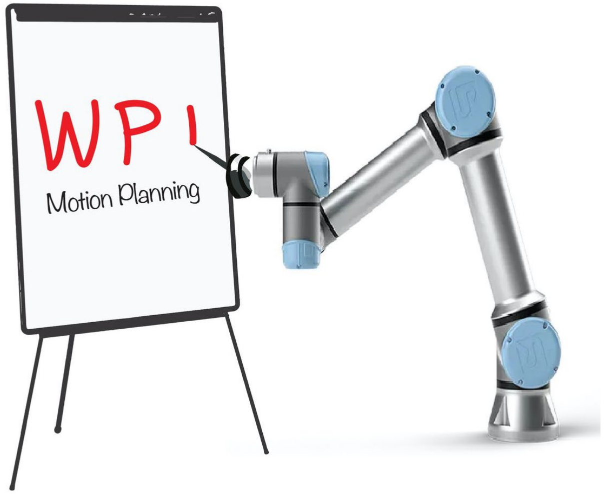
RBE550

Motion Planning

Kinodynamic Planning



Constantinos Chamzas

www.cchamzas.com

www.elpislab.org

Disclaimer

The slides are a compilation of work based on notes and slides mainly from from Erion Plaku, but also Lydia Kavraki, Constantinos Chamzas, Howie Choset, Morteza Lahijanian, David Hsu, Greg Hager, Mark Moll, G. Ayorkor Mills-Tetty, Hyungpil Moon, Zack Dodds, Zak Kingston, Nancy Amato, Steven Lavalle, Seth Hutchinson, George Kantor, Dieter Fox, Vincent Lee-Shue Jr., Prasad Narendra Atkar, Kevin Tantiseviand, Bernice Ma, David Conner, and students taking comp450/comp550 at Rice University.

Last Time Recap

- Kinematic constraints
- Dynamics constraints
- Integration of the dynamics
- Decoupled/Native Approach
- Roadmap-based methods for kinodynamic systems
- Tree-based methods for kinodynamic systems

Overview

- Kinodynamic Formulation Recap
- Roadmap-based methods for kinodynamic systems
- Dubins Curves
- Tree-based methods for kinodynamic systems
- Planning without distances with KPIECE
- Planning KinodynamicOptimal Paths with SST*

Kinematics for Wheeled System – Simple Car

Simple car

- How should we control the car?
  - Setting the speed v, i.e.,

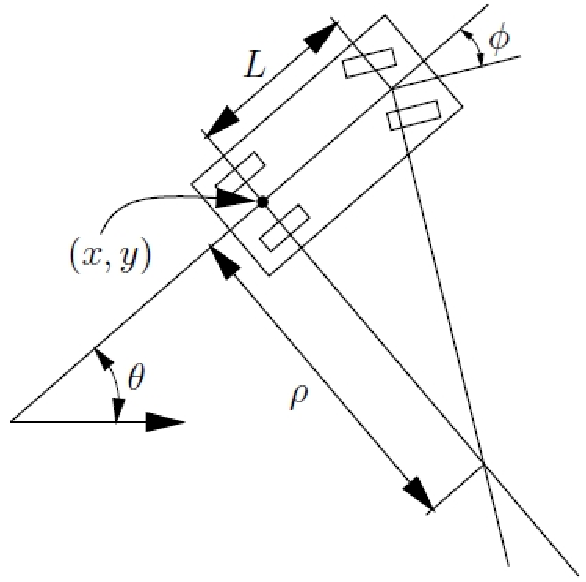
$$u_v = v$$

  - Setting the steering angle ϕ, i.e.,

$$u_\phi = \phi$$

  - Putting it all together:
    - Input controls: u_v (speed) and u_ϕ (steering angle)
    - Equation of motion:

$$\dot{x} = u_v \cos\theta$$
$$\dot{y} = u_v \sin\theta$$
$$\dot{\theta} = \frac{u_v}{L} \tan u_\phi$$



Variations of the Simple Car Model

- Equation of motion: $\dot{x} = u_v \cos\theta$ $\dot{y} = u_v \sin\theta$ $\dot{\theta} = \frac{u_v}{L} \tan u_\phi$

Different bounds give different models:

- **Tricycle**
  - $u_v \in [-1, 1]$ and $u_\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
  - Can it rotate in place?
- **Standard simple car**
  - $u_v \in [-1,1]$
  - $u_\phi \in (-\phi_{\max}, \phi_{\max})$ for some $\phi_{\max} < \frac{\pi}{2}$
- **Reeds-Shepp car**
  - $u_v \in \{-1, 0, 1\}$ (i.e., “reverse”, “park”, “forward”)
  - u_ϕ same as in the standard simple car
- **Dubins car**
  - $u_v \in \{0, 1\}$ (i.e., “park”, “forward”)
  - u_ϕ same as in the standard simple car

Motion Planning with Kinodynamical Constraints

- Planning Problem, given:
  - State space X
  - Control space U
  - Equations of motion as differential equations: $f: X \times U \rightarrow \dot{X}$
  - State-validity function $\mathrm{valid}\colon\ X \rightarrow \{\mathrm{true}, \mathrm{false}\}$, e.g., check collision
  - Goal function $\mathrm{goal}\colon\ X \rightarrow \{\mathrm{true}, \mathrm{false}\}$
  - Initial state $\boldsymbol{x}_0$
- Compute
  - a control trajectory $u\colon [0, T] \rightarrow U$ such that the resulting state trajectory $\boldsymbol{x}\colon [0, T] \rightarrow X$ obtained by integration is valid and reaches the goal:

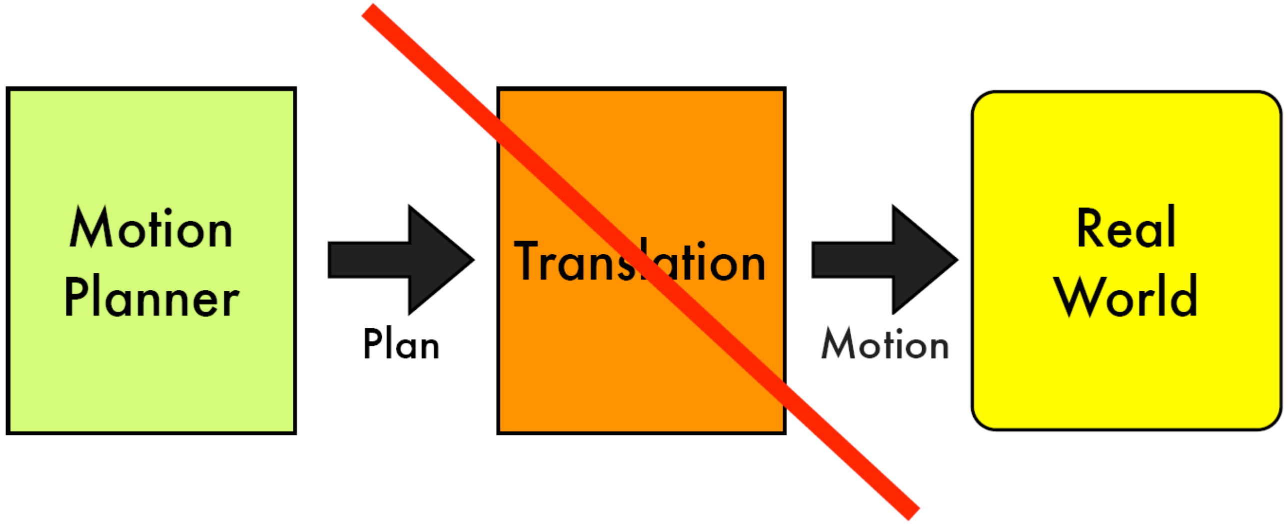
$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_0^t f(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau))\, d_\tau$$

s.t.

$$\forall t \in [0, T]\colon\ \mathrm{valid}\big(x(t)\big) = \mathrm{true}$$

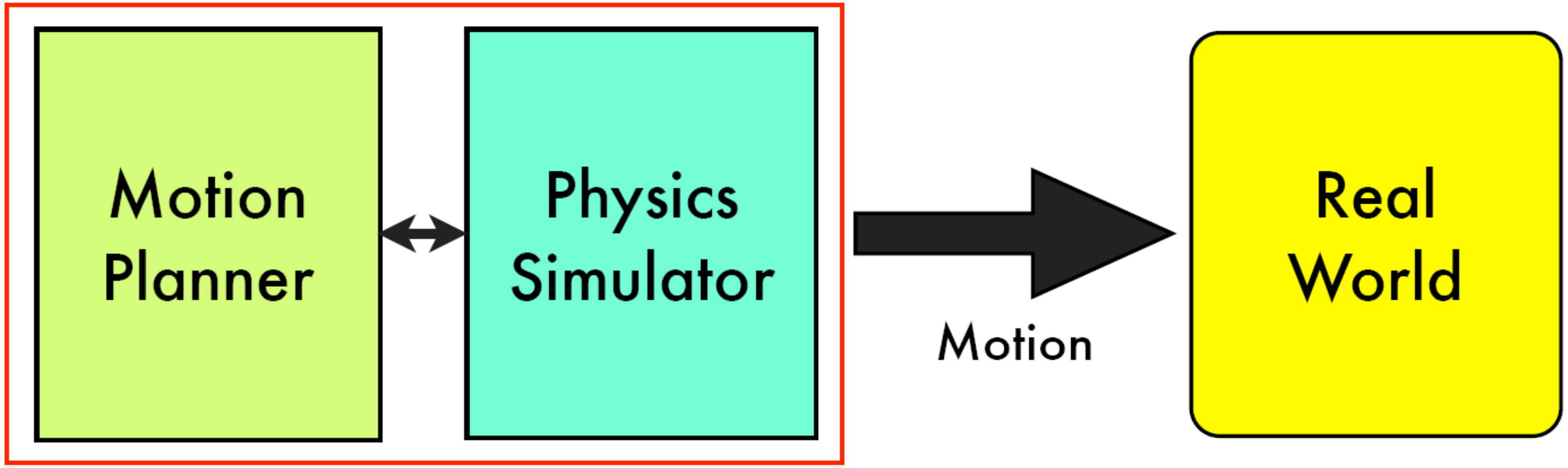
$$\exists t \in [0, T]\colon\ \mathrm{goal}\big(x(t)\big) = \mathrm{true}$$

Shortcomings of Decoupled Motion Planning



- Motions are often low quality or inadmissible
- Translating “collision-free” paths to physical motion is hard

Native Approach



- Integrate physical/differential constraints in the motion planner
- How can we do that we sampling based planners?

Kinodynamic Motion Planning with Roadmap Methods

0. Initialization

- Add $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\text{goal}}$ to roadmap vertex set V

1. Sampling

- Repeat several times

  $\boldsymbol{x} \leftarrow \text{StateSample}()$

  If $\text{IsStateValid}(\boldsymbol{x})$: add $\boldsymbol{x}$ to roadmap

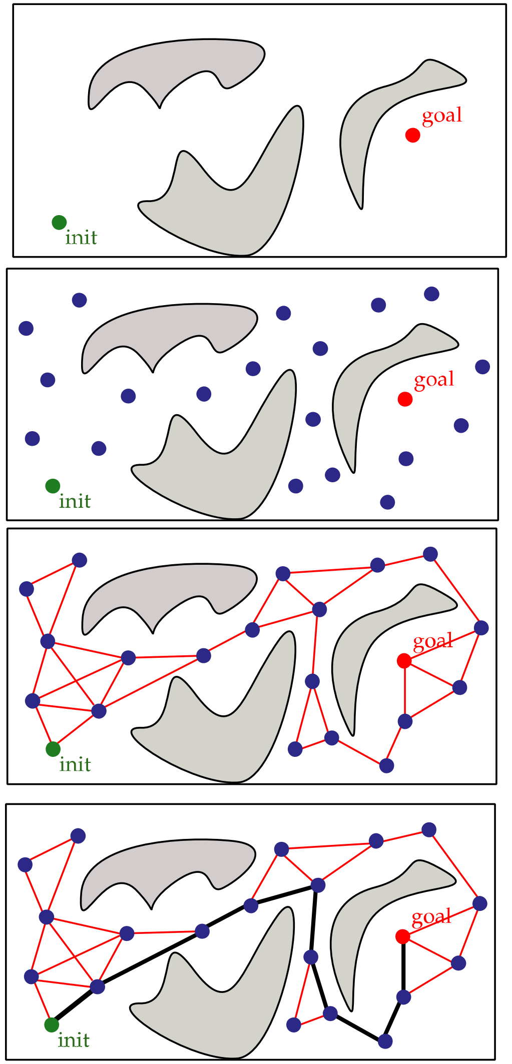
2. Connect Samples

$\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$

If $\text{IsTrajectoryValid}(\lambda) = \text{true}$: add edge to roadmap

3. Graph Search

- Search graph (V, E) for path from x_0 to x_{goal}



Kinodynamic Motion Planning with Roadmap Methods

0. **Initialization**
   - Add $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\text{goal}}$ to roadmap vertex set V
1. **Sampling**
   - Repeat several times

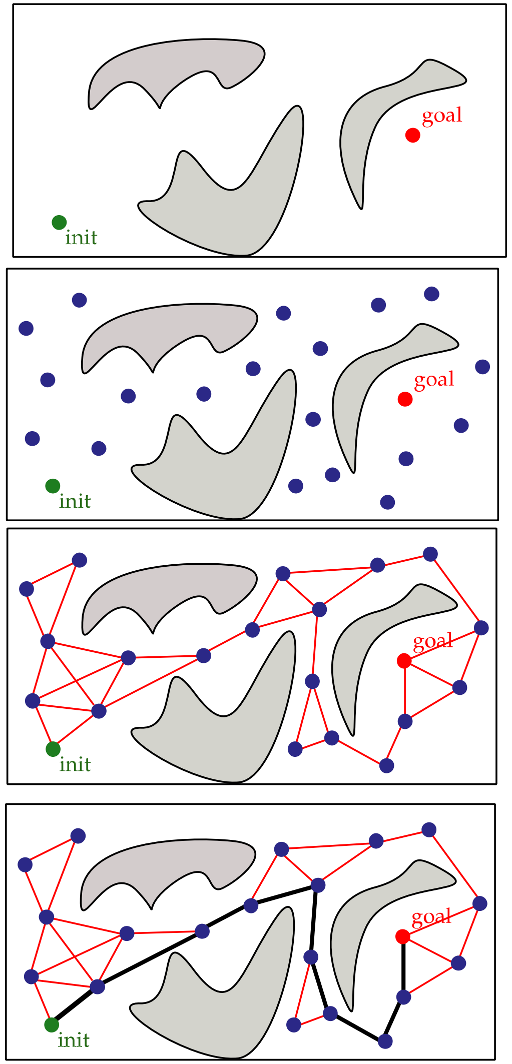
     $\boldsymbol{x} \leftarrow \text{StateSample}()$

     If $\text{IsStateValid}(\boldsymbol{x})$: add $\boldsymbol{x}$ to roadmap
2. **Connect Samples**

   For all neighbors $(\mathrm{x_a}, \mathrm{x_b})$

   $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$

   If $\text{IsTrajectoryValid}(\lambda) = \text{true}$: add edge to roadmap
3. **Graph Search**
   - Search graph (V, E) for path from x_0 to x_{goal}



Kinodynamic Motion Planning with Roadmap Methods

- $x \leftarrow \text{StateSample}()$ ✔
  - Generate random values for all state components
- $\text{IsStateValid}(\boldsymbol{x})$ ✔
  - place robot in the position and orientation (C-space) components of the state
  - check if the robot collides with the obstacles
  - check if velocity and other state components are within desired bounds
- $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$?
  - linear interpolation between x_a and x_b does **NOT** work and breaks underlying differential constraints
  - need to find control function $u : [0, T] \rightarrow U$ such that applying u to x_a for T time units ends at x_b
  - Known as **two-point boundary value problem (BVP)**
  - Cannot always be solved analytically, and numerical solutions increase computational cost

Dynamic Car:

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} v \cos\theta \\ v \sin\theta \\ (v/L)\tan\phi \\ u_1 \\ u_2 \end{pmatrix}$$

Kinodynamic Motion Planning with Roadmap Methods

0. Initialization

- Add $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\text{goal}}$ to roadmap vertex set V

1. Sampling

- Repeat several times

  $\boldsymbol{x} \leftarrow \text{StateSample}()$

  If $\text{IsStateValid}(\boldsymbol{x})$: add $\boldsymbol{x}$ to roadmap

2. Connect Samples

For all neighbors $(\mathrm{x_a}, \mathrm{x_b})$

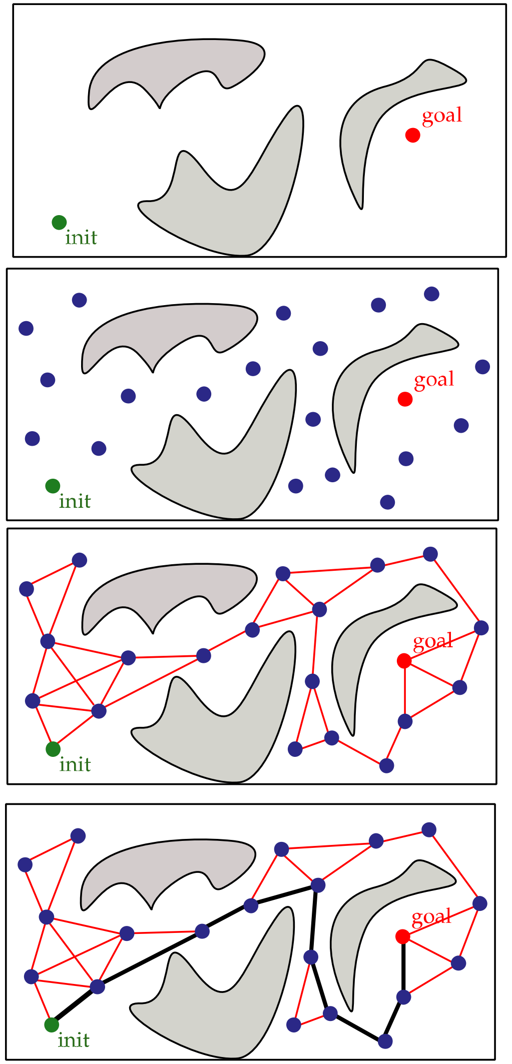
$\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$

BVP

If $\text{IsTrajectoryValid}(\lambda) = \text{true}$: add edge to roadmap

3. Graph Search

- Search graph (V, E) for path from x_0 to x_{goal}



Optimal Curves for Wheeled Vehicles

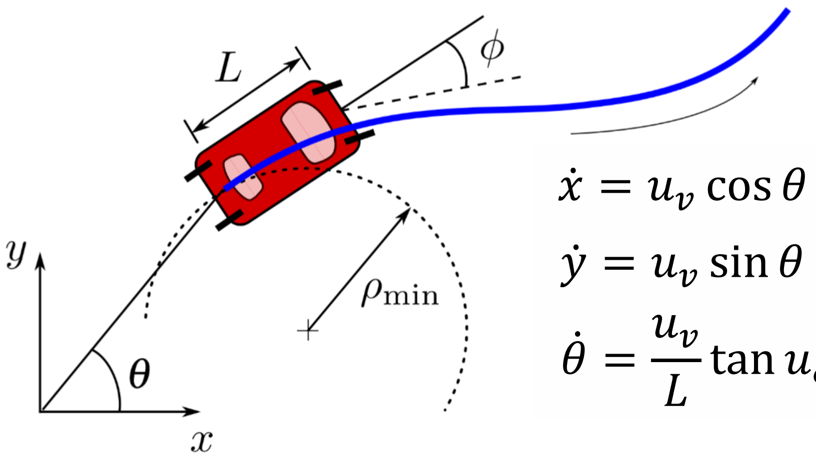
For some of the kinematic wheeled vehicle models the shortest path between a pair of configurations has been completely characterized.

Some of them include:

- Dubins Car (6 Dubins Curves)
- Reeds Shepp Car (42 Reeds-Shepp Curves)
- Differential Drive (9 Balkcom-Mason Curves)

14

Dubins Curves



$$\dot{x} = u_v \cos\theta$$
$$\dot{y} = u_v \sin\theta$$
$$\dot{\theta} = \frac{u_v}{L} \tan u_\phi$$

- Dubins car bounds
  - $u_v \in \{0, 1\}$ (i.e., “park”, “forward”)
  - $u_\phi \in (-\phi_{\max}, \phi_{\max})$ for some $\phi_{\max} < \frac{\pi}{2}$

1. We only need to stop at the goal so:

   $u_v = 1$ (i.e., “always forward”)

2. The max steering angle $\phi_{\max}$ imposes the minimum turning radius ρ_{min} and due to this the optimal path can be achieved only using:

   $u_\phi \in \{-\phi_{\max}, 0, \phi_{\max})$

3. They give rise to three possible controls (primitives):

Letters	Steering u_φ
S (Straight)	0
L (Left)	$\phi_{\max}$
R (Right)	$-\phi_{\max}$

4. By only using **three** consecutive primiti ves we can achieve optimal paths with this cost: $L(\tilde{q}, \tilde{u}) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} dt,$

Describing the Dubins curves

The three possible controls (letters)

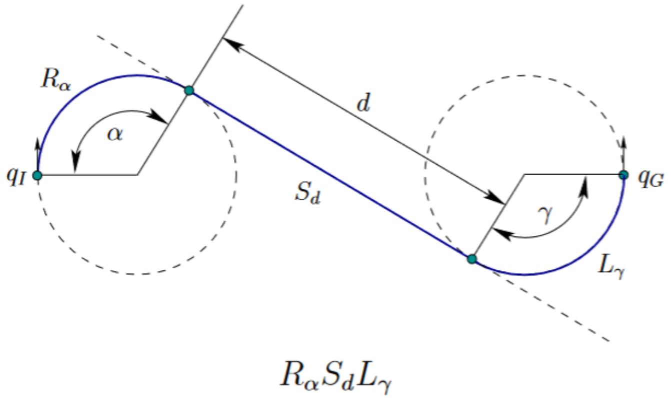
Letters	Steering u_φ
S (Straight)	0
L (Left)	$\phi_{\max}$
R (Right)	$-\phi_{\max}$

Dubins showed that we need only need 6 3-letter words to achieve the optimal path:

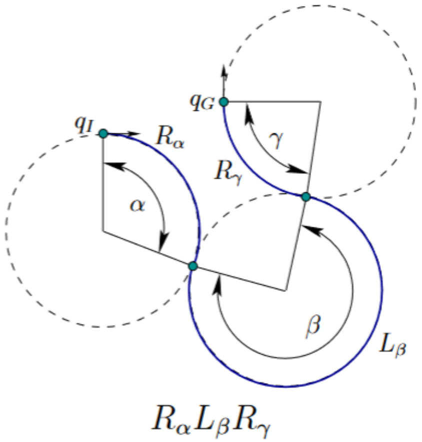
$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}.$$

Left and right means we move on a Circle, while S on a line, thus we have 2 types of motions:

$$\{CCC,\ CSC\}.$$

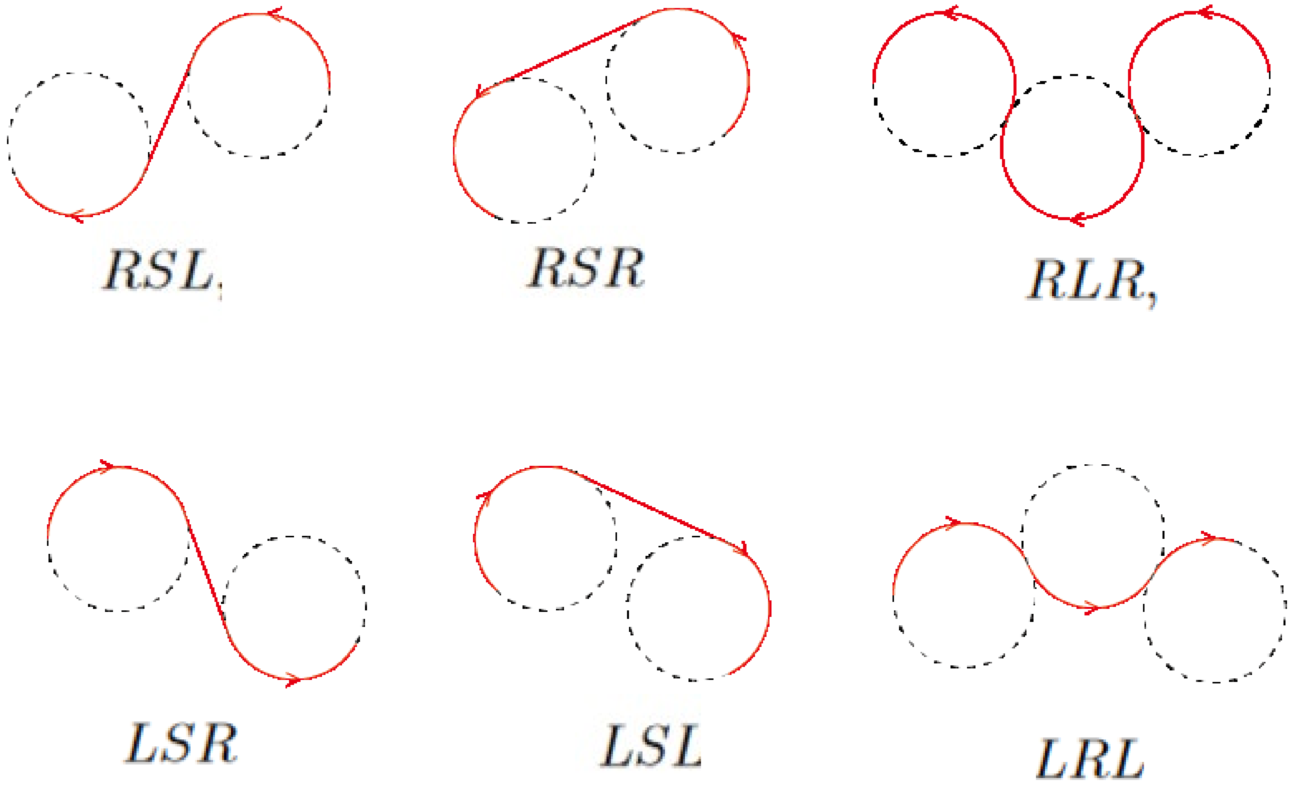


Circle Straight Circle



Circle Circle Circle

The 6 possible optimal Dubins curves



These are well defined because the first and last letter is always a known radius circle and the tangent line is unique each time

Kinodynamic Motion Planning with Roadmap Methods

0. **Initialization**
   - Add $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\text{goal}}$ to roadmap vertex set V
1. **Sampling**
   - Repeat several times

     $\boldsymbol{x} \leftarrow \text{StateSample()}$

     If $\text{IsStateValid}(\boldsymbol{x})$: add $\boldsymbol{x}$ to roadmap
2. **Connect Samples**

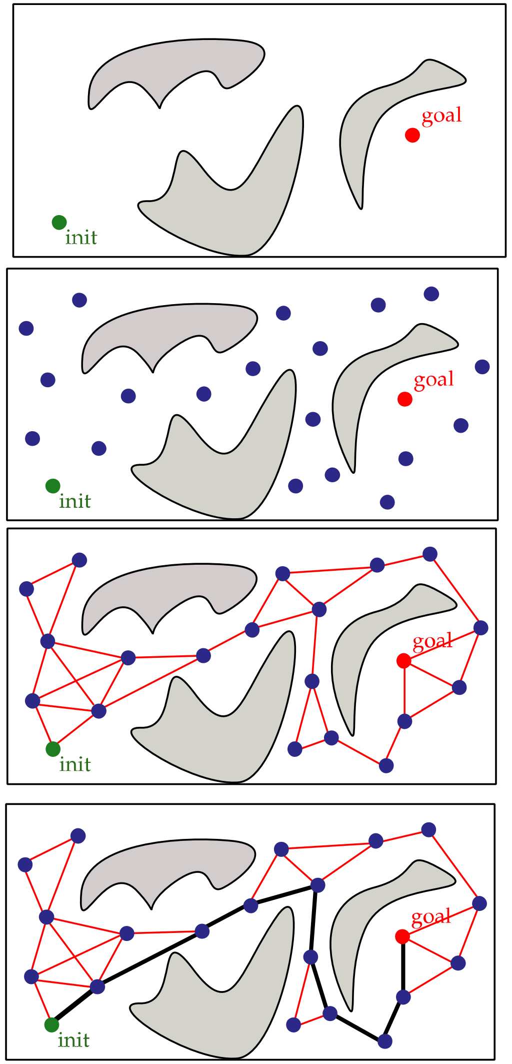
   For all neighbors $(\mathrm{x_a}, \mathrm{x_b})$

   $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$

   If $\text{IsTrajectoryValid}(\lambda) = \text{true}$: add edge to roadmap
3. **Graph Search**
   - Search graph (V, E) for path from x_0 to x_{goal}

We can solve

for some vehicles

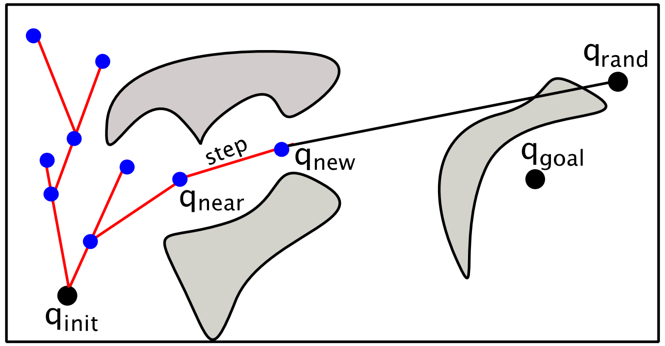


Can we completely avoid BVPs?

- Kinodynamic Formulation Recap
- Roadmap-based methods for kinodynamic systems
- Dubins Curves
- Tree-based methods for kinodynamic systems
- Planning without distances with KPIECE
- Planning Kinodynamic Near-Optimal Paths with SST*

Kinodynamic Motion Planning with Tree-Based Methods

Tree-based Approaches



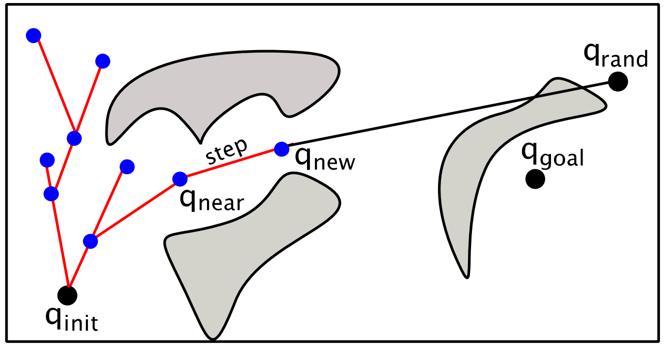
RRT

1: T ← create tree rooted at x_0
2: **While** solution not found **do**
   \\ select state from tree
3: $x_{rand} \leftarrow \text{StateSample}()$
4: x_{near} ← nearest state in T to x_{rand} according to distance ρ
   \\ add new branch to tree from selected state
5: $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_{near}, x_{rand})$
6: **if** $\text{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**
7: $x_{new} \leftarrow \lambda(step)$
8: add configuration x_{new} and edge (x_{near}, x_{new}) to T
   \\ check if a solution is found
9: **if** $\rho(x_{new}, x_{goal}) \approx 0$ **then**

10: **return** solution trajectory from root to x_{new}

Kinodynamic Motion Planning with Tree-Based Methods

Tree-based Approaches



RRT

1: $\mathrm{T} \leftarrow$ create tree rooted at x_0

2: **While** solution not found **do**

\\ select state from tree

3: $x_{rand} \leftarrow \mathrm{StateSample}()$

4: $x_{near} \leftarrow$ nearest state in T to x_{rand} according to distance ρ

\\ add new branch to tree from selected state

5: $\lambda \leftarrow \mathrm{GenerateLocalTrajectory}(x_{near}, x_{rand})$?

6: **if** $\mathrm{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**

7: $x_{new} \leftarrow \lambda(step)$

8: add configuration x_{new} and edge (x_{near}, x_{new}) to T

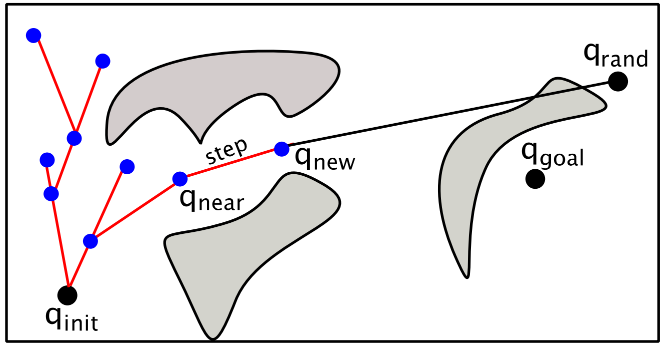
\\ check if a solution is found

9: **if** $\rho(x_{new}, x_{goal}) \approx 0$ **then**

10: **return** solution trajectory from root to x_{new}

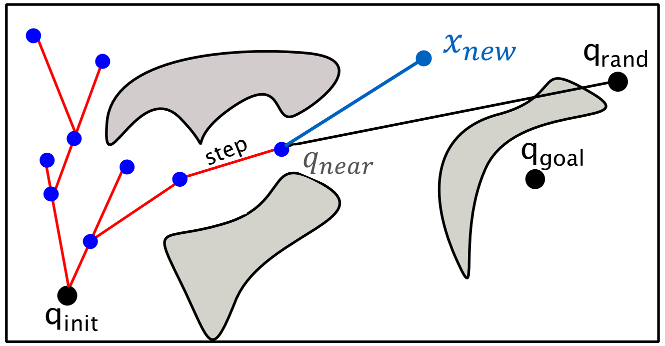
Kinodynamic Motion Planning with Tree-Based Methods

- $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_{near}, x_{rand})$
  - does it not create the same two-boundary value problems as in PRM?
  - is it necessary to connect to x_{rand}?
  - does it suffice to just come close to x_{rand}?



Kinodynamic Motion Planning with Tree-Based Methods

Idea for avoiding the 2-point Boundary Problem: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}

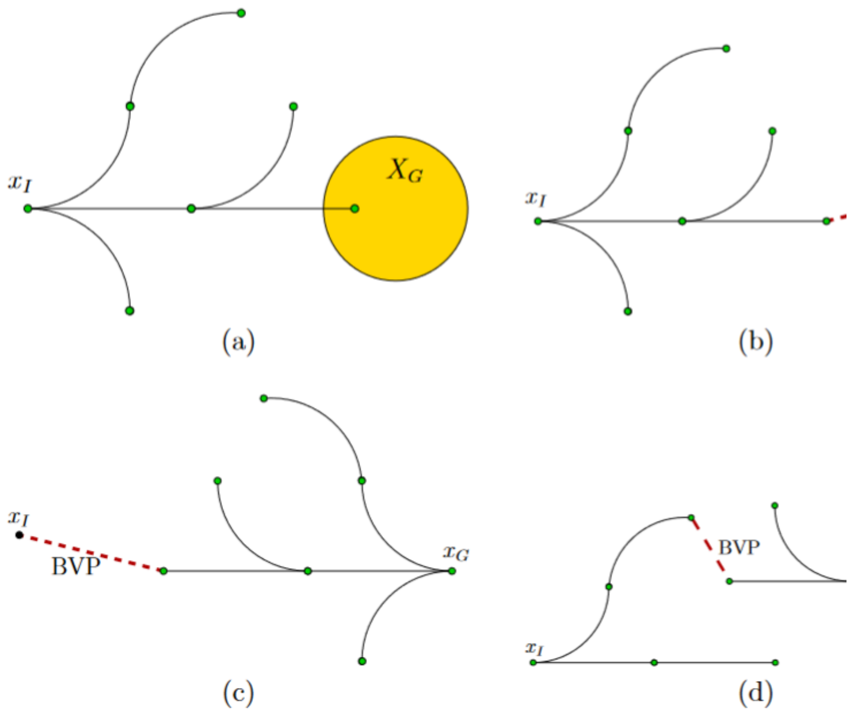


- Approach 1 – extend according to random control
  - Sample random control u in U
  - Integrate equations of motions when applying u to x_{near} for Δt units of time, i.e.,

$$\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$$

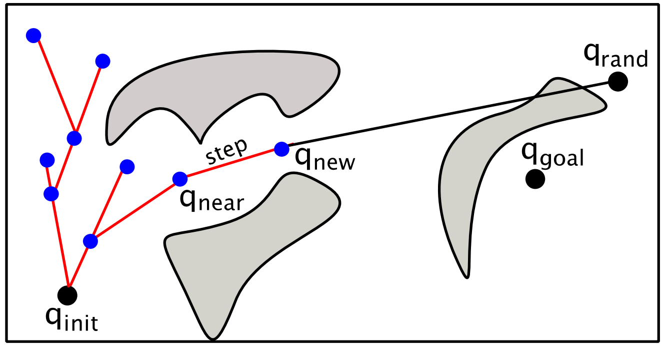
BVPS with Tree-Based Planners

- a) Tree-based with goal region (No BVP)
- b) Tree based with single goal (1 BVP)
- c) Backwards Tree (Many BVP)
- d) Bidirectional Trees (Many BVP)



Kinodynamic Motion Planning with Tree-Based Methods

Tree-based Approaches



RRT

$T \leftarrow$ create tree rooted at x_0

While solution not found **do**

\\\\ select state from tree

$x_{rand} \leftarrow \mathrm{StateSample}()$

$x_{near} \leftarrow$ nearest state in T to x_{rand} according to distance ρ

\\\\ add new branch to tree from selected state

$\lambda \leftarrow \mathrm{GenerateLocalTrajectory}(x_{near}, x_{rand})$

if $\mathrm{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**

$x_{new} \leftarrow \lambda(step)$

add configuration x_{new} and edge (x_{near}, x_{new}) to T

\\\\ check if a solution is found

if $\rho(x_{new}, x_{goal}) \approx 0$ **then**

return solution trajectory from root to x_{new}

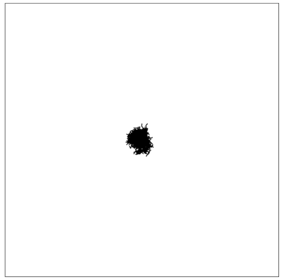
Is RRTs Bias preserved in kinodynamic Planning?

RRT

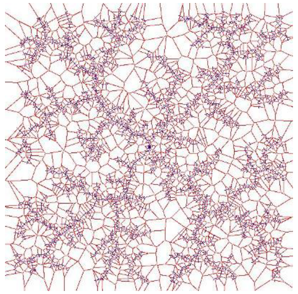
At each step, a random sample is taken and its nearest neighbor in the search tree computed. A new node is then created by extending the nearest neighbor toward the random sample.

Voronoi bias

At each iteration, the probability that a node is selected is proportional to the volume of its Voronoi region; hence, search is biased toward those nodes with the largest Voronoi regions (representing unexplored regions of the configuration space). This causes RRTs to rapidly explore.



Random Node Choice
(bad distance metric)



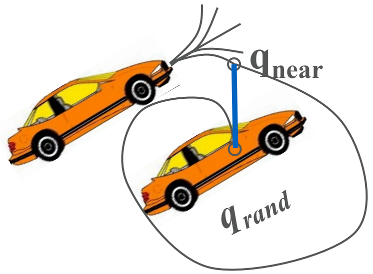
Voronoi Bias
(good distance metric)

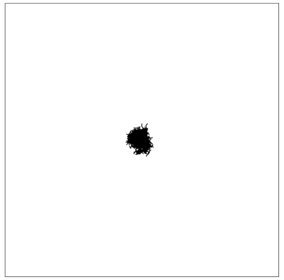
Geometric RRTs can rapidly cover unexplored regions

RRTs and Distance Metrics

- Hard to define *d*, the distance metric
  - Mixing velocity, position, rotation ,etc.

How do you pick a good q_{near}?





Configurations are close according to Euclidian metric, but actual distance is large

Avoiding computing Distances

- Kinodynamic Formulation Recap
- Roadmap-based methods for kinodynamic systems
- Dubins Curves
- Tree-based methods for kinodynamic systems
- Planning without distances with KPIECE
- Planning KinodynamicOptimal Paths with SST*

Main Points in KPIECE

- Motions
- Discretization Levels + Projections
- Importance of cells

Motions

Each motion $\mu = (s, u, t)$

state $s \in Q$,

control $u \in U$

a duration $t \in \mathbb{R}^{\geq 0}$.

A motion is similar to an edge in geometric planning, but can be split

It is possible to split a motion $\mu = (s, u, t)$ into $\mu_1 = (s, u, t_a)$ followed by $\mu_2 = (\int_{t_0}^{t_0+t_a} f(s(\tau), u) d\tau, u, t_b)$, where $s(\tau)$ identifies the state at time τ and $t_a + t_b = t$.

Discretization

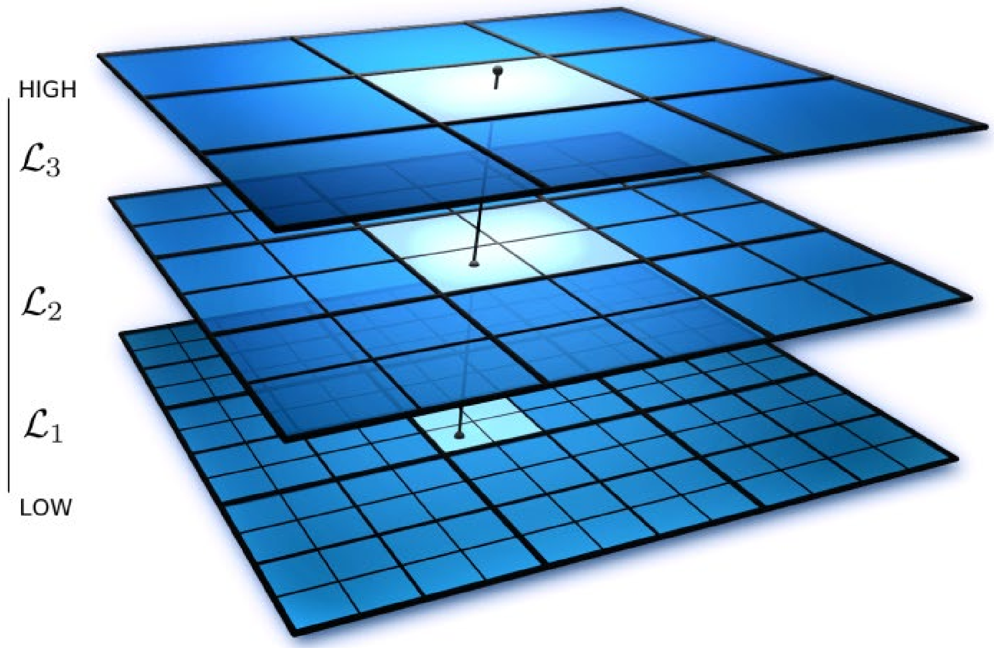


Fig. 1. An example discretization with three levels. The line intersecting the three levels defines a cell chain. Cell sizes at lower levels of discretization are integer multiples of the cell sizes at the level above.

This discretization consists of k levels L1 , ..., Lk ,

Each of these levels is a grid where cells are polytopes of fixed size.

the discretization is typically imposed on a projection of the state space, E(Q)

Motions on Discretizations

If a motion spans more than one cell at the same level of discretization, it is split into smaller motions such that no motions cross cell boundaries.

Motion

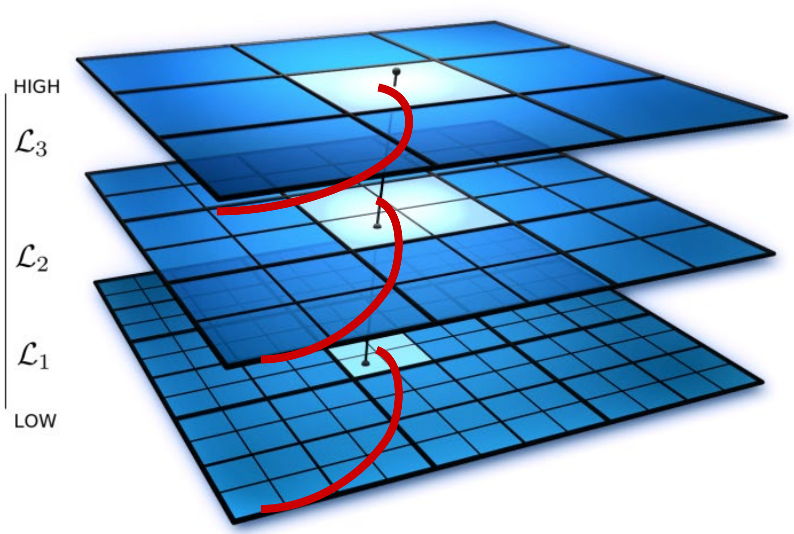
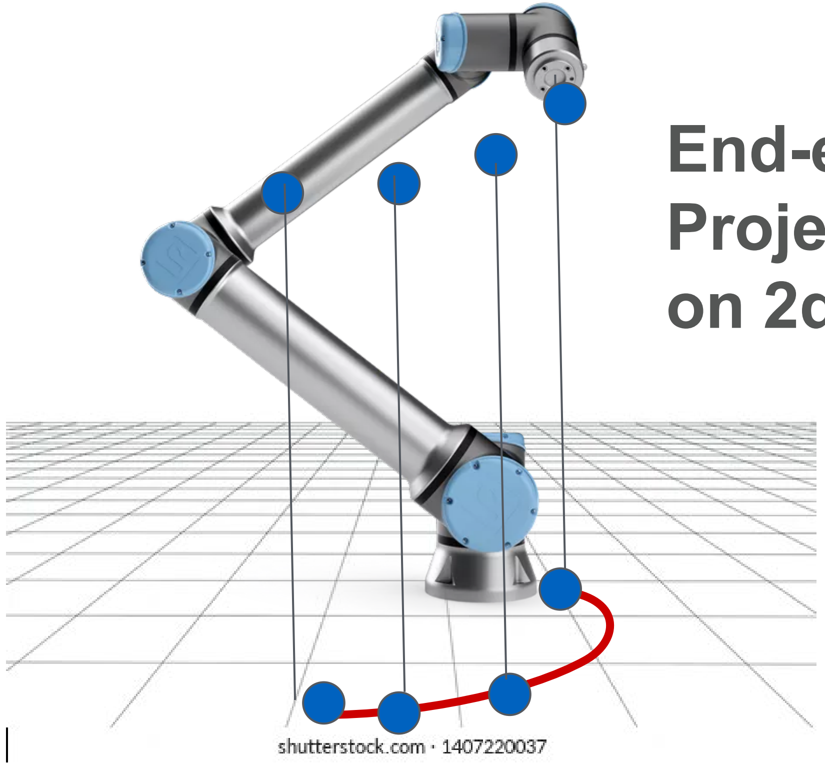


Fig. 1. An example discretization with three levels. The line intersecting the three levels defines a cell chain. Cell sizes at lower levels of discretization are integer multiples of the cell sizes at the level above.



End-effector Projection on 2d

E: [j1,j2,j3,j4,j5,j6] -> [x,y]

Importance Selection

The importance of a cell p, regardless of the level of discretization it is part of, is computed as:

$$Importance(p) = \frac{\log(\mathcal{I}) \cdot \mathtt{score}}{\mathcal{S} \cdot \mathcal{N} \cdot \mathcal{C}}$$

- **I** stands for the number of the iteration at which p was created,
- **score** is initialized to 1 but may later be updated to reflect the exploration progress
- **S** is the number of times p was selected for expansion (initialized to 1),
- **N** is the number of instantiated neighboring cells
- **C** is a positive measure of coverage for p,

Importance Selection

- Once a cell p is selected, if p ∈ not L1 ,
- The selection process continues recursively:
- an instantiated cell from Dp is subsequently selected until the last level of discretization is reached
- At the last level, a motion μ from Mp is picked
- A state s along μ is then chosen uniformly at random [line 7].
- Expanding the tree of motions continues from s [line 9].

Adding A motion to the exciting tree

Algorithm 2 ADDMOTION(s, u, t)

```
Split (s, u, t) into motions μ_1, ..., μ_k such that μ_i, i ∈ {1, ..., k} does not cross the boundary of any cell at the lowest level of discretization
for μ_∘ ∈ {μ_1, ..., μ_k} do
    Find the cell chain corresponding to μ_∘
    Instantiate cells in the chain, if needed
    Add μ_∘ to the cell at the lowest level in the chain
    Update coverage measures and lists of interior and exterior cells, if needed
end for
```

KPIECE Algorithm

Algorithm 1 KPIECE(q_{start}, $N_{iterations}$)

Let μ_0 be the motion of duration 0 containing solely q_{start}
Create an empty `Grid` data-structure G
G.AddMotion(μ_0)
for $i \leftarrow 1 ... N_{iterations}$ **do**
    Select a cell chain $\mathbf{c}$ from G, with a bias on exterior cells (70% - 80%)
    Select μ from $\mathbf{c}$ according to a half normal distribution
    Select s along μ
    Sample random control $u \in U$ and simulation time $t \in \mathbb{R}^+$
    Check if any motion $(s, u, t_\circ)$, $t_\circ \in (0, t]$ is valid (forward propagation)
    if a motion is found **then**
        Construct the valid motion $\mu_\circ = (s, u, t_\circ)$ with $t_\circ$ maximal
        If $\mu_\circ$ reaches the goal region, **return** path to $\mu_\circ$
        G.AddMotion($\mu_\circ$)
    end if
    for every level $\mathcal{L}_j$ **do**
        $P_j = \alpha + \beta \cdot$ (ratio of increase in coverage of $\mathcal{L}_j$ to simulated time)
        Multiply the `score` of cell p_j in $\mathbf{c}$ by P_j if and only if $P_j < 1$
    end for
end for

Takeway

By estimating the coverage of the cells (Derived from projection) we can choose which cell to expand, and then which node to expand.

Essentially we have replaced the q_near from RRT with this process.

Regarding the direction of the expansion it is just random

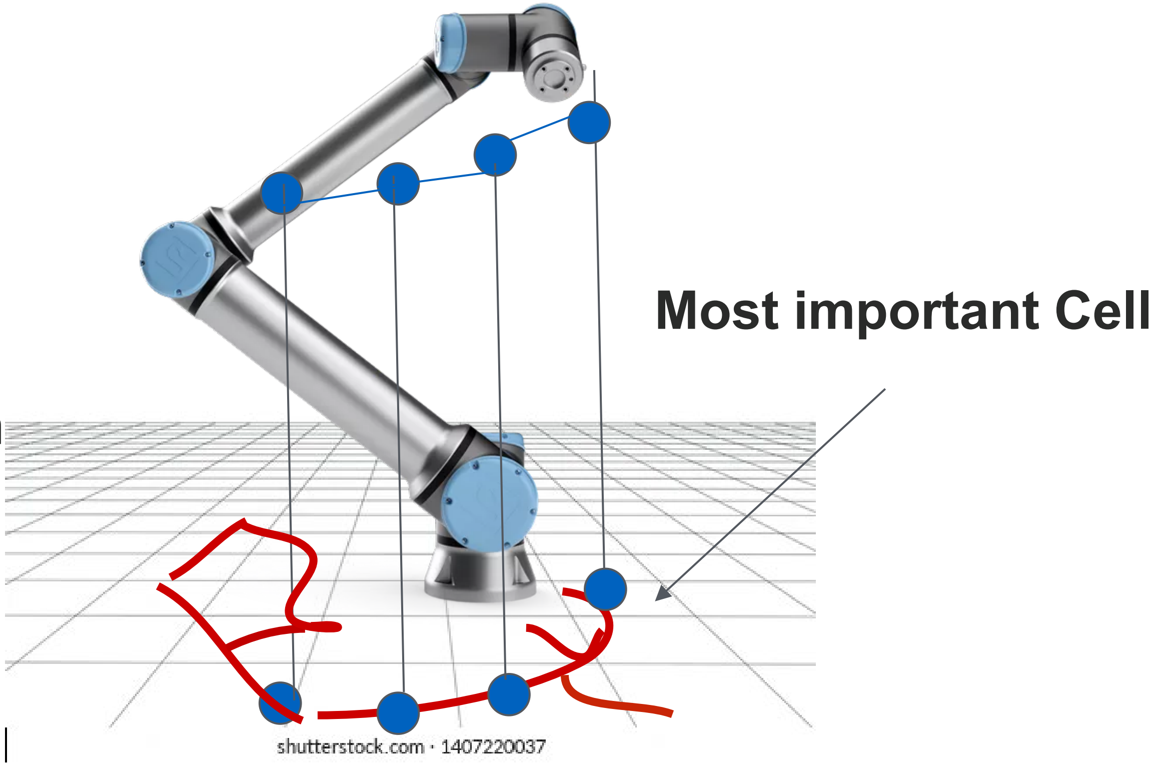
Intuitive Example

End-effector Projection on 2d

E: [j1,j2,j3,j4,j5,j6] -> [x,y]

$$Importance(p) = \frac{\log(\mathcal{I}) \cdot \texttt{score}}{\mathcal{S} \cdot \mathcal{N} \cdot \mathcal{C}}$$

I number of the iteration that p was created,
score the exploration progress
S is the number of times p was selected for expansion (initialized to 1),
N is the number of instantiated neighboring cells
C is a positive measure of coverage for p



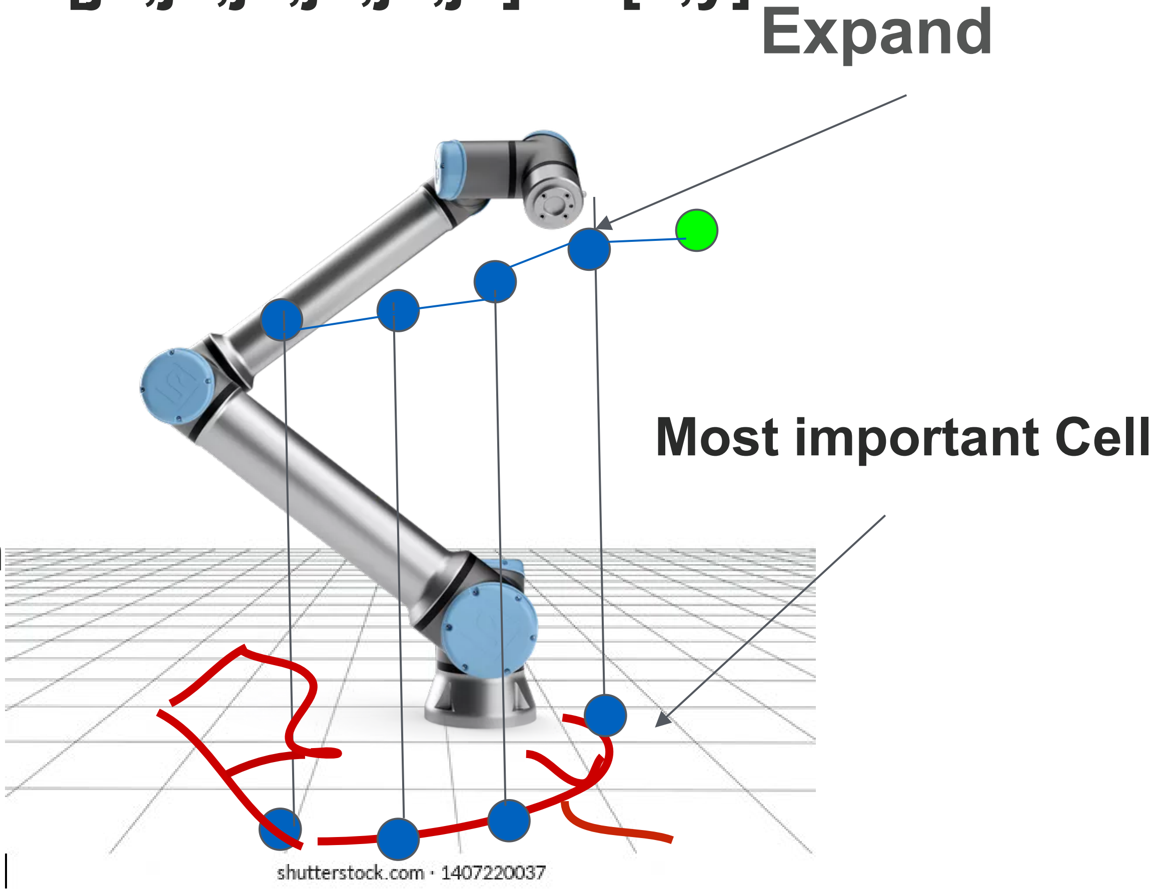
Intuitive Example

End-effector Projection on 2d

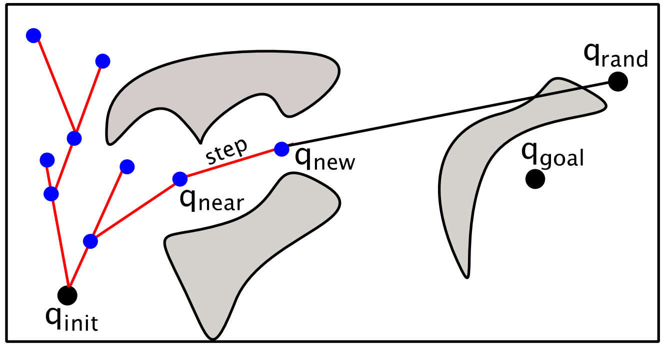
$$Importance(p) = \frac{\log(\mathcal{I}) \cdot \texttt{score}}{\mathcal{S} \cdot \mathcal{N} \cdot \mathcal{C}}$$

I number of the iteration that p was created,
score the exploration progress
S is the number of times p was selected for expansion (initialized to 1),
N is the number of instantiated neighboring cells
C is a positive measure of coverage for p

E: [j1,j2,j3,j4,j5,j6] -> [x,y]



Kinodynamic Motion Planning with Tree-Based Methods



Tree-based Approaches

RRT

1: $\mathrm{T} \leftarrow$ create tree rooted at x_0

2: **While** solution not found **do**

\\ select state from tree

3: $x_{rand} \leftarrow \mathrm{StateSample}()$

4: $x_{near} \leftarrow$ nearest state in T to x_{rand} according to distance ρ

\\ add new branch to tree from selected state

5: $\lambda \leftarrow \mathrm{GenerateLocalTrajectory}(x_{near}, x_{rand})$

6: **if** $\mathrm{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**

7: $x_{new} \leftarrow \lambda(step)$

8: add configuration x_{new} and edge (x_{near}, x_{new}) to T

\\ check if a solution is found

9: **if** $\rho(x_{new}, x_{goal}) \approx 0$ **then**

10: **return** solution trajectory from root to x_{new}

What about optimal motions?

- Kinodynamic Formulation Recap
- Roadmap-based methods for kinodynamic systems
- Dubins Curves
- Tree-based methods for kinodynamic systems
- Planning without distances with KPIECE
- Planning Kinodynamic Near-Optimal Paths with SST*

Kinodynamic Motion Planning with Stable Sparse RRT

- Provides asymptotic (near-)optimality for kinodynamic planning without access to a steering function
- Maintains only a sparse set of samples, in contrast to other tree-based method
- Converges fast to high-quality paths

Kinodynamic Motion Planning with Stable Sparse RRT

- Selecting best path cost nodes in neighborhoods for nearest neighbor queries and expansion

- Define three new sets: $\mathbb{V}_{\boldsymbol{acive}}$, $\mathbb{V}_{\boldsymbol{inactive}}$, $\boldsymbol{S}$ (witness)

- Only one active state within radius $\boldsymbol{\delta}$ neighborhoods around any $\boldsymbol{s} \in \boldsymbol{S}$

- Any state in $\mathbb{V}_{\boldsymbol{acive}}$ called the representative of the witness s

- Representatives change over time only when their root cost decreases

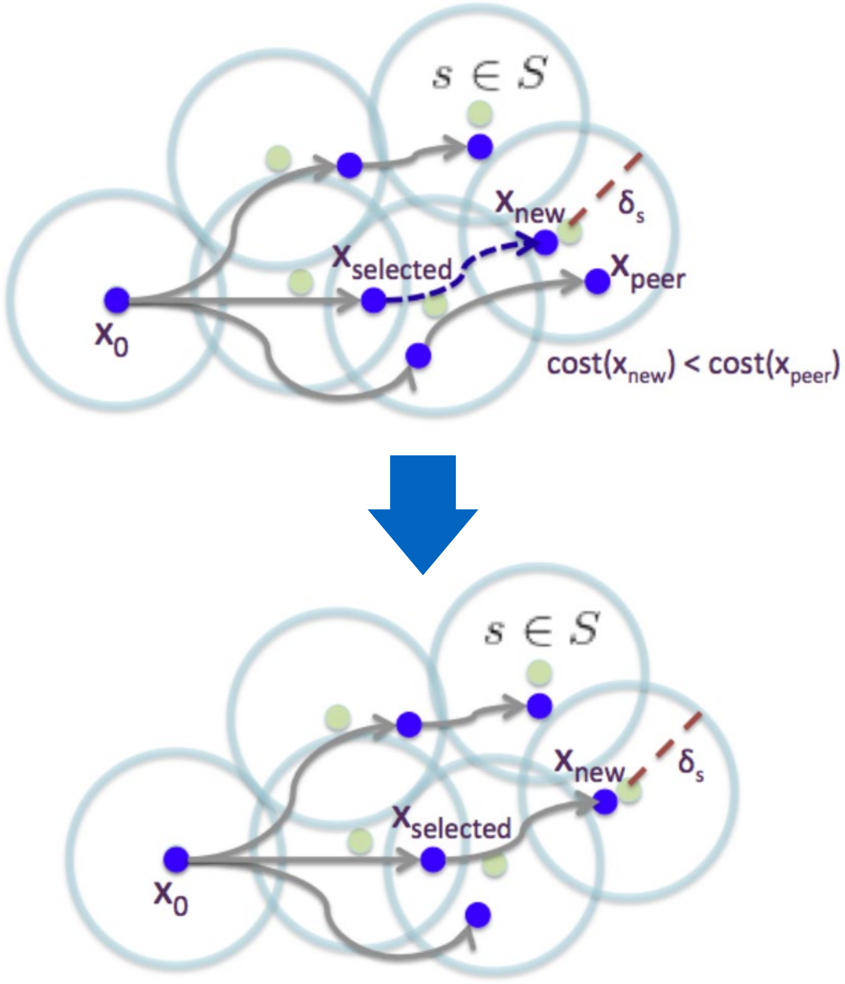
- Reducing $\boldsymbol{\delta}$ parameters over time to achieve asymptotic optimality

Kinodynamic Motion Planning with Stable Sparse RRT

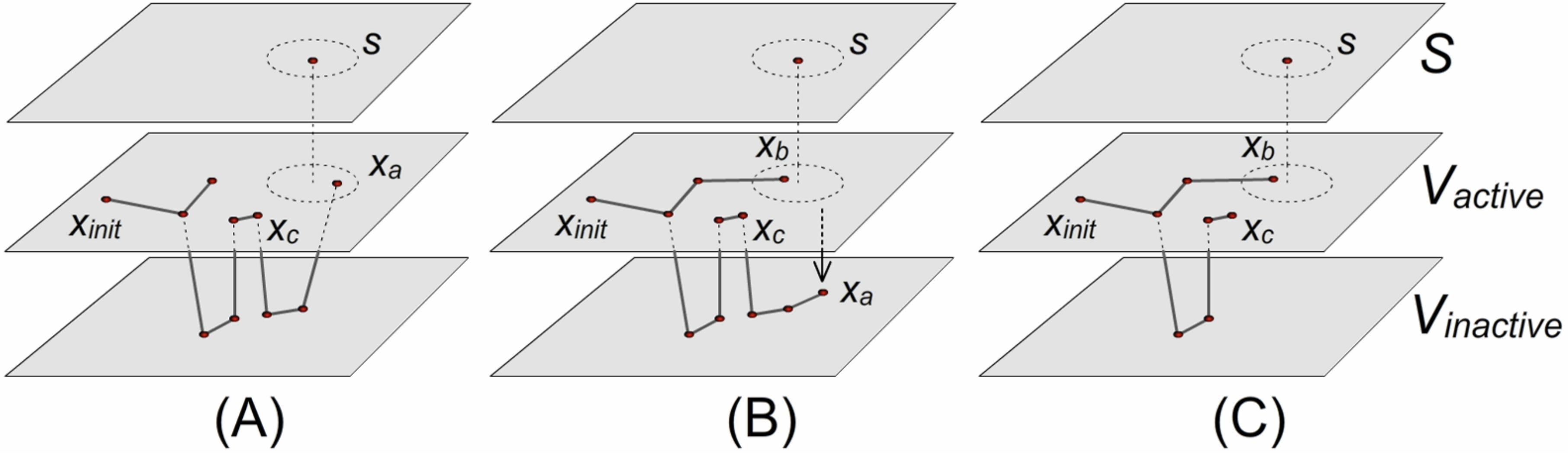
Algorithm 1: SST($\mathbb{X}$, $\mathbb{U}$, x_0, T_{prop}, N, δ_v, δ_s)

$i \leftarrow 0$; // Iteration counter
$\mathbb{V}_{active} \leftarrow \{x_0\}, \mathbb{V}_{inactive} \leftarrow \emptyset,\ \mathbb{V} \leftarrow \mathbb{V}_{active} \cup \mathbb{V}_{inactive}$; // Node sets
$\mathbb{E} \leftarrow \emptyset,\ G = \{V, \mathbb{E}\}$; // Initialize graph
$s_0 \leftarrow x_0,\ \ s_0.rep = x_0,\ \ S \leftarrow \{s_0\}$; // Initialize witness set
while $i{+}{+} < N$ **do**
    $s_{sample} \leftarrow$ Sample($\mathbb{X}$) ; // Uniform sampling in state space
    $x_{nearest} \leftarrow$ BestNear($\mathbb{V}_{active}$, s_{sample}, δ_v) ; // Return the BestNear node
    $x_{new} \leftarrow$ MonteCarlo-Prop($x_{nearest}$, $\mathbb{U}$, T_{prop}) ; // Propagate forward
    if CollisionFree($\overline{x_{nearest} \rightarrow x_{new}}$) **then**
        $s_{new} \leftarrow$ Nearest(S, x_{new}) ; // Get the nearest witness to x_{new}
        if $dist(x_{new}, s_{new}) > \delta_s$ **then**
            $S \leftarrow S \cup \{x_{new}\}$; // Add a new witness that is x_{new}
            $s_{new} \leftarrow x_{new}$;
            $s_{new}.rep \leftarrow NULL$;
        $x_{peer} \leftarrow s_{new}.rep$; // Get current represented node
        if $x_{peer} == NULL$ *or* cost(x_{new}) $<$ cost(x_{peer}) **then**
            $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \setminus \ \{x_{peer}\}$; // Removing old rep
            $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \cup \ \{x_{peer}\}$; // Making old rep inactive
            $s_{new}.rep \leftarrow x_{new}$; // Assign the new rep
            $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \cup \{x_{new}\},\ \mathbb{E} \leftarrow \mathbb{E} \cup \{\overline{x_{nearest} \rightarrow x_{new}}\}$; // Grow G
            while IsLeaf (x_{peer}) **and** $x_{peer} \in \mathbb{V}_{inactive}$ **do**
                $x_{parent} \leftarrow$ Parent(x_{peer});
                $\mathbb{E} \leftarrow \mathbb{E} \setminus \{\overline{x_{parent} \rightarrow x_{peer}}\}$; // Remove from G
                $\mathbb{V}_{inactive} \leftarrow \mathbb{V}_{inactive} \setminus \ \{x_{peer}\}$; // Remove from inactive set
                $x_{peer} \leftarrow x_{parent}$; // Recurse to parent if inactive

return G;

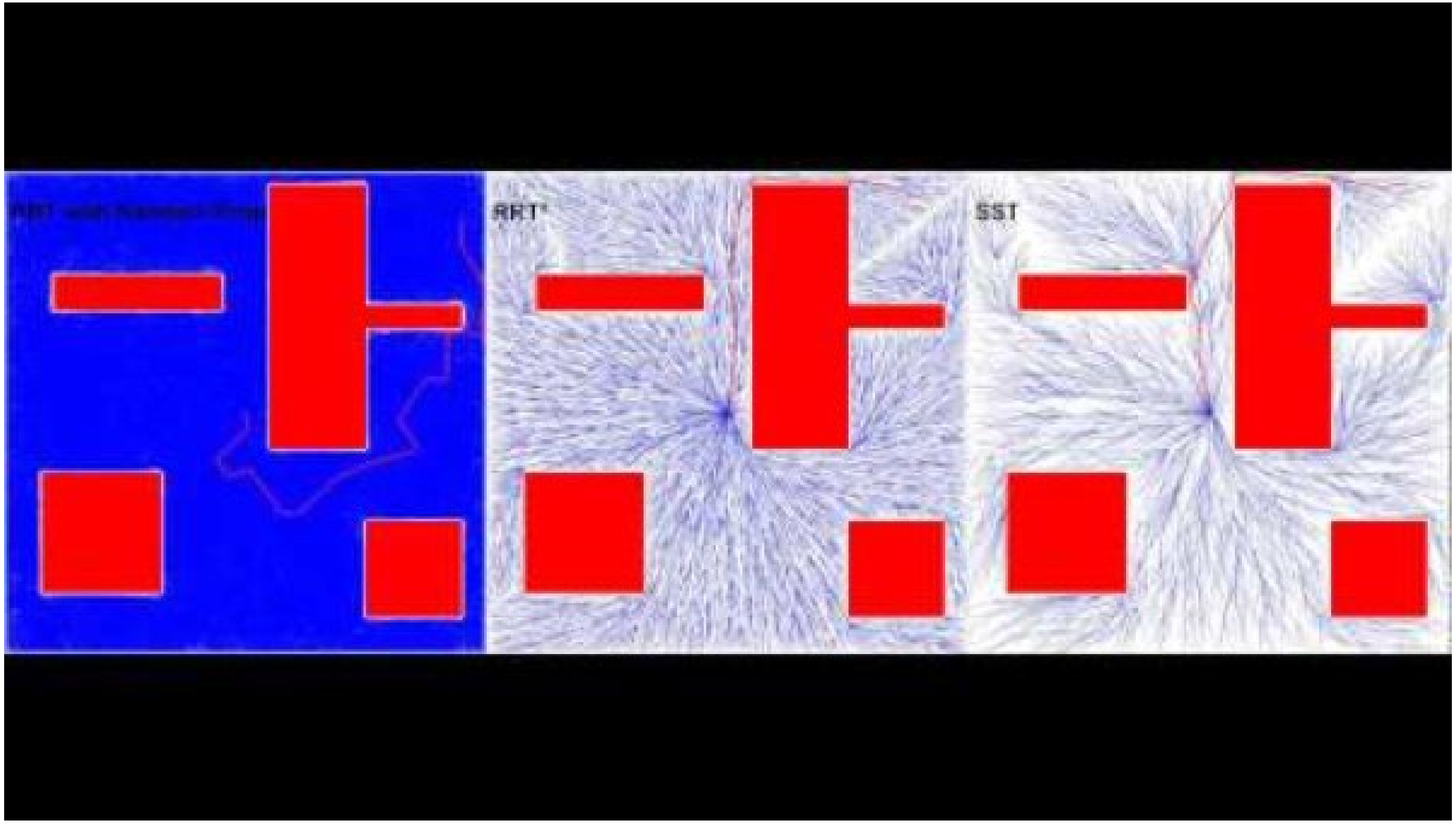


Kinodynamic Motion Planning with Stable Sparse RRT

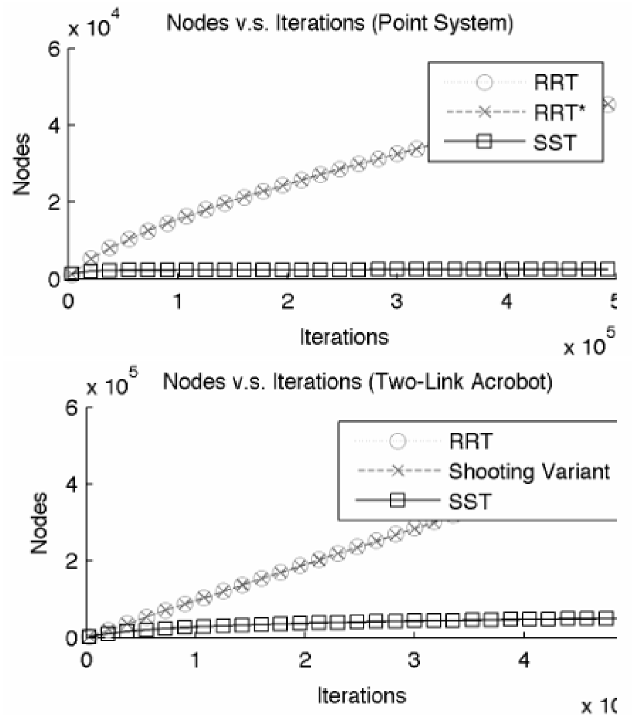


Relation between S and the V sets. A: A tree and a trajectory $\overline{x_{init} \rightarrow x_c \rightarrow x_a}$ where x_a is the representative of s; B: The algorithm extends $\overline{x_{init} \rightarrow x_b}$ where x_b has better cost than x_a. x_a is moved from V_{active} to $V_{inactive}$. C: The representative of s is now x_b (Lines 21-25 of Alg. 1). The trajectory $\overline{x_c \rightarrow x_a}$ in $\mathbb{V}_{inactive}$ is pruned.

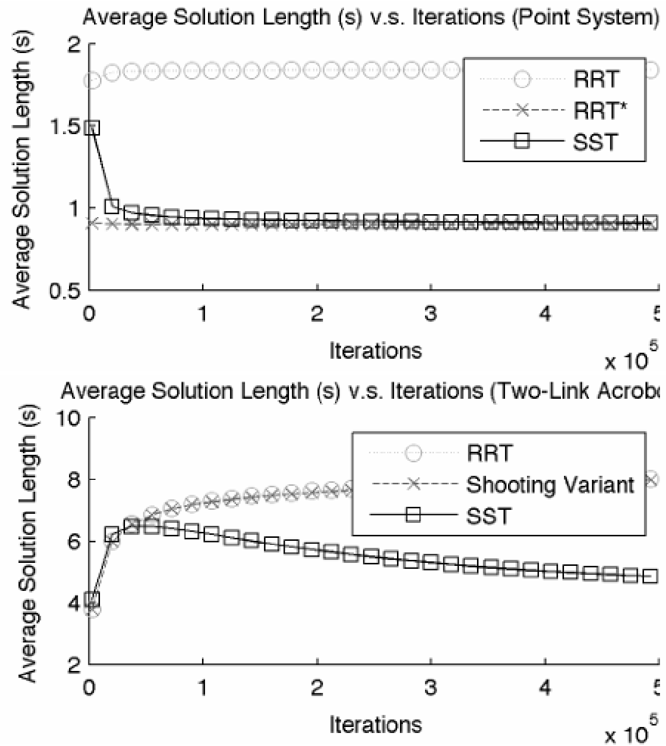
Kinodynamic Motion Planning with Stable Sparse Tree



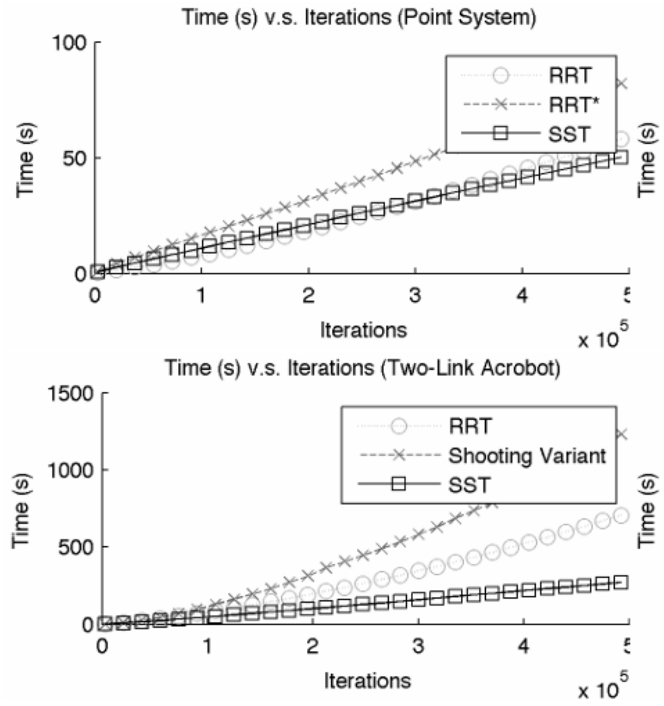
Kinodynamic Motion Planning with Stable Sparse RRT



Number of nodes



The average cost to each node in in the tree



The amount of time needed