Algorithms for Sensor-Based Robotics: Sampling-Based Motion Planning

Computer Science 336
http://www.cs.jhu.edu/~hager/Teaching/cs336

Professor Hager
http://www.cs.jhu.edu/~hager
Recall Earlier Methods

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

Explicit Construction of Configuration Space/Roadmaps
- PSPACE-complete
- Exponential dependency on dimension
- No practical algorithms

[fig from Jyh-Ming Lien]
The Basic Idea

- Capture the connectivity of $Q_{\text{free}}$ by a graph or network of paths.
RoadMap Definition

• A roadmap, RM, is a set of trajectories (i.e. \( f( t, q_A, q_B ) \)) such that for all \( q_{\text{start}} \in Q_{\text{free}} \) and \( q_{\text{goal}} \in Q_{\text{free}} \) can be connected by a path:

• The three ingredients of a roadmap
  – **Accessibility**: There is a path from \( q_{\text{start}} \in Q_{\text{free}} \) to some \( q' \in \text{RM} \)
  – **Departability**: There is a path from some \( q'' \in \text{RM} \) to \( q_{\text{goal}} \in Q_{\text{free}} \)
  – **Connectivity**: there exists a path in \( \text{RM} \) between \( q' \) and \( q'' \)

16-735, Howie Choset, with significant copying from G.D. Hager who loosely based his notes on notes by Nancy Amato
RoadMap Path Planning

1. Build the roadmap
   a) nodes are points in $Q_{\text{free}}$ or its boundary
   b) two nodes are connected by an edge if there is a free path between them (i.e. $f(t, q_A, q_B)$)

2. Connect $q_{\text{start}}$ and $q_{\text{goal}}$ points to the road map at point $q'$ and $q''$, respectively

3. Find a path on the roadmap between $q'$ and $q''$. The result is a path in $Q_{\text{free}}$ from start to goal
Roadmap
Roadmap

\[ \text{[Image of roadmap diagram]} \]
A Hard Problem
The Way Forward

- The hard part of path planning is to somehow represent the free configuration space
  - Recall the basic PSPACE-hard results of Reif and Canny.
  - Recall the limitations of potential fields and even navigation functions.

- Probabilistic RoadMap Planning (PRM) by Kavraki, 1996
  - samples to find free configurations
  - connects the configurations (creates a graph)
  - is designed to be a multi-query planner

- Expansive-Spaces Tree planner (EST) and Rapidly-exploring Random Tree planner (RRT)
  - are appropriate for single query problems
Remember the Basic Problem

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

How would you solve it?
How about Rolling the Dice?

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

How would you solve it?

![Diagram of a robotic system with initial and goal points]

**Hint**: How would you approximate $\pi$?
Sample

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

![Diagram showing a robotic system with an initial point (init) and a goal point, with shaded obstacles]

- **Sample points**
Discard

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

Sample points
- Discard samples that are in collision
Connect

- **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 #2

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

![Diagram](image)

- **Sample points**
- Discard samples that are in collision
- Connect neighboring samples via straight-line segments
- **Discard straight-line segments that are in collision**
Finally, a Roadmap

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

![Roadmap Diagram]

- 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
The Complete Algorithm

0. Initialization
add $q_{init}$ and $q_{goal}$ to roadmap vertex set $V$

1. Sampling
repeat several times
   $q \leftarrow \text{SAMPLE}()$
   if $\text{ISCOLLISIONFREE}(q) = \text{true}$
      add $q$ to roadmap vertex set $V$

2. Connect Samples
for each pair of neighboring samples $(q_a, q_b) \in V \times V$
   path $\leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$
   if $\text{ISCOLLISIONFREE}$(path) = true
      add $(q_a, q_b)$ to roadmap edge set $E$

3. Graph Search
search graph $(V, E)$ for path from $q_{init}$ to $q_{goal}$

[Kavraki, Švestka, Latombe, Overmars 1996]
Why Are Probabilistic Methods Effective?

• Mainly, they do not try to construct the free configuration space
  – depend on local sampling of configurations (depends on efficient collision detection)
  – make use of relatively “dumb” planners to connect nodes

• They are often probabilistically complete
  – the probability of correct solution can be as high as desired with enough time
Some Minor Issues

• Ideally, the resulting graph will be connected
  – if it is not it might mean the space is disconnected
  – it might mean we didn’t try hard enough (what is a hard case?)

• We haven’t specified what the distance or path planner is

• To use for queries we must
  – connect the start and goal configurations to the roadmap (usually just treat like nodes and perform same algorithm)
  – perform a graph search on the resulting graph
  – if desired, smooth the resulting path a bit

• Interesting to note that the queries can be used to add more nodes to the graph
Sampling and Neighbors

• Uniform sampling of configurations
  – need to take care that rotations are “fairly” sampled

• Selecting closest neighbors: kd-tree
  – Given: a set S of n points in d-dimensional space
  – Recursively
    • choose a plane P that splits S about evenly (usually in a coordinate dimension)
    • store P at node
    • apply to children $S_l$ and $S_r$
  – Requires $O(dn)$ storage, built in $O(dn \log n)$ time
  – Query takes $O(n^{1-1/d} + m)$ time where $m$ is # of neighbors
    • asymptotically linear in $n$ and $m$ with large $d$

• Selecting closest neighbors: cell-based method
  – when each point is generated, hash to a cell location
Distance Functions

• Distance between two configurations should reflect the likelihood that the planner will fail to find a path
  – close points, likely to succeed
  – far away, less likely

• Ideally, this is probably related to the area swept out by the robot
  – very hard to compute exactly
  – usually heuristic distance is used

• Typical approaches
  – Euclidean distance on some embedding of c-space
  – Alternative is to create a weighted combination of translation and rotational “distances”
  – Efficiency varies greatly depending on embedding
Local Planner

- The local planner should be reasonably fast and can be simple to implement

- A simple way is to do subdivision on straight line path
  - Decompose motion into a straight lines in configuration space
  - Split the line in half; check for collision
  - If none, recurse on halves until distance is small
PRM Applied to a Point Robot

$q = (x, y) \leftarrow \text{SAMPLE}()$
- $x \leftarrow \text{RAND}(\min_x, \max_x)$
- $y \leftarrow \text{RAND}(\min_y, \max_y)$

\text{ISSAMPLECOLLISIONFREE}(q)
- Point inside/outside polygon test

\text{path} \leftarrow \text{GENERATELOCALPATH}(q_a, q_b)$
- Straight-line segment from point $q_a$ to point $q_b$

\text{ISPATHCOLLISIONFREE}(\text{path})
- Segment-polygon intersection test
PRM for a Serial Chain

\[ q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow \text{SAMPLE}() \]
\[ \theta_i \leftarrow \text{RAND}(-\pi, \pi), \forall i \in [1, n] \]

\text{IsSampleCollisionFree}(q)
- Place chain in configuration \( q \) (forward kinematics)
- Check for collision with obstacles

\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
  \[ \text{path}(t) = (1 - t) \ast q_a + t \ast q_b \]

\text{IsPathCollisionFree}(\text{path})
- Incremental approach
- Subdivision approach
Sampling Strategies

• The narrow corridor problem
  – probability of finding a path related to joint visibility area under uniform sampling

• Other approaches:
  – Bridge planner
    • sample randomly
    • check pairs in collision to see if midpoint (or random distance) is not
    • Somehow use generalized Voronoi diagrams
  – Visibility-based sampling
    • only keep configuration that
      – cannot be connected to an existing component, or
      – connect 2 existing components
Sampling Strategies

• Recall the narrow corridor problem
  – probability of finding a path related to joint visibility area under uniform sampling
  – Few samples will be available in here

• Other approaches:
  – sampling near obstacles
    • Obstacle-Based PRM (OBPRM)
    • find samples in obstacles (uniform)
    • pick a random direction $v$
    • search for a free configuration in direction $v$
  – Gaussian OBPRM
    • generate a random sample
    • generate a Gaussian sample around the sample
    • only keep the sample if the Gaussian sample is in collision
  – Dilated obstacles
    • allow some interpenetration to enhance the likelihood of finding paths, then “fix up” later.
Obstacle Sampling

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

\[
\text{GENERATECOLLISIONFREECONFIG}
\]
\[
\begin{align*}
1: & \quad q_a \leftarrow \text{generate config uniformly at random} \\
2: & \quad \text{if ISCONFIGCOLLISIONFREE}(q_a) = \text{true} \text{ then} \\
3: & \quad \quad \text{return } q_a \\
4: & \quad \text{else} \\
5: & \quad \quad q_b \leftarrow \text{generate config uniformly at random} \\
6: & \quad \quad \text{path} \leftarrow \text{GENERATEPATH}(q_a, q_b) \\
7: & \quad \quad \text{for } t = \delta \text{ to } |\text{path}| \text{ by } \delta \text{ do} \\
8: & \quad \quad \quad \text{if ISCONFIGCOLLISIONFREE}(\text{path}(t)) \text{ then} \\
9: & \quad \quad \quad \quad \text{return } \text{path}(t) \\
10: & \quad \quad \text{return null}
\end{align*}
\]

[Amato, Bayazit, Dale, Jones, Vallejo: WAFR 1998]
Objective: Increase Sampling Inside/Near Narrow Passages
Approach: Sample from a Gaussian distribution biased near the obstacles

GENERATE_COLLISION_FREE_CONFIG
1: \( q_a \leftarrow \) generate config uniformly at random
2: \( r \leftarrow \) generate distance from Gaussian distribution
3: \( q_b \leftarrow \) generate config uniformly at random at distance \( r \) from \( q_a \)
4: \( \text{ok}_a \leftarrow \text{IsConfigCollisionFree}(q_a) \)
5: \( \text{ok}_b \leftarrow \text{IsConfigCollisionFree}(q_b) \)
6: \textbf{if} \( \text{ok}_a = \text{true} \) \textbf{and} \( \text{ok}_b = \text{false} \) \textbf{then return} \( q_a \)
7: \textbf{if} \( \text{ok}_a = \text{false} \) \textbf{and} \( \text{ok}_b = \text{true} \) \textbf{then return} \( q_b \)
8: \textbf{return} null
Bridge Sampling

Objective: Increase Sampling Inside/Near Narrow Passages

Approach: Create “bridge” between samples in collision

GENERATE_COLLISION_FREE_CONFIG
1: \( q_a \leftarrow \) generate config uniformly at random
2: \( q_b \leftarrow \) generate config uniformly at random
3: \( \text{ok}_a \leftarrow \text{IS_CONFIG_COLLISION_FREE}(q_a) \)
4: \( \text{ok}_b \leftarrow \text{IS_CONFIG_COLLISION_FREE}(q_b) \)
5: if \( \text{ok}_a = \text{false and ok}_b = \text{false} \) then
6: \( \text{path} \leftarrow \text{GENERATE_PATH}(q_a, q_b) \)
7: \( q \leftarrow \text{path}(0.5|\text{path}|) \)
8: if \( \text{IS_CONFIG_COLLISION_FREE}(q) \) then
9: \( \text{return } q \)
10: \( \text{return } \text{null} \)

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

10/8/2019
CS 336, G.D. Hager (w/slides from S. LaValle, E Plaku)
Visibility Sampling

**Objective:** Capture connectivity of configuration space with few samples

**Approach:** Generate samples that create new components or join existing components

```plaintext
GENERATE_COLLISION_FREE_CONFIG
1: q ← generate config uniformly at random
2: if IS_CONFIG_COLLISION_FREE(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

- Associate weight \( w(q) \) with each configuration \( q \) in the roadmap
- Weight \( w(q) \) indicates difficulty of region around \( q \)
  - \( w(q) = \frac{1}{1 + \text{deg}(q)} \)
  - \( w(q) \) = 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
Avoiding Cycles

- 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 ← E ∪ {(q_a, q_b)}
```

- Disjoint-set data structure is used to speed up computation of 
  SAMEROADMAPCOMPONENT(q_a, q_b)
What about Single 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
Single Query Planners

• Use ESTs (expansive space trees) and RRTs (rapidly exploring random trees.
  – also work for nonholonomic and kinodynamic planning
  – each of these is initial condition dependent (requires integration) and thus it is hard to build a graph offline

• Generally build two trees from start and goal configurations
  – each tree grows toward the other
  – once linkage is made defines a path from start to goal

• Key is to define sampling strategies that focus on
  – unexplored areas of the free space
  – move toward start or goal

• For completeness, necessary to show that the algorithm will eventually cover the entire space
Tree-Based Motion Planning

Grow a tree in the free configuration space from \( q_{\text{init}} \) toward \( q_{\text{goal}} \)

\[
\text{TreeSearchFramework}(q_{\text{init}}, q_{\text{goal}})
\]

1: \( \mathcal{T} \leftarrow \text{RootTree}(q_{\text{init}}) \)
2: while \( q_{\text{goal}} \) has not been reached do
3: \( q \leftarrow \text{SelectConfigFromTree}(\mathcal{T}) \)
4: \( \text{AddTreeBranchFromConfig}(\mathcal{T}, q) \)

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?
Expansive Space Trees (EST)

Push the tree frontier in the free configuration space

- 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
  - $w(q) = \frac{1}{1 + \text{deg}(q)}$
  - $w(q) = 1/(1 + \text{number of neighbors near } q)$
  - combination of different strategies

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

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

[LaValle, Motwani, Latombe: 1999]
BiDirectional Motion Planning

Grow two trees, rooted at $q_{\text{init}}$ and $q_{\text{goal}}$, 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

\text{BiTREE}(q_{\text{init}}, q_{\text{goal}})

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

- 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
Rapidly-exploring Random Trees

Pull the tree toward random samples in the configuration space

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

\[
\text{RRT}(q_{\text{init}}, q_{\text{goal}})
\]

▷ initialize tree
1: $T \leftarrow$ create tree rooted at $q_{\text{init}}$
2: while solution not found do

▷ select configuration from tree
3: $q_{\text{rand}} \leftarrow$ generate a random sample
4: $q_{\text{near}} \leftarrow$ nearest configuration in $T$ to $q_{\text{rand}}$ according to distance $\rho$

▷ add new branch to tree from selected configuration
5: path $\leftarrow$ generate path (not necessarily collision free) from $q_{\text{near}}$ to $q_{\text{rand}}$
6: if IsSubpathCollisionFree(path, 0, step) then
7: $q_{\text{new}} \leftarrow$ path(step)
8: add configuration $q_{\text{new}}$ and edge $(q_{\text{near}}, q_{\text{new}})$ to $T$

▷ check if a solution is found
9: if $\rho(q_{\text{new}}, q_{\text{goal}}) \approx 0$ then
10: return solution path from root to $q_{\text{new}}$
Improvements

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 \( q_{\text{rand}} \) is reached or a collision is found (**CONNECTRRT**)
- Add all the intermediate nodes to the tree
Improvements

Aspects for Improvement

- **BasicRRT** does not take advantage of $q_{goal}$
- Tree is pulled towards random directions based on the uniform sampling of
- 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 $\approx 0.05$
An Important Point

• Note that ESTs (also RRTs with modifications) can model non-holonomic and complex systems.

Sampling:
1) Choose configuration variables (e.g. steering and acceleration) at random.
2) Integrate kinematics forward to get a path segment
3) Check the path for collision
CREATE TREES IN ROADMAP
1: \( V \leftarrow \emptyset; E \leftarrow \emptyset \)
2: \textbf{while} \(|V| < n_{\text{trees}}\) \textbf{do}
3: \( T \leftarrow \text{create tree rooted at a collision-free configuration} \)
4: \( \text{use tree planner to grow } T \text{ for some time} \)
5: \( \text{add } T \text{ to roadmap vertices } V \)

SELECT WHICH TREES TO CONNECT
1: \( E_{\text{pairs}} \leftarrow \emptyset \)
2: \textbf{for each } \( T \in V \) \textbf{do}
3: \( S_{\text{neighs}} \leftarrow k \text{ nearest trees in } V \text{ to } T \)
4: \( S_{\text{rand}} \leftarrow r \text{ random trees in } V \)
5: \( E_{\text{pairs}} \leftarrow E_{\text{pairs}} \cup \{ (T, T') : T' \in S_{\text{neighs}} \cup S_{\text{rand}} \} \)

CONNECT TREES IN ROADMAP
1: \textbf{for each } \( (T_1, T_2) \in E_{\text{pairs}} \) \textbf{do}
2: \textbf{if} \ ARE TREES CONNECTED \( (T_1, T_2) = \text{false} \) \textbf{then}
3: \( \text{run bi-directional tree planner to connect } T_1 \text{ to } T_2 \)
4: \textbf{if} connection successful \textbf{then}
5: \( \text{add edge } (T_1, T_2) \text{ to roadmap} \)

SOLVE QUERY \((q_{\text{init}}, q_{\text{goal}})\)
1: \( T_{\text{init}} \leftarrow \text{create tree rooted at } q_{\text{init}} \)
2: \( T_{\text{goal}} \leftarrow \text{create tree rooted at } q_{\text{goal}} \)
3: \( \text{connect } T_{\text{init}} \text{ and } T_{\text{goal}} \text{ to roadmap} \)
4: \( \text{search roadmap graph for solution} \)
Analysis of PRM

• Goal: show **probabilistic completeness**: 
  – Suppose that \( a, b \in Q_{\text{free}} \) can be connected by a free path. PRM is **probabilistically complete** if, for any \((a,b)\)

\[
\lim_{n \to \infty} \Pr[(a,b)\text{FAILURE}] = 0
\]

where \( n \) is the number of samples used to construct the roadmap

• Basic idea:
  – reduce the path to a set of open balls in free space
  – figure out how many samples it will take to generate a pair of points in those balls
  – connect those points to create a path

• Assuming that a path between \( a \) and \( b \) exists, the probability of that PRM will fail depends on
  1. The length of the path
  2. The distance of the path to the obstacle
  3. The number of configuration in the roadmap
Analysis of PRM

• A path from a to b can be described by a function $\gamma : [0,1] \rightarrow Q_{\text{free}}$ with $\gamma(0) = a$ and $\gamma(1) = b$

• Let $\text{clr}(\gamma)$ be the minimum distance between $\gamma$ and any obstacle.

• Let $\mu$ be a volume measure on the space, for set $S \in \mathbb{R}^d, \mu(S)$ is the volume of set $S$

• $B_\rho(x)$ is a ball centered at $x$ of radius $\rho$

• For uniform sampling of $A \subset Q_{\text{free}}$ the probability that a random point $x \in Q_{\text{free}}$, then $\Pr(x \in A) = \mu(A)/\mu(Q_{\text{free}})$
A Simple Idea

Basic Idea

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

Components

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

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

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

where

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

- Note that clearance $\rho = \text{clr}(\gamma) > 0$

- Let $m = \left\lceil \frac{2L}{\rho} \right\rceil$. Then, $\gamma$ 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 $\overline{y_i y_{i+1}} \in Q_{\text{free}}$, since $y_i, y_{i+1} \in B_{\rho}(q_i)$

- $l_i \overset{\text{def}}{=} \text{indicator variable that there exists } y \in V \text{ s.t. } y \in B_{\rho/2}(q_i)$

- $\Pr[(a, b)\text{FAILURE}] \leq \Pr \left[ \bigvee_{i=1}^{m} l_i = 0 \right] \leq \sum_{i=1}^{m} \Pr[l_i = 0]$

  - Note that $\Pr[l_i = 0] = \left( 1 - \frac{\mu(B_{\rho/2}(q_i))}{\mu(Q_{\text{free}})} \right)^n$

  - i.e., probability that none of the $n$ PRM samples falls in $B_{\rho/2}(q_i)$

- $l_i$'s are independent because of uniform sampling in PRM

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

- $\frac{\mu(B_{\rho/2}(\cdot))}{\mu(Q_{\text{free}})} = \left( \frac{\rho}{2} \right)^d \frac{\mu(B_1(\cdot))}{\mu(Q_{\text{free}})} = \sigma \rho^d$

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

• Relationship between “difficulty” of space and number of samples (avoids dependence on length of path which is unknown)

• Relies on so-called \((\varepsilon, \alpha, \beta)\) expansiveness of space
  – \(\varepsilon\) measures the fraction of space reachable from any point
  • called \(\varepsilon\)-good if all points “see” at least \(\varepsilon\) of the space
  • \(\forall x \in Q_{\text{free}} \quad \mu(\text{reach}(x)) \geq \varepsilon \mu(Q_{\text{free}})\)
  – lookout_{\beta} of a set \(S\) is the set of points that can see at least \(\beta\) of \(Q_{\text{free}} - S\)

• A space is \((\varepsilon \alpha \beta)\) expansive if
  – it is \(\varepsilon\)-good
  – for any connected \(S \subseteq Q_{\text{free}} \quad \mu(\text{lookout}_{\beta}(S)) \geq \alpha \mu(S)\)
  – intuitively, this capture how easy it is to choose points that link up the space
Expansive Spaces

Reachability
Expansive Spaces

Only a small subset of A can see a large fraction of points in B
Given $n$ random samples, what are the odds the resulting graph is connected?
Summary

- **PRS** --- a basic sampling-based planner
  - useful for multiple queries
  - can be made efficient using various sampling and evaluation tricks

- **EST and RRT**
  - single query planners
  - generate trees and merge them
  - again many variations based on applications

- **Analysis**
  - general idea of completeness
  - particular proof based on sampling
  - other ideas based on structure of environment

- **Other Applications**