اصول علم ربات – اسلاید هفدهم

Fundamentals of Robotics – Slide 17

Motion Planning

دکتر مهدی جوانمردی

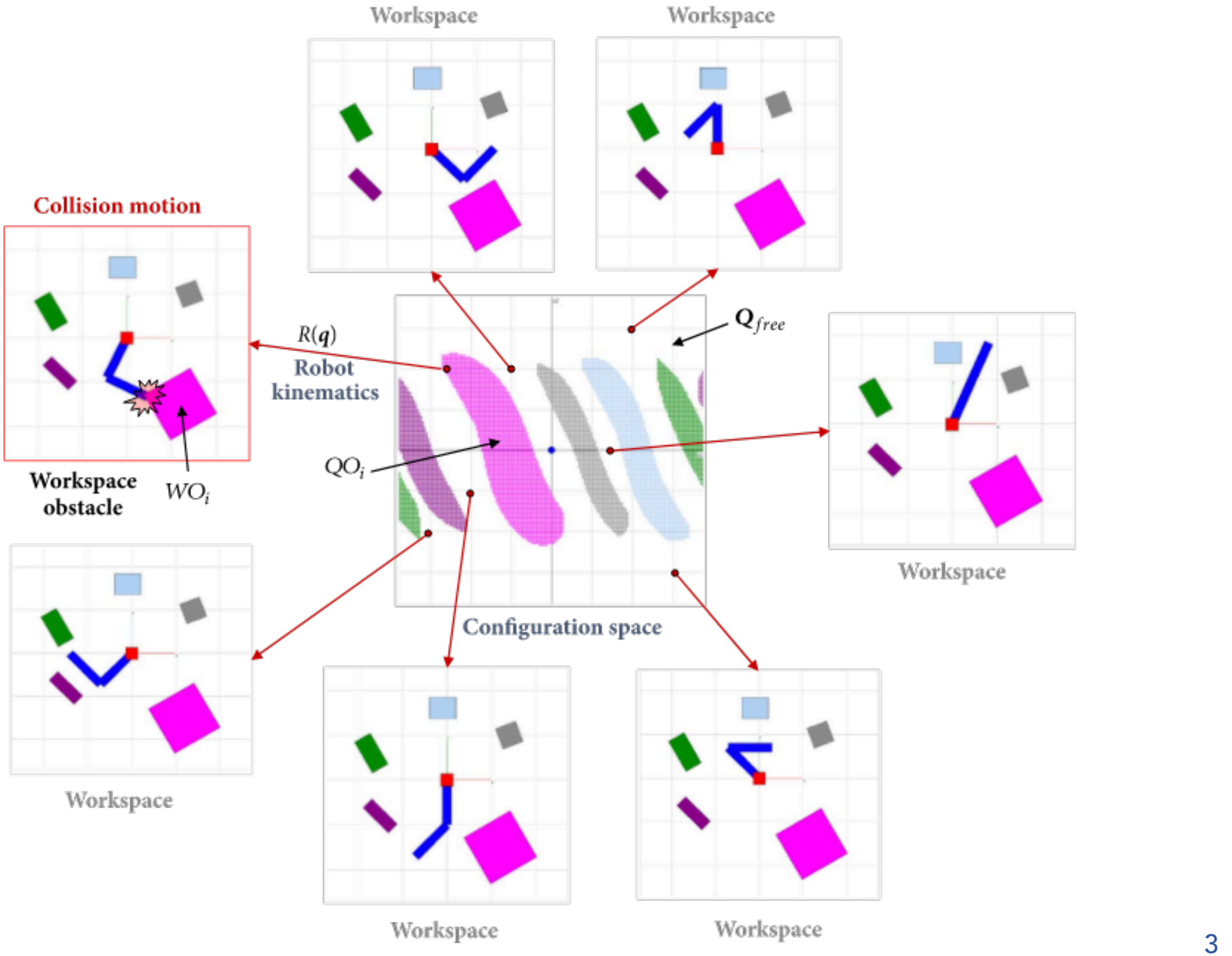
زمستان ۱۴۰۰ – بهار ۱۴۰۱

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

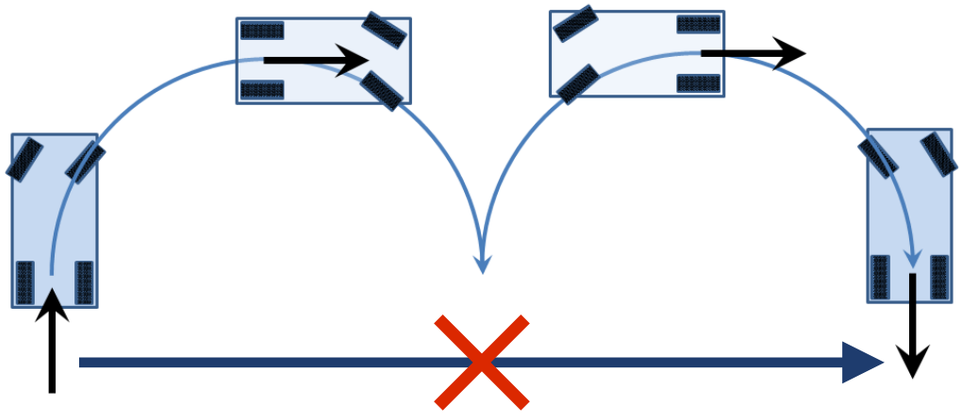
Motion planning: Core Challenges

- 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

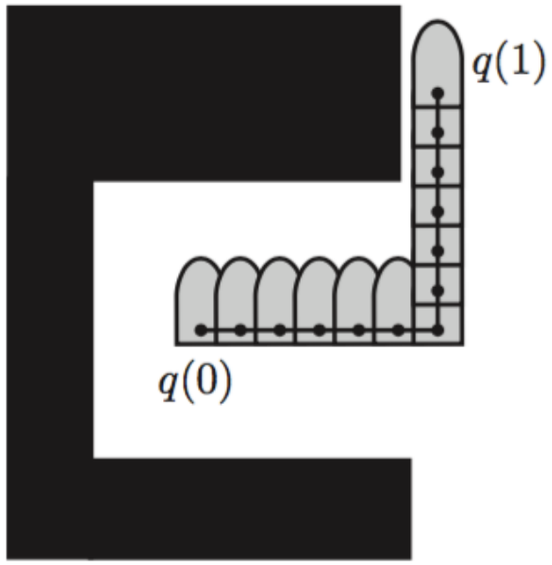
Complexity of C-space



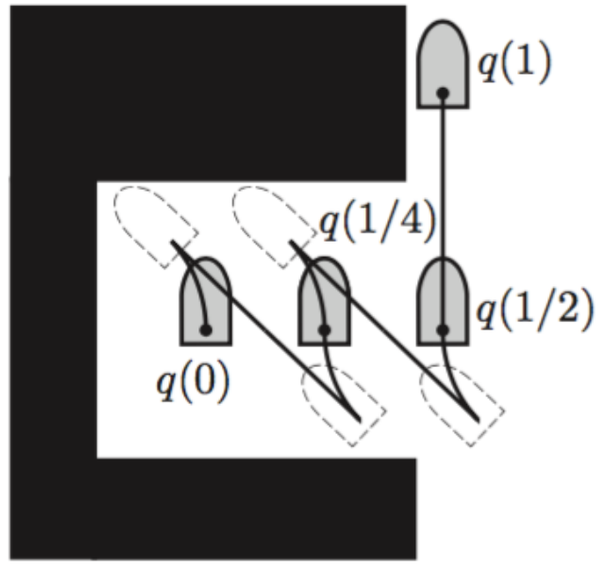
Non-holonomic path planning



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

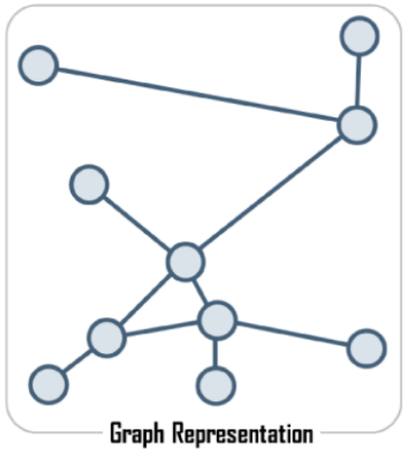
Non Holonomic path planning is a difficult problem!

Let's start with holonomic robots ...

Motion planning: Discretization of C-space

✓ **Discretization** of C-space

Combinatorial problems, Sampling problems

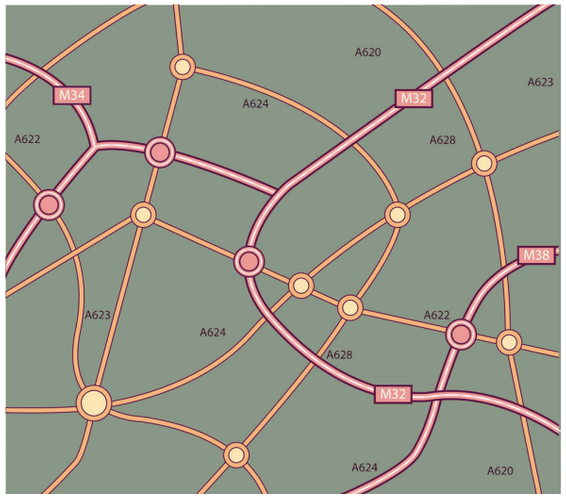


- **Roadmaps** ⇢
  - **Grids**
  - **Spatial decomposition**

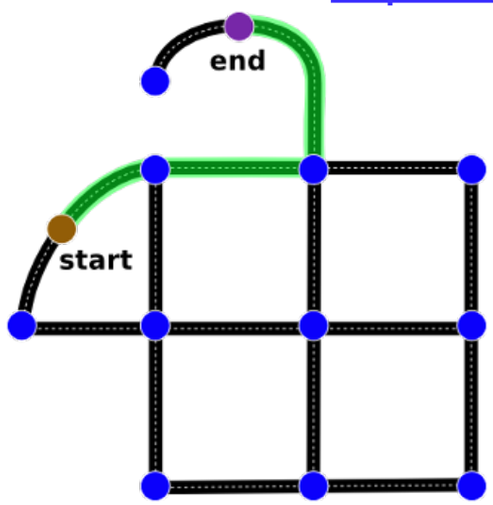
- **Graph search techniques**

  Dijkstra, A*, D*, BFS, DFS, Gradients

Roadmaps

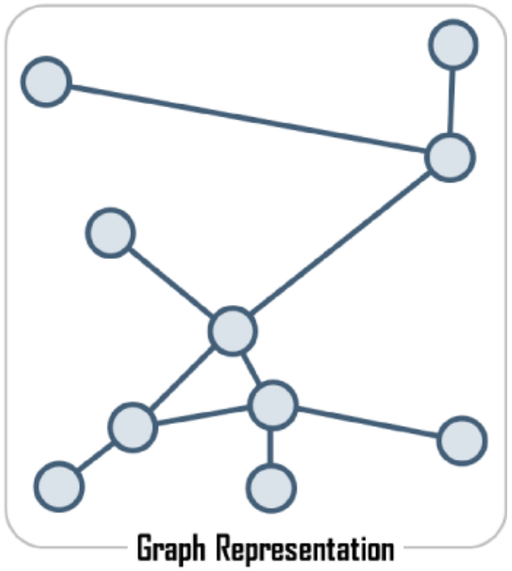


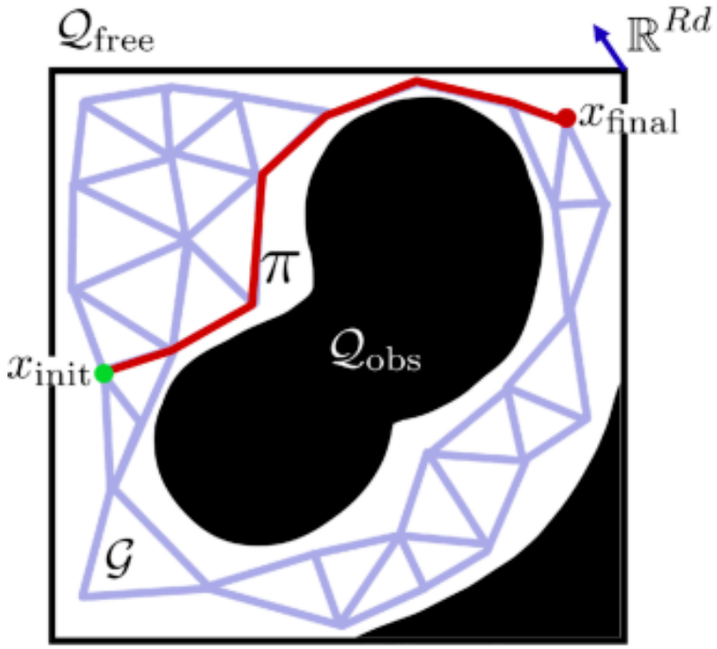
- Roadmaps are compact representations of the space freely accessible for motion
- Staying on the roads guarantees avoiding (known) obstacles
- Roadmaps are not limited to answer one specific query (from A to B), but can be used to find an admissible path for any desired query (**reusability**, **pre-processing**)
- **Roadmap**: Union of curves that create a topological graph

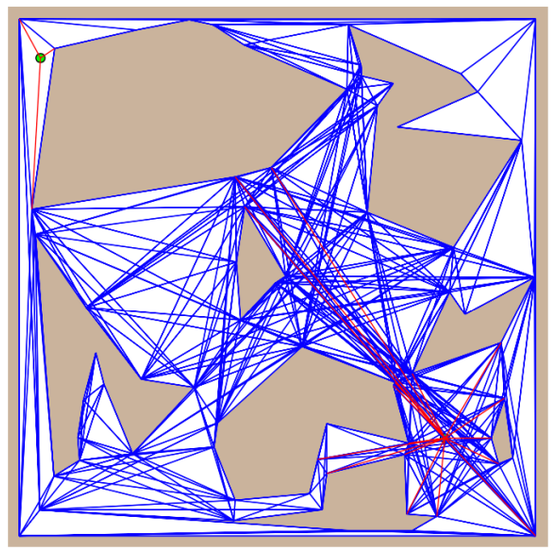


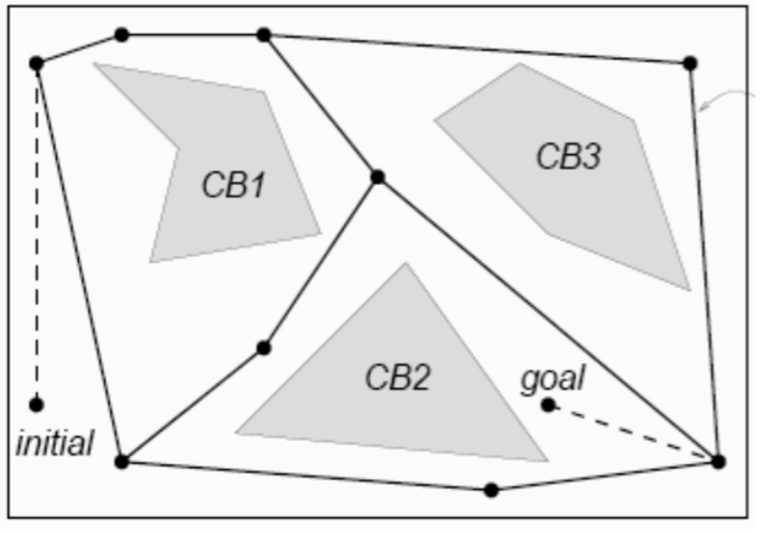
Roadmaps

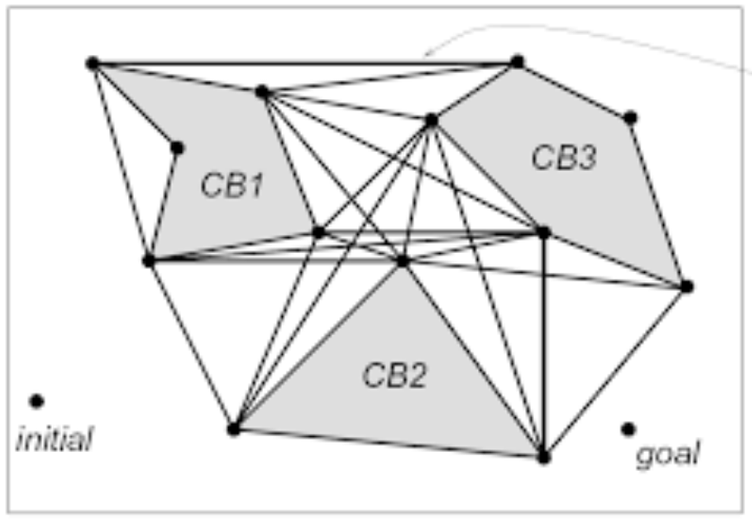
A road map is a graph in $\mathcal{C}_{free}$ in which each vertex is a configuration in $\mathcal{C}_{free}$ and each edge is a collision-free path through $\mathcal{C}_{free}$

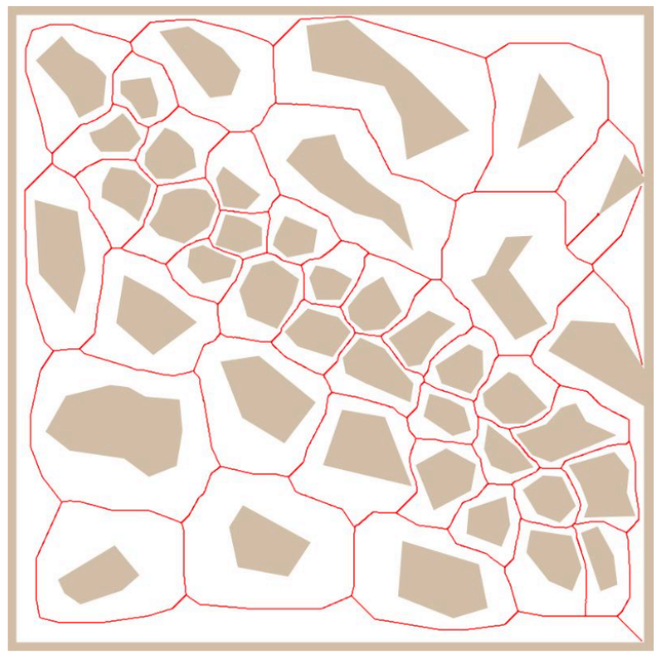




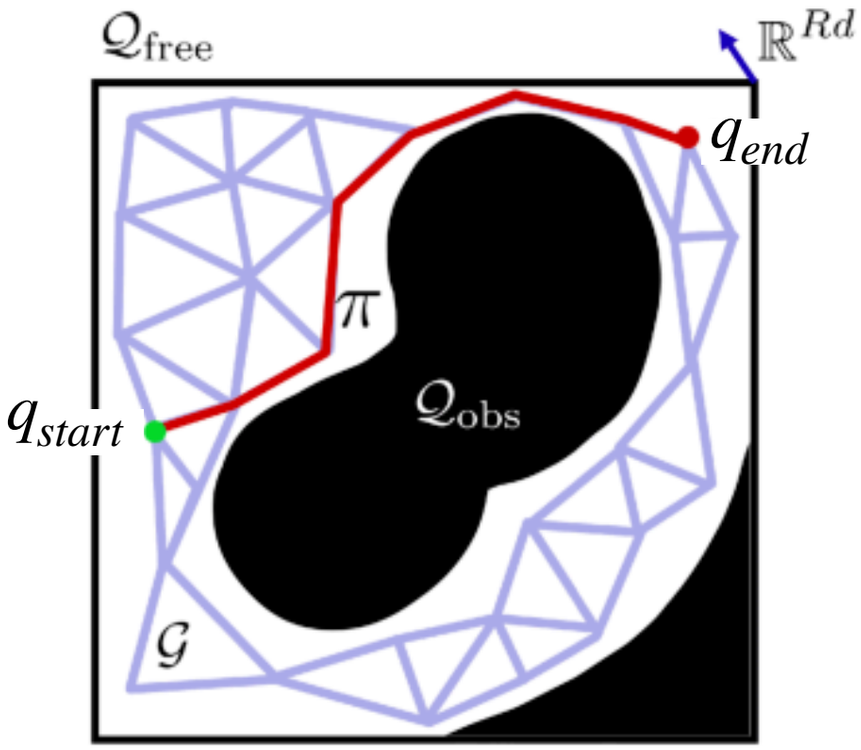


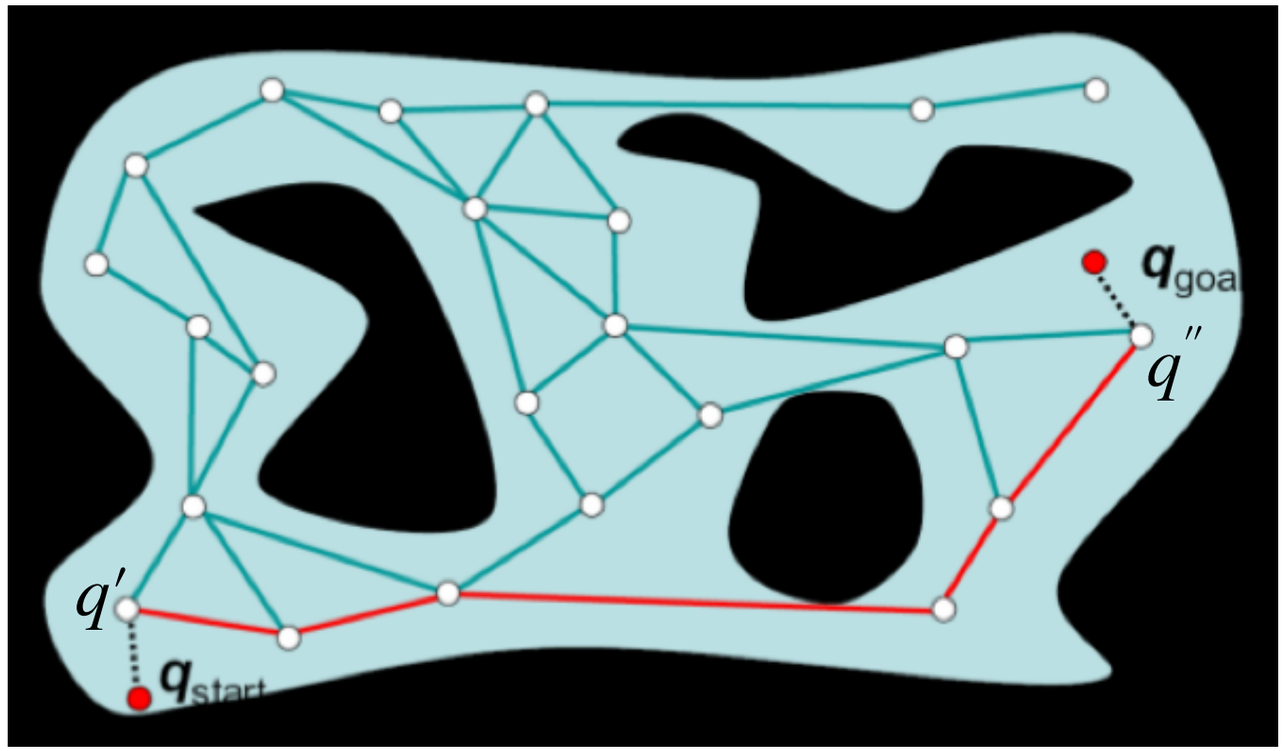






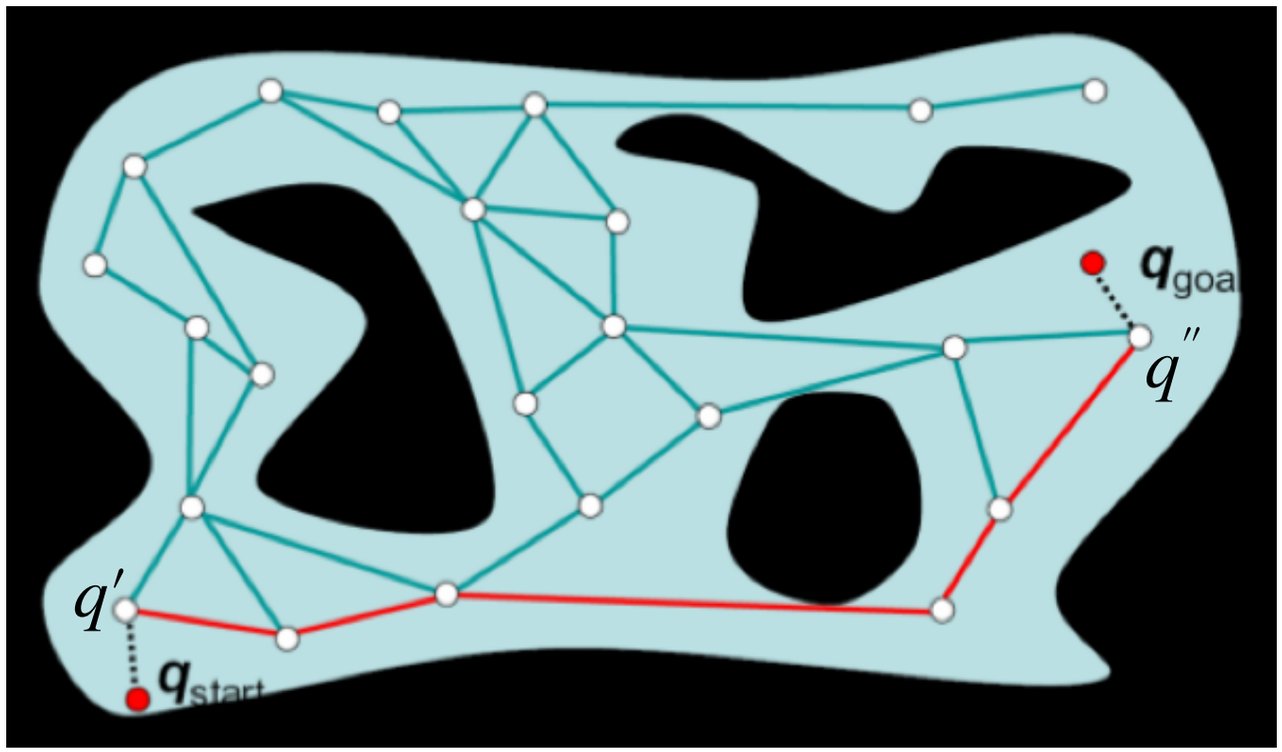
Roadmap for complete algorithms





- A roadmap, RM, is a union of curves such that for all start and goal points in Q_{free} that can be connected by a path:
  - **Accessibility:** There is a path from $q_{start} \in Q_{free}$ to some $q' \in$ RM
  - **Departability:** There is a path from some $q'' \in$ RM to $q_{goal} \in Q_{free}$
  - **Connectivity:** there exists a path in RM between q' and q''
  - **One dimensional**

Build and use a roadmap



1. Build the roadmap
   a) nodes are points in Q_{free} (or its boundary)
   b) two nodes are connected by an edge if there is a free path between them

2. Connect start end goal points to the road map
   at point q' and q'', respectively

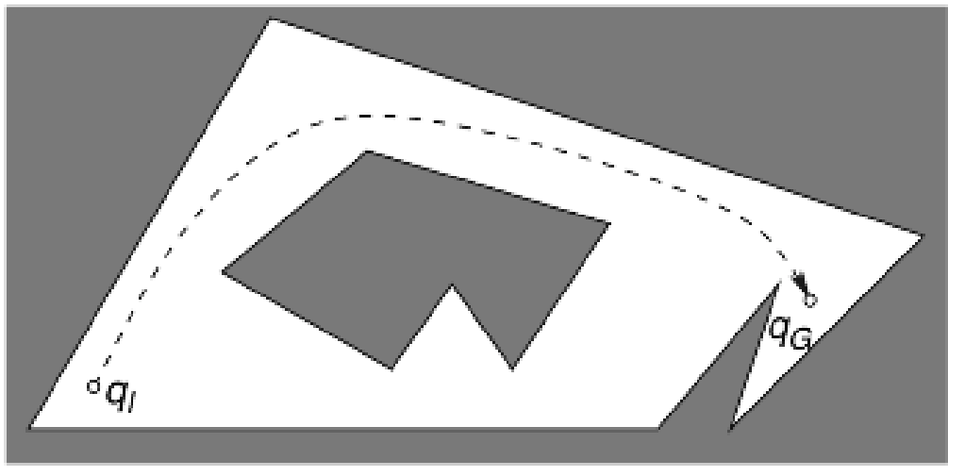
3. Connect find a path on the roadmap betwen q' and q''

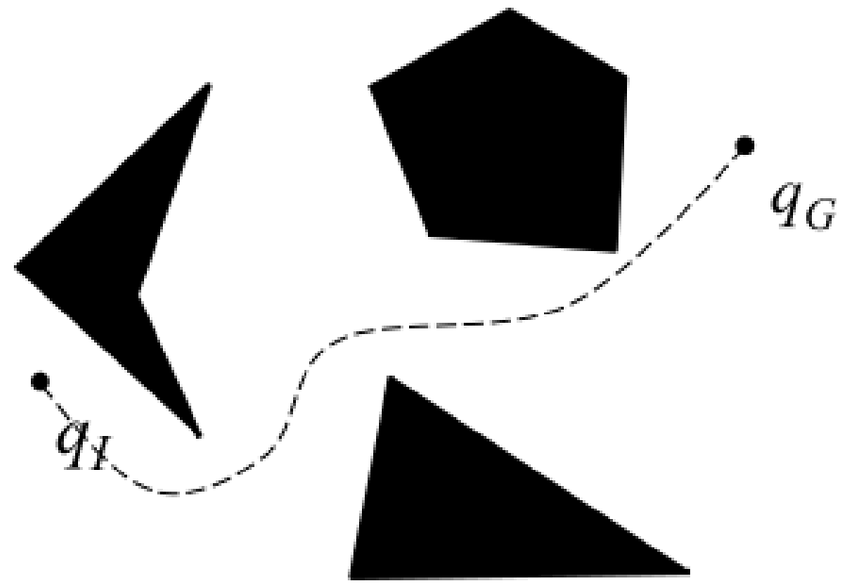
   The result is a path in Q_{free} from start to goal

   Question: what is the hard part here?

What is a good and complete roadmap?

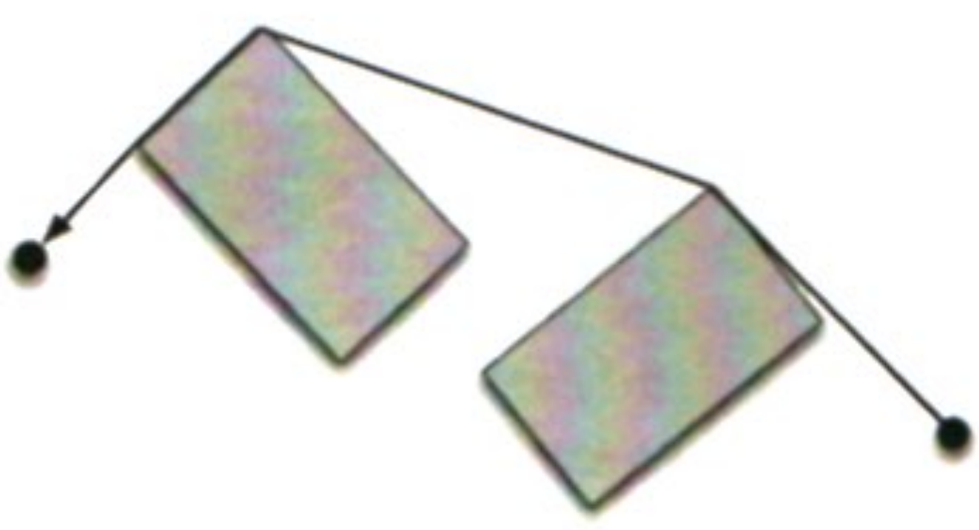
- Let's start, without (much) loss of generality (and performance), to assume to have an omnidirectional (holonomic) point robot moving in a polygonal world: $\mathcal{W} = \mathbb{R}^2$, $\mathcal{C} \subset \mathbb{R}^2$

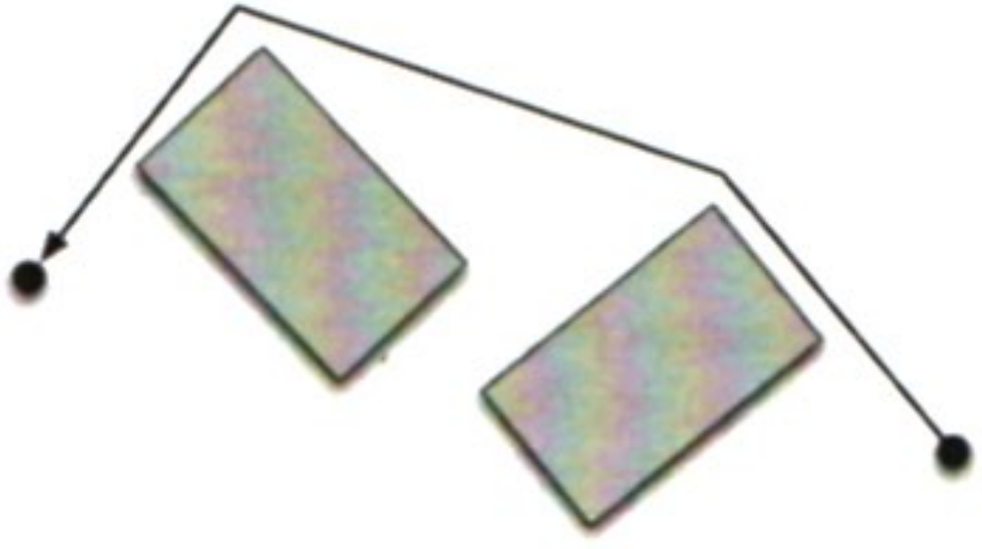




Semi-free space: touching the obstacles

- ***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 obstacles' frontier does not belong to the set)

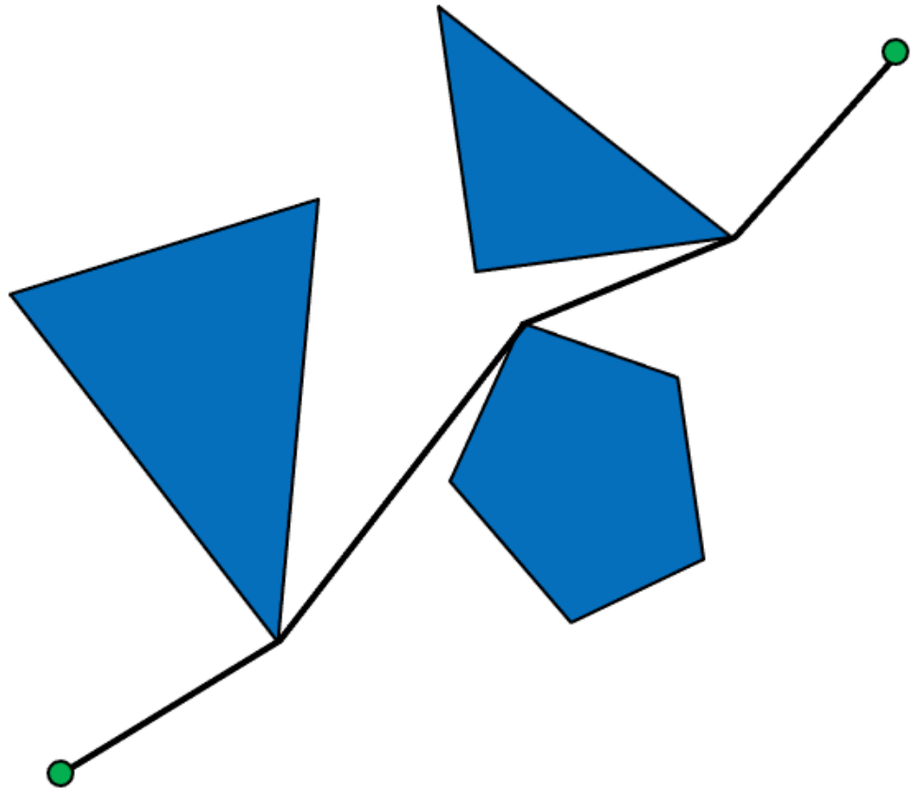




Optimal shortest path?

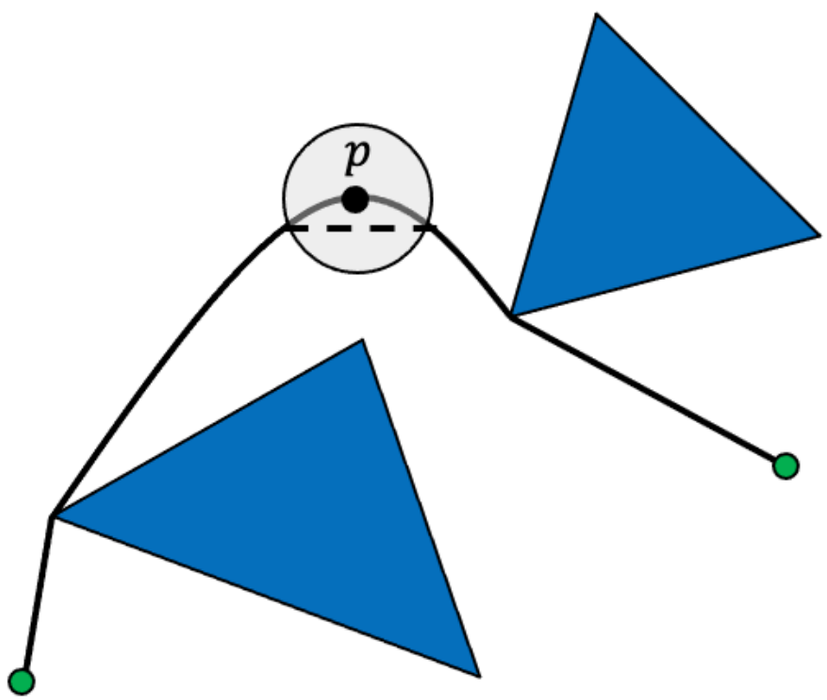
- Polygonal path: sequence of connected straight lines
- Inner vertex of polygonal path: vertex that is not beginning or end

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



Optimal shortest path?

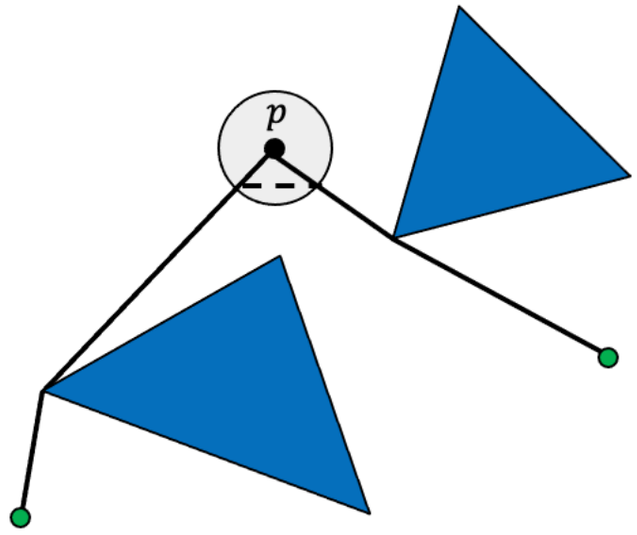
- Suppose for contradiction that shortest path is *not* polygonal
- Obstacles are polygonal ⇒ ∃ point p in interior of free space such that "(shortest) path through p is curved"
- p in free space ⇒ ∃ disc of free space around p
- Path through disc can be shortened by connecting points of entry and exit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



Optimal shortest path?

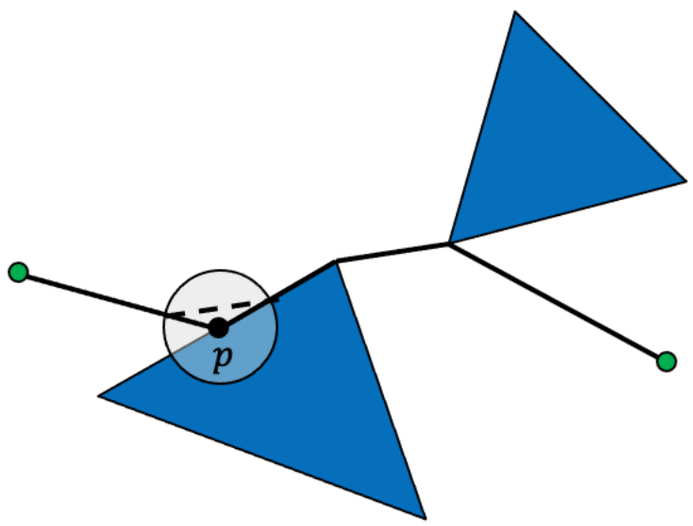
✓ Path is polygonal

- 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

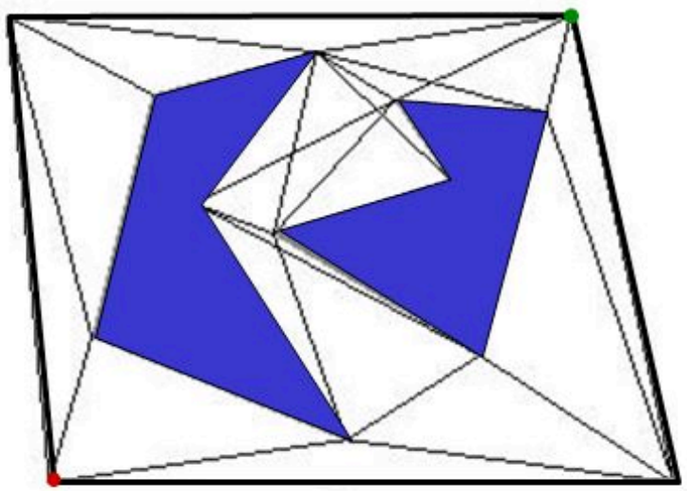
✓ Inner vertices are vertices of obstacles ■

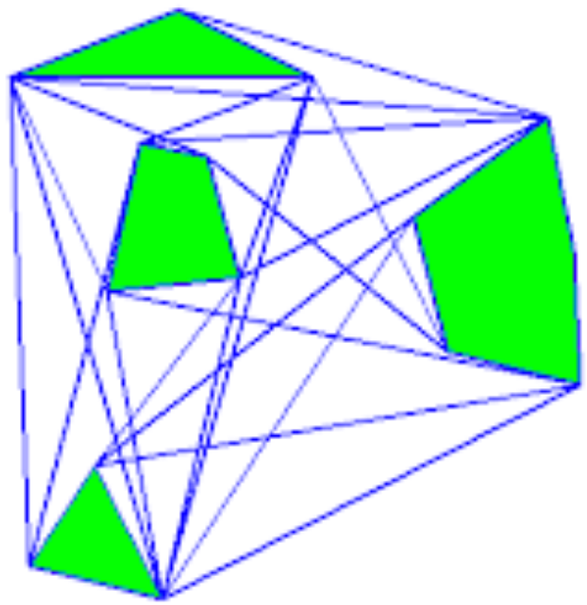


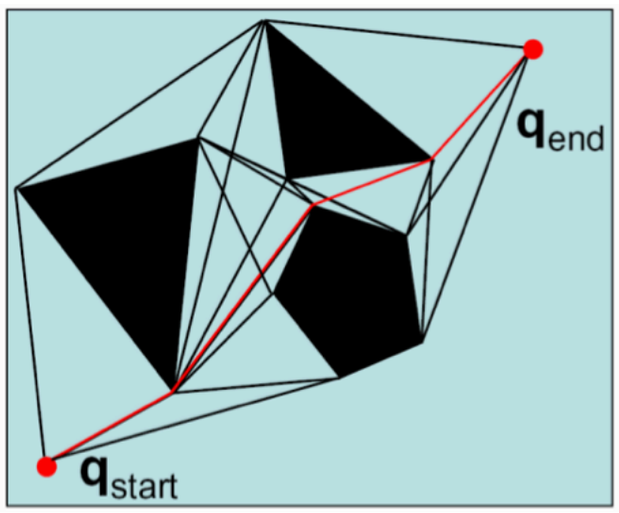
Visibility graphs

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

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

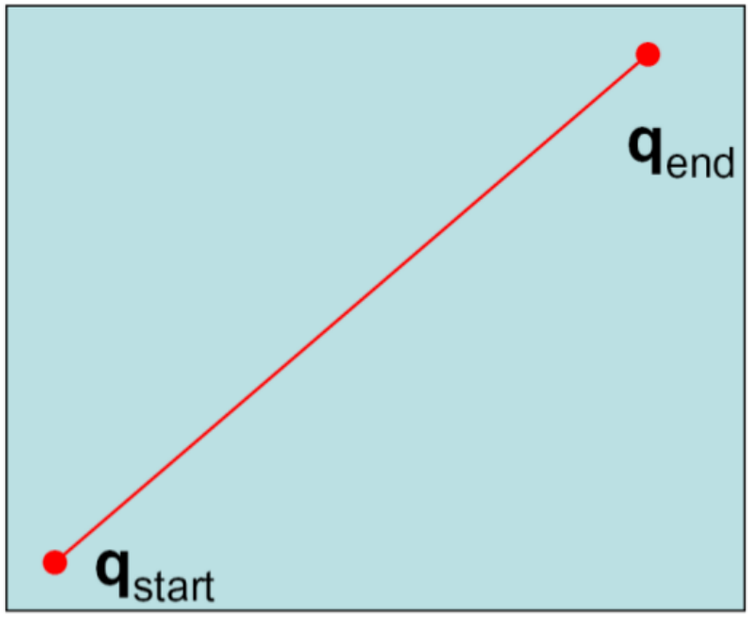




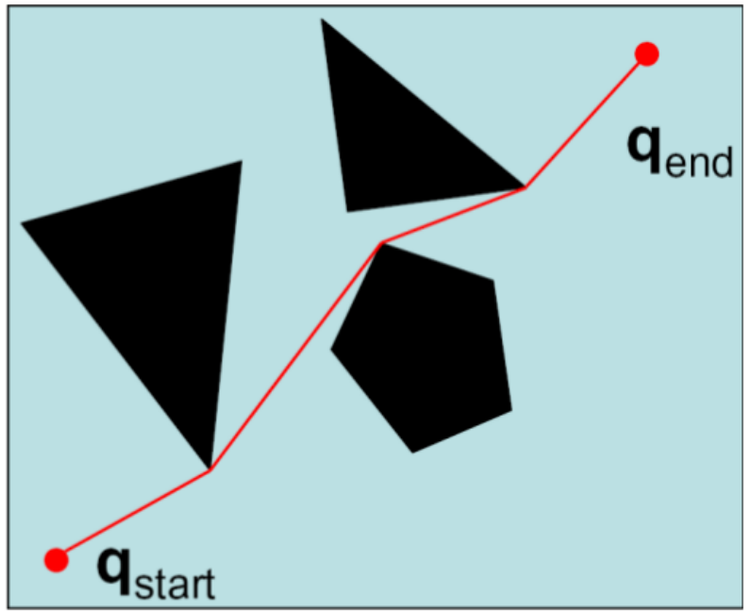


Visibility graphs

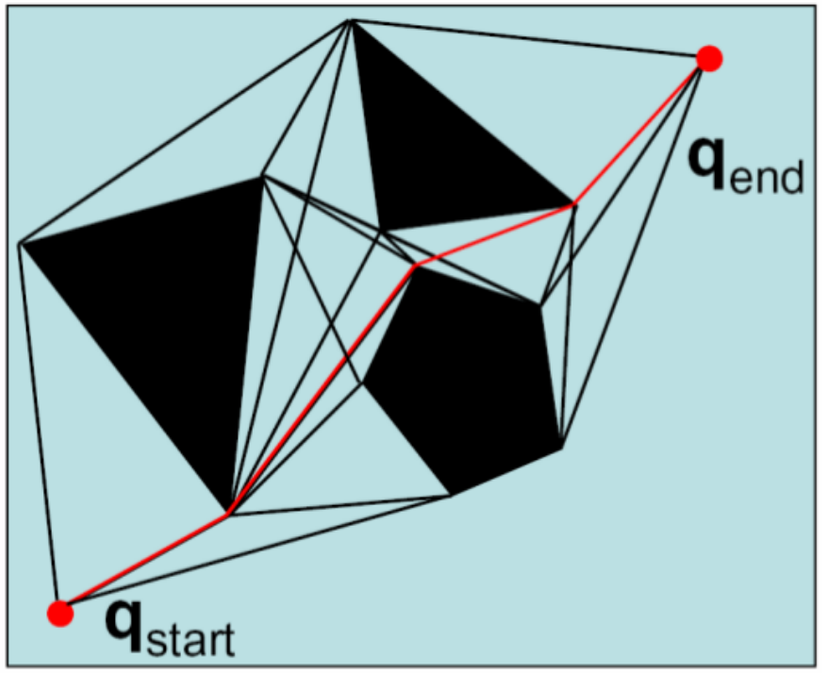
The previous theorem guarantees that the shortest path between $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{end}$ is a polygonal line connecting start and goal configuration through the vertices of the polygonal obstacles: it corresponds to the **shortest path in the visibility graph**



No obstacles, visibility graph only includes $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{end}$ and their straight line connection



Obstacles, the shortest path goes trough the vertices.

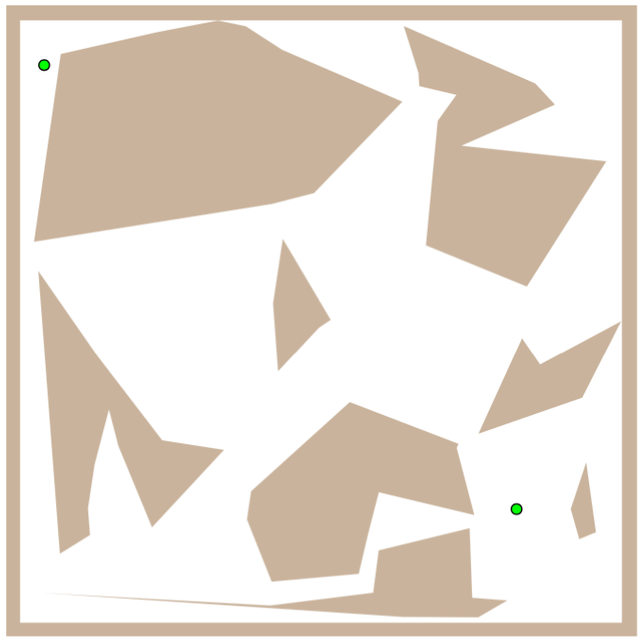


Visibility graph + shortest path.

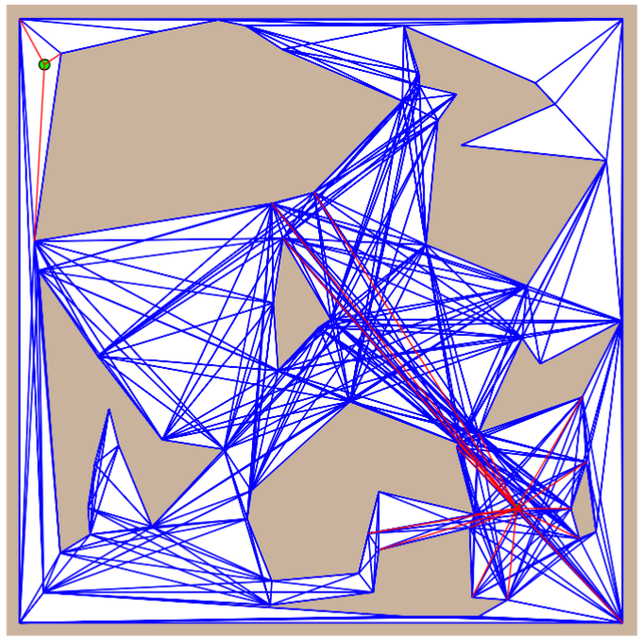
Construct and use visibility graphs

1. Compute visibility graph
2. Find the shortest path

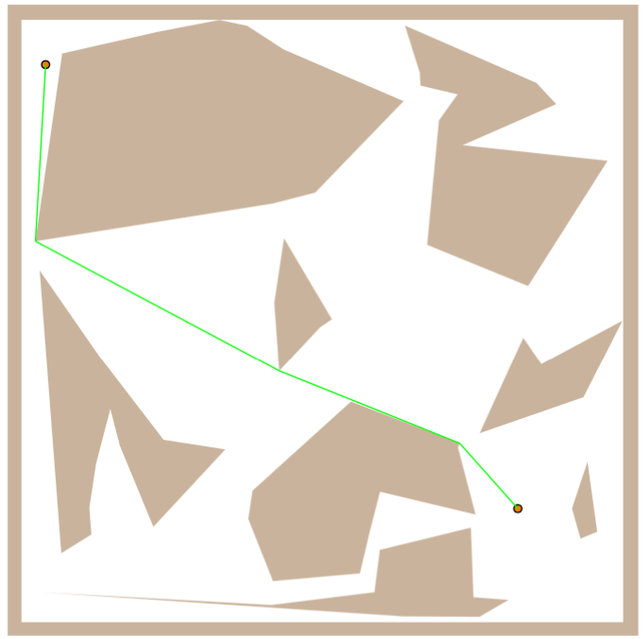
E.g., by Dijkstra's algorithm



Problem



Visibility graph

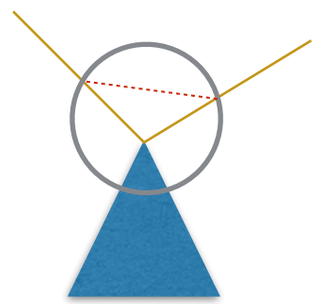


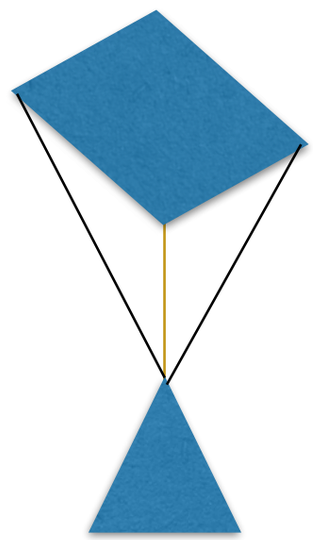
Found shortest path

Reduced Visibility graphs

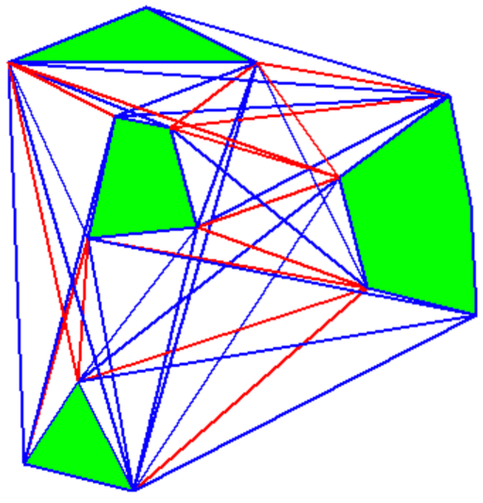
- The current graph as too many lines
  - lines to concave vertices
  - lines that "head into" the object

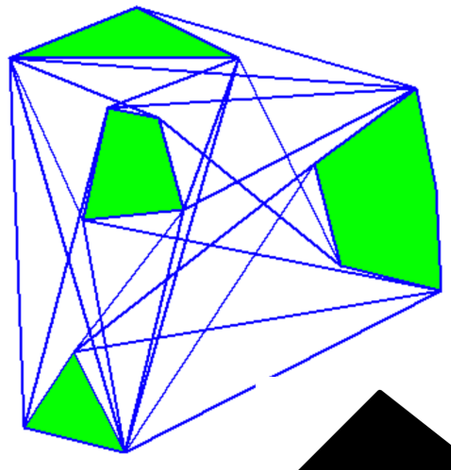
} Can't be on a shortest path

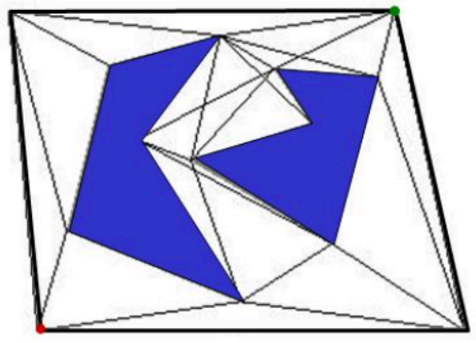


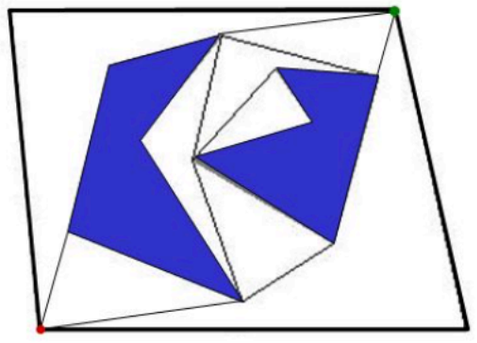


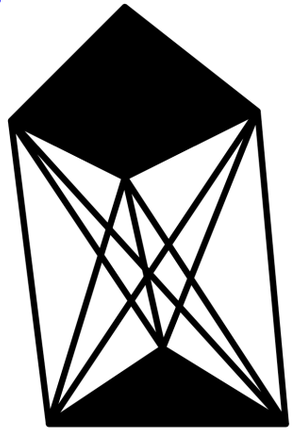
- A reduced visibility graph consists of
  - nodes that are convex
  - edges that are "tangent" (i.e. do not head into the object at either endpoint)

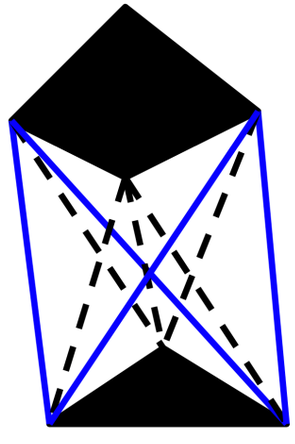








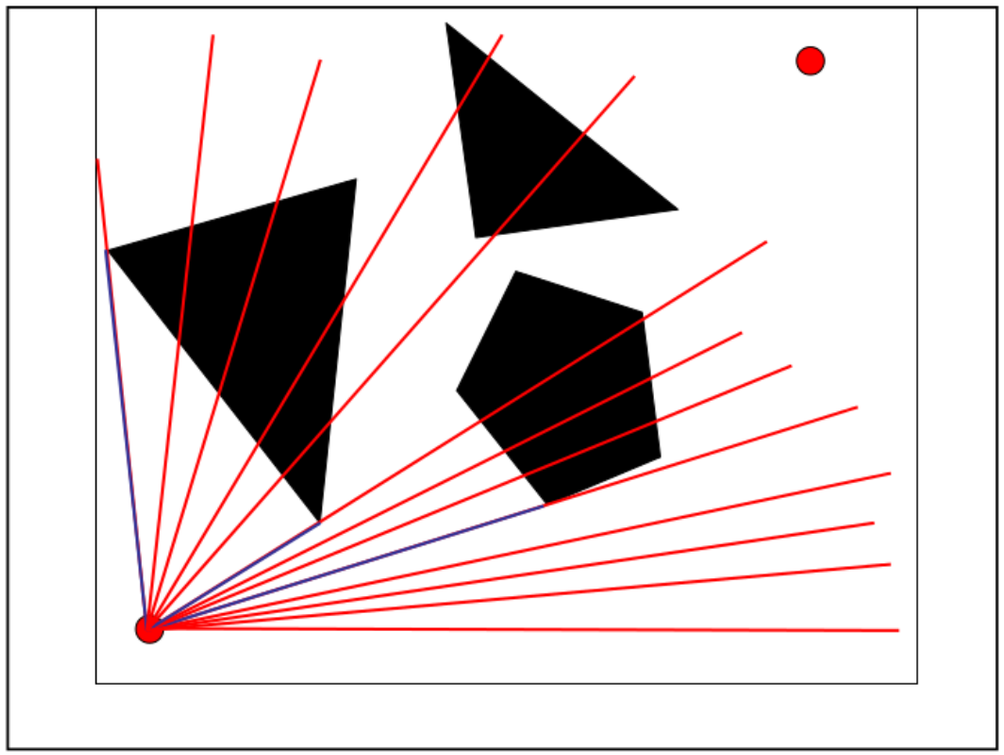


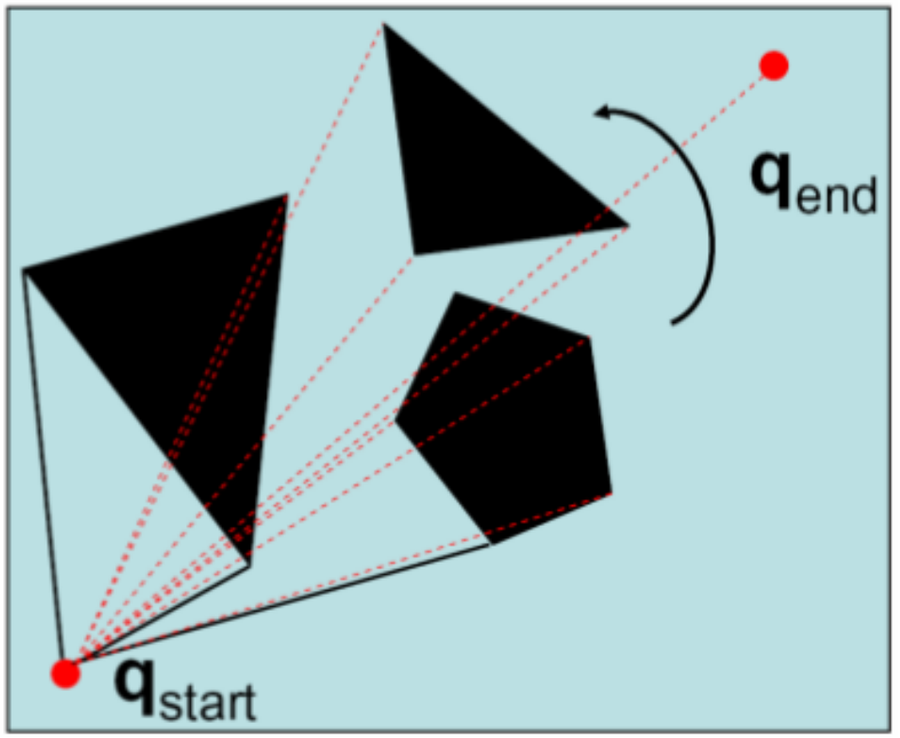


Construction of the visibility graph

- 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



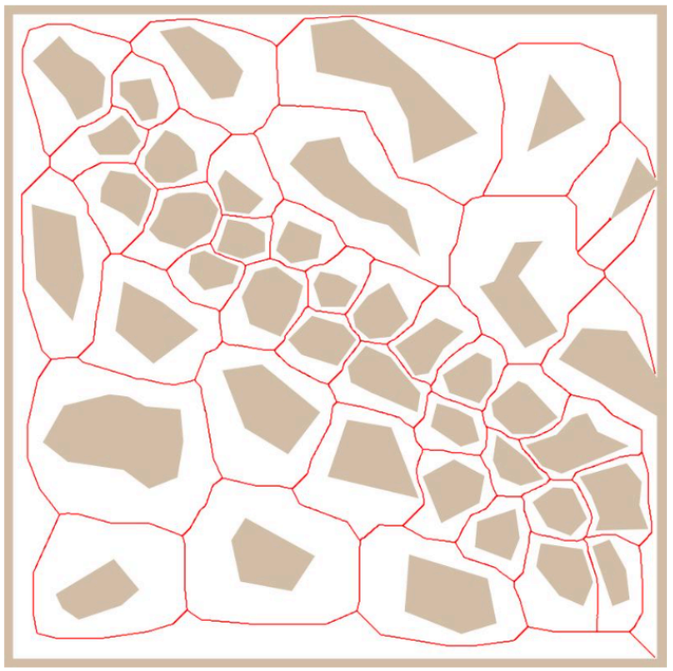


Shortcoming of visibility graphs

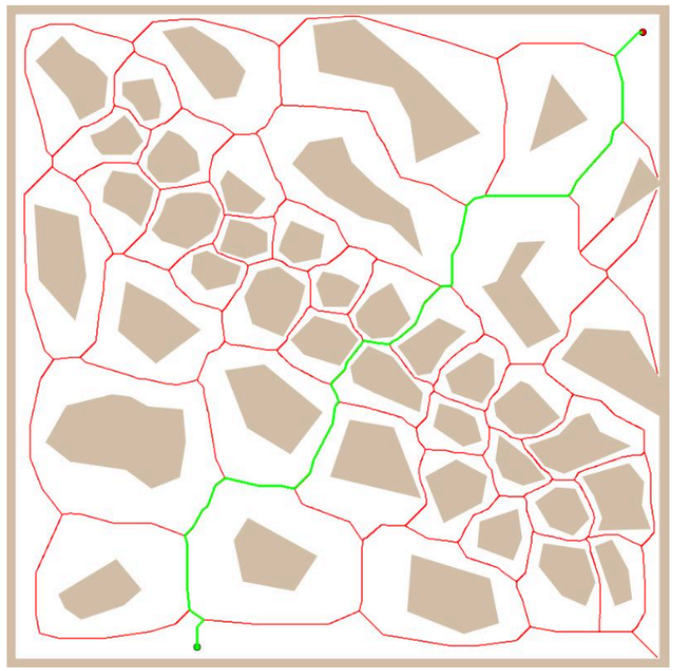
- Guarantee of shortest path but tries to stay as close as possible to obstacles, which is not precisely what we want in practice $\rightarrow$
- Any execution error will lead to a collision
- Complicated in more than 2 dimensions
- Finding a safe path (in practice) is maybe more important than strict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

Roadmaps as Voronoi graph

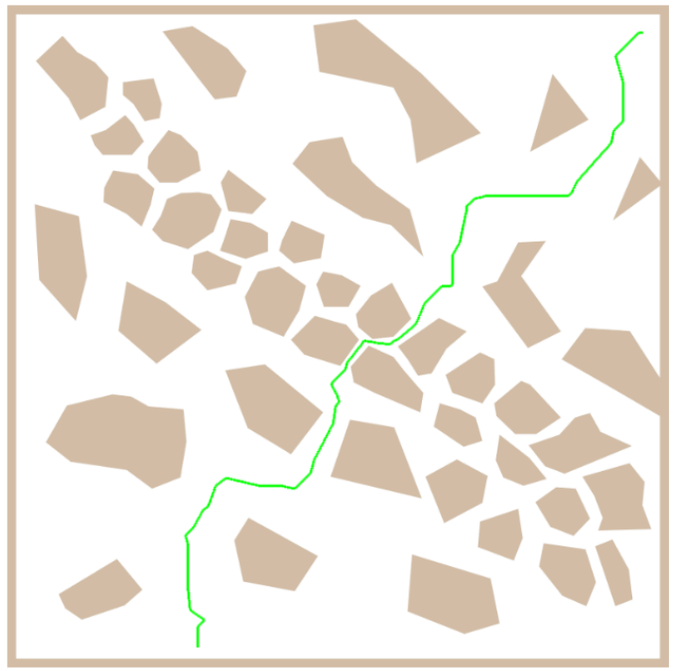
1. Roadmap is Voronoi graph that maximizes clearance from the obstacles
2. Start and goal positions are connected to the graph
3. Path is found using a graph search algorithm



Voronoi graph



Path in graph

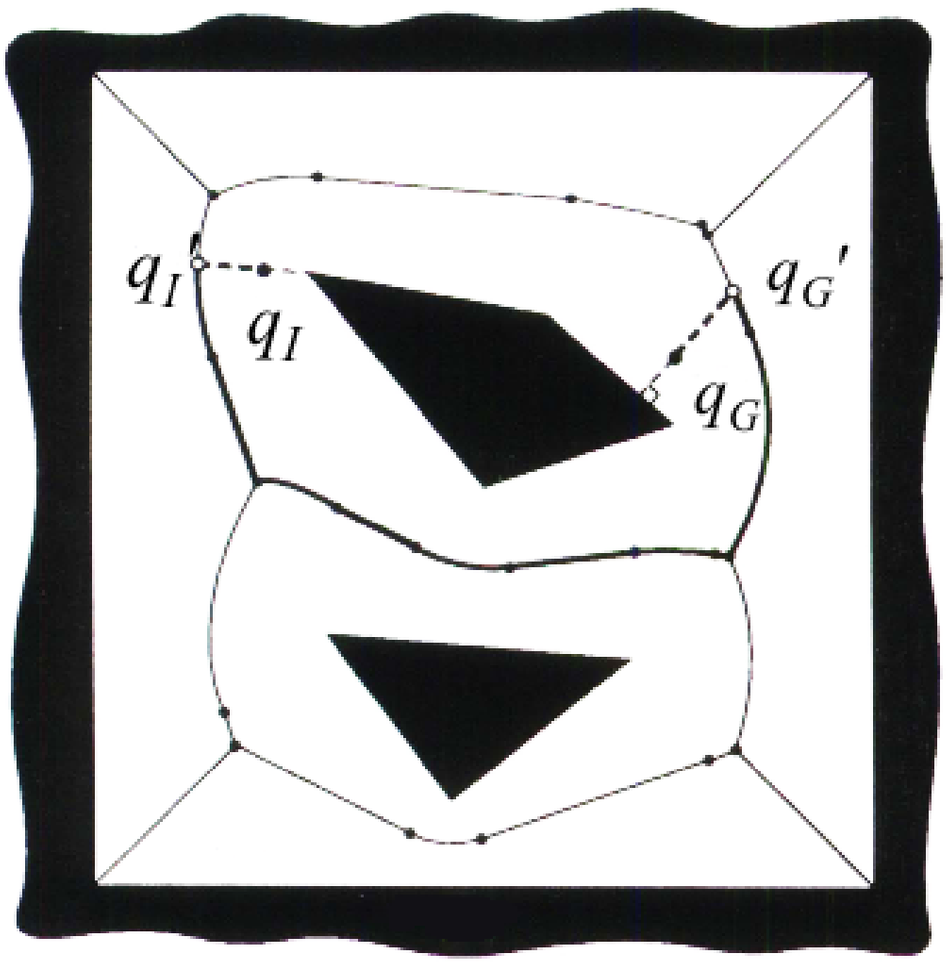


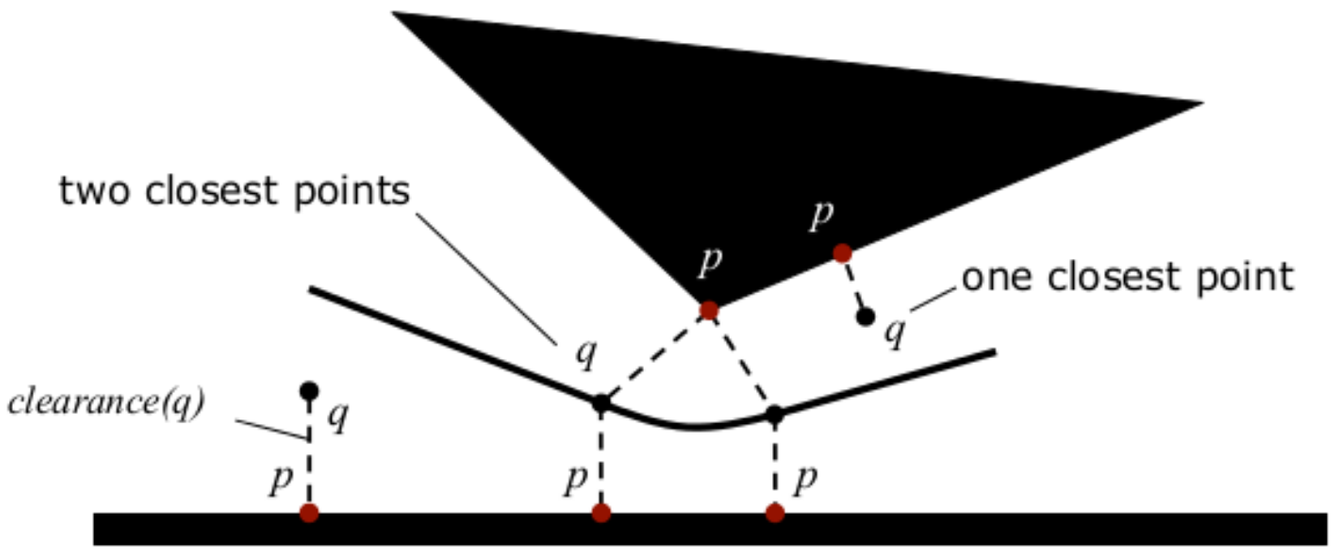
Found path

Generalized voronoi diagrams

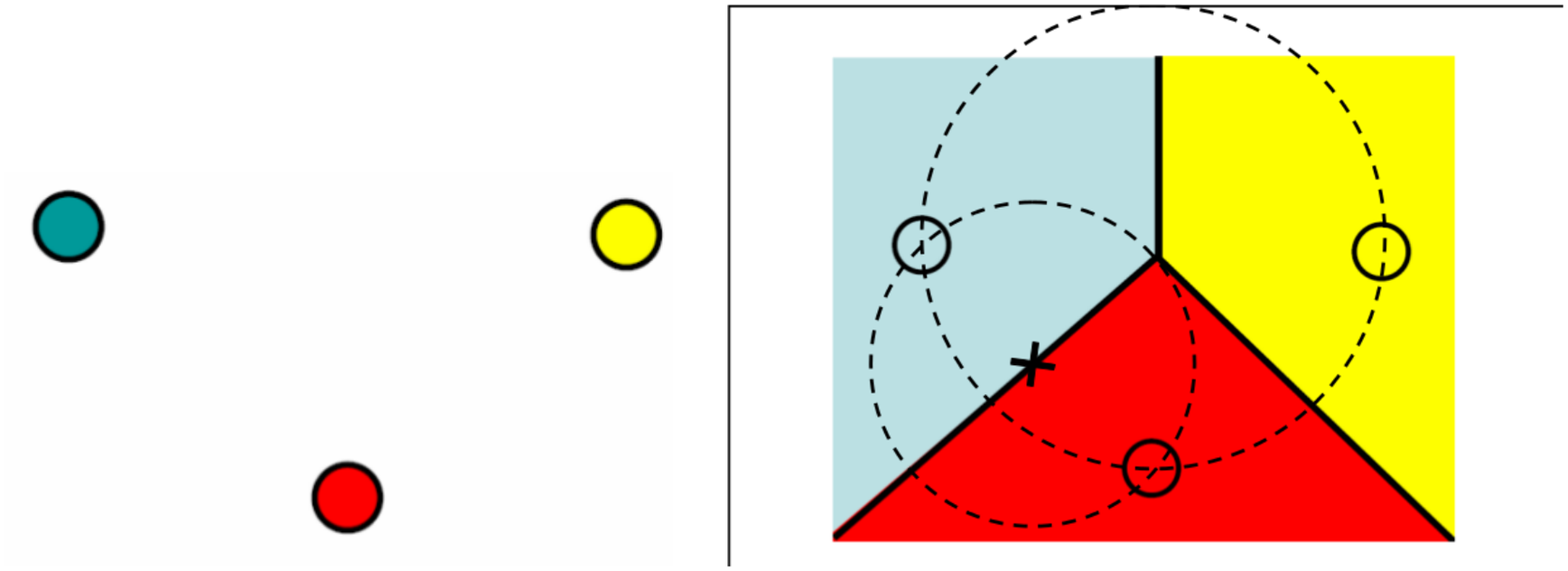
The locus of points that are equidistant from the closest two or more obstacle boundaries (in C_{obs}), including the workspace boundaries. In other words, the set of points q whose cardinality of the set of boundary points (in C_{obs}) with the same distance to q is greater than 1

The region with the same maximal clearance from all nearest obstacles



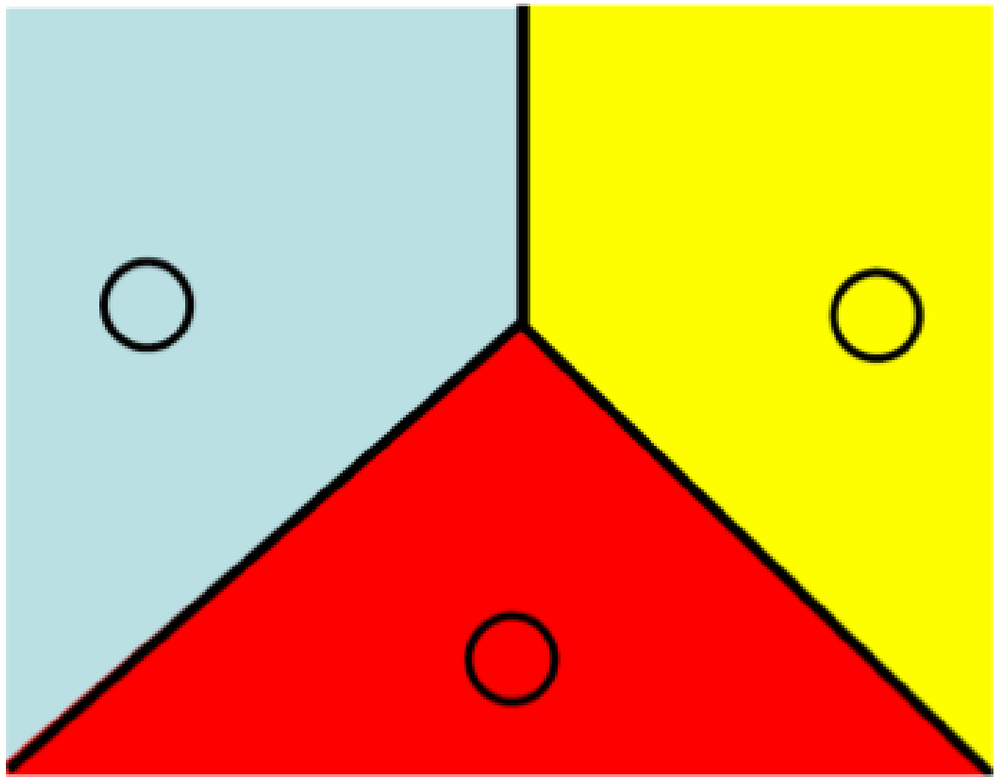


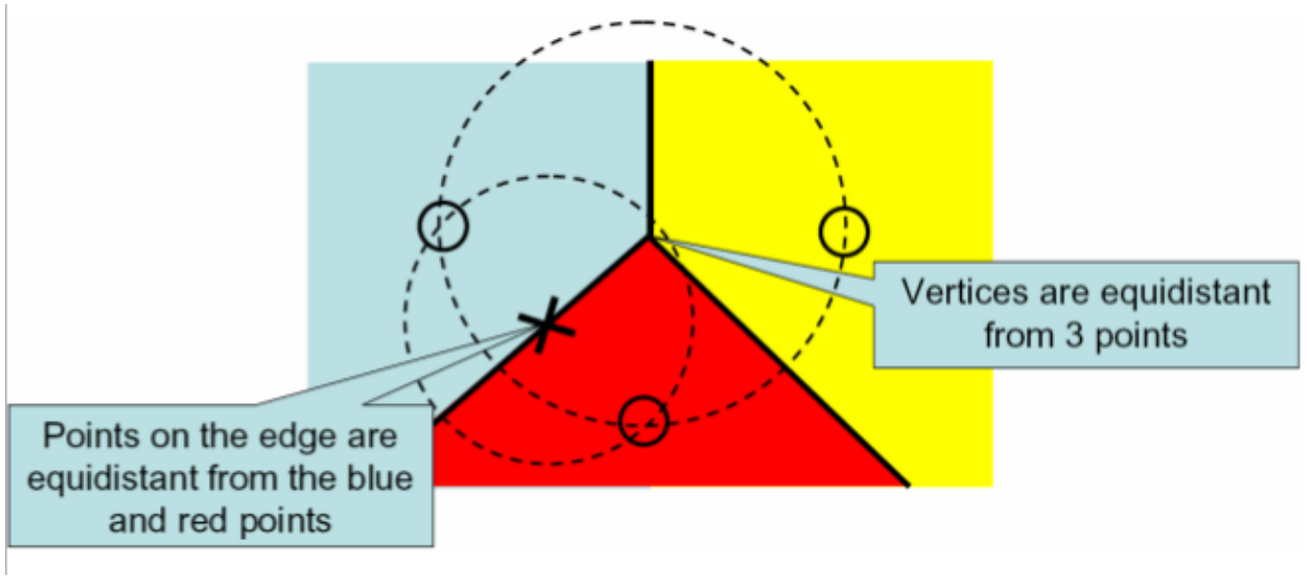
Vornonoi cells



- 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 → *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

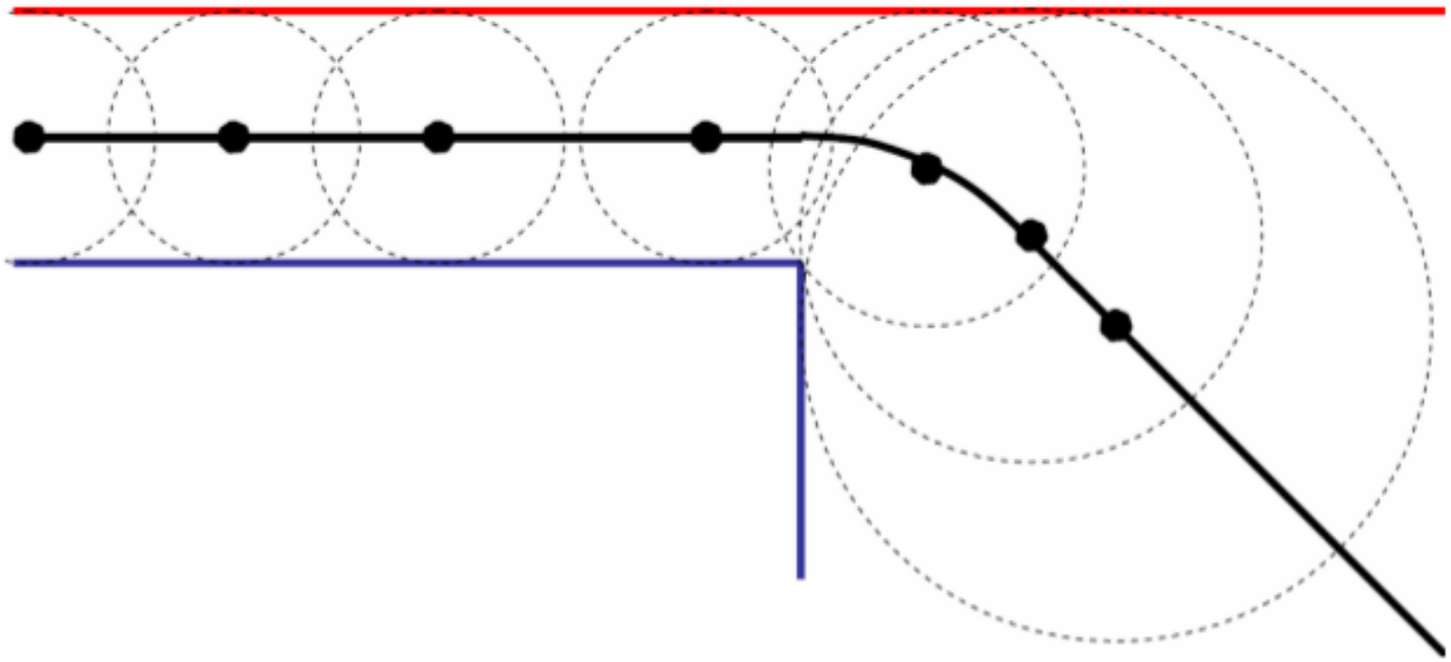
Voronoi diagrams





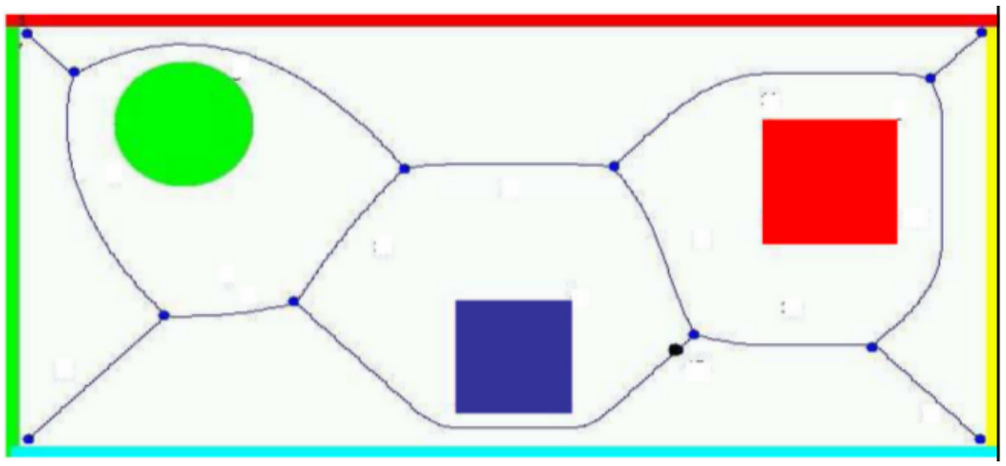
- Voronoi diagram: The set of line segments separating the regions corresponding to different colors
- Line segment: points equidistant from 2 data points
- Vertices: points equidistant from > 2 data points
- Complexity (in the plane): $\mathcal{O}(N \log N)$ in time, $\mathcal{O}(N)$ in space

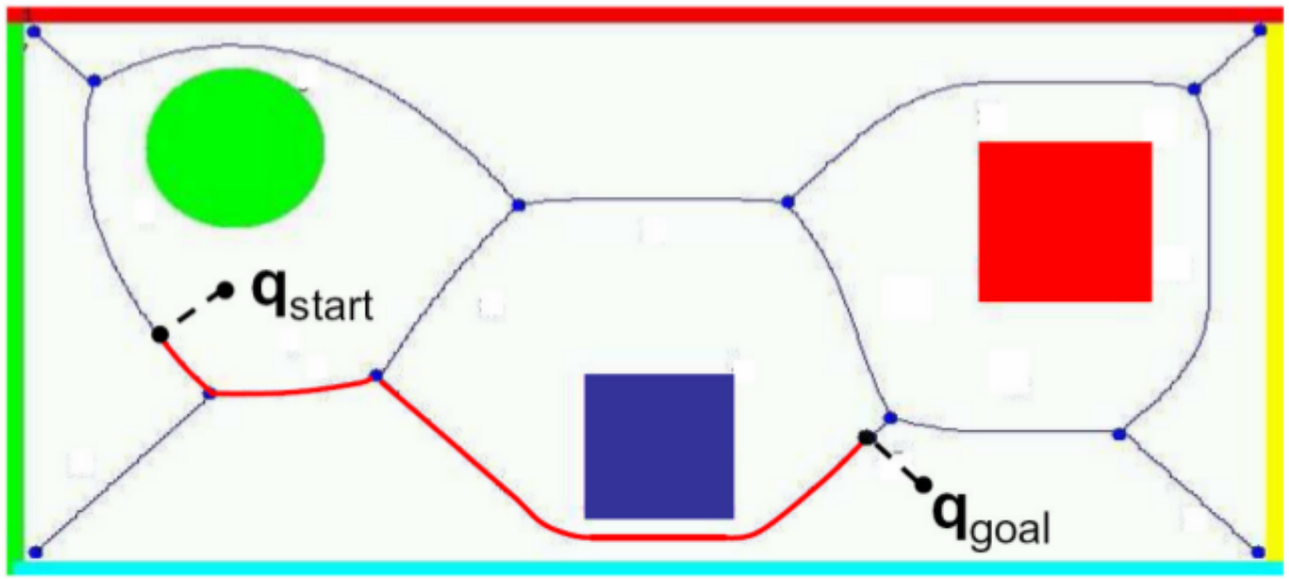
Voronoi diagrams for cluttered environments



- 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

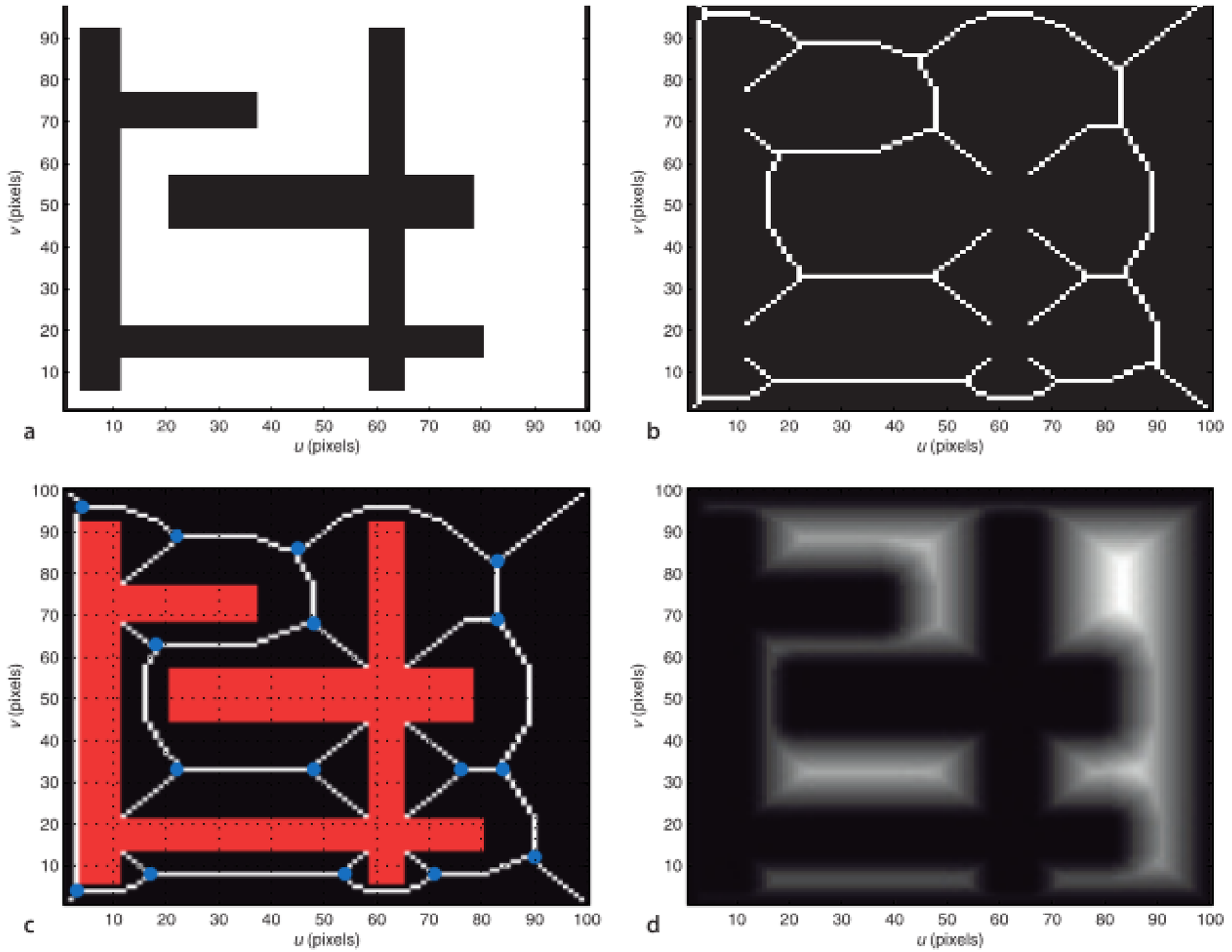
Planning in generalized Voronoi diagrams





- 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 generalized Voronoi graphs

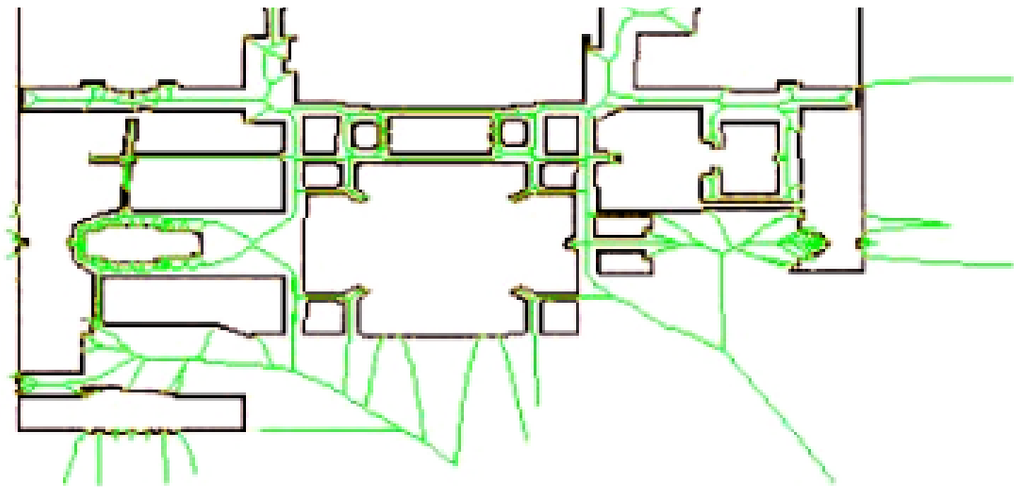


- Real environment → Voronoi skeleton → Overlapping the environment with the skeleton → A heat map for the distance from the nearest obstacle (*distance transform*)

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

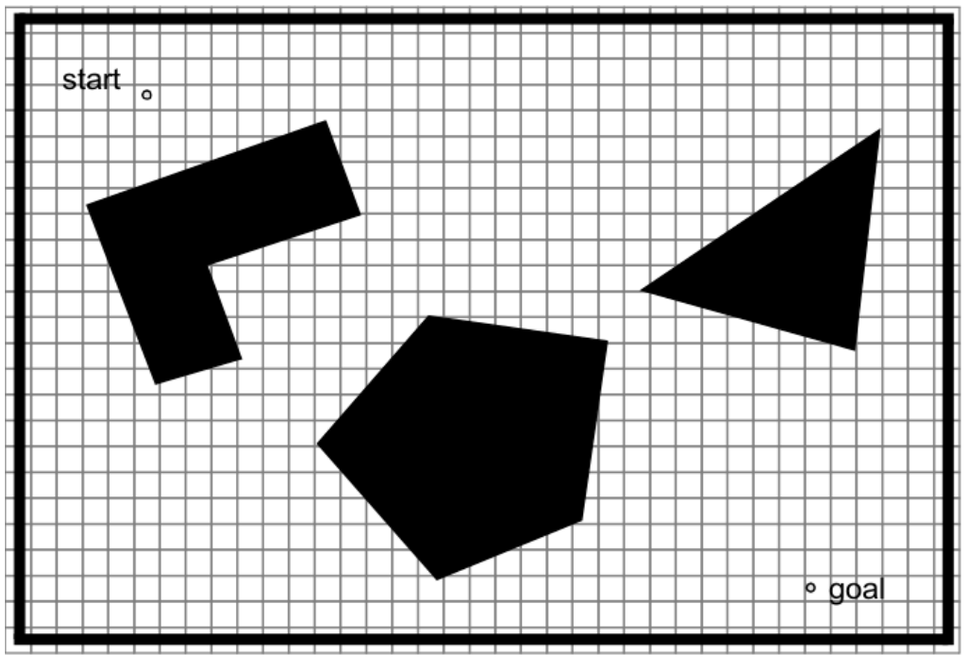


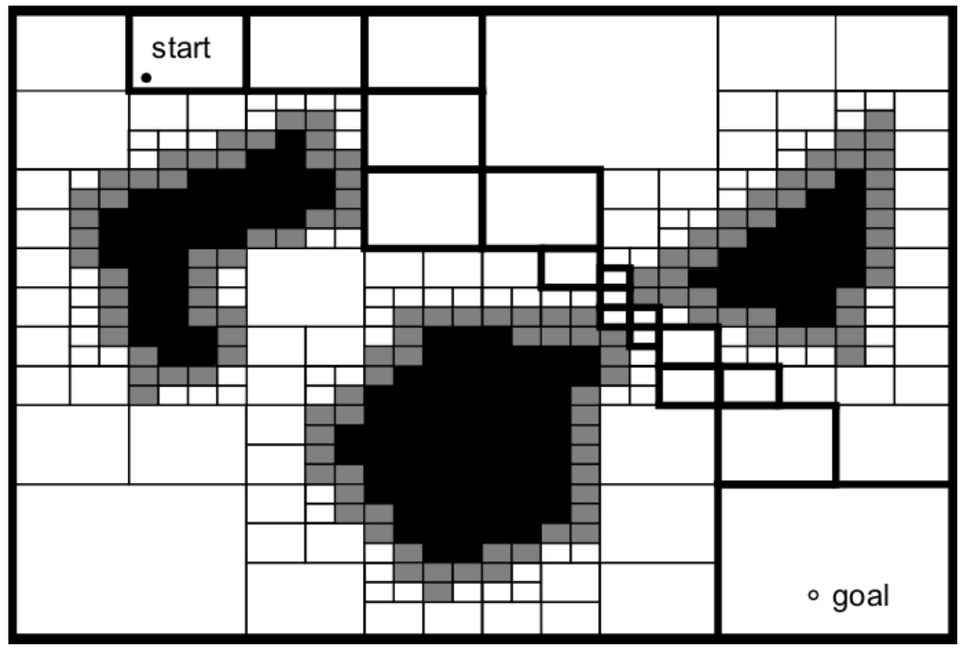
- **For robots with short-range sensors**: the path-planning algorithm maximizes the distance between the robot and objects in the environment, therefore navigating on a Voronoi path might be problematic, since the robot might not be able to use sensing to detect the objects around it and localize itself, being always too distant from the obstacles to sense them.

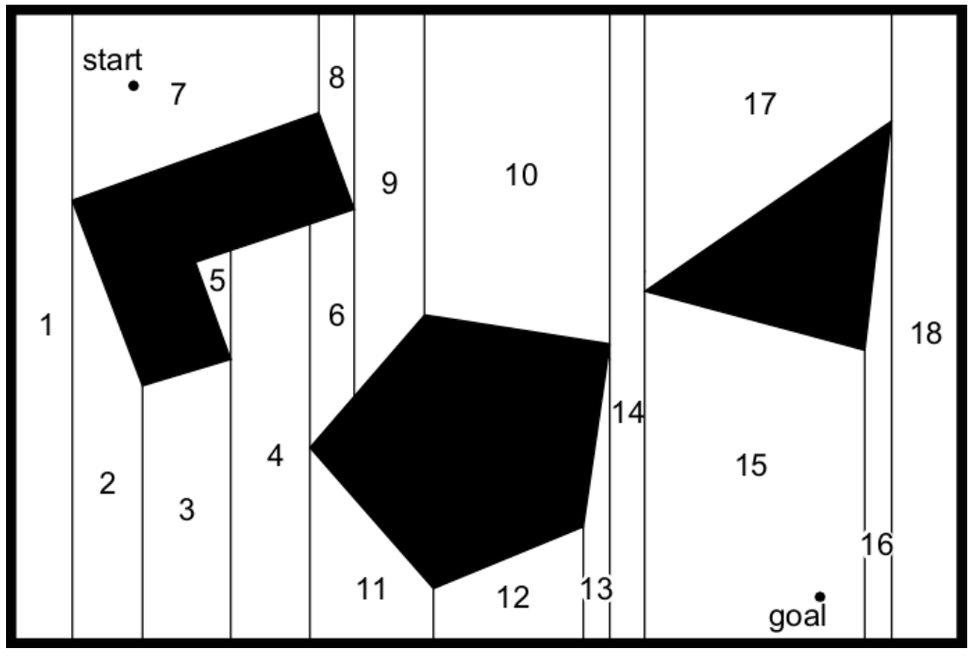
- **For robots with long-range sensors**: the Voronoi diagram method has over most other obstacle avoidance techniques the advantage of *executability*. In fact, following the Voronoi path results from maximizing the distance while maintaining equidistant from the surrounding objects, which can be done 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.

Discretization → Roadmap by spatial decomposition

1. A discrete, **cell-based spatial decomposition** is defined over the C-space, where any cell intersecting $\mathscr{C}_{\text{obs}}$ is marked as BLOCKED, FREE otherwise

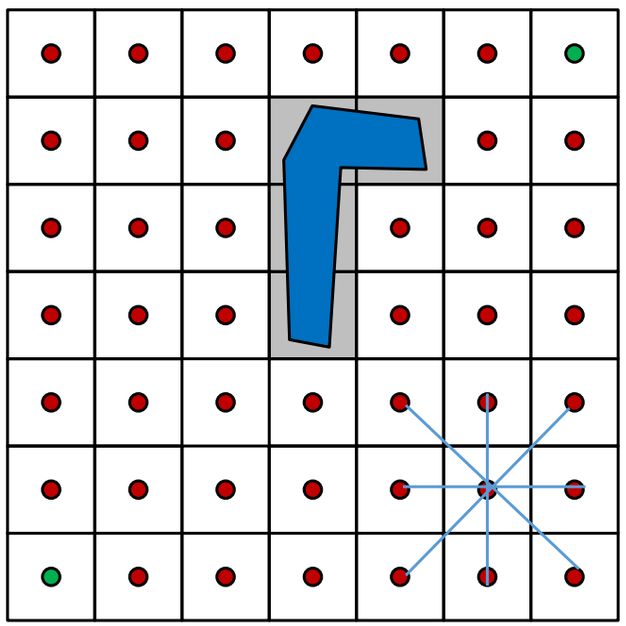


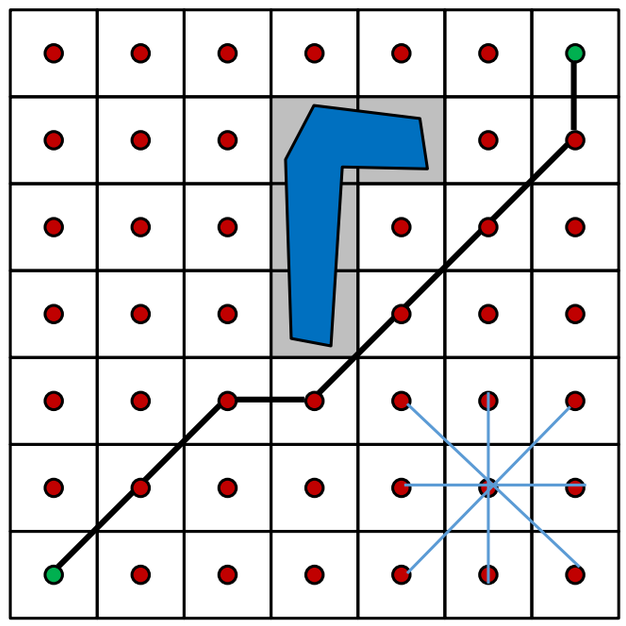




Discretization by spatial decomposition

1. A discrete, **cell-based spatial decomposition** is defined over the C-space, where any cell intersecting $\mathscr{C}_{\text{obs}}$ is marked as BLOCKED, FREE otherwise
2. Determine adjacent FREE cells and construct a **traversability / connectivity graph** among them: E.g., vertices can be placed in the mid of each cell, edges connect them
3. Find the cells in which the initial and goal configurations lie, and search for the best (e.g., shortest) path in the connectivity graph that joins initial and goal cells: the path consists of a sequence of FREE cells ↔ **node path in the graph**

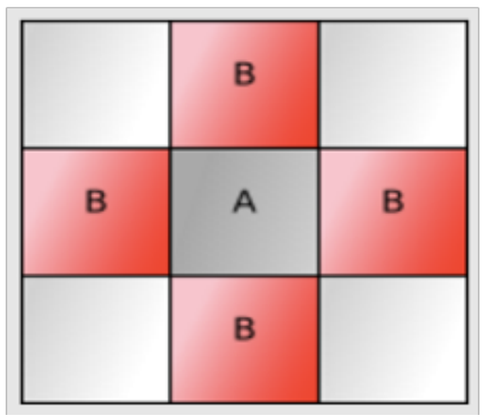


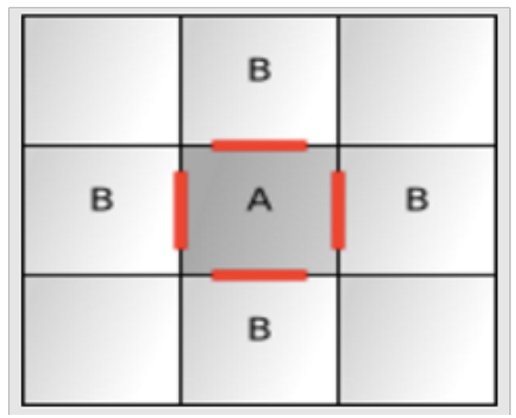


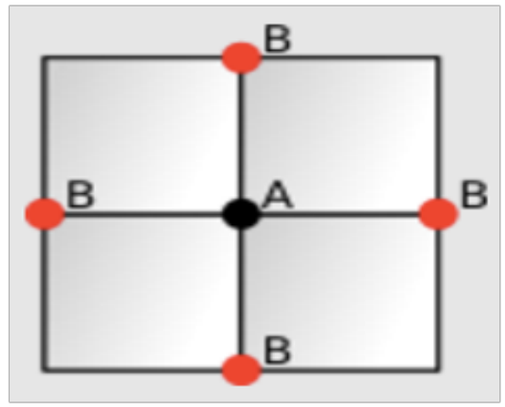
Discretization by spatial decomposition

Execution: For each cell in the path sequence, compute a navigation path within each cell for moving into the cell that follows in the path

- Navigation: it can be done in many different ways, like passing through the midpoints of the cell boundaries or by a sequence of wall-following motions and movements along straight lines

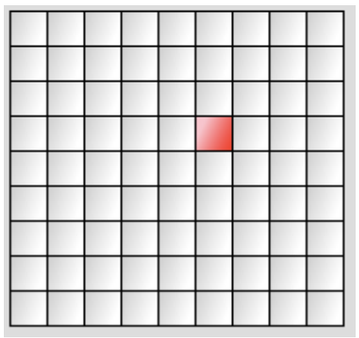


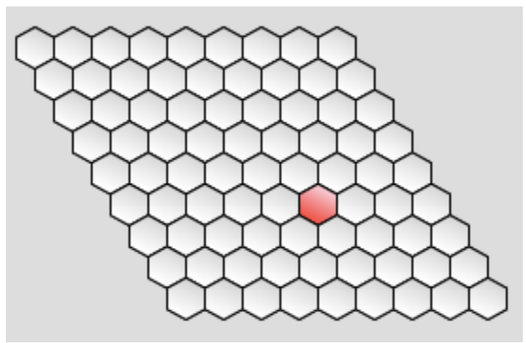




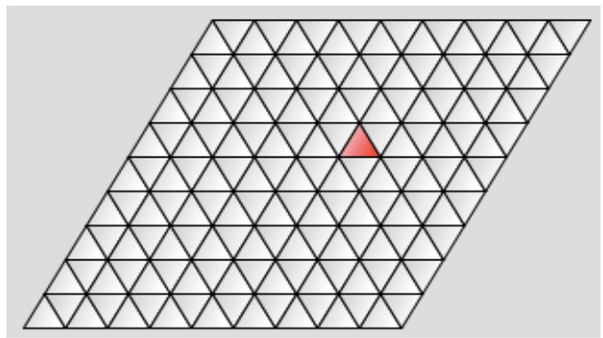
Cell shape, reference points

- Cells can have different *shapes*



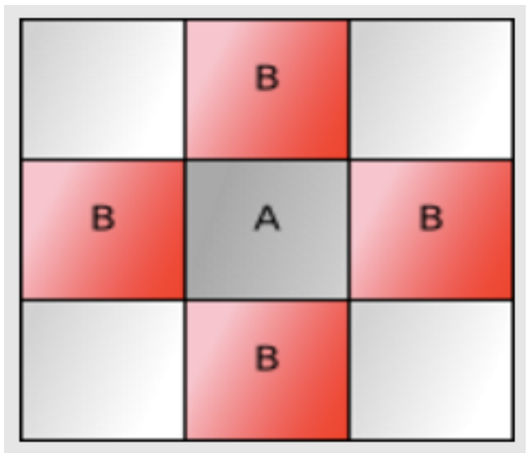


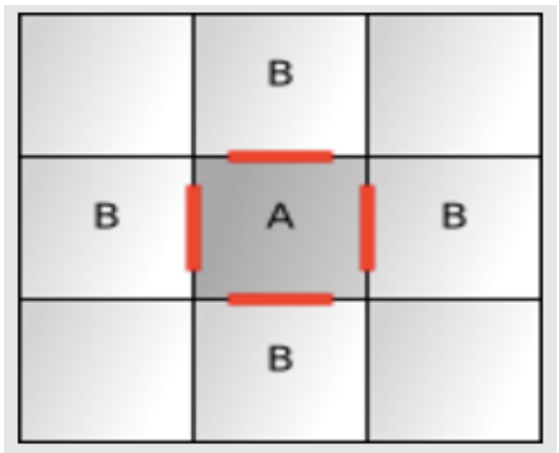
Reduce distance distortion

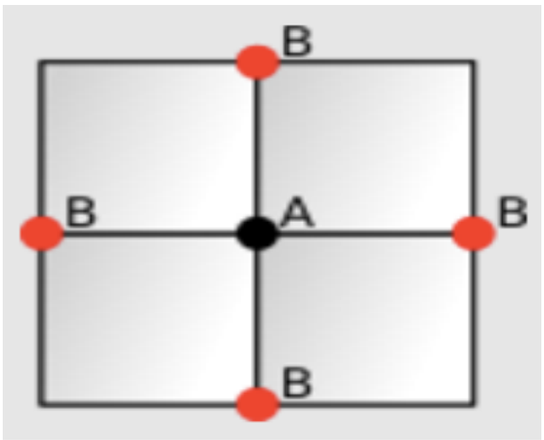


Small area / Large perimeter

- For adjacent free cells, cells' centers can be replaced by a point on edges or by vertices, for more flexibility for local motion

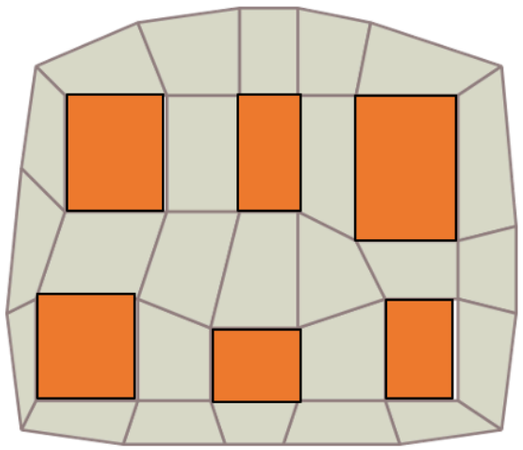


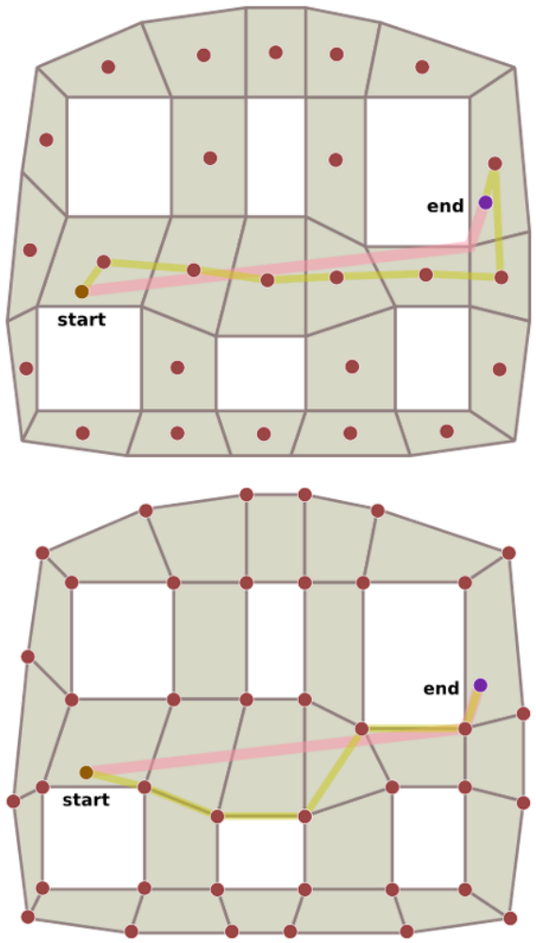


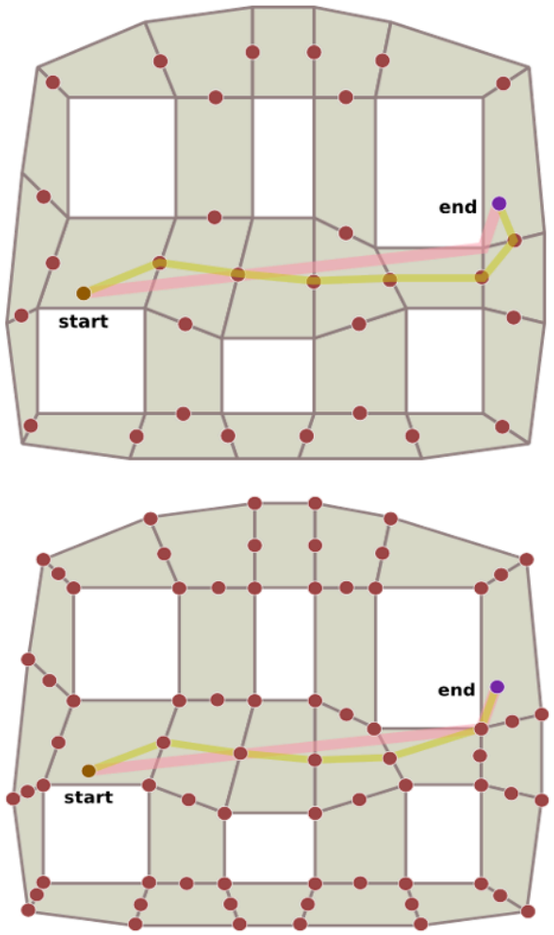


Cell shape, reference points

- *Meshes* can be used instead of uniform cells and different control points can be adopted for the graph







Discretization by spatial decomposition

Classification of spatial decomposition approaches:

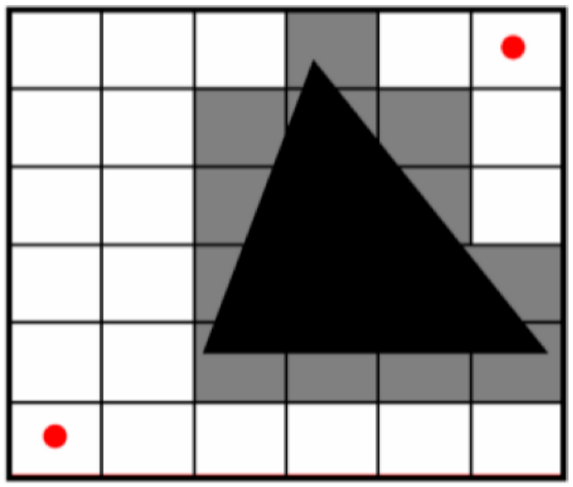
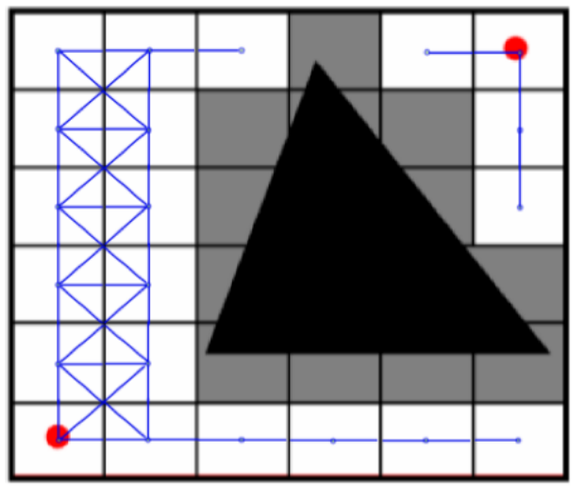
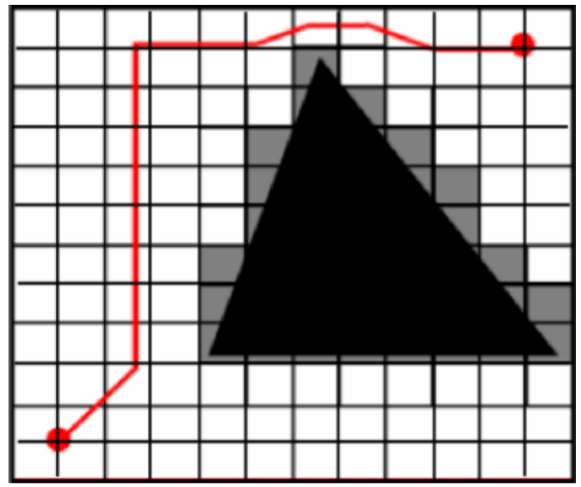
- ***Exact cell decomposition:*** Cell boundaries are placed as a function of the structure of the environment, such that the decomposition is *lossless,* no parts of $\mathscr{C}_{\text{free}}$ are removed because of the decomposition

- ***Approximate cell decomposition:*** The decomposition is *lossy,* resulting in an approximation of $\mathscr{C}_{\text{free}}$, that removes parts that are potentially accessible

The way we perform decomposition matters for completeness!

The shortest path through cell centers is the optimal path on the graph!

Approximate cell decomposition

- Let's consider a spatial decomposition implemented using a uniform grid of cells, all of the same shape (e.g., squares)

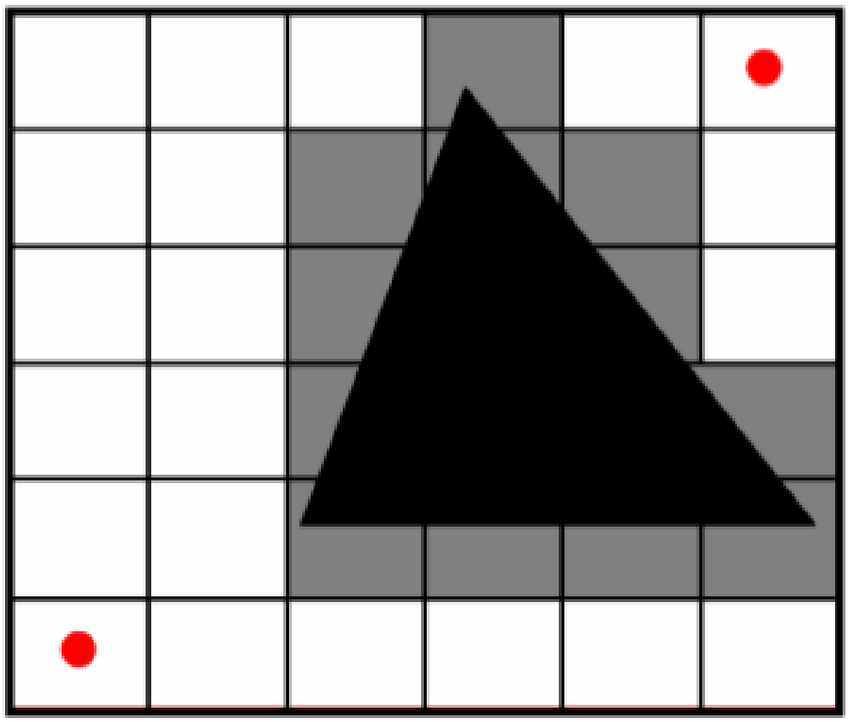
1. A discrete grid is defined over the entire C-space using a predefined shape
2. Any cell intersecting an obstacle is marked as BLOCKED, as FREE otherwise
3. A connectivity graph is constructed by placing a vertex in (the middle of) each cell and connecting the adjacent FREE cells
4. Given a query, the solution is the shortest path on the resulting connectivity graph

➡ The effective portion of C-space tagged as C_obs results enlarged because of discretization process → Resulting connectivity graph is an approximate roadmap

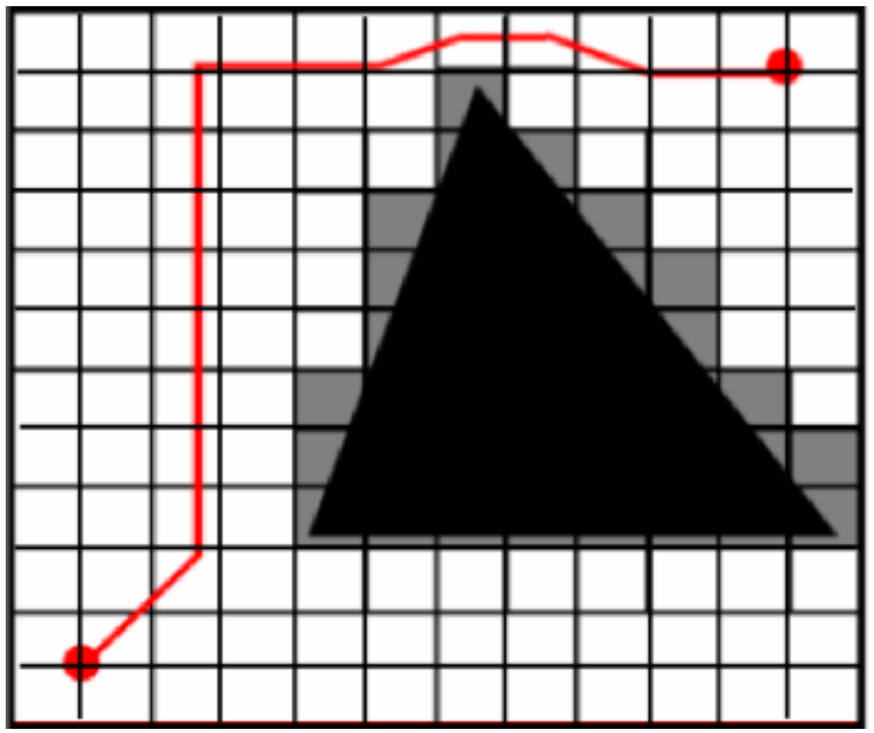
$\{0,1\}$ grid → Bitmap occupancy grid, $[0,1]$ grid → Probabilistic occupancy grid

Not complete

- Approximate cell decomposition is **easy to construct** (linear in the number of cells) and allows the use of efficient search algorithms specialized for grid representations, like the wavefront search algorithms (or *grassfire algorithms*)
- **Problem:** Even if a solution exists, the algorithm might report a failure, depending on the grid size → Narrow passages problems



With this cell size an admissible path does not exist



With this resolution the path exists, but the number of cells has greatly increased, and the connectivity graph with it

- A variable cell size can be a good trade-off →