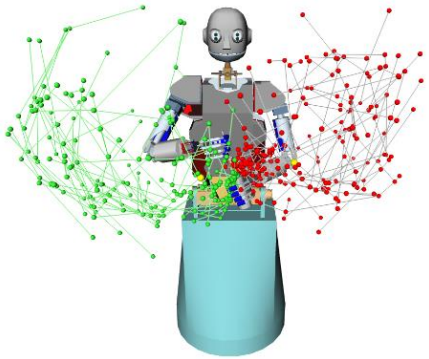
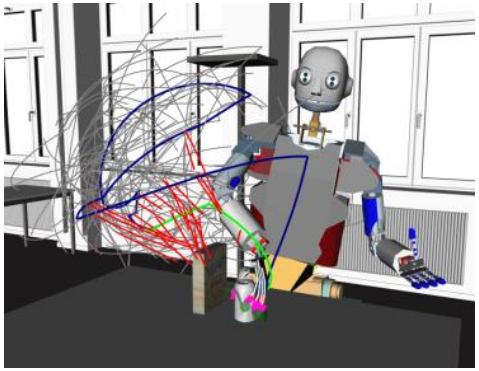
Karlsruher Institut für Technologie

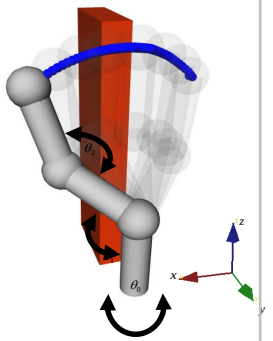
Hochperformante Humanoide Technologien

Robotics I: Introduction to Robotics
Chapter 7 – Motion Planning

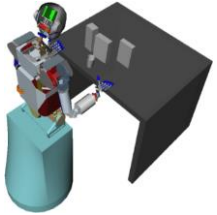
Tamim Asfour

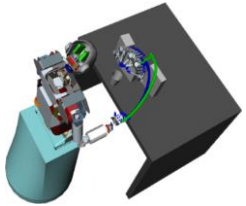
http://www.humanoids.kit.edu

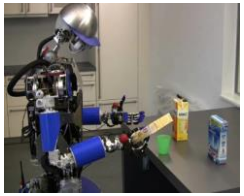





Motivation







Grasping Task → Scene Representation → Motion Planning → Sensor-based Execution

This chapter

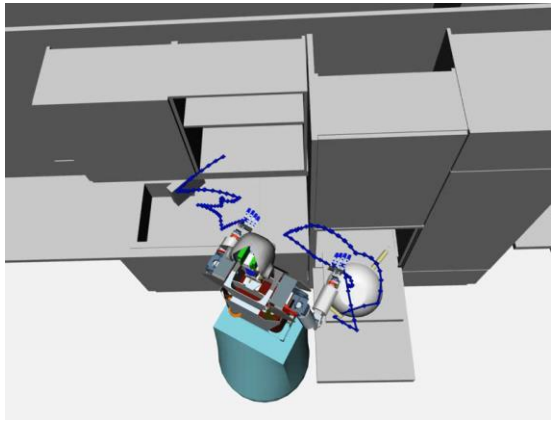
Kinematics
Continuous Path Control
Control
…

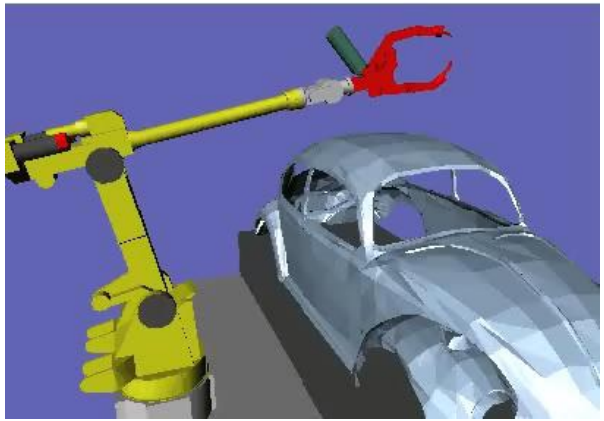
Table of Contents

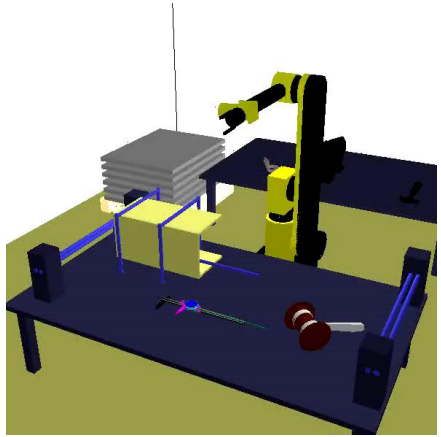
- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- Motion planning for manipulators

Motion Planning: Motivation (1)

Generation of a **collision-free** trajectory w.r.t. various **goals** and **constraints**







Motion Planning: Motivation (1)

Generation of a **collision-free** trajectory w.r.t. various **goals** and **constraints**

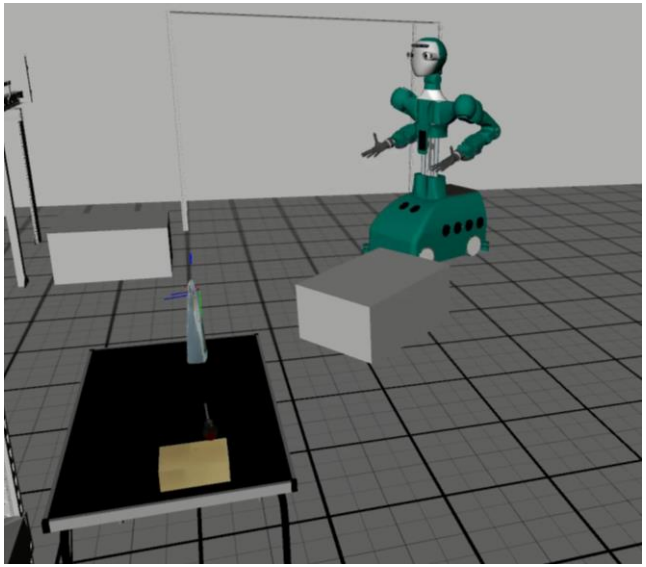


Table of Contents

- Motivation
- **Fundamentals of motion planning**
  - **Problem statement**
  - **Definitions**
  - **Concept formation**
  - **Problem categories**
- Path planning for mobile robots
- Motion planning for manipulators

Motion Planning: Problem Statement

- **Given:**
  - Configutration space C
  - Start configuration $\boldsymbol{q}_{start} \in C$
  - Goal configuration $\boldsymbol{q}_{goal} \in C$
- **Required**

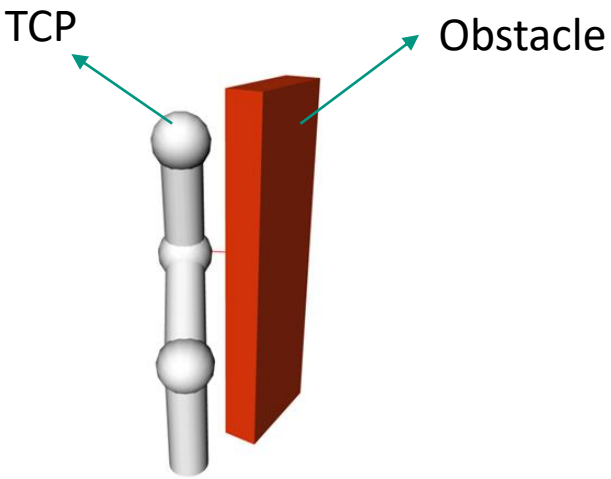
  Continuous trajectory

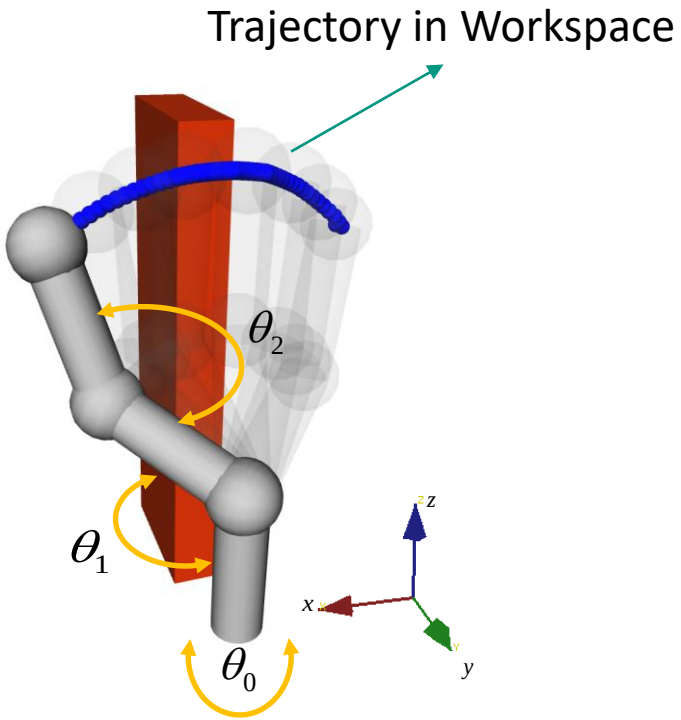
$$\tau: [0,1] \rightarrow C \text{ with } \tau(0) = \boldsymbol{q}_{start} \text{ and } \tau(1) = \boldsymbol{q}_{goal}$$

  With respect to
  - Contraints (joint limits, maximal acceleration, …)
  - Quality criteria (duration, energy, distance to obstacles, smoothness of the trajectory, …)
  - Additional and boundary contraints (Upright position of end-effector, …)

Workspace

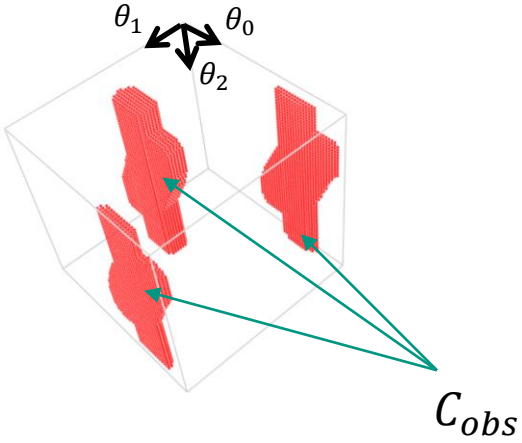
- W: cartesian space $\mathbb{R}^6$
- Tool Center Point (TCP)

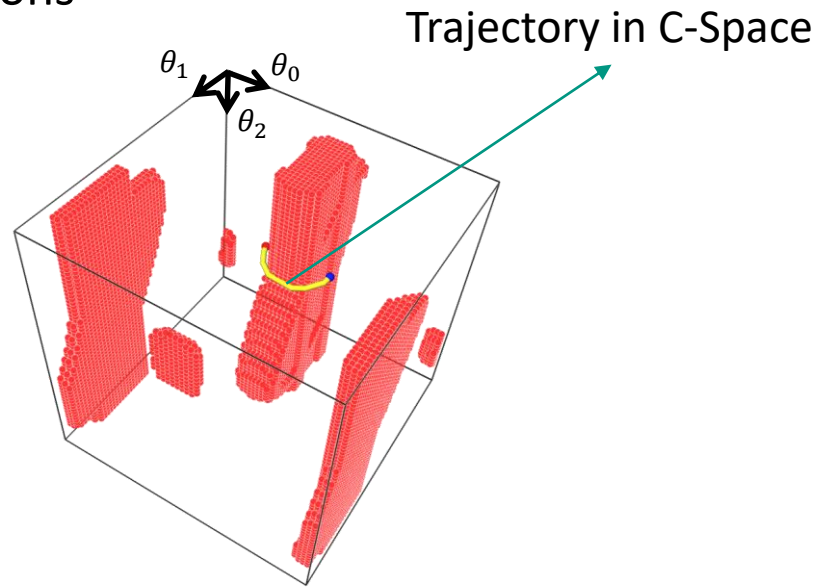




Configuration Space (C-Space)

- C: n-dimensional configuration space (C-Space)
- C_{free}: all collision-free configurations
- C_{obs}: all configurations resulting in collisions
- $C = C_{free} \cup C_{obs}$





Memory Consideration (1)

- **Is it possible to store configuration and obstacle space?**

- Assumptions:
  - Resolution of rotation joints: $1°$
  - Resolution of translatory joints : $1\ \mathrm{cm}$
  - Storage space per configuration: $1\ \mathrm{bit}$ (0: no obstacle, 1: obstacle)

- **Example:** 7-DoF robot arm (7 rotation joints)

$$360^7\ \mathrm{bit} \approx 7{,}8 \cdot 10^{17}\ \mathrm{bit} \approx 87\ \mathrm{Petabyte} = 87 \times 10^{15}\mathrm{Byte}$$

Memory Consideration (2)

- **Is it possible to only store the obstacle space?**

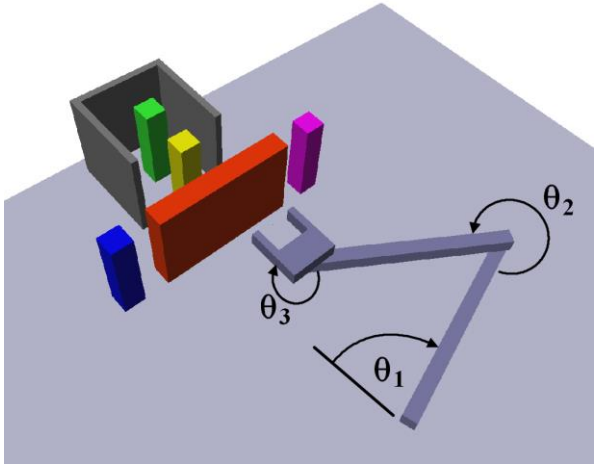
- Assumptions:
  - Only $0{,}1\ \%$ of the configuration space are in collision
  - We have to store each configuration
  - Robot with N DoF: Store N 32-bit floating point numbers per configuration (4 Byte)

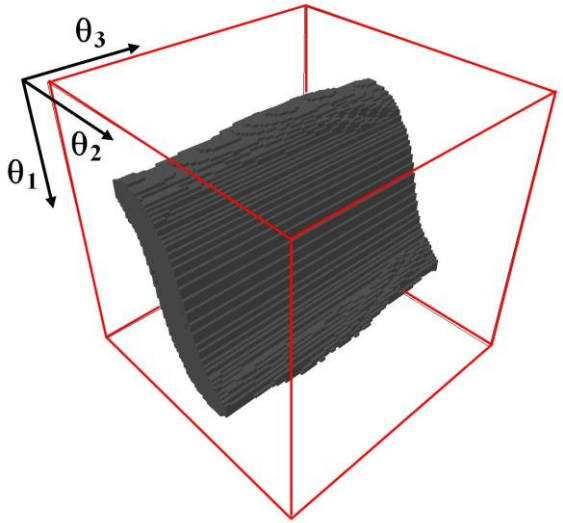
- Example: 7-DoF robot arm (7 rotation joints)

$$\frac{0{,}1}{100} \cdot\ 360^7 \cdot 7 \cdot 4\ \text{Byte} \approx 2{,}2 \cdot 10^{16}\ \text{Byte} = 19{,}5\ \text{Petabyte}$$

Workspace vs. Configuration Space

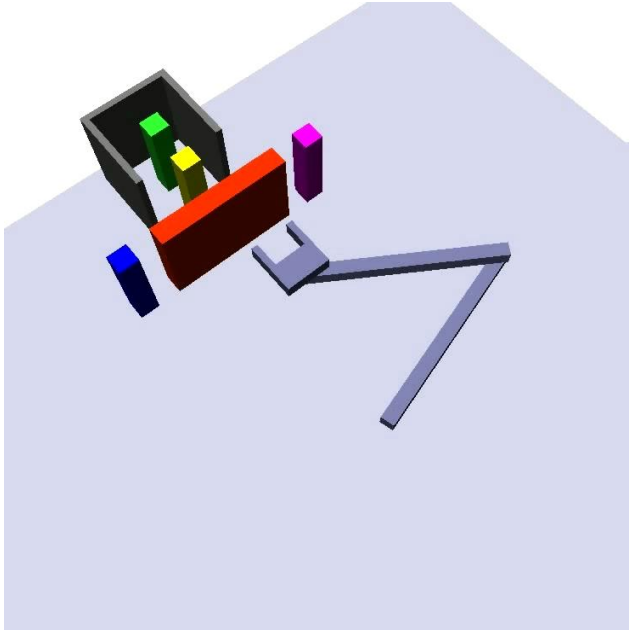
- Example: 3 DoF robot arm





Configuration and Obstacle Space

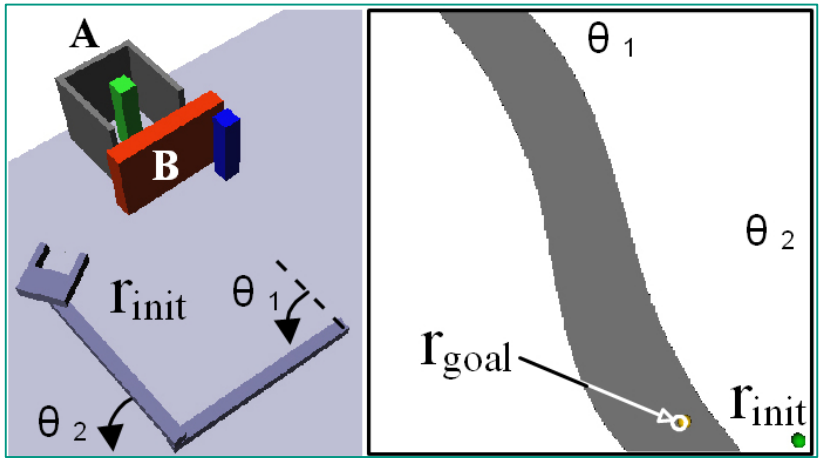
- C_{free} and C_{obs} can change during execution

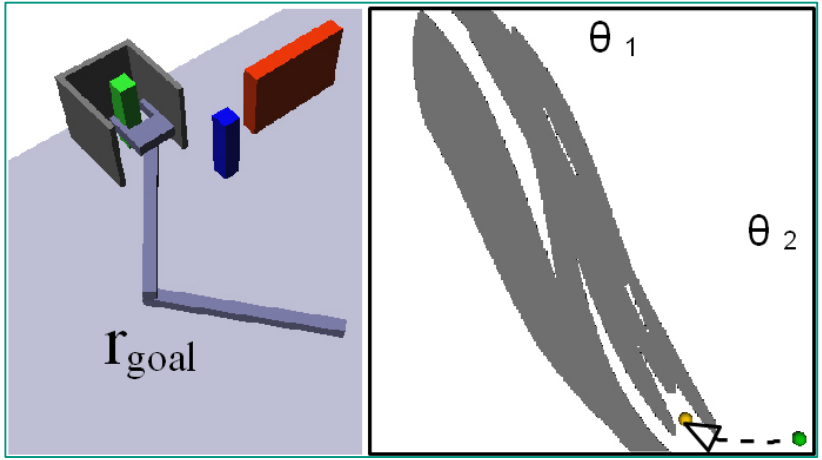


Goal: grasp the green object

Configuration and Obstacle Space

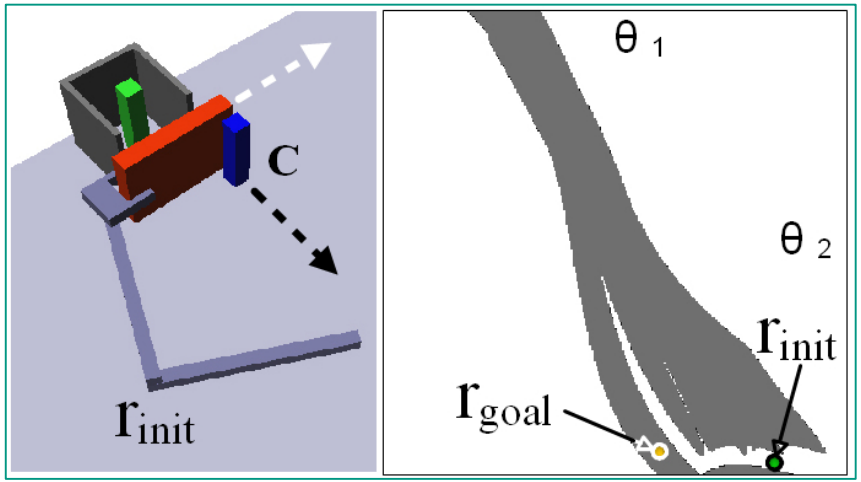
- How do changes in the workspace affect the C-Space?

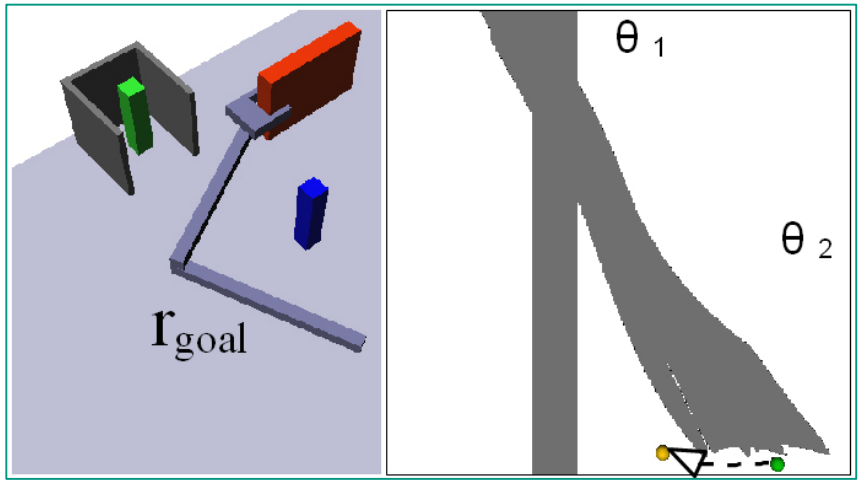




Configuration and Obstacle Space

- How do changes in the workspace affect C-Space?





Motion Planning: Definitions (1)

- **Configuration**

  A **configuration** $\boldsymbol{q} \in C$ describes the robot state as a **joint angles vector** in joint space.

- **Configuration Space**

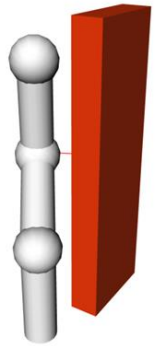
  The **configuration space** C of a robot R is the space of all possible joint angle configurations of R.

- Motions can also be planned in workspace (mobile robots & drones). In this case, the configuration is the position and orientation of the robot

Motion Planning: Definitions (2)

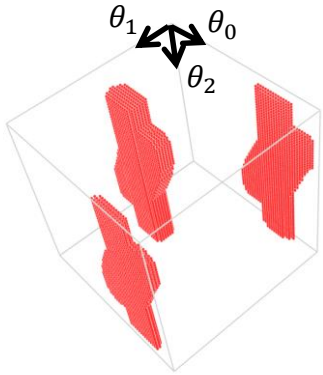
- **Obstacle in workspace**

  An **obstacle in workspace** O is the space occupied by an object in workspace.



- **Obstacle in C-Space**

  An **obstacle in configuration space** C_O is the set of points in configuration space C that lead to a collision with the obstacle O



- **Obstacle space**

  The **obstacle space** C_{obs} is the set of all obstacles in configuration space

$$C_{obs} = \bigcup_i C_{H_i}$$

Motion Planning: Definitions (3)

- The **free space** is the set of all points in C that do not lie in obstacle space C_{obs}

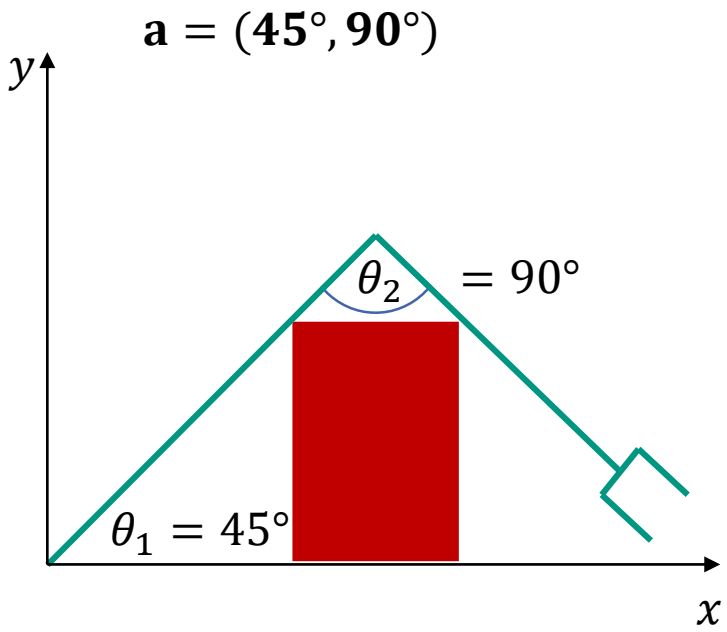
$$C_{free} = \{\boldsymbol{q} \in C \mid \boldsymbol{q} \notin C_{obs}\} = C \setminus C_{obs}$$

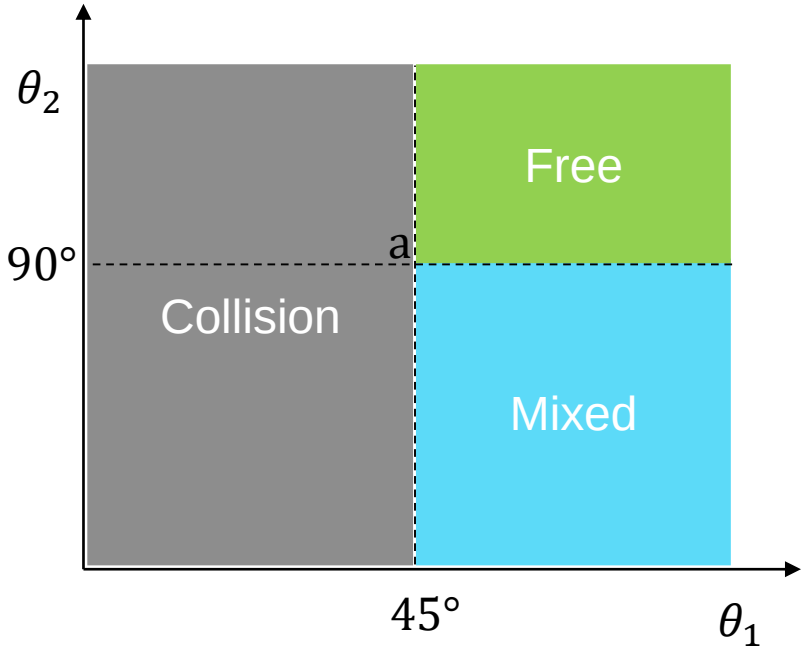
- Computation cost for calculation the free space C_{free}: $O(m^n)$
  - n: Degrees of freedom of the robot
  - m: Number of obstacles

- C_{free} cannot be efficiently calculated for complex kinematics

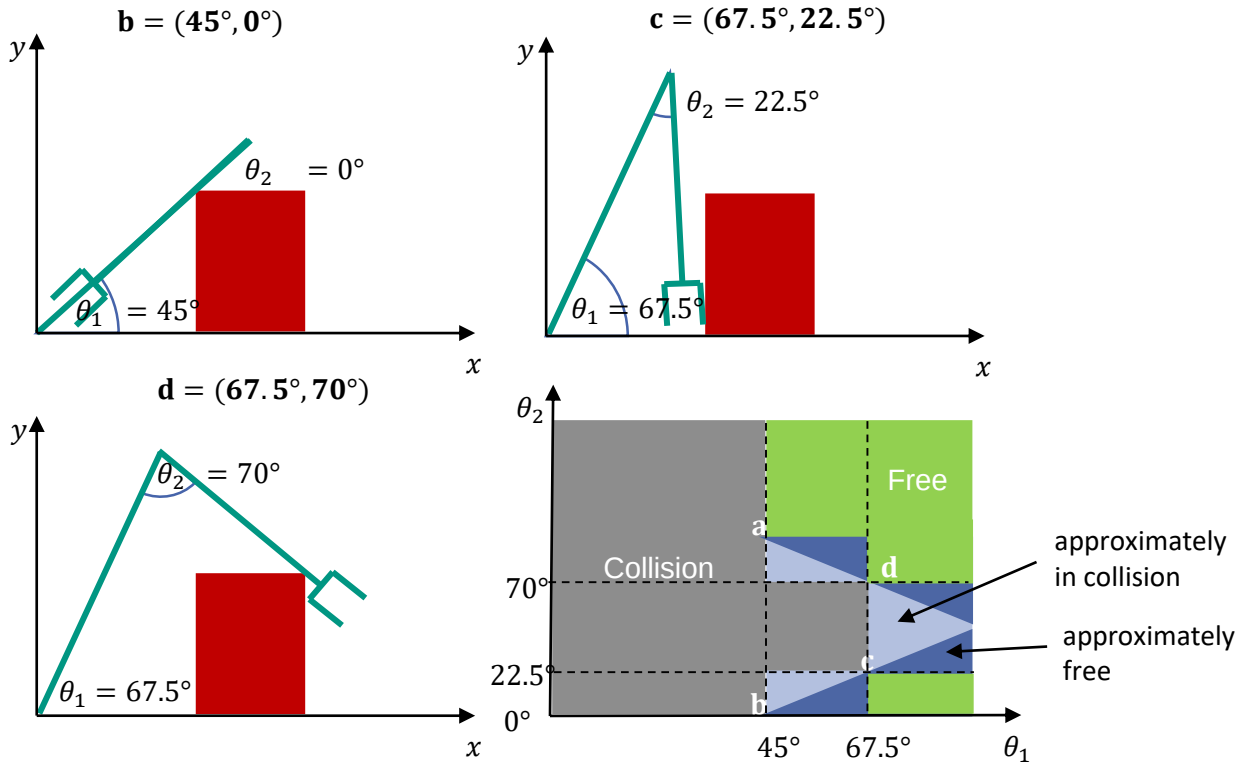
- Use of **approximations** for simplified representation of C_{free}

Approximation of Free Space (1)



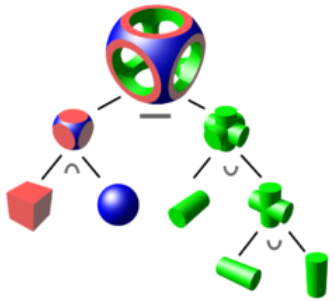


Approximation of Free Space (2)



Motion Planning: Definitions (1)

- **Environment model**
  - **Exact**: e.g. CSG (Constructed Solid Geometry)
    - Represent complex geometry by combining basic primitives (cubes, cylinders, spheres, rings, pyramids, ...) with logical operators.



wiki

  - **Approximated**: describe the environment through approximation (boxes, genaralized cylinders, polyhedrons, ...)

Motion Planning: Definitions (2)

- **Path planning**
  - Rigid object (e.g. mobile robot, autonomous vehicle)
  - 2d problem (position: x, y)
  - 3d problem (position: x, y; rotation: α) → **Piano Mover's Problem**
- **Motion planning**
  - Multi-body systems (e.g. robot arms, multi-robot systems)
  - High-dimensional problem
- **Constraints**
  - Global contraints: Limit the vaild configuration space, e.g. upright end-effector positions, minimal motor currents, etc.
  - Local constraints: Restrict transitions between configurations, e.g.
    - Non-holonomic vehicles cannot move sideways or turn on the spot
    - Staying within max. velocities or accelerations

Motion Planning: Definitions (3)

- **Complexity**
  In general motion planning is a **PSPACE-complete** problem
  - Deterministic Turing maschines can solve it only with polynomial (memory) space
  - Lower and upper bounds of complexityt NP ⊆ PSCAPE ⊆ EXPTIME, i.e. **NP-hard problem**

Motion Planning: Definitions (4)

- **Complete algorithm**
  A complete algorithm finds at **least one solution** for a problem or determines in finite time that no solution exists.
- **Randomized algorithm**
  A randomized algorithms usee **randomized values**, to speed up processing they often make use of **heuristics**.
- **Resolution complete algorithm**
  If a randomized algorithm is complete for a **discrete problem**, it is called resolution complete.

Motion Planning: Definitions (5)

- **Probabilistically complete algorithm**
  A probabilistically complete algorithm finds **at least one solution if it exists**.
  The probability of finding a solution converges to one over time.

  However, a probabilistically complete algorithm **cannot determine whether no solution exists**

Motion Planning: Problem Classes (1)

- **Class a**

  Known: complete world model
  complete set of constraints

  Required: collision-free trajectory from start to goal state

- **Class b**

  Known: incomplete world model
  incomplete set of constraints

  Required: collision-free trajectory from start to goal state

  Problem: collision with unknown objects

Motion Planning: Problem Classes (2)

- **Class c**
  - **Known:** time-variant world model (moving obstacles)
  - **Required:** collision-free trajectory from start to goal state
  - **Problem:** changing obstacles in time and space
- **Class d**
  - **Known:** time-variant world model
  - **Required:** trajectory to moving goal (**rendezvous problem**)
  - **Problem:** changing goal state in time and space

交汇问题

- **Class e**
  - **Known:** no world model
  - **Required:** collision-free trajectory from start to goal state
  - **Problem:** Mapping (creation of world model)

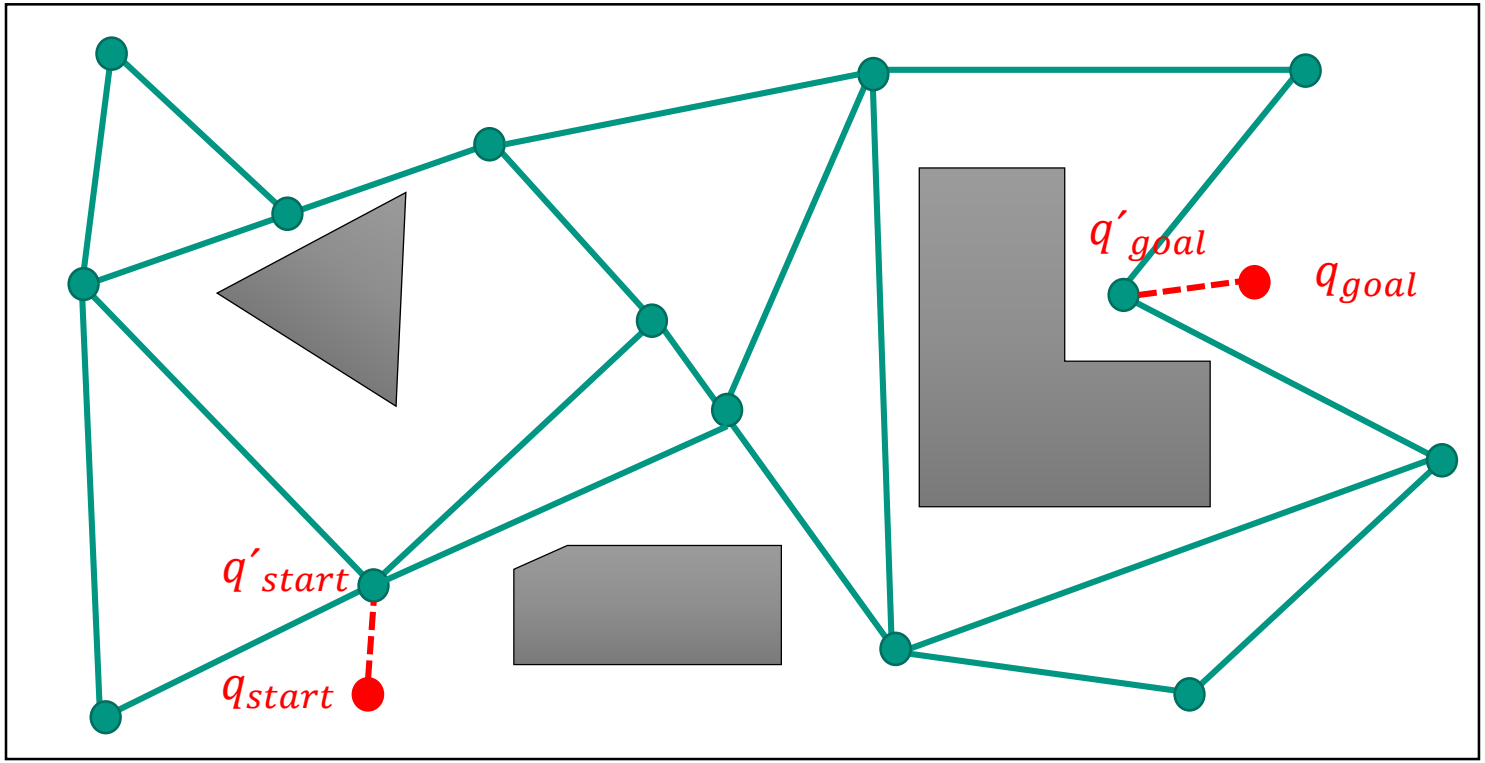
Table of Contents

- Motivation
- Fundamentals of motion planning
- **Path planning for mobile robots**
  - **Graph-based**
  - Potential fields
- Motion planning for manipulators

Path Planning for Mobile Robots

- **Given:**
  - 2D world model (e.g. road map)
  - Start and goal position $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{ziel}$
- **Required:**
  - (optimal) path from $\boldsymbol{q}_{start}$ to $\boldsymbol{q}_{ziel}$
- **Approach:**
  - **Construct** a path net (graph) W in C_{free}
  - Map $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{ziel}$ to the nearest vertices $\boldsymbol{q}'_{start}$ and $\boldsymbol{q}'_{ziel}$ in W (nearest neighbour)
  - **Search** for a path from $\boldsymbol{q}'_{start}$ to $\boldsymbol{q}'_{ziel}$ in W
  - Search for a path between $\boldsymbol{q}_{start}$ and $\boldsymbol{q}'_{start}$, and between $\boldsymbol{q}'_{ziel}$ and $\boldsymbol{q}_{ziel}$

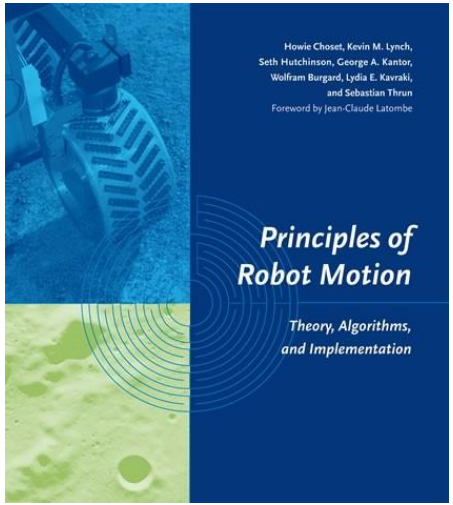
Approach – Visualization



Path Planning for Mobile Robots

Two Steps

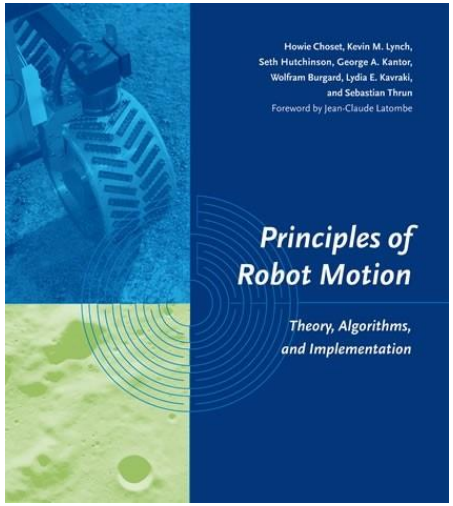
1. Generation of the graph W
2. Search in W



Path Planning for Mobile Robots

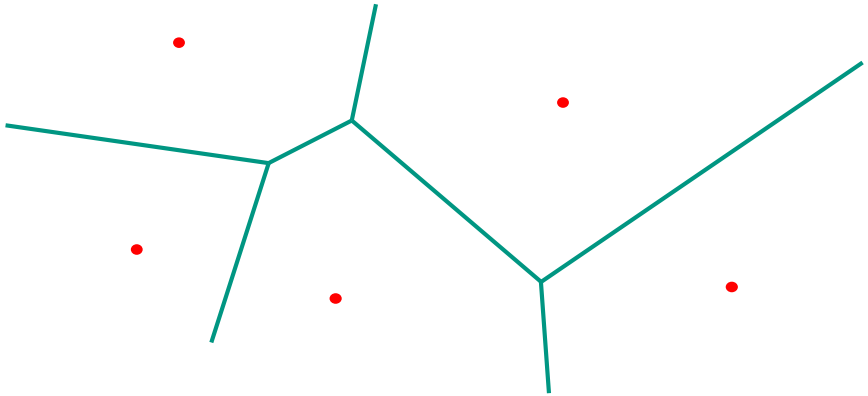
1. **Generation of the graph W**
   - Retraction methods, e.g. Voronoi Diagrams
   - Visibility Graph
   - Cell Decomposition

2. Search in W
   - Tree search
   - A*



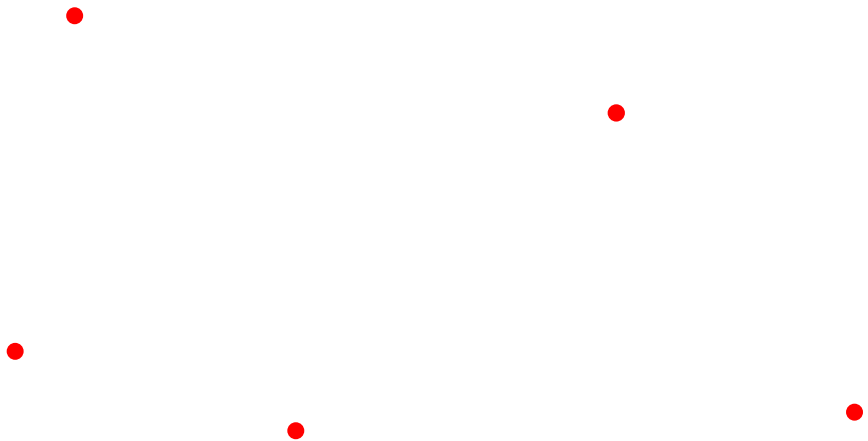
Voronoi Diagram

- Voronoi Diagram: **Partition of a space** in regions based on given points or areas that describe obstacles.
- A **region** of the Voronoi diagramm is defined as the **set of points**, whose distance to an obstacle is **less** than **to all other obstacles**.
- All **points on the boundary** between two regions of have the **same distance** to their own and to their **neighbouring obstacles**.



Voronoi Diagrams: Construction (1)

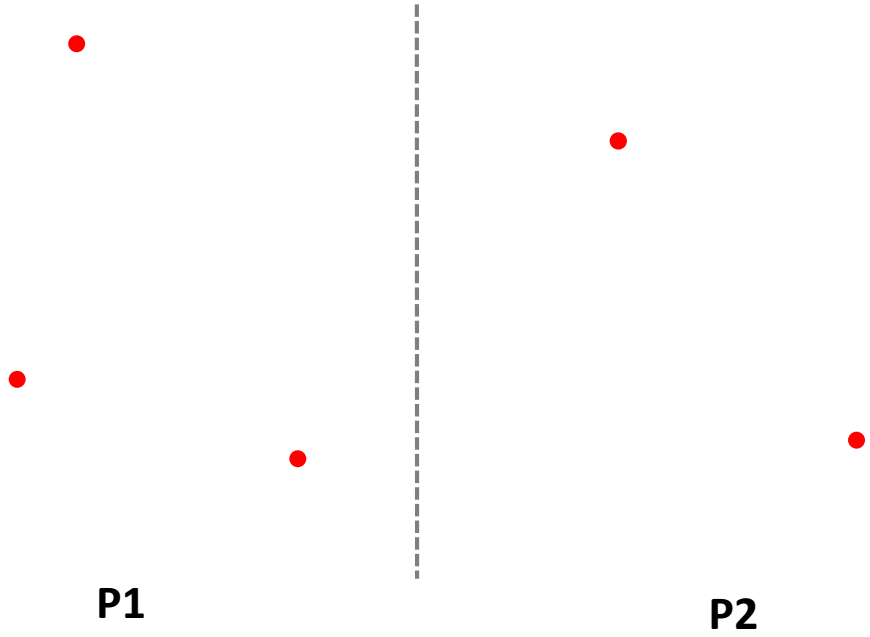
- Given **set of points P (obstacles)**



P

Voronoi Diagrams: Construction (2)

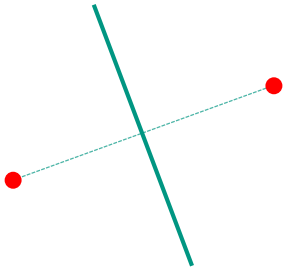
- **Divid P** into two approximately **equal subsets P1** and **P2**



Voronoi Diagrams: Construction (3)

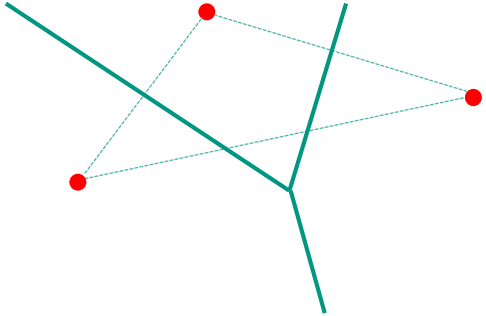
- By **recursively subdividing** the set of points, the creation of the Voronoi diagramm can be reduced to **two cases**.

Case 1: 2 points



the **perpendicular bisector** forms the Voronoi diagram

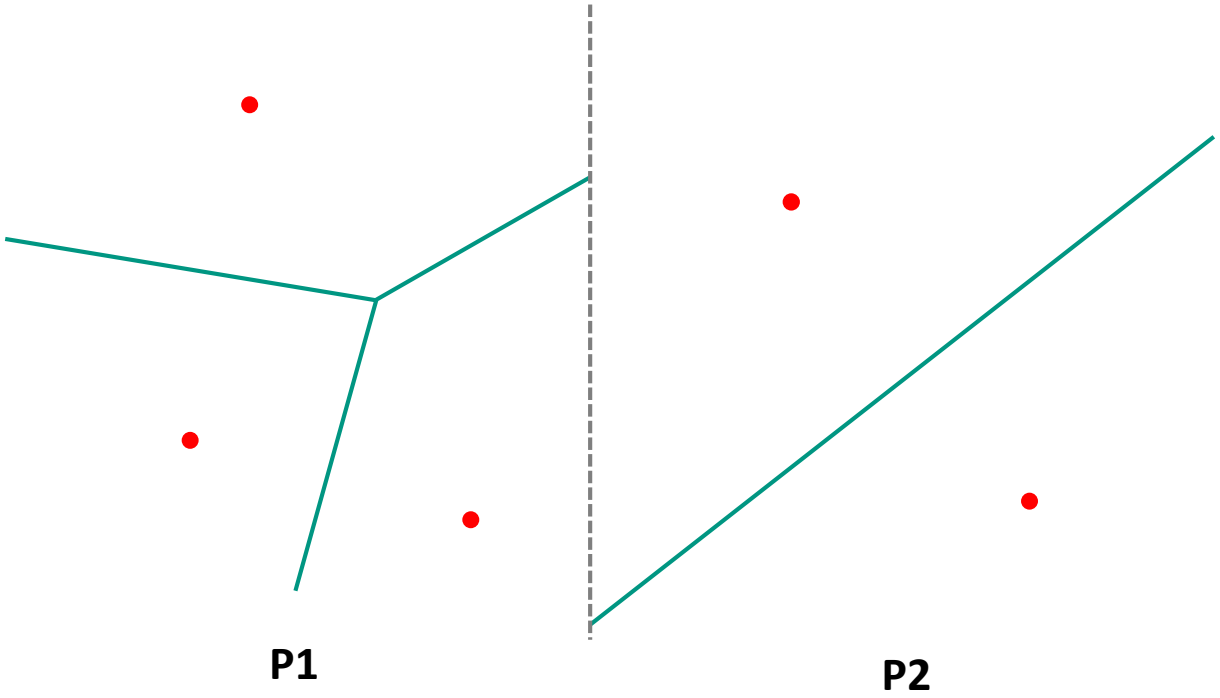
Case 2: 3 points



the **perpendicular bisector of all paired points** are cut off at the **common intersection point**

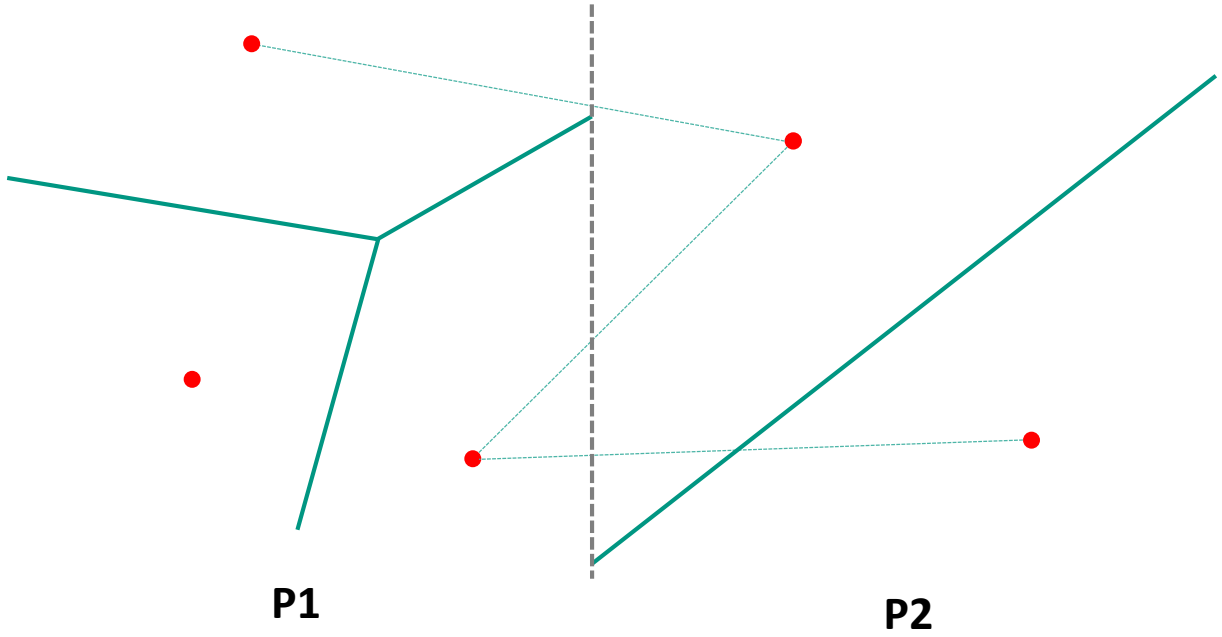
Voronoi Diagrams: Construction (4)

- Construct **Voronoi diagrams** for **P1 and P2**



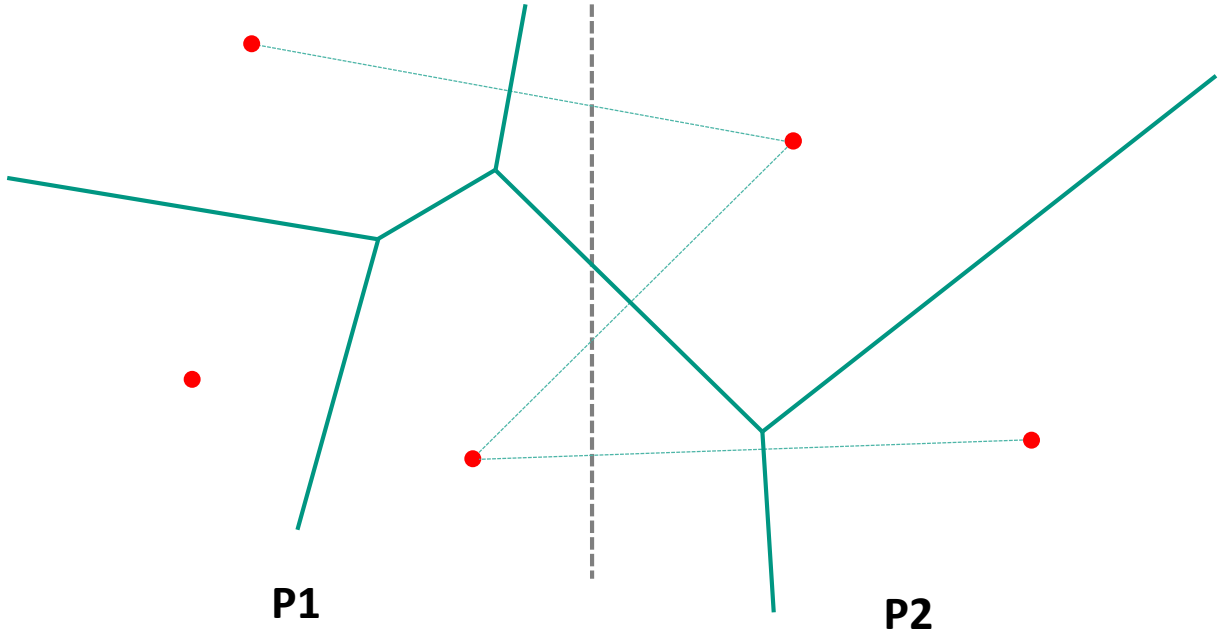
Voronoi Diagrams: Construction (5)

- **Merge** the Voronoi diagrams of **P1** and **P2**
  - **Connect** the nearest **neighbour** along the dividing line



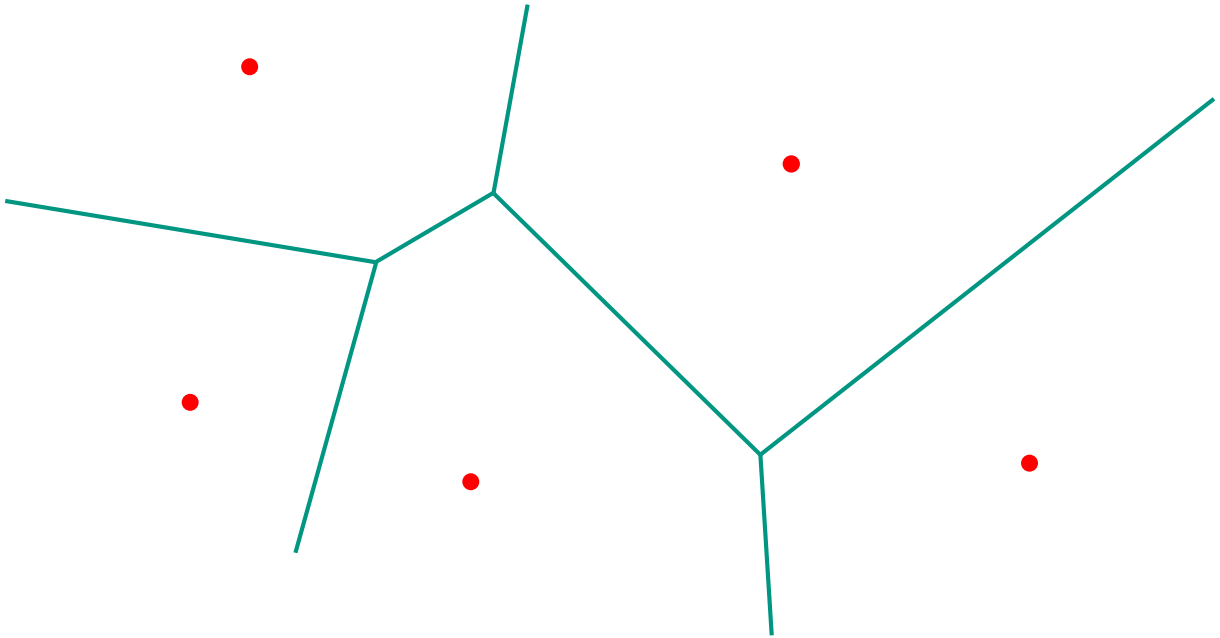
Voronoi Diagrams: Construction (6)

- **Merge** the Voronoi diagrams of **P1** and **P2**
  - Draw and cut off new **perpendicular bisector**



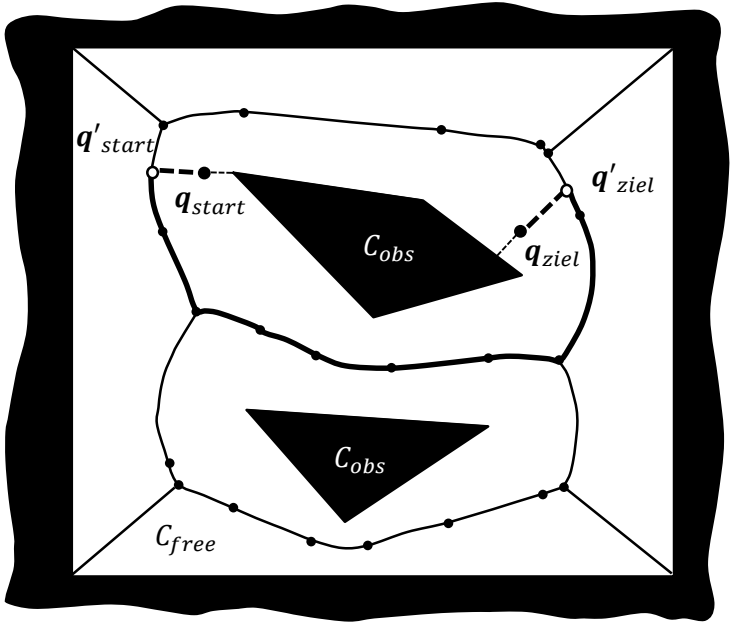
Voronoi Diagrams: Construction (7)

- Voronoi diagram for **P**



Voronoi Diagrams: Advantages and Disadvantages

Now: obstacles are represented by polygons

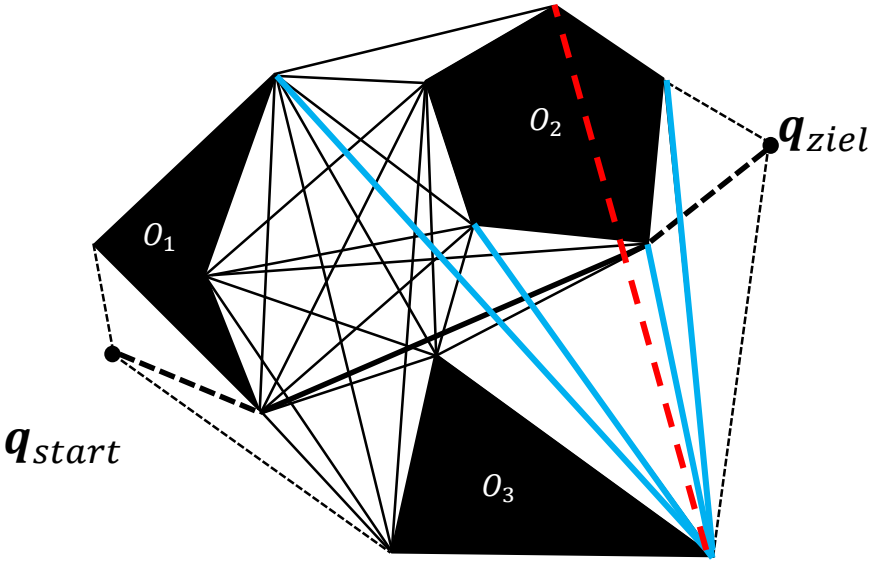


- Voronoi diagram consist of straight and parabolic segments
- Advantages:
  - **Maximal distance to obstacles**
- Disadvantages:
  - Path found is usually not the shortest
  - Only few paths are generated if there are few obstacles

Jean-Claude Latombe, Robot Motion Planning, Springer, 1991, page 173

Visibility Graph

- **Connect** each pair of **Vertices** on the **Edge** of $\mathcal{C}_{\text{free}}$ with a straight line segment if it does **not intersect an obstacle**
- Connect $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{goal}$ as well



Jean-Claude Latombe, Robot Motion Planning, Springer, 1991, page 156

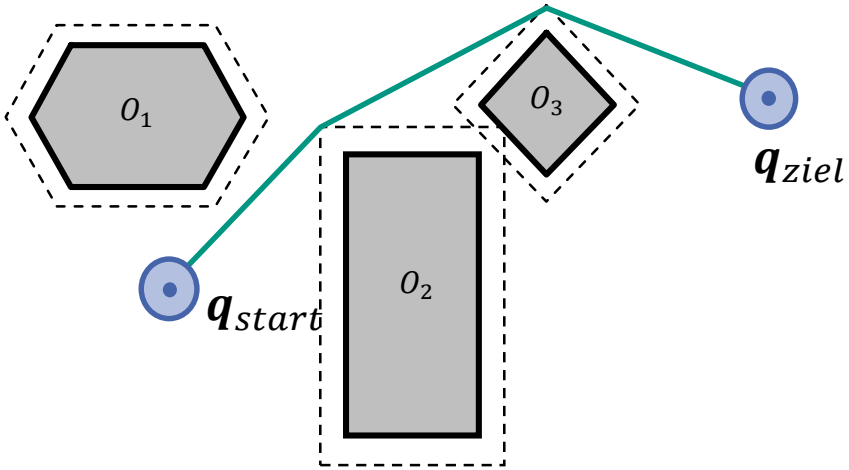
Visibility Graph: Advantages and Disadvantages

- Advantages:
  - Finds **optimal** (shortest) path
    - for 2d problems and
    - if the robot and all obstacles can be represented by convex polygons
- Disadvantages:
  - Paths are **not necessarily collision-free**, as **obstacle edges** can also be path segments.
  - Solution: **Expand the obstracles**

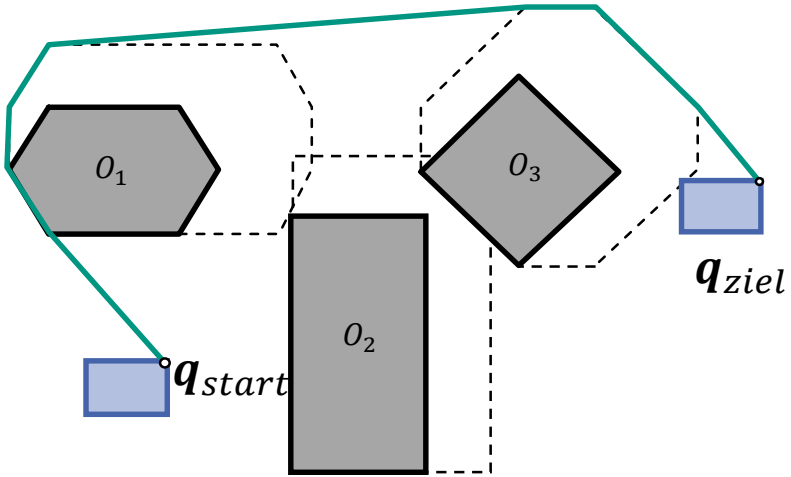
- Can also be used in $\mathbb{R}^3$, but the paths found are generally not the shortest paths.

Visibility Graph : Expansion of Obstacles

- Expand obstacles by the size of the robot → Grown Obstacles
- Expansion depends of the shape of the robot



Circle shaped robot



Rectangle shaped robot

Cell Decomposition

- Approach:
  1. decompose C_{free} in cells, that makes it easy to find a path between two configurations within the cell
  2. Represent the spatial layout in adjacency graph
  3. Search the optimal path from $\boldsymbol{q}_{start}$ to $\boldsymbol{q}_{goal}$ in the graph
- There are two kinds of cell decomposition**:**
  - Exact cell decomposition
  - Approximated cell decomposition

Exact Cell Decomposition

- Decomposition of free space C_{free} in cells c_i, with:
  - No overlap between cells

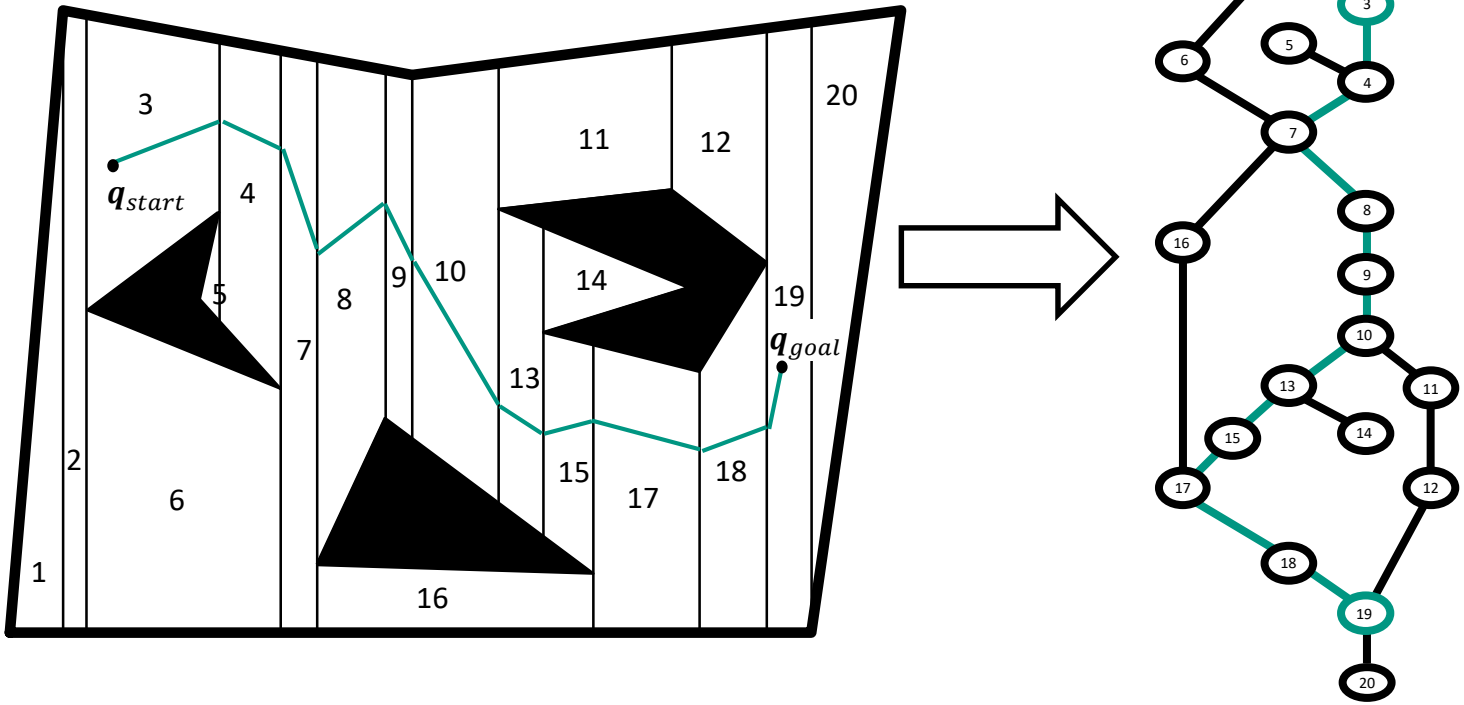
$$\forall i, k, i \neq k: c_i \cap c_k = \emptyset$$

  - C_{free} as the union of all c_i

$$\bigcup_{i=1}^{n} c_i = C_{free}$$

Exact Cell Decomposition

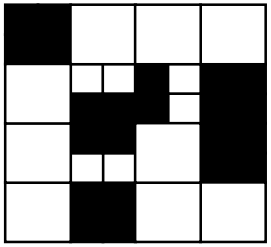
- Exact Cell Decomposition with **Line-Sweep**



Approximated Cell Decomposition

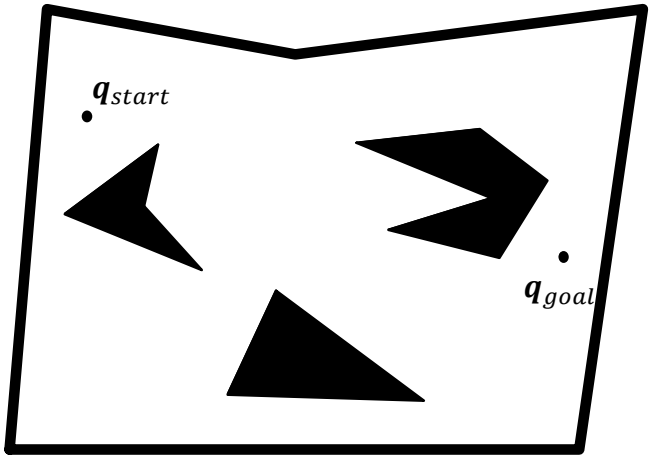
- Approach (e.g. Quadtree in 2D or Octree in 3D):
  1. Decompose free space C_{free} into cells of predefined shape (e.g. rectangles)
  2. If cell is not entirely within C_{free}, reduce its size and decompose it further
  3. repeat step 2 until the cell have a (predefined) minimal size
- **Advantage**
  - Simple decompostion and therefore simple path search
- **Disadvantage**
  - The free space can only be represented approximately

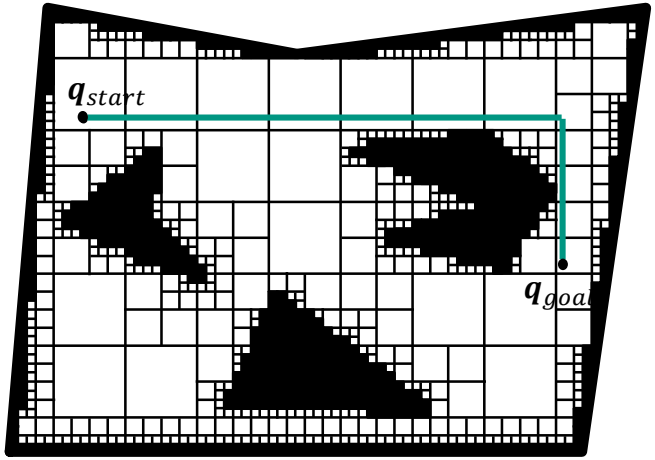
Approximated Cell Decomposition: Example



Obstacle space

Free space

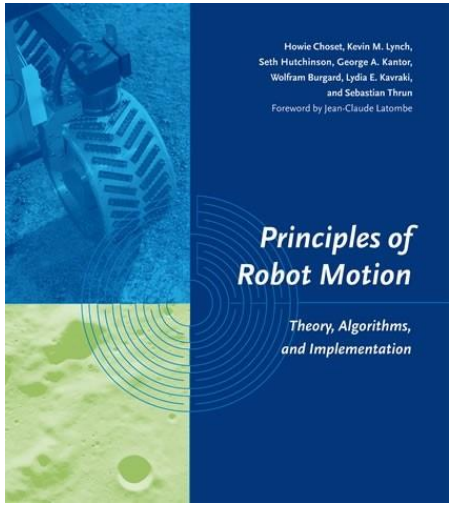




Path Planning for Mobile Robots

1. Generation of the graph W
   - Retraction methods, e.g. Voronoi Diagrams
   - Visibility Graph
   - Cell Decomposition

2. **Search in W**
   - Tree search
   - A*



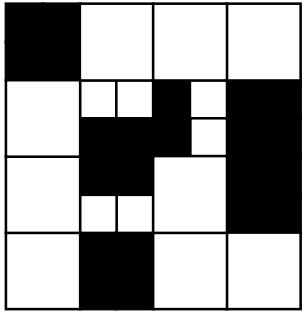
Tree search

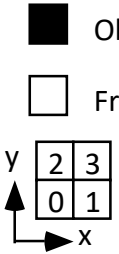
- Use case:
  - Mobile robot
  - 2D workspace and configuration space
- Representation of the configuration space as a **Quadtree**
  - Recursive decomposition of the space into cells
  - Cell are either free or contain an obstacle
- Motion planning:
  - Find cells containing start and goal configuration
  - Connect adjacent free cells from start to goal
  - Collison free path planning through free cells

四叉树，每个节点划分为4个象限

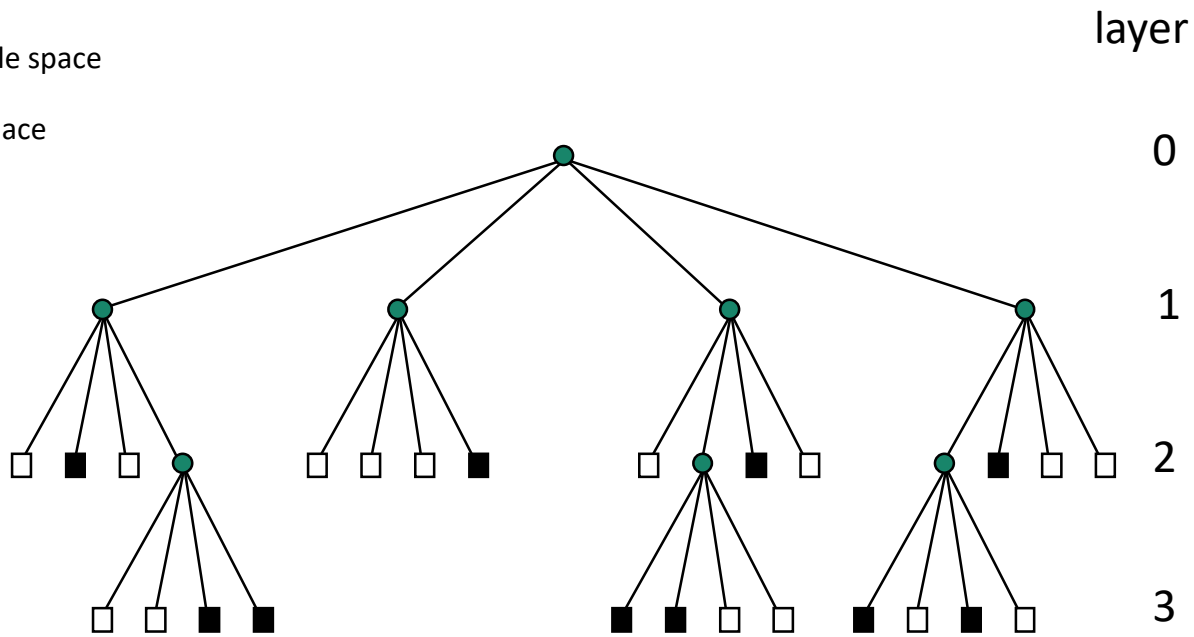
Tree search: Quadtree (1)

Represent configuration space as **Quadtree**

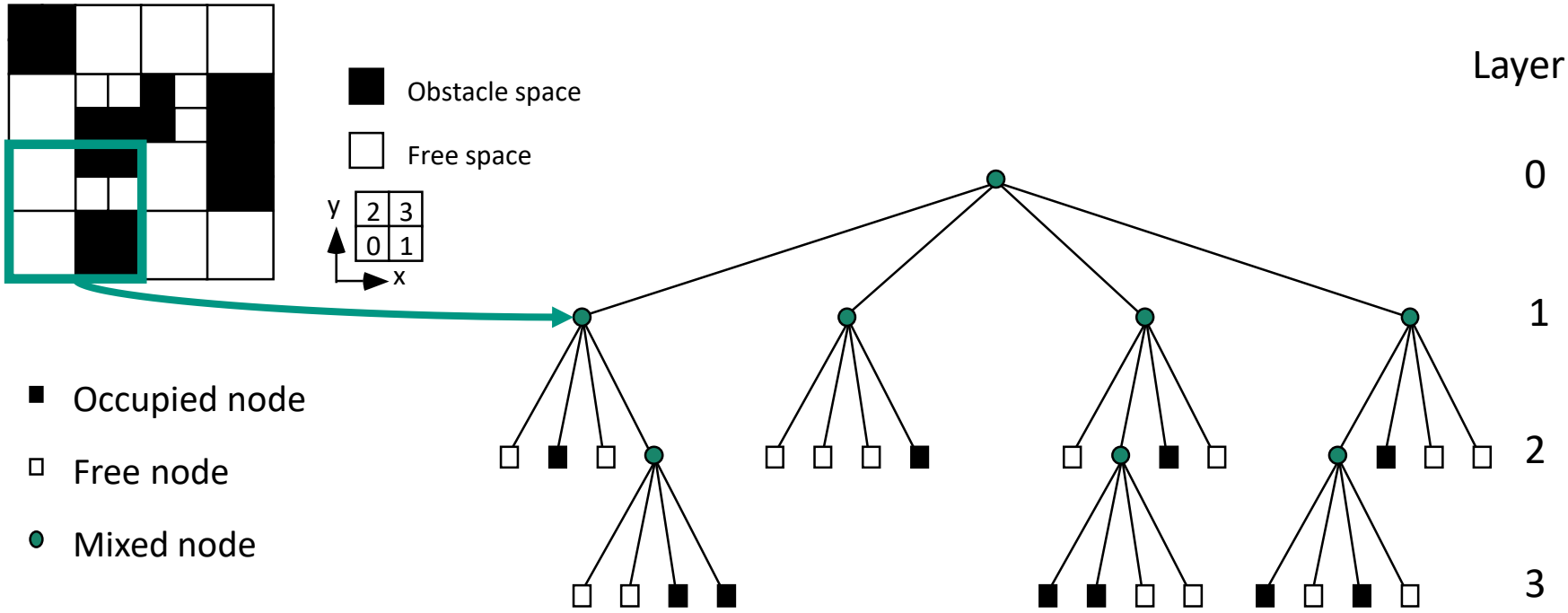




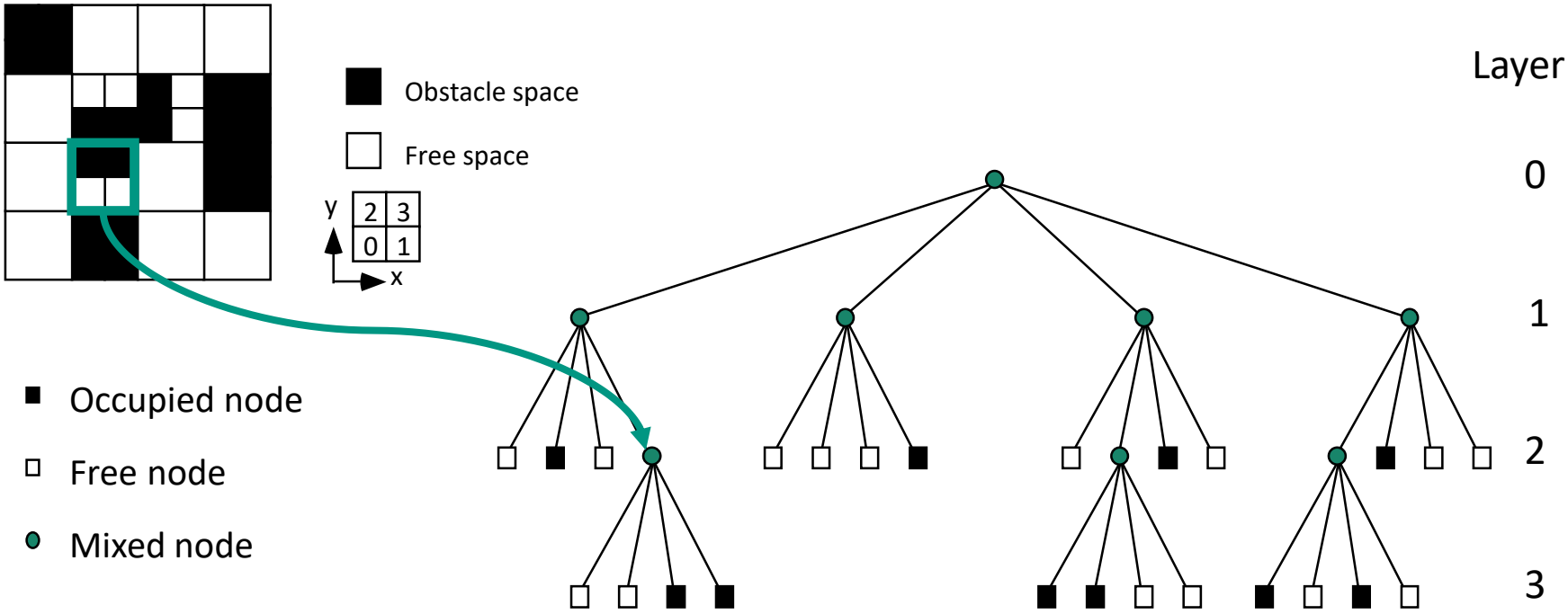
- Occupied node
- Free node
- Mixed node



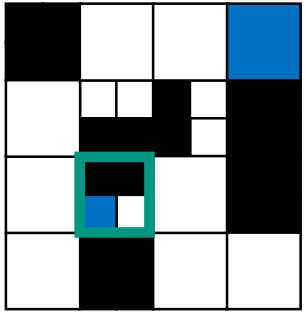
Tree search: Quadtree (2)

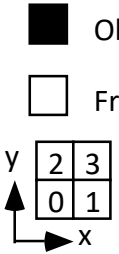


Tree search: Quadtree (3)

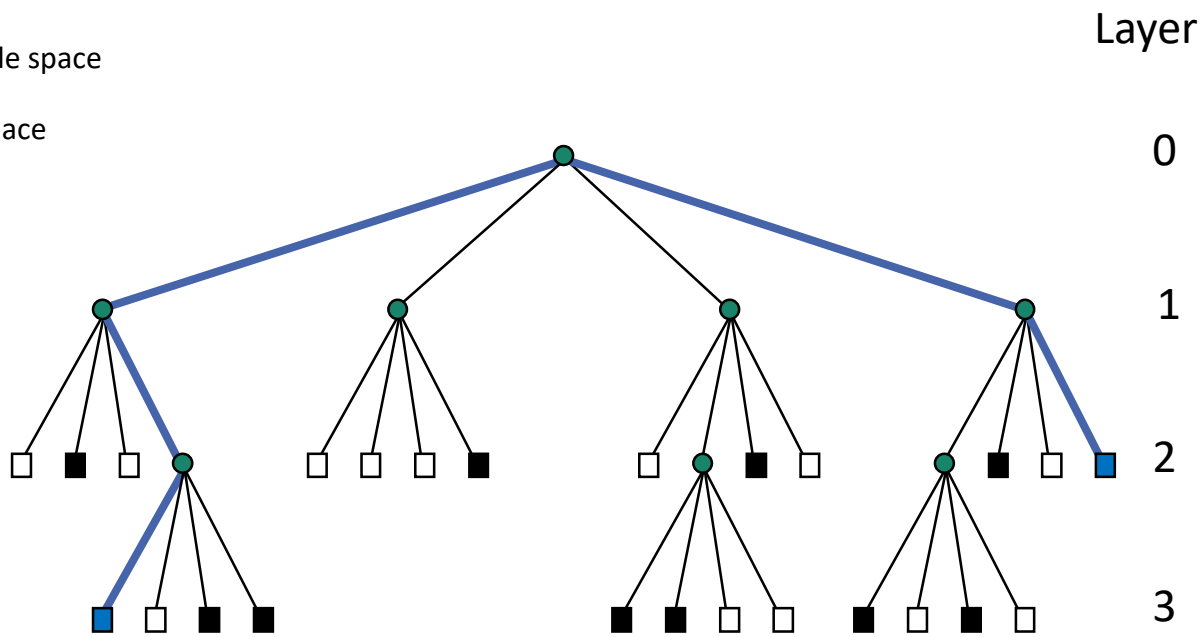


Baumsuche: Such einen Weg im Baum



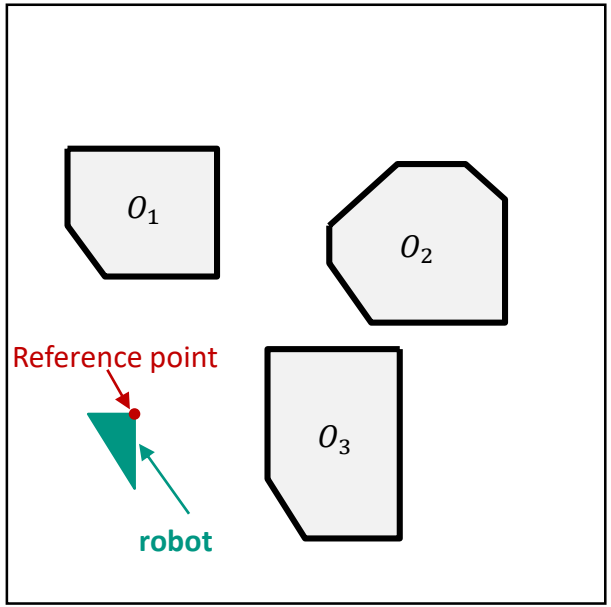


- Occupied node
- Free node
- Mixed node



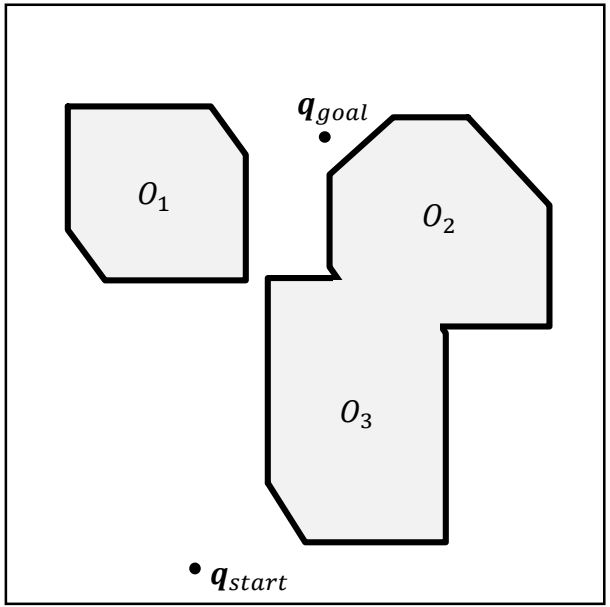
Tree search: Example (1)

Workspace



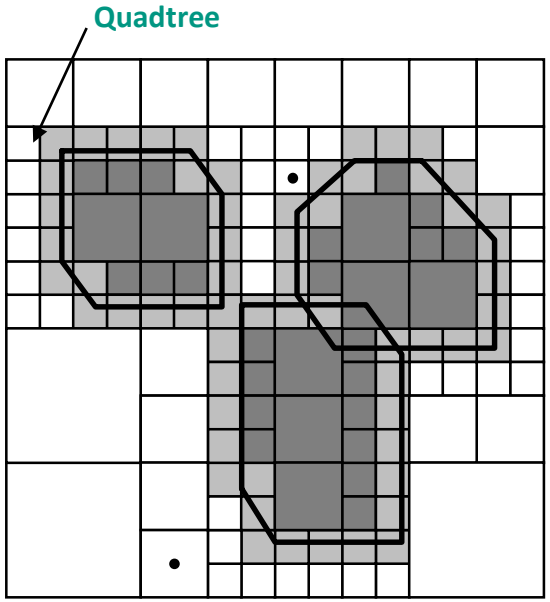
Workspace of the robot with obstacles

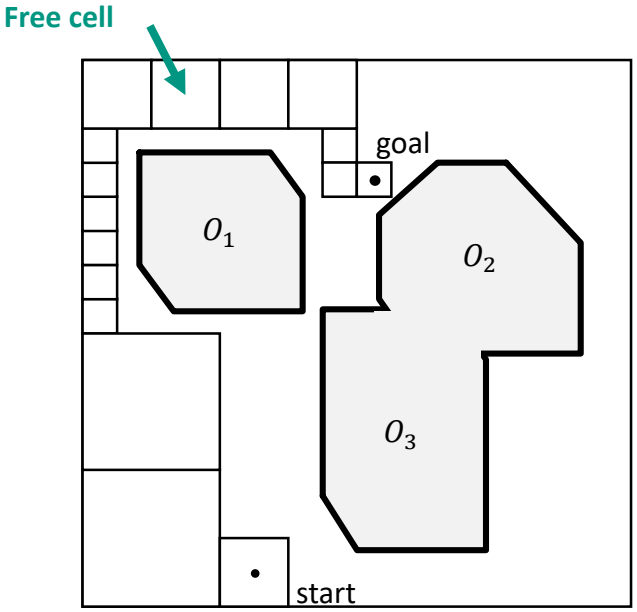
Configuration space



Configuration space of the robot (with expanded obstacles)

Tree search: Example (2)

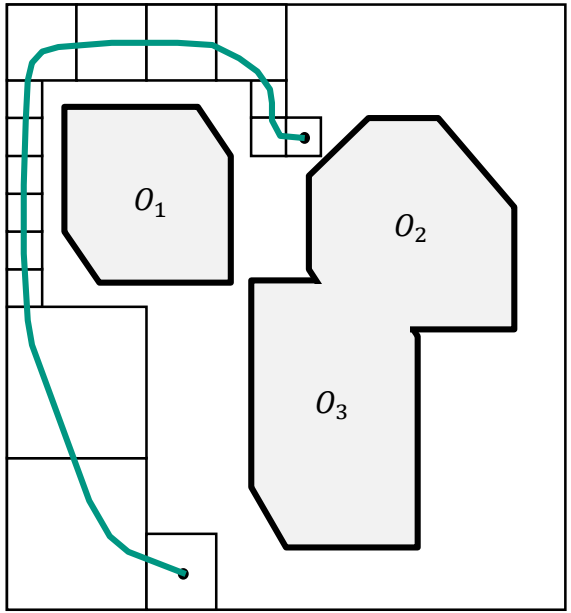




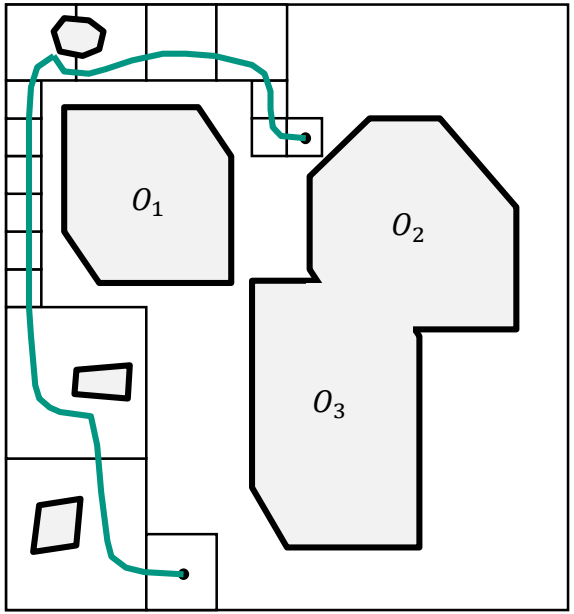
Decomposition of the configuration space into cells

Required: sequence of free cells from start to goal state

Tree search: Example (3)



Obstacle free path

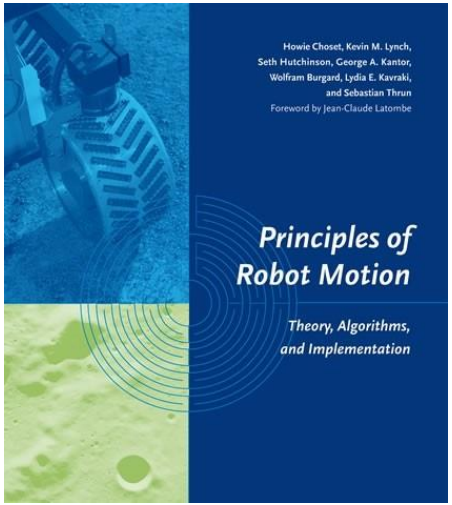


Evasion technique to avoid local obstacles

Path Planning for Mobile Robots

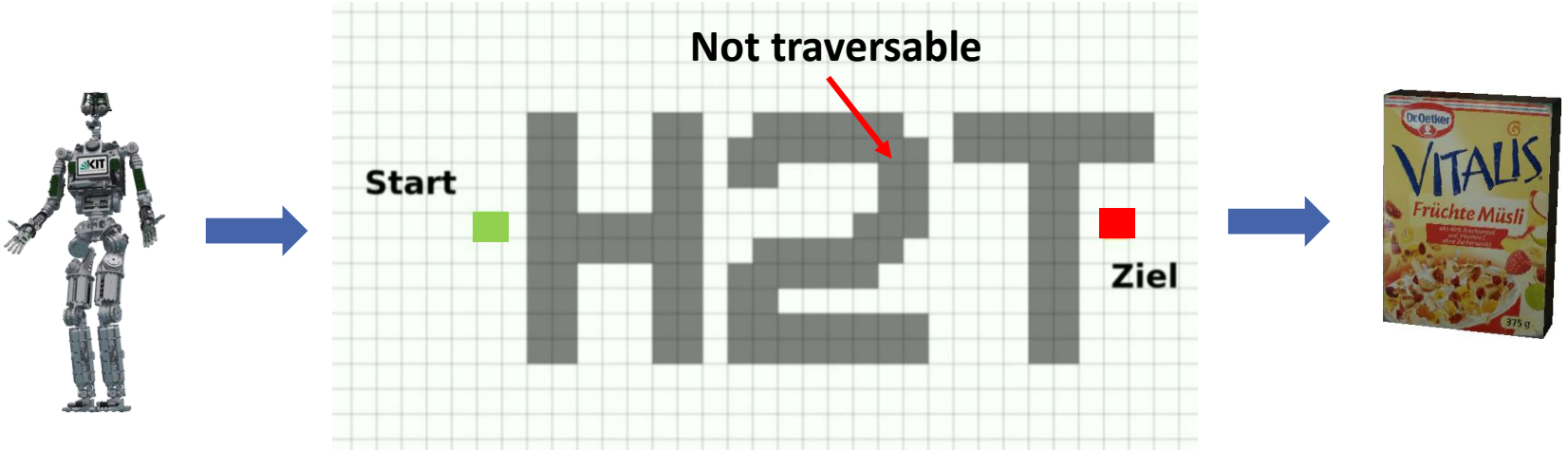
1. Generation of the graph W
   - Retraction methods, e.g. Voronoi Diagrams
   - Visibility Graph
   - Cell Decomposition

2. **Search in W**
   - Tree search
   - A*



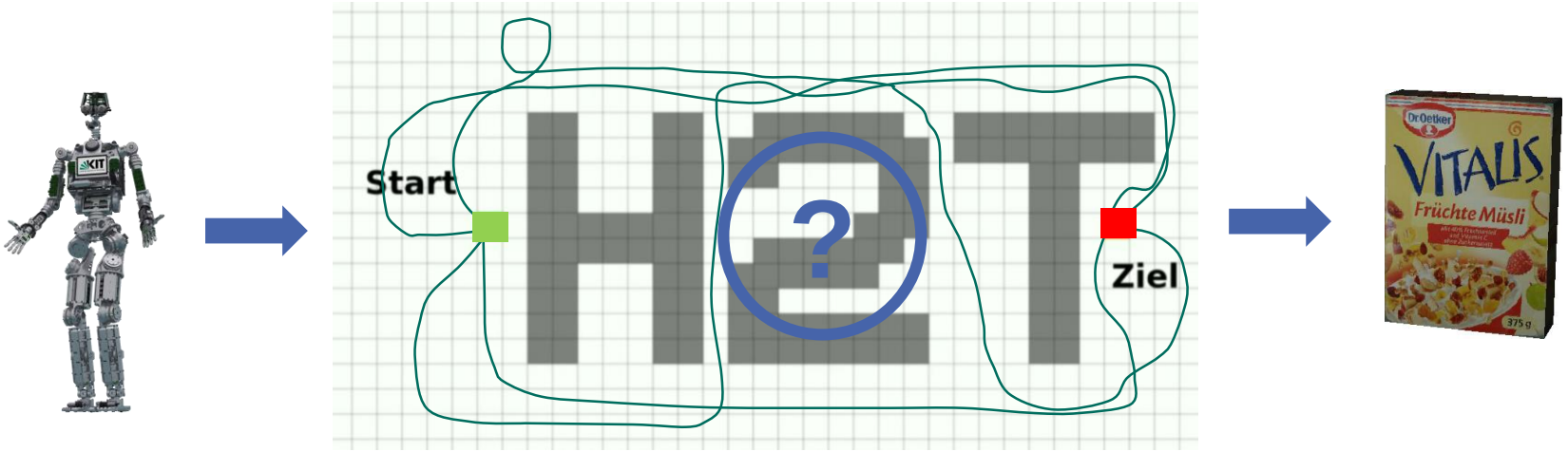
A*- Algorithm – Motivation

- Shortest path from start to goal



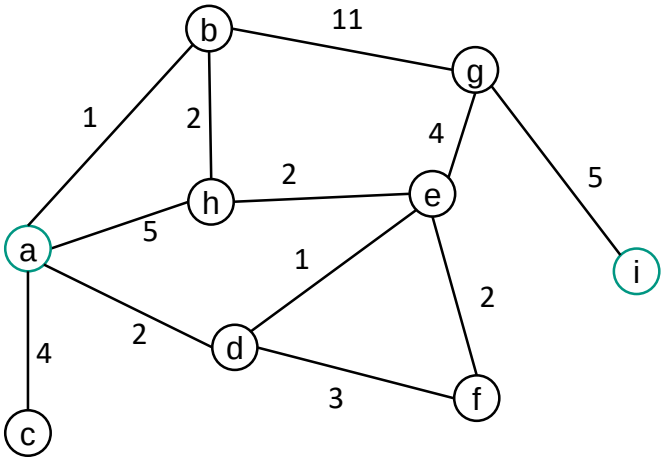
A*- Algorithm – Motivation

- Shortest path from start to goal



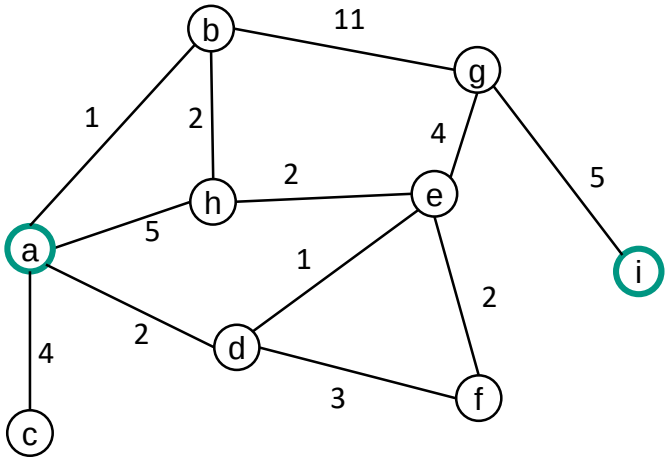
A*- Algorithm (1)

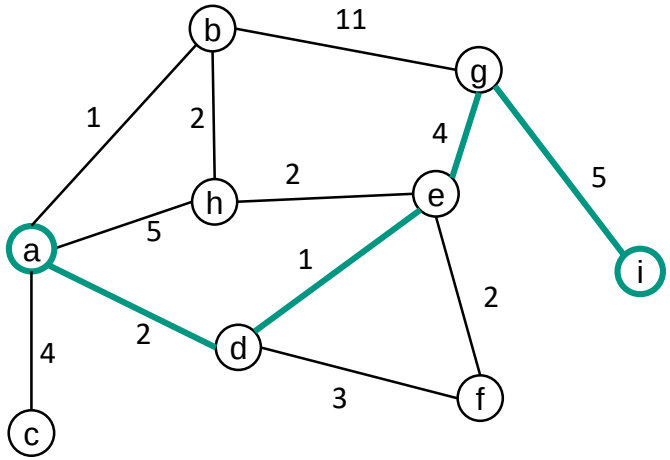
- Shortest path from start to goal
- A* is one of the most well-known **path planning algorithms in weighted graphs**



A*- Algorithm (2)

- Shortest path from start to goal
- A* is one of the most well-known **path planning algorithms in weighted graphs**

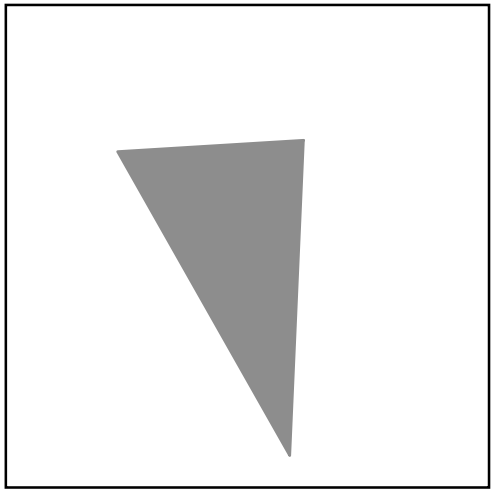


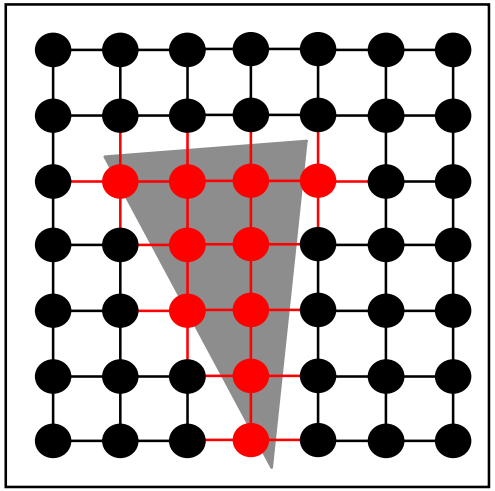


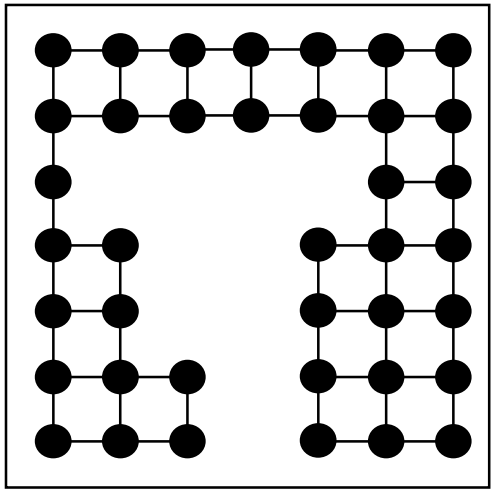
A*- Algorithm (3)

- Shortest path from start to goal
- A* is one of the most well-known **path planning algorithms in weighted graphs**

  → discretizing of the space is needed

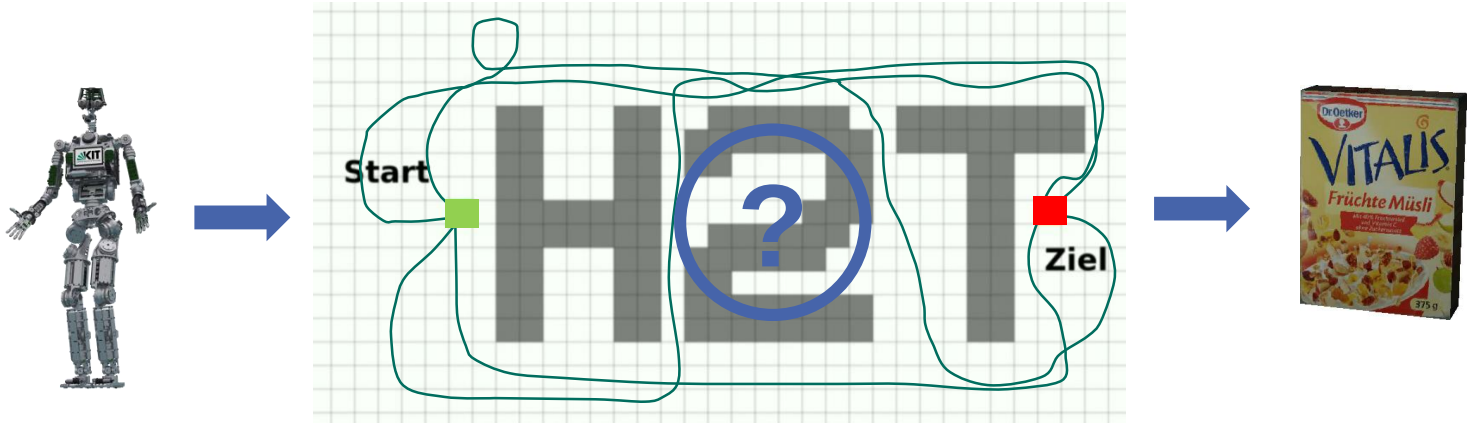






A*- Algorithm (4)

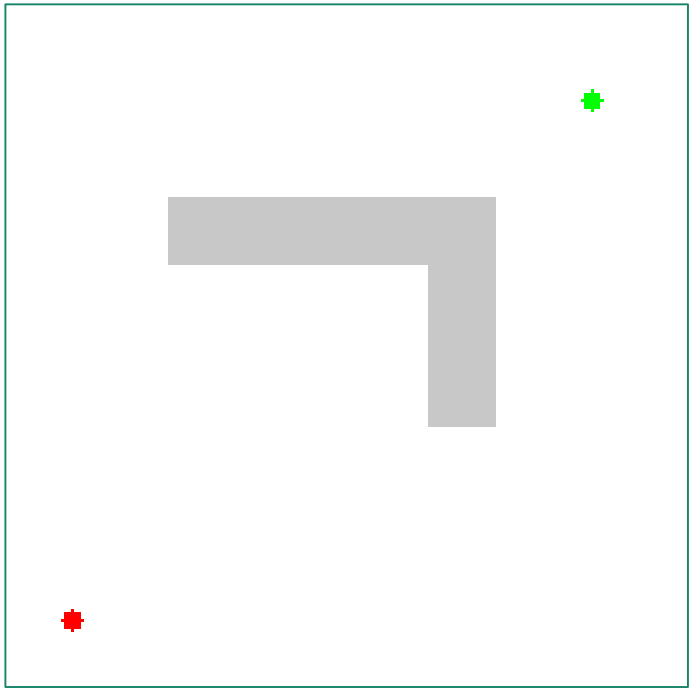
- Shortest path from start to goal



- **Cost function $f(x) = g(x) + h(x)$**
  - $g(x)$ actual costs from start to node x
  - $h(x)$ **estimated** costs from node x to goal node

A*- Algorithm (5)

- A* (“*A Star*”) is an algorithm for **best-first search**
- Finds the optimal path from start node v_{start} to a goal node v_{ziel}
- Optimal w.r.t. path costs
  (e.g. shortest path, shortest time,
  smallest edge weights, etc.)
- **Cost function $f(x) = g(x) + h(x)$**
  - $g(x)$ actual costs from start to node x
  - $h(x)$ **estimated** costs from node x to goal node



https://de.wikipedia.org/wiki/A*-Algorithmus

A*- Algorithm (6)

- Iterative approach
- Manage **two** node sets
  - **Open Set** O: nodes not visited yet
  - **Closed Set** C: already visited nodes
- **Update**: for a visited node v_n:
  - **Predecessor node** $pred(v_n)$
  - **accumulated cost,** to reach v_n: $g(v_n)$
  - **Heuristic** of expected cost to reach goal: $h(v_n)$
- **Initialize**
  - $O = \{v_s\}$
  - $C = \{\}$
  - $g(v_i) = \infty \quad 1 \leq i \leq K$ (or very big value for initial costs)
  - $g(v_s) = 0$

A*-Algorithm (7)

- **Algorithm**
  while $O \neq \emptyset$
  - Determine next node to expand
    - find $v_i \in O$ with minimal $f(v_i) = g(v_i) + h(v_i)$
  - if $v_i = v_{ziel}$
    solution found: traverse predecessor of v_i until v_{start} is reached
  - $O.remove(v_i)$
  - $C.add(v_i)$
  - **Update** all succesors v_j of v_i

    if $v_j \in C$, skip v_j

    if $v_j \notin O$, $O.add(v_j)$

    if $g(v_i) + cost(v_i, v_j) < g(v_j)$

    $g(v_j) = g(v_i) + cost(v_i, v_j)$

    $h(v_j) = heuristic(v_j, v_{ziel})$

    $pred(v_j) = v_i$

A*-Algorithm: Example (1)

- Grid with 15 nodes
- Find optimal path from v_2 to v_{13}
  - Only horizontal and vertical movements allowed
  - Costs:
    - Entering a grey cell: 1
    - Entering a yellow cell: 4
  - Heuristic h:
    Euclidean distance to v_{13}
    (e.g. from $v_{11}$to v_{13}: $h(v_{11}) = \sqrt{2}$)

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9
v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}

A*-Algorithm: Example (2)

- Initialize:

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9
v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}

A*-Algorithm: Example, Initializing

- Initialize:
  - $O = \{v_2\}$
    - $f(v_2) = 0 + h(v_2) = \sqrt{4^2 + 1^2} \approx 4.12$
  - $C = \{\}$

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9
v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}

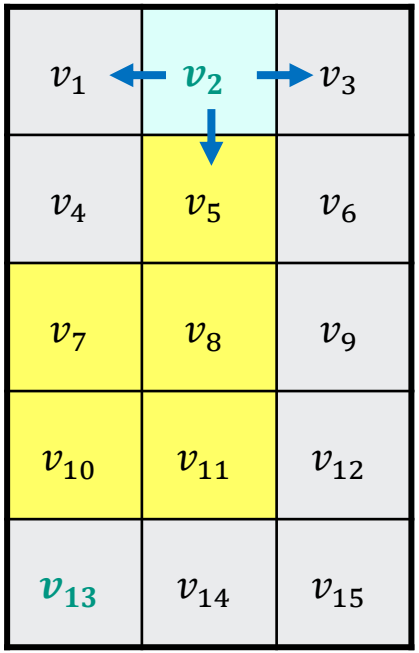
A*-Algorithm: Example, Step 1 (1)

- State:
  - $O = \{v_2\}$
    - $f(v_2) = 0 + h(v_2) = \sqrt{4^2 + 1^2} \approx 4.12$
  - $C = \{\,\}$
- Update:

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9
v_{10}	v_{11}	v_{12}
v_{13}	v_{14}	v_{15}

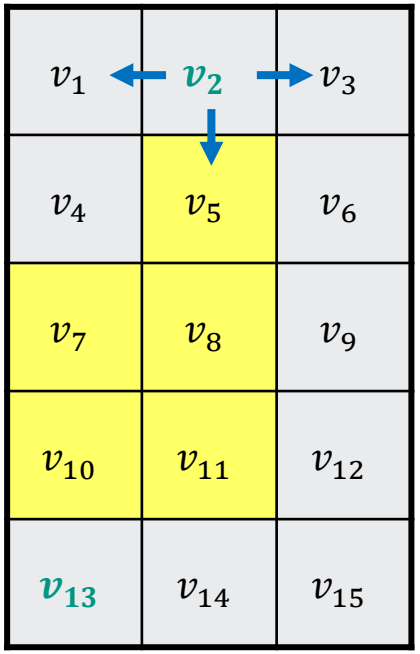
A*-Algorithm: Example, Step 1 (2)

- State:
  - $O = \{v_2\}$
    - $f(v_2) = 0 + h(v_2) = \sqrt{4^2 + 1^2} \approx 4.12$
  - $C = \{\}$
- Update:
  - Expand v_2
  - $O = \{v_1, v_3, v_5\}$
    - $f(v_1) = 1 + h(v_1) = 1 + 4 = 5$
    - $f(v_3) = 1 + h(v_3) = 1 + \sqrt{4^2 + 2^2} \approx 5.47$
    - $f(v_5) = 4 + h(v_5) = 4 + \sqrt{3^2 + 1^2} \approx 7.16$
  - $C = \{v_2\}$



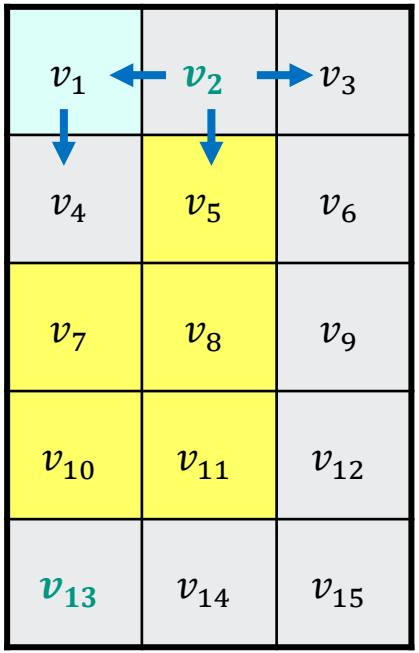
A*-Algorithm: Example, Step 2 (1)

- State:
  - $O = \{v_1, v_3, v_5\}$
    - $f(v_1) = 5$
    - $f(v_3) \approx 5.47$
    - $f(v_5) \approx 7.16$
  - $C = \{v_2\}$
- Update:



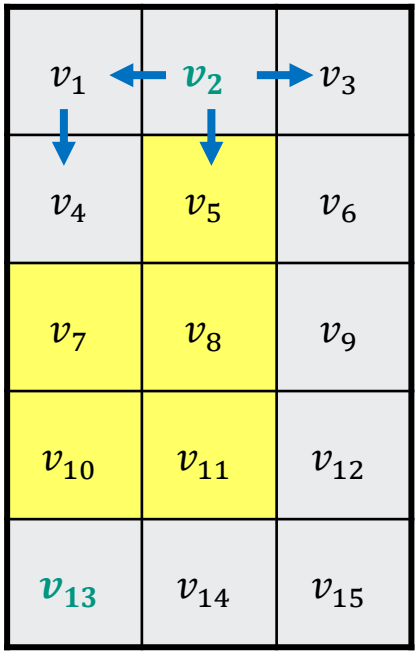
A*-Algorithm: Example, Step 2 (2)

- State:
  - $O = \{v_1, v_3, v_5\}$
    - $f(v_1) = 5$
    - $f(v_3) \approx 5.47$
    - $f(v_5) \approx 7.16$
  - $C = \{v_2\}$
- Update:
  - Expand v_1
  - $O = \{v_1, v_3, v_5, v_4\}$
    - $f(v_4) = 2 + h(v_4) = 2 + 3 = 5$
  - $C = \{v_2, v_1\}$



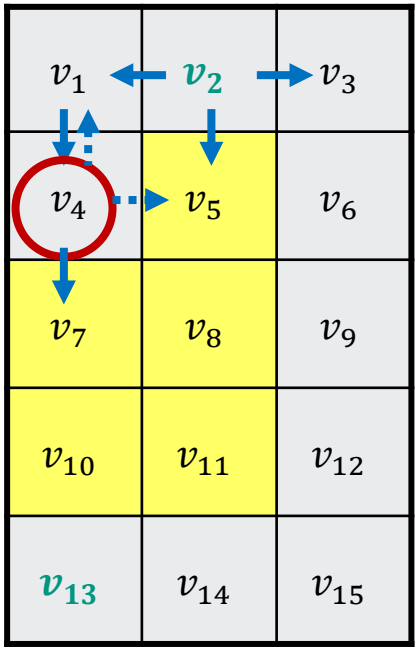
Task 4.1: A*-Algorithm, Step 3 (1)

- State:
  - $O = \{v_3, v_4, v_5\}$
    - $f(v_3) \approx 5.47$
    - $f(v_4) = 5$
    - $f(v_5) \approx 7.16$
  - $C = \{v_1, v_2\}$
- Update:



Task 4.1: A*-Algorithm, Step 3 (2)

- State:
  - $O = \{v_3, v_4, v_5\}$
    - $f(v_3) \approx 5.47$
    - $f(v_4) = 5$
    - $f(v_5) \approx 7.16$
  - $C = \{v_1, v_2\}$
- Update:
  - Expand v_4
  - $O = \{v_3, v_5, v_7\}$
    - $g(v_4) + cost(v_4, v_5) = 2 + 4 = 6 \geq g(v_5) = 4$
      $\Rightarrow$ No update
    - $f(v_7) = 6 + h(v_7) = 6 + 2 = 8$
  - $C = \{v_1, v_2, v_4\}$



A*-Algorithm: Properties

- Find **optimal solution**, if heuristic h **is suitable**
  - heuristic h is **suitable**, if it does not overestimate the costs to the goal

- A* is **optimal in efficiency** for all (suitable) heuristics h
  - No optimal algorithm that uses the same heuristic h visits less nodes than A*

- If $\forall x\colon\ h(x) = 0$: **Dijstra`s algorithm**, resulting in $f = g$
  - Greedy algorithm: does not use the distance to the goal
  - Visits more nodes than necessary

Recap

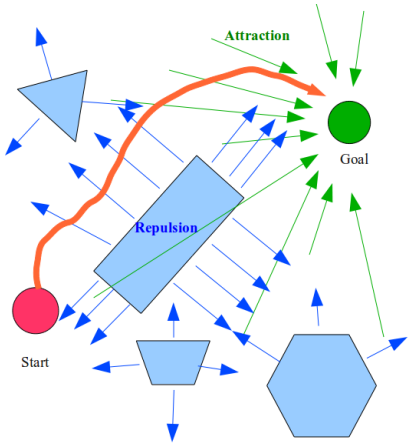
- Calculating collision-free trajectories in configuration space
  - **1. Step:** Efficient representation of free space by a graph
    - Retraction approaches, e.g., Voronoi diagram
    - Visibility graph
    - Cell decomposition
  - **2. Step:** Search optimal path in graph
    - A*
    - RRT
    - …

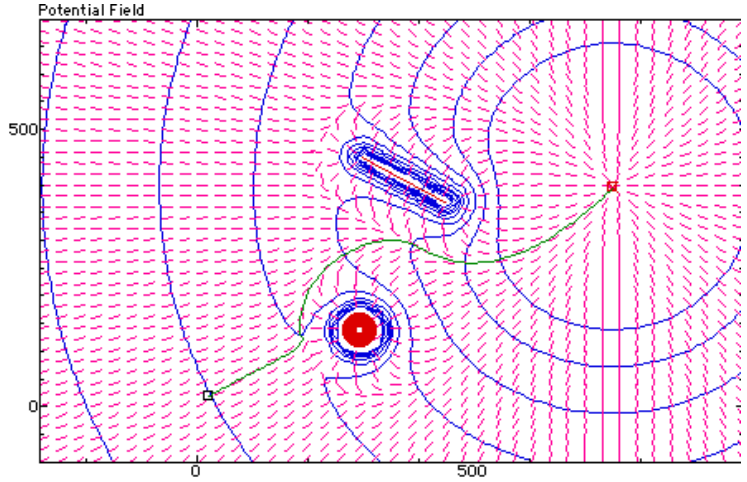
Table of Contents

- Motivation
- Fundamentals of motion planning
- **Path planning for mobile robots**
  - Graph-based
  - **Potential fields**
- Motion planning for manipulators

Potential Field Method

Developed for motion planning by Oussama Khatib in 1986





O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots" , International Journal on Robotics Research (IJRR), 5(1):90--98, Spring, 1986

Potential Fields (1)

- The robot moves under the influence of forces exerted on it by a potential field
- **Definition:**
  - A potential field U is a **scalar function** defined over the free space

$$U: \ C_{free} \rightarrow \mathbb{R}$$

  - The force acting on the robot at a point $\boldsymbol{q}$ of the potential field is the negative gradient of the potential field at that point (configuration)

$$F(\boldsymbol{q}) = -\nabla U(\boldsymbol{q})$$

Potentialfeld: Repulsive Potenzial

- Obstacles create a **repulsive potential** 排斥场
- However, the robot shall **not** be influenced for large distances to obstacles $(> \rho_0)$
- **Example (FIRAS function):**

$$U_{rep}(\boldsymbol{q}) = \begin{cases} \frac{1}{2}\nu\left(\frac{1}{\rho(\boldsymbol{q},\boldsymbol{q}_{obs})} - \frac{1}{\rho_0}\right)^2 & if \;\; \rho(\boldsymbol{q},\boldsymbol{q}_{obs}) \leq \rho_0 \\ 0 & else \end{cases}$$

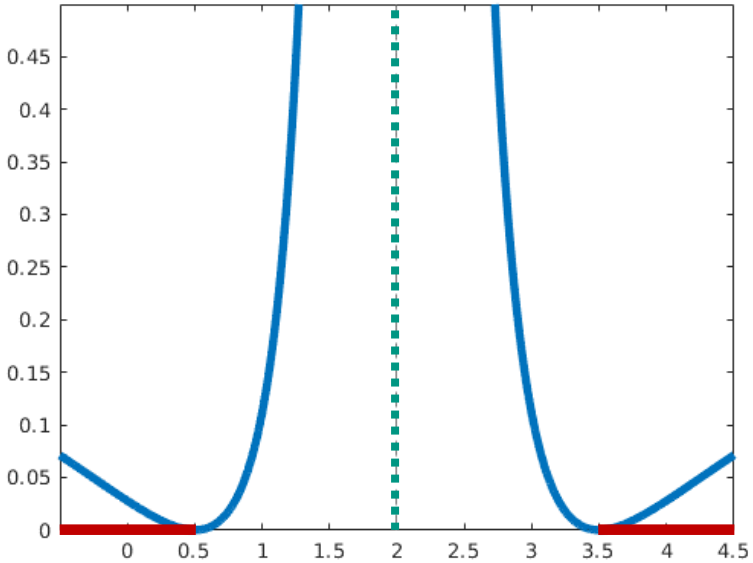
$\rho(\mathbf{q},\mathbf{q}_{obs}) = \|\boldsymbol{q} - \boldsymbol{q}_{obs}\|$ is the shortest distance between robot and obstacle

$$F_{rep} = -\nabla U_{rep} = \nu\left(\frac{1}{\rho(\boldsymbol{q},\boldsymbol{q}_{obs})} - \frac{1}{\rho_0}\right) \cdot \frac{1}{\rho(\boldsymbol{q},\boldsymbol{q}_{obs})^2} \cdot \frac{\boldsymbol{q} - \boldsymbol{q}_{obs}}{\rho(\boldsymbol{q},\boldsymbol{q}_{obs})}$$

$\frac{\partial \|\boldsymbol{x}\|_2}{\partial x_i} = \frac{x_i}{\|\boldsymbol{x}\|_2}$ $\frac{\partial \|\boldsymbol{x}\|_2}{\partial \boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$

Potential Field: Example for an Obstacle

$$U_{rep}(x) = \begin{cases} \left(\frac{1}{\|x-2\|} - \frac{1}{1.5}\right)^2 & if\ \|x-2\| \leq 1.5 \\ 0 & else \end{cases}$$



Obstacle

Near the obstacle

Else

Potential Fields (3)

- **Attractive potential**

  → There shall be only a single minimum, located at $\boldsymbol{q}_{ziel}$

- **Linear function** of the distance to the goal:

$$U_{attr}(\boldsymbol{q}) = k \cdot \|\boldsymbol{q} - \boldsymbol{q}_{goal}\|$$

$$F_{\boldsymbol{attr}}(\boldsymbol{q}) = -\nabla U_{attr}(\boldsymbol{q}) = -k \cdot \frac{\boldsymbol{q} - \boldsymbol{q}_{\boldsymbol{goal}}}{\|\boldsymbol{q} - \boldsymbol{q}_{\boldsymbol{goal}}\|}$$

$$\frac{\partial \|\boldsymbol{x}\|_2}{\partial x_i} = \frac{x_i}{\|\boldsymbol{x}\|_2} \qquad \frac{\partial \|\boldsymbol{x}\|_2}{\partial \boldsymbol{x}} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$$

- The force is constant, however, a decreasing force is desirable close to the goal
- Therefore, a quadratic function is used

Potential Fields (4)

- **Quadratic function** of the distance to the goal:

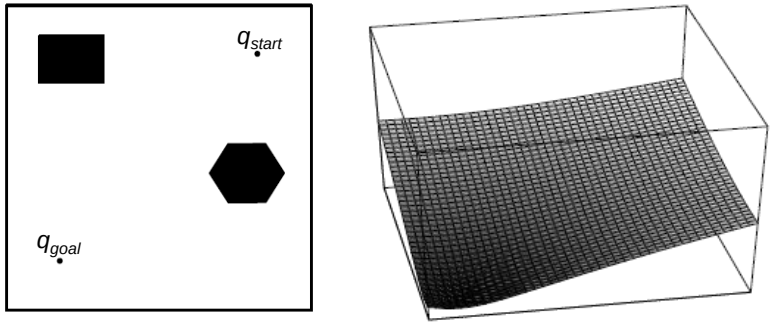
$$U_{attr}(\boldsymbol{q}) = k \cdot \frac{1}{2} \left\| \boldsymbol{q} - \boldsymbol{q}_{goal} \right\|^2$$

$$F_{attr}(\boldsymbol{q}) = -\nabla U_{attr}(\boldsymbol{q}) = -k \cdot \left(\boldsymbol{q} - \boldsymbol{q}_{goal} \right)$$

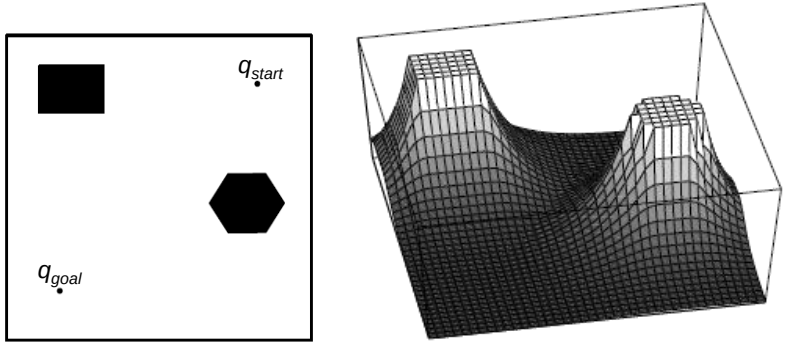
- Often, a combination of a linear and a quadratic function is used
  - Linear function, if far away from the goal
  - Quadratic function, if close to the goal

Potential Fields: Example (1)

- The **target** $\boldsymbol{q}_{goal}$ has the **attractive** potential U_{attr}

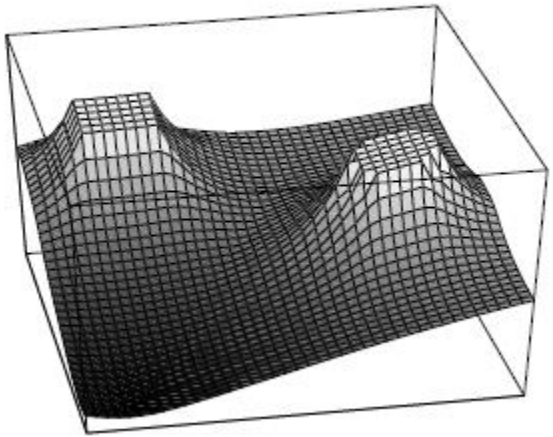


- The **obstacle space** C_{obs} has the **repulsive** potential U_{rep}



Potential Fields: Example (2)

- The sum of the acting forces determines the direction of motion
- Resulting potential field: $U(\boldsymbol{q}) = U_{attr}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q})$
- Resulting force field: $F(\boldsymbol{q}) = F_{attr}(\boldsymbol{q}) + F_{rep}(\boldsymbol{q})$



Potential Fields: Local Minima

- **Local minima:**

  Summing up U_{attr} and U_{rep} can lead to local minima of U.
  A robot moving along the negative gradient of the potential field can get „stuck“ in such local minimum.

- **Counter measures:**
  - Define U_{attr} and U_{rep} so that U does not have a local minimum except in $\boldsymbol{q}_{goal}$
  - In the search algorithm, apply techinques to „escape“ from local minima

Potential Fields: Parameters

- Choose suitable values for parameters ν (of the repulsive potential) and k (of the attractive potential); the ratio of these parameters is more important than their absolute values

- **For manipulators:**
  - Consider forces acting at the end-effector
  - Use Jacobi matrix to calculate the torques acting at the joints, which are induced by the force at the end-effector

$$\tau = J^T \cdot F$$

Table of Contents

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - Rapidly-exploring Random Trees (RRT)
  - Extensions of RRT
    - Constrained RRT
    - RRT*
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

Fundamentals of motion planning: Terms

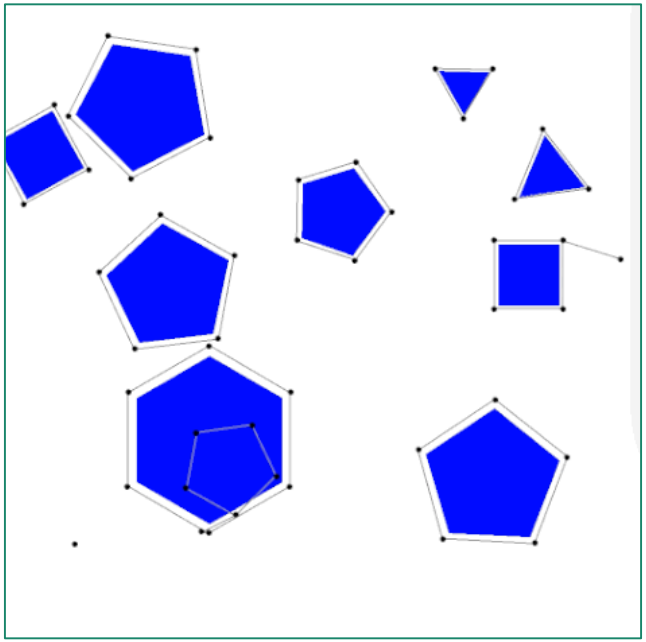
- **Path planning**
  - Rigid object (e.g. mobile robot, autonomous vehicle)
  - 2D problem (position: x, y)
  - 3D problem (position: x, y; rotation: α)
    → **Piano Mover's Problem** 如何从复杂的环境中找到一个物体的无碰撞路径。
- **Motion planning**
  - Multi-body systems (e.g. robot arms, multi-robot systems)
  - High-dimensional problem
- **Constraints**
  - Global contraints: Limit the allowed configuration space, e.g. upright end-effector positions, minimal motor currents, etc.
  - Local constraints: Limit the transitions between configurations, e.g. non-holonomic vehicles, max. velocities or accelerations

Table of Contents

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - Rapidly-exploring Random Trees (RRT)
  - Extensions of RRT
    - Constrained RRT
    - RRT*
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

Probabilistic Roadmaps (PRM) (1)

- Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; Overmars, M. H. (1996), "**Probabilistic roadmaps for path planning in high-dimensional configuration spaces**", IEEE Transactions on Robotics and Automation, 12 (4): 566–580, doi:10.1109/70.508439



Eric O. Scott
[CC BY-SA 4.0], via
Wikimedia Commons

Probabilistic Roadmaps (PRM) (2)

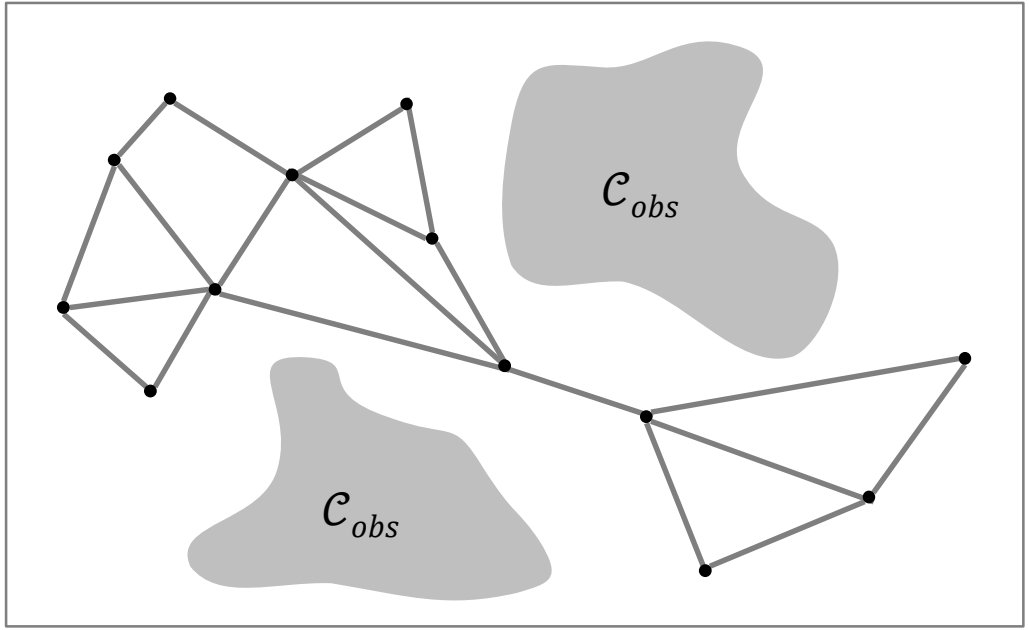
- **Multi-query** in **static** environment
- **Approximation of the free space** by graph (Roadmap)
  - More efficient than creating an explicit representation of the free space (C_{free})

PRM Algorithm

- **Step 1**: Sample and create a graph
  - Create a collision-free graph by choosing random points **(Sampling)**
- **Step 2**: Query
  - Connect $\boldsymbol{q}_{start}$ und $\boldsymbol{q}_{goal}$ to the graph
  - Search a path from $\boldsymbol{q}_{start}$ to $\boldsymbol{q}_{goal}$ on the graph

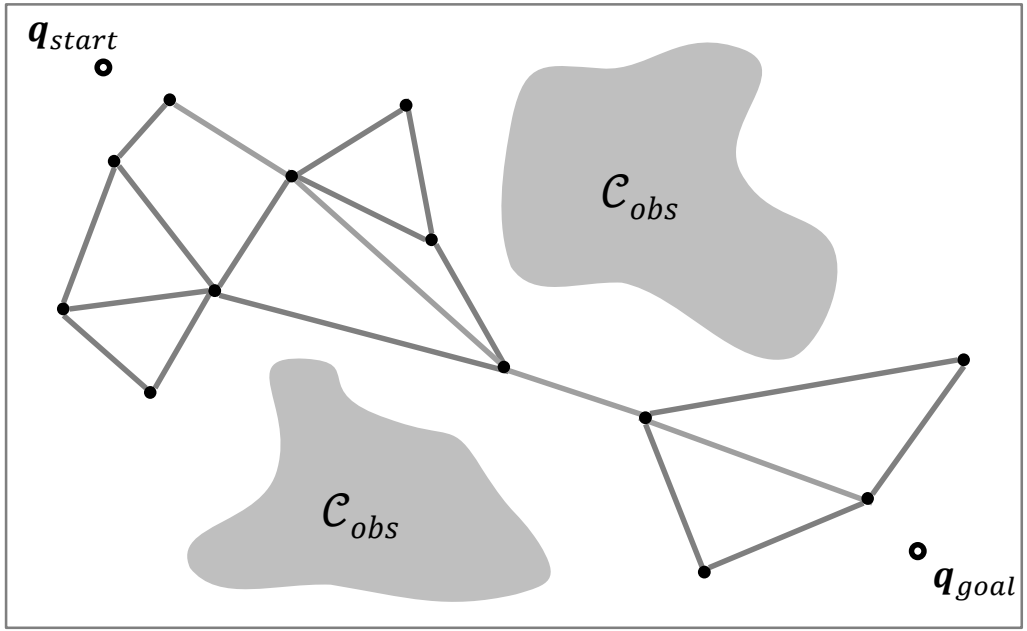
PRM: Sample and Create a Graph

- Randomly choose collision-free probe points (Sampling)
- Proble points are connected by collision-free paths using **fast local planners**



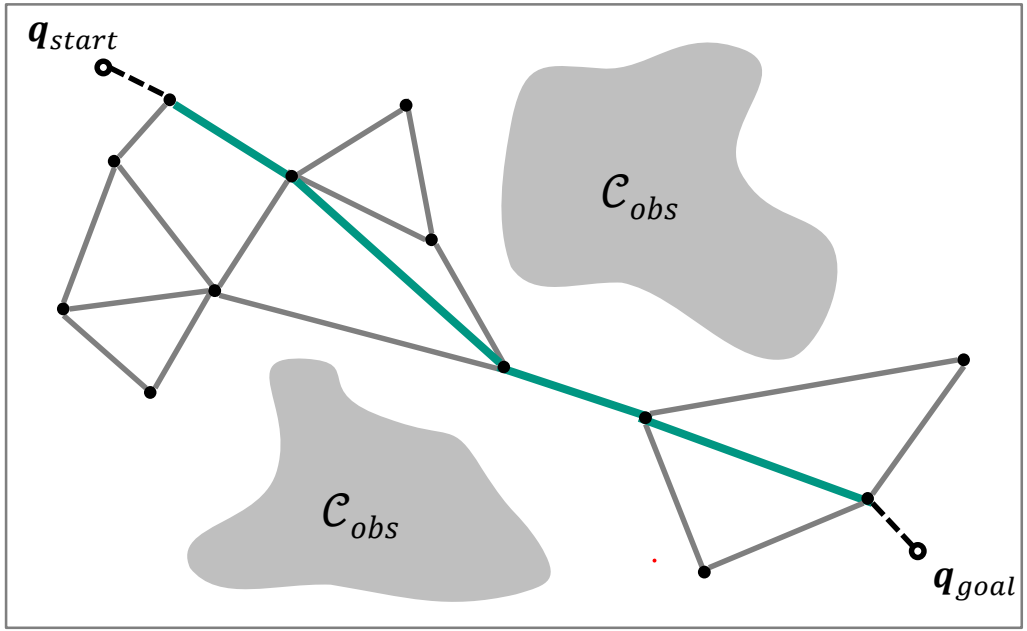
PRM: Query (1)

- Connect $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{goal}$ to the graph
- Search in the graph
  (**e.g., using A***)



PRM: Query (2)

- Connect $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{goal}$ to the graph
- Search in the graph (**e.g., using A***)



PRM: Construction of the Graph

- N: Number of nodes of the graph
- R: PRM, Graph
- Algorithm:
  - Sample N random configurations in C_{free}
  - Insert these configurations as nodes into R
  - For each node $v_i \in R$
    - Find the k next neighbors of v_i in R: $\mathcal{N}(v_i)$
    - Für each node $v \in \mathcal{N}(v_i)$
      - If there is a (new) collision-free path from v to v_i, add the vertex (v, v_i) to R
  - Result: R

← Local planning

PRM: Characteristics

- Very well suited for static environments
- Construct the graph once
  - Multiple queries can efficiently be processed **(multi-query)**
- Randomized approach to construct the graph **(probabilistic)**
  - Avoiding exponential growth of the runtime dependent on the dimension of the configuration space
- Strongly depends on the **sampling** strategy. Sampling from a uniform distribution (uniform sampling) may require many samples to correctly represent the connectivity of C_{free}
  - **Problem**: Narrow passages between obstacles
  - **Solution**: Increase sampling close to obstacles
- Not complete, as the graph only approximates the free space C_{free}
  - **Problem**: Potentially, valid paths are not found
  - **Solution**: Extend the graph, so that it is connected, and andy point in C_{free} can be reached directly from one node

PRM: Different Sampling Strategies

- **Random**:
  - Configurations are randomly sampled and checked for collision
- **Grid**:
  - Configurations are generated with discrete resolution
  - Resolution of cells determined hierarchically
- **Halton**:
  - Halton sequence: Determine a set of points that cover an area better than a grid
  - Based on the mathematical concept of discrepancy
- **Call-based**:
  - Sample within cells of decreasing size
  - Size is reduced in each iteration (e.g., to 1/8th)

Geraerts, Roland, and Mark H. Overmars. "A comparative study of probabilistic roadmap planners." Algorithmic Foundations of Robotics V. Springer Berlin Heidelberg, 2004. 43-57.

Content

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - **Dynamic Roadmaps (DRM)**
  - Rapidly-exploring Random Trees (RRT)
  - Extensions of RRT
    - Constrained RRT
    - RRT*
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

Dynamic Roadmaps (DRM)

Multiple queries **(multi-query)** for given kinematic chain

1. Step: Preprocessing

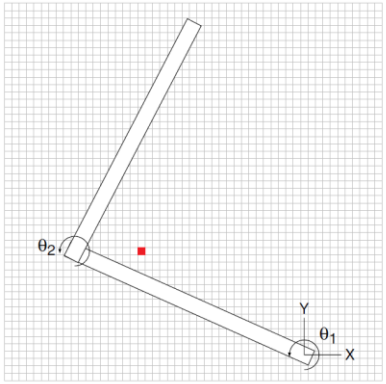
- Approximation of the configuration space by a roadmap (graph)
- Approximation of the work space by voxels (cubes)
- Mapping ϕ_{WC}: voxels → roadmap (nodes, edges)

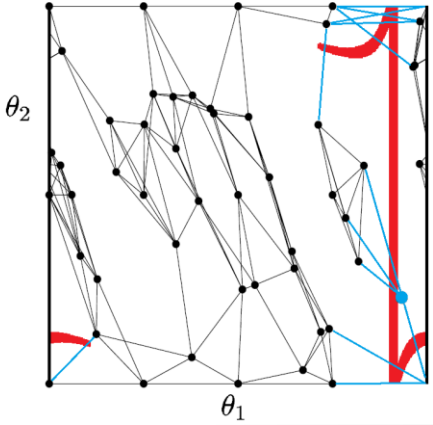
2. Step: Query

- Identify voxels with obstacle
- Adjust the roadmap
- Plan within the adjusted roadmap

DRM: Preprocessing

- **Sampling:** create a collision-free roadmap by sampling random points
  - Many points needed to find solutions in different environments
- **Mapping ϕ_{WC} :** check for collisions between all nodes/edges and all voxels of the work space (high computational effort)

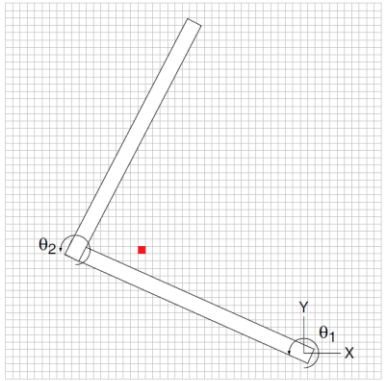


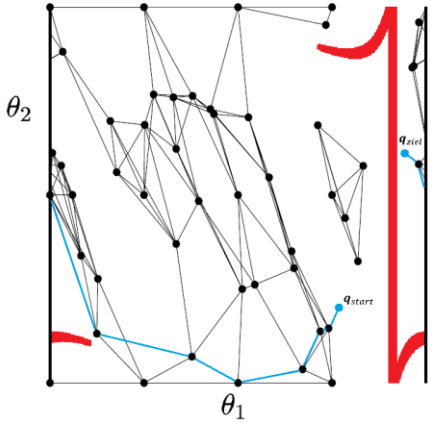


Leven, Peter, and Seth Hutchinson. "A framework for real-time path planning in changing environments." The International Journal of Robotics Research 21.12 (2002): 999-1030.

DRM: Query

- Identify all voxels with obstacle
- Delete all related edges and nodes of the roadmap
  - Related edges and nodes are determined using Φ_{WC}
- Connect q_{start} and q_{goal} to the graph
- Search a path from q_{start} to q_{goal} within the graph



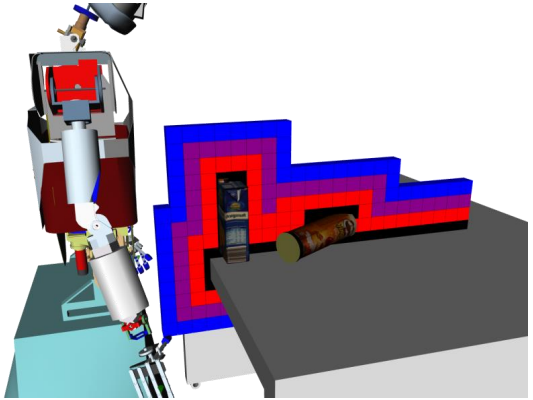


Leven, Peter, and Seth Hutchinson. "A framework for real-time path planning in changing environments." The International Journal of Robotics Research 21.12 (2002): 999-1030.

Distance Aware Dynamic Roadmap (DA-DRM)

- DRM searches for the shortest path → Path often is close to objects (obstacles)
- How can we take the distance to obstacles into consideration?
  - Define **safety margin** and delete all voxels within the safety margin
  - Calculate distances to obstacles for voxels that are close to obstacles
  - Assign a **higher weight** to edges of the roadmap which pass through such voxels

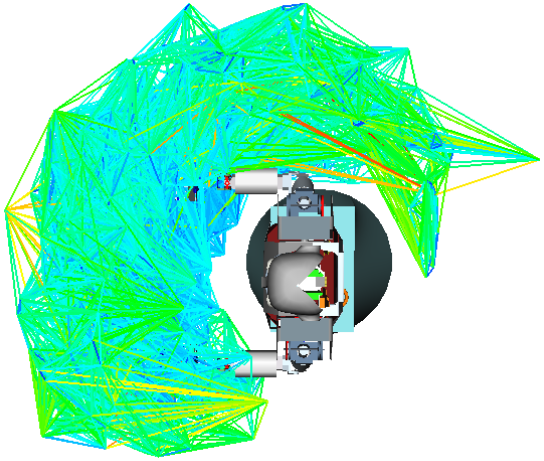
    → Shortest path within the roadmap but with higher distance to objects in the task space



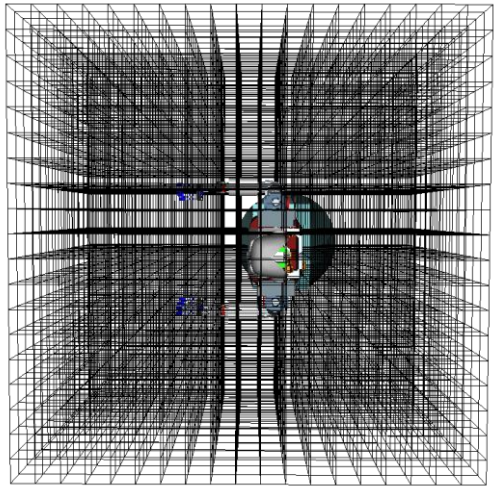
Knobloch, A., Vahrenkamp, N., Wächter, M. and Asfour, T., *Distance-Aware Dynamically Weighted Roadmaps for Motion Planning in Unknown Environments*, IEEE Robotics and Automation Letters (RA-L), vol. 3, no. 3, pp. 2016-2023, July, 2018

Distance Aware Dynamic Roadmap (DA-DRM)

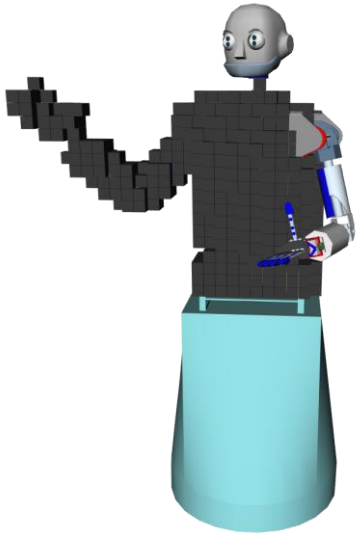
- Graph $G = (V, E)$
- Workspace to C-space mapping $\Phi_{WC}: W \rightarrow V \cup E$



Graph $G = (V, E)$

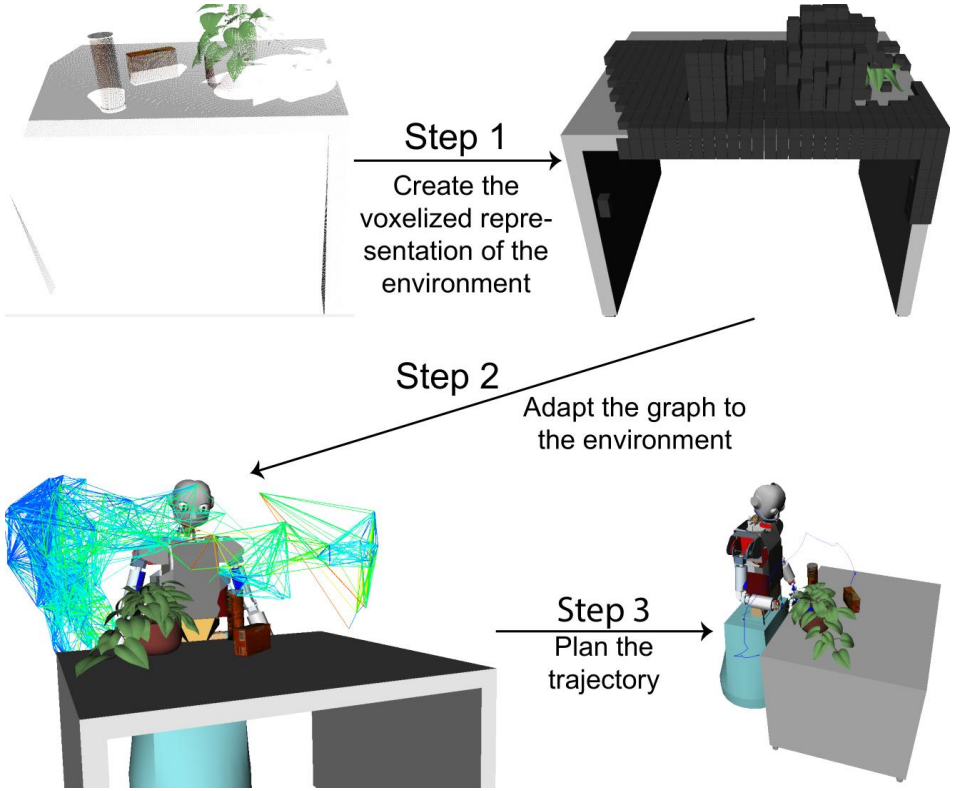


voxelized workspace W



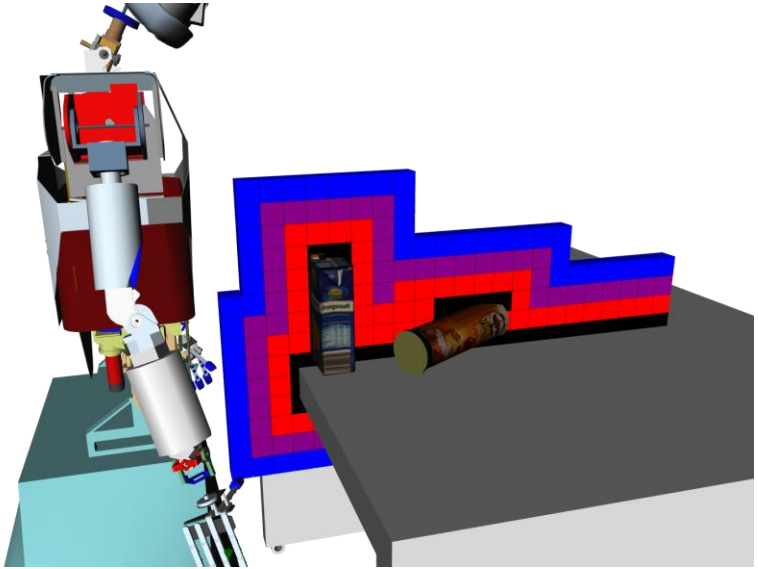
mapping Φ

Distance Aware Dynamic Roadmap (DA-DRM)



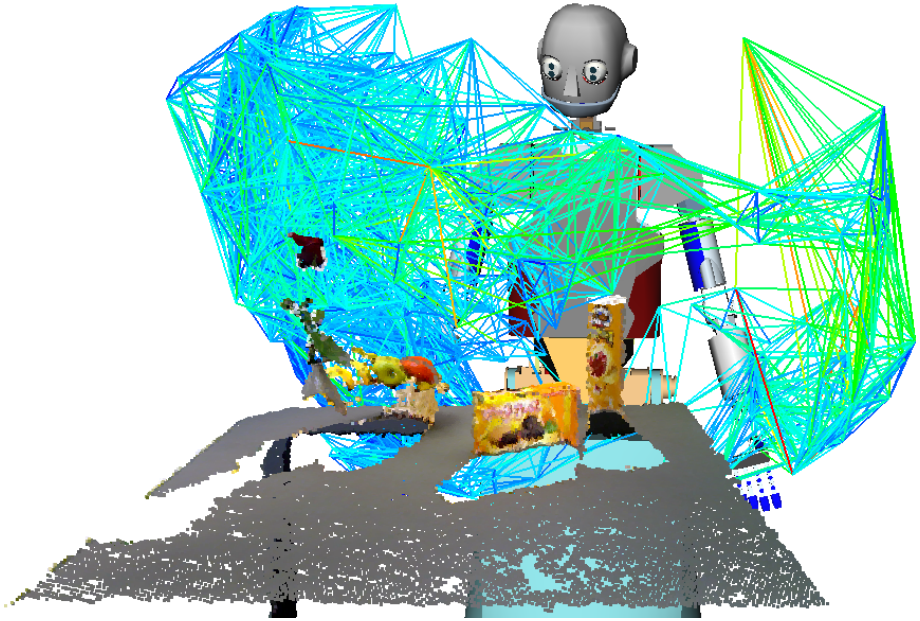
Motion Planning with DRM

- Instead of only binary (blocked/unblocked) information, store also the distance to obstacles in the graph → **weighted graph**



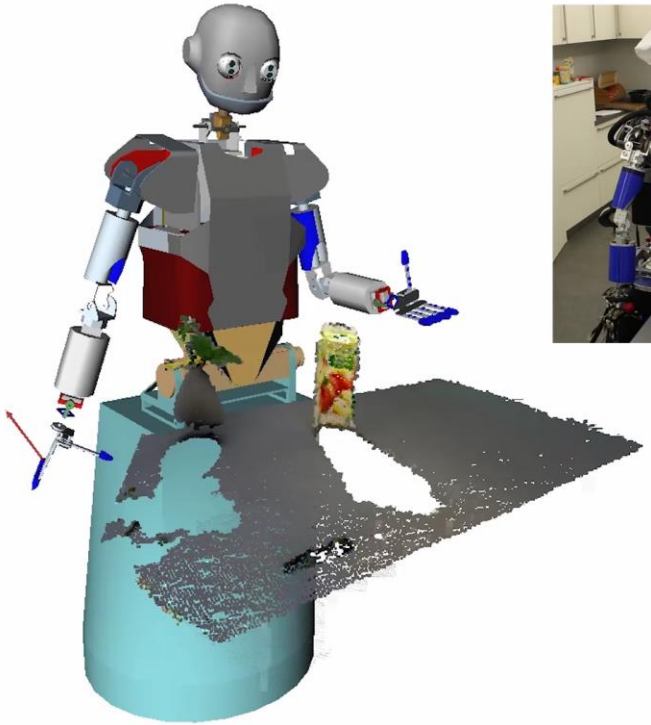
Motion Planning with DRM

- Search a trajectory within the weighted graph



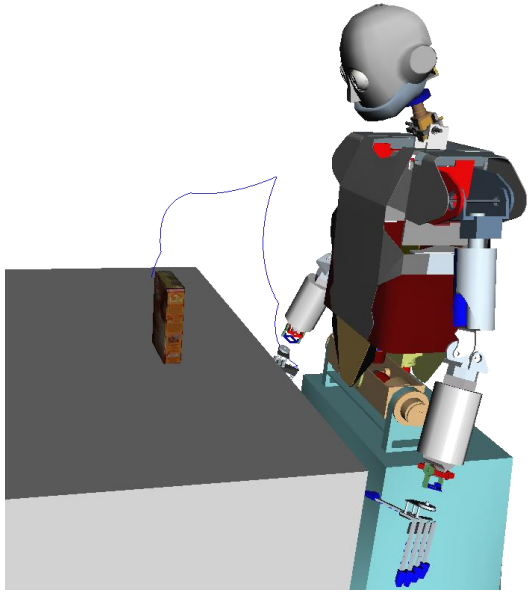
Execution on ARMAR-IIIa

2x

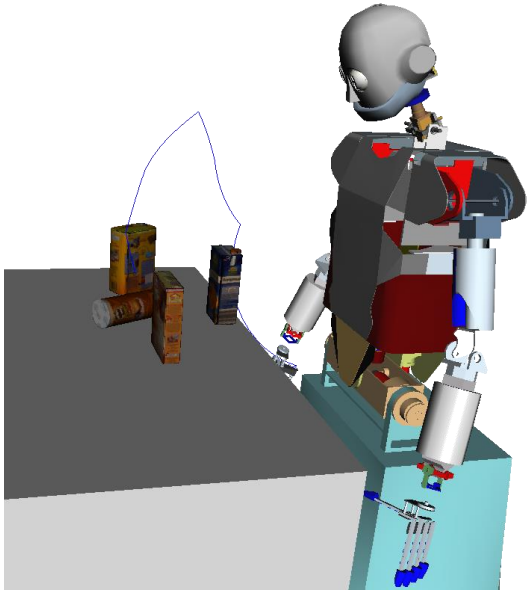


Evaluation

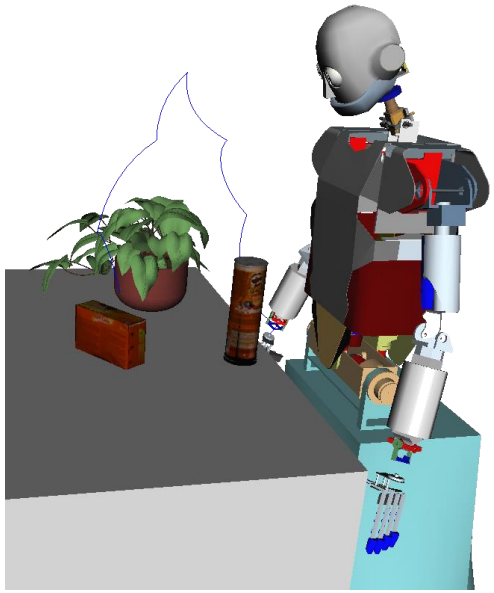
- Comparison RRT, DRM, DA-DRM



Scene A



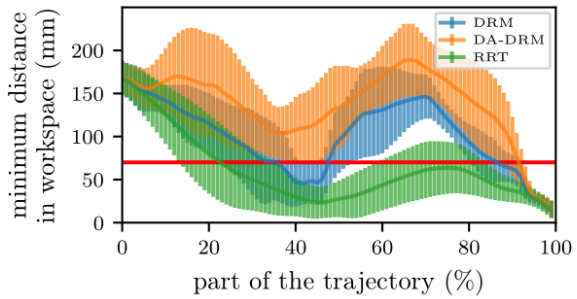
Scene B



Scene C

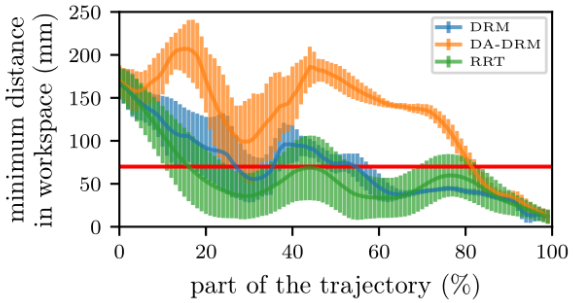
Evaluation

Scene A

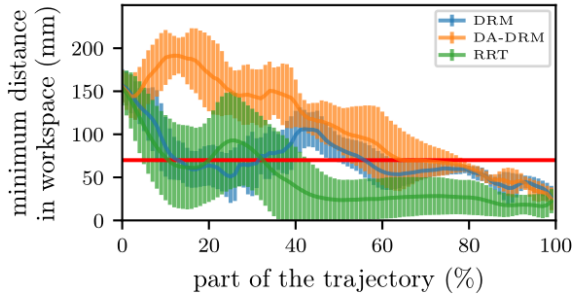


Scene B

Safety distance: 70 mm



Scene C



- RRT: Below 25mm (near to 0mm) obstacle distance
- DRM: Between 25mm and 50mm obstacle distance
- DA-DRM: Keeps the safety distance of 70mm until the end

Content

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - **Rapidly-exploring Random Trees (RRT)**
  - Extensions of RRT
    - Constrained RRT
    - RRT*
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

Rapidly-exploring Random Trees (RRTs)

- In contrast to PRMs
  - Algorithm for **single query**
  - **No preprocessing** needed
  - No problems with changing (dynamic) environments / kinematic chains
- **Probabilistically complete, randomized algorithm**
  - No guarantee that a solution is found within a time limit
  - If a solution exists, it will be found (runtime to infinity)
  - Does not terminate if no solution exists
- Efficient for high-dimensional problems
- Extensions of standard RRT for specific problems, e.g., narrow passages

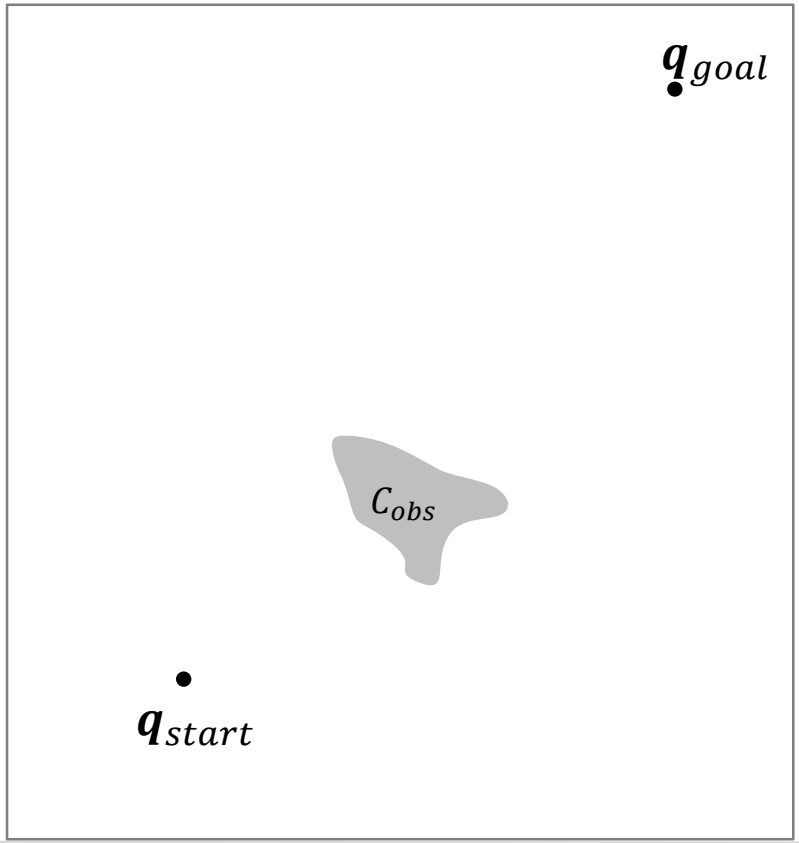
http://lavalle.pl/rrt

James Kuffner & Stephen LaValle

RRT: Principle (1)

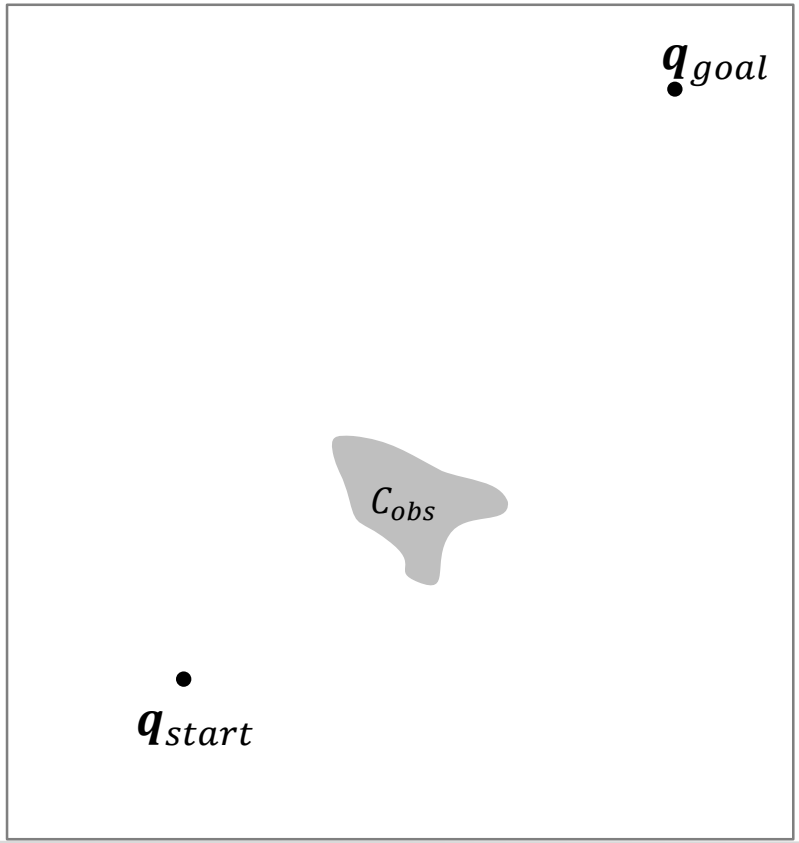
- The shape of C_{obs} in the configuration space is **unknown**

- Initialization of the RRT
  - Create empty tree T
  - Insert $\boldsymbol{q}_{start}$ into T



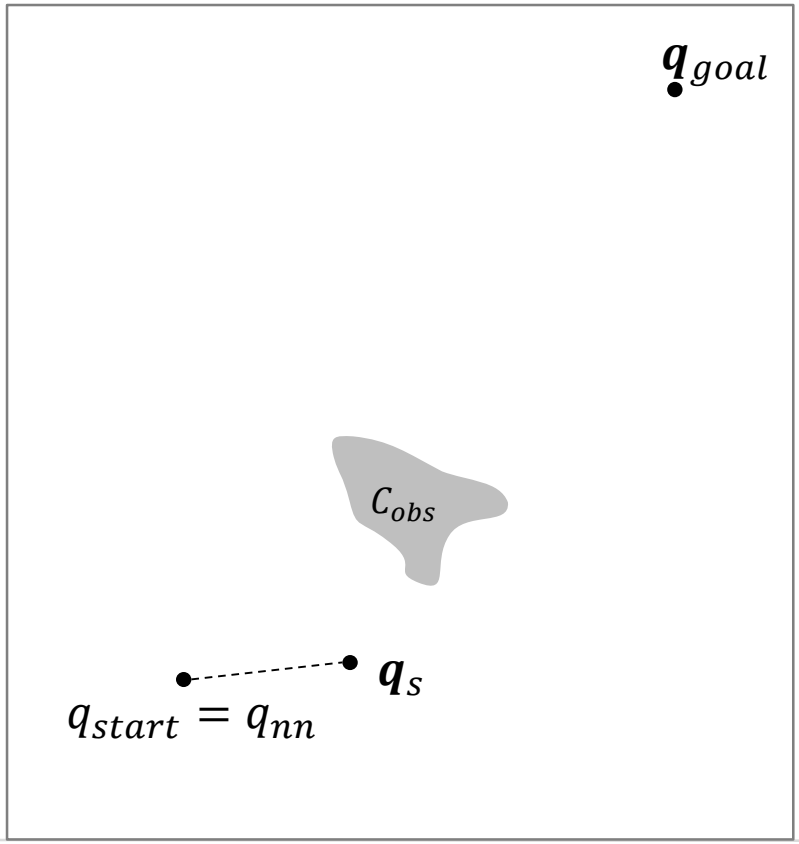
RRT: Principle (2)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add nodes on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



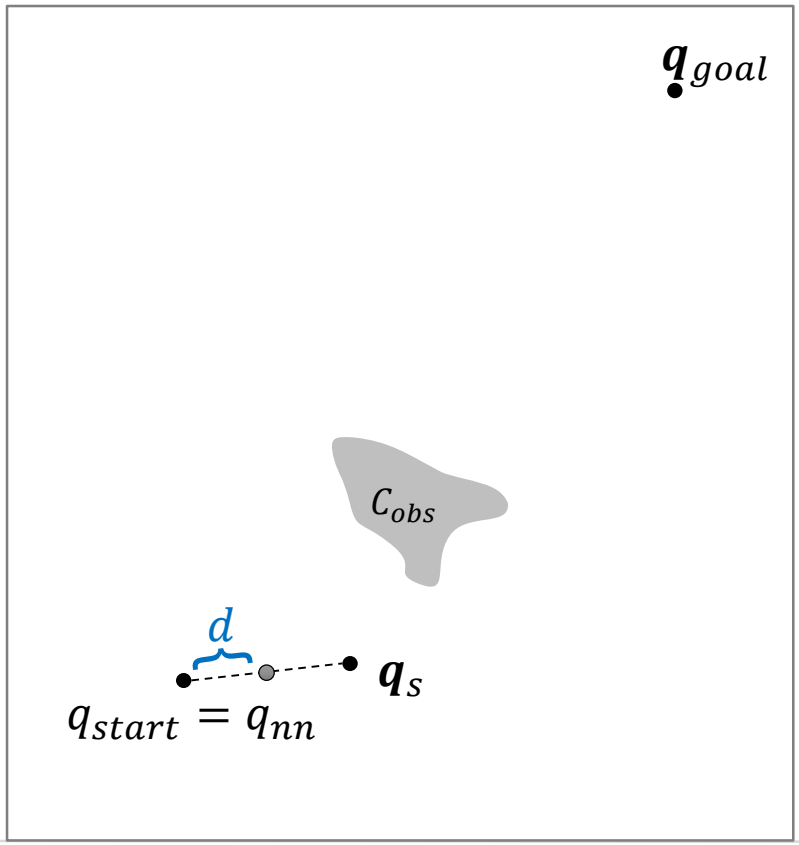
RRT: Principle (3)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



RRT: Principle (4)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.

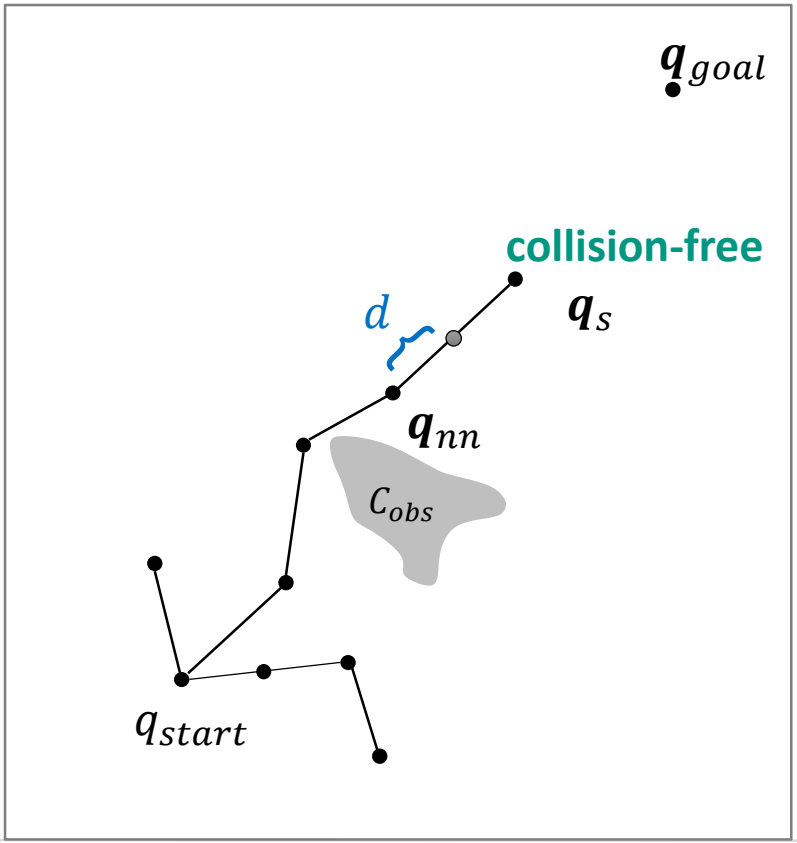


RRT: Principle (5)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.

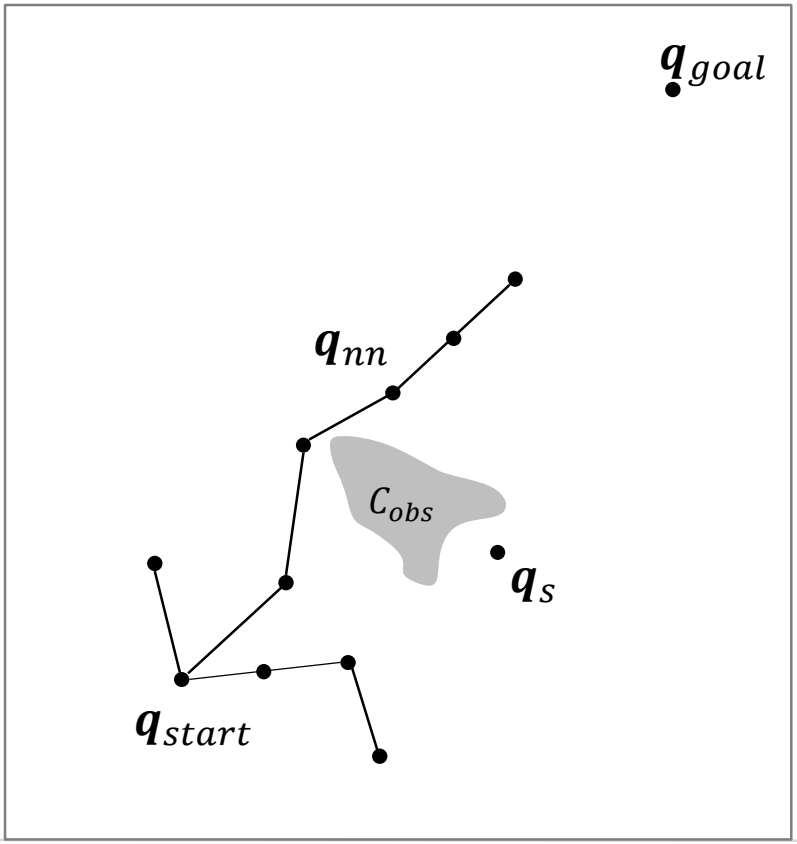
RRT: Principle (6)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



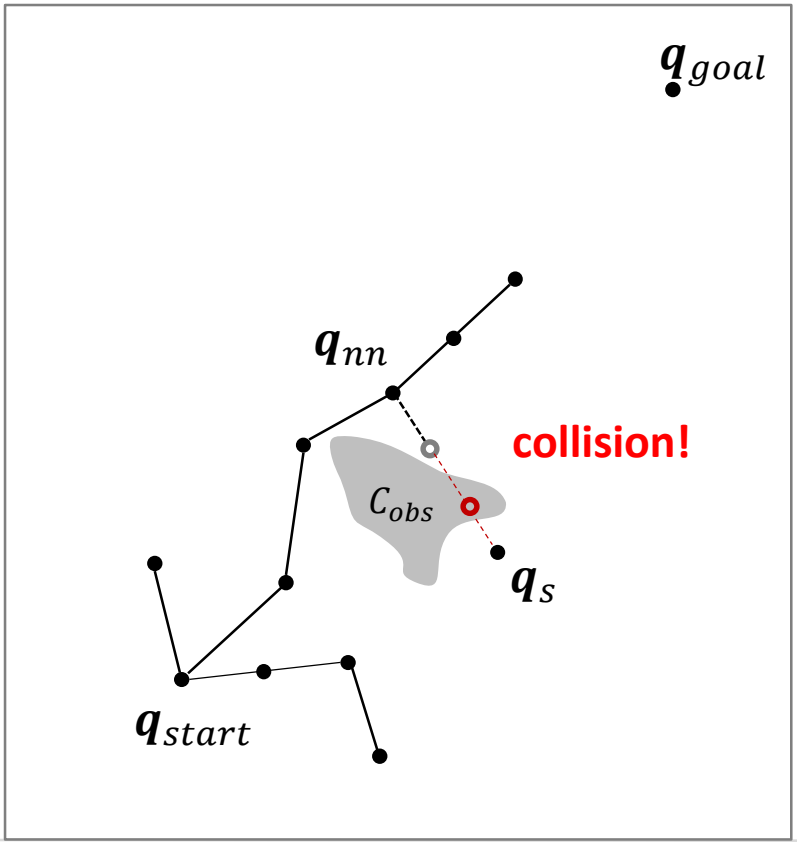
RRT: Principle (7)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



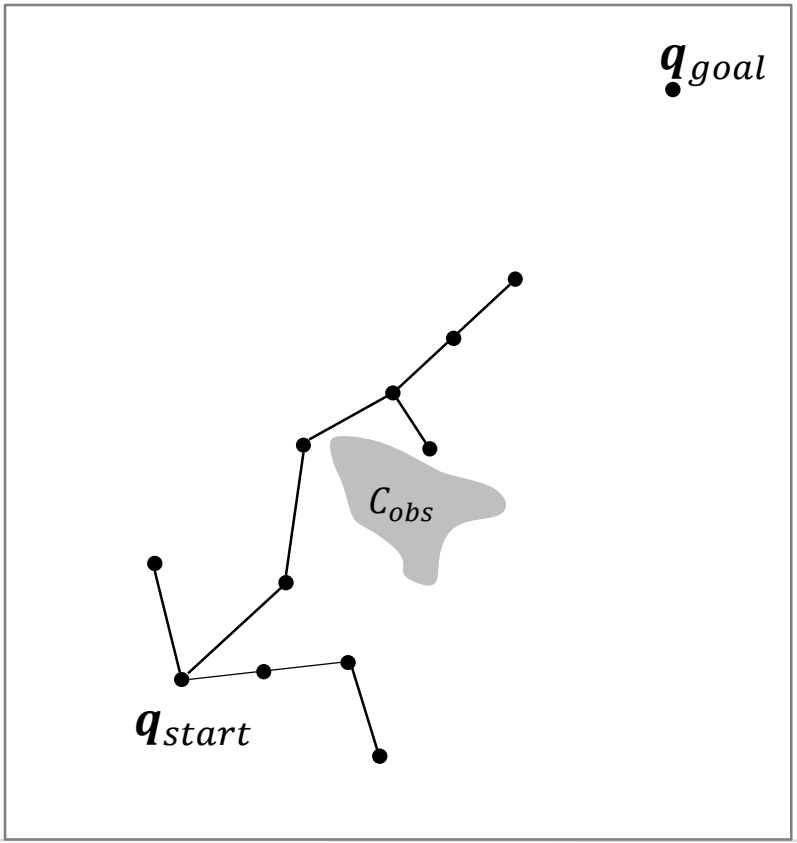
RRT: Principle (8)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



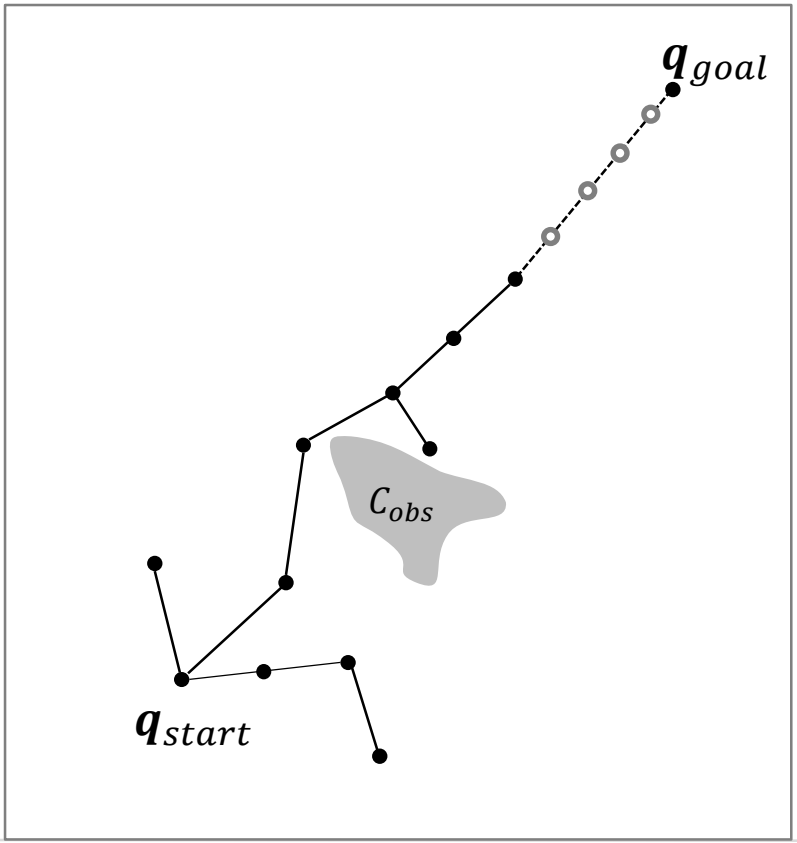
RRT: Principle (9)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.



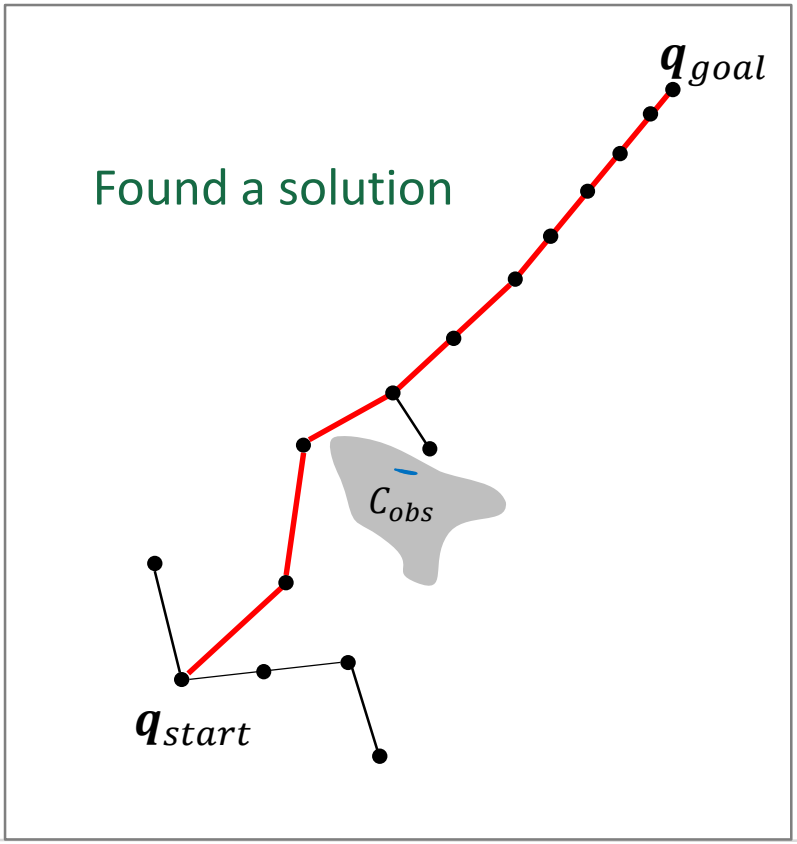
RRT: Principle (10)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.
- Check in every step whether $\boldsymbol{q}_{goal}$ can be connected to T



RRT: Principle (11)

- Iteration
  1. Sample a point $\boldsymbol{q}_s$
  2. Determine the next neighbor $\boldsymbol{q}_{nn}$ in T
  3. Add points on the connection between $\boldsymbol{q}_s$ and $\boldsymbol{q}_{nn}$ to T
     - With step size d
     - Check every part of the path for collision with C_{obs}
     - Stop when a collision has been detected
  4. Go to 1.
- Check in every k^{th} step whether $\boldsymbol{q}_{goal}$ can be connected to T



RRT: Probabilistic Completeness (1)

- RRTs are probabilistically complete
- **Intuition**: T spreads out evenly in the configuration space

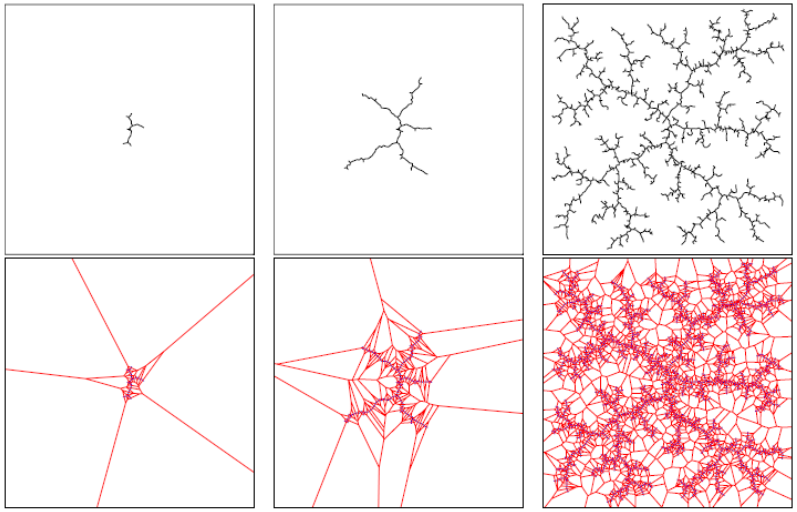
by Javed Hossain
[CC BY-SA 3.0], via
Wikimedia Commons

RRT: Probabilistic Completeness (2)

- The probability of a node in T to be expanded is proportional to the size of its Voronoi region. Larger Voronoi regions are favored for tree expansion.

RRT

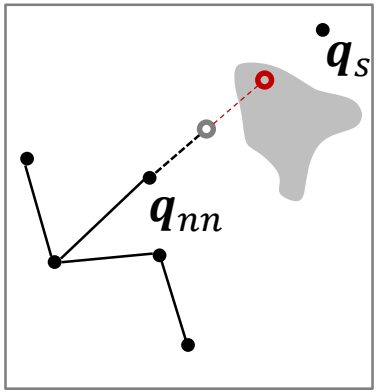
Voronoi regions of the RRT tree's nodes



J. J. Kuffner and S. M. LaValle, *RRT-connect: An efficient approach to single-query path planning, Proceedings IEEE International Conference on Robotics and Automation.* 2000, pp. 995-1001

RRT: Collision Detection (1)

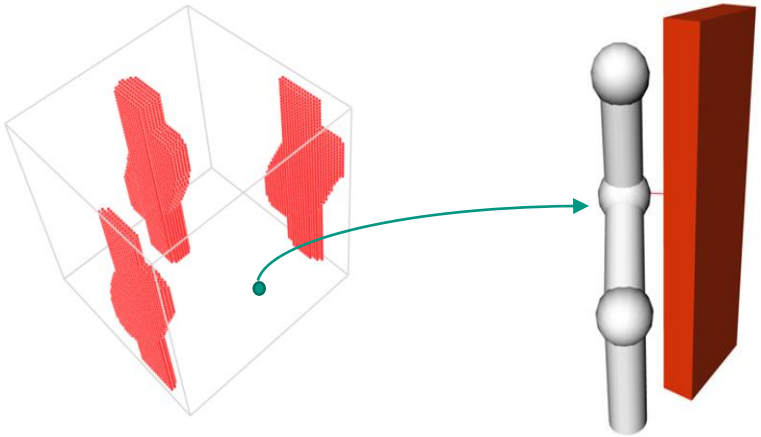
- C_{obs} is not known; how to check whether $\boldsymbol{q}_s$ can be reached without collision?

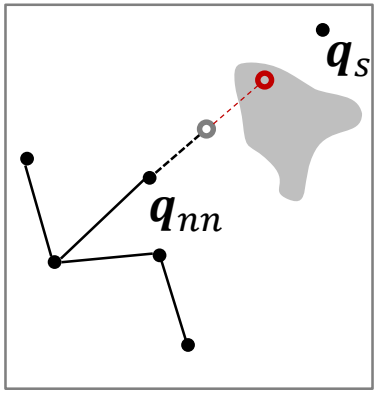


collision!

RRT: Collision Detection (2)

- C_{obs} is not known; how to check whether $\boldsymbol{q}_s$ can be reached without collision?
- Every tree node $\boldsymbol{q} \in C$ describes a robot configuration
  - **Perform collision detection in task space**

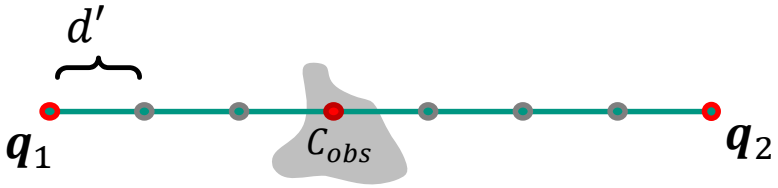


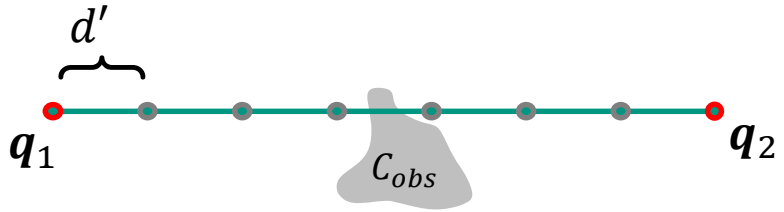


collision!

RRT: Collision Detection (3)

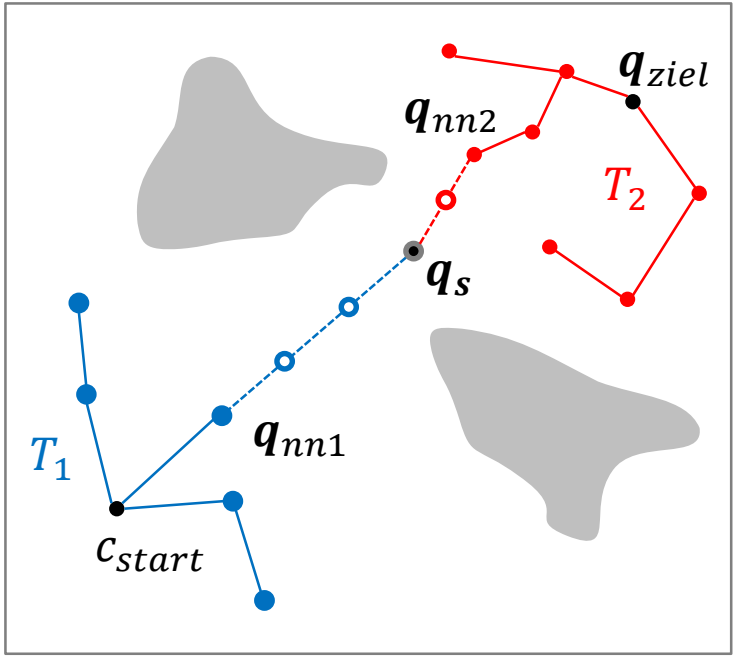
- Also connecting path segments need to be collision free
  - **Continuous Collision Detection** (CCD): Exact, but slow
  - Sampling-based approach
    - Check nodes along path segments
    - Fast (incremental distance calculations!), but not exact
    - Results depend on **sampling distance** d'





Bi-Directional RRTs

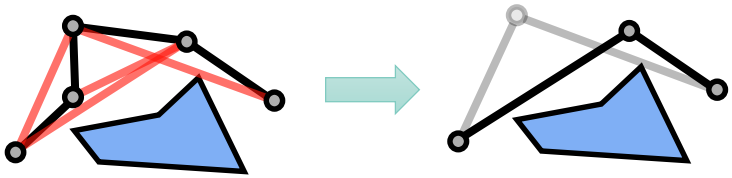
- **Two trees:**
  - T_1 starting from $\boldsymbol{q}_{start}$
  - T_2 starting from $\boldsymbol{q}_{goal}$
- Sampled points $\boldsymbol{q}_s$ expand both trees:
  - $\boldsymbol{q}_{nn,1}$ (nearest neighbor in T_1)
  - $\boldsymbol{q}_{nn,2}$ (nearest neighbor in T_2)
- A solution is found if both trees are connected to $\boldsymbol{q}_s$
- Original version similar to **RRT connect**

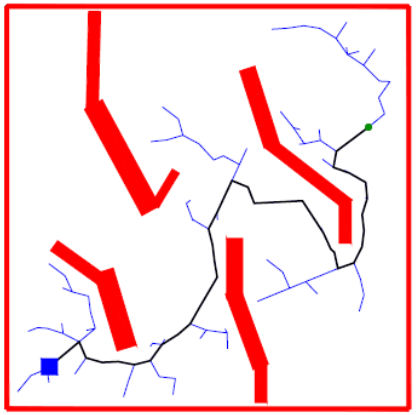


J. J. Kuffner and S. M. LaValle, *RRT-connect: An efficient approach to single-query path planning, Proceedings IEEE International Conference on Robotics and Automation.* 2000, pp. 995-1001

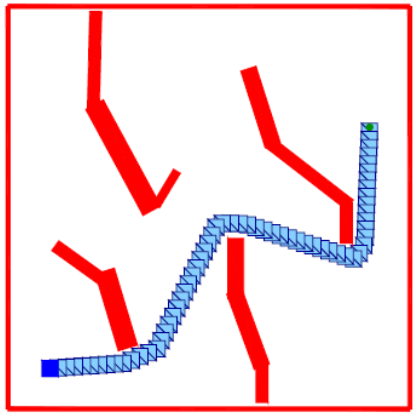
RRT: Post-Processing (Smoothing)

- Solutions can be improved by post-processing
  - Randomly choose two nodes of the solution‘s path
  - If the connection between these nodes is collision-free, add an edge between them and delete all nodes of the solution‘s path in between
  - Result: smooth trajectories





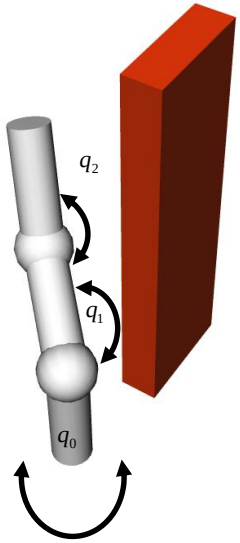
Path of original solution



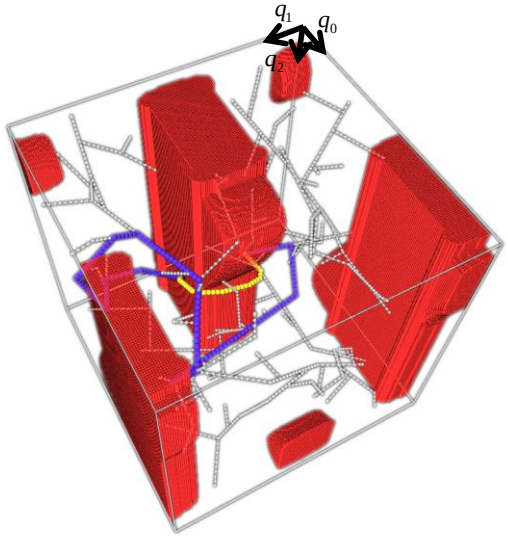
Path of smoothed solution

RRT: Example

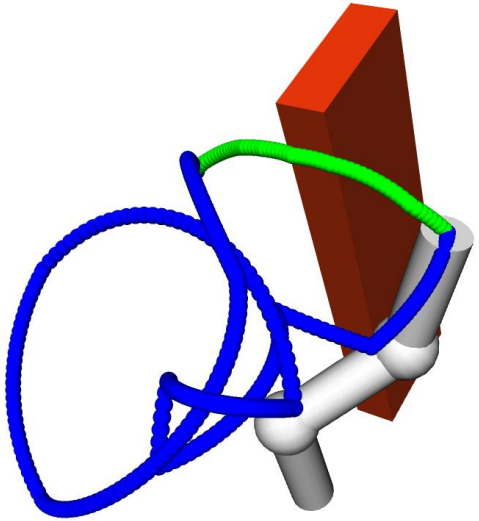
- Example with 3D configuation space



Start



RRT in configuration space
(red: C_{obs})



Found solution (blue) and
smoothed solution (green)

Content

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - Rapidly-exploring Random Trees (RRT)
  - **Extensions of RRT**
    - **Constrained RRT**
    - RRT*
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

Constrained RRT (1)

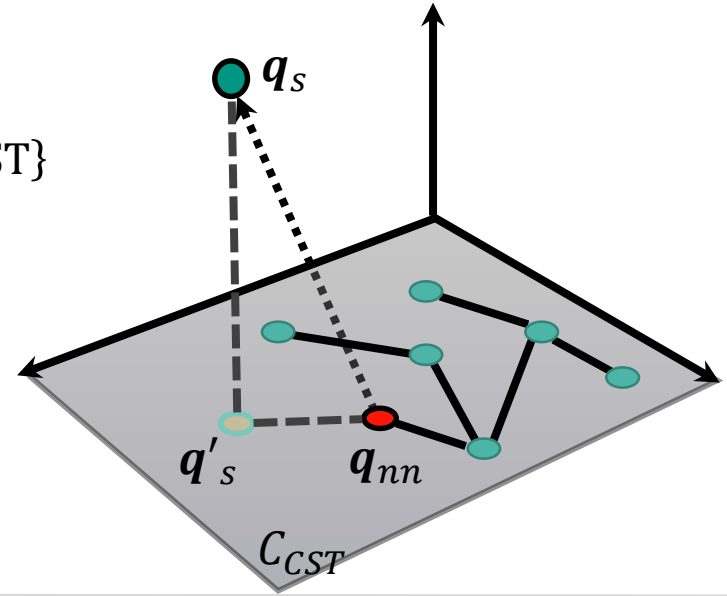
- Motion planning must satisfy multiple **constraints** to generate valid trajectories
  - Constant orientation of the end-effector
  - Balance of a bipedal robot
- **Constraints Space: $C_{CST} \subseteq C$**
- Constraints define lower-dimensional subsets (manifolds) in the configuration space
  - **Example:** The set of all configurations q that fulfill a constraint build a **plane in the three-dimensional configuration space**
  - Sampling-based approaches can not, by design, fulfill these constraints

Constrained RRT (2)

- **Idea**: Project a sample $\boldsymbol{q}_s$ to the configuration $\boldsymbol{q}'_s$, where q's satisfies all specified constraints
- **Example**: Let a constraint CST constitute a 2D manifold in $C_{CST} \subseteq C$

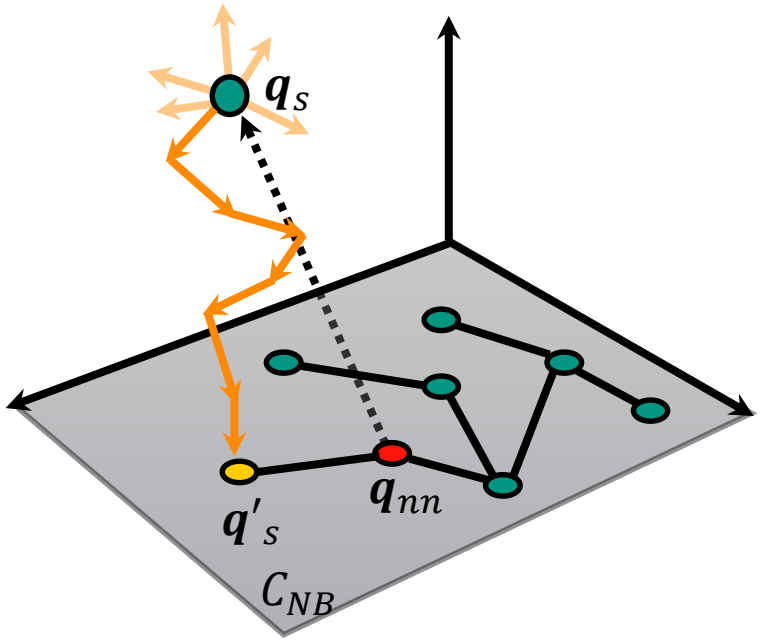
$$C_{CST} = \{\boldsymbol{q} \in C : \boldsymbol{q} \text{ statisfies CST}\}$$

- **Problem**: How to perform the projection?
  - Randomized Gradient Descent
  - First Order Retraction



Constrained RRT: Randomized Gradient Descent

- Tolerance for constraint: α
- Randomly determine n neighbors of $\boldsymbol{q}_s$ (in hyper sphere with radius d_{max})
- If a neighbor‘s distance to C_{CST} is smaller than the distance of $\boldsymbol{q}_s$ to C_{CST}, replace $\boldsymbol{q}_s$ by this neighbor
- Repeate until a maximum number of iterations is reached, or the distance of $\boldsymbol{q}_s$ to C_{CST} is smaller than α
- **Distance to C_{CST} in task space is required**
- **No directional information required**



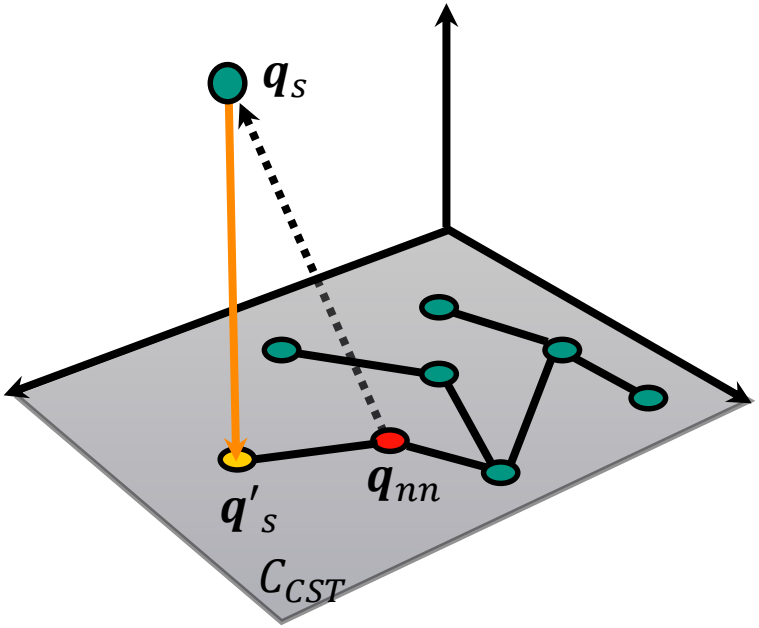
Constrained RRT: First Order Retraction

- Tolerance for constraint: α
- Jacobian matrix J provides directional information
- Calculation as for the inverse kinematics

$$\boldsymbol{q}'_s = \boldsymbol{q}_s - J(\boldsymbol{q}_s)^{\#}\Delta\boldsymbol{x}_s$$

$\Delta\boldsymbol{x}_s$ is the distance of $\boldsymbol{q}'_s$ distance to C_{CST} **in task space**

- **Distance measure to C_{CST} in task space is required**



M. Stilman, "*Global manipulation planning in robot joint space with task constraints*," IEEE Transactions on Robotics and Automation, vol. 26, no. 3, pp. 576–584, 2010.

Content

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - Rapidly-exploring Random Trees (RRT)
  - **Extensions of RRT**
    - Constrained RRT
    - **RRT***
    - Narrow passages
      - Dynamic Domain RRT
      - Bridge Sampling

RRT*

- **Problem:** RRTs yield trajectories that are usually not optimal
- **RRT*** optimizes the search space iteratively during the search

  ⇒ With sufficient time, the optimal path between q_{start} and q_{goal} is found
  ⇒ asymptotic optimality
- Optimization of the search tree in two steps:
  - Calculate **costs** of each new node (e.g., length of the path from the start node)
  - **Rewiring** of the search tree by adding new nodes
- **Disadvantage:**
  - Longer runtime (up to a factor of 30 in comparison to uni-directional RRT)
  - Uni-directional algorithm

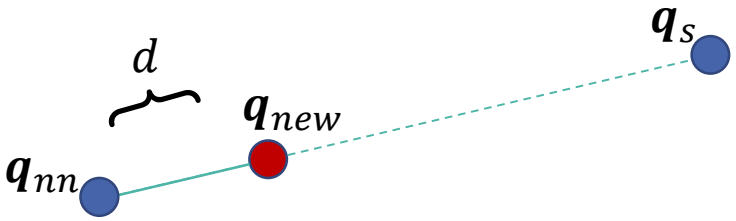
S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, Jan. 2011.

RRT*: Algorithm

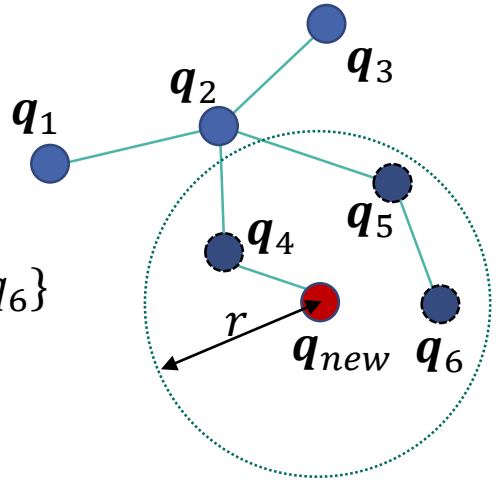
1. $\boldsymbol{q}_s = SampleRandom(C)$ // Sample random configuration
2. $\boldsymbol{q}_{nn} = NearestNeighbor(\boldsymbol{q}_s, T)$ // Determine the nearest neighbor
3. $\boldsymbol{q}_{new} = \boldsymbol{Steer}(\boldsymbol{q}_{nn}, \boldsymbol{q}_s, d)$ // Go a step in the direction of $\boldsymbol{q}_s$
4. $if\ !CollisionFreePath(\boldsymbol{q}_{nn}, \boldsymbol{q}_{new}): goto\ 1$ // Is the path collision-free?
5. $Q_{near} = \boldsymbol{Near}(T, \boldsymbol{q}_{new}, r)$ // All nodes within a maximum distance r to $\boldsymbol{q}_{new}$
6. $\boldsymbol{q}_{min} = \boldsymbol{MinCostPath}(Q_{near}, \boldsymbol{q}_{new})$ // $Cost(\boldsymbol{q}_{min}) + Cost(\boldsymbol{q}_{min}, \boldsymbol{q}_{new})$ minimal
7. $AddPath(T, \boldsymbol{q}_{min}, \boldsymbol{q}_{new})$ // Add path from $\boldsymbol{q}_{min}$ to $\boldsymbol{q}_{new}$
8. $\boldsymbol{Rewire}(T, \boldsymbol{q}_{new}, Q_{near})$ // Check edges to nodes in Q_{near}
9. $if\ !Timeout: goto\ 1$ // Next iteration

RRT*: Functions (1)

- $\boldsymbol{q}_{new} = Steer(\boldsymbol{q}_{nn}, \boldsymbol{q}_s, d)$
  - Go from $\boldsymbol{q}_{nn}$ towards $\boldsymbol{q}_s$ with a step size d
  - Check for collision
  - Create node $\boldsymbol{q}_{new}$



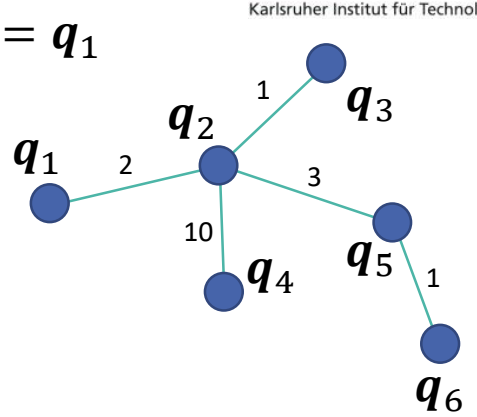
- $Q_{near} = Near(T, \boldsymbol{q}_{new}, r)$
  - Determine all nodes in T within a maximum distance r to $\boldsymbol{q}_{new}$

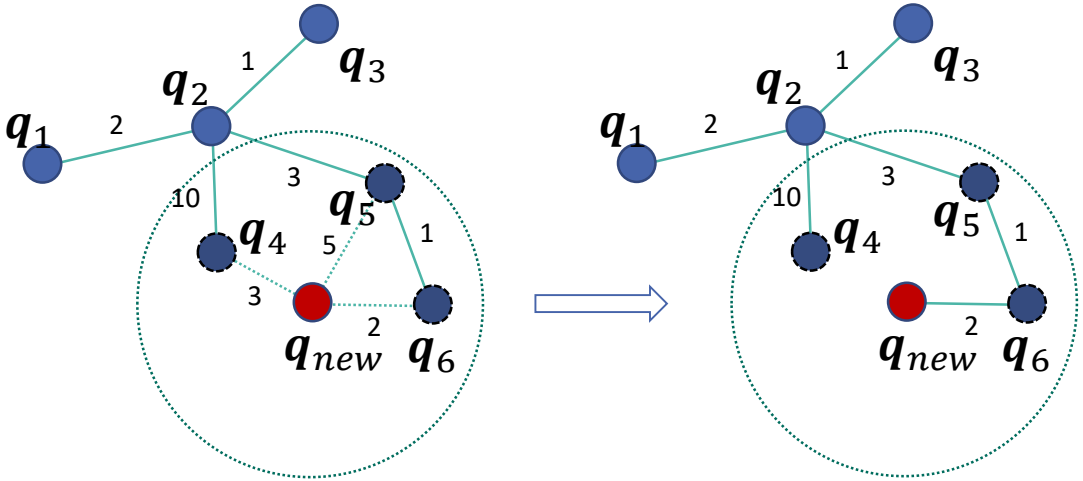


$Q_{near} = \{q_4, q_5, q_6\}$

RRT*: Functions (2)

- $Cost(q_i)$: Costs from q_{start} to q_i
- $Cost(q_a, q_b)$: Costs from q_a to q_b
- $q_{min} = MinCostPath(Q_{near}, q_{new})$: Determine $q_{min} \in Q_{near}$ so that the costs $Cost(q_{min}) + Cost(q_{min}, q_{new})$ of the path are minimal and the path is collision-free





$Cost(q_6)$
$= 2 + 3 + 1 = 6$

$q_{min} = q_6$

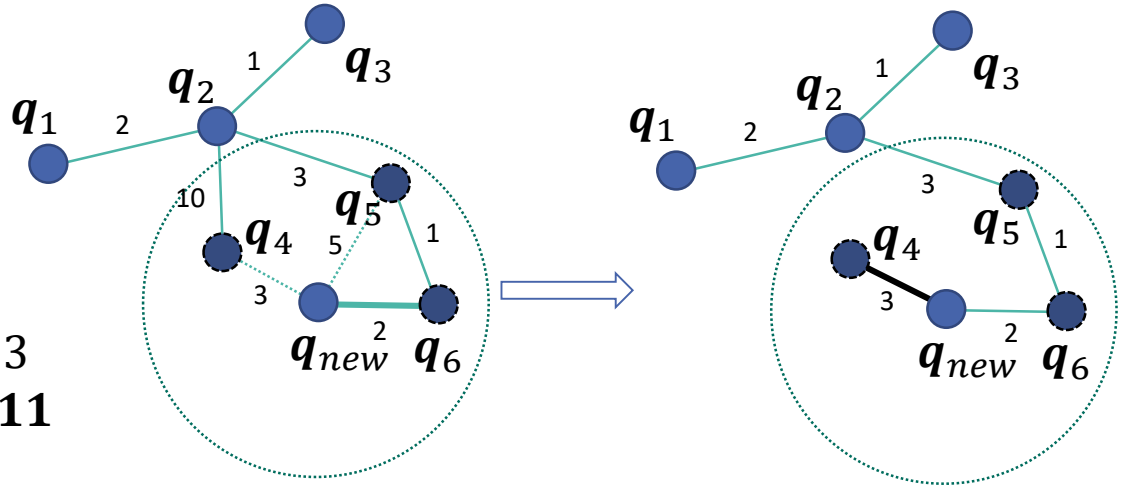
RRT*: Rewiring

- $Rewire(T, \boldsymbol{q}_{new}, Q_{near})$: Check for all $\boldsymbol{q}_{near} \in Q_{near}$ whether

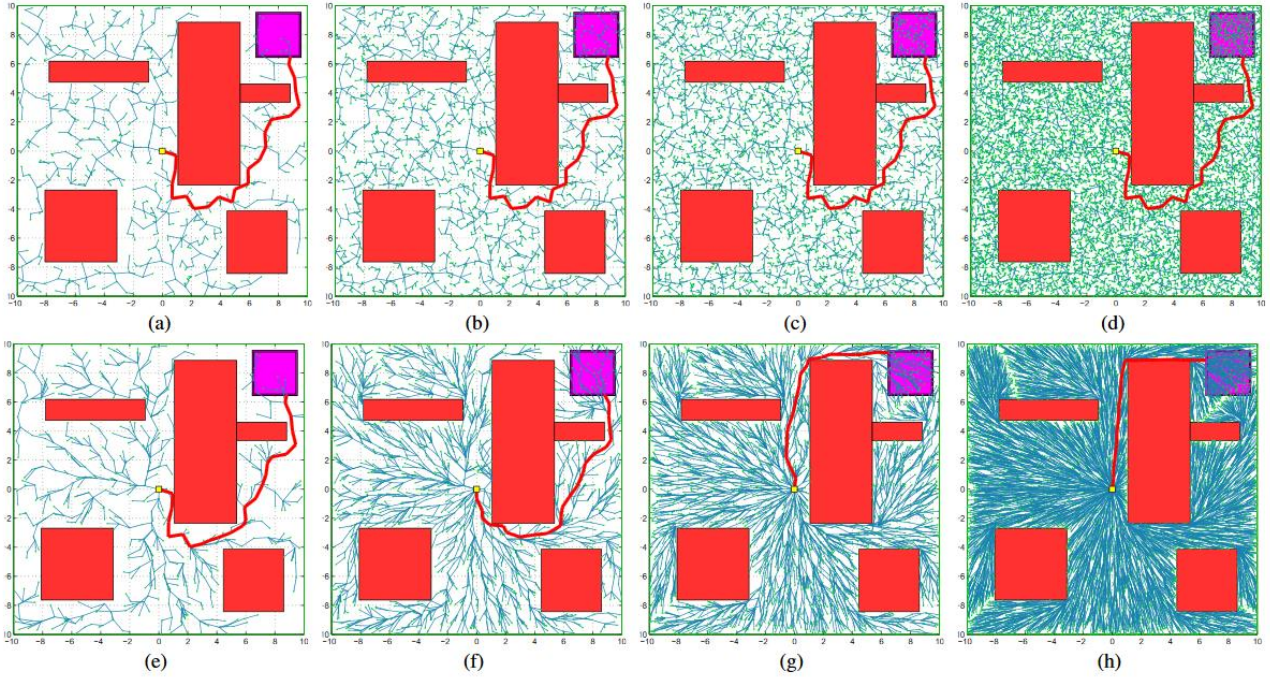
$$Cost(\boldsymbol{q}_{new}) + Cost(\boldsymbol{q}_{new}, \boldsymbol{q}_{near}) < Cost(\boldsymbol{q}_{near})$$

- Update connection to nodes in $\boldsymbol{q}_{near}$ (if cheaper and collision-free)

$Cost(q_4) = 12$
$Cost(q_6) = 6$
$Cost(q_{new}) = 8$
$Cost(q_{new}) + Cost(q_{new}, q_5) = 13$
$\boldsymbol{Cost(q_{new}) + Cost(q_{new}, q_4) = 11}$



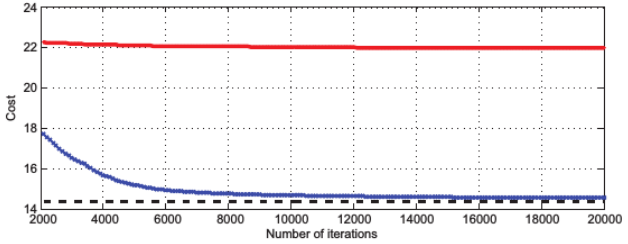
Comparison of RRT and RRT* (1)



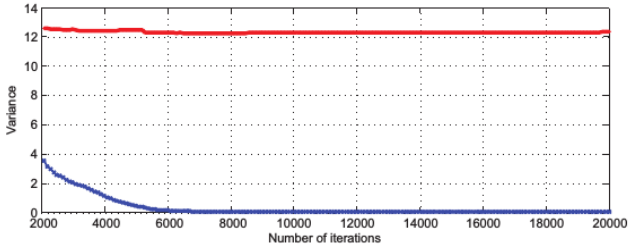
S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, Jan. 2011.

Comparison of RRT and RRT* (2)

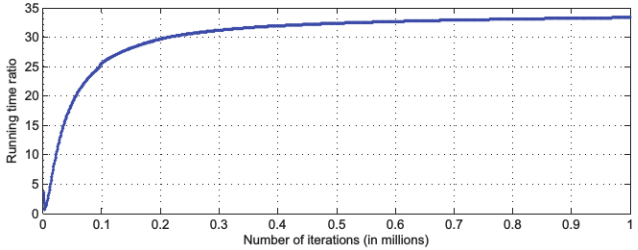
Path costs, averaged over 500 trials
(optimal solution: black)



Variance of the costs



Runtime ratio RRT* / RRT
(per iteration)



RRT

RRT*

S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, Jan. 2011.

Content

- Motivation
- Fundamentals of motion planning
- Path planning for mobile robots
- **Motion planning for manipulators**
  - Probabilistic Roadmaps (PRM)
  - Dynamic Roadmaps (DRM)
  - Rapidly-exploring Random Trees (RRT)
  - **Extensions of RRT**
    - Constrained RRT
    - RRT*
    - **Narrow passages**
      - **Dynamic Domain RRT**
      - **Bridge Sampling**

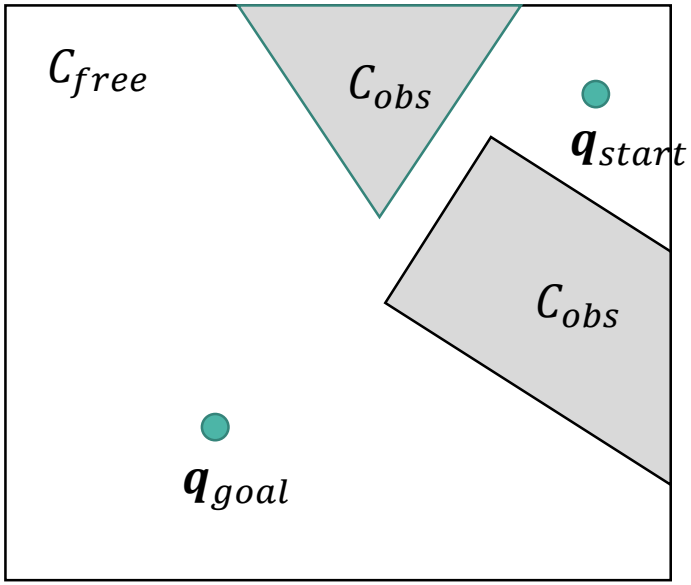
Narrow Passages: Motivation

- Classical RRTs determine new node $\boldsymbol{q}_s$ by **uniform sampling** in the configuration space C
- Result of uniform sampling:
  - **Many rather irrelevant samples**
    e.g., „in the middle of C_{free}“, far away from obstacles
  - **Few interesting samples**
    e.g., close to obstacles, in particular in narrow passages between two obstacles
- Classical RRTs can take much time to find a solution through a narrow passage

- **Main idea: Sampling is much cheapter than collision checks**

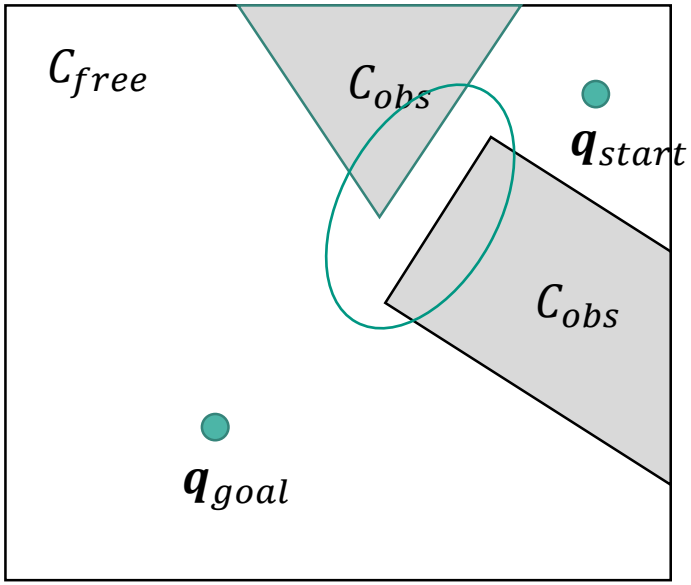
Narrow Passages: Example (1)

- Narrow passages in a 2D configuration space



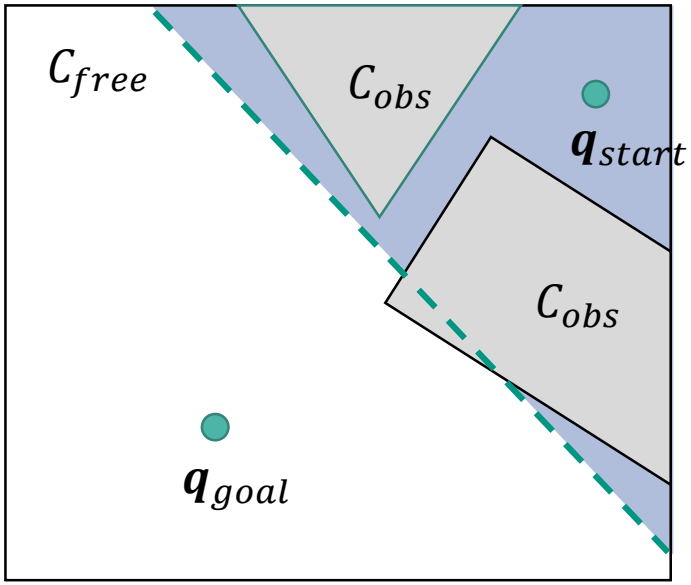
Narrow Passages: Example (2)

- Narrow passages in a 2D configuration space
- Low probability of a new sample to be in the area of the **narrow passage**



Narrow Passages: Dynamic Domain RRT (1)

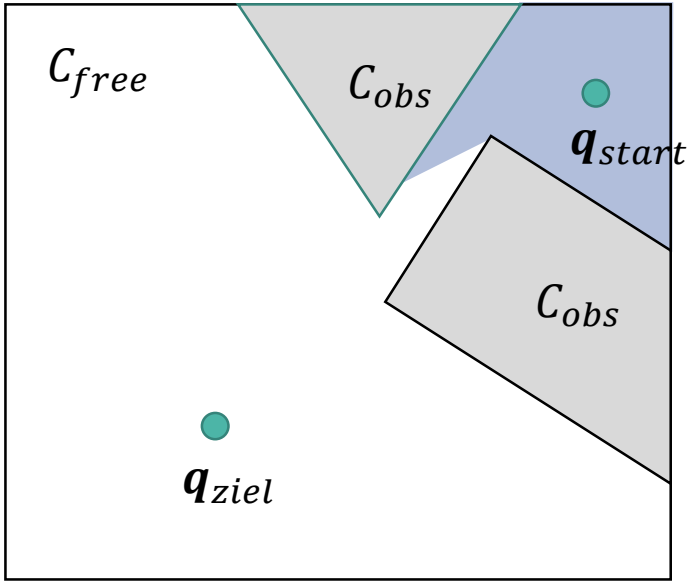
- **Problem:** RRTs do not detect narrow passages und cannot sample purposefully
- **Ideal:** Sample only in the visible Voronoi region of a node



Voronoi region of $\boldsymbol{q}_{start}$

Narrow Passages: Dynamic Domain RRT (2)

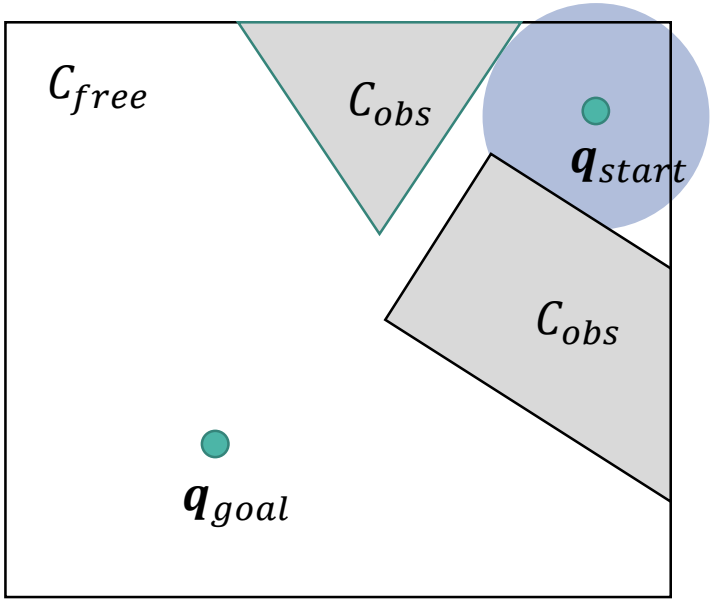
- **Problem:** RRTs erkennen enge Passagen nicht und können nicht zielgerichtet sampeln
- **Ideal:** Sample only in the visible Voronoi region of a node → collision check is not needed



Visible Voronoi region of q_{start}

Narrow Passages: Dynamic Domain RRT (3)

- **But:** Calculation of visible Voronoi regions is expensive
  (no explicit representation of the obstacle regions in the configuration space)
- **Instead:** Approximation of visible Voronoi regions by sphere with radius r
  (**Dynamic Domain**)



approximated visible Voronoi region of $\boldsymbol{q}_{start}$ (Dynamic Domain)

Dynamic Domain RRT (4)

- In the vicinity of obstacles, **Dynamic Domain RRT** limits the sampling area of a node to its Dynamic Domain (DD).
  - Initially, the DD radius r of each node is set to ∞.
    Sampling is performed in the whole Voronoi Region of the node.
  - During the RRT extension step, if no connection to a node can be determined, its DD radius is reduced to a predefined value R. Such nodes are called **boundary nodes**, as they are located at the boundary of C_{free} and C_{obs}.
- **Sampling:** A sample $\boldsymbol{q_s}$ is rejected, if $\boldsymbol{q_s}$ is outside the DD radius of its nearest neighbor $(dist(\boldsymbol{q_s}, \boldsymbol{q_{nn}}) > \boldsymbol{q_{nn}}.r)$

A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3856–3861, Apr. 2005.

Dynamic Domain RRT (5)

- In narrow passages, frequent collision checks are avoided, and no expansion attempts are made to distant and unreachable nodes.

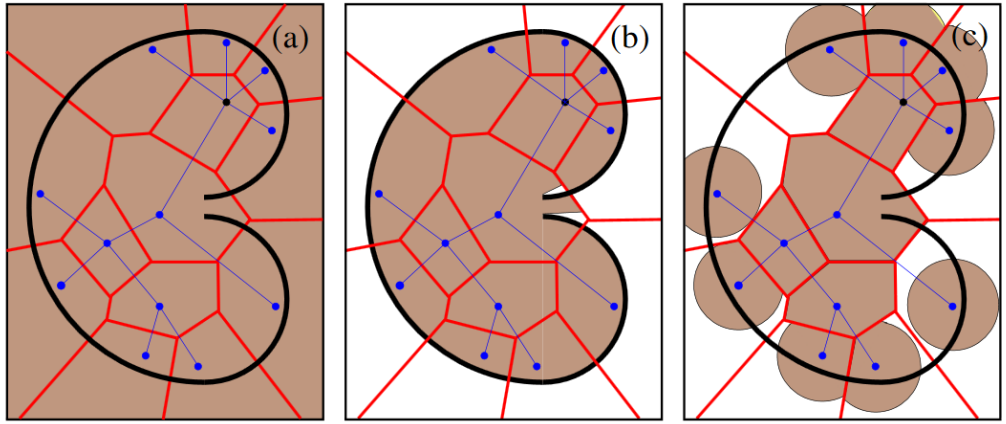
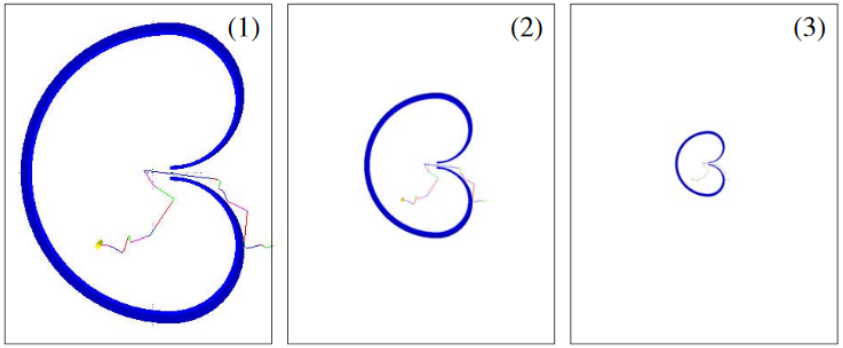


Fig. 3. For a set of points inside a bug trap different sampling domains are shown: (a) regular RRTs sampling domain, (b) visibility Voronoi region, (c) dynamic domain.

A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3856–3861, Apr. 2005.

Dynamic Domain RRT – Comparison to RRT

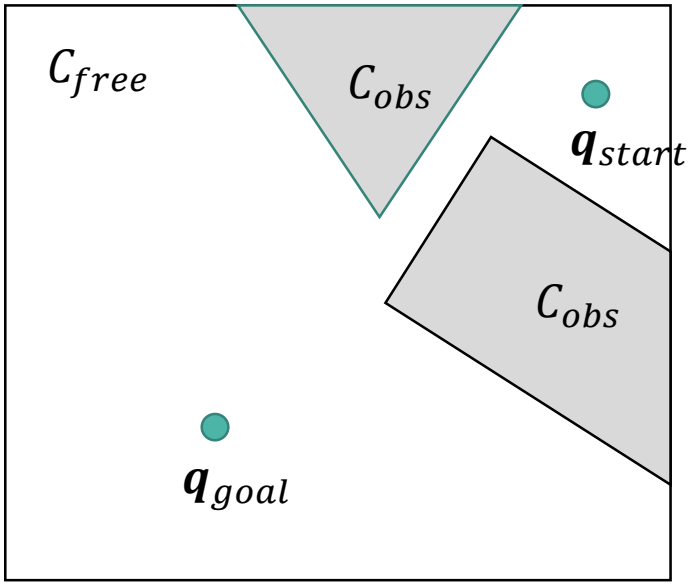


	Dynamic-Domain bi-RRT	bi-RRT
time (1)	0.4 sec	0.1 sec
no. nodes (1)	253	37
CD calls (1)	618	54
time (2)	2.5 sec	379 sec
no. nodes (2)	1607	6924
CD calls (2)	3751	781530
time (3)	1.6 sec	$>$ 80000 sec
no. nodes (3)	1301	–
CD calls (3)	3022	–

A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3856–3861, Apr. 2005.

Narrow Passages: Bridge Sampling (1)

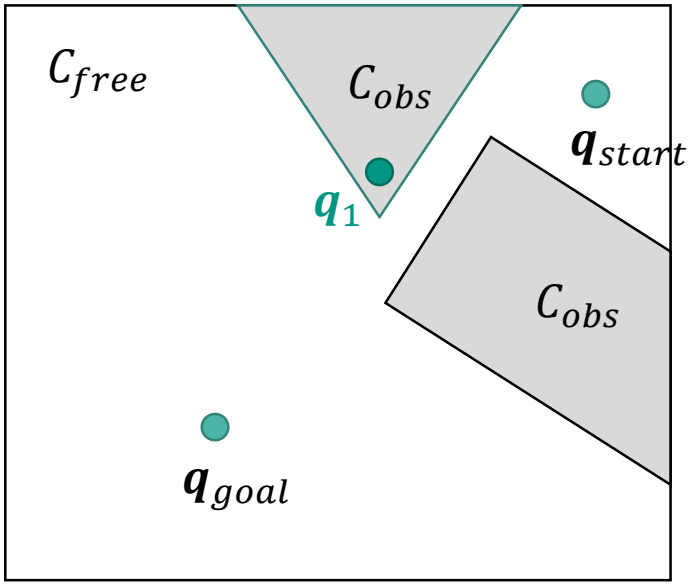
- **Idea**: Selectively choose points in narrow passages for the next sample



Narrow Passages: Bridge Sampling (2)

- **Idea**: Purposefully choose points in narrow passages for the next sample

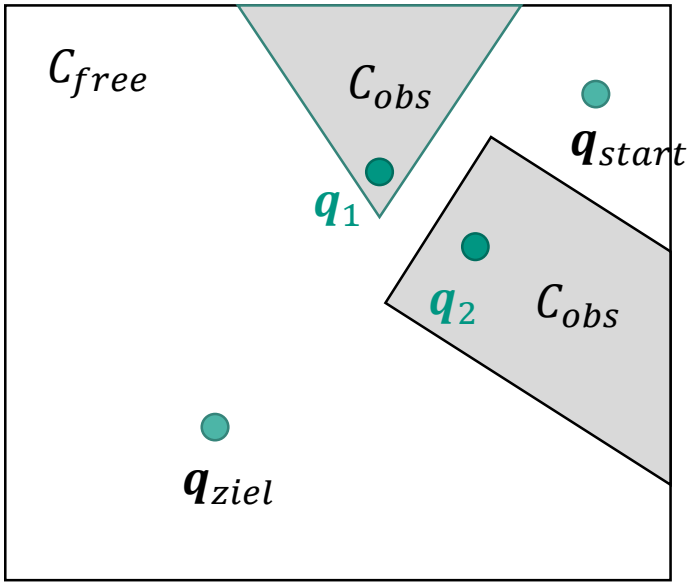
- **Approach**:
  1. Uniformly sample a random point $\boldsymbol{q}_1 \in C_{obs}$



Narrow Passages: Bridge Sampling (3)

- **Idea**: Selectively choose points in narrow passages for the next sample

- **Approach**:
  1. Uniformly sample a random point $\boldsymbol{q}_1 \in C_{obs}$
  2. Choose a second point $\boldsymbol{q}_2 \in C_{obs}$ in the vicinity of $\boldsymbol{q}_1$, following a suitable probability distribution

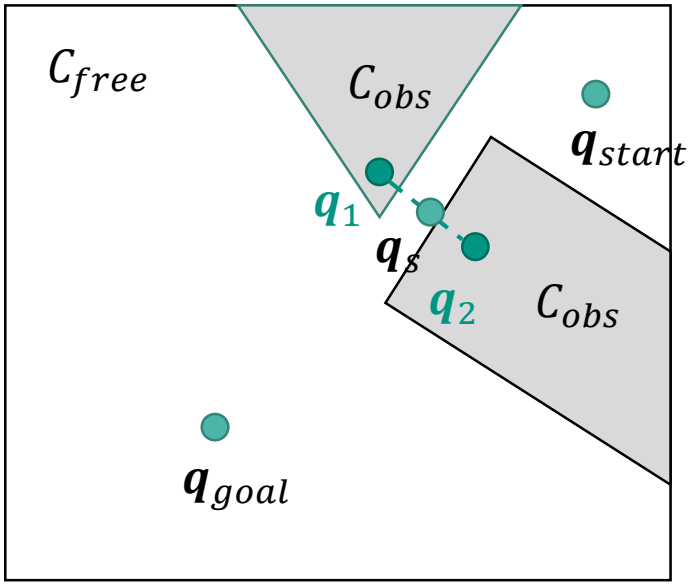


Narrow Passages: Bridge Sampling (4)

- **Idea**: Selectively choose points in narrow passages for the next sample

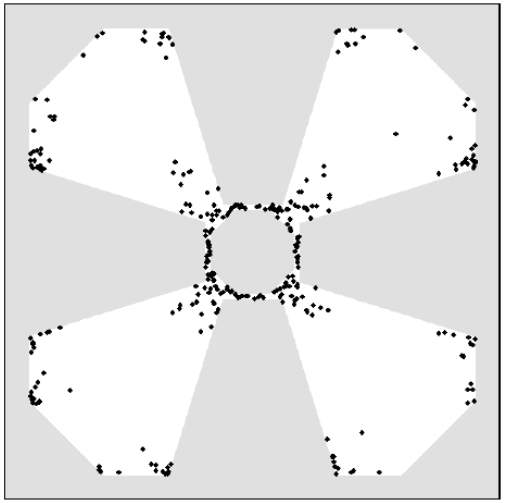
- **Approach**:
  1. Uniformly sample a random point $\boldsymbol{q}_1 \in C_{obs}$
  2. Choose a second point $\boldsymbol{q}_2 \in C_{obs}$ in the vicinity of $\boldsymbol{q}_1$, following a suitable probability distribution
  3. If the midpoint $\boldsymbol{q}_s$ between $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ is in C_{free}, use it as a new sample for the RRT (or the PRM)
  4. Repeat

在一个障碍物内取一点，然后在其一个半径内取另外一点，
确认他俩的重点是否在C_free内，若在，则为采样点



Narrow Passages: Bridge Sampling (5)

- **Bridge Sampling** increases the density of samples in interesting areas of the configuration space C
  - Interesting are areas in the vicinity of obstacles, especially in narrow passages
- Bridge Sampling can be used for RRTs and PRMs
- The core idea of the approach, the **Bridge Test**, can be calculated efficiently also in high-dimensional spaces



Samples generated by bridge sampling

Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. Reif, "*Narrow passage sampling for probabilistic roadmap planning*,"IEEE Transactions on Robotics, vol. 21, no. 6, pp. 1105–1115, 2005

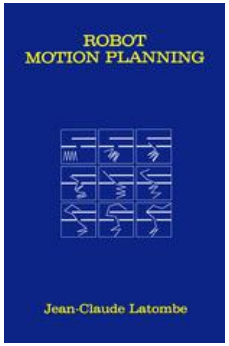
Software

- Simox
  - Simox is lightweight platform independent C++ toolbox containing three libraries for 3D simulation of robot systems, sampling based motion planning and grasp planning
  - https://gitlab.com/Simox/simox
  - Developed at H2T, KIT

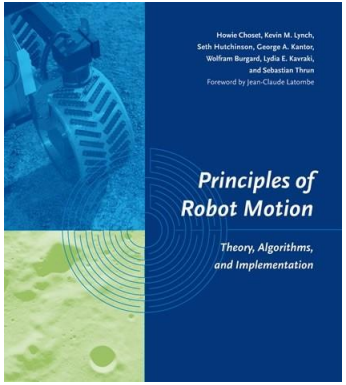
- The Open Motion Planning Library (OMPL)
  - OMPL consists of many state-of-the-art sampling-based motion planning algorithms
  - https://ompl.kavrakilab.org/
  - Kavraki Lab, Department of Computer Science, Rice University

Literature

- **Robot Motion Planning**
  Jean-Claude Latombe



- **Principles of Robot Motion: Theory, Algorithms, and Implementations**
  Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki and Sebastian Thrun



German Terms

English	German
Cell decomposition	Zellzerlegung
Constraint	Zwangsbedingung, oder Neben- und Randbedingung
Configuration	Konfiguration
Configuration space	Konfigurationsraum
Free space	Freiraum
Motion planning	Bewegungsplanung
Obstacle	Hindernis
Path planning	Pfadplanung
Potential field	Potentialfeld
Trajectory	Trajektorie
Visibility graph	Sichtgraph

German Terms

Englisch	Deutsch
Admissible	Zulässig
Best-first search	Bestensuche
Constraint	Nebenbedingung
Kernel	Kern
Narrow passages	Enge Passagen
Predecessor	Vorgängerknoten
Projection	Projektion
Sampling	Stichprobe

Engineering Humanoid Robots for a Better Life!

Best wishes for a bright holiday season and a peaceful new year

H2T High Performance Humanoid Technologies

Karlsruhe Institute of Technology
Institute for Anthropomatics and Robotics
KIT