COM SCI 188

Intro to Robotics

Lecture 10

Yuchen Cui

Winter 2026

Agenda

- Announcements
- Recap: SLAM

- TODAY: Motion Planning

Announcements

- Coding Assignment 2 due Next Monday
- Problem Set 3 out
  - Covers all materials till midterm
  - You'll need next week's lectures…

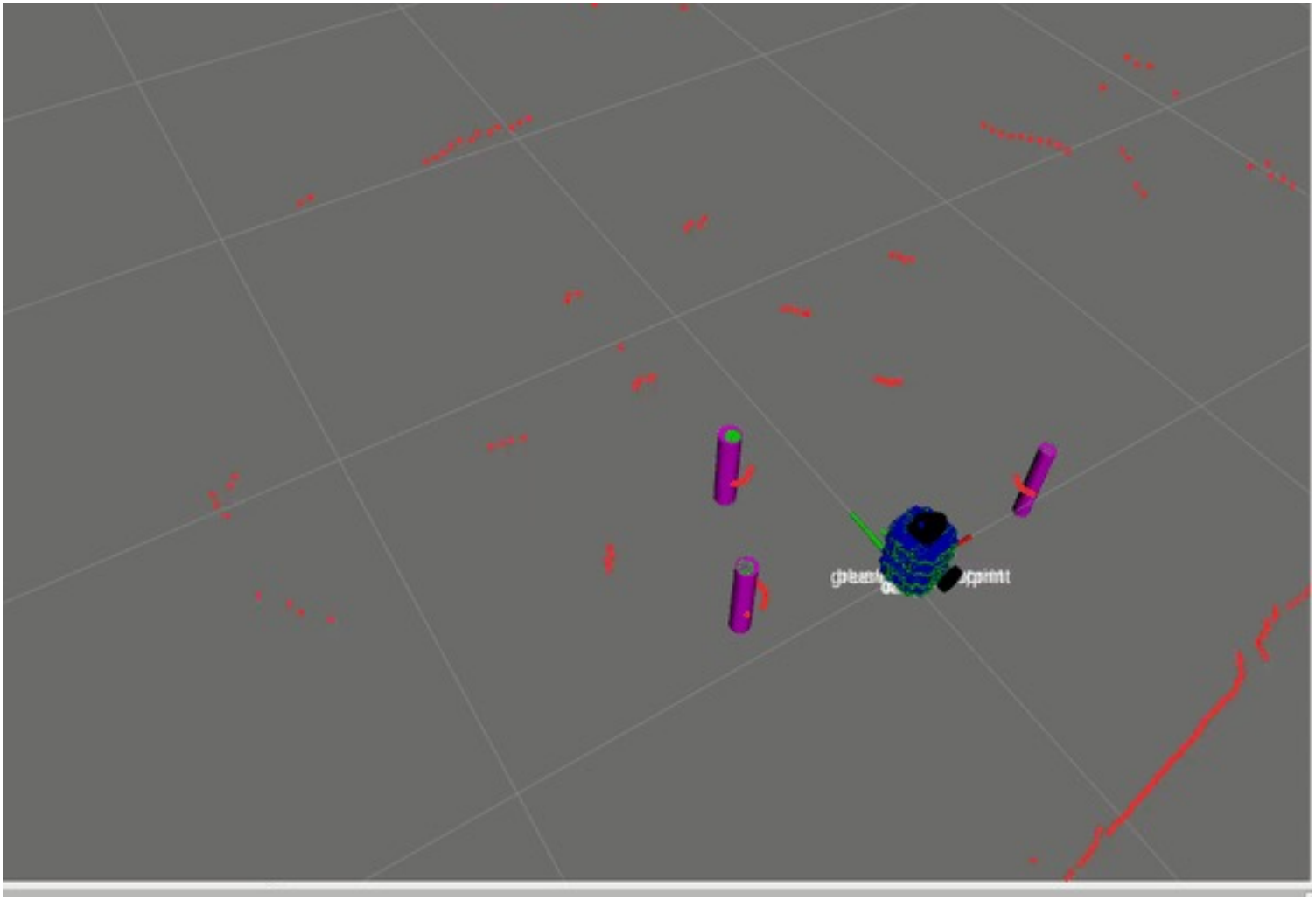
Recap: SLAM Algorithm

Algorithm 9.2: SLAM

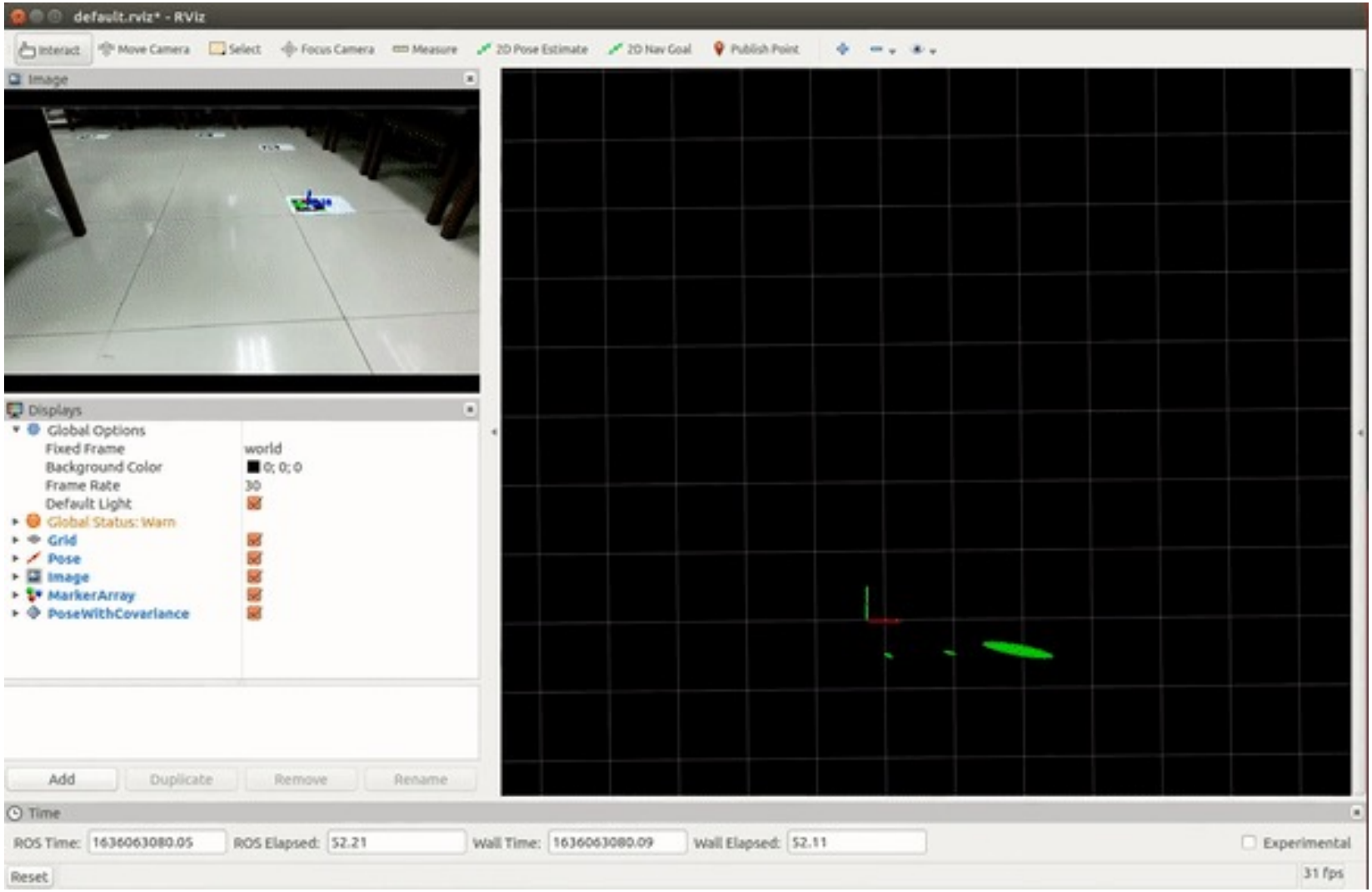
```
matrix m ← partial map          // Current map
matrix p                        // Perception map
matrix e                        // Expected map
coordinate c ← initial position // Current position
coordinate n                    // New position
coordinate array T              // Set of test positions
coordinate t                    // Test position
coordinate b ← none             // Best position
1: loop
2:     move a short distance                          (Move)
3:     n ← odometry(c)          // New position based on odometry   (Measure)
4:     p ← analyze sensor data                        (Measure)

5:     for every t in T         // T is the positions around n      (Model)
6:         e ← expected(m, t)   // Expected map at test position    (Model)
7:         if compare(p,e) better than b                (Correct)
8:             b ← t            // Best test position so far        (Correct)

9:     n ← b                    // Replace new position by best position   (Update)
10:    m ← update(m,p,n)        // Update map based on new position        (Update)
11:    c ← n                    // Current position is new position        (Update)
```



https://www.youtube.com/watch?v=ZHGrI5ZNQc0 https://github.com/ZhehaoLu/LiDAR_ArUco_EKFSLAM



https://github.com/ZhehaoLu/LiDAR_ArUco_EKFSLAM

Motion Planning

Slides Adapted from Stanford EE227/CS227A: Robot Perception
credit: Shuran Song

What is Motion Planning?

a.k.a. **path planning**

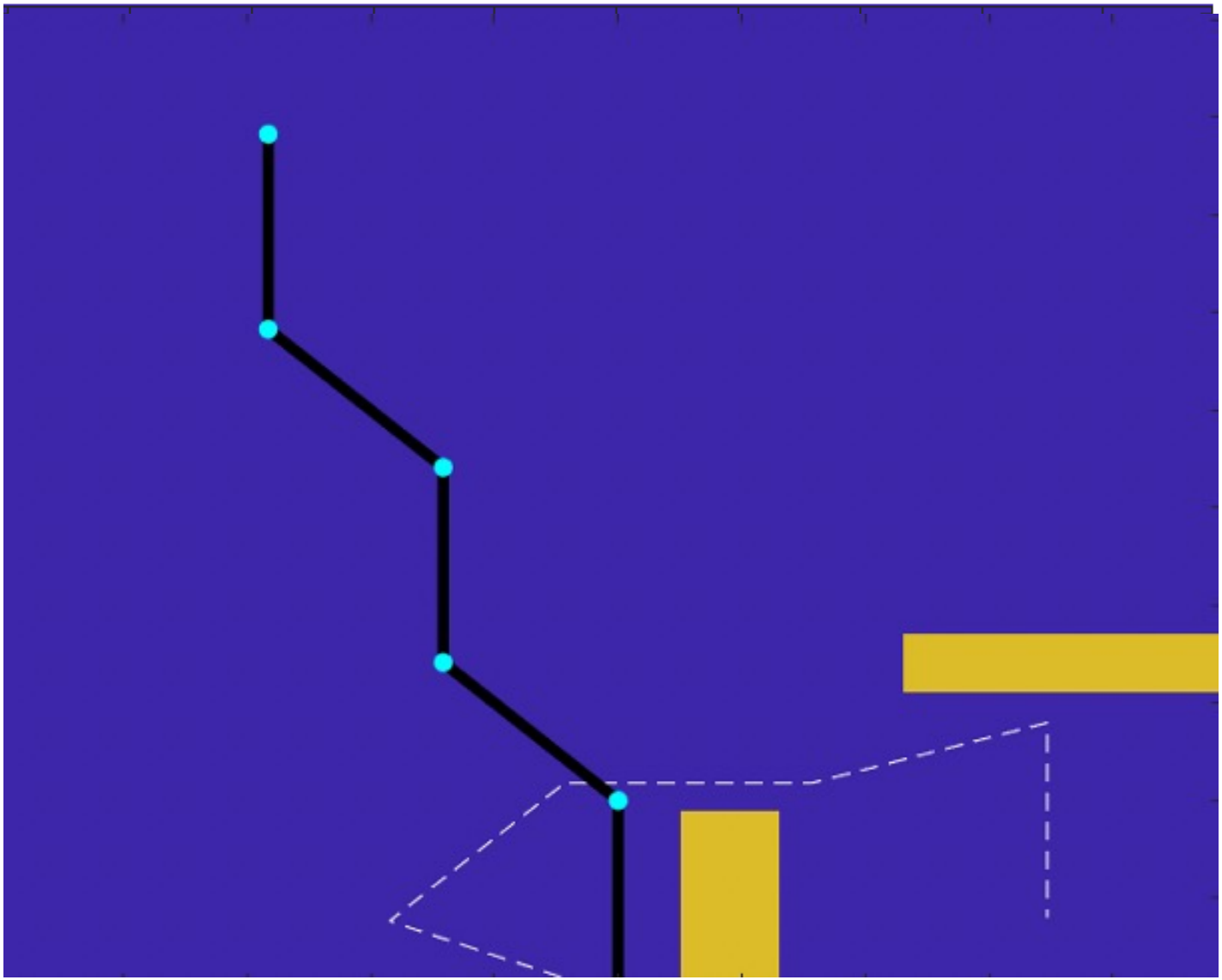
a.k.a. **the navigation problem**

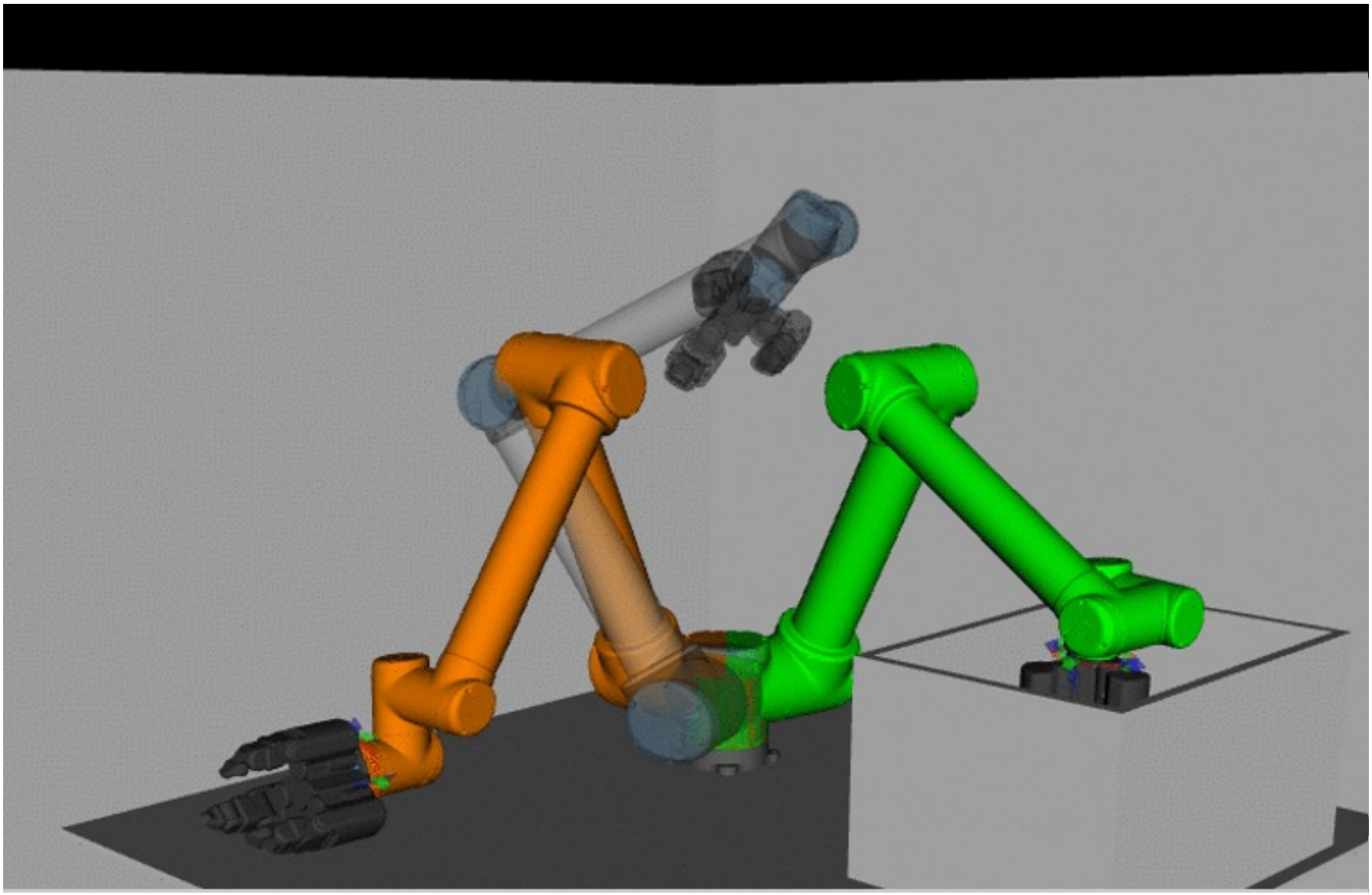
a.k.a. **piano mover's problem**

To find a sequence of valid configurations that moves the object from the source to destination.

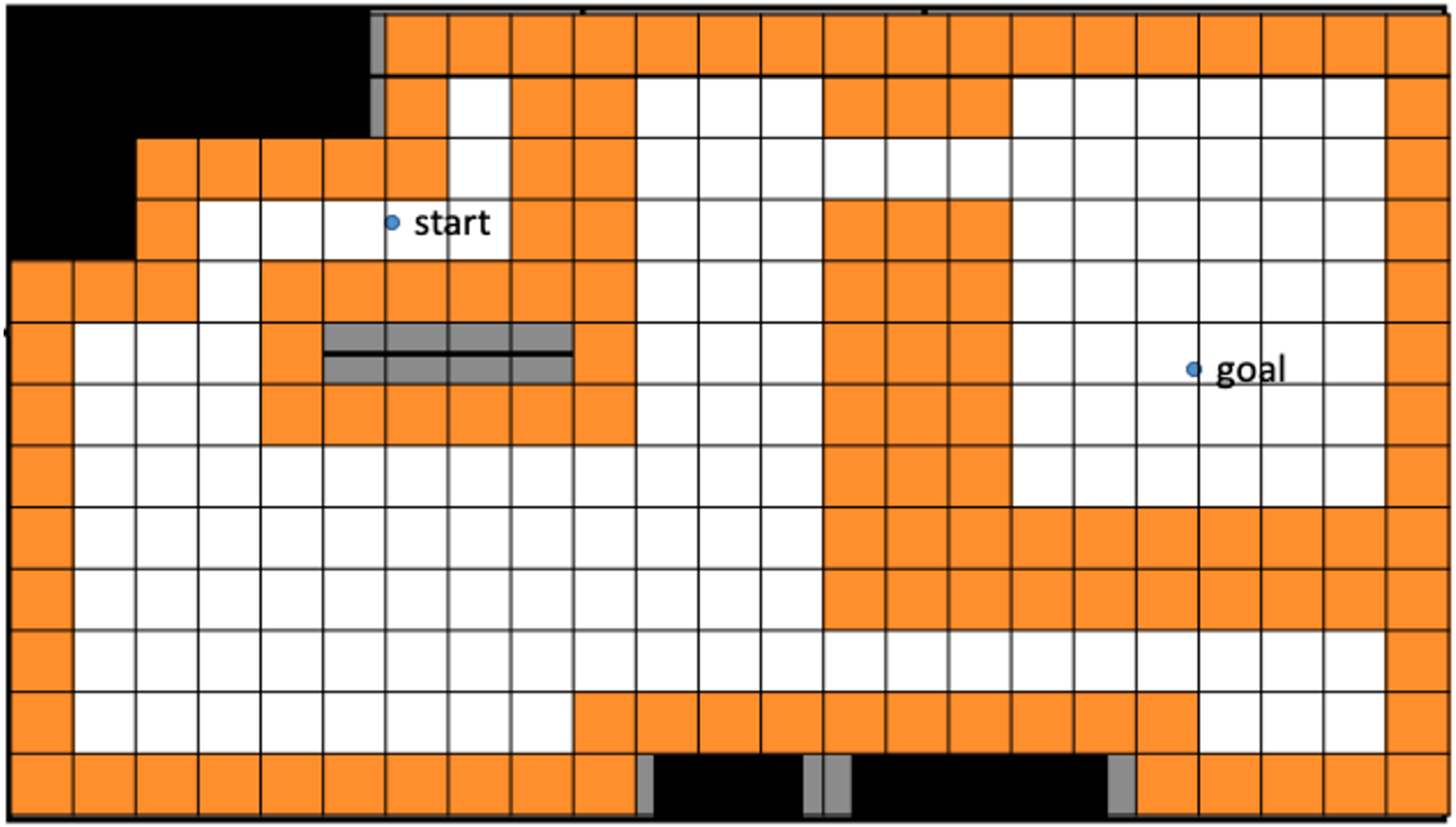
How something moves from A to B without crashing into stuff…


16/25 mph
AUTO

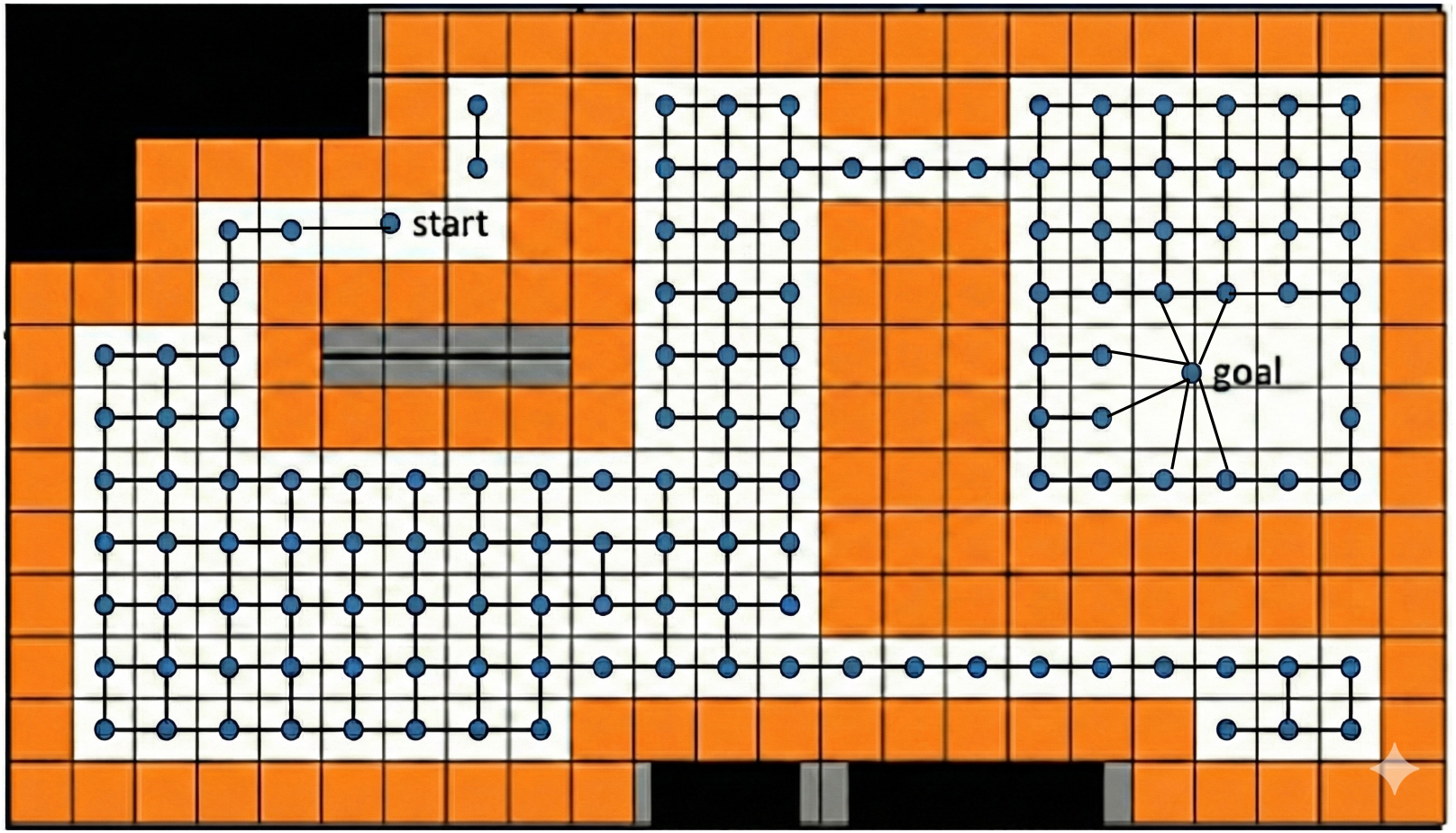




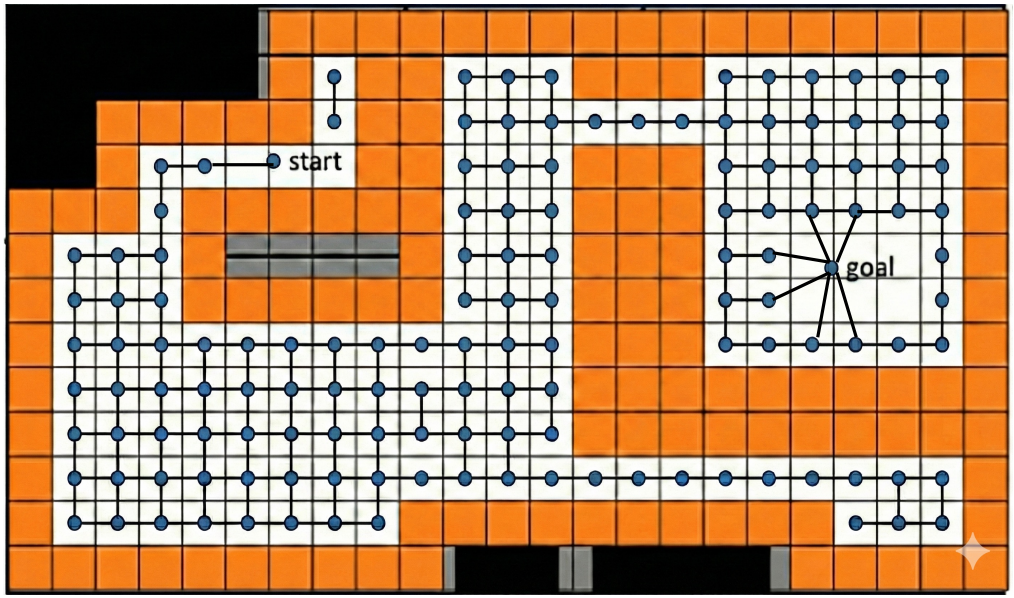
Occupancy Grid



Occupancy Grid



Occupancy Grid, accounting for C-Space

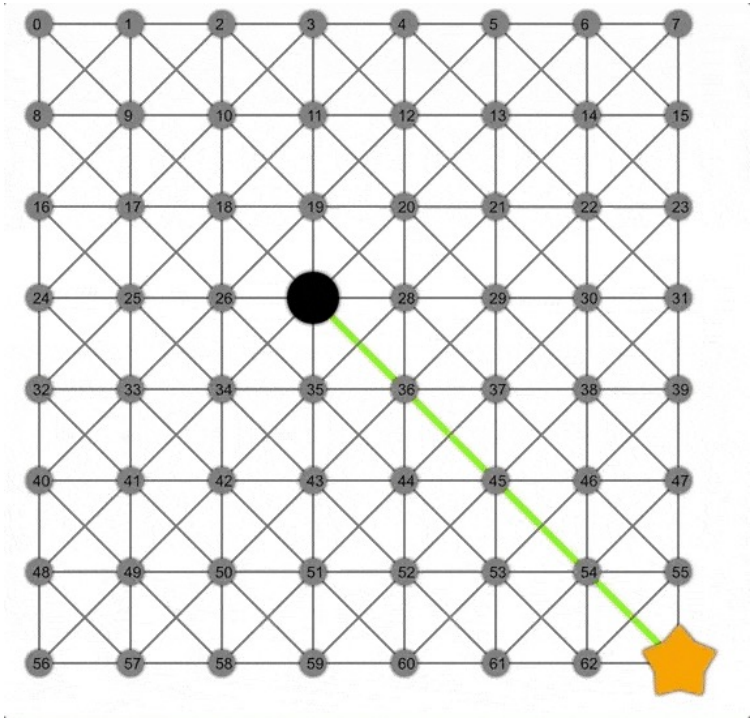


Apply graph search algorithms:

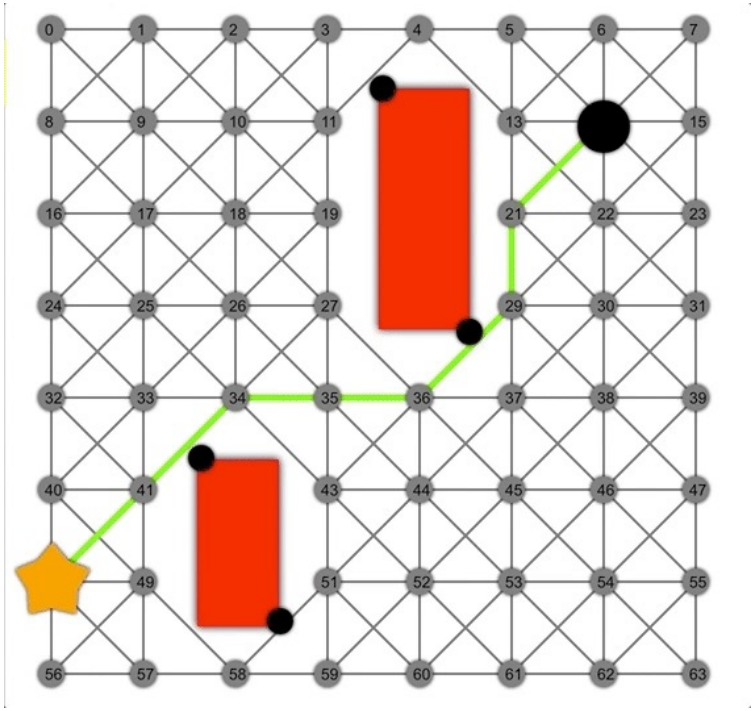
- Dijkstra
- A* Search

Task: find the shortest path from the starting cell to the ending cell.

Graph search

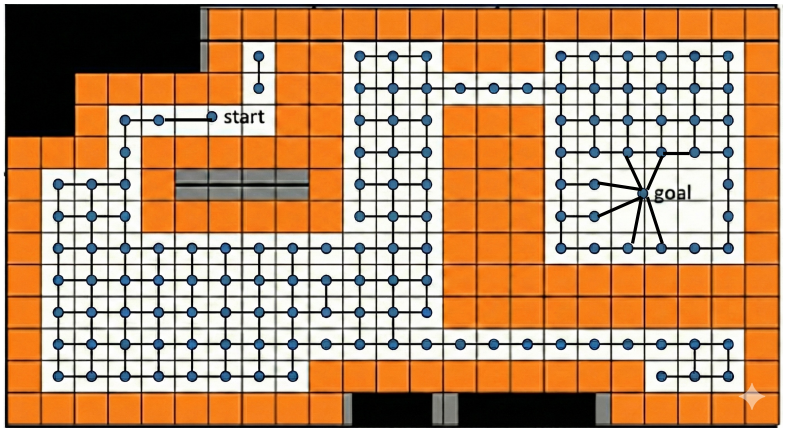


Dijkstra



A*

Occupancy Grid, accounting for C-Space



Task: find the shortest path from the starting cell to the ending cell.

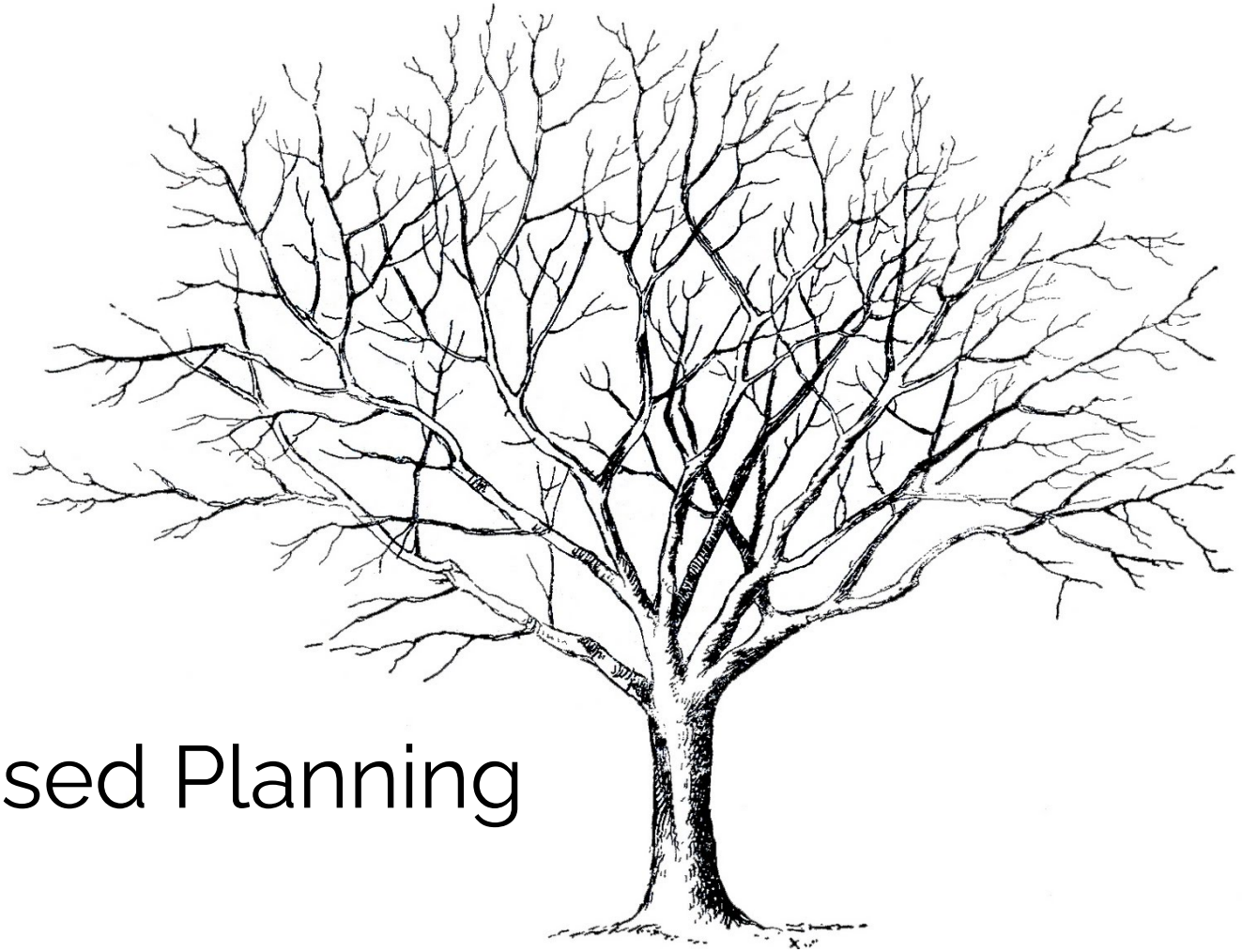
Apply graph search algorithms:

- Dijkstra
- A* Search

Problem: Occupancy grids perform ***exhaustive search*** across the state space (i.e., large graph many nodes and edges). **That is 10^6 nodes for a 6DoF arm…**

What can we do differently?

Sampling-Based Planning



Sampling based planning

KEY IDEA:
Rather than exhaustively explore ALL possibilities, randomly explore a smaller subset of possibilities while keeping track of progress

Then, search for collision-free path only on the sampling points.

Sampling based planning

Today we will introduce two method:

- PRM: Probabilistic Road Map
- RRT: Rapidly–Exploring Random Tree

Probabilistic Road Map (PRM)

Probabilistic Roadmap methods proceed in two phases:

1. Preprocessing Phase – to construct the roadmap *G*
2. Query (Search) Phase – to search a path from q_init and q_goal

The roadmap is an undirected graph $G = (N, E)$.

The nodes in N are a subset of configurations of the robot sampled that are collision free.

The edges in E correspond to feasible straight-line paths.

Probabilistic Road Map (PRM)

Build-Roadmap:

G.init() ; i = 0

While i < N

$\alpha \leftarrow$ SAMPLE

if $\alpha \in C_{free}$

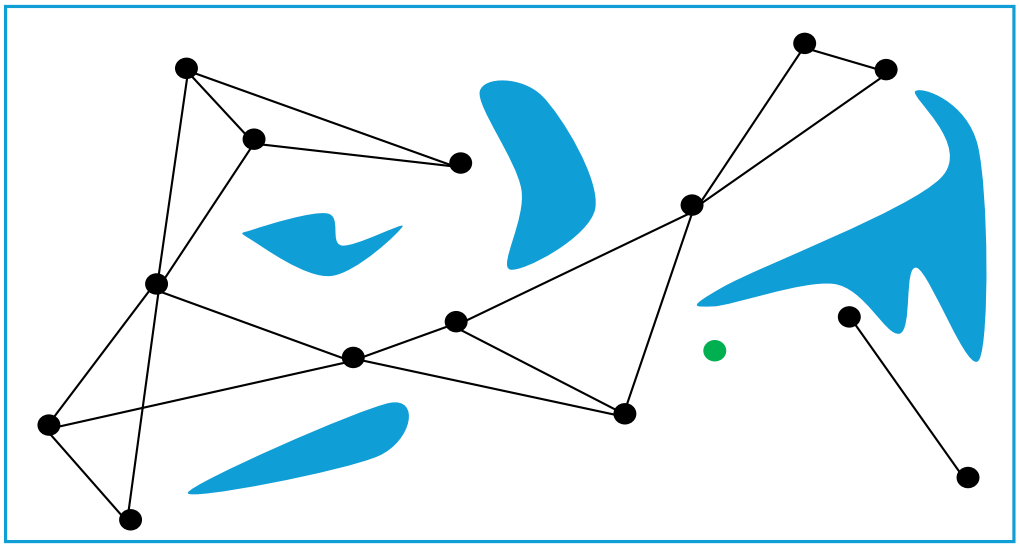
G.add Vertex(α); i = i+1

for each $q \in ngd(\alpha, G)$

if (connect(α , G))

G.addEdge(α , q)

Probabilistic Road Map (PRM)



Build-Roadmap:

G.init() ; i = 0

While i < N

$\alpha \leftarrow$ SAMPLE

if $\alpha \in C_{free}$

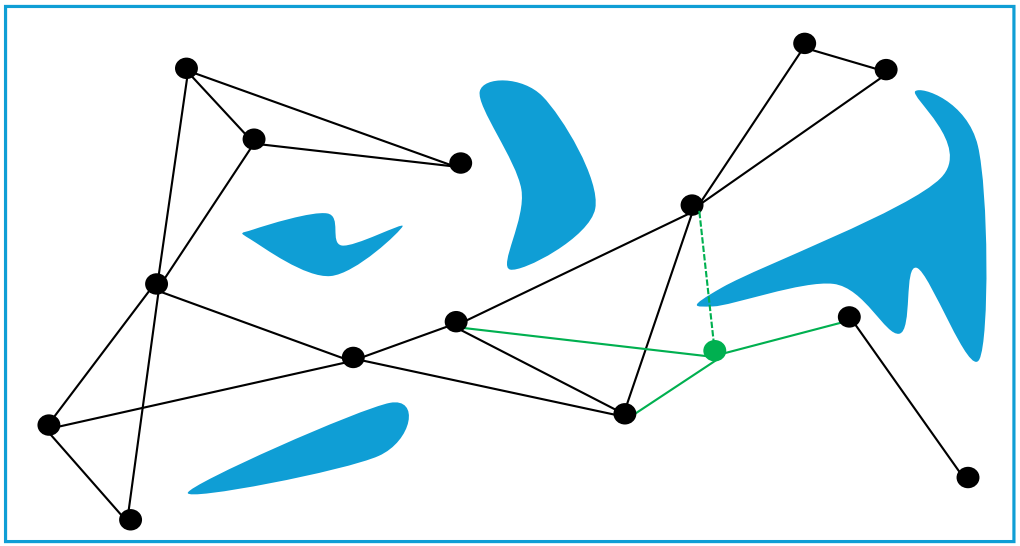
G.add Vertex(α); i = i+1

for each $q \in ngd(\alpha, G)$

if (connect(α , G))

G.addEdge(α , q)

Probabilistic Road Map (PRM)



Build-Roadmap:

G.init() ; i = 0

While i < N

$\alpha \leftarrow$ SAMPLE

if $\alpha \in C_{free}$

G.add Vertex(α); i = i+1

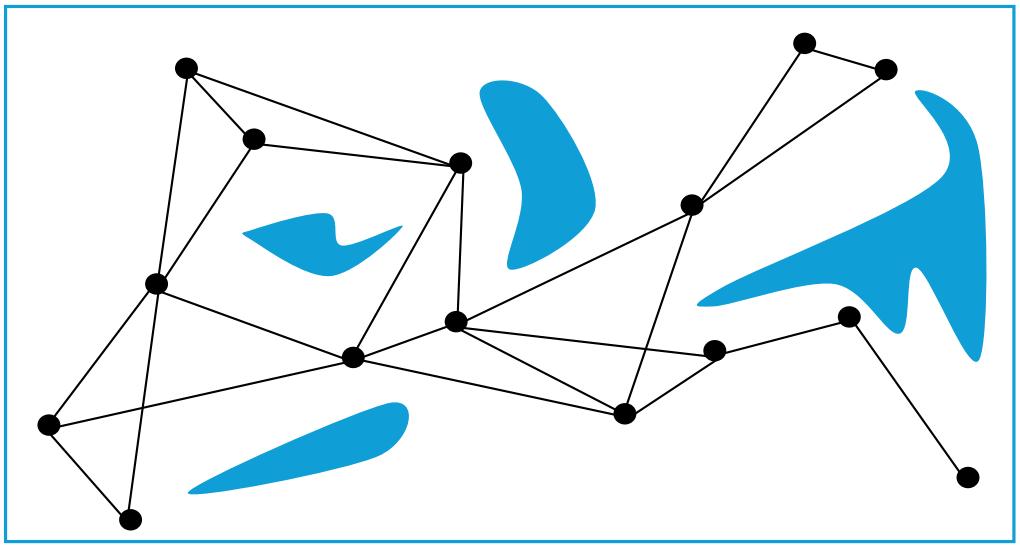
for each $q \in ngd(\alpha, G)$

if (connect(α , G))

G.addEdge(α , q)

"neighborhood"

Probabilistic Road Map (PRM)



Build-Roadmap:

G.init() ; i = 0

While i < N

$\alpha \leftarrow$ SAMPLE

if $\alpha \in C_{free}$

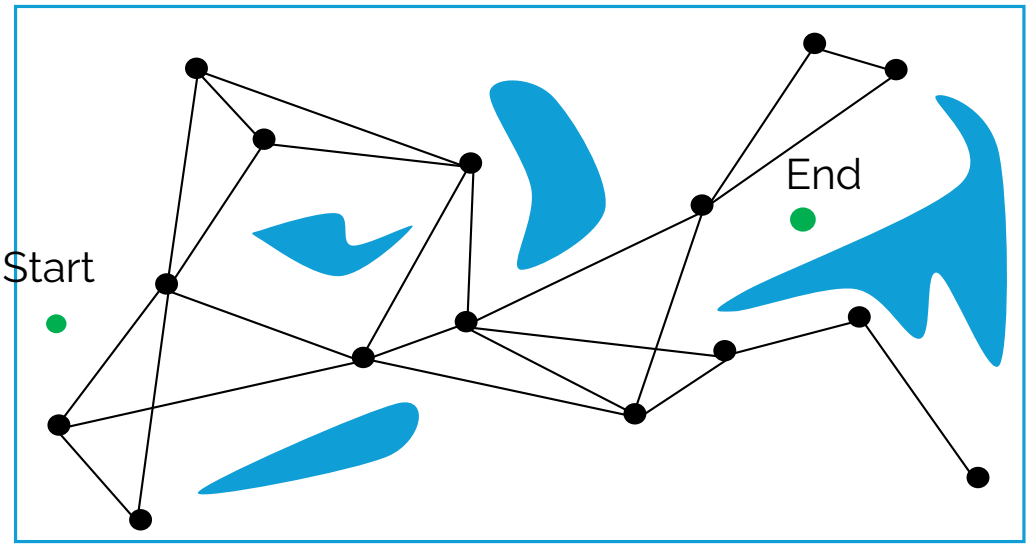
G.add Vertex(α); i = i+1

for each $q \in ngd(\alpha, G)$

if (connect(α , G))

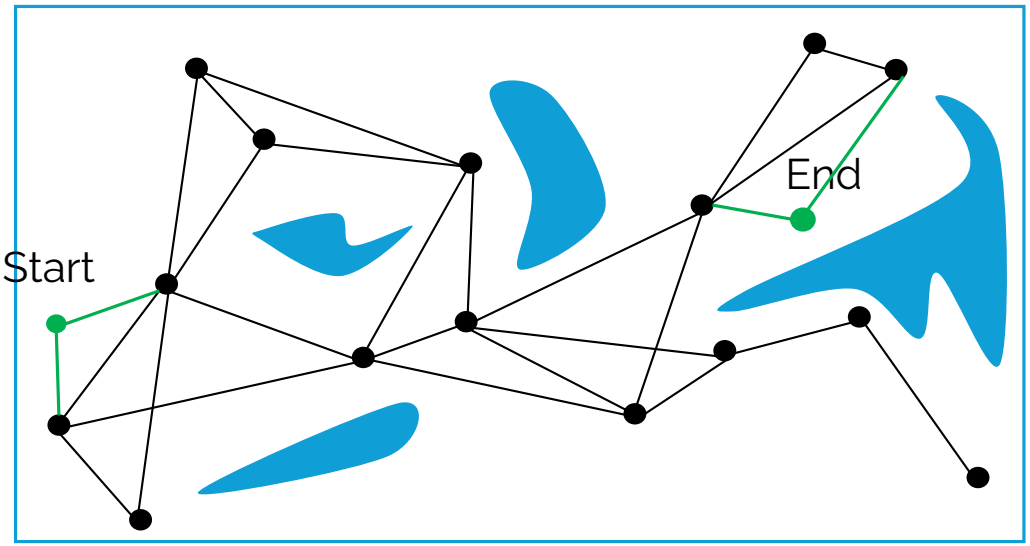
G.addEdge(α , q)

Probabilistic Road Map (PRM) – Phase 2



Query Phase:
Search a path given q_init and q_goal with the constructed graph

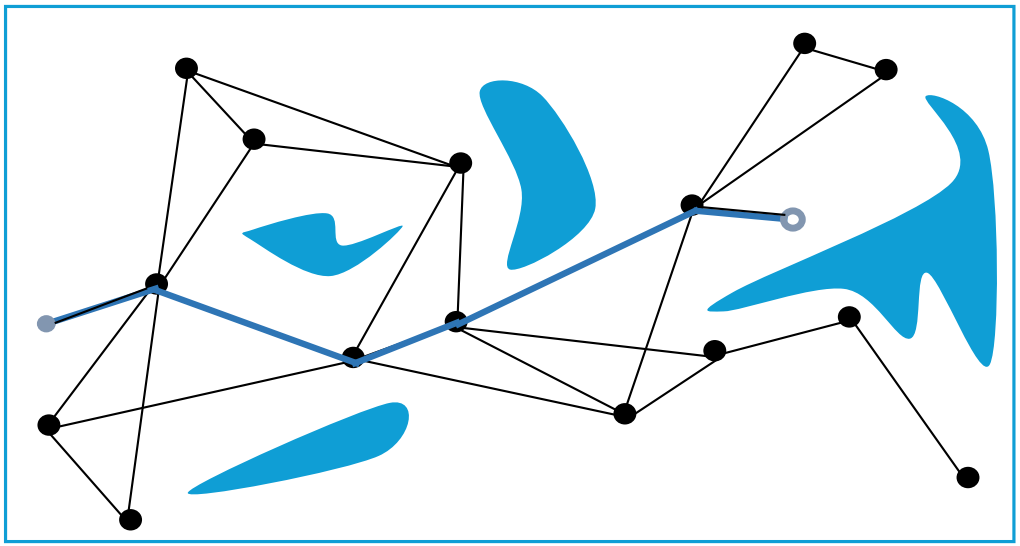
Probabilistic Road Map (PRM) – Phase 2



Frist, add these two node into the graph

Then, connect the possible edges to existing nodes

Probabilistic Road Map (PRM) – Phase 2



Frist, add these two node into the graph

Then, connect the possible edges to existing nodes

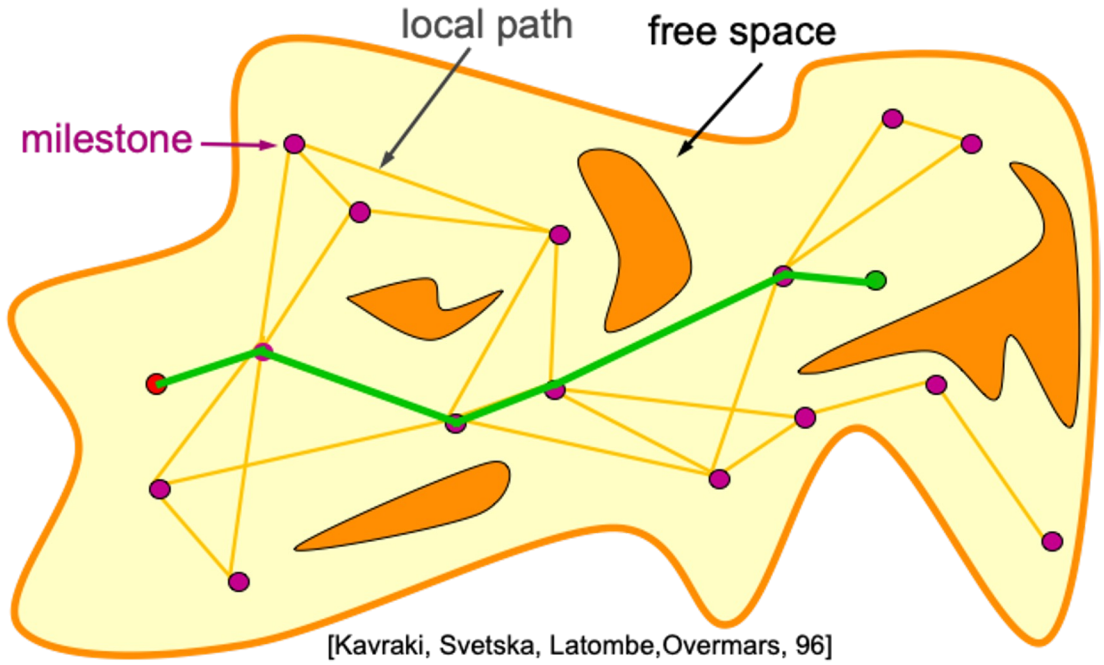
Finally, perform a graph search.

Some terminology

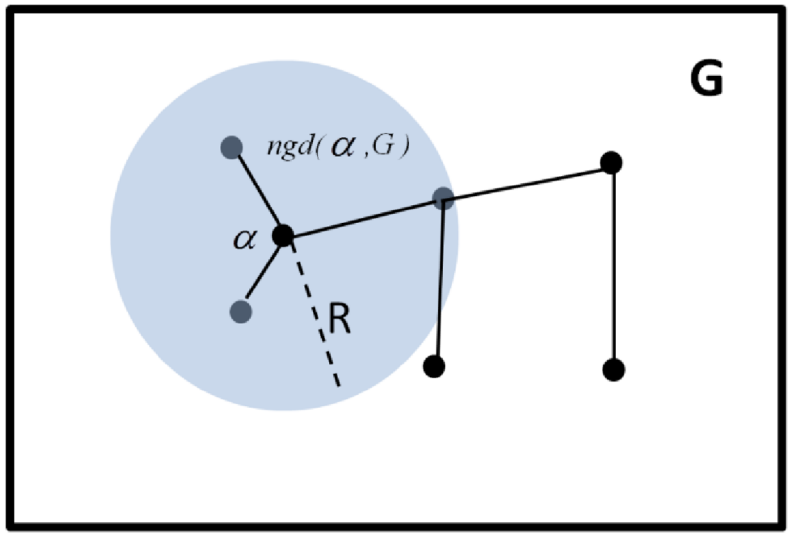
The graph G is called a **probabilistic roadmap**.

The nodes in G are called **milestones**.

Edge between two milestones called **local path (or roadways)**



A few design decisions:



How to sample a new point?

Randomly or using other “clever” methods? The biggest advantage for random sampling is that there is no bias.

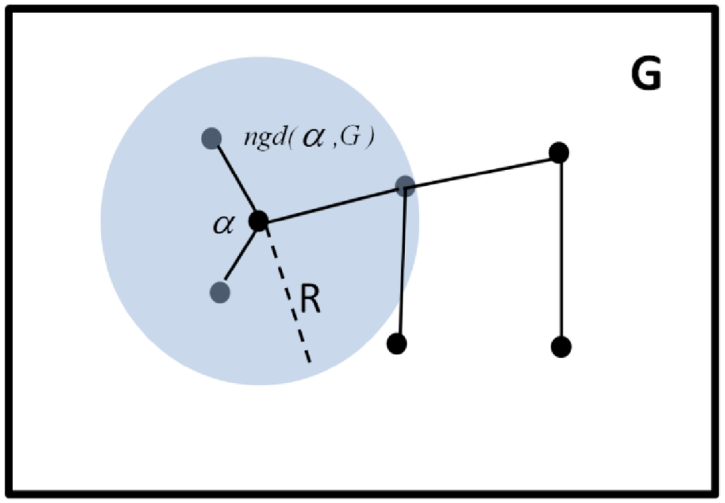
How to choose the distance of neighbor *R? L2 distance?* How far?

Defining distance

The PRM procedure relies upon a distance function, **Dist**, that can be used to measure the distance between two points in configuration space. This function takes as input the coordinates of the two points and returns a real number.

Common choices for distance functions include:

- The L1 distance : $Dist_1 = \sum_i |\mathbf{x}_i - \mathbf{y}_i|$
- The L2 distance : $Dist_2 = \sqrt{(\sum_i (\mathbf{x}_i - \mathbf{y}_i)^2)}$



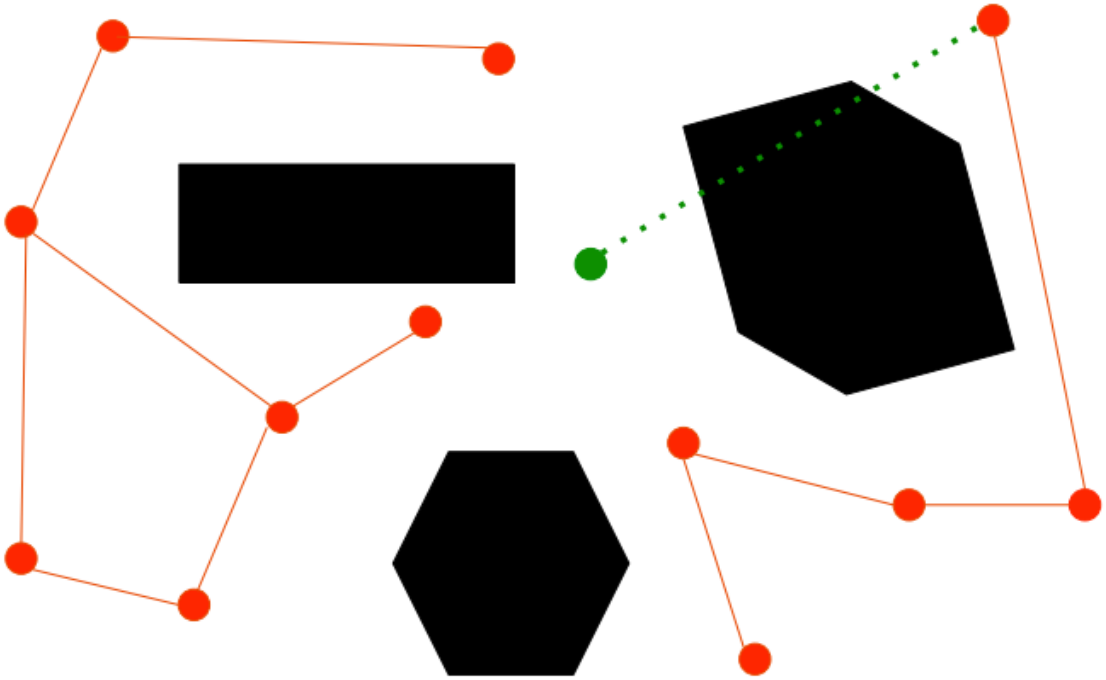
Defining distance with angle

Sometimes, configuration space correspond to **angular rotations**.
In these situations, we want ensure that the Dist function correctly reflects distances in the presence of wrap around. If there is no joint limit.

For example if θ1 and θ2 denote two angles between 0 and 360 degrees the expression below can be used to capture the angular displacement between them.

$$Dist(\theta_1, \theta_2) = \min(|\theta_1 - \theta_2|, (360 - |\theta_1 - \theta_2|))$$

Check for collision along a path

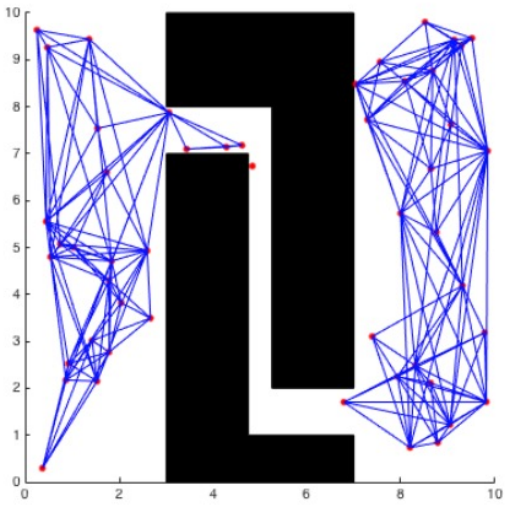


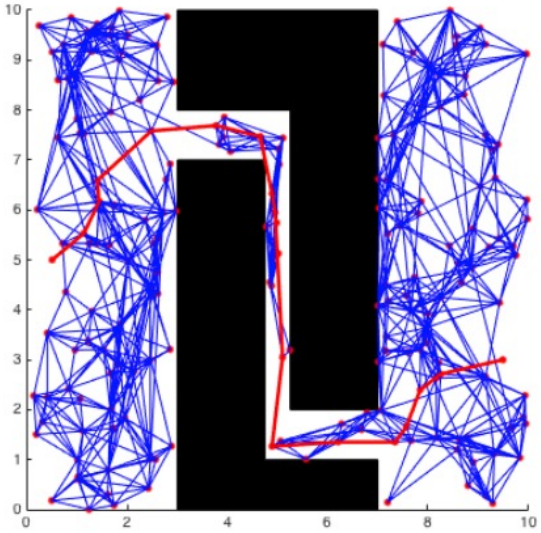
Method:
Uniformly sample points along the path.

Check: if all the points are in free space, we assume the path is collision free

Random Sample in PRM

It is possible to have a situation where the algorithm would fail to find a path even when one exists if the sampling procedure fails to generate an appropriate set of samples.

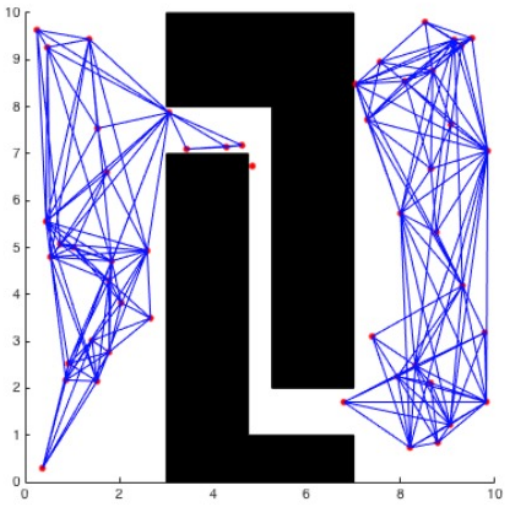


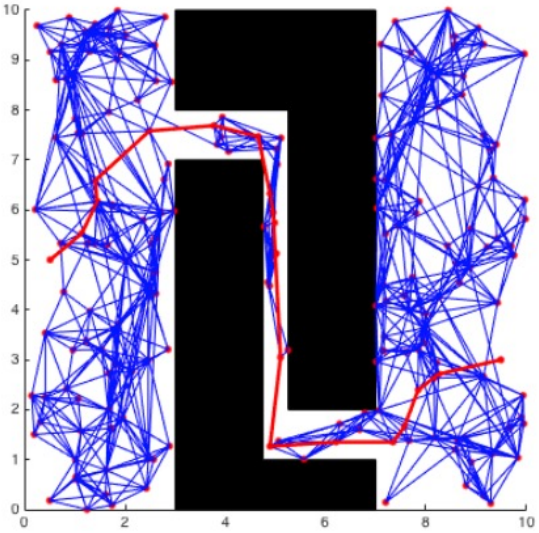


How to solve this problem?

Random Sample in PRM

It is possible to have a situation where the algorithm would fail to find a path even when one exists if the sampling procedure fails to generate an appropriate set of samples.





Possible solution:

1. More Samples

2. Sample more points close to C-space obstacles

Random Sample in PRM

It is possible to have a situation where the algorithm would fail to find a path even when the solution exists.

(often cased by sampling procedure fails to generate an appropriate set of samples)

The only thing we can say is If there is a route and the planner keeps adding random samples it will, eventually find a solution.

In summary: **PRM is not complete but probabilistically complete**

Summary: Probabilistic Road Map (PRM)

- Assumption:
  - Static obstacles
  - Many queries (start & end state) to be processed in the same environment

- Example applications:
  - Navigation in static virtual environments
  - Robot manipulator arm in a static workcell

- **Advantages**: one roadmap that works for many q_init and q_goal, repeatedly

- **Disadvantage**: it spends a lot of time on preprocessing (roadmap construction), and if environment changes after the road map is constructed, then we need to redo the process.

But in many cases, we have the start and end point in mind, can we use it to improve the sampling efficiency?

<break>

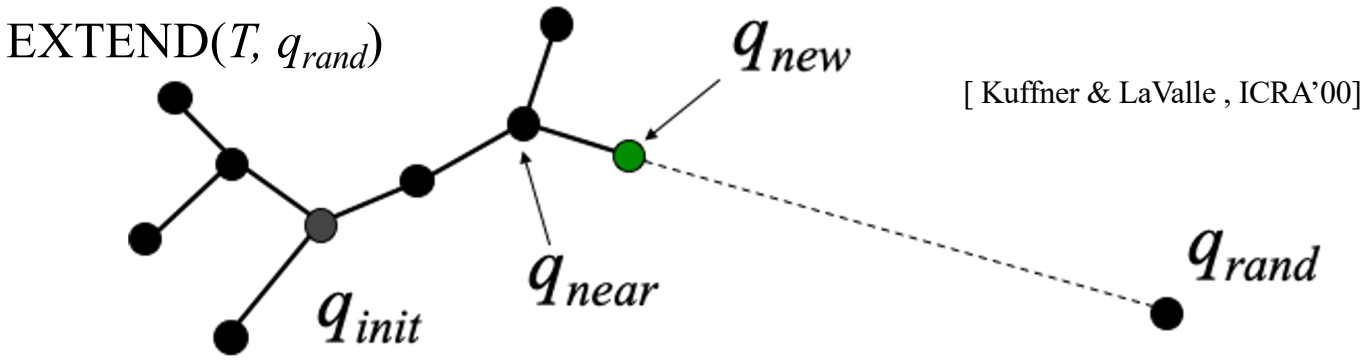
Rapidly-Exploring Random Tree (RRT)

Rapidly-Exploring Random Tree (RRT)

- Searches for a path from the initial configuration to the goal configuration by expanding a search tree - ***single root node (start),*** *no longer a general graph*
- For each step,
  - The algorithm samples a **new** configuration and expands the tree towards it.
  - The new sample can either be a random configuration or the goal configuration itself.

The Basic Idea: Iteratively expand the tree

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



[Kuffner & LaValle , ICRA'00]

RRT

Requires the following functions:

RRT

Requires the following functions:

x_{rand} = RandomSample()

Uniform random sampling of free configuration space

This can easily be done by sampling randomly in R^n (or whatever your configuration space is) and then discarding any points which fail the collision check

RRT

Requires the following functions:

x_{rand} = RandomSample()

Uniform random sampling of free configuration space

$x_{nearest}$ = Nearest(G, x_{rand})

Given point in Cspace, find vertex on tree that is closest to that point

RRT

Requires the following functions:

x_{rand} = RandomSample()

Uniform random sampling of free configuration space

$x_{nearest}$ = Nearest(G, x_{rand})

Given point in Cspace, find vertex on tree that is closest to that point

x_{new} = Steer($x_{nearest}$, x_{rand})

Find x_{new} take 'one step' towards x_{rand} from $x_{nearest}$

If (Dist ($x_{nearest}$, x_{rand}) <= delta)
    x_{new}= x_{rand}
else:
    Compute a new configuration x_{new}, that is along the path from $x_{nearest}$ to x_{rand} such that Dist($x_{nearest}$, x_{new}) = delta

RRT

Requires the following functions:

x_{rand} = RandomSample()

Uniform random sampling of free configuration space

$x_{nearest}$ = Nearest(G, x_{rand})

Given point in Cspace, find vertex on tree that is closest to that point

x_{new} = Steer($x_{nearest}$, x_{rand})

Find x_{new} take 'one step' towards x_{rand} from $x_{nearest}$

ObstacleFree(x, y)

Check if the path from x to y is collision free

Oftentimes, we assume delta is smaller enough, and we only need to check obstacleFree on the new point x_{new}

RRT

Input: Initial configuration x_{init}, number of vertices in tree N, incremental distance *delta*.

Note: end configuration is not part of the input for this version of RRT.

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$

for $i = 1$ to N

  $G \leftarrow (V, E)$

  $x_{rand} \leftarrow RandomSample()$

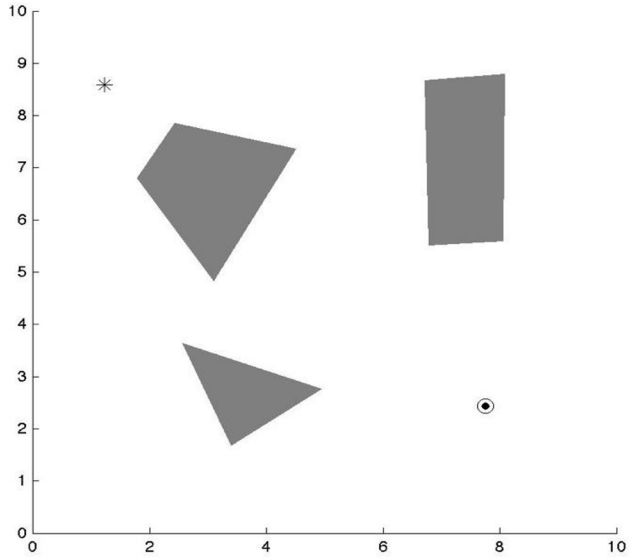
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$

  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$

  if $ObstacleFree(x_{nearest}, x_{new})$

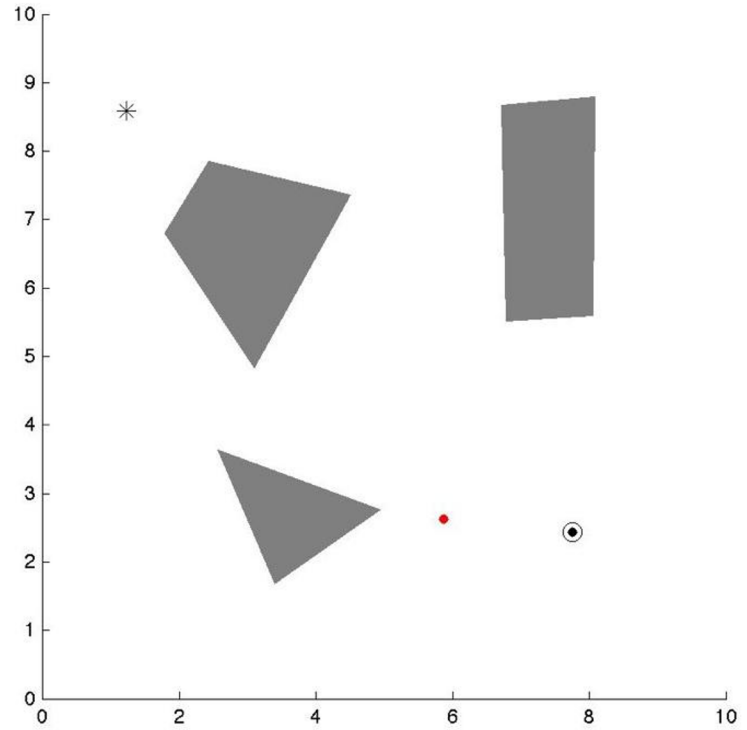
    $V \leftarrow V \cup \{x_{new}\}$

    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



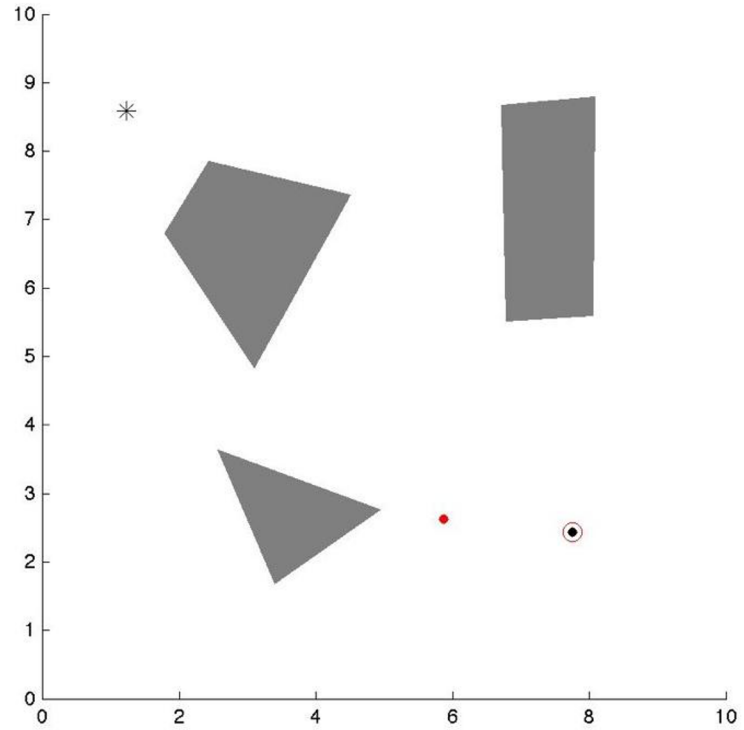
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



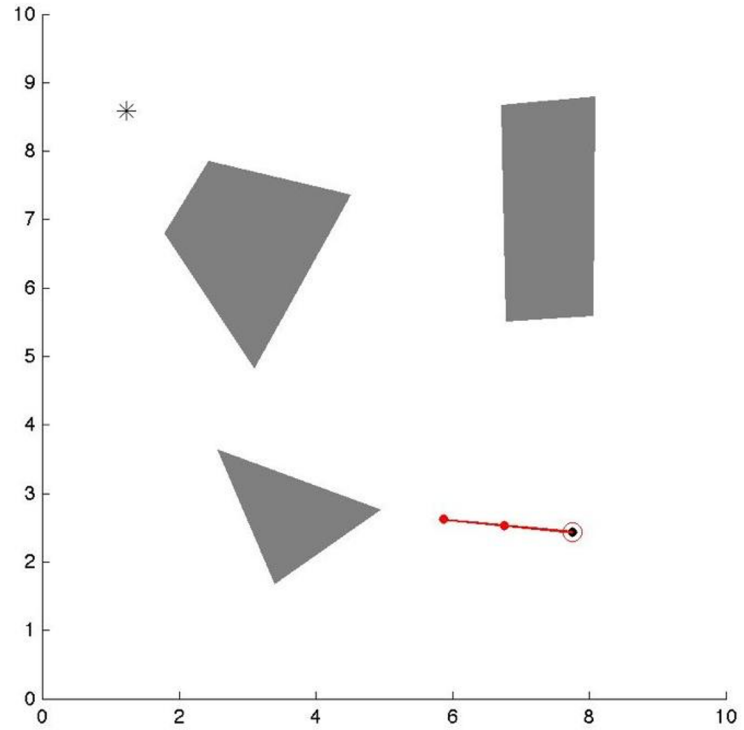
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



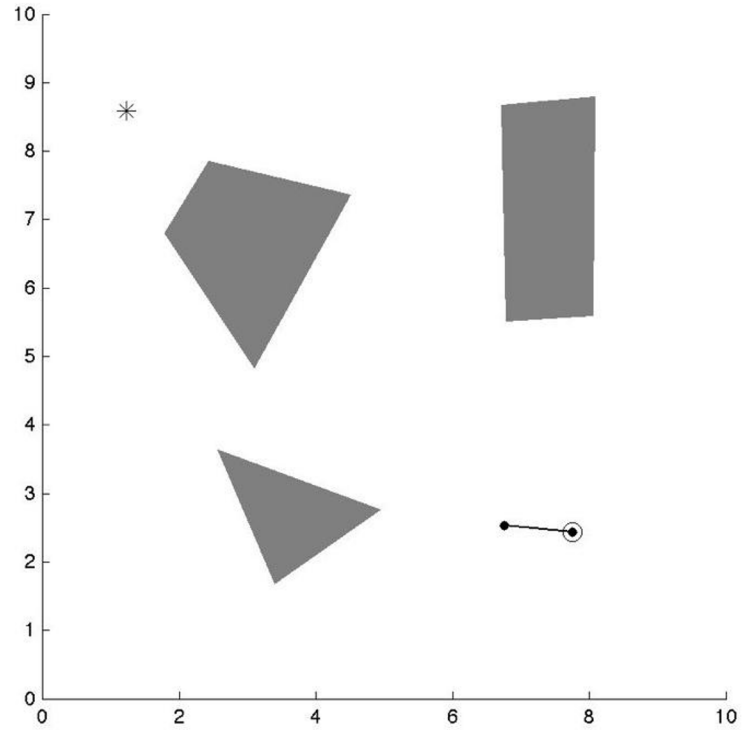
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



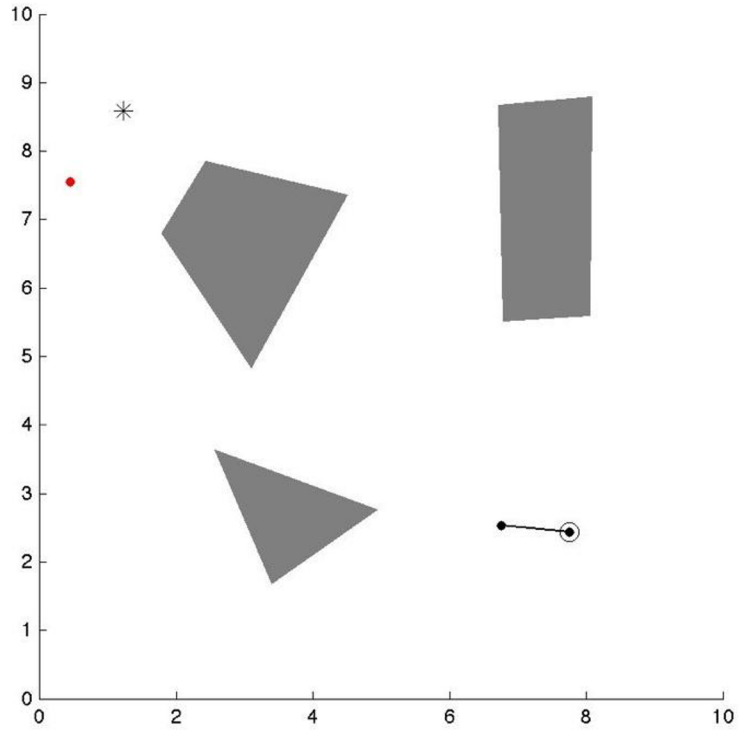
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



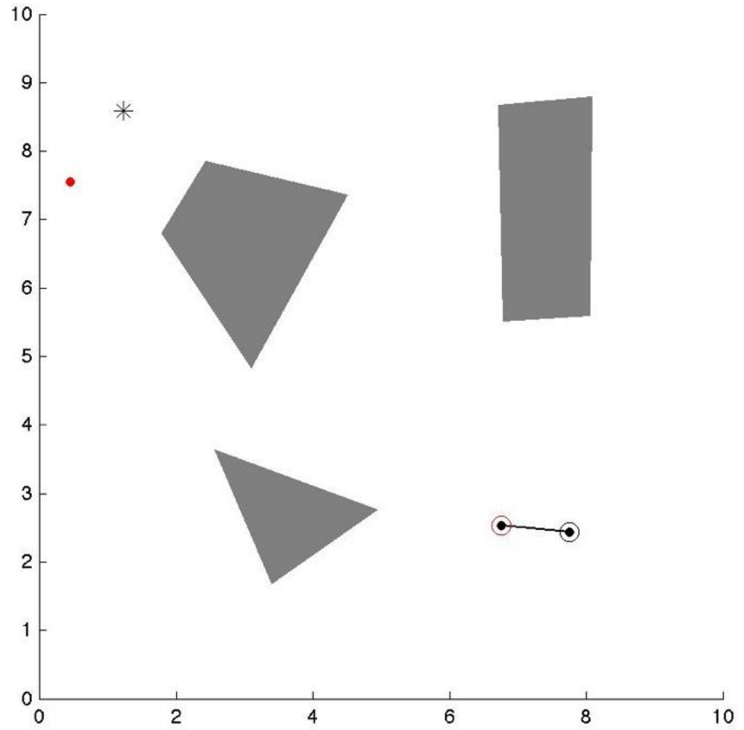
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



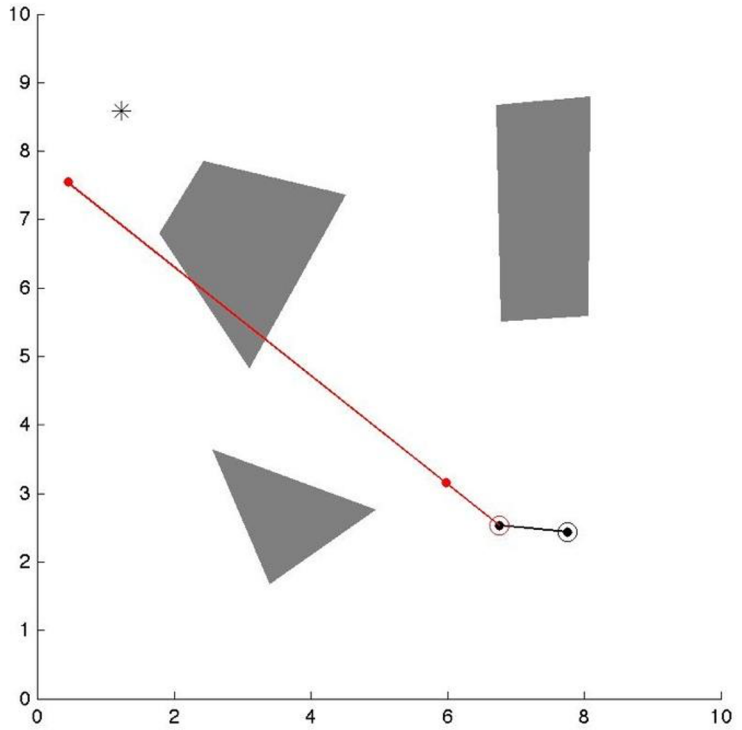
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



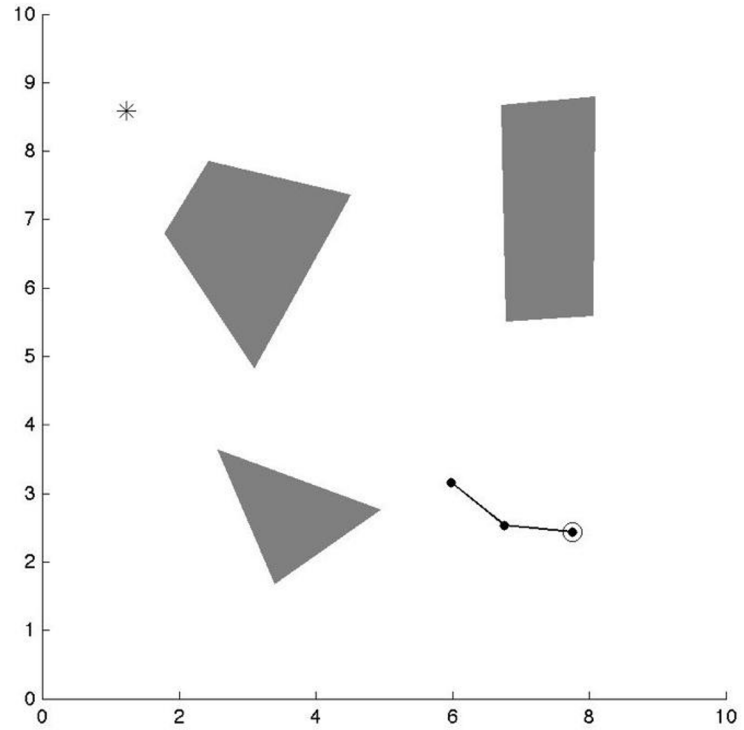
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



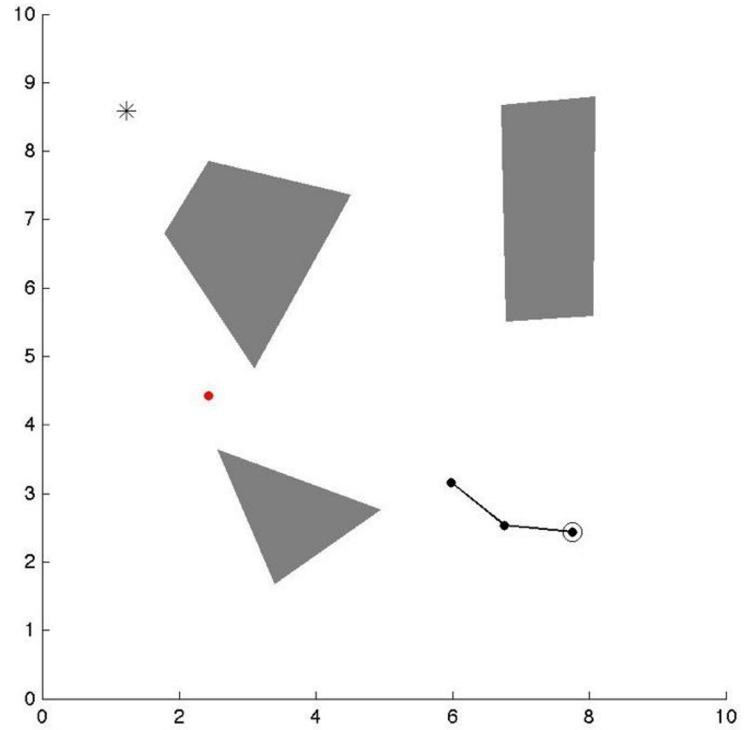
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



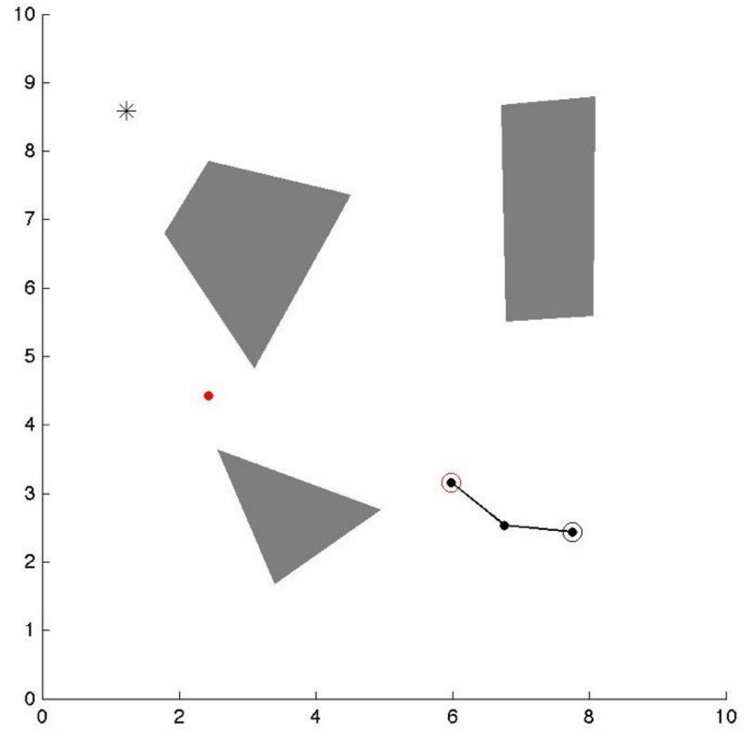
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



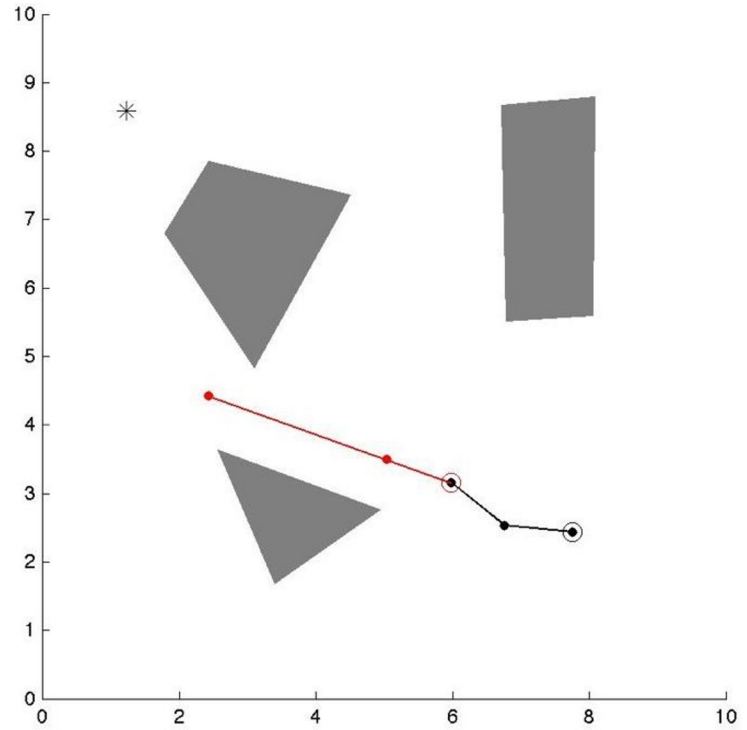
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



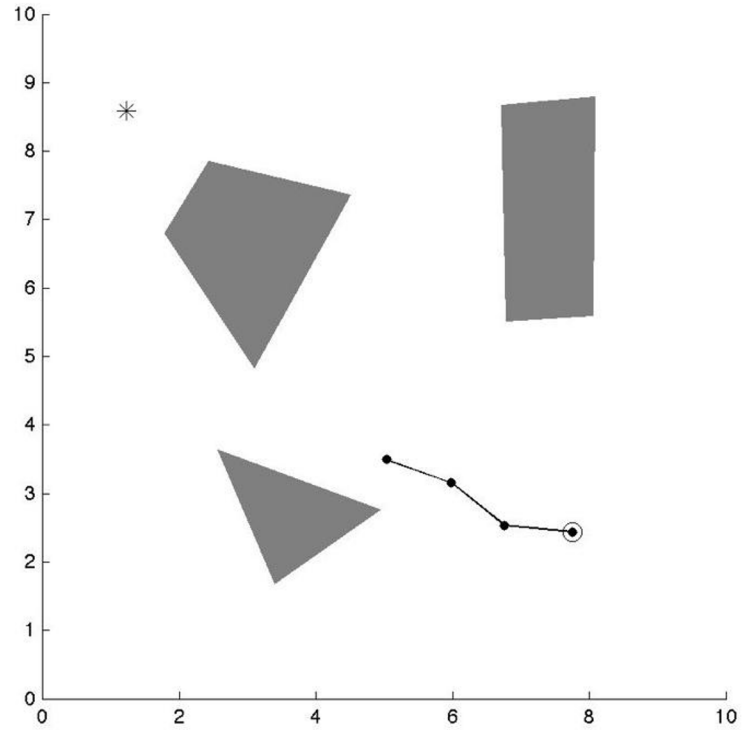
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



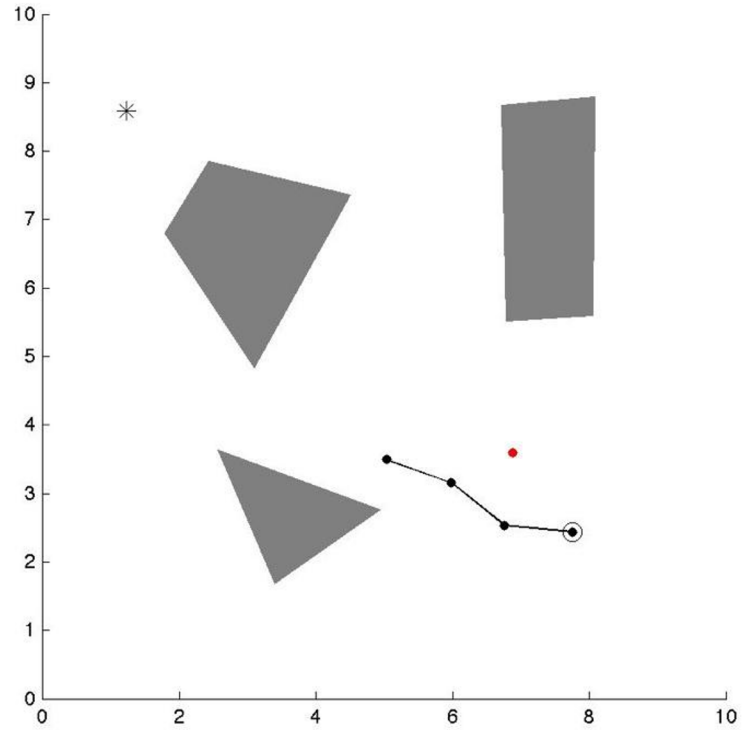
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



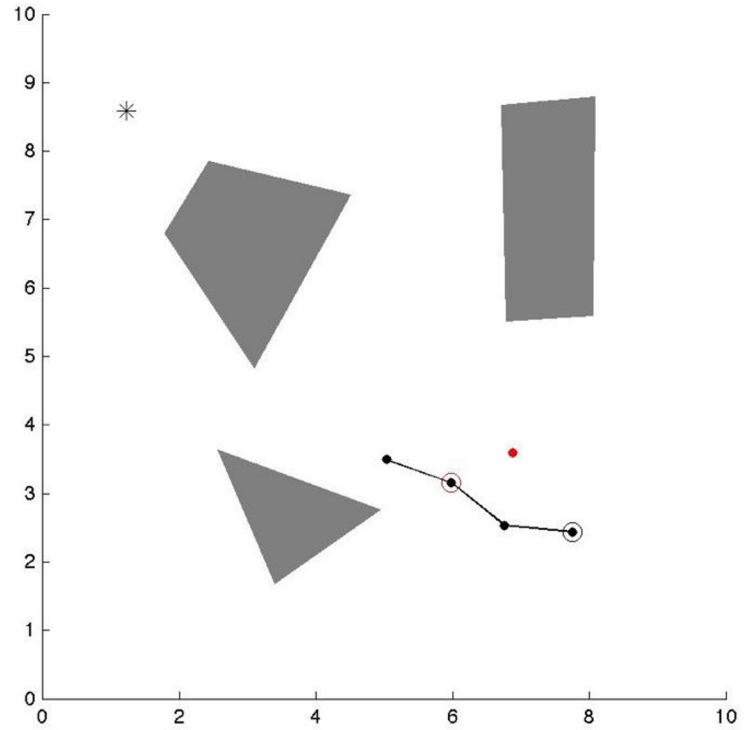
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



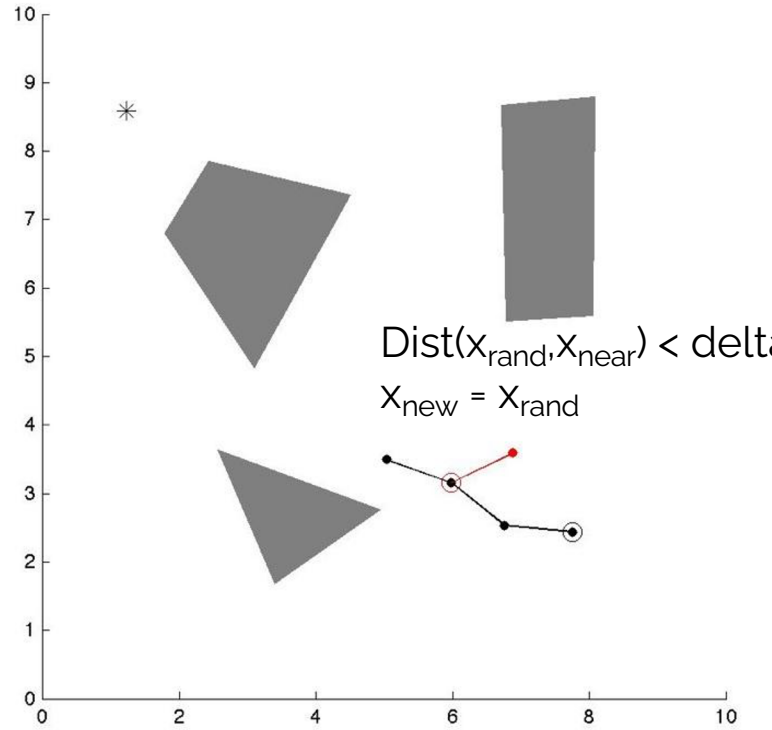
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$

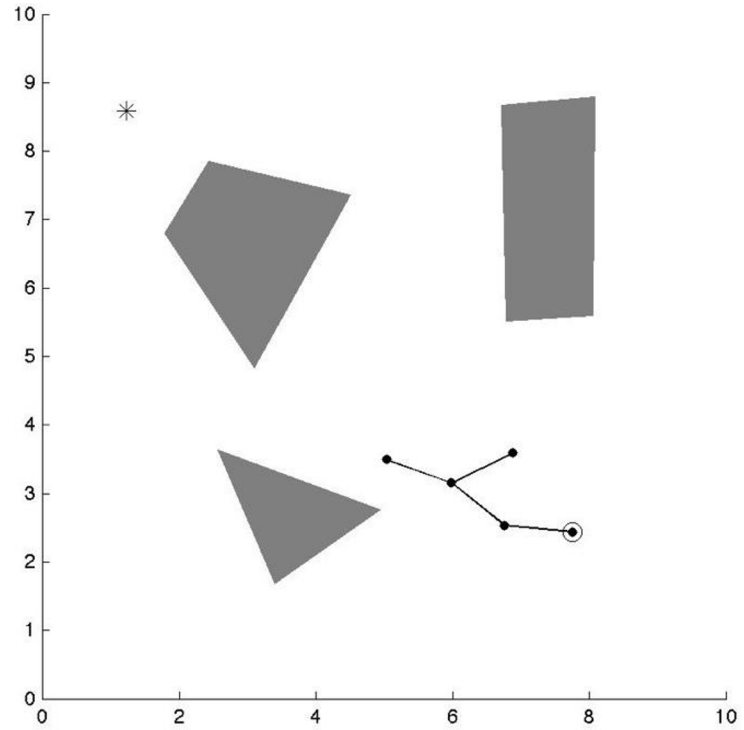


Dist(x_{rand},x_{near}) < delta

x_{new} = x_{rand}

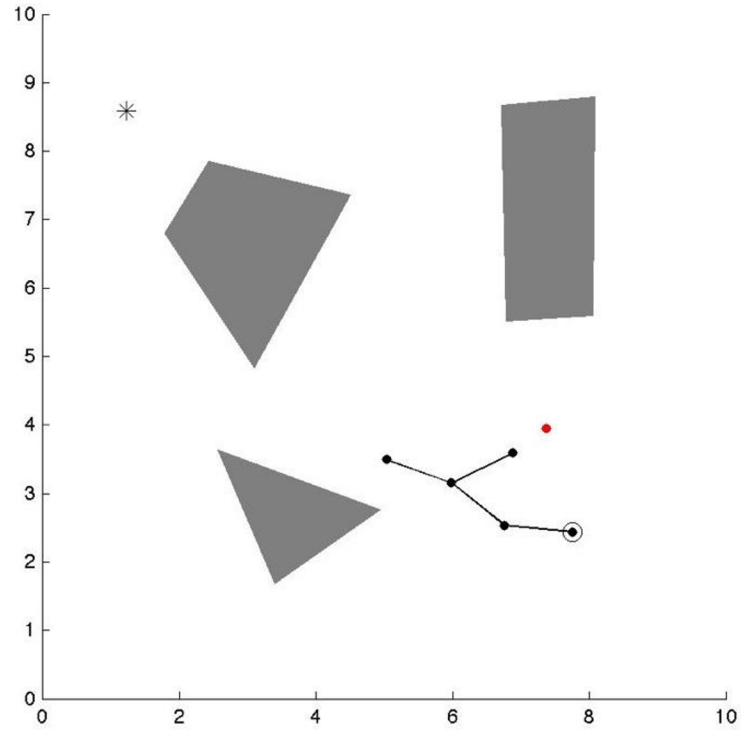
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



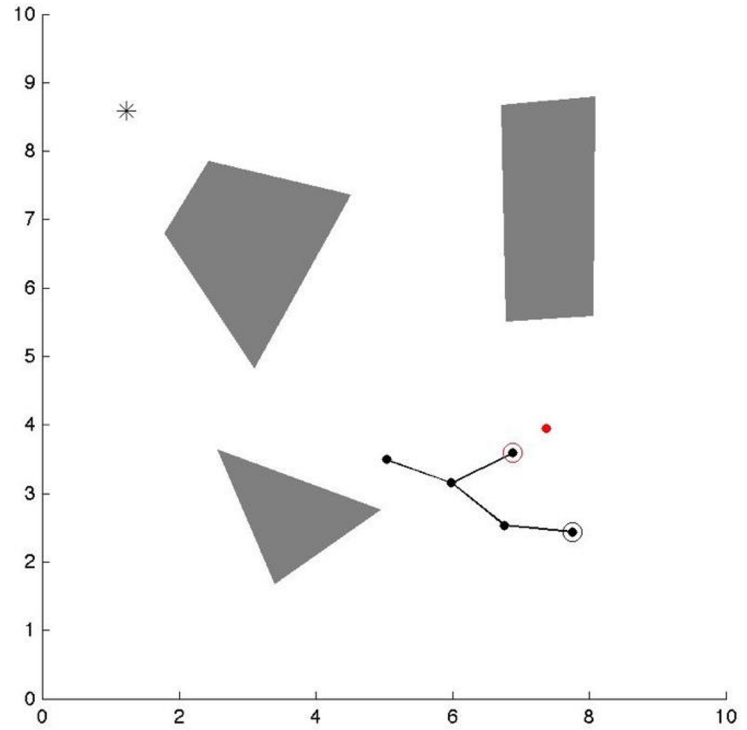
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



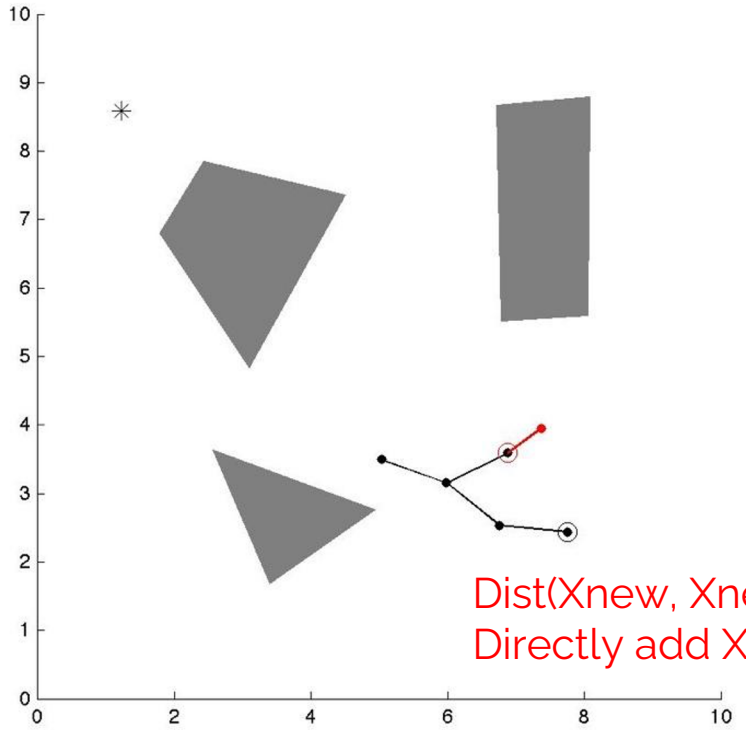
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

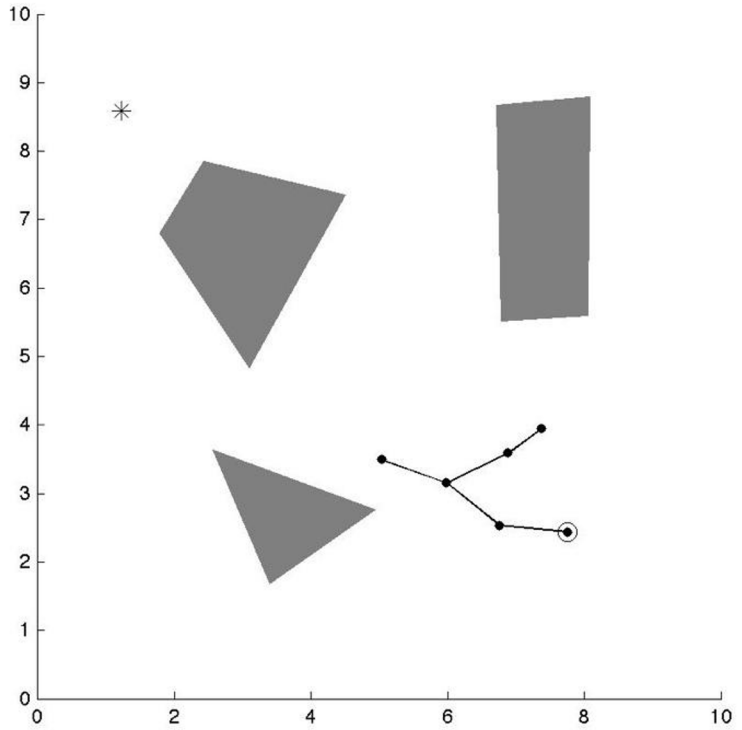
$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



Dist(Xnew, Xnearest) < delta
Directly add Xnew

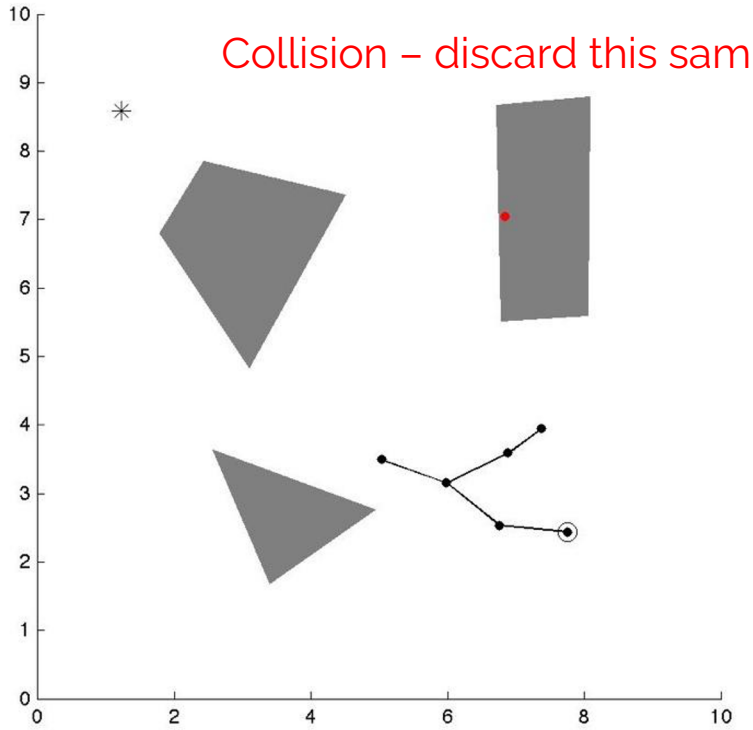
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



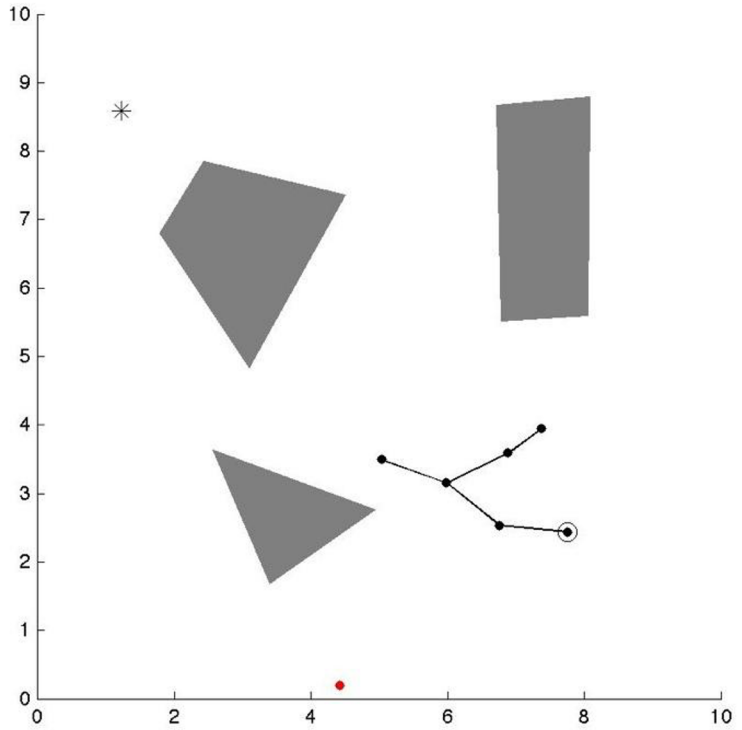
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



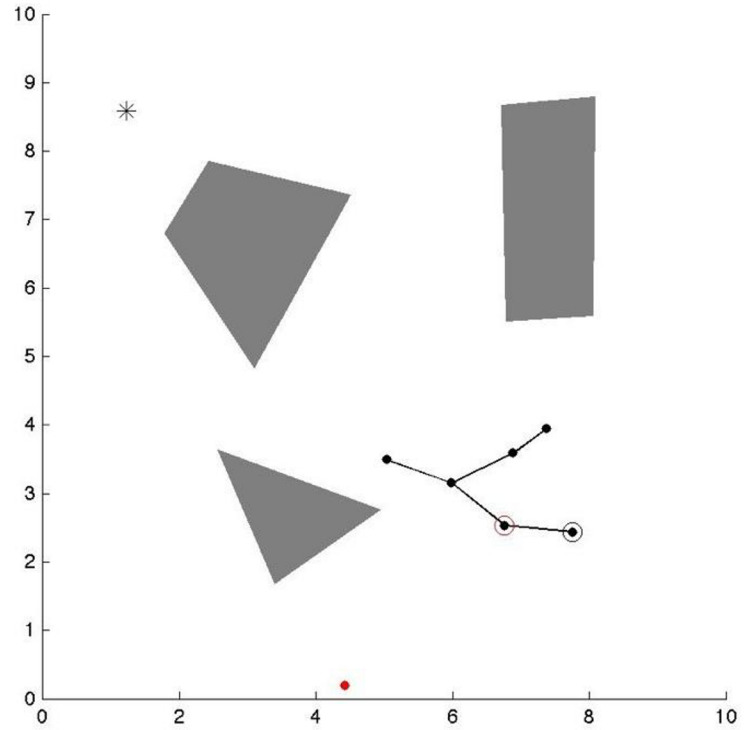
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



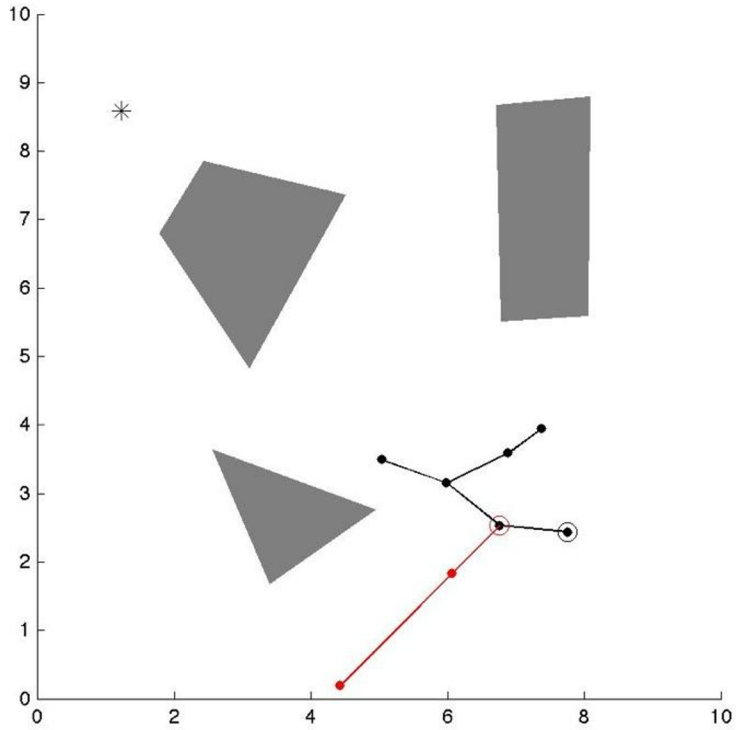
RRT

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

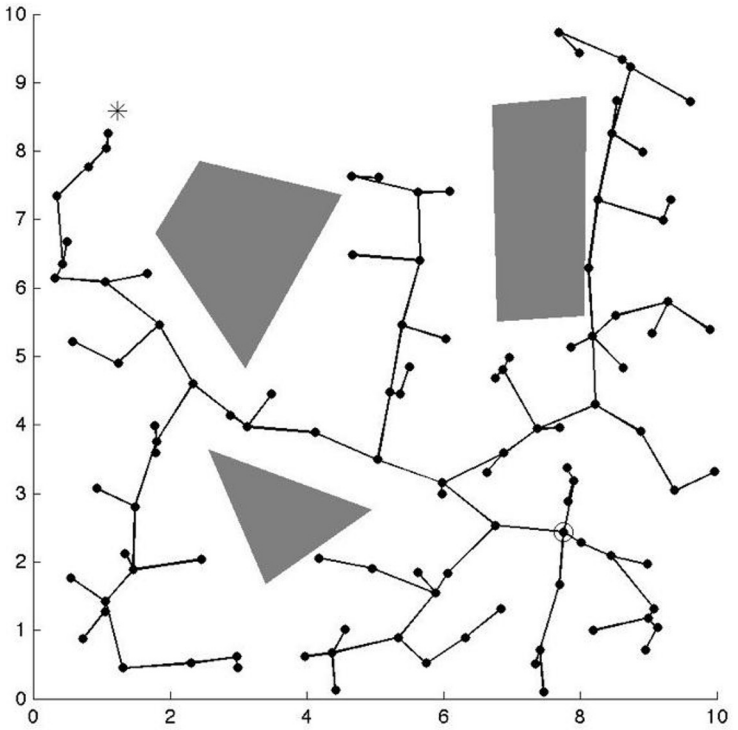
$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



RRT

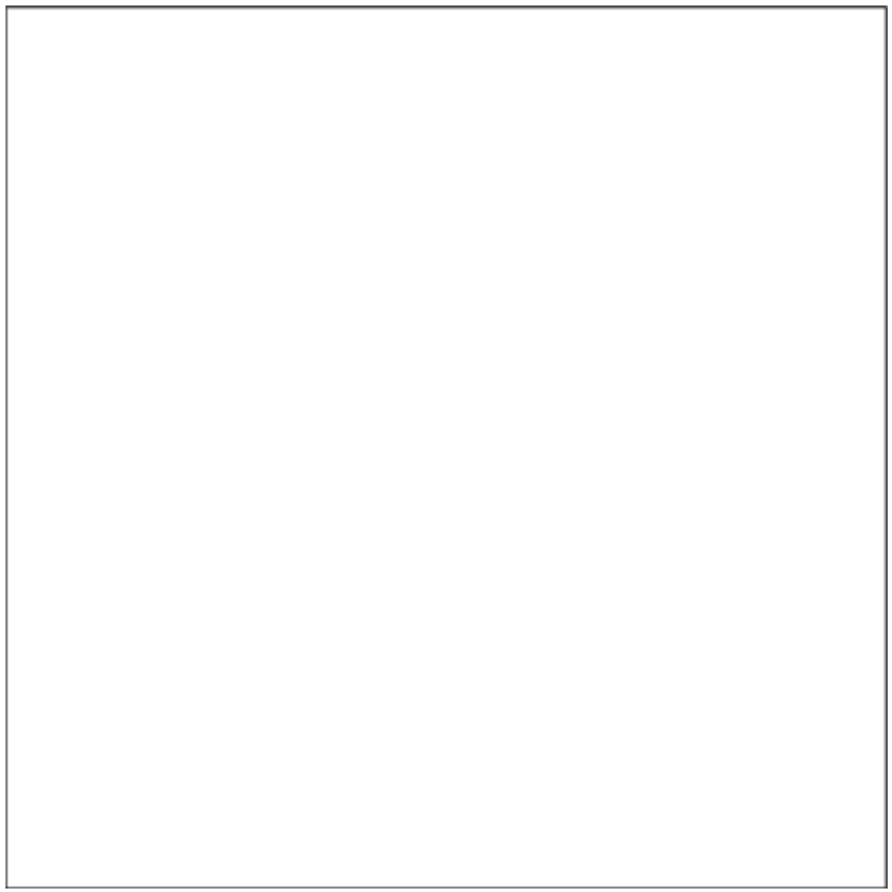
It is the *basic* verison of RRT.
In practice, there are a number of improvements we can do

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



Rapidly-Exploring Random Tree

Basic algorithm is simply exploring space - but we want it to actually head towards the goal!



RRT - Bias to Goal

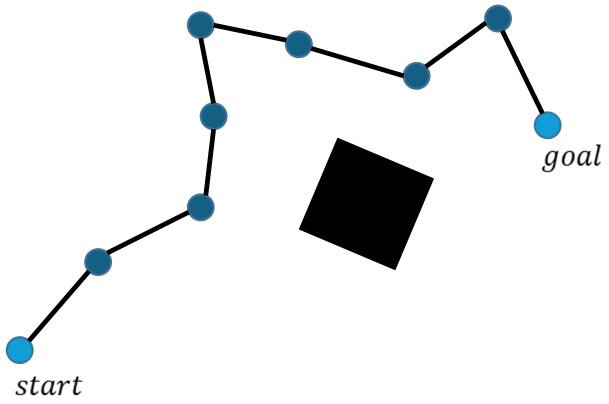
$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  with probability p
    $x_{rand} \leftarrow RandomSample()$
  otherwise
    $x_{rand} \leftarrow x_{goal}$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
    If Dist(x_{new}, x_{goal}) < delta
      path <- calculate the path from x_{init}, x_{goal}
      return path;

When generating a random sample, with some probability pick the goal instead of a random node when expanding

Return when reaches the goal

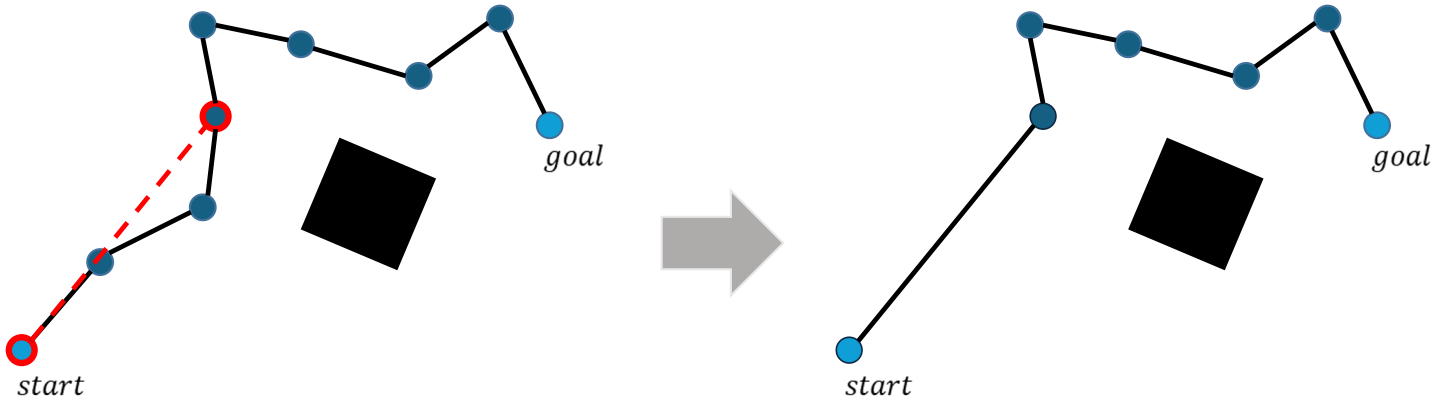
Optimizing the path

- Milestone-based paths are far from optimal and require additional refinement before they are usable
- A typical solution can look like this:



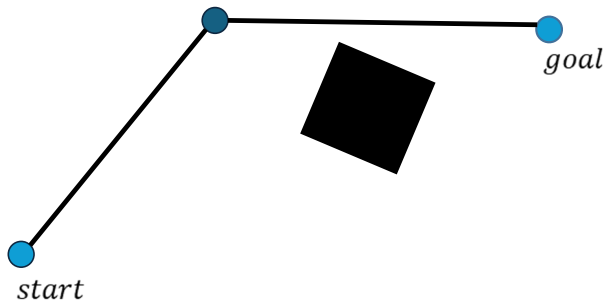
Optimizing the path

- A simple way to improve the path (after we find a solution), is to random pick two nodes, and check whether they can be connected by a straight line without collision. If so, use the line to shorten the path.

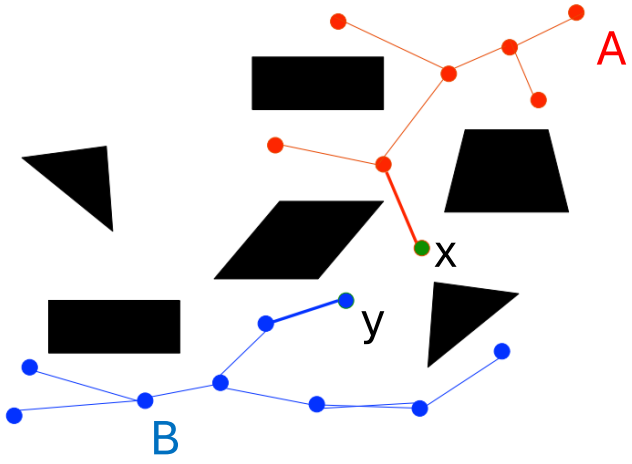


Optimizing the path

- Repeat for N iterations, or until no further improvements are being made
- The result is not an optimal path, but shorter and more efficient than the original



Bidirectional RRT:

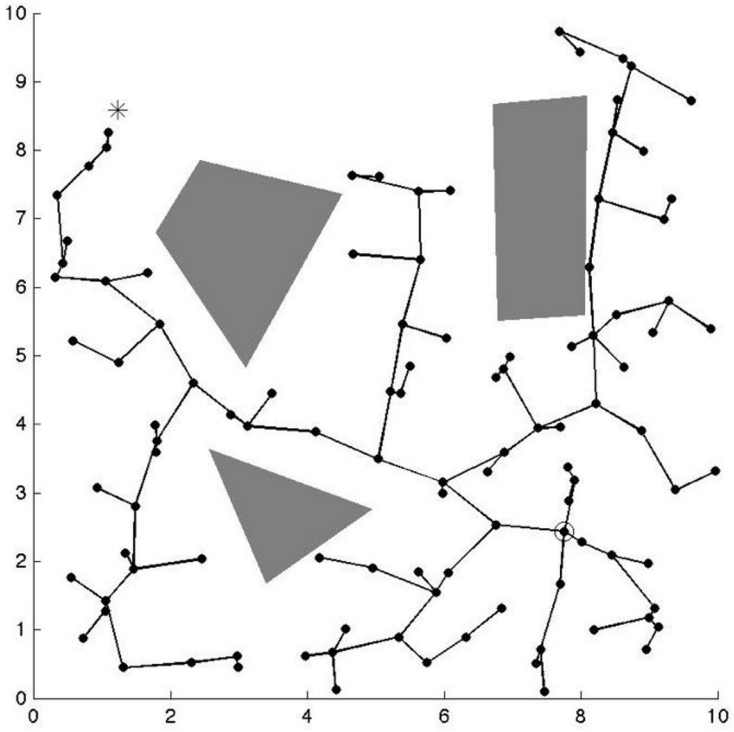


- Build two tree A from Start, B from End
- While not done
  - Extend Tree A by adding a new node
    - (sample + steer)
  - Find the closest node in Tree B to x : y
  - If (ObstacleFree(x,y)) – Check if you can bridge the 2 trees
    - Add edge between x and y. This completes a route between the root of Tree A and the root of Tree B. Return this route
  - Else
    - Swap Tree A and Tree B (in the next iteration, we will expand tree B)

RRT

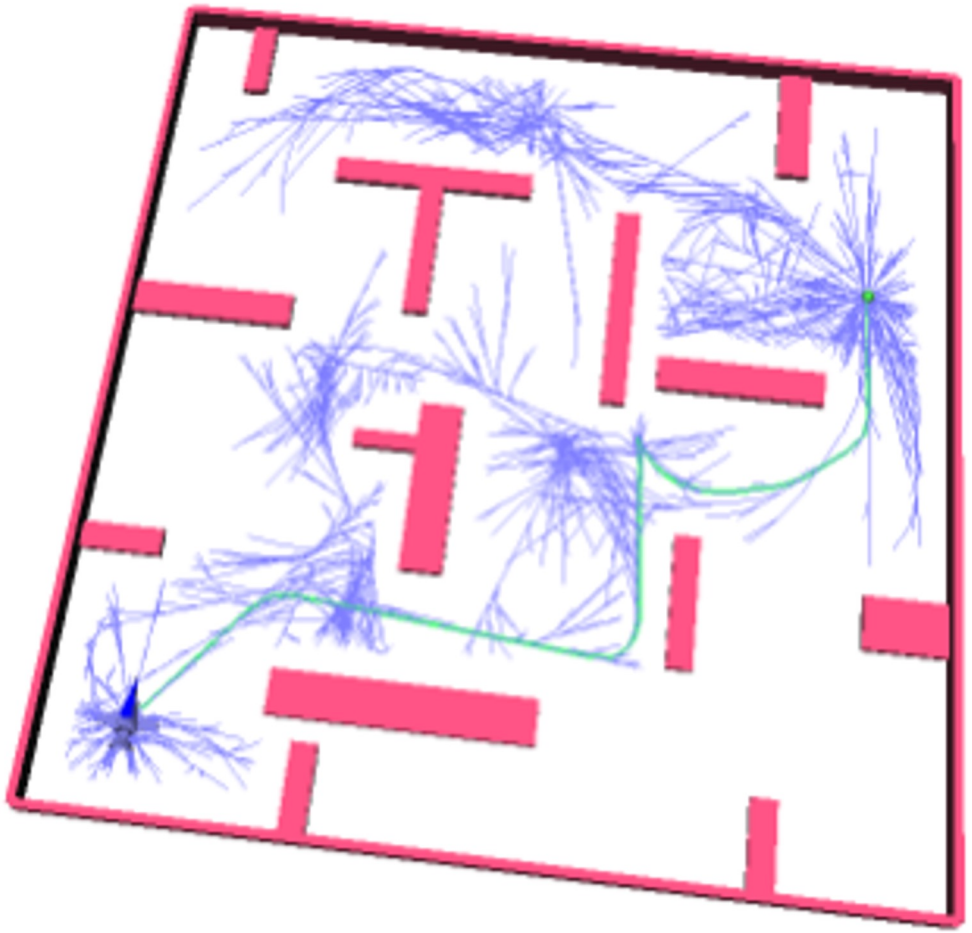
How to change the algorithm to handle high DoF robots?

$V \leftarrow \{x_{init}\}; \quad E \leftarrow \emptyset$
for $i = 1$ to N
  $G \leftarrow (V, E)$
  $x_{rand} \leftarrow RandomSample()$
  $x_{nearest} \leftarrow Nearest(G, x_{rand})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
  if $ObstacleFree(x_{nearest}, x_{new})$
    $V \leftarrow V \cup \{x_{new}\}$
    $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$



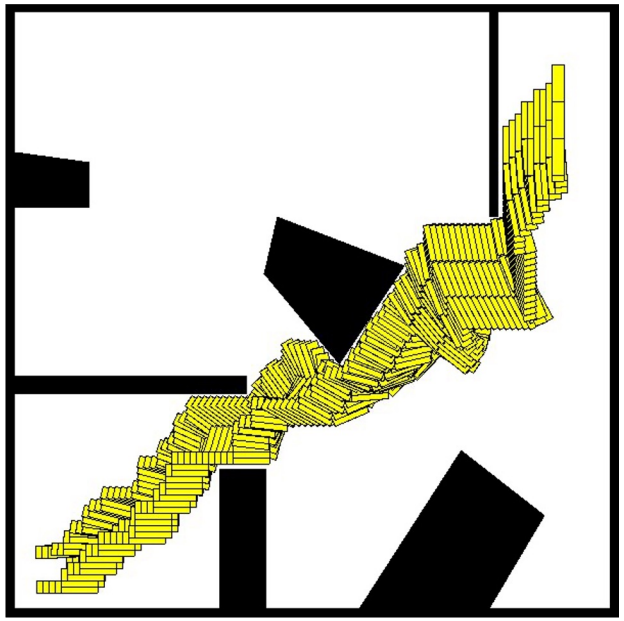
Examples: Hovercraft

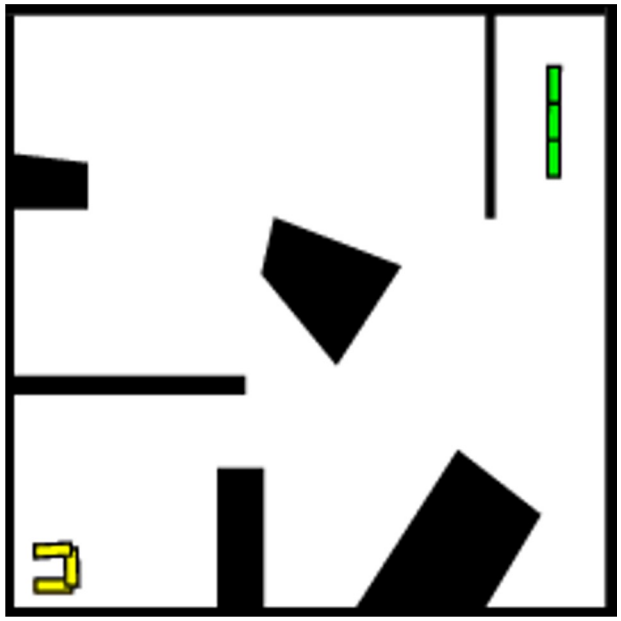
What's DoF of the robot?



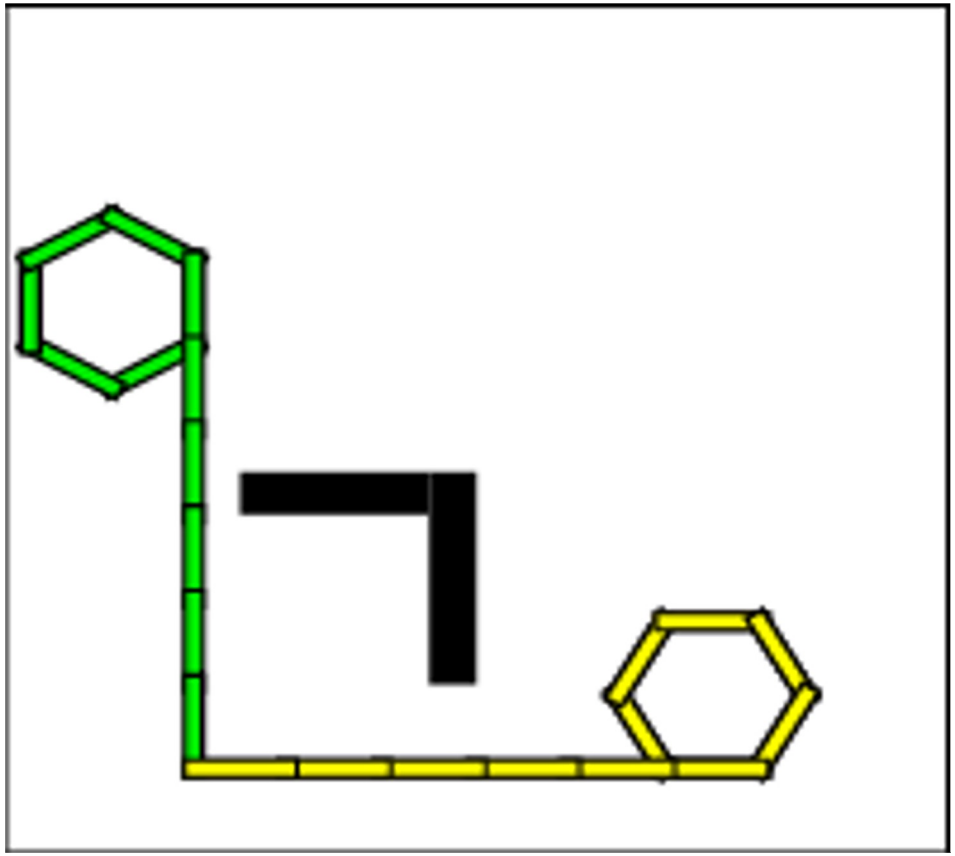
Examples: Articulated Robot

What's DoF
of the robot?



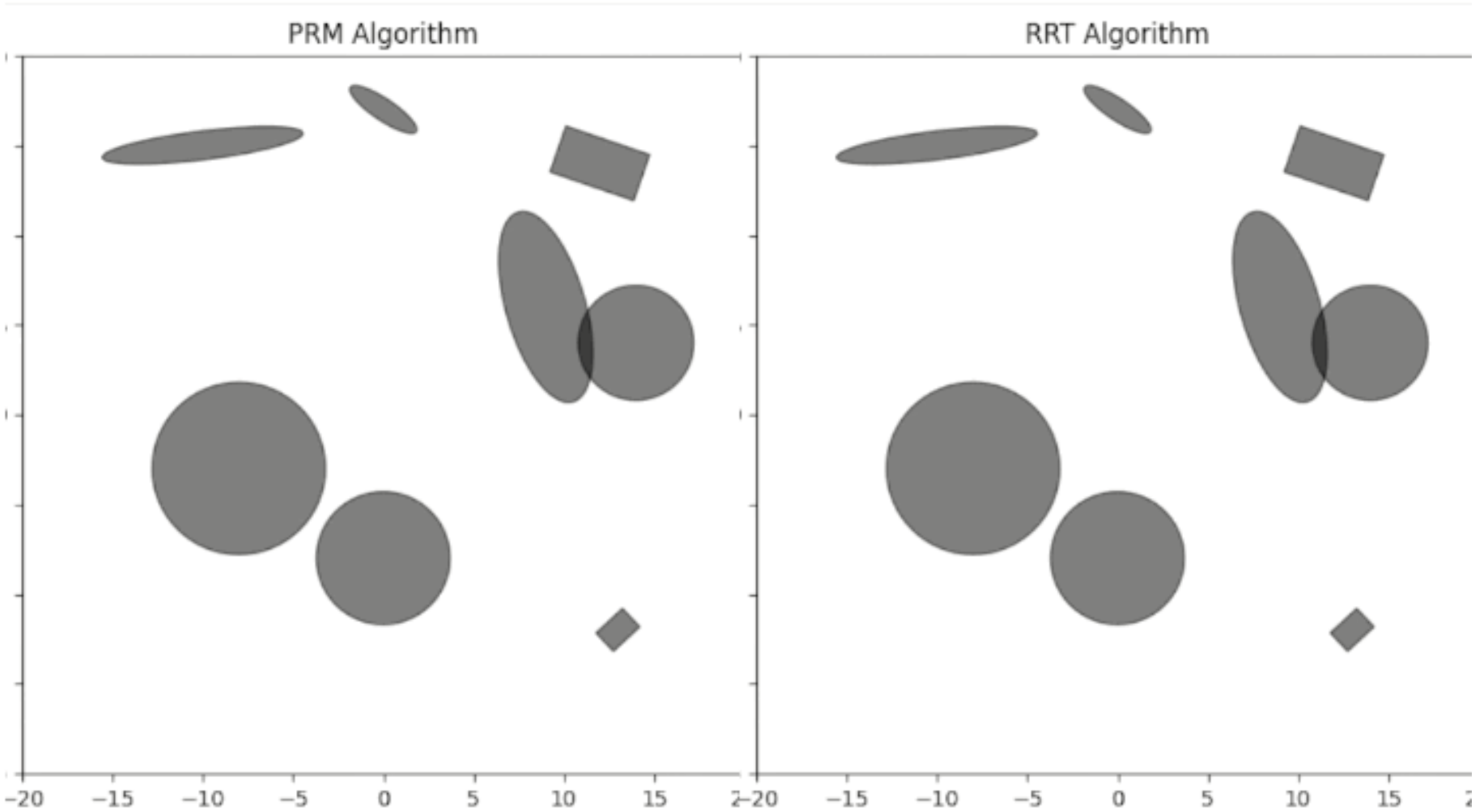


Examples: Highly Articulated Robot



Rapidly-Exploring Random Tree (RRT)

- **Advantages of RRT**: very fast
  - works well for dynamic environments (scene can change between each planning episode, but not within)
  - Work well for high degree of freedom planning problem.
- **Disadvantages**: Not optimal
  - In fact, it has been proven that the probability of RRT converging to an optimal solution is ZERO


PRM Algorithm
RRT Algorithm
−20
−15
−10
−5
0
5
10
15


That’s it for today!
Questions?