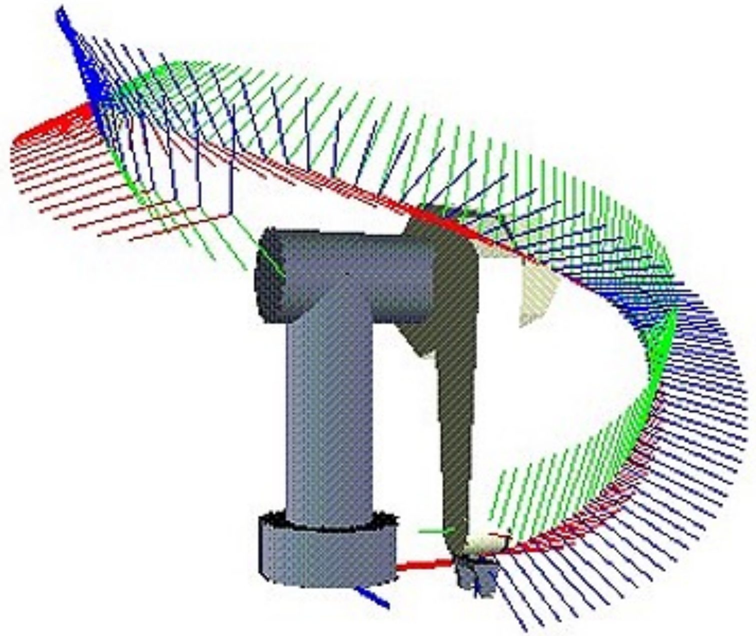
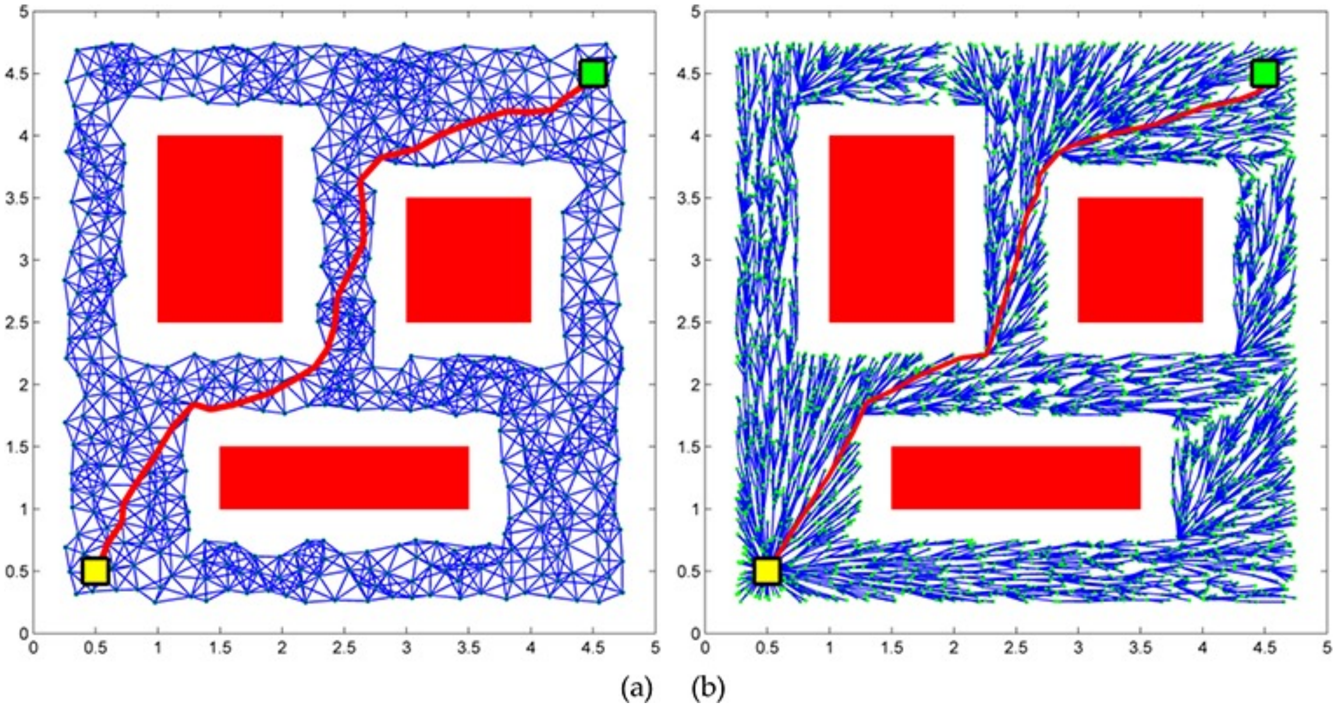
Motion Planning





Configuration space *C*-space

- Every point in the C-space *C* corresponds to a unique configuration q of the robot, and every configuration of the robot can be represented as a point in C-space.
- The configuration of a robot arm with n joints can be represented as a list of n joint positions, $\mathbf{q} = (\theta_1, \ldots, \theta_n)$.
- The free C-space *Cfree* consists of the configurations where the robot neither penetrates an obstacle nor violates a joint limit.
- The control inputs available to drive the robot are written as an m-vector $u \in U \subset Rm$, where $m = n$ for a typical robot arm.

State of the robot

The state of the robot is defined by its configuration and velocity, $x = (q, v) \in X$. For $q \in R^n$, typically we write $v = \dot{q}$.

The state x is simply the configuration q.

The notation q(x) indicates the configuration q corresponding to the state x, and $X_{free} = \{x \mid q(x) \in C_{free}\}$.

The equations of motion of the robot are written

$$\dot{x} = f(x, u)$$

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt. \qquad \text{Integral form}$$

Definition of Motion Planning

Given an initial state $x(0) = x_{start}$ and a desired final state x_{goal}, find a time T and a set of controls $u : [0,T] \rightarrow U$ such that the motion

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt.$$

satisfies $x(T) = x_{goal}$ and $q(x(t)) \in Cfree$ for all $t \in [0,T]$.

Types of Motion Planning Problems

- Path planning versus motion planning.
- Control inputs: $m = n$ versus $m < n$.
- Online versus offline
- Optimal versus satisficing
- Exact versus approximate
- With or without obstacles

Properties of Motion Planners

- Multiple-query versus single-query planning
- “Anytime” planning
- Completeness
- Computational complexity

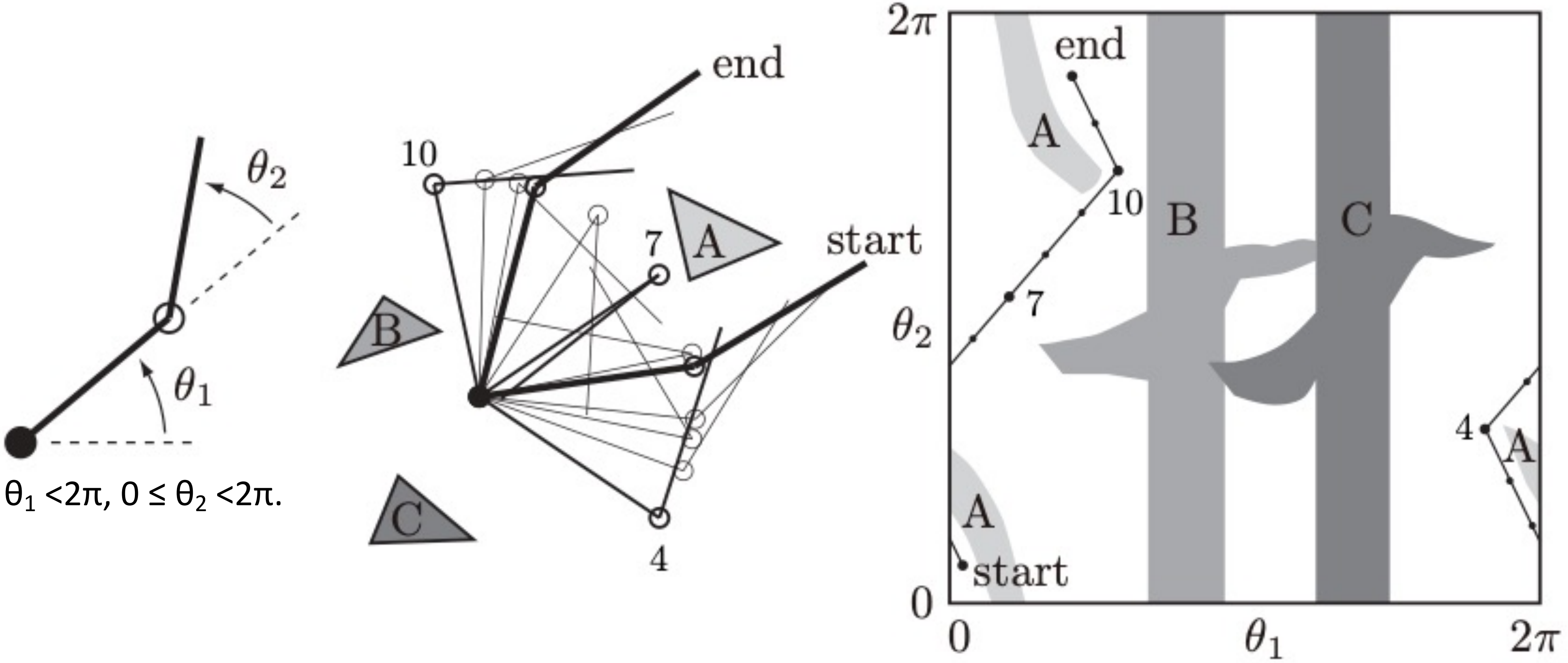
Motion Planning Methods

- Complete methods
- Grid methods
- Sampling methods
- Virtual potential fields
- Nonlinear optimization
- Smoothing

Configuration space

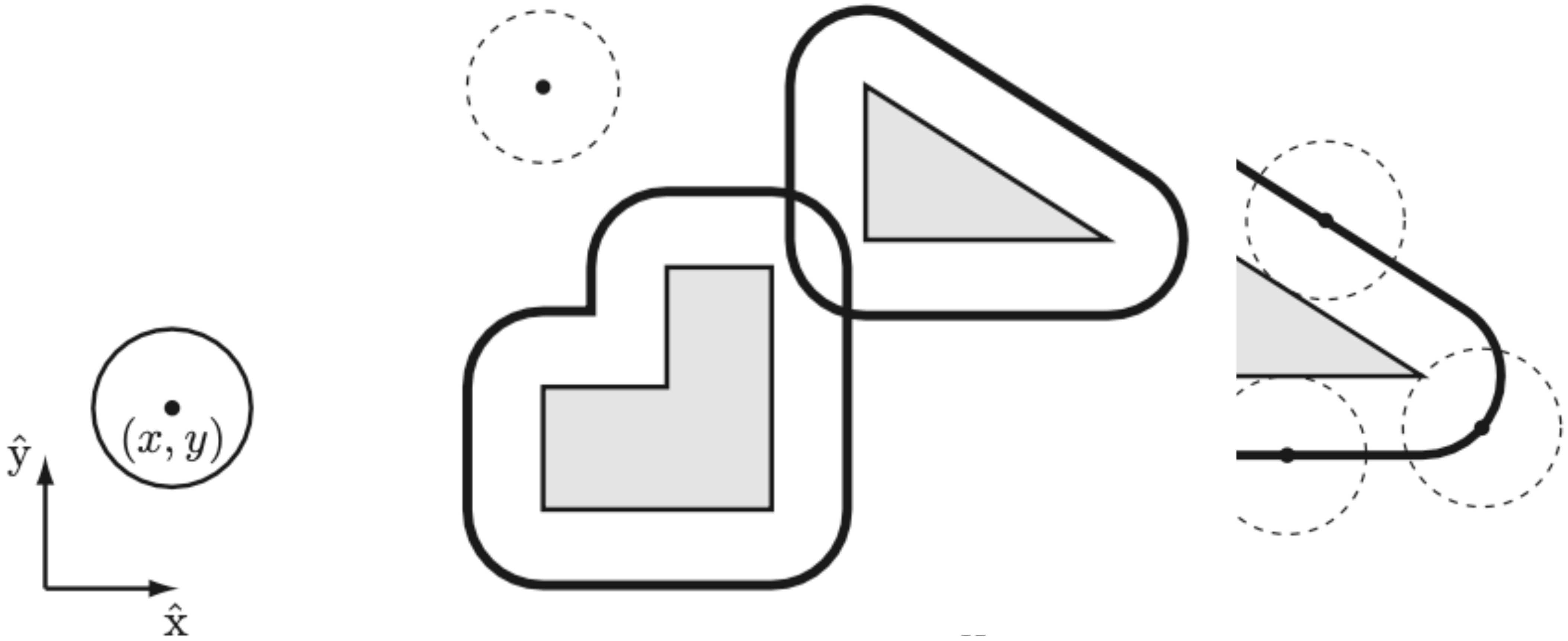
- The configuration space C considers two sets, the free space C_{free} and the obstacle space C_{obs}, where $C = C_{free} \cup C_{obs}$.
  - Joint limits are treated as obstacles in the configuration space.
- If the obstacles break C_{free} into separate connected components, and q_{start} and q_{goal} do not lie in the same connected component, then there is no collision-free path.

Planar arm

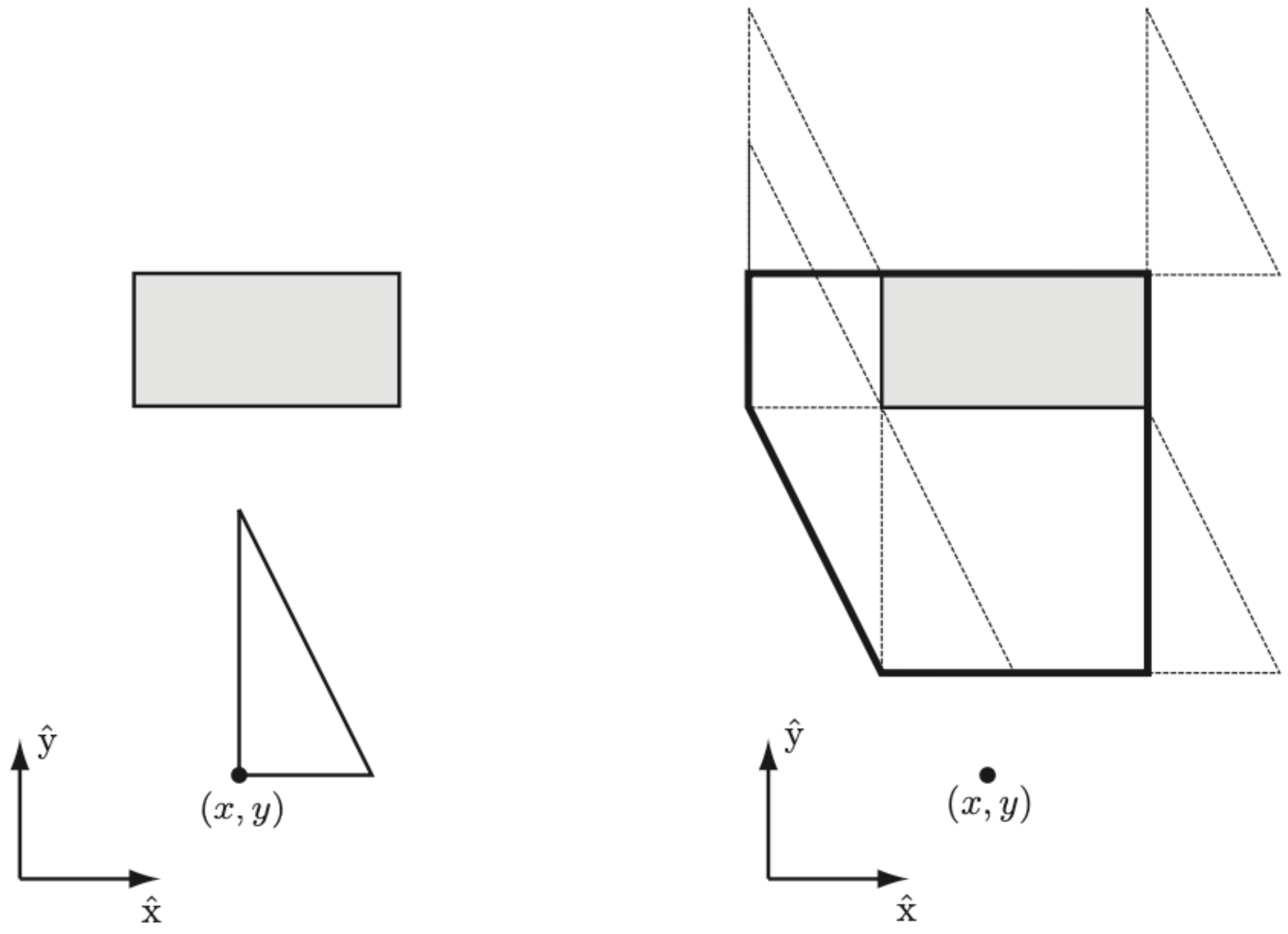


$0 \le \theta_1 < 2\pi,\ 0 \le \theta_2 < 2\pi.$

Circular mobile robot

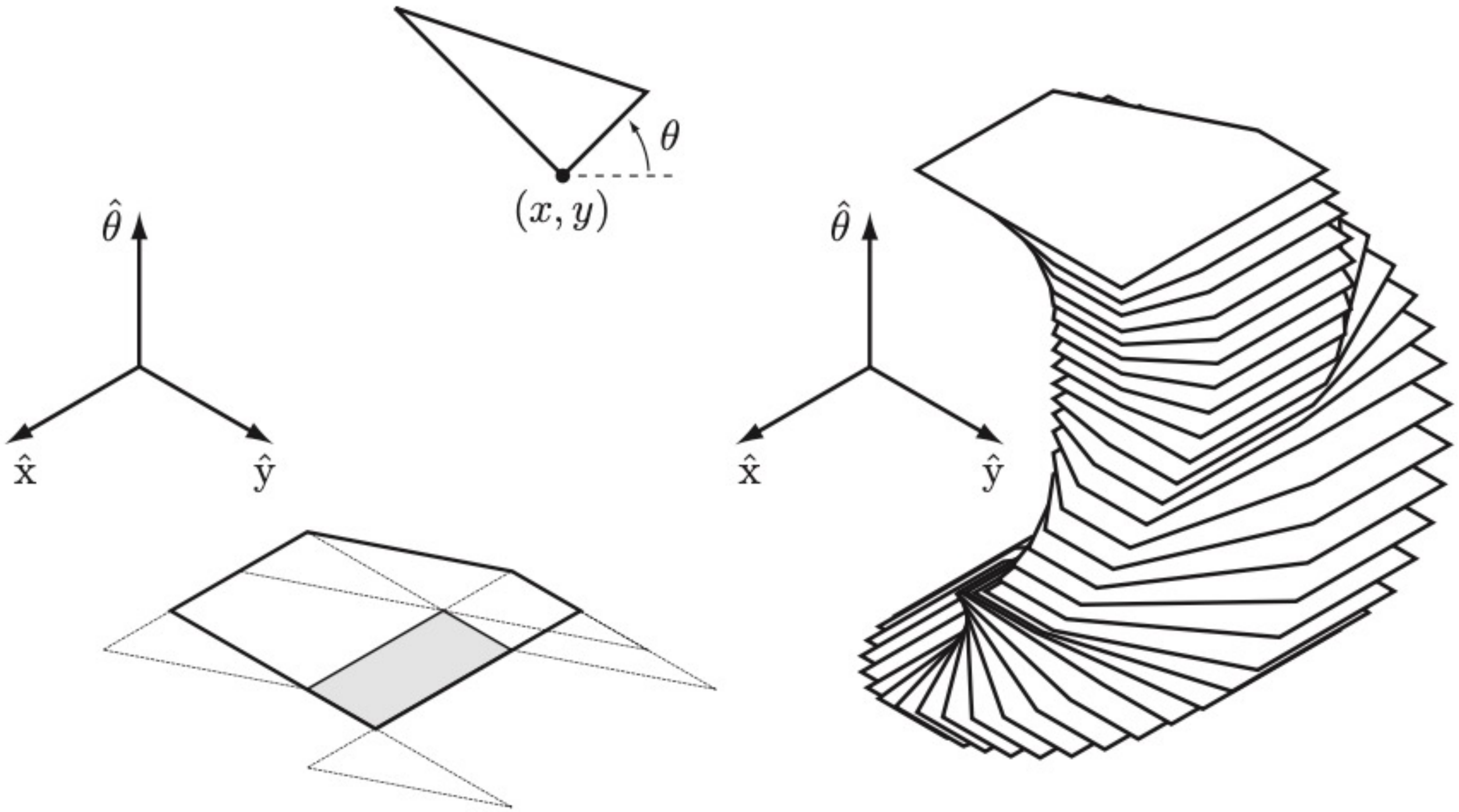


A Polygonal Planar Mobile Robot That Translates and Rotates



No rotation

A Polygonal Planar Mobile Robot That Translates and Rotates



10 slices with $\Delta\theta$=10°

Becomes a tridimensional space problem (x,y,θ)

Distance to Obstacles and Collision Detection

Given a C-obstacle B and a configuration q, let d(q, B) be the distance between the robot and the obstacle, where

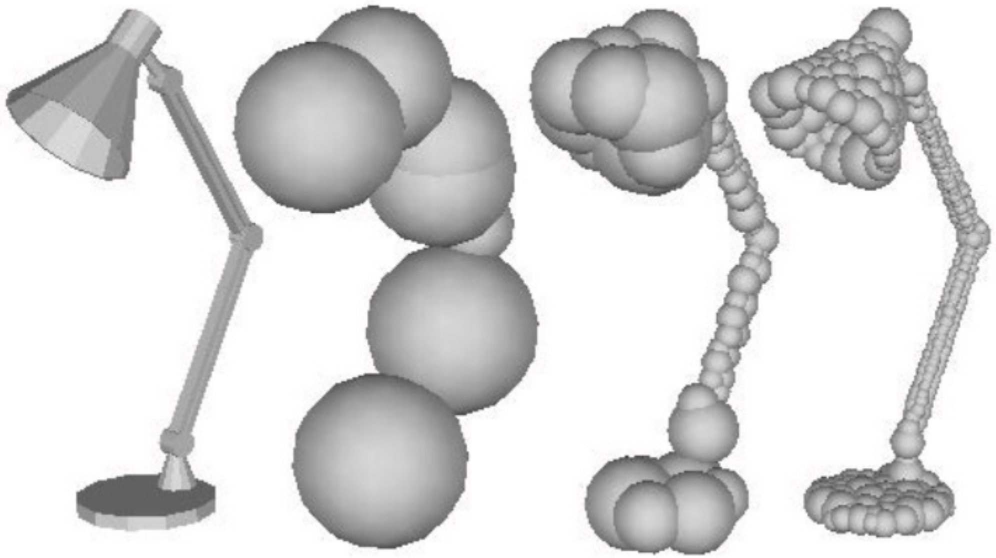
$$
\begin{aligned}
d(q, \mathcal{B}) &> 0 \qquad \text{(no contact with the obstacle)}, \\
d(q, \mathcal{B}) &= 0 \qquad \text{(contact)}, \\
d(q, \mathcal{B}) &< 0 \qquad \text{(penetration)}.
\end{aligned}
$$

Use the Euclidean distance

A collision–detection routine determines whether $d(q, B) \leq 0$ for any C-obstacle Bi. A collision-detection routine returns a binary result and may or may not utilize a distance-measurement algorithm at its core.

One popular distance-measurement algorithm is the Gilbert–Johnson–Keerthi (GJK) algorithm.

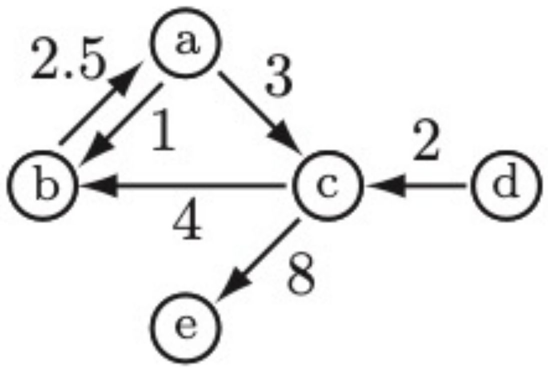
Approximation with spheres



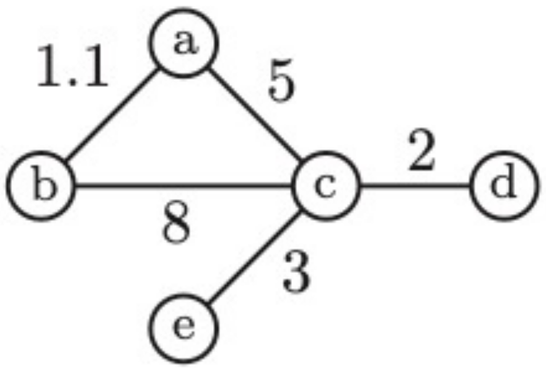
Given a robot at q represented by k spheres of radius R_i centered at $r_i(q)$, $i = 1, . . . , k$, and an obstacle B represented by l spheres of radius B_j centered at b_j, $j = 1,\ldots, l$ the distance between the robot and the obstacle can be calculated as

$$d(q,B)=\min\|r_i(q)-b_j\|-R_i-B_j.$$

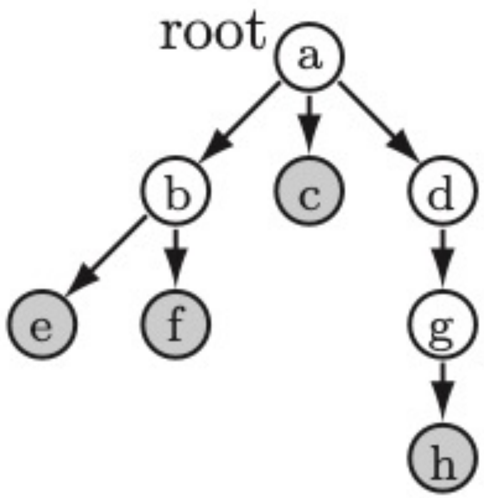
Graph and trees



Weighted diagraph



Weighted undirected graph



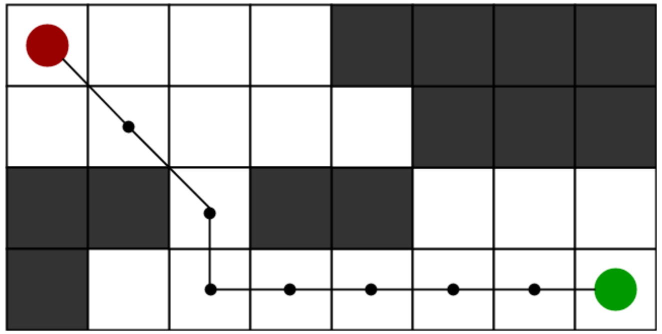
Tree

Graph search

- A* search

Algorithm 10.1 A^* search.

```
OPEN ← {1}
past_cost[1] ← 0, past_cost[node] ← infinity for node ∈ {2, ..., N}
while OPEN is not empty do
  current ← first node in OPEN, remove from OPEN
  add current to CLOSED
  if current is in the goal set then
    return SUCCESS and the path to current
  end if
  for each nbr of current not in CLOSED do
    tentative_past_cost ← past_cost[current]+cost[current,nbr]
    if tentative_past_cost < past_cost[nbr] then
      past_cost[nbr] ← tentative_past_cost
      parent[nbr] ← current
      put (or move) nbr in sorted list OPEN according to
          est_total_cost[nbr] ← past_cost[nbr] +
                heuristic_cost_to_go(nbr)
    end if
  end for
end while
return FAILURE
```



Other search methods

- Dijkstra’s algorithm
- Breadth-first search
- Suboptimal A* search