

CS 4610/5335 – Lecture 8

Sampling-based Motion Planning (continued)

Lawson L.S. Wong
Northeastern University
2/14/22

Material adapted from:

1. Robert Platt, CS 4610/5335
2. Marc Toussaint, U. Stuttgart Robotics Course
3. Frank Dellaert, Georgia Tech CS 6601
4. Emilio Frazzoli, MIT 16.410/16.413
5. Sebastian Thrun, Wolfram Burgard, & Dieter Fox,
Probabilistic Robotics

Announcements

Ex2 released, due 2/25

Make your project Piazza thread if you are in a team!

- Will have time to work on it later today as well

Upcoming: Localization and mapping

- Time to refresh probability knowledge
- Especially Gaussian distributions

Announcements

Ex2 released, due 2/25

Make your project Piazza thread if you are in a team!

- Will have time to work on it later today as well

Upcoming: Localization and mapping

- Time to refresh probability knowledge
- Especially Gaussian distributions

Some comments regarding Ex1

Outline

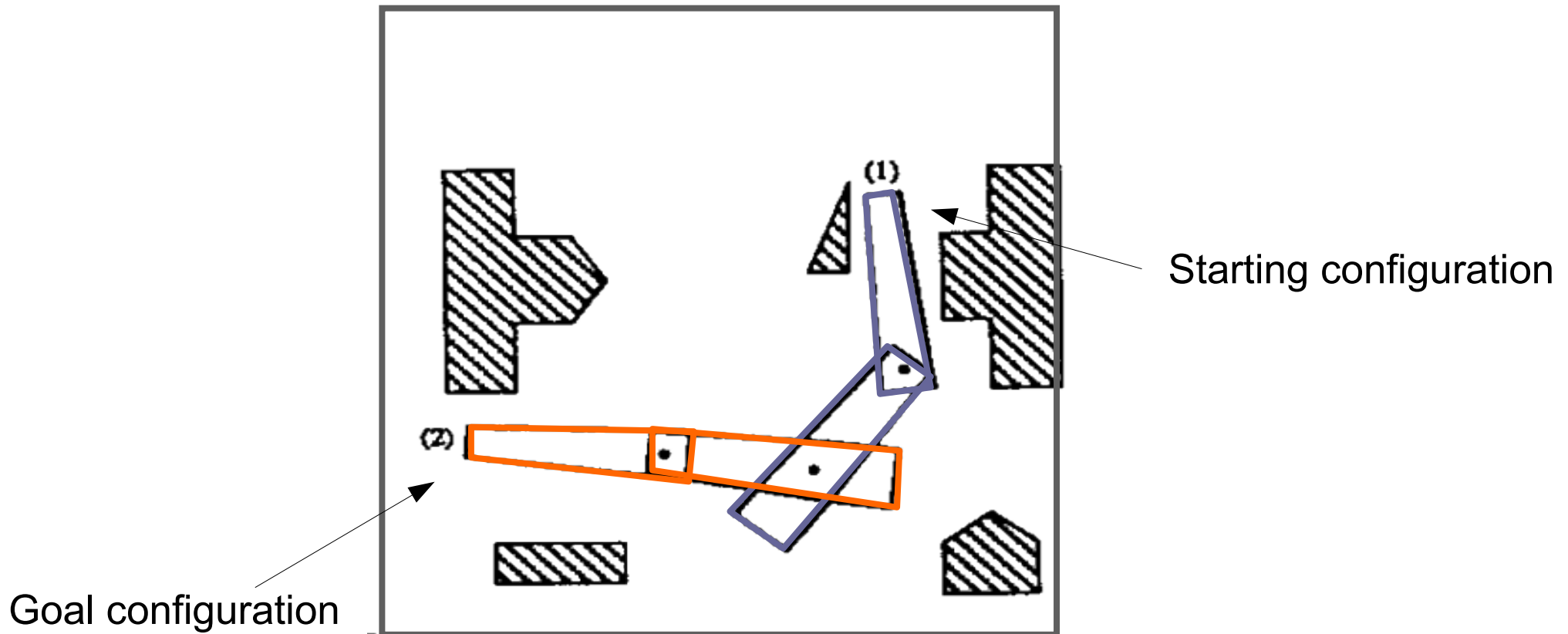
More on RRTs

Overview of localization

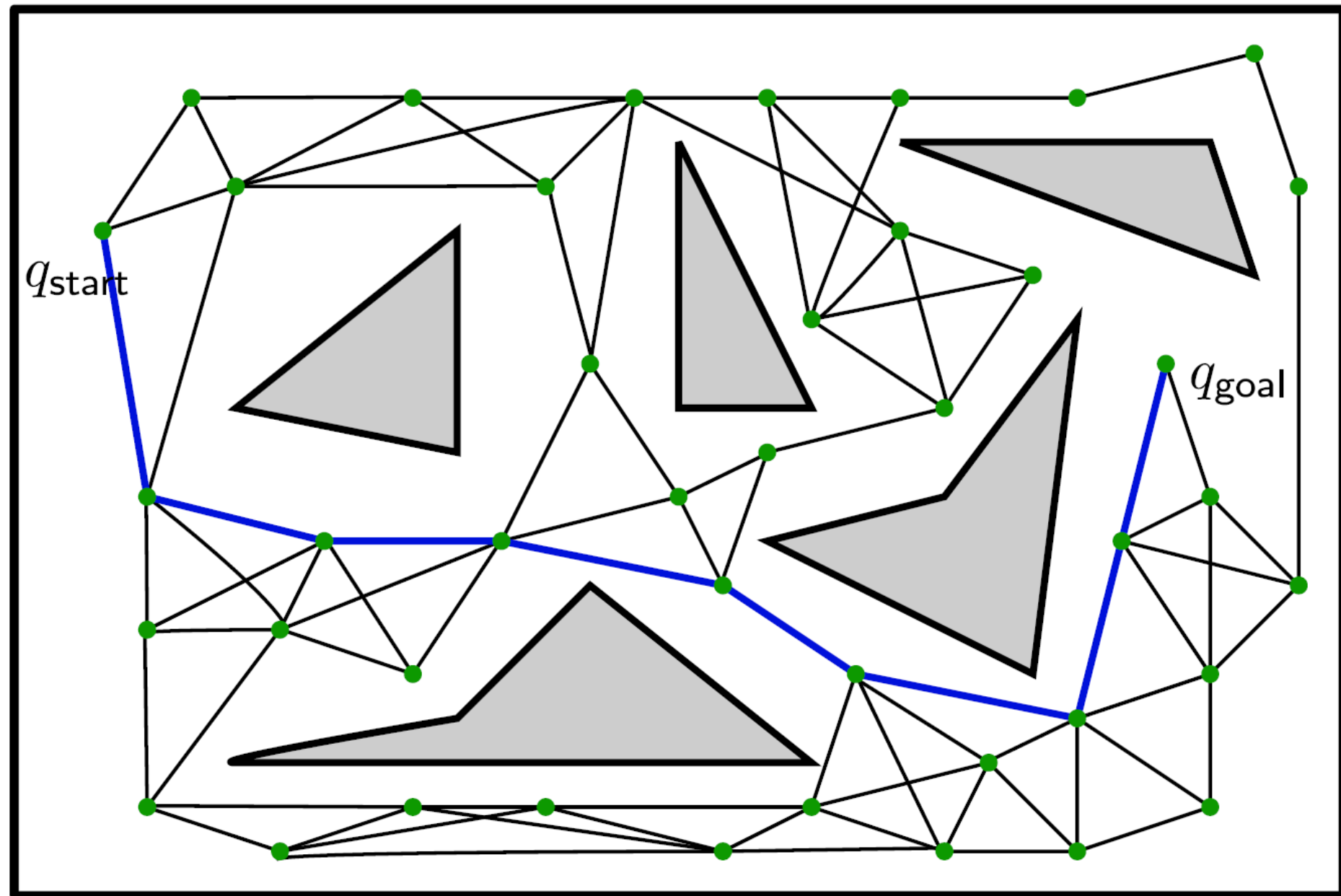
Project

Last week: Motion planning

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



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: 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  $\text{path}(q, q') \in Q_{\text{free}}$  then  $E \leftarrow E \cup \{(q, q')\}$ 
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

Recap: 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)

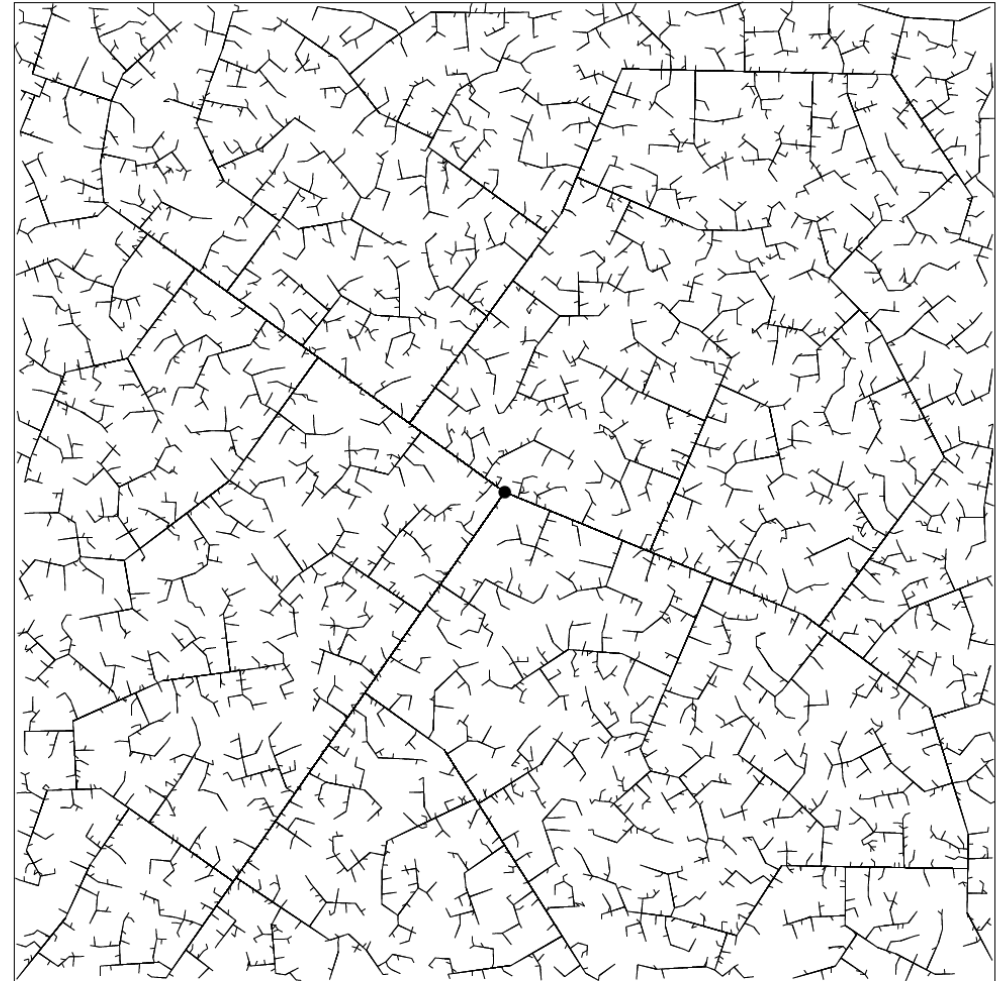
RRTs solve both problems:

- 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)



45 iterations



2345 iterations

Figure 5.19: In the early iterations, the RRT quickly reaches the unexplored parts.

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

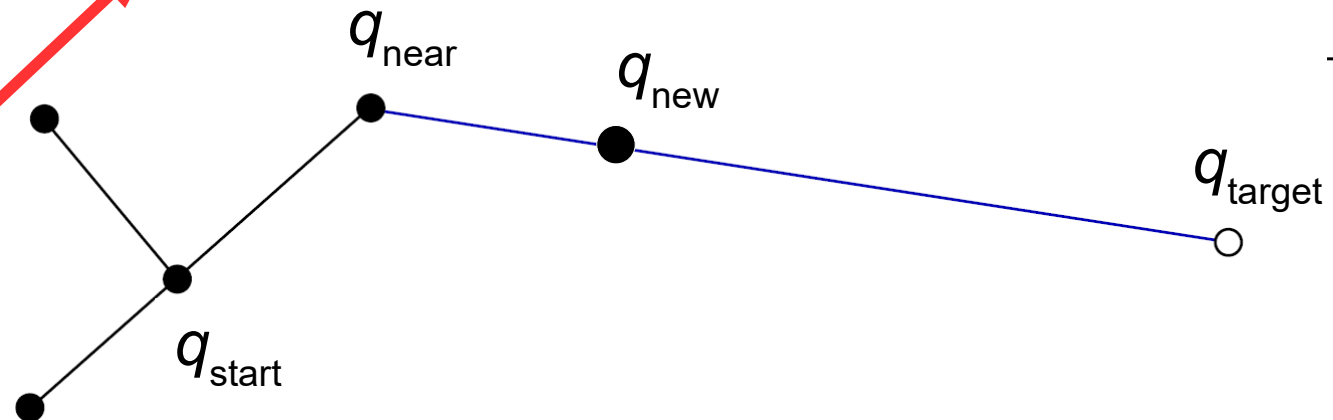
4: $q_{\text{near}} \leftarrow$ nearest neighbor of q_{target} in V

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**

and $(q_{\text{near}}, q_{\text{new}})$
collision free



Rapidly Exploring Random Trees


RRT growing directedly towards the goal

Input: q_{start} , q_{goal} , number n of nodes, stepsize α , β

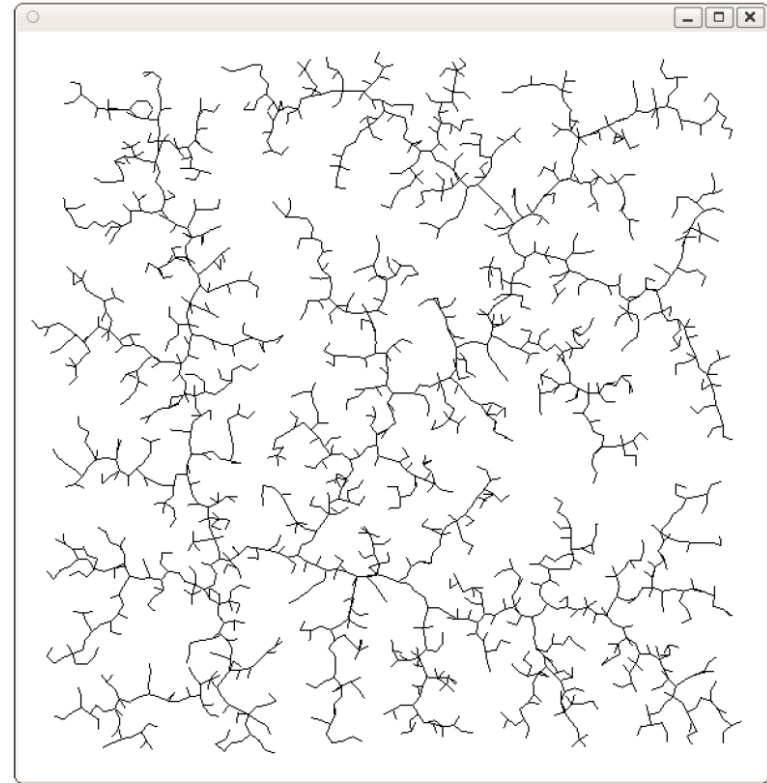
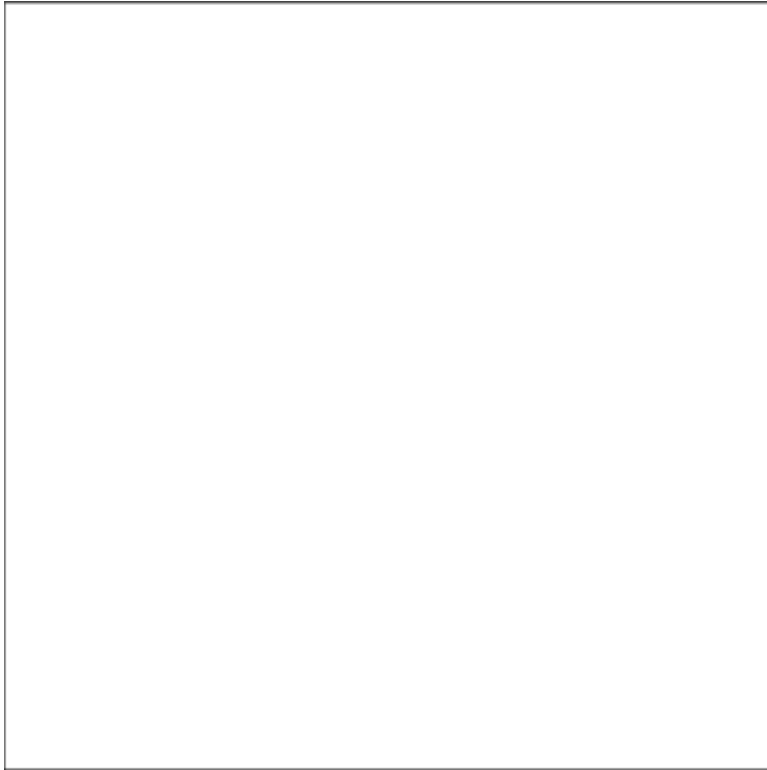
Output: tree $T = (V, E)$

- 1: initialize $V = \{q_{\text{start}}\}$, $E = \emptyset$
 - 2: **for** $i = 0 : n$ **do**
 - 3: **if** $\text{rand}(0, 1) < \beta$ **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**
-

and $(q_{\text{near}}, q_{\text{new}})$
collision free



Extracting the solution – it's a tree!



$$n = 2000$$

Each node in tree has pointer to parent

- Follow path from goal (leaf) back to start (root)

Other considerations in RRTs

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

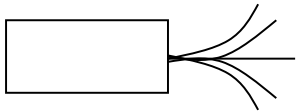
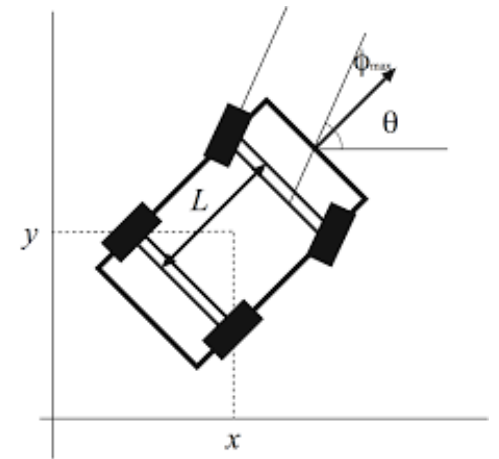
Kinodynamic planning

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

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** $V \leftarrow V \cup \{q_{\text{new}}\}$, $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$

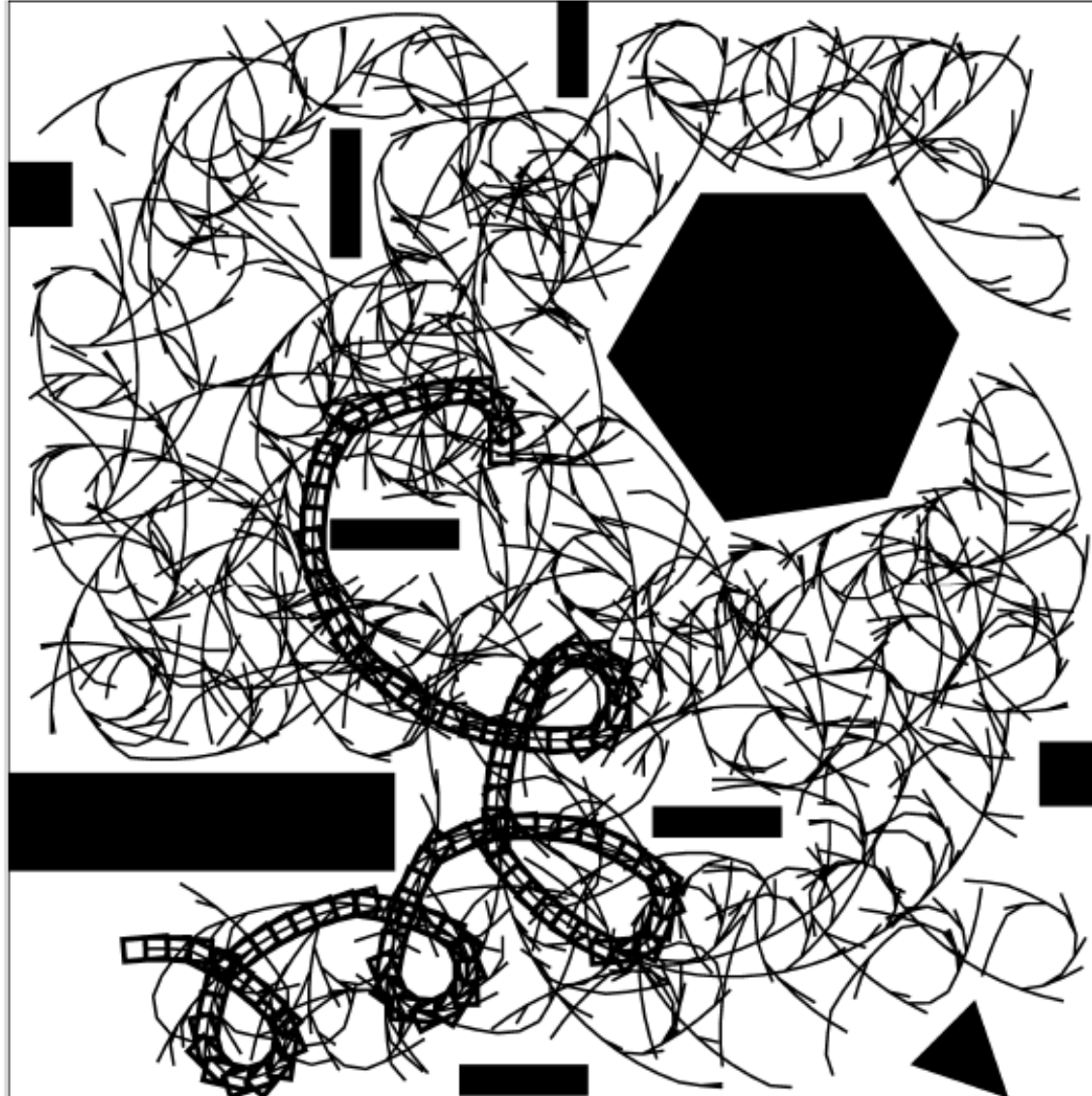
7: **end for**

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



Other considerations in RRTs

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

Probabilistic completeness

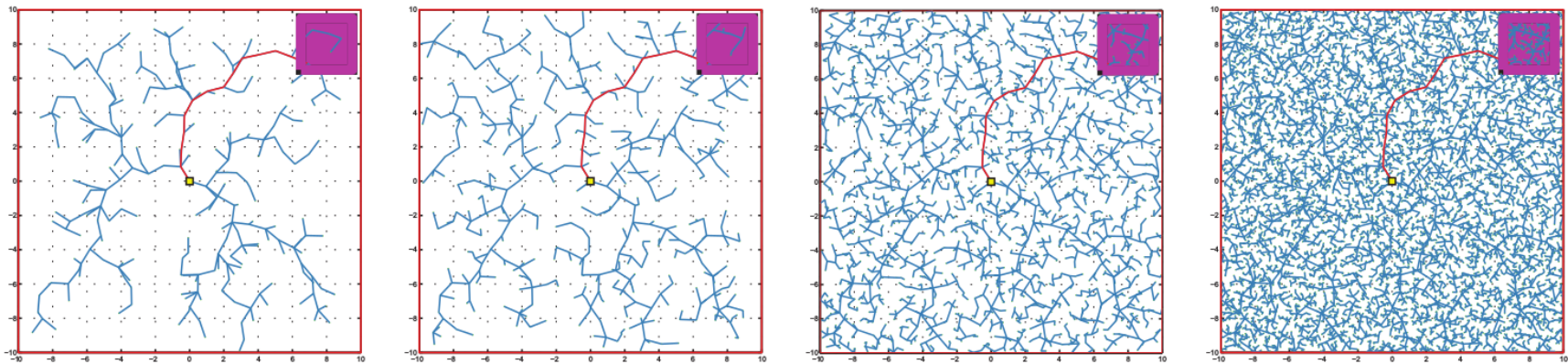
Theorem 16 (Probabilistic completeness of RRT (LaValle and Kuffner 2001)). *Consider a robustly feasible path planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. There exist constants $a > 0$ and $n_0 \in \mathbb{N}$, both dependent only on $\mathcal{X}_{\text{free}}$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

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

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  $V = \emptyset, E = \emptyset$ 
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  $\text{path}(q, q') \in Q_{\text{free}}$  then  $E \leftarrow E \cup \{(q, q')\}$ 
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.

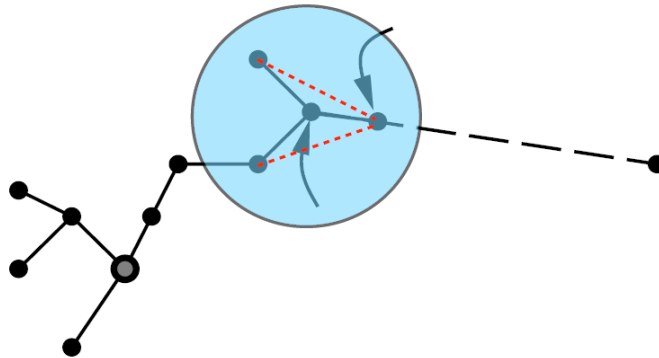
```
1  $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset;$ 
2 for  $i = 1, \dots, n$  do
3    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
4    $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}});$ 
5    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
6   if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
7      $X_{\text{near}} \leftarrow \text{Near}(G =$ 
       $(V, E), x_{\text{new}}, \min\{\gamma_{\text{RRG}}(\log(\text{card}(V)) /$ 
       $\text{card}(V))^{1/d}, \eta\});$ 
8      $V \leftarrow V \cup \{x_{\text{new}}\};$ 
9      $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{nearest}})\};$ 
10    foreach  $x_{\text{near}} \in X_{\text{near}}$  do
11      if  $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}})$  then
12         $E \leftarrow E \cup \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$ 
13
14 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

RRG



- 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  $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset;$ 
2 for  $i = 1, \dots, n$  do
3    $x_{\text{rand}} \leftarrow \text{SampleFree}_i;$ 
4    $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}});$ 
5    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
6   if  $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$  then
7      $X_{\text{near}} \leftarrow \text{Near}(G =$ 
8        $(V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V)) /$ 
9        $\text{card}(V))^{1/d}, \eta\});$ 
10     $V \leftarrow V \cup \{x_{\text{new}}\};$ 
11     $x_{\text{min}} \leftarrow x_{\text{nearest}}; c_{\text{min}} \leftarrow$ 
12     $\text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}));$ 
13    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Connect along a
14    minimum-cost path
15      if
16         $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \text{Cost}(x_{\text{near}})$ 
17         $+ c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$  then
18         $x_{\text{min}} \leftarrow x_{\text{near}}; c_{\text{min}} \leftarrow$ 
19         $\text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}))$ 
20     $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\};$ 
21    foreach  $x_{\text{near}} \in X_{\text{near}}$  do // Rewire the tree
22      if
23         $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \text{Cost}(x_{\text{new}})$ 
24         $+ c(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{Cost}(x_{\text{near}})$ 
25        then  $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
26         $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$ 
27 return  $G = (V, E);$ 

```

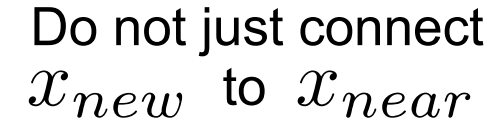
Do not just connect
 x_{new} to x_{near}

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*

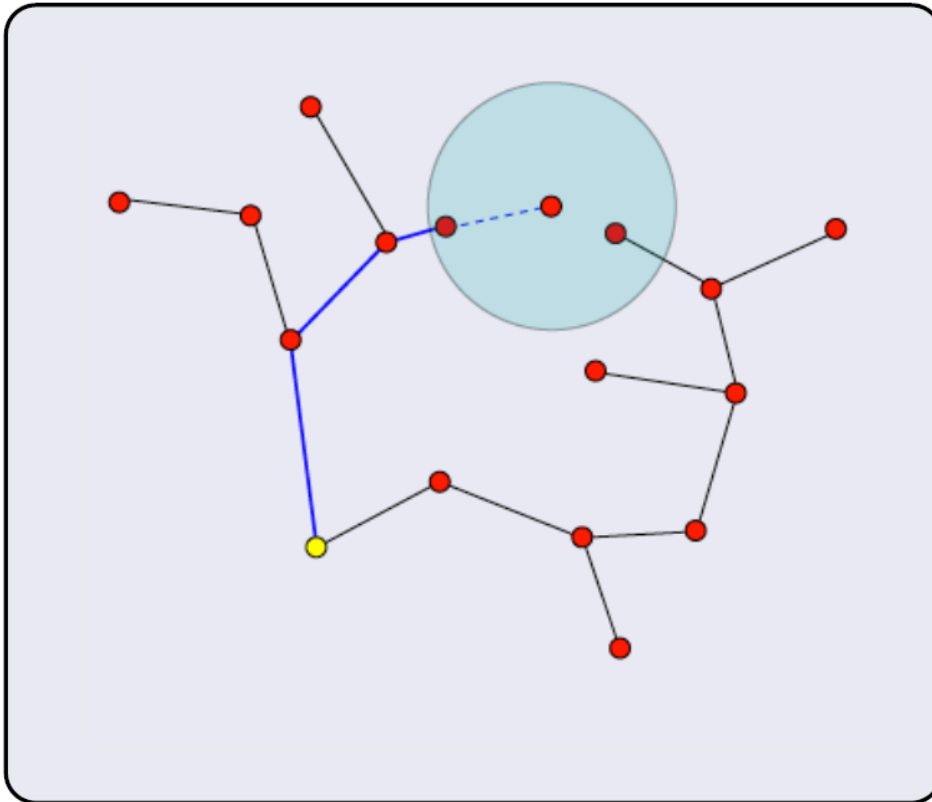


Get position and cost
of min-cost vertex in X_{near}

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}

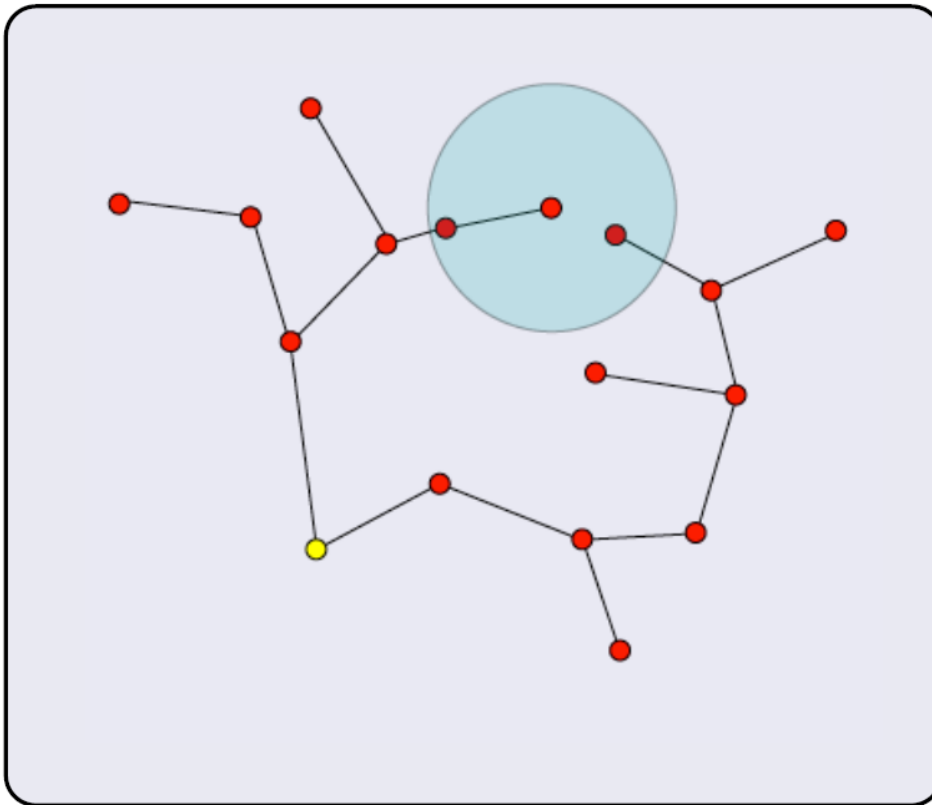
Attempt to connect to every vertex within a radius r

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}

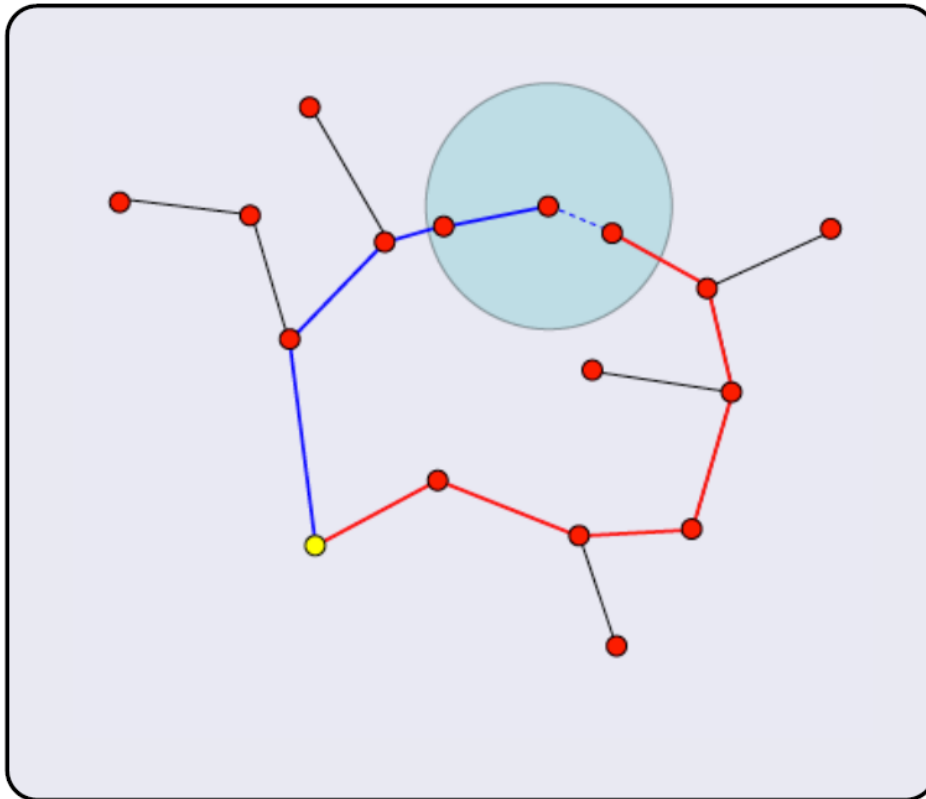
Attempt to connect to every vertex within a radius r

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}

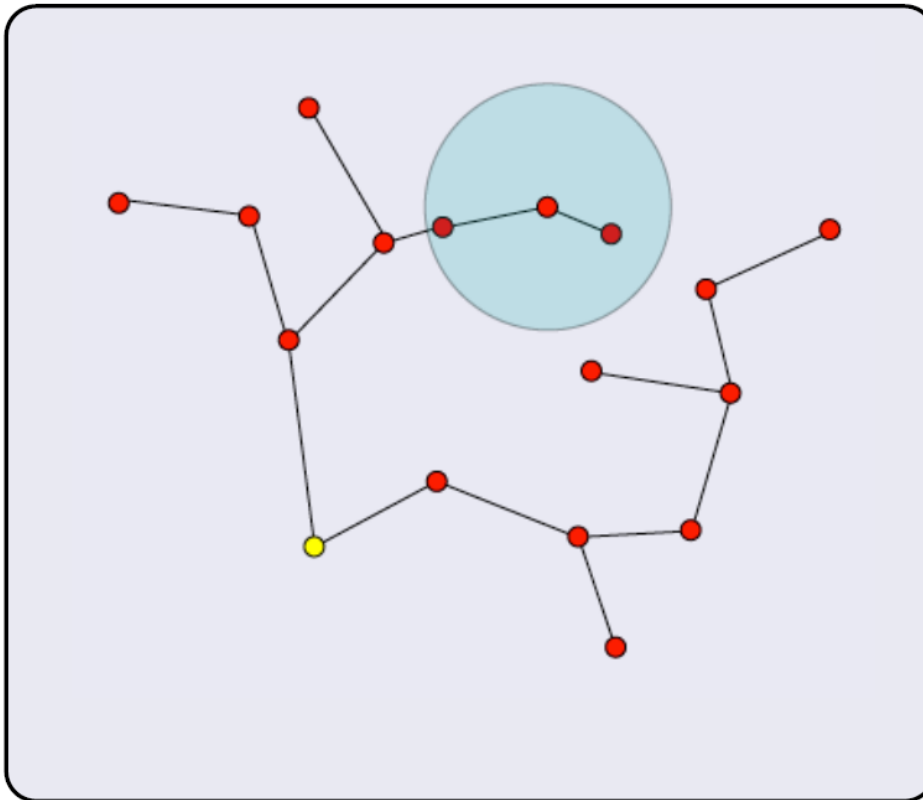
Attempt to connect to every vertex within a radius r

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*



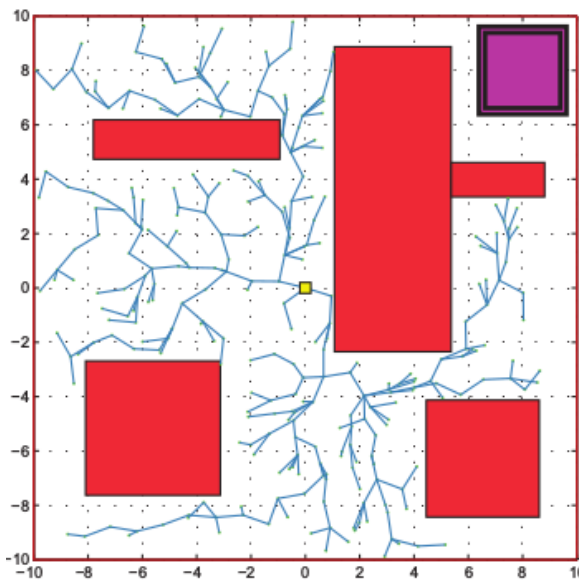
Do not just connect x_{new} to x_{near}

Attempt to connect to every vertex within a radius r

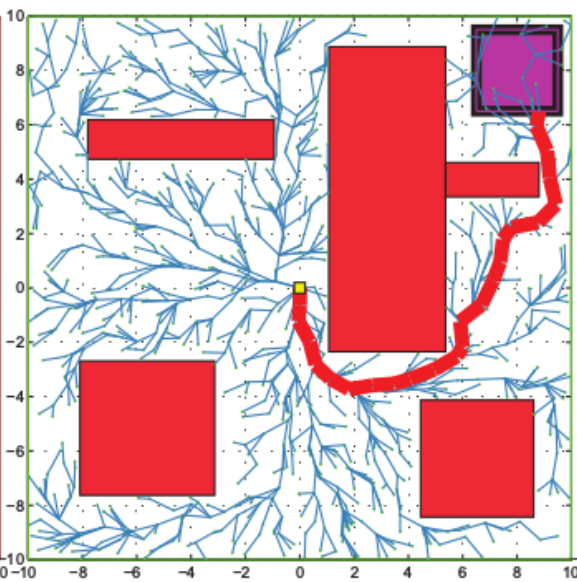
Get position and cost of min-cost vertex in X_{near}

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

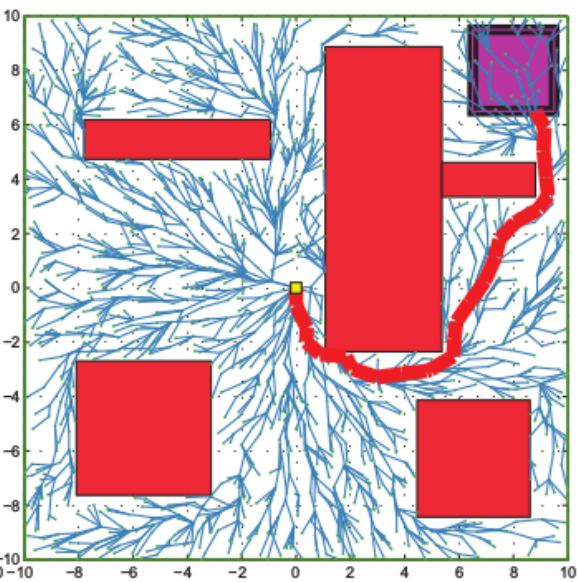
RRT*



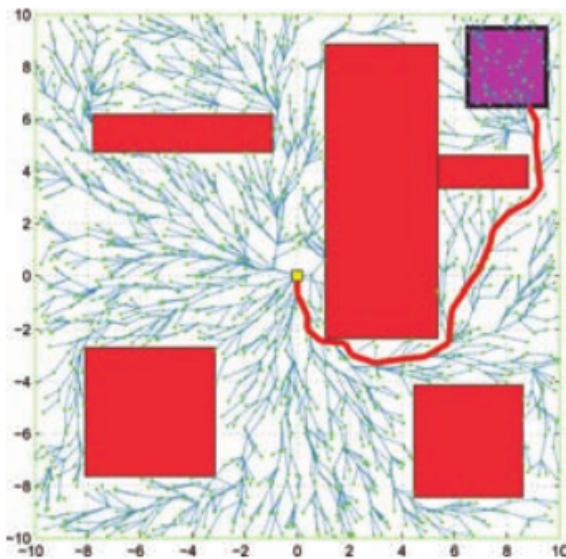
(a)



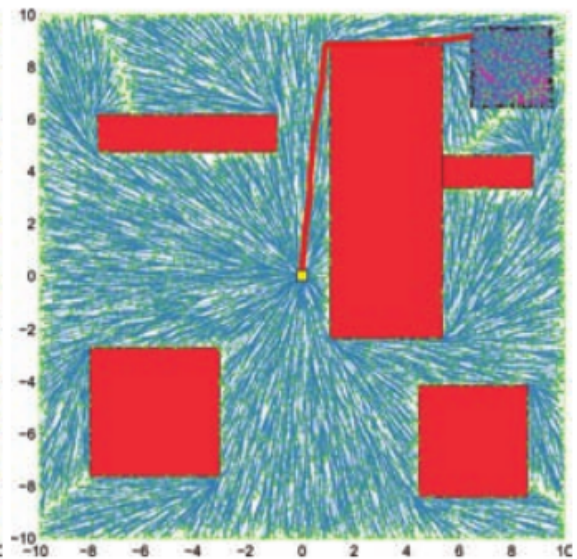
(b)



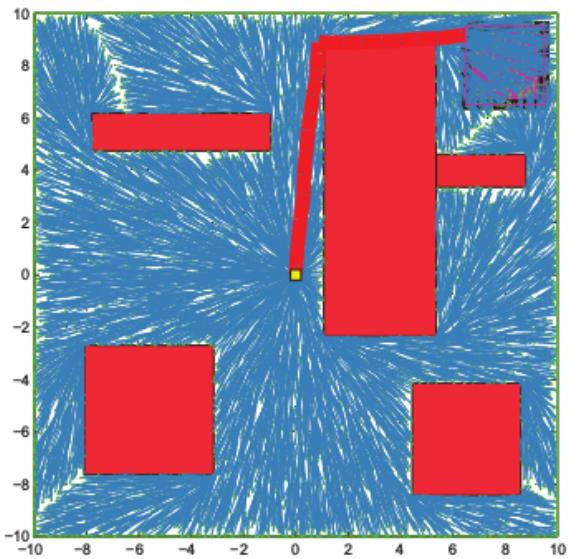
(c)



(d)



(e)



(f)

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

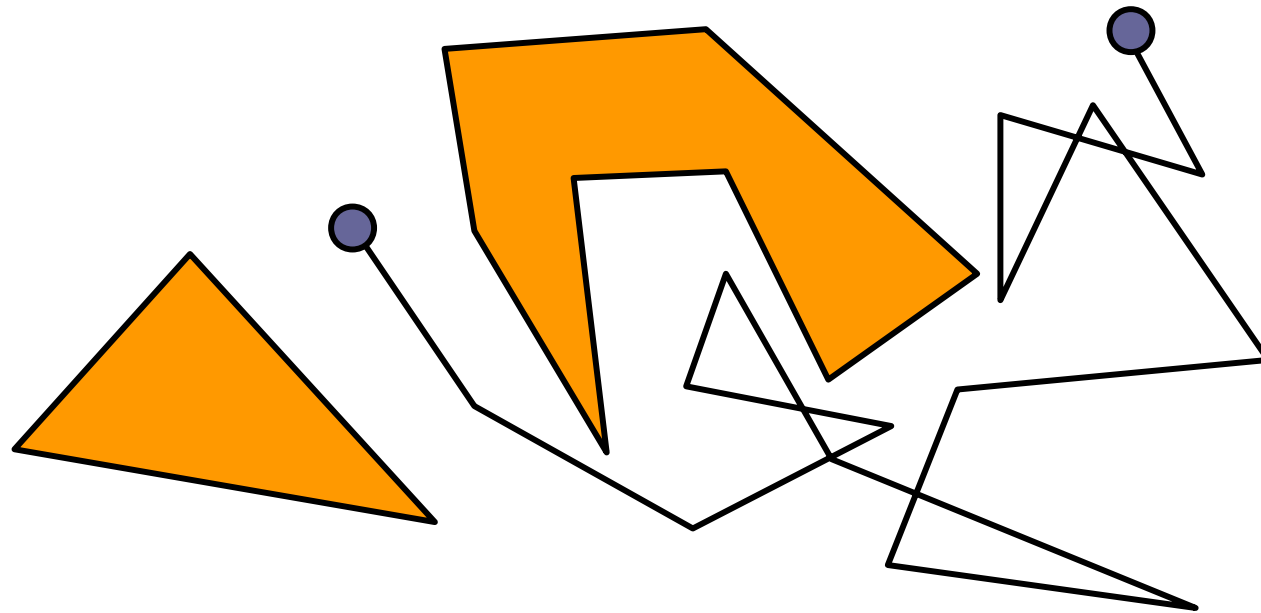
Path smoothing (applies to all methods so far)

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

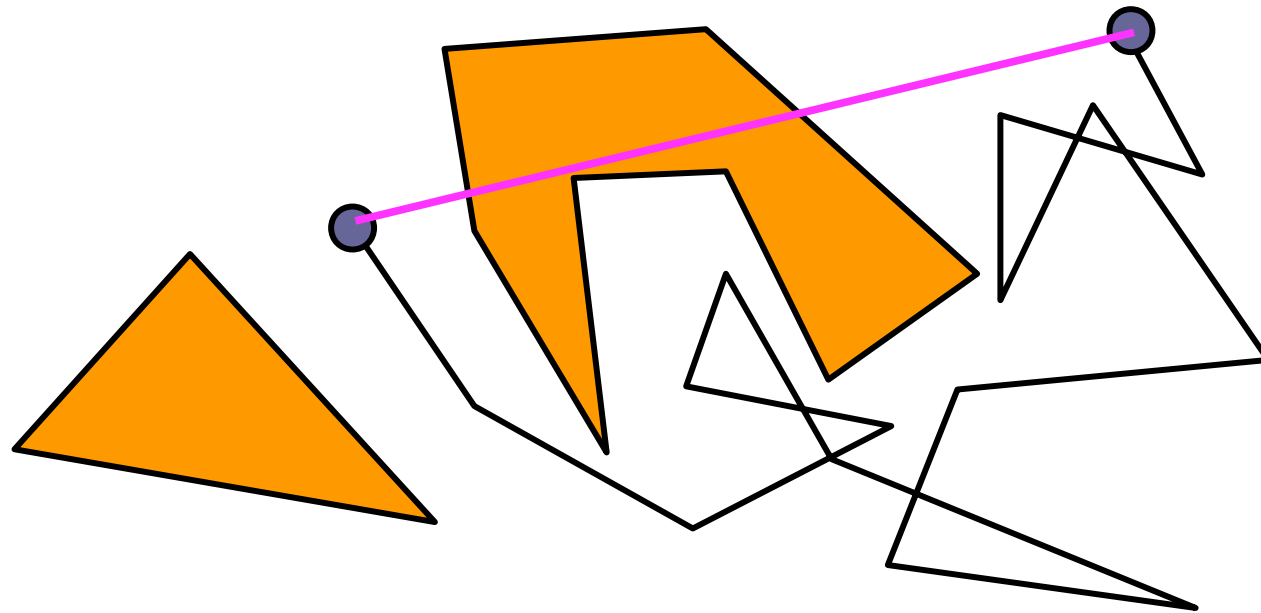
RRT* and PRM* converge to
(smooth) optimal paths in the limit,
but generally not possible to
run these algorithms long enough to converge

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

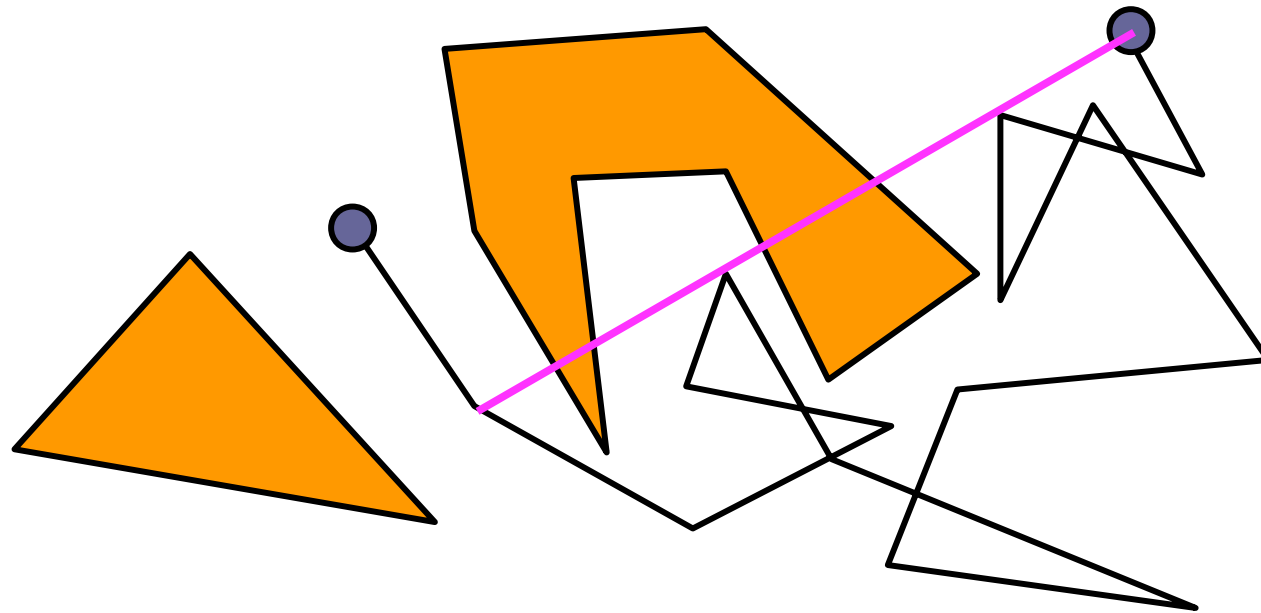
Smoothing the path



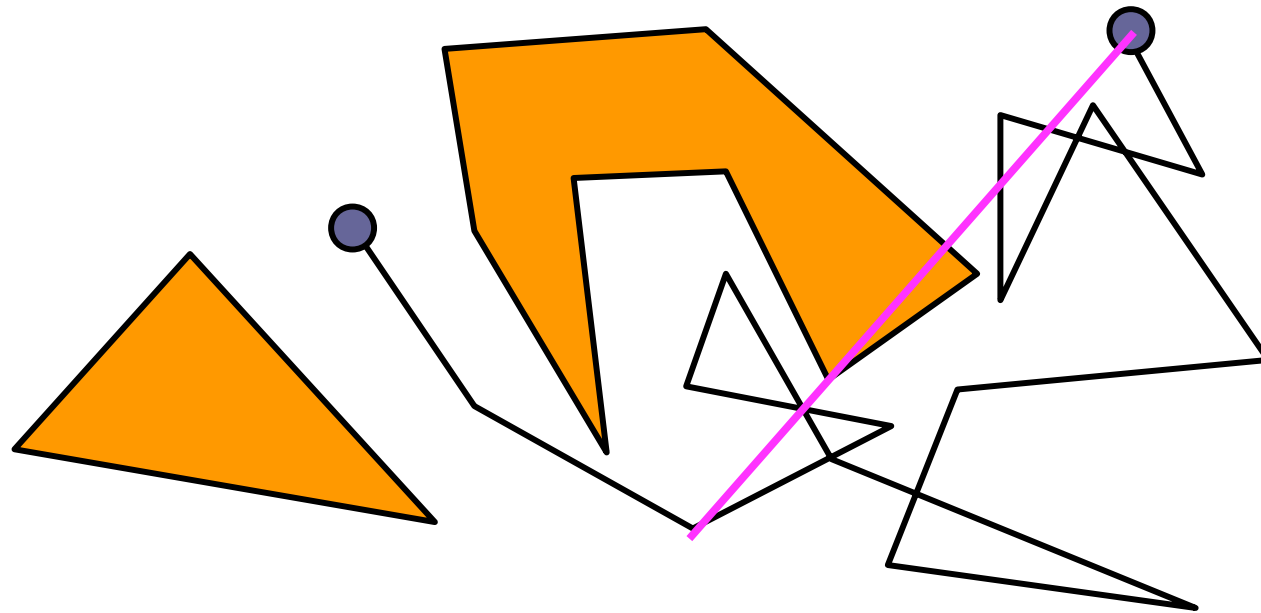
Smoothing the path



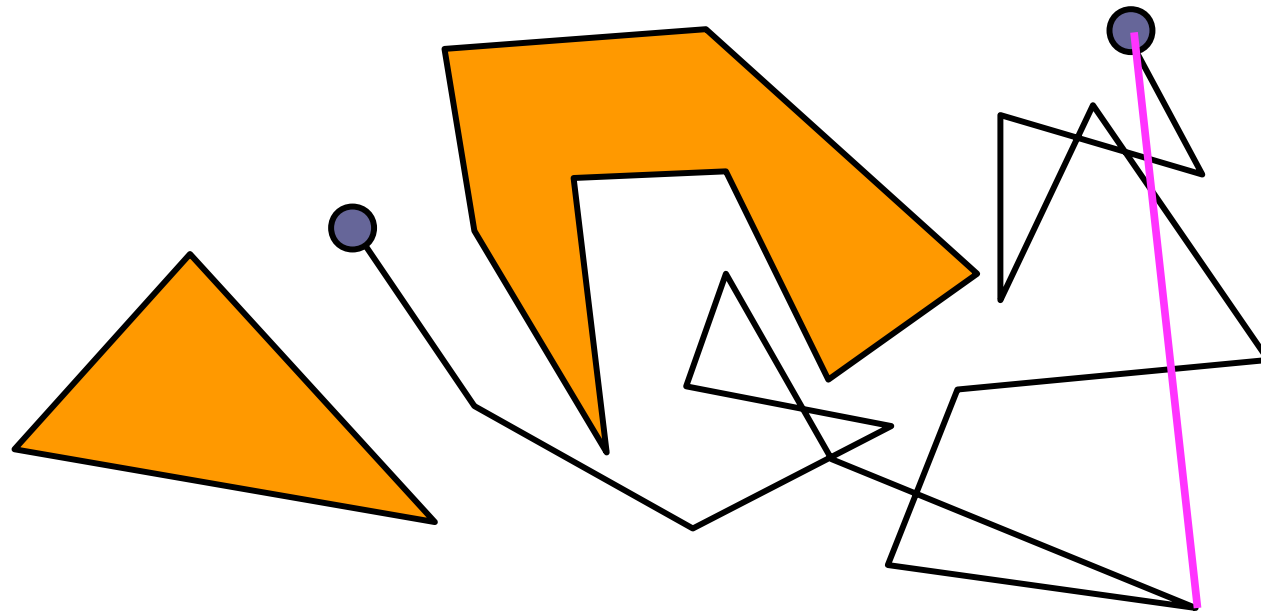
Smoothing the path



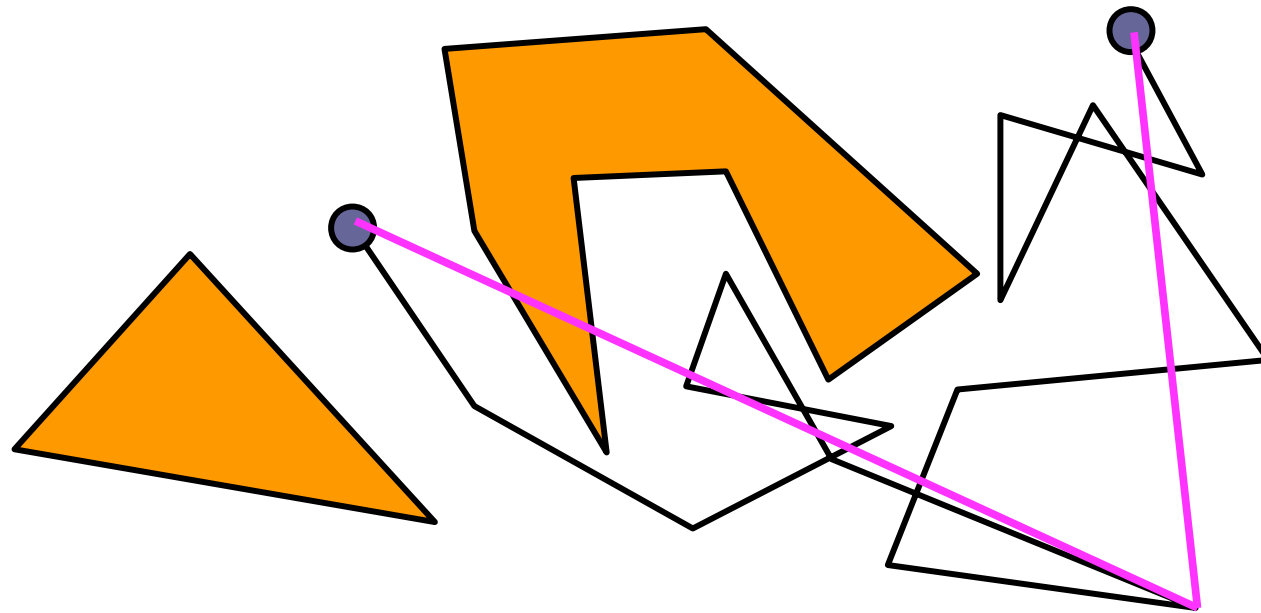
Smoothing the path



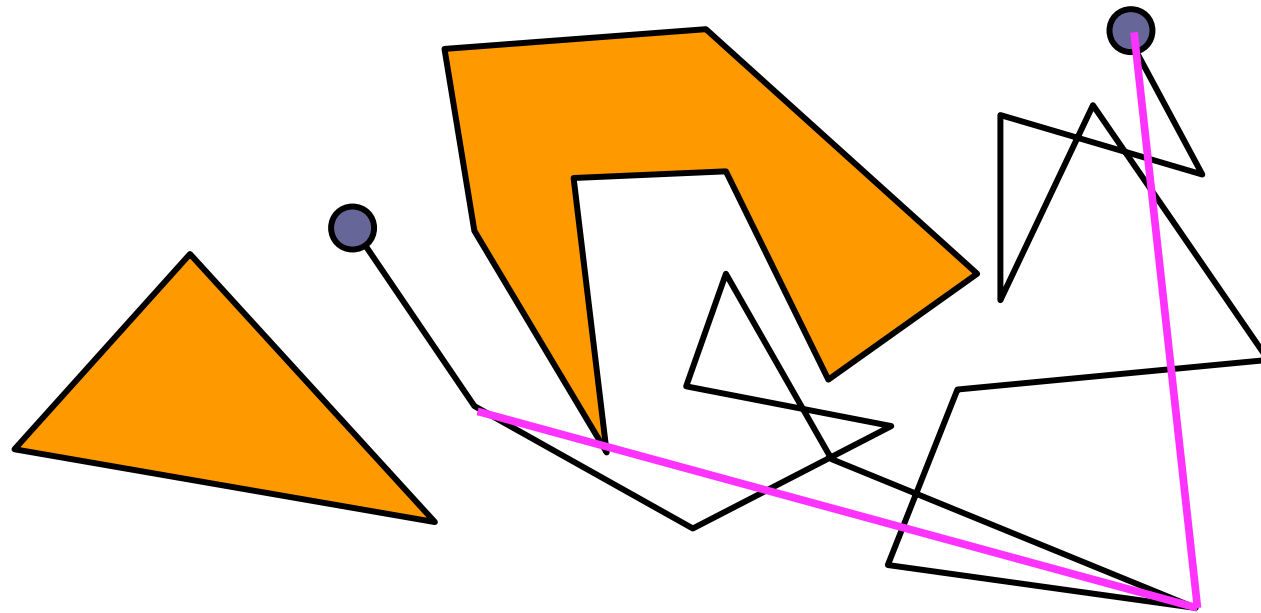
Smoothing the path



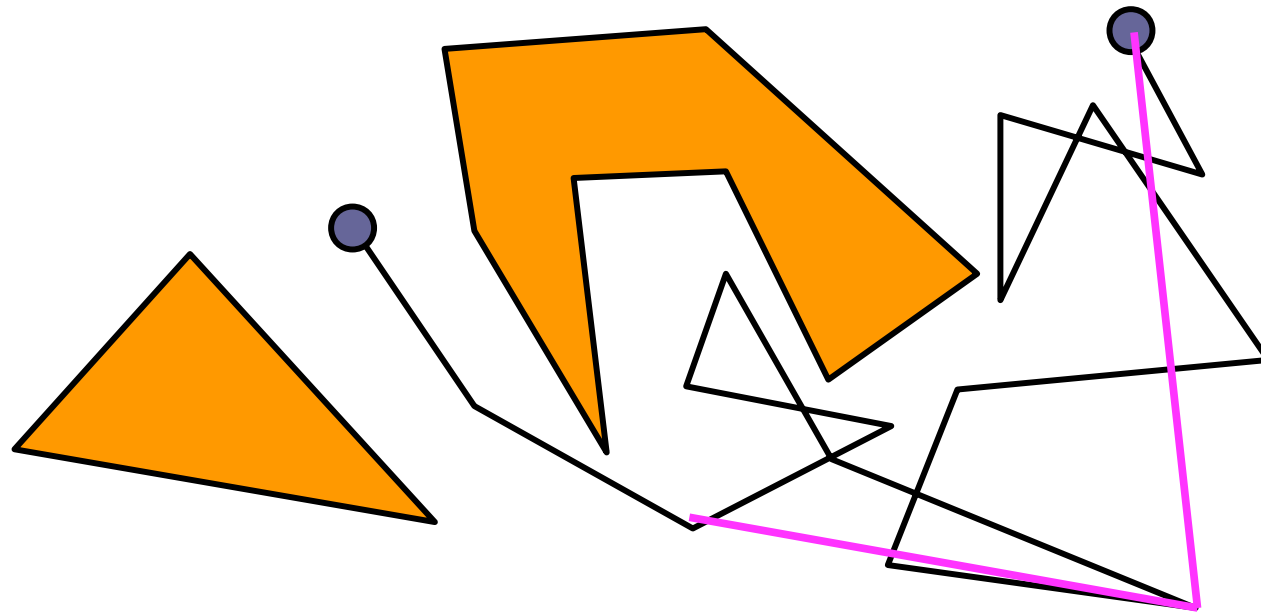
Smoothing the path



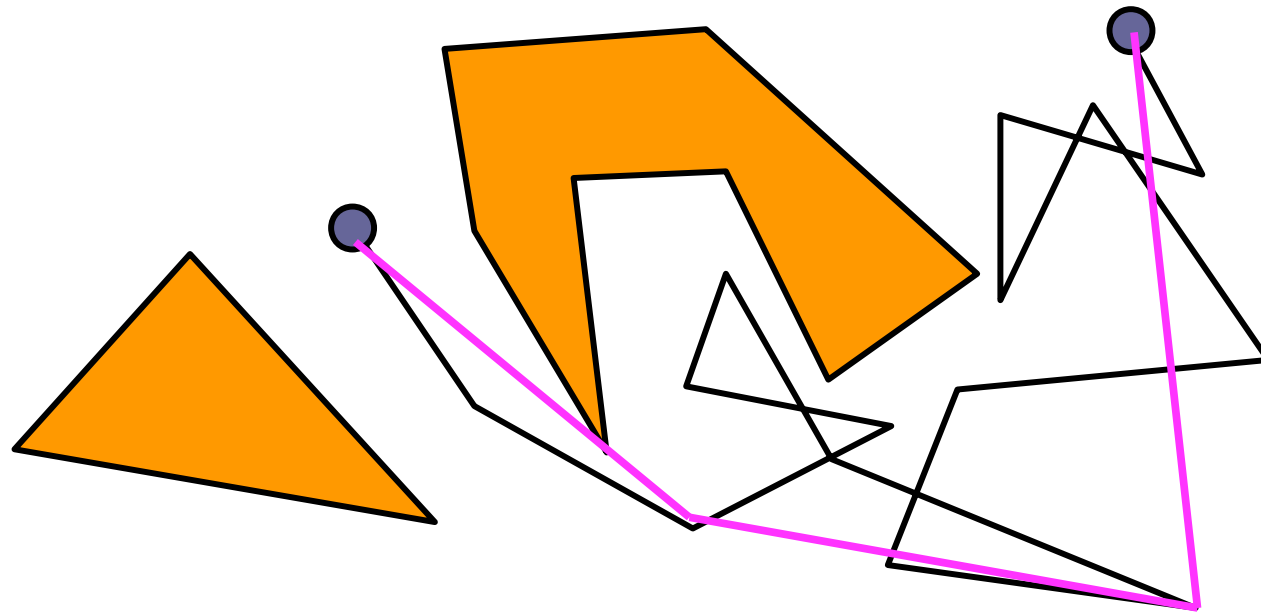
Smoothing the path



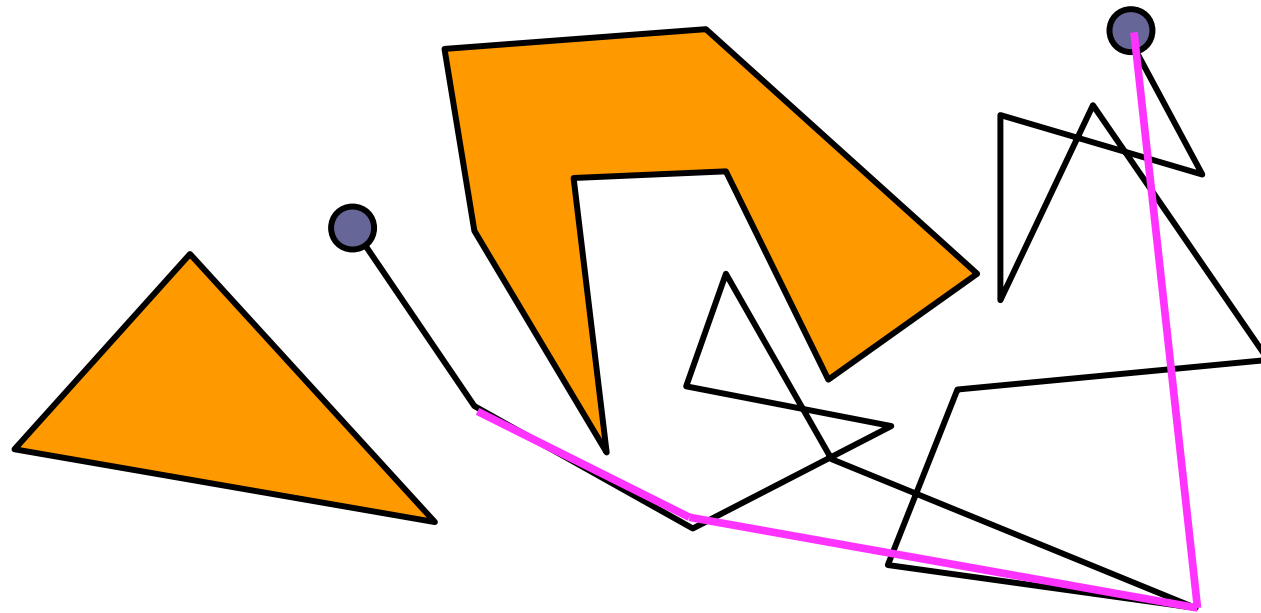
Smoothing the path



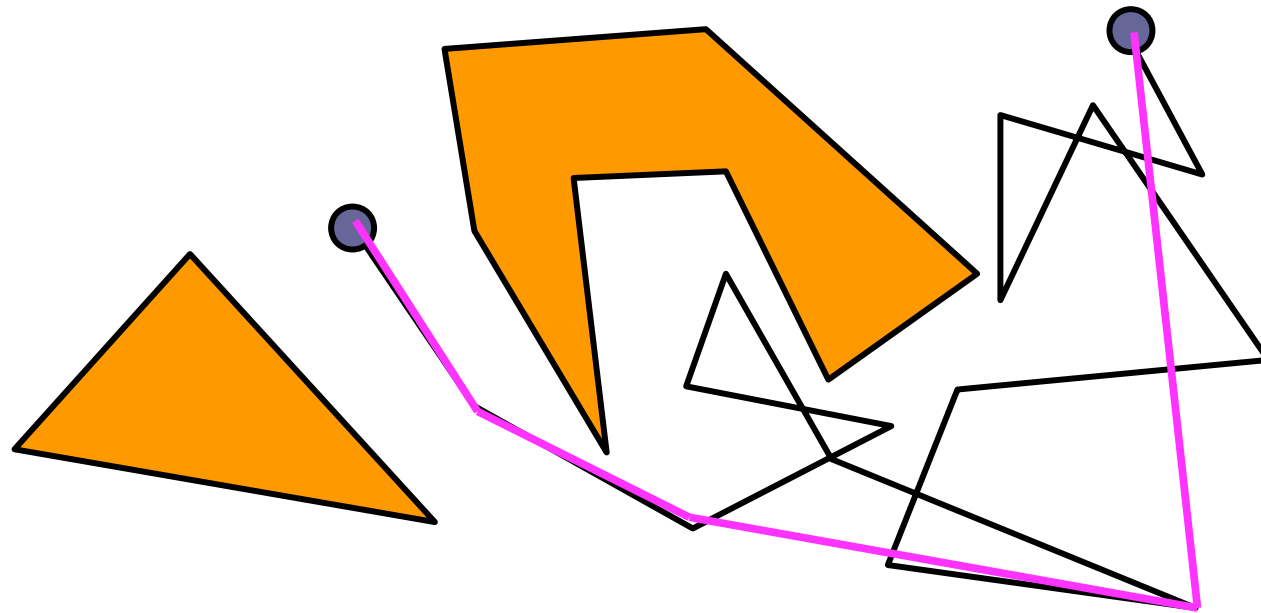
Smoothing the path



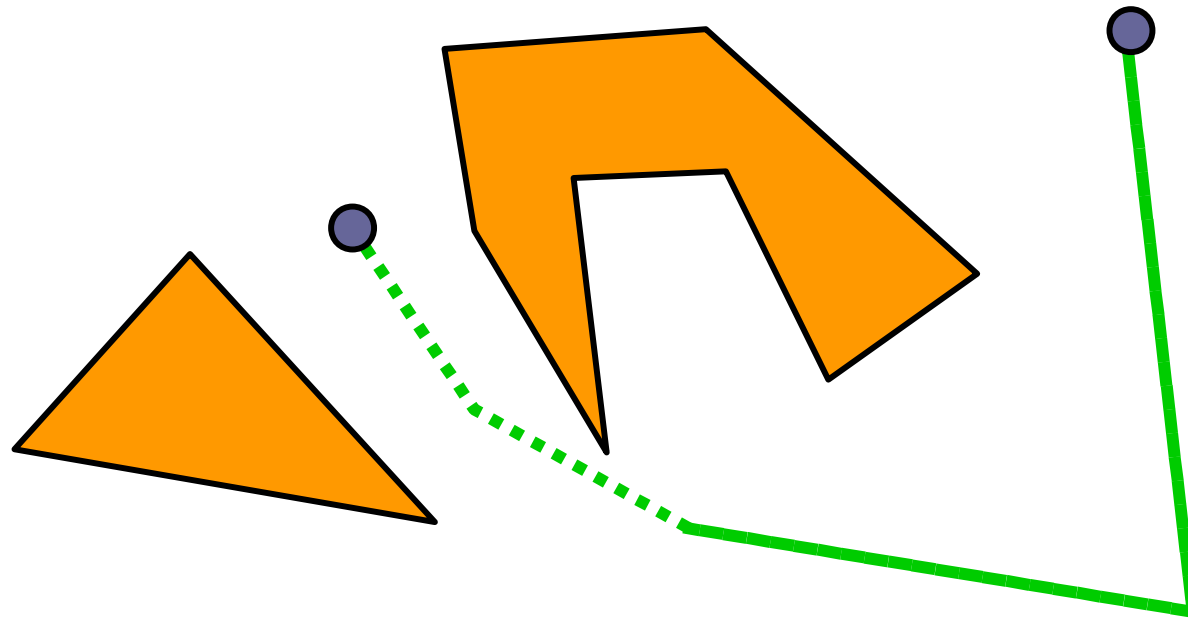
Smoothing the path



Smoothing the path



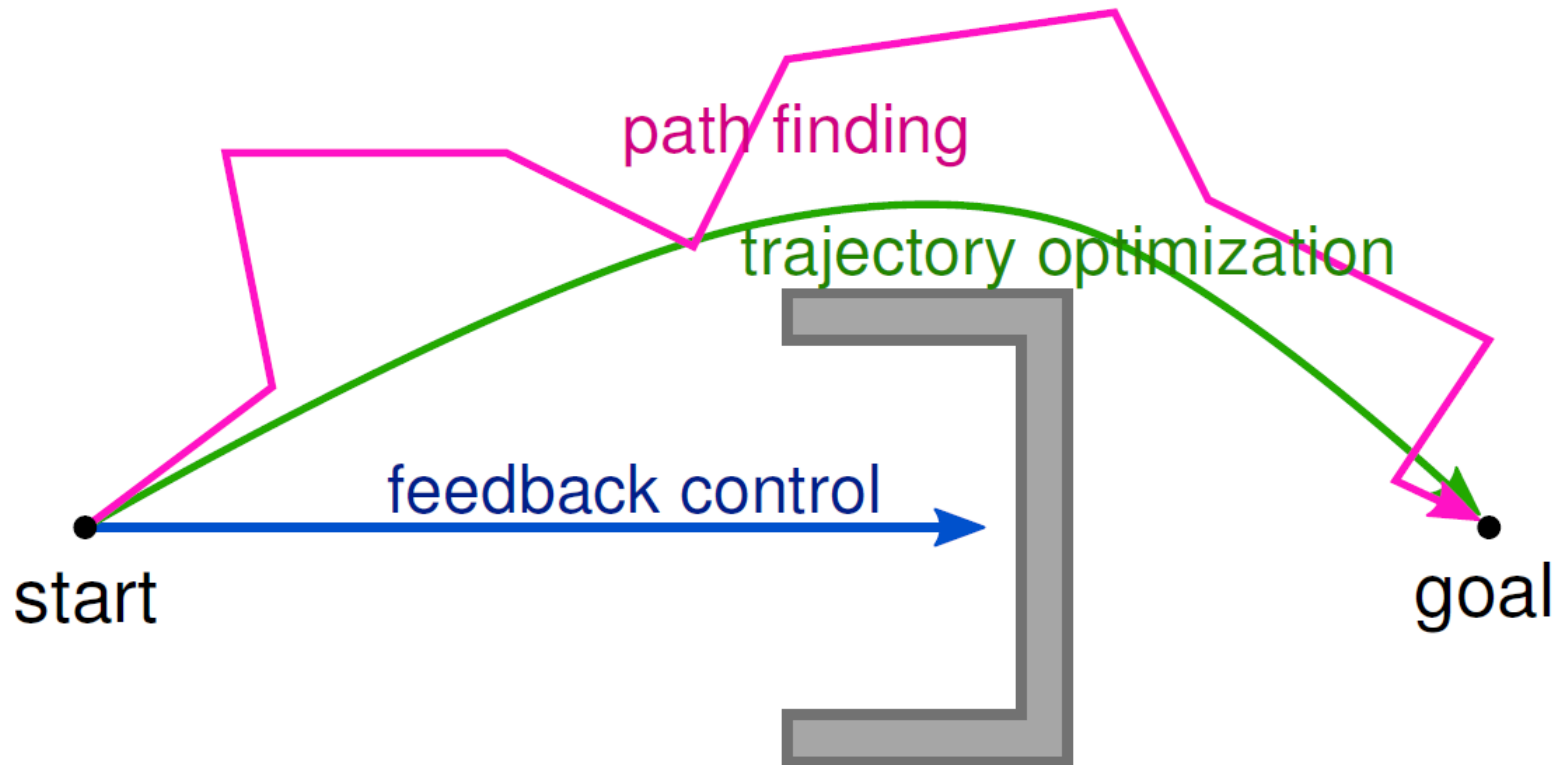
Smoothing the path



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

Outline

✓ More on RRTs

Overview of localization

Project

Mobile robot localization (Hallway example)

From
“Probabilistic Robotics”
(Ch. 7-8)
by
Sebastian Thrun,
Wolfram Burgard,
& Dieter Fox

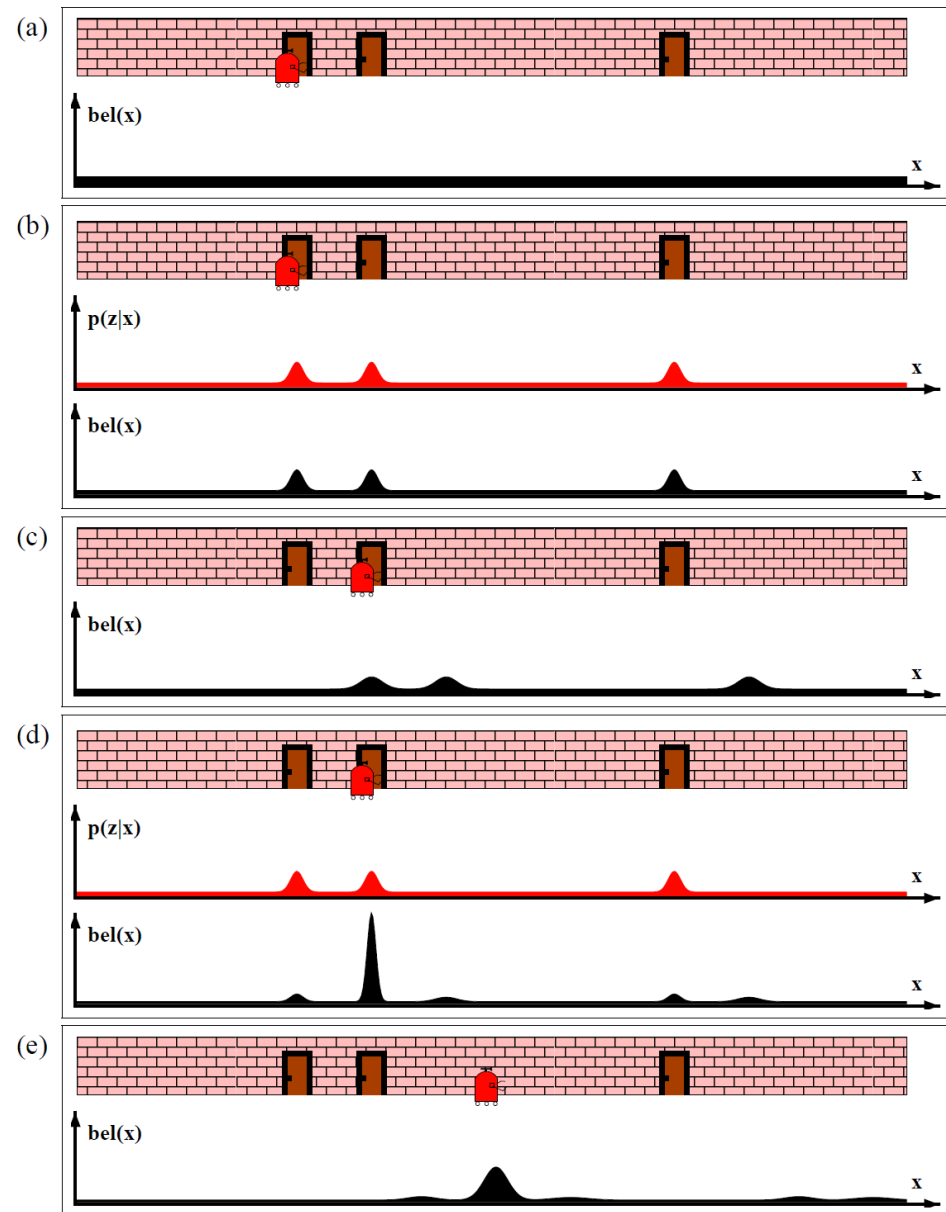
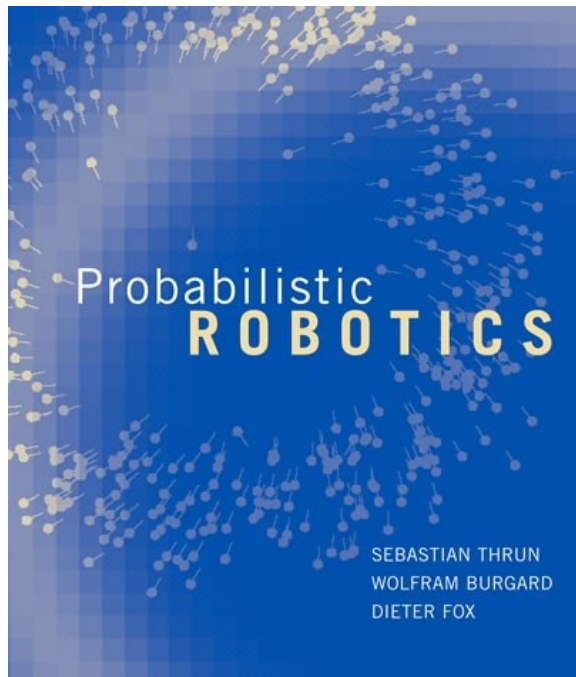


Figure 7.4 Illustration of the Markov localization algorithm. Each picture depicts the position of the robot in the hallway and its current belief $bel(x)$. (b) and (d) additionally depict the observation model $p(z_t | x_t)$, which describes the probability of observing a door at the different locations in the hallway.

Mobile robot localization (Hallway example)

From
“Probabilistic Robotics”
(Ch. 7-8)
by
Sebastian Thrun,
Wolfram Burgard,
& Dieter Fox

This is an important
problem!

- Ch. 2-4 (half of Part 1)
is about inference
- Ch. 7-8 (entire Part 2!)
is about localization
- There are 4 parts,
17 chapters in total

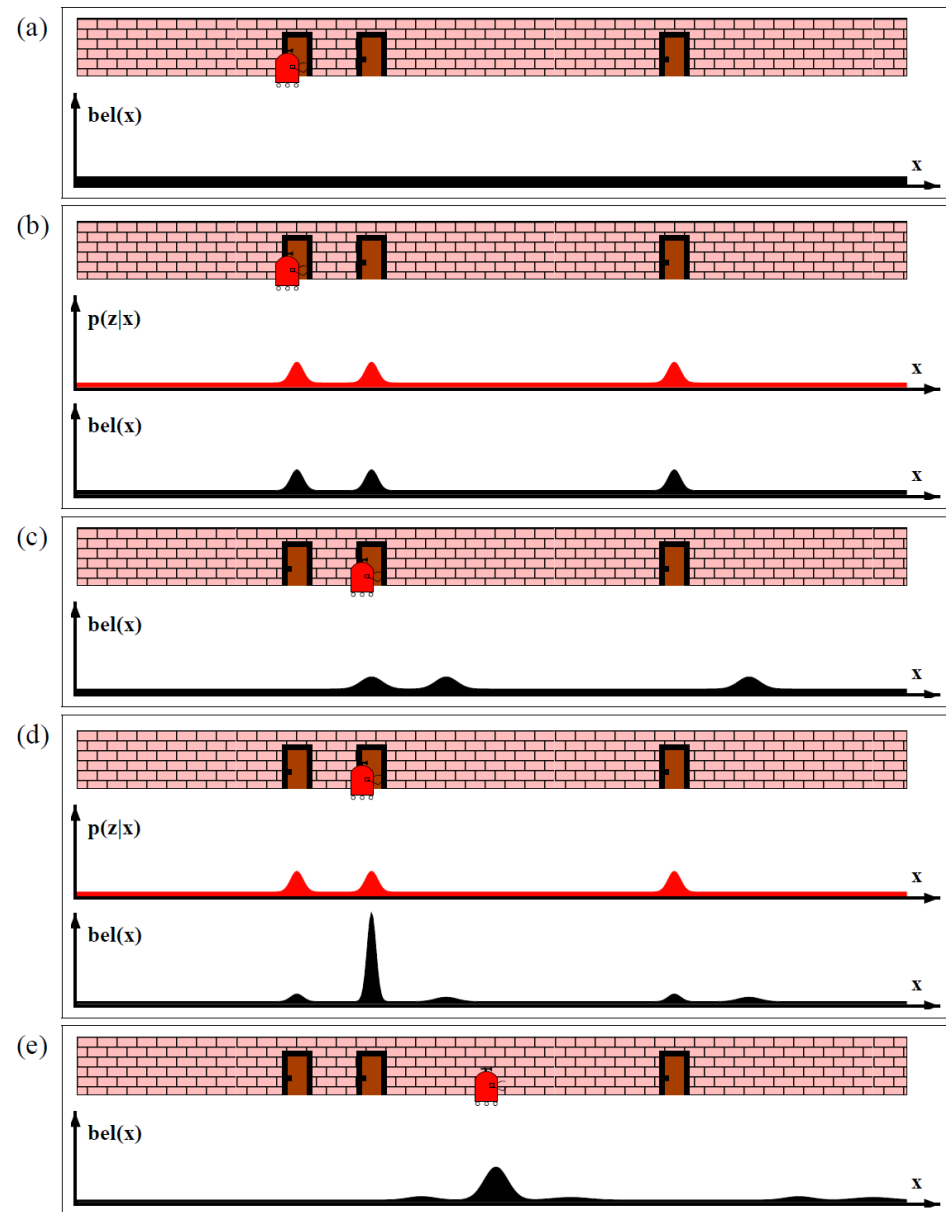


Figure 7.4 Illustration of the Markov localization algorithm. Each picture depicts the position of the robot in the hallway and its current belief $bel(x)$. (b) and (d) additionally depict the observation model $p(z_t | x_t)$, which describes the probability of observing a door at the different locations in the hallway.

Mobile robot localization (Hallway example)

From
“Probabilistic Robotics”
(Ch. 7-8)

by
Sebastian Thrun,
Wolfram Burgard,
& Dieter Fox

Continuous case;
Ideal outcome

We will study this
extensively with the
Kalman filter

– Get familiar with
Gaussian distributions!

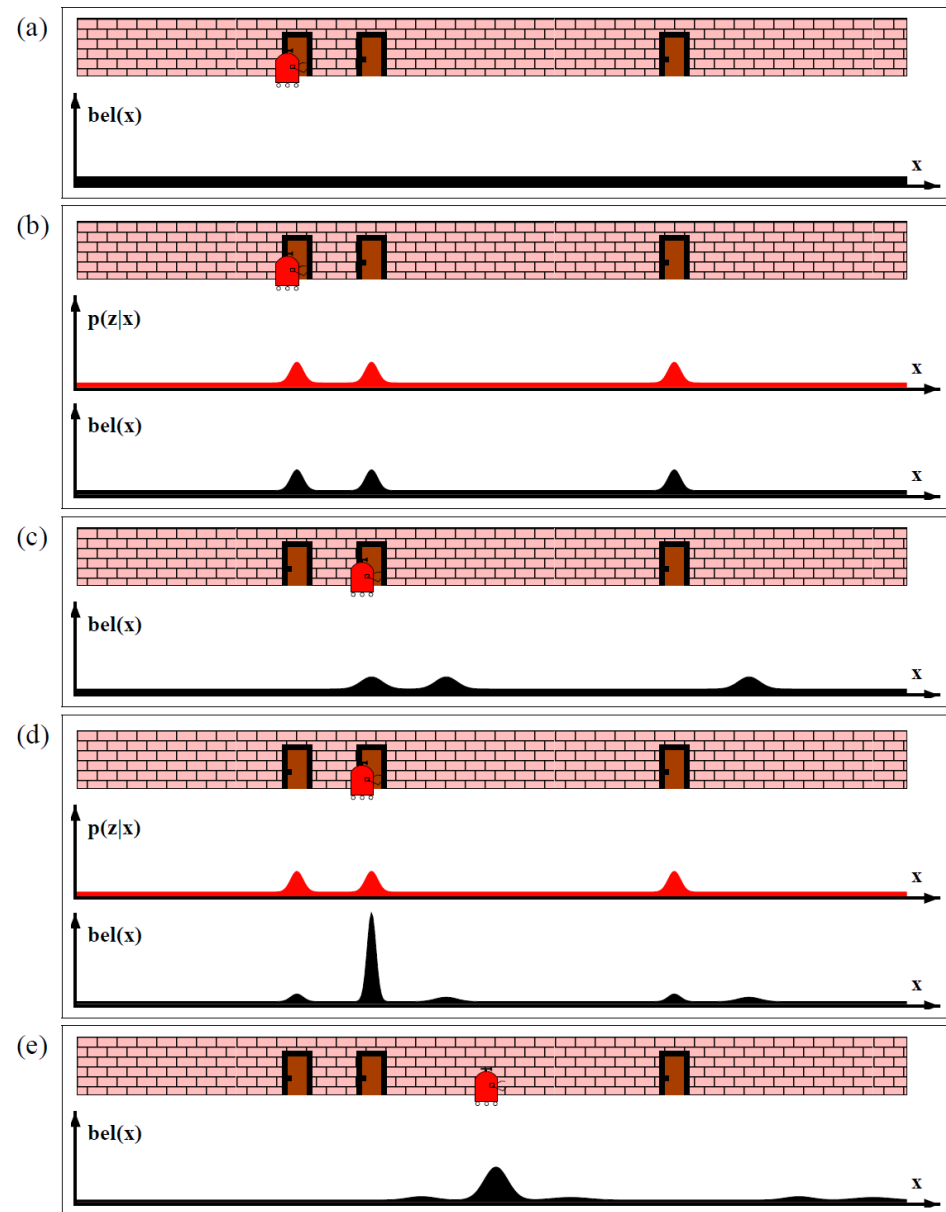


Figure 7.4 Illustration of the Markov localization algorithm. Each picture depicts the position of the robot in the hallway and its current belief $bel(x)$. (b) and (d) additionally depict the observation model $p(z_t | x_t)$, which describes the probability of observing a door at the different locations in the hallway.

Mobile robot localization (Hallway example)

From
“Probabilistic Robotics”
(Ch. 7-8)
by
Sebastian Thrun,
Wolfram Burgard,
& Dieter Fox

See appendix for
discrete case (1-D)

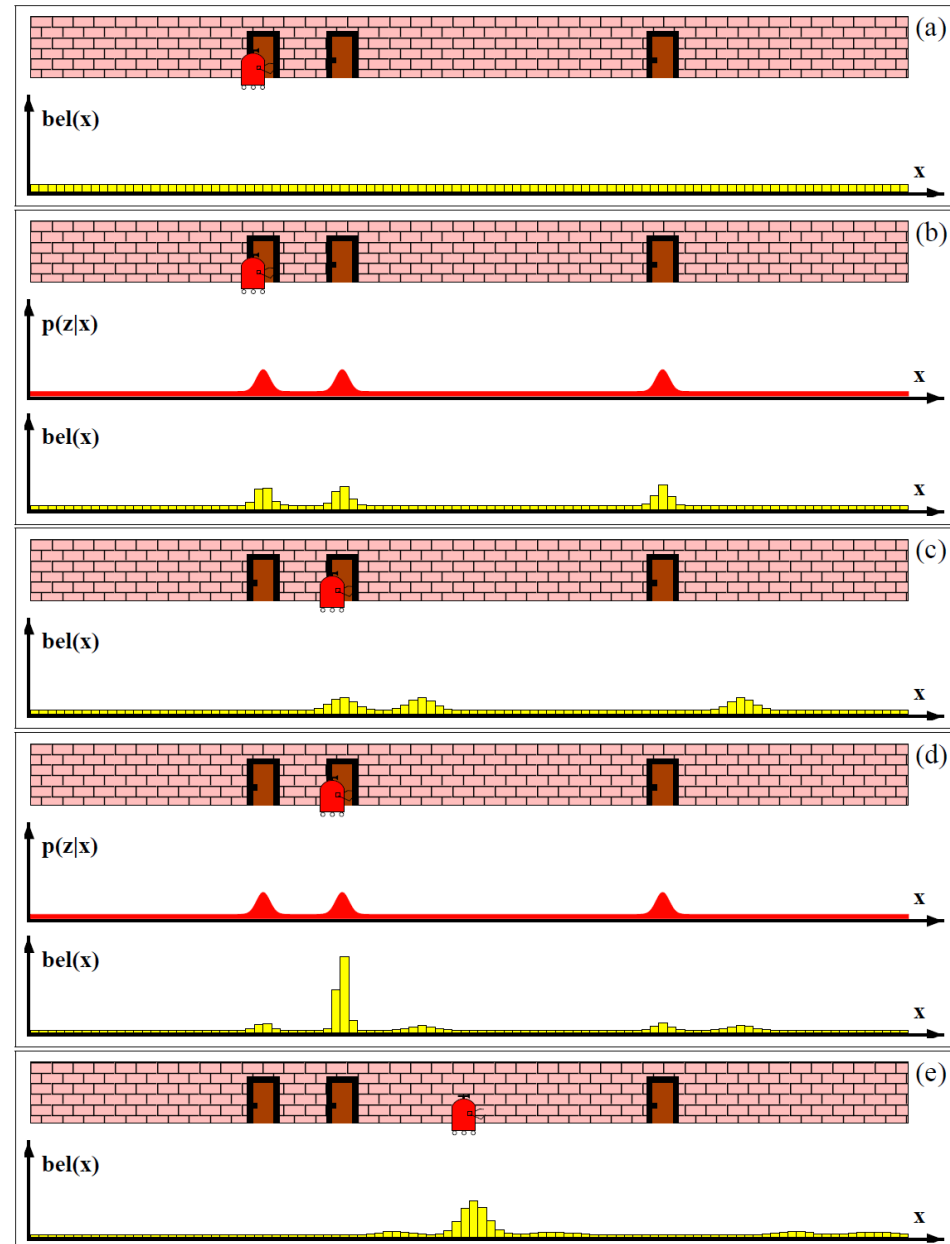


Figure 8.1 Grid localization using a fine-grained metric decomposition. Each picture depicts the position of the robot in the hallway along with its belief $bel(x_t)$, represented by a histogram over a grid.

Feedback

Piazza thread: 2/14 Lec 08 Feedback

Please post your answers to the following anonymously.

1. What did you like so far?
2. What was unclear?
3. How many hours did you spend on Ex1?
4. What was your favorite question on Ex1?
5. What was your least favorite question on Ex1?
6. Any additional feedback / comments?

Project

If you are not already in a team:

- Join the circle in the front of the classroom

If you already have a team:

- Work on your project thread (see @35)
 - Too many TBDs! Choose another (temp) name
- Figure out what hardware you need (see @33)
 - Including sensors and other add-ons
- Some past project topics also posted on @33

Hardware:

If you are an arm-only team:

- See a list of arm options on @33, or propose one
- Make a case for what hardware you need, and why

If you are a wheeled team:

- Duckiebot, Turtlebot3 (which one), or propose one

If you are a wheeled+arm team:

- Verify that the LoCoBot WX250 6-DOF is sufficient