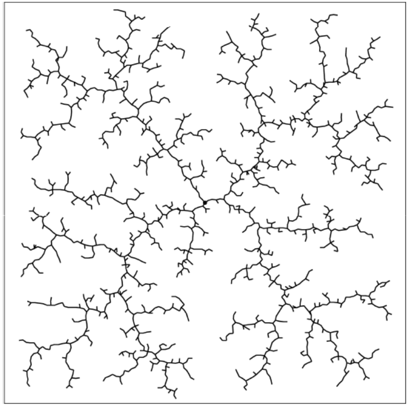
COMS W4733: Computational Aspects of Robotics

Lecture 19: Probabilistic Planning



Instructor: Tony Dear

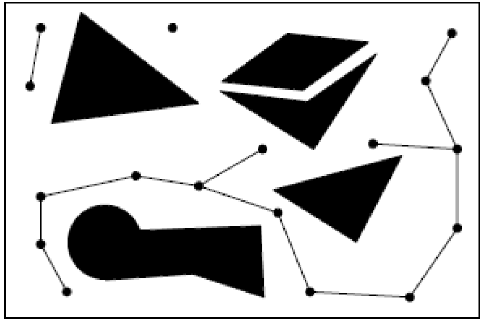
Many slide materials from H. Choset, N. Amato, S. Bhattarcharjee, G. D. Hager, S. LaValle, J. Kuffner, P. Allen, P. Abbeel

Sampling-Based Methods

- Planning in high-dimensional spaces is hard
- *Constructing* high-dimensional C-spaces is hard as well
- Do we always need a complete and accurate C-space representation?

- Idea: If we can find a sampling of valid (collision-free) configurations that can be connected, that may be enough to build a roadmap

- Sacrifice some optimality, completeness



Sampling-Based Methods

- Collision checking: Need to do this efficiently as part of sampling process
- *Probabilistic completeness*: If a solution exists, the planner will eventually find it, e.g. by increasing the density of sampling
- *Resolution completeness*: As with cell decomposition, solutions may exist if resolution is higher than some threshold but not lower
- In practical settings, sampling methods often work better than exact roadmap methods!

Probabilistic Roadmaps

- Idea: Randomly check different robot configurations $\boldsymbol{q}_{\mathrm{rand}}$
  - E.g., positions of a mobile robot in space, joint configurations of a manipulator

- If $\boldsymbol{q}_{\mathrm{rand}}$ is a collision-free configuration according to robot kinematics and collision checker, then add it as a node to our roadmap

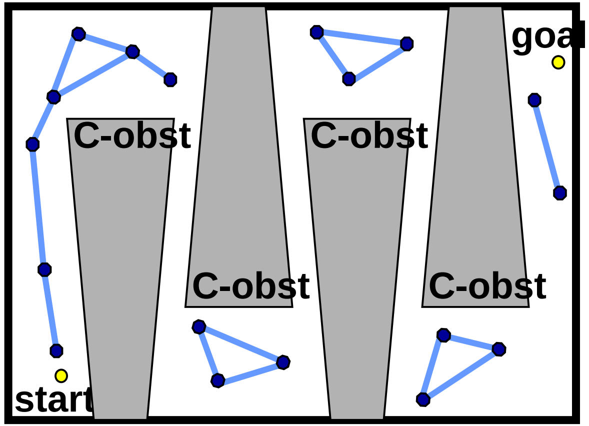
- Attempt to connect $\boldsymbol{q}_{\mathrm{rand}}$ to “nearby” nodes already in the graph using local paths generated by a *local planner*
- Collision-free local paths are added as edges to the roadmap

Probabilistic Roadmaps

- How to sample configurations? Uniform sampling is usually easiest
  - May not be sufficient in low-volume areas or narrow passageways

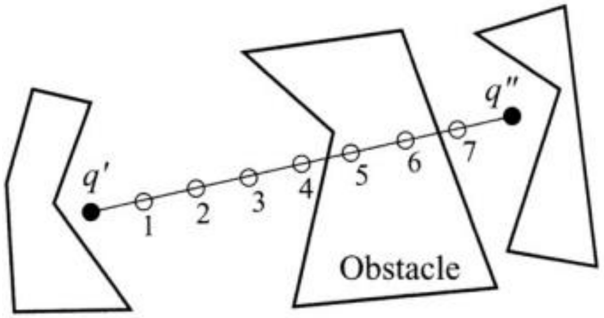
- How to check collisions? Often easier to check in the workspace

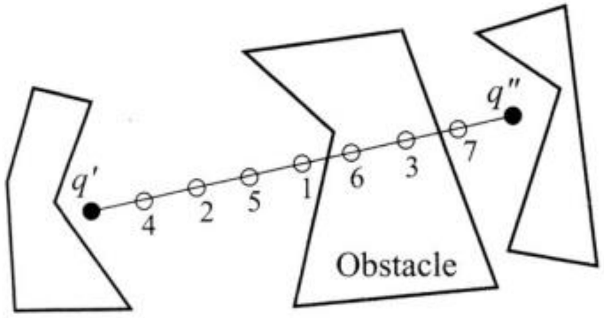
- What about the *local planner*?
- Try to find local paths to k nearest neighbors
- Try straight paths and then check for collisions



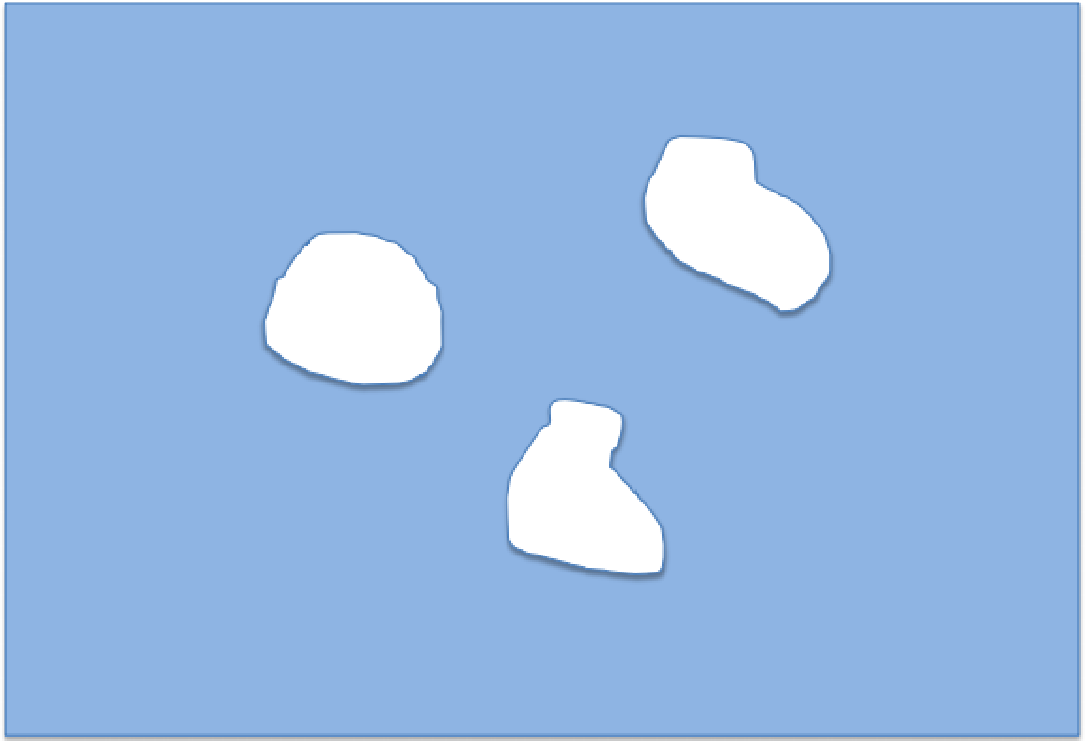
Collision Detection

- Many different ways of checking for collisions efficiently
- Ex: Detecting collisions between straight lines and polygons
- Naïve approach: Subsample discrete points along the line
- Check whether midpoint collides with any polygons, then recursively subdivide and check segment midpoints for collisions

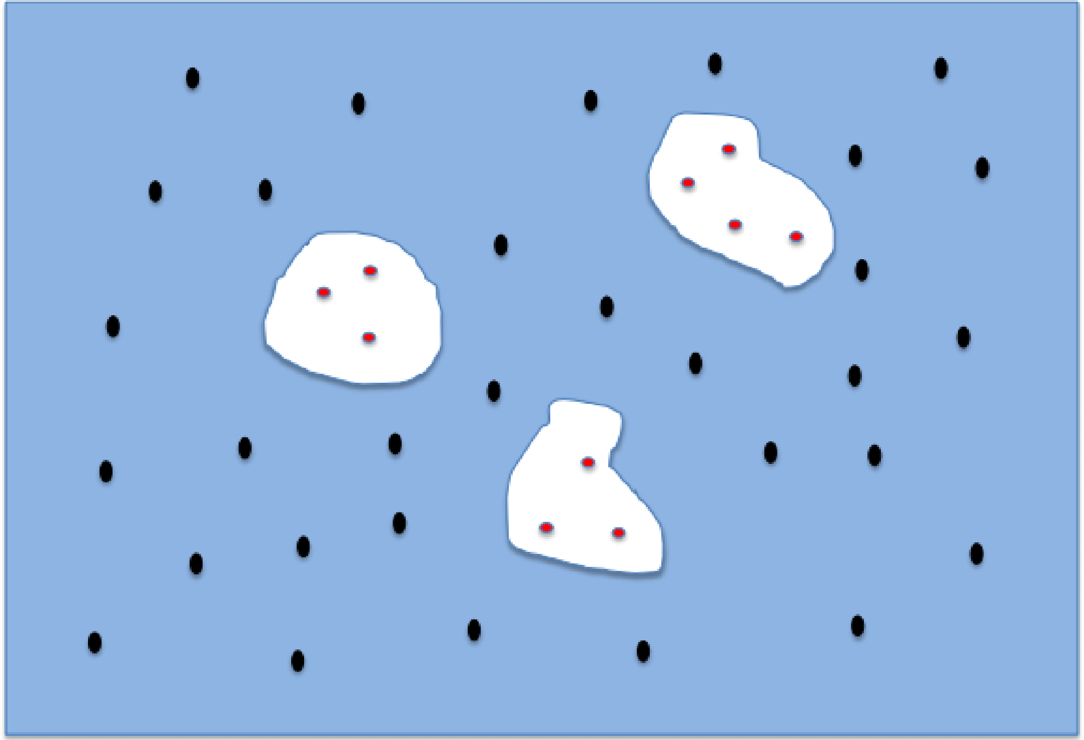




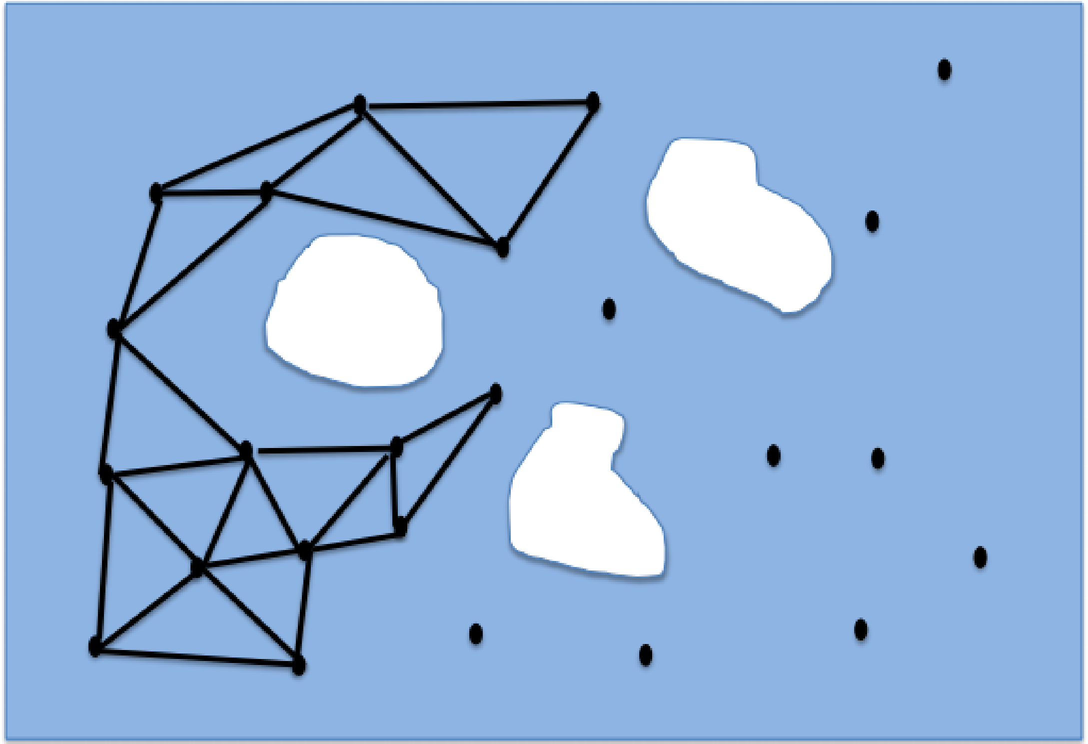
Example: PRM



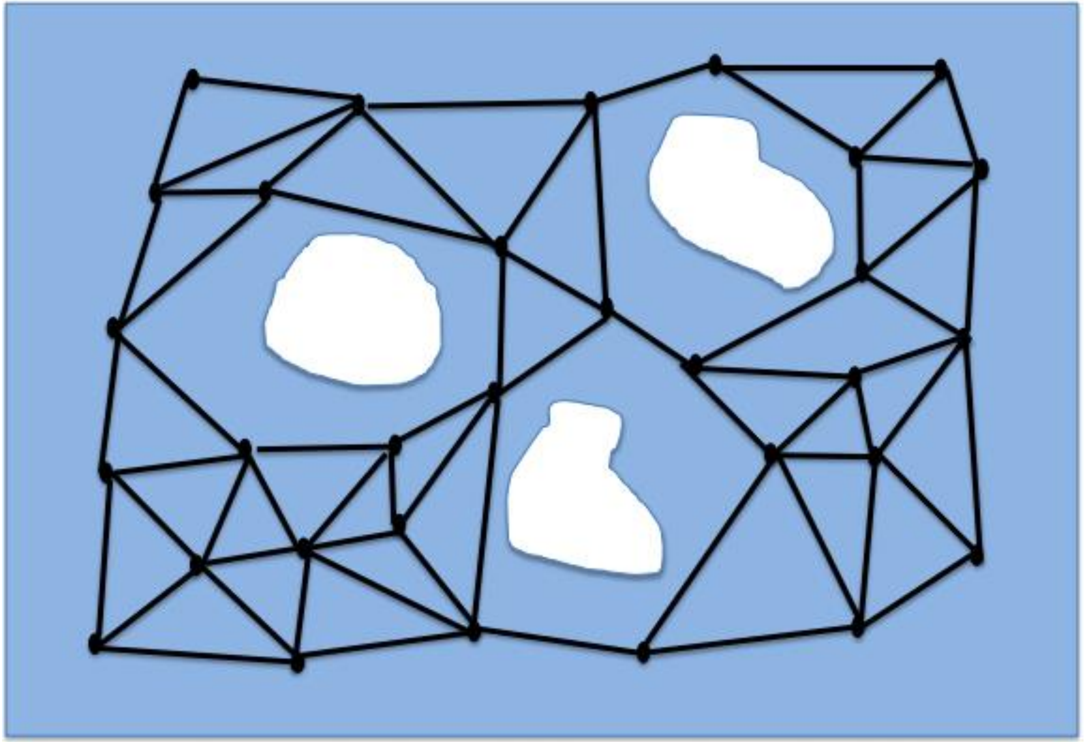
Example: PRM



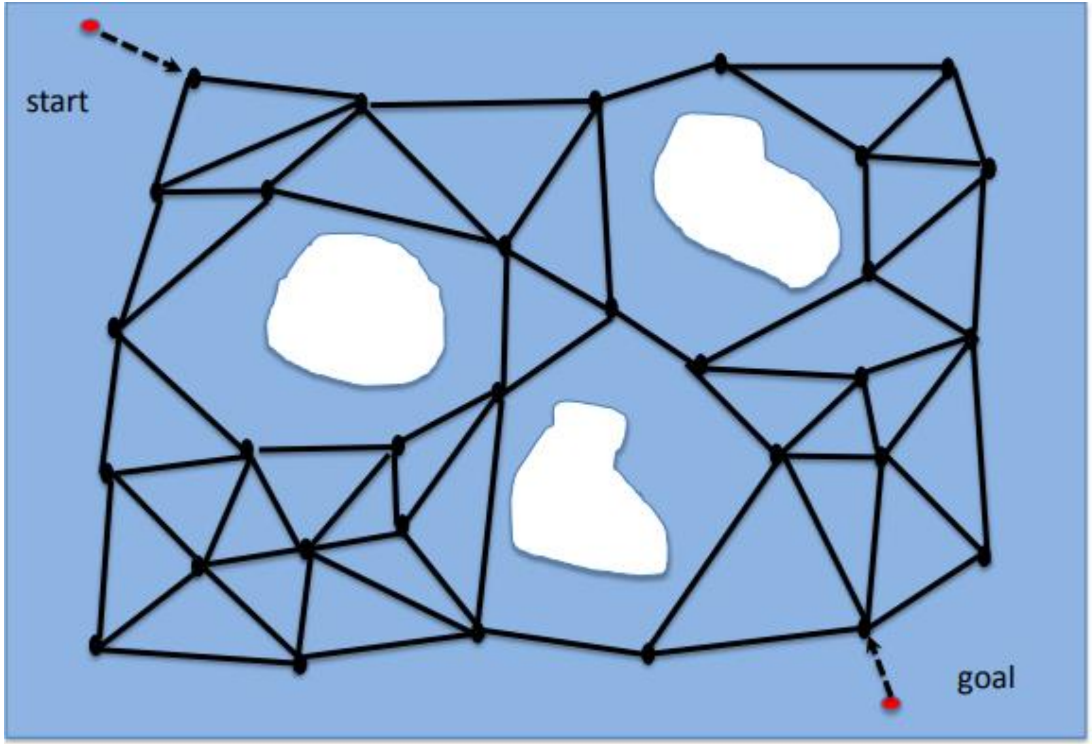
Example: PRM



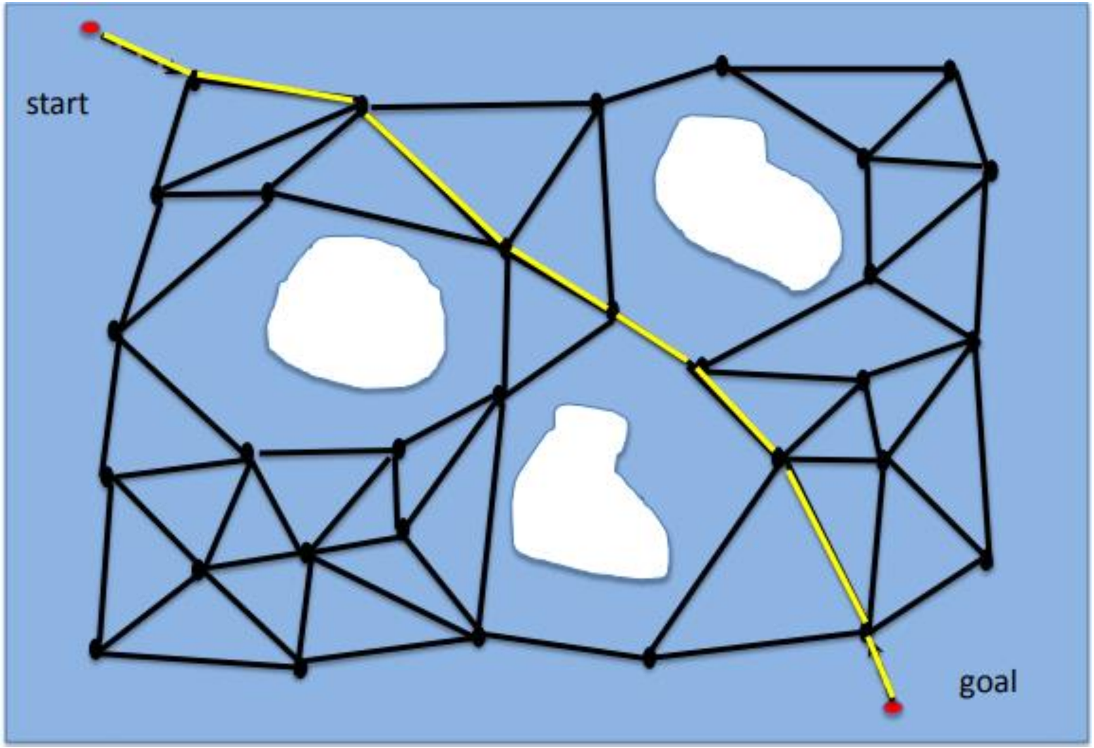
Example: PRM



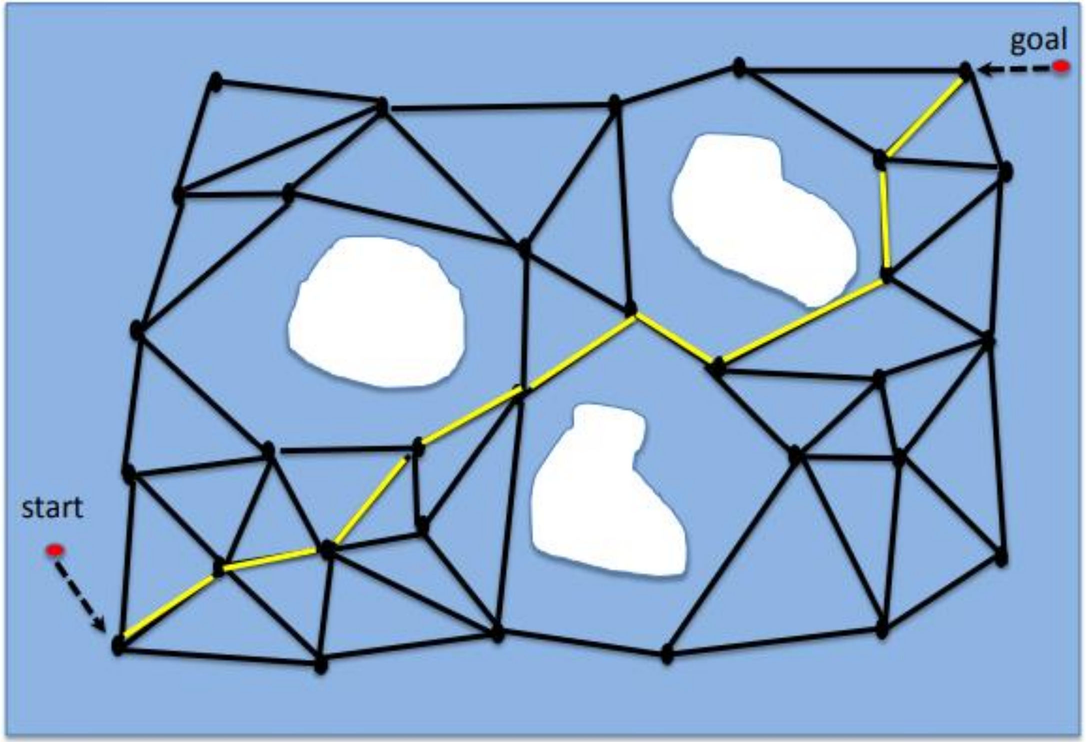
Example: PRM



Example: PRM



Example: PRM



PRM Construction

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
    repeat
        $q \leftarrow$ a random configuration in $\mathcal{Q}$
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
    for all $q' \in N_q$ **do**
        if $(q, q') \notin E$ **and** $\Delta(q, q') \neq \mathtt{NIL}$ **then**
            $E \leftarrow E \cup \{(q, q')\}$
        end if
    end for
end for

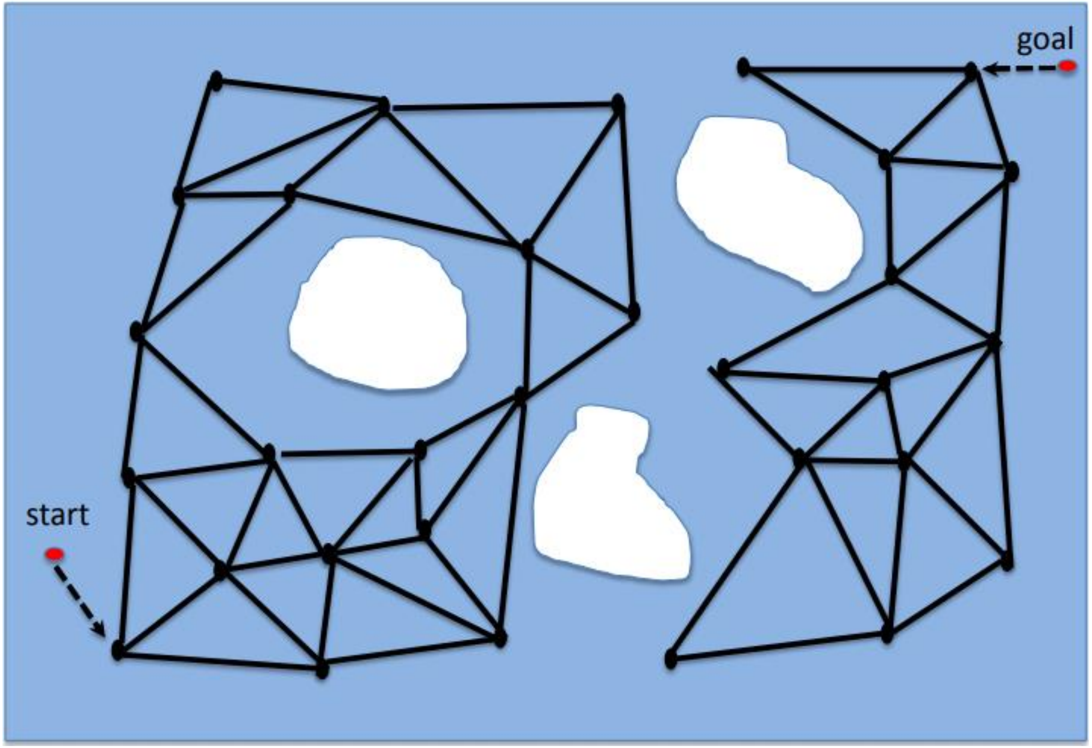
Searching for a Path

- Given start and goal configurations, we can retract them to the roadmap by connecting them to the nearest nodes with a collision-free path

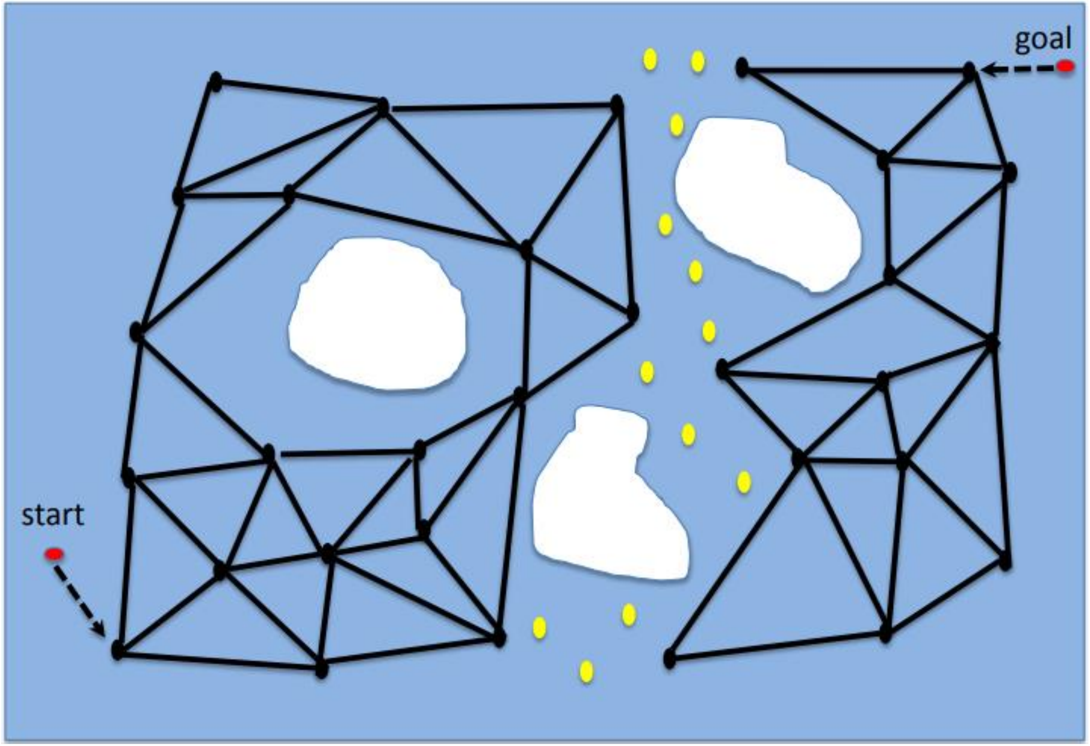
- Problem: What if we can’t find any valid local paths?
- Problem: What if start & goal are connected to different graph components?

- Need more samples! Try to focus on areas lacking the connections
  - Sample around start and goal if having trouble connecting them to roadmap
  - Sample in volume between connected components if components are disjoint

Example: Disjoint Components



Example: Disjoint Components



PRM Enhancement

- Goal: Connect as many disjoint components as possible

- One approach: Sample more nodes near narrow passages
- Focus sampling in regions around nodes with fewer neighbors

- Alternatively, try to find more sophisticated paths between components
- E.g. choose closest nodes on two components as candidates for connection
- Or maybe choose connection candidates randomly, but with bias for nodes with few neighbors (more likely near narrow passages)

PRM Enhancement

- To facilitate enhancement, we are interested in nodes in difficult regions

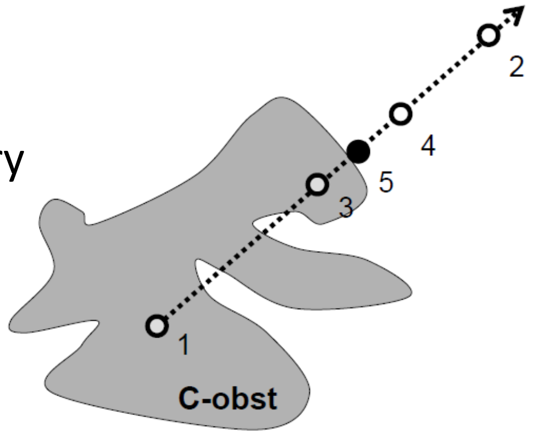
- Maintain a weight/heuristic for each node c as we construct the graph, e.g.
  - Number of neighbors of c within some predefined distance
  - Distance between c and nearest connected component not containing c

- Higher weights make it more likely for a node to be selected for enhancement
- We then attempt to connect the chosen candidates

Non-Uniform Sampling: OBPRM

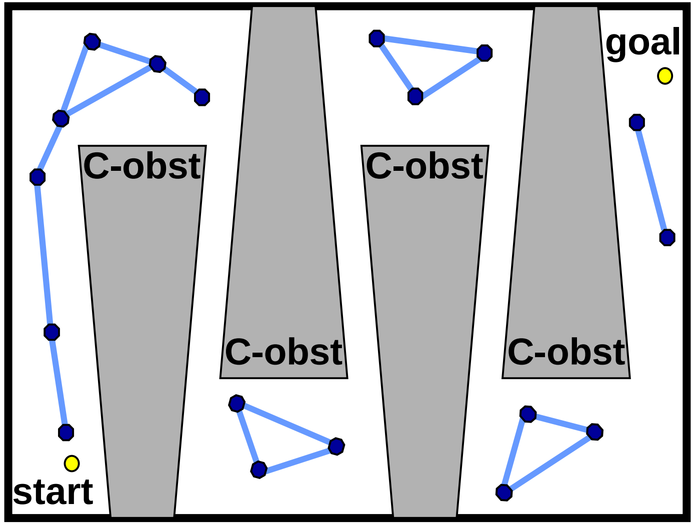
- Sampling uniformly in C-space is not always useful; ideally we want more samples near obstacles (narrow passages) and fewer in free space

- Idea: Use invalid (colliding) samples to find a valid sample on or near the boundary of the offending obstacle
  - Choose a random direction and small distance away
  - Use binary search to find corresponding point on boundary

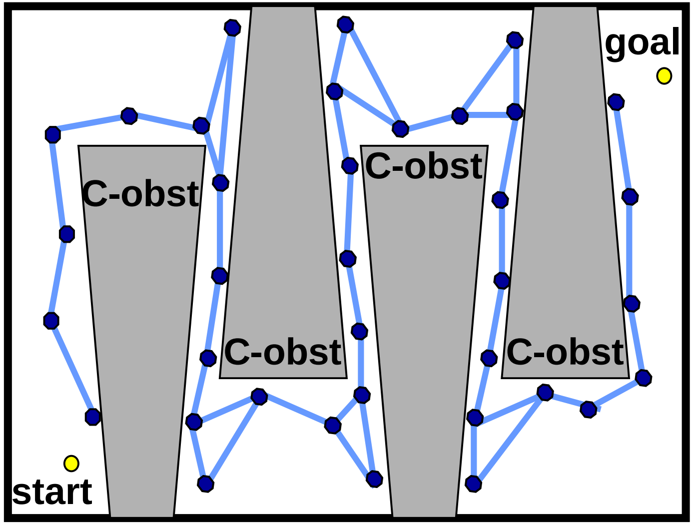


Example: OBPRM

PRM Roadmap



OBPRM Roadmap



Single-Query vs Multiple-Query

- PRMs (and roadmaps in general) are good for multiple queries
- If we change the start and/or goal configurations, we can reuse the same roadmap and simply compute retractions of the new configurations
- More flexibility, but will slow down roadmap construction

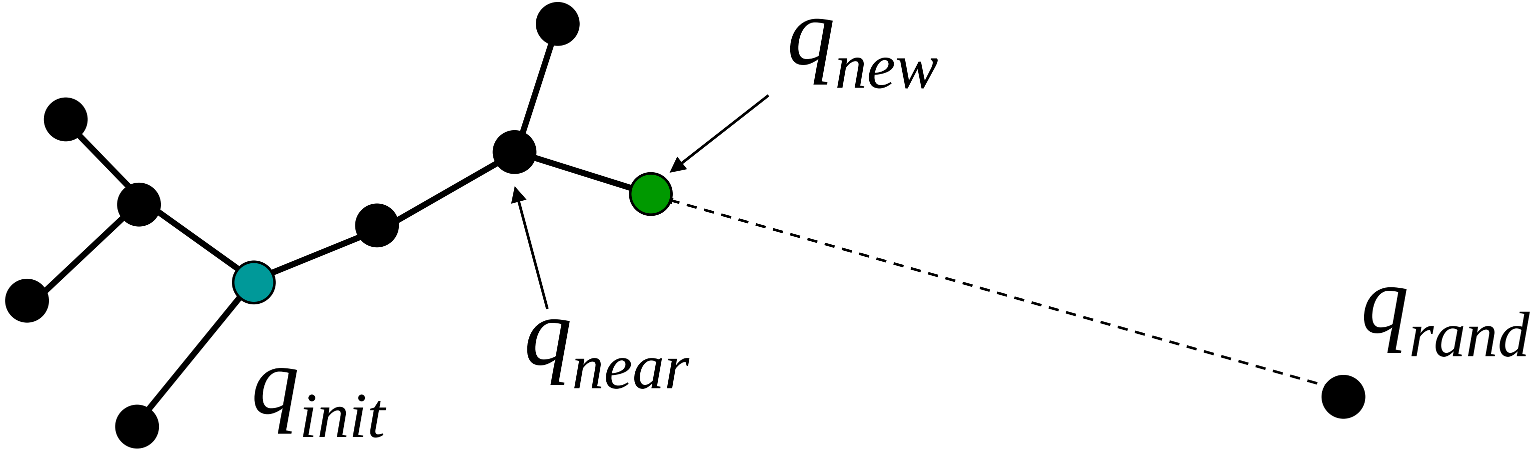
- If we don’t care to reuse our roadmap, we can perform sampling with a single query in mind—don’t need to worry about the entire C-space
- Definite stopping point: done sampling once a path is found

Rapidly-Exploring Random Trees

- Idea: Start a graph at the start or goal configurations and build outward
  - As opposed to arbitrarily placing samples all over the C-space
- New nodes added to the tree lie within a certain step-size δ from current tree

- Procedure: Sample random (free) configuration $\boldsymbol{q}_{\text{rand}}$
- Find closest tree node $\boldsymbol{q}_{\text{near}}$ to $\boldsymbol{q}_{\text{rand}}$; also find $\boldsymbol{q}_{\text{new}}$ a distance of step-size δ away from $\boldsymbol{q}_{\text{near}}$ in direction of $\boldsymbol{q}_{\text{rand}}$
- If path between $\boldsymbol{q}_{\text{near}}$ and $\boldsymbol{q}_{\text{new}}$ is collision-free, *expand* tree and connect $\boldsymbol{q}_{\text{near}}$ to new node $\boldsymbol{q}_{\text{new}}$
- $\boldsymbol{q}_{\text{rand}}$ is discarded afterward

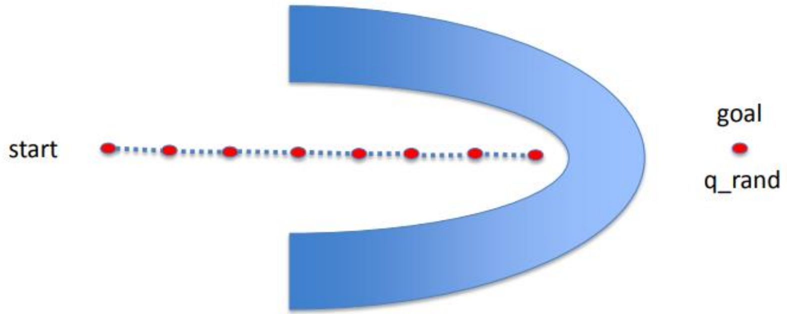
Graph Construction



Adding Bias to RRT Construction

- We want to "explore" outward, but we also want to make sure that we're making progress toward the goal as we build
- Idea: Add a bias when selecting q_{rand} to select goal some of the time (~5%)
- Tree will expand in goal direction on occasion

- Too much bias can lead to local minima! Ex: $q_{\text{rand}} = q_{\text{goal}}$ all the time:

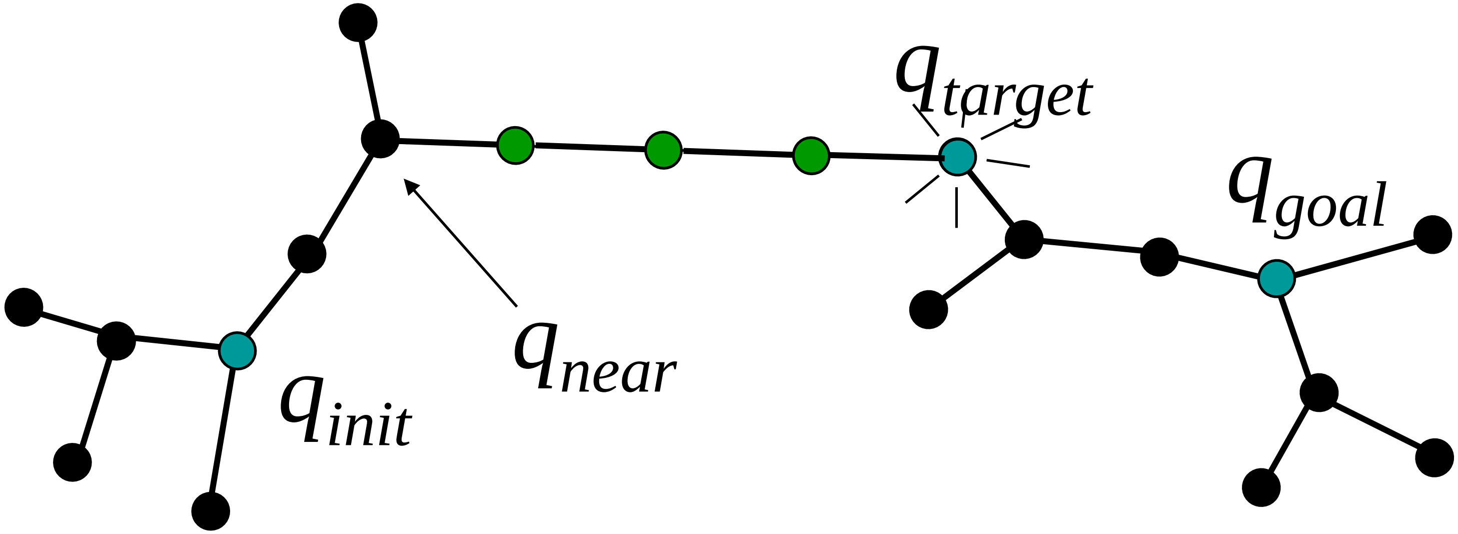


Bidirectional RRT

- We may increase efficiency even more if we grow RRTs from start and goal
- After growing for a while, we need a *connection* procedure

- Expand T_1 randomly and add node $\boldsymbol{q}_{1,\mathrm{new}}$
- Expand T_2 toward $\boldsymbol{q}_{1,\mathrm{new}}$; if connected, complete path is found
- If not connected but T_2 does expand and add $\boldsymbol{q}_{2,\mathrm{new}}$, swap roles and have T_1 expand toward $\boldsymbol{q}_{2,\mathrm{new}}$; iterate until finished or failure
- If not connected (collision) and no expansion, then not ready for connection

Example: Connecting RRTs



RRT Connect Algorithm

RRT_CONNECT ($q_{init,}$ q_{goal}) {

  T_a.*init*(q_{init}); T_b.*init*(q_{goal});

  for *k* = 1 to K **do**

    q_{rand} = RANDOM_CONFIG();

    if not (EXTEND(T_a, q_{rand}) = Trapped) **then**

      if (EXTEND(T_b, q_{new}) = Reached) **then**

        return PATH($T_{a,}$ T_b);

    SWAP($T_{a,}$ T_b);

  return Failure;

}

RRT Considerations

- RRT is fundamentally a balance between greedy search and exploration

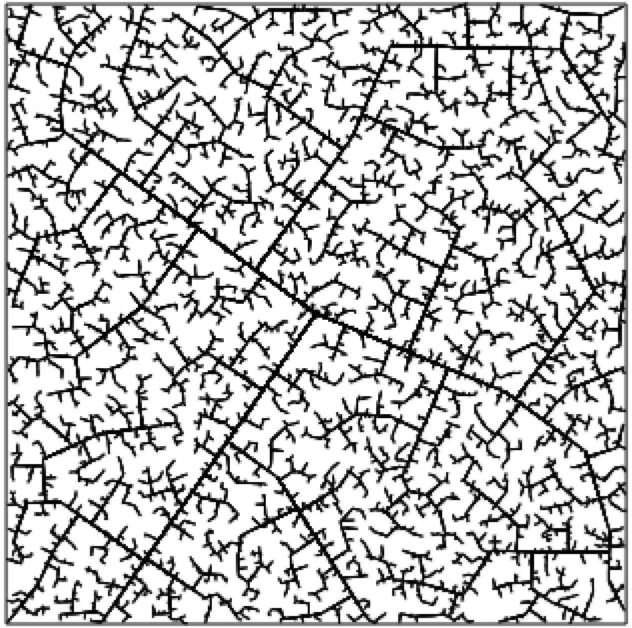
- Stepsize δ may be chosen as a function of free space; with many free regions, we can choose δ to be bigger and have a more greedy construction procedure

- RRT vertex distribution converges to sampling distribution in the long term
- Probability of finding a path increases **exponentially** with number of iterations

- RRT is probabilistically complete but not optimal (not even asymptotically)

RRT Limiting Behavior

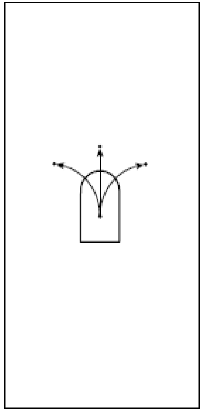
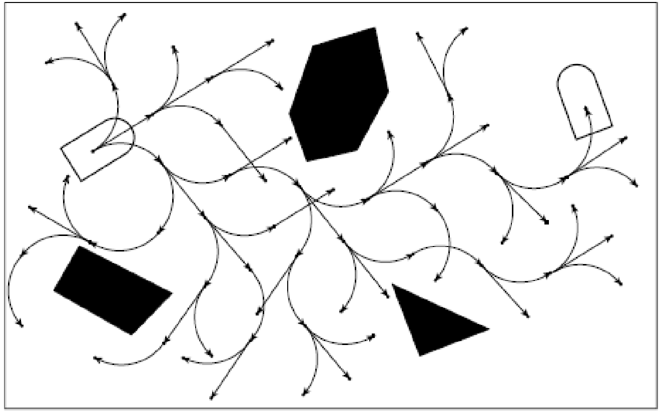
45 iterations



2345 iterations

RRTs for Nonholonomic Robots

- Robots with nonholonomic constraints cannot travel in arbitrary directions
- Instead, identify admissible motion primitives, e.g. straight paths and turns with constant curvature
- When expanding RRT, only add nodes that can be reached via a motion primitive from an existing node in RRT

RRT*

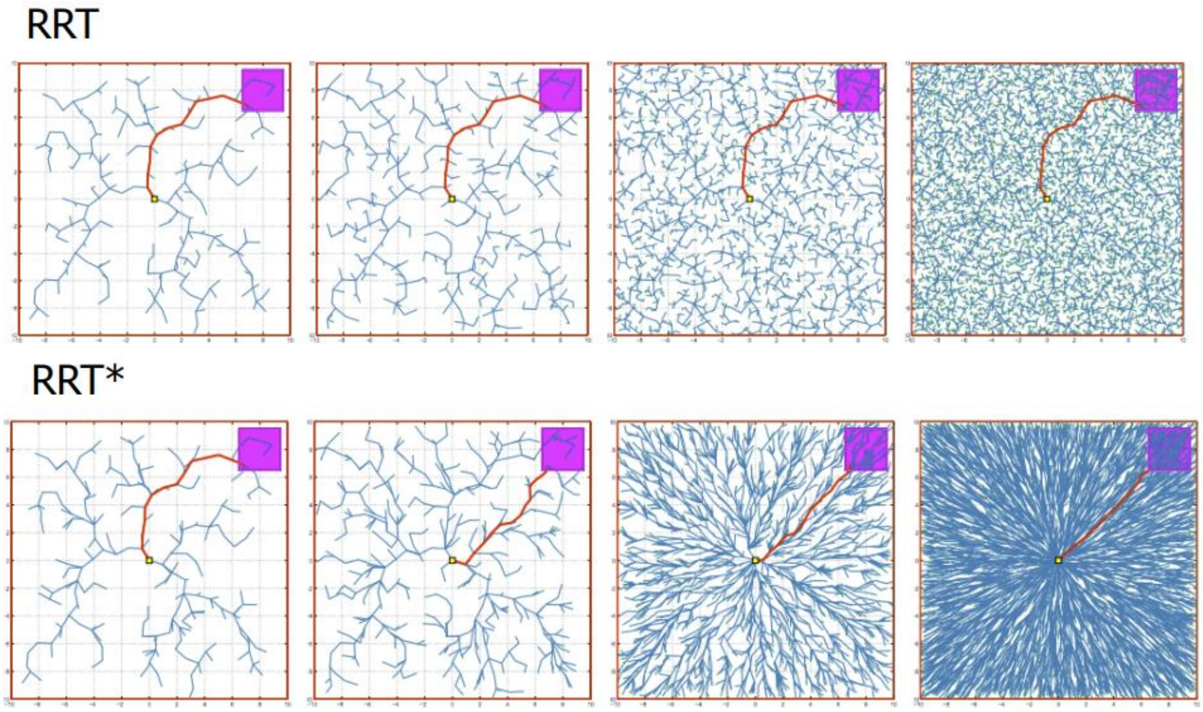
- As with PRMs, many different versions of RRTs suited for different purposes

- Rapidly-exploring random graph (**RRG**): New graph nodes can be connected to multiple nearby nodes in graph so far, leading to possible cycles

- Optimal RRT (**RRT***): Add multiple edges to new node as in RRG, but then prune out cycles so that only those corresponding to optimal (shortest) paths from root are kept
- Existing nodes may change parents as tree grows outward!
- Trees tend to be more fan-shaped, paths tend to be smoother

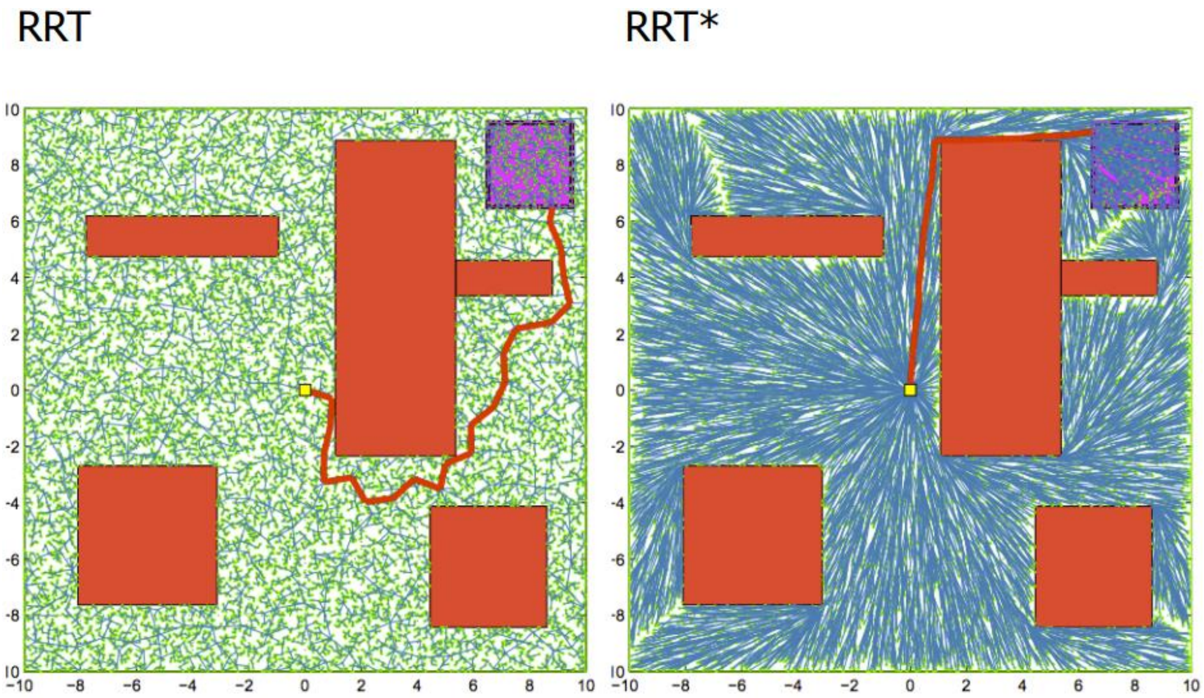
- Unlike RRT, this is asymptotically optimal (but at cost of increased computation)

RRT*



Source: Karaman and Frazzoli

RRT*



Source: Karaman and Frazzoli

Summary

- Sampling-based methods are very useful when we cannot or don’t need to build the entire C-space for planning
- PRM: Build an approximate roadmap through free C-space, especially regions near obstacles; supports multiple queries of different start/goal configurations
- RRT: Single-query; balance exploration and exploitation to find some coverage of C-space but simultaneously favor paths that solve given problem

- http://msl.cs.uiuc.edu/rrt/index.html
- https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree
- http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/