CSE-571

Sampling-Based Motion Planning

Various slides based on those from Pieter Abbeel, Zoe McCarthy

Many images from Lavalle, Planning Algorithms

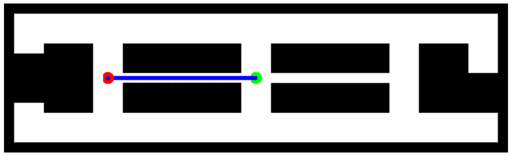
1

Motion/Path Planning

- Task:
  find a feasible (and cost-minimal) path/motion from the current configuration of the robot to its goal configuration (or one of its goal configurations)
- Two types of constraints:
  environmental constraints (e.g., obstacles)
  dynamics/kinematics constraints of the robot
- Generated motion/path should (objective):
  be any feasible path
  minimize cost such as distance, time, energy, risk, …

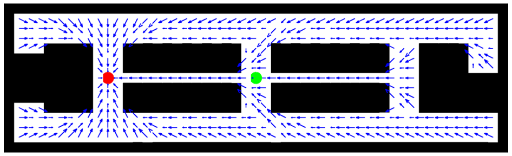
2

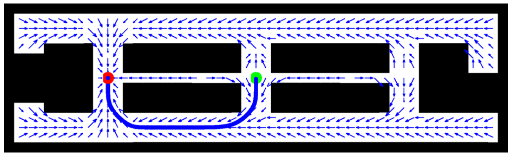
Deterministic, fully observable



3

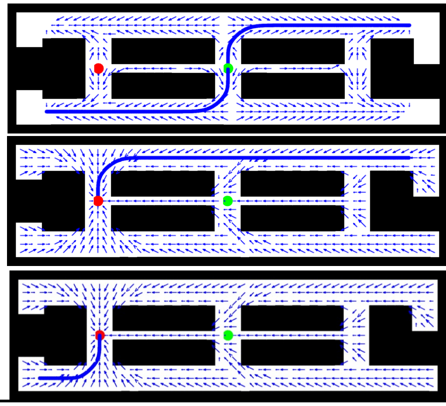
Stochastic, Fully Observable





4

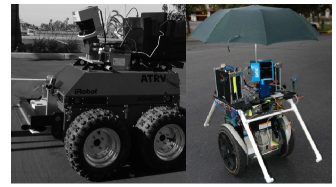
Stochastic, Partially Observable

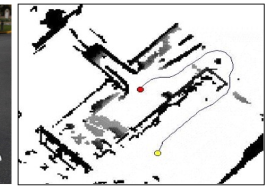


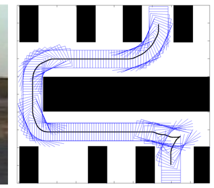
5

Motion/Path Planning

Examples (of what is usually referred to as path planning):

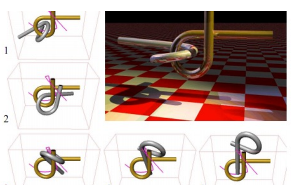


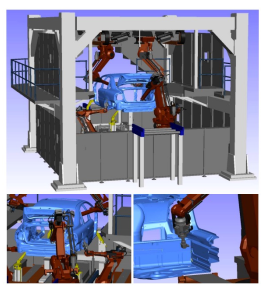




6

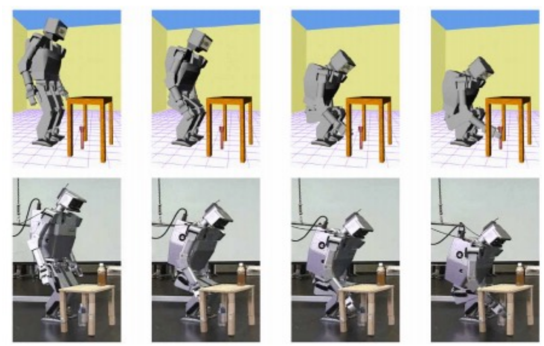
Examples





7

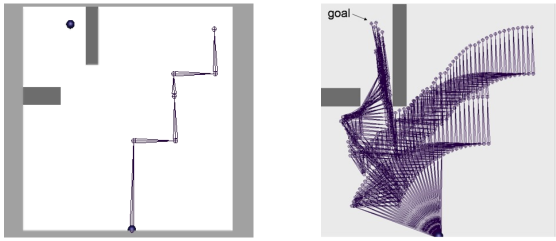
Examples



8

Motion/Path Planning

Examples (of what is usually referred to as motion planning):



Planned motion for a 6DOF robot arm

9

Motion Planning: Outline

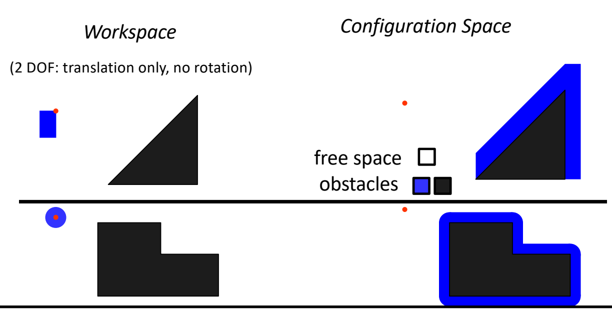
- Configuration Space
- Probabilistic Roadmap
- Rapidly-exploring Random Trees (RRTs)
- Extensions
- Smoothing

10

Configuration Space (C-Space)

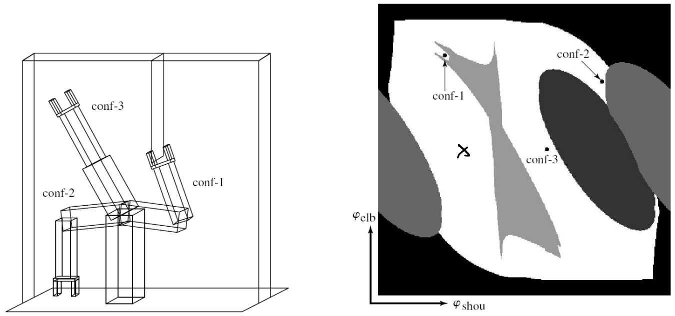
= { x | x is a pose of the robot}

- obstacles → configuration space obstacles



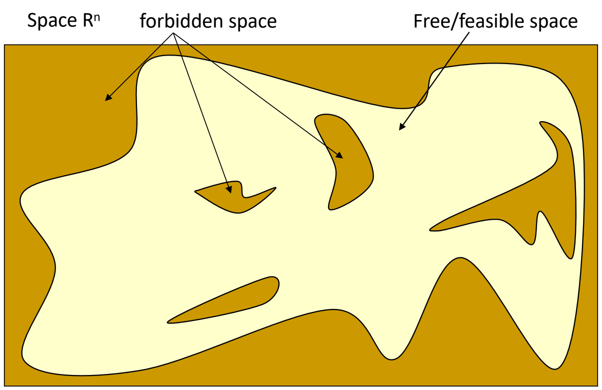
11

Motion planning



12

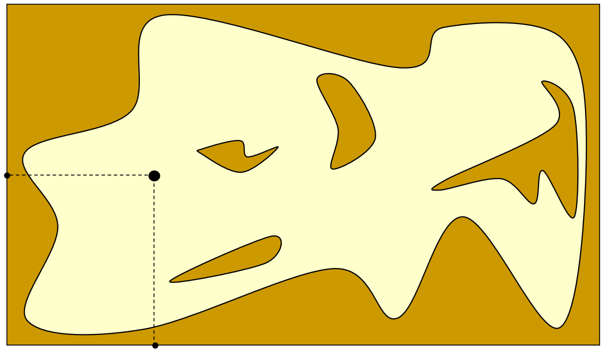
Probabilistic Roadmap (PRM)



13

Probabilistic Roadmap (PRM)

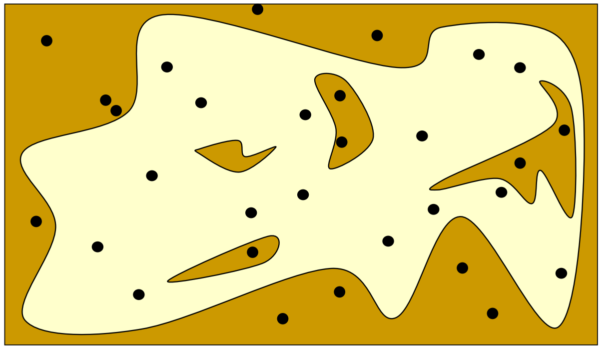
Configurations are sampled by picking coordinates at random



14

Probabilistic Roadmap (PRM)

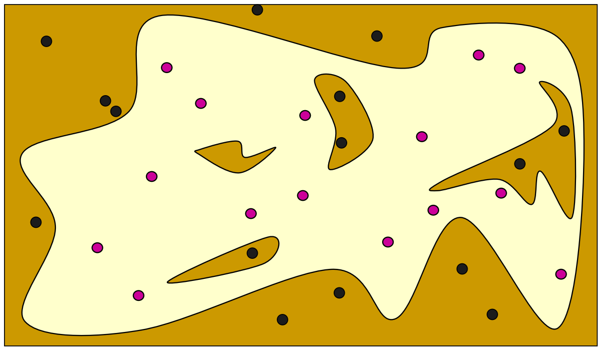
Configurations are sampled by picking coordinates at random



15

Probabilistic Roadmap (PRM)

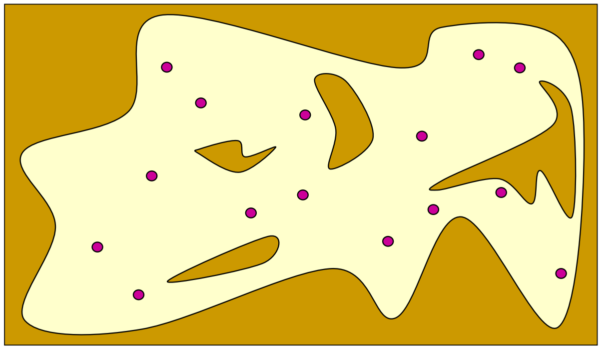
Sampled configurations are tested for collision



16

Probabilistic Roadmap (PRM)

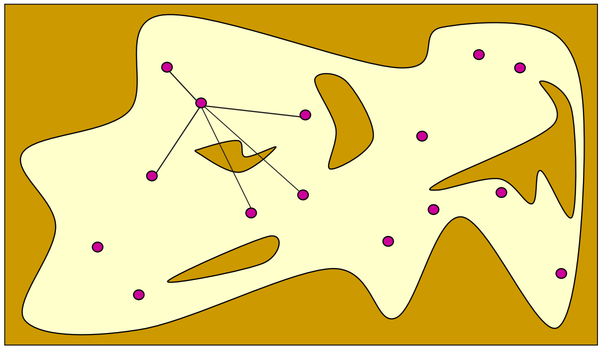
The collision-free configurations are retained as milestones



17

Probabilistic Roadmap (PRM)

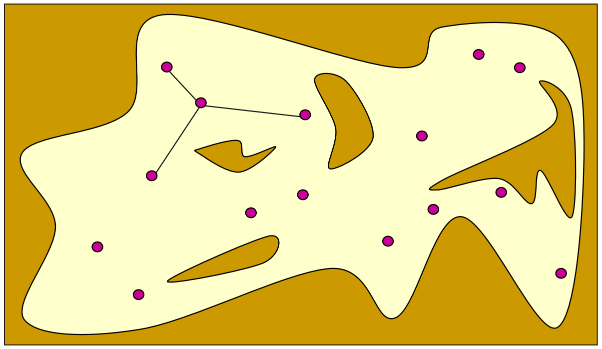
Each milestone is linked by straight paths to its nearest neighbors



18

Probabilistic Roadmap (PRM)

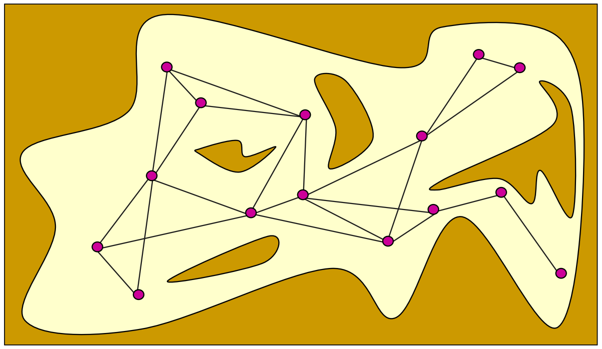
Each milestone is linked by straight paths to its nearest neighbors



19

Probabilistic Roadmap (PRM)

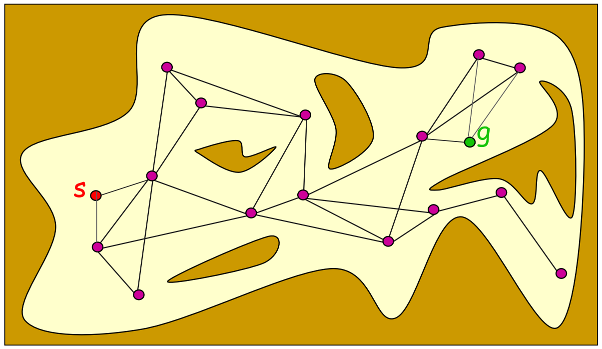
The collision-free links are retained as local paths to form the PRM



20

Probabilistic Roadmap (PRM)

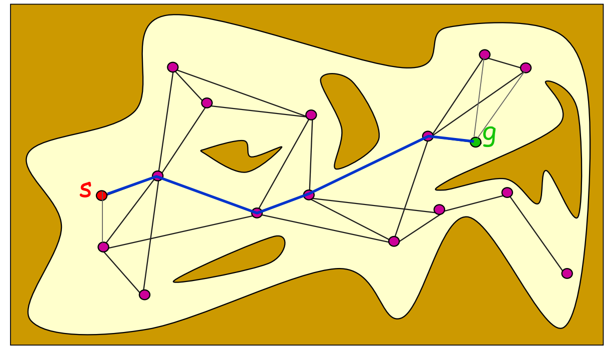
The start and goal configurations are included as milestones



21

Probabilistic Roadmap (PRM)

The PRM is searched for a path from s to g



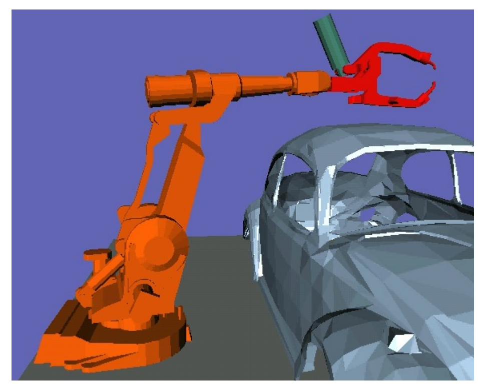
22

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

23

PRM Example 1



24

PRM Example 2

25

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

26

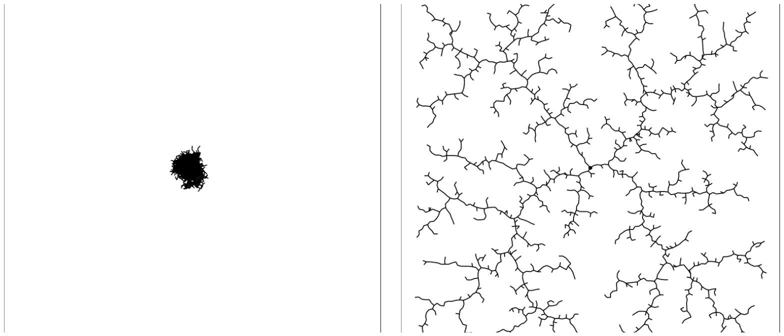
Rapidly exploring Random Tree (RRT)

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

27

How to Sample



28

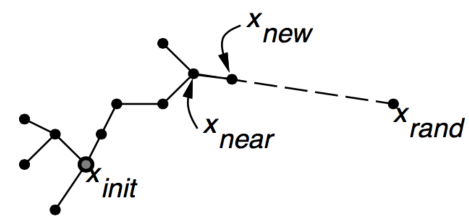
Rapidly exploring Random Tree (RRT)

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

29

Rapidly exploring Random Tree (RRT)

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



30

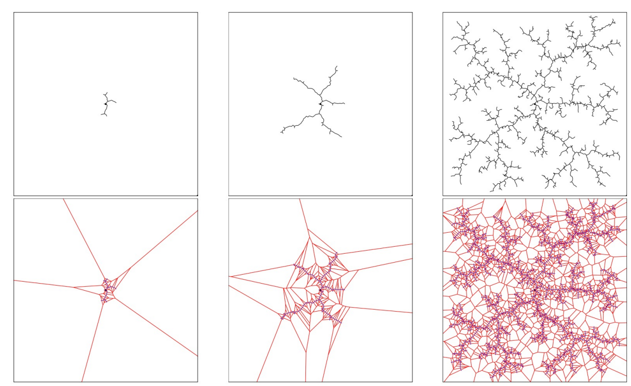
Rapidly exploring Random Tree (RRT)

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

31

Rapidly exploring Random Tree (RRT)



Source: LaValle and Kuffner 01

32

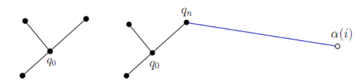
RRT Practicalities

- 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.

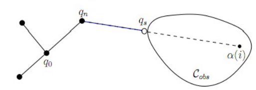
33

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

34

Growing RRT

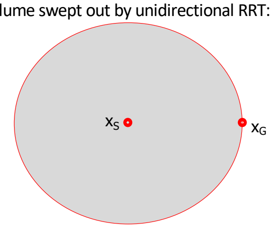
45 iterations 390 iterations

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

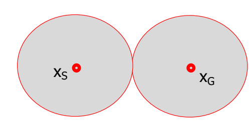
35

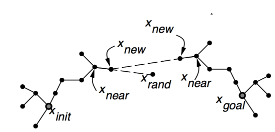
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:



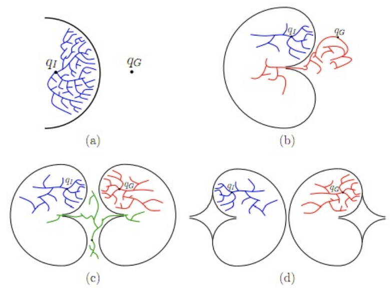


- Difference more and more pronounced as dimensionality increases

36

Multi-directional RRT

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



37

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

38

RRT*

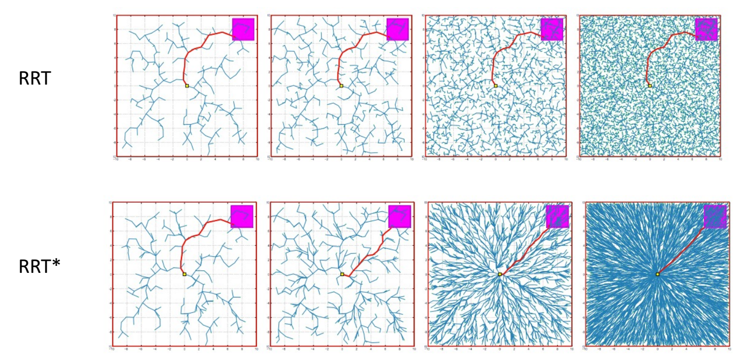
Algorithm 6: RRT*

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

Source: Karaman and Frazzoli

39

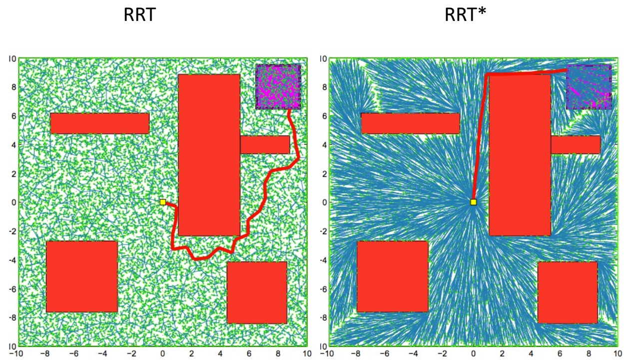
RRT*



Source: Karaman and Frazzoli

40

RRT*



Source: Karaman and Frazzoli

41

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.

42

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

43