# Autonomous and Mobile Robotics

Prof. Giuseppe Oriolo

## Motion Planning

# Introduction and Roadmap Methods

Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti

Sapienza Università di Roma

# motivation

- robots are expected to perform tasks in workspaces populated by obstacles

- autonomy requires that the robot is able to plan a collision-free motion from an initial to a final posture on the basis of geometric information

- information about the workspace geometry can be
  - entirely known in advance (off-line planning)
  - gradually discovered by the robot (on-line planning)

# the canonical problem

- robot $\mathcal{B}$ (kinematic chain with fixed or mobile base) moving in a workspace $\mathcal{W} = \mathrm{R}^N$, $N = 2$ or $3$
- $\mathcal{B}$ is free-flying in its configuration space $\mathcal{C}$, i.e., it is not subject to kinematic constraints of any kind
- obstacles $\mathcal{O}_1,\ldots,\mathcal{O}_p$ (fixed rigid objects in $\mathcal{W}$)

given a start configuration $\boldsymbol{q}_s$ and a goal configuration $\boldsymbol{q}_g$ of $\mathcal{B}$ in $\mathcal{C}$, plan a path that connects $\boldsymbol{q}_s$ to $\boldsymbol{q}_g$ and is safe, i.e., it does not cause a collision between the robot and the obstacles

- single-body robot in $\mathbb{R}^2$: piano movers' problem
  single-body robot in $\mathbb{R}^3$: generalized movers' problem

- extensions to the canonical problem:
  - moving obstacles
  - on-line planning
  - kinematic (e.g., nonholonomic) constraints
  - manipulation planning (requires contact)

- many methods that can solve the canonical problem can be appropriately modified to address one or more of these extensions

# obstacles in configuration space

- to formulate the motion planning problem, one needs to define the image of obstacles in $\mathcal{C}$

- $\mathcal{B}(\boldsymbol{q})$: the volume of $\mathcal{W}$ occupied by the robot at $\boldsymbol{q}$

- the image of obstacle $\mathcal{O}_i$ in $\mathcal{C}$ is the $i$-th $\mathcal{C}$-obstacle

$$\mathcal{CO}_i = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \mathcal{O}_i \neq \emptyset\}$$

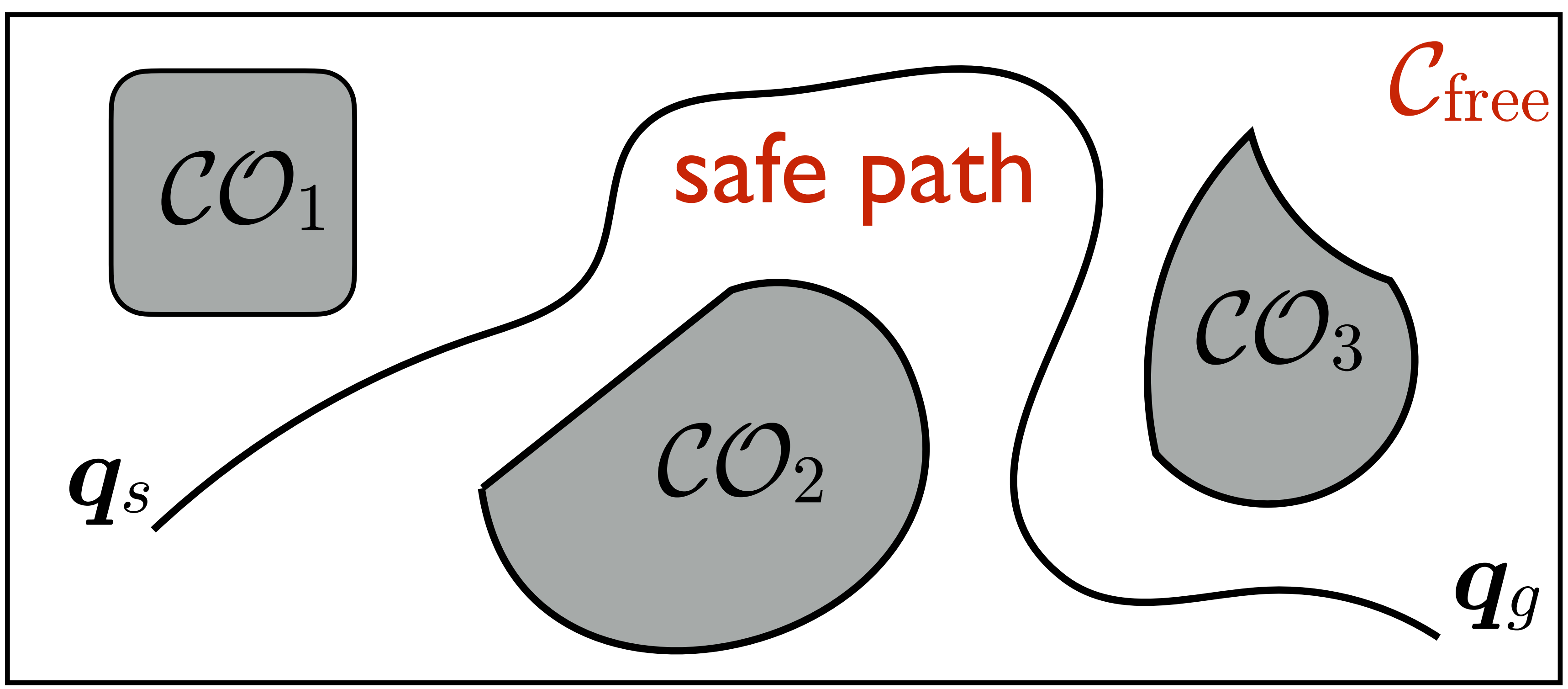
- the union of all the $\mathcal{C}$-obstacles is the $\mathcal{C}$-obstacle region

$$\mathcal{CO} = \bigcup_{i=1}^{p} \mathcal{CO}_i$$

- its complement is the free configuration space

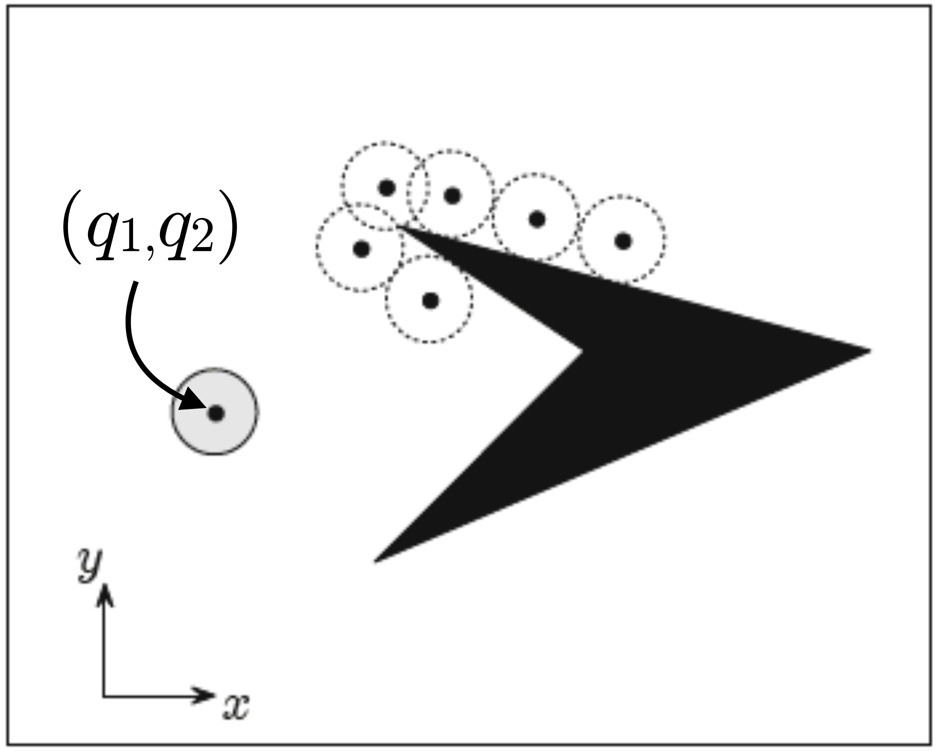
$$\mathcal{C}_{\text{free}} = \mathcal{C} - \mathcal{CO} = \{\boldsymbol{q} \in \mathcal{C} : \mathcal{B}(\boldsymbol{q}) \cap \left( \bigcup_{i=1}^{p} \mathcal{O}_i \right) = \emptyset\}$$

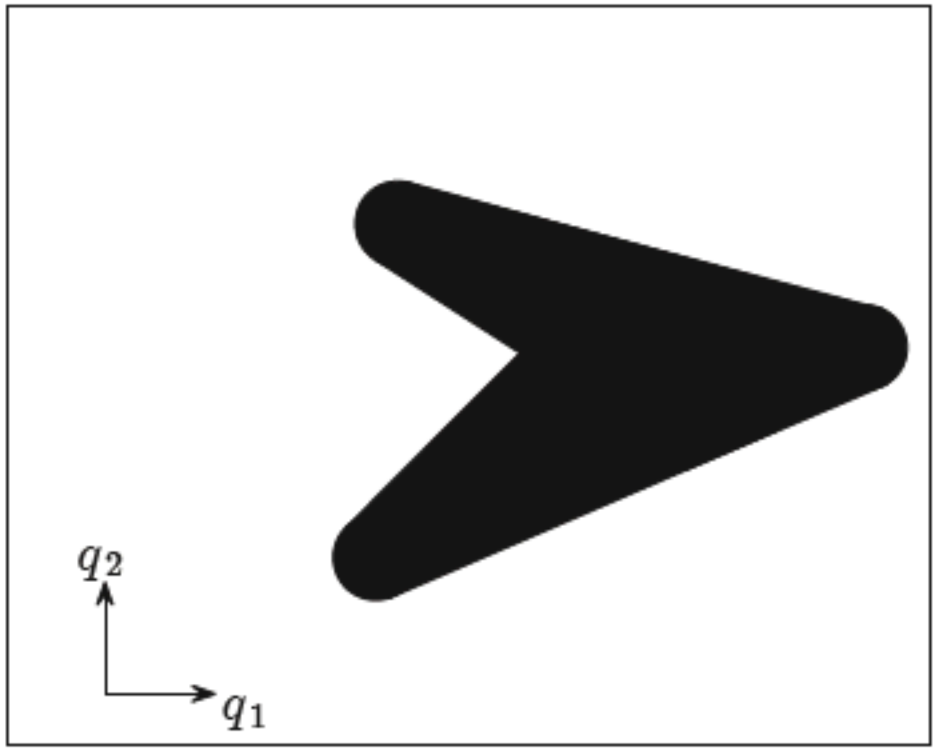
- hence, a configuration space path is safe if and only if it belongs to $\mathcal{C}_{\text{free}}$



# C-obstacles for single-body robots

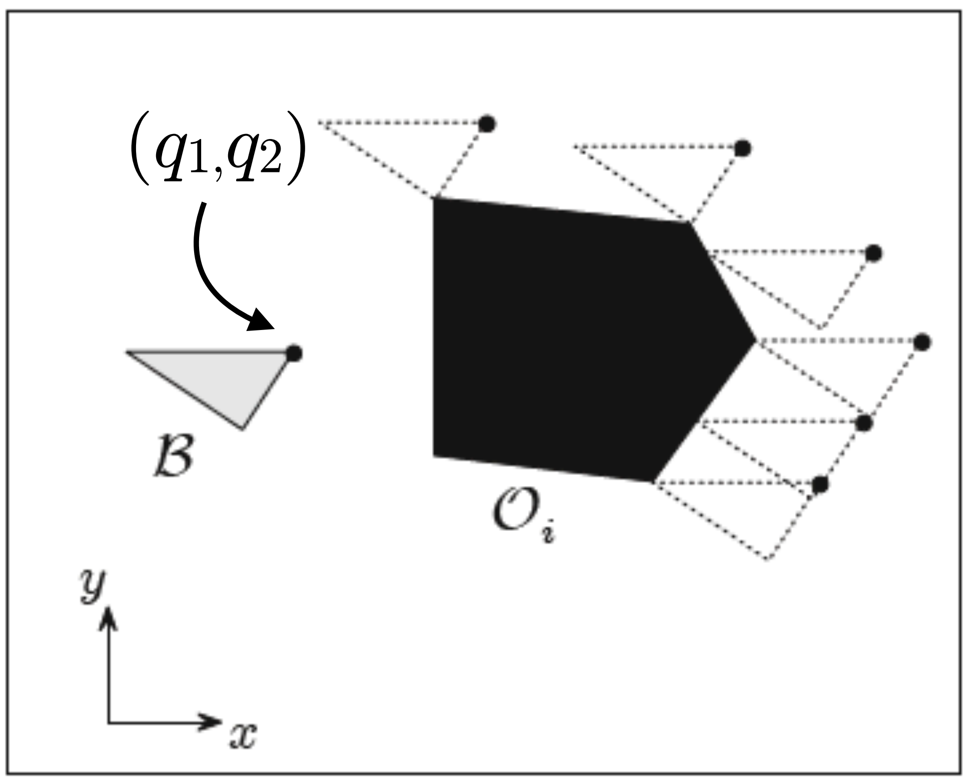
- for a point robot, $\mathcal{C}$ is a copy of $\mathcal{W}$ and $q$ are the robot Cartesian coords, so $\mathcal{C}$-obstacles are copies of obstacles
- for a disk robot free to translate (and rotate) in $\mathrm{R}^2$, $\mathcal{C}$ is a copy of $\mathcal{W}$ and $q$ are the center Cartesian coords, so the $\mathcal{C}$-obstacle grows isotropically w.r.t the obstacle

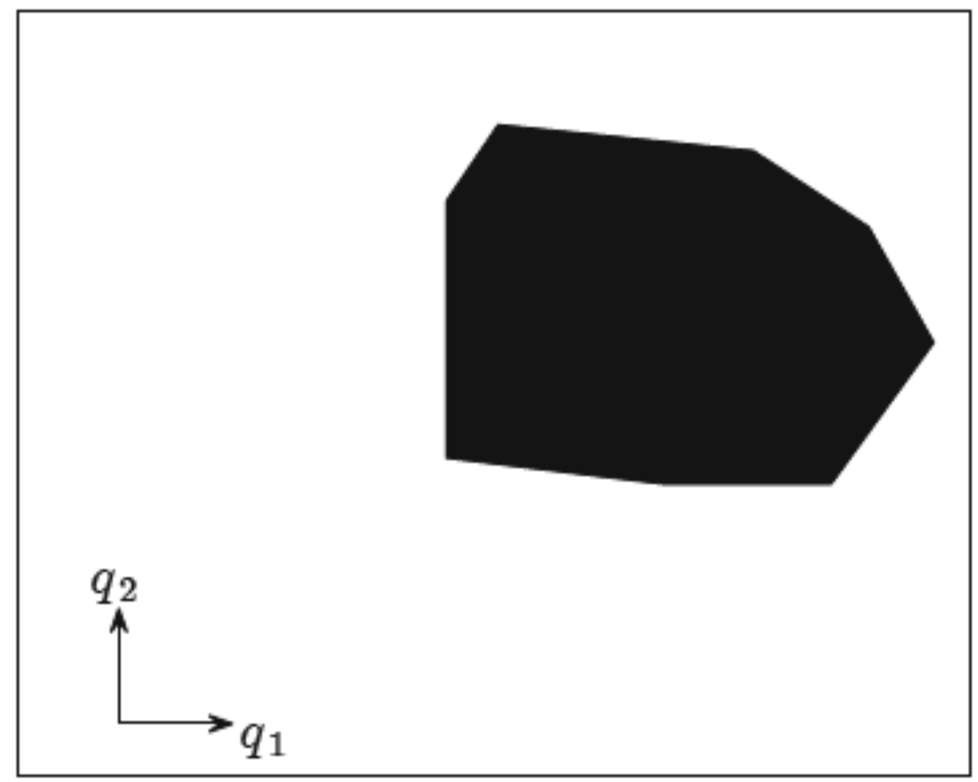


# C-obstacles for a single-body robots

- for a polygonal robot free to translate (with fixed orientation) in $\mathbb{R}^2$, $\mathcal{C}$ is a copy of $\mathcal{W}$ and $q$ are the Cartesian coords of (any) representative point

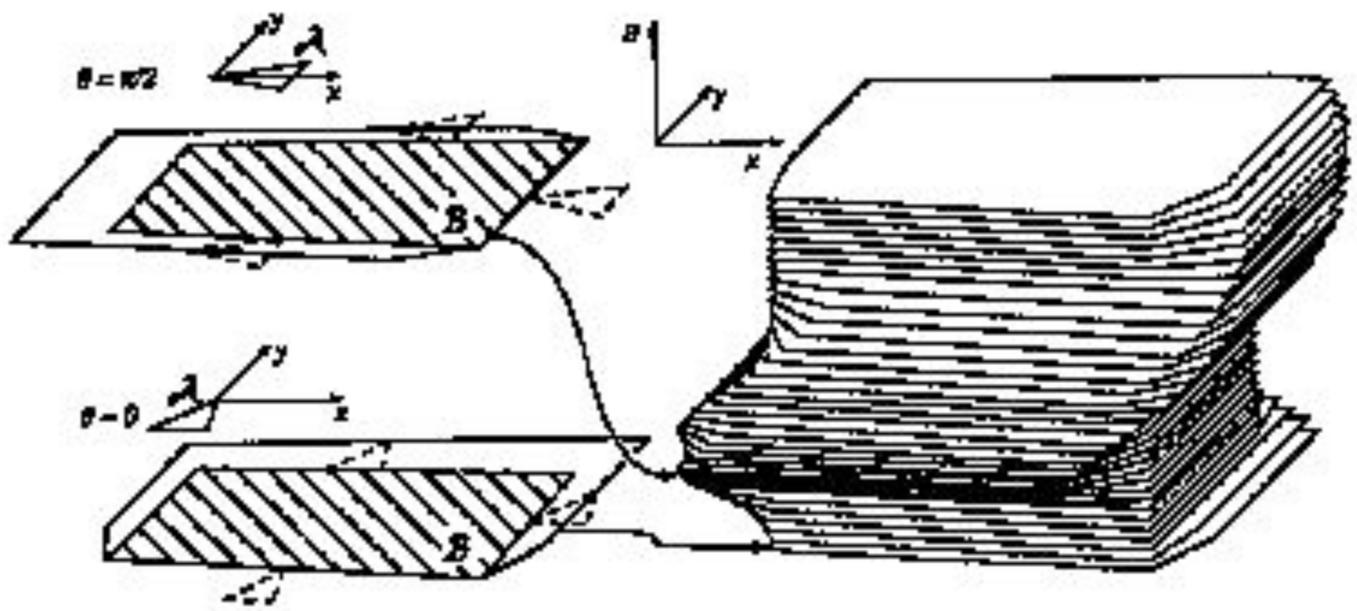





the $\mathcal{C}$-obstacle grows anisotropically w.r.t the obstacle

# C-obstacles for single-body robots

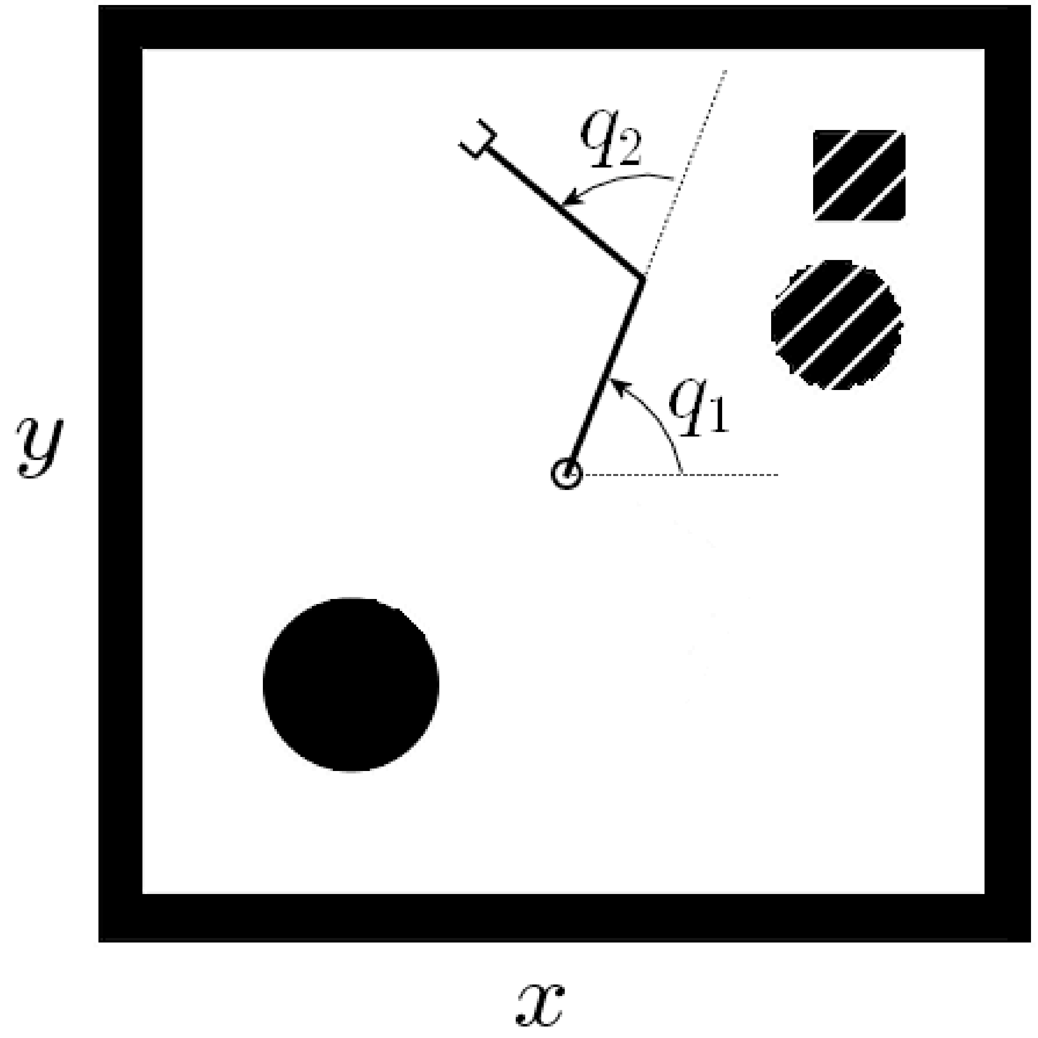
- for a polygonal robot free to translate and rotate on the plane

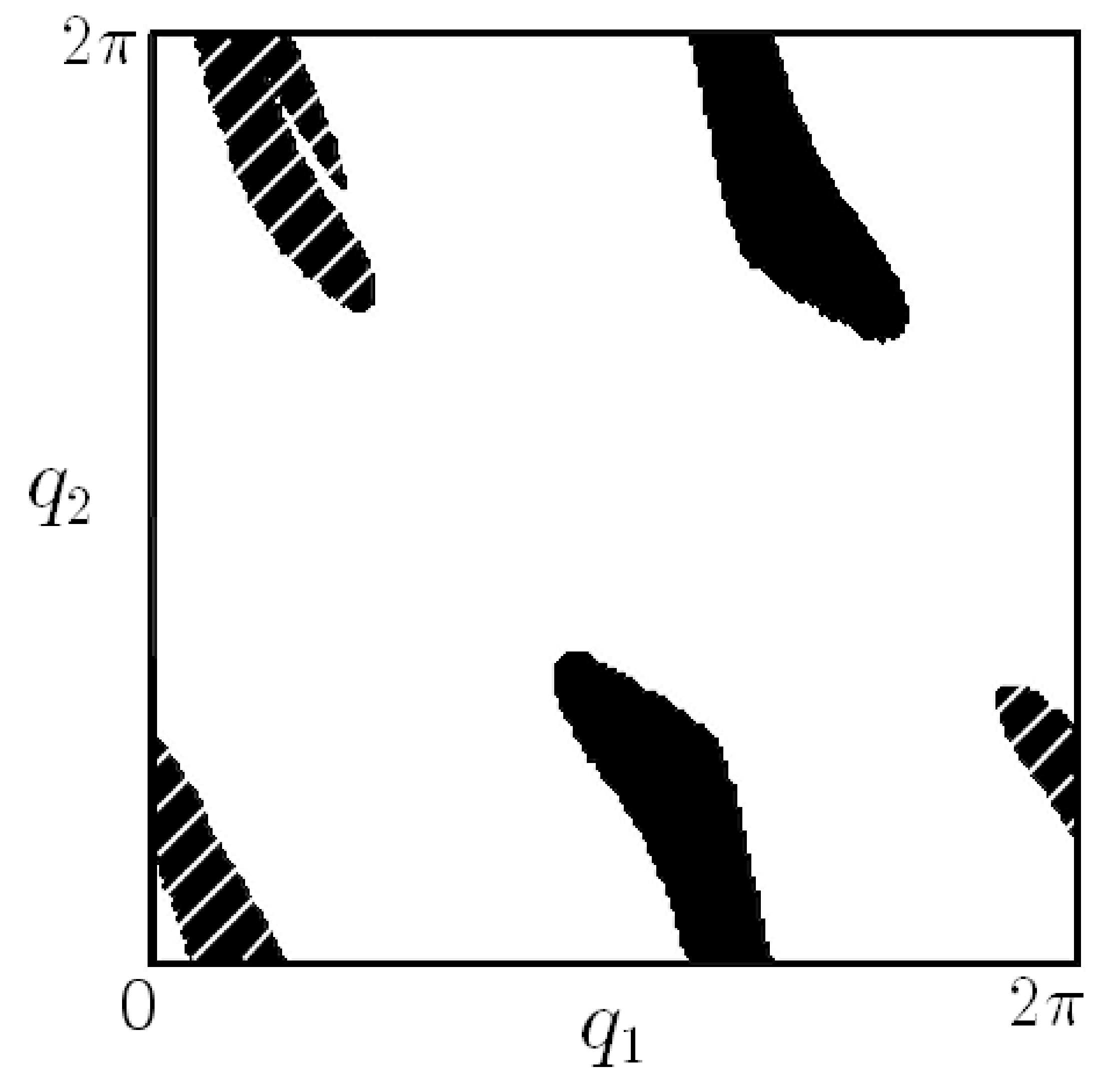


the $\mathcal{C}$-obstacle is obtained from the original obstacle via a "grow and stack" procedure

# C-obstacles for manipulators

for a 2R planar manipulator, scene 1

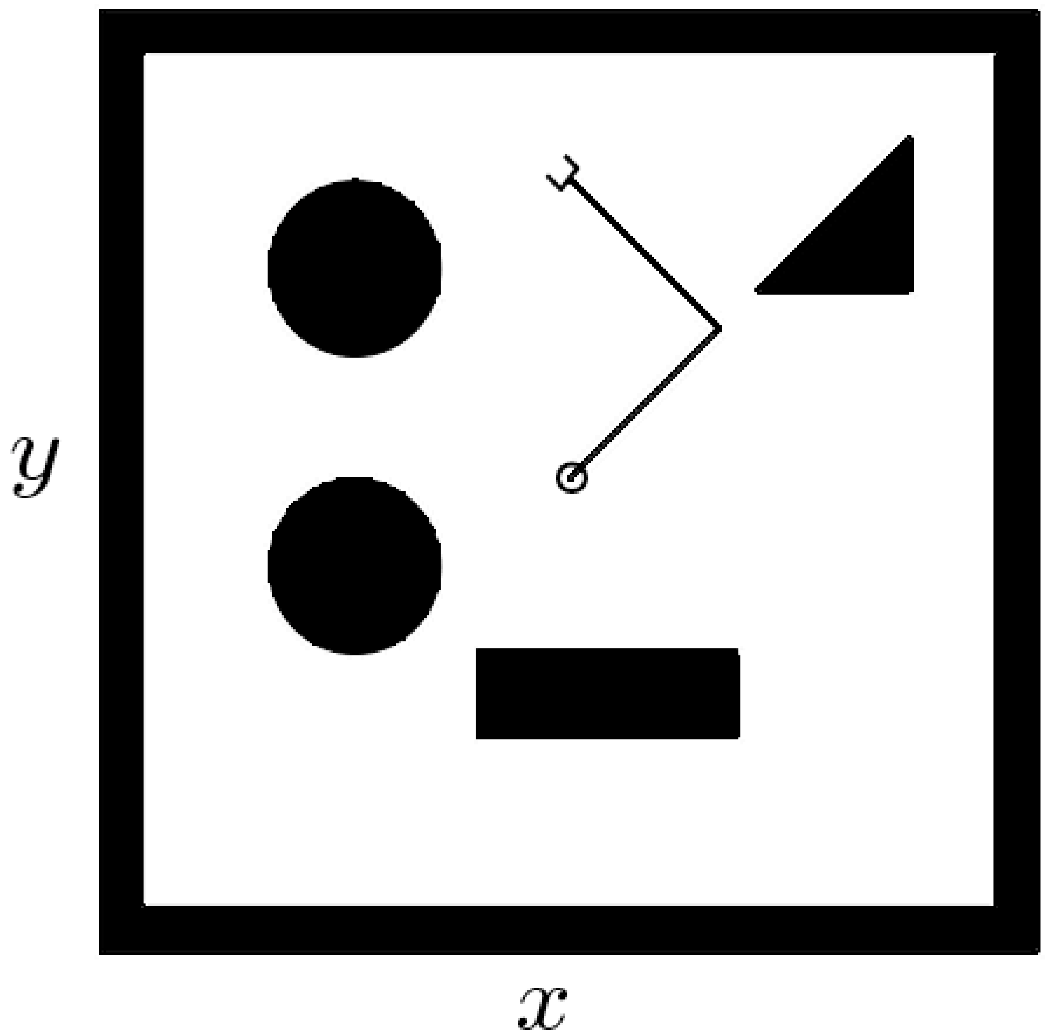


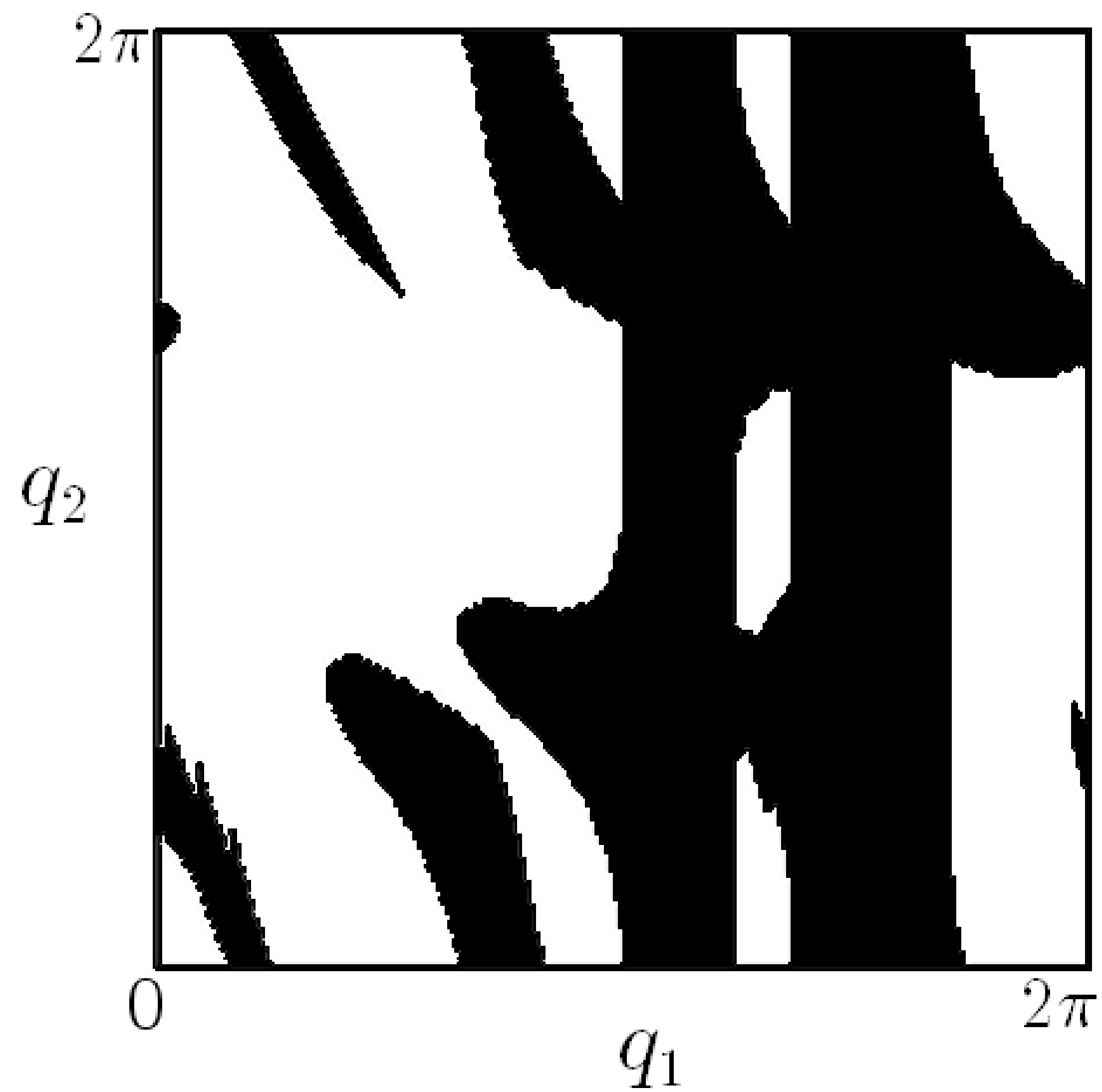




disjoint workspace obstacles may merge in $\mathcal{C}$

# C-obstacles for manipulators

for a 2R planar manipulator, scene 2




the free configuration space may be disconnected

# motion planning methods

- all work in the configuration space $\mathcal{C}$
- many need preliminary computation of the $\mathcal{C}$-obstacle region $\mathcal{CO}$, a highly expensive procedure (complexity is exponential in dim $\mathcal{C}$)
- computation of $\mathcal{CO}$ is computationally heavy:

  exact: requires an algebraic model of $\mathcal{O}_1,\ldots,\mathcal{O}_p$

  approximate: e.g., sample $\mathcal{C}$ using a regular grid, compute the volume occupied by the robot at each sample, and check for collisions between this volume and the obstacles

- on the other hand, checking if a single configuration is in collision is fast; efficient collision-checking algorithms exist, such as V-collide in $\mathrm{R}^2$ and I-collide in $\mathrm{R}^3$

## classification

1. roadmap methods

   represent the connectivity of $\mathcal{C}_{\rm free}$ by a sufficiently rich network of safe paths

   e.g., retraction, cell decomposition

2. probabilistic methods

   a particular instance of sampling-based methods where samples of $\mathcal{C}$ are randomly extracted

   e.g., PRM, RRT

3. artificial potential field methods

   a heuristic approach which is particularly suitable for on-line planning

# retraction method

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\mathrm{free}}$ a polygonal limited subset (its boundary $\partial\mathcal{C}_{\mathrm{free}}$ is entirely made of line segments)

- define the clearance of a configuration $\boldsymbol{q}$ in $\mathcal{C}_{\mathrm{free}}$ as

$$\gamma(\boldsymbol{q}) = \min_{\boldsymbol{s} \in \partial\mathcal{C}_{\mathrm{free}}} \|\boldsymbol{q} - \boldsymbol{s}\|$$

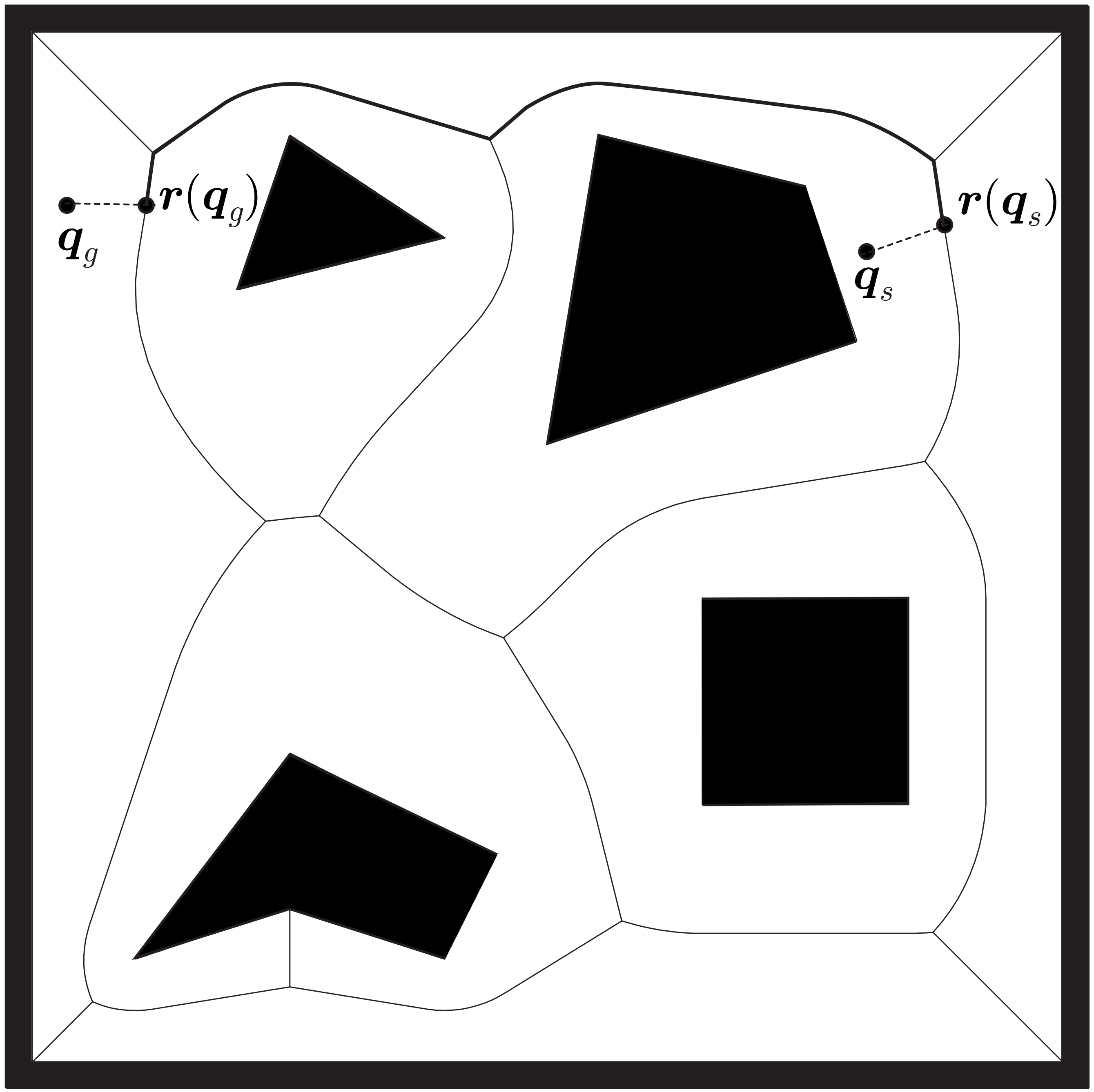
- define the neighbors of $\boldsymbol{q}$ as

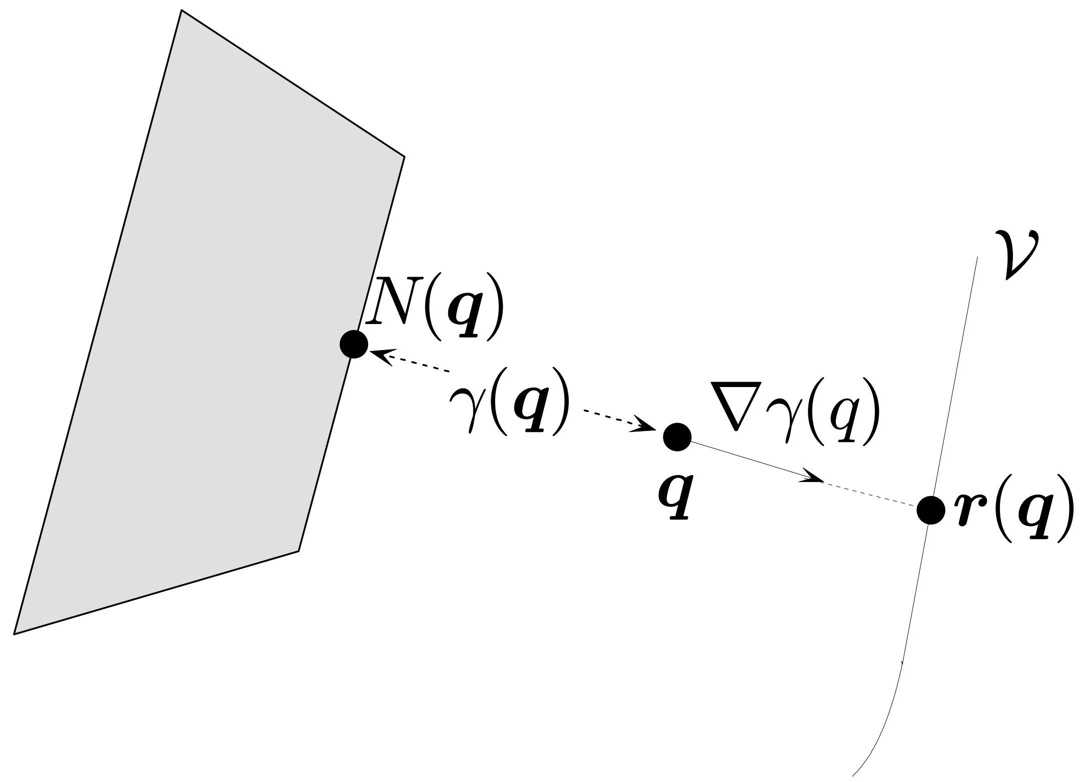
$$N(\boldsymbol{q}) = \{\boldsymbol{s} \in \partial\mathcal{C}_{\mathrm{free}} : \|\boldsymbol{q} - \boldsymbol{s}\| = \gamma(\boldsymbol{q})\}$$

- the generalized Voronoi diagram of $\mathcal{C}_{\mathrm{free}}$ is

$$\mathcal{V}(\mathcal{C}_{\mathrm{free}}) = \{\boldsymbol{q} \in \mathcal{C}_{\mathrm{free}} : \mathrm{card}(N(\boldsymbol{q})) > 1\}$$

- its elementary arcs are
  - rectilinear (edge-edge, vertex-vertex)
  - parabolic (edge-vertex)
- can be seen as a graph
  - elementary arcs as arcs
  - arc endpoints as nodes
- a natural roadmap as it maximizes safety






- to connect any $\boldsymbol{q}$ to $\mathcal{V}(\mathcal{C}_{\text{free}})$, use retraction: from $\boldsymbol{q}$, follow $\nabla\gamma$ up to the first intersection $\boldsymbol{r}(\boldsymbol{q})$ with $\mathcal{V}(\mathcal{C}_{\text{free}})$

- $\boldsymbol{r}(\cdot)$ preserves the connectivity of $\mathcal{C}_{\text{free}}$, i.e., $\boldsymbol{q}$ and $\boldsymbol{r}(\boldsymbol{q})$ lie in the same connected component of $\mathcal{C}_{\text{free}}$

- hence, a safe path exists between $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ if and only if a path exists on $\mathcal{V}(\mathcal{C}_{\text{free}})$ between $\boldsymbol{r}(\boldsymbol{q}_s)$ and $\boldsymbol{r}(\boldsymbol{q}_g)$

# algorithm

1. build the generalized Voronoi diagram $\mathcal{V}(\mathcal{C}_{\text{free}})$
2. compute the retractions $\boldsymbol{r}(\boldsymbol{q}_s)$ and $\boldsymbol{r}(\boldsymbol{q}_g)$
3. search $\mathcal{V}(\mathcal{C}_{\text{free}})$ for a sequence of arcs such that $\boldsymbol{r}(\boldsymbol{q}_s)$ belongs to the first and $\boldsymbol{r}(\boldsymbol{q}_g)$ to the last
4. if successful, return the solution path consisting of
    a. line segment from $\boldsymbol{q}_s$ to $\boldsymbol{r}(\boldsymbol{q}_s)$
    b. portion of first arc from $\boldsymbol{r}(\boldsymbol{q}_s)$ to its end
    c. second, third, ... , penultimate arc
    d. portion of last arc from its start to $\boldsymbol{r}(\boldsymbol{q}_g)$
    e. line segment from $\boldsymbol{r}(\boldsymbol{q}_g)$ to $\boldsymbol{q}_g$

   otherwise, report a failure

- graph search at step 3: if a minimum-length path is desired, label each arc with a cost equal to its length, and use $A^\star$ to compute a minimum-cost solution
- the retraction method is complete, i.e., finds a solution when one exists and reports failure otherwise; and multiple-query, as one can build $\mathcal{V}(\mathcal{C}_{\text{free}})$ once for all
- complexity: if $\mathcal{C}_{\text{free}}$ has $v$ vertices, $\mathcal{V}(\mathcal{C}_{\text{free}})$ has $O(v)$ arcs
    - step 1: $O(v\log v)$
    - step 2: $O(v)$
    - step 3: $O(v\log v)$ $(A^\star$ on a graph with $O(v)$ arcs)

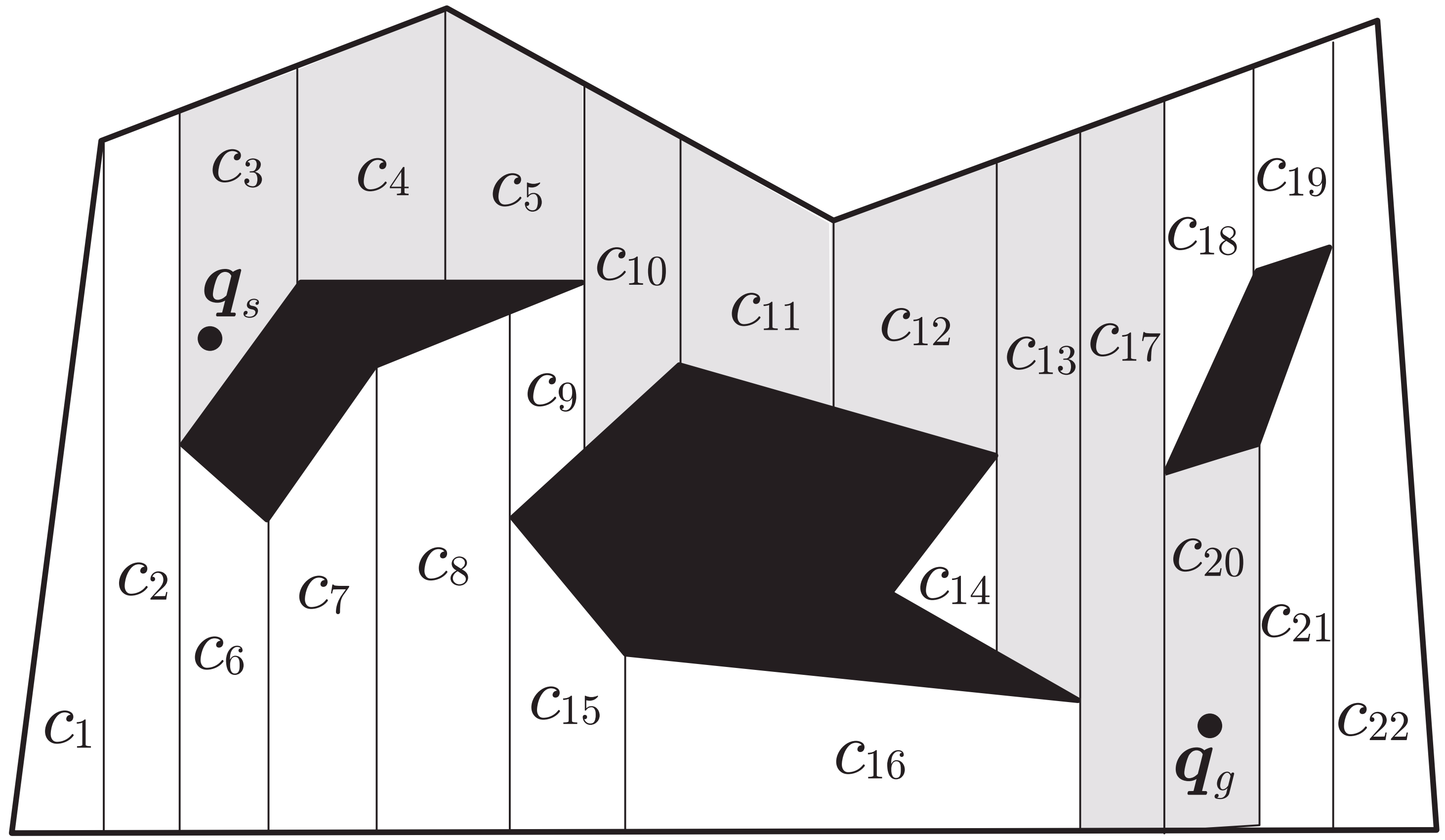
  altogether, the time complexity is $O(v\log v)$
- extensions (e.g., to higher-dimensional configuration spaces) are possible but quite complicated

# cell decomposition methods

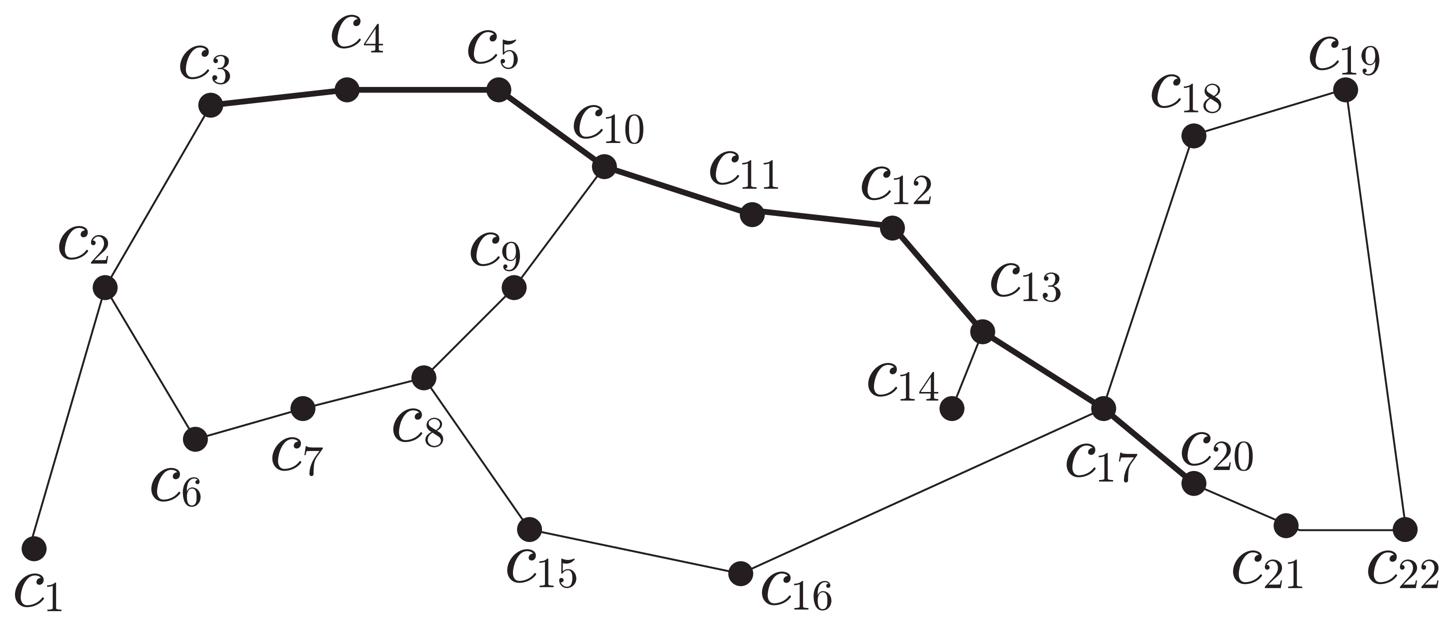
- idea: decompose $\mathcal{C}_{\text{free}}$ in cells, i.e., regions such that
  - it is easy to compute a safe path between two configurations in the same cell
  - it is easy to compute a safe path between two configurations in adjacent cells
- once a cell decomposition of $\mathcal{C}_{\text{free}}$ is computed, find a sequence of cells (channel) with $\boldsymbol{q}_s$ in the first and $\boldsymbol{q}_g$ in the last
- different methods are obtained depending on the type of cells used for the decomposition

# exact decomposition

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\mathrm{free}}$ a polygonal limited subset
- variable-shape cells are needed to decompose exactly $\mathcal{C}_{\mathrm{free}}$; a typical choice are convex polygons
- convexity guarantees that it is easy to plan in a cell and between adjacent cells
- the sweep-line algorithm can be used to compute a decomposition of $\mathcal{C}_{\mathrm{free}}$ into convex polygons




- sweep a line over $\mathcal{C}_{\text{free}}$; when it goes through a vertex, two segments (extensions) originate at the vertex
- an extension lying in $\mathcal{C}_{\text{free}}$ is part of the boundary of a cell; the rest are other extensions and/or parts of $\partial\mathcal{C}_{\text{free}}$
- the result is a trapezoidal decomposition




- build the associated connectivity graph $C$
- identify nodes (cells) $c_s$ and $c_g$ where $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ are
- use graph search to find a path on $C$ from $c_s$ to $c_g$; this represents a channel of cells
- extract from the channel a safe solution path, e.g., joining $\boldsymbol{q}_s$ to $\boldsymbol{q}_g$ via midpoints of common boundaries

## algorithm

1. compute a convex polygonal decomposition of $\mathcal{C}_{\text{free}}$
2. build the associated connectivity graph $C$
3. search $C$ for a channel of cells from $c_s$ to $c_g$
4. if successful, extract and return a solution path consisting of
   a. line segment from $\boldsymbol{q}_s$ to the midpoint of the common boundary between the first two cells
   b. line segments between the midpoints of consecutive cells
   c. line segment from the midpoint of the common boundary between the last two cells and $\boldsymbol{q}_g$

   otherwise, report a failure

- if a minimum-length channel is desired, define a modified connectivity graph with $\boldsymbol{q}_s$, $\boldsymbol{q}_g$ and all the midpoints as nodes, and line segments between nodes in the same cell as arcs, each with a cost equal to its length, and use $A^\star$ to compute a minimum-cost path
- the exact cell decomposition method is complete and multiple-query, as one can build the connectivity graph once for all
- complexity: if $\mathcal{C}_{\text{free}}$ has $v$ vertices, $C$ has $O(v)$ arcs
  - step 1: $O(v\log v)$
  - step 2: $O(v)$
  - step 3: $O(v\log v)$ $(A^\star$ on a graph with $O(v)$ arcs)

altogether, the time complexity is $O(v\log v)$

- a channel is more flexible than a roadmap because it contains an infinity of paths; this may be exploited to take into account nonholonomic constraints or to avoid unexpected obstacles during the motion

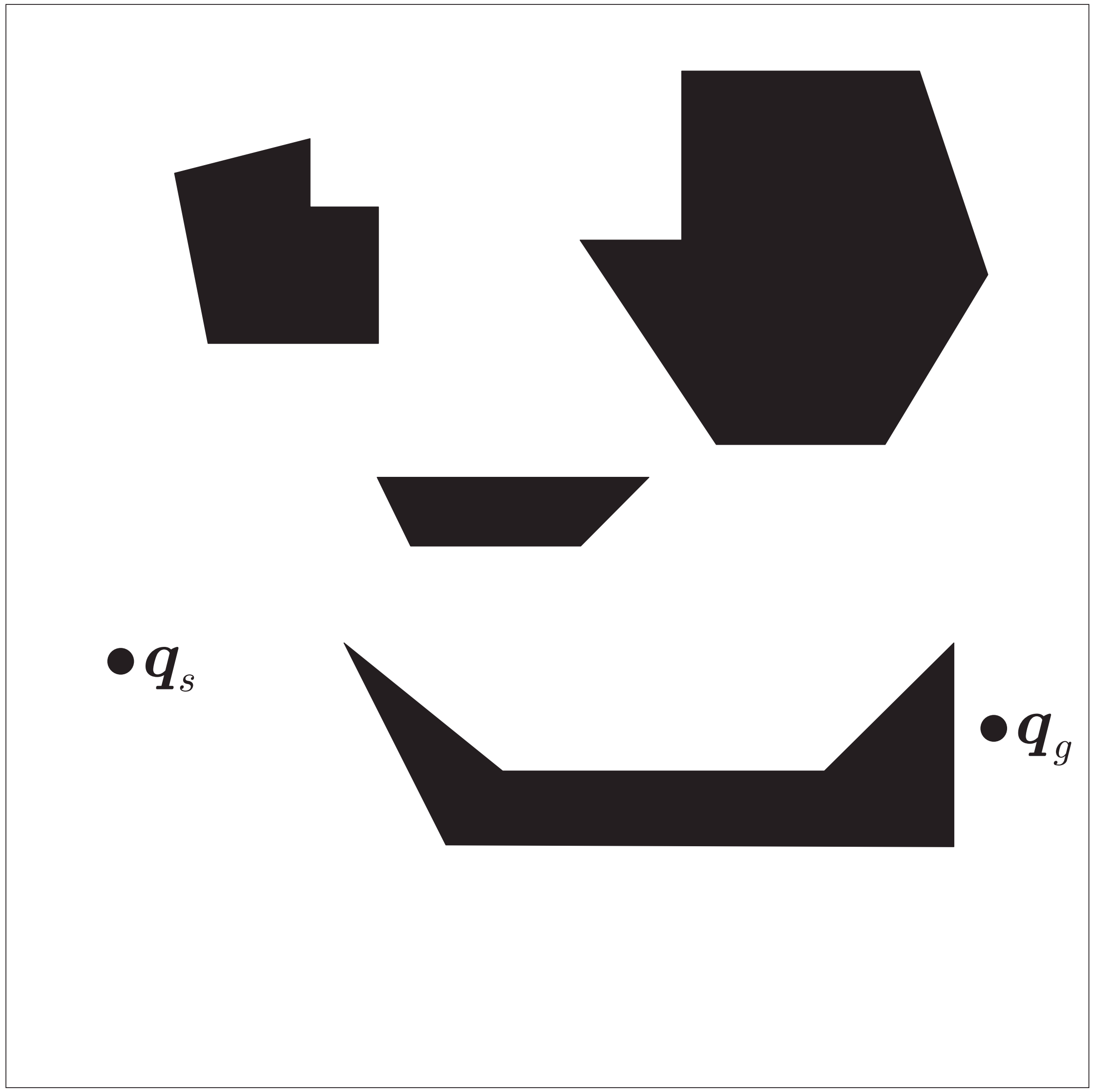
- the solution path is a broken line, but smoothing may be performed in a post-processing phase

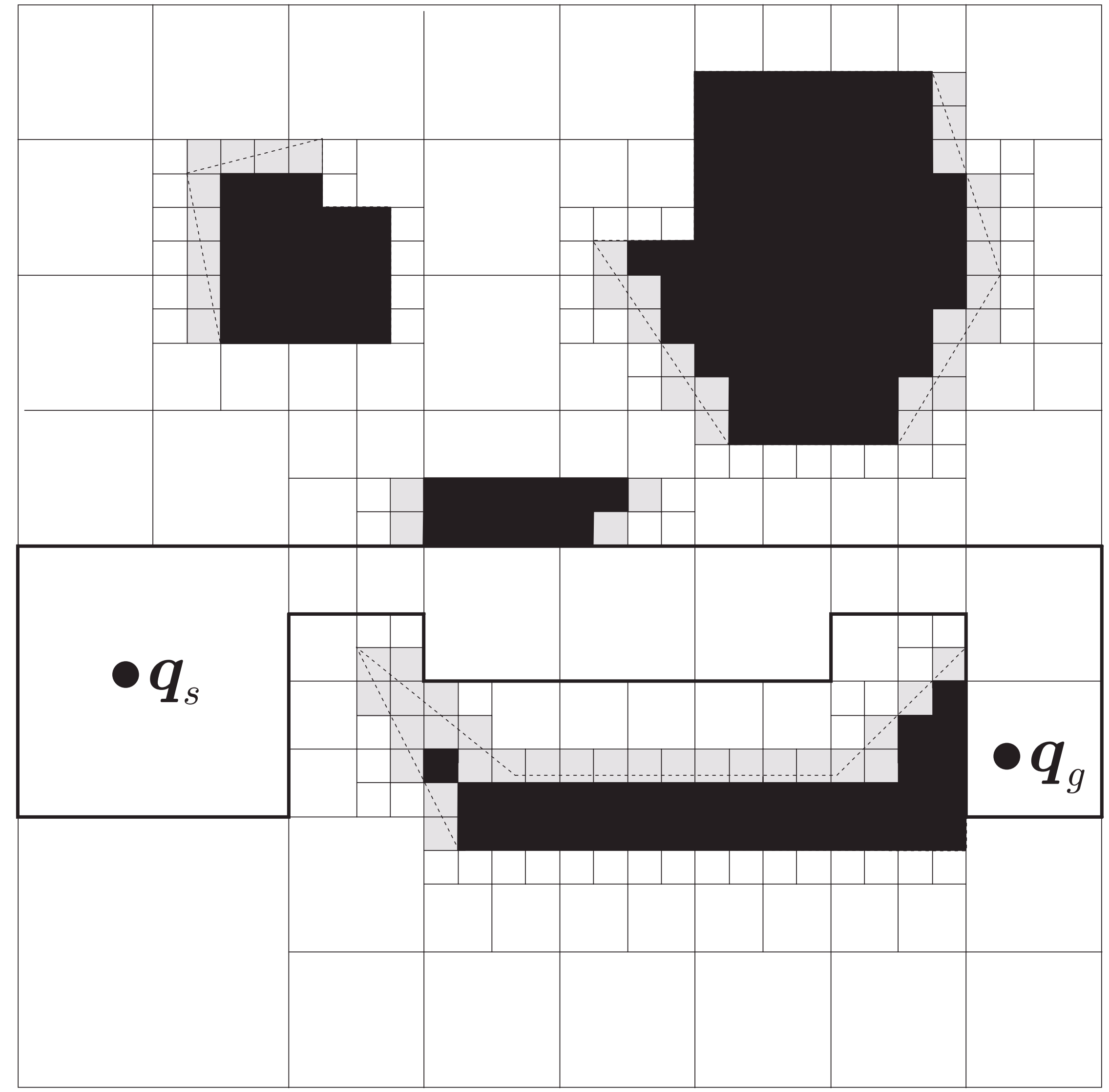
- if $\mathcal{C} = \mathrm{R}^3$ and $\mathcal{C}_{\mathrm{free}}$ is a polyhedral limited subset, the sweep-plane algorithm may be used to compute a decomposition of $\mathcal{C}_{\mathrm{free}}$ into convex polyhedra

- an extension to configuration spaces of arbitrary dimension exists but it is very inefficient: in fact, complexity is exponential in the dimension of $\mathcal{C}$

# approximate decomposition

- assume $\mathcal{C} = \mathrm{R}^2$ and $\mathcal{C}_{\mathrm{free}}$ a polygonal limited subset
- fixed-shape cells are used to obtain an approximate decomposition (by defect) of $\mathcal{C}_{\mathrm{free}}$; e.g., squares
- as in exact decomposition, convexity guarantees that it is easy to plan in a cell and between adjacent cells
- a recursive algorithm is used for decomposition to reach a trade-off between simplicity and accuracy




- start by dividing $\mathcal{C}$ in 4 cells and classifying each cell as
  - free, if its interior is completely in $\mathcal{C}_{\rm free}$
  - occupied, if it is completely in $\mathcal{CO}$
  - mixed, if it is neither free nor occupied

- build the connectivity graph $C$ associated to the current level of decomposition, with free and mixed cells as nodes and arcs between adjacent nodes
- identify nodes (cells) $c_s$ and $c_g$ where $\boldsymbol{q}_s$ and $\boldsymbol{q}_g$ are, and use graph search to look for a path (channel) on $C$ from $c_s$ to $c_g$; if it does not exist, report failure
- if a path (channel) exists on $C$ from $c_s$ to $c_g$, take any mixed cell in the path and decompose it as before
- repeat the above steps (build $C$, look for a path and decompose mixed cells) until a path is found made only of free cells, or until a minimum size has been reached for the cells; in the latter case, backtrack

- the above planning method is
  - resolution complete, in the sense that a solution is found if and only if one exists at the maximum allowed resolution
  - single-query, because the decomposition is guided by the given $\boldsymbol{q}_s$, $\boldsymbol{q}_g$
- recursive decomposition of cells can be implemented efficiently using quadtrees (trees whose internal nodes have exactly four children)
- the method is conceptually applicable to configuration spaces of arbitrary dimension: however, complexity is still exponential in the dimension of $\mathcal{C}$