Introduction to Robot Motion Planning & Navigation Module 5

Solmaz S. Kia (solmaz.eng.uci.edu)

solmaz@uci.edu

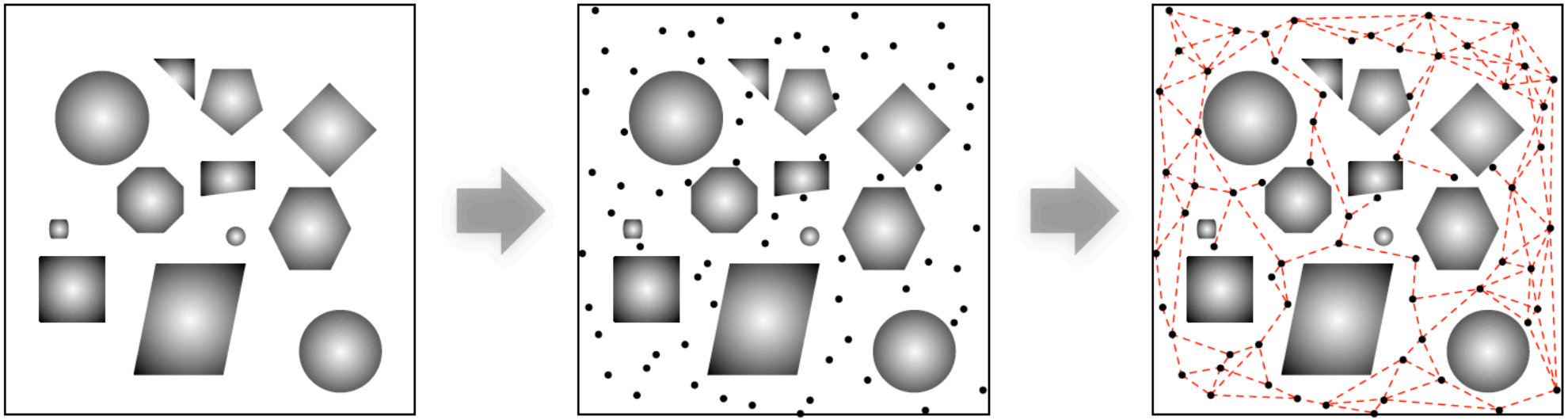
Mechanical and Aerospace Engineering Department

University of California Irvine

Chapter 5: Motion Planning via Sampling

- general roadmaps and their desirable properties,
- complete planners based on exact roadmap computation (specifically, we will review decomposition-based roadmaps and will introduce a novel shortest-paths visibility graph),
- shortest-path planning via visibility roadmap and shortest path search via Dijkstra's algorithm.
- general-purpose planners based on sampling and approximate roadmaps. (Sampling-based Roadmap Methods). For this general-purpose planners we will discuss:
  - connection rules for fixed resolution grid-based roadmaps,
  - connection rules for arbitrary-resolution methods,
  - comparison between sampling-based approximate and exact planners
- incremental sampling-based planning methods, including:
  - from multi-query to single-query,
  - rapidly-exploring random trees (RRT),
  - the application of receding-horizon incremental planners to sensor-based planning.

Motion Planning via Sampling



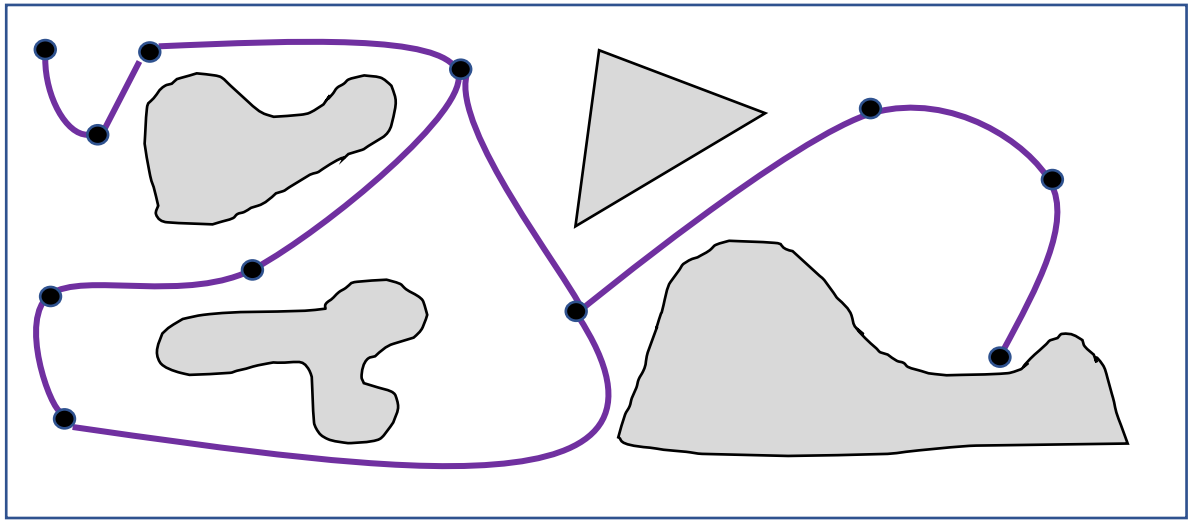
"cloud representation" of the free configuration space for robots and obstacles of arbitrary shape

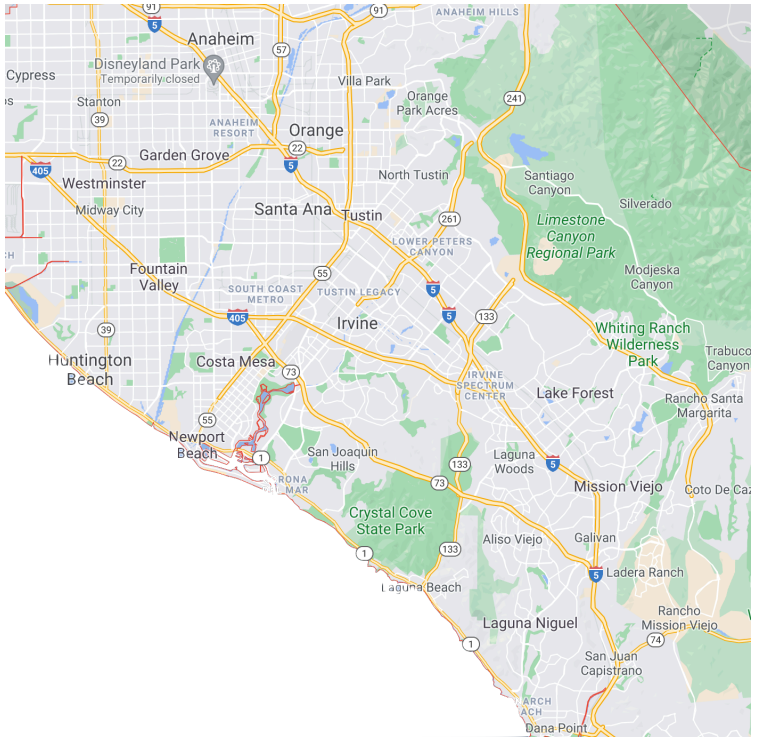
compute a roadmap from a cloud in the next chapter

Roadmaps

A roadmap is a collection of locations in the configuration space along with paths connecting them.

- With each path, we associate a positive weight that represents a cost for traveling along that path, for example, the path length or the travel time.
- Think of a roadmap as a weighted graph $G = (V, E, w)$, where w is a function that assigns the weight (e.g., path length) to each edge in E.





Roadmaps

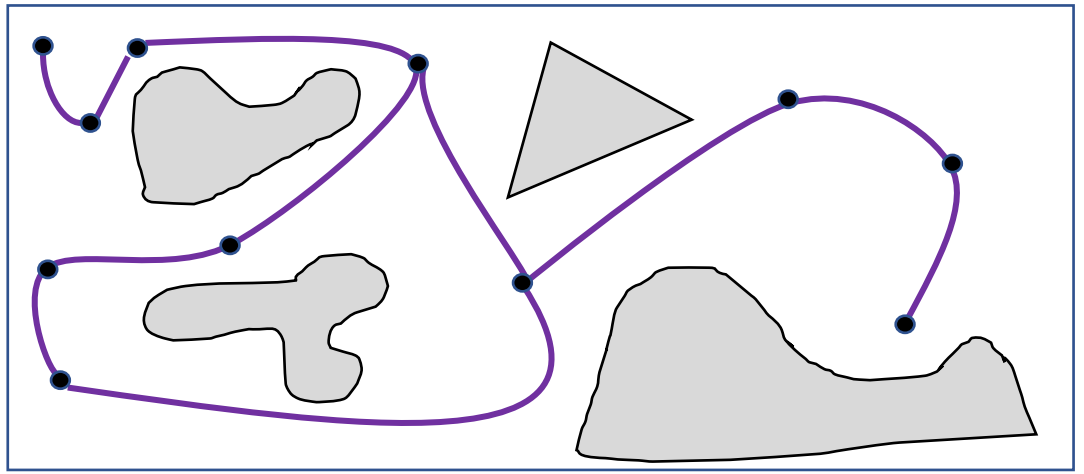
roadmap may have the following properties:

i. the roadmap is accessible if, for each point $q_{start} \in Q_{free}$, there is an easily computable path from q_{start} to some node in the roadmap,

ii. similarly, the roadmap is departable if, for each point $q_{goal} \in Q_{free}$, there is an easily computable path from some node in the roadmap to q_{goal}, and

iii. the roadmap is connectivity-preserving if, given a connected free configuration space (i.e., any two configurations in Q_{free} are connected by a path in Q_{free}), then any two locations of the roadmap are connected by a path in the roadmap,

iv. the roadmap is efficient with factor $\delta \geq 1$ if, for any two locations in the roadmap, say u and v, the path length from u to v along edges of the roadmap is no longer than δ times the length of the shortest path from u to v in the environment



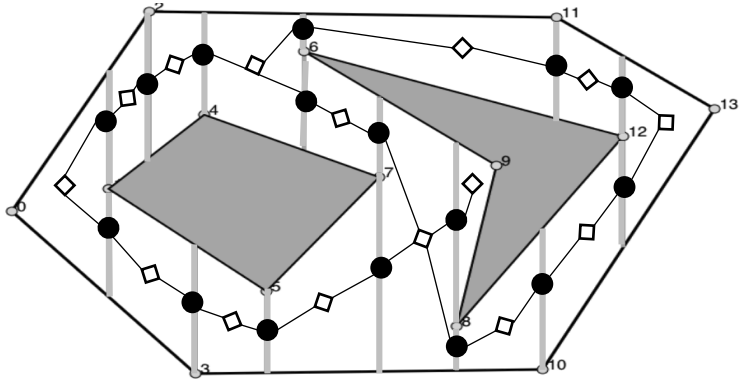
The notions of accessibility and departability are not fully specified as they depend upon the notion of "easily computable path."

Complete Planners on Exact Roadmaps

Decomposition-based roadmaps:

Example: sweeping trapezoidation algorithm

- guaranteed to be accessible and departable (via straight segments) and connectivity-preserving
- does not contain shortest paths among environment locations



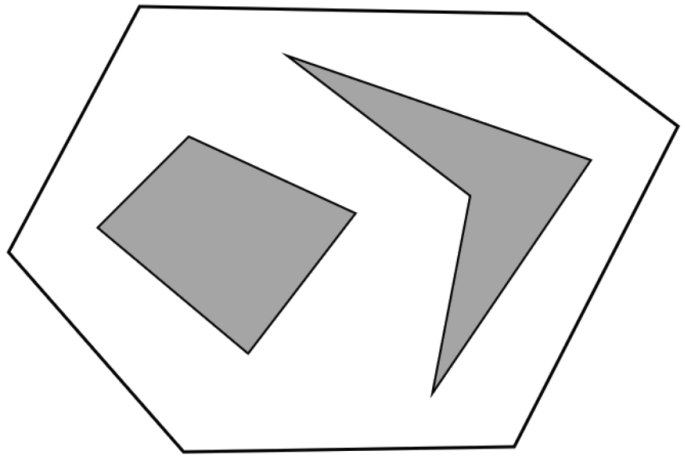
environments with polygonal obstacles.

Visibility roadmaps: the visibility graph $G = (V, E, w)$, is defined as

i. the nodes V of the visibility graph are all convex vertices of the polygons $O_1, \cdots, O_n$,

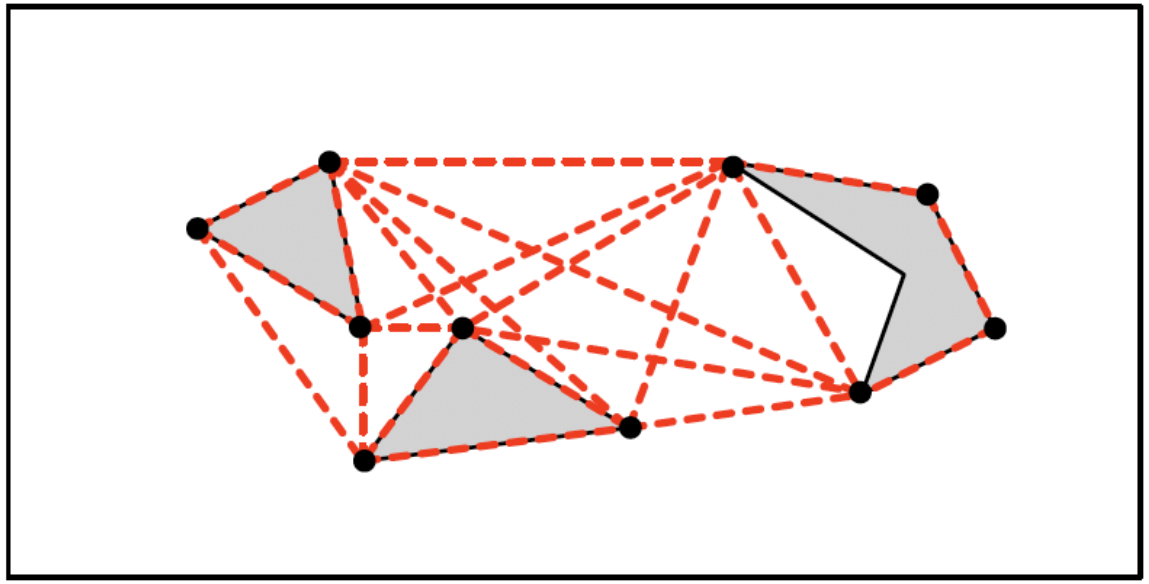
ii. the edges E of the visibility graph are all pairs of vertices that are visibly connected. That is, given $u, v \in V$, we add the edge $\{u, v\}$ to the edge set E if the straight-line segment between u and v is not in collision with any obstacle, and

iii. the weight of an edge $\{u, v\}$ is given by the length of the segment connecting u and v.



environments with polygonal obstacles.

Complete Planners on Exact Roadmaps



Computational complexity of visibility roadmap

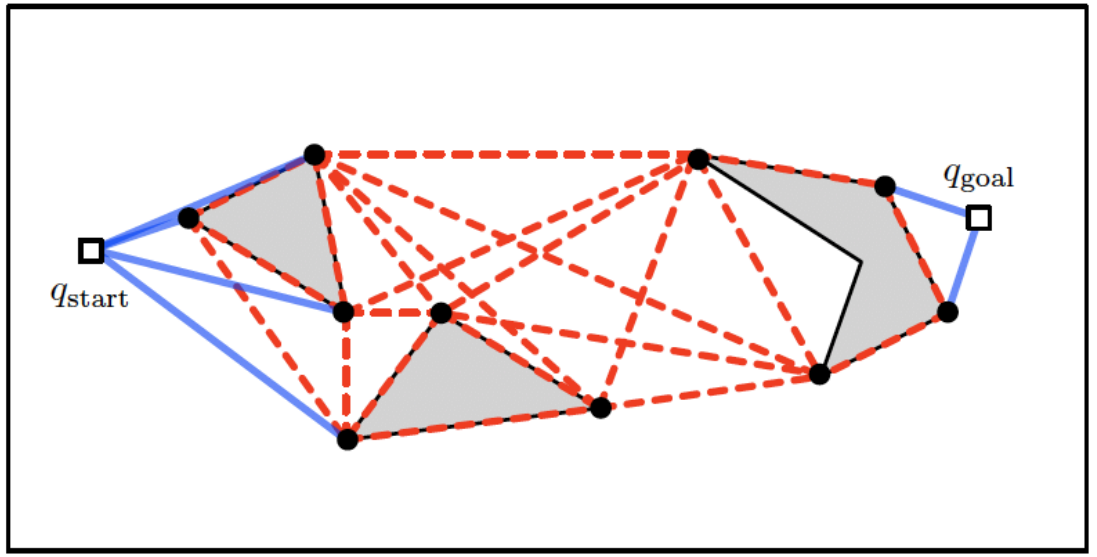
- Number of obstacle vertices is n, then the number of nodes in the visibility graph is at most n
- To compute the edges E, we need to check every pair of nodes $v_i, v_j \in V$ and see if the line segment intersects with any of the n obstacle edges.
- Thus, the graph can be computed in a total runtime of $O(n^3)$
- A more sophisticated implementation (de Berg et al., 2000) reduces this runtime to $O(n^2 \log(n))$

Complete Planners on Exact Roadmaps

Properties of visibility roadmap:

- If the free configuration space is connected, then the visibility graph is connected, departable, and accessible.

- The shortest path from start to goal is a path in the visibility graph. Hence, the roadmap obtained via the visibility graph is optimally efficient, in the sense that the efficiency factor δ is 1.



Theorem 5.1 (Shortest paths through polygonal obstacles) Consider a configuration space with polygonal configuration space obstacles.
Any shortest path in the free configuration space between q_{start} and q_{goal}

(i) consists of only straight line segments, and

(ii) has segments whose endpoints are either the start location q_{start}, the goal location q_{goal}, or an obstacle vertex (or, more precisely, a convex obstacle vertex if start and goal locations are not non-convex vertices).

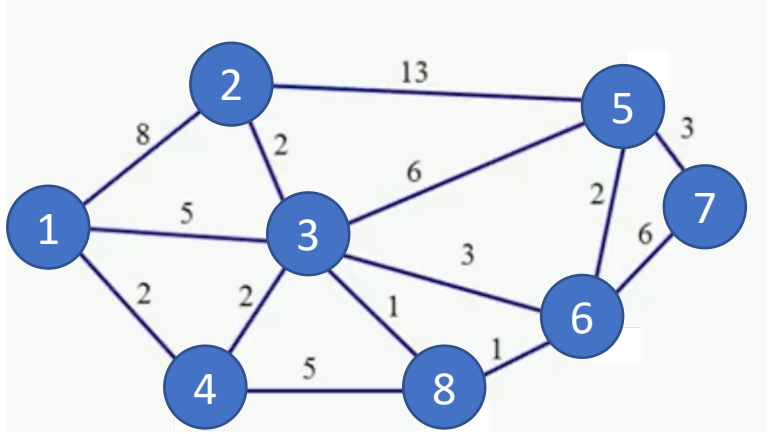
Shortest Paths in Weighted Graphs via Dijkstra's Algorithm

The minimum-weight path between two nodes, also called the shortest path in a weighted graph, is a path of minimum weight between the two nodes

Dijkstra's algorithm

Input: a weighted graph G and a start node v_{start}
Output: the `parent` pointer and `dist` values for each node in the graph G

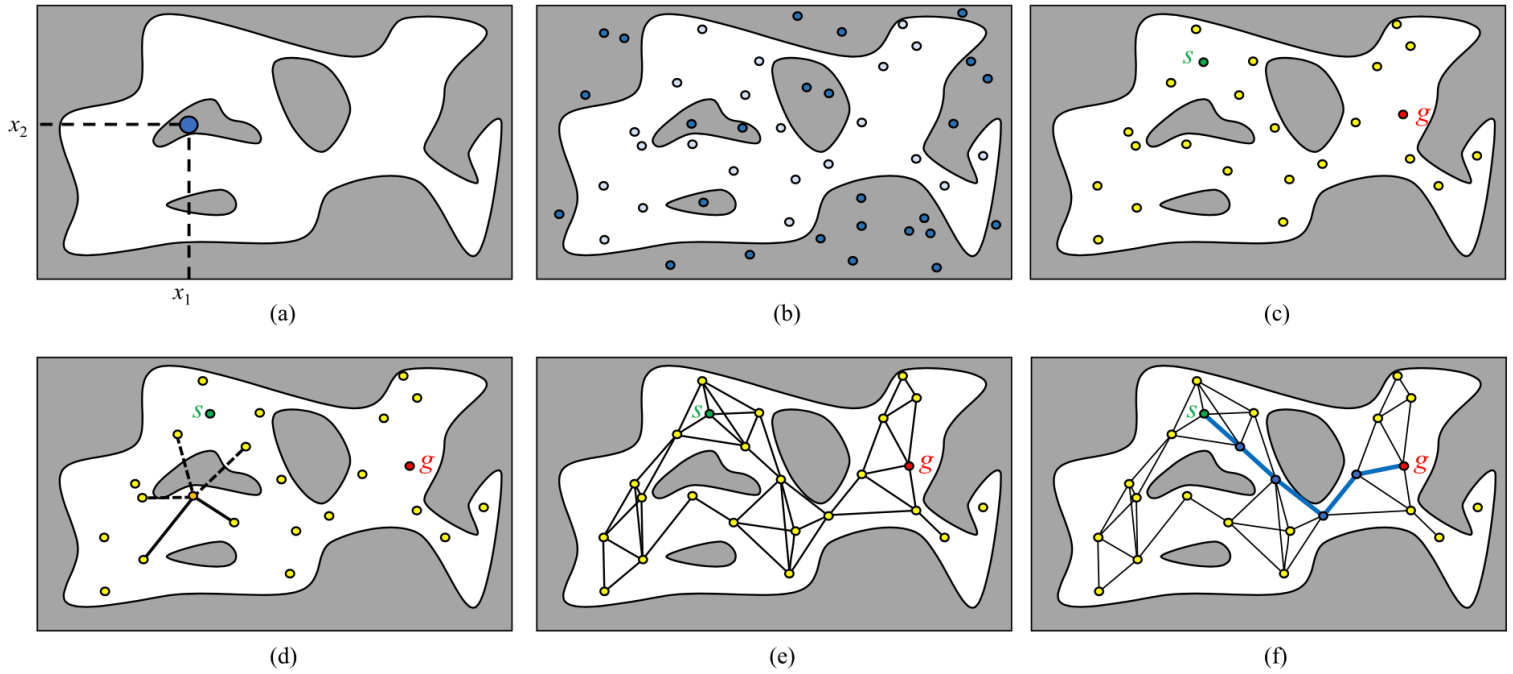
```
    // Initialization of distance and parent pointer for each node
 1: for each node v in G :
 2:       dist(v) := +∞
 3:       parent(v) := NONE
 4: dist(v_start) := 0
 5: parent(v_start) := SELF
 6: Q := set of all nodes in G
    // Main loop to grow the tree and update distance estimates
 7: while Q is not empty :
 8:       find node v in Q with smallest dist(v)
 9:       remove v from Q
10:       for each node w in Q connected to v by an edge :
11:            if dist(w) > dist(v) + weight(v, w) :
12:                 dist(w) := dist(v) + weight(v, w)
13:                 parent(w) := v
14: return parent values and dist values for all nodes v
```



- The shortest path tree as {parent(u), u} for each node u for which parent(u) is different from $+\infty$.
- Given a goal node v_{goal} we can use the parent values to reconstruct the sequence of nodes on the shortest path from v_{start} to v_{goal} using the The extract-path algorithm from Chapter 2 of Ref [1].

General-purpose Planners via Sampling-based Roadmaps

Probabilistic roadmaps (PRM) are an approximate roadmap of the robot's free space built by randomly sampling free connections and attempting connections between them. The roadmap can then be searched as usual for a path from the start to the goal.



The basic algorithm for constructing a PRM is:

1.Sample N configurations at random from the Q-space (plot a--b).

2.Add all feasible configurations and the start and goal to the roadmap. These are known as *milestones*. (plot c)

3.Test pairs of nearby milestones for visibility, and add visible edges to the roadmap. (plot d--e)

4.Search for a path from the start to the goal. (plot f)

It can be shown that the method is *probabilistically complete*,

- if it finds a path, the path will be feasible;
- if it does not find a path, then this answer might be incorrect;
- The chance of this incorrect answer decreases to as N increases, given some relatively mild assumptions on the shape of the free space.

General-purpose Planners via Sampling-based Roadmaps

probabilistic roadmap (PRM) algorithm

Input: number of sample points in roadmap $N \in \mathbb{N}$. Requires access to a sampling algorithm, collision detection algorithm, and a notion of neighborhood in Q

Output: a roadmap (V, E) for the free configuration space Q_{free}

initialize (V, E) to be the empty graph
// *compute a set of locations V in Q_{free}, via sampling & collision detection*
while number of nodes in V is less than N :
    compute a new sample q in the configuration space Q
    if the configuration q is collision-free :
        $V := V \cup \{q\}$
// *compute a set of paths E via the following connection rule*
for each sampled location q in V :
    for all other sampled locations p in a neighborhood of q :
        if the path from q to p hits no obstacle :
            $E := E \cup \{\text{path from } p \text{ to } q\}$
return (V, E)

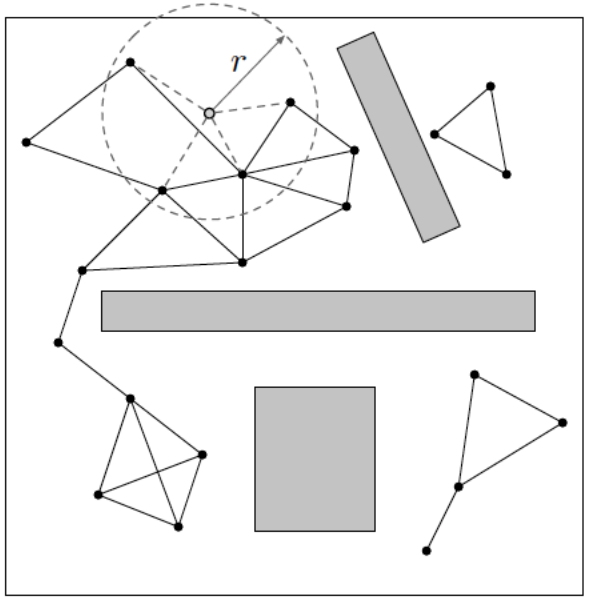
Neighborhood Functions

PRM: a motion planning method based on computing a roadmap for the free configuration space Q_{free} via

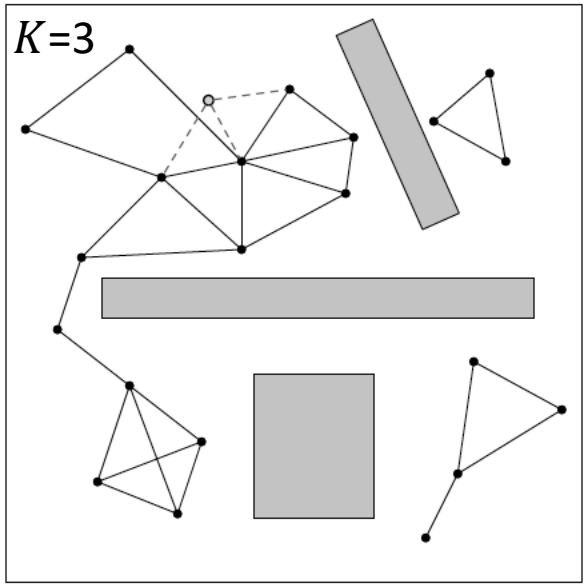
(1) sampling,
(2) collision detection, and
(3) a so-called connection rule.

A connection rule is an algorithm that decides when and how to (try to) compute a path connecting two nodes of a sampled configuration space.

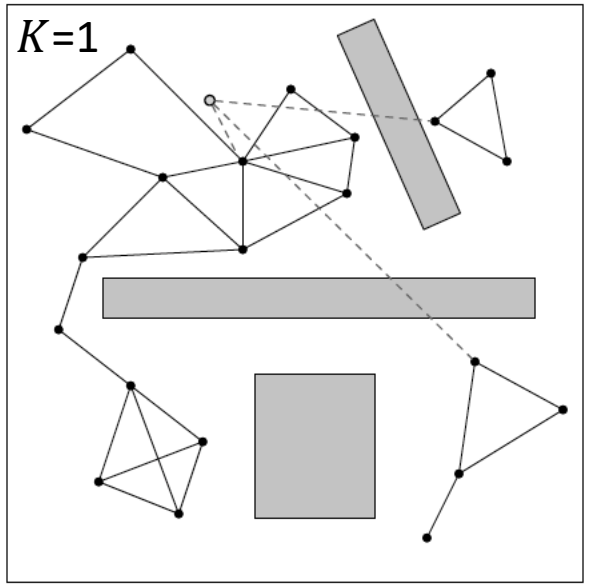
A significant computational expense in PRM and its variants, is computing nearest-neighbors (and near-neighbors).



r-radius rule: fix a radius $r > 0$ and select all locations p within distance r of q.



K-closest rule: select the K closest locations p to q.



component-wise K-closest rule: from each connected component of the current roadmap, select the K closest locations p to q.

Multiple-query and Single-query Scenarios

Roadmap-based methods are structured in general as a two-phase computation process:

(i) a preprocessing phase – given the free configuration space, compute the roadmap, followed by

(ii) a query phase – given start and goal locations, connect them to the roadmap and search the resulting graph.

Motion Planning, Multiple-query Solvers

multiple-query: problems, in which the same roadmap can be utilized multiple times to solve multiple motion planning problems in the same workspace.

Motion Planning , Single-query Solvers

single-query: (we do not need to compute a reusable roadmap.)

In this case we can compute a special roadmap to solve the specific query:

(i) computed directly as a function of the start location,

(ii) is just a tree, as cycles do not add new paths from the start location.

Note: For symmetry reasons that we need not worry about here, one often computes two trees, one originating from the start location and one originating from the goal location.

Potentially can use much less samples than multi-query PRMs

Incremental Tree-roadmap Computation

incremental tree-roadmap computation method

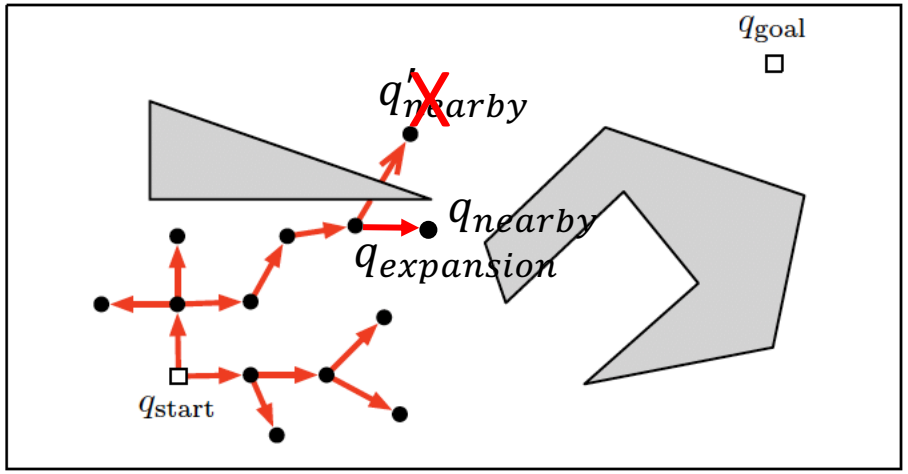
Input: start location q_{start}, number of sample points in tree roadmap $N \in \mathbb{N}$. Also requires access to a sampling algorithm, collision detection algorithm, and a distance notion on Q

Output: a tree roadmap (V, E) for the free configuration space Q_{free} containing q_{start}

1: initialize (V, E) to contain the start location q_{start} and no edges
2: **while** number of nodes in V is less than N :
3: select a node $q_{\text{expansion}}$ from V for expansion
4: choose a collision-free configuration q_{nearby} near $q_{\text{expansion}}$
5: **if** can find a collision-free path from $q_{\text{expansion}}$ to q_{nearby} :
6: $V := V \cup q_{\text{nearby}}$
7: $E := E \cup \{\text{collision-free path from } q_{\text{expansion}} \text{ to } q_{\text{nearby}}\}$
8: **return** (V, E)

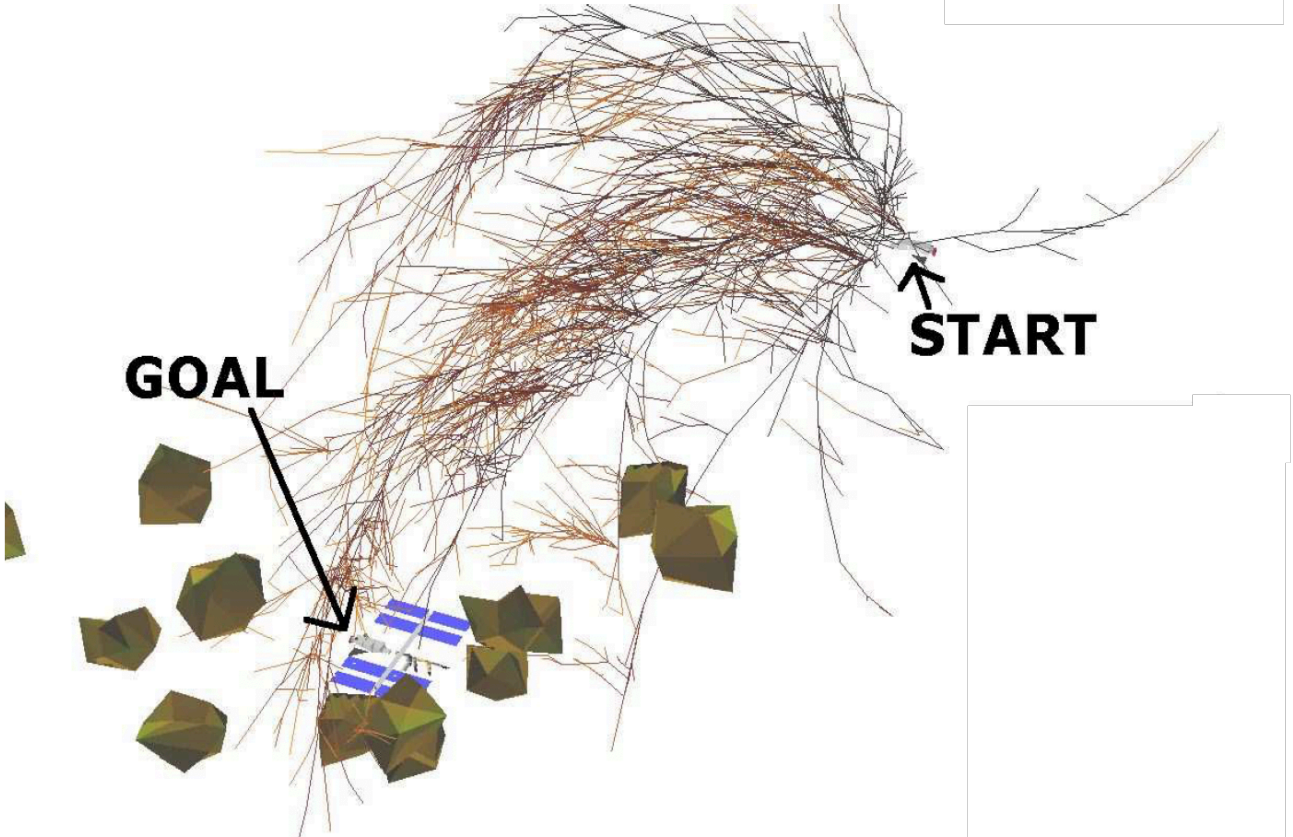
node selection

node expansion



Incremental tree-roadmap Computation: Expansive-Spaces Trees

Expansive-Spaces Trees (ESTs): an efficient single-query planner that covers the space between q_{start} and q_{goal}



Courtesy of [10.1109/ROBOT.2004.1308890](https://doi.org/10.1109/ROBOT.2004.1308890)

'Guided' EST

Incremental tree-roadmap Computation: Expansive-Spaces Trees

Build EST algorithm

Input:

q_0: the configuration where the tree is rooted
N: number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted q_0 at and has $\leq N$ configurations

1: $V \leftarrow \{q_0\}$
2: $\mathrm{E} \leftarrow \{\ \}$
3: **for** $i = 1$ to N **do**
4: $q \leftarrow$ a randomly chosen configuration from T
5: extend $EST(T, q)$
6: **end for**
7: **Return** T

Extend EST algorithm

Input:

$T = (V, E)$: an EST
q: a configuration from which to grow T

Output:

A new configuration q_{new} in the neighborhood of q, or NIL in case of failure

1: $q_{new} \leftarrow$ a random collision-free configuration from the neighborhood of q
2: **if** $\mathrm{q_{new}}$ is visible from q **then**
3: $V \leftarrow \{q_{new}\}$
4: $\mathrm{E} \leftarrow E \cup \{q, q_{new}\}$
5: return q_{new}
6: **else**
7: **Return** NIL
8: end if

Incremental tree-roadmap Computation: Rapidly-Exploring Random Tree

One of the most popular PRM variants is the Rapidly-Exploring Random Tree (RRT) algorithm, which grows a tree rather than a graph.

incremental tree-roadmap computation method

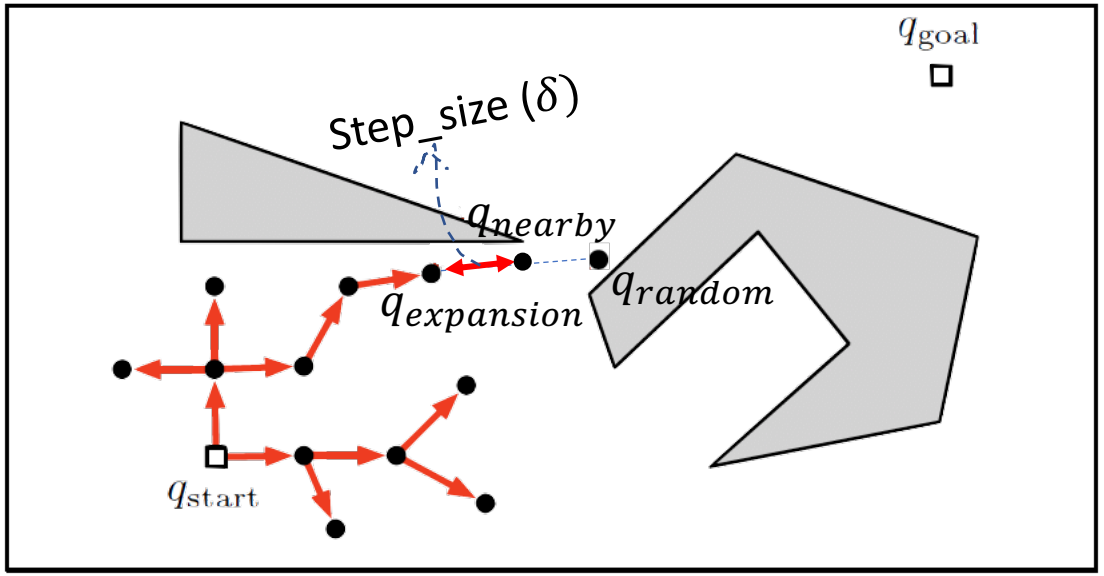
Input: start location q_{start}, number of sample points in tree roadmap $N \in \mathbb{N}$. Also requires access to a sampling algorithm, collision detection algorithm, and a distance notion on Q

Output: a tree roadmap (V, E) for the free configuration space Q_{free} containing q_{start}

1: initialize (V, E) to contain the start location q_{start} and no edges
2: **while** number of nodes in V is less than N :
3: select a node $q_{\text{expansion}}$ from V for expansion
4: choose a collision-free configuration q_{nearby} near $q_{\text{expansion}}$
5: **if** can find a collision-free path from $q_{\text{expansion}}$ to q_{nearby} :
6: $V := V \cup q_{\text{nearby}}$
7: $E := E \cup \{\text{collision-free path from } q_{\text{expansion}} \text{ to } q_{\text{nearby}}\}$
8: **return** (V, E)

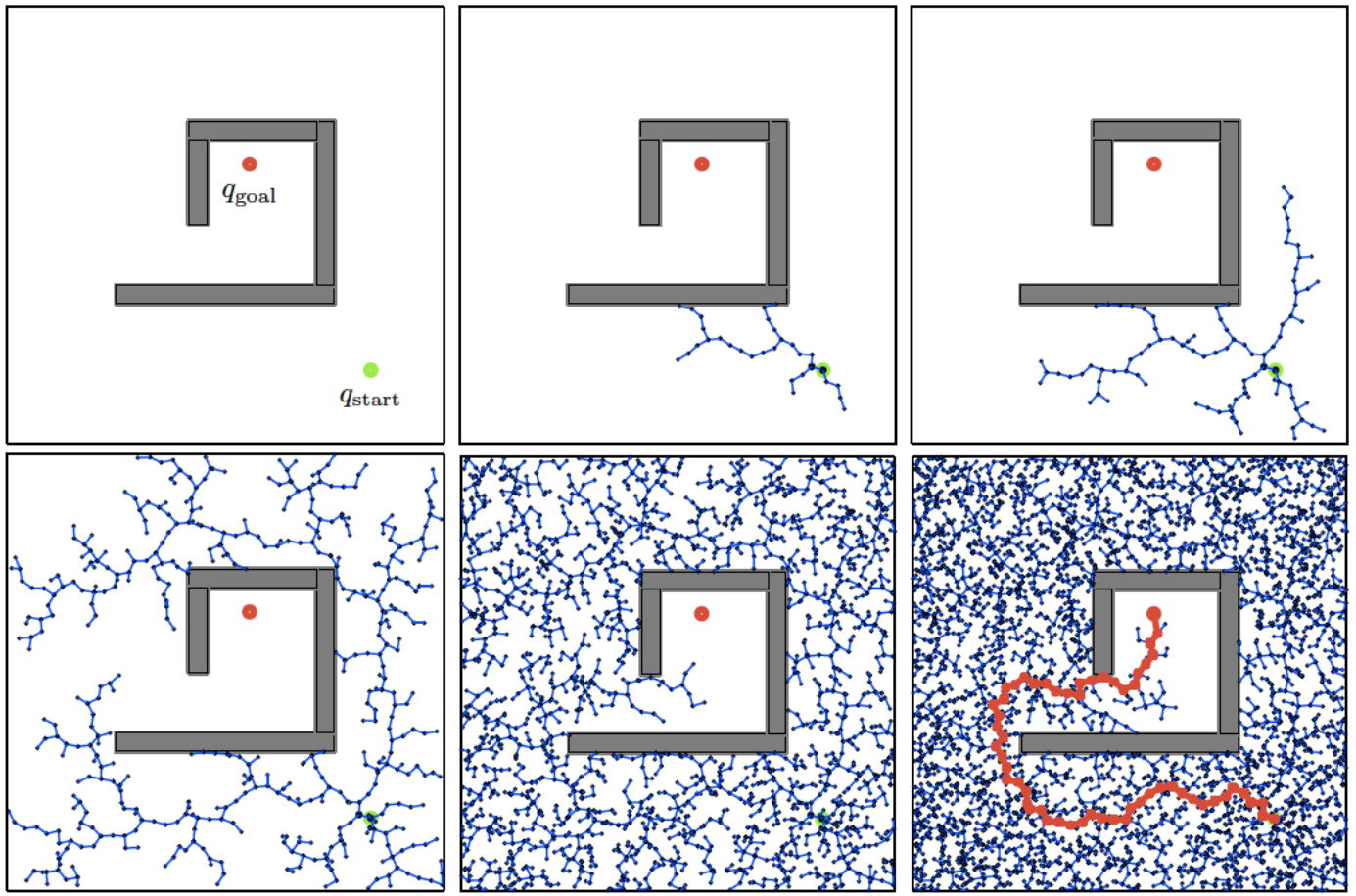
choose a random configuration q_{random} in Q and select for expansion the node $q_{expansion}$ from V that is closest to q_{random}

select a collision-free configuration q_{nearby} near $q_{expansion}$ by moving from $q_{expansion}$ towards q_{random}



RRT in Environment with Obstacles: an Example

One of the most popular PRM variants is the Rapidly-Exploring Random Tree (RRT) algorithm, which grows a tree rather than a graph.

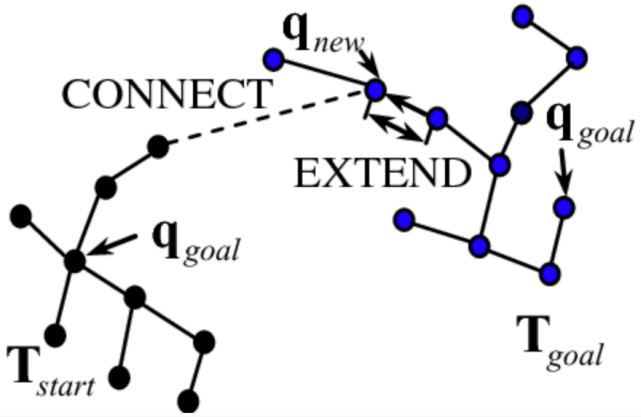


RRT-connect

Algorithm RRT-Connect

1. $T_s \leftarrow \{s\}$.

- $T_g \leftarrow \{g\}$.
- **for** $i = 1, \ldots, N$ **do**
- $q_{rand} \leftarrow Sample()$
- $q_e \leftarrow$ Extend-Tree(T_s, q_{rand}, δ) (extend start tree at most δ distance toward q_{rand})
- $q'_e \leftarrow$ Extend-Tree(T_g, q_{rand}, δ) (extend goal tree at most δ distance toward q_{rand})
- **if** $d(q_e, q'_e) \leq \delta$ and Visible(q_e, q'_e) **then** (trees are close enough)
- Add edge $q_e \rightarrow q'_e$ to connect T_s and T_g
- **return** the path from s to g
- **return** "no path"



Algorithm Extend-Tree(T, q_{rand}, δ)

1. $q_{near} \leftarrow Nearest(T, q_{rand})$
2. $q \leftarrow q_{near} + \min 1, \frac{\delta}{d(q_{rand}, q_{near})}(q_{rand} - q_{near})$
3. **if** Visible(q_{near}, q) **then**
4. Add edge $q_{near} \rightarrow q$ to T.
5. **return** q.
6. **return** q_{near}.

References:

[1] F. Bullo and S. L. Smith. Lecture notes on robotic planning and kinematics