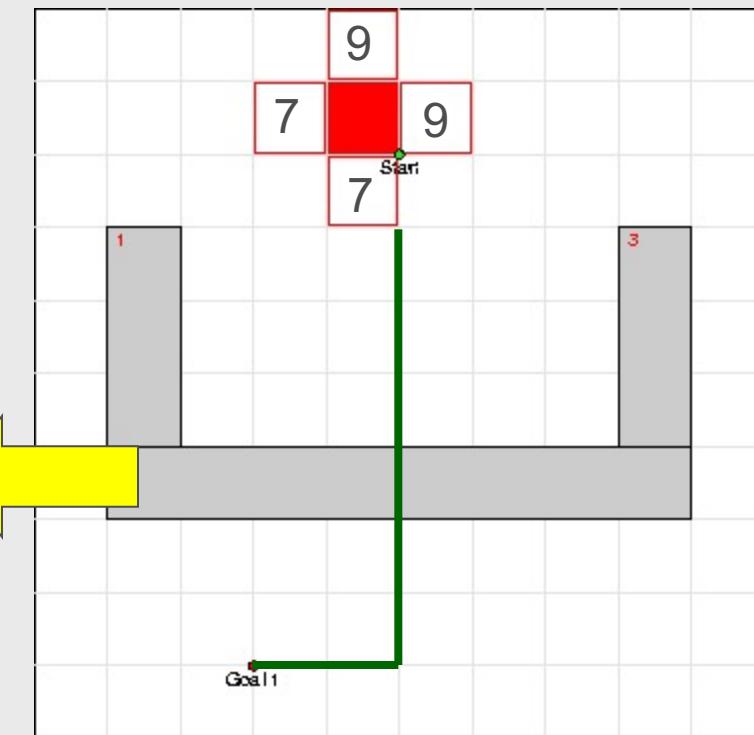


A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

- ▶ Manhattan

```
init (OPEN;CLOSED; s) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```



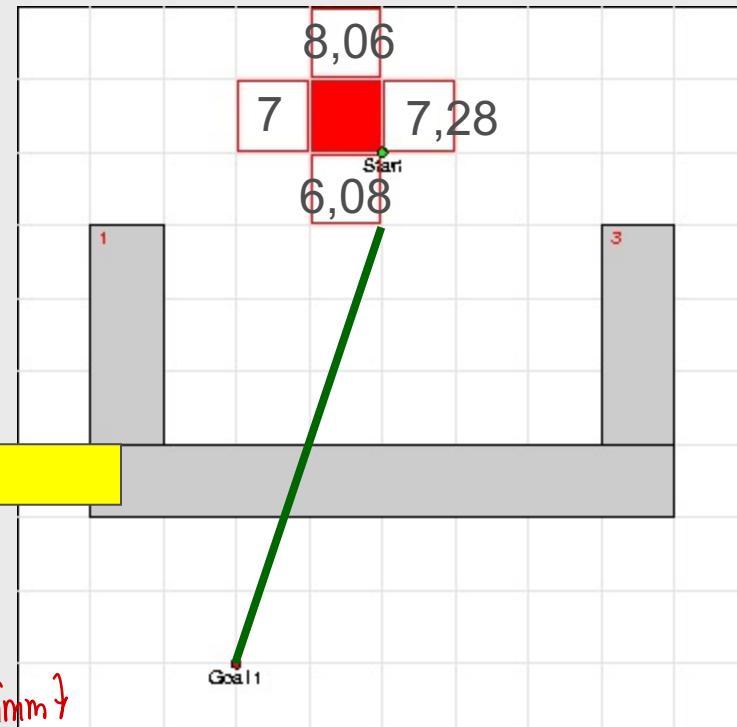
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

- ▶ Euclid

```
init (OPEN;CLOSED; s) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```

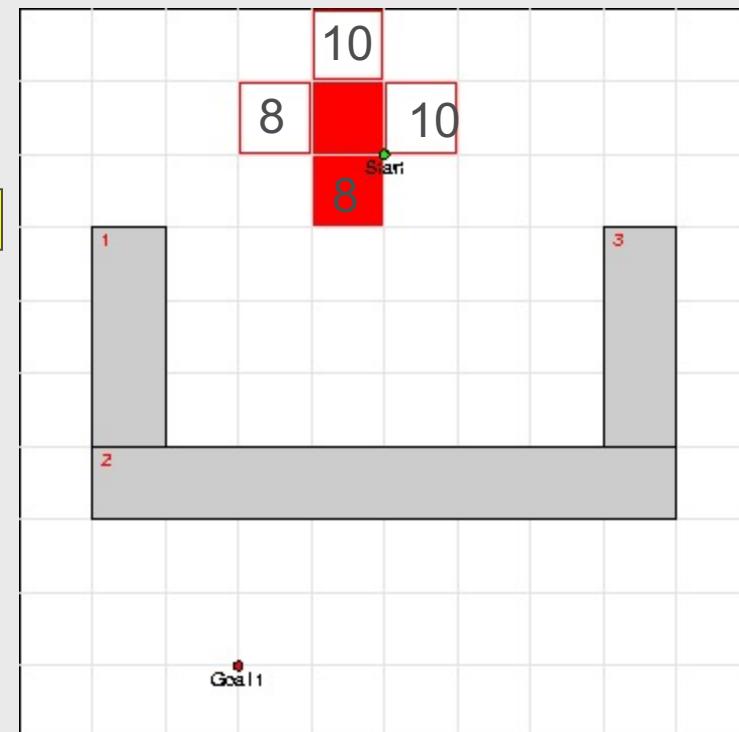
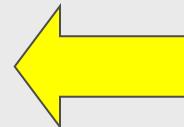
Knoten werden anhand von $h(n)$
bewertet und der beste Nachfolgeknoten wird bestimmt



A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

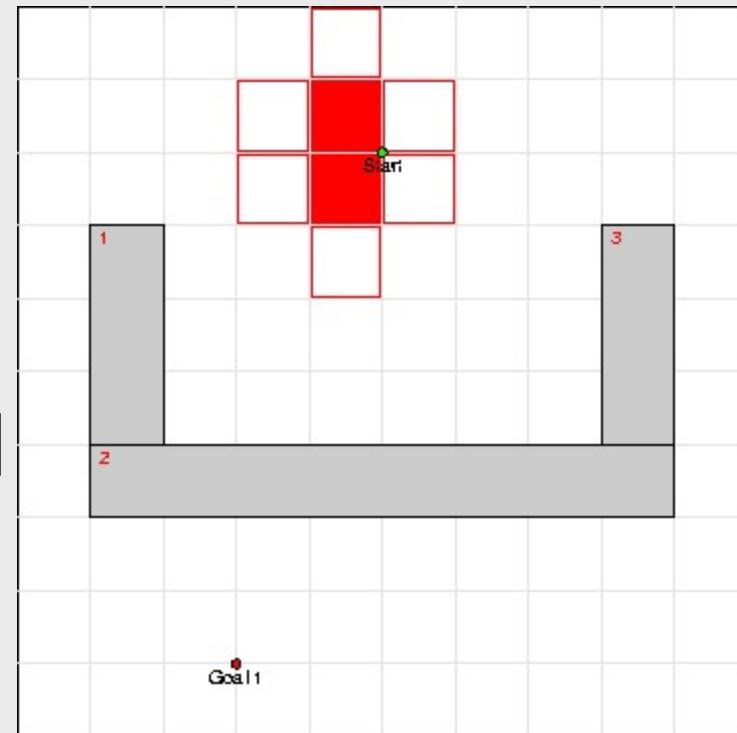
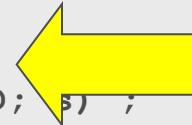
```
init (OPEN;CLOSED; s) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end
```



A*-search

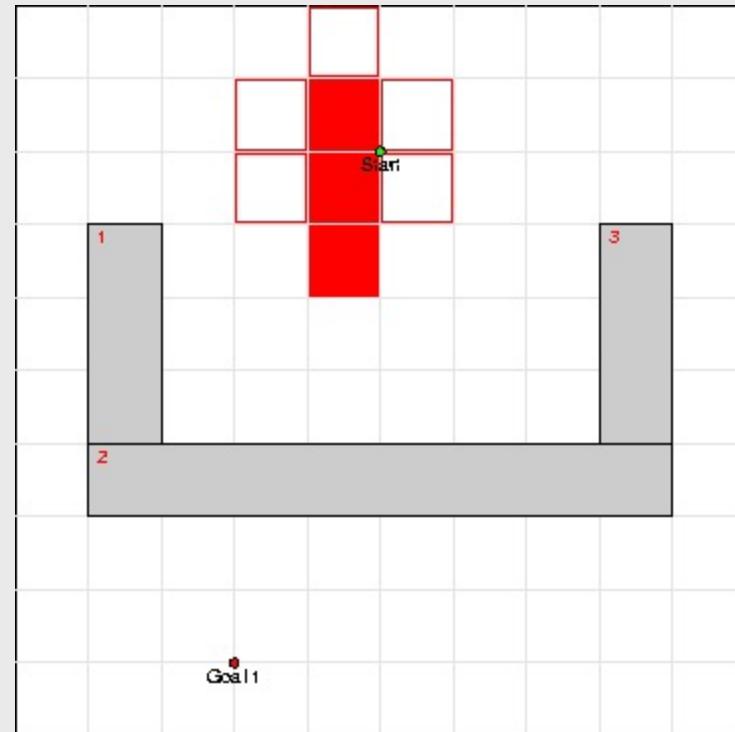
- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

```
init (OPEN;CLOSED; S) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; S) ;  
end
```



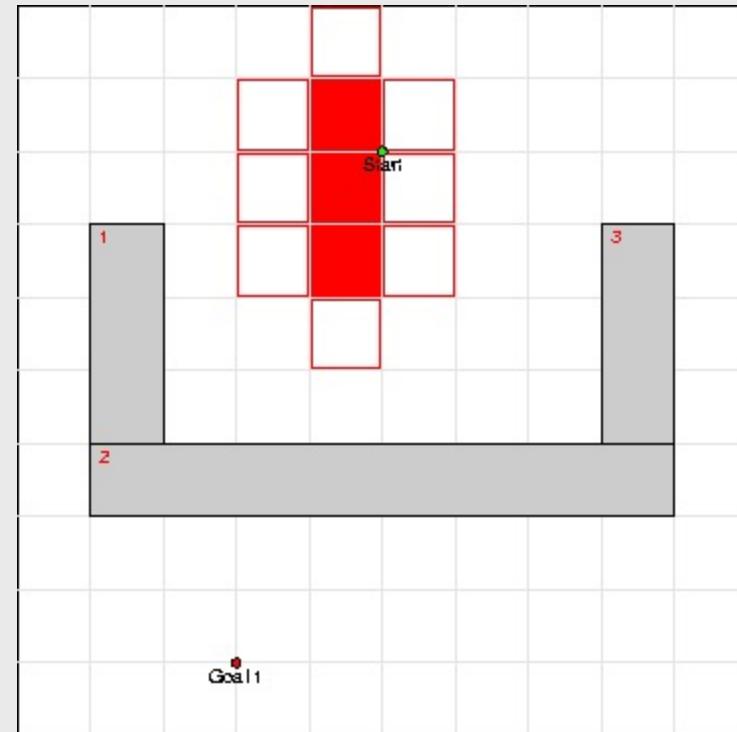
A*-search

- ▶ Graph search (A^*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



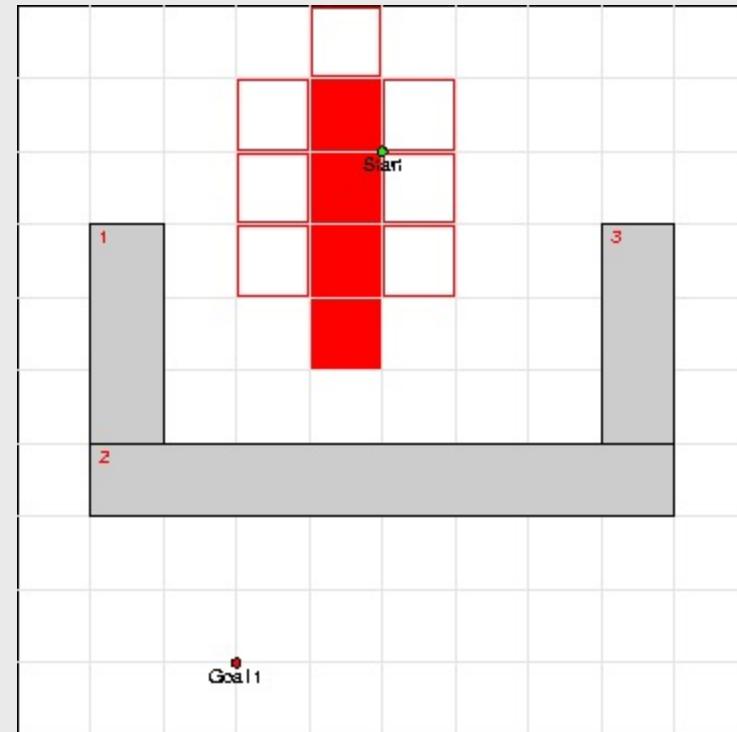
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



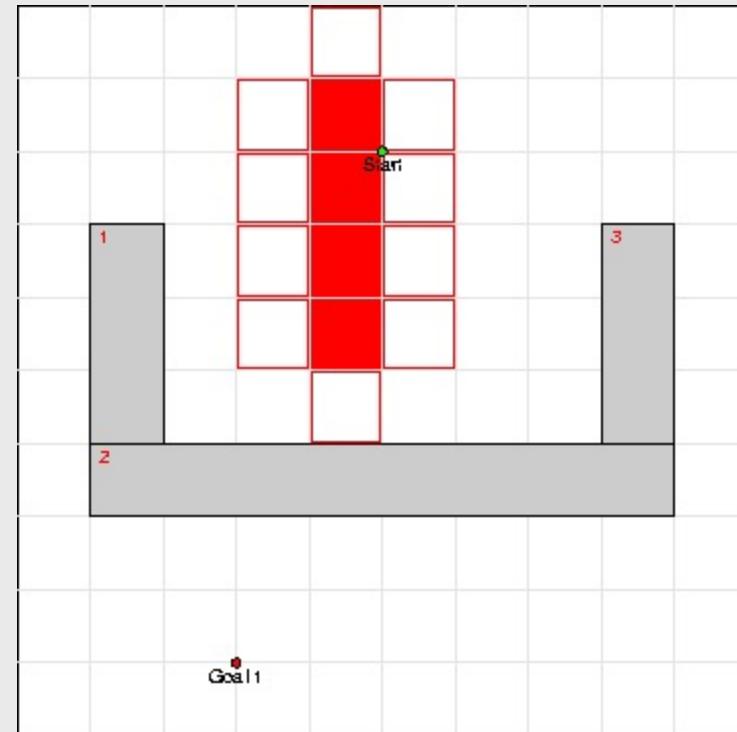
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



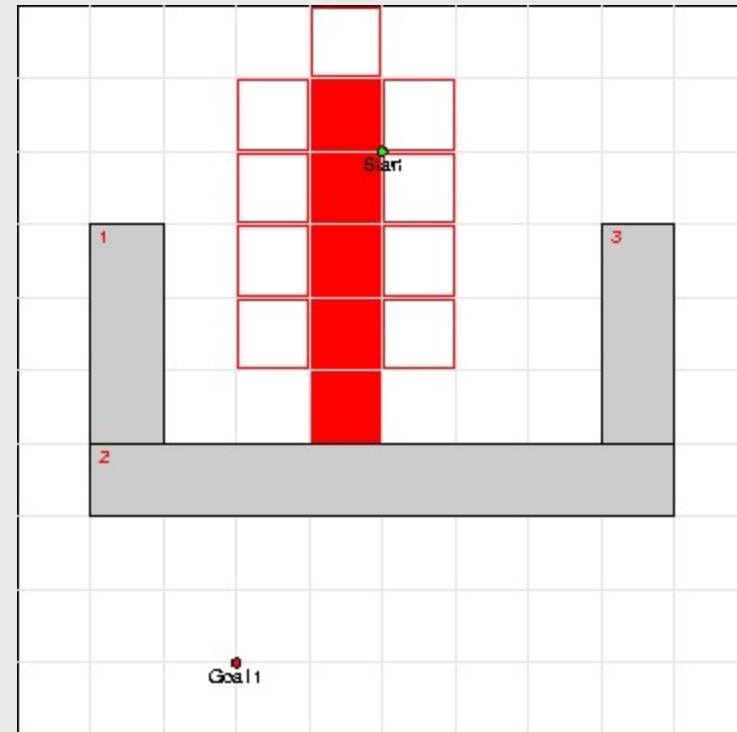
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

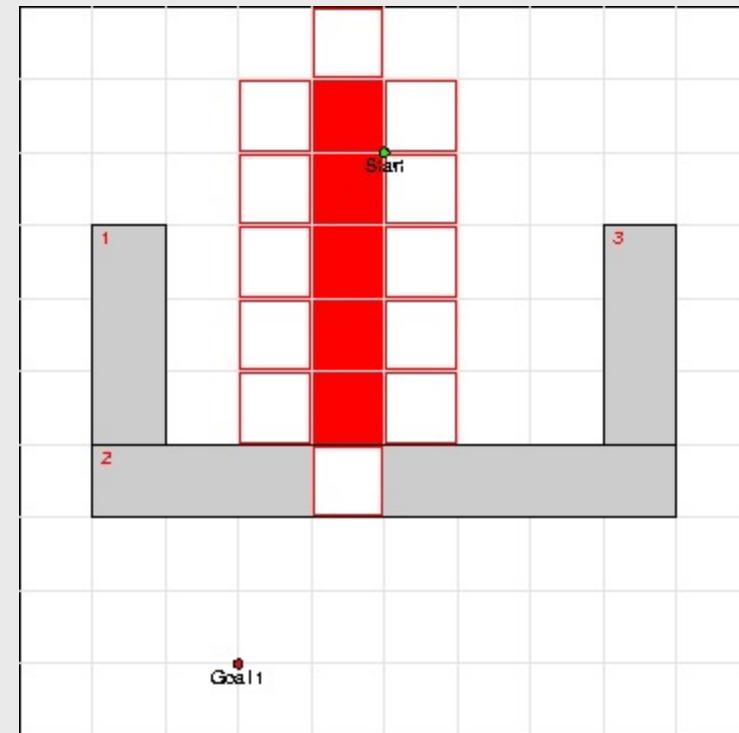


A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$

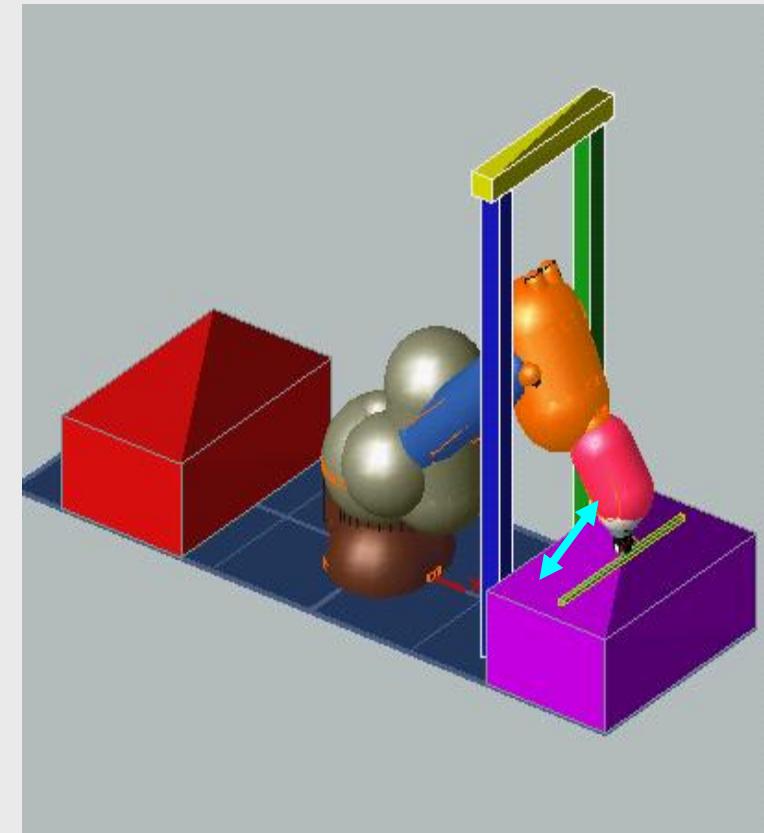
Bei jedem Schritt werden im Hintergrund die Hindernisse transformiert.

↳ Kollisionsprüfung



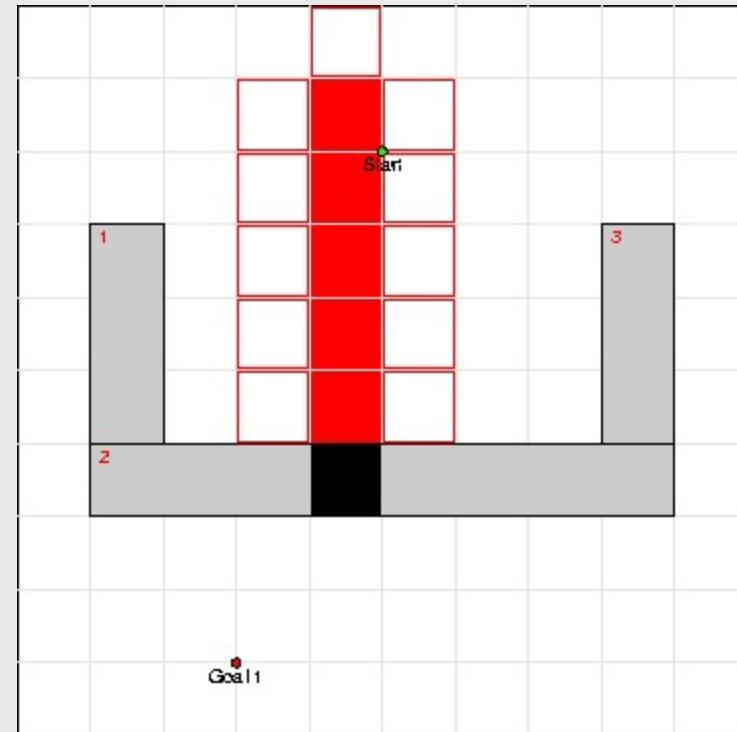
Collision detection

- ▶ Convex hull of robot and obstacles
- ▶ Approximation with primitive
- ▶ Distance-Calculation with Gilbert-Johnson-Kerthi-Algorithm



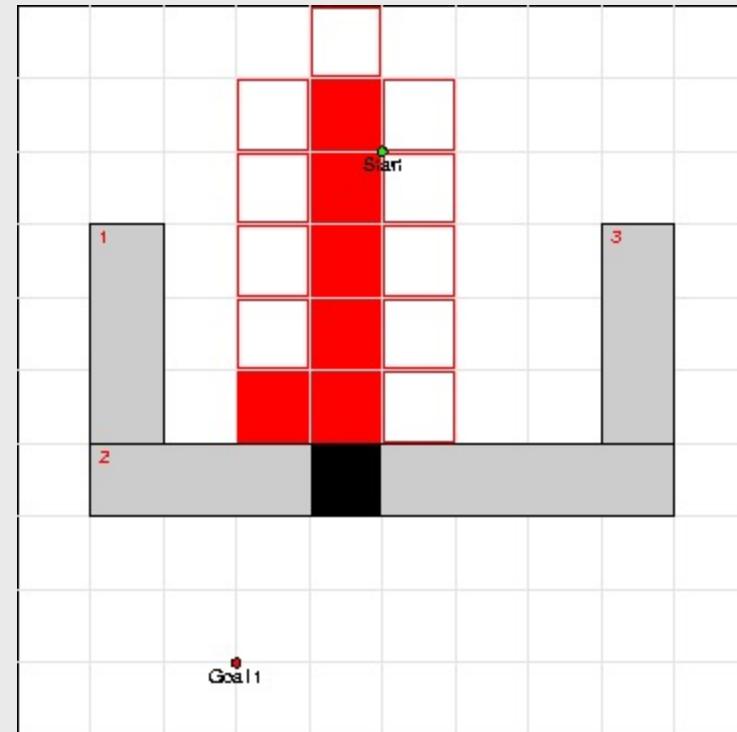
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



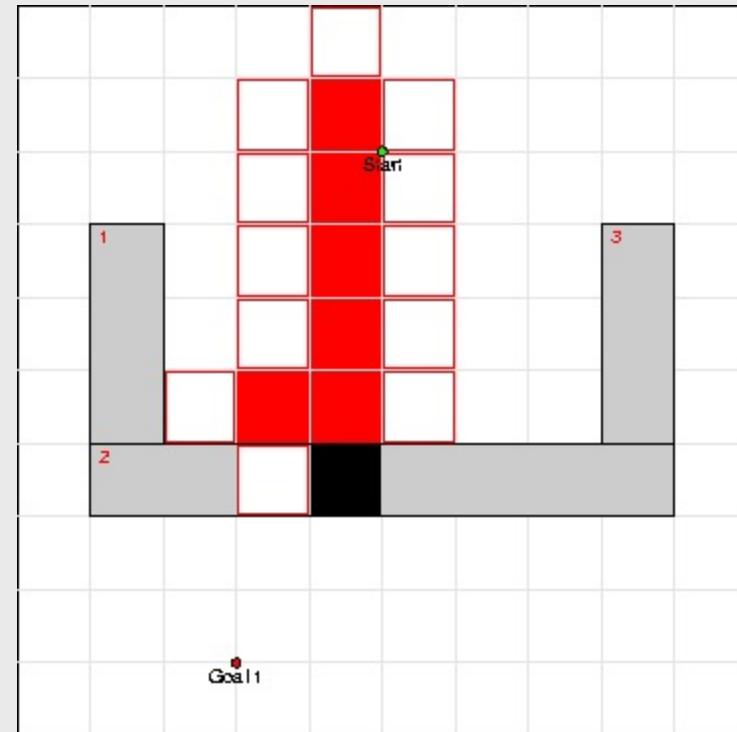
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



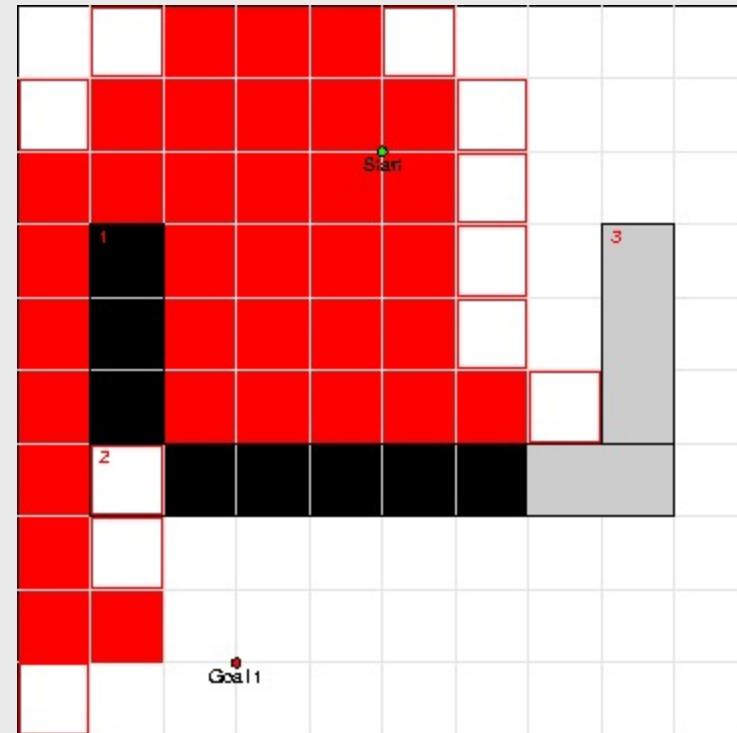
A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



A*-search

- ▶ Graph search (A*) in implicit C-space
- ▶ Heuristic evaluation function
$$f(n) = g(n) + h(n)$$



A*-search

- ▶ Graph search (A*) in implicit C-space

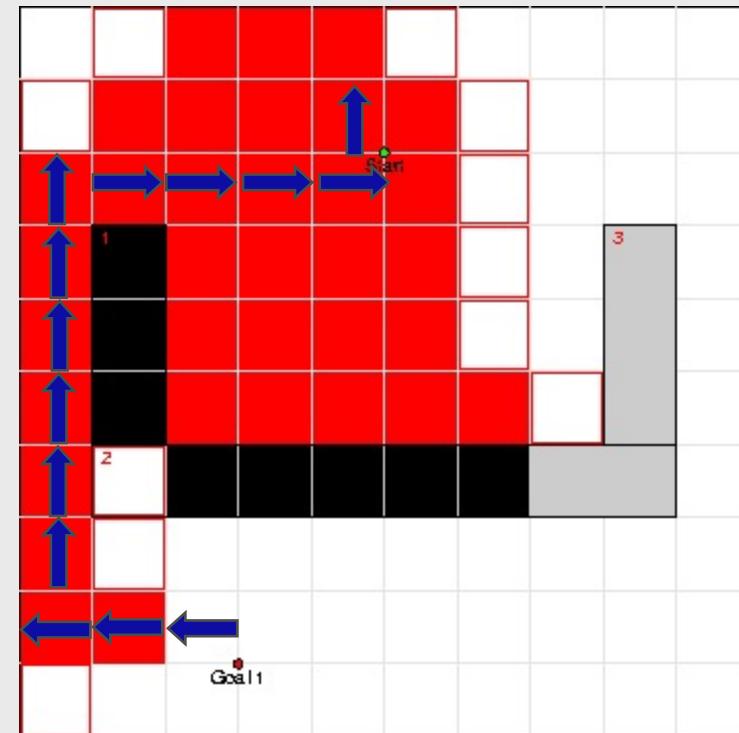
- ▶ Heuristic evaluation function

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

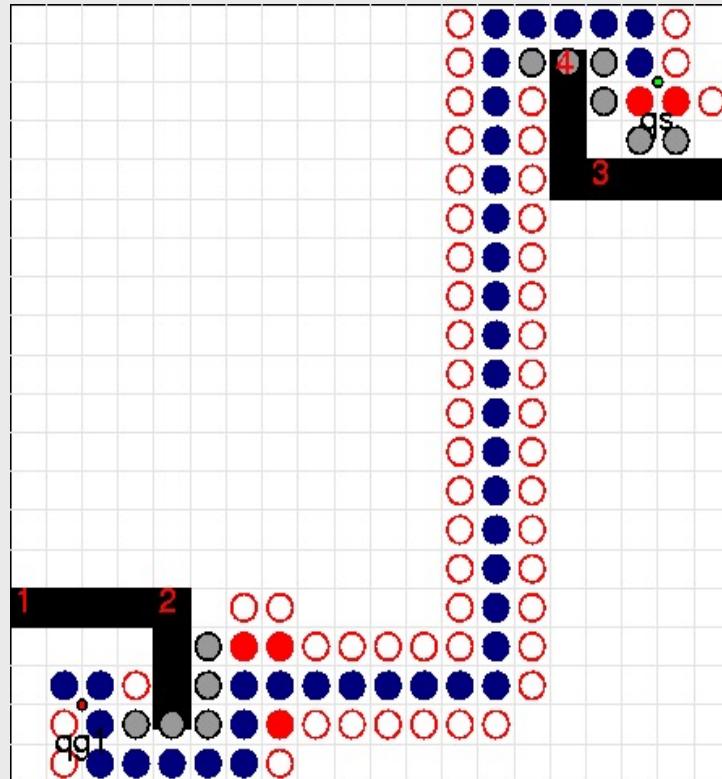
- ▶ Collecting solution

```
init (OPEN;CLOSED; s) ;  
goalFound = FALSE;  
while (OPEN ≠ Ø) do begin  
    n = best (OPEN) ;  
    if (n == G) then  
        goalFound = TRUE;  
        break;  
    endif  
    s = expandNode (n) ;  
    handleSuccessors (OPEN;CLOSED; s) ;  
end  
if (goalFound == TRUE) then  
    printSolution (: : : ) ;
```

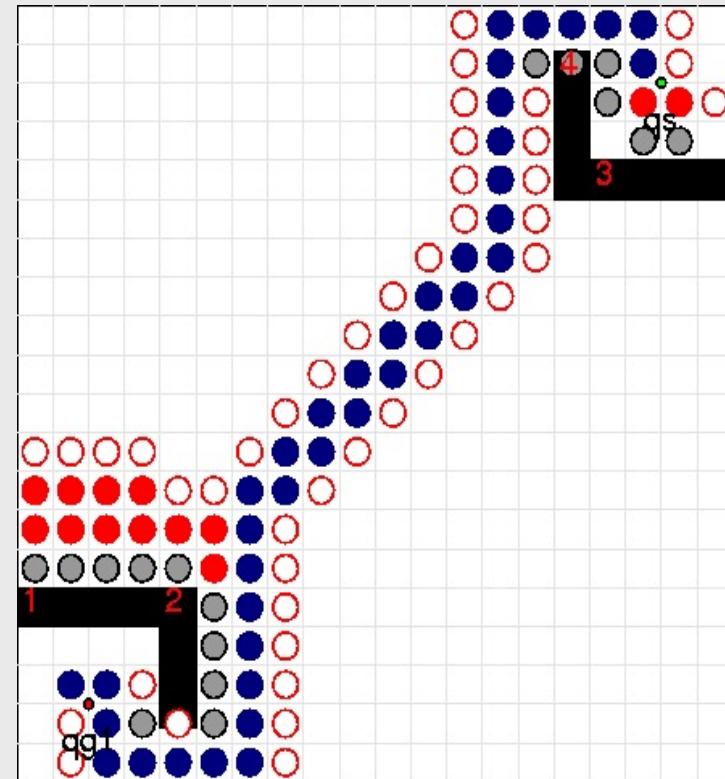
Reopening von nodes möglich



Comparison: Used Metrics



MANHATTAN



EUKLID

Modifying evaluation function

- ▶ Modified evaluation function:

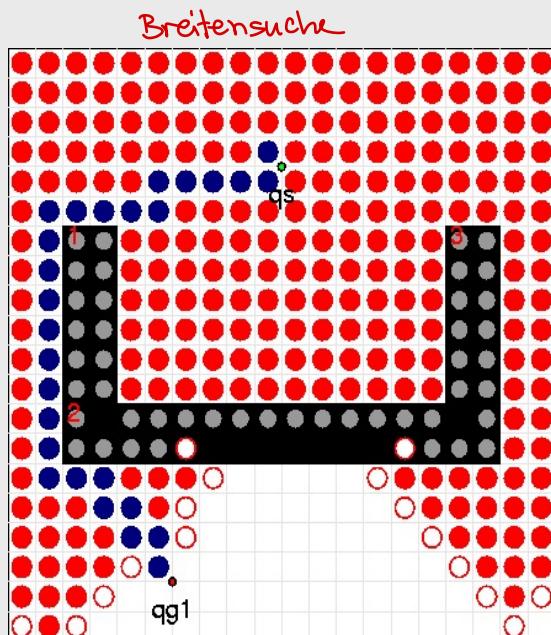
$$f(n) = (1-w) * g(n) + w * h(n)$$

- ▶ w : Heuristic weight

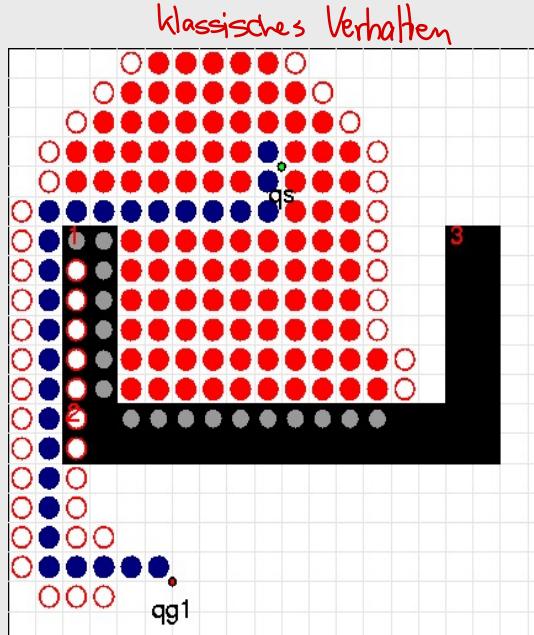
origineller A* hat keine Gewichtung!

← Gewichtung :

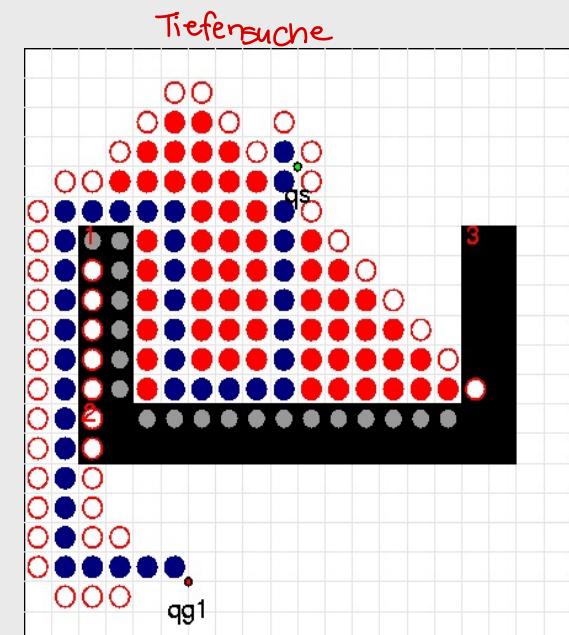
Tiefensuche, wie stark soll er in Breite suchen? $q_5 \Rightarrow$ klassisches Verhalten



$w = 0$



$w = 0.5$

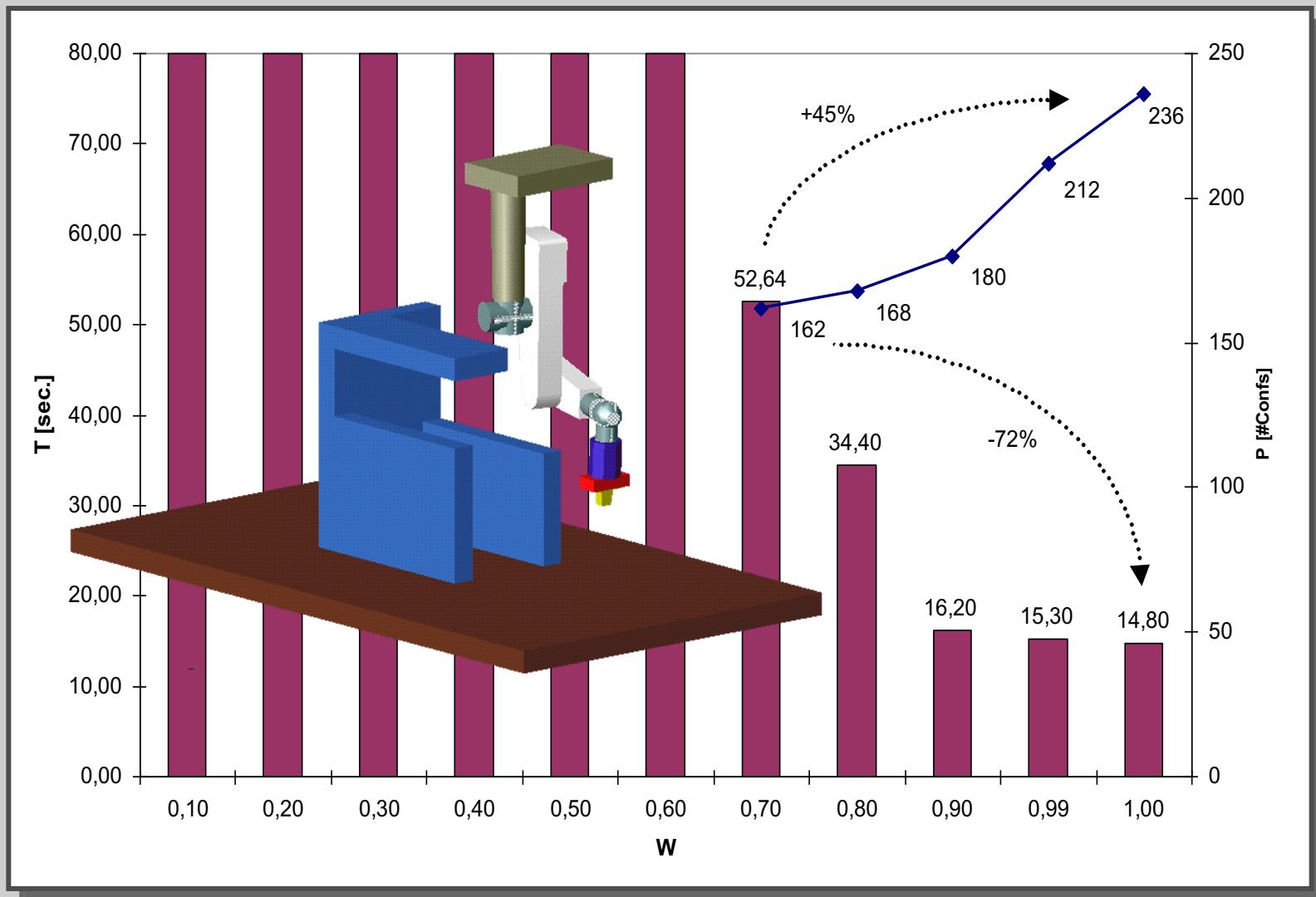


$w = 1$

IP-8-O_ASTAR Jupyter Notebook Übung → Verstehen

Heuristic weight (TRAP)

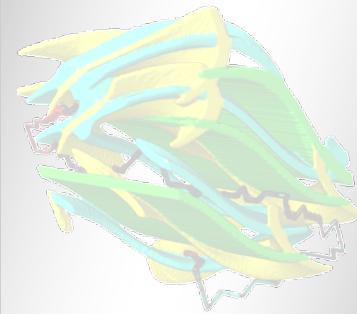
heuristik — Tiefensuche wird höher gewichtet



Extensions of A*

PRM Pfade müssen nachbearbeitet werden

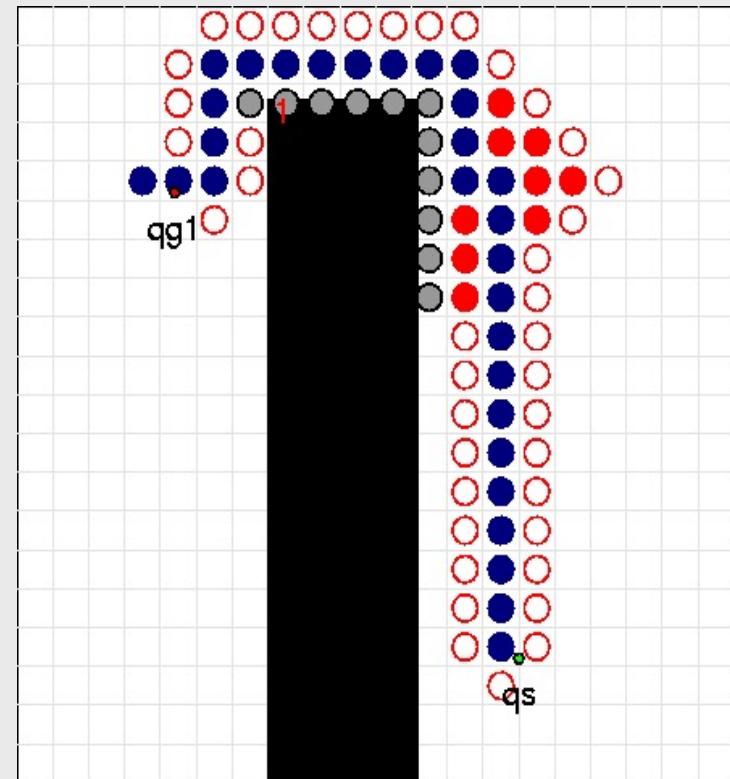
A* findet sehr optimalen Pfad



Unidirectional search

▶ Problem

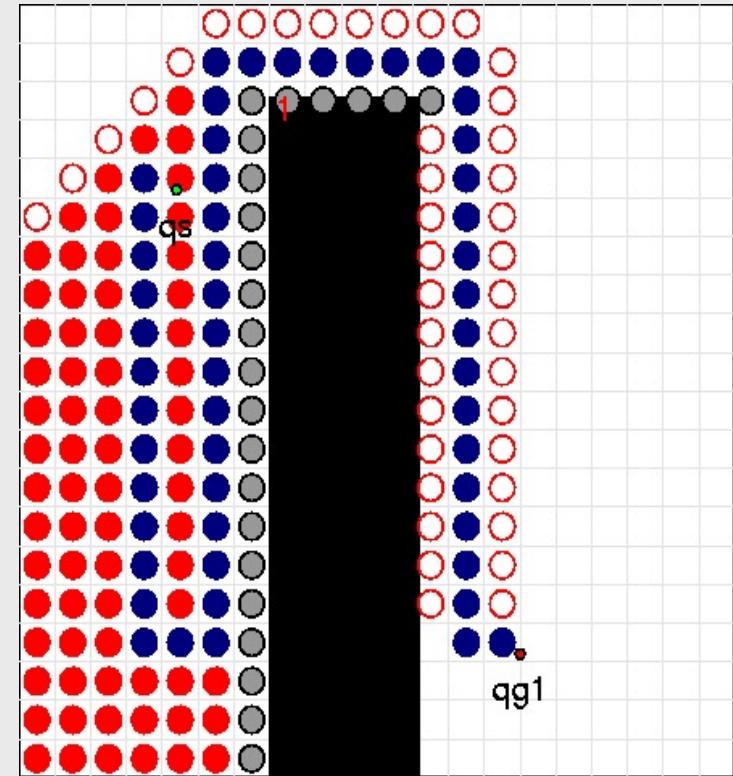
- ▶ Time needed to find solution is depending on **search direction**



Other direction

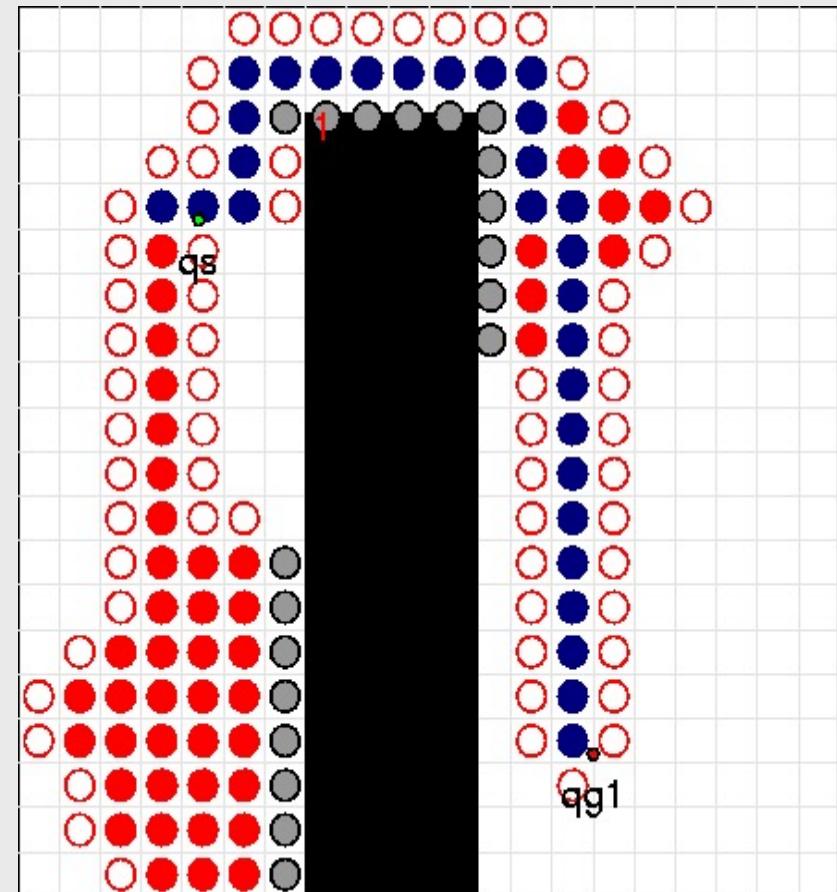
► Problem

- Time needed to find solution is depending on **search direction**



Bidirectional search

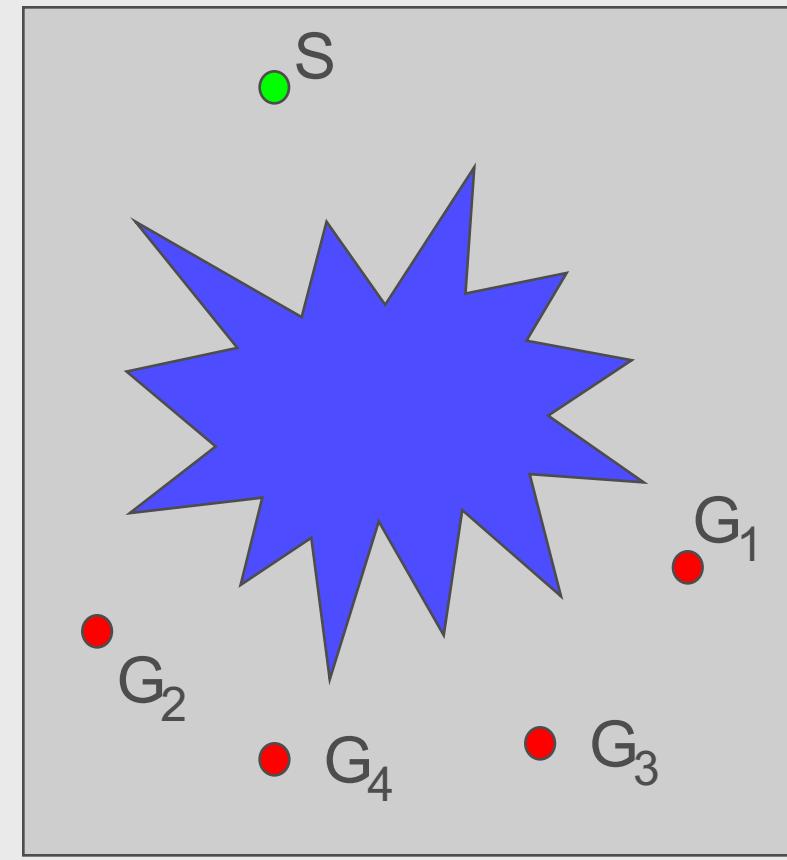
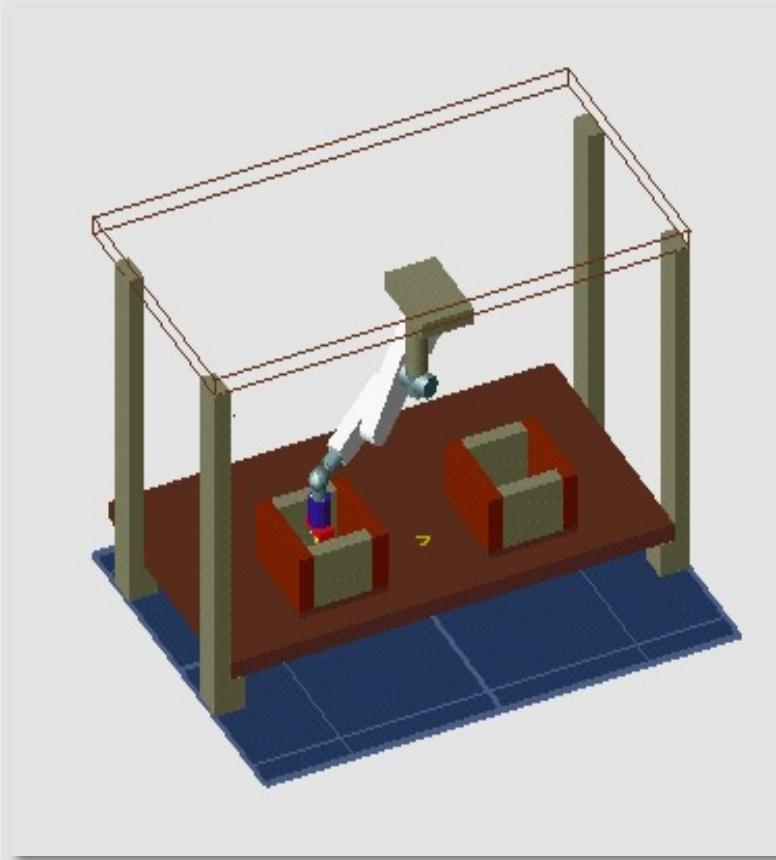
- ▶ Goal: combining forward and backward search
- ▶ Modifying A*-algorithm:
 - ▶ Two OPEN-Lists covering forward- and backward search
 - ▶ Nodes of the two lists are alternatingly expanded
 - ▶ Search algorithm ends, if a duplicate in shared CLOSED-List is found



Inverse kinematics: „More than one goal“

- ▶ Benchmark „STAR“:

- ▶ Goal configuration can be reached in 4 different ways



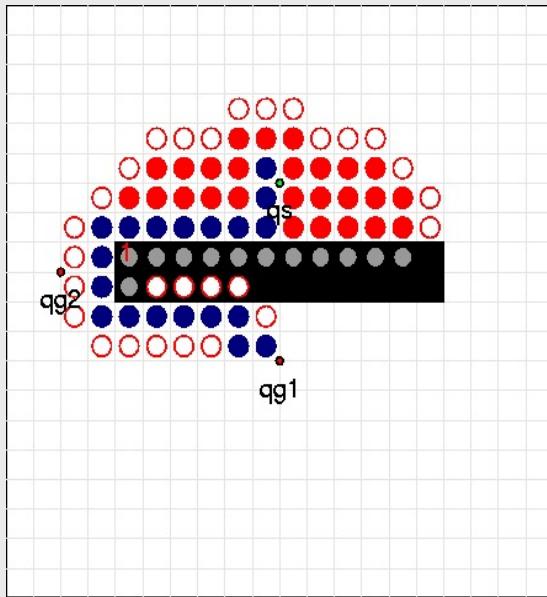
- ▶ How can this be used during search?

Dynamic goal switching

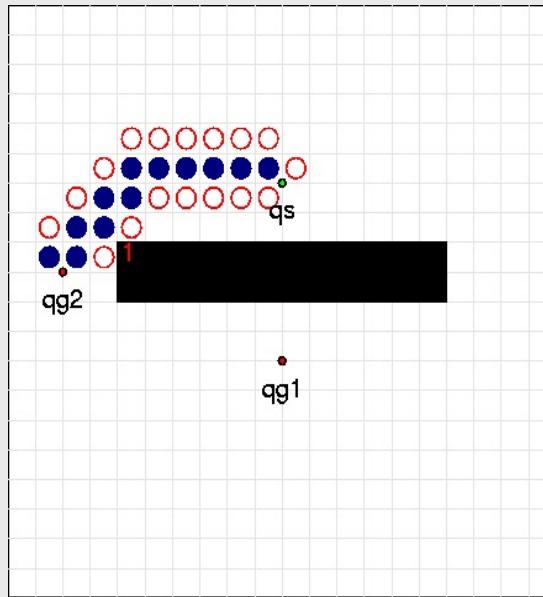
Idea

- ▶ include all possible goal configurations given by inverse kinematics during search
- ▶ In `handleSuccessors (...)` heuristic value to all goals are evaluated
 - ▶ Search direction is always heading for the nearest goal
 - ▶ Current goal will automatically change during search if it seems better suited

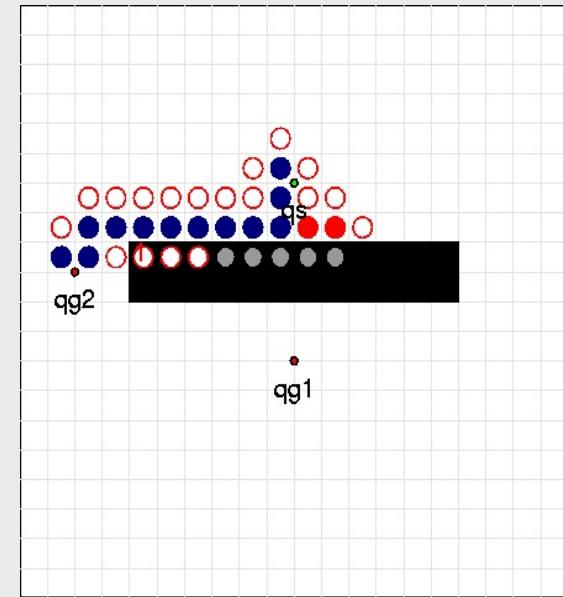
Dynamic goal switching



$q_s \rightarrow q_{g1},$
no goal switching
Nodes: 55



$q_s \rightarrow q_{g2},$
no goal switching
Nodes: 12



$q_s \rightarrow q_{g1} \& q_{g2},$
goal switching
Nodes: 19

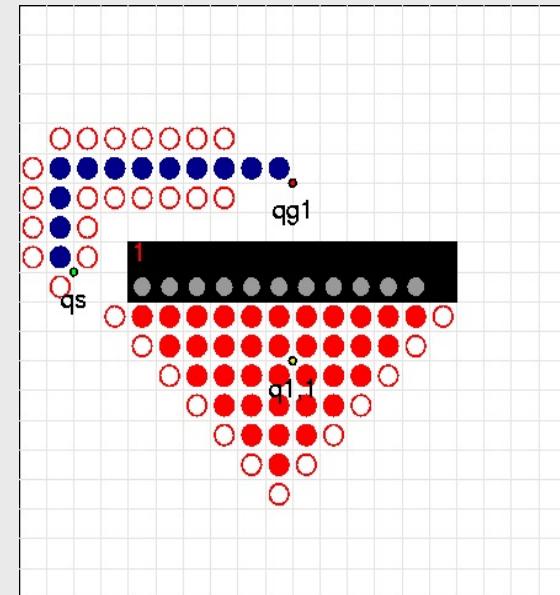
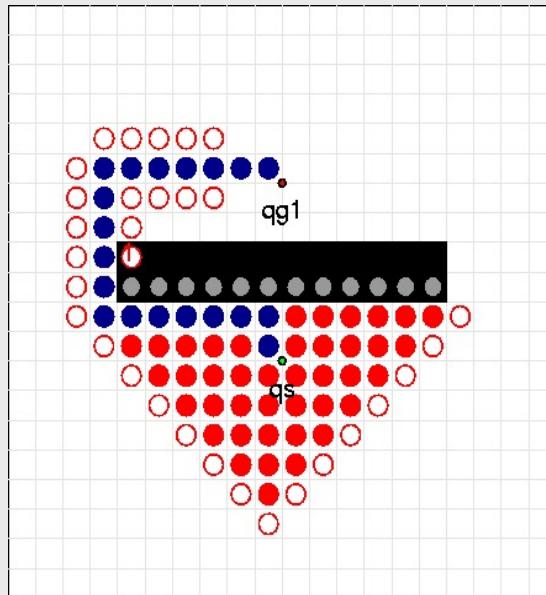
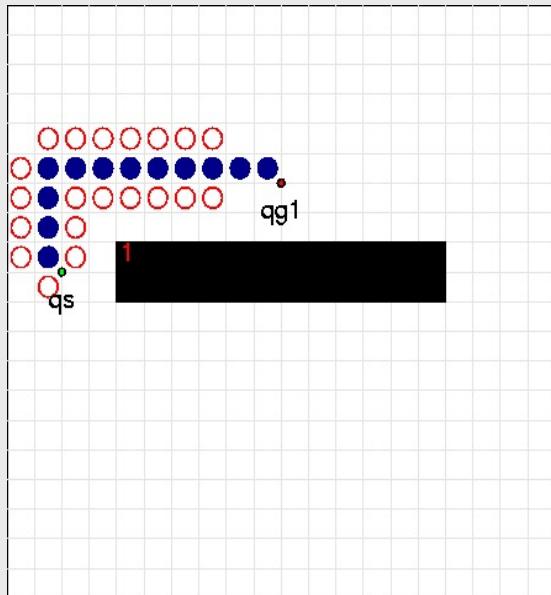
Dynamic start switching

- ▶ Goal switching works fine!
- ▶ Use same thing for start

Idea

- ▶ Insert all start configurations into OPEN-List
- ▶ Let A* automatically choose best one

Dynamic start switching



$q_s \rightarrow q_{g1}$,

no start switching

Nodes: 12

$q_{s2} \rightarrow q_{g2}$,

no start switching

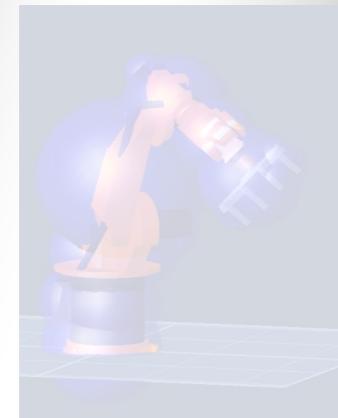
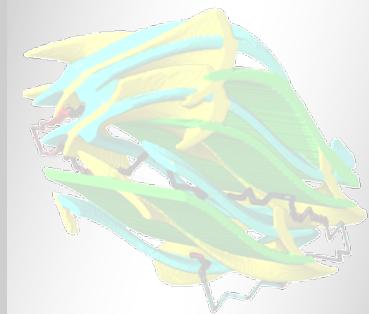
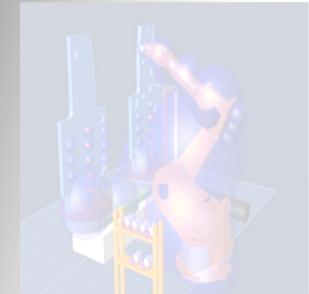
Nodes: 72

$q_{s1} \& q_{s2} \rightarrow q_{g1}$,

start switching

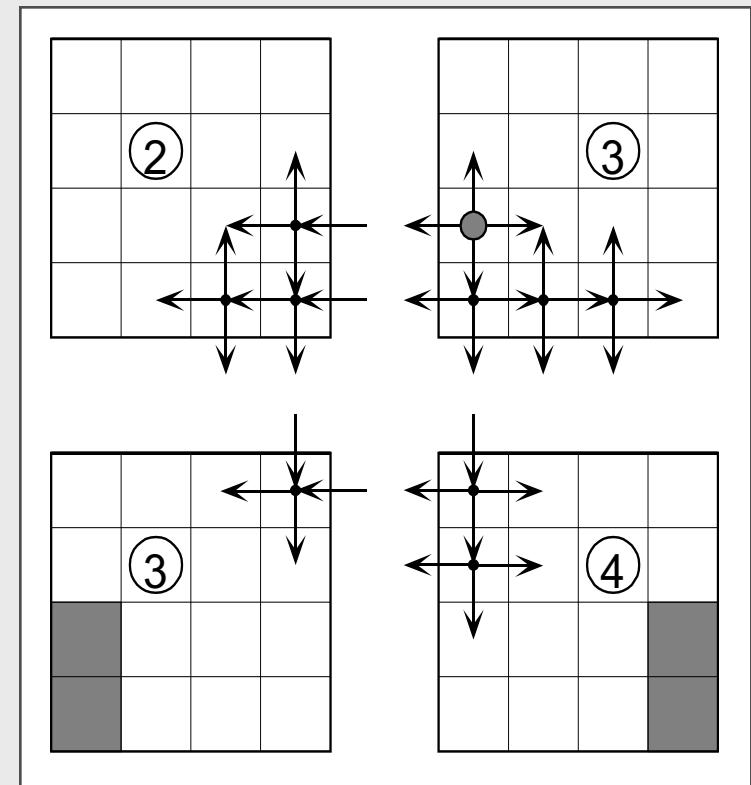
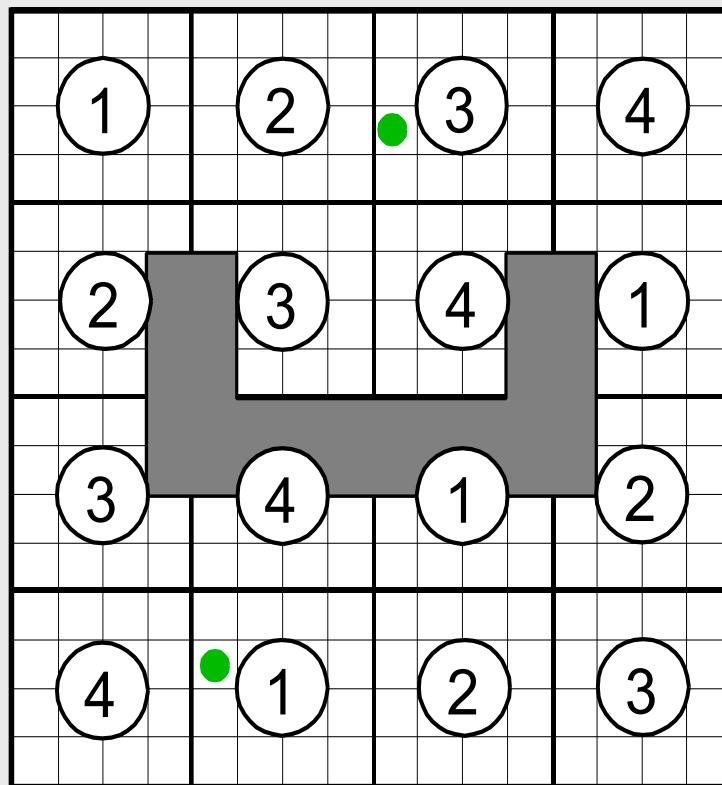
Nodes: 59

Parallel A*-Search



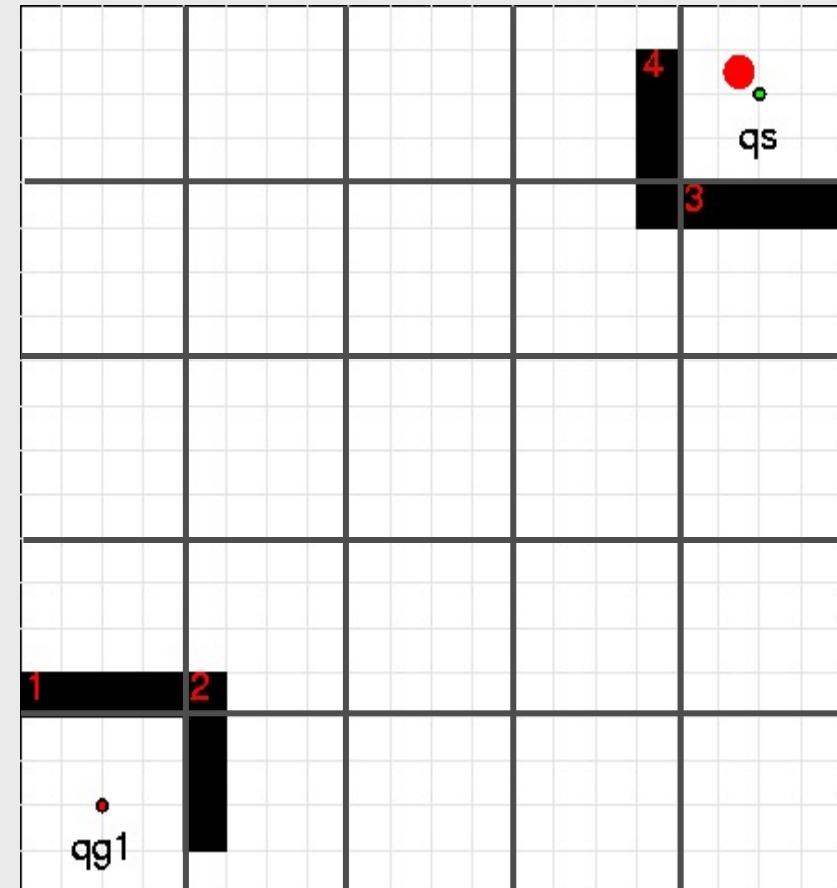
Parallel A*-Search: Idea

- ▶ C-space can easily be separated
- ▶ Search is done on more than one processor
- ▶ Speed up planning process



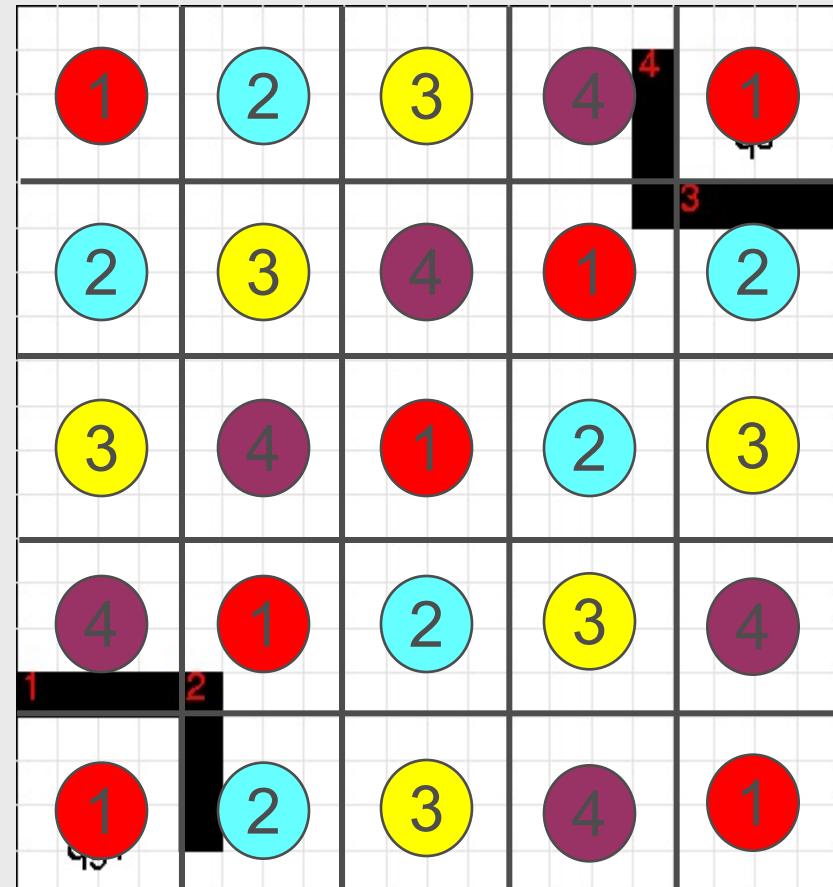
Parallel A*-Search

- ▶ Separation in hypercubes



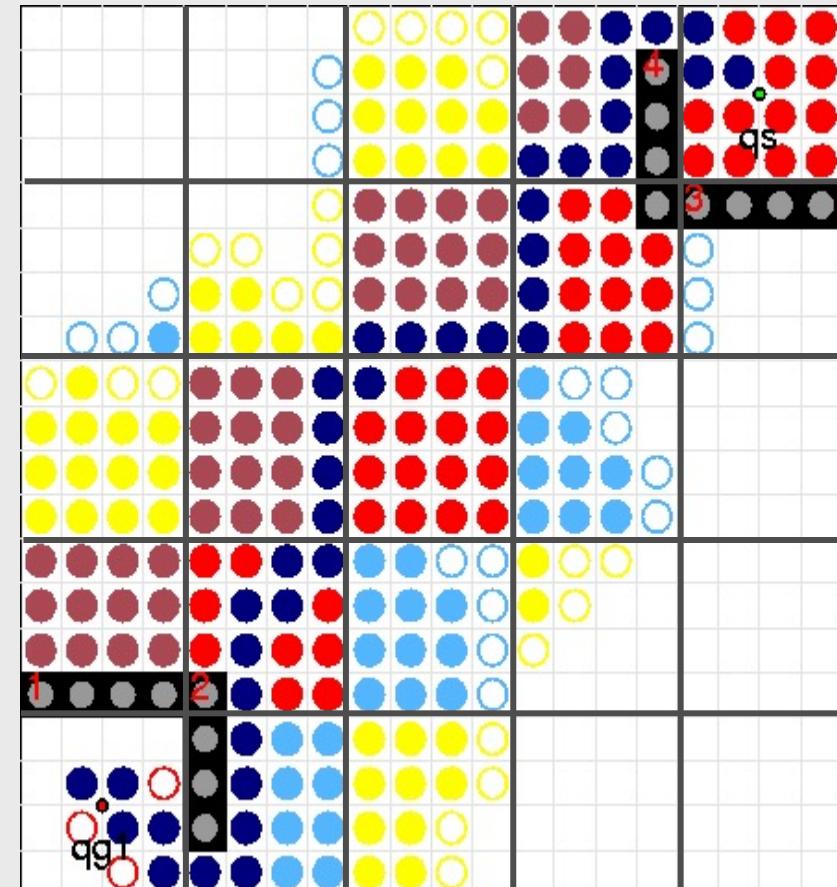
Parallel A*-Search

- ▶ Separation in hypercubes
- ▶ cyclic distribution of processors



Parallel A*-Search

- ▶ Separation in hypercubes
- ▶ cyclic distribution of processors
- ▶ Local A*-Search on every processor

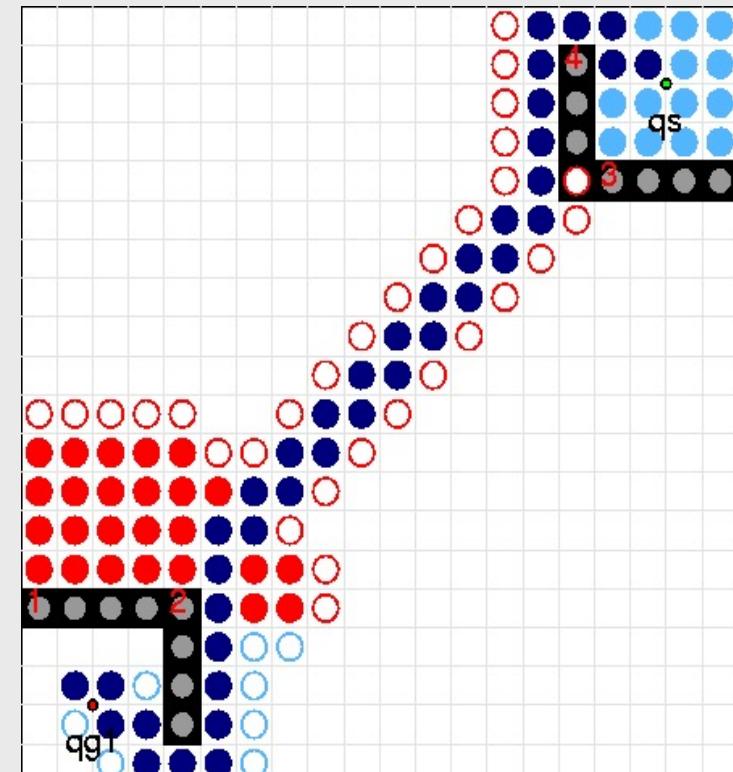


Parallel A*-Search: Size of hypercubes

- ▶ Solution is depending on size of hypercubes

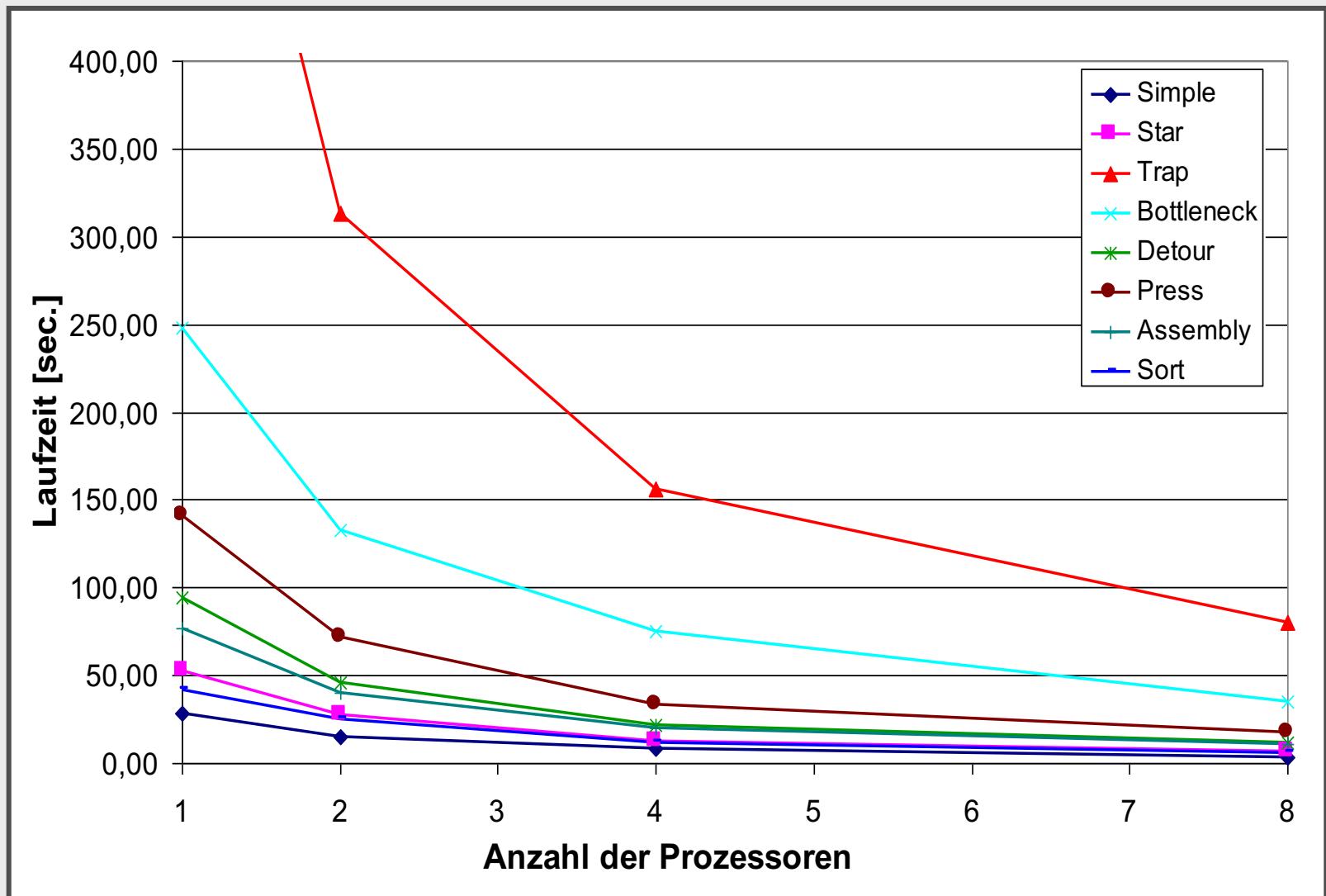


$b=4$



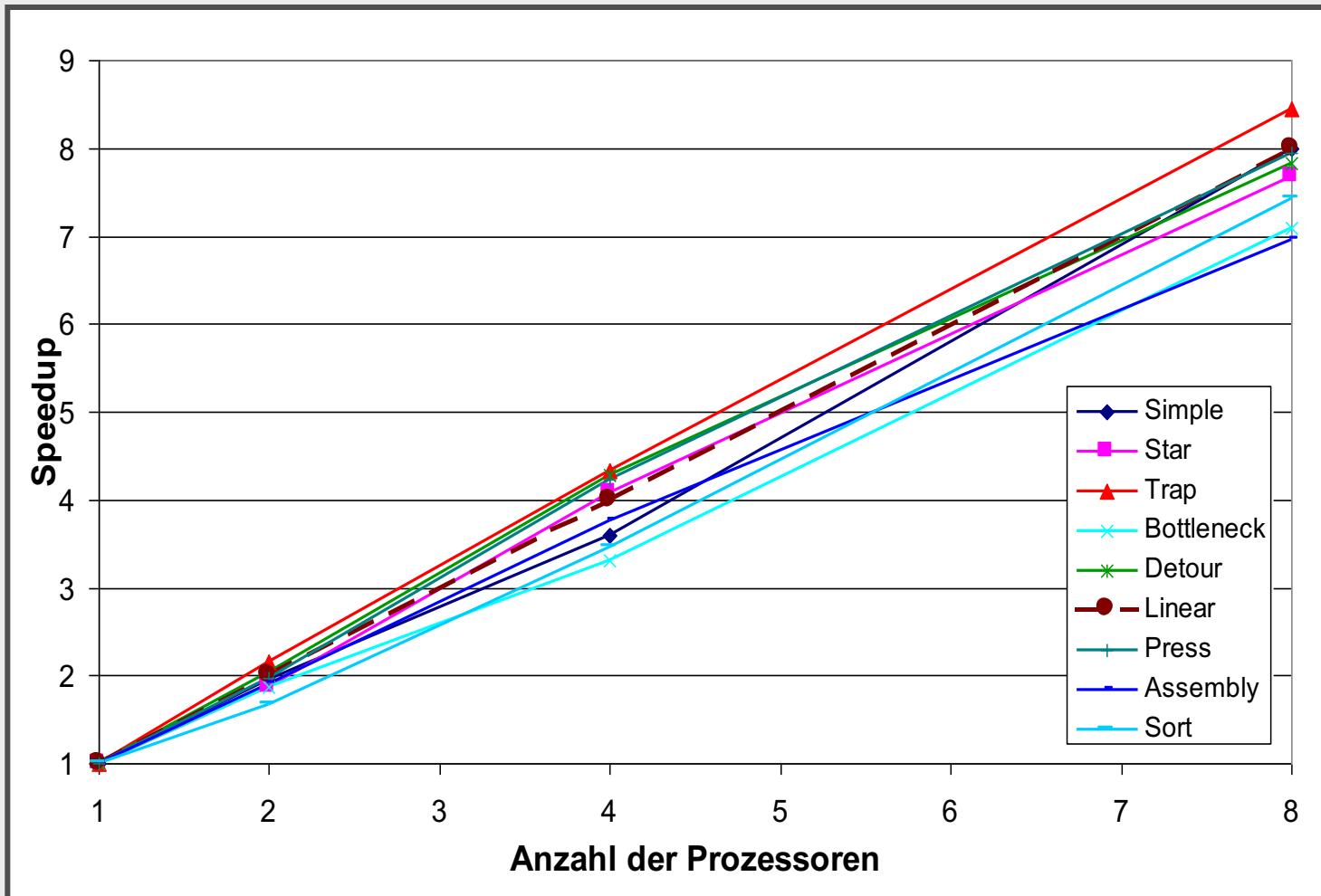
$b=16$

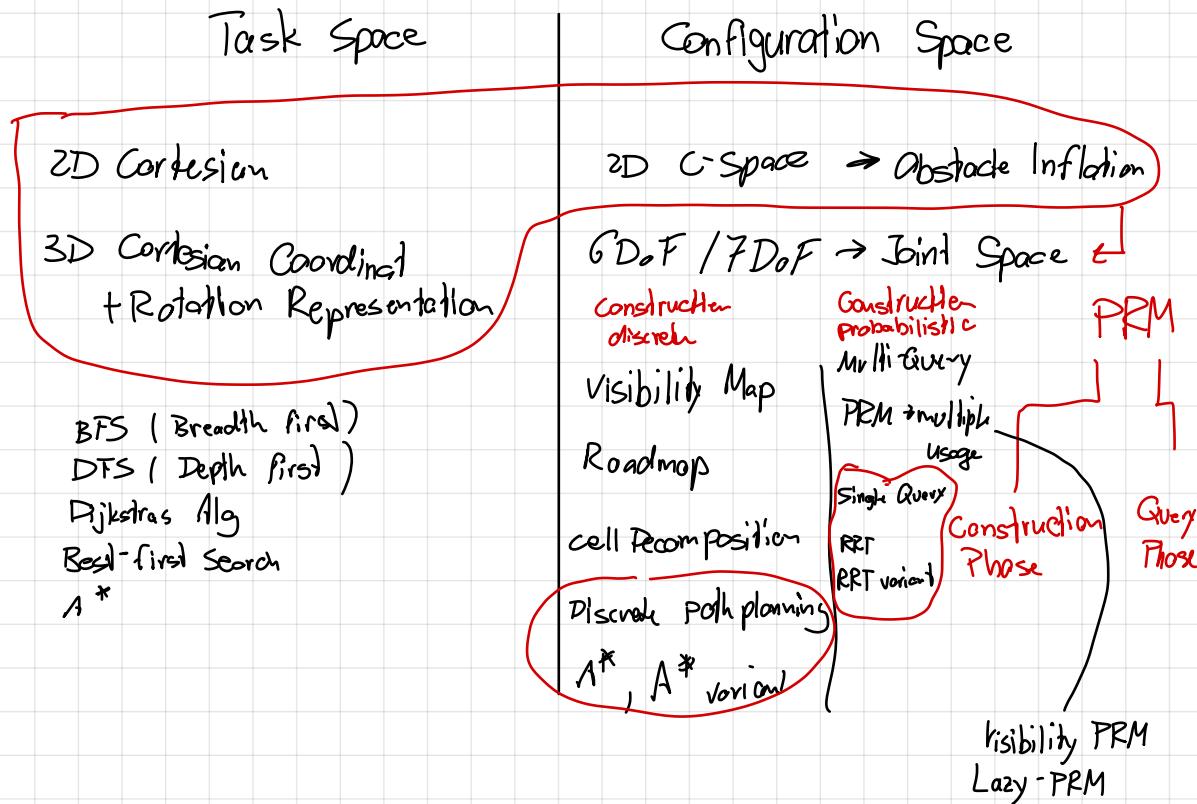
Planning time with parallel A*-Search



Speedup

► Speedup $S = T_{\text{serial}}/T_{\text{parallel}}$

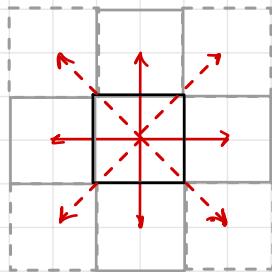




Mindmap von Yucheng prüfen lassen !!

Page 7 in PP

Node Edges



Successor \rightarrow neighbors (4 or 8)

Bildur

Breadth first Search 2 (or 3)

Depth first Search ()

Dijkstra's Search (BFS with cost optimization)

Best first Search

$$A^*: f(x) = \underset{\text{Dijkstra's depth}}{g(x)} + h(x)$$

$$\text{Best first search} = f(x)_b = h(x)$$

$$\begin{aligned} \text{Distance norm 1} \\ \text{norm 2} \\ \sqrt{\text{norm 2}} \end{aligned}$$

A^*

GitHub AtsushiSakai Python Robotics \rightarrow Path Planning

2. Teil Vorlesung

A^* Variants

Bas $\left\{ \begin{array}{l} \text{Bidirectional } A^* \\ \text{Multi-goals / starts } A^* \\ \text{Parallel } A^* \\ (\text{hierarchical } A^*) \end{array} \right.$

Venue's Extensions of A^*

Folie ↑

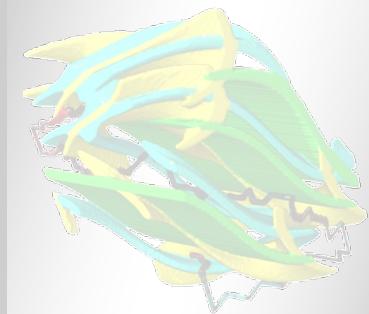
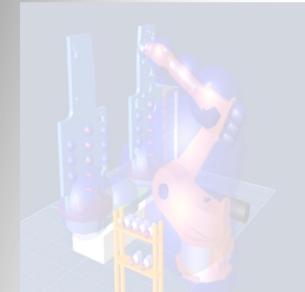
RRT

Basic RRT

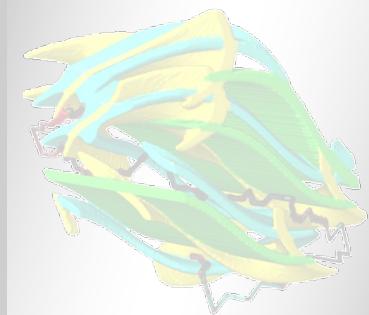
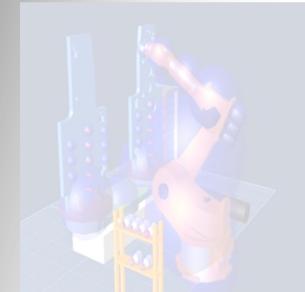
RRT connected

Bidirectional RRT

A* - Algorithm for path planning



Hierarchical A*-search

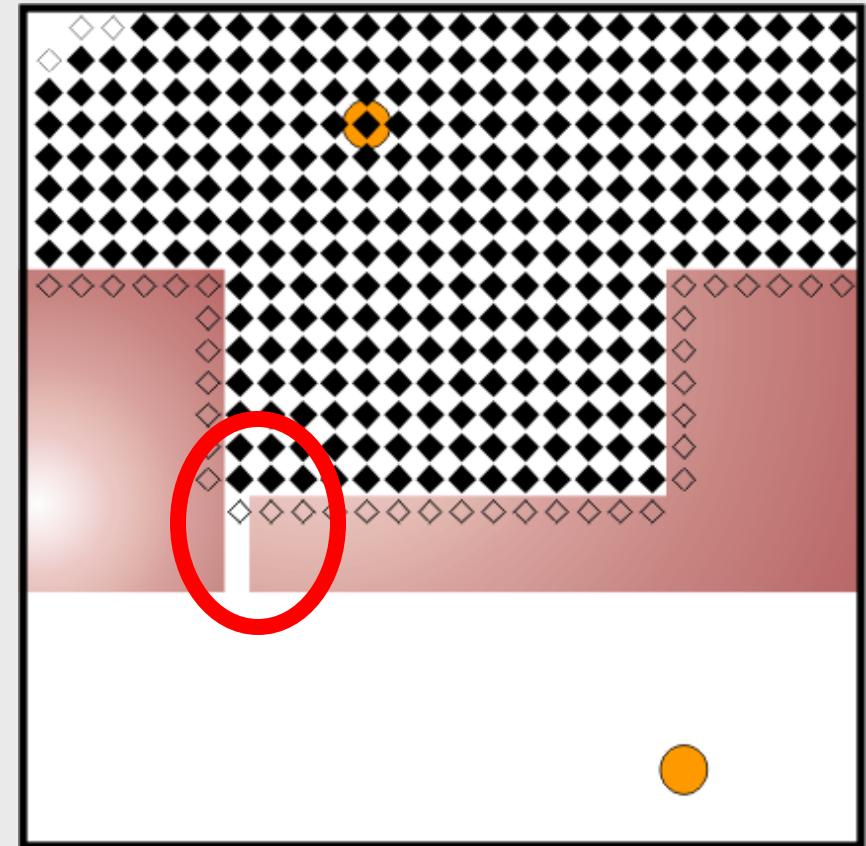
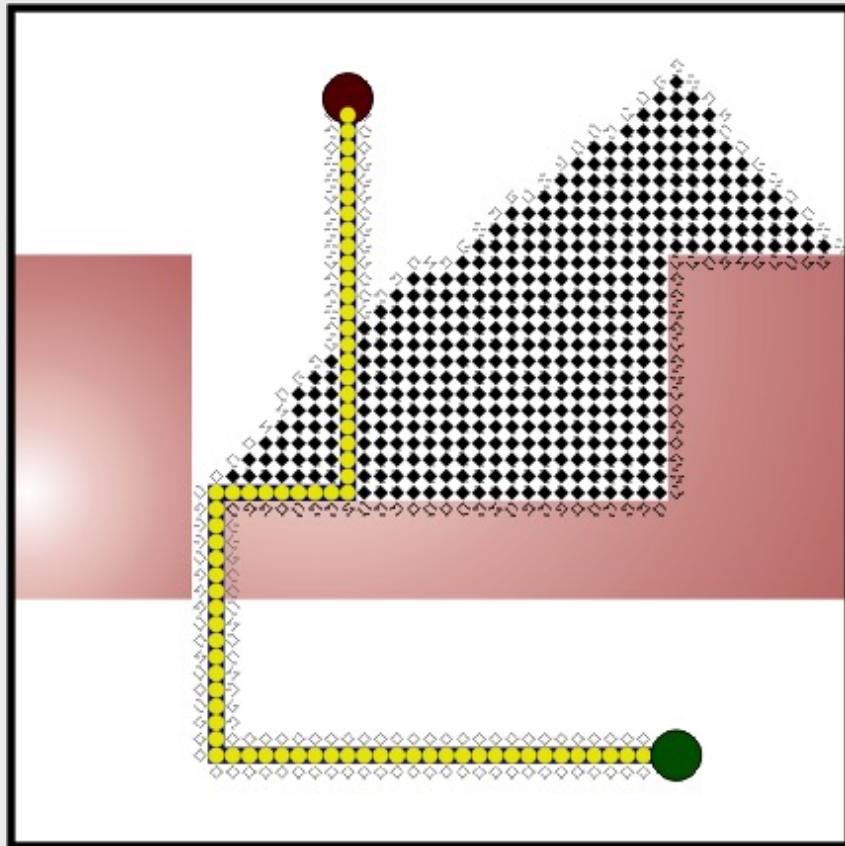


Problem

High complexity of C-space when planning robots with 6-DOF

→ **Choosing suitable discretization**

Choosing: Discretisation

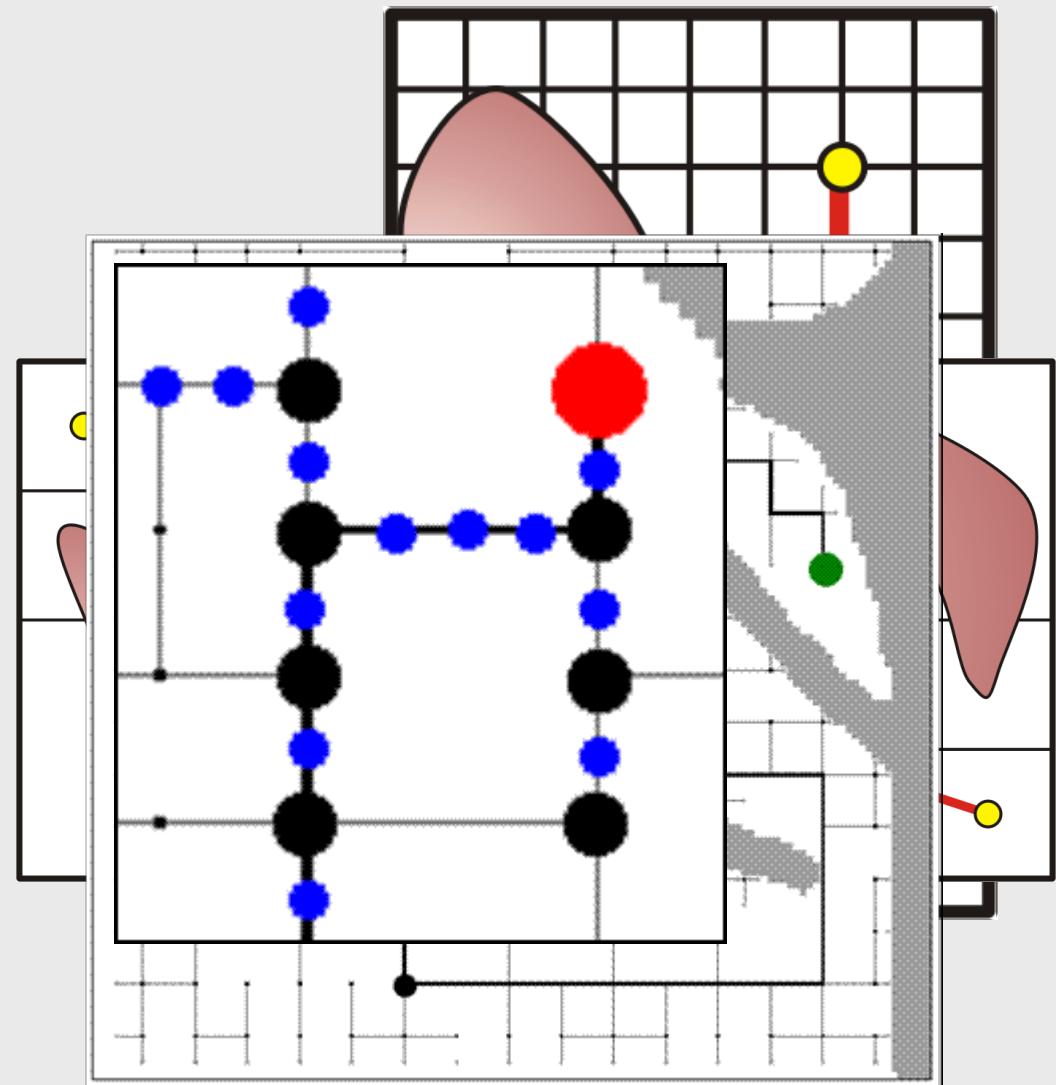


Hierarchical approaches

- ▶ [Warren93,Hocaoglu99]
 - ▶ Serial graduation
 - ▶ Parallel graduation

- ▶ [Zhu91,Yen94,Chen98]
 - ▶ Cell based

- ▶ [Autere99, Isto02]:
 - ▶ Hierarchical grid



Extended A*-Algorithm: Hierarchical A*

Problem:

High complexity of C-space when planning robots with 6-DOF

Solution:

Making steps in a “hierarchical” manner

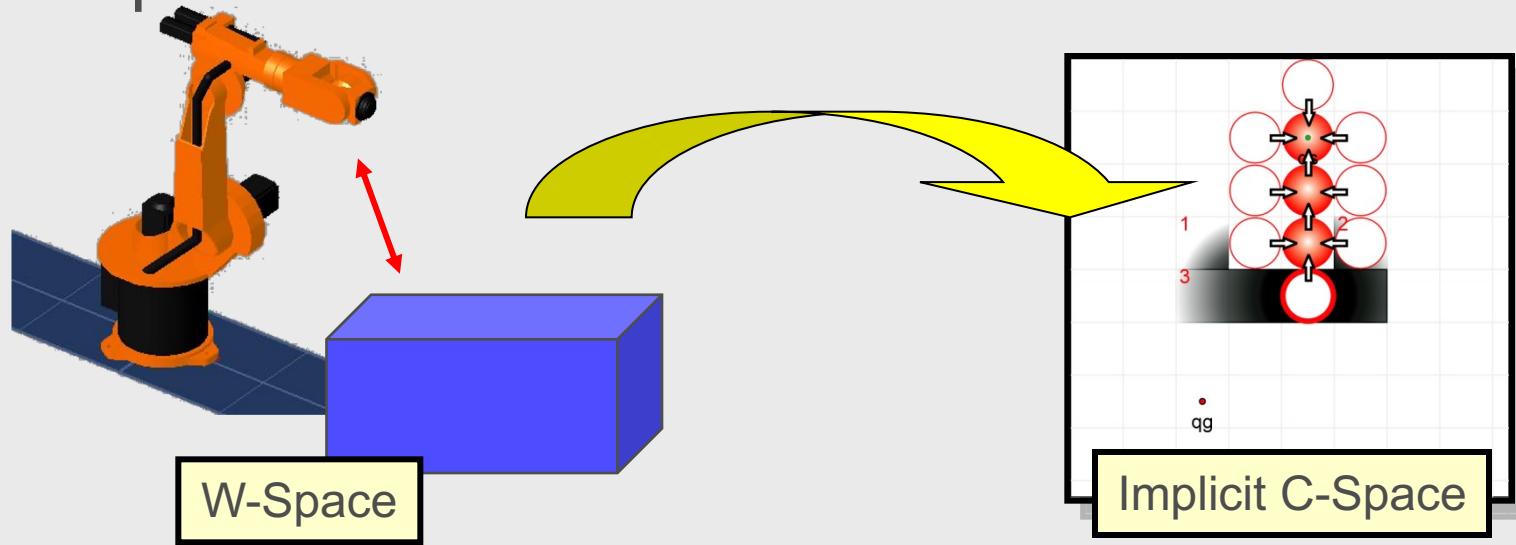
- ▶ Big steps, when much free space
- ▶ Small steps, when obstacles are nearby

How to make big steps? How to know the allowed size of the big steps?

Approximation of distance consumption

Question 1

- ▶ How to transform distances in W-space to distances in C-Space?

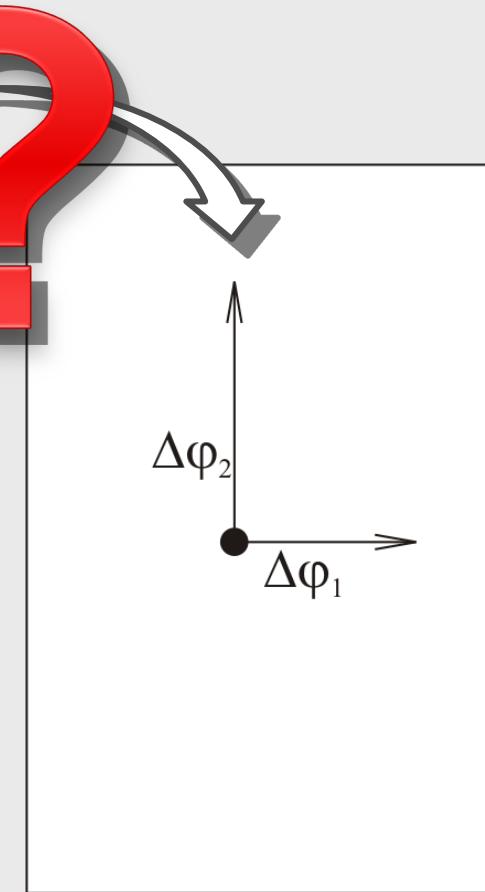
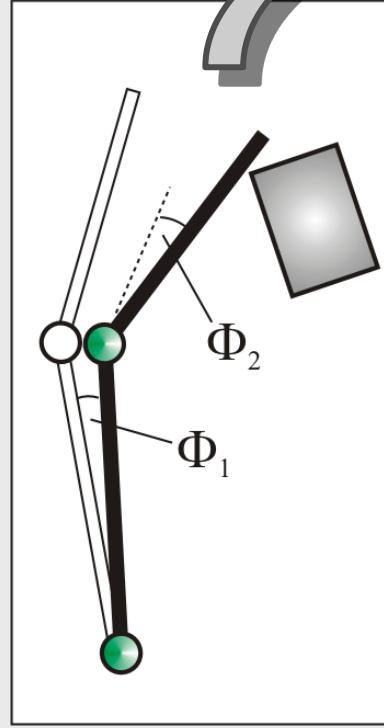
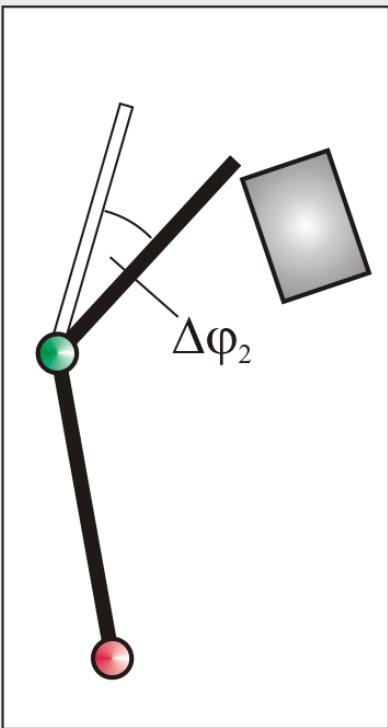
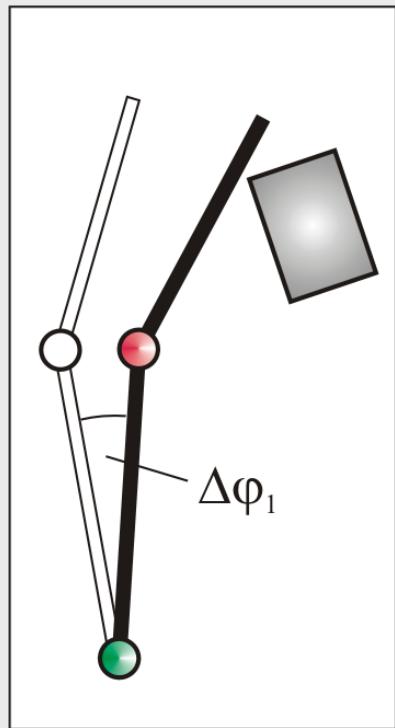


Question 1 can be solved by Question 2

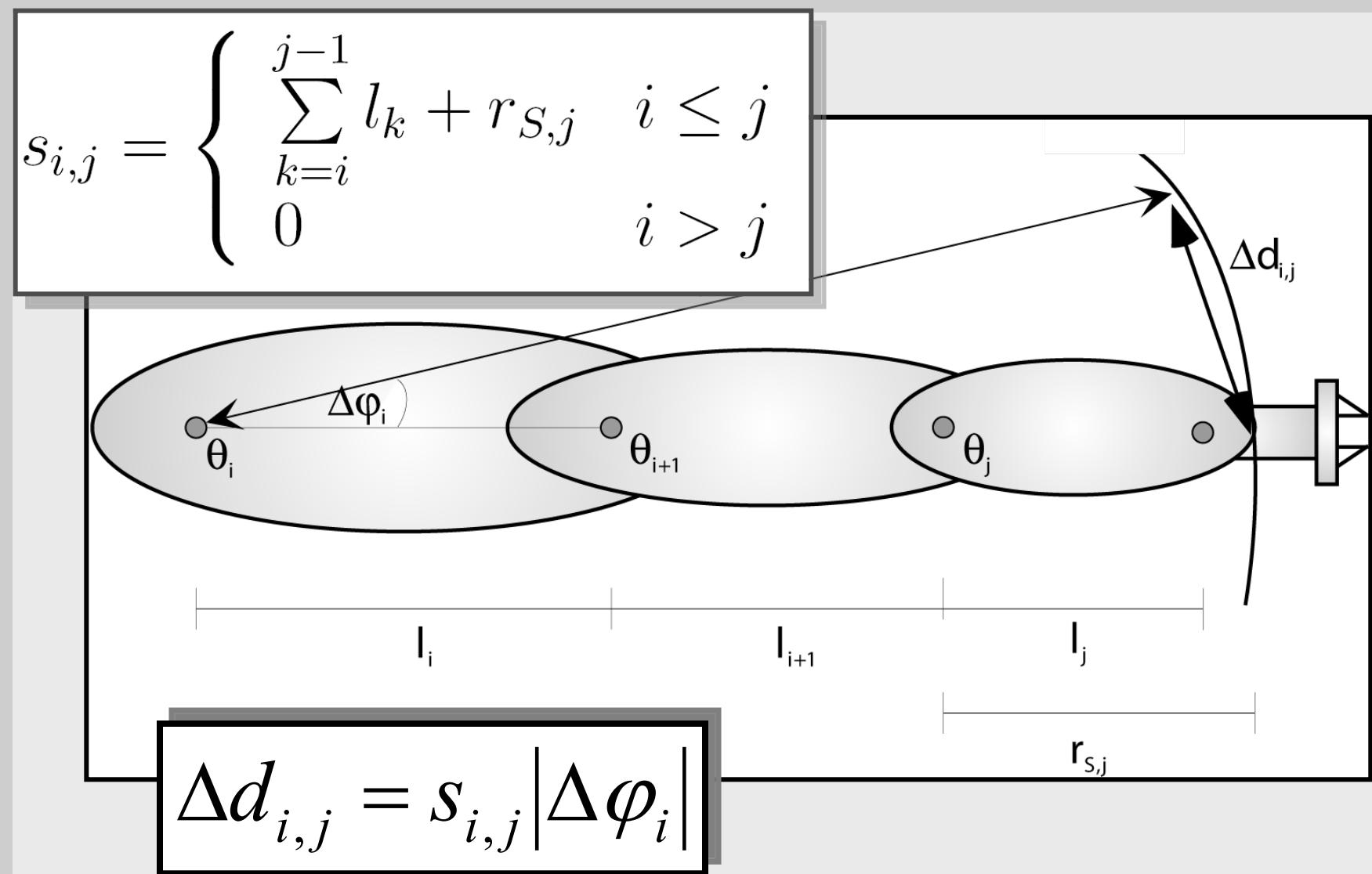
- ▶ How much will robot move in W-space if I move it in C-space?

Approximation of distance consumption

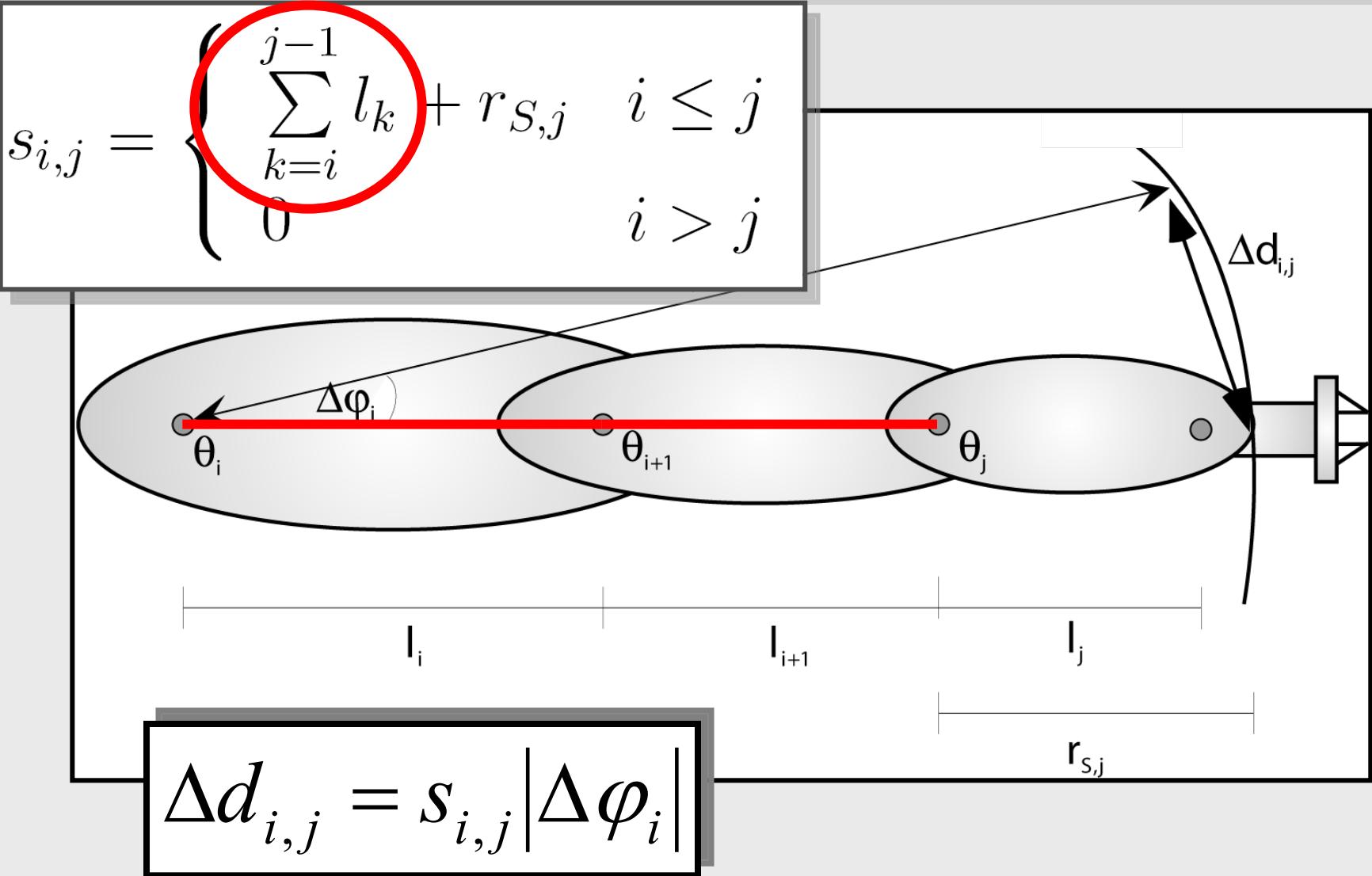
Principle



Approximation of distance consumption: worst case

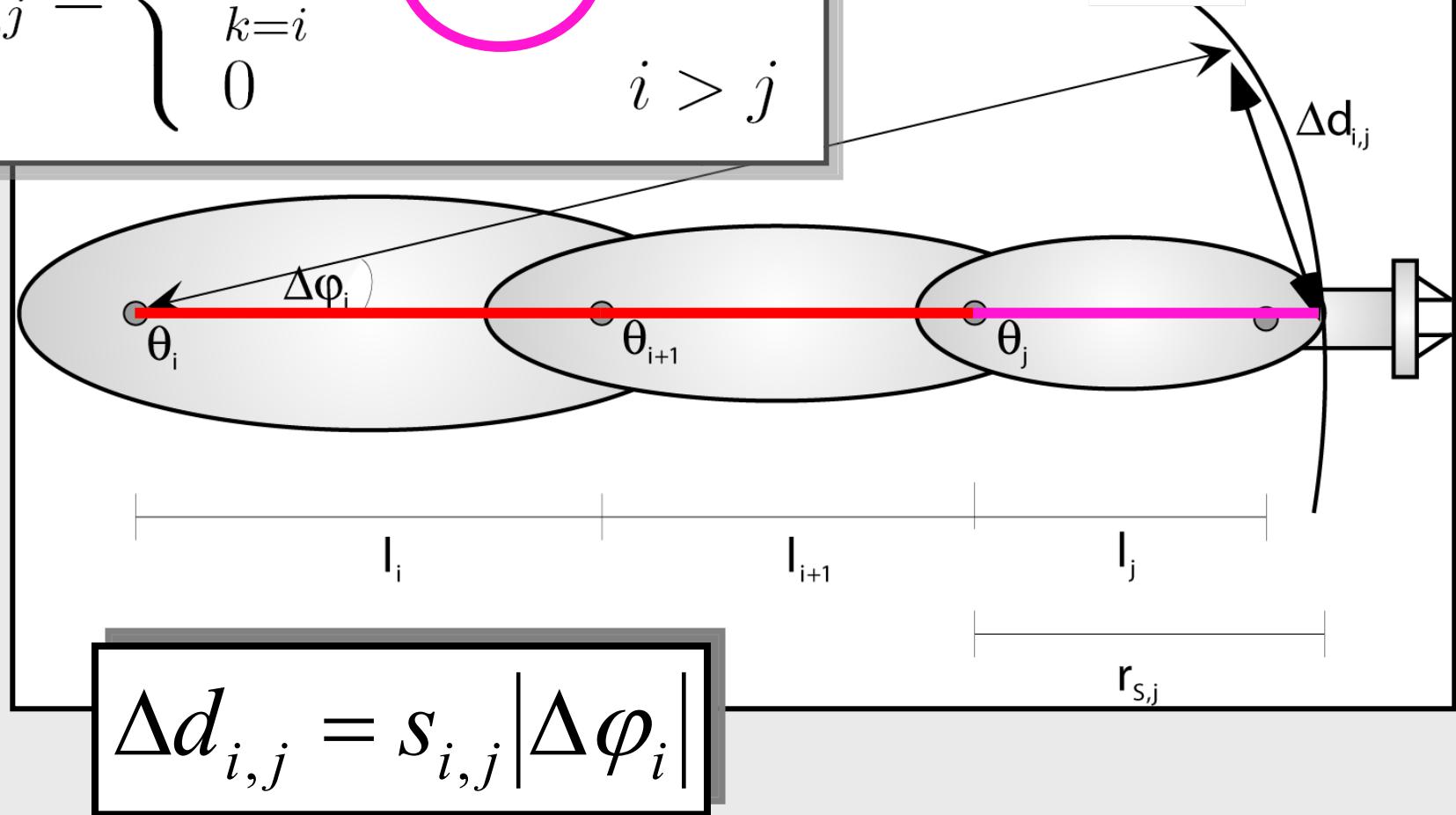


Approximation of distance consumption: worst case



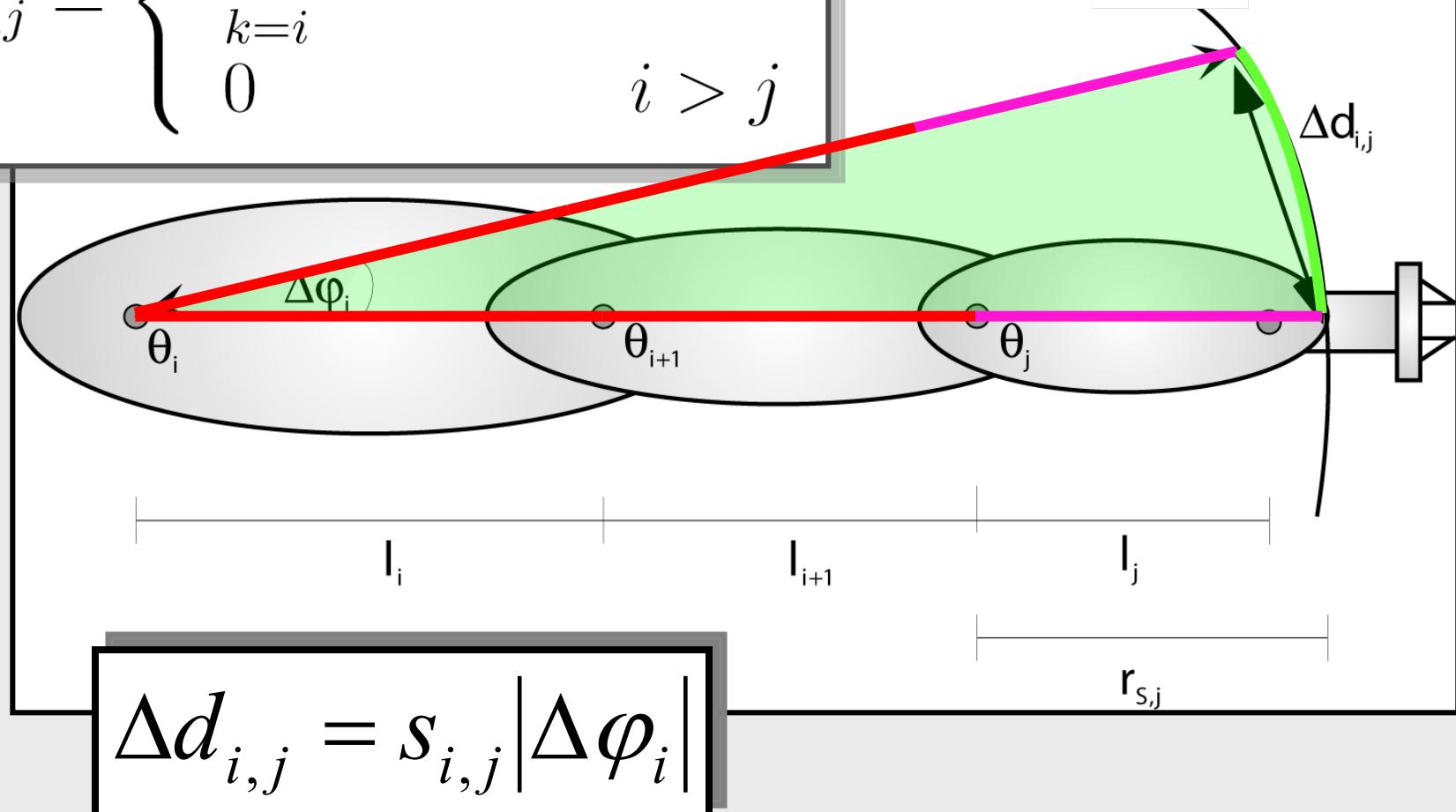
Approximation of distance consumption: worst case

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



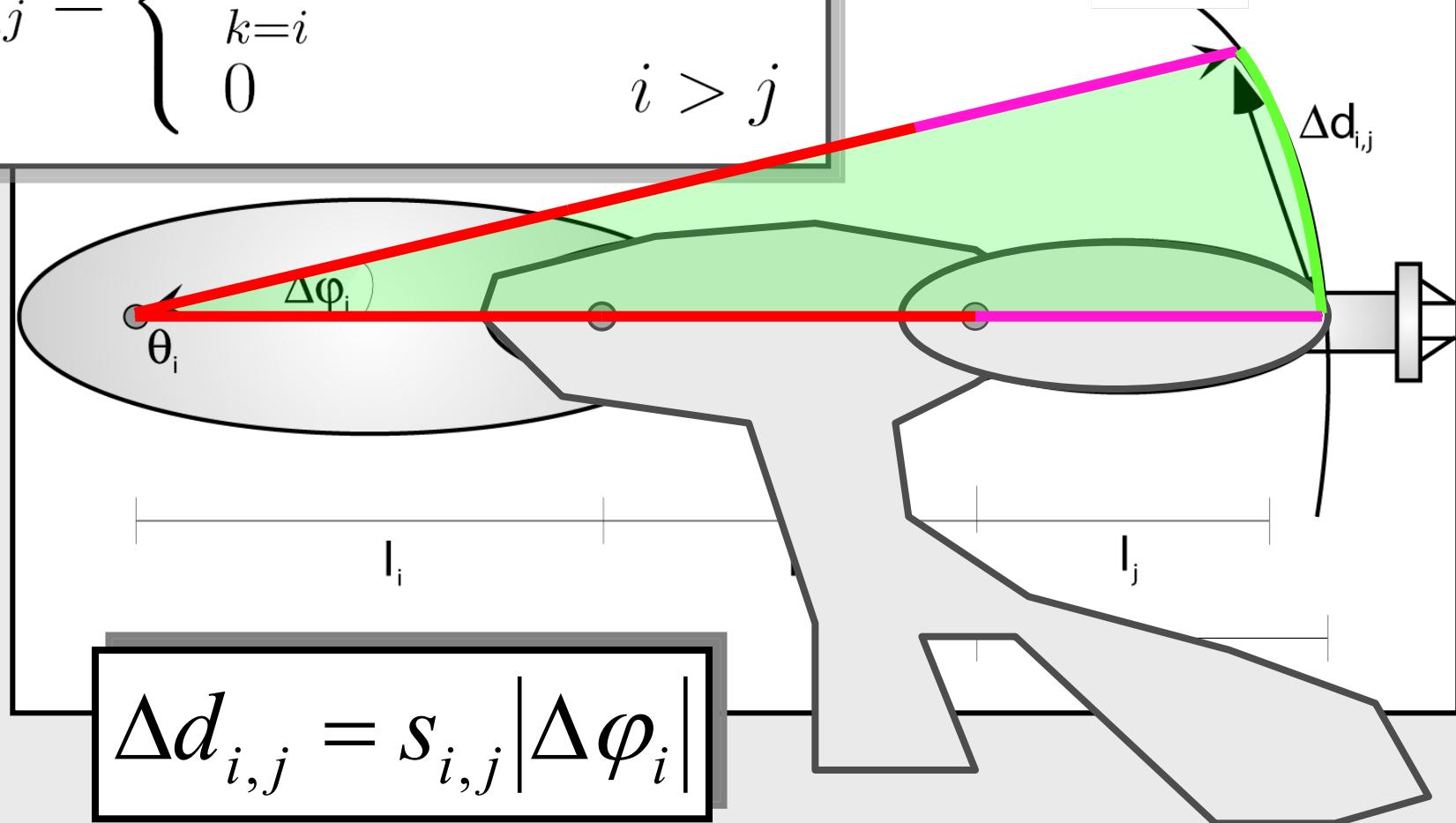
Approximation of distance consumption: worst case

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



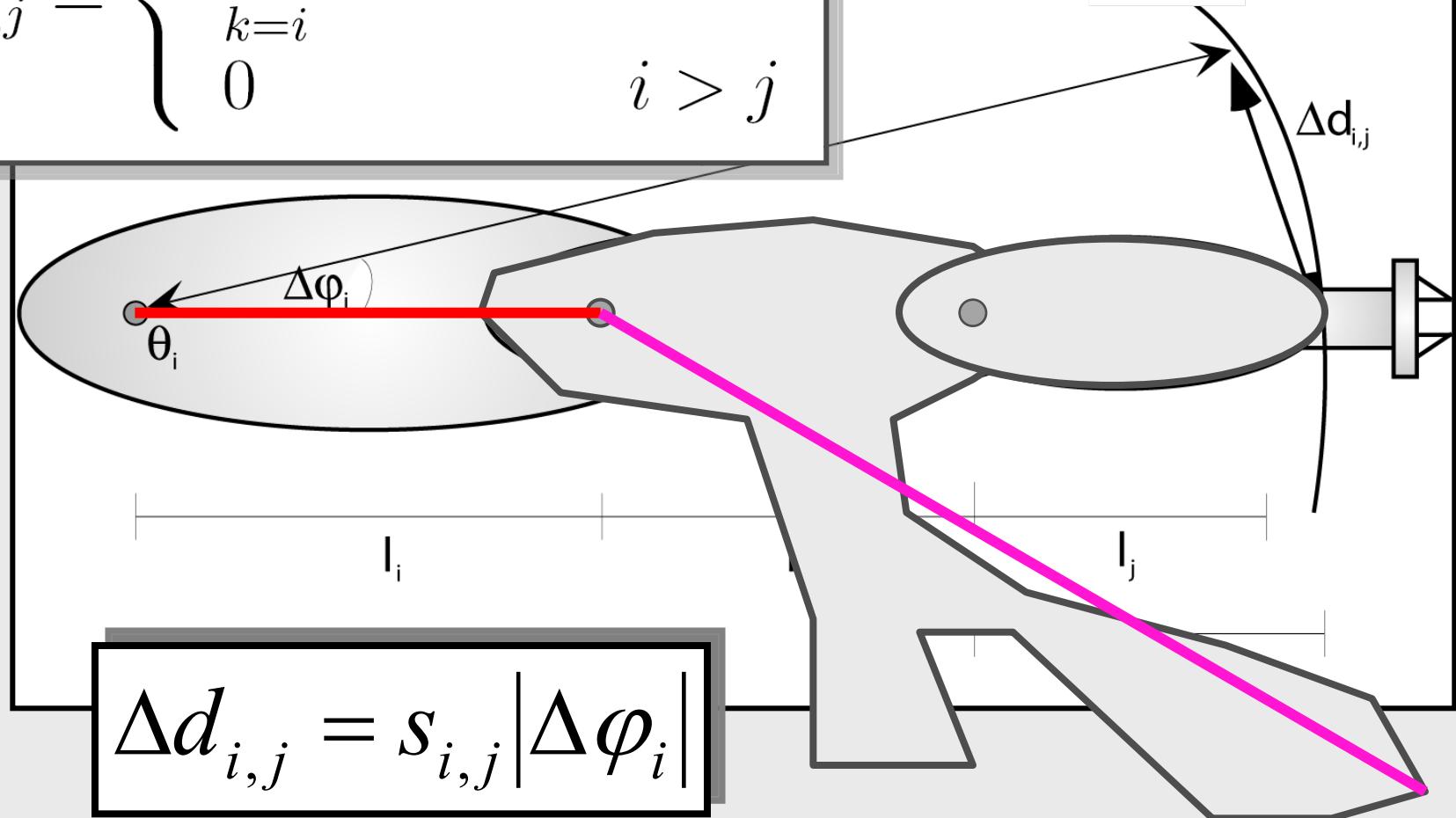
Approximation of distance consumption: worst case („different shape“)

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



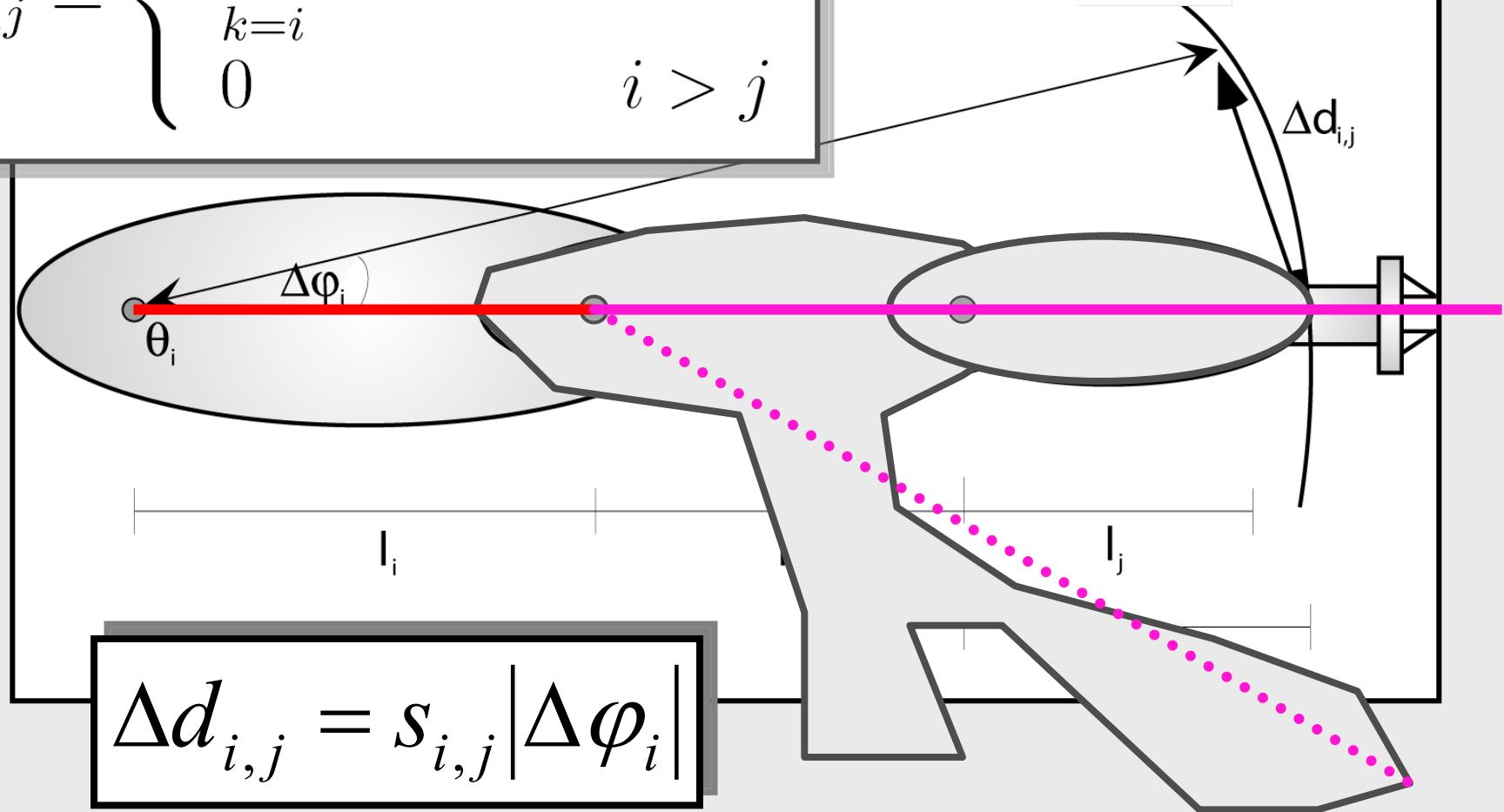
Approximation of distance consumption: worst case („different shape“)

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



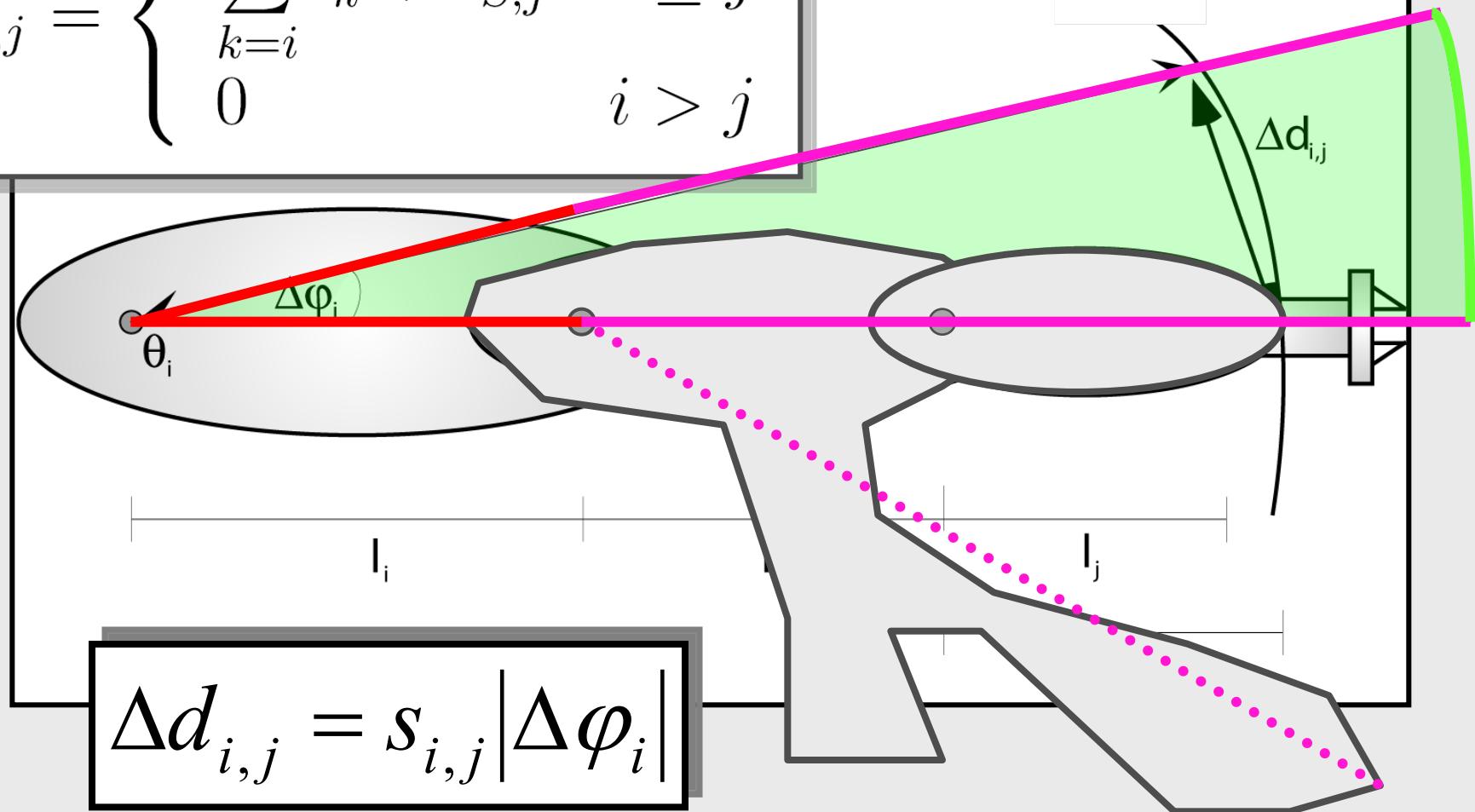
Approximation of distance consumption: worst case („different shape“)

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



Approximation of distance consumption: worst case („different shape“)

$$s_{i,j} = \begin{cases} \sum_{k=i}^{j-1} l_k + r_{S,j} & i \leq j \\ 0 & i > j \end{cases}$$



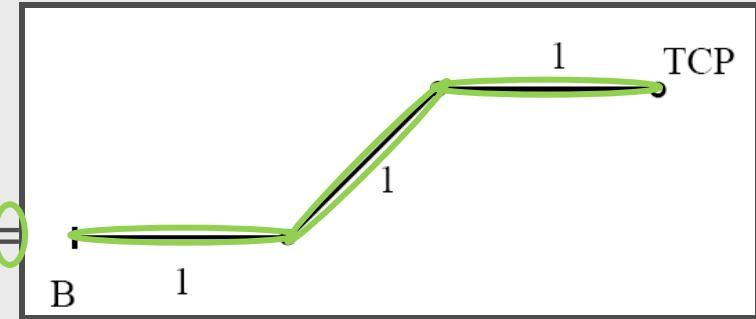
Distance Decrement Table

- ▶ The matrix build by the components $s_{i,j}$ is called **DD-Table (Distance Decrement table)**.
- ▶ Example:

- ▶ Robot with

- ▶ link length = $l_i = 1$, and

- ▶ extension of geometry = $r_{s,j}$



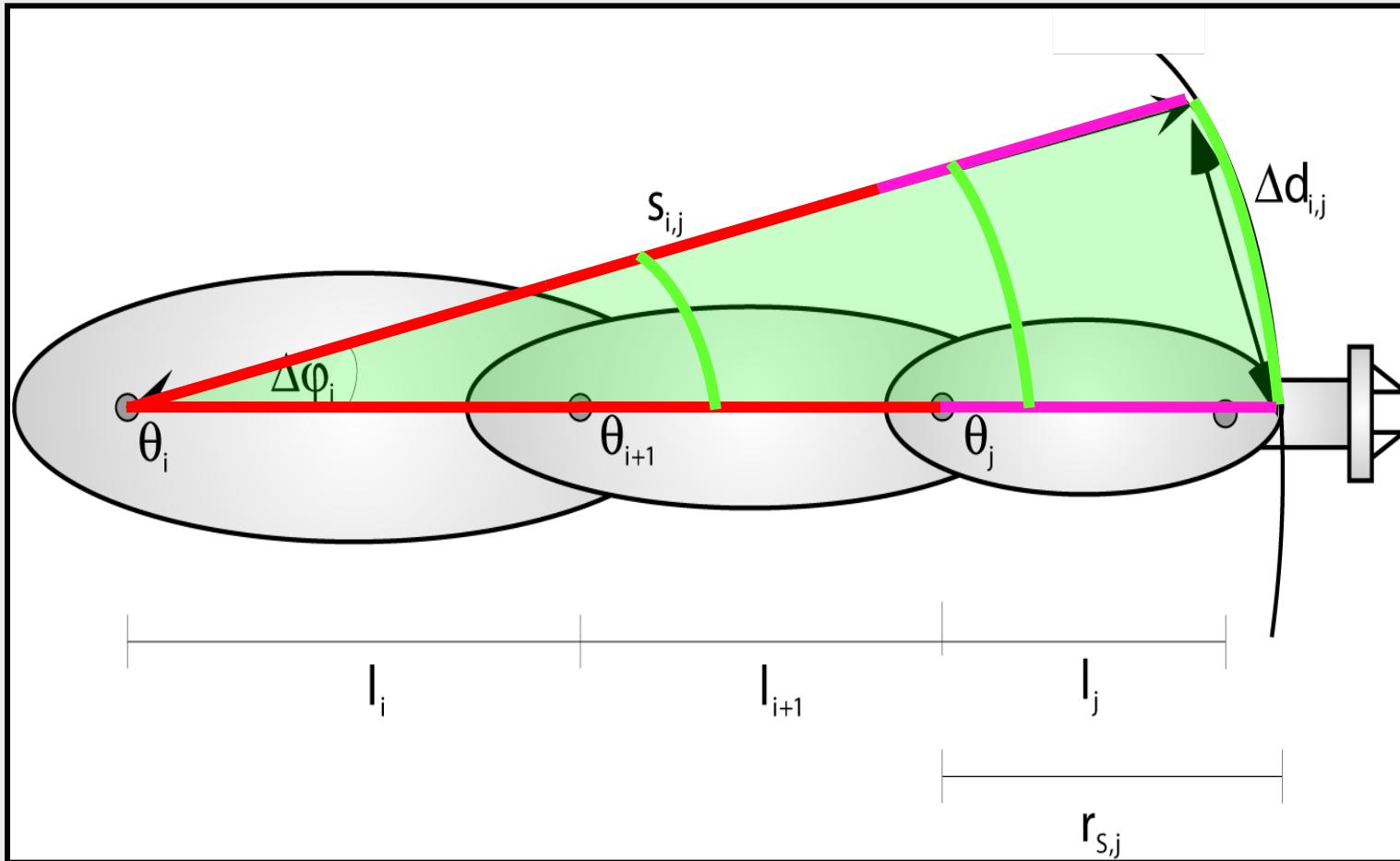
$$DD = [s_{i,j}] = \begin{bmatrix} 0 & 1+1 & 1+1+1 \\ 0 & 1 & 1+1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Computing free space based on DD-Table

- ▶ Movement of $\Delta\varphi_i$ of axis Θ_i causes in every link S_j with $j \geq i$ the distance consumption of $\Delta d_{i,j}$.

Computing free space based on DD-Table

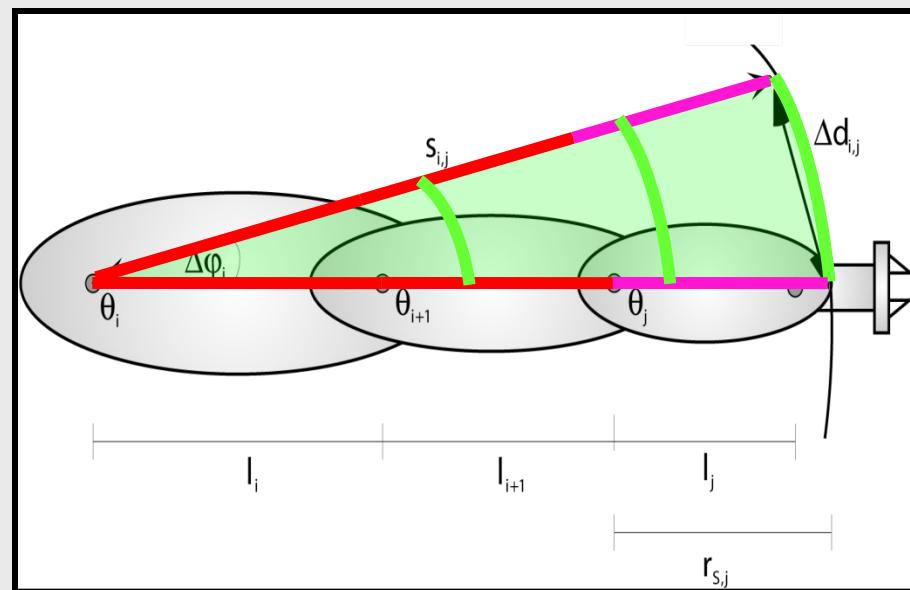
- Movement of $\Delta\varphi_i$ of axis Θ_i causes in every link S_j with $j \geq i$ the distance consumption of $\Delta d_{i,j}$.



Computing free space based on DD-Table

- ▶ Movement of $\Delta\varphi_i$ of axis Θ_i causes in every link S_j with $j \geq i$ the distance consumption of $\Delta d_{i,j}$.
- ▶ If d is computed distance in W-space. It has to be $\Delta d_{i,j} \leq d$ to avoid collision.
- ▶ The maximal possible space for joint i is then:

$$\Delta\varphi_{i\max} = \min_{j \geq i} \left| \frac{d}{S_{i,j}} \right|$$



Computing free space based on DD-Table

- ▶ Movement of $\Delta\varphi_i$ of axis Θ_i causes in every link S_j with $j \geq i$ the distance consumption of $\Delta d_{i,j}$.
- ▶ If d is computed distance in W-space. It has to be $\Delta d_{i,j} \leq d$ to avoid collision.
- ▶ The maximal possible space for joint i is then:

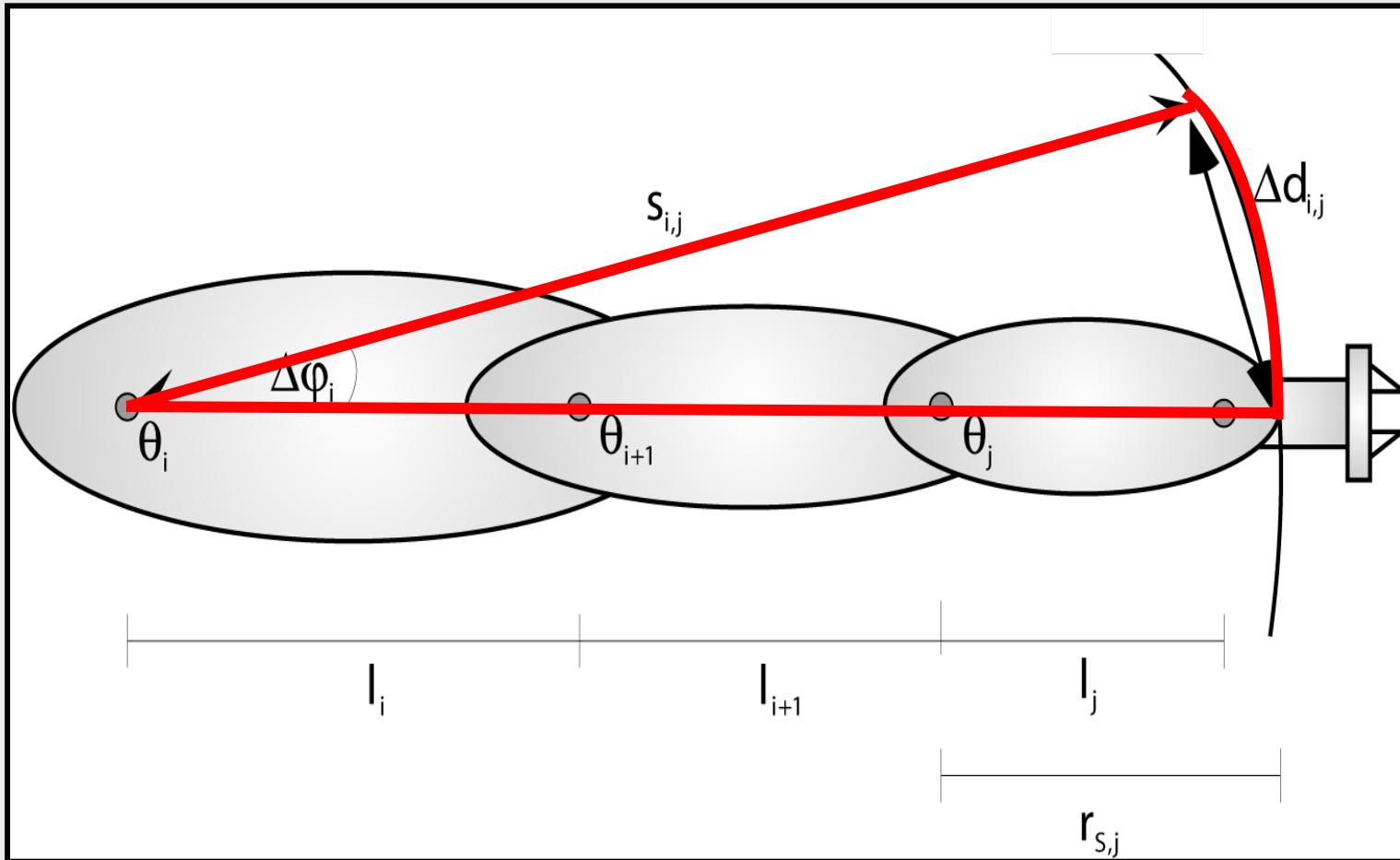
$$\Delta\varphi_{i\max} = \min_{j \geq i} \left| \frac{d}{s_{i,j}} \right|$$

- ▶ If several joints are moved distance consumption Δd_j of link S_j is:

$$\Delta d_j = \sum_{i=1}^j \Delta d_{i,j} = \sum_{i=1}^j s_{i,j} |\Delta\varphi_i| \quad \text{Keep in mind } s_{i,j} = 0 \text{ if } i > j$$

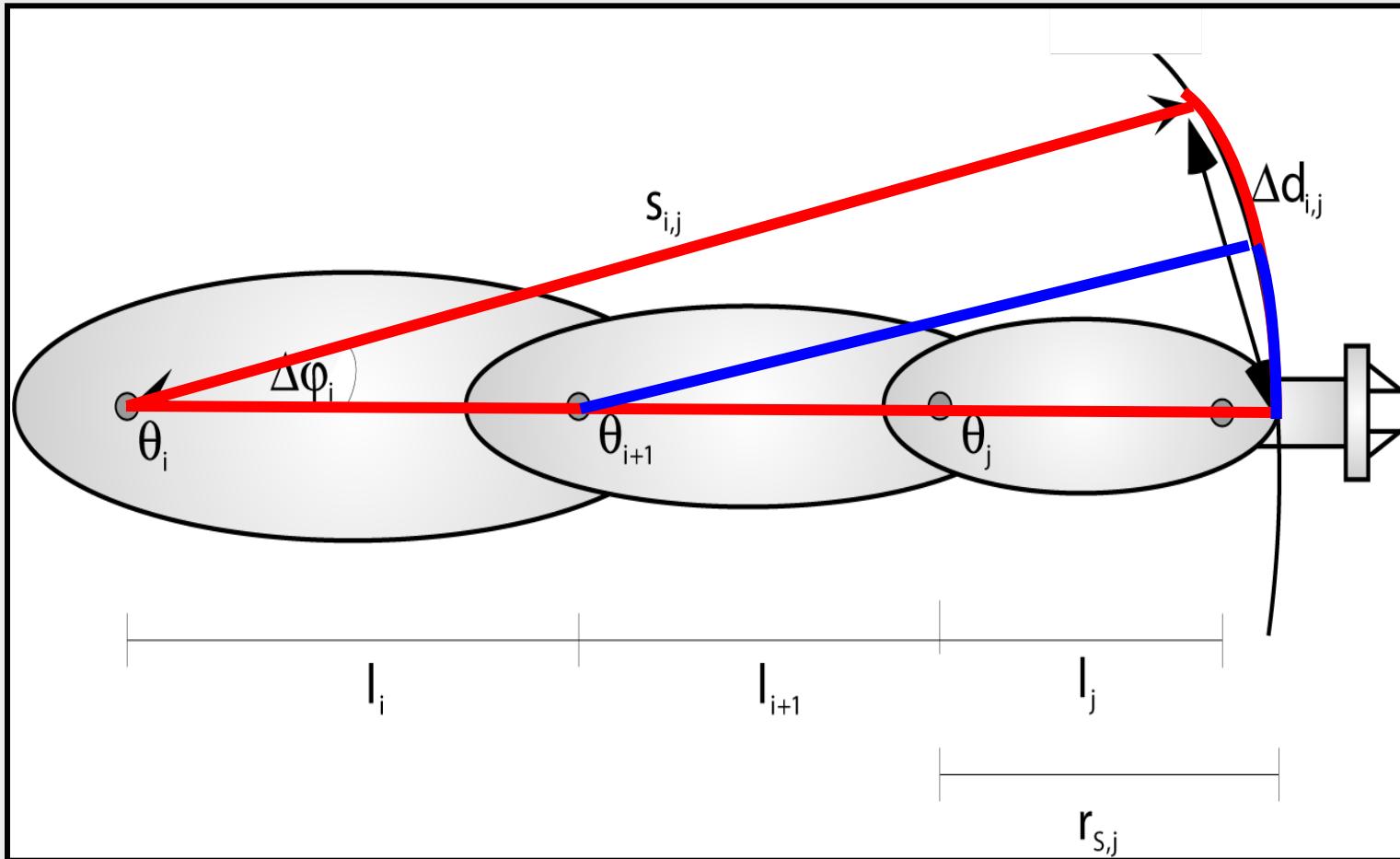
Several joints are moved

- ▶ Joint 1 moved



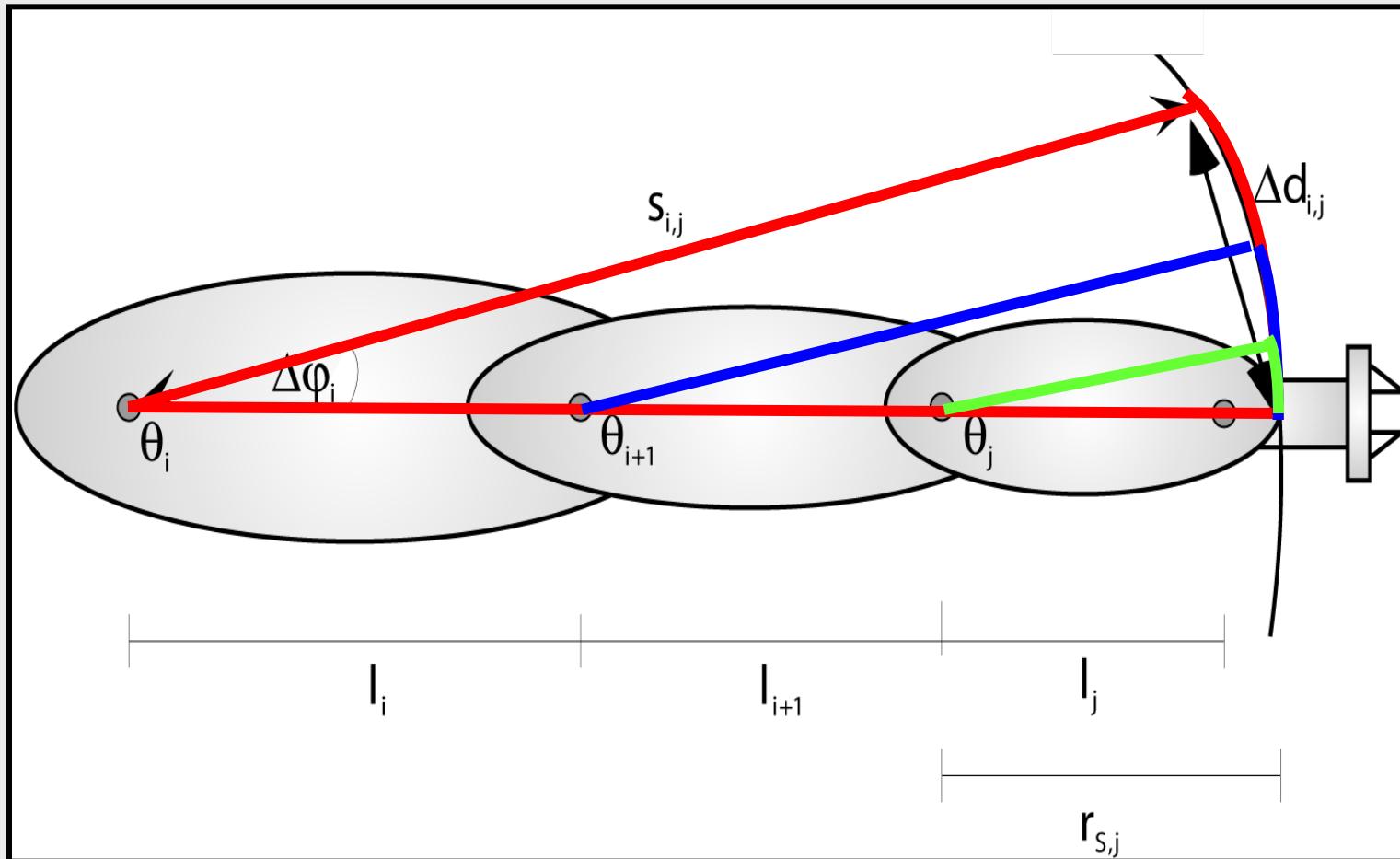
Several joints are moved

- Joint 1 and Joint 2 moved



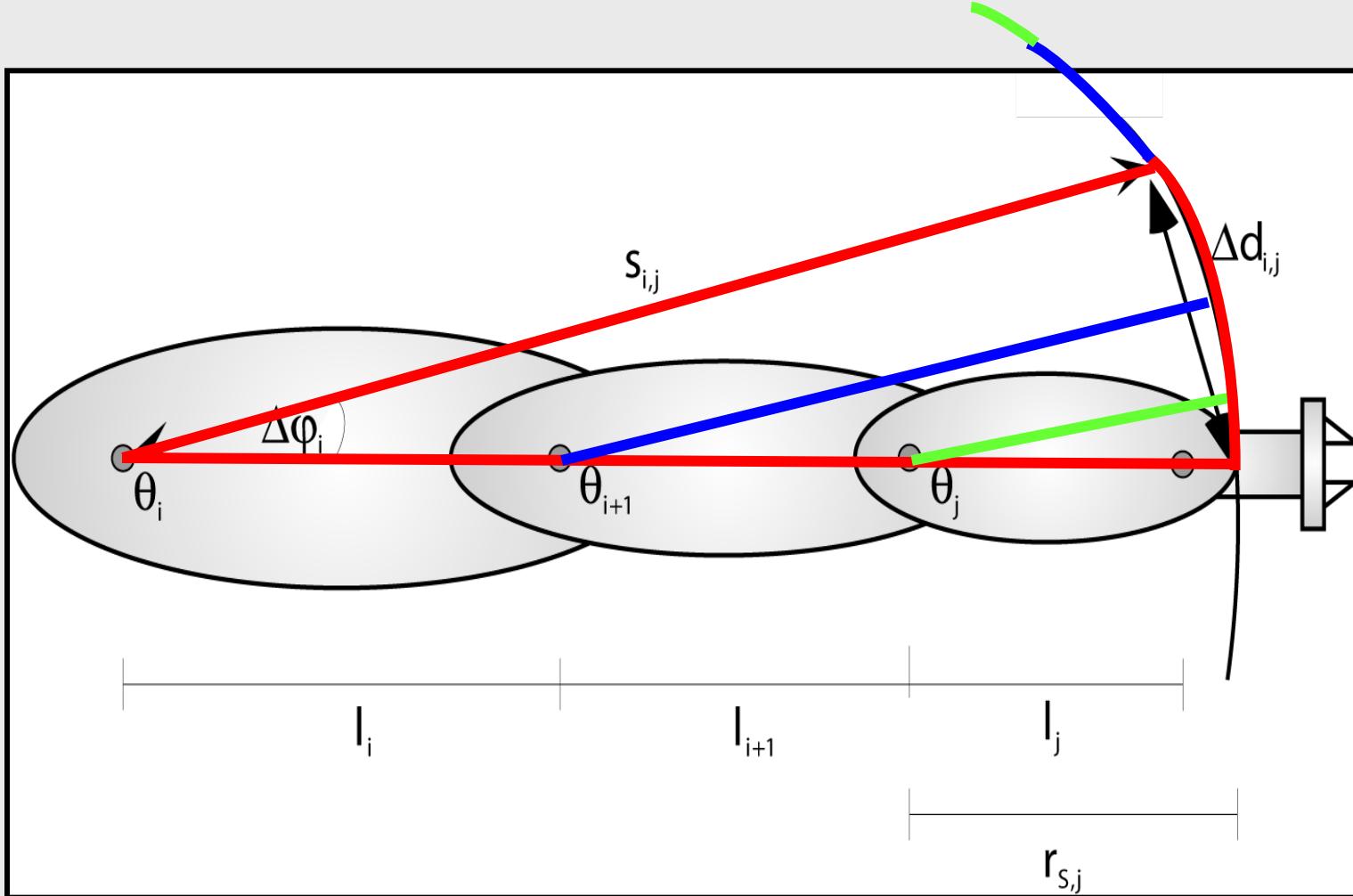
Several joints are moved

- Joint 1, Joint 2 and Joint 3 moved



Several joints are moved

- ▶ Add all distances



Computing free space based on DD-Table

- ▶ Estimate upper bound for distance consumption of link S_j :

$$\Delta d_j = \sum_{i=1}^j \Delta d_{i,j} = \sum_{i=1}^j s_{i,j} |\Delta \varphi_i| = \sum_{i=1}^j s_{i,j} \frac{|\Delta \varphi_i|}{|\Delta \varphi_{i \max}|} |\Delta \varphi_{i \max}|$$

$$\leq \sum_{i=1}^j s_{i,j} \frac{|\Delta \varphi_i|}{|\Delta \varphi_{i \max}|} \frac{d}{s_{i,j}} = d \sum_{i=1}^j \frac{|\Delta \varphi_i|}{|\Delta \varphi_{i \max}|} = \hat{\Delta d}_j$$


$$\Delta \varphi_{i \max} = \min_{j \geq i} \left| \frac{d}{s_{i,j}} \right|$$

Computing free space based on DD-Table

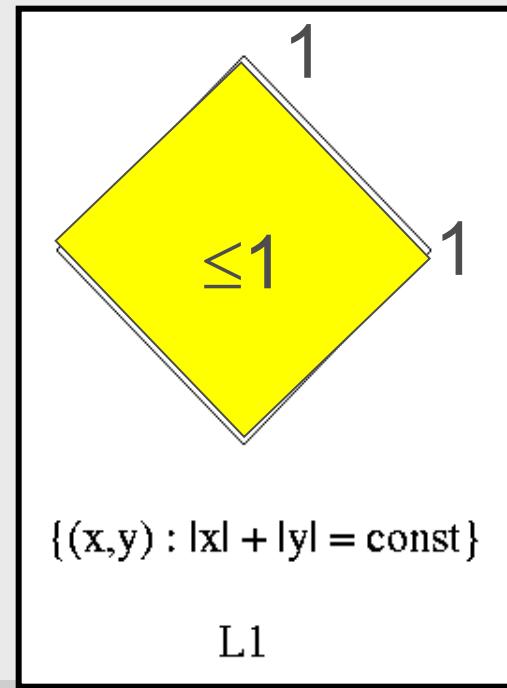
- To avoid collision, Δd_j for all links S_j has to be smaller than d . Therefore:

$$\hat{\Delta d}_j = d \sum_{i=1}^j \frac{|\Delta\varphi_i|}{|\Delta\varphi_{i \max}|} \leq d \Leftrightarrow \sum_{i=1}^j \frac{|\Delta\varphi_i|}{|\Delta\varphi_{i \max}|} \leq 1$$

- Remember L1-metric?

$$\sum_{i=1}^j \frac{|\Delta\varphi_i|}{|\Delta\varphi_{i \max}|} \leq 1$$

is a „circle“ with respect to
L1-metric and looks like →
A rhombe



Applying results to the grid

- ▶ The result of transforming W-space distances into C-space, has now only to be transformed on to the discretized grid →
- ▶ As we know, the transformation for joint values to grid coordinates is:

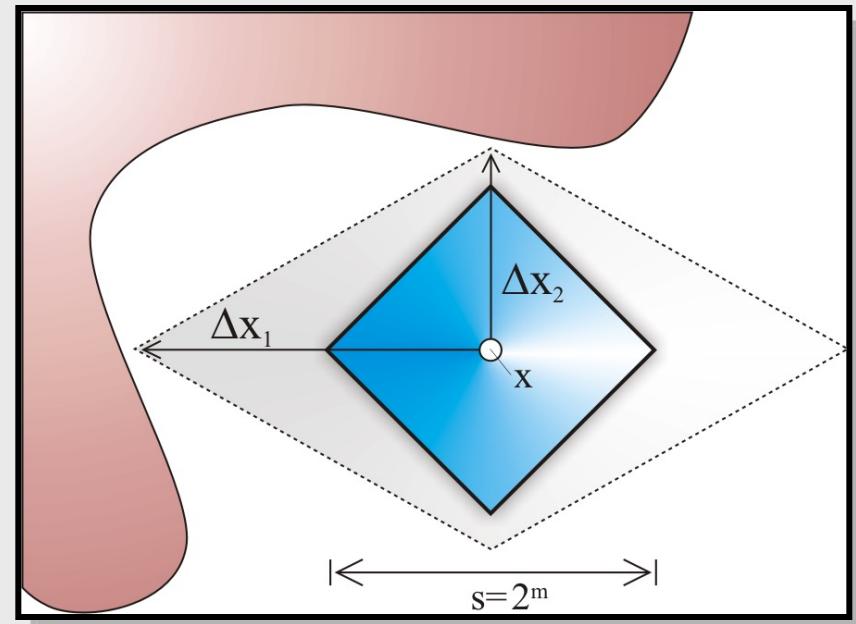
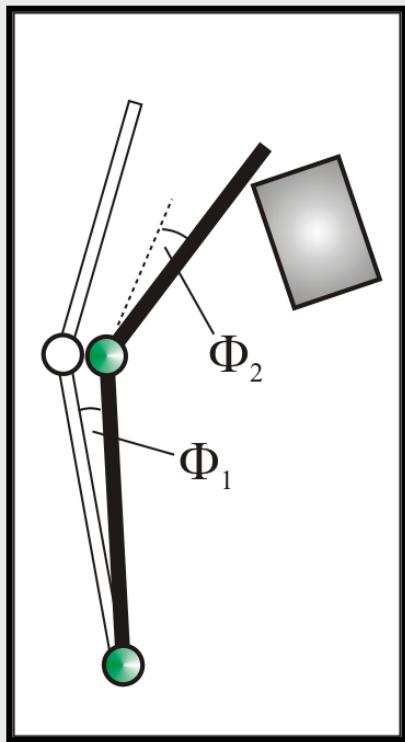
$$x_i = \left\lfloor \frac{\varphi_i - \Theta_{\min,i}}{\Delta\Theta_i} + \frac{1}{2} \right\rfloor$$

- ▶ Therefore:
$$\Delta x_{i \max} = \frac{\varphi_{i \max}}{\Delta\Theta_i}$$
- ▶ And every grid point p is exactly then in free space of x_i if:

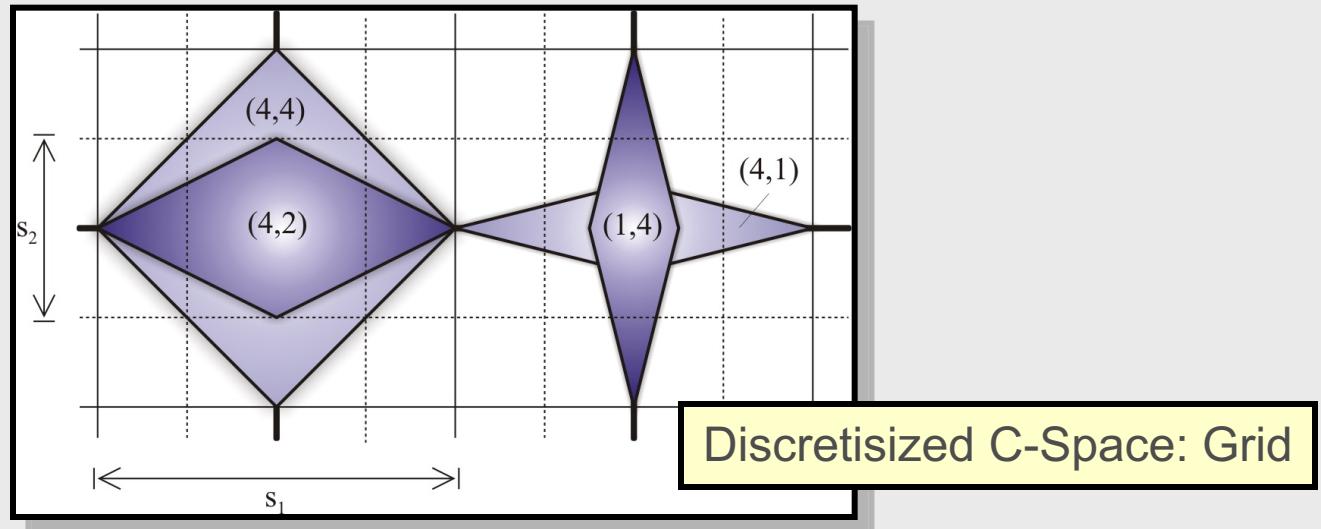
$$\sum_{i=1}^j \frac{|p - x_i|}{|\Delta x_{i \max}|} \leq 1$$

Conclusion

- ▶ A distance computed in W-space can be transformed in a rhomb shaped range in C-space



Size of nodes = stages

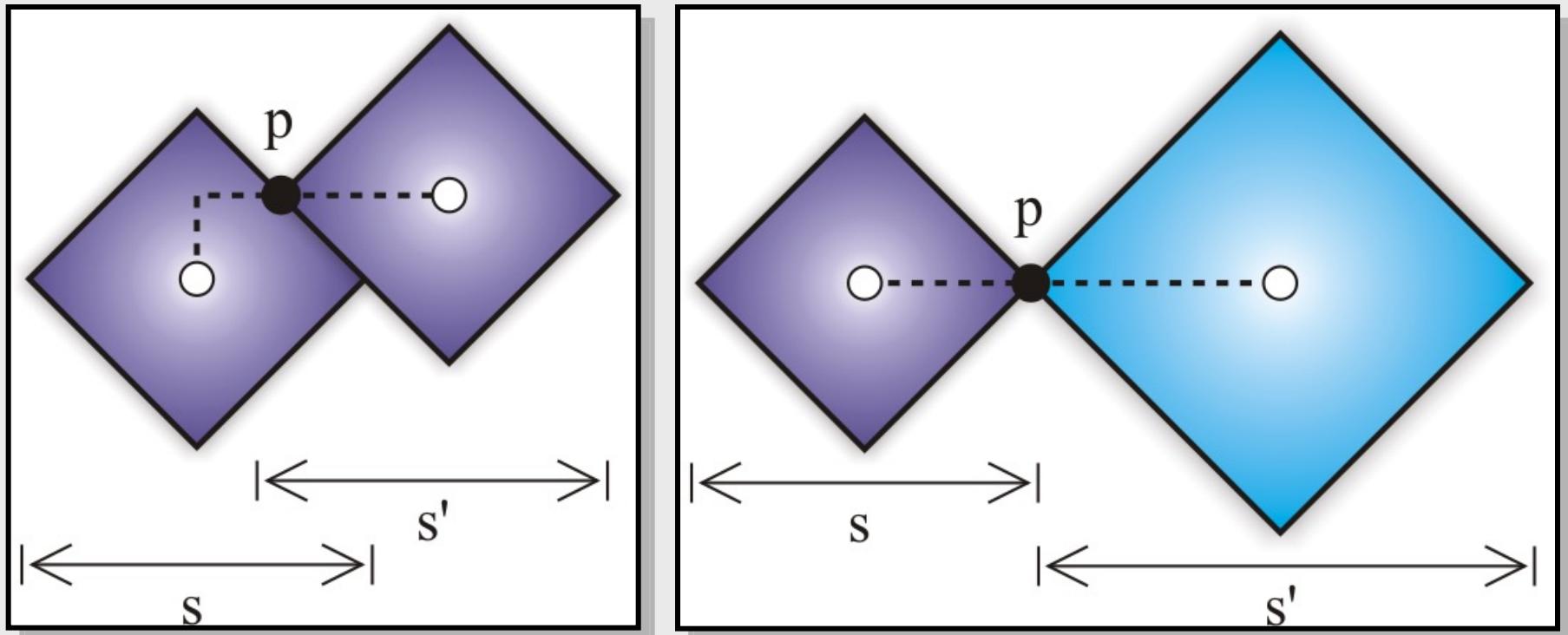


- ▶ The **smallest node** (stage 1) on the grid has size 1.
- ▶ Each following stage double size (1,2,4,8,...)
- ▶ Every direction has it's own stage m_i
- ▶ Stage m_i can be calculated by

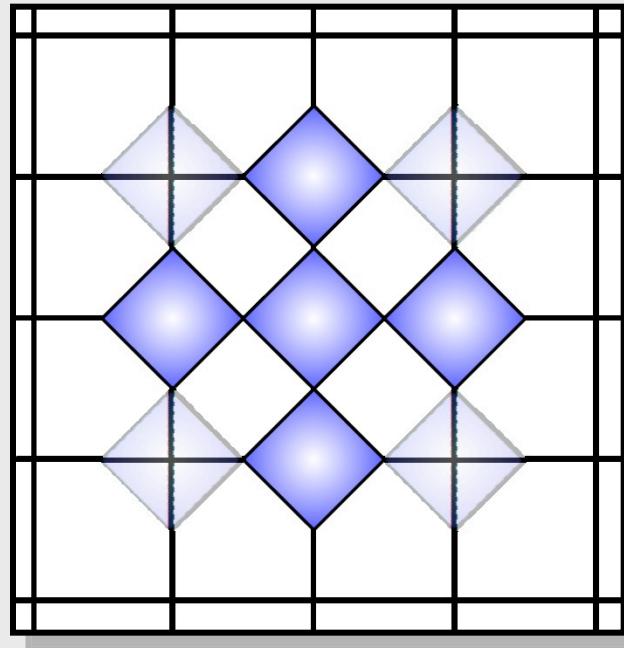
$$m_i = \left\lfloor 1 + \log_2 \Delta x_{i \max} \right\rfloor = \left\lfloor 1 + \log_2 \frac{\Delta \varphi_{i \max}}{\Delta \Theta_i} \right\rfloor$$

Connectivity of nodes

- ▶ Path is collision-free, if it is entirely inside the nodes.
- ▶ Best exploitation when nodes are connected tip on tip.



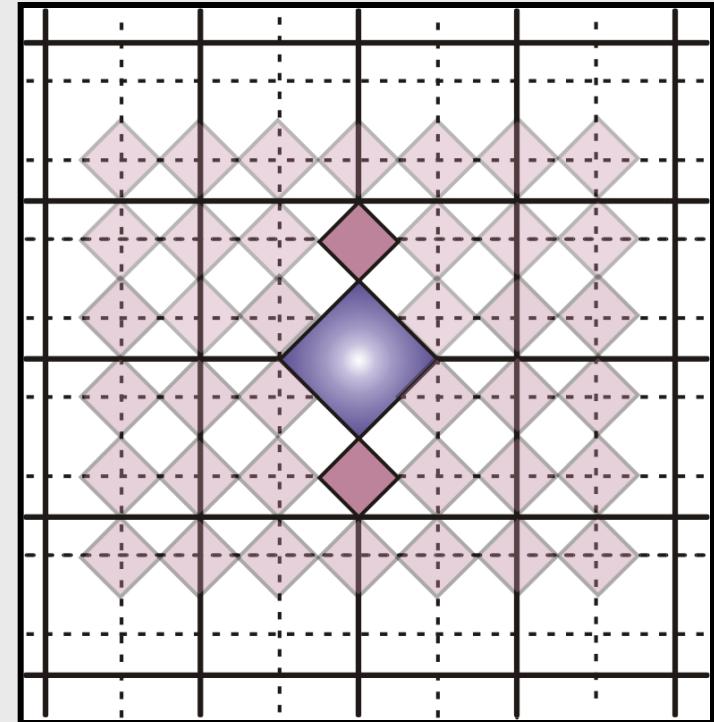
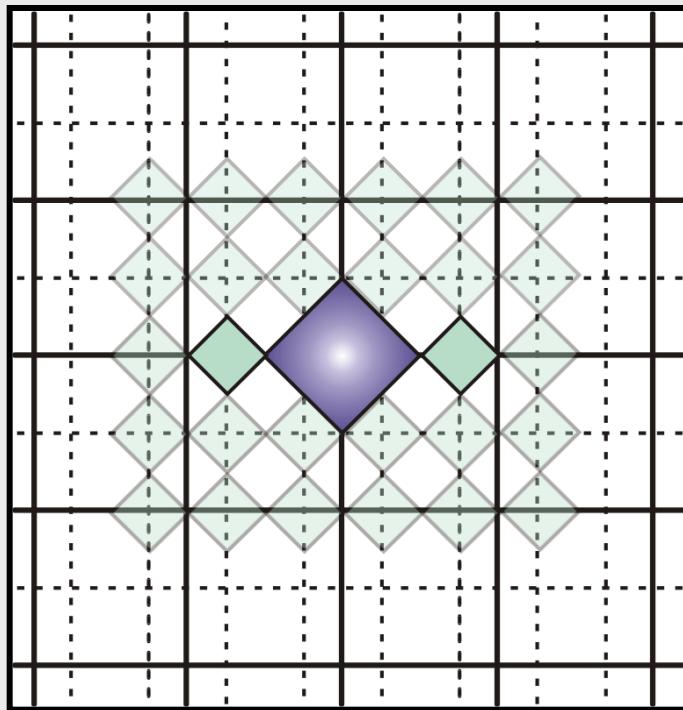
► A* without hierarchy



Connectivity: Problems

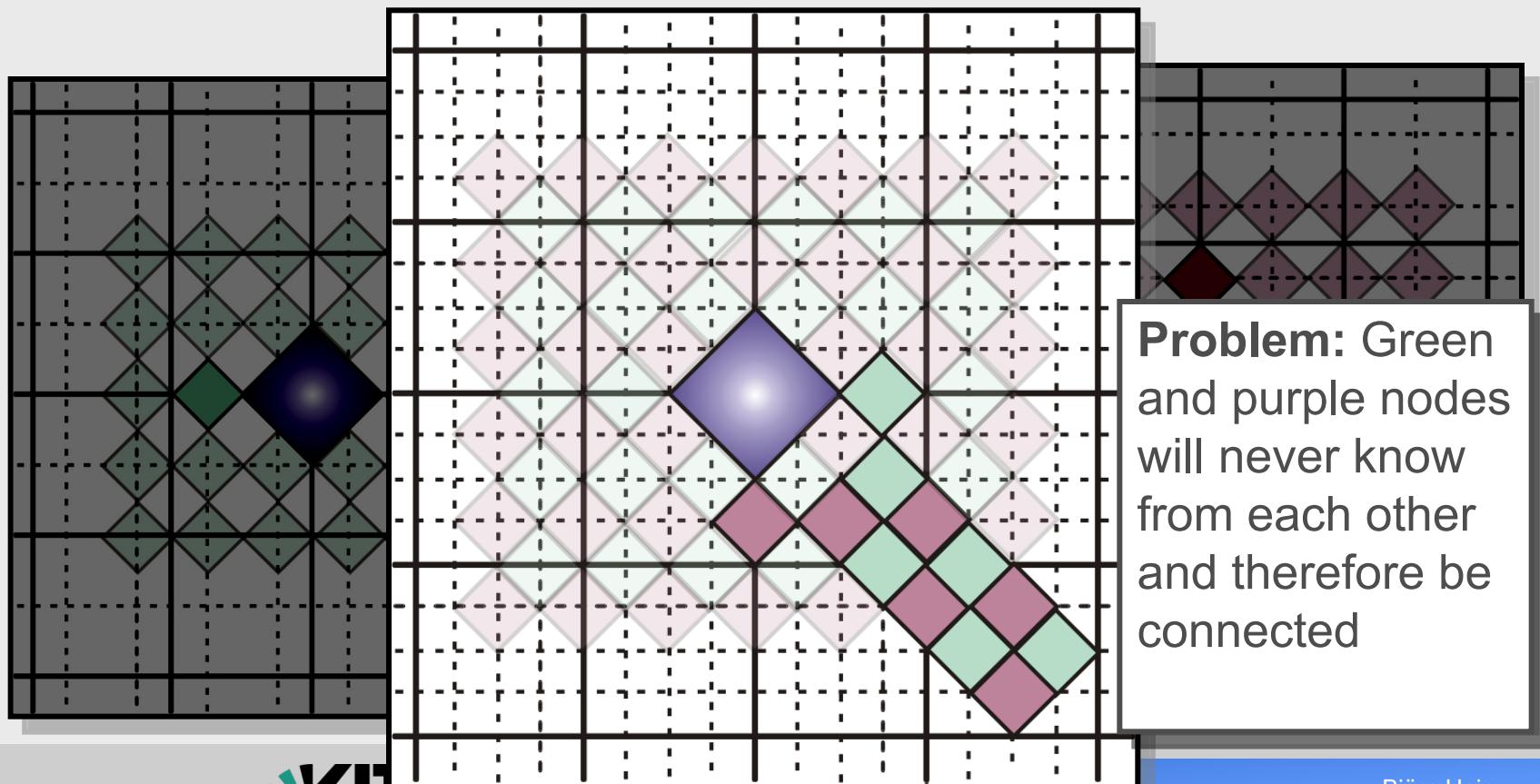
► First idea

- Directly connect the nodes corresponding to their size tip on tip

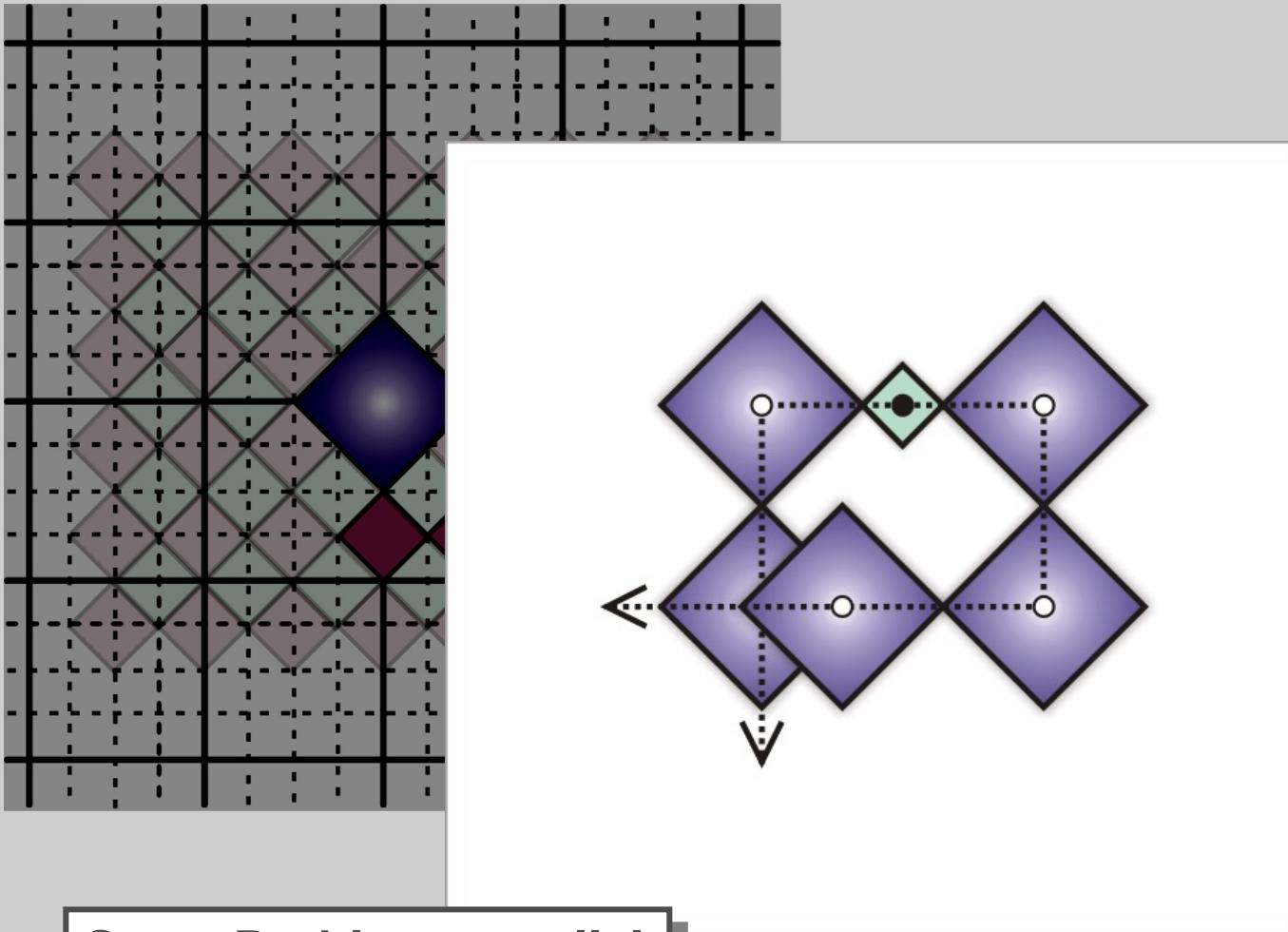


Connectivity: Problems

- ▶ First idea
 - ▶ Directly connect the nodes corresponding to their size tip on tip
- ▶ Problem
 - ▶ Parallel search front reducing efficiency

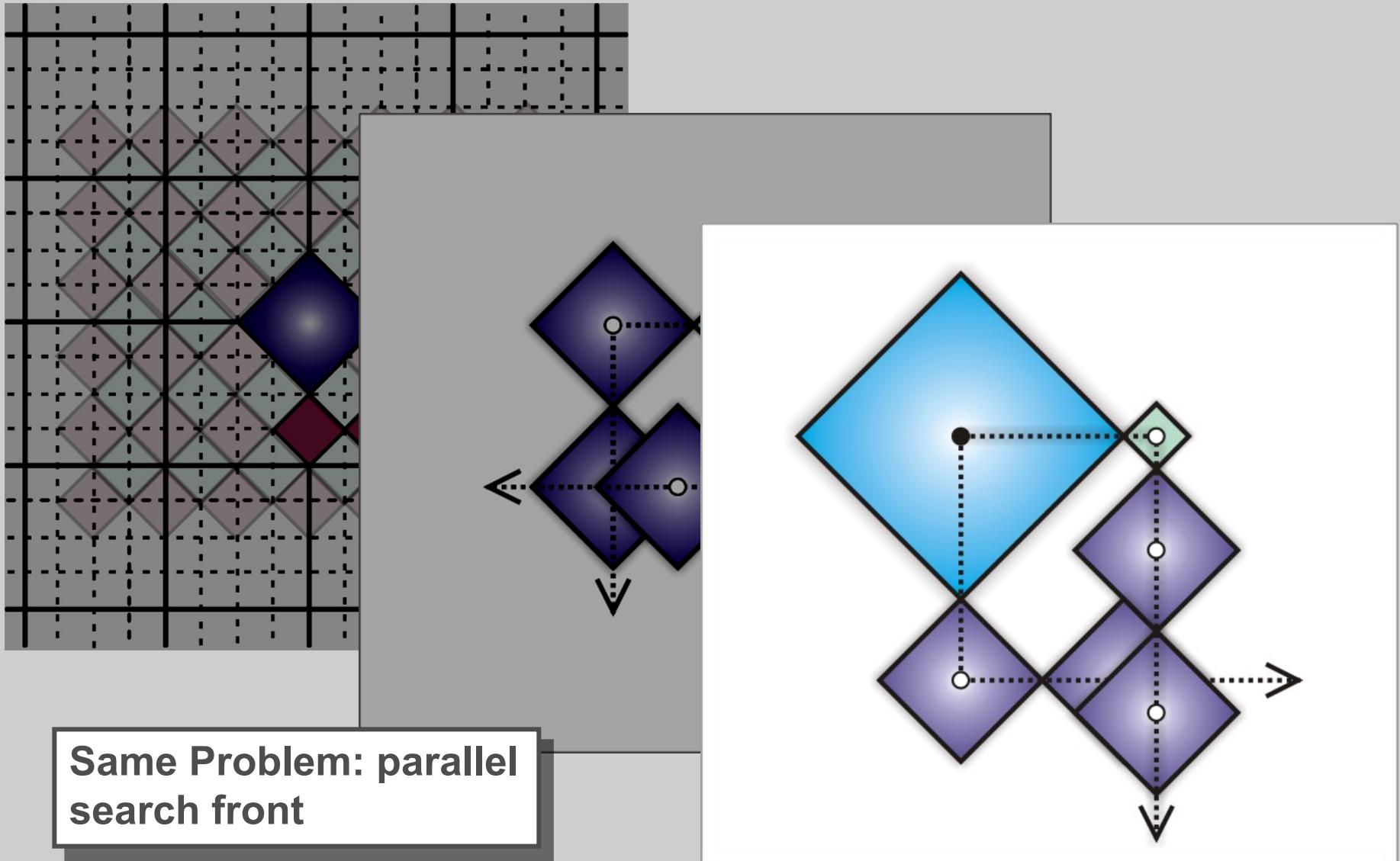


Connectivity: Problem

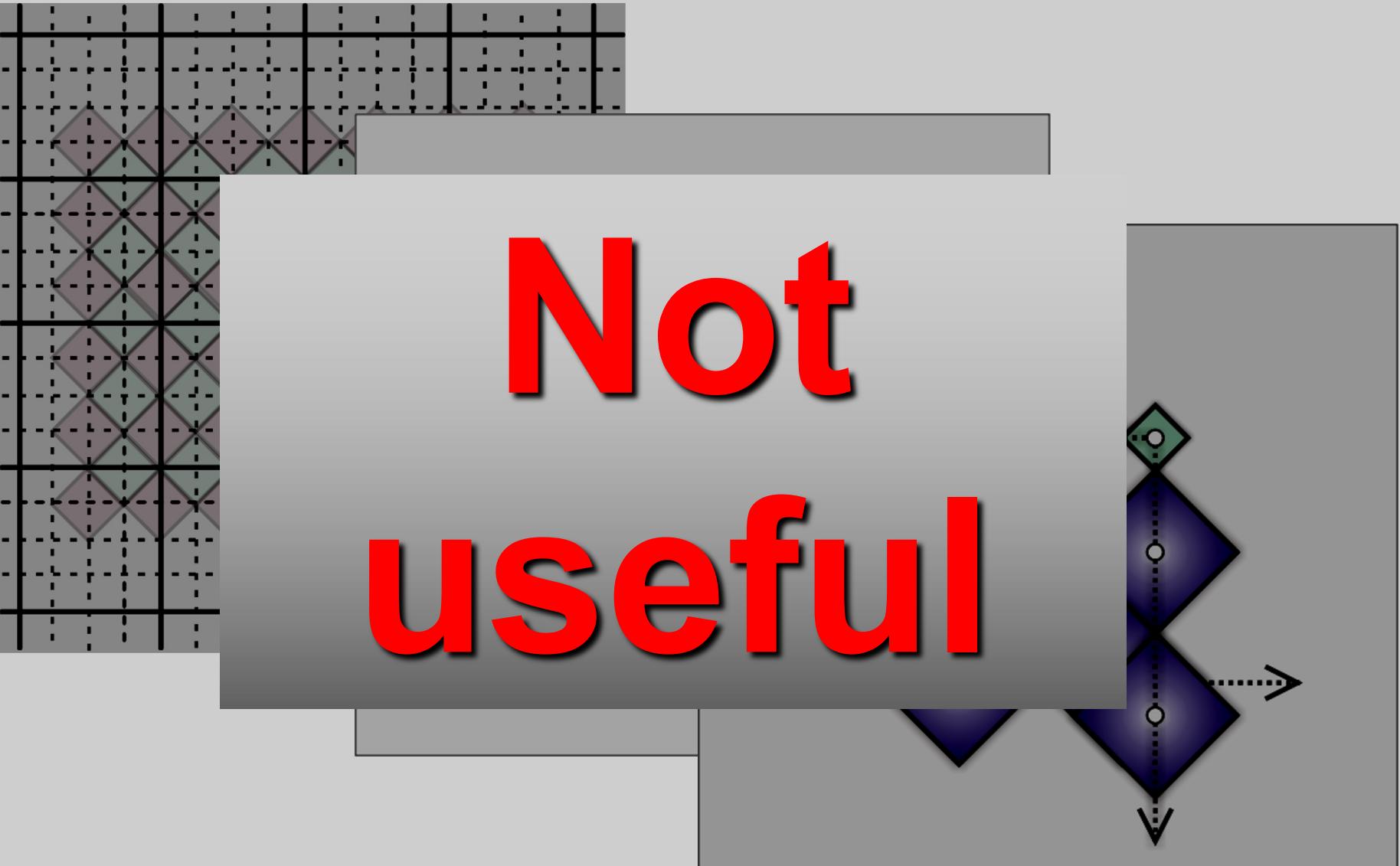


Same Problem: parallel
search front

Connectivity: problem

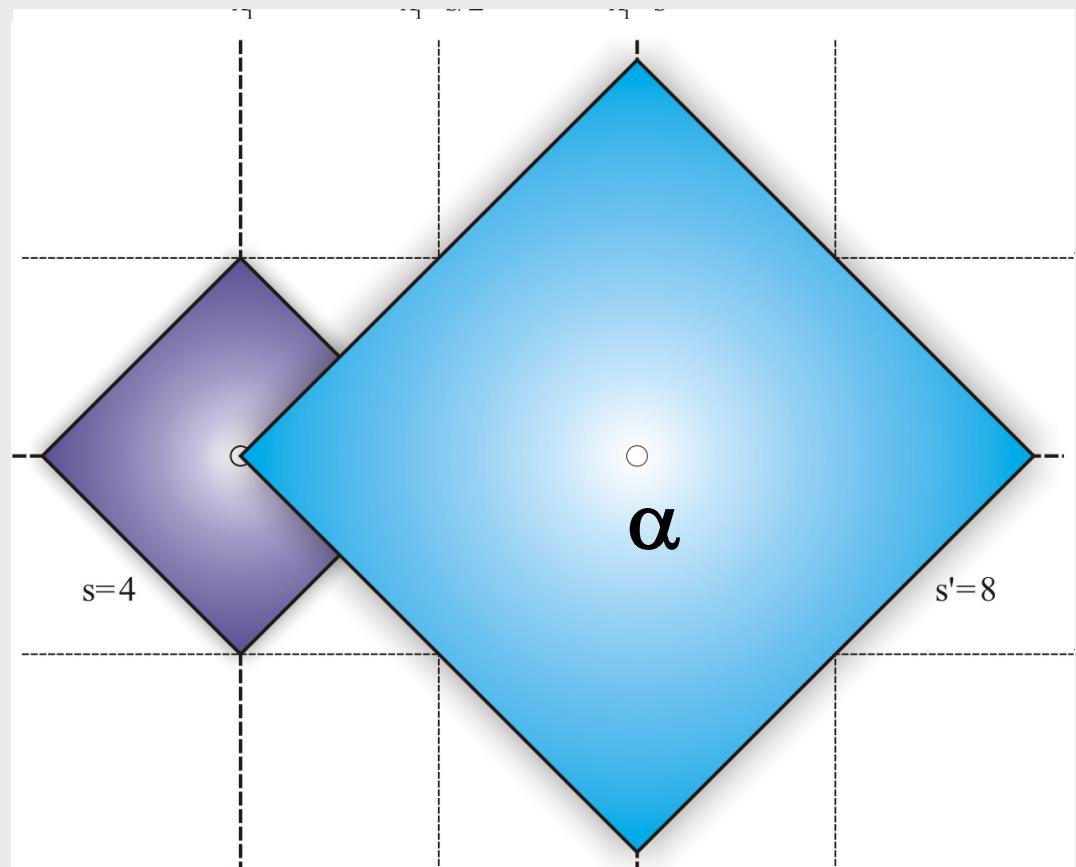


Connectivity: problem



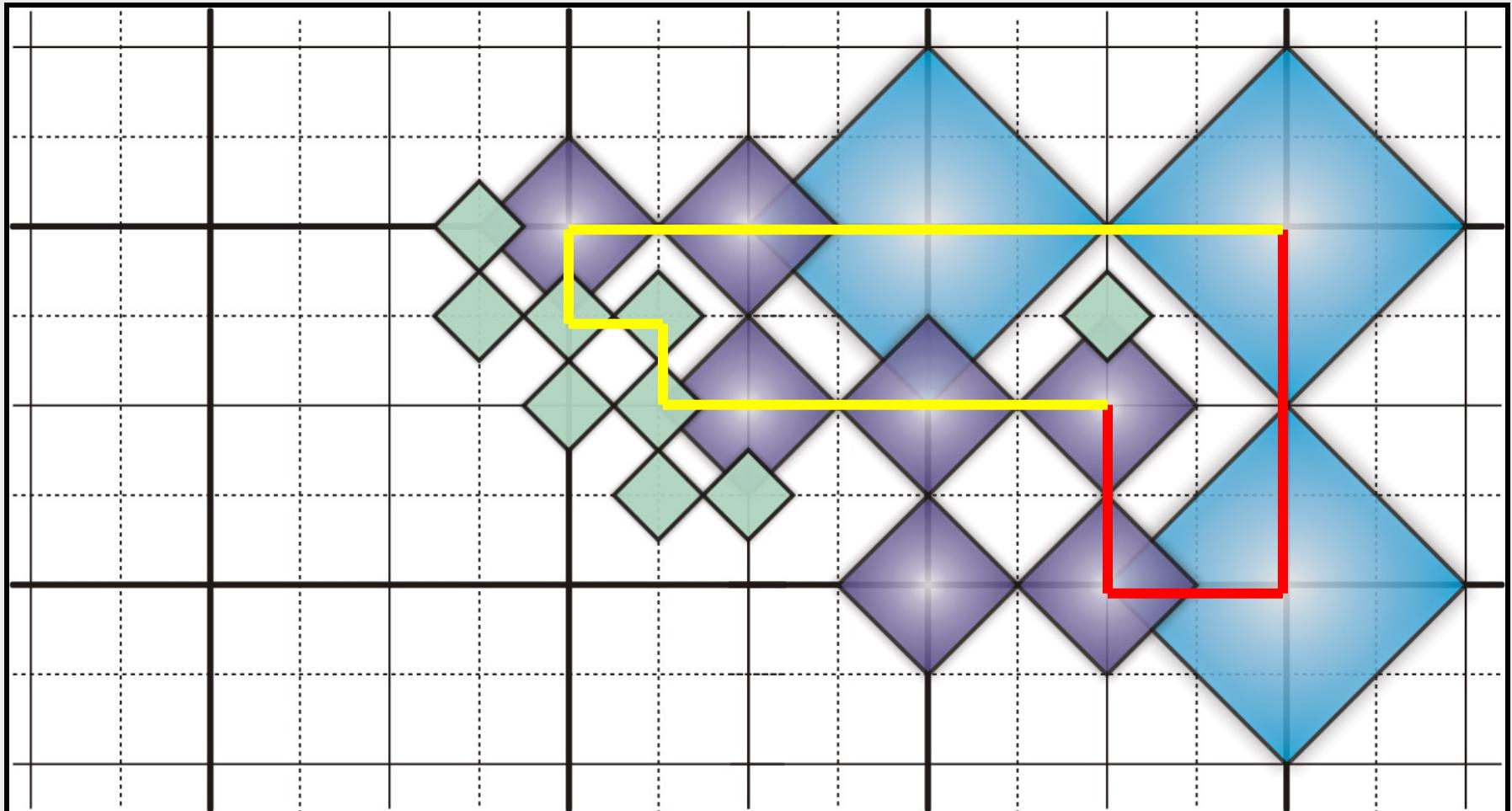
Connectivity: solution

- ▶ There are two different positions to place nodes:
 - ▶ On α -position all nodes with **same or higher** stages are placed
 - ▶ On β -position all **smaller nodes** are placed

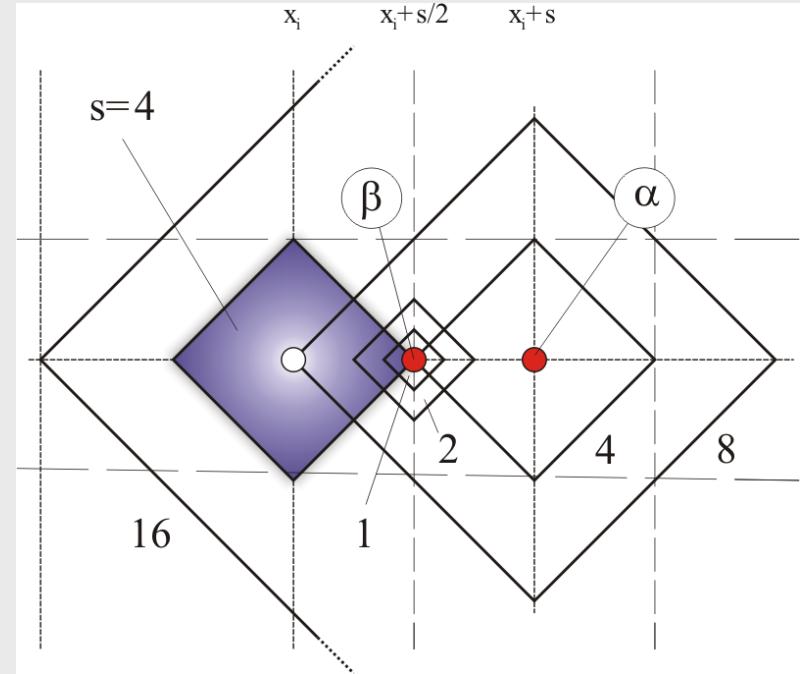
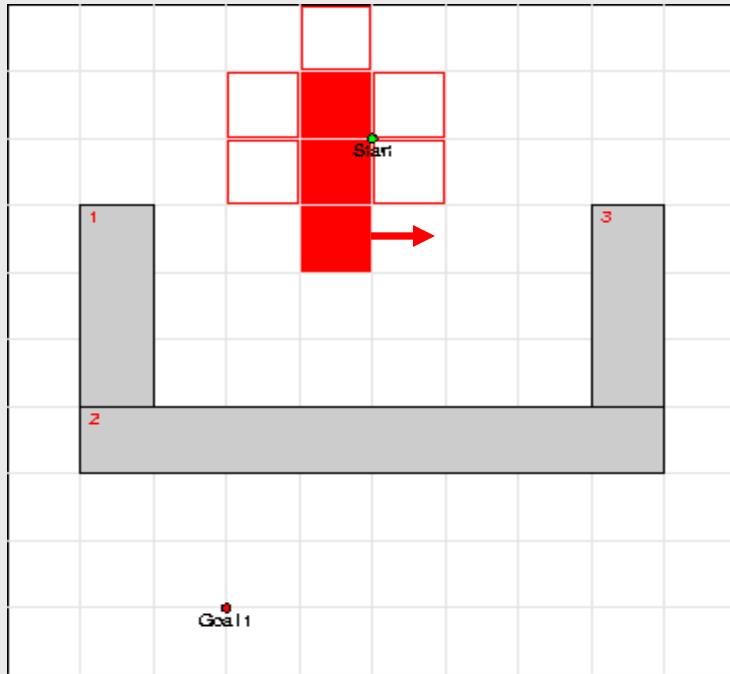


Connectivity: solution

- ▶ Using α - and β -position \rightarrow hierarchical grid
- ▶ There are no parallel search fronts



Hierarchical Expansion



- ▶ Instead of just expanding one node like in „Standard“-A*, several nodes with different stages are placed on α - and β -positions and added to the OPEN-list.
- ▶ Which and how many stages are generated leads to different **expansion strategies**

Adaption of the evaluation function

- ▶ Until now we use the modified evaluation function:
$$f(n) = (1 - w)g(n) + wh(n)$$
- ▶ To take advantage of the hierarchical stages, bigger nodes should be as better considered
- ▶ Therefore evaluation function as to be adapted:

$$f(n) = \frac{(1 - w)g(n) + wh(n)}{s}$$

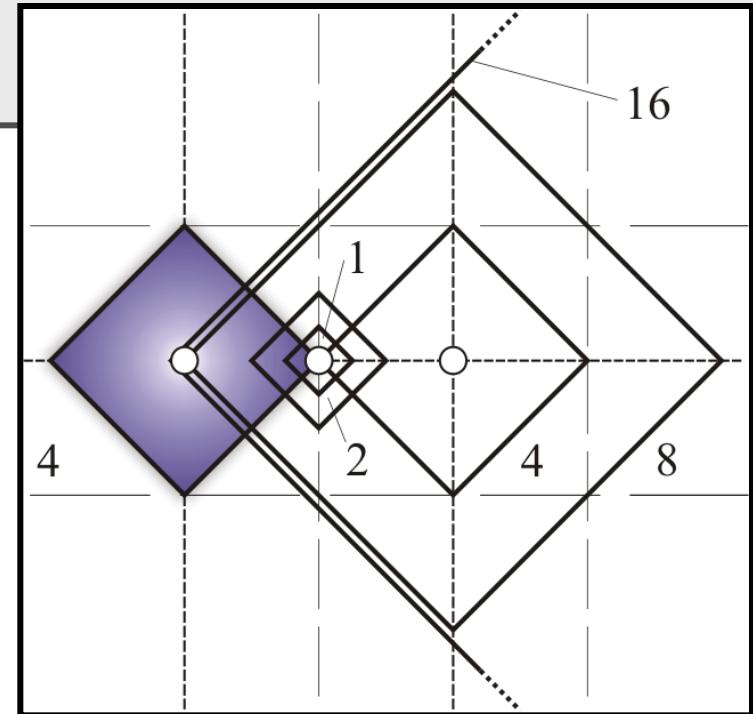
- ▶ If there are several nodes on the same position but with different stages, the biggest one is evaluated as the best one.

Greedy-Expansion

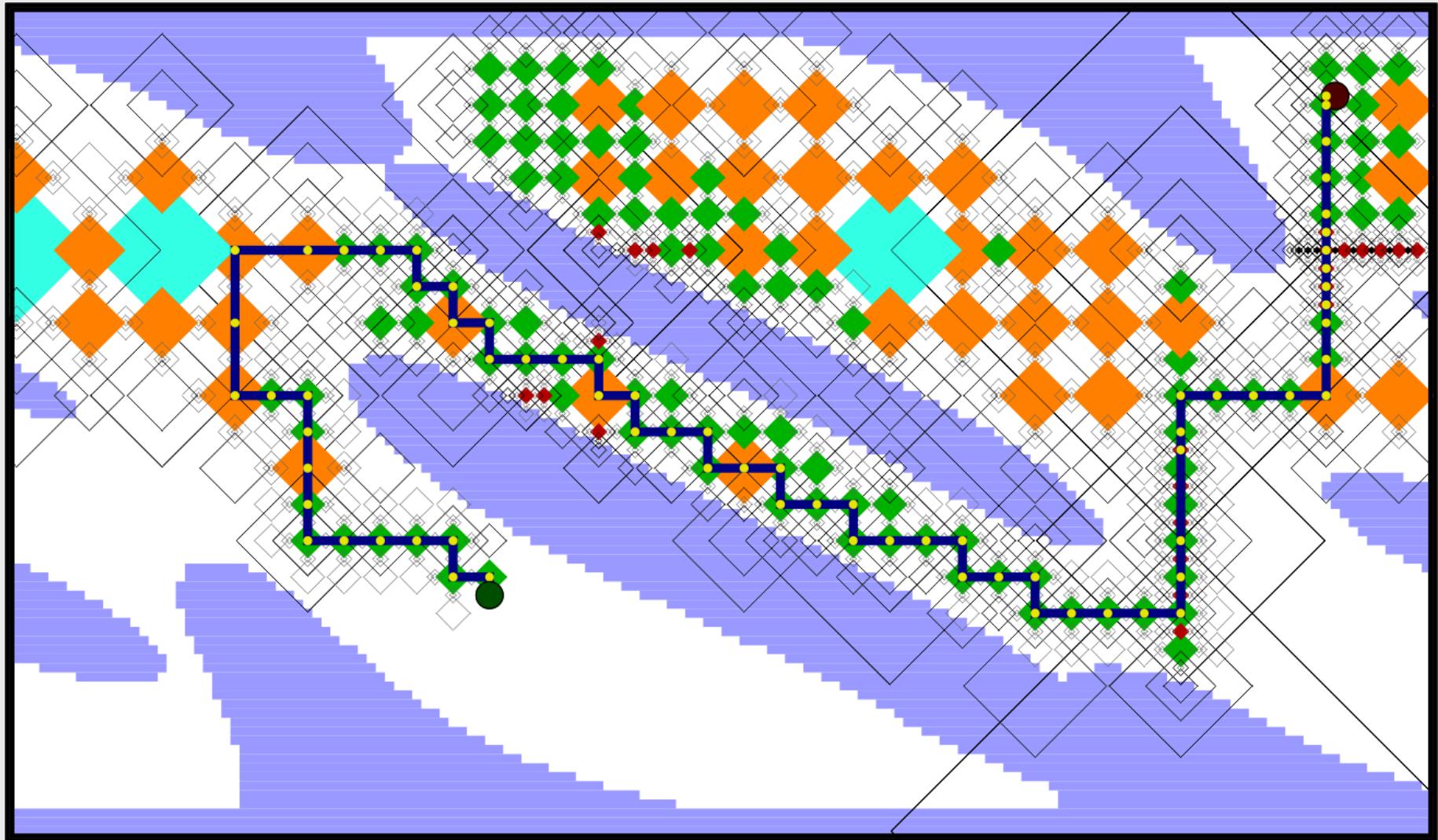
- ▶ First idea: All possible stages are generated

→ A*

```
procedure expandNode (N)
tNode : N;
tNode : results[];
begin
    int : dir;
    int : dim;
    int : stage;
    results = {};
    for (dir ∈ {-1, 1} ) do
        for (dim ∈ {1, . . . ,n}) do
            for (stage ∈ {1, 2, 4, . . . ,smax}) do
                results += generateNode( N; dir; dim; stage );
            end do
        end do
    end do
    return results;
end
```



Jungle-Greedy



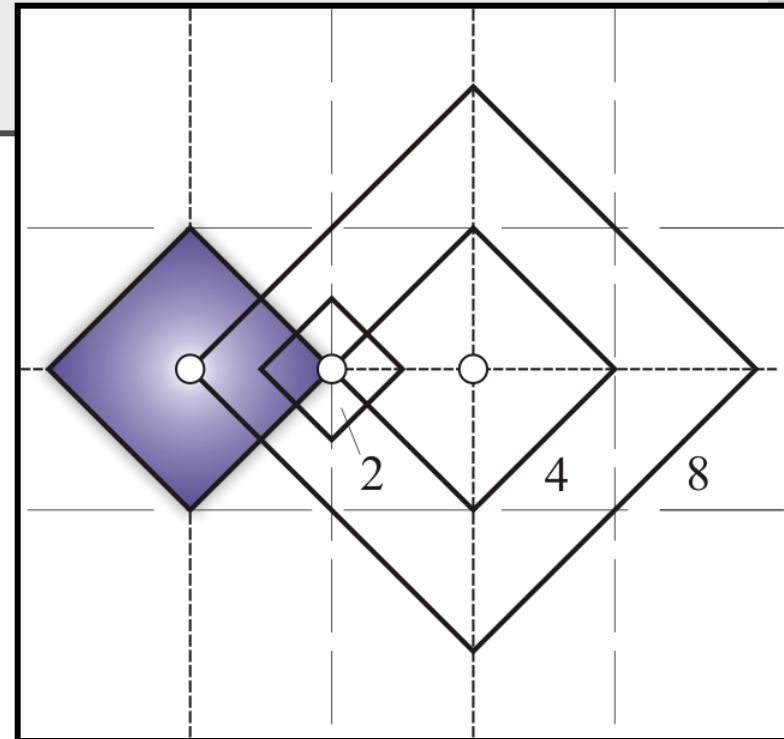
► Problem

- ▶ Using Greedy-Expansion a lot of nodes are generated and the OPEN-List grows rapidly
- ▶ All generated nodes have to be evaluated
- ▶ The only advantage is automatic adaption of the discretisation

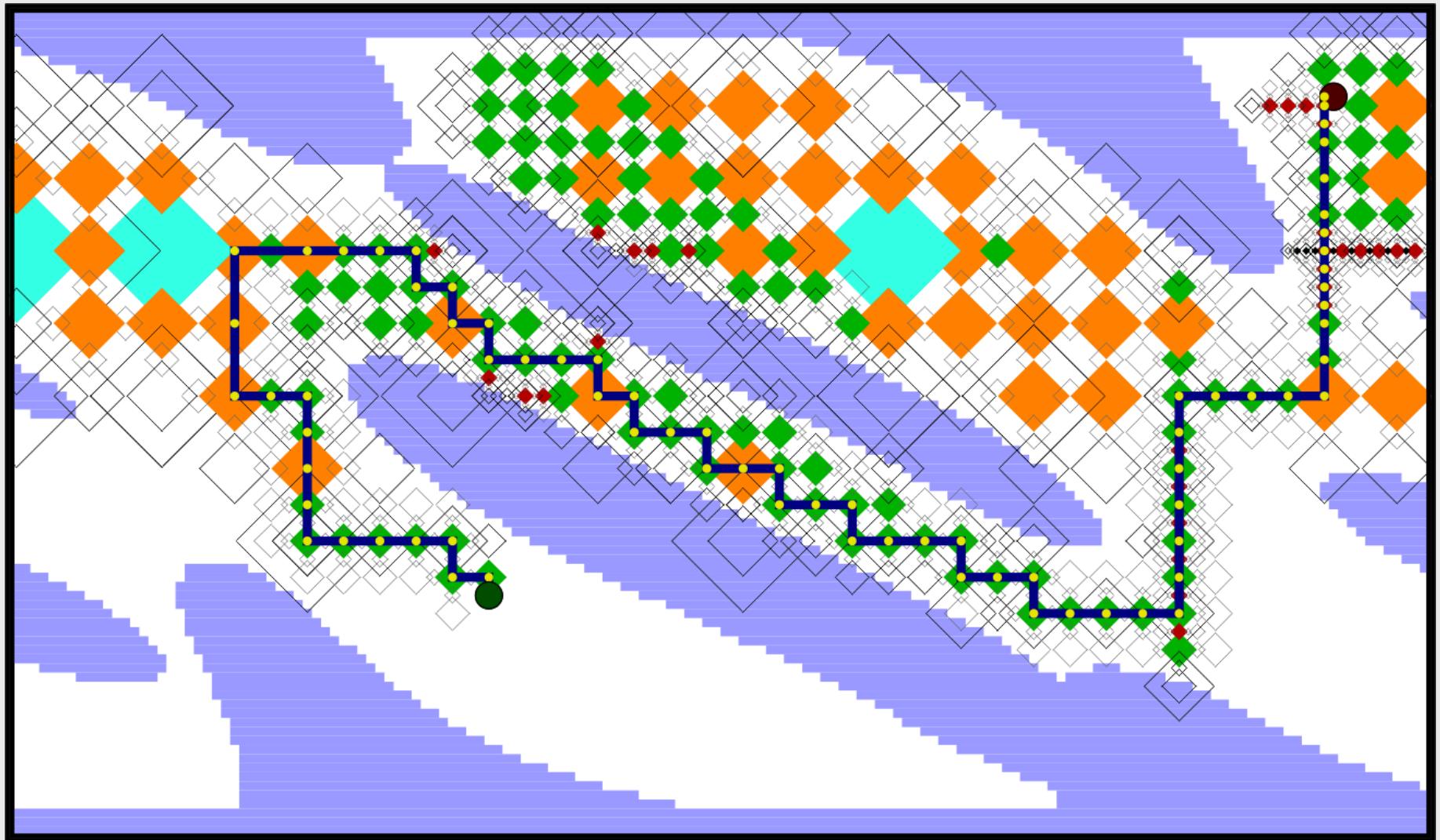
Greedy-Limited

- Successor are only generated in range [-1,1]

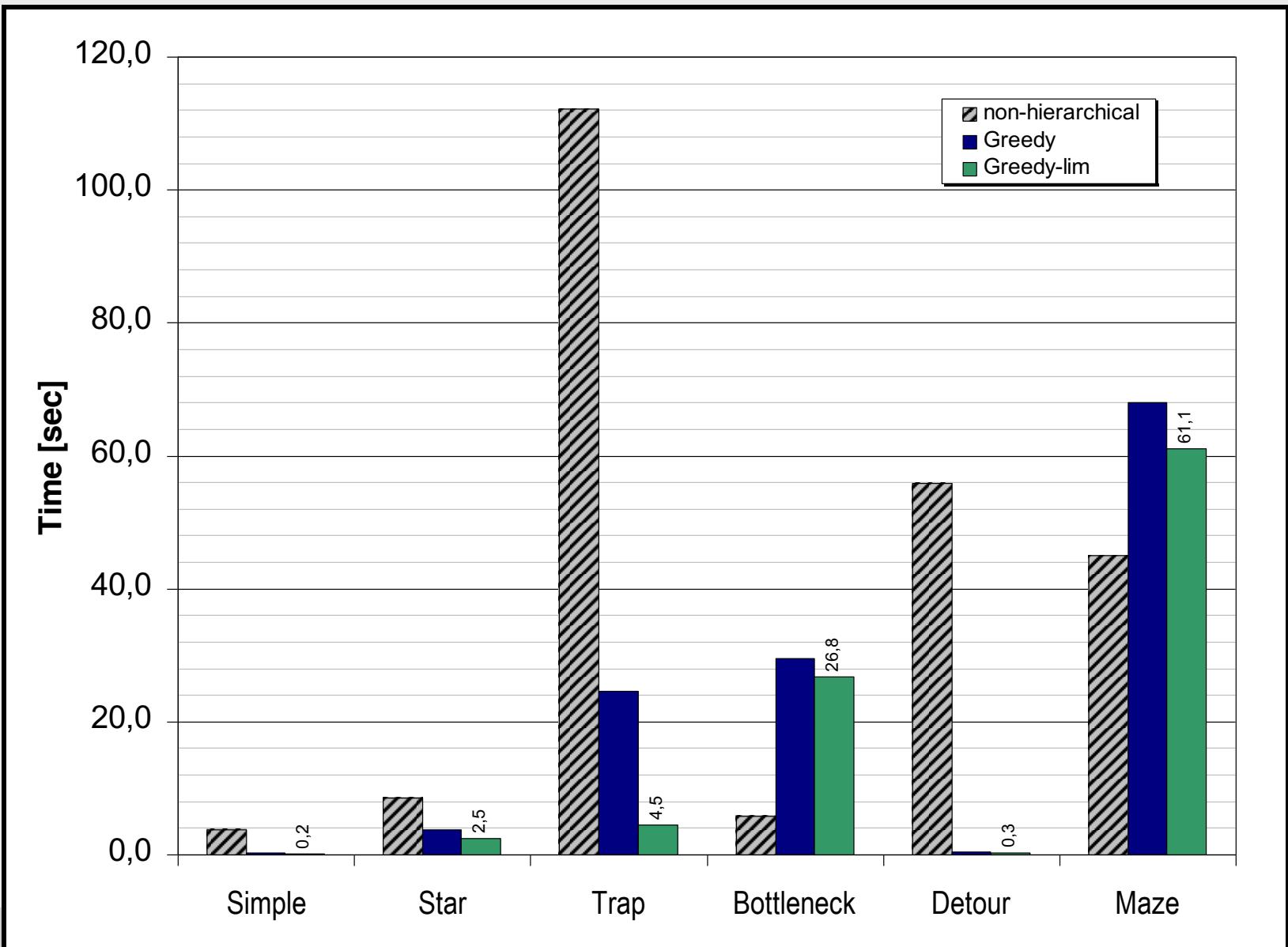
```
procedure expandNode (N)
tNode : N;
tNode : results[];
begin
    int : dir;
    int : dim;
    int : stage;
    results = {};
    for (dir ∈ {-1, 1} ) do
        for (dim ∈ {1, . . . ,n}) do
            for (stage ∈ {N.s/2, N.s, N.s*2}) do
                results += generateNode( N; dir; dim; stage );
            end do
        end do
    end do
    return results;
end
```



Jungle-Greedy-Limited



Comparison NON / Greedy/ Greedy-limited



Results until now

Positive:

- ▶ Free space is better exploited and search often faster

Negative:

- ▶ Many unnecessary nodes generated
- ▶ In narrow passages hierarchical planning is significantly worse

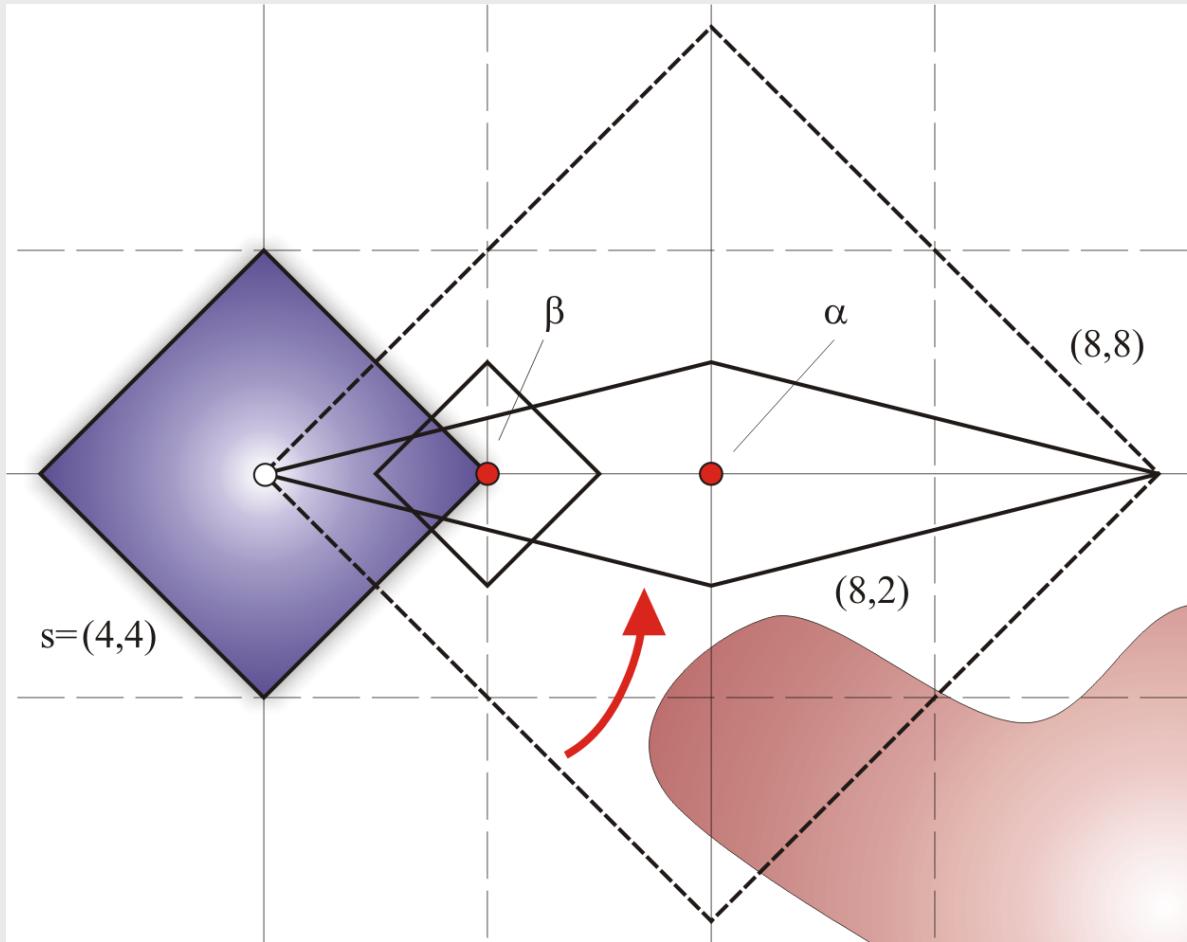
→ Reduce number of nodes

Other approach

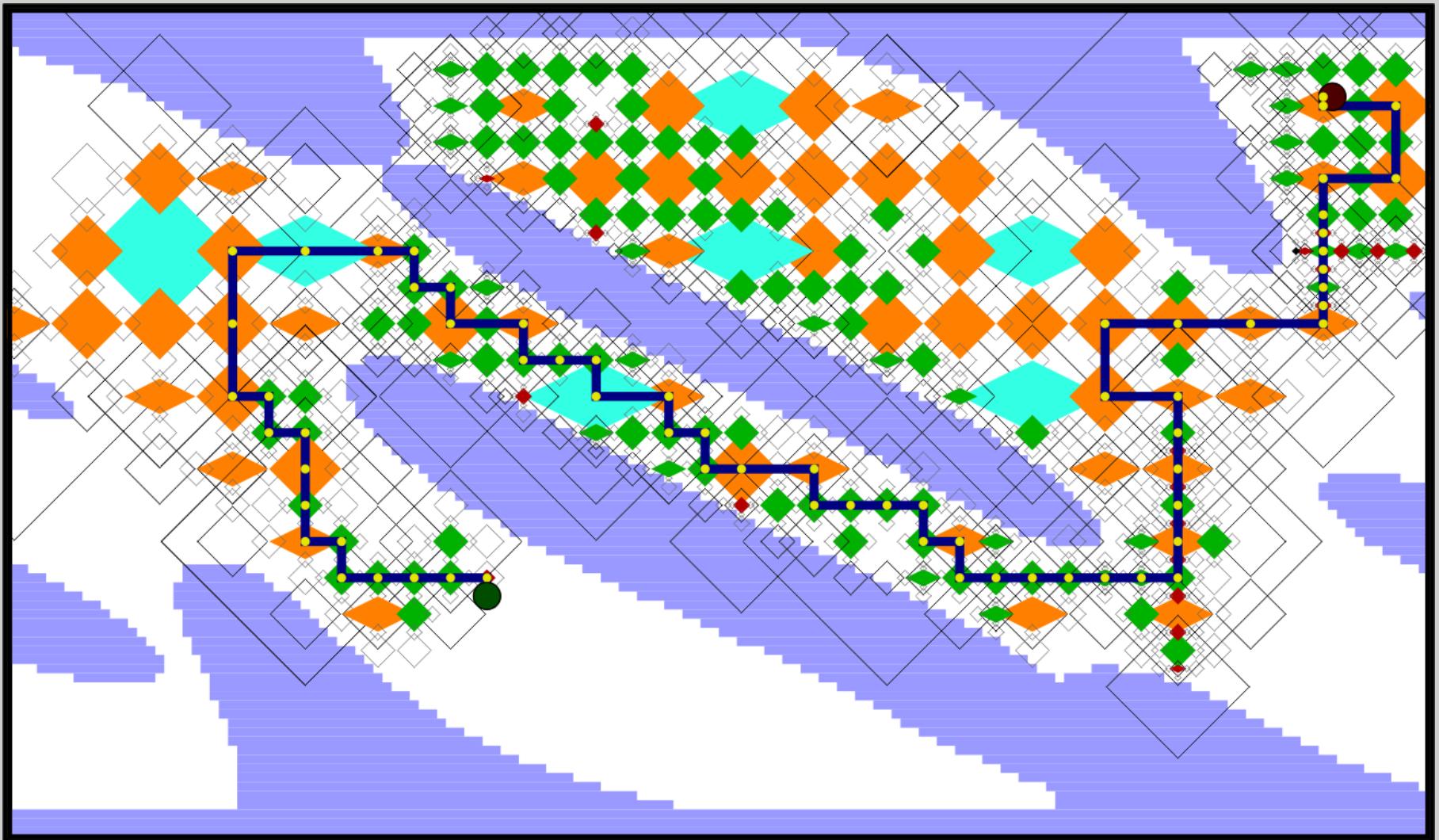
- ▶ At least we need nodes on two position: α - and β
- ▶ With Greedy-variant we place nodes with fixed sizes at these position
- ▶ New idea:
 - ▶ Just say that there is a node and then fit this node to the available space
 - ▶ Three expansions strategies were tested
 - ▶ Upsize
 - ▶ Downsize
 - ▶ Mixed

Down-Size-Expansion-strategy

- ▶ α - and β -nodes assumed as big as possible → reduce size if necessary

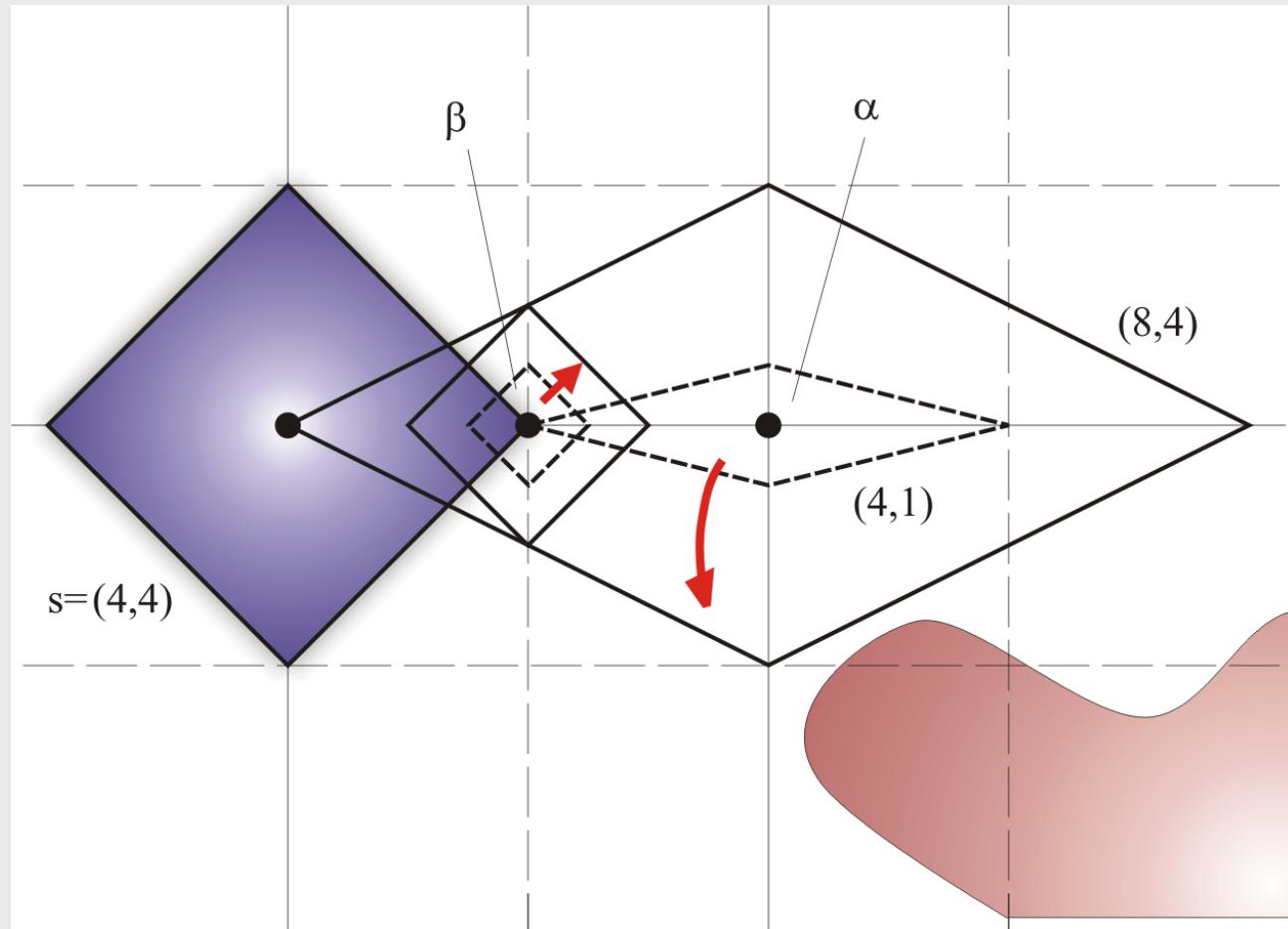


Jungle-Downsize

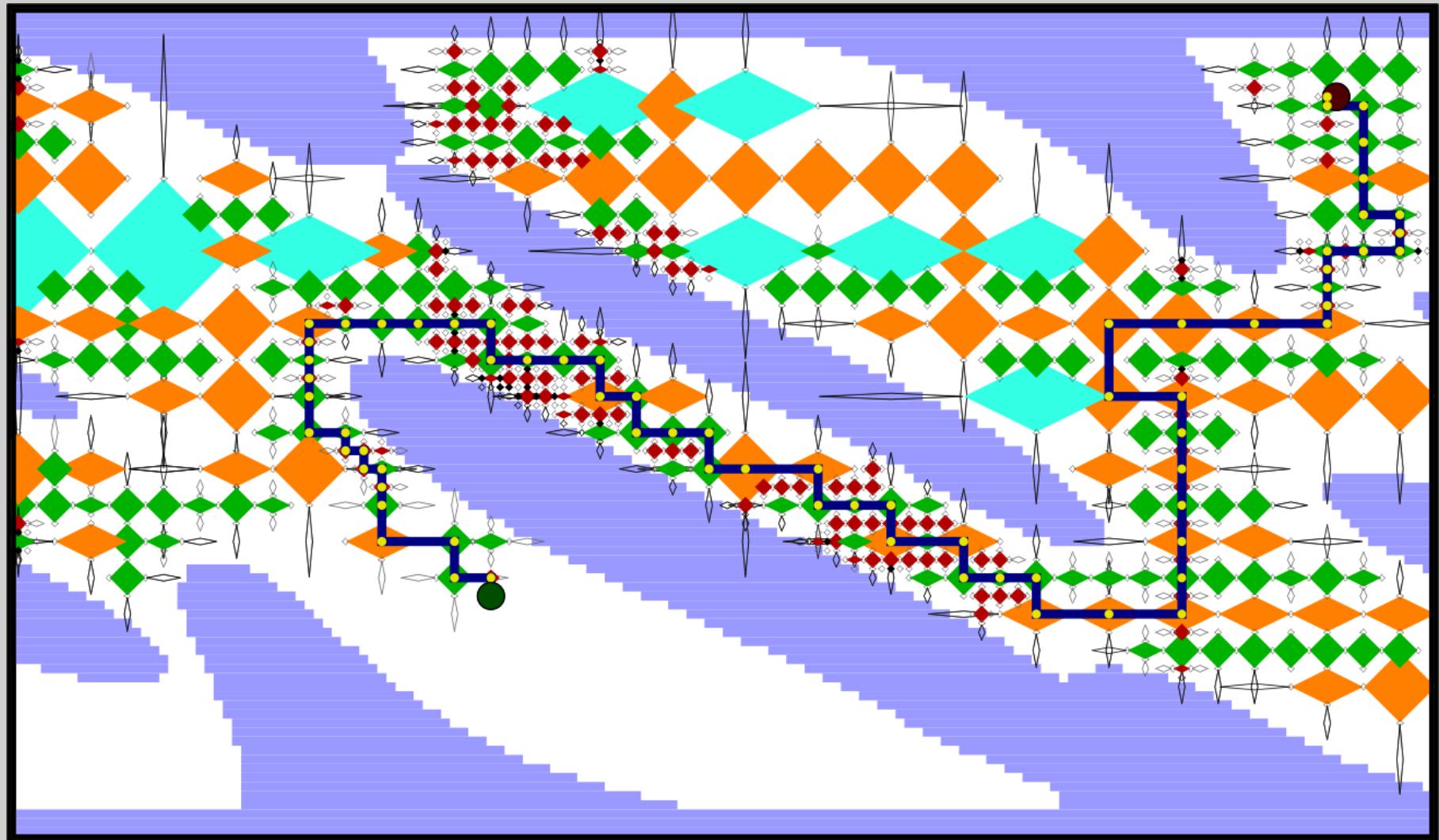


Upsize-Expansion-strategy

- ▶ α - and β -nodes assumed as small as possible \rightarrow inflate size if possible

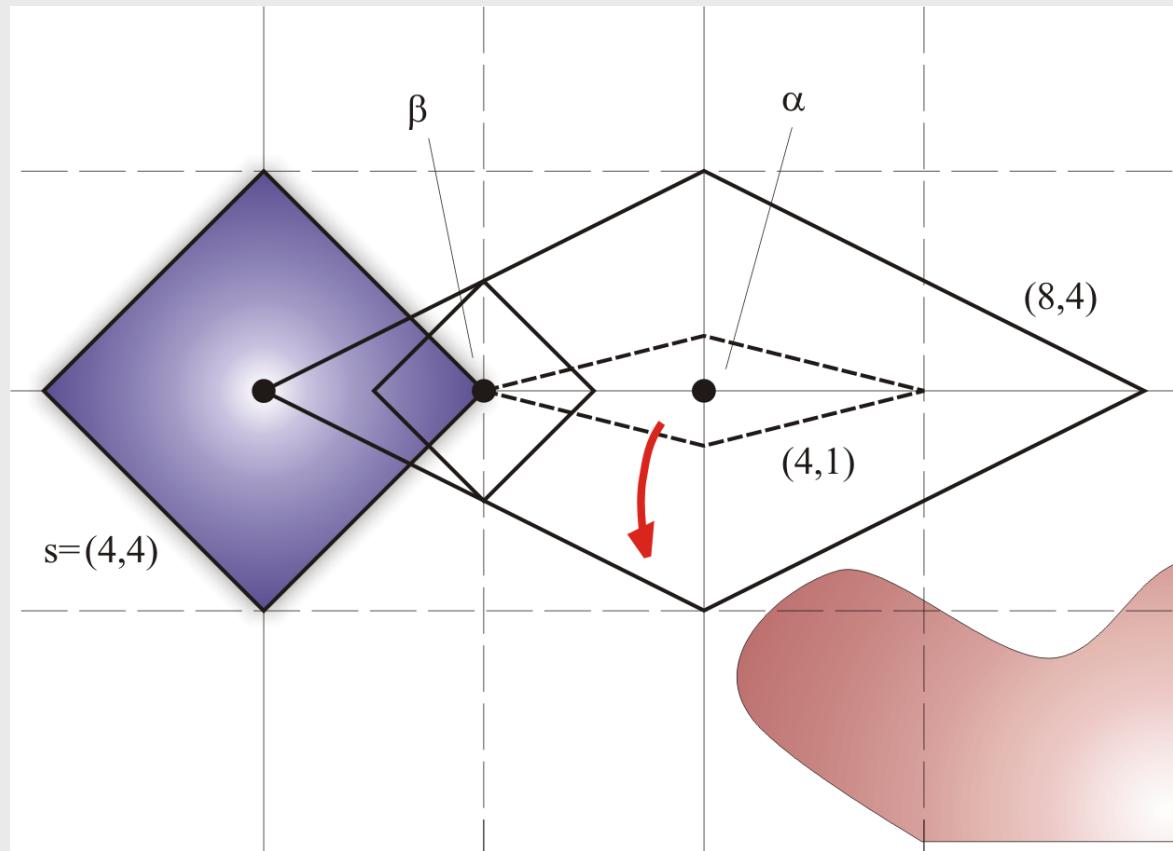


Jungle-Upsize

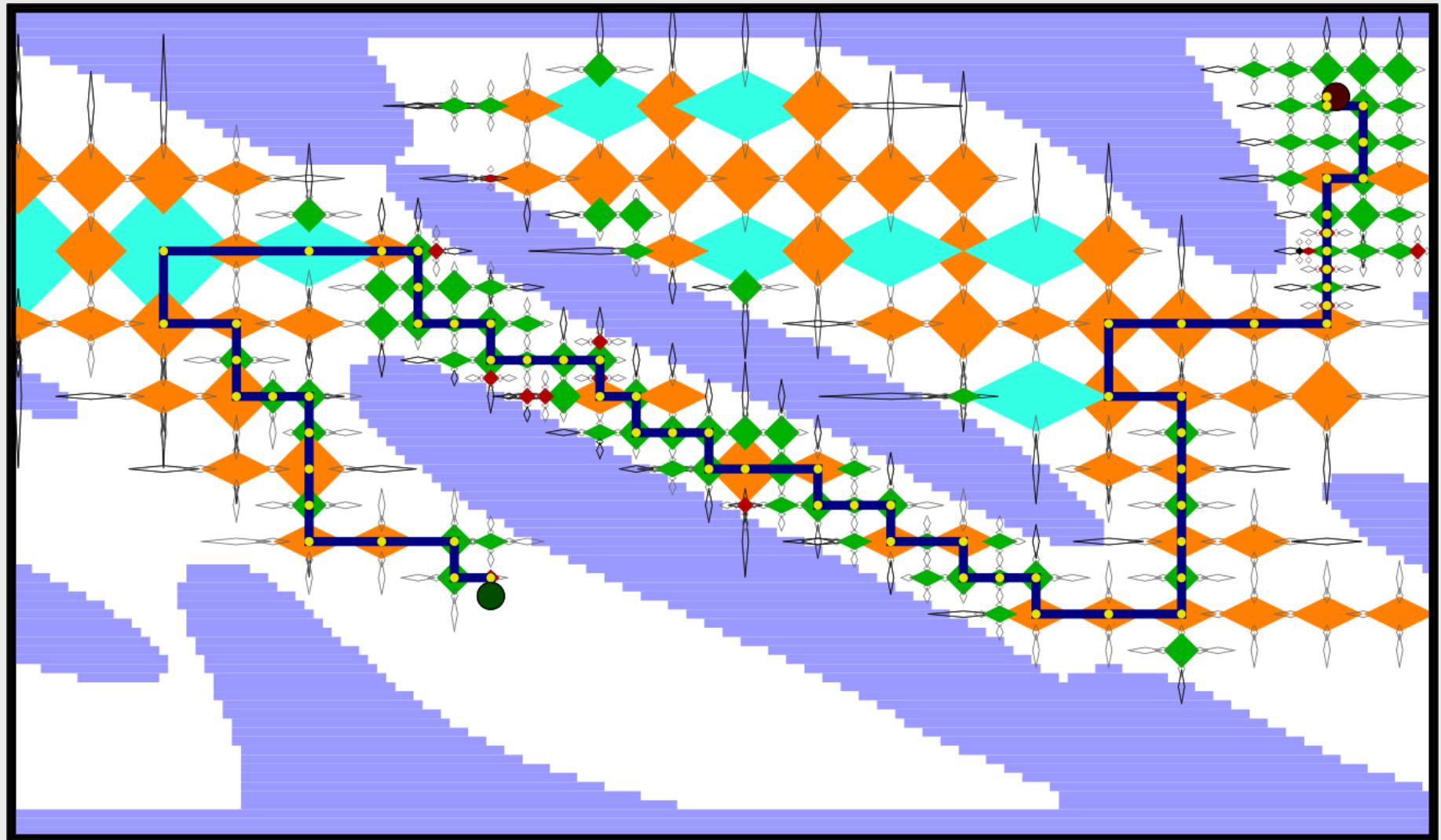


Mixed-Expansion-strategy

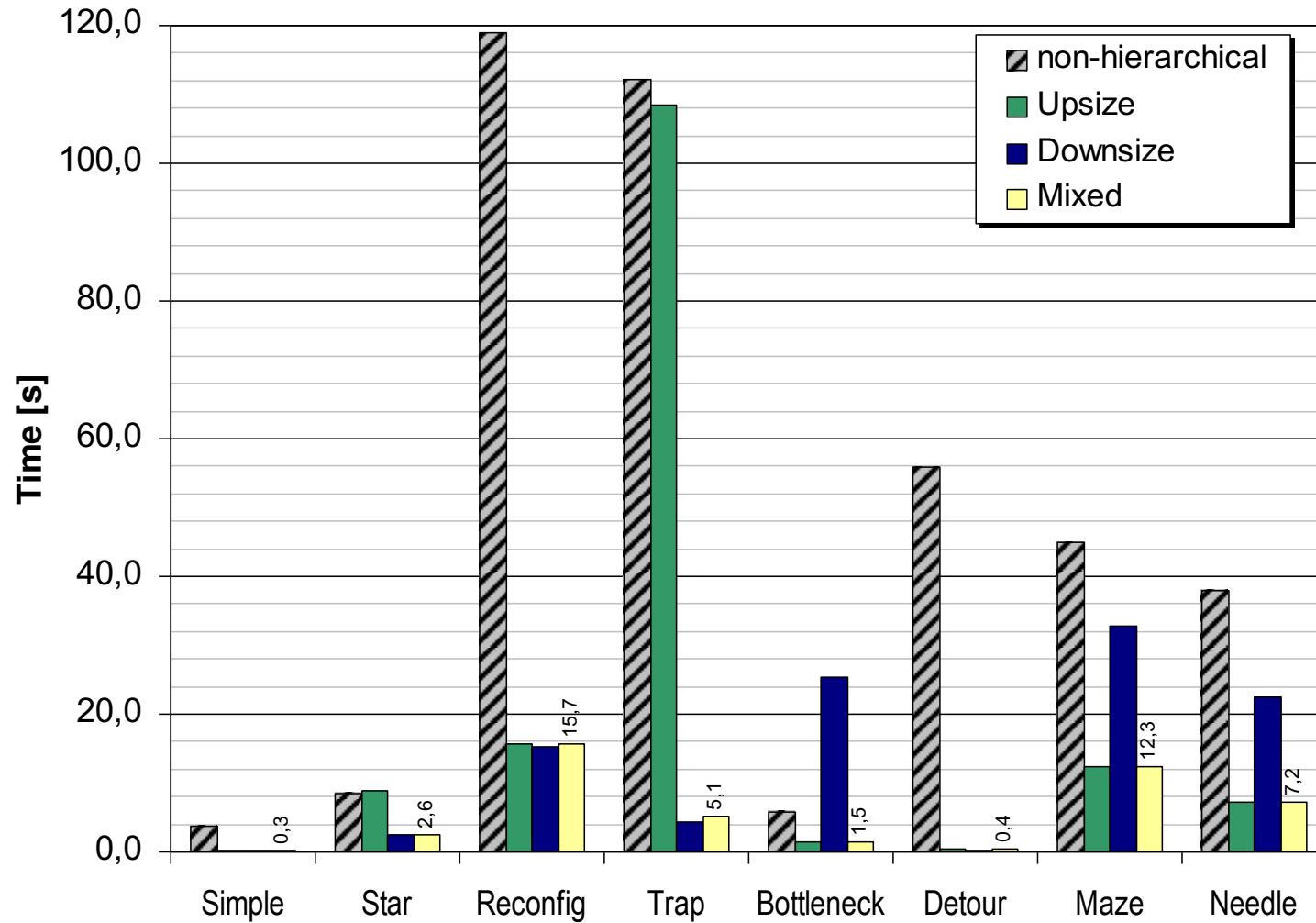
- ▶ α - assumed as big as possible and β -nodes assumed as small as possible → adapt size if necessary resp. possible



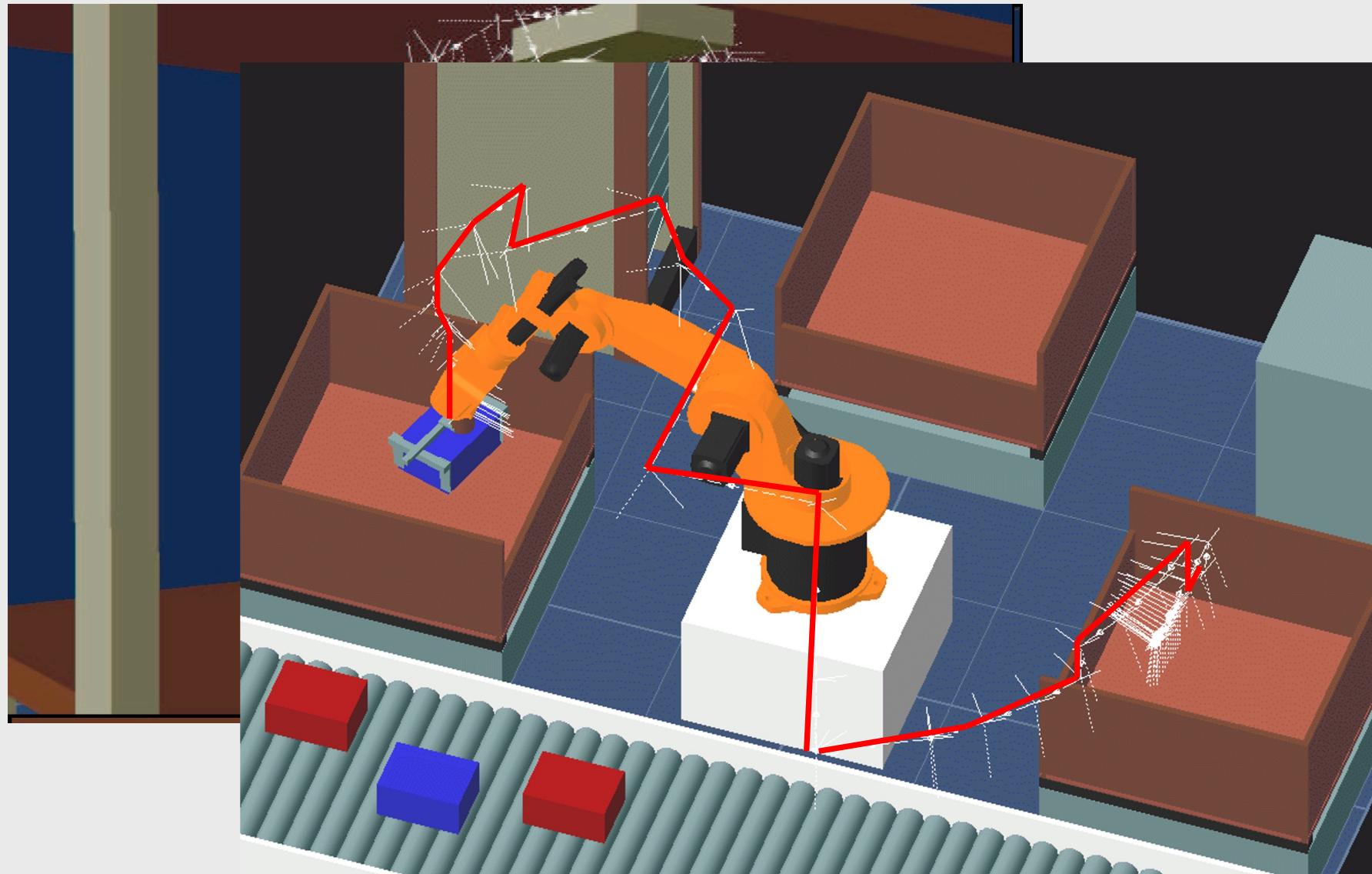
Jungle-Mixed



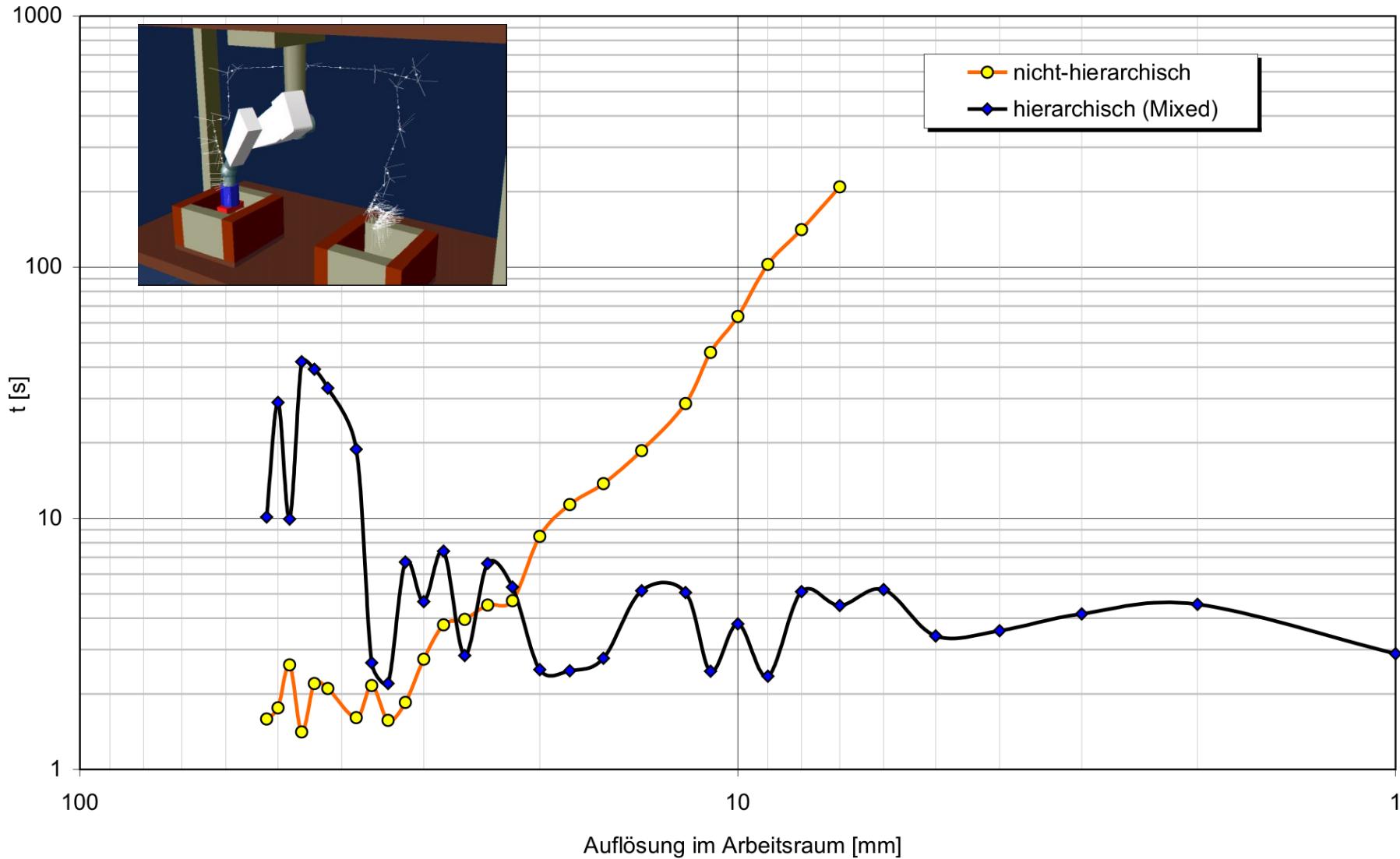
Comparison of the strategies



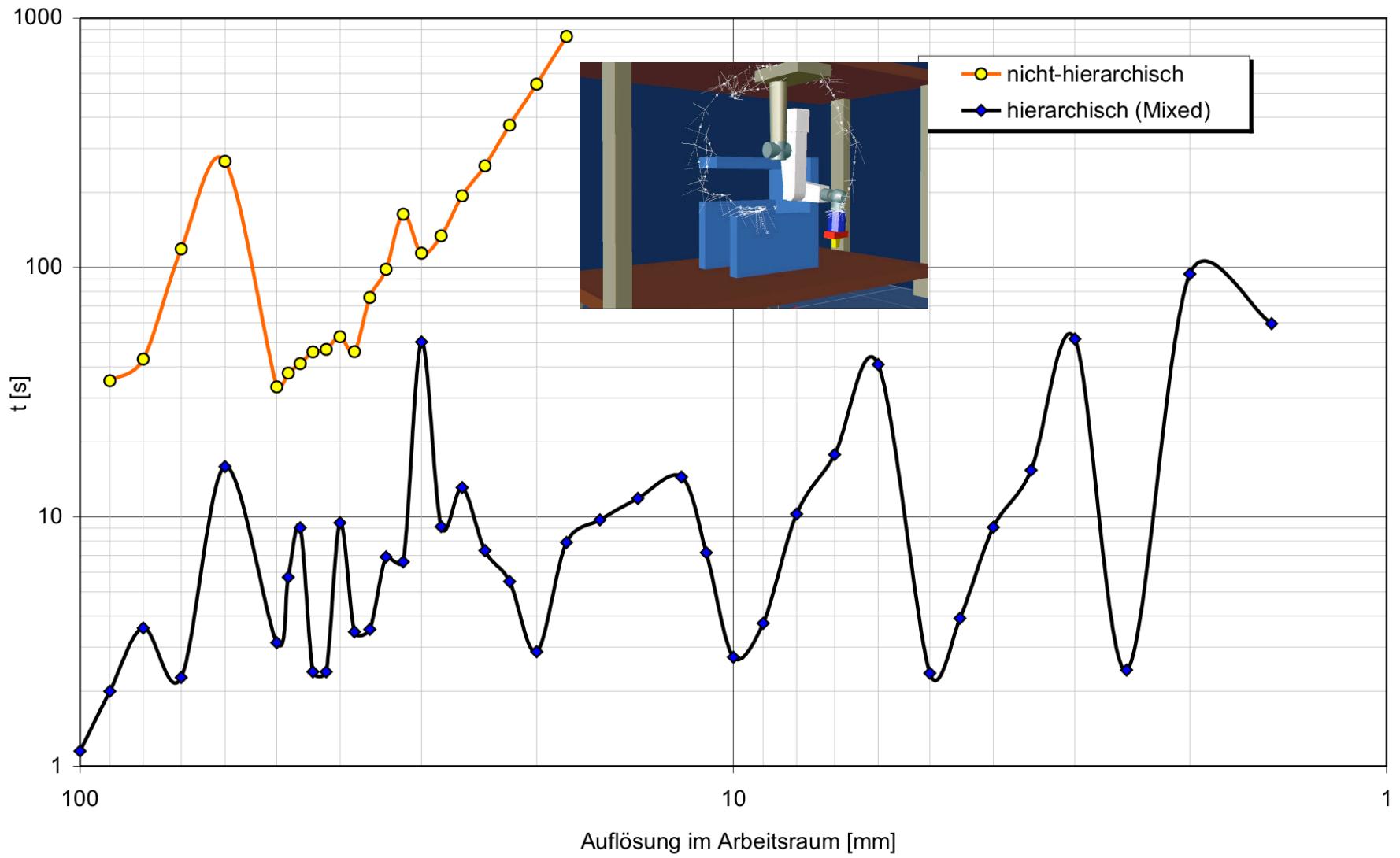
Results



Time: Benchmark Star



Time: Benchmark Trap



Conclusion

- ▶ Using a **hierarchical approach**, planning time can be **drastically reduced**
- ▶ More over: **resolution** can be **raised** without influencing planning time to much
- ▶ More **complicated tasks** are **solvable**

RDT (Rapidly growing Dense Tree)

RRT (Rapidly growing Random Tree)

RDT: Idea

- ▶ Incrementally construct a search tree
 - ▶ Gradually improve resolution
 - ▶ Without setting parameters for resolution
- Tree **densly** covers C-space
- ▶ A dense sequence of samples is used to guide incremental construction of the tree
 - ▶ Dense sequence = randomly generated → RDT → RRT
 - ▶ RRT is part of the RDT family

RDT: strategy

- ▶ $\alpha(i)$: infinite, dense sequence of samples in C-space
- ▶ Possible ways to get a sequence
 - ▶ Uniform
 - ▶ Probabilistic (dense with probability one)
 - ▶ Random with nonuniform bias allowed, as long it is dense with probability one.
- ▶ RDT is a topological graph $G(V,E)$
- ▶ $S \subset C_{\text{free}}$: all points reached by G

RDT: Algorithm (Simplest one)

- ▶ SIMPLE RDT (q_0)
- ▶ 1 `G.init(q_0) ;`
- ▶ 2 `for i = 1 to k do`
- ▶ 3 `G.add_vertex($\alpha(i)$) ;`
- ▶ 4 `$q_n \leftarrow \text{nearest}(S(G) ; \alpha(i))$;`
- ▶ 5 `G.add_edge($q_n ; \alpha(i)$) ;`

- ▶ $\alpha(i)$: infinite, dense sequence
- ▶ Above algorithm is the basic one when no obstacles are present.

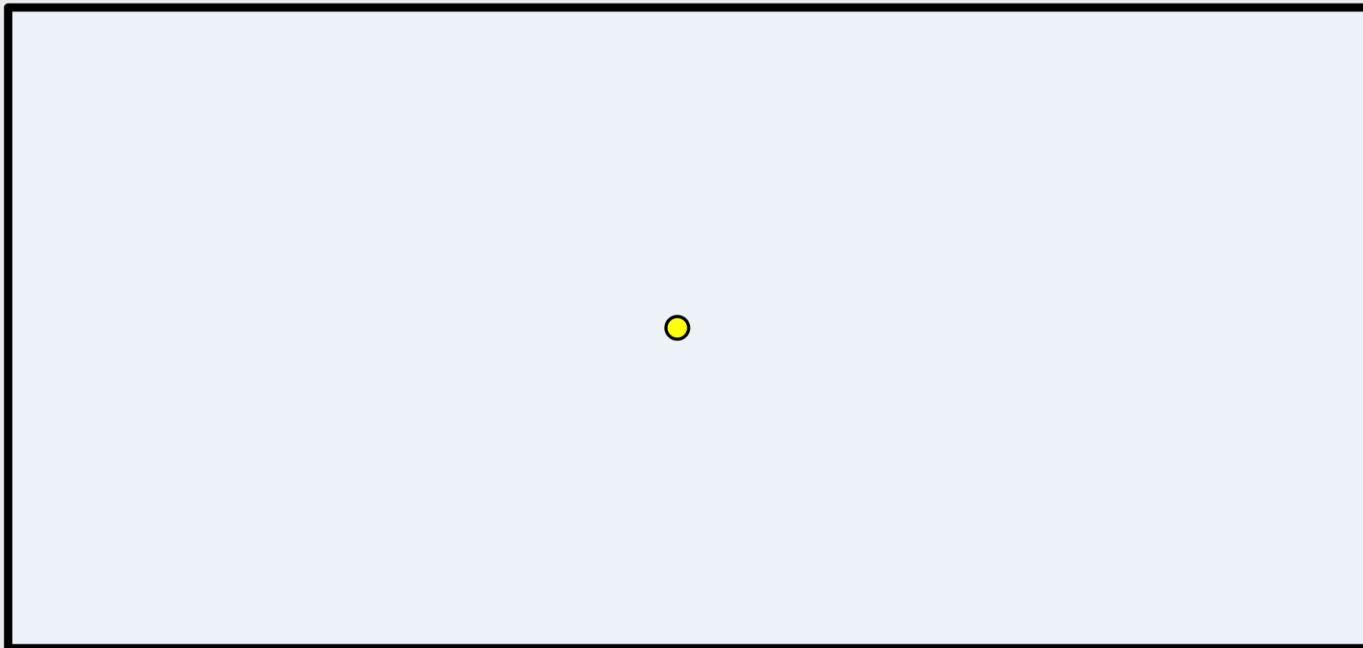
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶   3         G.add_vertex( $\alpha(i)$ );
▶   4          $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶   5         G.add_edge( $q_n; \alpha(i)$ );
```



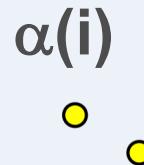
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶   3         G.add_vertex( $\alpha(i)$ );
▶   4          $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶   5         G.add_edge( $q_n; \alpha(i)$ );
```



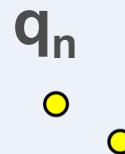
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶ 3     G.add_vertex( $\alpha(i)$ );
▶ 4      $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶ 5     G.add_edge( $q_n; \alpha(i)$ );
```



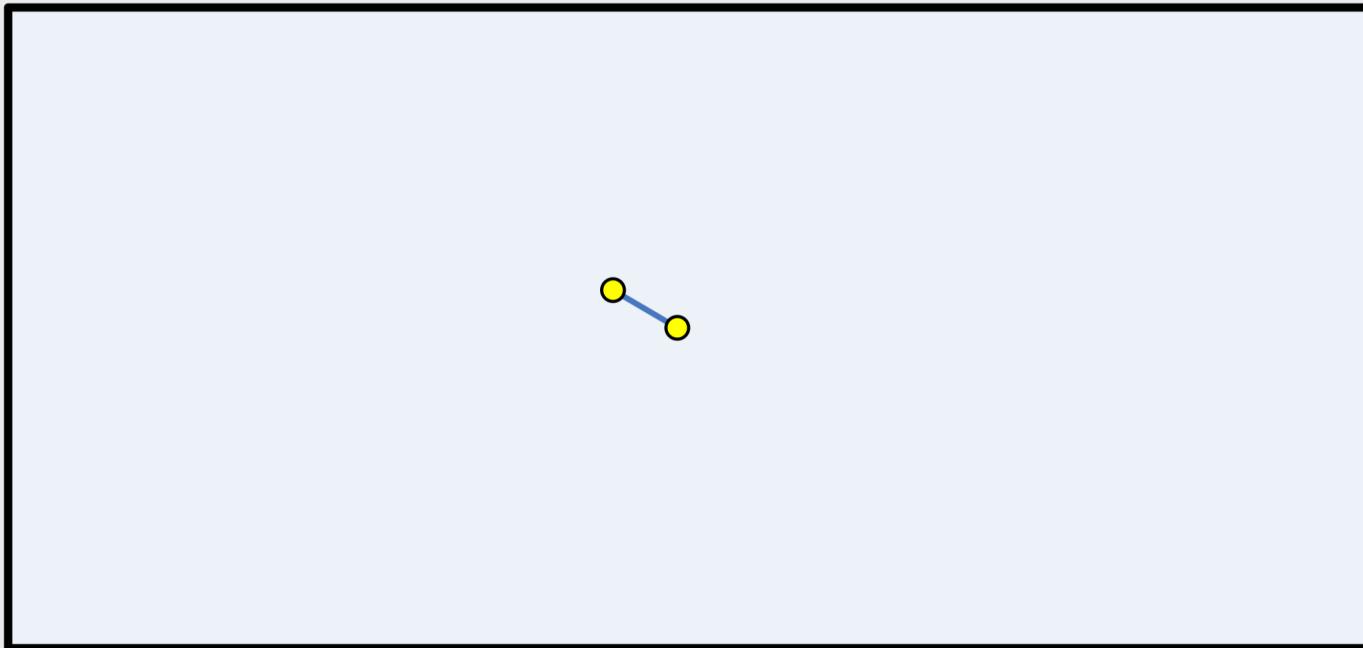
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶ 3     G.add_vertex( $\alpha(i)$ );
▶ 4      $q_n \leftarrow \text{nearest}(S(G); \alpha_{\dots})$ 
▶ 5     G.add_edge( $q_n; \alpha(i)$ );
```



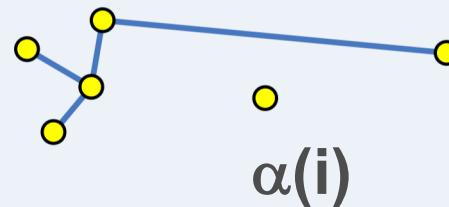
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶   3         G.add_vertex( $\alpha(i)$ );
▶   4          $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶   5         G.add_edge( $q_n; \alpha(i)$ )
```



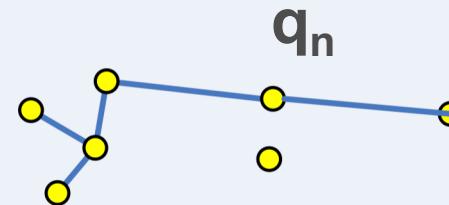
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶ 3     G.add_vertex( $\alpha(i)$ );
▶ 4      $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶ 5     G.add_edge( $q_n; \alpha(i)$ );
```



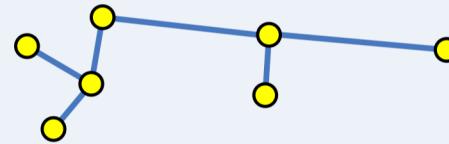
Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶ 3     G.add_vertex( $\alpha(i)$ );
▶ 4      $q_n \leftarrow \text{nearest}(S(G); \alpha$   t restricted to points)
▶ 5     G.add_edge( $q_n; \alpha(i)$ );
```



Example: Simplest one

```
▶ SIMPLE RDT( $q_0$ )
▶ 1 G.init( $q_0$ );
▶ 2 for i = 1 to k do
▶ 3     G.add_vertex( $\alpha(i)$ );
▶ 4      $q_n \leftarrow \text{nearest}(S(G); \alpha(i))$ ;
▶ 5     G.add_edge( $q_n; \alpha(i)$ );
```



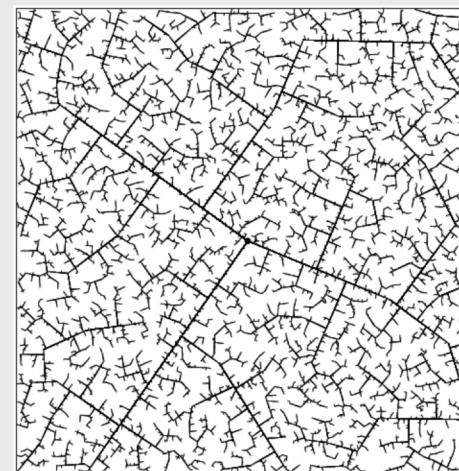
Example: Simplest one / Summary

- ▶ Every iteration the tree grows iteratively by connecting $\alpha(i)$ to S.
- ▶ In every iteration $\alpha(i)$ becomes a vertex \rightarrow tree is dense

$$C = [0; 1]^2, \\ q_0 = (1/2; 1/2).$$



45 iterations



2345 iterations

- ▶ RRT quickly reaches the unexplored parts.
- ▶ RRT is dense \rightarrow it gets arbitrarily to any point in the space.

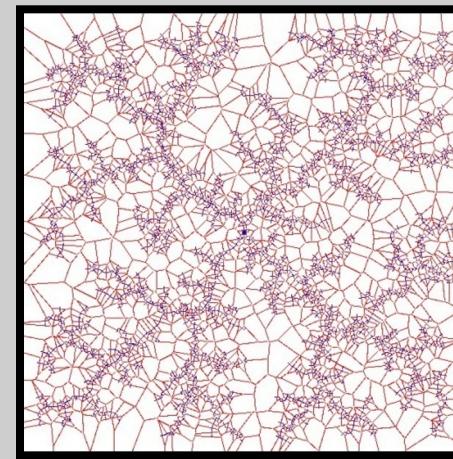
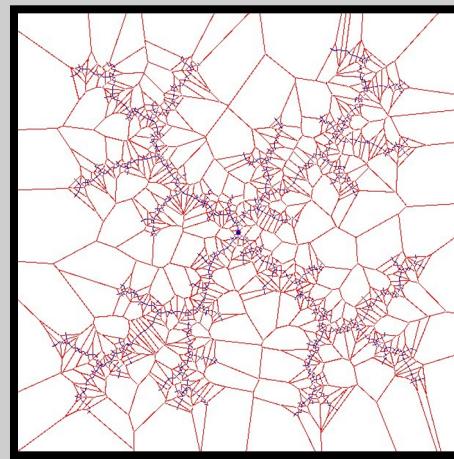
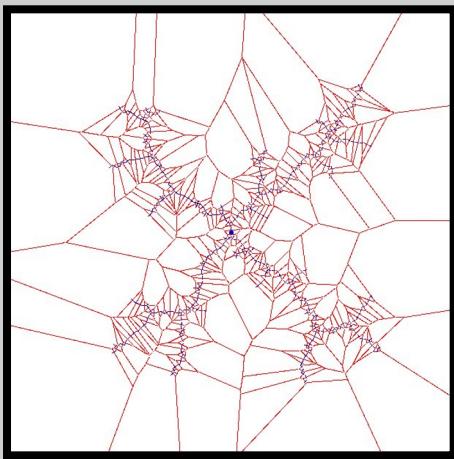
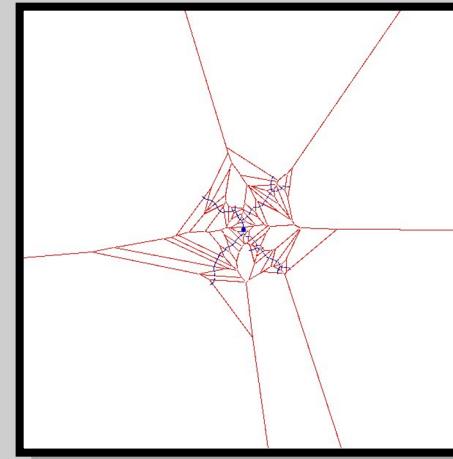
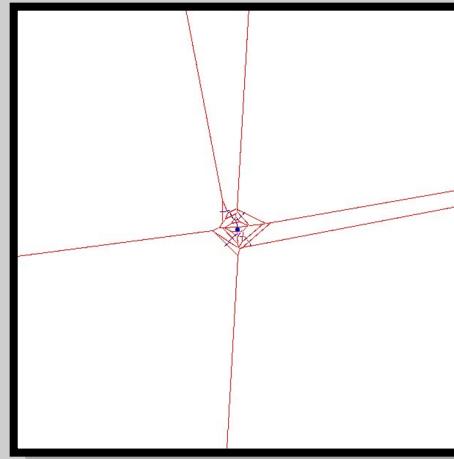
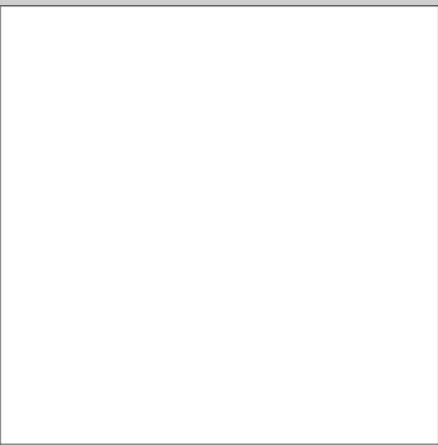
Example: RDT

RDT: properties / features

- ▶ First C-Space is examined into every direction → rapidly reaches corners
- ▶ Gradually, C-Space is filled up with finer branches
- ▶ In the end, C-Space is completely covered

- ▶ Because of tree gradually increasing resolution → perfect suited for sampling based motion planning

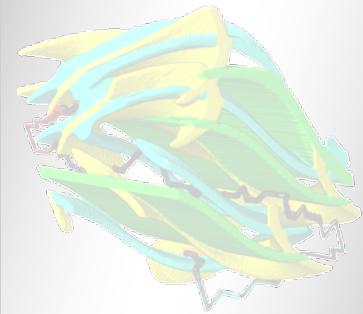
RDT and Voronoi



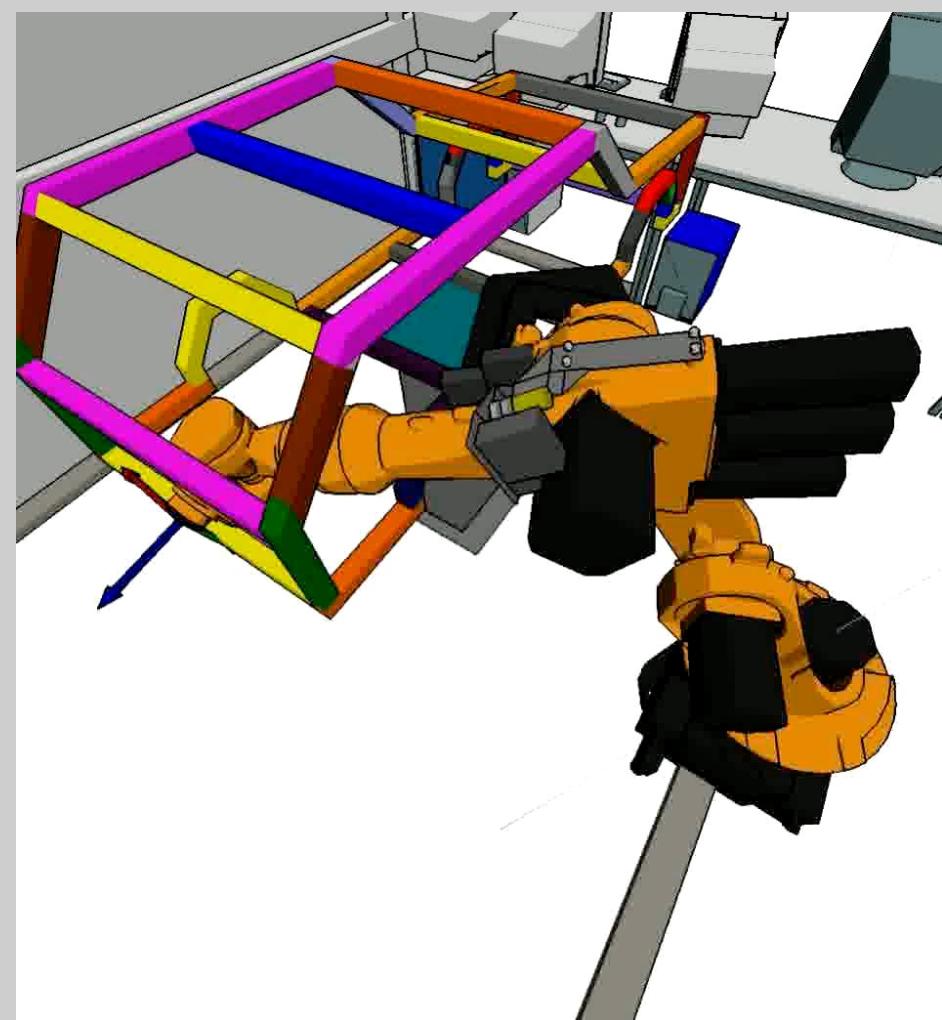
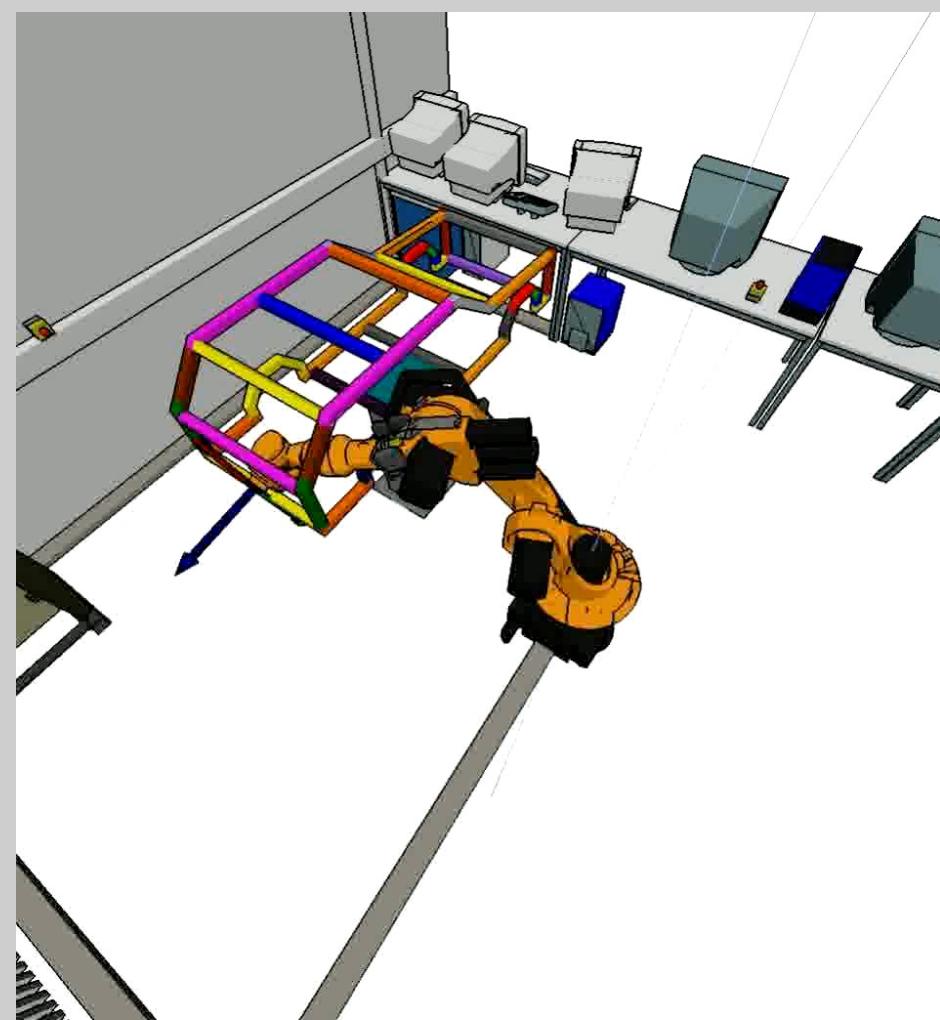
Using RDT for path planning

- ▶ Simple tree search
- ▶ Use proposed algorithm that takes obstacles into account
- ▶ Periodically add goal (e.g. every 100th step) and test whether it is reachable

Smoothing



Why smoothing?

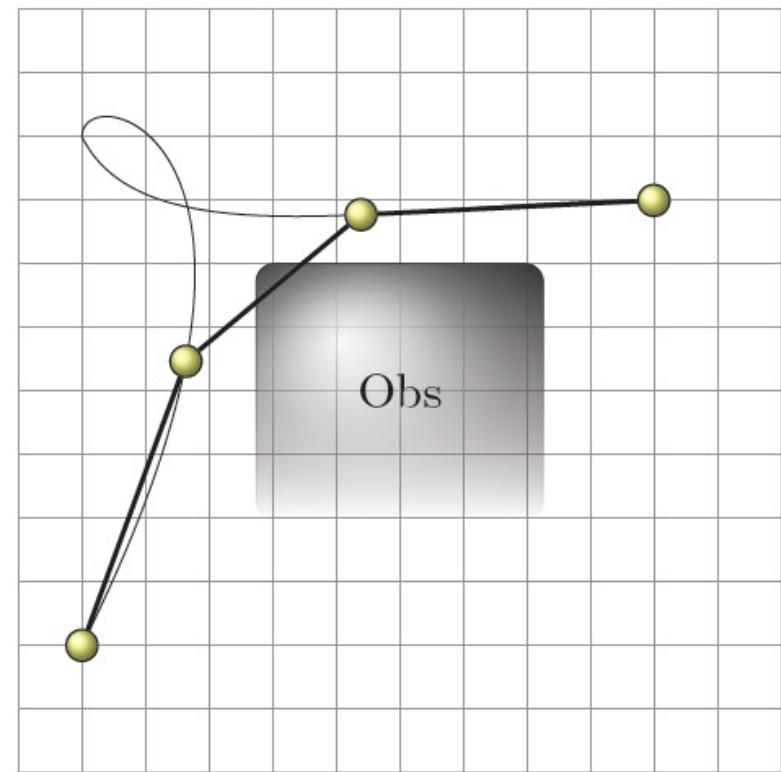
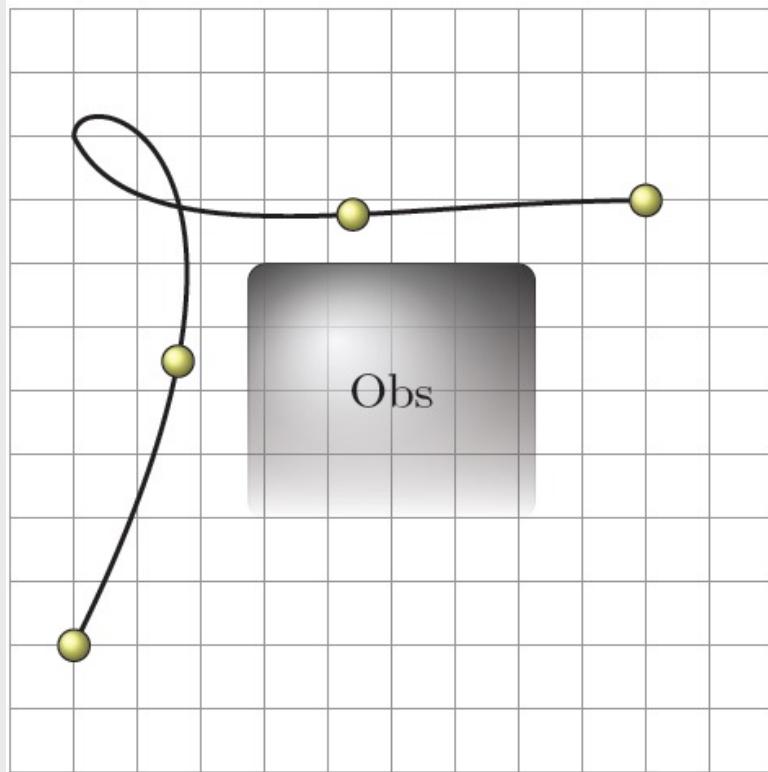


Why smoothing? ☺

- ▶ It's obvious that solutions given by the pathplanning algorithms A*, PRM, ... don't look very nice.....
 - ▶ BB-Method looks quite good, but sometimes unnecessary detours are made.
- Smoothing is always useful but it must be collision free
- ▶ Removing unnecessary points
 - ▶ Reducing path length

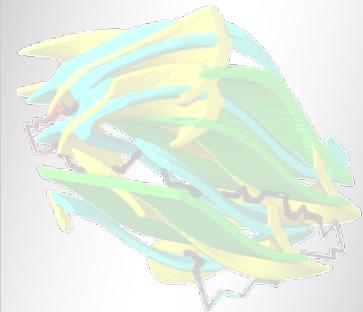
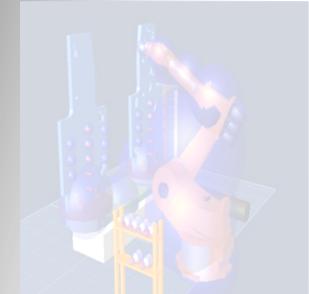
Problem in smoothing

- ▶ Direct connection leads to collision



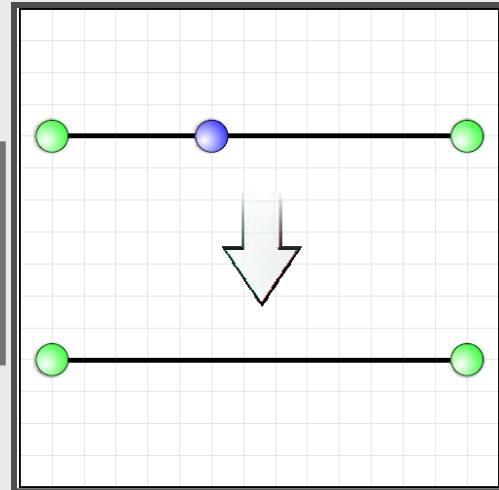
Smoothing Approach

Linear approximation examples

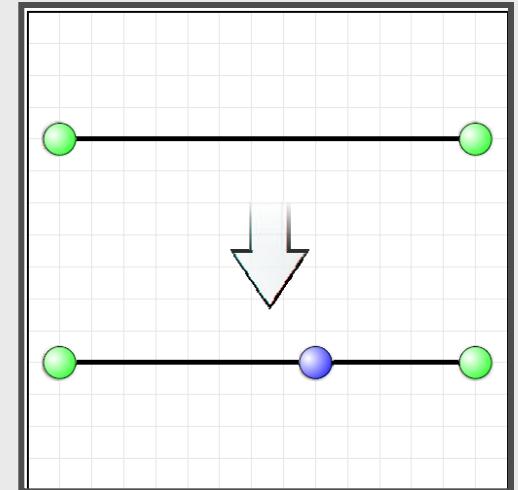


Necessary basic operations

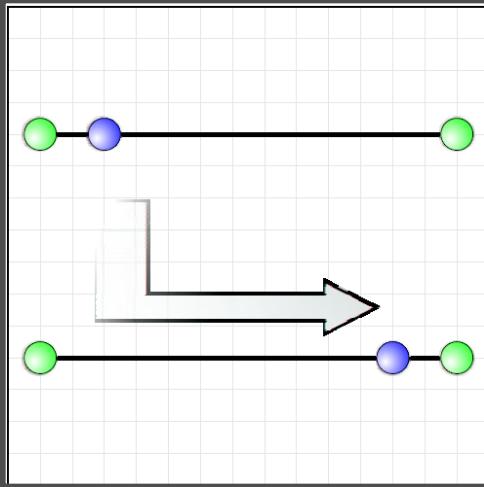
Removing
collinear
points



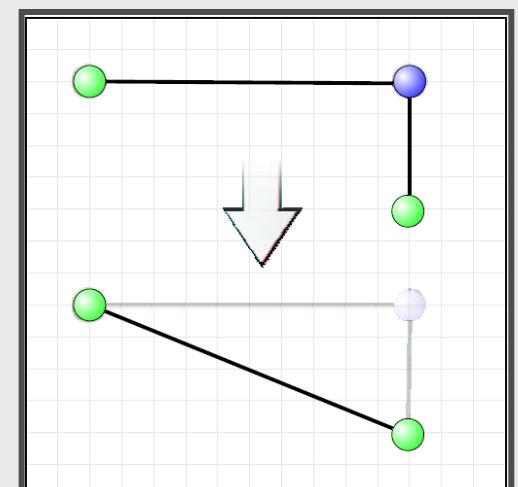
Adding
collinear
points



Shifting
collinear
points

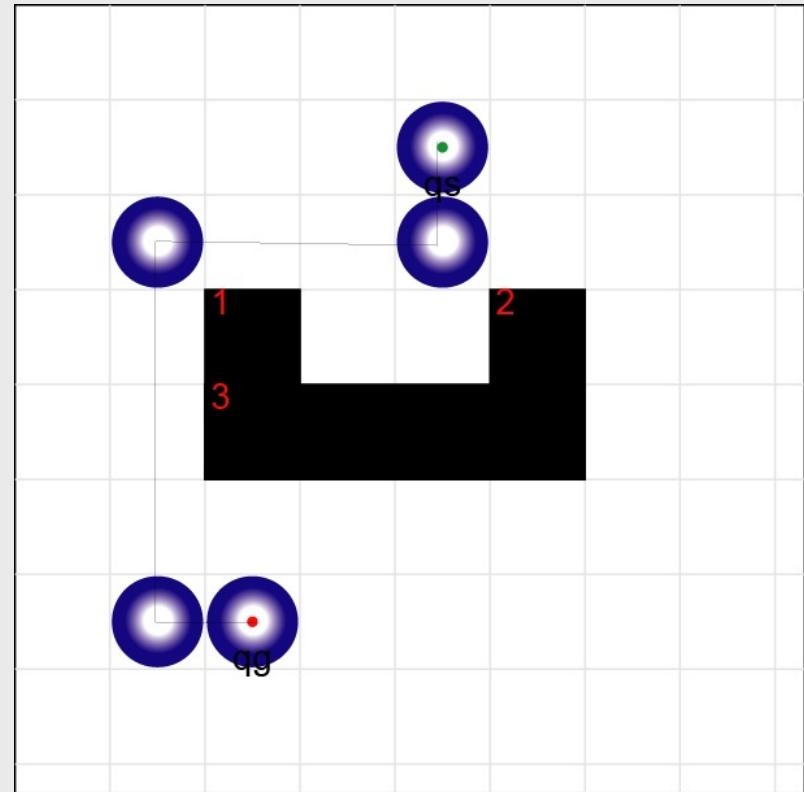
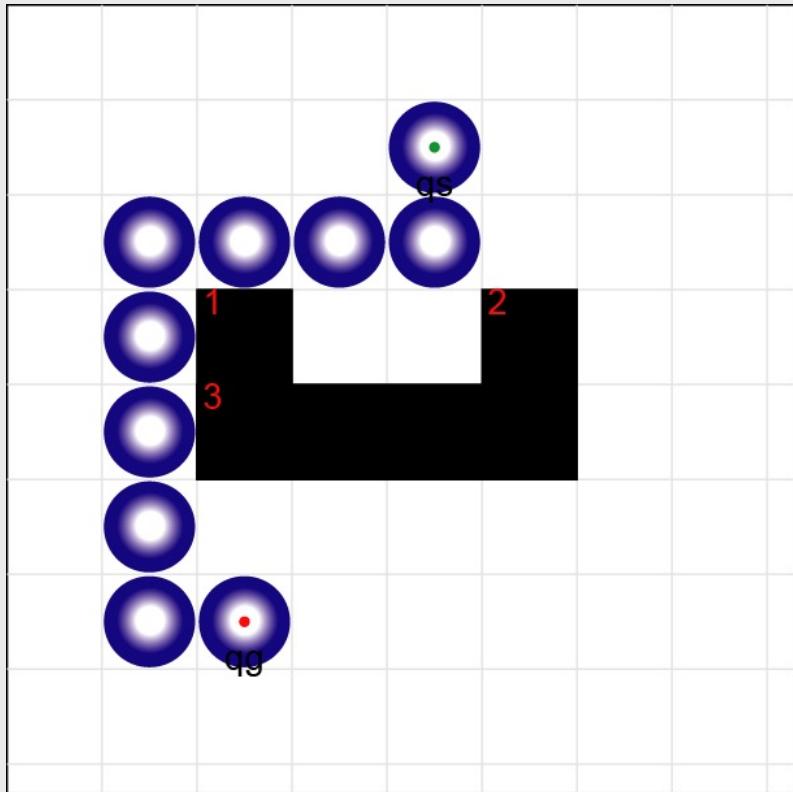


delete
triangle
points



First of all... remove collinear points

- Extremly necessary using A* and sometimes for RRT

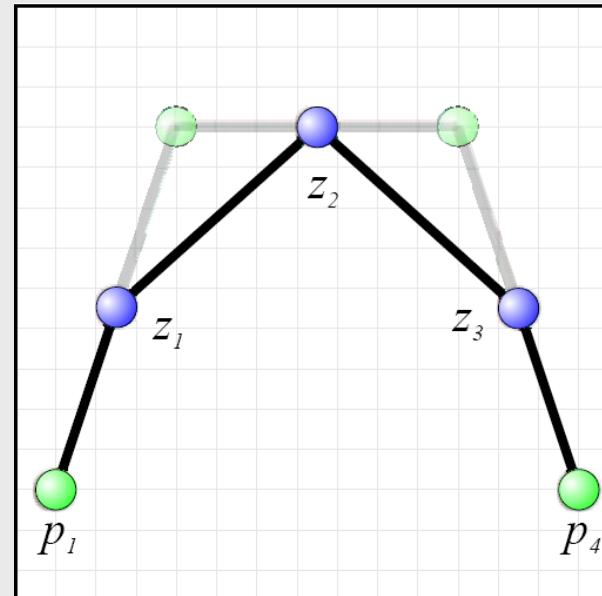
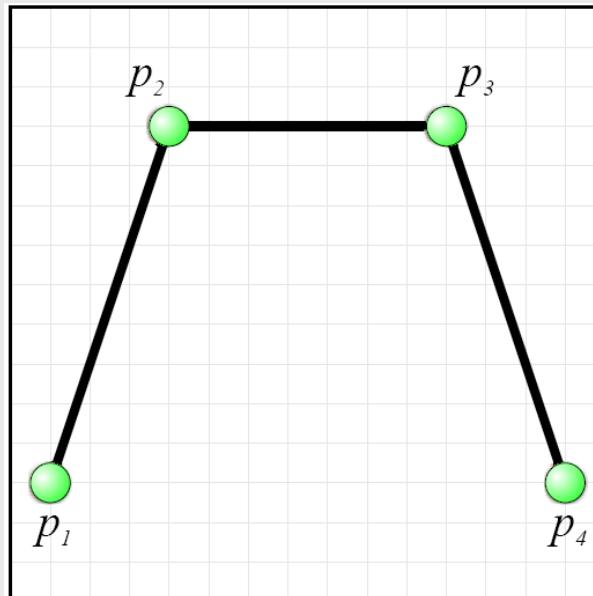


Mid-Point Approach

- For a triple of points, intermediate positions are generated

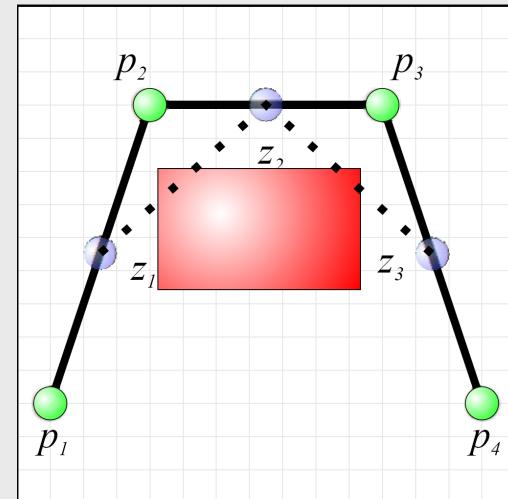
$$z_i = \frac{p_i + p_{i+1}}{2} \text{ and}$$

$$z_{i+1} = \frac{p_{i+1} + p_{i+2}}{2}$$



Mid-Point Approach (2)

- ▶ Algorithm ends
 - ▶ if no pathpoint can removed anymore



- ▶ distance between p_{i+1}, z_i, z_{i+1} is smaller than a certain value ε :

$$\forall i : d(p_{i+1}, z_i) < \varepsilon$$

and

$$\forall i : d(p_{i+1}, z_{i+1}) < \varepsilon$$

- ▶ This approach will end up in a Bezier-shaped curve

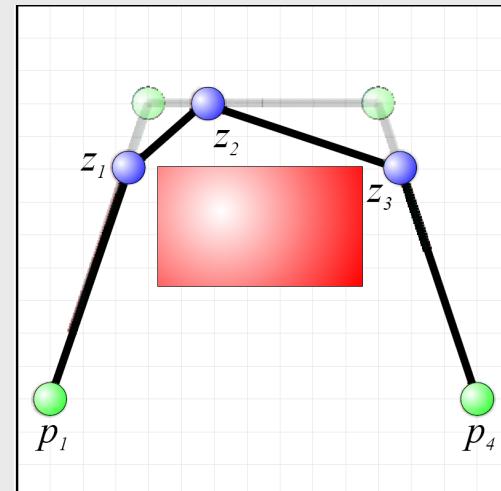
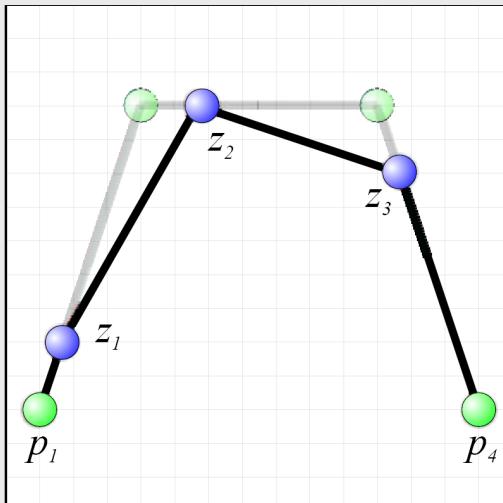
Mid-Point Approach (3)

- ▶ Possible extensions are
 - ▶ Dont use exactly mid of edges

$$z_i = \frac{u^* p_i + (1-u)p_{i+1}}{2}$$

$$z_{i+1} = \frac{u^* p_{i+1} + (1-u)p_{i+2}}{2}$$

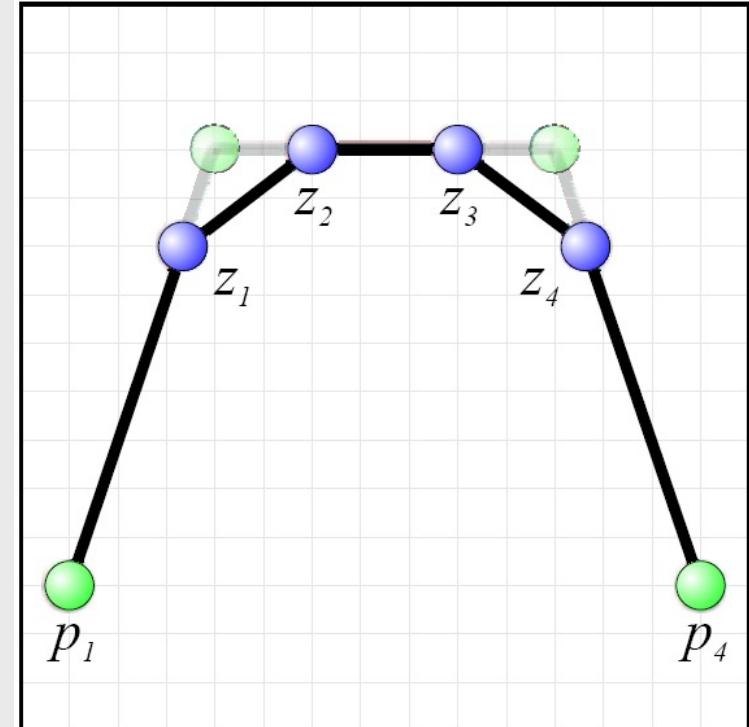
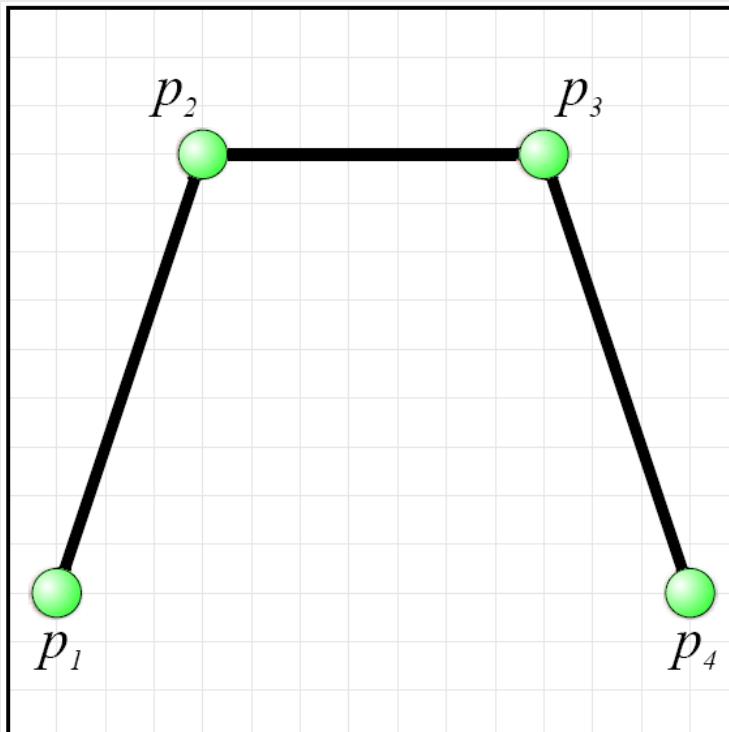
$$0 < u < 1, u \in \mathbb{R}$$



Or of course using any other kind of subdivision...

2-Point-Approach

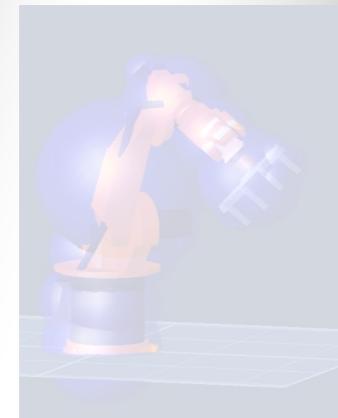
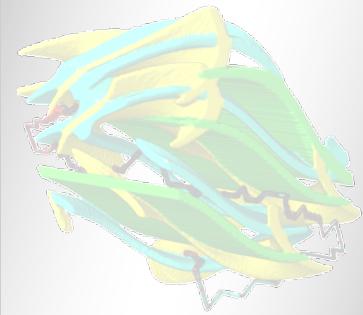
- ▶ Try to replace a pathpoint with 2 new pathpoints and a shorter path segment
 - ▶ Also converging to a curve
 - ▶ Path is shorter but more path points in it



Summary

- ▶ Both approaches Mid-Point and 2-Point will shorten path length avoiding obstacles
- ▶ But: they will also increase number of points of the path
 - bad, bad, bad for using it with an industrial robot controller! These controllers cannot execute fast movements as soon as many points are given in short distances.
 - Need an approach which removes as many points as possible, but also add necessary points

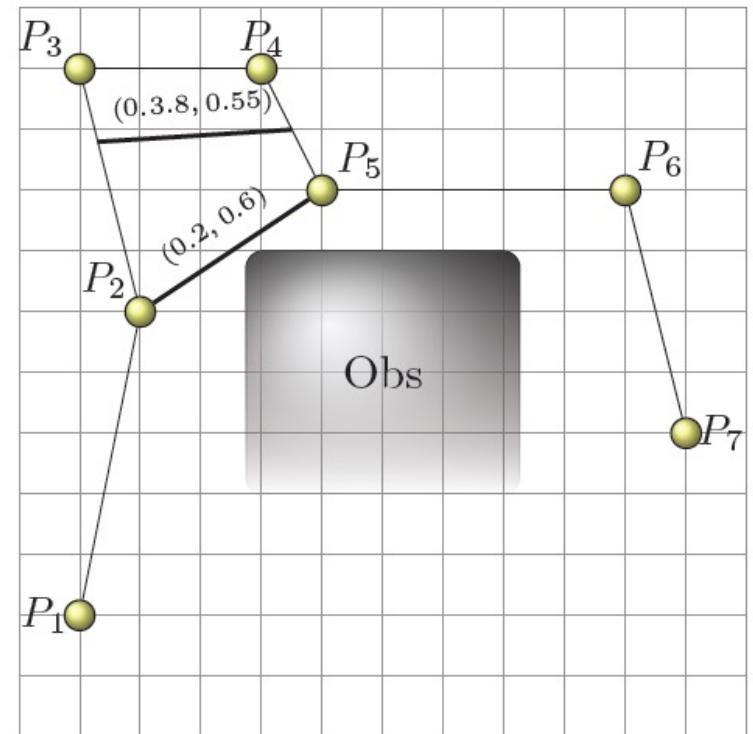
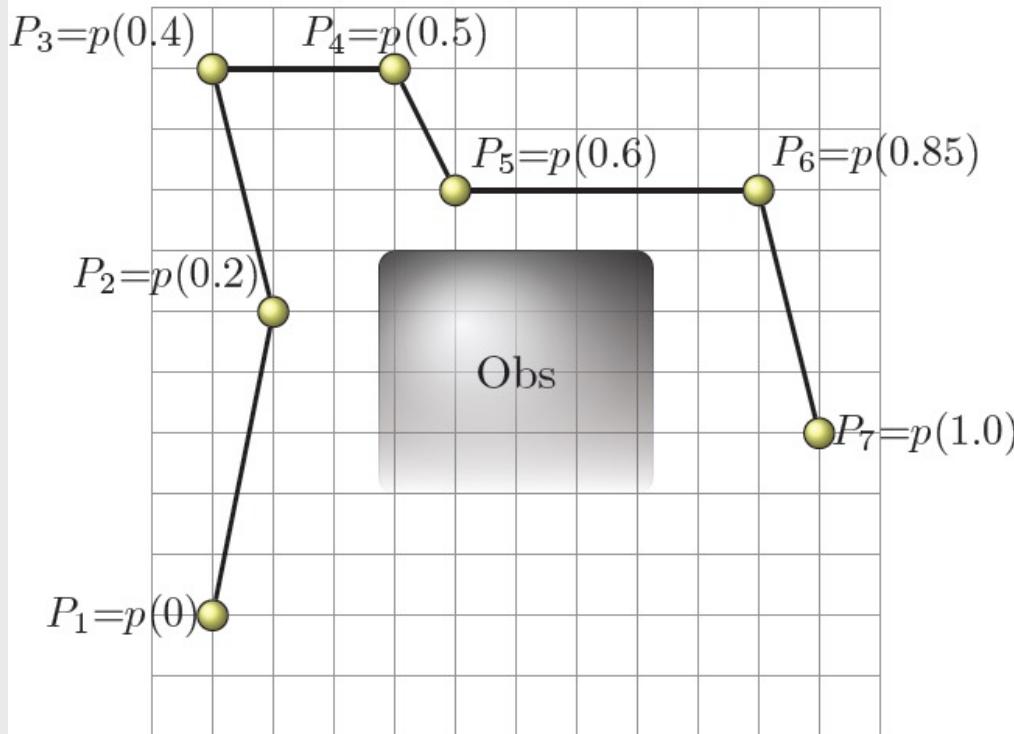
Generic Approach for Path Smoothing



How to

Path definition

- ▶ $P = p(t)$, t in $[0, 1]$
- ▶ $P_{\text{start}} = p(0)$, $P_{\text{end}} = p(1)$



▶ $(I_A, I_B) \rightarrow \text{path segment}$

Basic smoothing algorithm

Glätte(p):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

```
// Bestimme ein Pfadsegment  
 $(l_A, l_B) \leftarrow \text{WählePositionen}(p, wahr)$ 
```

Auswahlfunktion: zwei **Stützstellen des Pfades** durch Auswahl der korrespondierenden Laufvariablen

```
solange ? tue  
    erfolg  $\leftarrow \text{OptimierStrategie}(l_A, l_B, p)$   
     $(l_A, l_B) \leftarrow \text{WähleAbkürzung}(p, l_A, l_B, erfolg)$ 
```

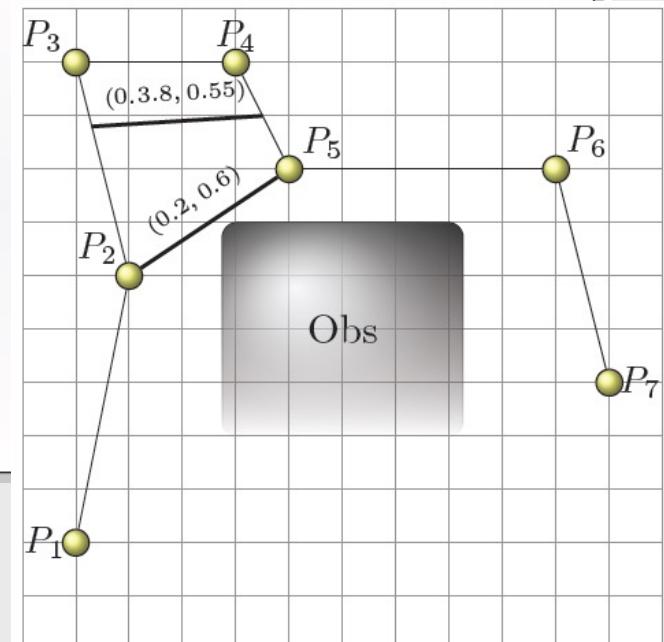
Optimisation strategy

OptimierStrategie(l_A, l_B, p):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

Eingabe : l_A, l_B aktuelle Positionen, zwischen denen getestet werden soll

```
// Wenn das Liniensegment frei ist
wenn FreieVerbindung( $l_A, l_B$ ) dann
    für alle Stützpunkte  $l \in (l_A, l_B)$  tue
        Entferne( $p, l$ )
    // Füge Abkürzung ein
    FügeNeuesSegmentEin( $p, l_A, l_B$ )
    zurück wahr
zurück falsch
```



Shortcut Strategies

- ▶ How to choose start and end point of shortcut?

Glätte(p):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

```
// Bestimme ein Pfadsegment  
( $l_A, l_B$ ) ← WählePositionen( $p, wahr$ )
```

```
solange ? tue  
    erfolg ← OptimierStrategie( $l_A, l_B, p$ )  
    ( $l_A, l_B$ ) ← WähleAbkürzung( $p, l_A, l_B, erfolg$ )
```

- ▶ Pure probabilistic (Did work out for path planners!)
- ▶ „Extended“ Probabilistic (Latombe)
- ▶ Deterministic (Bechthold & Glavina)

Shortcut Strategy: Pure Probabilistic

WähleAbkürzung($p, l_A, l_B, erfolg$):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

Eingabe : l_A, l_B aktuelle Position auf dem Pfad

Eingabe : $erfolg$, gibt an, ob der letzte Glättungsschritt erfolgreich war

// Bestimme ein Pfadsegment

$l_A \leftarrow \text{zufallszahl}(0, 1)$

$l_B \leftarrow \text{zufallszahl}(0, 1)$

zurück (l_A, l_B)

Shortcut Strategy: „Extended“ Probabilistic (Latombe)

WähleAbkürzung($p, l_A, l_B, erfolg$):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

Eingabe : l_A, l_B aktuelle Position auf dem Pfad

Eingabe : $erfolg$, gibt an, ob der letzte Glättungsschritt erfolgreich war

wenn $erfolg$ oder $l_A - l_B < \Delta l_{min}$ dann

$l_{rand} \leftarrow$ zufallszahl(0, 1)

$l_A \leftarrow 0$

$l_B \leftarrow 1$

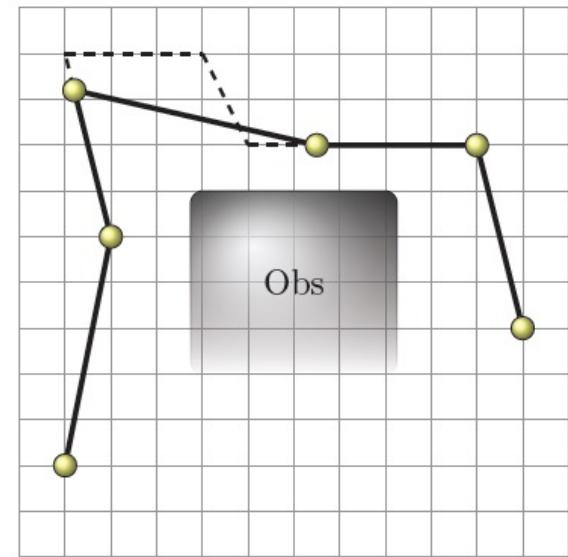
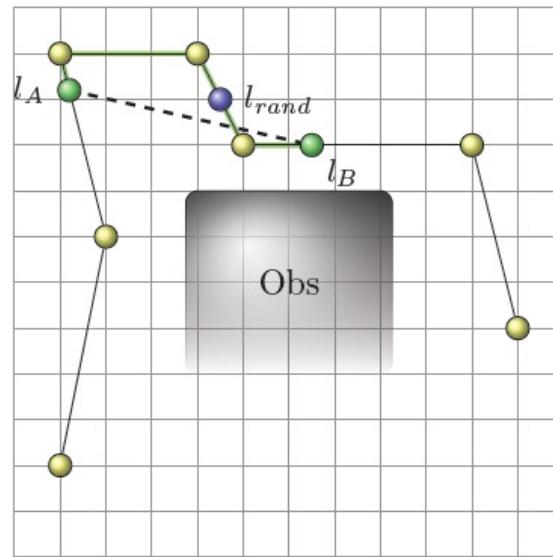
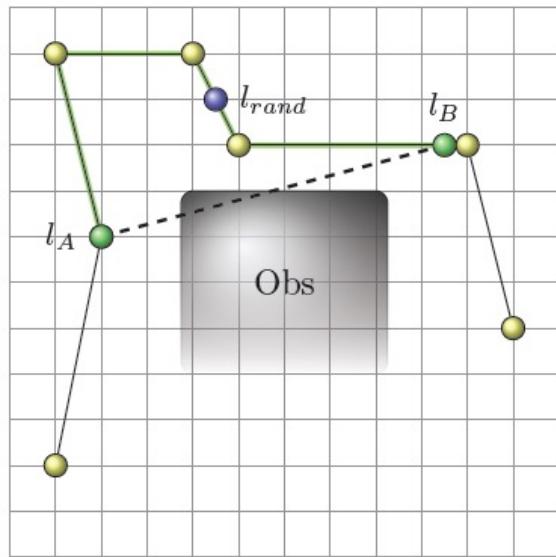
l_{rand} „globale“ Variable, d.h. Wert
bleibt bei weiterem Aufruf der
Funktion erhalten.

$l_A \leftarrow \frac{1}{2} * (l_A + l_{rand})$

$l_B \leftarrow \frac{1}{2} * (l_B + l_{rand})$

zurück (l_A, l_B)

Shortcut Strategy: „Extended“ Probabilistic (Latombe)



Shortcut Strategy: Deterministic (Bechthold & Glavina)

WähleAbkürzung($p, l_A, l_B, erfolg$):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

Eingabe : l_A, l_B aktuelle Position auf dem Pfad

Eingabe : $erfolg$, gibt an, ob der letzte Glättungsschritt erfolgreich war

wenn $erfolg$ oder $l_A - l_B < \Delta l_{min}$ dann

$l_w \leftarrow$ LiefereSchlechtestenStützpunkt(p)

$l_A \leftarrow$ Vorgänger(l_w)

$l_B \leftarrow$ Nachfolger(l_w)

sonst

$l_A \leftarrow \frac{1}{2} * (l_A + l_w)$

$l_B \leftarrow \frac{1}{2} * (l_B + l_w)$

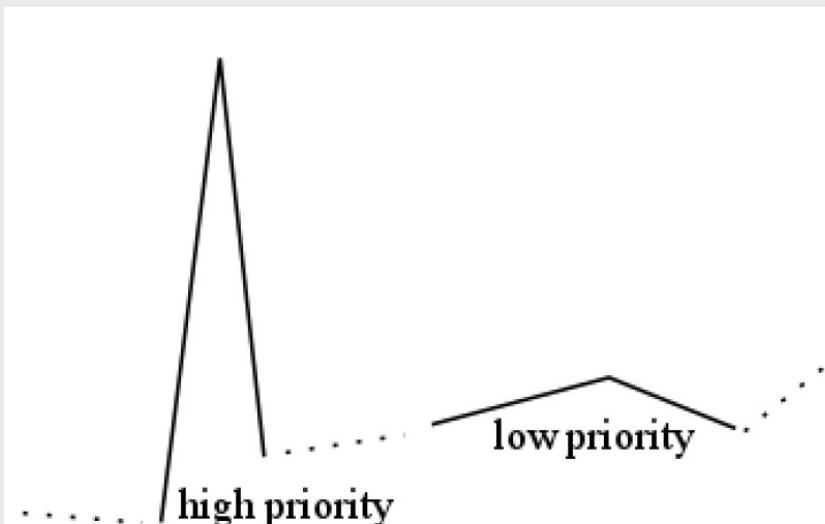
zurück (l_A, l_B)

Deterministic Shortcut: Evaluating points

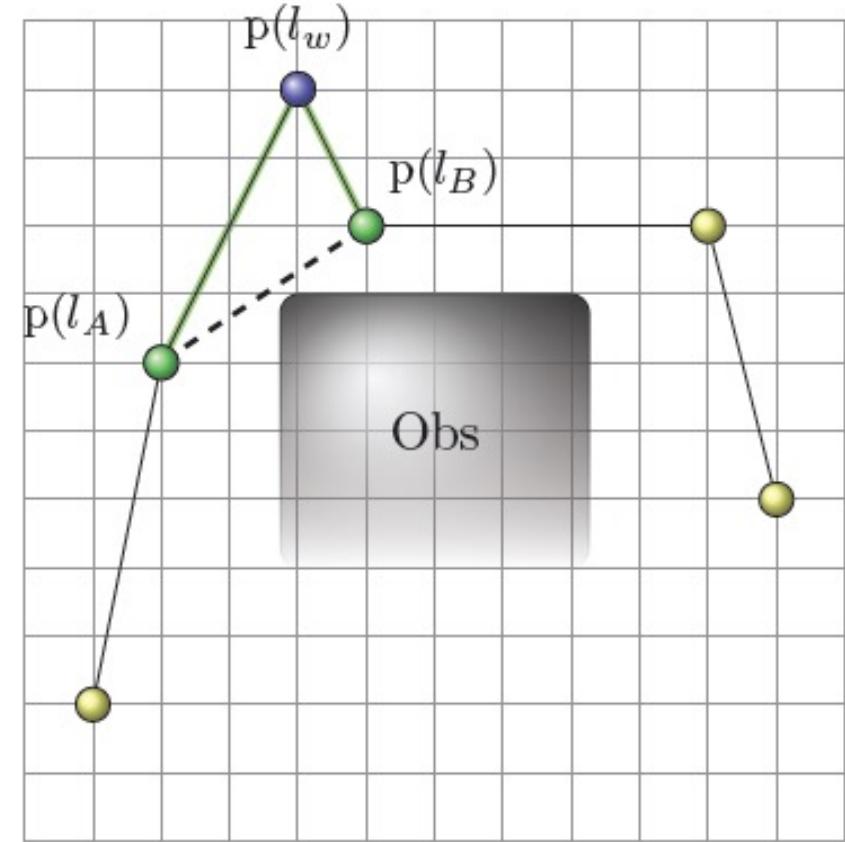
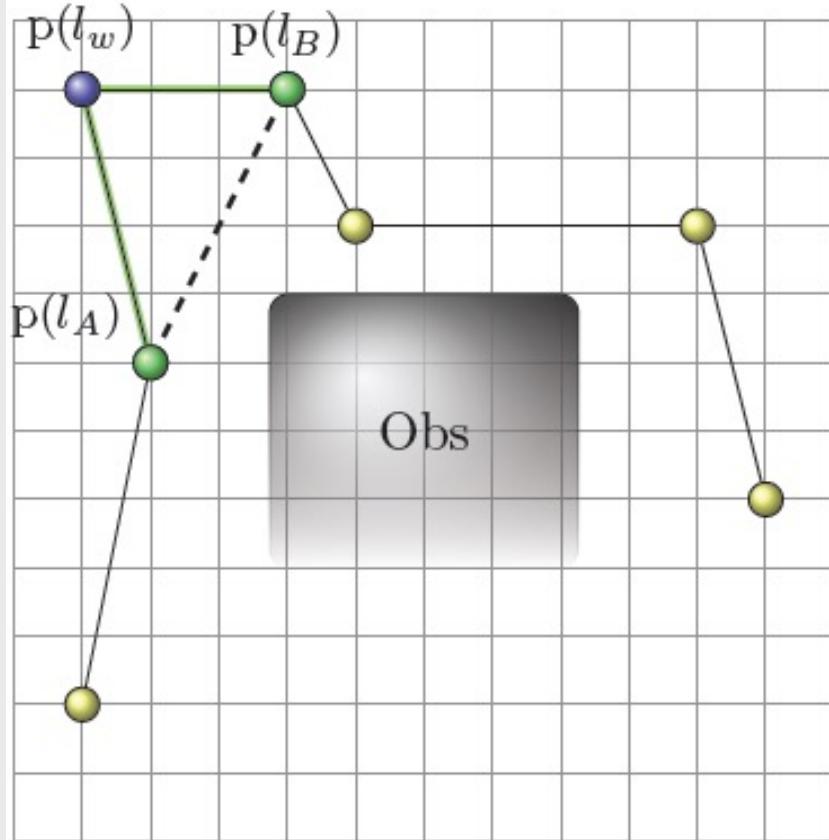
- ▶ Points are measured (evaluated) using:

$$Q_{p_2} = \frac{|p_1p_2| + |p_2p_3|}{|p_1p_3|}$$

- ▶ Points are then inserted in a **priority list** (constant time getting „worst one“)



Shortcut: choosing „worst“ point of path



Operation DelTree

► DelTree (=Mid-Point):

- Generating new points

$$p_{z1} = p_2 + \frac{\overrightarrow{p_2 p_1}}{2^t}$$

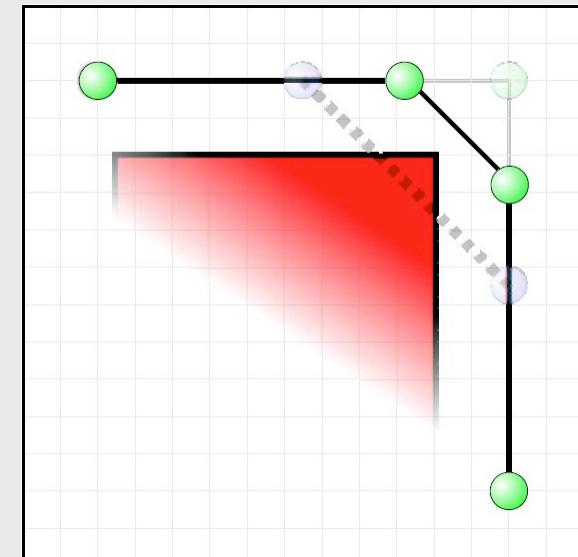
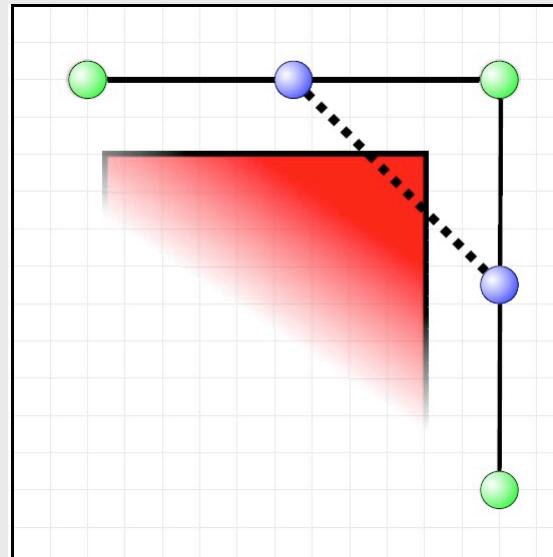
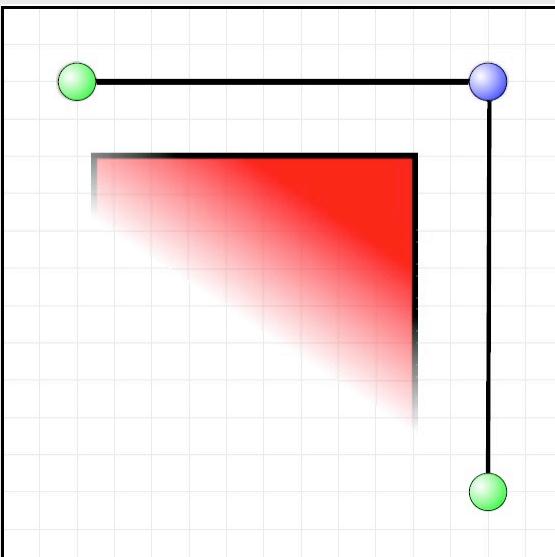
$$p_{z2} = p_2 + \frac{\overrightarrow{p_3 p_1}}{2^t}$$

- If connection fails \rightarrow bisection ($t = t+1$)

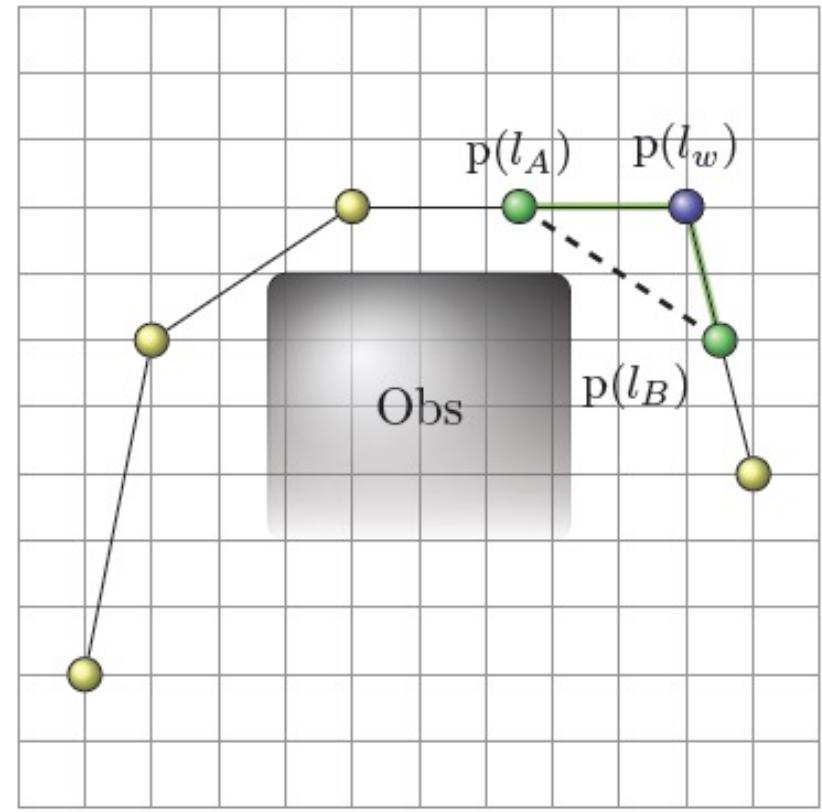
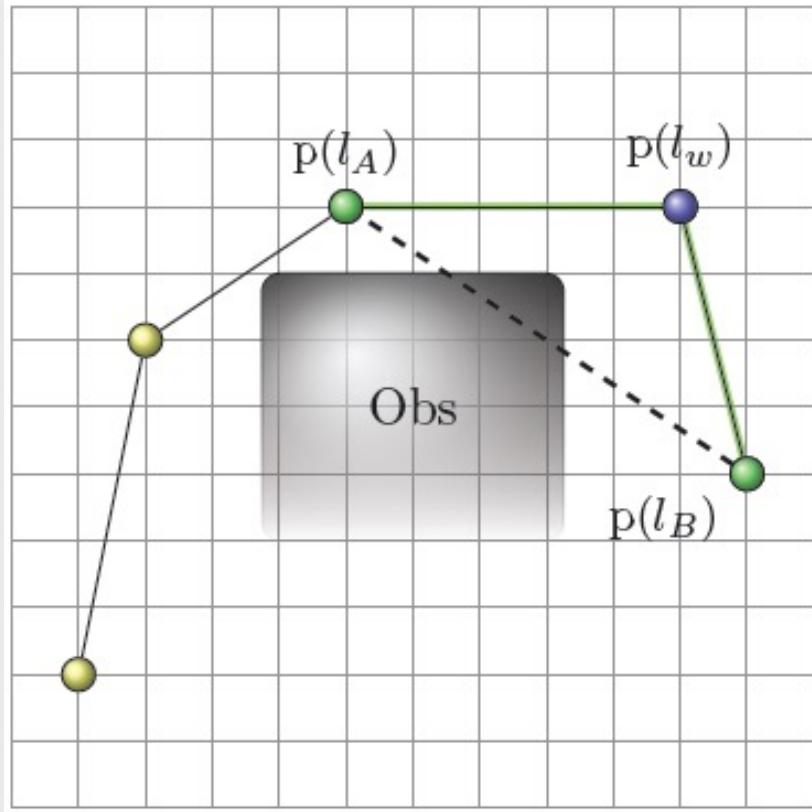
- Stopping if

$$\frac{|p_2 p_1|}{2^t} < \varepsilon$$

$$\frac{|p_2 p_3|}{2^t} < \varepsilon.$$



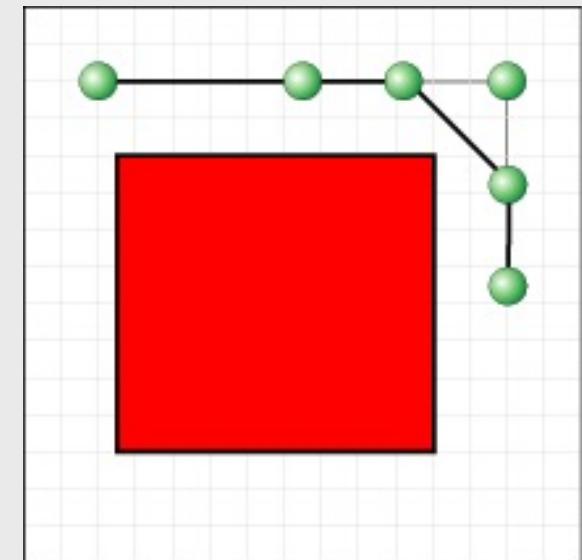
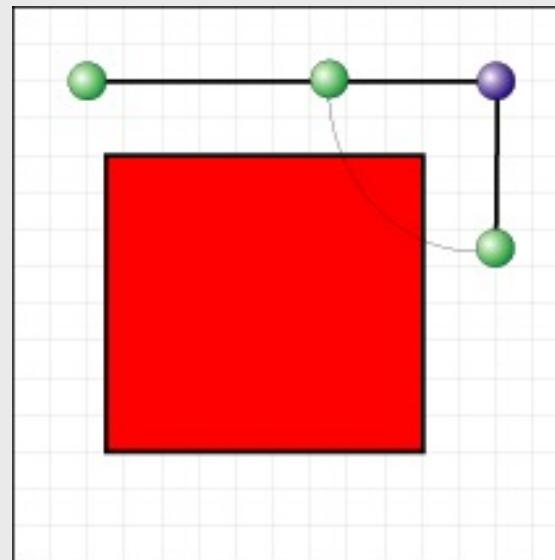
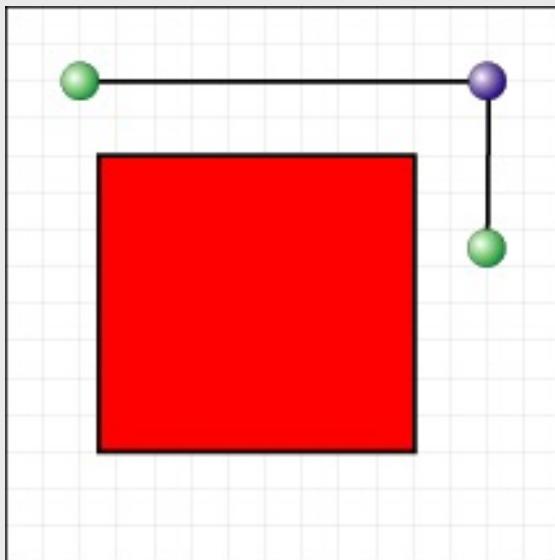
Operation DelTree



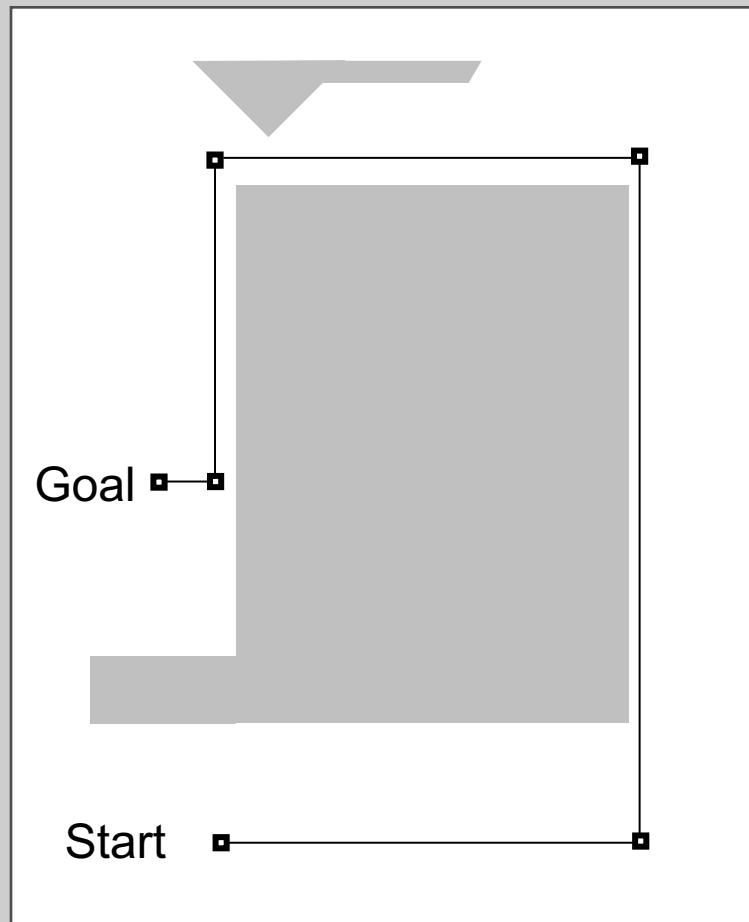
Operation DelEqual (alt. Strategy)

► DelEqual

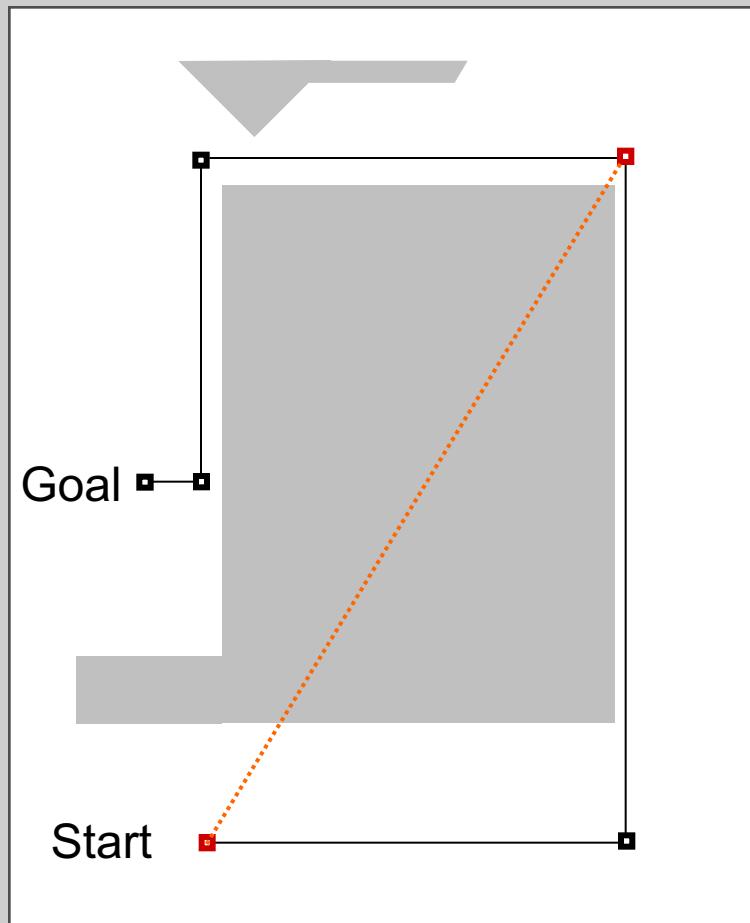
- Generate new point on longer segment, with length of shorter segment



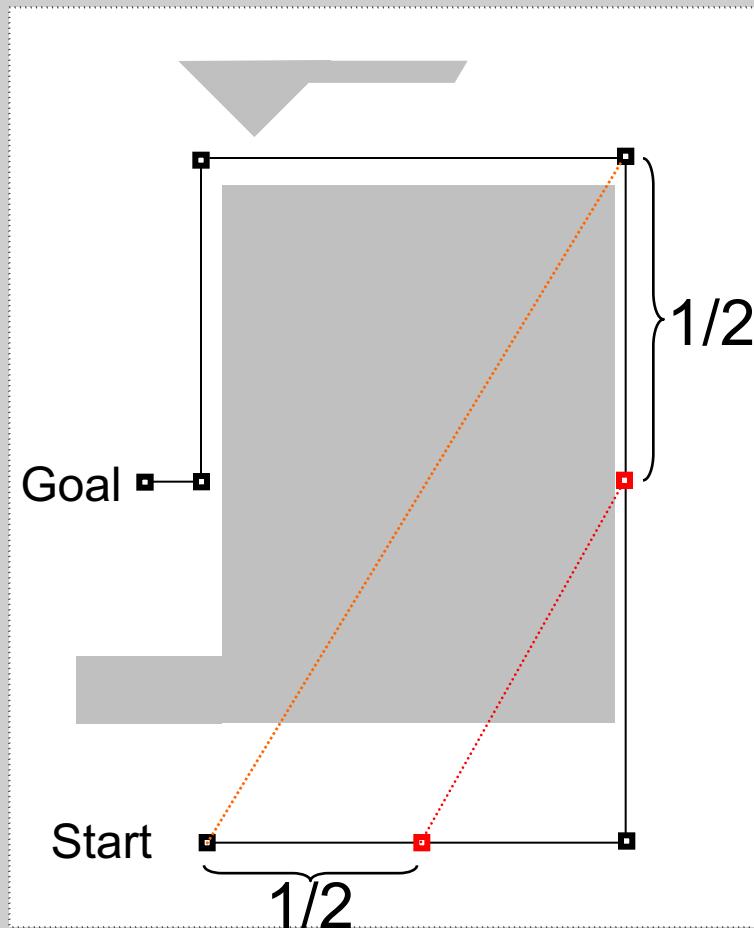
Example for a whole path



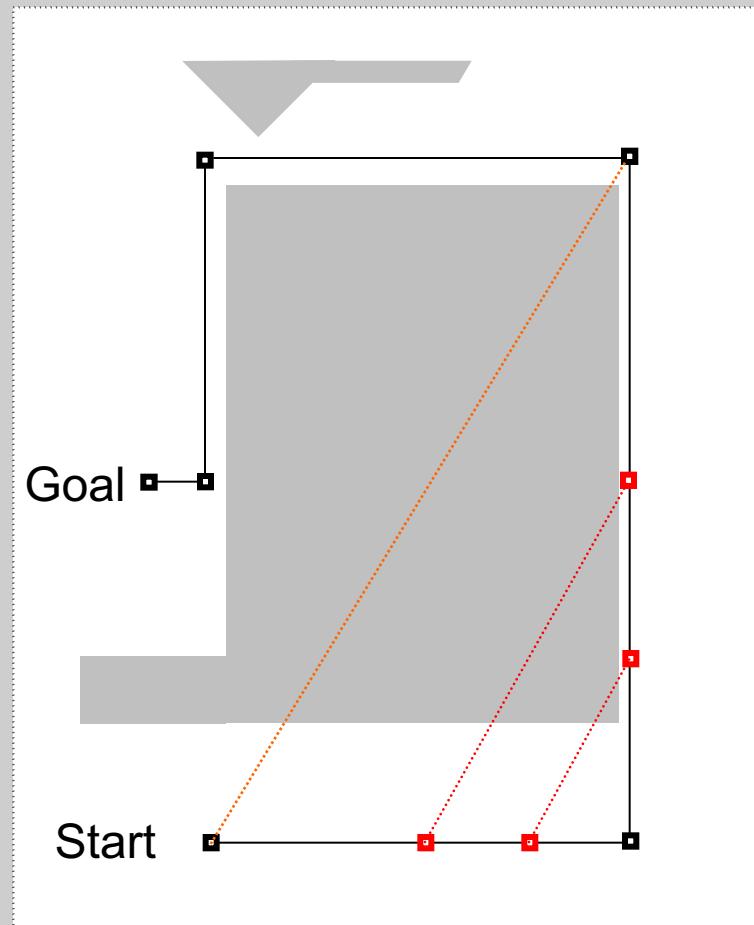
Example for a whole path



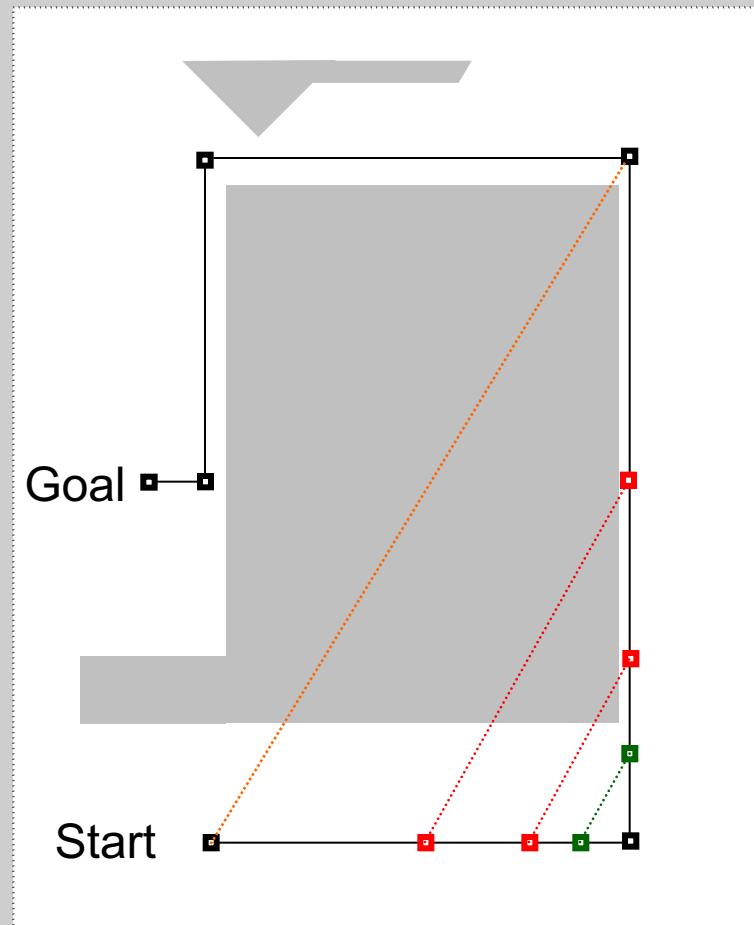
Example for a whole path



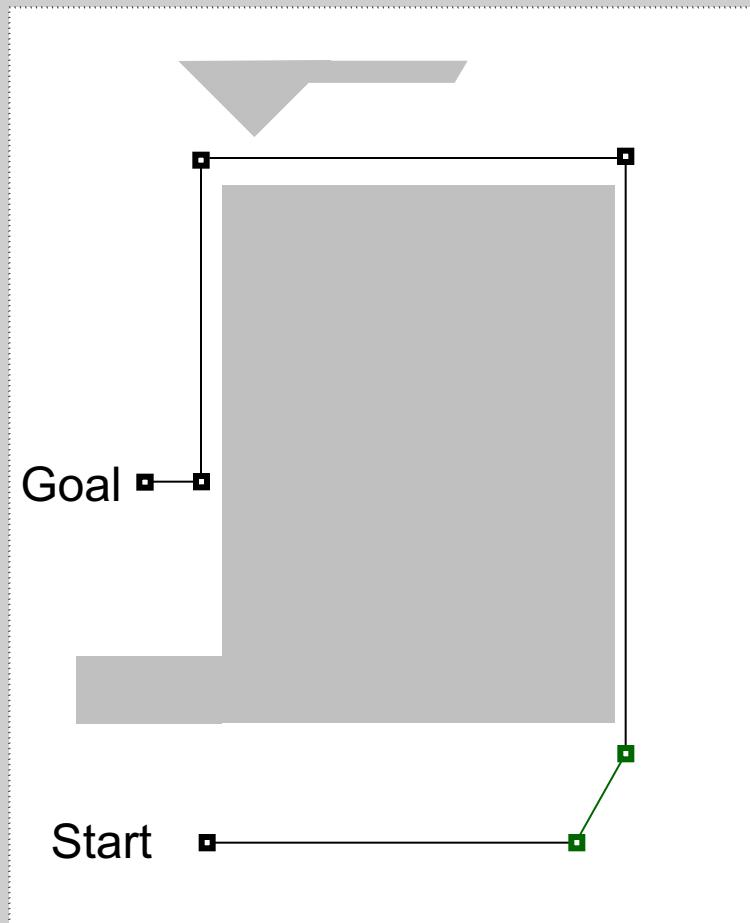
Example for a whole path



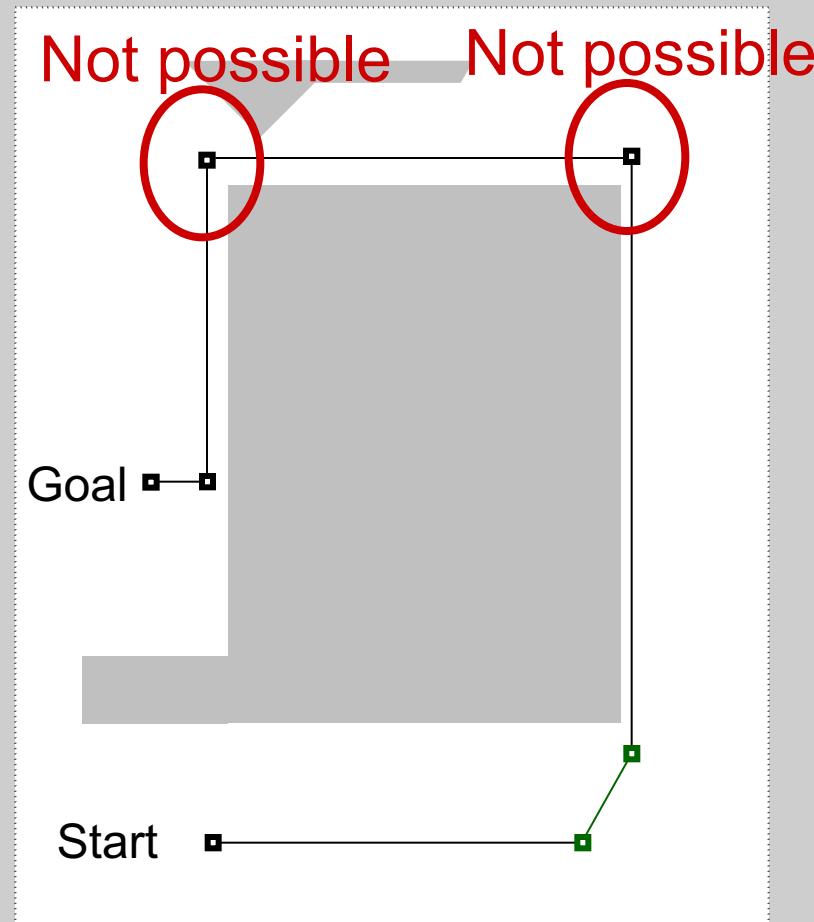
Example for a whole path



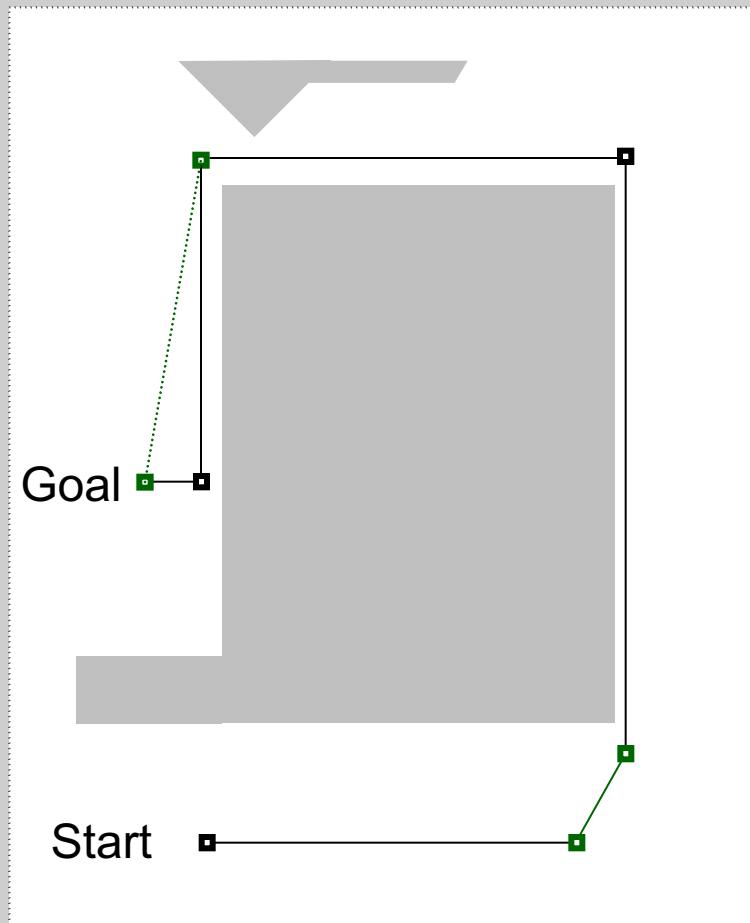
Example for a whole path



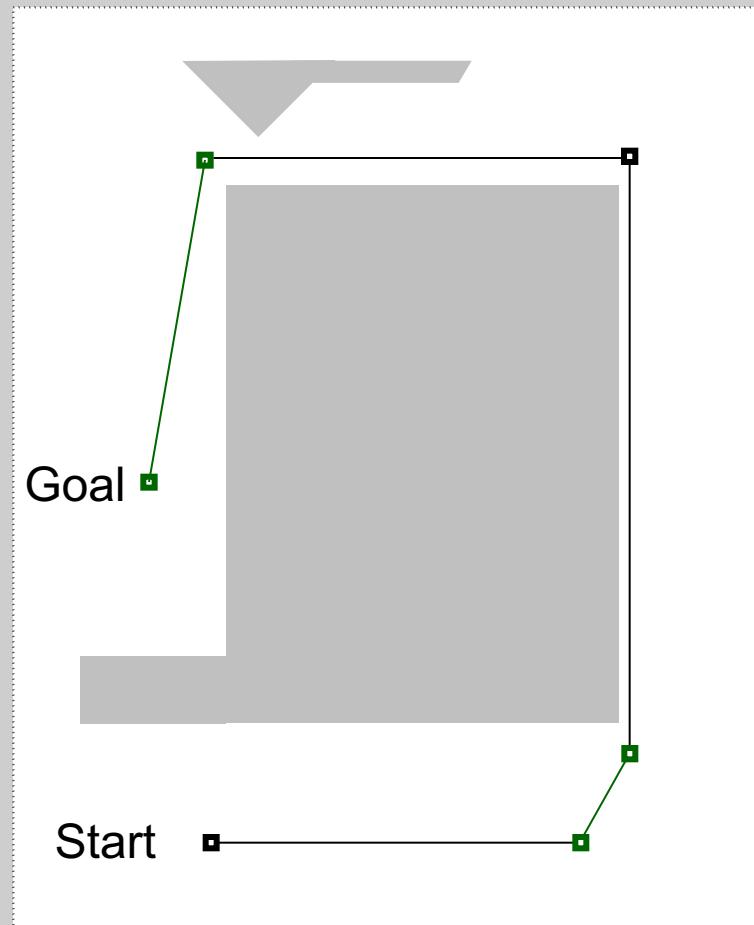
Example for a whole path



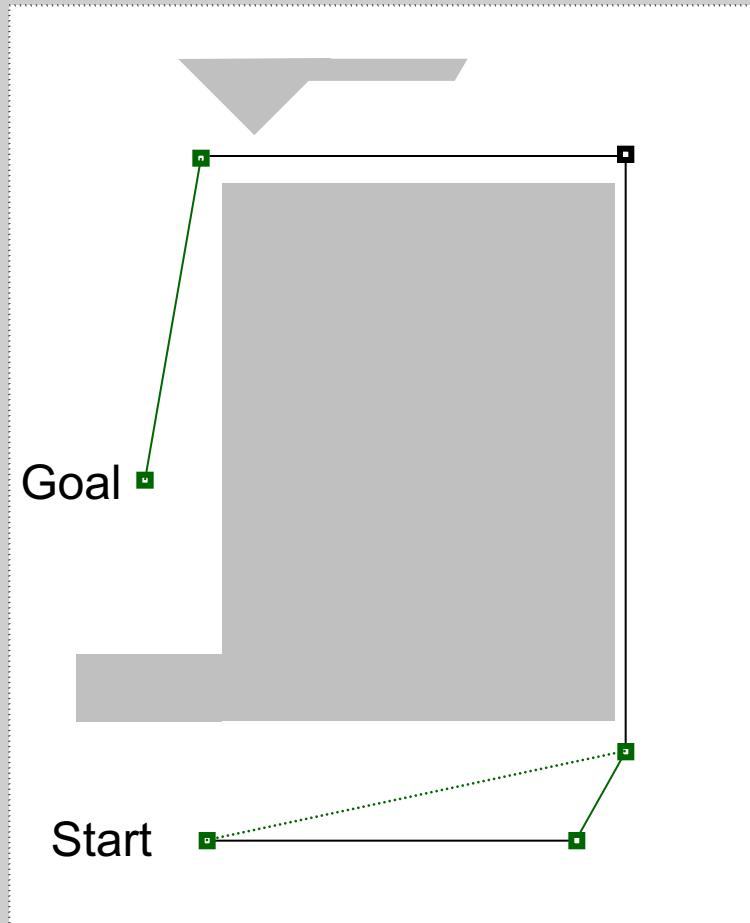
Example for a whole path



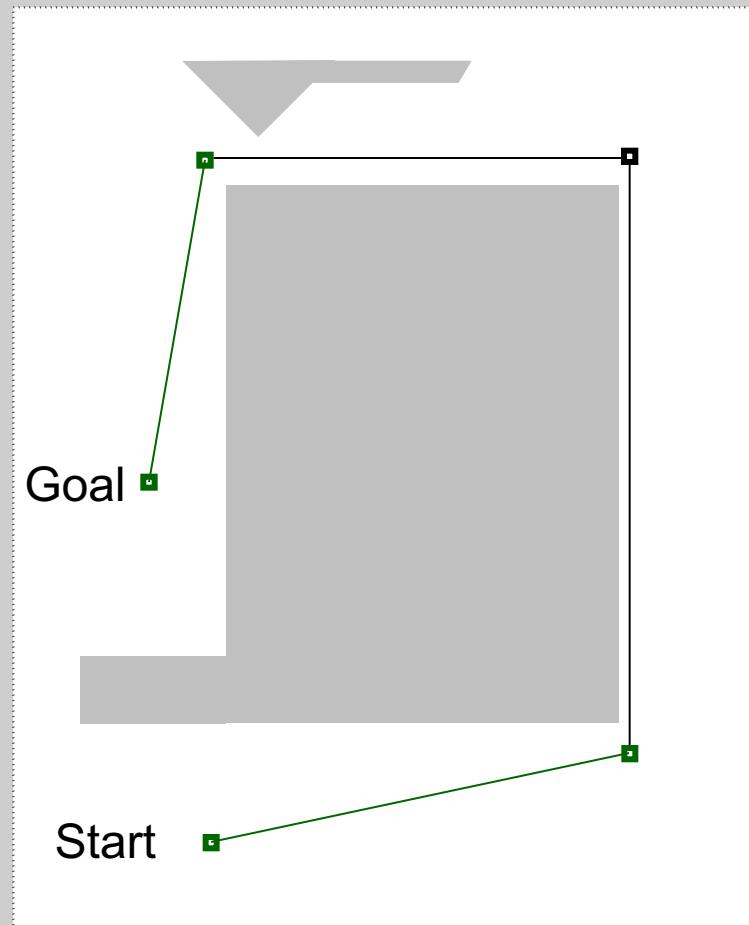
Example for a whole path



Example for a whole path

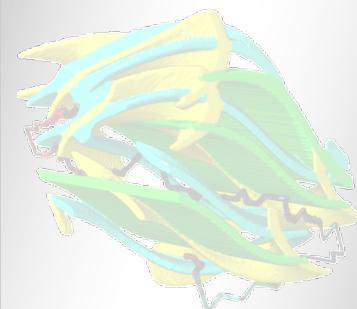
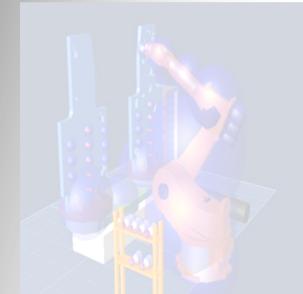
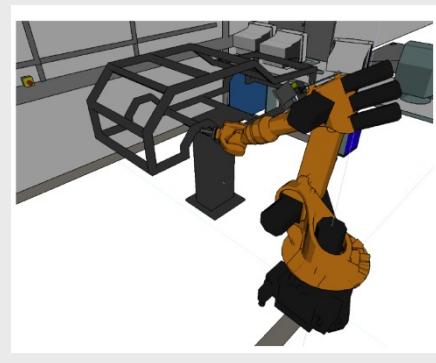
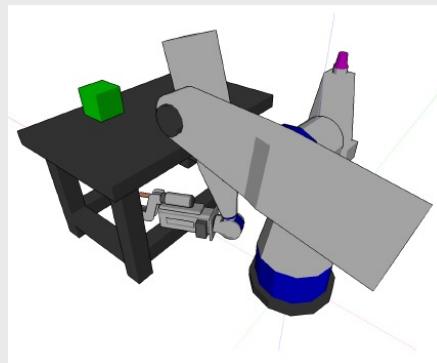
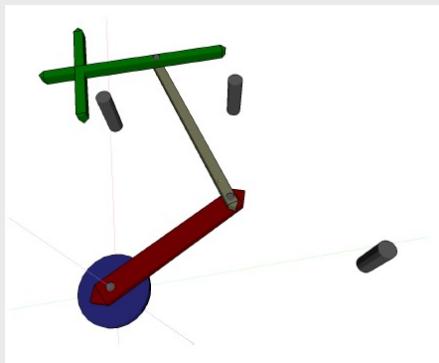


Example for a whole path : Result



Comparison of Smoothing strategies

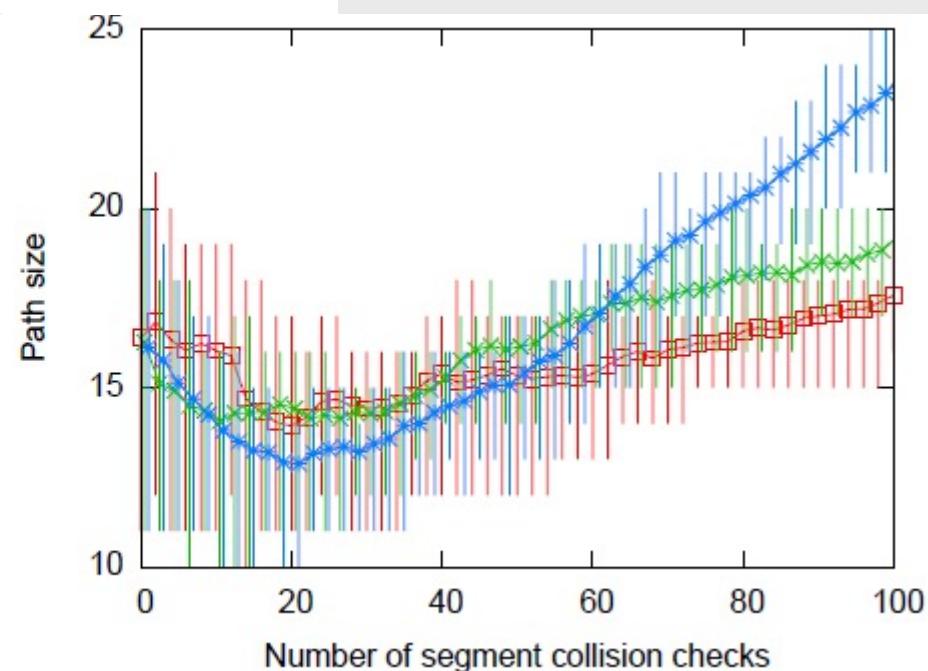
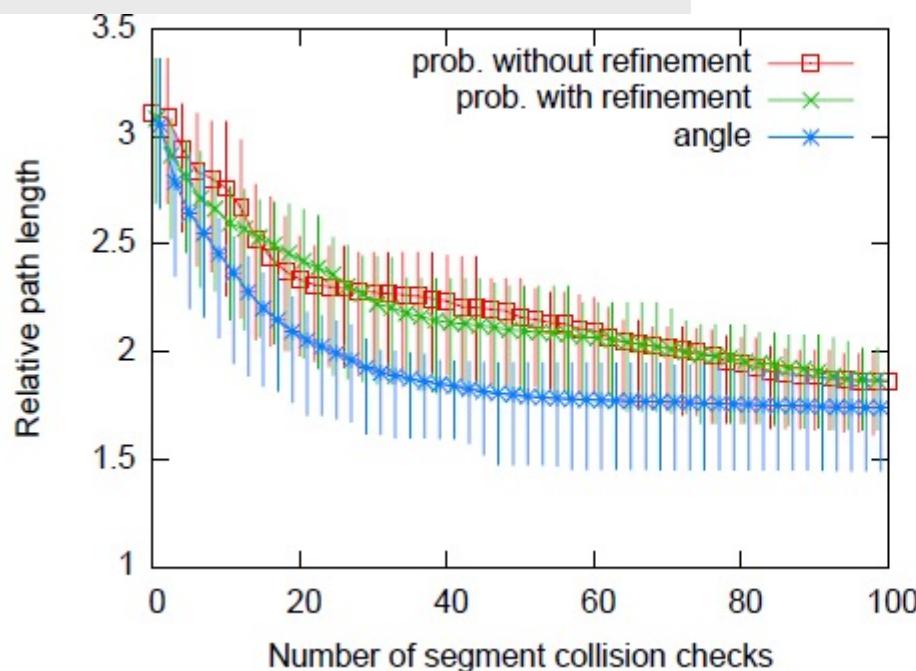
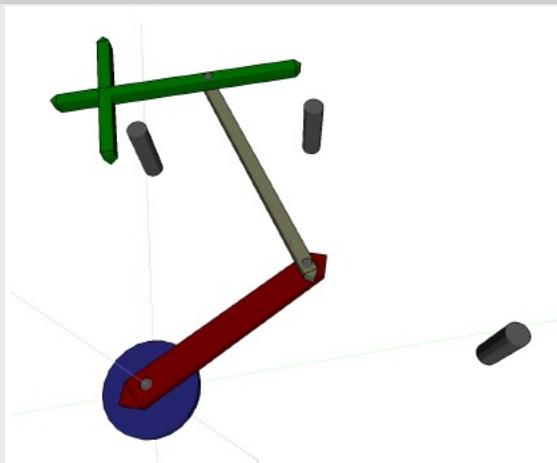
Test Environments



Comparison of different smoothing strategies

- ▶ Strategies:
 - ▶ Probabilistic
 - ▶ Probabilistic extended (Latombe)
 - ▶ Deterministic based on angle (Bechthold & Glavina)
- ▶ Tests for every test environment:
 - ▶ 3 different tasks executed 10 times
 - ▶ 3 different planer (k-Closest, Lazy, RRT)
- ▶ 90 experiments -> 90 executing pathplanner calls
- ▶ Calling 100 times smoothing algorithm
- ▶ Path quality: $\left(\frac{l_{\text{experiment}}}{\min(l)} \right)$

Smoothing Comparison: Benchmark I



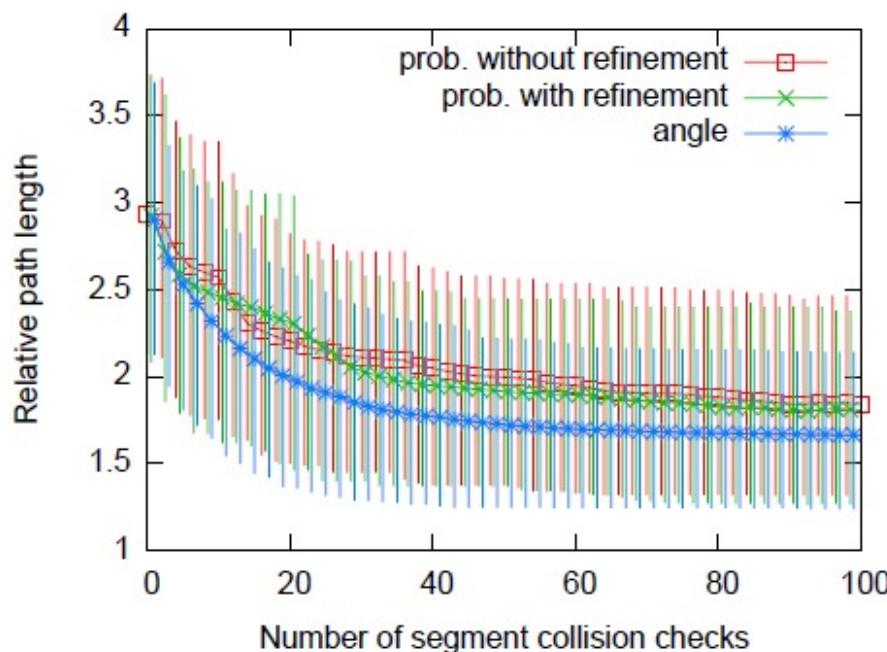
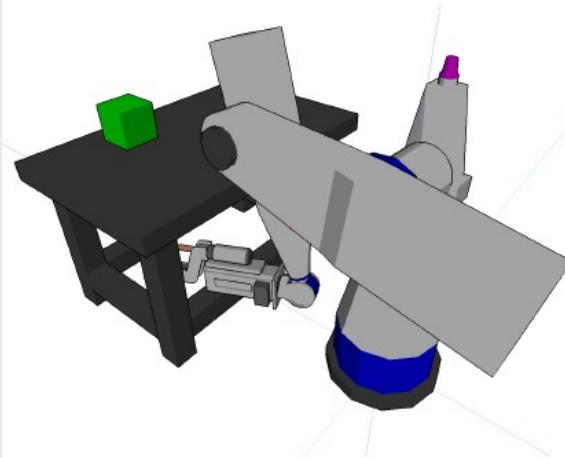
[Mages 06]

HIKA

Björn Hein

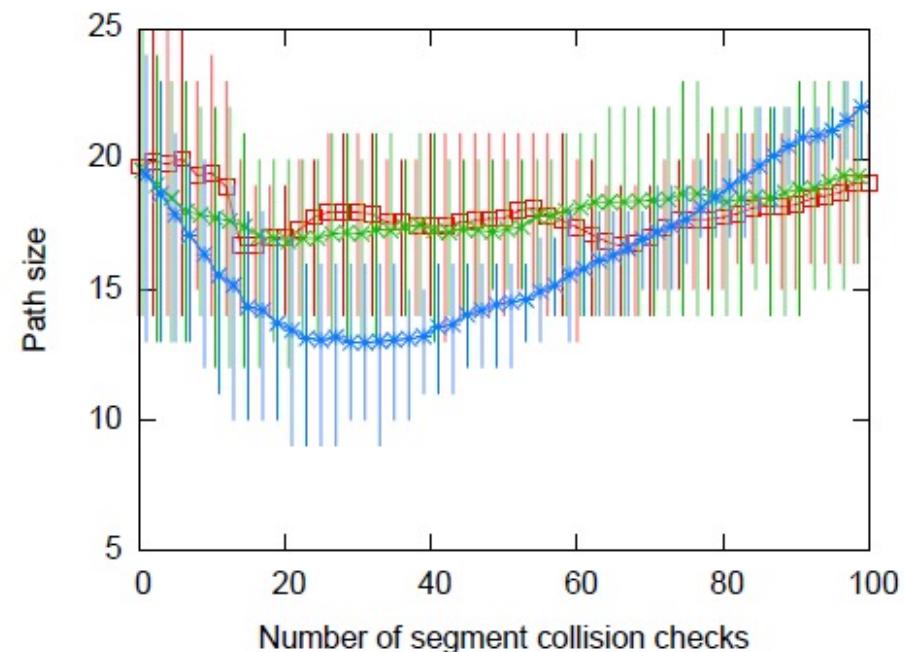
-47-

Smoothing Comparison: Benchmark II

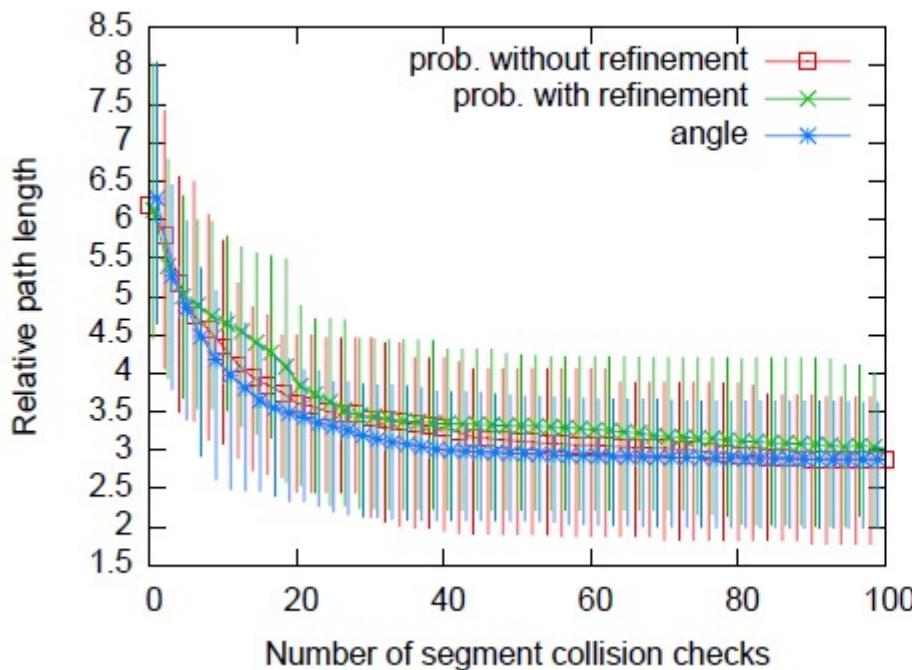
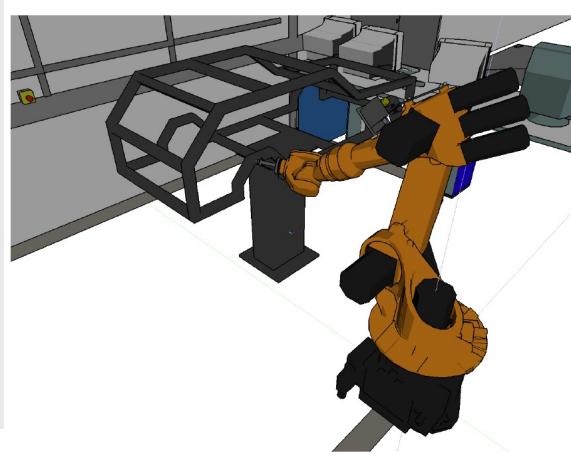


[Mages 06]

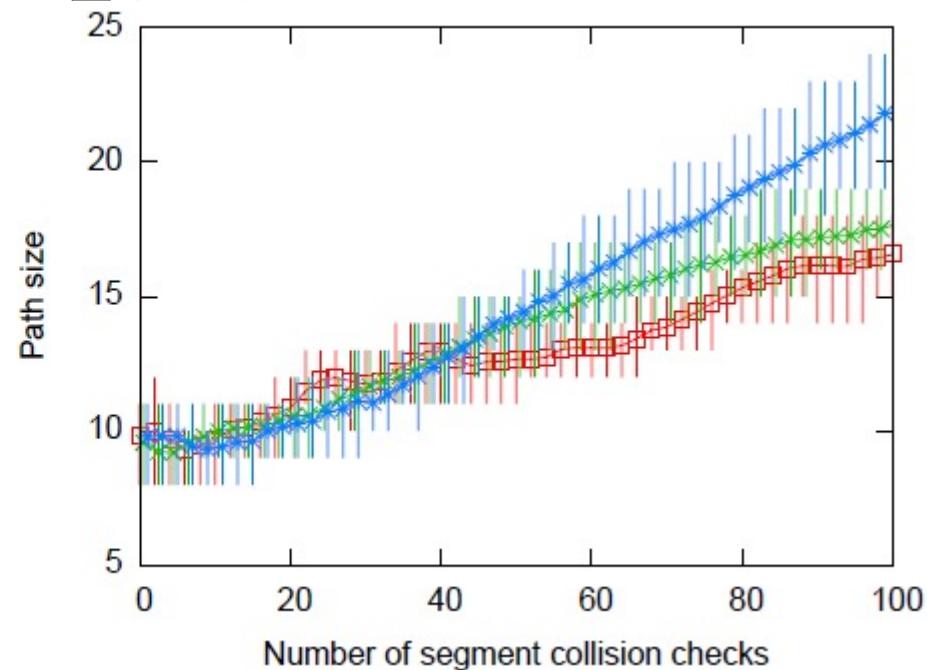
IKA



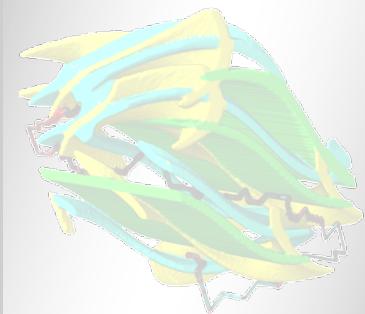
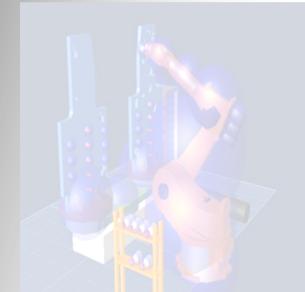
Smoothing Comparison: Benchmark III



[Mages 06]



Termination of smoothing?



Basic smoothing Algorithm with extended termination

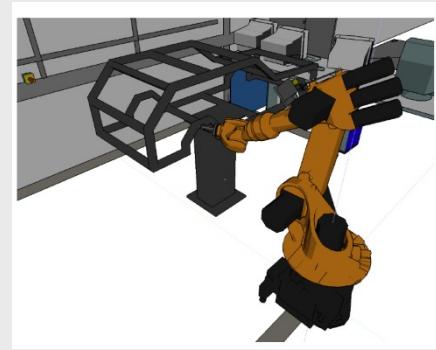
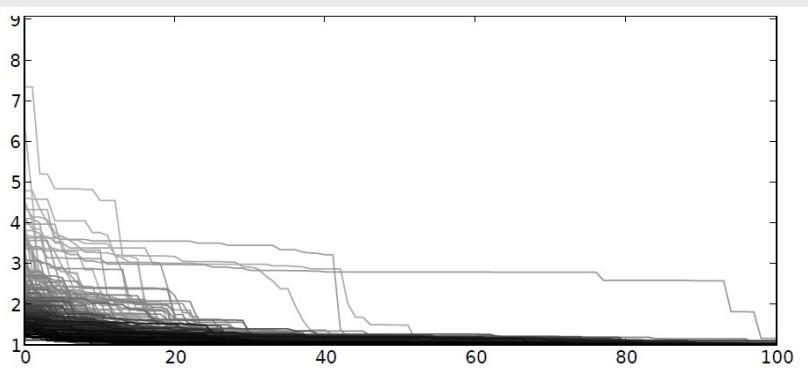
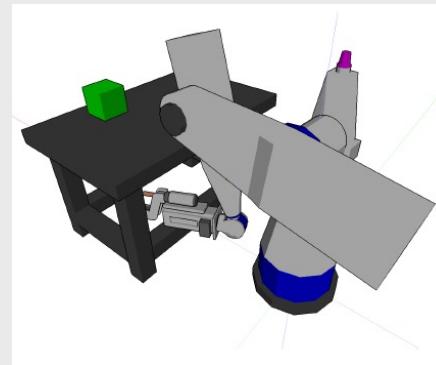
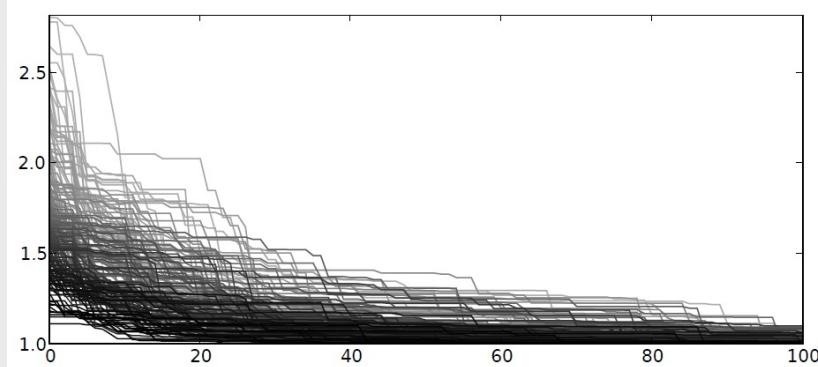
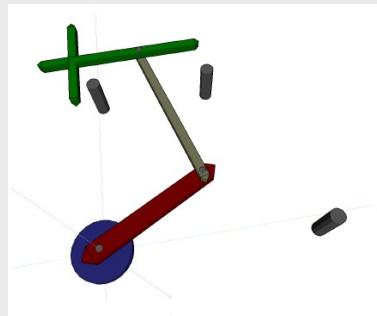
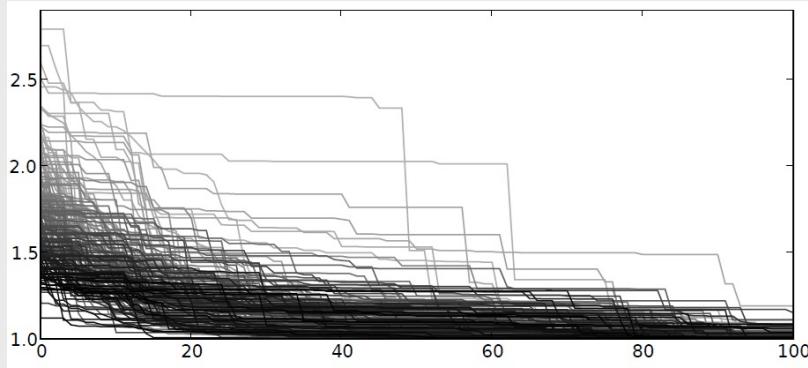
Glätte(p):

Eingabe : Pfad p bestehend aus einer geordneten Menge von Stützpunkten

```
// Bestimme ein Pfadsegment  
 $(l_A, l_B) \leftarrow \text{WählePositionen}(p, wahr)$ 
```

```
solange ? tue  
    erfolg  $\leftarrow \text{OptimierStrategie}(l_A, l_B, p)$   
     $(l_A, l_B) \leftarrow \text{WähleAbkürzung}(p, l_A, l_B, erfolg)$   
    wenn TERMINIERE( $p$ ) dann  
        zurück ;
```

Approximation Behavior: 100 steps



Termination strategy

Daten : Pfad p ist eine sortierte Menge von Stützpunkten

Daten : Schlange q

$l = p.\text{LÄNGE}();$

$q.\text{PUSH}(l);$

wenn $q.\text{GRÖSSE}() < s_{max}$ **dann zurück falsch;**

$q.\text{POP}();$ \leftarrow Älteste Element entfernen

wenn $q.\text{VARIANZ}() < v_{min}$ **dann zurück wahr;**

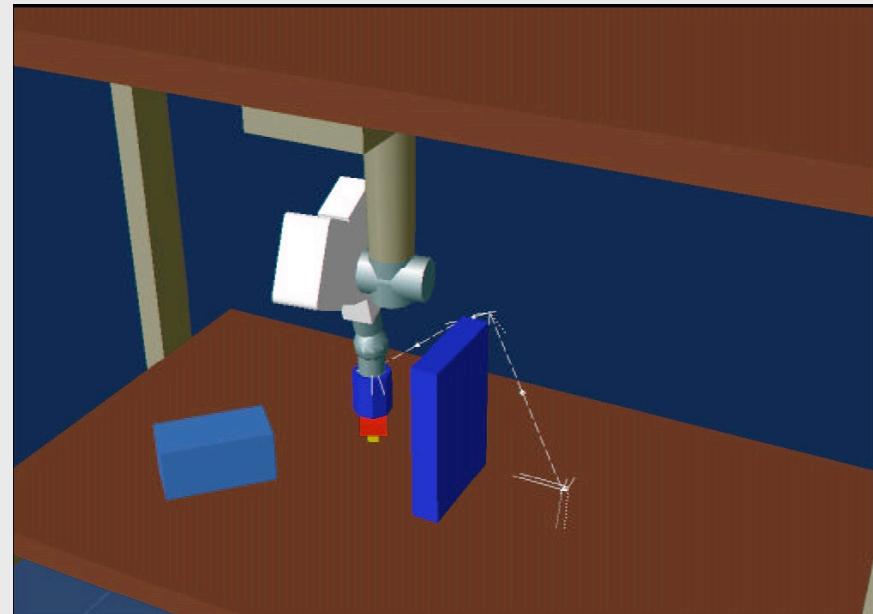
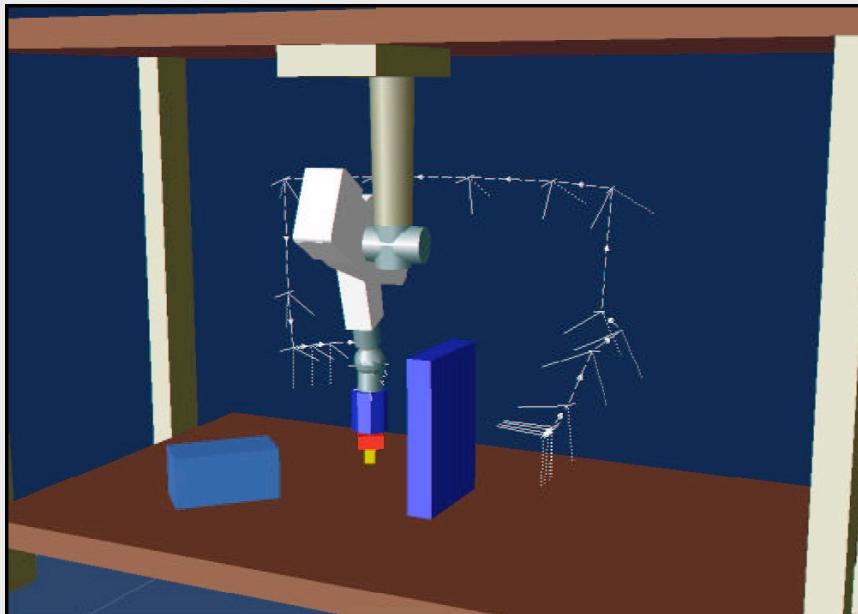
zurück falsch;

Conclusion

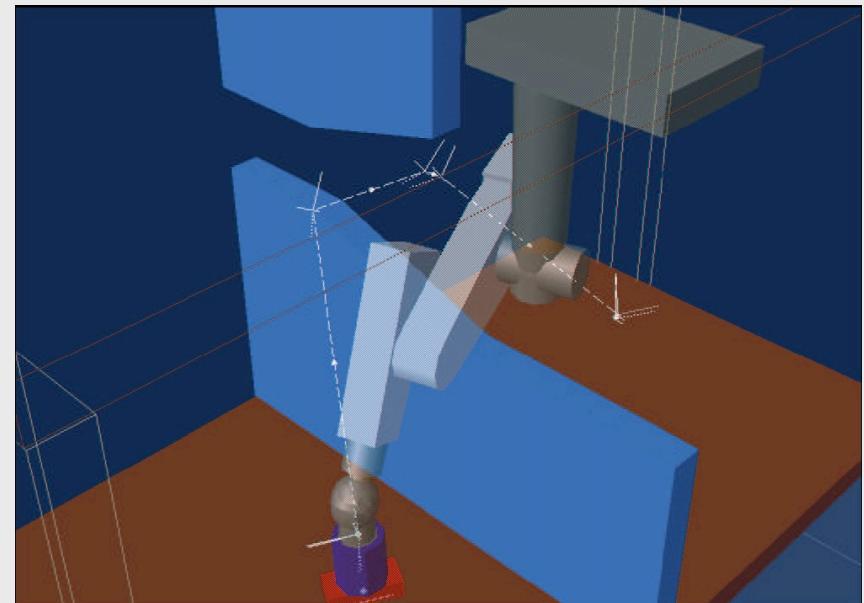
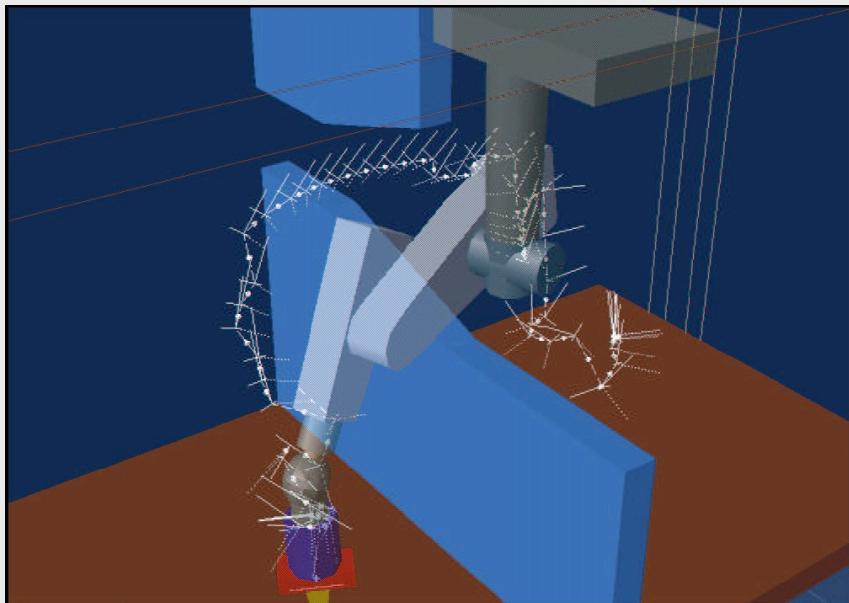
Observations:

- ▶ All three strategies tend to same path quality
- ▶ Deterministic approach tends to converge faster
- ▶ Variance of deterministic approach smaller during optimisation

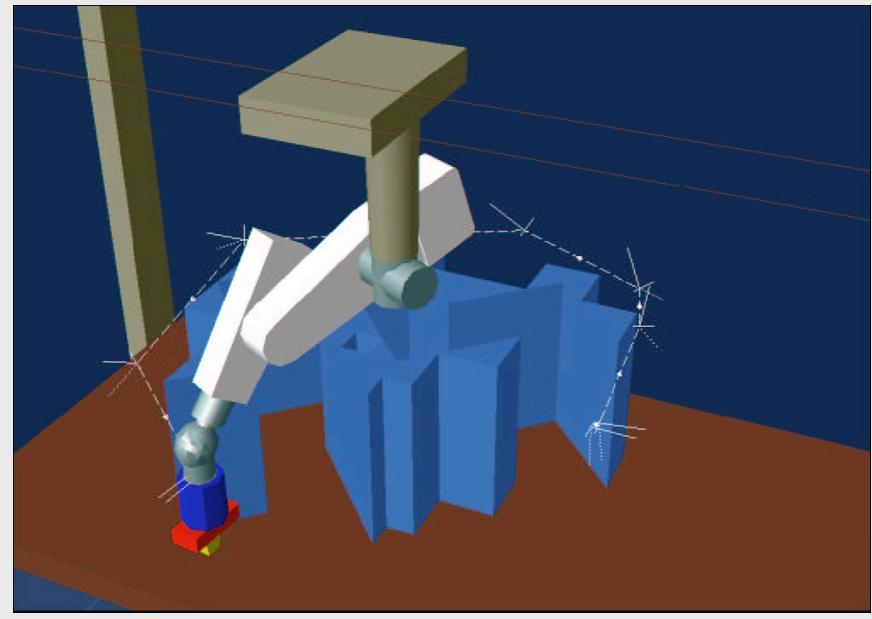
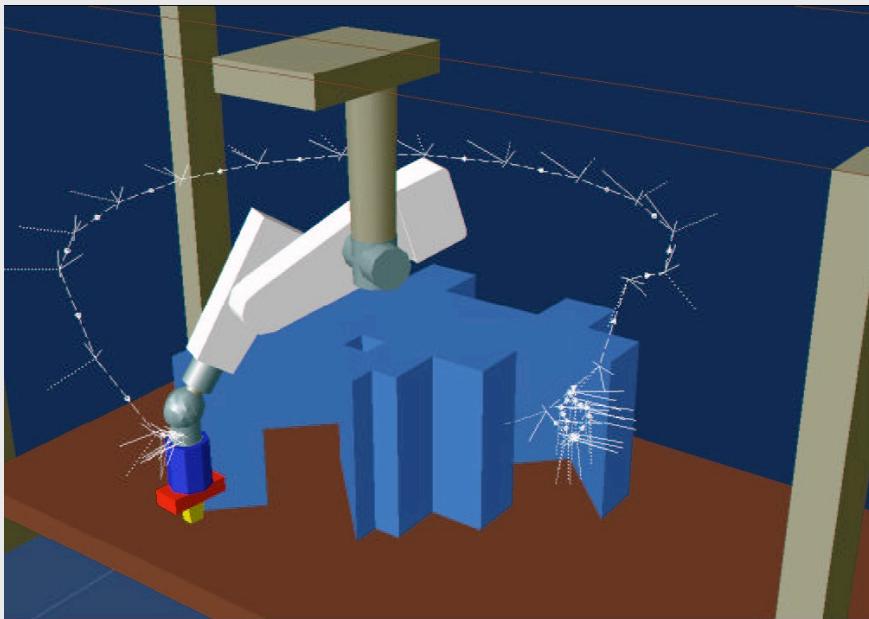
6-DOF-Example: Simple



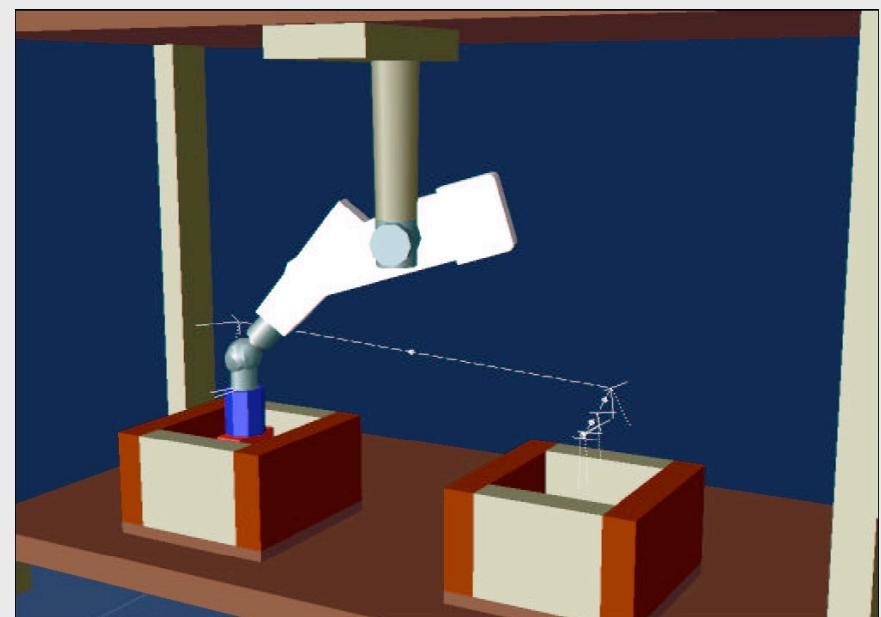
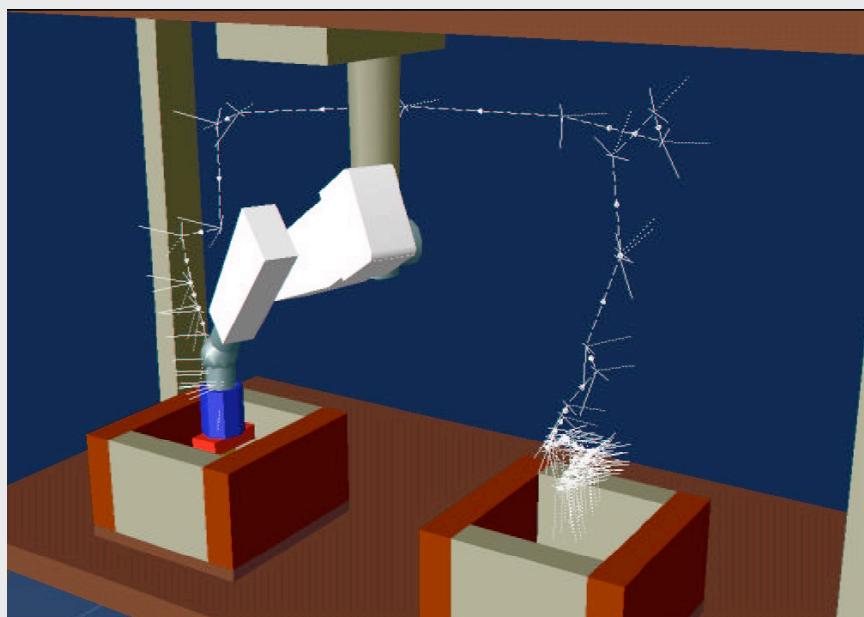
6-DOF-Example: Bottleneck



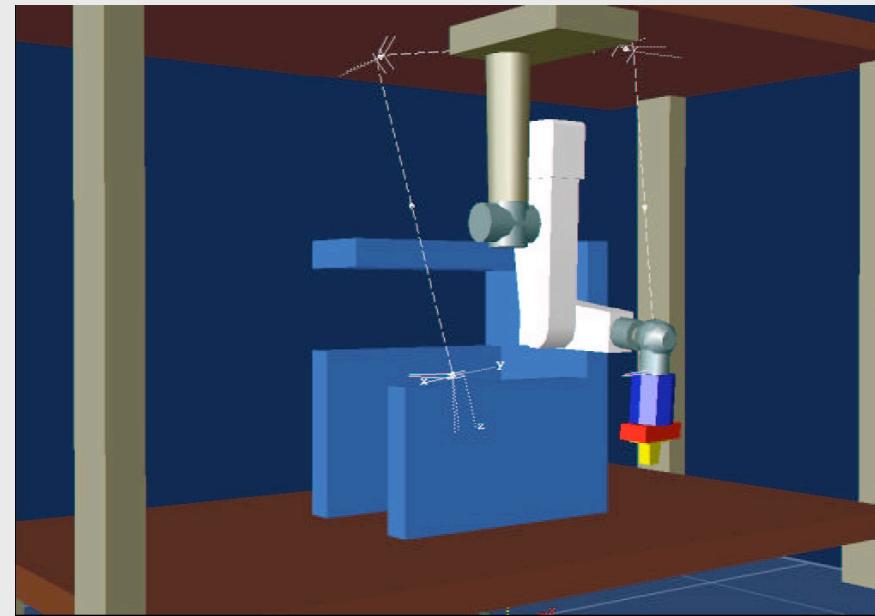
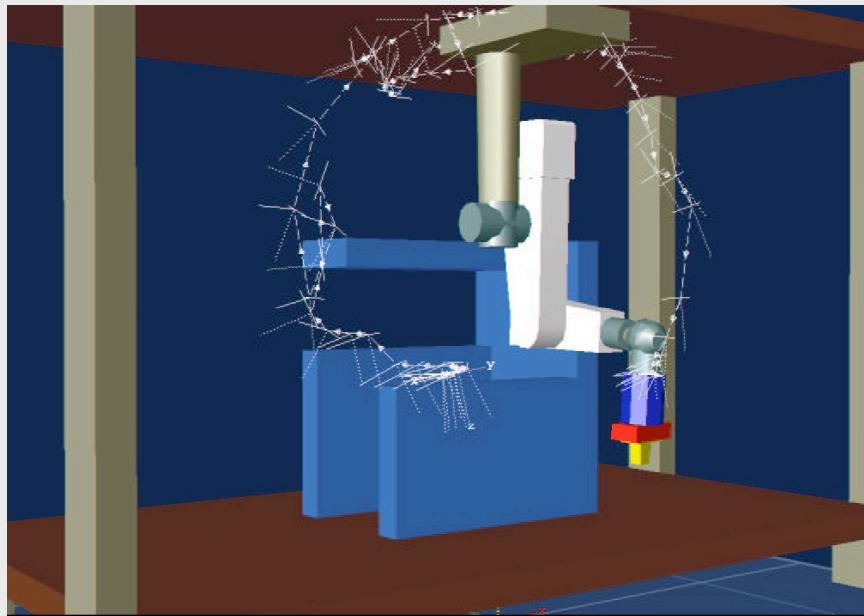
6-DOF-Example: Detour



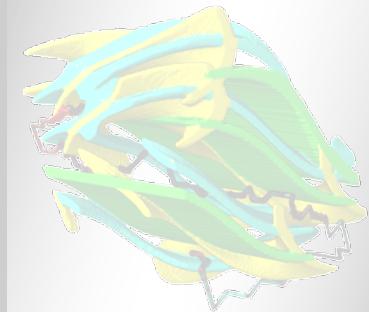
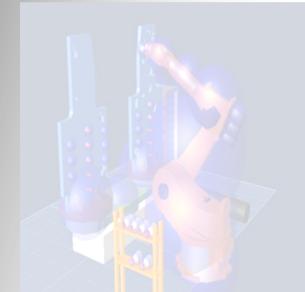
6-DOF-Example: Star



6-DOF-Example: Trap



Optimisation



Result of Smoothing

Positive :-)

- ▶ movements are combined as much as possible
- ▶ path is shortest around obstacles (tightened like a rubber band)

“Not so good :-(“

- ▶ robot must make a full stop at every path point.
 - ▶ Only then the path can guaranteed to be collision-free.



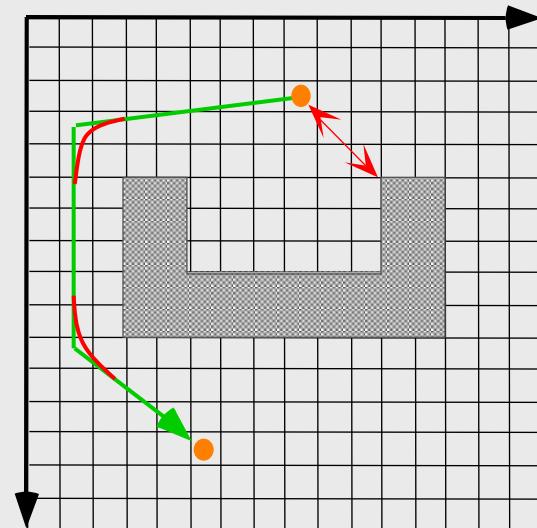
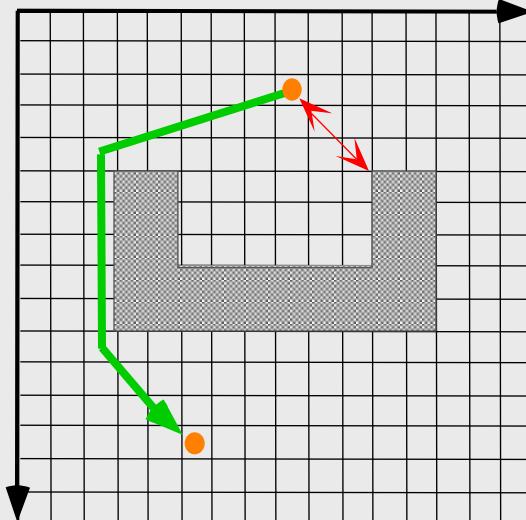
Optimisation

GOAL:

- ▶ Faster execution, without stopping
- ▶ more harmonious movement

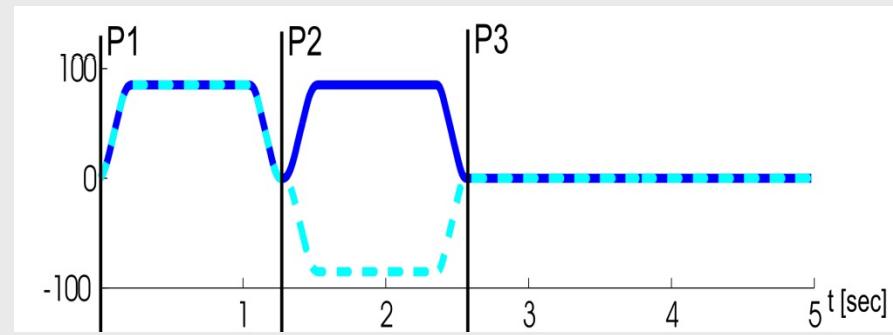
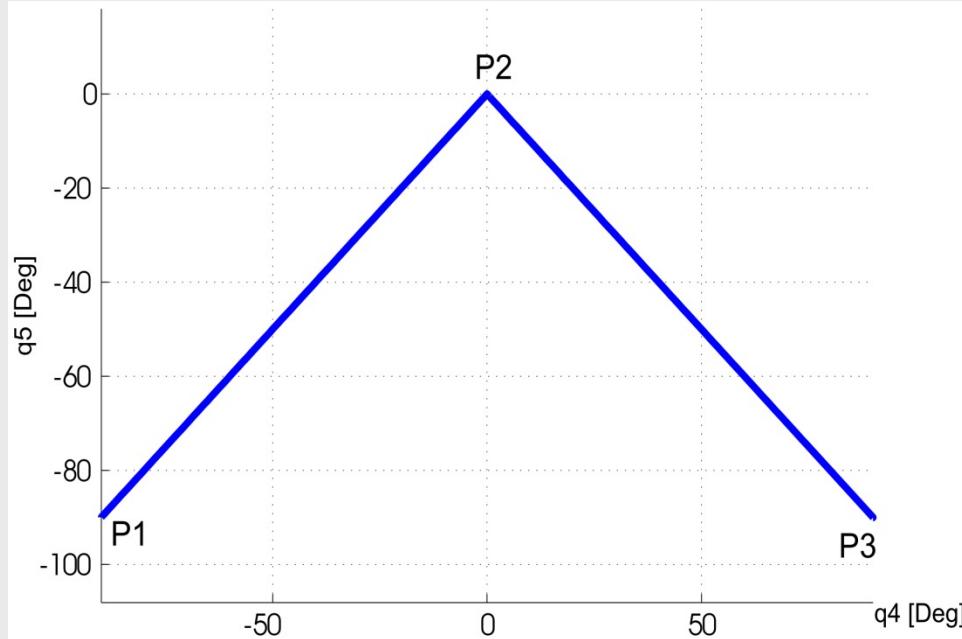
How?

- ▶ using PTP motion command with rounding



Simple PTP-Command

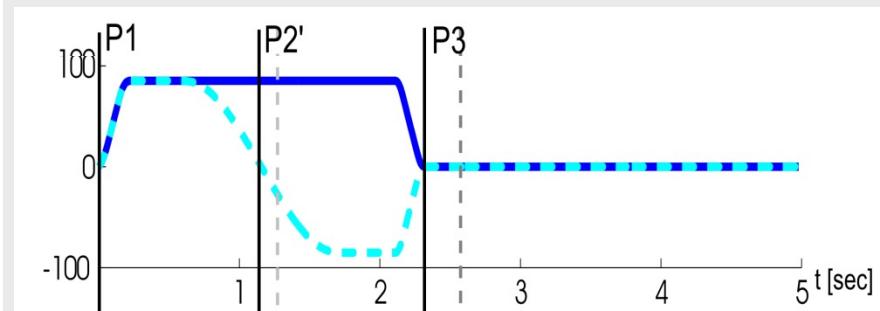
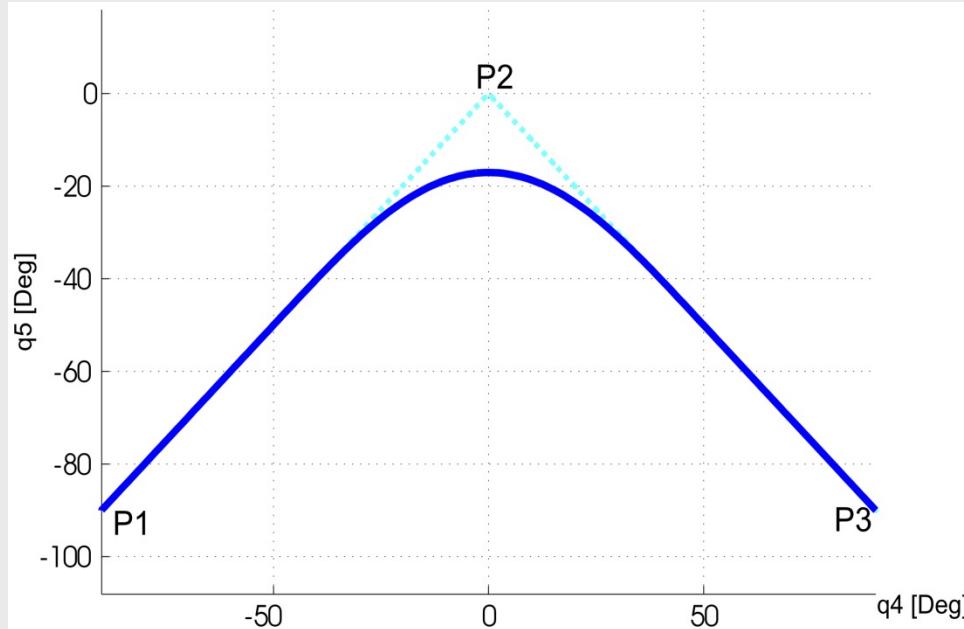
- Movement stops at every point



```
PTP(A1 0, A2 90, A3 -90 , A4 0, A5 -90, A6 -90) /* Start point */  
PTP_REL(A5 90, A6 90) (Anmerkung: A5=q4)  
PTP_REL(A5 90, A6 -90)
```

PTP-Command using Rounding

- Movement doesn't stop at point P2 (Rounding)



```
PTP(A1 0, A2 90, A3 -90 , A4 0, A5 -90, A6 -90) /* Start point */  
PTP_REL(A5 90, A6 90) C_PTP /* Factor is given by e.g. APO.CPTP=0.1 */  
PTP_REL(A5 90, A6 -90)
```

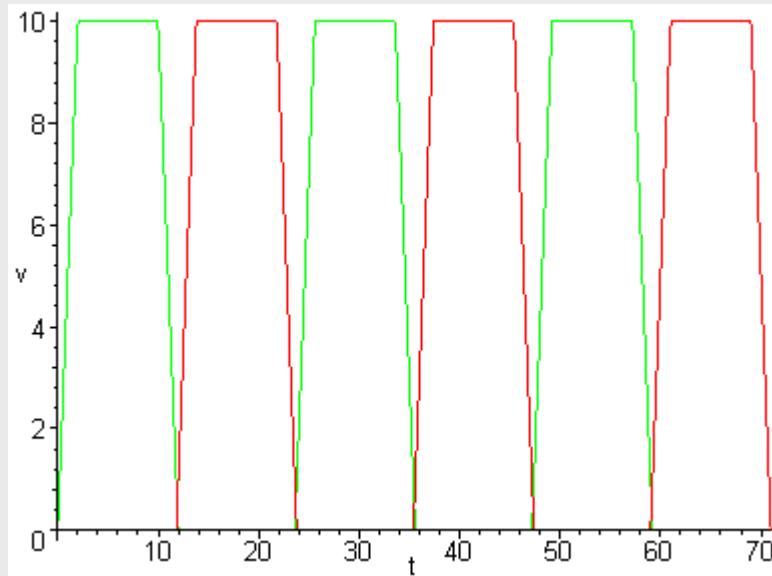
Rounding

Next movement begins, before the actual one is finished.

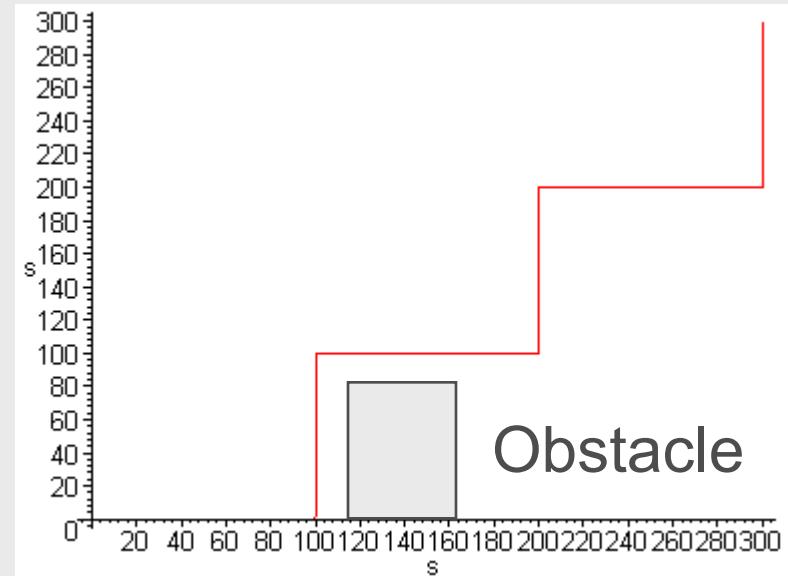
Rounding factor r (in percent) determines how much the two movements are overlapping.

Rounding examples

$$r = 0\%$$



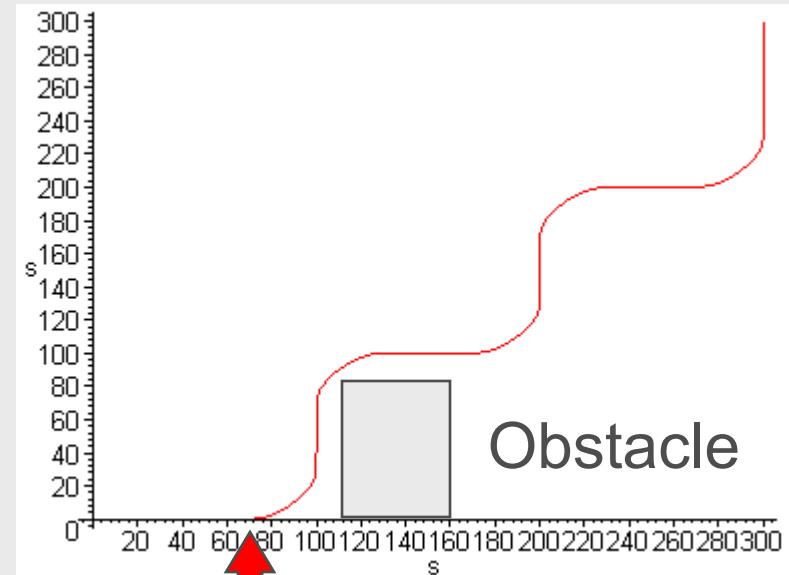
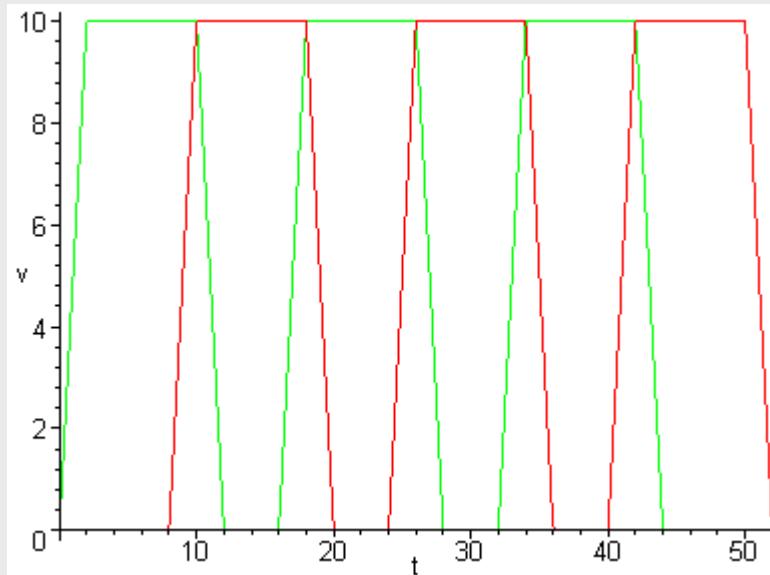
Velocity profiles



Movement in C-space

Rounding examples

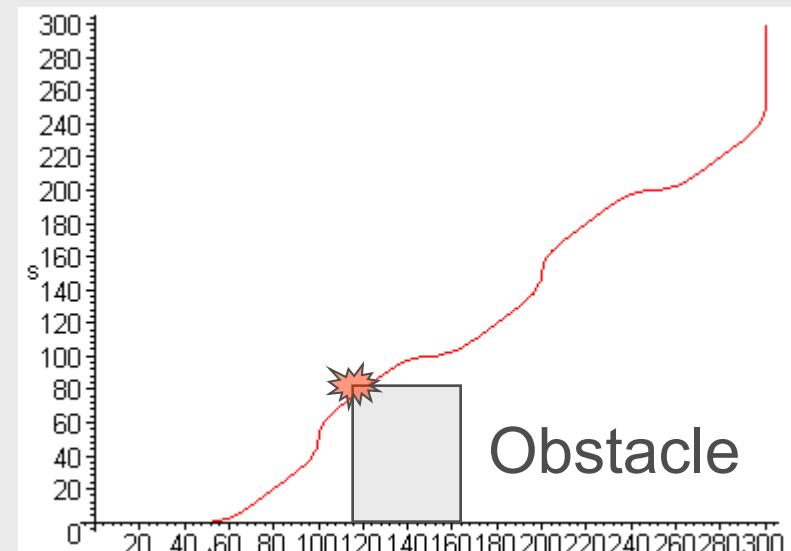
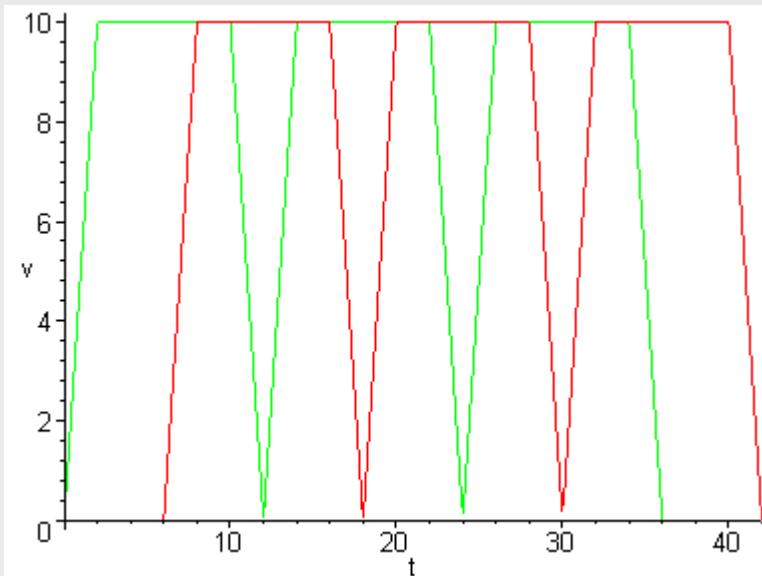
$$r = 30\%$$



Start of rounding

Rounding examples

$$r = 50\%$$



Start of rounding

Rounding: Task

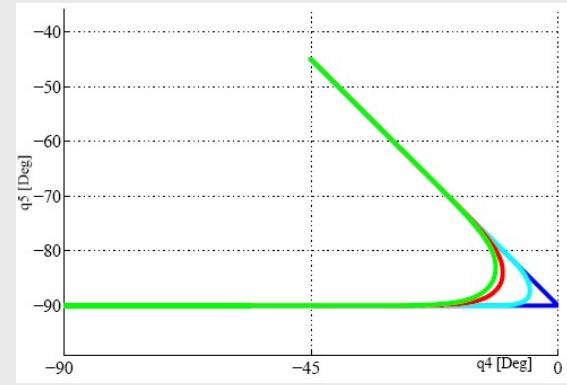
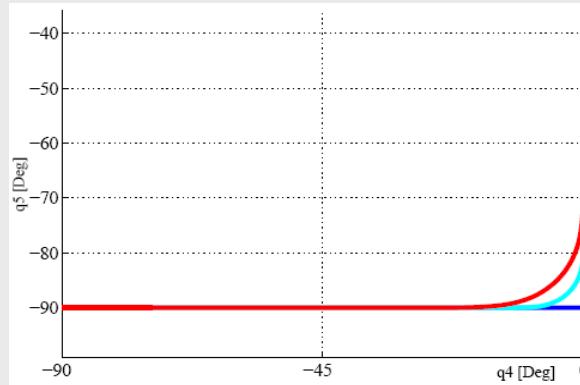
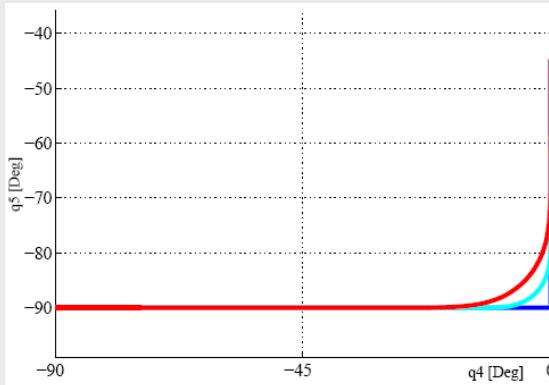
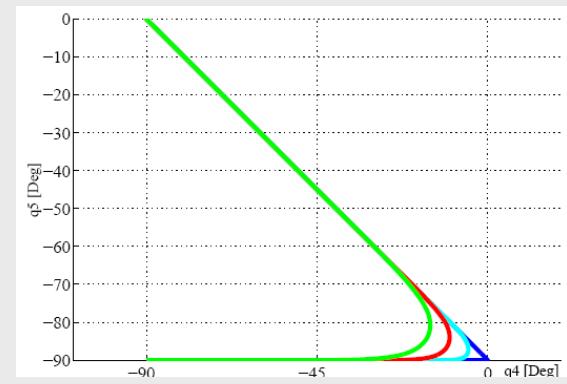
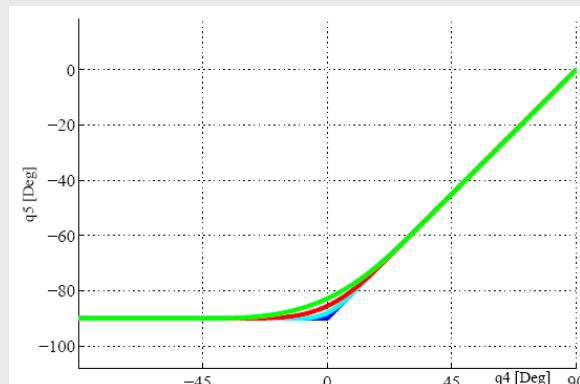
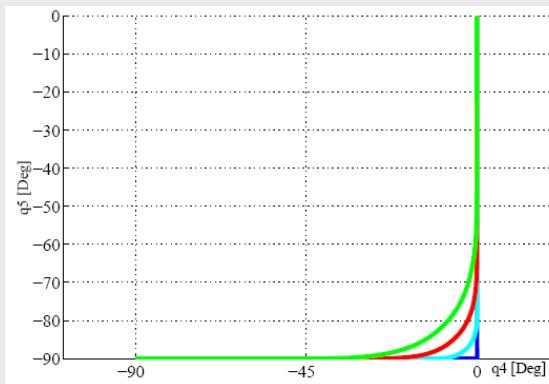
Modify path in such a way, that

in spite of using rounded movements

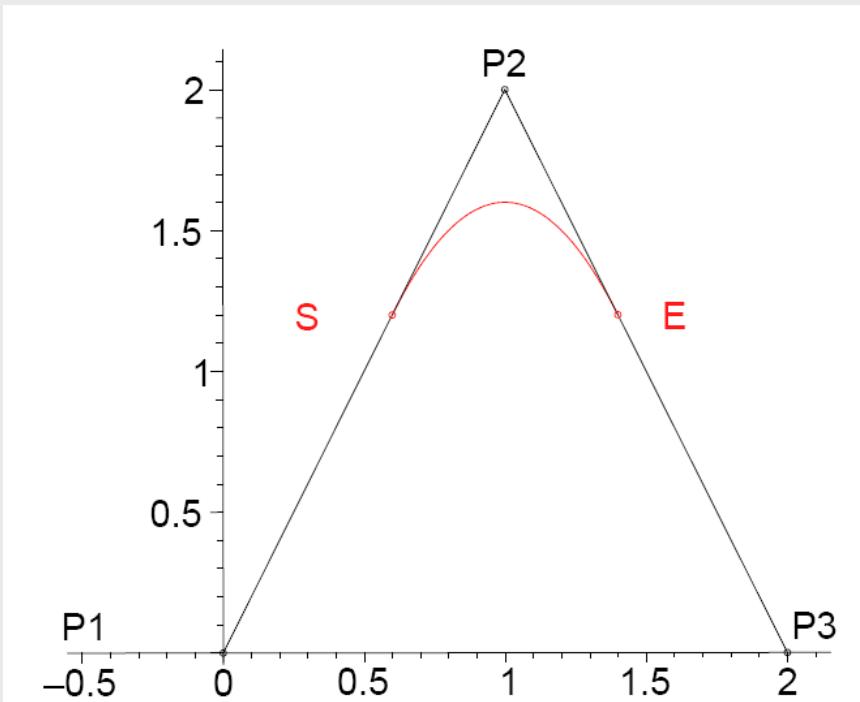
collisions are avoided

Rounding examples

- ▶ Using different paths and rounding factors

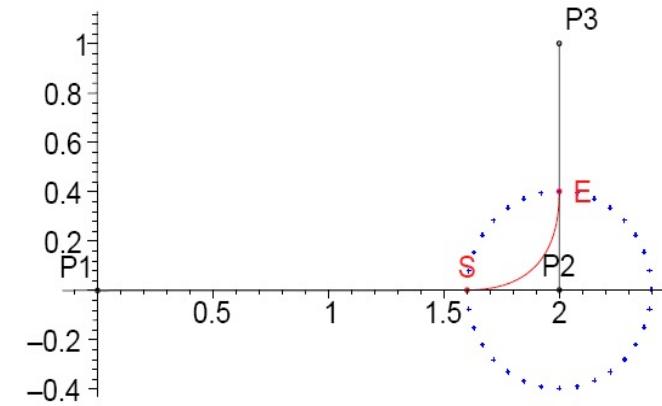
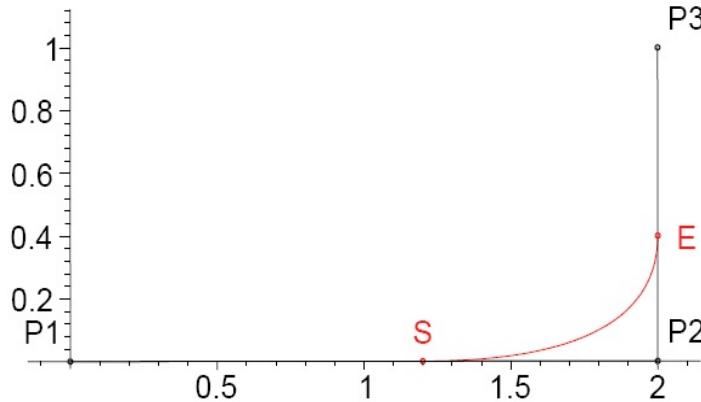


Model of rounding: first try

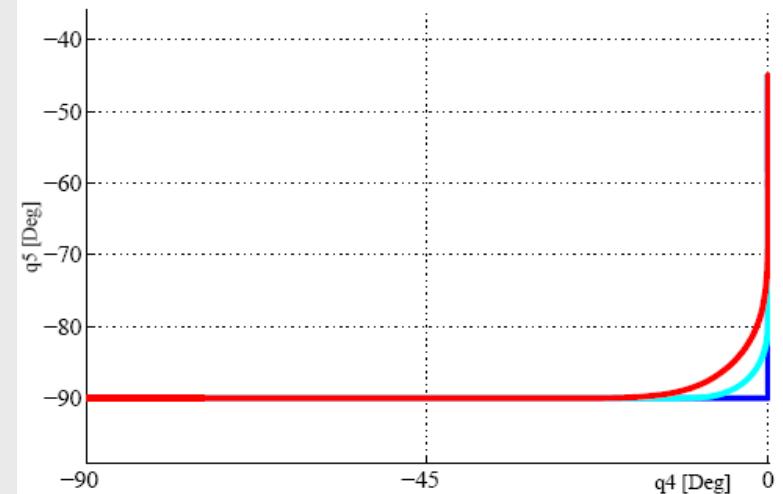


$$PA : \begin{cases} \text{Strecke : } \overrightarrow{P_1S} & \text{mit } S = r \cdot \overrightarrow{P_2P_1} + P_2 \\ \text{Parabel : } r \cdot \overrightarrow{P_2P_1} (t-1)^2 + r \cdot \overrightarrow{P_2P_3} t^2 + P_2 \\ \text{Strecke : } \overrightarrow{EP_3} & \text{mit } E = r \cdot \overrightarrow{P_2P_3} + P_2 \end{cases}$$

First try doesn't match 😞



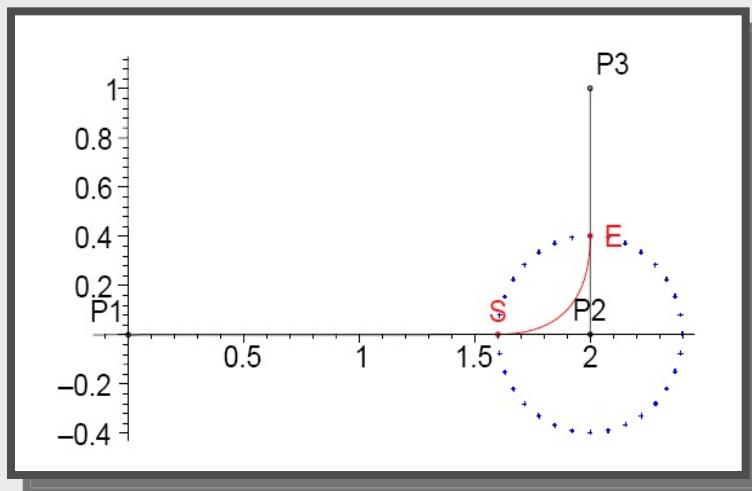
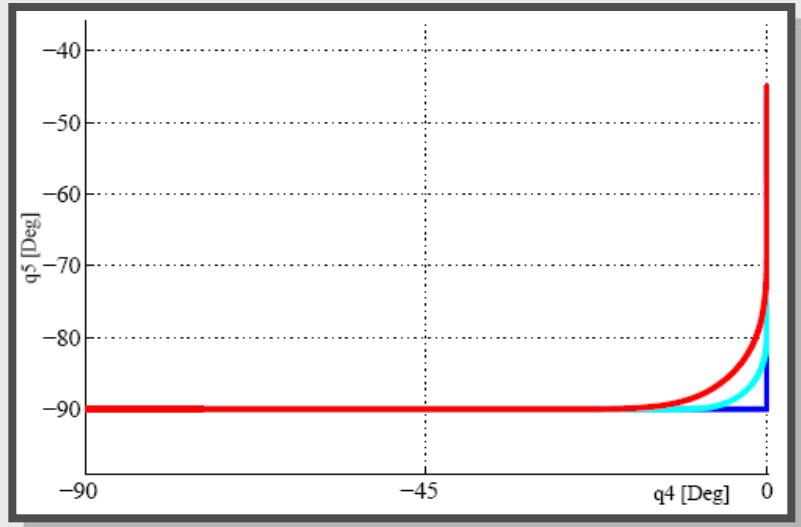
- ▶ On the left: „simple model“ doesn't represent reality if segments differ in lengths



Second model: take care of shortest segment

► Rounding „parabol“:

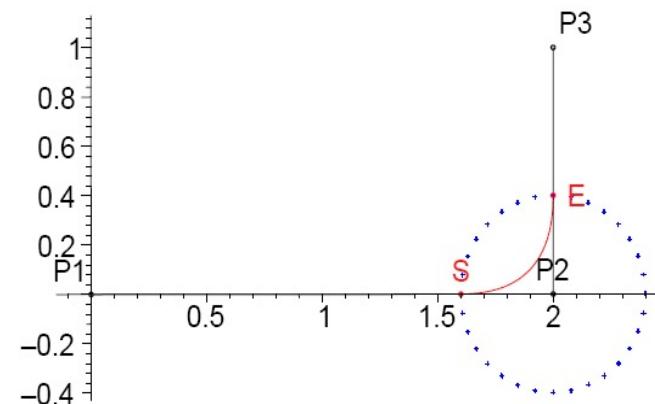
$$\vec{z}(t) = \begin{cases} r \cdot \overrightarrow{P_2P_1} (t-1)^2 + r \cdot \frac{|\overrightarrow{P_2P_1}|}{|\overrightarrow{P_2P_3}|} \overrightarrow{P_2P_3} t^2 + P_2 & \text{wenn } |\overrightarrow{P_2P_3}| \geq |\overrightarrow{P_2P_1}|, \\ r \cdot \frac{|\overrightarrow{P_2P_3}|}{|\overrightarrow{P_2P_1}|} \overrightarrow{P_2P_1} (t-1)^2 + r \cdot \overrightarrow{P_2P_3} t^2 + P_2 & \text{wenn } |\overrightarrow{P_2P_1}| \geq |\overrightarrow{P_2P_3}| \end{cases}$$



Second model: take care of shortest segment

► Rounding „parabol“:

$$\vec{z}(t) = \begin{cases} r \cdot \overrightarrow{P_2P_1} (t-1)^2 + r \cdot \frac{|\overrightarrow{P_2P_1}|}{|\overrightarrow{P_2P_3}|} \overrightarrow{P_2P_3} \\ r \cdot \frac{|\overrightarrow{P_2P_3}|}{|\overrightarrow{P_2P_1}|} \overrightarrow{P_2P_1} (t-1)^2 + r \cdot \overrightarrow{P_2P_3} \end{cases}$$

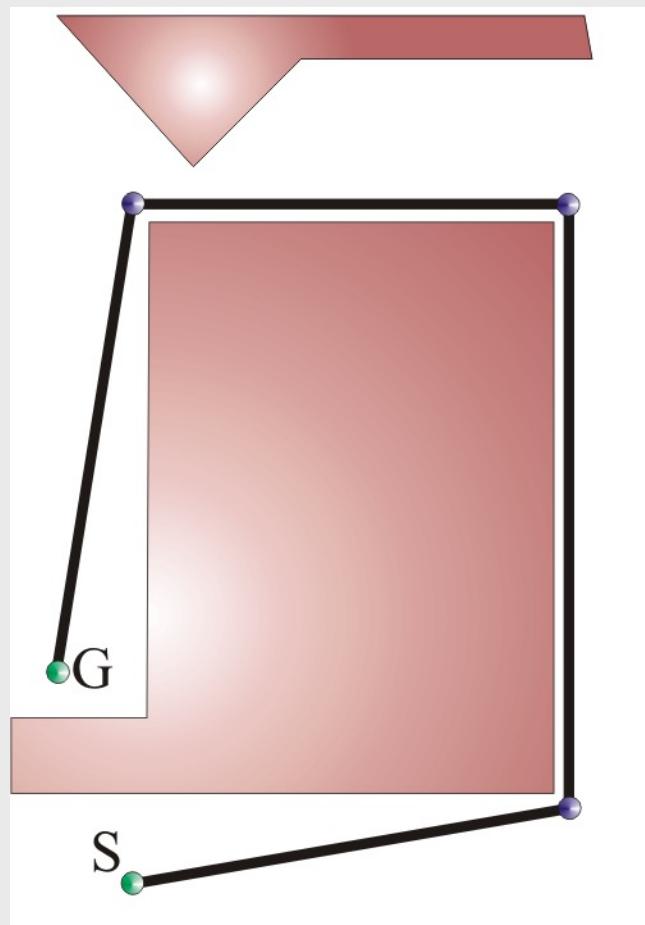


► Start and Endpoint:

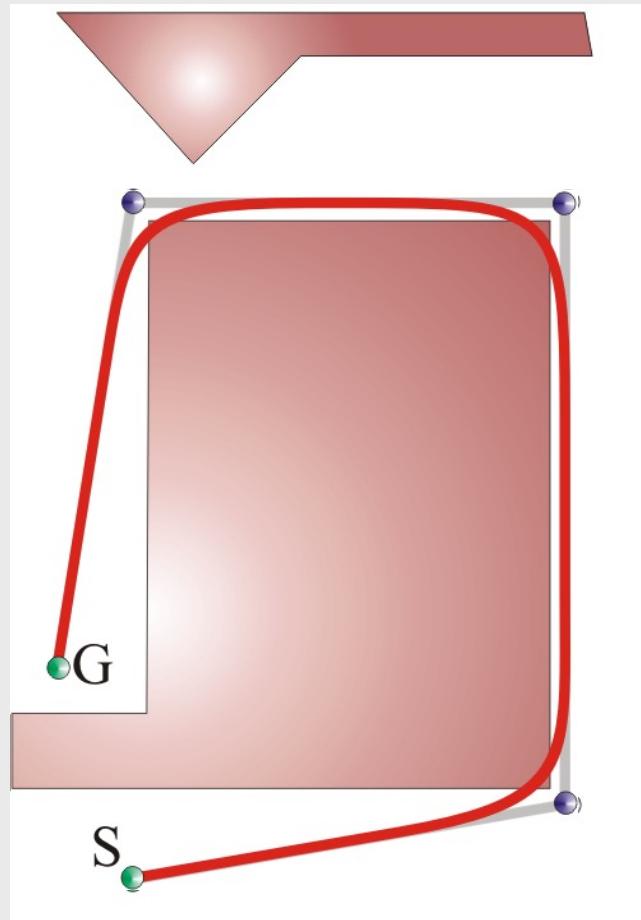
$$P_1 S \quad \text{mit} \quad \begin{cases} S = r \cdot \overrightarrow{P_2P_1} + P_2 & \text{wenn } |\overrightarrow{P_2P_1}| < |\overrightarrow{P_2P_3}| \\ S = r \cdot \frac{\overrightarrow{P_1P_2}|\overrightarrow{P_2P_3}|}{|\overrightarrow{P_1P_2}|} + P_2 & \text{wenn } |\overrightarrow{P_2P_1}| \geq |\overrightarrow{P_2P_3}| \end{cases}$$

$$EP_3 \quad \text{mit} \quad \begin{cases} E = r \cdot \overrightarrow{P_2P_3} + P_2 & \text{wenn } |\overrightarrow{P_2P_3}| < |\overrightarrow{P_1P_2}| \\ E = r \cdot \frac{\overrightarrow{P_2P_3}|\overrightarrow{P_1P_2}|}{|\overrightarrow{P_2P_3}|} + P_2 & \text{wenn } |\overrightarrow{P_2P_3}| \geq |\overrightarrow{P_1P_2}| \end{cases}.$$

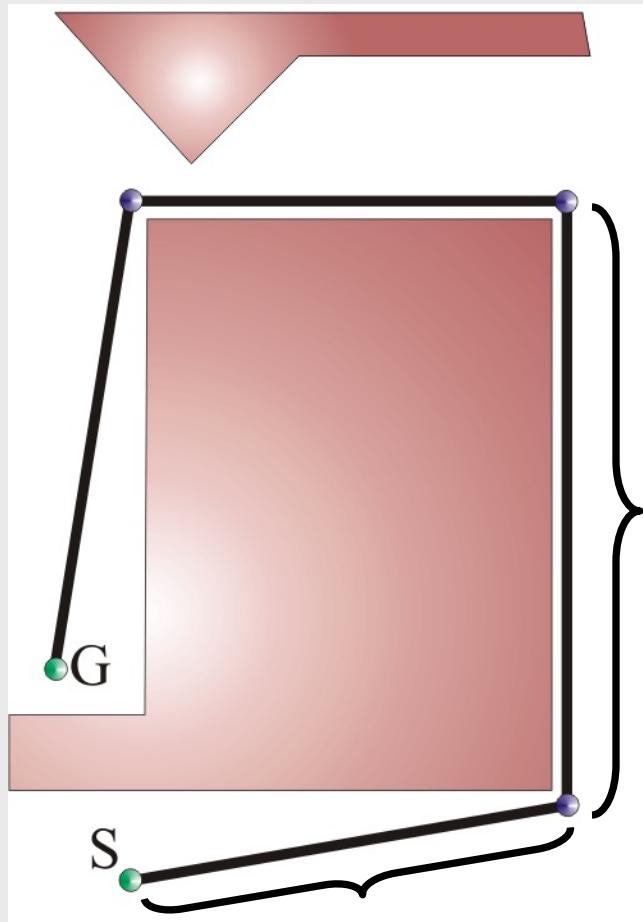
Situation after smoothing



Rounded movement collides

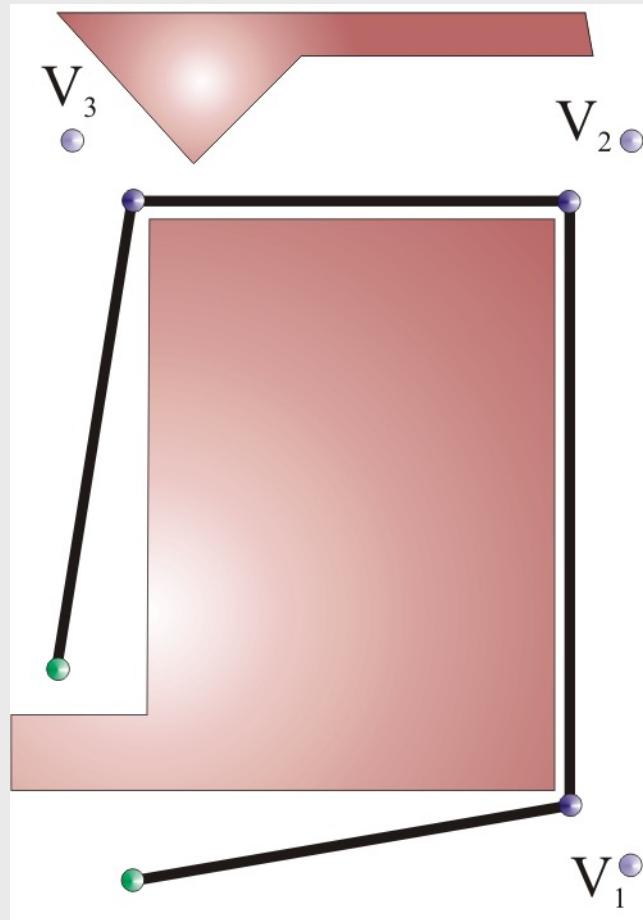


Situation after smoothing

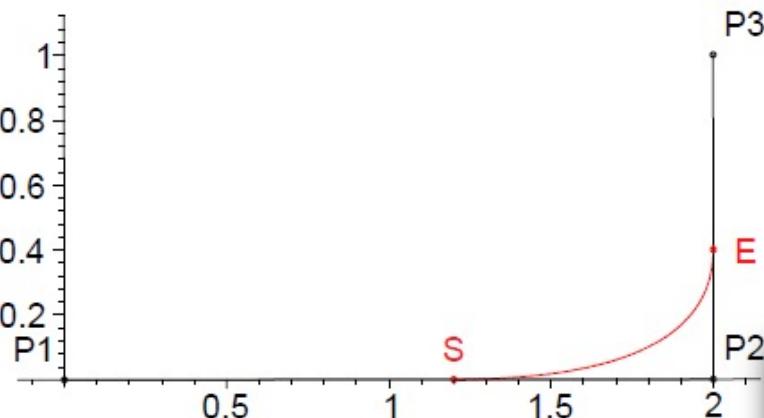


- ▶ Model of rounding Length of Segment
- ▶ Goal: hugest rounding factor as possible

Generation of virtual way points



„Inverse rounding“: first model (to get idea)

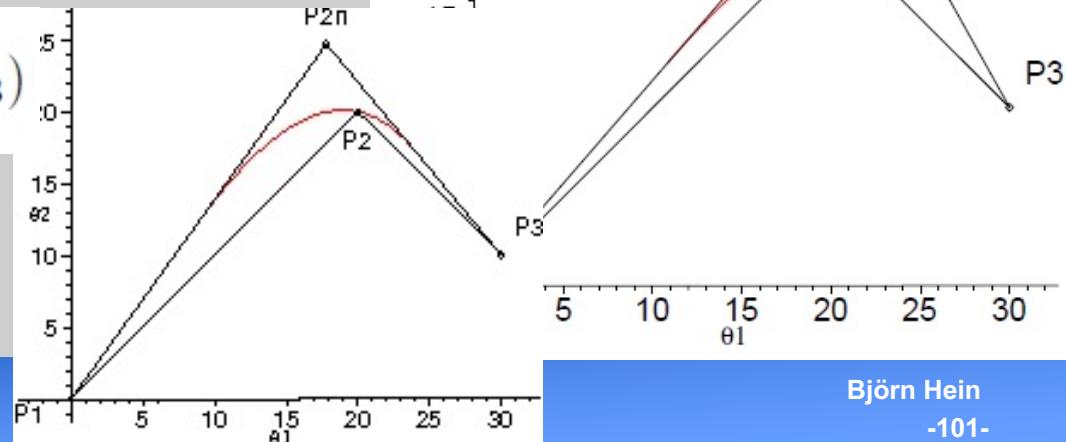


$$\vec{z}(t) = r \cdot \overrightarrow{P_2 P_1} (t-1)^2 + r \cdot \overrightarrow{P_2 P_3} t^2 + P_2,$$

$$\vec{z}_{neu}(0.5) = P_2$$

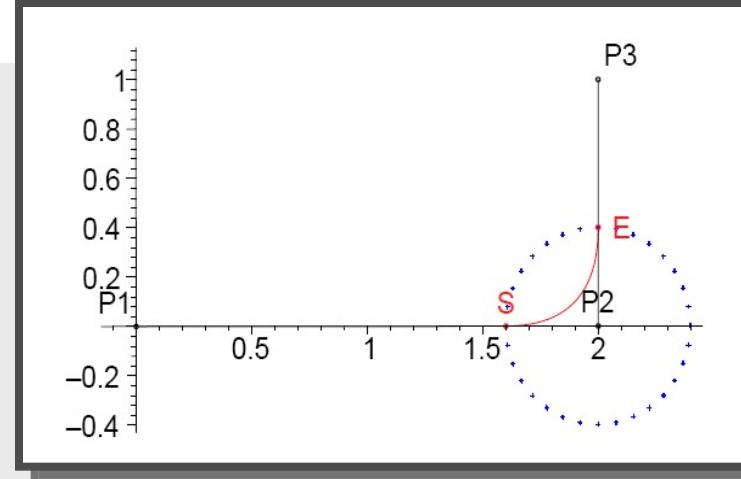
$$r \cdot \overrightarrow{P_{2n} P_1} \cdot 0.25 + r \cdot \overrightarrow{P_{2n} P_3} \cdot 0.25 + P_{2n} = P_2$$

$$P_{2n} = \frac{1}{2(r-2)}(rP_1 - 4P_2 + rP_3)$$



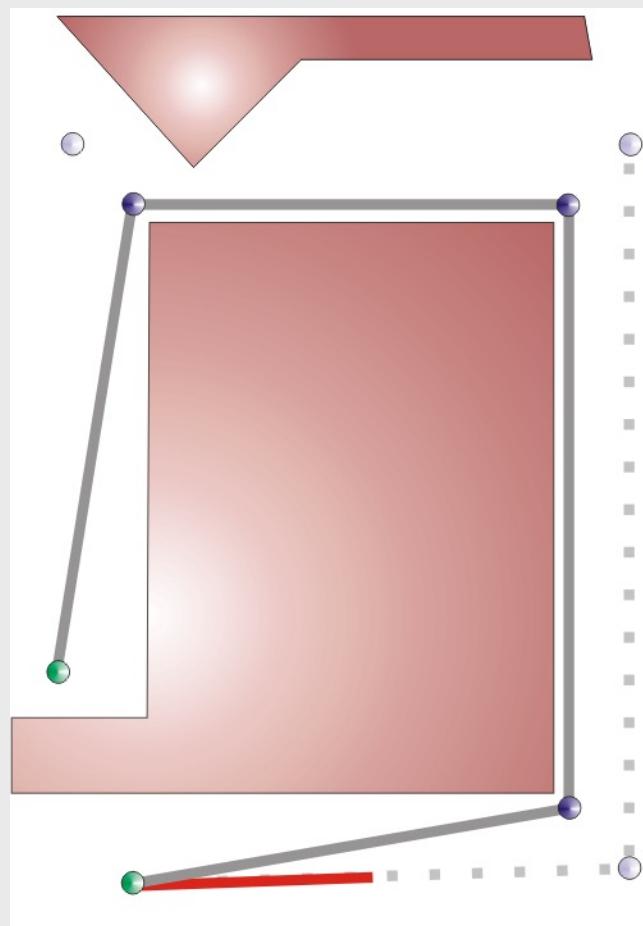
„inverse rounding“: second model

$$r \cdot \overrightarrow{P_2 P_1} (t - 1)^2 + r \cdot \frac{|\overrightarrow{P_2 P_1}|}{|\overrightarrow{P_2 P_3}|} \overrightarrow{P_2 P_3} t^2 + P_2 = P_3$$

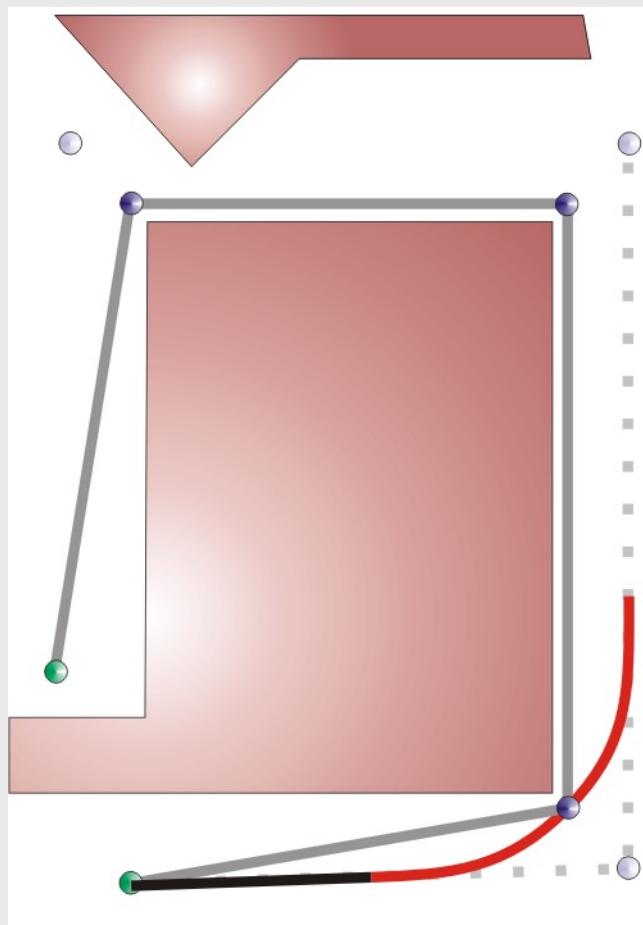


- ▶ Solution only solvable using numerical algorithms

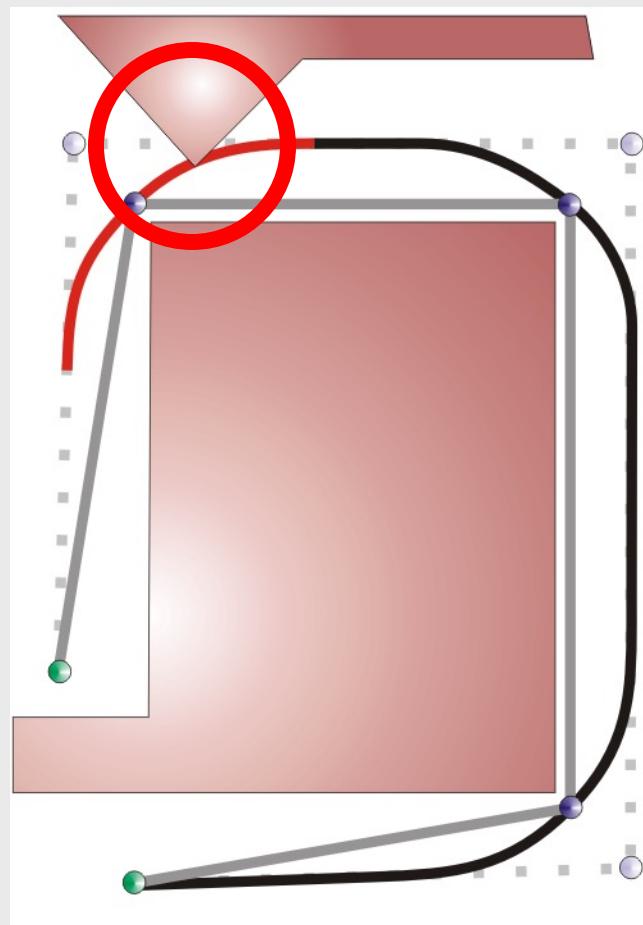
Collision testing of modified movement



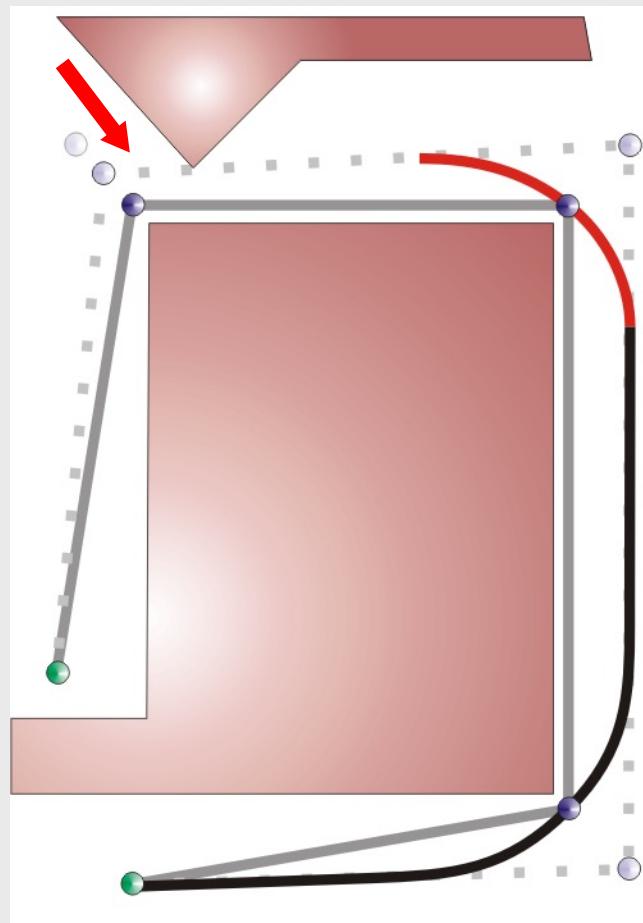
Collision testing of modified movement



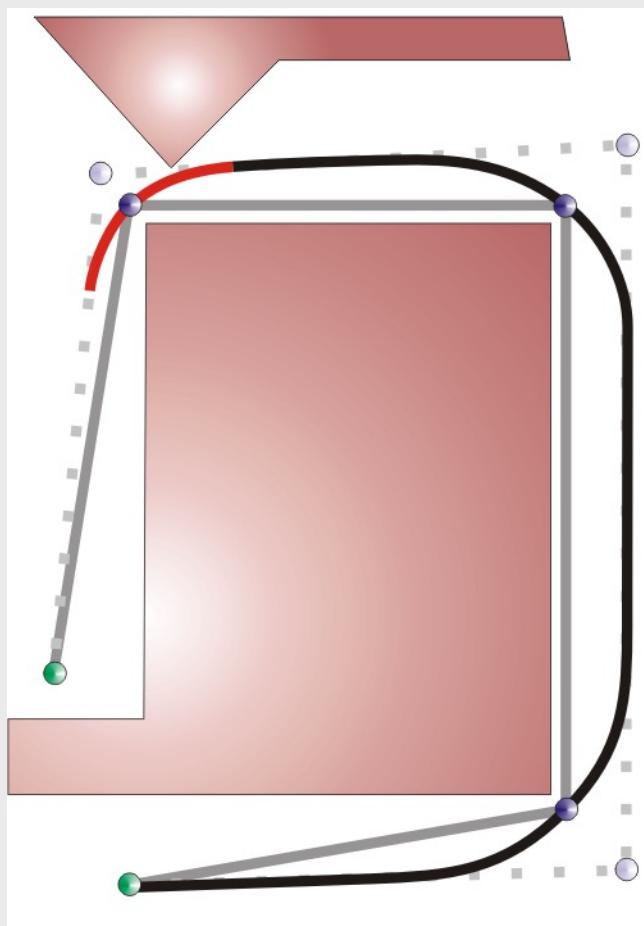
Collision testing of modified movement



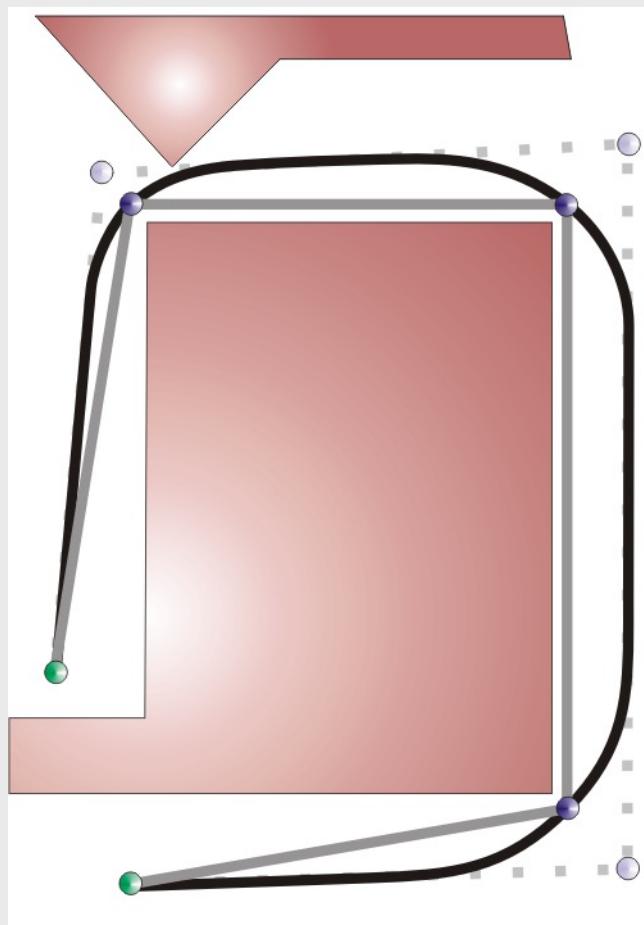
Collision testing of modified movement



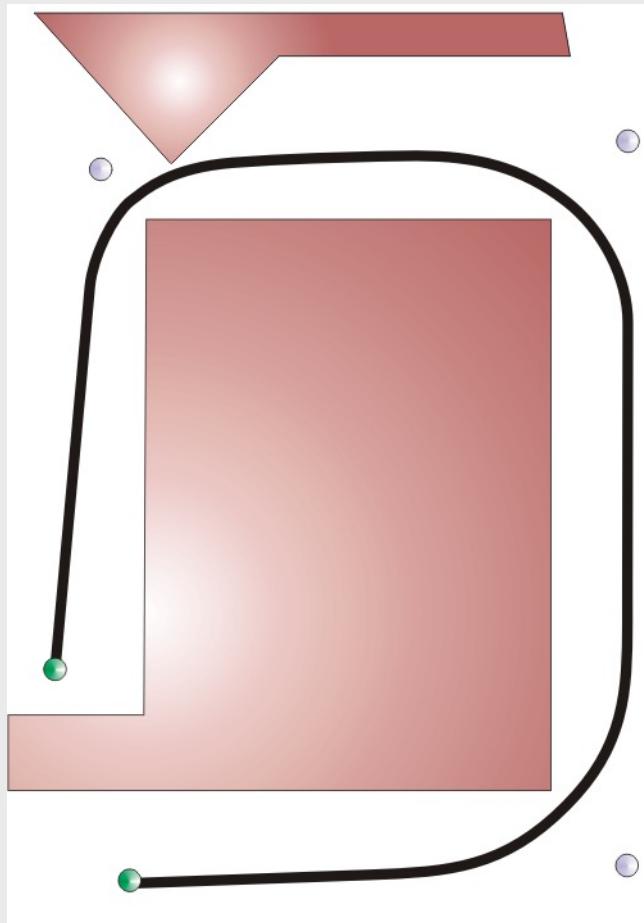
Collision testing of modified movement



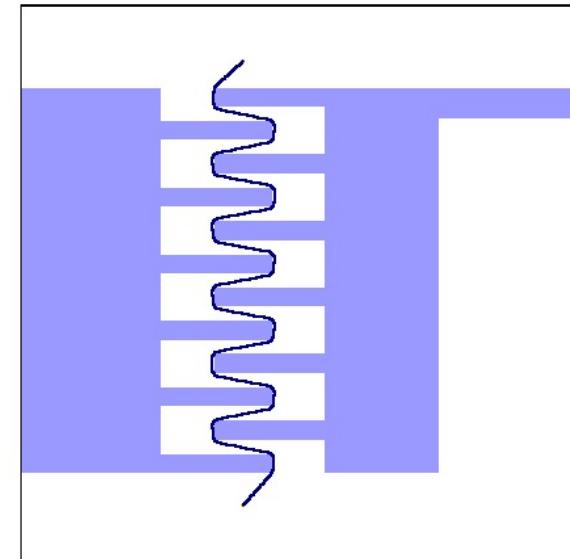
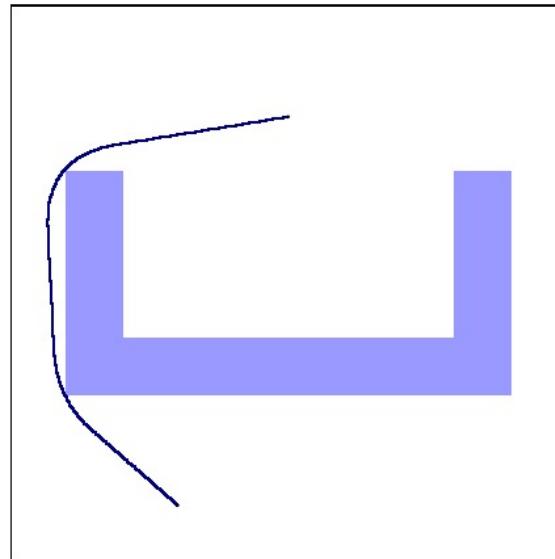
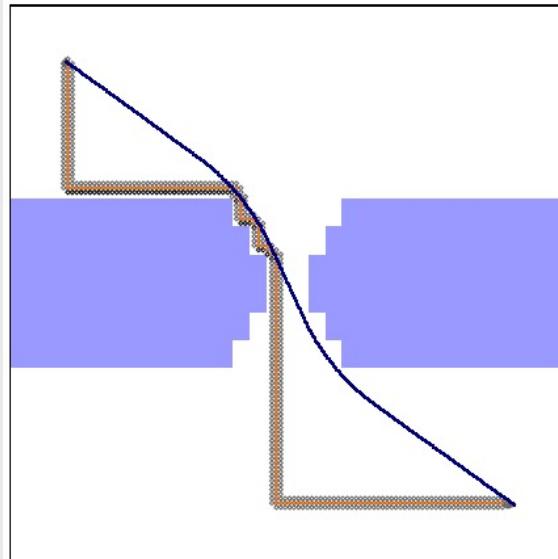
Collision testing of modified movement



Result



Example: 2D

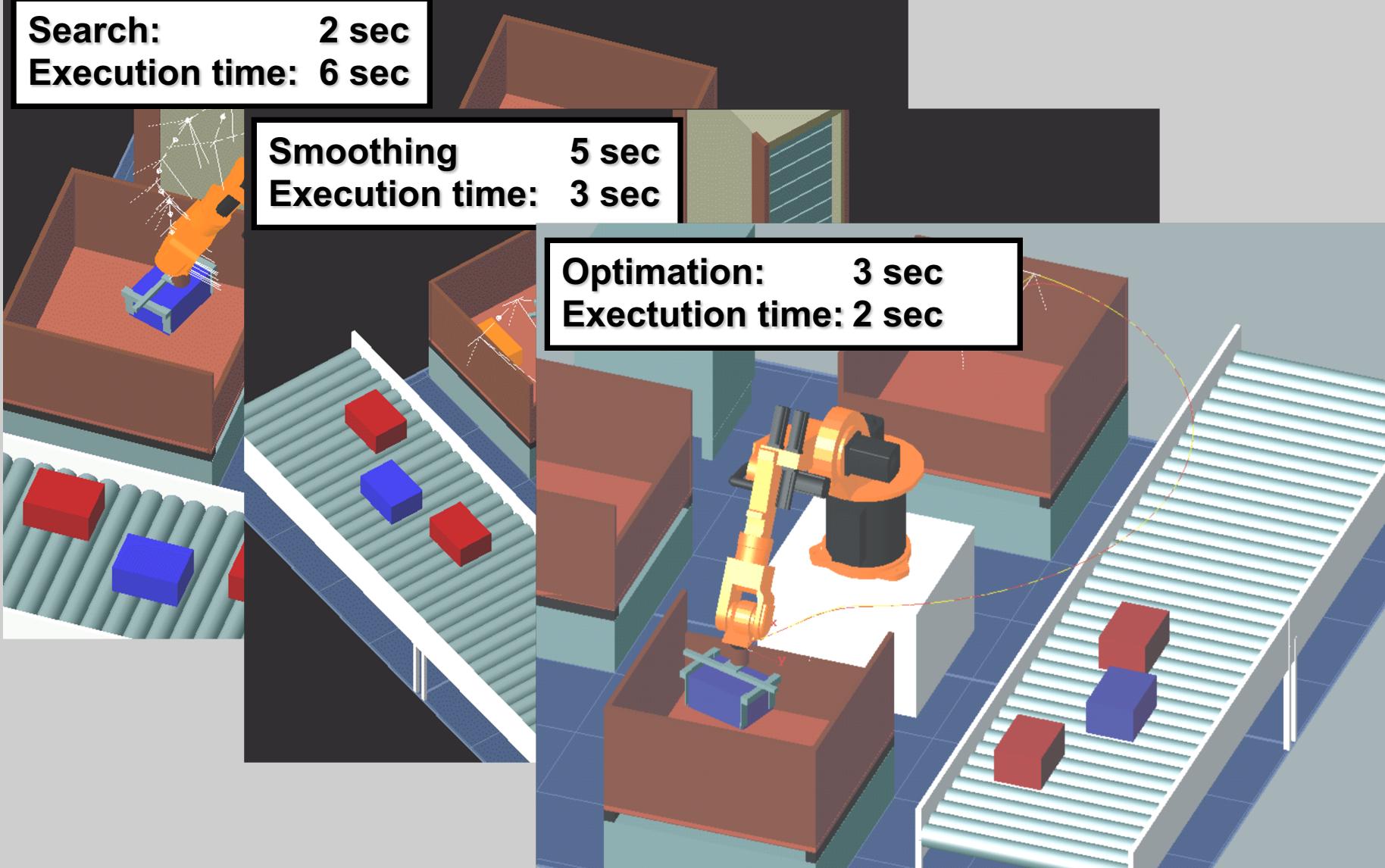


Example: SORT

Search: 2 sec
Execution time: 6 sec

Smoothing 5 sec
Execution time: 3 sec

Optimation: 3 sec
Execution time: 2 sec

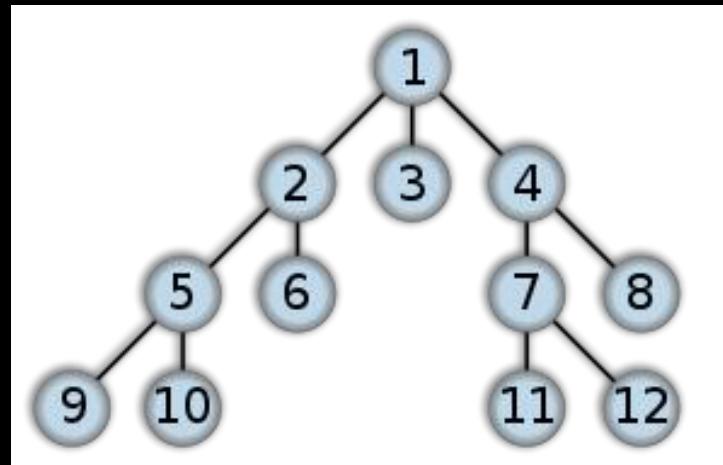


Sampling-based Planning 1

A lot of Material from Howie Choset, Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, J. Kuffner, D. Hsu

Previously...

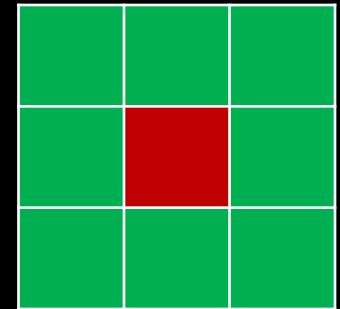
- We learned about discrete motion planning



- Discrete planning is best suited for
 - Low-dimensional motion planning problems
 - Problems where the control set can be easily discretized
- What if we need to plan in **high-dimensional** spaces?

Discrete Planning: The problem

- Discrete search run-time and memory requirements are very sensitive to branching factor (number of successors)
- Number of successors depend on dimension
- For a 3-dimensional 8-connected space, how many successors?
- For an n-dimensional 8-connected space, how many successors?



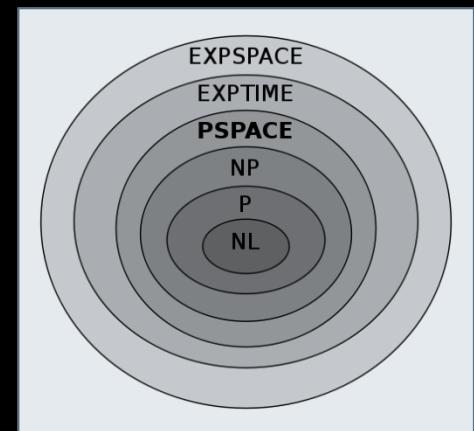
8-connected

The problem

- Need a path planning method that isn't so sensitive to dimensionality
- But:
 - Path planning is PSPACE-hard
[Reif 79, Hopcroft et al. 84, 86]
 - Complexity is exponential in dimension of the C-space [Canny 86]
- What if we weaken *completeness* and *optimality* requirements?



Real robots can have 20+ DOF!



Weakening requirements

Ideal	<u>Practical in High Dimensions</u>
Complete	Probabilistically Complete
Optimal	Feasible
*More recent methods show <i>asymptotic</i> optimality	

- **Probabilistic completeness:** A path planner is probabilistically complete if, given a solvable problem, the probability that the planner solves the problem goes to 1 as time goes to infinity.
- Feasibility: Path obeys all constraints (usually obstacles).
- A feasible path can be optimized *locally* after it is found

Sampling-based planning

- Main idea: Instead of systematically-discretizing the C-space, take samples in the C-space and use them to construct a path



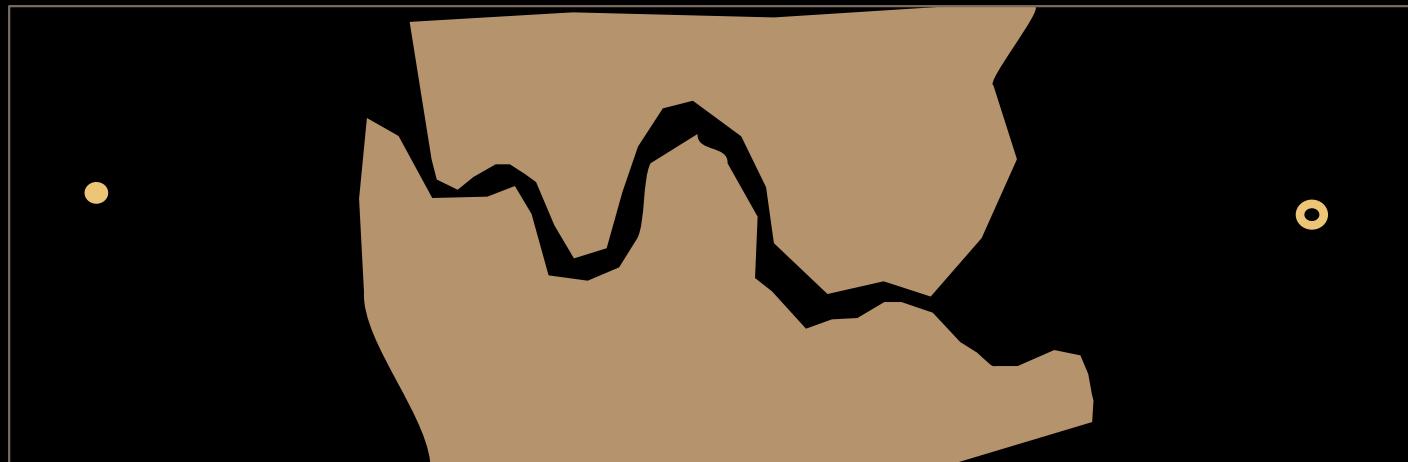
Sampling-based planning

Advantages

- Don't need to discretize C-space
- Don't need to explicitly represent C-space
- Easy to sample high-dimensional spaces

Disadvantages

- Probability of sampling an area depends on the area's size
 - Hard to sample *narrow passages*
- No strict completeness/optimality

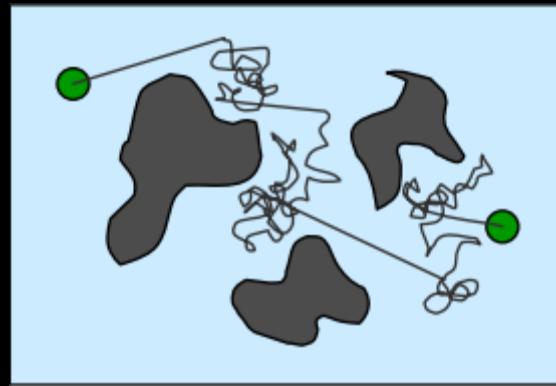


Outline

- RPP
- PRM
- Expansive Spaces

Randomized Path Planner (RPP)

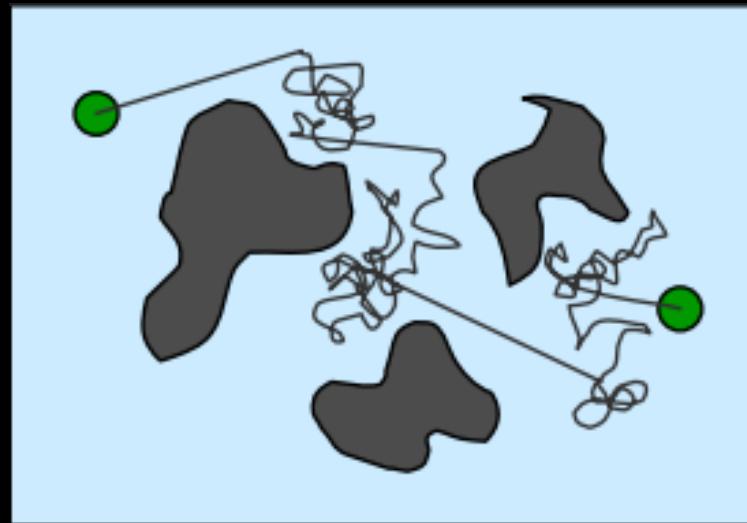
- Developed by Barraquand and Latombe in 1991 at Stanford



- Main idea: Follow a potential function, occasionally introduce random motion
 - Potential field biases search toward goal
 - Random motion avoids getting stuck in local minima

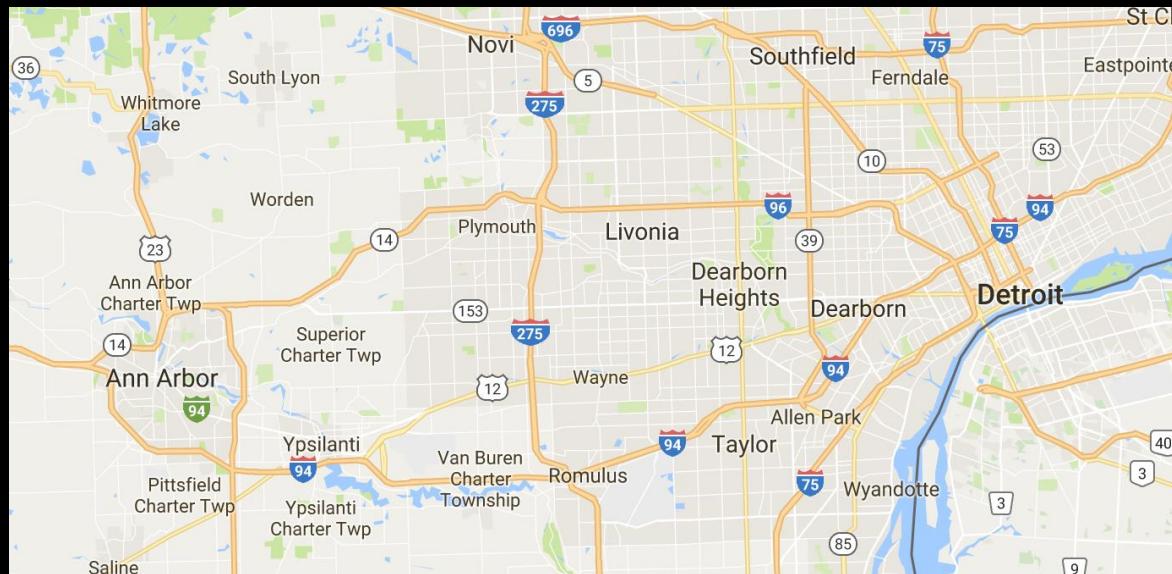
RPP

- Advantage: Doesn't get stuck in local minima
- Disadvantage: Parameters needed to
 - define potential field
 - decide when to apply random motion
 - how much random motion to apply



Probabilistic Roadmap (PRM)

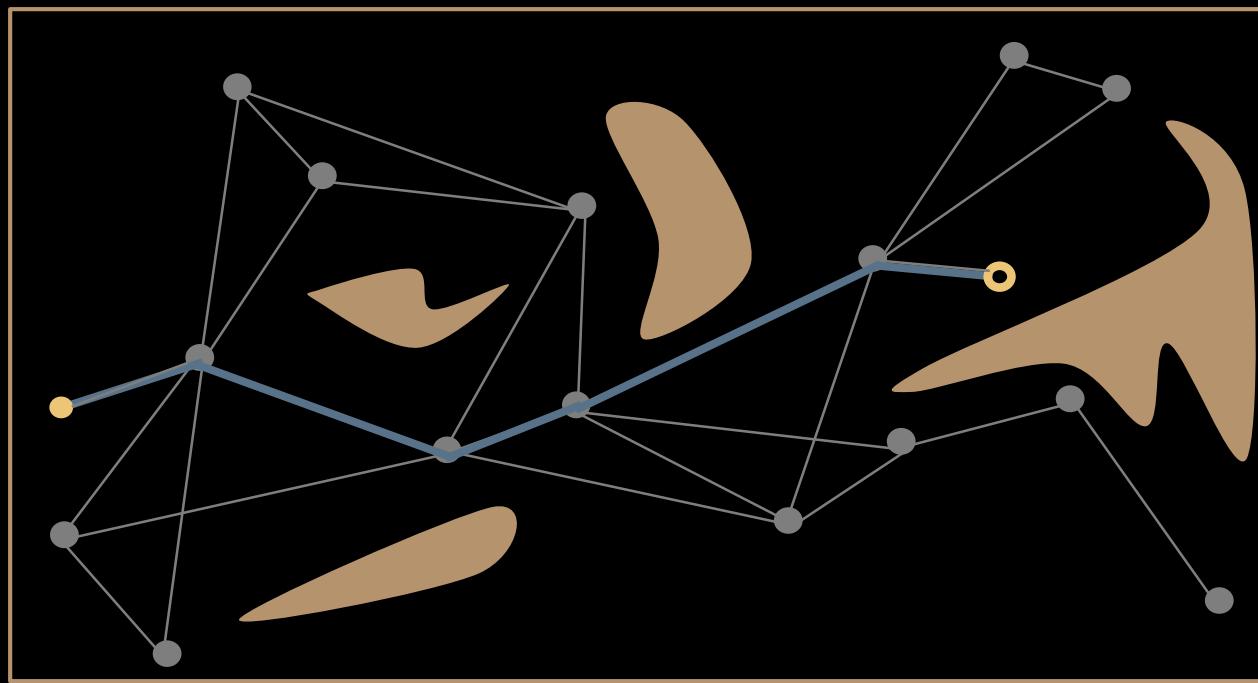
- Perhaps the most famous motion planning paper:
Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." IEEE Transactions on Robotics and Automation 12, no. 4, 1996.
- Main idea: Build a roadmap of the space from sampled points, search the roadmap to find a path
- Roadmap should capture the *connectivity* of the free space



Probabilistic Roadmap (PRM)

- Building a PRM: 2 phase process
- “Learning” Phase
 - Construction Step
 - Expansion Step
- Query Phase
 - Answer a given path planning query
- PRMs are known as *multi-query algorithms*, because roadmap can be re-used if environment and robot haven’t changed between queries.

PRM Example



“Learning” Phase

- Construction step: Build the roadmap by sampling random free configurations and connect them using a fast *local planner*
- Store these configurations as nodes in a graph
 - Note: In PRM literature, nodes are sometimes called “milestones”
- Edges of the graph are the paths between nodes found by the local planner

Construction Step

Start with an empty graph $G = (V, E)$

For $i = 1$ to MaxIterations

 Generate random configuration q ←

 If q is collision-free

 Add q to V

 Select k nearest nodes in V ←

 Attempt connection between each of these nodes and q using local planner

 If a connection is successful, add it as an edge in E

Construction Step

Start with an empty graph $G = (V, E)$

For $i = 1$ to MaxIterations

 Generate random configuration q 

 If q is collision-free

 Add q to V

 Select k nearest nodes in V 

 Attempt connection between each of these nodes and q using local planner

 If a connection is successful, add it as an edge in E

Sampling Collision-free Configurations

- Easiest and most common: uniform random sampling in C-space
 - Draw random value in allowable range for each DOF, combine into a vector
 - Place robot at the configuration and check collision
 - Repeat above until you get a collision-free configuration
 - AKA “Rejection Sampling”
- MANY ways to do this, many papers published, we will discuss more methods later

Construction Step

Start with an empty graph $G = (V, E)$

For $i = 1$ to MaxIterations

 Generate random configuration q ←

 If q is collision-free

 Add q to V

 Select k nearest nodes in V ←

 Attempt connection between each of these nodes and q using local planner

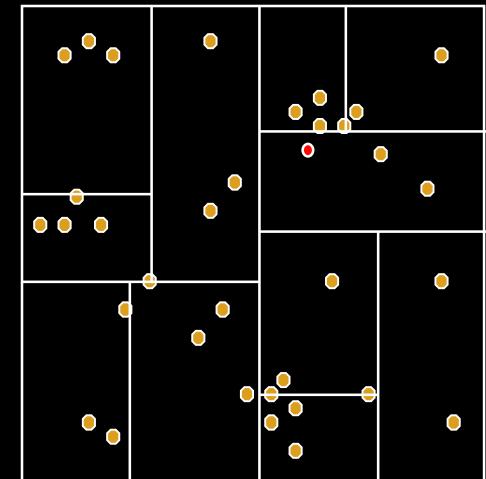
 If a connection is successful, add it as an edge in E

Finding Nearest Neighbors (NN)

- Need to decide a distance metric $D(q_1, q_2)$ to define “nearest”
- D should reflect likelihood of success of local planner connection (roughly)
 - If $D(q_1, q_2)$ is small, success should be likely
 - If $D(q_1, q_2)$ is large, success should be less likely
- By default, use Euclidian distance:
$$D(q_1, q_2) = \|q_1 - q_2\|$$
- Can weigh different dimensions of C-space differently
 - Often used to weigh translation vs. rotation

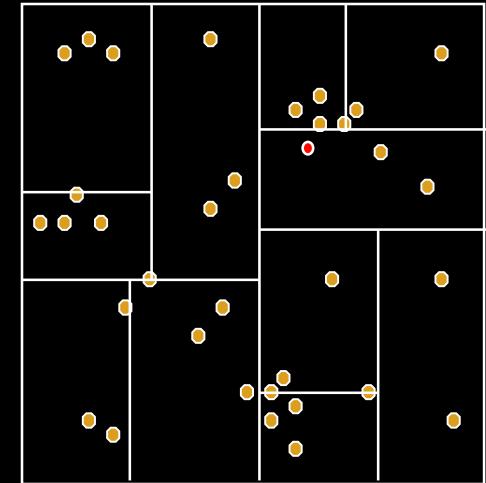
Finding Nearest Neighbors (NN)

- Two popular ways to do NN in PRM
 - Find k nearest neighbors (even if they are distant)
 - Find all nearest neighbors within a certain distance
- Naïve NN computation can be slow with 1000s of nodes, so use *kd-tree* to store nodes and do NN queries
 - A kd-tree is a data-structure that recursively divides the space into bins that contain points (like Oct-tree and Quad-tree)
 - NN then searches through bins (not individual points) to find nearest point

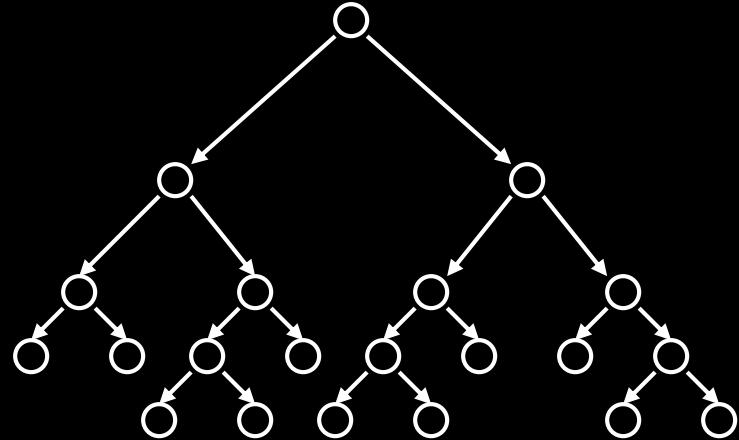
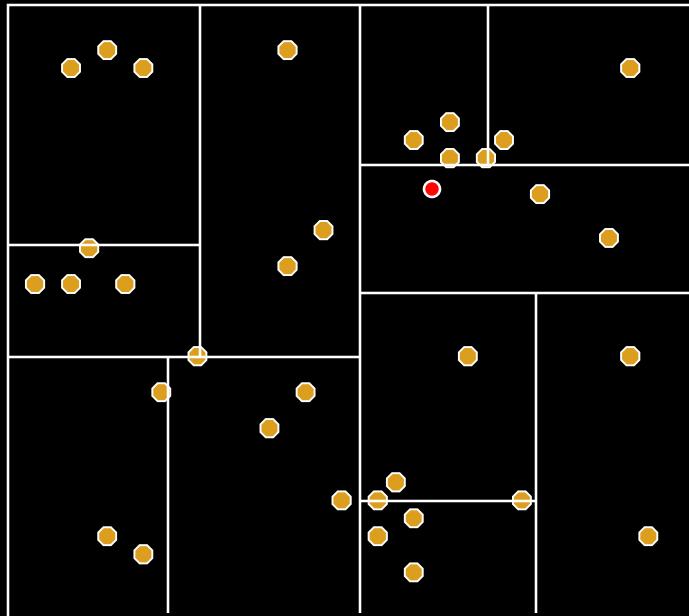


kd-trees (2D example)

- Data structure with one-dimensional splits in the data
- Algorithm
 - Choose x or y coordinate (alternate between them)
 - Choose the median of the coordinate for the points in this cell
 - this defines a horizontal or vertical line
 - can use other variants instead of median to select split line
 - Recurse on both sides until there is only one point left, which is stored as a leaf
- We get a binary tree
 - Size $O(n)$
 - Construction time $O(n \log n)$
 - Depth $O(\log n)$

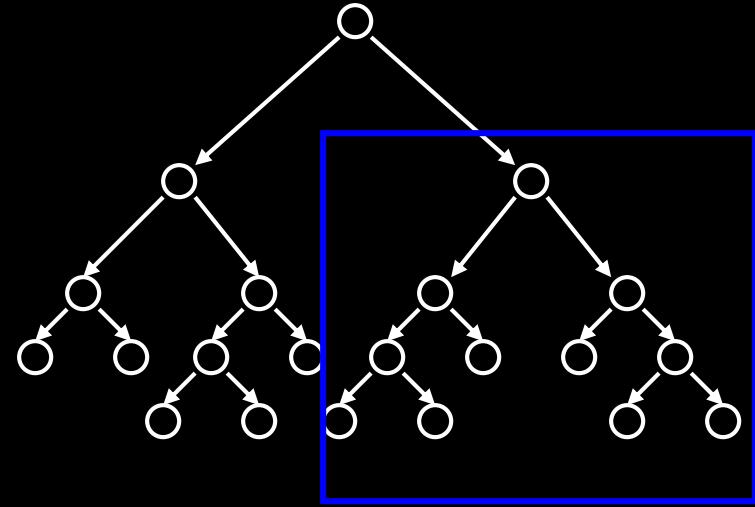
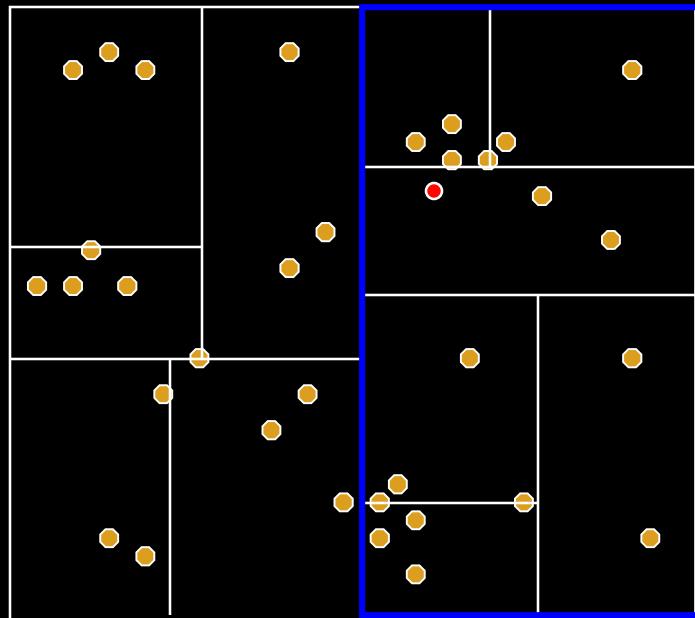


Nearest Neighbor with KD Trees



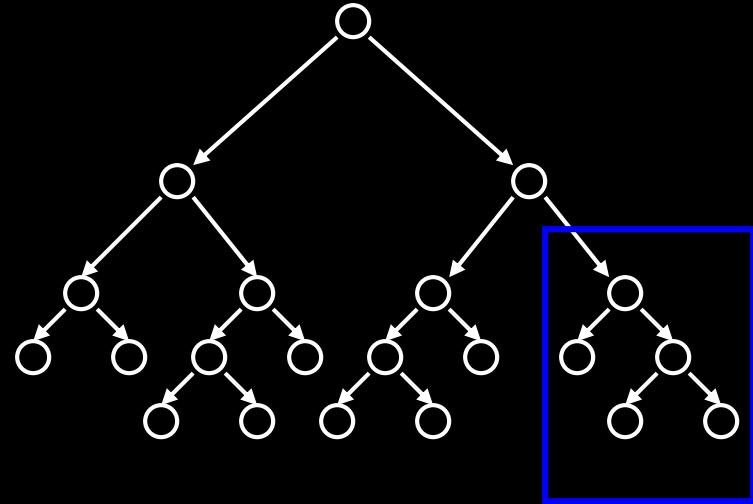
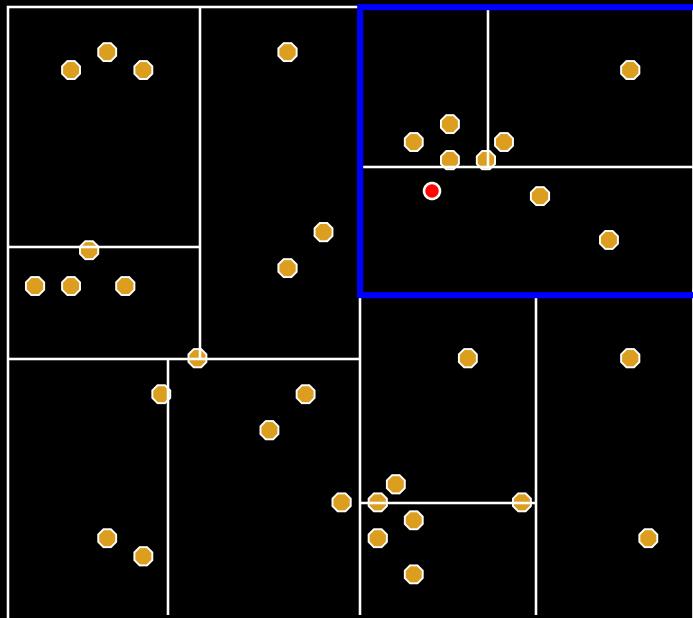
We traverse the tree looking for the nearest neighbor of the query point

Nearest Neighbor with KD Trees



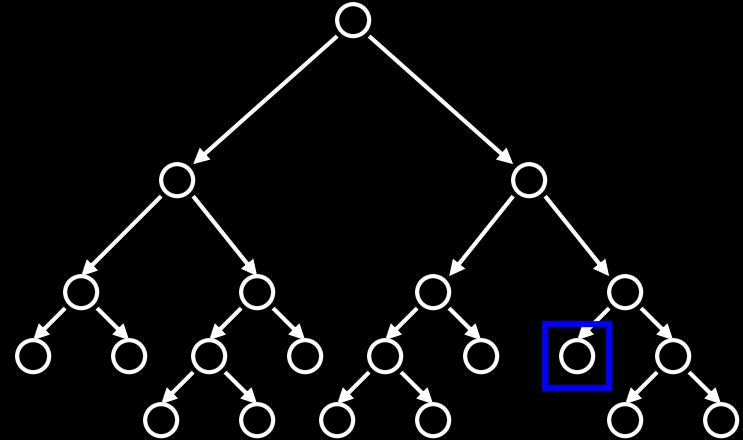
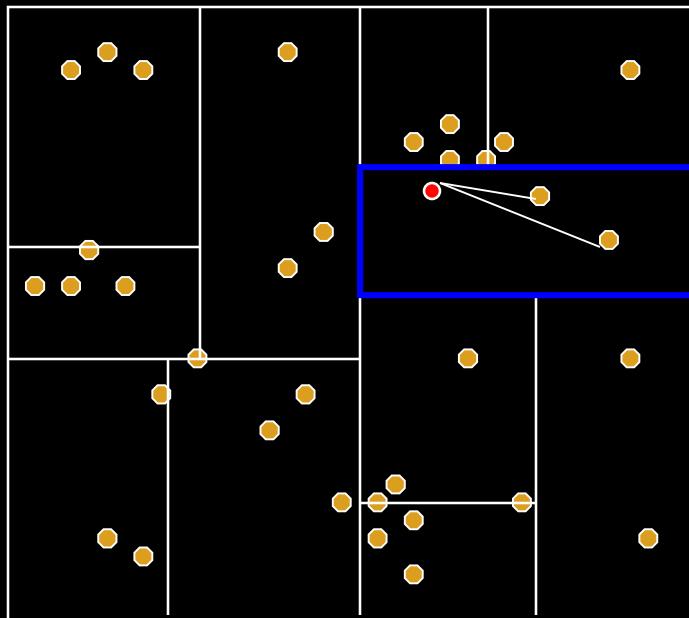
Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Nearest Neighbor with KD Trees



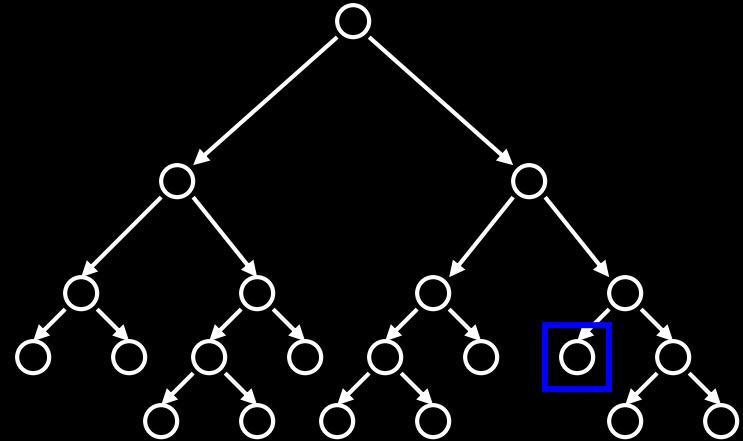
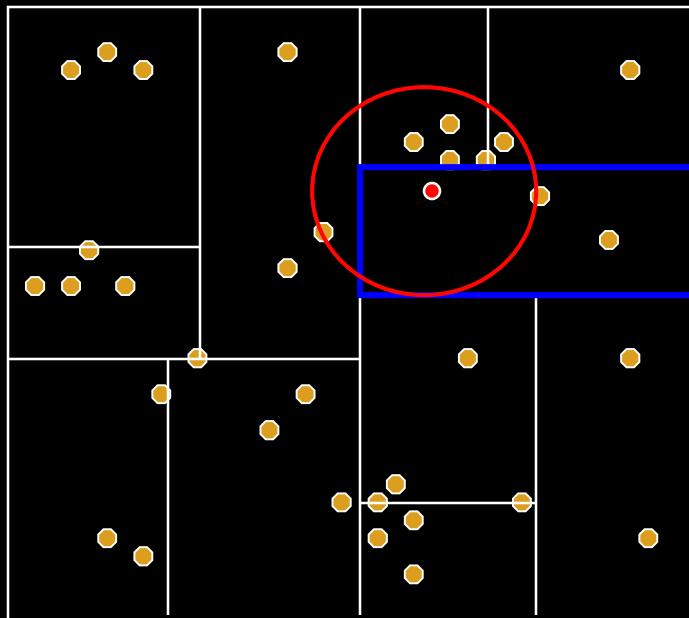
Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Nearest Neighbor with KD Trees



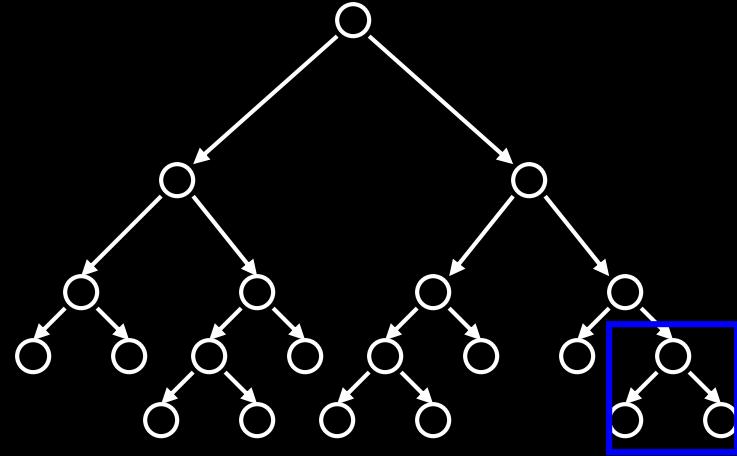
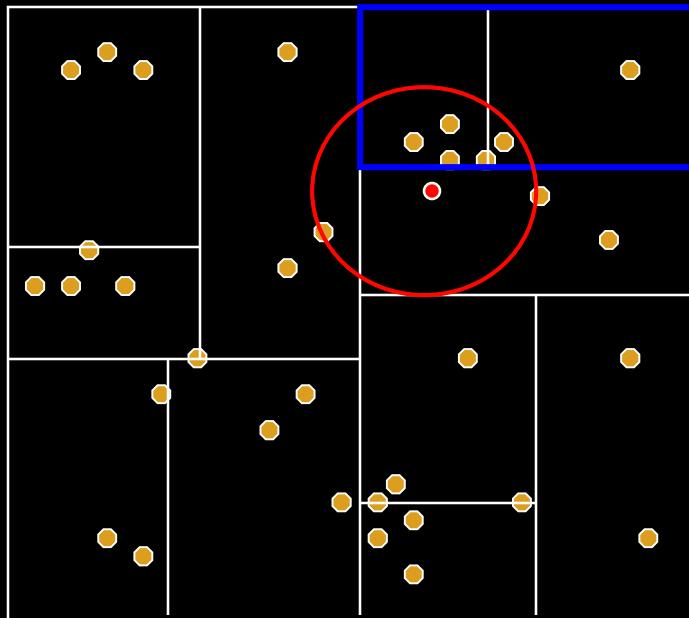
When we reach a leaf node: compute the distance to each point in the node.

Nearest Neighbor with KD Trees



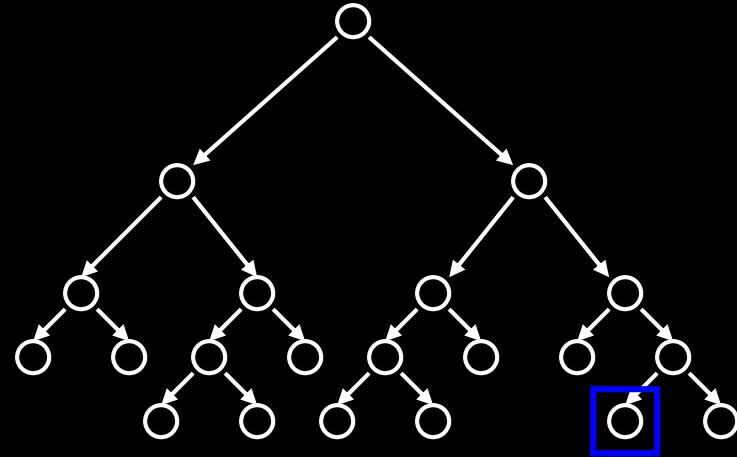
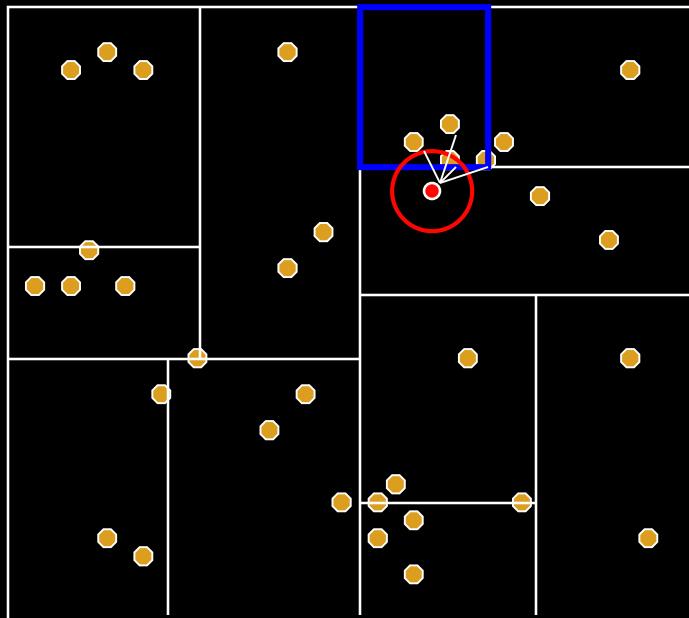
When we reach a leaf node: compute the distance to each point in the node.

Nearest Neighbor with KD Trees



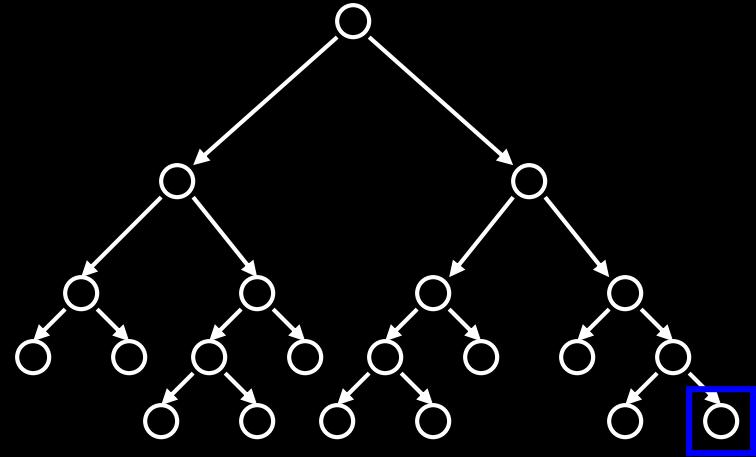
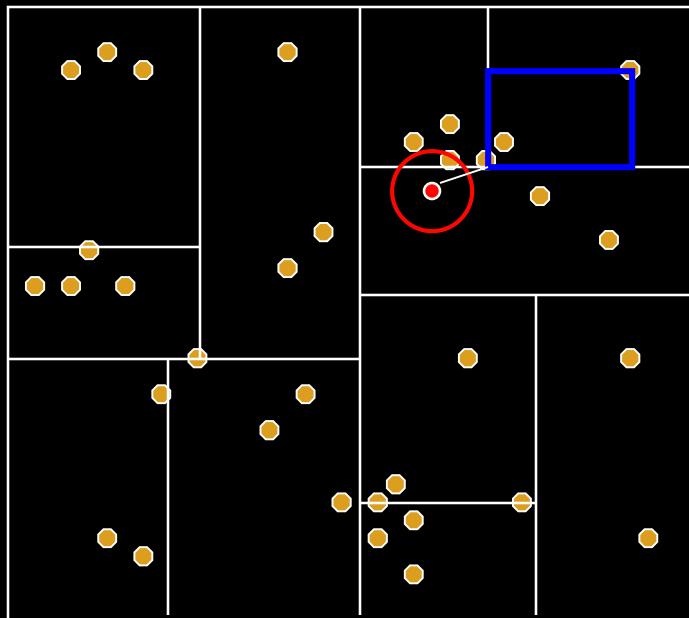
Then we can backtrack and try the other branch at each node visited.

Nearest Neighbor with KD Trees



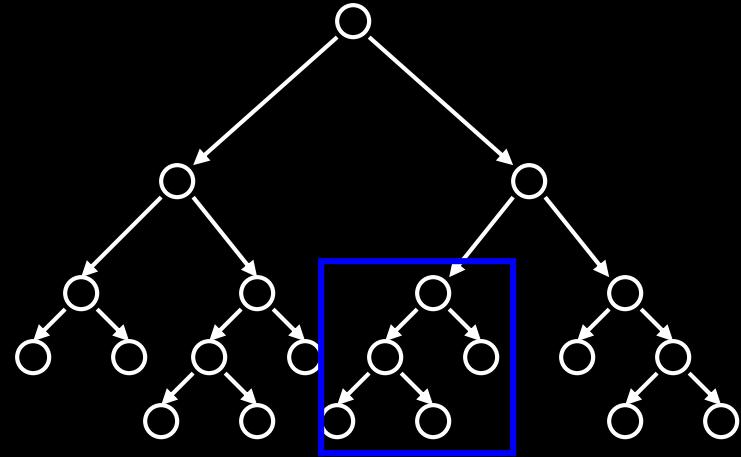
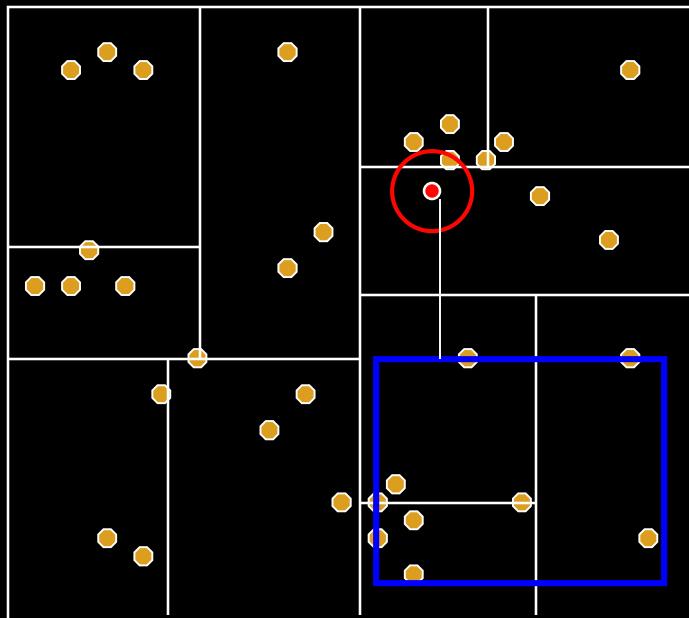
Each time a new closest node is found, we can update the distance bounds.

Nearest Neighbor with KD Trees



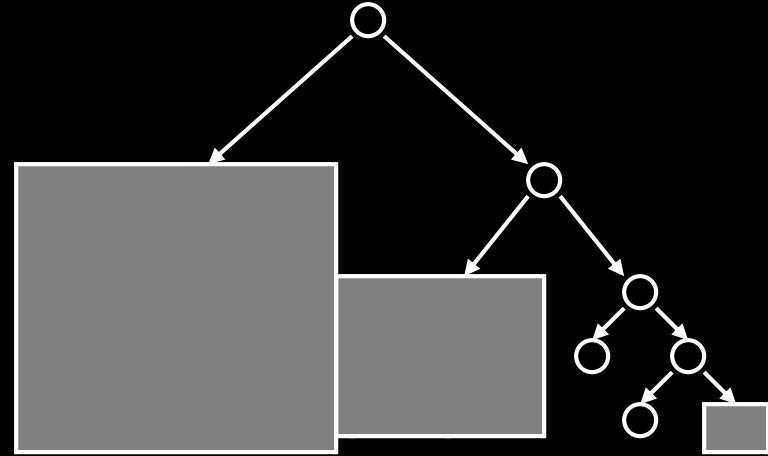
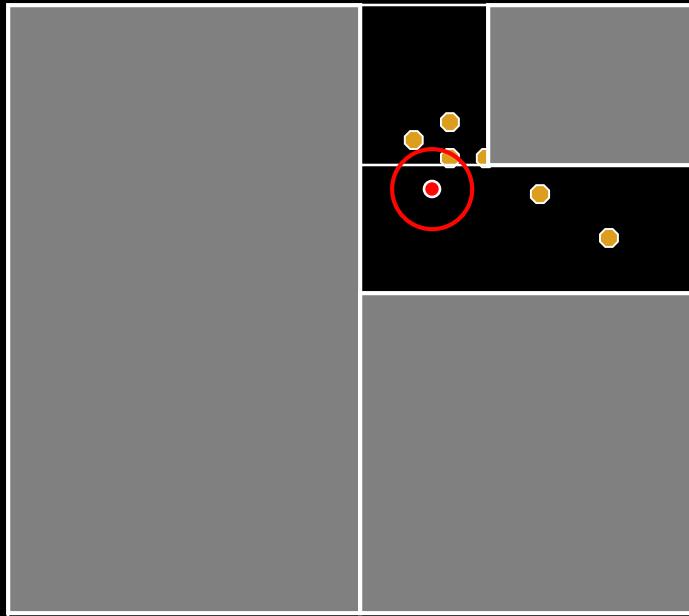
Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbor with KD Trees



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Nearest Neighbor with KD Trees



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could NOT include the nearest neighbor.

Using kd-trees

- kd-tree code is easy to find, don't implement it yourself
- kd-tree is much faster than naïve nearest-neighbor for large numbers of nodes
- BUT, cost of constructing a kd-tree is significant, so only regenerate tree once in a while (not for every new node!)
- Maintain two structures:
 - List of new nodes (clear after 1000 nodes added)
 - kd-tree (rebuilt after every 1000 new nodes)
- For each nearest-neighbor call, query kd-tree **and** list of new nodes to find nearest-neighbors

Construction Step

Start with an empty graph $G = (V, E)$

For $i = 1$ to MaxIterations

 Generate random configuration q ←

 If q is collision-free

 Add q to V

 Select k nearest nodes in V ←

 Attempt connection between each of these nodes and q using local planner

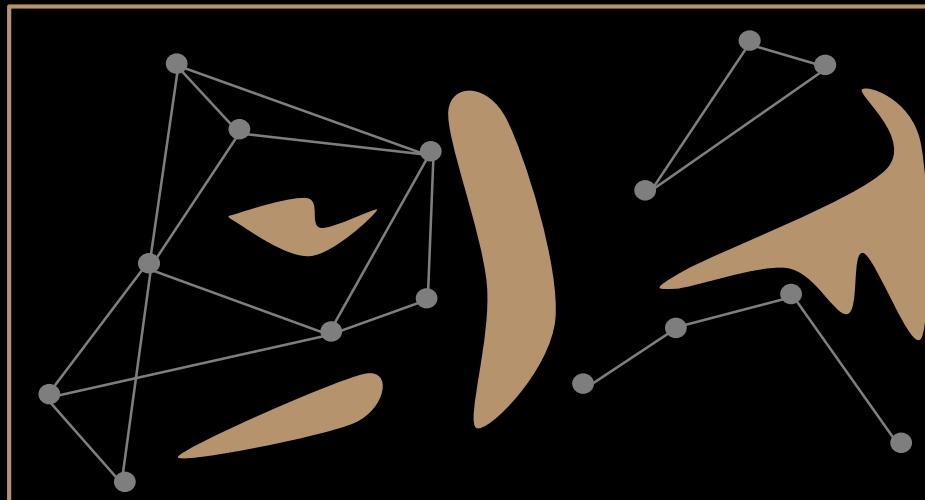
 If a connection is successful, add it as an edge in E

Local Planner

- In general, local planner can be anything that attempts to find a path between points, even another PRM!
- BUT, local planner needs to be fast b/c it's called many times by the algorithm
- Easiest and most common: Connect the two configurations with a straight line in C-space, check that the line is collision-free
 - Advantages:
 - Fast
 - Don't need to store local paths

Expansion step

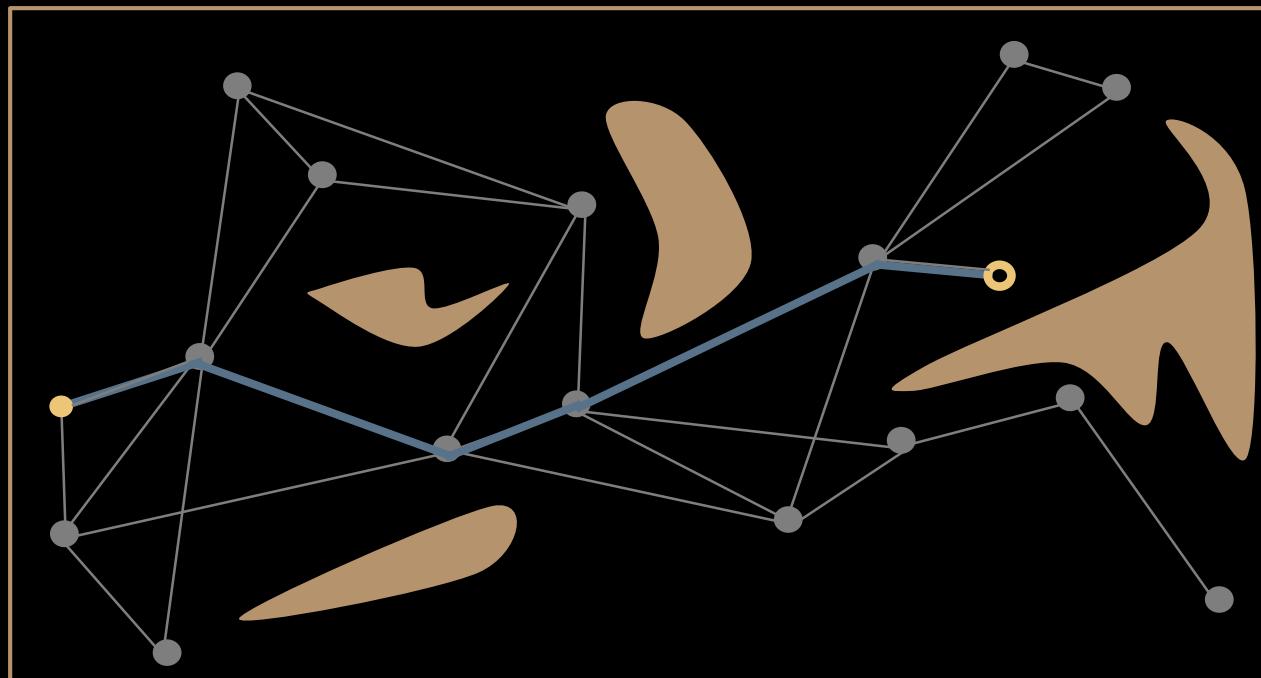
- Problem: Can have disconnected components that should be connected
 - I.e. you haven't captured the true connectivity of the space



- Expansion step uses heuristics to sample more nodes in an effort to connect disconnected components
 - Unclear how to do this the “right” way, very environment-dependent
 - See Kavraki et al. 1996 for one way to do this

Query Phase

- Given a start q_s and goal q_g
 - Connect them to the roadmap using local planner
 - May need to try more than k nearest neighbors before connection is made
 - Search G to find shortest path between q_s and q_g using A*/Dijkstra's/etc.



Path Shortening / Smoothing

- **Don't even think of using a path generated by a sampling-based planner without smoothing it first!!!**
- “Shortcut” Smoothing

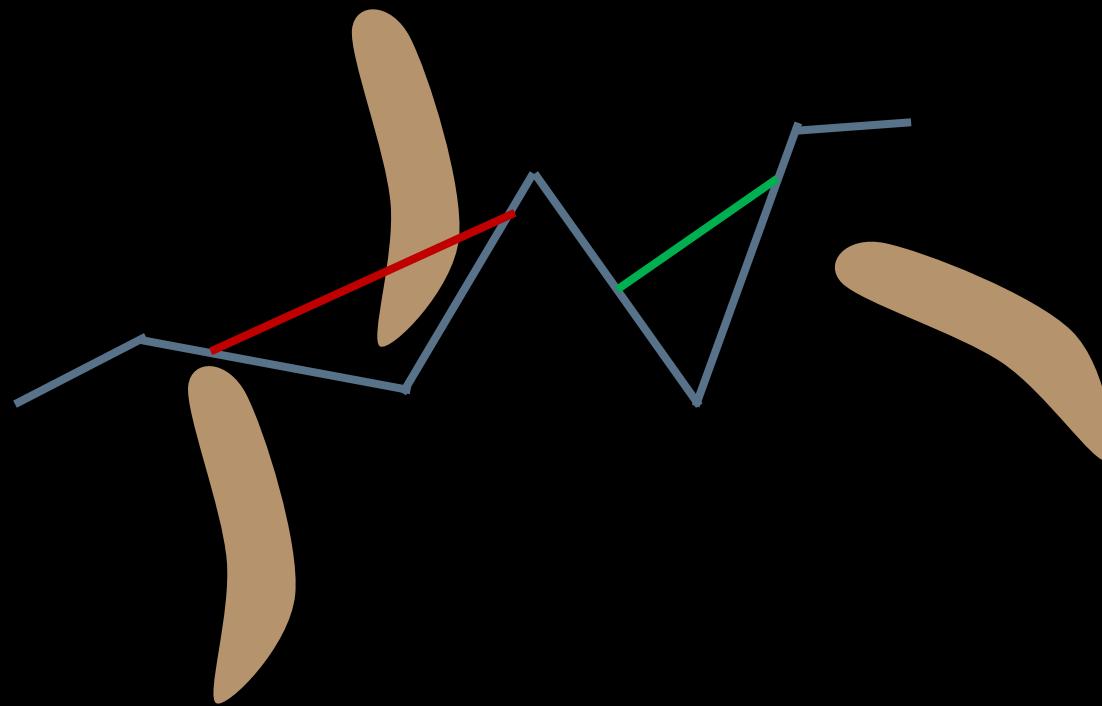
For $i = 0$ to MaxIterations

Pick two points, q_1 and q_2 , on the path randomly

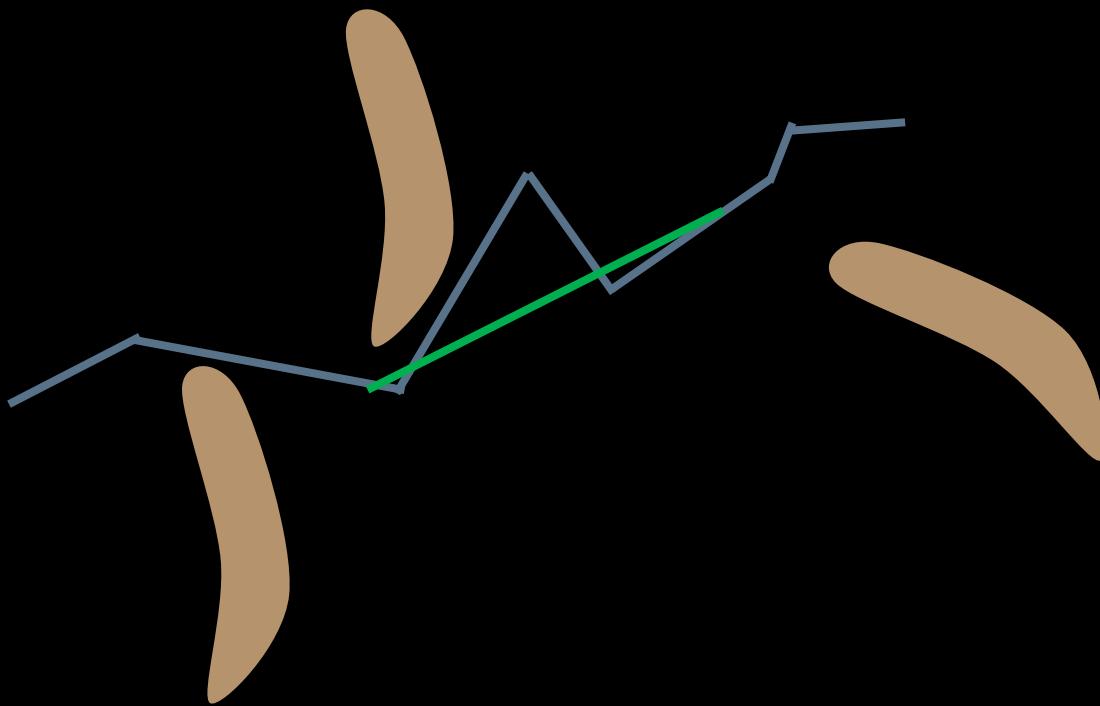
Attempt to connect (q_1, q_2) with a line segment

If successful, replace path between q_1 and q_2 with the line segment

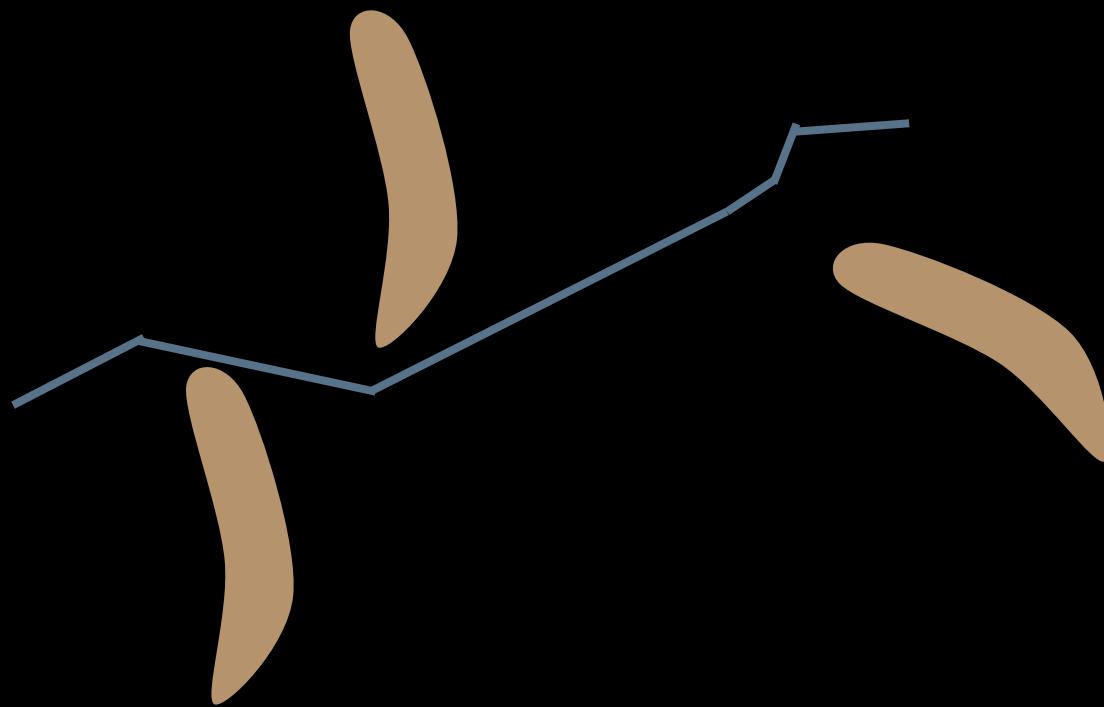
Shortcut Smoothing



Shortcut Smoothing

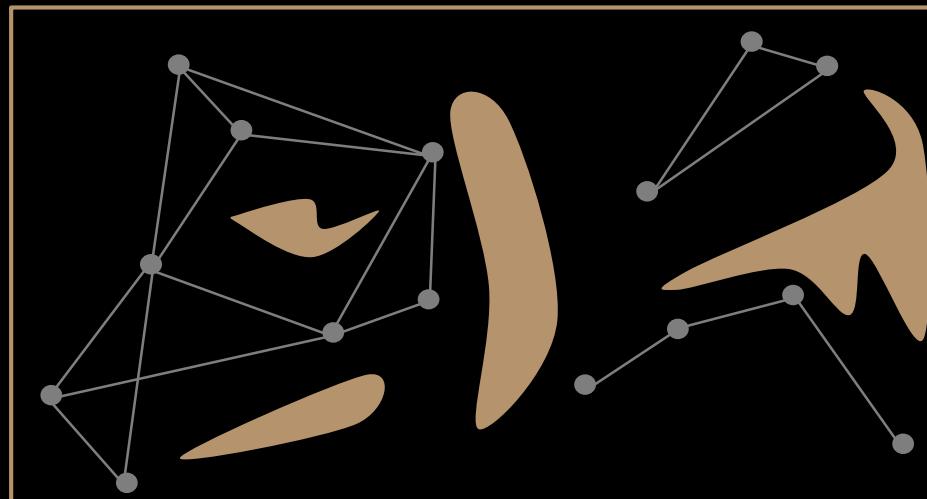


Shortcut Smoothing



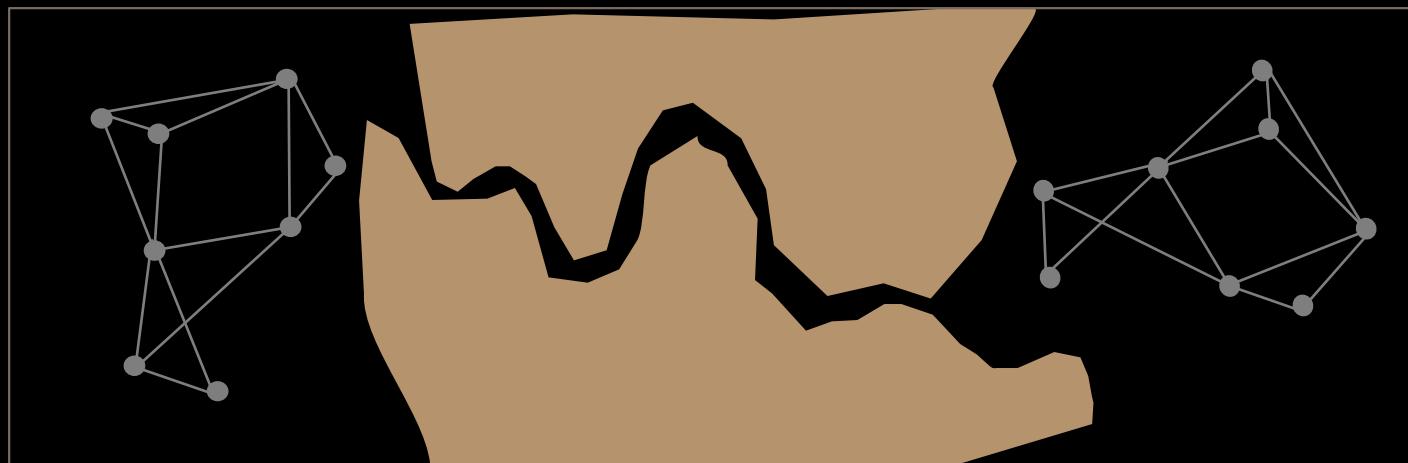
PRM Failure Modes

1. Can't connect q_s and q_g to any nodes in the graph
 - Come up with an example in the graph below
2. Can't find a path in the graph but a path is possible (more common)
 - Come up with an example in the graph below



Why do failures happen?

- Roadmap doesn't capture connectivity of space
 - Can run the learning phase longer
 - Can change sampling strategy to focus on narrow passages



- Local planner is too simple
 - Can use more sophisticated local planner

What happens in the limit?

- What if we ran the construction step of the PRM for infinite time...
 - What would the graph look like?
 - Would it capture the connectivity of the free space?
 - Would any start and goal be able to connect to the graph?
 - Is the PRM algorithm probabilistically complete?

Break

Expansive Spaces

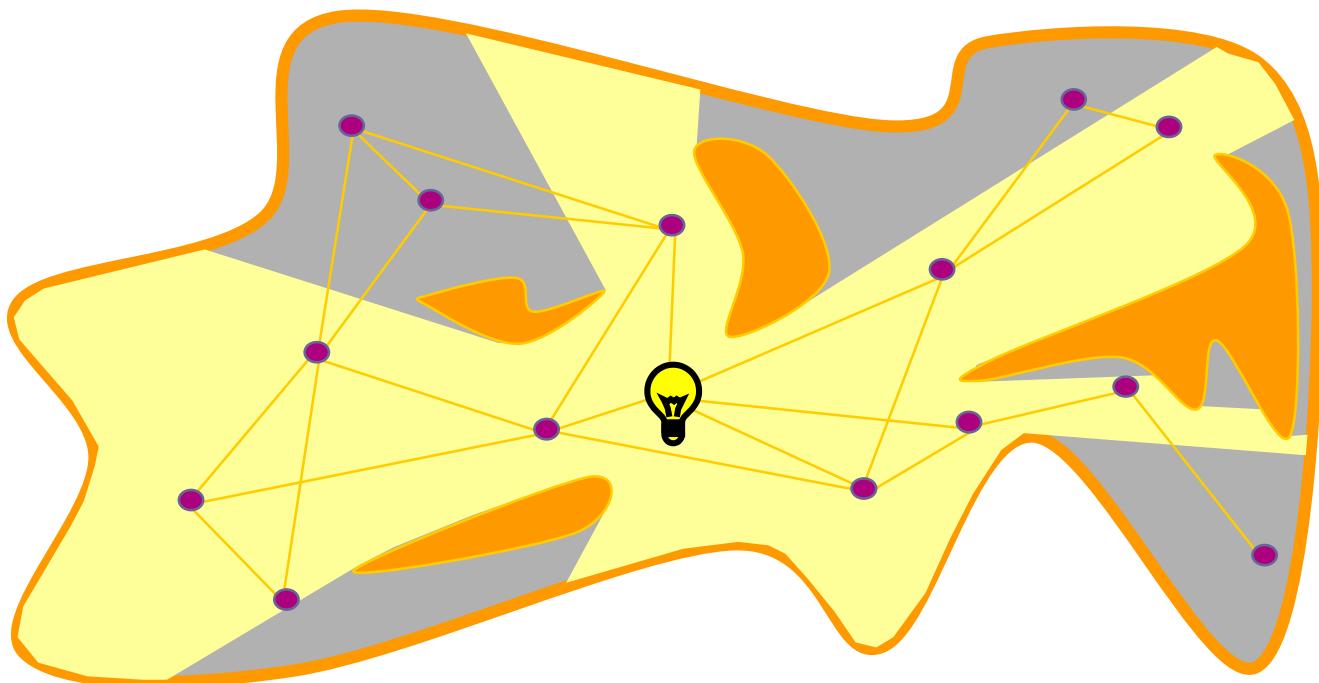


Analysis of Probabilistic Roadmaps

Slides from David Hsu

Issues of probabilistic roadmaps

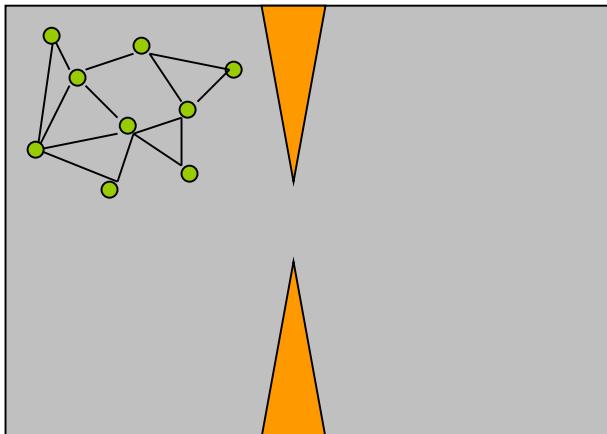
- ❑ Coverage
- ❑ Connectivity



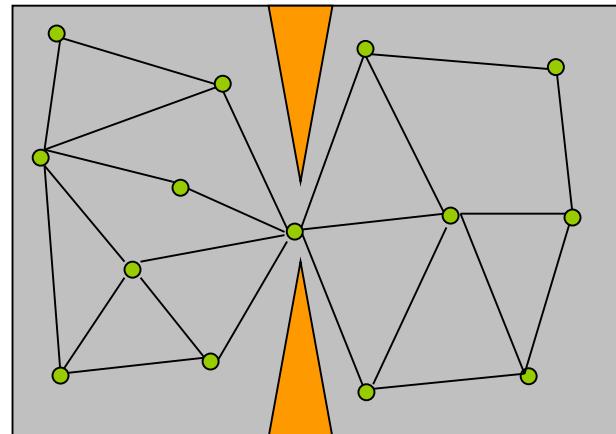
Is the coverage adequate?

- Milestones should be distributed so that almost **any** point of the configuration space can be connected by a straight line segment to one milestone.

Bad



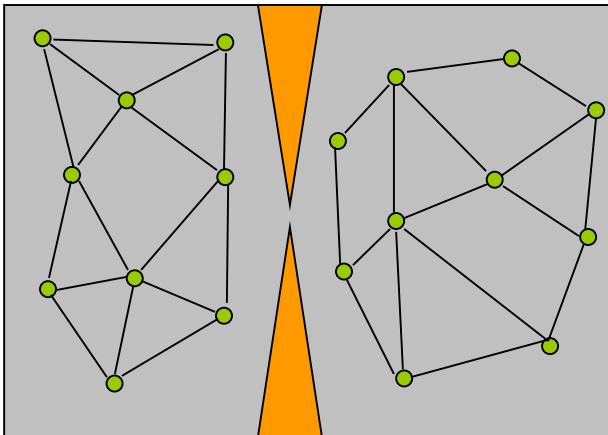
Good



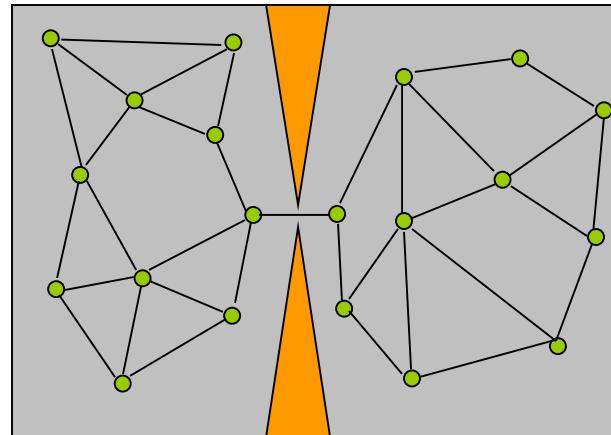
Connectivity

- There should be a one-to-one correspondence between the connected components of the roadmap and those of F (F is C_{free}).

Bad

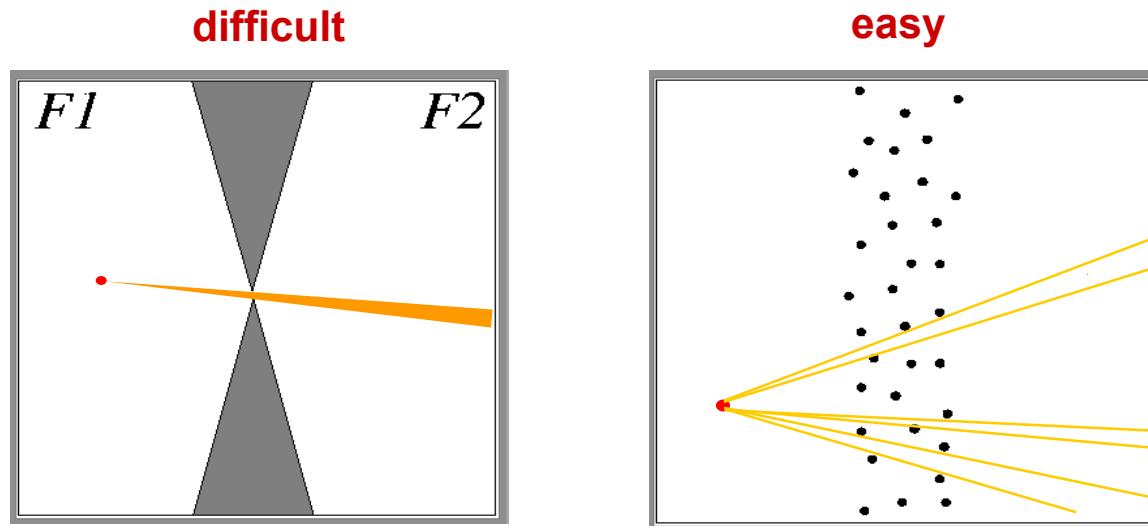


Good



Narrow passages

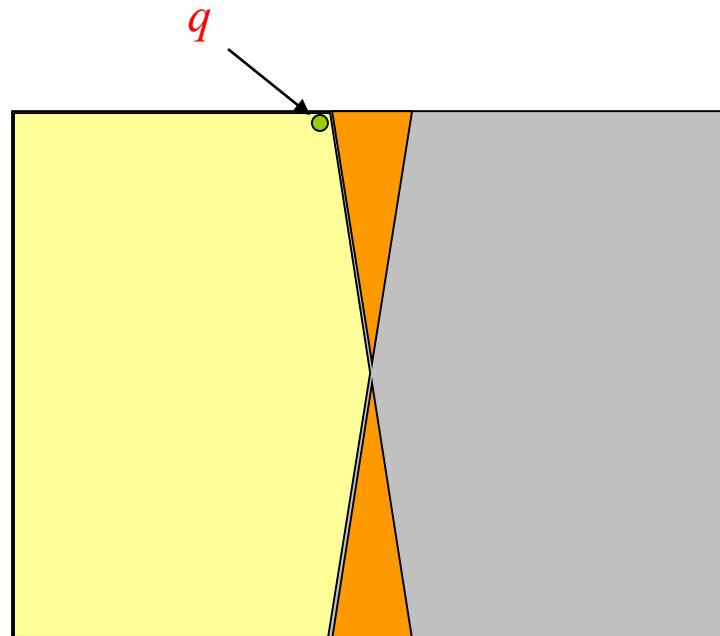
- Connectivity is difficult to capture when there are narrow passages.
- Narrow passages are difficult to define.



- How to characterize coverage & connectivity?
Expansiveness

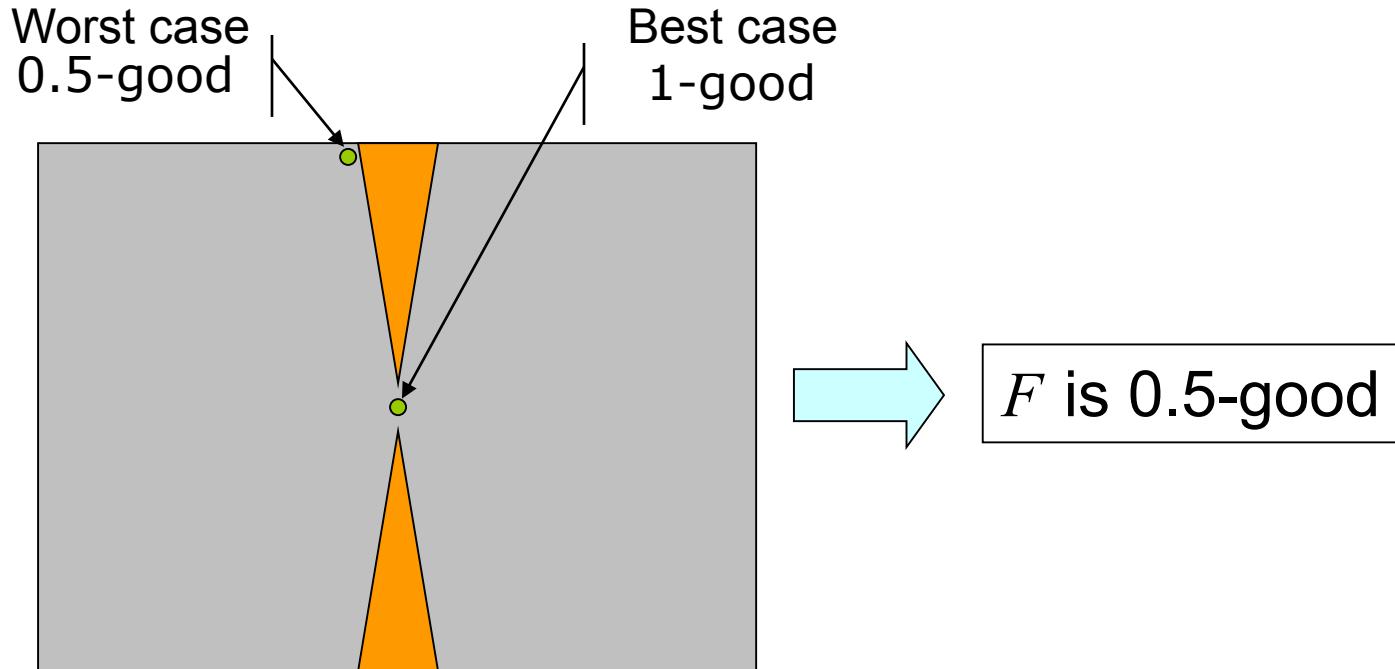
Definition: visibility set

- Visibility set of q
 - All configurations in F that can be connected to q by a straight-line path in F
 - All configurations “seen” by q



Definition: ϵ -good

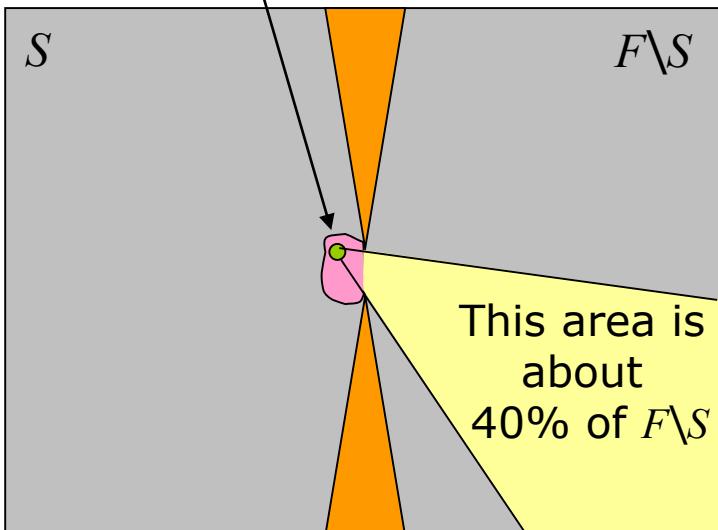
- Every free configuration sees at least ϵ fraction of the free space, ϵ in $(0, 1]$.



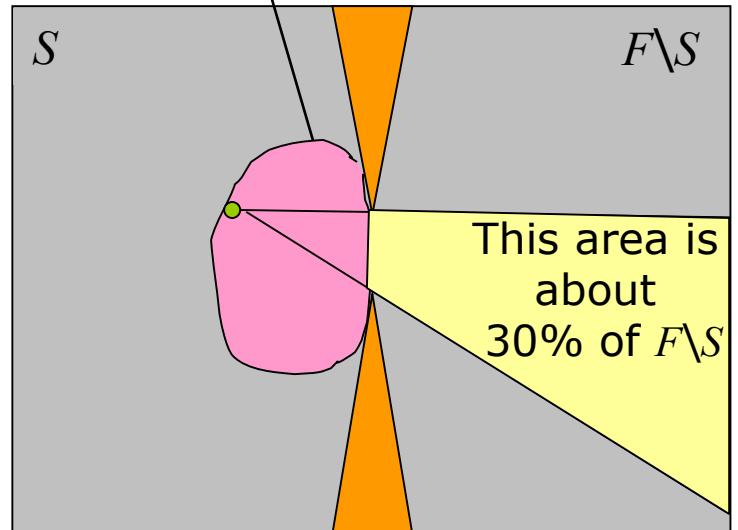
Definition: lookout of a subset S

- Subset of points in S that can see at least β fraction of $F \setminus S$, β is in $(0, 1]$.

0.4-lookout of S

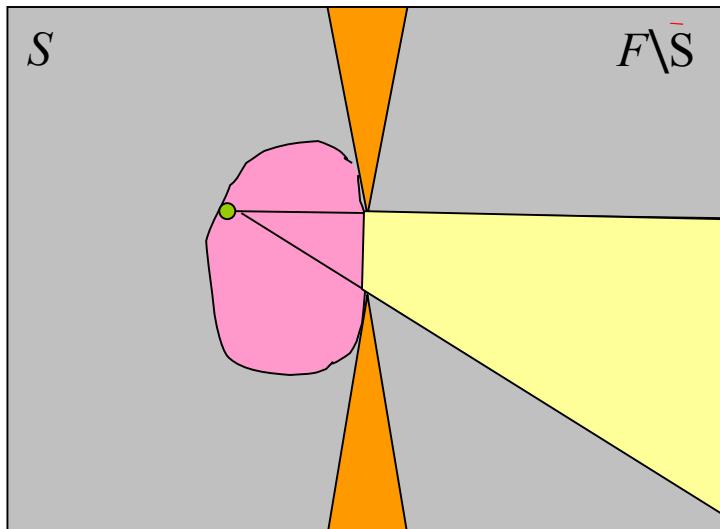


0.3-lookout of S



Definition: $(\varepsilon, \alpha, \beta)$ -expansive

- The free space F is $(\varepsilon, \alpha, \beta)$ -expansive if
 - Free space F is ε -good
 - For each subset S of F , its β -lookout is at least α fraction of S . $\varepsilon, \alpha, \beta$ are in $(0, 1]$



F is ε -good $\rightarrow \varepsilon=0.5$

β -lookout $\rightarrow \beta=0.4$

$$\frac{\text{Volume}(\beta\text{-lookout})}{\text{Volume}(S)} \rightarrow \alpha=0.2$$

F is $(\varepsilon, \alpha, \beta)$ -expansive,
where $\varepsilon=0.5$, $\alpha=0.2$, $\beta=0.4$.

Why expansiveness?

- ε , α , and β measure the expansiveness of a free space.
- Bigger ε , α , and β → easier to construct a roadmap with good connectivity and coverage.

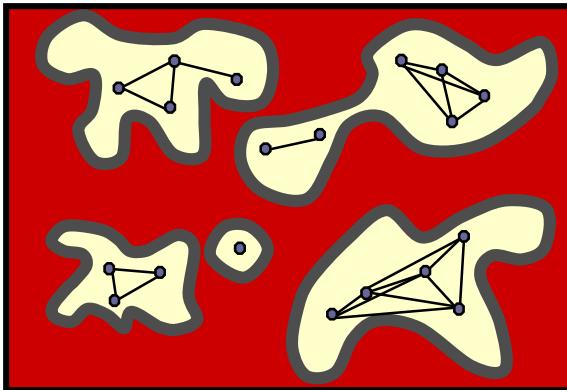
Theorem 1(Connectivity)

- Probability of achieving good connectivity **increases exponentially** with the number of milestones (in an expansive space).
- If $\varepsilon, \alpha, \beta$ decreases, then need to **increase the number of milestones** (to maintain good connectivity)

D. Hsu, J. C. Latombe and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications* Vol. 9, Nos. 4 & 5 (1999), pg. 495-512

Uniform sampling

- All-pairs path planning



- **Theorem 1 :** A roadmap of $\frac{16 \ln(1/\gamma)}{\varepsilon\alpha} + \frac{6}{\beta}$ uniformly-sampled milestones has the correct connectivity with probability at least $1 - \gamma$.

Theorem 2 (Coverage)

- Probability of achieving good coverage, **increases exponentially** with the number of milestones (in an expansive space).

Completeness

- Complete algorithms are slow.
 - A **complete** algorithm finds a path if one exists and reports no otherwise.
 - Example: Visibility graph
- Heuristic algorithms are unreliable.
 - Example: potential field
- **Probabilistic completeness**
 - Intuition: If there is a solution path, the algorithm will find it with high probability.

Probabilistic completeness

In an expansive space, the probability that a PRM planner fails to find a path when one exists goes to 0 **exponentially** in the number of milestones (~ running time).

[Kavraki, Latombe, Motwani, Raghavan,95]

[Hsu, Latombe, Motwani, 97]

Summary

□ Main result

- If a C-space is expansive, then a roadmap can be constructed efficiently with good connectivity and coverage.

□ Limitations in practice

- It does not tell you when to stop growing the roadmap.
- A planner stops when either a path is found or max steps are reached.
- $\varepsilon, \alpha, \beta$ are nearly impossible to compute in high-dimensional spaces

Homework

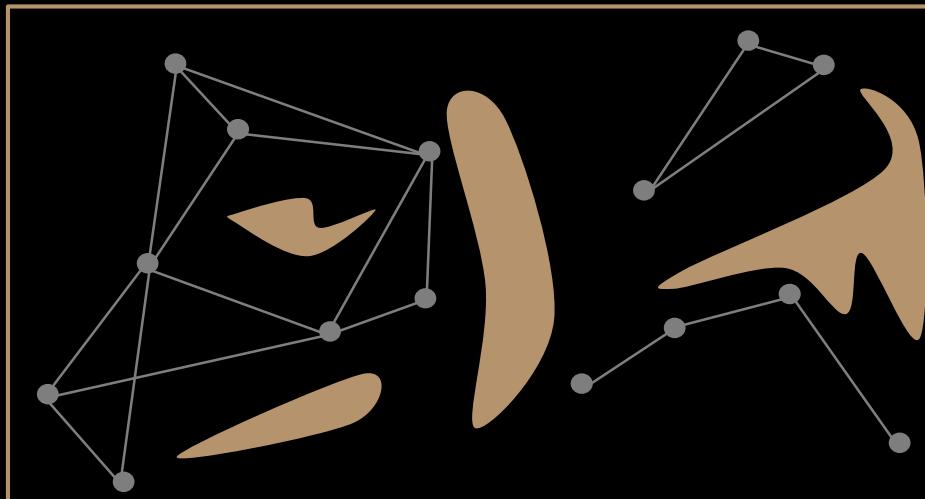
- HW2 is posted!
- Make sure you've read Chapter 5

Sampling-based Planning 2

A lot of Material from Howie Choset, Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, J. Kuffner

Last time...

- We discussed PRMs



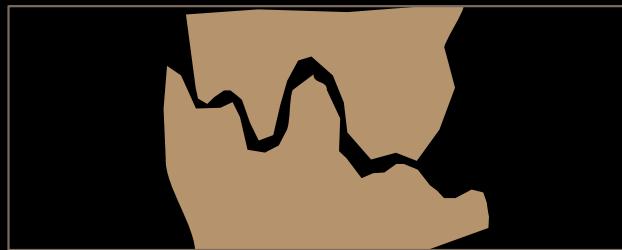
- Two issues with the PRM:
 1. Uniform random sampling misses narrow passages
 2. Exploring whole space, but all we want is a path

Outline

- Sampling strategies
- RRTs

Sampling Strategies

- Most common is uniform random sampling
 - The bigger the area, the more likely it will be sampled
 - Problem: Narrow passages

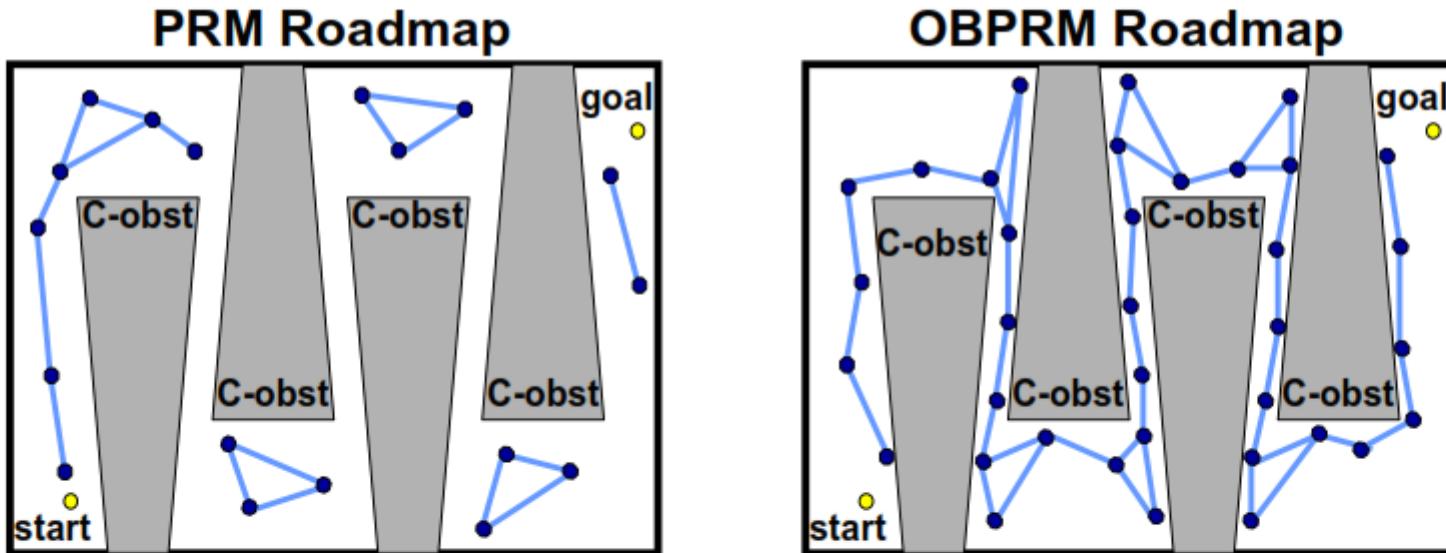


- Are narrow passages inherently bad?
 - Does A* running on a 2D grid have problems with narrow passages?

OBPRM: An Obstacle-Based PRM

[IEEE ICRA'96, IEEE ICRA'98, WAFR'98]

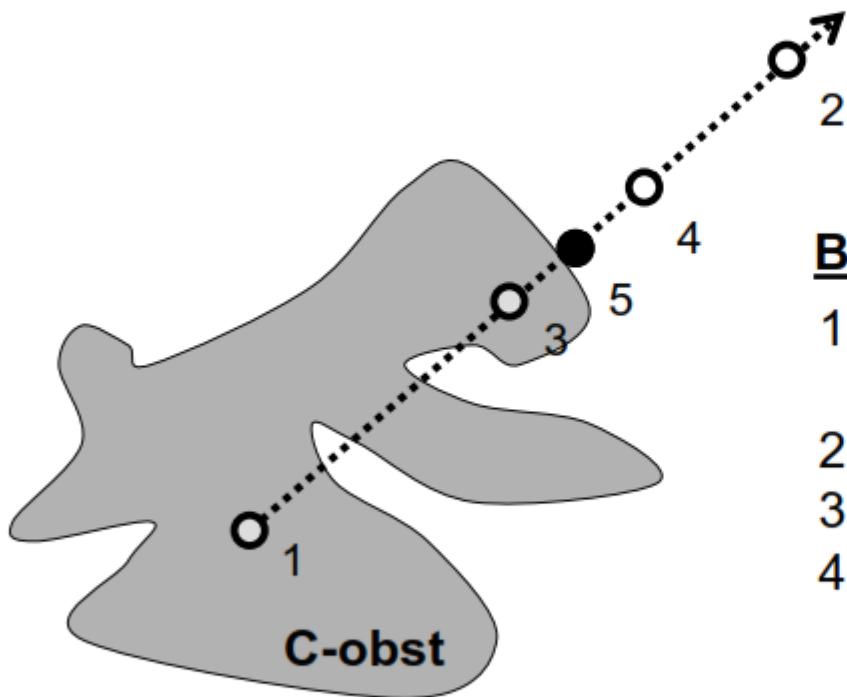
- To Navigate Narrow Passages we must sample in them
- most PRM nodes are where planning is easy (not needed)



Idea: Can we sample nodes near C-obstacle surfaces?

- we cannot explicitly construct the C-obstacles...

OBPRM: Finding Points on C-obstacles

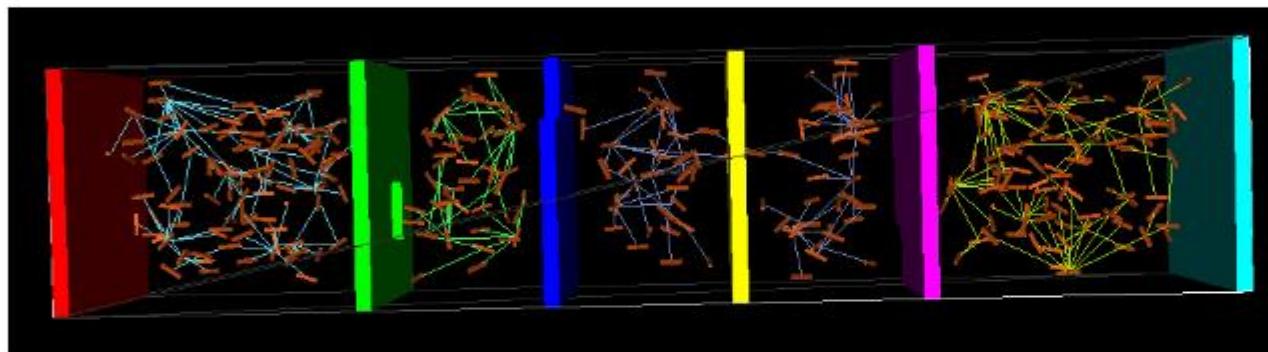


Basic Idea (for workspace obstacle S)

1. Find a point in S's C-obstacle
(robot placement colliding with S)
2. Select a random direction in C-space
3. Find a free point in that direction
4. Find boundary point between them
using binary search (collision checks)

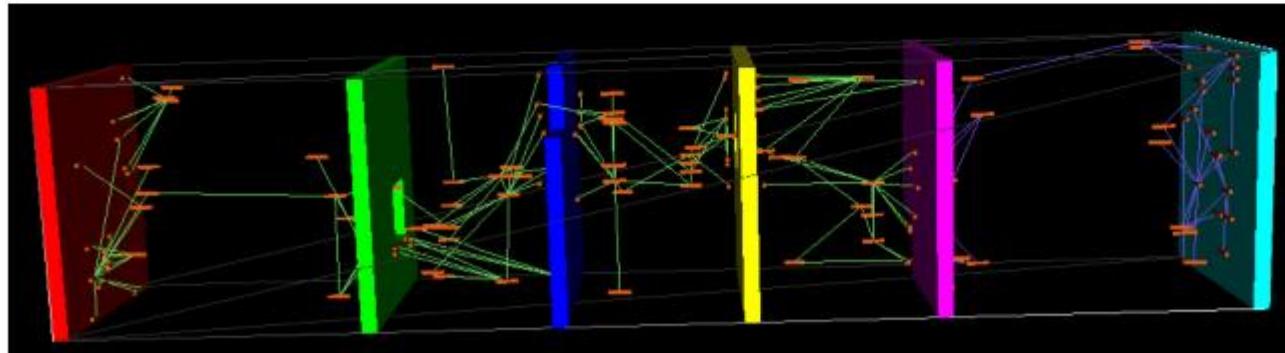
Note: we can use more sophisticated heuristics to try to cover C-obstacle

PRM vs OBPRM Roadmaps



PRM

- 328 nodes
- 4 major CCs

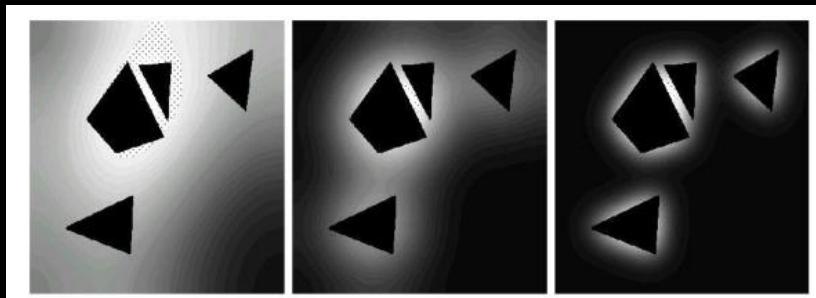


OBPRM

- 161 nodes
- 2 major CCs

Sampling strategies: Gaussian

- Gaussian sampler
 - Find a q_1 at random
 - Pick a q_2 from a Gaussian distribution centered at q_1
 - If **both** are in collision or collision-free, discard them, if one free, keep it

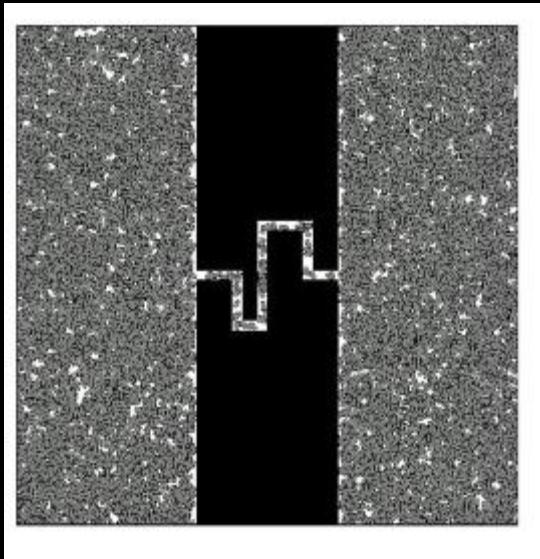


Sampling distribution for varying Gaussian width
(width decreasing from left to right)

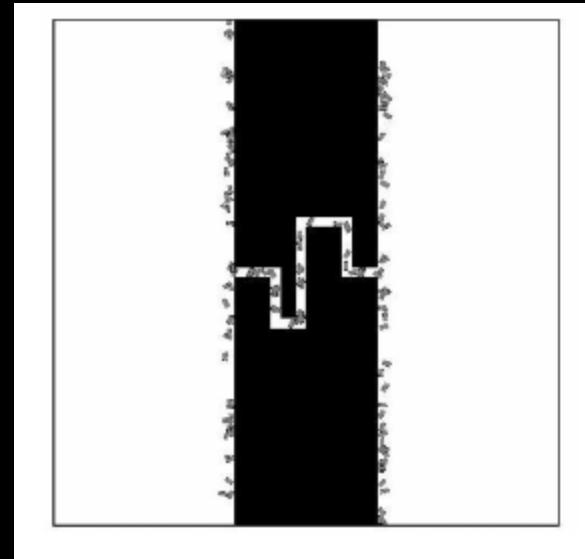
Boor, Valérie, Mark H. Overmars, and A. Frank van der Stappen. "The gaussian sampling strategy for probabilistic roadmap planners." *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE, 1999.

Sampling Strategies: Gaussian

- Performs well in narrow passages



Uniform Random Sampling



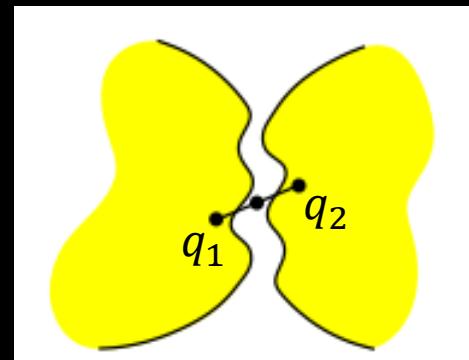
Gaussian Sampling

Sampling Strategies

- Can we come up with a case where obstacle-biased sampling is worse than uniform random sampling?

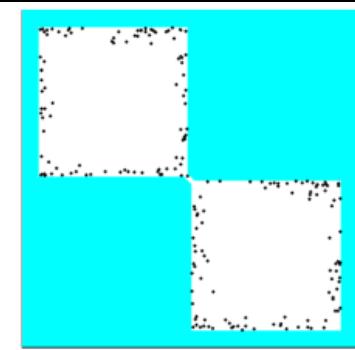
Sampling Strategies: Bridge

- Sample a q_1 that is in collision
- Sample a q_2 in neighborhood of q_1 using some probability distribution (e.g. gaussian)
- If q_2 in collision, get the midpoint of (q_1, q_2)
- Check if midpoint is in collision, if not, add it as a node

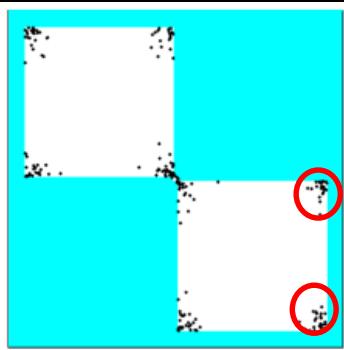


Hsu, David, et al. "The bridge test for sampling narrow passages with probabilistic roadmap planners." *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 3. IEEE, 2003.

Sampling Strategies: Bridge

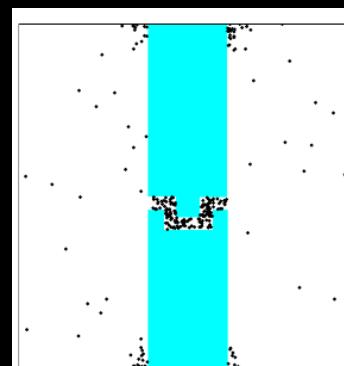


Gaussian
Sampling

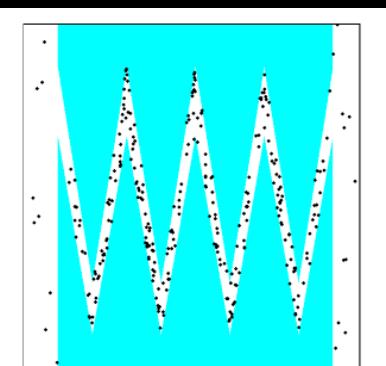


Bridge
Sampling

What's going on
at the corners?



Bridge Sampling performs
well in narrow passages



Sampling Strategies: Deterministic

- The problem: Random sampling (biased or not) can be unpredictable and irregular
 - Each time you run your algorithm you get a different sequence of samples, so performance varies
 - In the limit, space will be sampled well, but in finite time result may be irregular

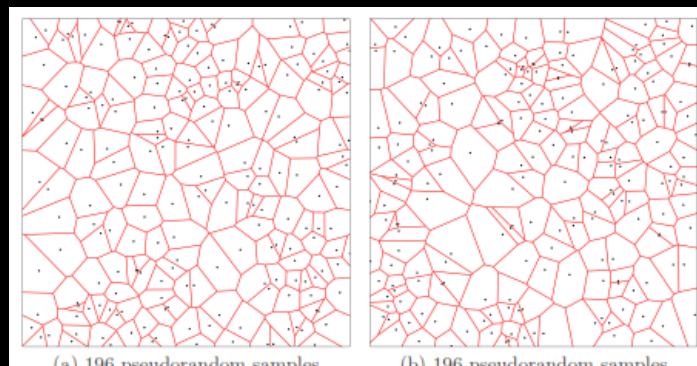


Figure 5.3: Irregularity in a collection of (pseudo)random samples can be nicely observed with Voronoi diagrams.

- Can we do better?

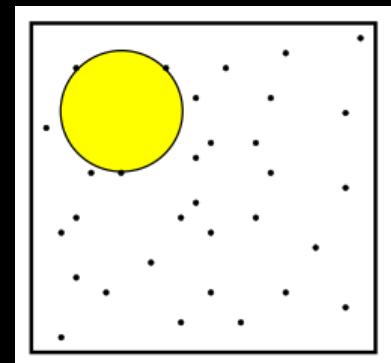
Sampling Strategies: Deterministic

- What do we care about?
 - Dispersion

$$\delta(P) = \sup_{x \in X} \left\{ \min_{p \in P} \{\rho(x, p)\} \right\}.$$

P is a finite set of points, (X, ρ) is a metric space (ρ is a distance metric)

In English: the radius of the largest empty ball



Sampling Strategies: Deterministic

- What do we care about?
 - Discrepancy

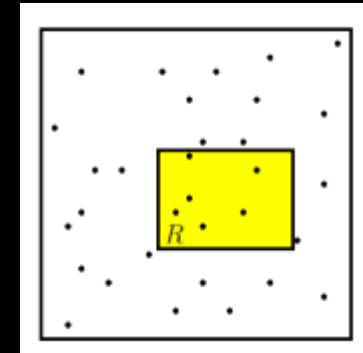
$$D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left\{ \left| \frac{|P \cap R|}{k} - \frac{\mu(R)}{\mu(X)} \right| \right\}$$

P is a finite set of points, $k = |P|$

R is the range space (set of axis-aligned boxes)

X is a metric space, μ is a measure of volume

- In English: The largest volume estimation error that can be obtained over all sets in \mathcal{R}
- Each term captures how well P can be used to estimate the volume of R .
 - Example: If $\mu(R)$ is 1/2 of $\mu(X)$, then we want 1/2 of the points in P to be in R .



Discrepancy

Sampling Strategies: Deterministic

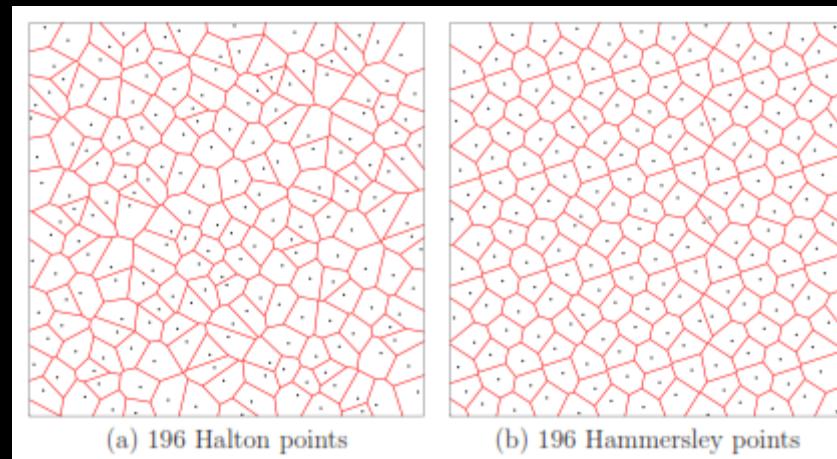
- *Deterministic Sampling*
 - Similar to discretization we saw in Discrete Motion Planning, but *order* of samples matters
 - Example: van der Corput sequence

	Naive	Reverse	Van der		
i	Sequence	Binary	Binary	Corput	Points in $[0, 1]$ / \sim
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	

Figure 5.2: The van der Corput sequence is obtained by reversing the bits in the binary decimal representation of the naive sequence.

Sampling Strategies: Deterministic

- Halton sequence: n-dimensional generalization of van der Corput sequence
- Hammersley sequence: Adaptation of Halton sequence that yields a better distribution. BUT need to know number of samples in advance.



- See LaValle's book for formulas.

Break

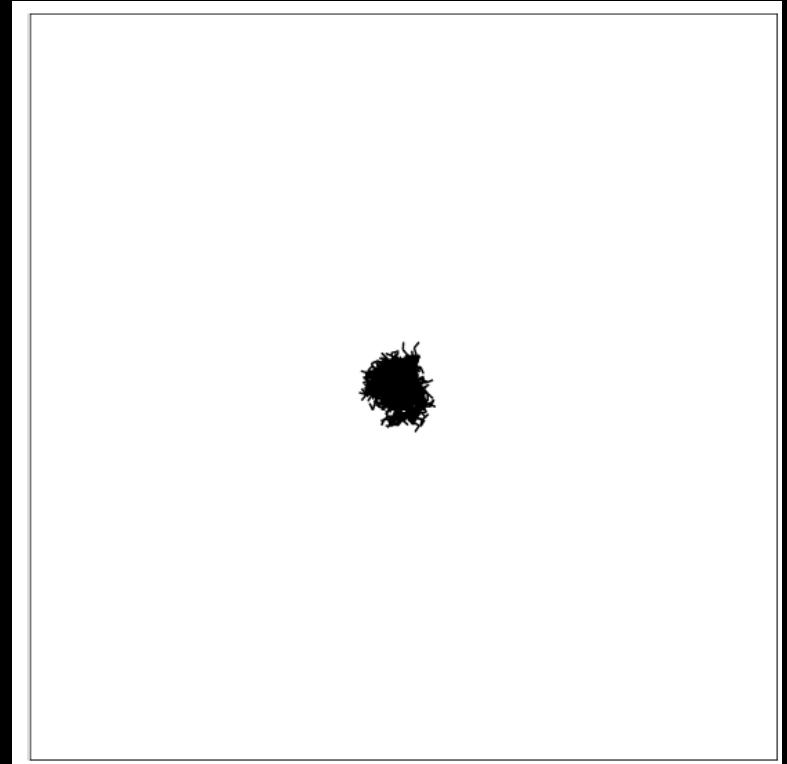
Rapidly-exploring Random Trees (RRTs)

Single-query methods

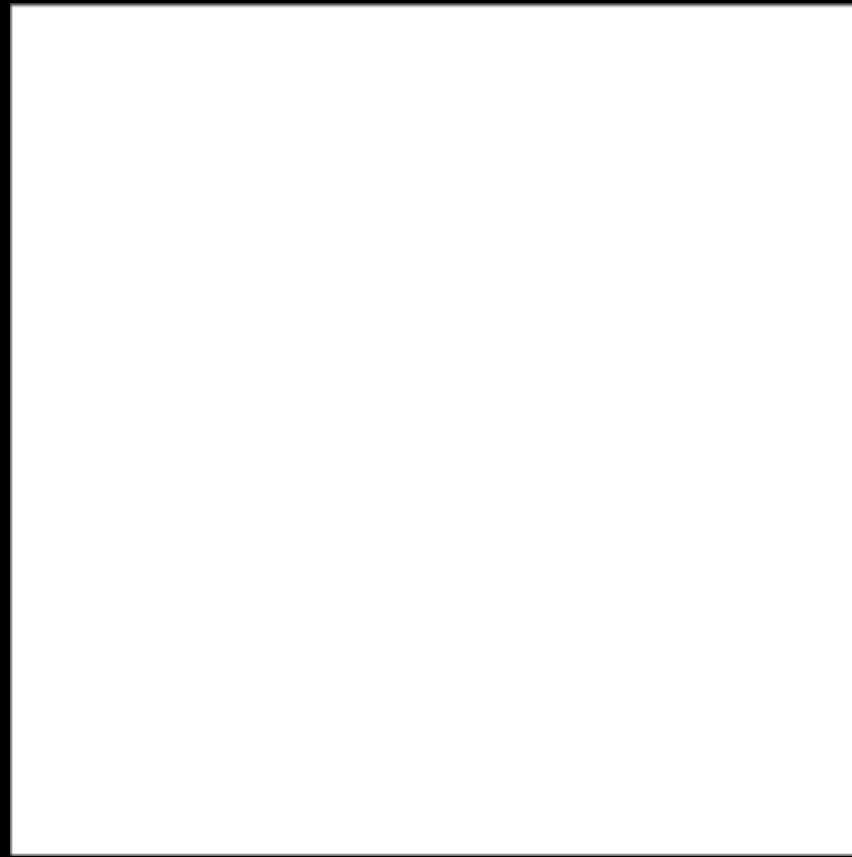
- Motivation: Why try to capture the connectivity of the whole space when all you need is one path?
- Algorithms:
 - Single-Query BiDirectional Lazy PRM (SBL-PRM)
 - Expansive Space Trees (EST)
 - **Rapidly-exploring Random Tree (RRT)**
- Key idea: Build a *tree* instead of a general graph.
- The tree “grows” in C_{free}
 - Like PRM, captures some connectivity
 - Unlike PRM, only explores what is connected to q_{start}

Naïve Tree Algorithm

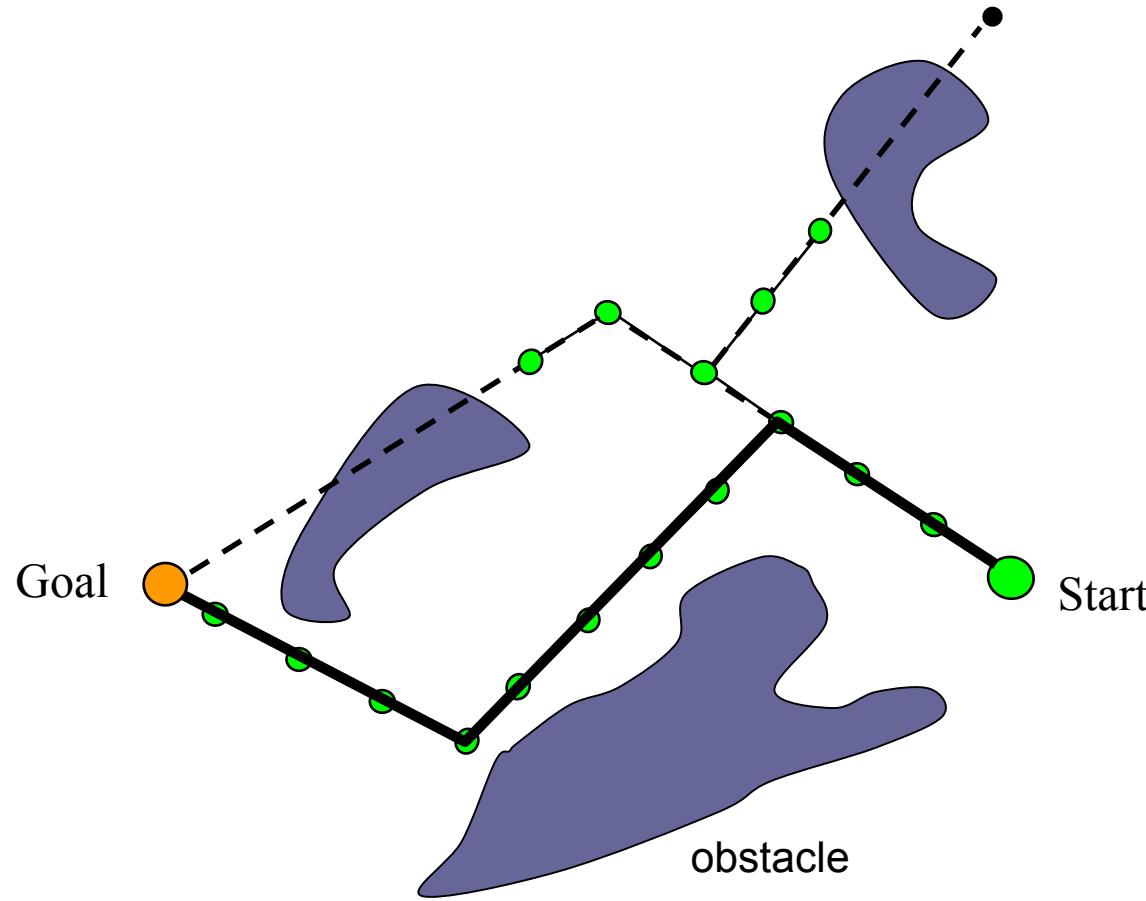
```
qnode = qstart
For i = 1 to NumberSamples
    qrand = Sample near qnode
    Add edge e = (qrand , qnode) if
    collision-free
    qnode = Pick random node of tree
```



RRT Growing in Empty Space

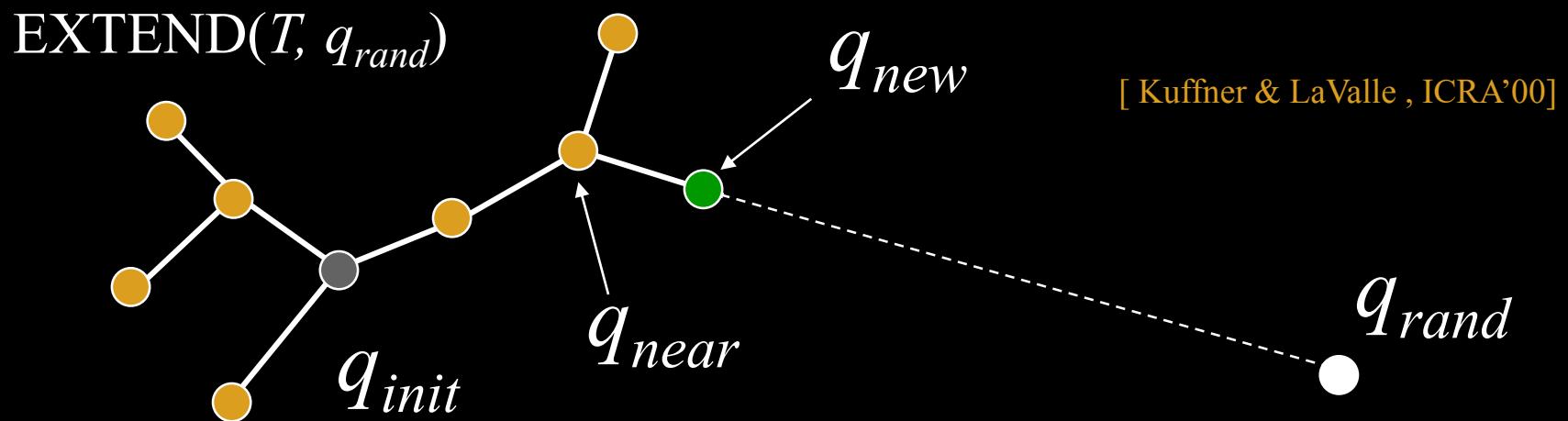


RRT with obstacles and goal bias

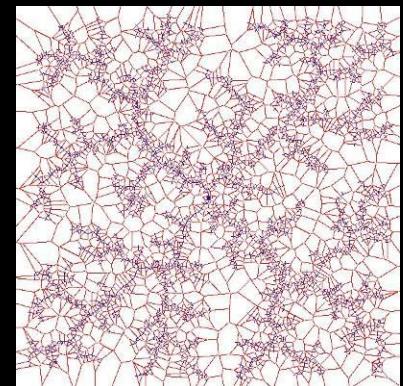
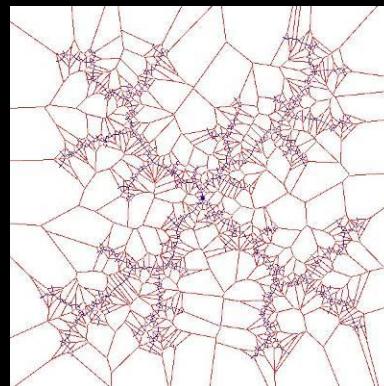
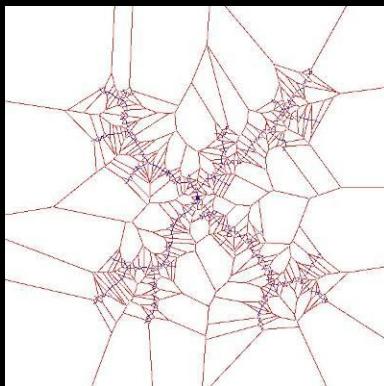
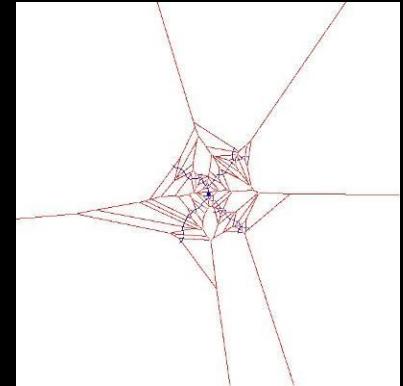
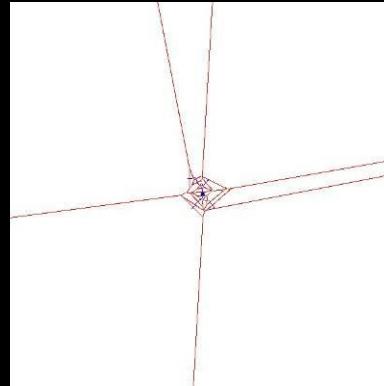
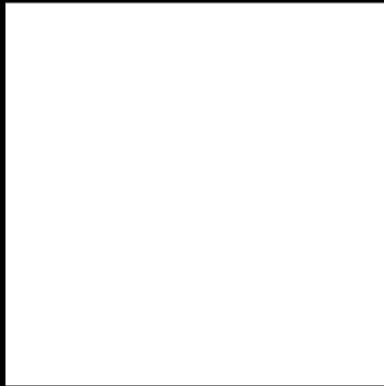


Path Planning with Rapidly-Exploring Random Trees (RRTs)

```
BUILD_RRT ( $q_{init}$ ) {  
     $T.init(q_{init})$ ;  
    for  $k = 1$  to K do  
         $q_{rand} = \text{RANDOM\_CONFIG}()$ ;  
        EXTEND( $T, q_{rand}$ )  
    }  
}
```



RRTs bias exploration toward large Voronoi regions



<http://msl.cs.uiuc.edu/rrt/gallery.html>

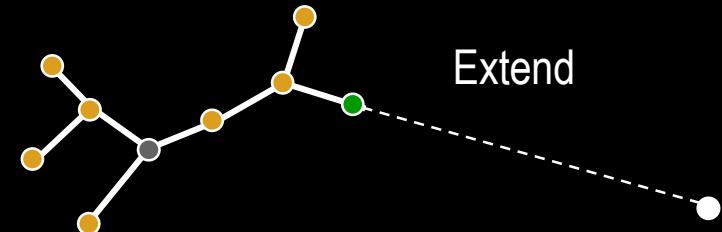
- What about obstacles?

RRT Goal Biasing

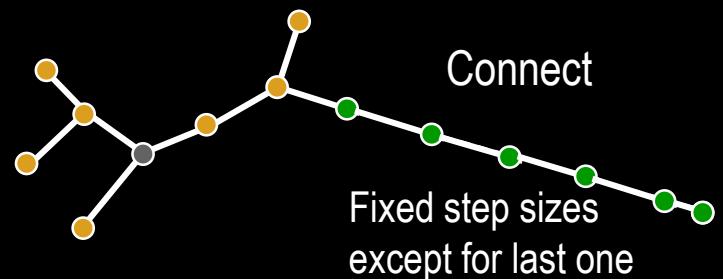
- In “pure” form RRTs are great at filling space, but we need a path!
- Need to bias RRTs toward goal to produce a path
 - When generating a random sample, with some probability pick the goal instead of a random node
 - This introduces another parameter
 - James Kuffner’s experience is that 5-10% is the right choice
- What happens if you set probability of sampling goal to 100%?

RRT Extension Types

- RRT-Extend
 - Take one step toward a random sample



- RRT-Connect
 - Step toward random sample until it is either
 - Reached
 - You hit an obstacle

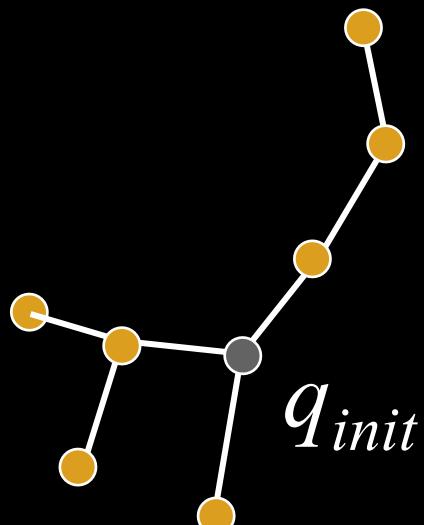


- Note that RRT-Connect is defined differently in some places online!

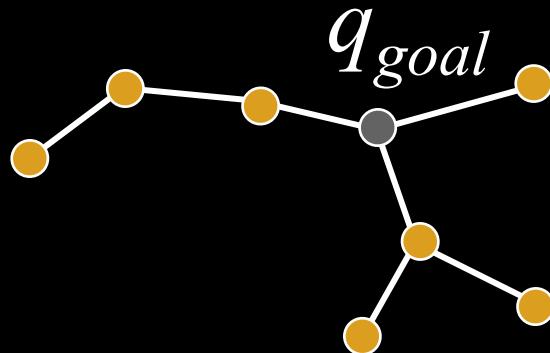
BiDirectional RRTs

- BiDirectional RRT
 - Grow trees from both start and goal
 - Try to get trees to connect to each other
 - Trees can both use Extend or both use Connect or one use Extend and one Connect
- BiDirectional RRT with Connect for both trees is my favorite, I always try this first
 - This variant has only one parameter; the step size

Example of BiDirectional RRT

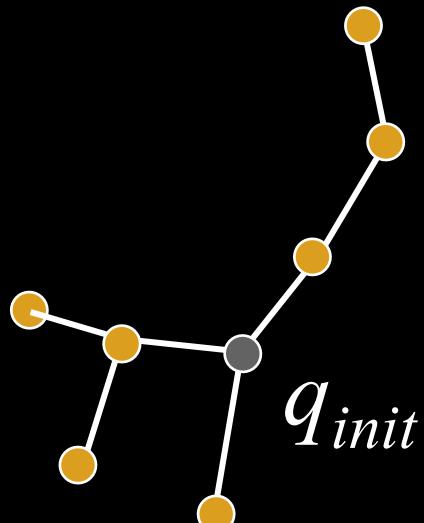


Connect

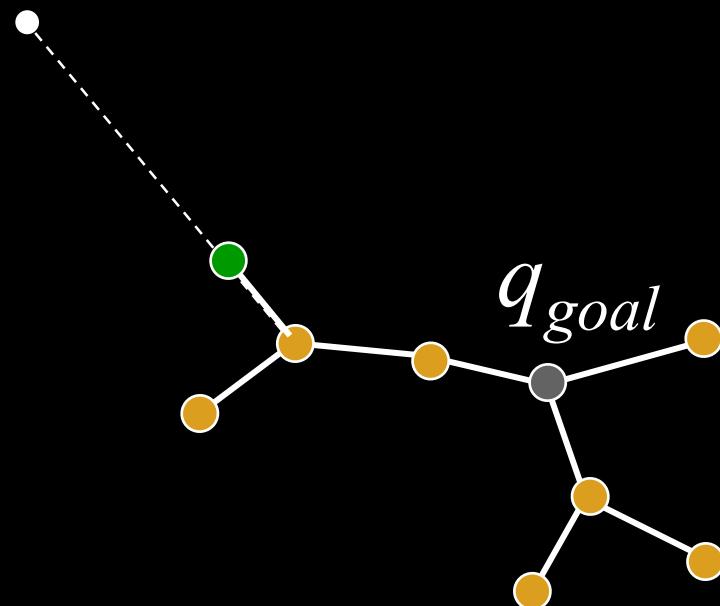


Extend

1) One tree grown using random target

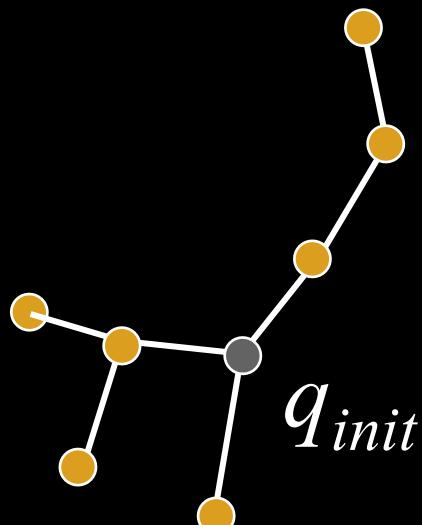


Connect

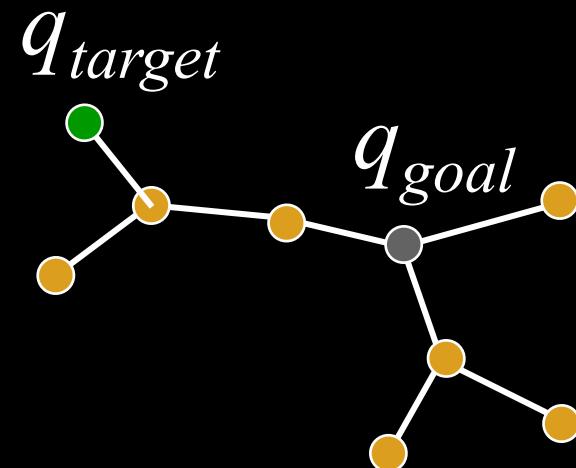


Extend

2) New node becomes target for other tree

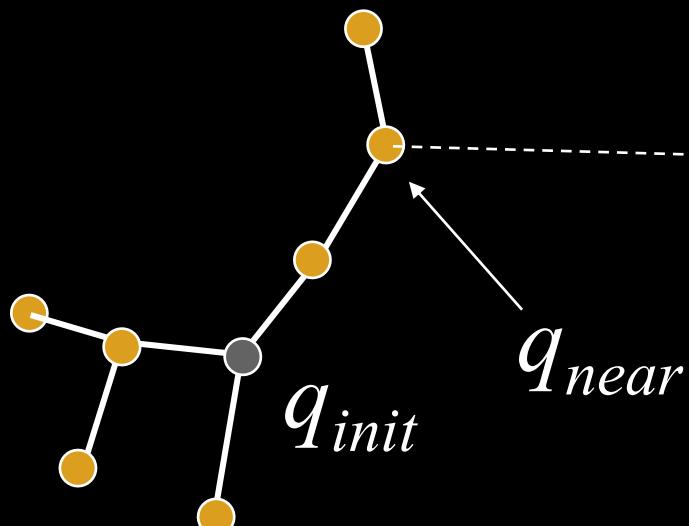


Connect

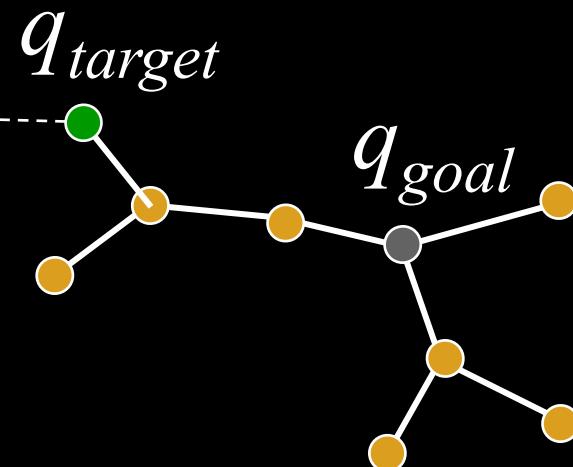


Extend

3) Calculate node “nearest” to target

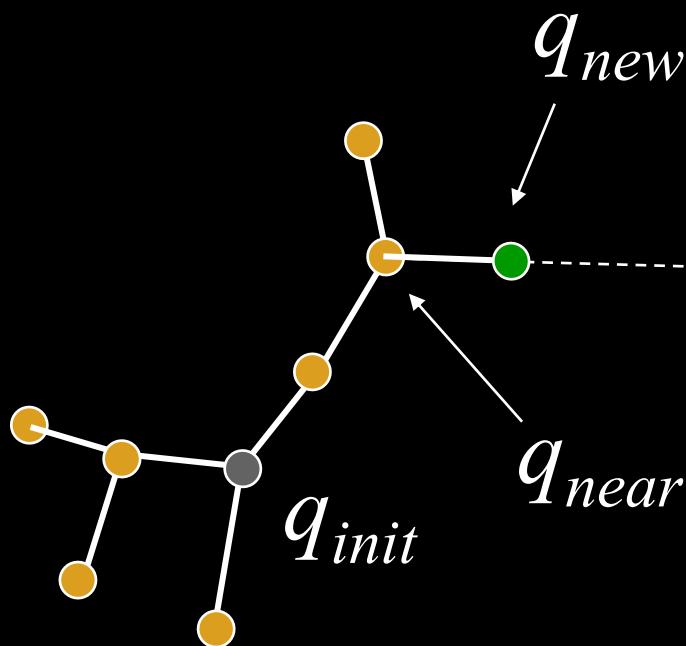


Connect

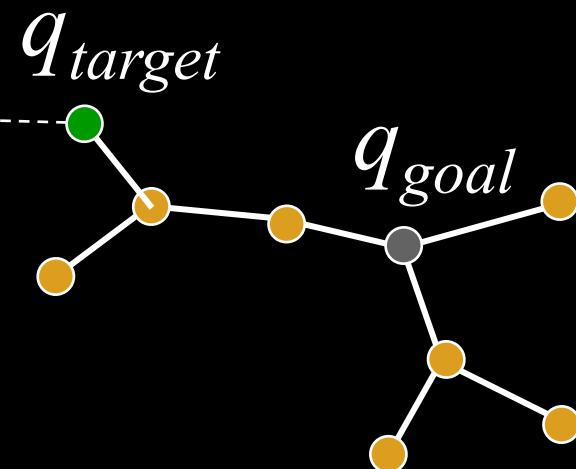


Extend

4) Try to add new collision-free branch

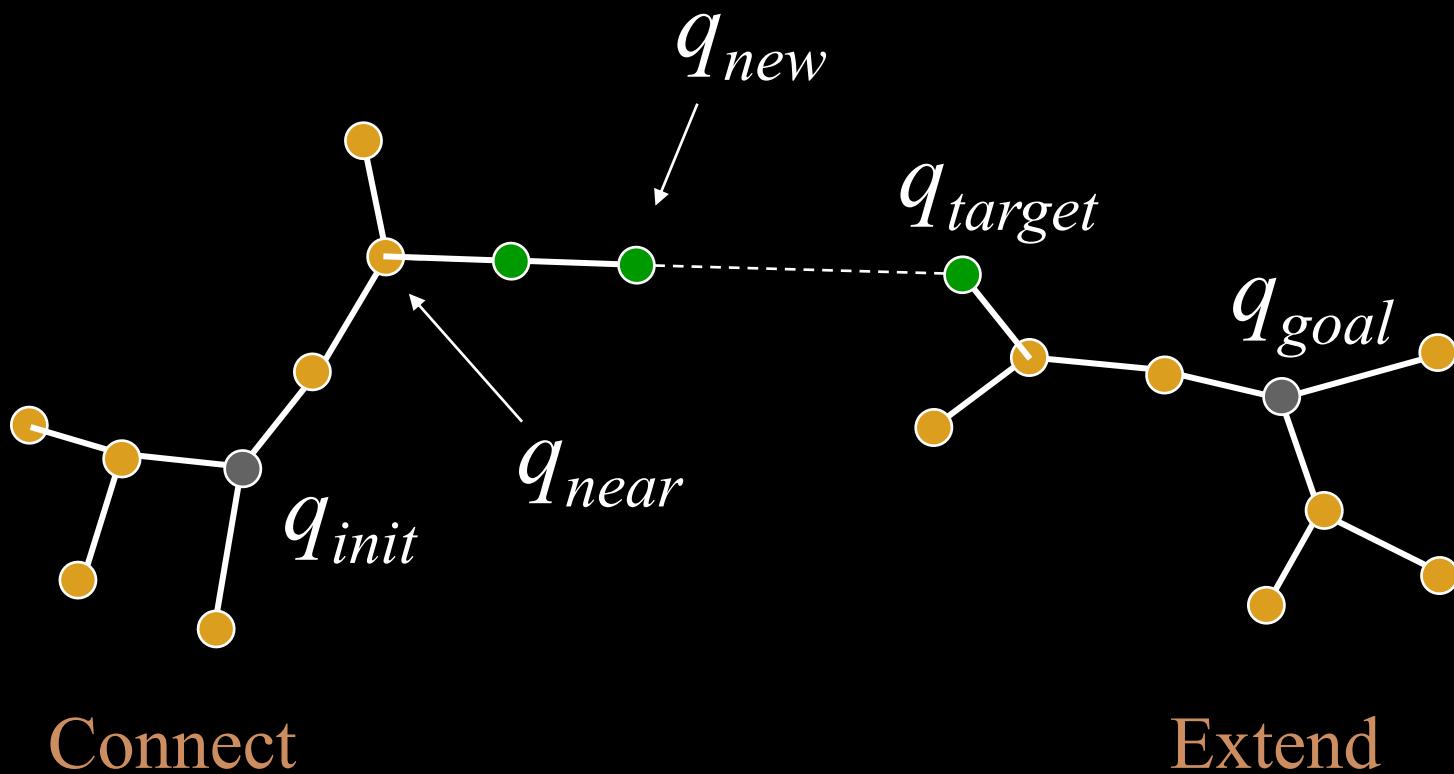


Connect

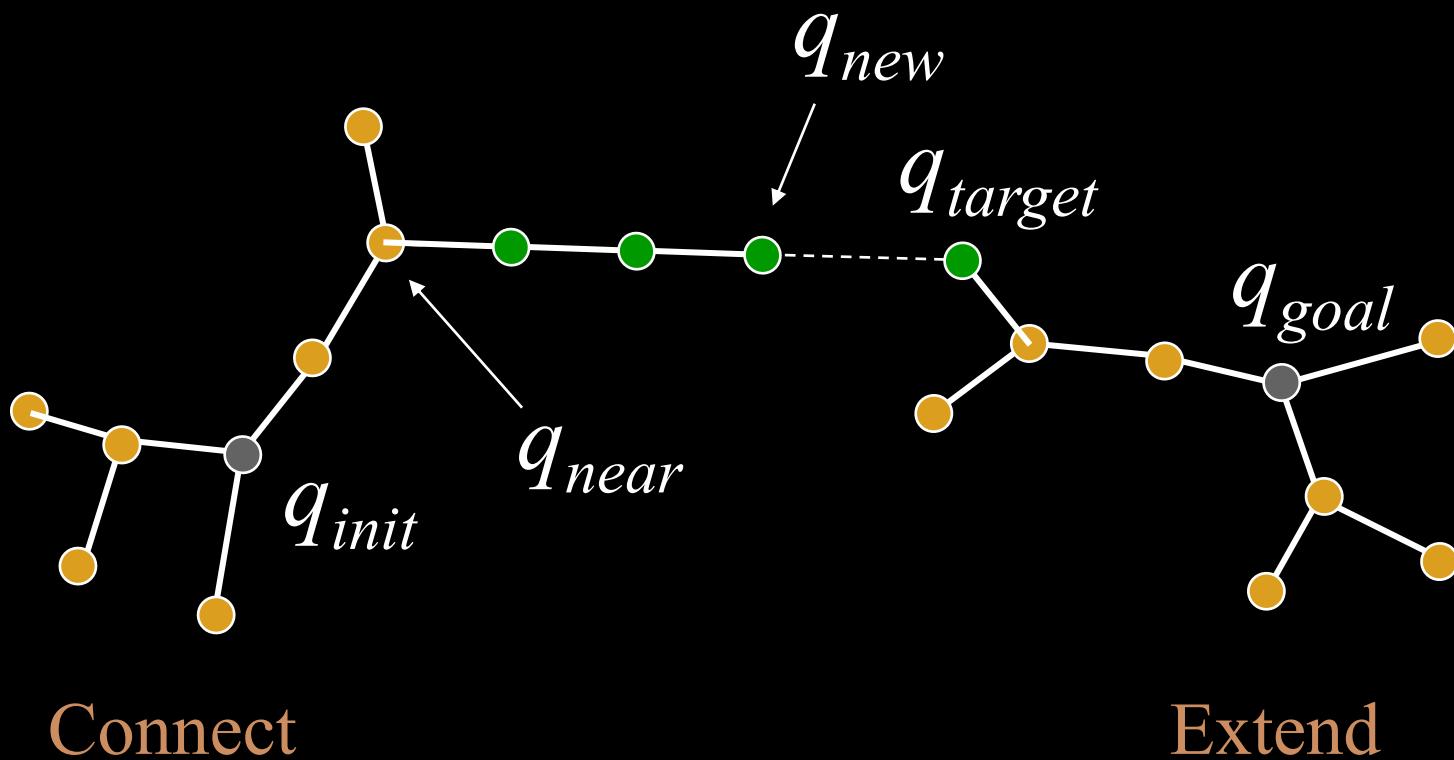


Extend

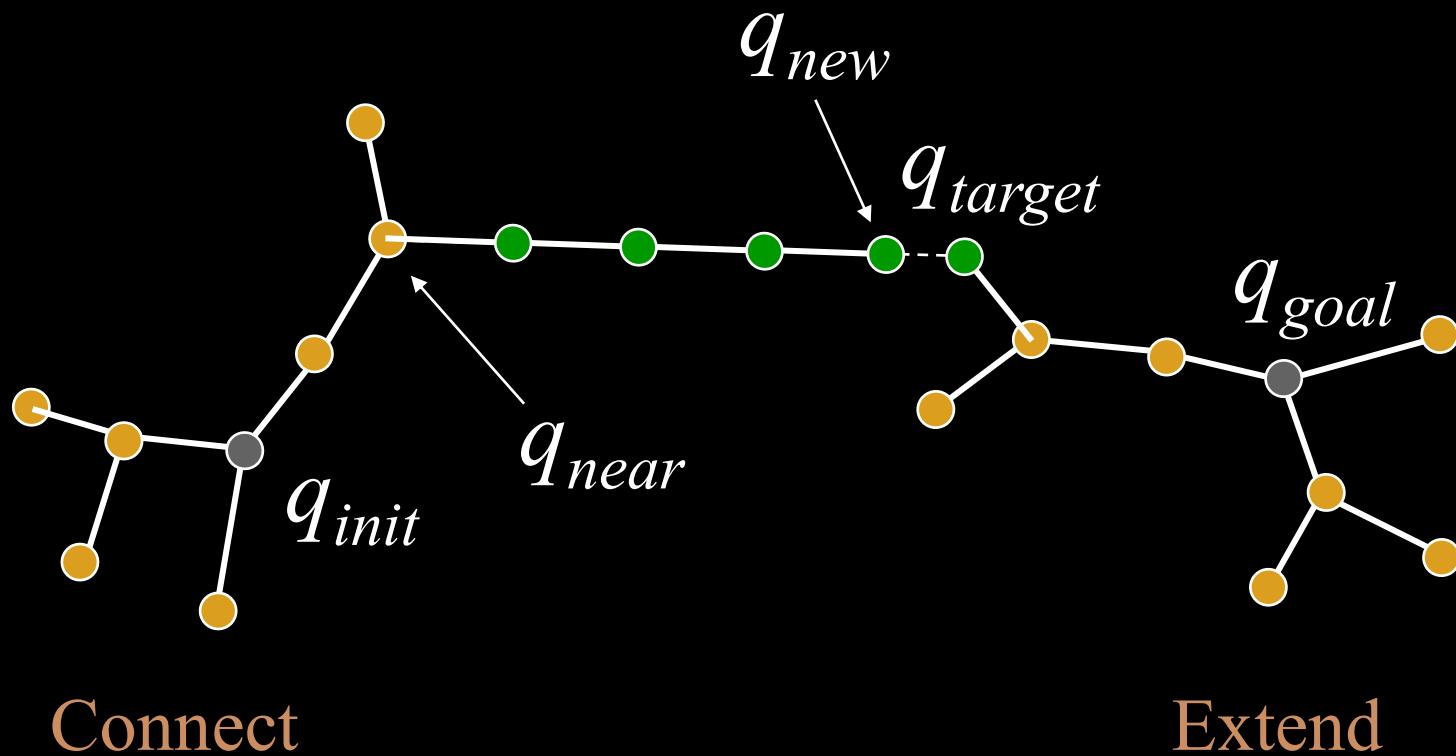
5) If successful, keep extending branch



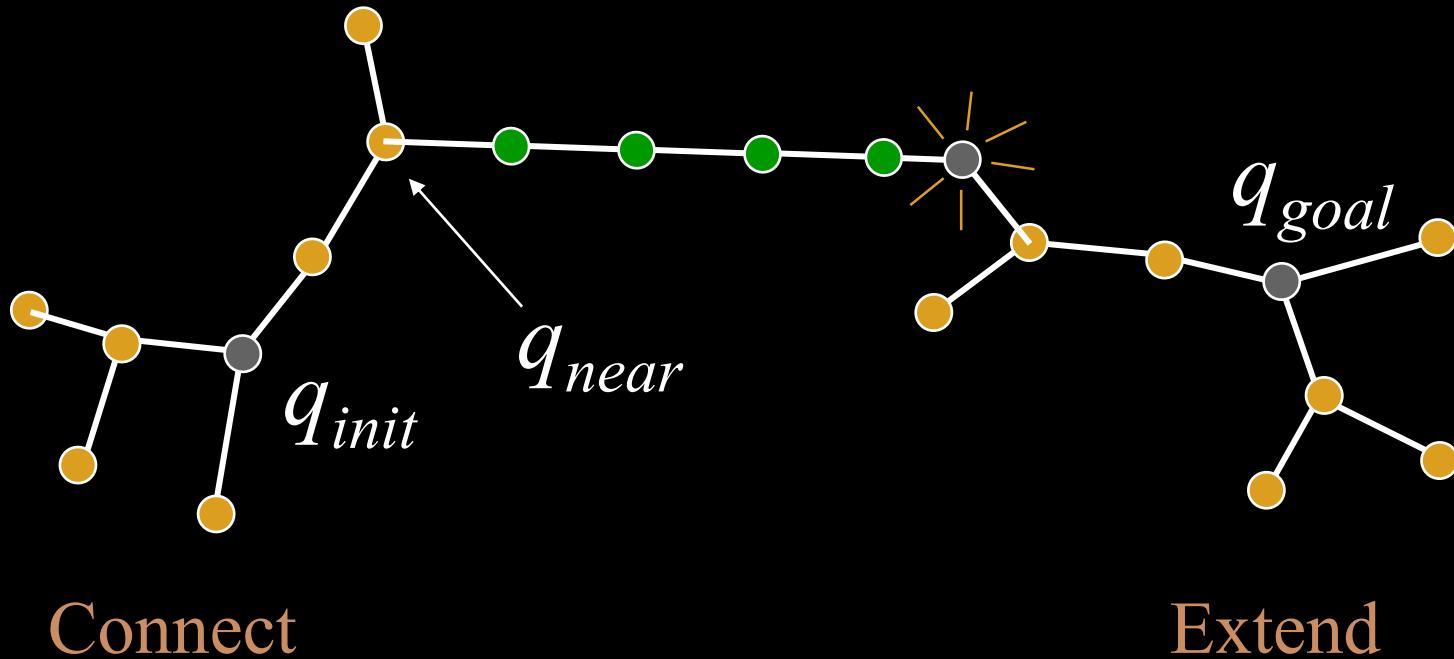
5) If successful, keep extending branch



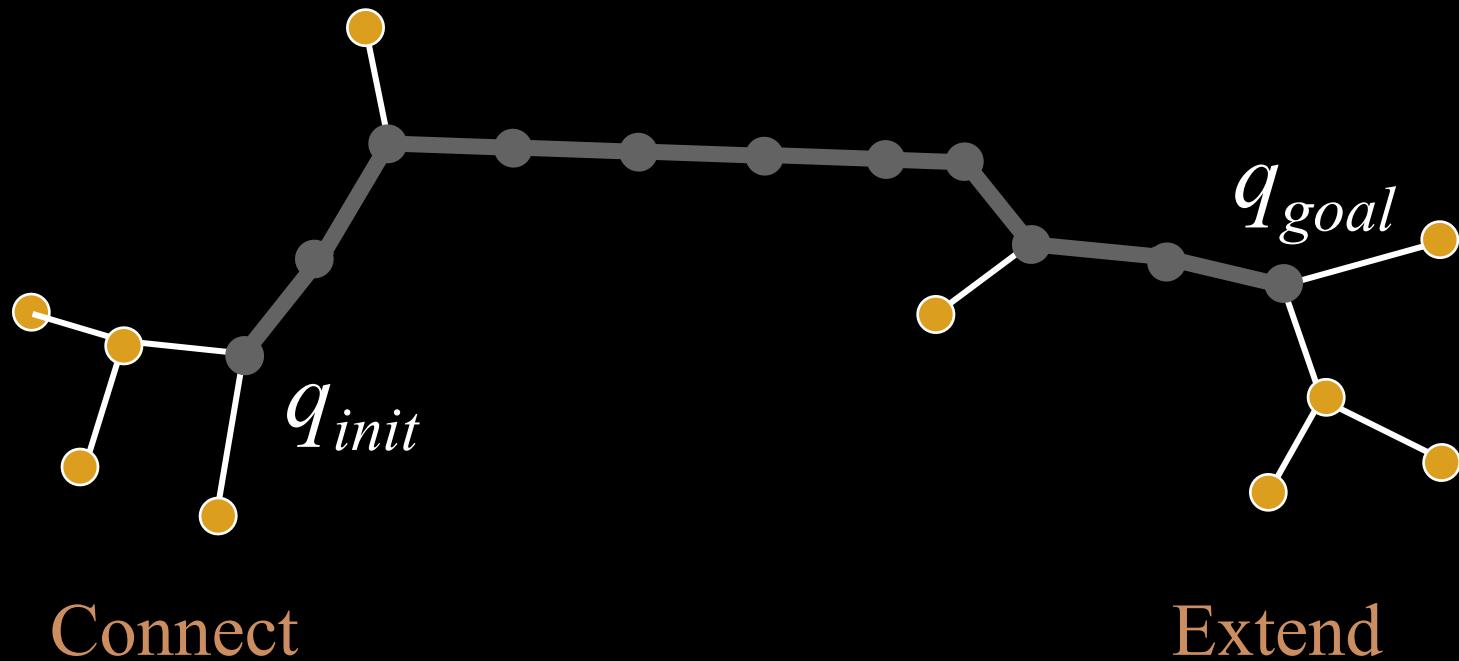
5) If successful, keep extending branch



6) Path found if branch reaches target



7) Return path connecting start and goal



Connect

Extend

Tree Swapping and Balancing

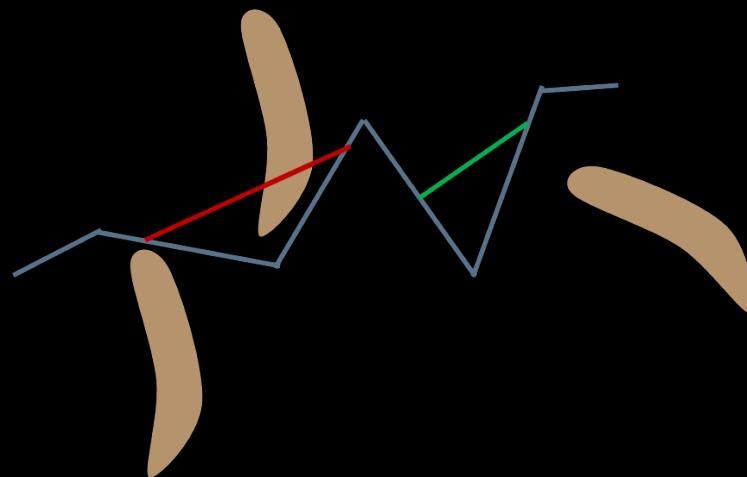
- Usually, swap trees after each iteration
- Some use Tree Balancing:

```
RDT BALANCED_BIDIRECTIONAL( $q_I, q_G$ )
1    $T_a.\text{init}(q_I); T_b.\text{init}(q_G);$ 
2   for  $i = 1$  to  $K$  do
3        $q_n \leftarrow \text{NEAREST}(S_a, \alpha(i));$ 
4        $q_s \leftarrow \text{STOPPING-CONFIGURATION}(q_n, \alpha(i));$ 
5       if  $q_s \neq q_n$  then
6            $T_a.\text{add\_vertex}(q_s);$ 
7            $T_a.\text{add\_edge}(q_n, q_s);$ 
8            $q'_n \leftarrow \text{NEAREST}(S_b, q_s);$ 
9            $q'_s \leftarrow \text{STOPPING-CONFIGURATION}(q'_n, q_s);$ 
10          if  $q'_s \neq q'_n$  then
11               $T_b.\text{add\_vertex}(q'_s);$ 
12               $T_b.\text{add\_edge}(q'_n, q'_s);$ 
13          if  $q'_s = q_s$  then return SOLUTION;
14      if  $|T_b| > |T_a|$  then SWAP( $T_a, T_b$ ); 
15  return FAILURE
```

- What is a situation where this would *help* performance?
- What is a situation where this would *hurt* performance?

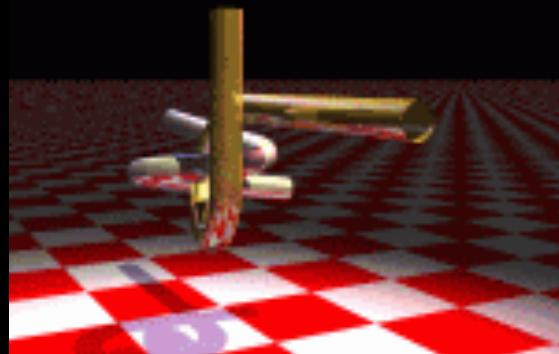
Path Smoothing/Optimization

- RRTs produce notoriously bad paths
 - Not surprising since no consideration of path quality
- **ALWAYS** smooth/optimize the returned path
 - Many methods exists, e.g. shortcut smoothing (from previous lecture)



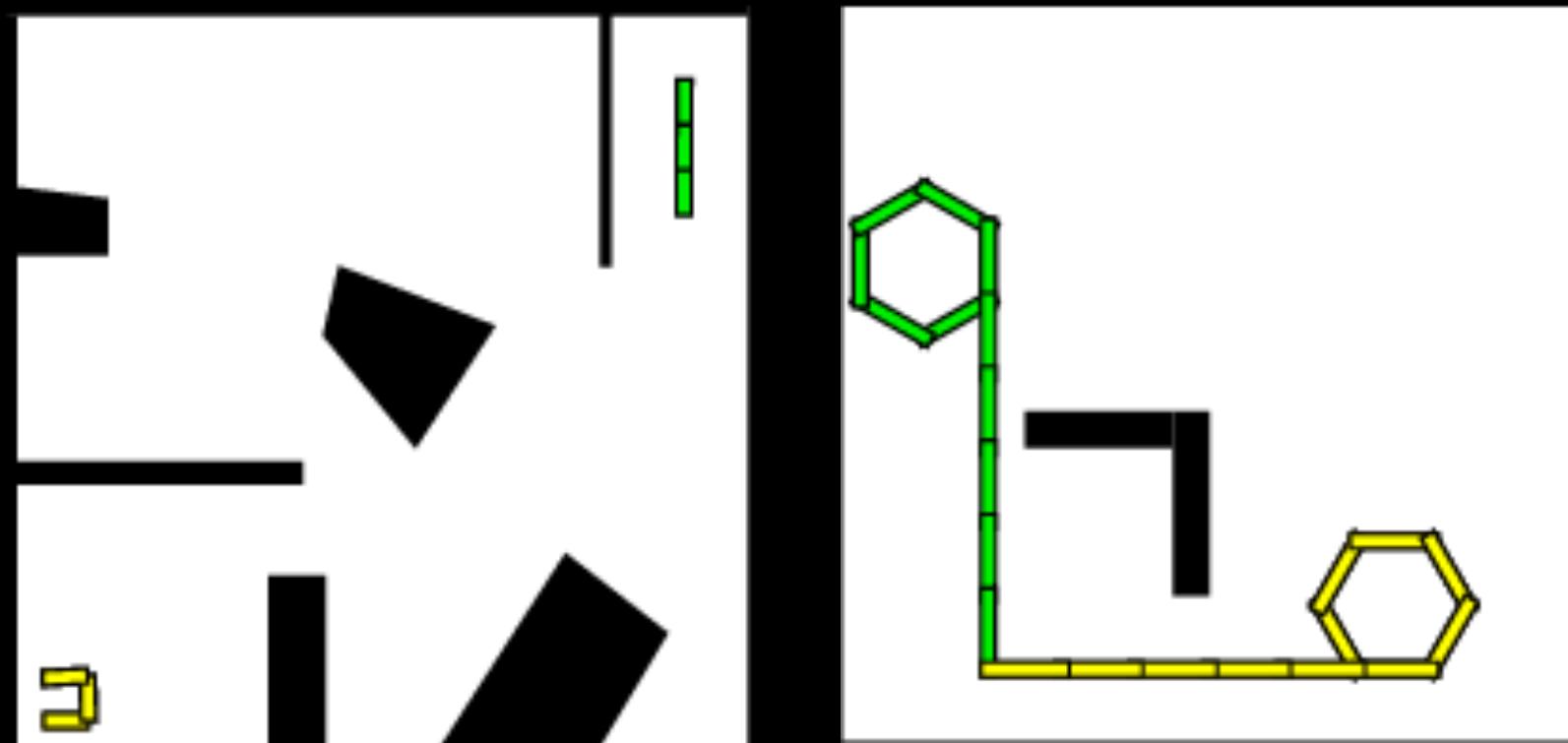
RRT Examples: The Alpha Puzzle

- VERY hard 6DOF motion planning problem (long, winding narrow passage)



- “*In 2001, it was solved by using a balanced bidirectional RRT, developed by James Kuffner and Steve LaValle. There are no special heuristics or parameters that were tuned specifically for this problem. On a current PC (circa 2003), it consistently takes a few minutes to solve*” –RRT website
- RRT became famous in large part because it was able to solve this puzzle

RRT Examples: Articulated Objects



RRT Analysis

The limiting distribution of vertices:

- **THEOREM:** \mathbf{X}_k converges to \mathbf{X} with probability 1 as time goes to infinity

\mathbf{X}_k : The RRT vertex distribution at iteration k

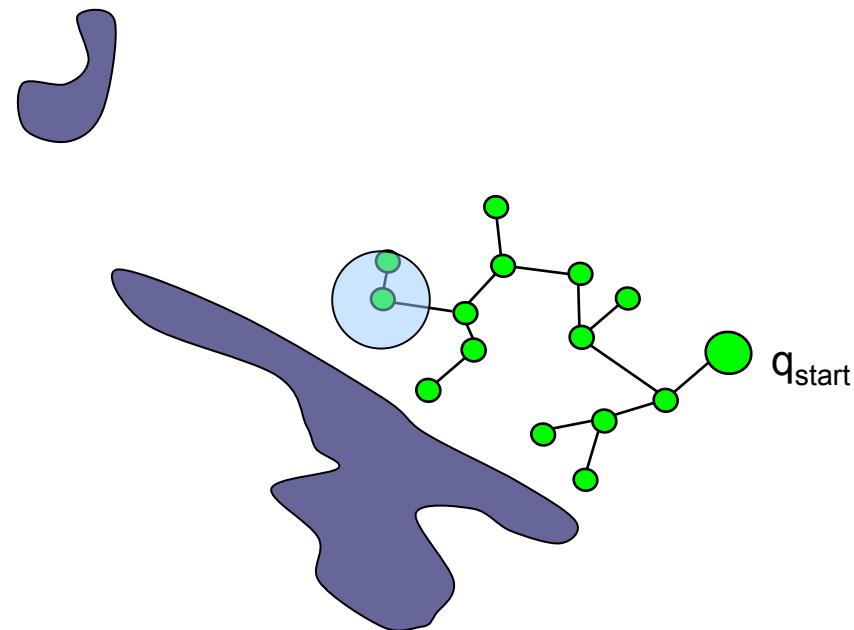
\mathbf{X} : The distribution used for generating samples

- If using uniform distribution, tree nodes converge to the free space

Probabilistic Completeness

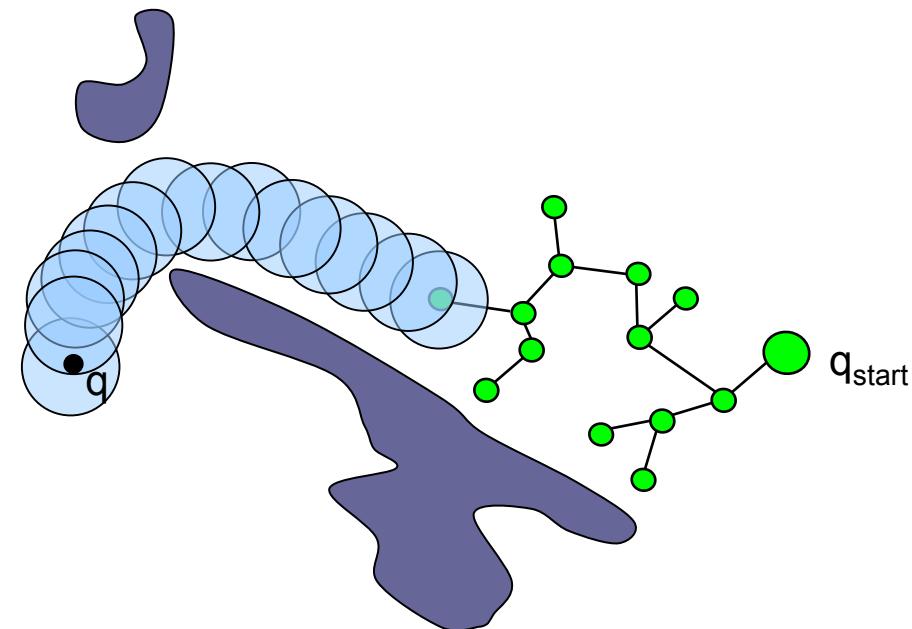
- **Definition:** A path planner is *probabilistically complete* if, given a solvable problem, the probability that the planner solves the problem goes to 1 as time goes to infinity.
- Will RRT explore the whole space?

RRT P.C. Proof



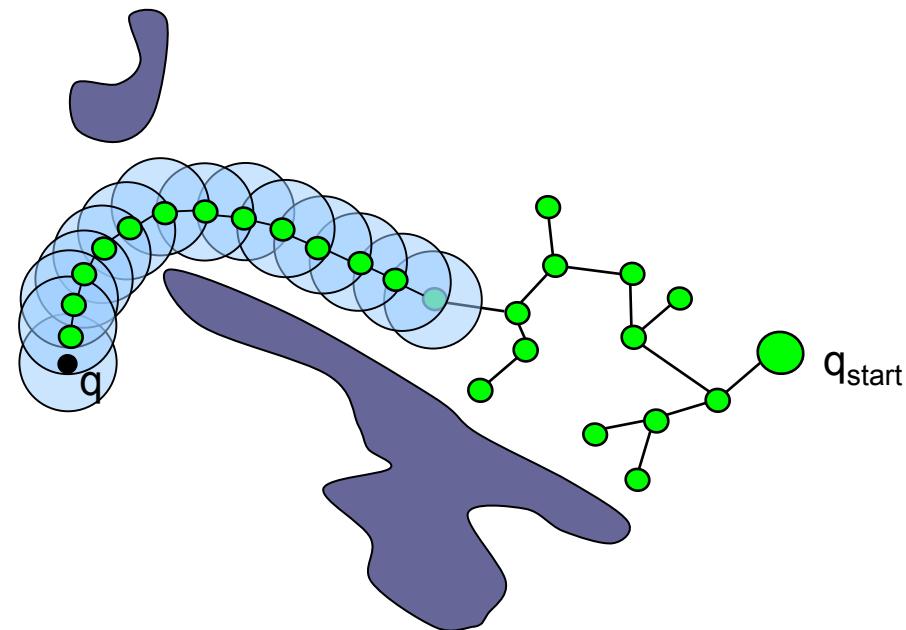
Kuffner and LaValle, ICRA, 2000

RRT P.C. Proof



Kuffner and LaValle, ICRA, 2000

RRT P.C. Proof



Kuffner and LaValle, ICRA, 2000

Probabilistic Completeness

- As the RRT reaches all of C_{free} to q_{start} , the probability that q_{rand} immediately becomes a new vertex approaches 1.
- So, is RRT probabilistically complete?

Sampling-Based Planning

- The good:
 - Provides fast *feasible* solution
 - Popular methods have few parameters
 - Works on practical problems
 - Works in high-dimensions
 - Works even with the wrong distance metric

Sampling-Based Planning

- The bad:
 - No quality guarantees on paths*
 - In practice: smooth/optimize path afterwards
 - No termination when there is no solution
 - In practice: set an arbitrary timeout
 - Probabilistic completeness is a weak property
 - Completeness in high-dimensions is impractical

*Asymptotically-optimal sampling-based planners can make some guarantees

Homework

- Read LaValle Ch. 14-14.5
- Read “How to Read a Paper” guide (link on website)
- Make sure you’ve started HW2!