CSE-571

Sampling-Based Motion Planning

Slides based on those from Pieter Abbeel, Zoe McCarthy

Many images from Lavalle, Planning Algorithms

Solve by Nonlinear Optimization for Control?

- Could try by, for example, following formulation:

$$\begin{aligned} \min_{u,x} \quad & (x_T - x_G)^\top (x_T - x_G) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \end{aligned}$$

X$_t$ can encode obstacles

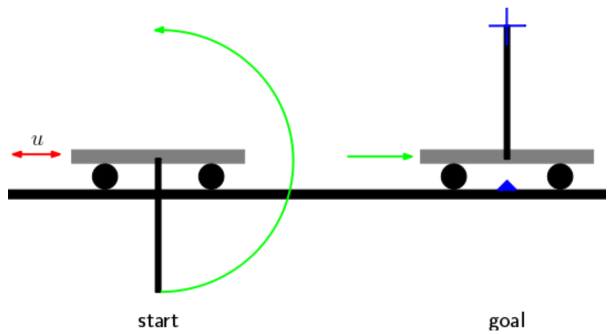
- Or, with constraints, (which would require using an infeasible method):

$$\begin{aligned} \min_{u,x} \quad & \|u\| \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \\ & X_T = x_G \end{aligned}$$

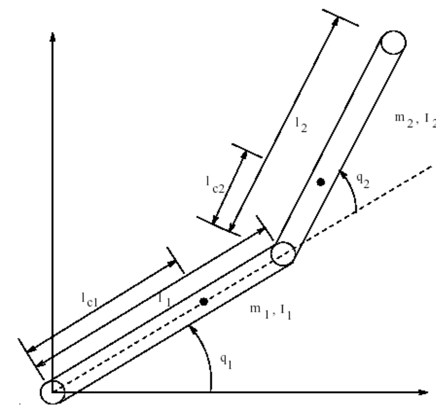
- *Can work surprisingly well, but for more complicated problems can get stuck in infeasible local minima*

Examples

Helicopter path planning

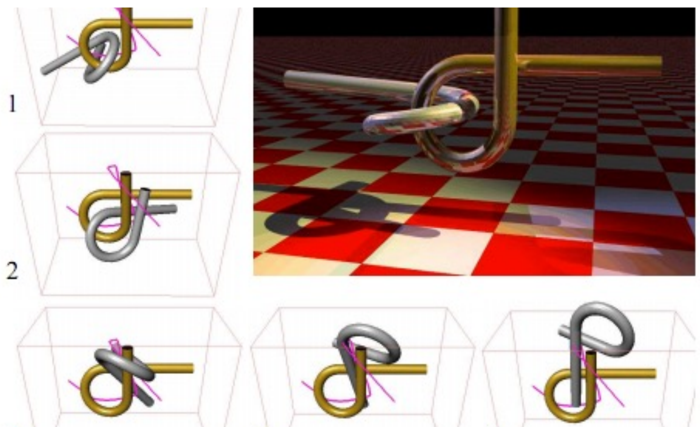


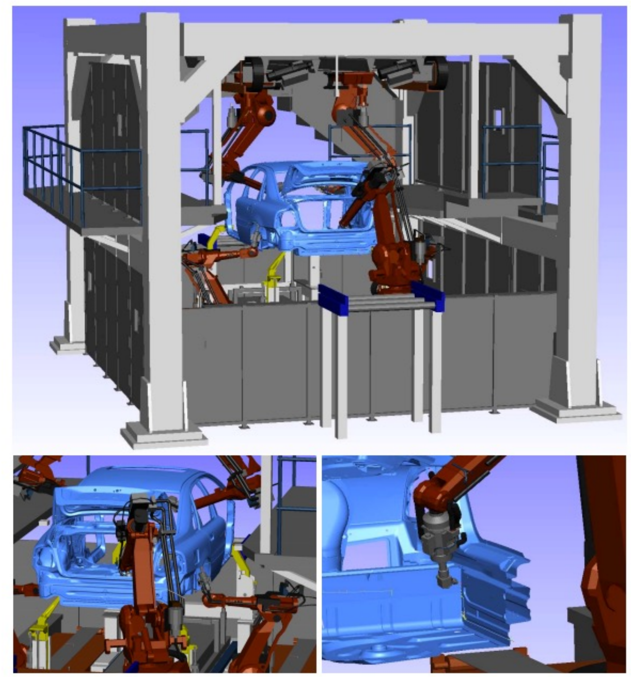
Cartpole swing-up



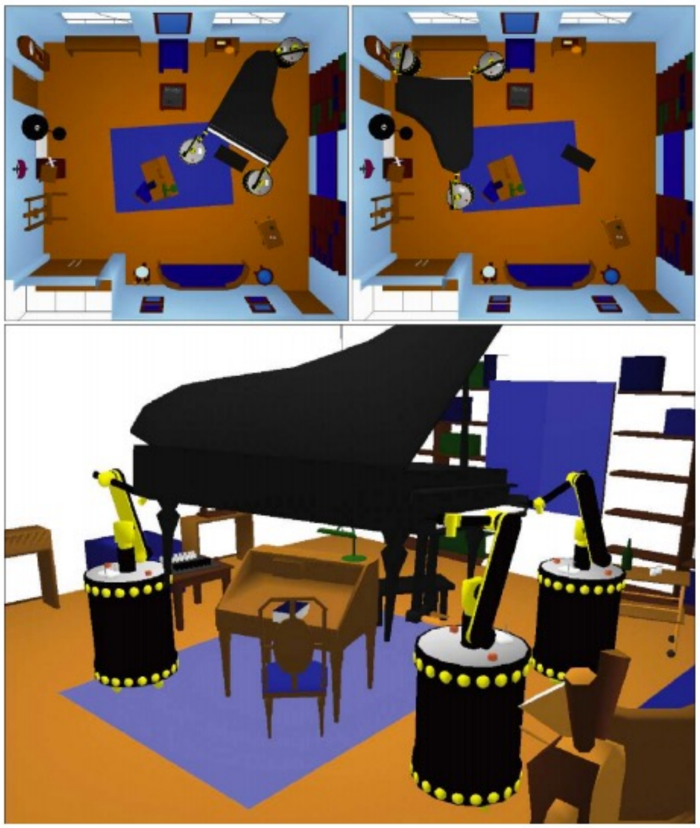
Acrobot

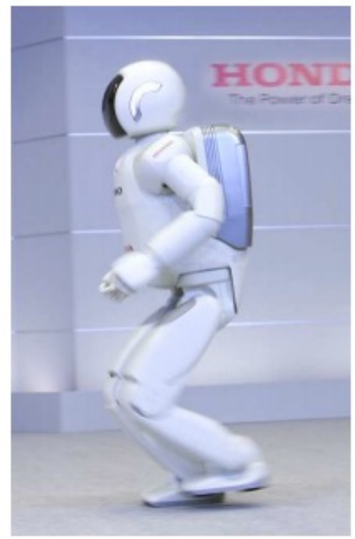
Examples

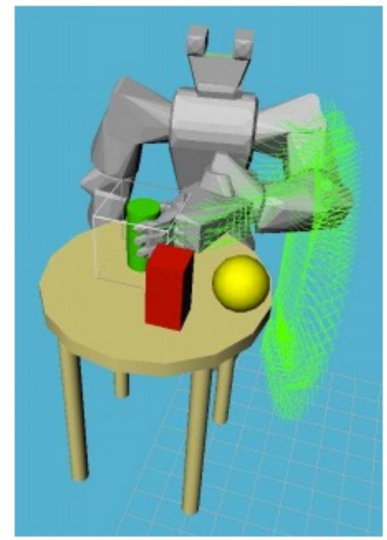




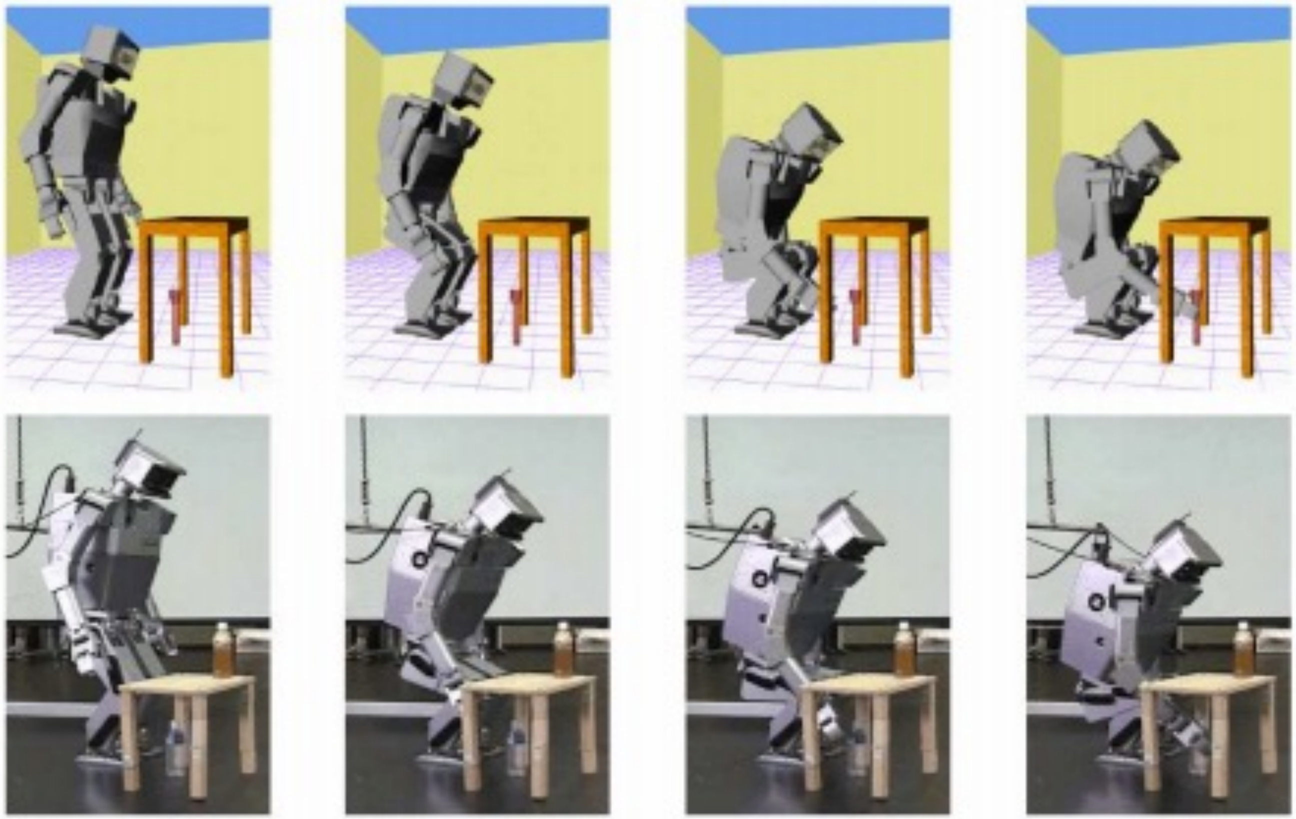
Examples







Examples



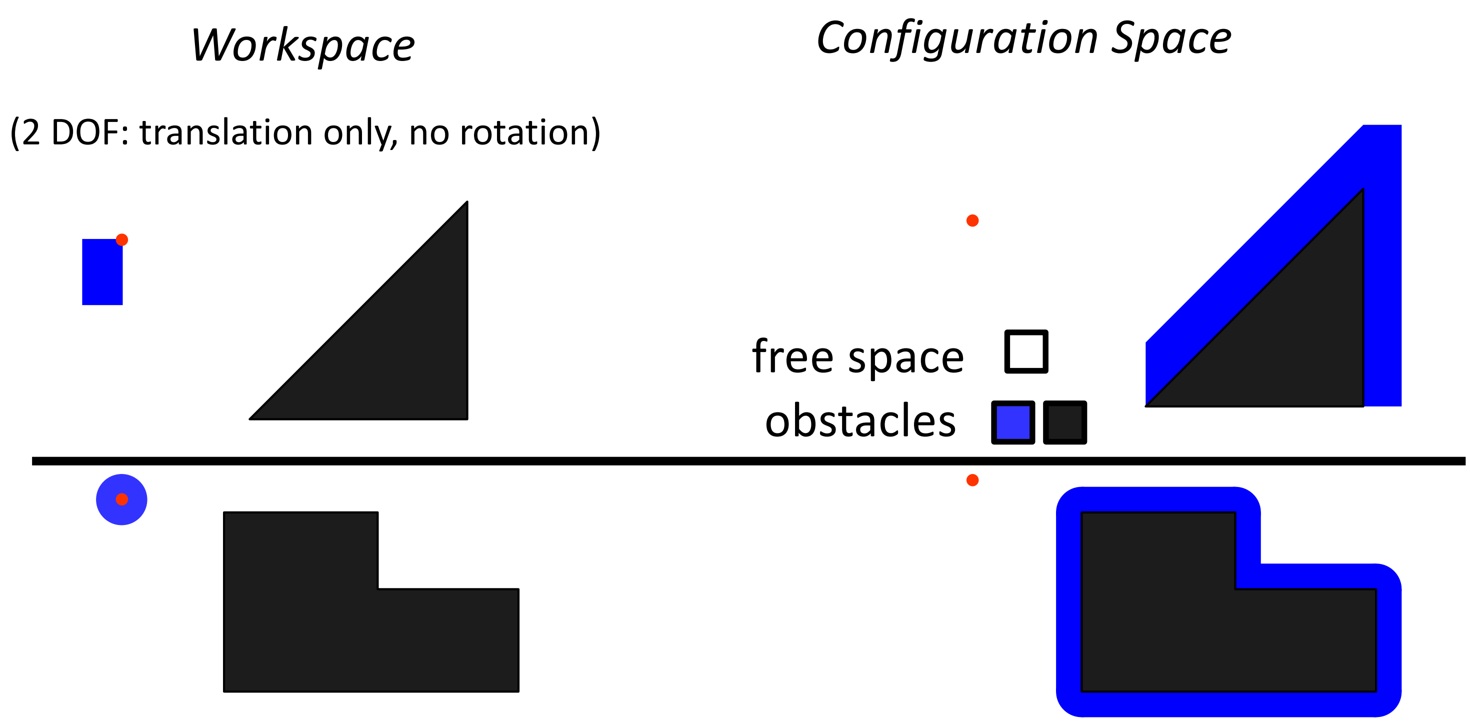
Motion Planning: Outline

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

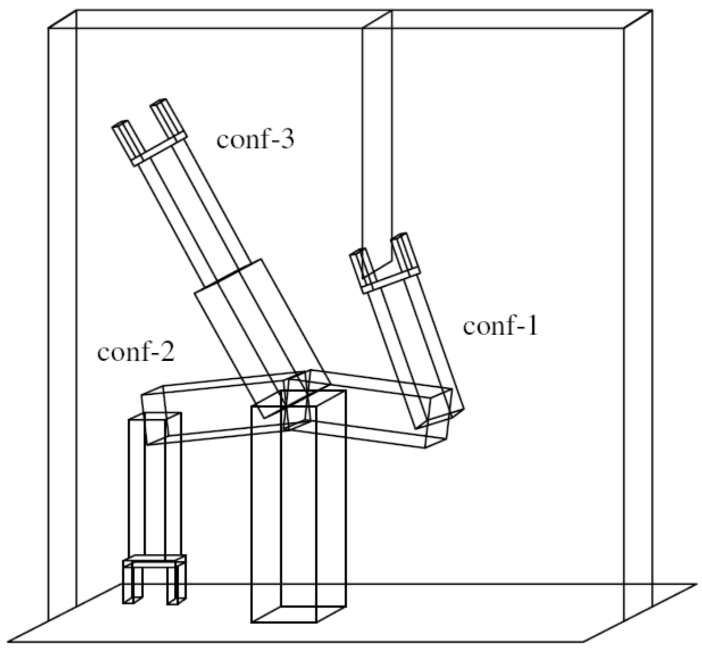
Configuration Space (C-Space)

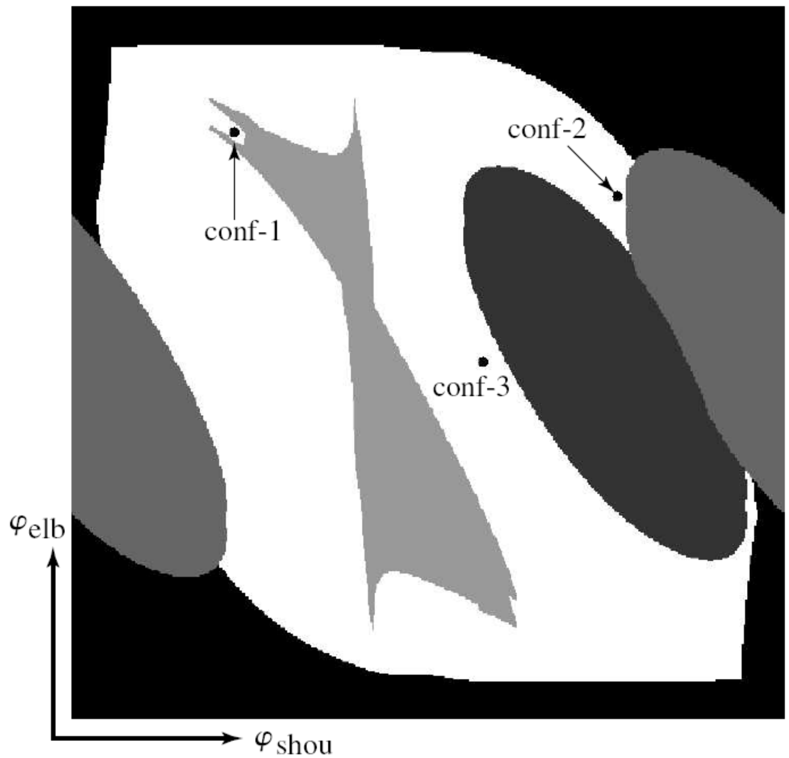
= { x | x is a pose of the robot}

- obstacles → configuration space obstacles

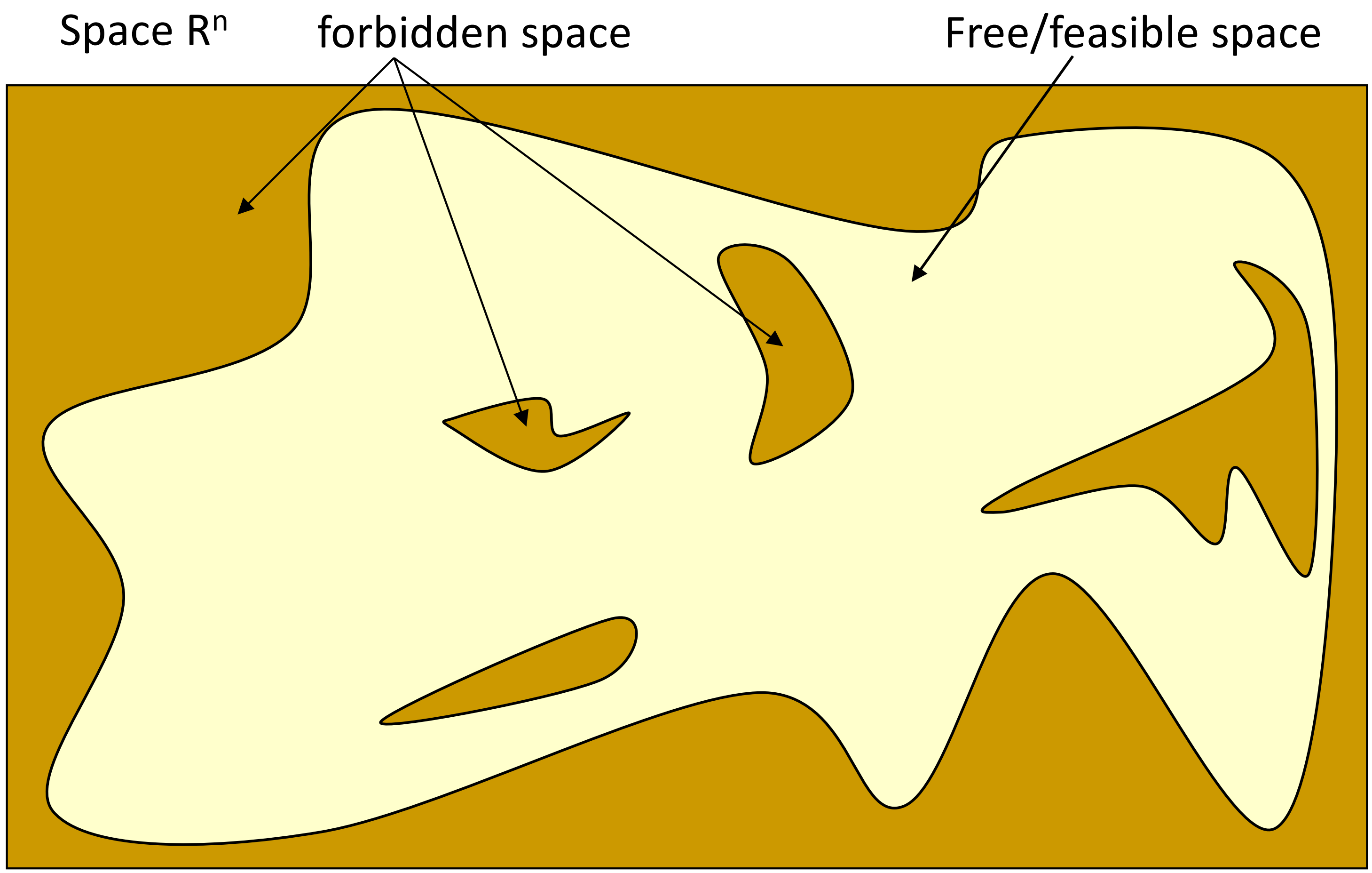


Motion planning



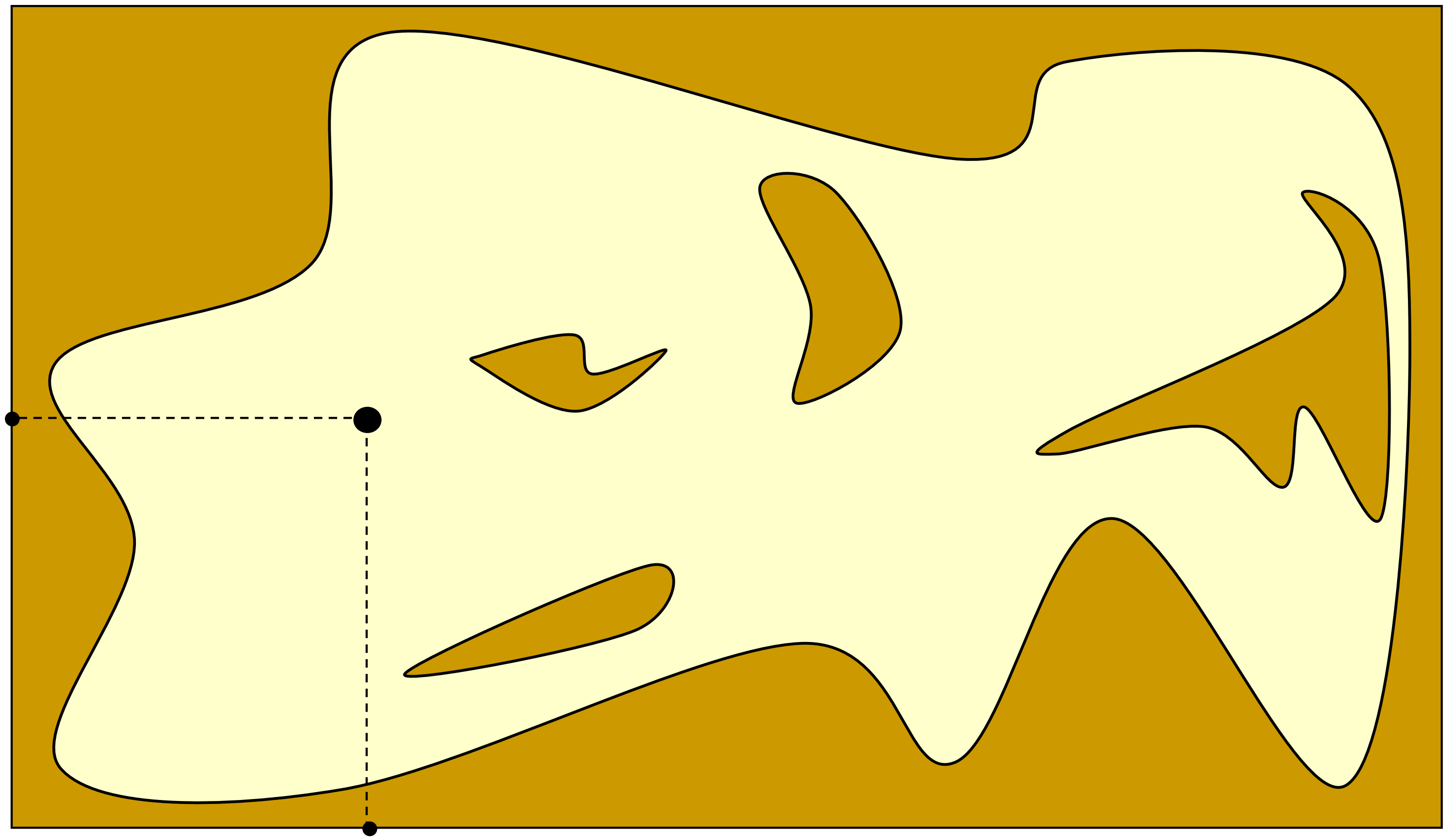


Probabilistic Roadmap (PRM)



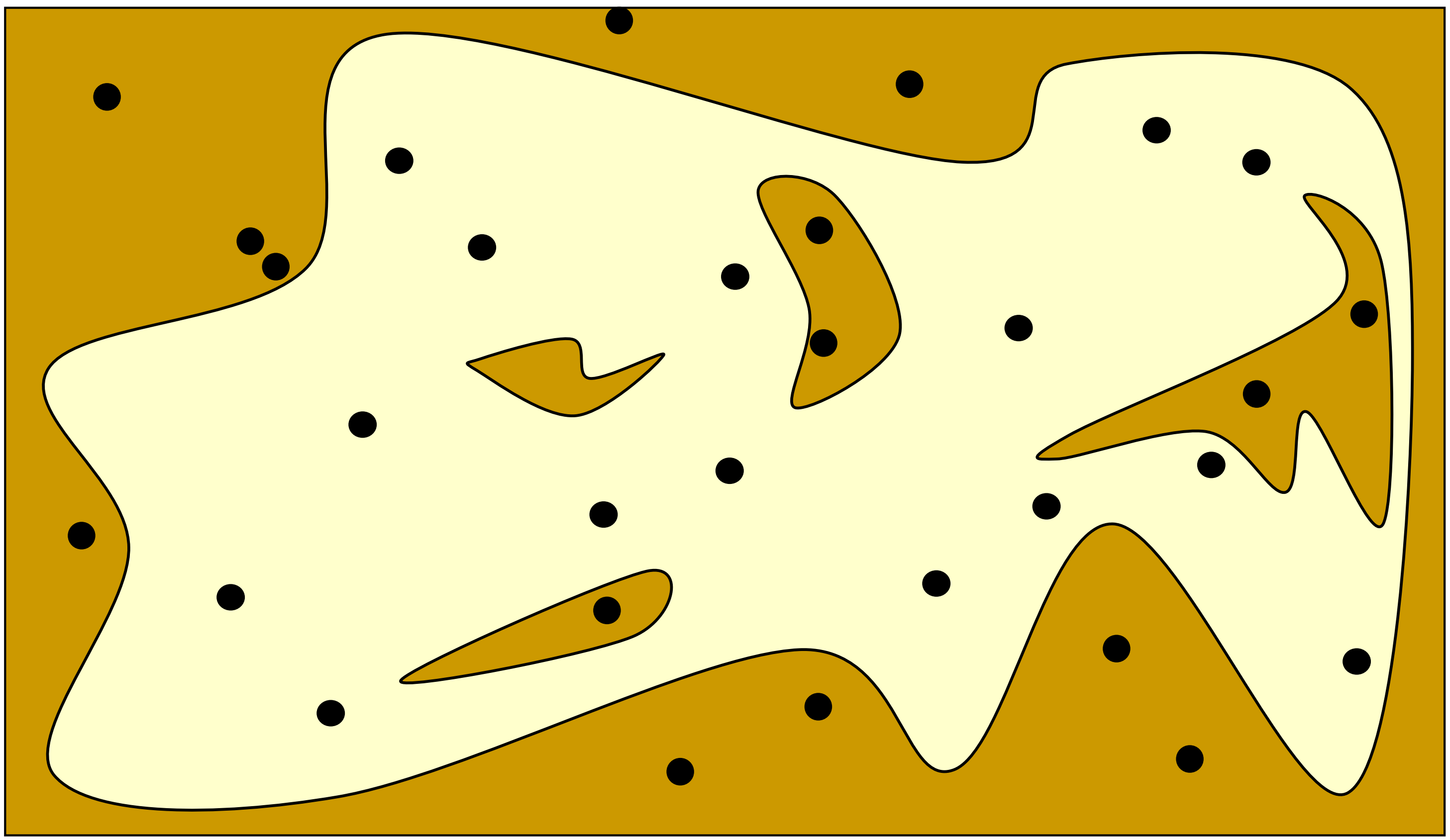
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



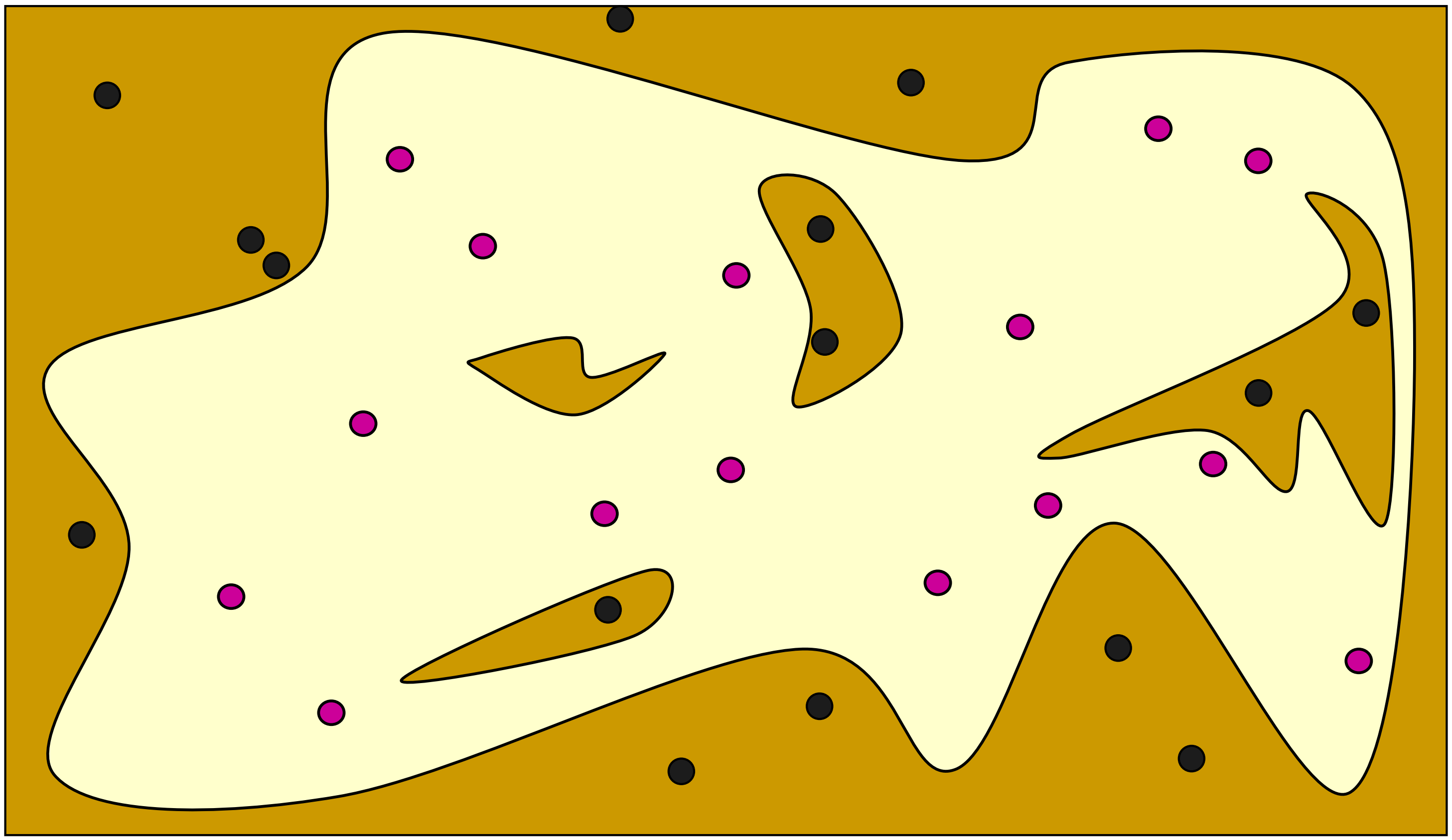
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



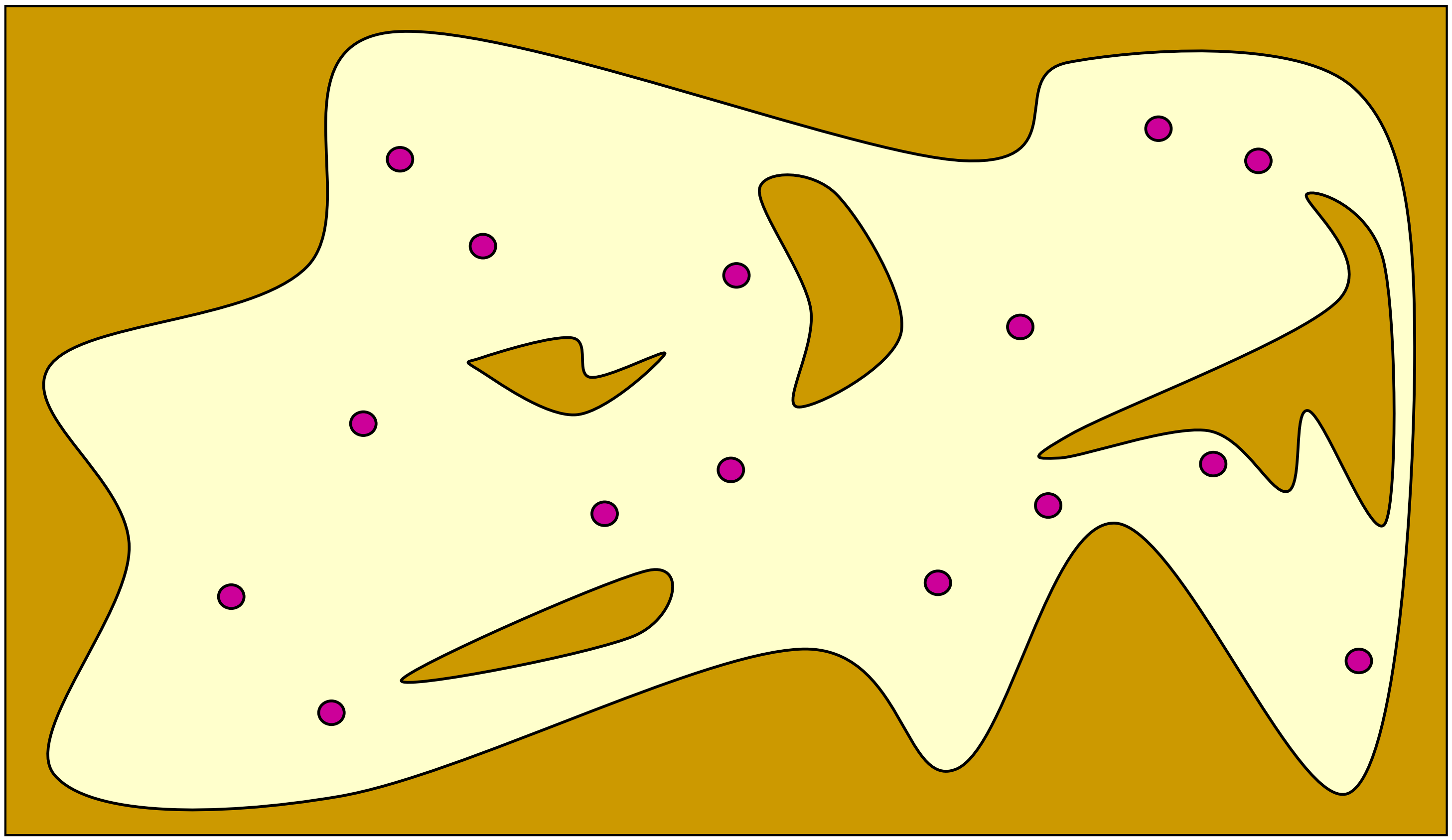
Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision



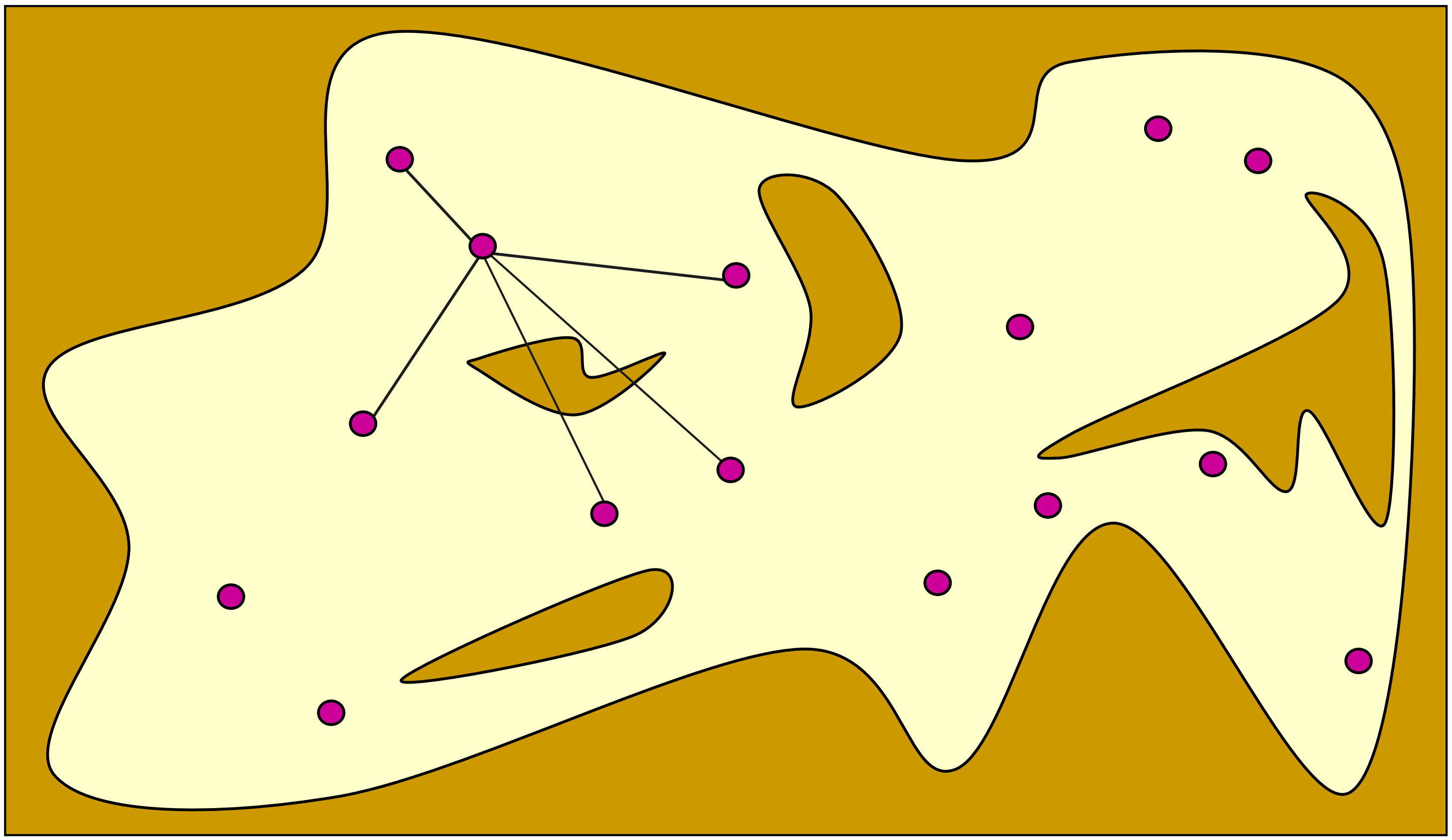
Probabilistic Roadmap (PRM)

The collision-free configurations are retained as milestones



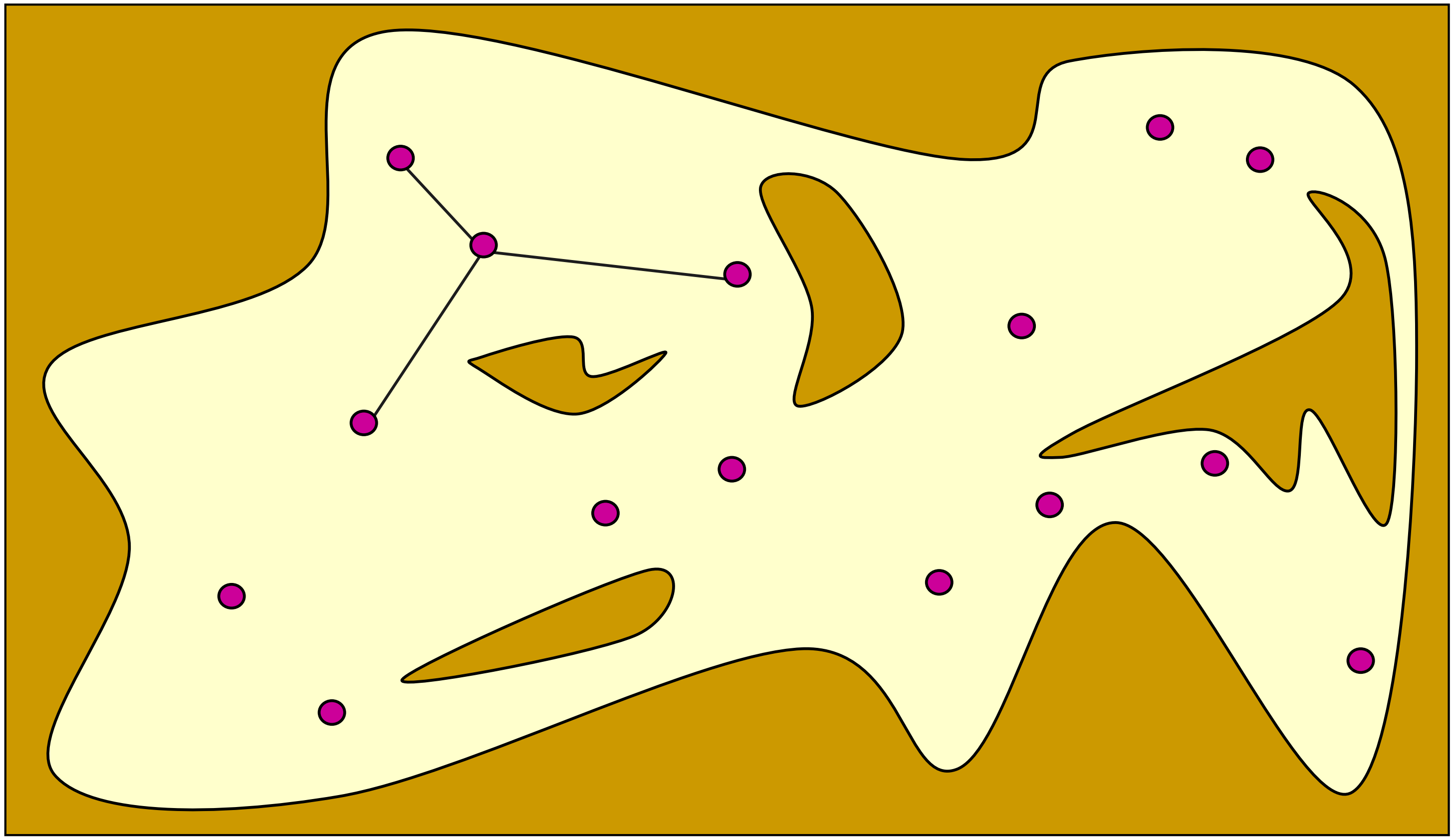
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



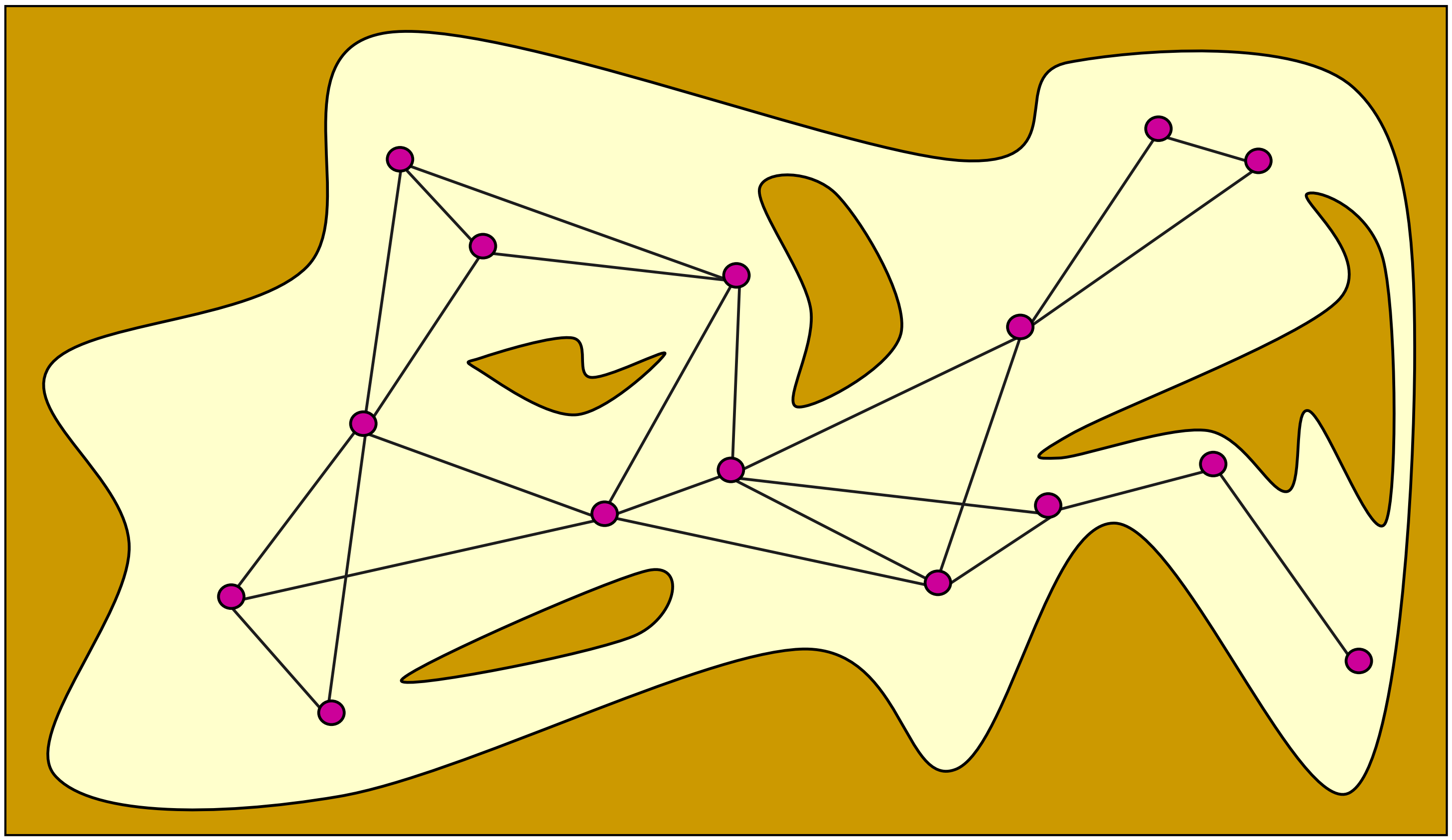
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



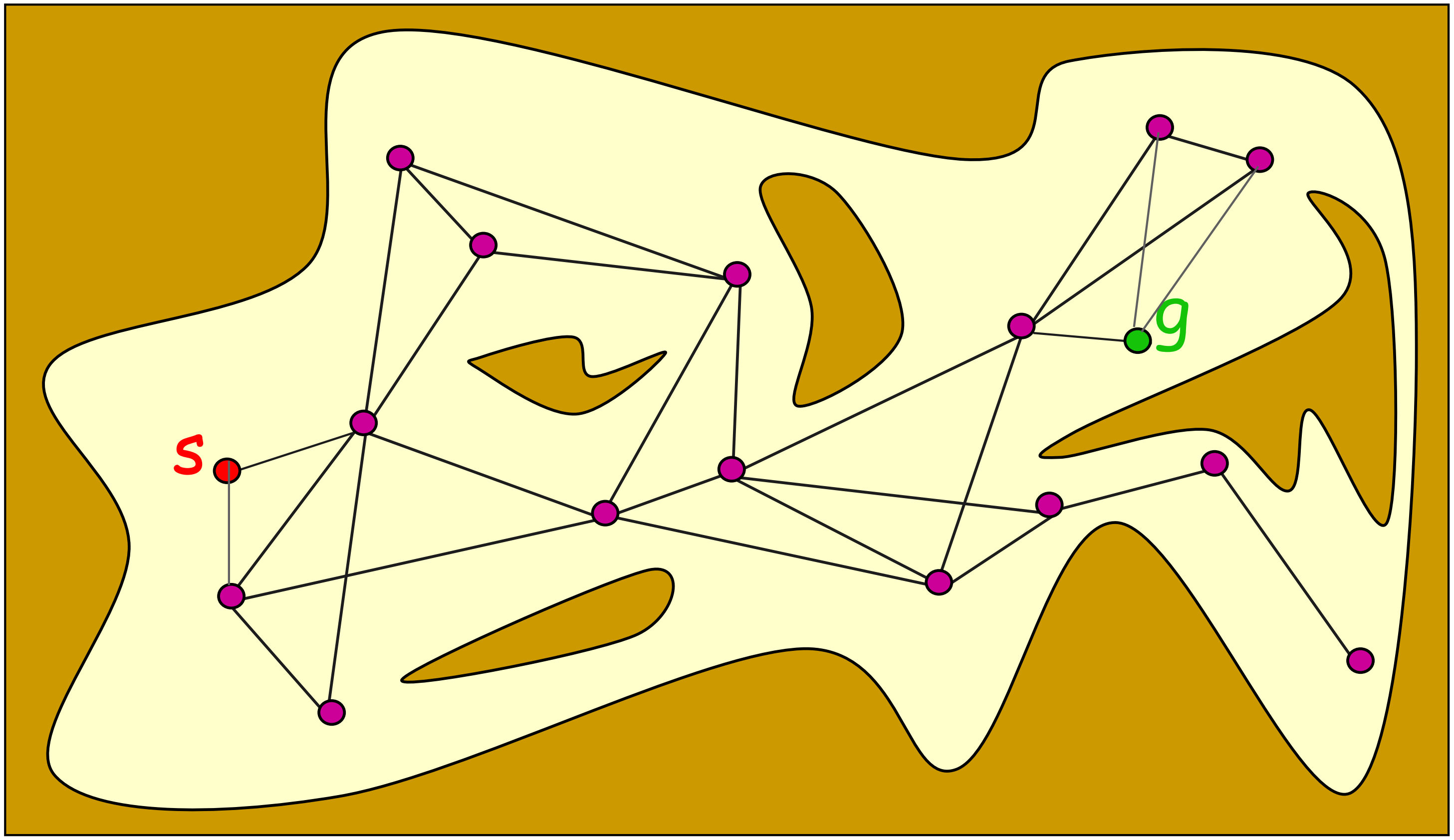
Probabilistic Roadmap (PRM)

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



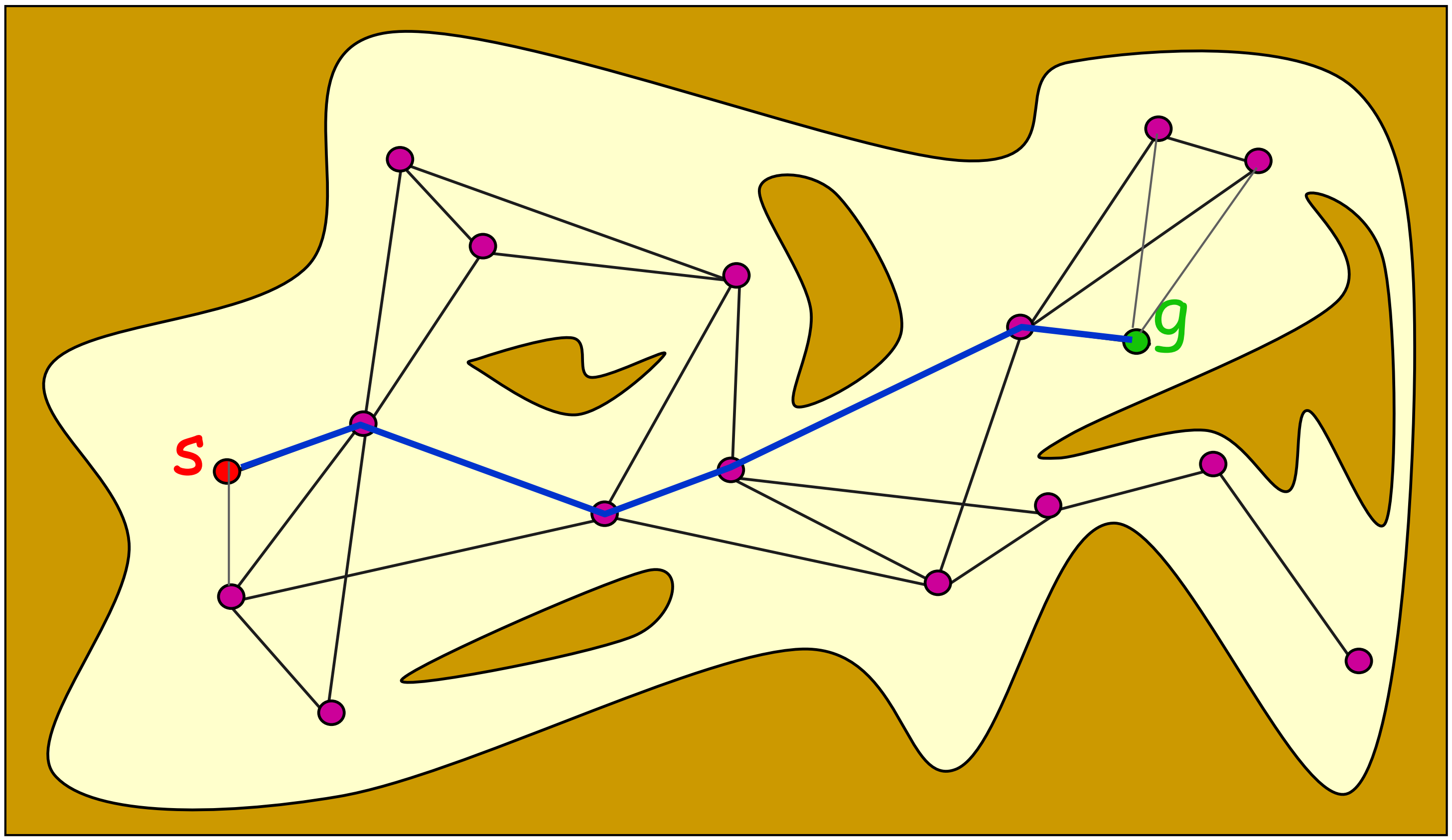
Probabilistic Roadmap (PRM)

The start and goal configurations are included as milestones



Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g

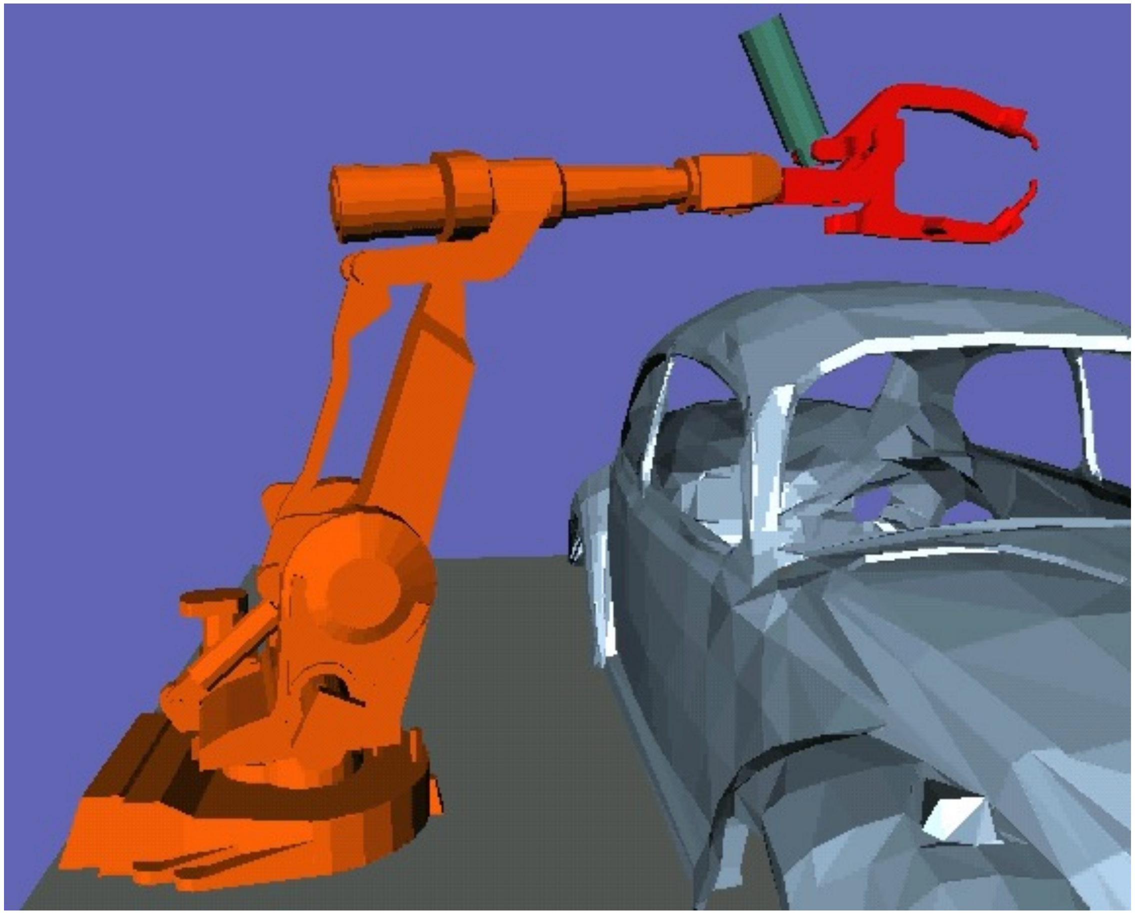


Probabilistic Roadmap

- Initialize set of points with x_S and x_G
- Randomly sample points in configuration space
- Connect nearby points if they can be reached from each other
- Find path from x_S to x_G in the graph

  - Alternatively: keep track of connected components incrementally, and declare success when x_S and x_G are in same connected component

PRM Example 1



PRM Example 2

PRM’s Pros and Cons

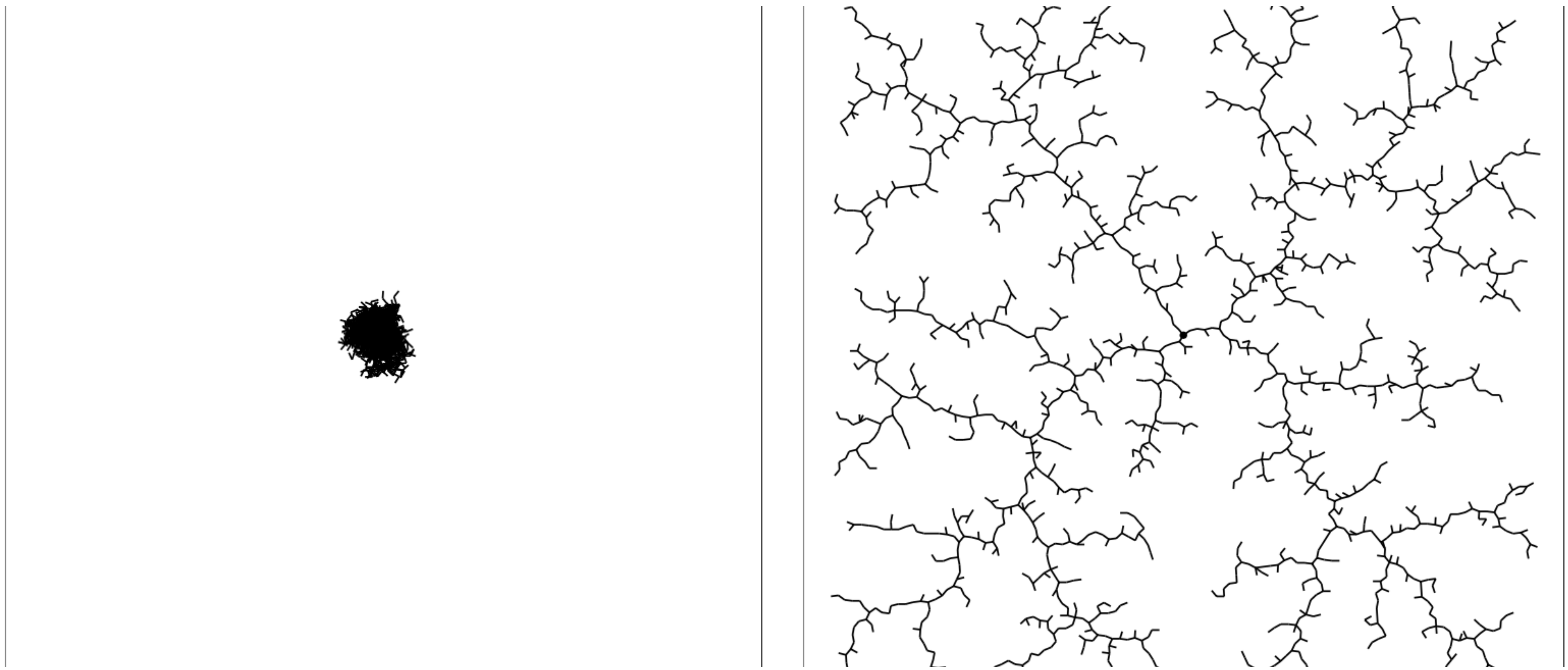
- Pro:
  - Probabilistically complete: i.e., with probability one, if run for long enough the graph will contain a solution path if one exists.
- Cons:
  - Required to solve 2-point boundary value problem
  - Build graph over state space but no focus on generating a path

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 above to ensure good coverage

How to Sample

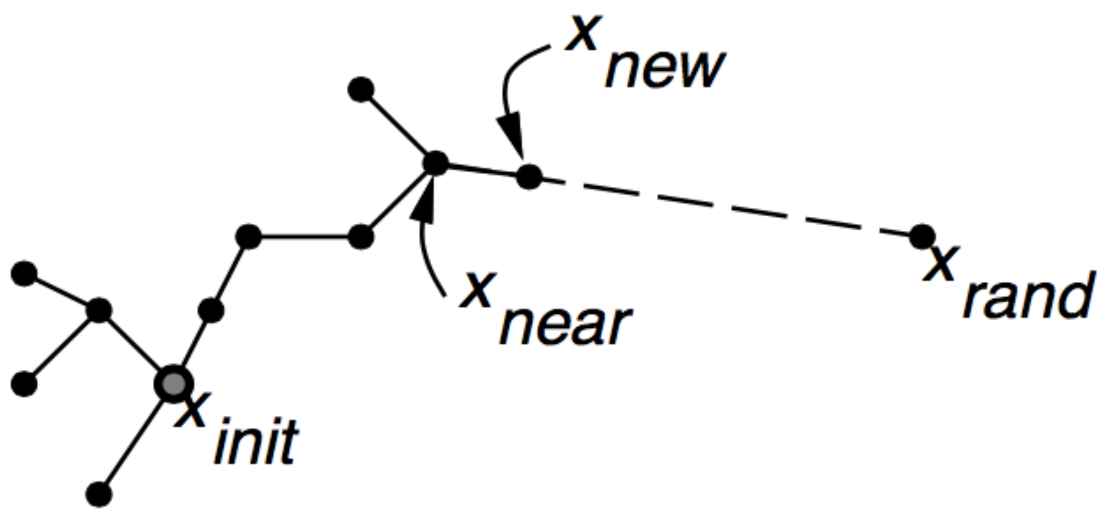


Rapidly exploring Random Tree (RRT)

- 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



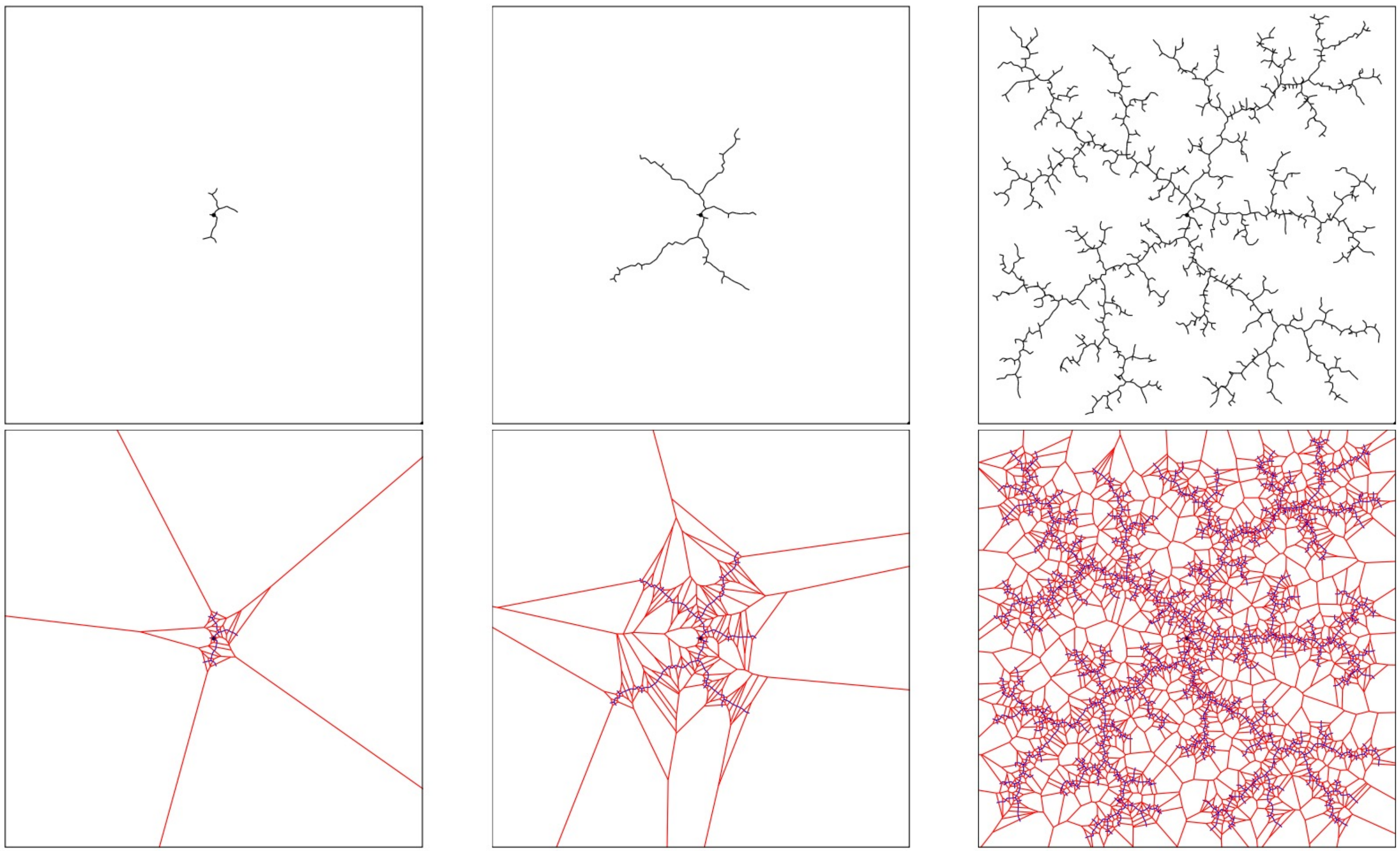
Rapidly exploring Random Tree (RRT)

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

Rapidly exploring Random Tree (RRT)



Source: LaValle and Kuffner 01

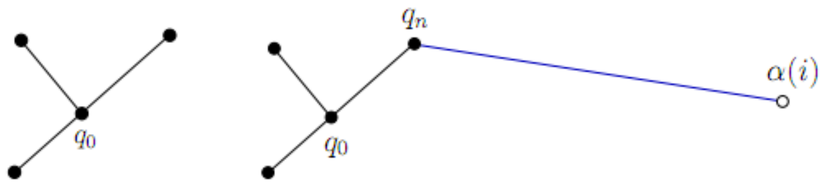
RRT Practicalities

- 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

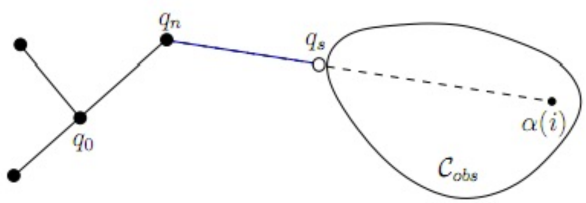
- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

RRT Extension

- No obstacles, holonomic:



- With obstacles, holonomic:



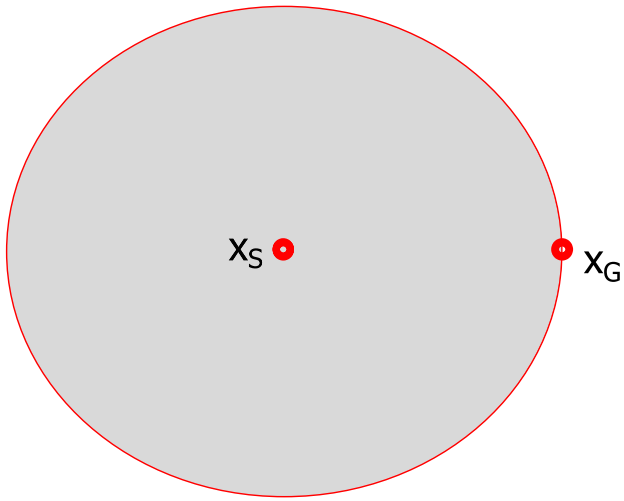
- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem

Growing RRT

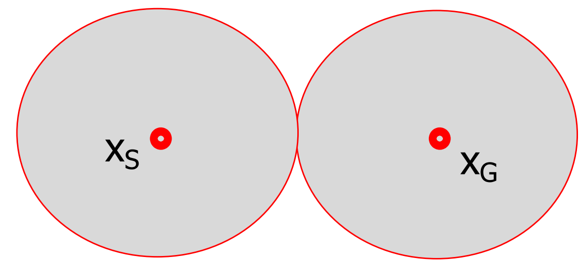
Demo: http://en.wikipedia.org/wiki/File:Rapidly-exploring_Random_Tree_(RRT)_500x373.gif

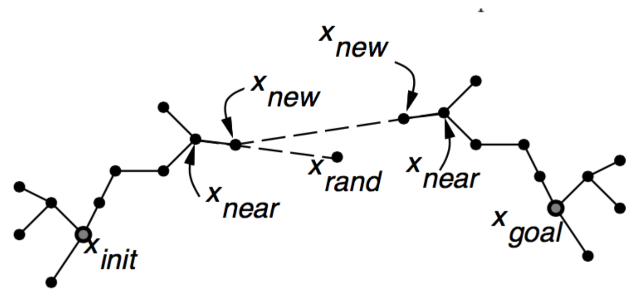
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:

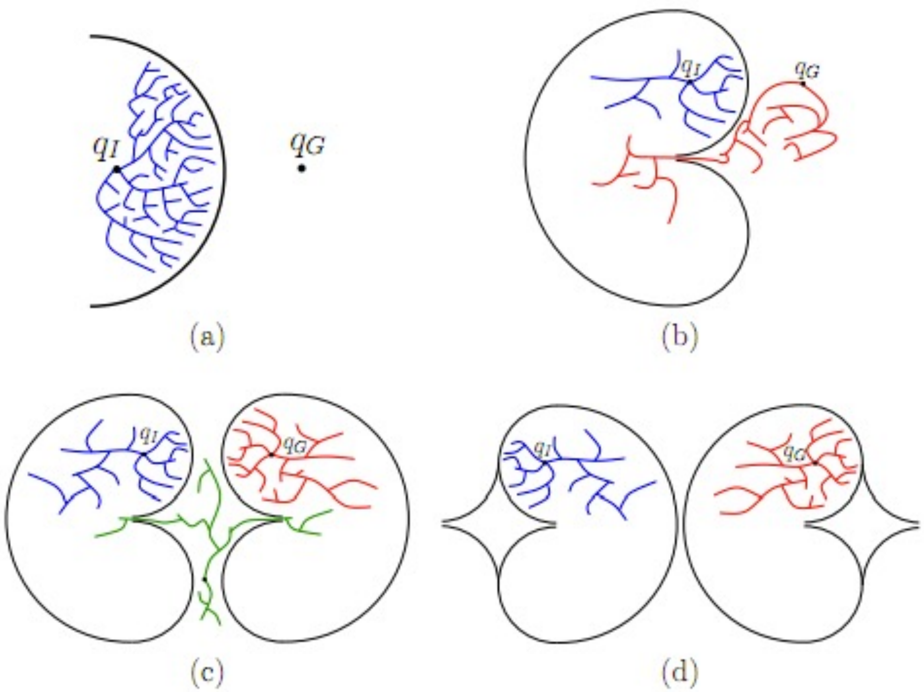




- Difference more and more pronounced as dimensionality increases

Multi-directional RRT

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



RRT*

- 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

RRT*

Algorithm 6: RRT*

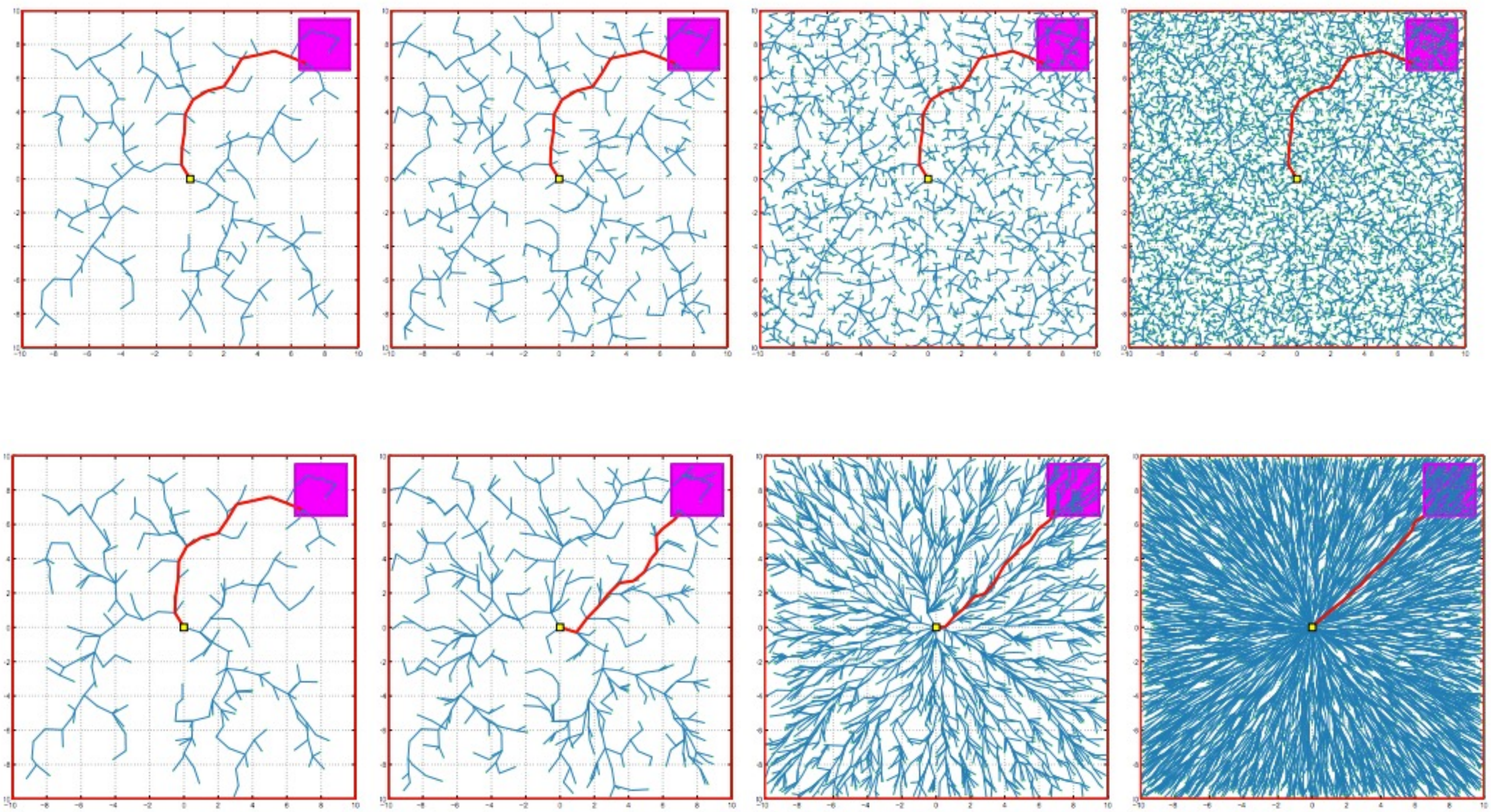
$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$
for $i = 1, \ldots, n$ **do**
    $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$
    $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
    $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})\ ;$
    if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
        $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})\ ;$
        $V \leftarrow V \cup \{x_{\text{new}}\};$
        $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$
        foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
            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**
                $x_{\text{min}} \leftarrow x_{\text{near}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
        $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$
        foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
            if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
            then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}});$
            $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E);$

Source: Karaman and Frazzoli

RRT*

RRT

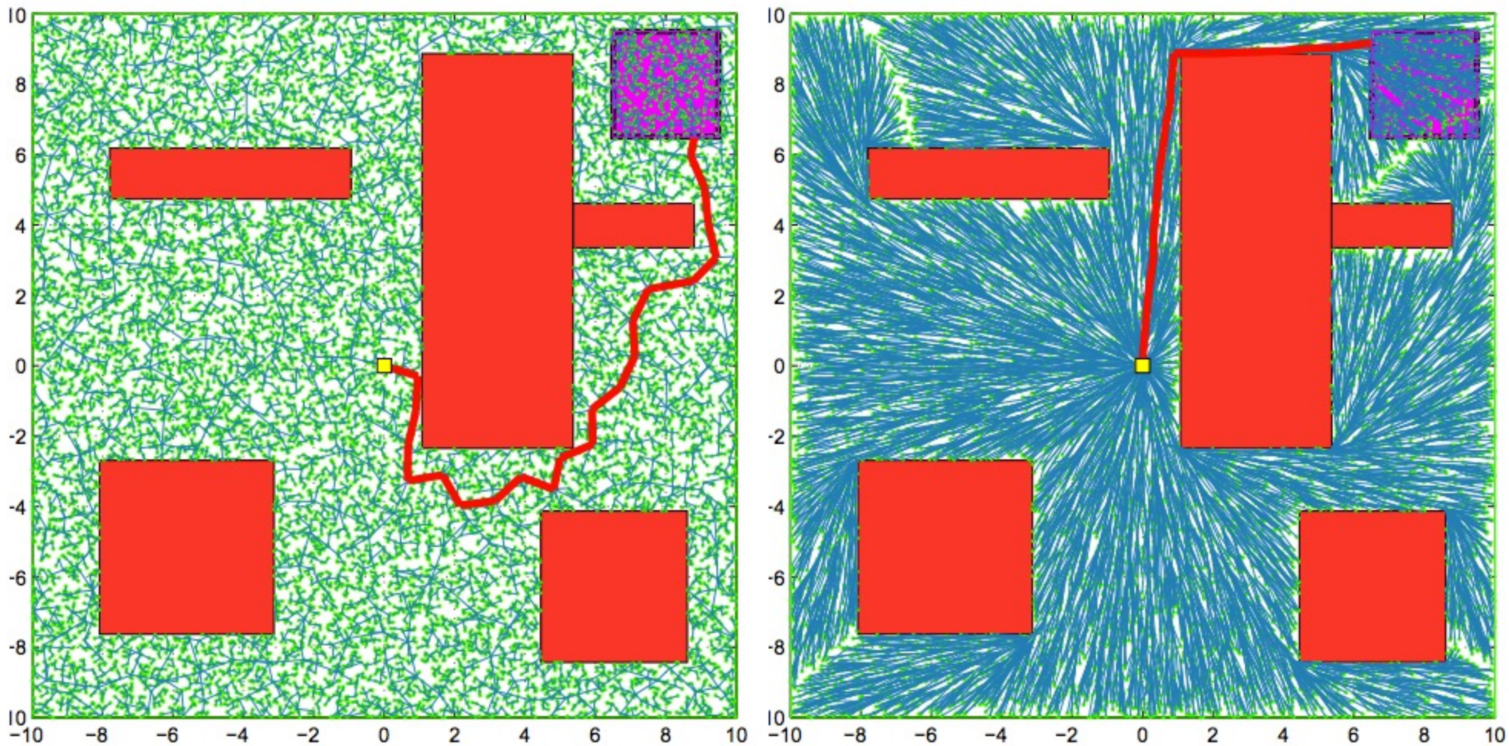
RRT*



Source: Karaman and Frazzoli

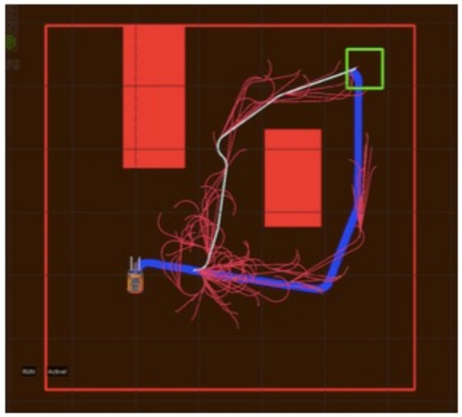
RRT*

RRT | RRT*

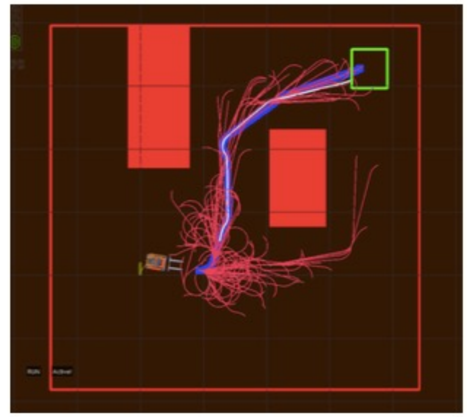


Source: Karaman and Frazzoli

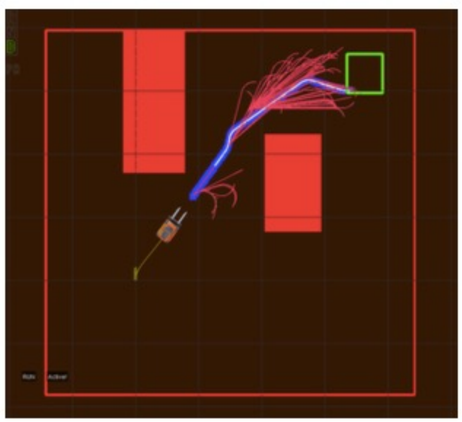
Real Time RRT*

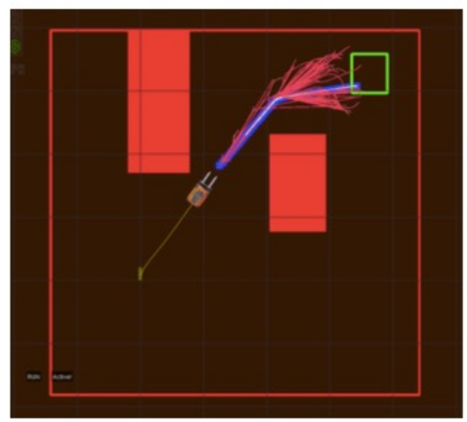


(a) RRT* run 1



(b) RRT* run 1





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
- Adam Fishman / Dieter Fox (https://rse-lab.cs.washington.edu/)
  - Motion Policy Networks: https://mpinets.github.io/
- Lydia Kavraki (http://www.kavrakilab.org/)
  - Motion in human workspaces: http://www.kavrakilab.org/nsf-nri-1317849.html