Probabilistic Roadmap Path Planning

Reference: Principles of Robot Motion
H. Choset et. al.
MIT Press
Probabilistic Roadmap Path Planning

• Explicit Geometry based planners (grown obstacles, Voronoi etc) impractical in high dimensional spaces.
• Exact solutions with complex geometries are provably exponential
• Sampling based planners can often create plans in high-dimensional spaces efficiently
• Rather than Compute the C-Space explicitly, we Sample it
Explicitly computing C-Space for more than 3 DOF is prohibitive!

Figure 7.1 Snapshots along a path of a planar manipulator with ten degrees of freedom. The manipulator has a fixed base and its first three links have prismatic joints—they can extend one and a half times their original length. (From Kavraki [22].)
Figure 7.2  Path-planning problems. (a) Industrial manipulation. (b) Welding. (c) Planning removal paths for a part (the "robot") located at the center of the figure. (d) Computer animation. (e) Planning aircraft motion. (f) Humanoid robot. (g) Folding of a small peptide molecule. (a) From Boka and Kavraki [54]; (b) from Hsu and Latombe [196]; (c) courtesy of Latombe; (d) from Koga, Kondo, Kuffner and Latombe [241]; (e) from Kuffner and LaValle [272]; (f) from Kuffner [248]; (g) from Amato [21].
Notion of Completeness in Planning

• Complete Planner: always answers a path planning query correctly in bounded time
• Probabilistic Complete Planner: if a solution exists, planner will eventually find it, using random sampling (e.g. Monte Carlo sampling)
• Resolution Complete Planner: same as above but based on a deterministic sampling (e.g. sampling on a fixed grid).
Sampling Based-Planners

- Do not attempt to explicitly construct the C-Space and its boundaries
- Simply need to know if a single robot configuration is in collision
- Exploits simple tests for collision with full knowledge of the space
- Collision detection is a separate module—can be tailored to the application
- As collision detection improves, so do these algorithms
- Different approaches for single-query and multi-query requests
PRM Planner

- Roadmap is a graph $G(V,E)$
- Robot configuration $q \rightarrow Q_{\text{free}}$ is a vertex
- Edge $(q_1, q_2)$ implies collision-free path between these robot configurations
- A metric is needed for $d(q_1,q_2)$ (e.g. Euclidean distance)
- Uses coarse sampling of the nodes, and fine sampling of the edges
- Result: a roadmap in $Q_{\text{free}}$
PRM Planner: Step 1, Learning the Map

- Initially empty Graph $G$
- A configuration $q$ is randomly chosen
- If $q \rightarrow Q_{\text{free}}$ then added to $G$ (collision detection needed here)
- Repeat until $N$ vertices chosen
- For each $q$, select $k$ closest neighbors
- Local planner $\Delta$ connects $q$ to neighbor $q'$
- If connect successful (i.e. collision free local path), add edge $(q, q')$
**Algorithm 6 Roadmap Construction Algorithm**

**Input:**

- $n$: number of nodes to put in the roadmap
- $k$: number of closest neighbors to examine for each configuration

**Output:**

A roadmap $G = (V, E)$

1. $V \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. **while** $|V| < n$ **do**
   4. **repeat**
      5. $q \leftarrow$ a random configuration in $Q$
      6. **until** $q$ is collision-free
      7. $V \leftarrow V \cup \{q\}$
   8. **end while**
9. **for all** $q \in V$ **do**
   10. $N_q \leftarrow$ the $k$ closest neighbors of $q$ chosen from $V$ according to $\text{dist}$
   11. **for all** $q' \in N_q$ **do**
       12. **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq \text{NIL}$ **then**
           13. $E \leftarrow E \cup \{(q, q')\}$
       **end if**
   14. **end for**
15. **end for**
Figure 7.3  An example of a roadmap for a point robot in a two-dimensional Euclidean space. The gray areas are obstacles. The empty circles correspond to the nodes of the roadmap. The straight lines between circles correspond to edges. The number of $k$ closest neighbors for the construction of the roadmap is three. The degree of a node can be greater than three since it may be included in the closest neighbor list of many nodes.
PRM Planner: Step 2, Finding a Path

• Given $q_{\text{init}}$ and $q_{\text{goal}}$, need to connect each to the roadmap
• Find $k$ nearest neighbors of $q_{\text{init}}$ and $q_{\text{goal}}$ in roadmap, plan local path $\Delta$
• Problem: Roadmap Graph may have disconnected components...
• Need to find connections from $q_{\text{init}}$, $q_{\text{goal}}$ to same component
• Once on roadmap, use Dijkstra algorithm
Algorithm 7 Solve Query Algorithm

Input:
$q_{\text{init}}$: the initial configuration
$q_{\text{goal}}$: the goal configuration
$k$: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:
A path from $q_{\text{init}}$ to $q_{\text{goal}}$ or failure

1: $N_{q_{\text{init}}} \leftarrow$ the $k$ closest neighbors of $q_{\text{init}}$ from $V$ according to $\text{dist}$
2: $N_{q_{\text{goal}}} \leftarrow$ the $k$ closest neighbors of $q_{\text{goal}}$ from $V$ according to $\text{dist}$
3: $V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
4: set $q'$ to be the closest neighbor of $q_{\text{init}}$ in $N_{q_{\text{init}}}$
5: repeat
6: if $\Delta(q_{\text{init}}, q') \neq \text{NIL}$ then
7: $E \leftarrow (q_{\text{init}}, q') \cup E$
8: else
9: set $q'$ to be the next closest neighbor of $q_{\text{init}}$ in $N_{q_{\text{init}}}$
10: end if
11: until a connection was successful or the set $N_{q_{\text{init}}}$ is empty
12: set $q'$ to be the closest neighbor of $q_{\text{goal}}$ in $N_{q_{\text{goal}}}$
13: repeat
14: if $\Delta(q_{\text{goal}}, q') \neq \text{NIL}$ then
15: $E \leftarrow (q_{\text{goal}}, q') \cup E$
16: else
17: set $q'$ to be the next closest neighbor of $q_{\text{goal}}$ in $N_{q_{\text{goal}}}$
18: end if
19: until a connection was successful or the set $N_{q_{\text{goal}}}$ is empty
20: $P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
21: if $P$ is not empty then
22: return $P$
23: else
24: return failure
25: end if
Figure 7.4  An example of how to solve a query with the roadmap from figure 7.3. The configurations $q_{\text{init}}$ and $q_{\text{goal}}$ are first connected to the roadmap through $q'$ and $q''$. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.
PRM Planner – unanswered questions

• How are random configurations chosen?
• How are closest neighbors found?
• How do we choose distance function?
• How are local path’s generated?
PRM Sampling and Connectivity

- **Sampling:** Uniform random sampling of $Q_{\text{free}}$
- Can be multi-dimensional (e.g. translation and rotation, both 2-D or 3-D or higher)
- **Connectivity:** need to find nearest neighbors
- Naïve search is $O(n)$
- K-D trees are efficient ways to find nearest neighbors
- **Cost:** $O(\sqrt{n})$ for $d=2$
(a) The way the plane is subdivided.  
(b) The corresponding binary tree.

**Figure 7.5** A kd-tree for ten points on a plane.

Applet: [http://donar.umiacs.umd.edu/quadtree/points/kdtree.html](http://donar.umiacs.umd.edu/quadtree/points/kdtree.html)
Local Planner

• Important aspect of PRM algorithm
• Tradeoff:
  – powerful planners are slow, but can find paths between relatively isolated nodes
  – fast planner is less accurate, more nodes need to be generated, called more often
• Local planner also needed for finding path from q_init and q_goal to roadmap
Local Planner

• Simplest: straight line planner. Connect $q$ and $q'$ by linear segment

• Now check the segment for collisions:
  – Incremental: use a small step size and iterate over the linear segment
  – Subdivision: use binary search decomposition to check for collisions
(a) Incremental: The algorithm returns failure after five collision checks.  
(b) Subdivision: The algorithm returns failure after three collision checks.

**Figure 7.6** Sampling along the straight line path between two configurations $q'$ and $q''$. The numbers correspond to the order in which each strategy checks the samples for collision.
Postprocessing: Path Improvement

- Once a path is found, it can be optimized.
- Try connecting non-adjacent configurations. Choose $q_1$ and $q_2$ randomly, try to connect.
- Greedy approach: try connecting points $q_0$, $q_1$, …$q_n$ to $q_{goal}$.
- If $q_k$ connects to $q_{goal}$, do the above with $q_k$ as $q_{goal}$.
Figure 7.7 Processing the path returned from PRM to get a shorter path with the greedy approach.
Example: 6-DOF Path Planning

- Robot: Rigid non-convex object in 3 space
- Obstacle: Solid wall with small opening
- Random configuration is chosen from $\mathbb{R}^3$ for translation
- Axis and angle of rotation randomly chosen for rotation (quaternion representation)
Collision Detection

• Given configuration $q$ and nearest neighbor $q'$ we can use straight line collision detection.

• Each configuration $q=(p,r)=(\text{trans},\text{quaternion})$

• Check for collision by interpolating along line $(p,p')$ and along spherical interpolation $(r,r')$. 
Figure 7.8  An example of a motion-planning problem where both the robot and the obstacles are a collection of polyhedral objects in three dimensions. Parts of the robot on the other side of the wall are indicated by the darker color. (a) The initial and goal configuration of the query. (b) A path produced from a PRM with $n = 1000$ and $k = 10$. 
Distance Calculation for Rigid Object in 3D

- Distance function needed between 2 configurations \( q, q' \)
- Ideally, distance is the swept volume of the robot as it moves between configurations \( q \) and \( q' \). Difficult to compute exactly
- Method I: Can approximate this distance with an embedding in a Euclidean metric space: \( d(q,q') = || \text{embed}(q) - \text{embed}(q') || \)
  - Choose set of \( p \) points on robot, concatenate them, and create a vector of size \( p \times \text{dimension of workspace} \).
  - Example of rigid object in 3D: Create vector of size \( 3p \), choosing \( p \) points on the object. Intuitively, a “sampling” of the object’s Euclidean domain.
  - For configuration \( q \), \( \text{embed}(q) \) is the vector of \( p \) points transformed by the translation and rotation that is configuration \( q \). Transform each of the \( p \) points into the vector \( \text{embed}(q) \).
  - Do the same for configuration \( q' \), create \( \text{embed}(q') \).
  - Distance is now Euclidean distance between the \( 3p \) vectors:
    \[
    d(q,q') = || \text{embed}(q) - \text{embed}(q') ||
    \]
- How do you choose the \( p \) points?
- Cheaper solution: choose 2 points \( p1 \) and \( p2 \) of maximum extent on the object.
- Method II: Separate a configuration \( q \) into a translation \( X \) and a rotation \( R \): \( q=(X,R) \)
  - Calculate a weighted distance function \( d(q,q') = w1||X-X'|| + w2 f(R,R') \).
  - Need to use a metric on rotations – quaternions are good for this
  - Weights \( w1 \) and \( w2 \) need to be chosen, no real insight into this
OBPRM: Obstacle PRM

- If tight, small regions of the Cspace are needed to create a path (e.g. small opening in the wall), sampling may miss this
- Problem areas tend to be near the obstacles in tight spaces
- Solution: generate configuration q. If q in collision, choose random direction v and move q away from obstacle in direction v a small distance. If q now in Q_free, use this node
- Biases sampling near obstacles
Single-Query Sampling Based Planners

• PRM samples the entire space, plans paths anywhere
• Single query planners don’t explore all of Q_free, only relevant parts
• PRM can be used this way, inserting q_init and q_goal in Graph at beginning, then checking for a path
Random search

Often combined with potential field methods to escape minima

“Filling in” local minima

Random walks

random walks are not perfect...
RRT: Rapidly-exploring Random Trees

- Idea: sample Q_free for path from q_init to q_goal
- Use 2 trees, rooted at q_init and q_goal.
- As trees grow, the eventually share a common node, and are merged into a path
RRTs

1) Maintain a tree of configurations reachable from the starting point.
2) Choose a point at random from free space
3) Find the closest configuration already in the tree
4) Extend the tree in the direction of the new configuration

connects global & local information
Growth of an RRT

Example growth of an RRT - Biased toward the unexplored free space at each step.

Voronoi diagrams
A Mature RRT

http://msl.cs.uiuc.edu/rrt
http://www.kuffner.org/james/humanoid/planning.php
Path Planning with RRT Algorithm

• 2 trees, $T_{\text{init}}$, $T_{\text{goal}}$, rooted at $q_{\text{init}}$, $q_{\text{goal}}$

• Each tree is expanded by:
  – $q_{\text{rand}}$ is generated from uniform dist.
  – $q_{\text{near}}$ is found, nearest tree node to $q_{\text{rand}}$
  – move step-size along line $(q_{\text{near}}, q_{\text{rand}})$ to $q_{\text{new}}$. If no collision, add $q_{\text{new}}$ to tree

• If trees merge, path is found
Figure 7.14  Adding a new configuration to an RRT. Configuration $q_{rand}$ is selected randomly from a uniform distribution in $Q_{free}$. Configuration $q$ is the closest configuration in $T$ to $q_{rand}$ (this configuration is denoted as $q_{near}$ in the algorithm). Configuration $q_{new}$ is obtained by moving $q$ by $\text{step\_size}$ toward $q_{rand}$. Only $q_{new}$ and the edge $(q, q_{new})$ are added to the RRT.
Algorithm 10 Build RRT Algorithm

\textbf{Input:}
\begin{itemize}
    \item $q_0$: the configuration where the tree is rooted
    \item $n$: the number of attempts to expand the tree
\end{itemize}

\textbf{Output:}
A tree $T = (V, E)$ that is rooted at $q_0$ and has $\leq n$ configurations

1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: \textbf{for} $i = 1$ to $n$ \textbf{do}
4: \hspace{1em} $q_{\text{rand}} \leftarrow \text{a randomly chosen free configuration}$
5: \hspace{1em} \text{extend RRT} ($T$, $q_{\text{rand}}$)
6: \textbf{end for}
7: \textbf{return} $T$

Algorithm 11 Extend RRT Algorithm

\textbf{Input:}
\begin{itemize}
    \item $T = (V, E)$: an RRT
    \item $q$: a configuration toward which the tree $T$ is grown
\end{itemize}

\textbf{Output:}
A new configuration $q_{\text{new}}$ toward $q$, or NIL in case of failure

1: $q_{\text{near}} \leftarrow \text{closest neighbor of} \ q \ \text{in} \ T$
2: $q_{\text{new}} \leftarrow \text{progress} \ q_{\text{near}} \ \text{by} \ \text{step-size} \ \text{along the straight line in} \ Q \ \text{between} \ q_{\text{near}} \ \text{and} \ q_{\text{rand}}$
3: \textbf{if} $q_{\text{new}}$ is collision-free \textbf{then}
4: \hspace{1em} $V \leftarrow V \cup \{q_{\text{new}}\}$
5: \hspace{1em} $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: \hspace{1em} \textbf{return} $q_{\text{new}}$
7: \textbf{end if}
8: \textbf{return} NIL
RRT Algorithm

- Algorithm sensitive to step-size
- How far do we move along line (\(q_{\text{near}}, q_{\text{rand}}\))?
- Can a greedier algorithm work better?
- Why not move all the way to \(q_{\text{rand}}\)?
Algorithm 12 Connect RRT Algorithm

Input:

\[ T = (V, E) : \text{an RRT} \]
\[ q : \text{a configuration toward which the tree } T \text{ is grown} \]

Output:

connected if \( q \) is connected to \( T \); failure otherwise

1: repeat
2: \( q_{\text{new}} \leftarrow \text{extend RRT} \ (T, q) \)
3: until \( (q_{\text{new}} = q \text{ or } q_{\text{new}} = \text{NIL}) \)
4: if \( q_{\text{new}} = q \) then
5: \( \text{return connected} \)
6: else
7: \( \text{return failure} \)
8: end if
RRT Tradeoffs

• If step-size is small, many nodes generated, close together
• As number of nodes increases, nearest-neighbor computation slows down
• May be better to only add the last sample along the line (q_near, q_rand)
Shaping the RRT

- q_rand determines what direction we go
- What if q_rand == q_goal?
- Very greedy algorithm. Introduces too much bias
- Becomes a potential field planner that gets stuck in local minima
- Idea: use uniform q_rand with occasional q_rand == q_goal (maybe we get lucky?)
- Introducing just .05 bias towards goal, results improve
Merging RRT’s

Figure 7.15  Merging two RRTs. Configuration $q_{\text{rand}}$ is generated randomly from a uniform distribution in $Q_{\text{free}}$. Configuration $q_1$ was extended to $q_{\text{rand}}$. $q_2$ is the closest configuration to $q_{\text{rand}}$ in $T_{\text{goal}}$. It was possible to extend $q_2$ to $q_{\text{rand}}$. As a result, $T_{\text{init}}$ and $T_{\text{goal}}$ were merged.
Algorithm 13 Merge RRT Algorithm

Input:
  $T_1$: first RRT
  $T_2$: second RRT
  $\ell$: number of attempts allowed to merge $T_1$ and $T_2$

Output:
  merged if the two RRTs are connected to each other; failure otherwise

1: for $i = 1$ to $\ell$ do
2:    $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
3:    $q_{\text{new}, 1} \leftarrow$ extend RRT ($T_1$, $q_{\text{rand}}$)
4:    if $q_{\text{new}, 1} \neq \text{NIL}$ then
5:      $q_{\text{new}, 2} \leftarrow$ extend RRT ($T_2$, $q_{\text{new}, 1}$)
6:      if $q_{\text{new}, 1} = q_{\text{new}, 2}$ then
7:        return merged
8:    end if
9:    SWAP($T_1$, $T_2$)
10:   end if
11: end for
12: return failure
Using RRTs

http://msl.cs.uiuc.edu/rrt/
Bidirectional search
Additional complexity

additional degrees of freedom
Additional complexity

additional degrees of freedom

xy projections
Additional complexity

additional degrees of freedom

xy projections
time-lapse paths
Additional complexity

articulated linkages
Additional complexity

articulated linkages