RBE550
Motion Planning
Tree-based Planners

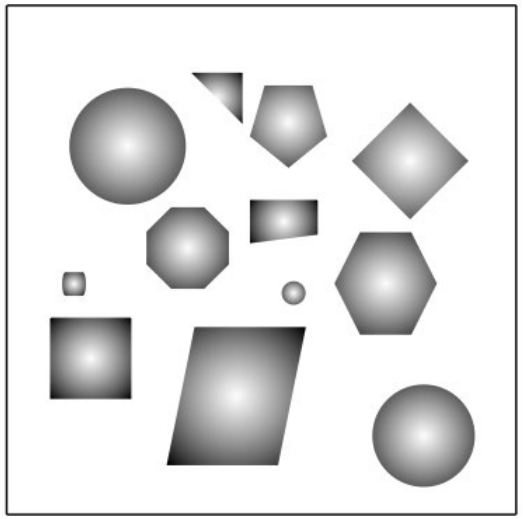
Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Disclaimer and Acknowledgments

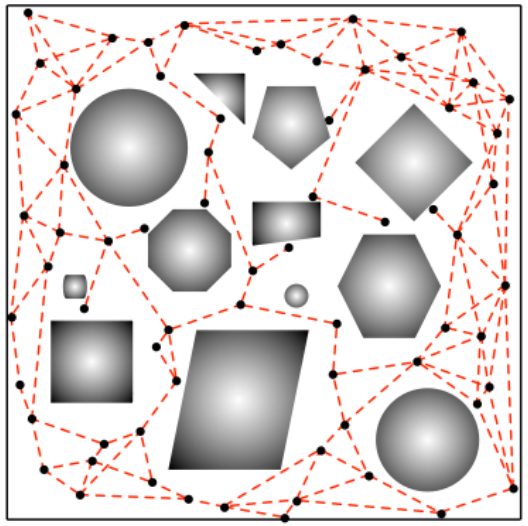
The slides are a compilation of work based on notes and slides mainly from Morteza Lahijanian, but also Lydia Kavraki, Howie Choset, Erion Plaku, Constantinos Chamzas, David Hsu, Greg Hager, Mark Moll, G. Ayorkor Mills-Tetty, Hyungpil Moon, Zack Dodds, Zak Kingston, Nancy Amato, Steven Lavalle, Seth 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: Multi-Query-Planners, Probabilistic RoadMap

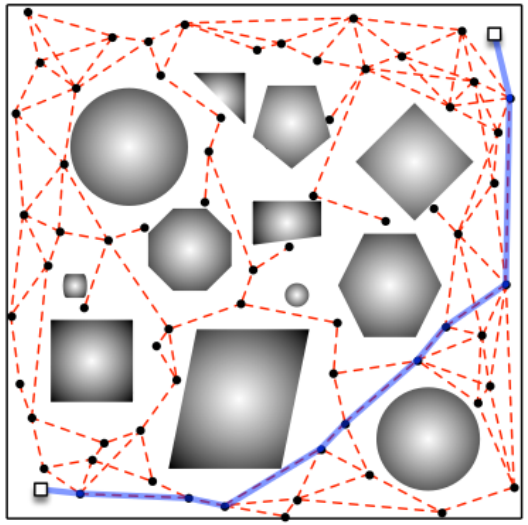
- Probabilistic Roadmap (PRM) is sampling-based technique for multi-query planning problems



PRM consists of 2 phases



1. Learning phase
   - Given robot and workspace
   - Generate a graph



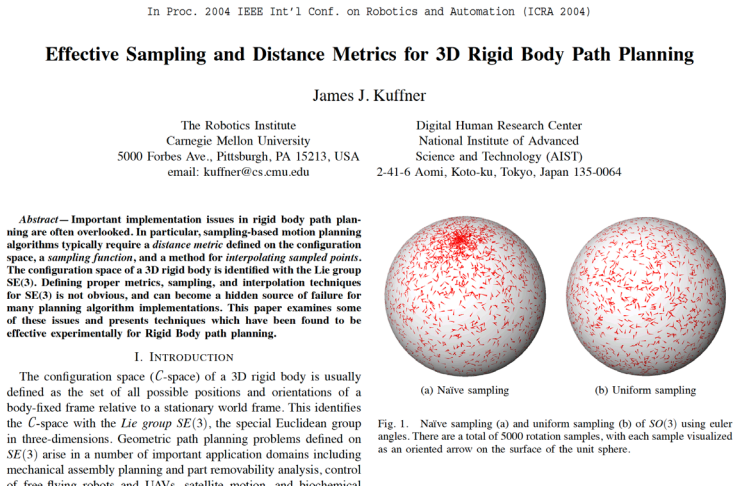
2. Query phase
   - Given start and goal configuration
   - Graph search (A*)

Multi-query Planner: For the same robot, workspace, only they Query Phase needs to be executed

Last Time: Sampling-Strategies and Efficient Data Structures

- Sampling Strategies
  - Gaussian Sampling
  - Bridge Sampling
  - Obstacle-Based Sampling
  - Experience-Based Sampling

- Efficient Data Structures
  - KD-tree,
  - R-tree,
  - M-tree,
  - GNAT,
  - iDistance
  - CoverTree



In Proc. 2004 IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)

Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning

James J. Kuffner

The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, USA
email: kuffner@cs.cmu.edu

Digital Human Research Center
National Institute of Advanced
Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo, Japan 135-0064

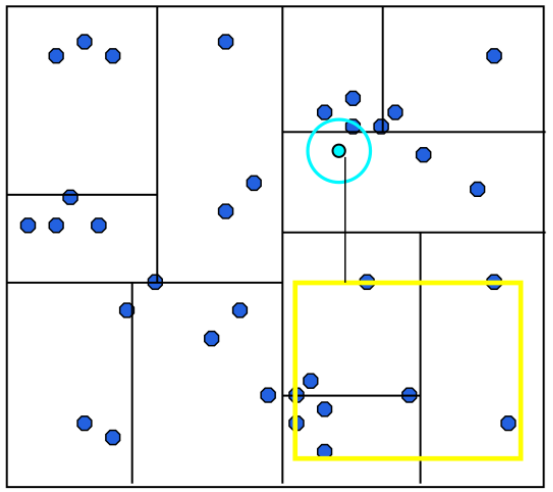
***Abstract*— Important implementation issues in rigid body path planning are often overlooked. In particular, sampling-based motion planning algorithms typically require a *distance metric* defined on the configuration space, a *sampling function*, and a method for *interpolating sampled points*. The configuration space of a 3D rigid body is identified with the Lie group SE(3). Defining proper metrics, sampling, and interpolation techniques for SE(3) is not obvious, and can become a hidden source of failure for many planning algorithm implementations. This paper examines some of these issues and presents techniques which have been found to be effective experimentally for Rigid Body path planning.**

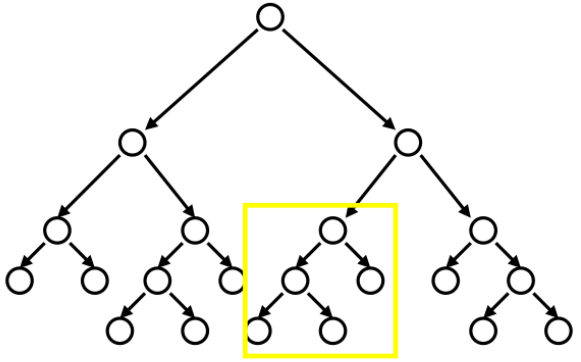
I. Introduction

The configuration space ($\mathcal{C}$-space) of a 3D rigid body is usually defined as the set of all possible positions and orientations of a body-fixed frame relative to a stationary world frame. This identifies the $\mathcal{C}$-space with the *Lie group* $SE(3)$, the special Euclidean group in three-dimensions. Geometric path planning problems defined on $SE(3)$ arise in a number of important application domains including mechanical assembly planning and part removability analysis, control of free-flying robots and UAVs, satellite motion, and biochemical

(a) Naïve sampling (b) Uniform sampling

Fig. 1. Naïve sampling (a) and uniform sampling (b) of $SO(3)$ using euler angles. There are a total of 5000 rotation samples, with each sample visualized as an oriented arrow on the surface of the unit sphere.





Overview Today

- Metrics in C-Space,

- Single Query Planners, e.g. Tree-Based Plannersb
  - Basic-RRT
- Improvements for Tree-based Planenrs
  - Goal Bias
  - Bi-Directional
  - Connect Heuristic
- Tree-Based Planners
  - RRT-Connect
  - EST,KPIECE,SRT
- Path-smoothing

Metric/Distance in Configuration Space

- A *metric* or *distance* function d in C is a map

$$d\colon\ C^2 \to \mathbb{R}^{\geq 0}$$

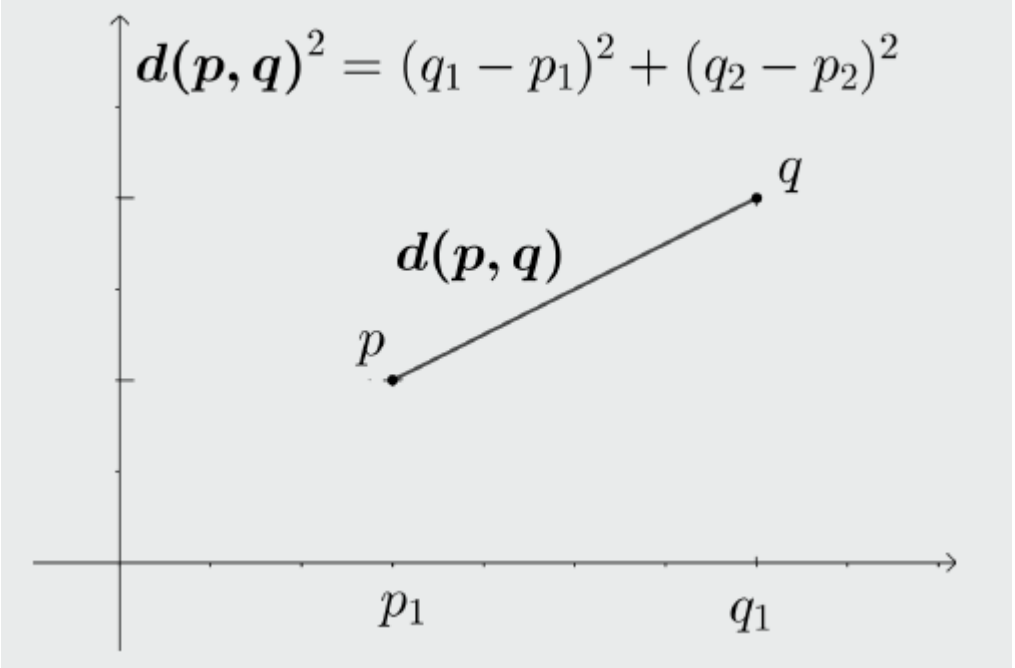
such that:

- $d(q_1, q_2) = 0$ if and only if $q_1 = q_2$ (identity of indiscernible)
- $d(q_1, q_2) = d(q_2, q_1)$ (symmetry)
- $d(q_1, q_2) \leq d(q_1, q_3) + d(q_3, q_2)$ (triangle inequality)

Euclidean Metric

Length of line segment between two points

$$d(x, x') = \sqrt{\sum_i (x_i - x'_i)^2} = \|x - x'\|_2.$$



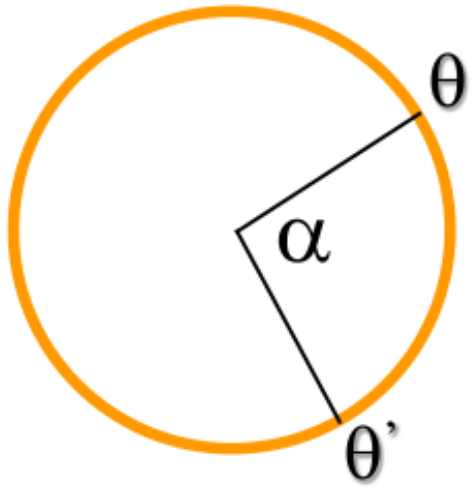
- The most commonly metric used in R^n spaces

Circular Metric SO(2)

Euclidean distance breaks down on spaces like the circle (is it a metric? other problems?)

- Between two distinct points, there are two paths: clockwise or counterclockwise

- Metric: $d(\theta, \theta') = \alpha = \min\{|\theta - \theta'|, \ 2\pi - |\theta - \theta'|\}$



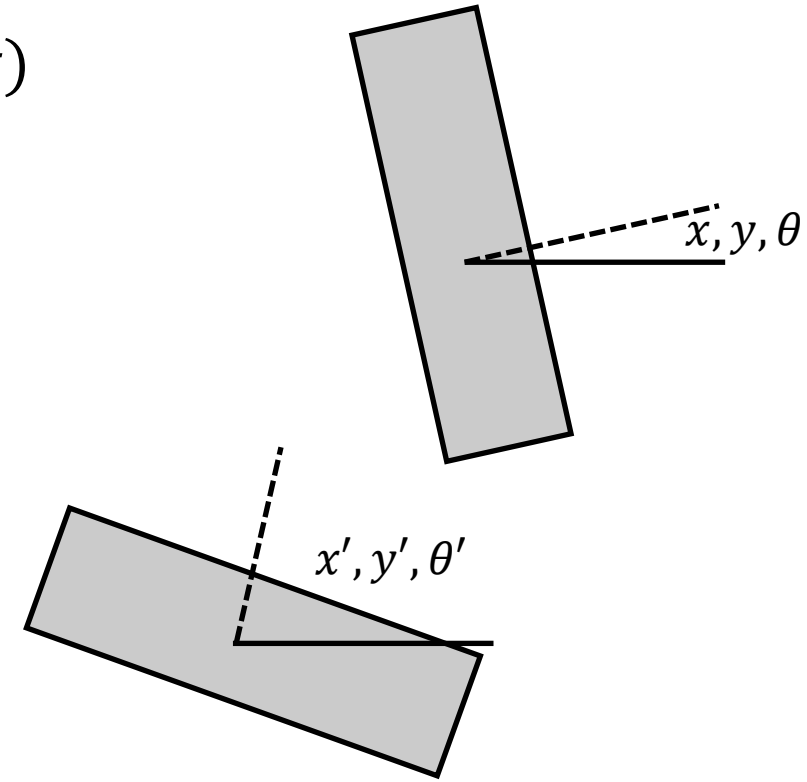
SE(2) Metric

Example in $\mathbb{R}^2 \times S^1$, combine Euclidean with Circular metric:

- $q = (x, y, \theta)$, $q' = (x', y', \theta')$ with $\theta, \theta' \in [0, 2\pi)$

$$\alpha = \min\{|\theta - \theta'|,\ 2\pi - |\theta - \theta'|\}$$

$$d_1(q, q') = \sqrt{(x - x')^2 + (y - y')^2 + \alpha^2}$$



SO(3) Metric, based on quaternions

q1, q2 are quaternions and θ represents the angle between them

The angle is just derived from the inner product

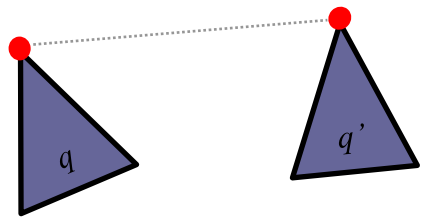
$$\theta = \cos^{-1}\left(2\langle q_1, q_2\rangle^2 - 1\right)$$

Workspace Metric

- Example in $\mathbb{R}^2$:
  - Robot A and point x of A
  - $x(q)$: location of x in the workspace when A is at configuration q
  - A distance d in C is defined by:

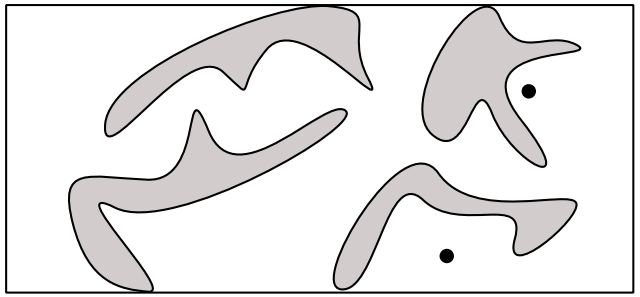
$$d(q, q') = \max_{x \in A} ||x(q) - x(q')||$$

where $||a - b||$ denotes the Euclidean distance between points a and b in the workspace



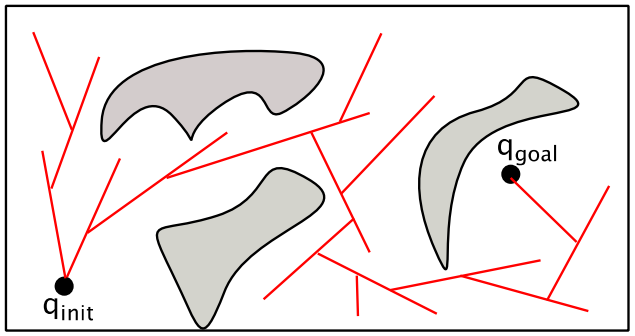
Single Query Sampling-based Planner

- **Idea:** for a single query, grow a tree in the free configuration space from q_{init} toward q_{goal}



TreeSearchFramework(q_{init} , q_{goal})

1: $T \leftarrow$ RootTree(q_{init})
2: **while** q_{goal} has not been reached **do**
3: $q \leftarrow$ SelectConfigFromTree(T)
4: AddTreeBranchFromConfig(T, q)



- **Critical Issues**
  - How should a configuration be selected from the tree? [SELECTION]
  - How should a new branch be added to the tree from the selected configuration [EXPANSION]

Random Tree (RTP)

- Steps
  - Pick a node at random
  - Sample a new node near it
  - Grow tree from the random node to the new node

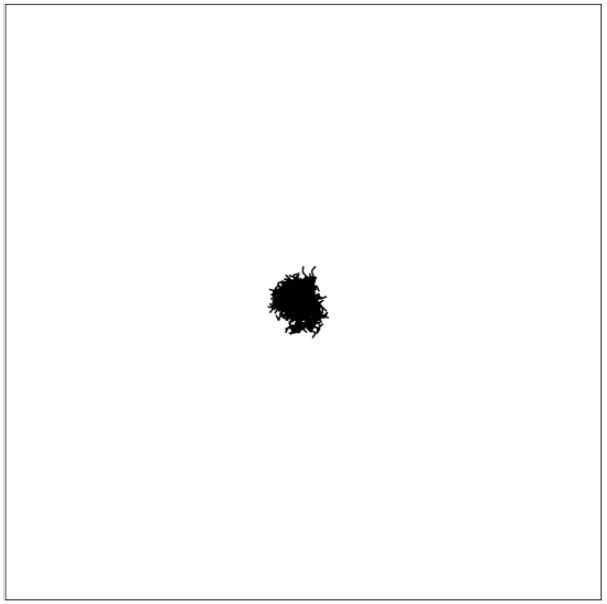
q_{node} = q_{start}

For i = 1 to NumberSamples

  q_{rand} = Sample **near** q_{node}

  Add edge e = (q_{rand} , q) if collision-free

  q_{node} = **Pick random node of tree**

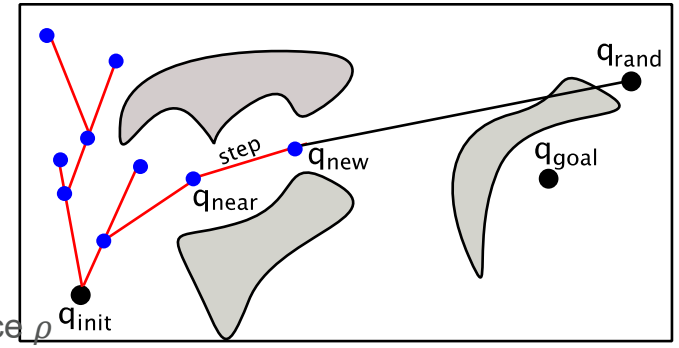


Does not explore much!

Rapidly-exploring Random Tree (RRT)

- **Idea:** Pull the tree toward random samples in the configuration space

RRT (q_{init}, q_{goal})
//initialize tree
1: T ← create tree rooted at q_{init}
2: **while** solution not found **do**
    // select configuration from tree
3: q_{rand} ← generate a random sample
4: q_{near} ← nearest configuration in T to q_{rand} according to distance ρ
    // add new branch to tree from selected configuration
5: $path$ ← generate path (not necessarily collision free) from q_{near} to q_{rand}
6: **if** $\text{IsSubpathCollisionFree}(path, 0, step)$ **then**
7: $q_{new} \leftarrow path(step)$
8: add configuration q_{new} and edge (q_{near}, q_{new}) to T
    //check if a solution is found
9: **if** $\rho(q_{new}, q_{goal}) \approx 0$ **then**
10: **return** solution path from root to q_{new}



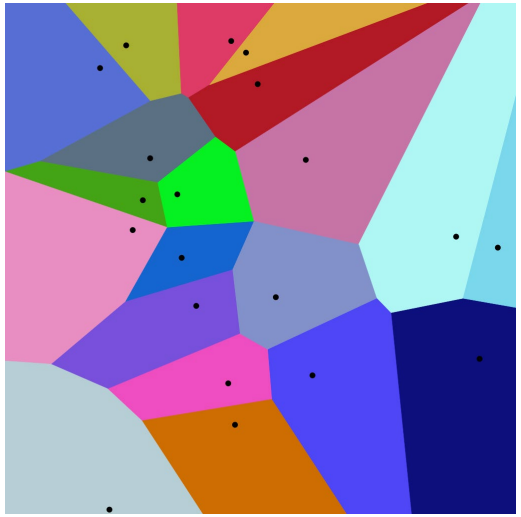
- RRT relies on nearest neighbors and distance metric $\rho: Q \times Q \leftarrow \mathbb{R}^{\geq 0}$
- RRT adds Voronoi bias to tree growth space

Voronoi Diagram – RRT Analysis

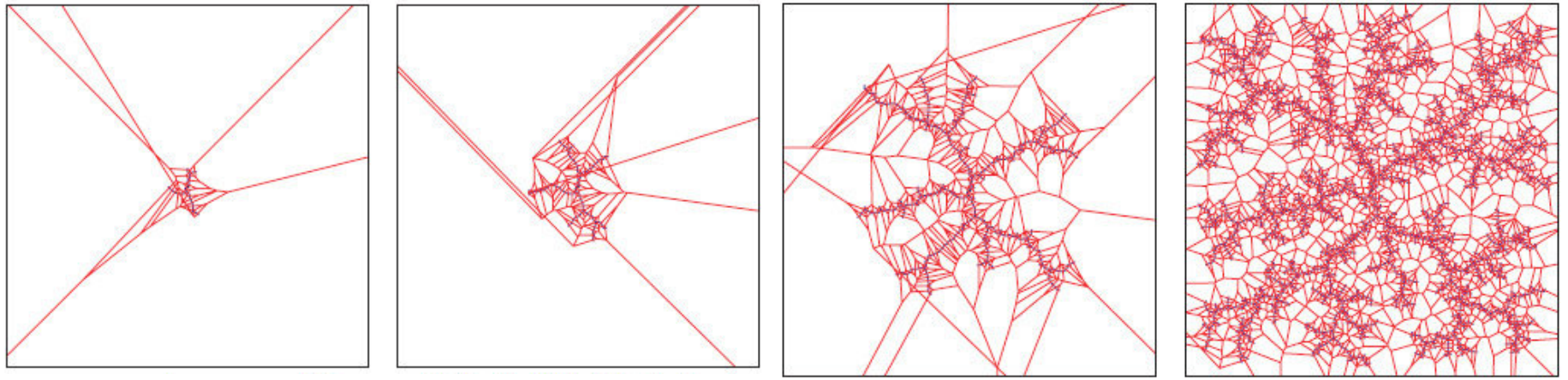
Voronoi region

Let $q_1, \ldots, q_K$ be a set of configurations on the state space $\mathcal{Q}$. The Voronoi region is defined as

$$R_k = \{q \in \mathcal{Q} \mid d(q, R_k) \leq d(q, R_j), \text{ for all } j \neq k\}$$



Voronoi Bias – RRT Analysis



- As random states are chosen from a uniform sampling distribution, it is more likely to expand an outer state.
- With this inherent Voronoi bias, the tree rapidly explores the configuration space

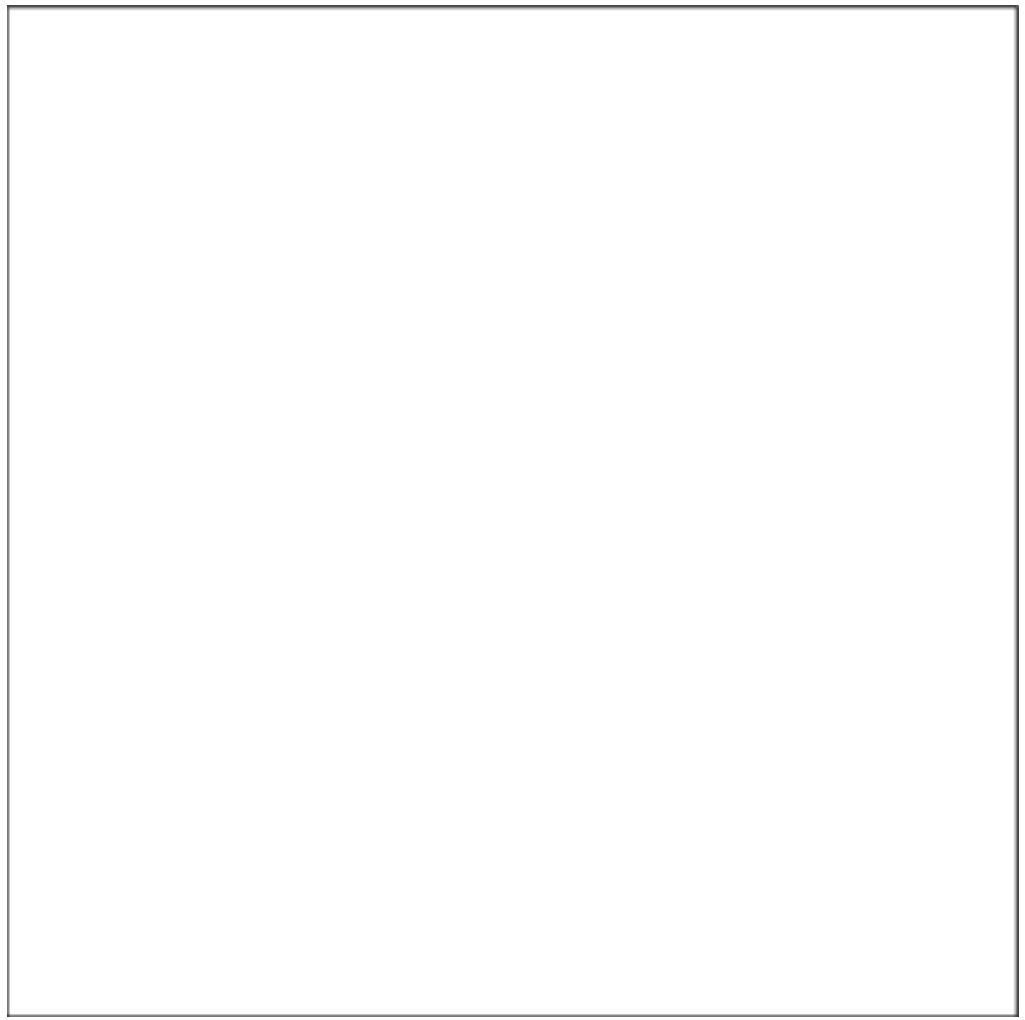
Free Space Coverage from RRT

45 iterations

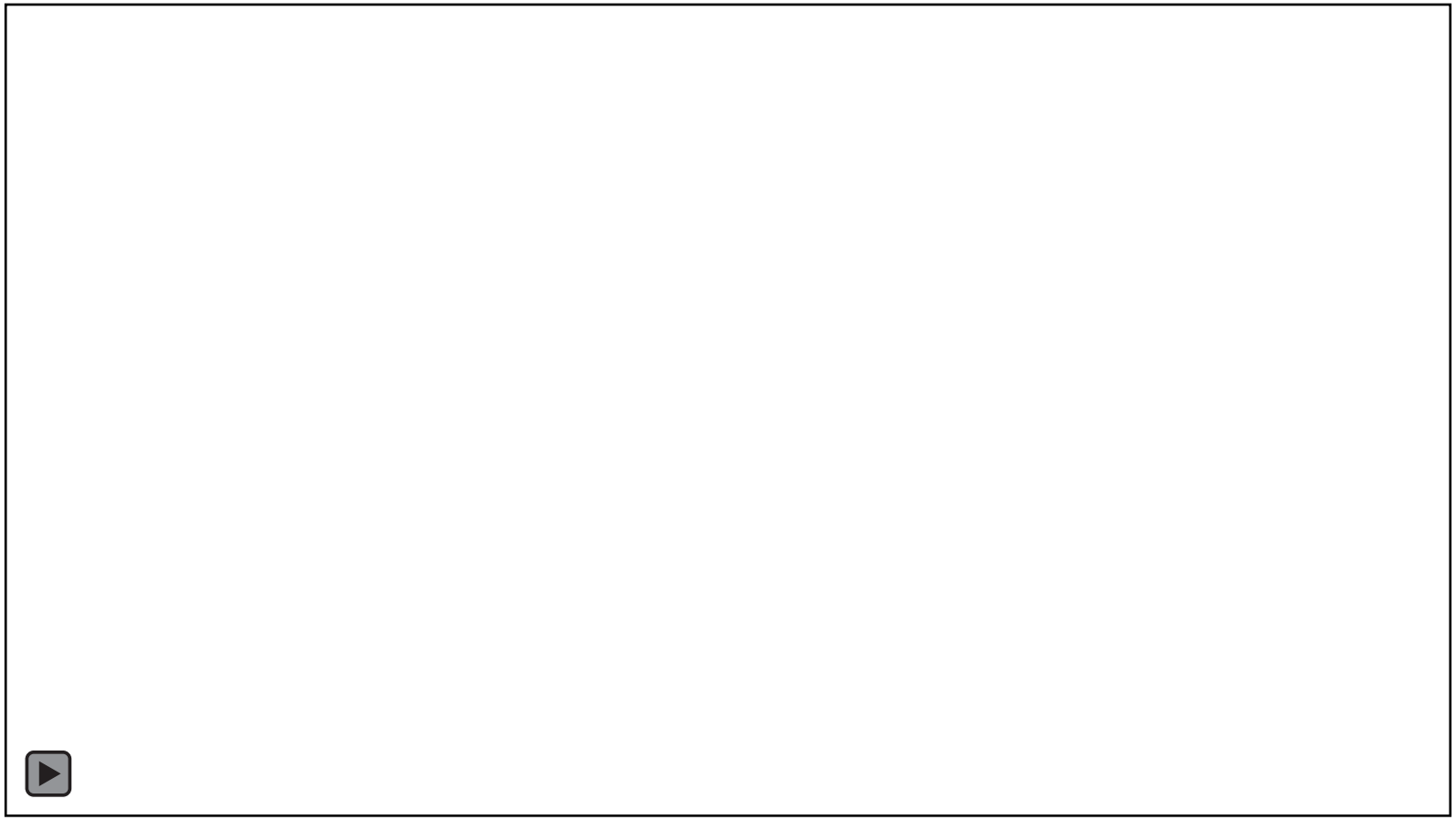
2345 iterations

Figure 5.19: In the early iterations, the RRT quickly reaches the unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

Free Space Coverage from RRT

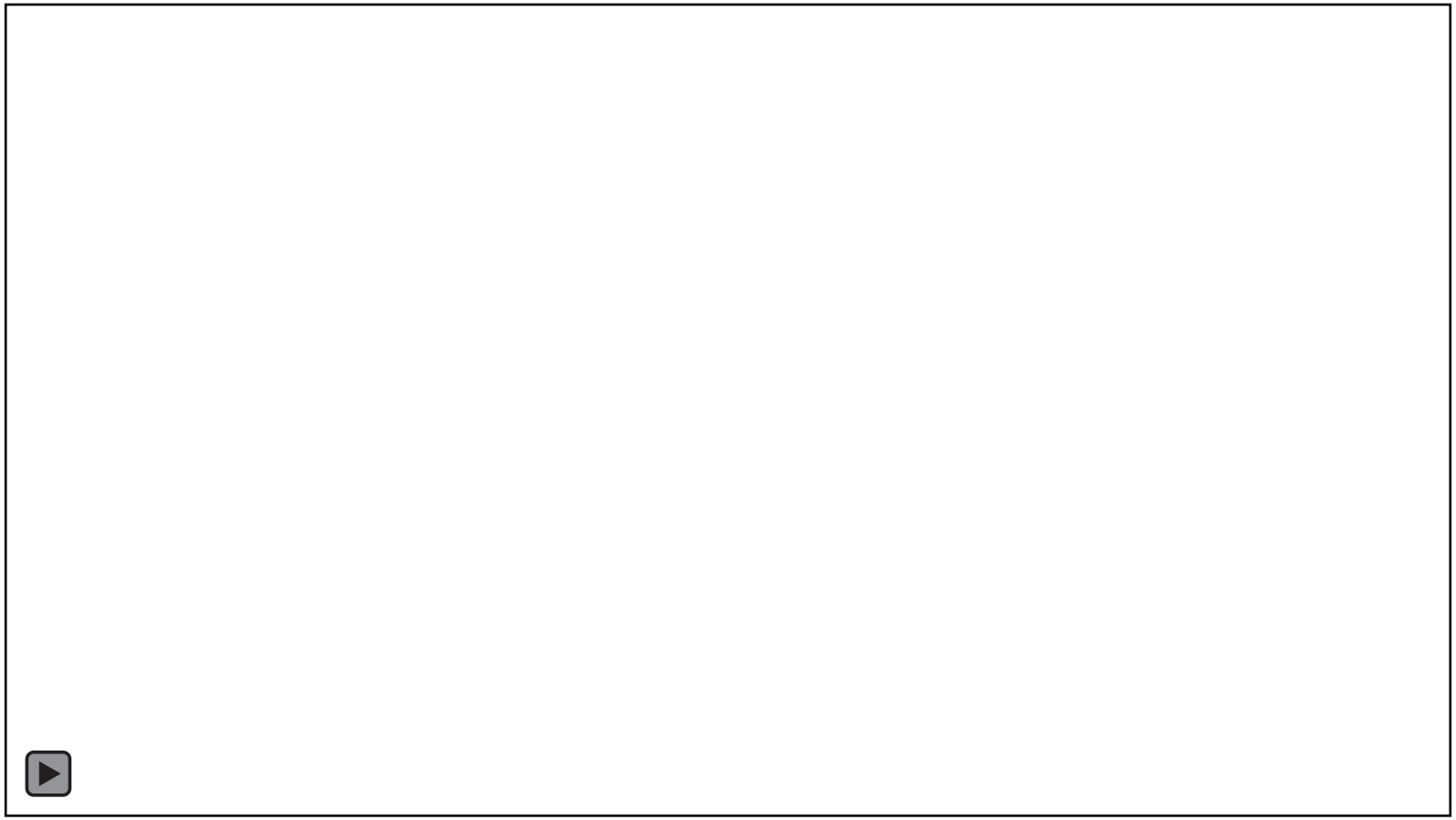


RRT-Expansion Step



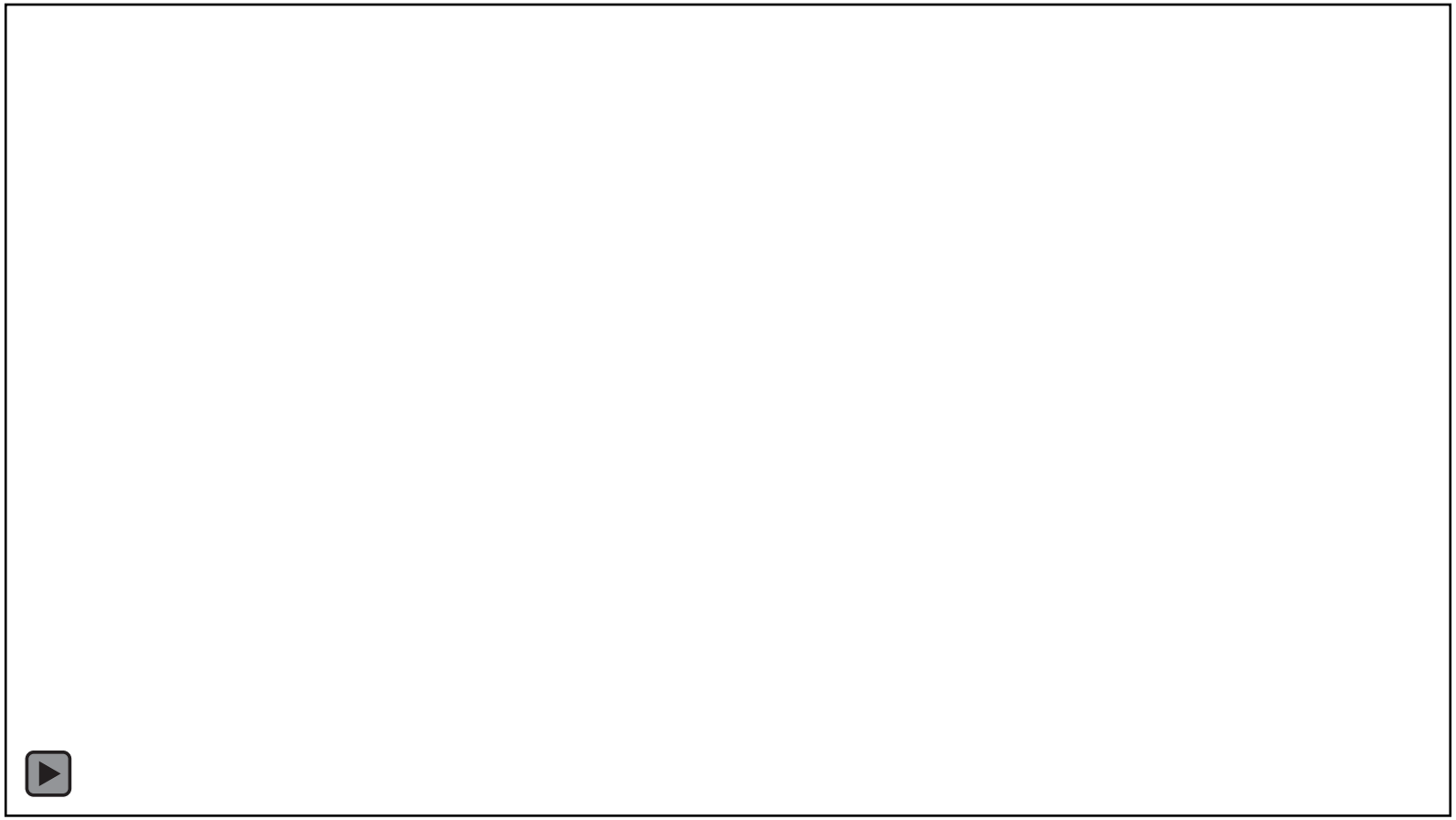
Video Credit: **Peter Murray**

RRT- Selecting node to extend



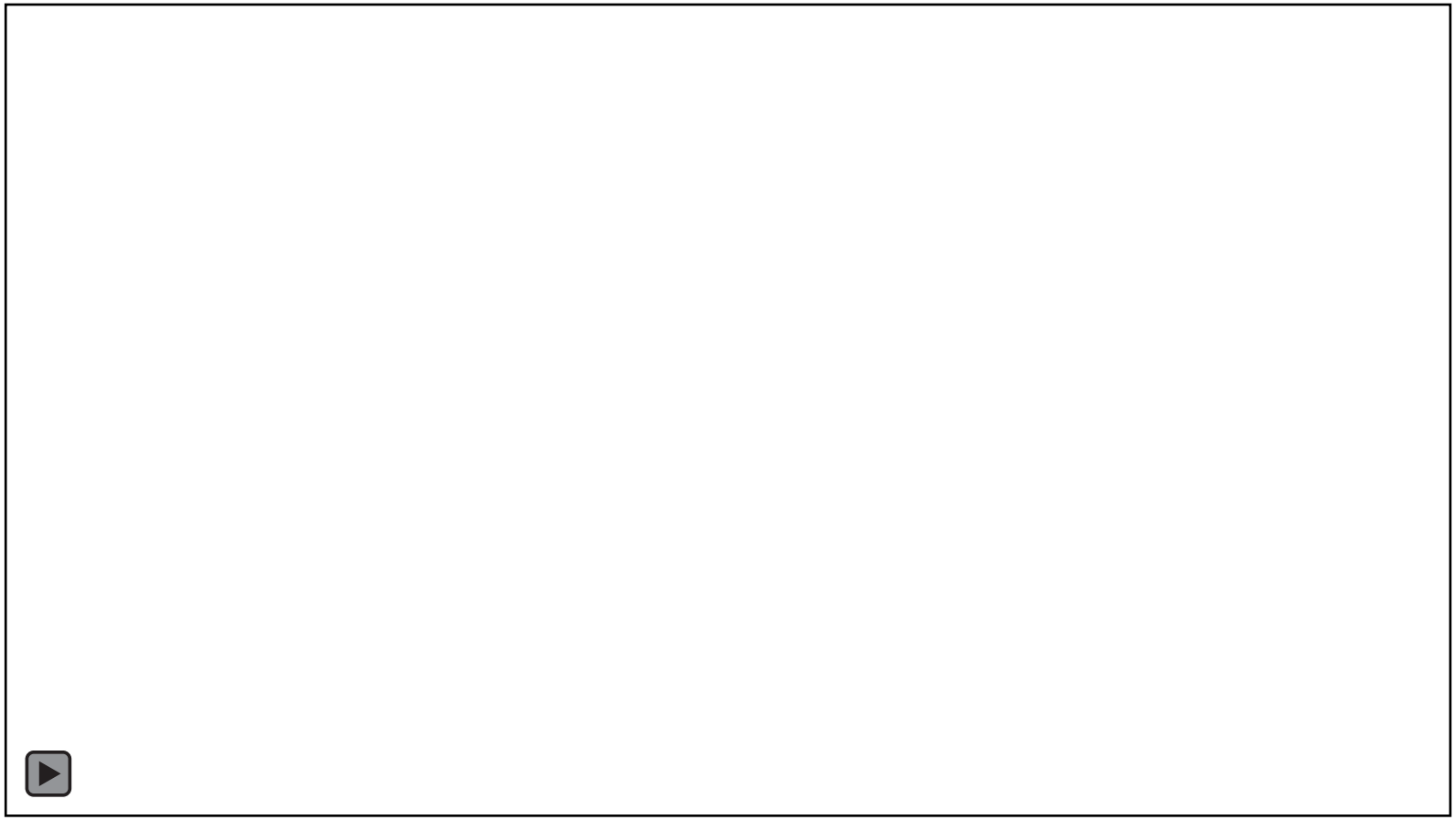
Video Credit: **Peter Murray**

RRT- Growth Illustration



Video Credit: **Peter Murray**

RRT- Voronoi -Bias



Video Credit: **Peter Murray**

Goal-Bias RRT-Improvements

Aspects for Improvement

- $\mathrm{BasicRRT}$ does not take advantage of q_{goal}
- Tree is pulled towards random directions based on the uniform sampling of Q

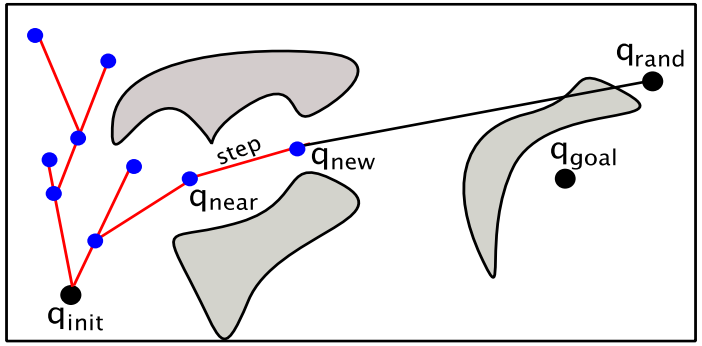
Suggested Improvements in the Literature

- Introduce goal-bias to tree growth (known as $\mathrm{GoalBiasRRT}$)
- q_{rand} is selected as q_{goal} with probability p
- q_{rand} is selected based on uniform sampling of Q with probability $1 - p$
- Probability p is commonly set to ≈ 0.05

Connect Heuristic RRT-Improvements

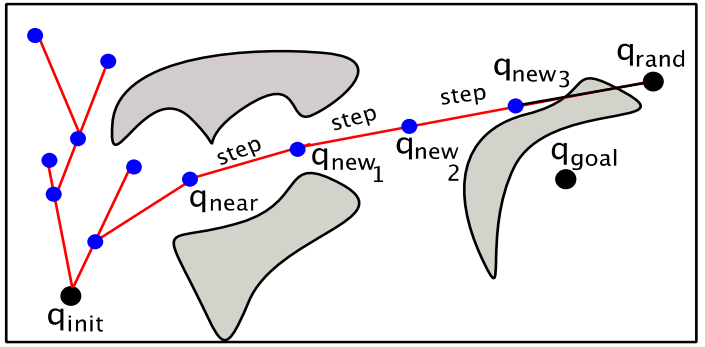
Aspects for Improvement

- BasicRRT takes only one small step when adding a new tree branch
- It slows down tree growth



Suggested Improvements in the Literature

- Take several steps until q_{rand} is reached or a collision is found (ConnectRRT)
- Add all the intermediate nodes to the tree



Bi-directional Trees

- **Idea:** grow two trees, rooted at q_{init} and q_{goal} towards each other
- Bi-directional trees improve computational efficiency compared to a single tree
- Fewer configurations in each tree, which imposes less of a computational burden
- Each tree explores a different part of the configuration space

BiTree(qinit , qgoal)

1: T_{init} ← create tree rooted at q_{init}
2: T_{goal} ← create tree rooted at q_{goal}
3: **while** solution not found **do**
4: add new branch to T_{init}
5: add new branch to T_{goal}
6: attempt to connect neighboring configurations from the two trees
7: **if** successful
8: **return** path from q_{init} to q_{goal}

RRT-Connect

```
BUILD_RRT(q_init)
1  T.init(q_init);
2  for k = 1 to K do
3      q_rand ← RANDOM_CONFIG();
4      EXTEND(T, q_rand);
5  Return T
```

```
EXTEND(T, q)
1  q_near ← NEAREST_NEIGHBOR(q, T);
2  if NEW_CONFIG(q, q_near, q_new) then
3      T.add_vertex(q_new);
4      T.add_edge(q_near, q_new);
5      if q_new = q then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

Figure 2: The basic RRT construction algorithm.

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

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

Figure 5: The RRT-Connect algorithm.

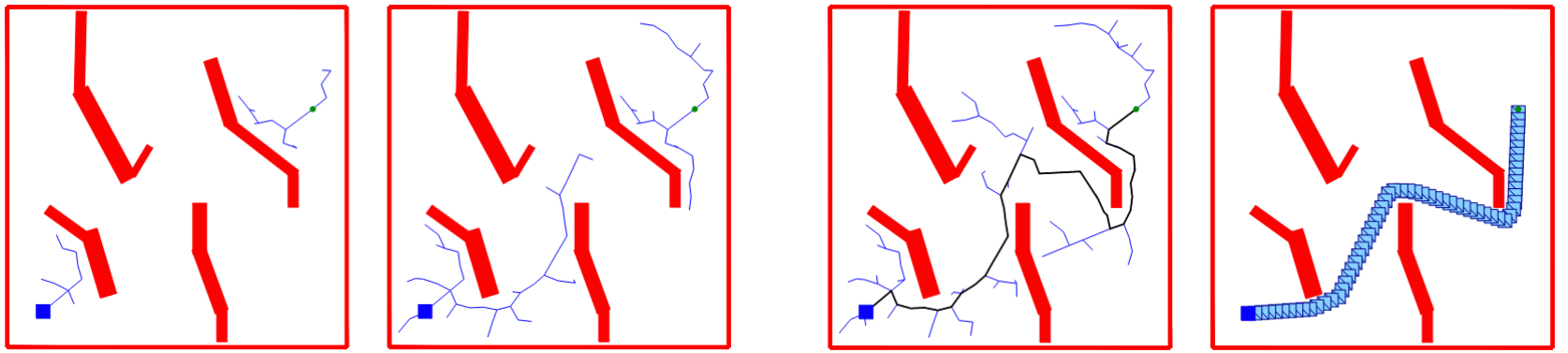
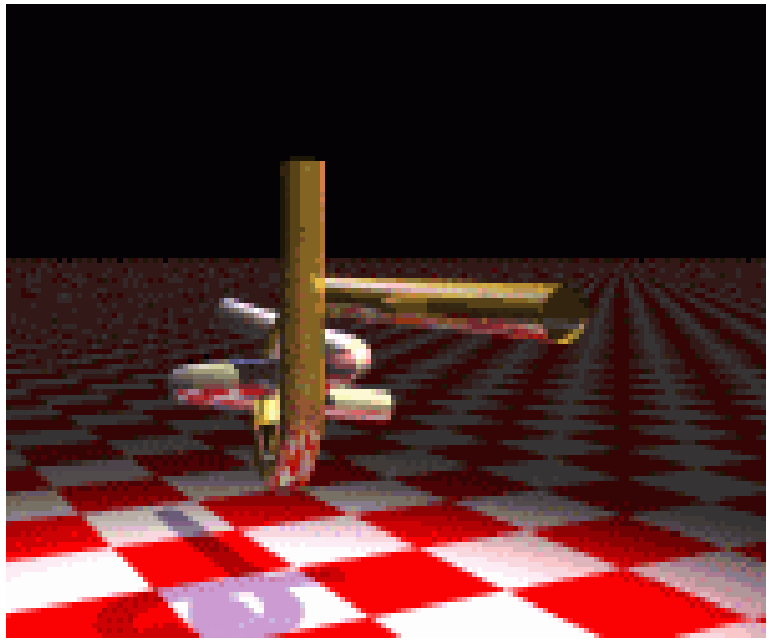


Figure 6: Growing two trees towards each other.

RRT Examples: The Alpha Puzzle

- VERY hard 6DOF motion planning problem (long, winding narrow passage)
- "*In 2001, it was solved by using a balanced bidirectional RRT, developed by James Kuffner and Steve LaValle. There are no special heuristics or parameters that were tuned specifically for this problem. On a current PC (circa 2003), it consistently takes a few minutes to solve*" –RRT website
- RRT became famous in large part because it was able to solve this puzzle



Expansive-Space Tree (EST)

- **Idea:** Push the tree frontier in the free configuration space

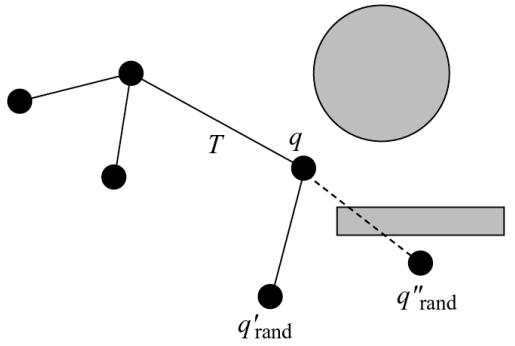
- EST relies on a probability distribution to guide tree growth
- EST associates a weight $w(q)$ with each tree configuration q
- $w(q)$ is a running estimate on importance of selecting q as the tree configuration from which to add a new tree branch
  - $w(q) = \frac{1}{1+\deg(q)}$
  - $w(q)$ = 1/(1+ number of neighbors near q)
  - combination of different strategies

Expansive-Space Tree (EST)

- **Idea:** Push the tree frontier in the free configuration space

SelectConfigFromTree

- select q in T with probability $\frac{w(q)}{\sum_{q \in T} w(q')}$



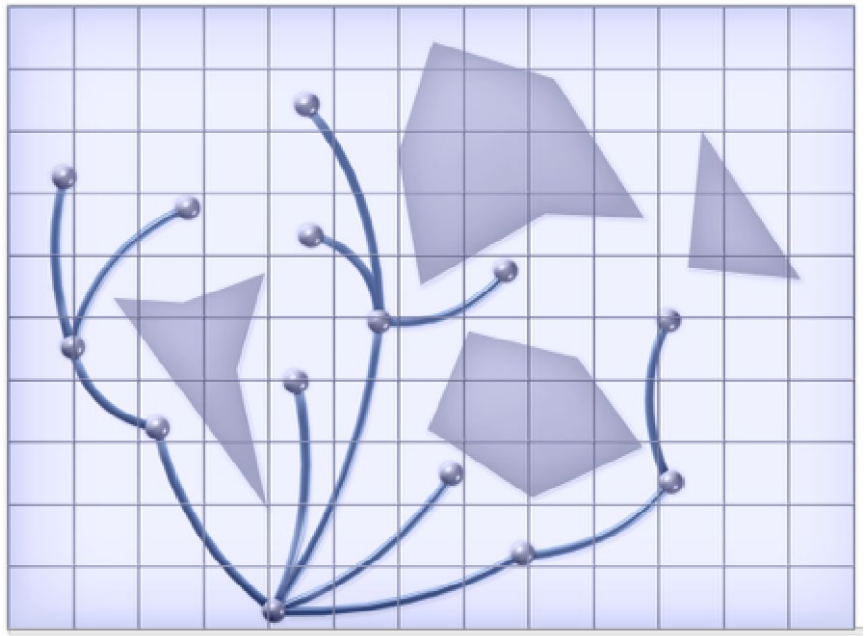
AddTreeBranchFromConfig(T , q)

- q_{near} ← sample a collision-free configuration near q
- $path$ ← generate path from q to q_{near}
- **if** path is collision-free **then** add q_{near} and (q, q_{near}) to T

Combine Sampling with Some Estimation of Coverage

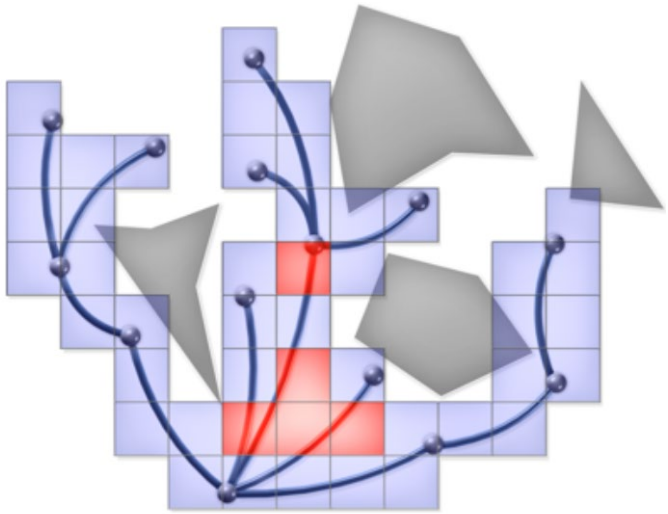
- EST:
  - Uses density of nodes to guide expansion (density bias)

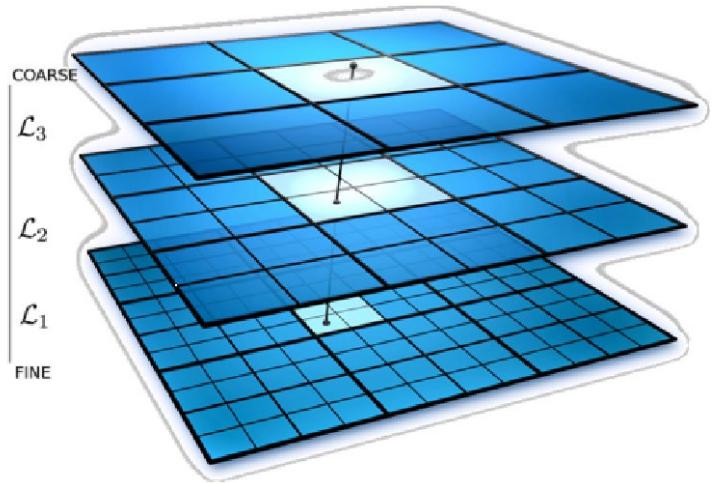
- Can we do better?
  - Use a grid to for density estimation and sampling



Combine Sampling with Some Estimation of Coverage

- KPIECE:
  - sampling and some estimation of coverage
  - Keeps tract of coverage by using discretization and by distinguishing the boundary (blue-squares) from the covered space (red-squares).
  - Keeping of coverage can be done in a hierarchical fashion. - Projections may be used.

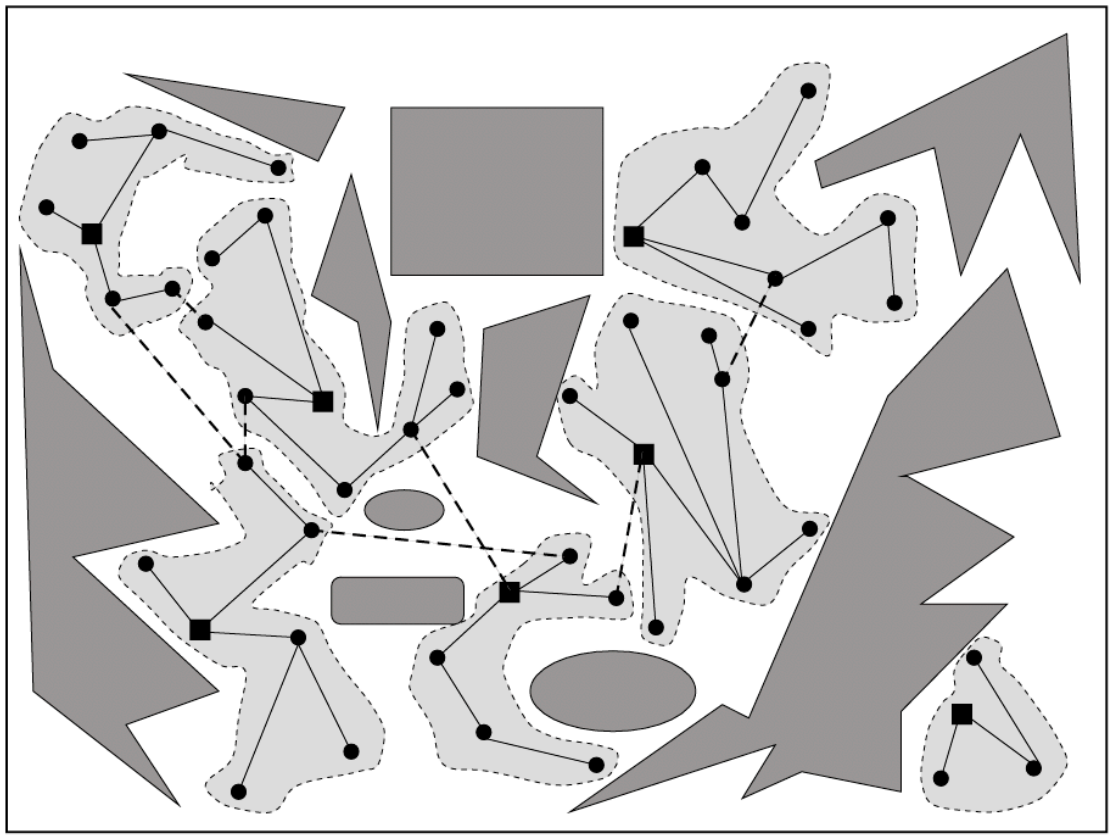




Sampling-based Roadmap of Trees (SRT)

Sampling-based Roadmap of Trees (SRT)

- Hierarchical planner
- Top level performs global sampling
  - PRM-based
- Bottom level performs local sampling
  - tree-based,
  - e.g., RRT, EST

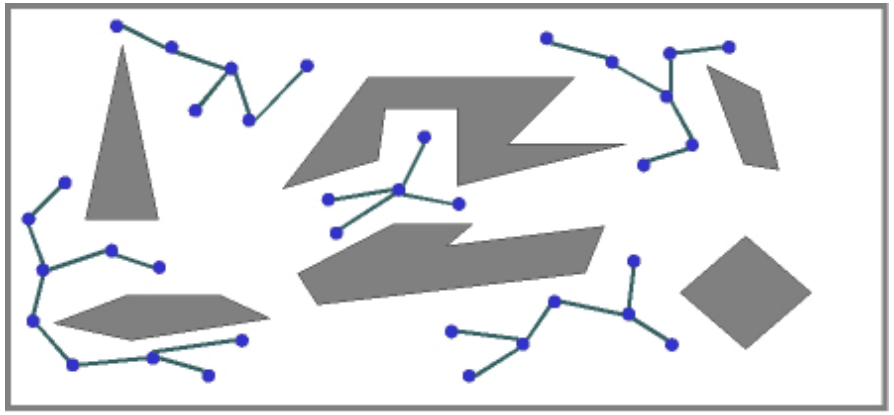


- Combines advantages of global and local sampling

Sampling-based Roadmap of Trees (SRT)

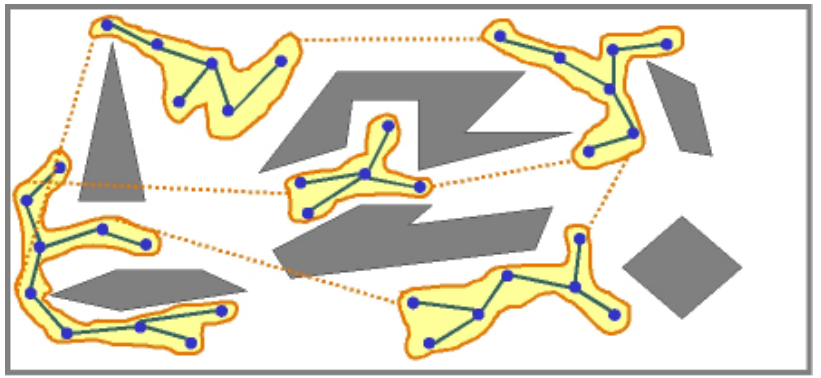
1. CreateTreesInRoadmap

1: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$
2: **while** $|V| < n_{trees}$ **do**
3: $T \leftarrow$ create tree rooted at a collision-free configuration
4: use tree planner to grow T for some time add T to roadmap vertices V



2. SelectWhichTreesToConnect

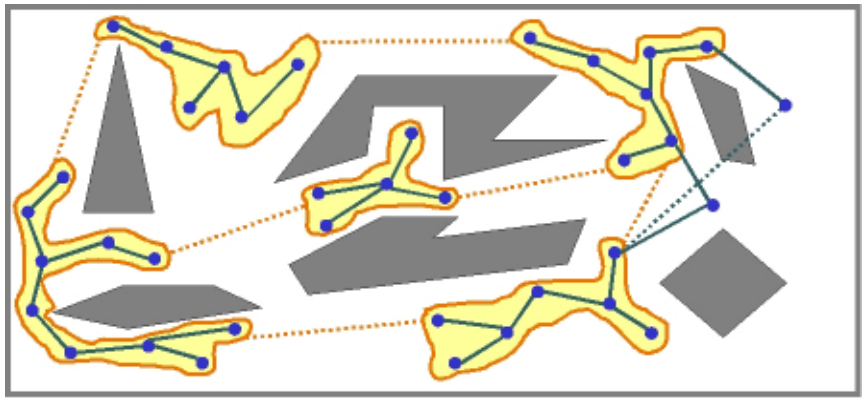
1: $E_{pairs} \leftarrow \emptyset$
2: **for** each $T \in V$ **do**
3: $S_{neighs} \leftarrow k$ nearest trees in V to T
4: $S_{rand} \leftarrow r$ random trees in V
5: $E_{pairs} \leftarrow E_{pairs} \cup \{(T, T') \mid T' \in (S_{neighs} \cup S_{rand})\}$



Sampling-based Roadmap of Trees (SRT)

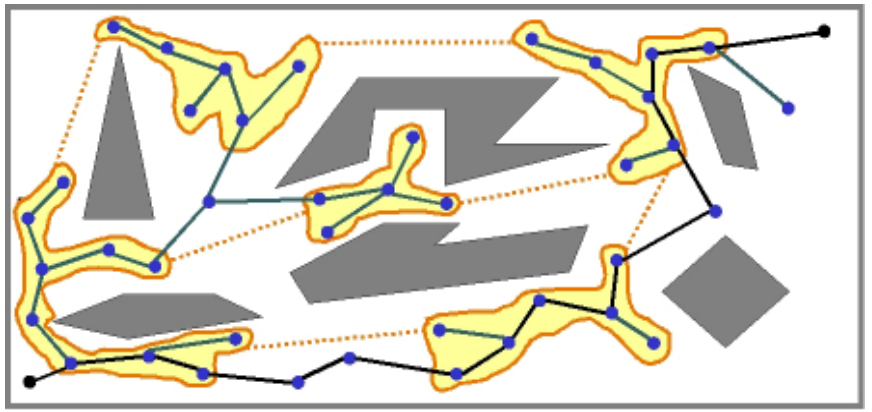
3. ConnectTreesInRoadmap

1: **for** each $(T_1, T_2) \in E_{pairs}$ **do**
2: **if** $\text{AreTreesConnected}(T_1, T_2) = false$ **then**
3: run bi-directional tree planner to connect T_1 to T_2
4: **if** connection successful **then**
5: add edge (T_1, T_2) to roadmap



3. SolveQuery (q_{init}, q_{goal})

1: $T_{init} \leftarrow$ create tree rooted at q_{init}
2: $T_{goal} \leftarrow$ create tree rooted at q_{goal}
3: connect T_{init} and T_{goal} to roadmap
4: search roadmap graph for solution



Sampling-Based Planners

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

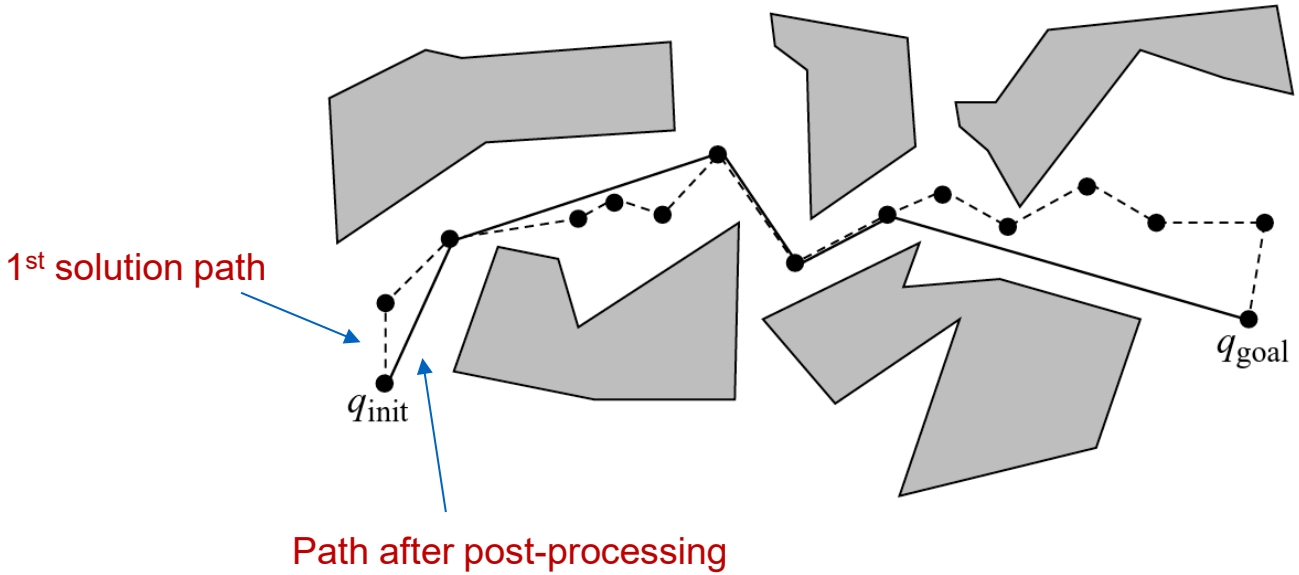
- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Path Smoothing

- Sophisticated path smooth methods exists
- In fact, sampling-based planners are rarely used without smoothing

Path Post Processing

- Solution paths produced by PRM planners tend to be long and non-smooth (due to sampling and edge connections)
- *Post processing* is commonly used to improve the quality of the paths
- A common practice is to repeatedly replace long paths by short paths



Path Post Processing

- Path smoothing methods

1. $\boldsymbol{ShortcutPath_1\ (q_1, q_2, \dots, q_n)}$

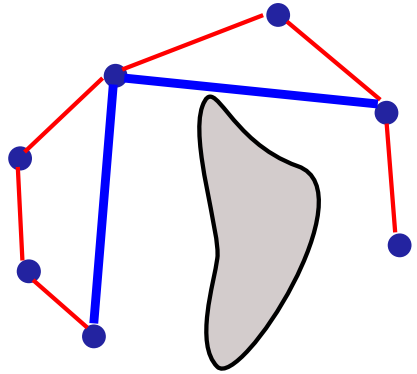
   1: **for** several times **do**

   2: select i and j uniformly at random from $1, 2, \dots, n$

   3: attempt to directly connect q_i to q_j

   4: if successful, remove the in-between nodes, i.e.,

   $q_{i+1}, \dots, q_j$



2. $\boldsymbol{ShortcutPath_2\ (q_1, q_2, \dots, q_n)}$

   1: **for** several times **do**

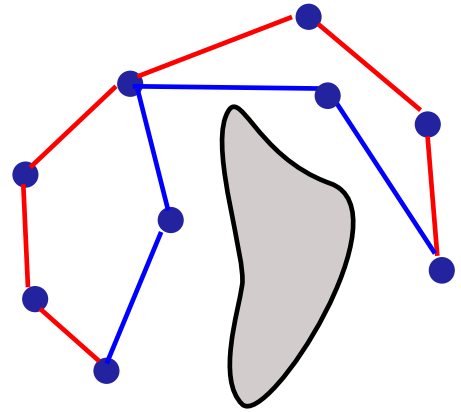
   2: select i and j uniformly at random from

   $1, 2, \dots, n$

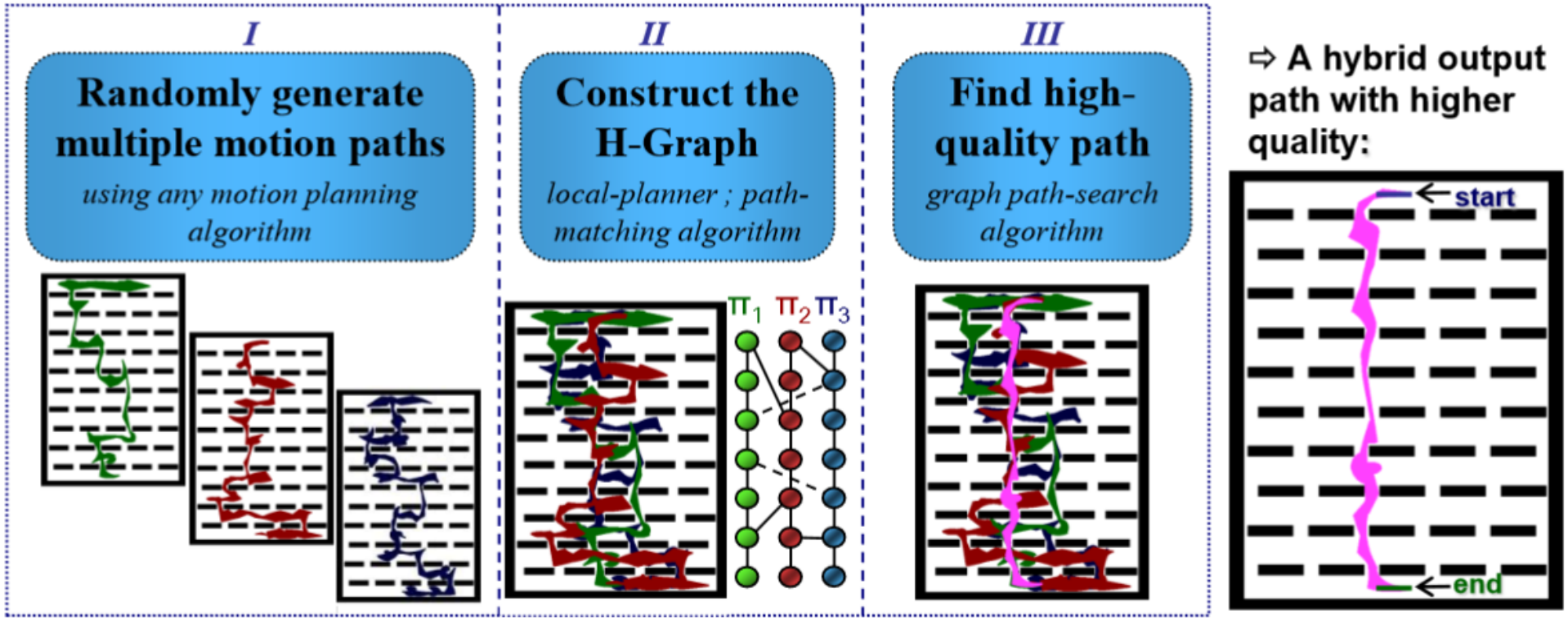
   3: $q \leftarrow$ generate collision-free sample

   4: if successful, replace the in-between nodes

   $q_{i+1}, \dots, q_j$ by q



Hybrid-Pathing



Raveh, Barak, Angela Enosh, and Dan Halperin. "A little more, a lot better: Improving path quality by a path-merging algorithm." *IEEE Transactions on Robotics* 27.2 (2011): 365-371.