ECE276B: Planning & Learning in Robotics
Lecture 9: Sampling-based Motion Planning

Instructor:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistant:

Hanwen Cao: h1cao@ucsd.edu

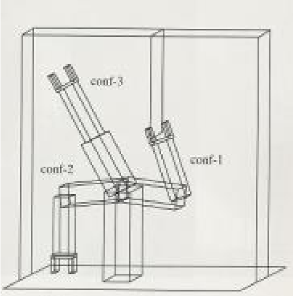
UC San Diego

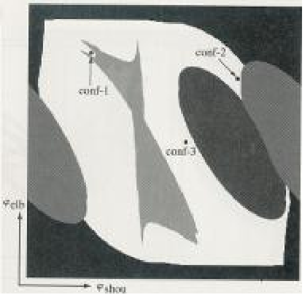
JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

Search-based vs Sampling-based Planning

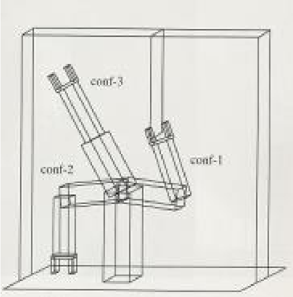
- Search-based planning:
  - Generates a graph by systematic discretization
  - Searches the graph for a path, guaranteeing to find one if it exists (resolution complete)
  - Can interleave the graph construction with the search, i.e., nodes added only when necessary
  - Provides finite-time suboptimality bounds on the solution
  - Computationally expensive in high dimensions

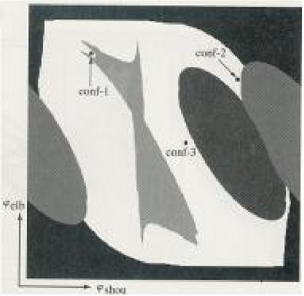


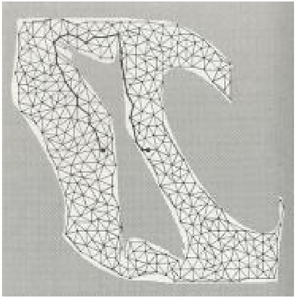


Search-based vs. Sampling-based Planning

- Sampling-based planning:
  - Generates a sparse sample-based graph
  - Searches the graph 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 graph construction with the search, i.e., samples added only when necessary
  - Provides asymptotic suboptimality bounds on the solution
  - Well-suited for high-dimensional planning: 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: $\mathbf{x}_s \in C_{free}$; Goal state: $\mathbf{x}_\tau \in C_{free}$
- **Path**: a continuous function $\rho : [0,1] \to C$; Set of all paths: $\mathcal{P}$
- **Feasible path**: a continuous function $\rho : [0,1] \to C_{free}$ such that $\rho(0) = \mathbf{x}_s$ and $\rho(1) = \mathbf{x}_\tau$; Set of all feasible paths: $\mathcal{P}_{s,\tau}$
- **Motion Planning Problem** Given a path planning problem $(C_{free}, \mathbf{x}_s, \mathbf{x}_\tau)$ and a cost function $J : \mathcal{P} \to \mathbb{R}_{\geq 0}$, find a feasible path ρ^* such that:

$$J(\rho^*) = \min_{\rho \in \mathcal{P}_{s,\tau}} J(\rho)$$

  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 $\mathbf{x} \in C$, returns a vertex $\mathbf{v} \in V$ that is closest to $\mathbf{x}$:

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

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

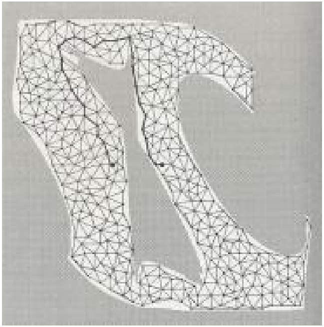
$$\textsc{Near}((V, E), \mathbf{x}, r) := \{\mathbf{v} \in V \mid \|\mathbf{x} - \mathbf{v}\| \leq r\}$$

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

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

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

Probabilistic Roadmap (PRM)



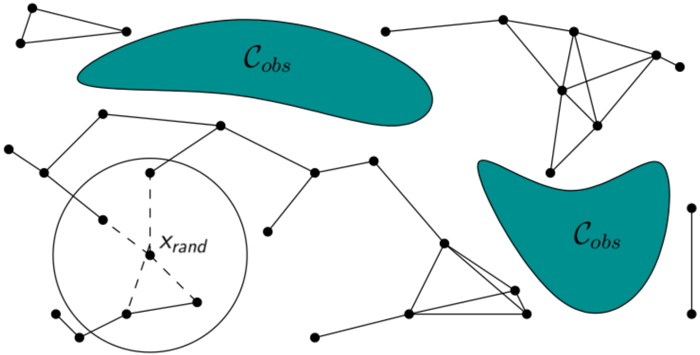
Step 1. **Construction Phase**: Build a roadmap (graph) G which, hopefully, should be accessible from any point in C_{free}

- **Nodes**: randomly sampled valid configurations $\mathbf{x}_i \in C_{free}$
- **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 $\mathbf{x}_s$ and goal configuration $\mathbf{x}_\tau$, connect them to the roadmap G, then search the augmented roadmap for a shortest path from $\mathbf{x}_s$ to $\mathbf{x}_\tau$

- **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: Construction Phase



Step 1: Construction Phase

Algorithm 1 PRM (construction phase)

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

- G.same_component$(\mathbf{x}_{rand}, \mathbf{x})$
  - ensures that $\mathbf{x}$ and $\mathbf{x}_{rand}$ are in different components of G
  - every connection decreases the number of connected components in G
  - efficient implementation using union-find algorithms
  - may be replaced by G.vertex_degree$(\mathbf{x}) < K$ for some fixed K (e.g., $K = 15$) if it is important to generate multiple alternative paths

Asymptotically 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*

```
V ← {x_s} ∪ {SampleFree()}_{i=1}^{n}; E ← ∅
for v ∈ V do
    for x ∈ Near((V, E), v, r*) \ {v} do
        if CollisionFree(v, x) then
            E ← E ∪ {(v, x), (x, v)}
return G = (V, E)
```

PRM vs RRT

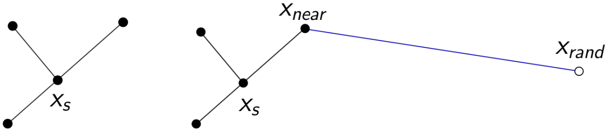
- **PRM**: a graph constructed from random samples. It can be searched for a path whenever a start node $\mathbf{x}_s$ and goal node $\mathbf{x}_\tau$ are specified. PRMs are well-suited for repeated planning between different pairs of $\mathbf{x}_s$ and $\mathbf{x}_\tau$ (*multiple queries*)

- **RRT**: a tree constructed from random samples with root $\mathbf{x}_s$. The tree is grown until it contains a path to $\mathbf{x}_\tau$. RRTs are well-suited for single-shot planning between a single pair of $\mathbf{x}_s$ and $\mathbf{x}_\tau$ (*single query*)

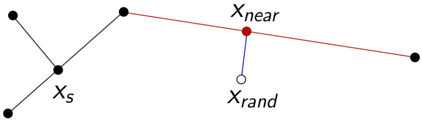
- **Rapidly Exploring Random Tree** (RRT):
  - One of the most popular planning techniques
  - Introduced by Steven LaValle in 1998
  - Many, many, many extensions and variants (articulated robots, kinematics, dynamics, differential constraints)
  - There exist extensions of RRTs that try to reuse a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

Rapidly Exploring Random Tree (RRT)

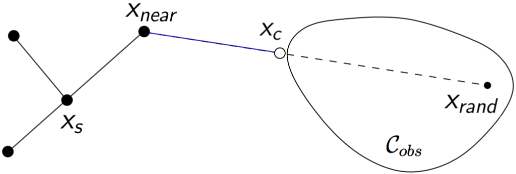
- Sample a new configuration $\mathbf{x}_{rand}$, find the nearest neighbor $\mathbf{x}_{nearest}$ in G and connect them:



- (Optional) if $\mathbf{x}_{nearest}$ 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)

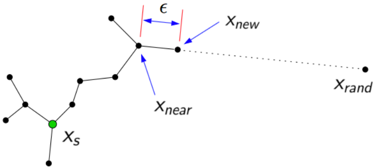
- What about the goal? Occasionally (e.g., every 100 iterations) add the goal configuration $\mathbf{x}_\tau$ and see if it gets connected to the tree
- RRT can be implemented in the original workspace (need to do collision checking) or in configuration space
- Challenges with a C-Space implementation:
  - 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 $\mathbf{x}_s$ build a graph (actually, tree) until the goal configuration $\mathbf{x}_\mathcal{T}$ is part of it

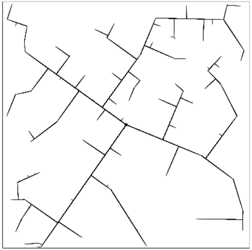
Algorithm 3 RRT

$V \leftarrow \{\mathbf{x}_s\}$; $E \leftarrow \emptyset$
for $i = 1 \dots n$ **do**
  $\mathbf{x}_{rand} \leftarrow \textsc{SampleFree}()$
  $\mathbf{x}_{nearest} \leftarrow \textsc{Nearest}((V, E), \mathbf{x}_{rand})$
  $\mathbf{x}_{new} \leftarrow \textsc{Steer}(\mathbf{x}_{nearest}, \mathbf{x}_{rand})$
  if $\textsc{CollisionFree}(\mathbf{x}_{nearest}, \mathbf{x}_{new})$ **then**
    $V \leftarrow V \cup \{\mathbf{x}_{new}\}$; $E \leftarrow E \cup \{(\mathbf{x}_{nearest}, \mathbf{x}_{new})\}$
return $G = (V, E)$

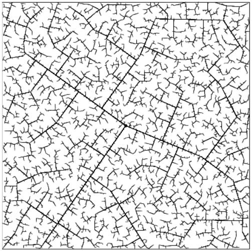


Rapidly Exploring Random Tree (RRT)

- RRT with $\epsilon = \infty$ (called Rapidly Exploring Dense Tree (RDT)):

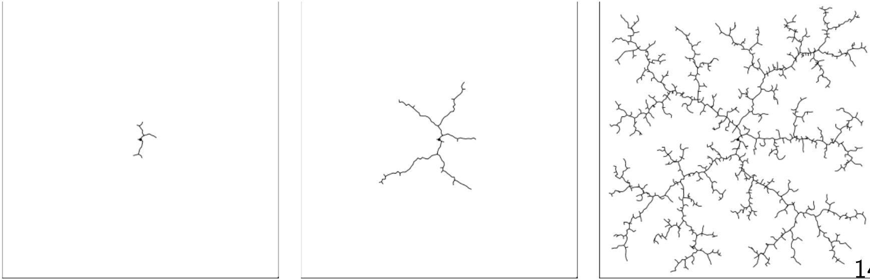


45 iterations



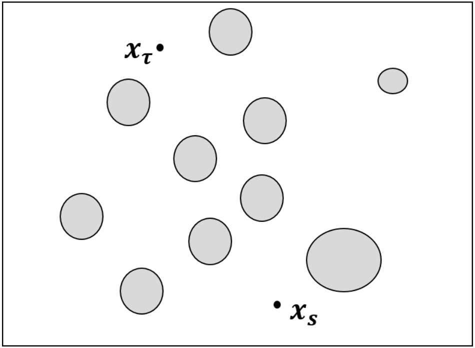
2345 iterations

- RRT with $\epsilon < \infty$



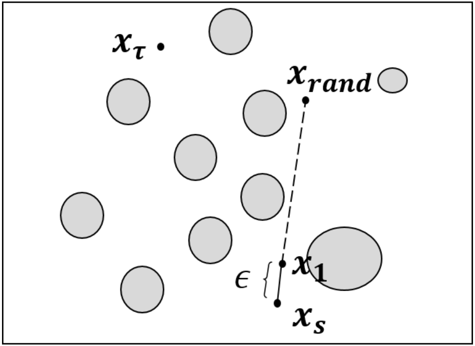
Example: RRT Algorithm

- Start node $\mathbf{x}_s$
- Goal node $\mathbf{x}_\tau$
- Gray obstacles



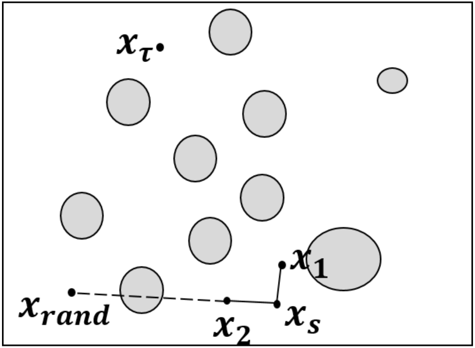
Example: RRT Algorithm

- Sample $\mathbf{x}_{rand}$ in the workspace
- Steer from $\mathbf{x}_s$ towards $\mathbf{x}_{rand}$ by a fixed distance ϵ to get $\mathbf{x}_1$
- If the segment from $\mathbf{x}_s$ to $\mathbf{x}_1$ is collision-free, insert $\mathbf{x}_1$ into the tree



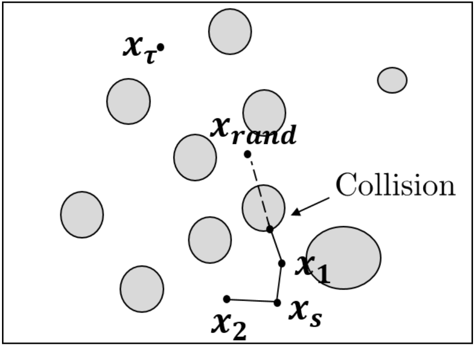
Example: RRT Algorithm

- Sample $\mathbf{x}_{rand}$ in the workspace
- Find the closest node $\mathbf{x}_{nearest}$ to $\mathbf{x}_{rand}$
- Steer from $\mathbf{x}_{nearest}$ towards $\mathbf{x}_{rand}$ by a fixed distance ϵ to get $\mathbf{x}_2$
- If the segment from $\mathbf{x}_{nearest}$ to $\mathbf{x}_2$ is collision-free, insert $\mathbf{x}_2$ into the tree



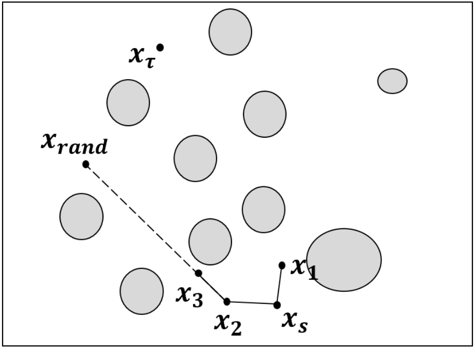
Example: RRT Algorithm

- Sample $\mathbf{x}_{rand}$ in the workspace
- Find the closest node $\mathbf{x}_{nearest}$ to $\mathbf{x}_{rand}$
- Steer from $\mathbf{x}_{nearest}$ towards $\mathbf{x}_{rand}$ by a fixed distance ϵ to get $\mathbf{x}_3$
- If the segment from $\mathbf{x}_{nearest}$ to $\mathbf{x}_3$ is collision-free, insert $\mathbf{x}_3$ into the tree



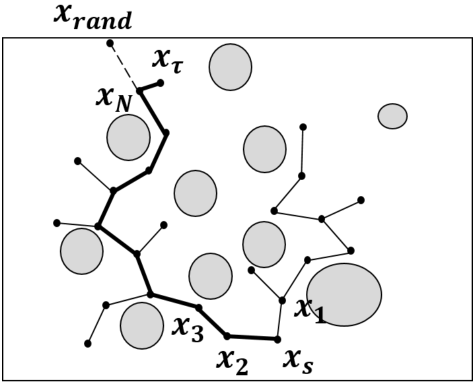
Example: RRT Algorithm

- Sample $\mathbf{x}_{rand}$ in the workspace
- Find the closest node $\mathbf{x}_{nearest}$ to $\mathbf{x}_{rand}$
- Steer from $\mathbf{x}_{nearest}$ towards $\mathbf{x}_{rand}$ by a fixed distance ϵ to get $\mathbf{x}_3$
- If the segment from $\mathbf{x}_{nearest}$ to $\mathbf{x}_3$ is collision-free, insert $\mathbf{x}_3$ into the tree



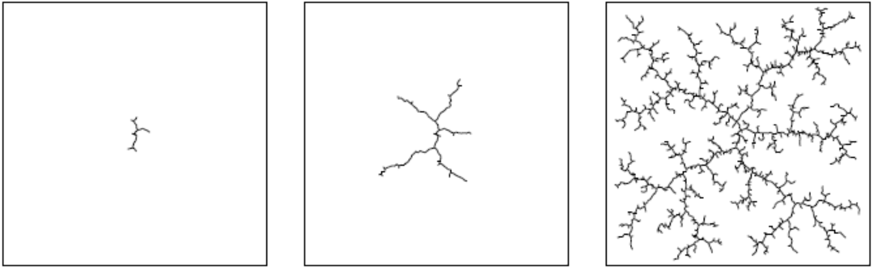
Example: RRT Algorithm

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

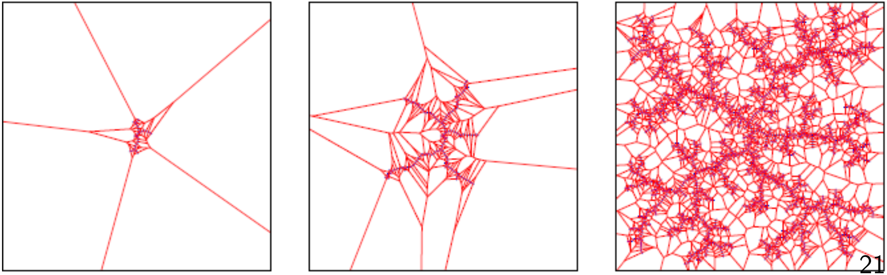


Sampling in RRTs

- The vanilla RRT algorithm provides uniform coverage of space

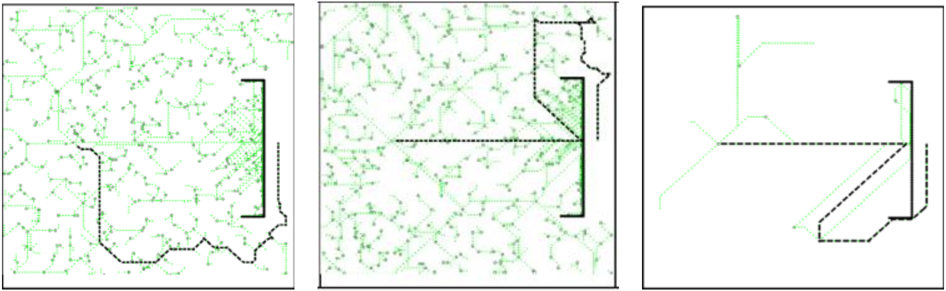


- Alternatively, the growth may be biased by the largest Voronoi region



Sampling in RRTs

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

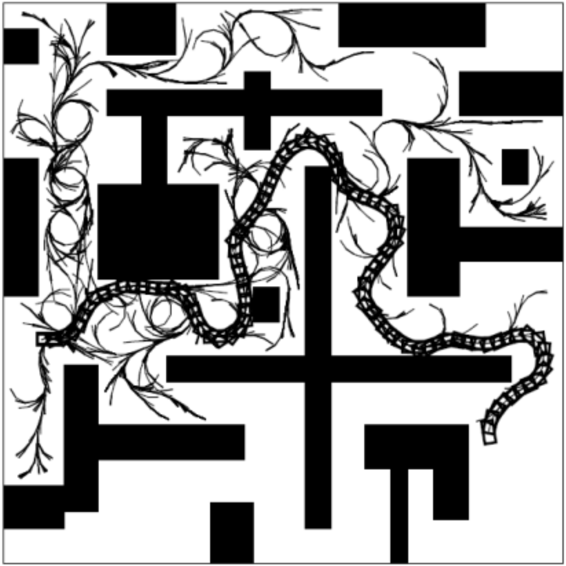


(a) $p_g = 0$ (b) $p_g = 0.1$ (c) $p_g = 0.5$

Handling Robot Dynamics with Steer()

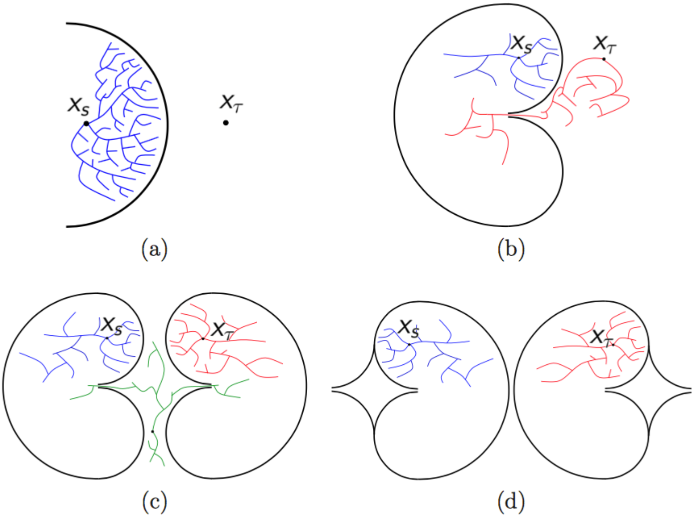
- Steer() extends the tree towards a given random sample $\mathbf{x}_{rand}$
- Consider a car-like robot with non-holonomic constraints (no sideways motion) in $SE(2)$. Obtaining a feasible path from $\mathbf{x}_{rand} = (0, 0, 90^\circ)$ to $\mathbf{x}_{nearest} = (1, 0, 90^\circ)$ is as challenging as the original planning problem
- Steer() resolves this by not requiring the motion to get all the way to $\mathbf{x}_{rand}$. Instead, apply the best control input for a fixed duration to obtain $\mathbf{x}_{new}$ and a dynamically feasible trajectory to it
- See: Y. Li, Z. Littlefield, K. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," The International Journal of Robotics Research, 2016.

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

$V_a \leftarrow \{\mathbf{x}_s\};\ E_a \leftarrow \emptyset;\ V_b \leftarrow \{\mathbf{x}_\tau\};\ E_b \leftarrow \emptyset$
for $i = 1 \dots n$ **do**
  $\mathbf{x}_{rand} \leftarrow \textsc{SampleFree}()$
  $\mathbf{x}_{nearest} \leftarrow \textsc{Nearest}((V_a, E_a), \mathbf{x}_{rand})$
  $\mathbf{x}_{new} \leftarrow \textsc{Steer}(\mathbf{x}_{nearest}, \mathbf{x}_{rand})$
  if $\mathbf{x}_{new} \neq \mathbf{x}_{nearest}$ **then**
    $V_a \leftarrow V_a \cup \{\mathbf{x}_{new}\};\ E_a \leftarrow \{(\mathbf{x}_{nearest}, \mathbf{x}_{new}), (\mathbf{x}_{new}, \mathbf{x}_{nearest})\}$
    $\mathbf{x}'_{nearest} \leftarrow \textsc{Nearest}((V_b, E_b), \mathbf{x}_{new})$
    $\mathbf{x}'_{new} \leftarrow \textsc{Steer}(\mathbf{x}'_{nearest}, \mathbf{x}_{new})$
    if $\mathbf{x}'_{new} \neq \mathbf{x}'_{nearest}$ **then**
      $V_b \leftarrow V_b \cup \{\mathbf{x}'_{new}\};\ E_b \leftarrow \{(\mathbf{x}'_{nearest}, \mathbf{x}'_{new}), (\mathbf{x}'_{new}, \mathbf{x}'_{nearest})\}$
    if $\mathbf{x}'_{new} = \mathbf{x}_{new}$ **then return** SOLUTION
  if $|V_b| < |V_a|$ **then** $\textsc{Swap}((V_a, E_a), (V_b, E_b))$
return FAILURE

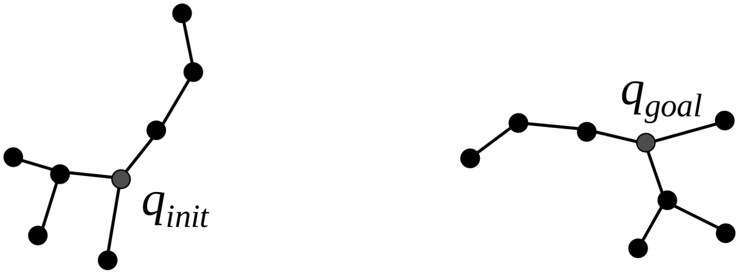
RRT-Connect (J. Kuffner and S. LaValle, ICRA, 2000)

- Bi-directional tree + attempts to connect the two trees at every iteration

Algorithm 5 RRT-Connect

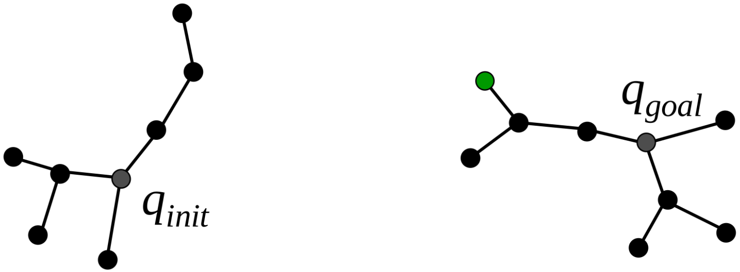
$V_a \leftarrow \{\mathbf{x}_s\};\ E_a \leftarrow \emptyset;\ V_b \leftarrow \{\mathbf{x}_\tau\};\ E_b \leftarrow \emptyset$
for $i = 1 \ldots n$ **do**
  $\mathbf{x}_{rand} \leftarrow \textsc{SampleFree}()$
  if not $\textsc{Extend}((V_a, E_a), \mathbf{x}_{rand}) =$ Trapped **then**
    if $\textsc{Connect}((V_b, E_b), \mathbf{x}_{new}) =$ Reached **then** $\triangleright$ $\mathbf{x}_{new}$ 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 $\textsc{Extend}((V, E), \mathbf{x})$
  $\mathbf{x}_{nearest} \leftarrow \textsc{Nearest}((V, E), \mathbf{x})$
  $\mathbf{x}_{new} \leftarrow \textsc{Steer}_\epsilon(\mathbf{x}_{nearest}, \mathbf{x})$
  if $\textsc{CollisionFree}(\mathbf{x}_{nearest}, \mathbf{x}_{new})$ **then**
    $V \leftarrow \{\mathbf{x}_{new}\};\ E \leftarrow \{(\mathbf{x}_{nearest}, \mathbf{x}_{new}), (\mathbf{x}_{new}, \mathbf{x}_{nearest})\}$
    if $\mathbf{x}_{new} = \mathbf{x}$ **then return** Reached **else return** Advanced
  return Trapped
function $\textsc{Connect}((V, E), \mathbf{x})$
  repeat *status* $\leftarrow \textsc{Extend}((V, E), \mathbf{x})$ **until** *status* $\neq$ *Advanced*
  return *status*

Example: Single RRT-Connect Iteration



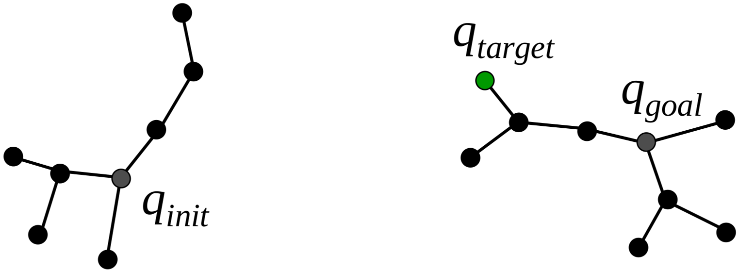
Example: Single RRT-Connect Iteration

- One tree is grown to a random target



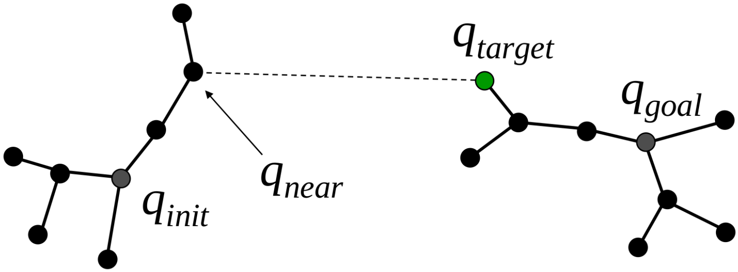
Example: Single RRT-Connect Iteration

- The new node becomes a target for the other tree



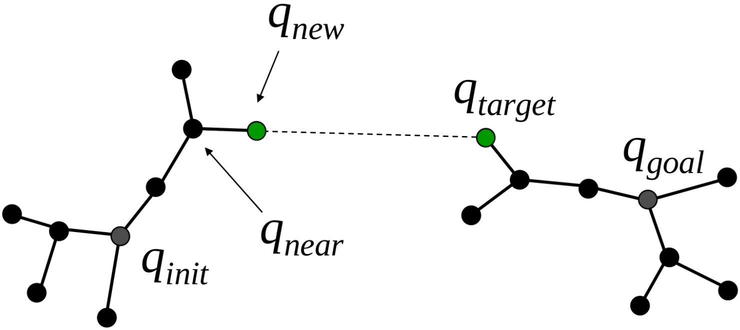
Example: Single RRT-Connect Iteration

- Determine the nearest node to the target



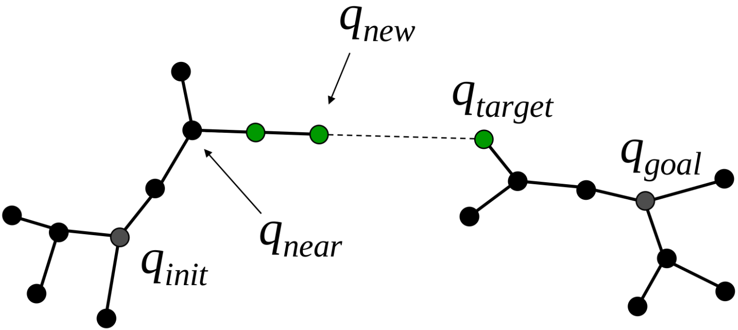
Example: Single RRT-Connect Iteration

- Try to add a new collision-free branch



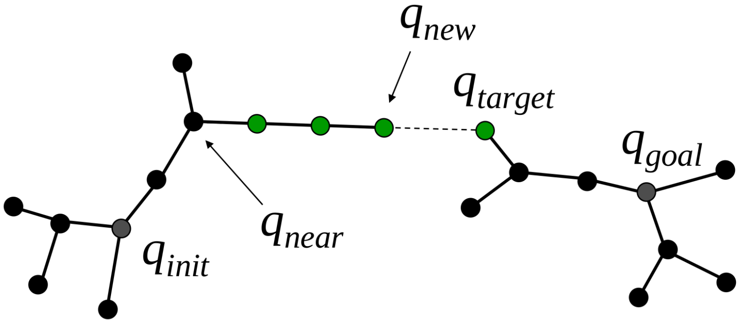
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



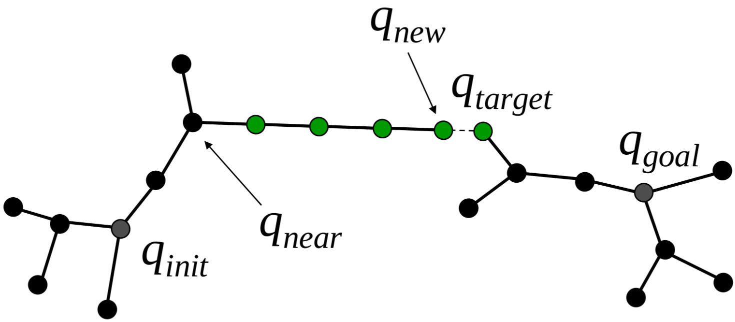
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



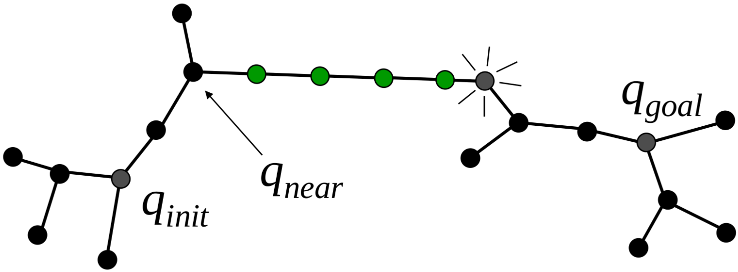
Example: Single RRT-Connect Iteration

- If successful, keep extending the branch



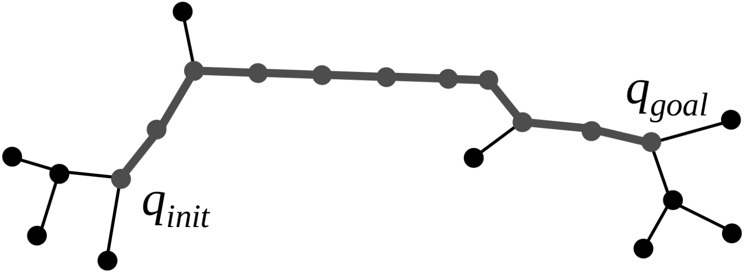
Example: Single RRT-Connect Iteration

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

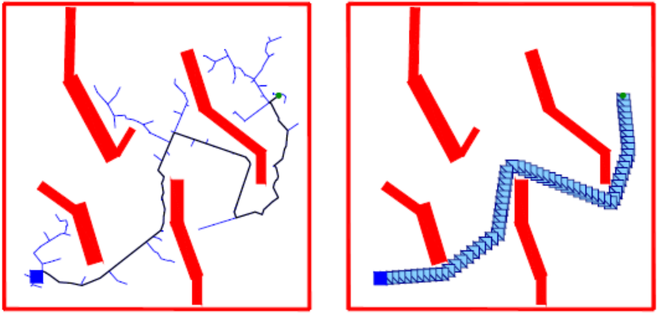


Example: Single RRT-Connect Iteration

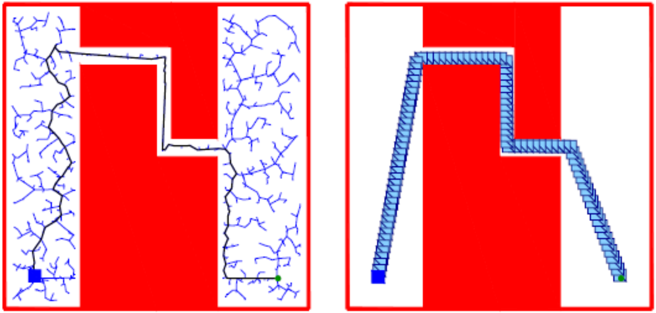
- If the branch reaches all the way to the target, a feasible path is found!



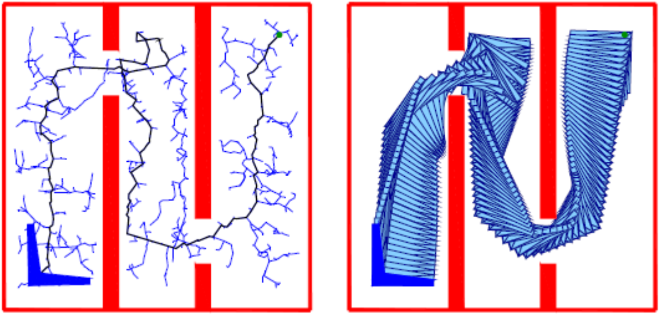
Example: RRT-Connect



Example: RRT-Connect



Example: RRT-Connect

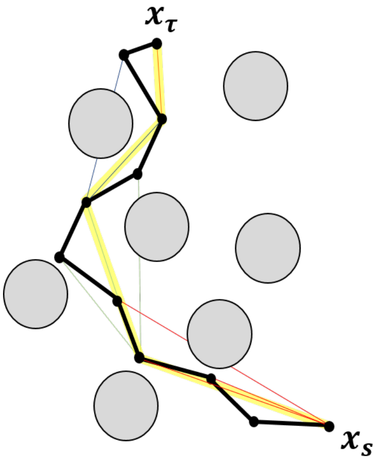


Why are RRTs so popular?

- The algorithm is very simple once the following subroutines are implemented:
  - Random sample generator
  - Nearest neighbor
  - Collision checker
  - Steer
- Pros:
  - A sparse graph 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 (see Gammell et al., BIT*, IJRR, 2020.)
- Cons:
  - Computed paths may be sub-optimal and require path smoothing as a post-processing step
  - Finding a feasible path in highly constrained environments (e.g., maze) is challenging

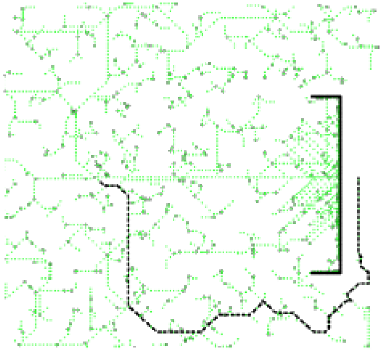
Path Smoothing

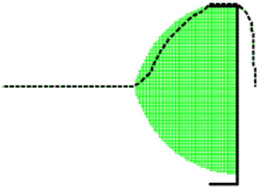
- Start with the initial point (1)
- Make connections to subsequent points in the path (2), (3), (4), ...
- When a connection collides with obstacles, add the previous waypoint to the smoothed path
- Continue smoothing from this point on



Search-based vs Sampling-based Planning

- RRT:
  - A sparse graph requires little memory and computation
  - Computed paths may be sub-optimal and require path smoothing
- 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

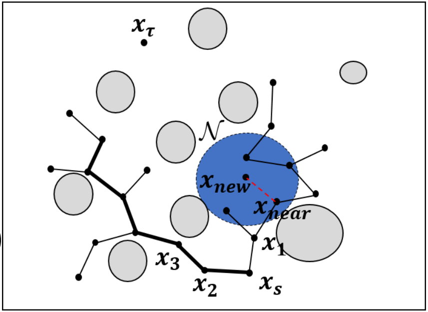
- RRT and RRT-Connect are **probabilistically complete**: the probability that a feasible path will be found if one exists, approaches 1 exponentially as the number of samples approaches infinity
- Assuming C_{free} is connected, bounded, and open, for any $x \in \mathcal{C}_{free}$, $\lim_{N\to\infty} \mathbb{P}(\|\mathbf{x} - \mathbf{x}_{nearest}\| < \epsilon) = 1$, where $\mathbf{x}_{nearest}$ is the closest node to $\mathbf{x}$ in G
- 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 $\mathbf{x}_{new}$ 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 $\mathbf{x}_{nearest} = \underset{\mathbf{x}_{near}\in\mathcal{N}}{\arg\min}\, g(\mathbf{x}_{near}) + c(\mathbf{x}_{near}, \mathbf{x}_{new})$ be the node in $\mathcal{N}$ on the currently known shortest path from $\mathbf{x}_s$ to $\mathbf{x}_{new}$

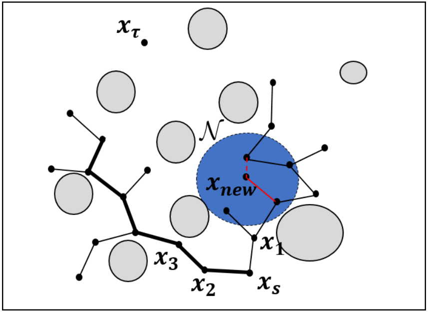


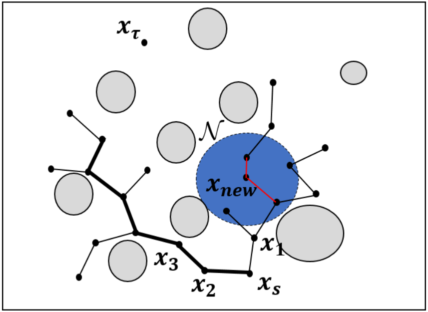
- $V \leftarrow V \cup \{\mathbf{x}_{new}\}$
- $E \leftarrow E \cup \{(\mathbf{x}_{nearest}, \mathbf{x}_{new})\}$
- Set the label of $\mathbf{x}_{new}$ to:

$$g(\mathbf{x}_{new}) = g(\mathbf{x}_{nearest}) + c(\mathbf{x}_{nearest}, \mathbf{x}_{new})$$

RRT*: Rewire Step

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

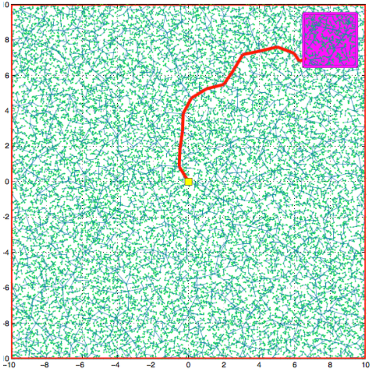




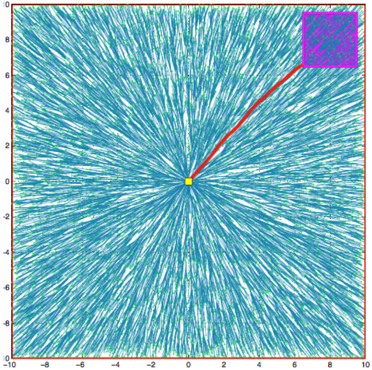
Algorithm 6 RRT*

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

RRT vs RRT*



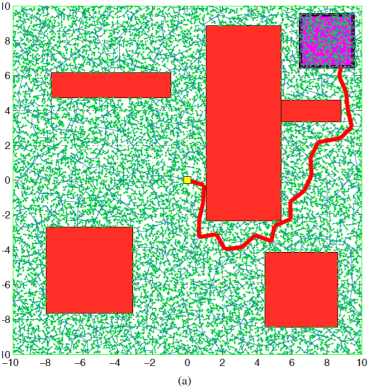
(a) RRT



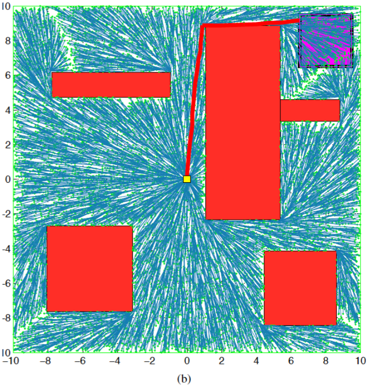
(b) RRT*

- 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).