CSE-571
Sampling-Based Motion Planning: RRTs

Various slides based on those from Pieter Abbeel, Zoe McCarthy
Many images from Lavalle, Planning Algorithms

1

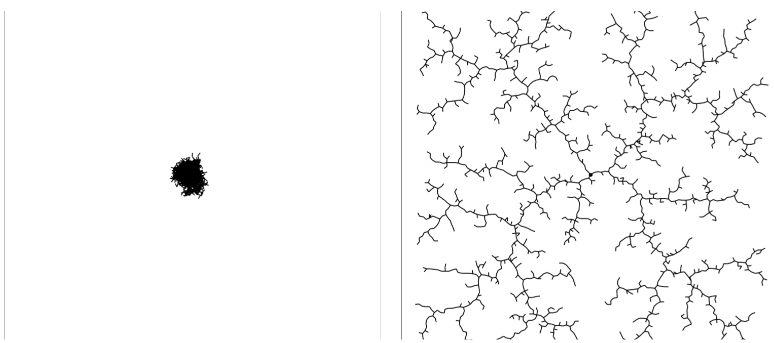
Rapidly exploring Random Tree (RRT)

Steve LaValle (98)

- Basic idea:
  - Build up a tree through generating “next states” in the tree by executing random controls
  - However: not exactly above to ensure good coverage

2

How to Sample



3

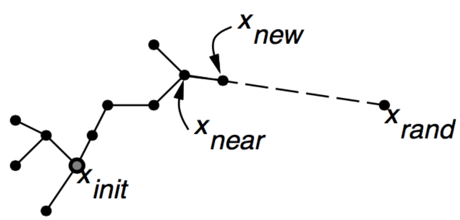
Rapidly exploring Random Tree (RRT)

- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region

4

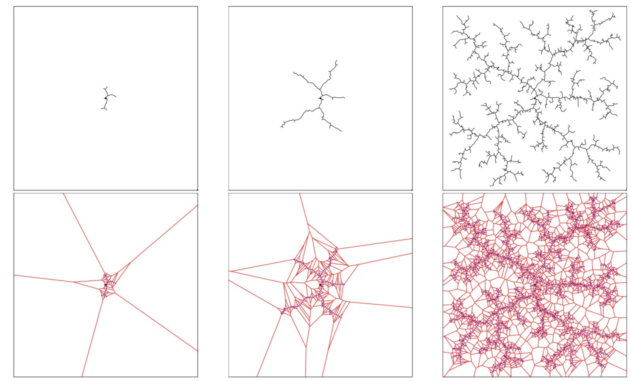
Rapidly exploring Random Tree (RRT)

- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region



5

Rapidly exploring Random Tree (RRT)



Source: LaValle and Kuffner 01

6

Rapidly exploring Random Tree (RRT)

GENERATE_RRT($x_{init}, K, \Delta t$)
- $\mathcal{T}$.init(x_{init});
- **for** $k = 1$ **to** K **do**
  - $x_{rand} \leftarrow$ RANDOM_STATE();
  - $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
  - $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
  - $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
  - $\mathcal{T}$.add_vertex(x_{new});
  - $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
- Return $\mathcal{T}$

RANDOM_STATE(): often uniformly at random over space with probability 99%, and the goal state with probability 1%, this ensures it attempts to connect to goal semi-regularly

7

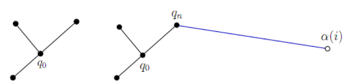
RRT Practicalities

- NEAREST_NEIGHBOR(x_{rand}, T): need to find (approximate) nearest neighbor efficiently
  - KD Trees data structure (upto 20-D) [e.g., FLANN]
  - Locality Sensitive Hashing
- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

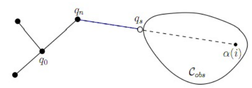
8

RRT Extension

- No obstacles, holonomic:



- With obstacles, holonomic:



- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem

9

Growing RRT

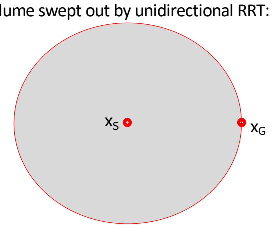
45 iterations 390 iterations

Demo: http://en.wikipedia.org/wiki/File:Rapidly-exploring_Random_Tree_(RRT)_500x373.gif

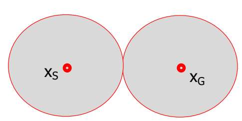
10

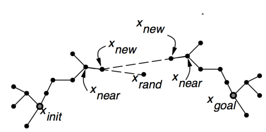
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:



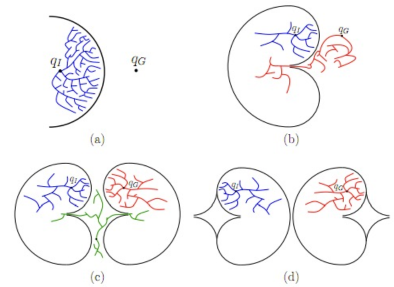


- Difference more and more pronounced as dimensionality increases

11

Multi-directional RRT

- Planning around obstacles or through narrow passages can often be easier in one direction than the other



12

RRT*

- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

13

RRT*

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

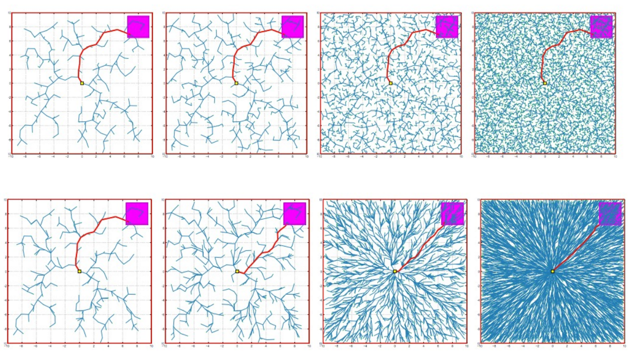
Source: Karaman and Frazzoli

14

RRT*

RRT

RRT*



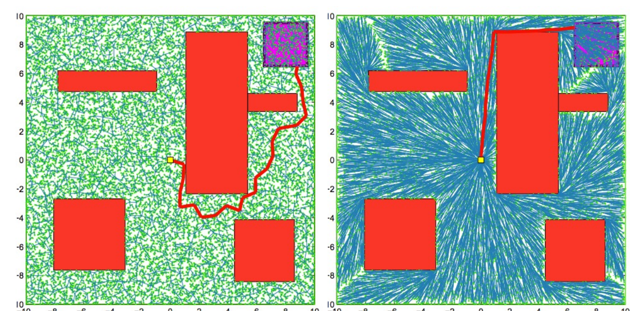
Source: Karaman and Frazzoli

15

RRT*

RRT

RRT*



Source: Karaman and Frazzoli

16

Smoothing

Randomized motion planners tend to find not so great paths for execution: very jagged, often much longer than necessary.

→ In practice: do smoothing before using the path

- Shortcutting:
  - along the found path, pick two vertices x_{t1}, x_{t2} and try to connect them directly (skipping over all intermediate vertices)
- Nonlinear optimization for optimal control
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

17

Additional Resources

- Marco Pavone (http://asl.stanford.edu/):
  - Sampling-based motion planning on GPUs: https://arxiv.org/pdf/1705.02403.pdf
  - Learning sampling distributions: https://arxiv.org/pdf/1709.05448.pdf
- Sidd Srinivasa (https://personalrobotics.cs.washington.edu/)
  - Batch informed trees: https://robotic-esp.com/code/bitstar/
  - Expensive edge evals: https://arxiv.org/pdf/2002.11853.pdf
- Adam Fishman / Dieter Fox (https://rse-lab.cs.washington.edu/)
  - Motion Policy Networks: https://mpinets.github.io/
- Lydia Kavraki (http://www.kavrakilab.org/)
  - Motion in human workspaces: http://www.kavrakilab.org/nsf-nri-1317849.html

18