16-843 – Manipulation Algorithms
Motion Planning

Motion Planning II
Motion Planning

Let $W = \mathbb{R}^m$ being the work space, $O \subset W$ the set of obstacles, $A(q)$ the robot in configuration $q \in C$

$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\}$

$C_{free} = C \setminus C_{obs}$

**Motion planning:** find a continuous path

$\tau : [0,1] \to C_{free}$

With $\tau(0) = q_I$, $\tau(1) = q_G$

Planning with robot being point in C-Space
Motion Planning

Overview of all motion planning we will handle:

• Combinatorial Planning
• Sampling based Planning
• Potential Functions
• Jacobian Transpose
• Decision Theory

Katharina Muelling (NREC, Carnegie Mellon University)
# Summary

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Visibility Graphs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Generalized Voronoi Diagrams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Exact Cell Decomposition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Approximate Cell Decomposition</td>
<td></td>
</tr>
<tr>
<td>Concept</td>
<td>• Explicit representation of $C_{obs}$</td>
<td>• Sample in C-Space</td>
</tr>
<tr>
<td></td>
<td>• Captures connectivity of $C_{free}$ in graph and finds path with graph search</td>
<td>• Use collision detection to probe for collisions with obstacles</td>
</tr>
<tr>
<td>Pros and Cons</td>
<td><strong>Pros</strong></td>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td></td>
<td>complete</td>
<td>Becomes intractable with increasing C-Space and object/robot complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>Complete</td>
<td>Probabilistic Complete</td>
</tr>
</tbody>
</table>

*Pros and Cons are highlighted for clarity.*
Visibility graph

A graph defined by:

- Nodes: $q_I, q_G$, obstacle vertex
- Edges: either obstacle edge or edge between nodes that does not intersect with any obstacle

Runtime: $O(n^2 \log n)$
Generalized Voronoi Diagram

**Idea:** Set of points that have maximal clearance from all nearest objects

Figure: Howie Choset
Cell Decomposition

Idea:
Decompose $C_{\text{free}}$ into regions (cells) and create connectivity graph that represents adjacencies

Two types of approach:
• **Exact Cell Decomposition**: Union of cells corresponds to $C_{\text{free}}$
• **Approximate Cell Decomposition**: Decomposes $C_{\text{free}}$ into cells with pre-defined shape; union of cells typically only approximates $C_{\text{free}}$ space

Katharina Muelling (NREC, Carnegie Mellon University)
Motion Planning

Overview of all motion planning we will handle:

• Combinatorial Planning ✔
• Sampling based Planning
• Potential Functions
• Jacobian Transpose
• Decision Theory
Sampling Based Planner

Combinatorial planning: Explicit modeling of $C_{free}$ and $C_{obs}$ to expensive

Sampling based planning: Makes no assumption about $C_{free}$

Idea:
• Take (random) samples from $C$, declare them as vertices in $C_{free}$
• Probe if configuration lies in $C_{free}$ using collision checking as a black box
• Try to connect nearby vertices with local planner

The collision checker as a black box
# Sampling Based Planner

## What have we learned so far?

### RRT

<table>
<thead>
<tr>
<th>Start with empty tree</th>
<th>Initialization: Initialize graph $G=(E,V)$ as undirected search graph with either empty set of edges and vertexes or with $E={q_I,q_G}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomly sample $q \in C_{free}$</td>
<td><strong>Vertex Selection:</strong> Sample a vertex $q_{new}$ in $C_{free}$ for expansion of $G$.</td>
</tr>
<tr>
<td>Connect to nearest neighbor</td>
<td><strong>Local Planning:</strong> Attempt to construct path between $q_{new}$ and existing vertexes in $V$. If no path can be found, go back to sampling.</td>
</tr>
<tr>
<td>Check if path found</td>
<td><strong>Check if done:</strong> If $q_{new}$ can be connected to existing vertexes in $V$, add edge to $G$ and check for termination criterion. If termination criterion true you are done, otherwise go back to sampling.</td>
</tr>
</tbody>
</table>
Sampling Based Planning

**Multi-Query**

Two phases:
1) Build Roadmap
2) Search

Assumes fixed obstacle setting over multiple queries.

**Probabilistic Road Maps**
[Kavaki et al.: Probabilistic roadmaps for path planning in high dimensional configuration spaces; 1996]

**Single-Query**

Iteratively build tree with search.

**Rapidly exploring Random Trees**
[Kuffner and LaValle; Rapidly exploring Random Trees; 2000]
Probabilistic Road Maps

Idea:
• Build a roadmap from sampled configuration space points
• Check if in $C_{\text{free}}$ by using a collision checker, and
• search the roadmap to find a path.

Not optimal, but feasible path

1. Phase – Learning Phase:
   • Construction Step: obtain reasonable connected graph with good coverage
   • Expansion Step: improve connectivity of difficult regions in C-Space

2. Phase – Query Phase:
   • Find path from $q_i$ to $q_G$

Probabilistic Road Maps

Construction Step:

Generate a uniformly distributed robot configuration $q_{\text{new}}$
Probabilistic Road Maps

Construction Step:

\[ G=(V,E) \quad V \leftarrow \emptyset, \quad E \leftarrow \emptyset \]

\[ q_{new} \leftarrow \text{random sample in } C \]

\[ \text{if } q_{new} \in C_{\text{free}} \]

Use collision checker to test if \( q_{new} \) is in \( C_{\text{free}} \)
or in \( C_{\text{obs}} \)
Probabilistic Road Maps

Construction Step:

\[ \begin{aligned}
G &= (V, E) \quad V \leftarrow \emptyset, E \leftarrow \emptyset \\
q_{\text{new}} &\leftarrow \text{random sample in } C \\
\text{if } q_{\text{new}} &\in C_{\text{free}} \\
\text{add } q_{\text{new}} &\text{ to } V \\
\end{aligned} \]

If \( q_{\text{new}} \in C_{\text{free}} \) add \( q_{\text{new}} \) to \( G \); 
If not, sample new
Probabilistic Road Maps

Construction Step:

\[ G = (V, E) \]

\[ V \leftarrow \emptyset, \ E \leftarrow \emptyset \]

\[ q_{new} \leftarrow \text{random sample in } C \]

if \( q_{new} \in C_{\text{free}} \)

add \( q_{new} \) to \( V \)

If \( q_{new} \in C_{\text{free}} \) add \( q_{new} \) to \( G \);
If not, sample new
Probabilistic Road Maps

Construction Step:

Find the set of all neighbors of $q_{\text{new}}$

$G=(V,E) \quad V \leftarrow \emptyset, \quad E \leftarrow \emptyset$

$q_{\text{new}} \leftarrow \text{random sample in } C$

if $q_{\text{new}} \in C_{\text{free}}$

add $q_{\text{new}}$ to $V$

for each $q \in \text{Neighbor}(G, q_{\text{new}})$
Probabilistic Road Maps

Construction Step:

Find the set of all neighbors of $q_{\text{new}}$ and connect each neighbor for which a path in $C_{\text{free}}$ exist.

$$G=(V,E) \quad V \leftarrow \emptyset, \quad E \leftarrow \emptyset$$

$q_{\text{new}} \leftarrow$ random sample in $C$

if $q_{\text{new}} \in C_{\text{free}}$

add $q_{\text{new}}$ to $V$

for each $q \in \text{Neighbor}(G,q_{\text{new}})$

If $\text{link}(q,q_{\text{new}})$

add edge $(q_{\text{new}},q)$ to $E$

Configuration space $C = \mathbb{R}^n$
Probabilistic Road Maps

Construction Step:

\[ G = (V, E) \]

\[ V \leftarrow \emptyset, \ E \leftarrow \emptyset \]

\[ \text{while } |E| < N \]

\[ q_{\text{new}} \leftarrow \text{random sample in } C \]

\[ \text{if } q_{\text{new}} \in C_{\text{free}} \]

\[ \text{add } q_{\text{new}} \text{ to } V \]

\[ \text{for each } q \in \text{Neighbor}(G, q_{\text{new}}) \]

\[ \text{if link}(q, q_{\text{new}}) \]

\[ \text{add edge } (q_{\text{new}}, q) \text{ to } E \]

Repeat

Milestones
Probabilistic Road Maps

Construction Step:

**Initialization:**

\[ G = (V, E), \quad V = \{\}, \quad E = \{\} \]

**Algorithm:**

while \( |E| < N \)

\( q_{\text{new}} \leftarrow \) random sample in \( C \)

\[ \text{if } q_{\text{new}} \in C_{\text{free}} \]

\( G.\text{addVertex}(q_{\text{new}}) \)

for each \( q \in \text{Neighbor}(G, q_{\text{new}}) \)

\[ \text{if link}(q,q_{\text{new}}) \]

\( G.\text{addEdge}(q_{\text{new}}, q) \)

What are the most difficult and time consuming steps?
Initialization:
\[ G = (V,E), \ V = \emptyset, \ E = \emptyset \]

Algorithm:
while \( |E| < N \)
\[ q_{\text{new}} \leftarrow \text{random sample in } C \]
\[ \text{if } q_{\text{new}} \in C_{\text{free}} \]
\[ G.\text{addVertex}(q_{\text{new}}) \]
for each \( q \in \text{Neighbor}(G, q_{\text{new}}) \)
\[ \text{If local_plan}(q, q_{\text{new}}) \]
\[ G.\text{addEdge}(q_{\text{new}}, q) \]
Collision Checking

How do we check if a given configuration $q$ is in collision with the objects in the environment?

**In the past**
Computed $C_{obs}$ a priori

- Combinatorial motion planning in $C_{free}$

**Now**
Use collision checker

- Sampling based motion planning in $C$

Collision Detection
True, false
$q$
Collision Checking

Object representations

Our 2D class world

Robot and obstacles represented by polygons
Collision Checking

Object representations

The 3D world of manipulation

**Composite of primitives**
 Mostly in simulation
 Faster collision checking due to simple structure, but usual overestimates true geometry

**Triangle meshes**
 Often used for real world applications
 Brute force: all-pair checking

**Voxel Grids**
 Easy to check intersections, but hard to rotate one grid relative to each other

From slides D. Berenson

wiki.thesimsresource.com

maltaannon.com

cse.iitkgp.ac.in

static1.1.sqspcdn.com
Collision Checking

Basic concept: Two phase collision detection

Broad phase:

• Avoid expensive computation by using simple bounding boxes around all objects
• Find pairs of potential collision candidates

Narrow phase:

• Individual pairs of potential collision bodies are checked carefully
Collision Checking

Approximation through Bounding Volumes

Axis Aligned Bounding Box (AABB)
Oriented Bounding Box (OBB)
Convex Hull

Tightness in approximation
Test expensive
Collision Checking

Hierarchical Methods

Bounding Volumes can be hierarchical organized by decomposing the object into a tree

**Construction:**
Split bounding box of one vertex into two smaller regions s.t.:
- Union covers same part of obstacles
- Both children are of similar size

AABB-Tree  OBB-Tree  AABB-Tree

[Image of AABB-Tree, OBB-Tree, and AABB-Tree]

Katharina Muelling (NREC, Carnegie Mellon University)
Collision Checking

Hierarchical Methods

Bounding Volumes can be hierarchical organized by decomposing the object into a tree

Collision checking:
Compare to object trees: start at root and check if bounding boxes intersect
  - No: no collision
  - Yes: Move on to children until reaching leaves

AABB-Tree
OBB-Tree
AABB-Tree
Collision Checking

How do I check collisions for edges?

So far: checked collision of individual configurations by creating the physical model of robot in specific configuration

Missing: How do we check an edge between two configurations?

In theory: Methods available to verify that path is collision free, but they are not easy to implement. See LaValle: Planning Algorithms, Chapter 5.3.4

In practice: Sample in interval [0,1] of path and call collision checker on individual configurations.
Collision Checking

**Summary**

Most time of planner for real-world tasks are still taken by collision checking

Trade off between accuracy, speed and memory usage

**Concepts to remember:**

- Narrow/Broad Phase collision checking
- Bounding Volumes: AABB, OBB
- Hierarchical Methods

Not covered in class: Swept Volumes
Collision Checking

Do I have to implement the collision checker myself?

No, there exist a couple of good libraries that help you with the collision checking, e.g.,

FCL

<table>
<thead>
<tr>
<th></th>
<th>Rigid Objects</th>
<th>Point Cloud</th>
<th>Deformable Objects</th>
<th>Articulated Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Continuous Collision Detection</td>
<td>✓</td>
<td>✓*</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Self Collision Detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Penetration Estimation</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Distance Computation</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Broad-phase Collision</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Pan et al.: FCL: A General Purpose Library for Collision and Proximity Queries

Other libraries: Bullet, Solid, Rapid, I-COLLIDE, ...
Probabilistic Road Maps

Construction Phase:

**Initialization:**
\[ G=(V,E), \ V=\{\}, \ E=\{\} \]

**Algorithm:**
while \(|E| < N\)
  \[ q_{new} \leftarrow \text{random sample in } C \]
  \[
  \begin{align*}
  &\text{if } q_{new} \in C_{\text{free}} \\
  &\quad \text{G.addVertex}(q_{new}) \\
  &\quad \text{for each } q \in \text{Neighbor}(G, q_{new}) \\
  &\quad\quad \text{if link}(q, q_{new}) \\
  &\quad\quad\quad \text{G.addEdge}(q_{new}, q)
  \end{align*}
  \]
Nearest Neighbors Computation

How do we find the nearest neighbor to connect the new sample?

1. What does nearest neighbor mean?
   - Need to define a distance metric $d(q_1, q_2)$ s.t.,
     - For small $d(q_1, q_2)$: likely to connect
     - For larger $d(q_1, q_2)$: less likely to connect
   - Common metric: $L_2$ metric (Euclidian)
     \[
     d(q_1, q_2) = \|q_1 - q_2\|
     \]

2. Different ways to define nearest neighbor
   - Find all nearest neighbors within a certain radius
   - Find $k$ nearest neighbors
   - All visible

Katharina Muelling (NREC, Carnegie Mellon University)
Nearest Neighbors Computation

How do we implement it?

Exact solutions:
• Check distance to all nodes edge segments in graph
• Can be very slow for high number of samples (usually the case)

Approximations:
• Insert vertices into an efficient data structure for nearest neighbor search
• In practice most widely used: Kd-trees
• High cost in creating kd-tree
• If there are more than $2^n$ nodes, kd-tree should be more efficient than naïve search
Probabilistic Road Maps

Construction Phase:

**Initialization:**
\[ G=(V,E), V=\{\}, E=\{\} \]

**Algorithm:**
while \( |E| < N \)

- \( q_{new} \leftarrow \text{random sample in } C \)
- \( \text{if } q_{new} \in C_{\text{free}} \)
  - \( G.\text{addVertex}(q_{new}) \)
  - \( \text{for each } q \in \text{Neighbor}(G, q_{new}) \)
  - \( \text{if } \text{link}(q, q_{new}) \)
  - \( G.\text{addEdge}(q_{new}, q) \)
Probabilistic Road Maps

Construction Phase:

Initialization:
\[ G = (V,E), \quad V = \{\}, \quad E = \{\} \]

Algorithm:
while \( |E| < N \)
    \( q_{\text{new}} \leftarrow \) random sample in \( C \)
    if \( q_{\text{new}} \in C_{\text{free}} \)
        G.addVertex(\( q_{\text{new}} \))
    for each \( q \in \text{Neighbor}(G, q_{\text{new}}) \)
        If \( \text{link}(q,q_{\text{new}}) \)
        G. addEdge(\( q_{\text{new}}, q \))
Local Planner

How do we create a local path for two samples?

We could use in theory any motion planner, but it needs to be fast -> Called many times

**Most common way:** direct line between two configurations

Test if path is collision free. If it is, local planner returns success. If not, local planner returns failure.
Probabilistic Road Maps

1. Phase – Learning Phase:
   • Construction Step: obtain reasonable connected graph with good coverage
   • Expansion Step: improve connectivity of difficult regions in C-Space

2. Phase – Query Phase:
   • Find path from $q_i$ to $q_G$

Probabilistic Road Maps

Expansion Step: Improve connectivity

In easy scenes with high number of sampled points: $G$ is most likely well connected

**But:** in more complex scenes, the graph might be disconnected and not capture the connectivity of $C_{\text{free}}$

Use heuristics to improve connectivity!
Probabilistic Road Maps

Expansion Step:
Heuristics to improve connectivity

Most heuristics exploit properties that are specific to the Problem/C-space shape

Vertex enhancement

Focus on vertices that were difficult to connect during construction
Defines weight for each node based on

\[ r(c) = \frac{n_f}{n_t + 1} \]

Expand difficult nodes through random walks

Other Heuristics to improve connectivity (during construction):
Different sampling strategies like Gaussian sampling, Bridge-test sampling
Probabilistic Road Maps

1. Phase – Learning Phase:
   • Construction Step: obtain reasonable connected graph with good coverage
   • Expansion Step: improve connectivity of difficult regions in C-Space

2. Phase – Query Phase:
   • Find path from $q_i$ to $q_G$

Probabilistic Road Maps

Query Phase:

**Path Planning**: For a new pair of start ($q_I$) and goal ($q_G$) configuration
1) Connect $q_I$ and $q_G$ to road map to $q'$ and $q''$
2) Find path from $q'$ and $q''$ in $G$
Probabilistic Road Maps

Query Phase:

**Path Planning**: For a new pair of start ($q_i$) and goal ($q_G$) configuration

1) Connect $q_i$ and $q_G$ to road map to $q'$ and $q''$
2) Find path from $q'$ and $q''$ in $G$
Query Phase:

**Is this a good path to follow?**
Randomized motion planners usually find jagged and longer path than necessary that are not suited for execution: -> need to smoothen
Path Smoothing:

**Shortcutting:**
- Along the found path: select randomly to points $q_1$ and $q_2$ and try to connect them
- If successful replace new path with old one.
- Repeat
Probabilistic Road Maps

Path Smoothing:

**Shortcutting:**
- Along the found path: select randomly to points $q_1$ and $q_2$ and try to connect them
- If successful replace new path with old one.
- Repeat
Probabilistic Road Maps

Path Smoothing:

**Shortcutting:**
- Along the found path: select randomly to points $q_1$ and $q_2$ and try to connect them.
- If successful replace new path with old one.
- Repeat

![Diagram showing path smoothing process](image)
Probabilistic Road Maps

Path Smoothing:

**Shortcutting:**
- Along the found path: select randomly to points $q_1$ and $q_2$ and try to connect them
- If successful replace new path with old one.
- Repeat

**Nonlinear optimization:**
- Specify object function to optimize path
Probabilistic Road Maps

What are scenarios in which the PRM might fail?

1. Accessibility or Departability constraint not fulfilled
   Why?
   Coverage
   Milestones should be distributed such that any point in $C_{free}$ can connect to $G$

2. Connectivity of $C_{free}$ not correctly represented in $G$
   Connectivity
   Connectivity of $G$ must represent connectivity of $C_{free}$
Probabilistic Road Maps

Expansive Spaces
D. Hsu et al.: Path Planning in Expansive Configuration Spaces; 1999

Theoretical analysis
Captures the difficulty of obtaining a good road map in terms of

\((\varepsilon, \alpha, \beta) - \text{expansiveness}\)

1. Guarantees that every point in \(C_{\text{free}}\) sees at least an \(\varepsilon\) fraction of the free space (\(\varepsilon\)-goodness)

\[ V(q) : \text{set of all configurations seen by } q \]

\(\varepsilon\)-goodness: sees at least \(\varepsilon\) fraction of space

\[ \mu(V(q)) \geq \mu(V(C_{\text{free}})) \]

\(C_{\text{free}}\) is \(\varepsilon\)-good if every configuration in \(C_{\text{free}}\) is \(\varepsilon\)-good
Probabilistic Road Maps

Expansive Spaces

D. Hsu et al.: Path Planning in Expansive Configuration Spaces; 1999

Theoretical analysis
Captures the difficulty of obtaining a good road map in terms of

$$(\varepsilon, \alpha, \beta) - \text{expansiveness}$$

2. Guarantees that every subset $S \subseteq C_{free}'$ has a relatively large lookout (has a $\beta$-lookout whose volume is at least $\alpha \mu(S)$)

$\beta$-lookout: Subset of $S$ that can see at least a $\beta$-fraction of $C_{free} \setminus S$
Probabilistic Road Maps

Expansive Spaces
D. Hsu et al.: Path Planning in Expansive Configuration Spaces; 1999

Theorem:
A roadmap of
\[ \frac{16\ln(1/\gamma) + 6}{\varepsilon \alpha \gamma} \]

Uniformly sampled milestones has correct connectivity with probability at least \((1-\gamma)\)

Cannot compute \(\varepsilon, \alpha, \) and \(\beta\) in advance, but:
1) Tells us that the probability that a roadmap does not conform to the connectivity of \(C_{free}\) decreases exponentially with the number of milestones
2) Need to increase milestones moderately when \(\varepsilon, \alpha, \) and \(\beta\) decrease
Probabilistic Road Maps

Summary

Idea:
• Build a roadmap from sampled configuration space points
• Check if in $C_{free}$ by using a collision checker, and
• search the roadmap to find a path.

Algorithm

$$G = (V, E) \quad V \leftarrow \emptyset, \quad E \leftarrow \emptyset$$

while $|E| < N$

$$q_{new} \leftarrow \text{random sample in } C$$

if $q_{new} \in C_{free}$

add $q_{new}$ to $V$

for each $q \in \text{Neighbor}(G, q_{new})$

If $\text{link}(q, q_{new})$

add edge $(q_{new}, q)$ to $E$

Properties:
• Provides feasible, but not optimal solution
• Works well on practical problems with fixed obstacle sets
• Probabilistic complete
• Problems with narrow passages
• Spending a lot of time of pre-processing to explore whole space to create road map
**Single Query Planner**

**Sampling Based Planning**

**Multi-Query**

Two phases:
1) Build Roadmap  
   2) Search

Assumes fixed obstacle setting over multiple queries.

**Probabilistic Road Maps**

[Kavaki et al.: Probabilistic roadmaps for path planning in high dimensional configuration spaces; 1996]

**Single-Query**

Iteratively build tree with search.

**Rapidly exploring Random Trees**

Kuffner and LaValle: Rapidly Exploring Random Trees; 2000

In a changing environment: Just interested once in the path and do not want to explore whole configuration space

Interleave 1) and two 2) until path found
Single Query Planner

Key Ideas

Idea 1:
• Add $q_I$ and $q_G$ to the milestones, connect them to $G$ and stop sampling if they are in the same connected component

Idea 2:
• Keep track of $q_I$ and $q_G$ connected components
• Sample with bias to $q_I$ and $q_G$

Idea 3:
• Make the graph a tree
Rapidly-exploring Random Trees (RRT)

Algorithm

\[ \text{RRT}(N, \Delta t, q_l): \]
\[ T.\text{init}(q_l) \]
\[ \text{For } k=1 \text{ to } N: \]

\[ q_l \rightarrow q_n \rightarrow q_r \]
Rapidly-exploring Random Trees (RRT)

Algorithm

\[ \text{RRT}(N, \Delta d, q_i): \]

\[ T.\text{init}(q_I) \]

For \( k = 1 \) to \( N \):

\[ q_r \leftarrow \text{random\_sample()} \]

\[ q_n \leftarrow \text{nearest\_neighbor}(q_r, T) \]

if new\_state\((q_r, q_n, q_c, u, \Delta t)\) then

\[ T.\text{add\_vertex}(q_c) \]

\[ T.\text{add\_edge}(q_n, q_c, q_i) \]

end if

Return \( T \)

Kuffner and LaValle: Rapidly Exploring Random Trees; ICRA 00
Rapidly-exploring Random Trees (RRT)

RRT is biased by large Voronoi regions

http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf
Rapidly-exploring Random Trees (RRT)

Goal Bias

Idea: bias sampling towards the goal configuration to create quickly a path to $q_G$

How:
With probability $p$ sample $q_G$ instead of uniform sampled configuration

Choose small $p$ (1 to 10 %)

If bias too strong: RRT becomes too greedy and gets stuck in local minima
Rapidly-exploring Random Trees (RRT)

LaValle: Planning Algorithms
Rapidly-exploring Random Trees (RRT)

Balanced Bidirectional Search

Sometimes it is easier to search from both directions

Idea:
- Grow trees from both \( q_i \) and \( q_G \)
- Make sure both trees are balanced: switch between extending and connecting after a few iterations
Rapidly-exploring Random Trees (RRT)

Summary

Idea:
• Probe and explore the configuration space by incrementally expanding from initial configuration into open space
• Use tree instead of graph that is rooted at $q_i$

Algorithm:
$T$.init($q_I$)
For $k=1$ to $N$:
  $q_r \leftarrow$ random_sample()
  $q_n \leftarrow$ nearest_neighbor($q_r, T$)
  if new_state($q_r, q_n, q_c, u, \Delta t$)
    $T$.add_vertex($q_c$)
    $T$.add_edge($q_n, q_c, q_l$)
Return $T$

Pros:
• Easy to implement
• Balance between greedy search and exploration with goal bias
• Well suited for high-dimensional practical problems

Cons:
• Much more sensitive to metric
• Only feasible path, not optimal
• Path needs to be smoothed

Katharina Muelling (NREC, Carnegie Mellon University)
## Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithms</strong></td>
<td>• Visibility Graphs&lt;br&gt;• Generalized Voronoi Diagrams&lt;br&gt;• Exact Cell Decomposition&lt;br&gt;• Approximate Cell Decomposition</td>
<td>• Probabilistic Road Maps&lt;br&gt;• Rapidly-exploring Random Trees</td>
</tr>
<tr>
<td><strong>Concept</strong></td>
<td>• Explicit representation of $C_{obs}$&lt;br&gt;• Captures connectivity of $C_{free}$ in graph and finds path with graph search</td>
<td>• Sample in C-Space&lt;br&gt;• Use collision detection to probe for collisions with obstacles</td>
</tr>
<tr>
<td><strong>Pros and Cons</strong></td>
<td><strong>Pros</strong>&lt;br&gt;complete&lt;br&gt;Becomes intractable with increasing C-Space and object/robot complexity</td>
<td><strong>Pros</strong>&lt;br&gt;More efficient in most high-dimensional practical problems&lt;br&gt;<strong>Cons</strong>&lt;br&gt;• Weaker guarantees&lt;br&gt;• Problems with narrow passages&lt;br&gt;• Path not optimal, needs smoothing</td>
</tr>
<tr>
<td><strong>Completeness</strong></td>
<td>Complete</td>
<td>Probabilistic Complete</td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>Number Queries</th>
<th>PRM</th>
<th>RRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple</td>
<td>Two steps:</td>
<td>Iteratively build tree with search</td>
</tr>
<tr>
<td>Single</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1) Pre-processing: Explore C-Space through randomly sampling + creating road map</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2) Search query for qI and qG</td>
<td></td>
</tr>
<tr>
<td>Pros</td>
<td>• Uniformly covering space</td>
<td>• Balance between greedy search and exploration</td>
</tr>
<tr>
<td></td>
<td>• Very effective in high dimensional spaces</td>
<td>• Efficient in high dimensional spaces</td>
</tr>
<tr>
<td>Cons</td>
<td>• Narrow passages</td>
<td>• Metric sensitive</td>
</tr>
<tr>
<td></td>
<td>• Needs lots of samples to explore whole space</td>
<td>• Can take a long time</td>
</tr>
</tbody>
</table>