Introduction to Robotics

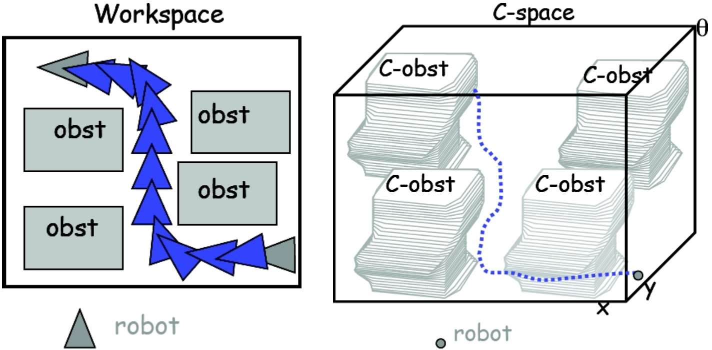
Sampling-Based Motion Planning

Erion Plaku

Department of Electrical Engineering and Computer Science
Catholic University of America

From Workspace to Configuration Space

- simple workspace obstacle transformed into complex configuration-space obstacle
- robot transformed into point in configuration space
- path transformed from swept volume to 1d curve



[fig from Jyh-Ming Lien]

Explicit Construction of Configuration Space/Roadmaps

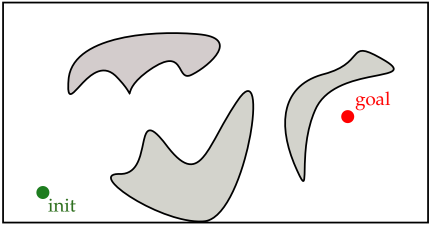
- PSPACE-complete
- Exponential dependency on dimension
- No practical algorithms

Motion Planning for a Point Robot in 2D

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

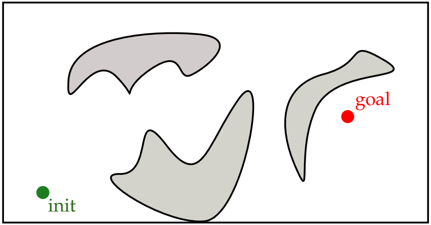
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?



- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

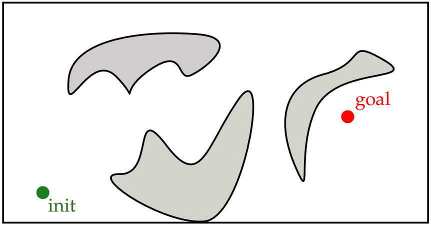
How would you solve it?



Hint: How would you approximate π?

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

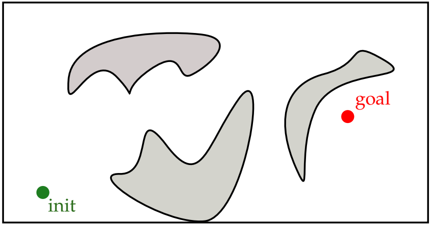
How would you solve it?



Hint: How would you approximate π?

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

How would you solve it?

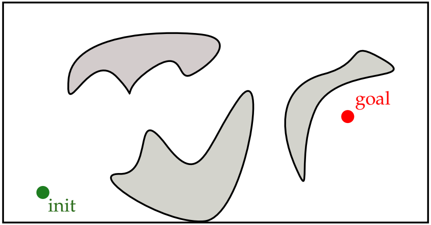


Hint: How would you approximate π?




- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

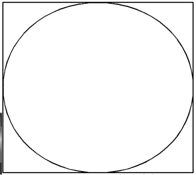
How would you solve it?



Hint: How would you approximate π?

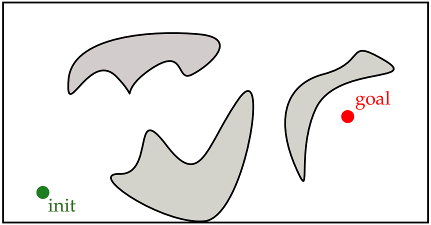






- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

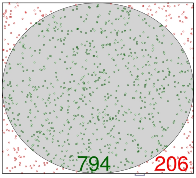
How would you solve it?



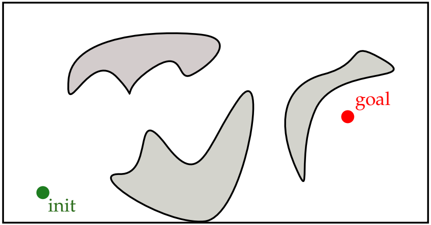
Hint: How would you approximate π?







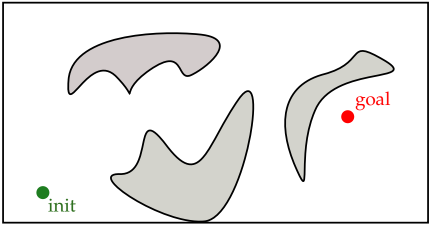
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



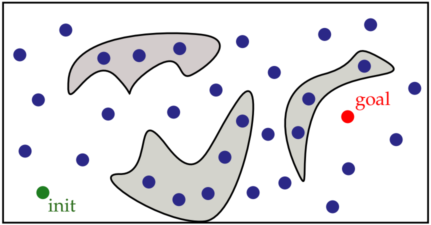
Monte-Carlo Idea:

- Define input space
- Generate inputs at random by *sampling* the input space
- Perform a deterministic computation using the input samples
- Aggregate the partial results into final result

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position

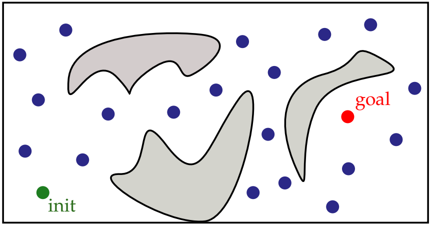


- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



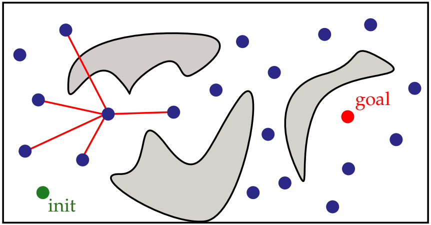
- Sample points

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



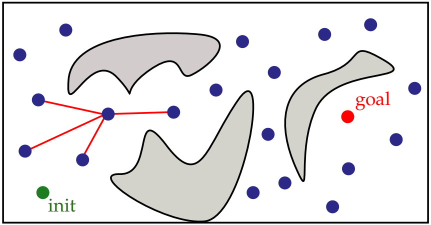
- Sample points
- Discard samples that are in collision

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



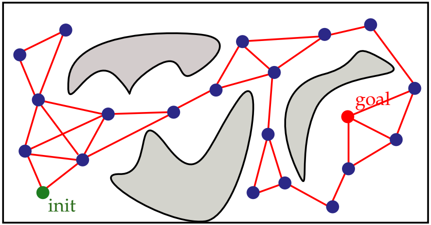
- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

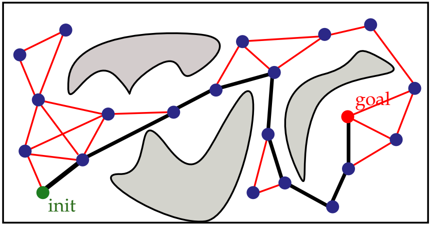
- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

⇒ Gives rise to a graph, called the *roadmap*

- Robotic system: Single point
- Task: Compute collision-free path from initial to goal position



- Sample points
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- Discard straight-line segments that are in collision

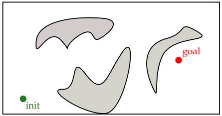
⇒ Gives rise to a graph, called the *roadmap*

⇒ Collision-free path can be found by performing graph search on the roadmap

[Kavraki, Švestka, Latombe, Overmars 1996]

0. Initialization

add q_{init} and q_{goal} to roadmap vertex set V



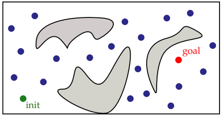
1. Sampling

repeat several times

$q \leftarrow \text{SAMPLE}()$

if $\text{ISCOLLISIONFREE}(q) = \texttt{true}$

add q to roadmap vertex set V



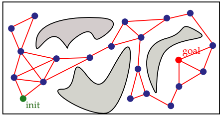
2. Connect Samples

for each pair of neighboring samples $(q_a, q_b) \in V \times V$

$\text{path} \leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$

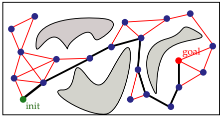
if $\text{ISCOLLISIONFREE}(\text{path}) = \texttt{true}$

add (q_a, q_b) to roadmap edge set E



3. Graph Search

search graph (V, E) for path from q_{init} to q_{goal}



Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach?

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach? No. It's more than that

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

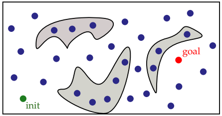
Is it then just a heuristic approach? No. It's more than that

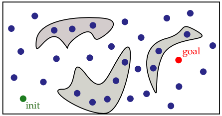
It offers *probabilistic completeness*

- When a solution exists, a probabilistically complete planner finds a solution with probability as time goes to infinity.
- When a solution does not exists, a probabilistically complete planner may not be able to determine that a solution does not exist.

$q = (x, y) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\mathsf{min}_x, \mathsf{max}_x)$
- $y \leftarrow \textsc{rand}(\mathsf{min}_y, \mathsf{max}_y)$



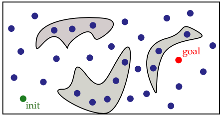


$q = (x, y) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$
- $y \leftarrow \textsc{rand}(\min_y, \max_y)$

$\textsc{IsSampleCollisionFree}(q)$

- Point inside/outside polygon test



$q = (x, y) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$
- $y \leftarrow \textsc{rand}(\min_y, \max_y)$

$\textsc{IsSampleCollisionFree}(q)$

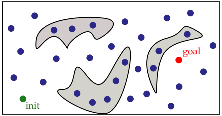
- Point inside/outside polygon test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Straight-line segment from point q_a to point q_b

$q = (x, y) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$
- $y \leftarrow \textsc{rand}(\min_y, \max_y)$



$\textsc{IsSampleCollisionFree}(q)$

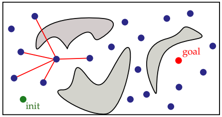
- Point inside/outside polygon test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

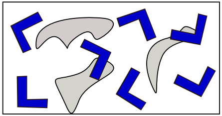
- Straight-line segment from point q_a to point q_b

$\textsc{IsPathCollisionFree}(\text{path})$

- Segment-polygon intersection test



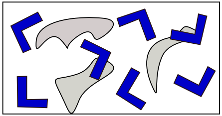
PRM Applied to 2D Rigid-Body Robot



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$;
  $\theta \leftarrow \text{RAND}(-\pi, \pi)$

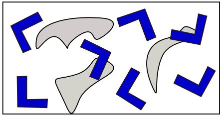
PRM Applied to 2D Rigid-Body Robot



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

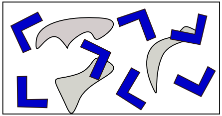


$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\mathsf{min}_x, \mathsf{max}_x)$; $y \leftarrow \textsc{rand}(\mathsf{min}_y, \mathsf{max}_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test



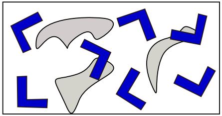
$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

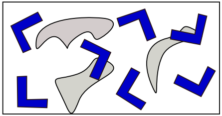
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

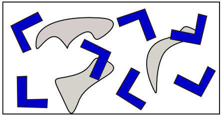
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$;
  $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

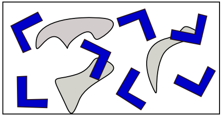
$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

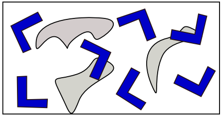
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\mathrm{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\mathrm{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\mathrm{path}(0) = q_a, \mathrm{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\mathrm{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\mathrm{path})$



$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$;
  $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

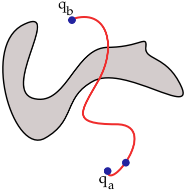
$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

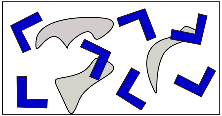
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$



$\textsc{IsPathCollisionFree}(\text{path})$

- Incremental approach



$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$

- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$; $\theta \leftarrow \text{RAND}(-\pi, \pi)$

$\text{ISSAMPLECOLLISIONFREE}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

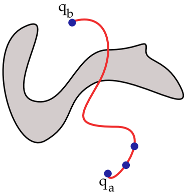
$\text{path} \leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$

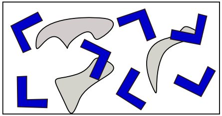
- Continuous function parameterized by time: $\text{path} : [0, 1] \to Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\text{ISPATHCOLLISIONFREE}(\text{path})$

- Incremental approach





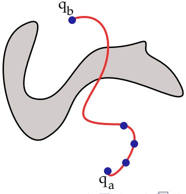
$q = (x, y, \theta) \leftarrow \textsc{Sample}()$
- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$;
  $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

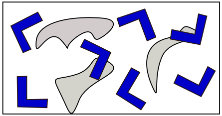
$\textsc{IsSampleCollisionFree}(q)$
- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$
- Incremental approach





$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

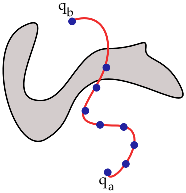
$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

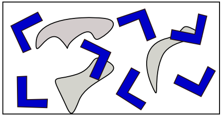
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$

- Incremental approach





$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

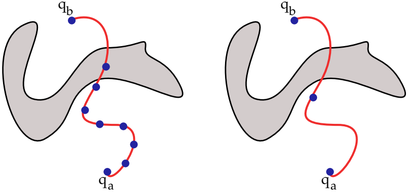
$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

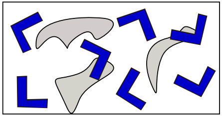
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$

- Incremental approach
- Subdivision approach





$q = (x, y, \theta) \leftarrow \textsc{Sample}()$

- $x \leftarrow \textsc{rand}(\min_x, \max_x)$; $y \leftarrow \textsc{rand}(\min_y, \max_y)$; $\theta \leftarrow \textsc{rand}(-\pi, \pi)$

$\textsc{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

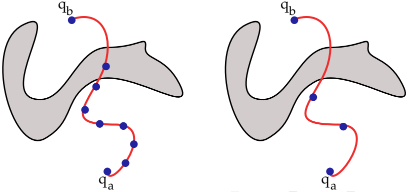
$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

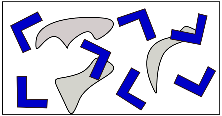
- Continuous function parameterized by time: $\text{path} : [0, 1] \to Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$

- Incremental approach
- Subdivision approach





$q = (x, y, \theta) \leftarrow \text{SAMPLE}()$

- $x \leftarrow \text{RAND}(\min_x, \max_x)$; $y \leftarrow \text{RAND}(\min_y, \max_y)$;
  $\theta \leftarrow \text{RAND}(-\pi, \pi)$

$\text{ISSAMPLECOLLISIONFREE}(q)$

- Place rigid body in position and orientation specified by q
- Polygon-polygon intersection test

$\text{path} \leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$

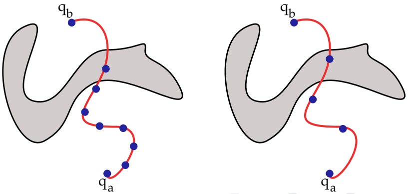
- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\text{ISPATHCOLLISIONFREE}(\text{path})$

- Incremental approach
- Subdivision approach

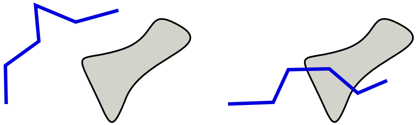
[piano] [manocha] [kcar] [tri] [buggy]

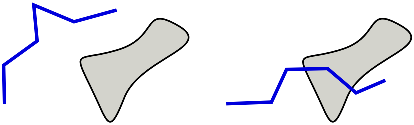


PRM Applied to Articulated Chain

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$

- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$





$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$

- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi)$, $\forall i \in [1, n]$

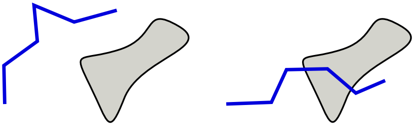
$\textsc{IsSampleCollisionFree}(q)$

- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

PRM Applied to Articulated Chain

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$

- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi), \forall i \in [1, n]$



$\textsc{IsSampleCollisionFree}(q)$

- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

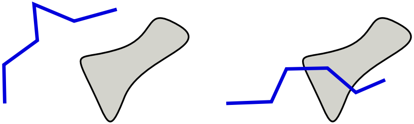
$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \textsc{Sample}()$

- $\theta_i \leftarrow \textsc{rand}(-\pi, \pi), \forall i \in [1, n]$



$\textsc{IsSampleCollisionFree}(q)$

- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles

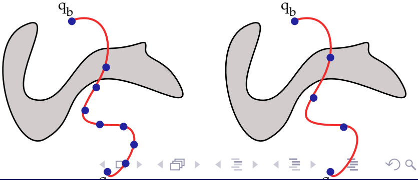
$\text{path} \leftarrow \textsc{GenerateLocalPath}(q_a, q_b)$

- Continuous function parameterized by time: $\text{path} : [0, 1] \rightarrow Q$
- Starts at q_a and ends at q_b: $\text{path}(0) = q_a, \text{path}(1) = q_b$
- Many possible ways of defining it, e.g., by linear interpolation

$$\text{path}(t) = (1 - t) * q_a + t * q_b$$

$\textsc{IsPathCollisionFree}(\text{path})$

- Incremental approach
- Subdivision approach



[everest] [skeleton] [knot] [manip]

Path Smoothing

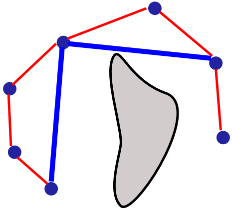
- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

Path Smoothing

- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

SMOOTHPATH($q_1, q_2, \ldots, q_n$) – one version

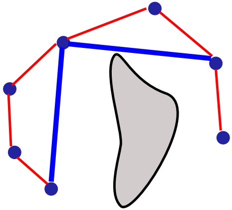
1: **for** several times **do**
2: select i and j uniformly at random from $1, 2, \ldots, n$
3: attempt to directly connect q_i to q_j
4: if successful, remove the in-between nodes, i.e., $q_{i+1}, \ldots, q_j$



- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- Post processing is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths

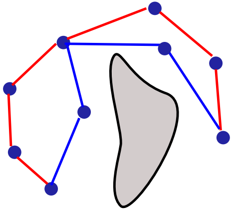
SmoothPath($q_1, q_2, \ldots, q_n$) – one version

```
for several times do
    select i and j uniformly at random from 1, 2, ..., n
    attempt to directly connect q_i to q_j
    if successful, remove the in-between nodes, i.e., q_{i+1}, ..., q_j
```

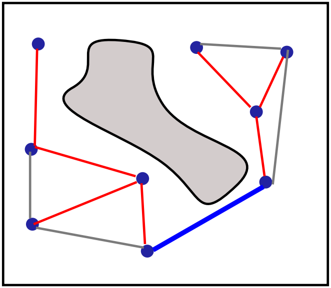


SmoothPath($q_1, q_2, \ldots, q_n$) – another version

```
for several times do
    select i and j uniformly at random from 1, 2, ..., n
    q ← generate collision-free sample
    attempt to connect q_i to q_j through q
    if successful, replace the in-between nodes q_{i+1}, ..., q_j by q
```

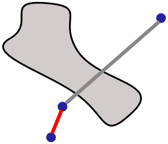


- Edge in cycle does not improve roadmap connectivity
- Edge is added to roadmap only if it connects two different roadmap components

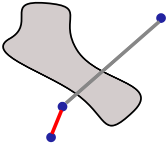


1: **if** SAMEROADMAPCOMPONENT(q_a, q_b) = `false` **then**
2: path ← GENERATEPATH(q_a, q_b)
3: **if** ISPATHCOLLISIONFREE(path) = `true` **then**
4: (q_a, q_b).path ← path
5: $E \leftarrow E \cup \{(q_a, q_b)\}$

- Disjoint-set data structure is used to speed up computation of SAMEROADMAPCOMPONENT(q_a, q_b)

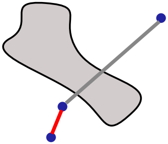


Edges between neighboring nodes are more likely to be collision free than edges between far away nodes



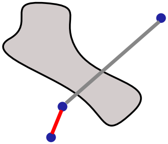
Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

- Common practice is to attempt to connect each node to k of its nearest neighbors



Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \to \mathbb{R}^{\geq 0}$, e.g.,

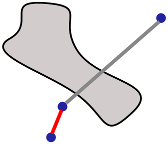


Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0}$, e.g.,
  - geodesic distances, i.e., shortest-path distances according to topology
  - weighted combination of translation and rotation components
  - Euclidean distance between selected robot points

Connecting Roadmap Nodes to Nearest Neighbors

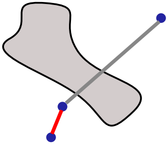
Edges between neighboring nodes are more likely to be collision free than edges between far away nodes



- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0}$, e.g.,
  - geodesic distances, i.e., shortest-path distances according to topology
  - weighted combination of translation and rotation components
  - Euclidean distance between selected robot points

Good distance metrics reflect the likelihood of successful connections

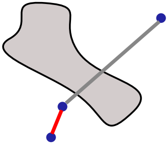
Connecting Roadmap Nodes to Nearest Neighbors



Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0}$, e.g.,
  - geodesic distances, i.e., shortest-path distances according to topology
  - weighted combination of translation and rotation components
  - Euclidean distance between selected robot points

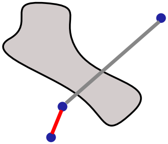
  Good distance metrics reflect the likelihood of successful connections
- Numerous algorithms/data structures for nearest-neighbors computations, e.g., kd-tree, R-tree, M-tree, V-tree, PR-tree, GNAT, iDistance, CoverTree



Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0}$, e.g.,
  - geodesic distances, i.e., shortest-path distances according to topology
  - weighted combination of translation and rotation components
  - Euclidean distance between selected robot points

  Good distance metrics reflect the likelihood of successful connections
- Numerous algorithms/data structures for nearest-neighbors computations, e.g., kd-tree, R-tree, M-tree, V-tree, PR-tree, GNAT, iDistance, CoverTree
- Computational challenges of nearest neighbors in high-dimensional spaces
  - Efficiency deteriorates rapidly
  - Not much better than brute-force approach



Edges between neighboring nodes are more likely to be collision free than edges between far away nodes

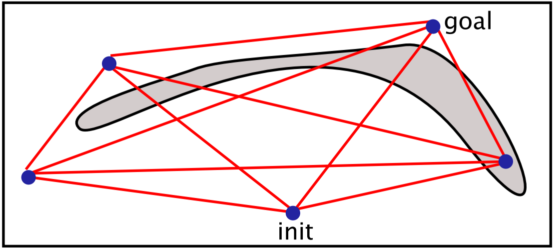
- Common practice is to attempt to connect each node to k of its nearest neighbors
- Nearest neighbors defined by some distance metric $\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0}$, e.g.,
  - geodesic distances, i.e., shortest-path distances according to topology
  - weighted combination of translation and rotation components
  - Euclidean distance between selected robot points

  Good distance metrics reflect the likelihood of successful connections
- Numerous algorithms/data structures for nearest-neighbors computations, e.g., kd-tree, R-tree, M-tree, V-tree, PR-tree, GNAT, iDistance, CoverTree
- Computational challenges of nearest neighbors in high-dimensional spaces
  - Efficiency deteriorates rapidly
  - Not much better than brute-force approach
- Alternative approach is to compute *approximate* nearest neighbors

  [Plaku, Kavraki: WAFR 2006, SDM 2007]
  - Minimal losses in accuracy of neighbors
  - No loss in accuracy of overall path planner
  - Significant computational gains

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



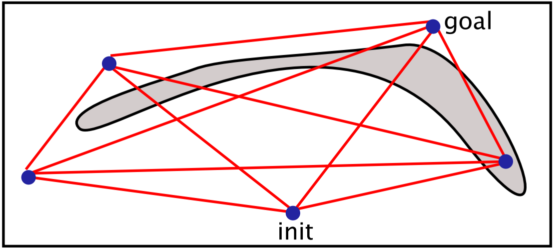
LazyRoadmapConstruction

1: $V \leftarrow V \cup \{q_{\text{init}}, q_{\text{goal}}\}$; $E \leftarrow \emptyset$
2: **for** several times **do**
3: $q \leftarrow$ generate config uniformly at random; $q.\text{checked} \leftarrow$ `false`; $V \leftarrow V \cup \{q\}$
4: **for** each pair $(q_a, q_b) \in V \times V$ **do**
5: $(q_a, q_b).\text{res} \leftarrow 1.0$; $(q_a, q_b).\text{path} \leftarrow \text{GeneratePath}(q_a, q_b)$; $E \leftarrow E \cup \{(q_a, q_b)\}$

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



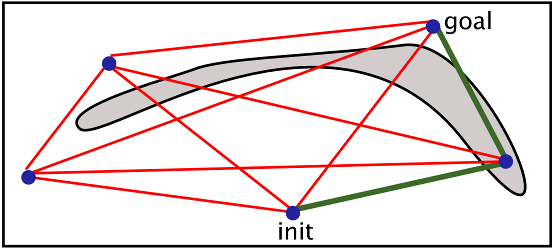
LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\text{checked} = \texttt{false}$ **and** IsConfigCollisionFree$(q_i) = \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\text{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\text{res} \leftarrow (q_i, q_{i+1}).\text{res}/2$; check $(q_i, q_{i+1}).\text{path}$ at resolution $(q_i, q_{i+1}).\text{res}$
11: **if** collision found in $(q_i, q_{i+1}).\text{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\text{path} \circ \cdots \circ (q_{n-1}, q_n).\text{path}$

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



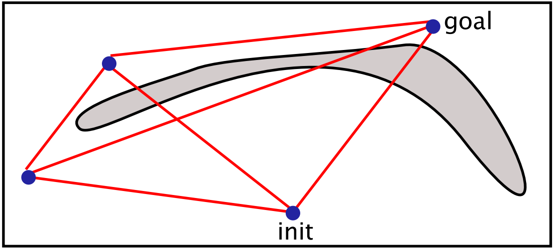
LazyRoadmapCollisionChecking

```
1:  for several times do
2:     [q_1, q_2, ..., q_n] ← search G = (V, E) for sequence of edges connecting q_init to q_goal
3:     for i = 1, 2, ..., n do
4:        if q_i.checked = false and IsConfigCollisionFree(q_i) = false then
5:           remove q_i from roadmap; goto line 2
6:        else
7:           q_i.checked ← true
8:     while no edge collisions are found and minimum resolution not reached do
9:        for i = 1, 2, ..., n − 1 do
10:          (q_i, q_{i+1}).res ← (q_i, q_{i+1}).res/2; check (q_i, q_{i+1}).path at resolution (q_i, q_{i+1}).res
11:          if collision found in (q_i, q_{i+1}).path then
12:             remove (q_i, q_{i+1}) from roadmap; goto line 2
13:    return (q_1, q_2).path ∘ ··· ∘ (q_{n−1}, q_n).path
```

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



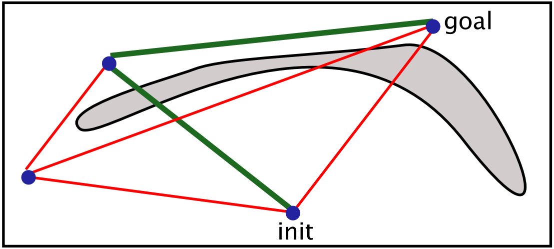
LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting $q_{\rm init}$ to $q_{\rm goal}$
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\mathrm{checked} = \texttt{false}$ **and** IsConfigCollisionFree$(q_i) = \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\mathrm{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\mathrm{res} \leftarrow (q_i, q_{i+1}).\mathrm{res}/2$; check $(q_i, q_{i+1}).\mathrm{path}$ at resolution $(q_i, q_{i+1}).\mathrm{res}$
11: **if** collision found in $(q_i, q_{i+1}).\mathrm{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\mathrm{path} \circ \cdots \circ (q_{n-1}, q_n).\mathrm{path}$

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



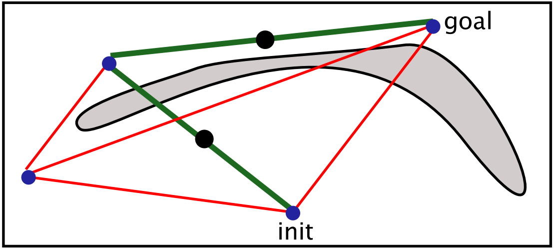
LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\text{checked} = \texttt{false}$ **and** IsConfigCollisionFree(q_i) $= \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\text{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\text{res} \leftarrow (q_i, q_{i+1}).\text{res}/2$; check $(q_i, q_{i+1}).\text{path}$ at resolution $(q_i, q_{i+1}).\text{res}$
11: **if** collision found in $(q_i, q_{i+1}).\text{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\text{path} \circ \cdots \circ (q_{n-1}, q_n).\text{path}$

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]

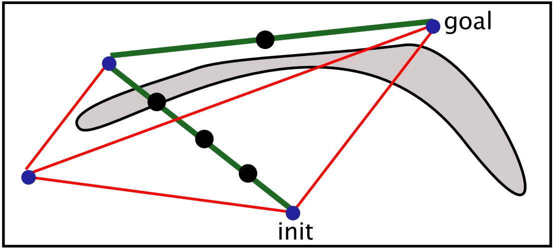


LazyRoadmapCollisionChecking

```
1:  for several times do
2:     [q_1, q_2, ..., q_n] ← search G = (V, E) for sequence of edges connecting q_init to q_goal
3:     for i = 1, 2, ..., n do
4:        if q_i.checked = false and IsConfigCollisionFree(q_i) = false then
5:           remove q_i from roadmap; goto line 2
6:        else
7:           q_i.checked ← true
8:     while no edge collisions are found and minimum resolution not reached do
9:        for i = 1, 2, ..., n − 1 do
10:          (q_i, q_{i+1}).res ← (q_i, q_{i+1}).res/2; check (q_i, q_{i+1}).path at resolution (q_i, q_{i+1}).res
11:          if collision found in (q_i, q_{i+1}).path then
12:             remove (q_i, q_{i+1}) from roadmap; goto line 2
13:    return (q_1, q_2).path ∘ ··· ∘ (q_{n−1}, q_n).path
```

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



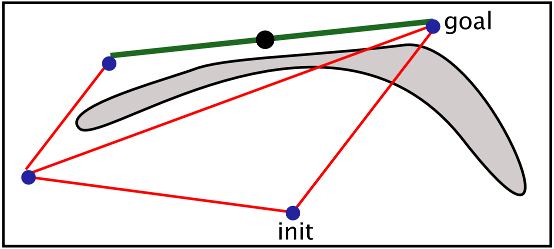
LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\text{checked} = \texttt{false}$ **and** IsConfigCollisionFree(q_i) $= \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\text{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\text{res} \leftarrow (q_i, q_{i+1}).\text{res}/2$; check $(q_i, q_{i+1}).\text{path}$ at resolution $(q_i, q_{i+1}).\text{res}$
11: **if** collision found in $(q_i, q_{i+1}).\text{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\text{path} \circ \cdots \circ (q_{n-1}, q_n).\text{path}$

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



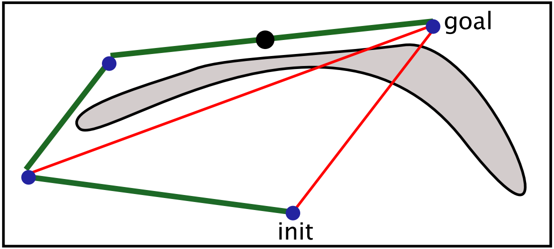
LazyRoadmapCollisionChecking

```
1:  for several times do
2:     [q_1, q_2, ..., q_n] ← search G = (V, E) for sequence of edges connecting q_init to q_goal
3:     for i = 1, 2, ..., n do
4:        if q_i.checked = false and IsConfigCollisionFree(q_i) = false then
5:           remove q_i from roadmap; goto line 2
6:        else
7:           q_i.checked ← true
8:     while no edge collisions are found and minimum resolution not reached do
9:        for i = 1, 2, ..., n − 1 do
10:          (q_i, q_{i+1}).res ← (q_i, q_{i+1}).res/2; check (q_i, q_{i+1}).path at resolution (q_i, q_{i+1}).res
11:          if collision found in (q_i, q_{i+1}).path then
12:             remove (q_i, q_{i+1}) from roadmap; goto line 2
13:    return (q_1, q_2).path ∘ ··· ∘ (q_{n−1}, q_n).path
```

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



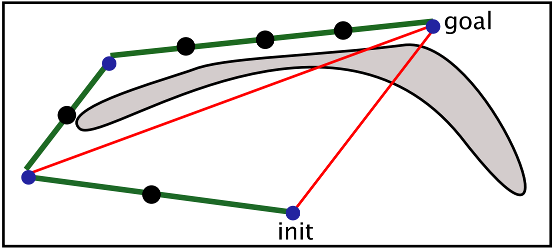
LazyRoadmapCollisionChecking

```
1:  for several times do
2:     [q_1, q_2, ..., q_n] ← search G = (V, E) for sequence of edges connecting q_init to q_goal
3:     for i = 1, 2, ..., n do
4:        if q_i.checked = false and IsConfigCollisionFree(q_i) = false then
5:           remove q_i from roadmap; goto line 2
6:        else
7:           q_i.checked ← true
8:     while no edge collisions are found and minimum resolution not reached do
9:        for i = 1, 2, ..., n − 1 do
10:          (q_i, q_{i+1}).res ← (q_i, q_{i+1}).res/2; check (q_i, q_{i+1}).path at resolution (q_i, q_{i+1}).res
11:          if collision found in (q_i, q_{i+1}).path then
12:             remove (q_i, q_{i+1}) from roadmap; goto line 2
13:    return (q_1, q_2).path ∘ ··· ∘ (q_{n−1}, q_n).path
```

Lazy PRM

Perform collision checking only when necessary

[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



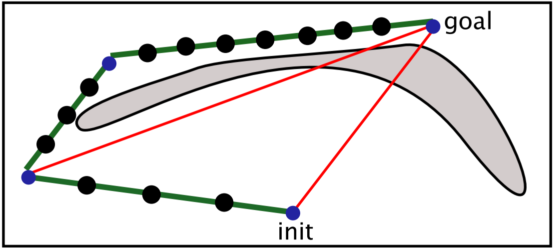
LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\text{checked} = \texttt{false}$ **and** IsConfigCollisionFree(q_i) $= \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\text{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\text{res} \leftarrow (q_i, q_{i+1}).\text{res}/2$; check $(q_i, q_{i+1}).\text{path}$ at resolution $(q_i, q_{i+1}).\text{res}$
11: **if** collision found in $(q_i, q_{i+1}).\text{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\text{path} \circ \cdots \circ (q_{n-1}, q_n).\text{path}$

Lazy PRM

Perform collision checking only when necessary

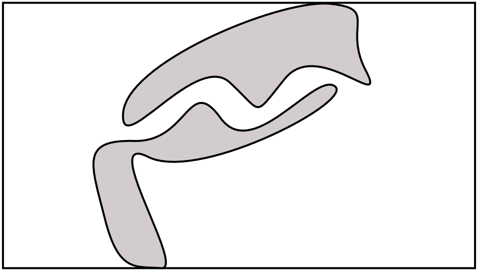
[Bohlin, Kavraki: Handbook on Randomized Computing 2000]



LazyRoadmapCollisionChecking

1: **for** several times **do**
2: $[q_1, q_2, \ldots, q_n] \leftarrow$ search $G = (V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1, 2, \ldots, n$ **do**
4: **if** $q_i.\text{checked} = \texttt{false}$ **and** IsConfigCollisionFree$(q_i) = \texttt{false}$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.\text{checked} \leftarrow \texttt{true}$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n-1$ **do**
10: $(q_i, q_{i+1}).\text{res} \leftarrow (q_i, q_{i+1}).\text{res}/2$; check $(q_i, q_{i+1}).\text{path}$ at resolution $(q_i, q_{i+1}).\text{res}$
11: **if** collision found in $(q_i, q_{i+1}).\text{path}$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).\text{path} \circ \cdots \circ (q_{n-1}, q_n).\text{path}$

Narrow-Passage Problem



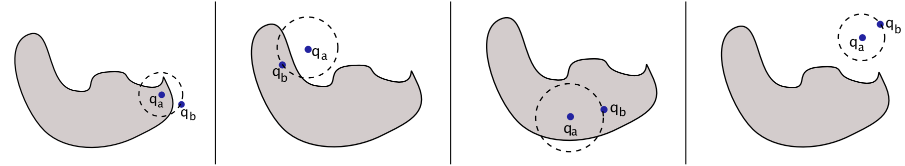
- Probability of generating samples via uniform sampling in a narrow passage is low due to the small volume of the narrow passage
- Generating samples inside a narrow passage may be critical to the success of the path planner
- Objective is then to design sampling strategies that can increase the probability of generating samples inside narrow passages

Gaussian Sampling in PRM

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Sample from a Gaussian distribution biased near the obstacles

GenerateCollisionFreeConfig [Boor, Overmars, van Der Stappen: ICRA 1999]

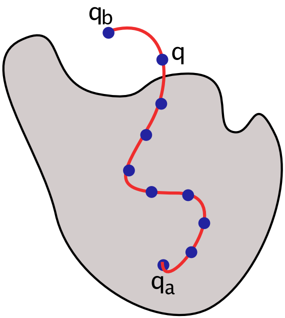
$q_a \leftarrow$ generate config uniformly at random
$r \leftarrow$ generate distance from Gaussian distribution
$q_b \leftarrow$ generate config uniformly at random at distance r from q_a
$\text{ok}_a \leftarrow \text{IsConfigCollisionFree}(q_a)$
$\text{ok}_b \leftarrow \text{IsConfigCollisionFree}(q_b)$
if $\text{ok}_a =$ `true` **and** $\text{ok}_b =$ `false` **then return** q_a
if $\text{ok}_a =$ `false` **and** $\text{ok}_b =$ `true` **then return** q_b
return `null`



Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Move samples in collision outside obstacle boundary

GENERATECOLLISIONFREECONFIG [Amato, Bayazit, Dale, Jones, Vallejo: WAFR 1998]

```
q_a ← generate config uniformly at random
if IsConfigCollisionFree(q_a) = true then
    return q_a
else
    q_b ← generate config uniformly at random
    path ← GeneratePath(q_a, q_b)
    for t = δ to |path| by δ do
        if IsConfigCollisionFree(path(t)) then
            return path(t)
return null
```



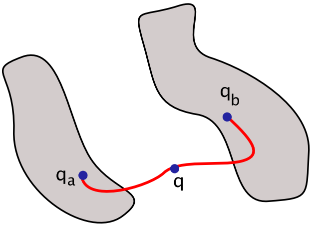
Bridge-based Sampling in PRM

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Create "bridge" between samples in collision

[Hsu, Jiang, Reif, Sun: ICRA 2003]

GENERATECOLLISIONFREECONFIG

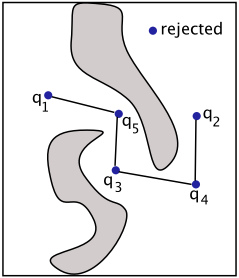
```
q_a ← generate config uniformly at random
q_b ← generate config uniformly at random

ok_a ← IsConfigCollisionFree(q_a)
ok_b ← IsConfigCollisionFree(q_b)

if ok_a = false and ok_b = false then
    path ← GeneratePath(q_a, q_b)
    q ← path(0.5|path|)
    if IsConfigCollisionFree(q) then
        return q

return null
```



Objective: Capture connectivity of configuration space with few samples
Approach: Generate samples that create new components or join existing components

GENERATECOLLISIONFREECONFIG [Nisseoux, Simeon, Laumond: Advanced Robotics J 2000]

1: $q \leftarrow$ generate config uniformly at random
2: **if** ISCONFIGCOLLISIONFREE(q) = `true` **then**
3: **if** q belongs to a new roadmap component **then**
4: **return** q
5: **if** q connects two roadmap components **then**
6: **return** q
7: **return** `null`



- q_1: creates new roadmap component
- q_2: creates new roadmap component
- q_3: creates new roadmap component
- q_4: connects two roadmap components
- q_5: connects two roadmap components

Importance Sampling

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Improve roadmap connectivity

- Construct roadmap using given sampling strategy
- Identify roadmap nodes that lie in regions that are hard to connect
- Sample more in these regions

Importance Sampling

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Improve roadmap connectivity

- Construct roadmap using given sampling strategy
- Identify roadmap nodes that lie in regions that are hard to connect
- Sample more in these regions

- Associate weight $w(q)$ with each configuration q in the roadmap
- Weight $w(q)$ indicates difficulty of region around q

Importance Sampling

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Improve roadmap connectivity

- Construct roadmap using given sampling strategy
- Identify roadmap nodes that lie in regions that are hard to connect
- Sample more in these regions

- Associate weight $w(q)$ with each configuration q in the roadmap
- Weight $w(q)$ indicates difficulty of region around q
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) =$ number of times connections from/to q have failed
  - combination of different strategies

Importance Sampling

Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Improve roadmap connectivity

- Construct roadmap using given sampling strategy
- Identify roadmap nodes that lie in regions that are hard to connect
- Sample more in these regions

- Associate weight $w(q)$ with each configuration q in the roadmap
- Weight $w(q)$ indicates difficulty of region around q
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) =$ number of times connections from/to q have failed
  - combination of different strategies

- Select sample with probability $\frac{w(q)}{\sum_{q' \in V} w(q')}$
- Generate more samples around q
- Connect new samples to neighboring roadmap nodes

Combine Different Sampling Strategies

- Each sampling strategy has its strengths and weakness
- Objective is to identify the appropriate sampling strategy for a given region

Combine Different Sampling Strategies

- Each sampling strategy has its strengths and weakness
- Objective is to identify the appropriate sampling strategy for a given region

- One common strategy is to assign a weight w_i to each sampler S_i
- A sampler S_i is then selected with probability

$$\frac{w_i}{\sum_j w_j}$$

- Sampler weight is updated based on quality of performance

Combine Different Sampling Strategies

- Each sampling strategy has its strengths and weakness
- Objective is to identify the appropriate sampling strategy for a given region

- One common strategy is to assign a weight w_i to each sampler S_i
- A sampler S_i is then selected with probability

$$\frac{w_i}{\sum_j w_j}$$

- Sampler weight is updated based on quality of performance

- Balance between being "smart and slow" and "dumb and fast"

Proof Outline: Probabilistic Completeness of PRM

Components

- Free configuration space Q_{free}: arbitrary open subset of $[0,1]^d$
- Local connector: connects $a, b \in Q_{\mathrm{free}}$ via a straight-line path and succeeds if path lies entirely in Q_{free}
- Collection of roadmap samples from Q_{free}

Components

- Free configuration space Q_{free}: arbitrary open subset of $[0, 1]^d$
- Local connector: connects $a, b \in Q_{\mathrm{free}}$ via a straight-line path and succeeds if path lies entirely in Q_{free}
- Collection of roadmap samples from Q_{free}

Let $a, b \in Q_{\mathrm{free}}$ such that there exists a path γ between a and b lying in Q_{free}. Then the probability that PRM correctly answers the query (a, b) after generating n collision-free configurations is given by

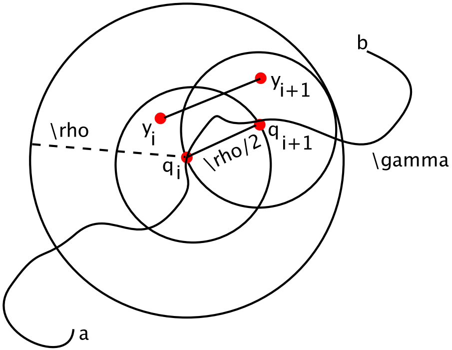
$$\mathrm{Pr}[(a, b)\mathrm{SUCCESS}] \geq 1 - \left\lceil \frac{2L}{\sigma} \right\rceil e^{\sigma \rho^d n},$$

where

- L is the length of the path γ
- $\rho = \mathrm{clr}(\gamma)$ is the clearance of path γ from obstacles
- $\sigma = \frac{\mu(B_1(\cdot))}{2^d \mu(Q_{\mathrm{free}})}$
- $\mu(B_1(\cdot))$ is the volume of the unit ball in $\mathbb{R}^d$
- $\mu(Q_{\mathrm{free}})$ is the volume of Q_{free}

Basic Idea

- Reduce path to a set of open balls in Q_{free}
- Calculate probability of generating samples in those balls
- Connect samples in different balls via straight-line paths to compute solution path

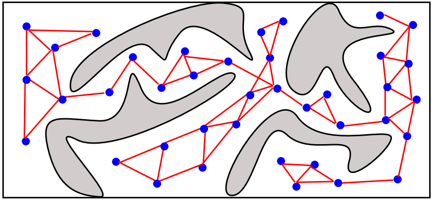


- Note that clearance $\rho = \mathrm{clr}(\gamma) > 0$
- Let $m = \left\lceil \frac{2L}{\rho} \right\rceil$. Then, γ can be covered with m balls $B_{\rho/2}(q_i)$ where $a = q_1, \ldots, q_m = b$
- Let $y_i \in B_{\rho/2}(q_i)$ and $y_{i+1} \in B_{\rho/2}(q_{i+1})$.
  Then, the straight-line segment $\overline{y_i y_{i+1}} \in Q_{\mathrm{free}}$, since $y_i, y_{i+1} \in B_\rho(q_i)$
- $I_i \stackrel{def}{=}$ indicator variable that there exists $y \in V$ s.t. $y \in B_{\rho/2}(q_i)$
- $\Pr[(a,b)\mathrm{FAILURE}] \leq \Pr\left[\bigvee_{i=1}^m I_i = 0\right] \leq \sum_{i=1}^m \Pr[I_i = 0]$
  - Note that $\Pr[I_i = 0] = \left(1 - \frac{\mu(B_{\rho/2}(q_i))}{\mu(Q_{\mathrm{free}})}\right)^n$
    i.e., probability that none of the n PRM samples falls in $B_{\rho/2}(q_i)$
  - I_i's are independent because of uniform samling in PRM

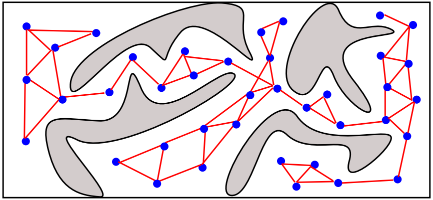
  Therefore, $\Pr[(a,b)\mathrm{FAILURE}] \leq m\left(1 - \frac{\mu(B_{\rho/2}(\cdot))}{\mu(Q_{\mathrm{free}})}\right)^n$
  - $\frac{\mu(B_{\rho/2}(\cdot))}{\mu(Q_{\mathrm{free}})} = \frac{\left(\frac{\rho}{2}\right)^d \mu(B_1(\cdot))}{\mu(Q_{\mathrm{free}})} = \sigma\rho^d$

  Therefore, $\Pr[(a,b)\mathrm{FAILURE}] \leq m\left(1 - \sigma\rho^d\right)^n \leq me^{-\sigma\rho^d} = \left\lceil \frac{2L}{\rho} \right\rceil e^{-\sigma\rho^d}$ □

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space

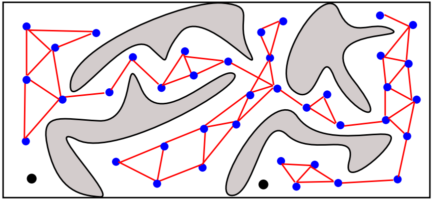


- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



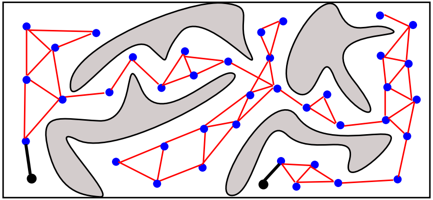
- Good when the objective is to solve *multiple* queries

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

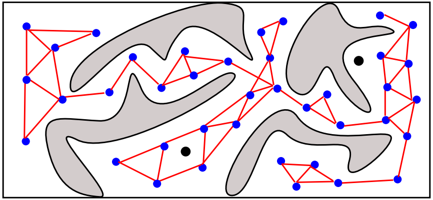
- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



- Good when the objective is to solve *multiple* queries

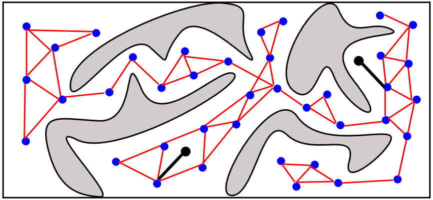
Motivation: Tree-based Motion Planning

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space



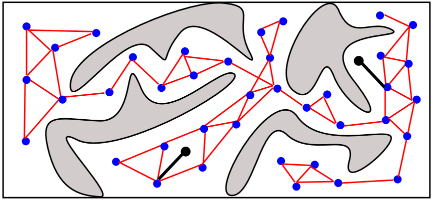
- Good when the objective is to solve *multiple* queries

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space

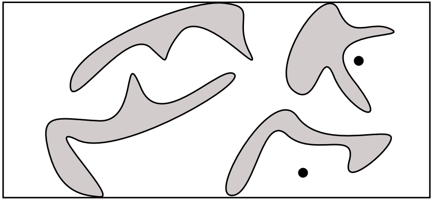


- Good when the objective is to solve *multiple* queries

- PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space

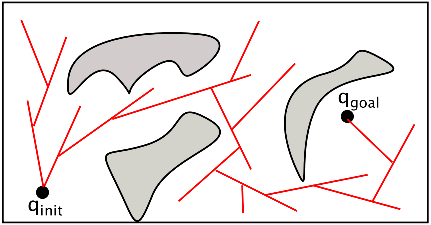


- Good when the objective is to solve *multiple* queries
- Maybe a bit too much when the objective is to solve a *single* query



General Idea

Grow a tree in the free configuration space from q_{init} *toward* q_{goal}



$\textsc{TreeSearchFramework}(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T} \leftarrow \textsc{RootTree}(q_{\text{init}})$

while q_{goal} has not been reached **do**

  $q \leftarrow \textsc{SelectConfigFromTree}(\mathcal{T})$

  $\textsc{AddTreeBranchFromConfig}(\mathcal{T}, q)$

Critical Issues

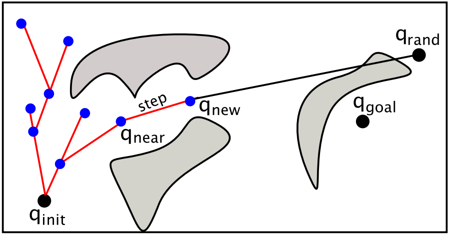
- How should a configuration be selected from the tree?
- How should a new branch be added to the tree from the selected configuration?

Rapidly-exploring Random Tree (RRT)

Pull the tree toward random samples in the configuration space

[LaValle, Kuffner: 1999]

- RRT relies on nearest neighbors and distance metric $\rho : Q \times Q \leftarrow \mathbb{R}^{\geq 0}$
- RRT adds Voronoi bias to tree growth



$\mathrm{RRT}(q_{\mathrm{init}}, q_{\mathrm{goal}})$

$\triangleright$ *initialize tree*

$\mathcal{T} \leftarrow$ create tree rooted at q_{init}

while solution not found **do**

$\triangleright$ *select configuration from tree*

$q_{\mathrm{rand}} \leftarrow$ generate a random sample

$q_{\mathrm{near}} \leftarrow$ nearest configuration in $\mathcal{T}$ to q_{rand} according to distance ρ

$\triangleright$ *add new branch to tree from selected configuration*

$\mathrm{path} \leftarrow$ generate path (not necessarily collision free) from q_{near} to q_{rand}

if $\textsc{IsSubpathCollisionFree}(\mathrm{path}, 0, \mathtt{step})$ **then**

$q_{\mathrm{new}} \leftarrow \mathrm{path}(\mathtt{step})$

add configuration q_{new} and edge $(q_{\mathrm{near}}, q_{\mathrm{new}})$ to $\mathcal{T}$

$\triangleright$ *check if a solution is found*

if $\rho(q_{\mathrm{new}}, q_{\mathrm{goal}}) \approx 0$ **then**

return solution path from root to q_{new}

Aspects for Improvement

Suggested Improvements in the Literature

Aspects for Improvement

- BasicRRT does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q
- In particular, tree growth is not directed towards q_{goal}

Suggested Improvements in the Literature

Rapidly-exploring Random Tree (RRT) (cont.)

Aspects for Improvement

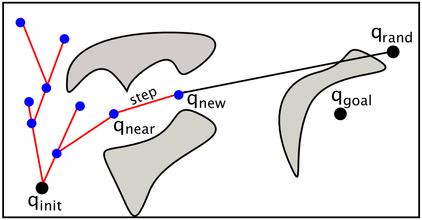
- BasicRRT does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q
- In particular, tree growth is not directed towards q_{goal}

Suggested Improvements in the Literature

- Introduce goal-bias to tree growth (known as GoalBiasRRT)
  - q_{rand} is selected as q_{goal} with probability p
  - q_{rand} is selected based on uniform sampling of Q with probability $1 - p$
  - Probability p is commonly set to ≈ 0.05

Aspects for Improvement

- BasicRRT takes only one small step when adding a new tree branch

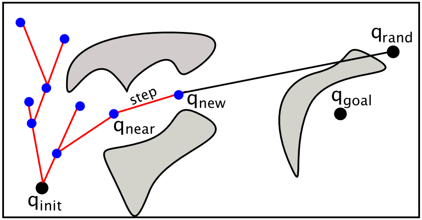


- This slows down tree growth

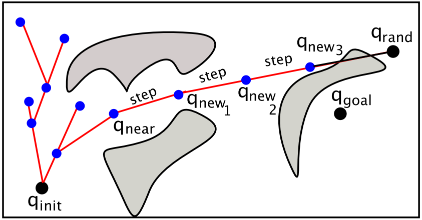
Suggested Improvements in the Literature

Aspects for Improvement

- BasicRRT takes only one small step when adding a new tree branch



- This slows down tree growth



Suggested Improvements in the Literature

- Take several steps until q_{rand} is reached or a collision is found (ConnectRRT)
- Add all the intermediate nodes to the tree

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

Expansive-Space Tree (EST)

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch

Expansive-Space Tree (EST)

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) = 1/(1+$ number of neighbors near $q)$
  - combination of different strategies

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) = 1/(1+$ number of neighbors near $q)$
  - combination of different strategies

SELECTCONFIGFROMTREE

- select q in $\mathcal{T}$ with probability $w(q)/\sum_{q' \in \mathcal{T}} w(q')$

Expansive-Space Tree (EST)

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) = 1/(1+$ number of neighbors near $q)$
  - combination of different strategies

$\textsc{SelectConfigFromTree}$

- select q in $\mathcal{T}$ with probability $w(q)/\sum_{q' \in \mathcal{T}} w(q')$

$\textsc{AddTreeBranchFromConfig}(\mathcal{T}, q)$

- $q_{\rm near} \leftarrow$ sample a collision-free configuration near q
- $\rm path \leftarrow$ generate path from q to $q_{\rm near}$
- if $\rm path$ is collision-free, then add $q_{\rm near}$ and $(q, q_{\rm near})$ to $\mathcal{T}$

Push the tree frontier in the free configuration space

[Hsu, Rock, Motwani, Latombe: 1999]

- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q) = 1/(1+$ number of neighbors near $q)$
  - combination of different strategies

SELECTCONFIGFROMTREE

- select q in $\mathcal{T}$ with probability $w(q)/\sum_{q' \in \mathcal{T}} w(q')$

ADDTREEBRANCHFROMCONFIG$(\mathcal{T}, q)$

- $q_{\text{near}} \leftarrow$ sample a collision-free configuration near q
- $\text{path} \leftarrow$ generate path from q to q_{near}
- if path is collision-free, then add q_{near} and (q, q_{near}) to $\mathcal{T}$

[play movie]

- Tree generally grows rapidly for the first few thousand iterations
- Tree growth afterwards slows down quite significantly
- Large number of configurations increases computational cost
- It becomes increasingly difficult to guide the tree towards previously unexplored parts of the free configuration space

Observations in High-Dimensional Problems

- Tree generally grows rapidly for the first few thousand iterations
- Tree growth afterwards slows down quite significantly
- Large number of configurations increases computational cost
- It becomes increasingly difficult to guide the tree towards previously unexplored parts of the free configuration space

Possible improvements?

Grow two trees, rooted at q_{init} and q_{goal}, towards each other

- Bi-directional trees improve computational efficiency compared to a single tree
- Growth slows down significantly later than when using a single tree
- Fewer configurations in each tree, which imposes less of a computational burden
- Each tree explores a different part of the configuration space

BiTree($q_{\mathrm{init}}, q_{\mathrm{goal}}$)

1: $\mathcal{T}_{\mathrm{init}} \leftarrow$ create tree rooted at q_{init}
2: $\mathcal{T}_{\mathrm{goal}} \leftarrow$ create tree rooted at q_{goal}
3: **while** solution not found **do**
4: add new branch to $\mathcal{T}_{\mathrm{init}}$
5: add new branch to $\mathcal{T}_{\mathrm{goal}}$
6: attempt to connect neighboring configurations from the two trees
7: if successful, return path from q_{init} to q_{goal}

Bi-directional Trees

Grow two trees, rooted at q_{init} and q_{goal}, towards each other

- Bi-directional trees improve computational efficiency compared to a single tree
- Growth slows down significantly later than when using a single tree
- Fewer configurations in each tree, which imposes less of a computational burden
- Each tree explores a different part of the configuration space

BiTree($q_{\mathrm{init}}, q_{\mathrm{goal}}$)

1: $\mathcal{T}_{\mathrm{init}} \leftarrow$ create tree rooted at q_{init}
2: $\mathcal{T}_{\mathrm{goal}} \leftarrow$ create tree rooted at q_{goal}
3: **while** solution not found **do**
4: add new branch to $\mathcal{T}_{\mathrm{init}}$
5: add new branch to $\mathcal{T}_{\mathrm{goal}}$
6: attempt to connect neighboring configurations from the two trees
7: if successful, return path from q_{init} to q_{goal}

- Different tree planners can be used to grow each of the trees
- E.g., RRT can be used for one tree and EST can be used for the other

High-dimensional Motion Planning

High-dimensional Motion Planning

- PRM provides *global* sampling of the configuration space

High-dimensional Motion Planning

- `PRM` provides *global* sampling of the configuration space
    - But, if sampling is sparse, then roadmap is disconnected
    - Moreover, dense sampling is impractical in high-dimensional spaces

High-dimensional Motion Planning

- `PRM` provides *global* sampling of the configuration space
  - But, if sampling is sparse, then roadmap is disconnected
  - Moreover, dense sampling is impractical in high-dimensional spaces
- Tree planner provides fast *local* exploration of area around root

High-dimensional Motion Planning

- `PRM` provides *global* sampling of the configuration space
  - But, if sampling is sparse, then roadmap is disconnected
  - Moreover, dense sampling is impractical in high-dimensional spaces
- Tree planner provides fast *local* exploration of area around root
  - But, tree growth slows down significantly in high-dimensional spaces
  - Although bi-directional trees offer some improvements, problems still remain

High-dimensional Motion Planning

- PRM provides *global* sampling of the configuration space
  - But, if sampling is sparse, then roadmap is disconnected
  - Moreover, dense sampling is impractical in high-dimensional spaces
- Tree planner provides fast *local* exploration of area around root
  - But, tree growth slows down significantly in high-dimensional spaces
  - Although bi-directional trees offer some improvements, problems still remain

Desired Properties for a Motion Planner

- Guides exploration towards goal
- Strikes right balance between breadth and depth of search

High-dimensional Motion Planning

- PRM provides *global* sampling of the configuration space
  - But, if sampling is sparse, then roadmap is disconnected
  - Moreover, dense sampling is impractical in high-dimensional spaces
- Tree planner provides fast *local* exploration of area around root
  - But, tree growth slows down significantly in high-dimensional spaces
  - Although bi-directional trees offer some improvements, problems still remain

Desired Properties for a Motion Planner

- Guides exploration towards goal
- Strikes right balance between breadth and depth of search

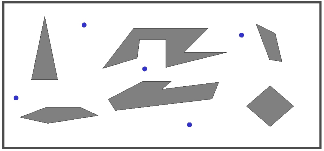
Sampling-based Roadmap of Trees (SRT)

[Plaku, Bekris, Chen, Ladd, Kavraki: Trans on Robotics 2005]

- Hierarchical planner
- Top level performs global sampling (PRM-based)
- Bottom level performs local sampling (tree-based, e.g., RRT, EST)
- Combines advantages of global and local sampling

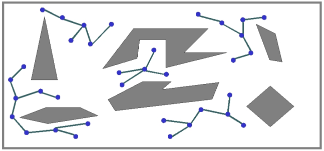
CreateTreesInRoadmap

```
V ← ∅; E ← ∅
while |V| < n_trees do
    𝒯 ← create tree rooted at a collision-free configuration
    use tree planner to grow 𝒯 for some time
    add 𝒯 to roadmap vertices V
```



CreateTreesInRoadmap

1: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$
2: **while** $|V| < n_{\text{trees}}$ **do**
3: $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
4: use tree planner to grow $\mathcal{T}$ for some time
5: add $\mathcal{T}$ to roadmap vertices V



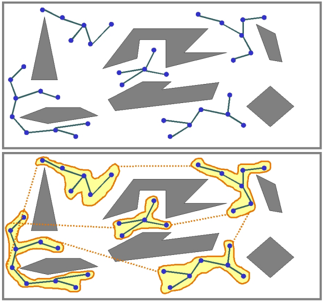
Sampling-based Roadmap of Trees (SRT) (cont.)

CREATETREESINROADMAP

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\mathrm{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SELECTWHICHTREESTOCONNECT

$E_{\mathrm{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\mathrm{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\mathrm{rand}} \leftarrow r$ random trees in V
  $E_{\mathrm{pairs}} \leftarrow E_{\mathrm{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\mathrm{neighs}} \cup S_{\mathrm{rand}}\}$



CreateTreesInRoadmap

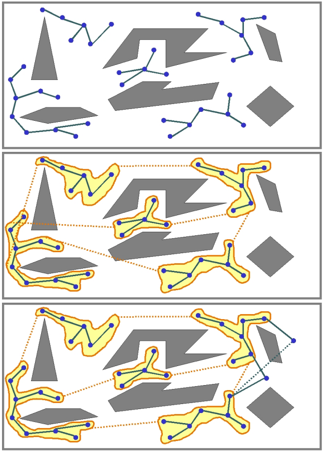
$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SelectWhichTreesToConnect

$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

ConnectTreesInRoadmap

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if AreTreesConnected$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap



CreateTreesInRoadmap

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SelectWhichTreesToConnect

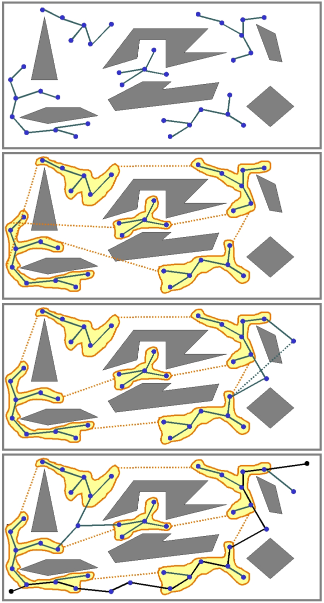
$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

ConnectTreesInRoadmap

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if AreTreesConnected$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap

SolveQuery$(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
$\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
connect $\mathcal{T}_{\text{init}}$ and $\mathcal{T}_{\text{goal}}$ to roadmap
search roadmap graph for solution



Sampling-based Roadmap of Trees (SRT) (cont.)

CreateTreesInRoadmap

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SelectWhichTreesToConnect

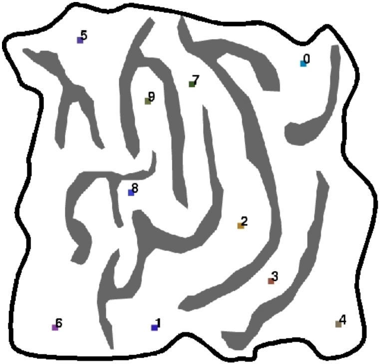
$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

ConnectTreesInRoadmap

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if AreTreesConnected$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap

SolveQuery$(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
$\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
connect $\mathcal{T}_{\text{init}}$ and $\mathcal{T}_{\text{goal}}$ to roadmap
search roadmap graph for solution



Sampling-based Roadmap of Trees (SRT) (cont.)

CREATETREESINROADMAP

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SELECTWHICHTREESTOCONNECT

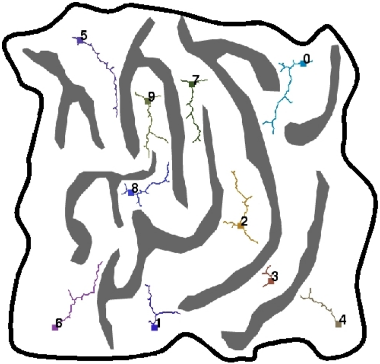
$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

CONNECTTREESINROADMAP

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if ARETREESCONNECTED$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap

SOLVEQUERY$(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
$\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
connect $\mathcal{T}_{\text{init}}$ and $\mathcal{T}_{\text{goal}}$ to roadmap
search roadmap graph for solution



Sampling-based Roadmap of Trees (SRT) (cont.)

CreateTreesInRoadmap

```
V ← ∅; E ← ∅
while |V| < n_trees do
    T ← create tree rooted at a collision-free configuration
    use tree planner to grow T for some time
    add T to roadmap vertices V
```

SelectWhichTreesToConnect

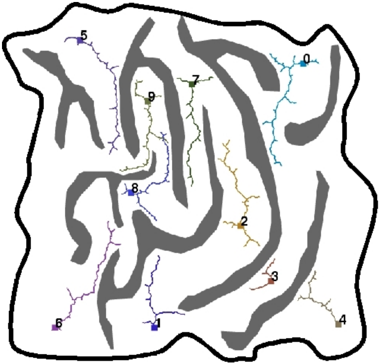
```
E_pairs ← ∅
for each T ∈ V do
    S_neighs ← k nearest trees in V to T
    S_rand ← r random trees in V
    E_pairs ← E_pairs ∪ {(T, T') : T' ∈ S_neighs ∪ S_rand}
```

ConnectTreesInRoadmap

```
for each (T1, T2) ∈ E_pairs do
    if AreTreesConnected(T1, T2) = false then
        run bi-directional tree planner to connect T1 to T2
        if connection successful then
            add edge (T1, T2) to roadmap
```

SolveQuery($q_{\mathrm{init}}, q_{\mathrm{goal}}$)

```
T_init ← create tree rooted at q_init
T_goal ← create tree rooted at q_goal
connect T_init and T_goal to roadmap
search roadmap graph for solution
```



Sampling-based Roadmap of Trees (SRT) (cont.)

CreateTreesInRoadmap

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SelectWhichTreesToConnect

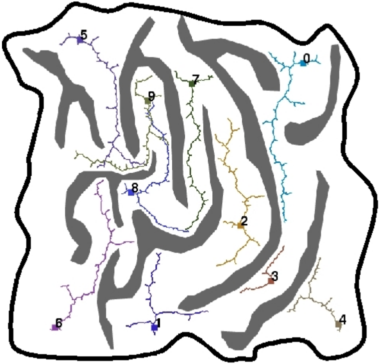
$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

ConnectTreesInRoadmap

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if AreTreesConnected$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap

SolveQuery$(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
$\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
connect $\mathcal{T}_{\text{init}}$ and $\mathcal{T}_{\text{goal}}$ to roadmap
search roadmap graph for solution



Sampling-based Roadmap of Trees (SRT) (cont.)

CreateTreesInRoadmap

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
while $|V| < n_{\text{trees}}$ **do**
  $\mathcal{T} \leftarrow$ create tree rooted at a collision-free configuration
  use tree planner to grow $\mathcal{T}$ for some time
  add $\mathcal{T}$ to roadmap vertices V

SelectWhichTreesToConnect

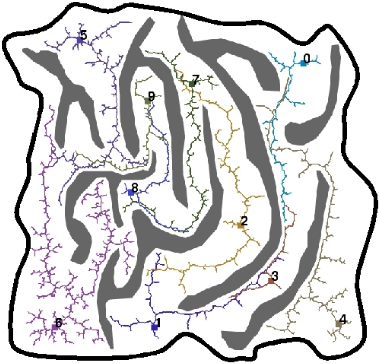
$E_{\text{pairs}} \leftarrow \emptyset$
for each $\mathcal{T} \in V$ **do**
  $S_{\text{neighs}} \leftarrow k$ nearest trees in V to $\mathcal{T}$
  $S_{\text{rand}} \leftarrow r$ random trees in V
  $E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{(\mathcal{T}, \mathcal{T}') : \mathcal{T}' \in S_{\text{neighs}} \cup S_{\text{rand}}\}$

ConnectTreesInRoadmap

for each $(\mathcal{T}_1, \mathcal{T}_2) \in E_{\text{pairs}}$ **do**
  if AreTreesConnected$(\mathcal{T}_1, \mathcal{T}_2) =$ `false` **then**
    run bi-directional tree planner to connect $\mathcal{T}_1$ to $\mathcal{T}_2$
    if connection successful **then**
      add edge $(\mathcal{T}_1, \mathcal{T}_2)$ to roadmap

SolveQuery$(q_{\text{init}}, q_{\text{goal}})$

$\mathcal{T}_{\text{init}} \leftarrow$ create tree rooted at q_{init}
$\mathcal{T}_{\text{goal}} \leftarrow$ create tree rooted at q_{goal}
connect $\mathcal{T}_{\text{init}}$ and $\mathcal{T}_{\text{goal}}$ to roadmap
search roadmap graph for solution



Advantages

- Explores small subset of possibilities by sampling
- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just a heuristic approach? No. It's more than that

It offers *probabilistic completeness*

- When a solution exists, a probabilistically complete planner finds a solution with probability as time goes to infinity.
- When a solution does not exists, a probabilistically complete planner may not be able to determine that a solution does not exist.