

Principles of Robot Autonomy I

Motion planning I: graph search algorithms

Attendance Form



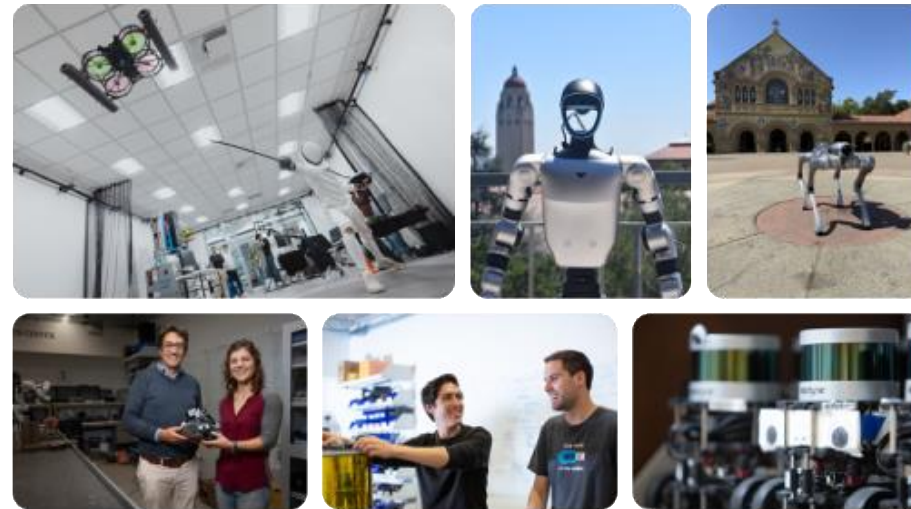
ASL Open House

Where foundation models meet the physical world.
Come see what we're building.

Hosted by Prof. Marco Pavone and the Autonomous Systems Lab

RESEARCH FOCUSES

Robot Foundation Models | Physical AI Safety | Embodied AI Agents | Data Flywheels



WHEN

Thursday, October 15
4:00 pm

WHERE

Durand 023

ALSO

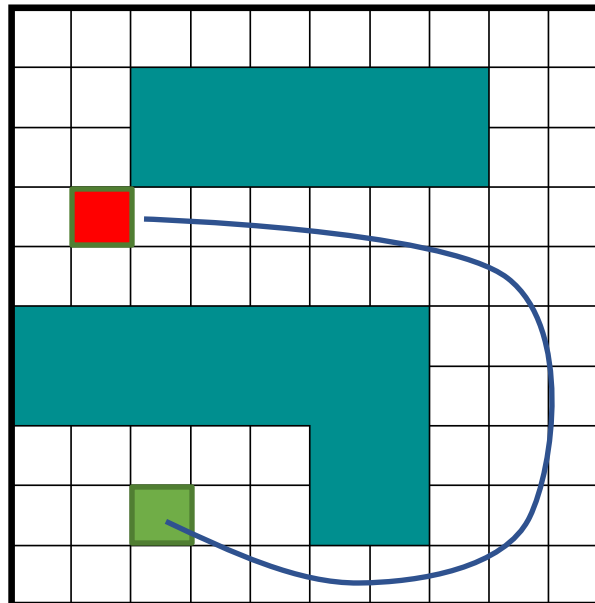
Food provided!

Agenda

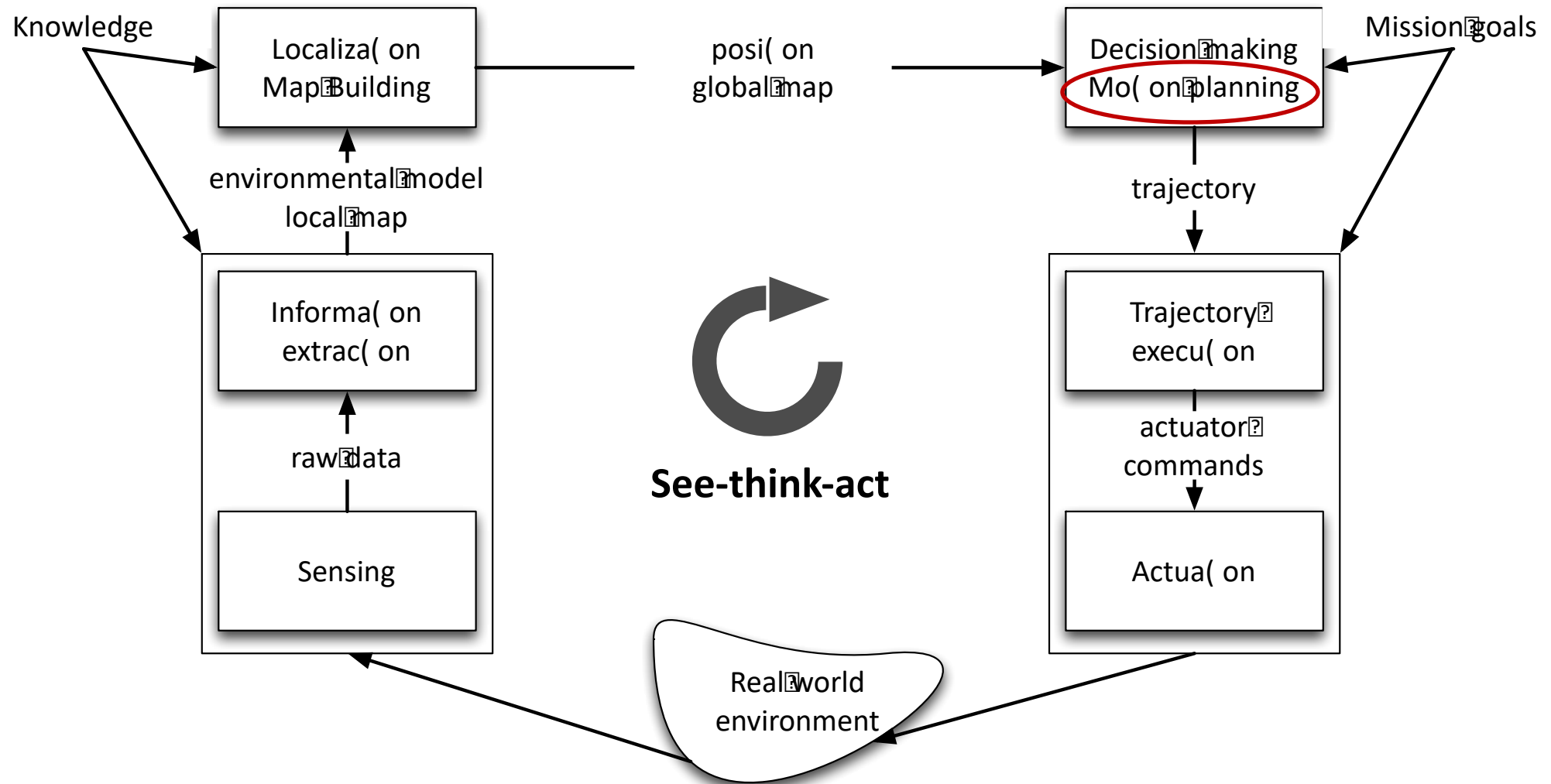
- Agenda
 - Introduction to motion planning
 - Search-based algorithms for motion planning
 - Configuration spaces and combinatorial motion planning
- Readings:
 - Chapter 4, sections 4.1 – 4.3 in D. Gammelli, J. Lorenzetti, K. Luo, G. Zardini, M. Pavone. *Principles of Robot Autonomy*. 2026.

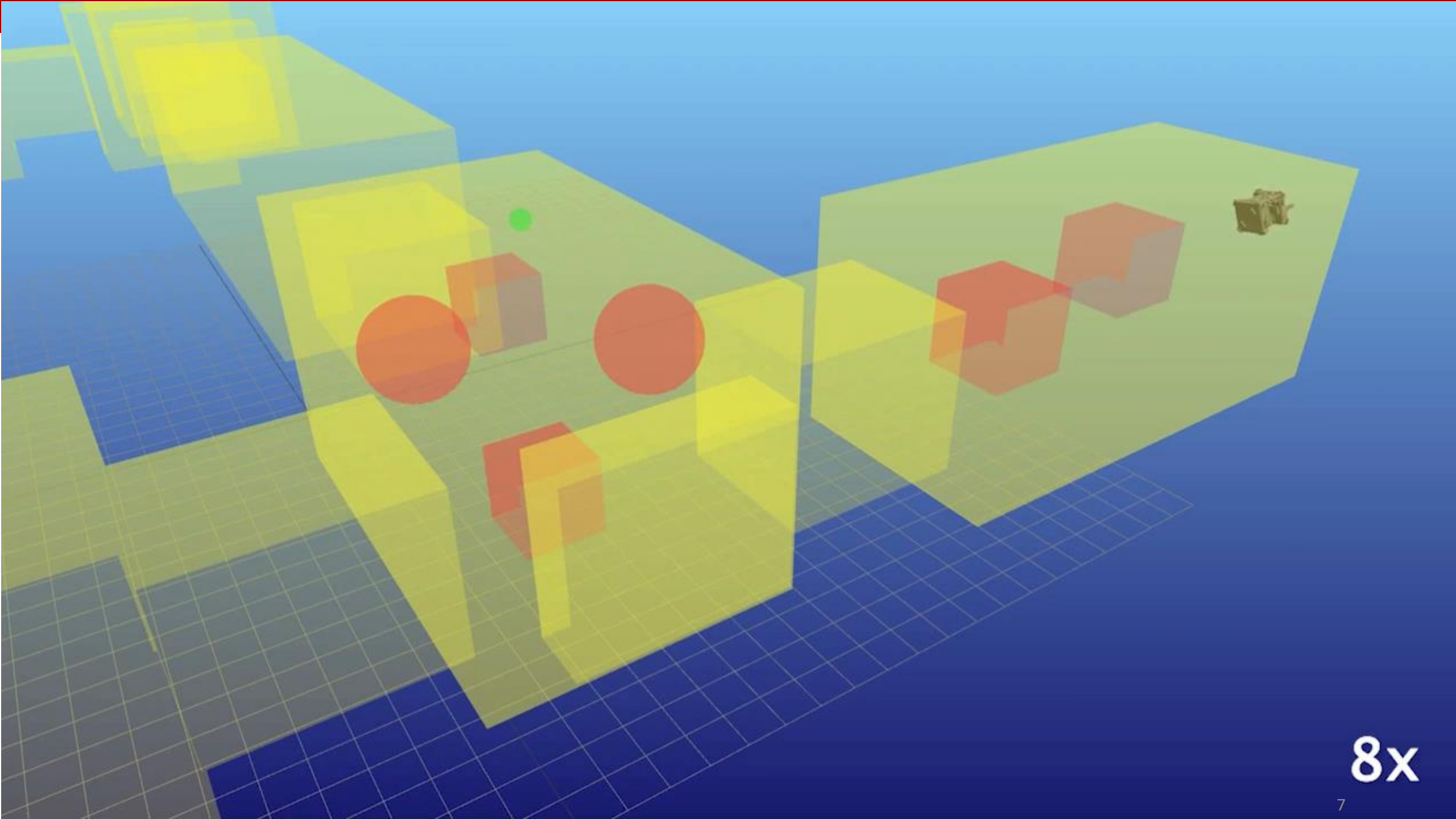
Motion planning

Problem definition: Compute sequence of actions that drives a robot from an initial condition to a terminal condition while avoiding obstacles, respecting motion constraints, and *possibly* optimizing a cost function



The see-think-act cycle

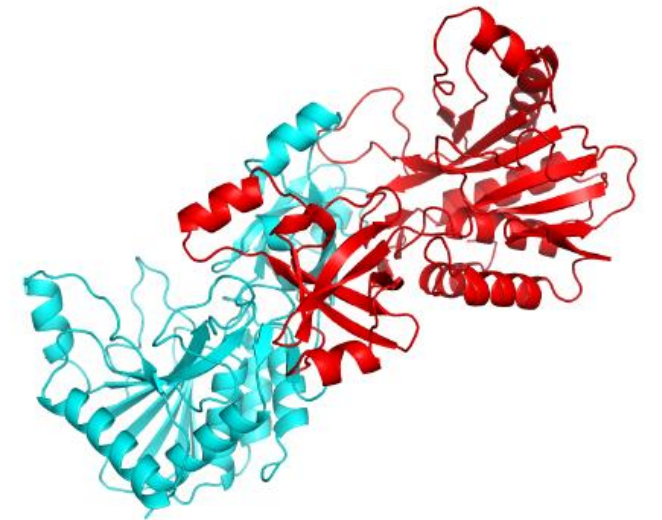
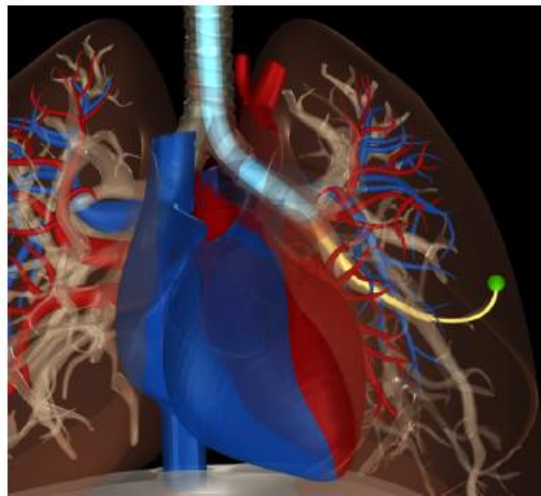
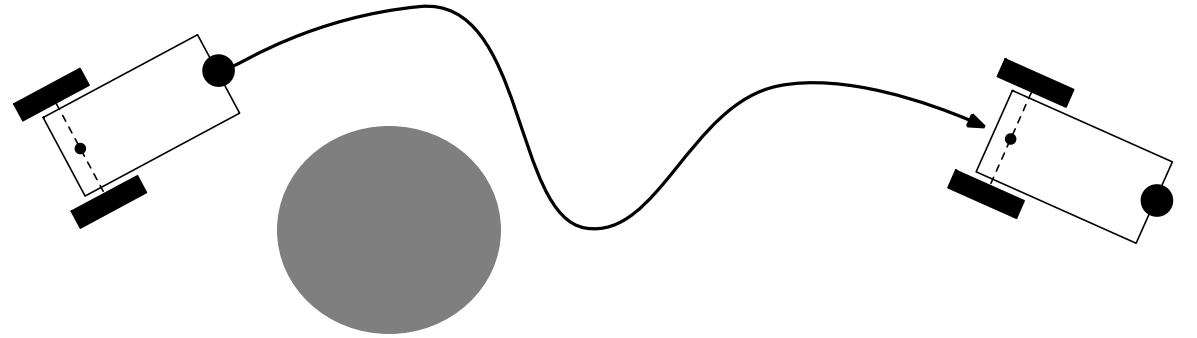




8x

More examples of motion planning

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

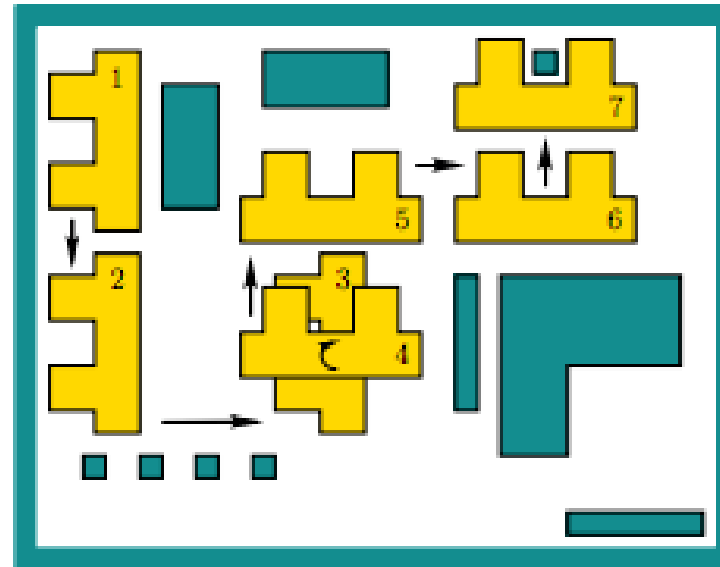
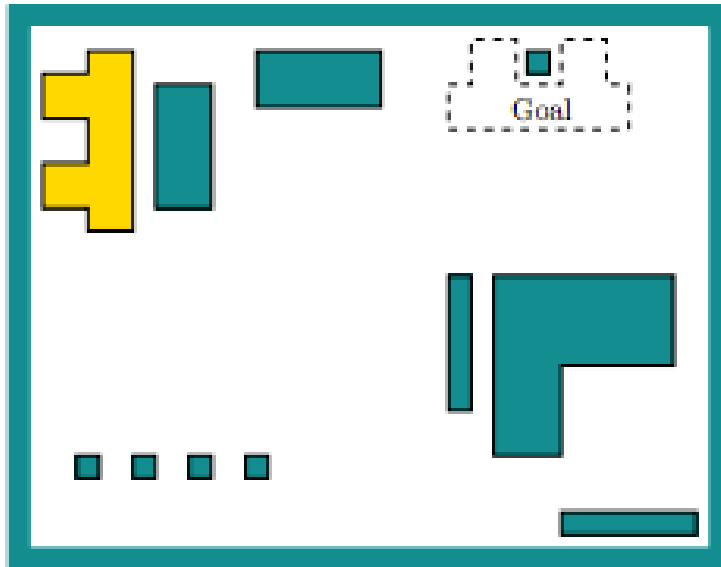


Some history

- Formally defined in the 1970s
- Development of exact, combinatorial solutions in the 1980s
- Development of sampling-based methods in the 1990s
- Deployment on real-time systems in the 2000s
- Current research: inclusion of differential and logical constraints, planning under uncertainty, parallel implementation, and more

Simplest setup

- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- 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

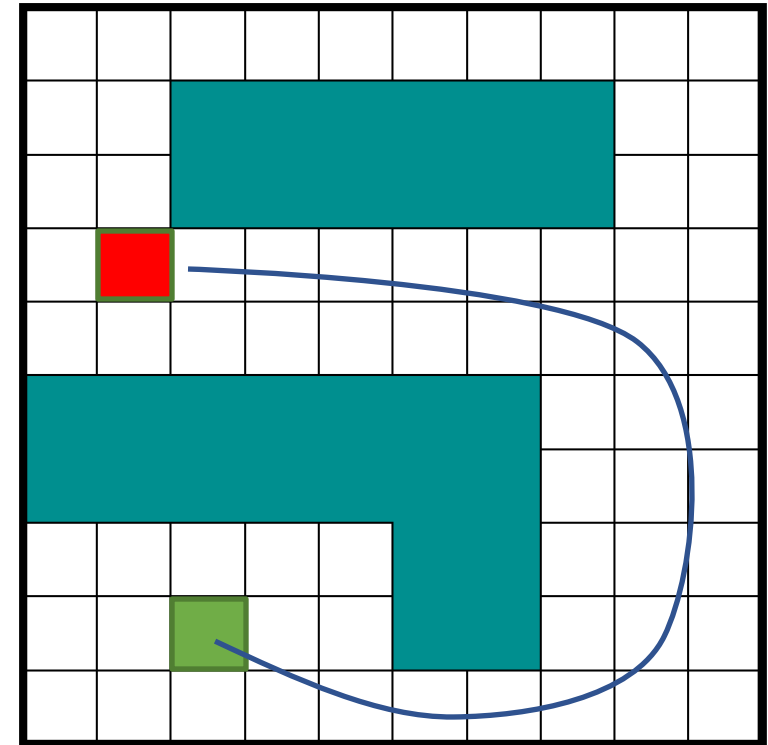


Popular approaches

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

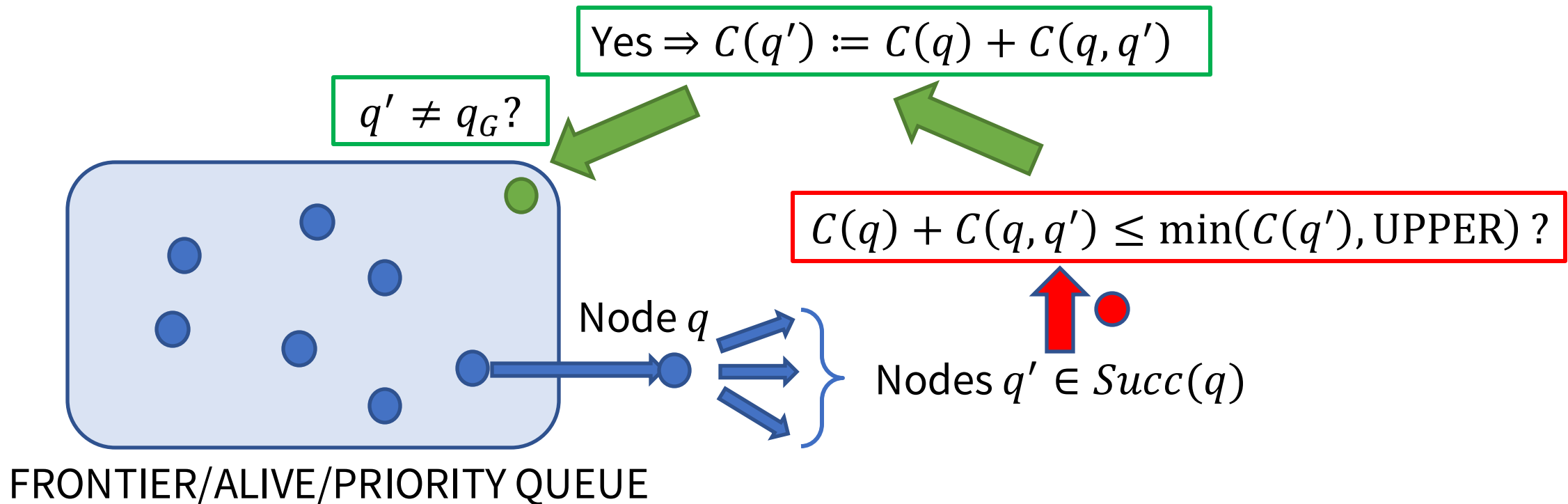
Grid-based approaches

- 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 $G = (V, E)$
 - Each vertex $v \in V$ represents a free cell
 - Edges $(v, u) \in E$ connect adjacent grid cells



Graph search algorithms

- Having determined decomposition, how to find “best” path?
- **Label-Correcting Algorithms:** $C(q)$: *cost-of-arrival* from q_I to q



Label correcting algorithm

Step 1. Remove a node q from frontier queue and for each child q' of q , execute step 2

Step 2. If $C(q) + C(q, q') \leq \min(C(q'), \text{UPPER})$, set $C(q') := C(q) + C(q, q')$ and set q to be the parent of q' . In addition, if $q' \neq q_G$, place q' in the frontier queue if it is not already there, while if $q' = q_G$, set UPPER to the new value $C(q) + C(q, q_G)$

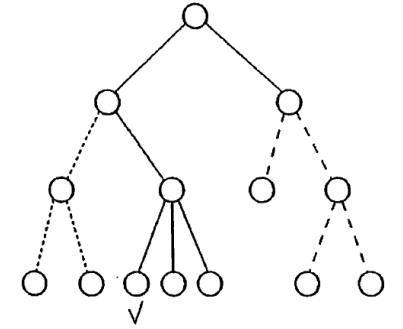
Step 3. If the frontier queue is empty, terminate, else go to step 1

Initialization: set the labels of all nodes to ∞ , except for the label of the origin node, which is set to 0

GetNext() ?

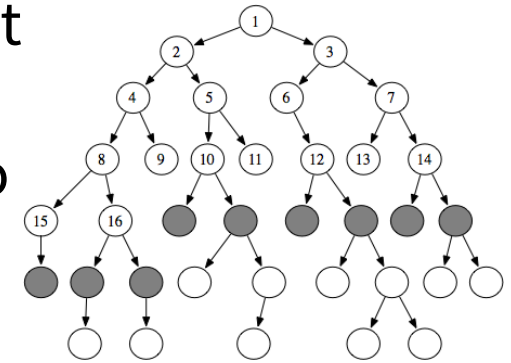
Depth-First-Search (DFS): Maintain Q as a **stack** – Last in/first out

- Lower memory requirement (only need to store part of graph)



Breadth-First-Search (BFS, Bellman-Ford): Maintain Q as a **list** – First in/first first out

- Update cost for all edges up to current depth before proceeding to greater depth
- Can deal with negative edge (transition) costs



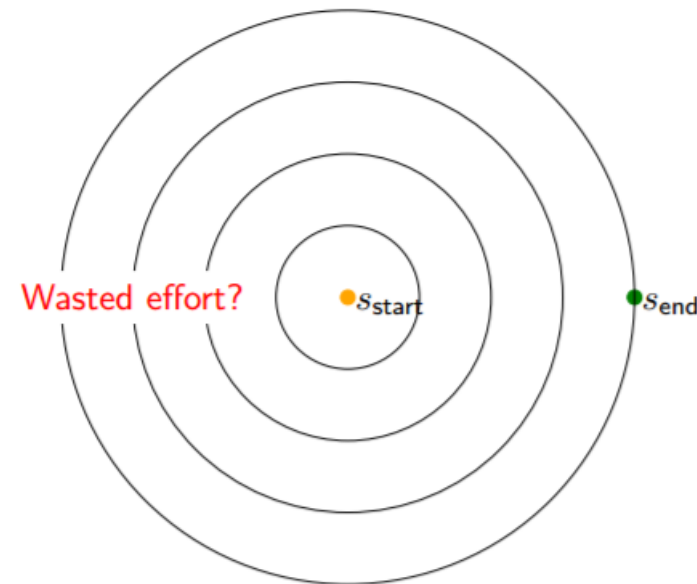
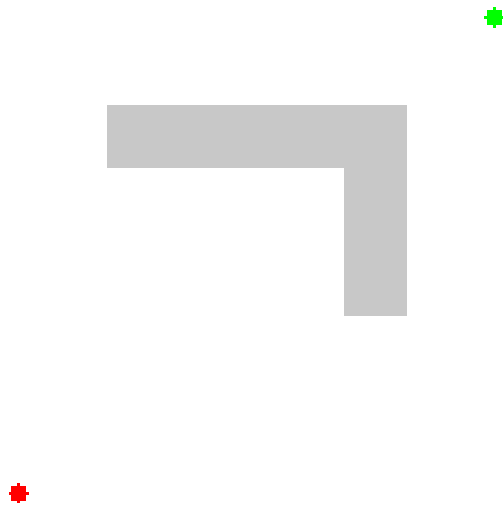
Best-First (BF, Dijkstra): Greedily select next q : $q = \operatorname{argmin}_{q \in Q} \mathcal{C}(q)$

- Node will enter the frontier queue at most *once*
- Requires costs to be non-negative

Correctness and improvements

Theorem

If a feasible path exists from q_I to q_G , then algorithm terminates in finite time with $C(q_G)$ equal to the optimal cost of traversal, $C^*(q_G)$.



* https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

A*: Improving Dijkstra

- Dijkstra orders by optimal “*cost-to-arrival*”
- Faster results if order by “*cost-to-arrival*”+ (approximate) “*cost-to-go*”
- That is, order by

$$C(q) + h(q)$$

where $h(q)$ is *a* heuristic for optimal cost-to-go (specifically, a positive *underestimate*)

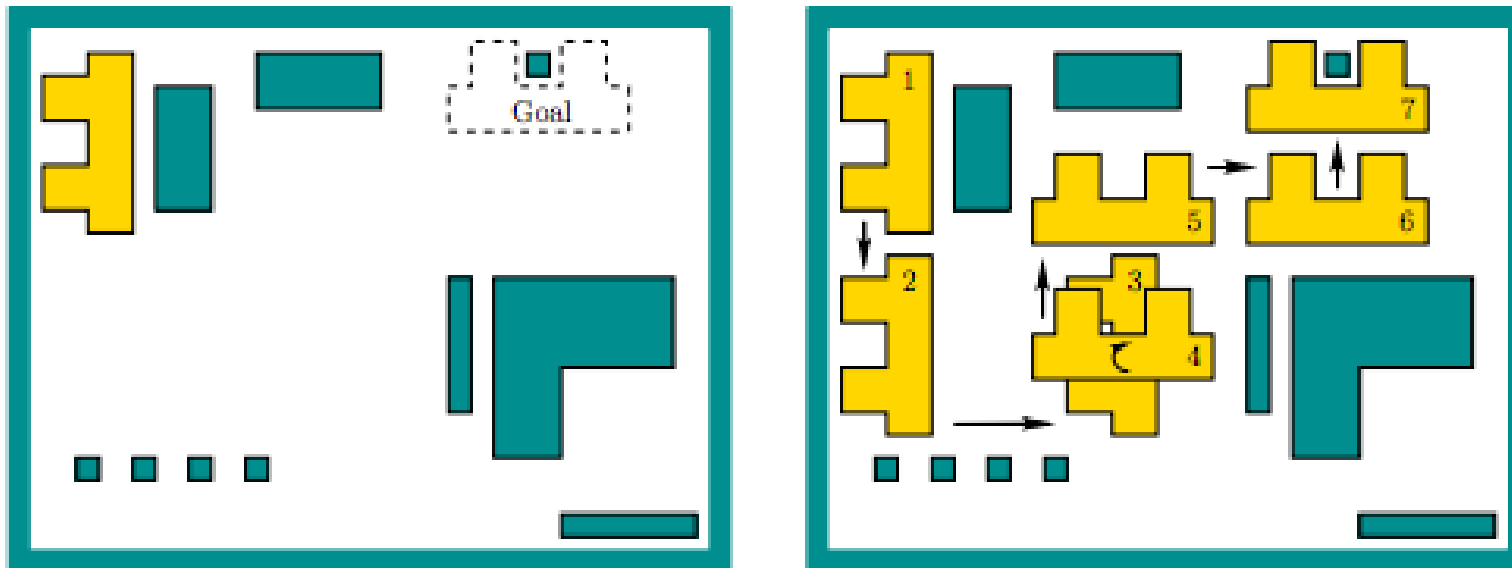
- This modification still guarantees that the algorithm will terminate with a shortest path
- Many variations are possible...

Grid-based approaches: summary

- Pros:
 - Simple and easy to use
 - Fast (for some problems)
- Cons:
 - Resolution dependent
 - Not guaranteed to find solution if grid resolution is not small enough
 - Limited to simple robots
 - Grid size is exponential in the number of DOFs

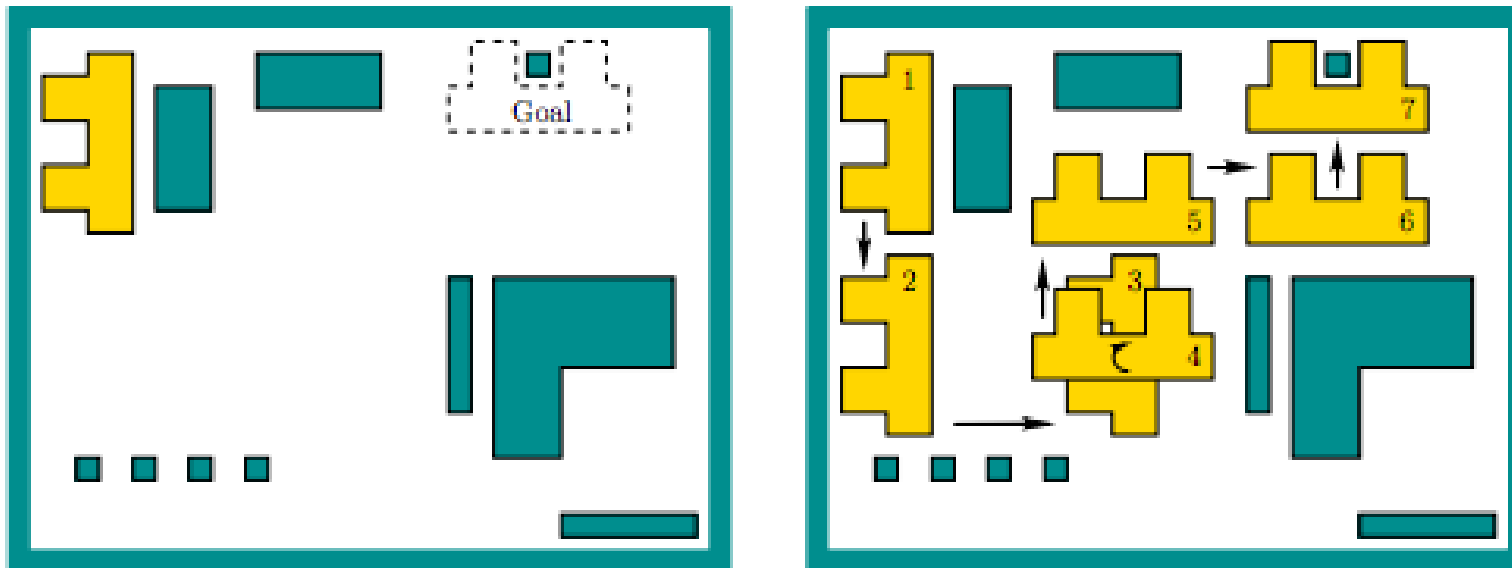
Back to continuous motion planning

- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- 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



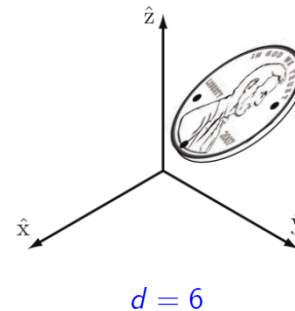
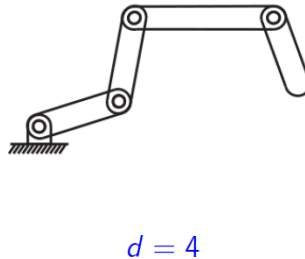
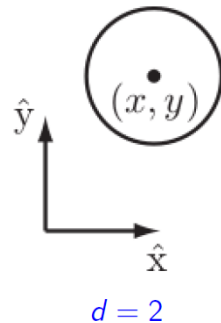
Back to continuous motion planning

Key point: motion planning problem described in the real-world, but it really lives in another space -- the **configuration** (C-) space!



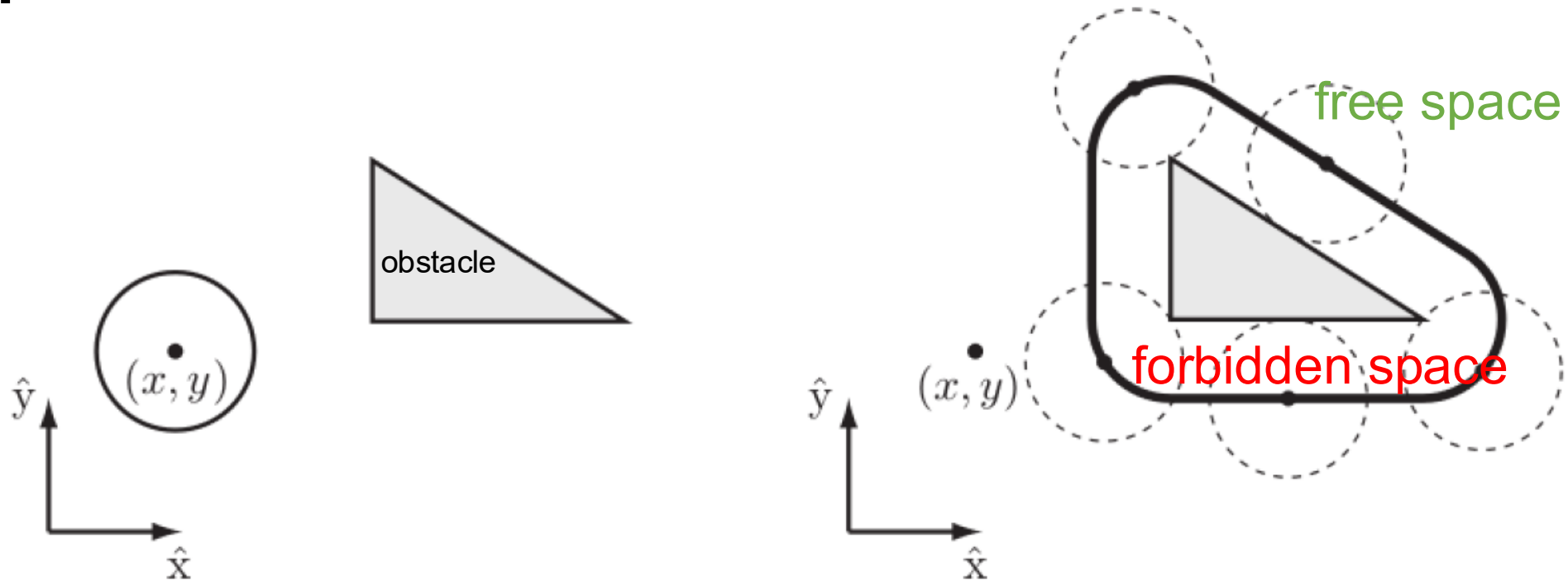
Configuration space

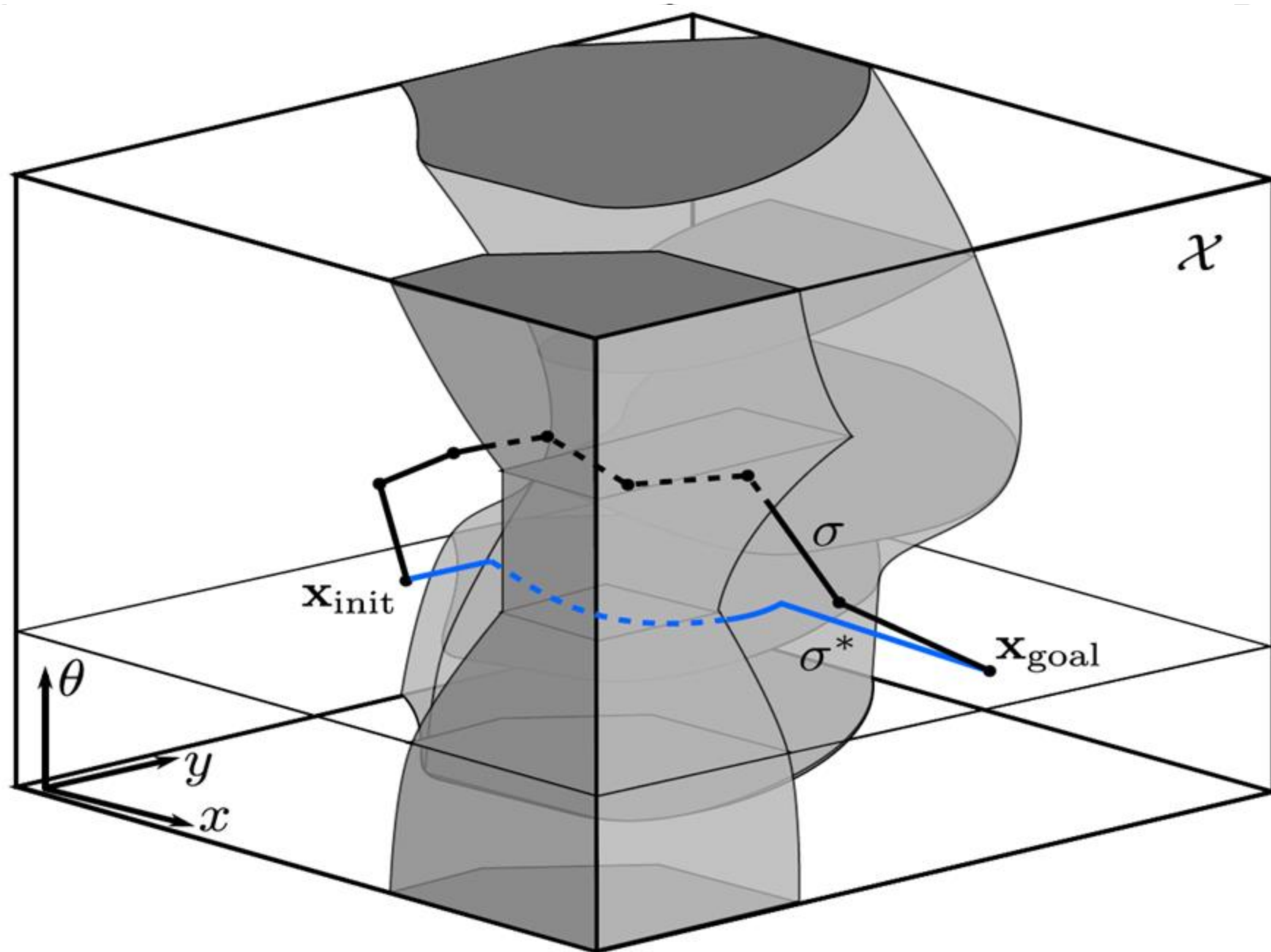
- **C- space**: captures all degrees of freedom (all rigid body transformations)
- More in detail, let $\mathcal{R} \subset \mathbb{R}^2$ be a polygonal robot (e.g., a triangle)
- The robot can rotate by angle θ or translate $(x_t, y_t) \in \mathbb{R}^2$
- Every combination $q = (x_t, y_t, \theta)$ yields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



Configuration free space

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

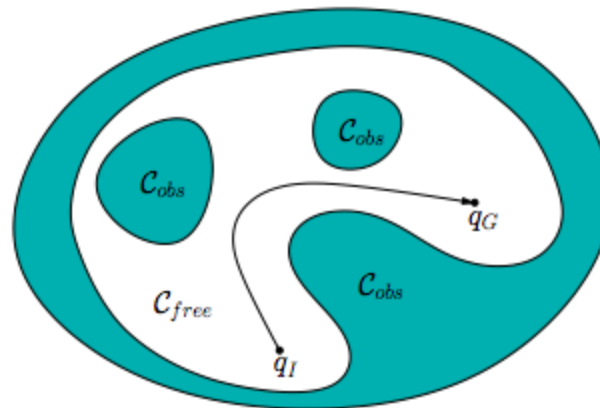




Bottom line: *explicitly* computing C free spaces in high-dimensional settings is hard!

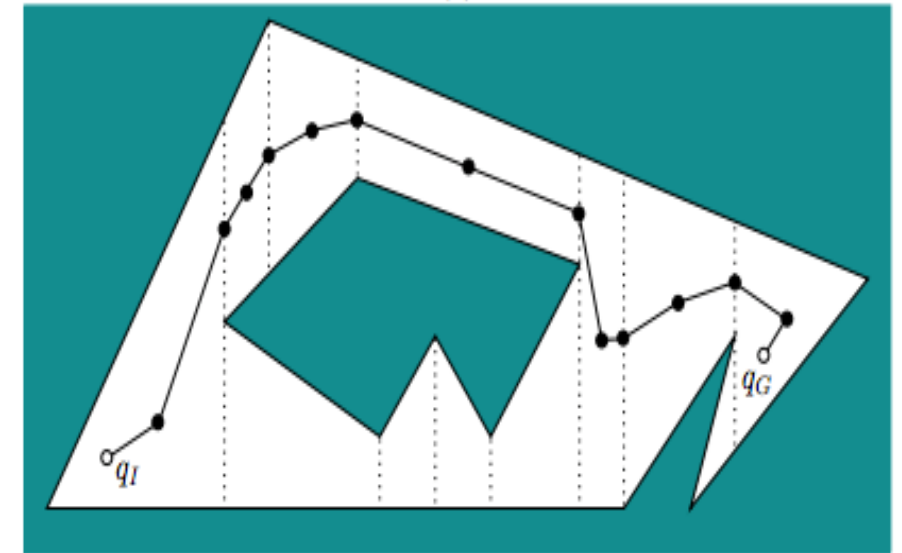
Planning in C -space

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



Combinatorial planning

- Combinatorial approaches to motion planning find paths through continuous configuration space **without resorting to approximations**
- **Key idea:** compute a roadmap, which provides a discrete representation of continuous motion planning problem without losing any of the original connectivity information needed to solve it
- Such approaches are typically complete (i.e., guaranteed to find a solution), but are typically limited to **small number of DOFs** due to the challenge of *exactly* computing C free spaces



A roadmap 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

Next time: sampling-based planning

