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

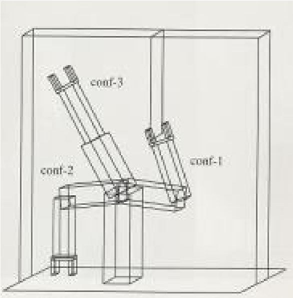
UC San Diego

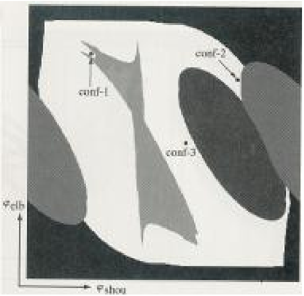
JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

Search-based vs Sampling-based Planning

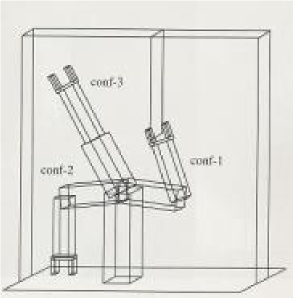
- Search-based planning:
  - Generates a systematic discrete representation (graph) of C_{free}
  - Searches the representation for a path guaranteeing to find one if it exists (resolution complete)
  - Can interleave the representation construction with the search, i.e., adds nodes only when necessary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

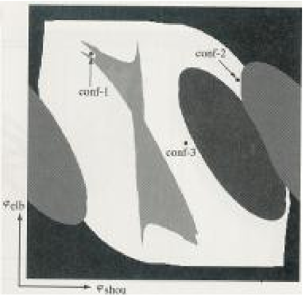


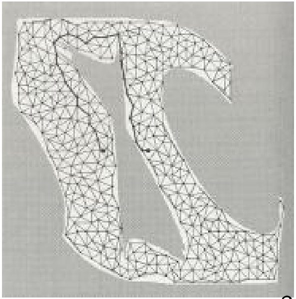


Search-based vs. Sampling-based Planning

- 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 $\rightarrow \infty$ (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: $x_\tau \in C_{free}$
- **Path**: a continuous function $Q : [0,1] \to C$; Set of all paths: $\mathbb{Q}$
- **Feasible path**: a continuous function $Q : [0,1] \to C_{free}$ such that $Q(0) = x_s$ and $Q(1) = x_\tau$; Set of all feasible paths: $\mathbb{Q}_{s,\tau}$
- **Motion Planning Problem** Given a path planning problem (C_{free}, x_s, x_τ) and a cost function $J : \mathbb{Q} \to \mathbb{R}_{\geq 0}$, find a feasible path Q^* such that:

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

  Report failure if no such path exists.

Primitive Procedures for Sampling-based Motion Planning

- $\textsc{Sample}$: returns iid samples from C
- $\textsc{SampleFree}$: returns iid samples from C_{free}
- $\textsc{Nearest}$: 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:

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

- $\textsc{Near}$: 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:

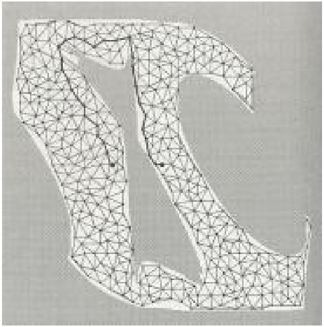
$$\textsc{Near}((V, E), x, r) := \{v \in V \mid \|x - v\| \leq r\}$$

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

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

- $\textsc{CollisionFree}$: given points $x, y \in C$, returns $\textsc{True}$ if the line segment between x and y lies in C_{free} and $\textsc{False}$ otherwise.

Probabilistic Roadmap (PRM)



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

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

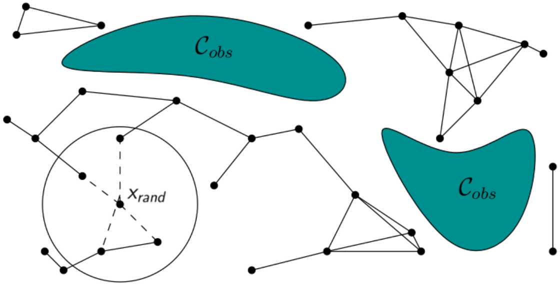
Step 2. **Query Phase**: Given a start configuration x_s and goal configuration x_τ, connect them to the roadmap G using a local planner, then search the augmented roadmap for a shortest path from x_s to x_τ

- **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)

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
for $i = 1, \ldots, n$ **do**
    $x_{rand} \leftarrow \textsc{SampleFree}()$
    $V \leftarrow V \cup \{x_{rand}\}$
    for $x \in \textsc{Near}((V, E), x_{rand}, r)$ **do** ▷ May use k nearest vertices
        if (**not** G.same_component(x_{rand}, x)) **and** $\textsc{CollisionFree}(x_{rand}, x)$ **then**
            $E \leftarrow E \cup \{(x_{rand}, x), (x, x_{rand})\}$
return $G = (V, E)$



Optimal Probabilistic Roadmap

- S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," IJRR, 2010.

- 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 $log(n)$:

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

Algorithm 2 PRM*

1: $V \leftarrow \{x_s\} \cup \{\textsc{SampleFree}()\}_{i=1}^{n}$; $E \leftarrow \emptyset$
2: **for** $v \in V$ **do**
3:     **for** $x \in \textsc{Near}((V, E), v, r^*) \setminus \{v\}$ **do**
4:         **if** $\textsc{CollisionFree}(v, x)$ **then**
5:             $E \leftarrow E \cup \{(v, x), (x, v)\}$
6: **return** $G = (V, E)$

PRM vs RRT

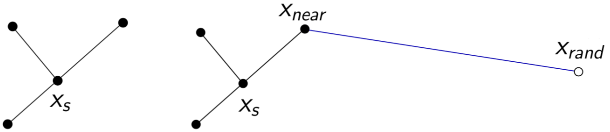
- **PRM**: a graph constructed from random samples. It can be search for a path whenever a start node x_s and goal node x_T are specified. PRMs are well-suited for repeated planning between different pairs of x_s and x_T (*multiple queries*)

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

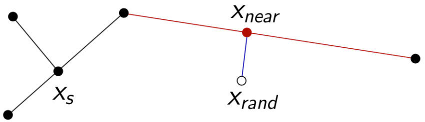
- **Rapidly Exploring Random Tree** (RRT):
  - One of the most popular planning techniques
  - Introduced by Steven LaValle in 1998
  - Many, many, many extensions and variants (articulated robots, kinematics, dynamics, differential constraints)
  - There exist extensions of RRTs that try to reuse a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

Rapidly Exploring Random Tree (RRT)

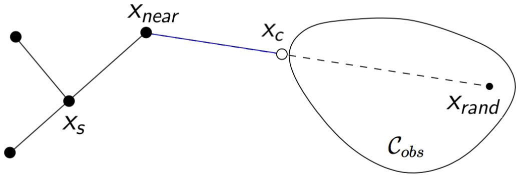
- Sample a new configuration x_{rand}, find the nearest neighbor x_{near} in G and connect them:



- If the nearest point x_{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)

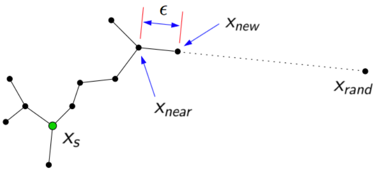
- What about the goal? Occasionally (e.g., every 100 iterations) add the goal configuration x_T and see if it gets connected to the tree
- RRT can be implemented in the original workspace (need to do collision checking) or in configuration space
- Challenges with a C-Space implementation:
  - What distance 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 ϵ 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: $\quad x_{rand} \leftarrow \textsc{SampleFree}()$
4: $\quad x_{nearest} \leftarrow \textsc{Nearest}((V, E), x_{rand})$
5: $\quad x_{new} \leftarrow \textsc{Steer}(x_{nearest}, x_{rand})$
6: $\quad$ **if** $\textsc{CollisionFree}(x_{nearest}, x_{new})$ **then**
7: $\quad\quad V \leftarrow V \cup \{x_{new}\}$; $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
8: **return** $G = (V, E)$

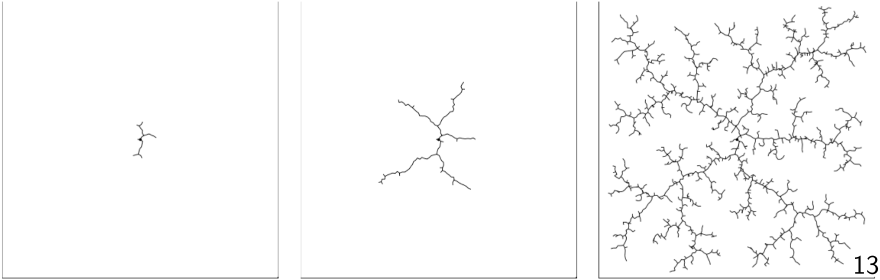


Rapidly Exploring Random Tree (RRT)

- RRT without ϵ (called Rapidly Exploring Dense Tree (RDT)):

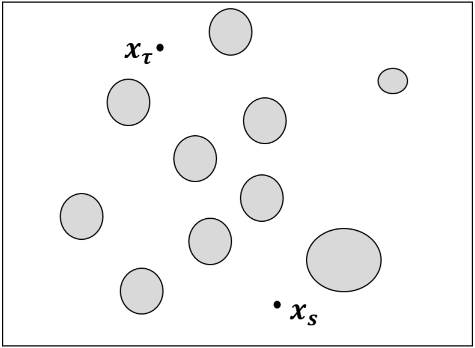
45 iterations   2345 iterations

- RRT with ϵ



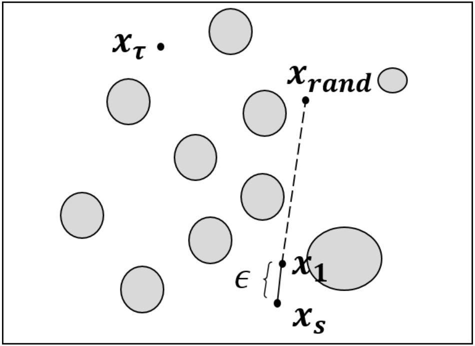
Example: RRT Algorithm

- Start node x_s
- Goal node x_τ
- Gray obstacles



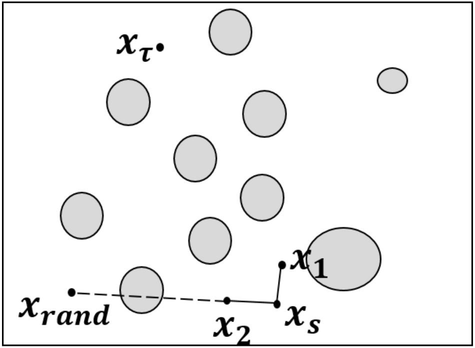
Example: RRT Algorithm

- Sample x_{rand} in the workspace
- Steer from x_s towards x_{rand} by a fixed distance ϵ to get x_1
- If the segment from x_s to x_1 is collision-free, insert x_1 into the tree



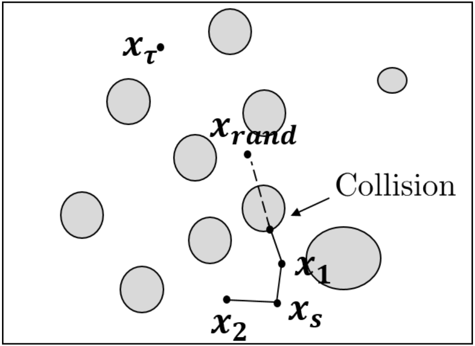
Example: RRT Algorithm

- 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 ϵ to get x_2
- If the segment from x_{near} to x_2 is collision-free, insert x_2 into the tree



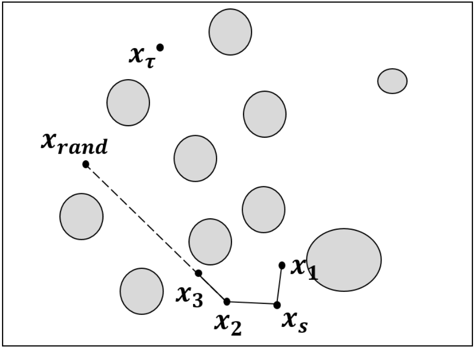
Example: RRT Algorithm

- Sample x_{rand} in the workspace
- Find the closest node x_{near} to x_{rand}
- Steer from x_{near} towards x_{rand} by a fixed distance ϵ to get x_3
- If the segment from x_{near} to x_3 is collision-free, insert x_3 into the tree



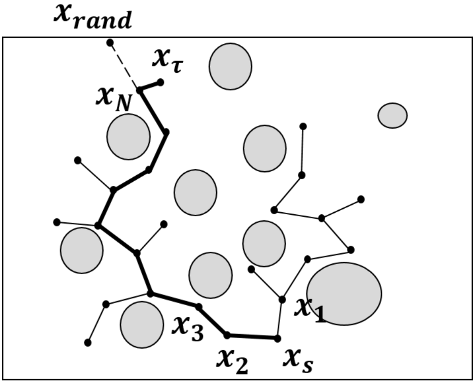
Example: RRT Algorithm

- 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 ϵ to get x_3
- If the segment from x_{near} to x_3 is collision-free, insert x_3 into the tree



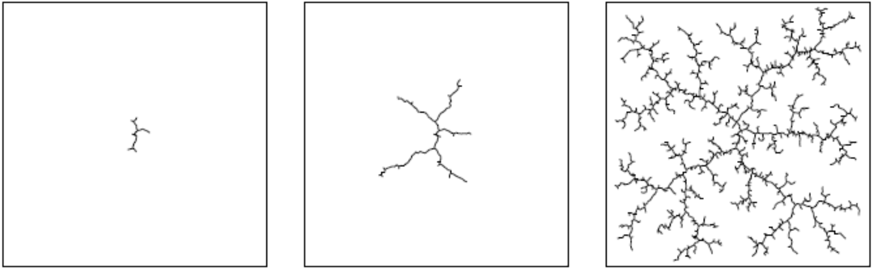
Example: RRT Algorithm

- Continue until a node that is a distance ϵ from the goal is generated
- Either terminate the algorithm or search for additional feasible paths

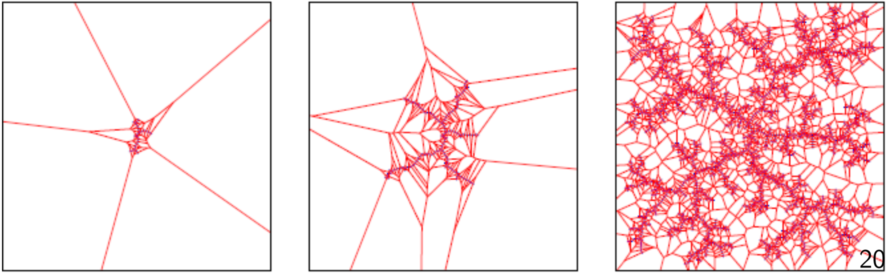


Sampling in RRTs

- The vanilla RRT algorithm provides uniform coverage of space

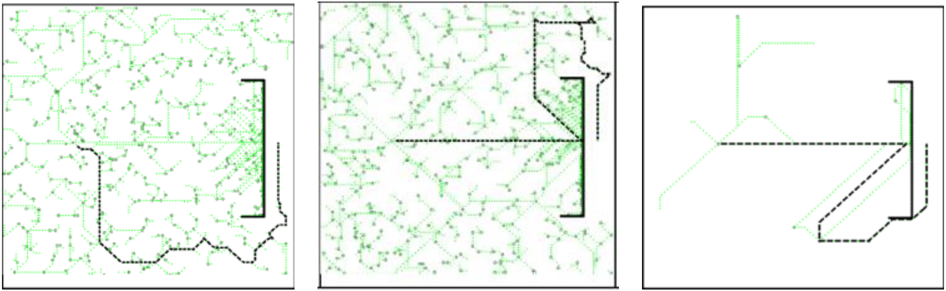


- Alternatively, the growth may be biased by the largest Voronoi region



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, $x_{rand} = x_T$

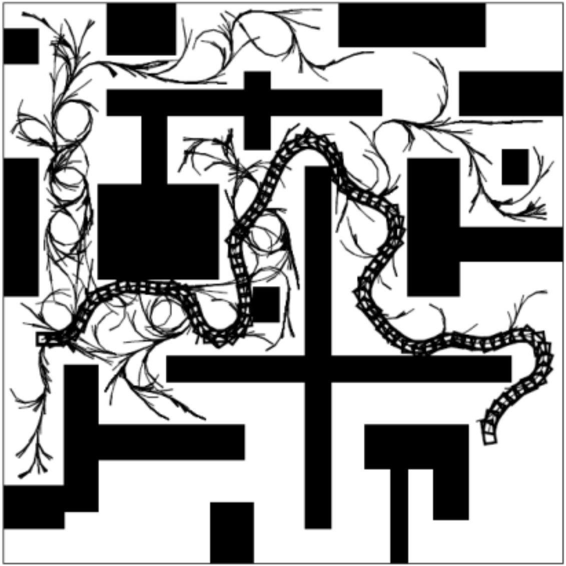


(a) $p_g = 0$ (b) $p_g = 0.1$ (c) $p_g = 0.5$

Handling Robot Dynamics with Steer()

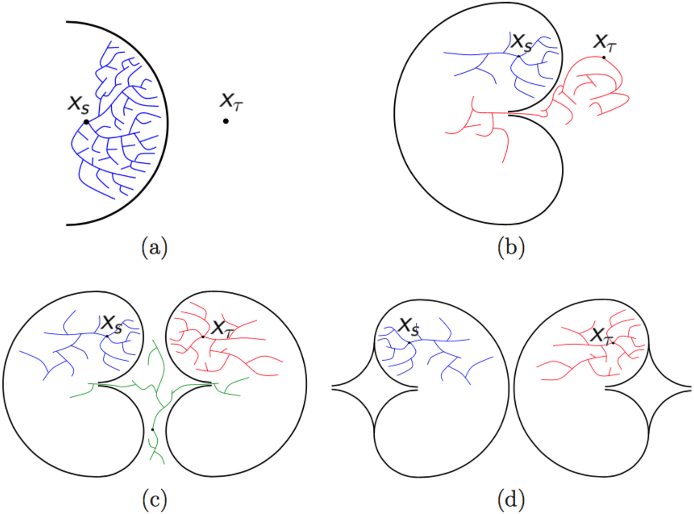
- Steer() extends the tree towards a given random sample x_{rand}
- Consider a car-like robot with non-holonomic constraints (can't slide sideways) in $SE(2)$. Obtaining a feasible path from $x_{rand} = (0, 0, 90^\circ)$ to $x_{near} = (1, 0, 90^\circ)$ is as hard as the original problem
- Steer() resolves this by not requiring the motion to get all the way to x_{rand}. We just apply the best control input for a fixed duration to obtain x_{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.



Bi-directional RRT

Algorithm 4 Bi-directional RRT

$V_a \leftarrow \{x_s\};\ E_a \leftarrow \emptyset;\ V_b \leftarrow \{x_T\};\ E_b \leftarrow \emptyset$
for $i = 1 \ldots n$ **do**
    $x_{rand} \leftarrow \textsc{SampleFree}()$
    $x_{nearest} \leftarrow \textsc{Nearest}((V_a, E_a), x_{rand})$
    $x_c \leftarrow \textsc{Steer}(x_{nearest}, x_{rand})$
    if $x_c \neq x_{nearest}$ **then**
        $V_a \leftarrow V_a \cup \{x_c\};\ E_a \leftarrow \{(x_{nearest}, x_c), (x_c, x_{nearest})\}$
        $x'_{nearest} \leftarrow \textsc{Nearest}((V_b, E_b), x_c)$
        $x'_c \leftarrow \textsc{Steer}(x'_{nearest}, x_c)$
        if $x'_c \neq x'_{nearest}$ **then**
            $V_b \leftarrow V_b \cup \{x'_c\};\ E_b \leftarrow \{(x'_{nearest}, x'_c), (x'_c, x'_{nearest})\}$
        if $x'_c = x_c$ **then return** SOLUTION
    if $|V_b| < |V_a|$ **then** $\textsc{Swap}((V_a, E_a), (V_b, E_b))$
return FAILURE

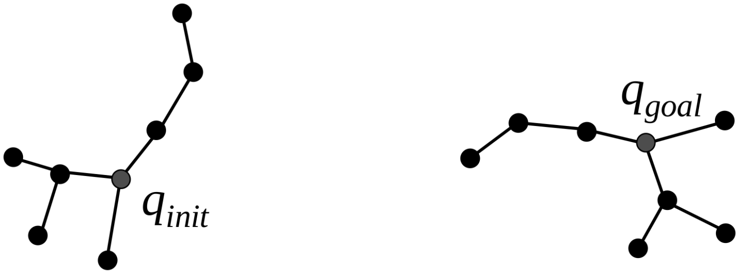
RRT-Connect (J. Kuffner and S. LaValle, ICRA, 2000)

- Bi-directional tree + relax the ϵ constraint on tree growth

Algorithm 5 RRT-Connect

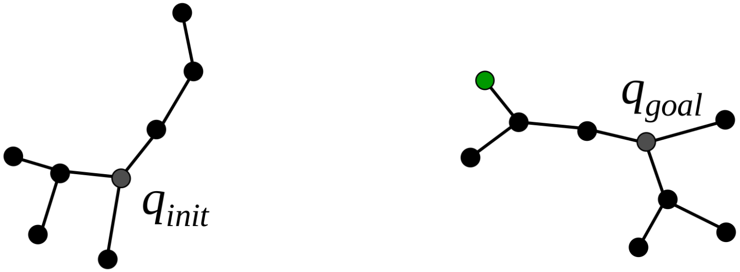
$V_a \leftarrow \{x_s\}$; $E_a \leftarrow \emptyset$; $V_b \leftarrow \{x_\tau\}$; $E_b \leftarrow \emptyset$
for $i = 1 \ldots n$ **do**
    $x_{rand} \leftarrow \textsc{SampleFree}()$
    if not $\textsc{Extend}((V_a, E_a), x_{rand})$ = Trapped **then**
        if $\textsc{Connect}((V_b, E_b), x_{new})$ = Reached **then** ▷ x_{new} was just added to (V_a, E_a)
            return $\textsc{Path}((V_a, E_a), (V_b, E_b))$
    $\textsc{Swap}((V_a, E_a), (V_b, E_b))$
return Failure
function $\textsc{Extend}((V, E), x)$
    $x_{nearest} \leftarrow \textsc{Nearest}((V, E), x)$
    $x_{new} \leftarrow \textsc{Steer}(x_{nearest}, x)$
    if $\textsc{CollisionFree}(x_{near}, x_{new})$ **then**
        $V \leftarrow \{x_{new}\}$; $E \leftarrow \{(x_{near}, x_{new}), (x_{new}, x_{near})\}$
        if $x_{new} = x$ **then return** Reached **else return** Advanced
    return Trapped
function $\textsc{Connect}((V, E), x)$
    repeat $status \leftarrow \textsc{Extend}((V, E), x)$ **until** $status \neq Advanced$
    return $status$

Example: Single RRT-Connect Iteration



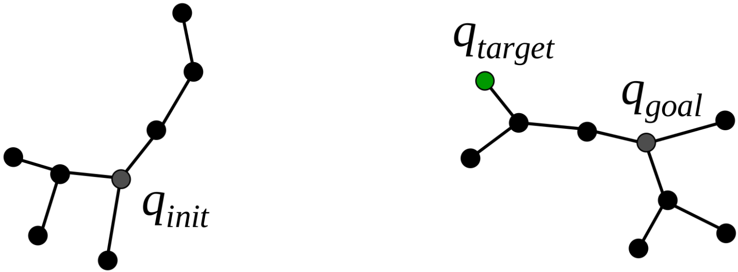
Example: Single RRT-Connect Iteration

- One tree is grown to a random target



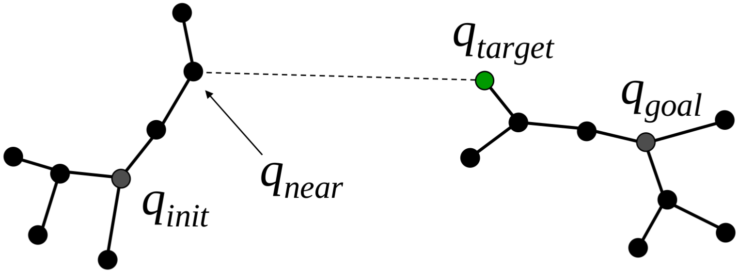
Example: Single RRT-Connect Iteration

- The new node becomes a target for the other tree



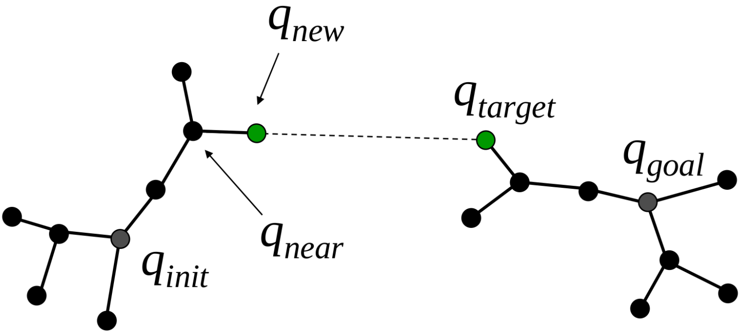
Example: Single RRT-Connect Iteration

- Determine the nearest node to the target



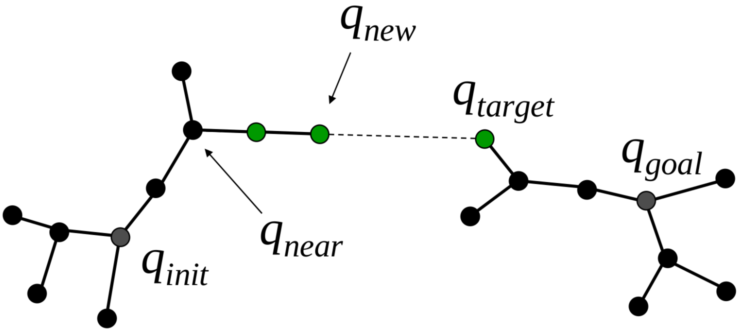
Example: Single RRT-Connect Iteration

- Try to add a new collision-free branch



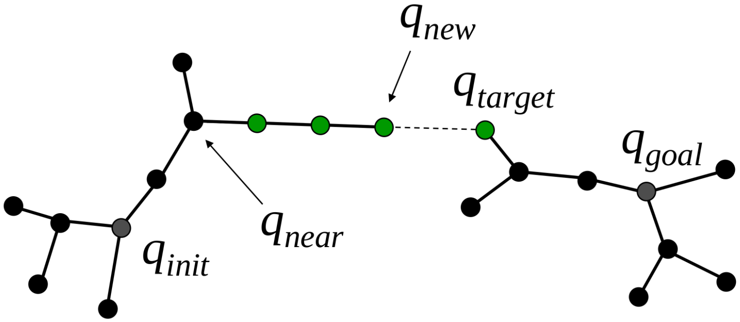
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



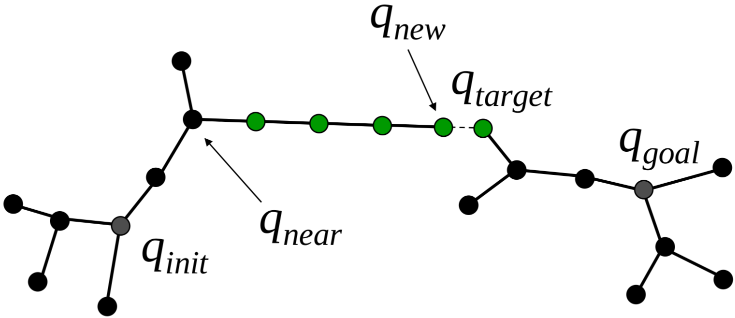
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



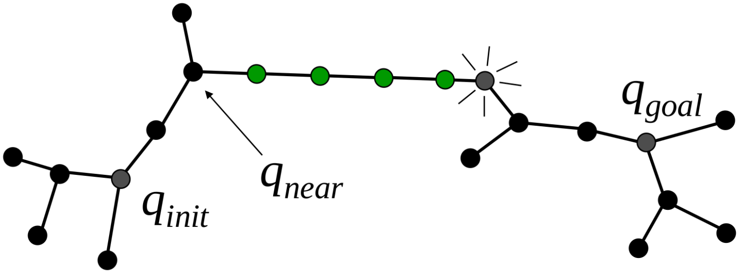
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



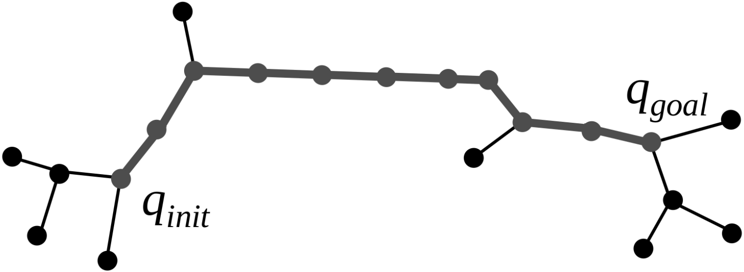
Example: Single RRT-Connect Iteration

- If the branch reaches all the way to the target, a feasible path is found!

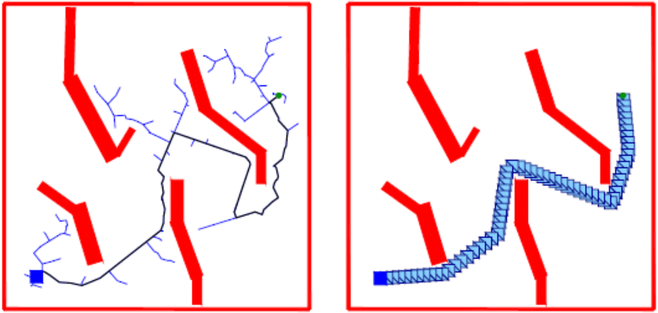


Example: Single RRT-Connect Iteration

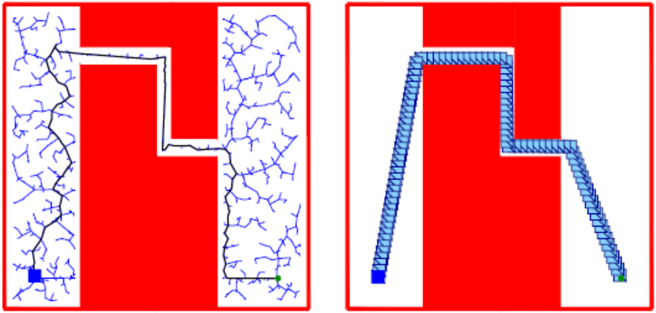
- If the branch reaches all the way to the target, a feasible path is found!



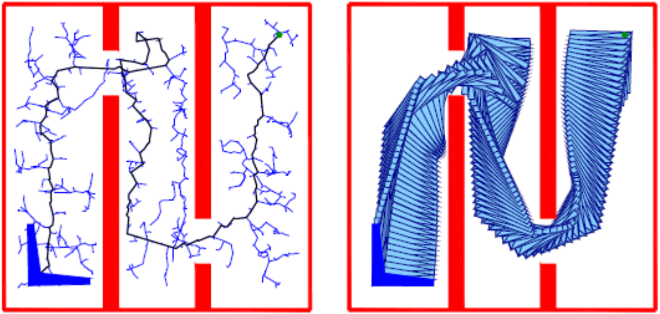
Example: RRT-Connect



Example: RRT-Connect



Example: RRT-Connect

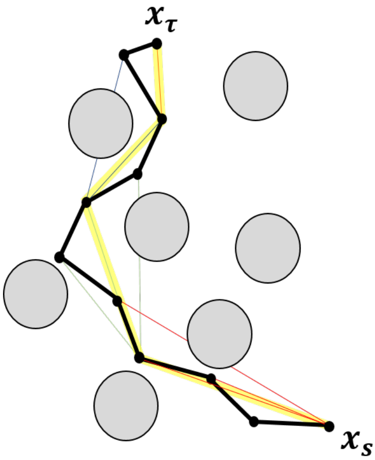


Why are RRTs so popular?

- 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

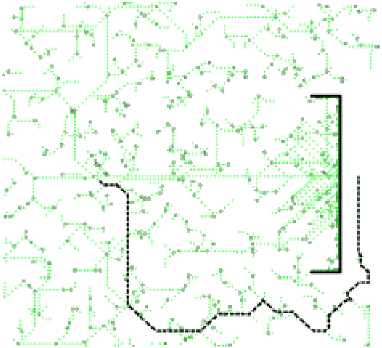
Path Smoothing

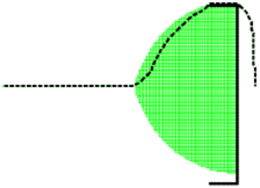
- Start with the initial point (1)
- Make connections to subsequent points in the path (2), (3), (4), ...
- When a connection collides with obstacles, add the previous waypoint to the smoothed path
- Continue smoothing from this point on



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

- 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*** (S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," IJRR, 2010)
  - RRT + rewiring of the tree to ensure asymptotic optimality
  - Contains two steps: **extend** (similar to RRT) and **rewire** (new)

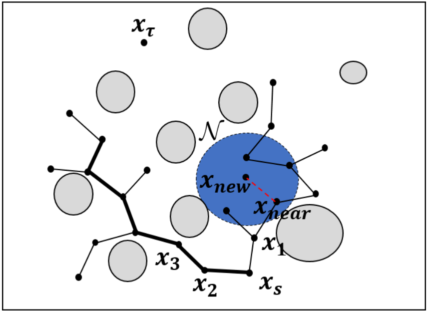
RRT*: Extend Step

- 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

$$r^* > 2\left(1 + \frac{1}{d}\right)^{1/d}\left(\frac{Vol(C_{free})}{Vol(\text{Unit d-ball})}\right)^{1/d}\left(\frac{\log|V|}{|V|}\right)^{(1/d)}$$

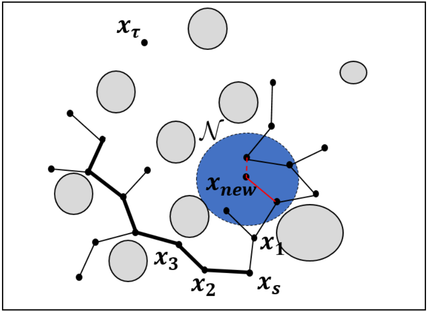
- Let $x_{nearest} = \arg\min_{x_{near} \in \mathcal{N}} g_{x_{near}} + c_{x_{near}, x_{new}}$ be the node in $\mathcal{N}$ on the currently known shortest path from x_s to x_{new}
- $V \leftarrow V \cup \{x_{new}\}$
- $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
- Set the label of x_{new} to:

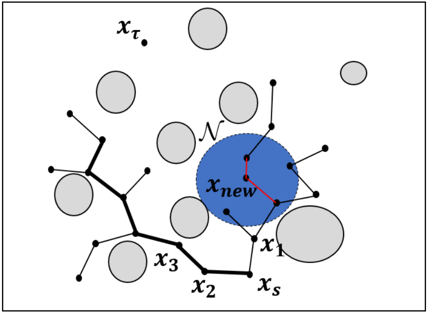
$$g_{x_{new}} = g_{x_{nearest}} + c_{x_{nearest}, x_{new}}$$



RRT*: Rewire Step

- Check all nodes $x_{near} \in \mathcal{N}$ to see if re-routing through x_{new} reduces the path length (**label correcting!**):
- If $g_{x_{new}} + c_{x_{new},x_{near}} < g_{x_{near}}$, then remove the edge between x_{near} and its parent and add a new edge between x_{near} and x_{new}

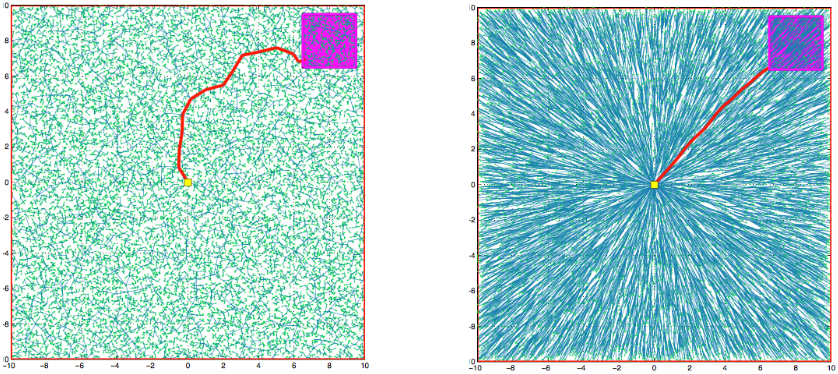




Algorithm 6 RRT*

$V \leftarrow \{x_s\}$; $E \leftarrow \emptyset$
for $i = 1 \dots n$ **do**
    $x_{rand} \leftarrow \textsc{SampleFree}()$
    $x_{nearest} \leftarrow \textsc{Nearest}((V, E), x_{rand})$
    $x_{new} \leftarrow \textsc{Steer}(x_{nearest}, x_{rand})$
    if $\textsc{CollisionFree}(x_{nearest}, x_{new})$ **then**
        $X_{near} \leftarrow \textsc{Near}((V, E), x_{new}, \min\{r^*, \epsilon\})$
        $V \leftarrow V \cup \{x_{new}\}$
        $c_{min} \leftarrow \textsc{Cost}(x_{nearest}) + \textsc{Cost}(Line(x_{nearest}, x_{new}))$
        for $x_{near} \in X_{near}$ **do** ▷ Extend along a minimum-cost path
            if $\textsc{CollisionFree}(x_{near}, x_{new})$ **then**
                if $\textsc{Cost}(x_{near}) + \textsc{Cost}(Line(x_{near}, x_{new})) < c_{min}$ **then**
                    $x_{min} \leftarrow x_{near}$
                    $c_{min} \leftarrow \textsc{Cost}(x_{near}) + \textsc{Cost}(Line(x_{near}, x_{new}))$
        $E \leftarrow E \cup \{(x_{min}, x_{new}\}$
        for $x_{near} \in X_{near}$ **do** ▷ Rewire the tree
            if $\textsc{CollisionFree}(x_{new}, x_{near})$ **then**
                if $\textsc{Cost}(x_{new}) + \textsc{Cost}(Line(x_{new}, x_{near})) < \textsc{Cost}(x_{near})$ **then**
                    $x_{parent} \leftarrow \textsc{Parent}(x_{near})$
                    $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$
return $G = (V, E)$

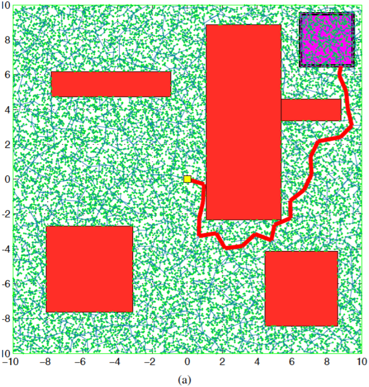
RRT vs RRT*



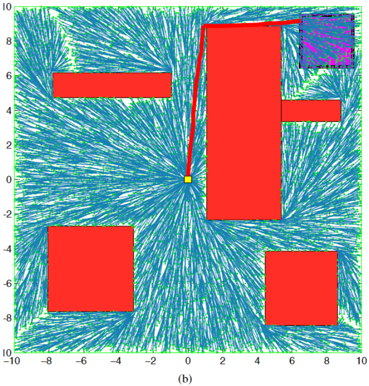
(a) RRT (b) RRT*

- 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*



(a) RRT



(b) RRT*

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