CS 4610/5335 – Lecture 7
Sampling-based Motion Planning

Lawson L.S. Wong
Northeastern University
2/9/22

Material adapted from:
1. Robert Platt, CS 4610/5335
2. Peter Corke, Robotics, Vision and Control
3. Marc Toussaint, U. Stuttgart Robotics Course
4. Steven M. LaValle, Planning Algorithms
5. Frank Dellaert, Georgia Tech CS 6601
6. Emilio Frazzoli, MIT 16.410/16.413

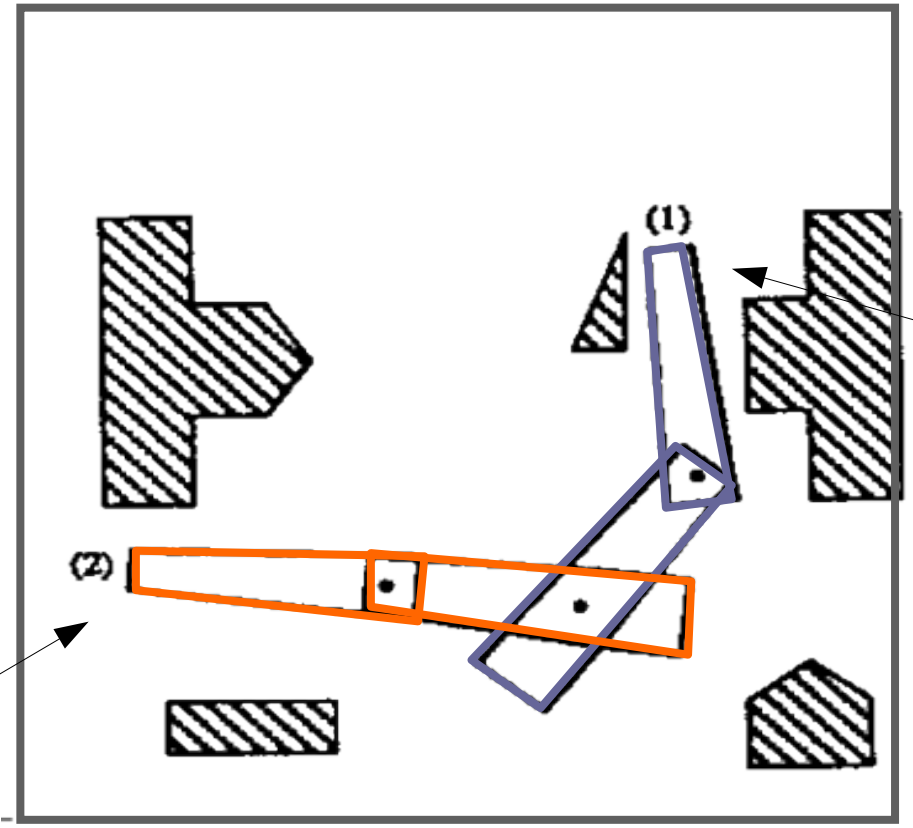
Announcements

Project

- If you have already formed a team
  (or possibly just looking for a final member),
  create a public Piazza thread for your project
- See Piazza @35 for things you should include
  (you can fill these out gradually)

Problem we want to solve this week

Given: Description of the robot and obstacles
Find: Path from start to goal that is collision-free



Starting configuration

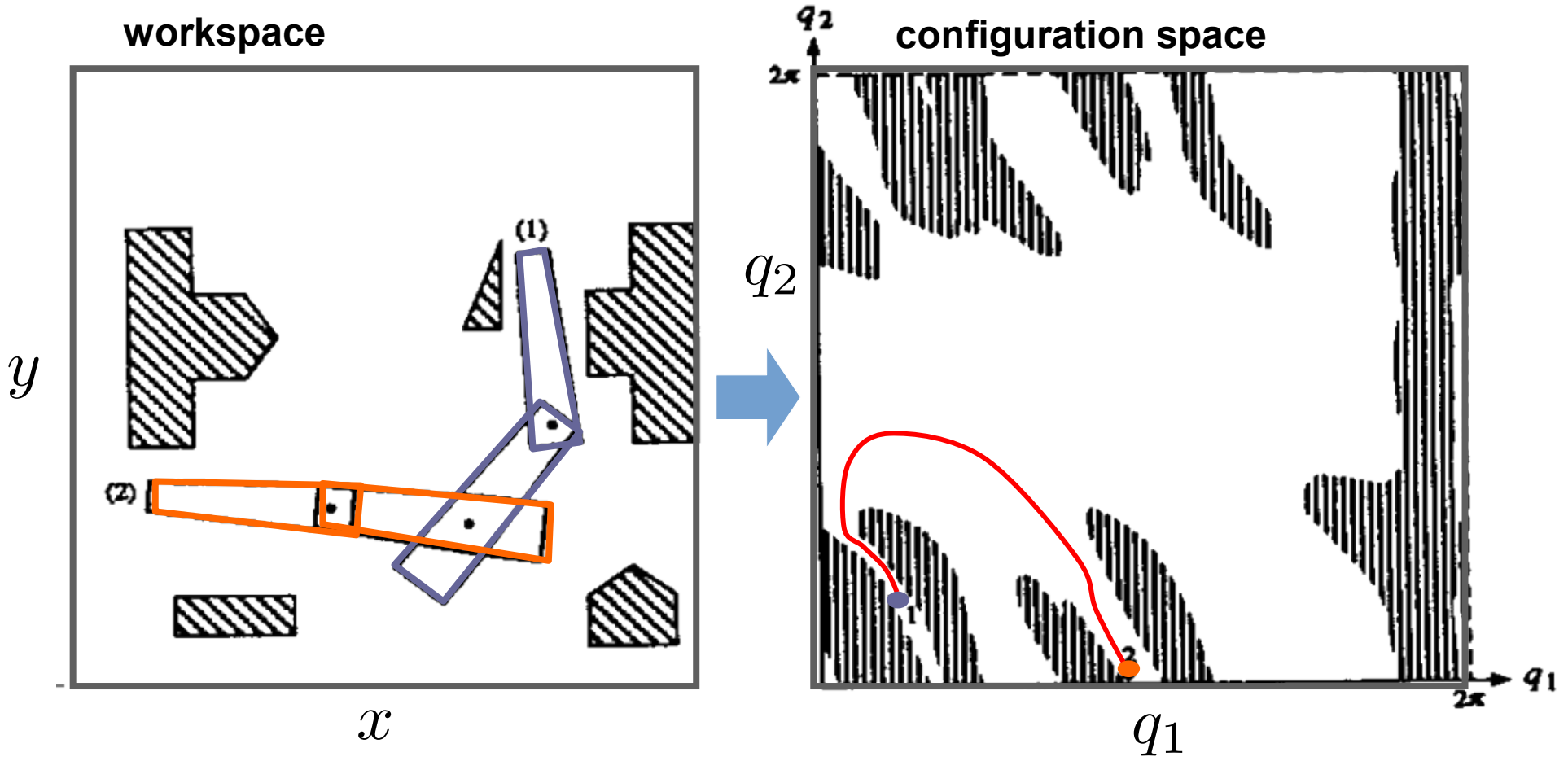
Goal configuration

Planning in “configuration space”

Given: Description of the robot and obstacles
Find: Path from start to goal that is collision-free

Key idea: Convert original planning problem
into a planning problem for a single point

Configuration space



Original problem
– plan path for robot arm

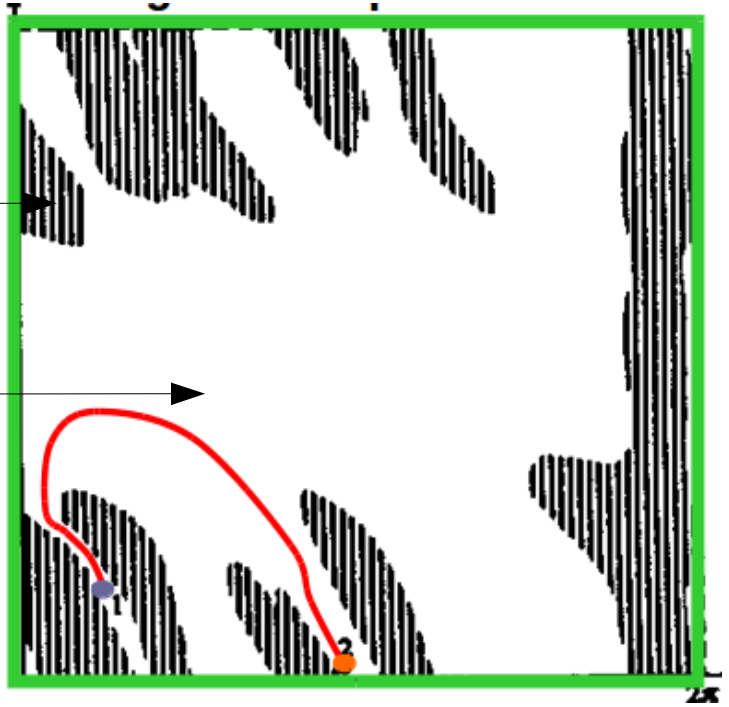
Equivalent problem:
– plan path for a point

Formalization of the motion planning problem

Configuration space: $\mathcal{C}$

Obstacle space: $\mathcal{C}_{obs}$

Free space: $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$



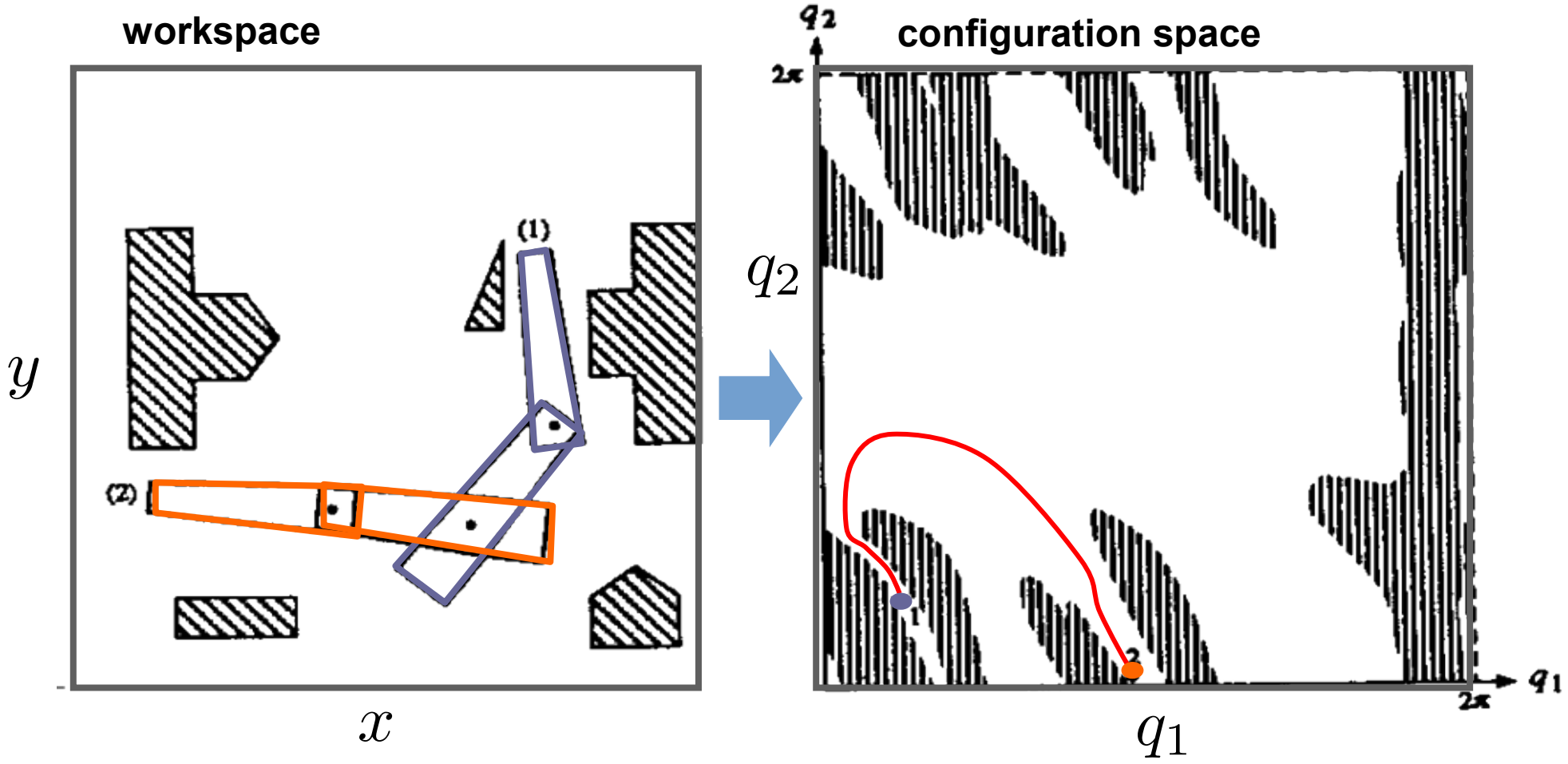
Path: $\sigma : [0,1] \rightarrow \mathcal{C}$ where σ must be continuous

Collision-free path: $\sigma(\tau) \in \mathcal{C}_{free}, \tau \in [0,1]$

$$\sigma(0) = x_{init}$$

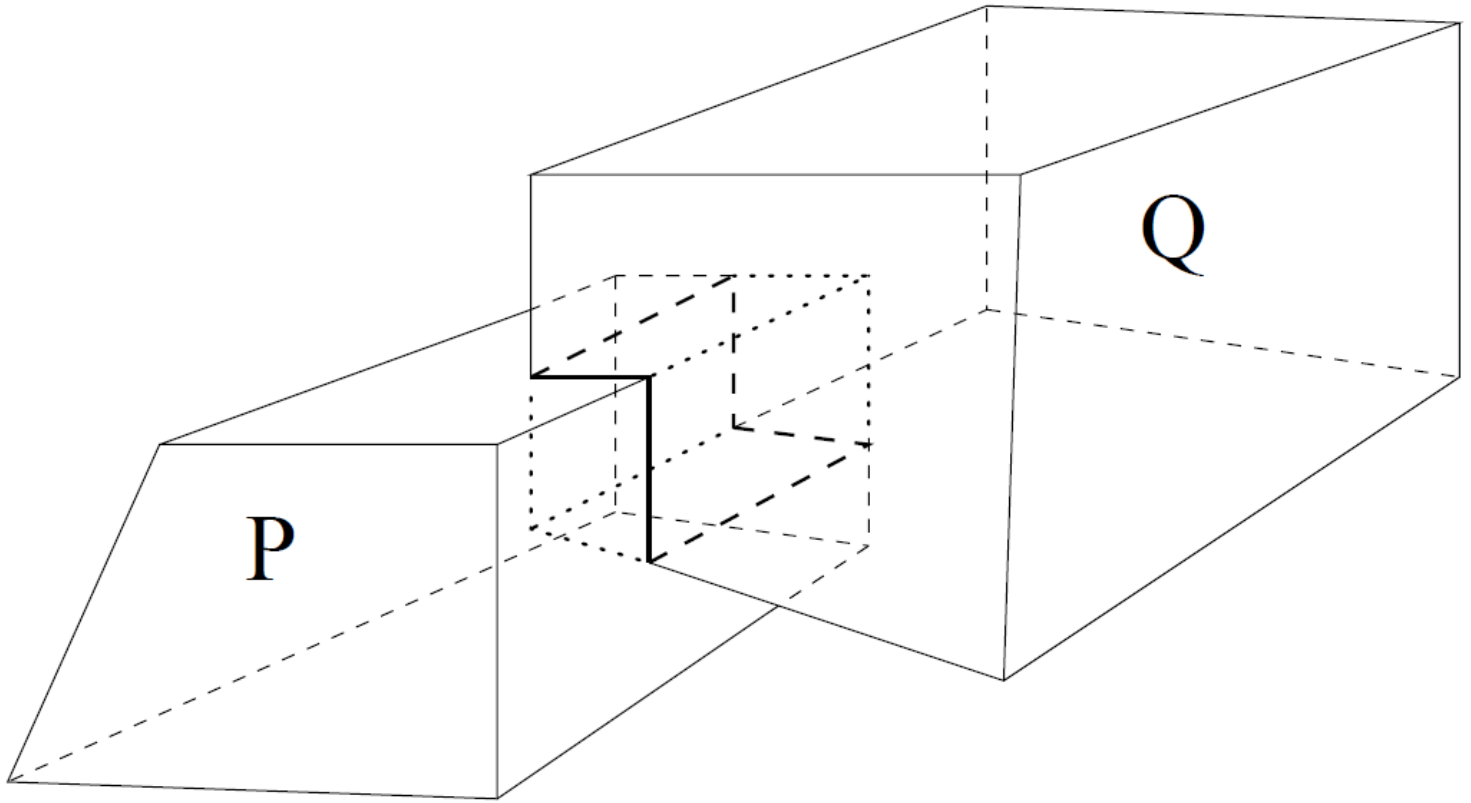
$$\sigma(1) \in X_{goal}$$

Naïve approach for constructing C-space obstacles



- Densely sample the space of all robot configurations
  - For each config q, use known environment and robot geometric models to construct the workspace shapes
  - Apply standard collision checking routines to check if in collision
    - If in collision, add to C_{obs} ; else add to C_{free}

Naïve approach for constructing C-space obstacles



Checking collision between two convex polyhedra is easy

All non-convex polyhedra can be decomposed into a union of convex polyhedra

Approximate all other shapes as polyhedra / unions of spheres

Methods for configuration-space motion planning

Good to know:

- Visibility graphs
- Voronoi diagrams
- Exact cell decomposition
- Approximate cell decomposition

Unfortunately, most of the above methods
only work for a small number of obstacles
in low-dimensional configuration spaces (2-3 DOF)

Must know (next time):

- Sampling-based motion planning
  - Probabilistic roadmap (PRM)
  - Rapidly-exploring random tree (RRT)

Outline

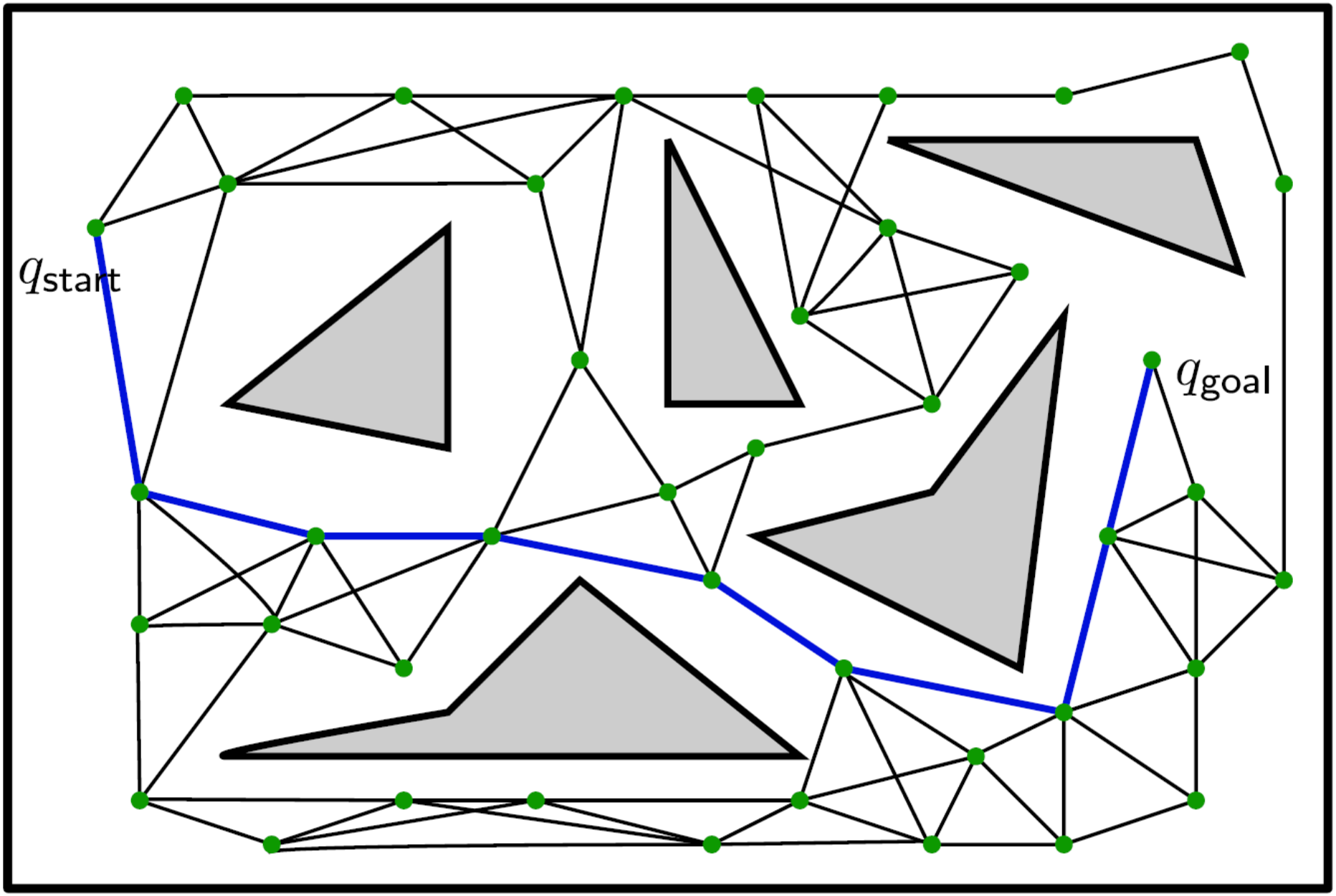
Probabilistic roadmap (PRM)

Rapidly exploring random tree (RRT)

Probabilistic roadmap (PRM)

- **Idea:** Take random samples from C, declare them as vertices if in C_{free}, try to connect nearby vertices with local planner
- The **local planner** checks if line-of-sight is collision-free (powerful or simple methods)
- Options for *nearby:* **k-nearest neighbors** or all neighbors within **specified radius**
- Configurations and connections are added to graph until roadmap is **dense enough**

Probabilistic Road Maps



Given the graph, use (e.g.) Dijkstra to find path from q_{start} to q_{goal}.

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

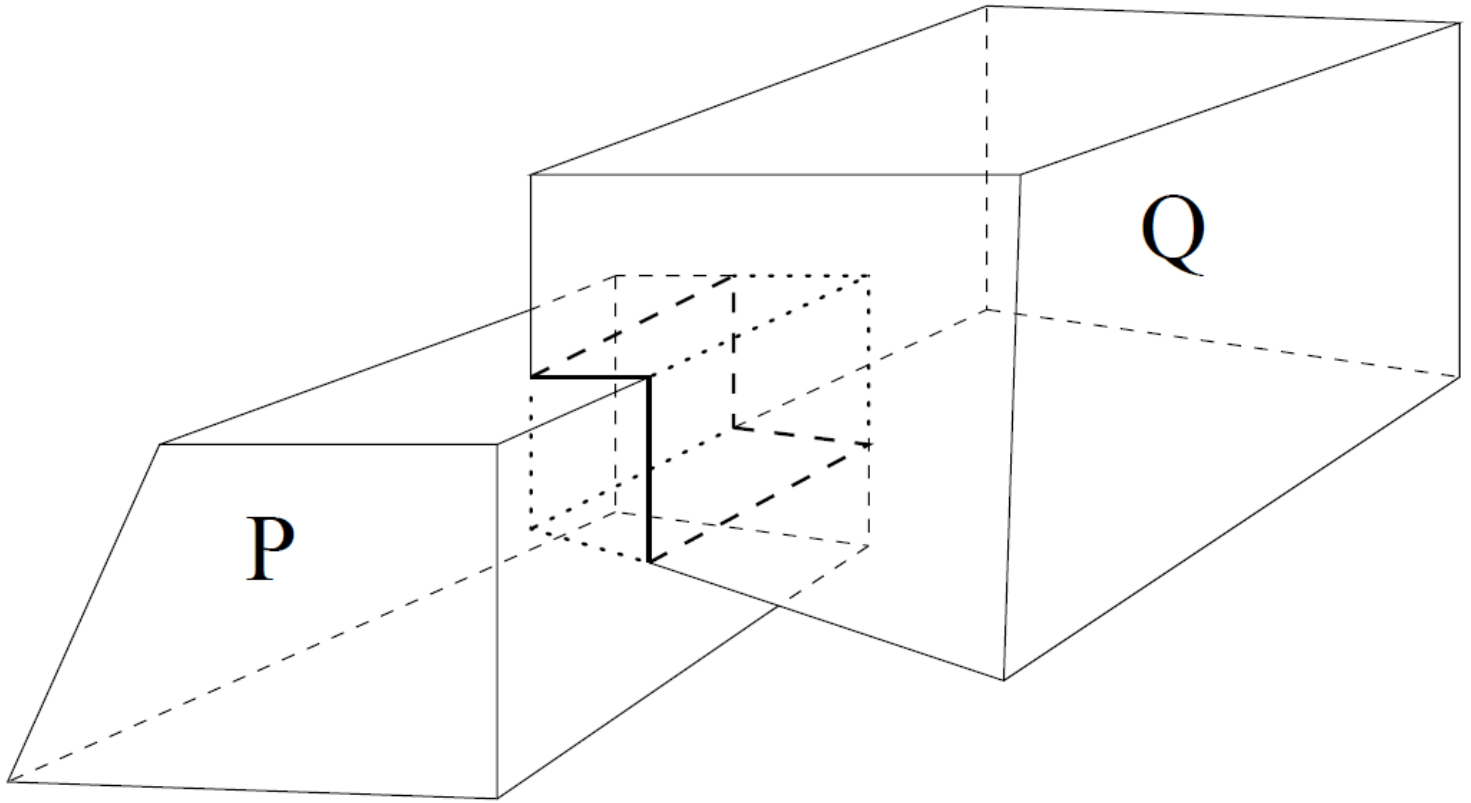
1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6:
7:
8:
9:
10:
11:

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6:
7:
8:
9:
10:
11:

Collision checking: Static configuration



Checking collision between two convex polyhedra is easy

All non-convex polyhedra can be decomposed into a union of convex polyhedra

Approximate all other shapes as polyhedra / unions of spheres

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

```
1
2
3
4
5
6: for all q ∈ V do                          // check if near points can be connected
7:     N_q ← k nearest neighbors of q in V
8:     for all q' ∈ N_q do
9:         if path(q, q') ∈ Q_free then E ← E ∪ {(q, q')}
10:    end for
11: end for
```

where $\text{path}(q, q')$ is a local planner (easiest: straight line)

Probabilistic Road Maps – generation

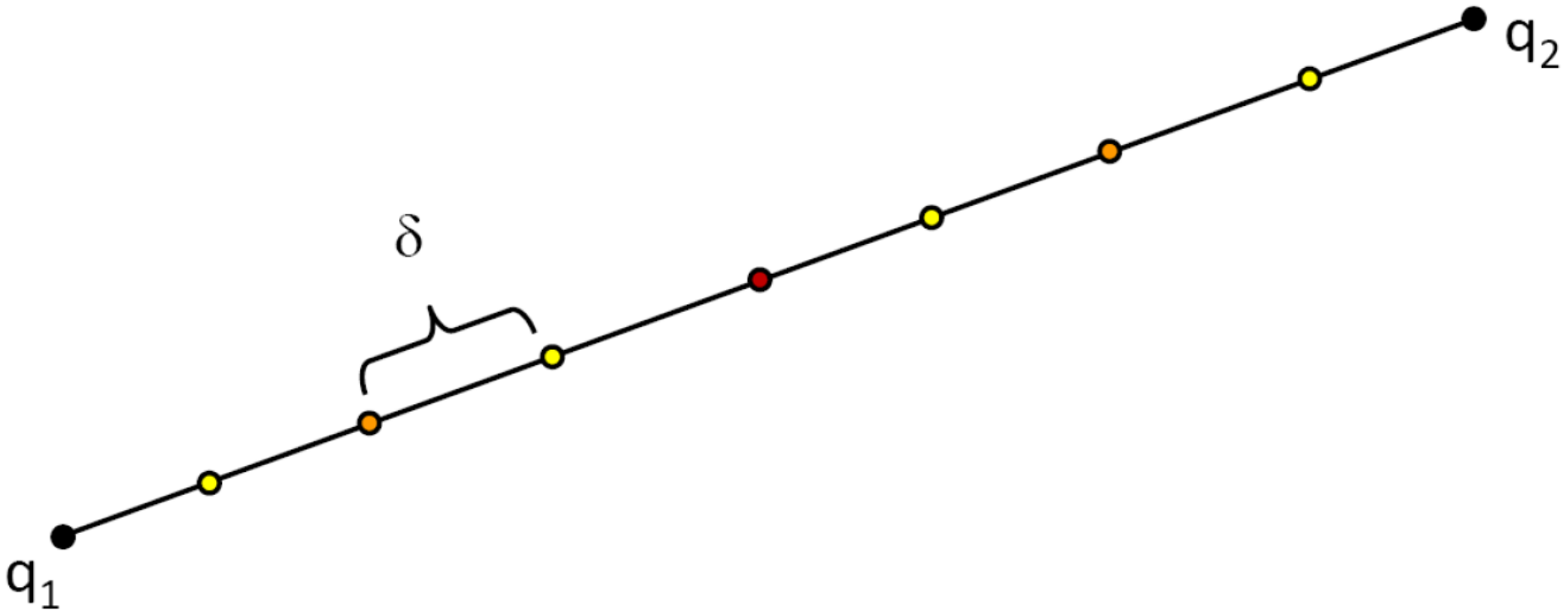
Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

```
1
2
3
4
5
6:  for all q ∈ V do                          // check if near points can be connected
7:      N_q ← k nearest neighbors of q in V
8:      for all q' ∈ N_q do
9:          if path(q, q') ∈ Q_free then E ← E ∪ {(q, q')}
10:     end for
11: end for
```

where $\text{path}(q, q')$ is a local planner (easiest: straight line)

Collision checking: Local path

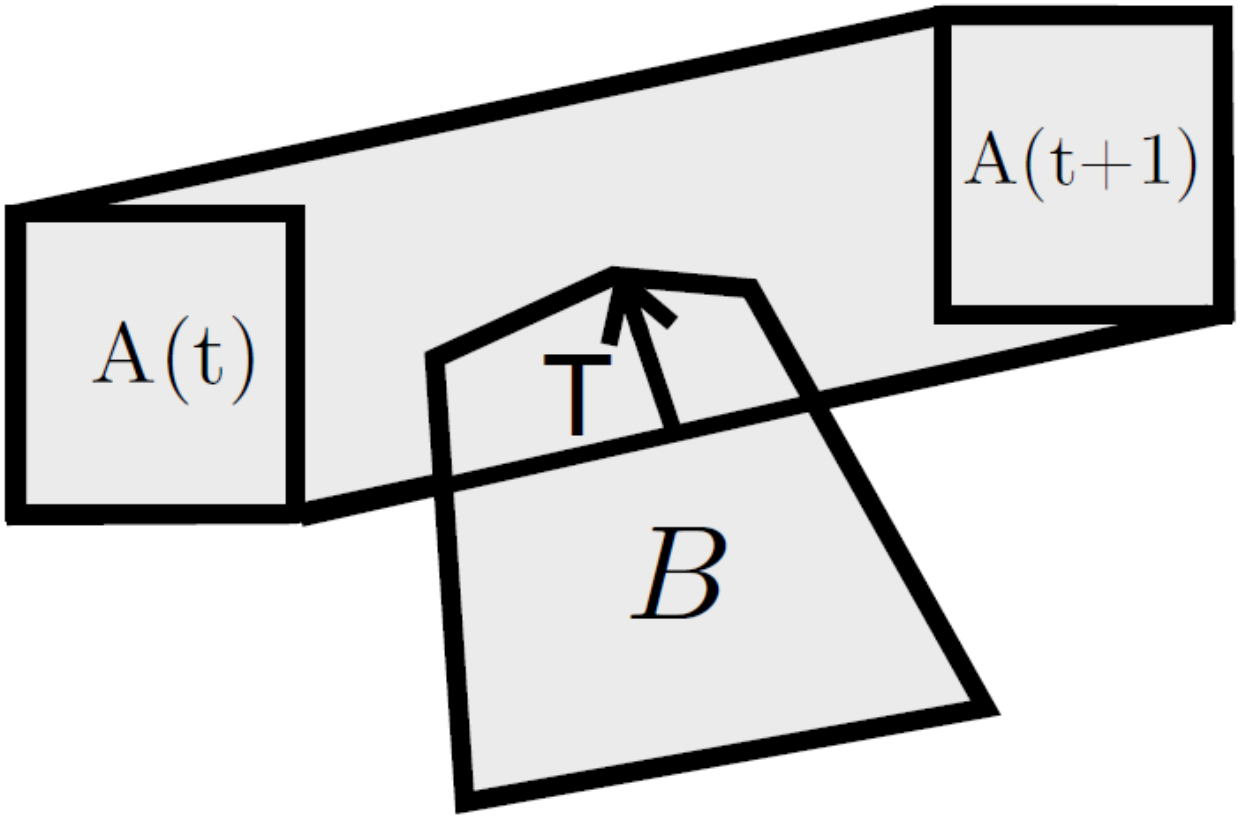
Local Planner



tests collisions up to a specified resolution δ

Check for collisions between small straight-line segments in C-space

Collision checking: Local path



Form the swept volume of the polyhedra by
taking the convex hull of two configurations

Collision checking: Local path

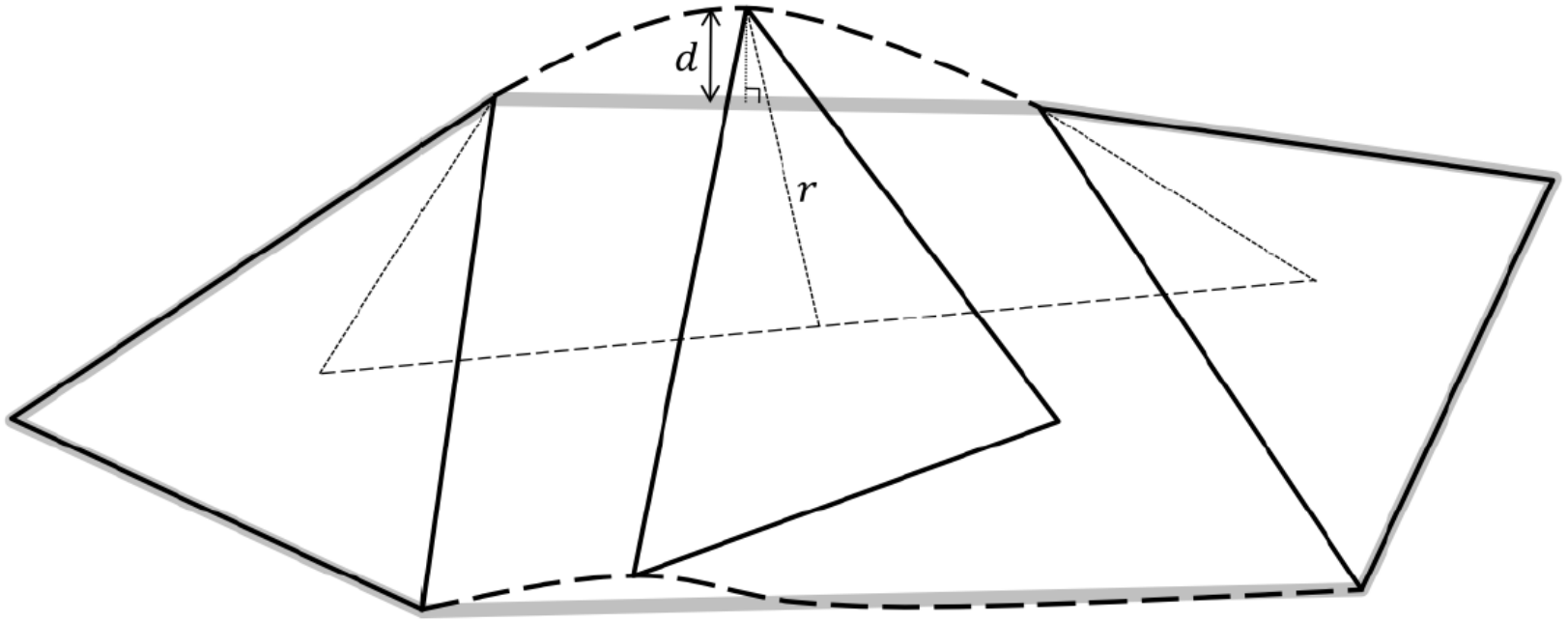


Fig. 6. Illustration of the difference between swept out shape and convex hull. The figure shows a triangle undergoing translation and uniform rotation. The swept-out area is enclosed by dotted lines, and the convex hull is shown by a thick gray line.

Form the swept volume of the polyhedra by
taking the convex hull of two configurations
Note: This could be an approximation – depends on collision check resolution

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

```
1: initialize V = ∅, E = ∅
2: while |V| < n do                                  // find n collision free points q_i
3:     q ← random sample from Q
4:     if q ∈ Q_free then V ← V ∪ {q}
5: end while
6: for all q ∈ V do                                  // check if near points can be connected
7:     N_q ← k nearest neighbors of q in V
8:     for all q' ∈ N_q do
9:         if path(q, q') ∈ Q_free then E ← E ∪ {(q, q')}
10:    end for
11: end for
```

where $\text{path}(q, q')$ is a local planner (easiest: straight line)

Alternatives:
- Connect all neighbors within some distance
- Connect to all possible points! (inefficient but exhaustive)

Other considerations in PRMs

- Reaching the goal
- Extracting the solution
- Completeness
- Optimality
- Sampling strategies

Probabilistic Road Maps – generation

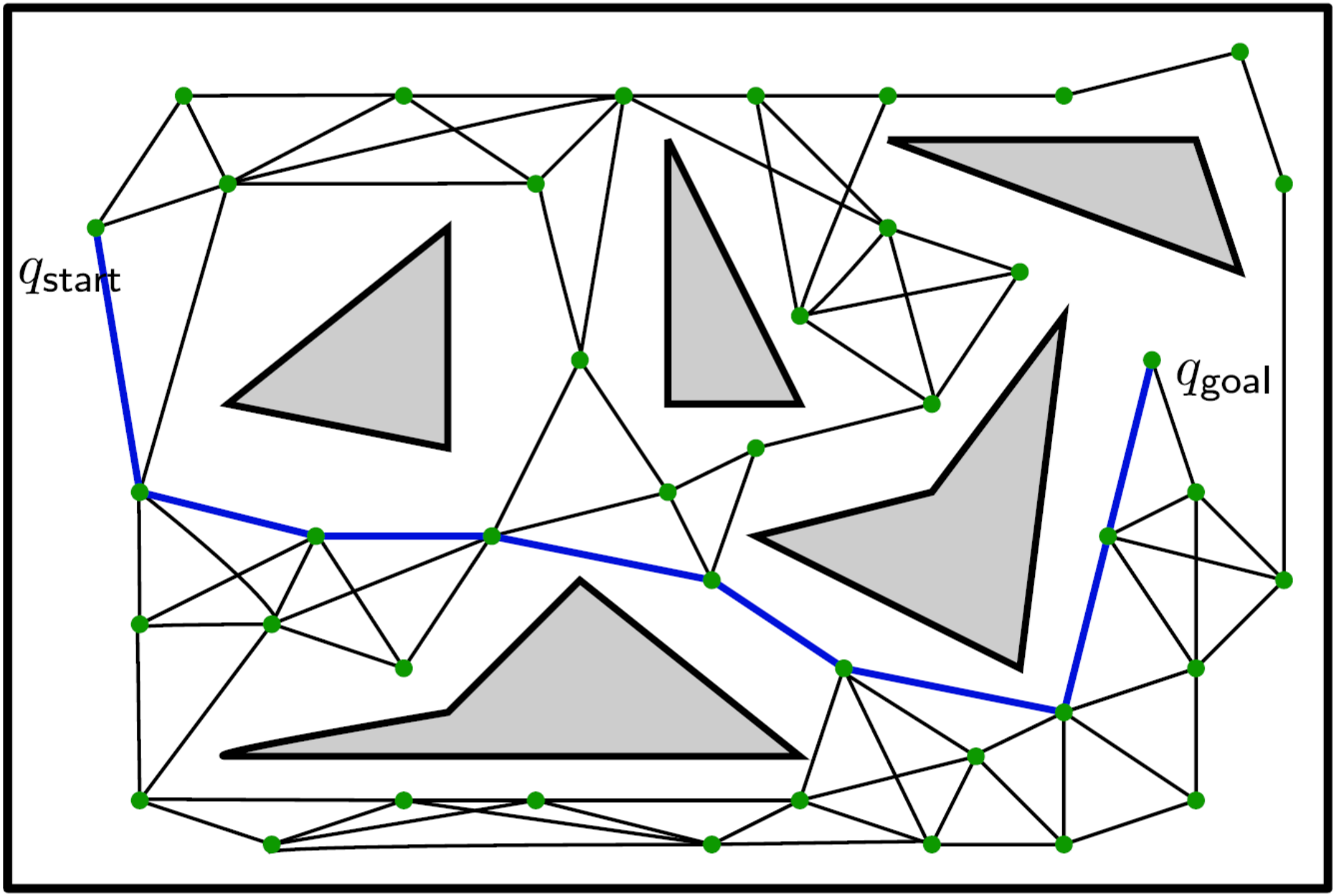
Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

```
1: initialize V = ∅, E = ∅
2: while |V| < n do                                   // find n collision free points q_i
3:     q ← random sample from Q
4:     if q ∈ Q_free then V ← V ∪ {q}
5: end while
6: for all q ∈ V do                                   // check if near points can be connected
7:     N_q ← k nearest neighbors of q in V
8:     for all q' ∈ N_q do
9:         if path(q, q') ∈ Q_free then E ← E ∪ {(q, q')}
10:    end for
11: end for
```

where $\text{path}(q, q')$ is a local planner (easiest: straight line)

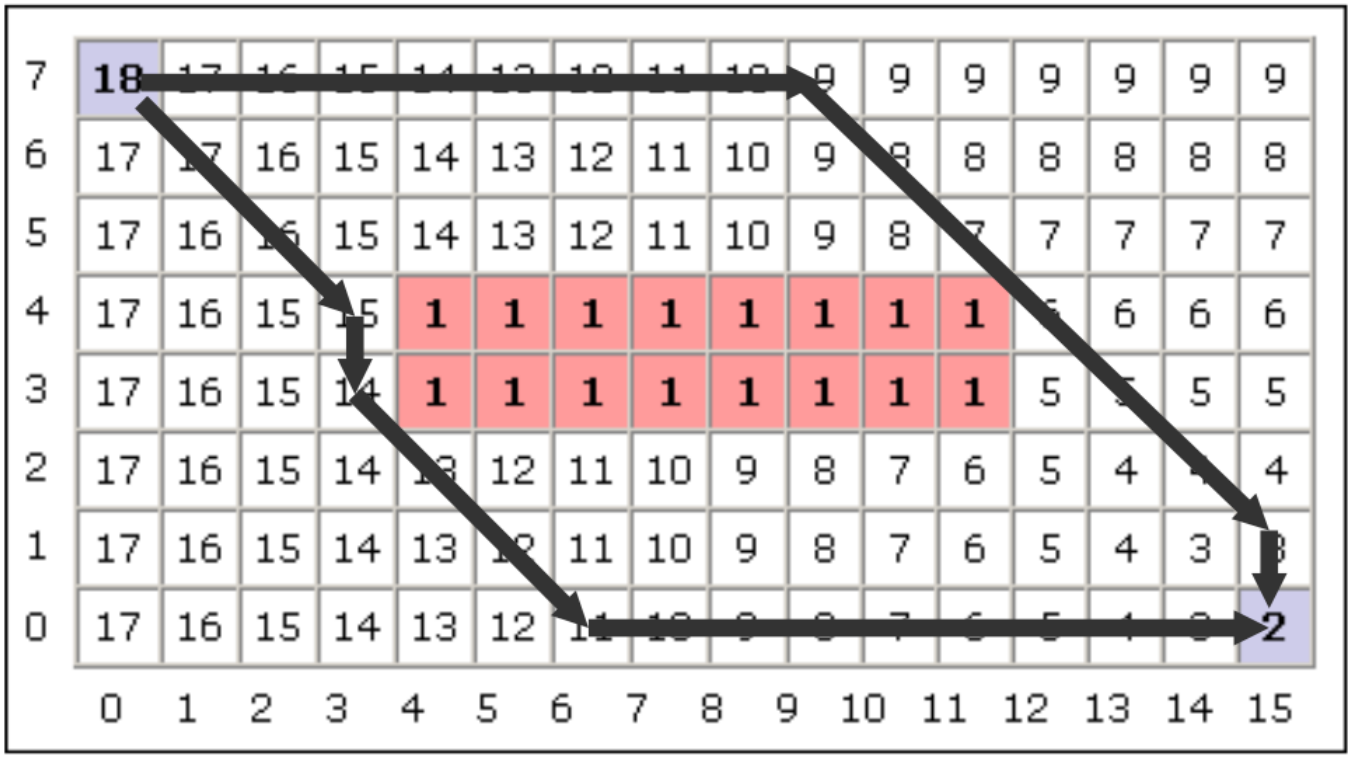
This produces a roadmap
Shortest paths between roadmap nodes can be found using Dijkstra's algorithm
To answer a motion planning query, connect init and target to closest node

Probabilistic Road Maps



Given the graph, use (e.g.) Dijkstra to find path from q_{start} to q_{goal}.

Completeness and optimality



Completeness: If solution exists, algorithm can find it

Optimality: Solution optimizes some objective (e.g., minimum distance)

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Does this algorithm work?
- Is it complete?
- Is it optimal?

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Does this algorithm work?
Is it complete? – depends on n
Is it optimal?

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Does this algorithm work?
Is it complete? – depends on n
Is it optimal? – depends on the graph, and the search algorithm

PRM theory

(for uniform sampling in d-dim space)

- Let $a, b \in Q_{\text{free}}$ and γ a path in Q_{free} connecting a and b.

  Then the probability that PRM found the path after n samples is

$$P(\text{PRM-success} \,|\, n) \geq 1 - \frac{2|\gamma|}{\varrho}\, e^{-\sigma \varrho^d n}$$

$\sigma = \frac{|B_1|}{2^d |Q_{\text{free}}|}$
$\varrho =$ clearance of γ (distance to obstacles)
(roughly: the exponential term are "volume ratios")

- This result is called *probabilistic complete* (one can achieve any probability with high enough n)
- For a given success probability, n needs to be exponential in d

PRM: Probabilistic completeness

Probability of not finding a solution
decreases exponentially with the number of vertices

PRM: Probabilistic completeness

Probability of not finding a solution
decreases exponentially with the number of vertices

Theorem 15 (Probabilistic completeness of sPRM (Kavraki et al. 1998)). *Consider a robustly feasible path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. *There exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$, *dependent only on* $\mathcal{X}_{\text{free}}$ *and* $\mathcal{X}_{\text{goal}}$, *such that*

$$\mathbb{P}(\{\exists\, x_{\text{goal}} \in V_n^{\text{sPRM}} \cap \mathcal{X}_{\text{goal}} : x_{\text{goal}} \text{ is connected to } x_{\text{init}} \text{ in } G_n^{\text{sPRM}}\}) > 1 - e^{-a\,n}, \quad \forall n > n_0.$$

PRM: Probabilistic completeness

Probability of not finding a solution
decreases exponentially with the number of vertices

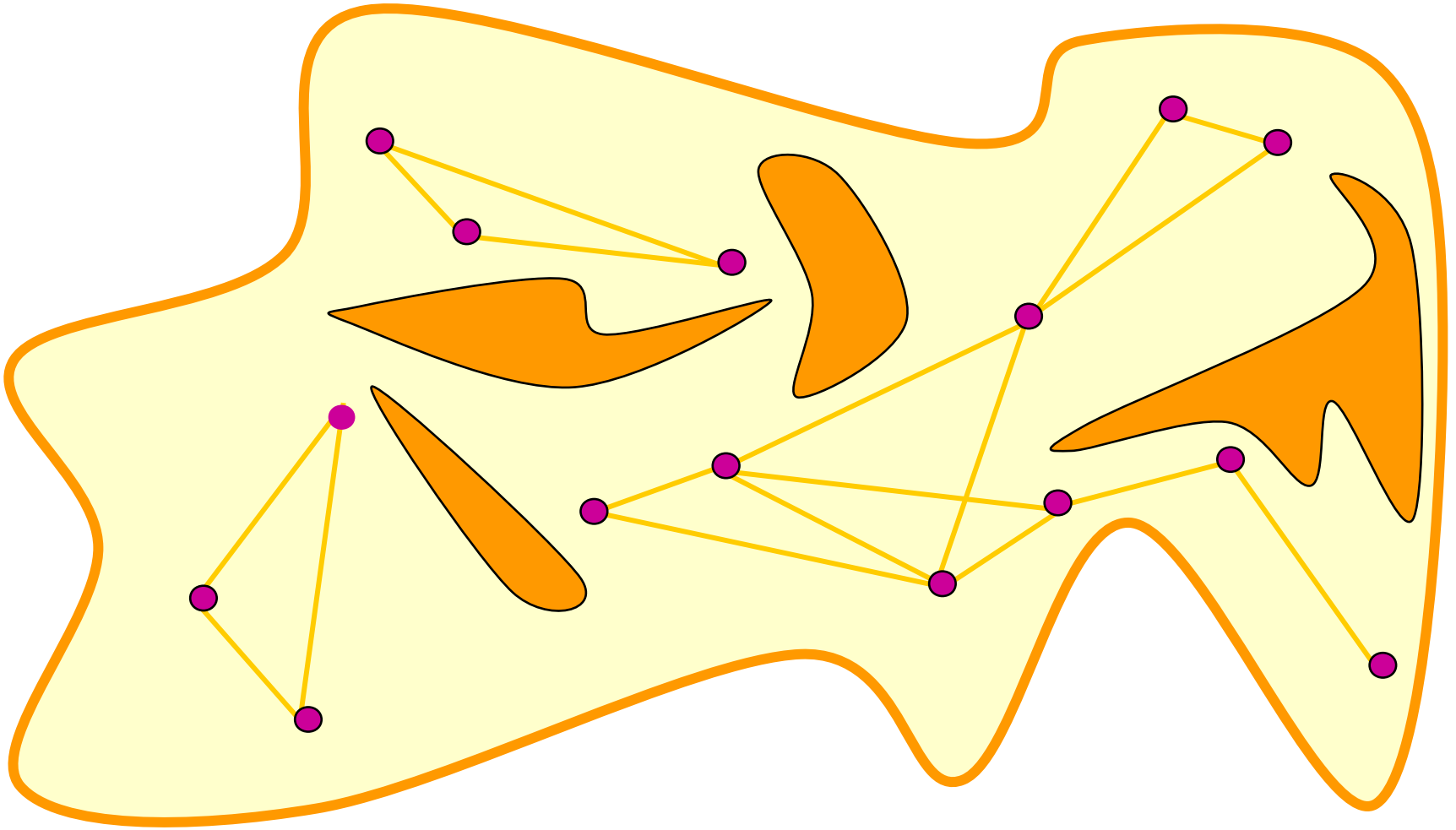
Theorem 15 (Probabilistic completeness of sPRM (Kavraki et al. 1998)). *Consider a robustly feasible path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. *There exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$, *dependent only on* $\mathcal{X}_{\text{free}}$ *and* $\mathcal{X}_{\text{goal}}$, *such that*

$$\mathbb{P}(\{\exists\, x_{\text{goal}} \in V_n^{\text{sPRM}} \cap \mathcal{X}_{\text{goal}} : x_{\text{goal}} \textit{ is connected to } x_{\text{init}} \textit{ in } G_n^{\text{sPRM}}\}) > 1 - e^{-a\,n}, \quad \forall n > n_0.$$

cf. “infinite monkey theorem”:
“A monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will type any given text, e.g., complete works of William Shakespeare, with probability one.”

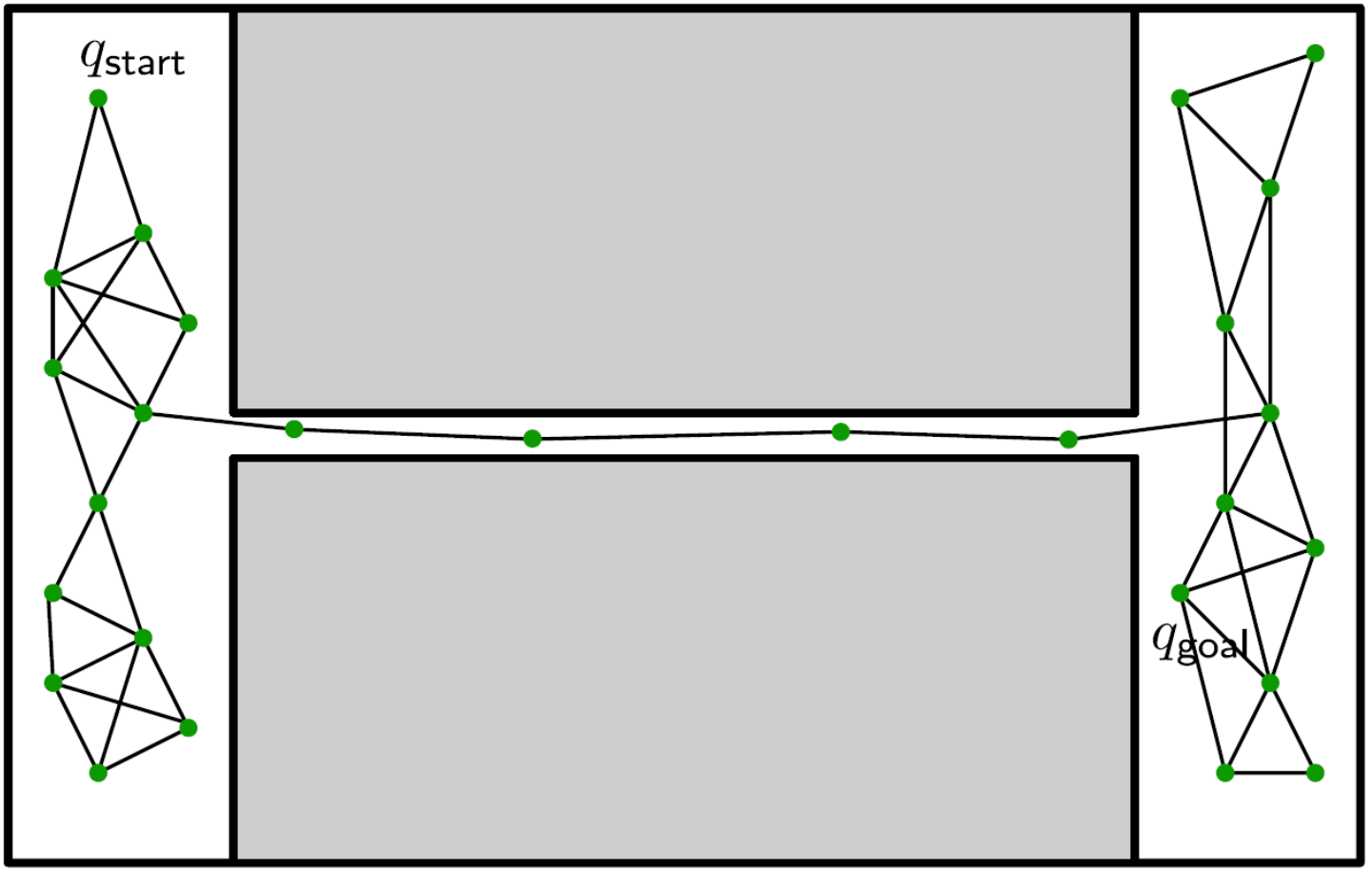
Sampling strategies

Problem: It may take a lot of samples to reach a fully connected graph



Sampling strategies

Problem: Narrow Passages



The smaller the gap (clearance ϱ) the more unlikely to sample such points.

Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

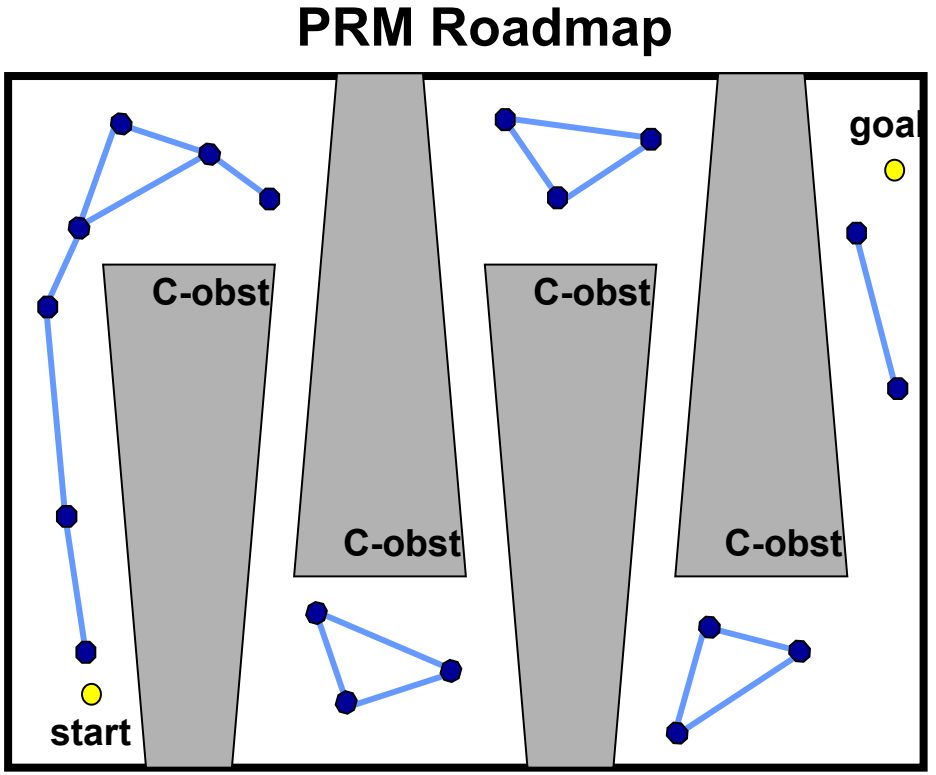
1: initialize $V = \emptyset$, $E = \emptyset$
2: **while** $|V| < n$ **do** *// find n collision free points q_i*
3: $q \leftarrow$ random sample from Q
4: **if** $q \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q\}$
5: **end while**
6: **for all** $q \in V$ **do** *// check if near points can be connected*
7: $N_q \leftarrow k$ nearest neighbors of q in V
8: **for all** $q' \in N_q$ **do**
9: **if** path$(q, q') \in Q_{\text{free}}$ **then** $E \leftarrow E \cup \{(q, q')\}$
10: **end for**
11: **end for**

where path(q, q') is a local planner (easiest: straight line)

Gaussian sampler

So far, we have only discussed uniform sampling...

Problem: Uniform sampling is not a great way
to find paths through narrow passageways



Gaussian sampler

Gaussian sampler:

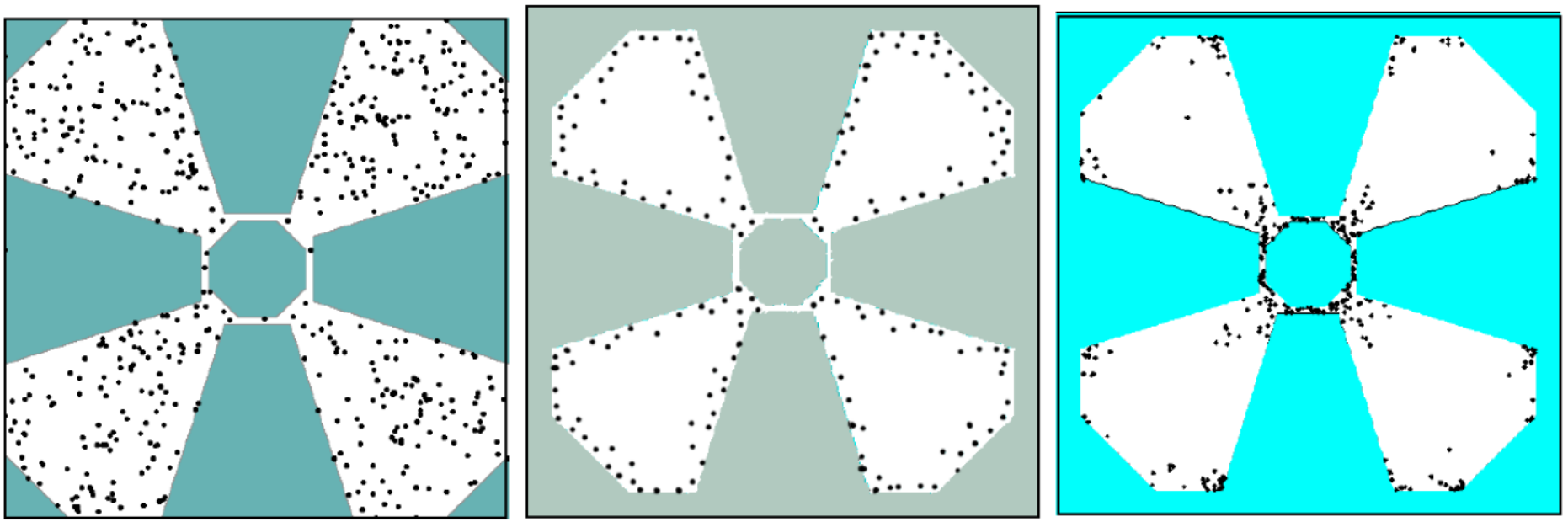
1. Sample points uniformly at random (as before)

2. For each sampled point, sample a second point from a Gaussian distribution centered at the first sampled point

3. Discard both samples if both samples are either free or in collision

4. Keep the free sample if the two samples are NOT both free or both in collision (keep the sample if the free/collision status of the second sample is different from the first)

Other PRM sampling strategies



uniform Gaussian Bridge

illustration from O. Brock's lecture

Gaussian: $q_1 \sim \mathcal{U}$; $q_2 \sim \mathcal{N}(q_1, \sigma)$; if $q_1 \in Q_{\text{free}}$ and $q_2 \notin Q_{\text{free}}$, add q_1 (or vice versa).

Bridge: $q_1 \sim \mathcal{U}$; $q_2 \sim \mathcal{N}(q_1, \sigma)$; $q_3 = (q_1 + q_2)/2$; if $q_1, q_2 \notin Q_{\text{free}}$ and $q_3 \in Q_{\text{free}}$, add q_3.

Outline

- ✔ Probabilistic roadmap (PRM)

Rapidly exploring random tree (RRT)

Rapidly exploring random tree (RRT)

Problems with PRM:

- Two steps: Graph construction, then graph search

- Hard to apply to problems where edges are directed
  (kinodynamic problems)

Rapidly exploring random tree (RRT)

Problems with PRM:

- Two steps: Graph construction, then graph search

- Hard to apply to problems where edges are directed
  (kinodynamic problems)

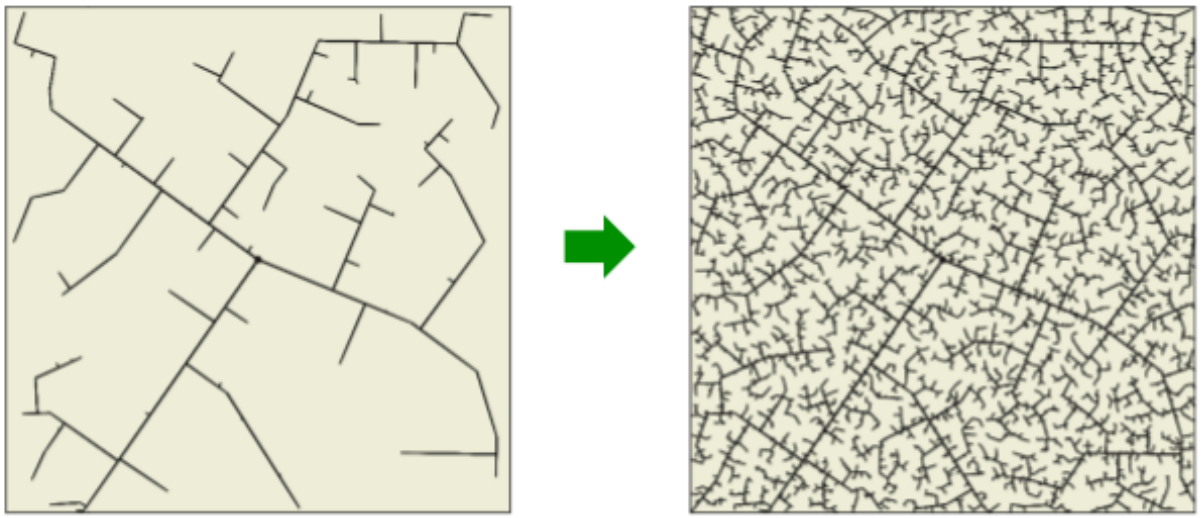
RRTs solve both problems:

- Create a tree instead of a graph:
  No graph search needed!

- Tree rooted at start or goal:
  Edges can be directed

Rapidly exploring random tree (RRT)

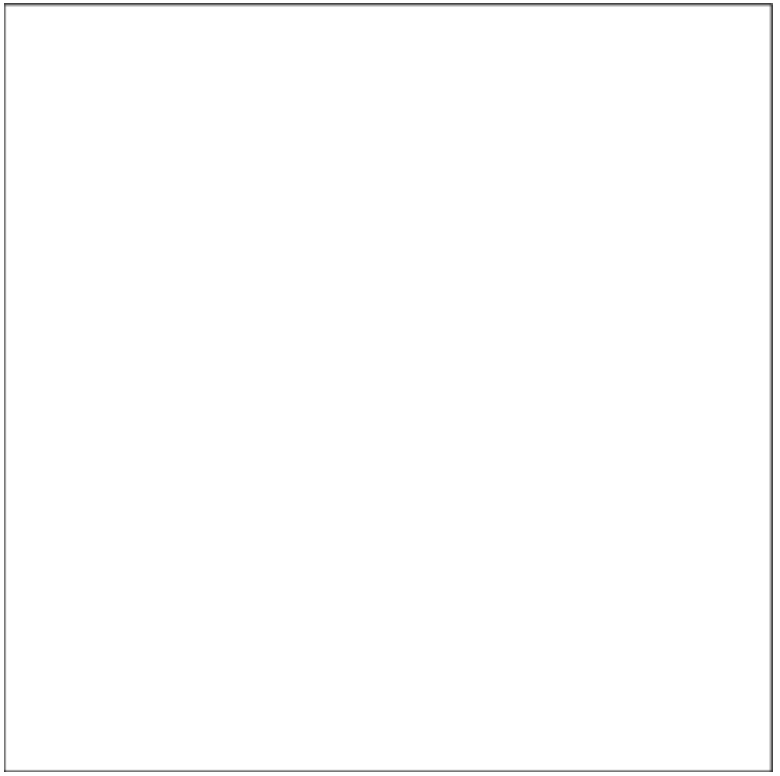
- **Idea:** aggressively probe and explore the C-space by **expanding incrementally** from an initial configuration q_0
- The explored territory is marked by a **tree rooted at q_0**

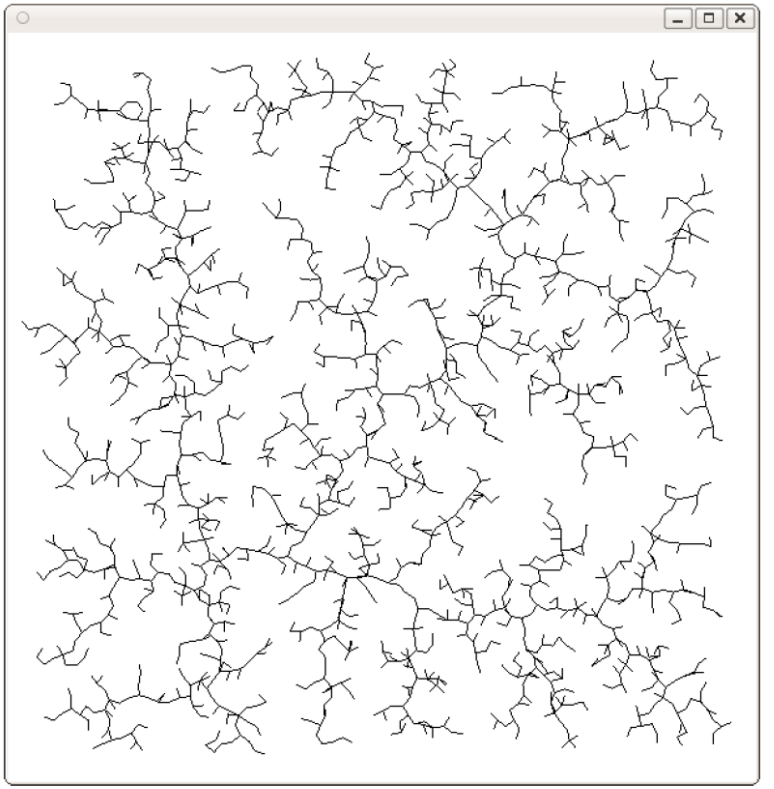


45 iterations

2345 iterations

Rapidly exploring random tree (RRT)





$n = 2000$

RRT algorithm (Wikipedia)

Algorithm [edit]

For a general configuration space C, the algorithm in pseudocode is as follows:

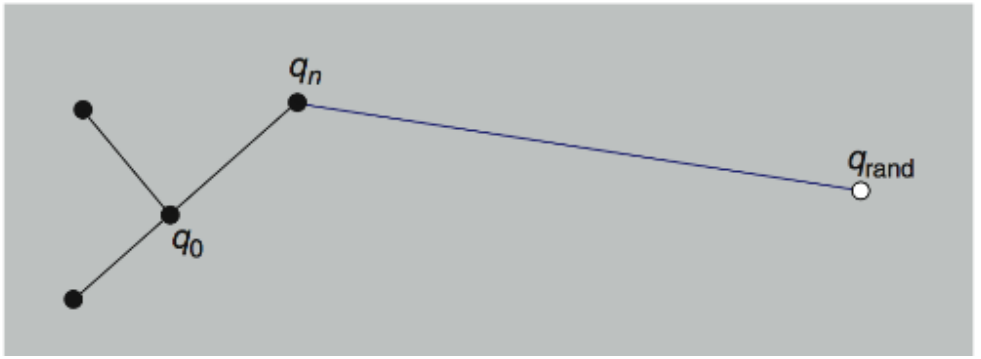
```
Algorithm BuildRRT
    Input: Initial configuration q_init, number of vertices in RRT K, incremental distance Δq)
    Output: RRT graph G

    G.init(q_init)
    for k = 1 to K do
        q_rand ← RAND_CONF()
        q_near ← NEAREST_VERTEX(q_rand, G)
        q_new ← NEW_CONF(q_near, q_rand, Δq)
        G.add_vertex(q_new)
        G.add_edge(q_near, q_new)
    return G
```

RRT algorithm (LaValle)

- The algorithm

Algorithm 1: RRT

```
1 G.init(q_0)
2 repeat
3     q_rand → RANDOM_CONFIG(C)
4     q_near ← NEAREST(G, q_rand)
5     G.add_edge(q_near, q_rand)
6 until condition
```



Connect nearest point with random point using a **local planner** that travels from q_{near} to q_{rand}

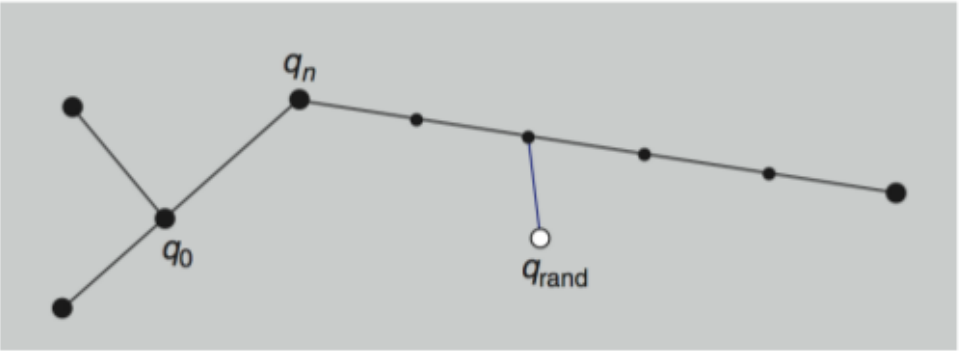
- No collision: add edge
- Collision: new vertex is q_i, as close as possible to C_{obs}

RRT algorithm (LaValle)

- The algorithm

Algorithm 1: RRT

```
1 G.init(q_0)
2 repeat
3 |   q_rand → RANDOM_CONFIG(C)
4 |   q_near ← NEAREST(G, q_rand)
5 |   G.add_edge(q_near, q_rand)
6 until condition
```



Several stategies to find q_{near} given the closest vertex on G:

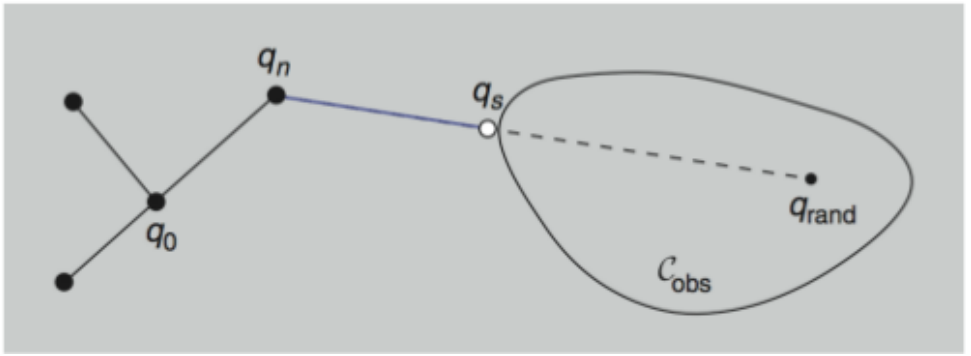
- Take closest vertex
- Check intermediate points at regular intervals and split edge at q_{near}

RRT algorithm (LaValle)

- The algorithm

Algorithm 1: RRT

```
1 G.init(q_0)
2 repeat
3 |   q_rand → RANDOM_CONFIG(C)
4 |   q_near ← NEAREST(G, q_rand)
5 |   G.add_edge(q_near, q_rand)
6 until condition
```



Connect nearest point with random point using a **local planner** that travels from q_{near} to q_{rand}

- No collision: add edge
- Collision: new vertex is q_i, as close as possible to C_{obs}

Rapidly Exploring Random Trees

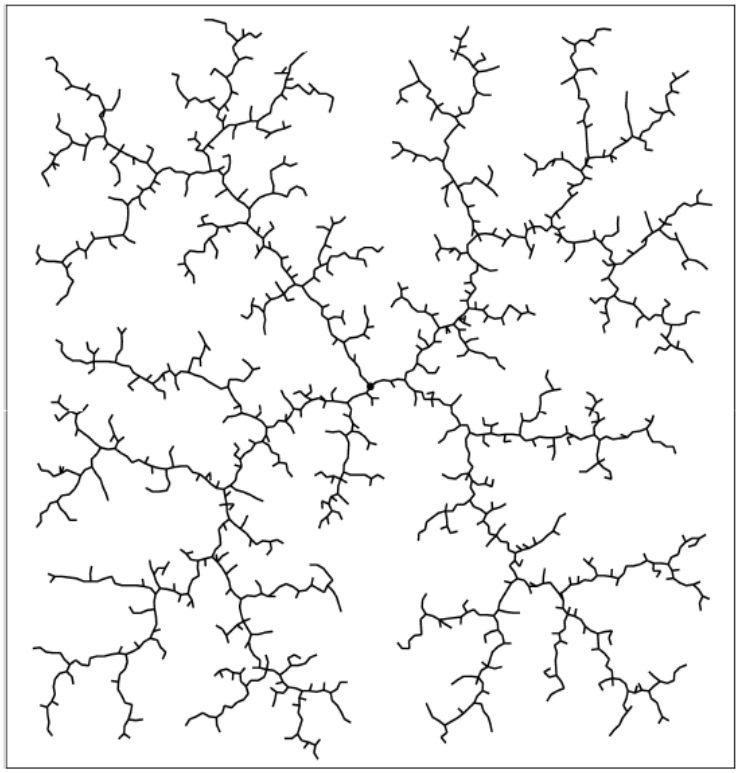
Simplest RRT with straight line local planner and step size α

Input: q_{start}, number n of nodes, stepsize α
Output: tree $T = (V, E)$

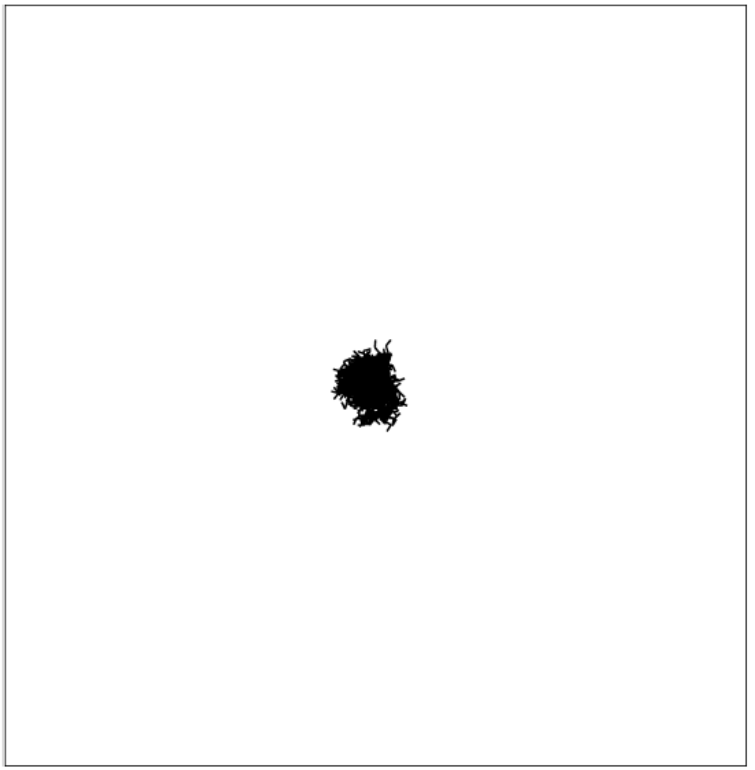
1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
2: **for** $i = 0 : n$ **do**
3: $\quad q_{\text{target}} \leftarrow$ random sample from Q
4: $\quad q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
5: $\quad q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
6: $\quad$ **if** $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
7: **end for**

RRT vs. naïve random tree

RRT



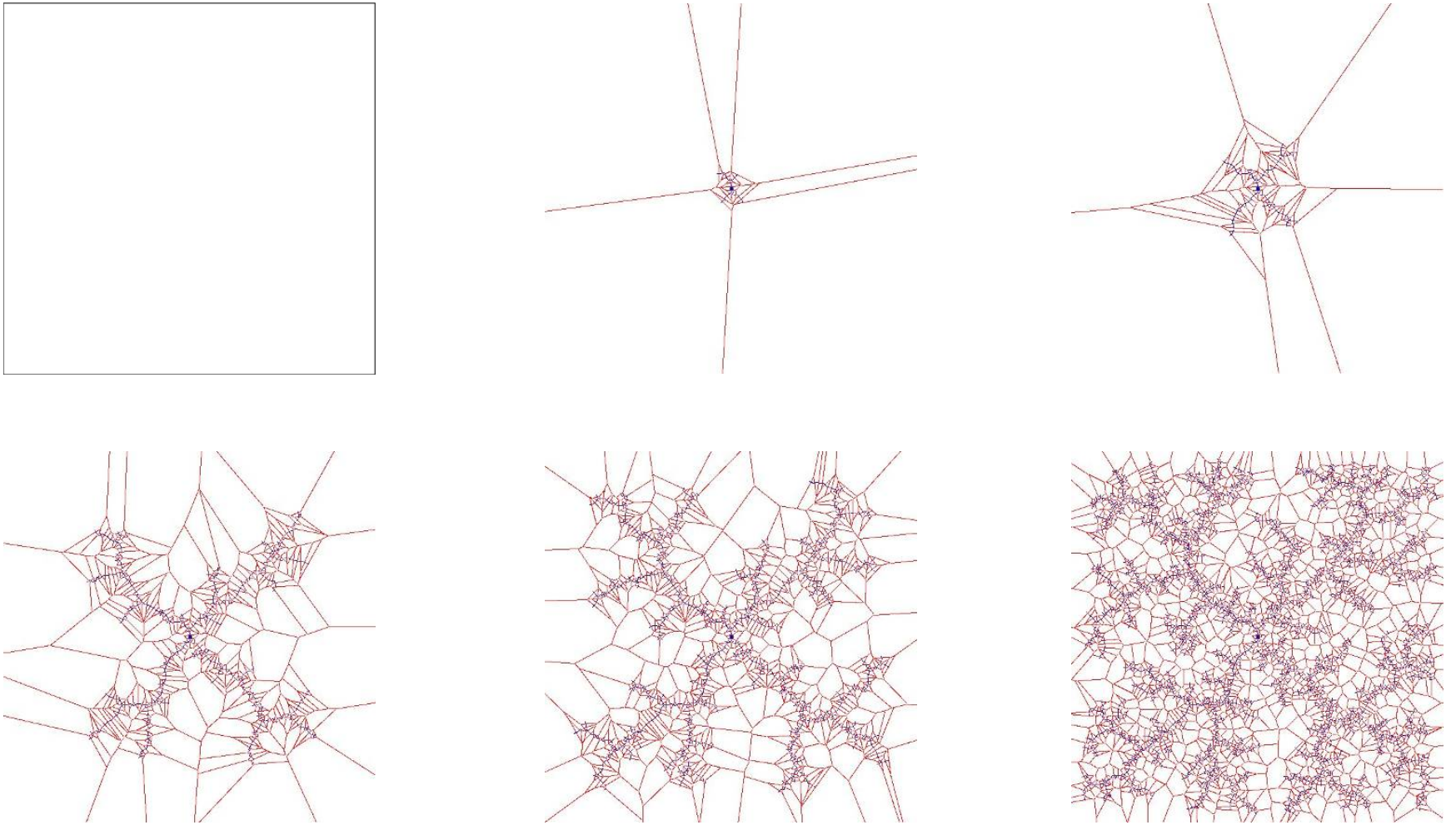
Naïve random tree



Growing the naïve random tree:

1. pick a node at random
2. sample a new node near it
3. grow tree from random node to new node

RRTs: Bias toward large Voronoi regions



http://lavalle.pl/rrt/gallery_2drrt.html

Other considerations in RRTs

- Reaching the goal
- Extracting the solution
- Kinodynamic planning
- Completeness
- Optimality
- Smoothing the path

Reaching the goal

- Bias toward larger spaces (Voronoi bias)

- Bias toward goal
    - When generating a random sample,
      with some probability (e.g., 5-10%) pick the goal
      instead of a random node when expanding

Rapidly Exploring Random Trees

RRT growing directedly towards the goal

Input: q_{start}, q_{goal}, number n of nodes, stepsize α, β
Output: tree $T = (V, E)$

1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
2: **for** $i = 0 : n$ **do**
3: **if** rand$(0, 1) < \beta$ **then** $q_{\text{target}} \leftarrow q_{\text{goal}}$
4: **else** $q_{\text{target}} \leftarrow$ random sample from Q
5: $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
6: $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
7: **if** $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
8: **end for**

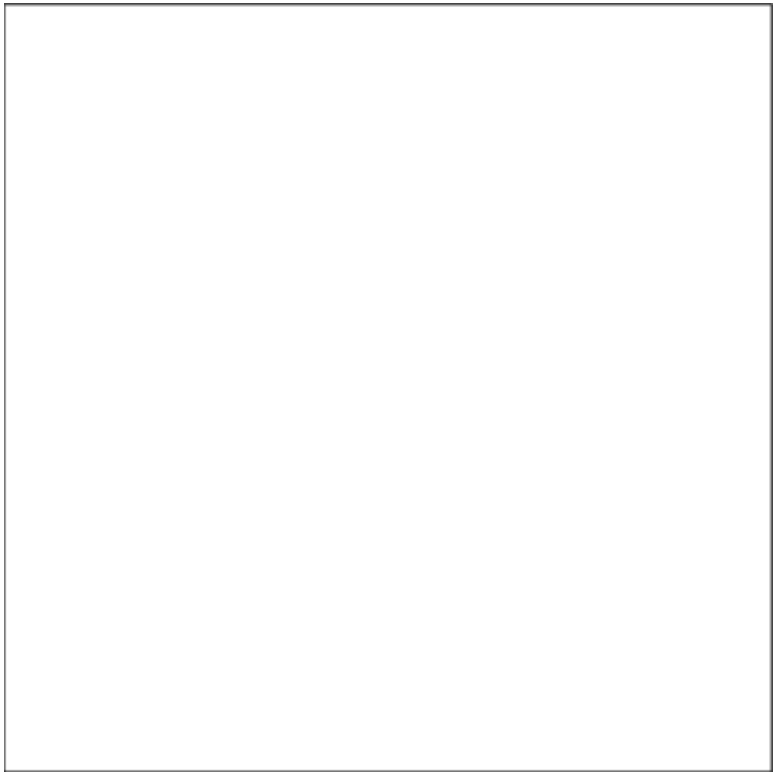
Rapidly Exploring Random Trees

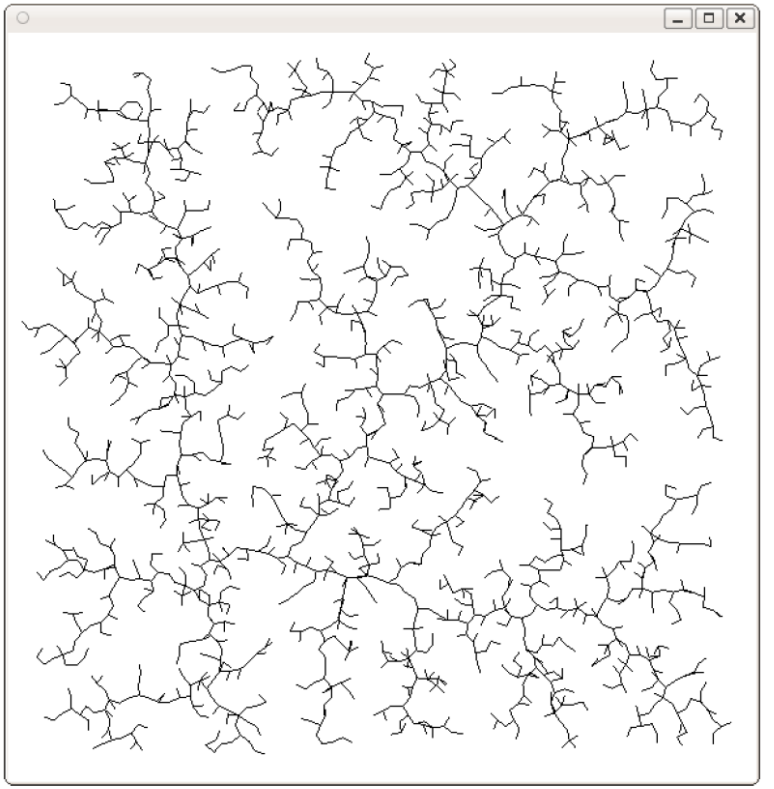
RRT growing directedly towards the goal

Input: q_{start}, q_{goal}, number n of nodes, stepsize α, β
Output: tree $T = (V, E)$

1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
2: **for** $i = 0 : n$ **do**
3: **if** rand$(0, 1) < \beta$ **then** $q_{\text{target}} \leftarrow q_{\text{goal}}$
4: **else** $q_{\text{target}} \leftarrow$ random sample from Q
5: $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
6: $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
7: **if** $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
8: **end for**

Extracting the solution – it's a tree!





$n = 2000$

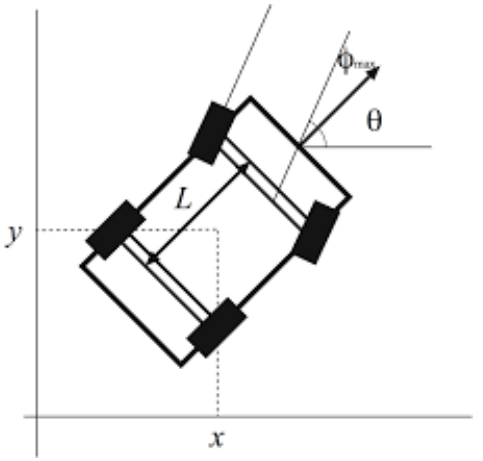
Kinodynamic planning

So far, assumed the system has no dynamics
- System can instantaneously move in any direction
  in configuration space (holonomic)

Kinodynamic planning

So far, assumed the system has no dynamics
- System can instantaneously move in any direction
  in configuration space (holonomic)

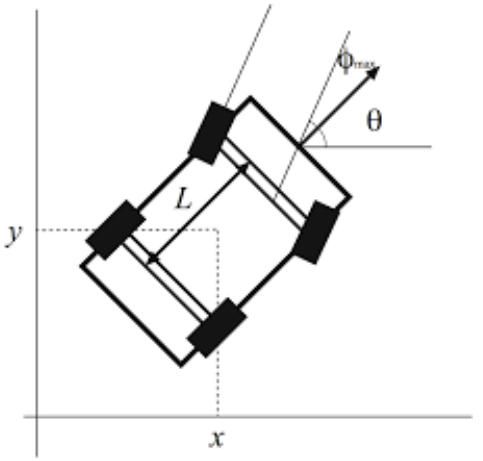
Not necessarily true, e.g., Dubins car:
- C-space: x-y position + velocity, angle
- Control forward velocity and steering angle
  (if reverse allowed: Reeds-Shepp car)

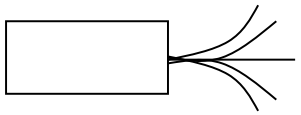


Kinodynamic planning

So far, assumed the system has no dynamics
- System can instantaneously move in any direction
  in configuration space (holonomic)

Not necessarily true, e.g., Dubins car:
- C-space: x-y position + velocity, angle
- Control forward velocity and steering angle
  (if reverse allowed: Reeds-Shepp car)

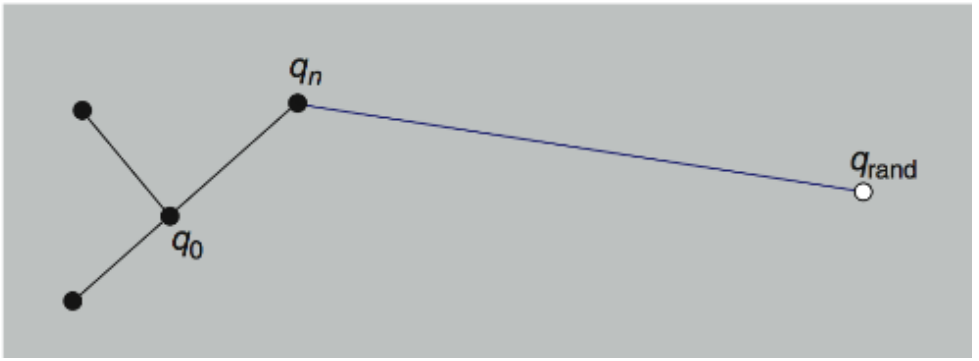




- Plan a path through c-space with
  corresponding control signals

$x_{t+1} = f(x_t, u_t)$ x – state u – control

Kinodynamic planning



Consider which control signal *u*
takes robot closest to random sample

Rapidly Exploring Random Trees

Simplest RRT with straight line local planner and step size α

Input: q_{start}, number n of nodes, stepsize α
Output: tree $T = (V, E)$

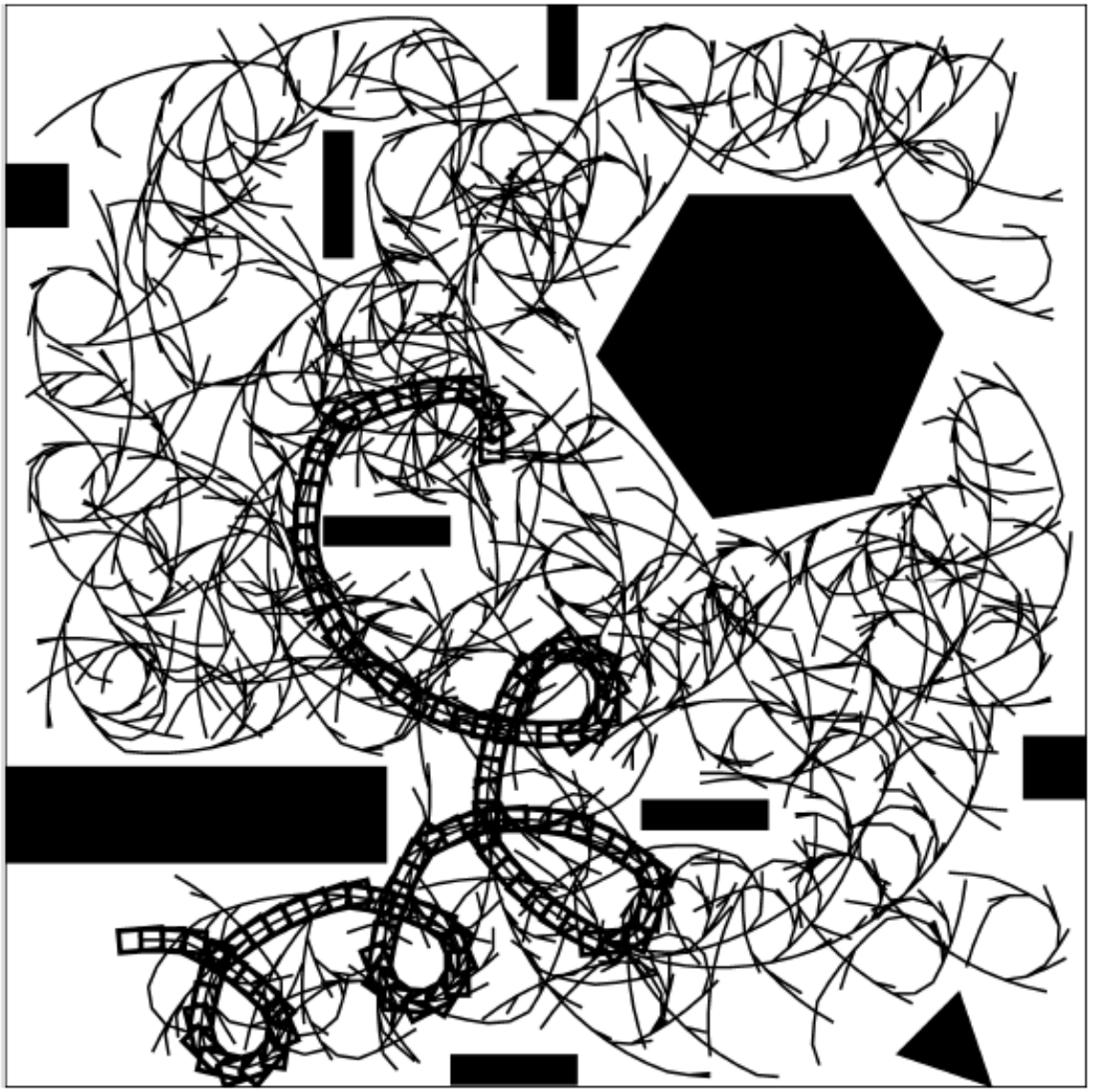
1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
2: **for** $i = 0 : n$ **do**
3: $q_{\text{target}} \leftarrow$ random sample from Q
4: $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V
5: $q_{\text{new}} \leftarrow q_{\text{near}} + \frac{\alpha}{|q_{\text{target}} - q_{\text{near}}|}(q_{\text{target}} - q_{\text{near}})$
6: **if** $q_{\text{new}} \in Q_{\text{free}}$ **then** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
7: **end for**

Replace highlighted line with a feasible trajectory:

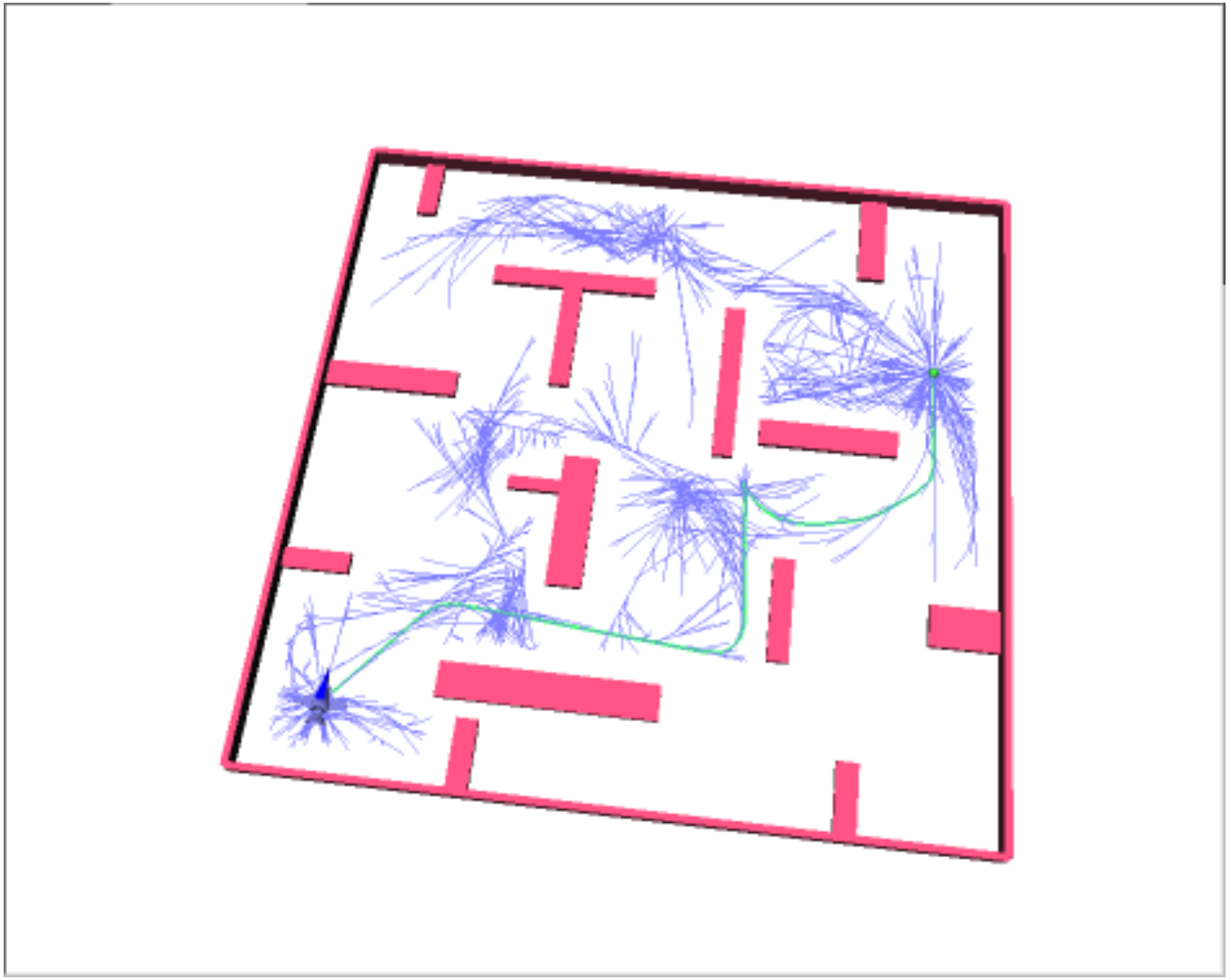
Try a small set of feasible actions/controls starting at q_{near}

Add trajectory that results in configuration closest to q_{target}

Kinodynamic planning: Left-turn only Dubins car



Kinodynamic planning: Hovercraft with 2 thrusters



http://lavalle.pl/rrt/gallery_2drot.html

Other considerations in RRTs

- Reaching the goal
- Extracting the solution
- Kinodynamic planning
- Completeness
- Optimality
- Smoothing the path

Probabilistic completeness

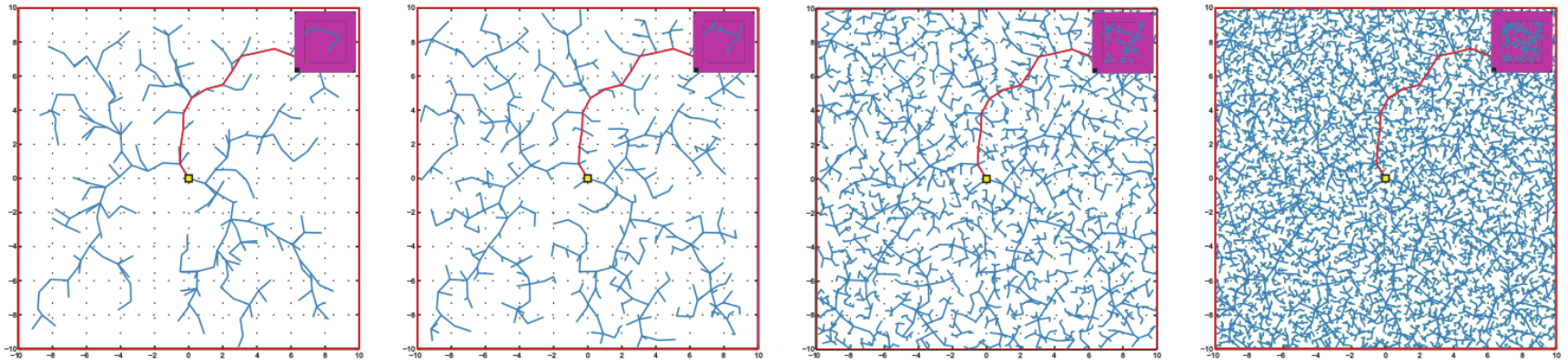
Theorem 16 (Probabilistic completeness of RRT (LaValle and Kuffner 2001)). *Consider a robustly feasible path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. *There exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$, *both dependent only on* $\mathcal{X}_{\text{free}}$ *and* $\mathcal{X}_{\text{goal}}$, *such that*

$$\mathbb{P}\left(\left\{V_n^{\text{RRT}} \cap \mathcal{X}_{\text{goal}} \neq \emptyset\right\}\right) > 1 - e^{-a\,n}, \quad \forall n > n_0.$$

Essentially the same as PRM

RRT does not find optimal paths

Theorem 33 (Non-optimality of RRT). *The RRT algorithm is not asymptotically optimal.*



Probabilistic Road Maps – generation

Input: number n of samples, number k number of nearest neighbors
Output: PRM $G = (V, E)$

```
1: initialize V = ∅, E = ∅
2: while |V| < n do                                   // find n collision free points q_i
3:     q ← random sample from Q
4:     if q ∈ Q_free then V ← V ∪ {q}
5: end while
6: for all q ∈ V do                                   // check if near points can be connected
7:     N_q ← k nearest neighbors of q in V
8:     for all q' ∈ N_q do
9:         if path(q, q') ∈ Q_free then E ← E ∪ {(q, q')}
10:    end for
11: end for
```

where $\text{path}(q, q')$ is a local planner (easiest: straight line)

Does this algorithm work?

Is it complete? – depends on n

Is it optimal? – depends on the graph, and the search algorithm

RRT with optimality

RRG and RRT*

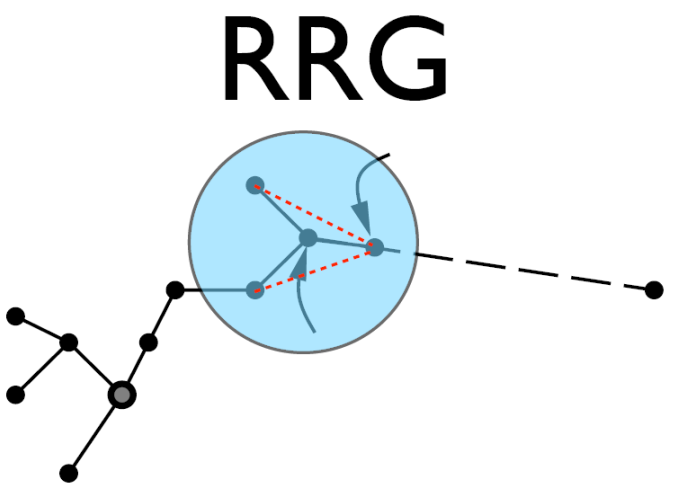
Algorithm 5: RRG.

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
2 **for** $i = 1, \ldots, n$ **do**
3 $\quad x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
4 $\quad x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
5 $\quad x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
6 $\quad$ **if** $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
7 $\quad\quad X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRG}}(\log(\text{card}(V)) / \text{card}(V))^{1/d}, \eta\})$;
8 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$;
$\quad\quad E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{nearest}})\}$;
9 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
10 $\quad\quad\quad$ **if** $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}})$ **then**
$\quad\quad\quad E \leftarrow E \cup \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$
11 **return** $G = (V, E)$;

Do not just connect x_{new} to x_{near}

Attempt to connect to every vertex within a radius *r*

RRT with optimality



- RRT algorithm extends the nearest vertex towards the sample.
- RRG also extends all vertices returned by the Near procedure (if first was success).

Do not just connect x_{new} to x_{near}

Attempt to connect to every vertex within a radius *r*

RRG

RRG is probabilistically complete
and asymptotically optimal.

Theorem 36 (Asymptotic optimality of RRG). *If* $\gamma_{\text{PRM}} > 2(1 + 1/d)^{1/d} \left(\frac{\mu(X_{\text{free}})}{\zeta_d}\right)^{1/d}$, *then the RRG algorithm is asymptotically optimal.*

RRG

RRG is probabilistically complete
and asymptotically optimal.

Theorem 36 (Asymptotic optimality of RRG). *If* $\gamma_{\text{PRM}} > 2(1 + 1/d)^{1/d} \left(\frac{\mu(X_{\text{free}})}{\zeta_d}\right)^{1/d}$, *then the RRG algorithm is asymptotically optimal.*

Why may we prefer RRT to RRG?

RRT with optimality: Take 2 (RRT*)

Algorithm 6: RRT*.

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

Do not just connect x_{new} to x_{near}

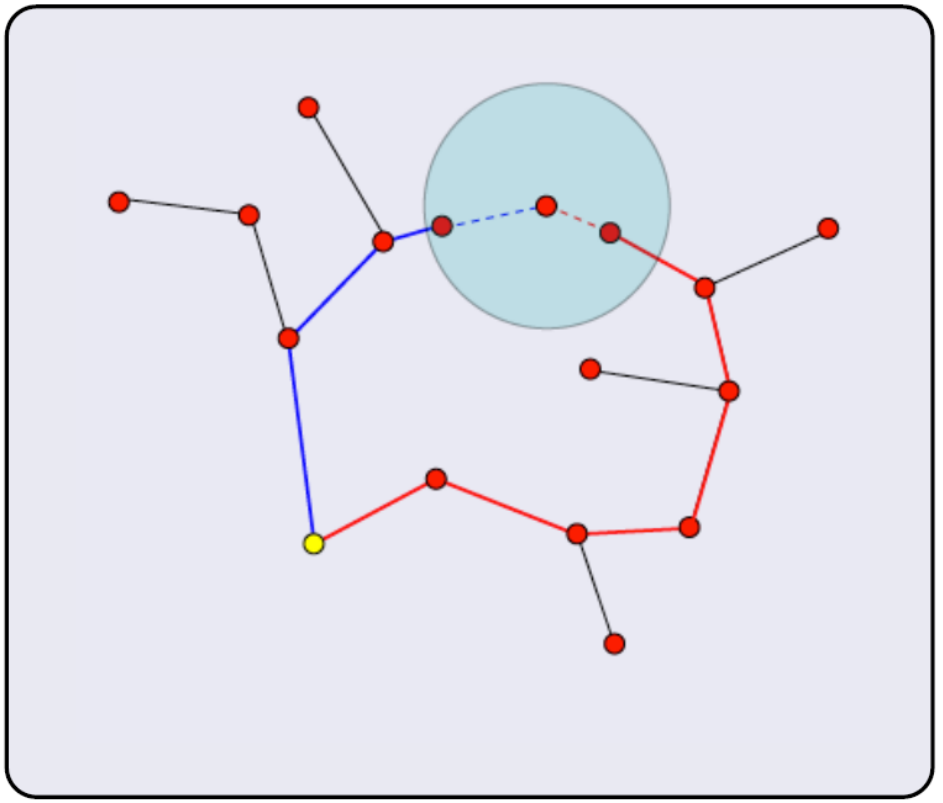
Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT with optimality: Take 2 (RRT*)

RRT*



Do not just connect x_{new} to x_{near}

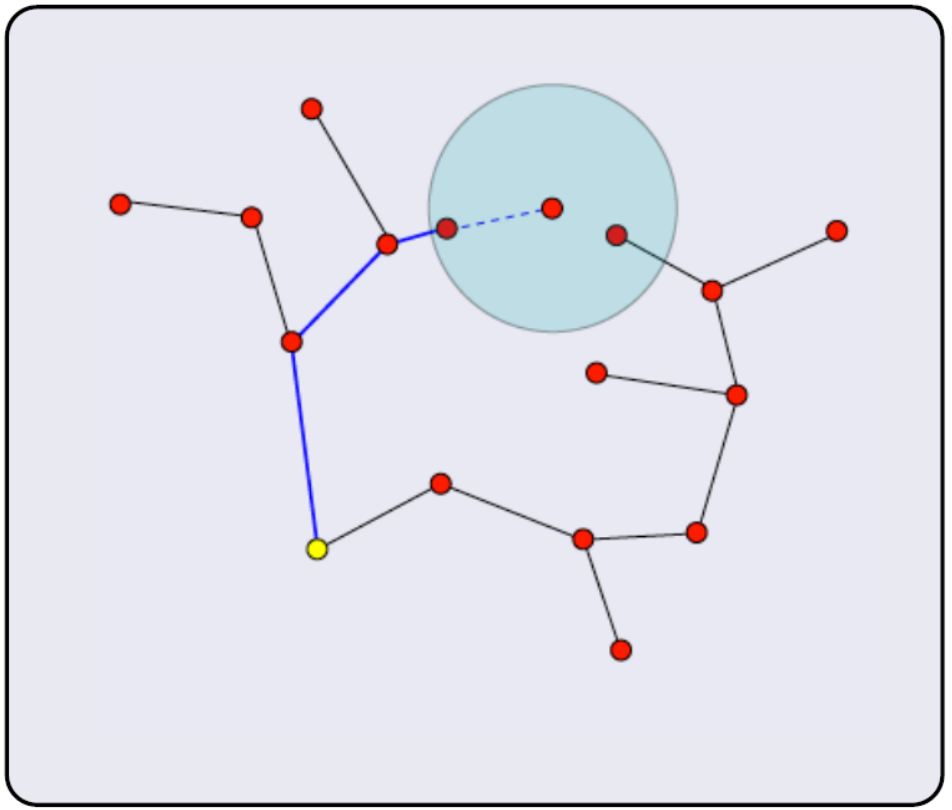
Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT with optimality: Take 2 (RRT*)

RRT*



Do not just connect x_{new} to x_{near}

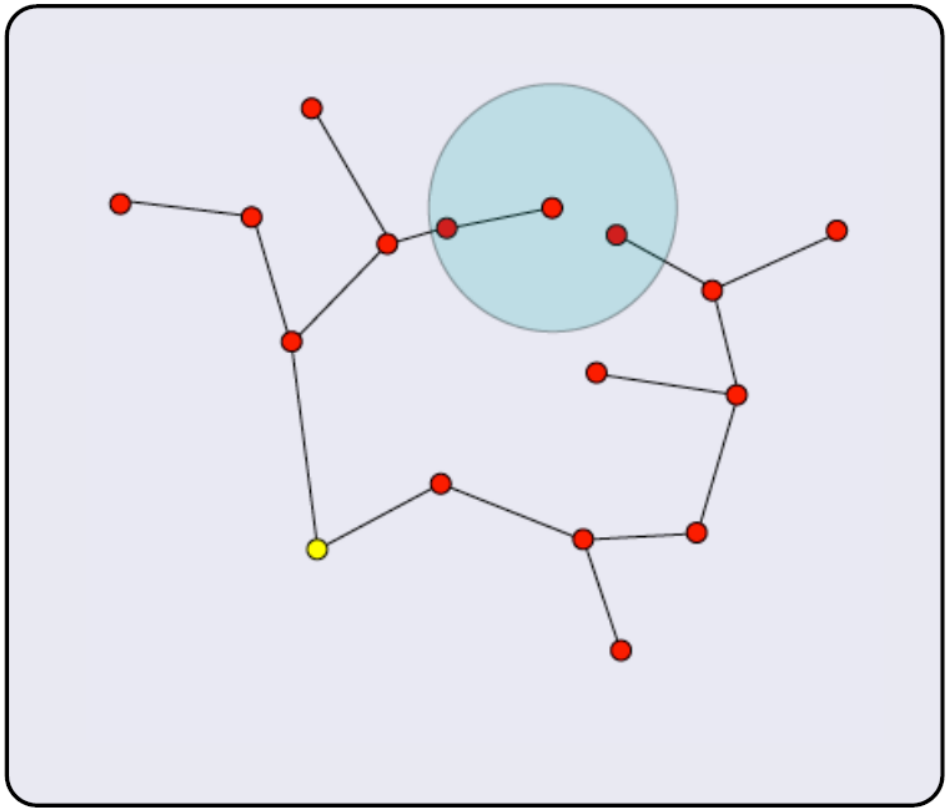
Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT with optimality: Take 2 (RRT*)

RRT*



Do not just connect x_{new} to x_{near}

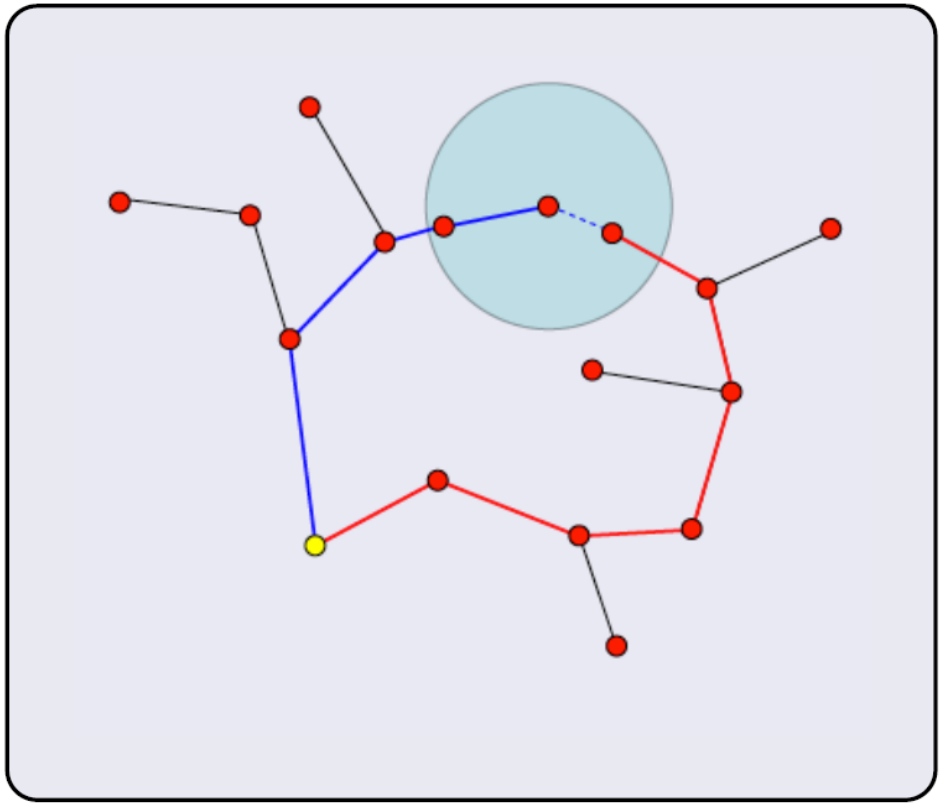
Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT with optimality: Take 2 (RRT*)

RRT*



Do not just connect x_{new} to x_{near}

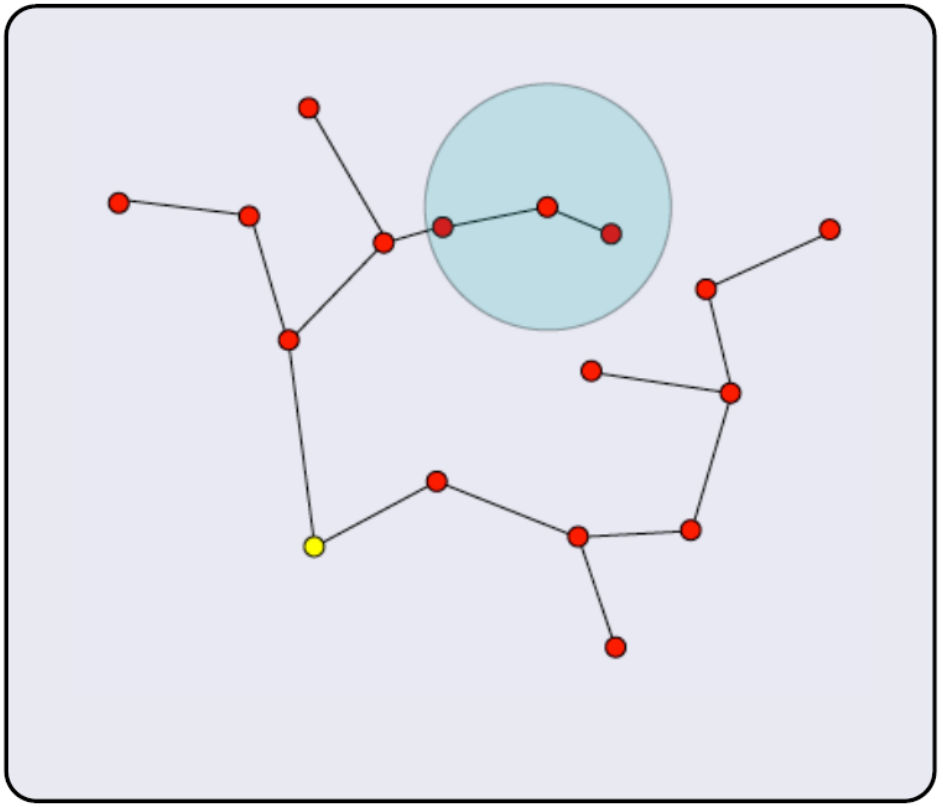
Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT with optimality: Take 2 (RRT*)

RRT*



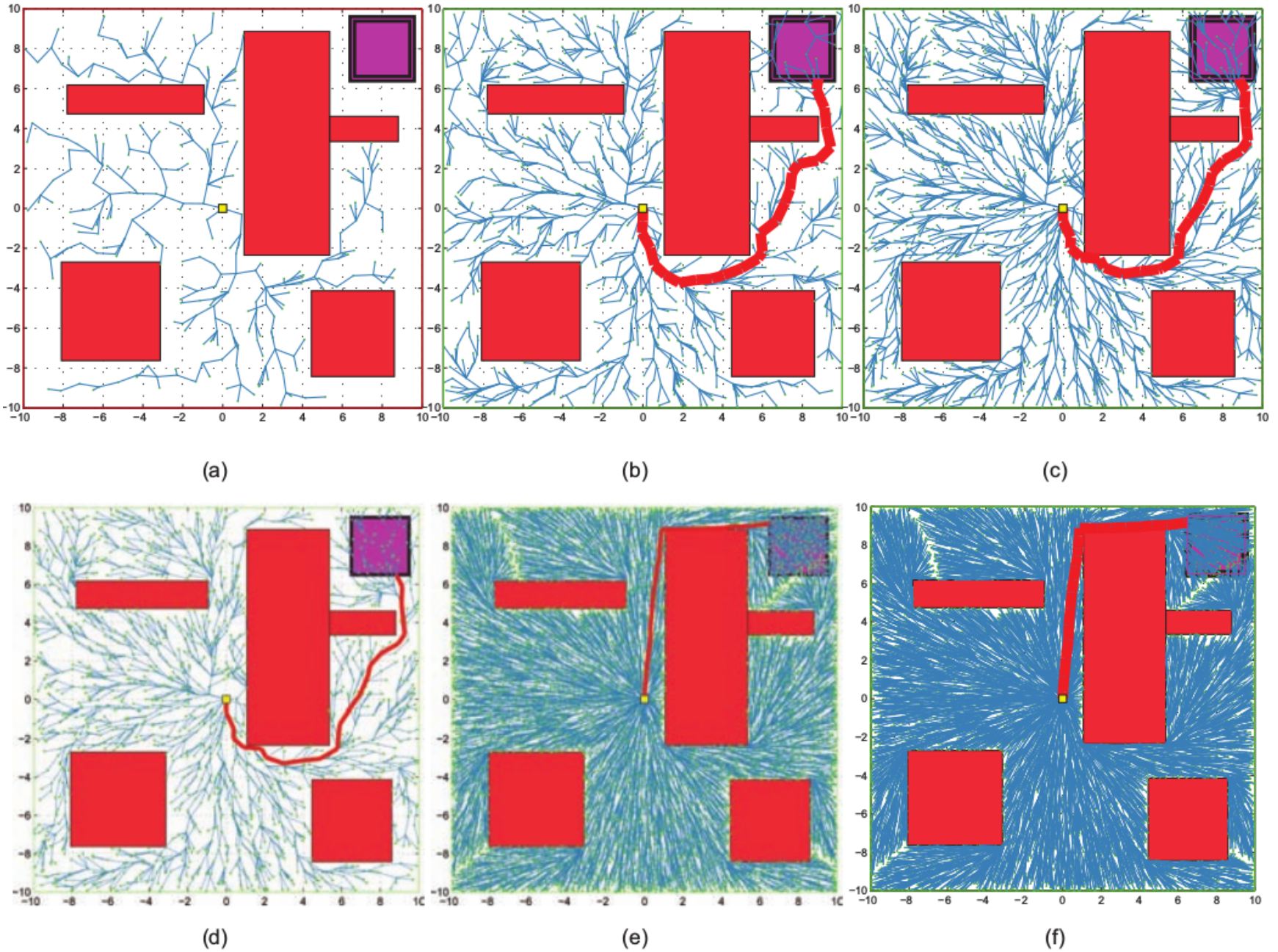
Do not just connect x_{new} to x_{near}

Attempt to connect to every vertex within a radius *r*

Get position and cost of min-cost vertex in X_{near}

Rewire parents of nodes in X_{near} to go through x_{new} if that is faster

RRT*



Other considerations in RRTs

- Reaching the goal
- Extracting the solution
- Kinodynamic planning
- Completeness
- Optimality
- Smoothing the path

Path smoothing (applies to all methods so far)

Paths produced by sampling-based motion planners
    are generally not smooth

RRT* and PRM* converge to
    (smooth) optimal paths in the limit,
    but generally not possible to
    run these algorithms long enough to converge

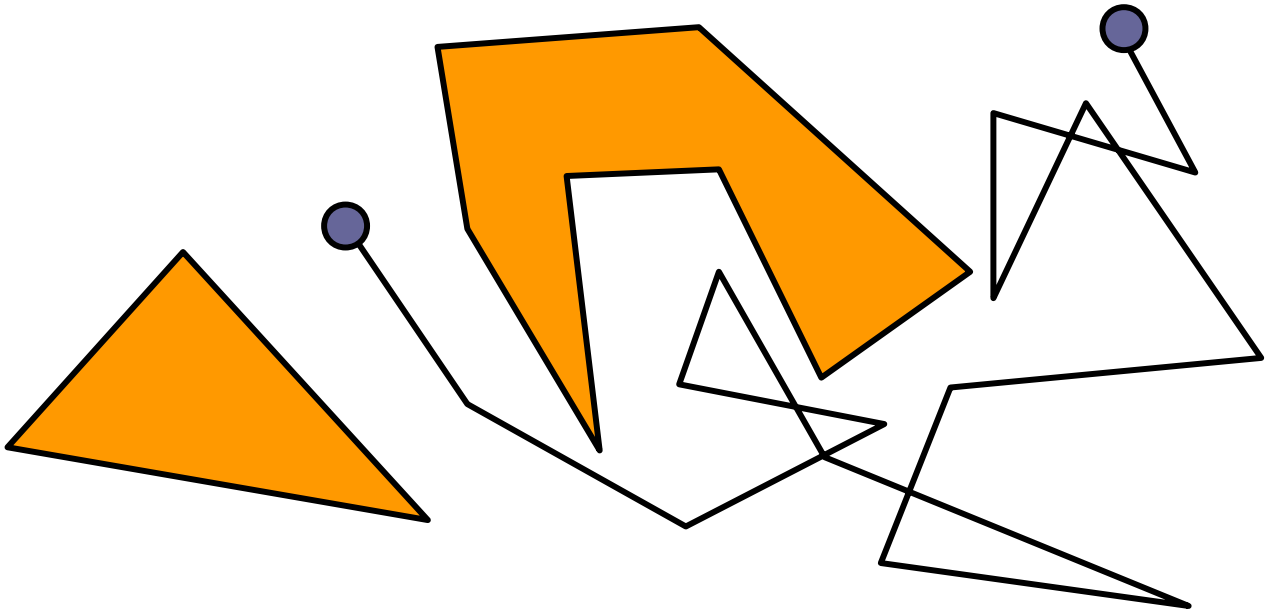
Path smoothing (applies to all methods so far)

Paths produced by sampling-based motion planners
    are generally not smooth

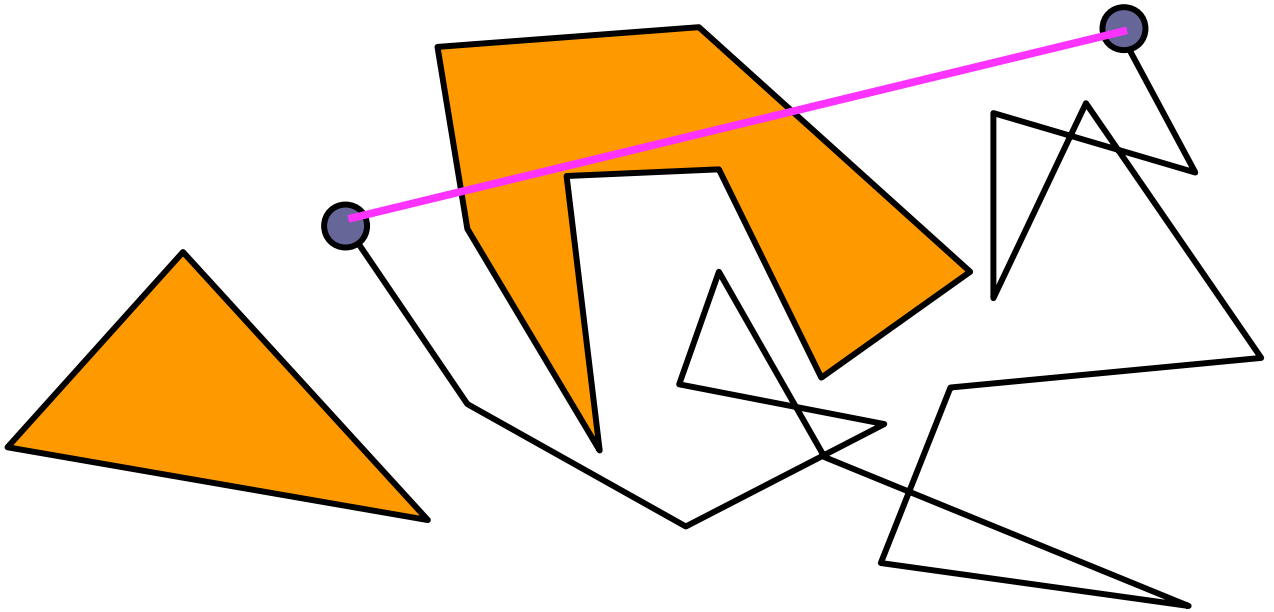
RRT* and PRM* converge to
    (smooth) optimal paths in the limit,
    but generally not possible to
    run these algorithms long enough to converge

Simple solution (not best but works):
    Consider removing unnecessary vertices

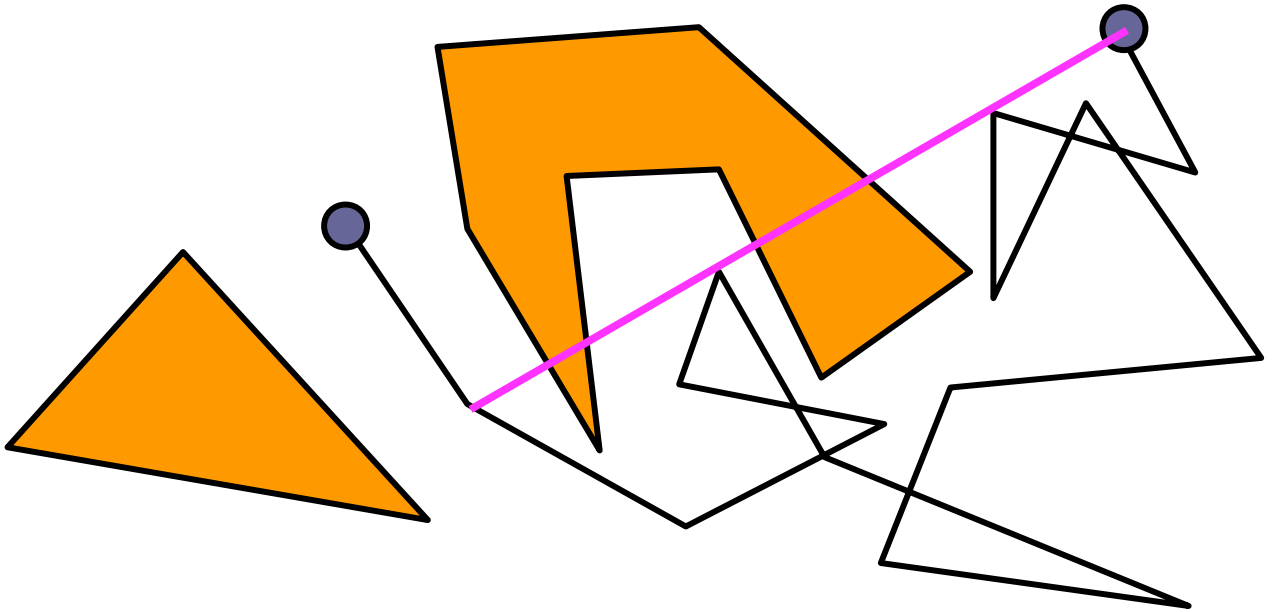
Smoothing the path



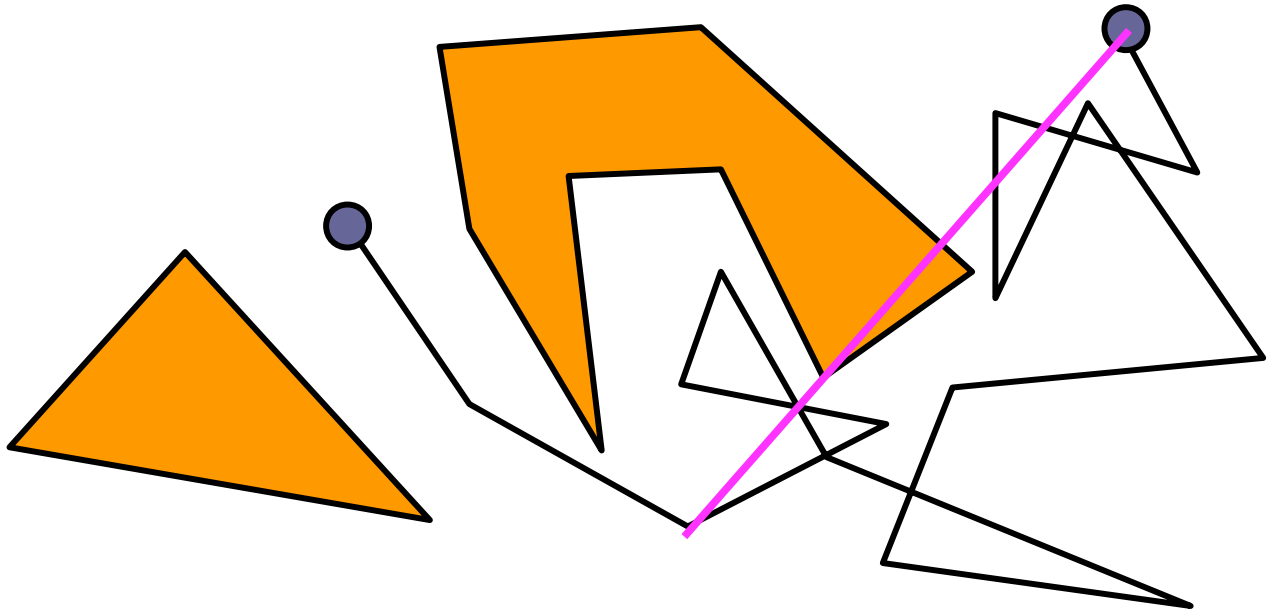
Smoothing the path



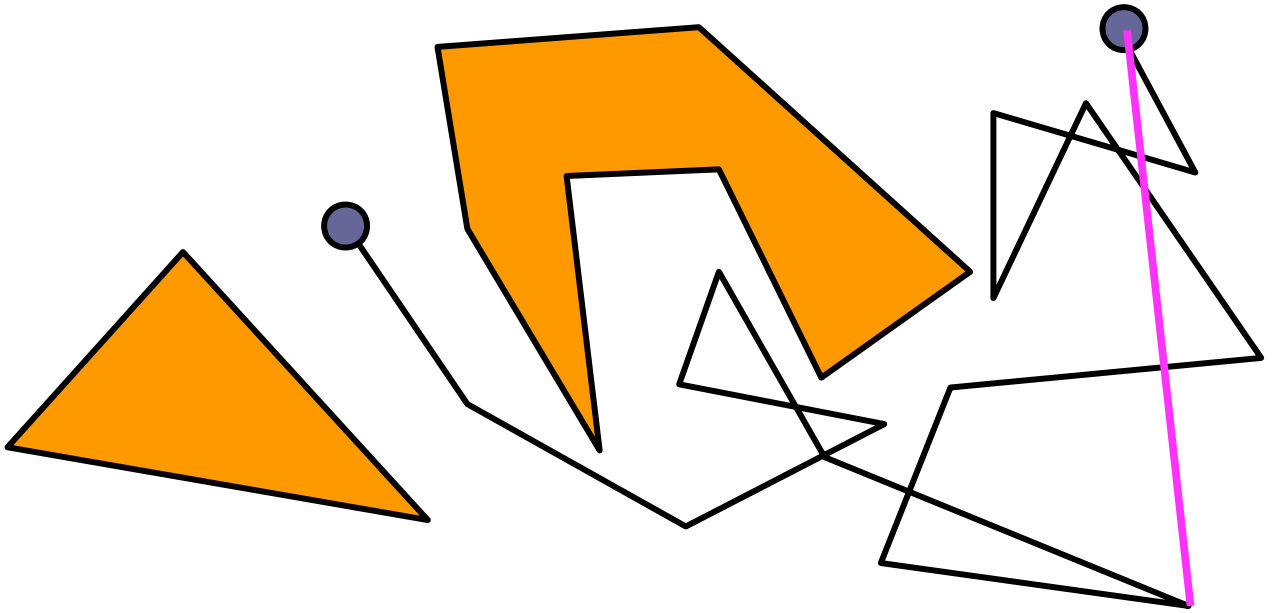
Smoothing the path



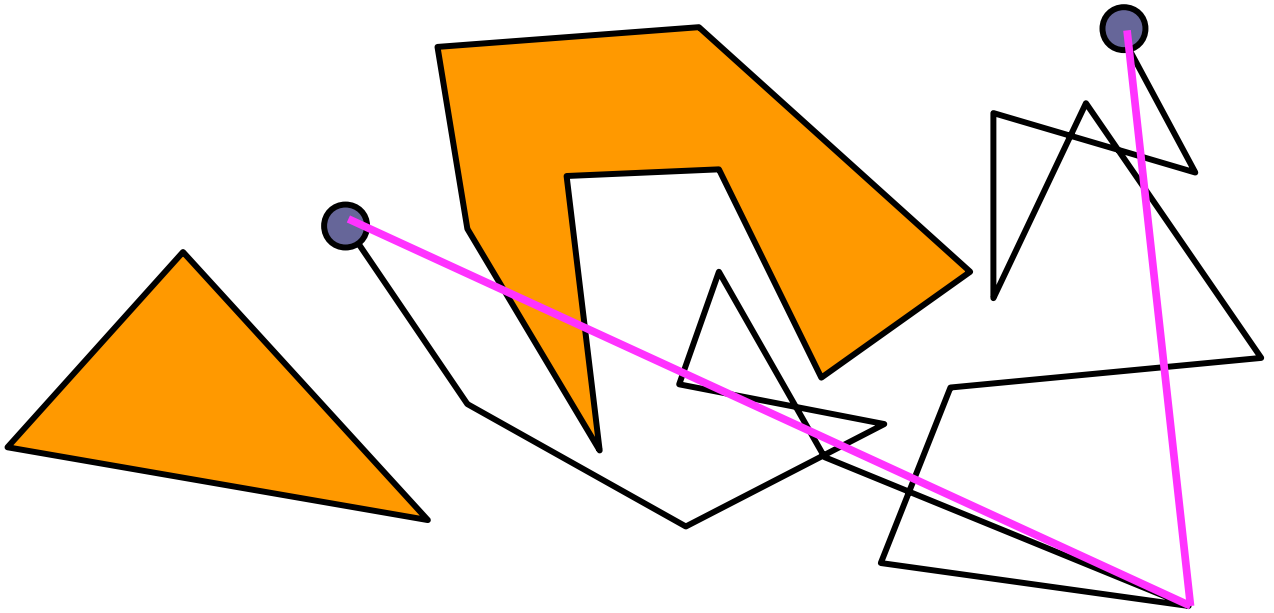
Smoothing the path



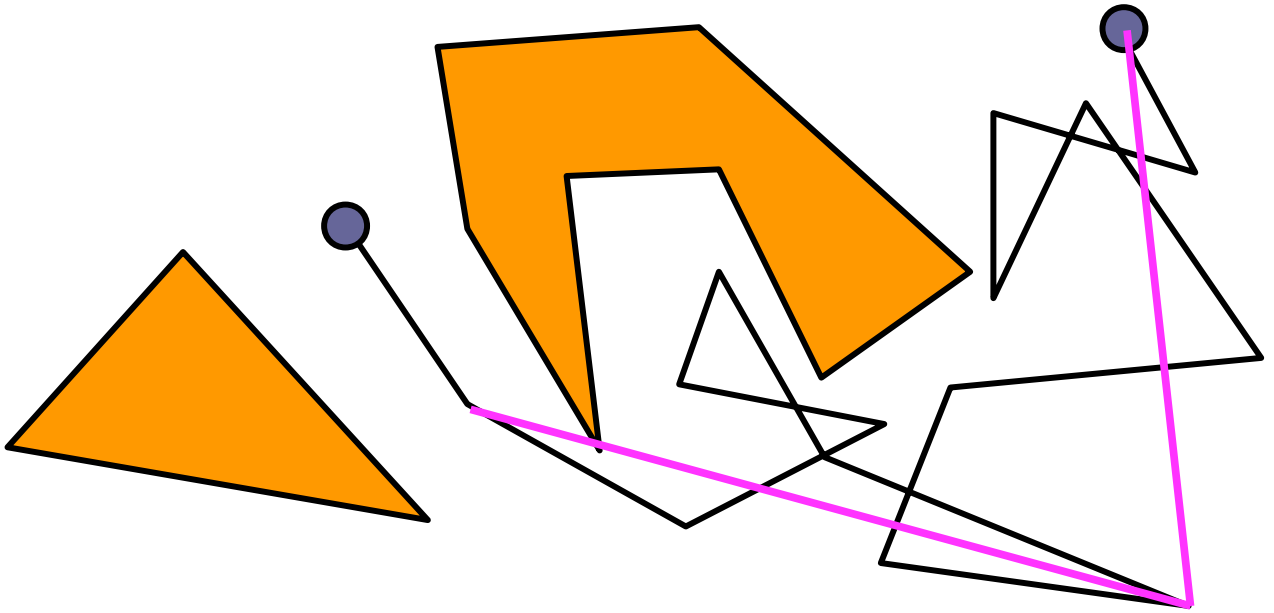
Smoothing the path



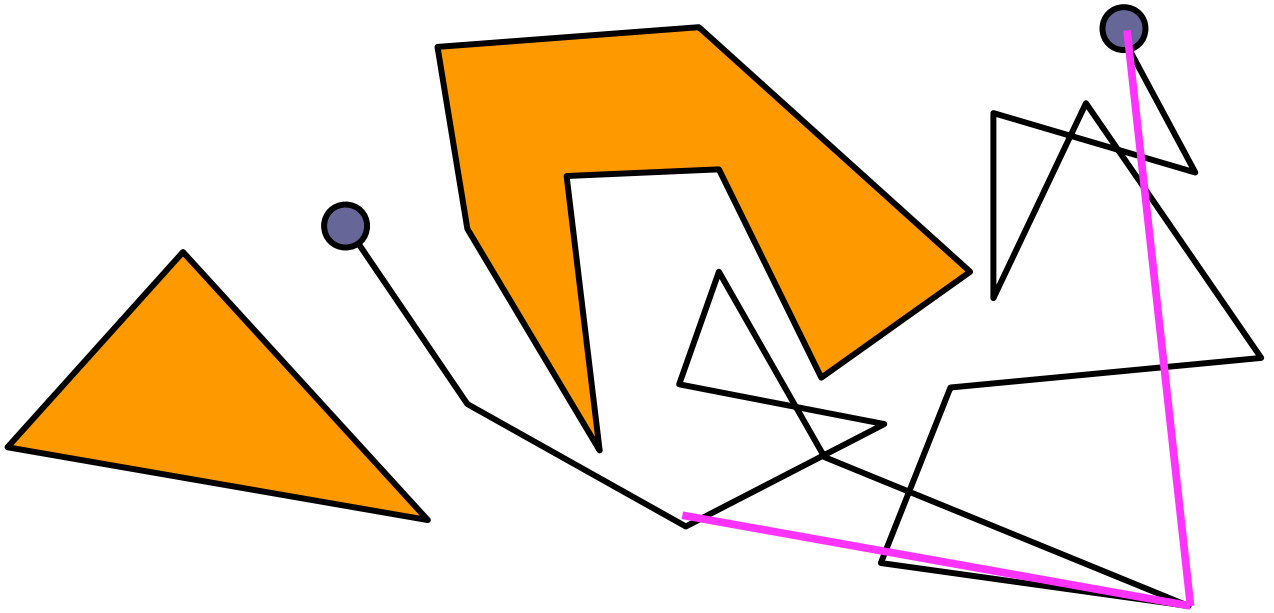
Smoothing the path



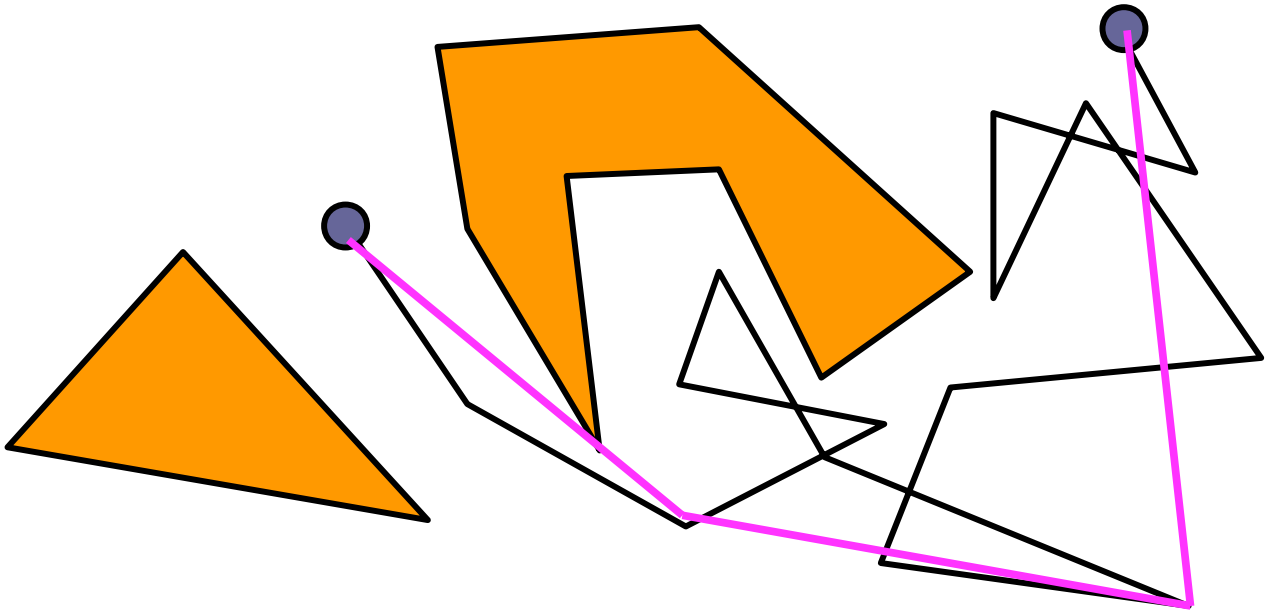
Smoothing the path



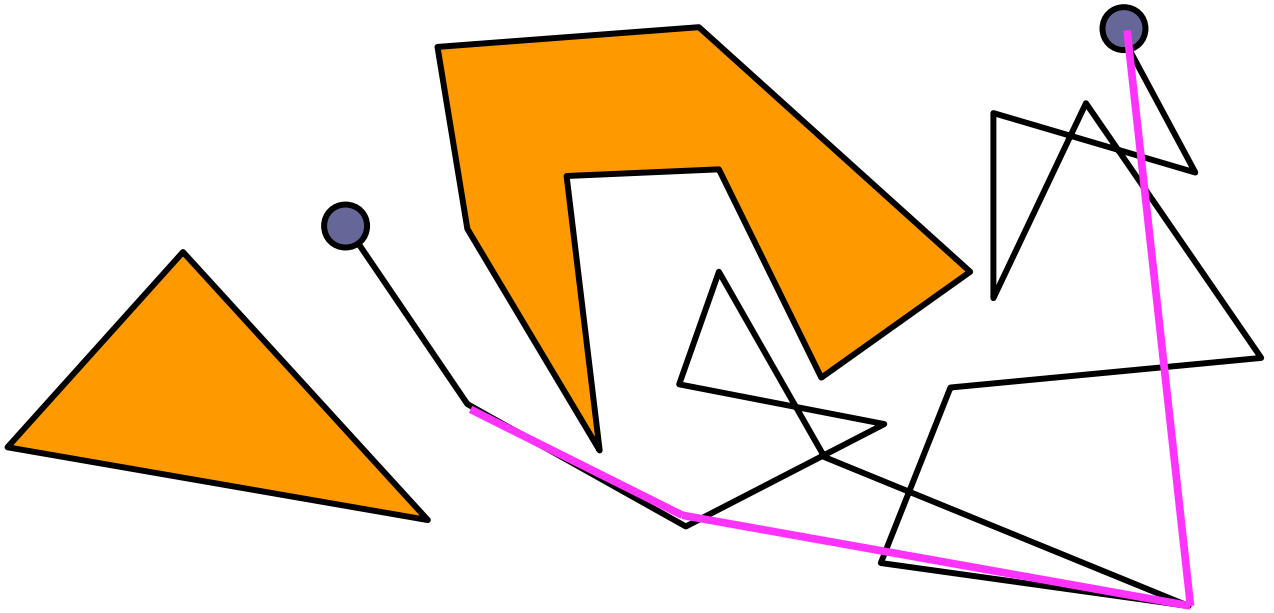
Smoothing the path



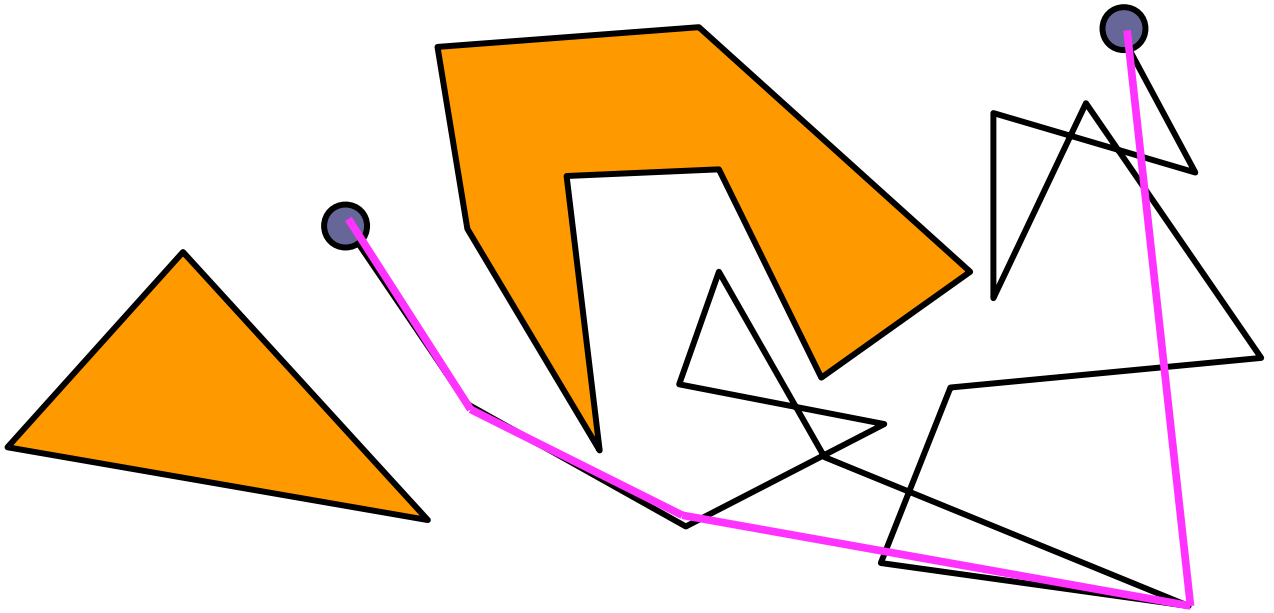
Smoothing the path



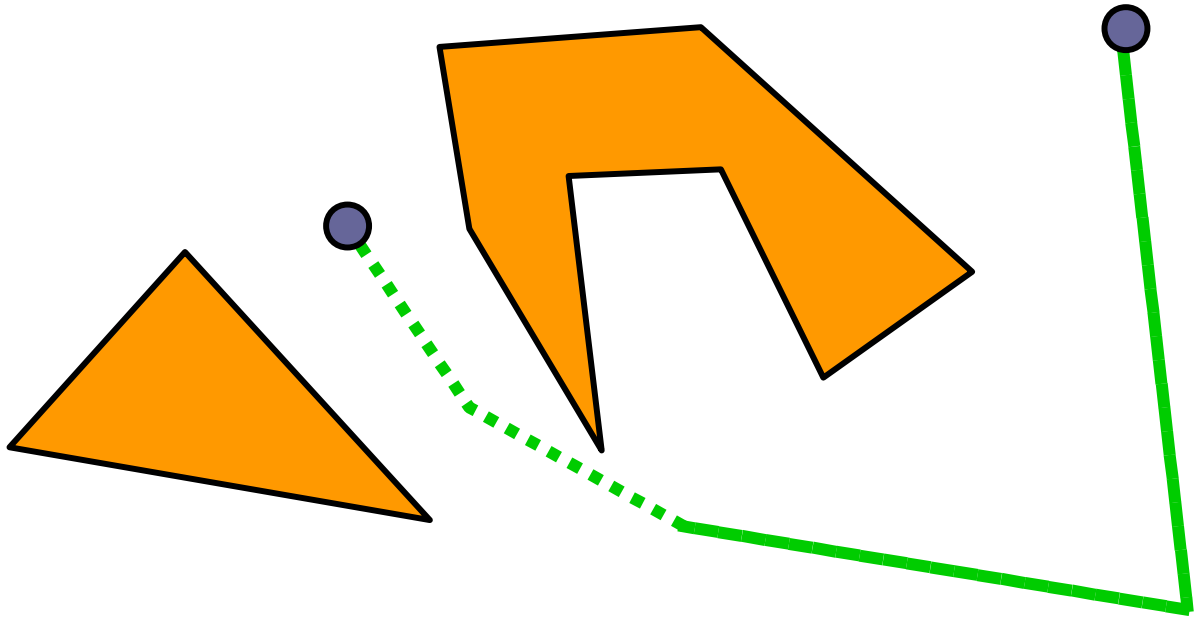
Smoothing the path



Smoothing the path



Smoothing the path



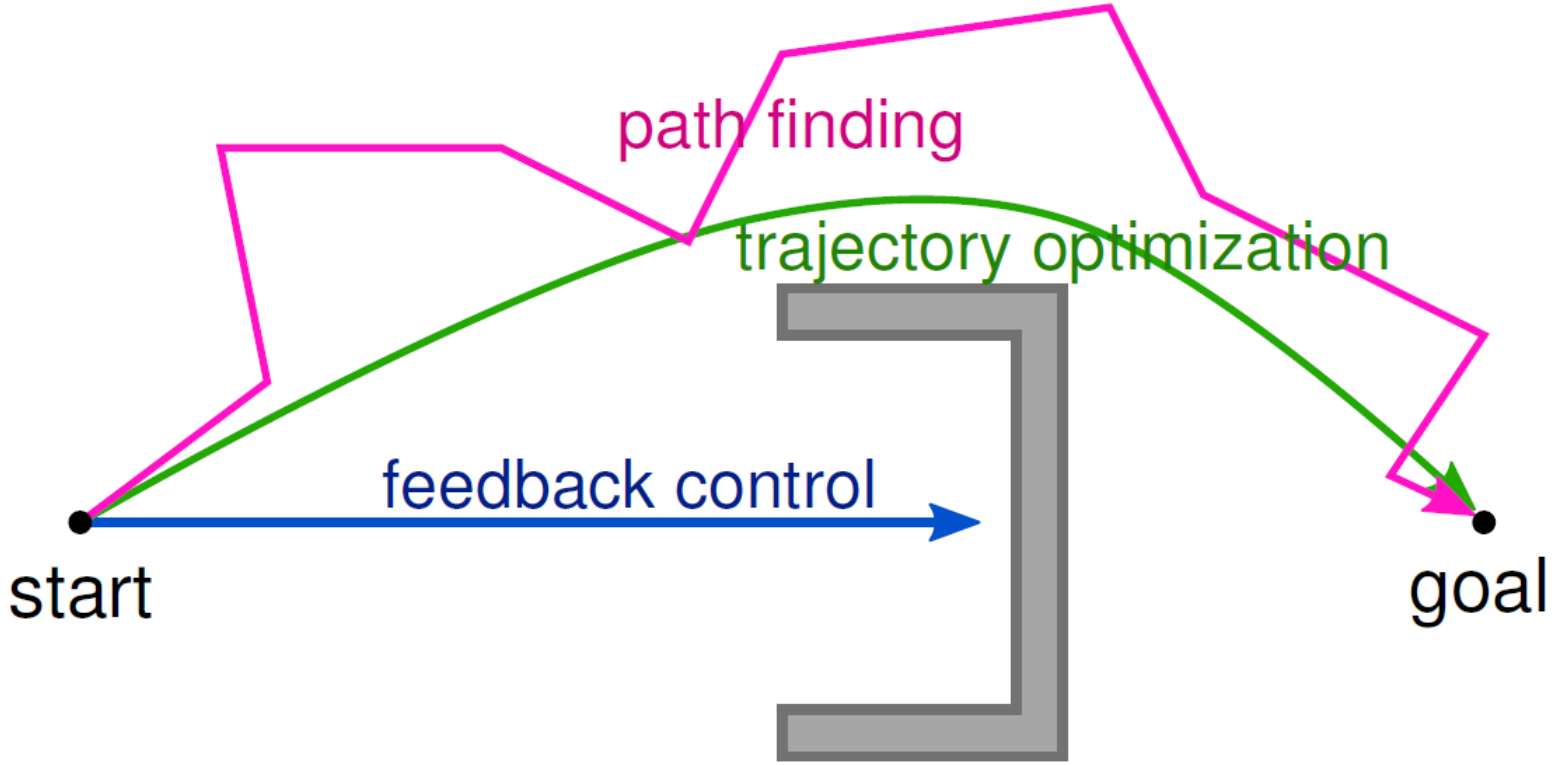
Summary: RRTs

- Pros (shared with PRMs):
  - Algorithmically very simple
  - Highly explorative
  - Allows probabilistic performance guarantees

- Pros (beyond PRMs):
  - Focus computation on single query $(q_{\text{start}}, q_{\text{goal}})$ problem
  - Trees from multiple queries can be merged to a roadmap
  - Can be extended to differential constraints (nonholonomic systems)

- To keep in mind (shared with PRMs):
  - The metric (for nearest neighbor selection) is sometimes critical
  - The local planner may be non-trivial

Feedback control, path finding, trajectory optim.



- Feedback Control: E.g., $q_{t+1} = q_t + J^\sharp(y^* - \phi(q_t))$
- Trajectory Optimization: $\text{argmin}_{q_{0:T}} f(q_{0:T})$
- Path Finding: Find some $q_{0:T}$ with only valid configurations

Feedback

Piazza thread: 2/9 Lec 07 Feedback

Please post your answers to the following anonymously.

1. What did you like so far?

2. What was unclear?

3. Any additional feedback / comments?