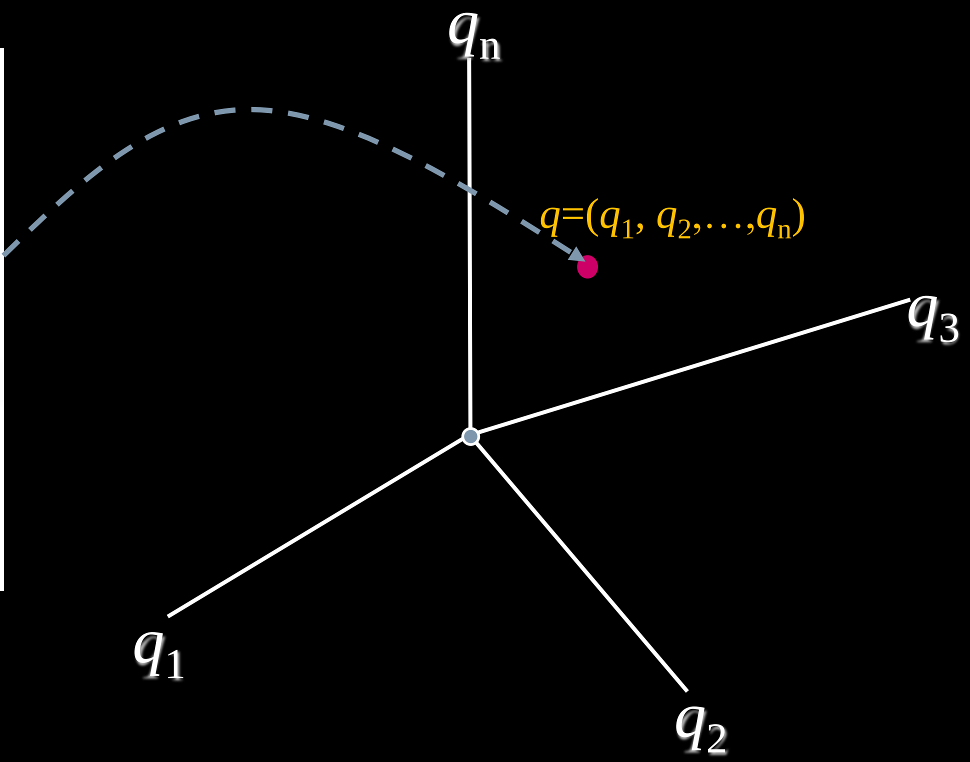
Motion Planning III - Sampling-based Planning

A lot of Material from Howie Choset, Nancy Amato, Sujay Bhattacharjee, G.D. Hager, S. LaValle, J. Kuffner, D. Hsu

Last time…

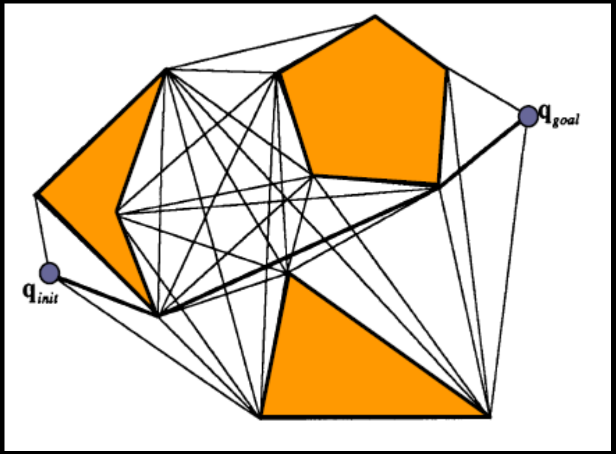
- We learned about configuration space (C-space)



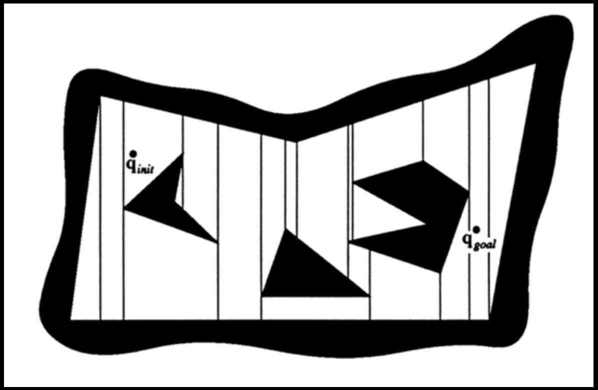
- How do we plan in **high-dimensional** C-spaces?

Exact methods: The problem

- Exact methods either find a solution or prove none exists
- Require computing C-space obstacles
  - Very computationally expensive!

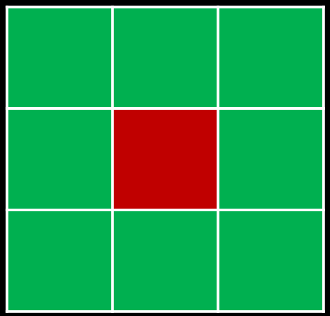


- Decomposition methods (cells, octree, etc.) also not practical b/c sweeping/decomposition is sensitive to dimension
  - Also requires either knowing C-space obstacle or lots of collision checking



Discrete Planning: The problem

- Discrete search run-time and memory requirements are very sensitive to branching factor (number of successors)
- Number of successors depend on dimension
- For a 3-dimensional 8-connected space, 26 successors
- For an n-dimensional 8-connected space, $3^n - 1$ successors
  - Increases very quickly!



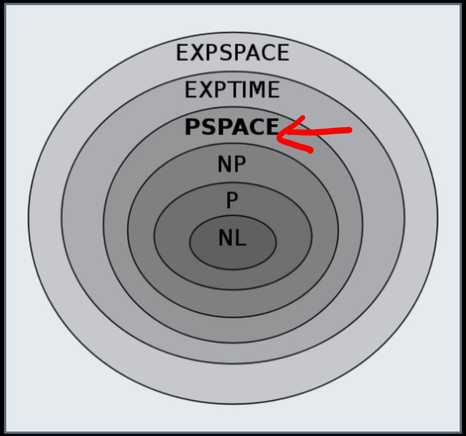
"8-connected"

The problem

- Need a path planning method that isn't so sensitive to dimensionality
- But:
  - Path planning is PSPACE-hard

    [Reif 79, Hopcroft et al. 84, 86]
  - Complexity is exponential in dimension of the C-space [Canny 86]
- What if we weaken *completeness* and *optimality* requirements?

Real robots can have 20+ DOF!



Weakening requirements

Ideal		Practical in High Dimensions
Complete	→	Probabilistically Complete
Optimal	→	Feasible

*More recent methods show asymptotic optimality

- Probabilistic completeness: A path planner is probabilistically complete if, given a solvable problem, the probability that the planner solves the problem goes to 1 as time goes to infinity.

- Feasibility: Path obeys all constraints (usually obstacles).
- A feasible path can be optimized *locally* after it is found

Sampling-based planning

- Main idea: Instead of systematically-discretizing the C-space, take samples in the C-space and use them to construct a graph

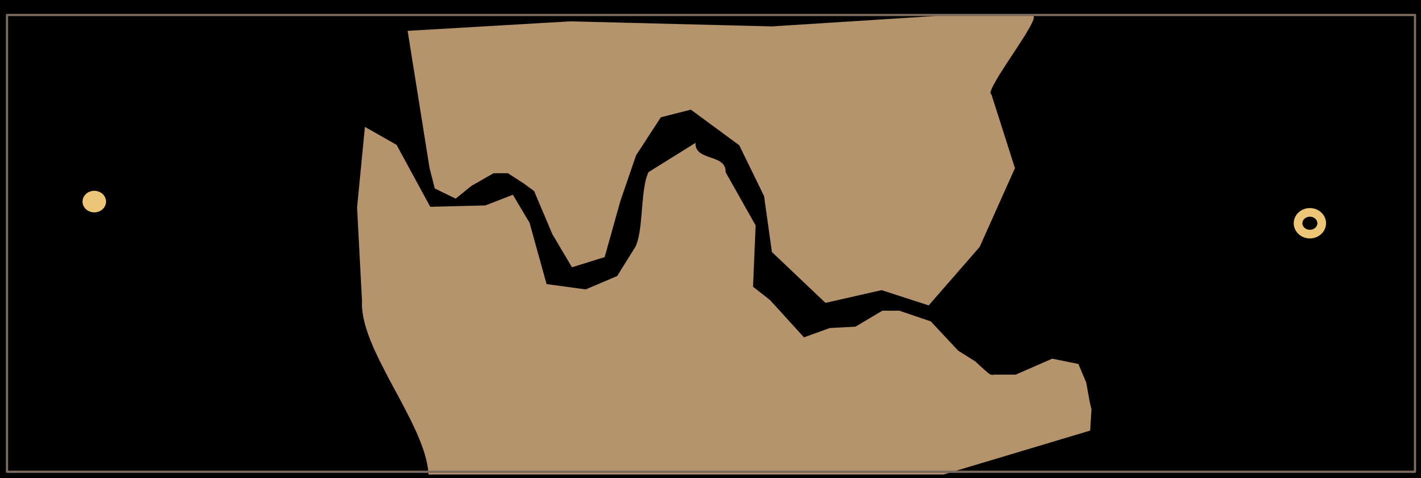
Sampling-based planning

<u>Advantages</u>

- Don’t need to discretize C-space
- Don’t need to explicitly represent C-space
- Easy to sample high-dimensional spaces

<u>Disadvantages</u>

- Probability of sampling an area depends on the area’s size
    - Hard to sample *narrow passages*
- No strict completeness/optimality

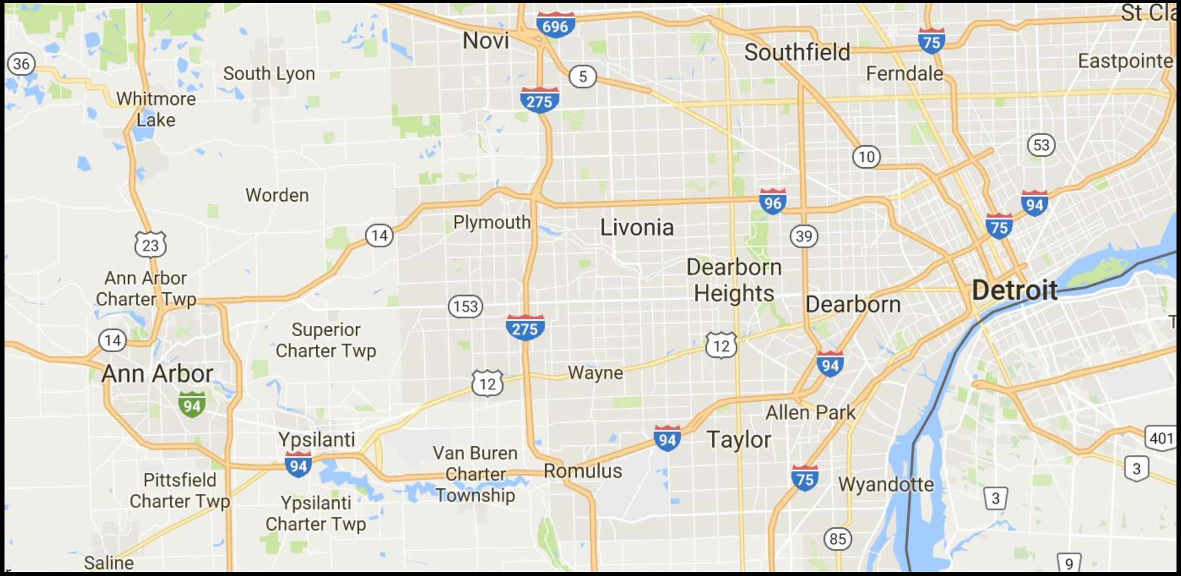


Outline

- PRM
- Sampling strategies
- RRT

Probabilistic Roadmap (PRM)

- Main idea: Build a roadmap of the space from sampled points, search the roadmap to find a path
- Roadmap should capture the *connectivity* of the free space

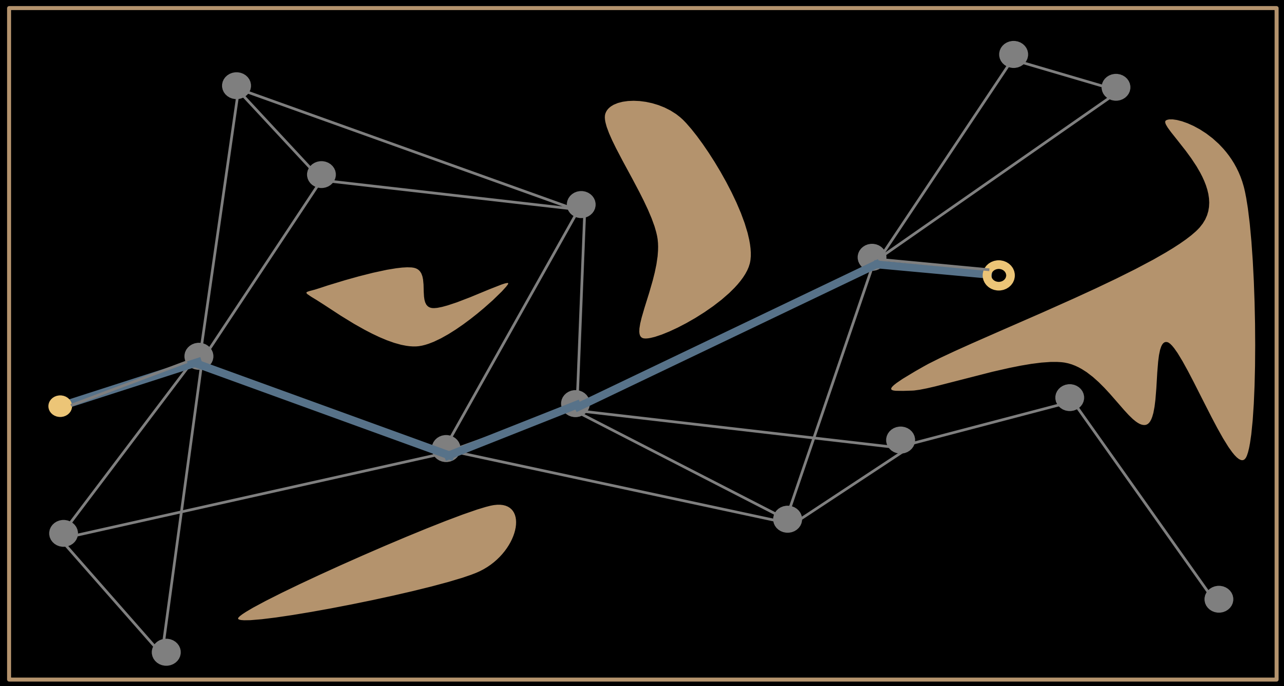


Probabilistic Roadmap (PRM)

- Building a PRM: 2 phase process
- "Learning" Phase (this is not really machine learning)
  - Construction Step
  - Expansion Step (not used in practice today)
- Query Phase
  - Answer a given path planning query

- PRMs are known as *multi-query algorithms*, because roadmap can be re-used if environment and robot haven't changed between queries.

PRM Example



“Learning” Phase

- Construction step: Build the roadmap by sampling random free configurations and connect them using a fast *local planner*
- Store these configurations as nodes in a graph
    - Note: In PRM literature, nodes are sometimes called “milestones”
- Edges of the graph are the paths between nodes found by the local planner

“Learning” Phase: Construction Step

Start with an empty graph G = (V,E)

For i = 1 to MaxIterations

Generate random configuration q

If q is collision-free

Add q to V

Select k nearest nodes to q in V

Attempt connection between each of these nodes and q using local planner

If a connection is successful, add it as an edge in E

"Learning" Phase: Construction Step

Start with an empty graph G = (V,E)

For i = 1 to MaxIterations

  Generate random configuration q

  If q is collision-free

    Add q to V

    Select k nearest nodes to q in V

    Attempt connection between each of these nodes and q using local planner

    If a connection is successful, add it as an edge in E

“Learning” Phase: Sampling Collision-free Configurations

- Easiest and most common: uniform random sampling in C-space
  - Draw random value in allowable range for each DOF, combine into a vector
  - Place robot at the configuration and check collision
  - Repeat above until you get a collision-free configuration
  - AKA “Rejection Sampling”

- MANY ways to do this, many papers published, we will discuss more methods later

“Learning” Phase: Construction Step

Start with an empty graph G = (V,E)

For i = 1 to MaxIterations

Generate random configuration q

If q is collision-free

Add q to V

Select k nearest nodes to q in V

Attempt connection between each of these nodes and q using local planner

If a connection is successful, add it as an edge in E

“Learning” Phase: Finding Nearest Neighbors (NN)

- Need to decide a distance metric $D(q_1,q_2)$ to define “nearest”
- D should reflect likelihood of success of local planner connection (roughly)
  - If $D(q_1,q_2)$ is small, success should be likely
  - If $D(q_1,q_2)$ is large, success should be less likely
- By default, use Euclidian distance:

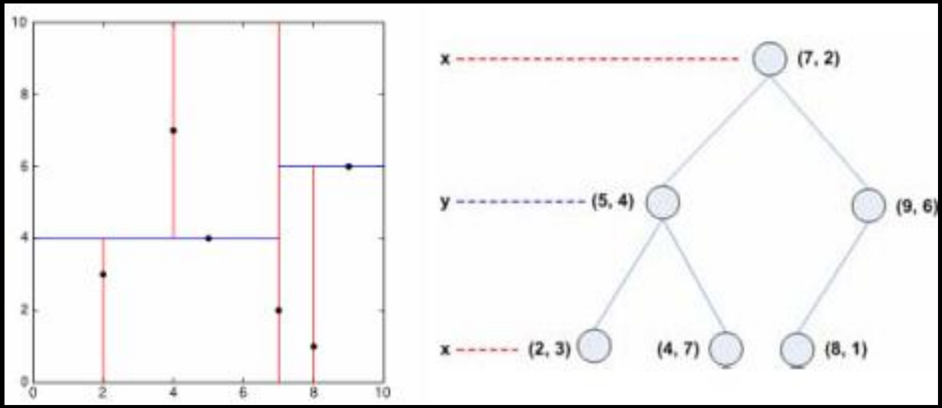
$$D(q_1,q_2) = ||q_1 - q_2||$$

- Can weigh different dimensions of C-space differently
  - Often used to weigh translation vs. rotation

“Learning” Phase: Finding Nearest Neighbors (NN)

- Two popular ways to do NN in PRM
  - Find k nearest neighbors (even if they are distant)
  - Find all nearest neighbors within a certain distance

- Naïve NN computation can be slow with 1000s of nodes, so use *kd-tree* to store nodes and do NN queries
  - A kd-tree is a data-structure that recursively divides the space into bins that contain points (like Oct-tree and Quad-tree)
  - NN then searches through bins (not individual points) to find nearest point
  - Much faster to use kd-tree for large numbers of nodes
  - BUT, cost of constructing a kd-tree is significant, so only regenerate tree once in a while (not for every new node!)
  - kd-tree code is easy to find online



“Learning” Phase: Construction Step

Start with an empty graph G = (V,E)

For i = 1 to MaxIterations

Generate random configuration q

If q is collision-free

Add q to V

Select k nearest nodes to q in V

Attempt connection between each of these nodes and q using local planner

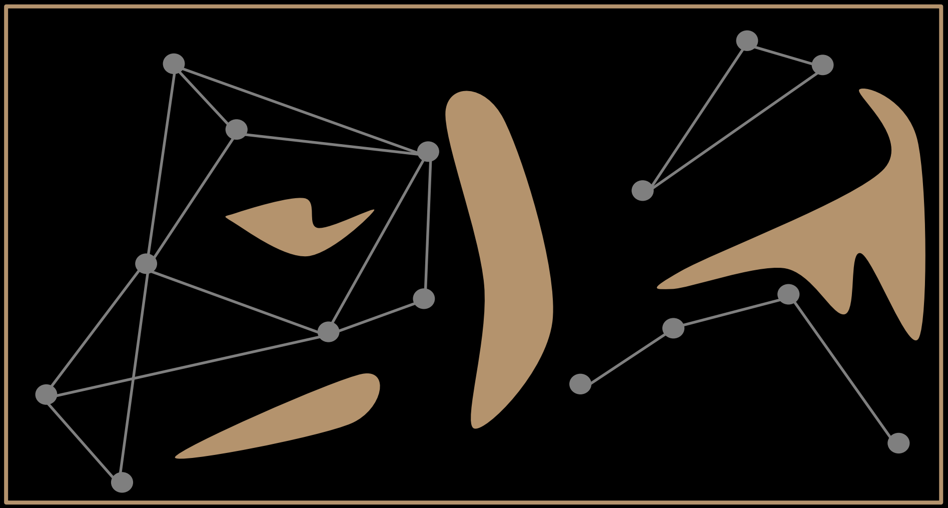
If a connection is successful, add it as an edge in E

“Learning” Phase: Local Planner

- In general, local planner can be anything that attempts to find a path between points, even another PRM!
- BUT, local planner needs to be fast b/c it’s called many times by the algorithm
- Easiest and most common: Connect the two configurations with a straight line in C-space, check that the line is collision-free
    - Advantages:
        - Fast
        - Don’t need to store local paths

“Learning” Phase: Expansion step (not used in practice today)

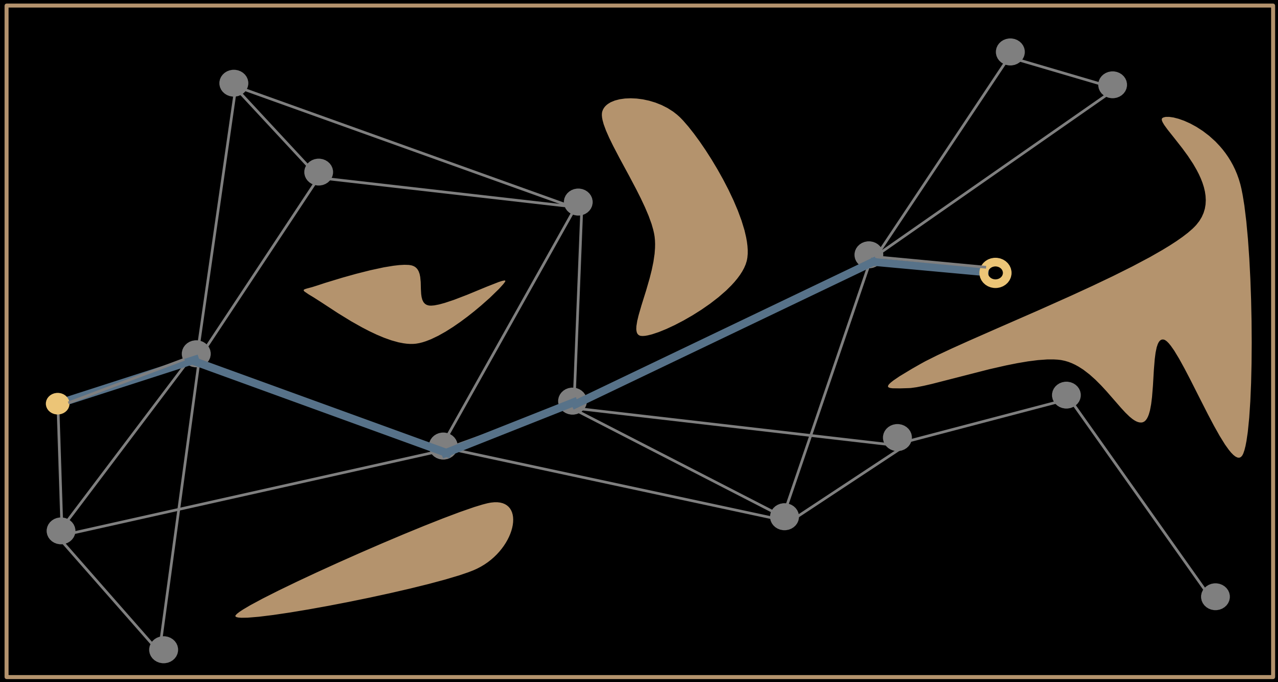
- Problem: Can have disconnected components that should be connected
  - I.e. you haven’t captured the true connectivity of the space



- Expansion step uses heuristics to sample more nodes in an effort to connect disconnected components
  - Unclear how to do this the “right” way, very environment-dependent
  - Not always used in modern implementations

Query Phase

- Given a start q_s and goal q_g
  1. Connect them to the roadmap using local planner
     - May need to try more than k nearest neighbors before connection is made
  2. Search G to find shortest path between q_s and q_g using A*/Dijkstra's/etc.



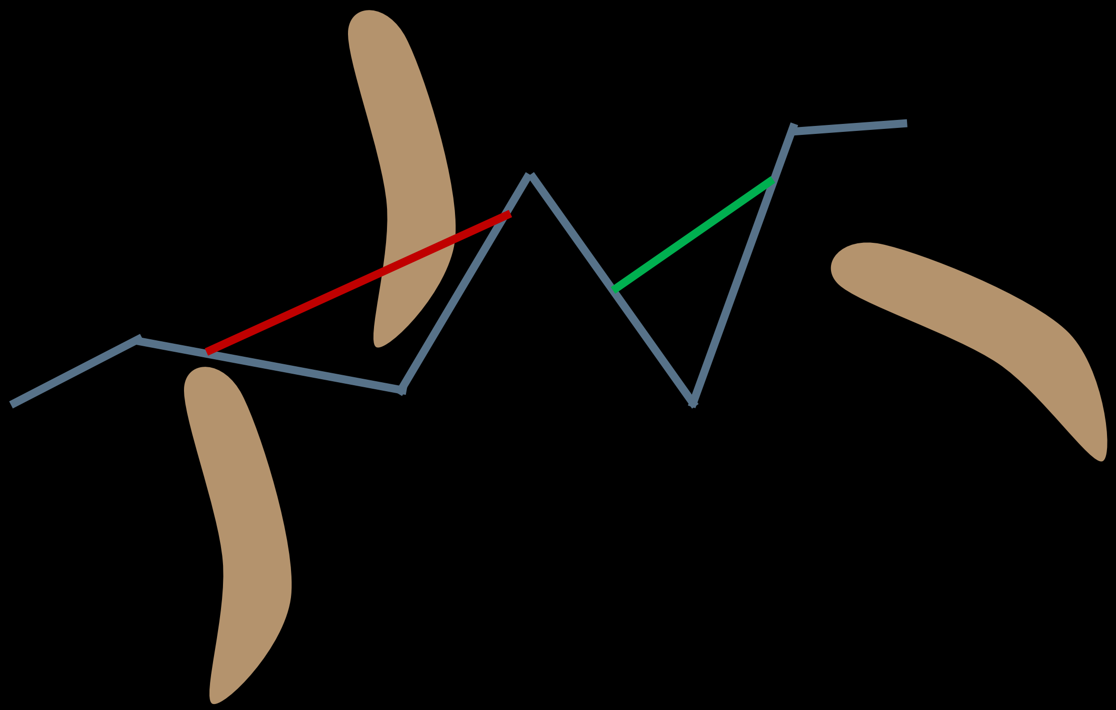
Path Shortening / Smoothing

- Don’t even think of executing a path generated by a sampling-based planner without smoothing it!!!

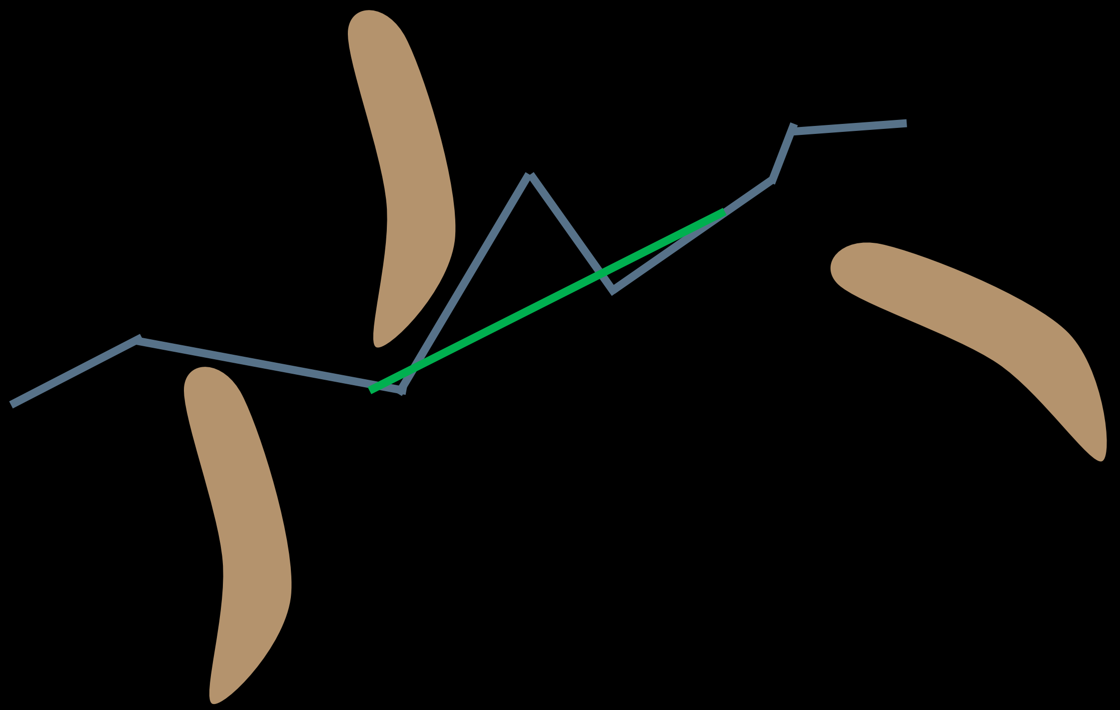
Shortcut Smoothing

```
For i = 0 to MaxIterations
    Pick two points, q1 and q2, on the path randomly
    Attempt to connect (q1, q2) with a line segment
    If successful, replace path between q1 and q2 with the line segment
```

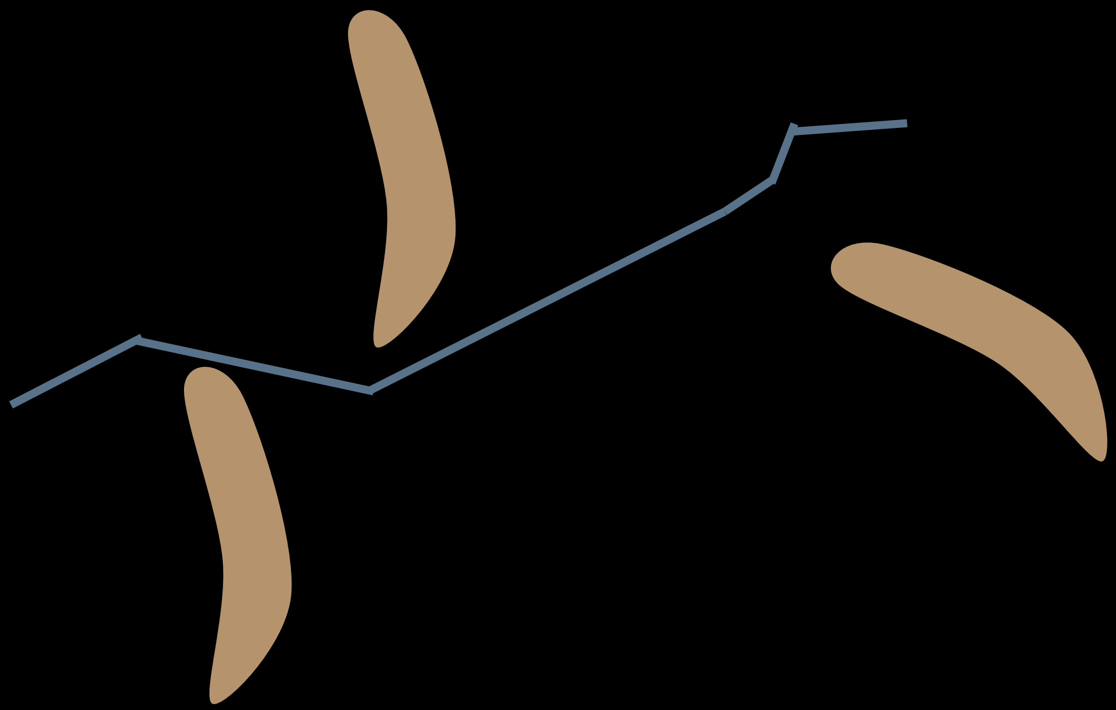
Shortcut Smoothing



Shortcut Smoothing

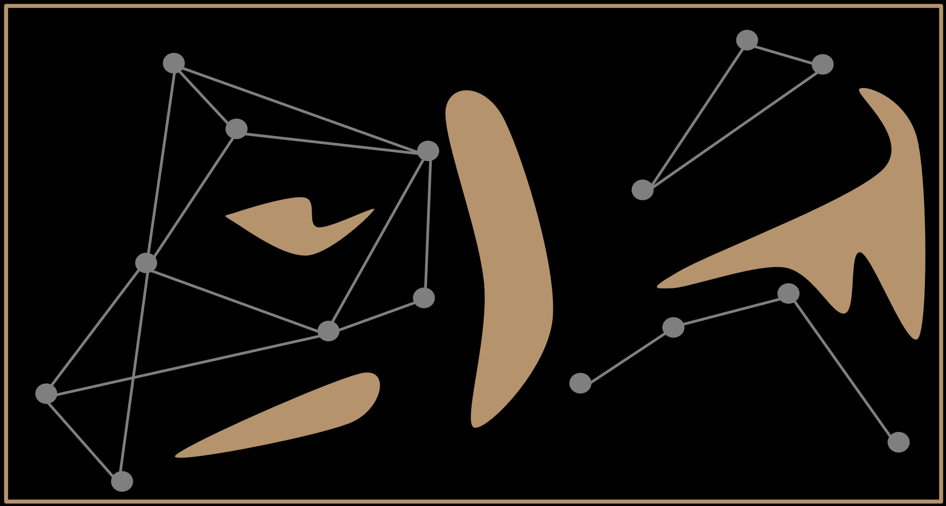


Shortcut Smoothing



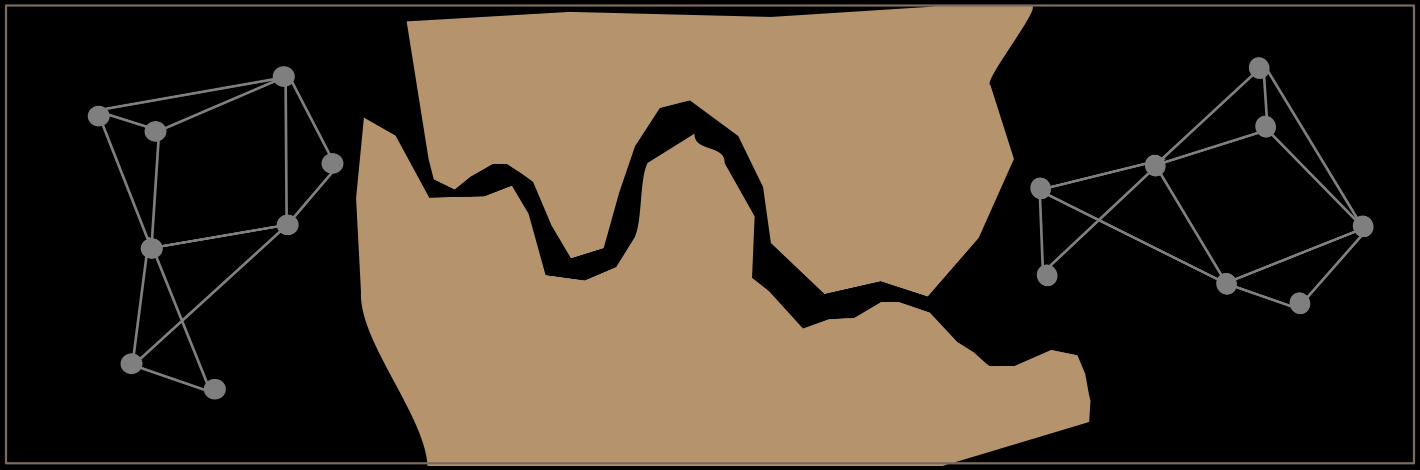
PRM Failure Modes

1. Can’t connect q_s and q_g to any nodes in the graph
   - Come up with an example in the graph below
2. Can’t find a path in the graph but a path is possible
   - Come up with in example in the graph below



Why do failures happen?

- Roadmap doesn't capture connectivity of space, to address this
  - Can run the learning phase longer
  - Can change sampling strategy to focus on narrow passages



- Local planner is too simple, to address this
  - Can use more sophisticated local planner

Completeness

- Complete algorithms are slow.
  - A **complete** algorithm finds a path if one exists and reports no otherwise.
  - Example: Visibility graph
- Heuristic algorithms are unreliable.
  - Example: potential field
- **Probabilistic completeness**
  - Intuition: If there is a solution path, the algorithm will find it with high probability.

Probabilistic Completeness

In an expansive space*, the probability that a PRM planner fails to find a path when one exists goes to 0 exponentially in the number of milestones (~ running time).

[Kavraki, Latombe, Motwani, Raghavan,95]
[Hsu, Latombe, Motwani, 97]

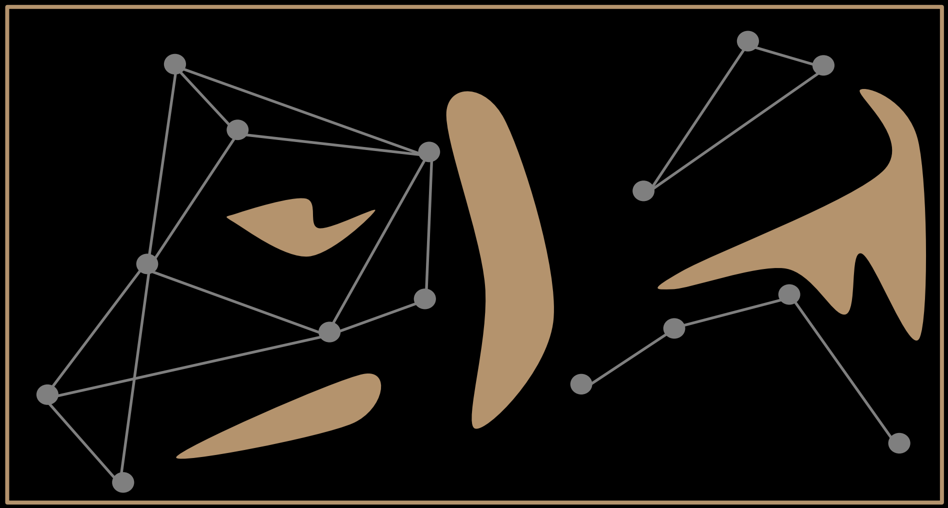
*Roughly, an expansive space is one where there are no infinitely-thin parts of free space.

What happens in the limit for PRM?

- What if we ran the construction step of the PRM for infinite time…
  - What would the graph look like?
  - Would it capture the connectivity of the free space?
  - Would any collision-free start and goal be able to connect to the graph?
  - Is the PRM algorithm probabilistically complete?

Break

PRM issues



- Two issues with the PRM:
  1. Uniform random sampling misses narrow passages
  2. Exploring whole space, but all we want is a path

Sampling Strategies

- Most common is uniform random sampling
  - The bigger the area, the more likely it will be sampled
  - Problem: Narrow passages

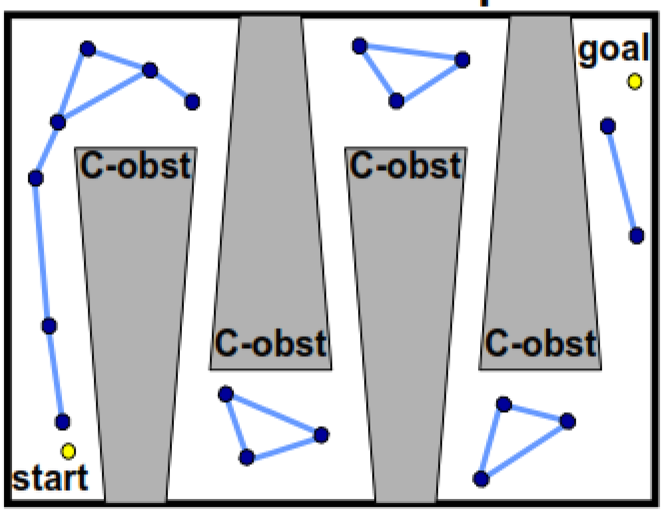
- Are narrow passages inherently bad?
  - Does A* running on a 2D grid have problems with narrow passages?

OBPRM: An Obstacle-Based PRM

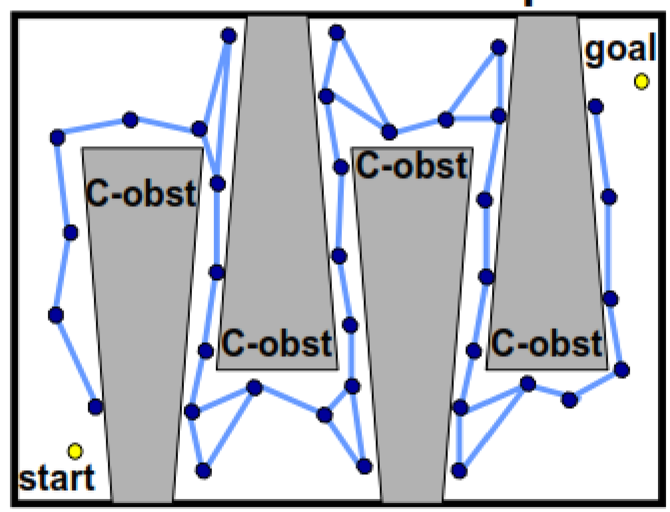
To Navigate Narrow Passages we must sample in them

- most PRM nodes are where planning is easy (not needed)

PRM Roadmap



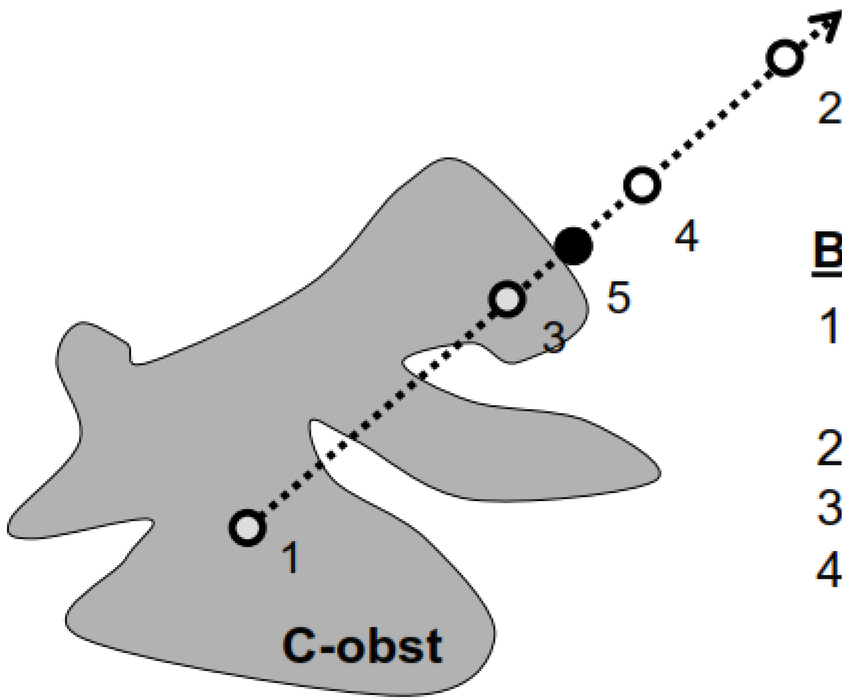
OBPRM Roadmap



Idea: Can we sample nodes near C-obstacle surfaces?

- we cannot explicitly construct the C-obstacles...

OBPRM: Finding Points on C-obstacles

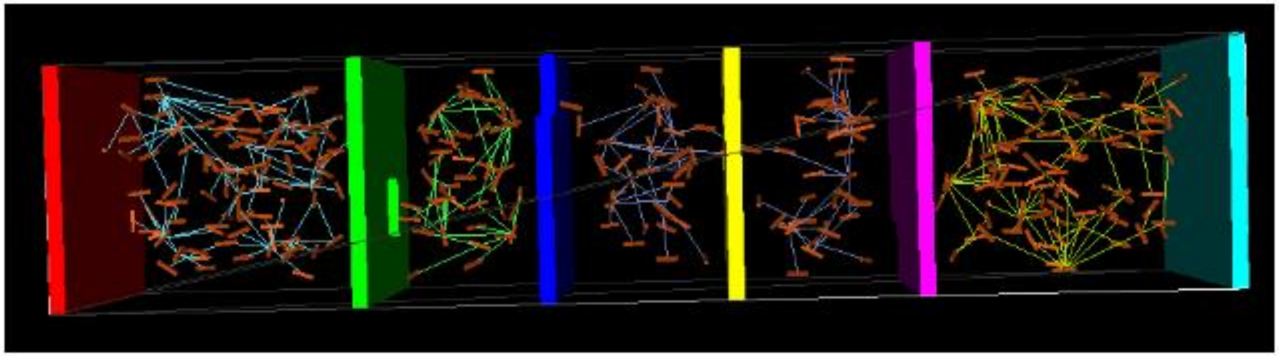


Basic Idea (for workspace obstacle S)

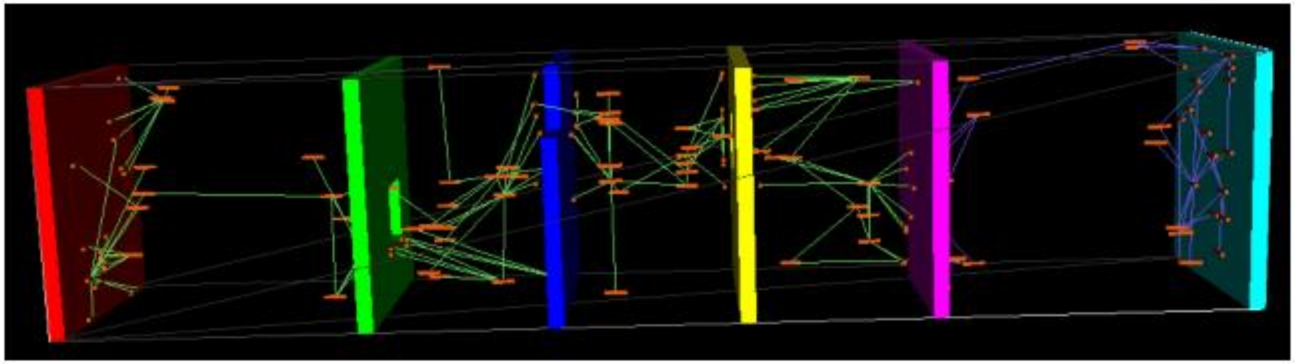
1. Find a point in S's C-obstacle (robot placement colliding with S)
2. Select a random direction in C-space
3. Find a free point in that direction
4. Find boundary point between them using binary search (collision checks)

Note: we can use more sophisticated heuristics to try to cover C-obstacle

PRM vs OBPRM Roadmaps



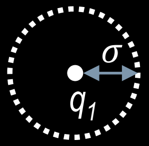
PRM
- 328 nodes
- 4 major CCs



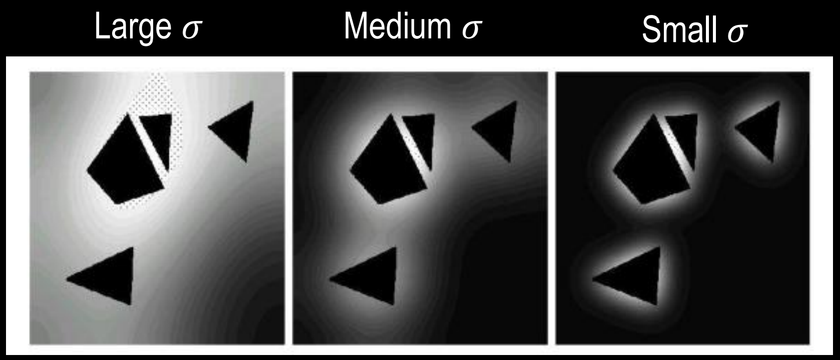
OBPRM
- 161 nodes
- 2 major CCs

Sampling strategies: Gaussian

- Gaussian sampler
  - Pick a q_1
  - Pick a q_2 from a Gaussian distribution centered at q_1



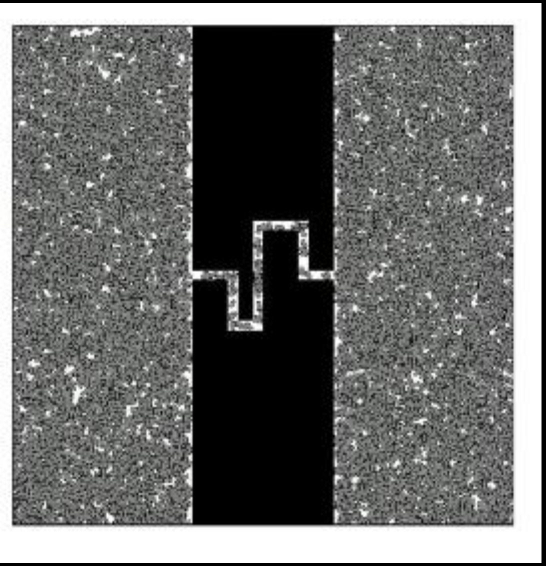
  - If **both** are in collision or collision-free, discard them, if one free, keep it



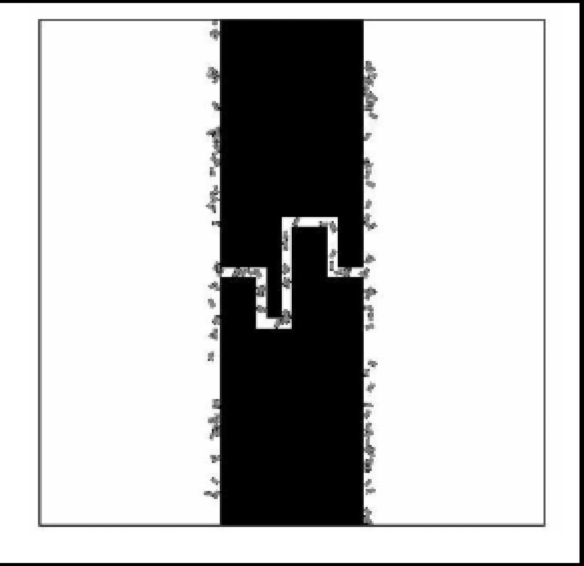
Sampling distribution for varying σ
(width decreasing from left to right)

Sampling Strategies: Gaussian

- Performs well in narrow passages



Uniform Random Sampling



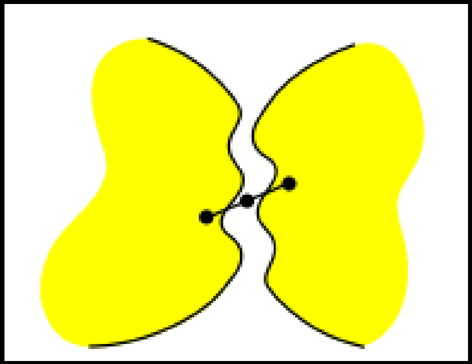
Gaussian Sampling

Sampling Strategies

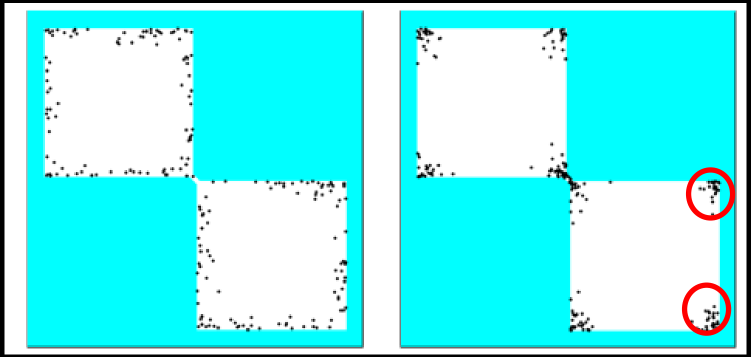
- Can we come up with a case where obstacle-biased sampling is worse than uniform random sampling?

Sampling Strategies: Bridge

- Sample a q_1 that is in collision
- Sample a q_2 in neighborhood of q_1 using some probability distribution (e.g. gaussian)
- If q_2 in collision, get the midpoint of (q_1, q_2)
- Check if midpoint is in collision, if not, add it as a node



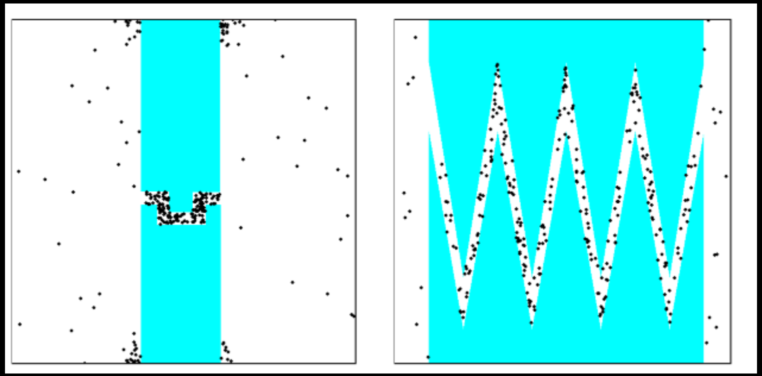
Sampling Strategies: Bridge



Gaussian Sampling

Bridge Sampling

What's going on at the corners?



Bridge Sampling performs well in narrow passages

Rapidly-exploring Random Trees (RRTs)

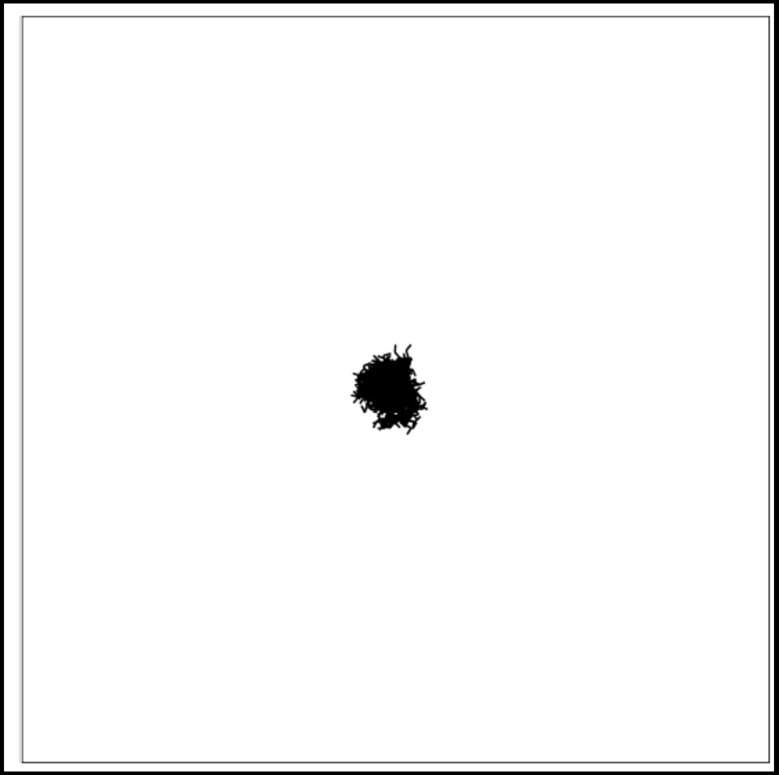
Single-query methods

- Motivation: Why try to capture the connectivity of the whole space when all you need is one path?
- Algorithms:
  - Single-Query BiDirectional Lazy PRM (SBL-PRM)
  - Expansive Space Trees (EST)
  - **Rapidly-exploring Random Tree (RRT)**
    - AKA "RDT" in the book
- Key idea: Build a *tree* instead of a general graph.
- The tree grows in C_{free}
  - Like PRM, captures some connectivity
  - Unlike PRM, only explores what is connected to q_{start}

Naïve Tree Algorithm

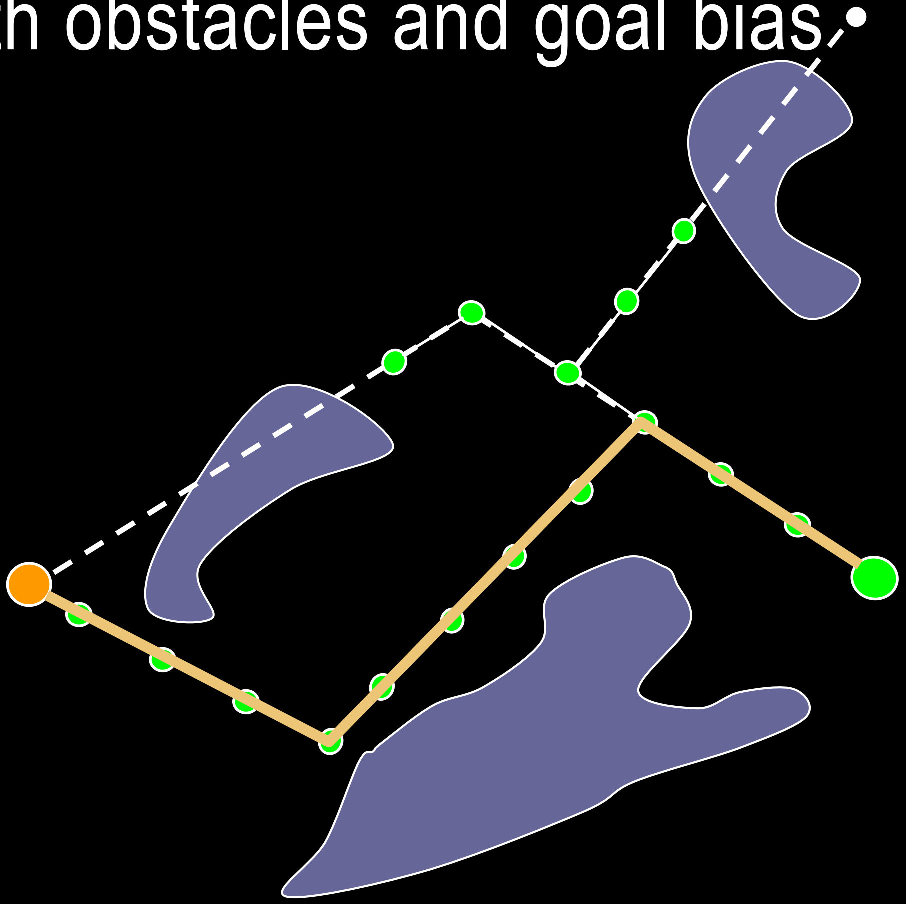
$q_{node} = q_{start}$

```
For i = 1 to NumberSamples
        qrand = Sample near qnode
        Add edge e = (qrand , q) if collision-free
        qnode = Pick random node of tree
```



RRT Growing in Empty Space

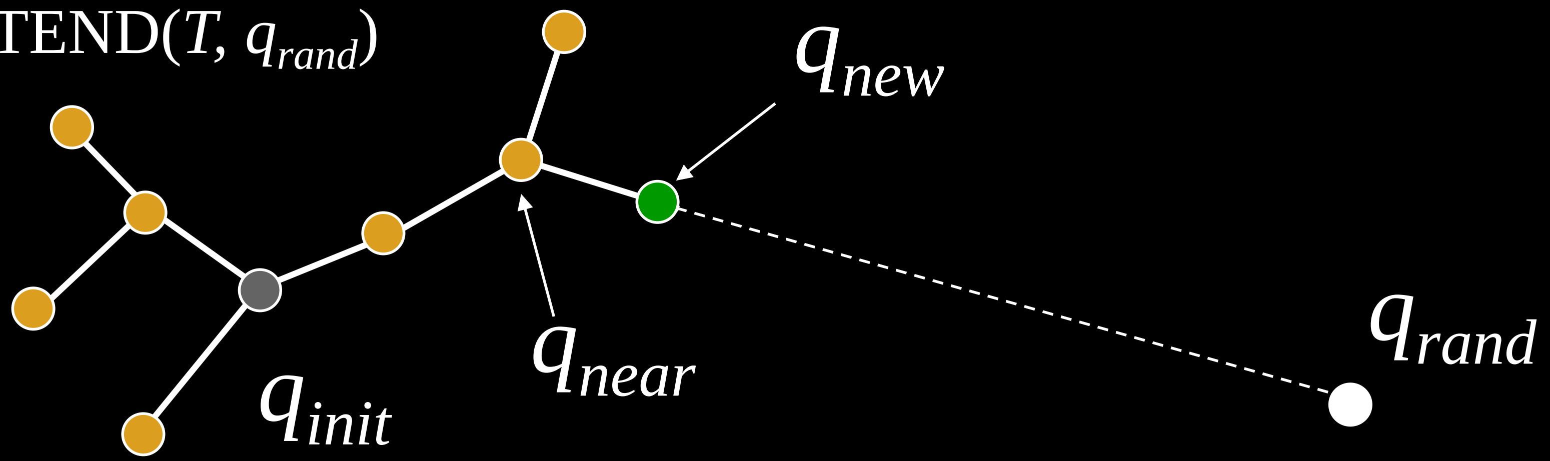
RRT with obstacles and goal bias



Path Planning with Rapidly-Exploring Random Trees (RRTs)

BUILD_RRT (q_{init}) {
  $T.init(q_{init})$;
  for k = 1 to K do
    q_{rand} = RANDOM_CONFIG();
    EXTEND(T, q_{rand})
}

EXTEND(T, q_{rand})



RRT Goal Biasing

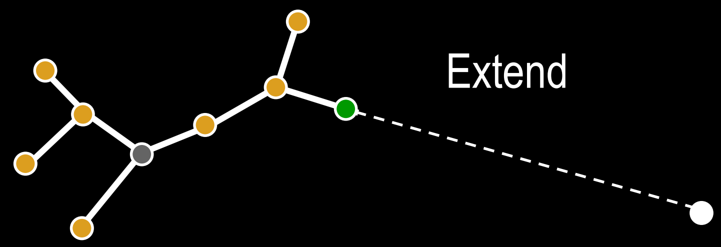
- In “pure” form RRTs are great at filling space, but we need a path!

- Need to bias RRTs toward goal to produce a path
  - When generating a random sample, with some probability pick the goal instead of a random node
  - This introduces another parameter
  - James Kuffner’s experience is that 5-10% is the right choice

- What happens if you set probability of sampling goal to 100%?

RRT Extension Types

- RRT-Extend
  - Take one step toward a random sample



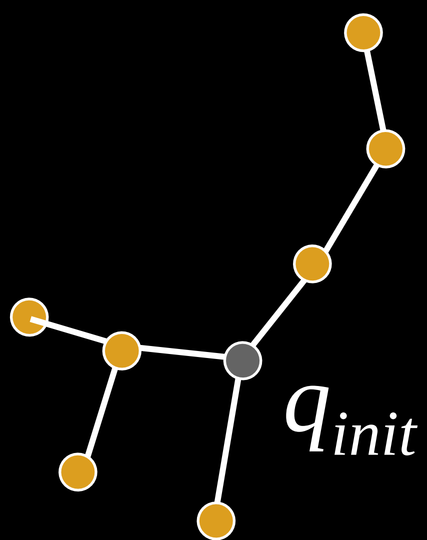
- RRT-Connect
  - Step toward random sample until it is either
    - Reached
    - You hit an obstacle

BiDirectional RRTs

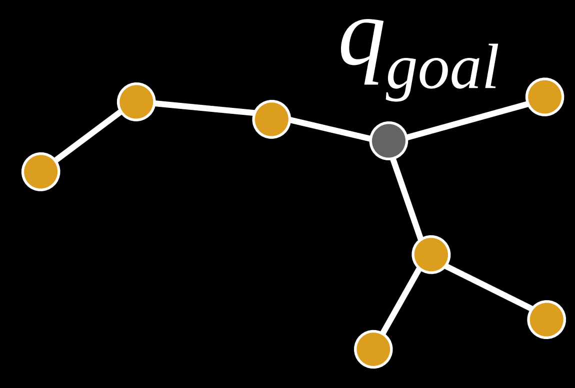
- BiDirectional RRT
  - Grow trees from both start and goal
  - Try to get trees to connect to each other
  - Trees can both use Extend or both use Connect or one use Extend and one Connect

- BiDirectional RRT with Connect for both trees is my favorite, I always try this first
  - This variant has only one parameter; the step size

Example of BiDirectional RRT

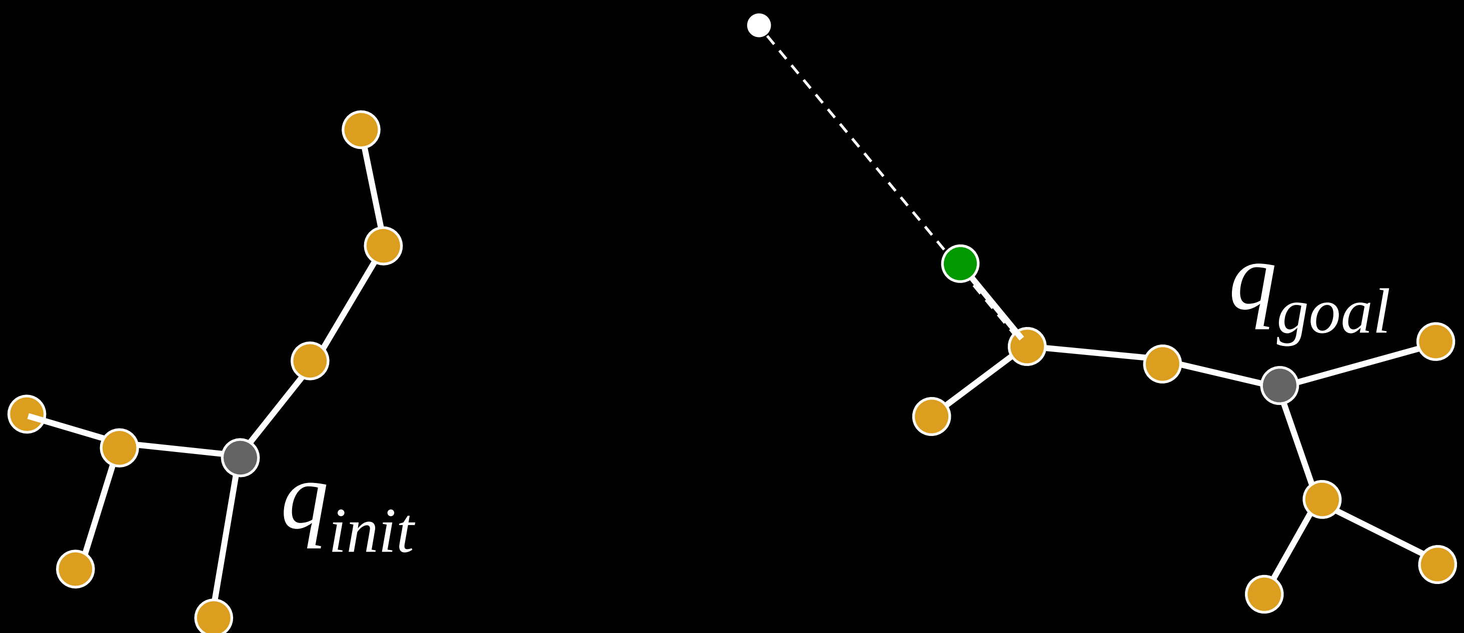


Connect



Extend

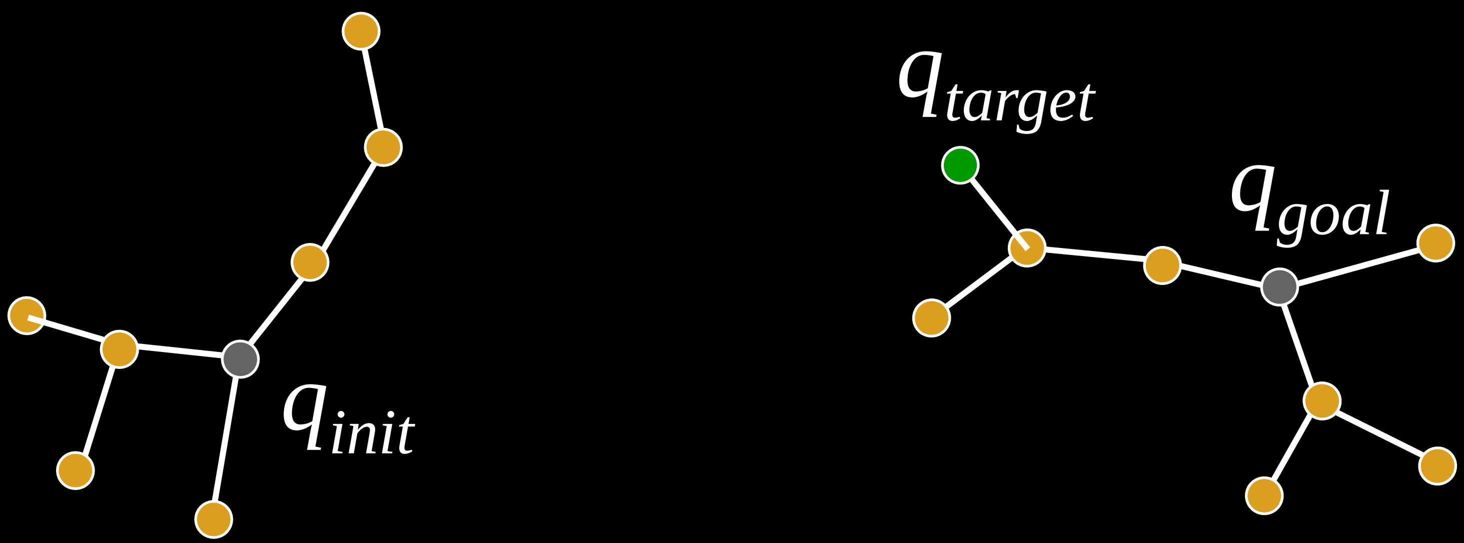
1) One tree grown using random target



Connect

Extend

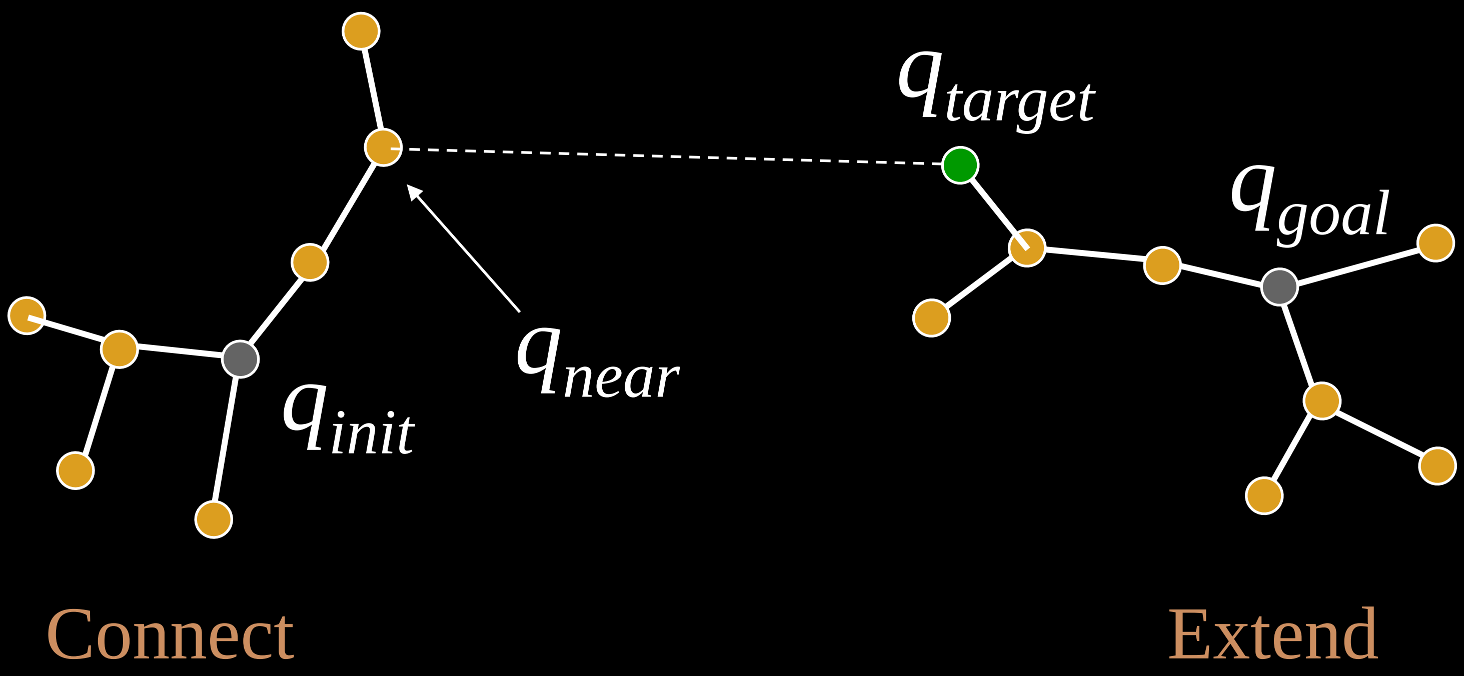
2) New node becomes target for other tree



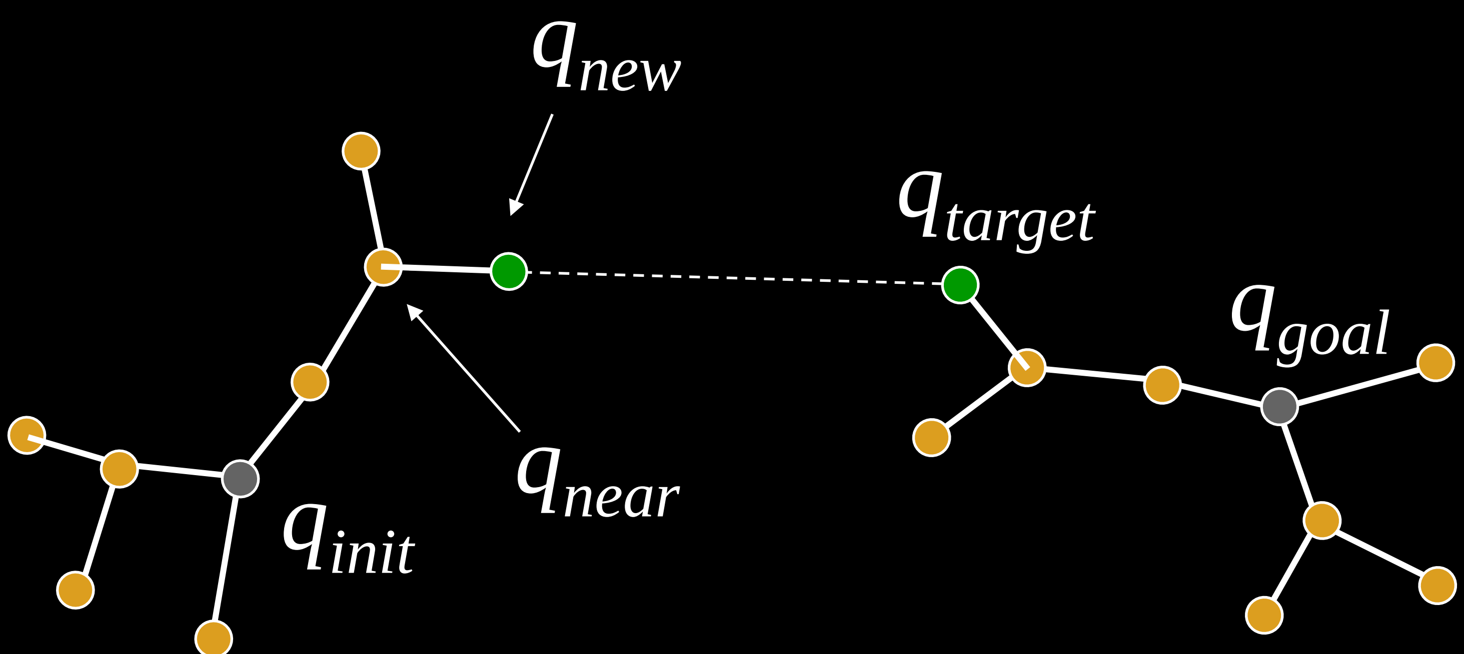
Connect

Extend

3) Calculate node "nearest" to target



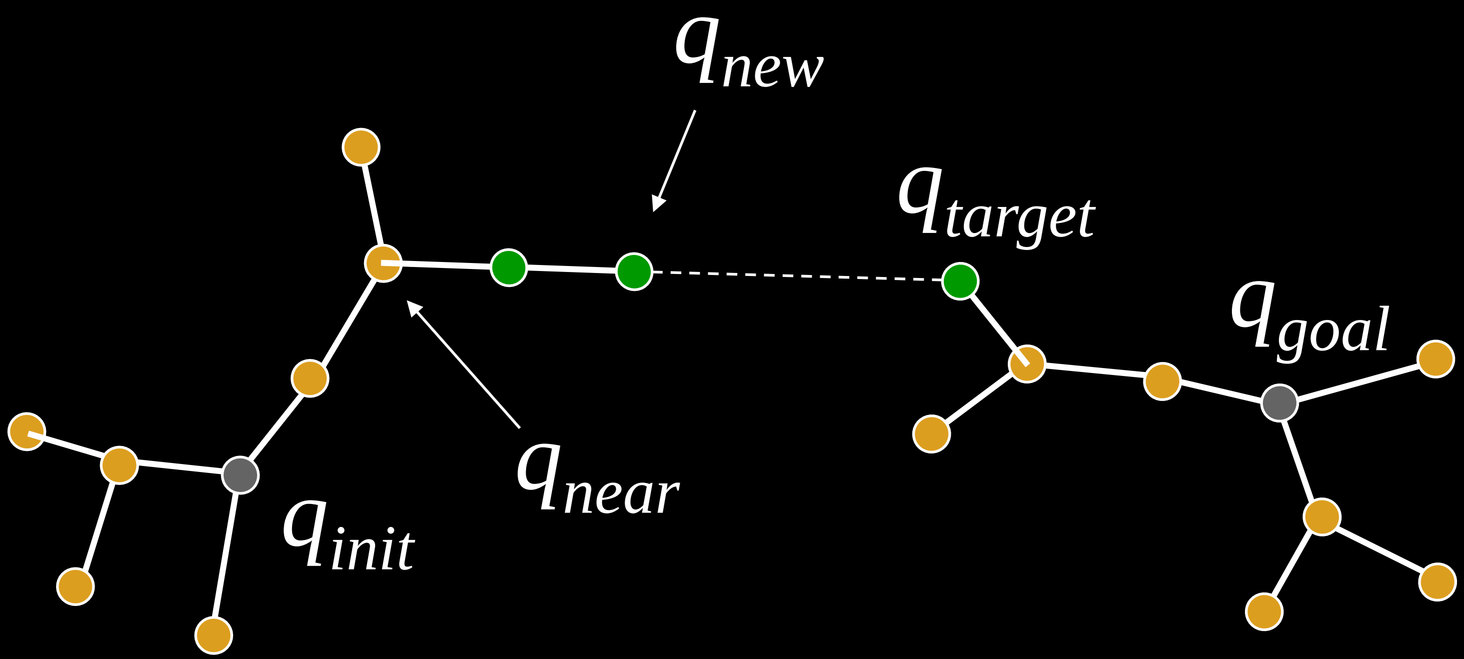
4) Try to add new collision-free branch



Connect

Extend

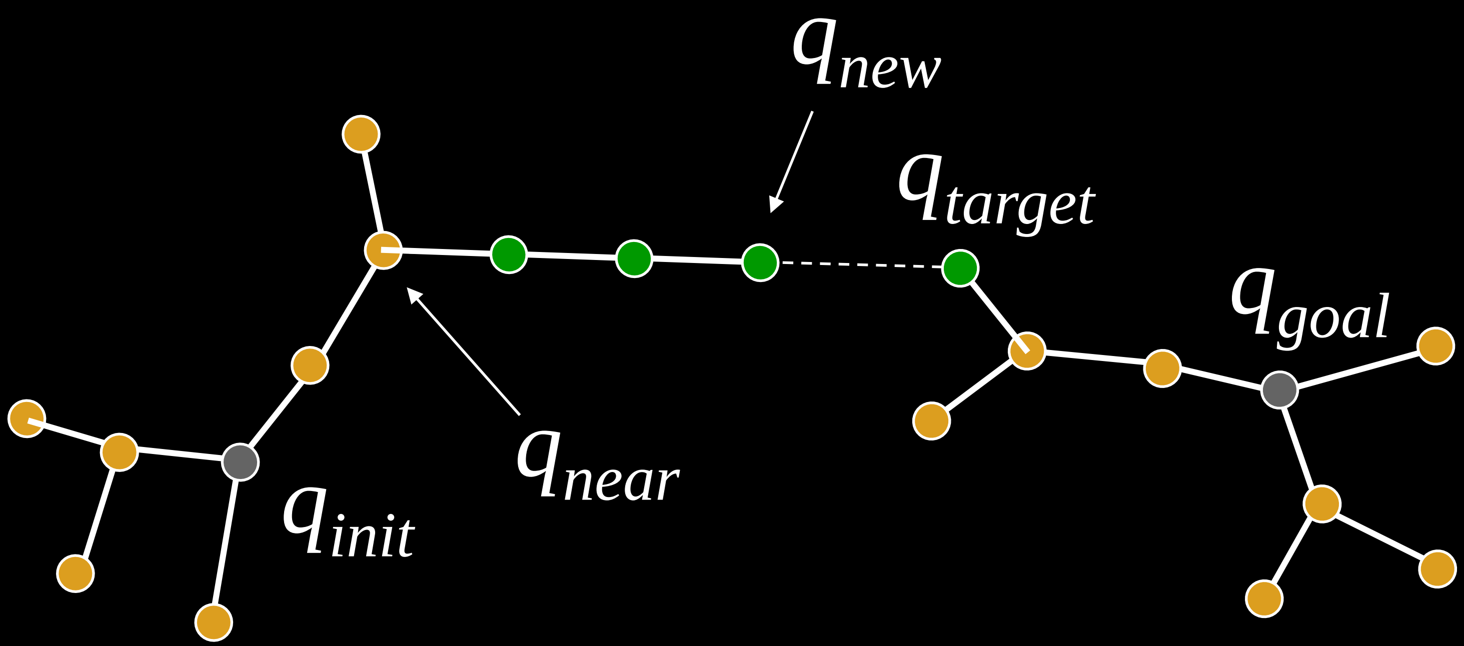
5) If successful, keep extending branch



Connect

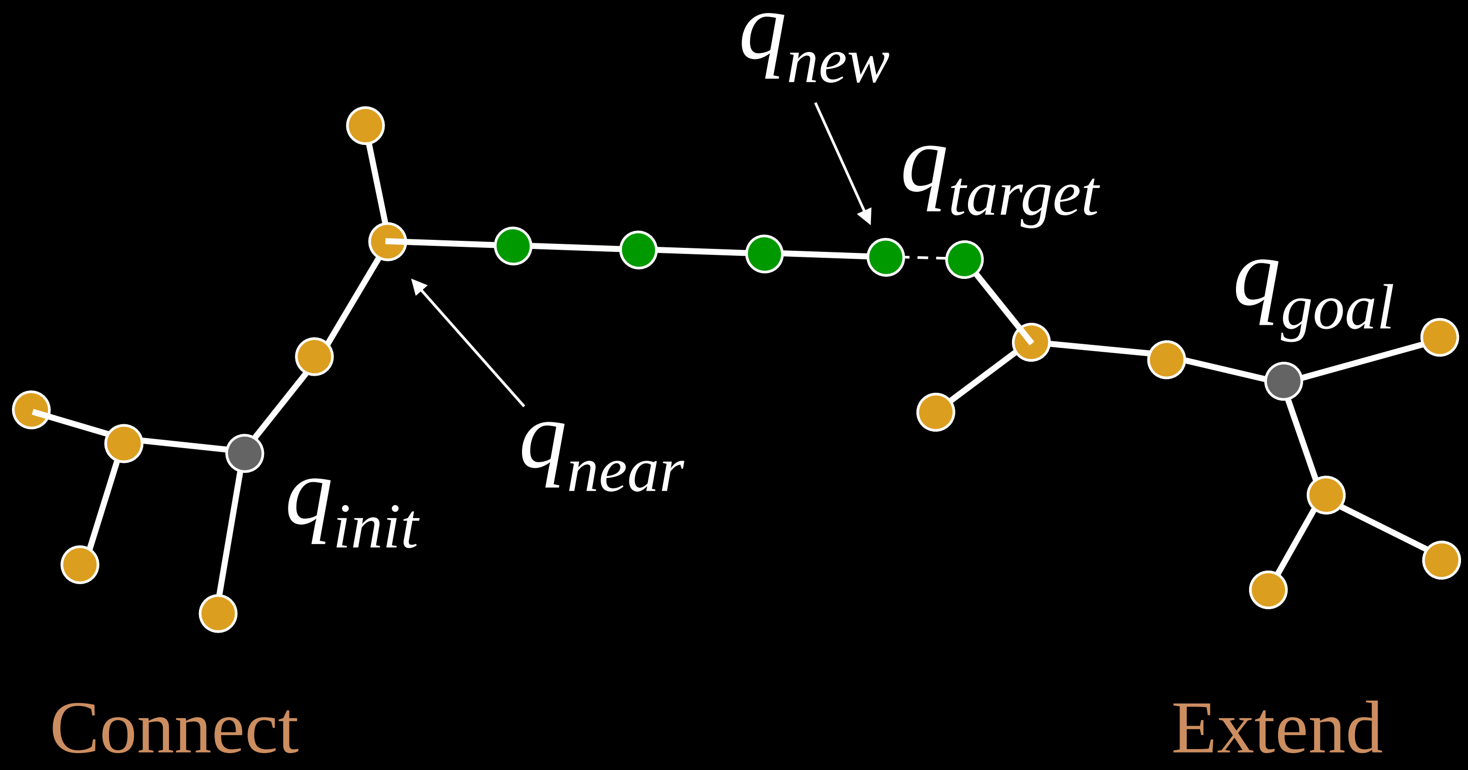
Extend

5) If successful, keep extending branch

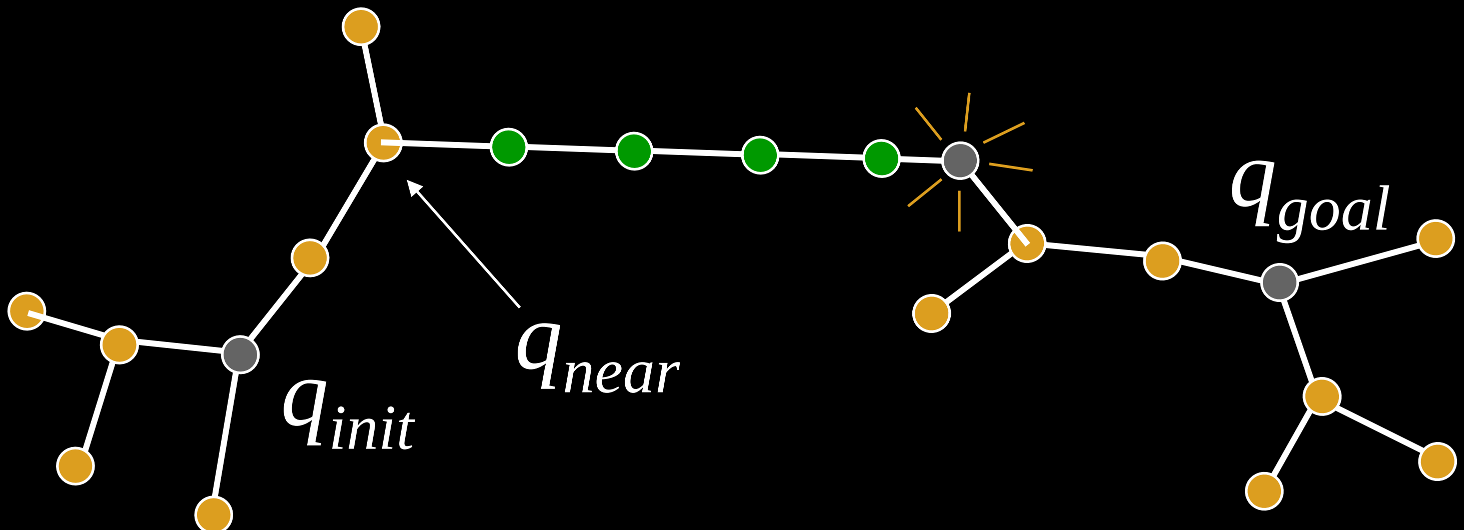


Connect Extend

5) If successful, keep extending branch

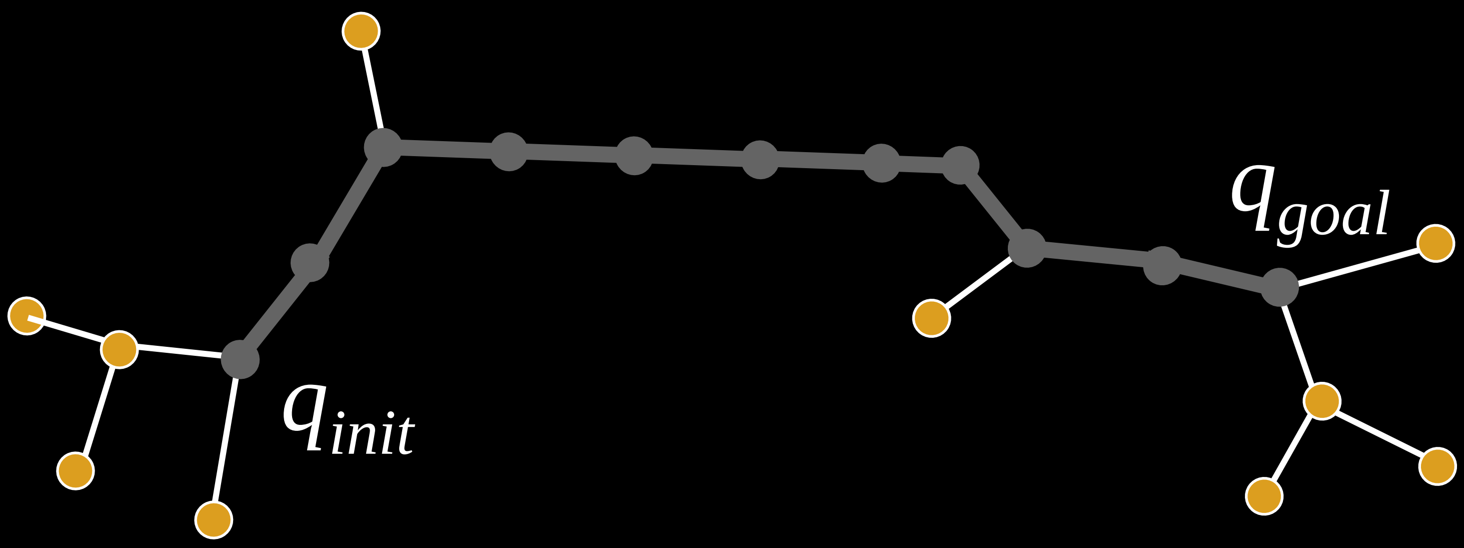


6) Path found if branch reaches target



Connect Extend

7) Return path connecting start and goal

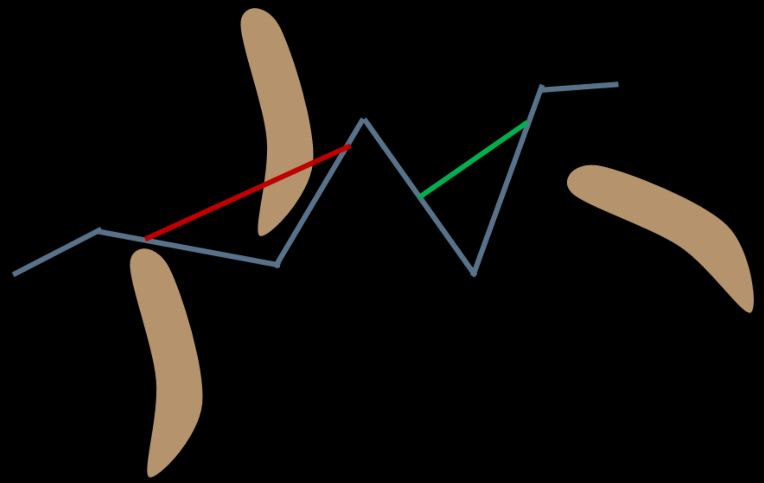


Connect

Extend

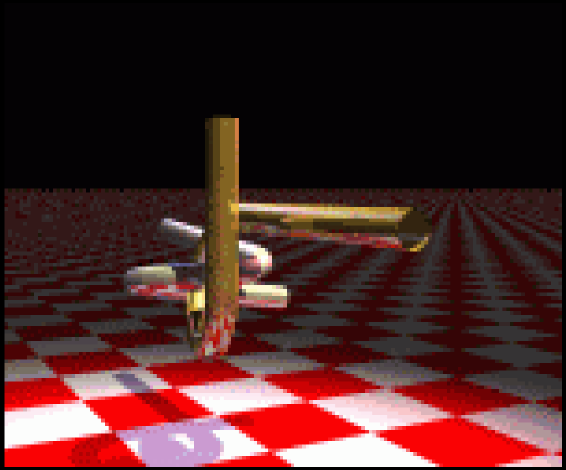
Path Smoothing/Optimization

- RRTs produce notoriously bad paths
  - Not surprising since no consideration of path quality
- ALWAYS smooth/optimize the returned path
  - Many methods exists, e.g. shortcut smoothing



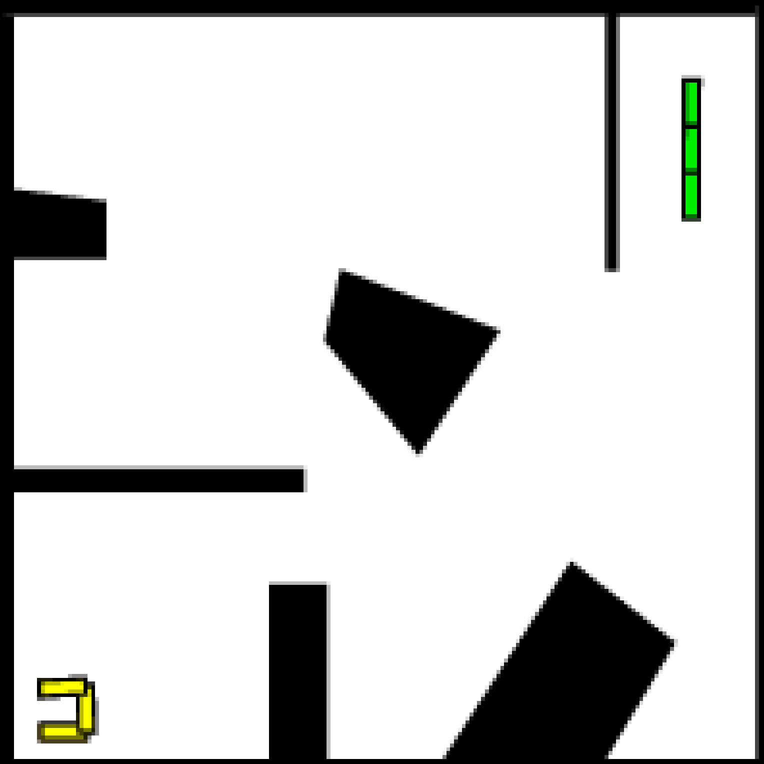
RRT Examples: The Alpha Puzzle

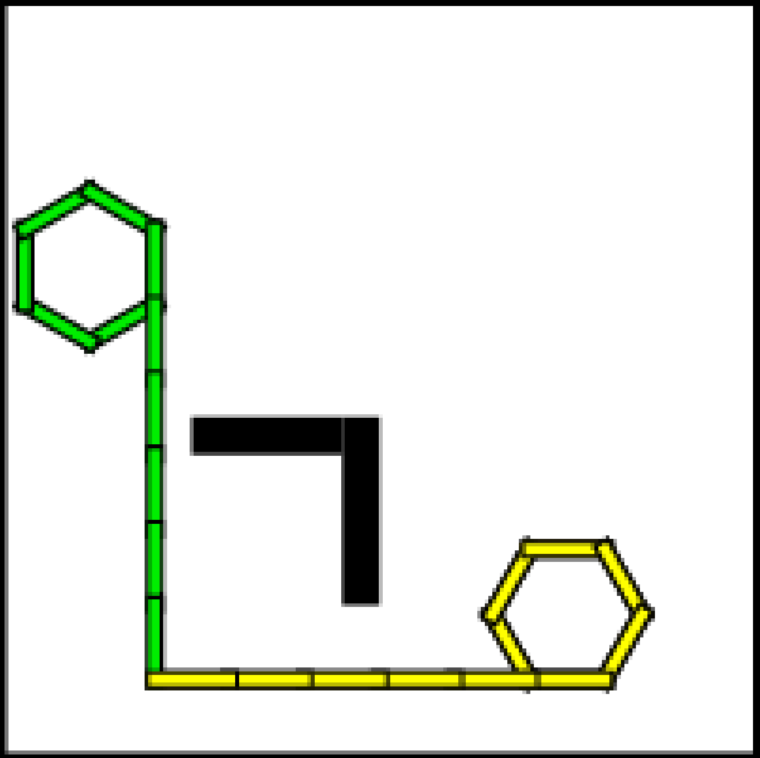
- VERY hard 6DOF motion planning problem (long, winding narrow passage)



- “*In 2001, it was solved by using a balanced bidirectional RRT, developed by James Kuffner and Steve LaValle. There are no special heuristics or parameters that were tuned specifically for this problem. On a current PC (circa 2003), it consistently takes a few minutes to solve*” –RRT website
- RRT became famous in large part because it was able to solve this puzzle

RRT Examples: Articulated Objects





RRT Analysis

The limiting distribution of vertices:

- THEOREM: $\boldsymbol{X_k}$ converges to $\boldsymbol{X}$ with probability 1 as time goes to infinity

  $\boldsymbol{X_k}$: The RRT vertex distribution at iteration k

  $\boldsymbol{X}$: The distribution used for generating samples

- If using uniform distribution, tree nodes converge to the free space
- Based on this, we can prove that RRT is probabilistically complete

Summary: Sampling-Based Planning

- The good:
  - Provides fast *feasible* solution
  - Popular methods have few parameters
  - Works on practical problems
  - Works in high-dimensions
  - Works even with the wrong distance metric

Summary: Sampling-Based Planning

- The bad:
  - No quality guarantees on paths*
    - In practice: smooth/optimize path afterwards
  - No termination when there is no solution
    - In practice: set an arbitrary timeout
  - Probabilistic completeness is a weak property
    - Completeness in high-dimensions is impractical

More recent methods have asymptotic optimality guarantees (e.g. RRT)

Homework

- LaValle Ch. 14-14.5