# Motion Planning Lecture 7

Kinodynamic Planning: kinodynamic RRT, SST*, AO-x
Geometric Planning: RRT-Connect, EST, PRM*

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

June 5, 2024

# Recap

- Tree-based motion planning: RRT (probabilistic complete (PC), but suboptimal)
- Asymptotic Optimality (AO)
- RRT* introduces rewiring (probablistic complete and asymptotically optimal)
- Proof sketches
  - PC RRT (by induction; series of connectable balls)
  - AO RRT* (by induction; use re-wiring to establish correct sequence)
- Informed RRT* and BIT*

# Optimal kinodynamic planning

# Kinodynamic planning

# Kinodynamic planning

## Given

- State space $\mathcal{Q}$ and free state space $\mathcal{Q}_{free}$
- Control space $\mathcal{U}$
- Dynamics $\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t))$
- Initial state $\mathbf{q}_{start} \in \mathcal{Q}_{free}$
- Goal region $\mathcal{Q}_{goal} \subset \mathcal{Q}_{free}$
- Cost $c = J(T, \mathbf{u}(t), \mathbf{q}(t))$

## Desired

- Trajectory $\pi : [0, T] \to \mathcal{Q}_{free} \times \mathcal{U}$

# Kinodynamic planning

- Feasible kinodynamic planning: Compute a trajectory $\pi$
- Optimal kinodynamic planning: From all feasible trajectories, select the one which minimizes the cost (we call it $\pi^\star$)

# Optimal kinodynamic planning

## Steering vs Forward Propagation

# Steering vs Forward Propagation

**Two variants**

Two types of kinodynamic planning depending on information available

- Steering
- Forward propagation

# Steering vs Propagation

Planners require access to dynamical function $\mathbf{f}$. This can be accomplished in two ways

- Steering: Given two states $\mathbf{q}$, $\mathbf{q}'$, compute controls to move robot from $\mathbf{q}$ to $\mathbf{q}'$
  - Involves solving a boundary-value problem (BVP)
  - Computationally expensive
  - Tricky if dynamical constraints are involved
- Forward Propagation: Given a state $\mathbf{q}$, a control $\mathbf{u}$, and a time $\Delta t$, compute the next state $\mathbf{q}'$ by applying control $\mathbf{u}$ for time $\Delta t$
  - Simple to compute
  - Does not require knowledge of system
  - Unclear how to use it for (optimal) planning

### Steering function

Planning with steering functions as generalized interpolation

- Reduction to geometric case
- Any geometric planner can be applied
- PRM, RRT*, or BIT*

# Forward propagation

Planning with forward propagation

- Difficult: Unclear how to exploit forward propagation
- How to make this optimal?
- Optimal kinodynamic motion planning: Naive random trees and SST*

# Optimal kinodynamic planning

## Meta algorithm

# Meta Algorithm (for kinodynamic planning with forward propagation)

**Meta($\mathbf{q}_{start}$, $\mathcal{Q}_{goal}$, $\mathcal{Q}$, $\mathcal{U}$, $\mathbf{f}$, $t_{prop}$)**

- T = InitializeTree($\mathbf{q}_{start}$)
- While Not Terminated
  - $\mathbf{q}_{select}$ = SelectNode(T, $\mathcal{Q}$)
  - $\mathbf{q}_{new}$ = Propagate($\mathbf{q}_{select}$, $\mathcal{U}$, $\mathbf{f}$, $t_{prop}$)
  - MaybeAddConnection($\mathbf{q}_{select}$, $\mathbf{q}_{new}$)

Note: In practice, we would terminate either after $N$ iterations, or when a path is found, or when a certain cost is reached, etc. For the theoretical analysis, however, we assume that the algorithm will not terminate (asymptotic optimality can only be reached in the limit).

# Meta Algorithm

## Select Node

- Uniform Selection: Pick a node from the graph at random
- Exploration First: Pick a node which increases explorative nature of algorithm (cover state space as quickly as possible)
- Best First: Pick a node on a high-quality path

## Propagate Node

## MaybeAddConnection

# Meta Algorithm

## Select Node

## Propagate Node

- Fixed Duration: Pick random controls, then apply them for a fixed time $t_{prop}$
- Monte-Carlo: Pick random control and random time, then propagate system forward
- Guided Monte-Carlo (”shooting” method): Select random target state. Sample $k$ controls and $k$ times. Propagate them forward and select the node nearest to target as return value.

## MaybeAddConnection

# Meta Algorithm

### Select Node

### Propagate Node

### MaybeAddConnection

- Collision-free: Add connection if no collision occured
- Prune dominated: Add connection if collision free and locally having the best cost.

# Meta Algorithm

**Instantiations of Meta algorithm**

Algorithms differ in how they implement the three modules "Select Node", "Propagate Node", and "MaybeAddConnection".

- Kinodynamic RRT (kRRT)
- Naive Random Trees (NRT)
- Stable sparse trees (SST*)

# Optimal kinodynamic planning

## Kinodynamic RRT

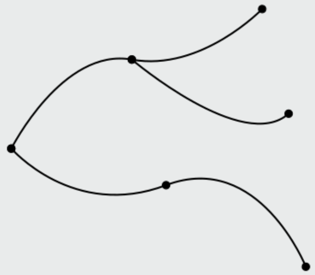
# Kinodynamic RRT

Kinodynamic RRT

- Select Node: Exploration First Selection
- Propagate Node: Guided Monte Carlo Propagation
- Maybe add connection: Collision-Free Checking

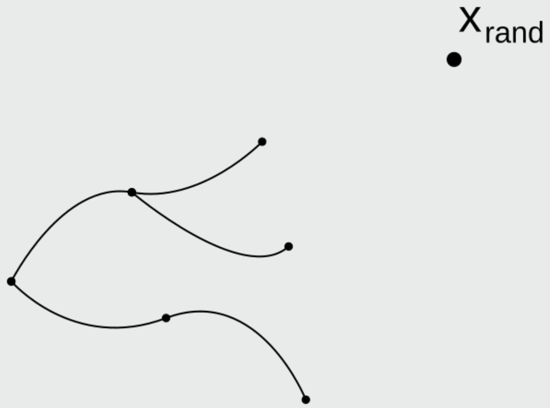
# Kinodynamic RRT

## Select Node



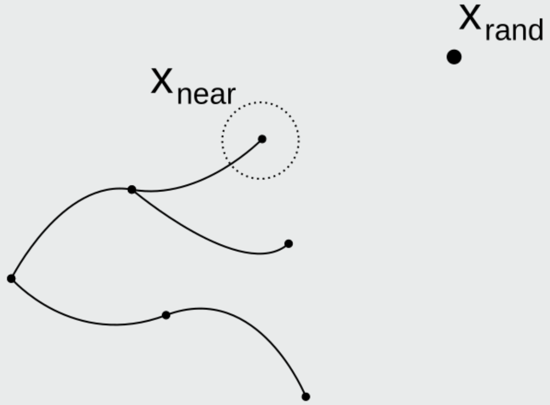
# Kinodynamic RRT

## Select Node



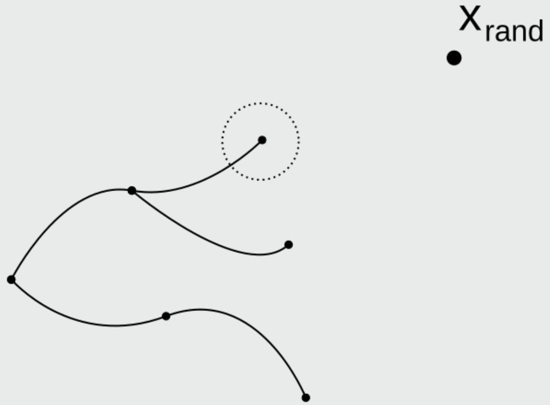
# Kinodynamic RRT

## Select Node



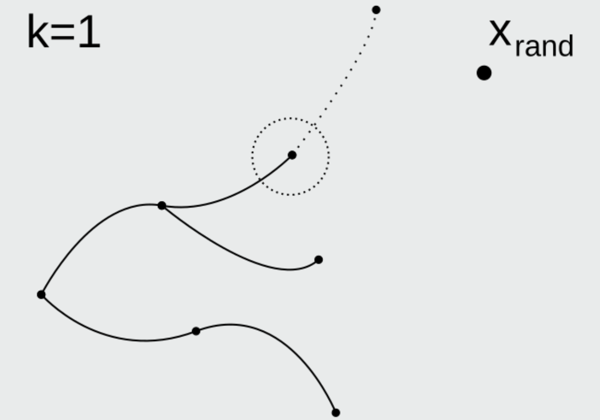
# Kinodynamic RRT

## Propagate Node



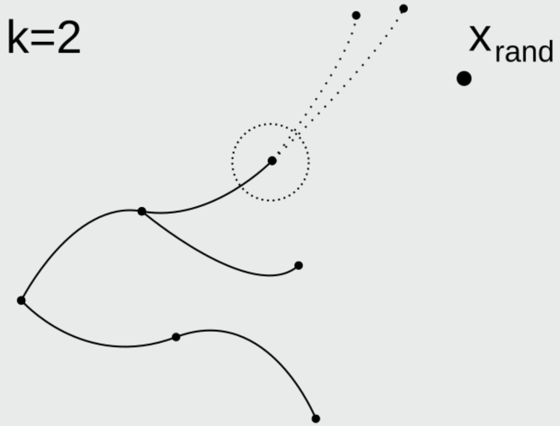
# Kinodynamic RRT

## Propagate Node



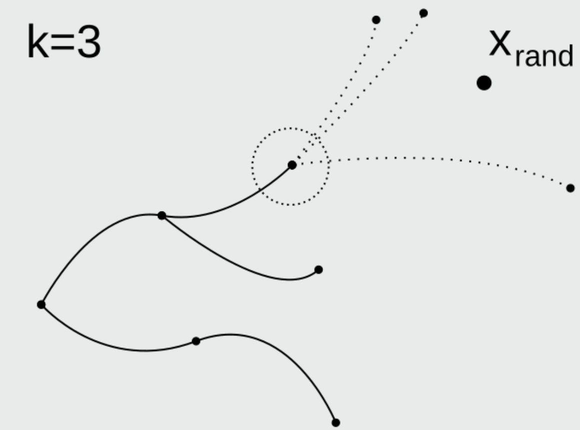
# Kinodynamic RRT

## Propagate Node



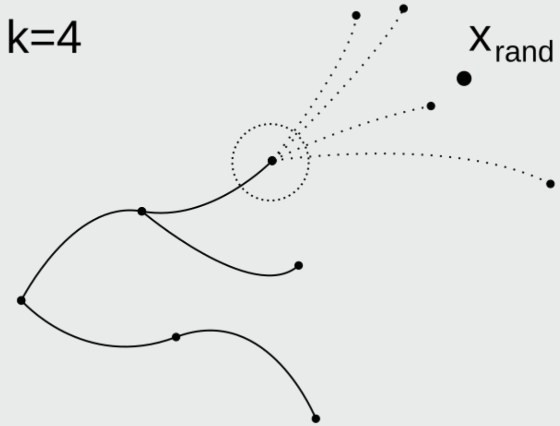
# Kinodynamic RRT

## Propagate Node



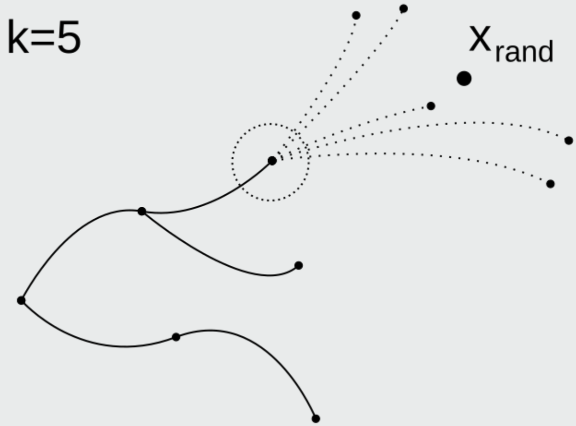
# Kinodynamic RRT

## Propagate Node



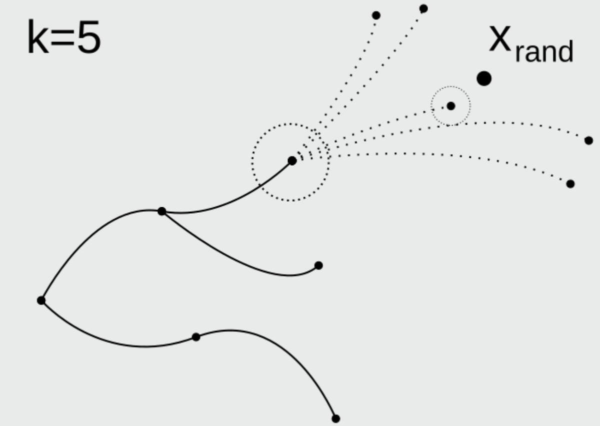
# Kinodynamic RRT

## Propagate Node



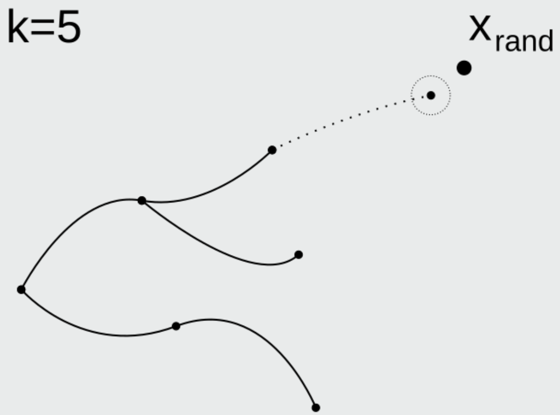
# Kinodynamic RRT

## Propagate Node



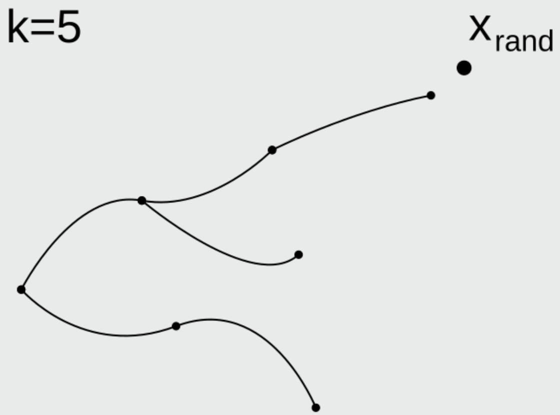
# Kinodynamic RRT

## Propagate Node



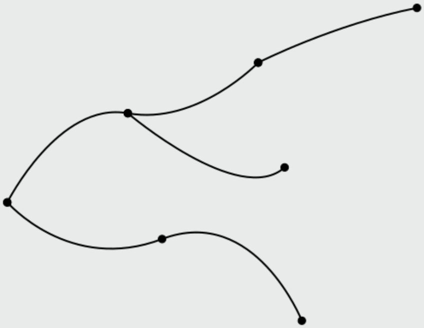
# Kinodynamic RRT

## Maybe Add Connection



# Kinodynamic RRT

## Maybe Add Connection



# Kinodynamic planning

**Properties**

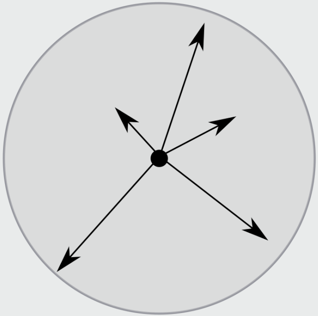
Kinodynamic RRT is probabilistically complete*

*For specific classes of dynamical systems.

LaVale and Kuffner, "Randomized Kinodynamic Planning", 2001
Kleinbort et al., "Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation", 2022

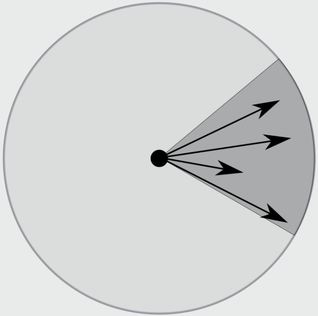
# Kinodynamic planning

## Small-space local controllability (SSLC) property



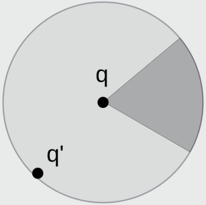
# Kinodynamic planning

## Small-space local controllability property

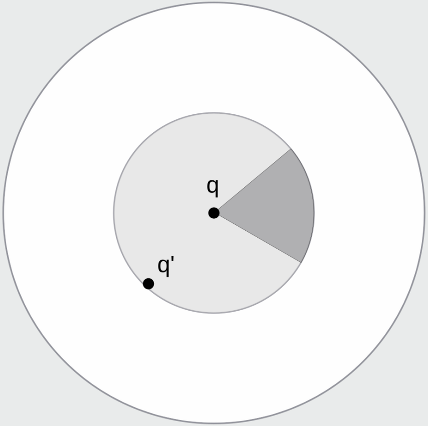


# Kinodynamic planning

## Small-space local controllability property

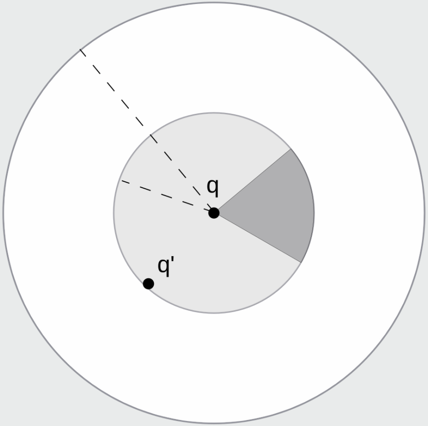


## Small-space local controllability property



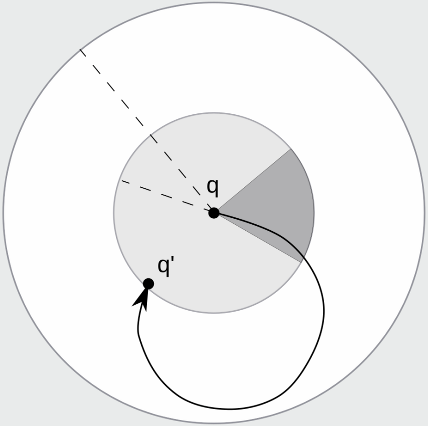
# Kinodynamic planning

## Small-space local controllability property



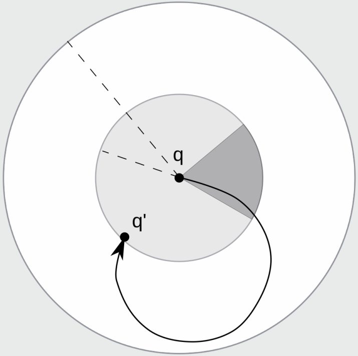
# Kinodynamic planning

## Small-space local controllability property



# Kinodynamic planning

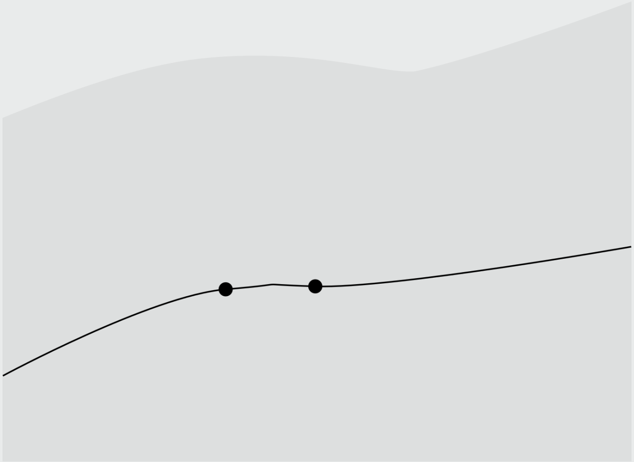
## Small-space local controllability property




Small-space local controllability property: Any configuration $q'$ at a distance less than $\delta$ is reachable from $q$ by an admissible trajectory included in a ball of size $\epsilon > \delta$.

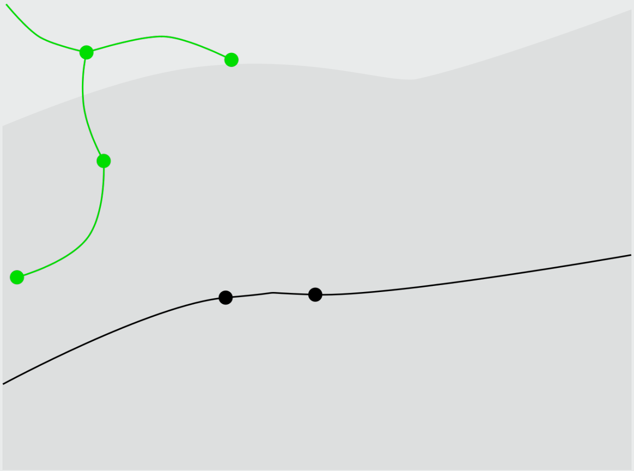
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



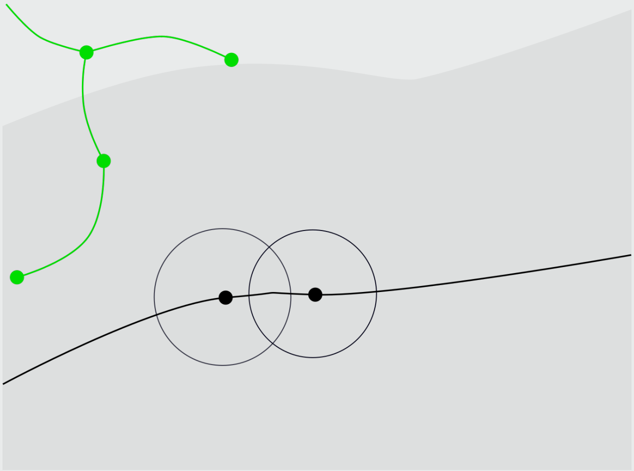
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



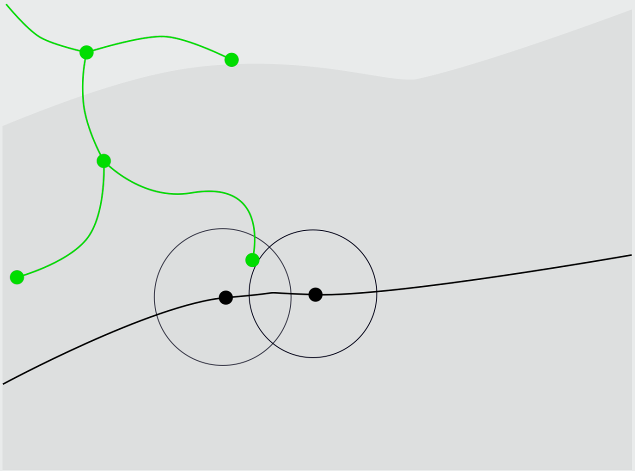
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



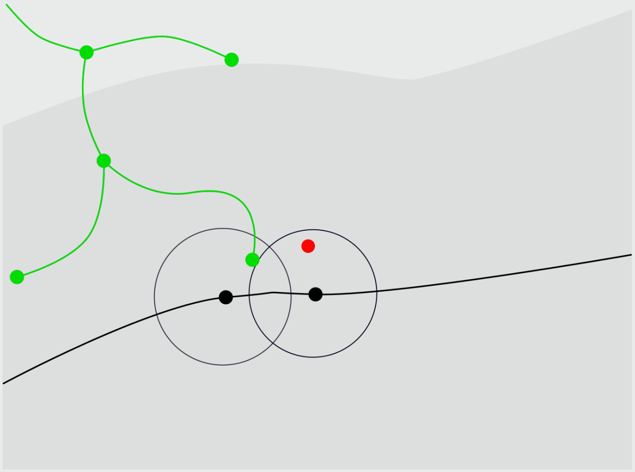
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



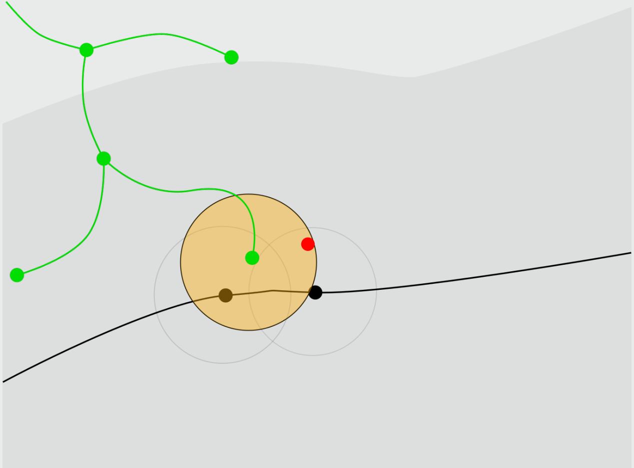
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



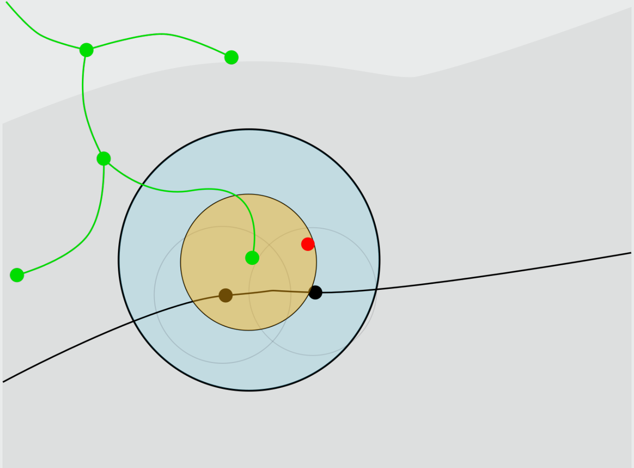
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



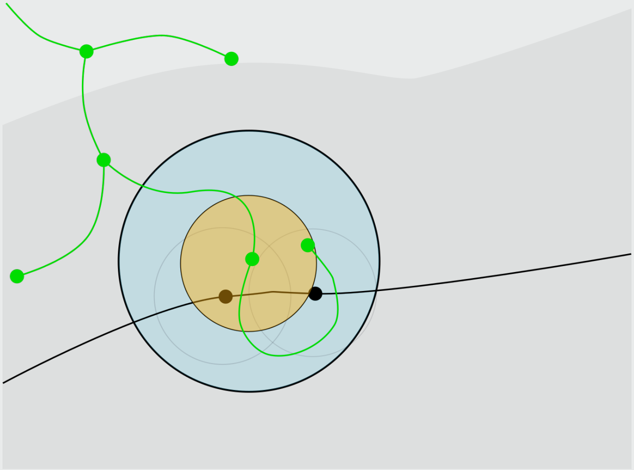
# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



# Kinodynamic planning

## Series-of-balls argument in kinodynamic planning



# Kinodynamic planning

**Question**

Is kinodynamic RRT also asymptotically optimal?

# Optimal kinodynamic planning

## Naive Random Trees

# Naive Random Trees

Naive Random Trees

- Select Node:
  Uniform Selection
- Propagate Node:
  Monte Carlo
- Maybe add connection:
  Collision-Free Checking

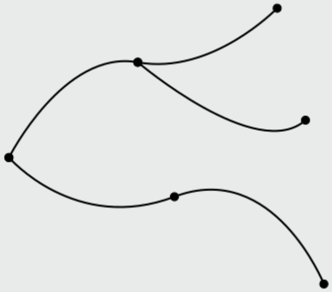
**Algorithm 2:** NAIVE_RANDOM_TREE($\mathbb{X}_f$, $\mathbb{U}$, $x_0$, $T_{prop}$, $N$)

$G = \{\mathbb{V} \leftarrow \{x_0\}, \mathbb{E} \leftarrow \emptyset\}$;
**for** $N$ *iterations* **do**
  $x_{selected} \leftarrow$ Uniform_Sampling($\mathbb{V}$);
  $x_{new} \leftarrow$MonteCarlo-Prop( $x_{selected}$, $\mathbb{U}$, $T_{prop}$ );
  **if** CollisionFree( $\overline{x_{selected} \rightarrow x_{new}}$ ) **then**
    $\mathbb{V} \leftarrow \mathbb{V} \cup \{x_{new}\}$;
    $\mathbb{E} \leftarrow \mathbb{E} \cup \{\overline{x_{selected} \rightarrow x_{new}}\}$;
**return** $G(\mathbb{V}, \mathbb{E})$;

Source: [1]

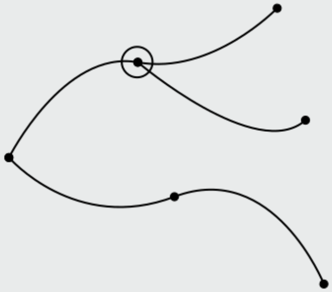
# Naive Random Trees

## Select Node



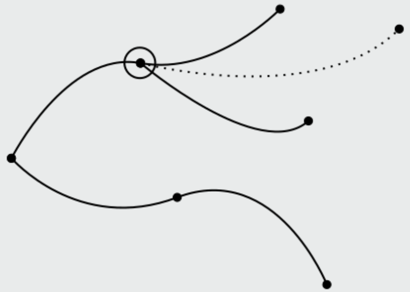
# Naive Random Trees

## Select Node



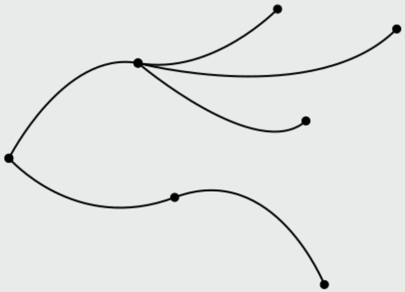
# Naive Random Trees

## Propagate Node



# Naive Random Trees

## Propagate Node



# Naive Random Trees

**Properties**

Naive Random Trees is asymptotically optimal

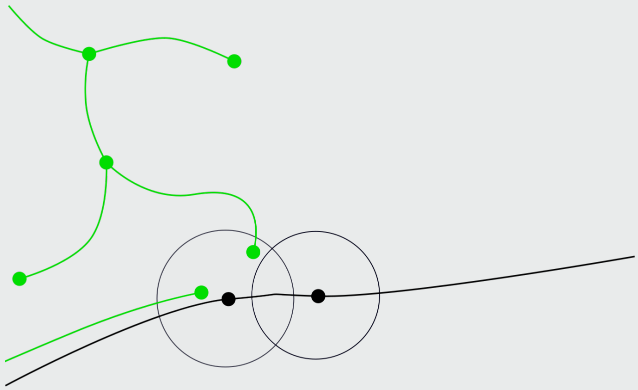
**Question**

Why is that so?

Y Li, Z Littlefield, KE Bekris, "Asymptotically Optimal Sampling-based Kinodynamic Planning", 2016

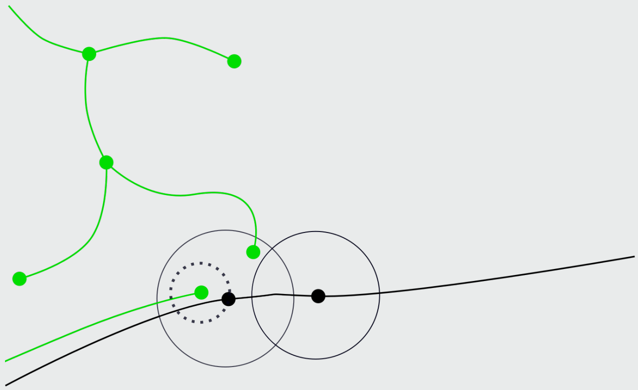
# Naive Random Trees

## Proof sketch



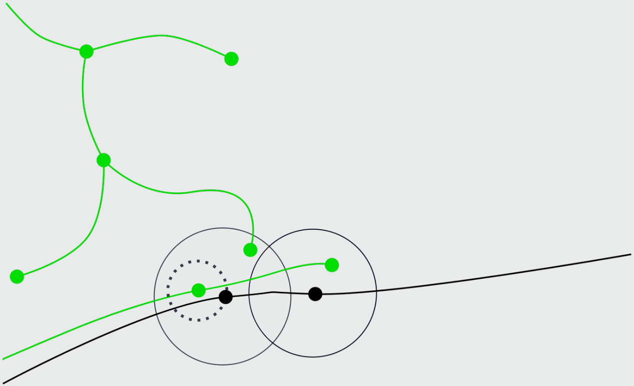
# Naive Random Trees

## Proof sketch



# Naive Random Trees

## Proof sketch



# Naive Random Trees

**Drawbacks**

- Selection of nodes uninformative
- High memory footprint

# Optimal kinodynamic planning

## Stable sparse trees (SST*)

# Sparse Stable Trees (SST)

Sparse stable trees (SST*)

- Select Node: Best First Selection
- Propagate Node: Monte Carlo
- Maybe add connection: Collision-Free Checking + Pruning

SST = Naive random trees with better selection and pruning

# Sparse Stable Trees (SST): Best First Selection

- Select node with lowest cost-to-come within a neighborhood

**Algorithm 6:** `Best_First_Selectio` $(\mathbb{X}, \mathbb{V}, \delta_{BN})$

$x_{rand} \leftarrow$ `Sample_State`$(\mathbb{X})$;
$X_{near} \leftarrow$`Near`$(\mathbb{V}, x_{rand}, \delta_{BN})$;
**If** $X_{near} = \emptyset$ **return** `Nearest`$(\mathbb{V}, x_{rand})$;
**Else return** $\arg\min_{x \in X_{near}} cost(x)$;

Source: [1]

# Sparse Stable Trees (SST)

### SST Pruning

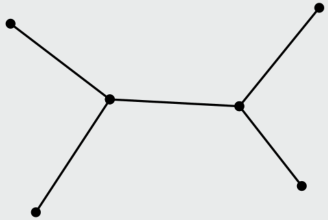
Pruning based on Witness set

### Witness Set

Set of states ($\mathcal{S} \subset \mathcal{Q}$) used as helper data structure.
Invariant for each $s \in \mathcal{S}$: only a single node of the search tree within radius $\delta_S$ represents that state $s$ and has best path cost from root.

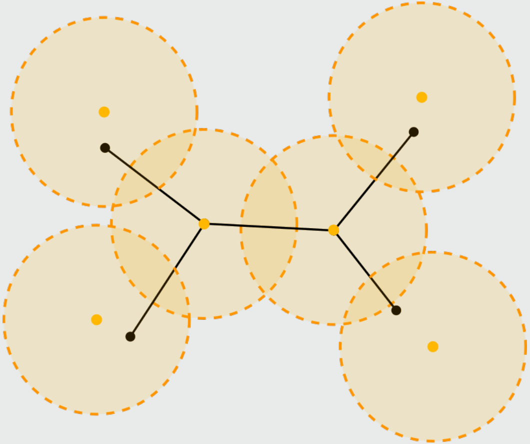
# Sparse Stable Trees (SST)

## Witness set: Search tree



# Sparse Stable Trees (SST)

**Witness set: witnesses $s \in \mathcal{S}$ in yellow**

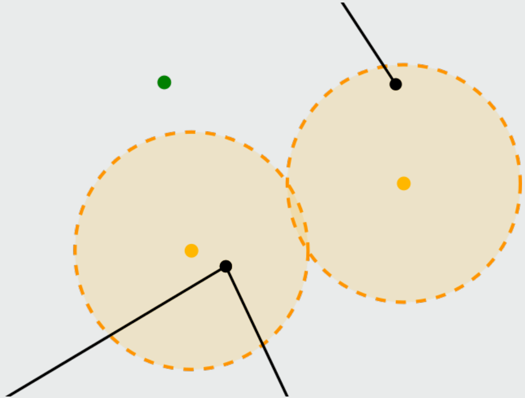


# Sparse Stable Trees (SST)

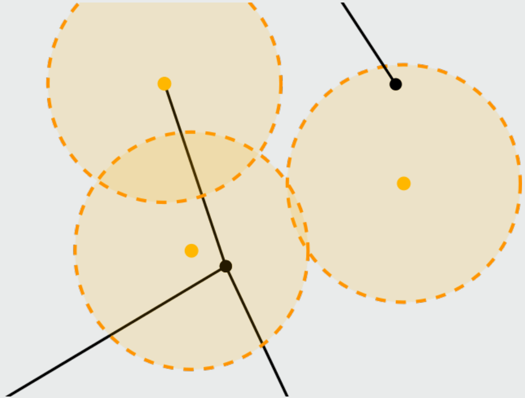
**Witness set**

Adding connections

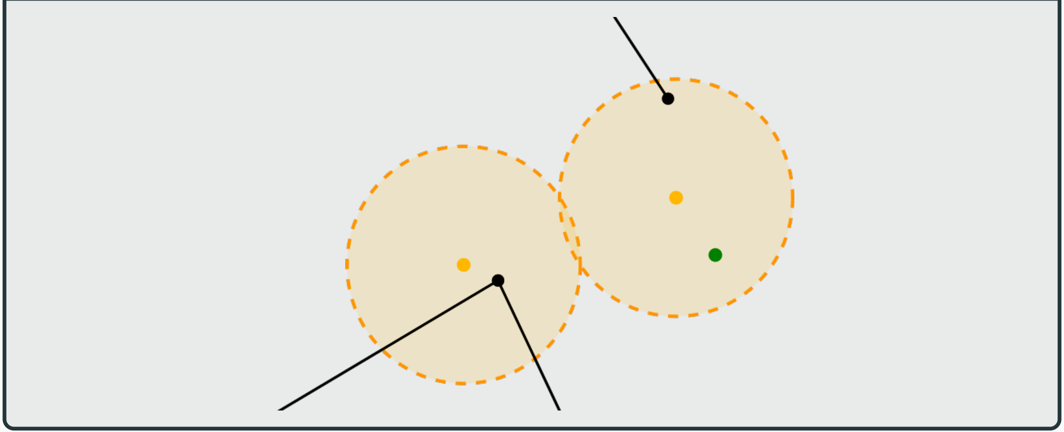
## Adding connection: Case 1 - No witness



## Adding connection: Case 1 - No witness

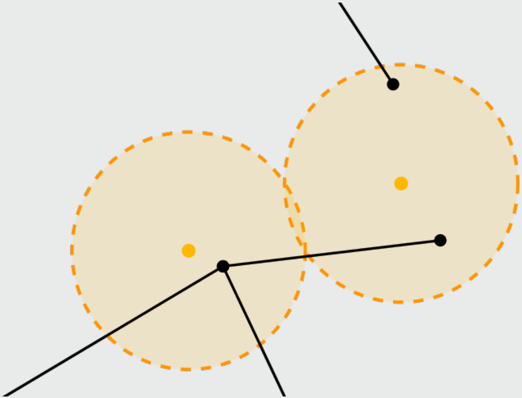


## Adding connection: Case 2 - Existing witness



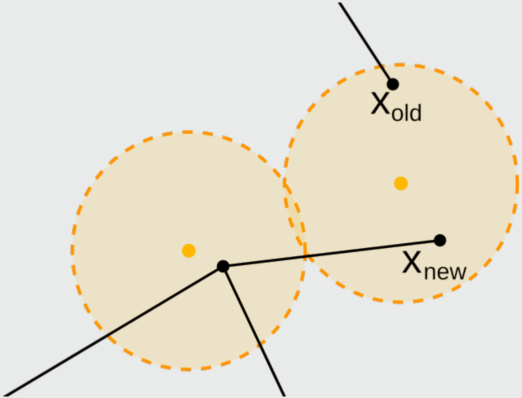
# Sparse Stable Trees (SST)

## Adding connection: Case 2 - Existing witness



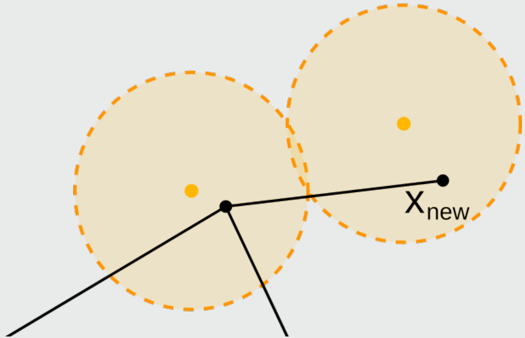
## Sparse Stable Trees (SST)

### Adding connection: Case 2 - Existing witness



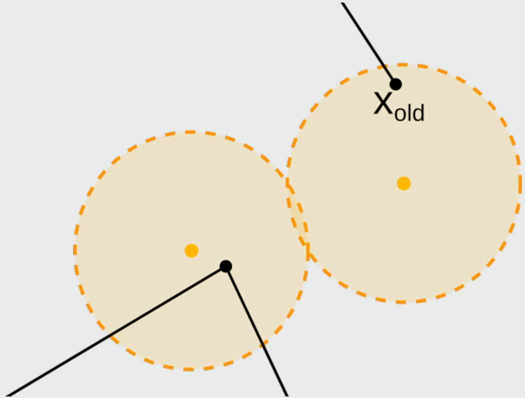
# Sparse Stable Trees (SST)

## Adding connection: Case 2a - Better cost



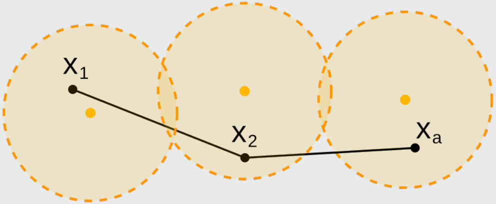
## Sparse Stable Trees (SST)

### Adding connection: Case 2b - Worse cost



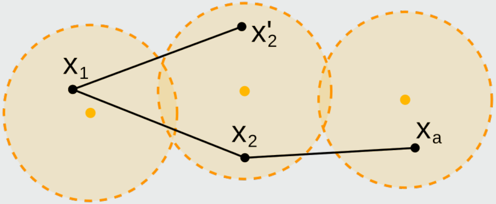
# Sparse Stable Trees (SST)

## Subtree pruning



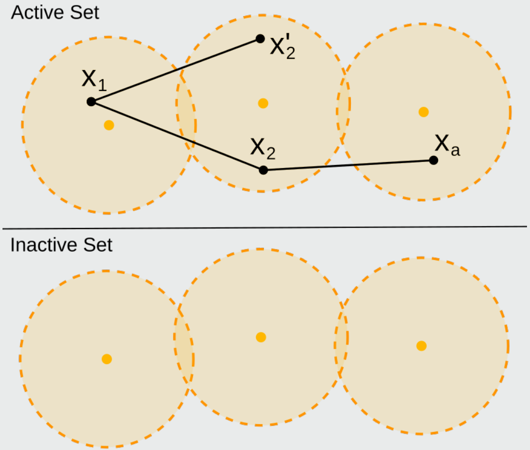
# Sparse Stable Trees (SST)

## Subtree pruning



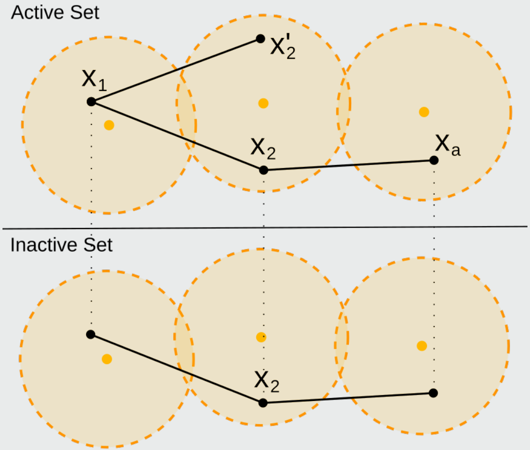
# Sparse Stable Trees (SST)

## Subtree pruning



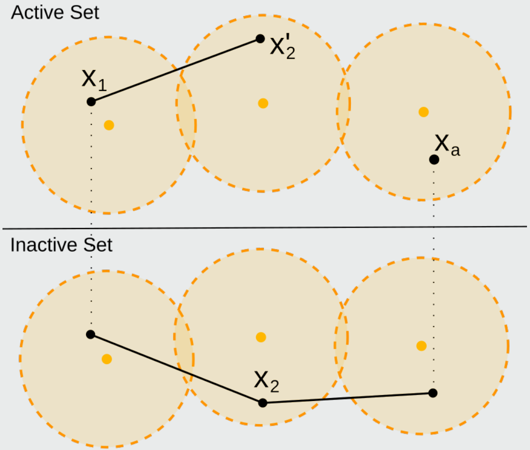
# Sparse Stable Trees (SST)

## Subtree pruning



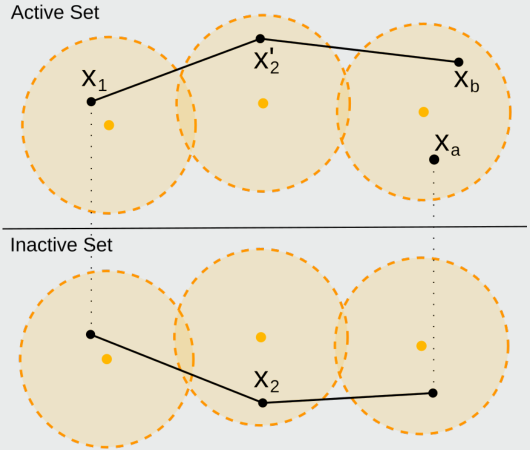
# Sparse Stable Trees (SST)

## Subtree pruning

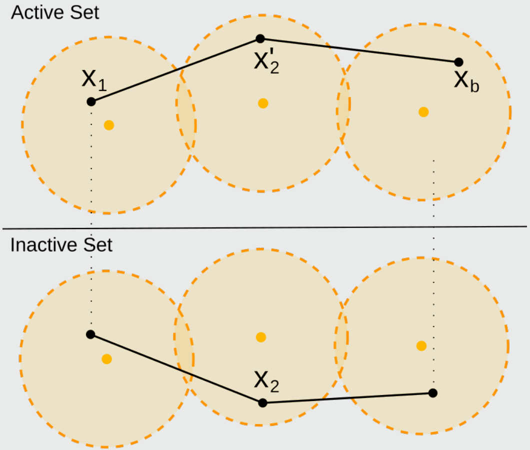


# Sparse Stable Trees (SST)

## Subtree pruning

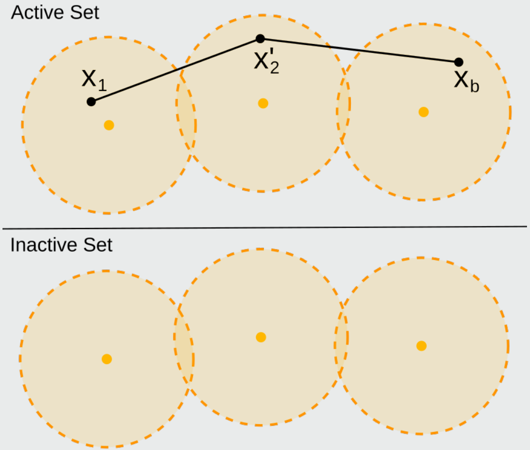


## Subtree pruning



# Sparse Stable Trees (SST)

## Subtree pruning



# Sparse Stable Trees (SST)

**Properties**

Sparse Stable Trees (SST*) is asymptotically near-optimal (AnO)

**Asymptotically near-optimal**

Planner finds a solution with cost at most $(1 + \epsilon)c*$

Y Li, Z Littlefield, KE Bekris, "Asymptotically Optimal Sampling-based Kinodynamic Planning", 2016

# Sparse Stable Trees (SST)

**Advantages**

- Selection always picks a locally optimal node
- Memory footprint is minimized

# Sparse Stable Trees (SST)

**Advantages**

- Selection always picks a locally optimal node
- Memory footprint is minimized

**Drawbacks**

- Need to choose a good $\delta_S$ for the witness radii
- AnO, not AO

# Optimal kinodynamic planning

## AO-x

# AO-x [2]

- AO-x is a meta algorithm
- Input is any feasible kinodynamic planner
- Idea: Convert the bounded-suboptimal version of optimal kinodynamic planning into a feasible kinodynamic problem
- Then iterate: Solve bounded-suboptimal version, compute best cost found, setup new bounded-suboptimal version with this cost, etc

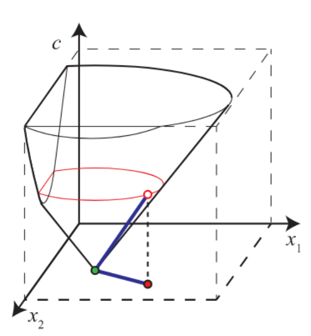
# State-Cost Space

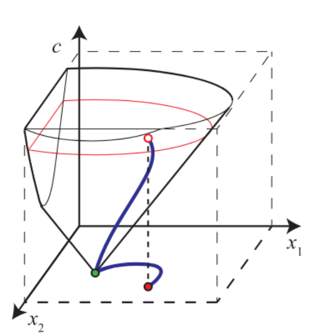
- Augment configuration space $\mathcal{Q}$ with a real cost dimension:

$$\mathcal{Q}' = \mathcal{Q} \times \mathbb{R}_0^+$$

- Augment dynamics $\mathbf{f}$, where $\mathbf{q}' = (\mathbf{q}, c)$:

$$\mathbf{f}'(\mathbf{q}', \mathbf{u}) = \Big(\mathbf{f}(\mathbf{q}, \mathbf{u}), \Delta c\Big)$$



## AO-x: Pseudo Code [2]

**Algorithm 2:** Asymptotically-optimal($\mathcal{P}$, $\mathcal{A}$, $n$).

Run $\mathcal{A}(\mathcal{P}_\infty)$ to obtain a first path $y_0$. If no solution exists, report '$\mathcal{P}$ has no solution.'
Let $c_0 = C(y_0)$.
**for** $i = 1, 2, \ldots, n$ **do**
    Run $\mathcal{A}(\mathcal{P}_{c_{i-1}})$ to obtain a new solution $y_i$.
    Let $c_i = C(y_i)$.
**return** $y_n$

- $\mathcal{A}$ is typically (kinodynamic) RRT or EST
- $\mathcal{P}_{\bar{c}}$ is a problem instance with cost bound $\bar{c}$

# AO-x: Proof of asymptotic optimality (AO)

**Assumptions**

1. $\mathcal{A}$ terminates in finite time, if solution within given bound $\bar{c}$ exists
2. $\mathcal{A}$ reduces cost by a nonnegligible amount.

$$E[c(y_i)|\bar{c}] - c^* \leq (1 - \omega)(\bar{c} - c^*) \qquad \text{for } \omega > 0,$$

where $c^*$ is the optimal cost and $\bar{c}$ the cost limit

# AO-x: Proof of asymptotic optimality (AO)

**Proof Goal**

Let $S_0, \ldots, S_n$ be random variables for $c(y_i) - c^*$. Then for any $\epsilon > 0$ we have:

$$\lim_{n \to \infty} P(S_n \geq \epsilon) = 0$$

Proof helpers:

- Markov inequality: $P(S_n \geq \epsilon) \leq E[S_n]/\epsilon$
- Assumption 2 (cost reduction by nonnegligible amount): $E[S_n | s_{n-1}] \leq (1 - \omega) s_{n-1}$

## AO-x: Proof of asymptotic optimality (AO)

Use $E[S_n|s_{n-1}] \leq (1-\omega)s_{n-1}$:

$$
\begin{aligned}
E[S_n] &= \int E[S_n|s_{n-1}]P(s_{n-1})ds_{n-1} \\
&\leq \int (1-\omega)s_{n-1}P(s_{n-1})ds_{n-1} \\
&= (1-\omega)\int s_{n-1}P(s_{n-1})ds_{n-1} \\
&= (1-\omega)E[S_{n-1}] \\
&= (1-\omega)^n E[S_0]
\end{aligned}
$$

# AO-x: Proof of asymptotic optimality (AO)

Use $E[S_n|s_{n-1}] \leq (1-\omega)s_{n-1}$:

$$\begin{aligned} E[S_n] &= \int E[S_n|s_{n-1}]P(s_{n-1})ds_{n-1} \\ &\leq \int (1-\omega)s_{n-1}P(s_{n-1})ds_{n-1} \\ &= (1-\omega)\int s_{n-1}P(s_{n-1})ds_{n-1} \\ &= (1-\omega)E[S_{n-1}] \\ &= (1-\omega)^n E[S_0] \end{aligned}$$

Use Markov inequality:

$$\begin{aligned} P(S_n \geq \epsilon) &\leq E[S_n]/\epsilon \\ P(S_n \geq \epsilon) &\leq (1-\omega)^n E[S_0]/\epsilon \end{aligned}$$

# AO-x: Proof of asymptotic optimality (AO)

Use $E[S_n|s_{n-1}] \leq (1-\omega)s_{n-1}$:

$$\begin{aligned}
E[S_n] &= \int E[S_n|s_{n-1}]P(s_{n-1})ds_{n-1} \\
&\leq \int (1-\omega)s_{n-1}P(s_{n-1})ds_{n-1} \\
&= (1-\omega)\int s_{n-1}P(s_{n-1})ds_{n-1} \\
&= (1-\omega)E[S_{n-1}] \\
&= (1-\omega)^n E[S_0]
\end{aligned}$$

Use Markov inequality:

$$P(S_n \geq \epsilon) \leq E[S_n]/\epsilon$$
$$P(S_n \geq \epsilon) \leq (1-\omega)^n E[S_0]/\epsilon$$

Take the limit:

$$\begin{aligned}
\lim_{n\to\infty} P(S_n \geq \epsilon) &\leq (1-\omega)^n E[S_0]/\epsilon \\
&= 0
\end{aligned}$$

# AO-x

**Advantages**

- Planner agnostic
- Enhances theoretical properties (AO)

# AO-x

**Advantages**

- Planner agnostic
- Enhances theoretical properties (AO)

**Drawbacks**

- By default, no re-use of data between iterations
- Unknown convergence rate (rather poor empirically)

# More Tree-based Geometric Motion Planning: EST, RRT-Connect

# EST: Expansive Space Trees (1) [4]

- Key insight: use explicit function rather than Voronoi bias for exploration

```
def EST(𝒬, 𝒲_free, ℬ(·), q_start, 𝒬_goal):
  𝒯 = (𝒱, ℰ) = ({q_start}, ∅)
  while True:
    q = randomly choose from 𝒱 with
      ↪ probability π_𝒯(q)
    p = random configuration near q
    if path q to p feasible:
      𝒱 = 𝒱 ∪ {p}
      ℰ = ℰ ∪ {path q to p}
      if p ∈ 𝒬_goal:
        return solution
```

# EST: Expansive Space Trees (1) [4]

- Key insight: use explicit function rather than Voronoi bias for exploration

```
def EST(𝒬, 𝒲_free, ℬ(·), q_start, 𝒬_goal):
  𝒯 = (𝒱, ℰ) = ({q_start}, ∅)
  while True:
    q =randomly choose from 𝒱 with
    ↪  probability π_𝒯(q)
    p =  random configuration near q
    if path q to p feasible:
      𝒱 = 𝒱 ∪ {p}
      ℰ = ℰ ∪ {path q to p}
      if p ∈ 𝒬_goal:
        return solution
```

# EST: Expansive Space Trees (1) [4]

- Key insight: use explicit function rather than Voronoi bias for exploration

```
def EST(𝒬, 𝒲_free, ℬ(·), q_start, 𝒬_goal):
  𝒯 = (𝒱, ℰ) = ({q_start}, ∅)
  while True:
    q = randomly choose from 𝒱 with
      ↪ probability π_𝒯(q)
    p = random configuration near q
    if path q to p feasible:
      𝒱 = 𝒱 ∪ {p}
      ℰ = ℰ ∪ {path q to p}
      if p ∈ 𝒬_goal:
        return solution
```

# EST: Expansive Space Trees (1) [4]

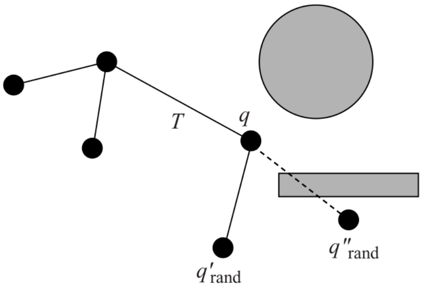
- Key insight: use explicit function rather than Voronoi bias for exploration

```
def EST(𝒬, 𝒲_free, ℬ(·), q_start, 𝒬_goal):
  𝒯 = (𝒱, ℰ) = ({q_start}, ∅)
  while True:
    q = randomly choose from 𝒱 with
      ↪ probability π_𝒯(q)
    p = random configuration near q
    if path q to p feasible:
      𝒱 = 𝒱 ∪ {p}
      ℰ = ℰ ∪ {path q to p}
      if p ∈ 𝒬_goal:
        return solution
```

# EST: Expansive Space Trees (1) [4]

- Key insight: use explicit function rather than Voronoi bias for exploration

```
def EST(𝒬, 𝒲_free, ℬ(·), q_start, 𝒬_goal):
  𝒯 = (𝒱, ℰ) = ({q_start}, ∅)
  while True:
    q =randomly choose from 𝒱 with
    ↪  probability π_𝒯(q)
    p = random configuration near q
    if path q to p feasible:
      𝒱 = 𝒱 ∪ {p}
      ℰ = ℰ ∪ {path q to p}
      if p ∈ 𝒬_goal:
        return solution
```




Source: [3]

# EST: Expansive Space Trees (2)

- Choice of probability density function $\pi_{\mathcal{T}}(\mathbf{q})$: Good exploration of $\mathcal{Q}_{\mathit{free}}$, e.g., proportional to dispersion
- $\pi_{\mathcal{T}}(\mathbf{q})$ often changes during the search

# EST: Expansive Space Trees (2)

- Choice of probability density function $\pi_{\mathcal{T}}(\mathbf{q})$: Good exploration of $\mathcal{Q}_{free}$, e.g., proportional to dispersion
- $\pi_{\mathcal{T}}(\mathbf{q})$ often changes during the search

**Online Dispersion Estimation**

- Discretize $\mathcal{Q}$ in a grid
- Count the number of $\mathbf{q} \in \mathcal{V}$ that belong to each grid cell
- Probability $\pi_{\mathcal{T}}(\mathbf{q})$ is inverse proportional to the number corresponding to the grid cell of $\mathbf{q}$

# EST: Expansive Space Trees (2)

- Choice of probability density function $\pi_{\mathcal{T}}(\mathbf{q})$: Good exploration of $\mathcal{Q}_{free}$, e.g., proportional to dispersion
- $\pi_{\mathcal{T}}(\mathbf{q})$ often changes during the search

**Online Dispersion Estimation**

- Discretize $\mathcal{Q}$ in a grid
- Count the number of $\mathbf{q} \in \mathcal{V}$ that belong to each grid cell
- Probability $\pi_{\mathcal{T}}(\mathbf{q})$ is inverse proportional to the number corresponding to the grid cell of $\mathbf{q}$

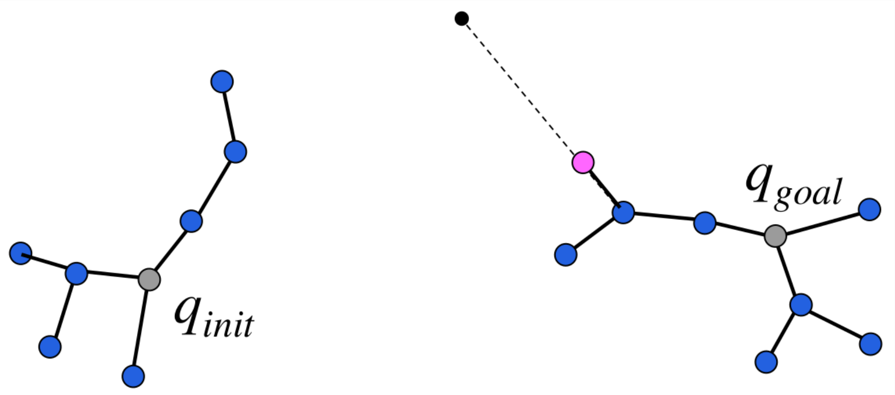
**EST Main Challenge**

Difficult to define $\pi_{\mathcal{T}}(\mathbf{q})$ efficiently.

## RRT-Connect (1) [5]

- Bidirectional search: Use two trees: one rooted at $\mathbf{q}_{start}$, one rooted at $\mathbf{q}_{goal}$
- Try to connect both trees

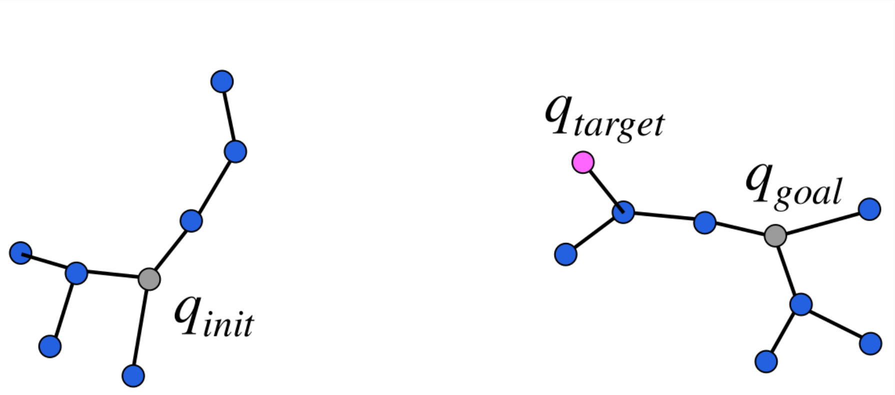
# RRT-Connect (2)




Source: [6]

- Sample $\mathbf{q}_{rand}$ and Extend the goal tree (right side)

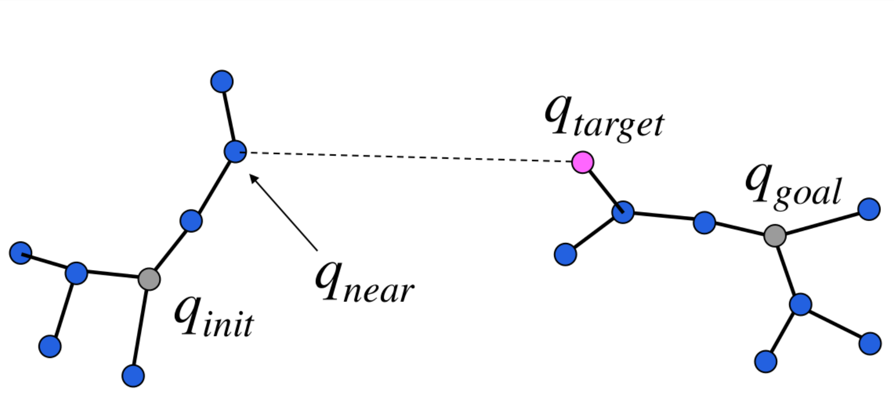
# RRT-Connect (3)




Source: [6]

- $\mathbf{q}_{target}$ is now the goal for the init tree (left side)

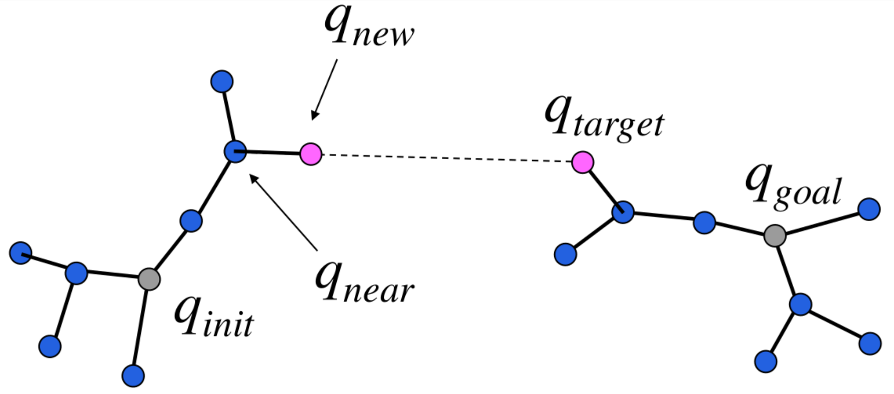
# RRT-Connect (4)




Source: [6]

- Calculate $\mathbf{q}_{near}$ (closest node to $\mathbf{q}_{target}$ in init tree)

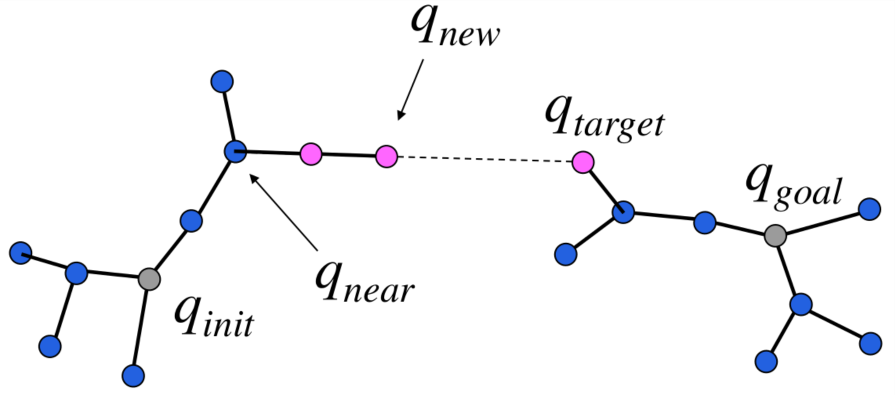
# RRT-Connect (5)




Source: [6]

- Try to connect $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$

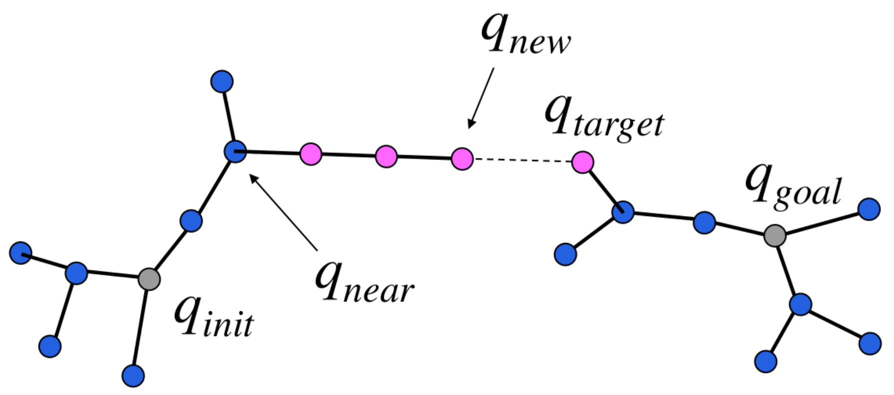
# RRT-Connect (5)




Source: [6]

- Try to connect $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$

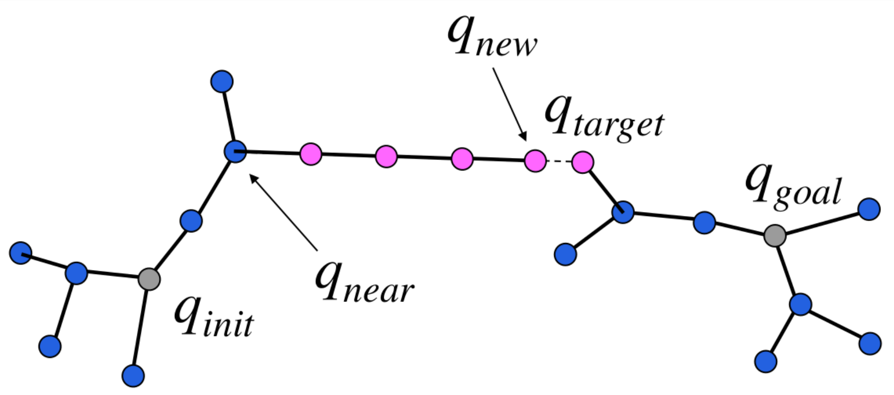
# RRT-Connect (5)




Source: [6]

- Try to connect $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$

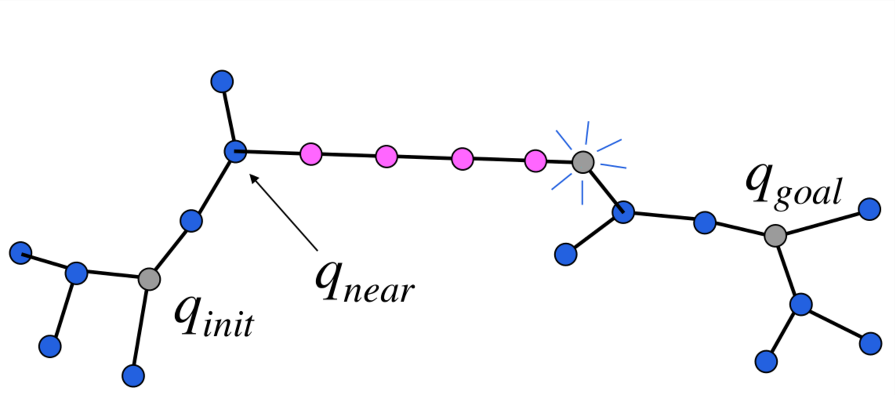
# RRT-Connect (5)




Source: [6]

- Try to connect $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$

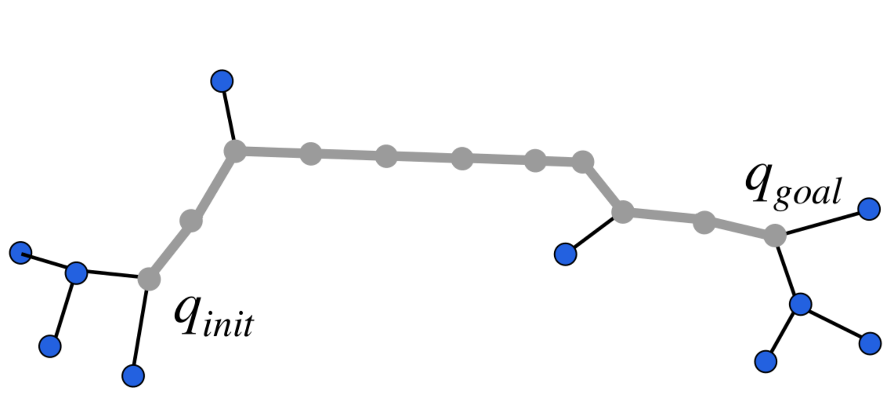
# RRT-Connect (5)




Source: [6]

- Try to connect $\mathbf{q}_{near}$ and $\mathbf{q}_{target}$

# RRT-Connect (6)




Source: [6]

- Solution is the path connecting $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$

# RRT-Connect (7)

Pseudo-Code from the original paper:

RRT_CONNECT_PLANNER($q_{init}, q_{goal}$)

- $\mathcal{T}_a$.init($q_{init}$); $\mathcal{T}_b$.init($q_{goal}$);
- **for** $k = 1$ **to** $K$ **do**
  - $q_{rand} \leftarrow$ RANDOM_CONFIG();
  - **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
    - **if** (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
      - Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
  - SWAP($\mathcal{T}_a, \mathcal{T}_b$);
- Return *Failure*

Source: [5]

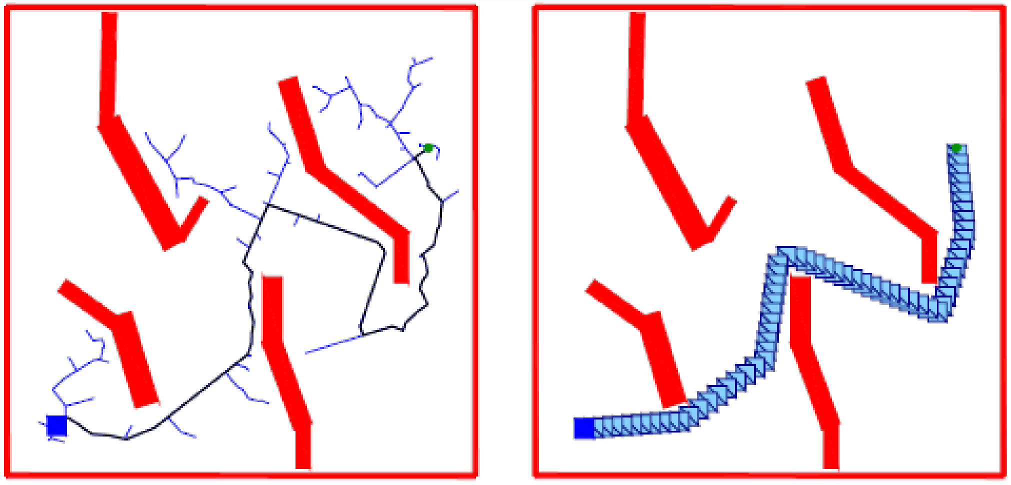
Pseudo-Code from the original paper:

```
RRT_CONNECT_PLANNER(q_init, q_goal)
    T_a.init(q_init); T_b.init(q_goal);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        if not (EXTEND(T_a, q_rand) = Trapped) then
            if (CONNECT(T_b, q_new) = Reached) then
                Return PATH(T_a, T_b);
        SWAP(T_a, T_b);
    Return Failure
```

Source: [5]

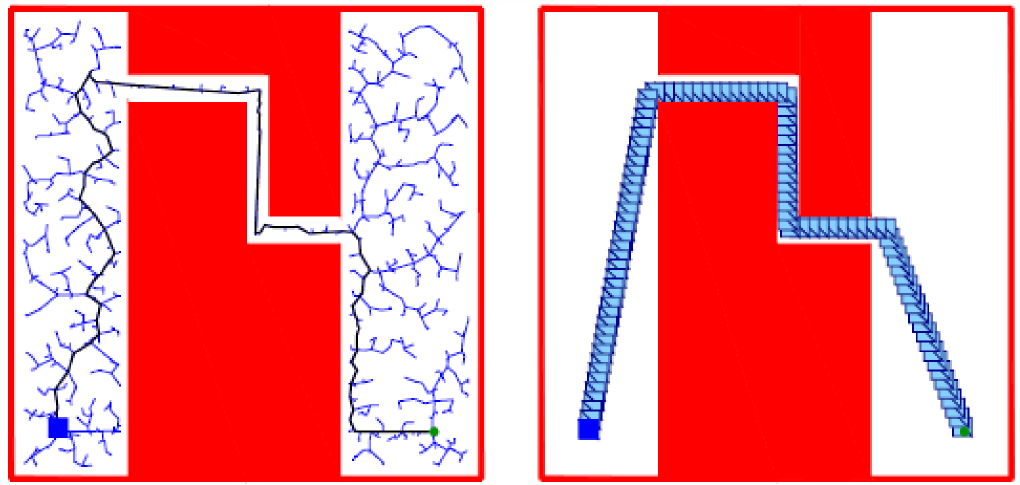
What is the purpose of SWAP here?

# RRT-Connect Examples (1)



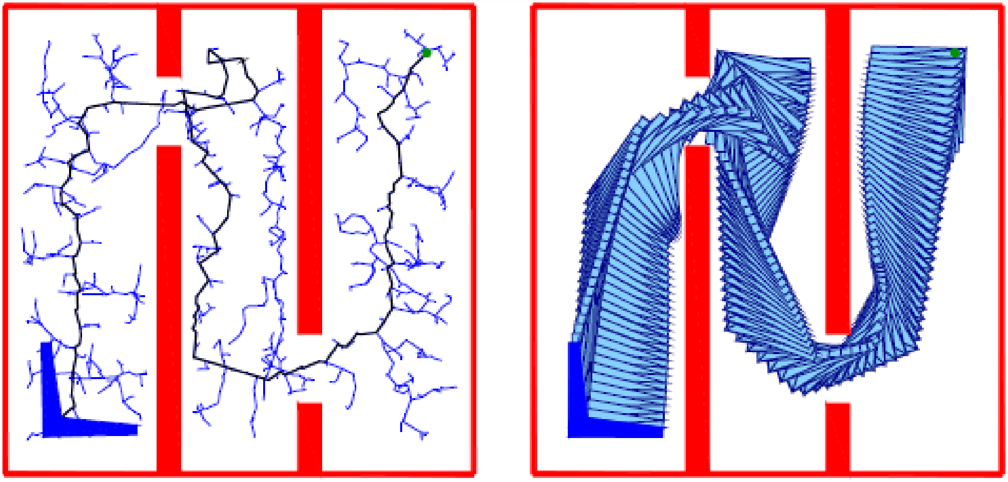
Source: [5]

## RRT-Connect Examples (2)



Source: [5]

# RRT-Connect Examples (3)



Source: [5]

# Asymptotic Optimal Geometric Motion Planning: PRM*

**Algorithm 4:** PRM$^*$

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n}$; $E \leftarrow \emptyset$;
**foreach** $v \in V$ **do**
    $U \leftarrow \texttt{Near}(G = (V, E), v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d}) \setminus \{v\}$;
    **foreach** $u \in U$ **do**
        **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E)$;

# PRM*

**Algorithm 4:** PRM*

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
**foreach** $v \in V$ **do**
    $U \leftarrow \texttt{Near}(G = (V, E), v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d}) \setminus \{v\}$;
    **foreach** $u \in U$ **do**
        **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E)$;

def GenPRM($\mathcal{Q}, \mathcal{W}_{free}, \mathcal{B}(\cdot), N$):
    # ...
    for $\mathbf{q}$ in $\mathcal{V}$:
        for $\mathbf{p}$ in $\{\mathbf{p} \in \mathcal{V} : \text{isNeighbor}(\mathbf{p}, \mathbf{q})\}$:
            if path $\mathbf{q}$ to $\mathbf{p}$ feasible:
                $\mathcal{E} = \mathcal{E} \cup \{\text{path } \mathbf{q} \text{ to } \mathbf{p}\}$
    return $\mathcal{G}$

- Pseudo code from [7]
- Consistent with our previous pseudo code of PRM (lecture 5)

**Algorithm 4:** PRM*

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
**foreach** $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d}) \setminus \{v\}$;
  **foreach** $u \in U$ **do**
    **if** $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
**return** $G = (V, E)$;

```
def GenPRM($\mathcal{Q}, \mathcal{W}_{free}, \mathcal{B}(\cdot), N$):
  # ...
  for q in $\mathcal{V}$:
    for p in $\{\mathbf{p} \in \mathcal{V} : \text{isNeighbor}(\mathbf{p}, \mathbf{q})\}$:
      if path q to p feasible:
        $\mathcal{E} = \mathcal{E} \cup \{\text{path } \mathbf{q} \text{ to } \mathbf{p}\}$
  return $\mathcal{G}$
```

- Pseudo code from [7]
- Consistent with our previous pseudo code of PRM (lecture 5)
- Neighbors are computed using the dynamic radius, depending on $|\mathcal{V}|$

# PRM*

**Algorithm 4:** PRM*

```
V ← {x_init} ∪ {SampleFree_i}_{i=1,...,n}; E ← ∅;
foreach v ∈ V do
    U ← Near(G = (V, E), v, γ_PRM(log(n)/n)^{1/d}) \ {v};
    foreach u ∈ U do
        if CollisionFree(v, u) then E ← E ∪ {(v, u), (u, v)}
return G = (V, E);
```

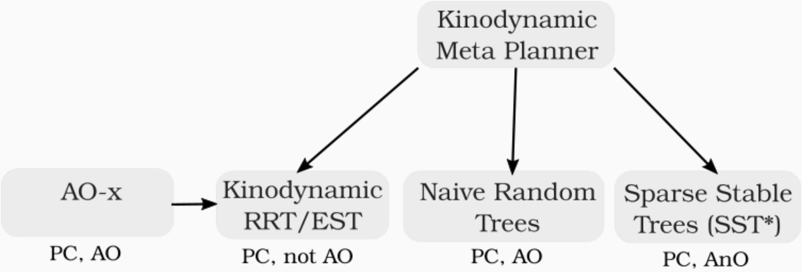
```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
  # ...
  for q in 𝒱:
    for p in {p ∈ 𝒱 : isNeighbor(p, q)}:
      if path q to p feasible:
        ℰ = ℰ ∪ {path q to p}
  return 𝒢
```

- Pseudo code from [7]
- Consistent with our previous pseudo code of PRM (lecture 5)
- `Neighbors` are computed using the dynamic radius, depending on $|\mathcal{V}|$

How does this work for parallel pre-processing and query?

# Conclusion

- Kinodynamic planners: kinodynamic RRT/EST, SST(*), AO-x




- Geometric planners: EST, RRT-Connect, PRM*

**Next Time**

- Open Motion Planning Library (OMPL)

# Suggested Reading


1. Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. "Asymptotically Optimal Sampling-Based Kinodynamic Planning". In: *I. J. Robotics Res.* 35.5 (2016), pp. 528–564. DOI: 10.1177/0278364915614386
2. Kris Hauser and Yilun Zhou. "Asymptotically Optimal Planning by Feasible Kinodynamic Planning in a State-Cost Space". In: *IEEE Trans. Robotics* 32.6 (2016), pp. 1431–1443. DOI: 10.1109/TRO.2016.2602363

[1] Yanbo Li, Zakary Littlefield, and Kostas E. Bekris. "Asymptotically Optimal Sampling-Based Kinodynamic Planning". In: *I. J. Robotics Res.* 35.5 (2016), pp. 528–564. DOI: 10.1177/0278364915614386.

[2] Kris Hauser and Yilun Zhou. "Asymptotically Optimal Planning by Feasible Kinodynamic Planning in a State-Cost Space". In: *IEEE Trans. Robotics* 32.6 (2016), pp. 1431–1443. DOI: 10.1109/TRO.2016.2602363.

[3] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Intelligent Robotics and Autonomous Agents Series. Cambridge, MA, USA: A Bradford Book, 2005. 630 pp. ISBN: 978-0-262-03327-5.

[4] D. Hsu, J.-C. Latombe, and R. Motwani. “Path Planning in Expansive Configuration Spaces”. In: *International Conference on Robotics and Automation*. Vol. 3. Apr. 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371.

[5] J.J. Kuffner and S.M. LaValle. “RRT-Connect: An Efficient Approach to Single-Query Path Planning”. In: *International Conference on Robotics and Automation*. Vol. 2. Apr. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.

[6] Dmitry Berenson. “Motion Planning: Robotics and Beyond”. In: (2021). URL: https://web.eecs.umich.edu/~dmitryb/courses/winter2021motionplanning/index.html.

[7] Sertac Karaman and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning". In: *International Journal of Robotics Research (IJRR)* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761.