16-311-Q Introduction to Robotics Fall'17

Lecture 26: Path Planning 1

Instructor:
Gianni A. Di Caro

PLANNING

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

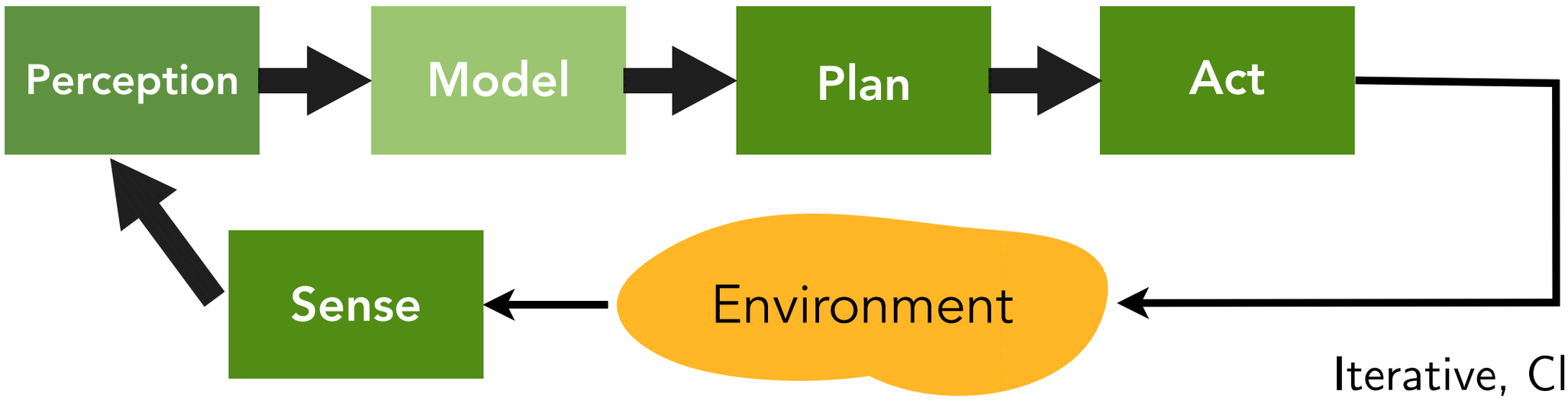
- ***Planning:*** Sequence of actions to achieve a given goal (usually without timing specifications), or towards achieving a given goal by achieving a set of sub-goals
- ***Scheduling:*** Planning + Timing (usually without space considerations)
- ***Task planning:*** Sequence of actions that accomplish 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!

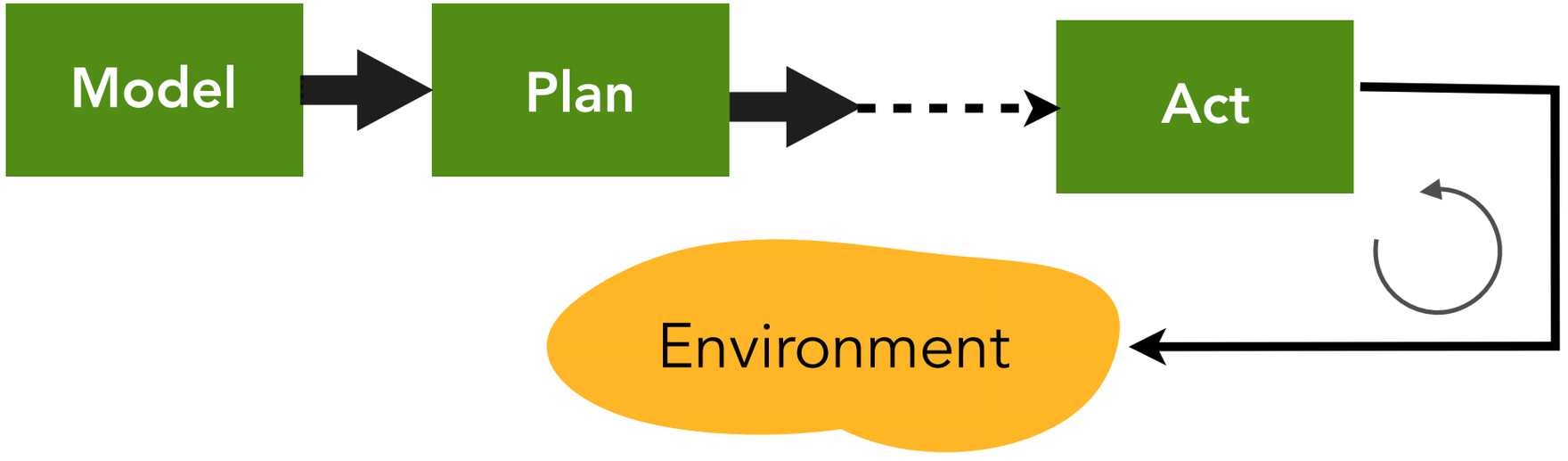
DELIBERATIVE ROBOT CONTROL ARCHITECTURE

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

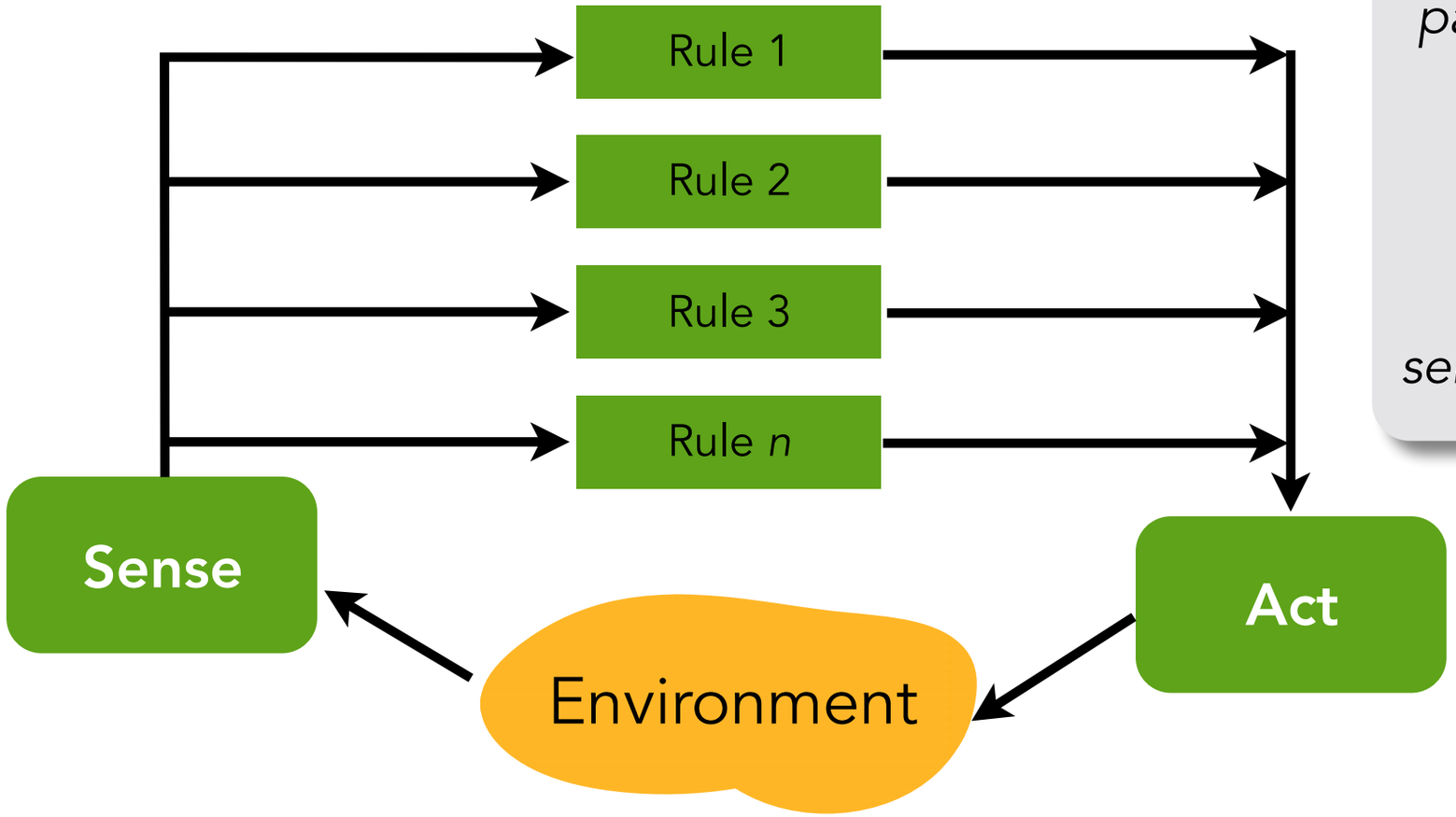
DELIBERATIVE DECISION-MAKING (IN FICTION)

Eagle eye (2008)

An "eagle eye" AI knowing everything and able to make very accurate plans on the spot. It requires accurate models of everything/everybody in the environment to issue reliable predictions about the (hectically changing) world and issue and revise plans accordingly

REACTIVE CONTROL ARCHITECTURES: *DON'T THINK, REACT!*

Sense-Act Transfer rules
(Behaviors)

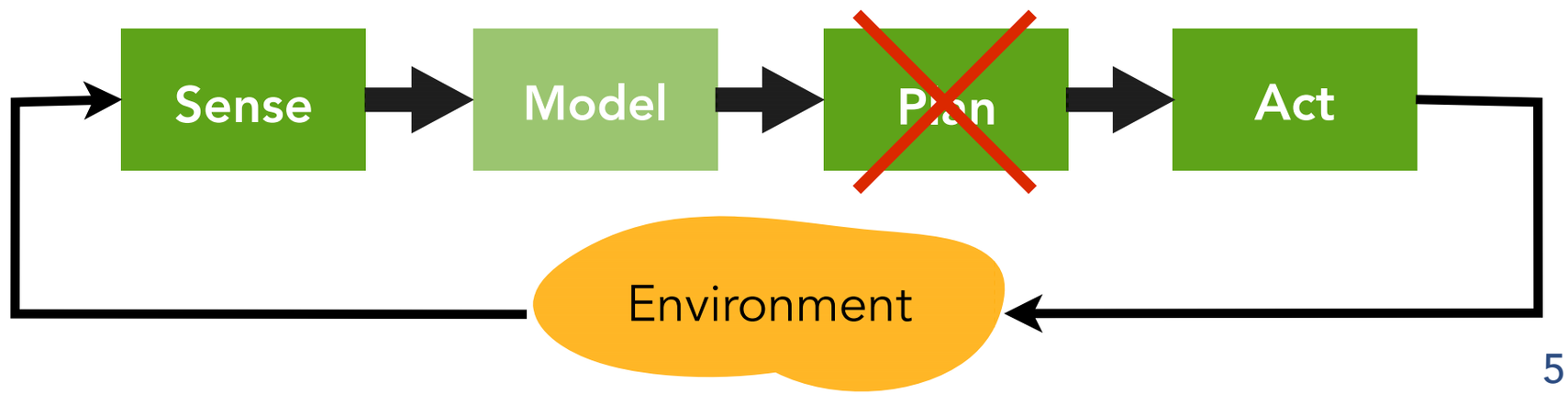


Ethological view (***Behavior***):
Direct mapping of sensory inputs to a pattern of motor actions that are then used to achieve a task

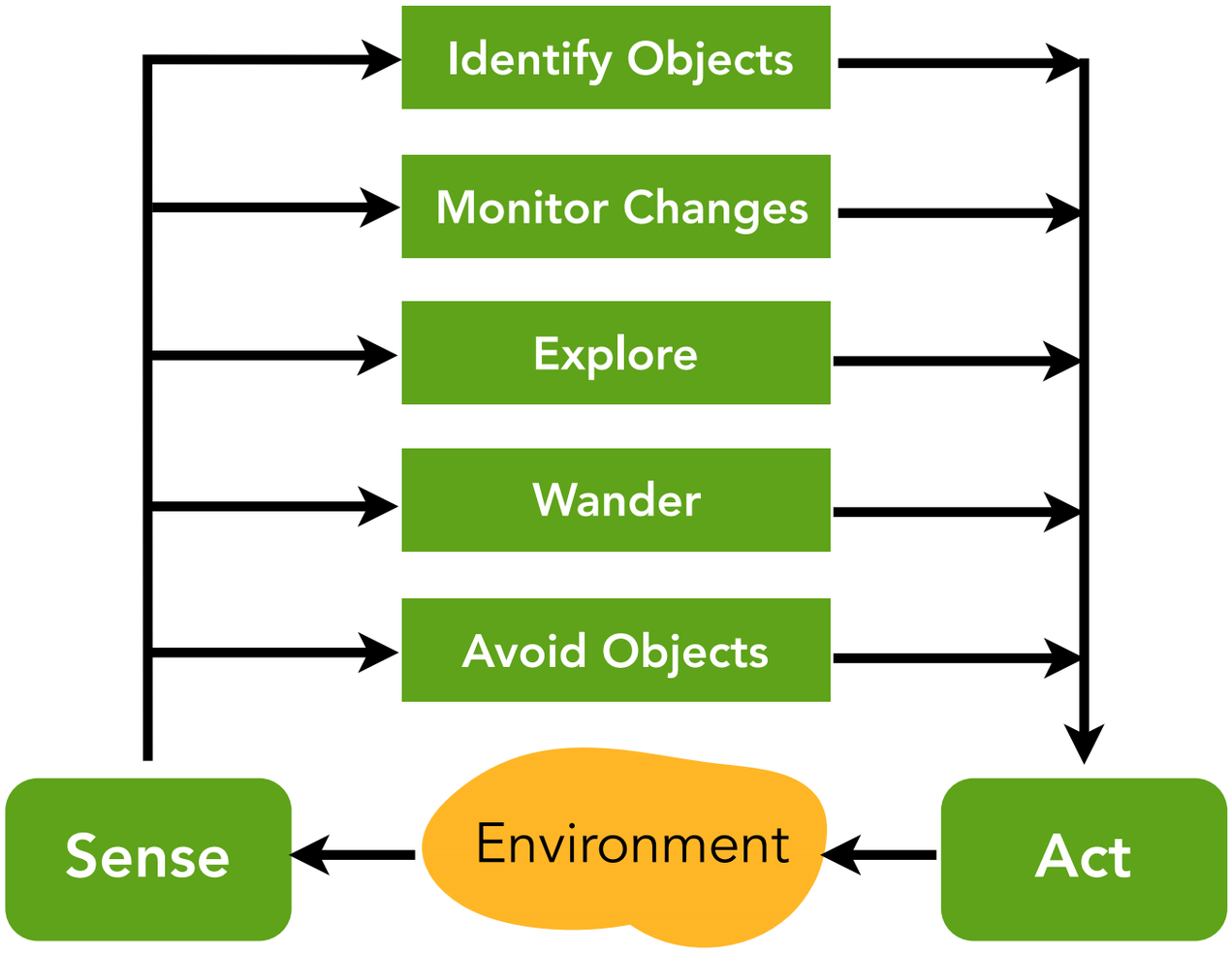
Mathematical view (***Function***):
A transfer function, transforming sensory inputs into actuator commands

Concurrent mode
vs.
Sequential mode

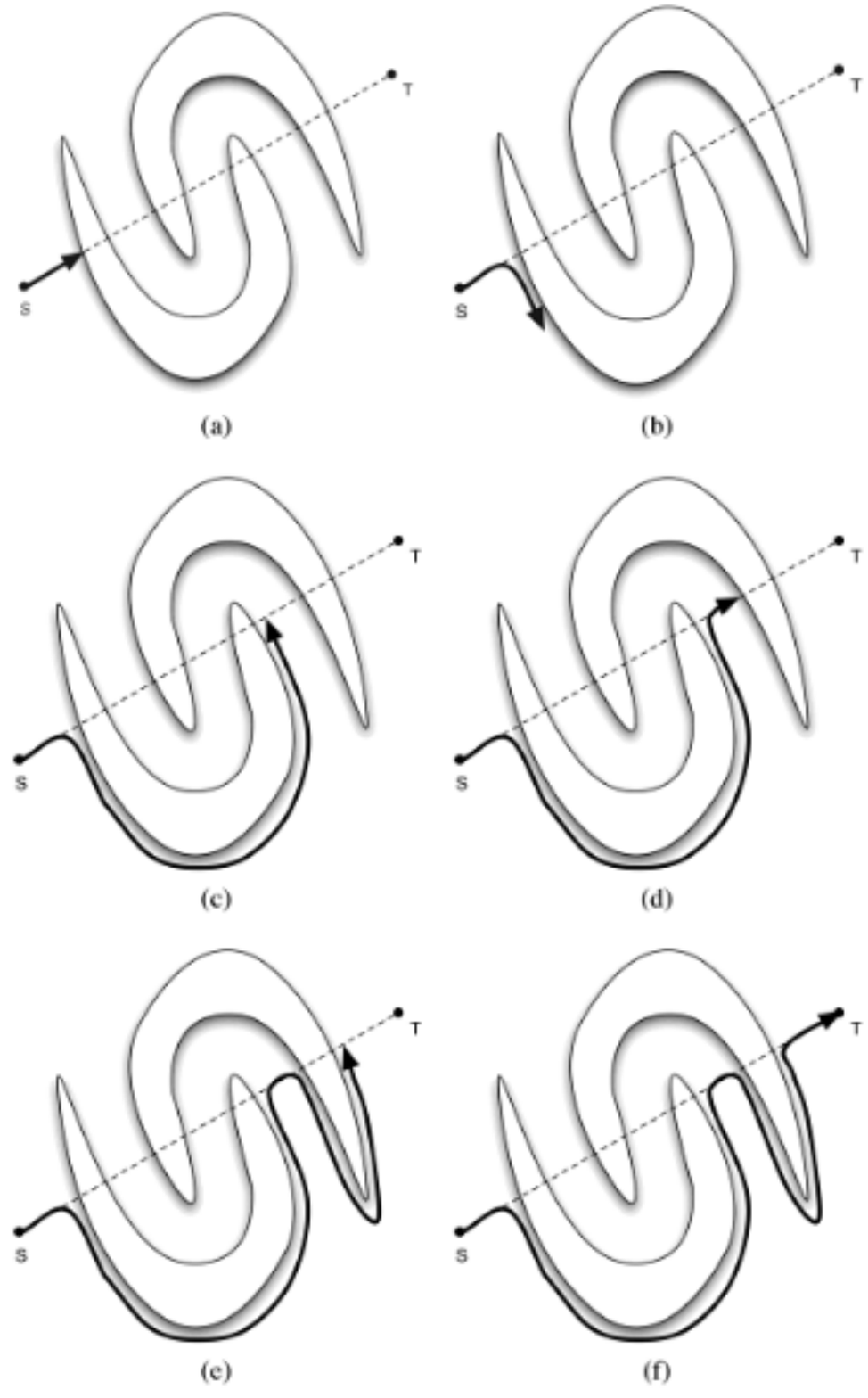
Vertical decomposition
vs.
Horizontal decomposition



REACTIVE CONTROL ARCHITECTURES: *DON'T THINK, REACT!*



A navigation behavior



Bug algorithms:
Example of reactive navigation

DELIBERATIVE VS. REACTIVE CONTROL ARCHITECTURES

Top-down problem solving

Requires a closed world

Deliberative paradigm

Internal models

Look ahead

Functional pipeline

Model free

React to inputs

Concurrent modules

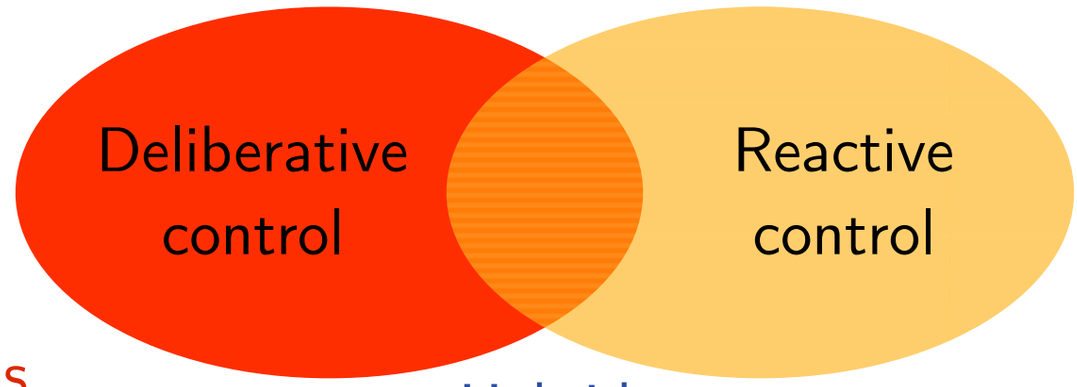
Reactive paradigm

Works in an open world

Bottom-up problem solving

HYBRID CONTROL ARCHITECTURES

The best of the two worlds!



Representations
Models
Planning

Hybrid control

Real-time
Multiple goals
Robustness
Flexibility
Modularity

How to combine ?

Strategic
planning / reasoning

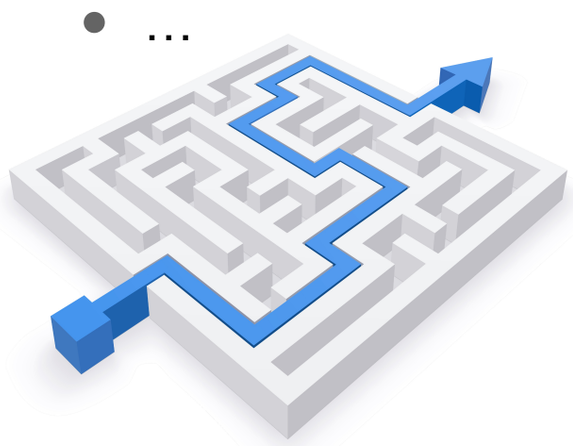
Low(er)-level
controls and behaviors

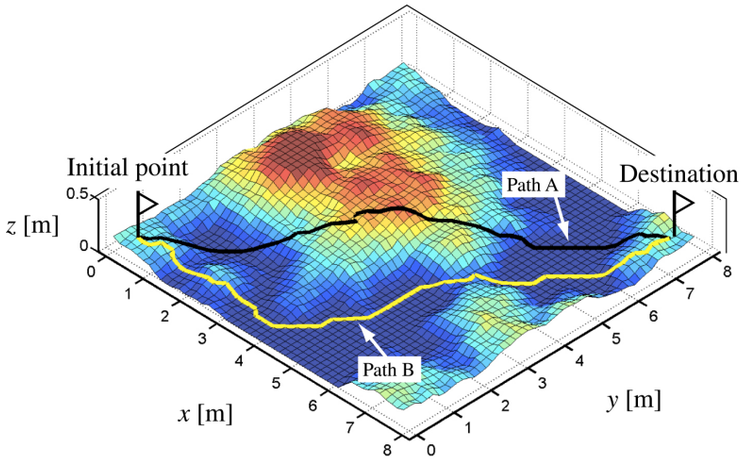
... the most used approach nowadays, but still a sort of art

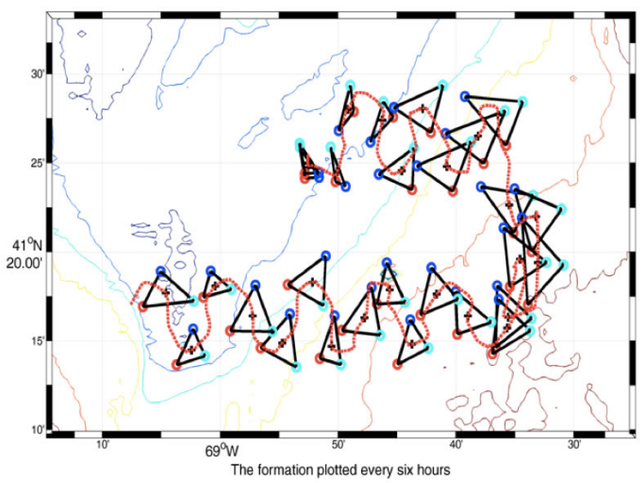
In the final assignment: (i) map-based Plan, (ii) use PI feedback-based control to guide motion, (iii) use EKF for continual state estimation, (iv) detect landmarks, (v) avoid obstacles

MOTION PLANNING

- ***Motion Planning:*** Generation of motions through space ...
  - **Path planning**: Generation of a (feasible) path for going from A to B
  - **Trajectory planning:** Generation of a path for going from A to B specifying how to move based on velocity, time, and kinematics.
  - **Coverage planning:** Visit all places (only once)
  - **Formation planning:** Move from A to B in a formation under specified rules ...
  - ...





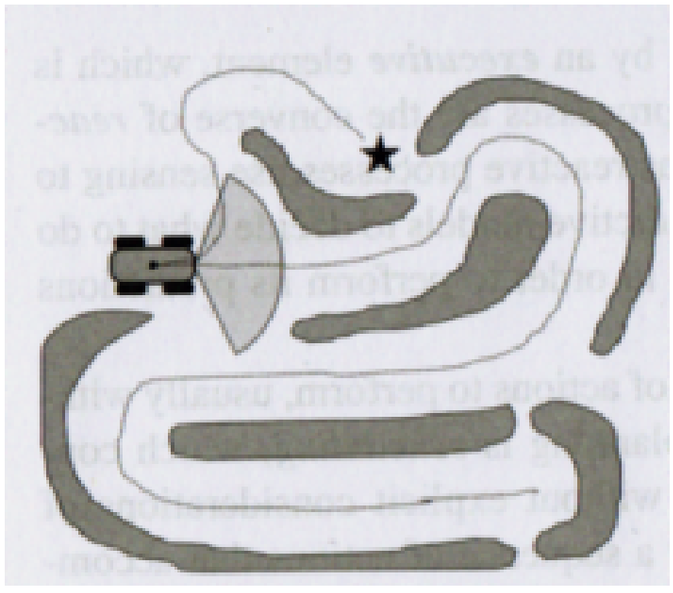


Open Motion Planning Library

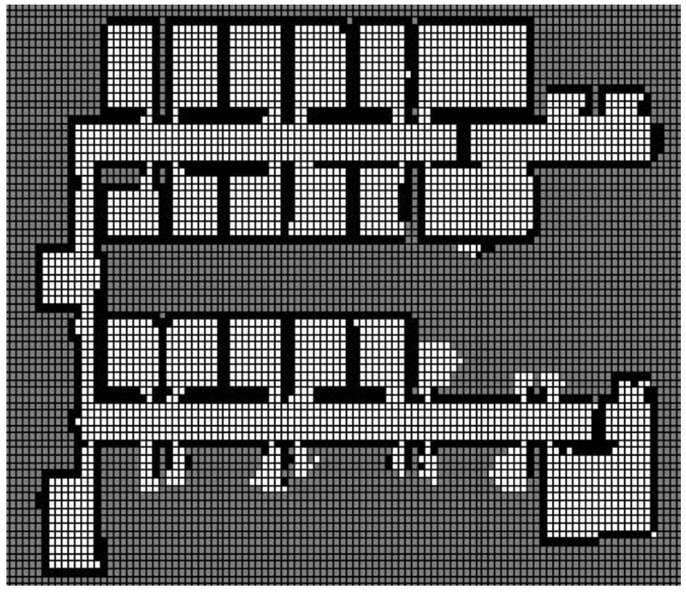
http://ompl.kavrakilab.org/gallery.html

PARAMETERS / DESIGN CHOICES

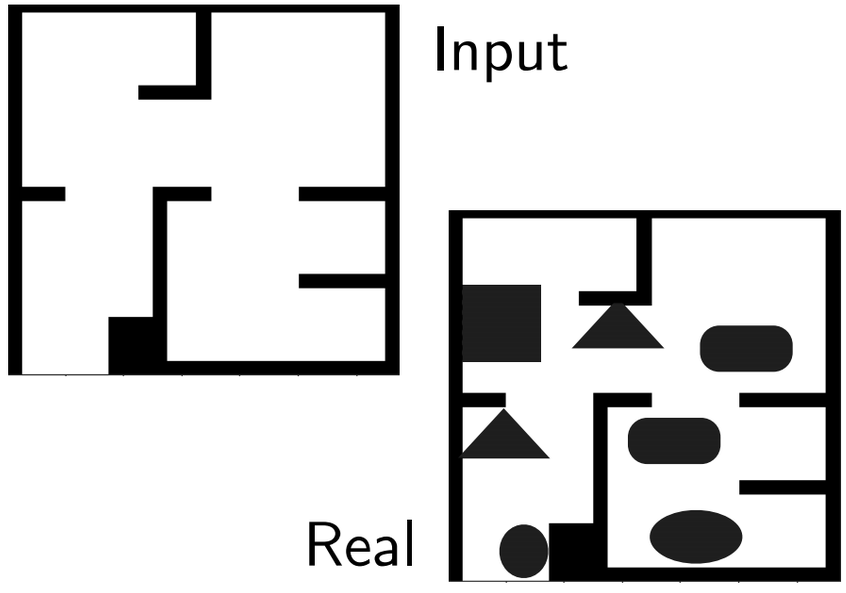
- How far should we look into the future? ➔ **Depth of look ahead** ➔ The farthest, the more *reliable* the plan is and the more *expensive* computation is
  - Model Predictive Control (MPC): iterative replanning up to a certain horizon
- For motion planning, what is the spatial resolution which is required for the input 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 models**?



Without planning / looking ahead things can get arbitrarily bad!



A representation (occupancy grid) extremely fine-grained → Might incur in computational issues



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

PATH PLANNING: PROBLEM FORMULATION

Inputs:

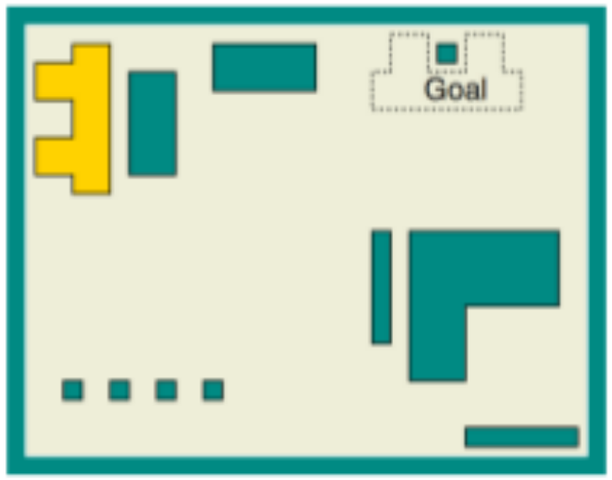
- **Start pose** of the robot
- **Goal pose**
- Geometric and kinematic **description of the robot**
- (Geometric) **description of the world** including existing obstacles

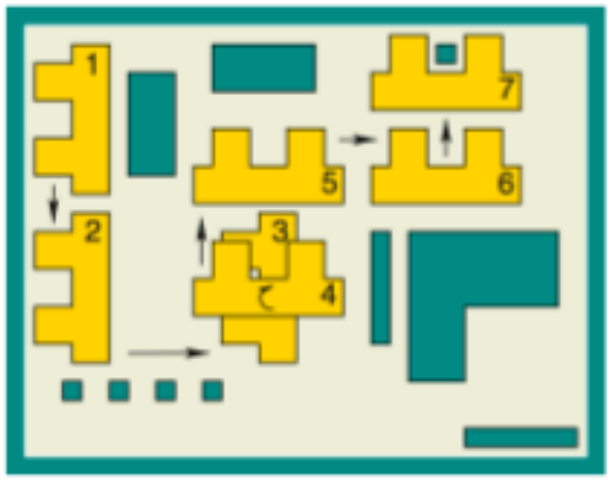
Goal(s):

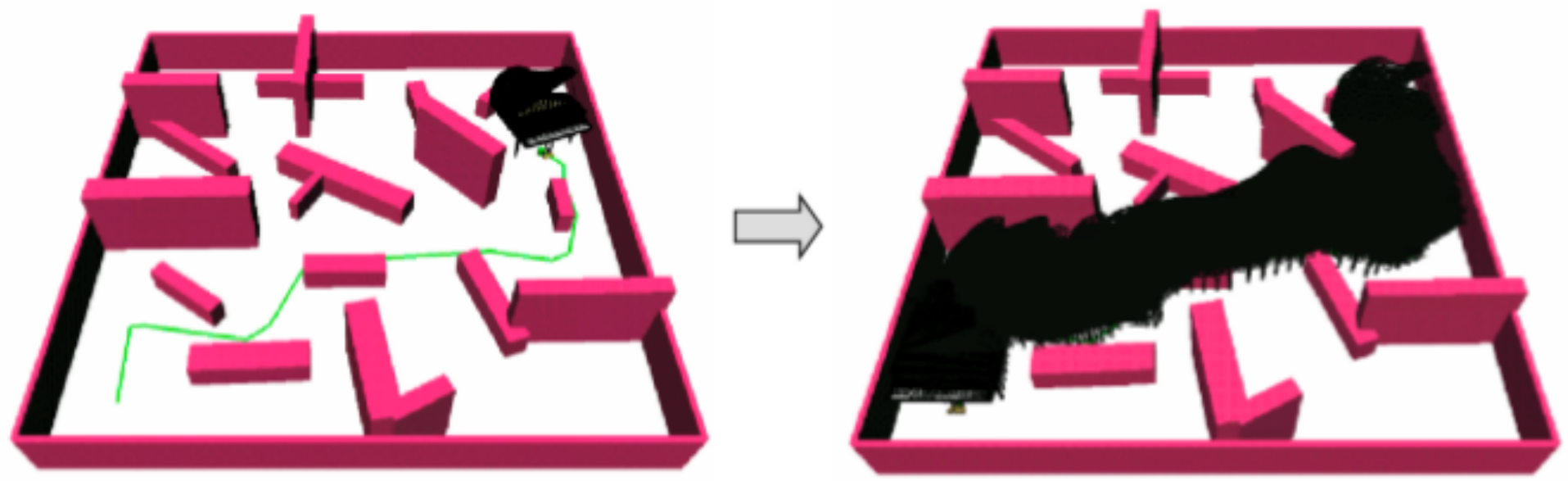
- Find a **path that complies with robot motion constraints** (***feasibility***) and moves the robot gradually from start to goal poses **never colliding with an obstacle** (***admissibility***)
- If no feasible + admissible solution exist ➔ Report a *failure*
- ***Optimality:*** If more than one solution 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 ...

PATH PLANNING EXAMPLES: THE PIANO MOVER'S PROBLEM







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:

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

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

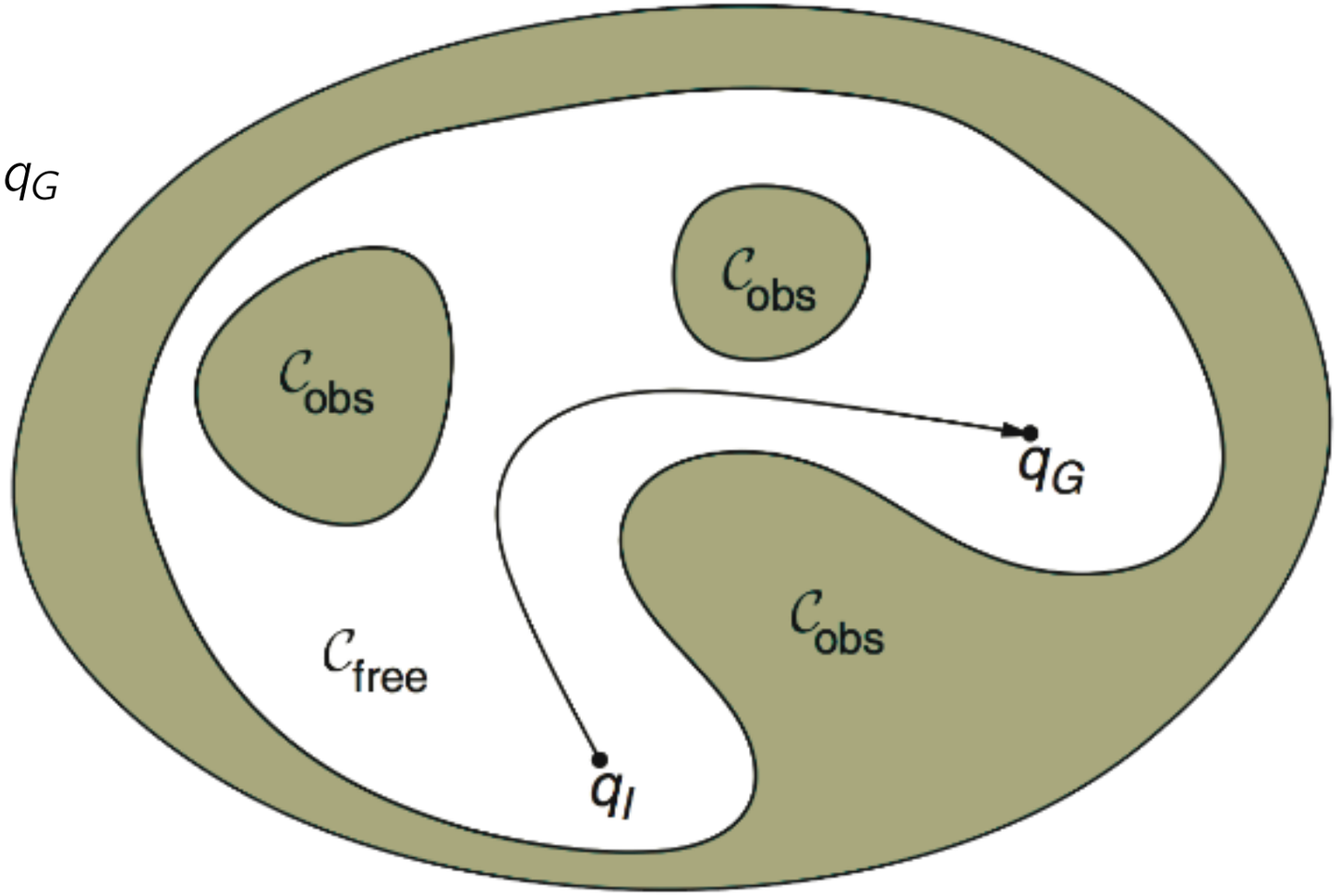
- The *start configuration* is q_I

  The *final (target) configuration* is q_G

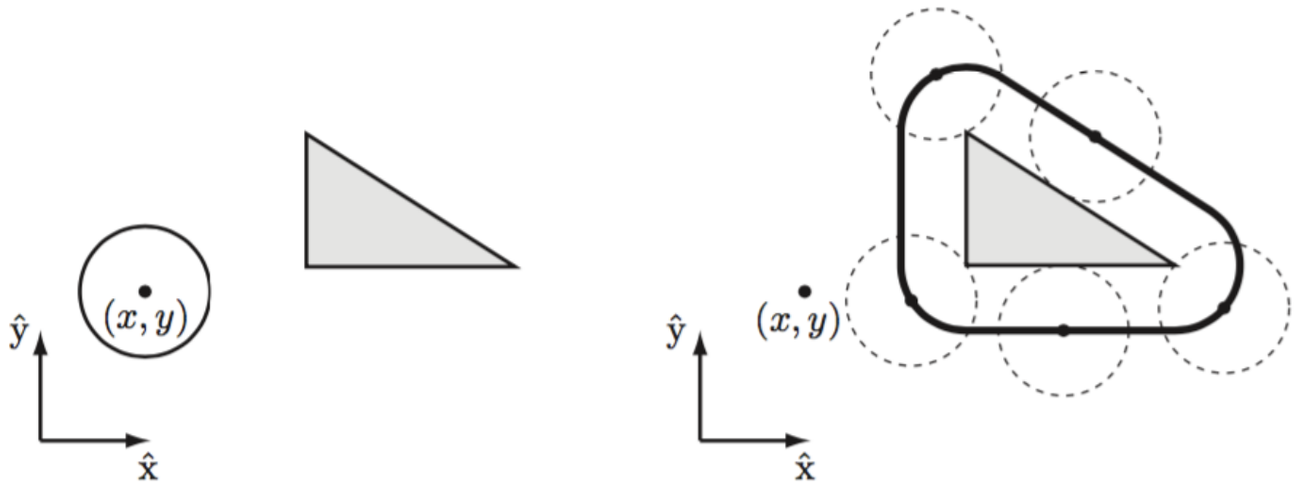
- Path planning amounts to find a continuous parametric path:

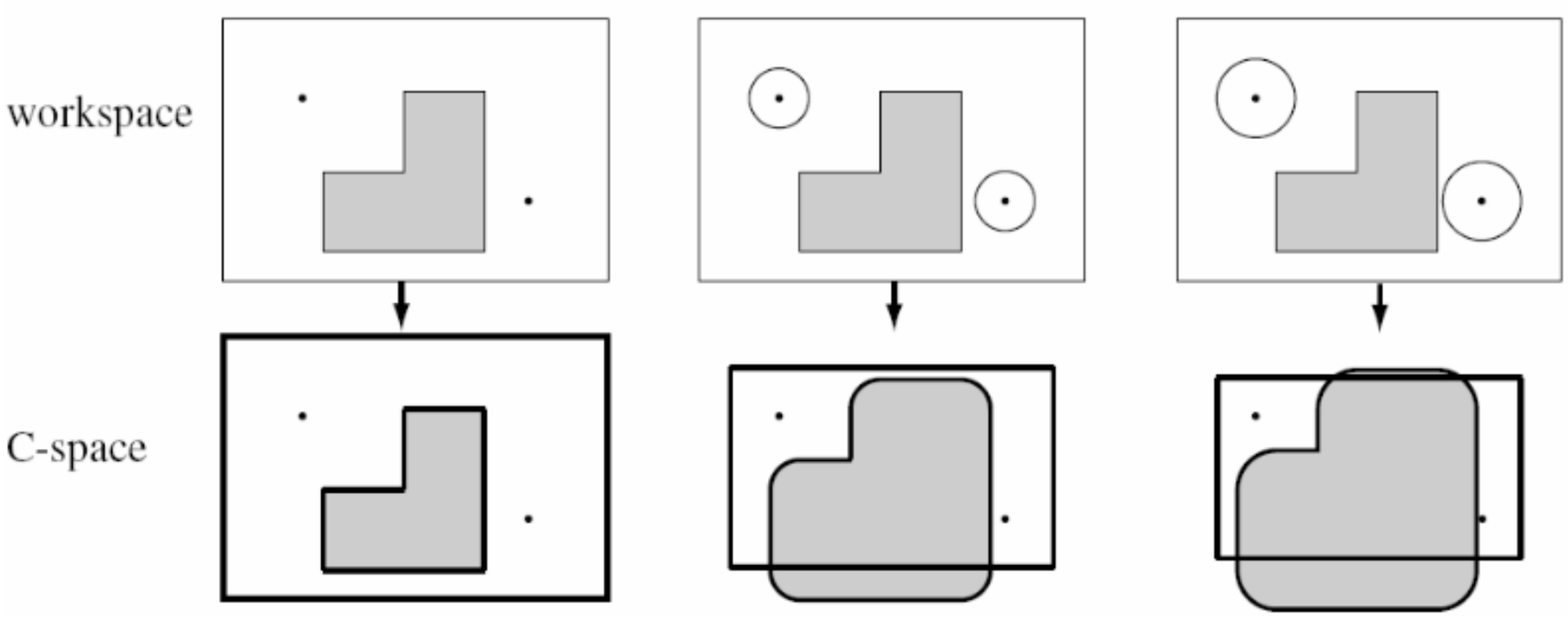
  $\mathcal{P} : [0, 1] \mapsto \mathcal{C}_{free}$

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



CONSTRUCTION OF $\mathcal{C}_{FREE}$ IN PRESENCE OF OBSTACLES



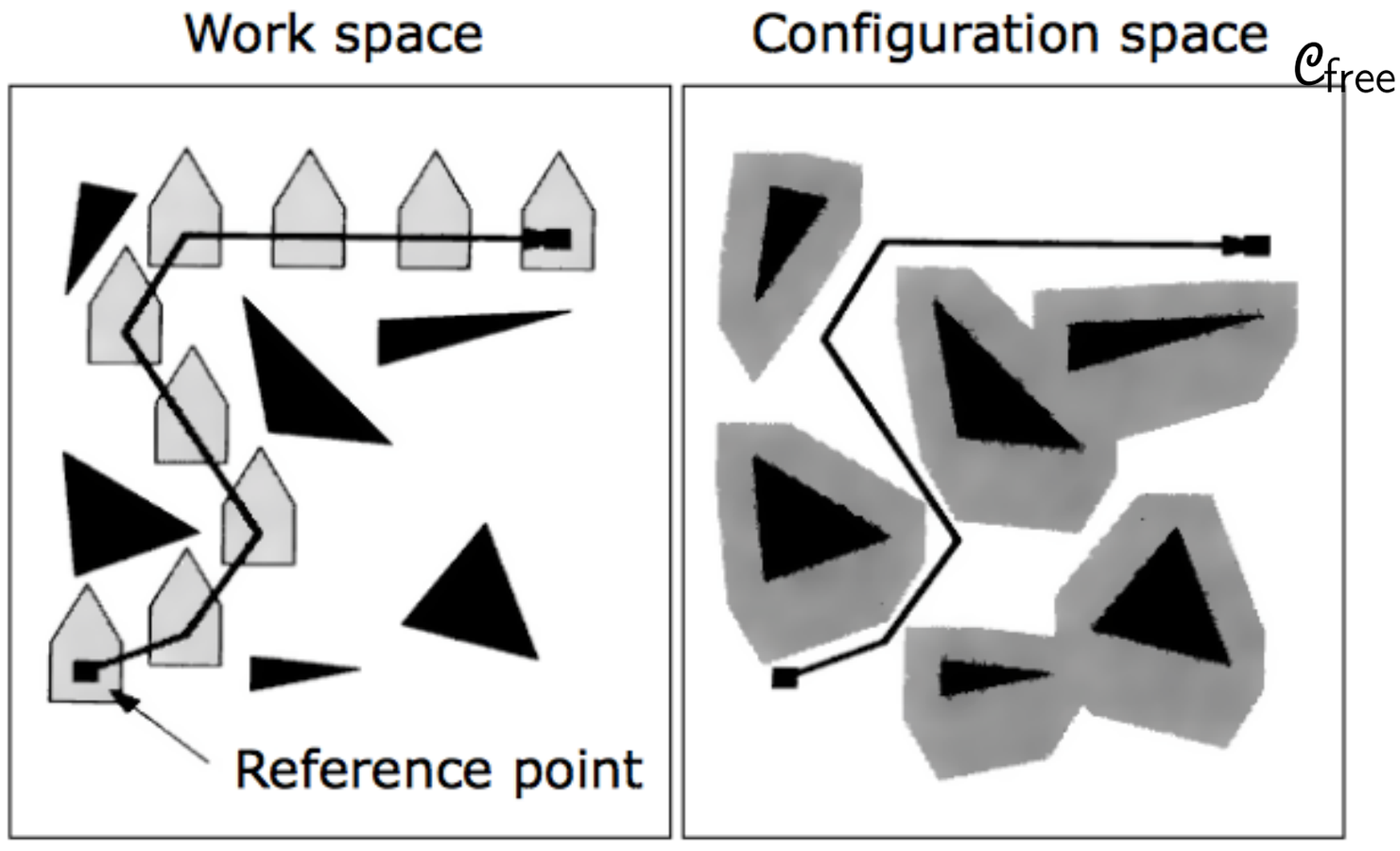


One *reference point* is selected:

The **accessible C-space**, $\mathcal{C}_{free}$, is obtained wrt the reference point by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius

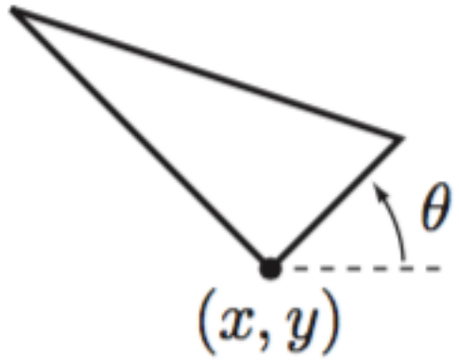
CONFIGURATION SPACE IN PRESENCE OF OBSTACLES

Special case: The robot is a *polygonal* one and can only *translate*

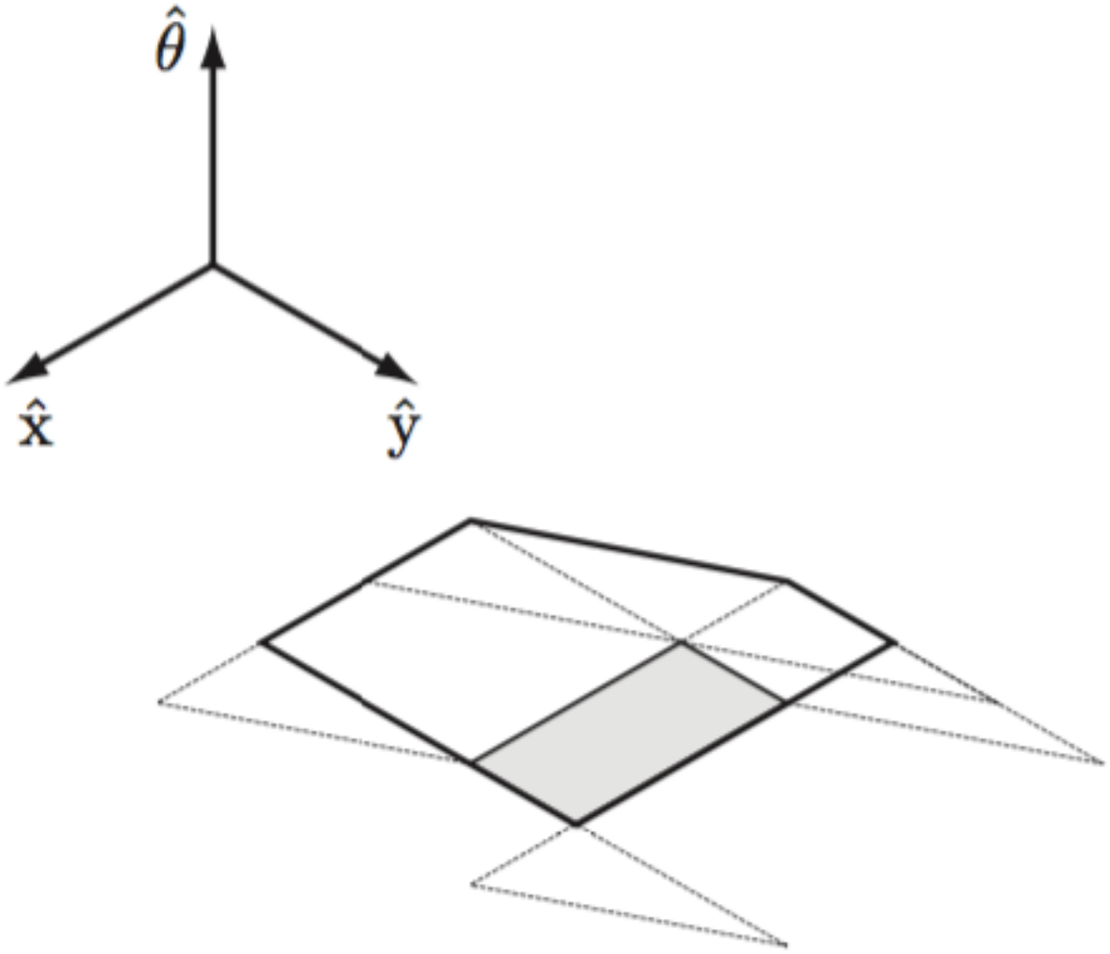


For *motion planning and navigation* a mobile robot of ***any shape*** can be "reduced" to a **point**, precisely the **chosen reference point,** as long as all reasonings are done in the **C-space and the $\mathcal{C}_{\text{free}}$ space is computed**

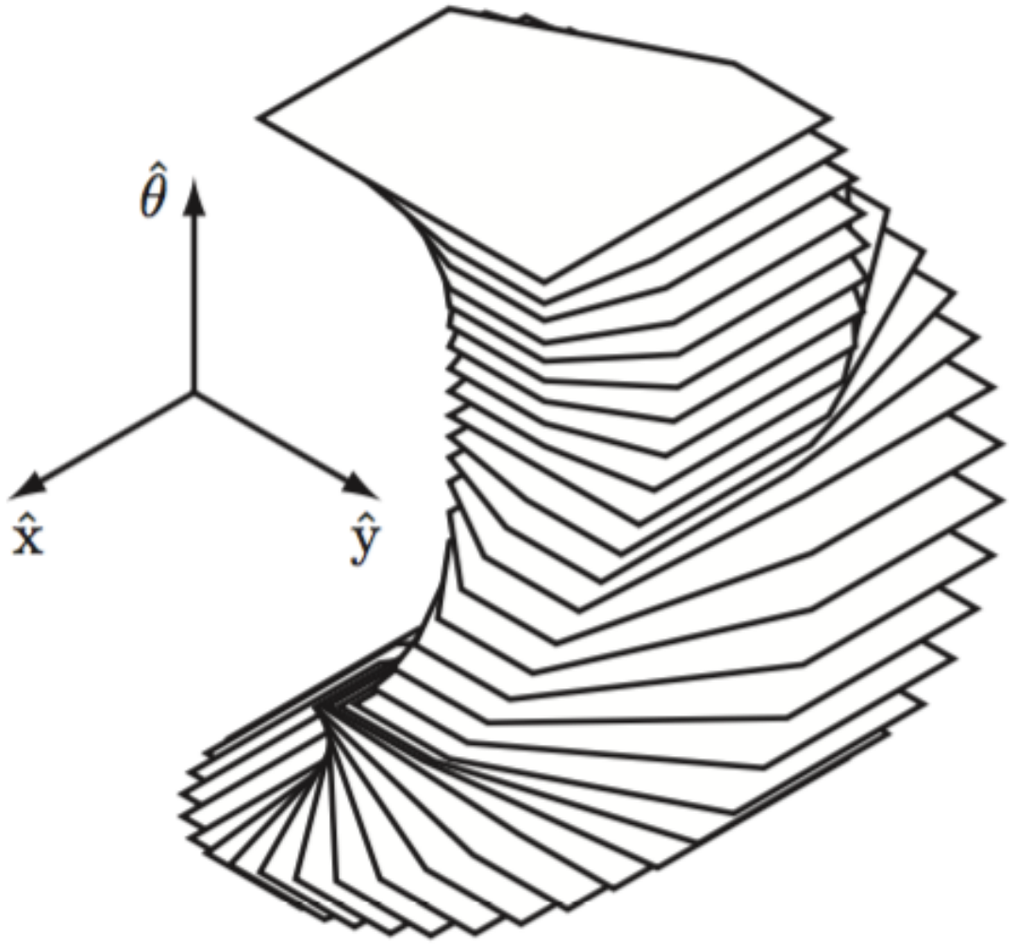
CONFIGURATION SPACE IN PRESENCE OF OBSTACLES



A polygonal robot that can both translate and rotate

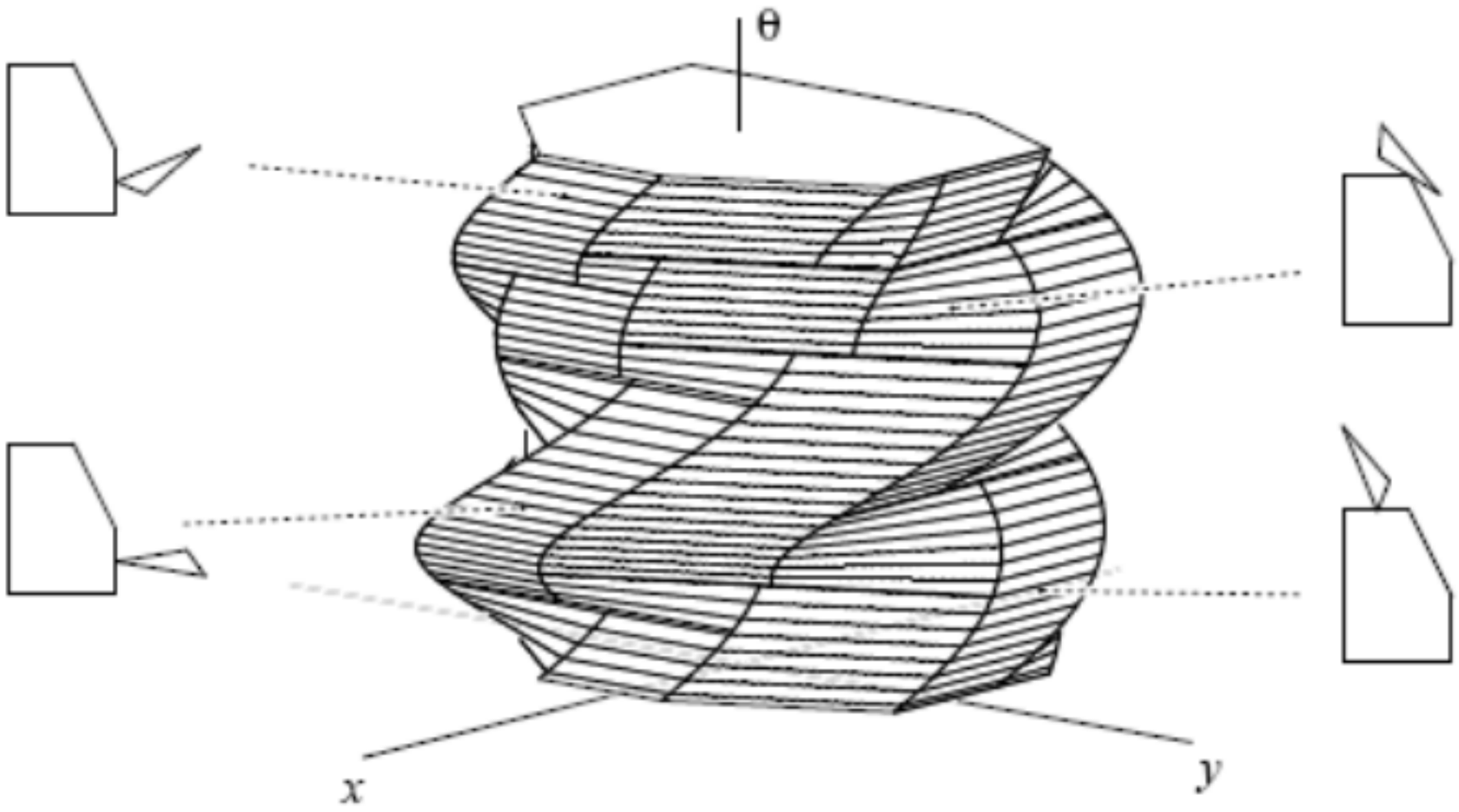


C-space obstacle for the case the robot cannot rotate (θ is fixed to 0)



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

ANOTHER EXAMPLE

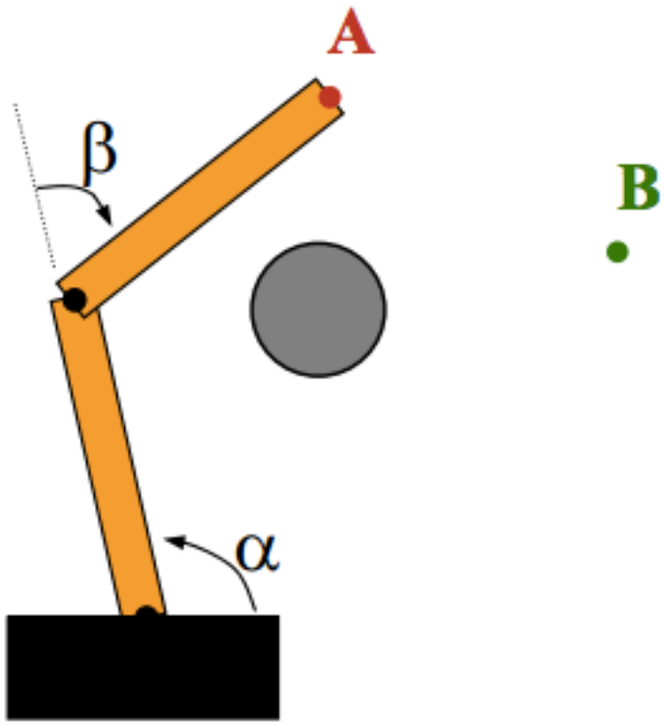


Polygonal robot rotating and translating in a 2D workspace with polygonal obstacles

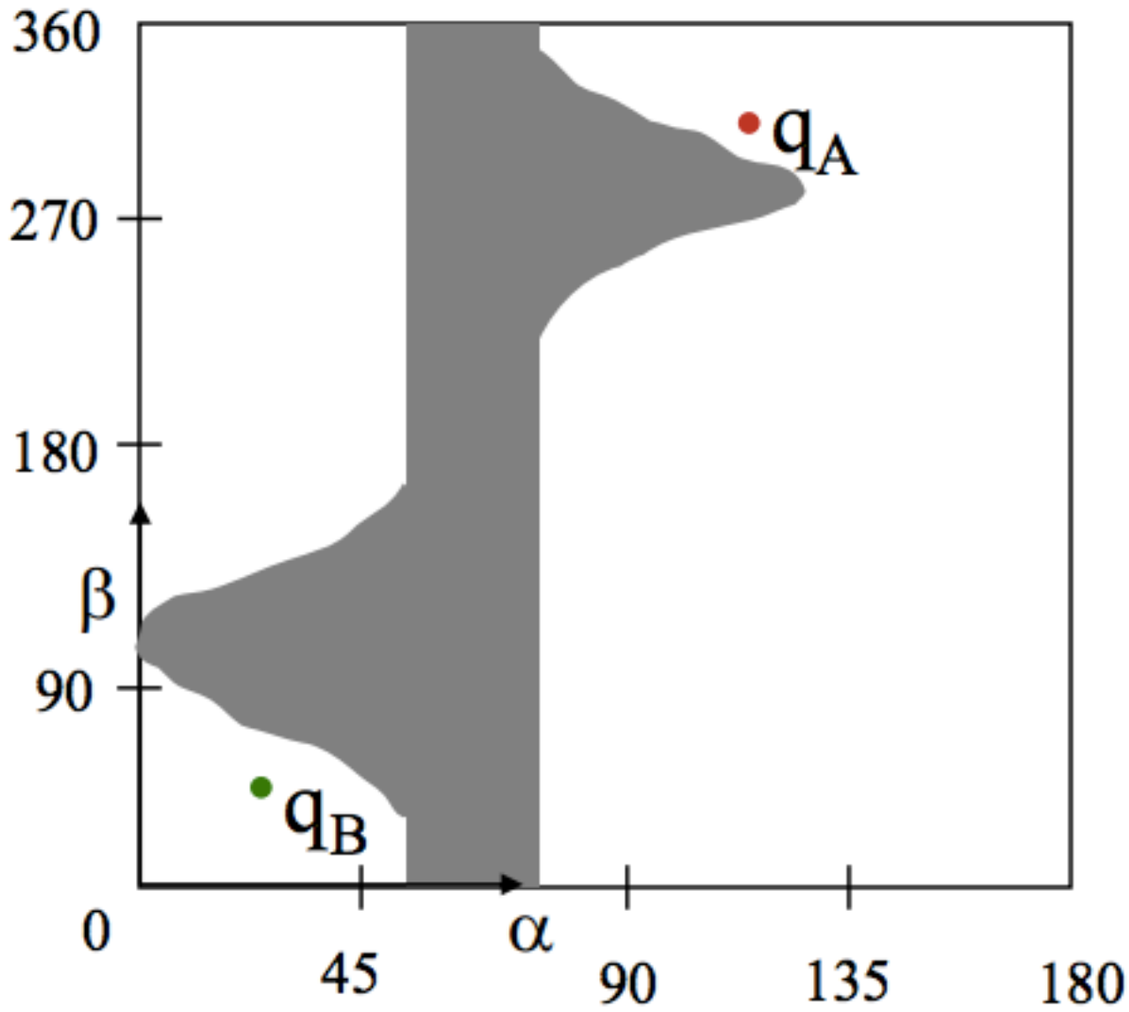
Resulting C-space representation SE(2), where the configuration vector is (x,y,θ)

TOPOLOGY OF C-SPACE

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

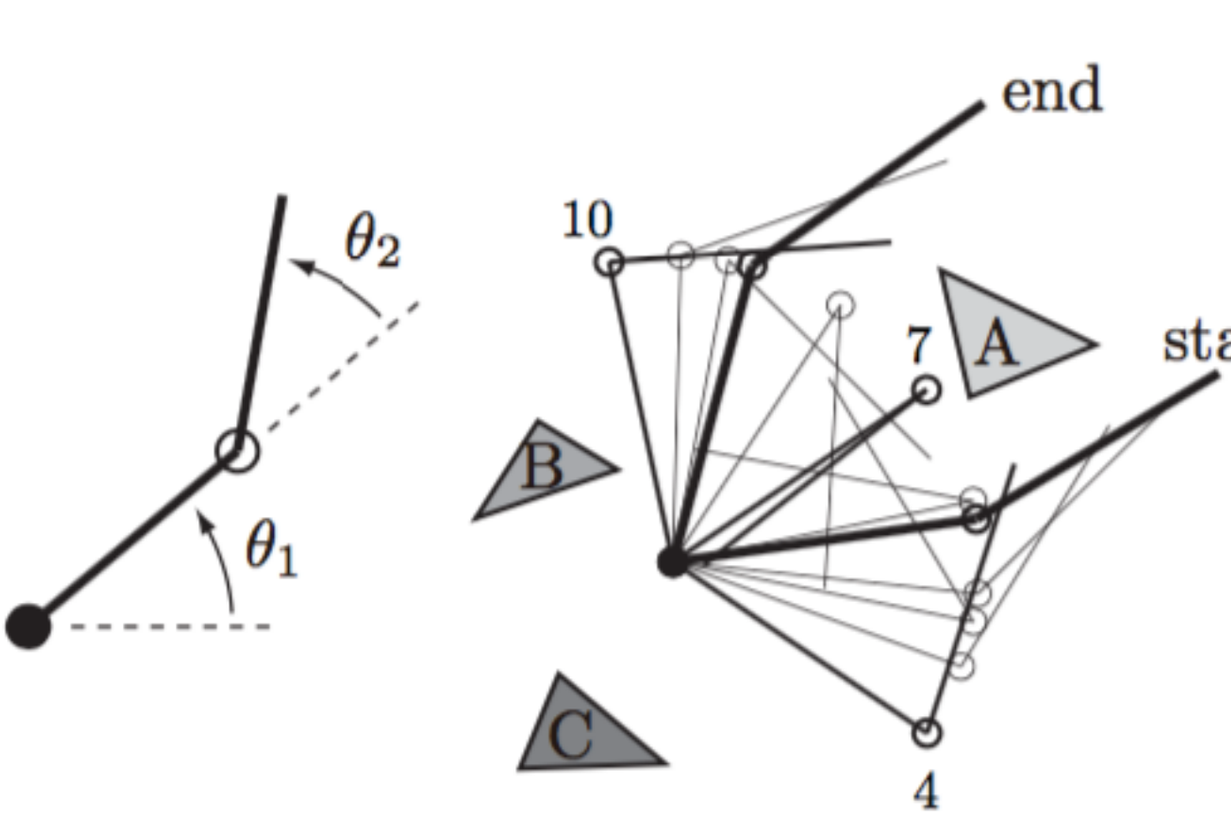


How the two-link robot arm can move from A to B based on the presence of the grey obstacle?

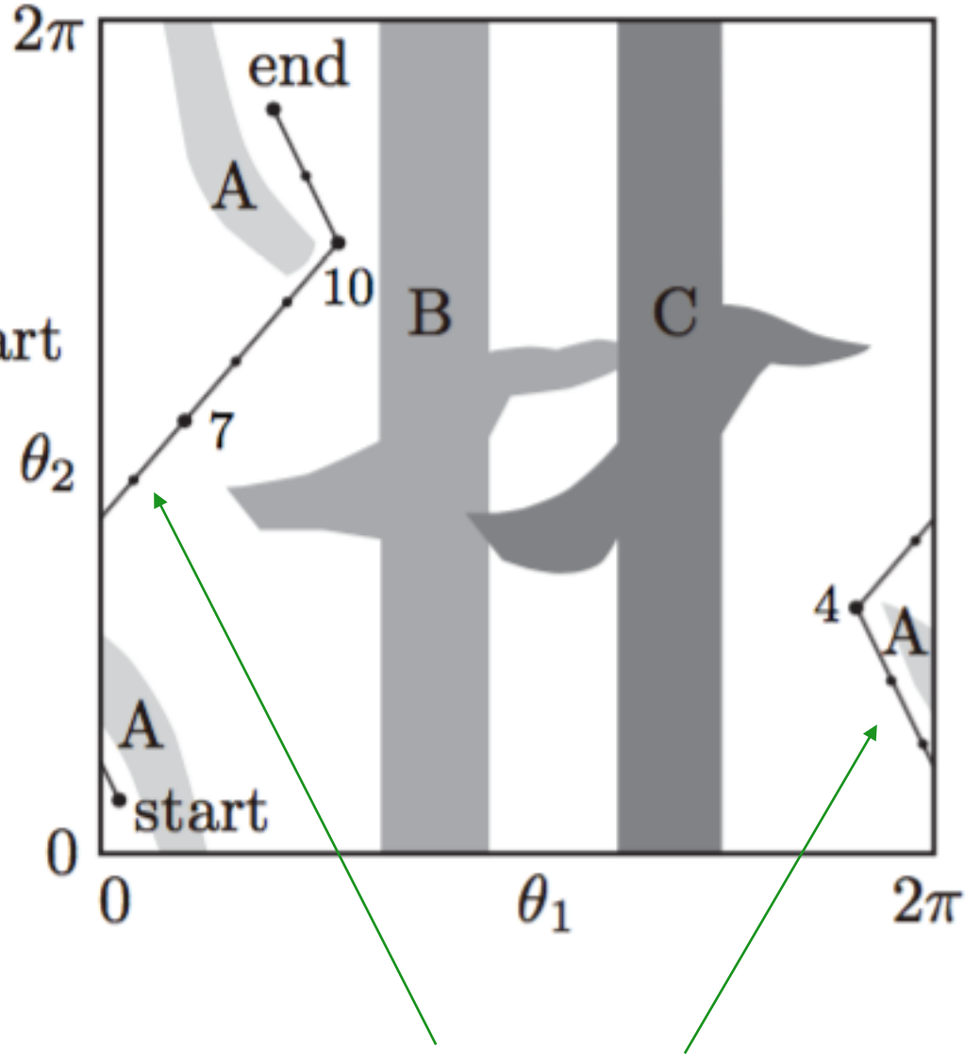


Resulting C-space representation of the obstacle for the robot arm

TOPOLOGY OF C-SPACE



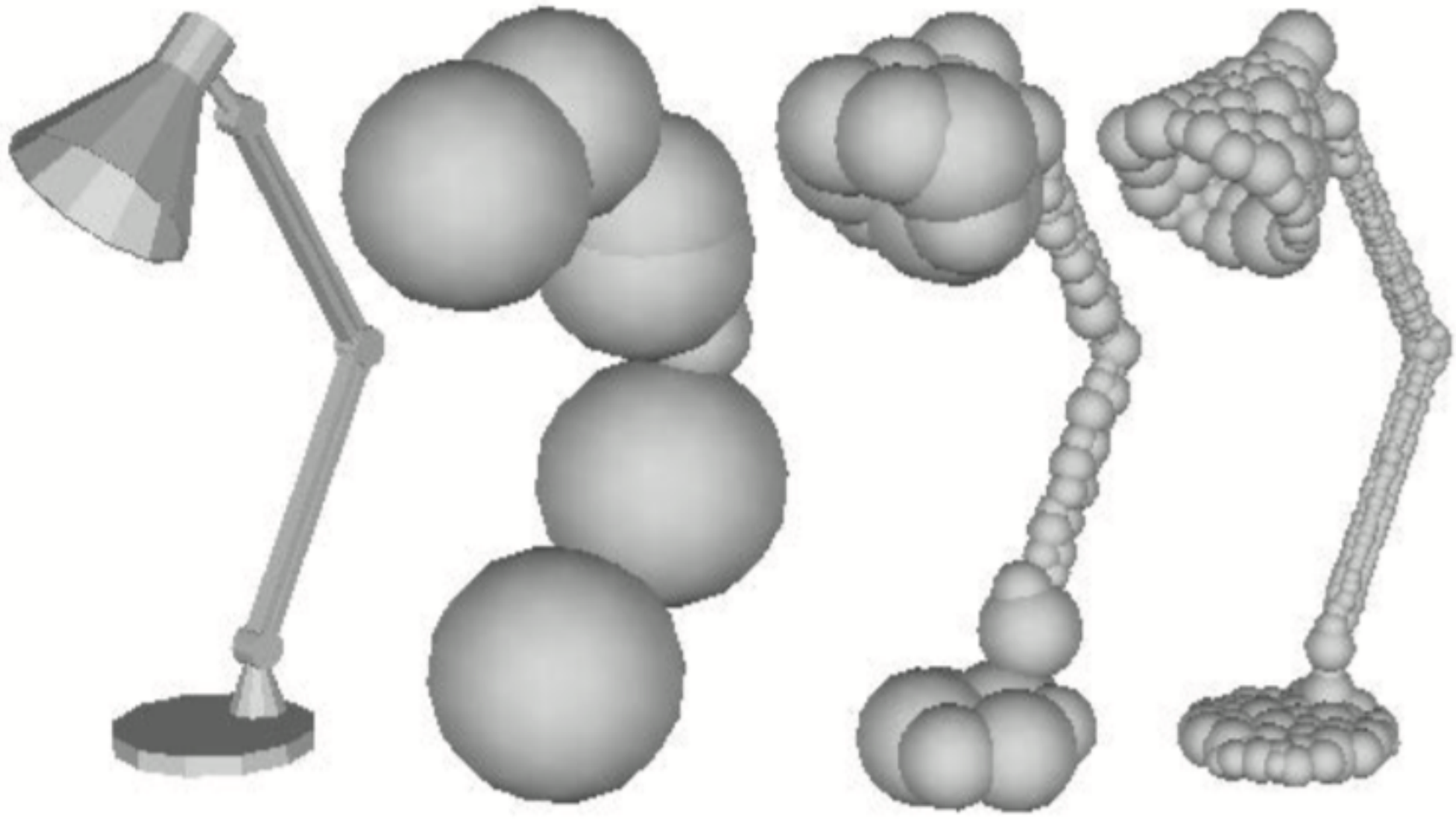
How the two-link robot arm can move from start to end based on the presence of the three obstacles A, B, and C?



Path, with 3 intermediate points, labeled as 4, 7, 10

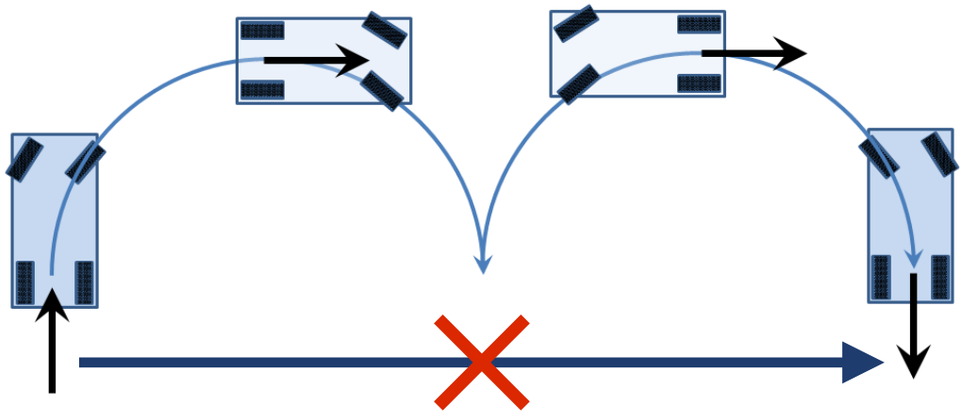
Any configuration lying inside a C-obstacle corresponds to collision with the obstacle by the robot arm in the workspace. The (complex) topology of the C-space obstacles break C_{free} into three connected components

APPROXIMATING COMPLEX OBSTACLES

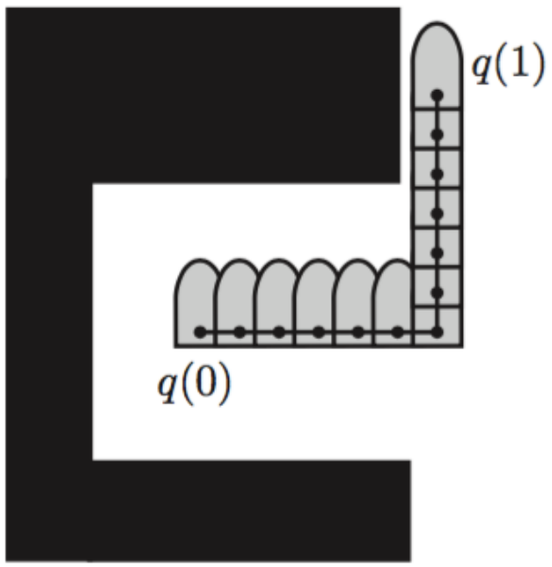


Example of a lamp approximated by spheres: the approximation improves as the number of spheres used to represent the lamp increases.

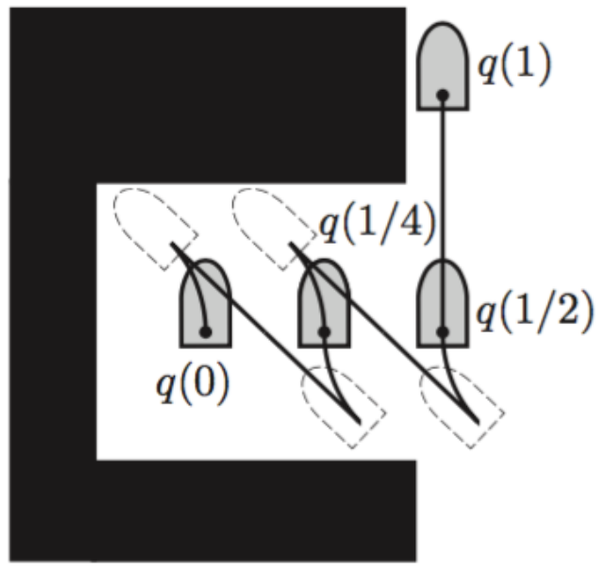
NON HOLONOMIC PATH PLANNING



Two-moves car parking with forward and backward motion capabilities but no no side-way motion



No constraints on the maximum turning angle



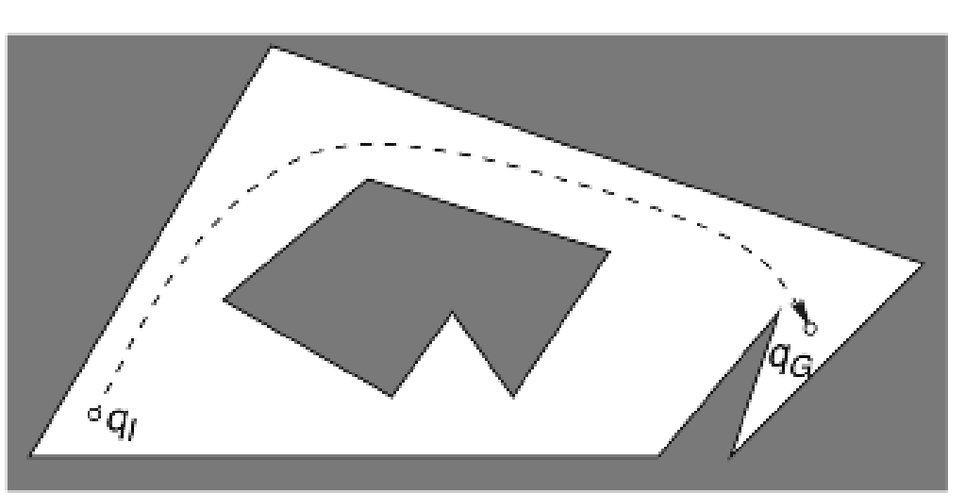
Constraints on the maximum turning angle

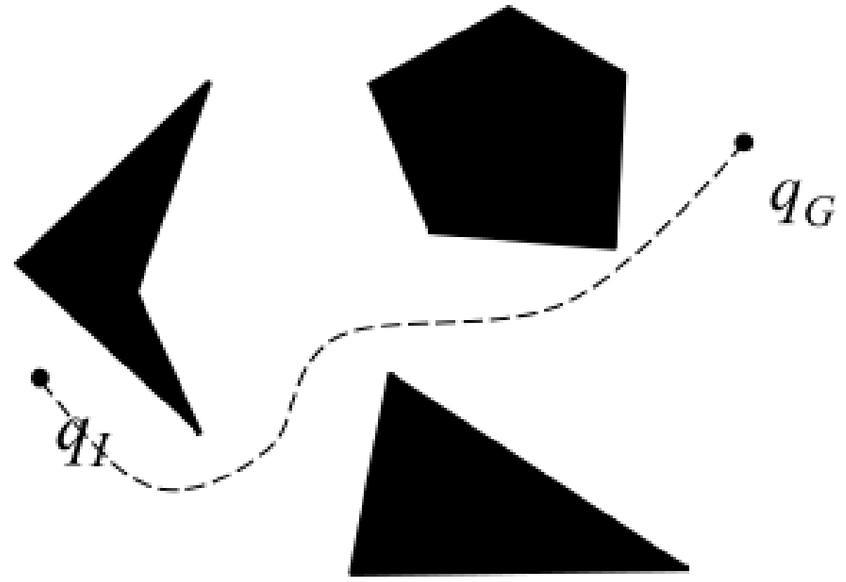
Non Holonomic path planning is a difficult problem!

We have already considered how to deal with non holonomic path planning in the case of obstacle-free workspace ...

LET'S START WITH HOLONOMIC ROBOTS

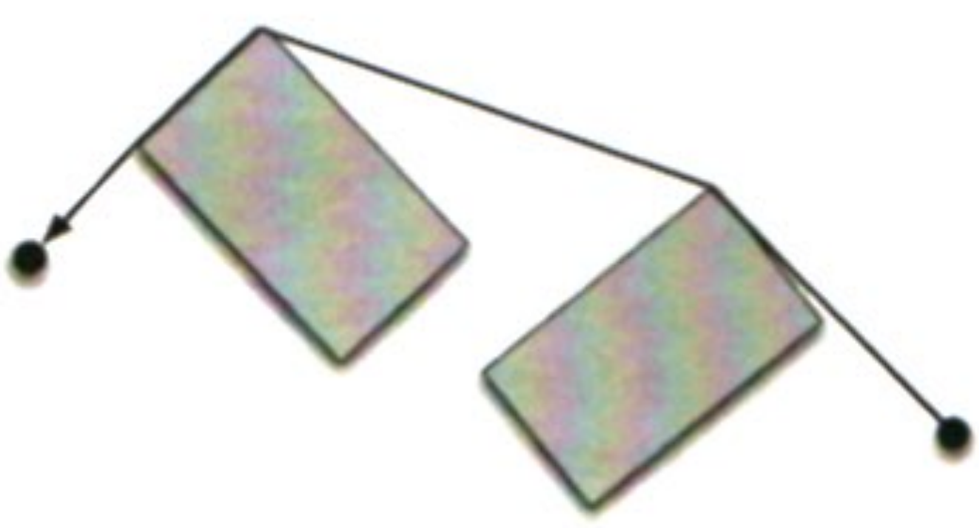
- Let's start, without (much) loss of generality (and performance), to assume to have an omnidirectional (holonomic) point robot moving in a polygonal world: $\mathcal{W} = \mathbb{R}^2$, $\mathcal{C} \subset \mathbb{R}^2$

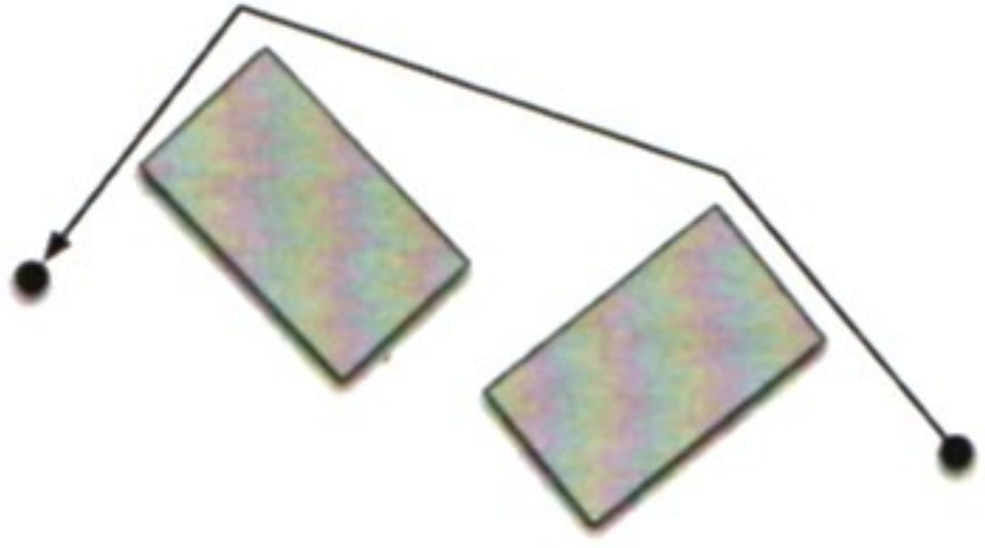




SEMI-FREE SPACE: TOUCHING THE OBSTACLES

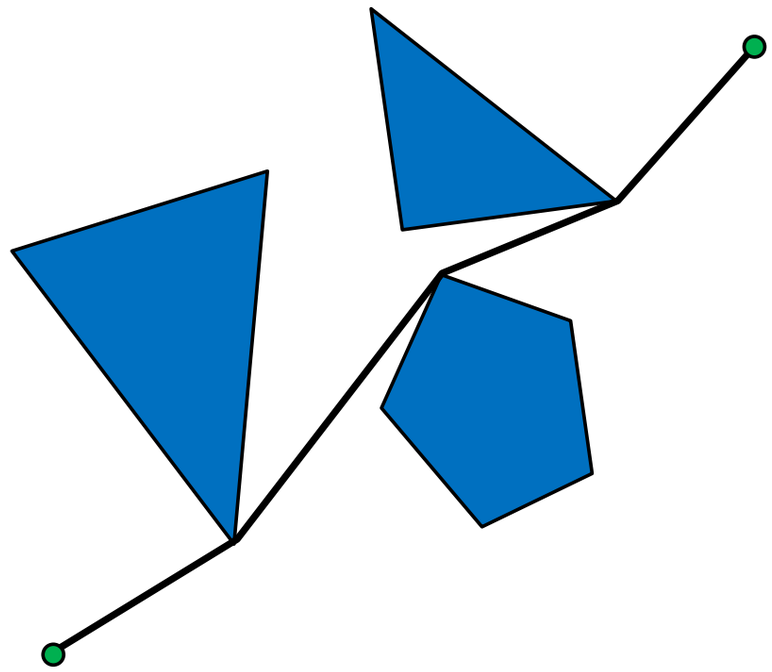
- ***Semi-free paths:*** the obstacles can be touched
- The semi-free space is a *closed set*, which can be (also) useful to prove optimality (the free space is an open set, since the obstacles' frontier does not belong to the set)





OPTIMAL SHORTEST PATH?

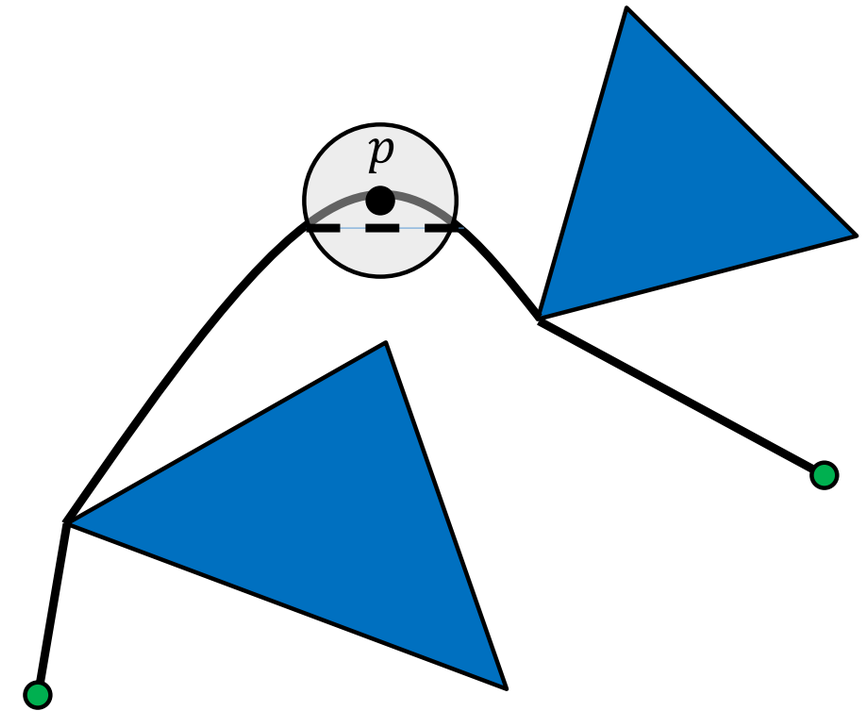
- Polygonal path: sequence of connected straight lines
- Inner vertex of polygonal path: vertex that is not beginning or end



- 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

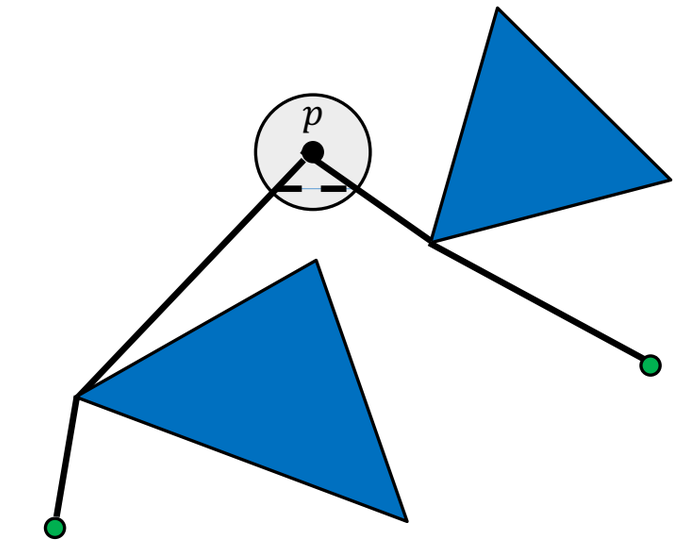
✓ Path is polygonal

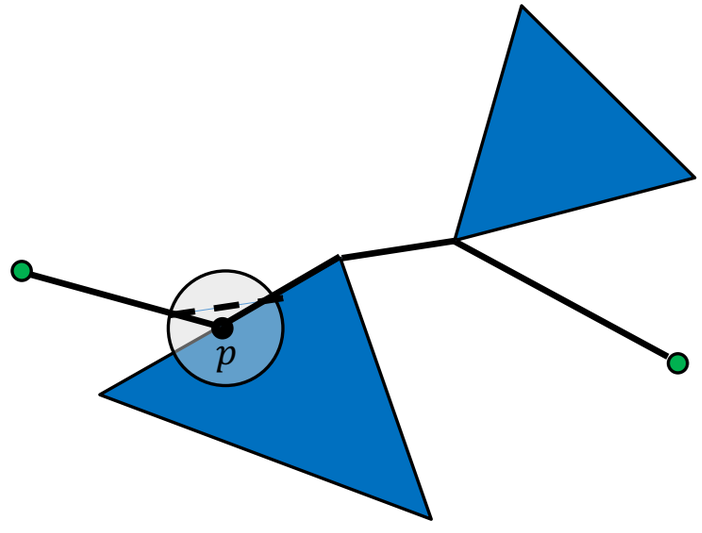
- Vertex cannot lie in interior of free space, otherwise we can do the same trick and shorten the path (that would not then be the shortest)

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

✓ Inner vertices are vertices of obstacles

■

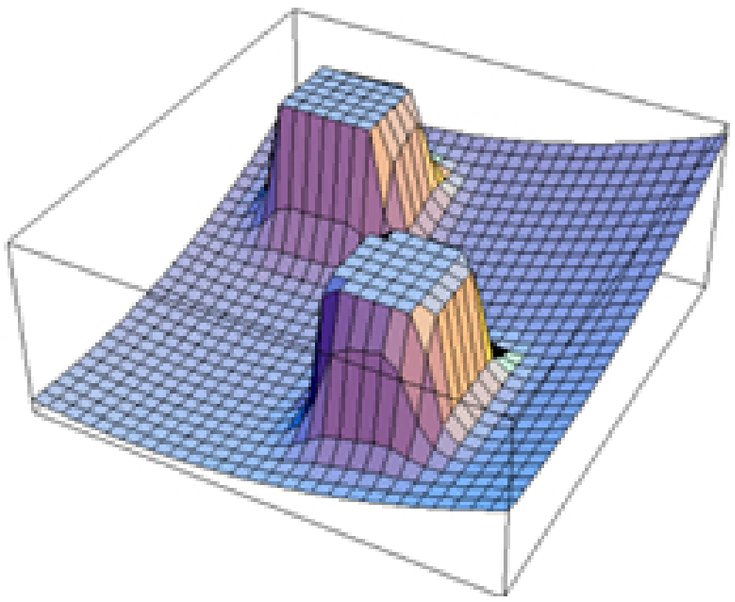


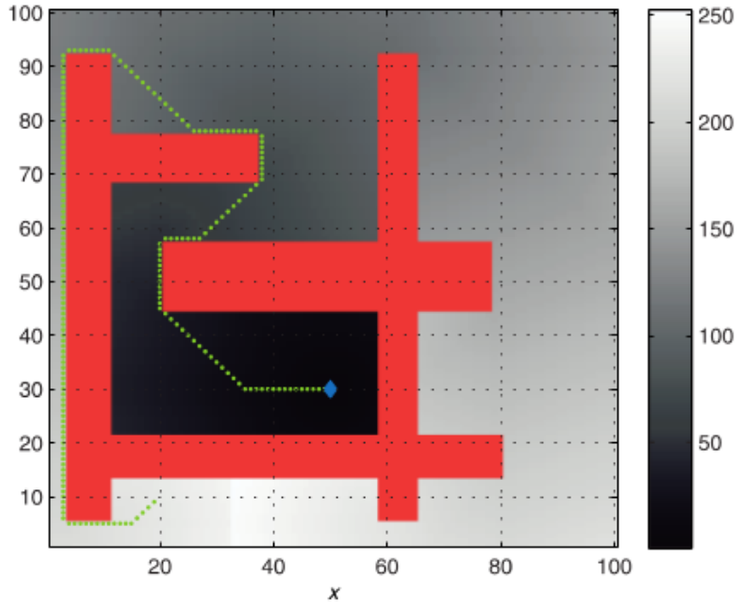


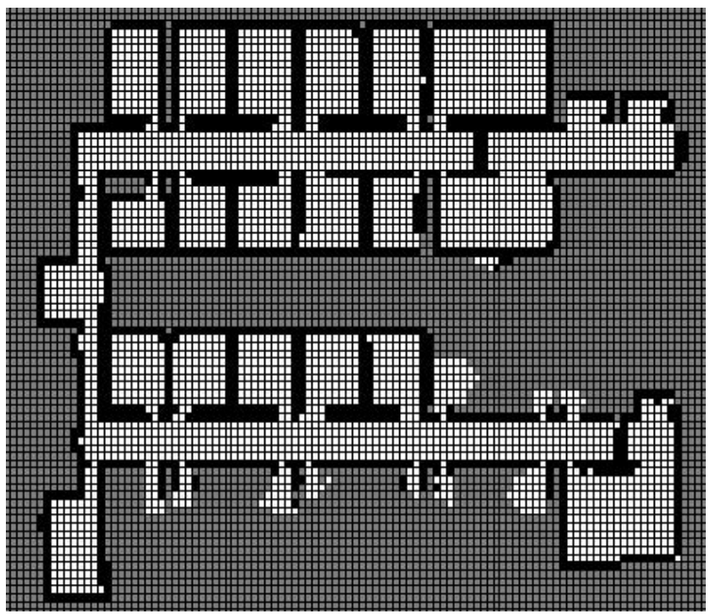
REPRESENTING THE C-SPACE

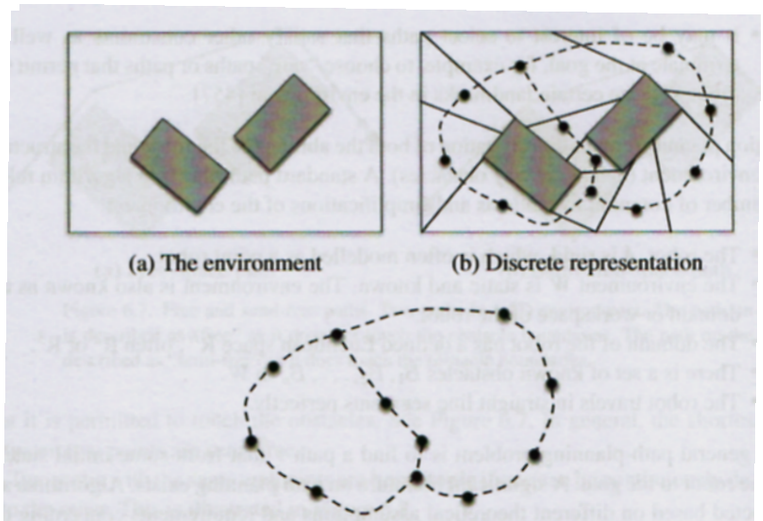
- The admissible search space can be treated as:
  - Continuous space: Challenging to model and demanding from a computational point of view (example approach: *Potential fields*)
  - Discrete space: Most common approach, usually based on deriving a (computationally tractable) graph representation of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

  ...

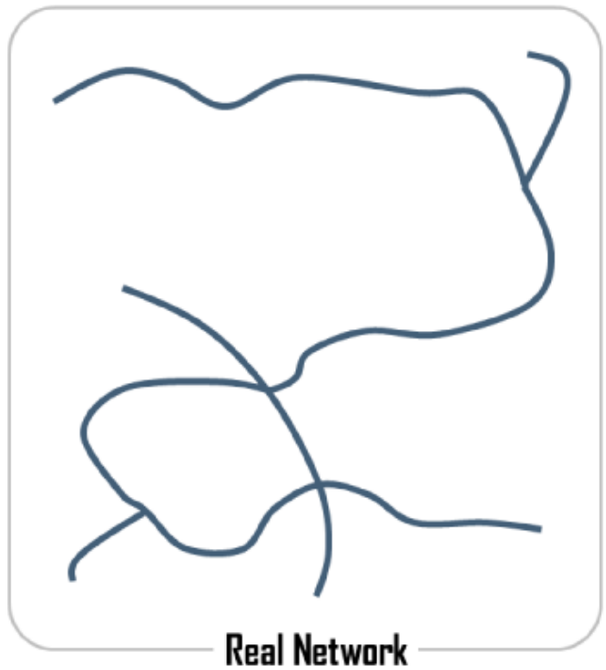




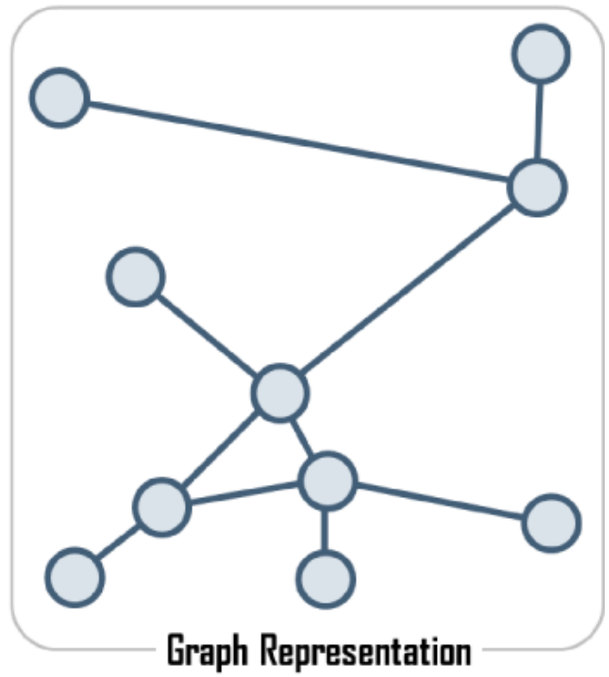




ROAD MAPS



Real Network



Graph Representation

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 $\mathcal{C}_{free}$

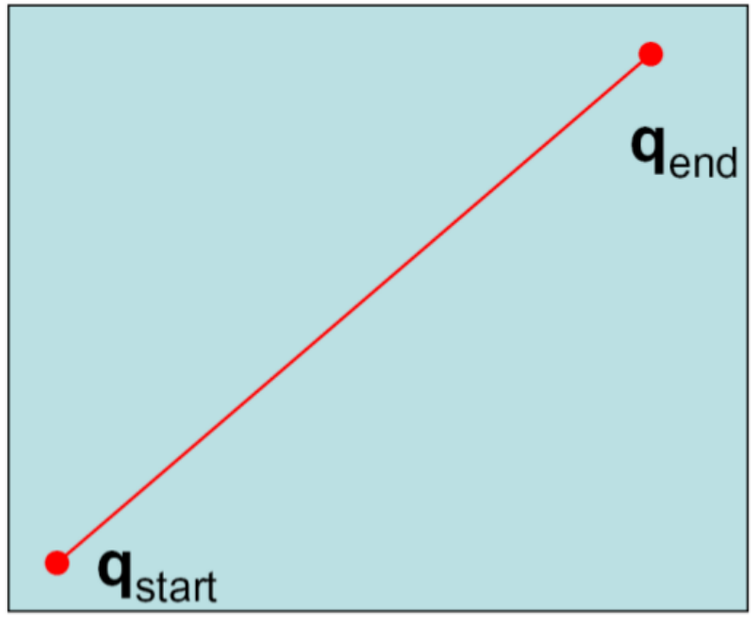
- Instead of defining the problem and computing the paths specifically for the start and goal configurations, it's more flexible to define a general and compact **road map**, which can be used to plan the path between two arbitrary points
- A (small) graph is pre-computed such that staying "on the roads"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

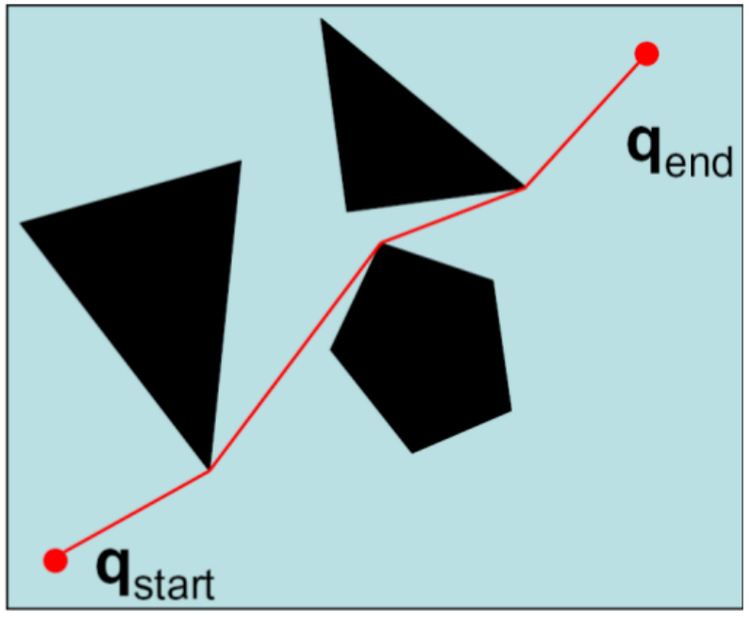
VISIBILITY GRAPHS

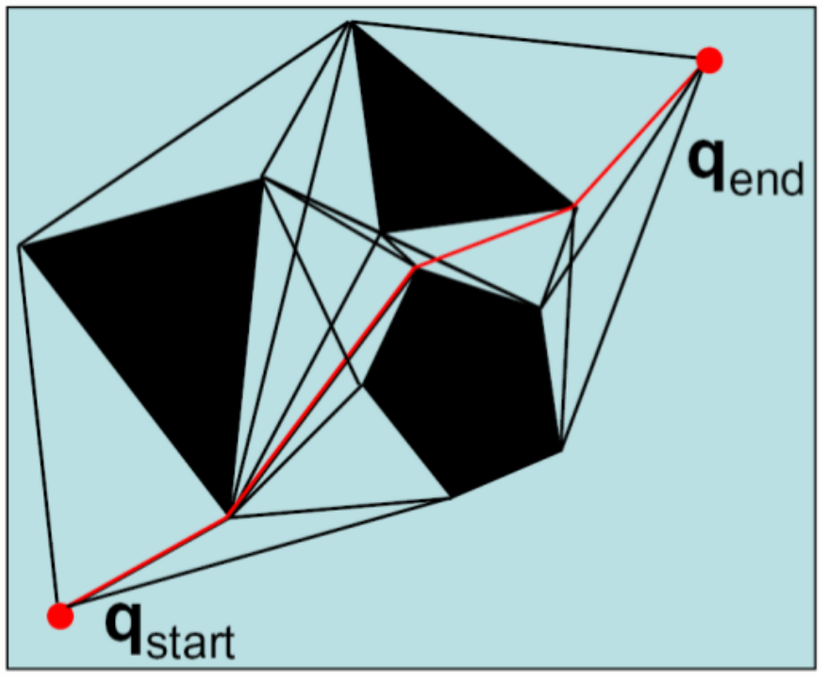
Given polygonal obstacles, a road map can be defined as visibility graph $\mathscr{V}$=(V, E):

- 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**



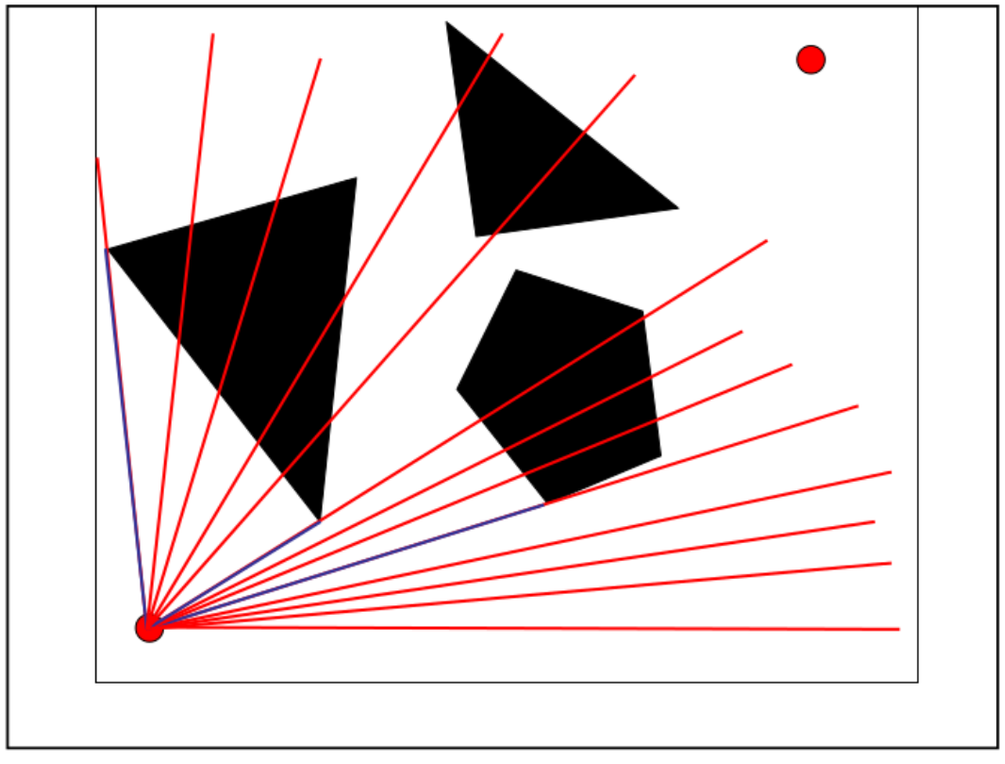


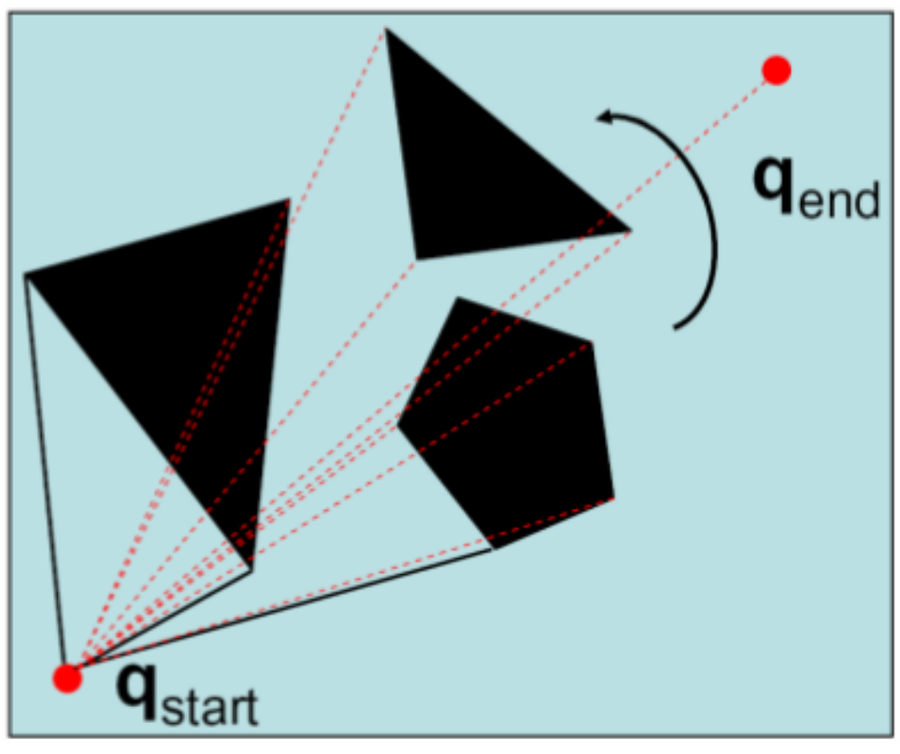


CONSTRUCTION OF THE VISIBILITY GRAPH

- Naive approach: $\mathcal{O}(N^3)$ (N number of vertices)
- Sweep: $\mathcal{O}(N^2 \log N)$
- Optimal: $\mathcal{O}(N^2)$

Sweep: Sweep a line originating at each vertex and record those lines that end at visible vertices





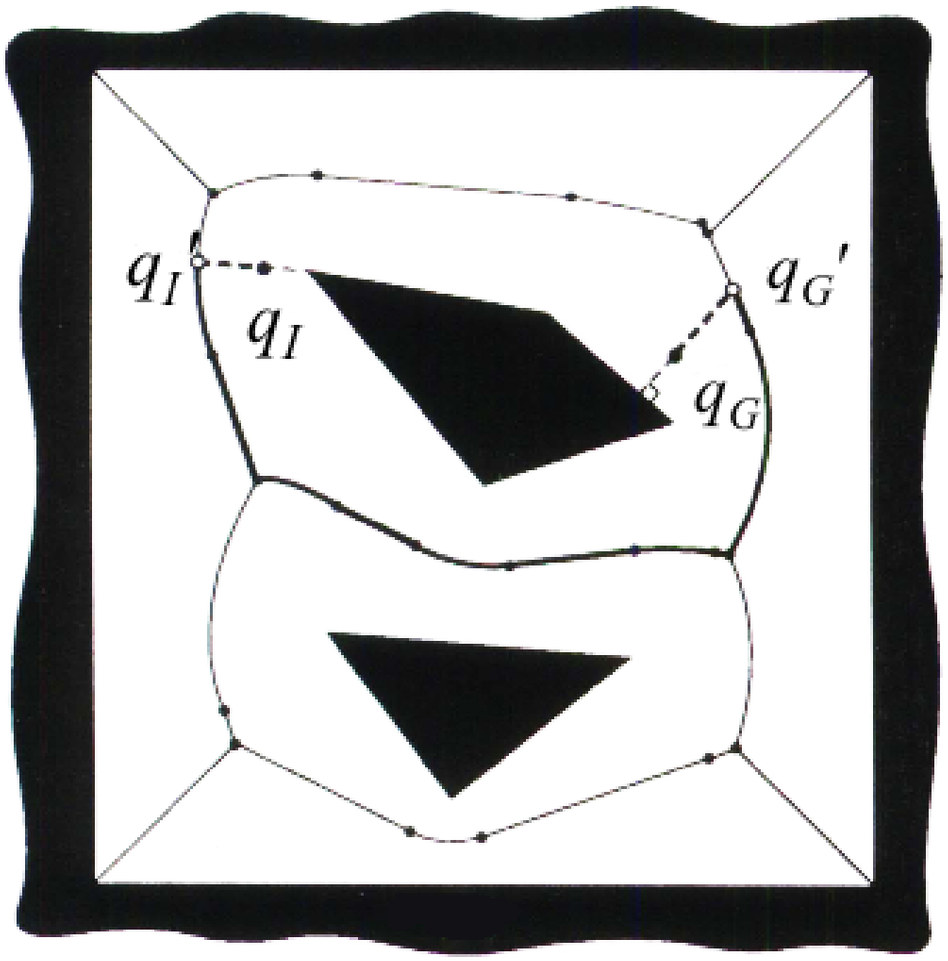
SHORTCOMING OF VISIBILITY GRAPHS

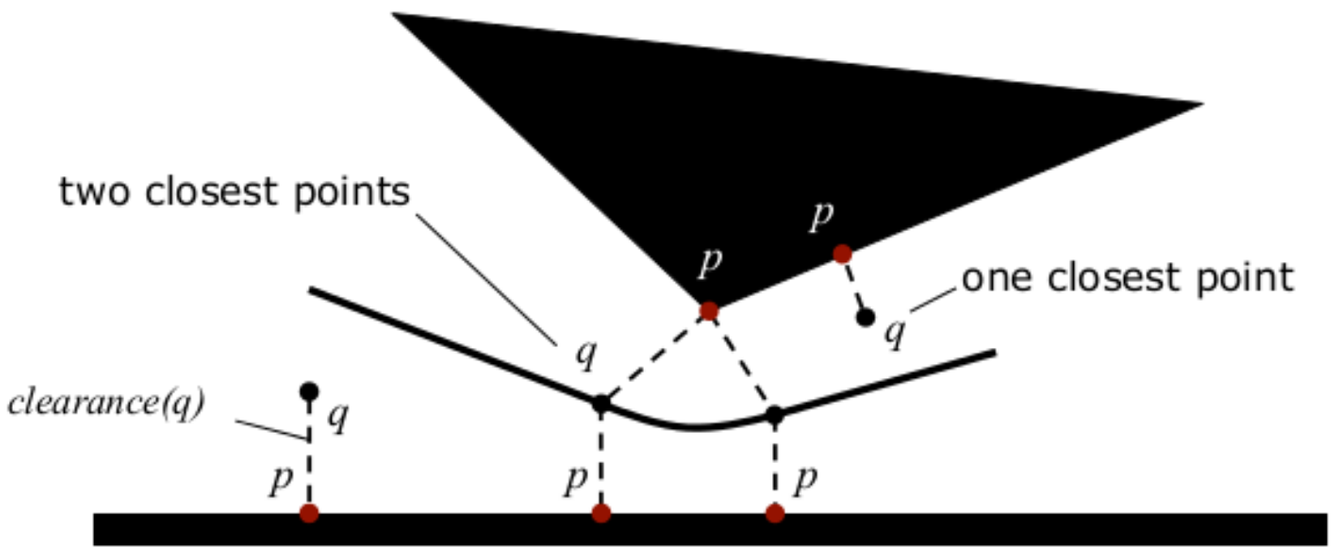
- Guarantee of shortest path but tries to stay as close as possible to obstacles, which is not precisely what we want in practice $\rightarrow$
- Any execution error will lead to a collision
- Complicated in more than 2 dimensions
- Finding a safe path (in practice) is maybe more important than strict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

GENERALIZED VORONOI DIAGRAMS

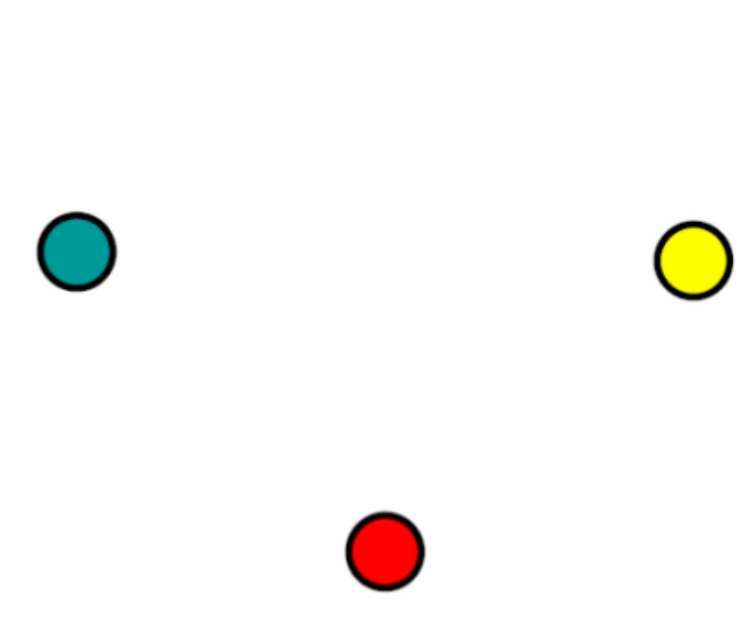
The locus of points that are equidistant from the closest two or more obstacle boundaries (in C_{obs}), including the workspace boundaries. In other words, the set of points q whose cardinality of the set of boundary points (in C_{obs}) with the same distance to q is greater than 1

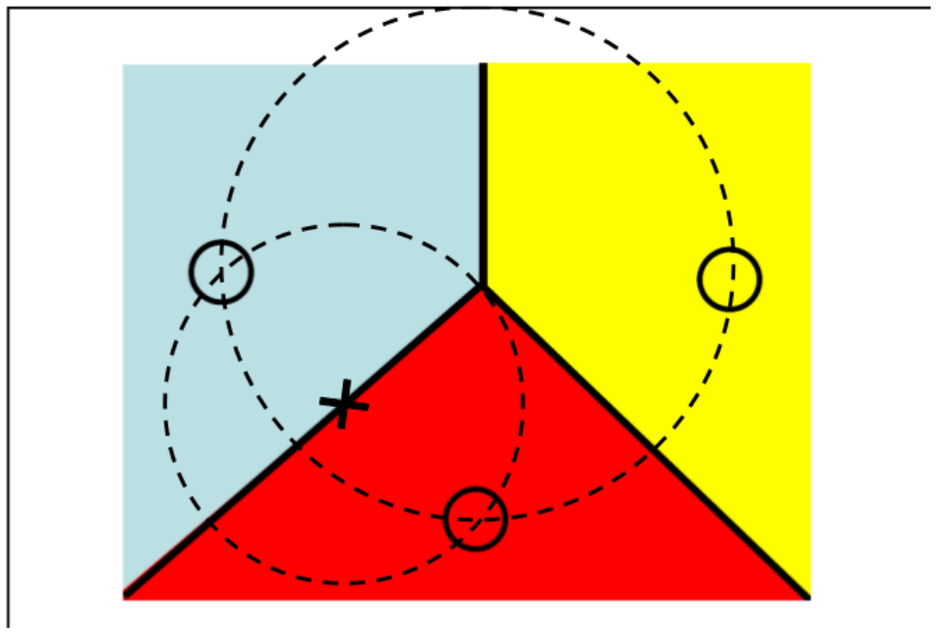
The region with the same maximal clearance from all nearest obstacles





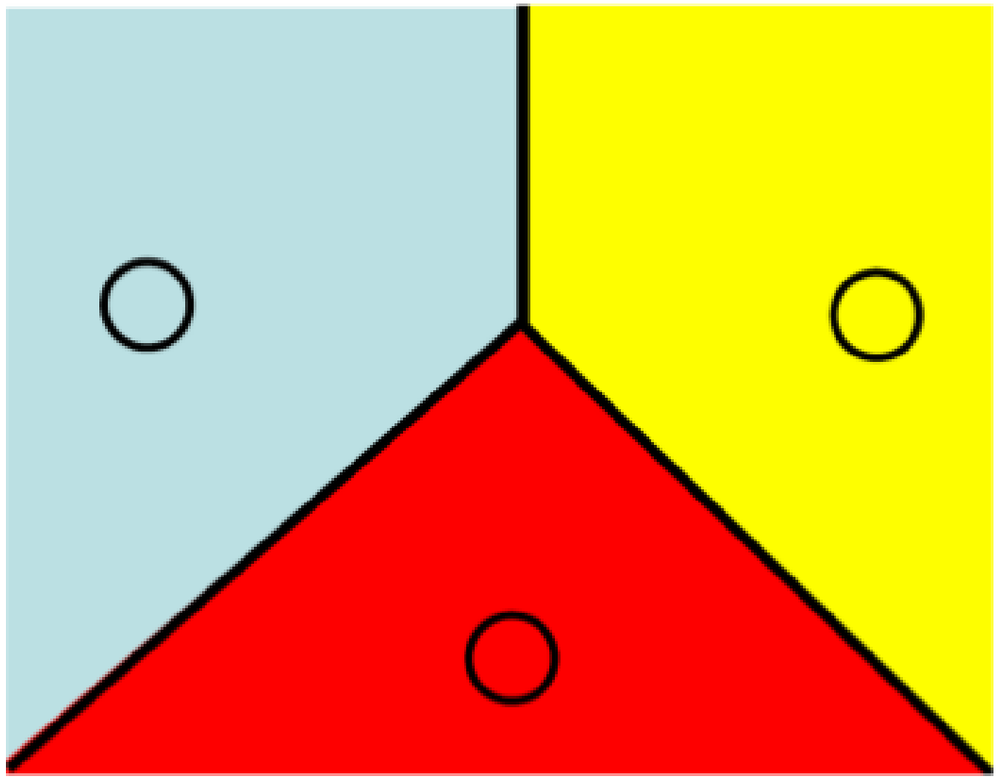
VORNONOI CELLS

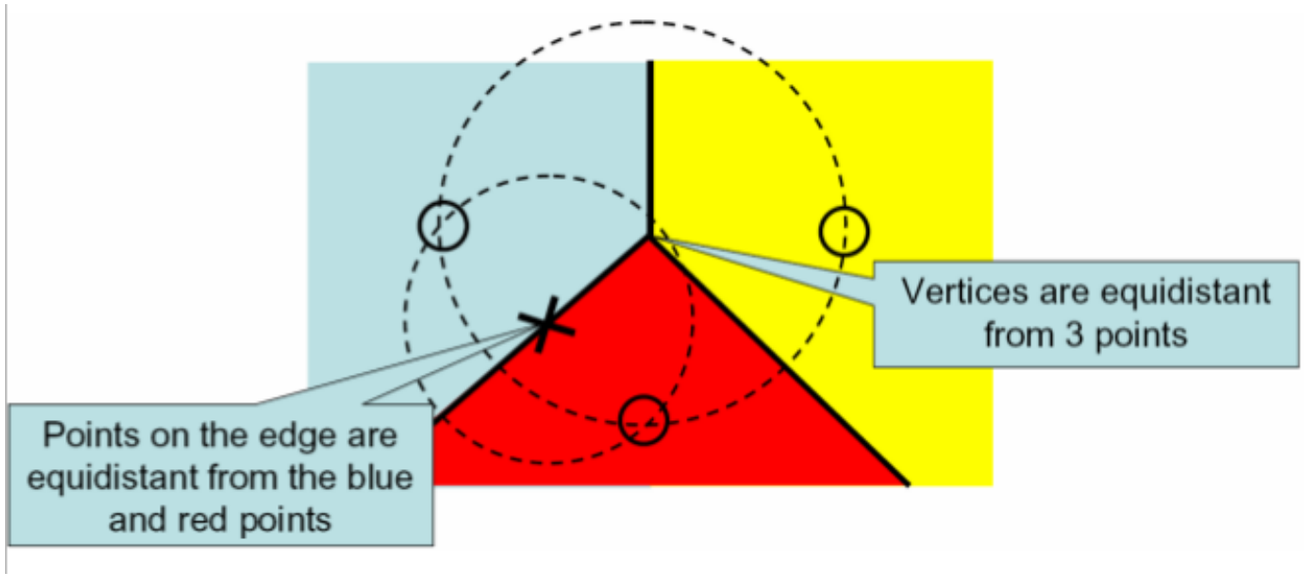




- A set of data points, the *generators*, is given
- Each generator point defines a ***Voronoi cell*** that consists of every other point whose "distance" to the generator is less than or equal to the its distance to any other generator point (distance is well defined in Euclidean spaces, but it can 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

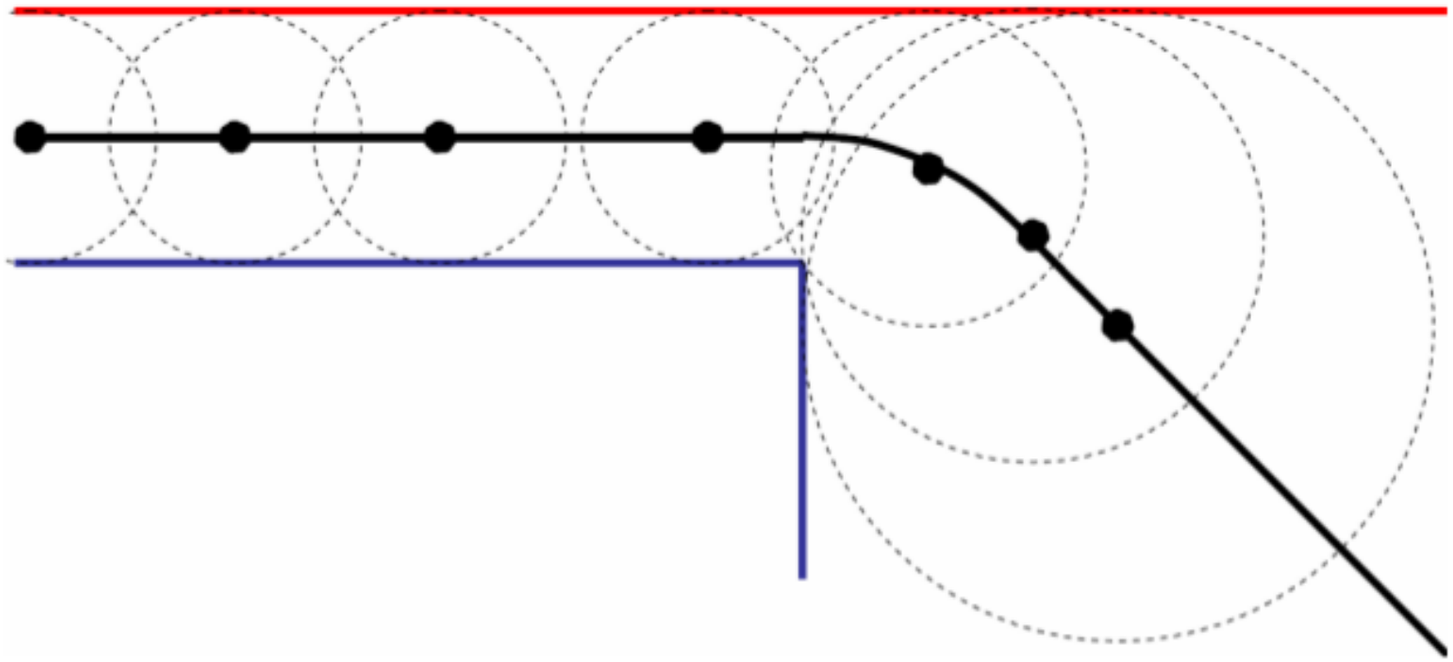
VORONOI DIAGRAMS





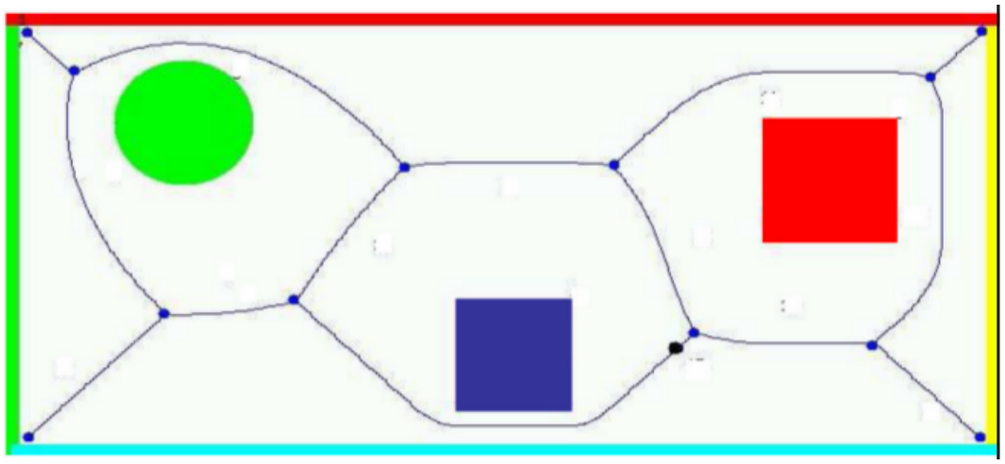
- Voronoi diagram: The set of line segments separating the regions corresponding to different colors
- Line segment: points equidistant from 2 data points
- Vertices: points equidistant from > 2 data points
- Complexity (in the plane): $\mathcal{O}(N \log N)$ in time, $\mathcal{O}(N)$ in space

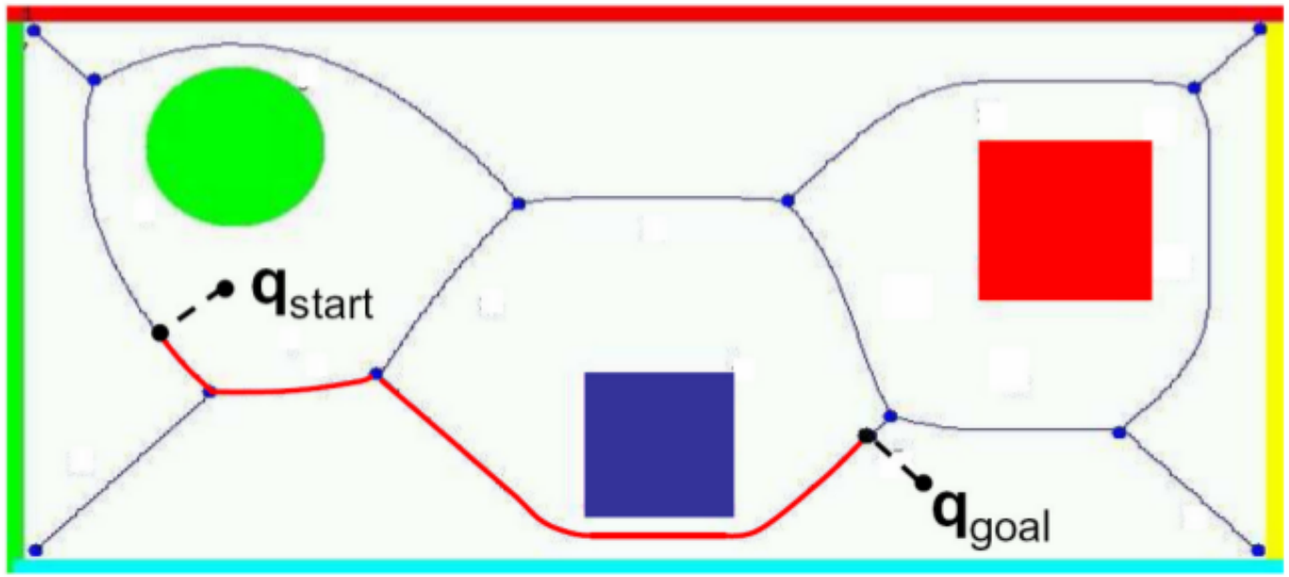
VORONOI DIAGRAMS FOR CLUTTERED ENVIRONMENTS



- 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

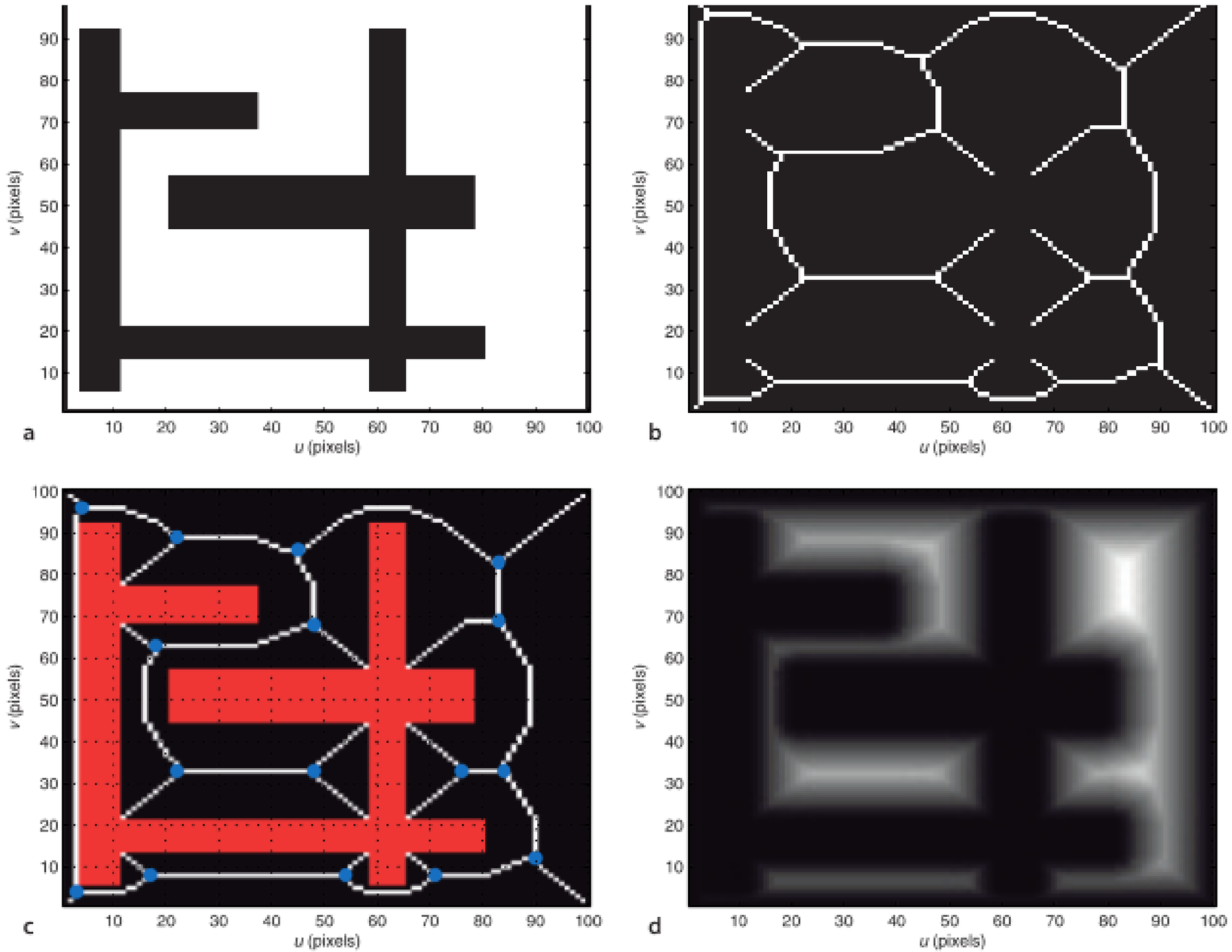
PLANNING IN VORONOI DIAGRAMS





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

PLANNING IN VORONOI GRAPHS

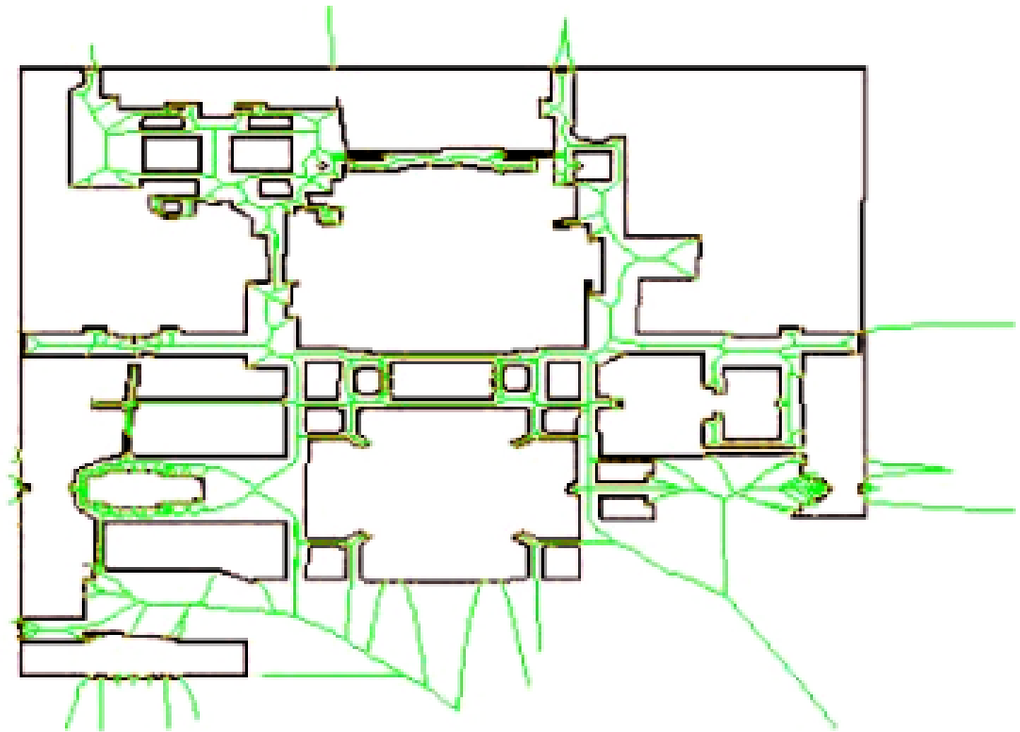


- Real environment → Voronoi skeleton → Overlapping the environment with the skeleton → A heat map for the distance from the nearest obstacle (*distance transform*)

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”) $\rightarrow$ 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.