Sampling-Based Motion Planning:
Single-Query Planners

CSCI 545 Introduction to Robotics
Instructor: Stefanos Nikolaidis
Samping-Based Methods

- Multi-Query Planning Algorithms
- Single-Query Planning Algorithms
Samping-Based Methods

• Multi-Query Planning Algorithms
  • Cache roadmap for multiple queries
  • Environment is static

• Single-Query Planning Algorithms
  • Recompute roadmap for each query
  • Environment can change across queries (not within)
Rapidly-Exploring Random Trees (RRTs)
Rapidly-Exploring Random Trees (RRTs)
Algorithm 10 Build RRT Algorithm

Input:
  $q_0$: the configuration where the tree is rooted
  $n$ : the number of attempts to expand the tree

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: for $i = 1$ to $n$ do
4:   $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5:   extend RRT ($T, q_{\text{rand}}$)
6: end for
7: return $T$

Algorithm 11 Extend RRT Algorithm

Input:
  $T = (V, E)$: an RRT
  $q$: a configuration toward which the tree $T$ is grown

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

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

Algorithm 10 Build RRT Algorithm

Input:
- $q_0$: the configuration where the tree is rooted
- $n$: the number of attempts to expand the tree

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: for $i = 1$ to $n$ do
4: \quad $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5: \quad extend RRT $(T, q_{\text{rand}})$
6: end for
7: return $T$

Algorithm 11 Extend RRT Algorithm

Input:
- $T = (V, E)$: an RRT
- $q$: a configuration toward which the tree $T$ is grown

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

$q_{\text{near}} \leftarrow$ closest neighbor of $q$ in $T$
2: $q_{\text{new}} \leftarrow$ progress $q_{\text{near}}$ by step-size along the straight line in $Q$ between $q_{\text{near}}$ and $q_{\text{rand}}$
3: if $q_{\text{new}}$ is collision-free then
4: \quad $V \leftarrow V \cup \{q_{\text{new}}\}$
5: \quad $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: \quad return $q_{\text{new}}$
7: end if
8: return NIL
Rapidly-Exploring Random Trees (RRTs)
Algorithm 10 Build RRT Algorithm

Input:

$q_0$: the configuration where the tree is rooted
$n$: the number of attempts to expand the tree

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: for $i = 1$ to $n$ do
4:   $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5:   extend RRT $(T, q_{\text{rand}})$
6: end for
7: return $T$

Algorithm 11 Extend RRT Algorithm

Input:

$T = (V, E)$: an RRT
$q$: a configuration toward which the tree $T$ is grown

Output:

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

1: $q_{\text{near}} \leftarrow$ closest neighbor of $q$ in $T$
$q_{\text{new}} \leftarrow$ progress $q_{\text{near}}$ by step size along the straight line in $Q$ between $q_{\text{near}}$ and $q_{\text{rand}}$
3: if $q_{\text{new}}$ is collision-free then
4:   $V \leftarrow V \cup \{q_{\text{new}}\}$
5:   $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6:   return $q_{\text{new}}$
7: end if
8: return NIL
Rapidly-Exploring Random Trees (RRTs)
Algorithm 10 Build RRT Algorithm

Input:
$q_0$: the configuration where the tree is rooted
$n$: the number of attempts to expand the tree

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: for $i = 1$ to $n$ do
4:   $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
5:   extend RRT ($T$, $q_{\text{rand}}$)
6: end for
7: return $T$

Algorithm 11 Extend RRT Algorithm

Input:
$T = (V, E)$: an RRT
$q$: a configuration toward which the tree $T$ is grown

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

1: $q_{\text{near}} \leftarrow$ closest neighbor of $q$ in $T$
2: $q_{\text{new}} \leftarrow$ progress $q_{\text{near}}$ by step_size along the straight line in $Q$ between $q_{\text{near}}$ and $q_{\text{rand}}$
3: if $q_{\text{new}}$ is collision-free then
4:   $V \leftarrow V \cup \{q_{\text{new}}\}$
5:   $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
6: return $q_{\text{new}}$
7: end if
8: return NIL
Rapidly-Exploring Random Trees (RRTs)

• When to stop?
  • when inside a prespecified goal region
Rapidly-Exploring Random Trees (RRTs)

• When to stop?
  • when inside a prespecified goal region

• How to retrieve the path?
Rapidly-Exploring Random Trees (RRTs)

• When to stop?
  • when inside a prespecified goal region

• How to retrieve the path?
  • retain a pointer to parents
Rapidly-Exploring Random Trees (RRTs)
Rapidly-Exploring Random Trees (RRTs)
Rapidly-Exploring Random Trees (RRTs)
Rapidly-Exploring Random Trees (RRTs)
Intuition behind RRT

- Vertices that are isolated, on the boundary, have higher "visibility" and are more likely to be selected.
Transition-RRTs
Algorithm 1: Transition-based RRT

\begin{algorithm}
\textbf{input} : the configuration space $CS$;
  
  the cost function $c : CS \rightarrow \mathbb{R}_*^+$;

  the root $q_{\text{init}}$ and the goal $q_{\text{goal}}$;

\textbf{output} : the tree $T$;

\begin{algorithmic}
\begin{align*}
\text{begin} & \quad T \leftarrow \text{InitTree}(q_{\text{init}}); \\
\text{while not} & \quad \text{StopCondition}(T, q_{\text{goal}}) \text{ do} \\
& \quad q_{\text{rand}} \leftarrow \text{SampleConf}(CS); \\
& \quad q_{\text{near}} \leftarrow \text{BestNeighbor}(q_{\text{rand}}, T); \\
& \quad \text{if not} \quad \text{Extend}(T, q_{\text{rand}}, q_{\text{near}}, q_{\text{new}}) \text{ Continue;} \\
& \quad \text{if TransitionTest}(c(q_{\text{near}}), c(q_{\text{new}}), d_{\text{near-new}}) \\
& \quad \quad \text{and MinExpandControl}(T, q_{\text{near}}, q_{\text{rand}}) \text{ then} \\
& \quad & \quad \quad \text{AddNewNode}(T, q_{\text{new}}); \\
& \quad & \quad \quad \text{AddNewEdge}(T, q_{\text{near}}, q_{\text{new}}); \\
\text{end} &
\end{align*}
\end{algorithmic}
\end{algorithm}
**Algorithm 1: Transition-based RRT**

```pseudo
input : the configuration space $CS$; 
        the cost function $c : CS \rightarrow \mathbb{R}_+^*$; 
        the root $q_{init}$ and the goal $q_{goal}$; 

output : the tree $T$; 

begin 
    $T \leftarrow$ InitTree($q_{init}$); 
    while not StopCondition($T$, $q_{goal}$) do 
        $q_{rand} \leftarrow$ SampleConf($CS$); 
        $q_{near} \leftarrow$ BestNeighbor($q_{rand}$, $T$); 
        if not Extend($T$, $q_{rand}$, $q_{near}$, $q_{new}$) continue; 
        if TransitionTest($c(q_{near})$, $c(q_{new})$, $d_{near-new}$) 
        and MinExpandControl($T$, $q_{near}$, $q_{rand}$) then 
            AddNewNode($T$, $q_{new}$); 
            AddNewEdge($T$, $q_{near}$, $q_{new}$); 
    end 
end
```
Algorithm 1: Transition-based RRT

\[ p_{ij} = \begin{cases} 
\exp\left(-\frac{\Delta c_{ij}^*}{K*T}\right) & \text{if } \Delta c_{ij}^* > 0 \\
1 & \text{otherwise.}
\end{cases} \]
Transition-RRTs

Keys:
r: toggle rotation
i: place initial position (red) under mouse
g: place goal position (green) under mouse
1: run TRRT, slow speed
2: run TRRT, fast speed
0: Pause TRRT
c: clear tree
Robotic Lime Picking by considering leaves as Permeable Obstacles

Heramb Nemlekar*, Ziang Liu*, Suraj Kothawade, Sherdil Niyaz, Barath Raghavan and Stefanos Nikolaidis

* equal contribution

ICAROS Lab
University of Southern California
Definitions

• A robustly feasible planning problem
Completeness

• When is an algorithm *complete*?
Completeness and Optimality

• When is an algorithm *complete*?
  • If there is a solution, the algorithm will find it, otherwise it will report failure

• Probabilistic Completeness: a *weaker* notion of completeness

• Probabilistic Optimality: a *weaker* notion of optimality
Probabilistic Completeness

• An algorithm ALG is probabilistically complete if, for any robustly feasible motion planning problem defined by
  \[ P = (Q_{free}, q_s, q_g) \]

\[
\lim_{{N \to \infty}} P(\text{ALG returns a solution to } P) = 1
\]

Is RRT probabilistically complete?
Asymptotic Optimality

• We let $Y_{N}^{RRT}$ the cost of the best path from running RRT $N$ times and $c^*$ the cost of the optimal path.
Asymptotic Optimality

- We let $Y_N^{RRT}$ the cost of the best path from running RRT $N$ times and $c^*$ the cost of the optimal path.

$$P\left(\left\{ \lim_{N \to \infty} Y_N^{RRT} > c^* \right\} \right) = 1$$
Anytime RRTs

• Find a suboptimal path quickly, then improve over time.
  • Produce an initial path of cost $C_s$
  • Next run, we stop only if $C'_s < C_s$
Anytime RRTs

• Find a suboptimal path quickly, then improve over time.
  • Produce an initial path of cost $C_s$
  • Next run, we stop only if $C'_s < C_s$

• Idea 1: keep old trees
  • Might converge to a local minimum
  • Might bias future solutions

• Idea 2: build a new tree from scratch
Anytime RRTs

• Find a suboptimal path quickly, then improve over time.
  • Produce an initial path of cost $C_s$
  • Next run, we stop only if $C'_s < C_s$

• Idea 1: keep old trees
  • Might converge to a local minimum
  • Might bias future solutions

• Idea 2: build a new tree from scratch

• Better Ideas: Use a heuristic to consider a subset of nodes of previous nodes that lead to an improved solution
Bidirectional RRTs

- Start from both start node and goal node and merge trees
Bidirectional RRTs

Algorithm 1: IKBiRRT($q_s$, $W$)

1. $T_a$.Init($q_s$); $T_b$.Init(NULL);
2. while TimeRemaining() do
3.     $T_{goal} = \text{GetBackwardTree}(T_a, T_b)$;
4.     if $T_{goal}$.size = 0 or rand(0, 1) $< P_{sample}$ then
5.         AddIKSolutions($T_{goal}$, $W$);
6.     else
7.         $q_{rand} \leftarrow \text{RandConfig}()$;
8.         $q_{near}^{a} \leftarrow \text{NearestNeighbor}(T_a, q_{rand})$;
9.         $q_{reached}^{a} \leftarrow \text{Extend}(T_a, q_{near}^{a}, q_{rand})$;
10.        $q_{near}^{b} \leftarrow \text{NearestNeighbor}(T_b, q_{reached}^{a})$;
11.        $q_{reached}^{b} \leftarrow \text{Extend}(T_b, q_{near}^{b}, q_{reached}^{a})$;
12.        if $q_{reached}^{a} = q_{reached}^{b}$ then
13.            $P \leftarrow \text{ExtractPath}(T_a, q_{reached}^{a}, T_b, q_{reached}^{b})$;
14.            return SmoothPath($P$);
15.        else
16.            Swap($T_a$, $T_b$);
17.        end
18. end
19. return NULL;
Bidirectional RRTs
Shortcutting

• Output of RRTs is almost unusable.

• Simple Shortcutting Algorithm
Shortcutting

• Output of RRTs is almost unusable.

• Simple Shortcutting Algorithm:
  Repeat:
    Select two points randomly
    Attempt to connect them
    If path shortest than previous one, replace path
Special Case: Time-Varying Planning
Time-Varying Planning

• Time-Varying Obstacles $O(t)$

• State-space?
Time-Varying Planning

• Time-Varying Obstacles \( O(t) \)

• State-space: \( x = (q, t) \)

• \( X_{\text{obs}}? \)
Time-Varying Planning

- Time-Varying Obstacles $O(t)$

- State-space: $x = (q, t)$

- $X_{obs} = \{(q, t) \in X | A(q) \cap O(t) \neq \emptyset\}$

- Goal: compute a continuous time-monotonic path
  \[ \tau[0, 1] \rightarrow X_{free} \]
Time-Varying Planning

• Time-Varying Obstacles \( O(t) \)

• State-space: \( x = (q, t) \)

\[ X_{obs} = \{ (q, t) \in X | A(q) \cap O(t) \neq \emptyset \} \]

• Goal: compute a continuous time-monotonic path

\[ \tau[0, 1] \rightarrow X_{free} \]

\[ s_1 \leq s_2 \rightarrow t_1 \leq t_2, (q_1, t_1) = \tau(s_1), (q_2, t_2) = \tau(s_2) \]
Time-Varying Planning Example

- Piecewise-linear motion
Planning in Discrete-Continuous Spaces

CSCI 545 Introduction to Robotics
Instructor: Stefanos Nikolaidis
Modes
Problem Setting

• State space $X = Q \times M$. A state is represented as $x = (q, m)$, where $q \in Q$ and $m \in M$

• An obstacle region $O(m)$ that depends on the mode $m$

• A robotic arm $A(m)$

• $Q_{obs} = \{(q, m) \in X | A(q, m) \cap O(m) \neq \emptyset\}$

• Each state $x$ has a finite action space $U(x)$
Problem Setting

• Each state $x$ has a finite action space $U(x)$

• A mode transition function $f_m : X \times U \rightarrow M$
Modes

Layer $c$
Problem Setting

- Each state $x$ has a finite action space $U(x)$
- A mode transition function $f_m : X \times U \to M$
- A state transition function $f(x, u) = (q, f_m(x, u))$
- An initial state $x_s \in X_{free}$
- A goal region $X_G \in X_{free}$
Goal

• Compute a continuous path: \( \tau : [0, 1] \rightarrow Q_{free} \)

• Compute an action trajectory: \( \sigma : [0, 1] \rightarrow U \)
Example

\[ M = \{ \text{OPEN, CLOSED} \} \]
Example

\[ x_s = (q_s, CLOSED), x_g = (q_g, OPEN) \]
Example

\[ x_s = (q_s, \text{CLOSED}), x_g = (q_g, \text{OPEN}) \]
Example

\[ x_s = (q_s, CLOSED), x_g = (q_g, OPEN) \]
Example

O(OPEN)

O(CLOSED)
Example

O(OPEN)

O(CLOSED)
Example

\[q_g\]

\[O(OPEN)\]

\[q_1\]

\[O(CLOSED)\]

\[q_s\]

\[q_1\]
Example

$O(OPEN)$

$O(CLOSED)$

$q_g$

$q_1$

$q_s$

$q_1$
Special Case: Manipulation Planning
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \) \quad U = \{\text{Grasp, Ungrasp}\}

Part: \( P, Q^P = \mathbb{R}^1 \)
Example

[Diagram showing the process of grasping and ungrasping with an arrow labeled 'Grasp' and another arrow labeled 'Ungrasp' pointing in the opposite direction. The word 'Transfer' is also indicated on the right side of the diagram.]
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \)
Part: \( P, Q^P = \mathbb{R}^1 \)

\[ U = \{\text{Grasp, Ungrasp}\} \]

When can grasp occur?
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \)  \( U = \{\text{Grasp, Ungrasp}\} \)

Part: \( P, Q^P = \mathbb{R}^1 \)  \( q^A = q^P \)
Example

$q^P$

$q^A$
Example

$q^P$

How can we represent the transfer phase?
Example

$q^P$

TRANSFER

$q^A$
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \) \quad U = \{\text{Grasp, Ungrasp}\}

Part: \( P, Q^P = \mathbb{R}^1 \)
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \)  
Part: \( P, Q^P = \mathbb{R}^1 \)
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \)  \( U = \{\text{Grasp, Ungrasp}\} \)

Part: \( P, Q^P = \mathbb{R}^1 \)
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \) \hspace{1cm} U = \{\text{Grasp, Ungrasp}\}

Part: \( P, Q^P = \mathbb{R}^1 \)
Example

\[ M = \{\text{Transit, Transfer}\} \]

Robot: \( A, Q^A = \mathbb{R}^1 \) \hspace{1cm} \text{U} = \{\text{Grasp, Ungrasp}\}

Part: \( P, Q^P = \mathbb{R}^1 \)
Example

\[
\begin{align*}
q^P & \quad q^g \quad q_s \\
q^A &
\end{align*}
\]
Example

\[ q^P \]

\[ q^g \]

\[ q_s \]
Example

$q^P$

$q^g$

$q^s$

$q^A$
Manipulation Planning

• Sample configurations where mode transition can occur (valid grasp configuration with resting object)
Manipulation Planning

• Compute sub-roadmaps
Manipulation Planning

- Attempt to connect each pair of vertices in the two roadmaps with transit or transfer paths
Manipulation Planning
Manipulation Planning

- Connect to goal configuration
Manipulation Planning

• Search graph