Sampling-Based Motion Planning

Pieter Abbeel
UC Berkeley EECS

Many images from Lavalle, Planning Algorithms

Motion Planning

- **Problem**
  - Given start state x_S, goal state x_G
  - Asked for: a sequence of control inputs that leads from start to goal
- Why tricky?
  - Need to avoid obstacles
  - For systems with underactuated dynamics: can’t simply move along any coordinate at will
    - E.g., car, helicopter, airplane, but also robot manipulator hitting joint limits

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

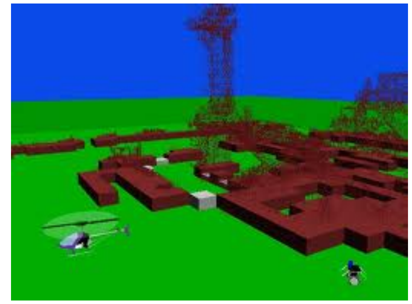
- Or, with constraints, (which would require using an infeasible method):

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

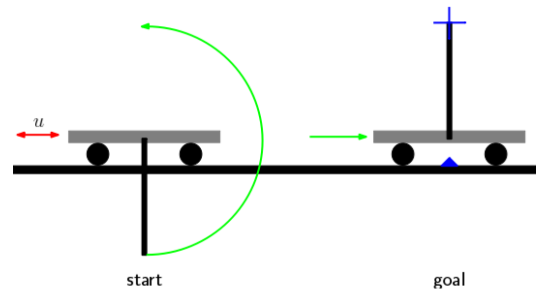
- *Can work surprisingly well, but for more complicated problems with longer horizons, often get stuck in local maxima that don't reach the goal*

Examples

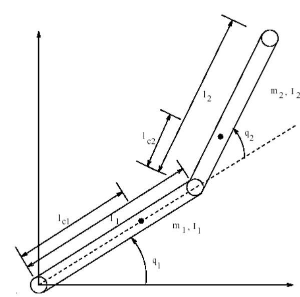
- Helicopter path planning



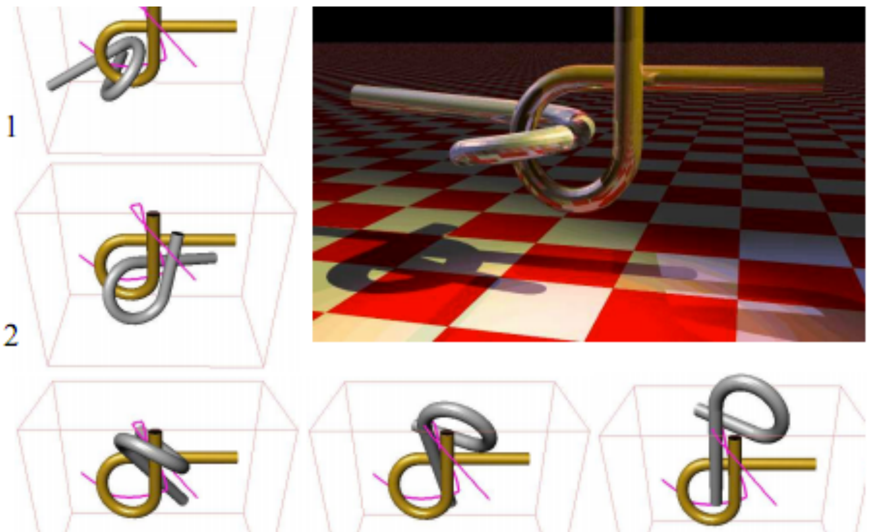
- Swinging up cart-pole

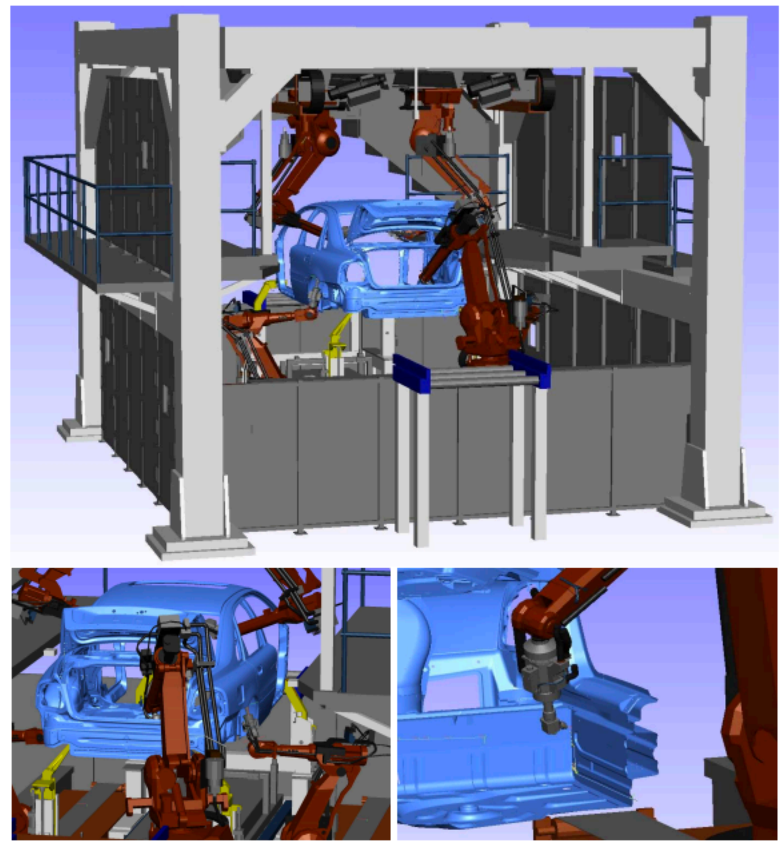


- Acrobot

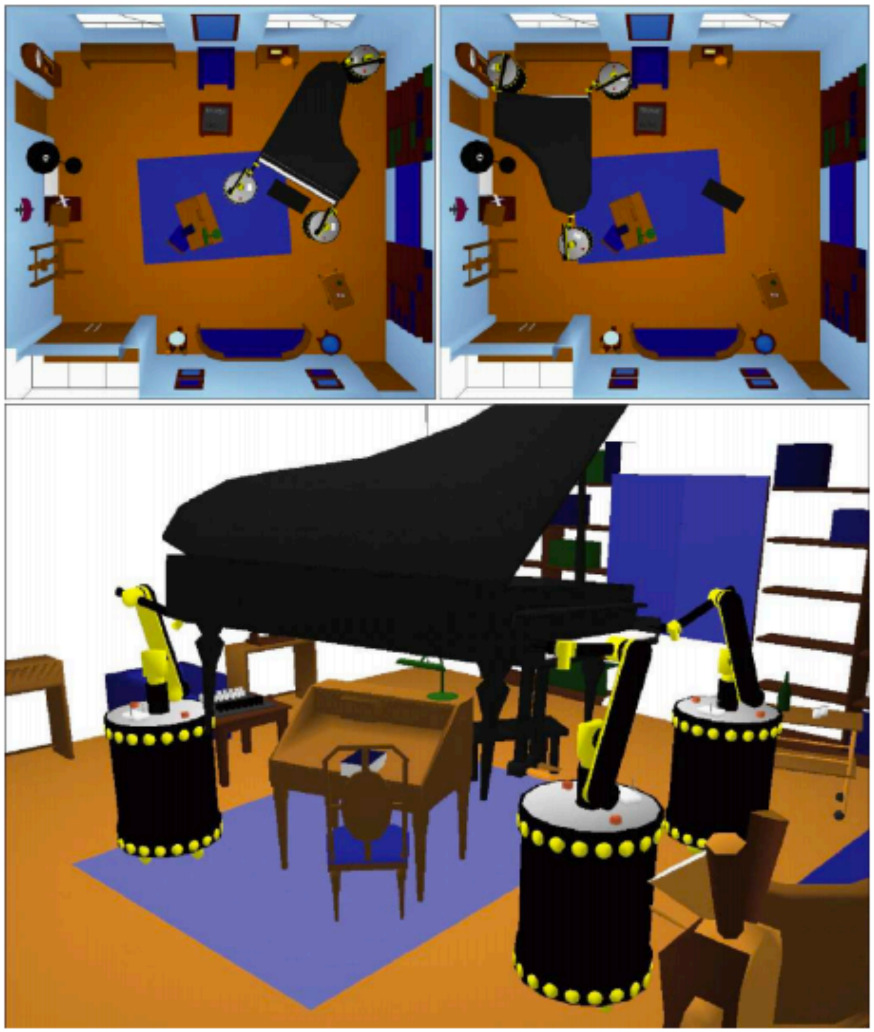


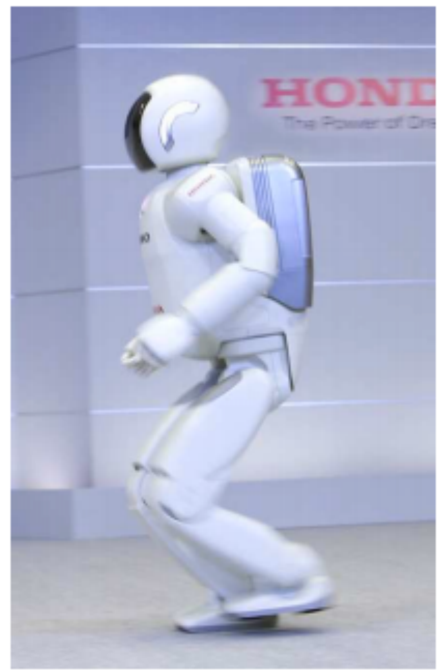
Examples

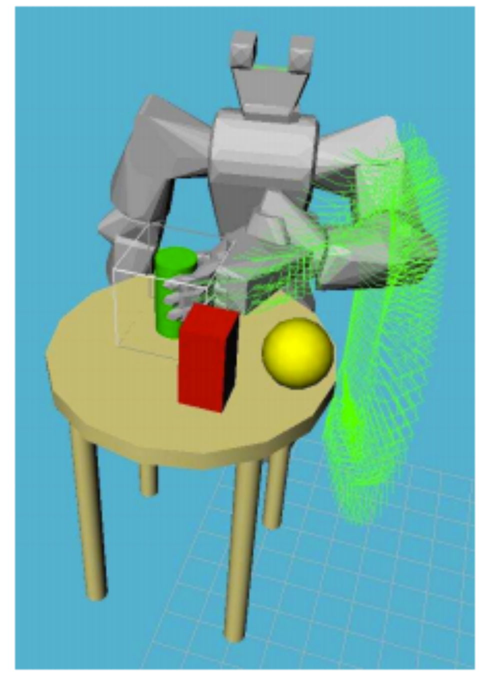




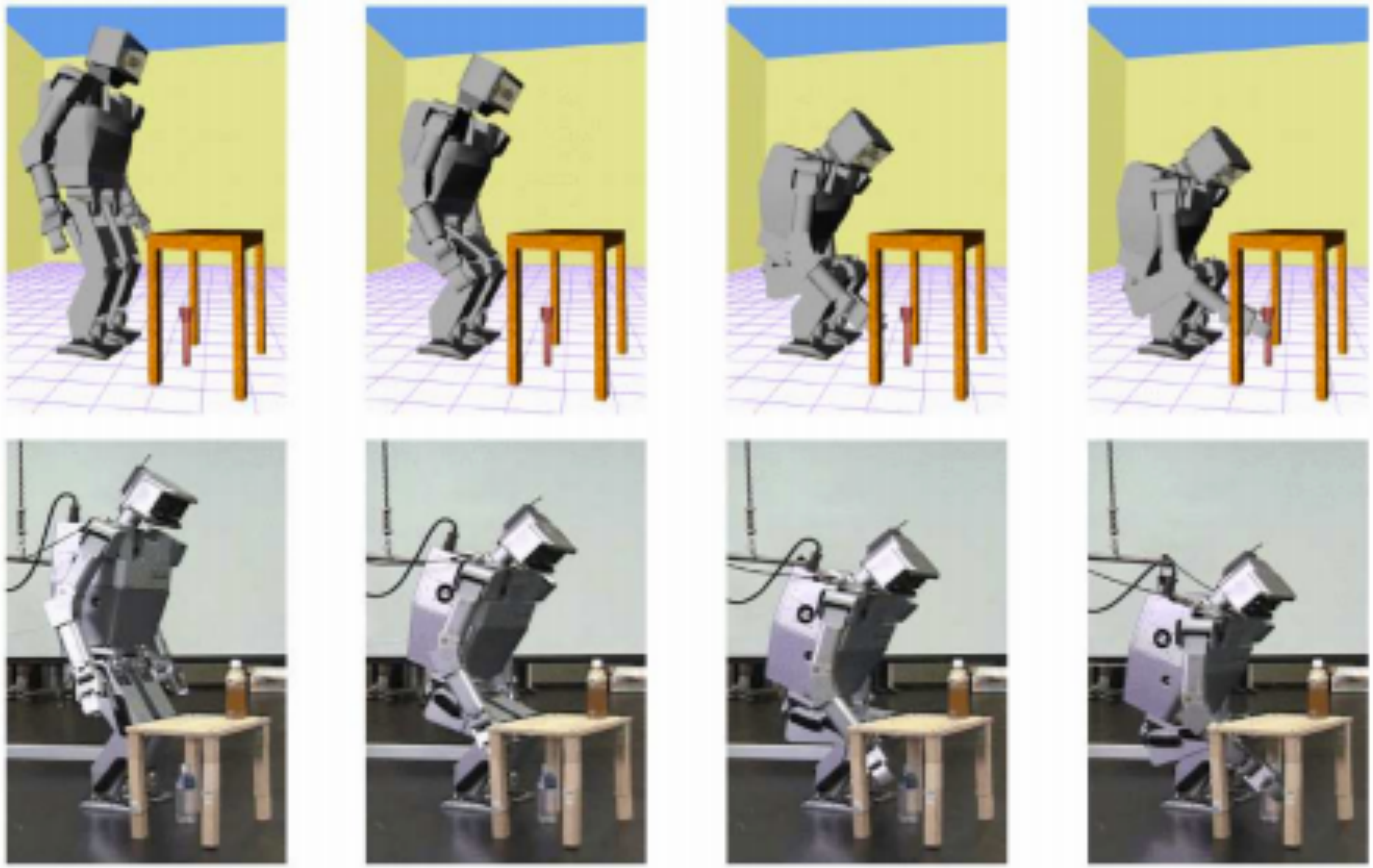
Examples







Examples



Motion Planning: Outline

- Configuration Space
- Probabilistic Roadmap
  - Boundary Value Problem
  - Sampling
  - Collision checking
- Rapidly-exploring Random Trees (RRTs)
- Smoothing

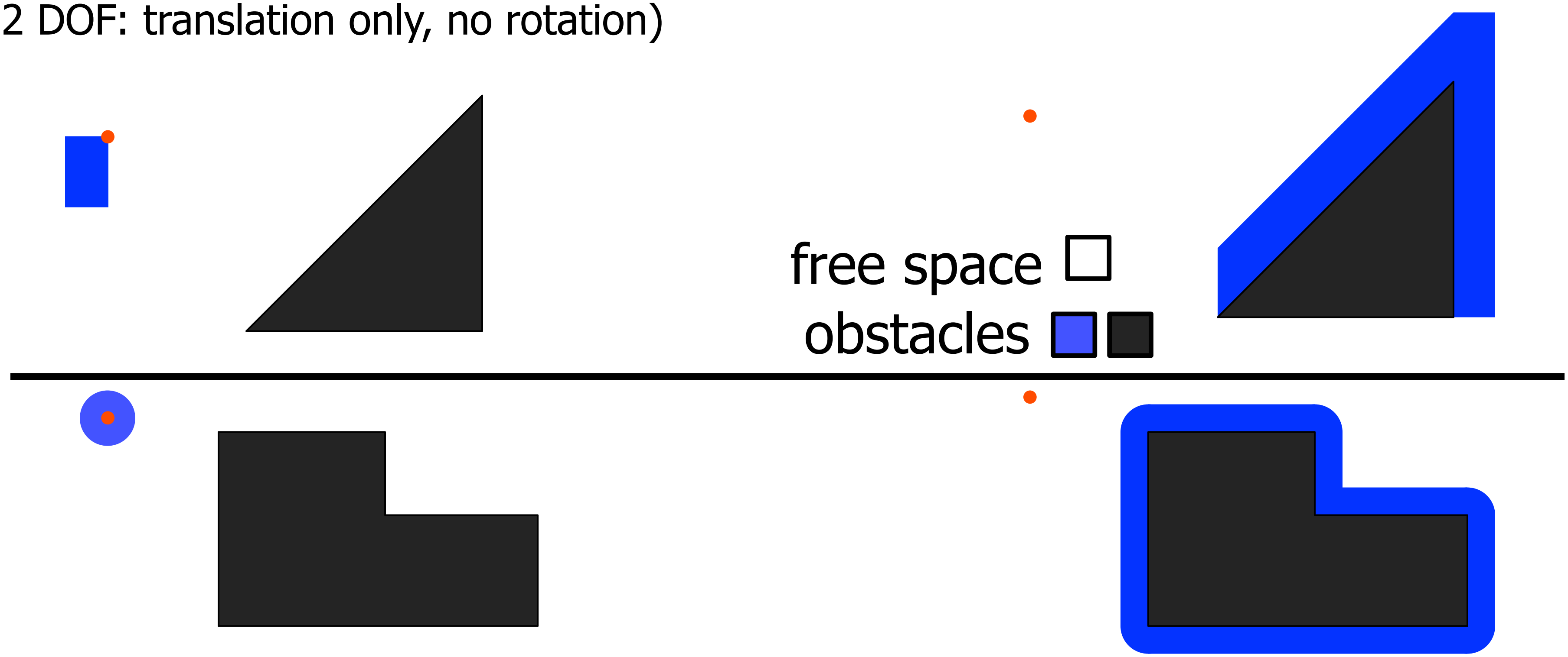
Configuration Space (C-Space)

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

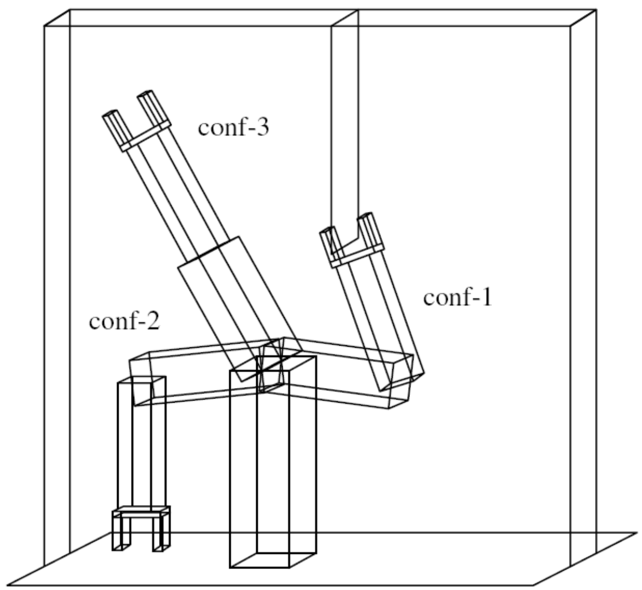
- obstacles → configuration space obstacles

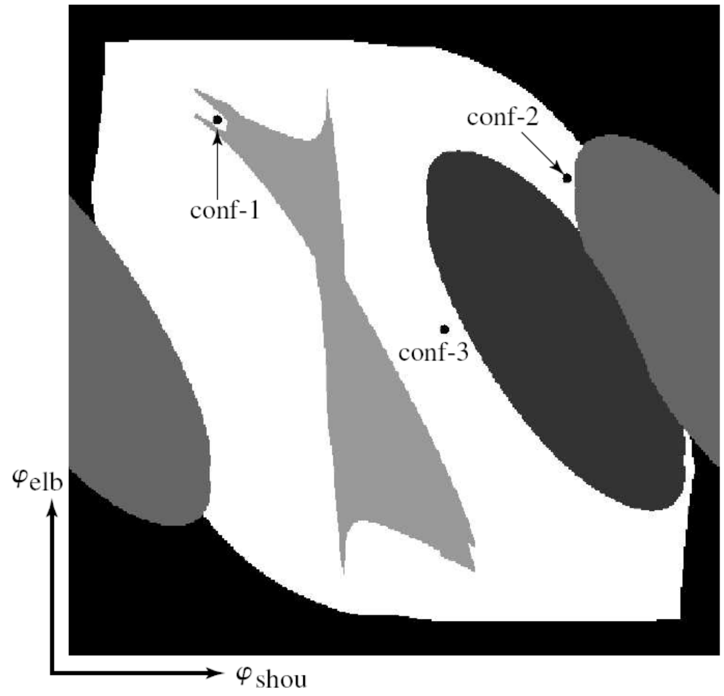
Workspace   *Configuration Space*

(2 DOF: translation only, no rotation)

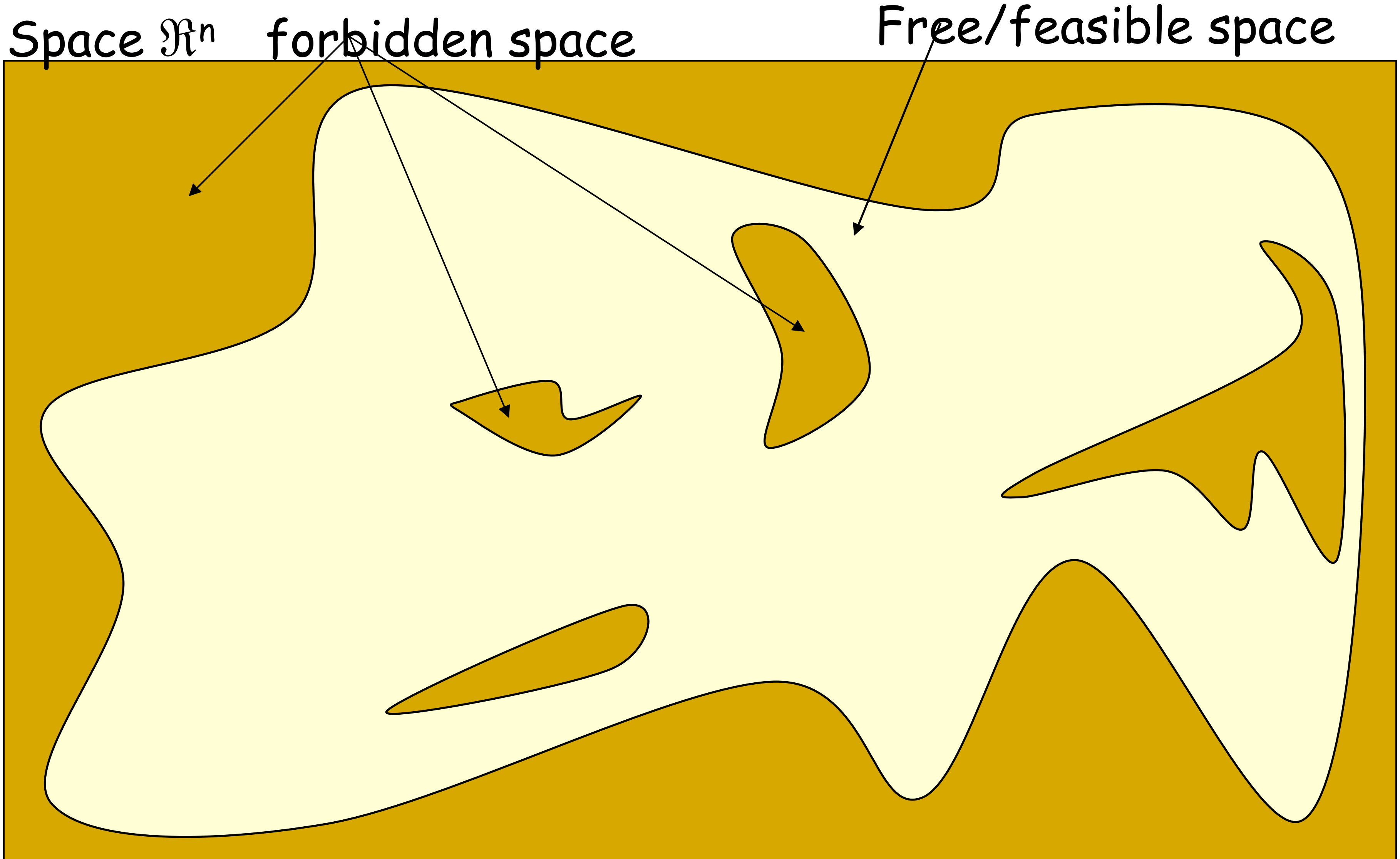


Motion planning



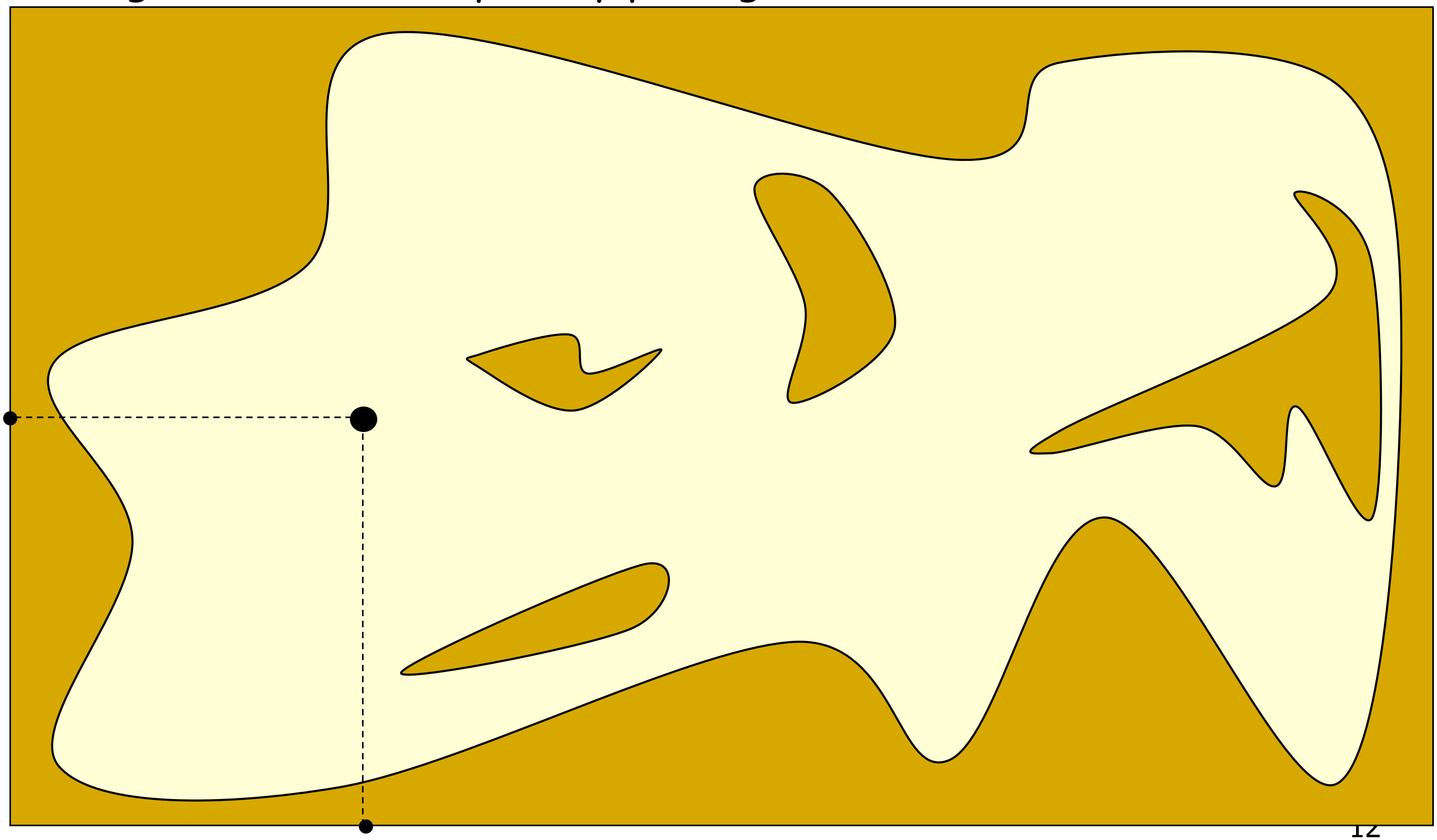


Probabilistic Roadmap (PRM)



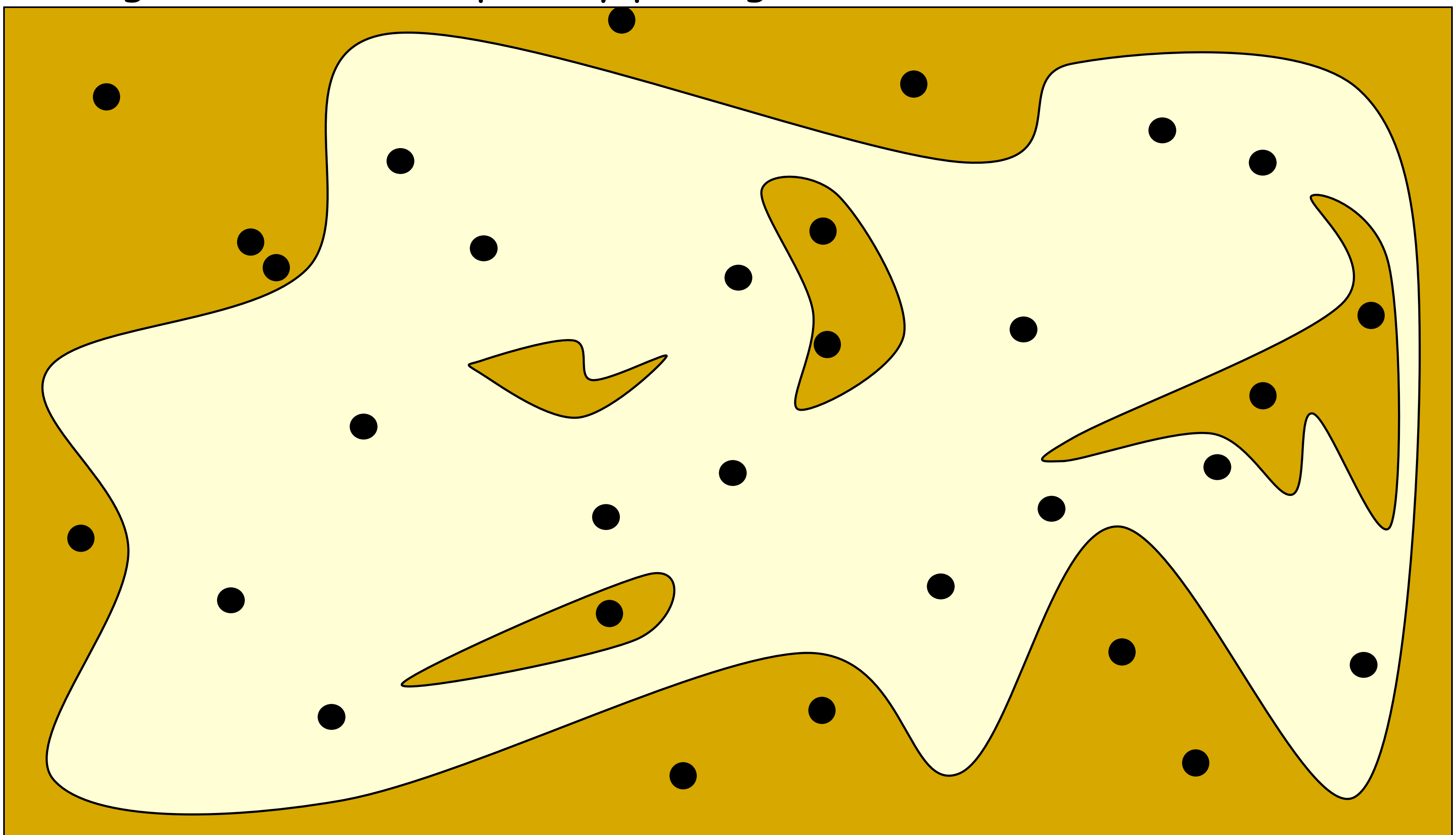
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



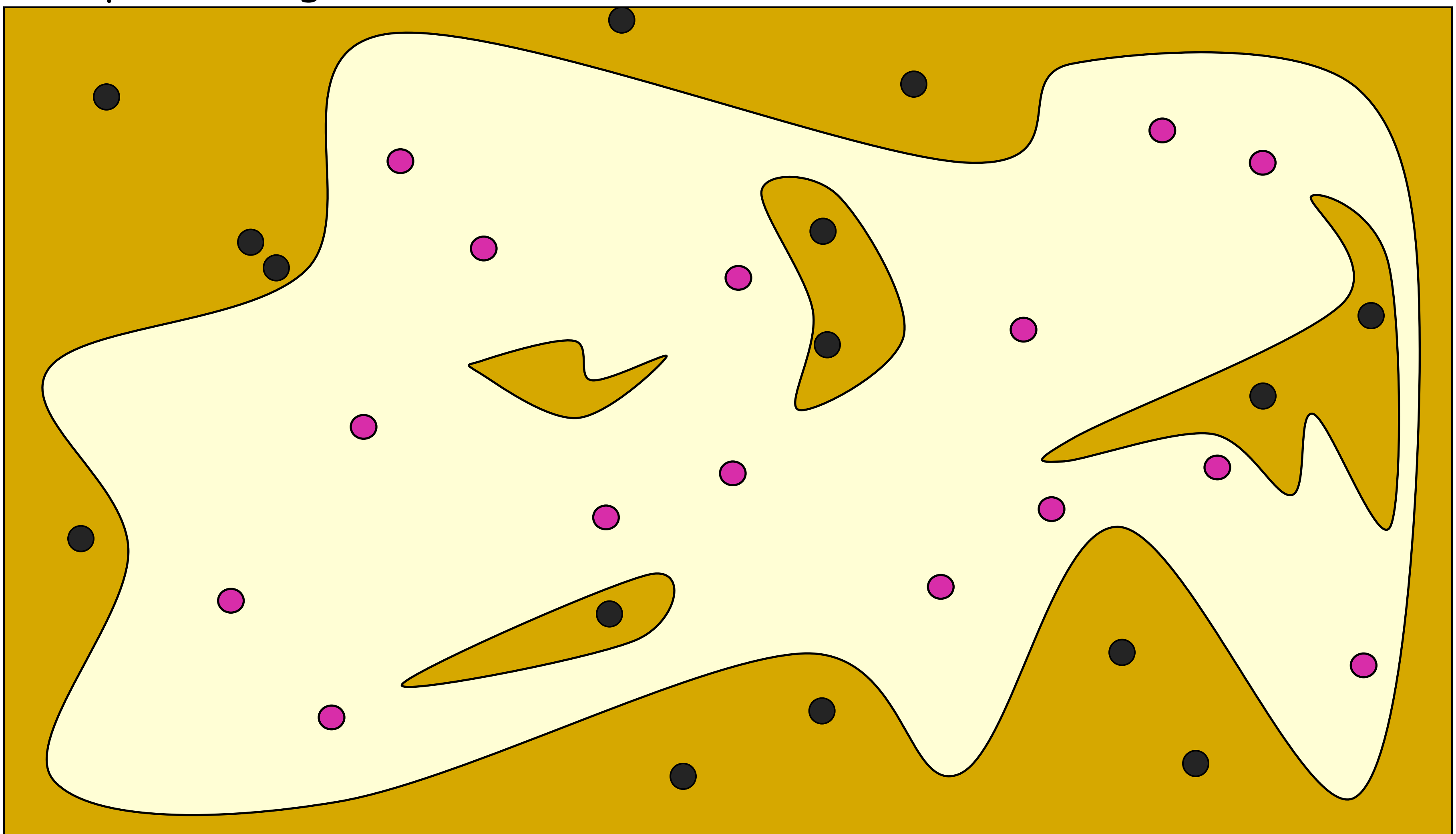
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



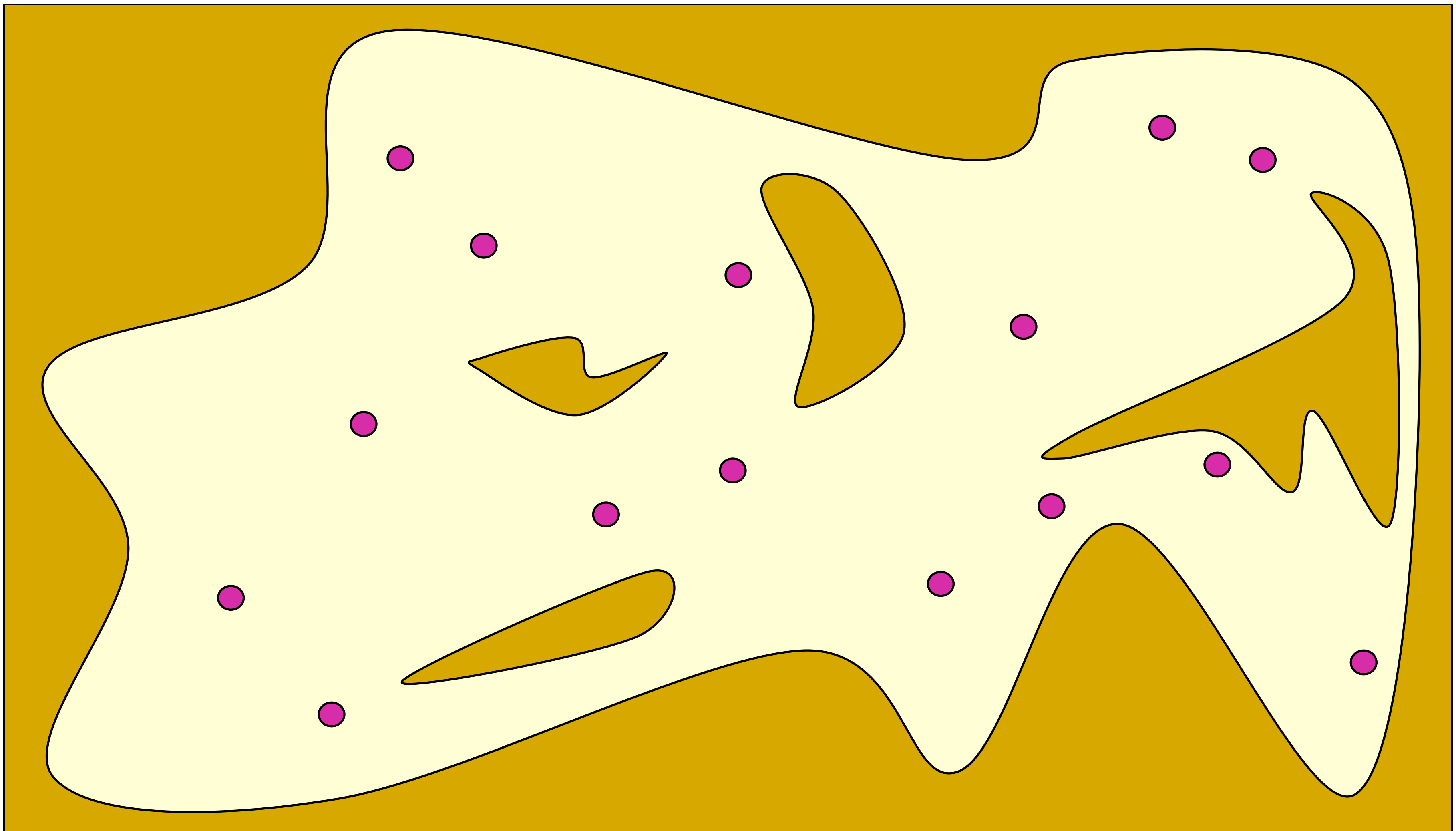
Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision



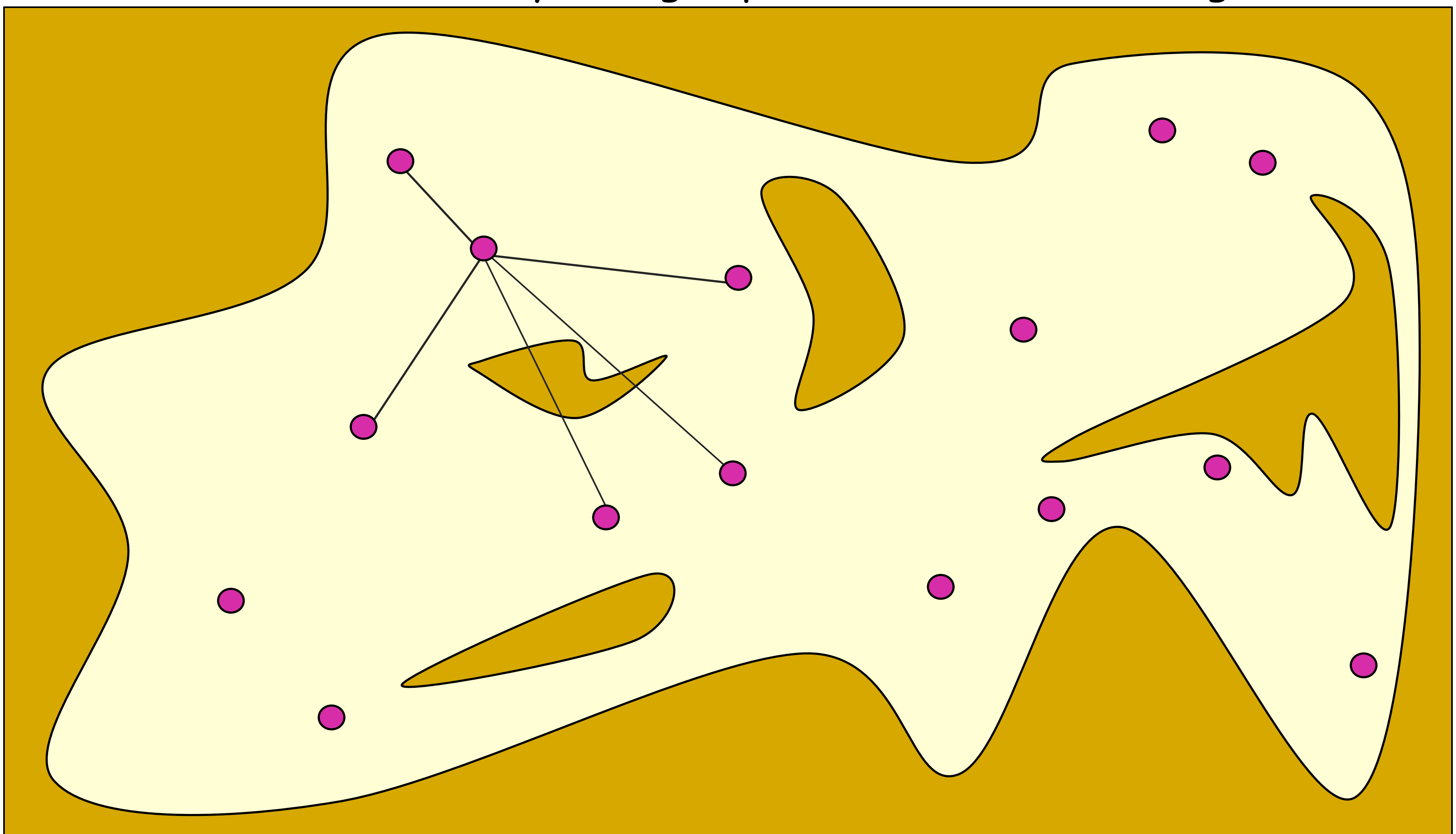
Probabilistic Roadmap (PRM)

The collision-free configurations are retained as milestones



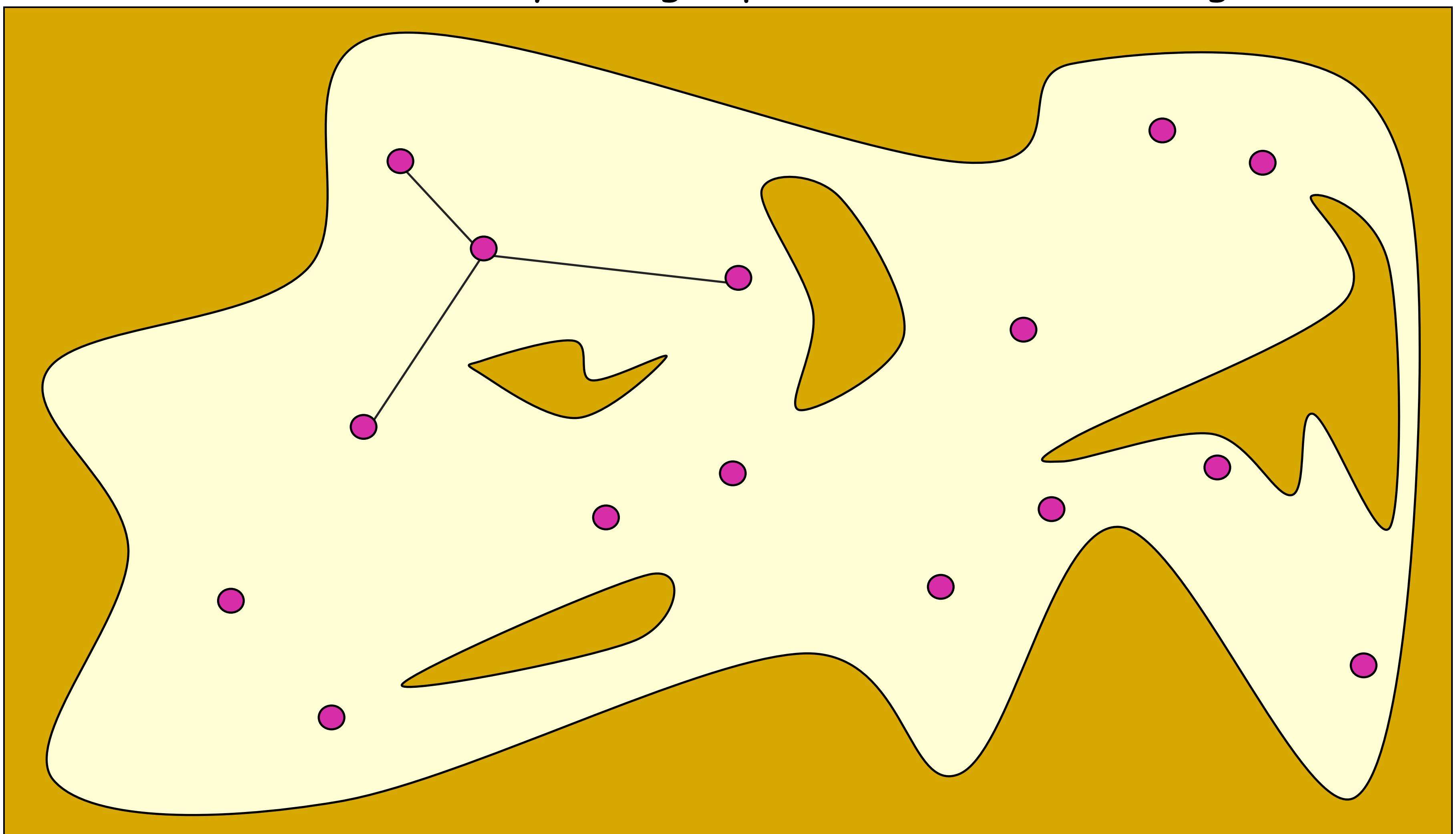
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



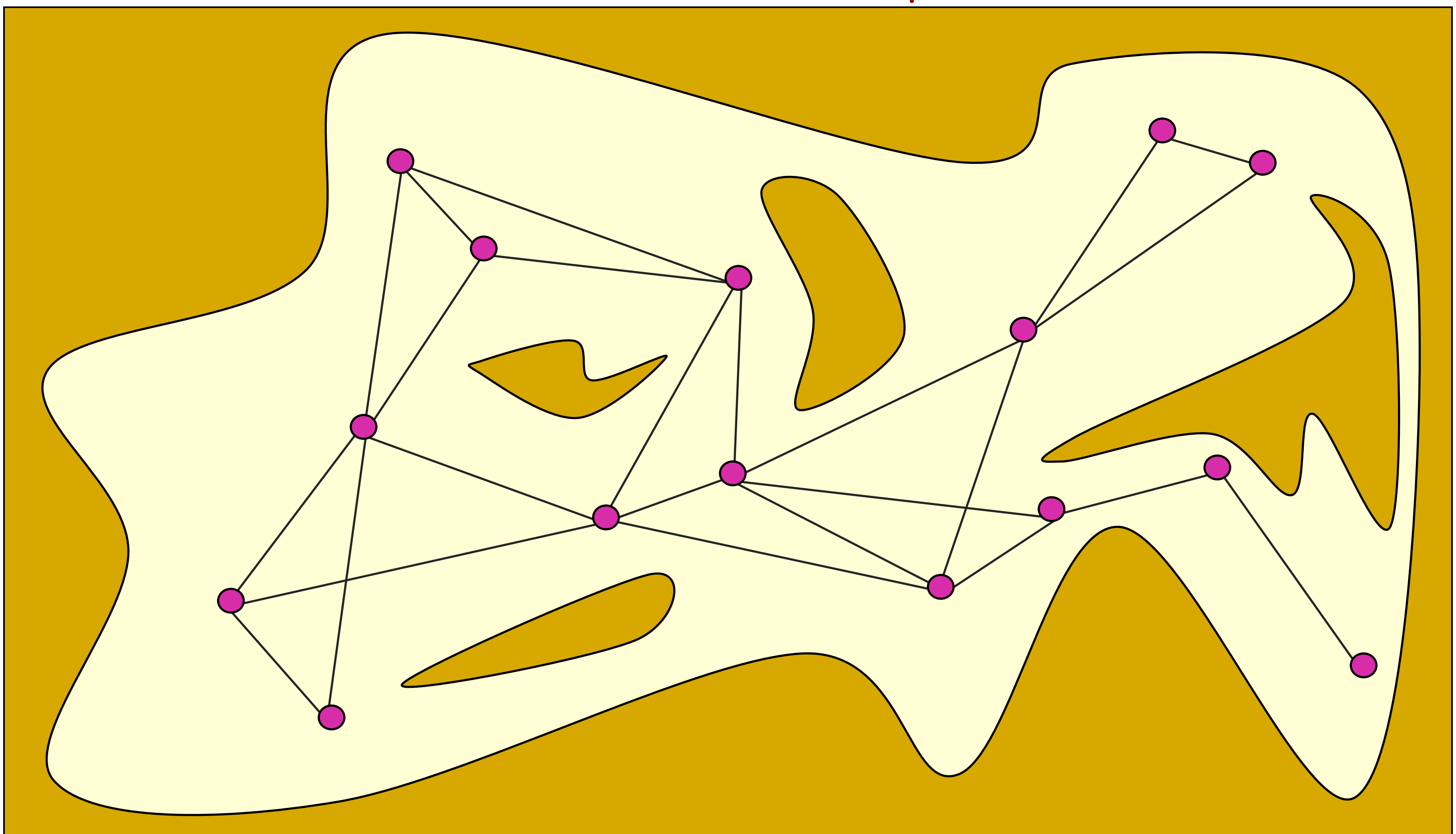
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



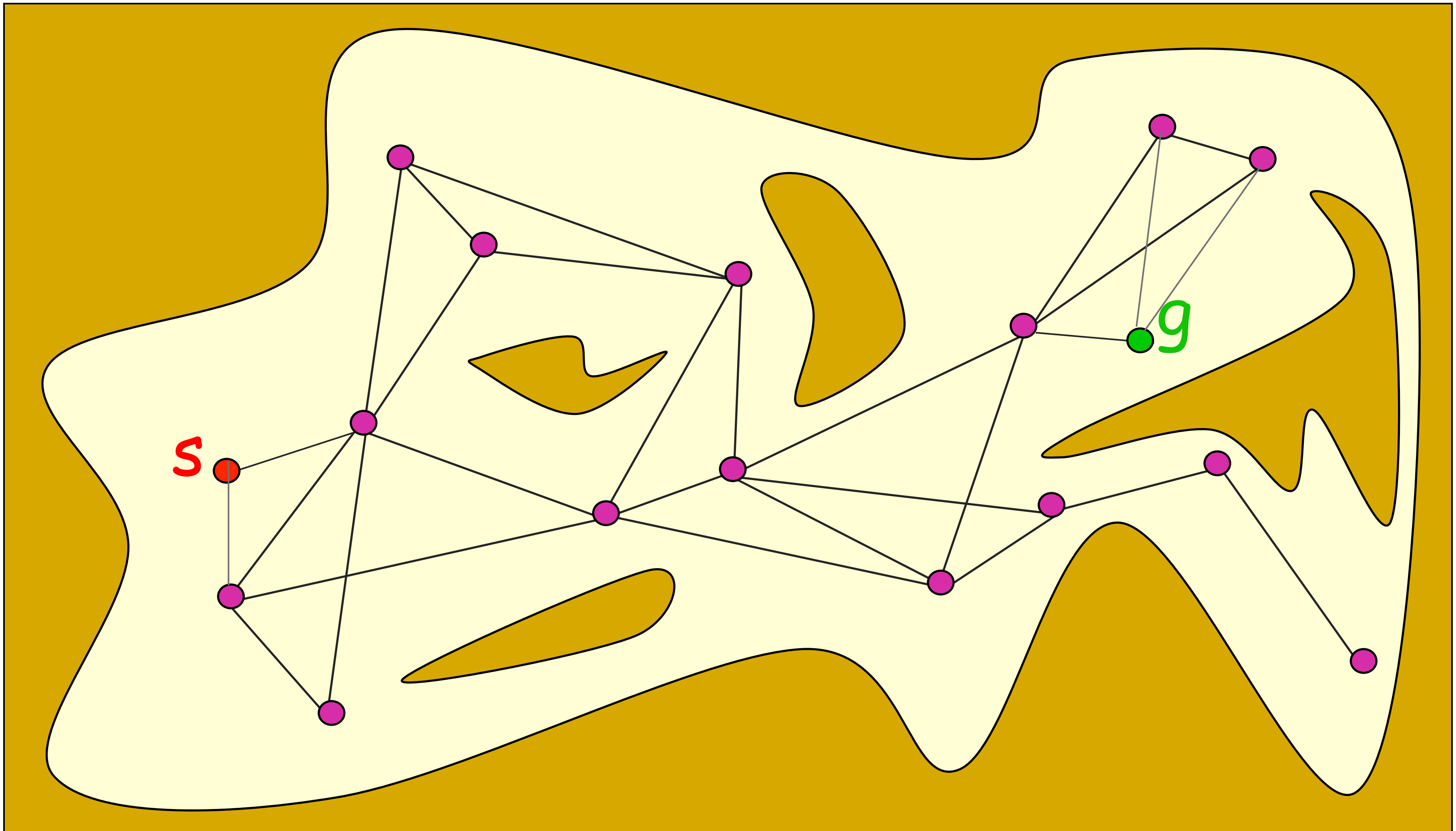
Probabilistic Roadmap (PRM)

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



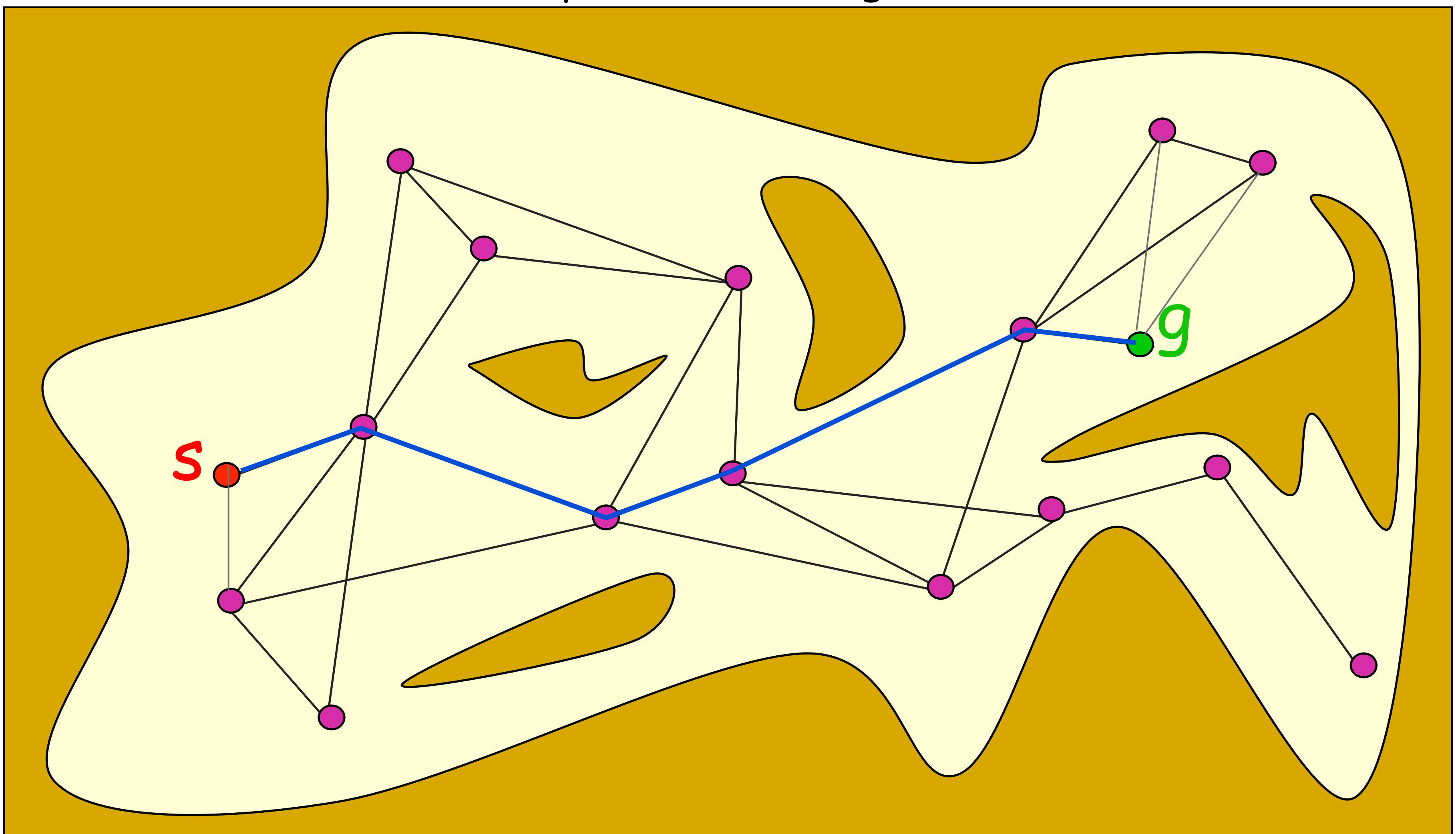
Probabilistic Roadmap (PRM)

The start and goal configurations are included as milestones



Probabilistic Roadmap (PRM)

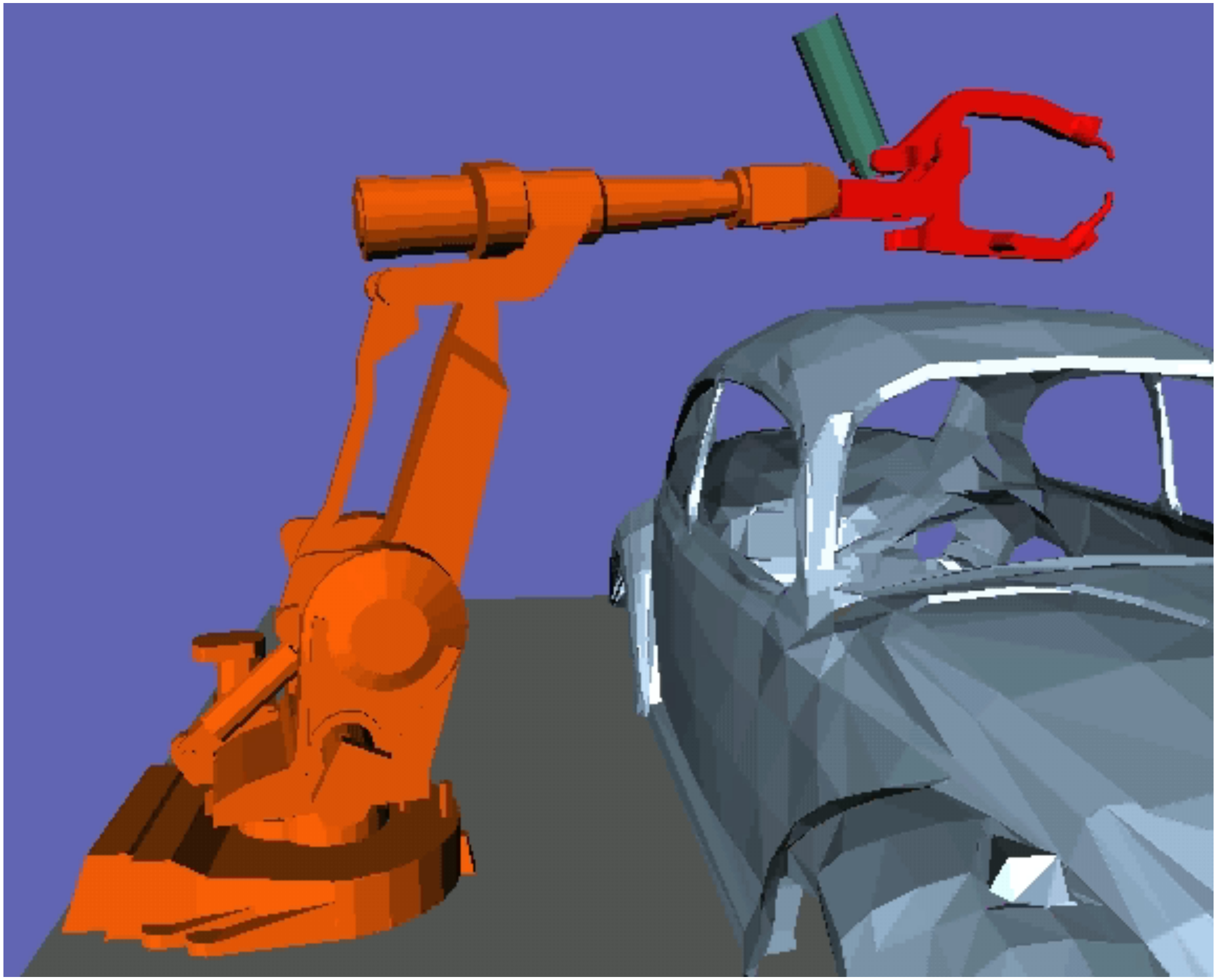
The PRM is searched for a path from s to g



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



PRM example 2

Sampling

- How to sample uniformly at random from $[0,1]^n$?
  - Sample uniformly at random from [0,1] for each coordinate
- How to sample uniformly at random from the surface of the n-D unit sphere?
  - Sample from n-D Gaussian → isotropic; then just normalize
- How to sample uniformly at random for orientations in 3-D?

PRM: Challenges

1. Connecting neighboring points: Only easy for holonomic systems (i.e., for which you can move each degree of freedom at will at any time). Generally requires solving a Boundary Value Problem

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

Typically solved without collision checking; later verified if valid by collision checking

2. Collision checking:

Often takes majority of time in applications (see Lavalle)

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 particular focus on generating a path

Rapidly exploring Random Trees

- 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

Rapidly-exploring Random Trees (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

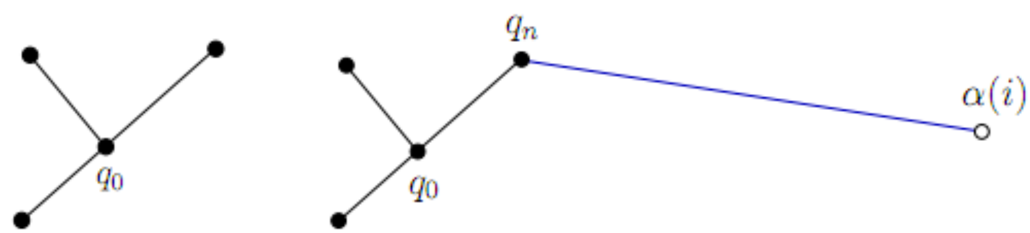
RRT Practicalities

- 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

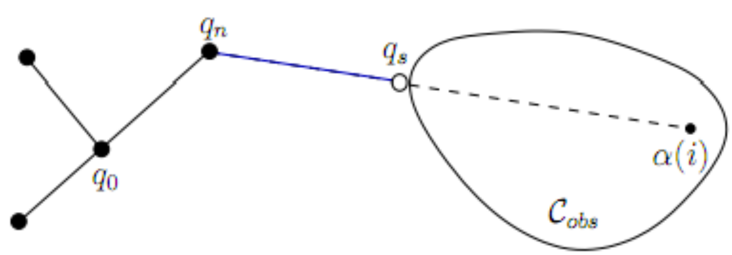
- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

RRT Extension

- 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

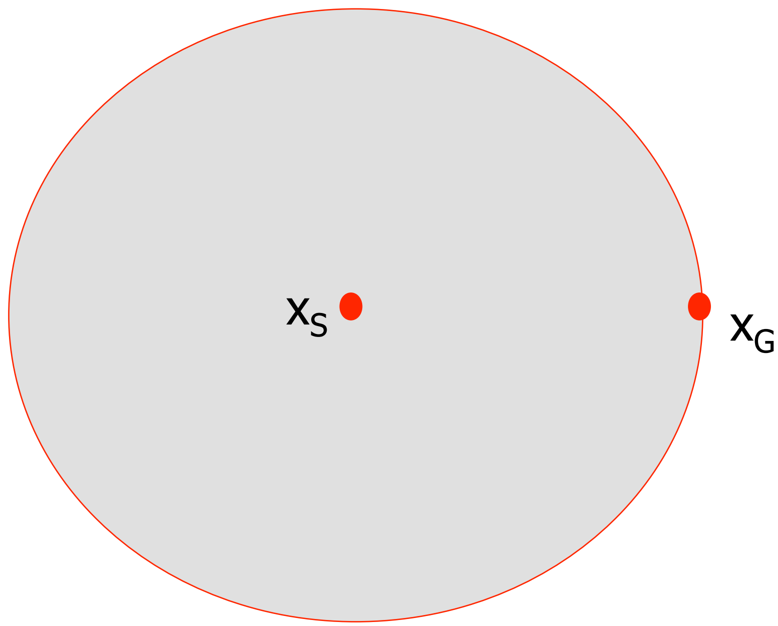
Growing RRT

45 iterations 390 iterations

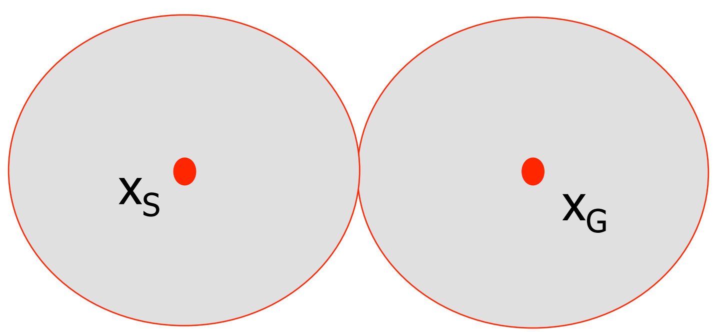
Demo: http://en.wikipedia.org/wiki/File:Rapidly-exploring_Random_Tree_(RRT)_500x373.gif

Bi-directional RRT

- Volume swept out by unidirectional RRT:



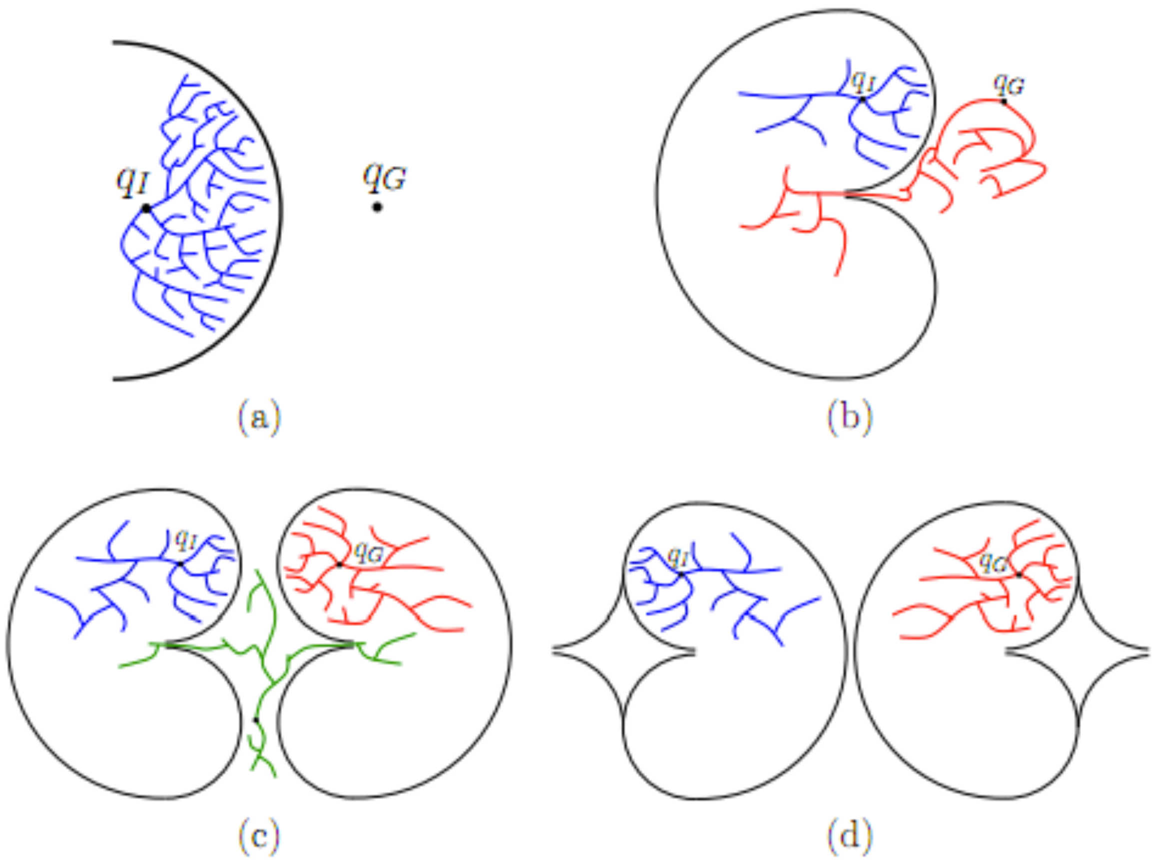
- Volume swept out by bi-directional RRT:



- Difference becomes even more pronounced in higher dimensions

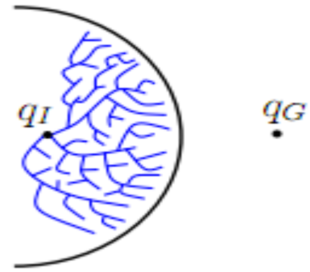
Multi-directional RRT

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



Resolution-Complete RRT (RC-RRT)

- Issue: nearest points chosen for expansion are (too) often the ones stuck behind an obstacle



RC-RRT solution:

- Choose a maximum number of times, m, you are willing to try to expand each node
- For each node in the tree, keep track of its Constraint Violation Frequency (CVF)
- Initialize CVF to zero when node is added to tree
- Whenever an expansion from the node is unsuccessful (e.g., per hitting an obstacle):
  - Increase CVF of that node by 1
  - Increase CVF of its parent node by 1/m, its grandparent 1/m^2, …
- When a node is selected for expansion, skip over it with probability CVF/m

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 $G = (V, E)$;

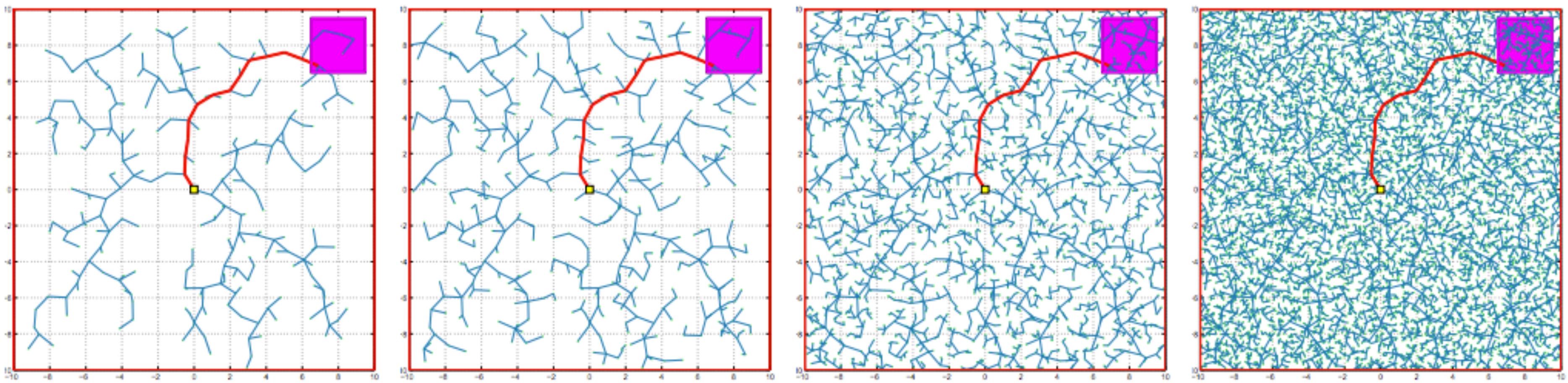
Source: Karaman and Frazzoli

RRT*

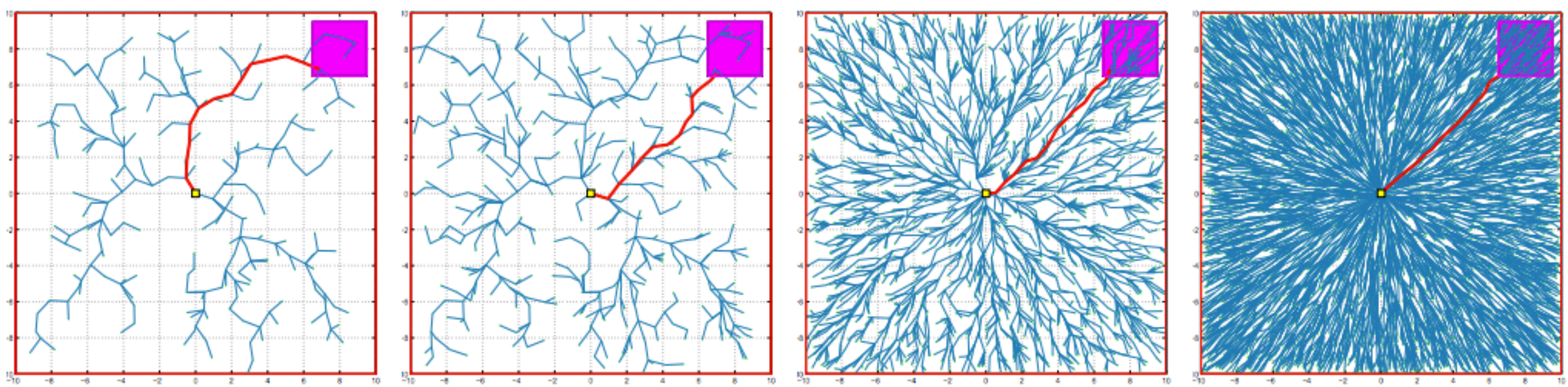
- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

RRT*

RRT



RRT*

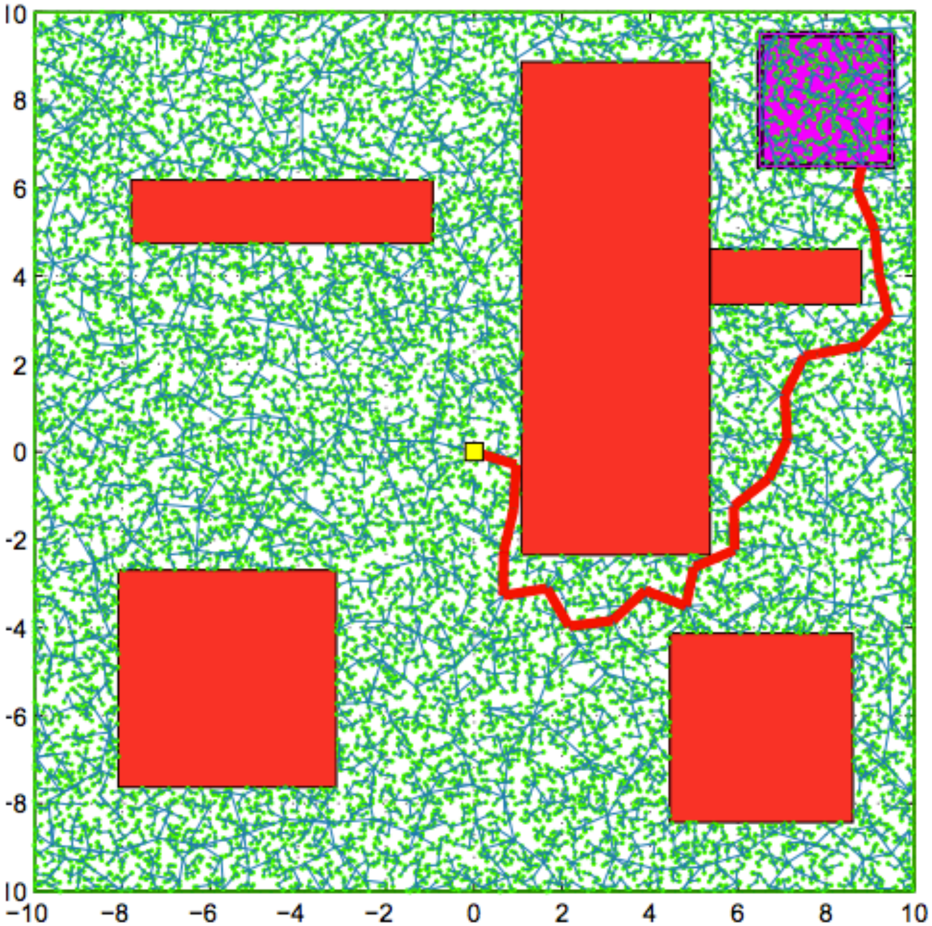


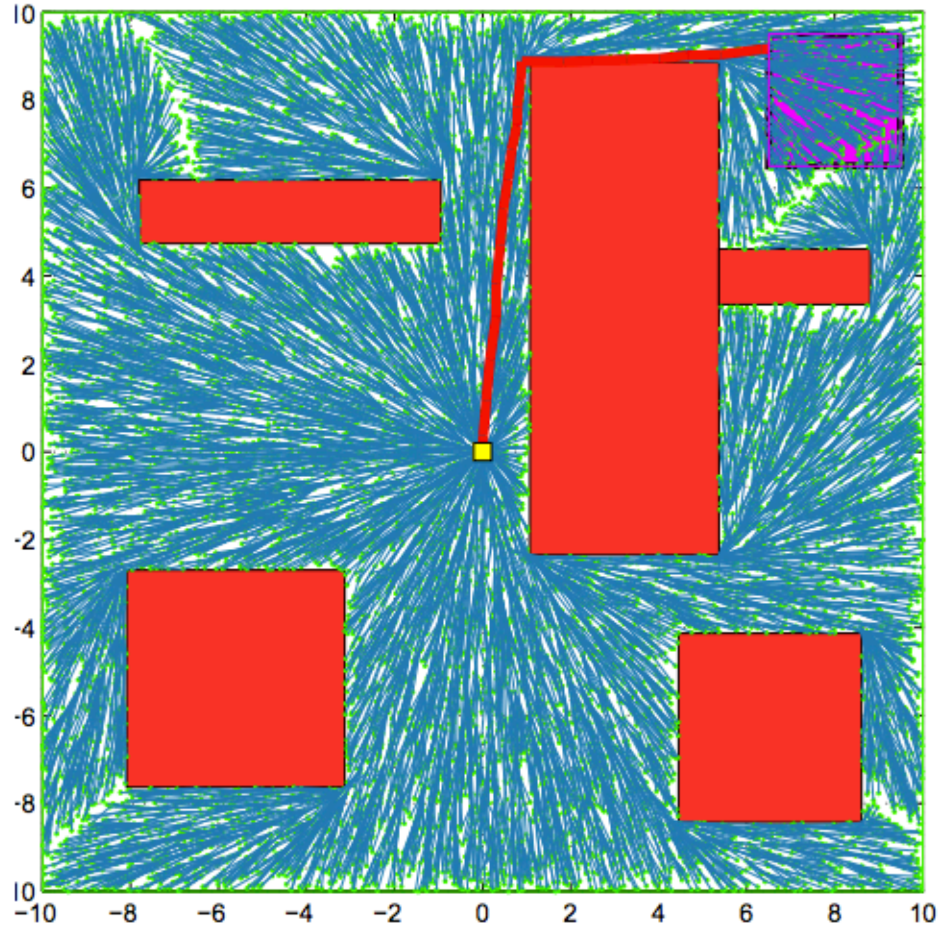
Source: Karaman and Frazzoli

RRT*

RRT

RRT*

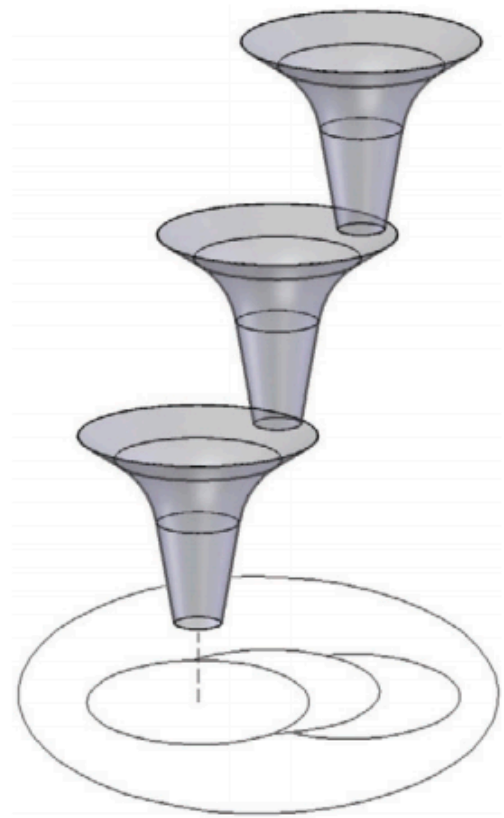




Source: Karaman and Frazzoli

LQR-trees (Tedrake, IJRR 2010)

- Idea: grow a randomized tree of stabilizing controllers to the goal
  - Like RRT
  - Can discard sample points in already stabilized region



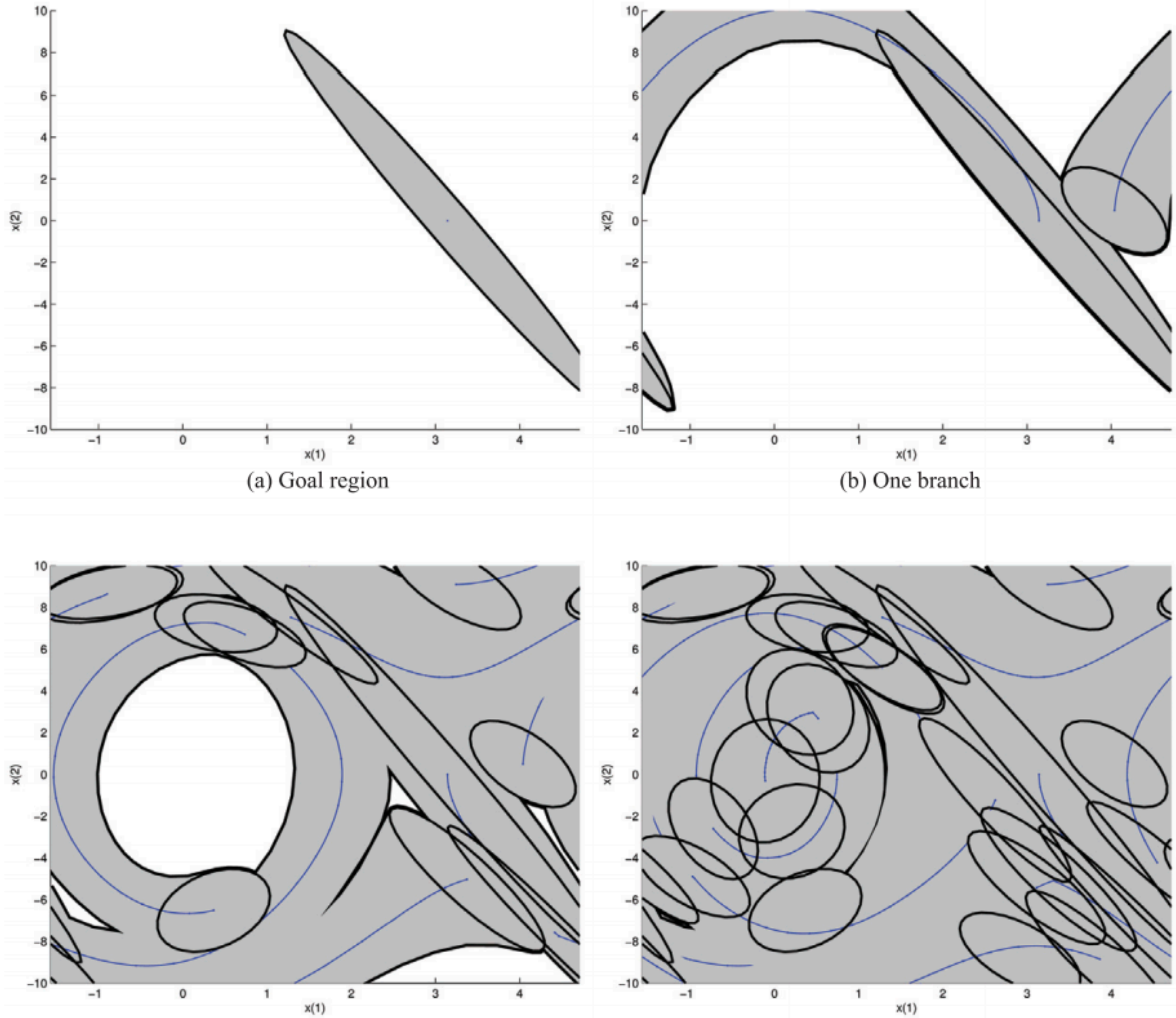
LQR-trees (Tedrake)

Algorithm 1 LQR-tree ($\mathbf{f}, \mathbf{x}_G, \mathbf{u}_G, \mathbf{Q}, \mathbf{R}$)

$[\mathbf{A}, \mathbf{B}] \Leftarrow$ linearization of $\mathbf{f}(\mathbf{x}, \mathbf{u})$ around $(\mathbf{x}_G, \mathbf{u}_G)$
$[\mathbf{K}, \mathbf{S}] \Leftarrow \mathrm{LQR}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$
$\rho_c \Leftarrow$ level set computed as described in §3.1.1
T.init($\{\mathbf{x}_g, \mathbf{u}_g, \mathbf{S}, \mathbf{K}, \rho_c, \mathrm{NULL}\}$)
for $k = 1$ to K **do**
    $\mathbf{x}_{\mathrm{rand}} \Leftarrow$ random sample
    if $\mathbf{x}_{\mathrm{rand}} \in \mathcal{C}_k$ **then**
        continue
    end if
    $[t, \mathbf{x}_0(t), \mathbf{u}_0(t)]$ from trajectory optimization with a "final tree constraint"
    if $\mathbf{x}_0(t_f) \notin \mathcal{T}_k$ **then**
        continue
    end if
    $[\mathbf{K}(t), \mathbf{S}(t)]$ from time-varying LQR
    $\rho_c \Leftarrow$ level set computed as in §3.1.1
    $i \Leftarrow$ pointer to branch in T containing $\mathbf{x}_0(t_f)$
    T.add-branch($\mathbf{x}_0(t), \mathbf{u}_0(t), \mathbf{S}(t), \mathbf{K}(t), \rho_c, i$)
end for

Ck: stabilized region after iteration k

LQR-trees (Tedrake)



(a) Goal region (b) One branch

(c) Six branches (d) Thirteen branches

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.