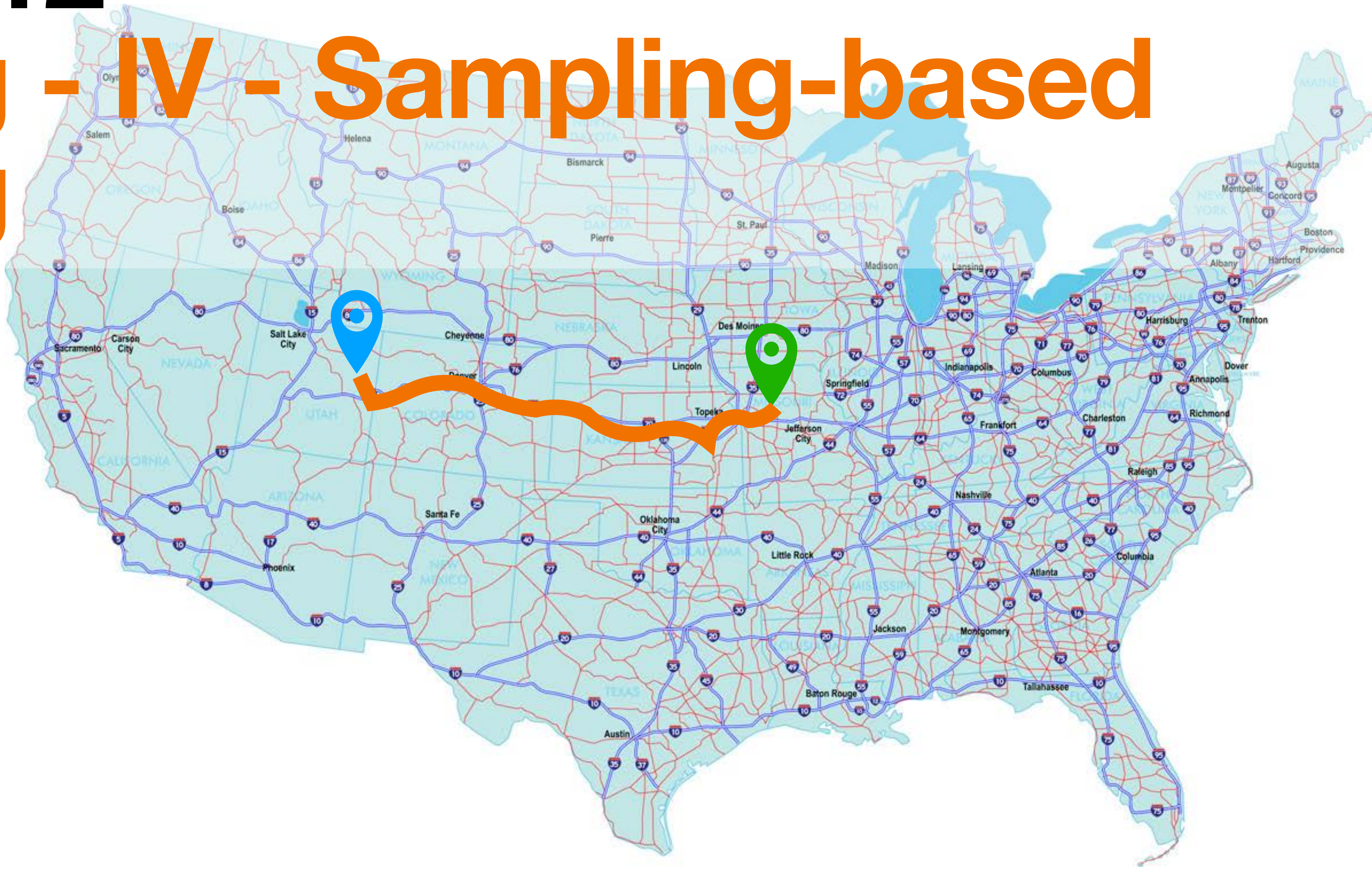
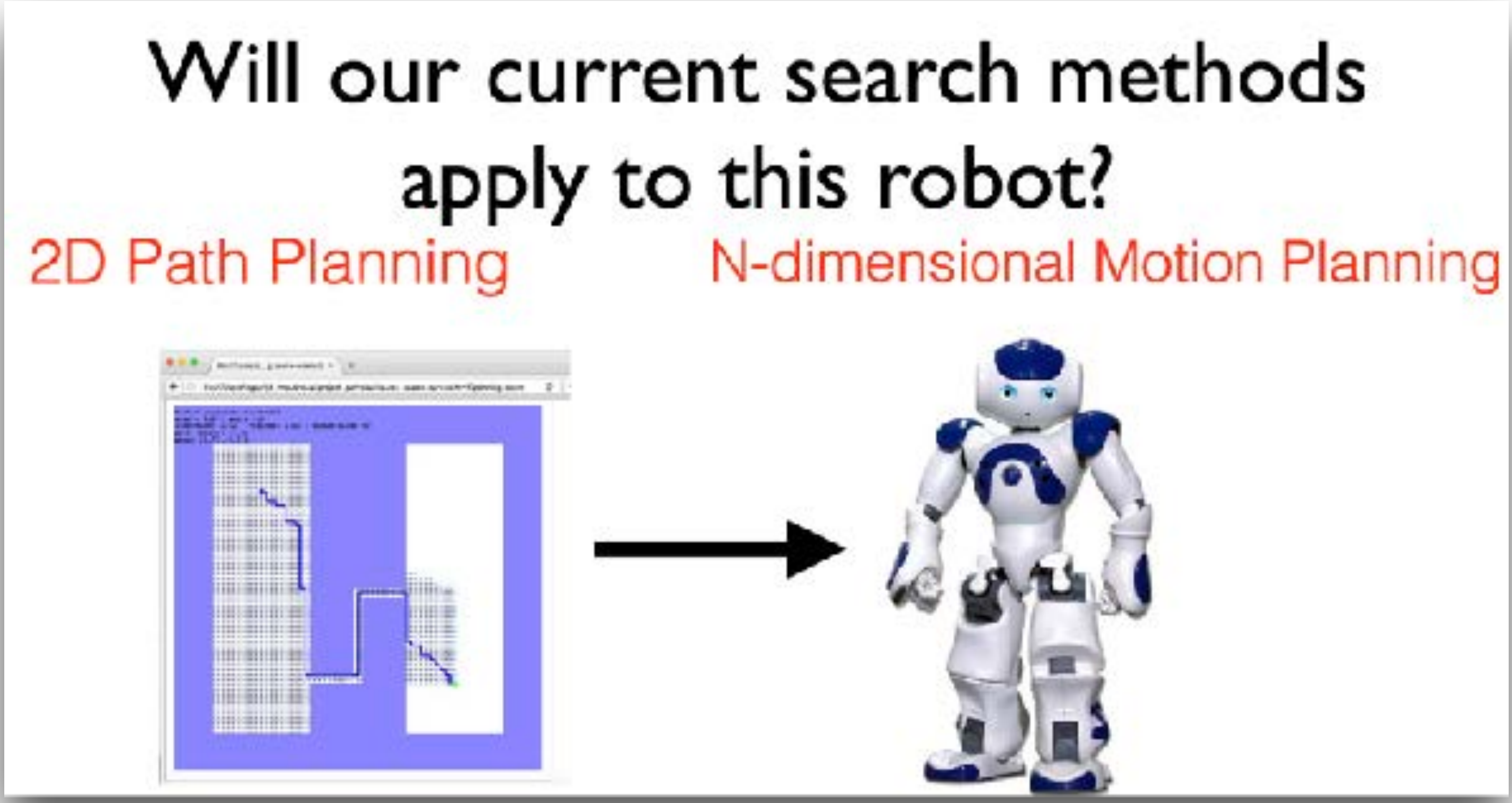
Lecture 12
Planning - IV - Sampling-based Planning

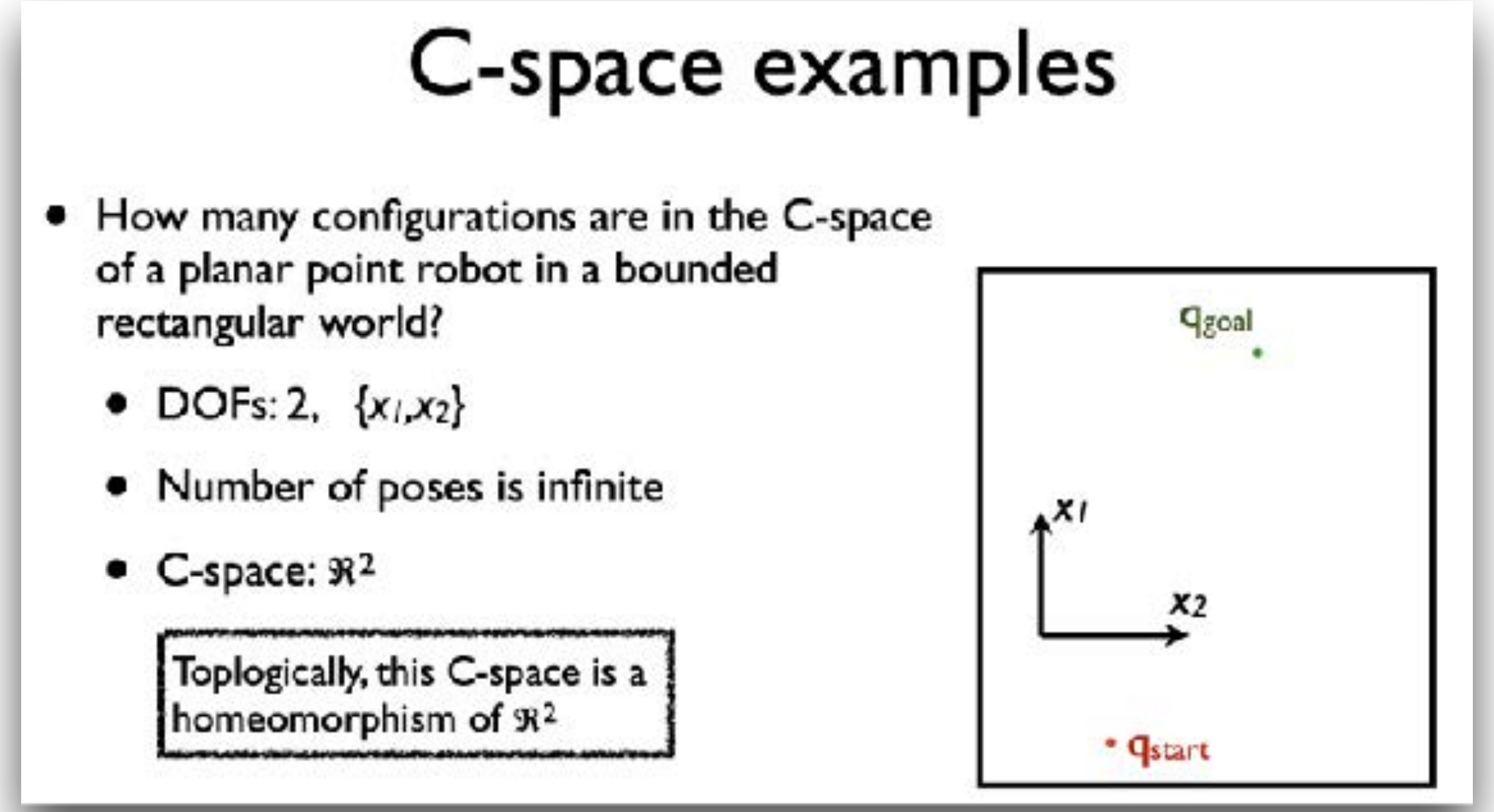


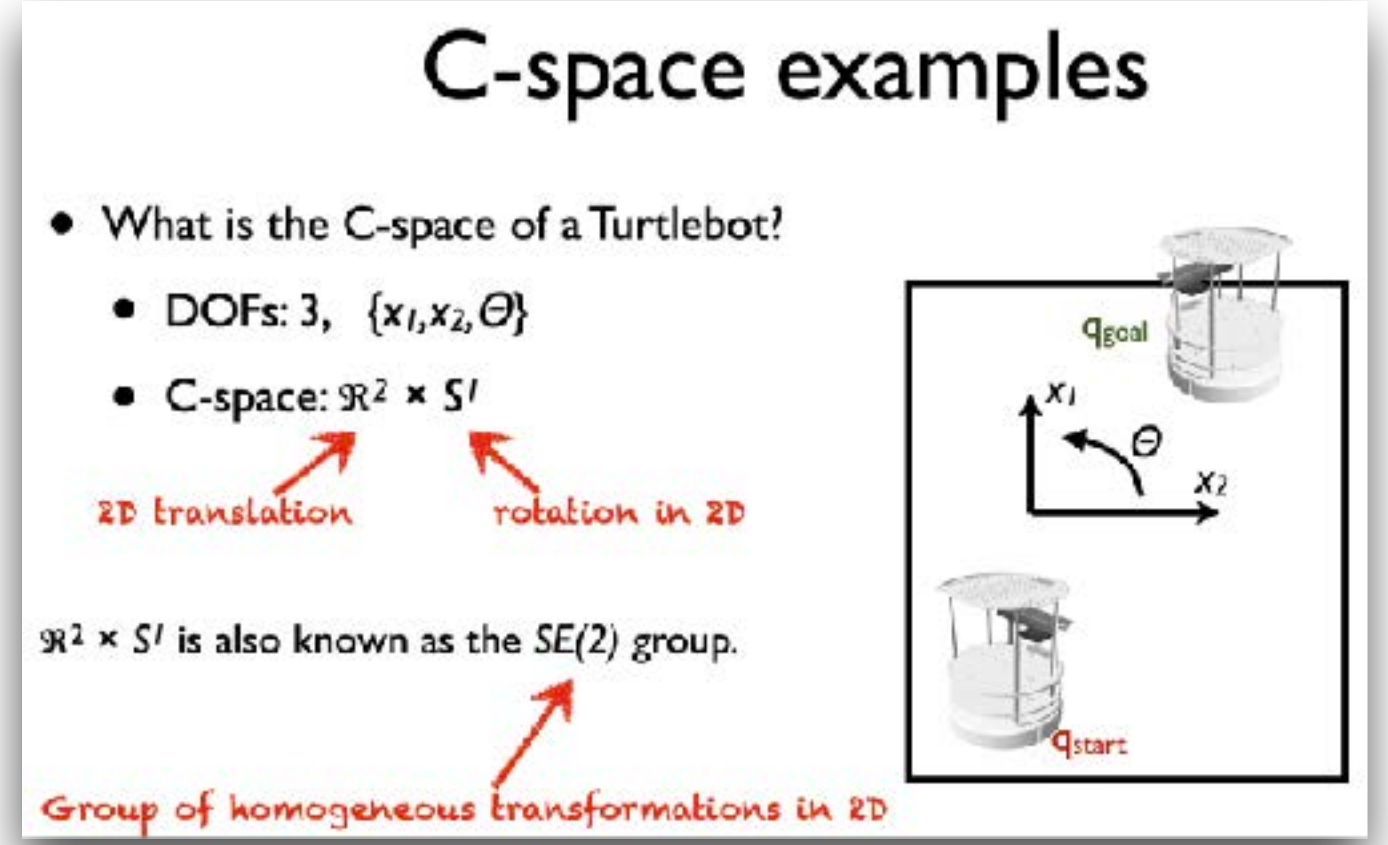
Course Logistics

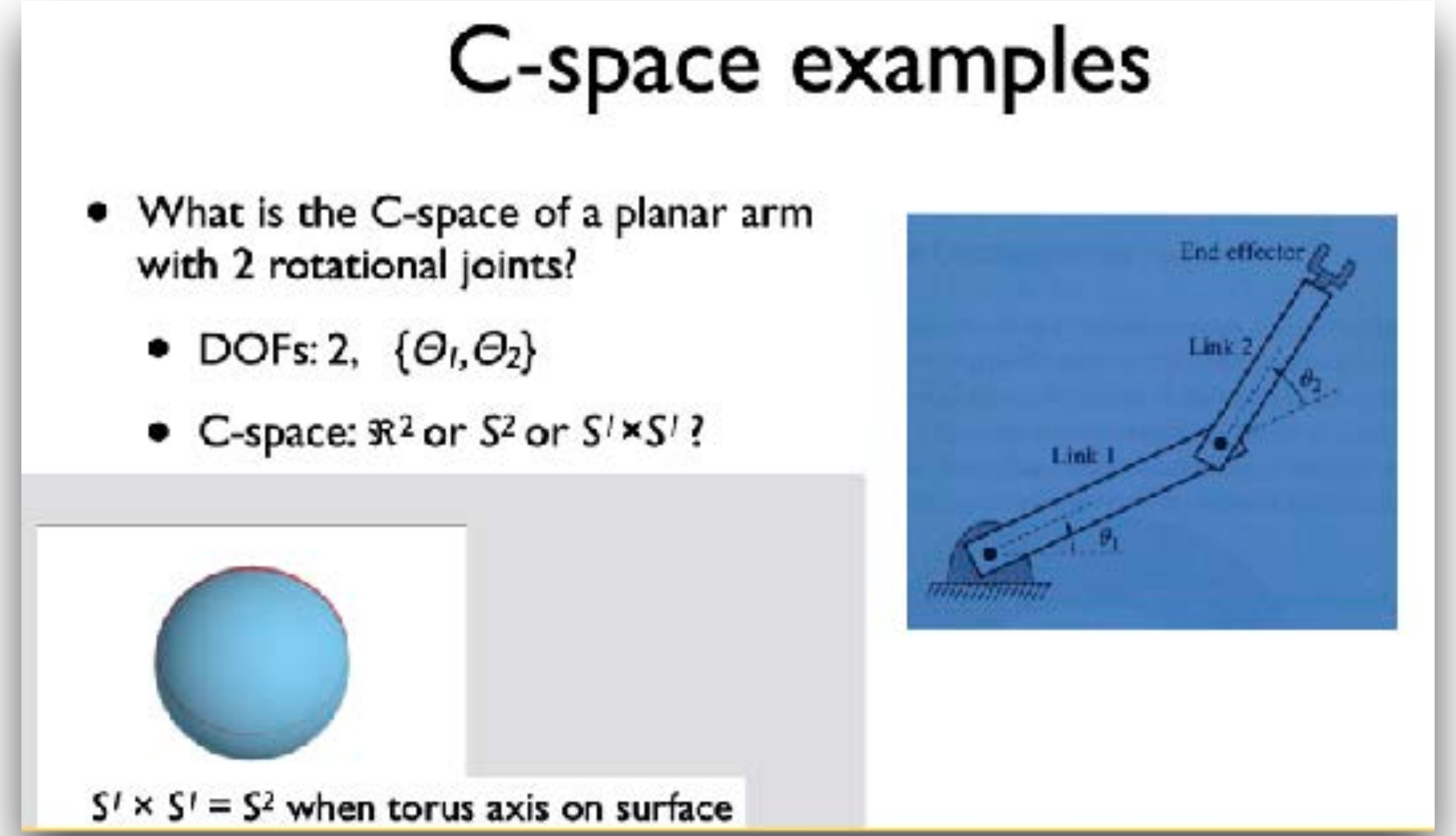
- Quiz 6 will be posted tomorrow at noon and will be due on Wed at noon.
- Project 4 is due on Wed 02/28.
- Project 5 will be posted on 02/28 and will be due on 03/13.

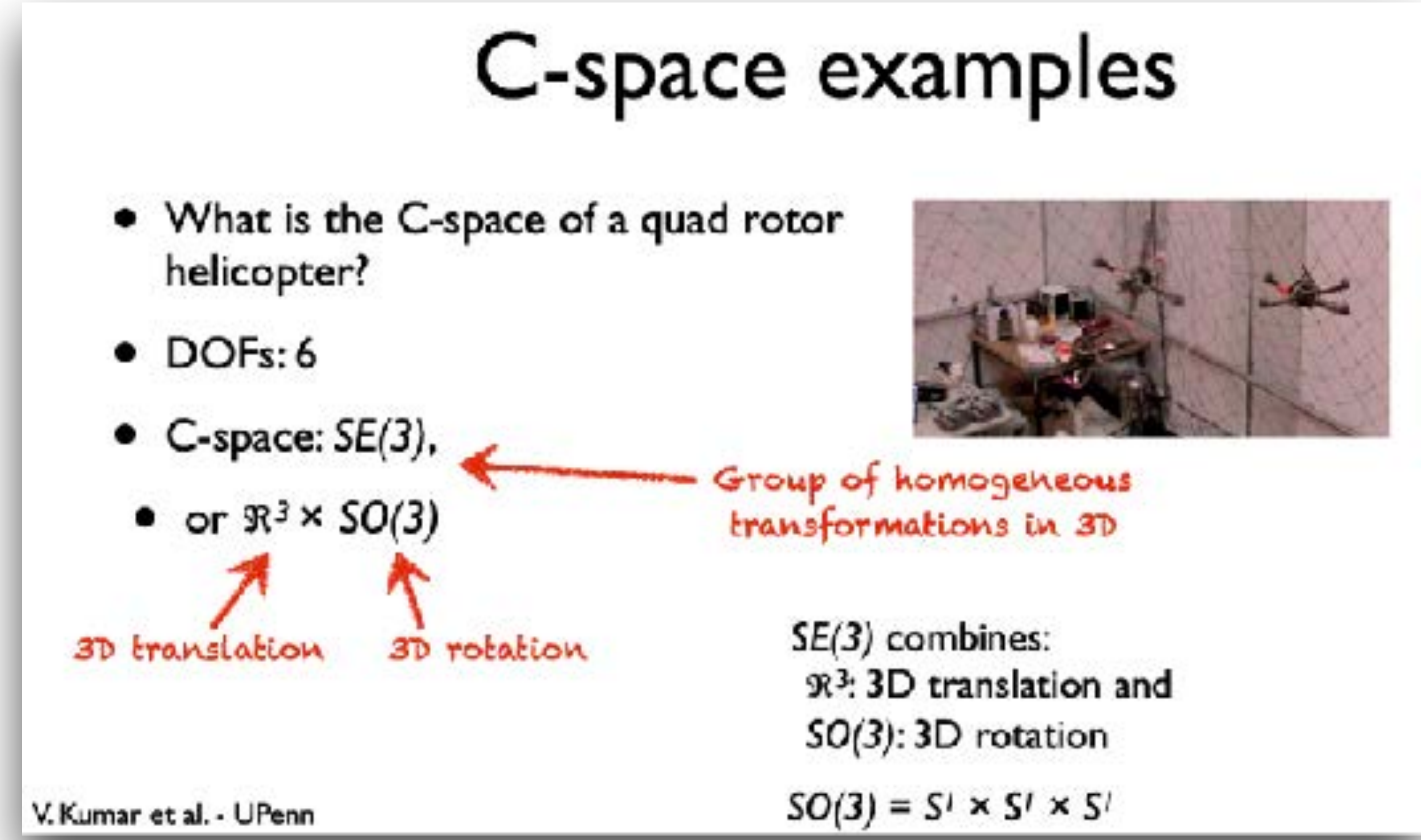
Previously

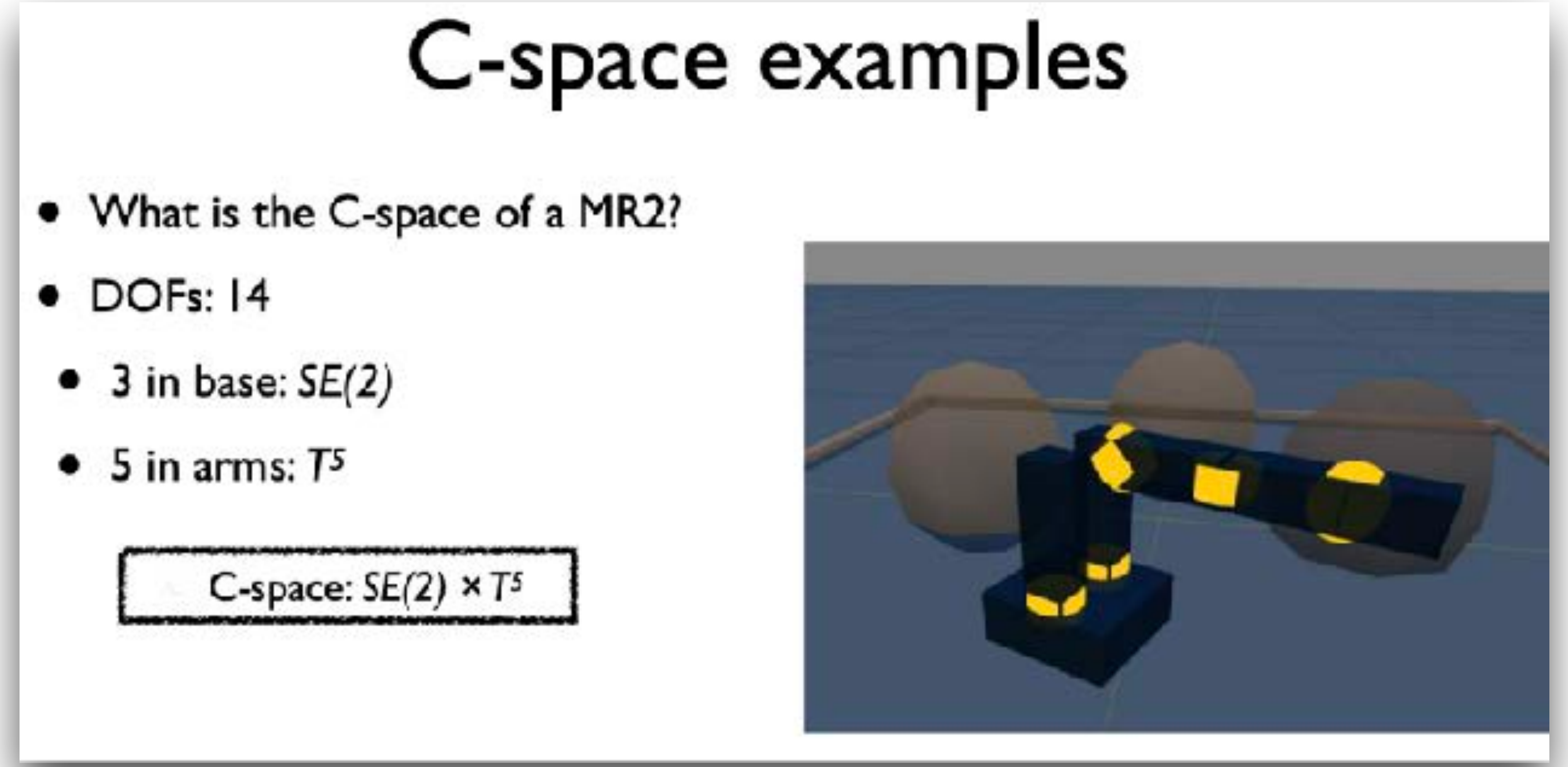

Will our current search methods apply to this robot?
2D Path Planning
N-dimensional Motion Planning


C-space examples
How many configurations are in the C-space of a planar point robot in a bounded rectangular world?
DOFs: 2, $\{x_1, x_2\}$
Number of poses is infinite
C-space: $\Re^2$
Toplogically, this C-space is a homeomorphism of $\Re^2$
q_{goal}
x_1
x_2
q_{start}

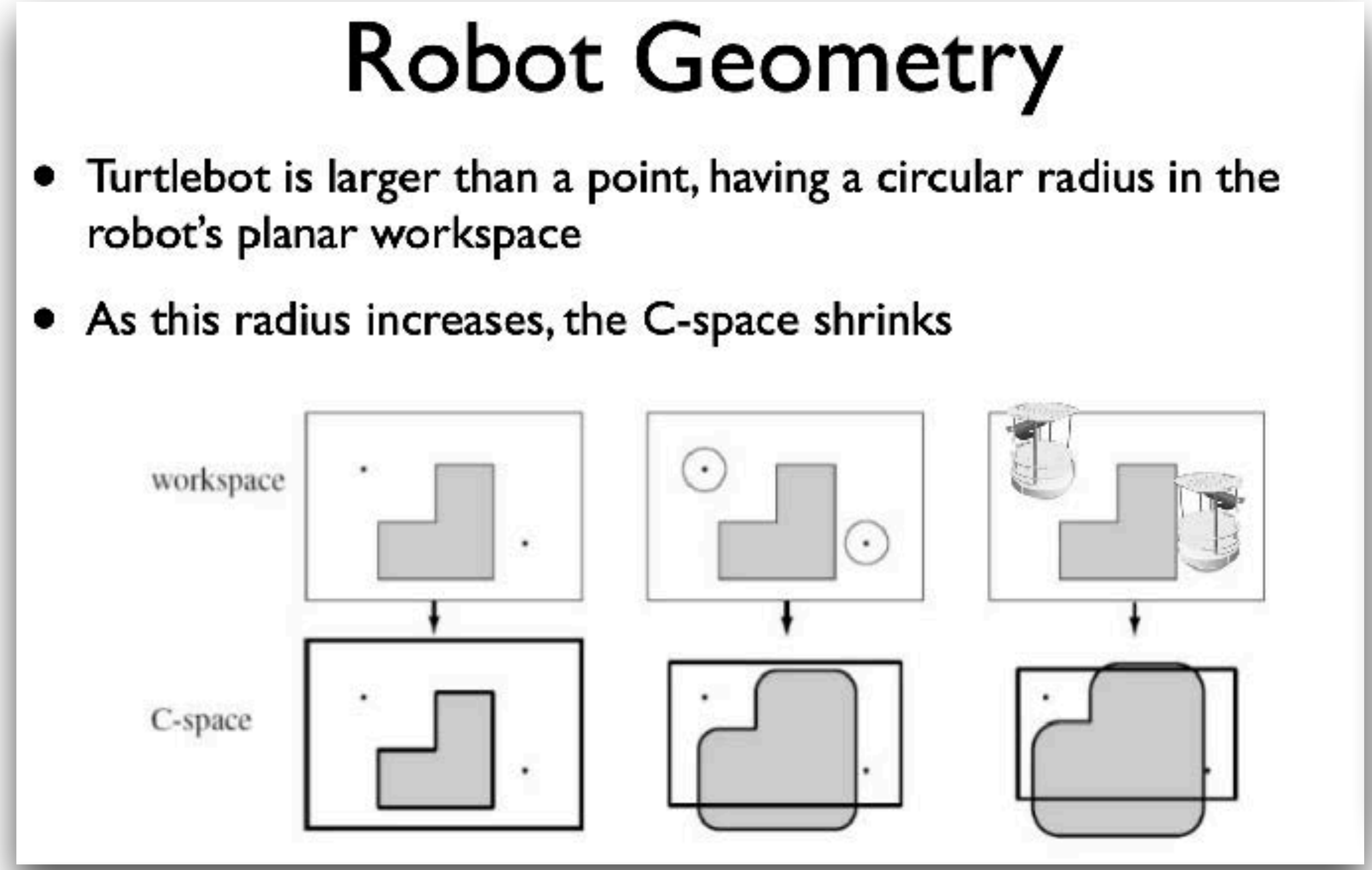

C-space examples
What is the C-space of a Turtlebot?
DOFs: 3, $\{x_1, x_2, \Theta\}$
C-space: $\Re^2 \times S^1$
2D translation
rotation in 2D
$\Re^2 \times S^1$ is also known as the $SE(2)$ group.
Group of homogeneous transformations in 2D
q_{goal}
x_1
Θ
x_2
q_{start}

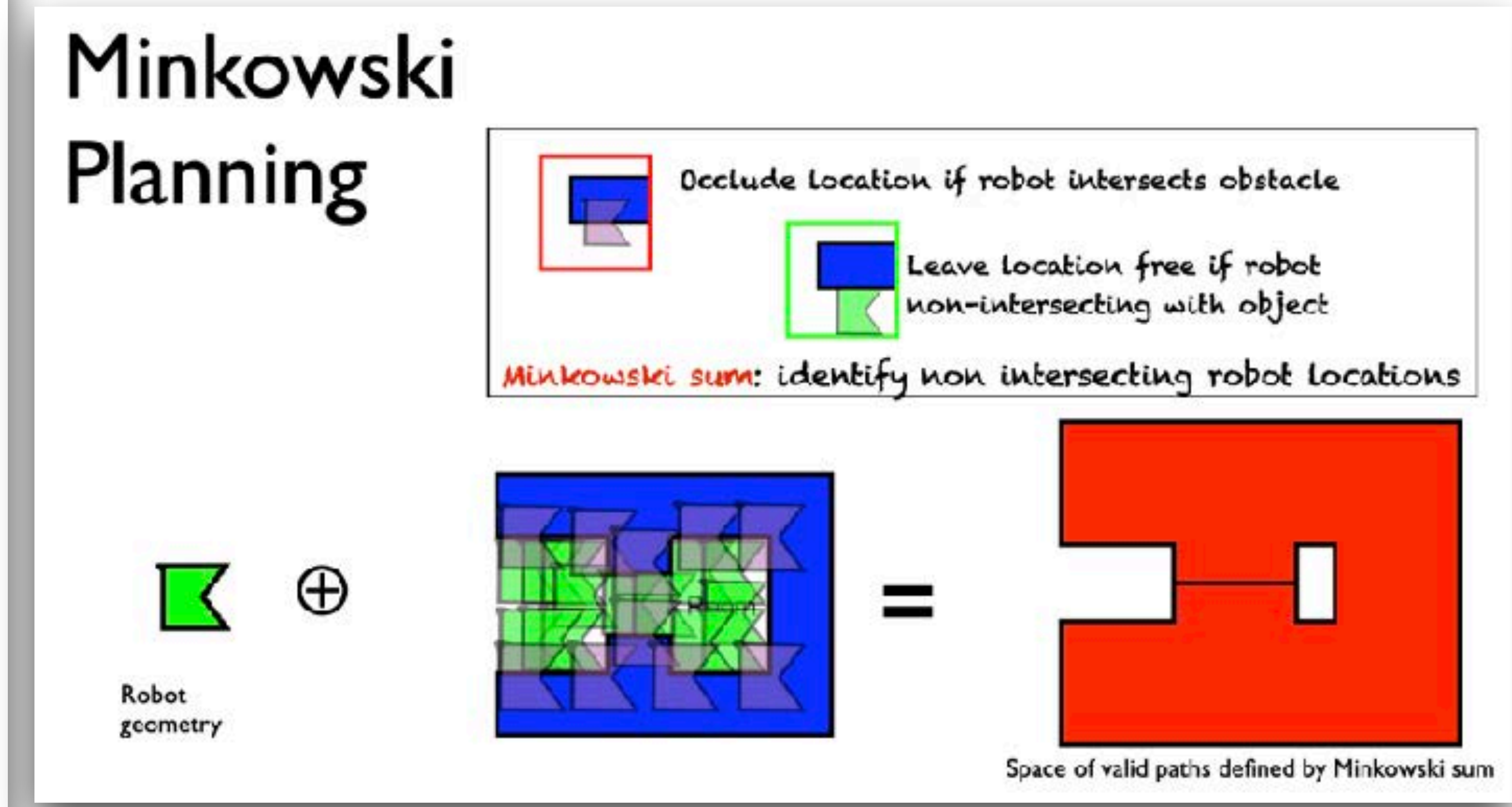

C-space examples
What is the C-space of a planar arm with 2 rotational joints?
DOFs: 2, $\{\Theta_1, \Theta_2\}$
C-space: $\Re^2$ or S^2 or $S^1 \times S^1$?
$S^1 \times S^1 = S^2$ when torus axis on surface
End effector
Link 2
Link 1
θ_2
θ_1

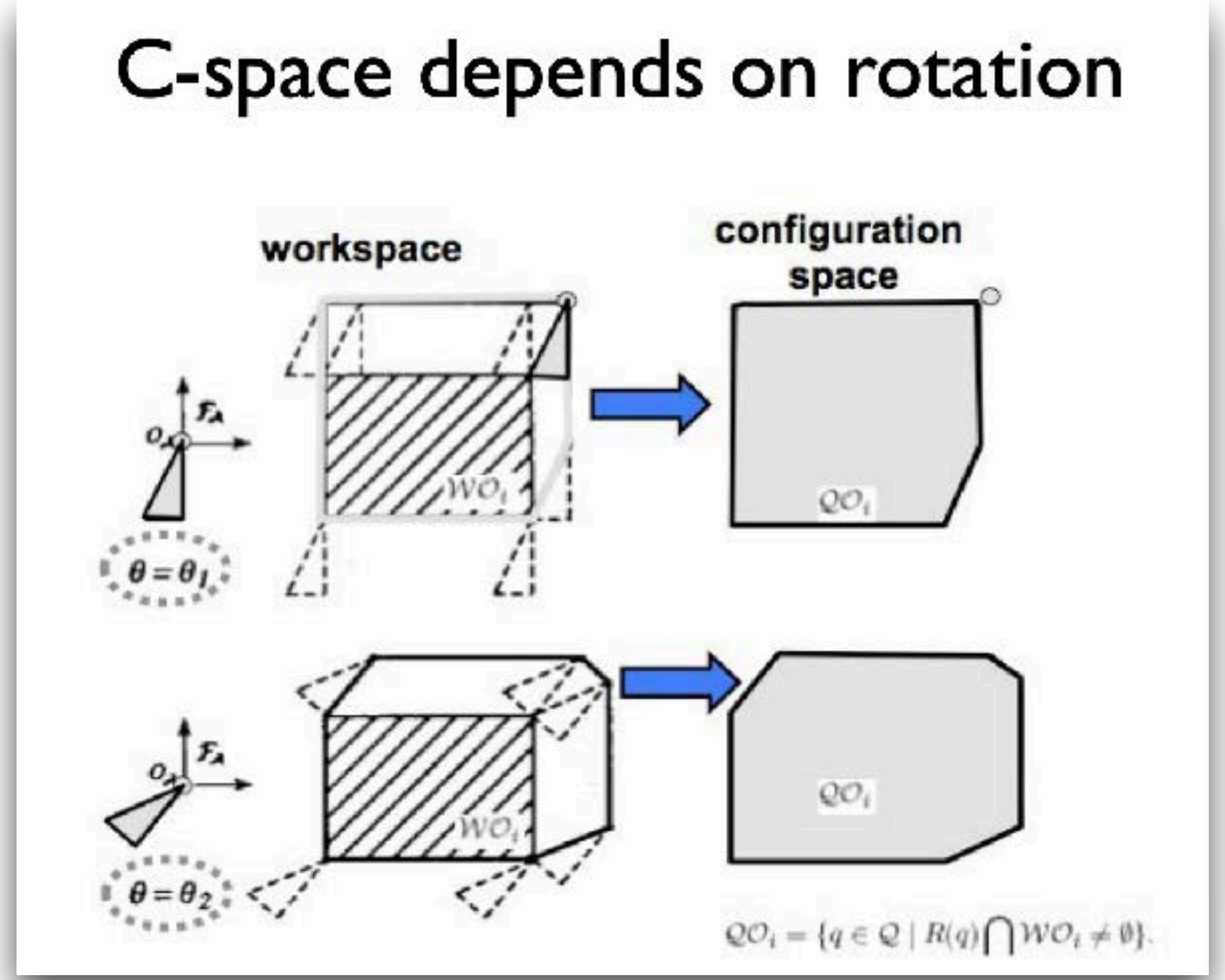

C-space examples
What is the C-space of a quad rotor helicopter?
DOFs: 6
C-space: $SE(3)$,
or $\Re^3 \times SO(3)$
Group of homogeneous transformations in 3D
3D translation
3D rotation
$SE(3)$ combines:
$\Re^3$: 3D translation and
$SO(3)$: 3D rotation
$SO(3) = S^1 \times S^1 \times S^1$
V. Kumar et al. - UPenn


C-space examples
What is the C-space of a MR2?
DOFs: 14
3 in base: $SE(2)$
5 in arms: T^5
C-space: $SE(2) \times T^5$

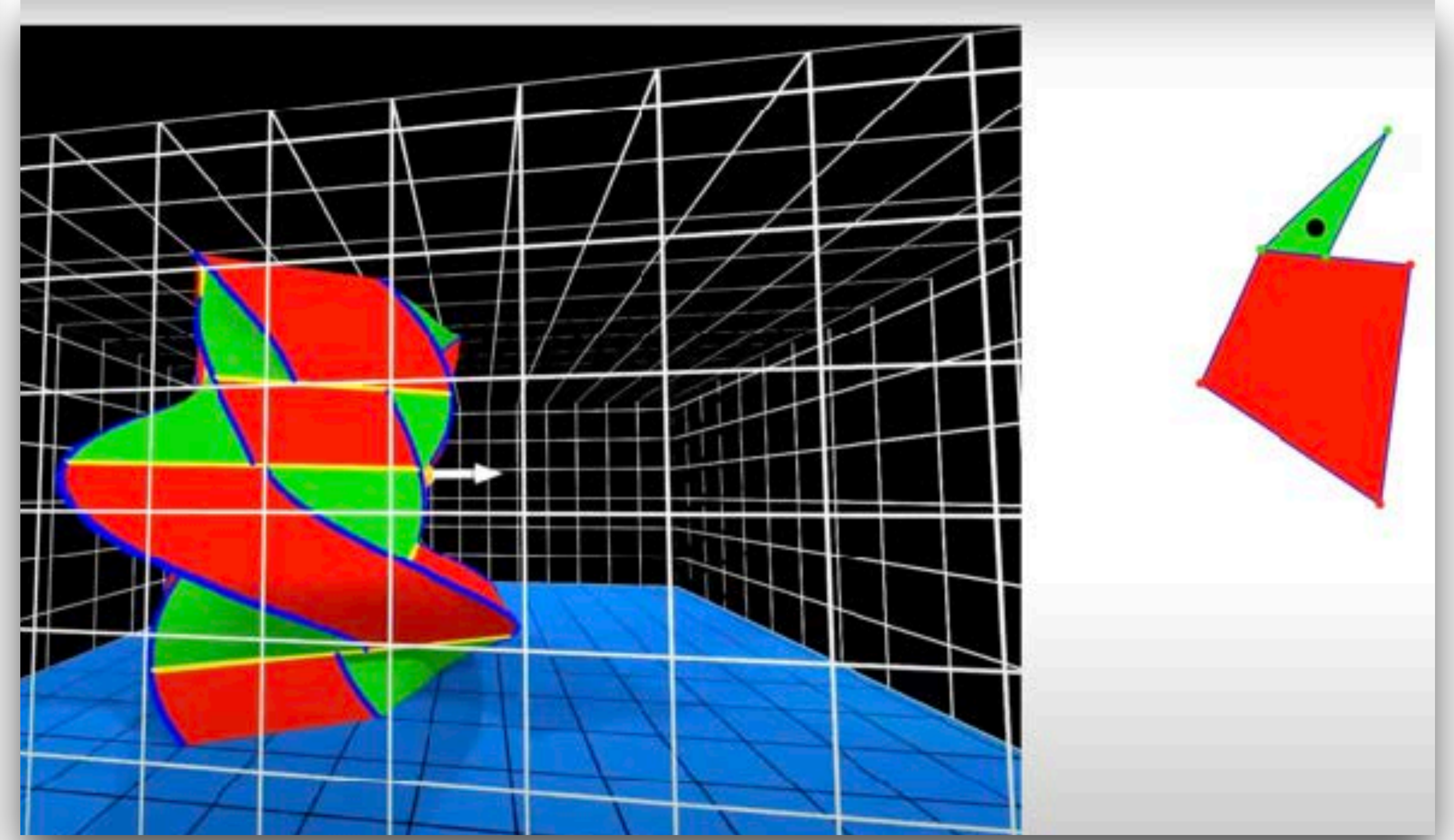
Previously

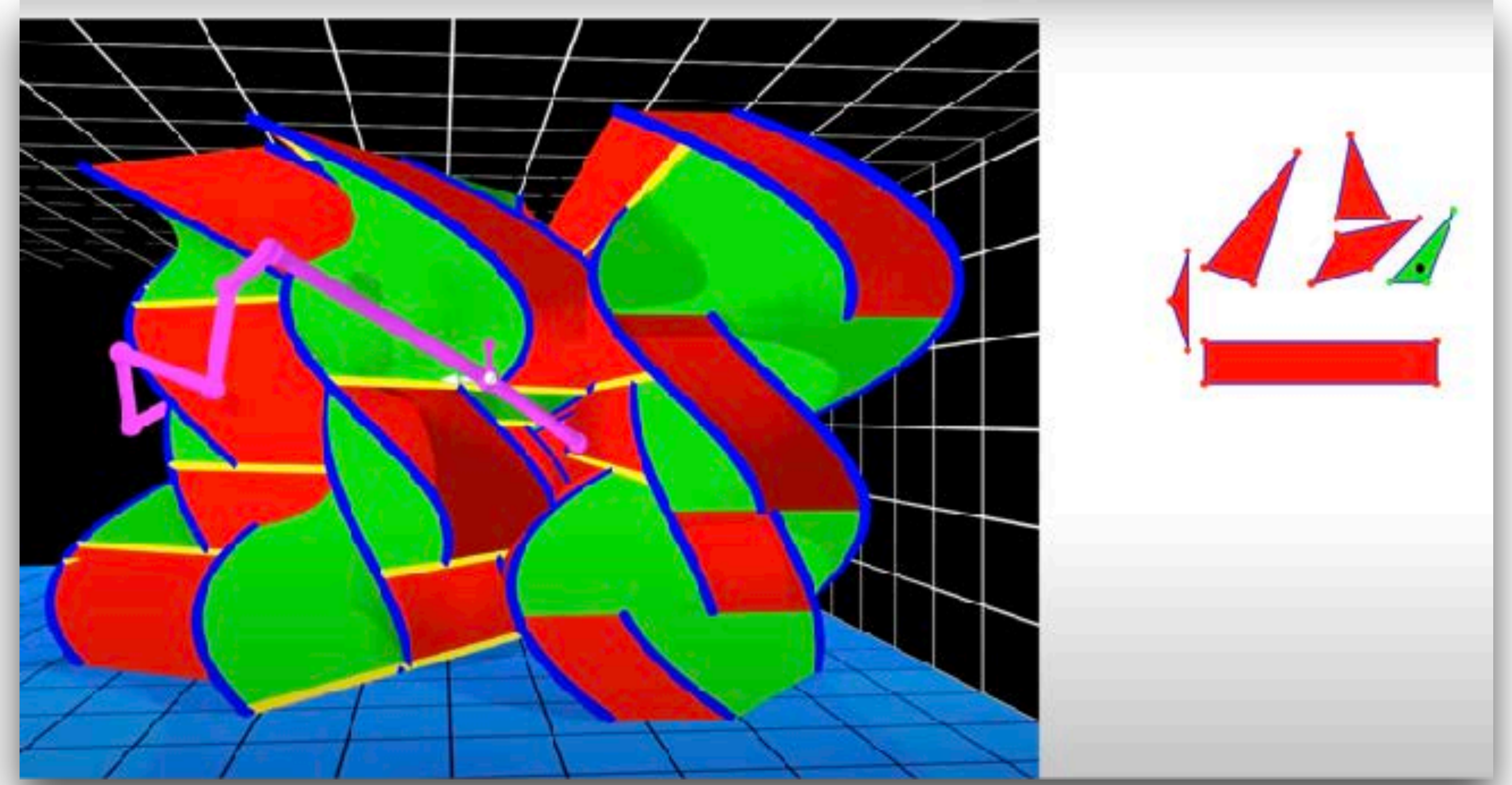






Robot is a point in the C-space!



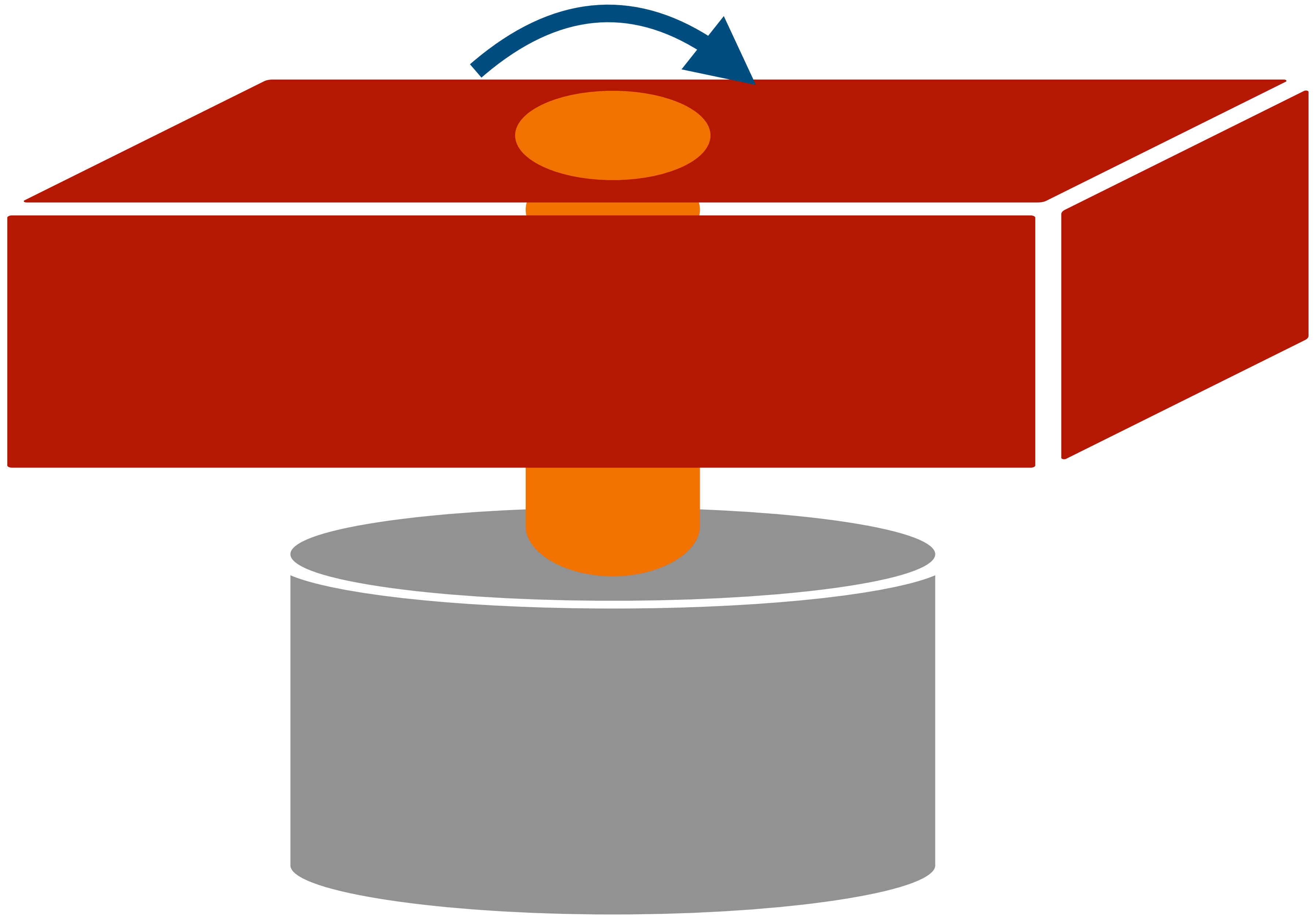


DOF formally:

$$\text{dof} = \sum \text{freedoms of rigid bodies} \; - \text{\# of independent constraints}$$

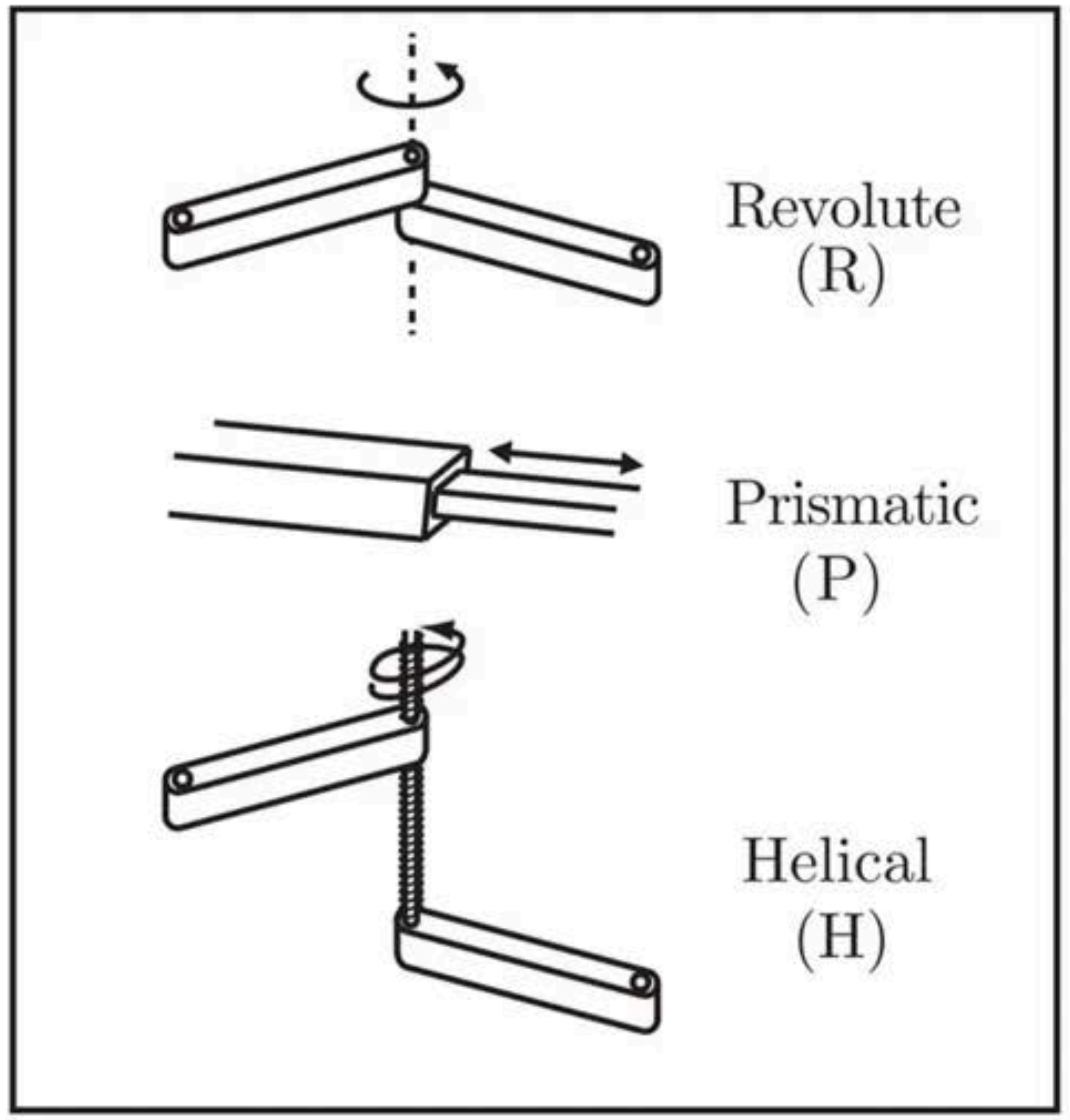
often comes from joints

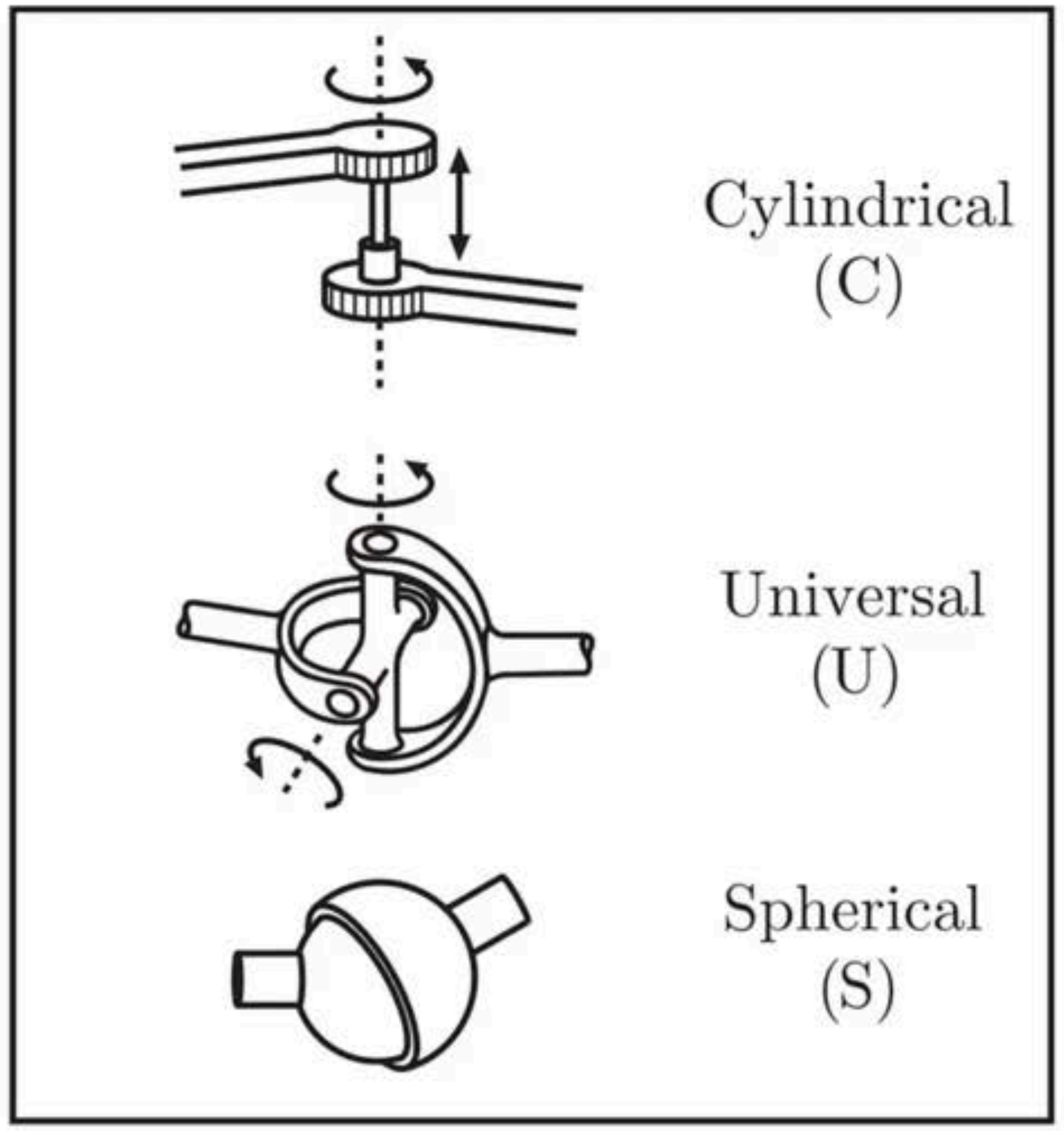
What is the degrees of freedom for this rigid body?



If we add a revolute joint, what happens to the degrees of freedom of this system?

Joints & Constraints





Joint type	dof f
Revolute (R)	1
Prismatic (P)	1
Helical (H)	1
Cylindrical (C)	2
Universal (U)	2
Spherical (S)	3

From the book MODERN ROBOTICS
by Kevin M. Lynch and Frank C. Park May 3, 2017

DOF formally

$$\text{dof} = \sum \text{freedoms of rigid bodies} - \#\text{ of independent constraints}$$

$N = \#$ of bodies, including the ground

$J = \#$ of joints

$m = 6$ for spatial bodies; 3 for planar bodies

$$\text{dof} = \underbrace{m(N-1)}_{\text{Rigid body freedoms}} - \underbrace{\sum_{i=1}^{J} c_i}_{\text{Joint Constraints}}$$

Rigid body freedoms

Joint Constraints

$$\text{dof} = m(N-1) - \sum_{i=1}^{J} (m - f_i)$$

Only applicable if the constraints provided by the joints are independent →

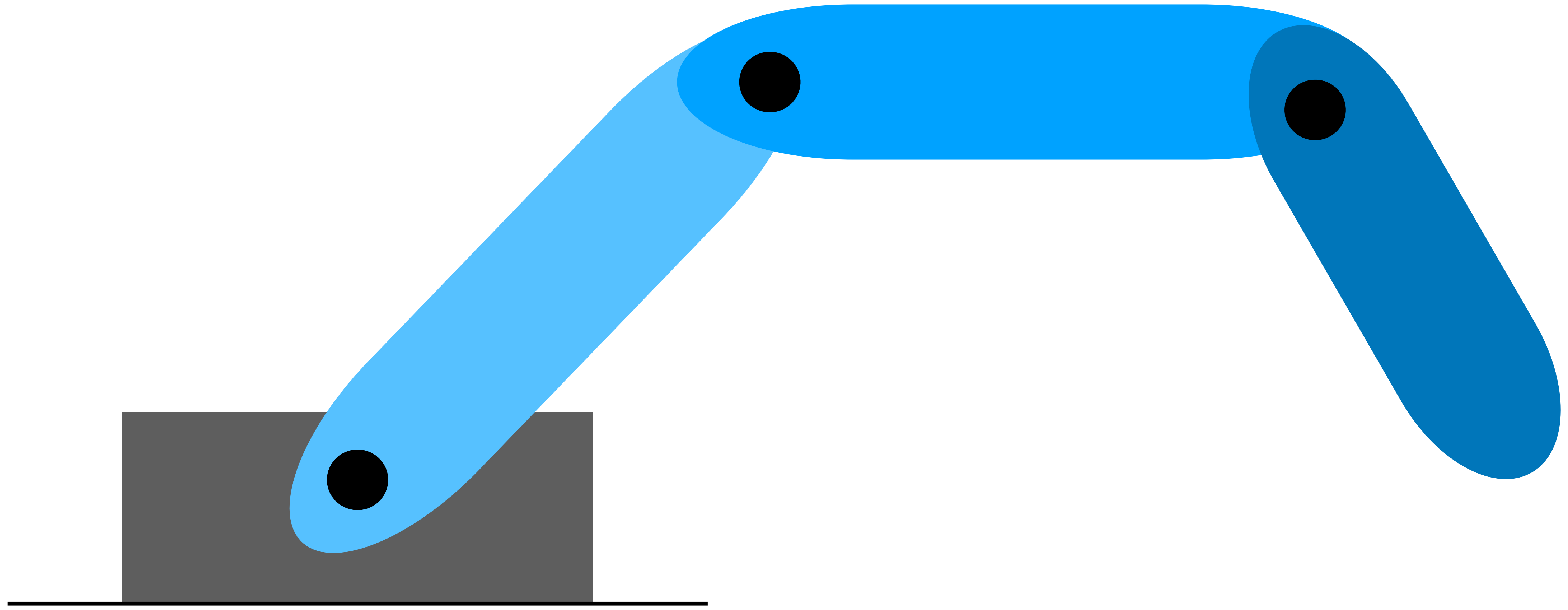
$$\text{dof} = m(N-J-1) + \sum_{i=1}^{J} (f_i)$$

Grübler's formula

DOF formally: Example 1

$$\text{dof} = m(N - J - 1) + \sum_{i=1}^{J} (f_i)$$

Joint type	dof f	Constraints c between two planar rigid bodies	Constraints c between two spatial rigid bodies
Revolute (R)	1	2	5
Prismatic (P)	1	2	5
Helical (H)	1	N/A	5
Cylindrical (C)	2	N/A	4
Universal (U)	2	N/A	4
Spherical (S)	3	N/A	3



3R Serial “open-chain” Robot

$m =$ [redacted]

$J =$ [redacted]

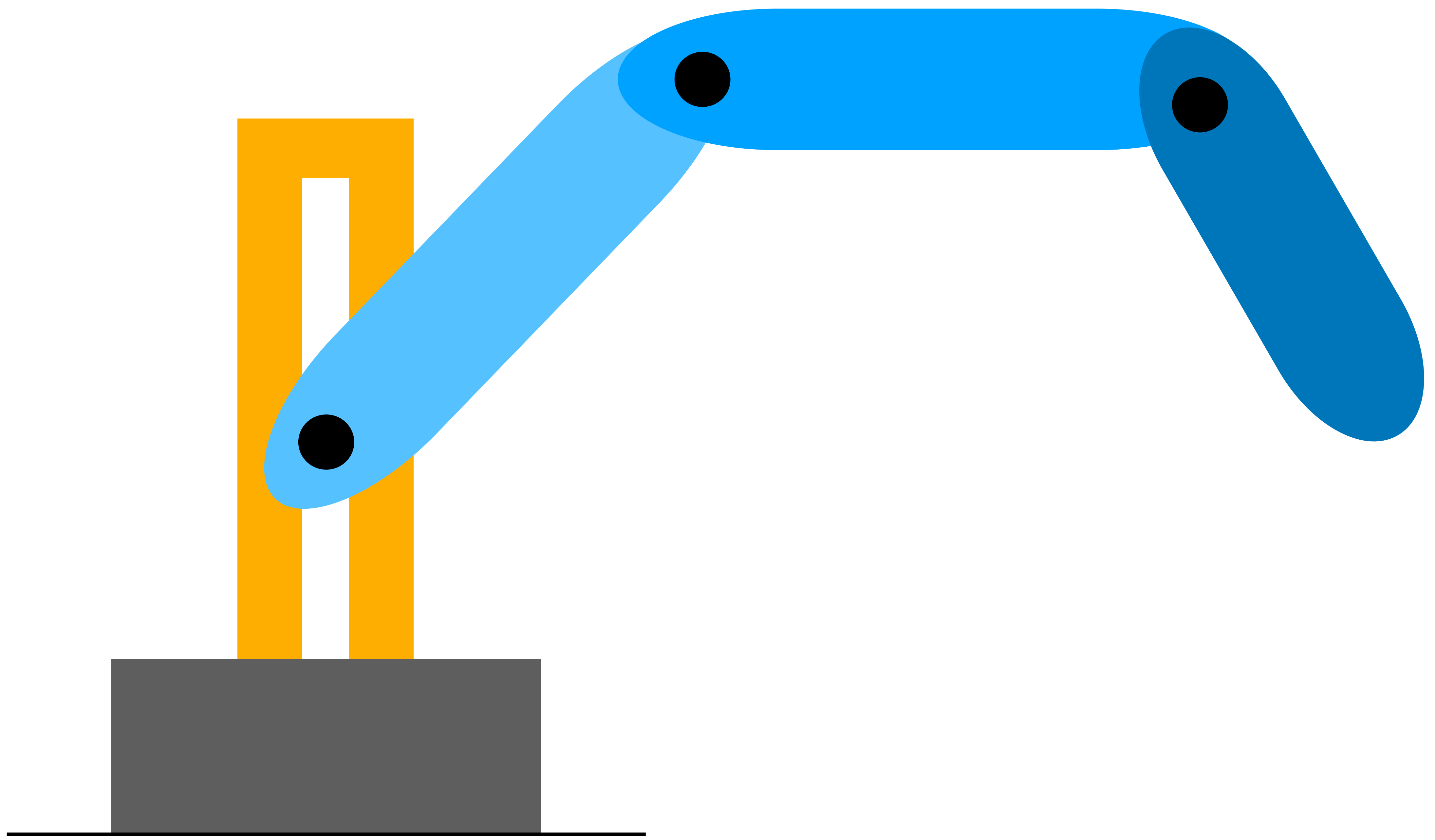
$N =$ [redacted]

dof = [redacted]

DOF formally: Example 2

$$\text{dof} = m(N - J - 1) + \sum_{i=1}^{J} (f_i)$$

Joint type	dof f	Constraints c between two planar rigid bodies	Constraints c between two spatial rigid bodies
Revolute (R)	1	2	5
Prismatic (P)	1	2	5
Helical (H)	1	N/A	5
Cylindrical (C)	2	N/A	4
Universal (U)	2	N/A	4
Spherical (S)	3	N/A	3



$$m = 3$$
$$J = 3$$
$$N = 4$$

$$\text{dof} = 3(4 - 3 - 1) + 3 = 3$$

DOF formally

$$\text{dof} = \sum \text{freedoms of rigid bodies} - \text{\# of independent constraints}$$

N = # of bodies, including the ground

J = # of joints

$m = 6$ for spatial bodies; 3 for planar bodies

$$\text{dof} = \underbrace{m(N-1)}_{\text{Rigid body freedoms}} - \underbrace{\sum_{i=1}^{J} c_i}_{\text{Joint Constraints}}$$

Rigid body freedoms

Joint Constraints

$$\text{dof} = m(N-1) - \sum_{i=1}^{J} (m - f_i)$$

??? Only applicable if the constraints provided by the joints are independent →

$$\text{dof} = m(N-J-1) + \sum_{i=1}^{J} (f_i)$$

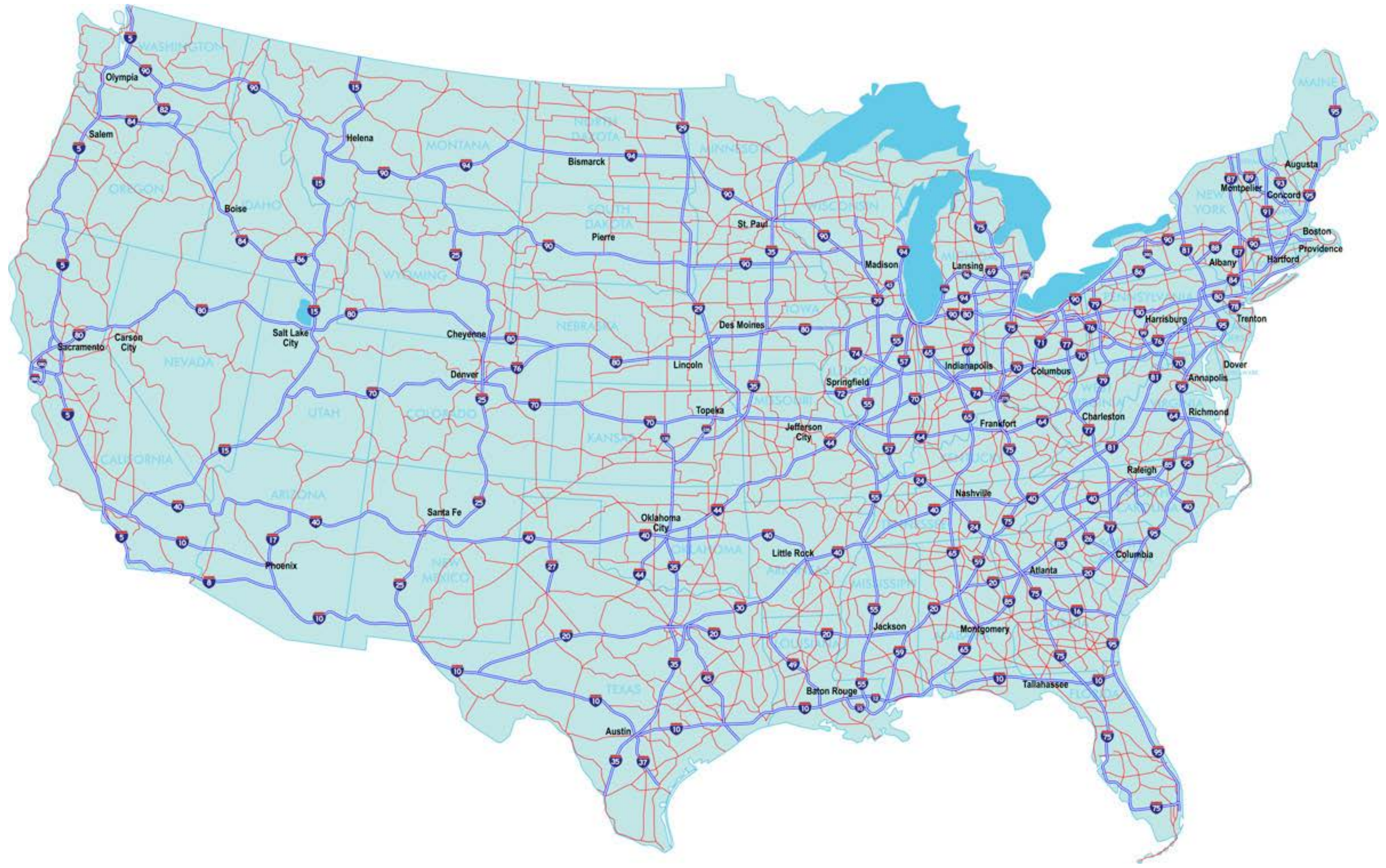
Grübler's formula

Sampling-based Planning

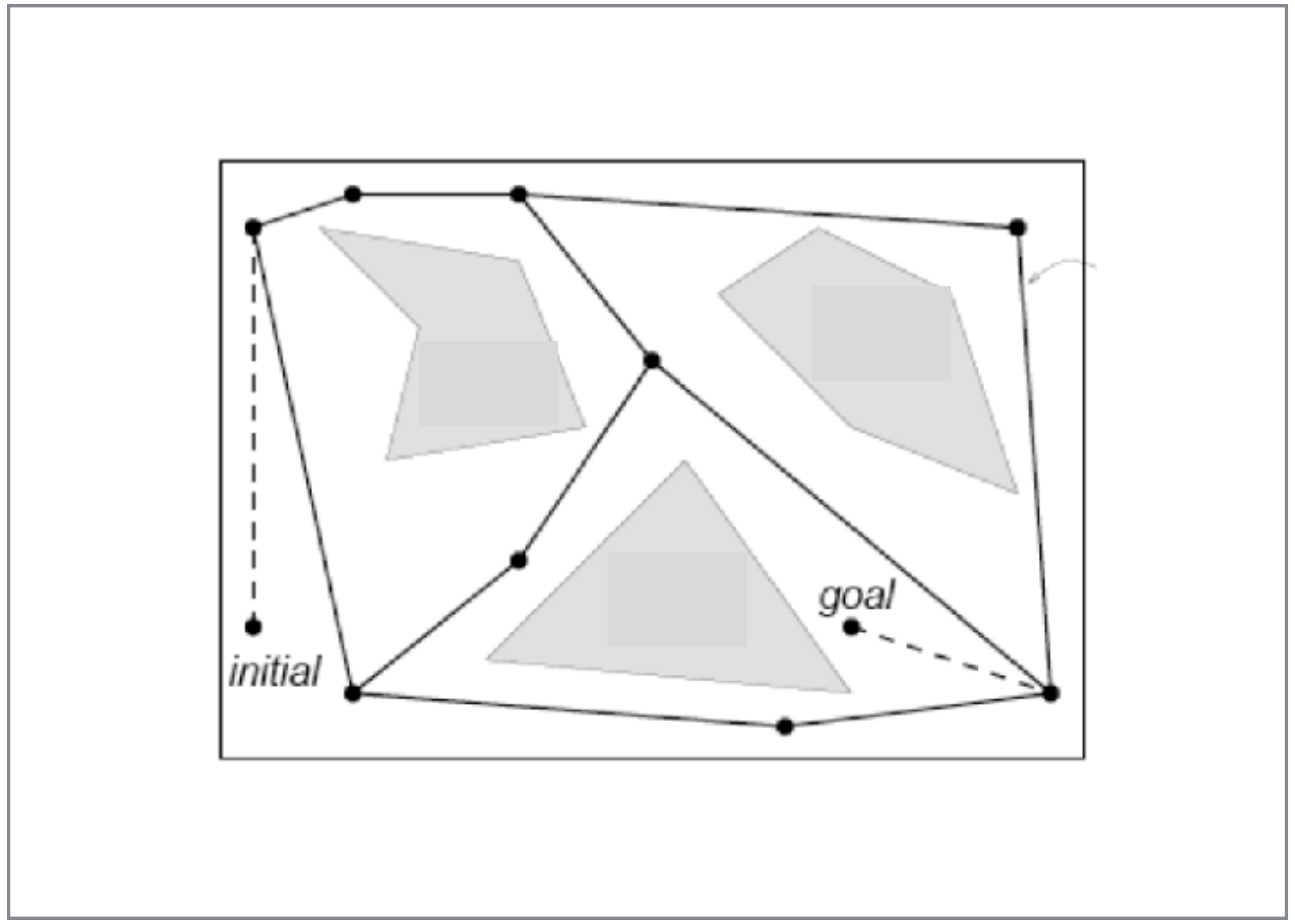
Approaches to motion planning

- Bug algorithms: Bug[0-2], Tangent Bug
- Graph Search (fixed graph)
  - Depth-first, Breadth-first, Dijkstra, A-star, Greedy best-first
- **Sampling-based Search (build graph):**
  - **Probabilistic Road Maps, Rapidly-exploring Random Trees**
- Optimization and local search:
  - Gradient descent, Potential fields, Simulated annealing, Wavefront

Roadmaps

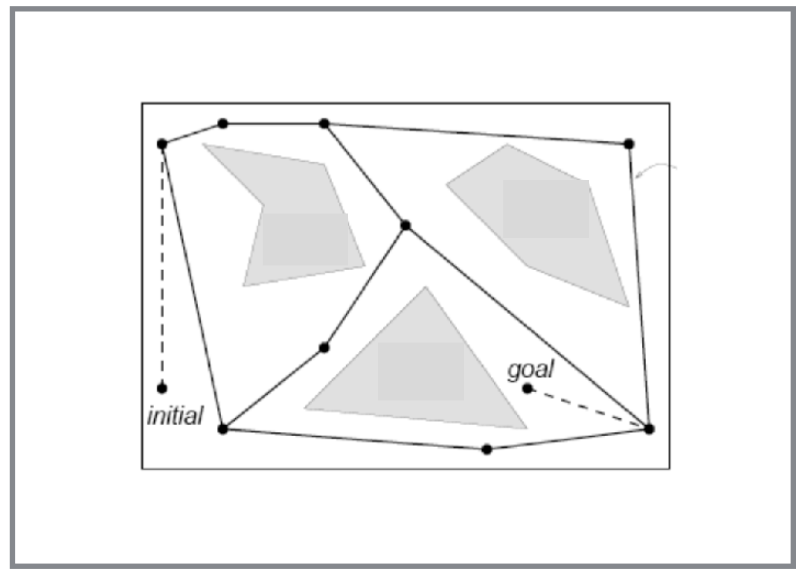


Roadmap over geolocations

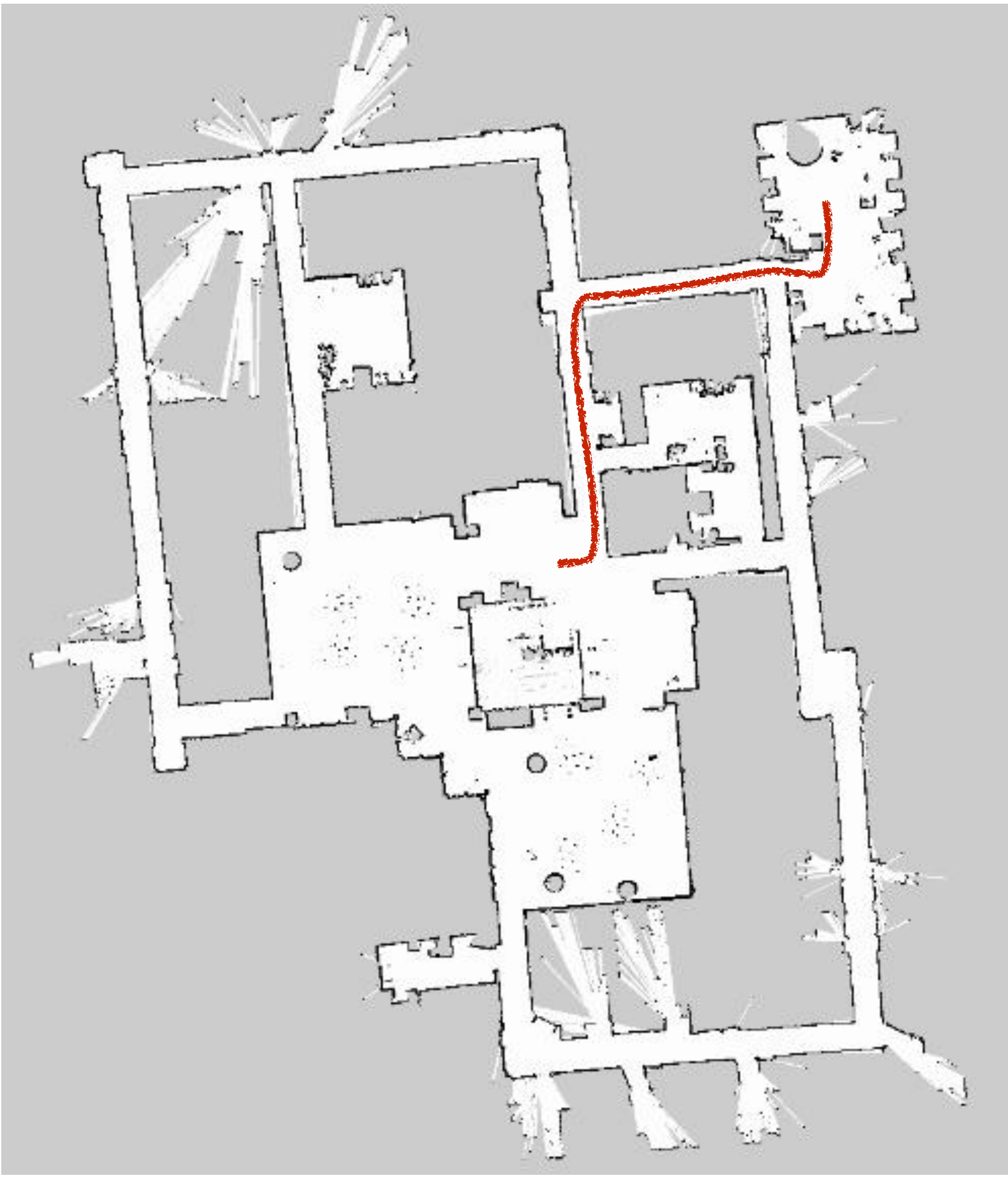


Roadmap over robot configurations

Roadmaps



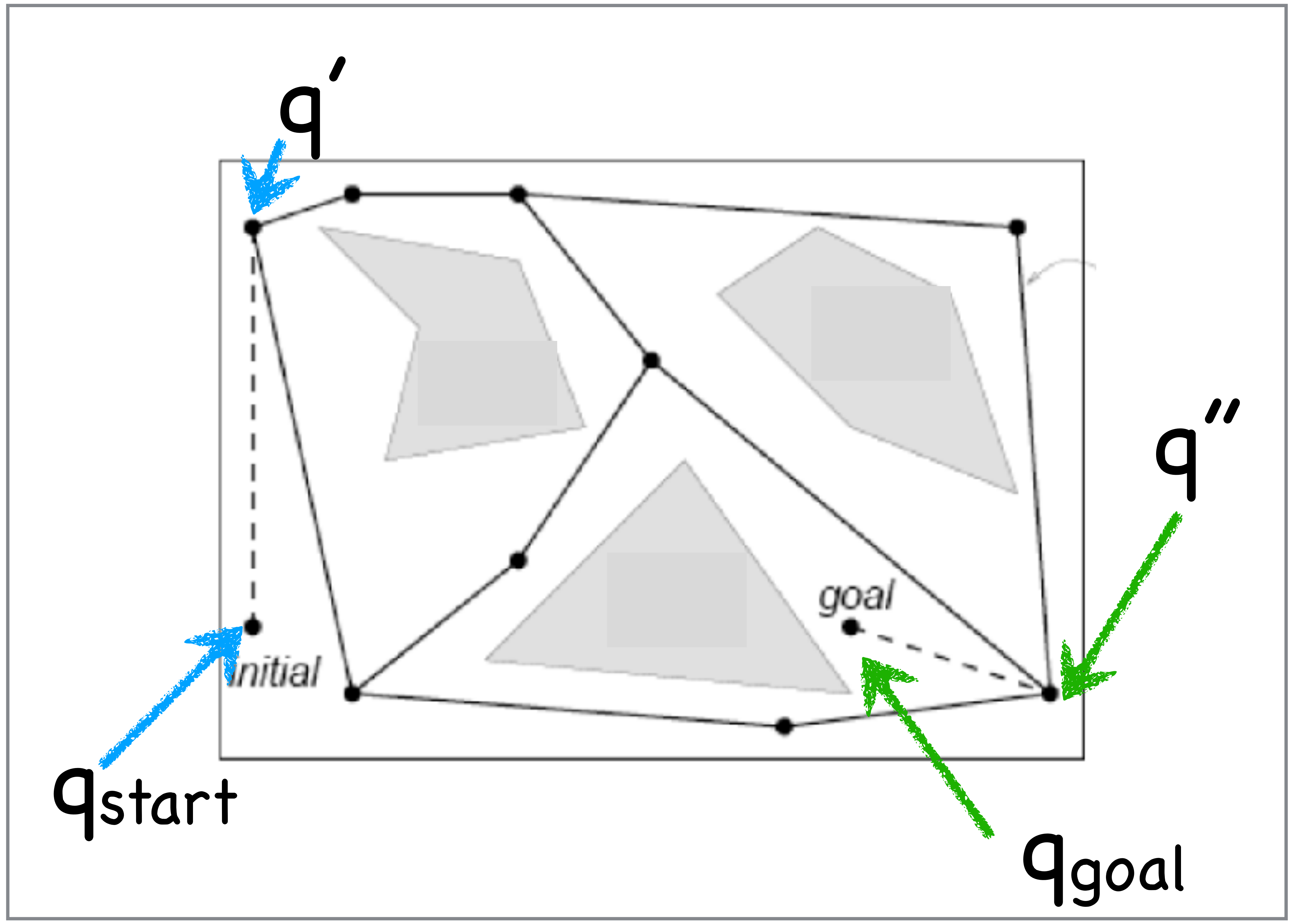
- Graph search assumed C-space as a fixed uniform grid
    - finite set of discretized cells
- How does this scale beyond planar navigation?
    - curse of dimensionality
- Roadmaps are a more general notion of graphs in C-space



Slide borrowed from Michigan Robotics autorob.org

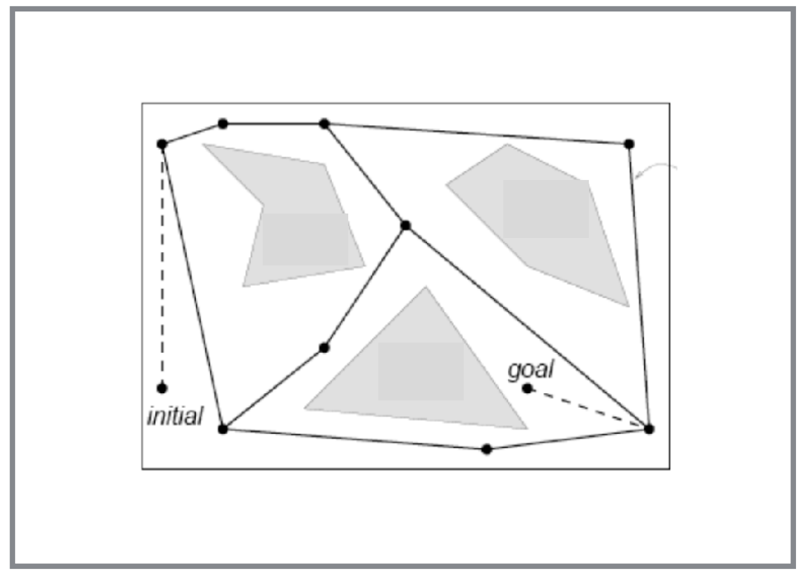
Roadmap Definition

- A roadmap *RM* is a union of curves s.t. all start and goal points in C-space (Q_{free}) can be connected by a path



- Roadmap properties:
  - **Accessibility**: There is a path from $q_{start} \in Q_{free}$ to some $q' \in RM$
  - **Departability**: There is a path from $q'' \in RM$ to $q_{goal} \in Q_{free}$
  - **Connectivity**: there exists a path in *RM* between q' and q''

Basic Roadmap Planner



1) **Build** the roadmap *RM* as graph *G(V,E)*
   - *V*: nodes are "valid" in C-space in Q_{free}
     - a configuration q is valid if it is not in collision and within joint limits
   - *E*: an edge $e(q_1,q_2)$ connects two nodes if a free path connects q_1 and q_2
     - all configurations along edge assumed to be valid
2) **Connect** start and goal configurations to *RM* at q' and q'', respectively
3) **Find** path in *RM* between q' and q''

How to build a roadmap?

How to build a roadmap?

2 Approaches

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles

Probabilistic:

C-space sampling

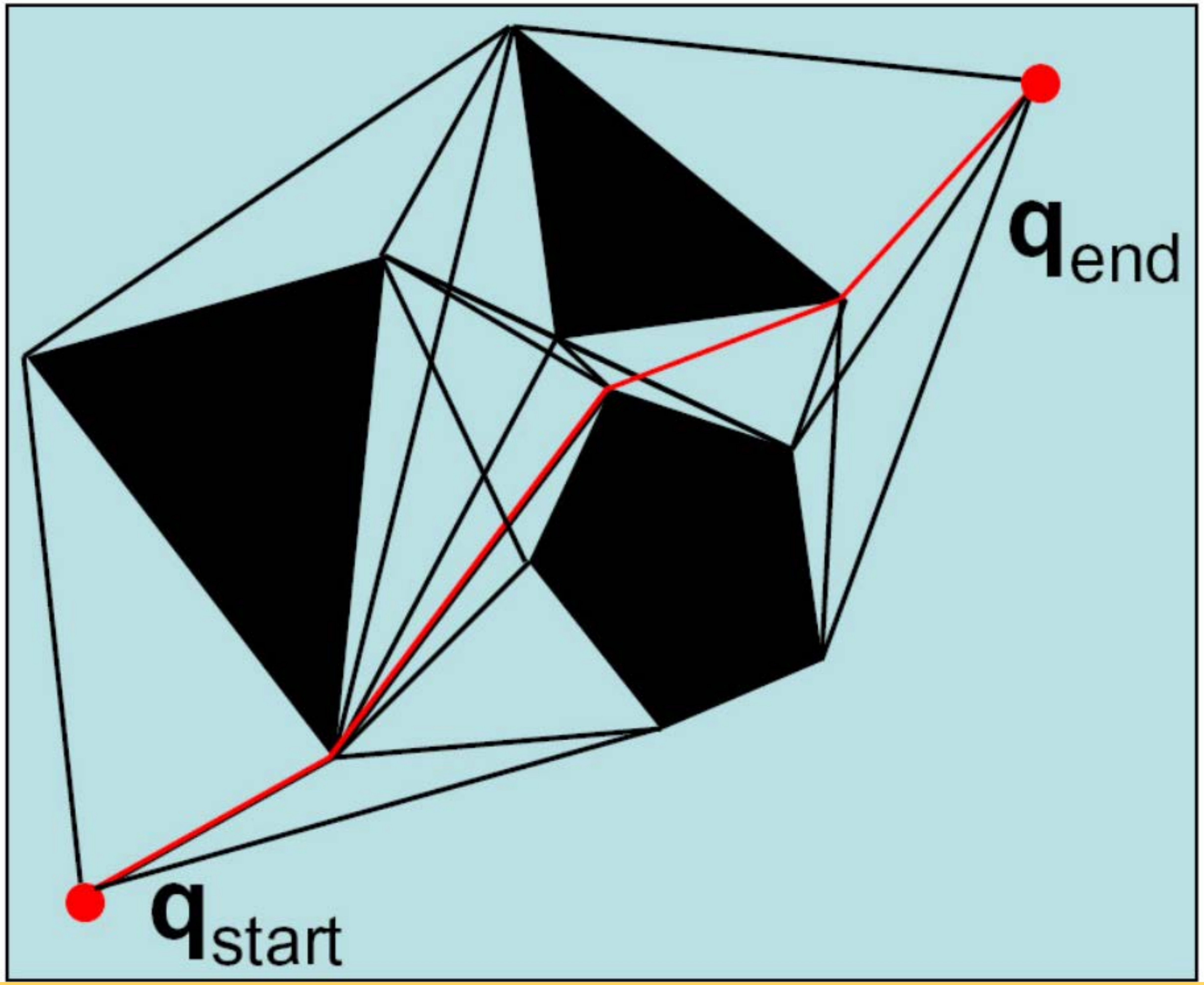
- Probabilistic Roadmap (PRM)
  - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
  - sample and connect vertices in trees rooted at start and goal configuration

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles



s

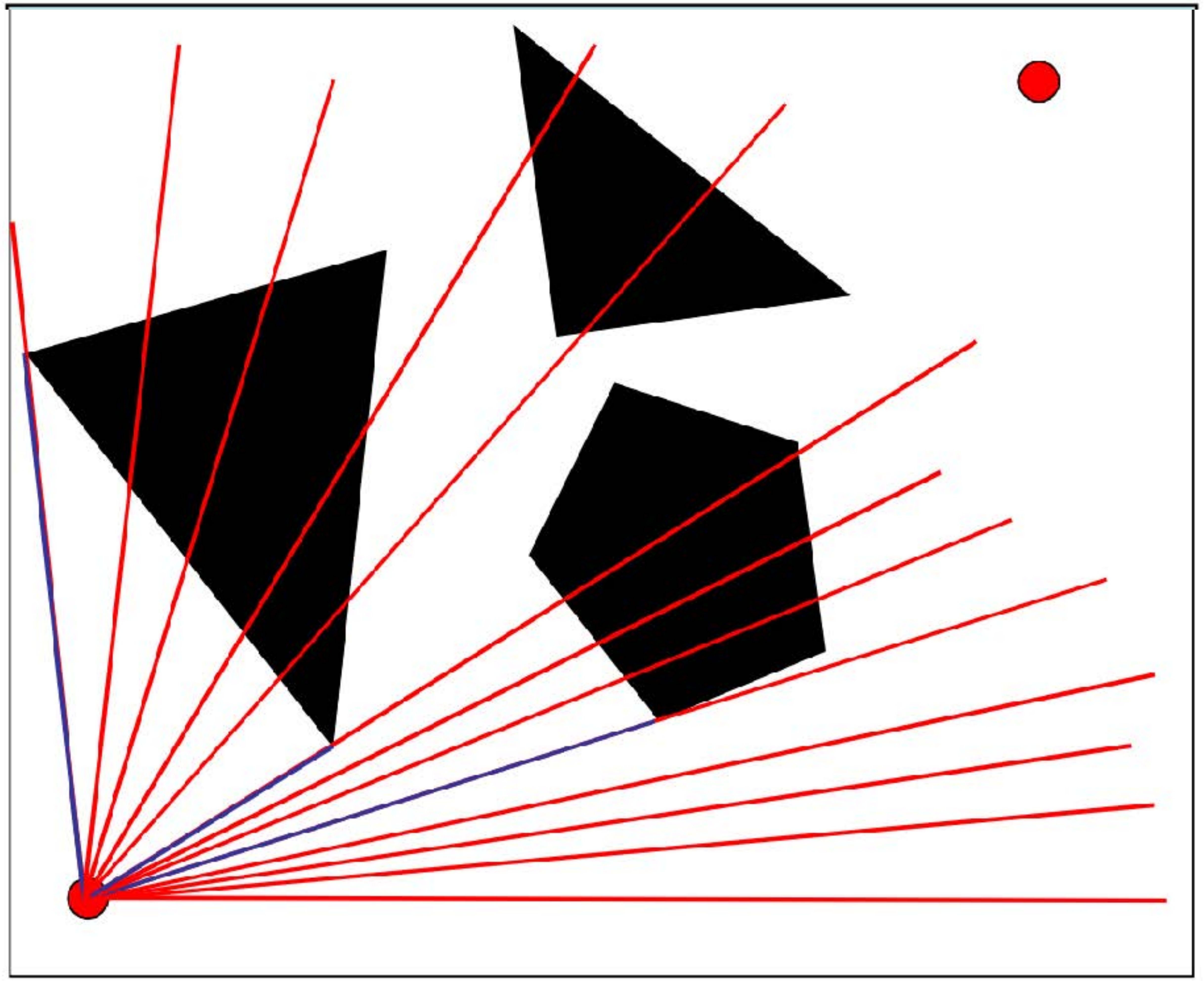
es
on

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles

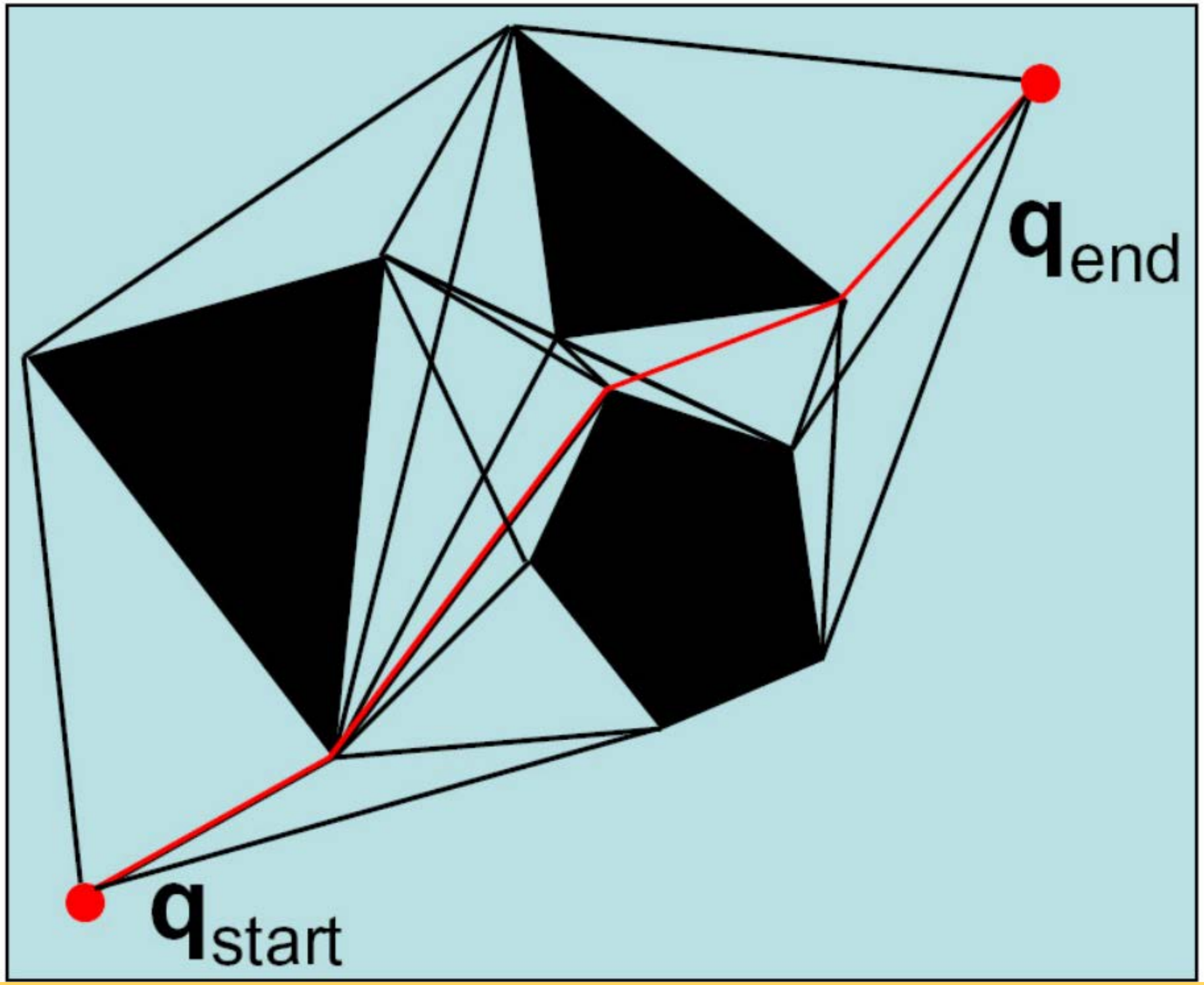


2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
    - trace lines connecting obstacle polygon vertices
- Voronoi Planning
    - trace edges equidistant from obstacles

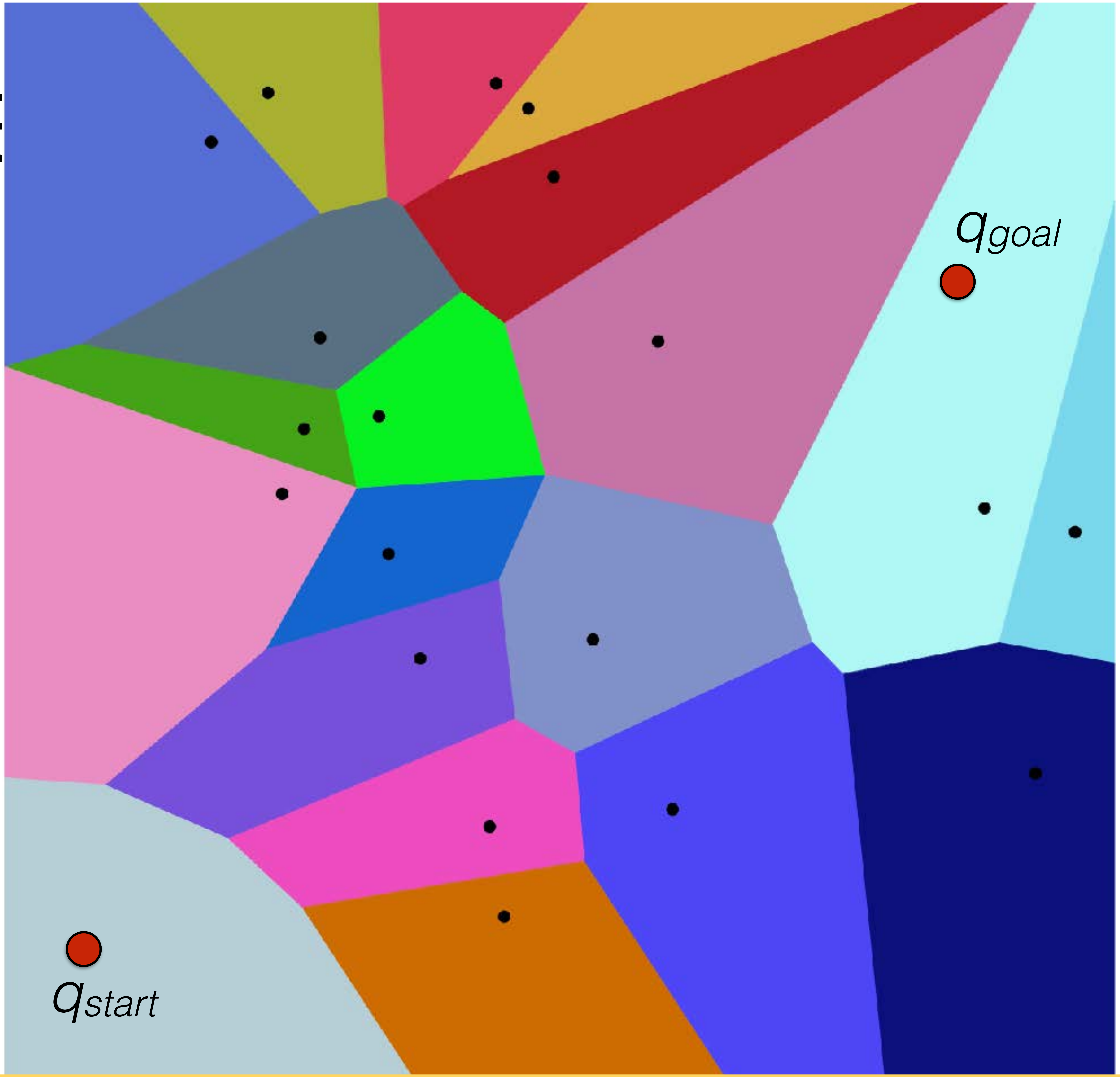


2 Approache

Deterministic:

complete algorithms

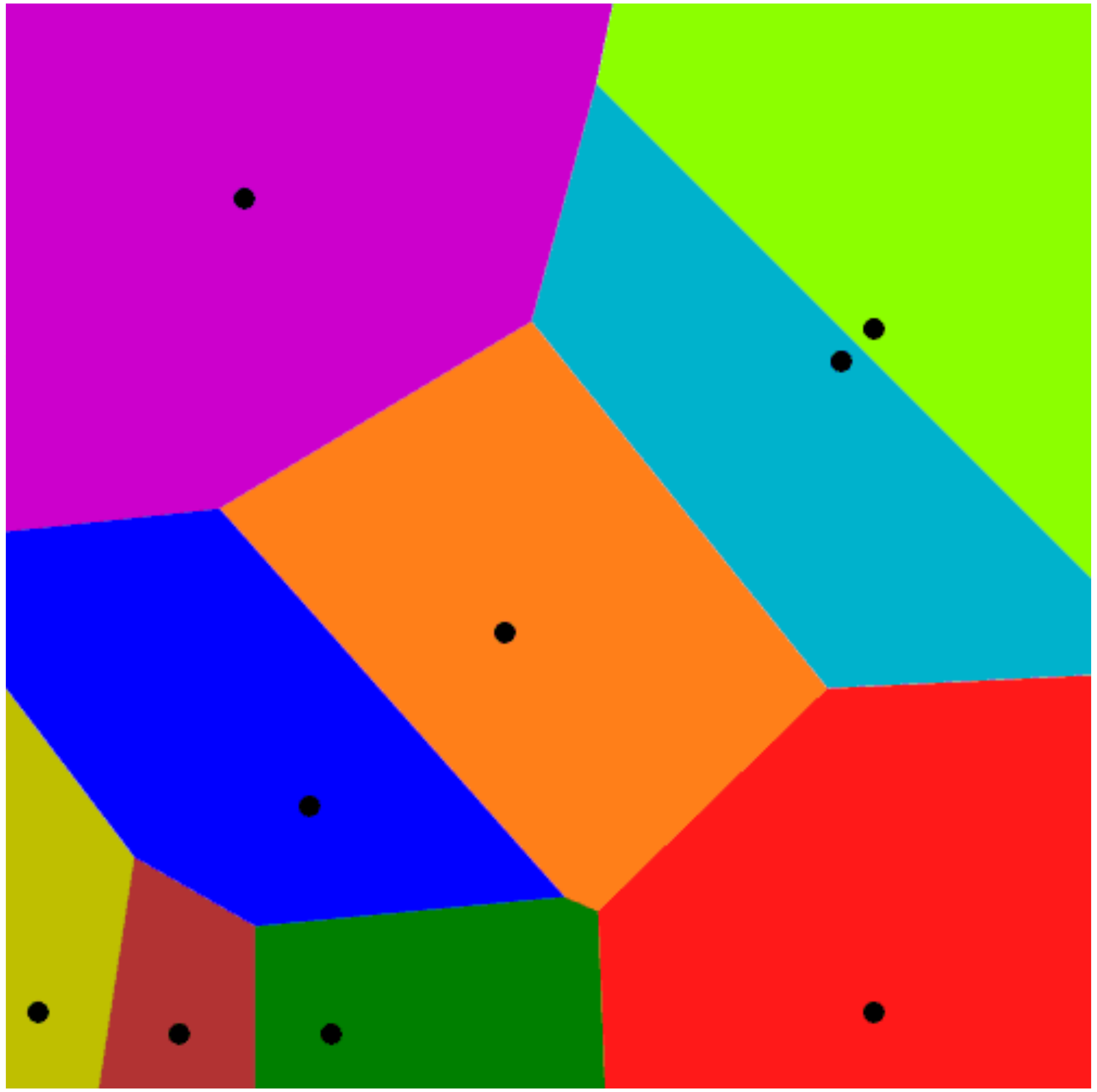
- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles



Slide borrowed from Michigan Robotics autorob.org

Voronoi Diagram

- Given *N* input points in a *d* dimensional space
- Find region boundaries such that each point on a boundary are equidistant to two or more input points
- Delaunay triangulation is a dual to the Voronoi diagram



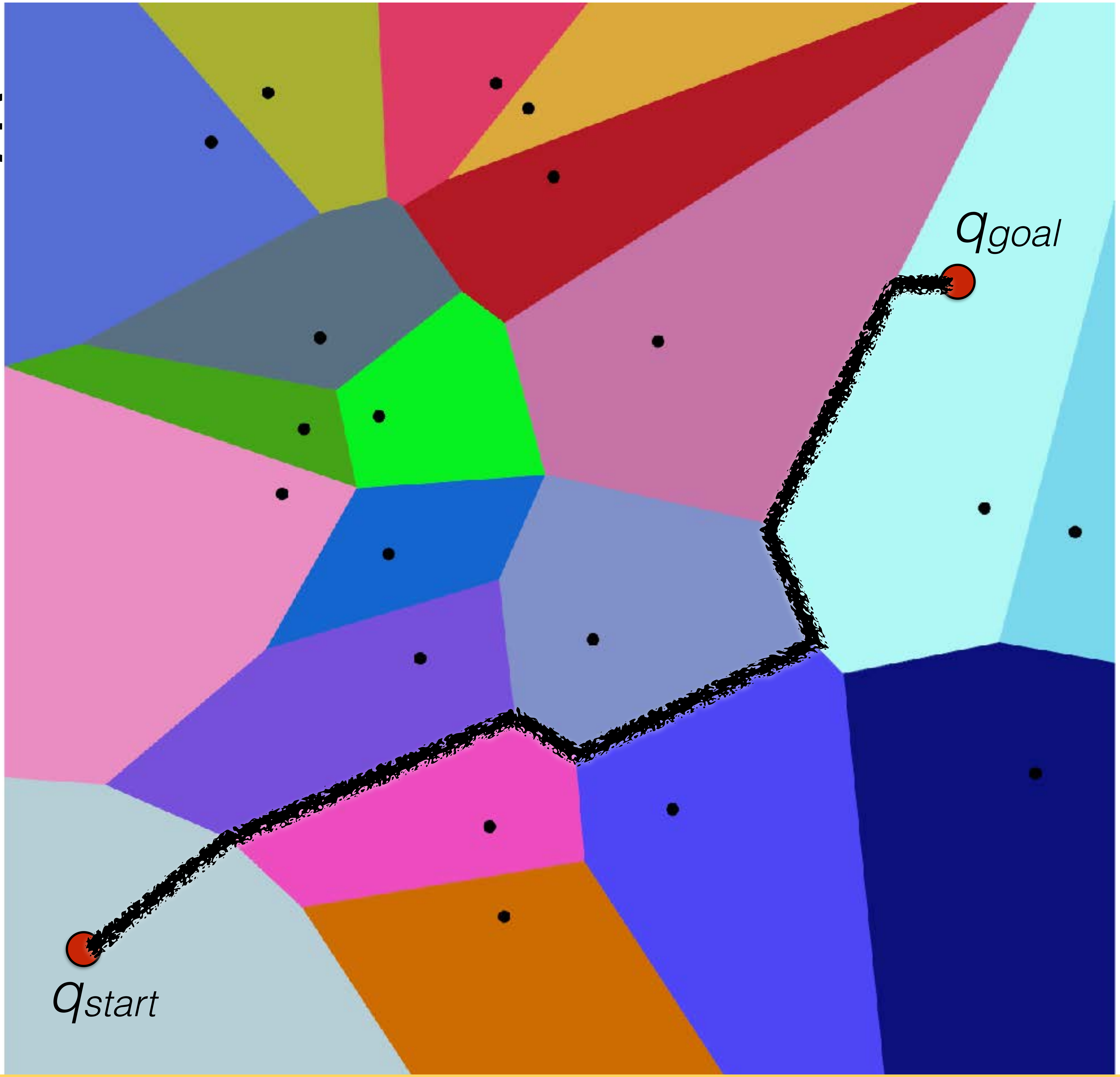
https://en.wikipedia.org/wiki/Voronoi_diagram#/media/File:Voronoi_growth_euclidean.gif

2 Approache

Deterministic:

complete algorithms

- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles



Slide borrowed from Michigan Robotics autorob.org

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
  - trace lines connecting obstacle polygon vertices
- Voronoi Planning
  - trace edges equidistant from obstacles

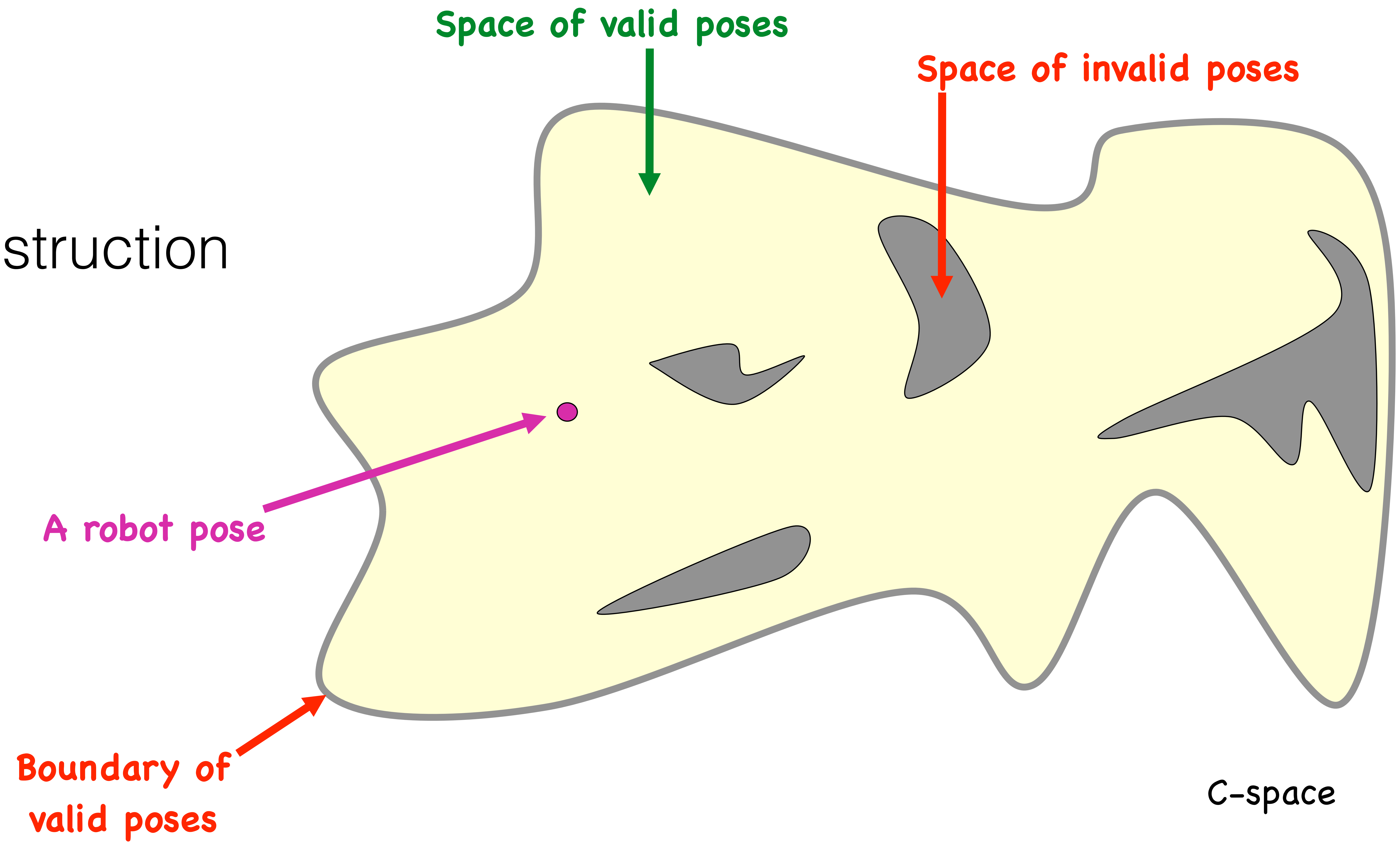
Probabilistic:

C-space sampling

- Probabilistic Roadmap (PRM)
  - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
  - sample and connect vertices in trees rooted at start and goal configuration

Probabilistic road maps

- Two phases
  - Roadmap construction
  - Path Query

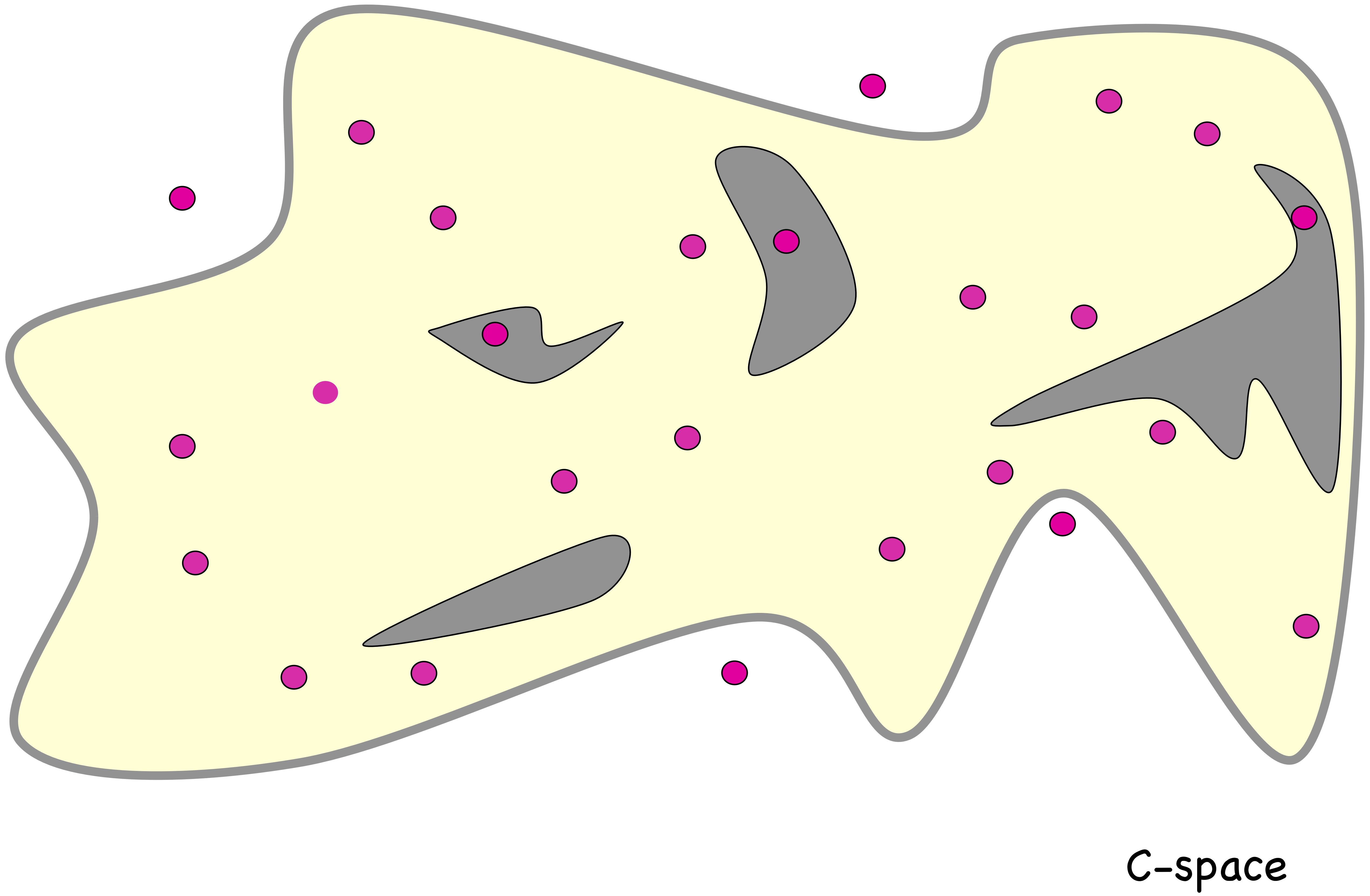


PRM: construction phase

1) **Select N sample poses at random**

2) Eliminate invalid poses

3) Connect neighboring poses

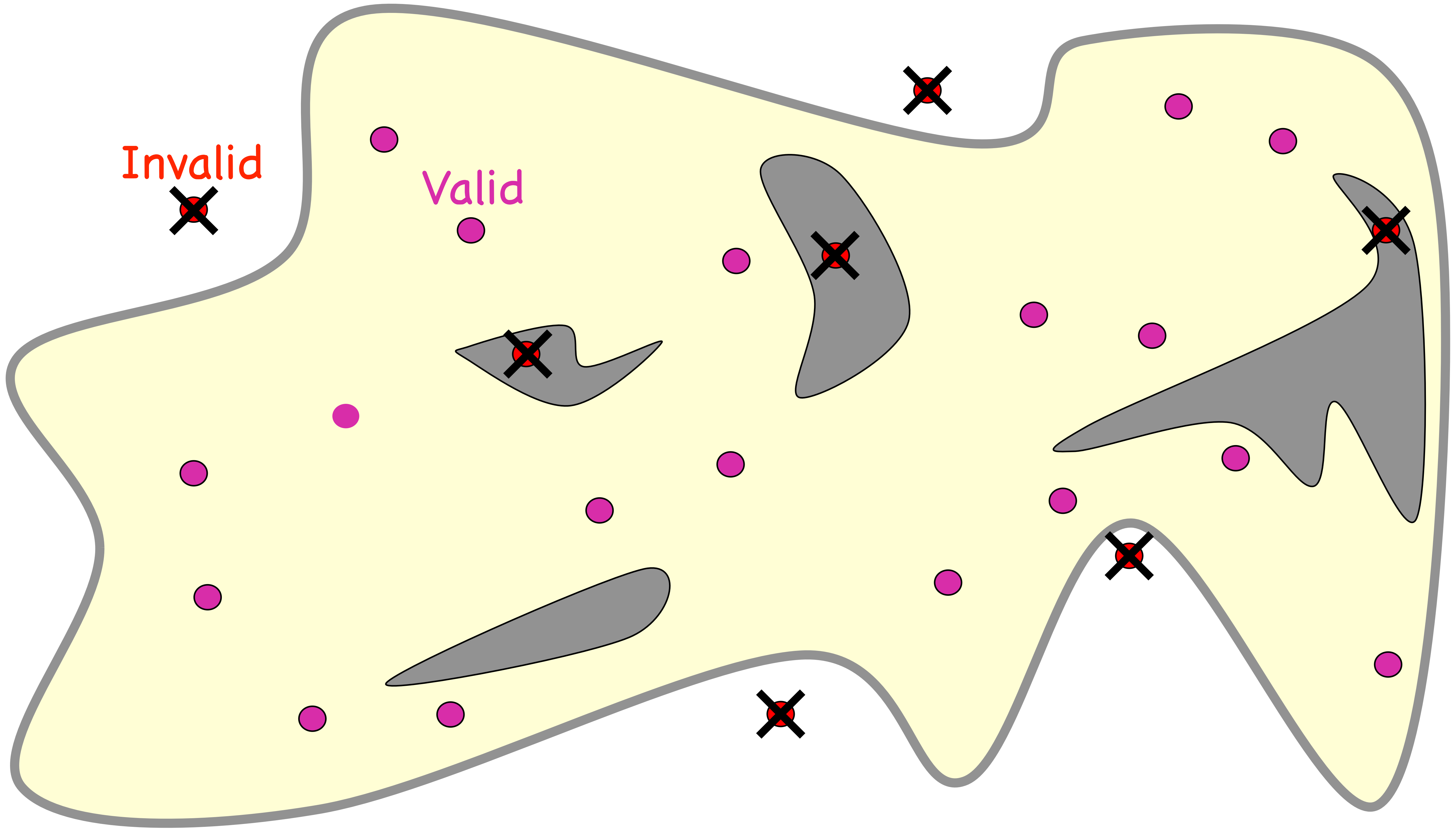


PRM: construction phase

1) Select N sample poses at random

2) **Eliminate invalid poses**

3) Connect neighboring poses



Collision detection will be covered later

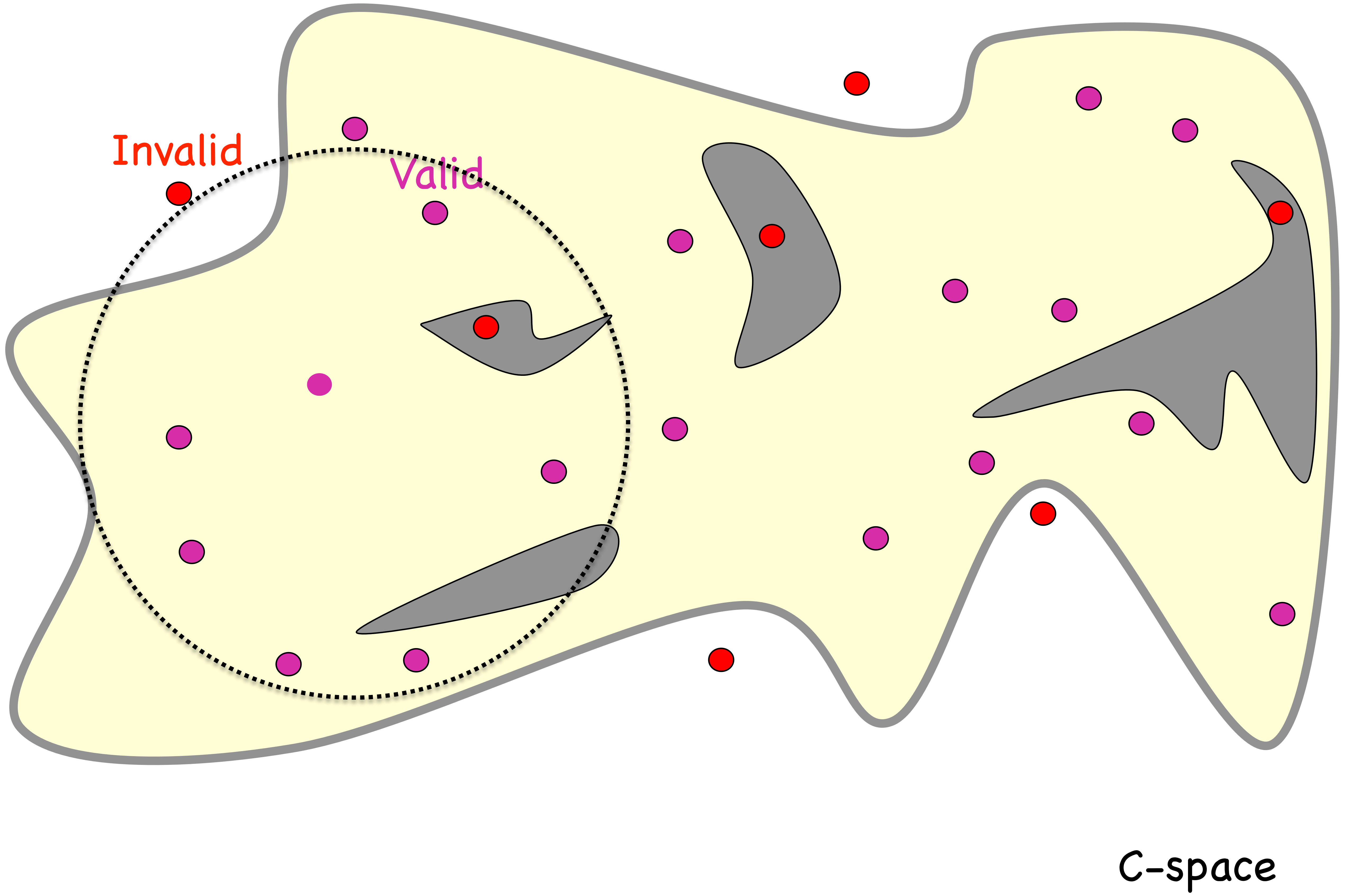
Slide borrowed from Michigan Robotics autorob.org

PRM: construction phase

1) Select N sample poses at random

2) Eliminate invalid poses

3) **Connect neighboring poses**



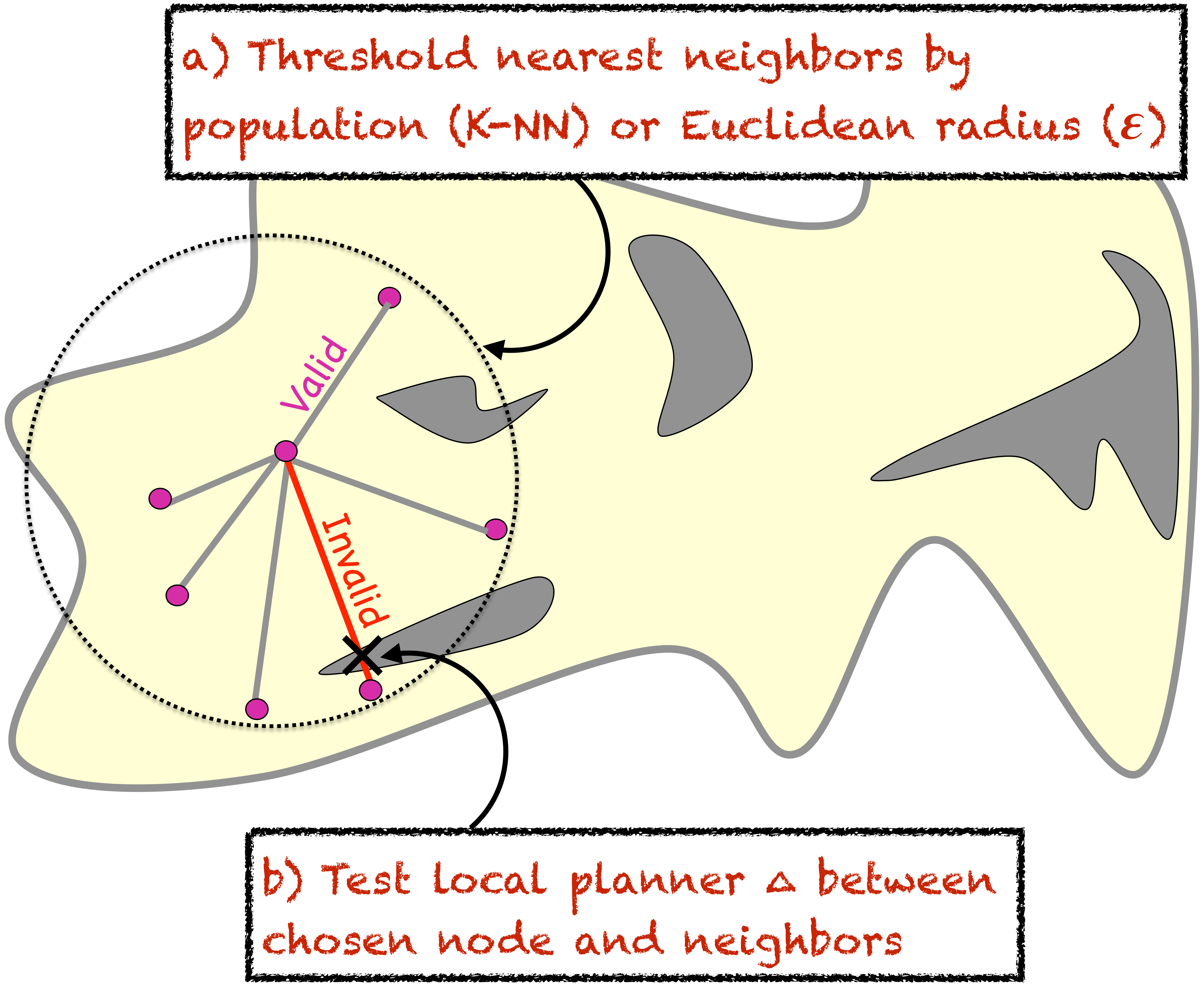
Slide borrowed from Michigan Robotics autorob.org

PRM: construction phase

1) Select N sample poses at random

2) Eliminate invalid poses

3) **Connect neighboring poses**



Slide borrowed from Michigan Robotics autorob.org

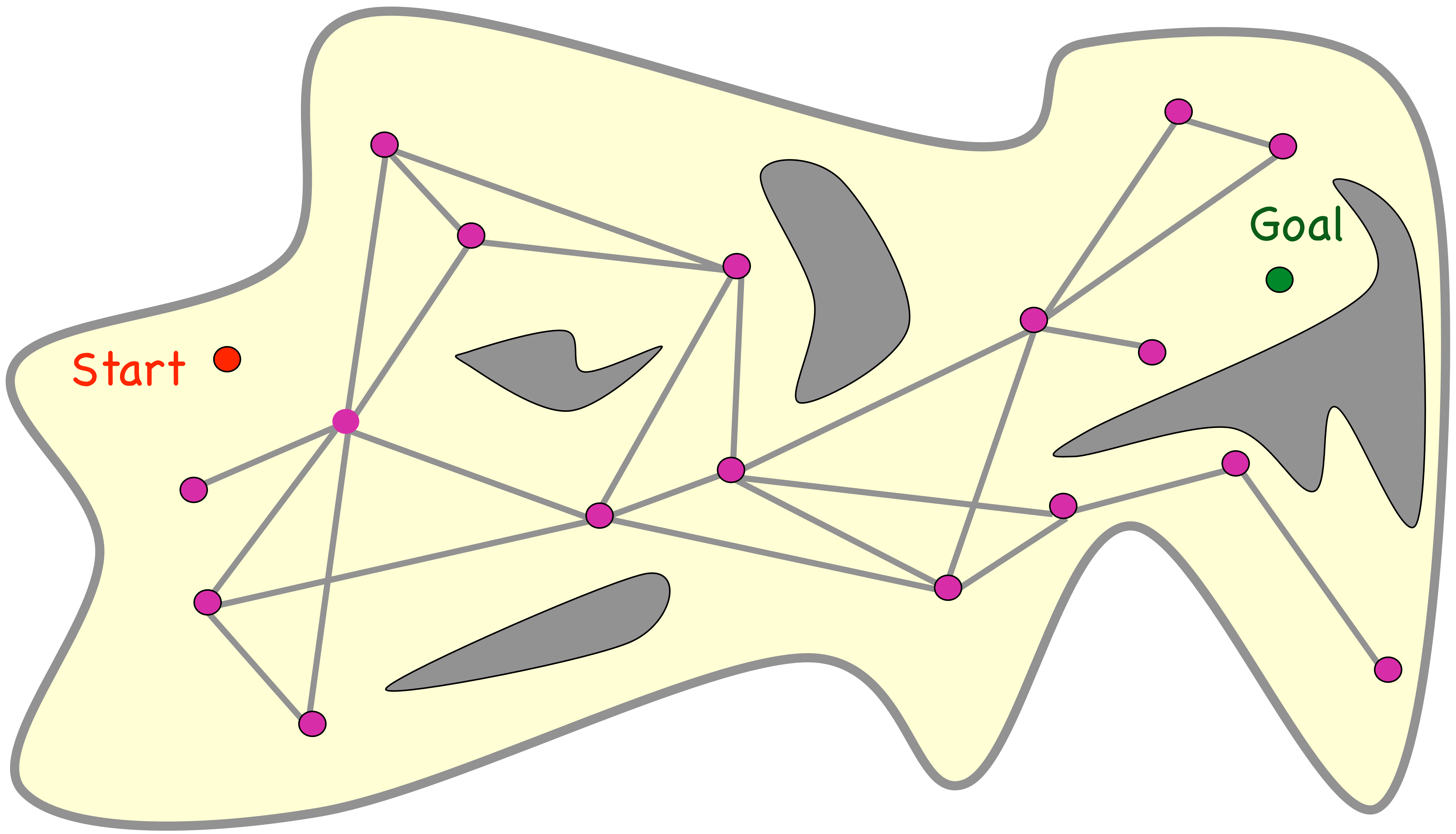
PRM: query phase

1) **Given constructed roadmap, start pose, and goal pose**

2) Attach goal and start to nearest roadmap entry nodes

3) Search for path between roadmap entry nodes

4) Return path with entry and departure edges



Slide borrowed from Michigan Robotics autorob.org

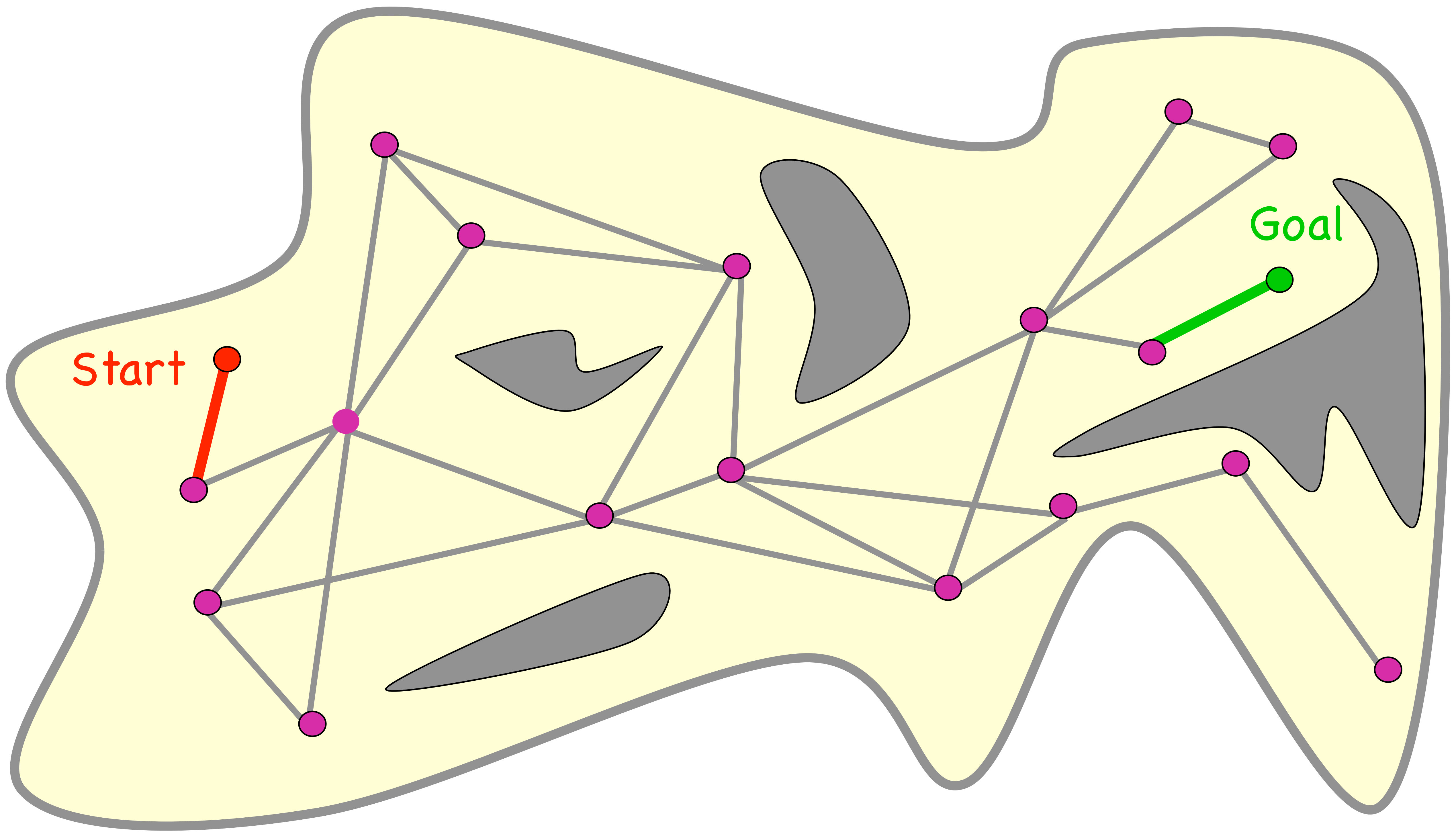
PRM: query phase

1) Given constructed roadmap, start pose, and goal pose

2) **Attach goal and start to nearest roadmap entry nodes**

3) Search for path between roadmap entry nodes

4) Return path with entry and departure edges



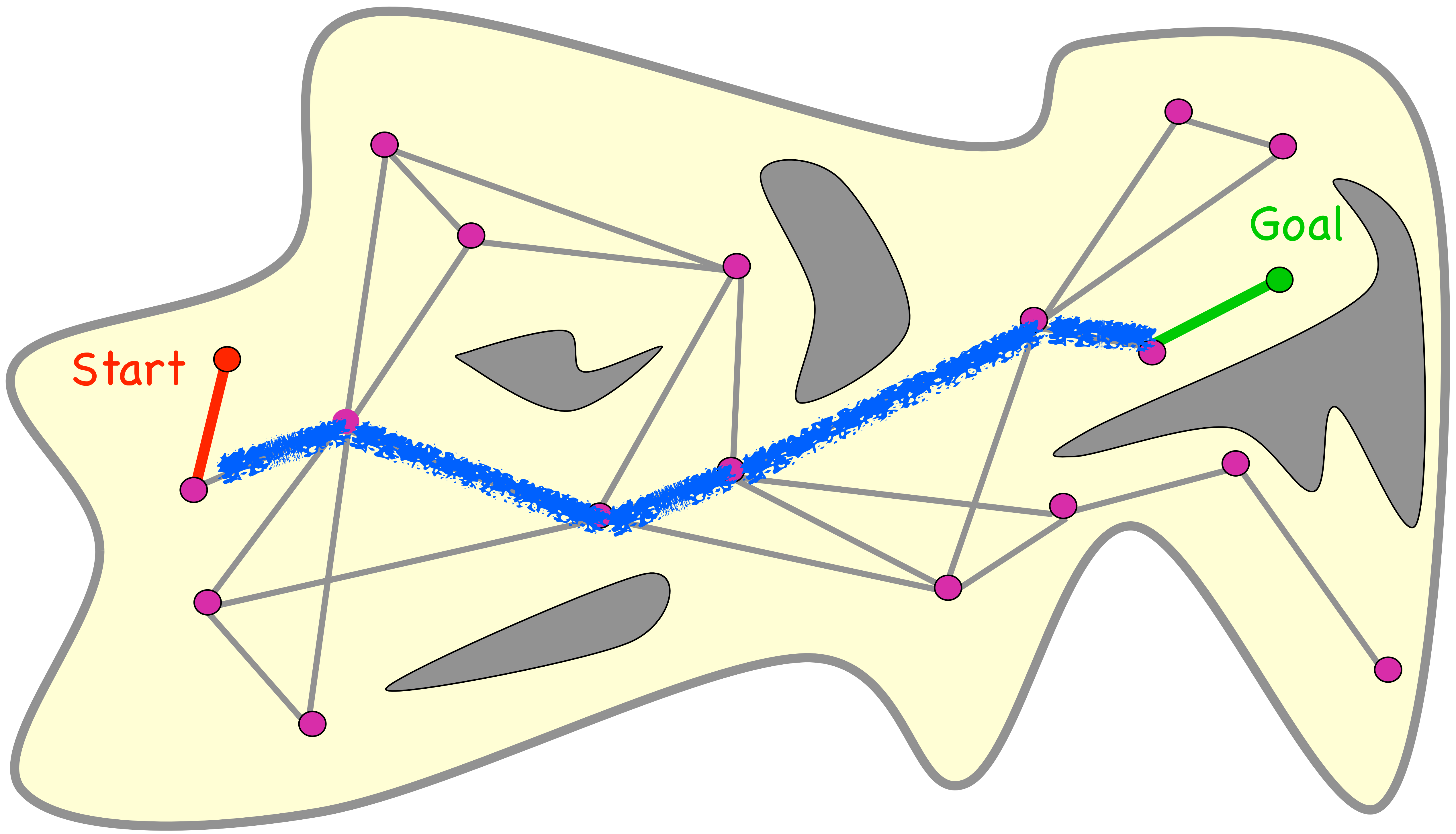
PRM: query phase

1) Given constructed roadmap, start pose, and goal pose

2) Attach goal and start to nearest roadmap entry nodes

3) **Search for path between roadmap entry nodes**

4) Return path with entry and departure edges



Remember: graph search algorithms
A*, Dijkstra, BFS, DFS

Slide borrowed from Michigan Robotics autorob.org

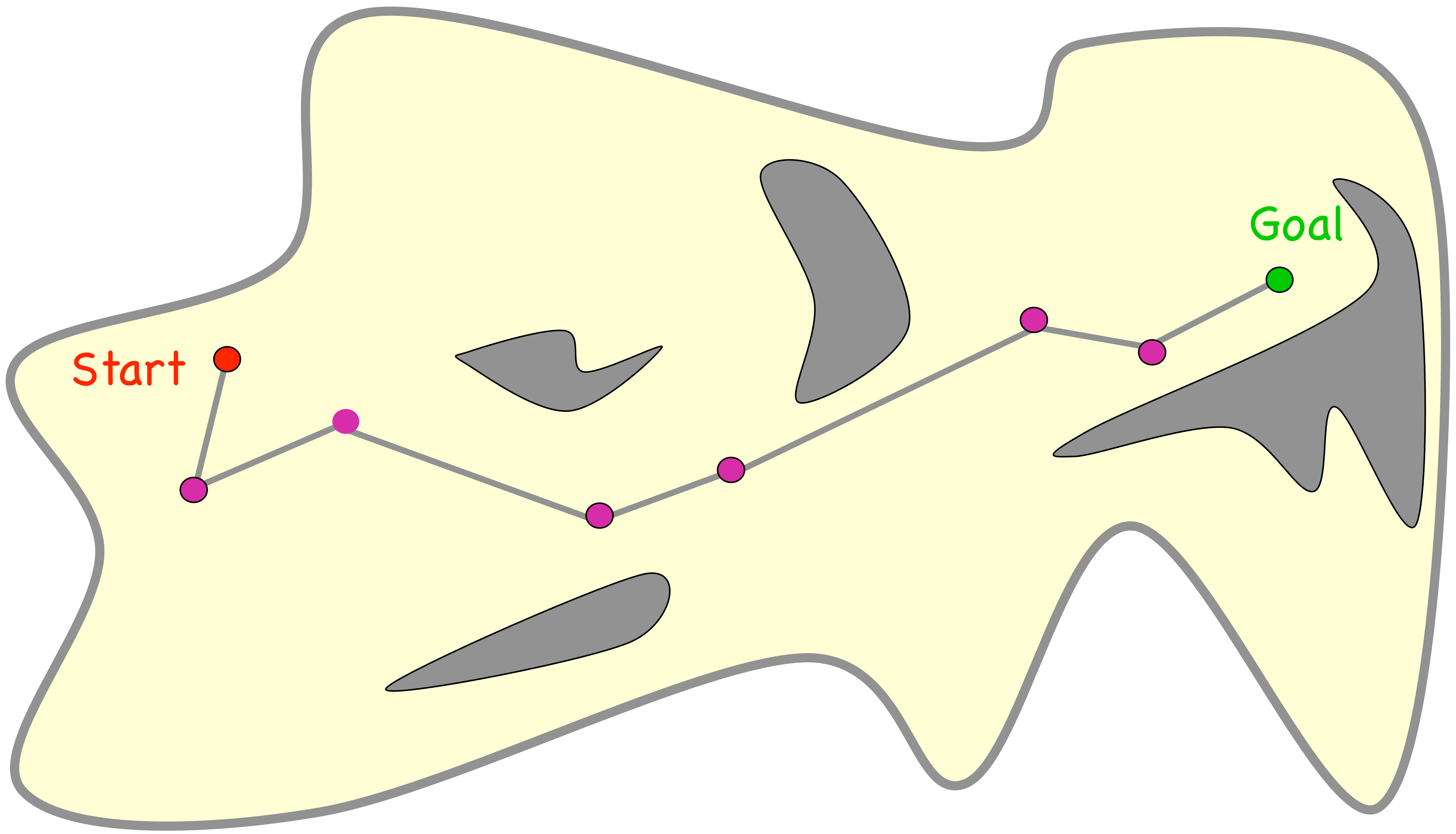
PRM: query phase

1) Given constructed roadmap, start pose, and goal pose

2) Attach goal and start to nearest roadmap entry nodes

3) Search for path between roadmap entry nodes

4) **Return path with entry and departure edges**



Multi-query planning: Considerations

i.e. if you will be querying the map multiple times (PRM by design allows this)

- Number of samples wrt. C-space dimensionality
- Balanced sampling over C-space
- Choice of distance (e.g., Euclidean)
- Choice of local planner (e.g., line subdivision)
- Selecting neighbors: (e.g., K-NN, kd-tree, cell hashing)

2 Approaches to Roadmaps

Deterministic:

complete algorithms

- Visibility Graph
    - trace lines connecting obstacle polygon vertices
- Voronoi Planning
    - trace edges equidistant from obstacles

Probabilistic:

C-space sampling

- Probabilistic Roadmap (PRM)
    - sample and connect vertices in graph for multiple planning queries
- Rapidly-exploring Random Tree (RRT)
    - sample and connect vertices in trees rooted at start and goal configuration

Single Query Planning

- Given specific start and goal configurations
- Grow trees from start and goal towards each other
- Path is found once trees connect
- Focus sampling in unexplored areas of C-space and moving towards start/goal
- Common algorithms:
  - ESTs (expansive space trees)
  - **RRTs (rapidly exploring random trees)**

RRT Algorithm

[Kuffner, LaValle 2000]

RRT Algorithm

Extend graph towards a random configuration and repeat

BUILD_RRT(q_{init})

$\mathcal{T}$.init(q_{init});

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

  $q_{rand} \leftarrow$ RANDOM_CONFIG();

  EXTEND($\mathcal{T}, q_{rand}$);

Return $\mathcal{T}$

[Kuffner, LaValle 2000]

RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT(q_init)
  T.init(q_init);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      EXTEND(T, q_rand);
  Return T
```

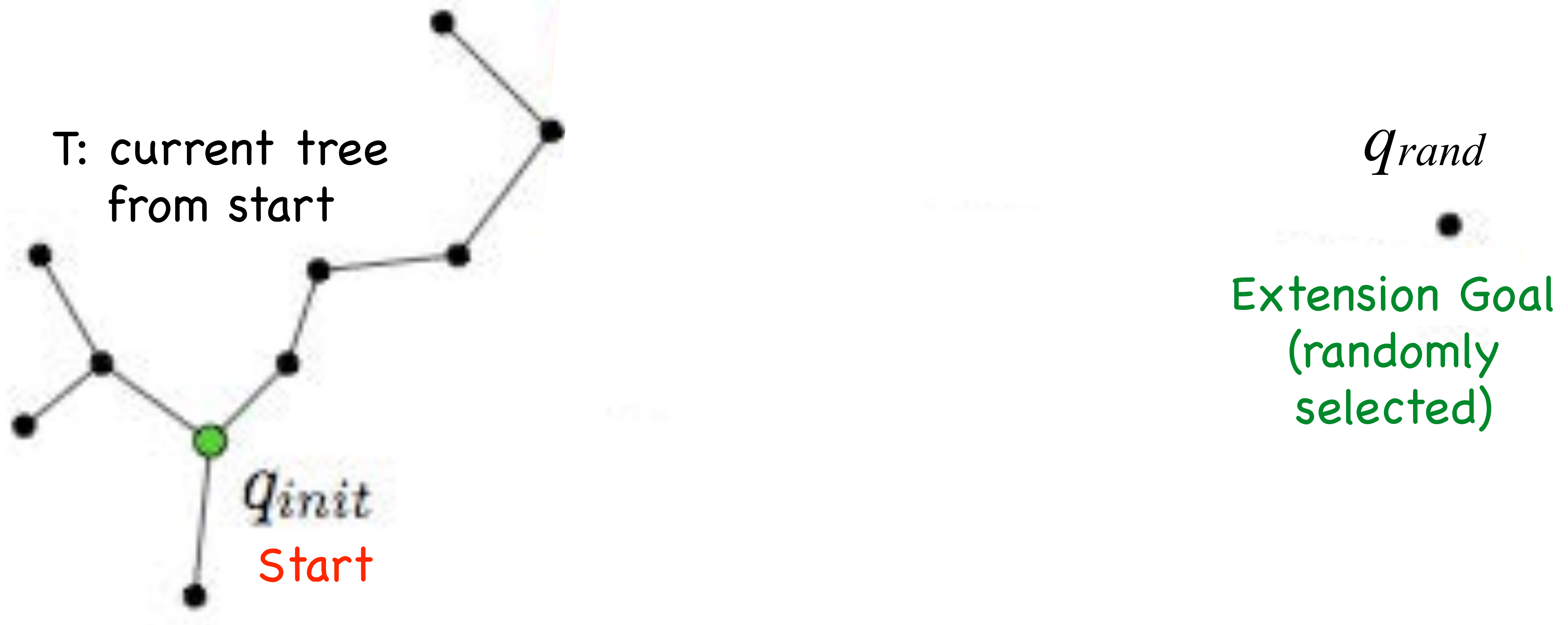


Figure 3: The EXTEND operation.

[Kuffner, LaValle 2000]

RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT(q_init)
  T.init(q_init);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      EXTEND(T, q_rand);
  Return T
```

```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  Return Trapped;
```

Extend graph towards a random configuration

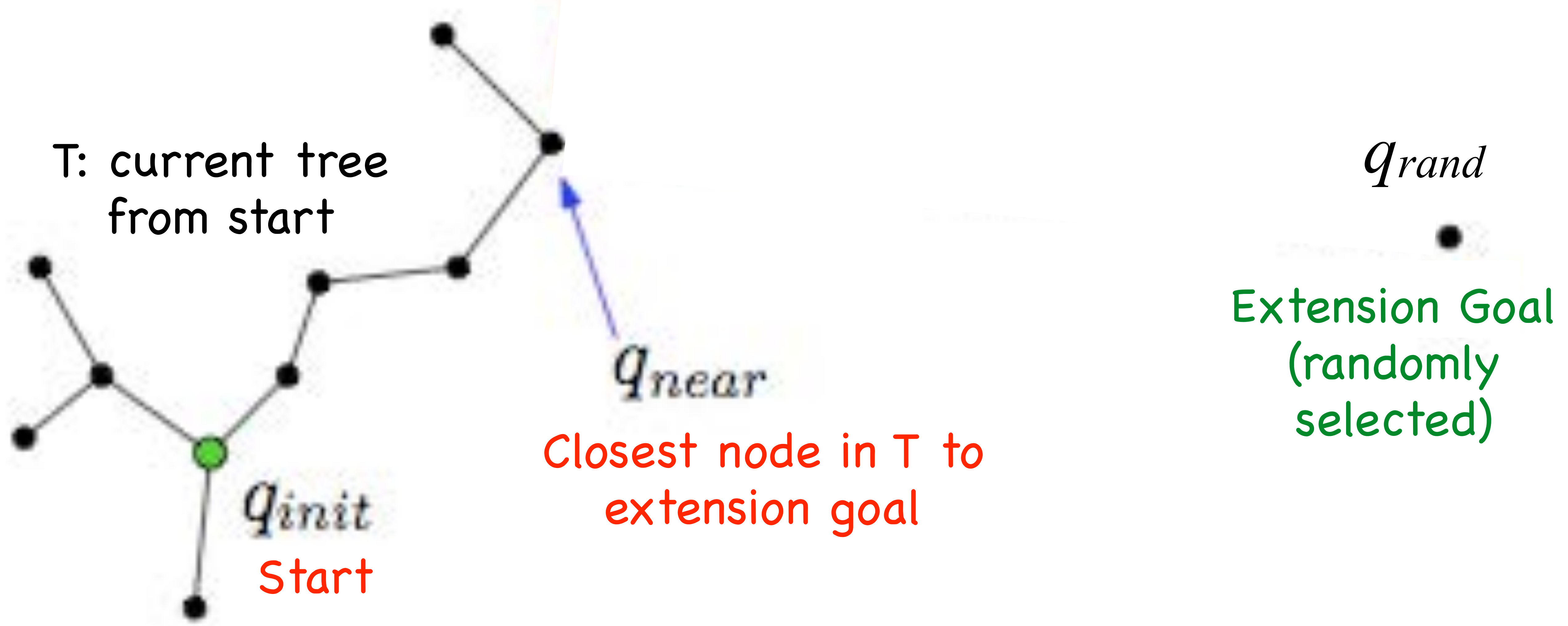


Figure 3: The EXTEND operation.

[Kuffner, LaValle 2000]

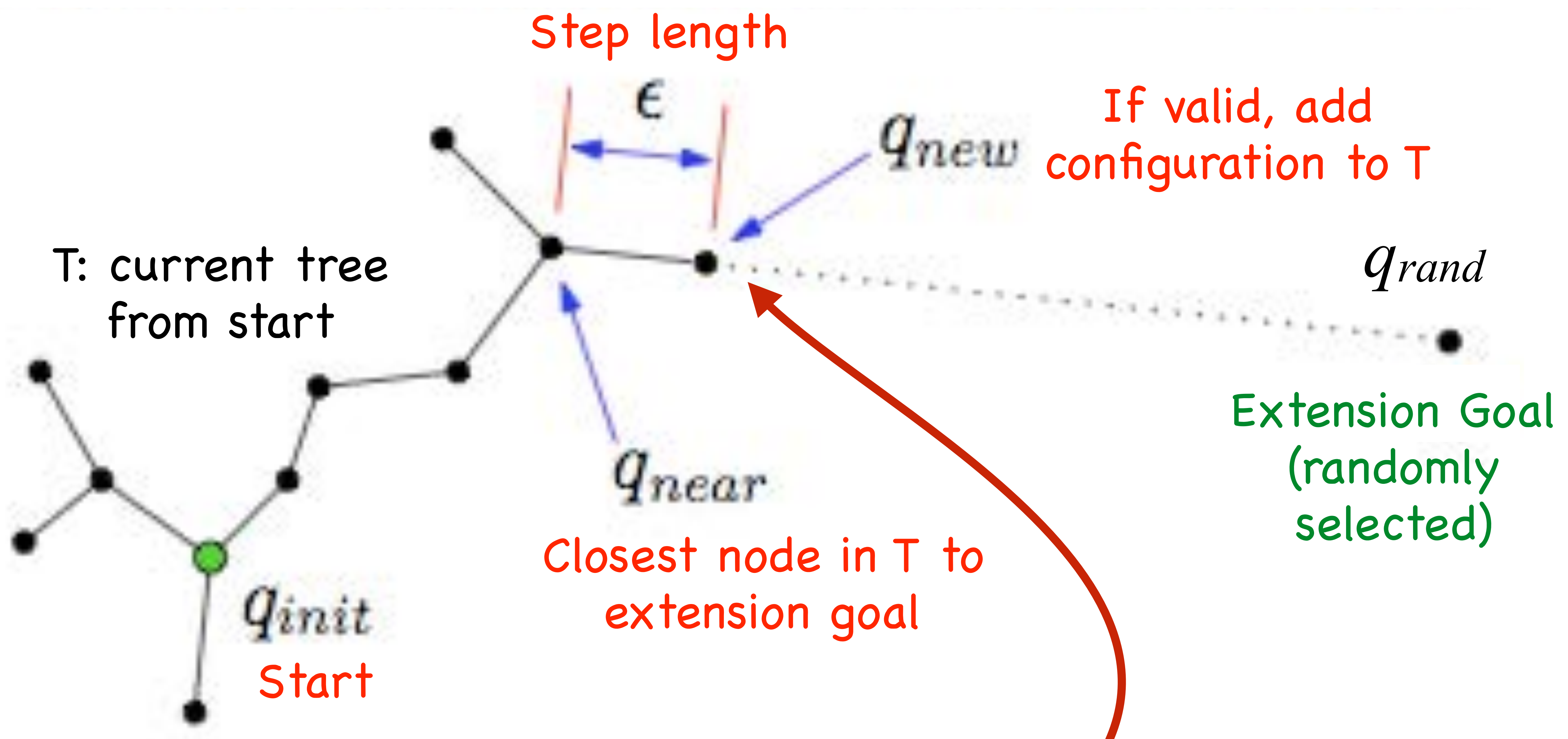
RRT Algorithm

Extend graph towards a random configuration and repeat

```
BUILD_RRT(q_init)
  T.init(q_init);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      EXTEND(T, q_rand);
  Return T
```

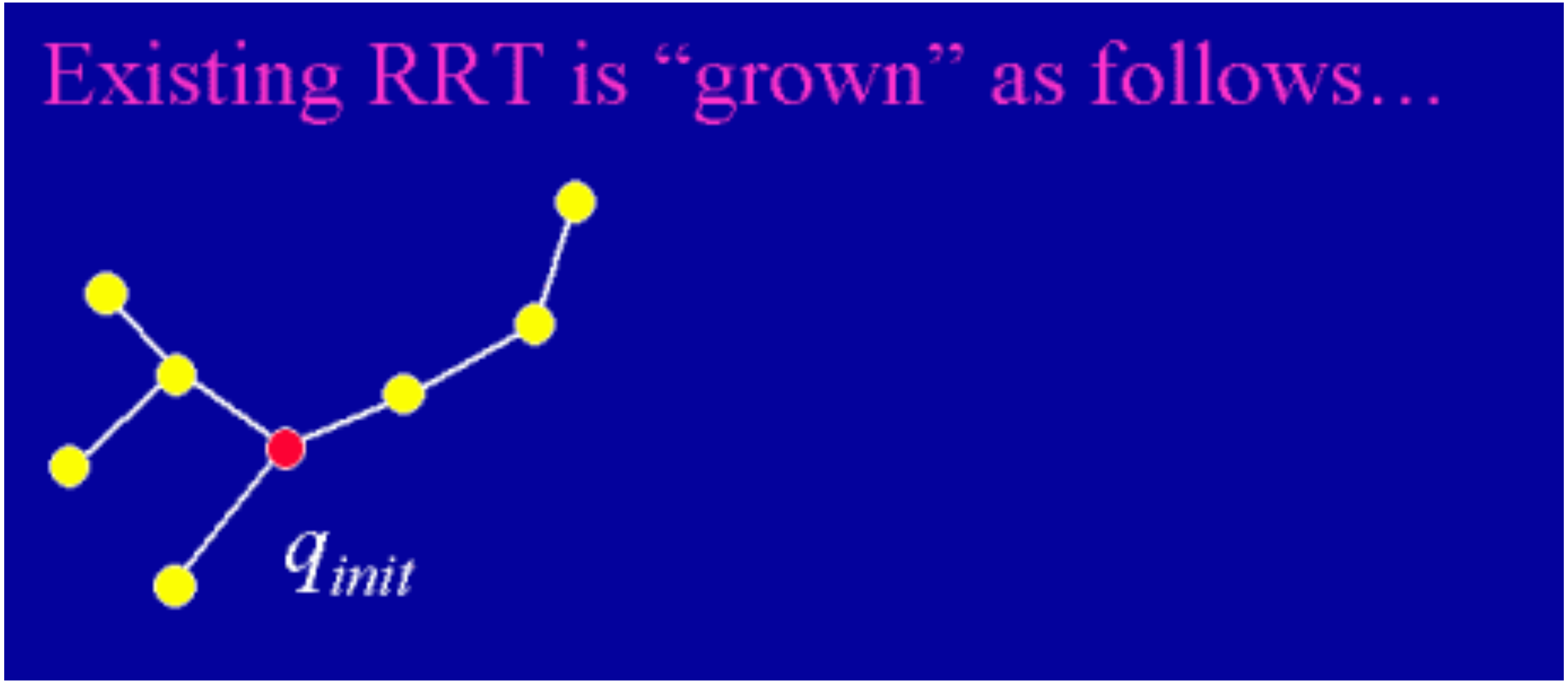
```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  Return Trapped;
```

Extend graph towards a random configuration



Generate and test new configuration along vector in C-space from q_{near} to q_{rand}

RRT Extend animation

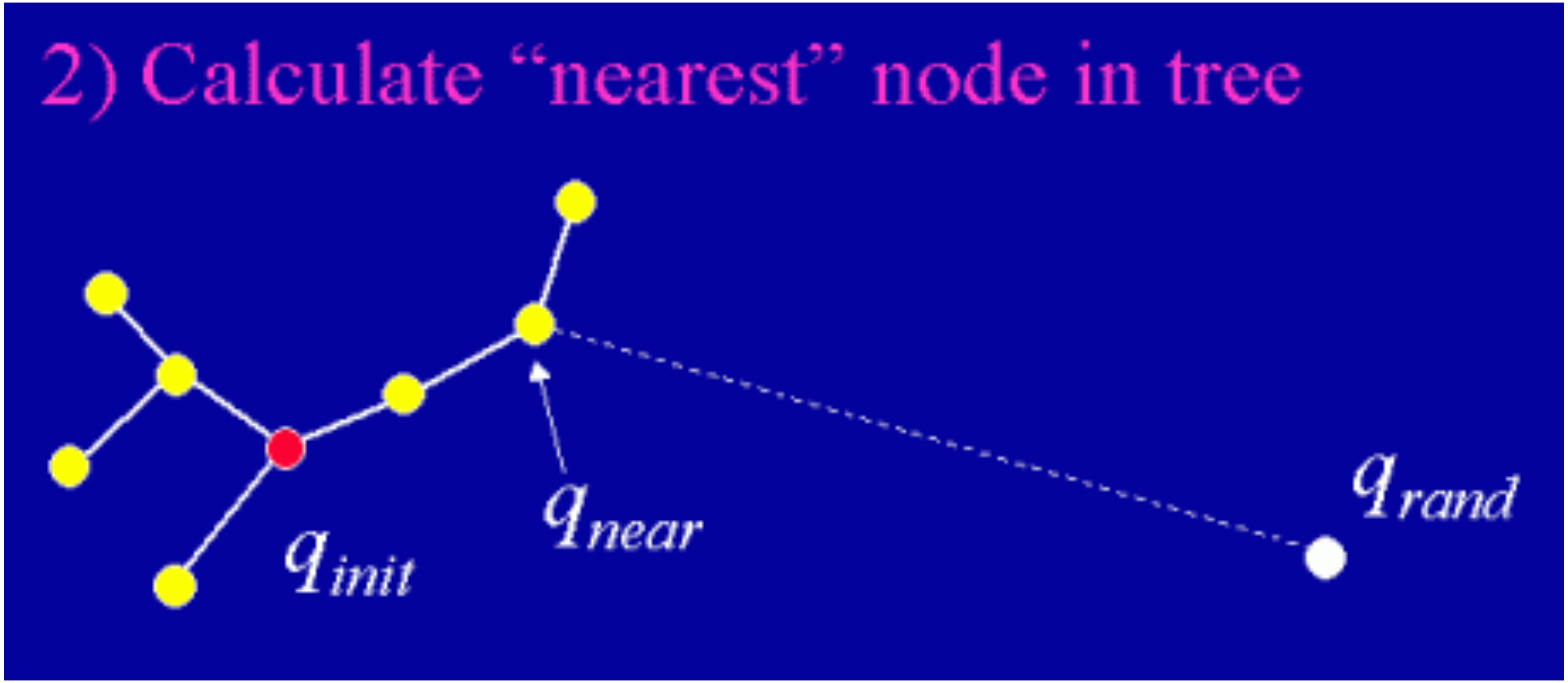


http://www.kuffner.org/james/plan/algorithm.php

RRT Extend animation

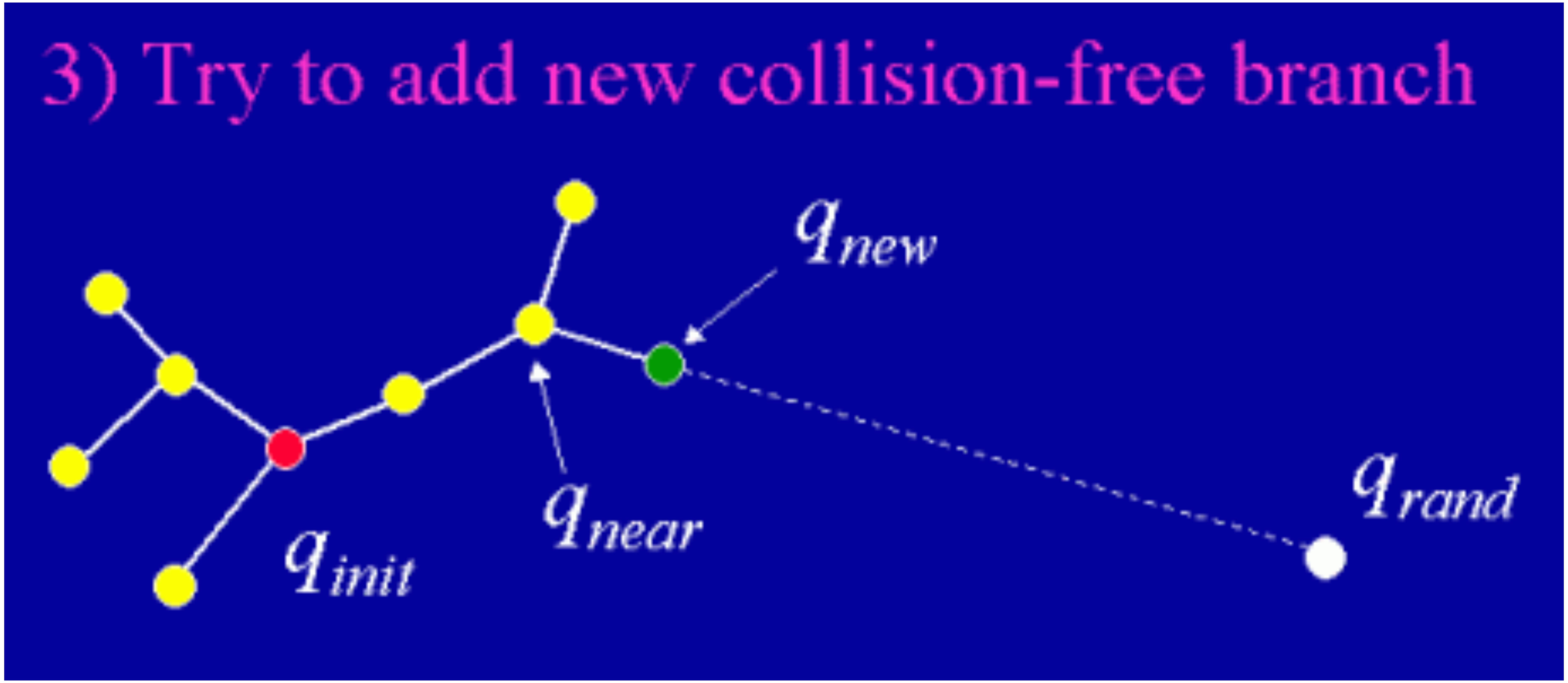
http://www.kuffner.org/james/plan/algorithm.php

RRT Extend animation



http://www.kuffner.org/james/plan/algorithm.php

RRT Extend animation



http://www.kuffner.org/james/plan/algorithm.php

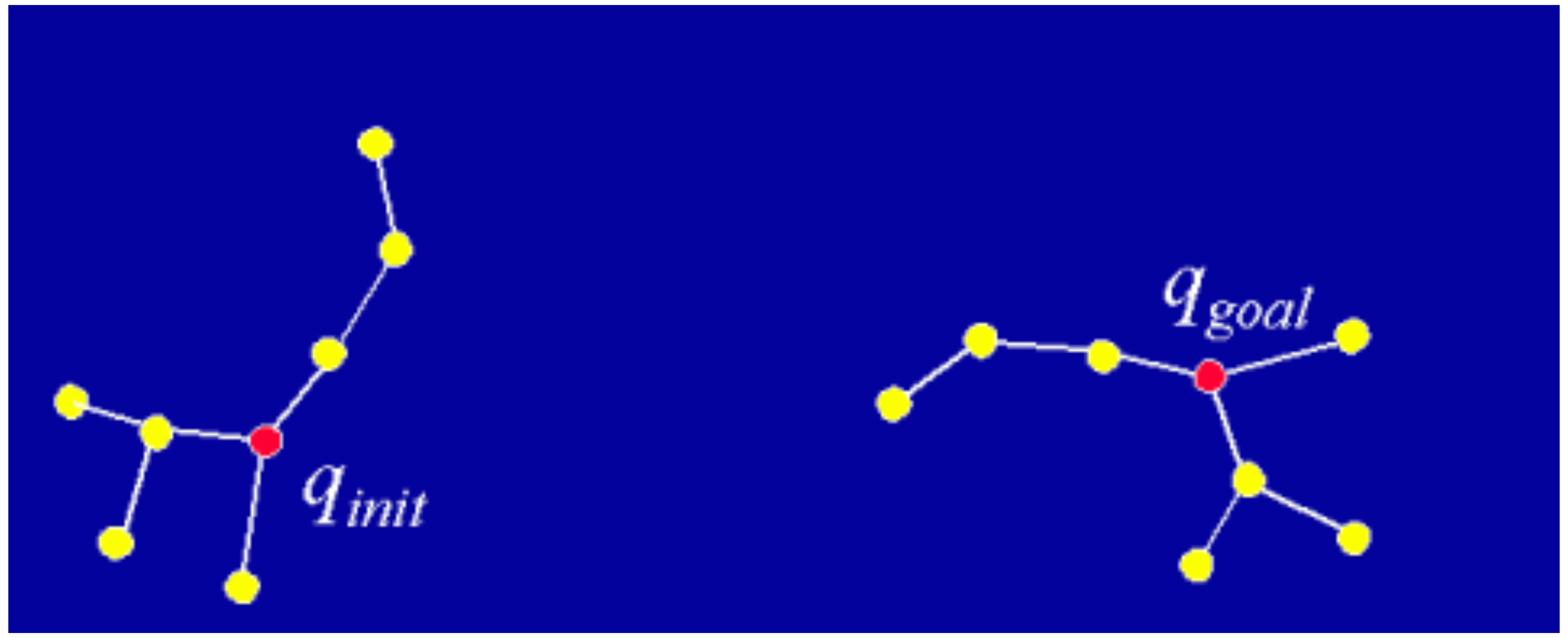
Demo

RRT Connect

[Kuffner, LaValle 2000]

0) Use 2 trees (A and B) rooted at start and goal configurations

RRT_CONNECT_PLANNER(q_{init}, q_{goal})
1 $\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
2 **for** $k = 1$ **to** K **do**
3 $q_{rand} \leftarrow$ RANDOM_CONFIG();
4 **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
5 **if** (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
6 Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
7 SWAP($\mathcal{T}_a, \mathcal{T}_b$);
8 Return *Failure*



[Kuffner, LaValle 2000]

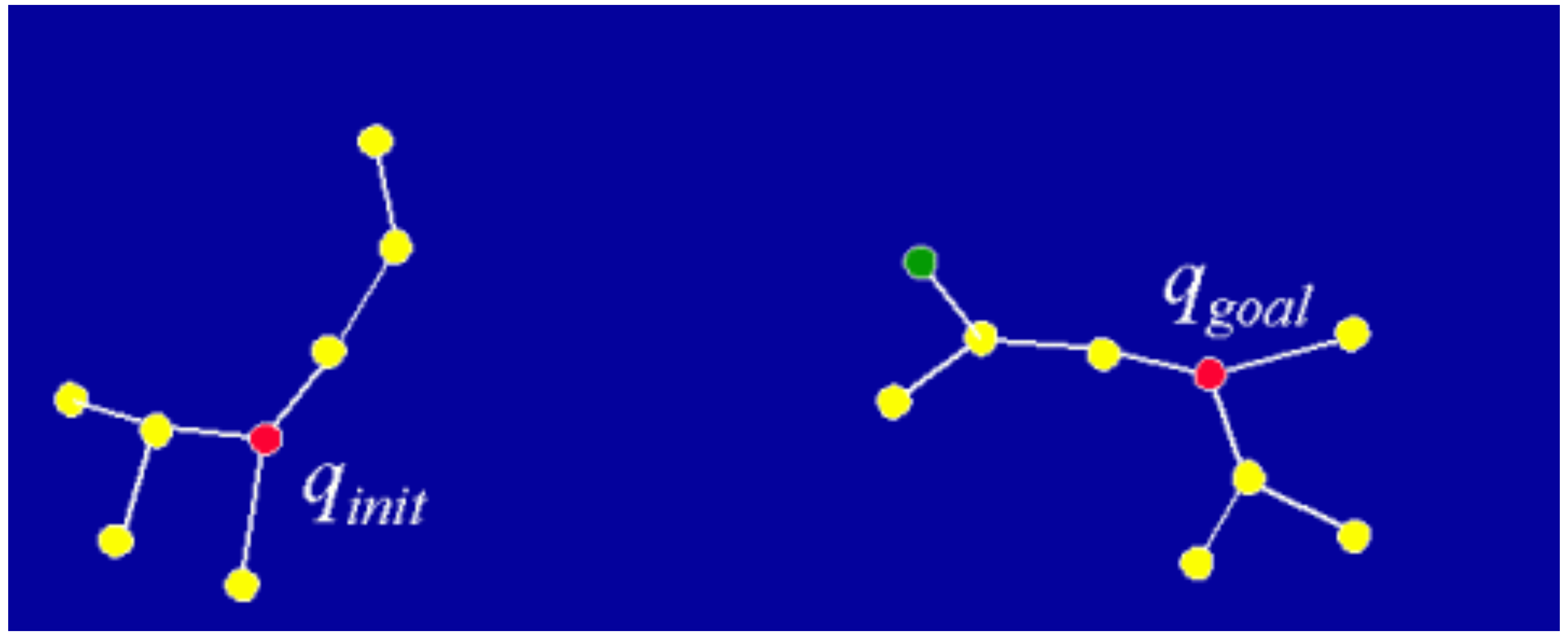
RRT Connect

0) Use 2 trees (A and B) rooted at start and goal configurations

RRT_CONNECT_PLANNER(q_{init}, q_{goal})
- $\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
- **for** $k = 1$ **to** K **do**
  - $q_{rand} \leftarrow$ RANDOM_CONFIG();
  - **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
    - **if** (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
      - Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
  - SWAP($\mathcal{T}_a, \mathcal{T}_b$);
- Return *Failure*

EXTEND($\mathcal{T}, q$)
- $q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
- **if** NEW_CONFIG(q, q_{near}, q_{new}) **then**
  - $\mathcal{T}$.add_vertex(q_{new});
  - $\mathcal{T}$.add_edge(q_{near}, q_{new});
  - **if** $q_{new} = q$ **then**
    - Return *Reached*;
  - **else**
    - Return *Advanced*;
- Return *Trapped*;

1) Extend tree A towards a random configuration



RRT Connect

[Kuffner, LaValle 2000]

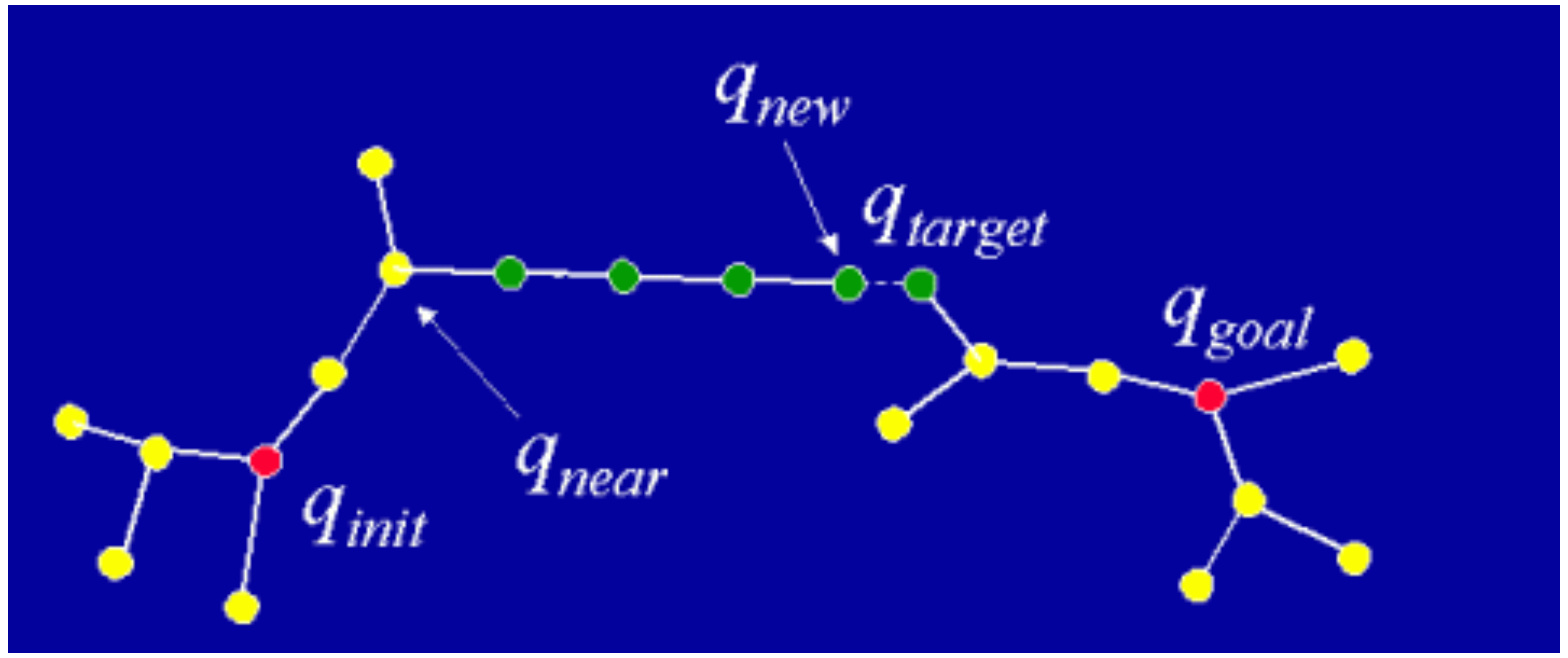
0) Use 2 trees (A and B) rooted at start and goal configurations

```
RRT_CONNECT_PLANNER(q_init, q_goal)
  T_a.init(q_init); T_b.init(q_goal);
  for k = 1 to K do
      q_rand ← RANDOM_CONFIG();
      if not (EXTEND(T_a, q_rand) = Trapped) then
          if (CONNECT(T_b, q_new) = Reached) then
              Return PATH(T_a, T_b);
      SWAP(T_a, T_b);
  Return Failure
```

```
EXTEND(T, q)
  q_near ← NEAREST_NEIGHBOR(q, T);
  if NEW_CONFIG(q, q_near, q_new) then
      T.add_vertex(q_new);
      T.add_edge(q_near, q_new);
      if q_new = q then
          Return Reached;
      else
          Return Advanced;
  Return Trapped;
```

1) Extend tree A towards a random configuration

```
CONNECT(T, q)
  repeat
      S ← EXTEND(T, q);
  until not (S = Advanced)
  Return S;
```



2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor

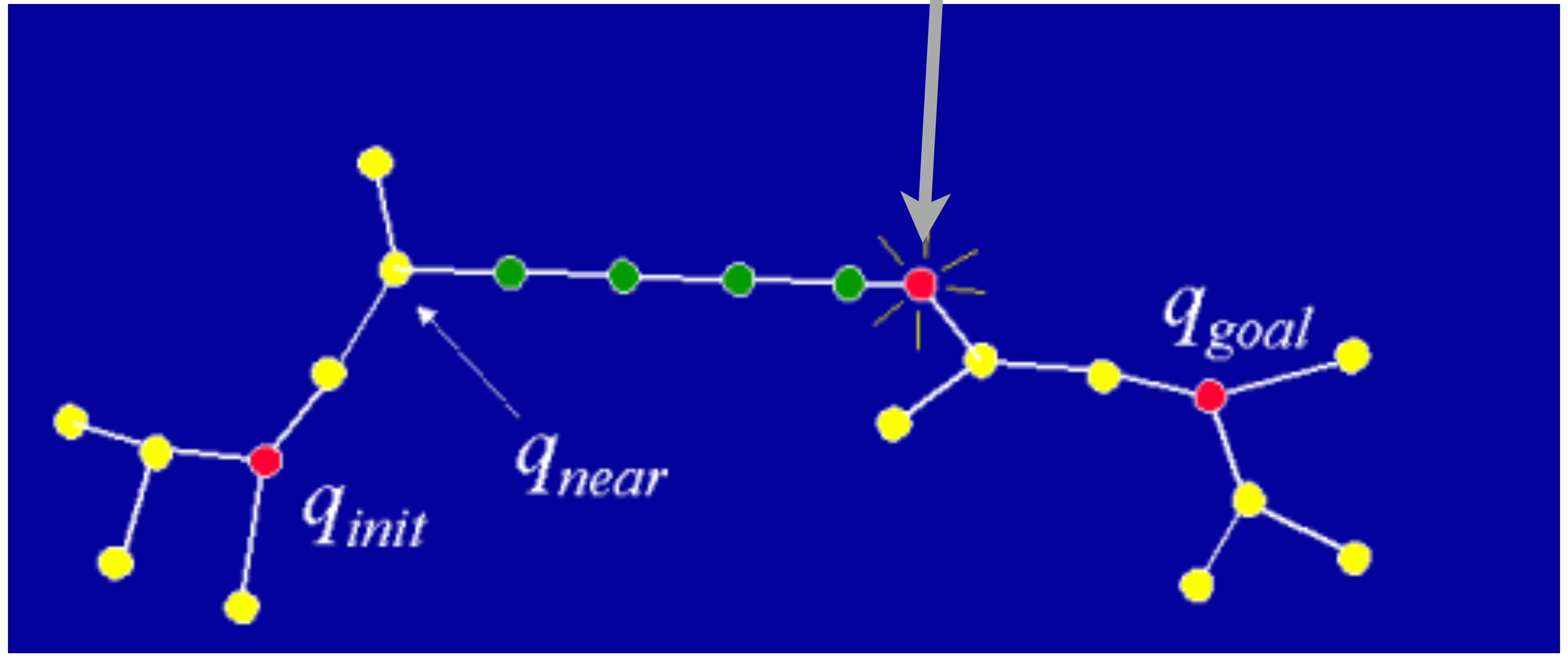
[Kuffner, LaValle 2000]

RRT Connect

0) Use 2 trees (A and B) rooted at start and goal configurations

```
RRT_CONNECT_PLANNER(q_init, q_goal)
    T_a.init(q_init); T_b.init(q_goal);
    for k = 1 to K do
        q_rand ← RANDOM_CONFIG();
        if not (EXTEND(T_a, q_rand) = Trapped) then
            if (CONNECT(T_b, q_new) = Reached) then
                Return PATH(T_a, T_b);
        SWAP(T_a, T_b);
    Return Failure
```

search succeeds if trees connect




```
EXTEND(T, q)
    q_near ← NEAREST_NEIGHBOR(q, T);
    if NEW_CONFIG(q, q_near, q_new) then
        T.add_vertex(q_new);
        T.add_edge(q_near, q_new);
        if q_new = q then
            Return Reached;
        else
            Return Advanced;
    Return Trapped;
```

1) Extend tree A towards a random configuration

```
CONNECT(T, q)
    repeat
        S ← EXTEND(T, q);
    until not (S = Advanced)
    Return S;
```

2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor

RRT Connect

[Kuffner, LaValle 2000]

0) Use 2 trees (A and B) rooted at start and goal configurations

RRT_CONNECT_PLANNER(q_{init}, q_{goal})
$\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
for $k = 1$ **to** K **do**
    $q_{rand} \leftarrow$ RANDOM_CONFIG();
    if not (EXTEND($\mathcal{T}_a, q_{rand}$) $=$ *Trapped*) **then**
        if (CONNECT($\mathcal{T}_b, q_{new}$) $=$ *Reached*) **then**
            Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
Return *Failure*

EXTEND($\mathcal{T}, q$)
$q_{near} \leftarrow$ NEAREST_NEIGHBOR($q, \mathcal{T}$);
if NEW_CONFIG(q, q_{near}, q_{new}) **then**
    $\mathcal{T}$.add_vertex(q_{new});
    $\mathcal{T}$.add_edge(q_{near}, q_{new});
    if $q_{new} = q$ **then**
        Return *Reached*;
    else
        Return *Advanced*;
Return *Trapped*;

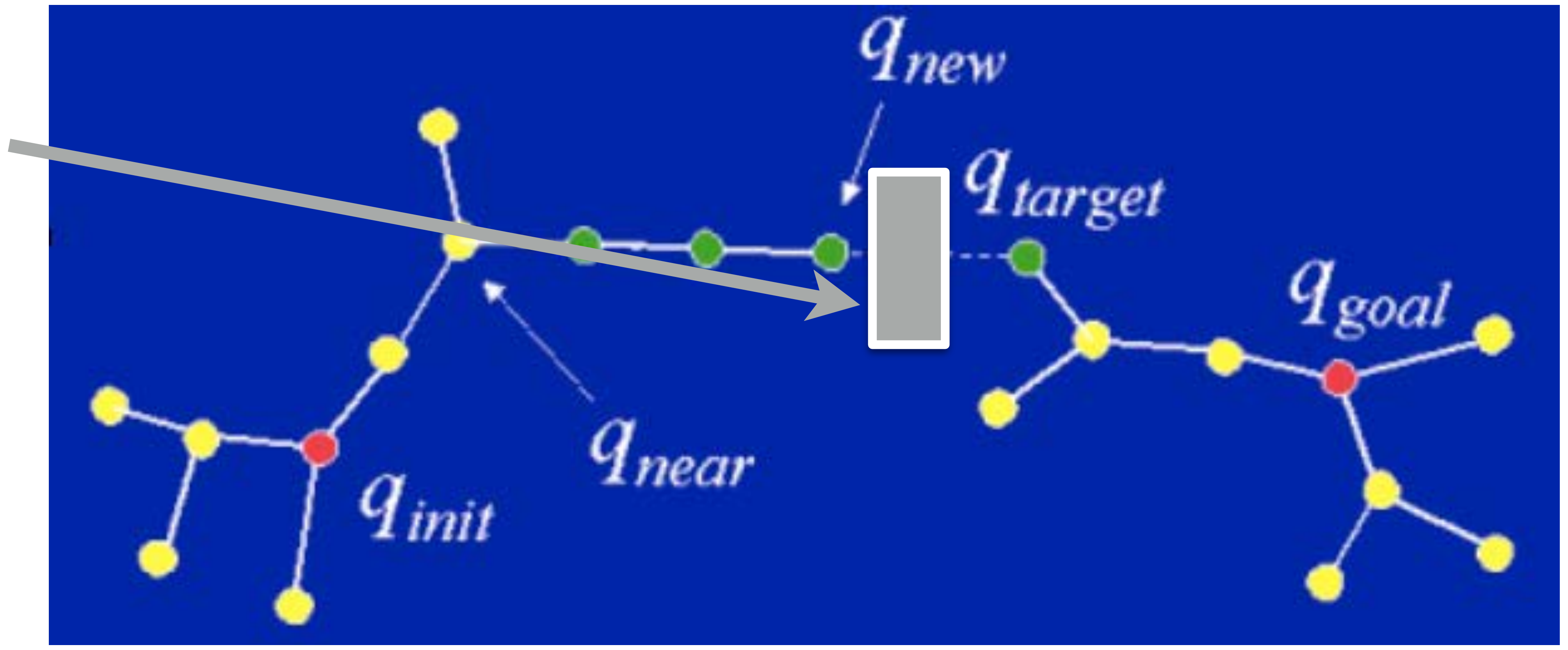
1) Extend tree A towards a random configuration

CONNECT($\mathcal{T}, q$)
repeat
    $S \leftarrow$ EXTEND($\mathcal{T}, q$);
until not ($S =$ *Advanced*)
Return S;

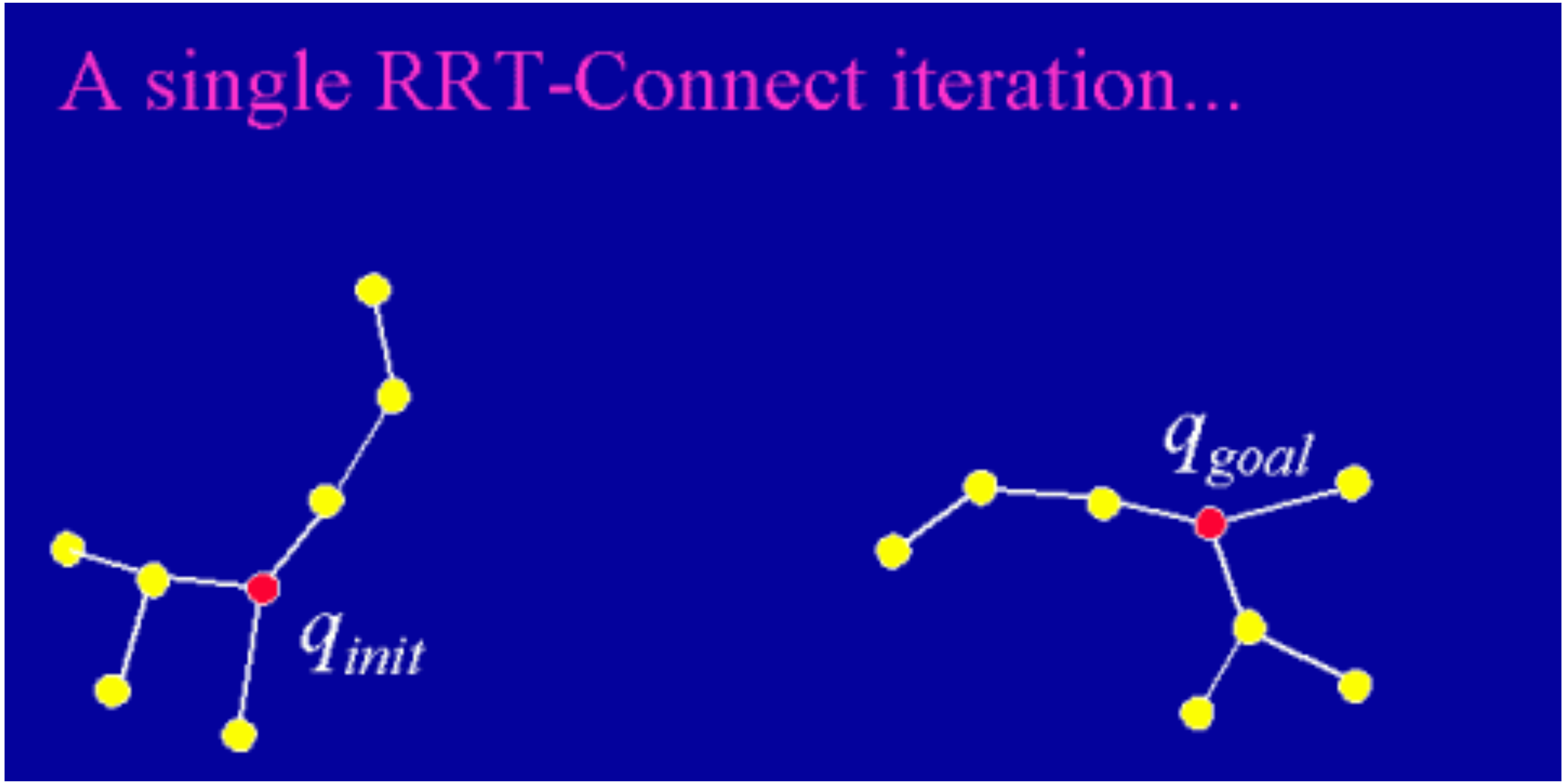
2) Try to connect tree B to tree A by extending repeatedly from its nearest neighbor

3) reverse roles for trees A and B and repeat

collision encountered

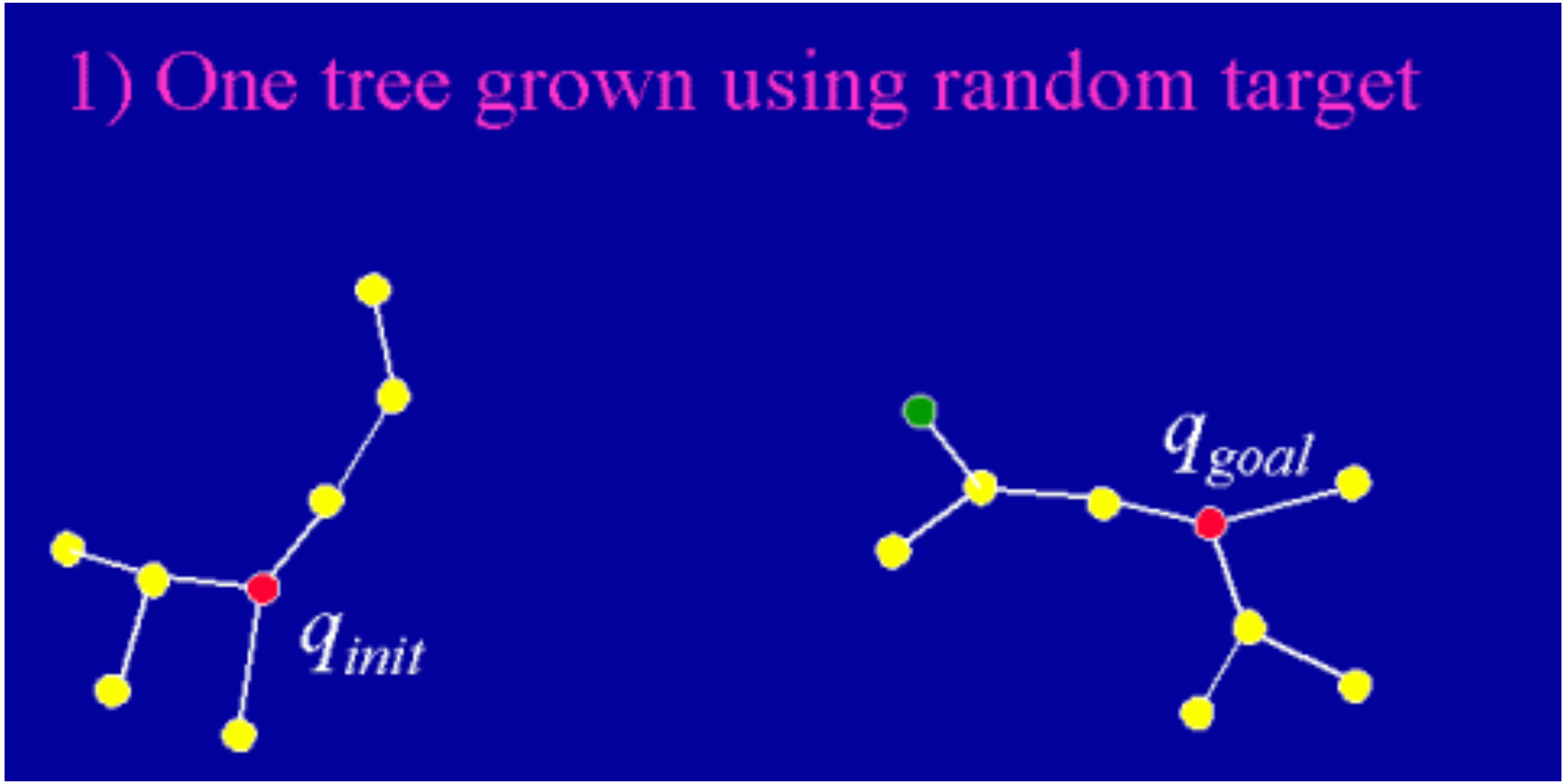


RRT-Connect animation



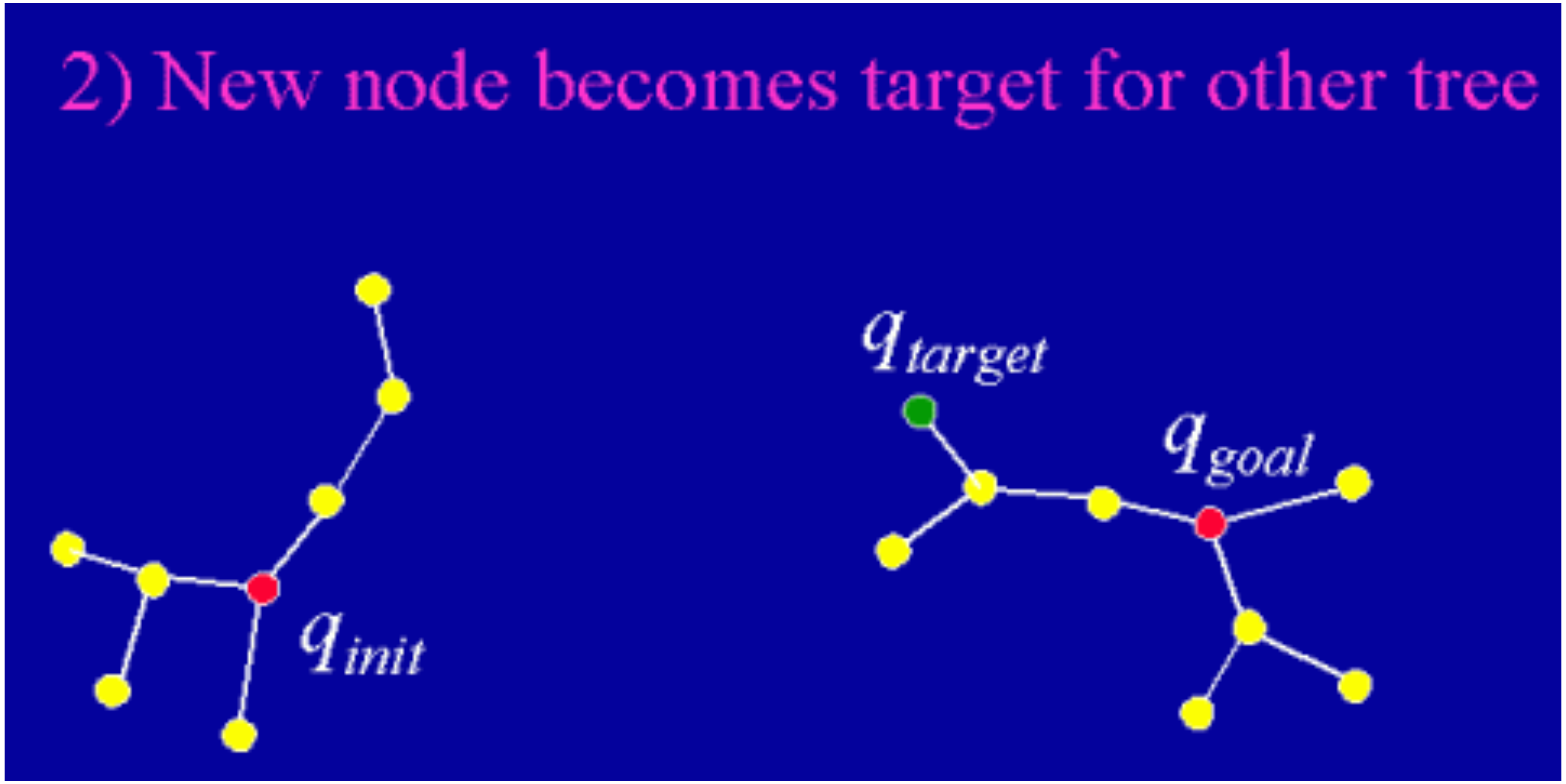
http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation



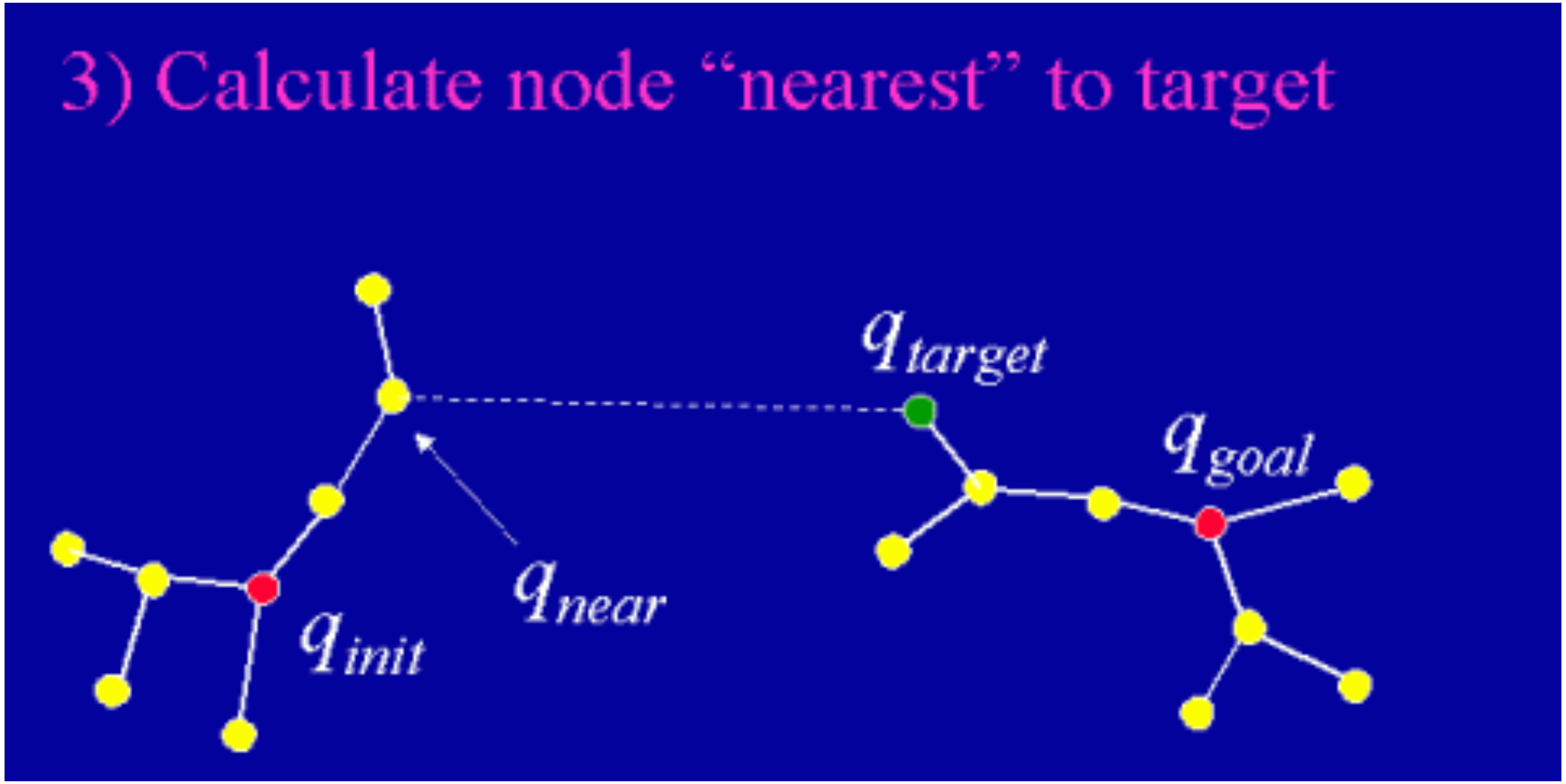
http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation



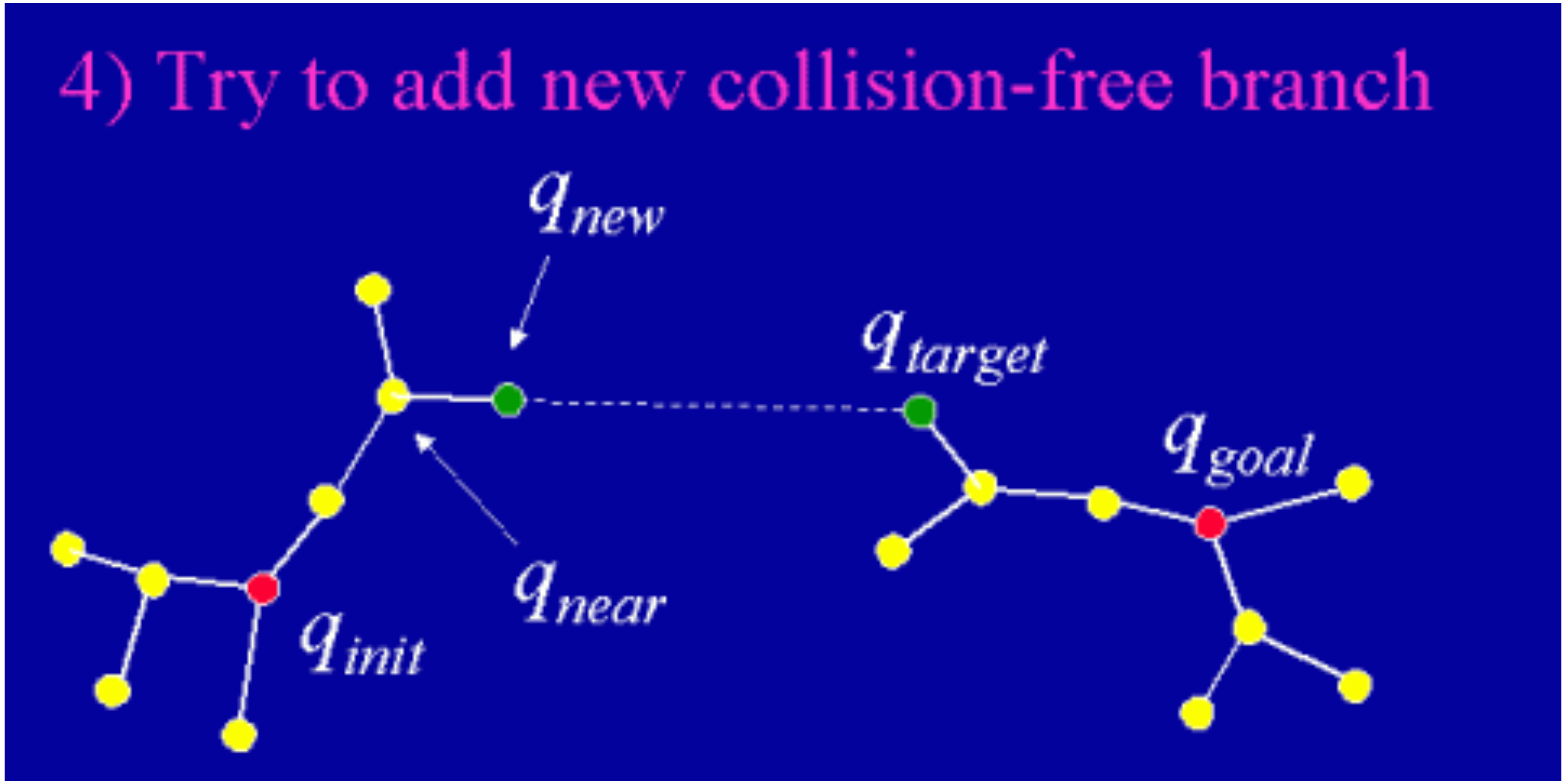
http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation



http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation



http://www.kuffner.org/james/plan/algorithm.php

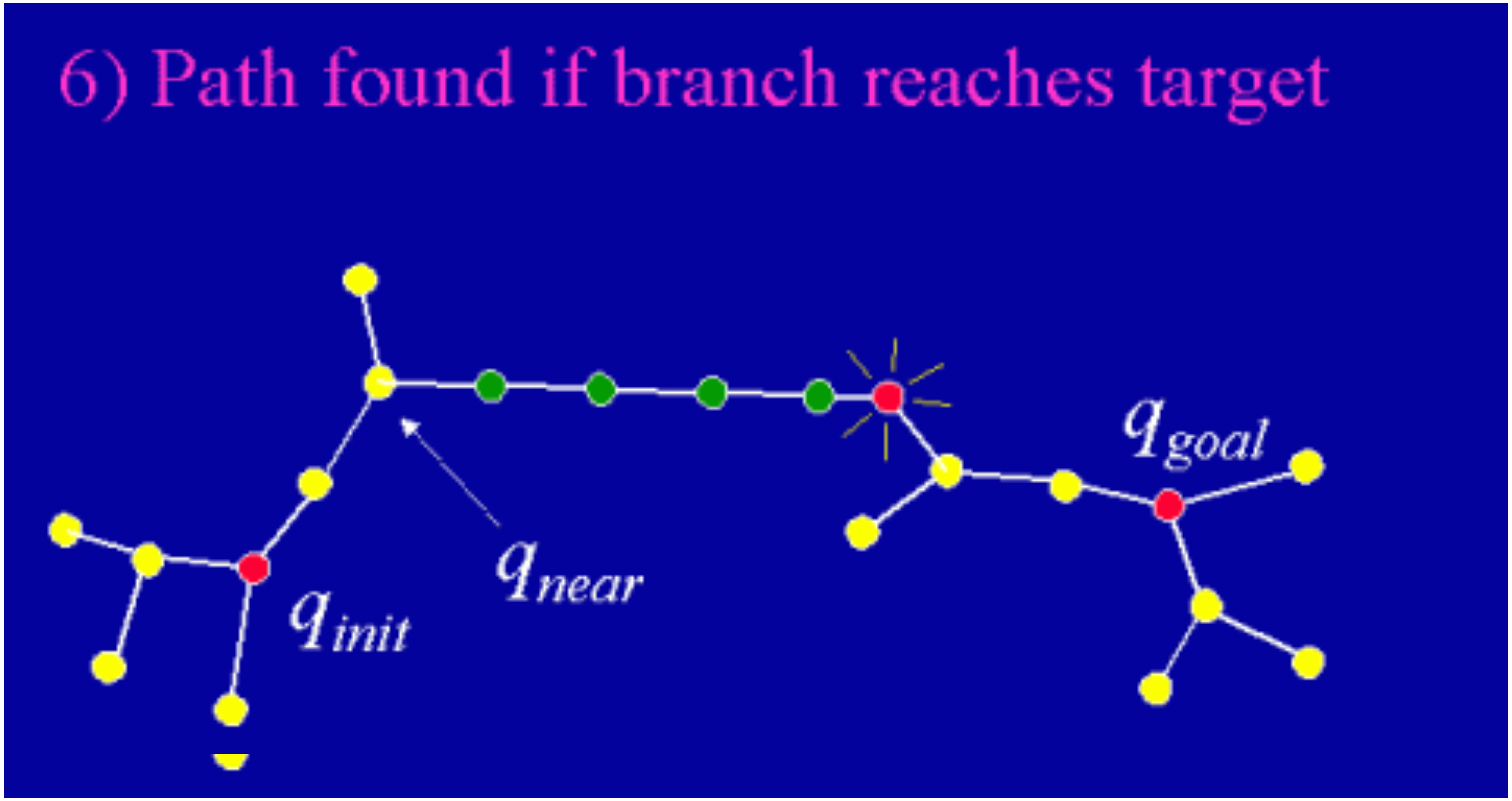
RRT-Connect animation

http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation

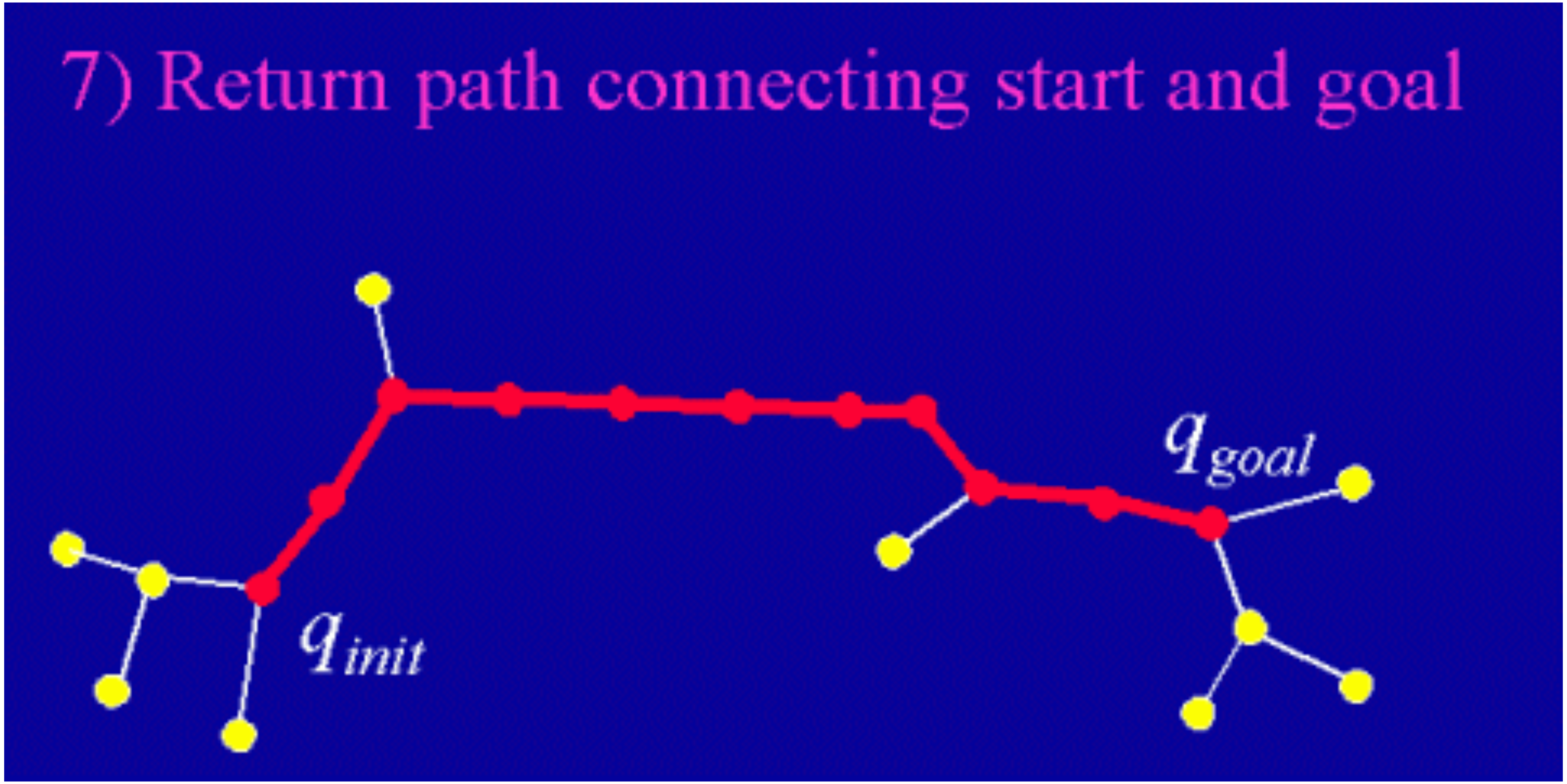
http://www.kuffner.org/james/plan/algorithm.php

RRT-Connect animation



http://www.kuffner.org/james/plan/algorithm.php

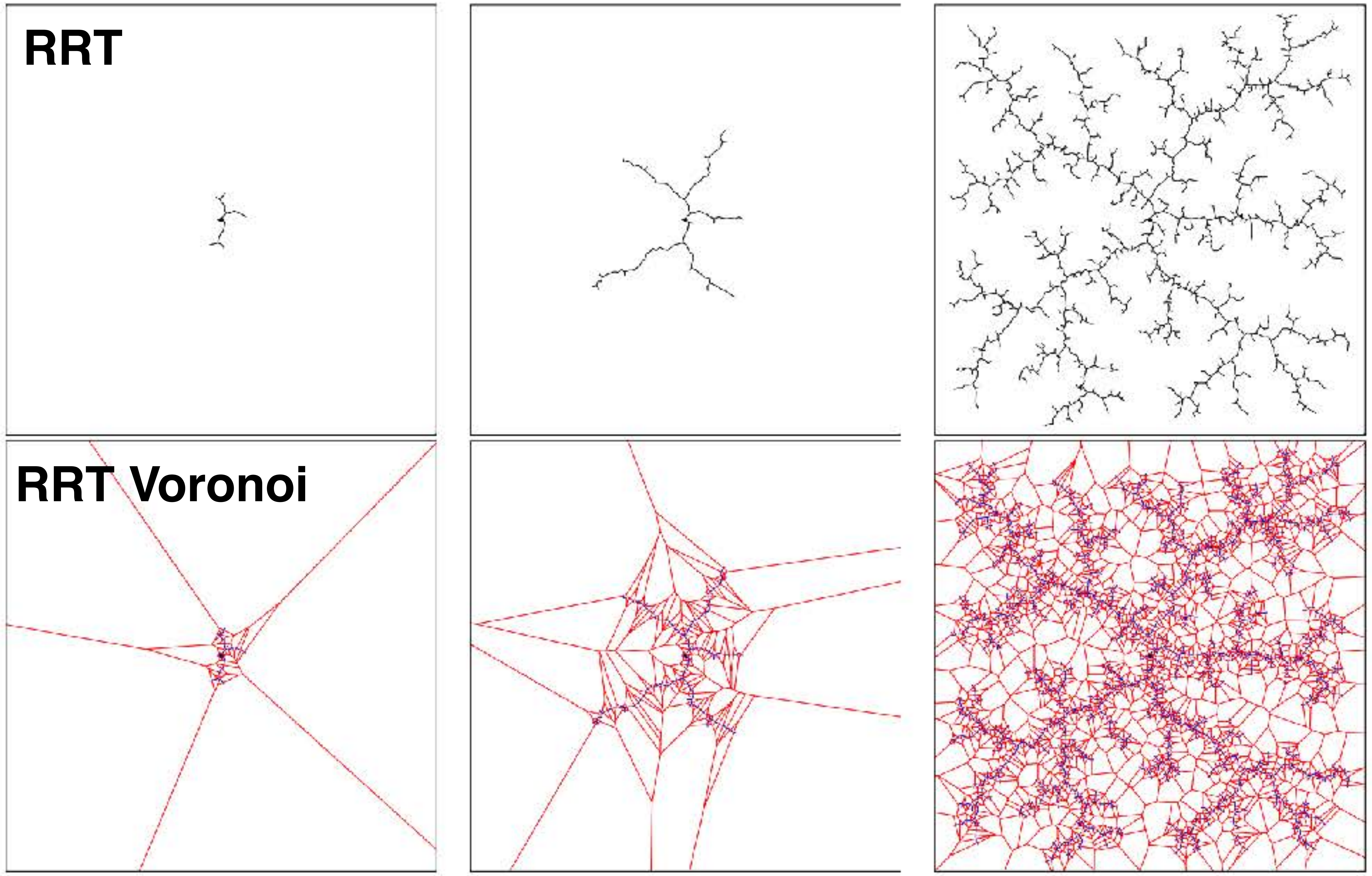
RRT-Connect animation



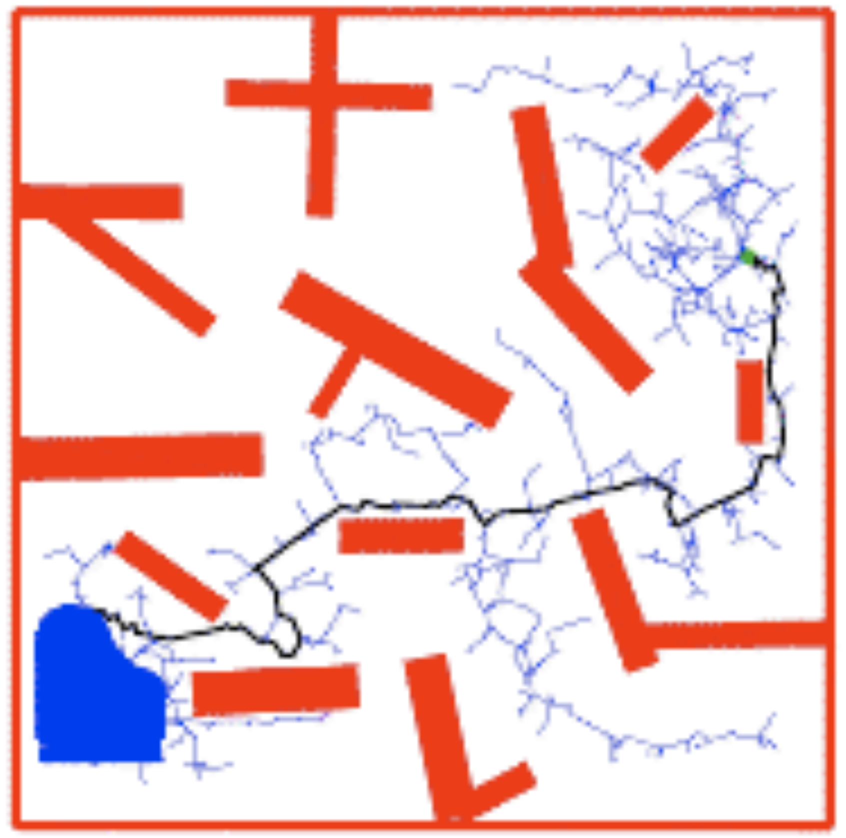
http://www.kuffner.org/james/plan/algorithm.php

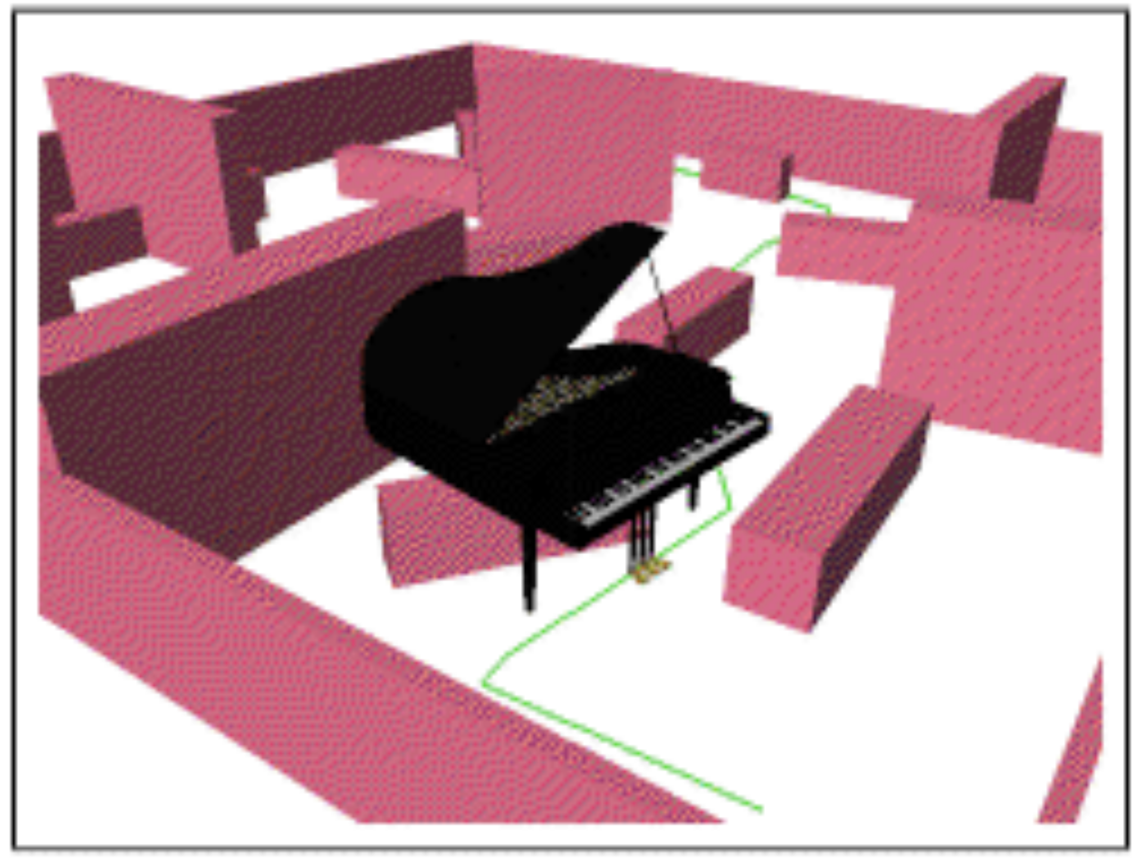
RRT Probabilistic Completeness

- RRTs converge to a uniform coverage of C-space as the number of samples increases
- Probability a vertex is selected for extension is proportional to its area in Voronoi diagram

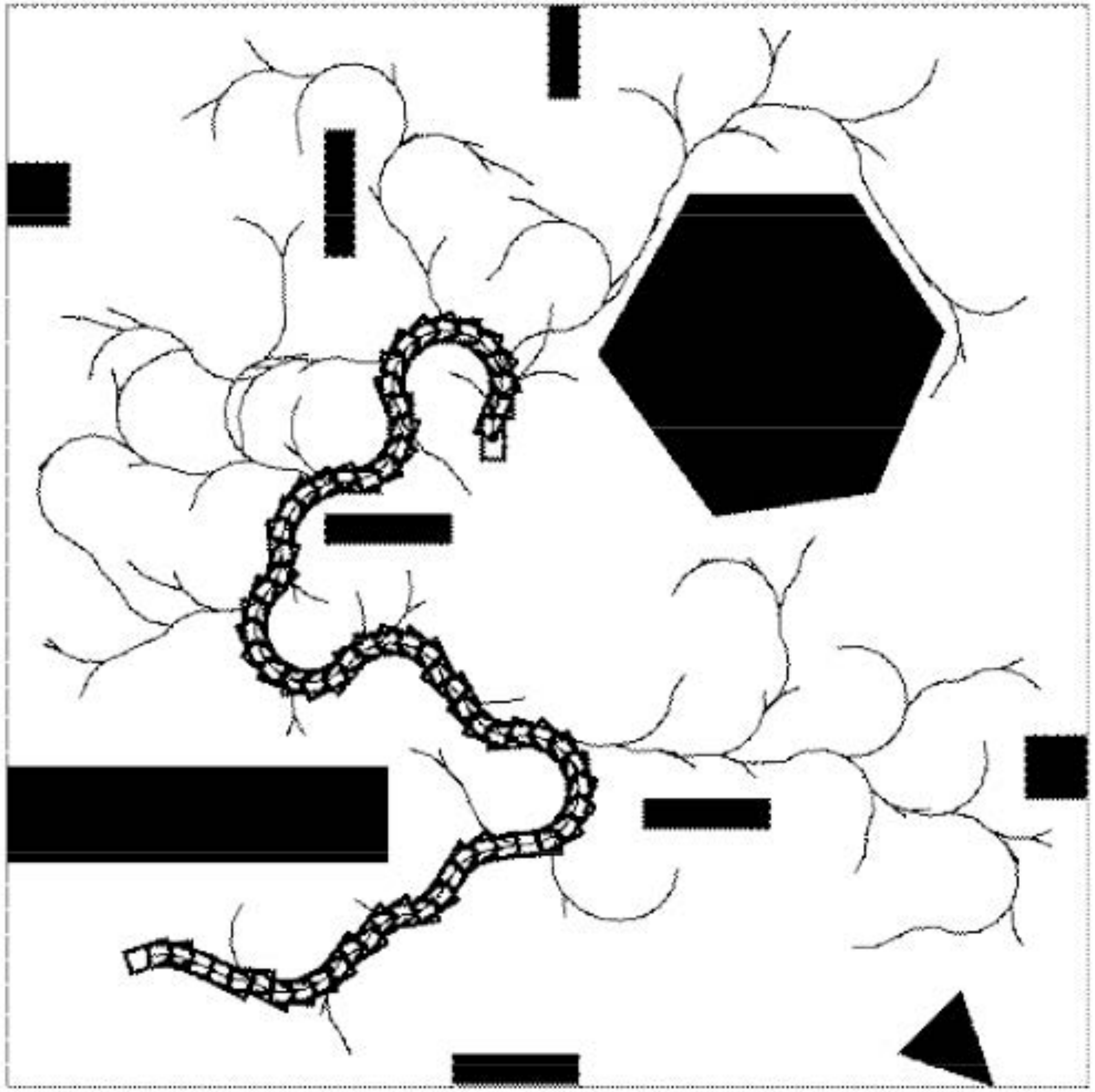


Piano Mover's Problem

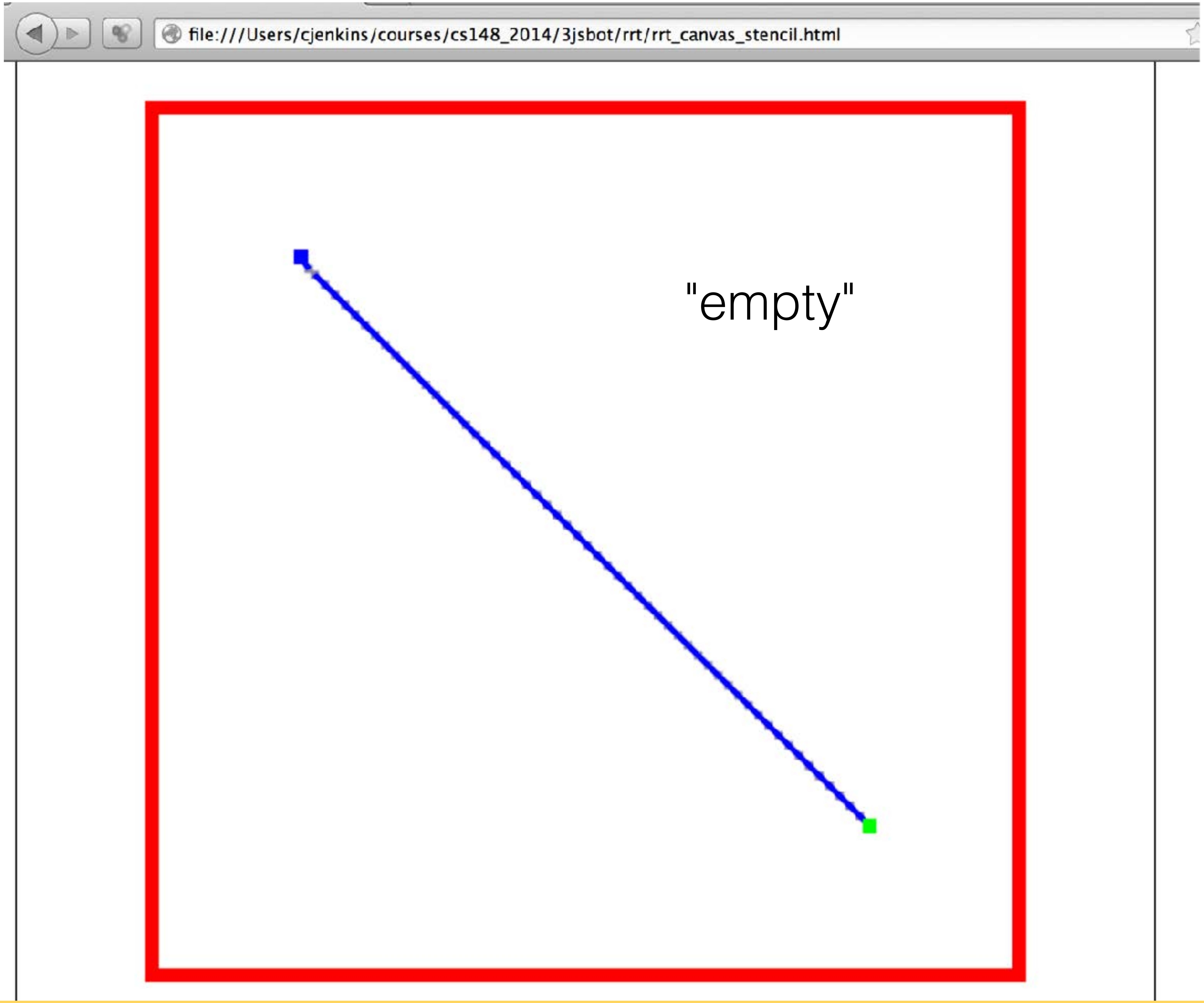


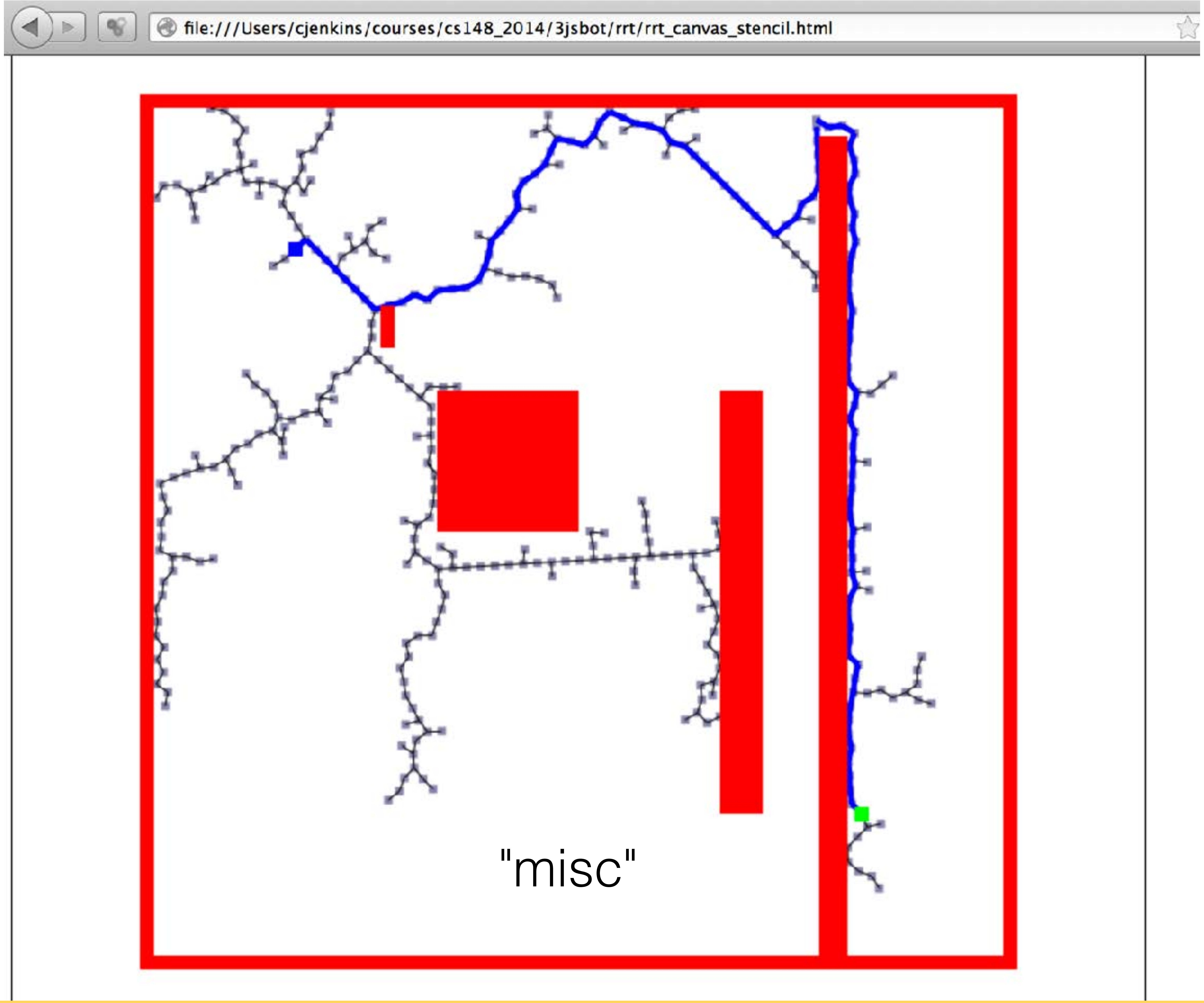


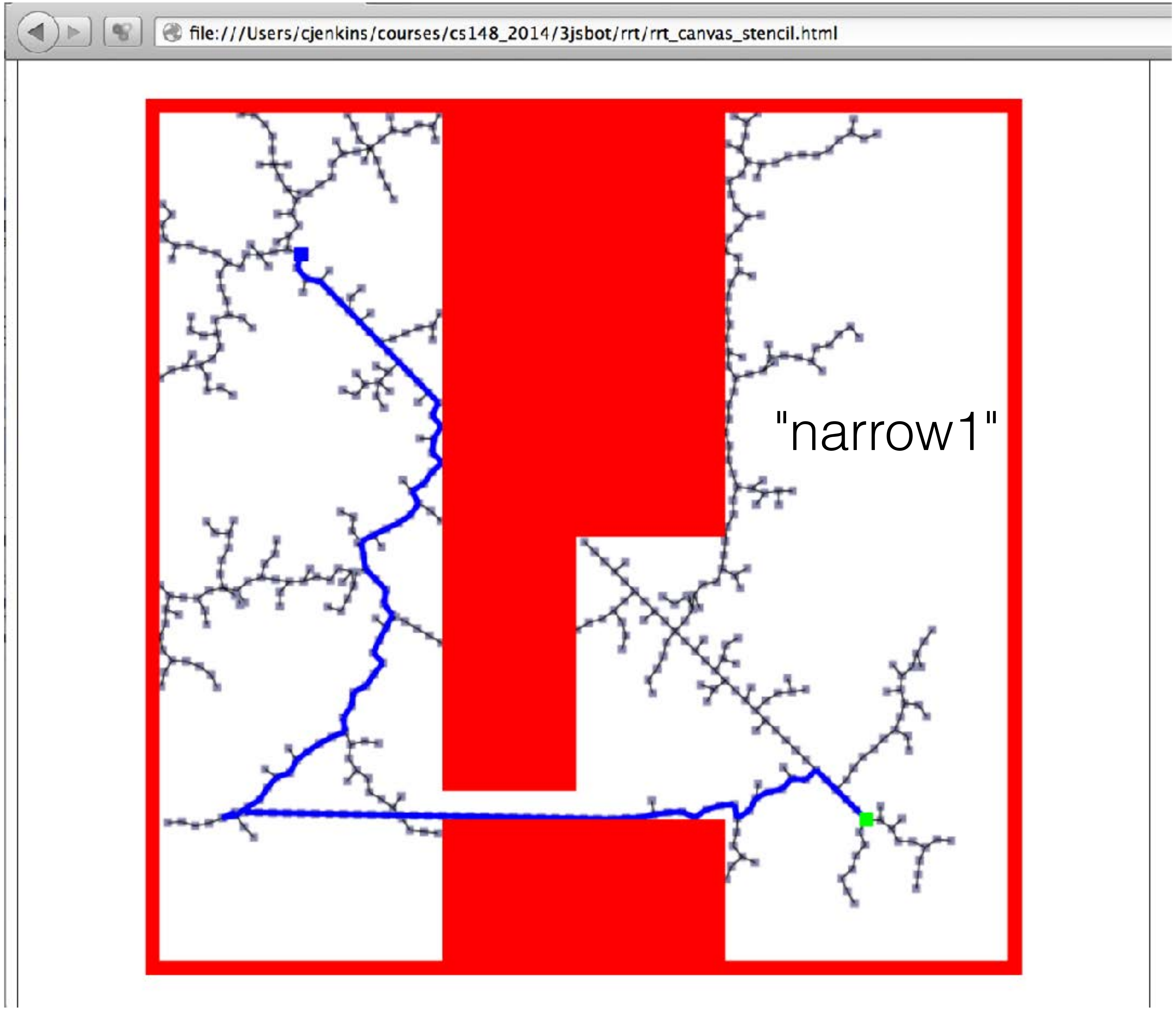
A Car-Like Robot



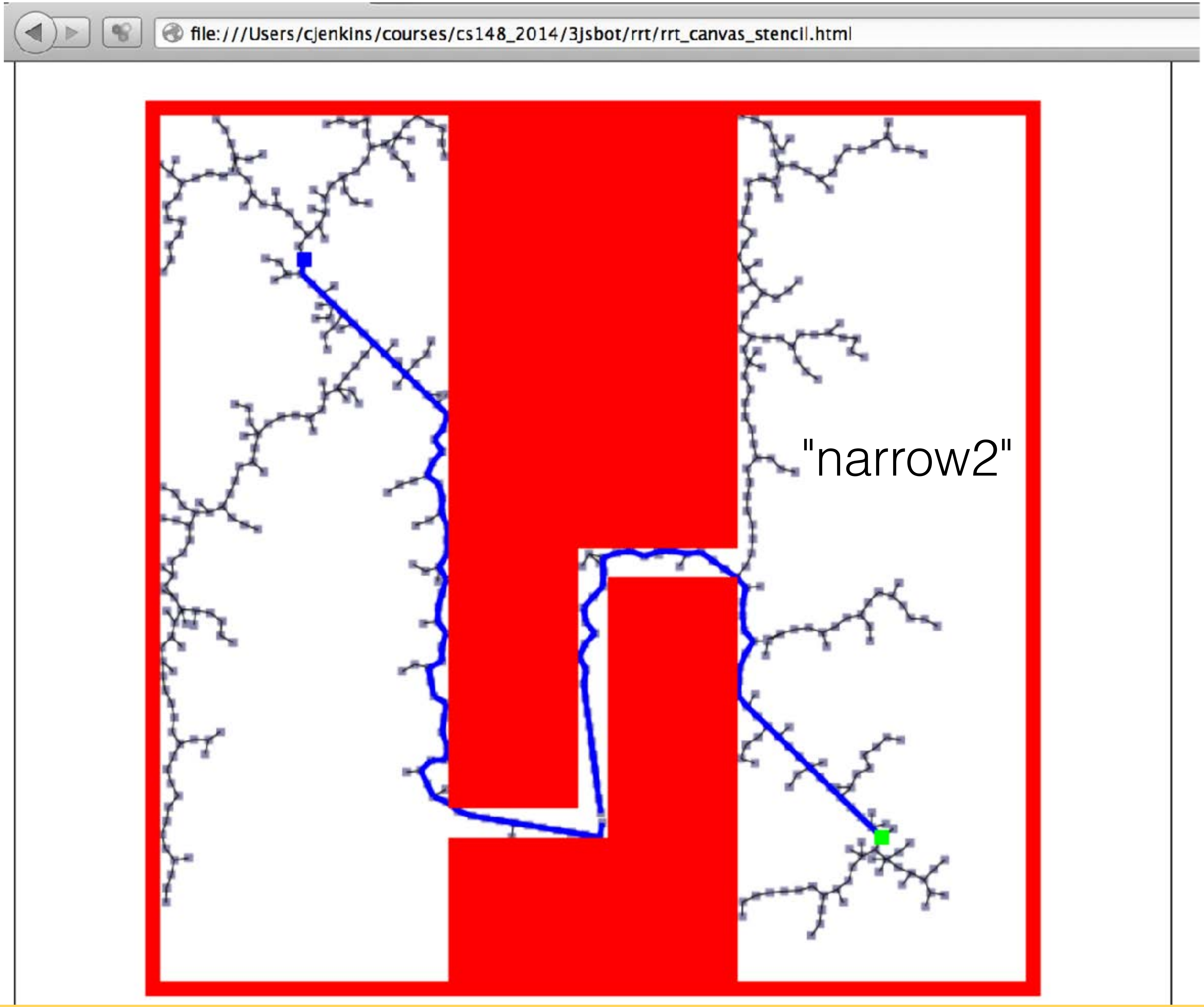
Canvas Stencil Examples

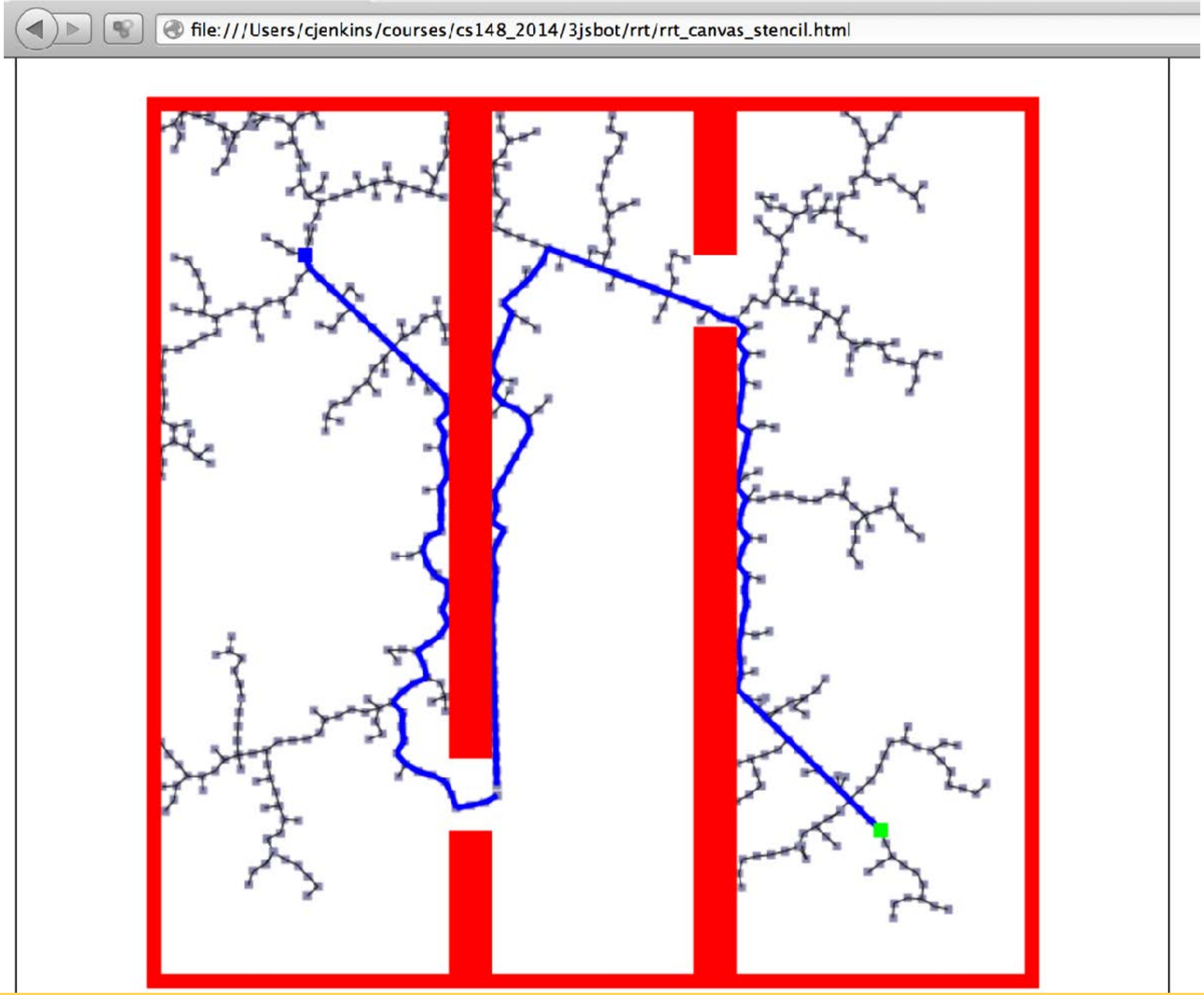

file:///Users/cjenkins/courses/cs148_2014/3jsbot/rrt/rrt_canvas_stencil.html
"empty"


file:///Users/cjenkins/courses/cs148_2014/3jsbot/rrt/rrt_canvas_stencil.html
"misc"



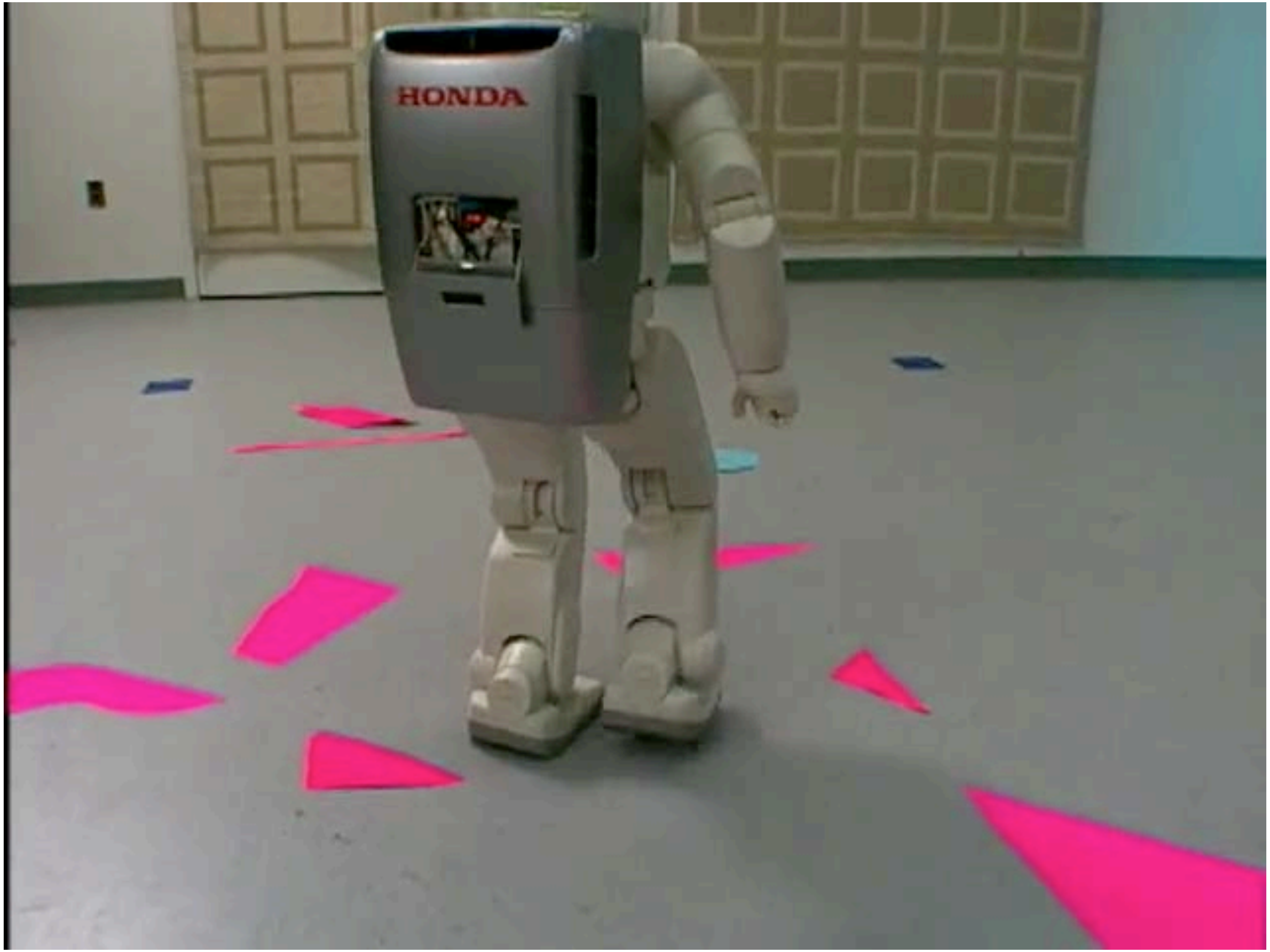
Slide borrowed from Michigan Robotics autorob.org


file:///Users/cjenkins/courses/cs148_2014/3jsbot/rrt/rrt_canvas_stencil.html
"narrow2"



"three_sections"

“We’ve made robot history”



Kuffner/Asimo Discovery Channel feature - https://www.youtube.com/watch?v=wtVmbiTfm0Q

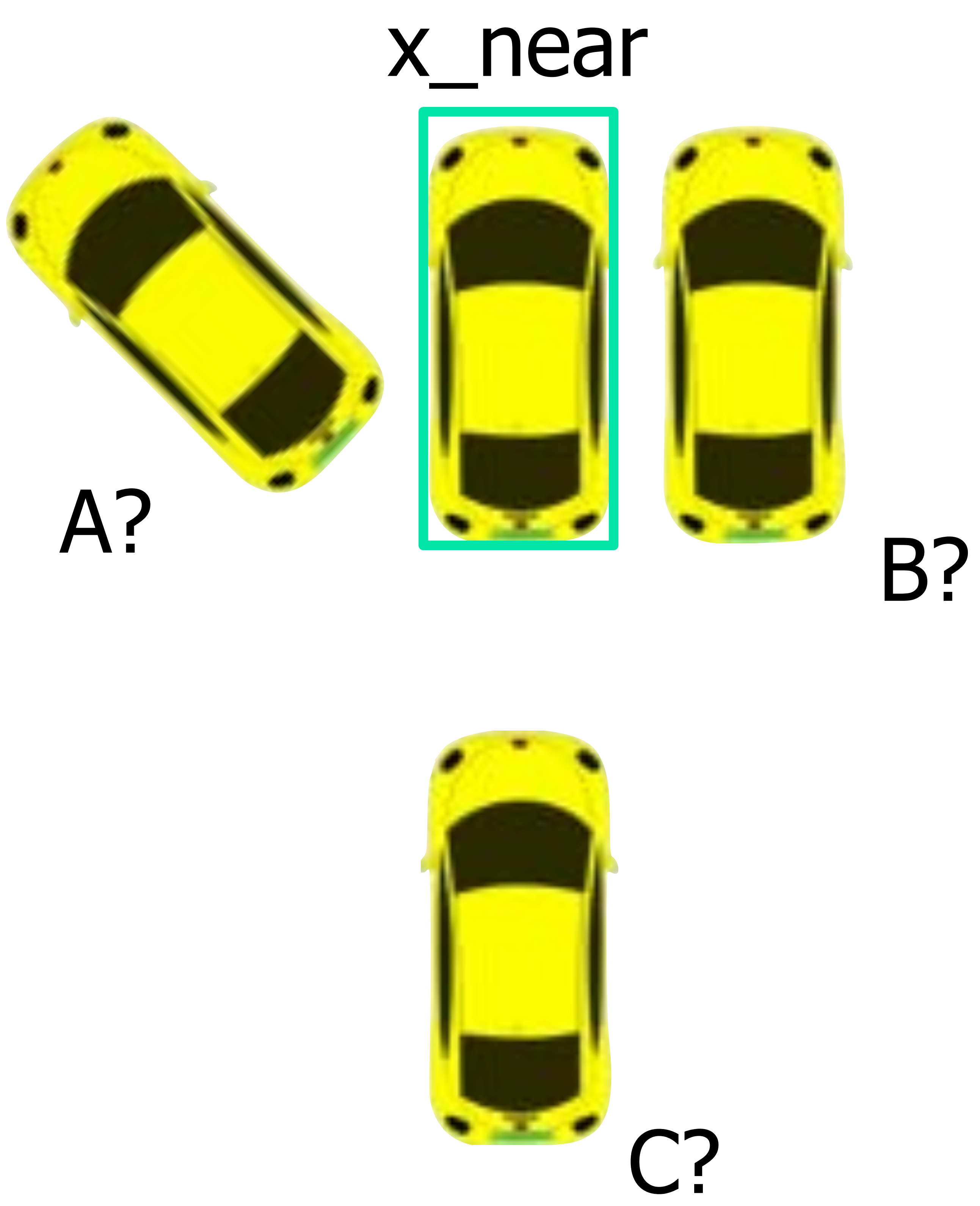
RRT Practicalities

- NEAREST_NEIGHBOR(x_{rand}, T): need to find (approximate) nearest neighbor efficiently
  - KD Trees data structure (upto 20-D) [e.g., FLANN]
  - Locality Sensitive Hashing

- SELECT_INPUT(x_{rand}, x_{near})
  - Two point boundary value problem
    - If too hard to solve, often just select best out of a set of control sequences. This set could be random, or some well chosen set of primitives.

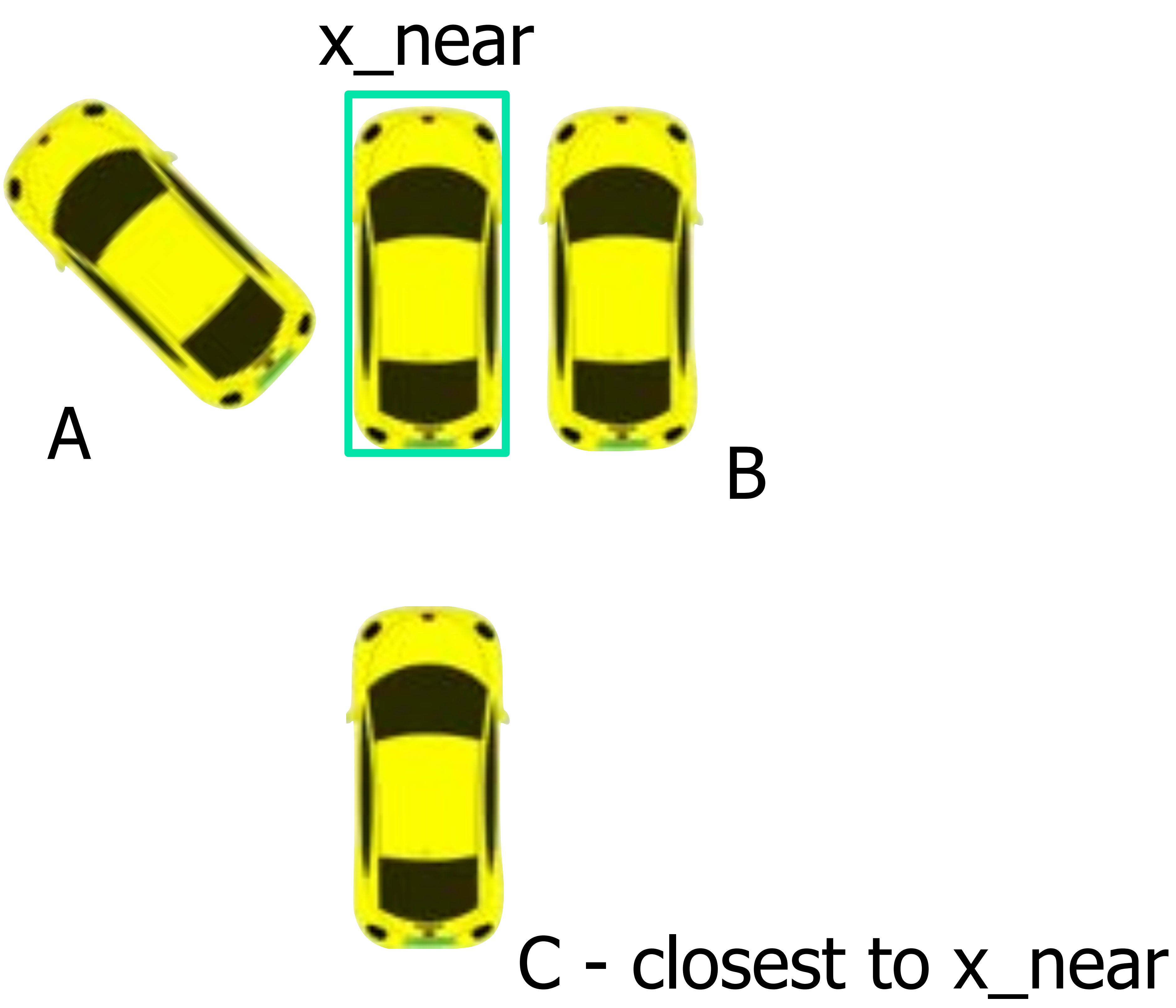
RRT Extension

- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem



RRT Extension

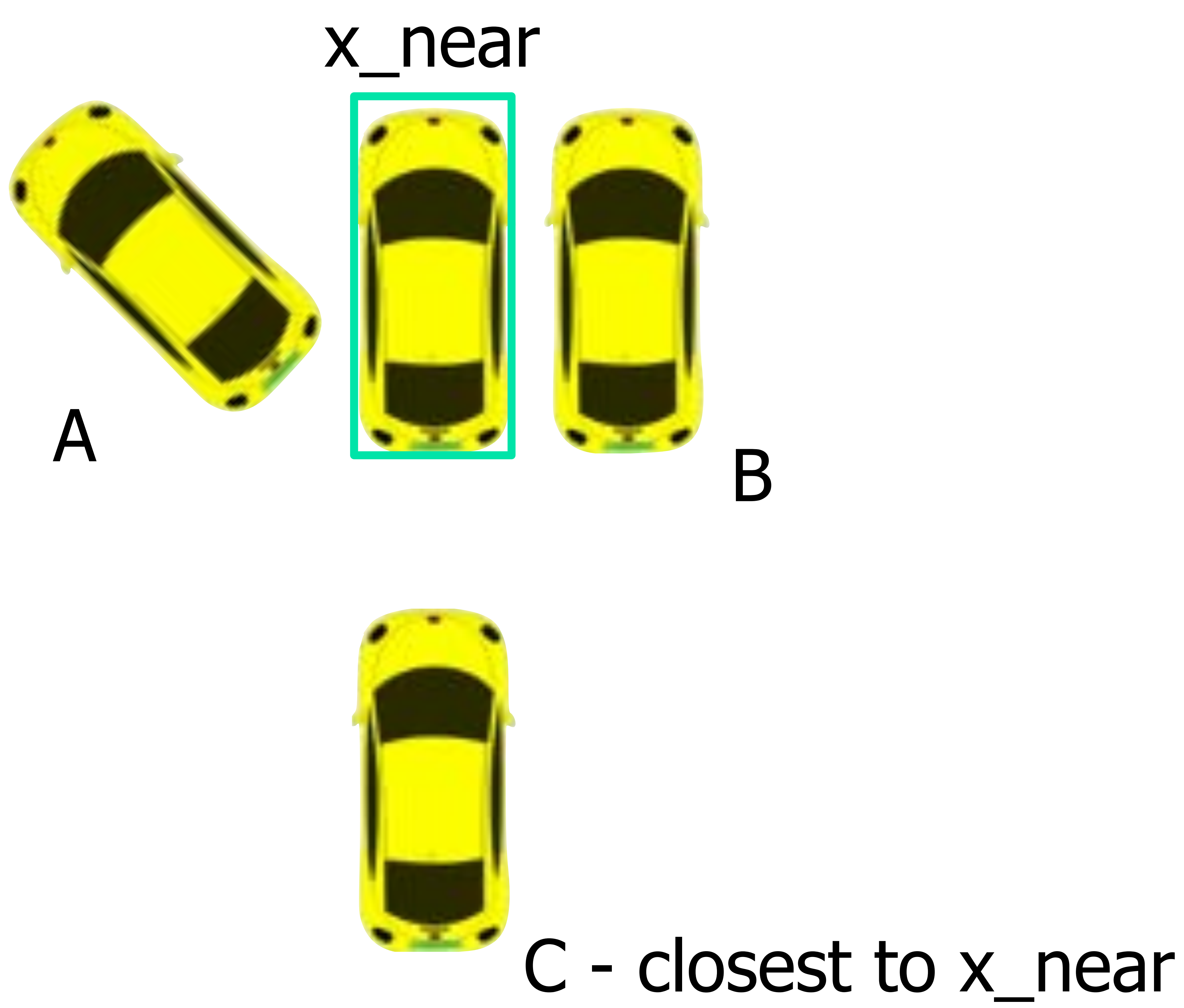
- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem

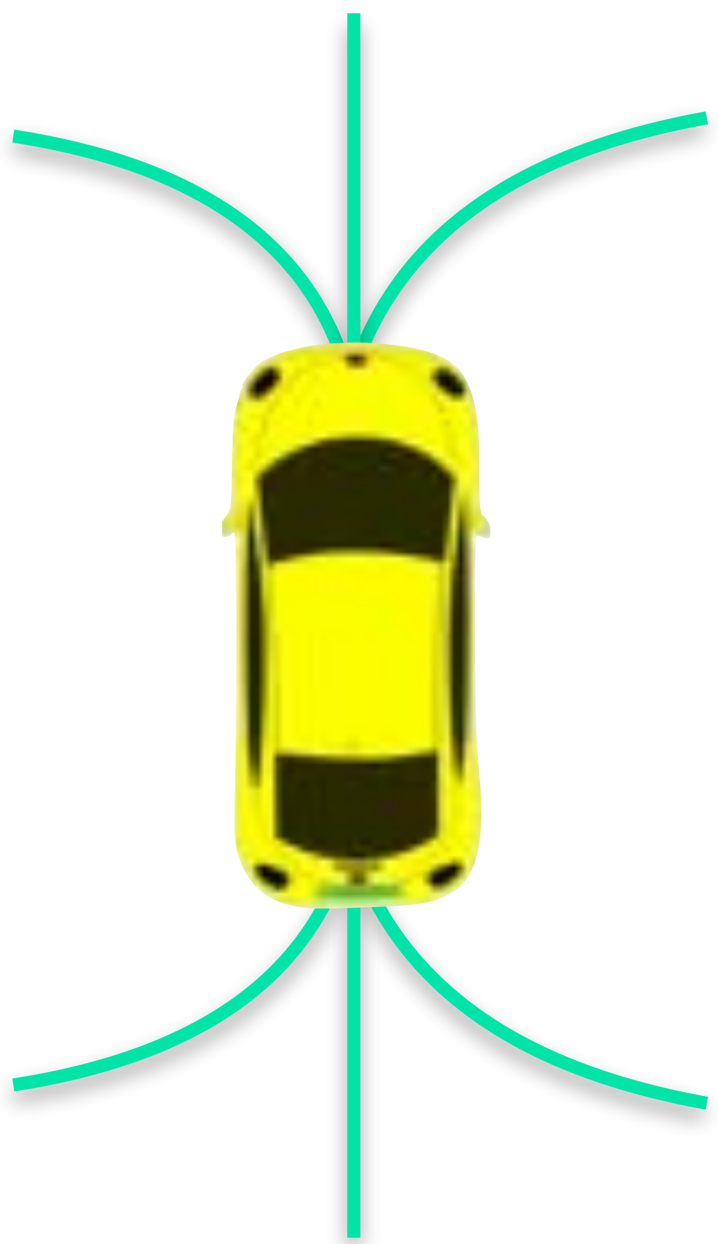


Slide borrowed from Dieter Fox

RRT Extension

- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem

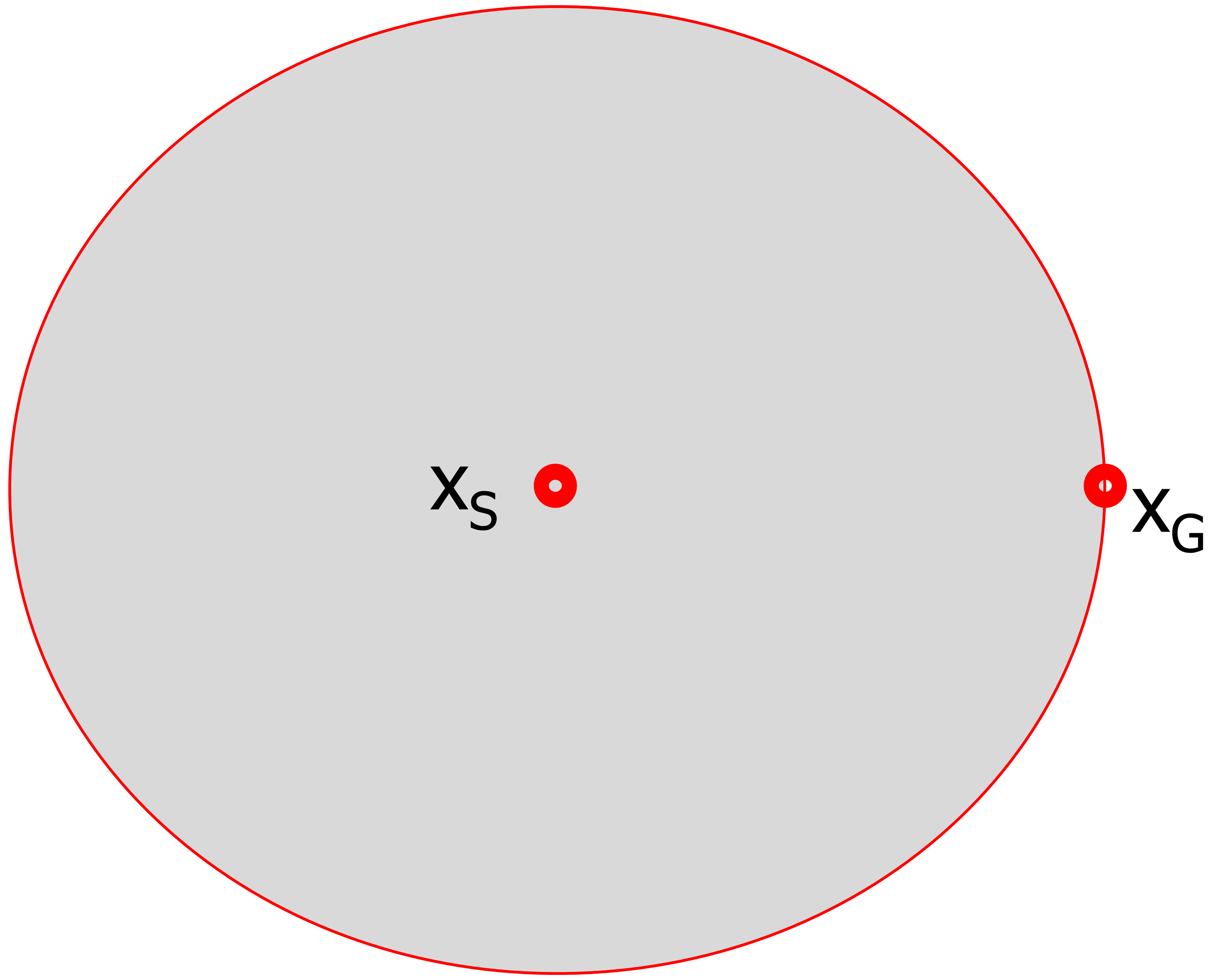




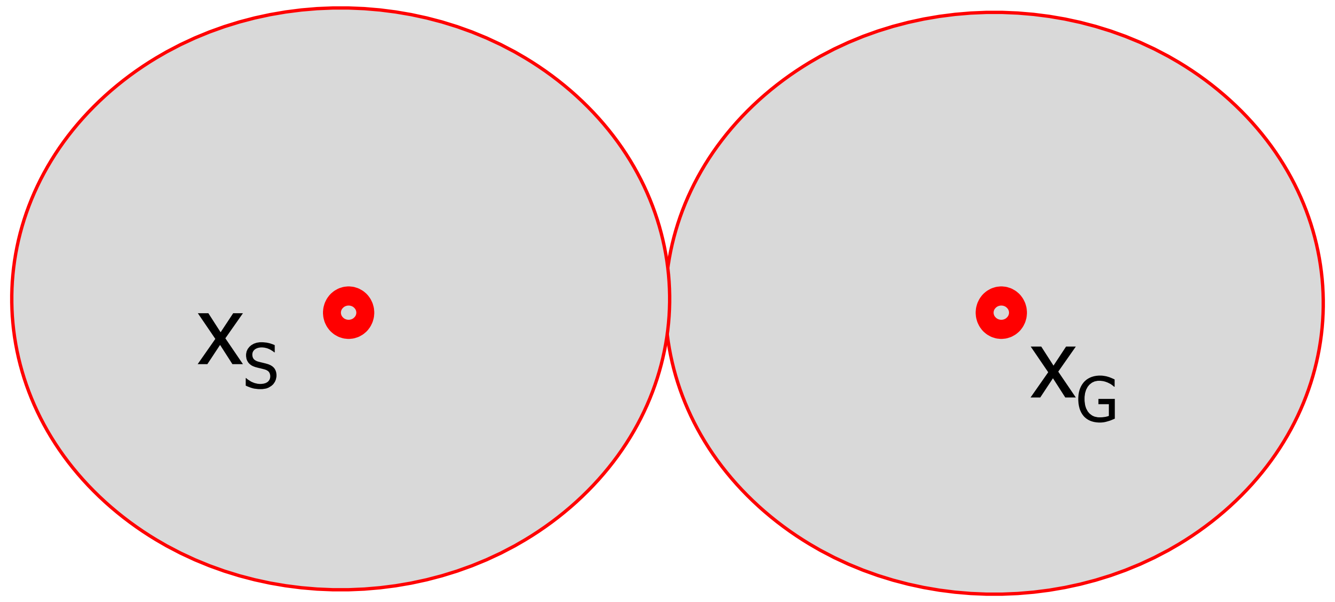
Slide borrowed from Dieter Fox

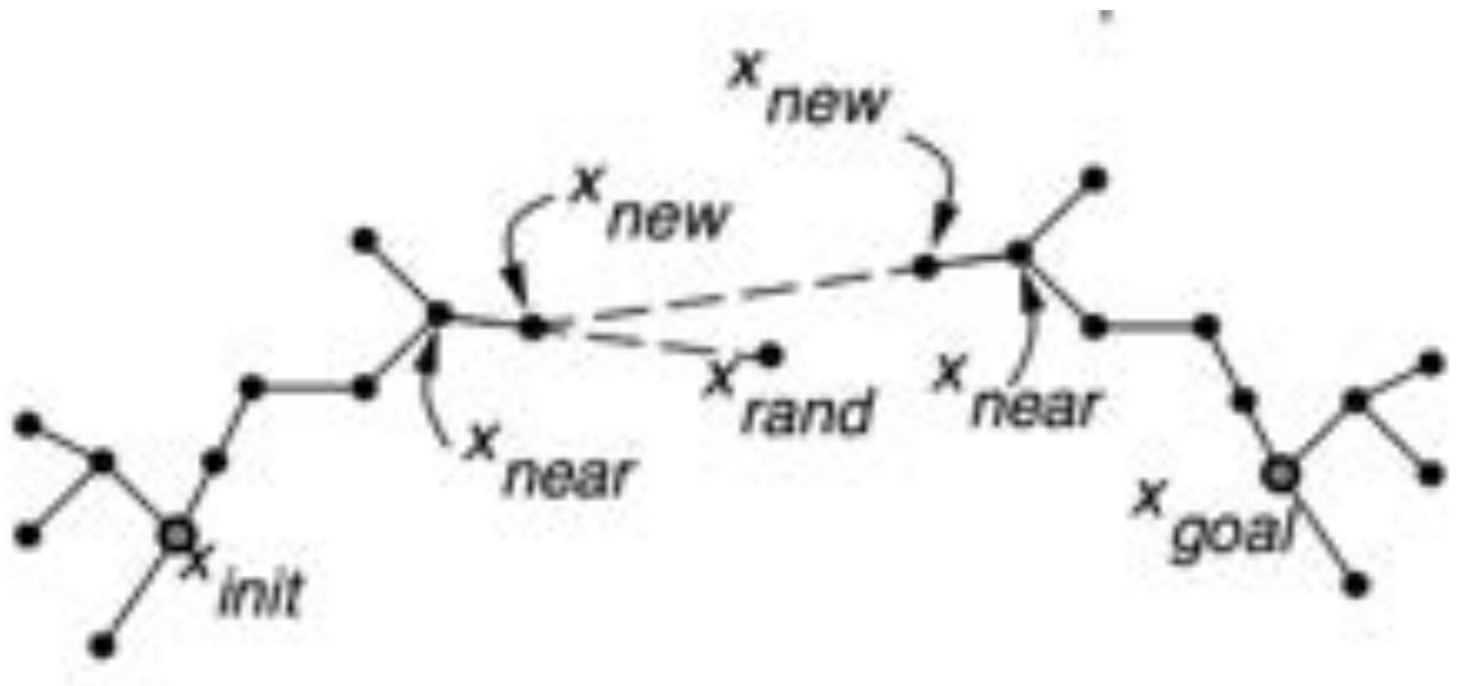
Bi-directional RRT

- Volume swept out by unidirectional RRT:



- Volume swept out by bi-directional RRT:

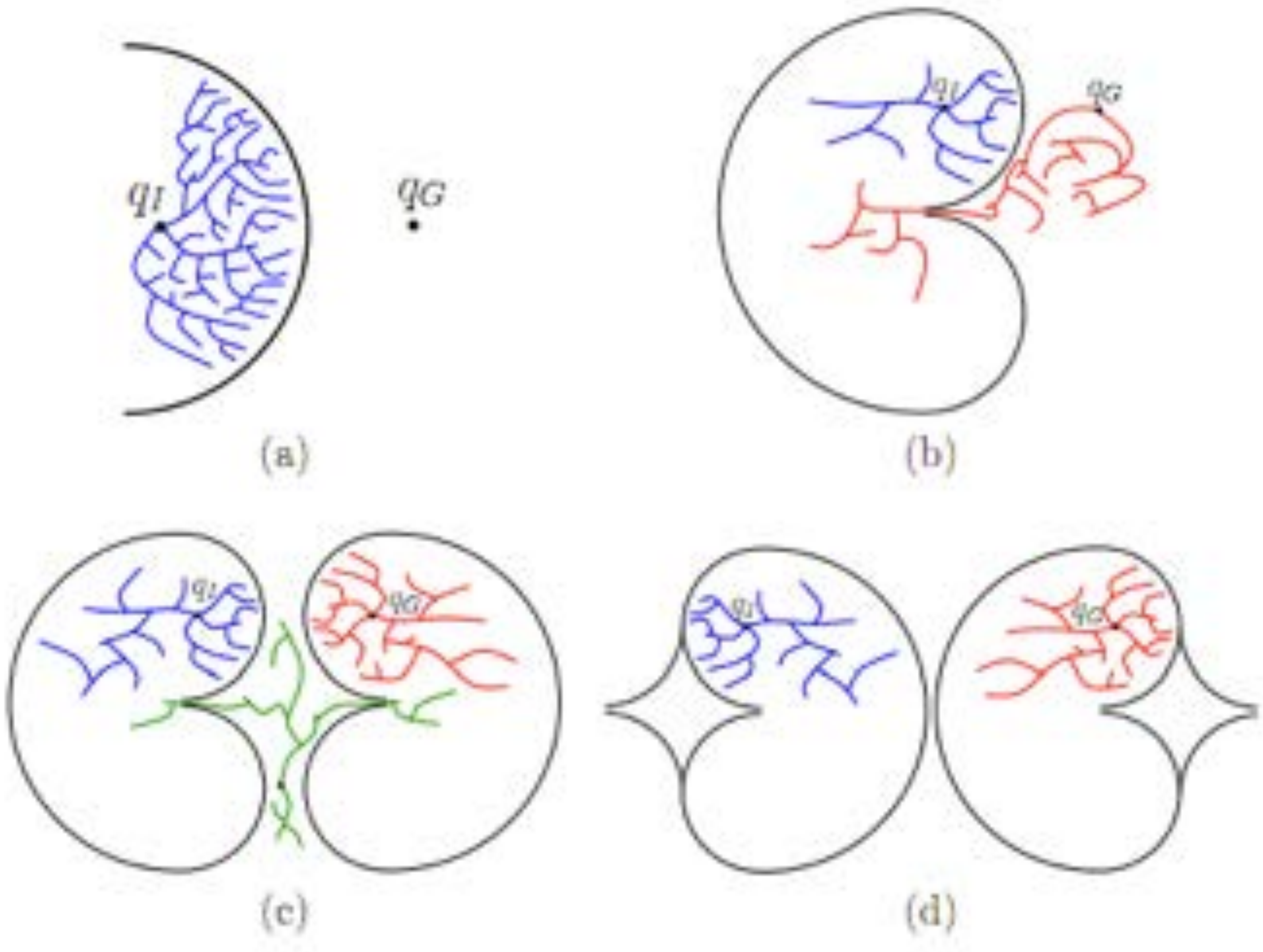




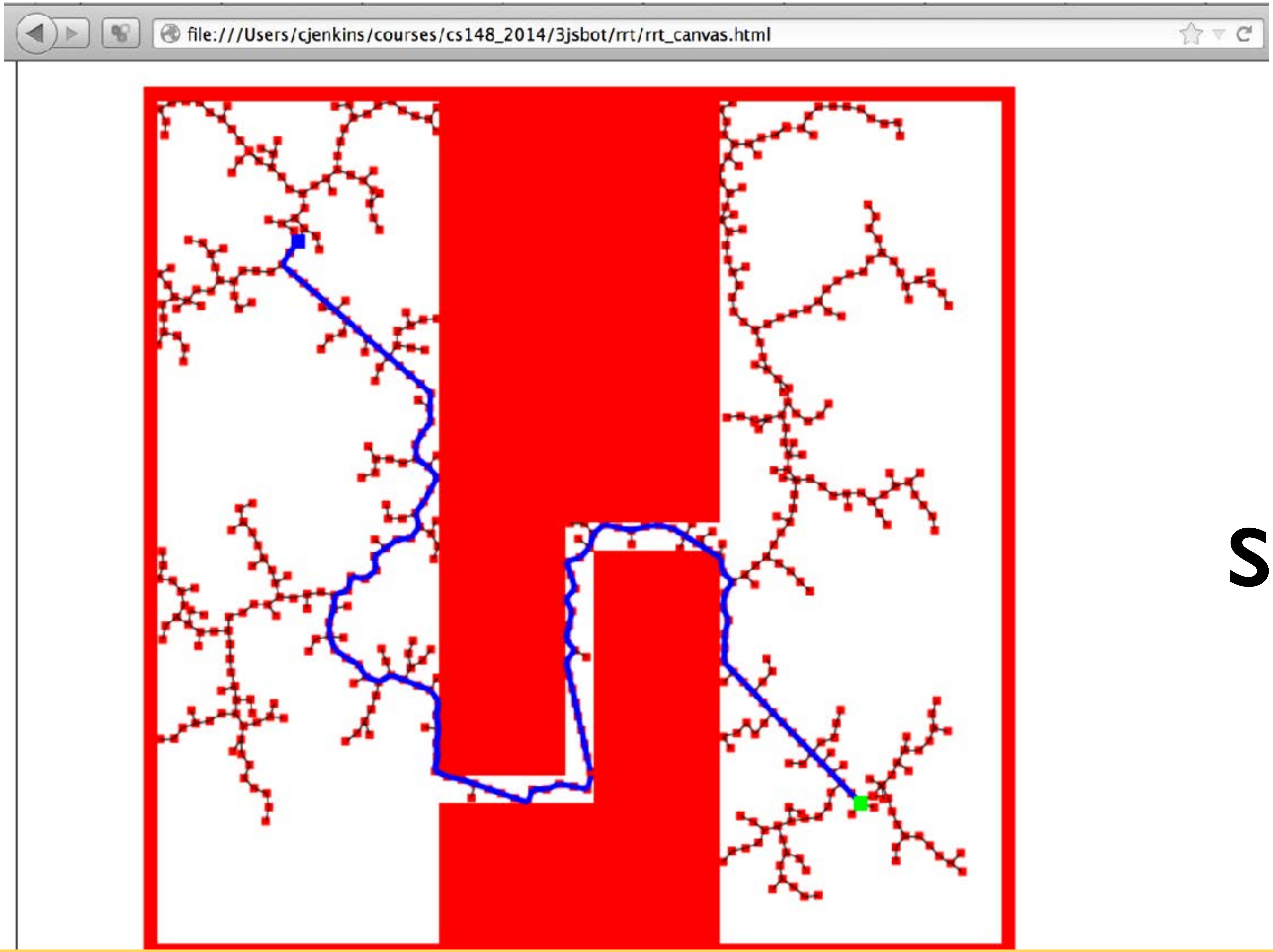
- Difference more and more pronounced as dimensionality increases

Multi-directional RRT

- Planning around obstacles or through narrow passages can often be easier in one direction than the other



RRTs can take a lot of time...



Is there a simpler way?

RRT*

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\};\ E \leftarrow \emptyset;$
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**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT*}}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})\ ;$
    $V \leftarrow V \cup \{x_{\text{new}}\};$
    $x_{\text{min}} \leftarrow x_{\text{nearest}};\ c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}));$
    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 $G = (V, E);$

FIND x_new

ADD x_new to G
FIND neighbors to x_new in the G

FIND edge to x_new from neighbors with least cost
ADD that to G

REWIRE the edges in the neighborhood if any least cost path exists from the root to the neighbors via x_new

Source: Karaman and Frazzoli

RRT*

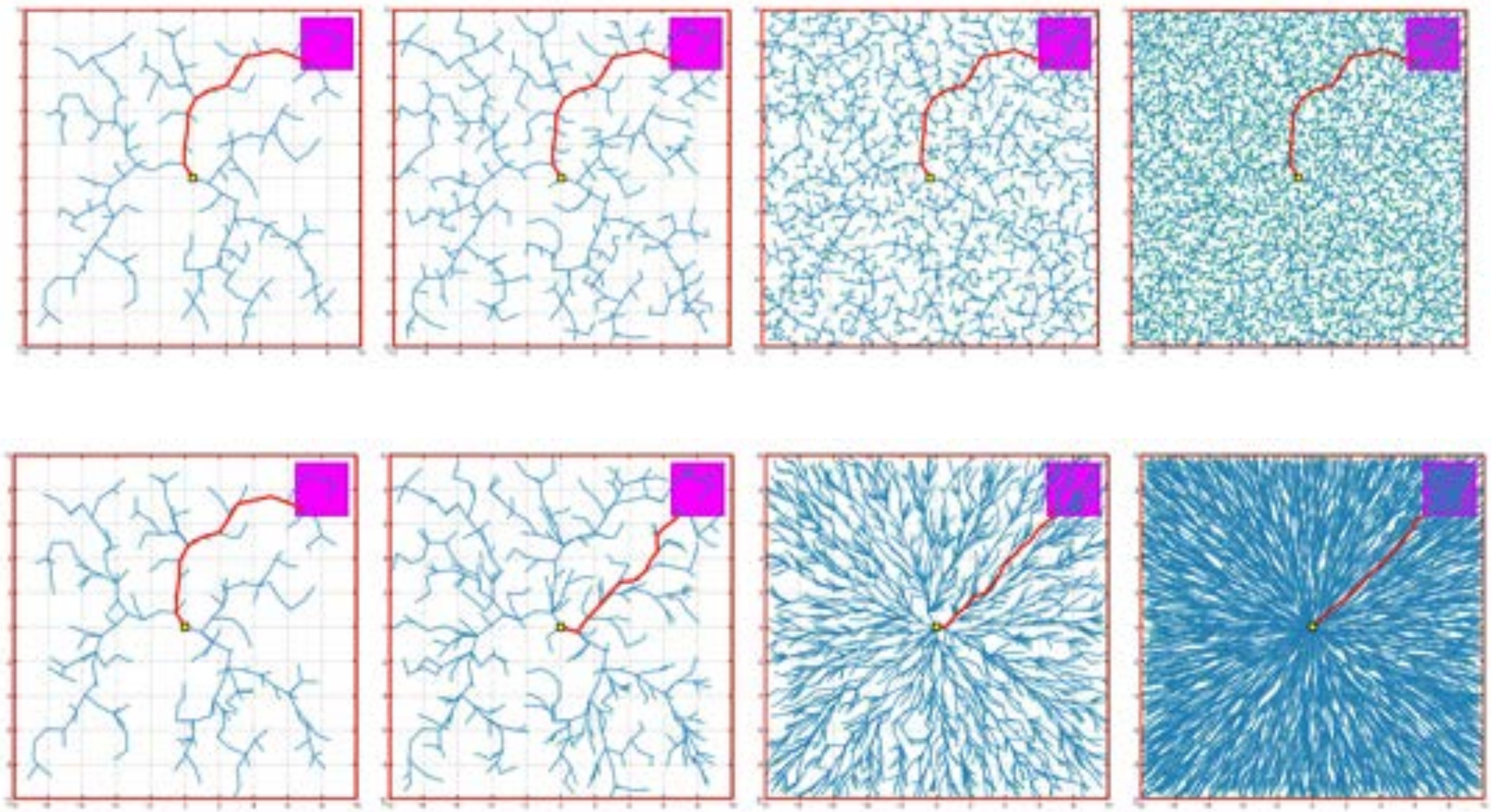
- Asymptotically optimal
- Main idea:
  - Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent

Demonstration - https://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/

RRT*

RRT

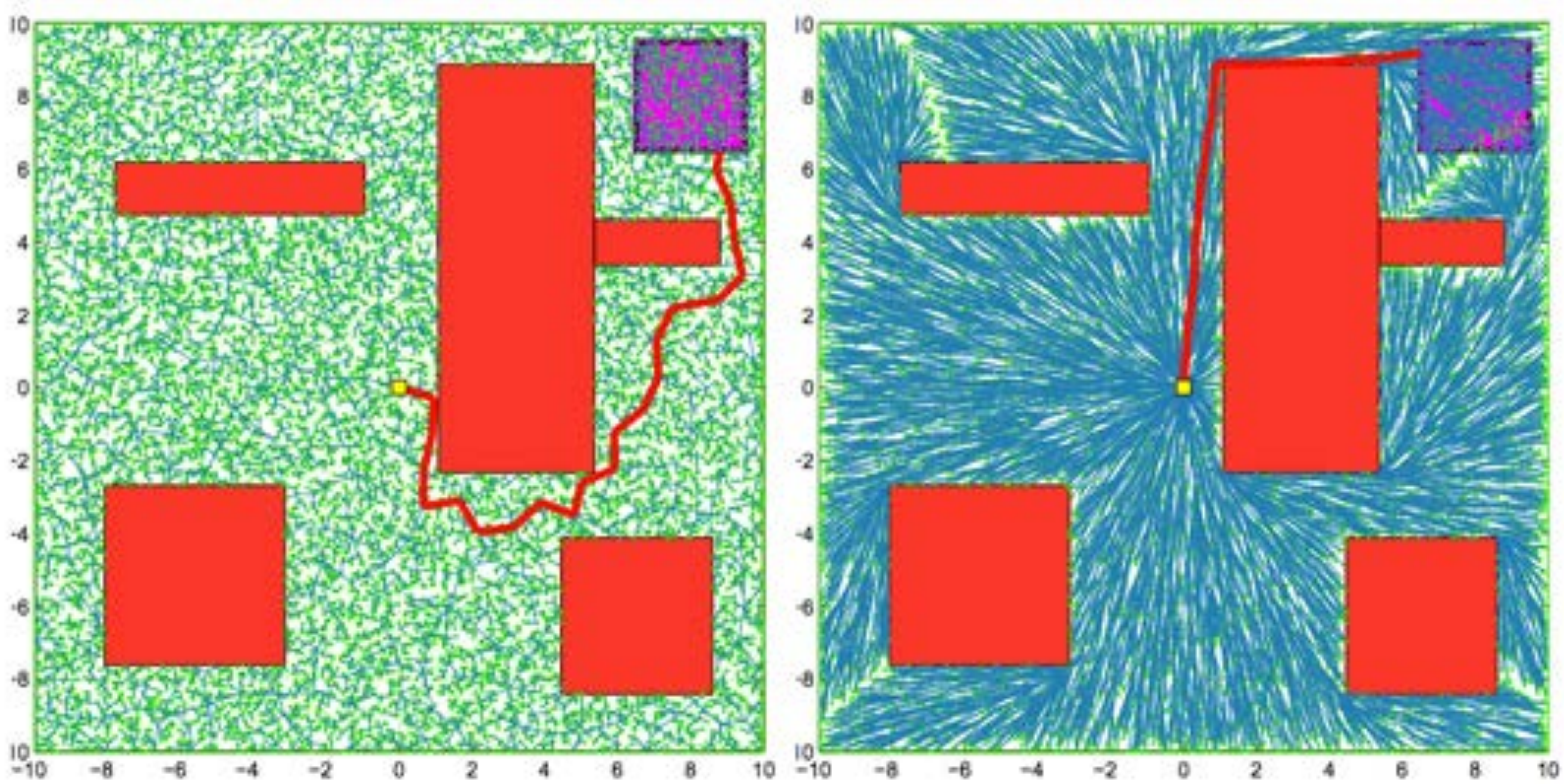
RRT*



Source: Karaman and Frazzol

RRT*

RRT RRT*



Source: Karaman and Frazzoli

Smoothing

Randomized motion planners tend to find not so great paths for execution: very jagged, often much longer than necessary.

→ In practice: do smoothing before using the path

- Shortcutting:
  - along the found path, pick two vertices x_{t1}, x_{t2} and try to connect them directly (skipping over all intermediate vertices)
- Nonlinear optimization for optimal control
  - Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

Additional Resources

- Marco Pavone (http://asl.stanford.edu/):
  - Sampling-based motion planning on GPUs: https://arxiv.org/pdf/1705.02403.pdf
  - Learning sampling distributions: https://arxiv.org/pdf/1709.05448.pdf
- Sidd Srinivasa (https://personalrobotics.cs.washington.edu/)
  - Batch informed trees: https://robotic-esp.com/code/bitstar/
  - Expensive edge evals: https://arxiv.org/pdf/2002.11853.pdf
  - Lazy search: https://personalrobotics.cs.washington.edu/publications/mandalika2019gls.pdf
- Michael Yip (https://www.ucsdarclab.com/)
  - Neural Motion Planners: https://www.ucsdarclab.com/neuralplanning
- Lydia Kavraki (http://www.kavrakilab.org/)
  - Motion in human workspaces: http://www.kavrakilab.org/nsf-nri-1317849.html

Next Lecture

Planning - V - Collision Detection