اصول علم ربات – اسلاید شانزدهم

Fundamentals of Robotics – Slide 16

Motion planning 1

دکتر مهدی جوانمردی

زمستان–بهار ۱۴۰۱

[slides adapted from Gianni Di Caro, @CMU with permission]

Planning

Robot: *goal oriented machine that can sense, plan and act*

To plan or not to plan, that is the question ….

- ***Planning*:** Sequence of actions to achieve a given goal (usually without timing specifications), or towards achieving a given goal by achieving a set of sub-goals
- ***Scheduling*:** Planning + Timing (usually without space considerations)
- ***Task planning*:** Sequence of actions that accomplish a large goal (e.g., building a house)
- ***Motion Planning*:** Generation of motions through space …

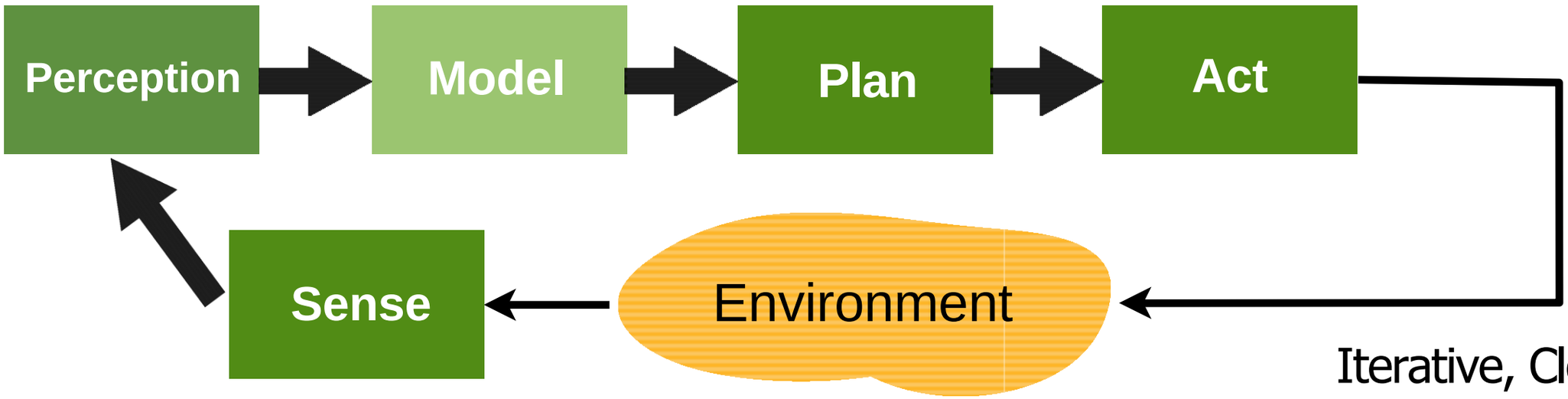
Planning is a **top-down** approach to problem solving that requires a (reliable) ***model*** of the world / problem, to be able to effectively *reason* about it!

Deliberative robot control architectures

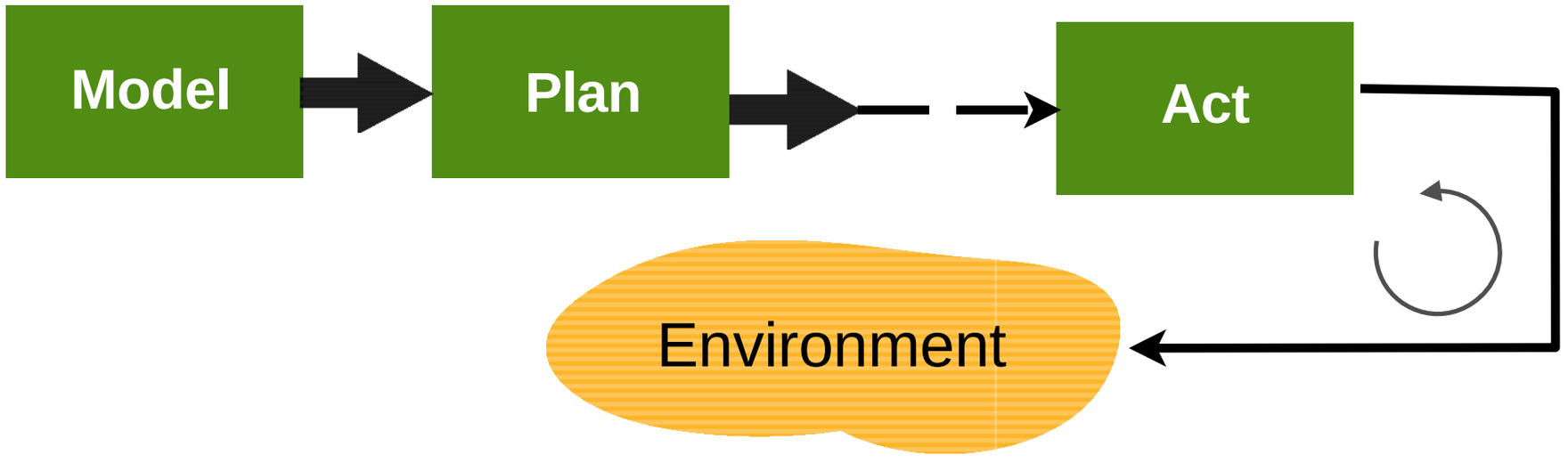
Deliberation: Thoughtfulness in decision and action

Thinking hard: model, reason and plan

Top-down approach to problem solving

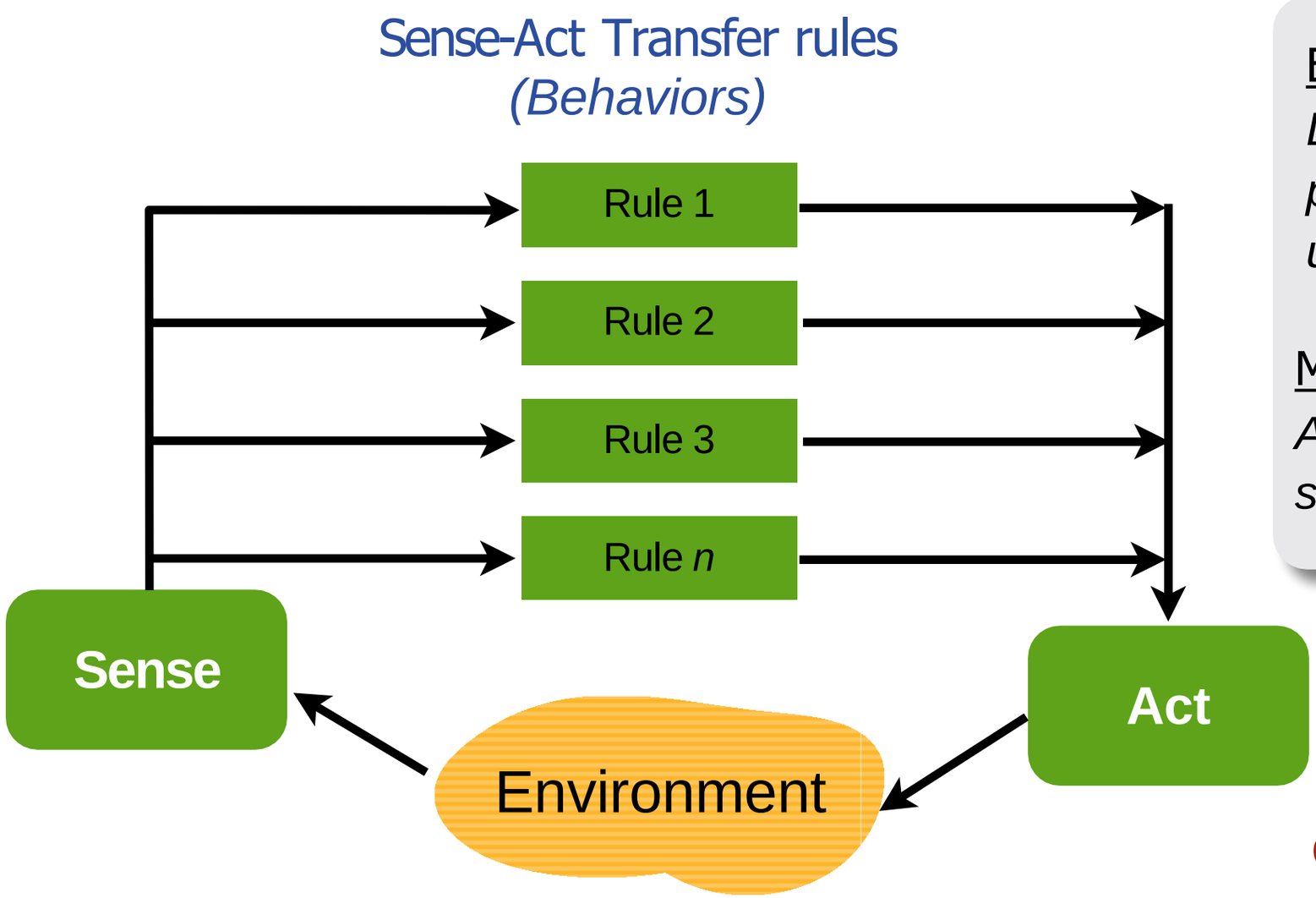


Iterative, Closed-loop
SENSE-(MODEL)-PLAN-ACT



Open-loop
[MODEL-PLAN]-ACT

Reactive control architectures: Don't Think, React!

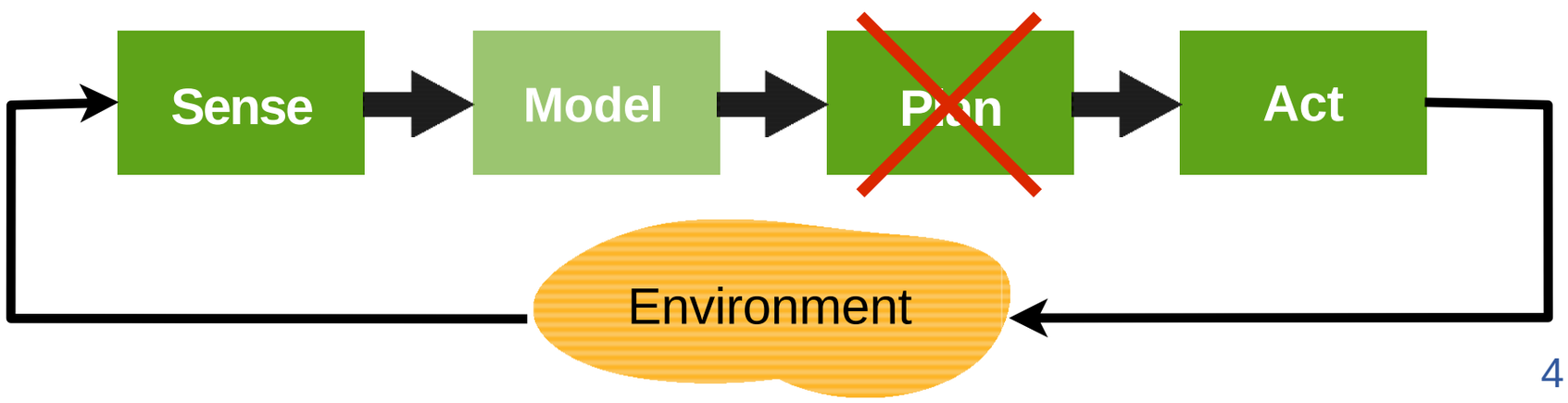


Ethological view (***Behavior***):
Direct mapping of sensory inputs to a pattern of motor actions that are then used to achieve a task

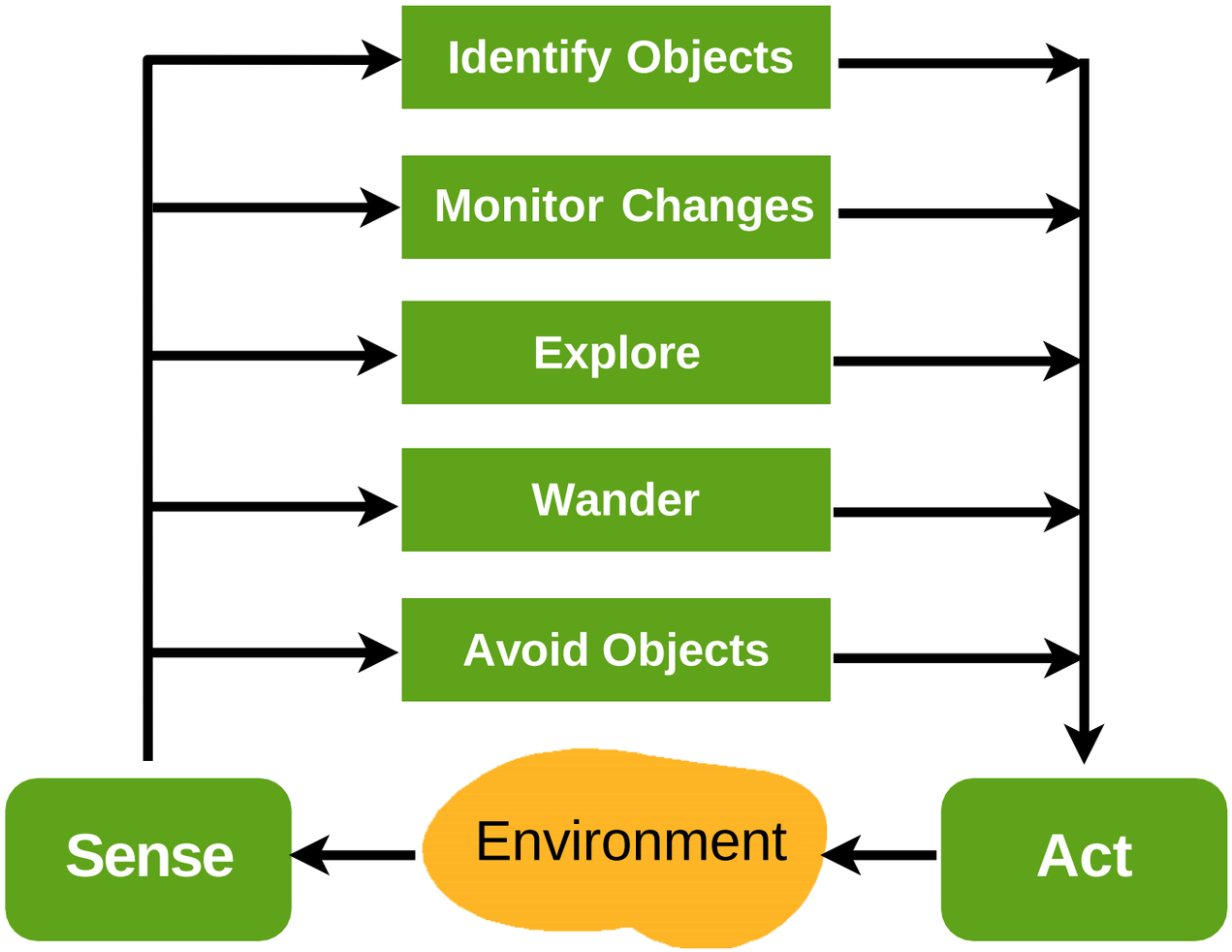
Mathematical view (***Function***):
A transfer function, transforming sensory inputs into actuator commands

Vertical decomposition
vs.
Horizontal decomposition

Concurrent mode
vs.
Sequential mode

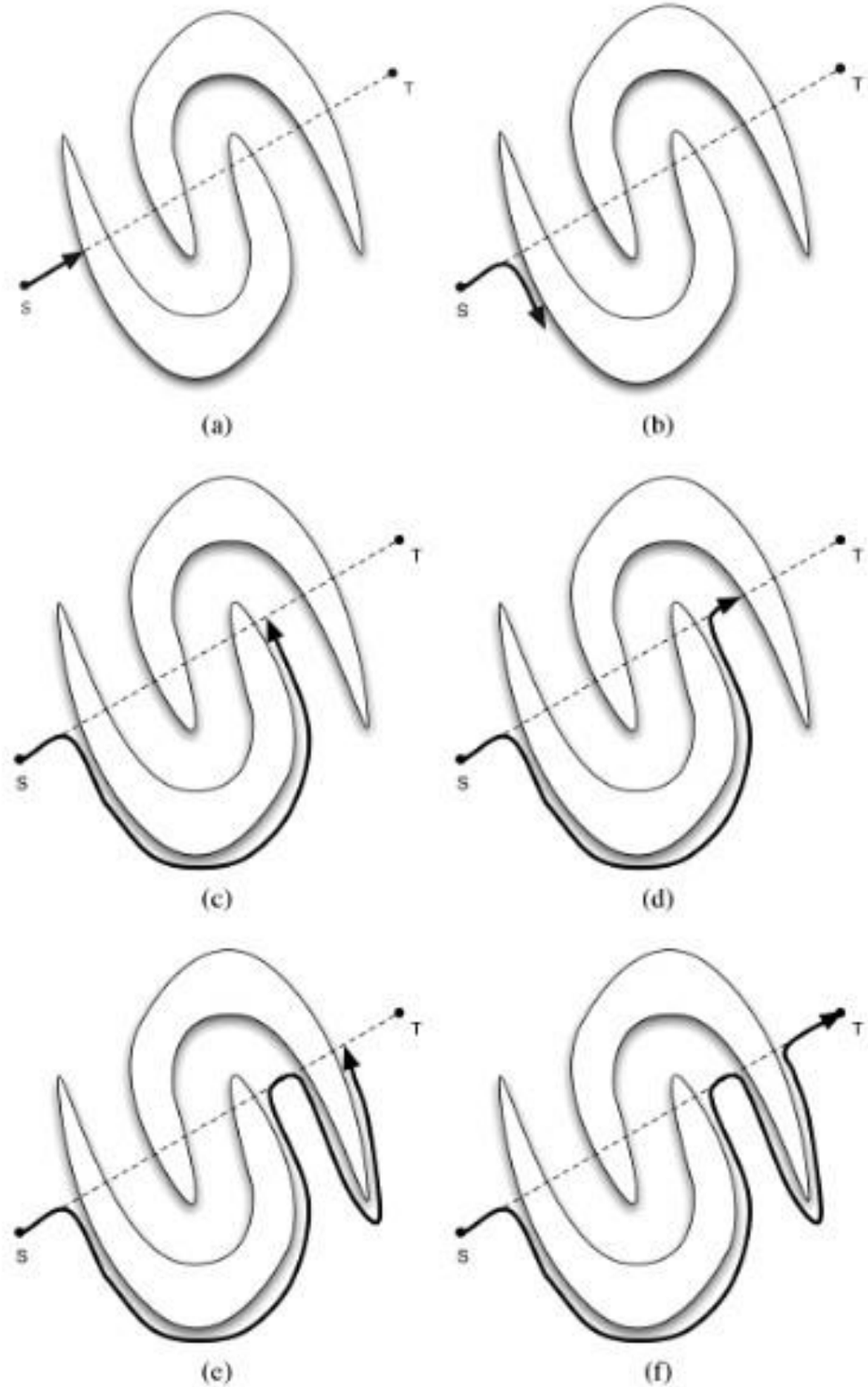


Reactive control architectures: Don’t Think, React!

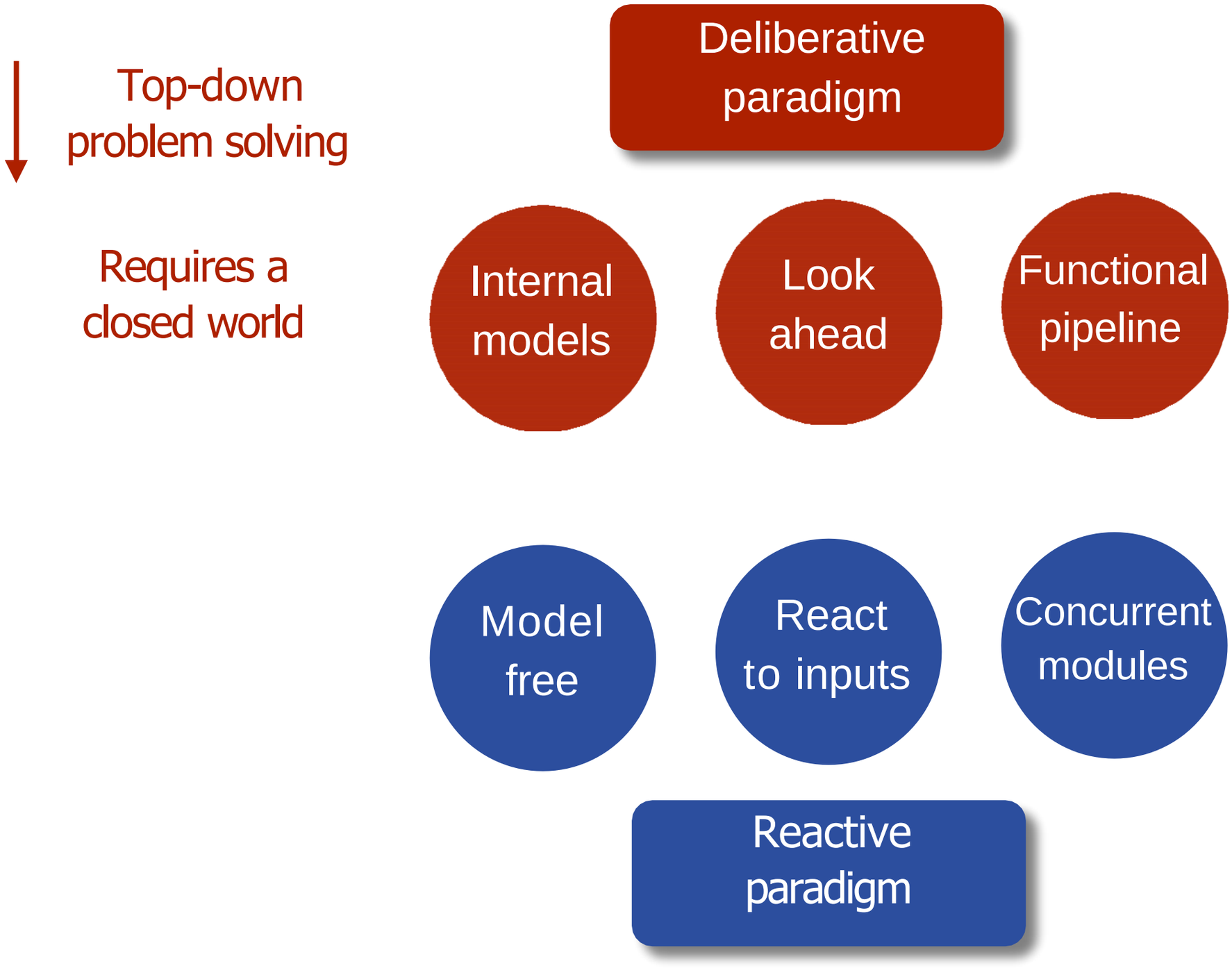


A navigation behavior

Bug algorithms:
Example of reactive navigation

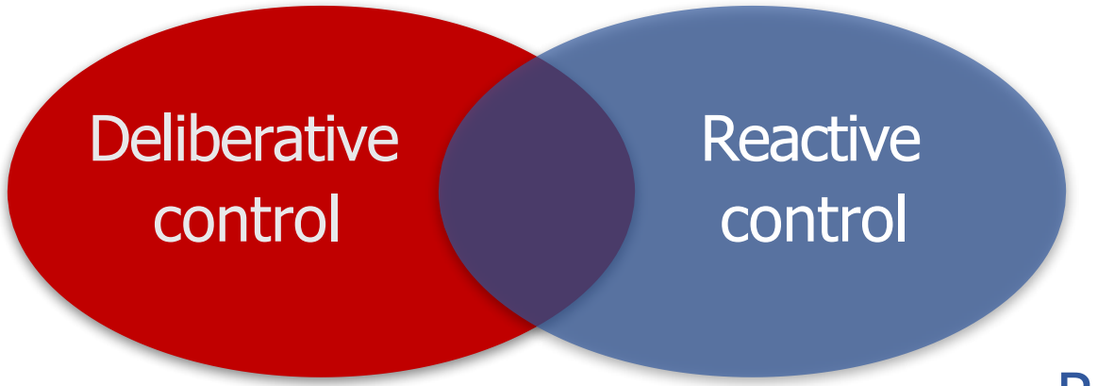


Deliberative vs. Reactive control architectures



Hybrid control architectures

The best of the two worlds!



Representations
Models
Planning

Hybrid control

Real-time
Multiple goals
Robustness
Flexibility
Modularity

Strategic
planning / reasoning

How to combine ?

Low(er)-level
controls and behaviors

… the most used approach nowadays, but still a sort of art

Motion planning

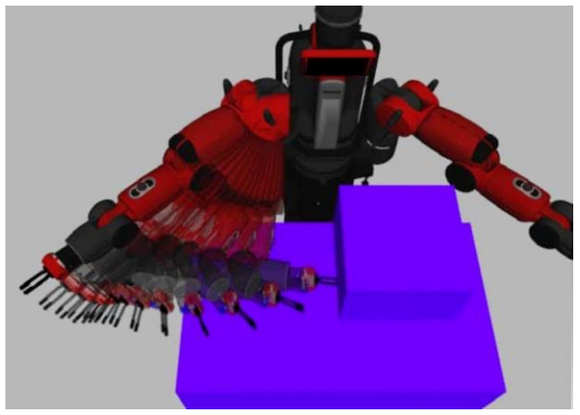
- ***Motion Planning:*** Specification of motion through space (and time)

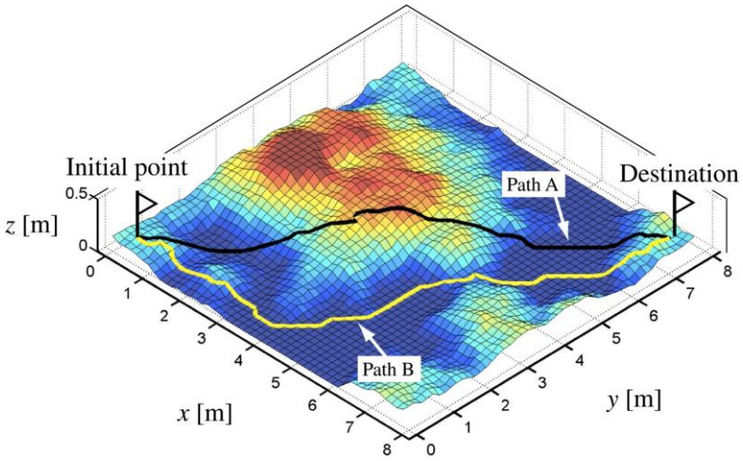
Motion planning

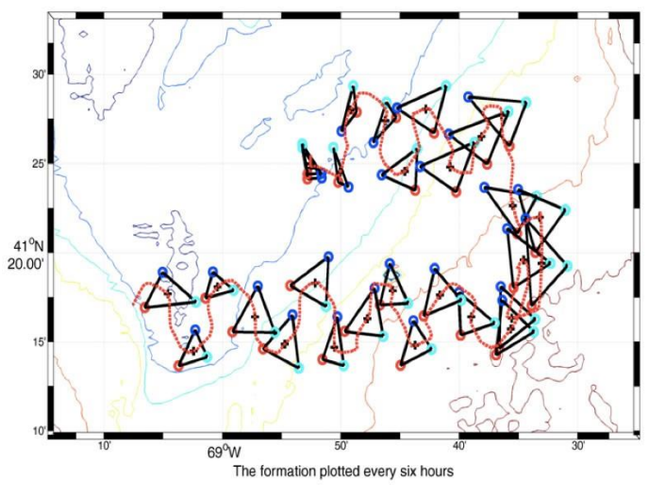
- ***Motion Planning:*** Specification of motion through space (and time)

Motion planning

- ***Motion Planning:*** Specification of motion through space (and time)
  - **Path planning**: Generation of a (feasible) path for going from configuration A to B
  - **Trajectory planning:** Generation of a path for going from A to B specifying how to move based on velocity, time, differential constraints, and dynamics
  - **Coverage planning:** Visit (or observe with sensors) all places (at least or only once)
  - **…**







Open Motion Planning Library

http://ompl.kavrakilab.org/gallery.html

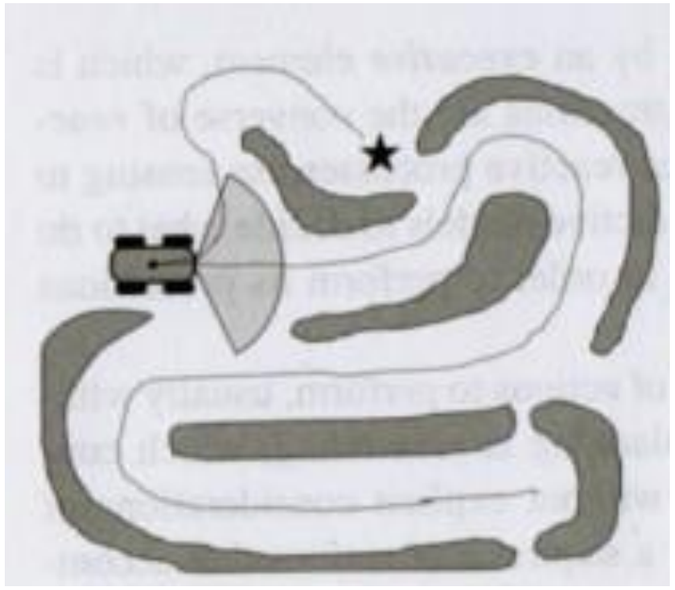
General parameters / design choices

- How far should we look into the future? **Depth of look ahead**
  The farthest, the more *reliable* the plan is and the more *expensive* computation is

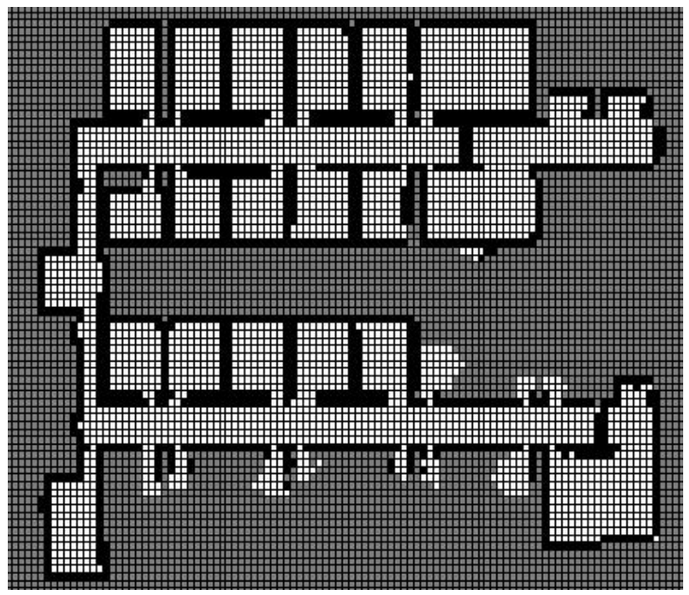
  ✓ Model Predictive Control (MPC): iterative replanning up to a certain horizon

- What *spatial resolution* map model? **Accuracy of spatial representation**
  The higher the resolution the higher the computational requirements

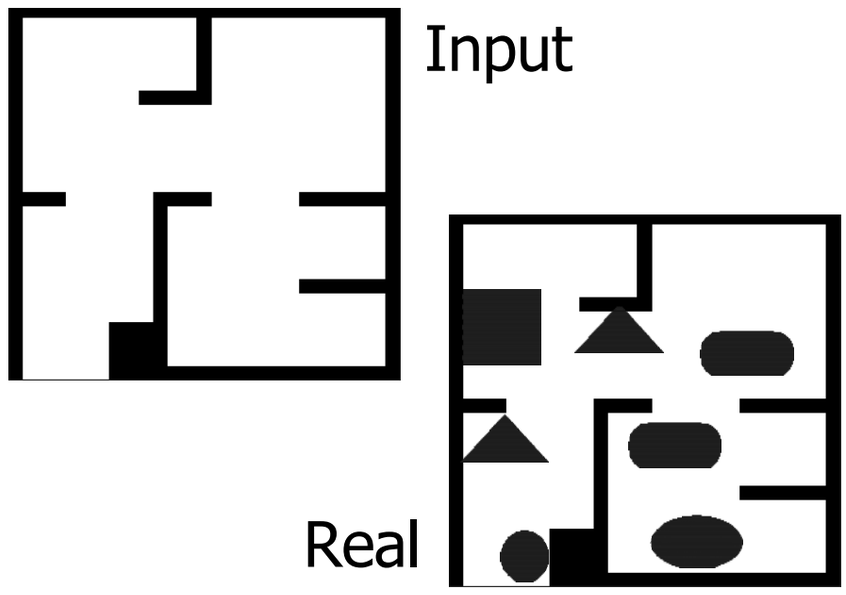
- How realistic / reliable is the input model (the map)?
  What are the effects of **imprecise or partial models**?



Without planning / looking ahead things can get arbitrarily bad!



An extremely fine-grained representation (occupancy grid) might require heavy computation



A plan good / feasible in the incomplete input map could easily be not feasible in the real environment

Path planning: Problem formulation

Inputs:

- **Start pose** of the robot
- **Goal pose**
- Geometric and kinematic **description of the robot**
- (Geometric) **description of the world** including existing obstacles

Goal(s):

- Find a **path that complies with robot motion constraints** (***feasibility***) and moves the robot gradually from start to goal poses **never colliding with an obstacle** (***admissibility***)
- If no feasible + admissible solution exist ➔ Report a *failure*
- ***Optimality:*** If more than one solution exists, select the best
- ***Completeness:*** If at least one feasible + admissible path exists, the path planning algorithm is able to *find it.*

Optimality metrics: **Shortest distance**, **Minimum time**, Minimum curvature, Max smoothness, Maximal safety, Min/Max …

Free and occluded space

❖ **Obstacle region**: $\mathcal{O},\ \mathcal{O} \subseteq \mathcal{W}$

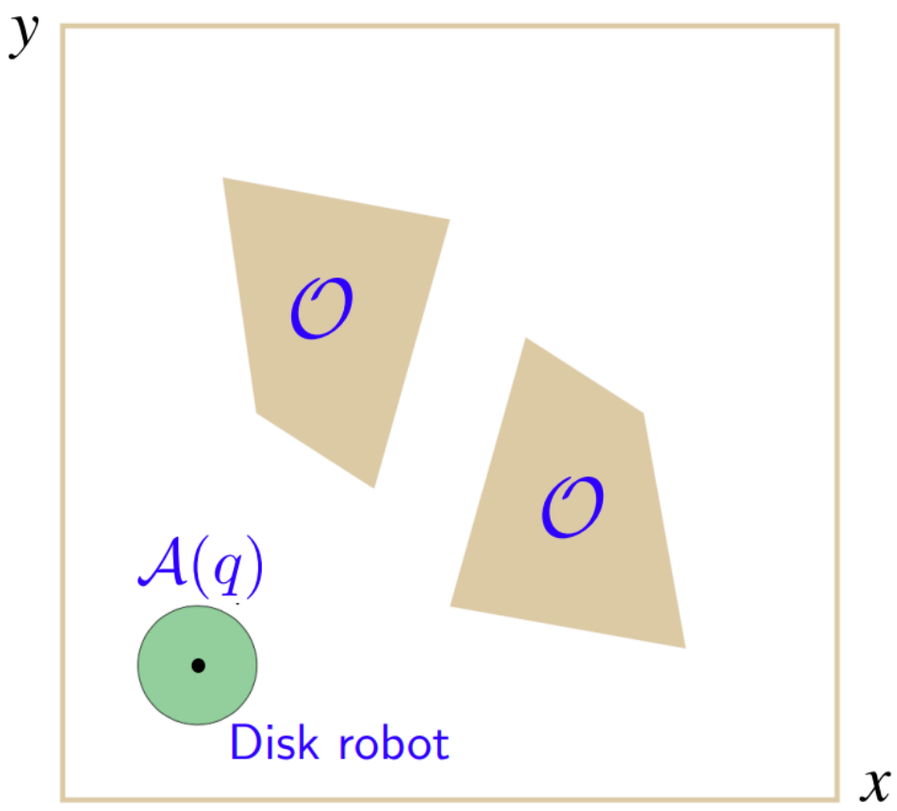
set of all points in the workspace that lie in one or more **obstacles**

- In general, we will safely assume that $\mathcal{W} \subseteq \mathbb{R}^2$ or $\mathcal{W} \subseteq \mathbb{R}^3$

❖ **Robot**: $\mathcal{A},\ \mathcal{A} \subseteq \mathcal{W}$

set of all points in the workspace that are covered/occupied by the robot

- $\mathcal{A}(q)$ is the set of workspace points occupied by the robot when in configuration q



$\mathcal{W} \subset \mathbb{R}^2$

$q = (x, y), \quad \mathcal{C} = \mathcal{W} \backslash \mathcal{O}$

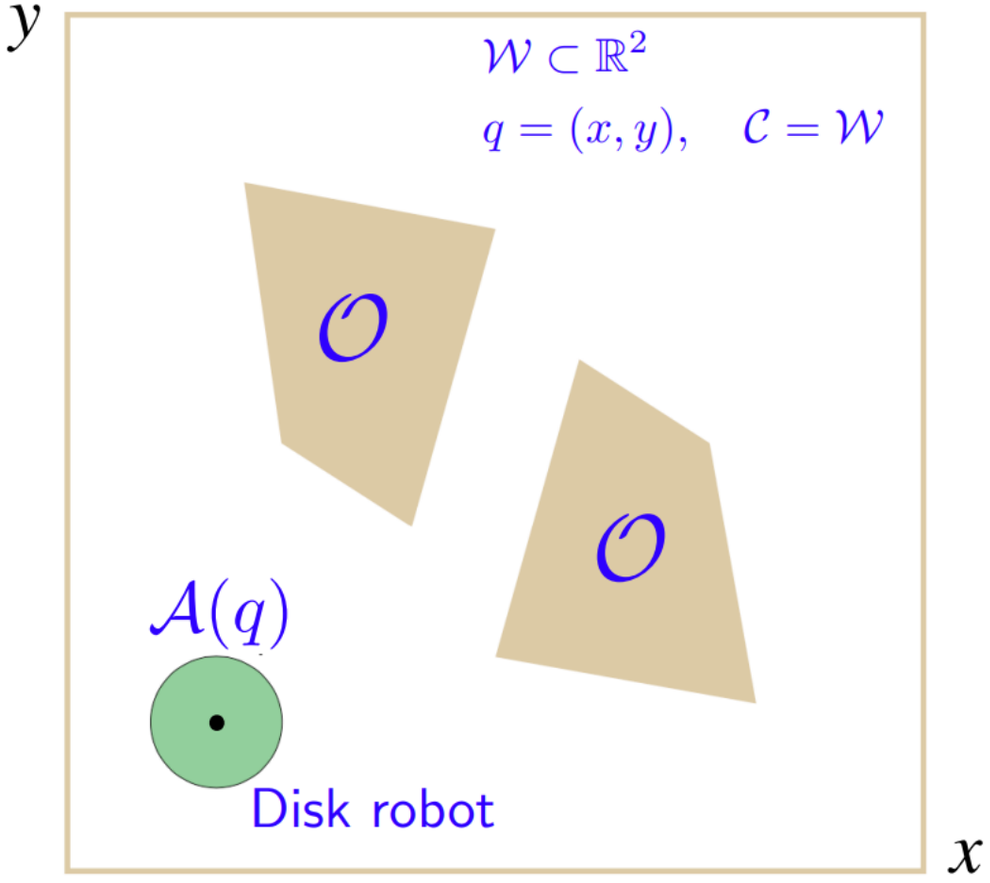
Free and occluded space configuration space

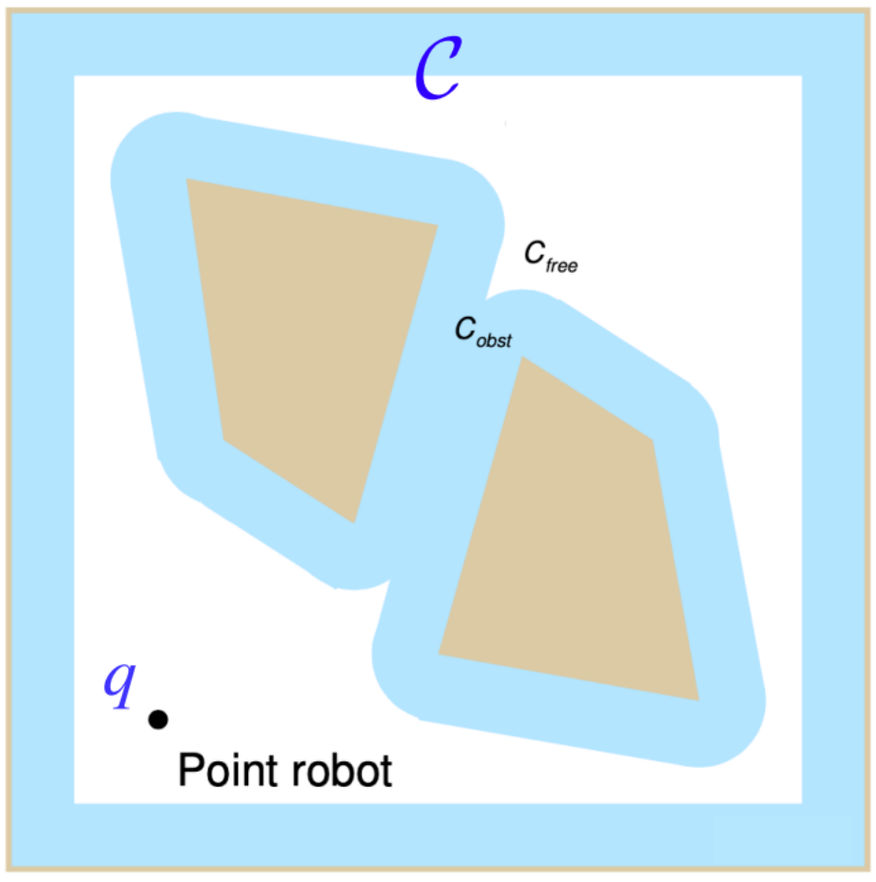
- **Obstacle region for robot in *configuration space***: $\mathcal{C}_{obs} \subseteq \mathcal{C}$

  set of all configurations q at which the *transformed* robot $\mathcal{A}(q)$ would intersect the obstacle region $\mathcal{O}$

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

- **Free space region for robot in *configuration space***: $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$

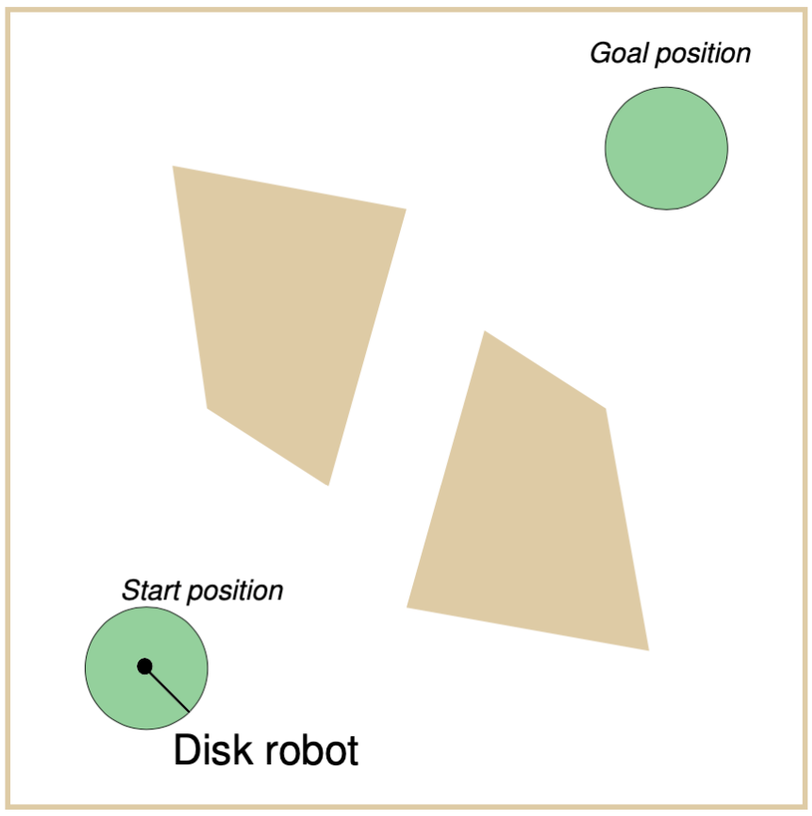




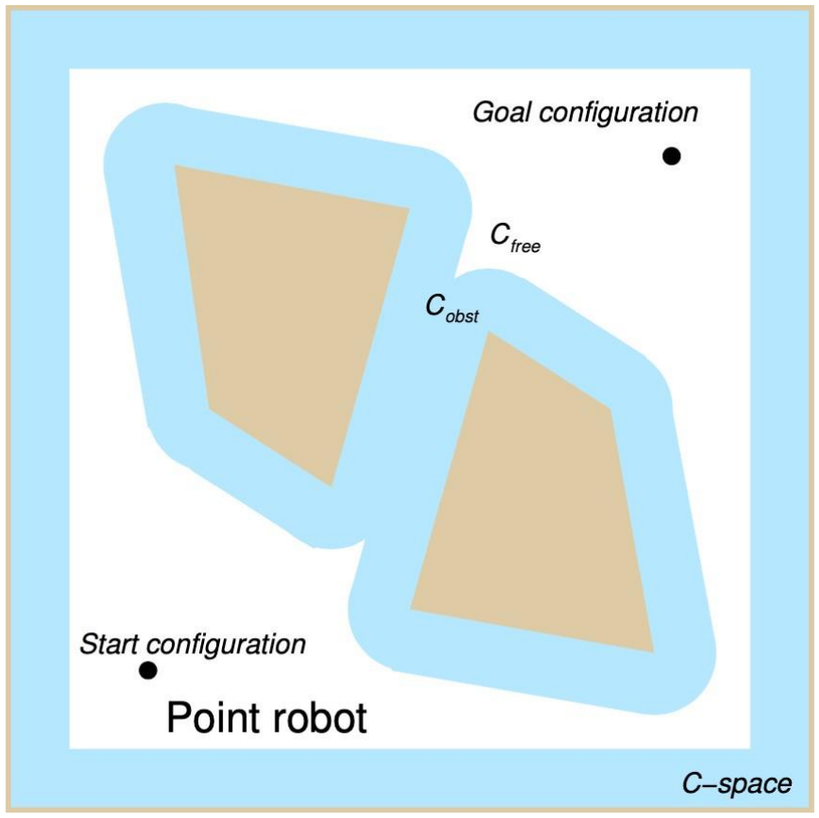
$\mathcal{C}_{obs}$ +

Motion planning: Point robot in configuration space

➡ For the purpose of **motion planning**, once geometry of robot and of environment, including the obstacles, are represented in **configuration space**, the robot is conveniently reduced to a **point**, *since its geometry is already accounted in* C_{free}



Motion planning problem in geometrical representation of $\mathcal{W}$



Motion planning problem in $\mathcal{C}$-space representation

C_{obs} has been obtained by **enlarging the obstacles** by the disk of radius

By applying Minkowski sum: $\mathcal{O} \oplus \mathcal{A} = \{x + y \mid x \in \mathcal{O}, y \in \mathcal{A}\}$

Free space and obstacle region in the c-space

- With $\mathcal{W}$ (e.g., $\mathcal{W} = \mathbb{R}^m$) being the workspace, $\mathcal{O} \subseteq \mathcal{W}$ the set of obstacles, $\mathcal{A}(q)$ the robot in configuration $q \in \mathcal{C}$, then:

$$\mathcal{C}_{free} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} = \emptyset\}$$

$$\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$$

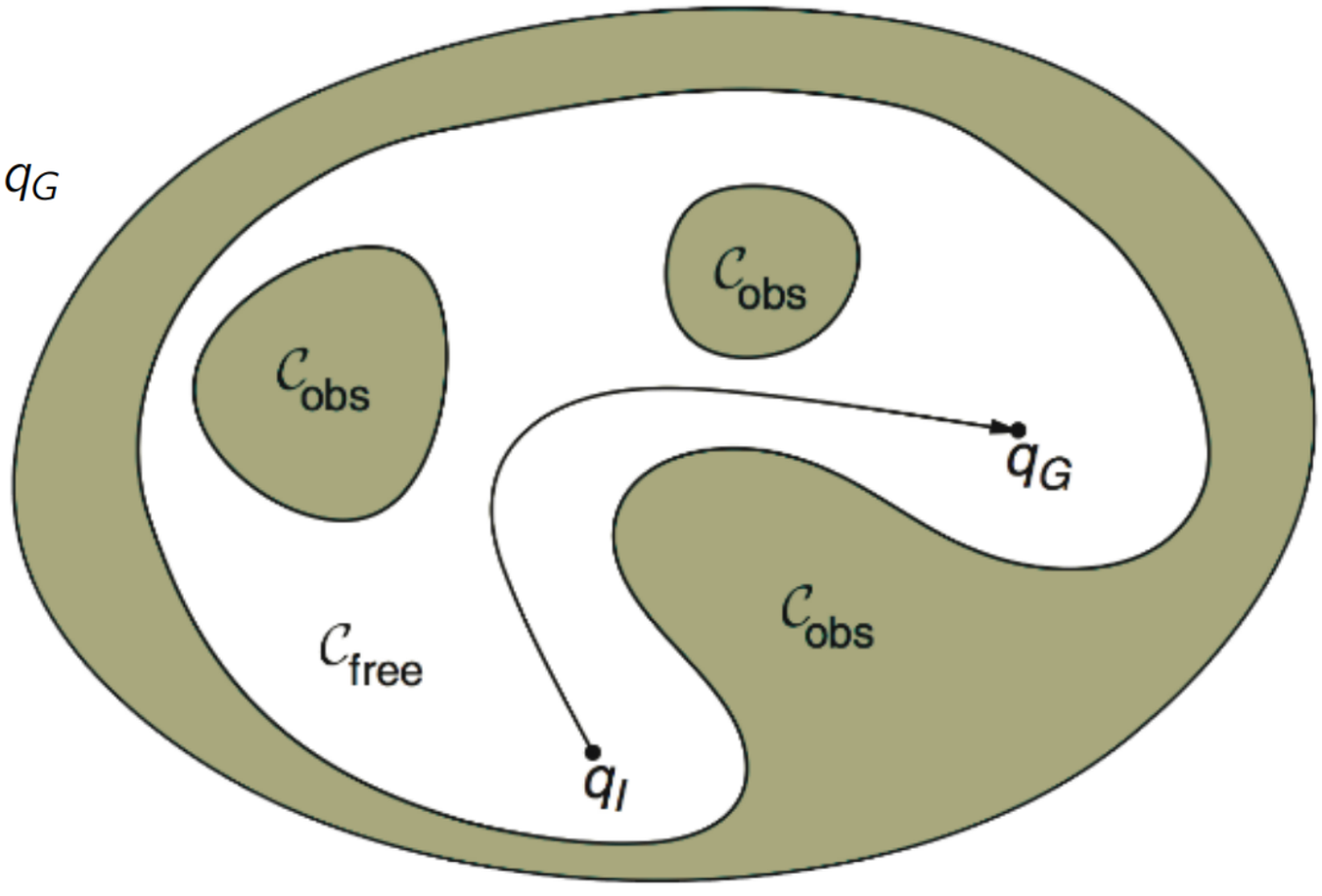
- The *start configuration* is q_I

  The *final (target) configuration* is q_G

- Path planning amounts to find a continuous parametric path:

  $\mathcal{P} : [0,1] \mapsto \mathcal{C}_{free}$

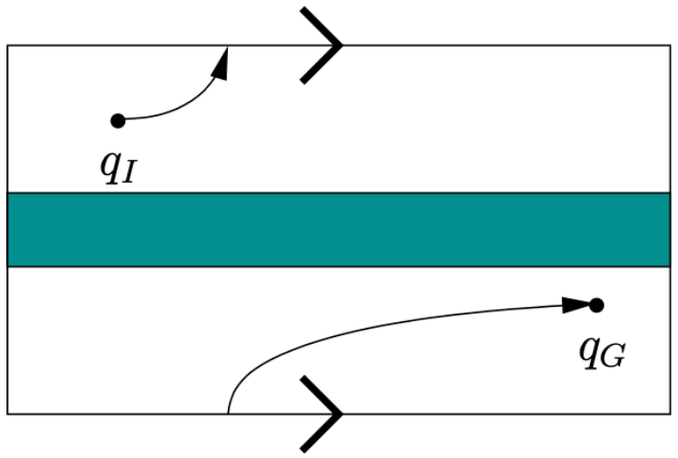
  with $\mathcal{P}(0) = q_I, \quad \mathcal{P}(1) = q_G$

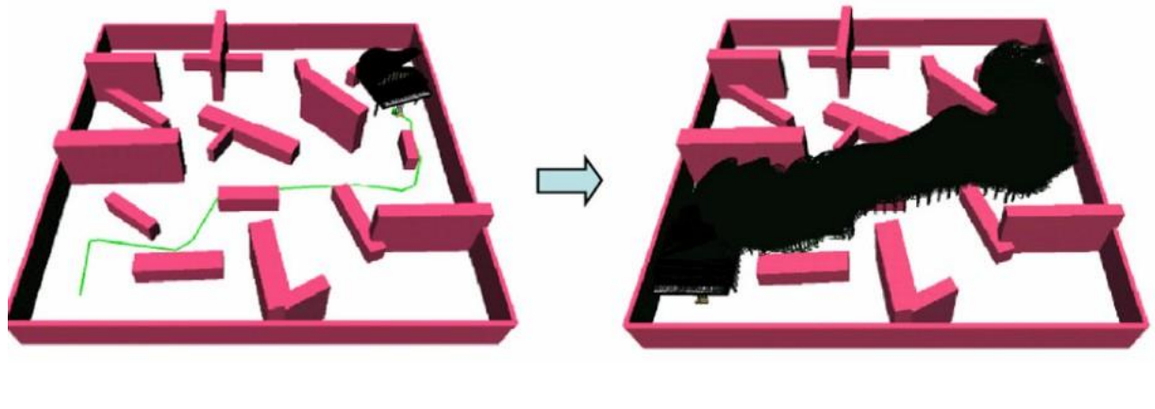


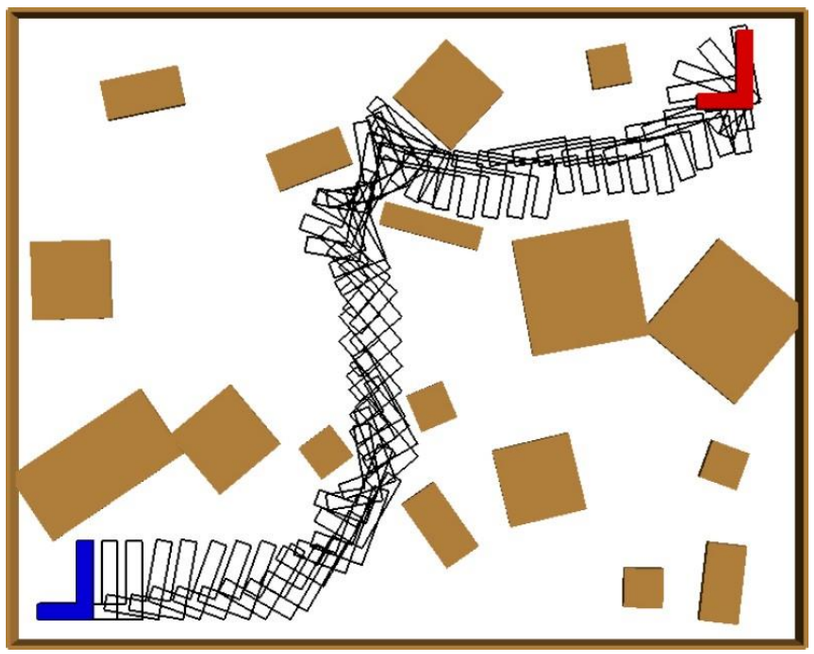
Motion planning: (Basic) Path planning (Piano mover's problem)

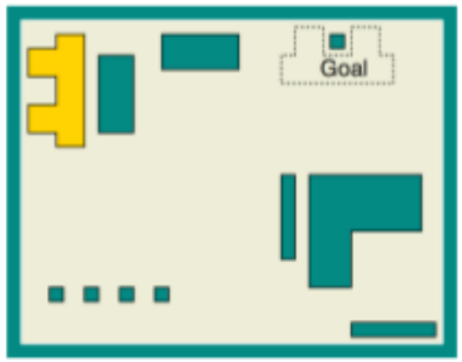
- $\pi(s), s \in [0,1]$ defines a continuous set of transformations / configurations for the robot which are purely **geometric**

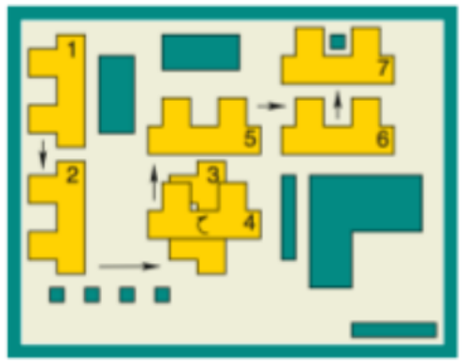
  - $q(s), q(s + ds)$ are neighbor configurations in the **topological** space representing $\mathcal{C}$











- Velocity controls u e.g., v, ω, to actuate the path?
- What if the robot is non-holonomic?

Motion planning: Trajectory planning

Inputs:

- **Start state** of the robot, $\xi(0) = \xi_I = (q_I, \dot{q}_I)$
- **Goal state,** $\xi_G = (q_G, \dot{q}_G)$
- **Description of the robot:** Geometry, kinematics, dynamics
- **Description of the world:** Map, including existing obstacles

Goal:

- Find:
  - **A time T**
  - **A set of controls u to be issued from 0 to T,** $u : [0, T] \rightarrow \mathcal{U}$

    e.g., $(v_0, \omega_0), (v_1, \omega_1), \ldots, (v_T, \omega_T)$
- Such that, based on the equations of motion and robot, and on definition of $\mathcal{C}_{free}$
  - ☑ $\xi(T) = \xi_G$
  - ☑ $q(\xi(t)) \in \mathcal{C}_{free}\ \forall t \in [0, T]$

+ Feedback-based control to ensure correct execution of plan

Path planning: required or desirable objectives

☑ Find a **path** π that moves the robot from initial to goal configurations **never colliding with obstacles** (***admissibility***)

☑ Find a **path that complies with robot motion constraints** (***feasibility***)

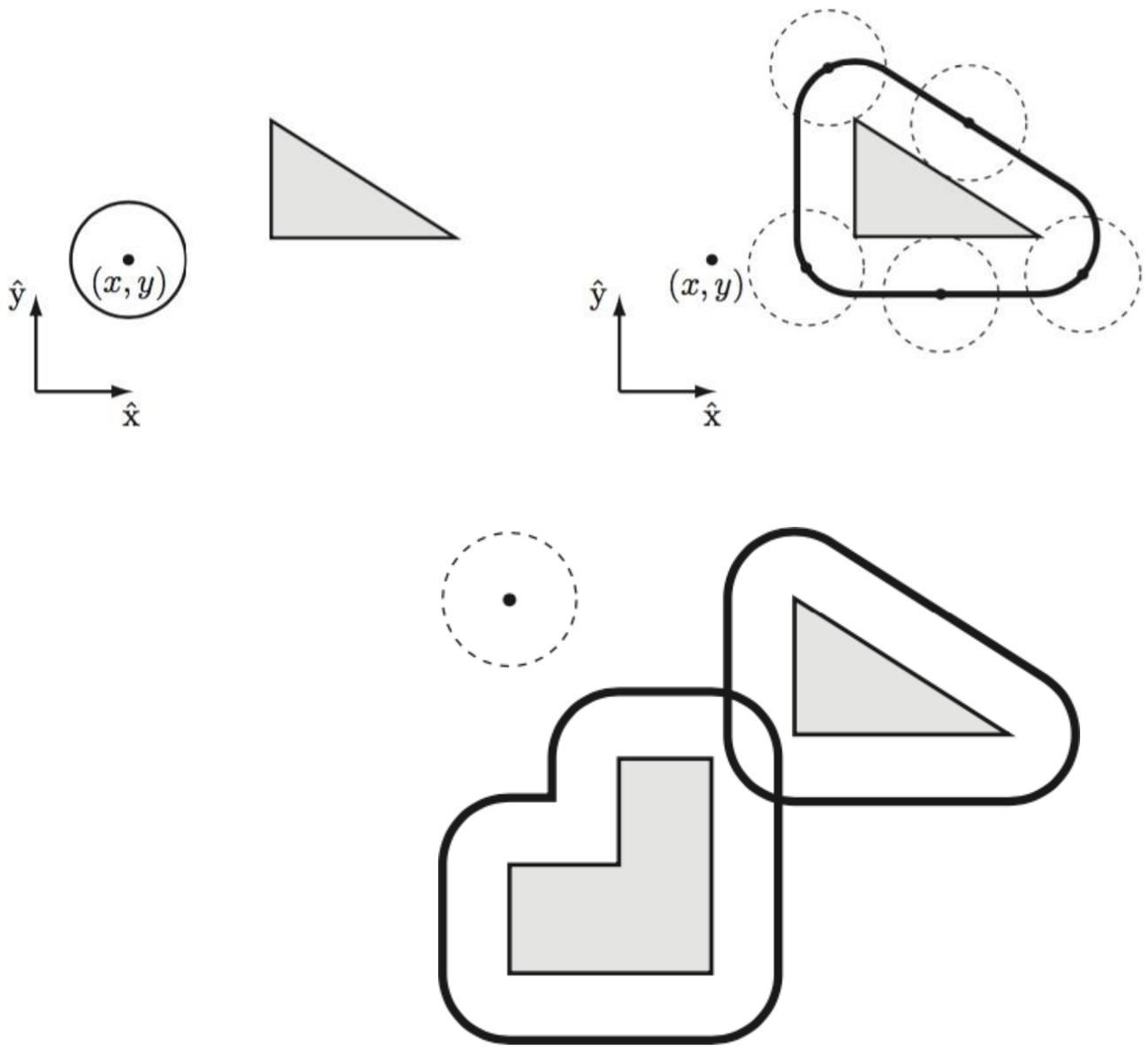
☑ If no feasible + admissible solution exist → Report a *failure* (PSPACE-hard)

☐ ***Optimality:*** If more than one solution exists, select the best

☐ ***Completeness:*** If at least one feasible + admissible path exists, the path planning algorithm is able to *find it.*

Optimality metrics: **Shortest distance**, **Minimum time**, Minimum curvature, Min effort, Max smoothness, Maximal safety, Min/Max ...

Construction of C-free in presence of obstacles

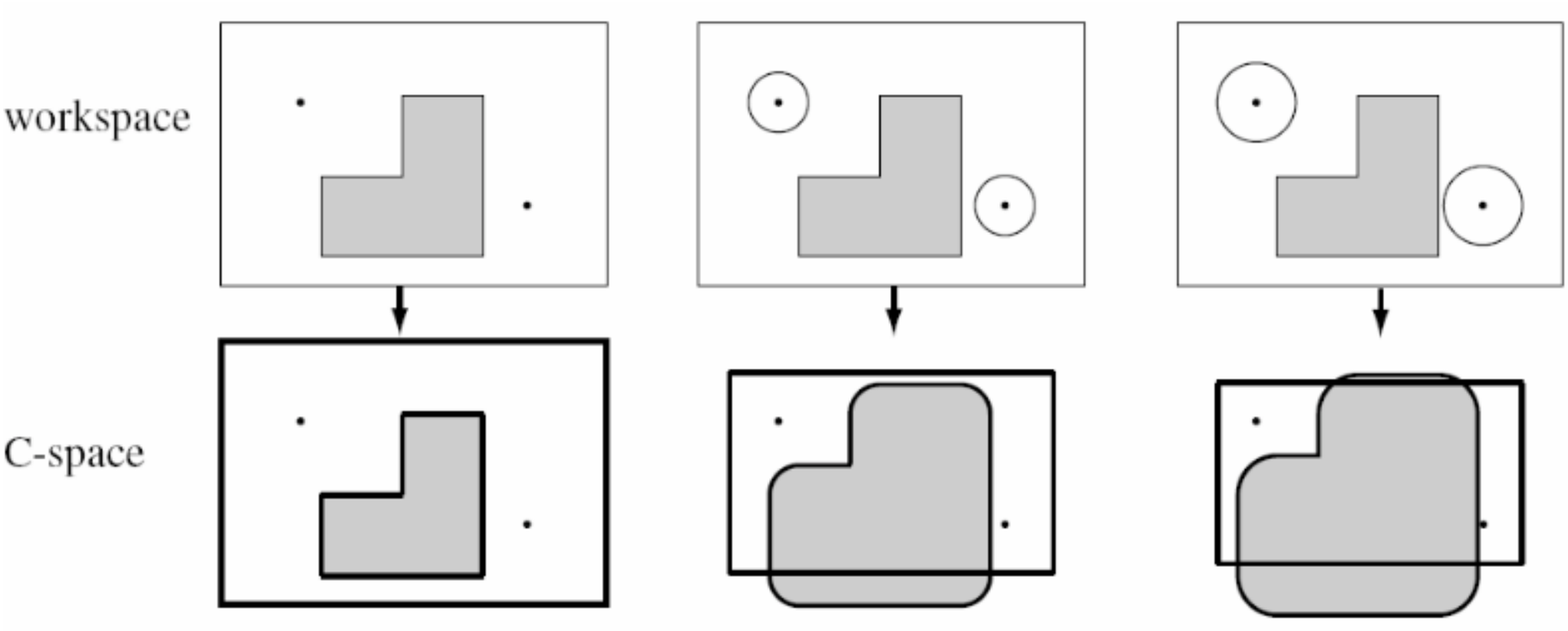


Single *reference point* (single-body robots):

C_{free} the accessible C-space, is obtained w.r.t. the reference point by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius

Construction of C-free in presence of obstacles

Disk robots
with different radius/size



Construction of C-free: polygonal robot, only translation

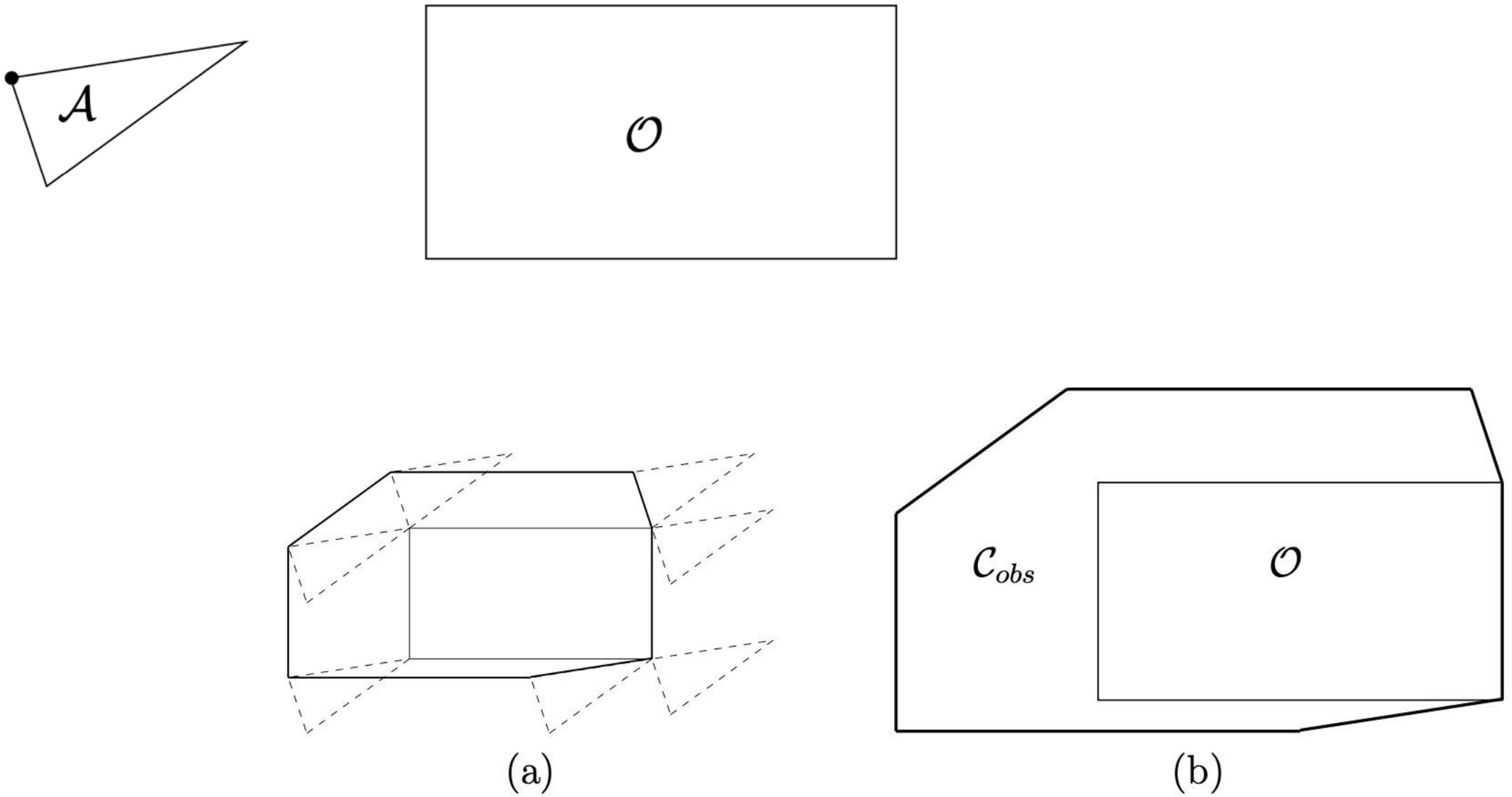
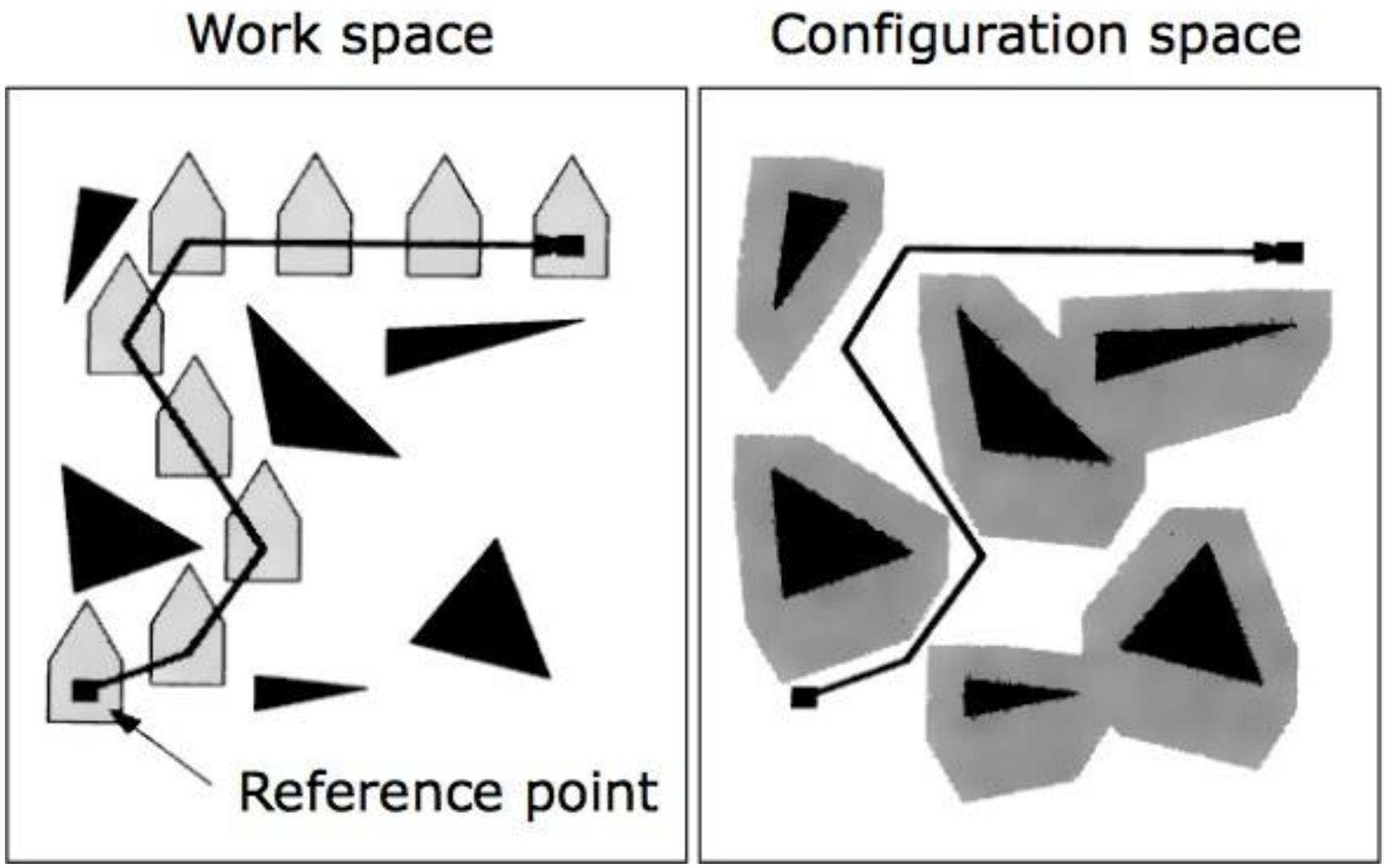


Figure 4.14: (a) Slide the robot around the obstacle while keeping them both in contact. (b) The edges traced out by the origin of $\mathcal{A}$ form $\mathcal{C}_{obs}$.

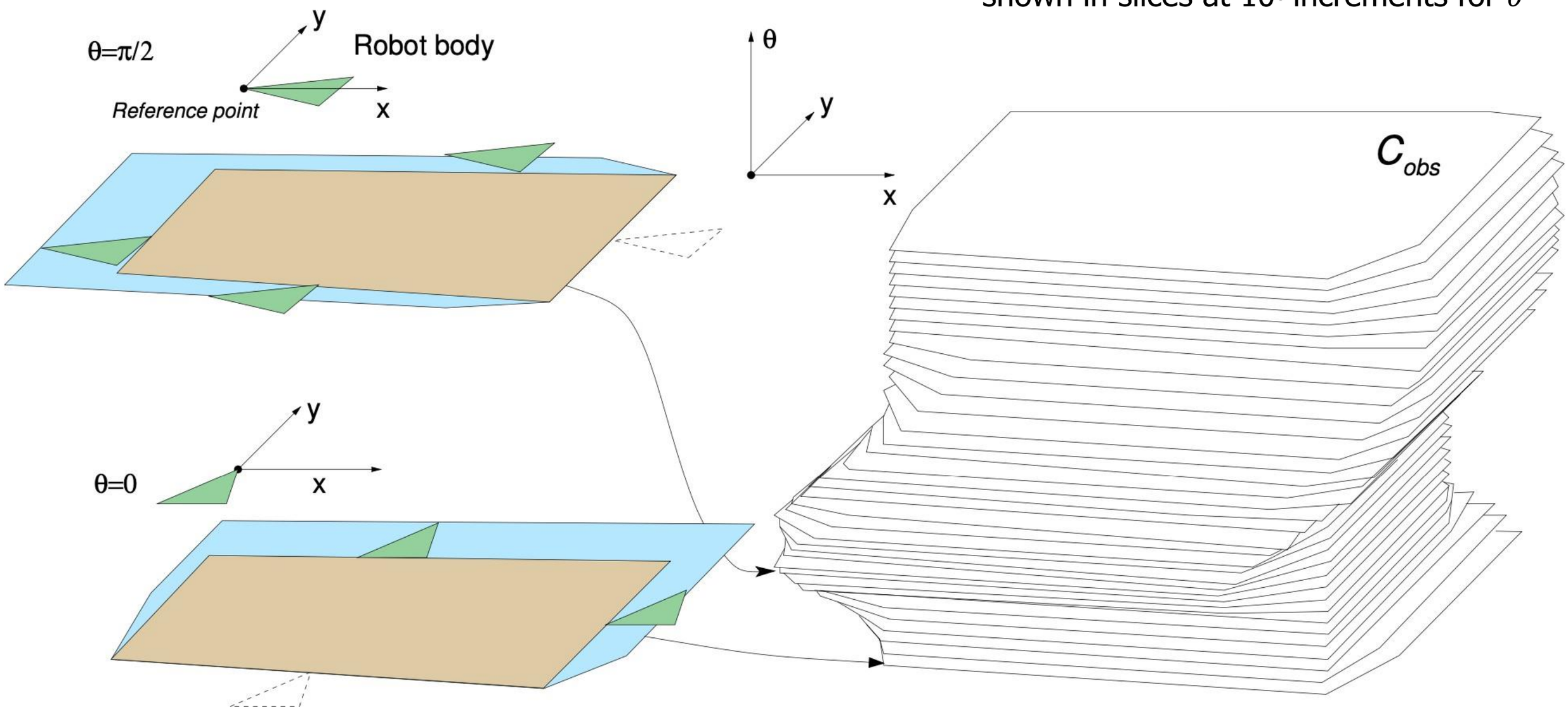
Steven M. LaValle, Planning Algorithms, Cambridge University Press, 844 pp., 2006

Construction of C-free: polygonal robot, only translation



Construction of C-free: polygonal robot, translation + rotation

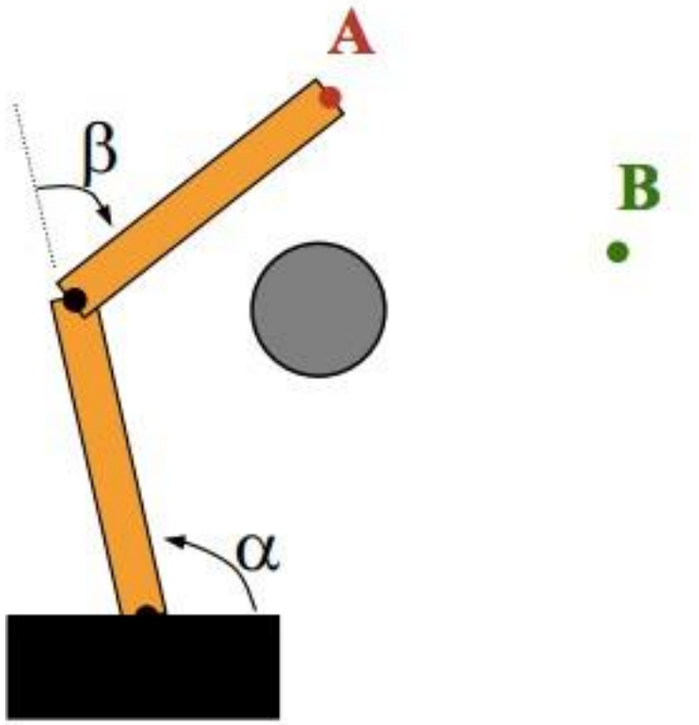
Full 3-dimensional C-space obstacle shown in slices at 10^o increments for θ



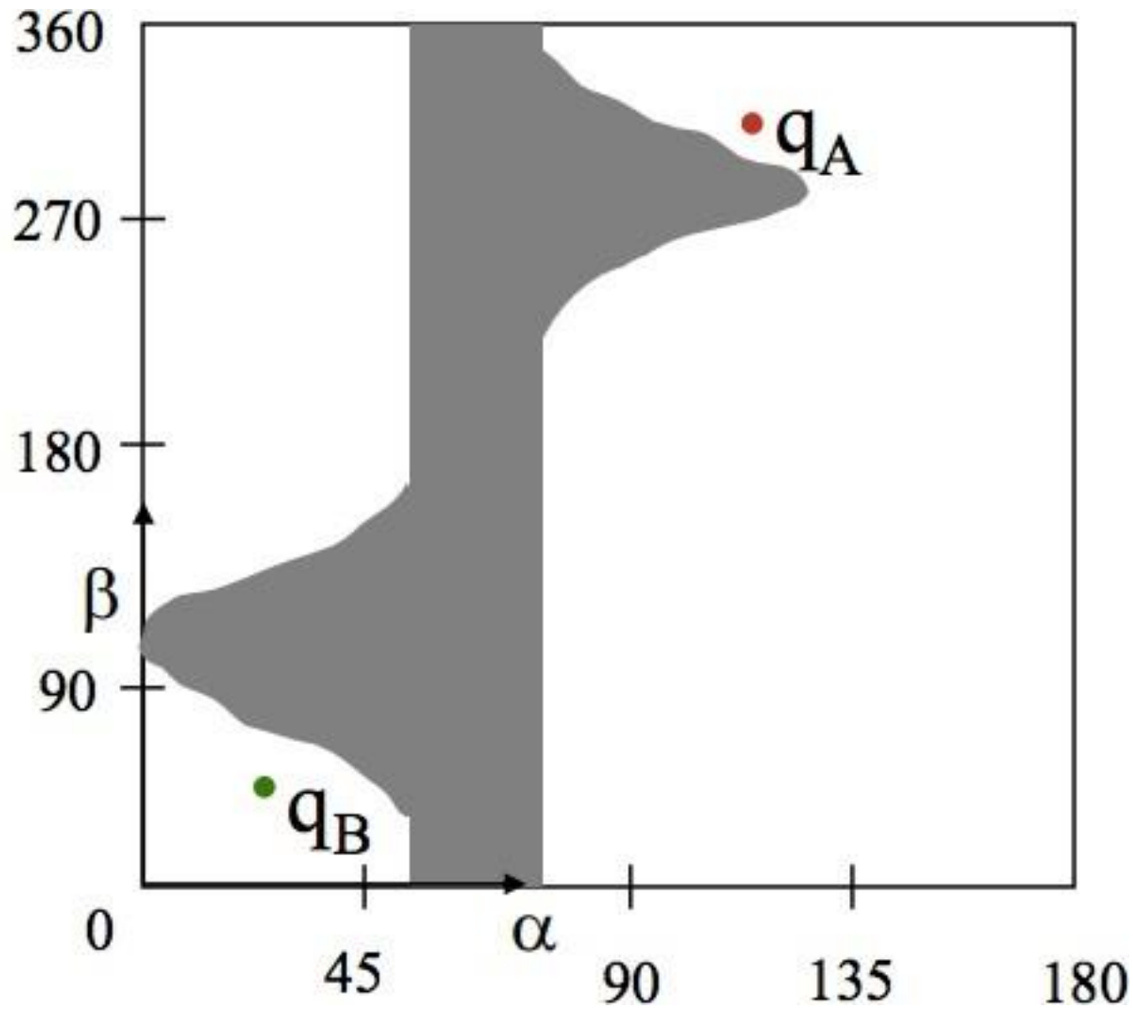
Simple polygonal robot and simple polygonal obstacle → Very complex configuration space!

Construction of C-free: multi-link manipulator

The construction of the free C-space can be very difficult for complex shapes and cluttered environments



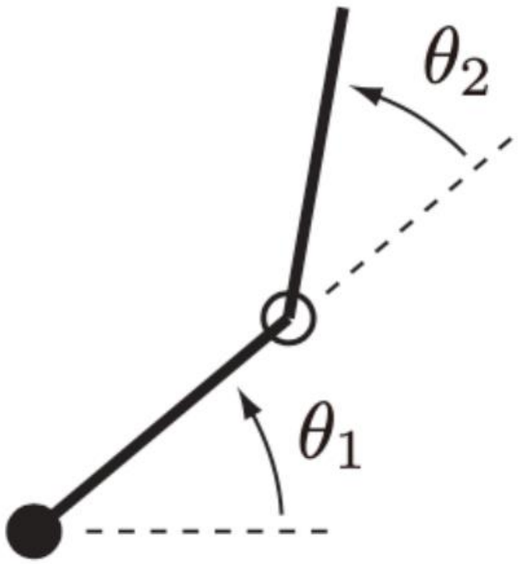
How the two-link robot arm can move from A to B based on the presence of the grey obstacle?

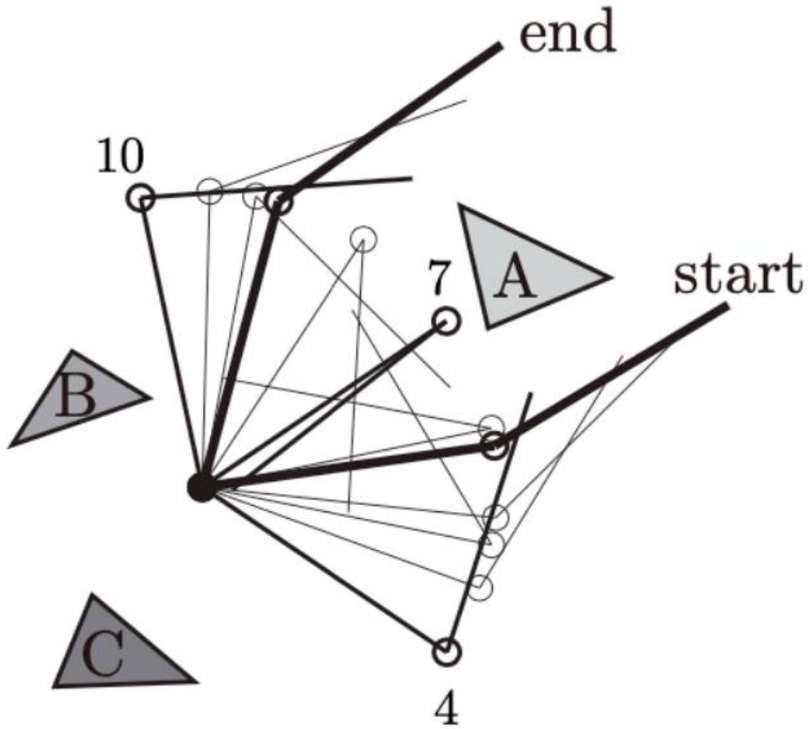


Resulting C-space representation of the obstacle for the robot arm

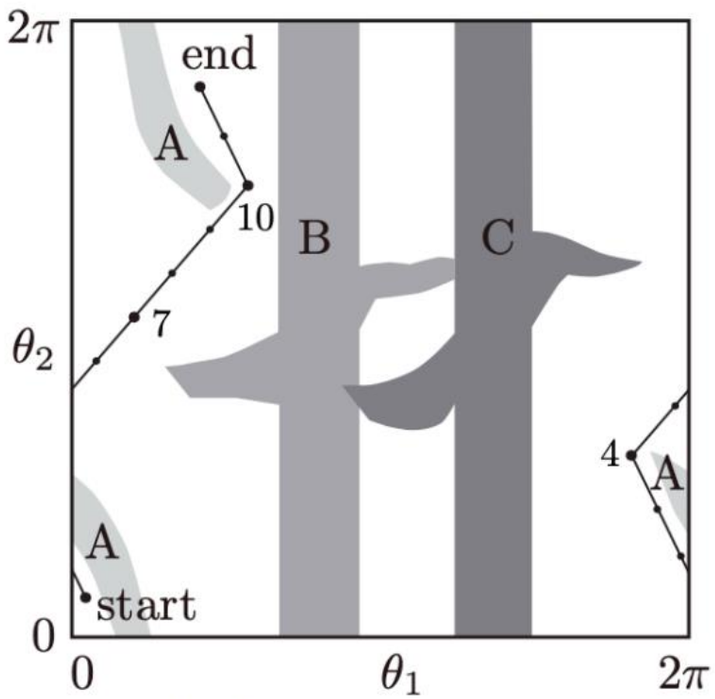
Construction of C-free: multi-link manipulator

Simple(st) arm robot: 2-link planar arm with revolute joints





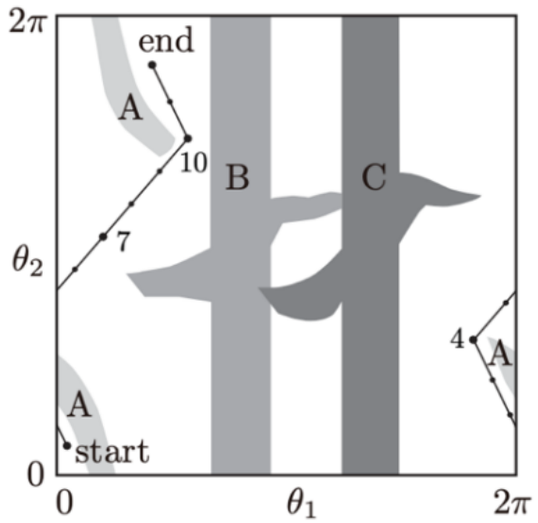
Path from start to end, in workspace

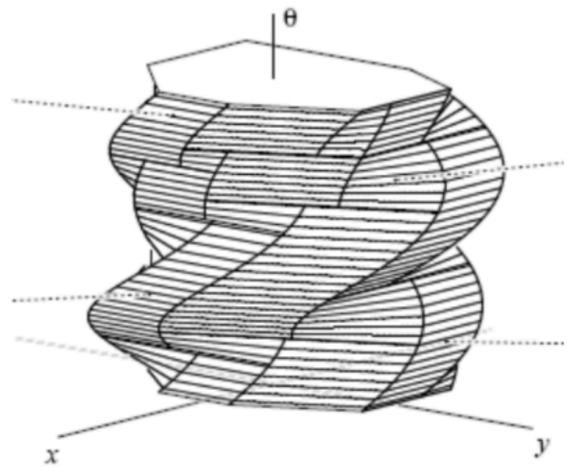


Path from start to end, in configuration space (3 intermediate points are shown)

- Note that the obstacles break $\mathcal{C}_{free}$ into three connected components: for a path to exist, the query pair must lie inside the same connected component

Construction of C-free: impractical





- Deterministic algorithms exist for automatic calculation of configuration space in presence of obstacles, but in the case of general shapes have a complexity which is exponential in the dimension of configuration space

J. Canny, PAMI, 8(2):200–209, 1986

(mathematical)

- Explicit representation of $\mathcal{C}_{free}$ is impractical to compute.

LaValle, Chapter 4

Approximation of C-space

☑ For the purpose of planning, it might be sufficient to be able to efficiently compute for each robot configuration q the (Euclidean) distance $d(q, \beta)$ to a C-obstacle β

$d(q, \beta) > 0$ no contact with the obstacle

$d(q, \beta) = 0$ surface contact with the obstacle

$d(q, \beta) < 0$ penetration into the obstacle

Distance measurement algorithm:

compute $d(q, \beta)$

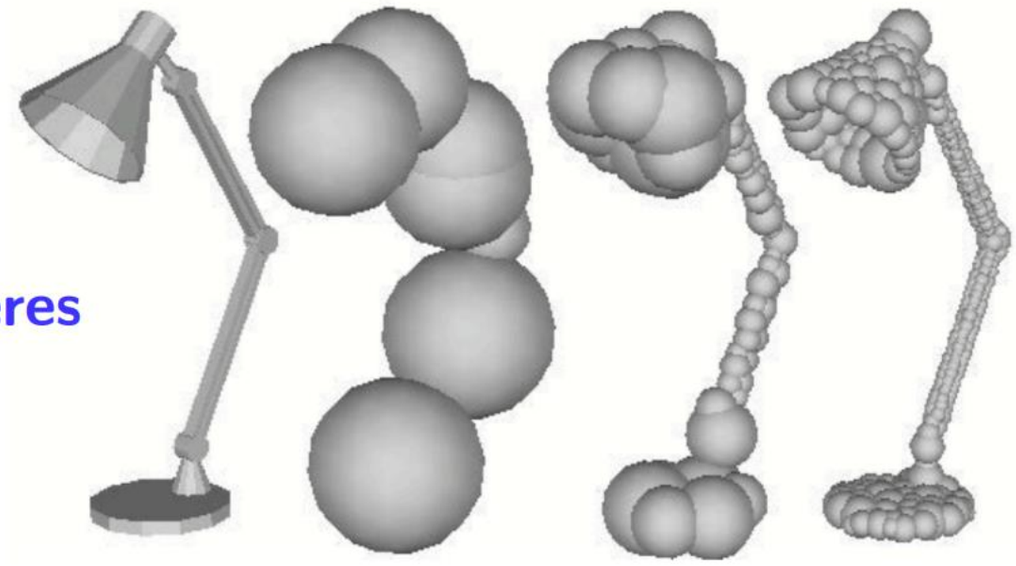
Collision-detection routine

```
Collision_detection(q, 𝒞):
  forall β_i in 𝒞:
    if d(q, β_i) <= 0:
        return True
    else:
        return False
```

Distance measurement algorithms

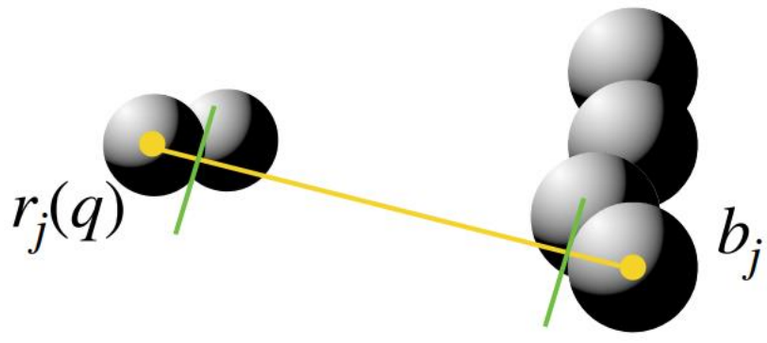
- Gilbert–Johnson–Keerthi (GJK): efficient computation of the distance between two convex bodies, possibly represented by triangular meshes.
  → Any robot or obstacle can be treated as the union of multiple convex bodies

- **Special case** (simple, efficient): approximate robot and obstacles as union of possibly overlapping **spheres**



Given a robot at q represented by k spheres of radius R_i centered at $r_i(q)$, $i = 1, \ldots, k$, and an obstacle $\mathcal{B}$ represented by ℓ spheres of radius B_j centered at b_j, $j = 1, \ldots, \ell$, the distance between the robot and the obstacle can be calculated as

$$d(q, \mathcal{B}) = \min_{i,j} \|r_i(q) - b_j\| - R_i - B_j.$$



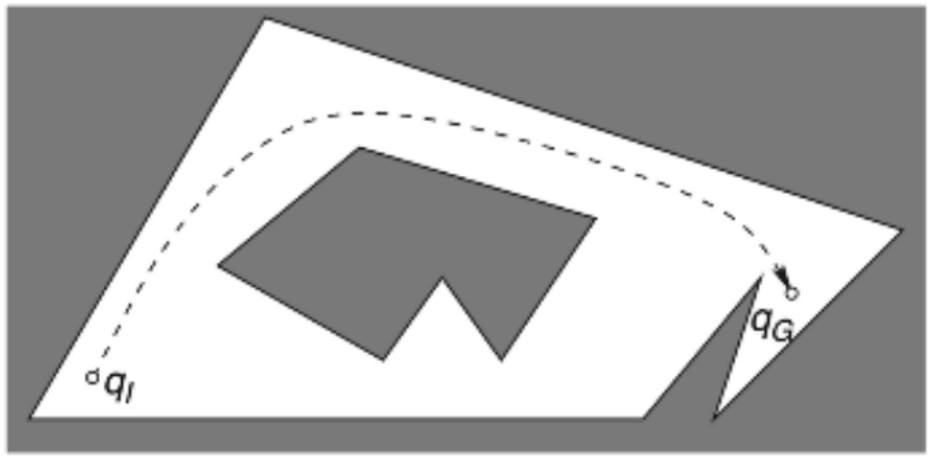
➡ Increasing the number and adapting the size of the spheres increases precision of the approximation, but also increases computation time

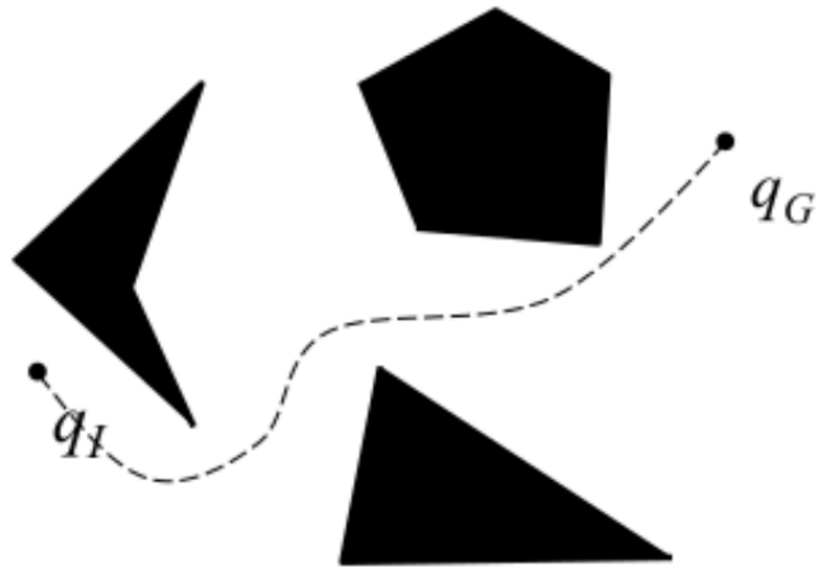
Motion planning: Core Challenges

- Constructing an exact or precise approximation of C-space
- C-space is a potentially highly dimensional, continuous topological space
- Completeness: find a path if it exists, or report a failure
- Optimality vs. the selected metric
- Non-holonomic robots, differential constraints

Let's start with holonomic robots

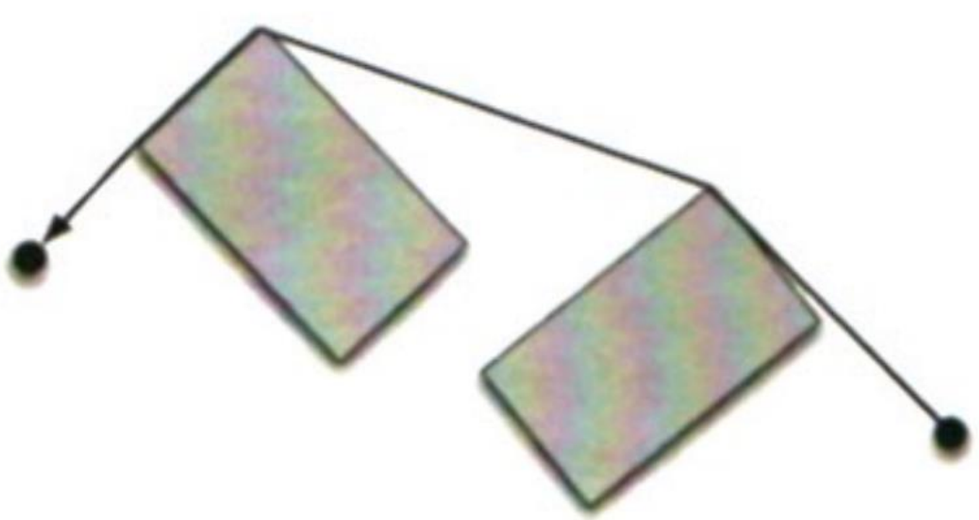
- Let's start, without (much) loss of generality (and performance), to assume to have an omnidirectional (holonomic) point robot moving in a polygonal world: $\mathcal{W} = \mathbb{R}^2$, $\mathcal{C} \subset \mathbb{R}^2$

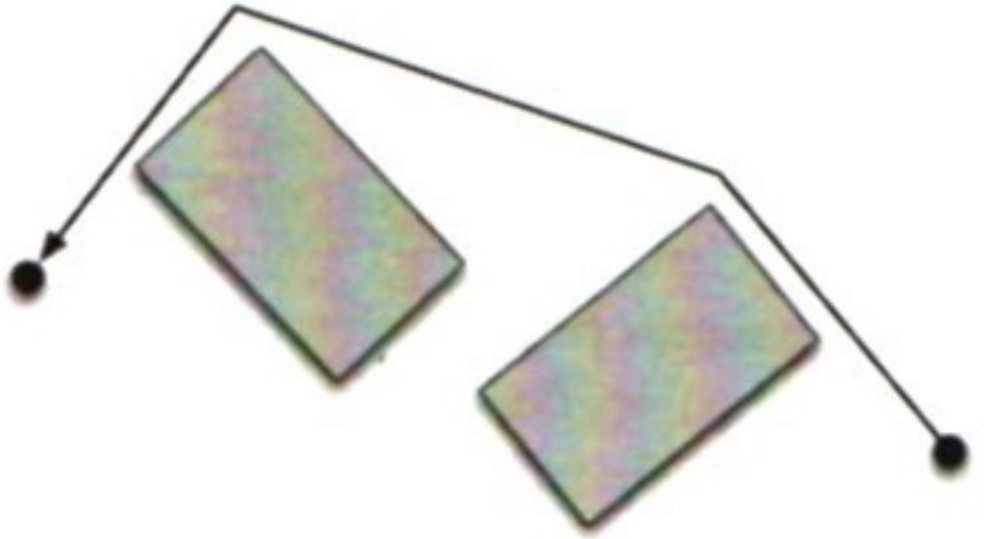




Semi-free space: Touching the obstacles

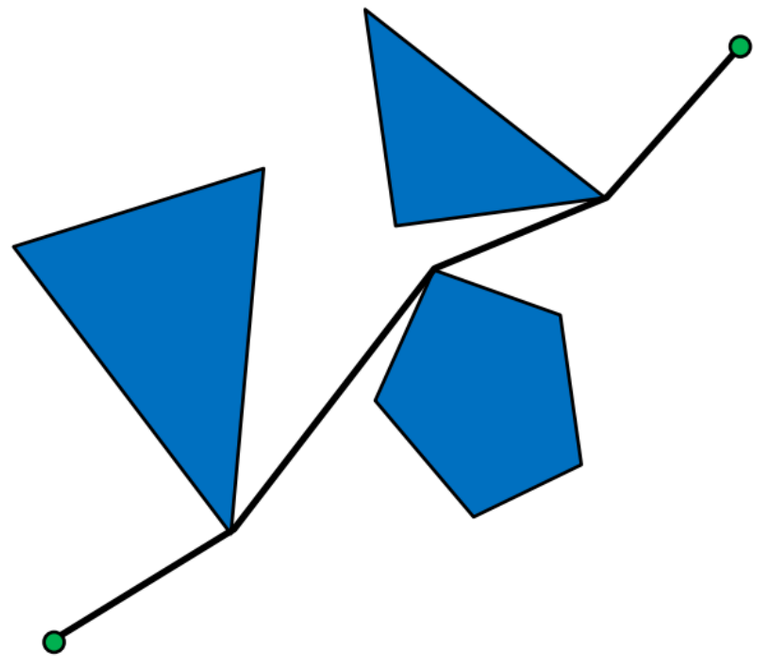
- ***Semi-free paths:*** the obstacles can be touched
- The semi-free space is a *closed set*, which can be (also) useful to prove optimality (the free space is an open set, since the obstacles' frontier does not belong to the set)





Optimal shortest path?

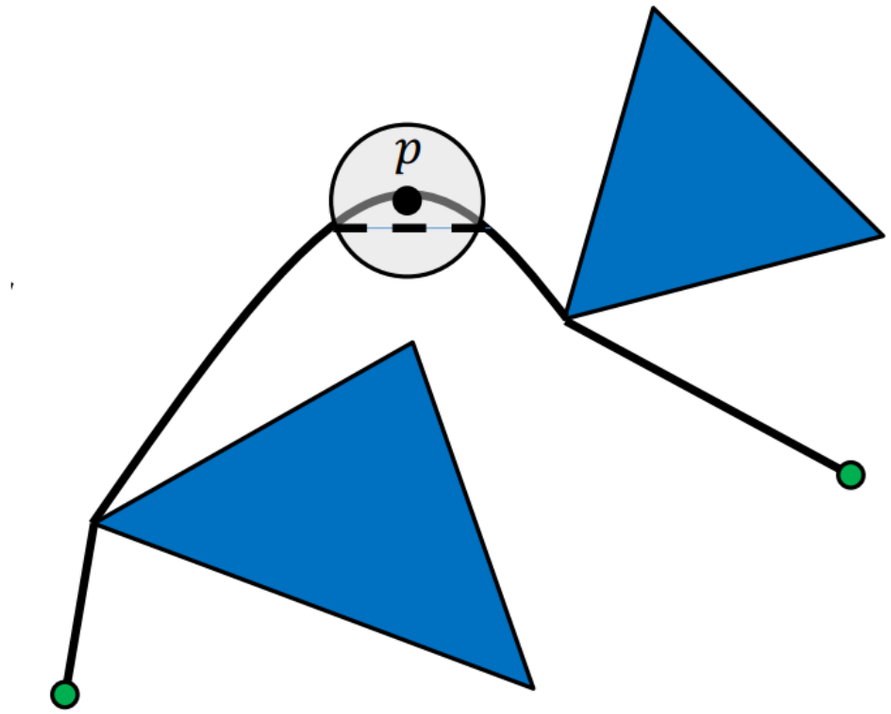
- Polygonal path: sequence of connected straight lines
- Inner vertex of polygonal path: vertex that is not beginning or end



- Theorem: Assuming *polygonal obstacles*, a shortest path is a *polygonal path* whose inner vertices are vertices of obstacles

Optimal shortest path?

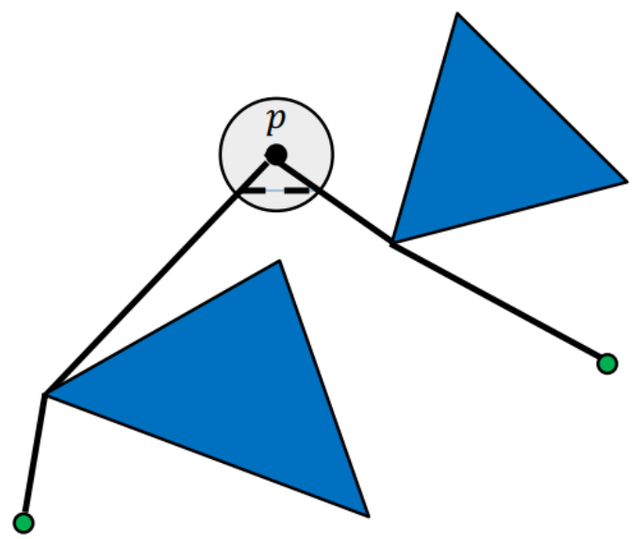
- Suppose for contradiction that shortest path is *not* polygonal
- Obstacles are polygonal ⇒ ∃ point p in interior of free space such that "(shortest) path through p is curved"
- p in free space ⇒ ∃ disc of free space around p
- Path through disc can be shortened by connecting points of entry and exit
- → *Path it's polygonal!* (also true in free space)



Optimal shortest path?

✓ Path is polygonal

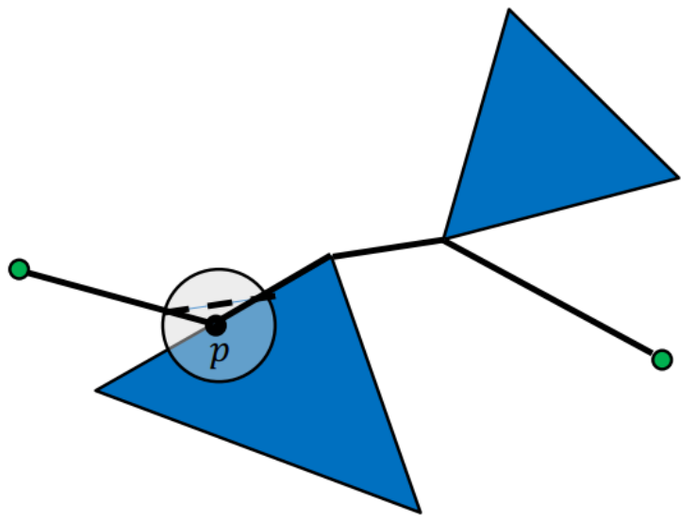
- Vertex cannot lie in interior of free space, otherwise we can do the same trick and shorten the path (that would not then be the shortest)



- Vertex cannot lie on an edge, otherwise we can do the same trick

✓ Inner vertices are vertices of obstacles

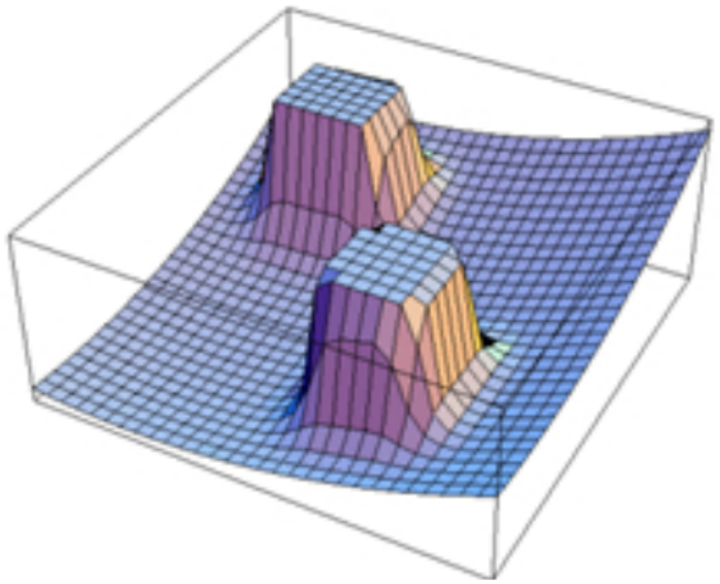
■

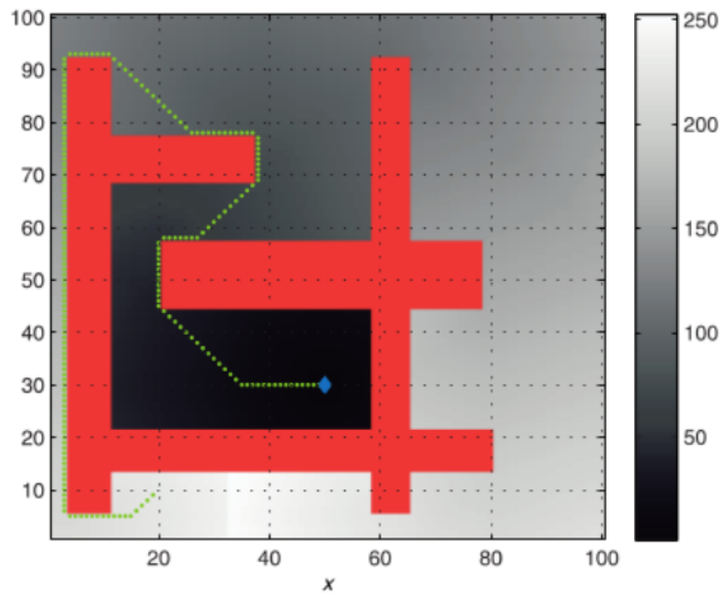


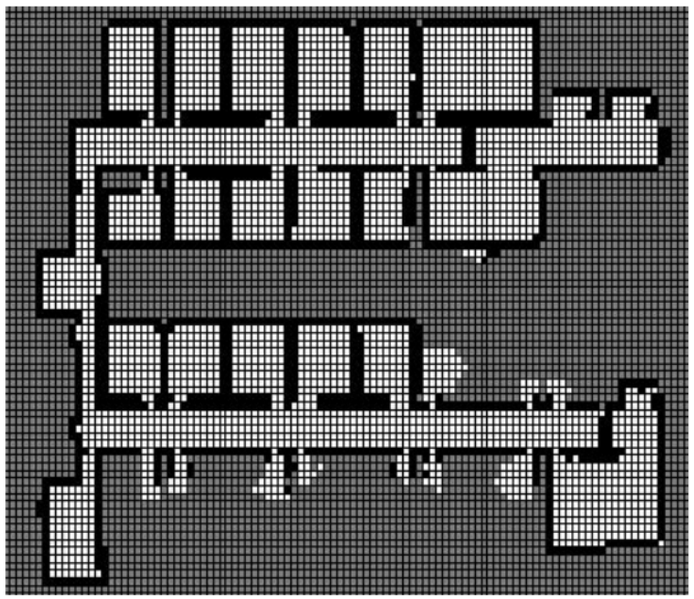
Representing the C-Space

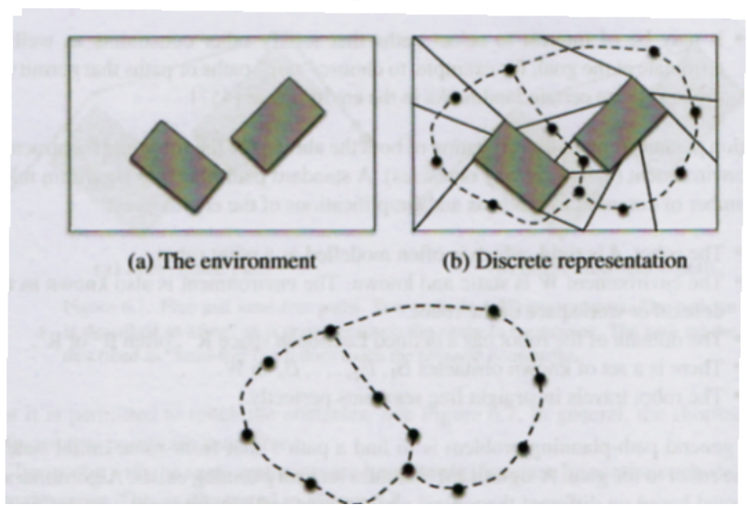
- The admissible search space can be treated as:
  - Continuous space: Challenging to model and demanding from a computational point of view (example approach: *Potential fields*)
  - Discrete space: Most common approach, usually based on deriving a (computationally tractable) graph representation of the environment → Combinatorial problem that can be attacked using a number of existing techniques (Dijkstra, A*, Dynamic programming)

  ...

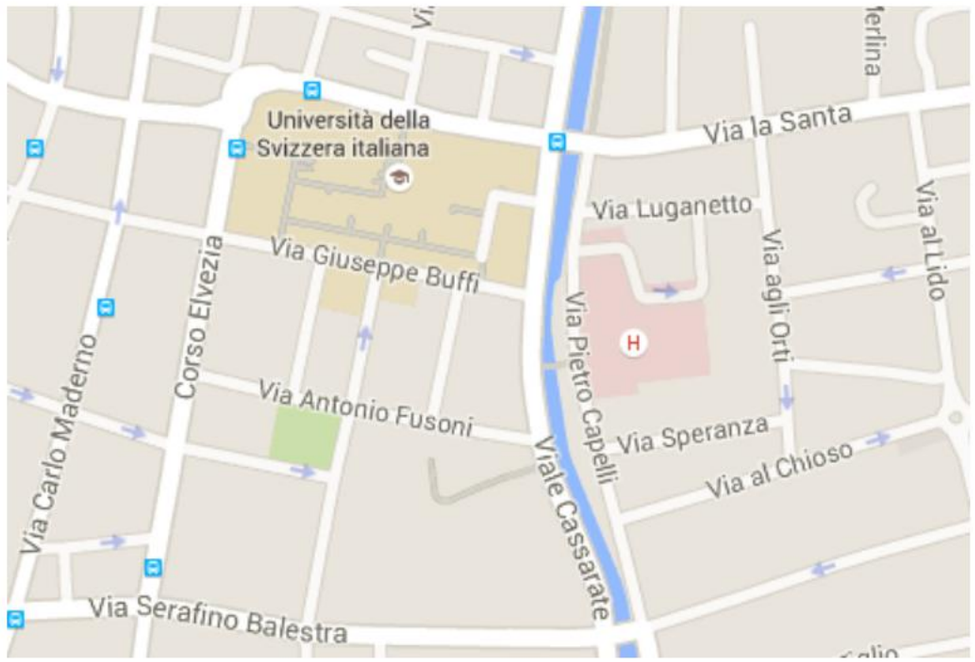


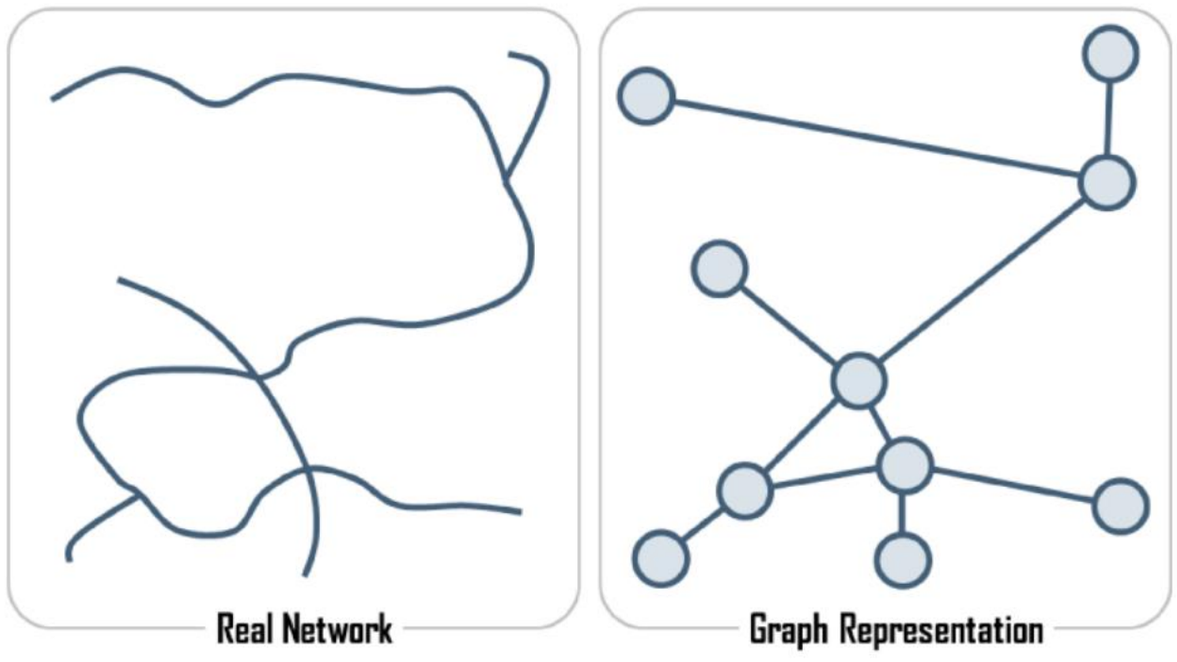






Road maps





A road map is a graph in $\mathcal{C}_{free}$ in which each vertex is a configuration in $\mathcal{C}_{free}$ and each edge is a collision-free path through $\mathcal{C}_{free}$

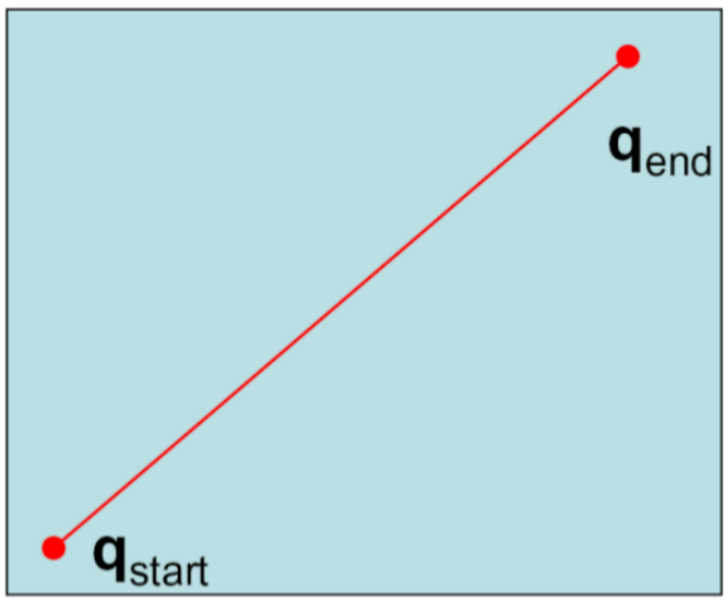
- Instead of defining the problem and computing the paths specifically for the start and goal configurations, it's more flexible to define a general and compact **road map**, which can be used to plan the path between two arbitrary points
- A (small) graph is pre-computed such that staying "on the roads" guarantees avoiding obstacles
- The "roads" represent effective navigable space

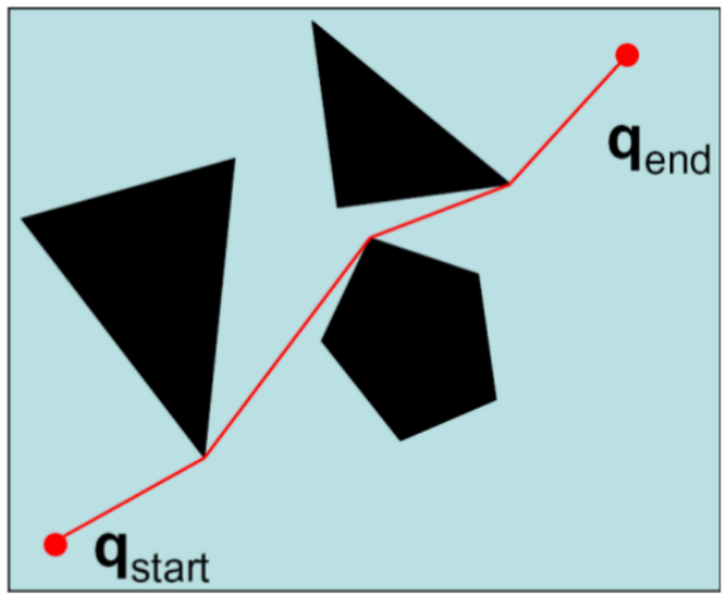
Visibility graphs

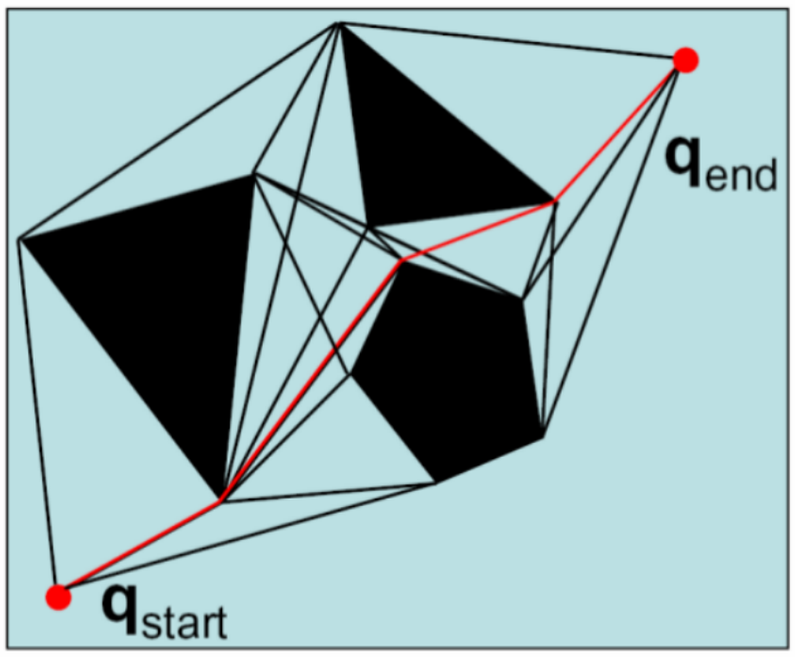
Given polygonal obstacles, a road map can be defined as visibility graph $\mathscr{V}$=(V, E):

- V = set of vertices of the polygons (in $\mathscr{C}_{\text{semi-free}}$) ∪ $\{\boldsymbol{q}_{\text{start}}, \boldsymbol{q}_{\text{end}}\}$
- E = set of unblocked (i.e., visible) line segments between the vertices in V

The previous theorem guarantees that the shortest path between $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{end}}$ is a polygonal line connecting start and goal configuration through the vertices of the polygonal obstacles: it corresponds to the **shortest path in the visibility graph**







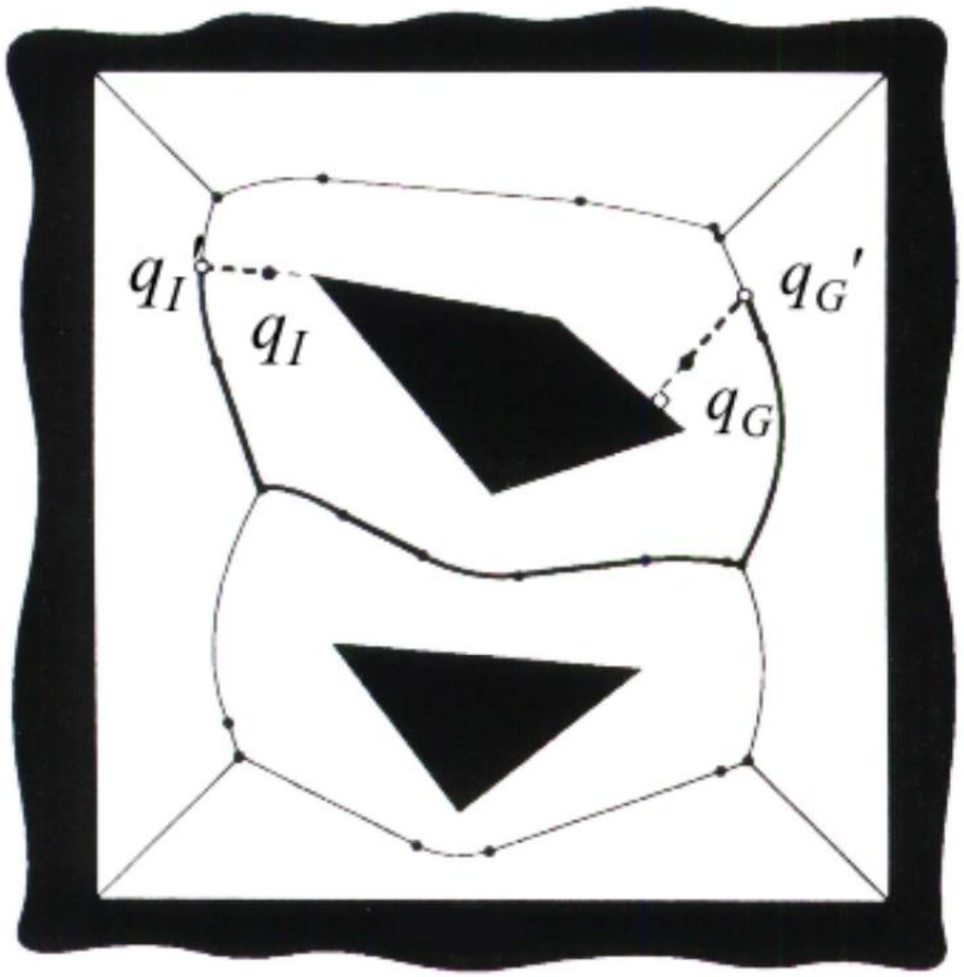
Shortcoming of visibility graphs

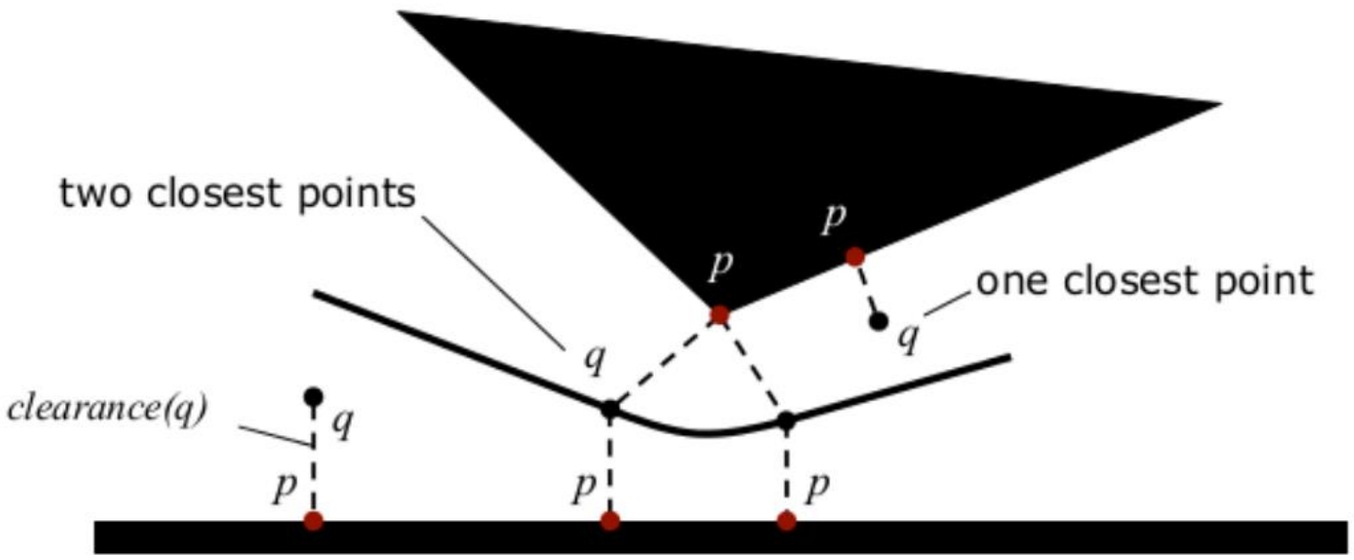
- Guarantee of shortest path but tries to stay as close as possible to obstacles, which is not precisely what we want in practice $\rightarrow$
- Any execution error will lead to a collision
- Complicated in more than 2 dimensions
- Finding a safe path (in practice) is maybe more important than strict optimality . . . $\rightarrow$

Generalized Voronoi diagrams

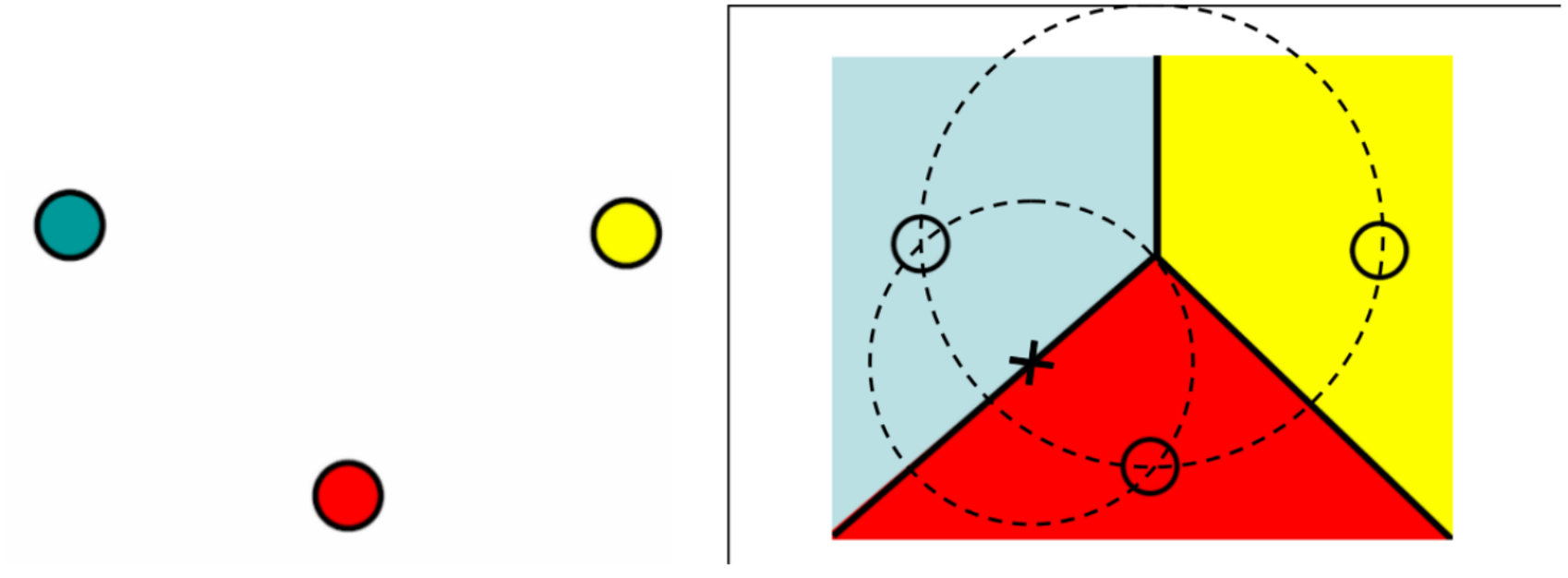
The locus of points that are equidistant from the closest two or more obstacle boundaries (in C_{obs}), including the workspace boundaries. In other words, the set of points q whose cardinality of the set of boundary points (in C_{obs}) with the same distance to q is greater than 1

The region with the same maximal clearance from all nearest obstacles



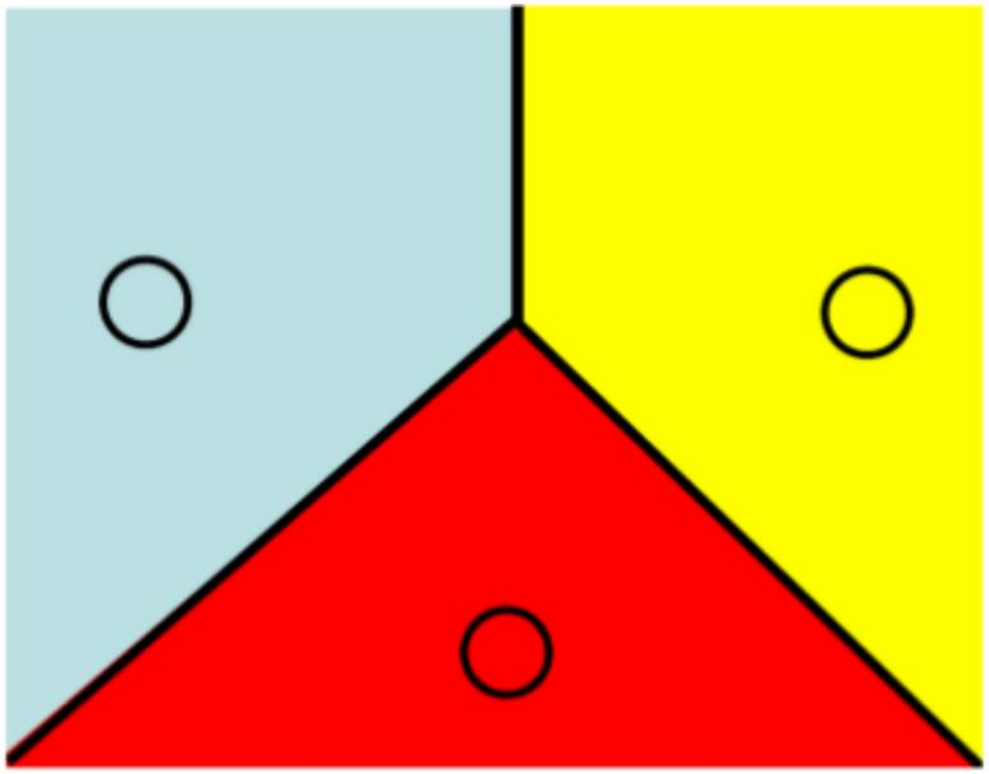


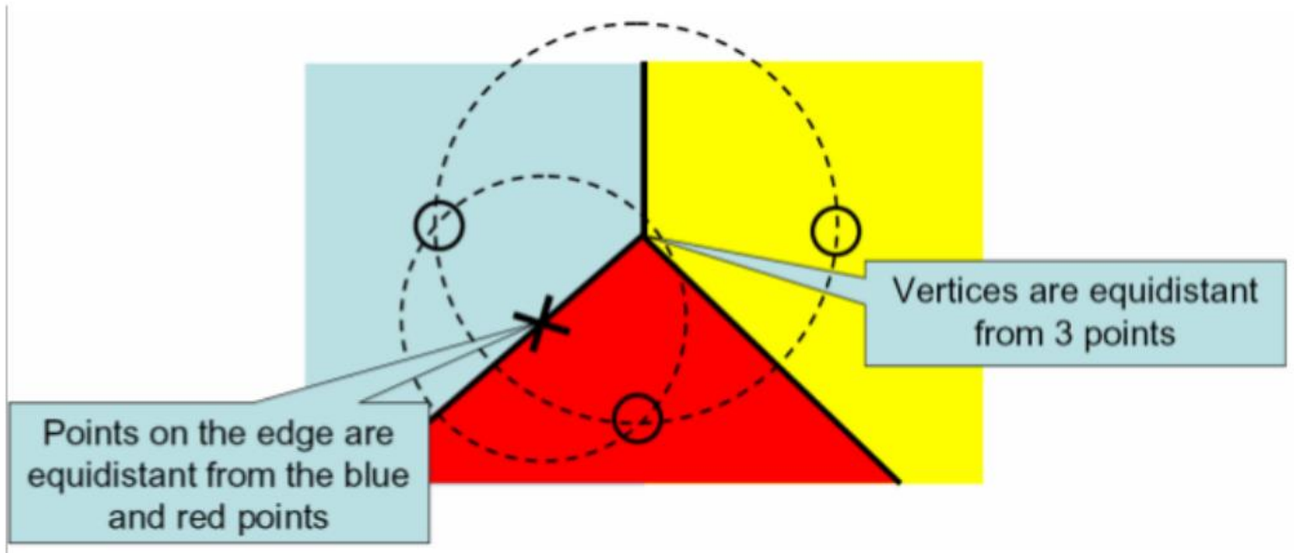
Voronoi cells



- A set of data points, the *generators*, is given
- Each generator point defines a ***Voronoi cell*** that consists of every other point whose "distance" to the generator is less than or equal to the its distance to any other generator point (distance is well defined in Euclidean spaces, but it can be generalized)
- Each cell is obtained from the intersection of half-spaces $\rightarrow$ *Convex polygon*
- ***Voronoi diagram:*** line segments that correspond to all the points in the plane that are equidistant to the two nearest generator points
- ***Voronoi vertices:*** the points equidistant to three (or more) generators

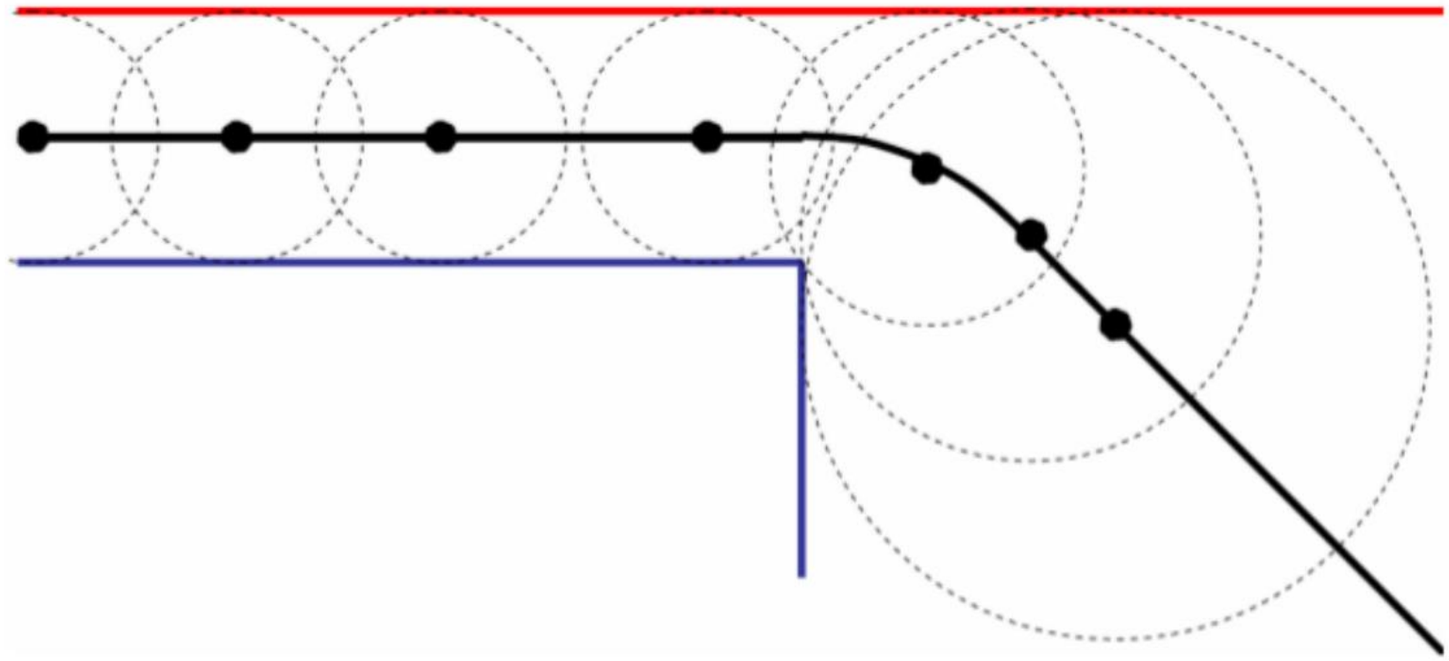
Voronoi diagram





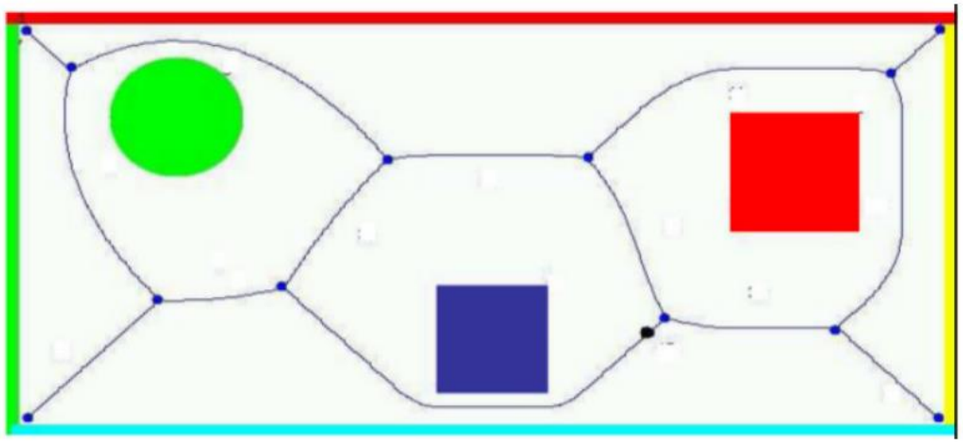
- Voronoi diagram: The set of line segments separating the regions corresponding to different colors
- Line segment: points equidistant from 2 data points
- Vertices: points equidistant from > 2 data points
- Complexity (in the plane): $\mathcal{O}(N \log N)$ in time, $\mathcal{O}(N)$ in space

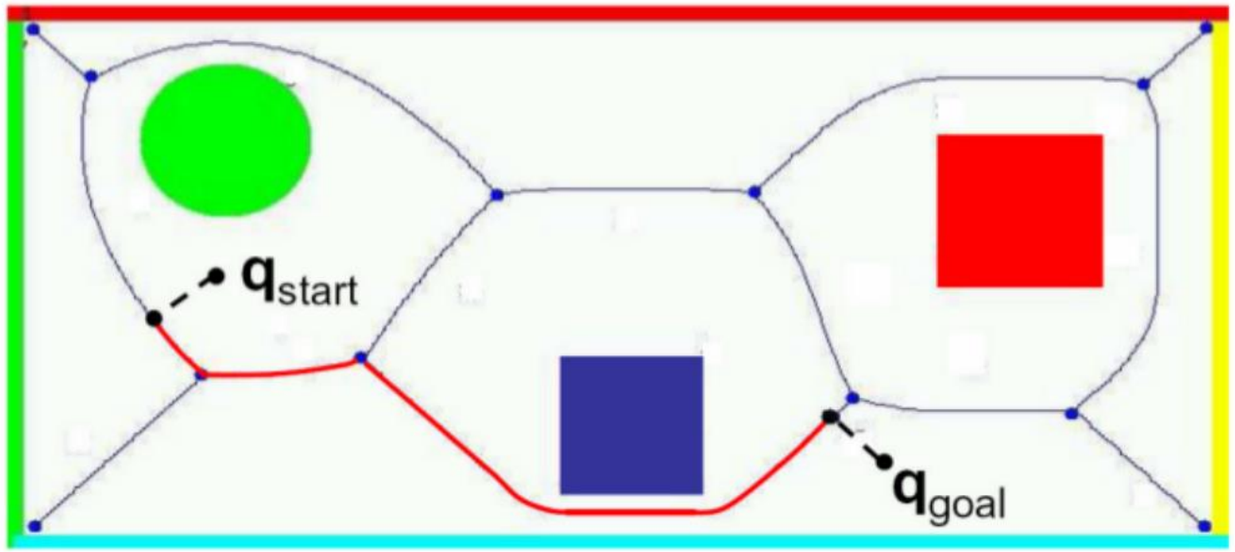
Voronoi diagrams for cluttered environments



- Edges are combinations of straight line segments and segments of quadratic curves
- Straight edges: Points equidistant from 2 lines
- Curved edges: Points equidistant from one corner and one line
- At any point on the Voronoi diagram, the distance to nearby obstacles cannot be increased by any (differential) motion local to the diagram

Planning in Voronoi diagrams





- Find the closest points on the Voronoi skeleton to the desired start and goal points
- Compute the shortest path on the Voronoi graph

Pros and cons of Voronoi planning

- Difficult to compute in higher dimensions or non polygonal worlds
- . . . But approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic (“stay away from obstacles”) → Can lead to paths that are too much conservative
- . . . But for an uncertain robot staying away from the obstacles is a good idea
- Unnatural/counterproductive attraction to open space (see figure)
- Can be unstable: Small changes in obstacle configuration can lead to large changes in the diagram

Pros and cons of Voronoi planning

- **For robots with short-range sensors**: the path-planning algorithm maximizes the distance between the robot and objects in the environment, therefore navigating on a Voronoi path might be problematic, since the robot might not be able to use sensing to detect the objects around it and localize itself, being always too distant from the obstacles to sense them.
- **For robots with long-range sensors**: the Voronoi diagram method has over most other obstacle avoidance techniques the advantage of *executability*. In fact, following the Voronoi path results from maximizing the distance while maintaining equidistant from the surrounding objects, which can be done relatively easily with a good range finder. In this way, the robot can naturally stay on the Voronoi edges, mitigating, for instance, odometry inaccuracy for localization and path-following.