ECE276B: Planning & Learning in Robotics
Lecture 8: Sampling-based Planning

Instructor:
Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:
Zhichao Li: zhl355@eng.ucsd.edu
Ehsan Zobeidi: ezobeidi@eng.ucsd.edu
Ibrahim Akbar: iakbar@eng.ucsd.edu
Search-based vs Sampling-based Planning

▶ Search-based planning:
  ▶ Generates a systematic discrete representation (graph) of $C_{\text{free}}$
  ▶ Searches the representation for a path guaranteeing to find one if it exists (resolution complete)
  ▶ Can interleave the representation construction with the search, i.e., adds nodes only when necessary
  ▶ Provides suboptimality bounds on the solution
  ▶ Can get computationally expensive in high dimensions
Search-based vs. Sampling-based Planning

- **Sampling-based planning:**
  - Generates a sparse sample-based representation (graph) of $C_{free}$
  - Searches the representation for a path guaranteeing that the probability of finding one if it exists approaches 1 as the number of iterations $\to \infty$ (probabilistically complete)
  - Can interleave the representation construction with the search, i.e., adds samples only when necessary
  - Provides asymptotic suboptimality bounds on the solution
  - Well-suited for high-dimensional planning as it is faster and requires less memory than search-based planning in many domains
Motion Planning Problem

- Configuration space: $C$; Obstacle space: $C_{obs}$; Free space: $C_{free}$

- Initial state: $x_s \in C_{free}$; Goal state: $x_T \in C_{free}$

- **Path**: a continuous function $Q : [0, 1] \rightarrow C$; Set of all paths: $\mathcal{Q}$

- **Feasible path**: a continuous function $Q : [0, 1] \rightarrow C_{free}$ such that $Q(0) = x_s$ and $Q(1) = x_T$; Set of all feasible paths: $\mathcal{Q}_{s,T}$

- **Motion Planning Problem** Given a path planning problem $(C_{free}, x_s, x_T)$ and a cost function $J : \mathcal{Q} \rightarrow \mathbb{R}_{\geq 0}$, find a feasible path $Q^*$ such that:

$$J(Q^*) = \min_{Q \in \mathcal{Q}_{s,T}} J(Q)$$

Report failure if no such path exists.
Primitive Procedures for Sampling-based Motion Planning

- **Sample**: returns iid samples from $C$

- **SampleFree**: returns iid samples from $C_{\text{free}}$

- **Nearest**: given a graph $G = (V, E)$ with $V \subset C$ and a point $x \in C$, returns a vertex $v \in V$ that is closest to $x$:

  $$\text{Nearest}((V, E), x) := \arg \min_{v \in V} \|x - v\|$$

- **Near**: given a graph $G = (V, E)$ with $V \subset C$, a point $x \in C$, and $r > 0$, returns the vertices in $V$ that are within a distance $r$ from $x$:

  $$\text{Near}((V, E), x, r) := \{v \in V \mid \|x - v\| \leq r\}$$

- **Steer**: given points $x, y \in C$ and $\epsilon > 0$, returns a point $z \in C$ that minimizes $\|z - y\|$ while remaining within $\epsilon$ from $x$:

  $$\text{Steer}_\epsilon(x, y) := \arg \min_{z : \|z - x\| \leq \epsilon} \|z - y\|$$

- **CollisionFree**: given points $x, y \in C$, returns TRUE if the line segment between $x$ and $y$ lies in $C_{\text{free}}$ and FALSE otherwise.
Probabilistic Roadmap (PRM)

Step 1. **Preprocessing Phase**: Build a roadmap (graph) $G$ which, hopefully, should be accessible from any point in $C_{\text{free}}$

- **Nodes**: randomly sampled valid configurations $x_i \in C_{\text{free}}$
- **Edges**: added between samples that are easy to connect with a simple local controller (e.g., follow straight line)

Step 2. **Query Phase**: Given a start configuration $x_s$ and goal configuration $x_T$, connect them to the roadmap $G$ using a local planner, then search the augmented roadmap for a shortest path from $x_s$ to $x_T$

- **Pros and Cons**:
  - Simple and highly effective in high dimensions
  - Can result in suboptimal paths, no guarantees on suboptimality
  - Difficulty with narrow passages
  - Useful for multiple queries with different start and goal in the same environment
Step 1: Preprocessing Phase

**Algorithm 1** PRM (preprocessing phase)

1: \( V \leftarrow \emptyset; \ E \leftarrow \emptyset \)
2: for \( i = 1, \ldots, n \) do
3: \( x_{\text{rand}} \leftarrow \text{SAMPLEFREE()} \)
4: \( V \leftarrow V \cup \{x_{\text{rand}}\} \)
5: for \( x \in \text{NEAR}((V, E), x_{\text{rand}}, r) \) do \( \triangleright \) May use \( k \) nearest vertices
6: if (not \( G.\text{same\_component}(x_{\text{rand}}, x) \)) and \( \text{COLLISIONFREE}(x_{\text{rand}}, x) \) then
7: \( E \leftarrow E \cup \{(x_{\text{rand}}, x); (x, x_{\text{rand}})\} \)
8: return \( G = (V, E) \)
Optimal Probabilistic Roadmap


- To achieve an asymptotically optimal PRM, the connection radius $r$ should decrease such that the average number of connections attempted from a roadmap vertex is proportional to $\log(n)$:

\[
r^* > 2 \left(1 + \frac{1}{d}\right)^{1/d} \left(\frac{Vol(C_{free})}{Vol(Unit \ d-ball)}\right)^{1/d} \left(\frac{\log(n)}{n}\right)^{1/d}
\]

Algorithm 2 PRM*

1. $V \leftarrow \{x_s\} \cup \{\text{SAMPLEFREE()}\}_{i=1}^n; \ E \leftarrow \emptyset$
2. for $v \in V$ do
3.     for $x \in \text{NEAR}((V, E), v, r^*) \setminus \{v\}$ do
4.         if $\text{COLLISIONFREE}(v, x)$ then
5.             $E \leftarrow E \cup \{(v, x), (x, v)\}$
6. return $G = (V, E)$
**PRM vs RRT**

- **PRM**: A graph constructed from random samples. It can be searched for a path whenever a start node \( x_s \) and goal node \( x_\tau \) are specified. PRMs are well-suited for repeated planning between different pairs of \( x_s \) and \( x_\tau \) (*multiple queries*).

- **RRT**: A tree is constructed from random samples with root \( x_s \). The tree is grown until it contains a path to \( x_\tau \). RRTs are well-suited for single-shot planning between a single pair of \( x_s \) and \( x_\tau \) (*single query*).

**Rapidly Exploring Random Tree (RRT):**

- One of the most popular planning techniques
- Introduced by Steven LaValle in 1998
- Many, many, many extensions and variants (articulated robots, kinematics, dynamics, differential constraints)
- There exist extensions of RRTs that try to reuse a previously constructed tree when replanning in response to map updates.
Rapidly Exploring Random Tree (RRT)

- Sample a new configuration $x_{\text{rand}}$, find the nearest neighbor $x_{\text{near}}$ in $G$ and connect them:

- If the nearest point $x_{\text{near}}$ lies on an existing edge, then split the edge:

- If there is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by a collision detection algorithm.
Rapidly Exploring Random Tree (RRT)

- What about the goal? Occasionally (e.g., every 100 iterations) add the goal configuration $x_T$ and see if it gets connected to the tree.

- RRT can be implemented in the original workspace (need to do collision checking) or in configuration space.

- Challenges with a C-Space implementation:
  - What distance function do we use to find the nearest configuration?
    - e.g., distance along the surface of a torus for a 2 link manipulator.
  
  - An edge represents a path in C-Space. How do we construct a collision-free path between two configurations?
    - We do not have to connect the configurations all the way. Instead, use a small step size $\epsilon$ and a local steering function to get closer to the second configuration.
Rapidly Exploring Random Tree (RRT)

- **No preprocessing**: starting with an initial configuration $x_s$ build a graph (actually, tree) until the goal configuration $x_T$ is part of it

**Algorithm 3 RRT**

1: $V \leftarrow \{x_s\}; E \leftarrow \emptyset$
2: for $i = 1 \ldots n$ do
3: \hspace{1em} $x_{\text{rand}} \leftarrow \text{SAMPLEFREE}()$
4: \hspace{1em} $x_{\text{nearest}} \leftarrow \text{NEAREST}((V, E), x_{\text{rand}})$
5: \hspace{1em} $x_{\text{new}} \leftarrow \text{STEER}(x_{\text{nearest}}, x_{\text{rand}})$
6: \hspace{1em} if $\text{COLLISIONFREE}(x_{\text{nearest}}, x_{\text{new}})$ then
7: \hspace{2em} $V \leftarrow V \cup \{x_{\text{new}}\}; E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$
8: return $G = (V, E)$
Rapidly Exploring Random Tree (RRT)

- RRT without $\epsilon$ (called Rapidly Exploring Dense Tree (RDT)):

- RRT with $\epsilon$

![Images of RRT and RDT with different numbers of iterations](image-url)
Example: RRT Algorithm

- Start node $x_s$
- Goal node $x_\tau$
- Gray obstacles
Example: RRT Algorithm

- Sample $x_{\text{rand}}$ in the workspace
- Steer from $x_s$ towards $x_{\text{rand}}$ by a fixed distance $\epsilon$ to get $x_1$
- If the segment from $x_s$ to $x_1$ is collision-free, insert $x_1$ into the tree
Example: RRT Algorithm

- Sample $x_{rand}$ in the workspace
- Find the closest node $x_{near}$ to $x_{rand}$
- Steer from $x_{near}$ towards $x_{rand}$ by a fixed distance $\epsilon$ to get $x_2$
- If the segment from $x_{near}$ to $x_2$ is collision-free, insert $x_2$ into the tree
Example: RRT Algorithm

- Sample $x_{\text{rand}}$ in the workspace
- Find the closest node $x_{\text{near}}$ to $x_{\text{rand}}$
- Steer from $x_{\text{near}}$ towards $x_{\text{rand}}$ by a fixed distance $\epsilon$ to get $x_3$
- If the segment from $x_{\text{near}}$ to $x_3$ is collision-free, insert $x_3$ into the tree
Example: RRT Algorithm

▸ Sample $x_{rand}$ in the workspace

▸ Find the closest node $x_{near}$ to $x_{rand}$

▸ Steer from $x_{near}$ towards $x_{rand}$ by a fixed distance $\epsilon$ to get $x_3$

▸ If the segment from $x_{near}$ to $x_3$ is collision-free, insert $x_3$ into the tree
Example: RRT Algorithm

- Continue until a node that is a distance $\epsilon$ from the goal is generated.
- Either terminate the algorithm or search for additional feasible paths.
Sampling in RRTs

- The vanilla RRT algorithm provides uniform coverage of space.
- Alternatively, the growth may be biased by the largest Voronoi region.
Sampling in RRTs

- Goal-biased sampling: with probability $(1 - p_g)$, $x_{rand}$ is chosen as a uniform sample in $C_{free}$ and with probability $p_g$, $x_{rand} = x_\tau$.

- (a) $p_g = 0$
- (b) $p_g = 0.1$
- (c) $p_g = 0.5$
Handling Robot Dynamics with Steer()

- Steer() extends the tree towards a given random sample \( x_{\text{rand}} \).

- Consider a car-like robot with non-holonomic constraints (can't slide sideways) in \( SE(2) \). Obtaining a feasible path from \( x_{\text{rand}} = (0, 0, 90^\circ) \) to \( x_{\text{near}} = (1, 0, 90^\circ) \) is as hard as the original problem.

- Steer() resolves this by not requiring the motion to get all the way to \( x_{\text{rand}} \). We just apply the best control input for a fixed duration to obtain \( x_{\text{new}} \) and a dynamically feasible trajectory to it.
Example: 5 DOF Kinodynamic Planning for a Car
Bug Traps

- Growing two trees, one from start and one for goal, often has better performance in practice.
Algorithm 4 Bi-directional RRT

1: \( V_a \leftarrow \{x_s\}; \ E_a \leftarrow \emptyset; \ V_b \leftarrow \{x_T\}; \ E_b \leftarrow \emptyset \)
2: \textbf{for} \( i = 1 \ldots n \) \textbf{do}
3: \quad \( x_{\text{rand}} \leftarrow \text{SAMPLEFREE()} \)
4: \quad \( x_{\text{nearest}} \leftarrow \text{NEAREST}((V_a, E_a), x_{\text{rand}}) \)
5: \quad \( x_c \leftarrow \text{STEER}(x_{\text{nearest}}, x_{\text{rand}}) \)
6: \quad \textbf{if} \( x_c \neq x_{\text{nearest}} \) \textbf{then}
7: \quad \quad \( V_a \leftarrow V_a \cup \{x_c\}; \ E_a \leftarrow \{(x_{\text{nearest}}, x_c), (x_c, x_{\text{nearest}})\} \)
8: \quad \quad \( x'_{\text{nearest}} \leftarrow \text{NEAREST}((V_b, E_b), x_c) \)
9: \quad \quad \( x'_c \leftarrow \text{STEER}(x'_{\text{nearest}}, x_c) \)
10: \quad \quad \textbf{if} \( x'_c \neq x'_{\text{nearest}} \) \textbf{then}
11: \quad \quad \quad \( V_b \leftarrow V_b \cup \{x'_c\}; \ E_b \leftarrow \{(x'_{\text{nearest}}, x'_c), (x'_c, x'_{\text{nearest}})\} \)
12: \quad \quad \quad \textbf{if} \( x'_c = x_c \) \textbf{then} \textbf{return} \text{SOLUTION}
13: \quad \quad \textbf{if} \( |V_b| < |V_a| \) \textbf{then} \text{SWAP}((V_a, E_a), (V_b, E_b))
14: \textbf{return} \text{FAILURE}
Bi-directional tree + relax the $\epsilon$ constraint on tree growth

**Algorithm 5** RRT-Connect

1: $V_a \leftarrow \{x_s\}; E_a \leftarrow \emptyset; V_b \leftarrow \{x_\tau\}; E_b \leftarrow \emptyset$
2: for $i = 1 \ldots n$ do
3: \hspace{0.5cm} $x_{\text{rand}} \leftarrow \text{SAMPLEFREE}()$
4: \hspace{0.5cm} if not $\text{EXTEND}((V_a, E_a), x_{\text{rand}}) = \text{Trapped}$ then
5: \hspace{1cm} if $\text{CONNECT}((V_b, E_b), x_{\text{new}}) = \text{Reached}$ then $\triangleright x_{\text{new}}$ was just added to $(V_a, E_a)$
6: \hspace{1cm} return $\text{PATH}((V_a, E_a), (V_b, E_b))$
7: \hspace{0.5cm} $\text{SWAP}((V_a, E_a), (V_b, E_b))$
8: return Failure
9: function $\text{EXTEND}((V, E), x)$
10: \hspace{0.5cm} $x_{\text{nearest}} \leftarrow \text{NEAREST}((V, E), x)$
11: \hspace{0.5cm} $x_{\text{new}} \leftarrow \text{STEER}(x_{\text{nearest}}, x)$
12: \hspace{0.5cm} if $\text{COLLISIONFREE}(x_{\text{near}}, x_{\text{new}})$ then
13: \hspace{1cm} $V \leftarrow \{x_{\text{new}}\}; E \leftarrow \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$
14: \hspace{1cm} if $x_{\text{new}} = x$ then return Reached else return Advanced
15: return Trapped
16: function $\text{CONNECT}((V, E), x)$
17: \hspace{0.5cm} repeat $\text{status} \leftarrow \text{EXTEND}((V, E), x)$ until $\text{status} \neq \text{Advanced}$
18: return $\text{status}
Example: Single RRT-Connect Iteration

$q_{init}$

$q_{goal}$
Example: Single RRT-Connect Iteration

- One tree is grown to a random target

$q_{\text{init}}$ $q_{\text{goal}}$
Example: Single RRT-Connect Iteration

- The new node becomes a target for the other tree.
Example: Single RRT-Connect Iteration

- Determine the nearest node to the target

![Diagram showing a tree structure with nodes and edges representing the RRT-Connect iteration process.](image-url)
Example: Single RRT-Connect Iteration

- Try to add a new collision-free branch
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch

![Diagram with nodes and arrows indicating the process of extending the branch]
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch
Example: Single RRT-Connect Iteration

- If the branch reaches all the way to the target, a feasible path is found!
Example: Single RRT-Connect Iteration

- If the branch reaches all the way to the target, a feasible path is found!
Example: RRT-Connect
Example: RRT-Connect
Example: RRT-Connect
Why are RRTs so popular?

- The algorithm is very simple once the following subroutines are implemented:
  - Random sample generator
  - Nearest neighbor
  - Collision checker
  - Steer

- Pros:
  - Sparse exploration requires little memory and computation
  - RRTs find feasible paths quickly in practice
  - Can add heuristics on top, e.g., bias the sampling towards the goal

- Cons:
  - Solutions can be highly sub-optimal and require path smoothing as a post-processing step
  - The smoothed path is still restricted to the same homotopy class
Path Smoothing

- Start with the initial point (1)
- Make connections to subsequent points in the path (2), (3), (4), ...
- When a connection collides with obstacles, add the previous waypoint to the smoothed path
- Continue smoothing from this point on
Search-based vs Sampling-based Planning

- **RRT:**
  - Sparse exploration requires little memory and computation
  - Solutions can be highly sub-optimal and require post-processing (path smoothing) which may be difficult

- **Weighted A\*:**
  - Systematic exploration may require a lot of memory and computation
  - Returns a path with (sub-)optimality guarantees
RRT: Probabilistic Completeness but No Optimality

RRT and RRT-Connect are **probabilistically complete**: the probability that a feasible path will be found if one exists, approaches 1 exponentially as the number of samples approaches infinity.

Assuming $C_{\text{free}}$ is connected, bounded, and open, for any $x \in C_{\text{free}}$, $\lim_{N \to \infty} \mathbb{P}(|x - x_{\text{near}}| < \epsilon) = 1$, where $x_{\text{near}}$ is the closest node to $x$ in $T$.

RRT is **not optimal**: the probability that RRT converges to an optimal solution, as the number of samples approaches infinity, is zero under reasonable technical assumptions (S. Karaman, E. Frazzoli, RSS’10).

**Problem**: once we build an RRT we never modify it.


- RRT + rewiring of the tree to ensure asymptotic optimality
- Contains two steps: **extend** (similar to RRT) and **rewire** (new)
RRT*: Extend Step

- Generate a new potential node $x_{new}$ identically to RRT
- Instead of finding the closest node in the tree, find all nodes within a neighborhood $\mathcal{N}$ of radius $\min\{r^*, \epsilon\}$ where
  \[
  r^* > 2 \left(1 + \frac{1}{d}\right)^{1/d} \left(\frac{\text{Vol}(C_{\text{free}})}{\text{Vol}(\text{Unit d-ball})}\right)^{1/d} \left(\frac{\log|V|}{|V|}\right)^{(1/d)}
  \]
- Let $x_{\text{nearest}} = \arg\min_{x_{\text{near}} \in \mathcal{N}} g_{x_{\text{near}}} + c_{x_{\text{near}}, x_{new}}$ be the node in $\mathcal{N}$ on the currently known shortest path from $x_s$ to $x_{new}$

- $V \leftarrow V \cup \{x_{new}\}$
- $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{new})\}$
- Set the label of $x_{new}$ to:
  \[
  g_{x_{new}} = g_{x_{\text{nearest}}} + c_{x_{\text{nearest}}, x_{new}}
  \]
RRT*: Rewire Step

- Check all nodes $x_{\text{near}} \in \mathcal{N}$ to see if re-routing through $x_{\text{new}}$ reduces the path length (label correcting!):

- If $g_{x_{\text{new}}} + c_{x_{\text{new}}, x_{\text{near}}} < g_{x_{\text{near}}}$, then remove the edge between $x_{\text{near}}$ and its parent and add a new edge between $x_{\text{near}}$ and $x_{\text{new}}$
Algorithm 6 RRT*

1: \( V \leftarrow \{x_s\}; E \leftarrow \emptyset \)
2: \textbf{for} \( i = 1 \ldots n \) \textbf{do}
3: \( x_{\text{rand}} \leftarrow \text{SAMPLEFREE()} \)
4: \( x_{\text{nearest}} \leftarrow \text{NEAREST}(\langle V, E \rangle, x_{\text{rand}}) \)
5: \( x_{\text{new}} \leftarrow \text{STEER}(x_{\text{nearest}}, x_{\text{rand}}) \)
6: \textbf{if} \ \text{COLLISIONFREE}(x_{\text{nearest}}, x_{\text{new}}) \ \textbf{then}
7: \( X_{\text{near}} \leftarrow \text{NEAR}(\langle V, E \rangle, x_{\text{new}}, \min\{r^*, \epsilon\}) \)
8: \( V \leftarrow V \cup \{x_{\text{new}}\} \)
9: \( c_{\text{min}} \leftarrow \text{COST}(x_{\text{nearest}}) + \text{COST}(\text{Line}(x_{\text{nearest}}, x_{\text{new}})) \)
10: \textbf{for} \( x_{\text{near}} \in X_{\text{near}} \) \textbf{do} \quad \triangleright \text{Extend along a minimum-cost path}
11: \textbf{if} \ \text{COLLISIONFREE}(x_{\text{near}}, x_{\text{new}}) \ \textbf{then}
12: \textbf{if} \ \text{COST}(x_{\text{near}}) + \text{COST}(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}} \ \textbf{then}
13: \( x_{\text{min}} \leftarrow x_{\text{near}} \)
14: \( c_{\text{min}} \leftarrow \text{COST}(x_{\text{near}}) + \text{COST}(\text{Line}(x_{\text{near}}, x_{\text{new}})) \)
15: \( E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\} \)
16: \textbf{for} \( x_{\text{near}} \in X_{\text{near}} \) \textbf{do} \quad \triangleright \text{Rewire the tree}
17: \textbf{if} \ \text{COLLISIONFREE}(x_{\text{new}}, x_{\text{near}}) \ \textbf{then}
18: \textbf{if} \ \text{COST}(x_{\text{new}}) + \text{COST}(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{COST}(x_{\text{near}}) \ \textbf{then}
19: \( x_{\text{parent}} \leftarrow \text{PARENT}(x_{\text{near}}) \)
20: \( E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\} \)
21: \textbf{return} \( G = (V, E) \)
RRT vs RRT*

- Same nodes in the tree, only the edge connections are different. Notice how the RRT* edges are almost straight lines (optimal paths).
RRT vs RRT*

▶ Same nodes in the tree, only the edge connections are different. Notice how the RRT* edges are almost straight lines (optimal paths).