Motion and Path Planning
– Graph-based methods –
CSC398 Autonomous Robots

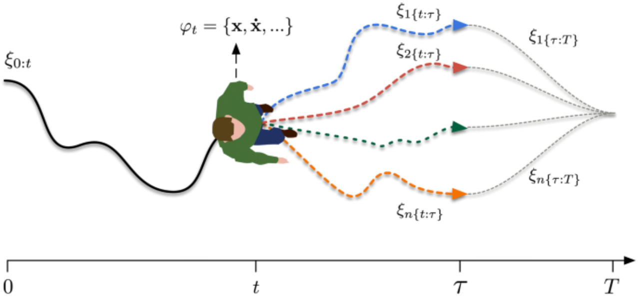
Ubbo Visser

Department of Computer Science
University of Miami

October 29, 2024

UNIVERSITY
OF MIAMI

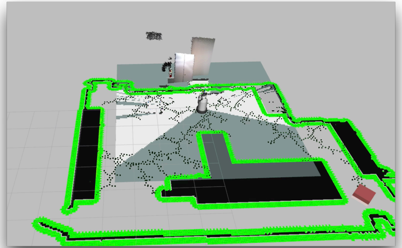
Outline



Source: Pena & Visser (2020): ITP: Inverse Trajectory Planning for Human Pose Prediction Künst Intell 34, 209–225.

Motion Planning in Robotics

Definition and Aim

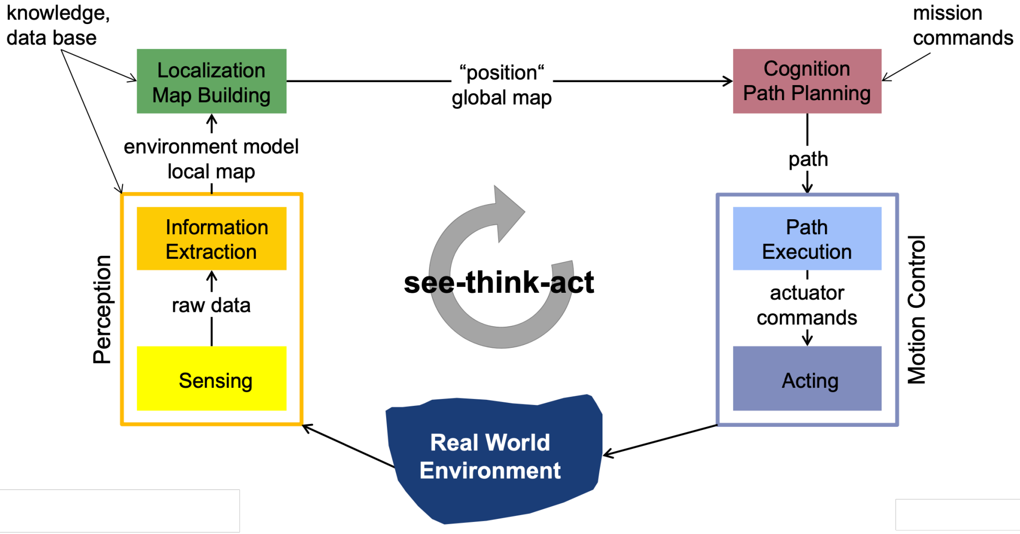


- **Definition:** Calculating a sequence of feasible movements for a robot to achieve a specific goal without collisions or constraint violations.
- **Aim:** Enable autonomous robots to navigate and interact in dynamic, complex environments safely and efficiently.

Suggested Readings:

- *Principles of Robot Motion: Theory, Algorithms, and Implementations*, Howie Choset et al. (2005), MIT Press.
- *Planning Algorithms*, Steven M. LaValle (2006), Cambridge University Press.
- *Robot Motion Planning and Control*, edited by Jean-Paul Laumond (1998), Springer LNCIS.

Perception - Cognition - Action cycle

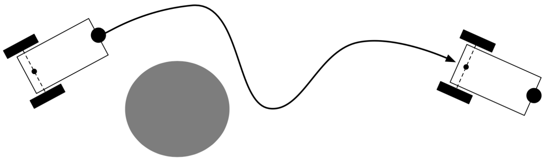


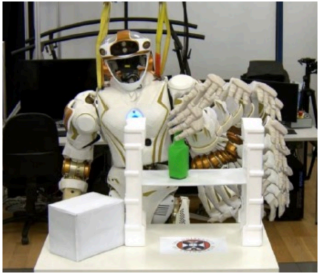
Source: Siegwart et. al (2018): Autonomous Mobile Robots, Lecture ETH Zürich

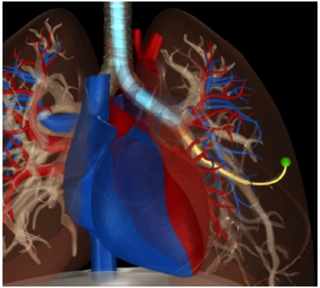
Examples of motion planning

More examples

- Steering autonomous vehicles.
- Controlling humanoid robot
- Surgery planning
- Protein folding
- ...





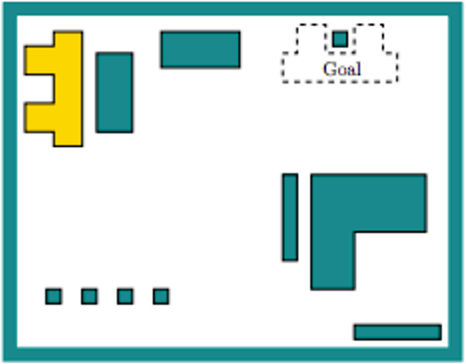


Some history

- Formally defined in the 1970s
- Development of exact, combinatorial solutions in the 1980s
- Development of sampling-based methods in the 1990s
- Development of sampling-based methods in the 1990s
- Current research: inclusion of differential and logical constraints, planning under uncertainty, parallel implementation, feedback plans and more

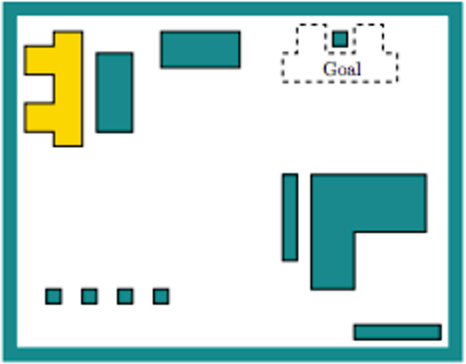
Simple setup

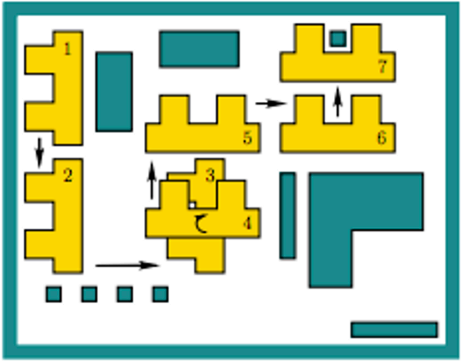
- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- The robot is a rigid polygon
- Problem: given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region



Simple setup

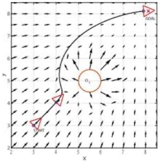
- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- The robot is a rigid polygon
- Problem: given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region

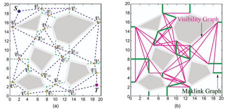


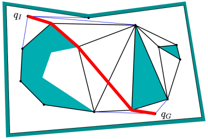


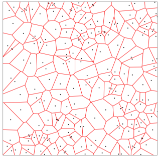
Popular approaches

- *Potential fields:* create forces on the robot that pull it toward the goal and push it away from obstacles [Rimon, Koditschek, '92].
- *Grid-based planning*: discretizes problem into grid and runs a graph-search algorithm (Dijkstra, A*, ...) [Stentz, '94]
- *Combinatorial planning:* constructs structures in the configuration (C-) space that completely capture all information needed for planning [LaValle, '06]
- *Sampling-based planning:* uses collision detection algorithms to probe and incrementally search the C-space for a solution, rather than completely characterizing all of the C_{free} structure [Kavraki et al, '96; LaValle, Kuffner, '06, etc.]



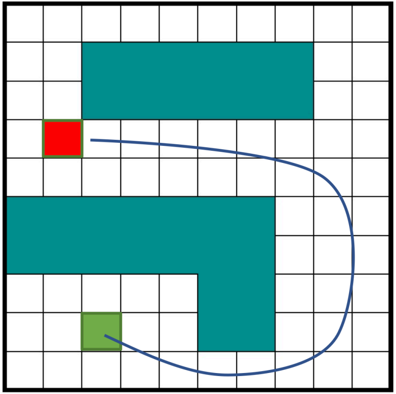






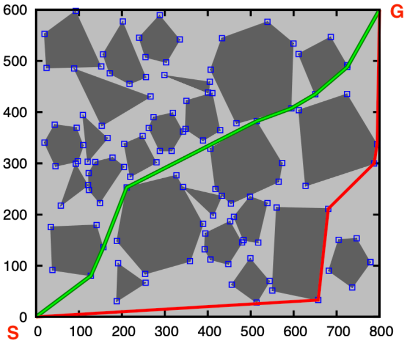
Grid-based approaches

- Discretize the continuous world into a grid
  - Each grid cell is either free or forbidden
  - Robot moves between adjacent free cells
  - **Goal:** find sequence of free cells from start to goal
- Mathematically, this corresponds to pathfinding in a discrete graph $\mathcal{G} = \mathcal{V}, \mathcal{E}$
  - Each vertex $v \in \mathcal{V}$ represents a free cell
  - Edges $v, u \in \mathcal{E}$ connect adjacent grid cells

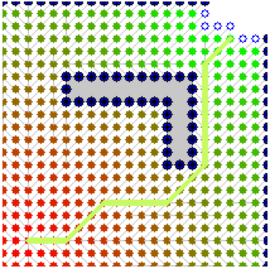


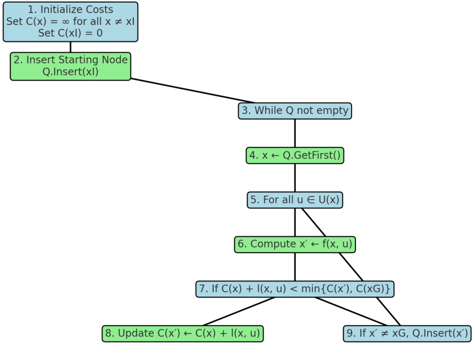
Grid-based approaches - Graph search

- Having determined decomposition, how to find optimal path?
- **Label-Correcting Algorithms:** $C(q)$: cost of path from S to G
- Idea: progressively discover shorter paths from the origin to every other node i
- Produce optimal plans by making small modifications to the general forward-search algorithm
- Here: Uniform cost search, Dijkstra



Grid-based approaches - Graph search (2)



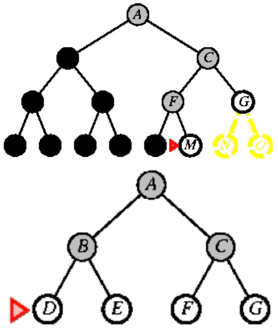


Animation:https://upload.wikimedia.org/wikipedia/commons/2/23/Dijkstras_progress_animation.gif

Grid-based approaches - Graph search (3)

GetNext()?

- Which node is returned by GetNext()?
- Depth-First-Search (DFS): Maintain $\mathcal{Q}$ as a **stack** – LIFO: Last in/first out. Comment: Lower memory requirement (only need to store part of graph) but incomplete if an infinite path
- Breadth-First-Search (BFS): Maintain $\mathcal{Q}$ as a **list** – FIFO: First in/first first out. Comment: Update cost for all edges up to the current depth before proceeding to a greater depth. Can deal with negative edge (transition) costs.
- Best-First (BF, Dijkstra, A*): (Greedily) select next q: $q = argmin_{q \in Q} C(q)$. Comment: Repeated states. Cost monoton increasing, non-negative. Heuristics! A* complete and optimal.

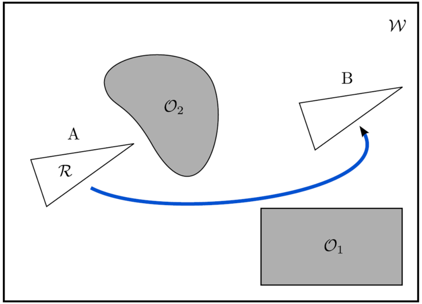


Grid-based approaches - Summary

- Pros:
  - Simple, easy to use
  - Fast (depending on grid size)
- Cons:
  - Dependent on resolution, i.e., if grid size too small no solution might be reached
  - Limited to simple robots: grid size is exponential in number of DOFs

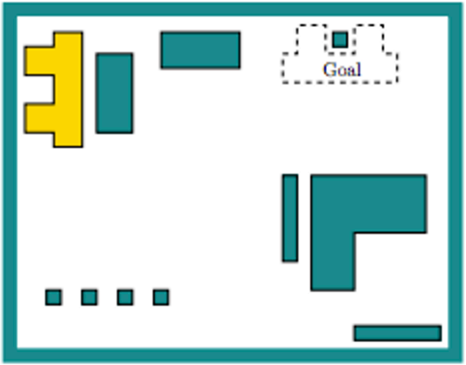
Continuous motion planning

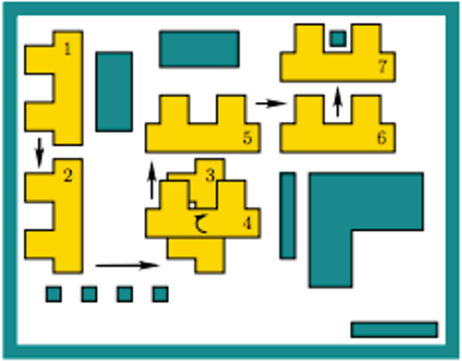
- A robot is a geometric entity operating in continuous space
- *Combinatorial techniques* for motion planning capture the structure of this continuous space; Particularly, the regions in which the robot is not in collision with obstacles.
- Such approaches are typically complete, i.e., guaranteed to find a solution; and sometimes even an optimal one



Simple setup - revisit

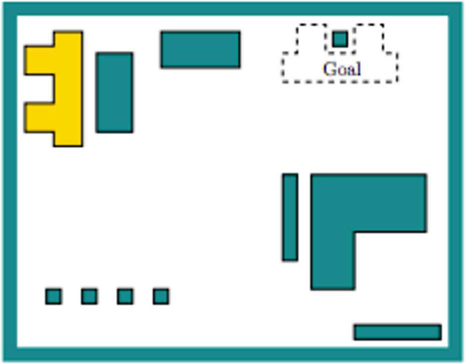
- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- The robot is a rigid polygon
- Problem: given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region

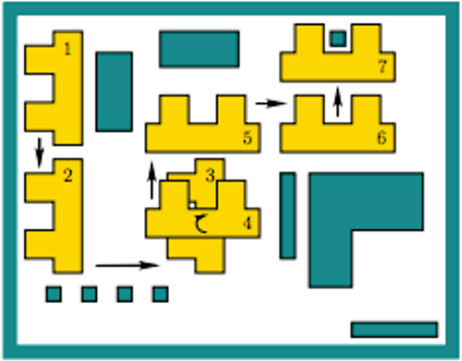




Simple setup - revisit

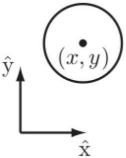
- Most important: motion planning problem described in the real world, but it really lives in another space – the configuration (C-) space!



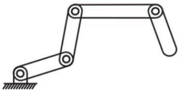


Configuration space

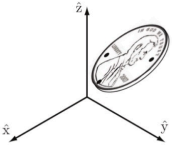
- C- space: captures all degrees of freedom (all rigid body transformations)
- In more detail, let $\mathcal{R} \in \mathbb{R}^2$ be a polygonal robot (e.g., a triangle)
- The robot can rotate by angle θ or translate $(x_t, y_t) \subset \mathbb{R}^2$
- Every combination $q = (x_t, y_t, \theta)$ yields a unique robot placement: configuration
- Meaning: the C-space is a subset of $\mathbb{R}^3$
- Note: $\theta \pm 2\pi$ yields equivalent rotations $\Rightarrow$ C-space is: $\mathbb{R}^2 \times \mathcal{S}^1$



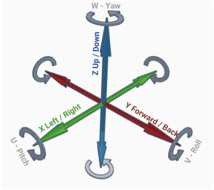
$d = 2$



$d = 4$

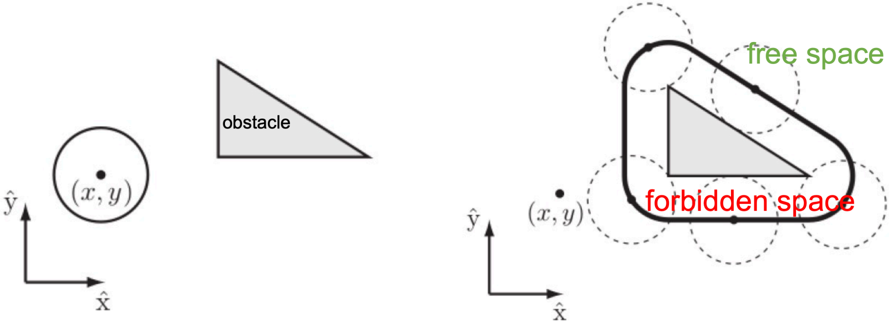


$d = 6$

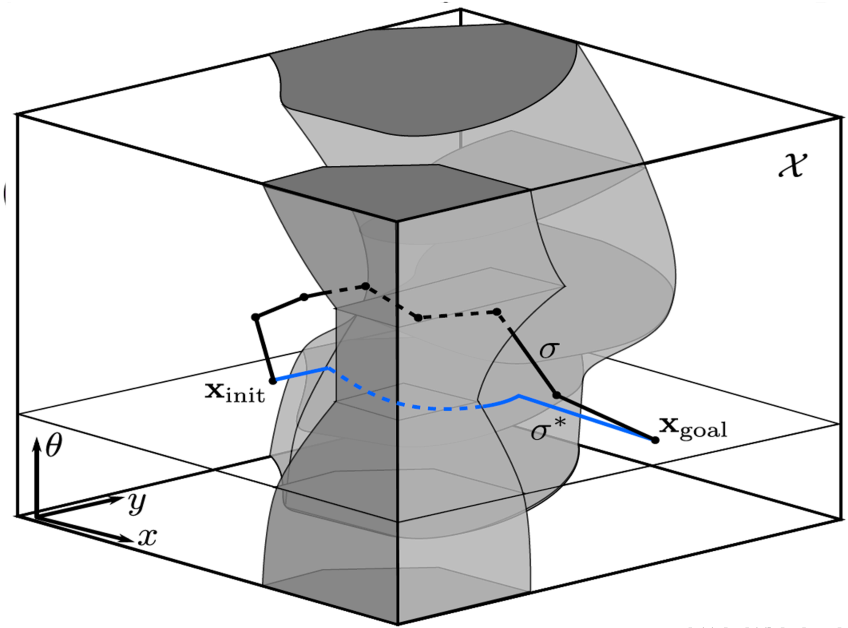


Configuration free space

- The subset $\mathcal{F} \subseteq \mathcal{C}$ of all collision free configurations is the **free space**

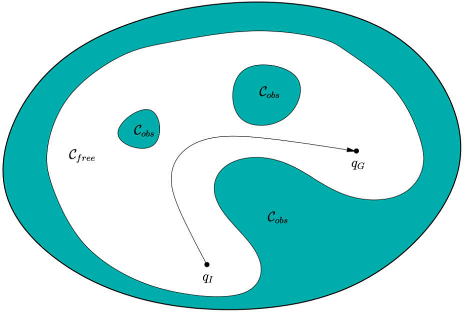


Configuration free space



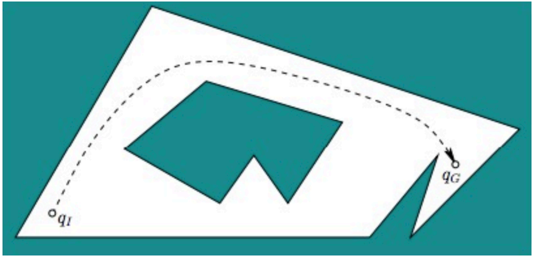
Planning in C-space

- Let $R(q) \subset W$ denote set of points in the world occupied by robot when in configuration space q
- Robot in collision $\Leftrightarrow R(q) \cap 0 \neq \emptyset$
- Accordingly, free space is defined as: $C_{free} = \{q \in C | R(q) \cap 0 \neq \emptyset\}$
- Path planning problem in C-space: compute a **continuous** path: $\tau : [0, 1] \to C_{free}$, with $\tau(0) = q_1$ and $\tau(1) = q_G$



Combinatorial planning

- Key idea: compute a roadmap, which is a graph in which each vertex is a configuration in C_{free} and each edge is a path through C_{free} that connects a pair of vertices



Free-space roadmaps

Given a complete representation of the free space, we compute a roadmap that captures its connectivity

A roadmap should preserve:

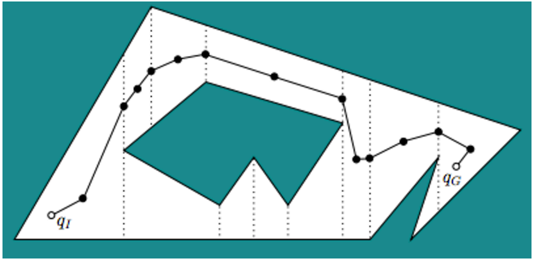
- Accessibility: it is always possible to connect some q to the roadmap (e.g., $q_1 \rightarrow s_1$, $q_G \rightarrow s_2$)
- Connectivity: if there exists a path from q_1 to q_G, there exists a path on the roadmap from s_1 to s_2

Main point: a roadmap provides a discrete representation of the continuous motion planning problem without losing any of the original connectivity information needed to solve it

Cell decomposition

Typical approach: cell decomposition. General requirements:

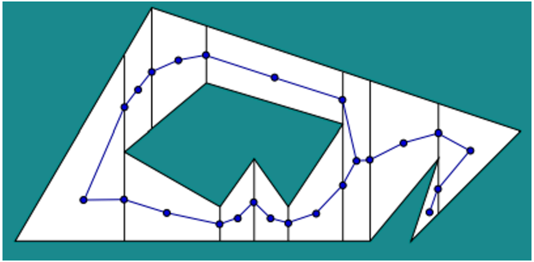
- Decomposition should be easy to compute
- Each cell should be easy to traverse (ideally convex)
- Adjacencies between cells should be straightforward to determine



Computing a trapezoidal cell decomposition

For every vertex (corner) of the forbidden space:

- Extend a vertical ray until it hits the first edge from top and bottom
  - Compute intersection points with all edges, and take the closest ones
  - More efficient approaches exists

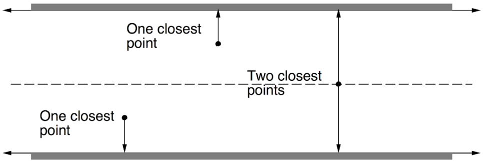


Other roadmaps

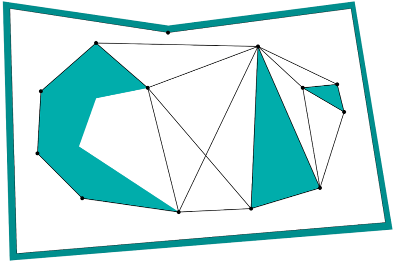
For every vertex (corner) of the forbidden space:

- Extend a vertical ray until it hits the first edge from top and bottom
  - Compute intersection points with all edges, and take the closest ones
  - More efficient approaches exists

Maximum clearance (medial axis)



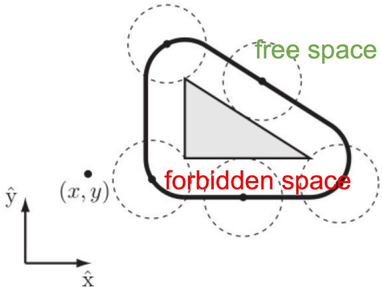
Minimum distance (visibility graph)



Note: No loss in optimality for a proper choice of discretization

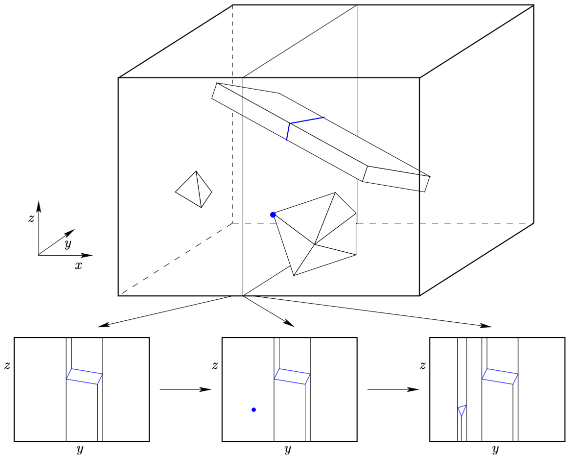
Caveat: free-space computation

- The free space is not known in advance
- We need to compute this space given the ingredients
  - Robot representation, i.e., its shape (polygon, polyhedron, ...)
  - Representation of obstacles
- To achieve this we do the following:
  - Contract the robot into a point
  - In return, inflate (or stretch) obstacles by the shape of the robots



Higher dimensions

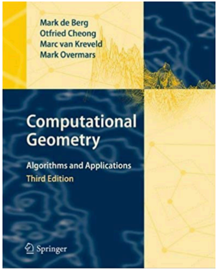
Extensions to higher dimensions is challenging ⇒ algebraic decomposition methods



Additional resources on combinatorial planning

For every vertex (corner) of the forbidden space:

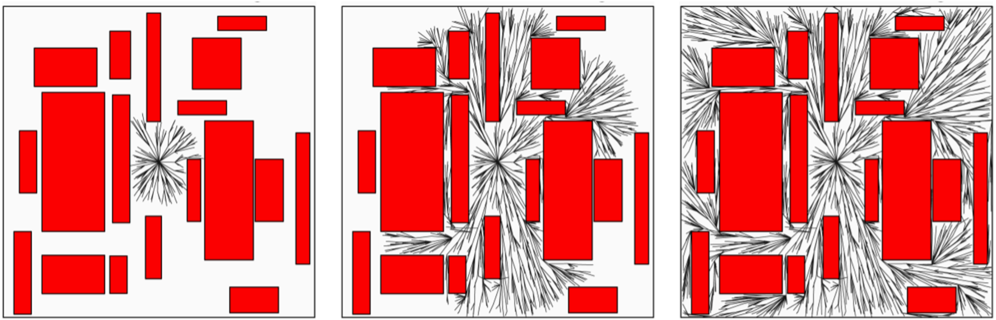
- Visualization of C-space for polygonal robot: `https://www.youtube.com/watch?v=SBFwgR4K1Gk`
- Algorithmic details for Minkowski sums and trapezoidal decomposition: de Berg et al., "Computational geometry: algorithms and application", 2008
- Implementation in C++: Computational Geometry Algorithms Library



Combinatorial planning: summary

- These approaches are complete and even optimal in some cases, do not discretize or approximate the problem
- Have theoretical guarantees on the running time (complexity is known)
- Usually limited to small number of DOFs
- Problem specific: each algorithm applies to a specific type of robot/problem (intractable for many problems)
- Difficult to implement: require special software to reason about geometric data structures (CGAL)

Next: sampling-based planning



Acknowledgements

Acknowledgement

This slide deck is based on material from the Stanford ASL and ETH Zürich