CSE-571
Sampling-Based Motion Planning

Built on Dieter's Spring 2020 slides
Slides based on Pieter Abbeel, Zoe McCarthy
Many images from Lavalle, Planning Algorithms

Motion Planning: Outline

- Configuration Space
- Probabilistic Roadmap
- Rapidly-exploring Random Trees (RRTs)
- Extensions
- Smoothing

Configuration Space (C-Space)

Any configuration of a robot can be described as a unique point in its configuration space (C-space)

Configuration Space (C-Space)

Any configuration of a robot can be described as a unique point in its configuration space (C-space)

obstacles → configuration space

obstacles *Workspace*

Configuration Space

Configuration Space (C-Space)

Any configuration of a robot can be described as a unique point in its configuration space (C-space)

obstacles → configuration space

obstacles *Workspace*

Configuration Space

(2 DOF: translation only, no rotation)

Configuration Space (C-Space)

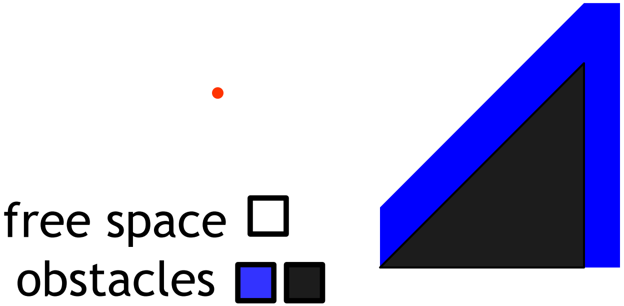
Any configuration of a robot can be described as a unique point in its configuration space (C-space)

obstacles → configuration space

obstacles *Workspace*

Configuration Space

(2 DOF: translation only, no rotation)



Configuration Space (C-Space)

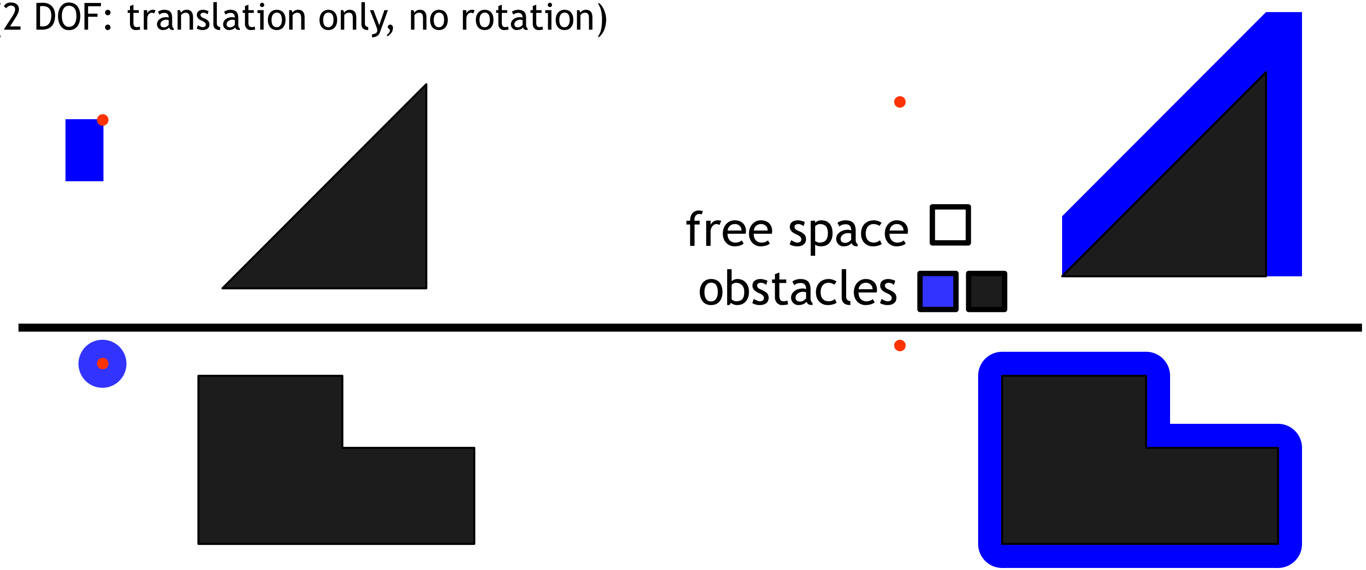
Any configuration of a robot can be described as a unique point in its configuration space (C-space)

obstacles → configuration space

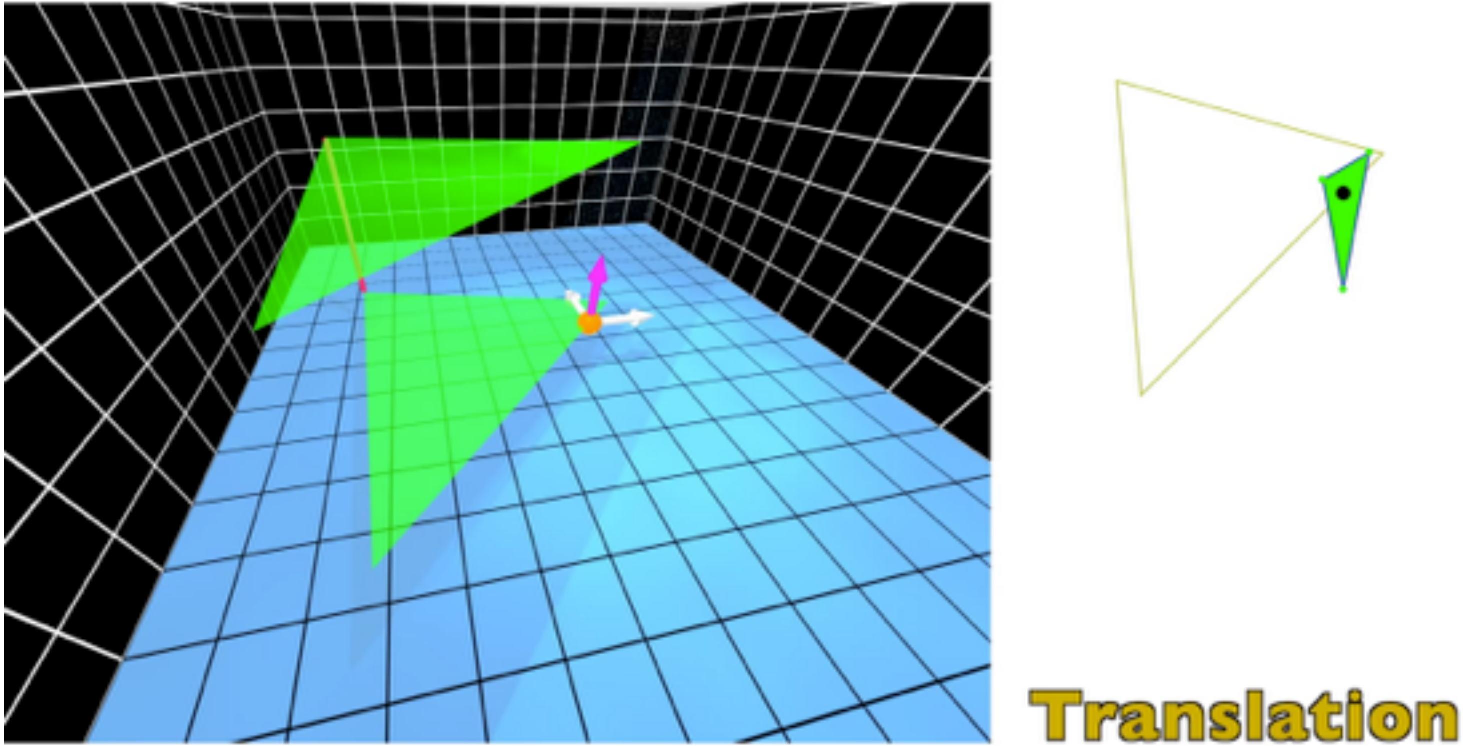
obstacles *Workspace*

Configuration Space

(2 DOF: translation only, no rotation)

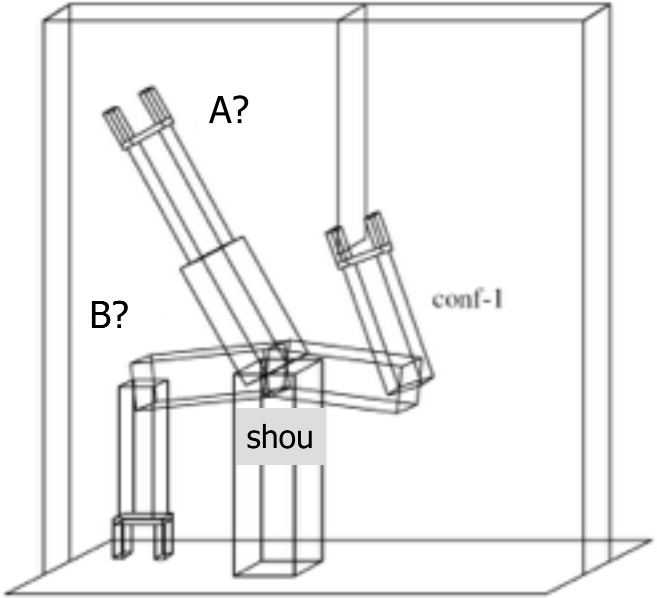


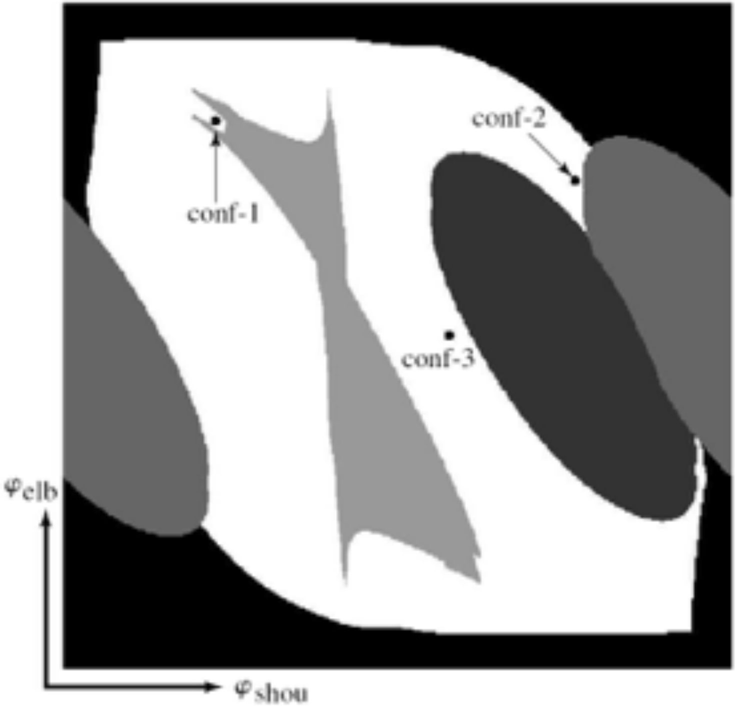
Configuration Space (C-Space)



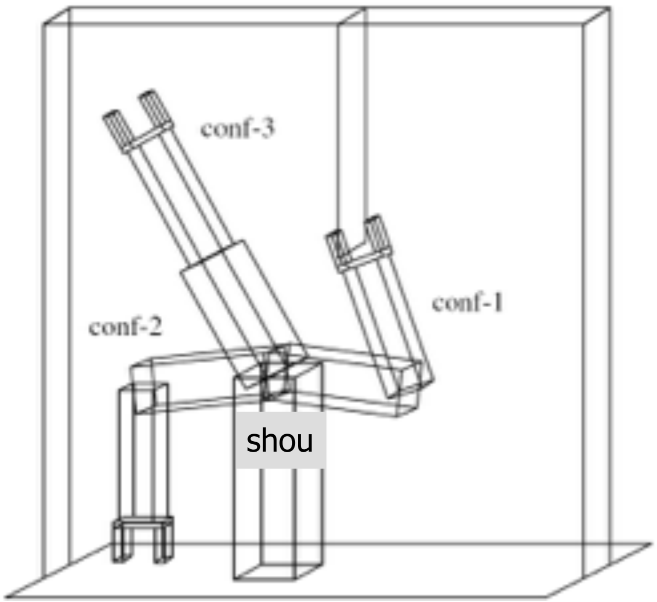
Visualization developed by Dror Atariah and Günter Rote - https://www.youtube.com/watch?v=SBFwgR4K1Gk

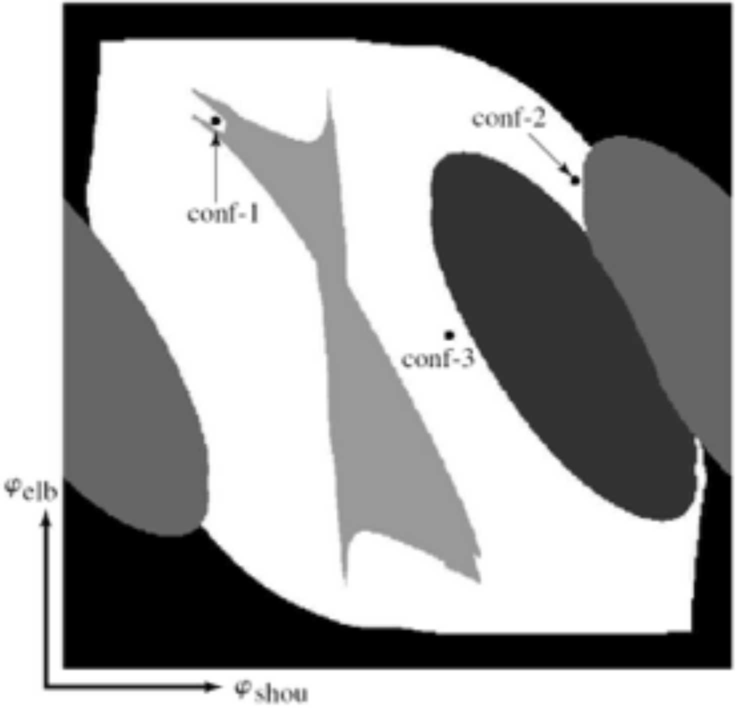
Motion planning



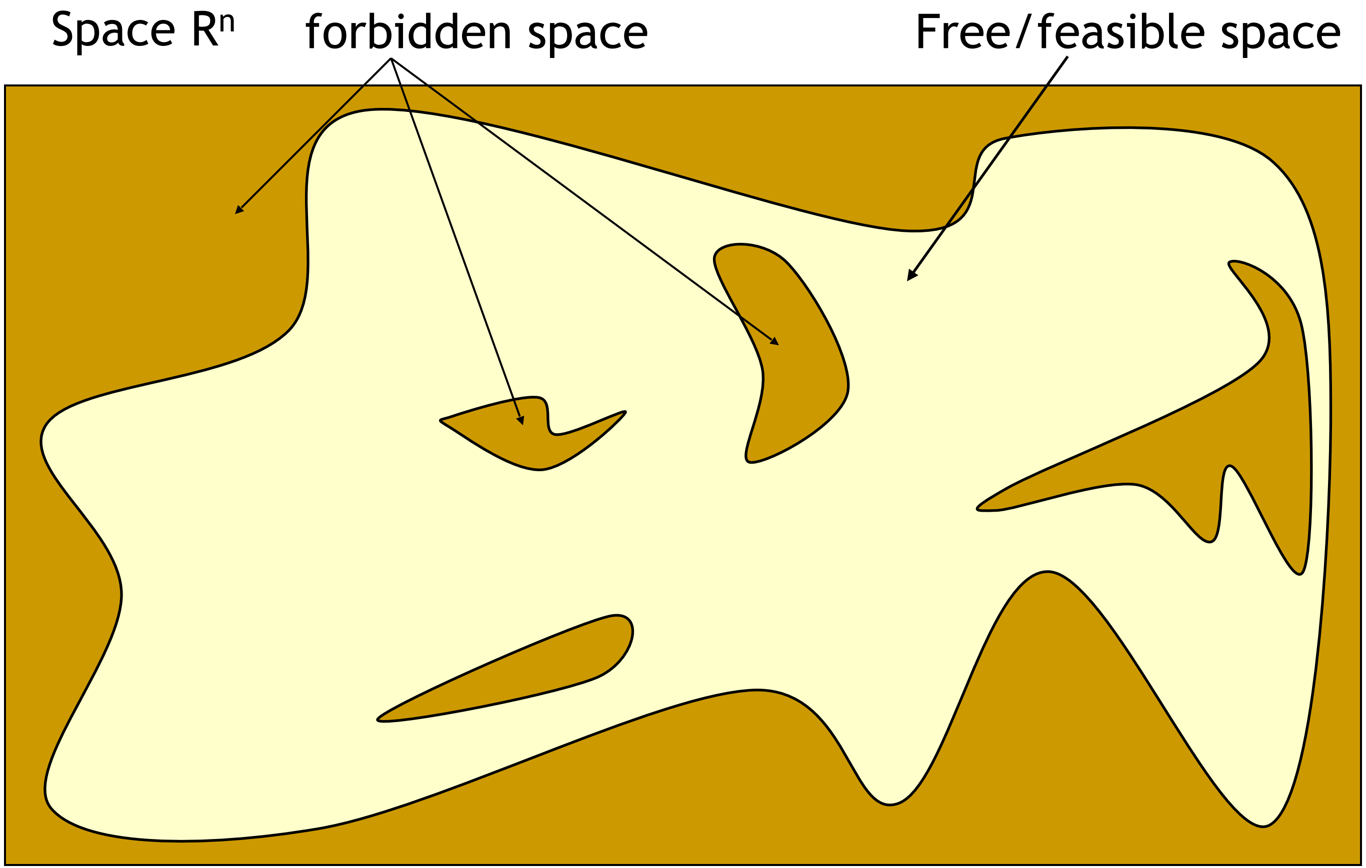


Motion planning



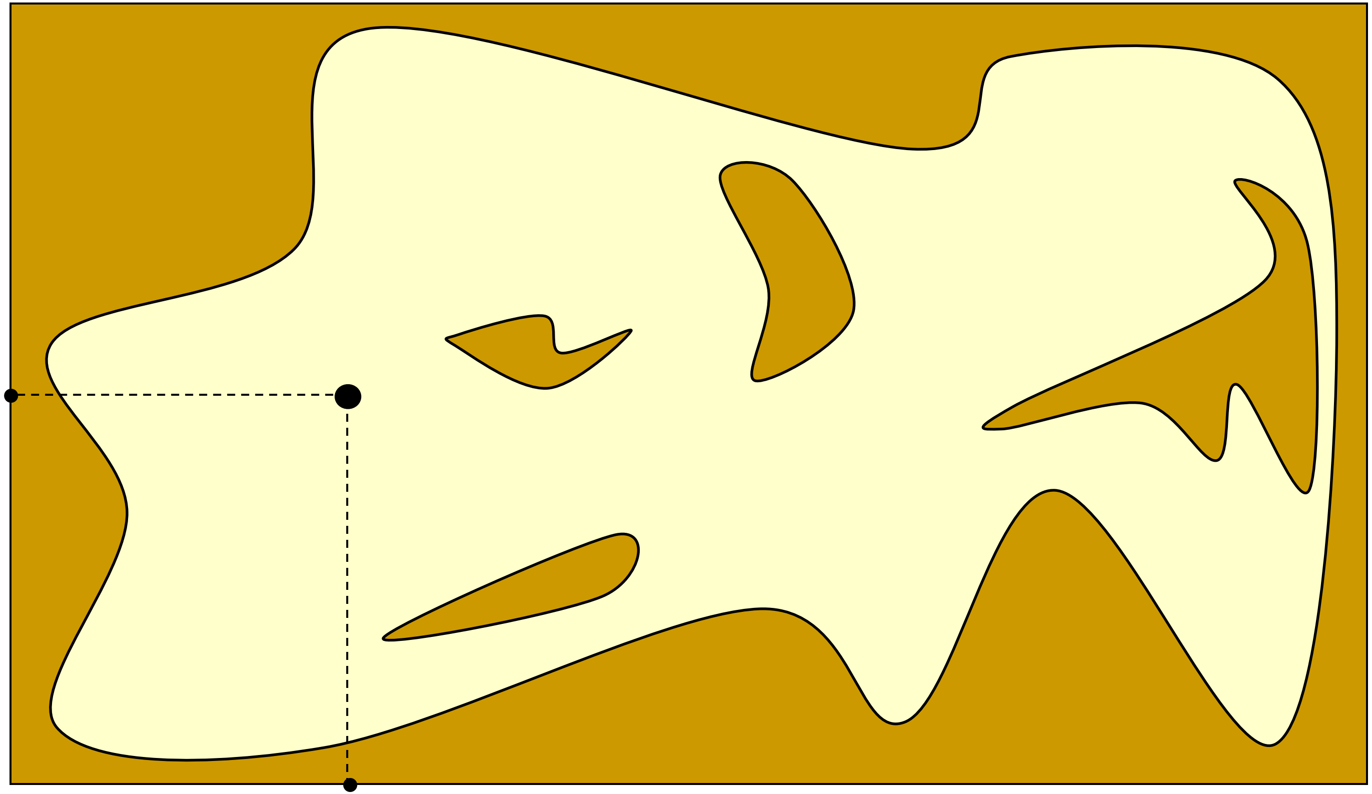


Probabilistic Roadmap (PRM)



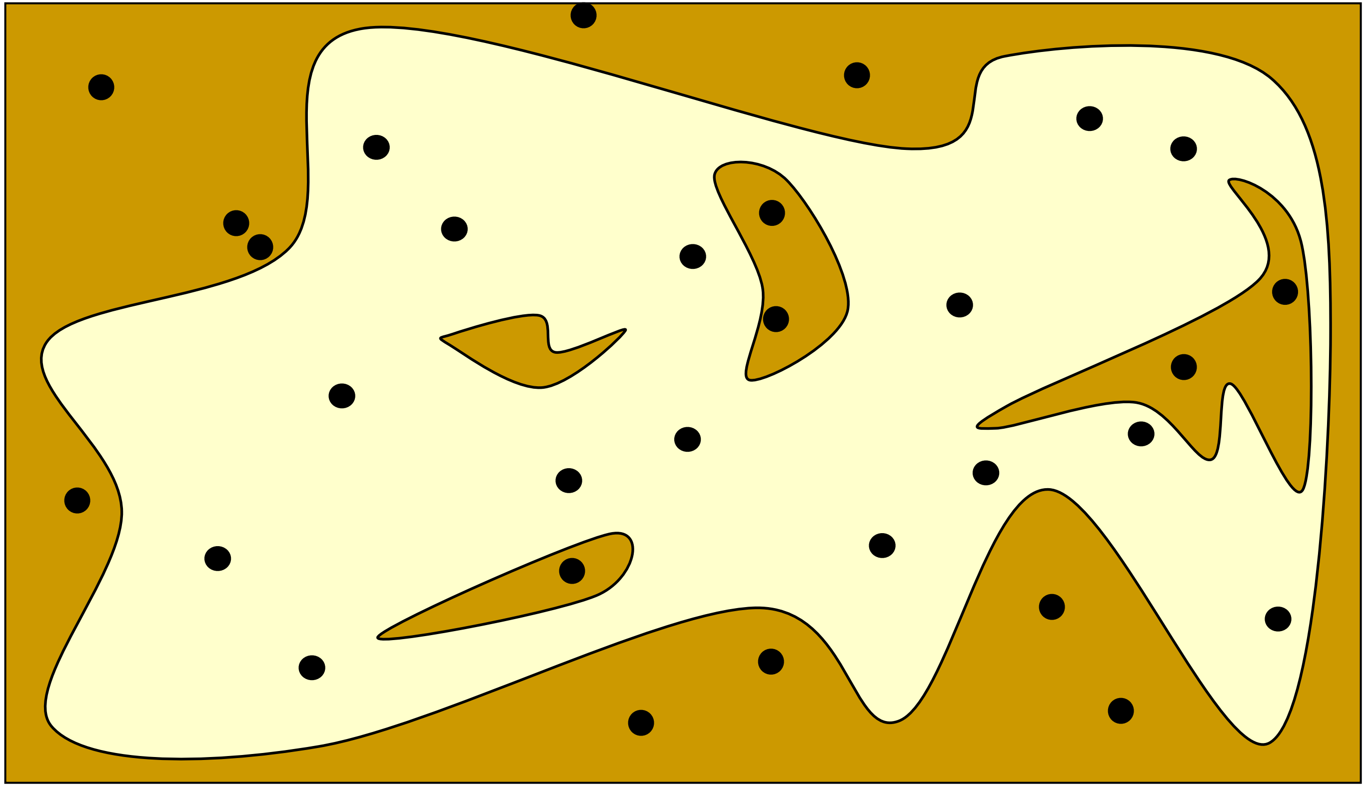
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



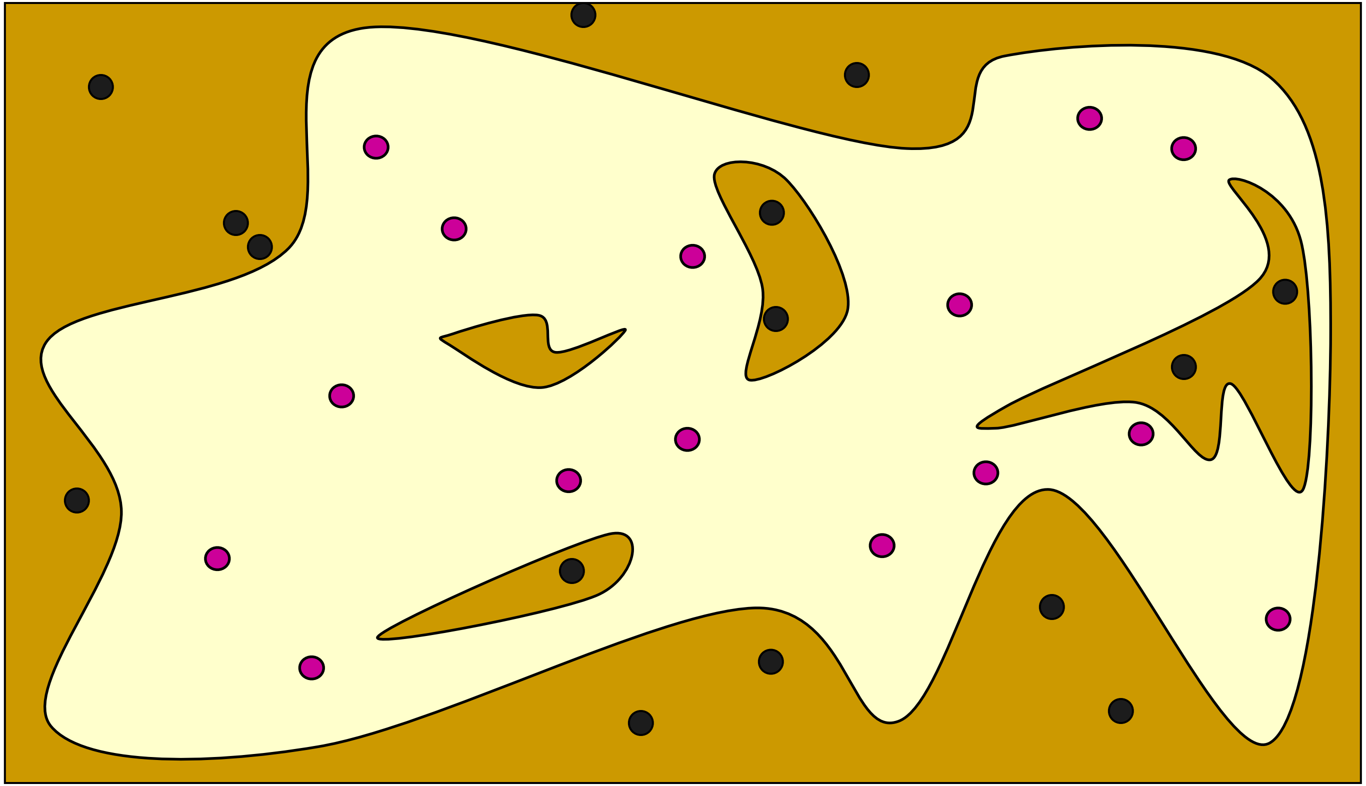
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



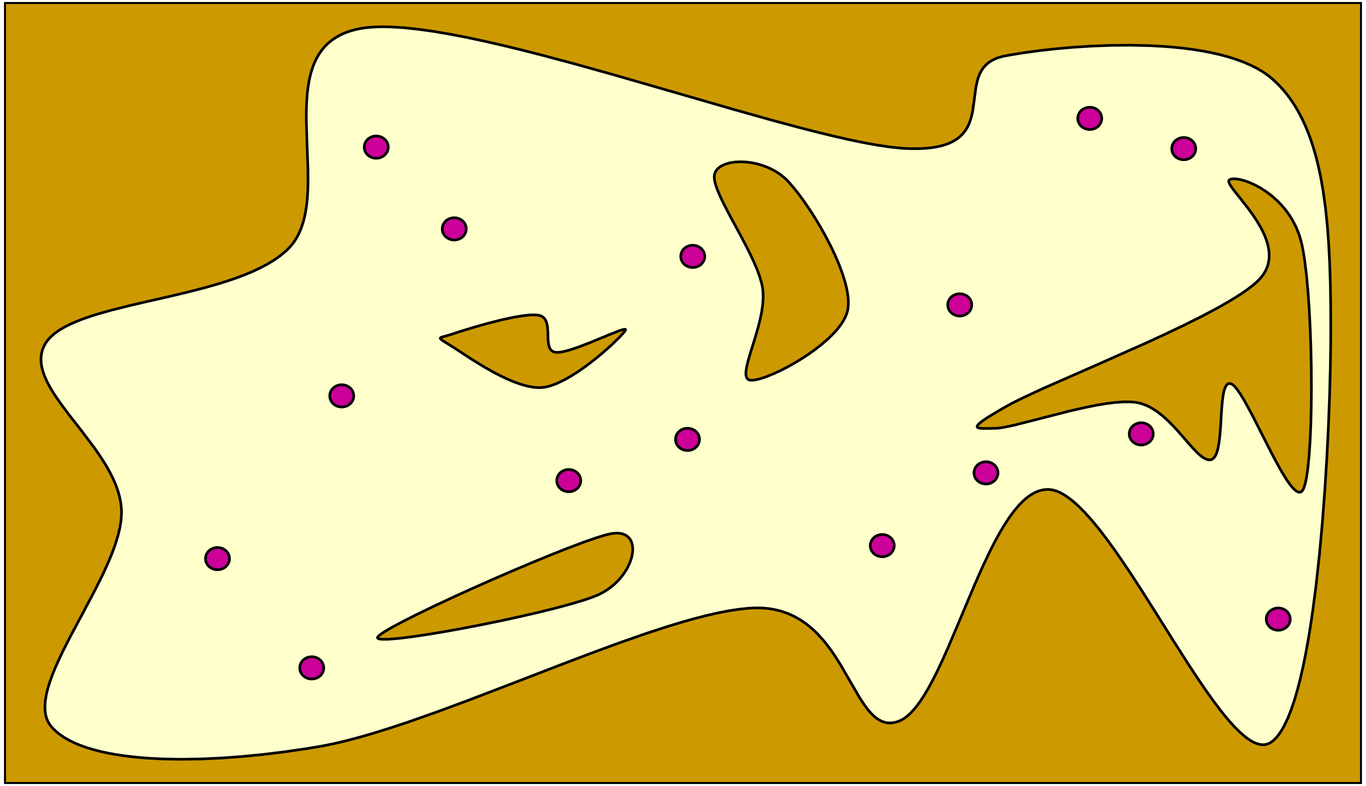
Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision



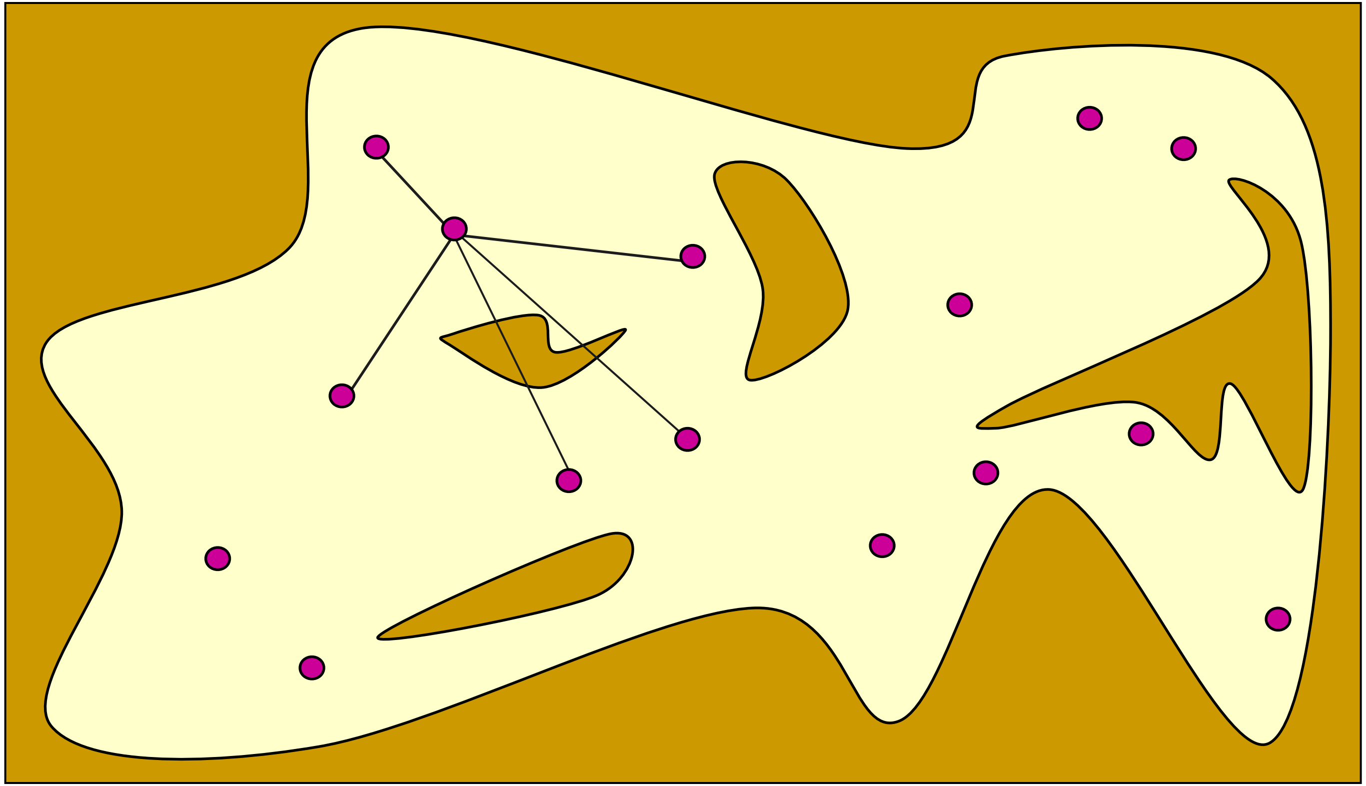
Probabilistic Roadmap (PRM)

The collision-free configurations are retained as milestones



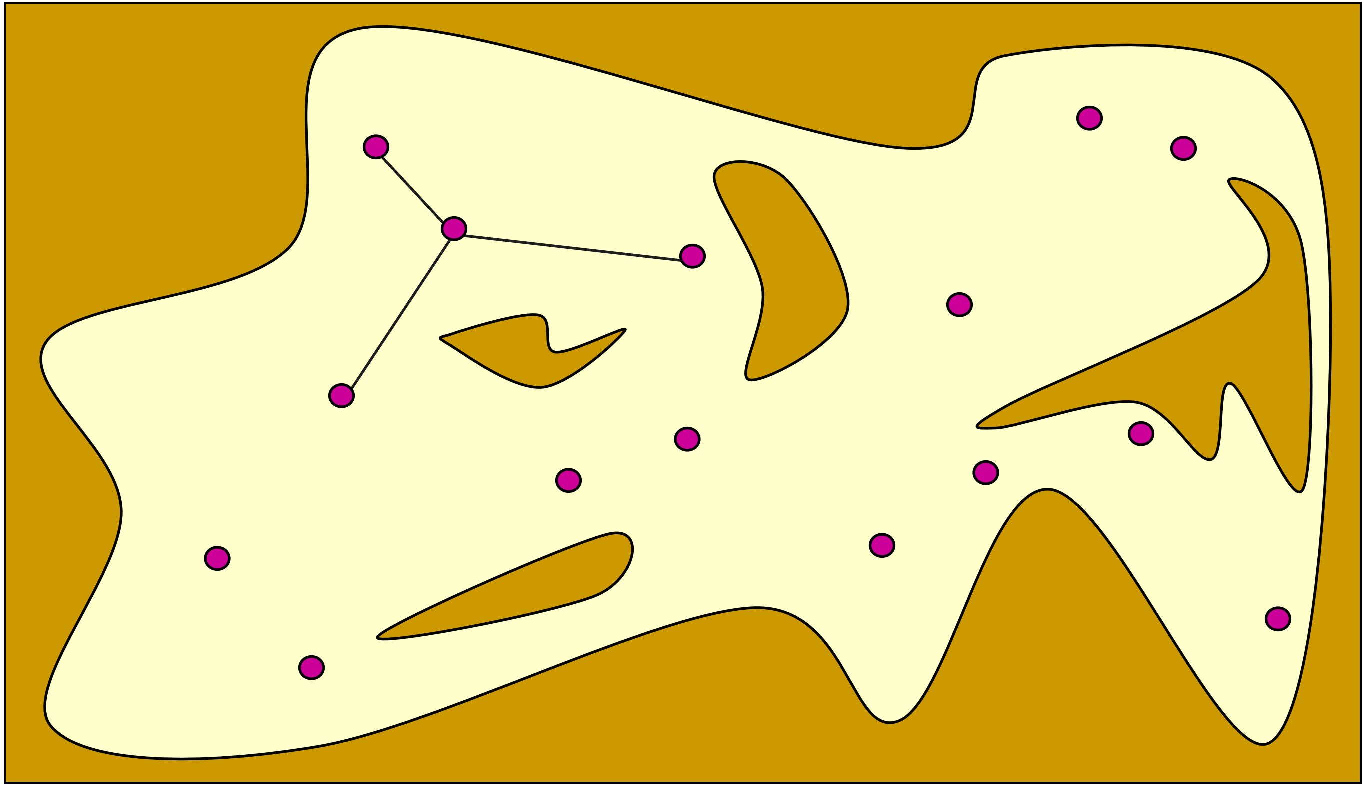
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



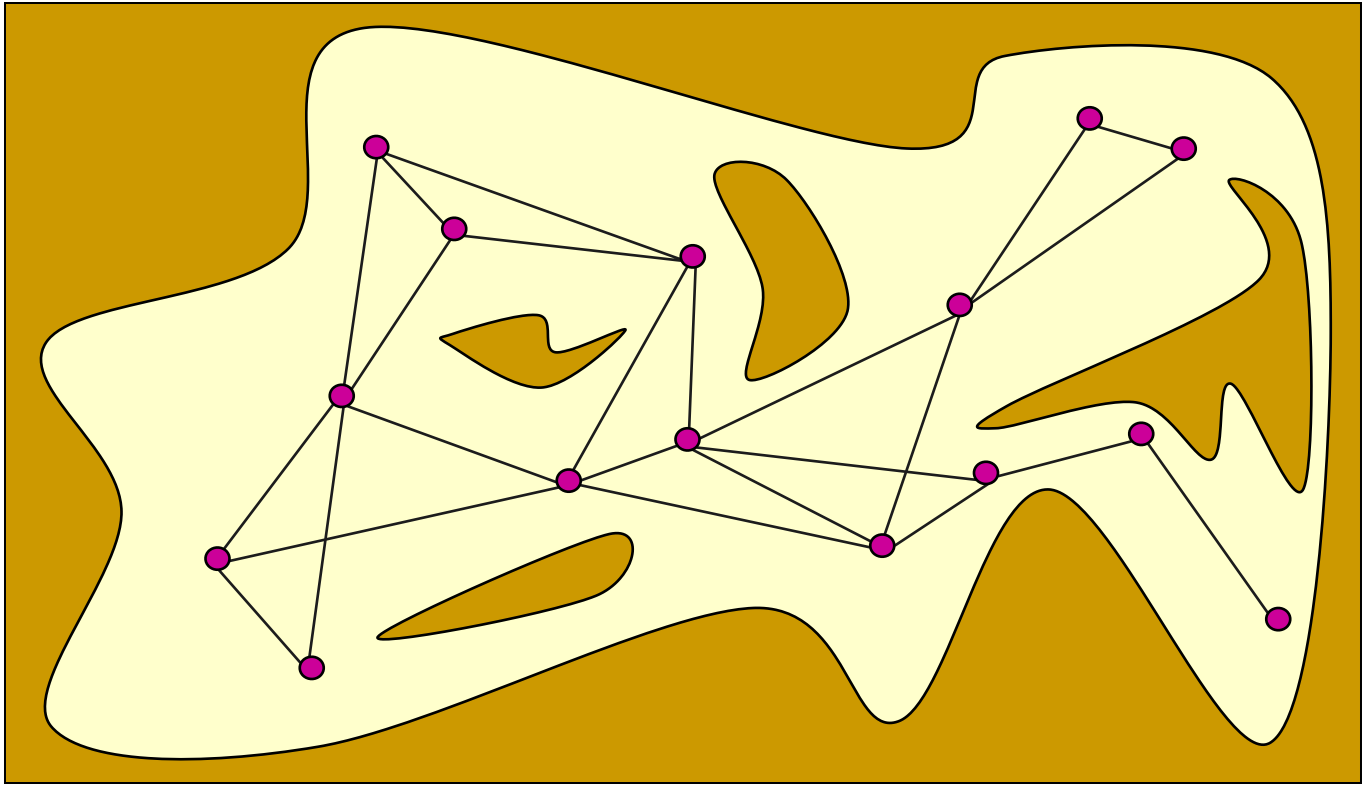
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



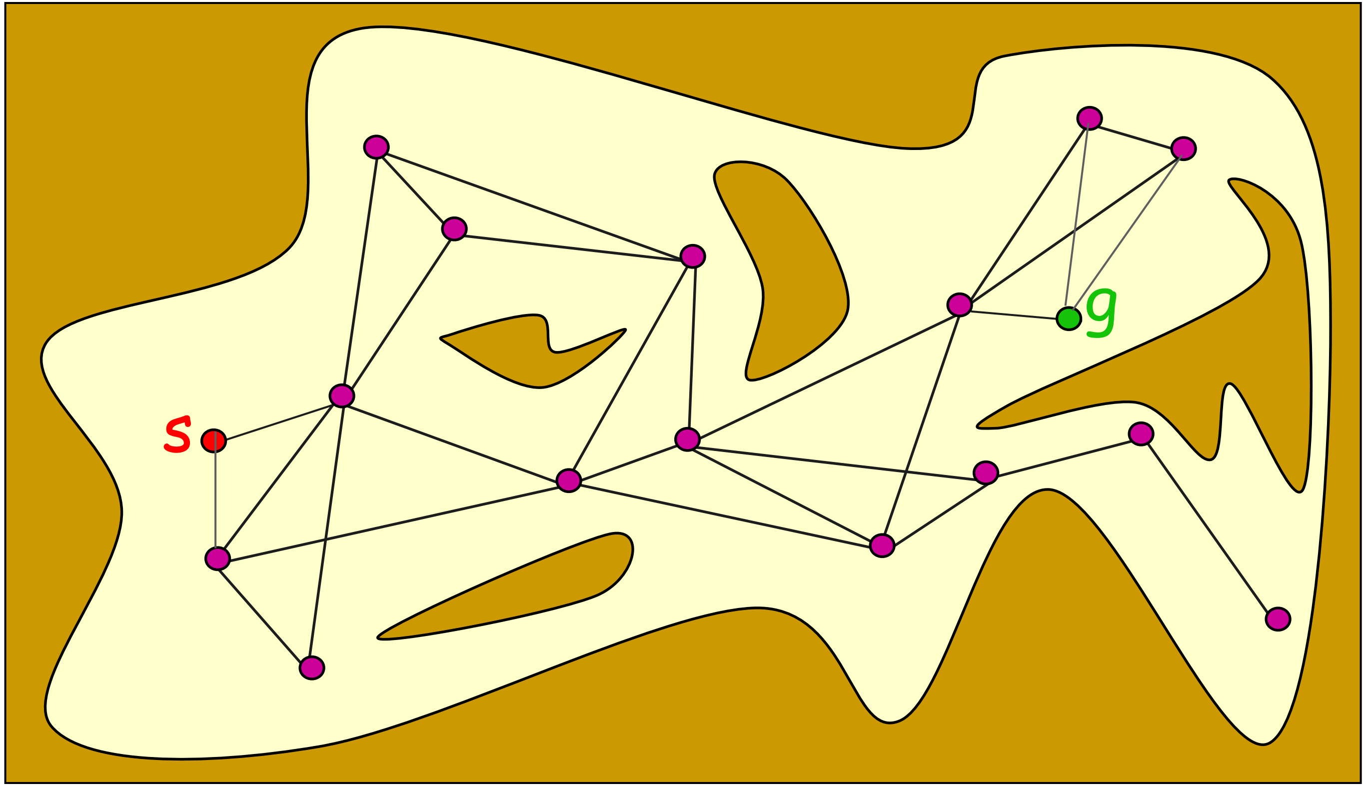
Probabilistic Roadmap (PRM)

The collision-free links are retained as local paths to form the PRM



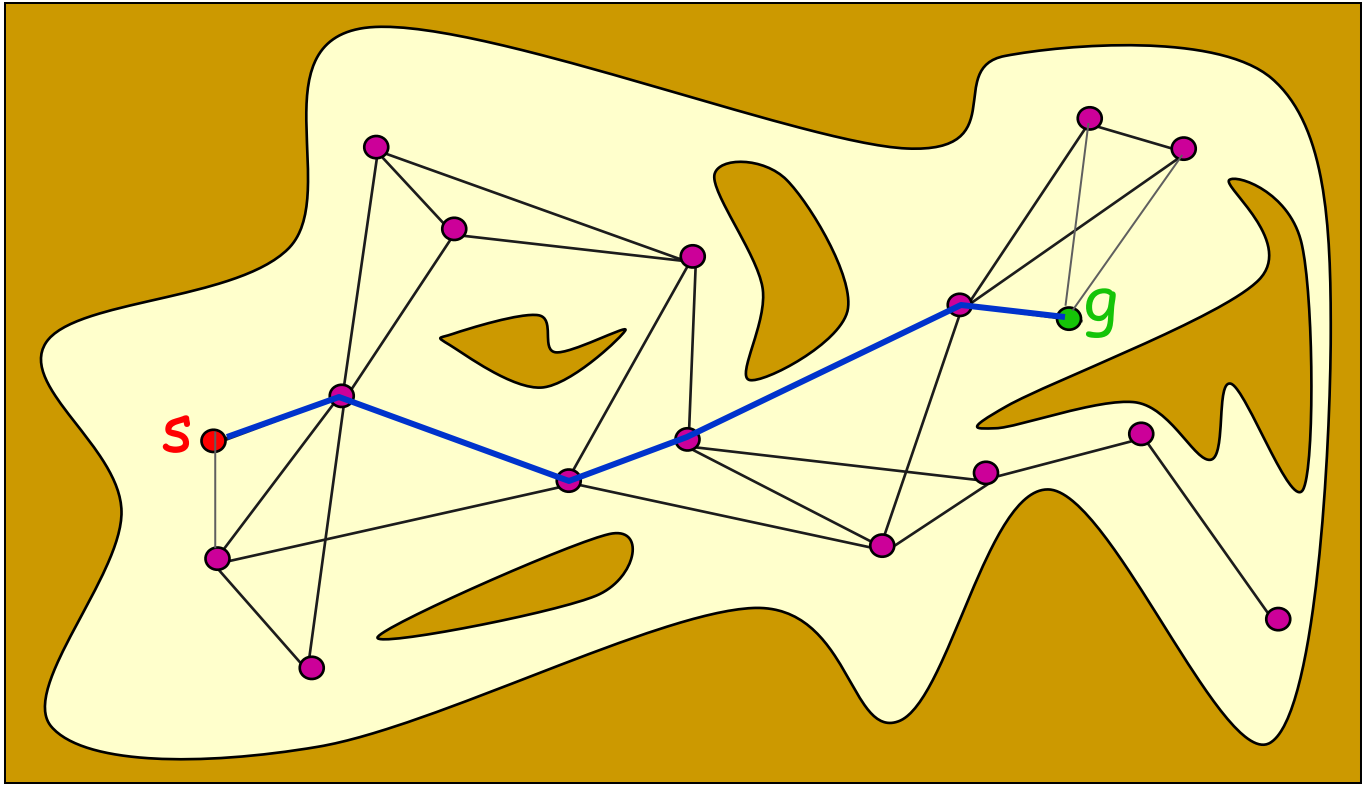
Probabilistic Roadmap (PRM)

The start and goal configurations are included as milestones



Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g



Probabilistic Roadmap (PRM)

for $i = 1, \ldots, N$ **do**
  $q_i \leftarrow$ sample from $\mathcal{C}_{free}$
  add q_i to Roadmap R
end for
for $i = 1, \ldots, N$ **do**
  $\mathcal{N}(q_i) \leftarrow k$ closest neighbors of q_i
  for each $q \in \mathcal{N}(q_i)$ **do**
    if there is a collision free local path from q to q_i and there is not already an edge from q to q_i **then**
      add an edge from q to q_i to the Roadmap R
    end if
  end for
end for
return R

Probabilistic Roadmap (PRM)

for $i = 1, \ldots, N$ **do**
    $q_i \leftarrow$ sample from $\mathcal{C}_{free}$
    add q_i to Roadmap R
end for
for $i = 1, \ldots, N$ **do**
    $\mathcal{N}(q_i) \leftarrow k$ closest neighbors of q_i
    for each $q \in \mathcal{N}(q_i)$ **do**
        if there is a collision free local path from q to q_i and there is not already an edge from q to q_i **then**
            add an edge from q to q_i to the Roadmap R
        end if
    end for
end for
return R

The resulting R depends on:

- N - number of samples
- k - number of neighbors
- Sampler
- Local path planner

Probabilistic Roadmap (PRM)

```
for i = 1, ..., N do
    q_i ← sample from C_free
    add q_i to Roadmap R
end for
for i = 1, ..., N do
    N(q_i) ← k closest neighbors of q_i
    for each q ∈ N(q_i) do
        if there is a collision free local path from q to q_i and there is not already
        an edge from q to q_i then
            add an edge from q to q_i to the Roadmap R
        end if
    end for
end for
return R
```

The resulting R depends on:

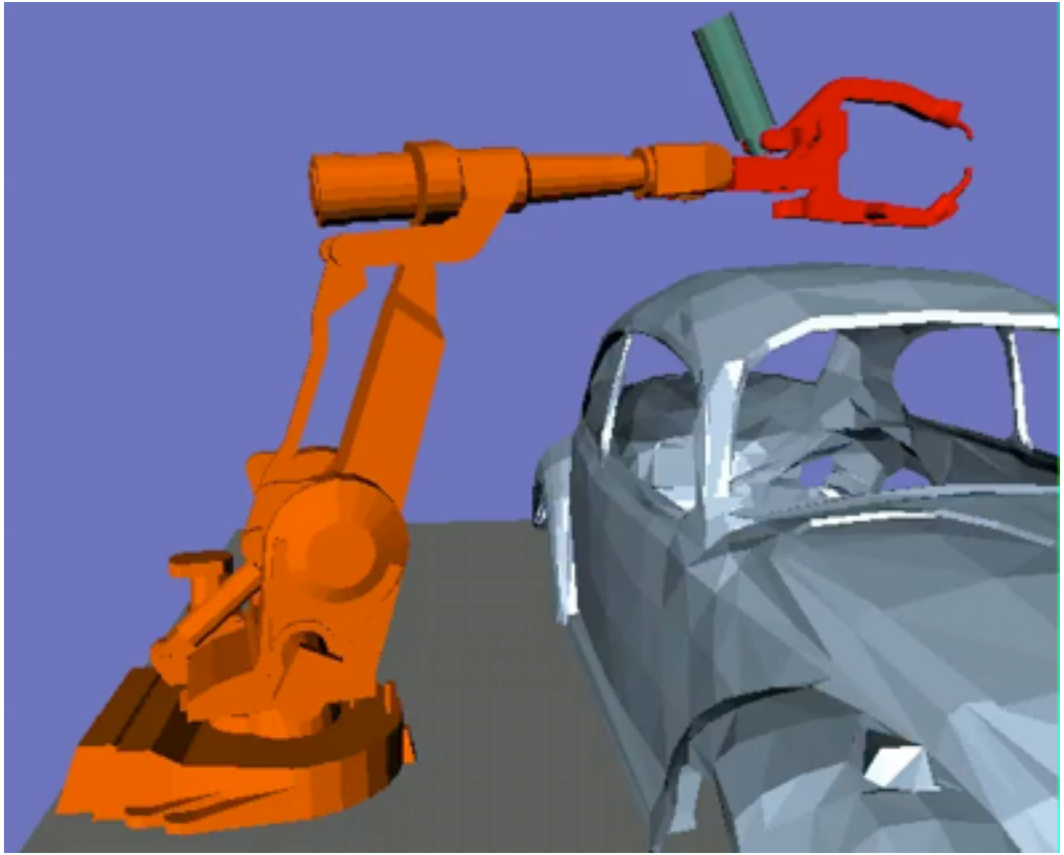
- N - number of samples
- k - number of neighbors
- Sampler
- Local path planner

PRM is a multiple-query planner: invest time in generating a good representation of the free C-space, that can be used to solve several motion planning problems.

Probabilistic Roadmap (PRM)

Demonstration - https://demonstrations.wolfram.com/ProbabilisticRoadmapMethodForRobotArm/

PRM Example



PRM’s Pros and Cons

- Pro:
  - Probabilistically complete: i.e., with probability one, if run for long enough the graph will contain a solution path if one exists.
- Cons:
  - Build graph over state space but no focus on generating a path

Rapidly exploring Random Tree (RRT)

Steve LaValle (98)

- Basic idea:
  - Build up a tree through generating “next states” in the tree by executing random controls
  - However: not exactly to ensure good coverage

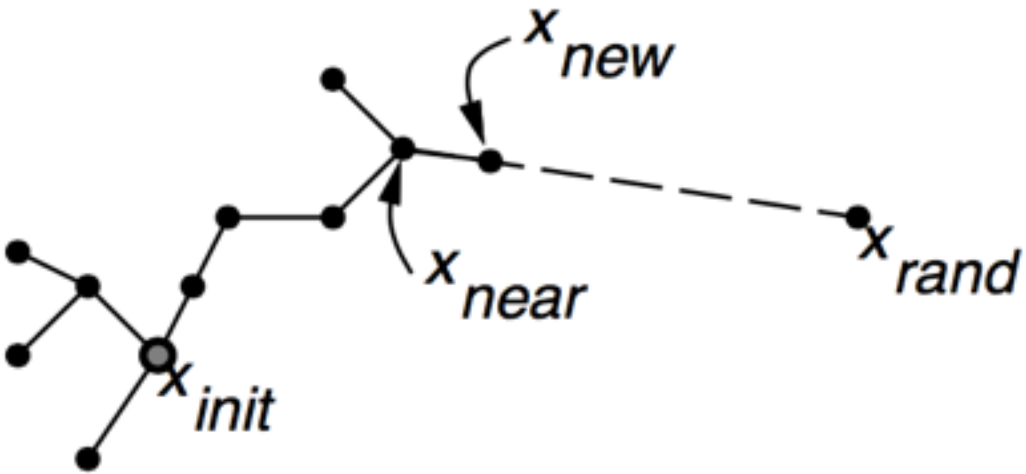
Demonstration - https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

Rapidly exploring Random Tree (RRT)

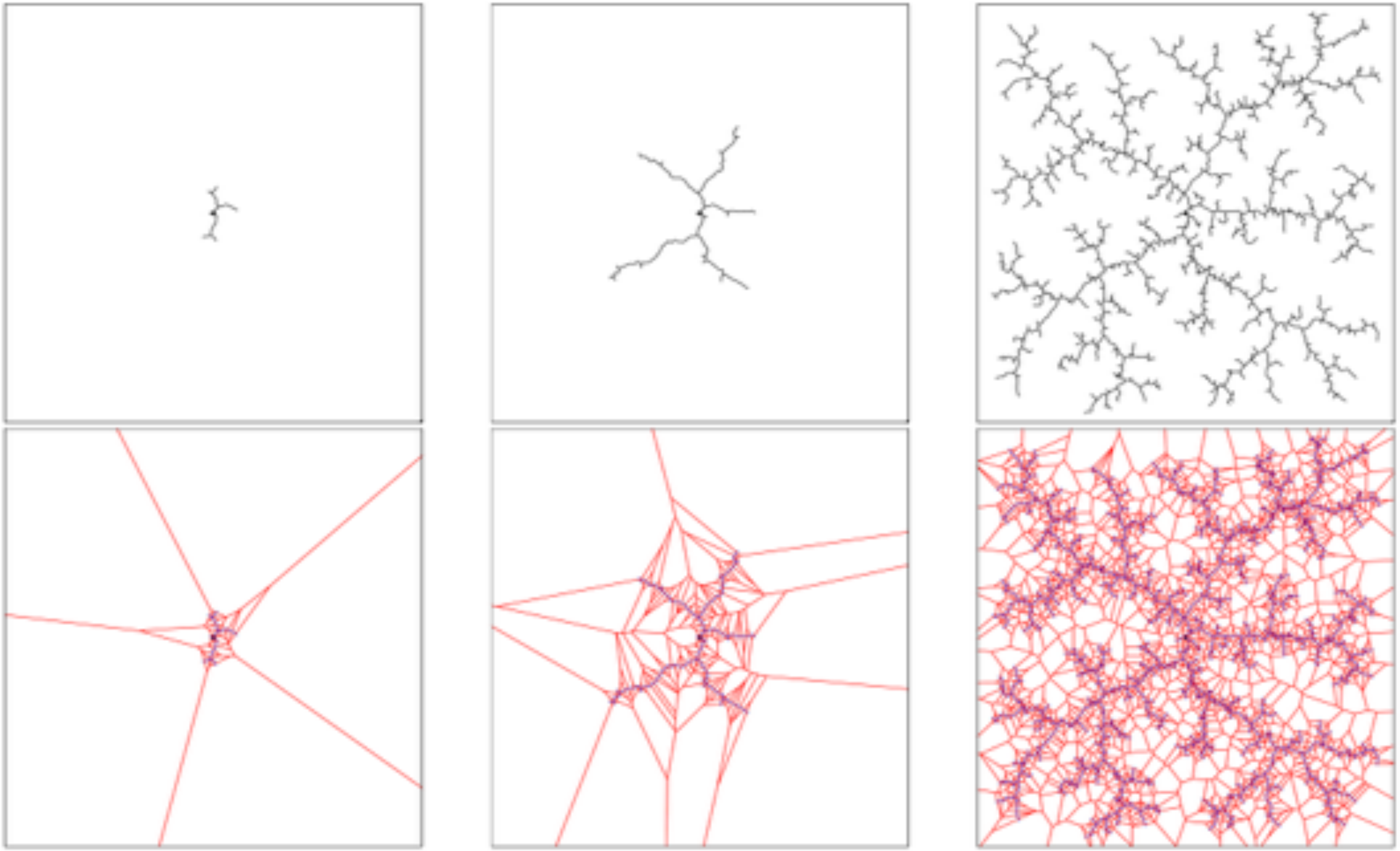
- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region

Rapidly exploring Random Tree (RRT)

- Select random point, and expand nearest vertex towards it
  - Biases samples towards largest Voronoi region



Rapidly exploring Random Tree (RRT)

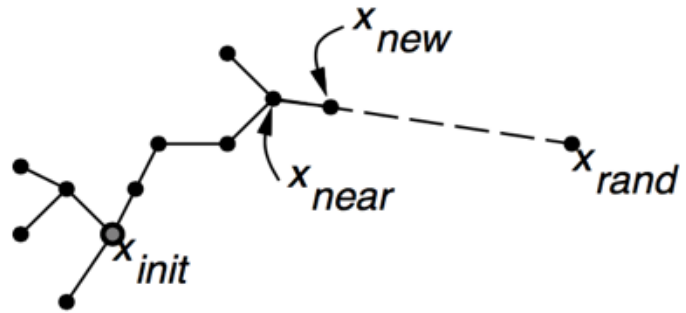


Source: LaValle and Kuffner 01

Rapidly exploring Random Tree (RRT)

GENERATE_RRT($x_{init}, K, \Delta t$)

$\mathcal{T}$.init(x_{init});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
    $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
    $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
    $\mathcal{T}$.add_vertex(x_{new});
    $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
Return $\mathcal{T}$



RANDOM_STATE(): often uniformly at random over space with probability 99%, and the goal state with probability 1%, this ensures it attempts to connect to goal semi-regularly

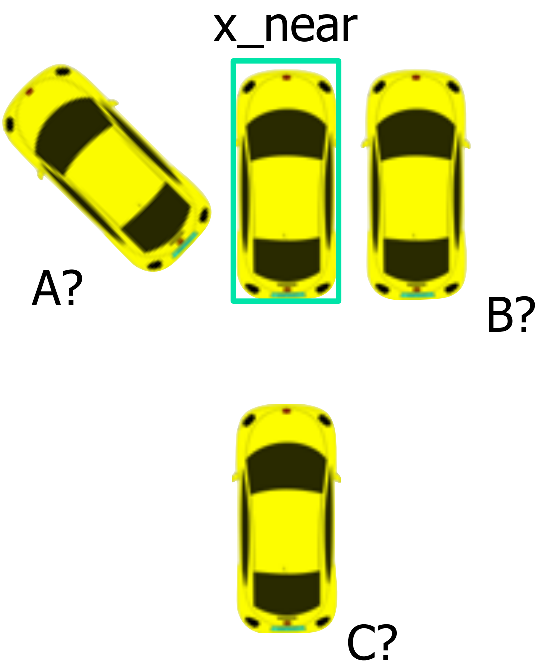
RRT Practicalities

- NEAREST_NEIGHBOR(x_{rand}, T): need to find (approximate) nearest neighbor efficiently
  - KD Trees data structure (upto 20-D) [e.g., FLANN]
  - Locality Sensitive Hashing

- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

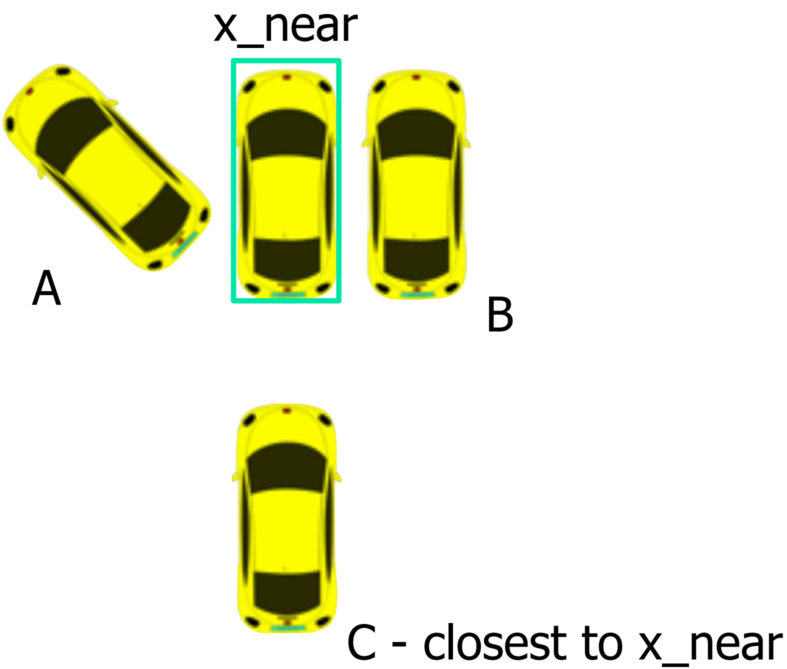
RRT Extension

- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem



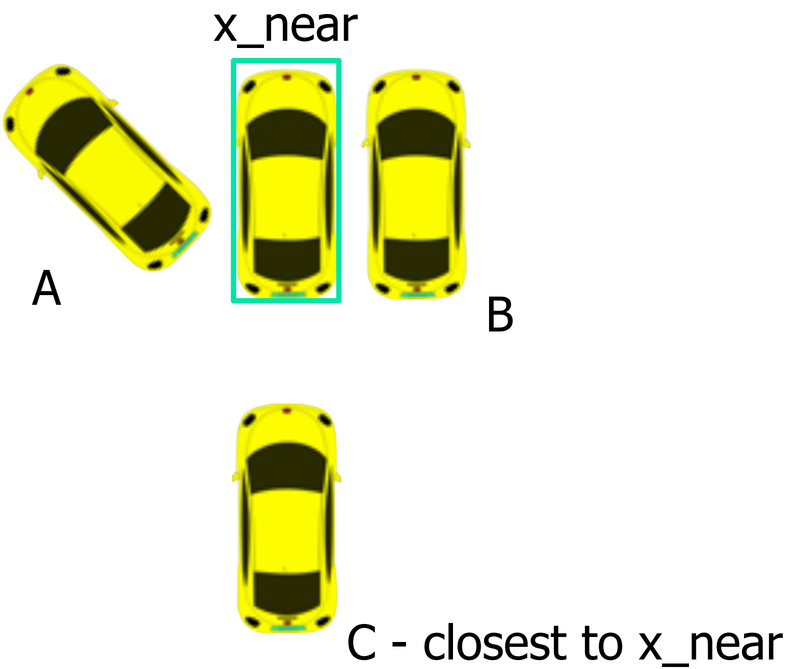
RRT Extension

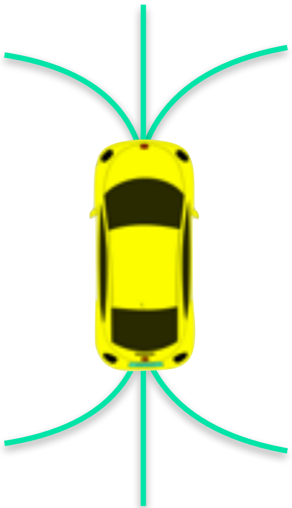
- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem



RRT Extension

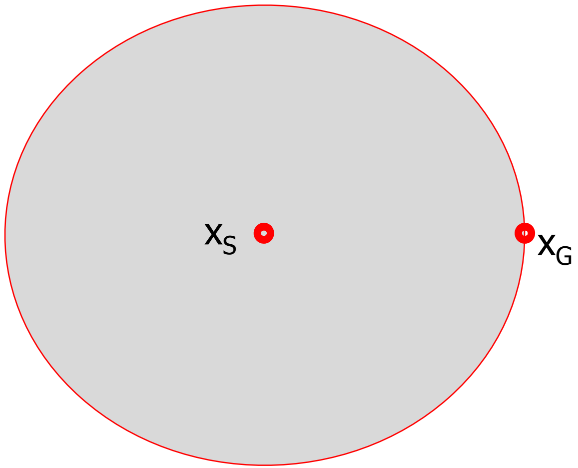
- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem



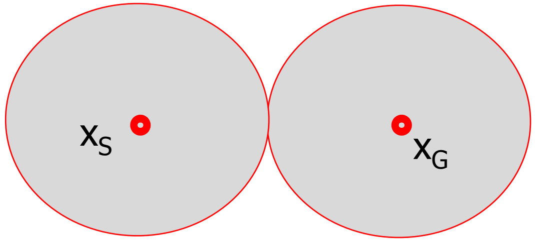


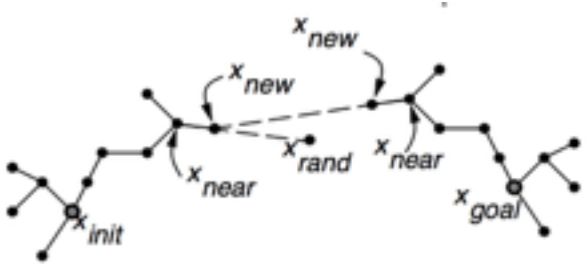
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:

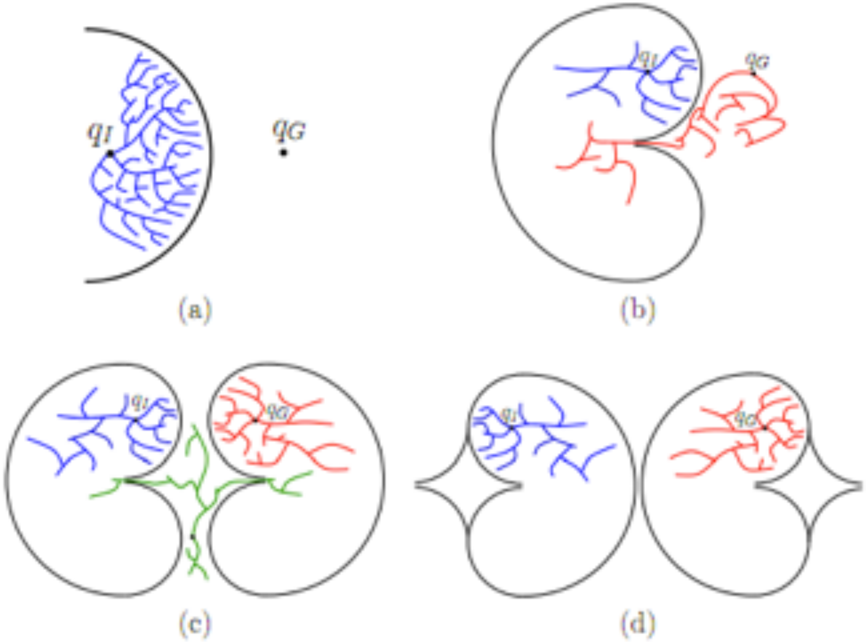




- Difference more and more pronounced as dimensionality increases

Multi-directional RRT

- Planning around obstacles or through narrow passages can often be easier in one direction than the other



RRT*

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$

for $i = 1, \ldots, n$ **do**

$\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$

$\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$

$\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$

FIND x_new

$\quad$ **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**

$\quad\quad X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})\ ;$

$\quad\quad V \leftarrow V \cup \{x_{\text{new}}\};$

ADD x_new to G
FIND neighbors to x_new in the G

$\quad\quad x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$

$\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path

$\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**

$\quad\quad\quad\quad x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$

$\quad\quad E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$

FIND edge to x_new from neighbors with least cost
ADD that to G

$\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree

$\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$

$\quad\quad\quad$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$

$\quad\quad\quad\quad E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

REWIRE the edges in the neighborhood if any least cost path exists from the root to the neighbors via x_new

return $G = (V, E);$

Source: Karaman and Frazzoli

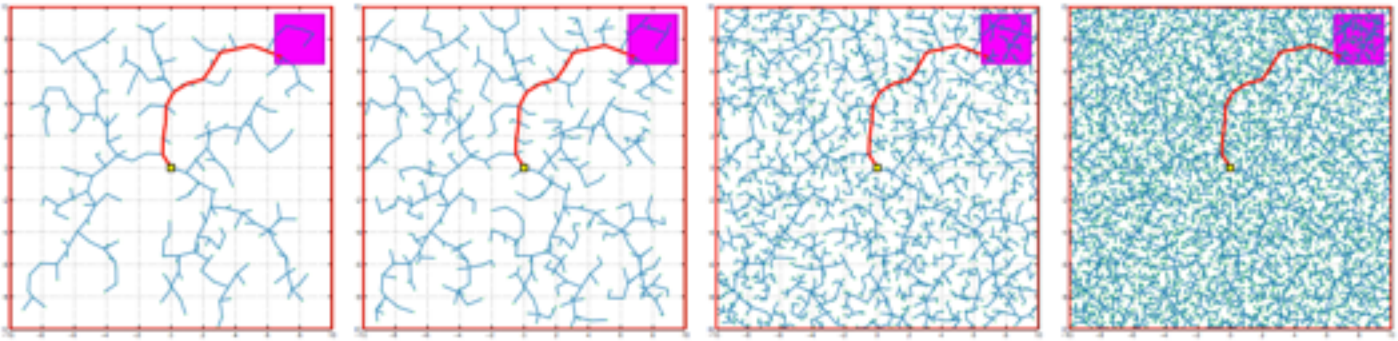
RRT*

- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

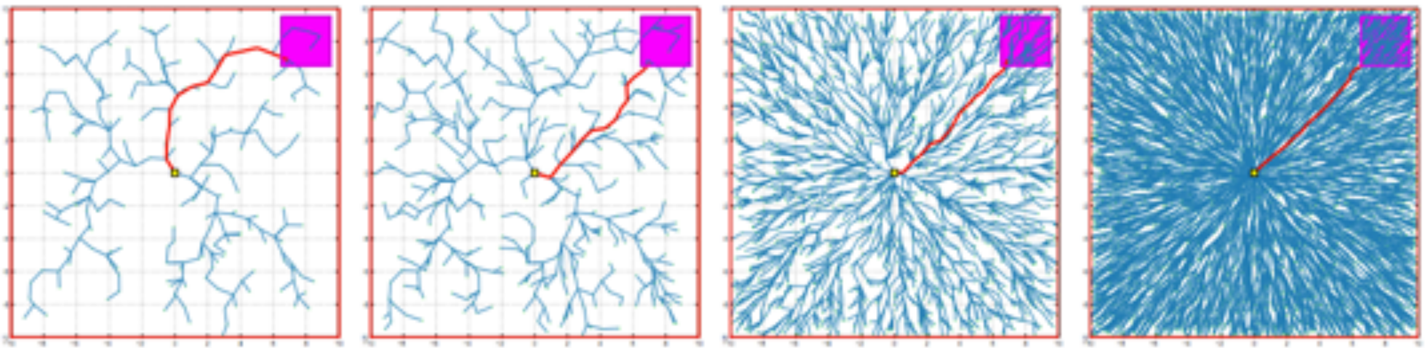
Demonstration - https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

RRT*

RRT



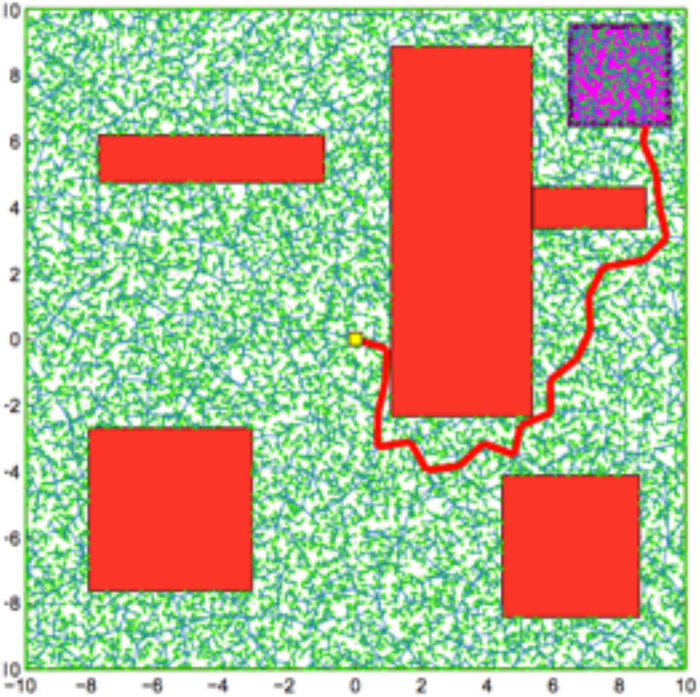
RRT*



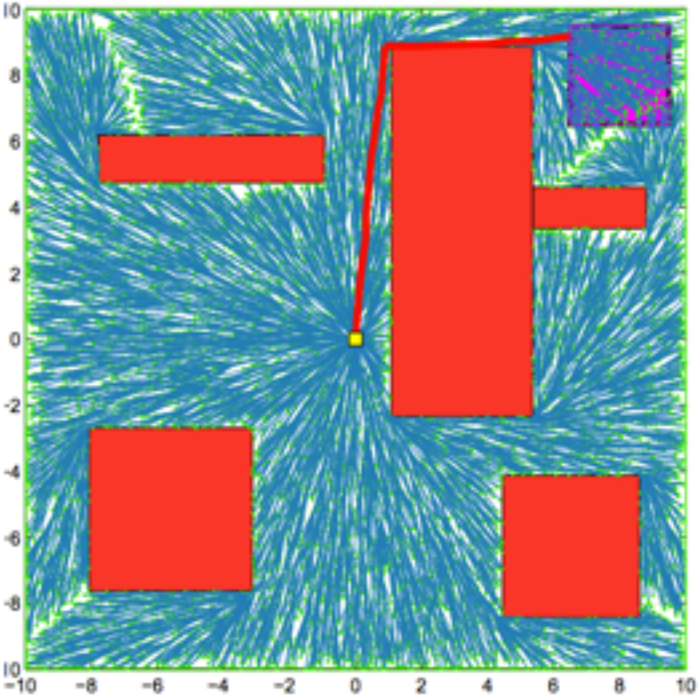
Source: Karaman and Frazzoli

RRT*

RRT



RRT*



Source: Karaman and Frazzoli

Smoothing

Randomized motion planners tend to find not so great paths for execution: very jagged, often much longer than necessary.

→ In practice: do smoothing before using the path

- Shortcutting:
  - along the found path, pick two vertices x_{t1}, x_{t2} and try to connect them directly (skipping over all intermediate vertices)
- Nonlinear optimization for optimal control
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

Additional Resources

- Marco Pavone (http://asl.stanford.edu/):
  - Sampling-based motion planning on GPUs: https://arxiv.org/pdf/1705.02403.pdf
  - Learning sampling distributions: https://arxiv.org/pdf/1709.05448.pdf
- Sidd Srinivasa (https://personalrobotics.cs.washington.edu/)
  - Batch informed trees: https://robotic-esp.com/code/bitstar/
  - Expensive edge evals: https://arxiv.org/pdf/2002.11853.pdf
  - Lazy search: https://personalrobotics.cs.washington.edu/publications/mandalika2019gls.pdf
- Michael Yip (https://www.ucsdarclab.com/)
  - Neural Motion Planners: https://www.ucsdarclab.com/neuralplanning
- Lydia Kavraki (http://www.kavrakilab.org/)
  - Motion in human workspaces: http://www.kavrakilab.org/nsf-nri-1317849.html