Motion Planning

Challenges:

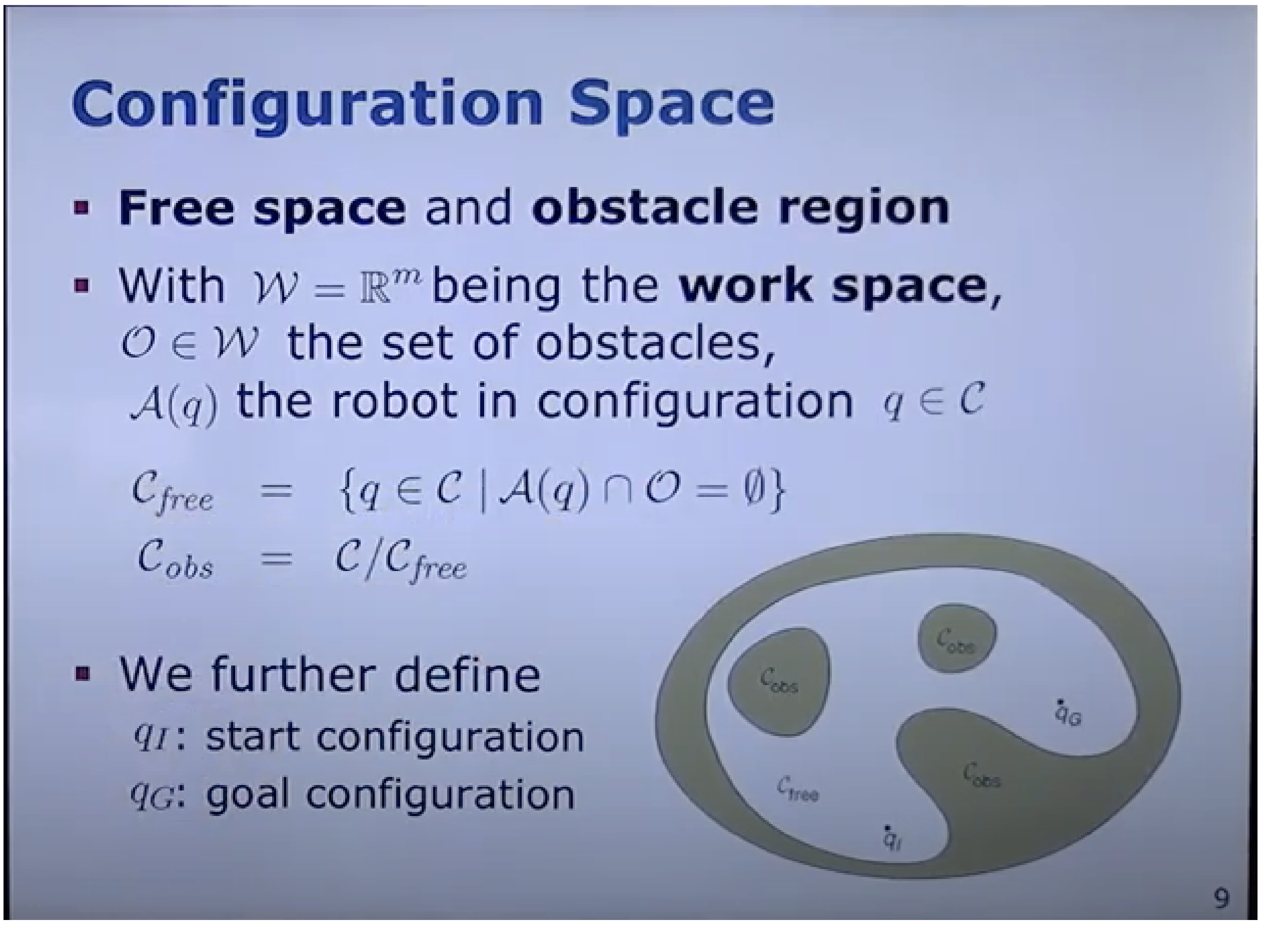
- Calculate the optimal path
- Take potential uncertainties (in the action) into account
- Quickly generate actions in the case of unforeseen objects
- Moving obstacles

Problem:

- Given a start pose, a desired goal poste, a geometric description of the robot, a geometric representation of the environment.
- Find a short path that moves the robot gradually from start to goal while never touching any obstacle. => Cost-optimal path.

Configuration Space:

Motion planning is defined in the configuration space, often called C-space: the positions of all robot points relative to a fixed coordinate system. Usually a configuration is expressed as a vector of positions and orientations. The configuration space (C-space) is the space of all possible configurations. The topology of this space is usually not that of a Cartesian space. The C-space is described as a topological manifold C-space is often discretized (e.g. grid-map)

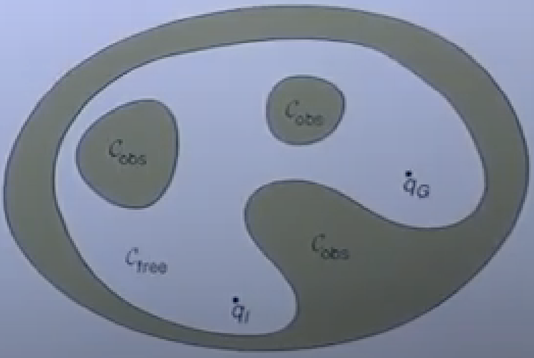


Configuration Space

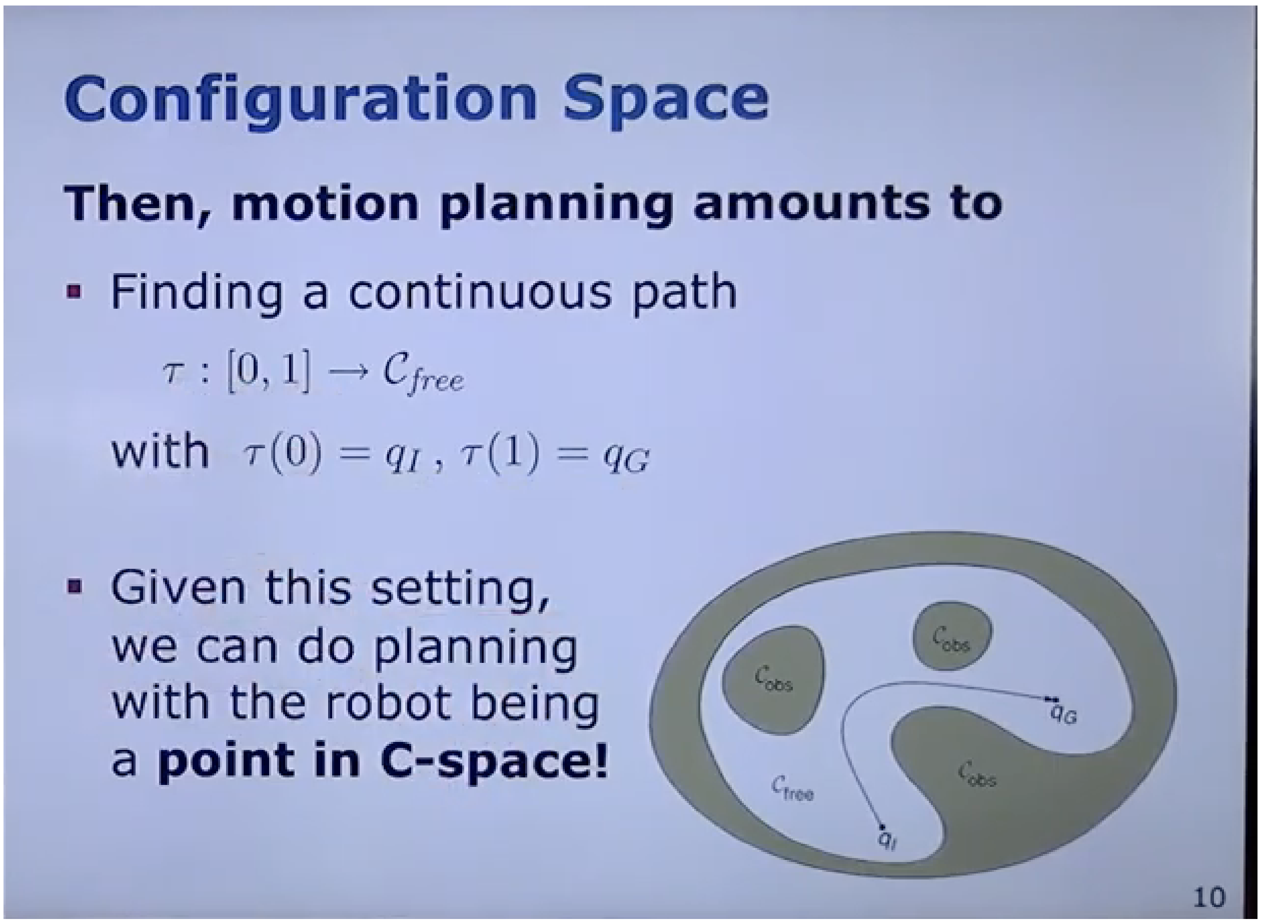
- **Free space** and **obstacle region**
- With $\mathcal{W} = \mathbb{R}^m$ being the **work space**,
  $\mathcal{O} \in \mathcal{W}$ the set of obstacles,
  $\mathcal{A}(q)$ the robot in configuration $q \in \mathcal{C}$

$$
\begin{aligned}
\mathcal{C}_{free} &= \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} = \emptyset\} \\
\mathcal{C}_{obs} &= \mathcal{C} / \mathcal{C}_{free}
\end{aligned}
$$

- We further define
  q_I: start configuration
  q_G: goal configuration



9



Configuration Space

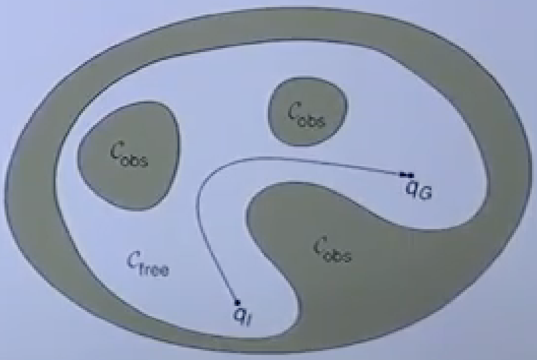
Then, motion planning amounts to

- Finding a continuous path

  $\tau : [0,1] \rightarrow \mathcal{C}_{free}$

  with $\tau(0) = q_I$, $\tau(1) = q_G$

- Given this setting, we can do planning with the robot being a **point in C-space!**

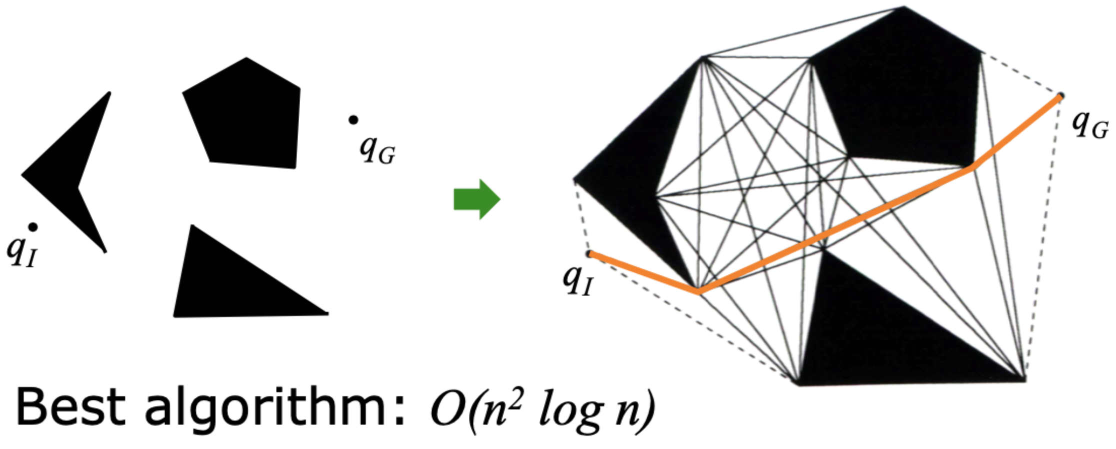


10

C-space discretization:

- combinatorial planning: captures the connectivity of free space into a graph and finds solution through search (warning: complexity grows exponentially)

- **Visibility graphs**: construct a path as a polygonal line connecting qi (start configuration) and qg (goal configuration) through vertices of Cobs

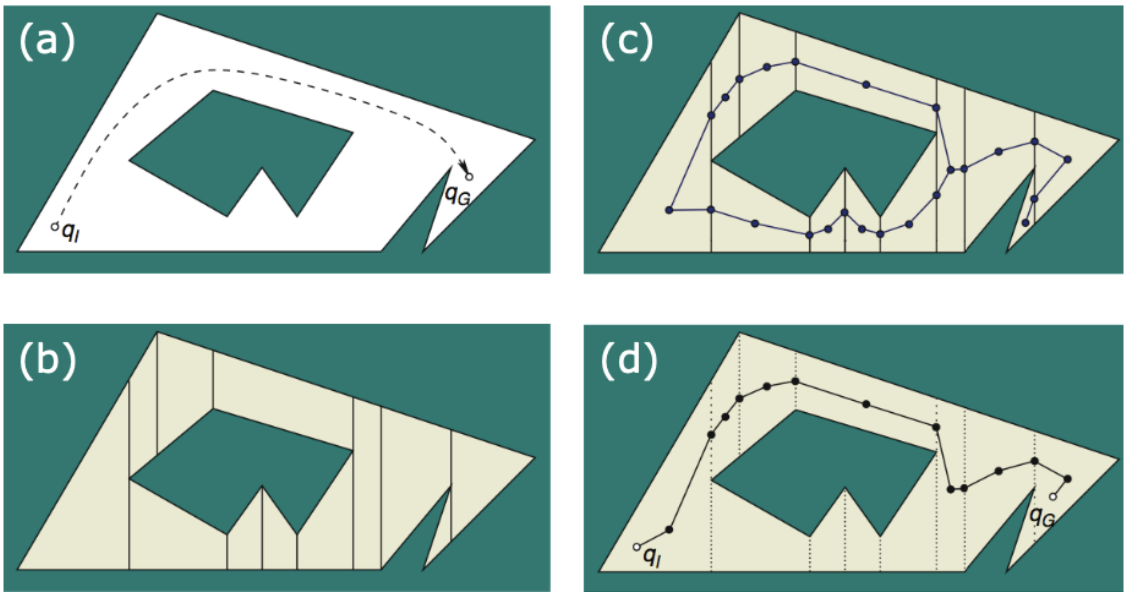


▪ Best algorithm: $O(n^2 \log n)$

- **Voronoi diagrams**: Defined to be the set of points q whose cardinality of the set of boundary points of Cobs with the same distance to q is greater than 1. Informally: the place with the same maximal clearance from all nearest obstacles.

- **Exact cell decomposition**: decompose Cfree into non-overlapping cells, construct connectivity graph to represent adjacencies, then search. Popular implementation:

   i. Decompose Cfree into trapezoids with vertical side segments by shooting rays upward and downward from each polygon vertex
   ii. Place one vertex in the interior of every trapezoid, pick e.g. the centroid
   iii. Place one vertex in every vertical segment
   iv. Connect the vertices



- **Approximate cell decomposition**: Approximate decomposition uses cells with the same simple predefined shape.

Combinatorial planning techniques are elegant and complete (they find a solution if it exists, report failure otherwise) But: become quickly intractable when C-space dimensionality increases (or n resp.) Combinatorial explosion in terms of facets to represent A, O abd Cobs especially when rotations bring in non- linearities and make C a nontrivial manifold.

All these techniques produce a road map which is a graph in Cfree in which each vertex (or intersection between 2 lines) is a configuration in Cfree and each edge is a collision-free path through Cfree.

- sampling-based planning: incrementally search the free-space for a solution using local collison-detection. Abandon the concept of explicitly characterizing Cfree and Cobs and leave the algorithm in the dark when exploring Cfree. The only light is provided by a collision-detection algorithm, that probes C to see whether some configuration lies in Cfree.
- Probabilistic road maps (PRM) [Kavraki et al., 92]

- Idea: Take random samples from C, declare them as vertices if in Cfree, try to connect nearby vertices with local planner
- The local planner checks if line-of-sight is collision-free (powerful or simple methods)
- Options for nearby: k-nearest neighbors or all neighbors within specified radius
- Configurations and connections are added to graph until roadmap is dense enough

- Rapidly exploring random trees (RRT) [Lavalle and Kuffner, 99]

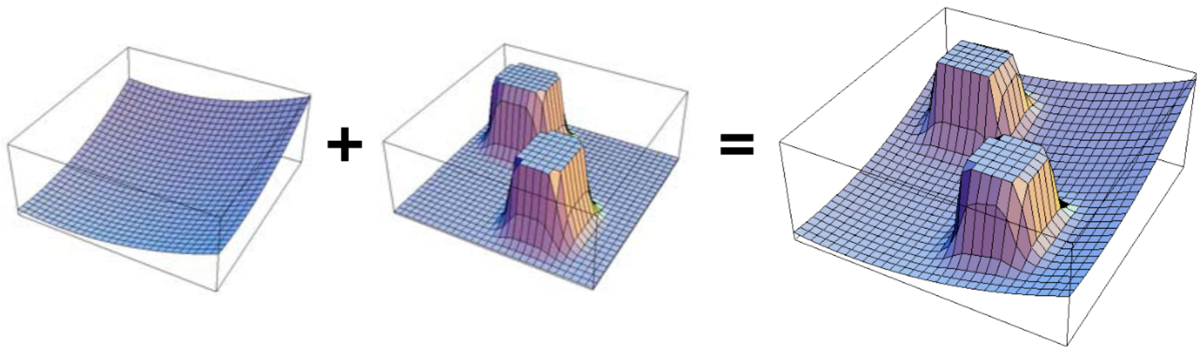
- **Idea**: aggressively probe and explore the C-space by expanding incrementally from an initial configuration q0
- The explored territory is marked by a tree rooted at q0 Bidirectional search: Grow two RRTs, one from qI, one from qG. In every other step, try to extend each tree towards the newest vertex of the other tree

All techniques discussed so far aim at cap- turing the connectivity of Cfree into a graph **Potential Field methods follow a different idea:** The robot, represented as a point in C, is modeled as a particle under the influence of a artificial potential field U

- Potential function

$$\mathbf{U}(q) = \mathbf{U}_{att}(q) + \mathbf{U}_{rep}(q)$$

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



- Simply perform **gradient descent**
- C-pace typically discretized in a grid

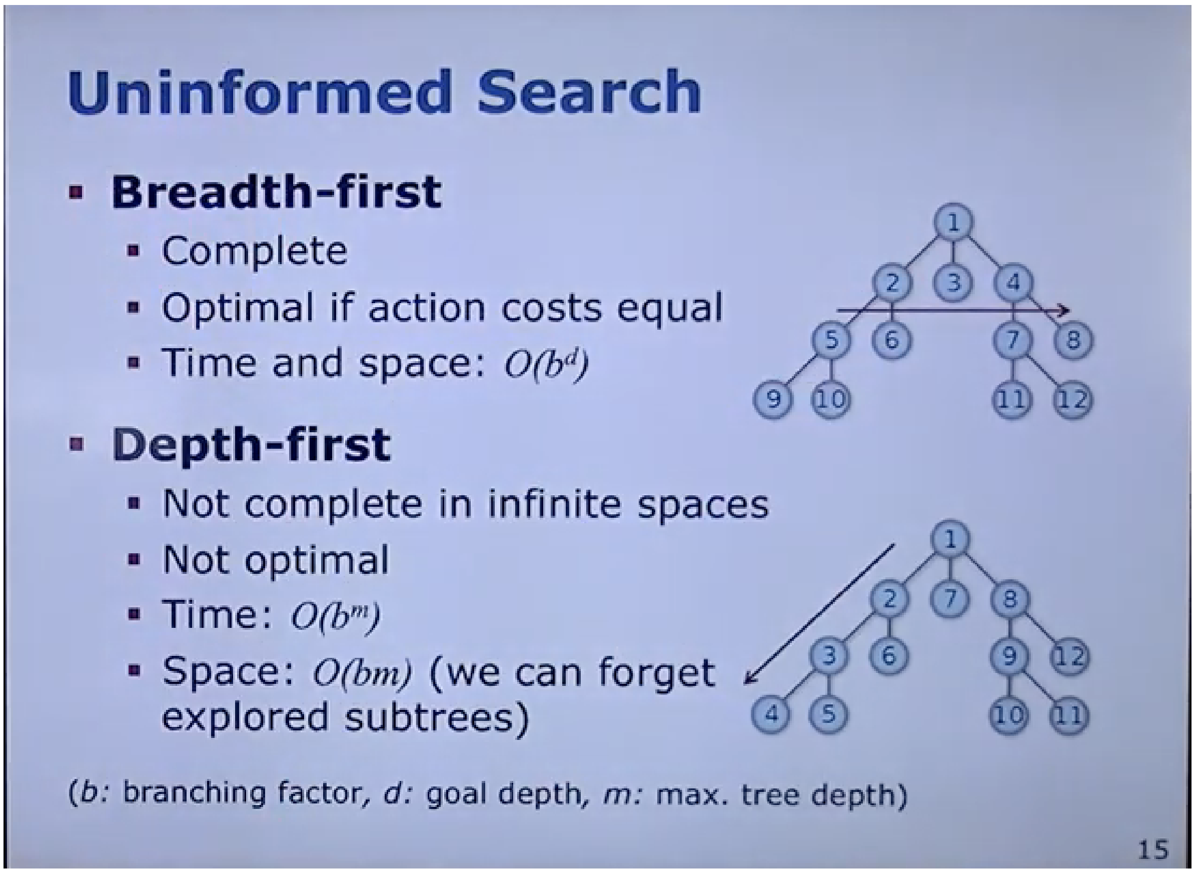
U superimposes:

- Repulsive forces from obstacles
- Attractive force from goal

Main problems: robot gets stuck in local minima Way out: Construct local-minima-free navigation function ("NF1"), then do gradient descent (e.g. bushfire from goal) The gradient of the potential function defines a vector field (similar to a policy) that can be used as feedback control strategy, relevant for an uncertain robot However, potential fields need to represent Cfree explicitely. This can be too costly.

The search can happen in different ways:

- blank or uninformed search: breadth-first, uniform cost, depth-first, bidirectional... search. Expanding different nodes. Often not used in robotics



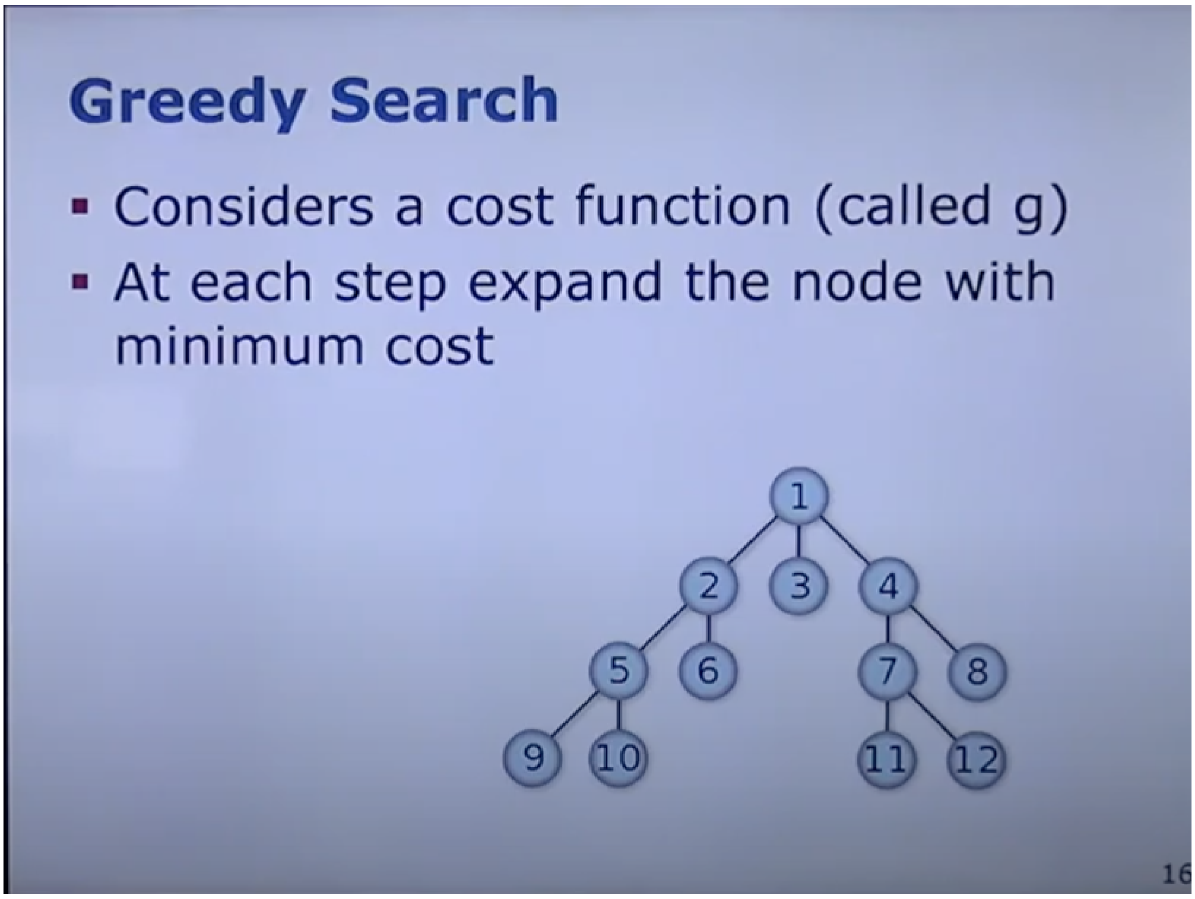
Uninformed Search

- **Breadth-first**
  - Complete
  - Optimal if action costs equal
  - Time and space: $O(b^d)$
- **Depth-first**
  - Not complete in infinite spaces
  - Not optimal
  - Time: $O(b^m)$
  - Space: $O(bm)$ (we can forget explored subtrees)

(b: branching factor, d: goal depth, m: max. tree depth)

15

- Informed search: further information about the domain through heuristics. e.g. greedy best-first search, A*, D* and many variants.



Greedy Search

- Considers a cost function (called g)
- At each step expand the node with minimum cost

16

If uniform cost for every edge, greedy search = breadth-first search A_start The heuristics should always under-estimate the real costs (given h* = perfect heuristics, h <= h*) Heuristics are problem-specific. Good ones (admissible, efficient) for our task are:

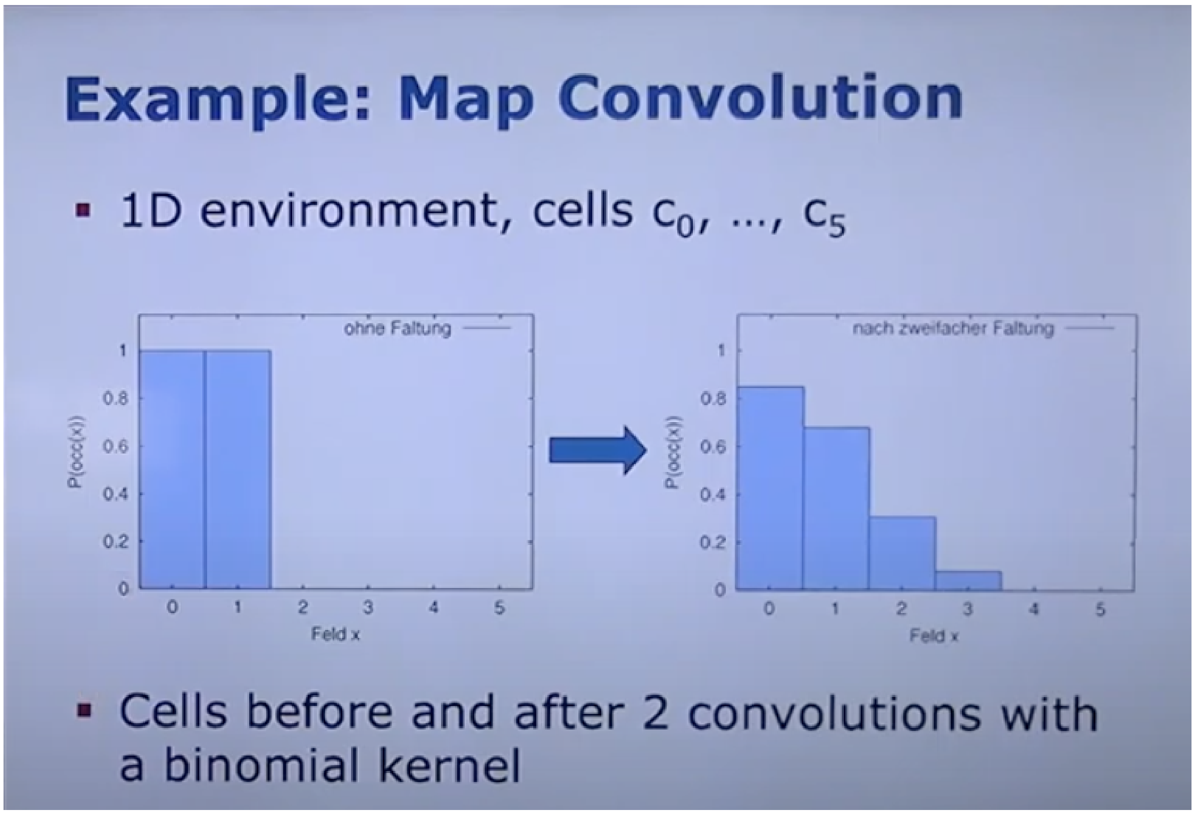
- Straight-line distance hSLD(n) (as with any routing problem)
- Octile distance: Manhattan distance extended to allow diagonal moves
- Deterministic Value Iteration/Dijkstra hVI(n)

A_star_flow_chart

In A*-based path planning, the robot usually knows its location, computes its path based on a correct map and correct motion commands are executed.

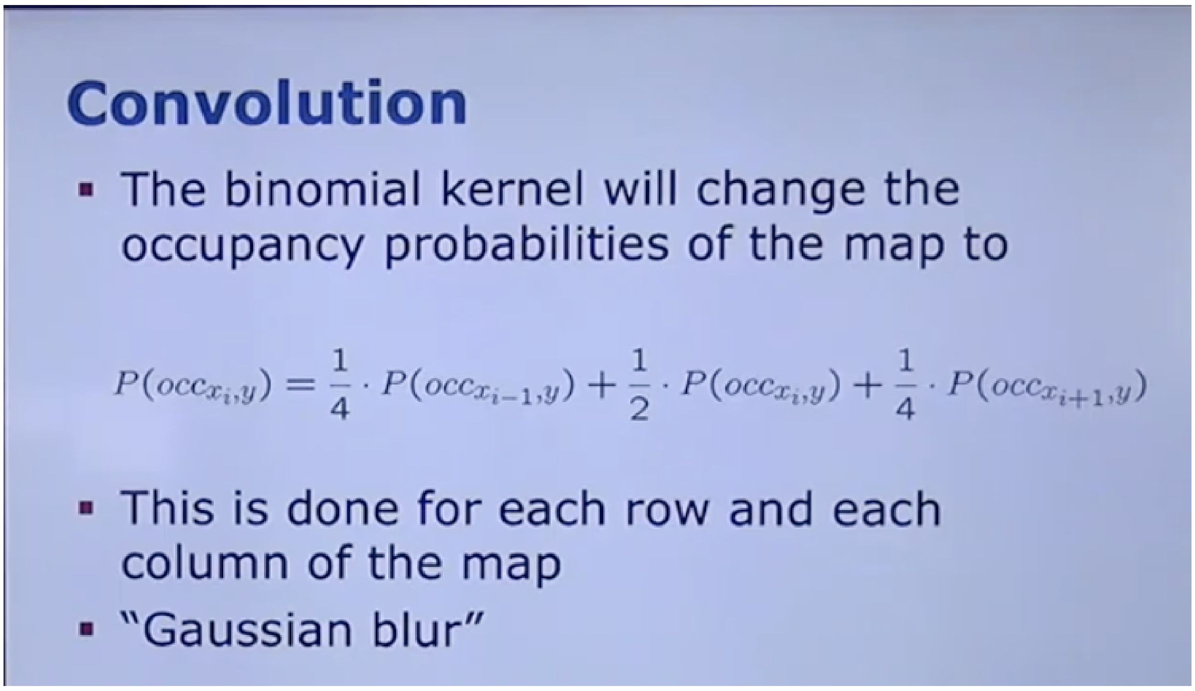
Tricks:

- Convolution of the grid map (increases occupied space - convolution blurs the map M with kernel k e.g. a Gaussian kernel)



Example: Map Convolution

- 1D environment, cells $c_0, \ldots, c_5$
- Cells before and after 2 convolutions with a binomial kernel



Convolution

- The binomial kernel will change the occupancy probabilities of the map to

$$P(occ_{x_i,y}) = \frac{1}{4} \cdot P(occ_{x_{i-1},y}) + \frac{1}{2} \cdot P(occ_{x_i,y}) + \frac{1}{4} \cdot P(occ_{x_{i+1},y})$$

- This is done for each row and each column of the map
- "Gaussian blur"

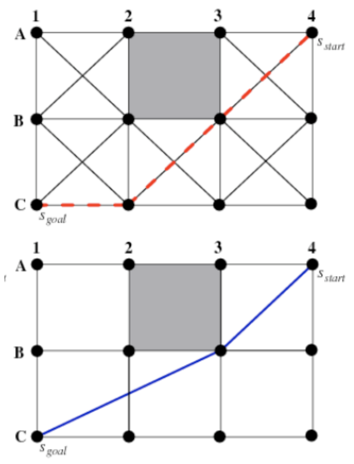
- make obstacles bigger

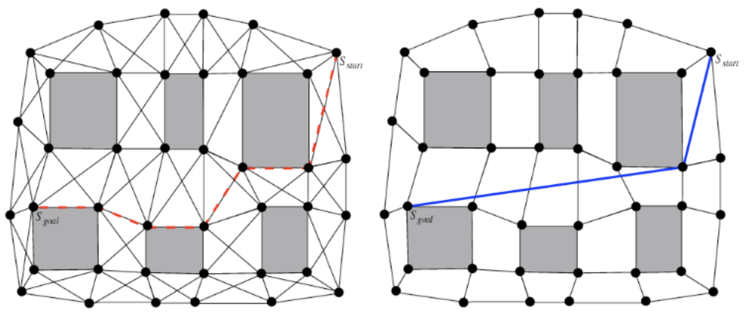
A* in convolved maps:

- costs are a product of path length and occupancy probability of the cells
- cells with higher occupancy probability (e.g. caused by convolution) are avoided by the robot so obstacles are avoided.
- this technique is fast and reliable for moderate noise motion

Any-Angle A*

- **Problem:** A* search only considers paths that are **constrained to graph edges**
- This can lead to **unnatural**, grid-aligned, and **suboptimal** paths



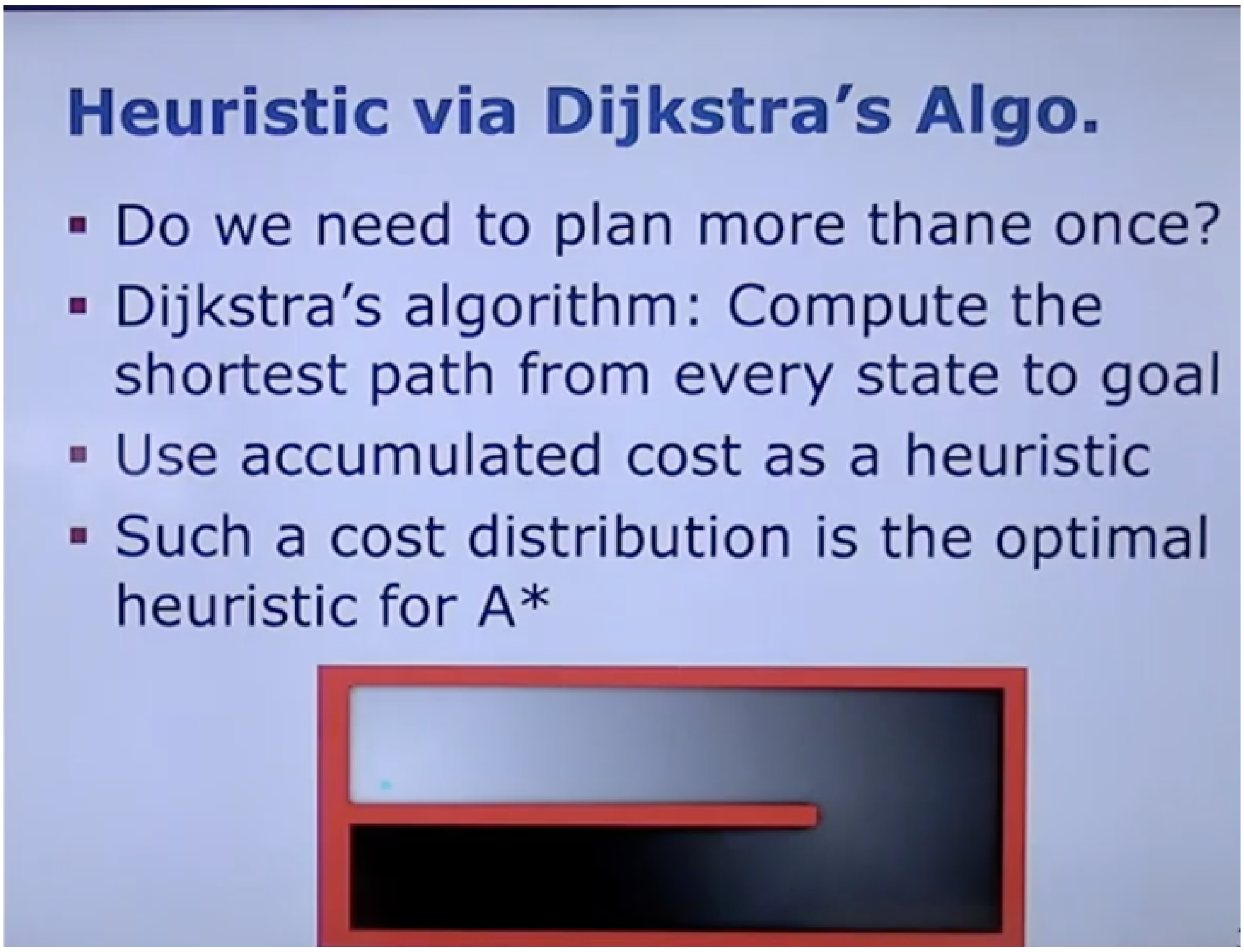


Pictures from *[Nash et al. AAAI'07]*

Any-Angle A*

- Different approaches:
  - **A* on Visibility Graphs**
    Optimal solutions in terms of path length!
  - **A* with post-smoothing**
    Traverse solution and find pairs of nodes with direct line of sight, replace by line segment
  - **Field D*** *[Ferguson and Stentz, JFR'06]*
    Interpolates costs of points not in cell centers. Builds upon D* family, able to efficiently replan
  - **Theta*** *[Nash et al. AAAI'07, AAAI'10]*
    Extension of A*, nodes can have non-neighboring successors based on a line-of-sight test

Heuristic via Dijkstra's algorithm In case we need to re-plan many times



Heuristic via Dijkstra's Algo.

- Do we need to plan more thane once?
- Dijkstra's algorithm: Compute the shortest path from every state to goal
- Use accumulated cost as a heuristic
- Such a cost distribution is the optimal heuristic for A*

Problem: In unknown, partially known or dynamic environments, the planned path may be blocked and we need to replan. Can this be done efficiently, avoiding to replan the entire path? **Idea:** Incrementally repair path keeping its modifications local around robot pose. Several approaches implement this idea: D* (Dynamic A*) [Stentz, ICRA'94, IJCAI'95], D* Lite [Koenig and Likhachev, AAAI'02], Field D* [Ferguson and Stentz, JFR'06]

Measurement of a search algorithm:

- Completeness
- Optimality
- Time complexity
- Space complexity

Expand to higher-dimensional space (5D)

5-D state space:

- 3-D pose (position[x,y] and orientation [yaw angle])
- 2-D velocity (translational, rotational)

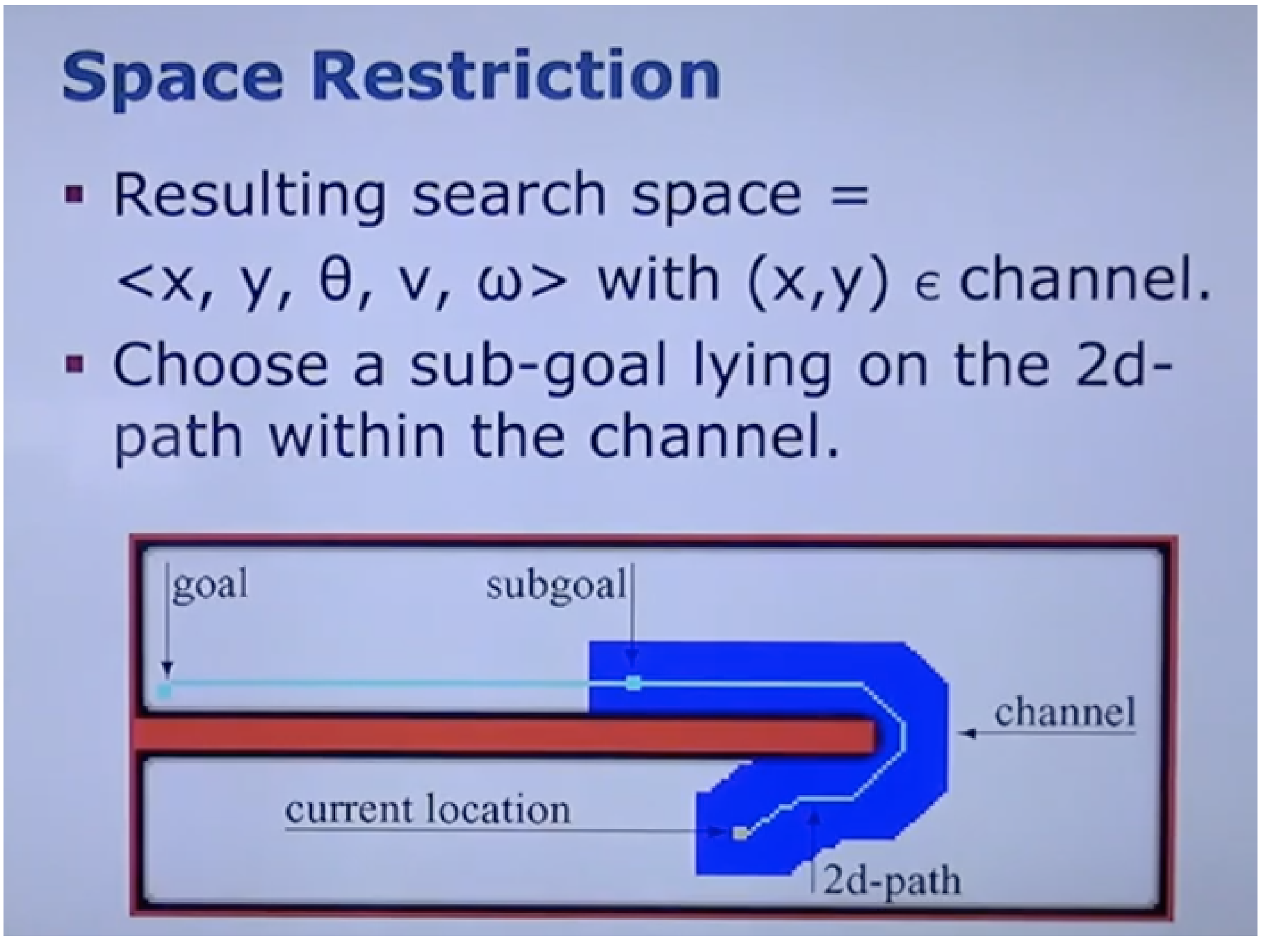
It enables to take into account kinematic constraints (e.g. acceleration).

The new pose of the robot is a result of the motion equations where we can introduce constraints (e.g. velocity changes).

The issue of searching in a 5D-space is its computational costs so we need to restrict the search space. Idea: start with 2D-space (x,y) to find the shortest path and consider that the fastest path is close to the shortest path.

5 steps:

1. **Update the grid map** based on sensor data (e.g. person walking). Here the map is represented as a 2d-occupancy grid map, convolution of the map increases security distance, detected obstacles are added and cells discovered free are cleared)
2. _Use standard 2D A to find a a trajectory_* in <x,y> space using the updated map
3. **Setup a restricted 5D configuration space** based on step 2 (like a funnel of possible search space). We choose a sub-goal lying on the 2nd path within the channel
4. Find a path in the 5D space to reach the sub-goal by using A* in the restricted 5D space. To estimate cells costs, perform a deterministic 2D value iteration within the channel



Space Restriction

- Resulting search space = $\langle x, y, \theta, v, \omega \rangle$ with $(x,y) \in$ channel.
- Choose a sub-goal lying on the 2d-path within the channel.

Dynamic Environments

- Dynamic window approaches
- Grid-map-based planning
- Nearness-Diagram-navigation
- Vector-Field-Histogram+
- A*, D*, D* Lite, Ara*, MDP (Markov Decision Process)...

Layered Architecture:

It is common to subdivide the problem of motion planning into a global and local planning task:

- An approximate global planner computes paths ignoring the kinematic and dynamic vehicle constraints. Planning (low frequency) =>sub-goal (e.g. position 5m ahead).

- An accurate local planner accounts for the constraints and generates (sets of) feasible local trajectories ("collision avoidance"). Collision Avoidance (high frequency) => reacts to obstacles

Controller (e.g. steering angles, velocity...)

https://www.youtube.com/watch?v=eS3QWc8uNNM&t=1171s