ECE276B: Planning & Learning in Robotics
Lecture 5: Configuration Space

Lecturer:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Tianyu Wang: tiw161@eng.ucsd.edu

Yongxi Lu: yol070@eng.ucsd.edu

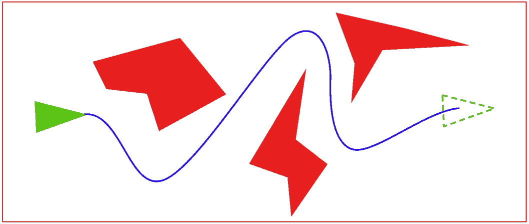
UC San Diego

JACOBS SCHOOL OF ENGINEERING

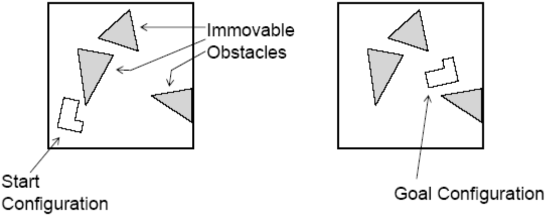
Electrical and Computer Engineering

The Shortest Path Problem and Motion Planning

- The shortest path (SP) problem is closely related to motion planning in robotics

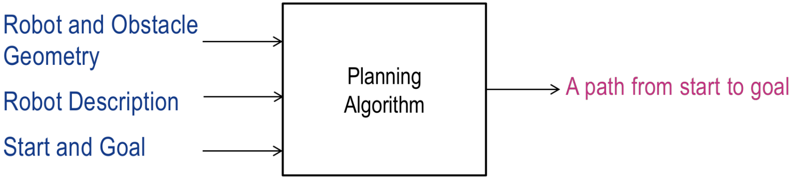


- We discussed a finite-space formulation of the SP problem but robot motion planning frequently requires continuous state and control spaces (also known as the **Piano Movers Problem**)

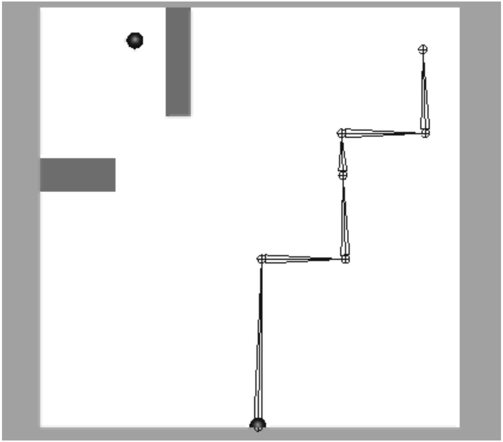


What is Motion Planning?

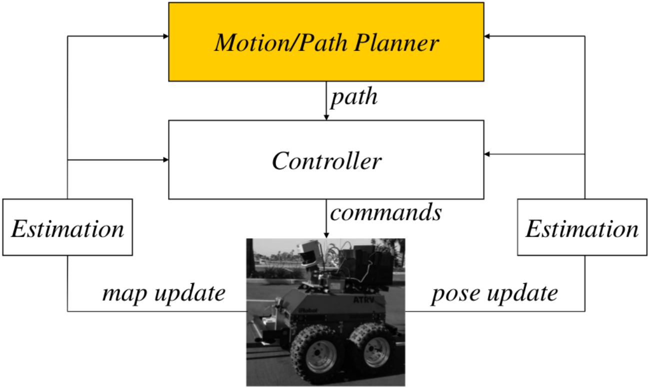
- Objective: find a feasible (and cost-minimal) path from the current configuration of the robot to its goal configuration
- Cost function: distance, time, energy, risk, etc.
- Constraints:
  - environmental constraints (e.g., obstacles)
  - dynamics/kinematics constraints of the robot



Example



Planning vs Control



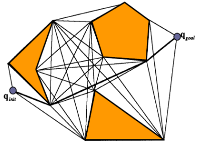
- Historical distinction between planning (global reasoning) and control (local reasoning)
  - **Planning**: the automatic generation of global collision-free trajectories
  - **Control**: the automatic generation of control inputs for local, reactive trajectory tracking
- Nowadays both interpreted as optimal control/reinforcement learning

Analyzing Motion Planning Algorithms

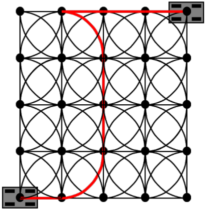
- **Completeness**: a planning algorithm is called complete if it:
  - returns a feasible solution, if one exists;
  - returns FAIL in finite time, otherwise
- **Optimality**:
  - a planning is optimal if it returns a path with shortest length J^* among all possible paths from start to goal
  - a planning algorithm is ϵ**-suboptimal** if it returns a path with length $J \leq \epsilon J^*$ for $\epsilon \geq 1$ and J^* - the optimal length
- **Efficiency**: a planning algorithm is efficient if it finds a solution in the least possible time (for all inputs)
- **Generality**: can handle high-dimensional robots or environments and various obstacle or dynamics/kinematics constraints

Motion Planning Approaches

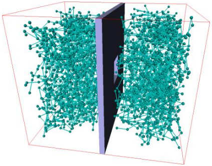
- **Exact algorithms** in continuous space
  - Either find a solution or prove none exist
  - Very computationally expensive
  - Unsuitable for high-dimensional spaces



- **Search-based Planning**
  - Discretize the configuration space into a graph
  - Solve the SP problem via a LC algorithm
  - Computationally expensive in high-dim spaces
  - Resolution completeness and suboptimality guarantees



- **Sampling-based Planning**
  - Sample the configuration space to construct a graph incrementally and construct a path from the samples
  - Efficient in high-dim spaces but problems with "narrow passages"
  - Weak completeness and optimality guarantees



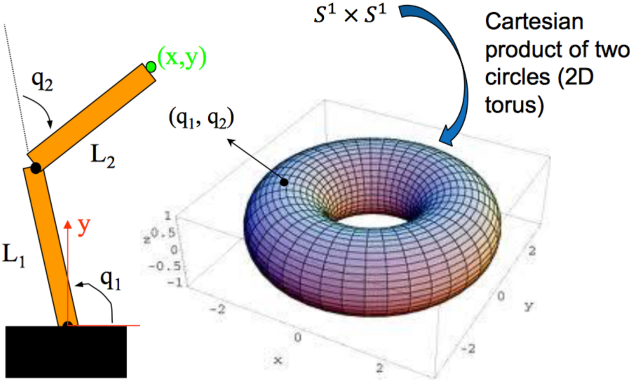
Configuration Space

- A **configuration** is a specification of the position of **every** point on a robot.
- A configuration q is usually expressed as a vector of the Degrees Of Freedom (DOF) of the robot:

$$q = (q_1, \ldots, q_n)$$

  - 3 DOF: Differential drive robot $(x, y, \theta) \in SE(2)$
  - 6 DOF: Quadrotor $(p, R) \in SE(3)$
  - 7 DOF: 7-link manipulator (humanoid arm): $(\theta_1, \ldots, \theta_7) \in [-\pi, \pi)^7$
- **Configuration space** C is the set of all possible robot configurations. The dimension of C is the minimum number of DOF needed to completely specify a robot configuration.

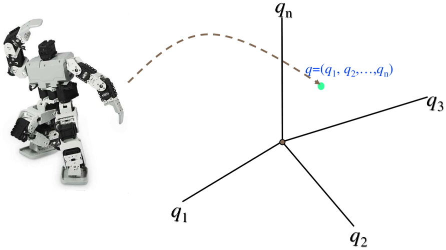
Example: C-Space of a Two Link Manipulator

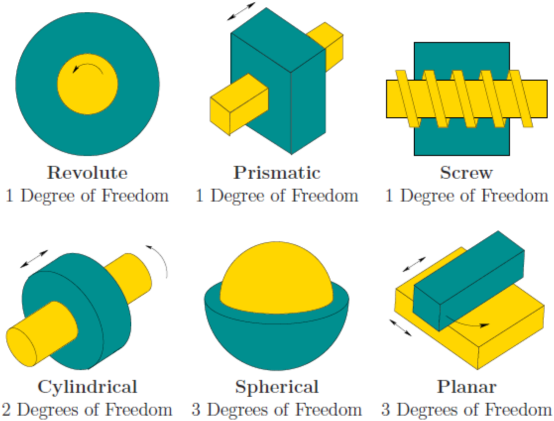


Topology of Configuration Space

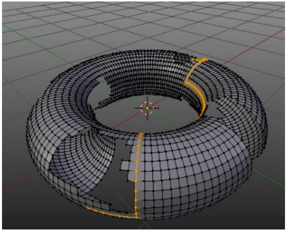
Degrees of Freedom of Robots with Joints

- An **articulated object** is a set of rigid bodies connected by joints.
- Examples of articulated robots: arms, humanoids

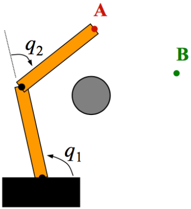




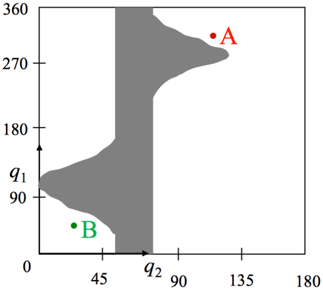
Obstacles in C-Space



- A configuration q is collision-free, or **free**, if the robot placed at q does not intersect any obstacles in the workspace
- The **free space** $C_{free} \subseteq C$ is the set of all free configurations
- The **occupied space** $C_{obs} \subseteq C$ is the set of all configurations in which the robot collides either with an obstacle or with itself (self-collision)



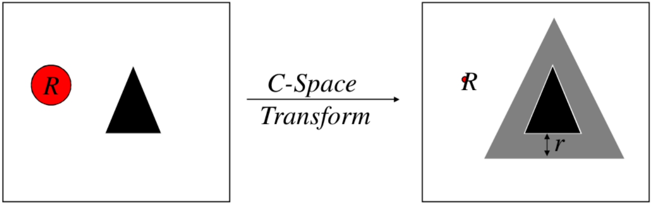
An obstacle in the robot's workspace



The C-space representation

How do we compute C_{obs}?

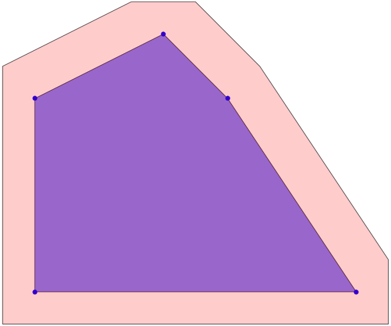
- **Input**: polygonal robot body R and polygonal obstacle O in environment
- **Output**: polygonal obstacle CO in configuration space
- **Assumption**: the robot translates only
- **Idea**:
  - Circular robot: expand all obstacles by the radius of the robot
  - Symmetric robot: Minkowski (set) sum
  - Asymmetric robot: Minkowski (set) difference

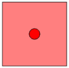


C_{obs} for Symmetric Robots

- The obstacle CO in C-Space is obtained via the Minkowski sum of the obstacle set O and the robot set R:

$$CO = O \oplus R := \{a + b \mid a \in O, b \in R\}$$



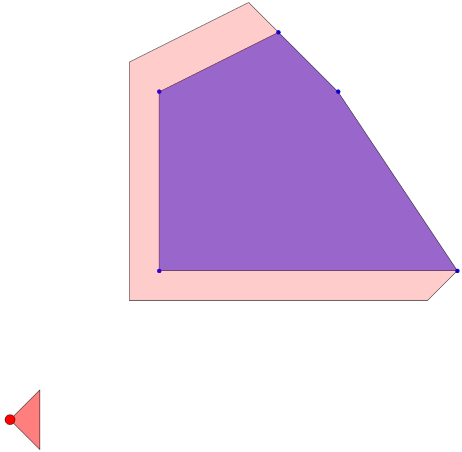


C_{obs} for Asymmetric Robots

- In the general case when the robot is not symmetric about the origin, it turns out that the correct operation is the **Minkowski difference**:

$$CO = O \ominus R := \{a - b \mid a \in O, b \in R\}$$

- This means "flip" the robot and then take Minkowski sum

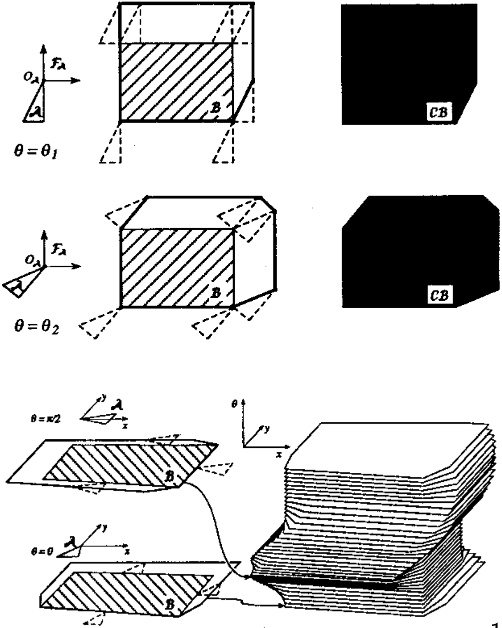


Properties of C_{obs}

- Properties of C_{obs}
  - If O and R are **convex**, then C_{obs} is **convex**
  - If O and R are **closed**, then C_{obs} is **closed**
  - If O and R are **compact**, then C_{obs} is **compact**
  - If O and R are **algebraic**, then C_{obs} is **algebraic**
  - If O and R are **connected**, then C_{obs} is **connected**
- After a C-Space transform, planning can be done for a point robot
  - **Advantage**: planning for a point robot is very efficient
  - **Disadvantage**: need to transform the obstacles every time the map is updated (e.g., if the robot is circular, $O(n)$ methods exist to compute distance transforms)
  - **Disadvantage**: very expensive to compute in higher dimensions
  - **Alternative**: plan in the original space and only check configurations of interest for collisions

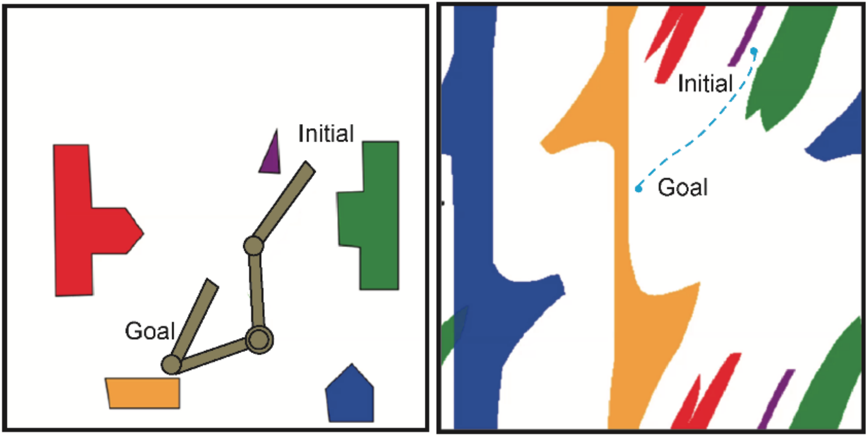
Minkowski Sums in Higher Dimensions

- The configuration space for a rigid non-circular robot in a 2D world is 3 dimensional



Configuration Space for Articulated Robots

- The configuration space for a N-DOF robot arm is N-dimensional
- Computing exact C-Space obstacles becomes complicated!



Workspace Configuration space

Motion Planning as Graph Search Problem

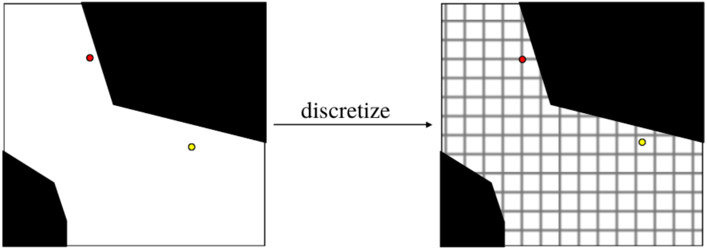
- Motion planning as a shortest path problem on a graph:
  1. Decide:
     a) pre-compute the C-Space
     b) perform collision checking on the fly
  2. Construct a graph representing the planning problem
  3. Search the graph for a (hopefully, close-to-optimal) path
- Often collision checking, graph construction, and planning are all interleaved and performed on the fly

Graph Construction

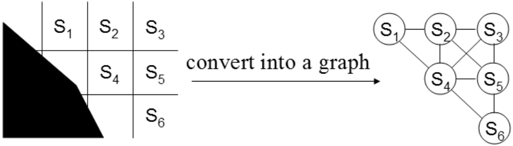
- **Cell decomposition**: decompose the free space into simple cells and represent its connectivity by the adjacency graph of these cells
  - X-connected grids
  - Tree decompositions
  - Lattice-based graphs
- **Skeletonization**: represent the connectivity of free space by a network of 1-D curves:
  - Visibility graphs
  - Generalized Voronoi diagrams
  - Other Roadmaps

X-connected Grid

1. Overlay a uniform grid over the C-space

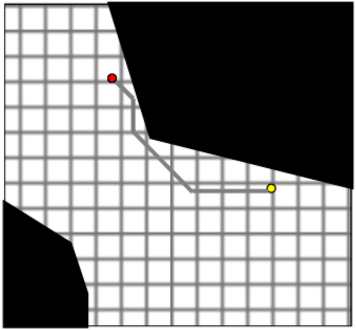


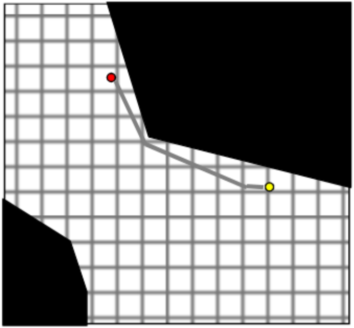
2. Convert the grid into a graph:



X-connected Grid

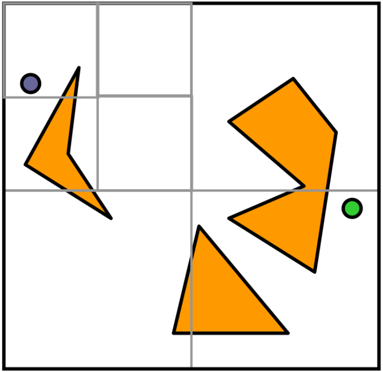
- How many neighbors?
  - 8-connected grid: paths restricted to 45° turns
  - 16-connected grid: paths restricted to 22.5° turns
  - 3-D (x, y, θ) discretization of $SE(2)$

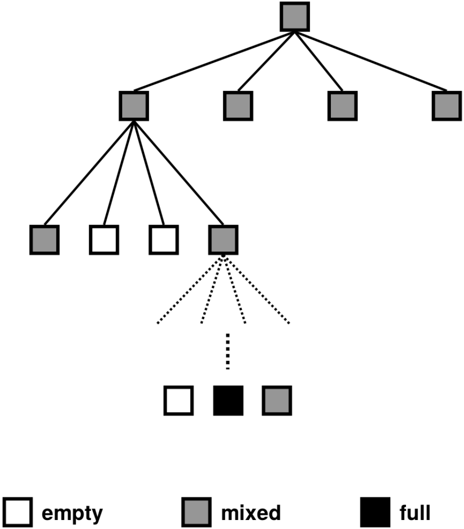




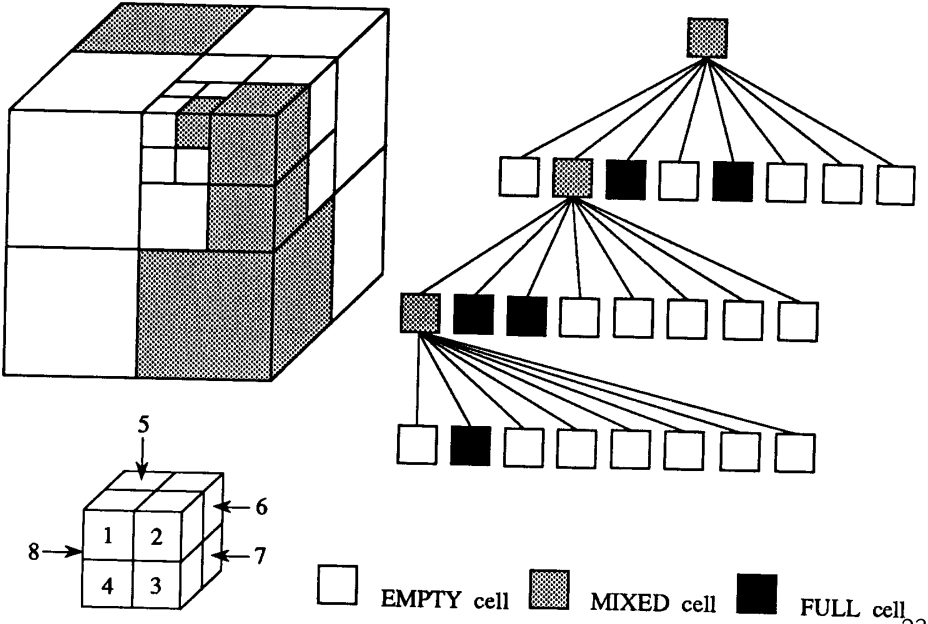
- Problems:
  1. What should we do with partially blocked cells?
  2. Discretization leads to a very dense graph in high dimensions and many of the transitions are difficult to execute due to dynamics constraints

Quadtree Adaptive Decomposition



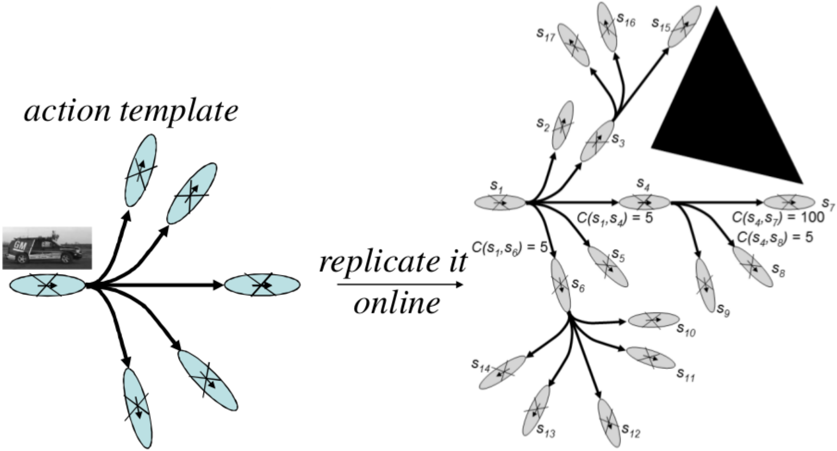


Octree Adaptive Decomposition



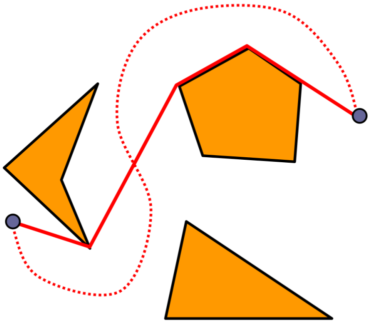
Lattice-based Graph

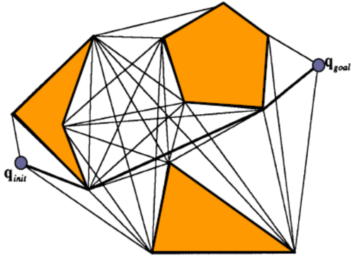
- Instead of dense discretization, construct a graph by a recursive application of a finite set of dynamically feasible motions (e.g., action template, motion primitive, movement primitive, macro action, etc.)
- **Pros**: sparse graph, feasible paths
- **Cons**: possible incompleteness



Visibility Graph

- Shakey Project, SRI [Nilsson, 1969]
- Also called **Shortest Path Roadmap**
- **Shortest paths are like rubber-bands**: if there is a collision-free path between two points, then there is a piecewise linear path that bends only at the obstacle vertices.
- **Visibility Graph**:
  - Nodes: start, goal, and all obstacle vertices
  - Edges: between any two vertices that "see" each other, i.e., the edge does not intersect obstacles or is an obstacle edge





Visibility Graph Construction

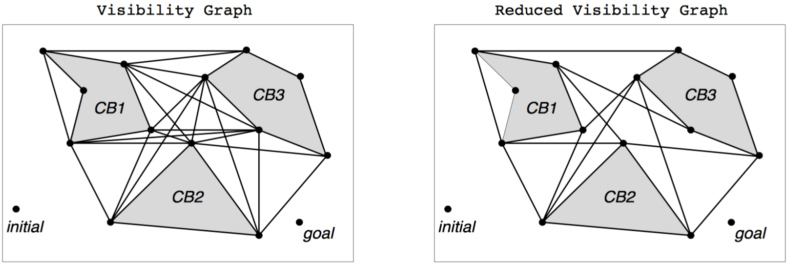
Algorithm 1 Visibility Graph Construction

```
1:  Input: q_I, q_G, polygonal obstacles
2:  Output: visibility graph G
3:  for every pair of nodes u, v do                         ▷ O(n^2)
4:      if segment(u, v) is an obstacle edge then           ▷ O(n)
5:          insert edge(u, v) into G
6:      else
7:          for every obstacle edge e do                    ▷ O(n)
8:              if segment(u, v) intersects e then
9:                  break and go to line 3
10:         insert edge(u, v) into G
```

- **Time complexity**: $O(n^3)$ but can be reduced to $O(n^2 \log n)$ with rotational sweep or even to $O(n^2)$ with an optimal algorithm
- **Space complexity**: $O(n^2)$

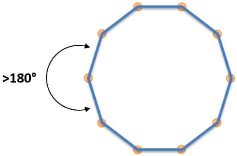
Reduced Visibility Graph

- In fact, not all edges are needed
- **Reduced visibility graph** – keep only edges between consecutive **reflex vertices** and **bitangents**
- A vertex of a polygonal obstacle is **reflex** if the exterior angle (computed in C_{free}) is larger than π
- A **bitangent edge** must touch two **reflex vertices** that are mutually visible from each other, and the the line must extend outward past each of them without poking into C_{obs}

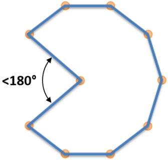


Reflex Vertices and Bitangents

- A vertex of a polygonal obstacle is **reflex** if the exterior angle (computed in C_{free}) is larger than π



(a) 10 reflex vertices

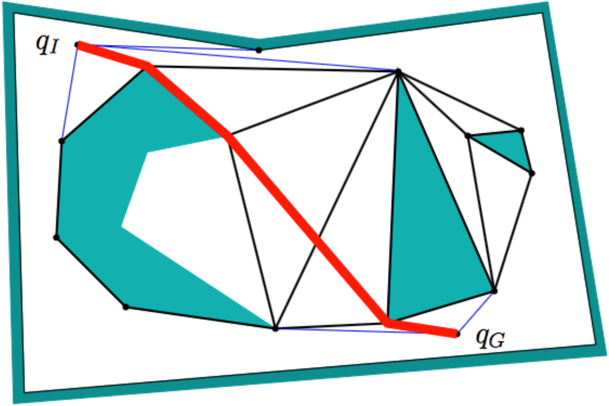


(b) 9 reflex vertices

- A **bitangent edge** must touch two **reflex vertices** that are mutually visible from each other, and the the line must extend outward past each of them without poking into C_{obs}

Reduced Visibility Graph

- The reduced visibility graph includes edges between consecutive reflex vertices on C_{obs} and bitangent edges
- The shortest path in a reduced visibility graph is the shortest path between start q_I and goal q_G

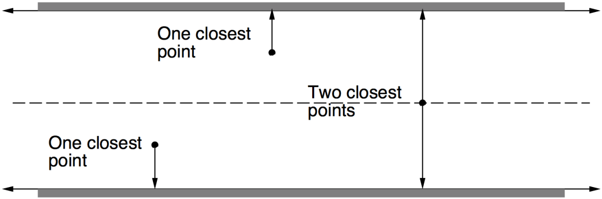


Visibility Graph

- What do we need to construct a reduced visibility graph?
  - Subroutine to check if a vertex is reflex
  - Subroutine to check if two vertices are visible
  - Subroutine to check if there exists a bitangent
- Pros:
  - independent of the size of the environment
  - can make multiple shortest path queries for the same graph, i.e., the environment remains the same but the start and goal change
- Cons:
  - **shortest paths always graze the obstacles**
  - hard to deal with a non-uniform cost function
  - hard to deal with non-polygonal obstacles
  - can get expensive in high dimensions with a lot of obstacles

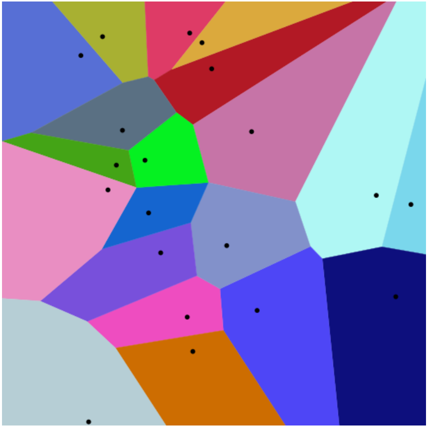
Generalized Voronoi Diagram

- **Voronoi diagram**: set of all points that are equidistant to two nearest obstacles
- Based on the idea of maximizing clearance instead of minimizing travel distance
- Also known as
  - maximum clearance roadmap (robotics)
  - skeletonization (computer vision)
  - retractions (topology)
- Suppose we have just two (linear) obstacles (e.g., a corridor). What is the set of points that keeps the robots as far away from the (C-Space) obstacles as possible?



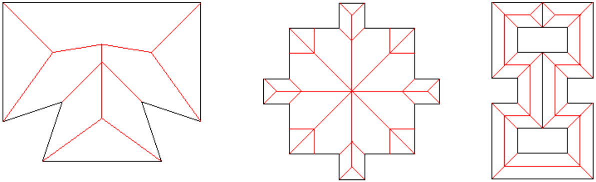
Voronoi Diagram

- Suppose we just have n point obstacles o_i
- The **Voronoi cell** of o_i is a subset of the plane that is closer to o_i than any other point
- Voronoi diagrams have many other applications, e.g., points represent fire stations and the Voronoi cells give their serving areas



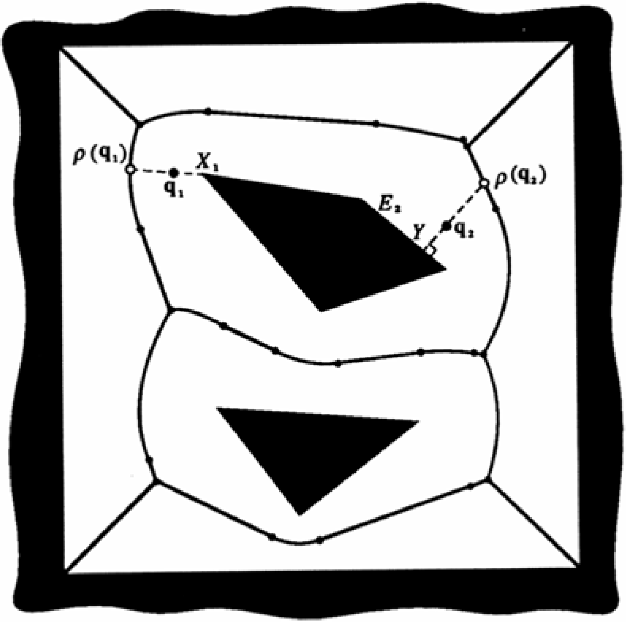
Voronoi Diagram

- Construction
  - Naive implementation: take every pair of obstacle features, compute locus of equally spaced points, and take the intersection
  - Efficient algorithms available, e.g., CGAL
  - Add a shortest path from start to the nearest segment of the diagram
  - Add a shortest path from goal to the nearest segment of the diagram



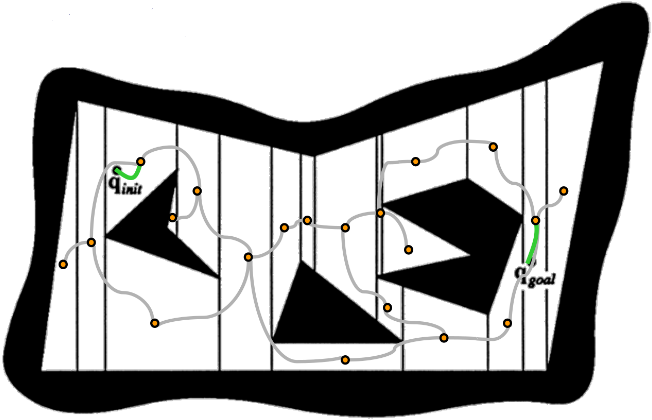
- Time complexity for n points in $\mathbb{R}^d$: $O(n \log n + n^{\lceil d/2 \rceil})$
- Space complexity: $O(n)$
- Pros:
  - paths tend to stay away from obstacles
  - independent of the size of the environment
- Cons:
  - difficult to construct in higher dimensions
  - can result in highly suboptimal paths

Voronoi Diagram



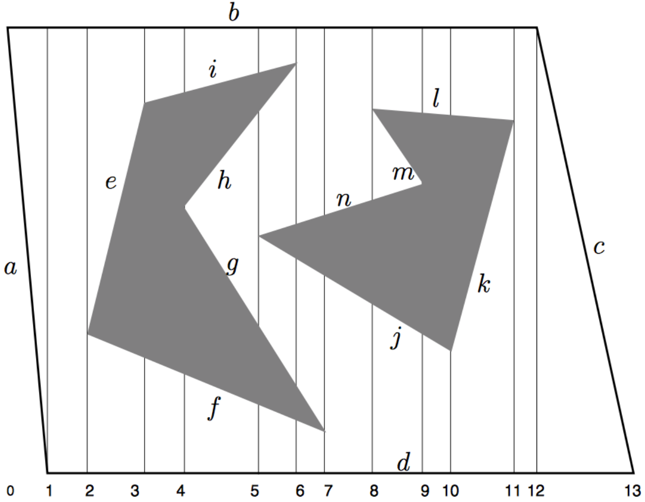
Trapezoidal Decomposition

- The free space C_{free} is represented by a collection of non-overlapping trapezoids whose union is exactly C_{free}:
- Draw a vertical line from every vertex until you hit an obstacle
  - **Nodes**: trapezoid centroids and line midpoints
  - **Edges**: between every pair of nodes whose cells are adjacent

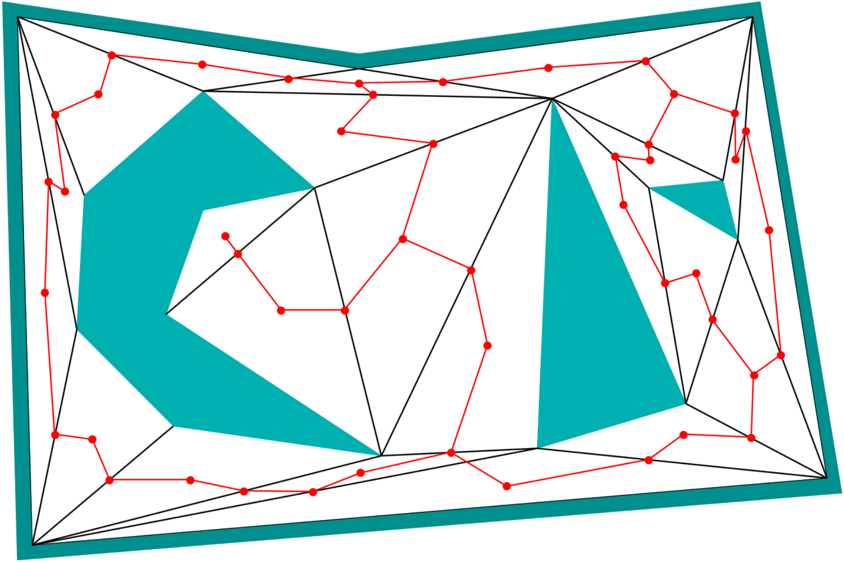


Cylindrical Decomposition

- Similar to trapezoidal decomposition, except the vertical lines continue after obstacles
- Generalizes better to high dimensions and complex configuration spaces



Triangular Decomposition



Probabilistic Roadmaps

- Construction:
  - Randomly sample valid configurations
  - Add edges between samples that are easy to connect with a simple local controller (e.g., follow straight line)
  - Add start and goal configurations to the graph with appropriate edges
- Pros and Cons:
  - Very popular: simple and highly effective in high dimensions
  - Can result in suboptimal paths, no guarantees on suboptimality
  - Difficulty with narrow passages