CSE-571
Sampling-Based Motion Planning

Slides based on those from Pieter Abbeel, Zoe McCarthy
Many images from Lavalle, Planning Algorithms

1

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):

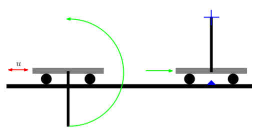
$$\begin{aligned} \min_{u,x} \quad & \|u\| \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \quad \forall t \\ & u_t \in \mathcal{U}_t \\ & x_t \in \mathcal{X}_t \\ & x_0 = x_S \\ & X_T = x_G \end{aligned}$$

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

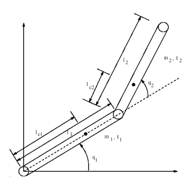
2

Examples


Helicopter path planning

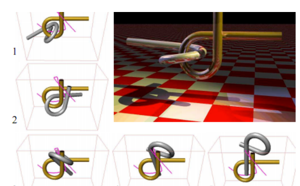


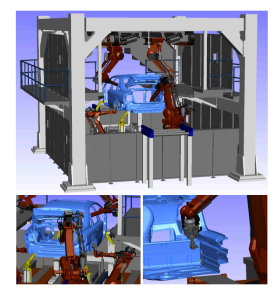
Cartpole swing-up


Acrobot

3

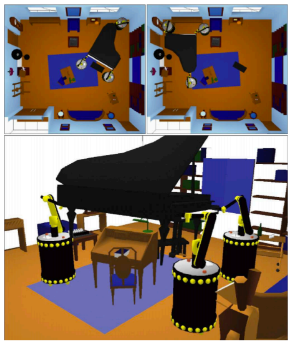
Examples

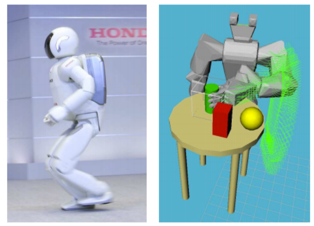




4

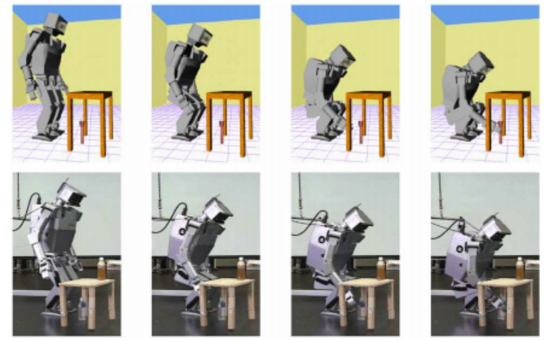
Examples





5

Examples



6

Motion Planning: Outline

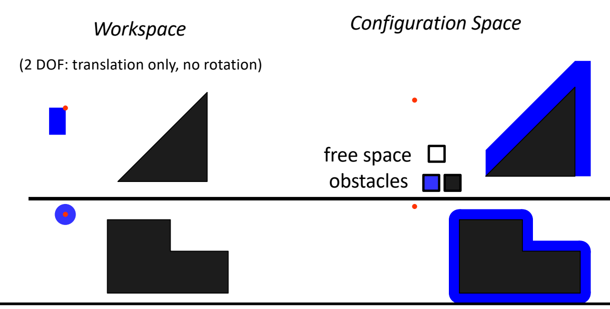
- Configuration Space
- Probabilistic Roadmap
- Rapidly-exploring Random Trees (RRTs)
- Extensions
- Smoothing

7

Configuration Space (C-Space)

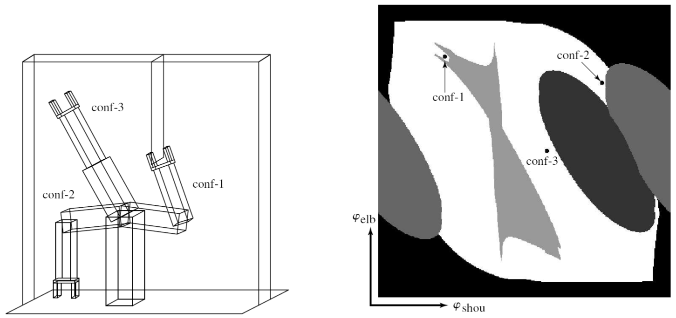
= { x | x is a pose of the robot}

- obstacles → configuration space obstacles



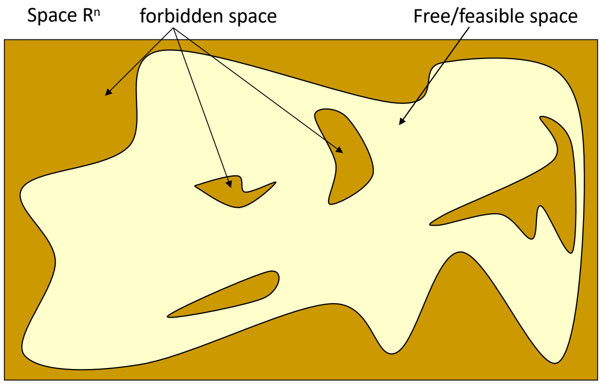
8

Motion planning



9

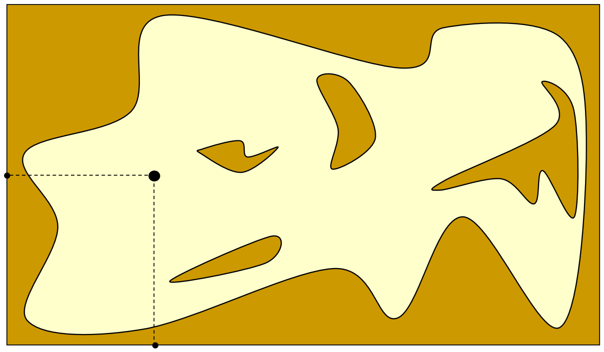
Probabilistic Roadmap (PRM)



10

Probabilistic Roadmap (PRM)

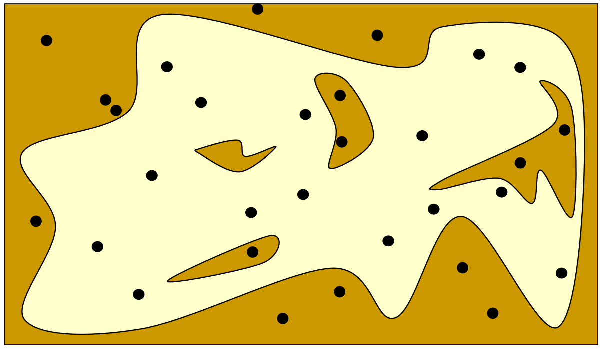
Configurations are sampled by picking coordinates at random



11

Probabilistic Roadmap (PRM)

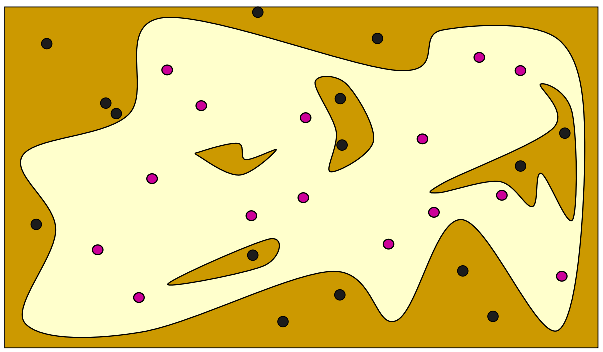
Configurations are sampled by picking coordinates at random



12

Probabilistic Roadmap (PRM)

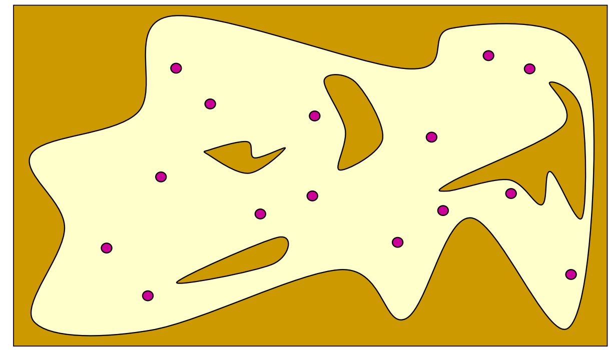
Sampled configurations are tested for collision



13

Probabilistic Roadmap (PRM)

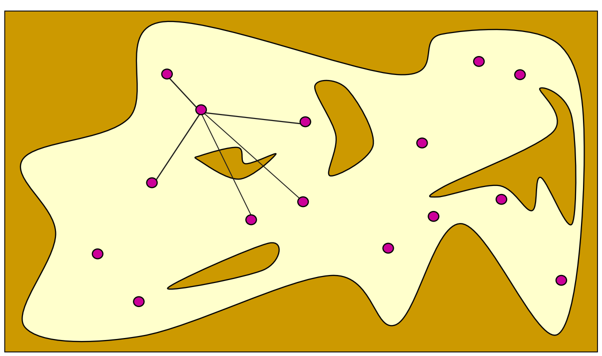
The collision-free configurations are retained as milestones



14

Probabilistic Roadmap (PRM)

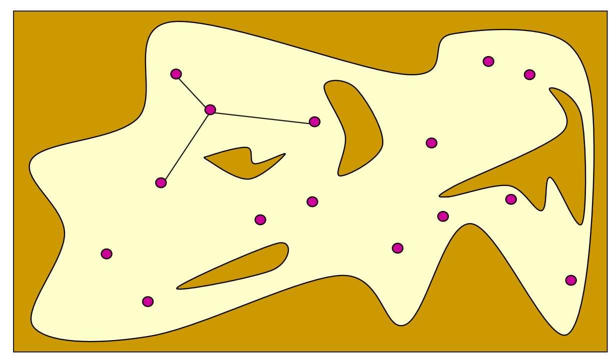
Each milestone is linked by straight paths to its nearest neighbors



15

Probabilistic Roadmap (PRM)

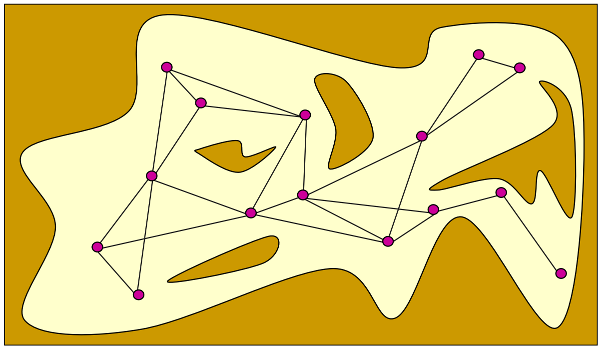
Each milestone is linked by straight paths to its nearest neighbors



16

Probabilistic Roadmap (PRM)

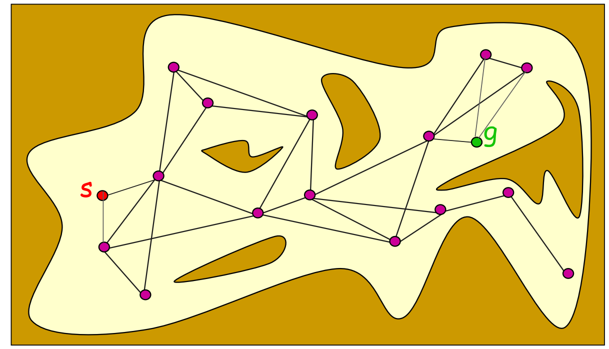
The collision-free links are retained as local paths to form the PRM



17

Probabilistic Roadmap (PRM)

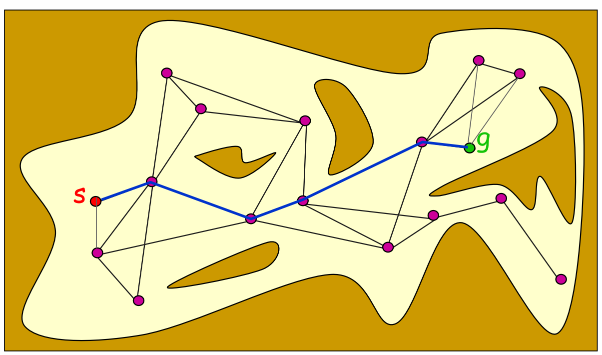
The start and goal configurations are included as milestones



18

Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g



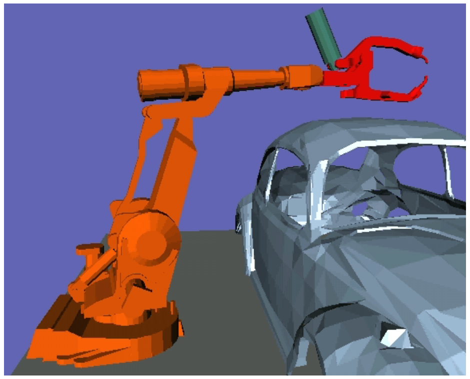
19

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

20

PRM Example 1



21

PRM Example 2

22

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

23

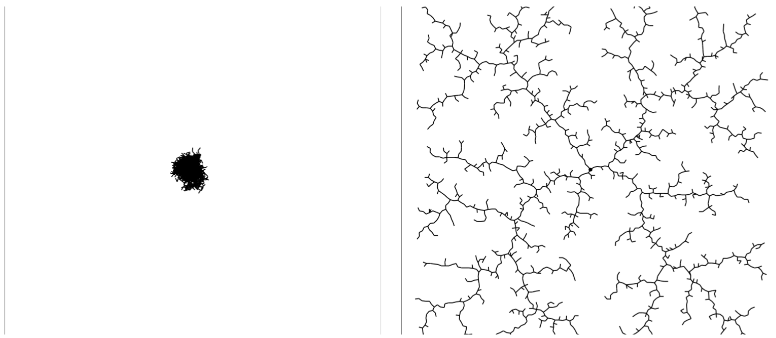
Rapidly exploring Random Tree (RRT)

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

24

How to Sample



25

Rapidly exploring Random Tree (RRT)

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

26

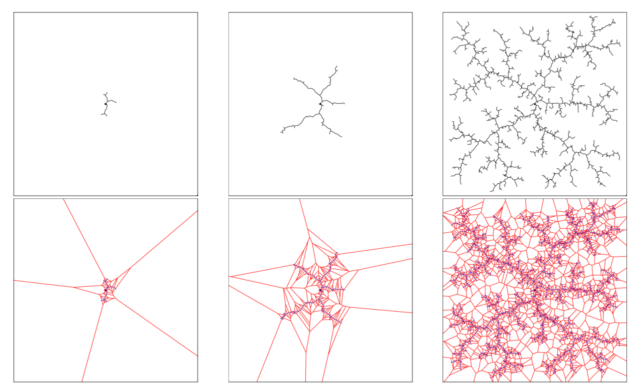
Rapidly exploring Random Tree (RRT)

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

28

Rapidly exploring Random Tree (RRT)



Source: LaValle and Kuffner 01

29

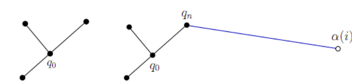
RRT Practicalities

- 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.

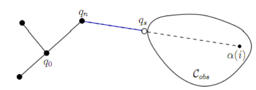
30

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

31

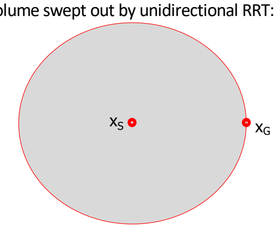
Growing RRT

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

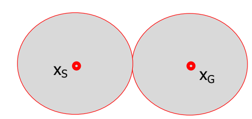
32

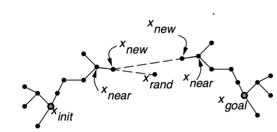
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:



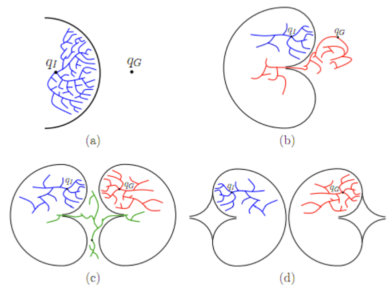


- Difference more and more pronounced as dimensionality increases

33

Multi-directional RRT

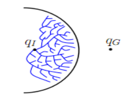
- Planning around obstacles or through narrow passages can often be easier in one direction than the other



34

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

35

RRT*

Algorithm 6: RRT*

```
V ← {x_init}; E ← ∅;
for i = 1, ..., n do
    x_rand ← SampleFree_i;
    x_nearest ← Nearest(G = (V, E), x_rand);
    x_new ← Steer(x_nearest, x_rand) ;
    if ObtacleFree(x_nearest, x_new) then
        X_near ← Near(G = (V, E), x_new, min{γ_RRT*(log(card (V))/ card (V))^(1/d), η}) ;
        V ← V ∪ {x_new};
        x_min ← x_nearest; c_min ← Cost(x_nearest) + c(Line(x_nearest, x_new));
        foreach x_near ∈ X_near do                    // Connect along a minimum-cost path
            if CollisionFree(x_near, x_new) ∧ Cost(x_near) + c(Line(x_near, x_new)) < c_min then
                x_min ← x_near; c_min ← Cost(x_near) + c(Line(x_near, x_new))
        E ← E ∪ {(x_min, x_new)};
        foreach x_near ∈ X_near do                                    // Rewire the tree
            if CollisionFree(x_new, x_near) ∧ Cost(x_new) + c(Line(x_new, x_near)) < Cost(x_near)
            then x_parent ← Parent(x_near);
            E ← (E \ {(x_parent, x_near)}) ∪ {(x_new, x_near)}
return G = (V, E);
```

Source: Karaman and Frazzoli

36

RRT*

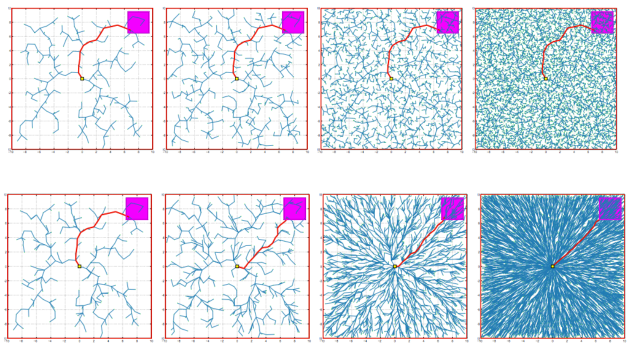
- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

37

RRT*

RRT

RRT*

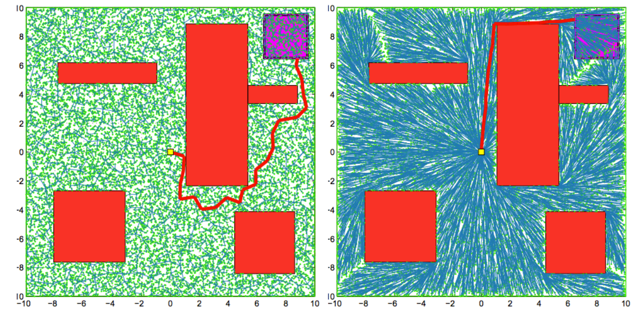


Source: Karaman and Frazzoli

38

RRT*

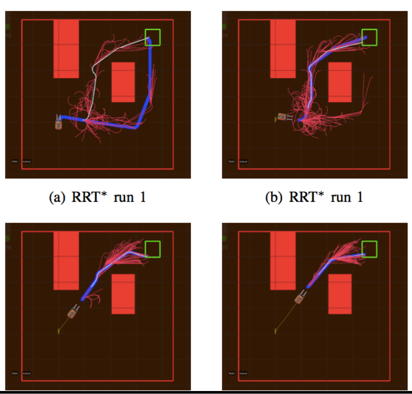
RRT RRT*



Source: Karaman and Frazzoli

39

Real Time RRT*



(a) RRT* run 1 (b) RRT* run 1

40

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.

41

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
- 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

42