Algorithmic Motion Planning (236610)
Lecture 4—Sampling-based planners (1)

Oren Salzman

Computer Science Department, Technion

Today's lecture

Sampling-based motion planners—probabilistically-complete algorithms for motion planning

- Multi-query motion planning
- Primitive operations in sampling-based motion planning

For additional material see Ch. 7 of Principles of Robot Motion* book.

* H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun: *Principles of Robot Motion: Theory, Algorithms, and Implementation*, MIT press

Animation by Javed Hossain, adapted from https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

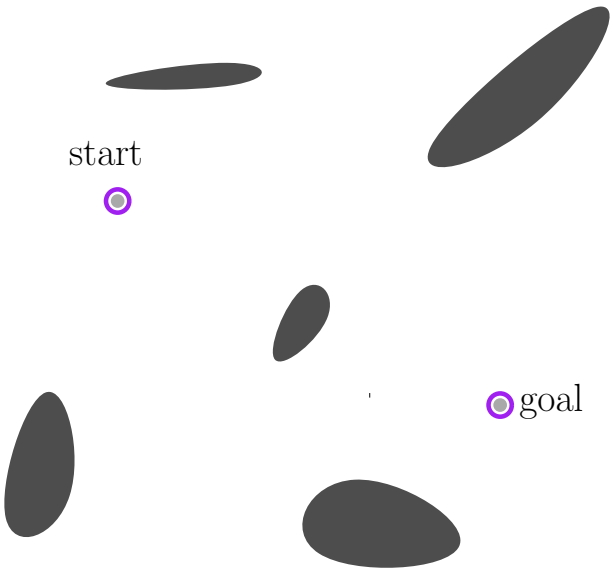
Motivation

- Exact methods rely on an explicit representation of $\mathcal{X}_{\text{free}}$
- Their running time is exponential in the complexity of the obstacles
- Paradigm shift—sampling-based motion planning
  - Approximate the structure of the continuous $\mathcal{X}_{\text{free}}$ using a dsicrete graph called a roadmap $\mathcal{G}$
  - The roadmap $\mathcal{G}$ is constructed by randomly sampling configurations
  - Applicable to single-query and multi-query problems

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

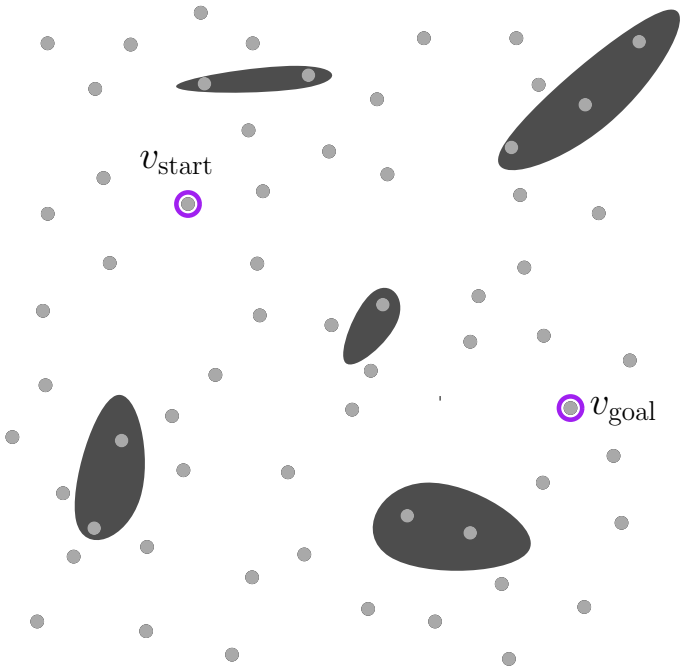


- sampler
- collision detector
- nearest neighbors
- local planner
- path planner

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

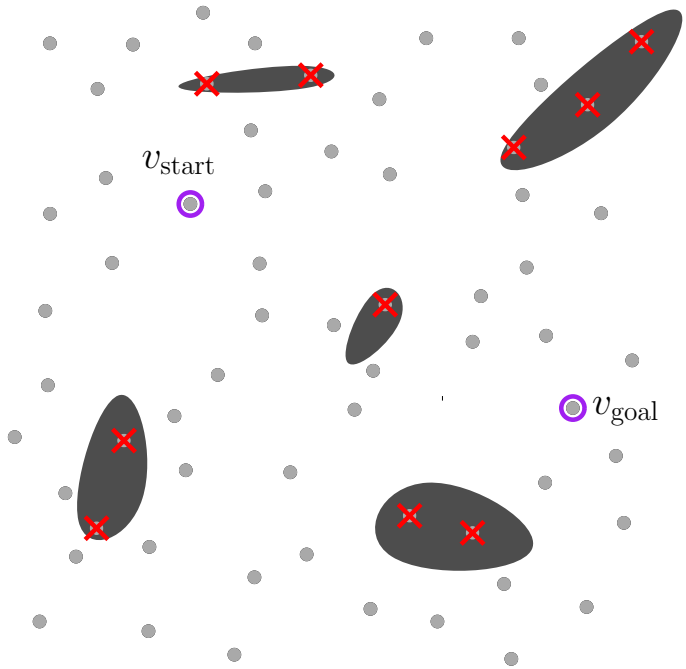


- sampler
- collision detector
- nearest neighbors
- local planner
- path planner

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

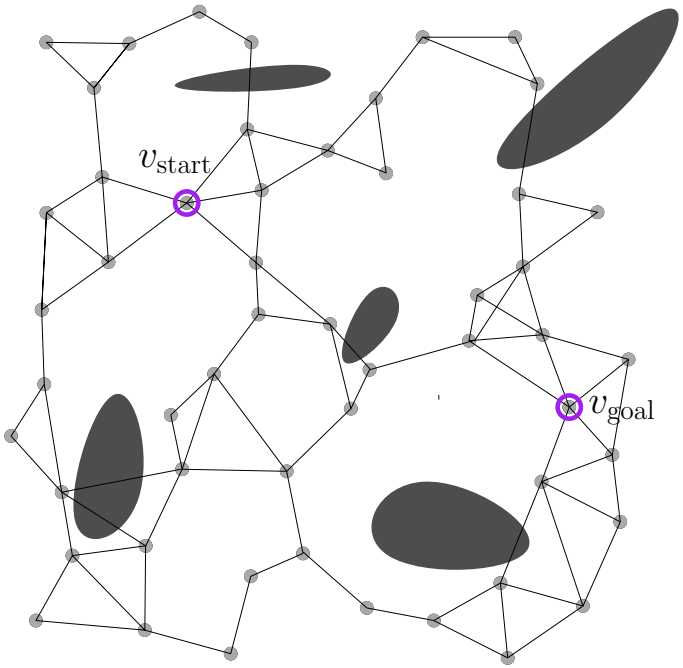


- sampler
- collision detector
- nearest neighbors
- local planner
- path planner

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

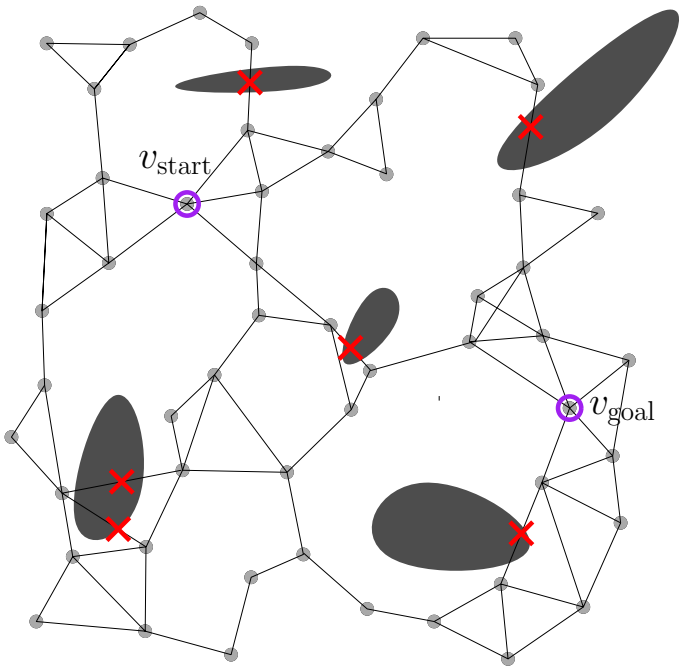


- sampler
- collision detector
- nearest neighbors
- local planner
- path planner

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

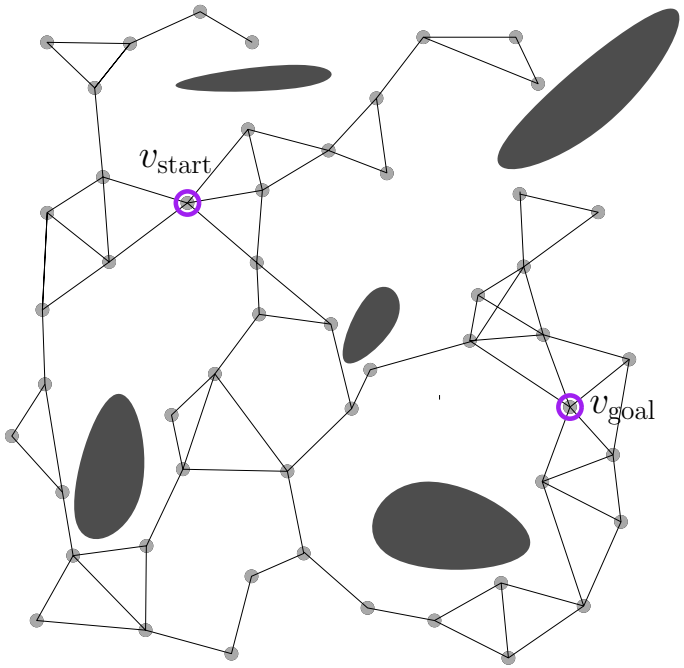


- sampler
- collision detector
- nearest neighbors
- local planner
- path planner

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations

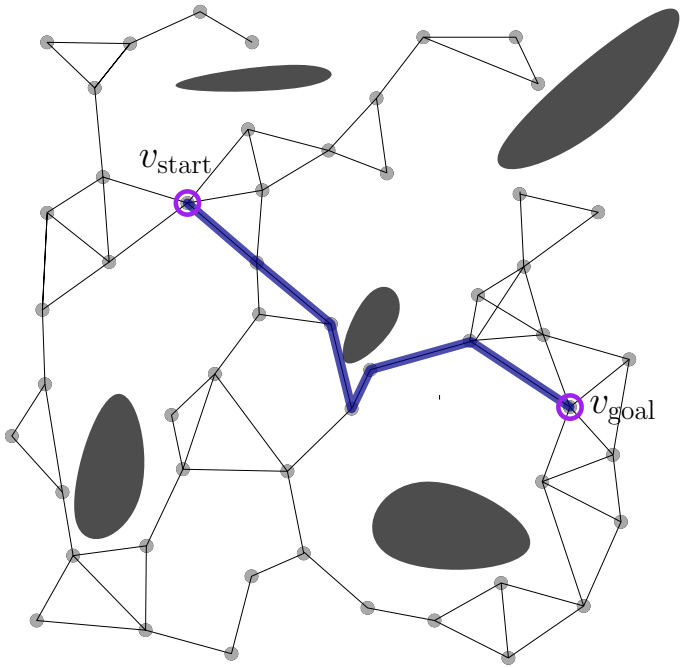


- `sampler`
- `collision detector`
- `nearest neighbors`
- `local planner`
- `path planner`

Solving the motion-planning problem via roadmaps

Probabilistic Roadmap Method (PRM) [Kavraki et al.96, . . .]

- Approximate the structure of $\mathcal{X}$ via a roadmap or a graph $\mathcal{G}$
- Vertices of $\mathcal{G}$ are configurations
- Edges of $\mathcal{G}$ are local paths connecting configurations



- `sampler`
- `collision detector`
- `nearest neighbors`
- `local planner`
- `path planner`

PRM—algorithmic description

Preprocessing phase

Input : The C-space $\mathcal{X}$;
no. of iterations n

Output: Roadmap $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

1: $\mathcal{V} = \emptyset$; $\mathcal{E} = \emptyset$; $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E})$;
2: $\mathcal{V} \leftarrow$ `sample_free`$(\mathcal{X}, n)$
3: **for all** $x, y \in \mathcal{V}$ s.t. x is close to y **do**
4: **if** `collision_free`$(\overline{x, y})$ **then**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, y)\}$
6: **return** $\mathcal{G}$

Query phase

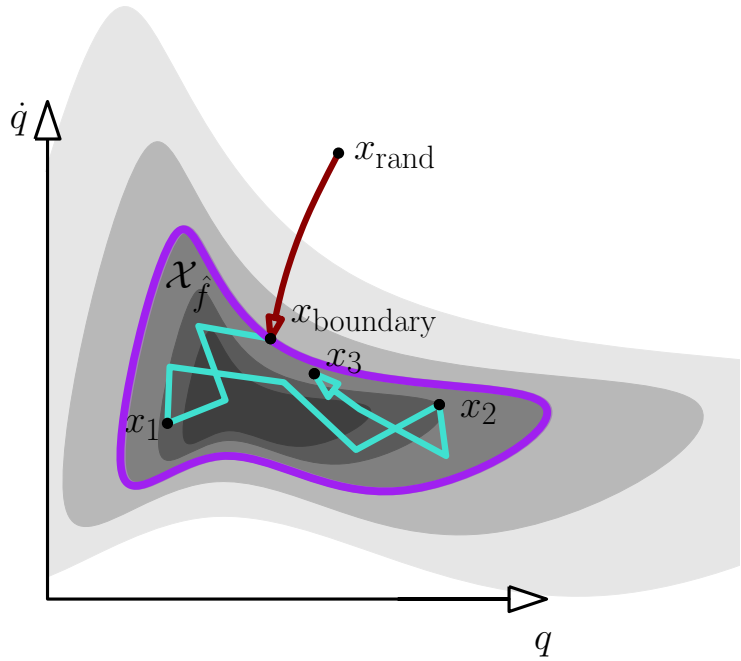
Input : Roadmap $\mathcal{G}$;
query x_{start}, x_{goal}

Output: Collision-free path γ connecting x_{start} to x_{goal}

1: **for all** $x \in \mathcal{V}$ s.t. x is close to x_{start} **do**
2: **if** `collision_free`$(\overline{x, x_{\text{start}}})$ **then**
3: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, x_{\text{start}})\}$
4: **for all** $x \in \mathcal{V}$ s.t. x is close to x_{goal} **do**
5: **if** `collision_free`$(\overline{x, x_{\text{goal}}})$ **then**
6: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, x_{\text{goal}})\}$
7: **return** `shortest_path`$(\mathcal{G}, x_{\text{start}}, x_{\text{goal}})$

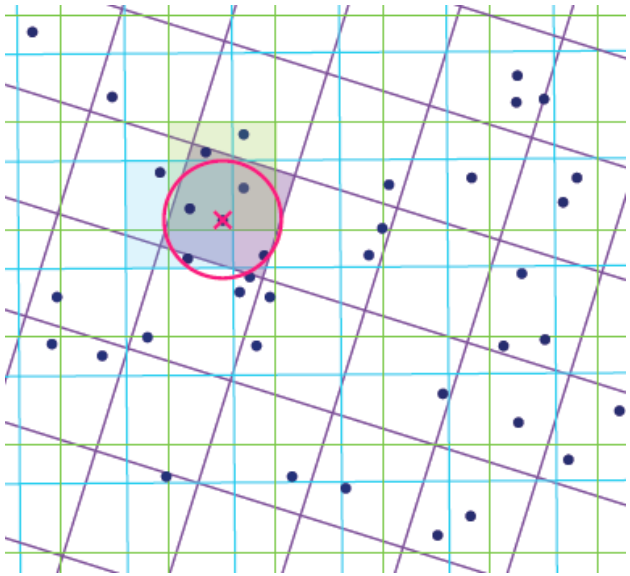
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



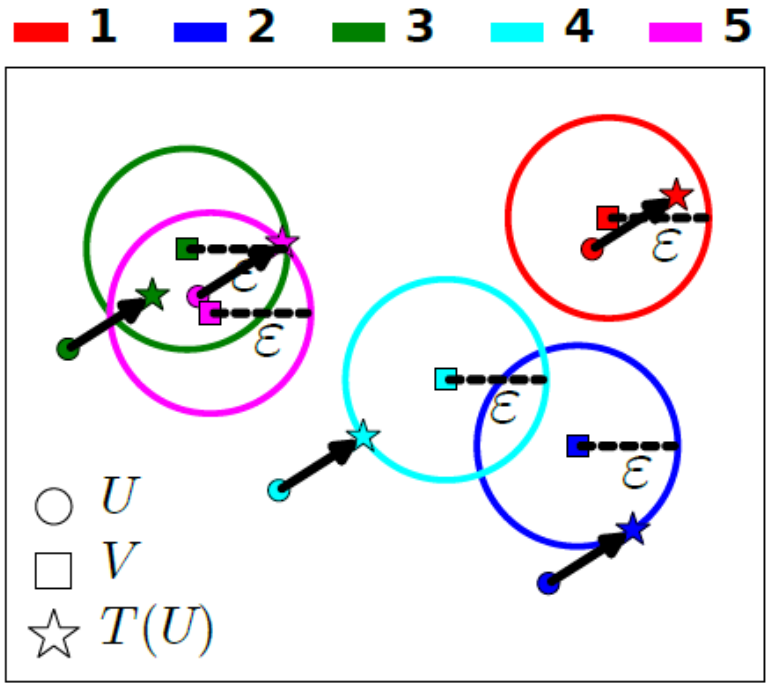
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



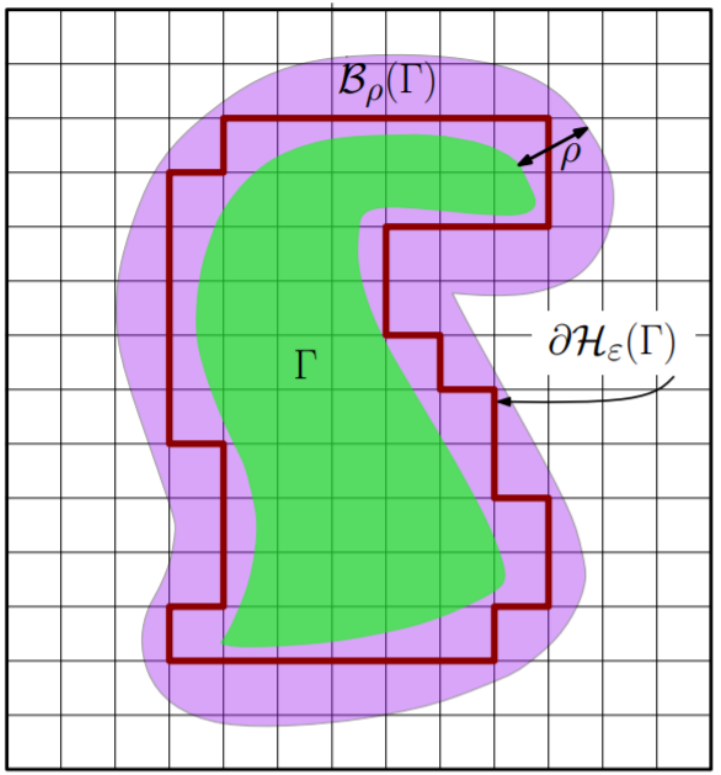
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



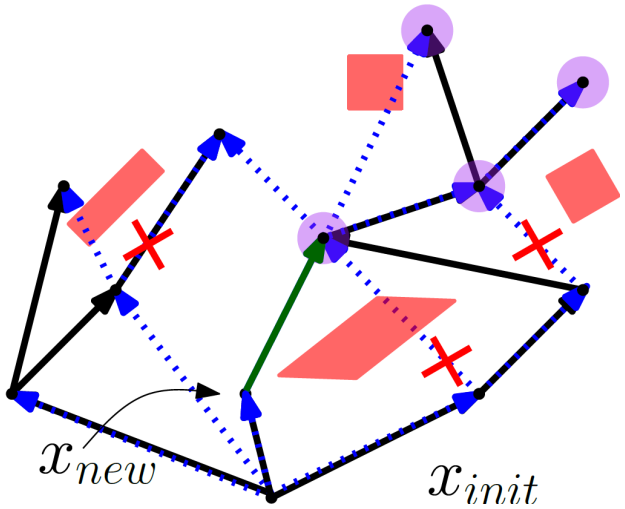
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



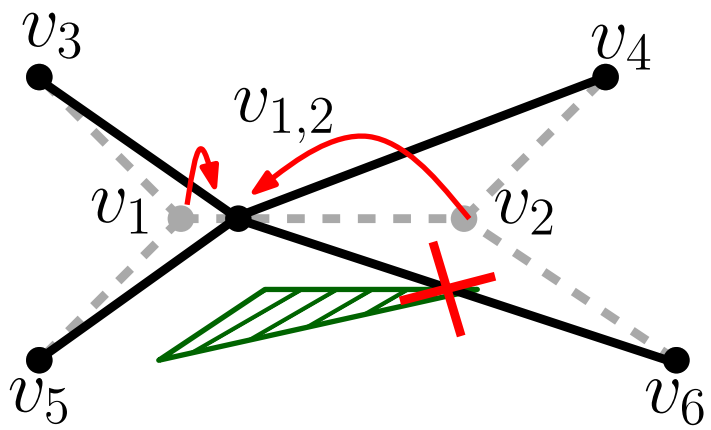
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



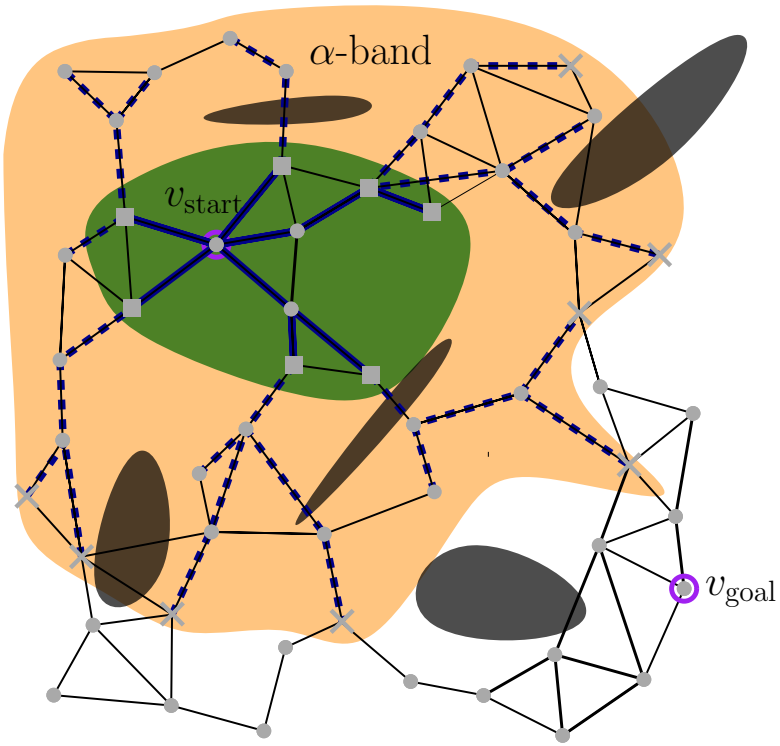
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



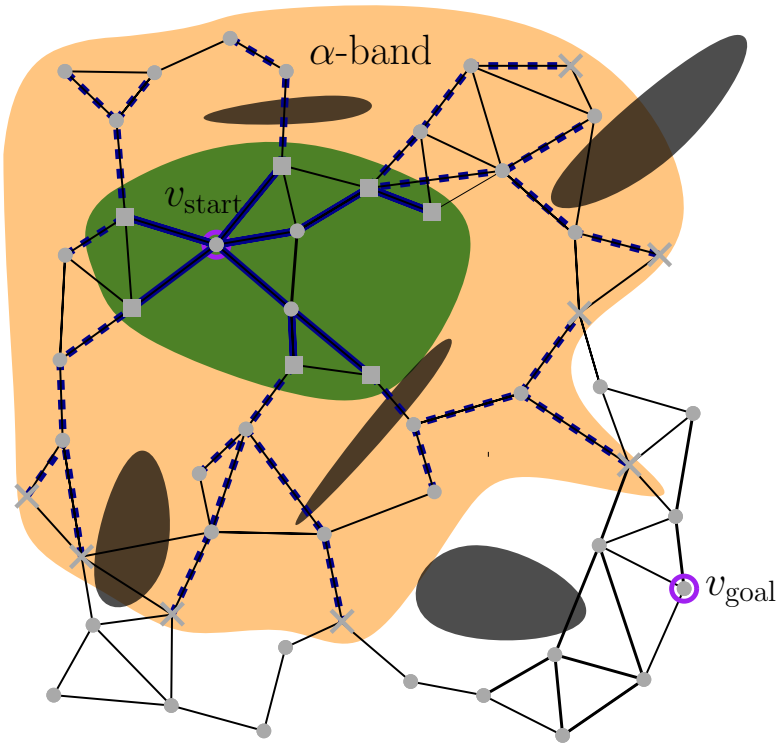
General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- . . .



General computational challenges in motion planning

- How to efficiently and effectively sample $\mathcal{X}$?
- How to efficiently compute nearest-neighbors in $\mathcal{X}$?
- What metric should be used to evaluate distances in $\mathcal{X}$?
- How to ensure convergence to high-quality paths?
- How to efficiently compute high-quality paths?
- How to compress a given roadmap $\mathcal{G}$?
- How to efficiently and effectively search $\mathcal{G}$?
- ...



Sampling configurations

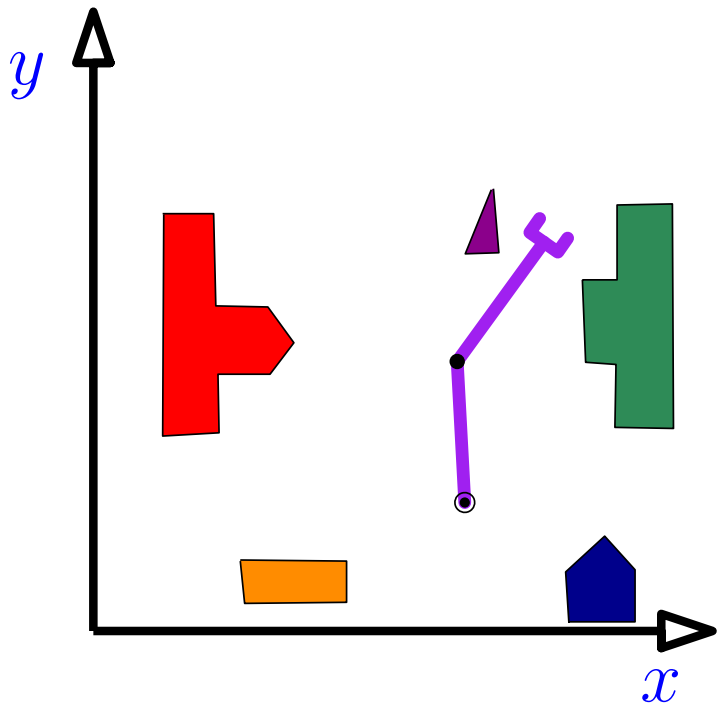
- Sampling can have a dramatic effect on the success rate of answering queries
- Sampling needs to be fast
- Sampling needs to be effective (what does this mean?)

1: $\mathcal{V} = \emptyset$; $\mathcal{E} = \emptyset$; $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E})$;
2: $\mathcal{V} \leftarrow$ `sample_free`$(\mathcal{X}, n)$
3: **for all** $x, y \in \mathcal{V}$ s.t. x is close to y **do**
4: **if** `collision_free`$(\overline{x, y})$ **then**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, y)\}$
6: **return** $\mathcal{G}$

Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

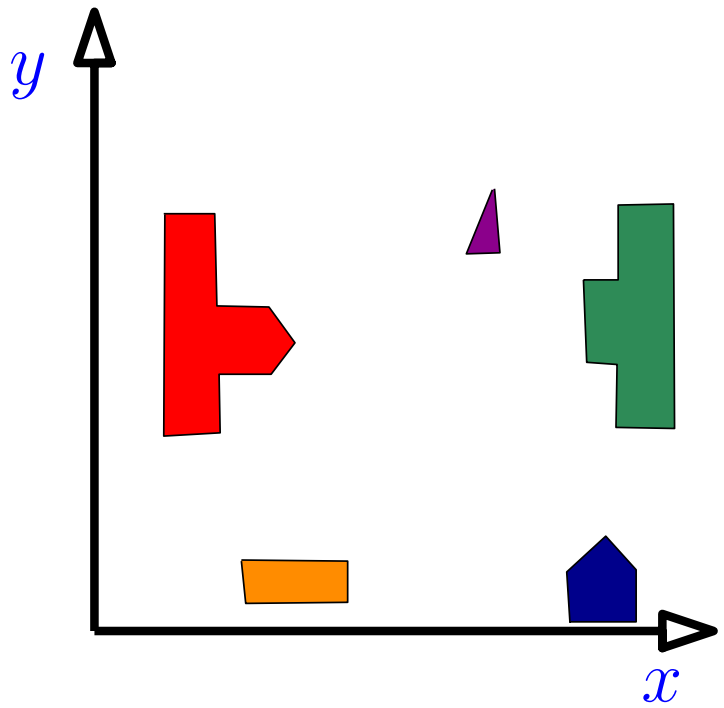
Example (1)—$\mathcal{R}$ is a planar 2R manipulator ($\mathcal{X} = \text{SO(2)}$)

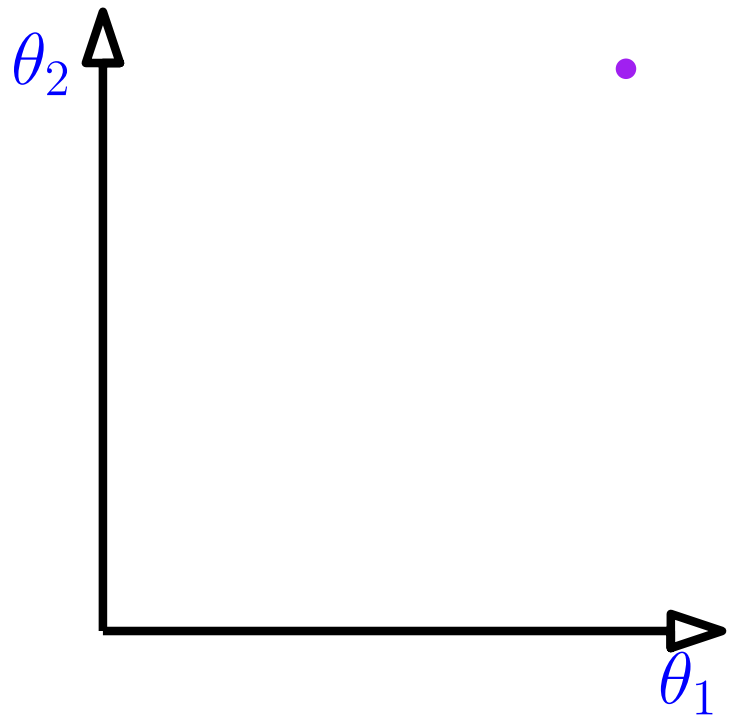


Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

Example (1)—$\mathcal{R}$ is a planar 2R manipulator ($\mathcal{X} = \text{SO(2)}$)

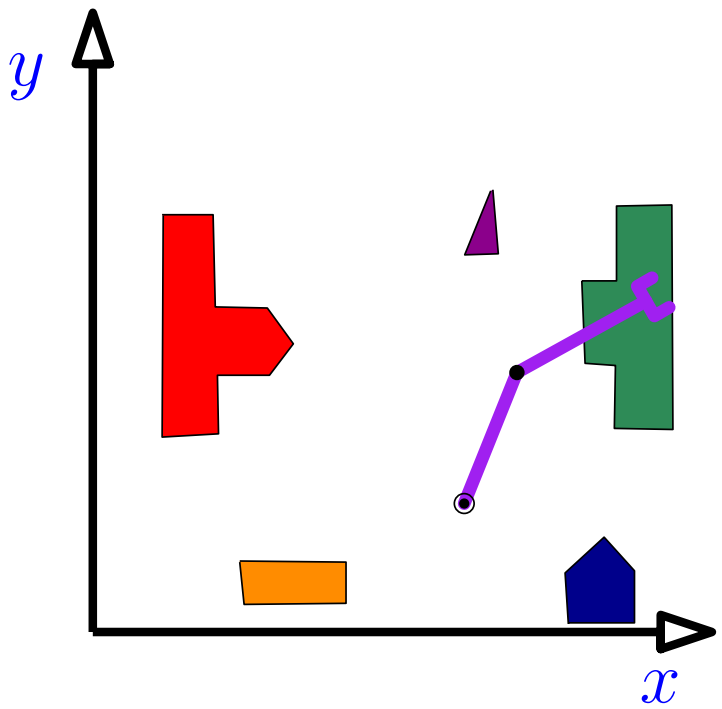


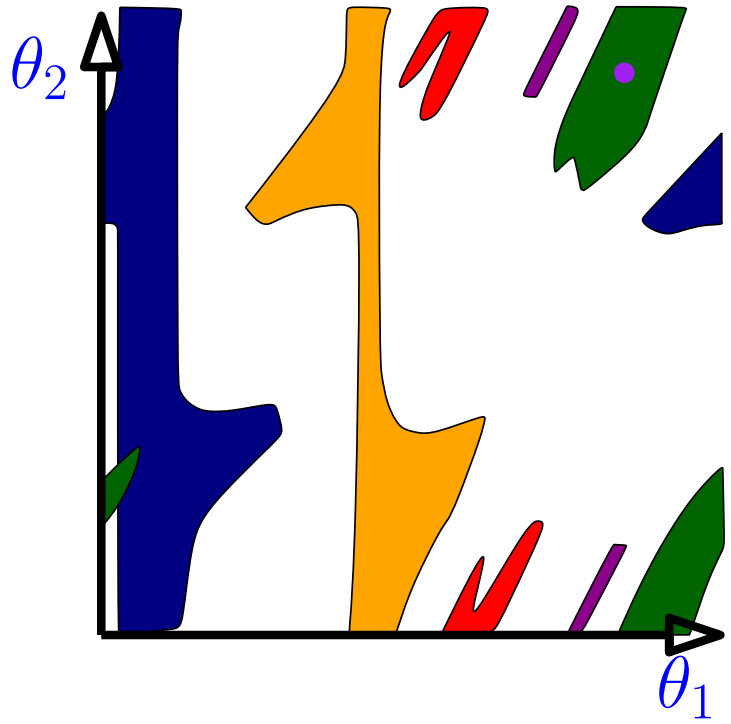


Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

Example (1)—$\mathcal{R}$ is a planar 2R manipulator ($\mathcal{X} = \text{SO(2)}$)

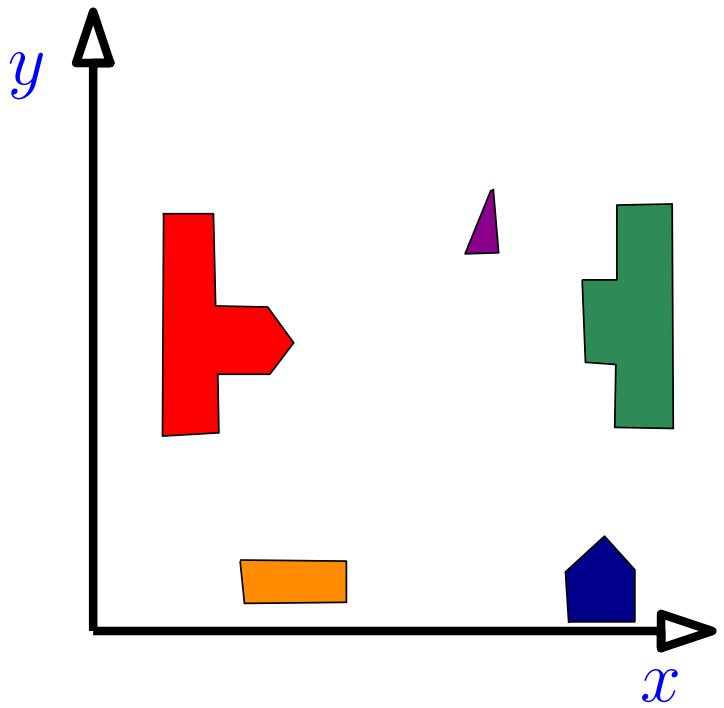


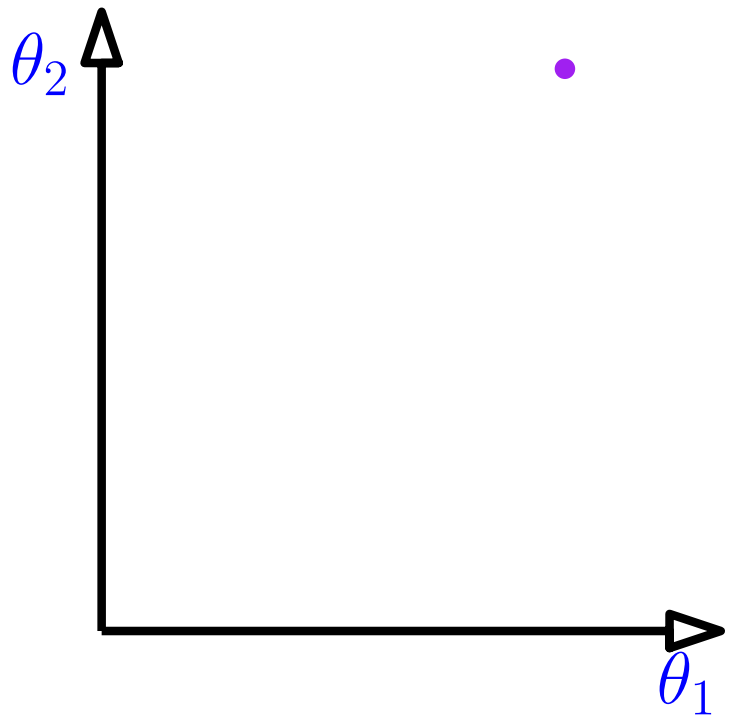


Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

Example (1)—$\mathcal{R}$ is a planar 2R manipulator ($\mathcal{X} = \text{SO(2)}$)

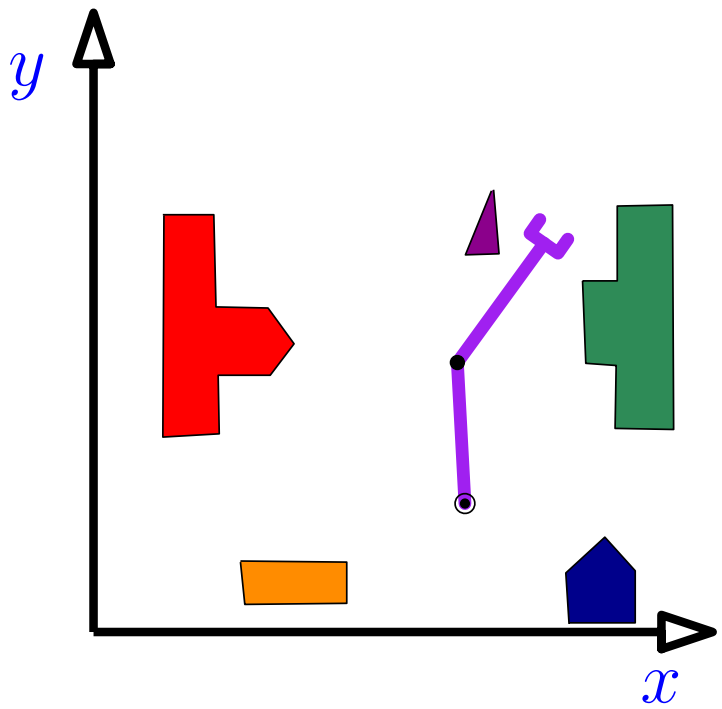


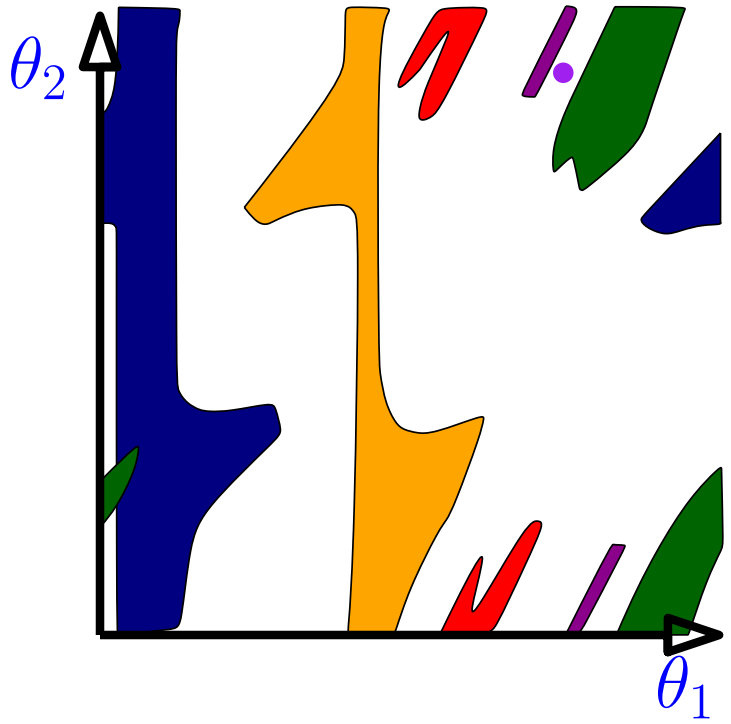


Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

Example (1)—$\mathcal{R}$ is a planar 2R manipulator ($\mathcal{X} = \text{SO(2)}$)





Sampling configurations—uniform sampling

- Easiest and most common: uniform random rejection sampling
  - Sample a configuration $q_{\text{rand}} \in \mathcal{X}$ uniformly at random
  - Compute robot's workspace location when in q_{rand} and check for collision
  - Repeat until $q_{\text{rand}} \in \mathcal{X}_{\text{free}}$

Example (2)—$\mathcal{R}$ is a rigid body moving in space ($\mathcal{X} = \text{SE(3)} = \mathbb{R}^3 \times \text{SO(3)}$)

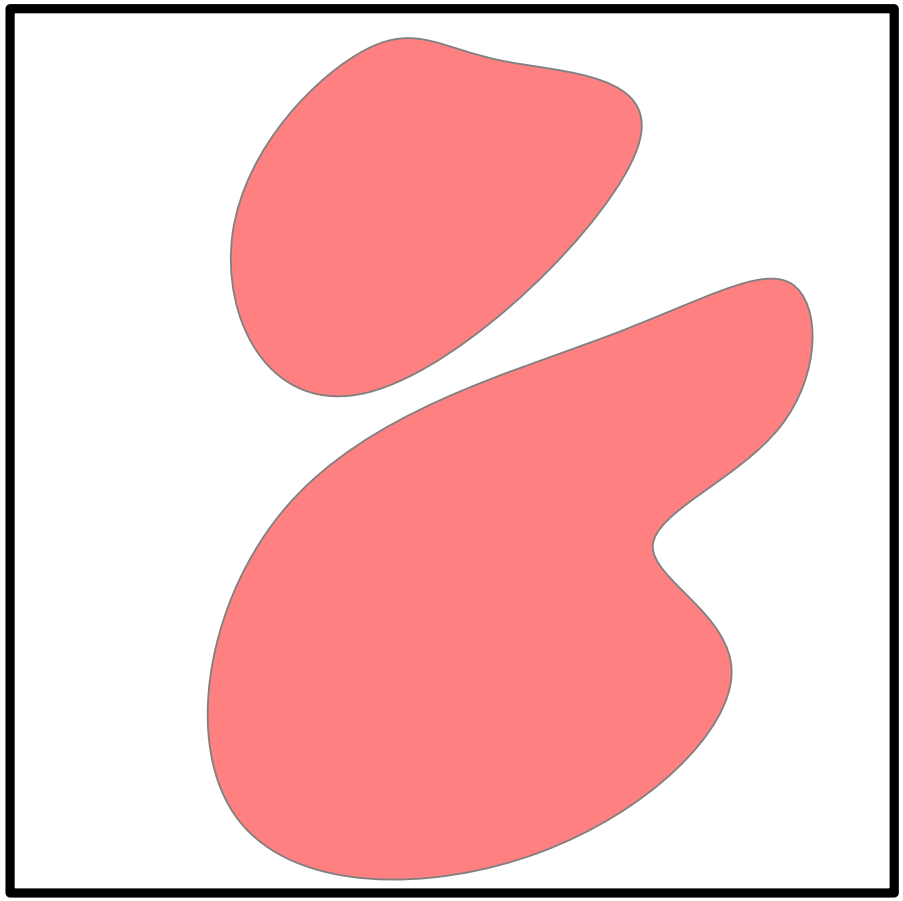
- It is easy to sample uniformly in $\mathbb{R}^3$
- How to sample uniformly in SO(3)?
  - Depends on representation (Euler angles, 3D matrices, quaternions)
  - Roughly speaking, this corresponds to randomly sampling from $\mathbb{S}^3$

SO(3) is the 3D rotation group—the group of all rotations about the origin of three-dimensional Euclidean space $\mathbb{R}^3$

$\mathbb{S}^n$ is the n-dimensional sphere in $\mathbb{R}^{n+1}$

Sampling configurations—uniform sampling

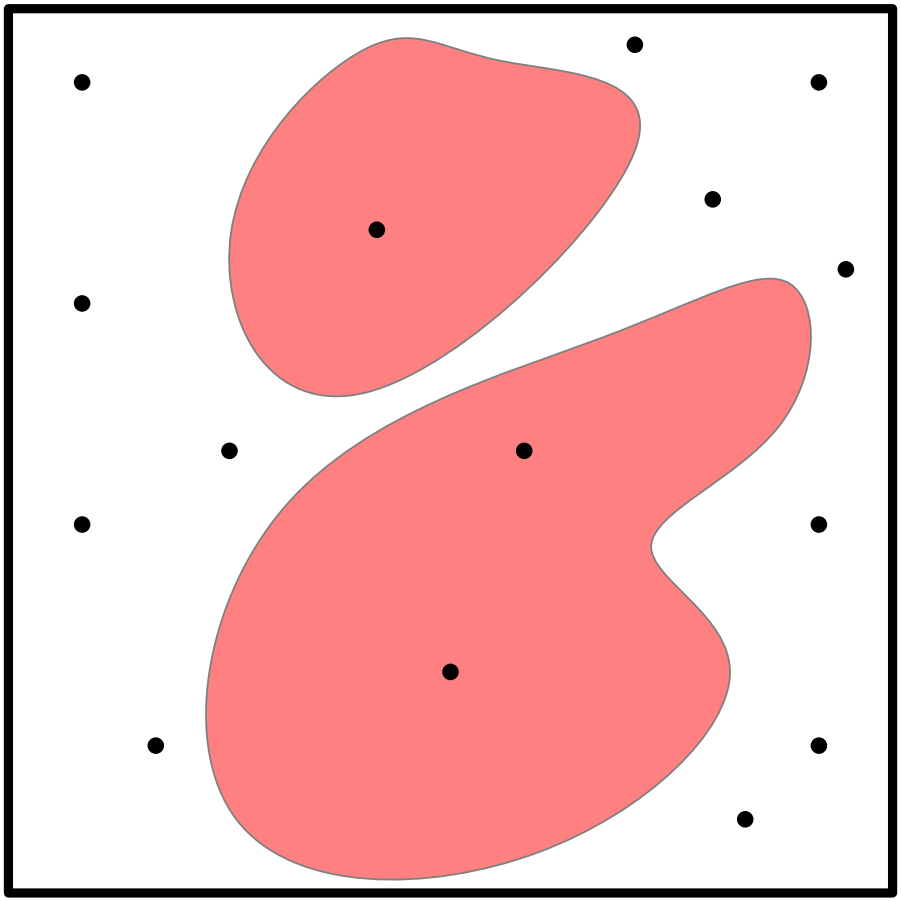
- Is uniform good? Multiple samples in open regions are not as useful as samples in narrow passages



- Numerous alternative sampling approaches

Sampling configurations—uniform sampling

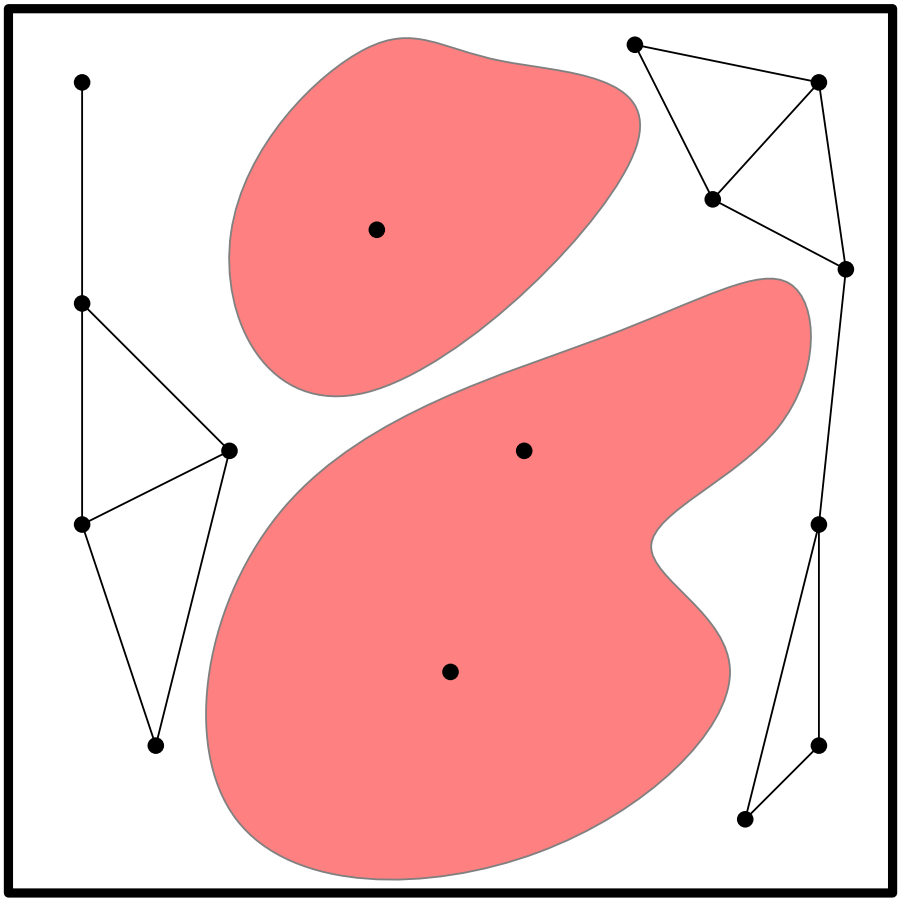
- Is uniform good? Multiple samples in open regions are not as useful as samples in narrow passages



- Numerous alternative sampling approaches

Sampling configurations—uniform sampling

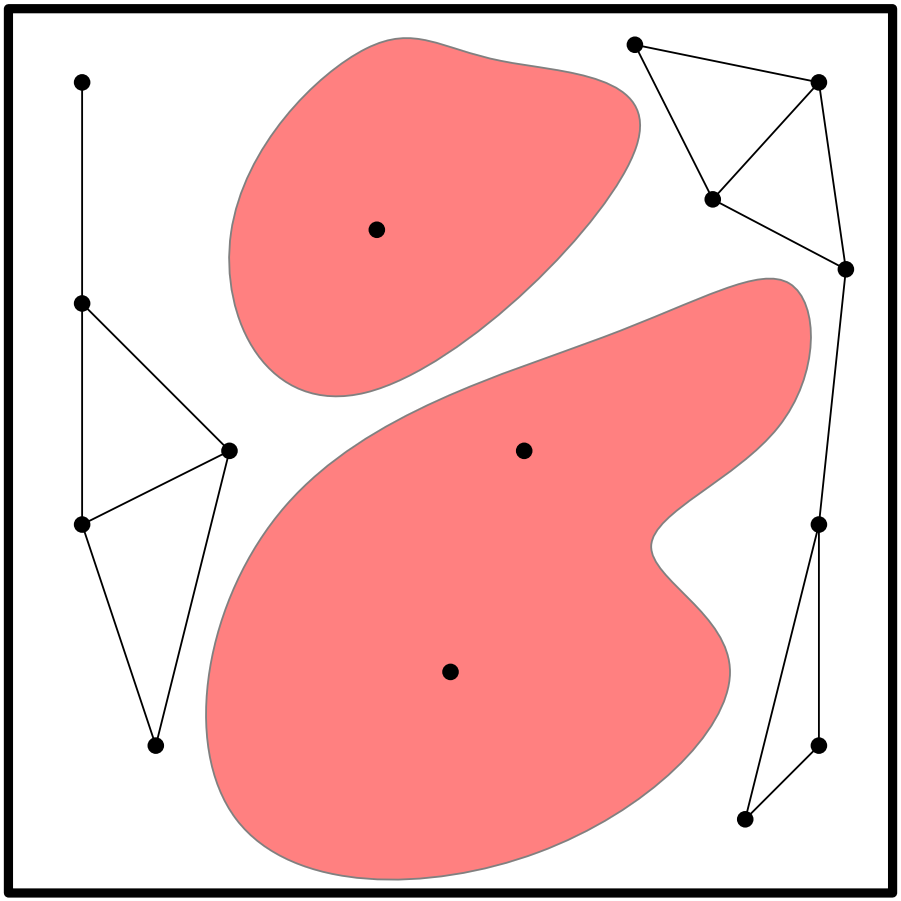
- Is uniform good? Multiple samples in open regions are not as useful as samples in narrow passages



- Numerous alternative sampling approaches

Sampling configurations—uniform sampling

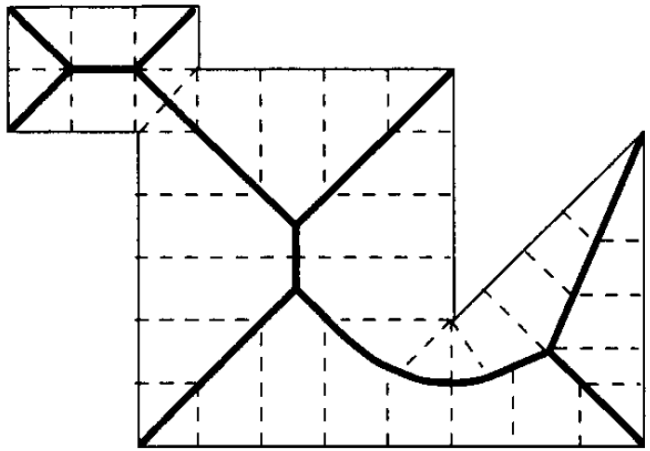
- Is uniform good? Multiple samples in open regions are not as useful as samples in narrow passages



- Numerous alternative sampling approaches

Sampling configurations—alternative approaches

- Intuition (1) - sample as far away from the obstacles
- Medial axis sampling [Wilmarth, Amato, Stiller99; Holleman, Kavraki 00; Lien, Thomas, Amato03]
  - A maximal ball is a ball $B(x, r) \subset \mathcal{X}$ such that no other ball can be a proper subset
  - The centers of all maximal balls trace out a one-dimensional set of points referred to as the medial axis
  - Easy to compute in $\mathcal{W}$, much harder in $\mathcal{X}$

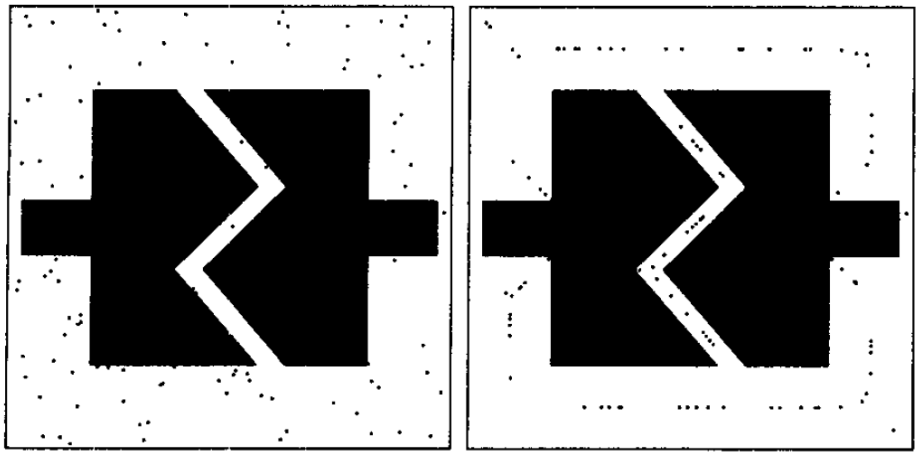


Medial axis of a polygon

Figures adapted from [Wilmarth, Amato, Stiller99]

Sampling configurations—alternative approaches

- Intuition (1) - sample as far away from the obstacles
- Medial axis sampling [Wilmarth, Amato, Stiller99; Holleman, Kavraki 00; Lien, Thomas, Amato03]
  - A maximal ball is a ball $B(x, r) \subset \mathcal{X}$ such that no other ball can be a proper subset
  - The centers of all maximal balls trace out a one-dimensional set of points referred to as the medial axis
  - Easy to compute in $\mathcal{W}$, much harder in $\mathcal{X}$

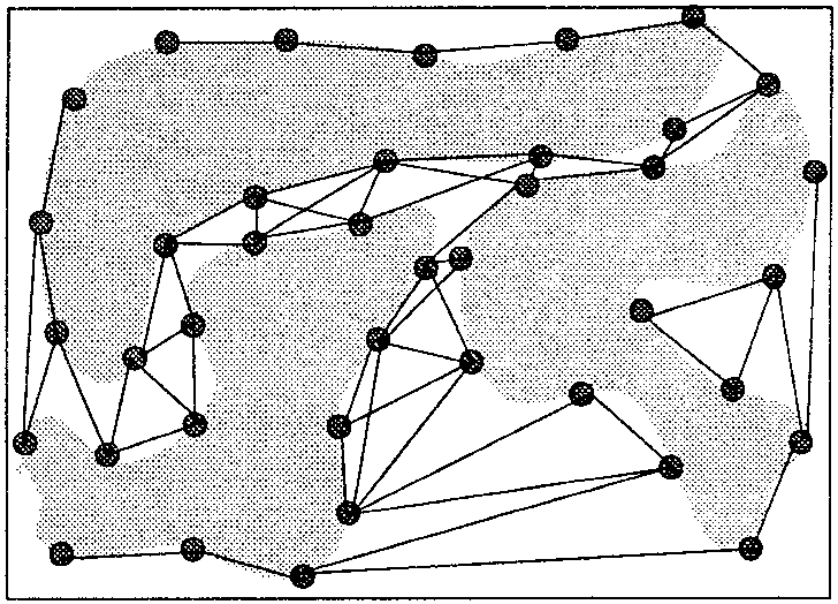


uniform sampling vs. medial-axis sampling

Figures adapted from [Wilmarth, Amato, Stiller99]

Sampling configurations—alternative approaches

- Intuition (2) - sample near obstacle boundary
- Obstacle-based sampling [Amato, Wu96; Yeh, Thomas, Eppstein Amato12]

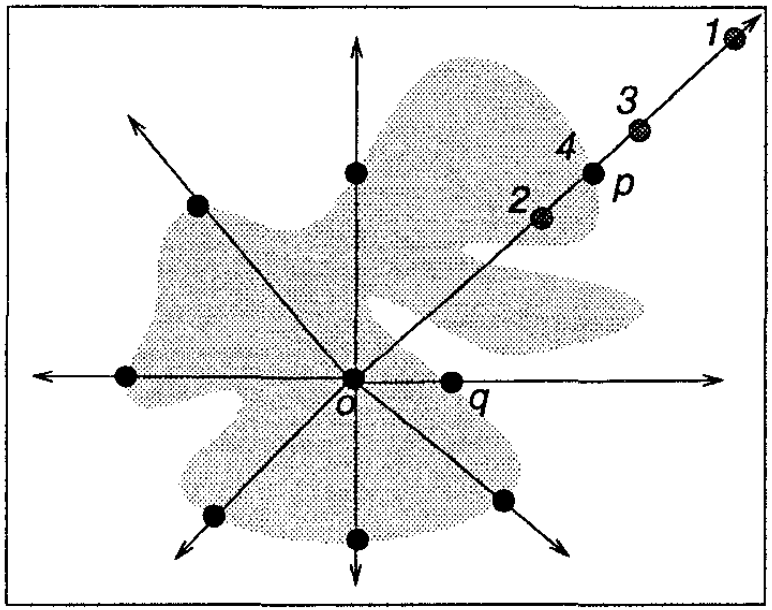


Roadmap

Figure adapted from [Amato, Wu96Wilmarth, Amato, Stiller99]

Sampling configurations—alternative approaches

- Intuition (2) - sample near obstacle boundary
- Obstacle-based sampling [Amato, Wu96; Yeh, Thomas, Eppstein Amato12]



Generating samples in $\mathcal{X}$

Figure adapted from [Amato, Wu96Wilmarth, Amato, Stiller99]

Sampling configurations—alternative approaches

Many works also considered deterministic sampling sequences

- Grids
- Halton sequences
  - Appear to be random for many purposes
  - Generalization of van der Corput sequences for arbitrary dimensions
  - Constructed by reversing the base-n representation of the sequence of natural numbers

More on this in Ch. 5 of LaValle's book

i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	

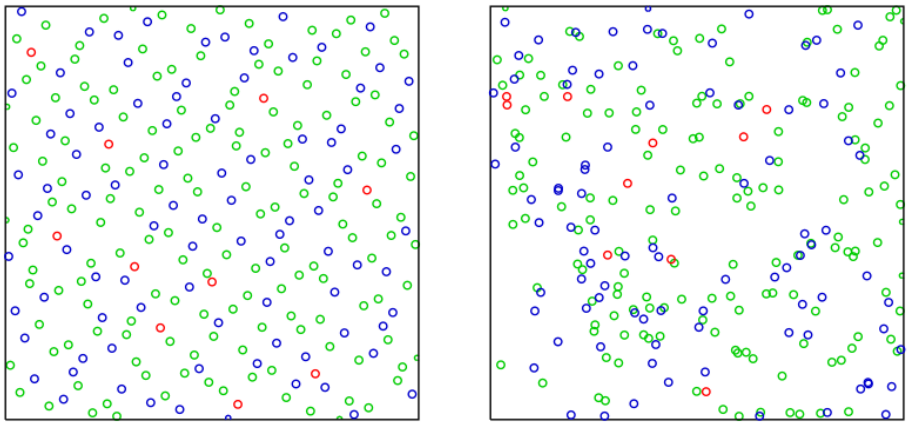
Figure 5.2: The van der Corput sequence is obtained by reversing the bits in the binary decimal representation of the naive sequence.

Sampling configurations—alternative approaches

Many works also considered deterministic sampling sequences

- Grids
- Halton sequences
  - Appear to be random for many purposes
  - Generalization of van der Corput sequences for arbitrary dimensions
  - Constructed by reversing the base-n representation of the sequence of natural numbers

More on this in Ch. 5 of LaValle's book



256 points from Halton sequence (left) and a pseudonumber random generator (right).
Figure adapted from https://en.wikipedia.org/wiki/Halton_sequence

Connecting configurations

- Two typical approaches
  - Connect every point to all points within distance r
  - Connect every point to k closest points

1: $\mathcal{V} = \emptyset$; $\mathcal{E} = \emptyset$; $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E})$;
2: $\mathcal{V} \leftarrow$ `sample_free`$(\mathcal{X}, n)$
3: **for all** $x, y \in \mathcal{V}$ s.t. x is close to y **do**
4: **if** `collision_free`$(\overline{x, y})$ **then**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, y)\}$
6: **return** $\mathcal{G}$

- How to compute r-NN or k-NN?
- How to choose k or r?
- What is the distance metric we use?

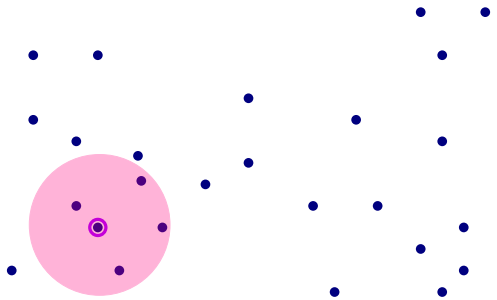
Connecting configurations

- Trivial implementation is using a linear search: $O(n)$ per sample
- Advanced data structures exists
  - They typically preprocess a set of points to efficiently answer NN queries
  - Variants include exact / approximate algorithms that are metric-specific / metric agnostic appropriate for low- / high-dimensional spaces
  - Examples include Voronoi diagram [Descartes 1644 (?)] *kd*-trees [Bently75] LSH [Indyk, Motwani98] RTG [Aiger, Kaplan, Sharir14]
- Their applicability and effectiveness changes with the metric used
- FLANN `https://github.com/mariusmuja/flann` contains a collection of algorithms and a system for automatically choosing the best algorithm and optimum parameters depending on a dataset

Connecting configurations—Randomly Transformed Grids [Aiger, Kaplan, Sharir14]

- Given a set P of n points in $\mathbb{R}^d$ and a radius r, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability

- The algorithm:
  1. Place a d-dimensional axis-parallel grid of cell size c with a random shift
  2. Partition P into the grid cells
  3. Compute distances between all points in the same cell
  4. Report (p, q) if $||p - q||_2 \leq r$
  5. Repeat steps (1)-(4) m times

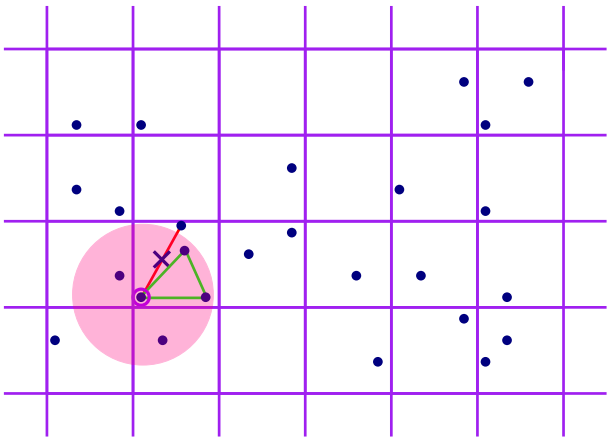


The algorithm is easy to implement (why?)
Running time is $C(d) \cdot (n + k) \log n$ ($C(d)$ is a constant, k is the number of reported points
The analysis requires a bit more work . . .

Connecting configurations—Randomly Transformed Grids [Aiger, Kaplan, Sharir14]

- Given a set P of n points in $\mathbb{R}^d$ and a radius r, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability

- The algorithm:
  1. Place a d-dimensional axis-parallel grid of cell size c with a random shift
  2. Partition P into the grid cells
  3. Compute distances between all points in the same cell
  4. Report (p, q) if $||p - q||_2 \leq r$
  5. Repeat steps (1)-(4) m times



The algorithm is easy to implement (why?)

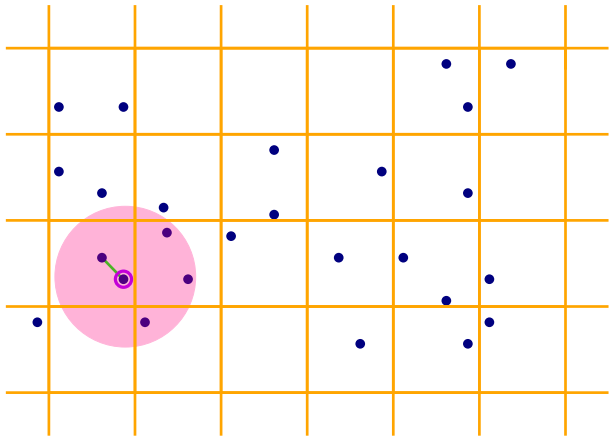
Running time is $C(d) \cdot (n + k) \log n$ ($C(d)$ is a constant, k is the number of reported points

The analysis requires a bit more work . . .

Connecting configurations—Randomly Transformed Grids [Aiger, Kaplan, Sharir14]

- Given a set P of n points in $\mathbb{R}^d$ and a radius r, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability

- The algorithm:
  1. Place a d-dimensional axis-parallel grid of cell size c with a random shift
  2. Partition P into the grid cells
  3. Compute distances between all points in the same cell
  4. Report (p, q) if $||p - q||_2 \leq r$
  5. Repeat steps (1)-(4) m times

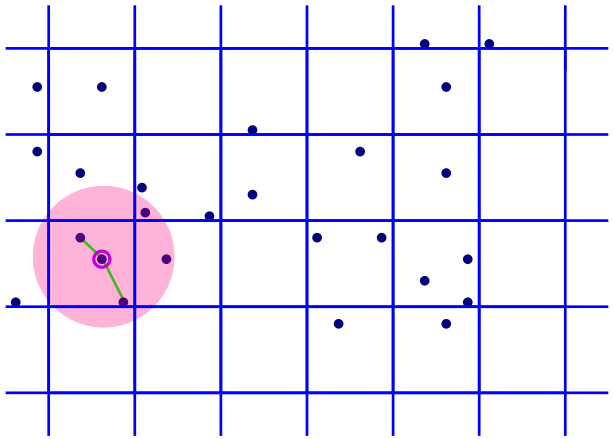


The algorithm is easy to implement (why?)
Running time is $C(d) \cdot (n + k) \log n$ ($C(d)$ is a constant, k is the number of reported points
The analysis requires a bit more work . . .

Connecting configurations—Randomly Transformed Grids [Aiger, Kaplan, Sharir14]

- Given a set P of n points in $\mathbb{R}^d$ and a radius r, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability

- The algorithm:
  1. Place a d-dimensional axis-parallel grid of cell size c with a random shift
  2. Partition P into the grid cells
  3. Compute distances between all points in the same cell
  4. Report (p, q) if $||p - q||_2 \leq r$
  5. Repeat steps (1)-(4) m times

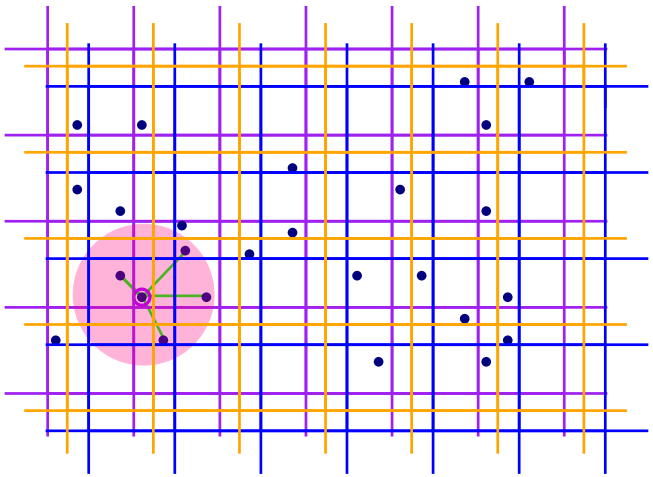


The algorithm is easy to implement (why?)
Running time is $C(d) \cdot (n + k) \log n$ ($C(d)$ is a constant, k is the number of reported points
The analysis requires a bit more work . . .

Connecting configurations—Randomly Transformed Grids [Aiger, Kaplan, Sharir14]

- Given a set P of n points in $\mathbb{R}^d$ and a radius r, RTG reports all-pairs of points $p, q \in P$ such that $||p - q||_2 \leq r$, with high probability

- The algorithm:
  1. Place a d-dimensional axis-parallel grid of cell size c with a random shift
  2. Partition P into the grid cells
  3. Compute distances between all points in the same cell
  4. Report (p, q) if $||p - q||_2 \leq r$
  5. Repeat steps (1)-(4) m times



The algorithm is easy to implement (why?)
Running time is $C(d) \cdot (n + k) \log n$ ($C(d)$ is a constant, k is the number of reported points
The analysis requires a bit more work . . .

Collision detection (CD)

- Collision detection is performed in the workspace $\mathcal{W}$
- Has to be fast—CD dominates the running time of many planners
- Often there is a trade-off between accuracy, speed, and memory usage

1: $\mathcal{V} = \emptyset$; $\mathcal{E} = \emptyset$; $\mathcal{G} \leftarrow (\mathcal{V}, \mathcal{E})$;
2: $\mathcal{V} \leftarrow$ `sample_free`$(\mathcal{X}, n)$
3: **for all** $x, y \in \mathcal{V}$ s.t. x is close to y **do**
4: **if** `collision_free`$(\overline{x, y})$ **then**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, y)\}$
6: **return** $\mathcal{G}$

- Depends on how obstacles in $\mathcal{W}$ or modeled (polygonal mesh, voxelized point cloud etc.)
- Depends if we are interested in CD for a point or for an edge

Collision detection (CD)—Bounding Volume Hierarchies

- A recursive partitioning of objects for quick pruning of irrelevant intersection tests, represented as a tree
- The root bounds the entire object
- The leaves bound a small number of features
- Queries answered by traversing two trees from the root to the leaves
- Variants depend on the type of bounding volume

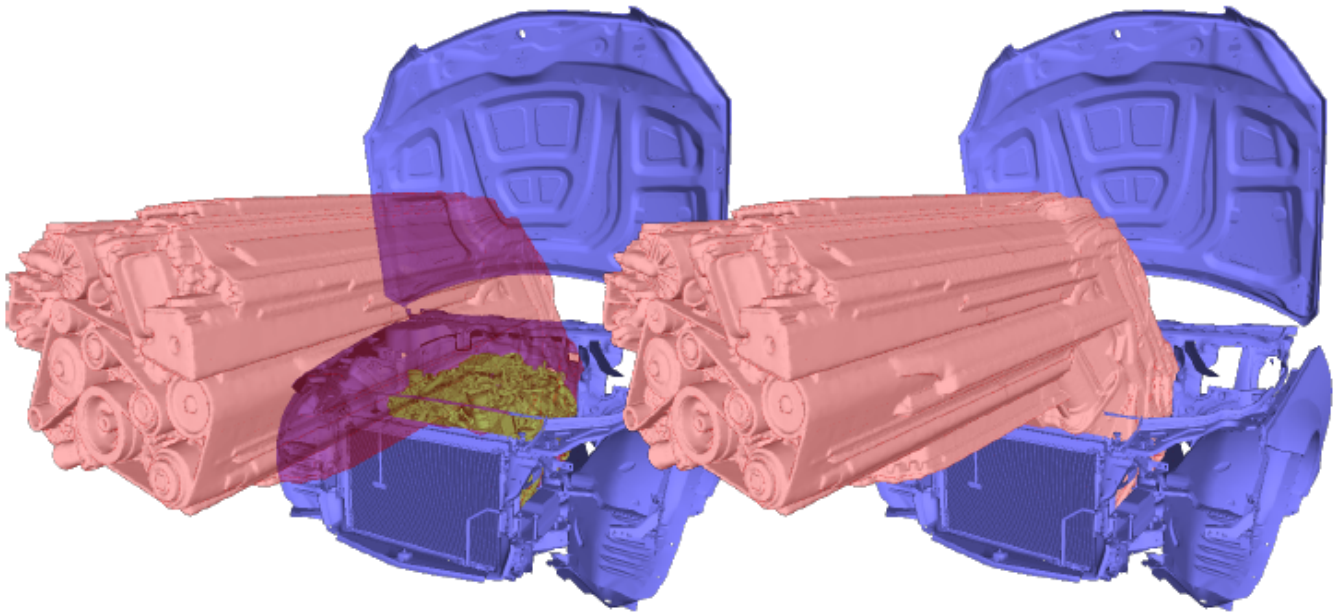
FCL `https://github.com/flexible-collision-library/fcl`—library for performing proximity queries on pairs of geometric models composed of triangles

Collision detection (CD)—Bounding Volume Hierarchies

Local planning—exact vs. approximate

We learned how to test if one configuration is collision free or not. What about a path in $\mathcal{X}$?

- Option (1)—exact CD via swept volume computation
- Typically computationally expensive

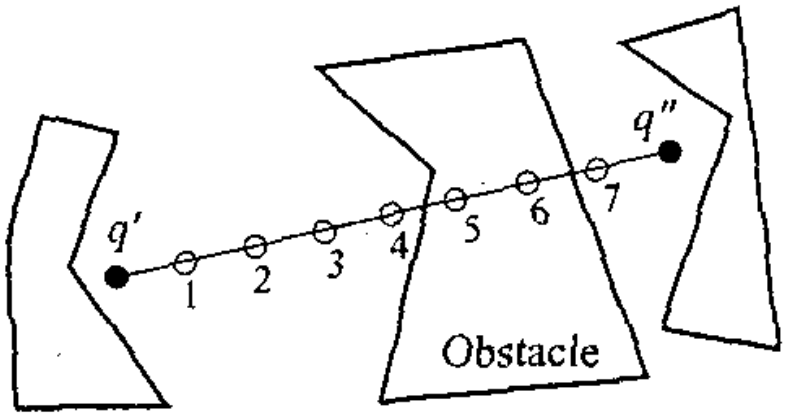


* Figure adapted from [von Dziegielewski et al.15]

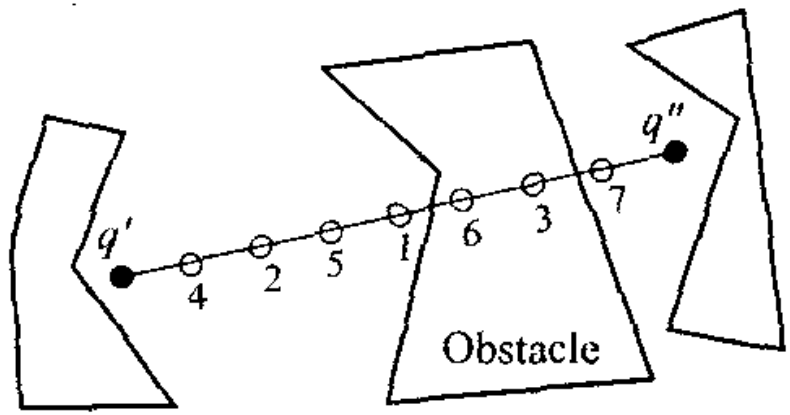
Local planning—exact vs. approximate

We learned how to test if one configuration is collision free or not. What about a path in $\mathcal{X}$?

- Option (2)—discretize path and run CD for each configuration
- Can avoid collisions by, e.g., inflating robot



(a) Incremental: The algorithm returns failure after five collision checks.



(b) Subdivision: The algorithm return failure after three collision checks.

* Figure adapted from Principles of robot motion [Choset et al.05]

Path planning

- We can run any shortest path algorithm (Dijkstra, A*, etc)
- If collision detection is performed online, this may be computationally expensive
- Lazy planners are often used (more on this in the future...)

1: **for all** $x \in \mathcal{V}$ s.t. x is close to x_{start} **do**
2: **if** `collision_free`$(\overline{x, x_{\text{start}}})$ **then**
3: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, x_{\text{start}})\}$
4: **for all** $x \in \mathcal{V}$ s.t. x is close to x_{goal} **do**
5: **if** `collision_free`$(\overline{x, x_{\text{goal}}})$ **then**
6: $\mathcal{E} \leftarrow \mathcal{E} \cup \{(x, x_{\text{goal}})\}$
7: **return** `shortest_path`$(\mathcal{G}, x_{\text{start}}, x_{\text{goal}})$

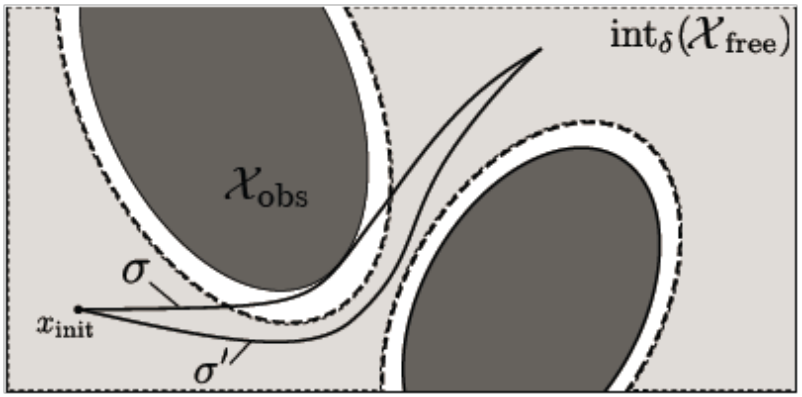
Probabilistic Completeness

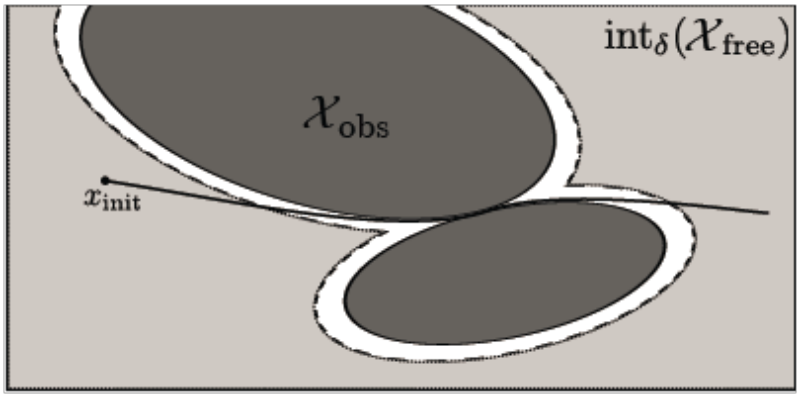
Definition [Probabilistic Completeness]

An algorithm ALG is probabilistically complete if, for any robustly-feasible motion-planning problem $\mathcal{P} = (\mathcal{X}_{\text{free}}, x_{init}, \mathcal{X}_{goal})$,

$$\lim_{n \to \infty} \Pr[\text{ALG returns a solution to } \mathcal{P}] = 1$$

- A robust solution remains a solution if obstacles are "dilated" by some small δ.

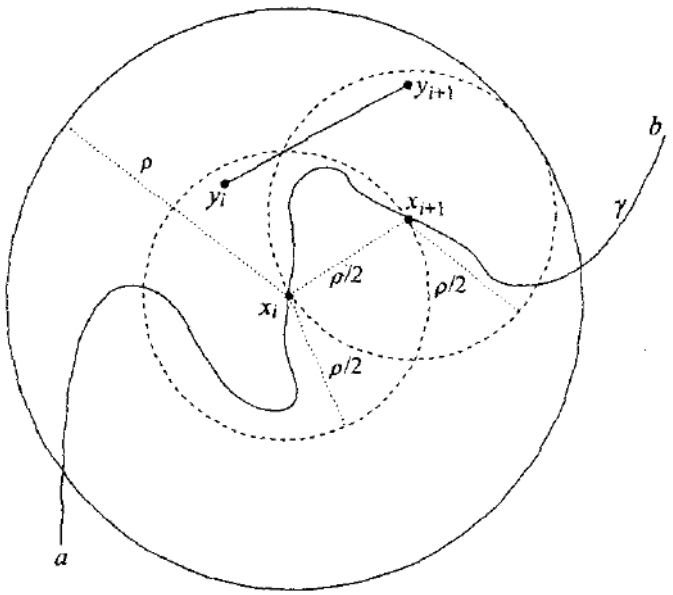




Probabilistic Completeness—PRM

Thm.

PRM with connection radius $r = \infty$ is probabilistically complete



Proof: on board (See also, Principles of Robot motion, Ch. 7.)

Computational complexity [Kleinbort,S.,Halperin 16]

Given n points sampled uniformly at random from a d-dimensional unit cube, what is k_r, the expected number of points in a ball of radius r?

$$k_r = n \cdot \mu(B_r(\cdot)) = n \cdot r^d \mu(B_1(\cdot)).$$

- $\mu(\cdot)$—volume of a set
- $B_r(\cdot)$—d-dimensional ball of radius r

Computational complexity [Kleinbort,S.,Halperin 16]

Given n points sampled uniformly at random from a d-dimensional unit cube, what is k_r, the expected number of points in a ball of radius r?

$$k_r = n \cdot \mu(B_r(\cdot)) = n \cdot r^d \mu(B_1(\cdot)).$$

- $\mu(\cdot)$—volume of a set
- $B_r(\cdot)$—d-dimensional ball of radius r

Computational complexity [Kleinbort,S.,Halperin 16]

The preprocessing step takes

$$T_{CD}(N) + T_{NN}(N)$$

- $T_{CD}(N)$—expected overall time spent on collision detection
- $T_{NN}(N)$—expected overall time spent on nearest-neighbor computation

Here,

$$T_{CD}(N) = \#_{CD} \cdot Q_{CD} + \#_{LP} \cdot Q_{LP}$$

Computational complexity [Kleinbort,S.,Halperin 16]

The preprocessing step takes

$$T_{CD}(N) + T_{NN}(N)$$

- $T_{CD}(N)$—expected overall time spent on collision detection
- $T_{NN}(N)$—expected overall time spent on nearest-neighbor computation

Here,

$$T_{CD}(N) = \#_{CD} \cdot Q_{CD} + \#_{LP} \cdot Q_{LP}$$

Computational complexity [Kleinbort,S.,Halperin 16]

The preprocessing step takes

$$T_{CD}(N) + T_{NN}(N)$$

- $T_{CD}(N)$—expected overall time spent on collision detection
- $T_{NN}(N)$—expected overall time spent on nearest-neighbor computation

Here,

$$T_{CD}(N) = \#_{CD} \cdot Q_{CD} + \#_{LP} \cdot Q_{LP}$$

- $\#_{CD} = N$
- $Q_{CD} = O(m^2)$— upper bound, in practice $\approx O(m \log^2 m)$
- $\#_{LP} = n \cdot k_r$
- $Q_{LP} = O(r \cdot Q_{CD}) = O(rm^2)$

Computational complexity [Kleinbort,S.,Halperin 16]

The preprocessing step takes

$$T_{CD}(N) + T_{NN}(N)$$

- $T_{CD}(N)$—expected overall time spent on collision detection
- $T_{NN}(N)$—expected overall time spent on nearest-neighbor computation

Here,

$$T_{CD}(N) = \#_{CD} \cdot Q_{CD} + \#_{LP} \cdot Q_{LP}$$

- $T_{NN}(N) = \Omega(n \cdot \log n + n \cdot k_r)$

Computational complexity [Kleinbort,S.,Halperin 16]

Take-home message:

- Asymptotically, nearest neighbor computation dominates the running time of the algorithm
- Local planning dominates collision detection time
- In practice, local planning dominates nearest-neighbor computation in many cases
- The radius r to connect neighbors has a *dramatic* effect on the running time of the algorithm

Connection radius r

Recall that for a connection radius r, the expected number of neighbors k_r is

$$k_r = O(n \cdot r^d).$$

- $r = O(1)$—Then $k_r = O(n)$
- $r = O\left(\left(\frac{1}{n}\right)^{1/d}\right)$—Then $k_r = O(1)$
- $r = O\left(\left(\frac{\log n}{n}\right)^{1/d}\right)$—Then $k_r = O(\log n)$

What happens if r is too small? too large?

Connection radius r

Recall that for a connection radius r, the expected number of neighbors k_r is

$$k_r = O(n \cdot r^d).$$

- $r = O(1)$—Then $k_r = O(n)$
- $r = O\left(\left(\frac{1}{n}\right)^{1/d}\right)$—Then $k_r = O(1)$
- $r = O\left(\left(\frac{\log n}{n}\right)^{1/d}\right)$—Then $k_r = O(\log n)$

What happens if r is too small? too large?

Connection radius r

Recall that for a connection radius r, the expected number of neighbors k_r is

$$k_r = O(n \cdot r^d).$$

- $r = O(1)$—Then $k_r = O(n)$
- $r = O\left(\left(\frac{1}{n}\right)^{1/d}\right)$—Then $k_r = O(1)$
- $r = O\left(\left(\frac{\log n}{n}\right)^{1/d}\right)$—Then $k_r = O(\log n)$

What happens if r is too small? too large?

PRM * [Karaman Frazzoli11]

Run PRM with connection radius

$$r(n) \approx \gamma_{\text{PRM}}(d) \left(\frac{\log n}{n} \right)^{1/d}$$

n - number of nodes, d - dimension of configuration space

Thm. 20 [Karaman, Frazzoli11]

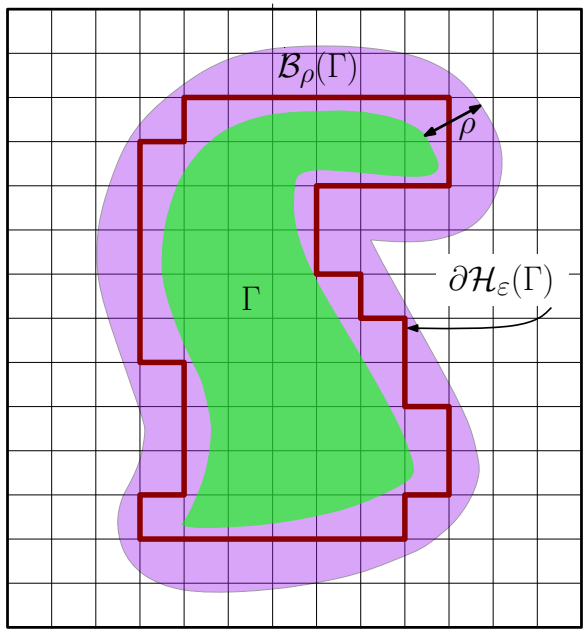
There exists a constant $\gamma > 0$ such that PRM with connection radius $r(n) = \gamma n^{1/d}$ is not probabilistically complete.

Thm. 34 [Karaman, Frazzoli11]

The PRM * algorithm is asymptotically optimal.

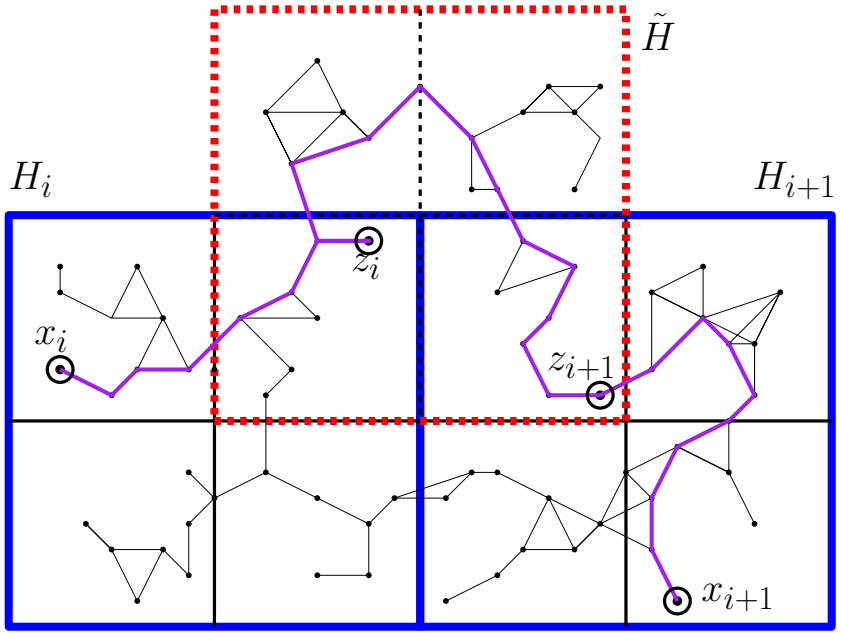
PRM *—analysis

- Karaman and Frazzolli [Karaman, Frazzoli11] proved that PRM * is asymptotically optimal.
- They conjectured that a sampling-based planner has a certain property if the underlying Random Geometric Graph (RGG) has the same property
- This was settled by Solovey et al. [Solovey S. Halperin 18]



PRM *—analysis

- Karaman and Frazzolli [Karaman, Frazzoli11] proved that PRM * is asymptotically optimal.
- They conjectured that a sampling-based planner has a certain property if the underlying Random Geometric Graph (RGG) has the same property
- This was settled by Solovey et al. [Solovey S. Halperin 18]



Further reading

- Sampling-based motion planning—LaValle's book [Ch.5], Principles of Robot Motion [Ch.7],
- Collision detection—CG book [Ch. 39]
- Nearest Neighbor search—CG book [Ch. 43]