16-782
Planning & Decision-making in Robotics

Planning Representations/Search Algorithms:
RRT, RRT-Connect, RRT*

Maxim Likhachev
Robotics Institute
Carnegie Mellon University
Probabilistic Roadmaps (PRMs)

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

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

- **Great for problems where a planner has to plan many times for different start/goal pairs (step 1 needs to be done only once)**
- **Not so great for single shot planning**
No preprocessing step: starting with the initial configuration $q_I$ build the graph (actually, tree) until the goal configuration $g_G$ is part of it.
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

\begin{verbatim}
BUILD_RRT(q_{init})
1   \mathcal{T}.init(q_{init});
2   \textbf{for } k = 1 \textbf{ to } K \textbf{ do}
3       q_{rand} \leftarrow \text{RANDOM\_CONFIG}();
4       \text{EXTEND}(\mathcal{T}, q_{rand});
5   \textbf{Return } \mathcal{T}
\end{verbatim}

\begin{verbatim}
EXTEND(\mathcal{T}, q)
1   q_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(q, \mathcal{T});
2   \textbf{if } \text{NEW\_CONFIG}(q, q_{near}, q_{new}) \textbf{ then}
3       \mathcal{T}.add\_vertex(q_{new});
4       \mathcal{T}.add\_edge(q_{near}, q_{new});
5   \textbf{if } q_{new} = q \textbf{ then}
6       \textbf{Return } Reached;
7   \textbf{else}
8       \textbf{Return } Advanced;
9   \textbf{Return } Trapped;
\end{verbatim}

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle

EXTEND operation
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

**BUILD_RRT(qinit)**
1. $\mathcal{T}.\text{init}(q_{\text{init}})$;
2. for $k = 1$ to $K$ do
3. $q_{\text{rand}} \leftarrow \text{RANDOM_CONFIG}()$;
4. $\text{EXTEND}(\mathcal{T}, q_{\text{rand}})$;
5. Return $\mathcal{T}$

**EXTEND(\mathcal{T}, q)**
1. $q_{\text{near}} \leftarrow \text{NEAREST_NEIGHBOR}(q, \mathcal{T})$;
2. if $\text{NEW_CONFIG}(q, q_{\text{near}}, q_{\text{new}})$ then
3. $\mathcal{T}.\text{add}\_\text{vertex}(q_{\text{new}})$;
4. $\mathcal{T}.\text{add}\_\text{edge}(q_{\text{near}}, q_{\text{new}})$;
5. if $q_{\text{new}} = q$ then
6. Return Reached;
7. else
8. Return Advanced;
9. Return Trapped;

Path to the goal is a path in the tree from $q_{\text{init}}$ to the vertex closest to goal
Selects closest vertex in the tree
Moves by at most $\epsilon$ from $q_{\text{near}}$ towards $q$

EXTEND operation

Borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

- RRT provides uniform coverage of space

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

- RRT provides uniform coverage of space

Pros/cons?

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

- Alternatively, the growth is always biased by the largest unexplored region.

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Rapidly Exploring Random Trees (RRTs) [LaValle, ’98]

- Alternatively, the growth is always biased by the largest unexplored region

*Under what assumptions?*

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Bi-directional growth of the tree

+ 

relax the \( \varepsilon \) constraint on the growth of the tree
**RRT-Connect [Kuffner & LaValle, ‘00]**

---

RRT\_CONNECT\_PLANNER\((q_{init}, q_{goal})\)
\[
\begin{align*}
1 & \quad T_a.\text{init}(q_{init}); \ T_b.\text{init}(q_{goal}); \\
2 & \quad \text{for } k = 1 \text{ to } K \text{ do} \\
3 & \quad \quad q_{\text{rand}} \leftarrow \text{RANDOM\_CONFIG}(); \\
4 & \quad \quad \text{if not } (\text{EXTEND}(T_a, q_{\text{rand}}) = \text{Trapped}) \text{ then} \\
5 & \quad \quad \quad \text{if } (\text{CONNECT}(T_b, q_{new}) = \text{Reached}) \text{ then} \\
6 & \quad \quad \quad \quad \text{Return PATH}(T_a, T_b); \\
7 & \quad \quad \text{SWAP}(T_a, T_b); \\
8 & \quad \text{Return Failure}
\end{align*}
\]

CONNECT\((T, q)\)
\[
\begin{align*}
1 & \quad \text{repeat} \\
2 & \quad \quad S \leftarrow \text{EXTEND}(T, q); \\
3 & \quad \text{until not } (S = \text{Advanced}) \\
4 & \quad \text{Return } S;
\end{align*}
\]

---

*borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle*
RRT-Connect [Kuffner & LaValle, ‘00]

RRT.CONNECT.PLANNER(q_{init}, q_{goal})
1 \(T_a\).init(q_{init}); \(T_b\).init(q_{goal});
2 for \(k = 1\) to \(K\) do
3 \(q_{rand} \leftarrow \) RANDOM\_CONFIG();
4 if not (EXTEND(\(T_a, q_{rand}\)) = Trapped) then
5 \( \) if (CONNECT(\(T_b, q_{new}\)) = Reached) then
6 \( \) Return PATH(\(T_a, T_b\));
7 \( \) SWAP(\(T_a, T_b\));
8 Return Failure

CONNECT(\(T, q\))
1 repeat
2 \(S \leftarrow \) EXTEND(\(T, q\));
3 until not (\(S = Advanced\))
4 Return \(S\);

Why swap the trees?

CONNECT function grows the tree by more than just one \(\epsilon\)

tries to grow \(T_b\) to \(q_{new}\) that was just added to \(T_a\)

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
RRT-Connect [Kuffner & LaValle, ‘00]

• For any \( q \in C_{\text{free}} \), \( \lim_{k \to \infty} P[d(q) < \varepsilon] = 1 \), where \( d(q) \) is a distance from configuration \( q \) to the closest vertex in the tree, and assuming \( C_{\text{free}} \) is connected, bounded and open.

• RRT-Connect is probabilistically complete: \( \text{as } \# \text{ of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists} \)
RRT-Connect [Kuffner & LaValle, ‘00]

• For any $q \in C_{\text{free}}$, $\lim_{k \to \infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration $q$ to the closest vertex in the tree, and assuming $C_{\text{free}}$ is connected, bounded and open.

• RRT-Connect is probabilistically complete: as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists.

Is RRT-Connect asymptotically (as $k \to \infty$) optimal?

No, more on this later.
RRT-Connect [Kuffner & LaValle, ‘00]

- For any $q \in C_{\text{free}}$, $\lim_{k \to \infty} P[d(q) < \varepsilon] = 1$, where $d(q)$ is a distance from configuration $q$ to the closest vertex in the tree, and assuming $C_{\text{free}}$ is connected, bounded and open.

- RRT-Connect is probabilistically complete: as # of samples approaches infinity, the algorithm is guaranteed to find a solution if one exists.

Applicability of RRT vs. RRT-Connect to kinodynamic planning?
Sampling-based approaches

Typical setup:

- Run PRM/RRT/RRT-Connect/…

- Post-process the generated solution to make it more optimal

An important but often time-consuming step

Could also be highly non-trivial
Post-processing

Any ideas how to post-process it?

Consider this path generated by RRT or PRM or A* on a grid-based graph:
Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

\[
\text{NewPath=}[]; \quad \text{P=start point, } P1 = \text{point } P+1 \text{ along the path}
\]

\[
\text{while } P \text{ !}= \text{goal point}
\]

\[
\text{while line segment } [P,P1+1] \text{ is obstacle-free AND } P1+1 < \text{goal point}
\]

\[
P1 = \text{point } P1+1 \text{ along the path;}
\]

\[
\text{NewPath}+=[P,P1]; \quad P = P1; \quad P1 = \text{point } P+1 \text{ along the path;}
\]
Short-cutting a path consisting of a series of points

\( \text{NewPath} = []; P = \text{start point}, \ P1 = \text{point } P+1 \text{ along the path} \)

while \( P \neq \text{goal point} \)

while line segment \([P,P1+1]\) is obstacle-free AND \( P1+1 < \text{goal point} \)

\( P1 = \text{point } P1+1 \text{ along the path}; \)
\( \text{NewPath} += [P,P1]; P = P1; P1 = \text{point } P+1 \text{ along the path}; \)
Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

  \( \text{NewPath} = []; P = \text{start point}, P1 = \text{point } P+1 \text{ along the path} \)
  
  while \( P \neq \text{goal point} \)
    
    while line segment \([P,P1+1]\) is obstacle-free AND \( P1+1 < \text{goal point} \)
      
      \( P1 = \text{point } P1+1 \text{ along the path;} \)
      
      \( \text{NewPath} += [P,P1]; P = P1; P1 = \text{point } P+1 \text{ along the path;} \)
Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points
  
  \[\text{NewPath}=[]; \ P=\text{start point}, \ P1 = \text{point } P+1 \text{ along the path}\]
  
  while \(P \neq \text{goal point}\)
  
  while line segment \([P,P1+1]\) is obstacle-free AND \(P1+1 < \text{goal point}\)
  
  \(P1 = \text{point } P1+1 \text{ along the path};\)
  
  \(\text{NewPath}+= [P,P1]; \ P = P1; \ P1 = \text{point } P+1 \text{ along the path};\)
Simple Post-processing via Short-cutting

• Short-cutting a path consisting of a series of points

\[ \text{NewPath} = []; \ P = \text{start point}, \ P1 = \text{point } P+1 \text{ along the path} \]

\[ \text{while } P \neq \text{goal point} \]

\[ \text{while line segment } [P, P1+1] \text{ is obstacle-free AND } P1+1 < \text{goal point} \]

\[ P1 = \text{point } P1+1 \text{ along the path}; \]

\[ \text{NewPath} += [P, P1]; \ P = P1; \ P1 = \text{point } P+1 \text{ along the path}; \]
Simple Post-processing via Short-cutting

- Short-cutting a path consisting of a series of points

$NewPath=[];\ P=start\ point,\ P1 = point\ P+1\ along\ the\ path$

while $P\ != goal\ point$

while line segment $[P,P1+1]$ is obstacle-free AND $P1+1 < goal\ point$

$P1 = point\ P1+1\ along\ the\ path;$

$NewPath+= [P,P1];\ P = P1;\ P1 = point\ P+1\ along\ the\ path;$
Examples of RRT in action

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Examples of RRT in action

borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Examples of RRT in action

 borrowed from “RRT-Connect: An Efficient Approach to Single-Query Path Planning” paper by J. Kuffner & S. LaValle
Examples of RRT

5DOF kinodynamic planning for a car

borrowed from “Rapidly-Exploring Random Trees: A new tool for Path Planning” paper by S. LaValle
PRMs vs. RRTs

- PRMs construct a roadmap and then searches it for a solution whenever $q_I, g_G$ are given
  - well-suited for repeated planning in between different pairs of $q_I, g_G$ (multiple queries)

- RRTs construct a tree for a given $q_I, q_G$ until the tree has a solution
  - well-suited for single-shot planning in between a single pair of $q_I, g_G$ (single query)
  - There exist extensions of RRTs that try to reuse a previously constructed tree when replanning in response to map updates
RRTs vs A*-based planning

- **RRTs:**
  - sparse exploration, usually little memory and computations required, works well in high-D
  - solutions can be highly sub-optimal, requires post-processing, which in some cases can be very hard to do, the solution is still restricted to the same homotopic class
RRTs vs A*-based planning

- RRTs:
  - does not incorporate a (potentially complex) cost function
  - there exist versions (e.g., RRT*) that try to incorporate the cost function and converge to a provably least-cost solution in the limit of samples (but typically computationally more expensive than RRT)
RRTs vs A*-based planning

• A* and weighted A* (wA*):
  – returns a solution with optimality (or sub-optimality) guarantees with respect to the discretization used
  – explicitly minimizes a cost function
  – requires a thorough exploration of the state-space resulting in high memory and computational requirements
Sampling in RRTs

**Uniform:** $q_{rand}$ is a random sample in $C_{free}$

**Goal-biased:** with a probability $(1-P_g)$, $q_{rand}$ is chosen as a random sample in $C_{free}$, with probability $P_g$, $q_{rand}$ is set to $g_F$
Sampling in RRTs

- **Uniform**: $q_{\text{rand}}$ is a random sample in $C_{\text{free}}$

- **Goal-biased**: with a probability $(1 - P_g)$, $q_{\text{rand}}$ is chosen as a random sample in $C_{\text{free}}$, with probability $P_g$, $q_{\text{rand}}$ is set to $g_G$

Very useful!
RRT* [Karaman & Frazzoli, ‘06]

RRT

+ “re-wiring of nodes”
Properties of RRT again…

Is RRT asymptotically (in the limit of the number of samples) complete?

Is RRT asymptotically (in the limit of the number of samples) optimal?

Why?
**RRT* [Karaman & Frazzoli, ‘06]**

Main loop (same as in RRT):

1. $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
2. while $i < N$ do
   3. $G \leftarrow (V, E)$;
   4. $x_{\text{rand}} \leftarrow \text{Sample}(i)$; $i \leftarrow i + 1$;
   5. $(V, E) \leftarrow \text{Extend}(G, x_{\text{rand}})$;

Extend($G, x$) (same as in RRT + “re-wiring”):

1. $V' \leftarrow V$; $E' \leftarrow E$;
2. $x_{\text{nearest}} \leftarrow \text{Nearest}(G, x)$;
3. $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x)$;
4. if ObstacleFree($x_{\text{nearest}}, x_{\text{new}}$) then
   5. $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
   6. $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
   7. $X_{\text{near}} \leftarrow \text{Near}(G, x_{\text{new}}, |V|)$;
   8. for all $x_{\text{near}} \in X_{\text{near}}$ do
      9. if ObstacleFree($x_{\text{near}}, x_{\text{new}}$) then
         10. $c' \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}))$;
         11. if $c' < \text{Cost}(x_{\text{new}})$ then
            12. $x_{\text{min}} \leftarrow x_{\text{near}}$;
   13. $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
   14. for all $x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ do
      15. if ObstacleFree($x_{\text{new}}, x_{\text{near}}$) and
         Cost($x_{\text{near}}$) > Cost($x_{\text{new}}$) + $c(\text{Line}(x_{\text{new}}, x_{\text{near}}))$
         then
            16. $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$;
            17. $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
            18. $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
18. return $G' = (V', E')$

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
RRT\* [Karaman & Frazzoli, ‘06]

Main loop (same as in RRT):

1. $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset; i \leftarrow 0;
2. \text{while } i < N \text{ do}
   3. \quad G \leftarrow (V,E);
   4. \quad x_{\text{rand}} \leftarrow \text{Sample}(i); i \leftarrow i + 1;
   5. \quad (V,E) \leftarrow \text{Extend}(G,x_{\text{rand}});

Extend(G,x) (same as in RRT + “re-wiring”):

1. $V' \leftarrow V; E' \leftarrow E$
2. $x_{\text{nearest}} \leftarrow \text{Nearest}(G,x)$
3. $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}},x)$
4. \text{if } \text{ObstacleFree}(x_{\text{nearest}},x_{\text{new}}) \text{ then}
   5. \quad $V' \leftarrow V' \cup \{x_{\text{new}}\}$
   6. \quad $x_{\text{min}} \leftarrow x_{\text{nearest}}$
   7. \quad $X_{\text{near}} \leftarrow \text{Near}(G,x_{\text{new}},|V|)$
   8. \quad \text{for all } x_{\text{near}} \in X_{\text{near}} \text{ do}
      9. \quad \quad \text{if } \text{ObstacleFree}(x_{\text{near}},x_{\text{new}}) \text{ then}
         10. \quad \quad \quad c' \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}},x_{\text{new}}))$
         11. \quad \quad \quad \text{if } c' < \text{Cost}(x_{\text{new}}) \text{ then}
             12. \quad \quad \quad \quad x_{\text{min}} \leftarrow x_{\text{near}}$
   13. \quad $E' \leftarrow E' \cup \{(x_{\text{min}},x_{\text{new}})\}$
   14. \quad \text{for all } x_{\text{near}} \in X_{\text{near}} \setminus \{x_{\text{min}}\} \text{ do}
      15. \quad \quad \text{if } \text{ObstacleFree}(x_{\text{new}},x_{\text{near}}) \text{ and }
         \quad \quad \quad \text{Cost}(x_{\text{near}}) > \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}},x_{\text{near}}))
         \quad \quad \quad \text{then}
             16. \quad \quad \quad x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$
             17. \quad \quad \quad $E' \leftarrow E' \setminus \{(x_{\text{parent}},x_{\text{near}})\}$
             18. \quad \quad \quad $E' \leftarrow E' \cup \{(x_{\text{new}},x_{\text{near}})\}$
   18. return $G' = (V',E')$

Re-wiring:
Checking if we can improve (re-wire) the cost of other nodes near the new node $x_{\text{new}}$

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
RRT* [Karaman & Frazzoli, ’06]

Main loop (same as in RRT, but has “re-wiring”):

\[ X_{\text{near}}: \text{set of all vertices } v \text{ in } V \text{ s.t. they lie within radius } r \text{ from } x_{\text{new}}, \text{ where} \]

\[ r = \min\left(\left(\frac{\gamma \log |V|}{\delta} \right)^{1/d}, \ 8\right), \]

\( d \) – dimensionality of space, \( \delta \) – volume of unit hyperball, \( \gamma \) – user defined constant

Re-wiring:
Checking if we can improve (re-wire) the cost of other nodes near the new node \( x_{\text{new}} \)

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
RRT* [Karaman & Frazzoli, ‘06]

Main loop (same as in RRT + "re-wiring"):

\[ X_{\text{near}} \] set of all vertices \( v \) in \( V \) s.t. they lie within radius \( r \) from \( x_{\text{new}} \), where

\[
r = \min\left(\left(\frac{\gamma \log|V|}{\delta |V|}\right)^{1/d}, \varepsilon\right),
\]

\( d \) – dimensionality of space, \( \delta \) – volume of unit hyperball, \( \gamma \) – user defined constant

---

RRT* (unlike RRT) is asymptotically optimal: converges to an optimal solution in the limit of the number of samples

---

Checking if we can improve (re-wire) the cost of other nodes near the new node \( x_{\text{new}} \)

---

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
RRT vs RRT*

The growth of the RRT tree over time & its effect on the solution

The growth of the RRT* tree over time & its effect on the solution

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
RRT vs RRT*

The growth of the RRT tree over time & its effect on the solution

The growth of the RRT* tree over time & its effect on the solution

Any downsides to RRT* as compared to RRT?

borrowed from “Incremental Sampling-based Algorithms for Optimal Motion Planning” paper by S. Karaman & E. Frazzoli
What You Should Know…

- Pros and Cons of RRT, PRM, RRT-Connect, RRT*
- How RRT, RRT-Connect and RRT* operate
- What guarantees RRT/RRT* provide
- Simple shortcutting algorithm