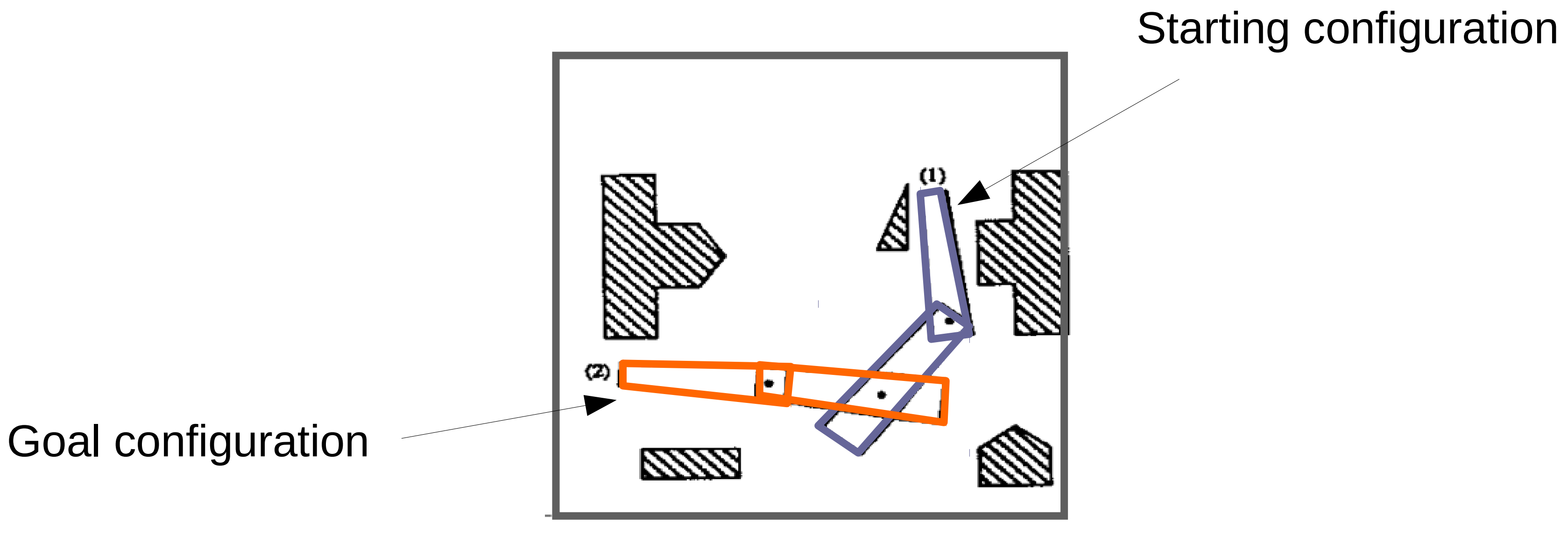
Sample Based Motion Planning

Robert Platt
Northeastern University

Problem we want to solve

Given:
– a point-robot (robot is a point in space)
– description of obstacle space and free space
– a start configuration and goal region

Find:
– a collision-free path from start to goal



Problem we want to solve

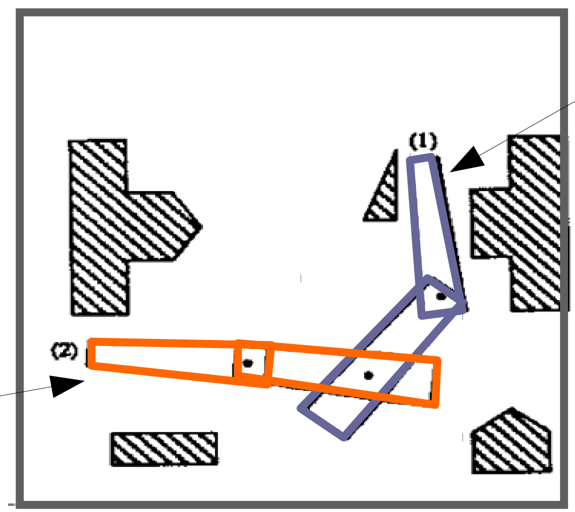
<u>Given:</u>

– configuration space $\mathcal{C}$
– free space $\mathcal{C}_{free}$
– start state $x_{init} \in \mathcal{C}_{free}$
– goal region $X_{goal} \subset \mathcal{C}_{free}$

<u>Find:</u>

– a collision-free path σ, such that $\sigma(0) = x_{init}$ and $\sigma(1) \in X_{goal}$

Starting configuration



Goal configuration

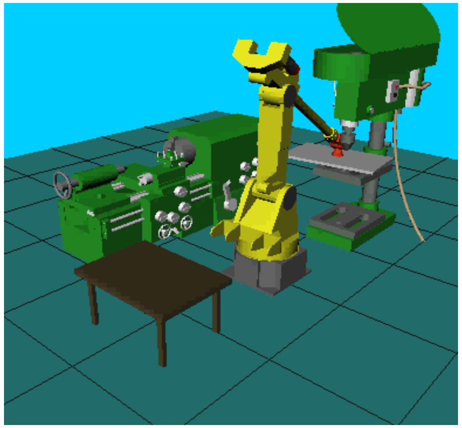
Problem we want to solve

Given:
– configuration space $\mathcal{C}$
– free space $\mathcal{C}_{free}$
– start state $x_{init} \in \mathcal{C}_{free}$
– goal region $X_{goal} \subset \mathcal{C}_{free}$

Find:
– a collision-free path σ, such that $\sigma(0) = x_{init}$ and $\sigma(1) \in X_{goal}$

Assumptions:
– the position of the robot can always be measured perfectly
– the motion of the robot can always be controlled perfectly

For example: think about a robot workcell in a factory...



Key challenge: high dimensions and complex geometry of free space

None of the methods we have looked at so far scale well to manipulator path planning

– e.g. a 6-DOF UR5 arm

Methods studied so far:

- – visibility graphs
- – Voronoi diagrams
- – cell decomposition
- – potential functions

Only work for small num of obstacles

Only work for small dimensional spaces

Not complete

Motion planning problem hardness

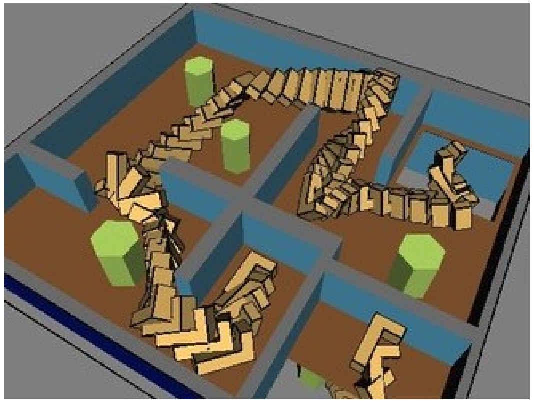
<u>The general path planning problem is PSPACE-hard</u>

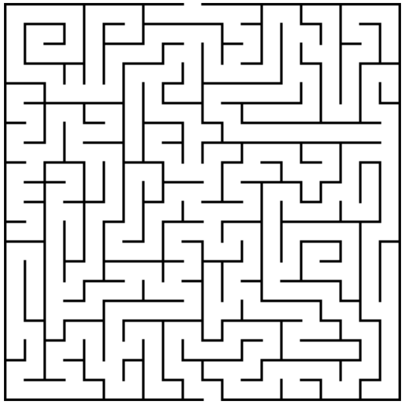
– pspace-hard in complexity of free space, e.g. measured by number of facets in total polyhedral obstacles

  – in the worst case, path planning requires solving an arbitrary difficult maze

  – complexity generally increases exponentially in the dimension of the configuration space

– the best we can do is find anytime algorithms that solve “simple” problems quickly while retaining completeness for arbitrary problems





Motion planning problem hardness

The general path planning problem is PSPACE-hard

- – pspace-hard in complexity of free space, e.g. measured by number of facets in total polyhedral obstacles
  - – in the worst case, path planning requires solving an arbitrary difficult maze
  - – complexity generally increases exponentially in the dimension of the configuration space
- – the best we can do is find anytime algorithms that solve “simple” problems quickly while retaining completeness for arbitrary problems

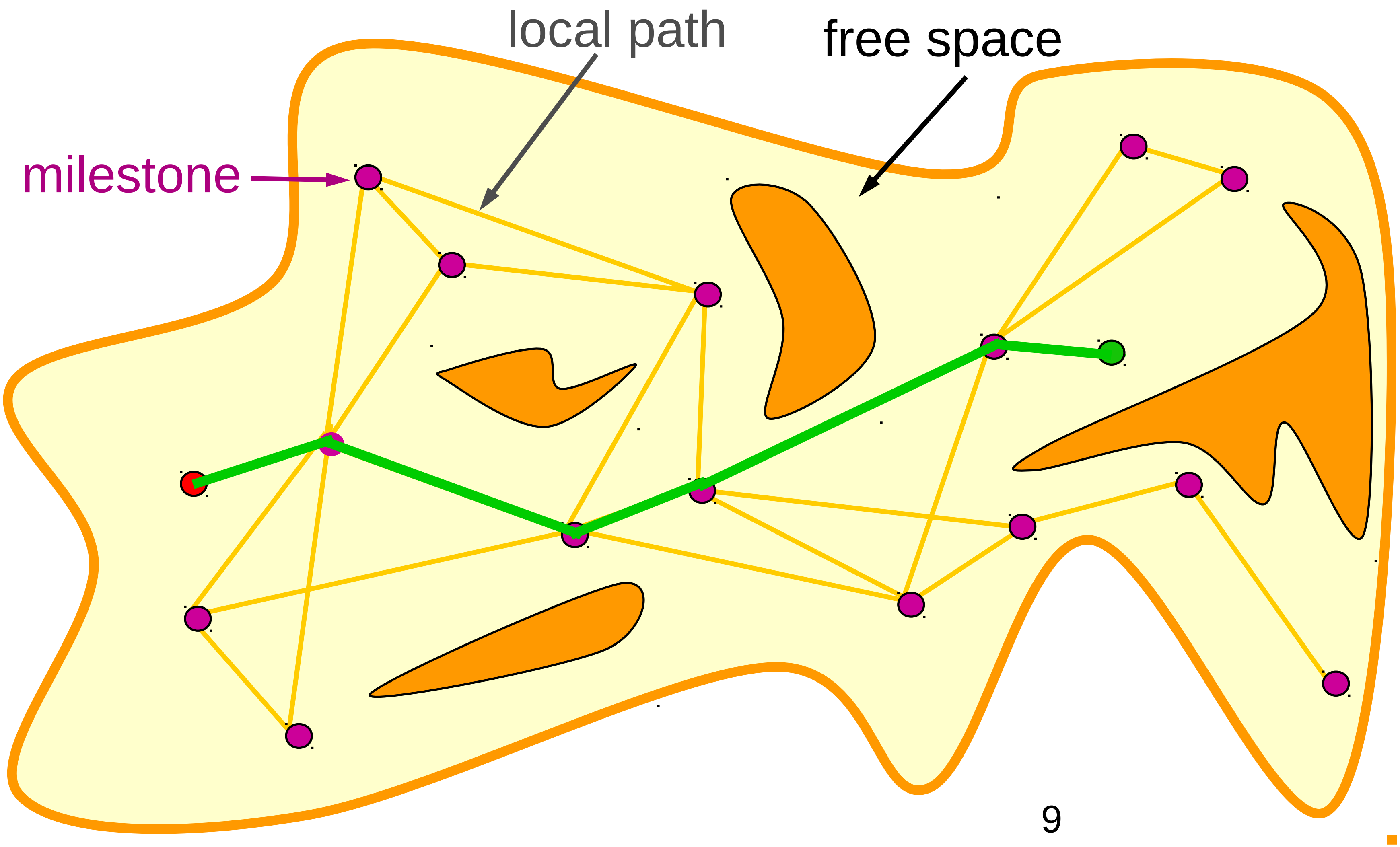
Another key practical challenge: most of the methods above require a preprocessing step where workspace obstacles are projected into the configuration space

- – this is just as hard as the motion planning problem itself

Simple PRM (sPRM)

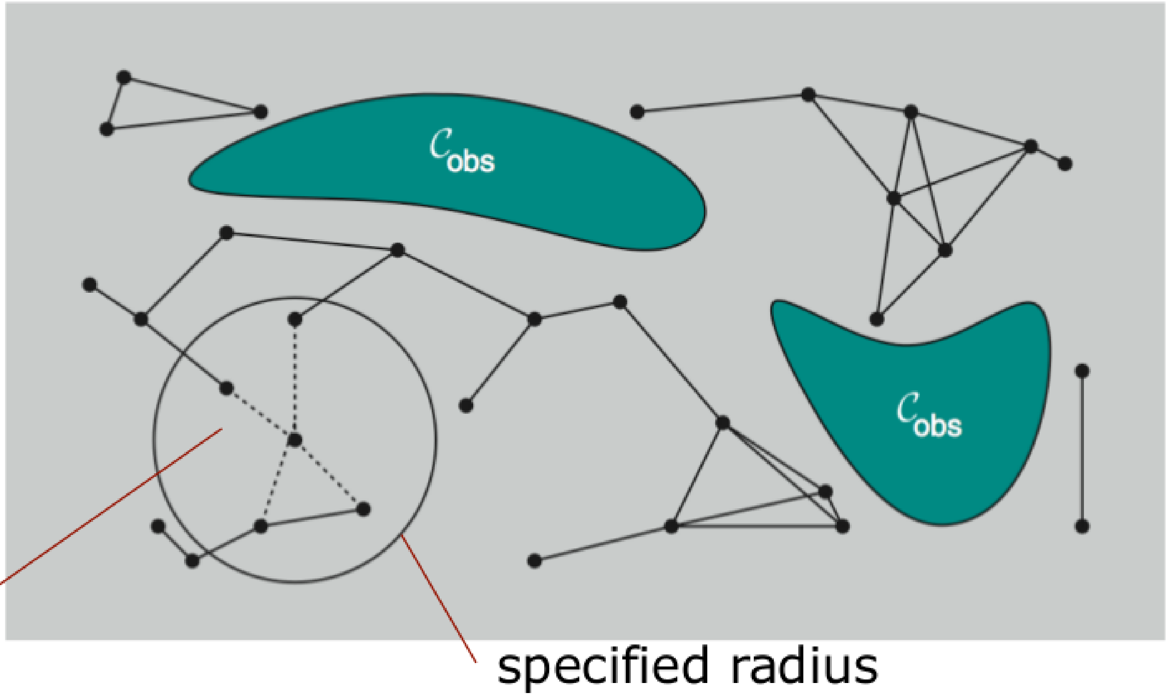
- **Idea:** Take random samples from C, declare them as vertices if in C_{free}, try to connect nearby vertices with local planner
- The **local planner** checks if line-of-sight is collision-free (powerful or simple methods)
- Options for *nearby:* **k-nearest neighbors** or all neighbors within **specified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

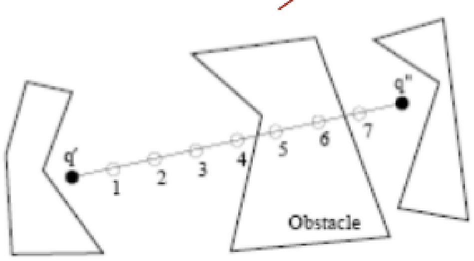
Simple PRM (sPRM)



Simple PRM (sPRM)

- Example





Example local planner

What means "nearby" on a manifold?
Defining a good metric on C is crucial

Simple PRM (sPRM)

<u>SampleFree</u>: sample a state from $\mathcal{C}_{free}$

Algorithm 2: sPRM

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\dots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

<u>Near</u>: return the set of vertices in *G* within radius *r* of *v*

<u>CollisionFree</u>: check whether a line segment between v and u is completely within $\mathcal{C}_{free}$

Question

Algorithm 2: sPRM

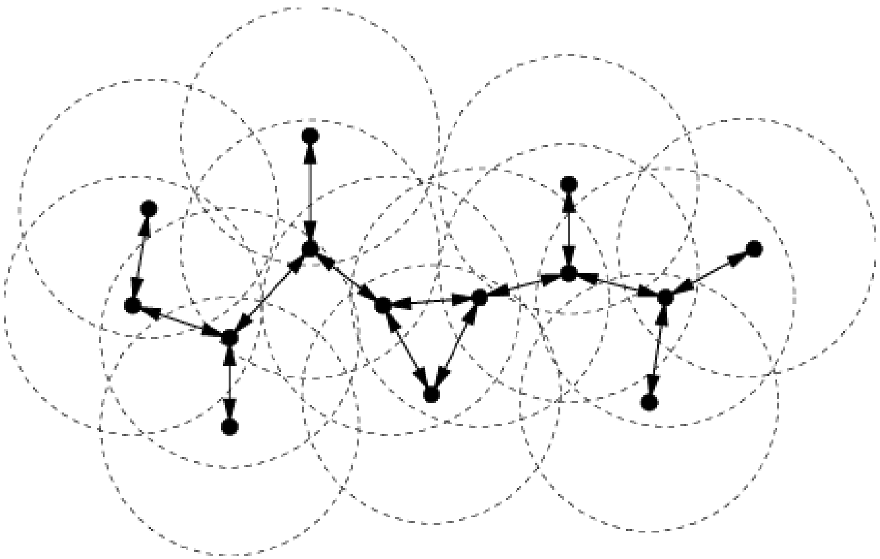
$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

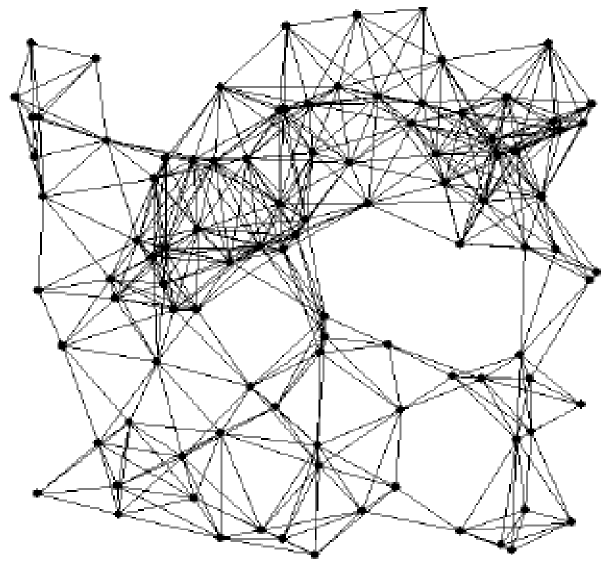
For a fully connected graph created using sPRM parameterized by radius r, consider a vertex that is the nearest neighbor of another. What is the maximum distance between these two vertices?

sPRM

What kind of graph does sPRM find?

Definition 5 (Random r-disc graph) *Let $r \in \mathbb{R}_{>0}$, and $n, d \in \mathbb{N}$. A random r-disc graph $G^{\text{disc}}(n, r)$ in d dimensions is a graph whose n vertices, $\{X_1, X_2, \ldots, X_n\}$, are independent, uniformly distributed random variables in $(0,1)^d$, and such that (X_i, X_j), $i, j \in \{1, \ldots, n\}$, $i \neq j$, is an edge if and only if $\|X_i - X_j\| < r$.*





Question

What kind of graph does sPRM find?

Definition 5 (Random r-disc graph) *Let $r \in \mathbb{R}_{>0}$, and $n, d \in \mathbb{N}$. A random r-disc graph $G^{\mathrm{disc}}(n, r)$ in d dimensions is a graph whose n vertices, $\{X_1, X_2, \ldots, X_n\}$, are independent, uniformly distributed random variables in $(0,1)^d$, and such that (X_i, X_j), $i, j \in \{1, \ldots, n\}$, $i \neq j$, is an edge if and only if $\|X_i - X_j\| < r$.*

Does the graph become connected as *n* becomes large?

sPRM

What kind of graph does sPRM find?

Definition 5 (Random r-disc graph) *Let $r \in \mathbb{R}_{>0}$, and $n, d \in \mathbb{N}$. A random r-disc graph $G^{\text{disc}}(n, r)$ in d dimensions is a graph whose n vertices, $\{X_1, X_2, \ldots, X_n\}$, are independent, uniformly distributed random variables in $(0, 1)^d$, and such that (X_i, X_j), $i, j \in \{1, \ldots, n\}$, $i \neq j$, is an edge if and only if $\|X_i - X_j\| < r$.*

Does the graph become connected as *n* becomes large?

Theorem 7 (Connectivity of random r-disc graphs (Penrose, 2003)) *Let $G^{\text{disc}}(n, r)$ be a random r-disc graph in d dimensions. Then,*

$$\lim_{n\to\infty} \mathbb{P}\left(\{G^{\text{disc}}(n, r) \text{ is connected }\}\right) = \begin{cases} 1, & \text{if } \zeta_d r^d > \log(n)/n, \\ 0, & \text{if } \zeta_d r^d < \log(n)/n, \end{cases}$$

where ζ_d is the volume of the unit ball in d dimensions.

https://www.youtube.com/watch?v=twjnAE3SjJw

Question

<u>What kind of graph does sPRM find?</u>

Definition 5 (Random r-disc graph) *Let $r \in \mathbb{R}_{>0}$, and $n, d \in \mathbb{N}$. A random r-disc graph $G^{\mathrm{disc}}(n, r)$ in d dimensions is a graph whose n vertices, $\{X_1, X_2, \ldots, X_n\}$, are independent, uniformly distributed random variables in $(0,1)^d$, and such that (X_i, X_j), $i, j \in \{1, \ldots, n\}$, $i \neq j$, is an edge if and only if $\|X_i - X_j\| < r$.*

<u>Does the graph become connected as *n* becomes large?</u>

Theorem 7 (Connectivity of random r-disc graphs (Penrose, 2003)) *Let $G^{\mathrm{disc}}(n, r)$ be a random r-disc graph in d dimensions. Then,*

$$\lim_{n\to\infty} \mathbb{P}\left(\{G^{\mathrm{disc}}(n,r) \text{ is connected }\}\right) = \begin{cases} 1, & \text{if } \zeta_d r^d > \log(n)/n, \\ 0, & \text{if } \zeta_d r^d < \log(n)/n, \end{cases}$$

where ζ_d is the volume of the unit ball in d dimensions.

Volume of unit ball in d dim

<u>For sPRM, express the number of edges in the graph as a function of n as n goes to infinity:</u>

$$\lim_{n\to\infty} |\mathcal{E}| = ?$$

Probabilistic completeness of PRM

SPRM is *not* complete: not guaranteed to find a solution for any finite value of *n*

However, it is *probabilistically complete* in the following sense:

Definition 14 (Probabilistic completeness). *An algorithm ALG is probabilistically complete, if, for any robustly feasible path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$,

$$\liminf_{n\to\infty} \mathbb{P}(\{\exists x_{\text{goal}} \in V_n^{\text{ALG}} \cap \mathcal{X}_{\text{goal}} \text{ such that } x_{\text{init}} \text{ is connected to } x_{\text{goal}} \text{ in } G_n^{\text{ALG}}\}) = 1.$$

Finds a path with probability 1 as the number of vertices increases as long as such a path if robustly feasible

Probabilistic completeness of PRM

S

H

Infinite monkey theorem:

A monkey typing keys randomly on a keyboard will produce any given text (the works of William Shakespeare) *with probability one.*

increases as long as such a path if robustly feasible

Probabilistic completeness of PRM

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*

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

Probability of *not* finding a solution to a robustly feasible problem decreases exponentially with the number of vertices

Optimality

$c : \Sigma \to \mathbb{R}_{\geq 0}$

Recall: $\sigma : [0, 1] \to \mathcal{C}$

Cost: $c(\sigma) =$ cost of path

Definition 24 (Asymptotic optimality). *An algorithm ALG is asymptotically optimal if, for any path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$ *and cost function* $c : \Sigma \to \mathbb{R}_{\geq 0}$ *that admit a robustly optimal solution with finite cost* c^**,*

$$\mathbb{P}\left(\left\{\limsup_{n\to\infty} Y_n^{\text{ALG}} = c^*\right\}\right) = 1.$$

Cost of min cost path found by the algorithm

Optimal cost

Optimality of sPRM

Theorem 30 (Asymptotic optimality of sPRM)**.** *The sPRM algorithm is asymptotically optimal.*

Theorem 31 (Non-optimality of k-nearest sPRM)**.** *The k-nearest sPRM algorithm is not asymptotically optimal, for any constant* $k \in \mathbb{N}$.

K-nearest sPRM is variation where connect vertices
w/ k-NN instead of neighbors within radius

PRM*

<u>Problem w/ sPRM</u>: number of edges grows nearly quadratically with the number of edges

Algorithm 2: sPRM

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then** $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

Num NEAR vertices grows linearly w/ *n*

<u>Idea</u>: reduce the connection radius as the number of vertices grows

– BUT: if you do it too quickly, the graph becomes asymptotically disconnected

PRM*

Idea: set r to exactly $r = \left(\frac{\log(n)}{\zeta_d n}\right)^{\frac{1}{d}}$

Volume of unit ball in d dim

Recall:

Theorem 7 (Connectivity of random r-disc graphs (Penrose, 2003)) *Let $G^{\mathrm{disc}}(n, r)$ be a random r-disc graph in d dimensions. Then,*

$$\lim_{n\to\infty} \mathbb{P}\left(\{G^{\mathrm{disc}}(n,r) \text{ is connected }\}\right) = \begin{cases} 1, & \text{if } \zeta_d r^d > \log(n)/n, \\ 0, & \text{if } \zeta_d r^d < \log(n)/n, \end{cases}$$

where ζ_d is the volume of the unit ball in d dimensions.

Technically, we need to adjust *r* for the volume of obstacles:

$$r = \gamma_{PRM} \left(\frac{\log(n)}{n}\right)^{\frac{1}{d}}$$

PRM*

Algorithm 2: sPRM.

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then**
      $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

Algorithm 4: PRM*.

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;
foreach $v \in V$ **do**
  $U \leftarrow \texttt{Near}(G = (V, E), v, \gamma_{\text{PRM}}(\log(n)/n)^{1/d}) \setminus \{v\}$;
  foreach $u \in U$ **do**
    if $\texttt{CollisionFree}(v, u)$ **then**
      $E \leftarrow E \cup \{(v, u), (u, v)\}$
return $G = (V, E)$;

PRM* adds a constant number of edges on each step, but remains asyptotically optimal

Theorem 34 (Asymptotic optimality of PRM*). *If* $\gamma_{\text{PRM}} > 2(1 + 1/d)^{1/d} \left(\frac{\mu(X_{\text{free}})}{\zeta_d}\right)^{1/d}$, *then the PRM* algorithm is asymptotically optimal.*

PRM sampling strategies

Algorithm 2: sPRM.

$V \leftarrow \{x_{\text{init}}\} \cup \{\texttt{SampleFree}_i\}_{i=1,\ldots,n}$; $E \leftarrow \emptyset$;

foreach $v \in V$ **do**

  $U \leftarrow \texttt{Near}(G = (V, E), v, r) \setminus \{v\}$;

  foreach $u \in U$ **do**

    if $\texttt{CollisionFree}(v, u)$ **then**

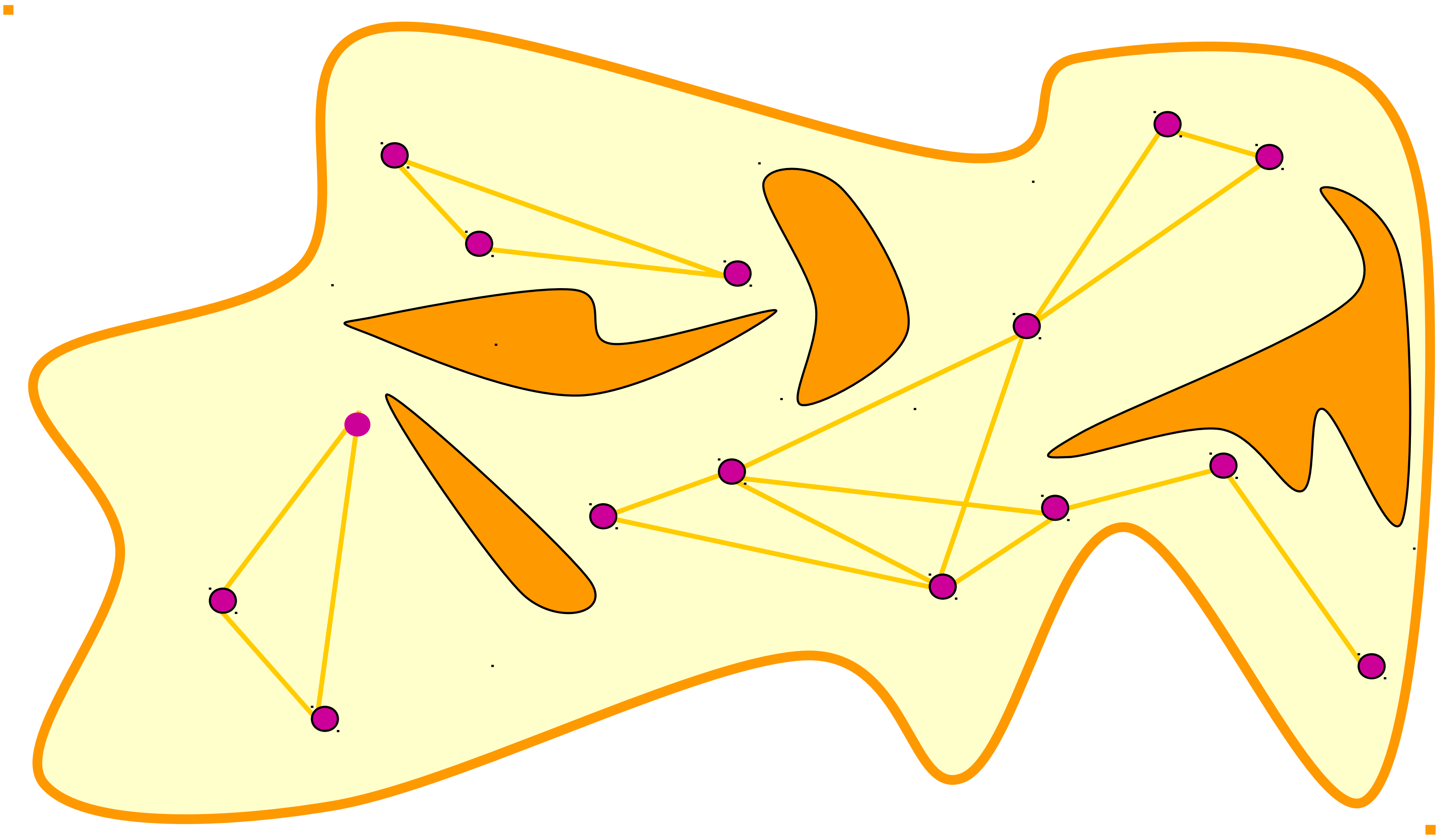
      $E \leftarrow E \cup \{(v, u), (u, v)\}$

return $G = (V, E)$;

Can we do better with a smarter sampling strategy?

PRM sampling strategies

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



PRM sampling strategies

Algorithm 1: PRM (preprocessing phase).

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$;

for $i = 0, \ldots, n$ **do**

    $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;

    $U \leftarrow \texttt{Near}(G = (V, E), x_{\text{rand}}, r)$;

    $V \leftarrow V \cup \{x_{\text{rand}}\}$;

    foreach $u \in U$, *in order of increasing* $\|u - x_{\text{rand}}\|$, **do**

        if x_{rand} *and* u *are not in the same connected component of* $G = (V, E)$ **then**

            if $\texttt{CollisionFree}(x_{\text{rand}}, u)$ **then**

                $E \leftarrow E \cup \{(x_{\text{rand}}, u), (u, x_{\text{rand}})\}$;

return $G = (V, E)$;

Let’s think about the “online” version of algorithm

PRM sampling strategies

Algorithm 1: PRM (preprocessing phase).

$V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
for $i = 0, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $U \leftarrow \texttt{Near}(G = (V, E), x_{\text{rand}}, r)$;
  $V \leftarrow V \cup \{x_{\text{rand}}\}$;
  foreach $u \in U$, *in order of increasing* $\|u - x_{\text{rand}}\|$, **do**
    if x_{rand} *and* u *are not in the same connected component of* $G = (V, E)$ **then**
      if $\texttt{CollisionFree}(x_{\text{rand}}, u)$ **then**
        $E \leftarrow E \cup \{(x_{\text{rand}}, u), (u, x_{\text{rand}})\}$;
return $G = (V, E)$;

Let’s think about the “online” version of algorithm

Resampling

Idea: expand vertices that are close to obstacles

1. Sample a vertex to expand
   – select vertices for which many link failures have occurred

Expand nodes w/ prob:

$$w(q) = \frac{rate(q)}{\sum_{q \in V} rate(q)}$$

Where:

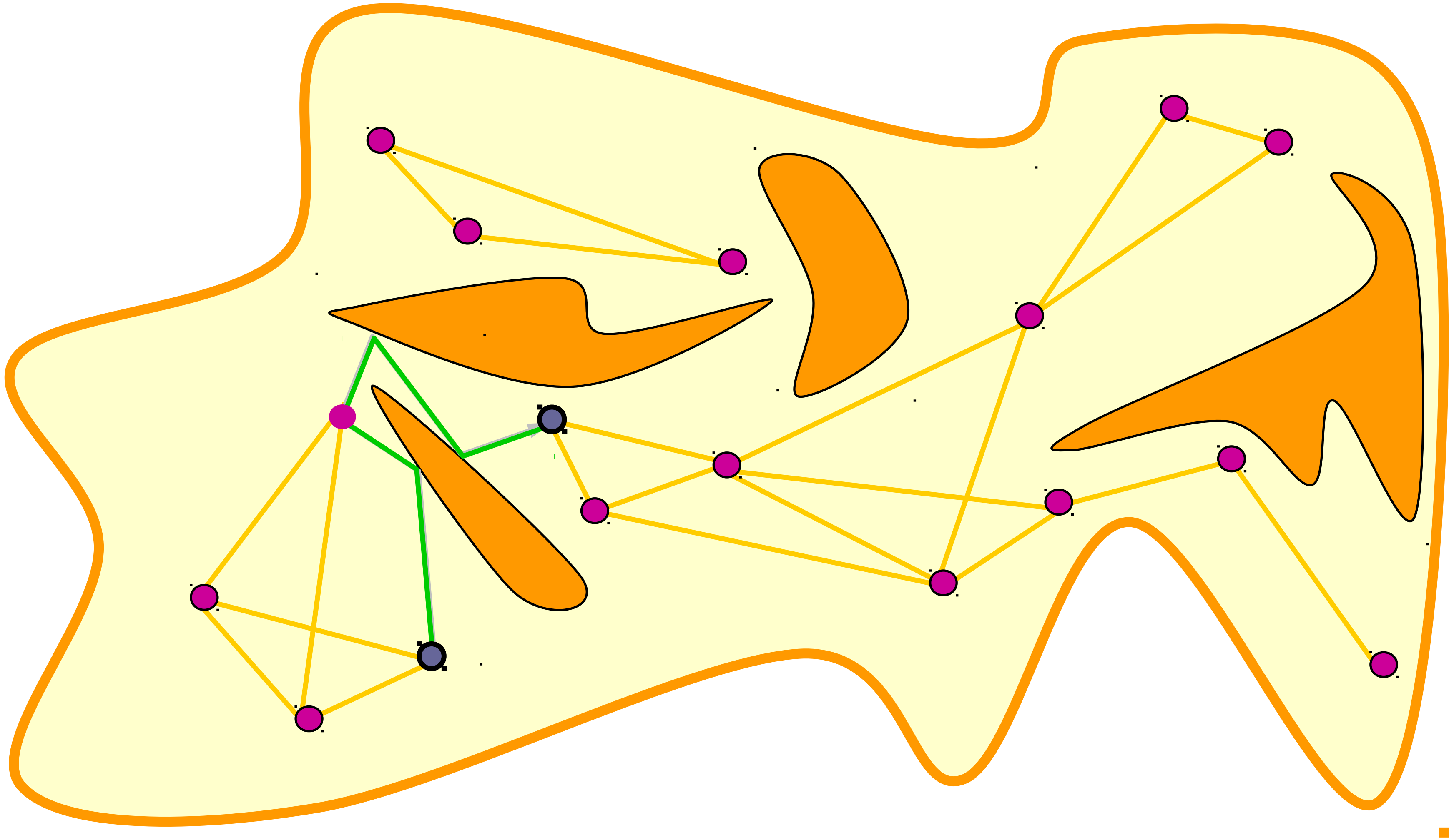
$$rate(q) = \frac{\text{num failed link attempts}}{\text{num link attempts}}$$

2. Pick a random motion direction in c-space and move in this direction until an obstacle is hit.

3. When a collision occurs, choose a new random direction and proceed for some distance.

4. Add the resulting nodes and edges to the tree. Re-run tree connection step.

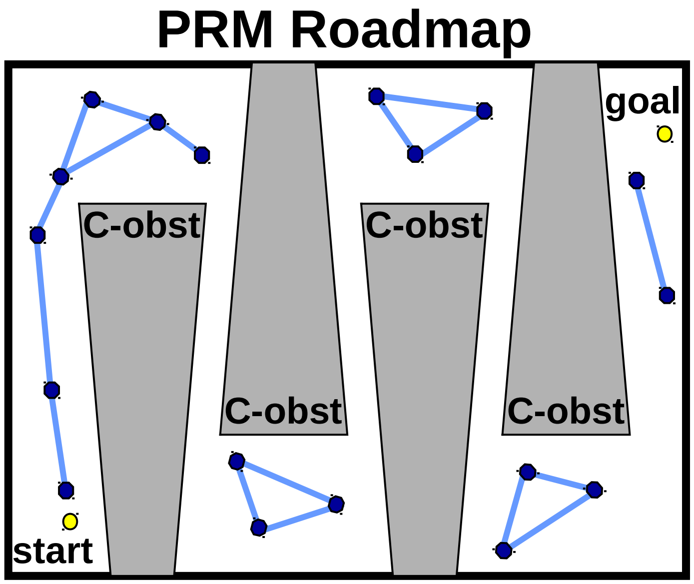
Resampling (expansion)



Gaussian sampler

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

<u>Problem</u>: uniform sampling is not a great way to find paths through narrow passageways.



Gaussian sampler

<u>Gaussian sampler:</u>

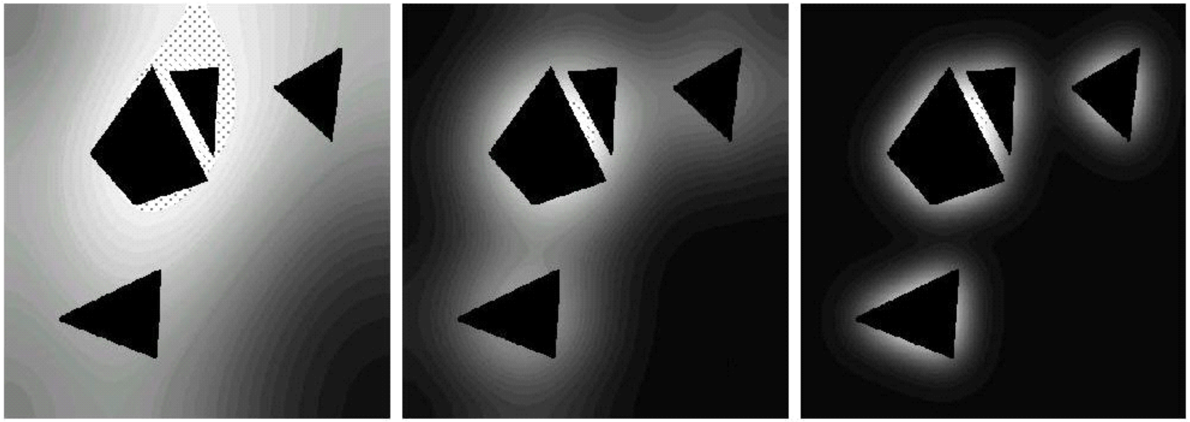
1. Sample points uniformly at random (as before)

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

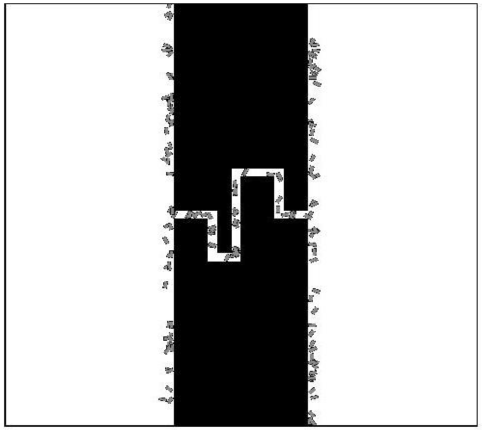
3. Discard both samples if both samples are either free or in collision

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

Gaussian sampler



Probability of sampling a point under the Gaussian sampler as a function of distance from a c-space obstacle



Example of samples drawn from Gaussian sampler

Lazy PRM

Single query problem: you are only interested in connecting start and goal configurations. Don’t care about cull connectivity of the map.

Lazy PRM idea: only check edges that could potentially be on the shortest path through the graph.

Lazy PRM Precomputation: roadmap construction

- Nodes
  - Randomly chosen configurations, which may or may **not** be collision-free
  - No call to **CLEAR**
- Edges
  - an edge between two nodes if the corresponding configurations are close according to a suitable metric
  - no call to **LINK**

Lazy PRM

Query processing:

Using UCS or A*

1. Find a shortest path in the roadmap
2. Check whether the nodes and edges in the path are free.
3. If yes, then done. Otherwise, remove the nodes or edges in violation. Go to (1).

We either find a collision-free path, or exhaust all paths in the roadmap and declare failure.

Rapidly Exploring Random Trees (RRTs)

Problems with PRM:

– two steps: graph construction, then graph search

– hard to apply to problems where edges are directed, i.e. kinodynamic problems

RRTs solve both of these problems:

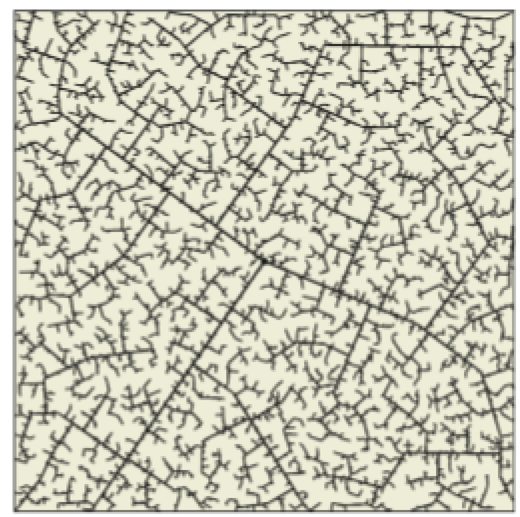
– create a tree instead of graph: no graph search needed!

– tree rooted at start or goal – edges can be directed

RRT Algorithm

- **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

RRT Algorithm

- The algorithm: Given C and q_0

Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add\_edge}(q_{near}, q_{rand})$
until *condition*

Sample from a **bounded region** centered around q_0

E.g. an axis-aligned relative random translation or random rotation

(but recall sampling over rotation spaces problem)

RRT Algorithm

- The algorithm

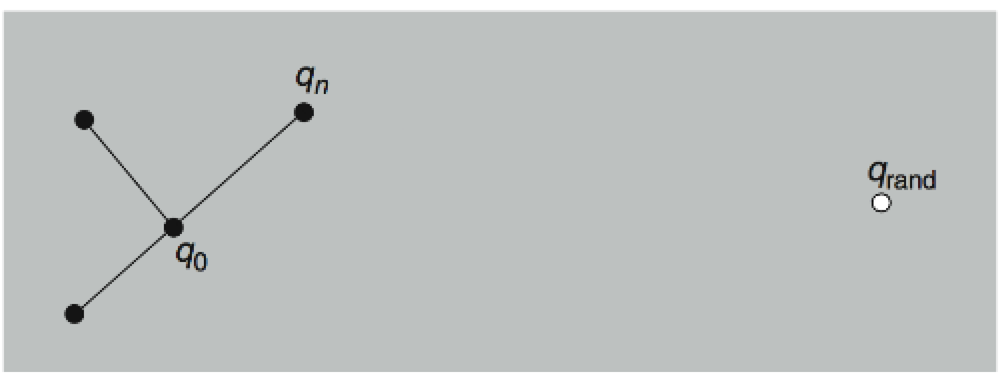
Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add\_edge}(q_{near}, q_{rand})$
until *condition*

Finds closest vertex in G using a **distance function**

$$\rho : \mathcal{C} \times \mathcal{C} \rightarrow [0, \infty)$$

formally a ***metric*** defined on C

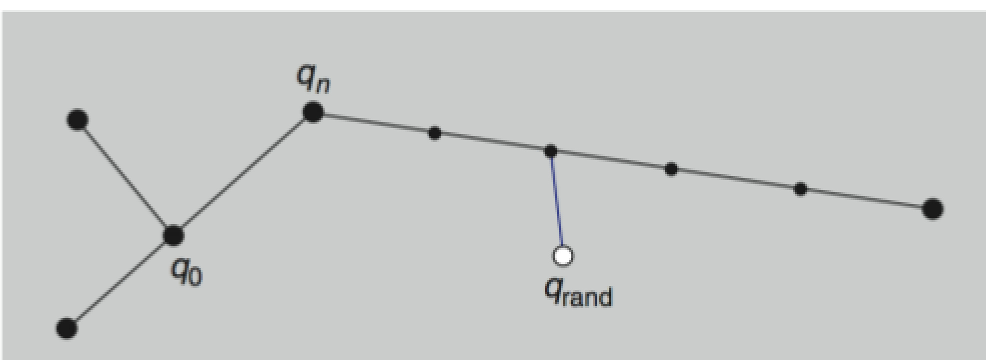


RRT Algorithm

- The algorithm

Algorithm 1: RRT

G.init(q_0)
repeat
    $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
    $q_{near} \leftarrow$ NEAREST(G, q_{rand})
    G.add_edge(q_{near}, q_{rand})
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

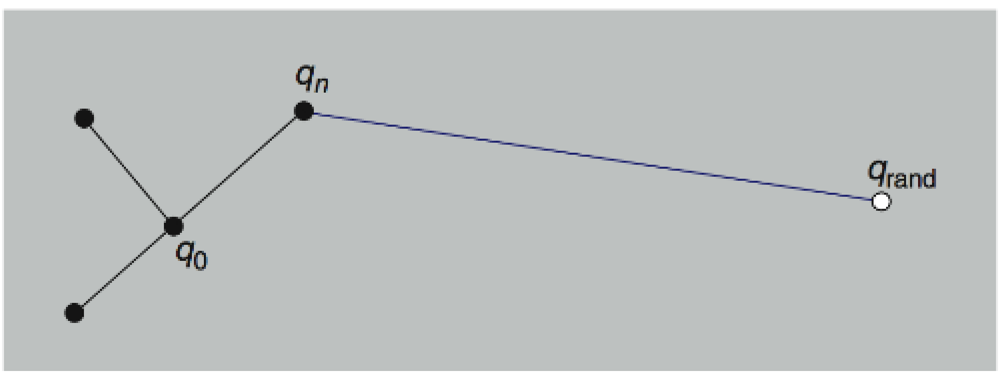
- The algorithm

Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add\_edge}(q_{near}, q_{rand})$
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

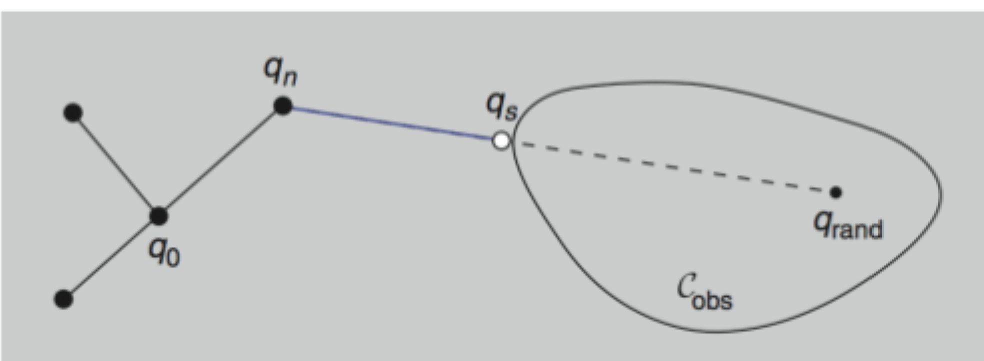
- The algorithm

Algorithm 1: RRT

$G.\text{init}(q_0)$
repeat
  $q_{rand} \rightarrow \text{RANDOM\_CONFIG}(\mathcal{C})$
  $q_{near} \leftarrow \text{NEAREST}(G, q_{rand})$
  $G.\text{add\_edge}(q_{near}, q_{rand})$
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}



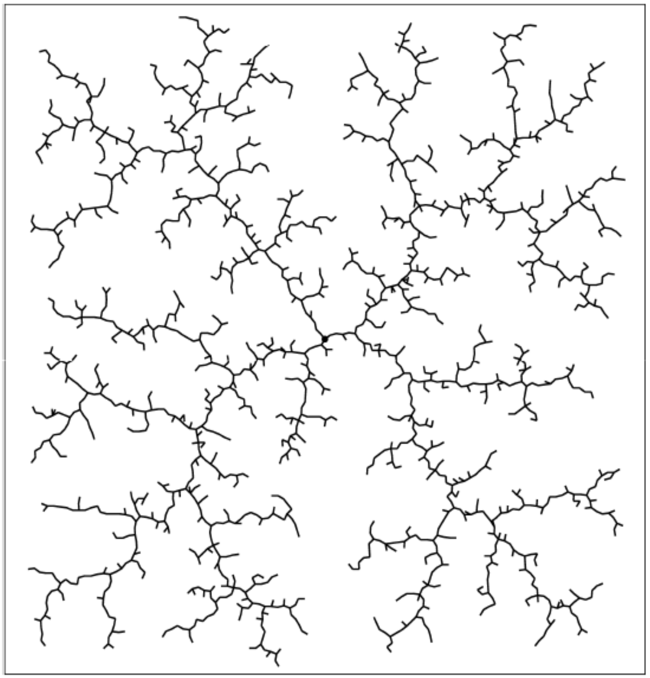
RRT Algorithm

- How to perform path planning with RRTs?
  1. Start RRT at q_I
  2. At every, say, 100th iteration, force $q_{rand} = q_G$
  3. If q_G is reached, problem is solved

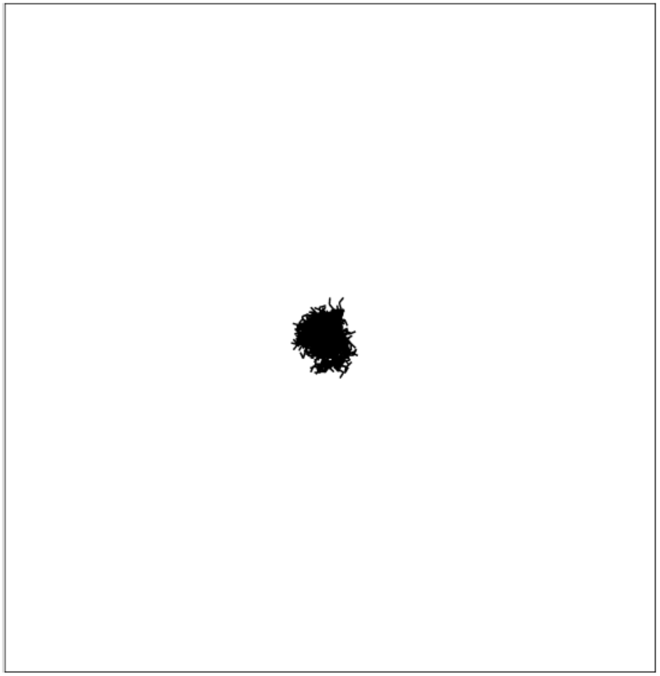
- Why not picking q_G every time?

RRT versus a naïve random tree

RRT

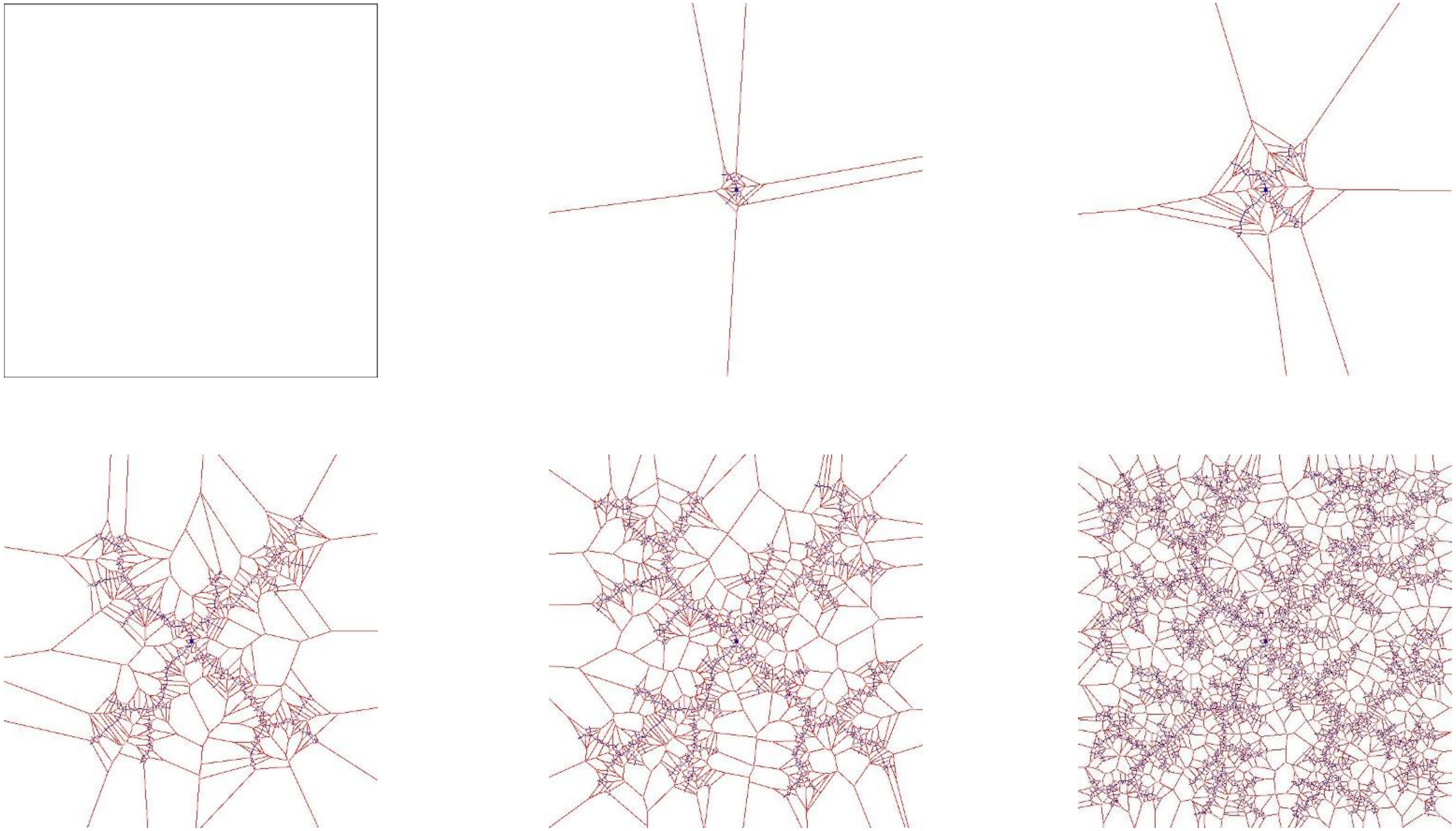


Naïve random tree



<u>Growing the naïve random tree</u>:
1. pick a node at random
2. sample a new node near it
3. grow tree from random node to new node

RRTs and
Bias toward large Voronoi regions



http://msl.cs.uiuc.edu/rrt/gallery.html

Biases

- Bias toward larger spaces
- Bias toward goal
  - When generating a random sample, with some probability pick the goal instead of a random node when expanding
  - This introduces another parameter
  - 5-10% is probably the right choice

RRT probabilistic completeness

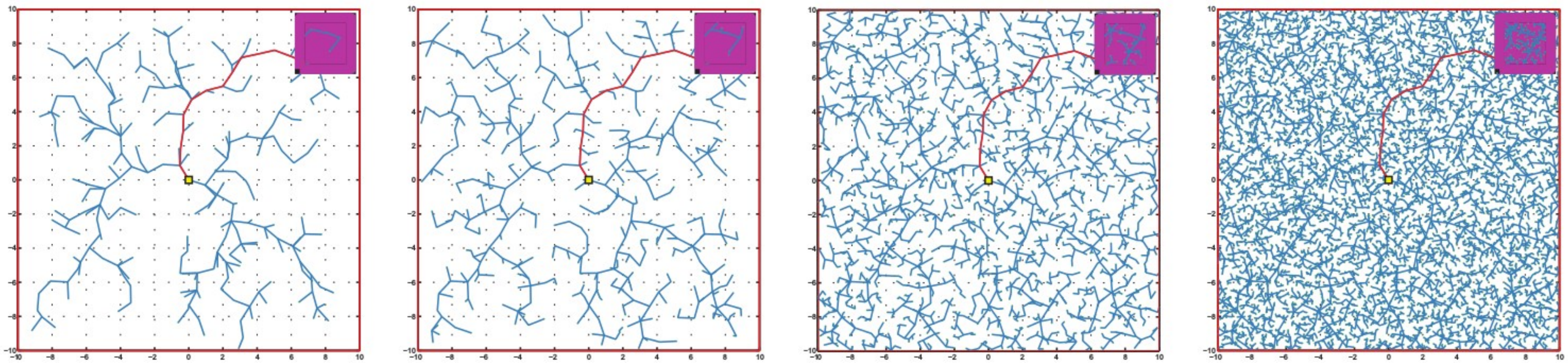
Theorem 16 (Probabilistic completeness of RRT (LaValle and Kuffner 2001)). *Consider a robustly feasible path planning problem* $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. *There exist constants* $a > 0$ *and* $n_0 \in \mathbb{N}$, *both dependent only on* $\mathcal{X}_{\text{free}}$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

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

Notice that this is exactly the same bound as for sPRM.

RRT does not find optimal paths

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



Is there a version of RRT that is optimal?

Yes: RRG and RRT*

Algorithm 5: RRG.

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

Don't just connect x_{new} to x_{near}

Attempt to connect to every vertex within a radius r

Use same variable radius as in PRM*

RRG Properties

RRG is complete … how do you know?

RRG Properties

RRG is complete … how do you know?

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

RRG Properties

RRG is complete … how do you know?

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

But, why might RRT still be preferable to RRG?

RRT* Algorithm

Algorithm 6: RRT*.

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

Don't just connect x_{new} to x_{near}

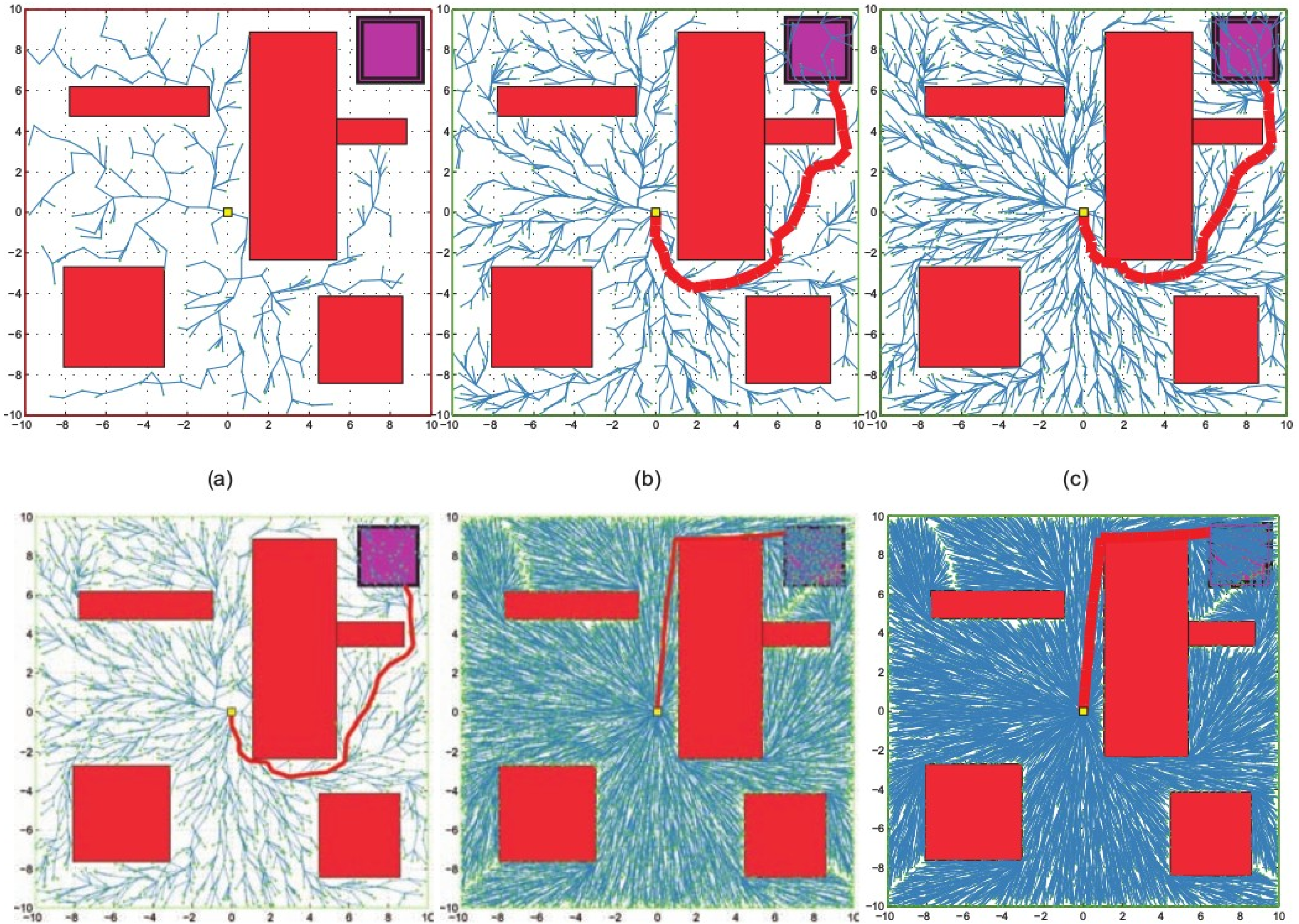
Attempt to connect to every vertex within a radius r

Use same variable radius as in PRM*

Get position and cost of min-cost vertex in

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

RRT* Algorithm



(a) (b) (c)

(d) (e) (f)

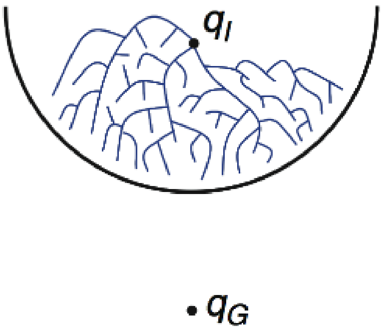
RRT* Algorithm

RRT* is complete … how do you know?

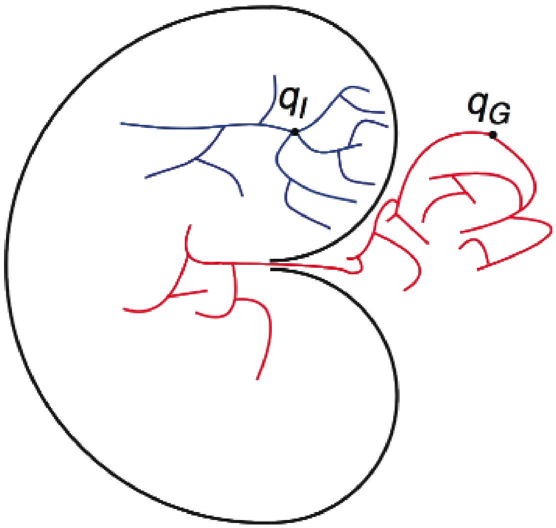
Theorem 38 (Asymptotic optimality of RRT*). *If* $\gamma_{\mathrm{RRT}^*} > (2\,(1 + 1/d))^{1/d} \left(\frac{\mu(X_{\mathrm{free}})}{\zeta_d}\right)^{1/d}$, *then the RRT* algorithm is asymptotically optimal.*

Bidirectional RRT (RRT Connect)

- However, some problems require more effective methods: **bidirectional search**
- Grow **two** RRTs, one from q_I, one from q_G
- In every other step, try to extend each tree towards the newest vertex of the other tree



Filling a well



A bug trap

Bidirectional RRT (RRT Connect)

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

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

for k = 1 to K do

q_{rand} = RANDOM_CONFIG();

if (q_{new}= EXTEND(T_a, q_{rand}) == Reached) then

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

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

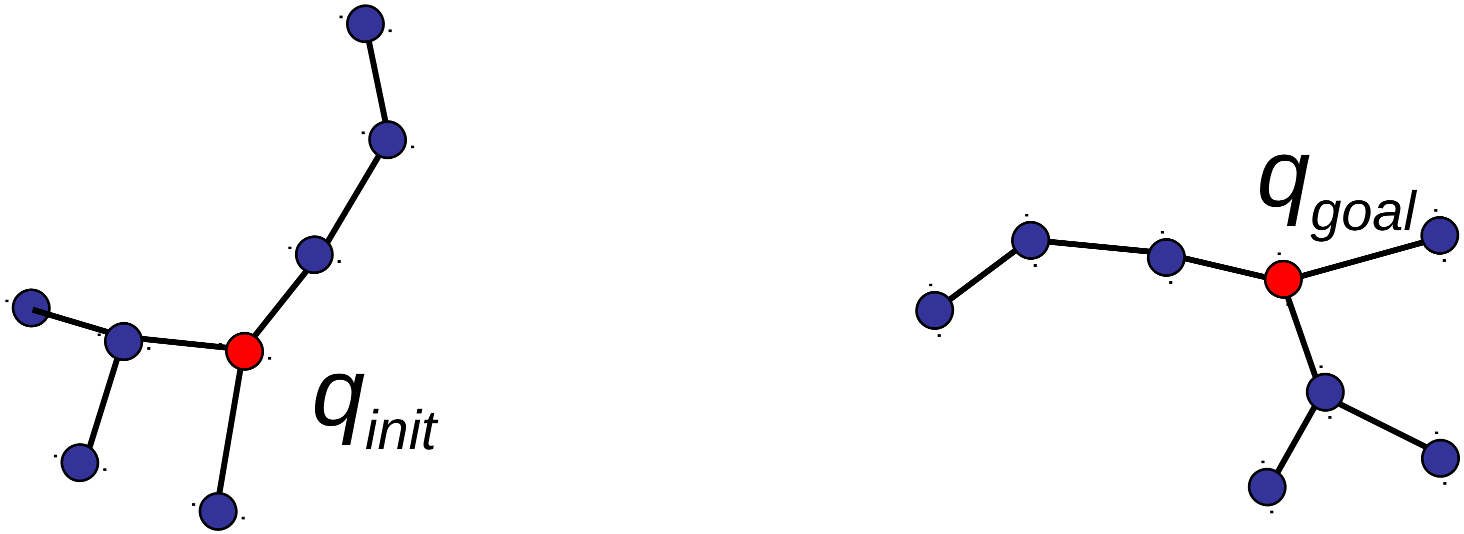
SWAP($T_{a,}$ T_b);

Return Failure;

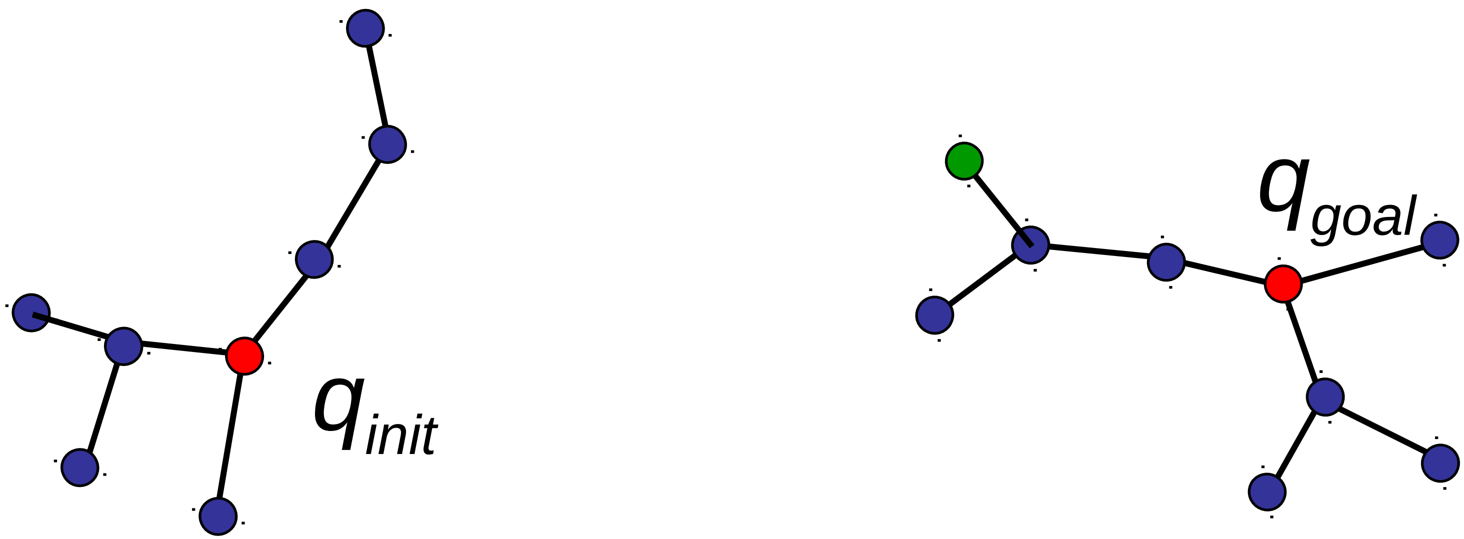
}

Instead of switching, use T_a as smaller tree.

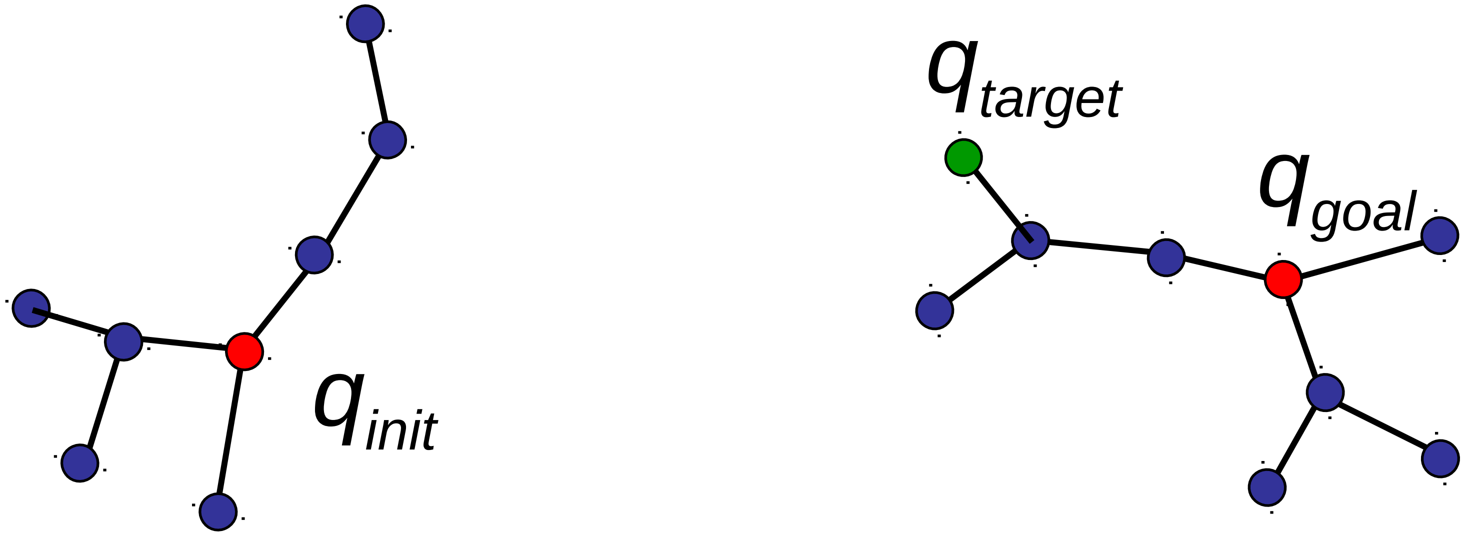
A single RRT-Connect iteration...



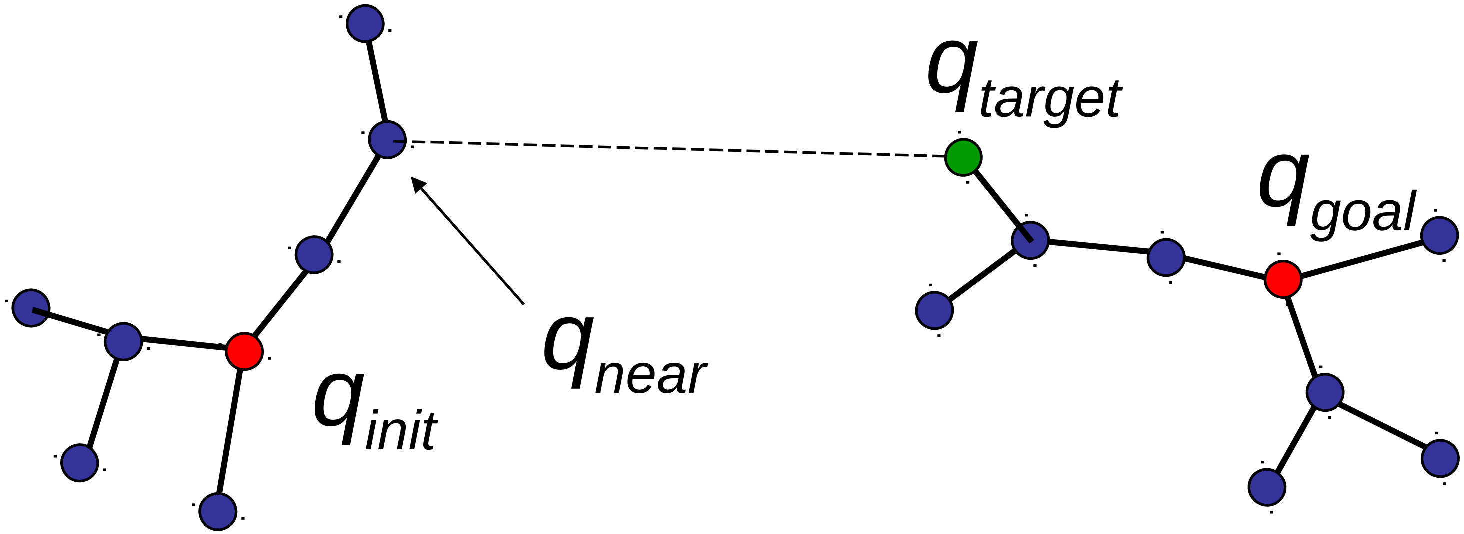
1) One tree grown using random target



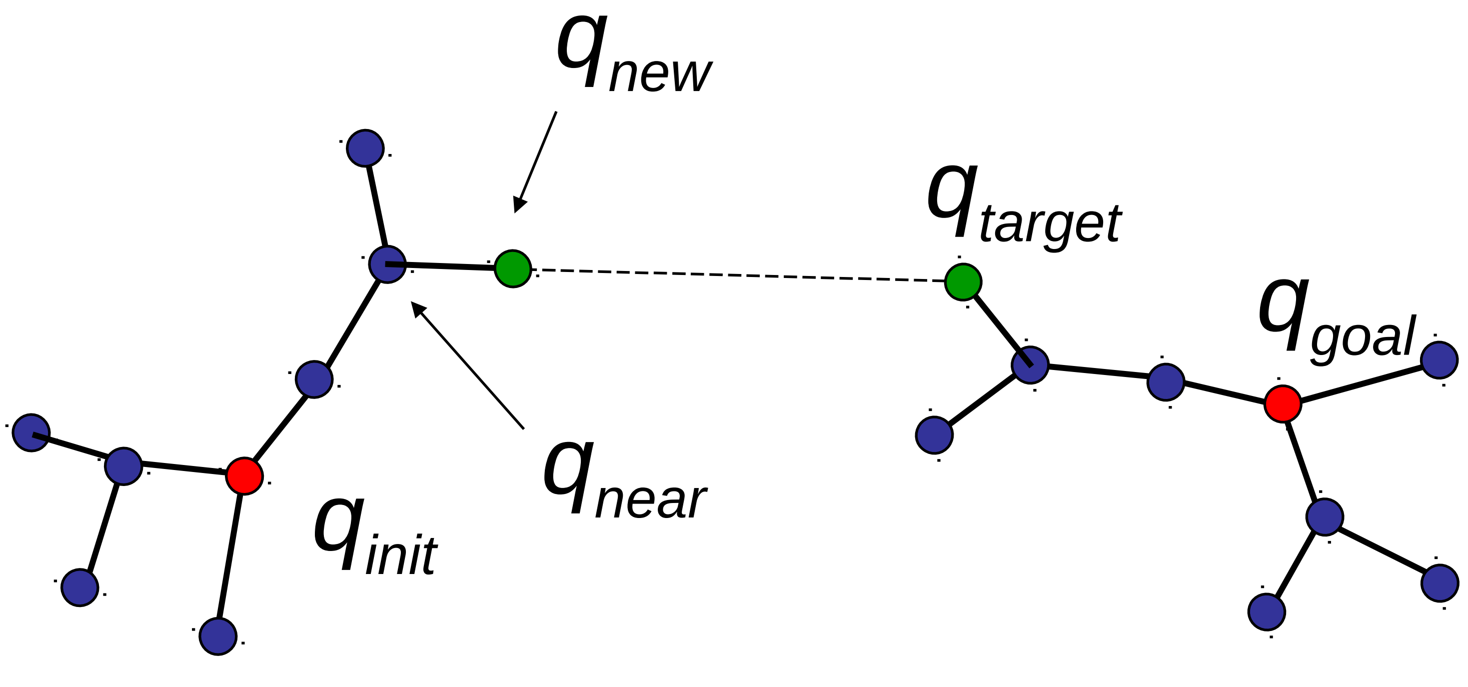
2) New node becomes target for other tre



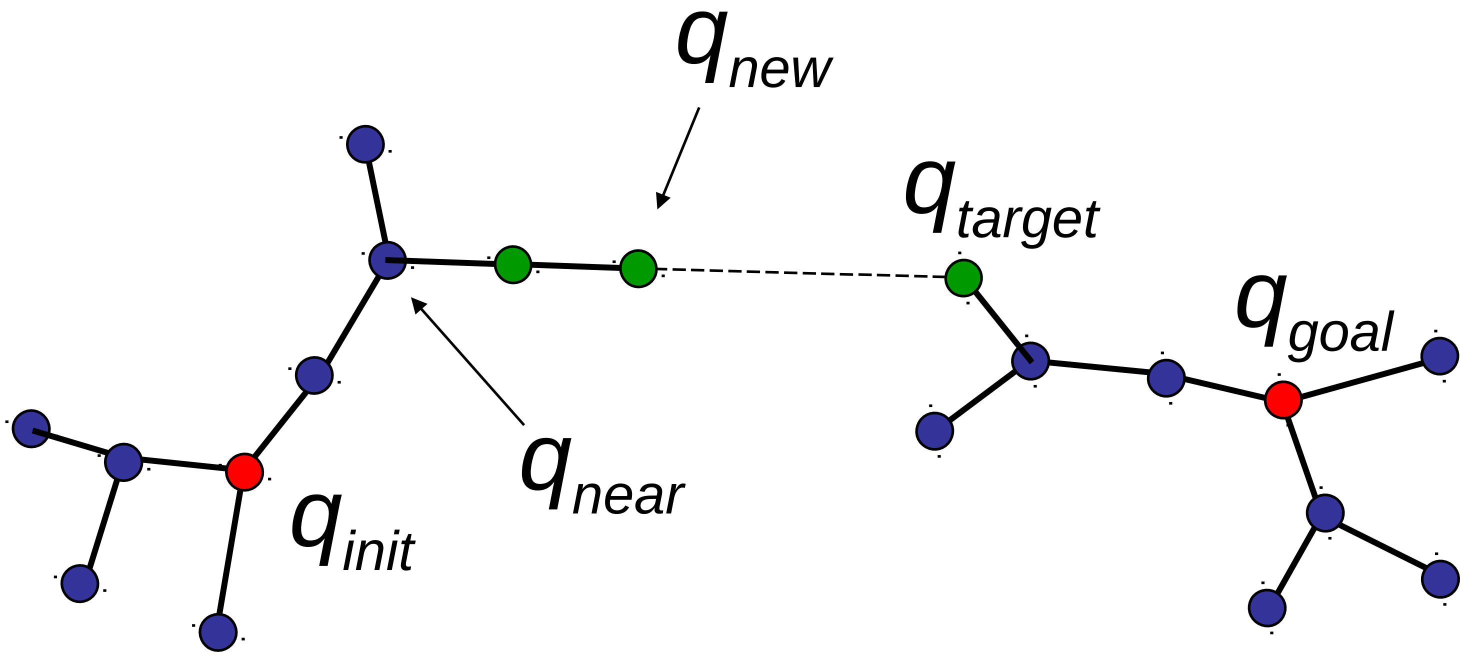
3) Calculate node "nearest" to target



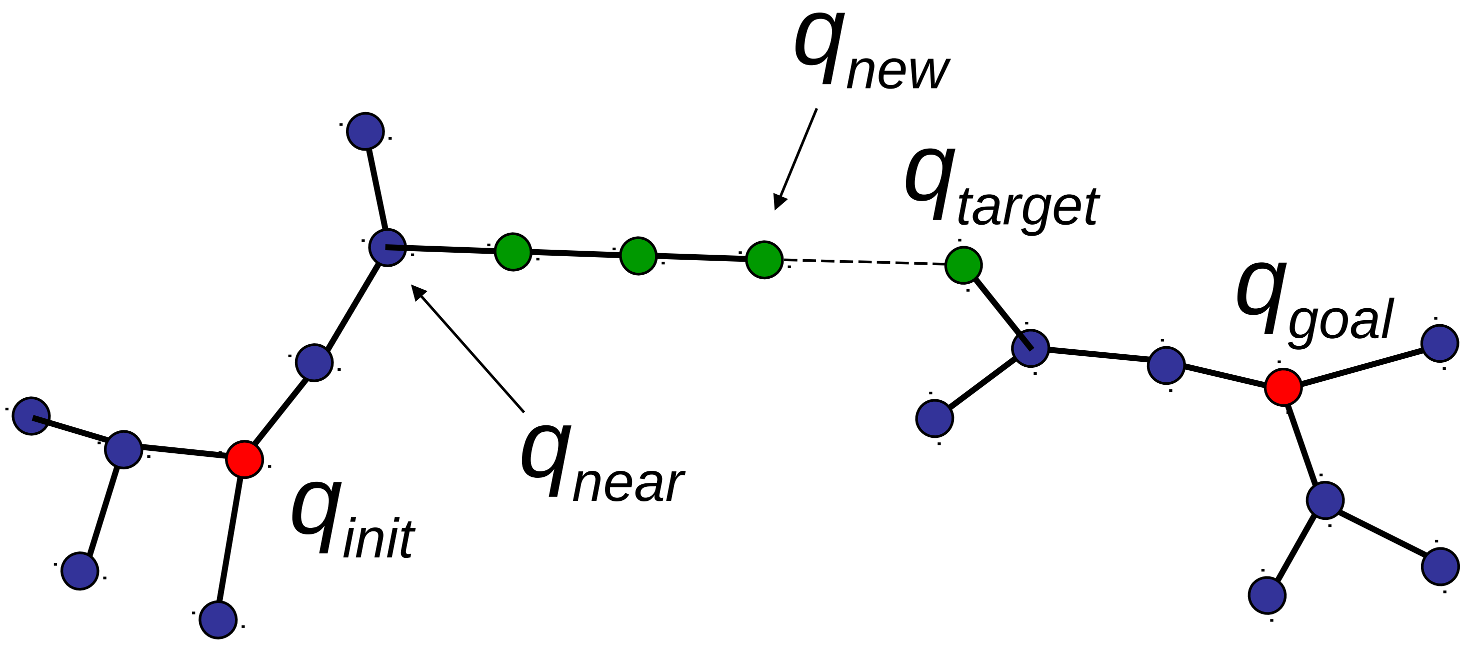
4) Try to add new collision-free branch



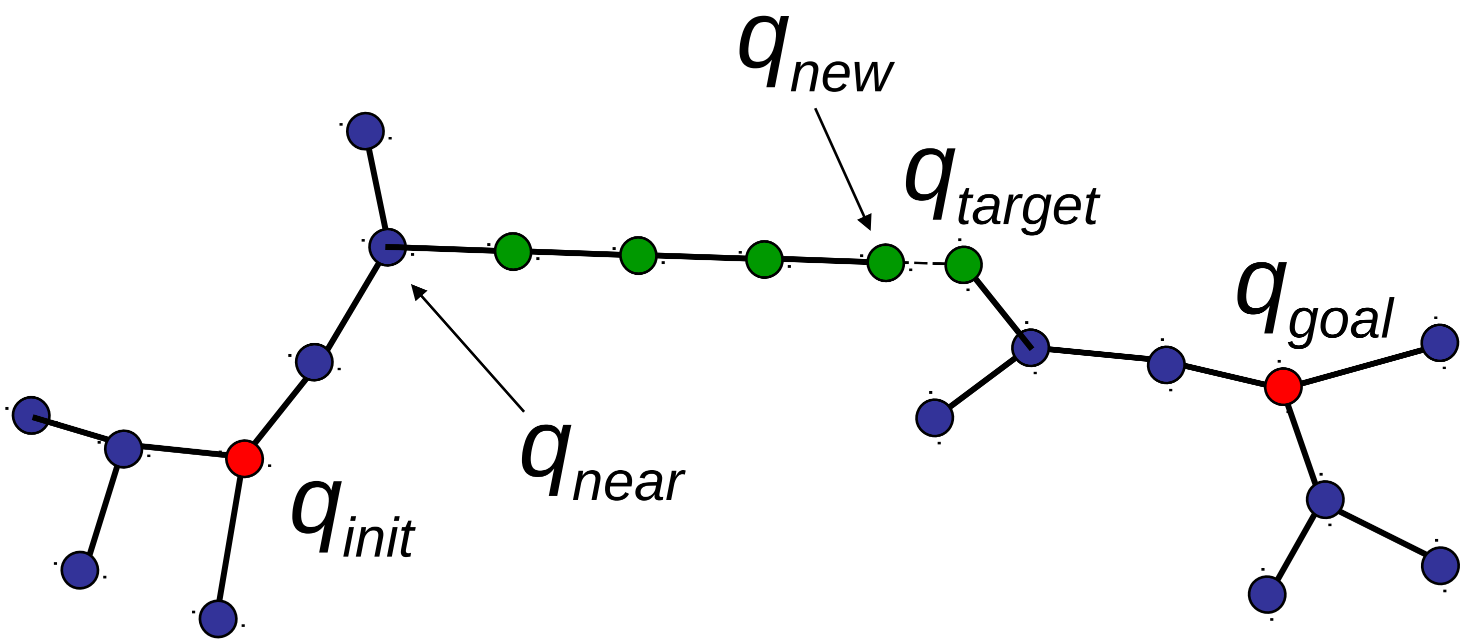
5) If successful, keep extending branch



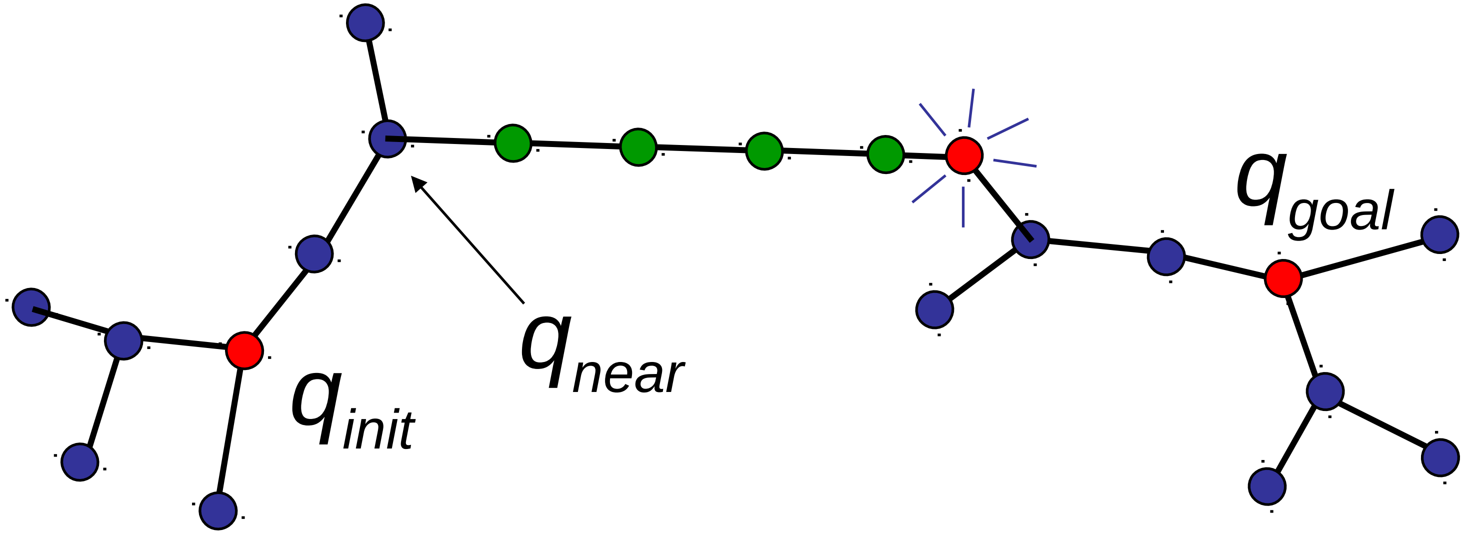
5) If successful, keep extending branch



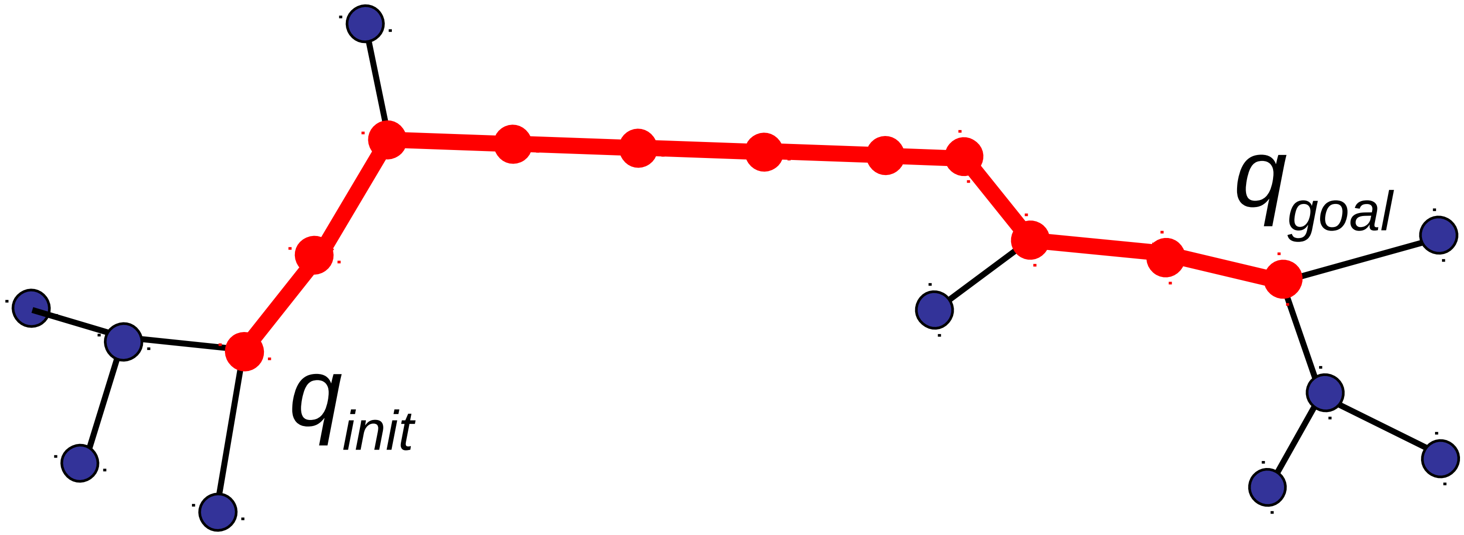
5) If successful, keep extending branch



6) Path found if branch reaches target



7) Return path connecting start and goal

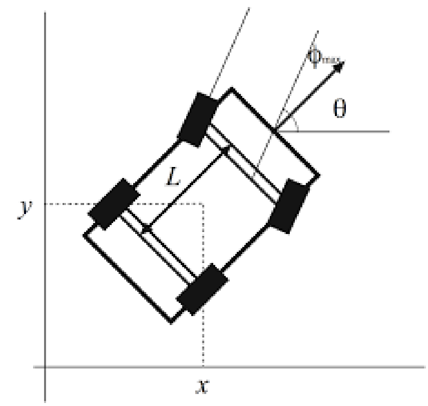


Bidirectional RRT (RRT Connect)

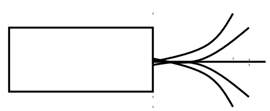
Is bi-directional RRT always better?

Kinodynamic planning with RRTs

So far, we have assumed that the system has no dynamics
– the system can instantaneously move in any direction in c-space
– but what if that's not true???



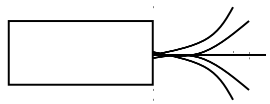
Consider the Dubins car:
– c-space: x-y position and velocity, angle
– control forward velocity and steering angle
– plan a path through c-space with the corresponding control signals



$$x_{t+1} = f(x_t, u_t)$$

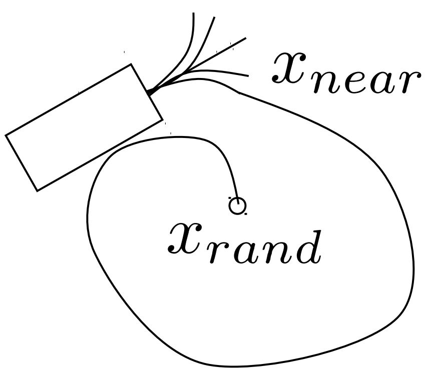
<u>where:</u>
x_t – state (x/y position and velocity, steering angle)
u_t – control signal (forward velocity, steering angle)

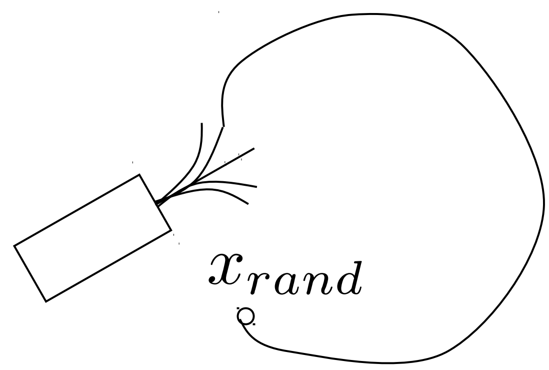
Kinodynamic planning with RRTs



$$x_{t+1} = f(x_t, u_t)$$

$$u^* = \arg\min_u (d(x_{rand}, f(x_{near}, u)))$$





But, what if x_{near} isn't the right node to expand ??

So, what do they do?

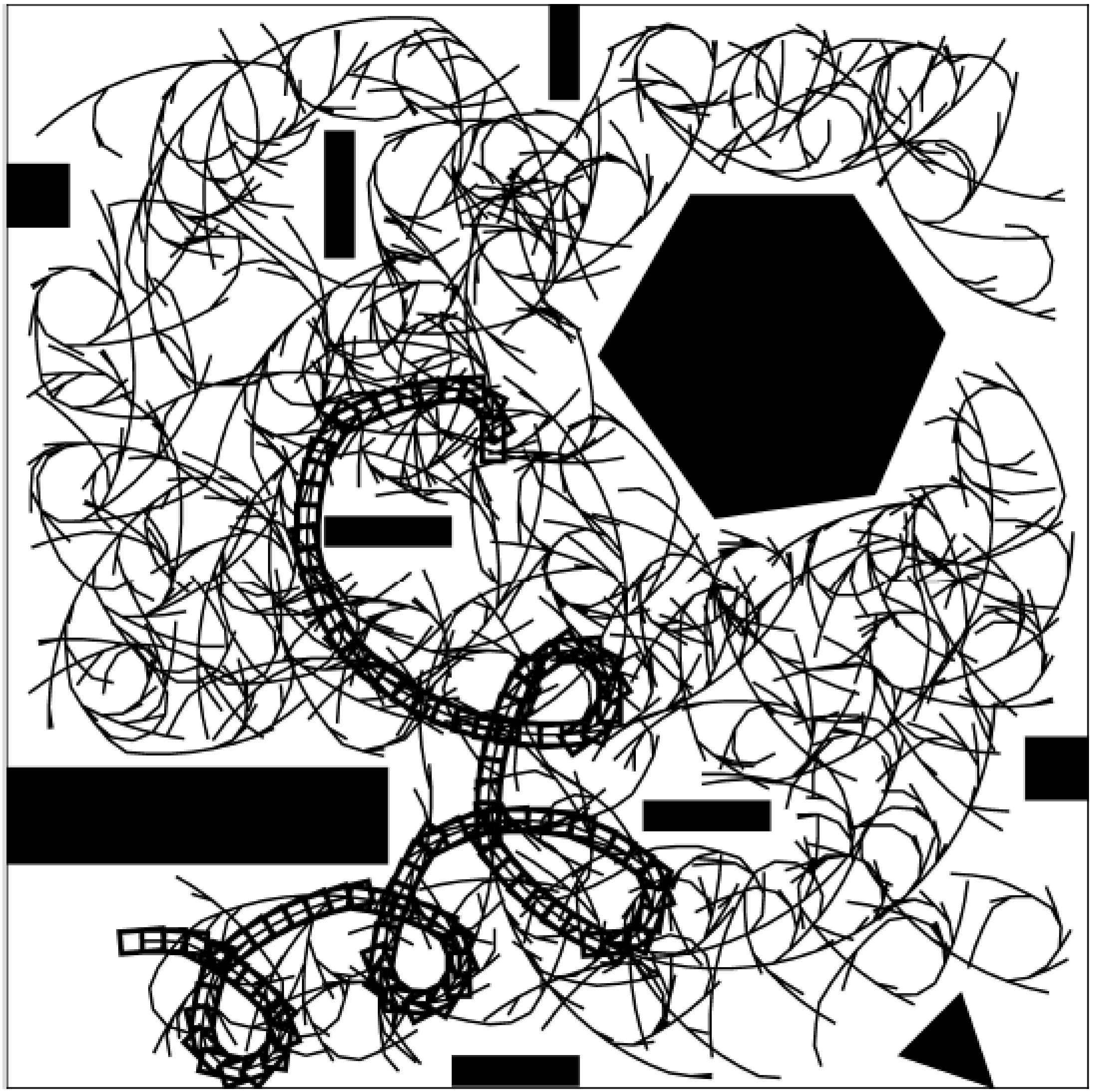
- Use nearest neighbor anyway

- As long as heuristic is not bad, it helps
  (you have already given up completeness and optimality, so what the heck?)

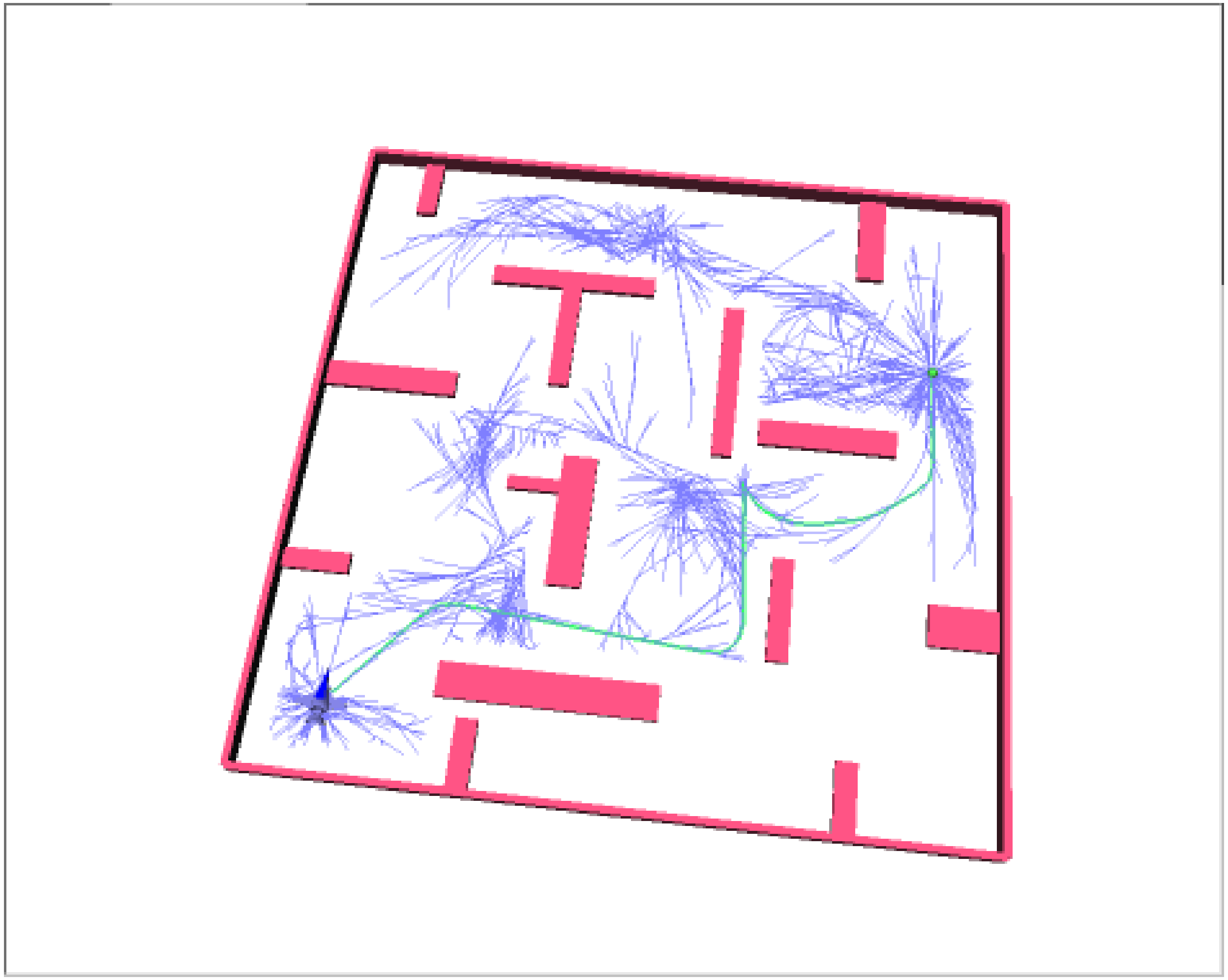
- Nearest neighbor calculations begin to dominate the collision avoidance

- Remember K-D trees

Left-turn only forward car



Hovercraft with 2 Thusters

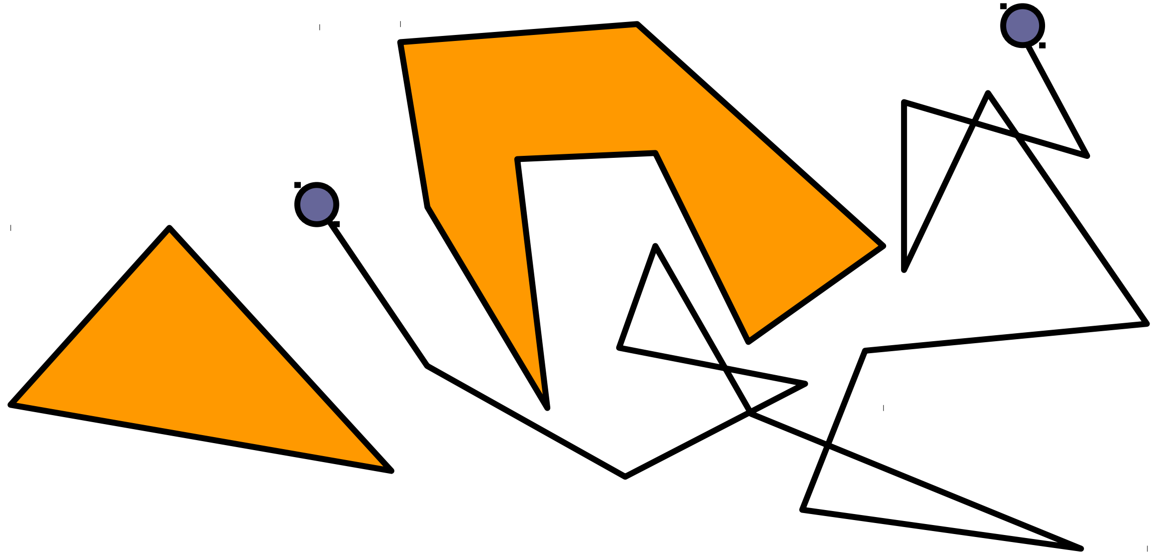


Path Smoothing

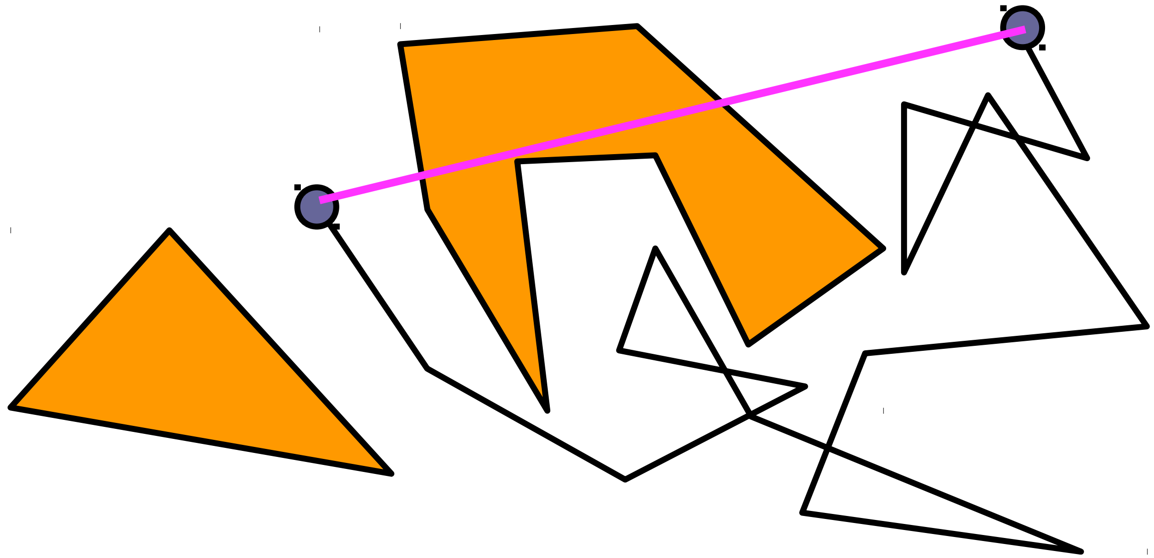
Paths produced by sample based planners are generally not smooth

– RRT* and PRM* converge to optimal paths in the limit, but it’s generally not possible to run these algorithms long enough to converge.

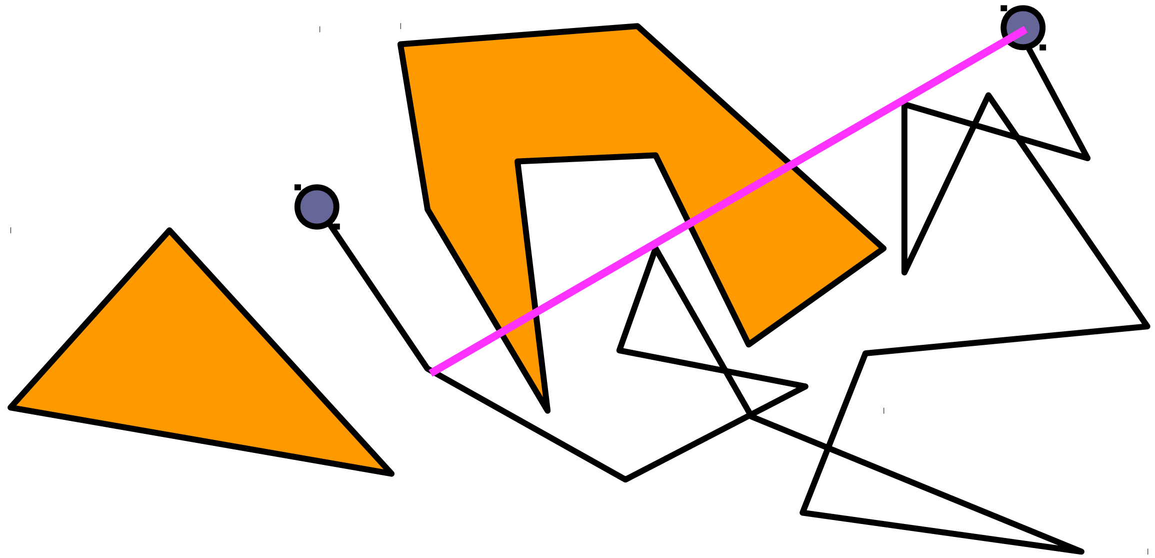
Smoothing the path



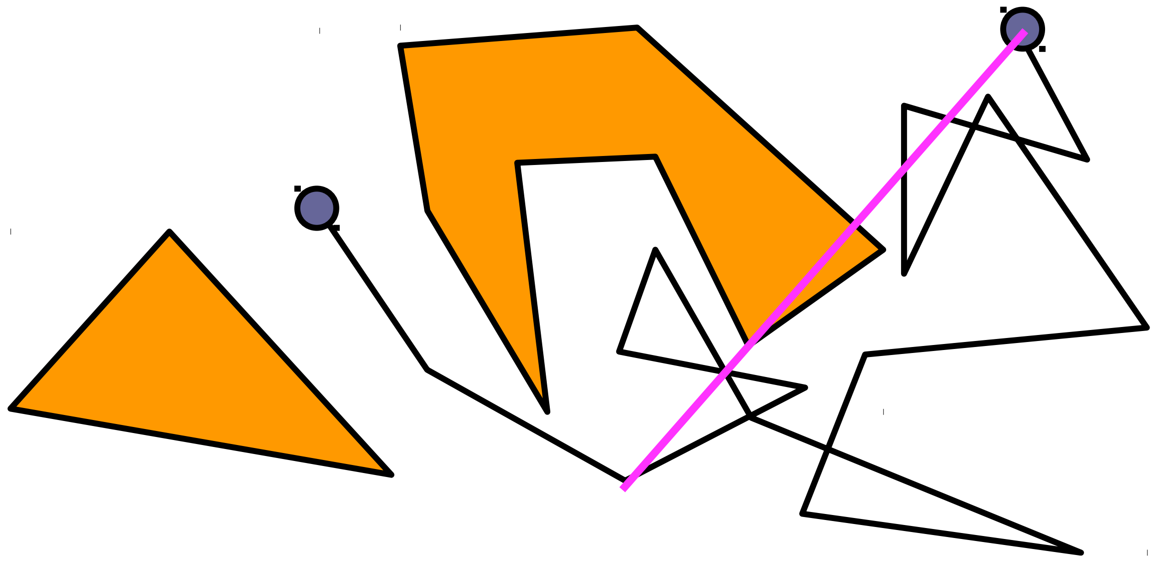
Smoothing the path



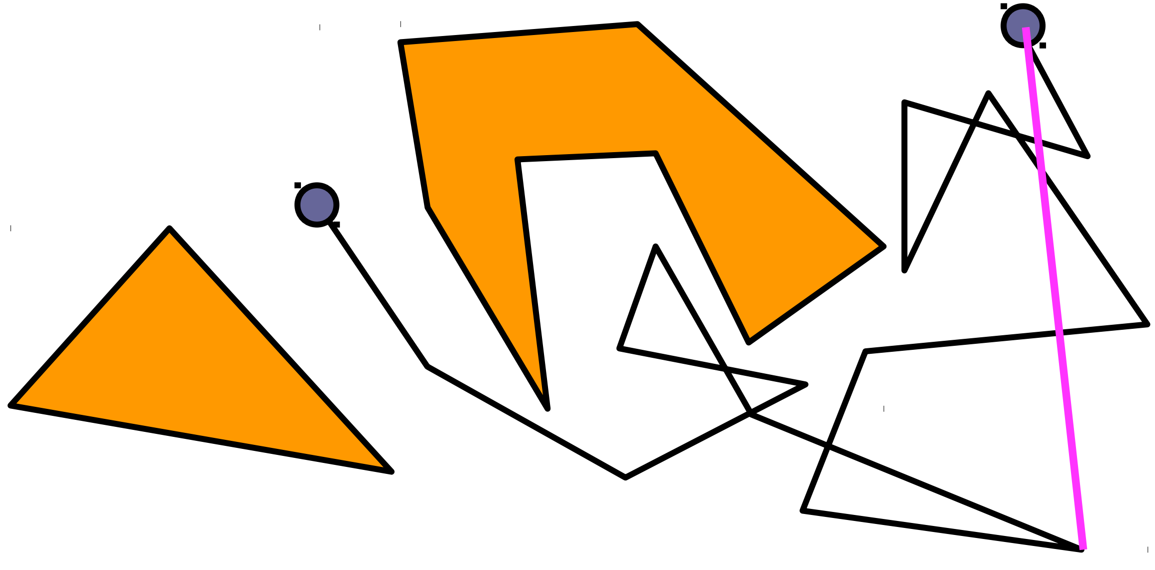
Smoothing the path



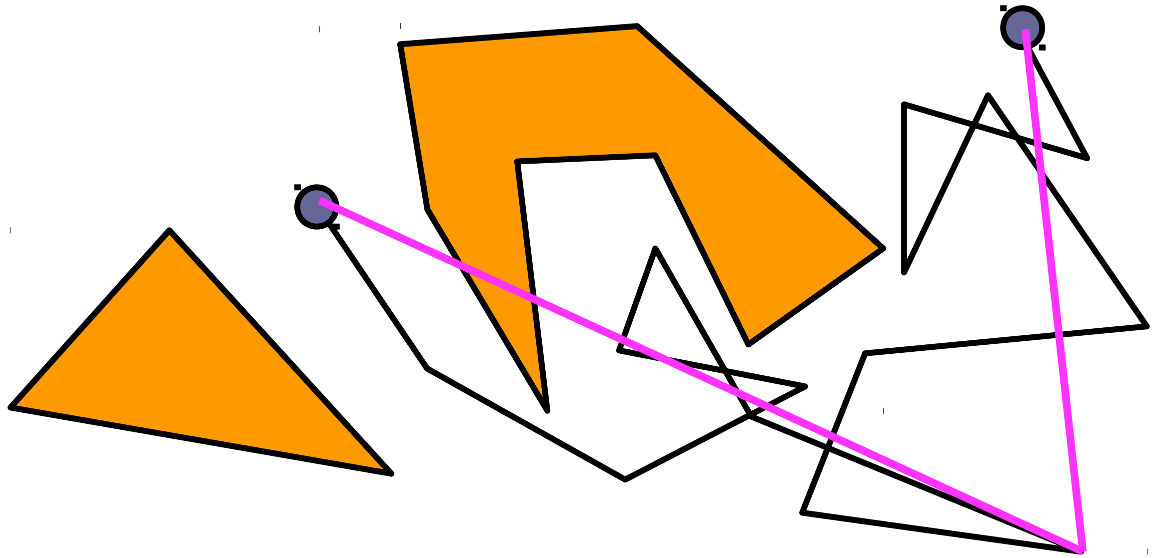
Smoothing the path



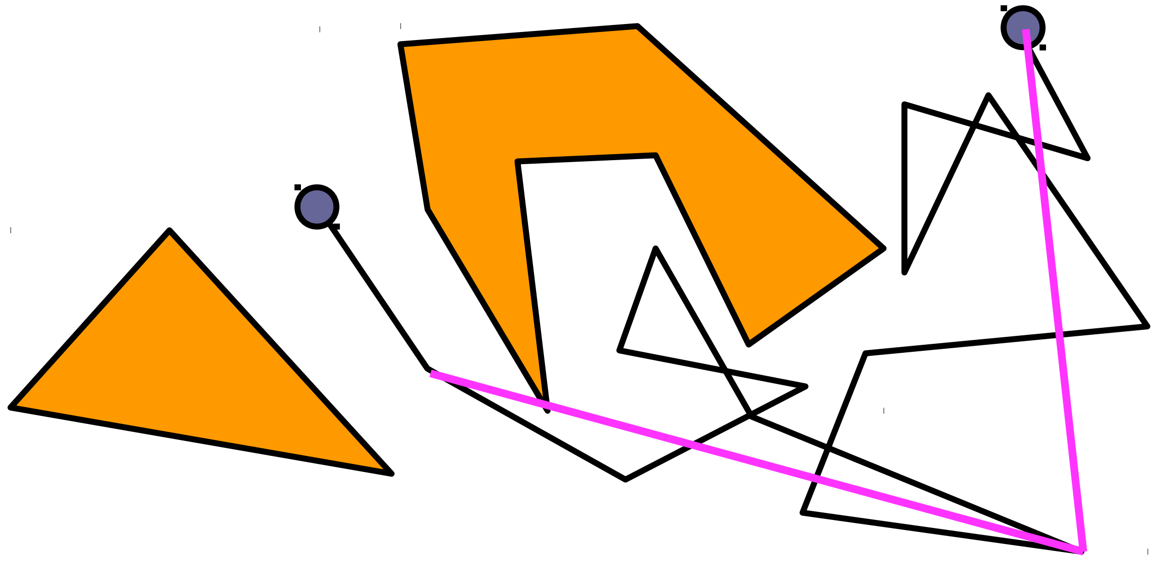
Smoothing the path



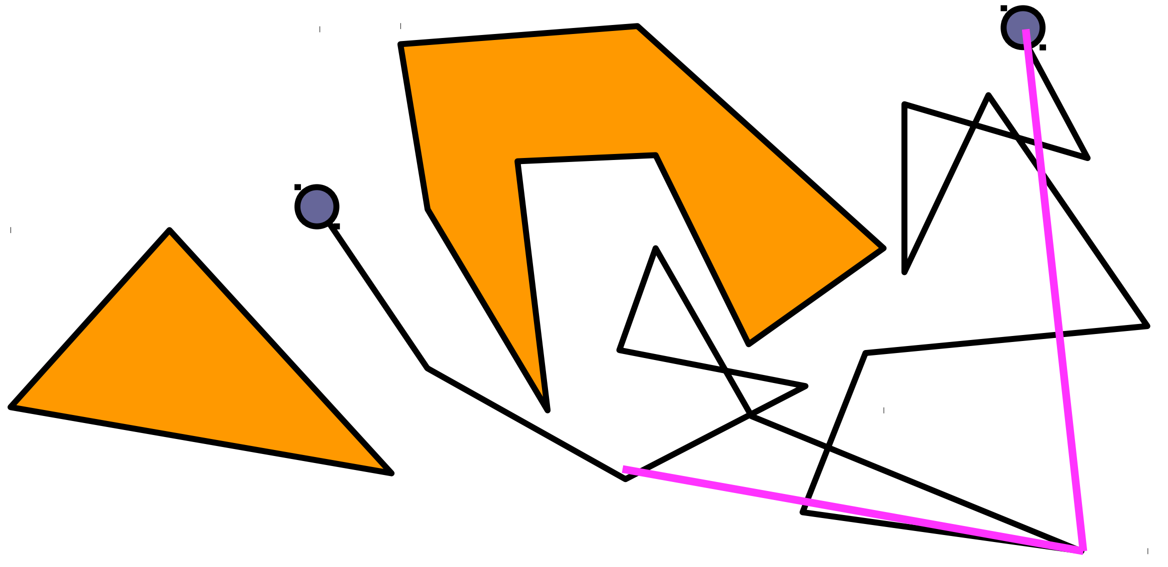
Smoothing the path



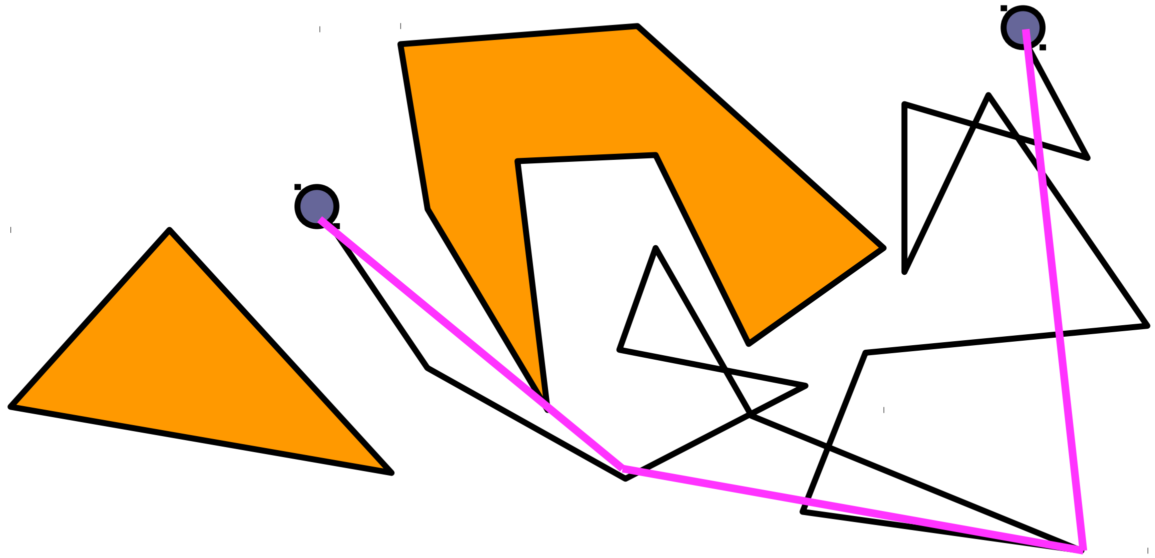
Smoothing the path



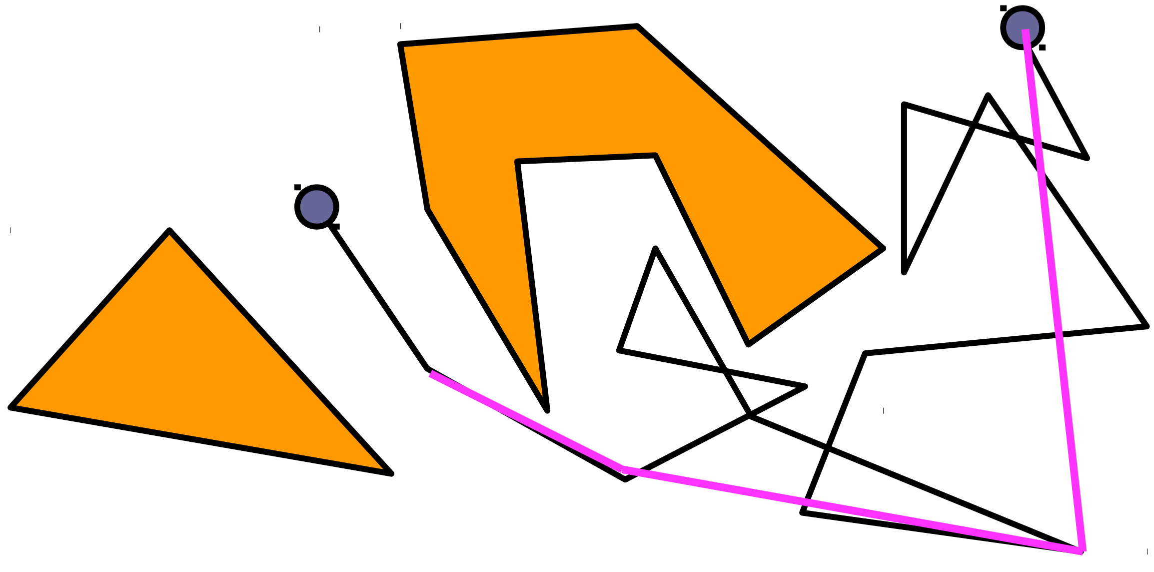
Smoothing the path



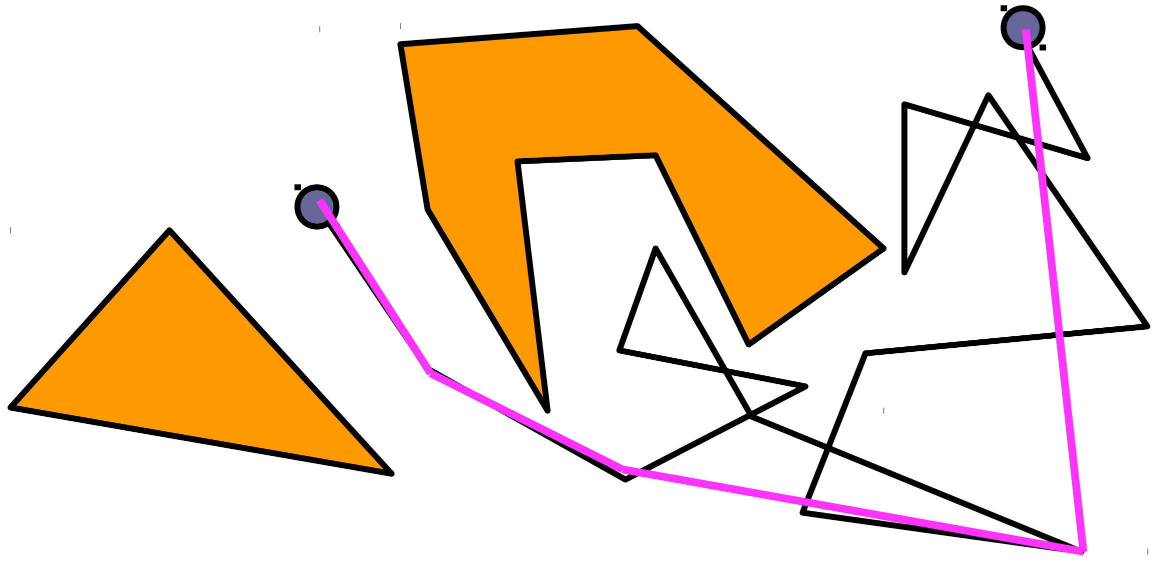
Smoothing the path



Smoothing the path



Smoothing the path



Smoothing the path