CSE-571
Sampling-Based Motion Planning

Slides based on those from Pieter Abbeel, Zoe McCarthy
Many images from Lavalle, Planning Algorithms
Could try by, for example, following formulation:

\[
\begin{align*}
\min_{u,x} & \quad (x_T - x_G)^T (x_T - x_G) \\
\text{s.t.} & \quad x_{t+1} = f(x_t, u_t) \quad \forall t \\
& \quad u_t \in \mathcal{U}_t \\
& \quad x_t \in \mathcal{X}_t \\
& \quad x_0 = x_S 
\end{align*}
\]

Or, with constraints, (which would require using an infeasible method):

\[
\begin{align*}
\min_{u,x} & \quad ||u|| \\
\text{s.t.} & \quad x_{t+1} = f(x_t, u_t) \quad \forall t \\
& \quad u_t \in \mathcal{U}_t \\
& \quad x_t \in \mathcal{X}_t \\
& \quad x_0 = x_S \\
& \quad X_T = x_G 
\end{align*}
\]

Can work surprisingly well, but for more complicated problems can get stuck in infeasible local minima.
Examples

Helicopter path planning

Cartpole swing-up

Acrobot
Examples
Examples
Examples
Motion Planning: Outline

- Configuration Space
- Probabilistic Roadmap
- Rapidly-exploring Random Trees (RRTs)
- Extensions
- Smoothing
Configuration Space (C-Space)

\[ \{ x \mid x \text{ is a pose of the robot} \} \]

- obstacles $\rightarrow$ configuration space obstacles

**Workspace**

(2 DOF: translation only, no rotation)

**Configuration Space**

free space □

obstacles □ □
Motion planning
Probabilistic Roadmap (PRM)

Space $\mathbb{R}^n$  
forbidden space  
Free/feasible space
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random.
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random
Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision
The collision-free configurations are retained as milestones.
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors.
The collision-free links are retained as local paths to form the PRM
The start and goal configurations are included as milestones.
The PRM is searched for a path from s to g
Probabilistic Roadmap

- Initialize set of points with $x_S$ and $x_G$
- Randomly sample points in configuration space
- Connect nearby points if they can be reached from each other
- Find path from $x_S$ to $x_G$ in the graph

- Alternatively: keep track of connected components incrementally, and declare success when $x_S$ and $x_G$ are in same connected component
PRM Example 1
PRM Example 2
PRM’s Pros and Cons

- **Pro:**
  - Probabilistically complete: i.e., with probability one, if run for long enough the graph will contain a solution path if one exists.

- **Cons:**
  - Required to solve 2-point boundary value problem
  - Build graph over state space but no focus on generating a path
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
How to Sample
Rapidly exploring Random Tree (RRT)

- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region
Rapidly exploring Random Tree (RRT)

- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region
**Rapidly exploring Random Tree (RRT)**

\[
\text{GENERATE}_\text{RRT}(x_{\text{init}}, K, \Delta t)
\]

1. \( \mathcal{T}.\text{init}(x_{\text{init}}); \)
2. \textbf{for } k = 1 \textbf{ to } K \textbf{ do}
3. \( x_{\text{rand}} \leftarrow \text{RANDOM\_STATE}(); \)
4. \( x_{\text{near}} \leftarrow \text{NEAREST\_NEIGHBOR}(x_{\text{rand}}, \mathcal{T}); \)
5. \( u \leftarrow \text{SELECT\_INPUT}(x_{\text{rand}}, x_{\text{near}}); \)
6. \( x_{\text{new}} \leftarrow \text{NEW\_STATE}(x_{\text{near}}, u, \Delta t); \)
7. \( \mathcal{T}.\text{add\_vertex}(x_{\text{new}}); \)
8. \( \mathcal{T}.\text{add\_edge}(x_{\text{near}}, x_{\text{new}}, u); \)
9. \textbf{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
Rapidly exploring Random Tree (RRT)
RRT Practicalities

- **NEAREST_NEIGHBOR**(\(x_{\text{rand}}, T\)): need to find (approximate) nearest neighbor efficiently
  - KD Trees data structure (upto 20-D) [e.g., FLANN]
  - Locality Sensitive Hashing

- **SELECT_INPUT**\(\left(x_{\text{rand}}, x_{\text{near}}\right)\)
  - 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.
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
Growing RRT

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

- Volume swept out by unidirectional RRT:

- Volume swept out by bi-directional RRT:

- Difference more and more pronounced as dimensionality increases
Planning around obstacles or through narrow passages can often be easier in one direction than the other.
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
Algorithm 6: RRT*

1. $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset$
2. for $i = 1, \ldots, n$ do
3.     $x_{\text{rand}} \leftarrow \text{SampleFree};$
4.     $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}});$
5.     $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$
6.     if $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ then
7.         $X_{\text{near}} \leftarrow \text{Near}(G = (V, E), x_{\text{new}}, \min\{c_{\text{RRT}} \cdot (\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\});$
8.         $V \leftarrow V \cup \{x_{\text{new}}\};$
9.         $x_{\text{min}} \leftarrow x_{\text{nearest}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}));$
10.    foreach $x_{\text{near}} \in X_{\text{near}}$ do
11.        // Connect along a minimum-cost path
12.        if $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \land \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ then
13.            $x_{\text{min}} \leftarrow x_{\text{near}}; c_{\text{min}} \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}));$
14.    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$
15.    foreach $x_{\text{near}} \in X_{\text{near}}$ do
16.        // Rewire the tree
17.        if $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \land \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{Cost}(x_{\text{near}})$ then
18.            $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$
19.            $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\};$
20.    return $G = (V, E);$
RRT*
RRT*  

Source: Karaman and Frazzoli
Real Time RRT*
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 $X_{t_1}, X_{t_2}$ 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.
Additional Resources

- Marco Pavone (http://asl.stanford.edu/):
  - Sampling-based motion planning on GPUs: https://arxiv.org/pdf/1705.02403.pdf

- Sidd Srinivasa (https://personalrobotics.cs.washington.edu/)
  - Batch informed trees: https://robotic-esp.com/code/bitstar/

- 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