CS 287 Advanced Robotics (Fall 2019)

Lecture 9: Motion Planning

Lecture by: Huazhe (Harry) Xu
Slides by: Pieter Abbeel
UC Berkeley EECS

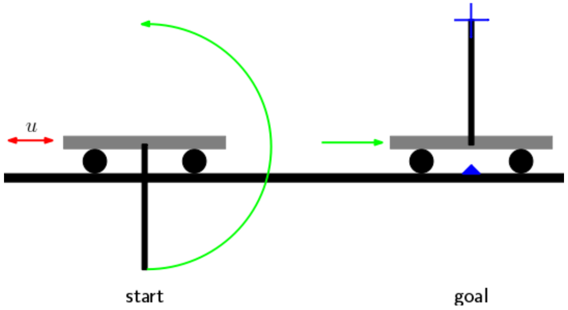
Many images from Lavalle, Planning Algorithms

Motion Planning

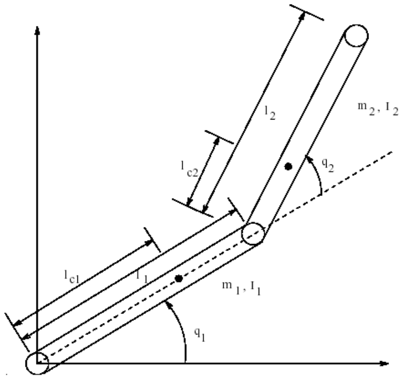
- **Problem**
  - Given start state x_S, goal state x_G
  - Asked for: a sequence of control inputs that leads from start to goal
- Why tricky?
  - Need to avoid obstacles
  - For systems with underactuated dynamics: can't simply move along any coordinate at will
    - E.g., car, helicopter, airplane,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

Examples

Helicopter path planning

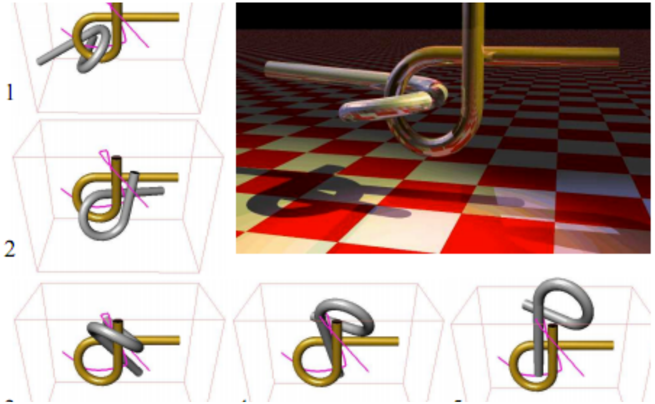


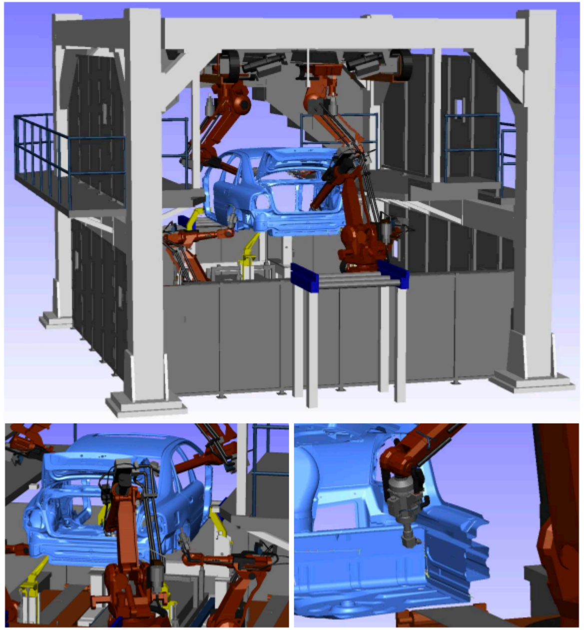
Cartpole swing-up



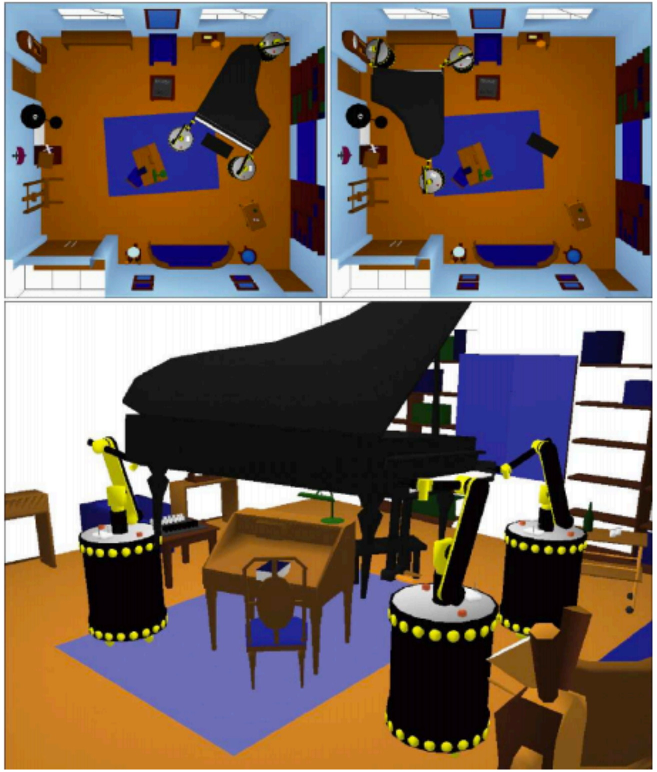
Acrobot

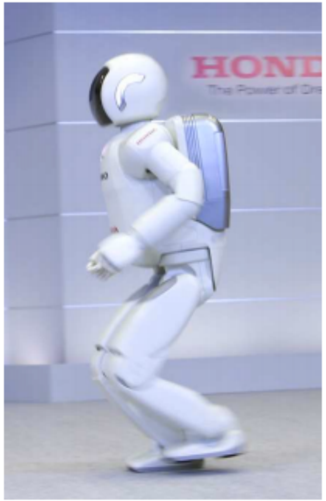
Examples

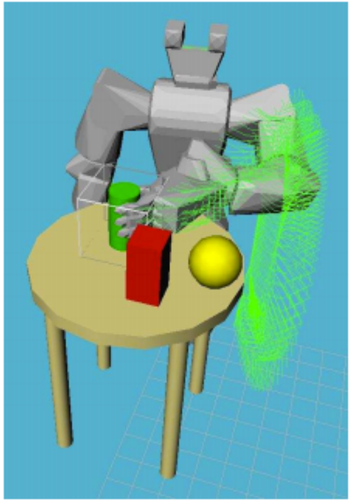




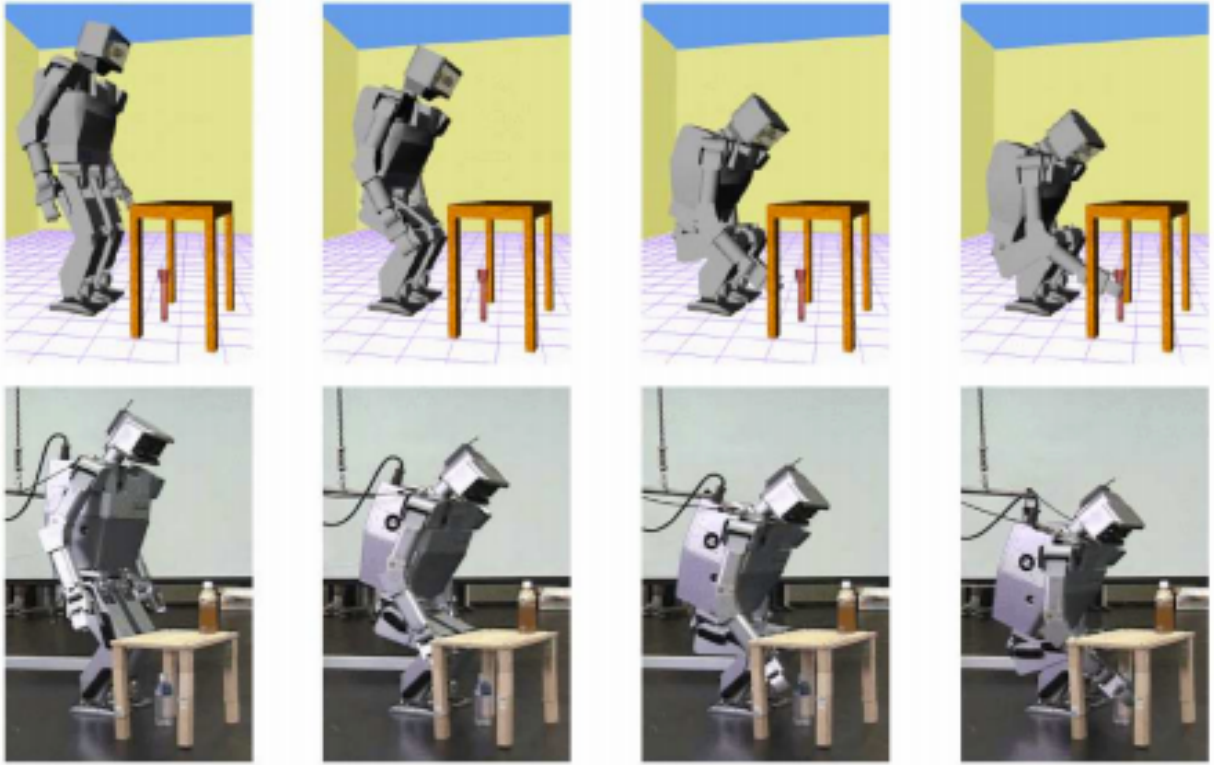
Examples







Examples



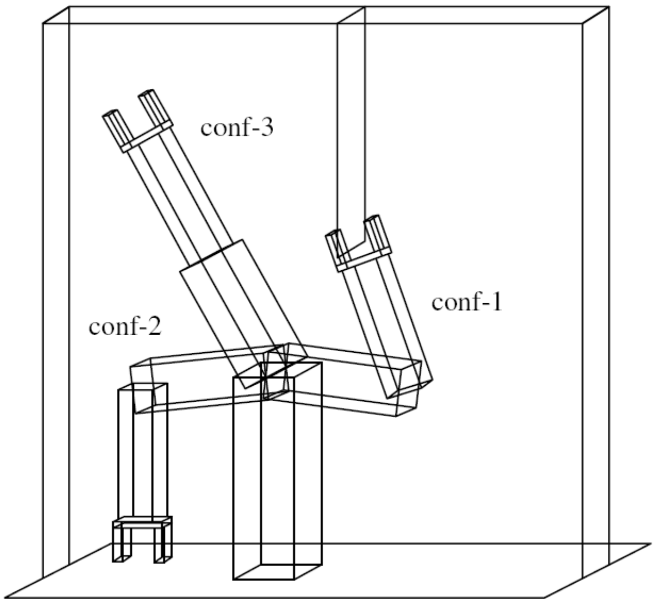
Motion Planning: Outline

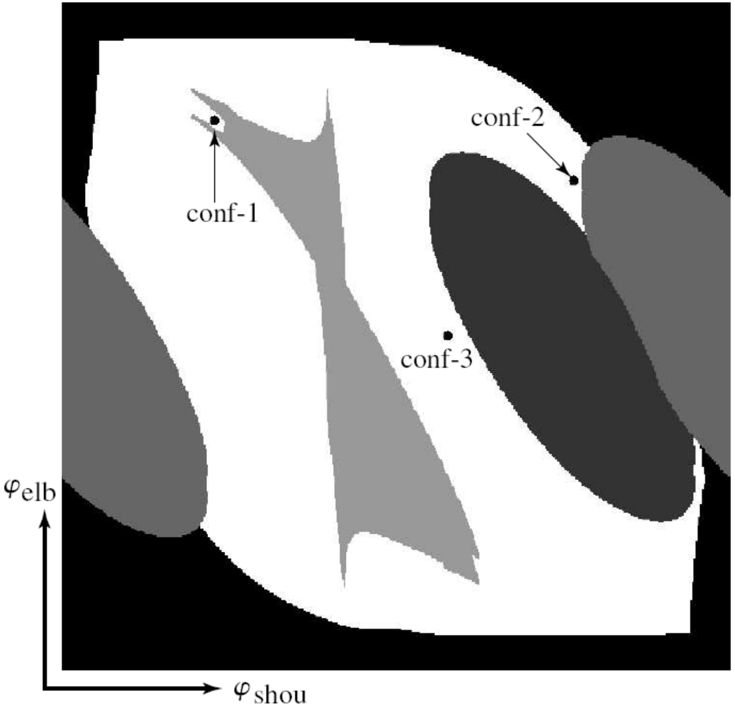
- Configuration Space
- Optimization-based Motion Planning
- Sampling-based Motion Planning
  - Probabilistic Roadmap
  - Rapidly-exploring Random Trees (RRTs)
  - Smoothing

Motion Planning: Outline

- ***Configuration Space***
- Optimization-based Motion Planning
- Sampling-based Motion Planning
  - Probabilistic Roadmap
  - Rapidly-exploring Random Trees (RRTs)
  - Smoothing

Motion planning





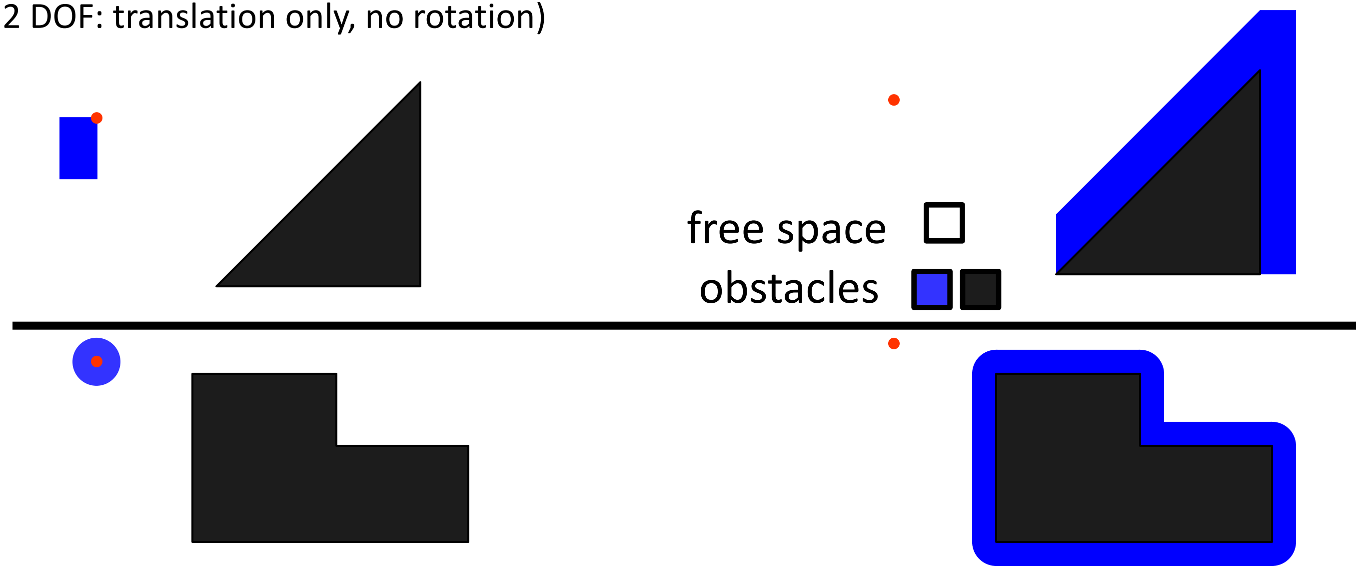
Configuration Space (C-Space)

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

- obstacles → configuration space obstacles

Workspace *Configuration Space*

(2 DOF: translation only, no rotation)



Motion Planning: Outline

- Configuration Space
- ***Optimization-based Motion Planning***
- Sampling-based Motion Planning
  - Probabilistic Roadmap
  - Rapidly-exploring Random Trees (RRTs)
  - Smoothing

Optimization-based Motion Planning

- Reactive control
  - Potential-based methods (Khatib ‘86)
- Optimize over entire trajectory
  - Elastic bands (Quinlan and Khatib ‘93)
  - CHOMP (Ratliff et al. ‘09) and variants (STOMP, ITOMP)
  - ***Trajopt (Schulman, et al 2013)***

Solve by Nonlinear Optimization for Control?

- Could try by, for example, following formulation:

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

$\mathcal{X}_t$ can encode obstacles

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

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

Trajectory Optimization

$$\min_{\theta_{1:T}} \quad \sum_t \|\theta_{t+1} - \theta_t\|^2 + \text{other costs}$$

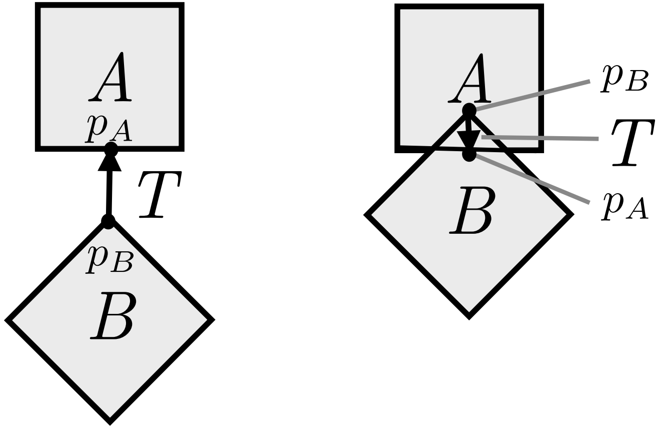
subject to $\quad \theta_0 = \text{start state}, \quad \theta_T \text{ in goal set}$

joint limits

for all robot parts, for all obstacles:
no collision ⟶ ***non-convex***

Solution method: sequential convex optimization

Collision Constraints



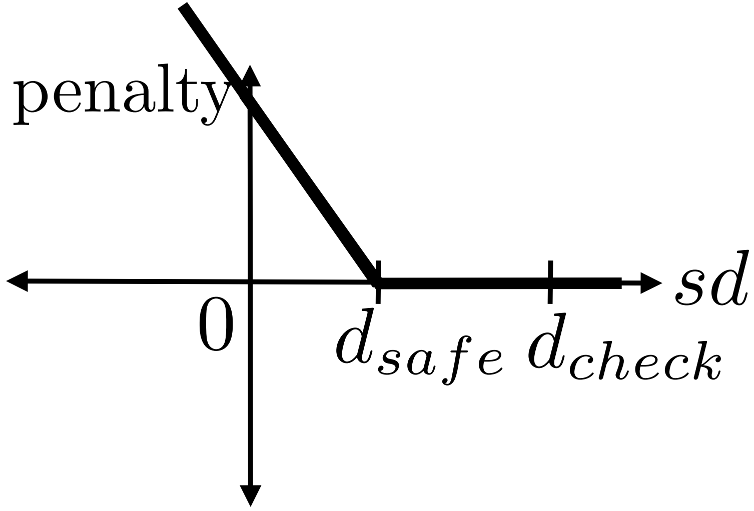
$sd > 0$ $\quad$ $sd < 0$

$$sd_{AB}(\theta) \approx \hat{n} \cdot (p_B - p_A(\theta))$$

$$\approx sd_{AB}(\theta_0) - \hat{n}^\top J_{P_A}(\theta_0)(\theta - \theta_0)$$

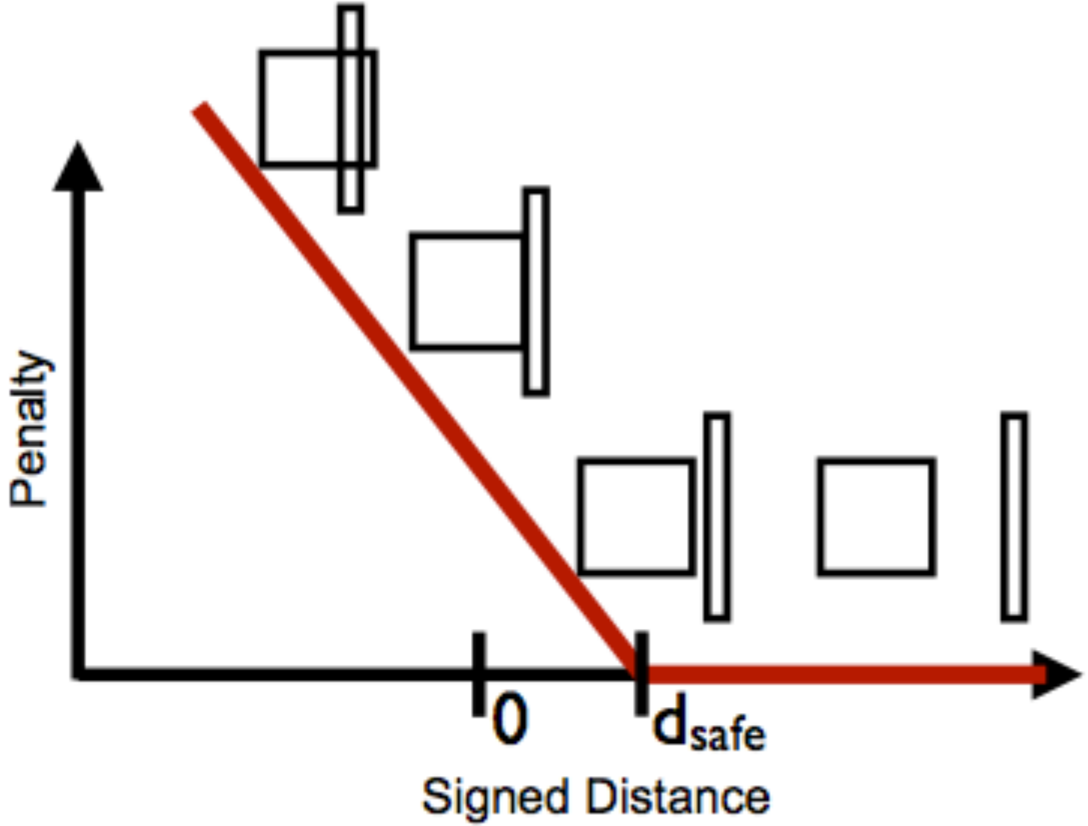
[SD from: Gilbert-Johnson-Keerthi (GJK) algorithm and Expanding Polytope Algorithm (EPA)]

Penalty for Collision Constraints

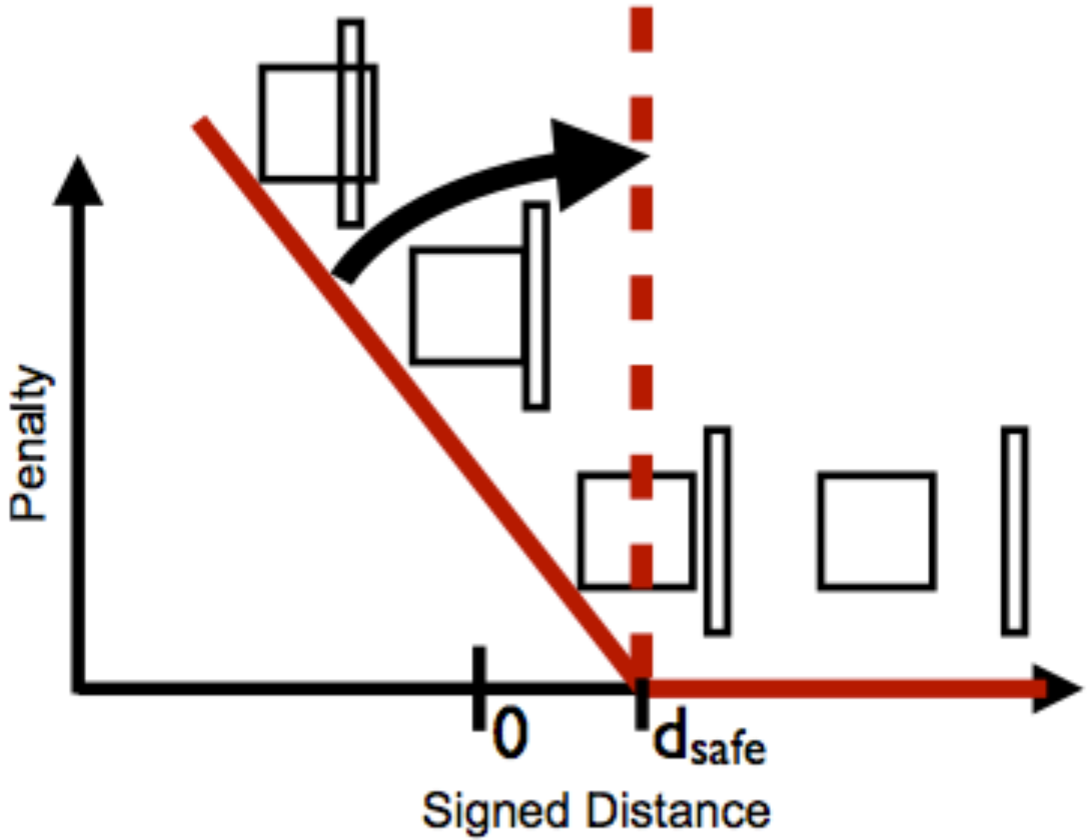


$$
\begin{aligned}
sd_{AB}(\theta) &\approx \hat{n} \cdot (p_B - p_A(\theta)) \\
&\approx sd_{AB}(\theta_0) - \hat{n}^\top J_{P_A}(\theta_0)(\theta - \theta_0)
\end{aligned}
$$

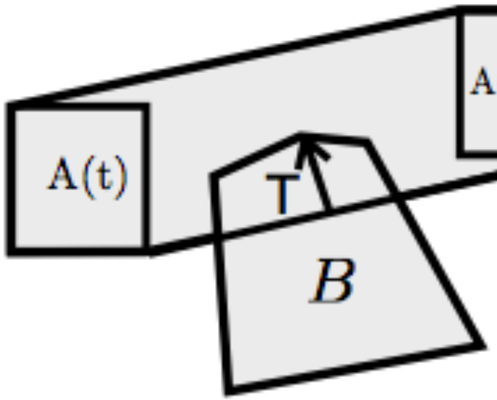
Collision Constraint as L1 Penalty



Collision Constraint as L1 Penalty



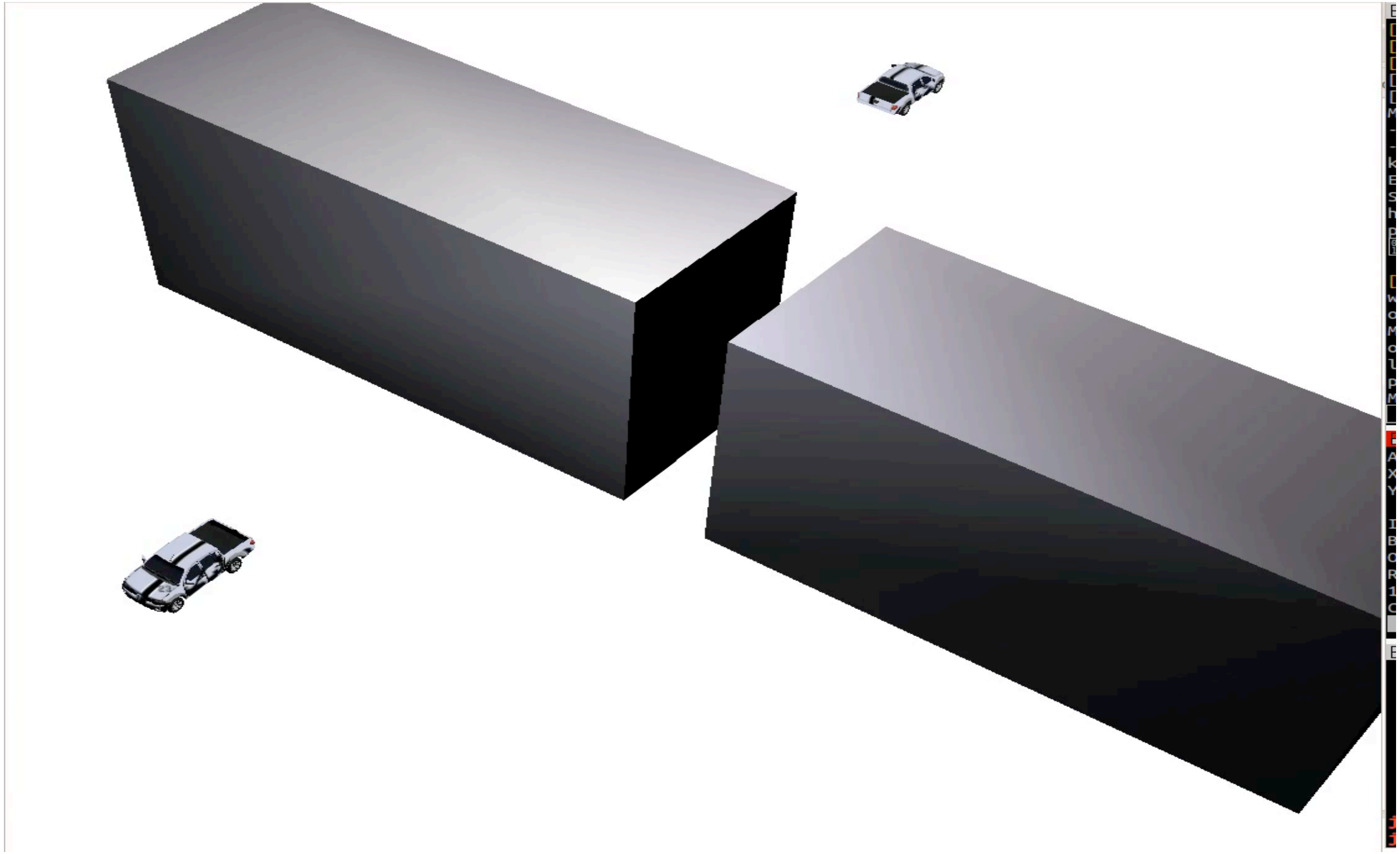
Continuous-Time Safety



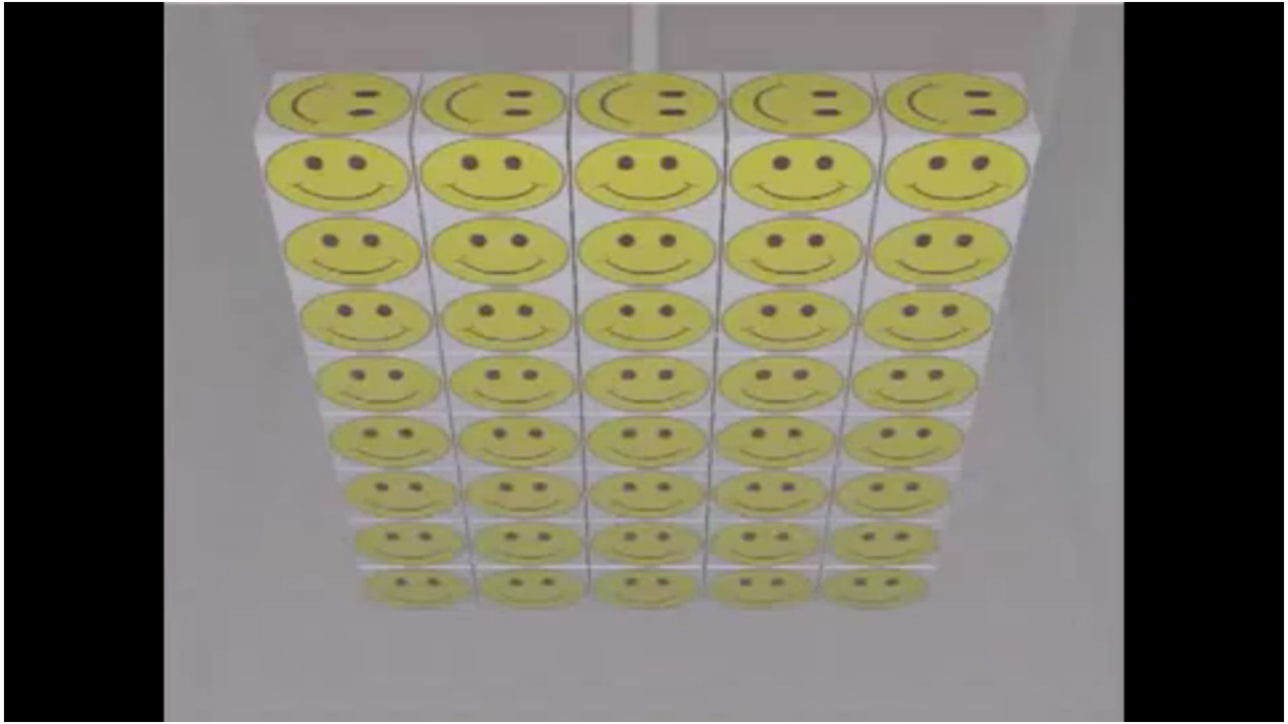
Collision check against swept-out volume

- Allows coarsely sampling trajectory
  - Overall faster
- Finds better local optima

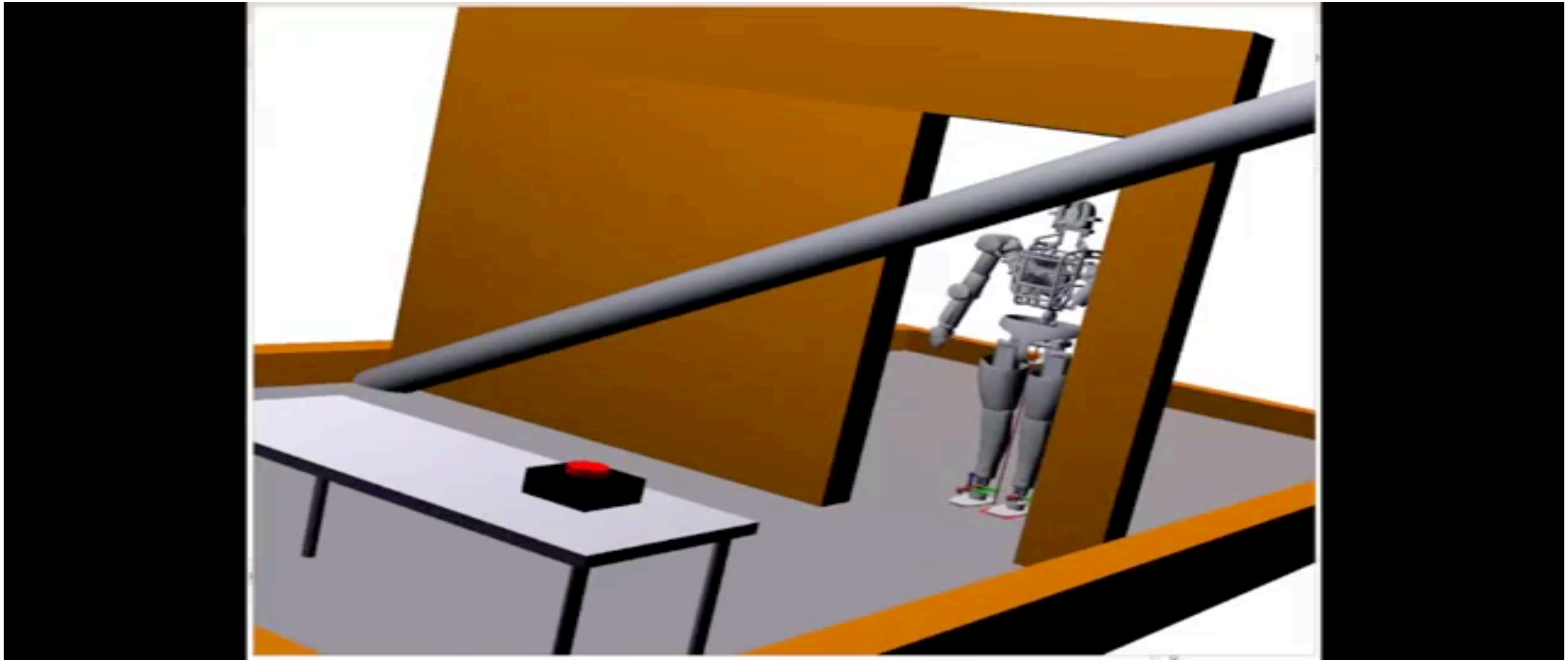
Collision-free Path for Dubin's Car



Experiments: Industrial Box Picking



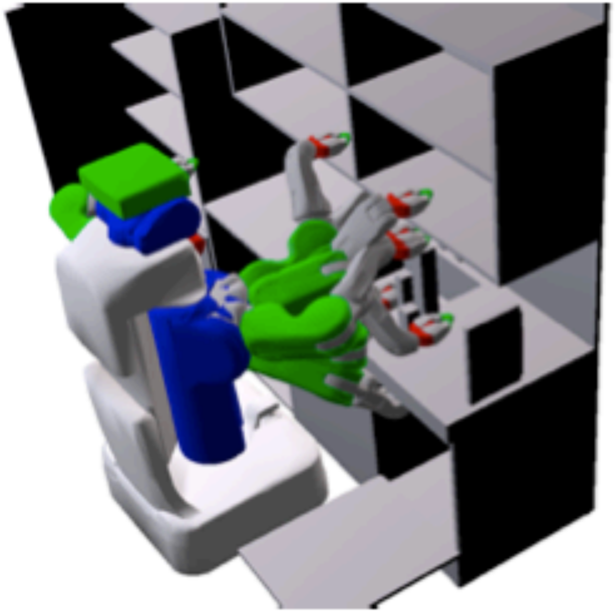
Experiments: DRC Robot



Benchmark

7 DOF (one arm)

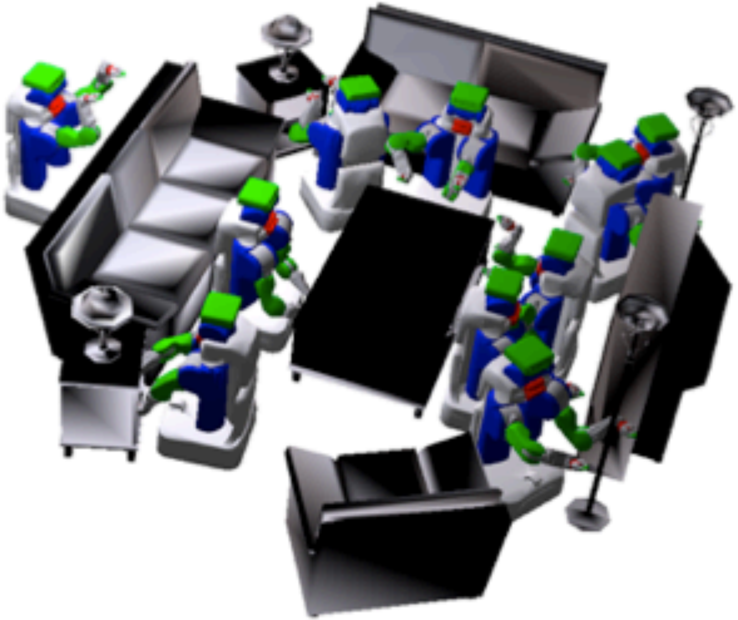
198 problems



example scene (taken from MoveIt collection)

18 DOF (two arms + base + torso)

96 problems



example scene (imported from Trimble 3d Warehouse / Google Sketchup)

Benchmark Results

Arm planning (7 DOF) 10s limit			
	Trajopt	BiRRT (*)	CHOMP
success	99%	97%	85%
time (s)	0.32	1.2	6.0
path length	1.2	1.6	2.6

Full body (18 DOF) 30s limit			
	Trajopt	BiRRT (*)	CHOMP (**)
success	84%	53%	N/A
time (s)	7.6	18	N/A
path length	1.1	1.6	N/A

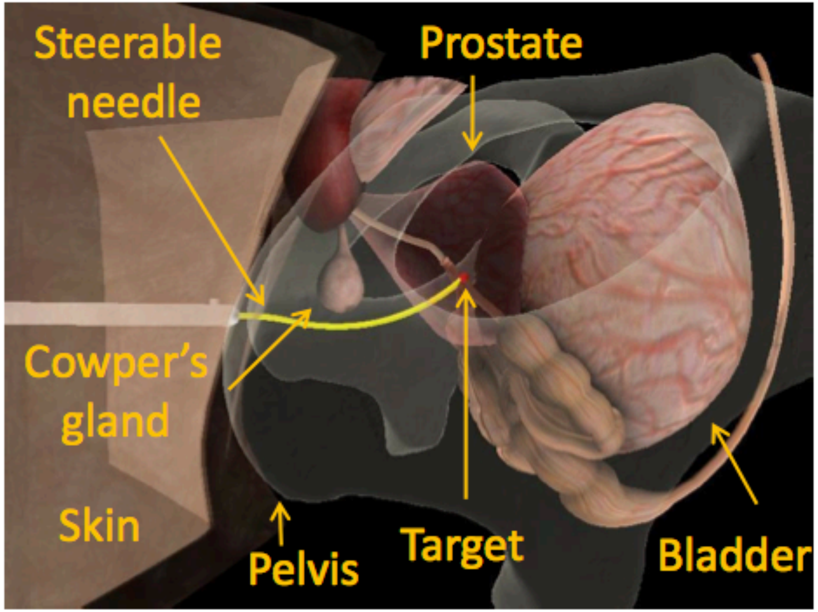
(*) Top-performing algorithm from MoveIt/OMPL
(**) Not supported in available implementation

[RSS 2013]

Experiments: PR2

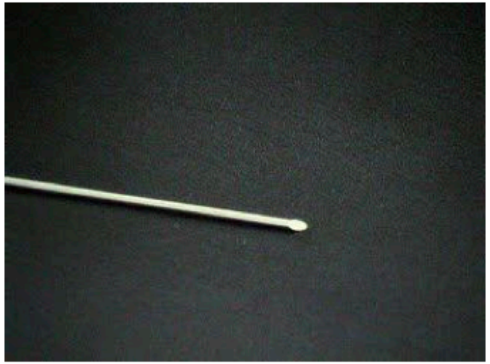
Steerable Needle

Steerable needles inside phantom tissue

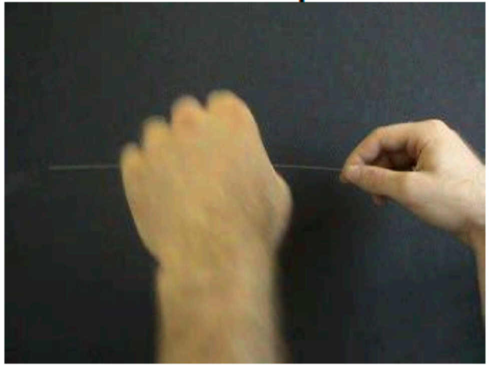


Steerable needles navigate around sensitive structures (simulated)

Steerable Needle

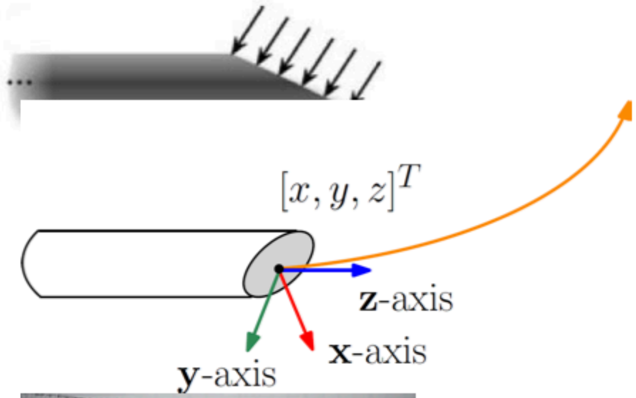


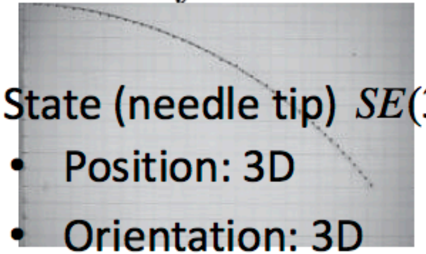
Bevel-tip



Highly flexible

[Webster, Okamura, Cowan, Chirikjian, Goldberg, Alterovitz United States Patent 7,822,458. 2010]





State (needle tip) $SE(3) : \mathbb{R}^3 \times SO(3)$

- Position: 3D
- Orientation: 3D

Follows constant curvature paths

Steerable Needle: Opt Formulation

$$\min_{\bar{\mathcal{X}},\mathcal{U}} \ \alpha_\Delta \mathrm{Cost}_\Delta + \alpha_\phi \mathrm{Cost}_\phi + \alpha_\mathcal{O} \mathrm{Cost}_\mathcal{O},$$

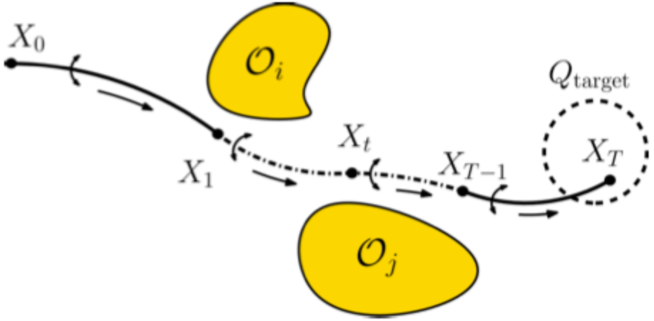
$$\text{s.t.} \quad \log((X_t \cdot \exp(\mathbf{w}_t^\wedge) \cdot \exp(\mathbf{v}_t^\wedge))^{-1} \cdot X_{t+1})^\vee = \mathbf{0}_6,$$

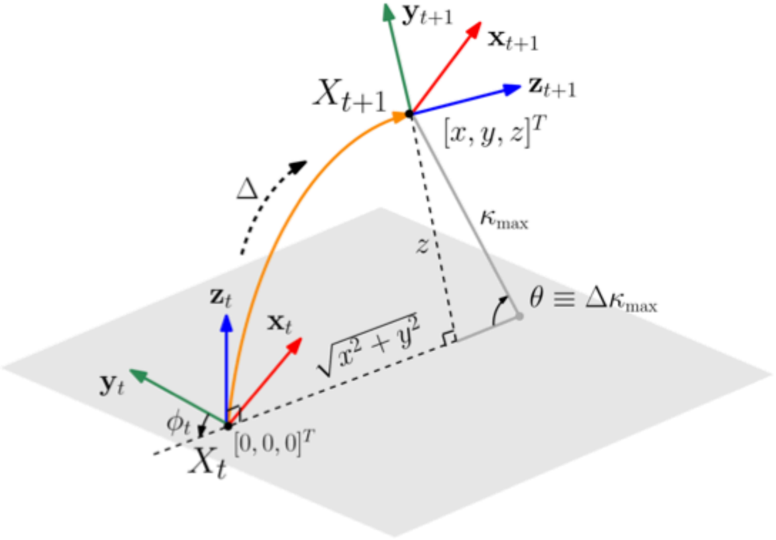
$$\mathrm{sd}(X_t, X_{t+1}, \mathcal{O}_i) \geq d_{\mathrm{safe}} + d_{\mathrm{arc}},$$

$$X_0 \in Q_{\mathrm{entry}}, \ \ X_T \in Q_{\mathrm{target}},$$

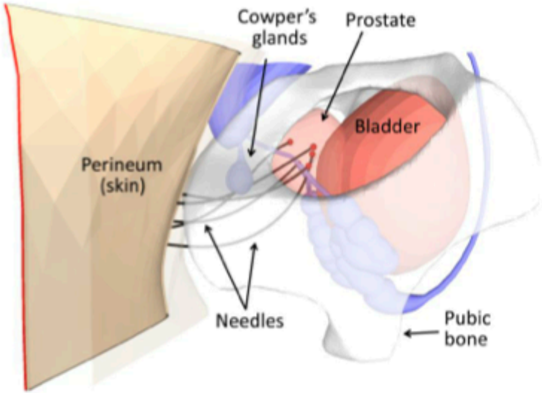
$$-\pi \leq \phi_t \leq \pi,$$

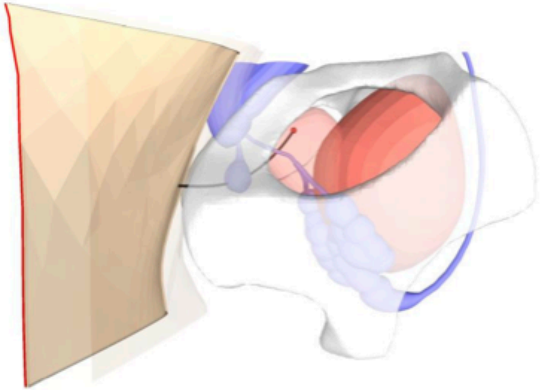
$$\kappa_t = \kappa_{\max}$$



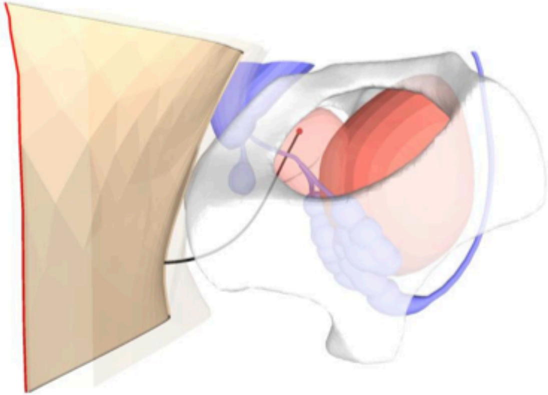


Steerable Needle: Plans





(a) Smaller clearance from obstacles (Cowper's glands) with $\alpha_{\mathcal{O}} = 1$.



(b) Larger clearance from obstacles with $\alpha_{\mathcal{O}} = 10$.

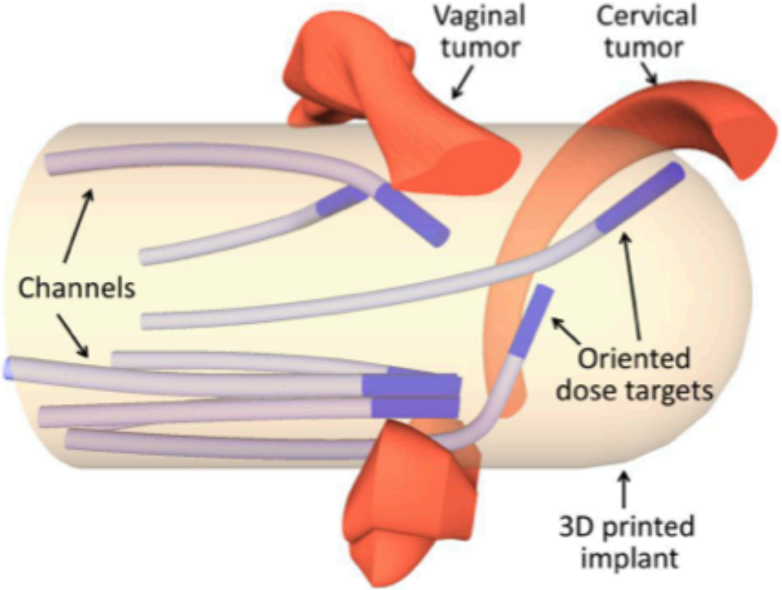
Steerable Needle: Results

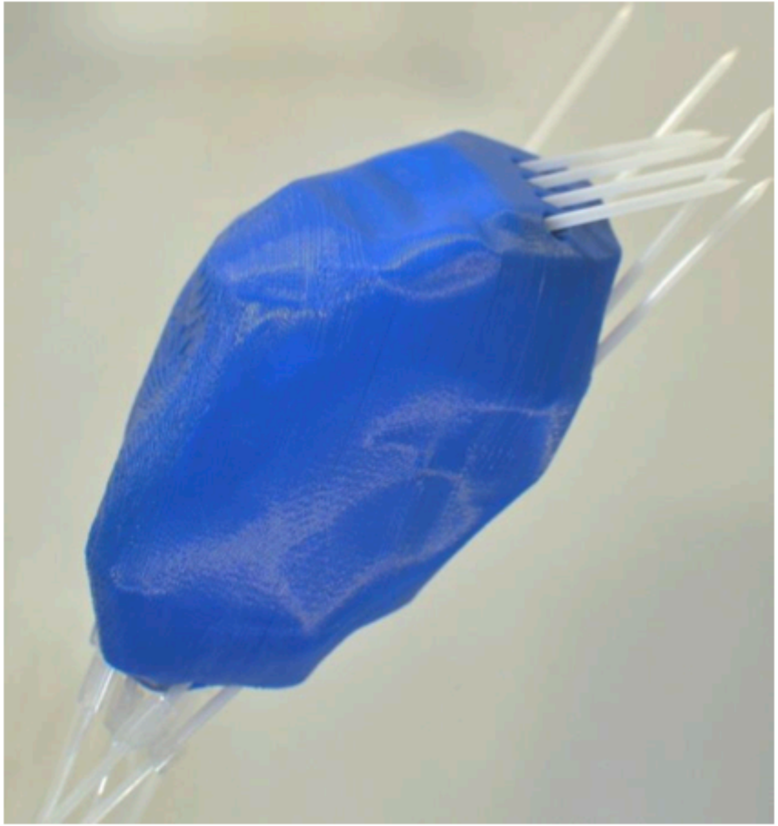
	RRT	collocation $\alpha_{\mathcal{O}} = 1$	shooting $\alpha_{\mathcal{O}} = 1$	collocation $\alpha_{\mathcal{O}} = 10$	shooting $\alpha_{\mathcal{O}} = 10$
solved%	67.3%	76.0%	80.3%	79.0%	89.5%
time (s)	9.8 ± 8.1	1.8 ± 1.2	1.6 ± 1.7	1.9 ± 1.3	1.8 ± 1.7
path length	11.1 ± 1.5	11.3 ± 1.4	11.6 ± 1.7	11.9 ± 1.7	13.1 ± 2.3
twist cost	34.9 ± 10.0	1.4 ± 1.4	1.0 ± 1.0	1.6 ± 1.6	1.0 ± 1.0
clearance	0.5 ± 0.4	0.7 ± 0.5	0.5 ± 0.3	1.3 ± 0.4	1.2 ± 0.5

Performance of our approach on the single needle planning case.

Why is minimizing twist important?

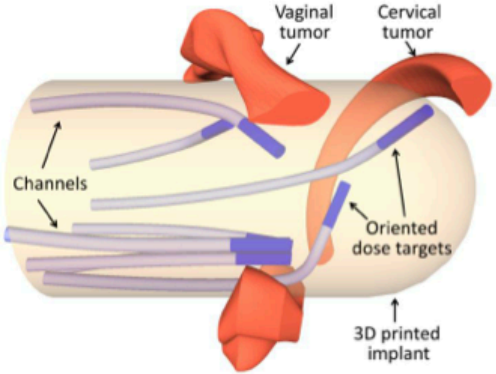
Channel Layout (Brachytherapy Implants)





Channel Layout: Opt Formulation

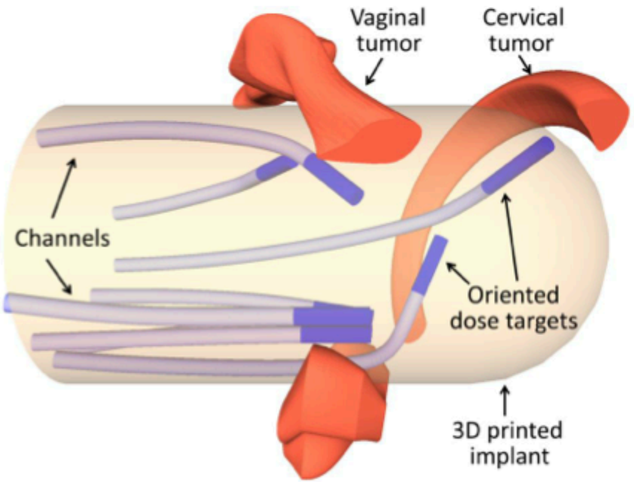
$$
\begin{aligned}
\min_{\bar{\mathcal{X}},\mathcal{U}} \quad & \alpha_{\Delta}\mathrm{Cost}_{\Delta} + \alpha_{\phi}\mathrm{Cost}_{\phi} + \alpha_{\mathcal{O}}\mathrm{Cost}_{\mathcal{O}}, \\
\text{s.t.} \quad & \log((X_t \cdot \exp(\mathbf{w}_t^{\wedge}) \cdot \exp(\mathbf{v}_t^{\wedge}))^{-1} \cdot X_{t+1})^{\vee} = \mathbf{0}_6, \\
& \mathrm{sd}(X_t, X_{t+1}, \mathcal{O}_i) \geq d_{\mathrm{safe}} + d_{\mathrm{arc}}, \\
& X_0 \in Q_{\mathrm{entry}}, \;\; X_T \in Q_{\mathrm{target}}, \\
& -\pi \leq \phi_t \leq \pi, \\
& 0 \leq \kappa_t \leq \kappa_{\max}, \\
& \Delta \sum_{t=0}^{T-1} \kappa_t \leq c_{\max} \quad \text{for channel planning,}
\end{aligned}
$$



Channel Layout: Results

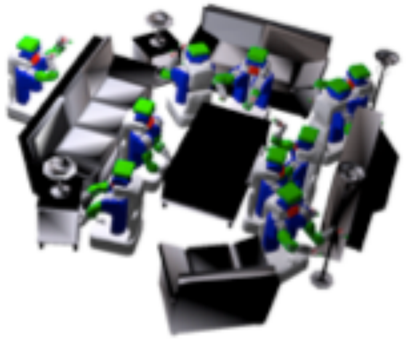
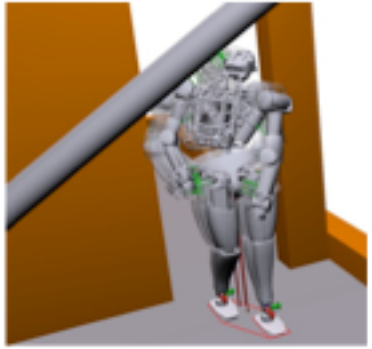
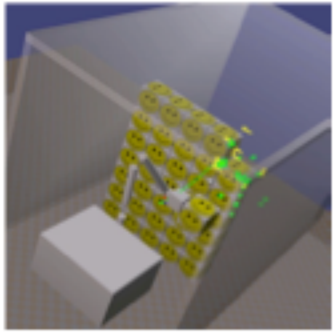
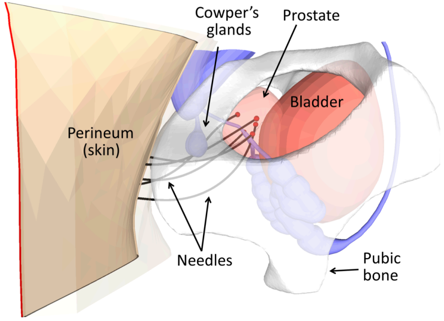
	RRT	backward shooting
solved%	74.0%	98.0%
time (s)	30.8 ± 17.9	27.7 ± 9.8
path length	41.3 ± 0.3	38.9 ± 0.1
twist cost	65.5 ± 8.4	4.1 ± 1.1

Performance of our approach on the channel layout planning

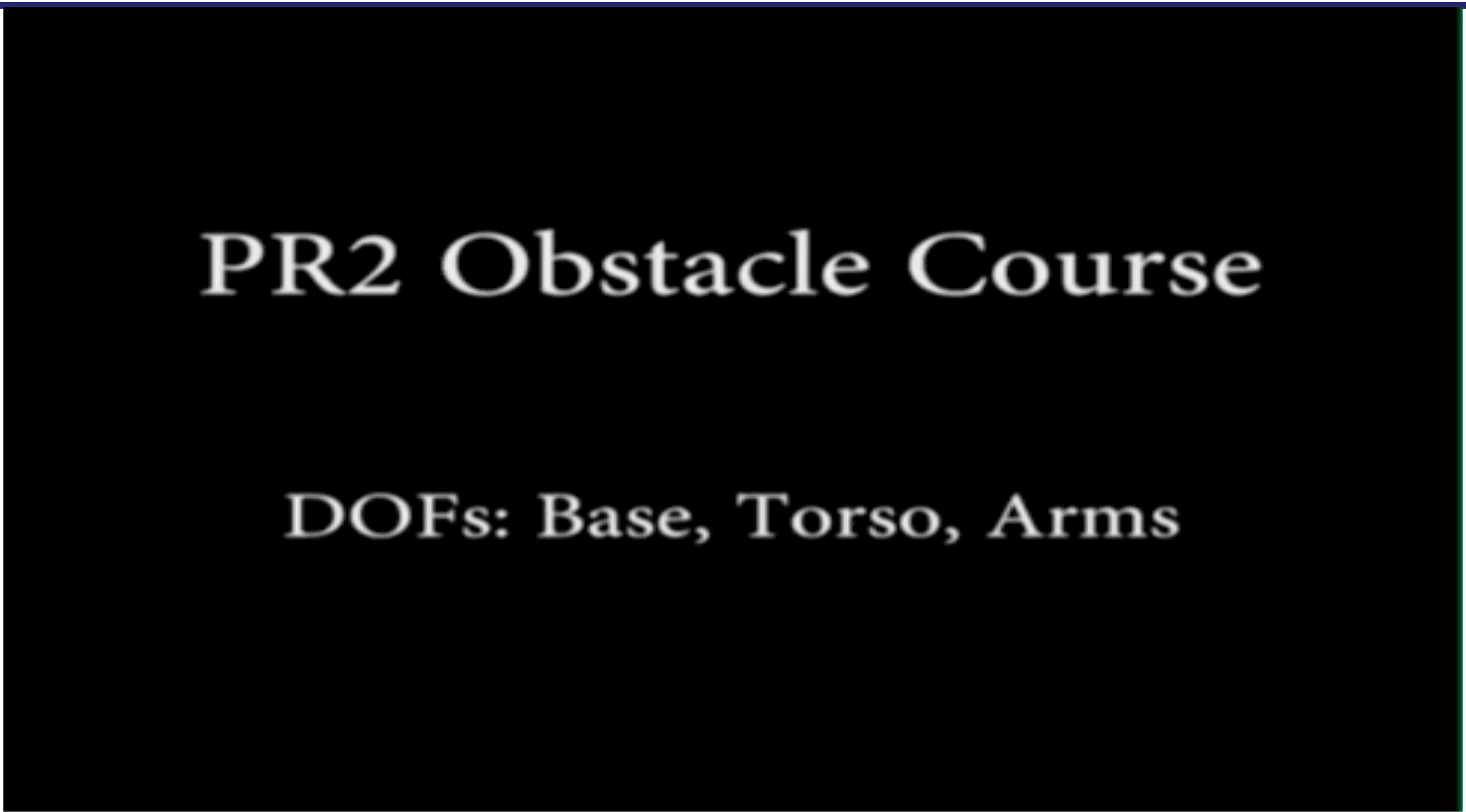


Try It Yourself

- Code and docs: rll.berkeley.edu/trajopt
- Benchmark: github.com/joschu/planning_benchmark

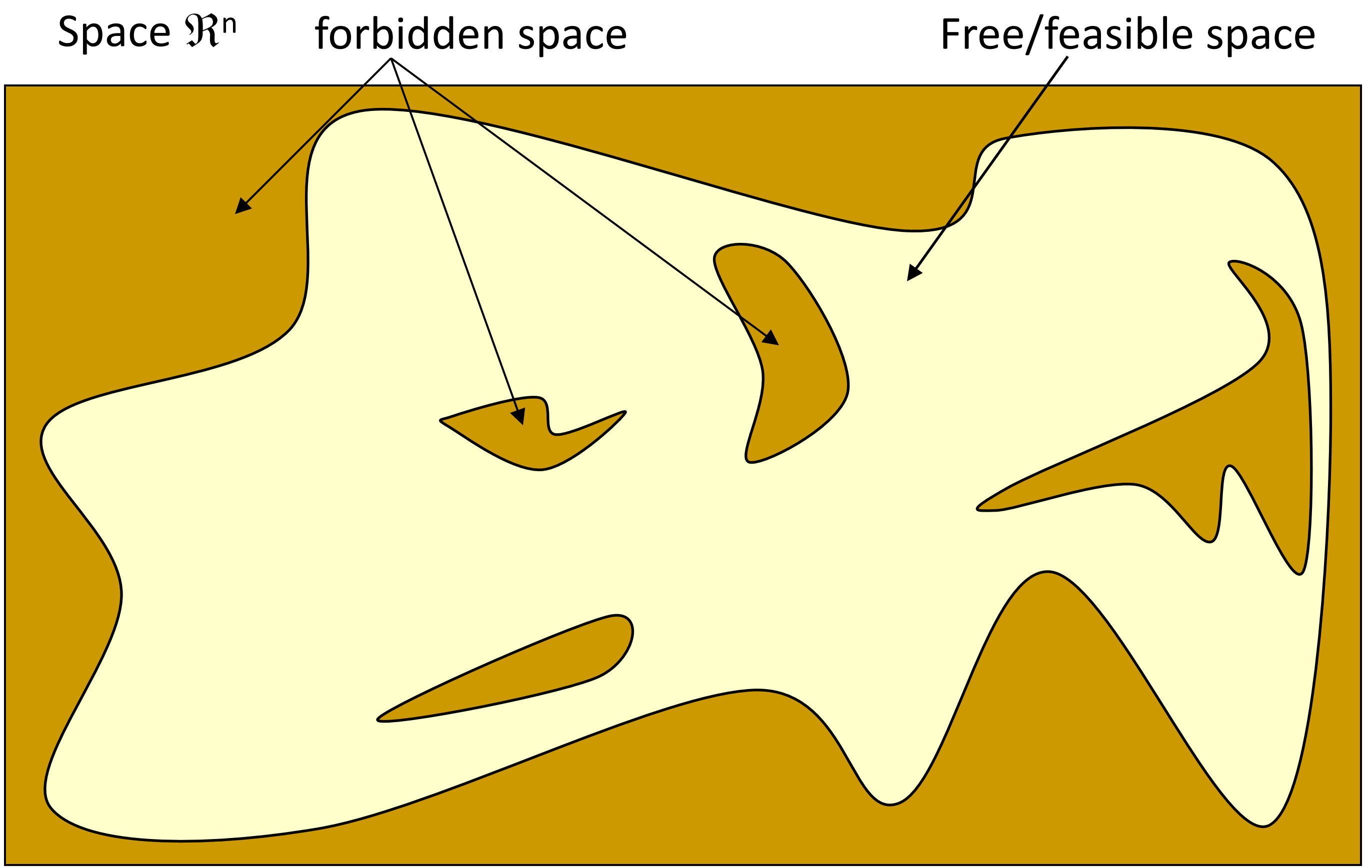
Experiments: PR2



Motion Planning: Outline

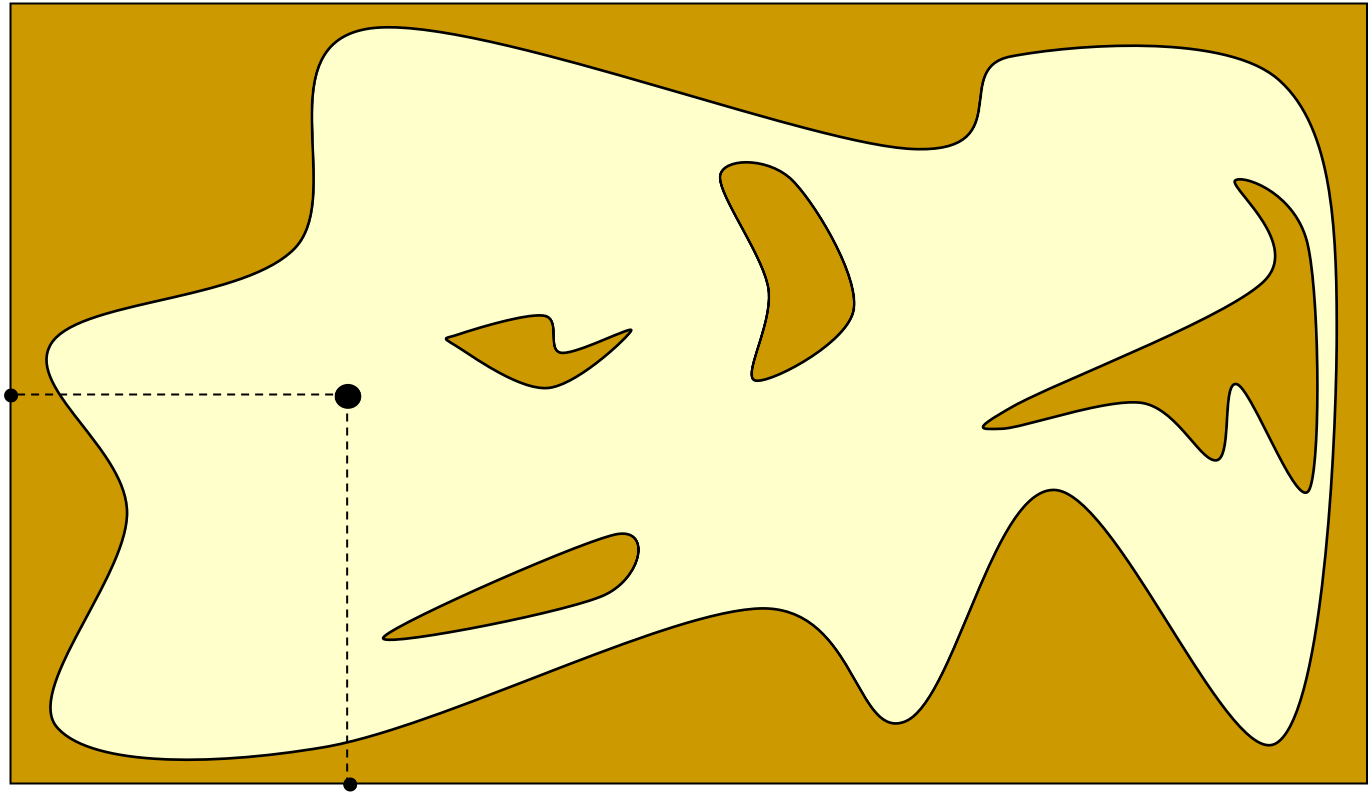
- Configuration Space
- Optimization-based Motion Planning
- ***Sampling-based Motion Planning***
  - ***Probabilistic Roadmap***
  - Rapidly-exploring Random Trees (RRTs)
  - Smoothing

Probabilistic Roadmap (PRM)



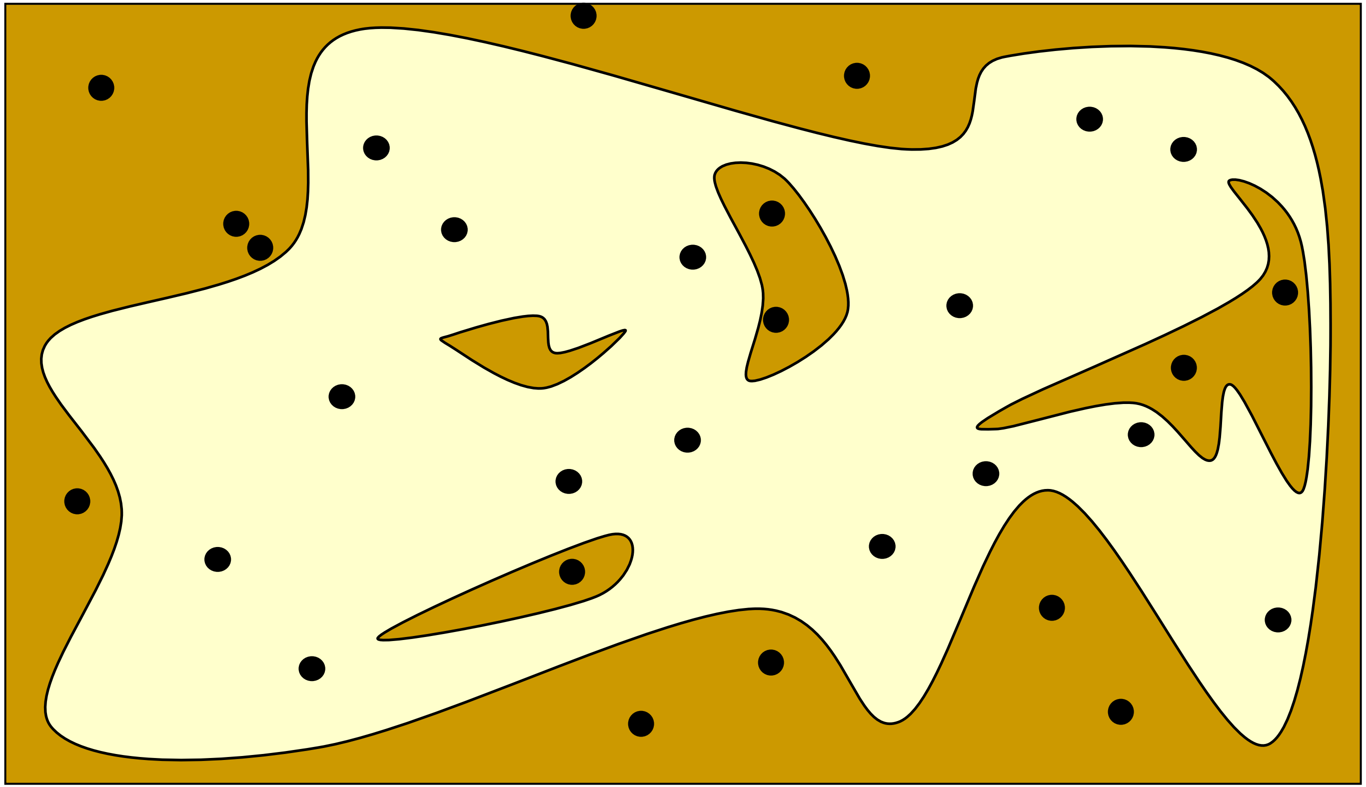
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



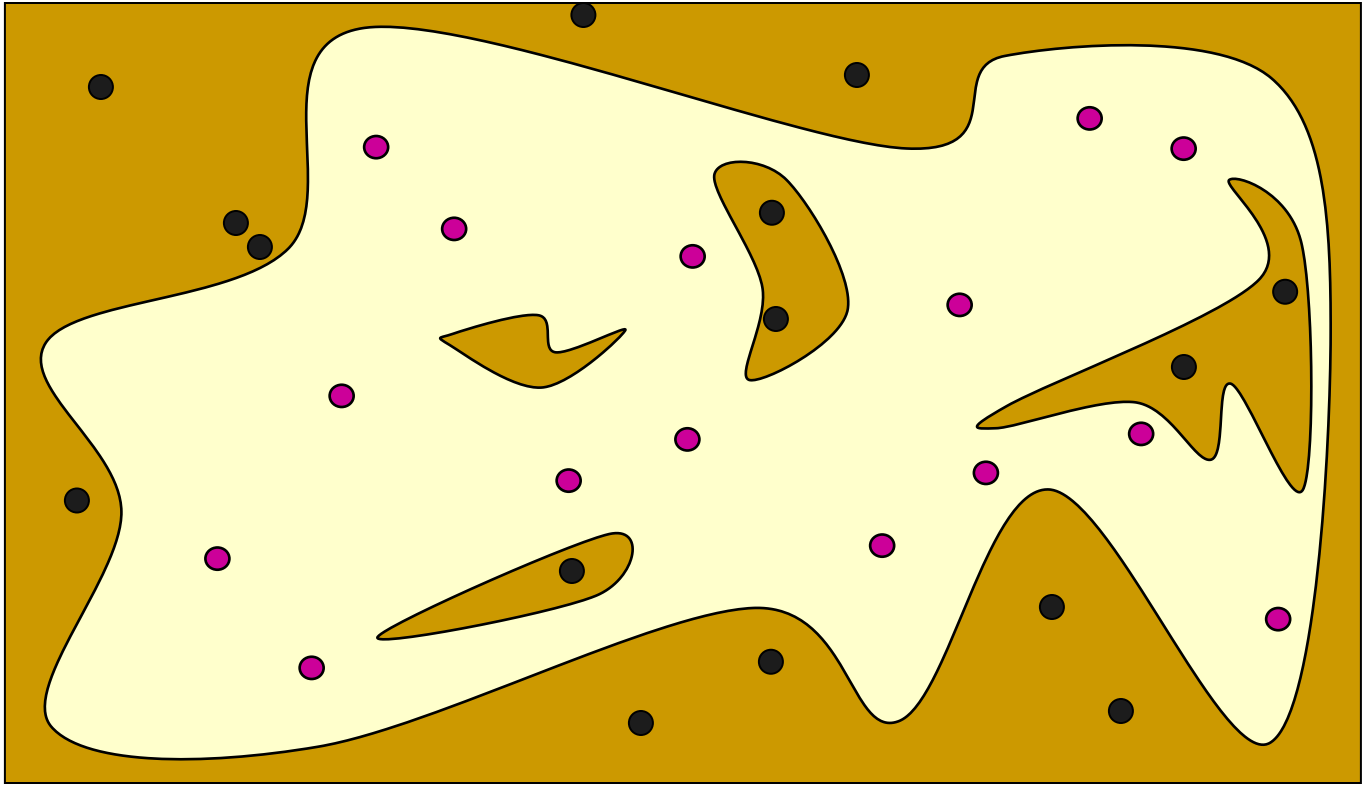
Probabilistic Roadmap (PRM)

Configurations are sampled by picking coordinates at random



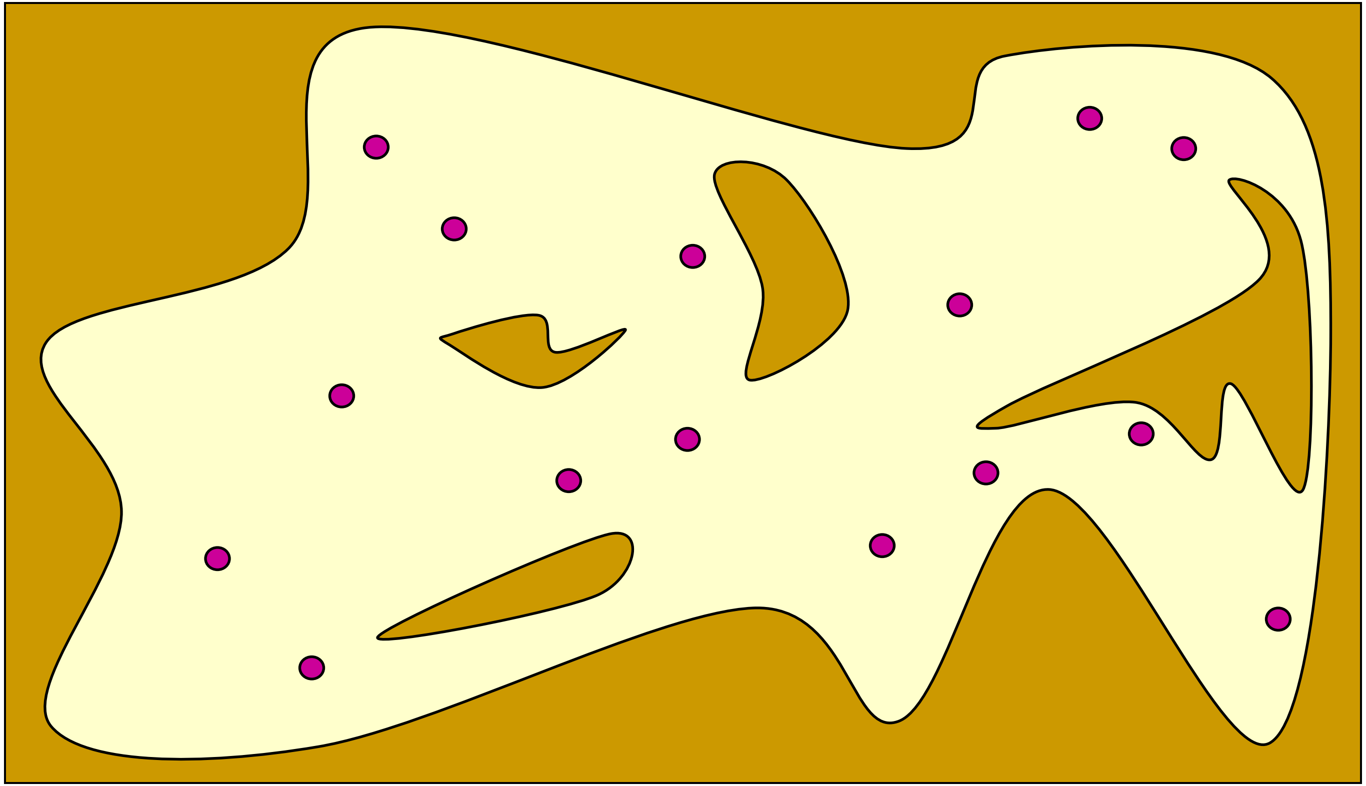
Probabilistic Roadmap (PRM)

Sampled configurations are tested for collision



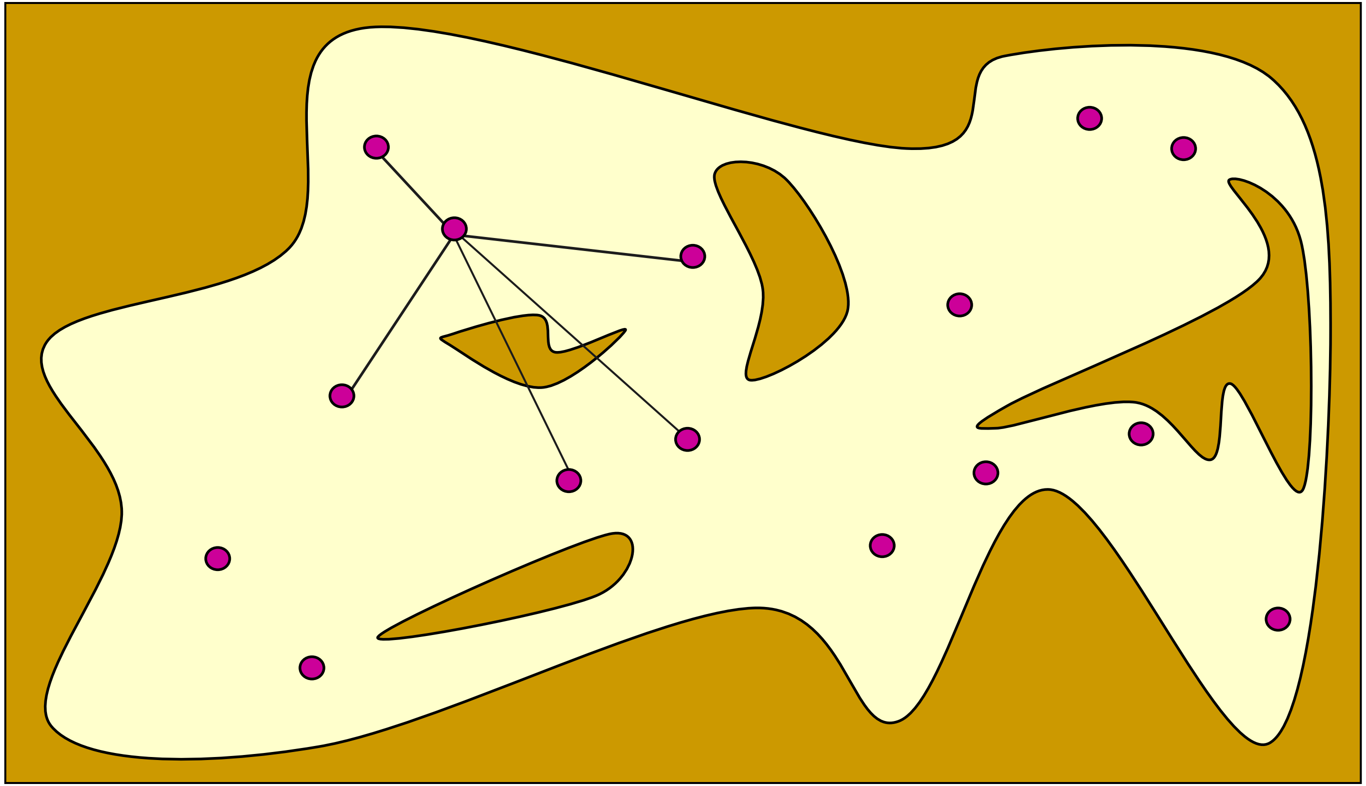
Probabilistic Roadmap (PRM)

The collision-free configurations are retained as milestones



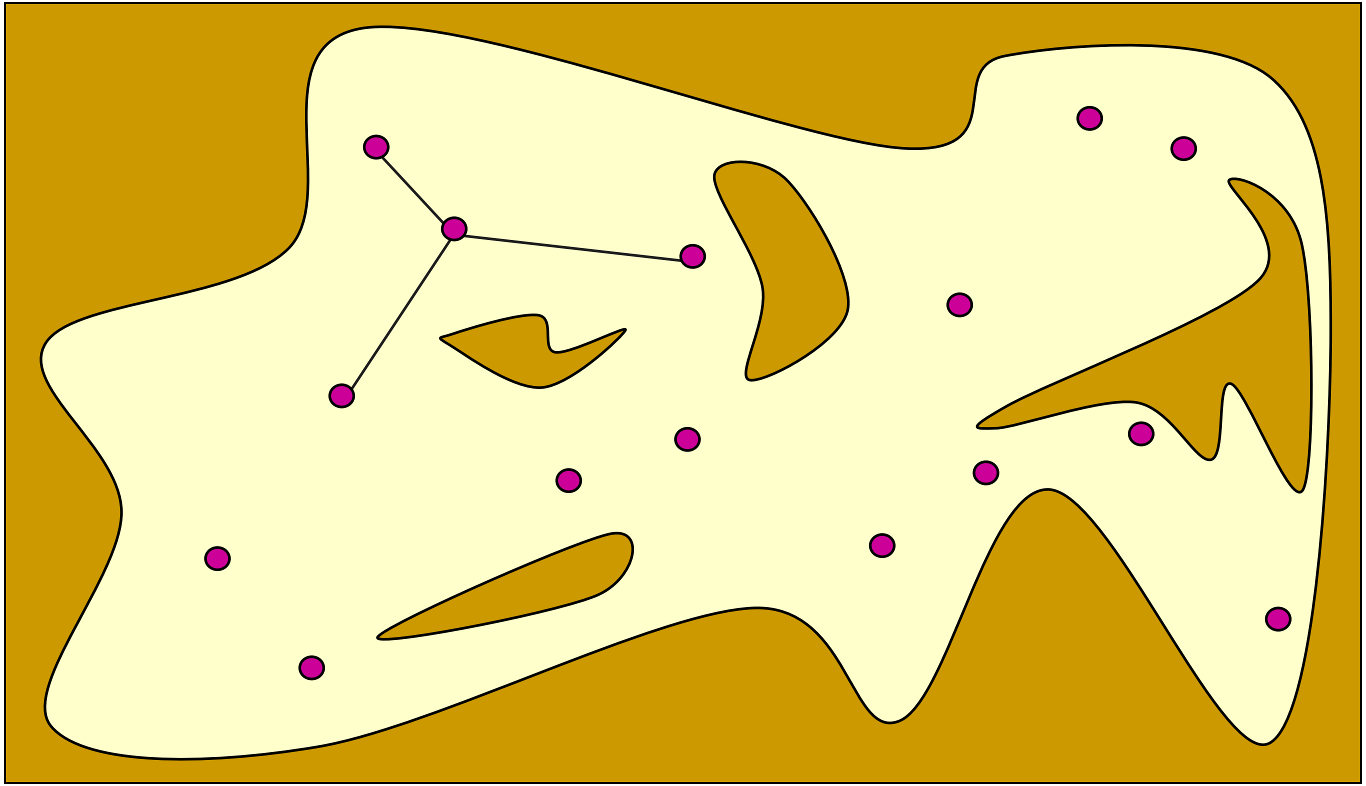
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



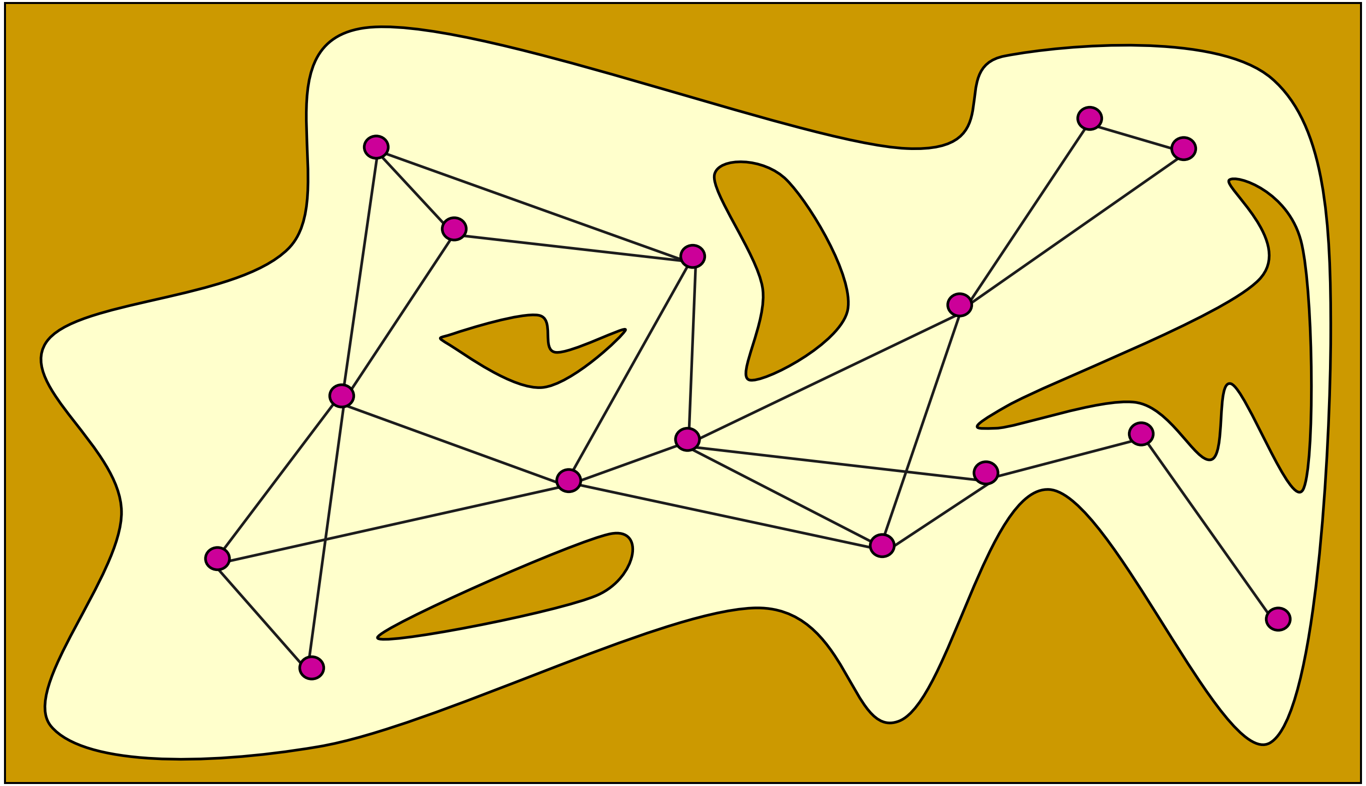
Probabilistic Roadmap (PRM)

Each milestone is linked by straight paths to its nearest neighbors



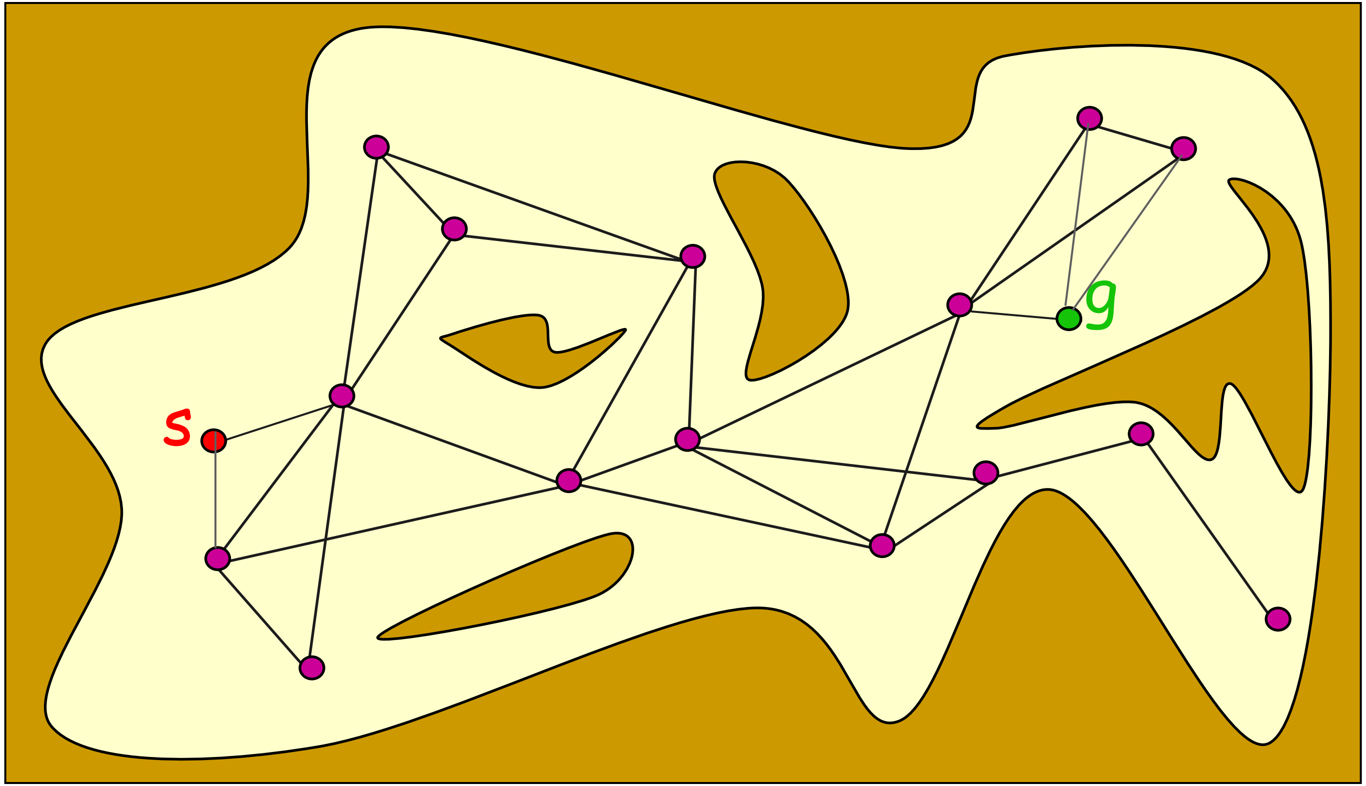
Probabilistic Roadmap (PRM)

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



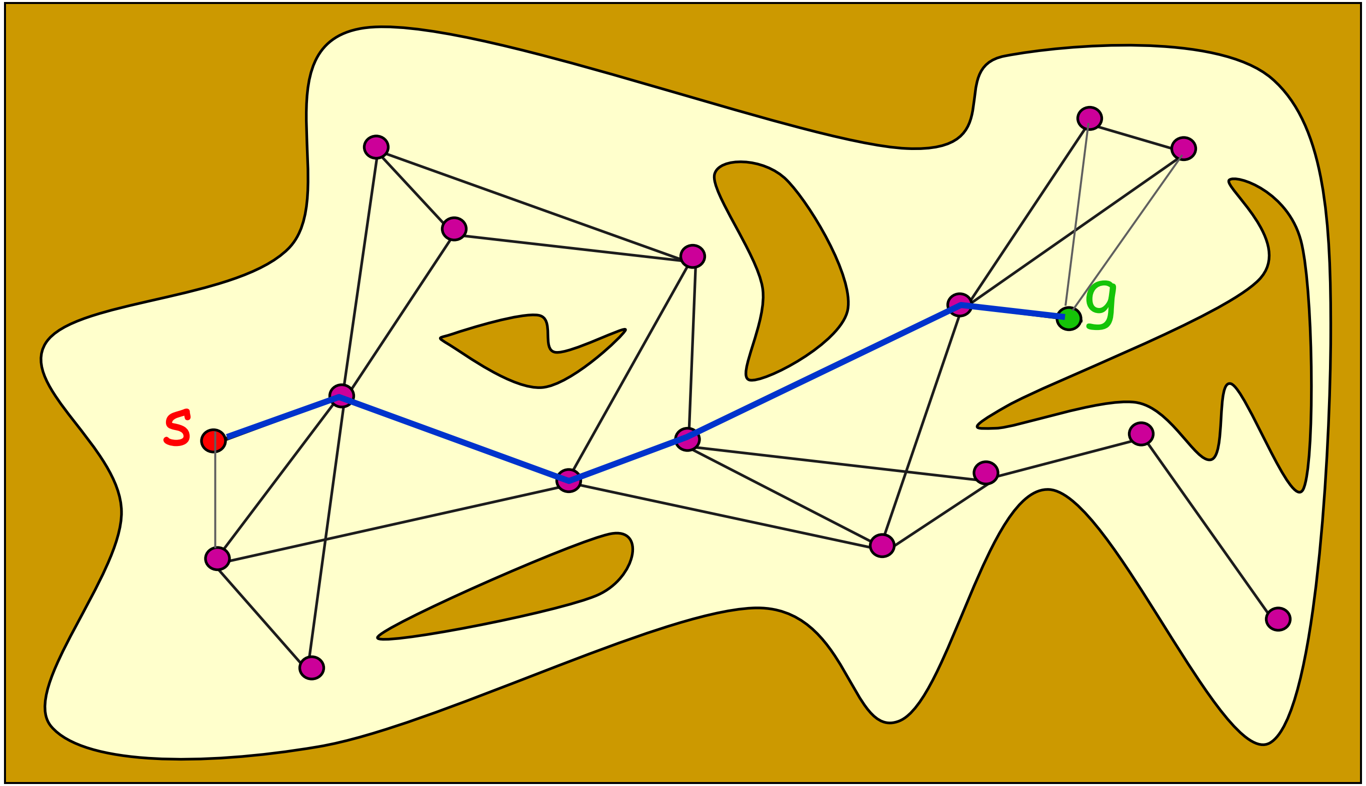
Probabilistic Roadmap (PRM)

The start and goal configurations are included as milestones



Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g

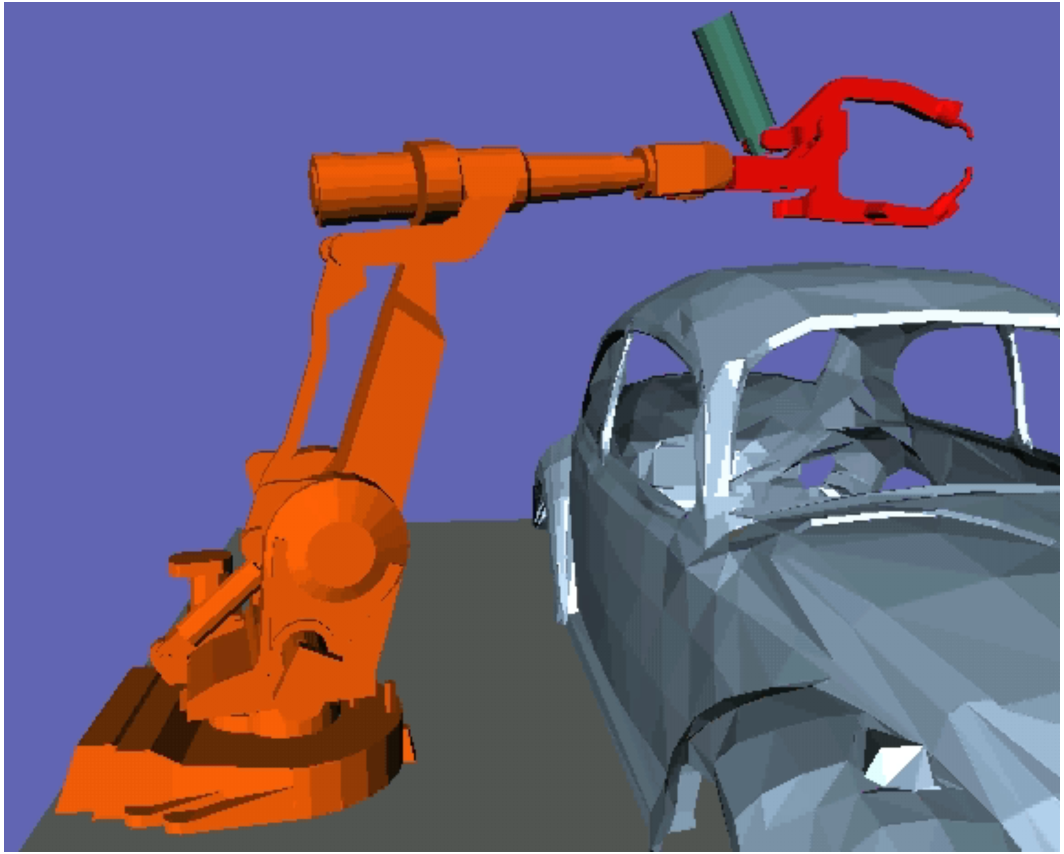


Probabilistic Roadmap

- 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

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

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

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 entire state space, which might be unnecessarily expensive when what's needed is connecting specific start and goal

Motion Planning: Outline

- Configuration Space
- Optimization-based Motion Planning
- ***Sampling-based Motion Planning***
  - Probabilistic Roadmap
  - ***Rapidly-exploring Random Trees (RRTs)***
  - Smoothing

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

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);

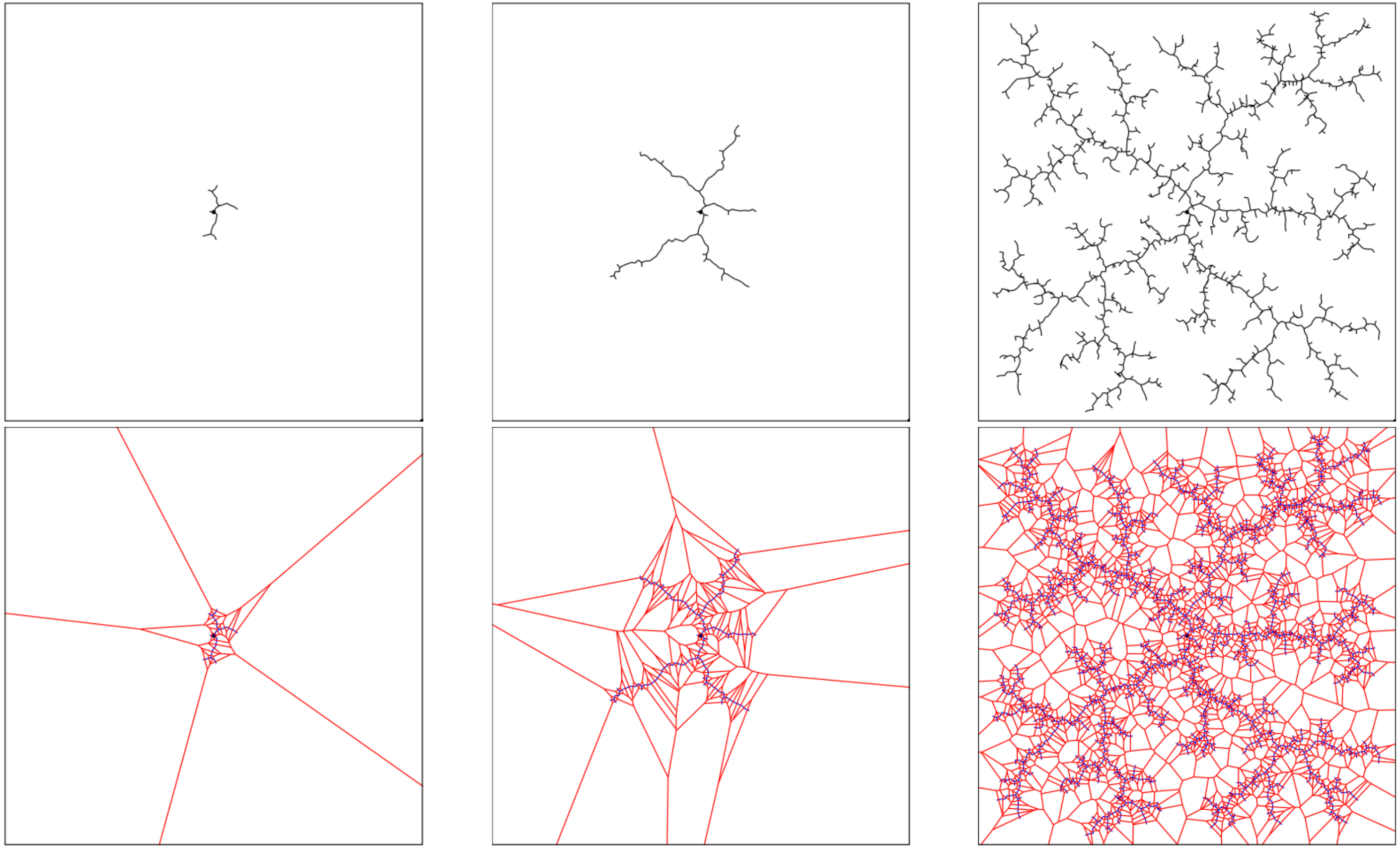
Return $\mathcal{T}$

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
SELECT_INPUT(): often a few inputs are sampled, and one that results in x_new closest to x_rand is retained; sometimes optimization is run to find the best input

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

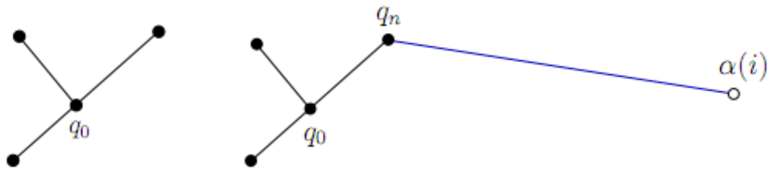
RRT Practicalities

- 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

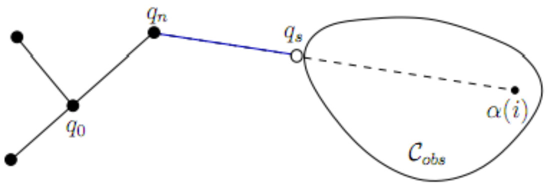
- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

RRT Extension

- No obstacles, holonomic:



- With obstacles, holonomic:



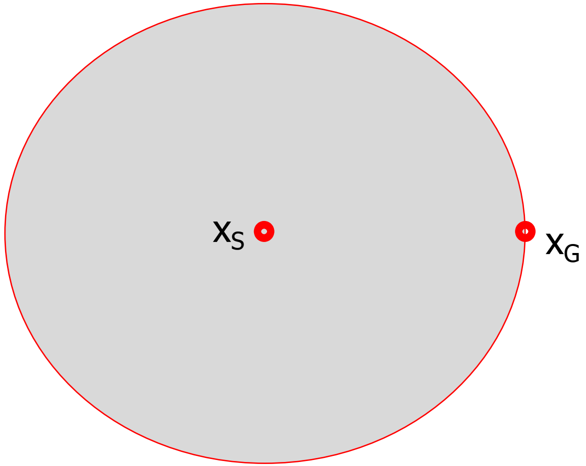
- Non-holonomic: approximately solve two-point boundary value problem (often rough approximation: pick best of a few random control sequences)

Growing RRT

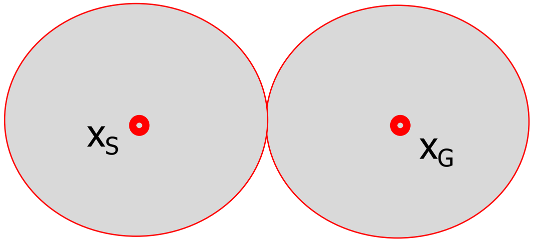
Demo: http://en.wikipedia.org/wiki/File:Rapidly-exploring_Random_Tree_(RRT)_500x373.gif

Bi-directional RRT

- Volume swept out by unidirectional RRT:



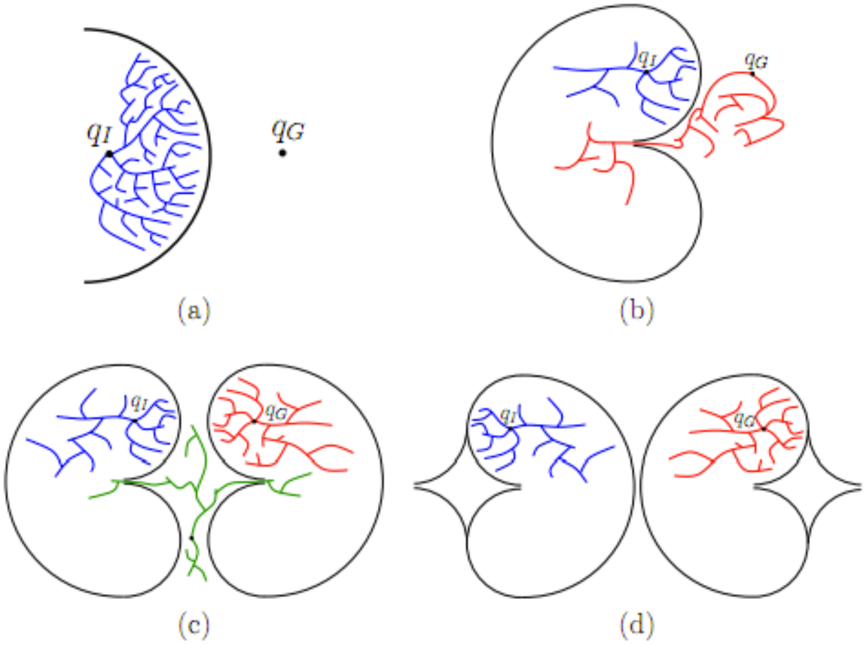
- Volume swept out by bi-directional RRT:



- Difference more and more pronounced as dimensionality increases

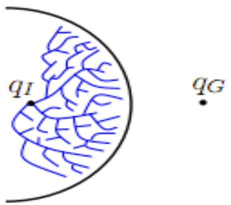
Multi-directional RRT

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



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

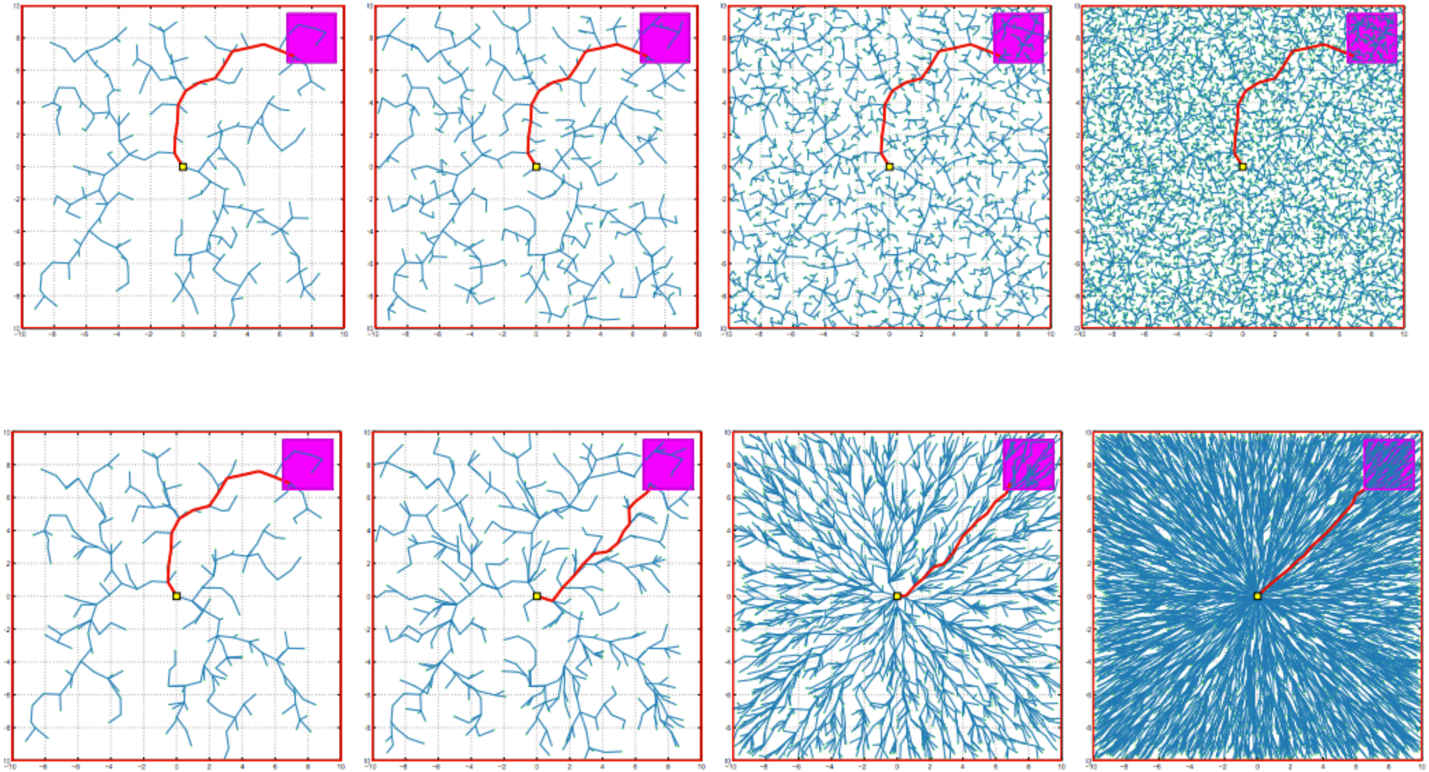
RRT*

- 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

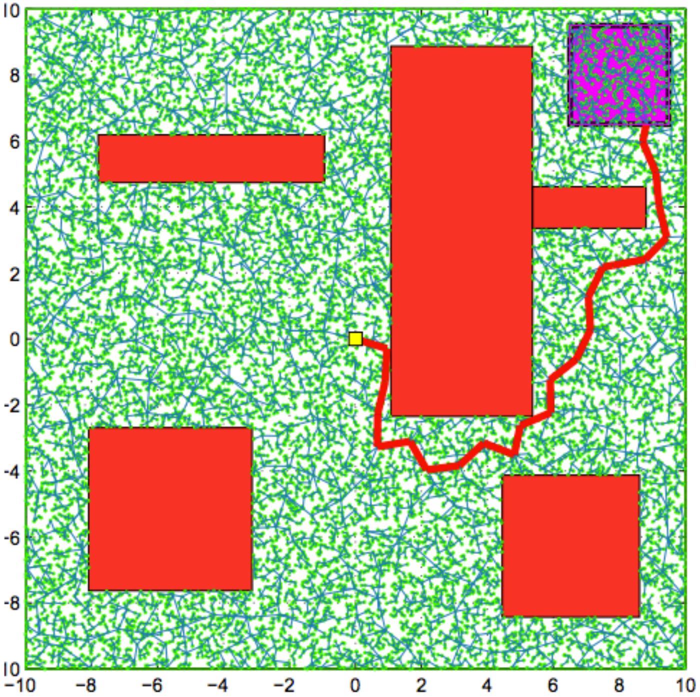
RRT*



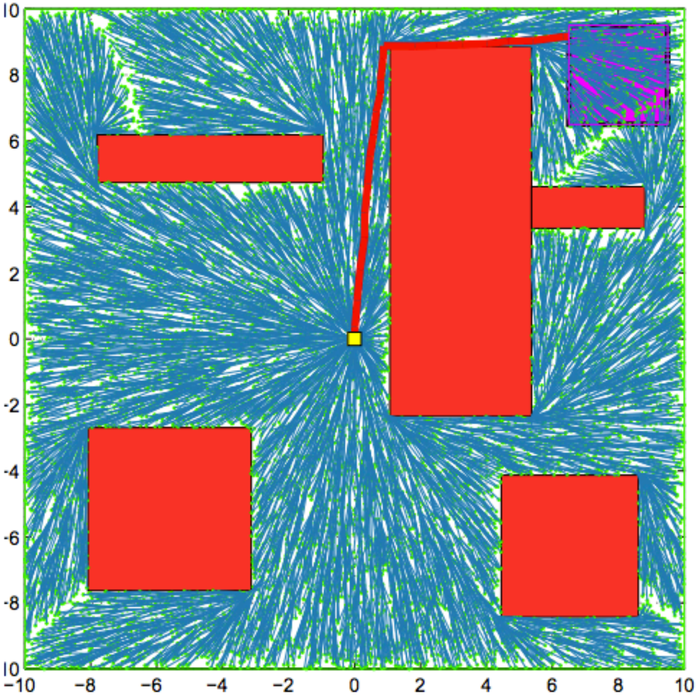
Source: Karaman and Frazzoli

RRT*

RRT



RRT*



Source: Karaman and Frazzoli

RRT* Kinodynamics

- Requires 2-point boundary value problem solution for optimality
- Li, Littlefield, Bekris 2014 proved that you can get asymptotic optimality from random sampling control trajectories in an RRT like fashion (Naïve Random Tree), without solving a 2-point boundary value problem
- They also show that using pruning can make this efficient in an algorithm called SST*

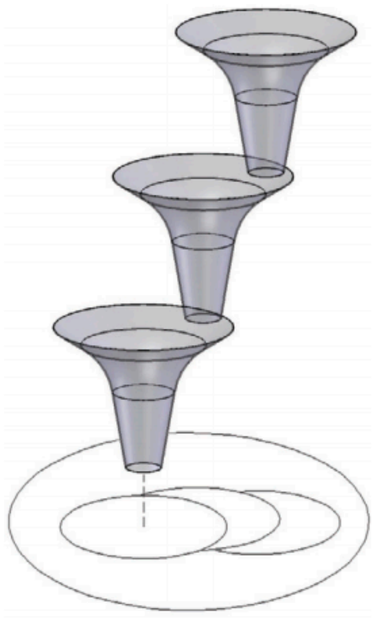
PRM* Probabilistic Bounds

- Dobson, Moustakides, Bekris 2014
- Gave finite time bounds for the current best path I_n being within a certain threshold of the optimal cost length I_n^* for a fixed delta of the form:

$$\mathbb{P}(|I_n - I^\star_{\epsilon_n}| \leq \delta \cdot I^\star_{\epsilon_n}) \leq \mathbb{P}_{success}$$

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



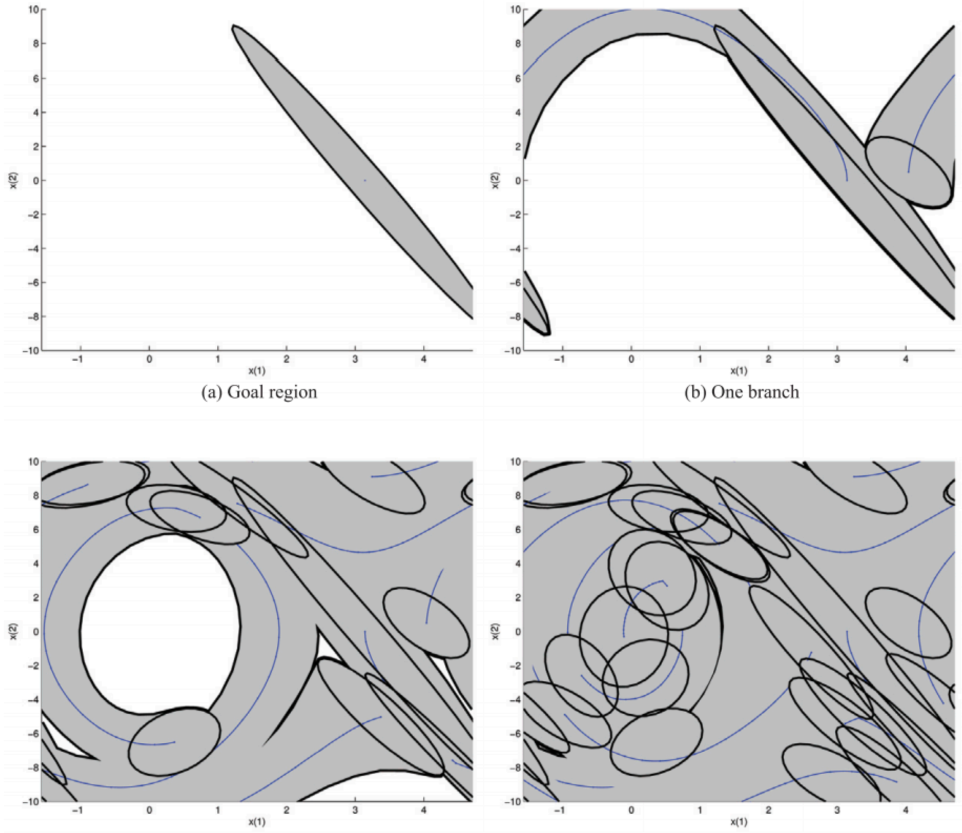
LQR-trees (Tedrake)

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 $(\mathbf{x}_G, \mathbf{u}_G)$
$[\mathbf{K}, \mathbf{S}] \Leftarrow \text{LQR}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$
$\rho_c \Leftarrow$ level set computed as described in §3.1.1
T.init($\{\mathbf{x}_g, \mathbf{u}_g, \mathbf{S}, \mathbf{K}, \rho_c, \text{NULL}\}$)
for $k = 1$ to K **do**
  $\mathbf{x}_{\text{rand}} \Leftarrow$ random sample
  if $\mathbf{x}_{\text{rand}} \in \mathcal{C}_k$ **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

Motion Planning: Outline

- Configuration Space
- Optimization-based Motion Planning
- ***Sampling-based Motion Planning***
  - Probabilistic Roadmap
  - Rapidly-exploring Random Trees (RRTs)
  - ***Smoothing***

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 (trajopt)***
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.