Robot Path Planning

Sampling-Based Planners

- PRM: Probabilistic Roadmap Methods
- RRTs: Rapidly-exploring Random Trees

Sampling-Based Planners

- Explicit Geometry based planners (VGRAPH, Voronoi) are impractical in high dimensional spaces.
- Exact solutions with complex geometries are provably exponential
- Sampling based planners can often create plans in high-dimensional spaces efficiently
- Rather than *Compute* the collision free space explicitly, we *Sample* it

Sampling-Based Planners

- Idea: Generate random configuration of robot in C-Space
- Check to see if it is in C-Free or collides with a member of C-Obstacles
- Find N collision free configs, link them into a graph
- Uses fast collision detection - full knowledge of C-Obstacles
- Collision detection is separate module - can be application and robot specific
- Different approaches for single-query and multi-query requests:
  - Single: Is there a path from Configuration A to Configuration B?
  - Multiple: Is there a path between ANY 2 configurations

Sampling-Based Planners

- Complete Planner: always answers a path planning query correctly in bounded time, including no-path
- Probabilistic Complete Planner: if a solution exists, planner will eventually find it, using denser and denser random sampling
- Resolution Complete Planner: same as above but based on a deterministic sampling (e.g. sampling on a fixed grid).

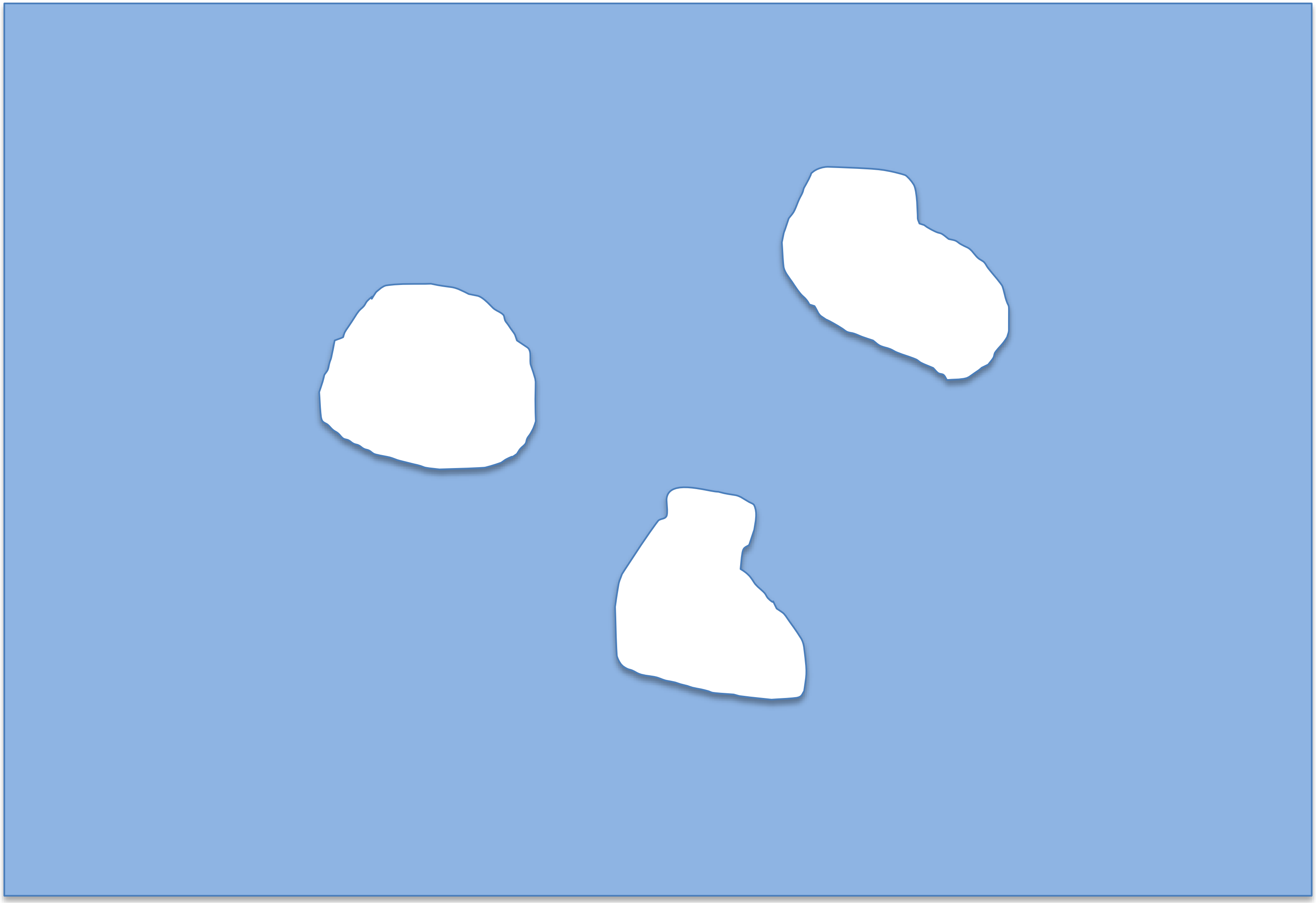
Probabilistic Roadmap Planner - PRM

- Roadmap is a graph G(V,E)
- Robot configuration q in Q-Free is a vertex
- Edge (q1, q2) implies collision-free path between these robot configurations – local planner needed here
- A metric is needed for distance between configurations: dist(q1,q2) (e.g. Euclidean distance)
- Uses coarse sampling of the nodes, and fine sampling of the edges
- Collison free vertices, edges form a roadmap in Q-Free

PRM Roadmap Construction

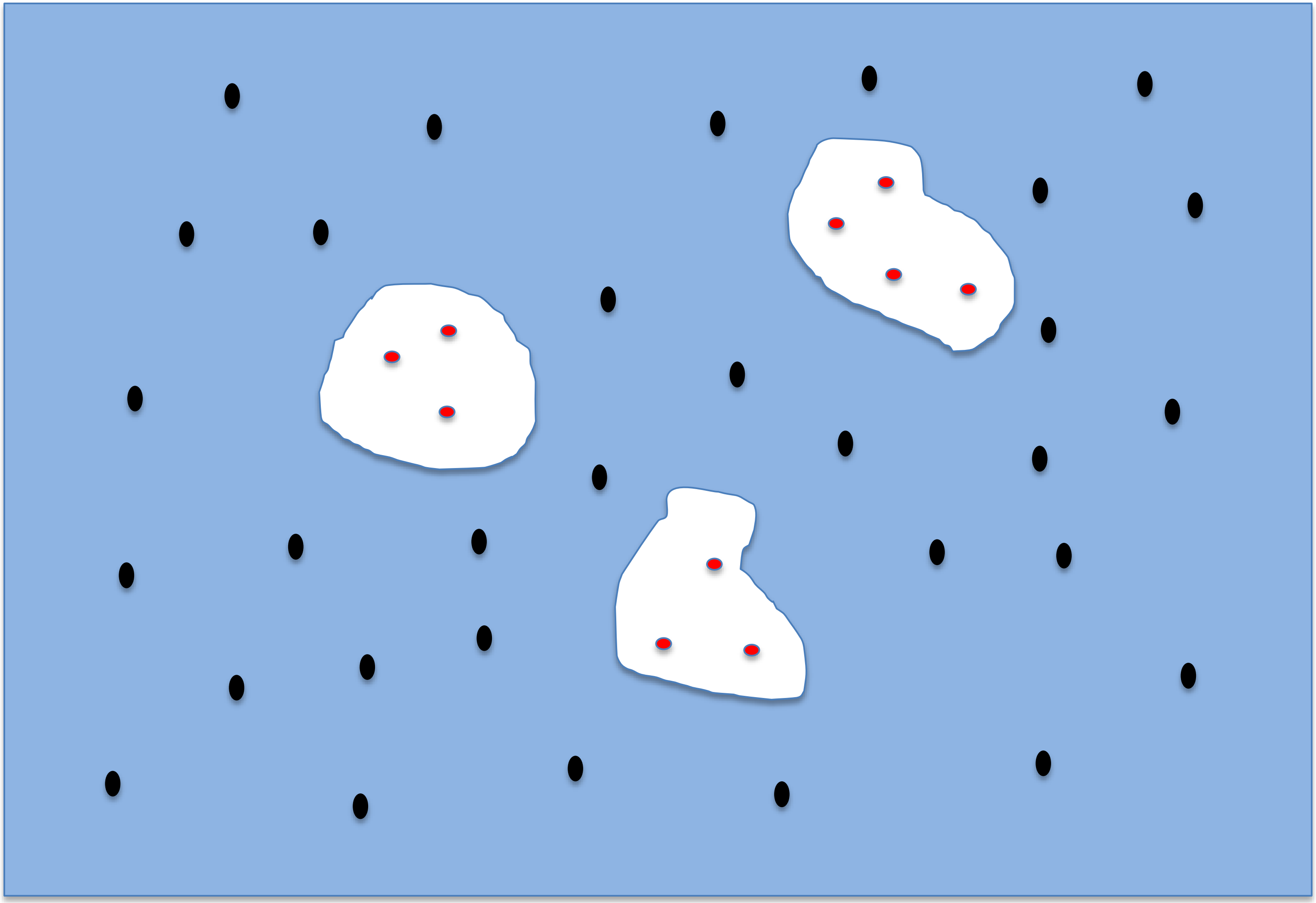
- Initially empty roadmap Graph G
- A robot configuration q is randomly chosen
- If q→Q-Free (collision free configuration) then add to G
- Repeat until N vertices chosen
- For each vertex q, select k nearest neighbors
- <u>Local planner</u> tries to connect q to neighbor q'
- If connect successful (i.e. collision free local path), add edge (q, q')

PRM



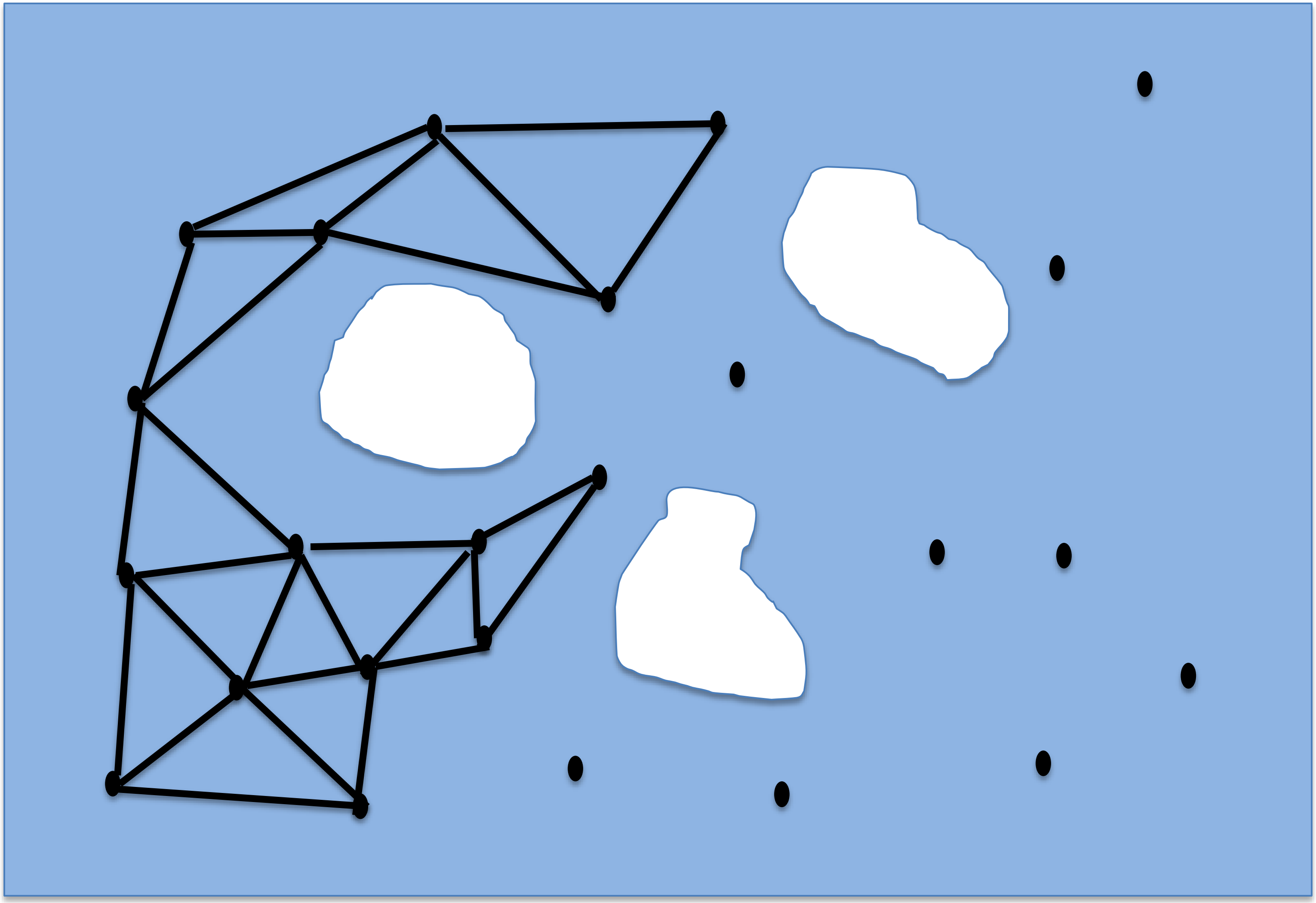
2D planar environment with obstacles

PRM



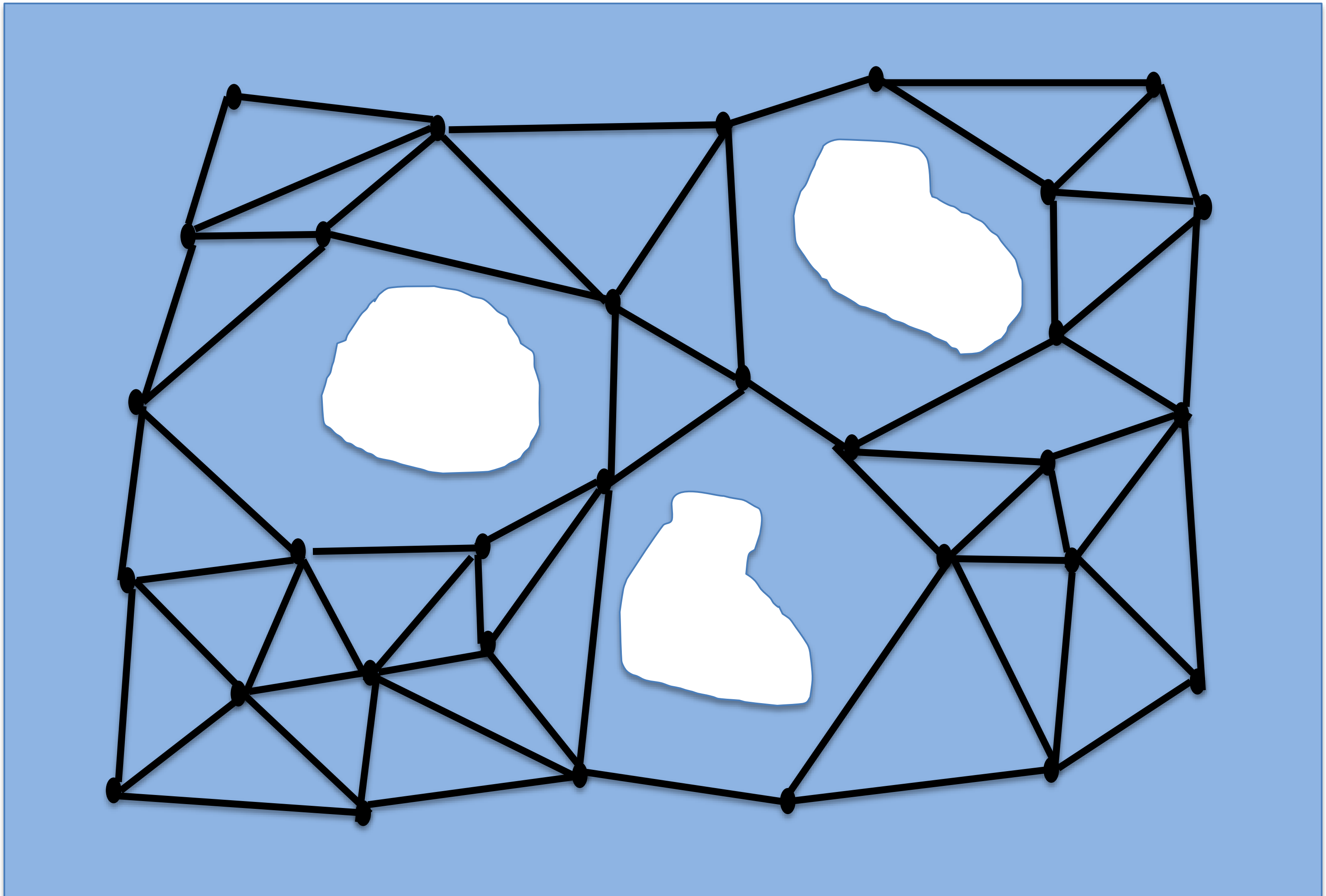
1. Randomly sample C-Space for N collision-free configurations

PRM



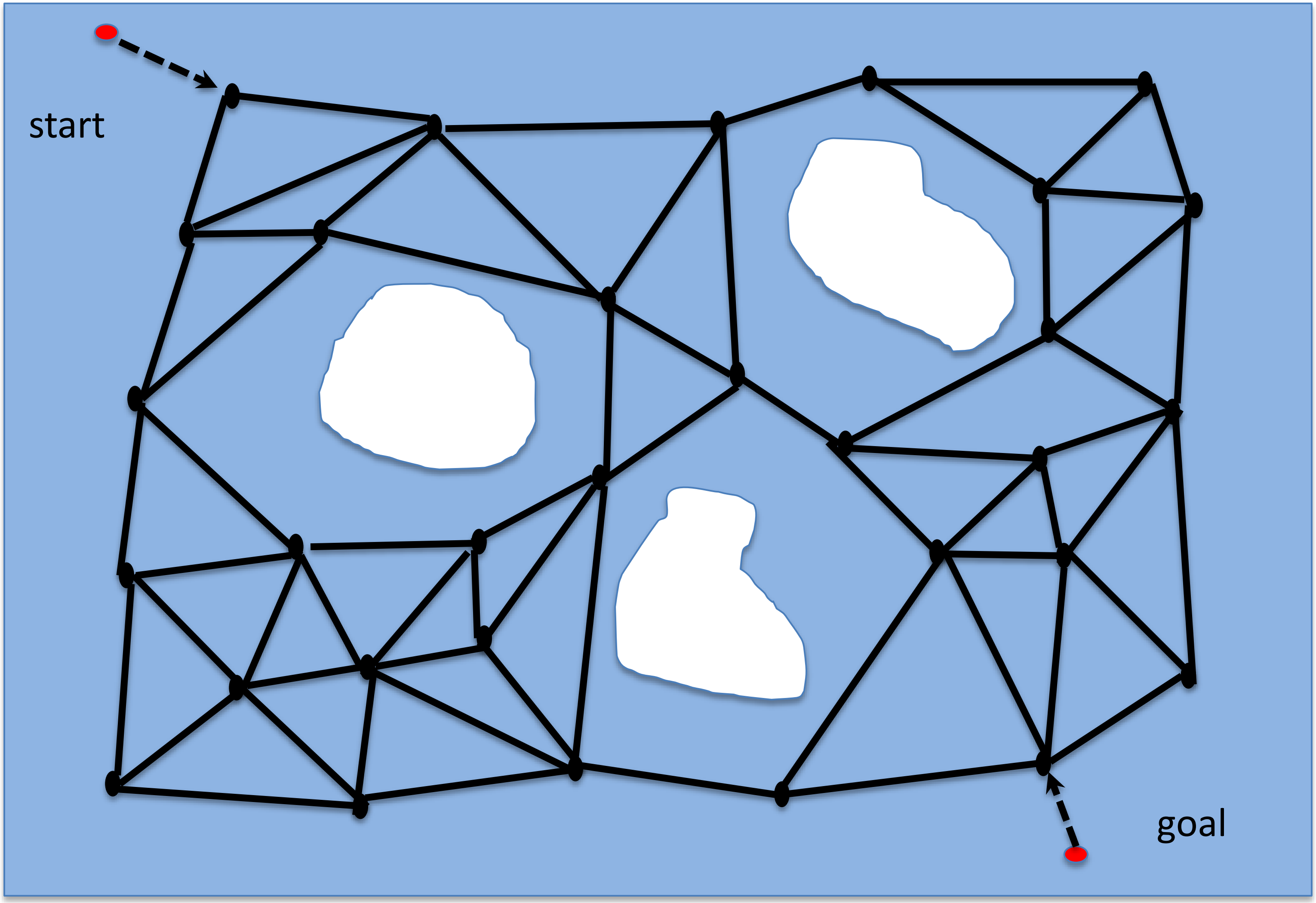
2. Link each vertex in Q-Free with K nearest neighbors

PRM



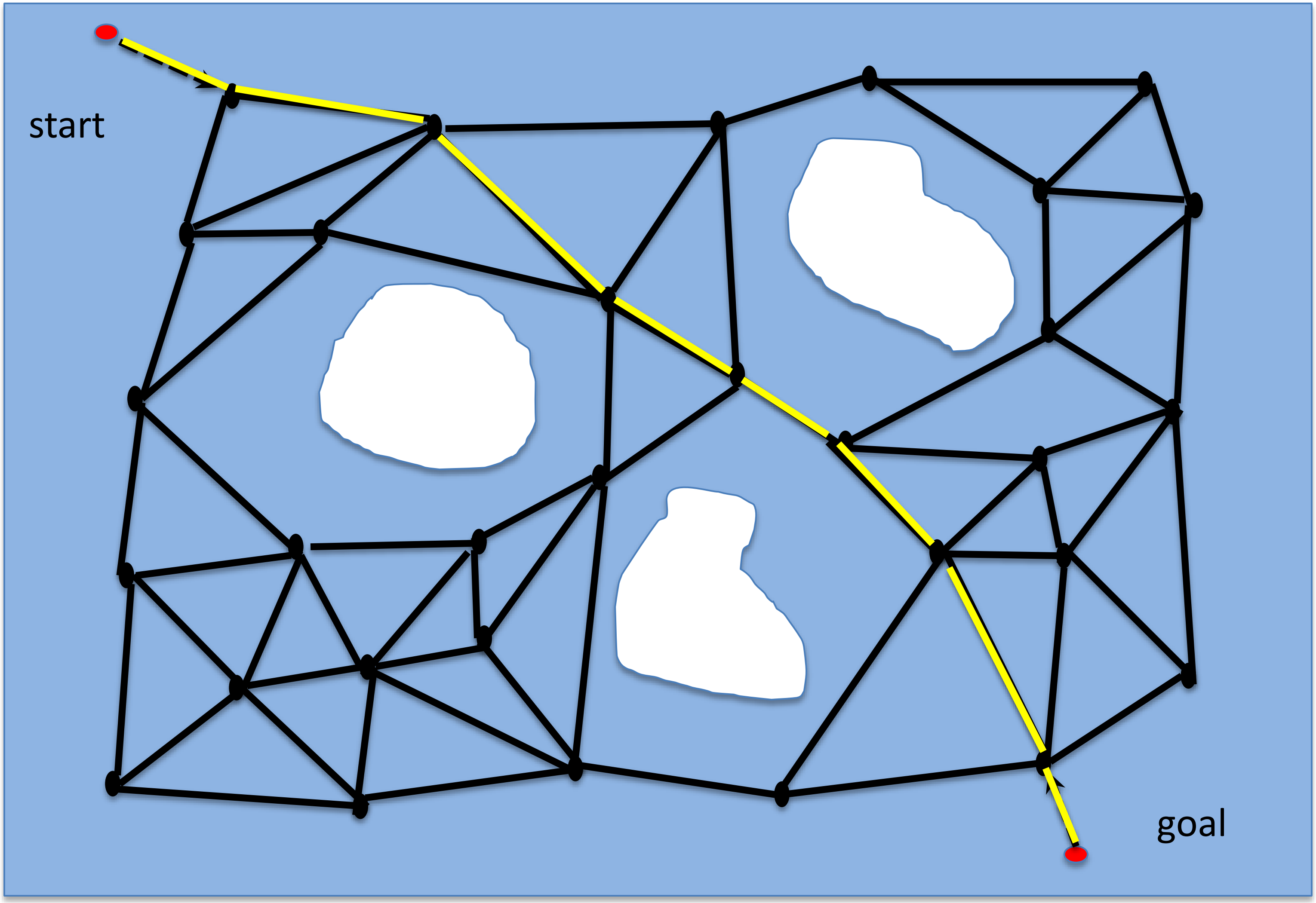
2. Link each vertex in Q-Free with K nearest neighbors

PRM



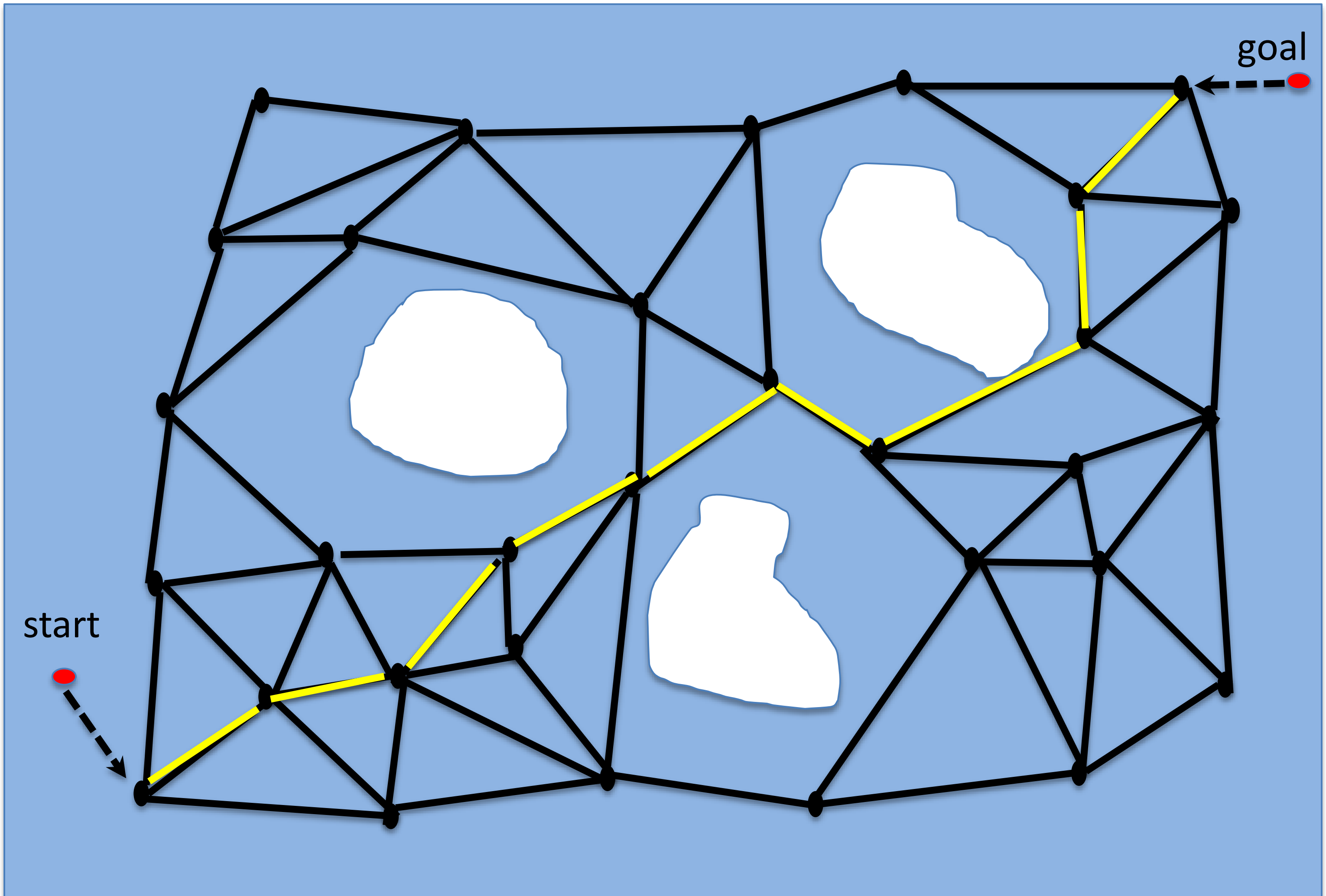
3. Connect start and goal to nearest node in roadmap

PRM



4. Graph Search for shortest path

PRM



Handles multiple queries-once on roadmap, finds a path

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
    repeat
        $q \leftarrow$ a random configuration in $\mathcal{Q}$
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
    for all $q' \in N_q$ **do**
        if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ `NIL` **then**
            $E \leftarrow E \cup \{(q, q')\}$
        end if
    end for
end for

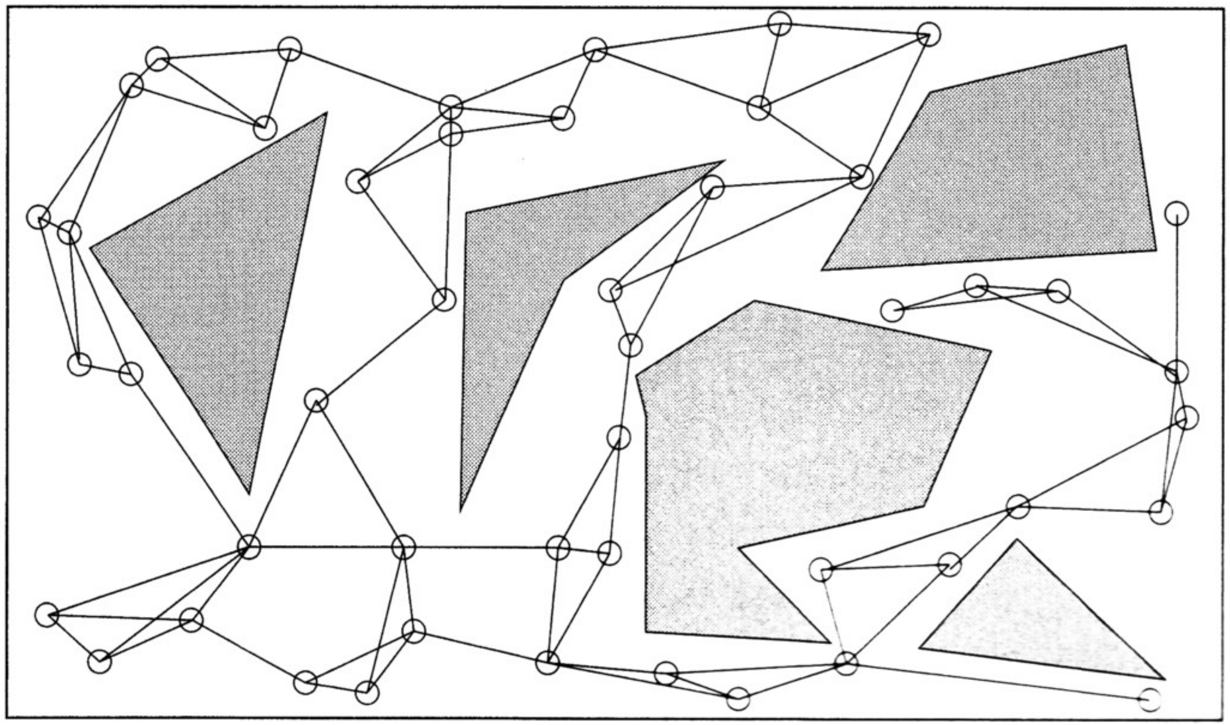


Figure 7.3 An example of a roadmap for a point robot in a two-dimensional Euclidean space. The gray areas are obstacles. The empty circles correspond to the nodes of the roadmap. The straight lines between circles correspond to edges. The number of k closest neighbors for the construction of the roadmap is three. The degree of a node can be greater than three since it may be included in the closest neighbor list of many nodes.

PRM Planner: Step 2, Finding a Path

- Given q_init and q_goal, need to connect each to the roadmap
- Find *k* nearest neigbors of q_init and q_goal in roadmap, plan local path Δ
- Problem: Roadmap Graph may have disconnected components…
- Need to find connections from q_init, q_goal to same component
- Once on roadmap, use Dijkstra algorithm

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

$N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
$N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
$V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
repeat
    if $\Delta(q_{\text{init}}, q') \neq \texttt{NIL}$ **then**
        $E \leftarrow (q_{\text{init}}, q') \cup E$
    else
        set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
    end if
until a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
repeat
    if $\Delta(q_{\text{goal}}, q') \neq \texttt{NIL}$ **then**
        $E \leftarrow (q_{\text{goal}}, q') \cup E$
    else
        set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
    end if
until a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
$P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
if P is not empty **then**
    return P
else
    return `failure`
end if

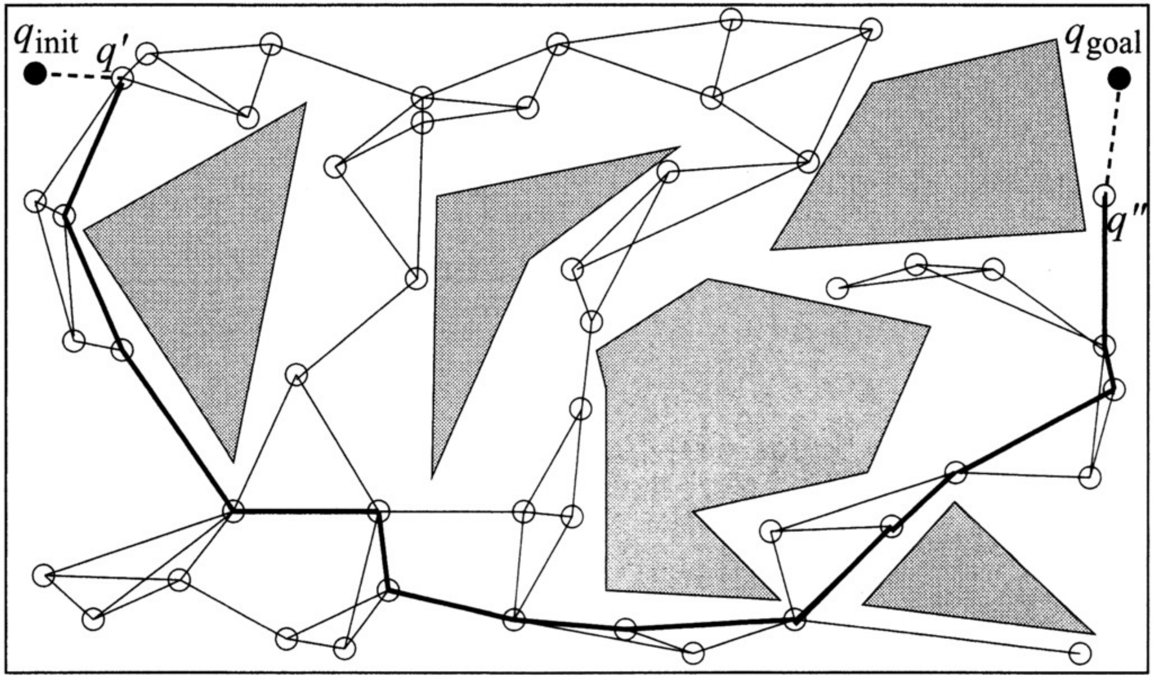
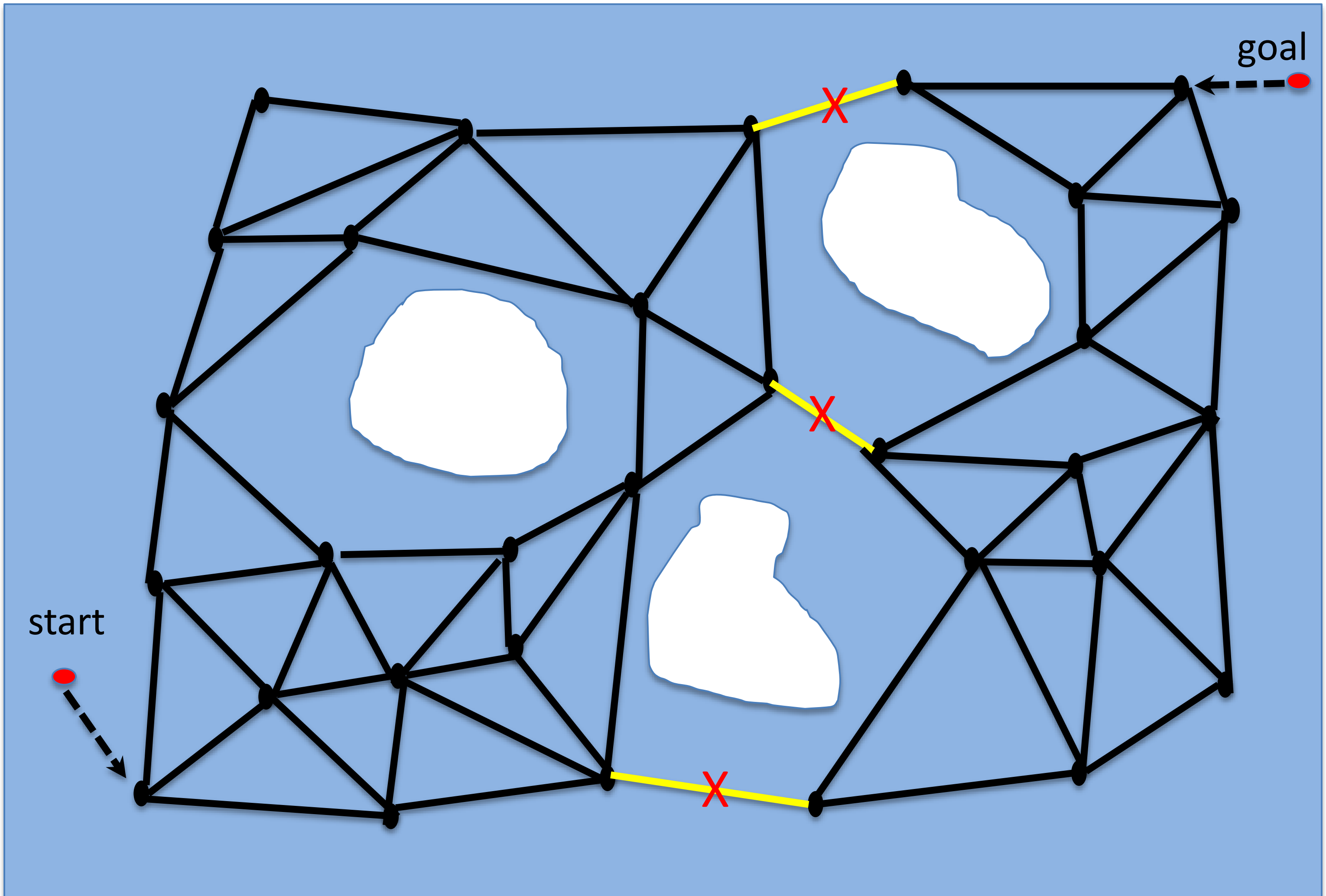


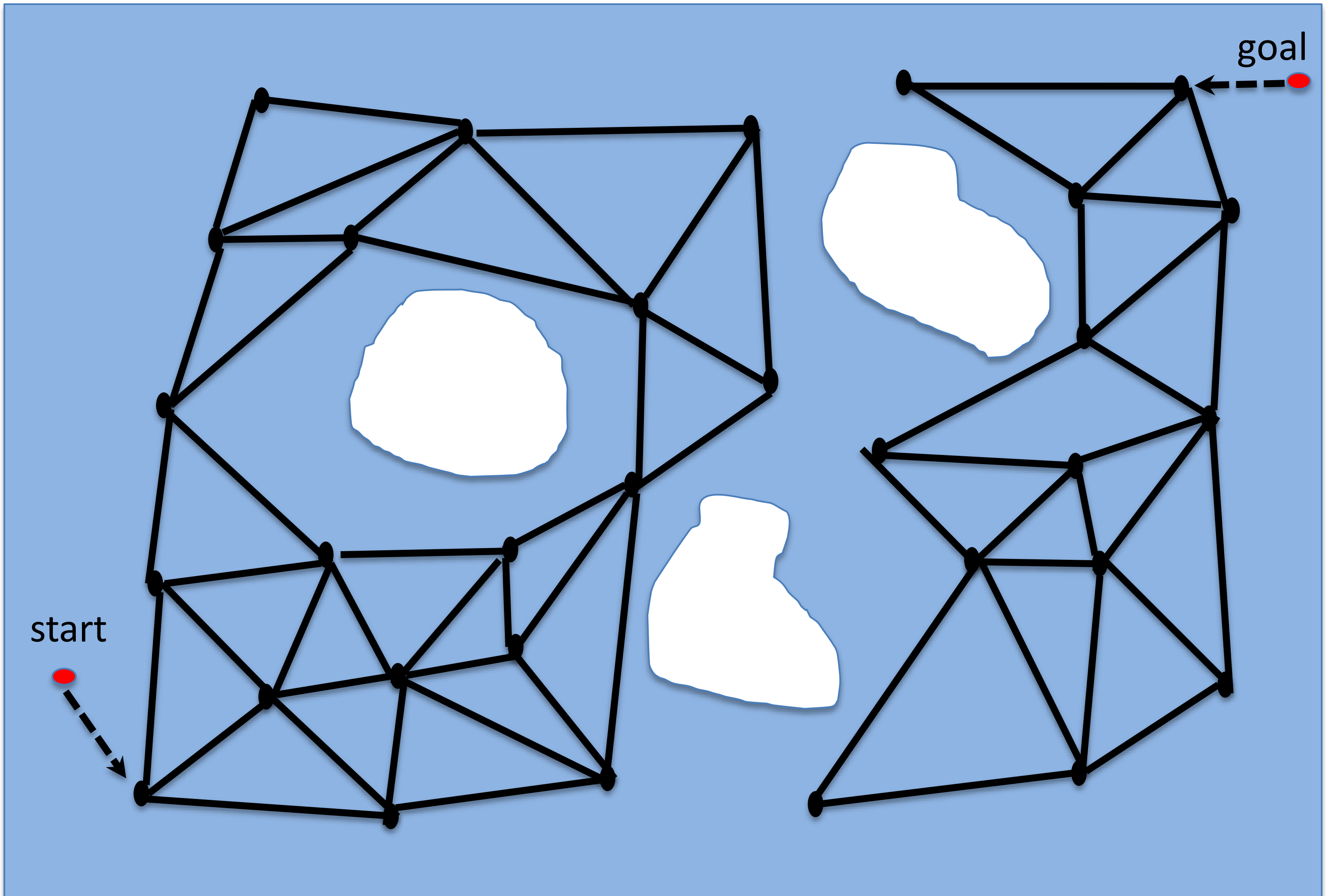
Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

PRM



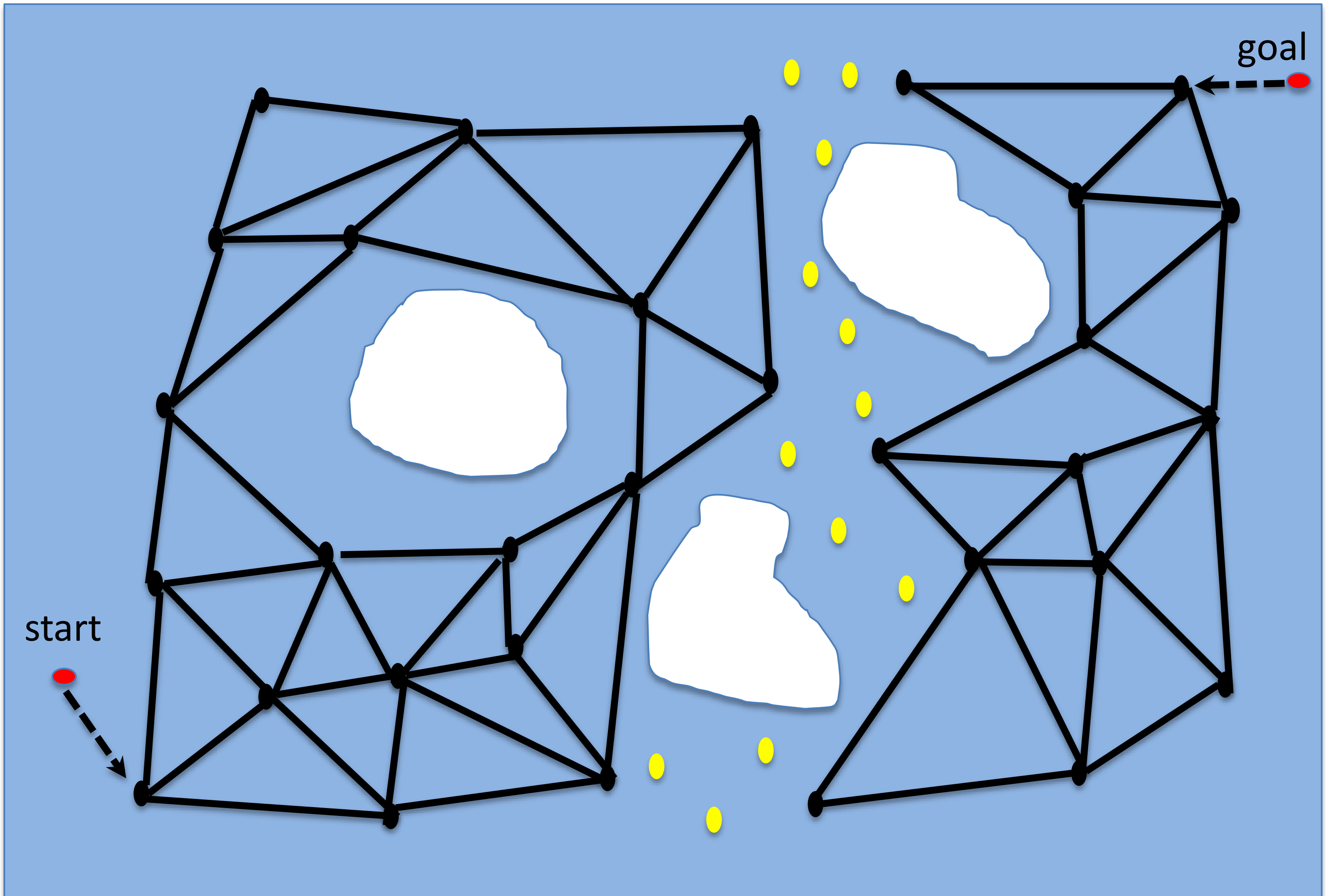
Problem: Graph may not be fully connected!

PRM



Problem: Graph may not be fully connected!

PRM



Solution: Denser sampling – more and closer neighbors

PRM Planner Details

Choosing configurations:

- Use random sampling of entire C-Space
- However, collision free areas are easy to navigate, don't need many samples
- Collision regions are where planner needs to find denser samples –tight navigation areas
- OBPRM: Obstacle-Based PRM
  - if config q is in collision, then re-sample in the vicinity of the collision to find safe config near obstacle
  - Choose random direction and small distance from q to generate nearby sample in Q-Free
  - Biases sampling to regions where collisions likely

PRM Planner Details

Finding nearest neighbors:

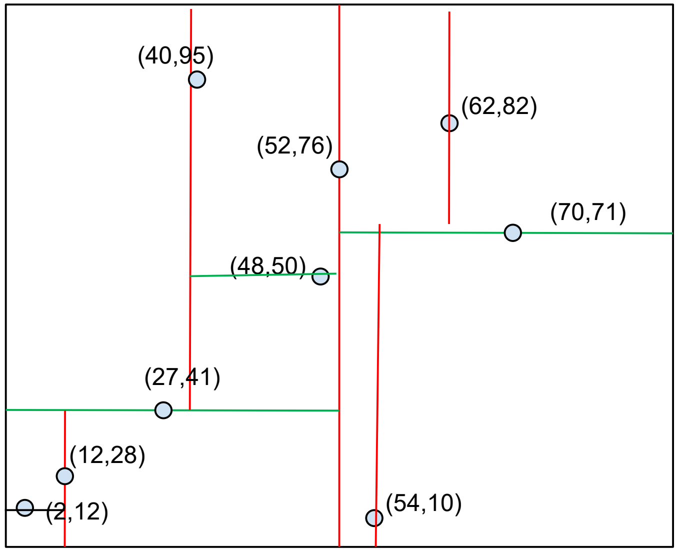
- Brute force search – cost is O(N)
- Faster method: Use K-D tree

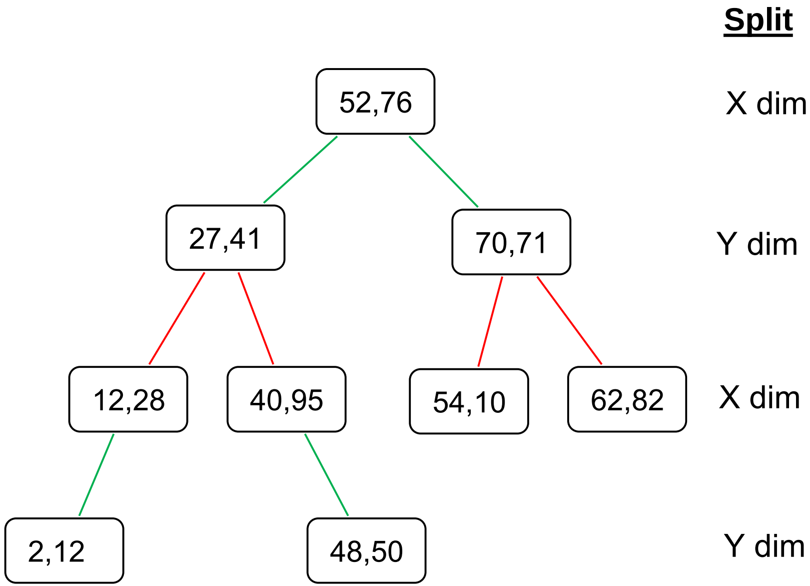
- K-D tree decomposes dimensions by splitting into 2 regions alternating each dimension
- Search is fast and efficient
- Cost is O(sqrt(N)) for dimension D=2

KD-Tree Construction

Order of insertion:

(52,76), (27,41), (12,28), (70,71), (2,12), (40,95), (62,82), (54,10), (48,50)

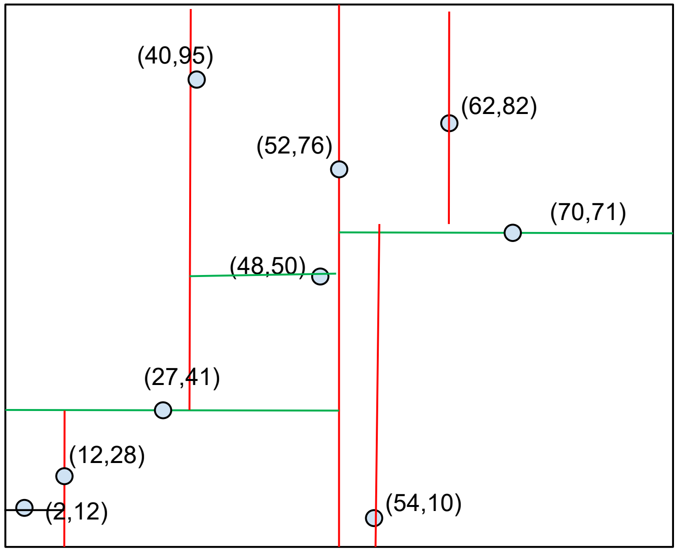


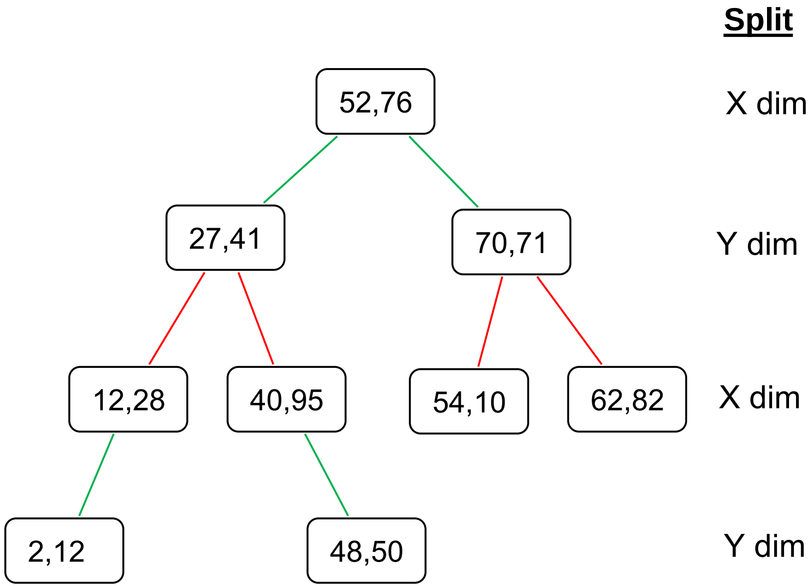


KD-Tree Fast Range Query

Find points in Rectangle around Query point. Example: find all points in rectangle 10<X<30, 25<Y<45.

Once points found, a simple distance calculation finds nearest neighbor

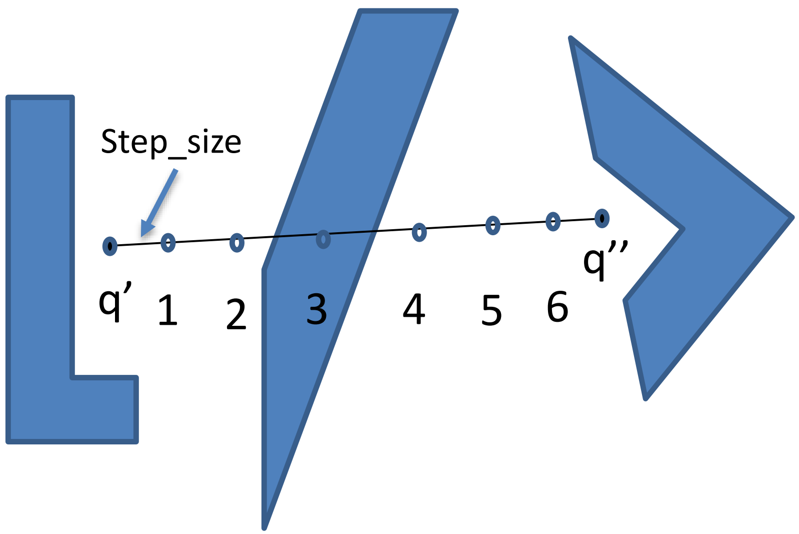


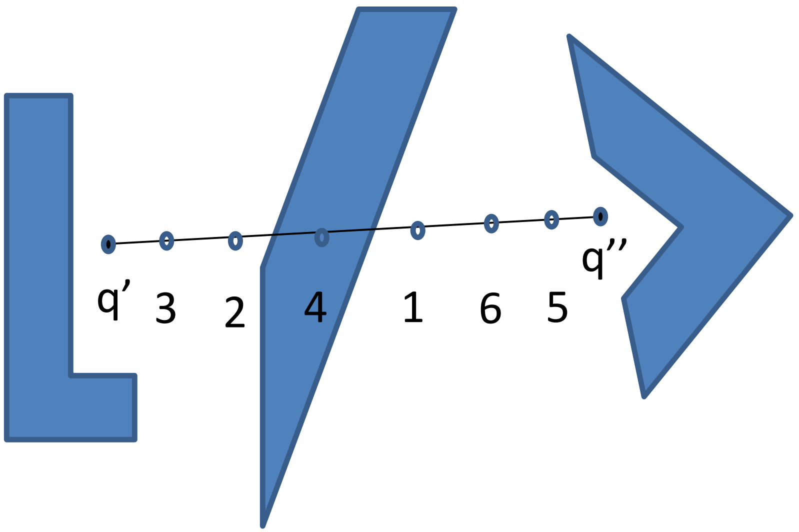


returns (27,41),(12,28)- only searches part of tree

Local Planner

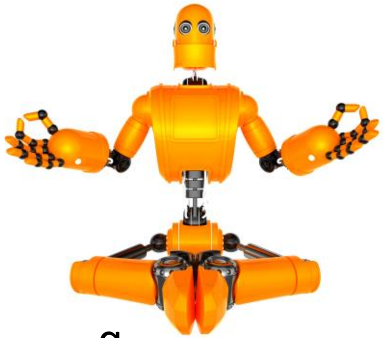
- Used to find collision free paths between nearby nodes
- Also used to connect q_start and q_goal to the roadmap
- Called frequently, needs to be efficient
- Incremental: sample along straight line path in C-Space
- Step-size needs to be small to find collisions
- Subdivision: Check midpoint of straight line path, recursively sample segment’s midpoints for collisions





Distance Function

q1



q

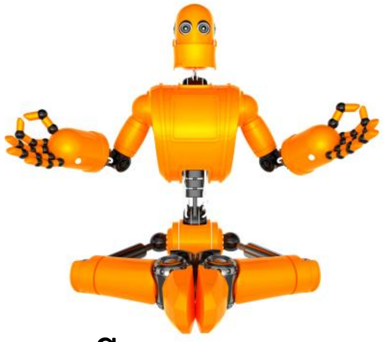
q2

- Is configuration q “closer” to q1 or q2?
- Distance metric needed between 2 configurations
- Ideally, distance is the swept volume of robot as it moves between configs q and q’ - difficult to compute
- Each config is vector of joint angles
- Possible metric: take sum of joint angle differences?

$$\sum_{i=1}^{N} (\theta_i - \theta_i')^2$$

But this ignores movement (trans. and rotation) of the robot!

Distance Function



- Articulated robots: choose set of P points on robot, concatenate them, and create a vector of size P · D (dimension of workspace).
- Intuitively, a "sampling" of the object's Euclidean domain.
- For configuration q, **sample(q)** is the vector of P points transformed by the <u>translation and rotation</u> that is config q
- Transform each of the P points into the vector **sample(q).** Do same for configuration q', create **sample(q').**
- In 3D, distance is Euclidean distance between the 3·P vectors:

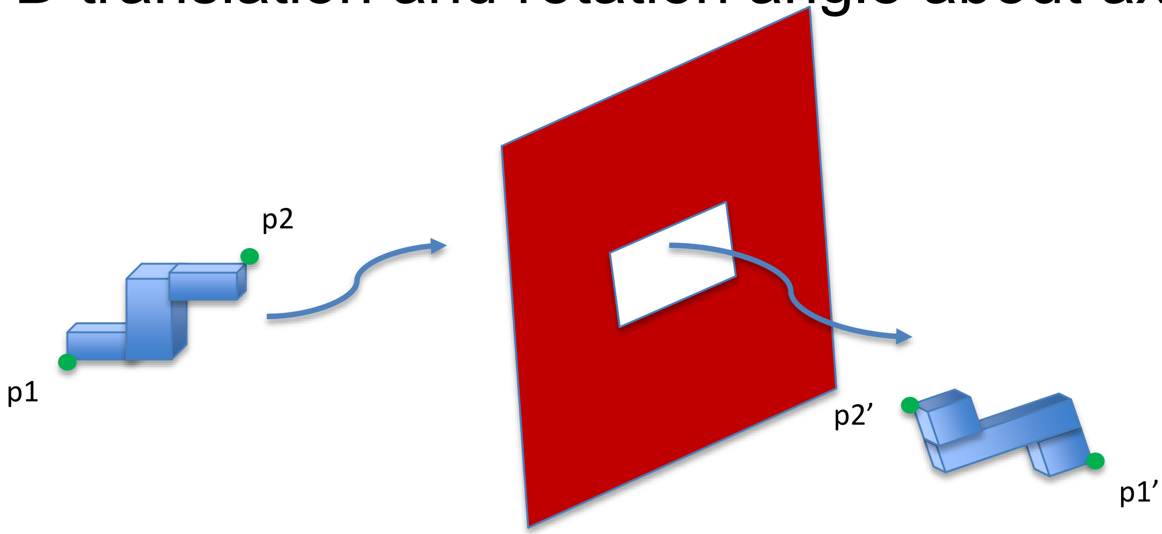
$$\mathbf{d(q,q') = \| sample(q) - sample(q') \|}$$

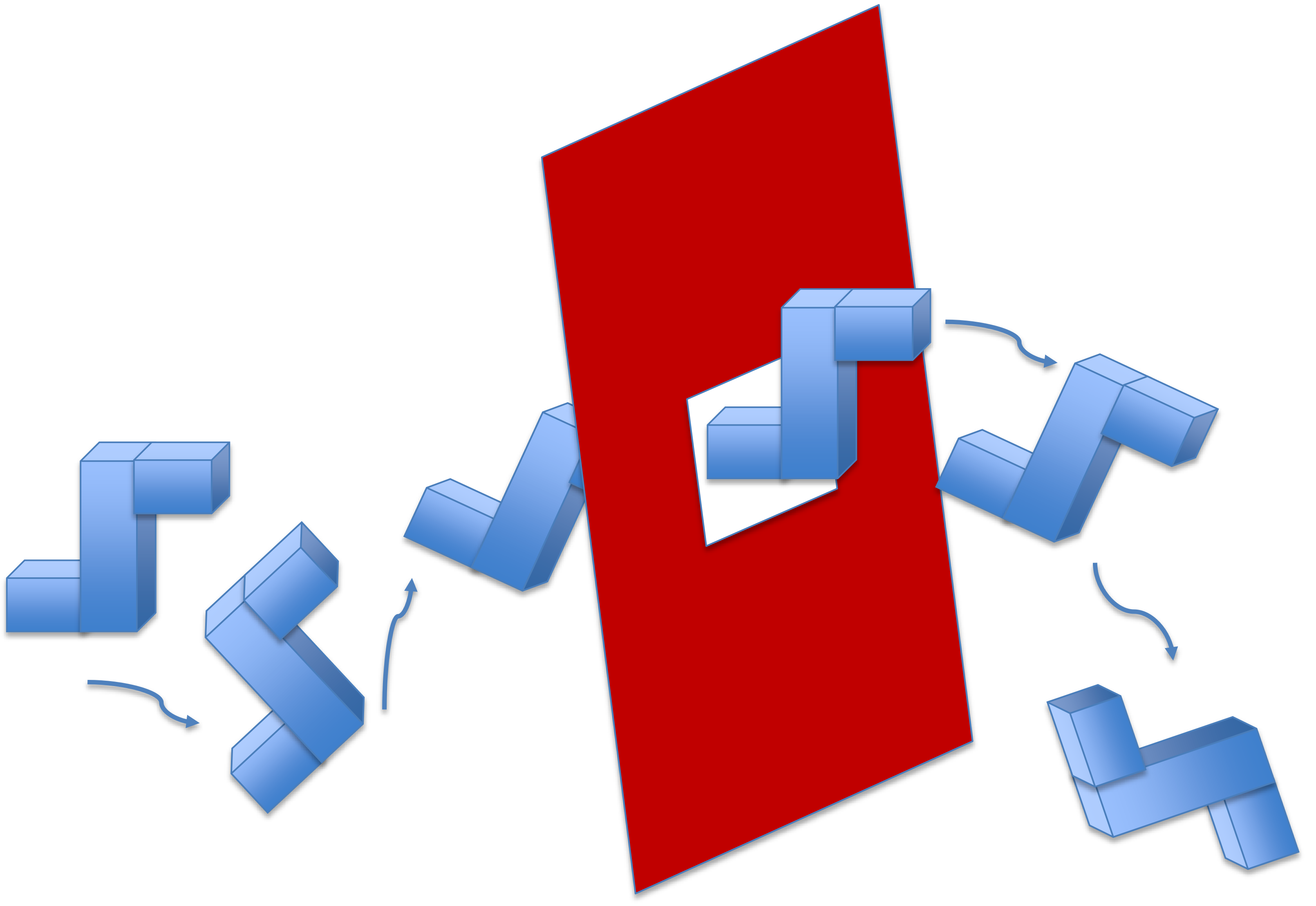
- Rigid robot: just choose 2 points of maximal extent as samples

6-DOF Path Planning Example

- Robot: Rigid non-convex object in 3 space
- Obstacle: Solid wall with small opening
- Configuration of solid object: q=(Translation, Rotation)
- Random X,Y,Z configuration is chosen for translation
- Random axis and angle of rotation chosen for rotation
- Distance measure uses 2 extreme points on object,

  p1 and p2: ||p1 - p1'|| + ||p2 - p2'||
- Local planner: Check for collision by interpolating along 3-D translation and rotation angle about axis



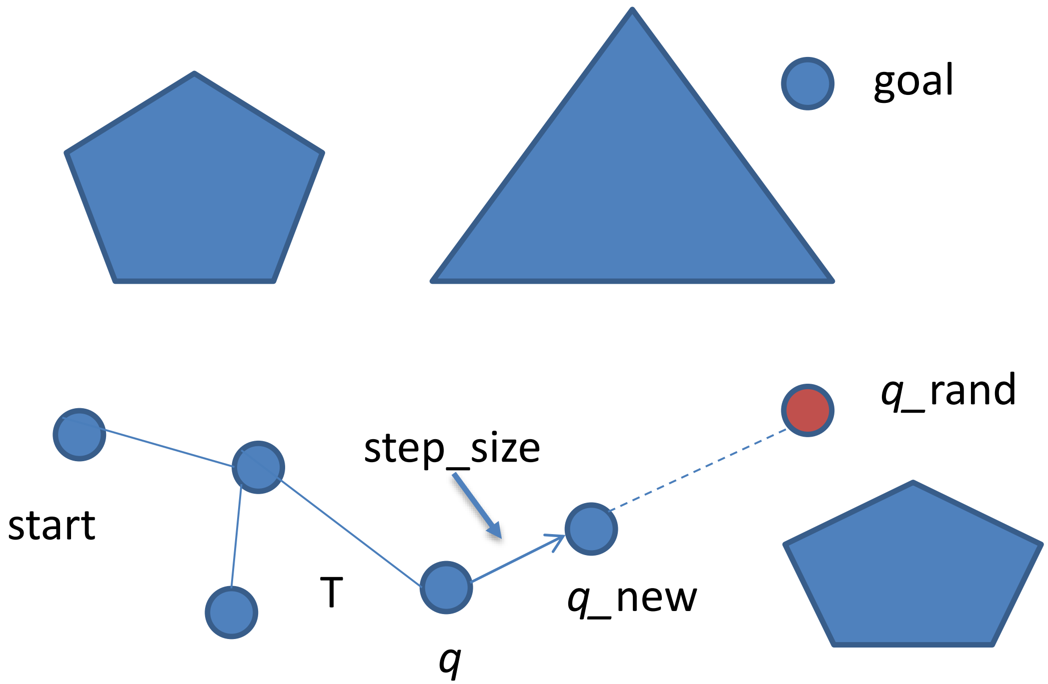


video

RRT: Rapidly-exploring Random Trees

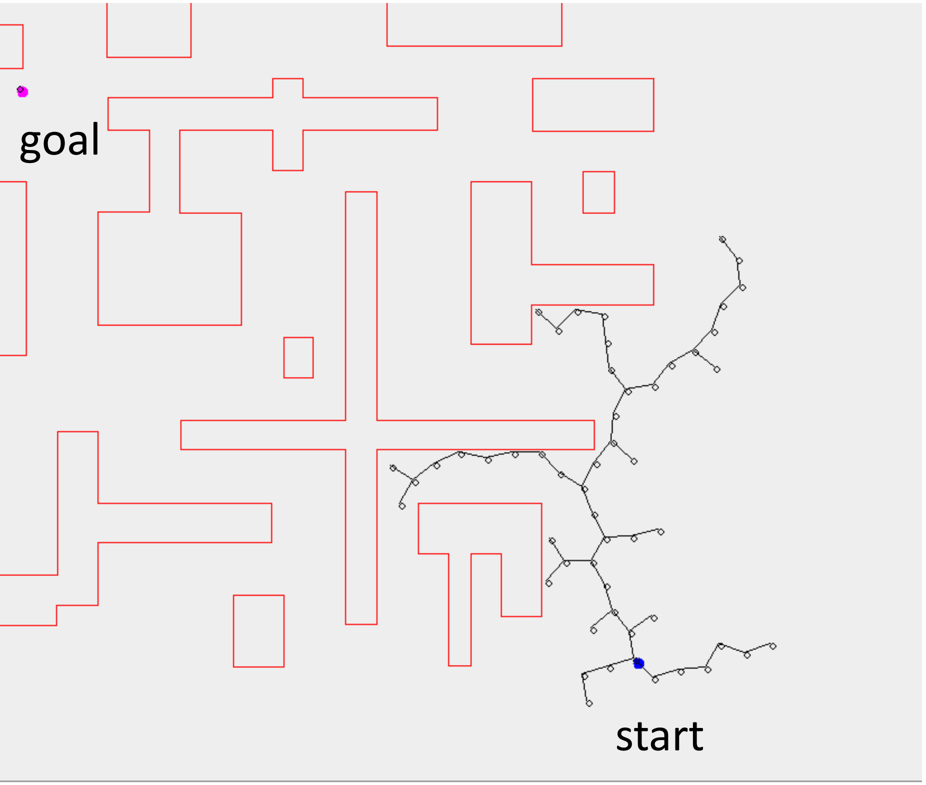
- Single query planner to get from config A to config B
- Randomly sample Q-Free for path from q_start to q_goal, growing a tree towards goal
- Can use 2 trees, rooted at q_start and q_goal.
- As trees grow, the eventually share a common node, and are merged into a path

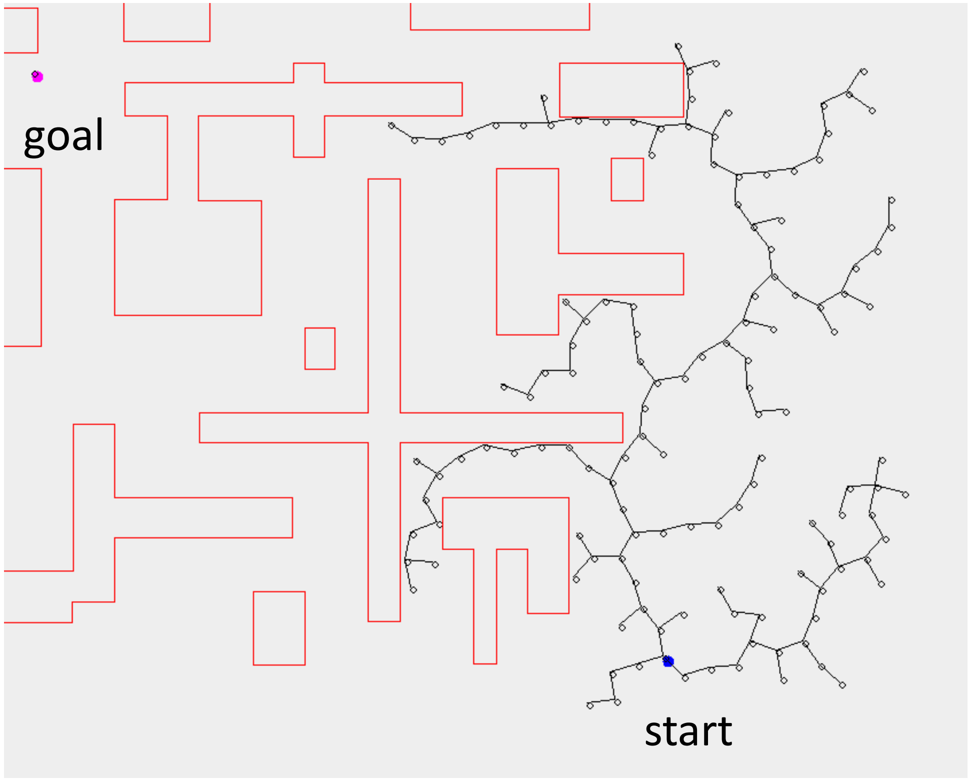
RRT: Build Tree Algorithm



- Start node is root of tree
- Generate new random config q_rand
- Find nearest tree node q
- Move along path (q, q_rand) distance step_size
- If collision free, add q_new as new tree node
- Repeat…

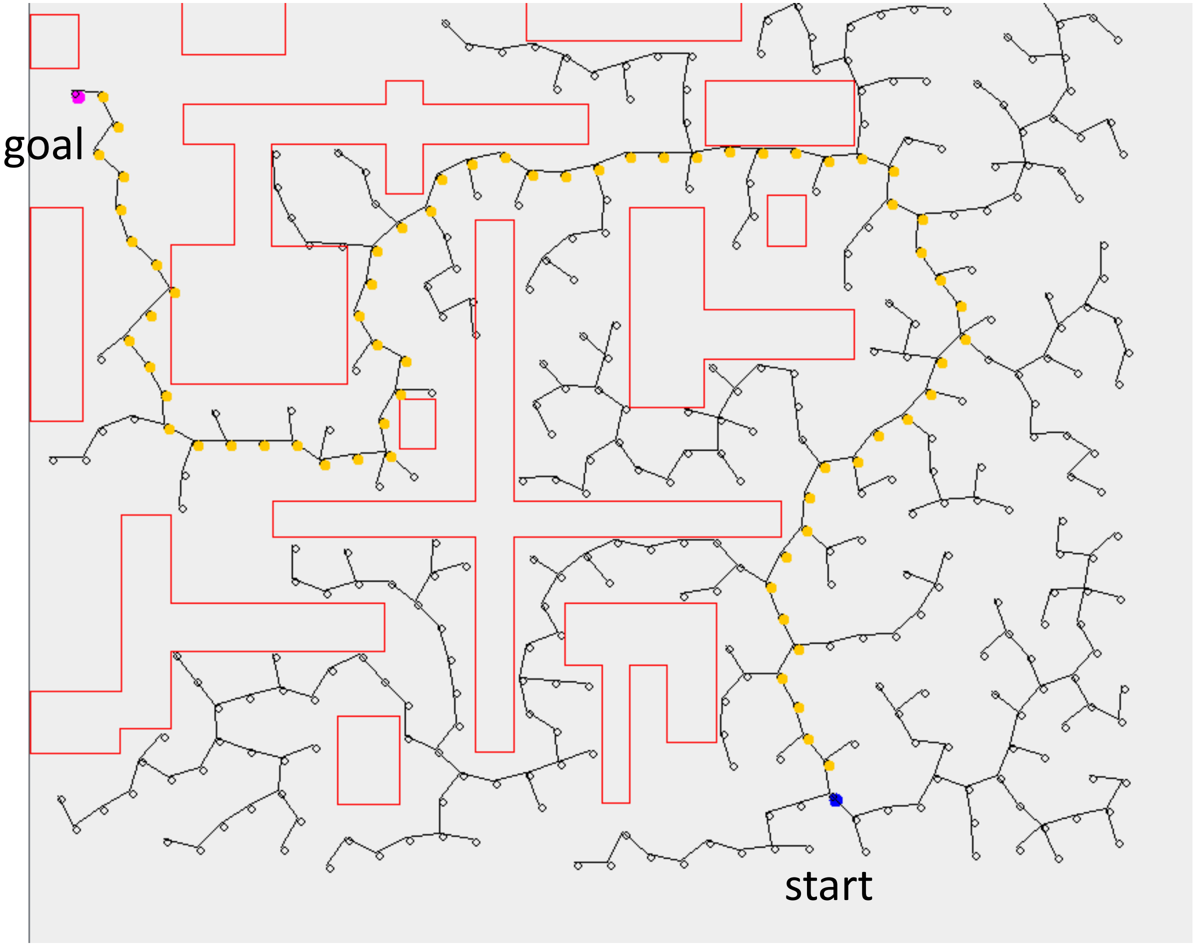
RRTs





- Expand tree, one node a time, from start node
- Randomly generate new sample config each time
- Try to connect sample to nearest node in the tree
- Create new node small distance (step_size) towards sample (if collision free) – local planner invoked here

RRTs



- Once tree reaches the goal, we have a path
- Path is not optimal in any sense
- Path can be different each time - stochastic
- Scales to higher dimensions

Algorithm 10 Build RRT Algorithm

Input:

q_0: the configuration where the tree is rooted

n : the number of attempts to expand the tree

Output:

A tree $T = (V, E)$ that is rooted at q_0 and has $\leq n$ configurations

$V \leftarrow \{q_0\}$
$E \leftarrow \emptyset$
for $i = 1$ to n **do**
    $q_{\text{rand}} \leftarrow$ a randomly chosen free configuration
    extend RRT (T, q_{rand})
end for
return T

Algorithm 11 Extend RRT Algorithm

Input:

$T = (V, E)$: an RRT

q: a configuration toward which the tree T is grown

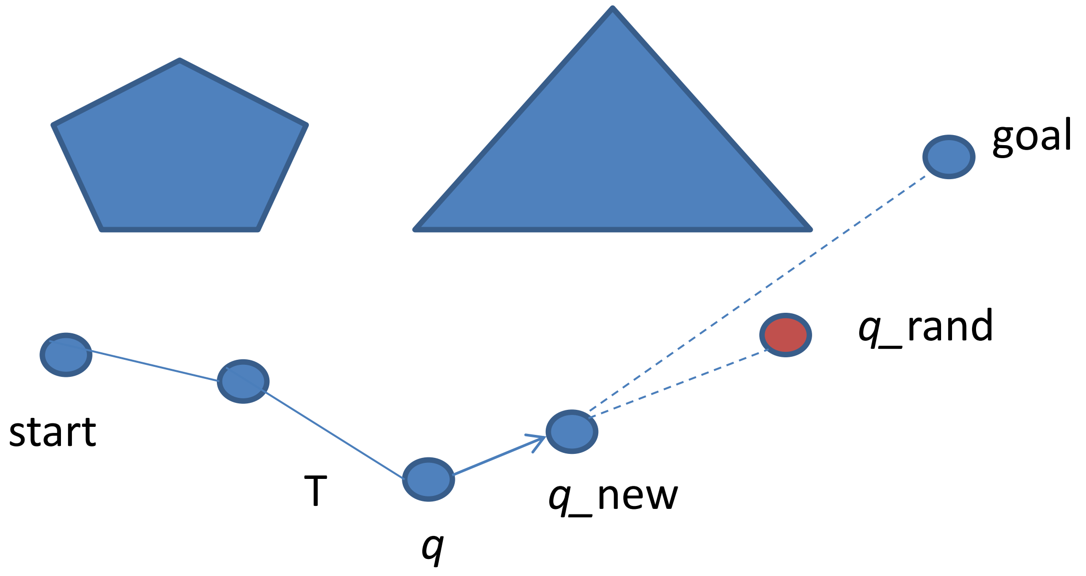
Output:

A new configuration q_{new} toward q, or NIL in case of failure

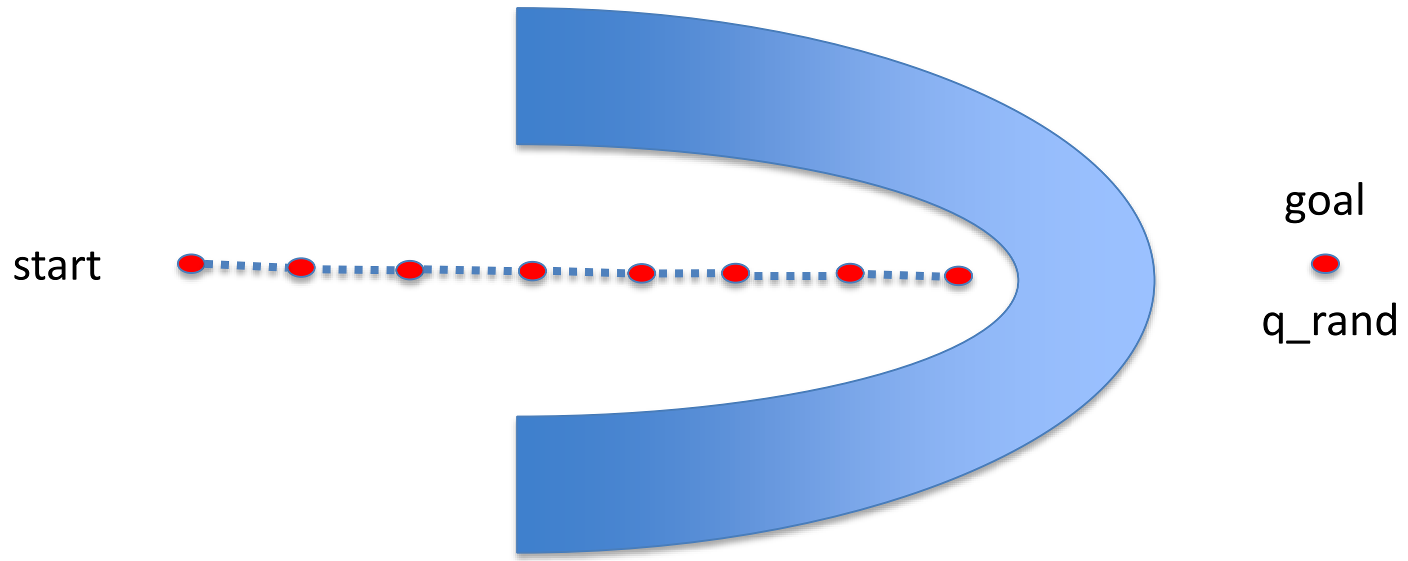
$q_{\text{near}} \leftarrow$ closest neighbor of q in T
$q_{\text{new}} \leftarrow$ progress q_{near} by step_size along the straight line in Q between q_{near} and q_{rand}
if q_{new} is collision-free **then**
    $V \leftarrow V \cup \{q_{\text{new}}\}$
    $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}})\}$
    return q_{new}
end if
return NIL

RRT: How do we reach the goal?

1. As we add node q_new, see if it is within step_size of goal
   - If so, see if we can add edge (q_new, q_goal)
2. Bias: q_rand determines what direction we go
   - What if q_rand == q_goal?
   - Greedy algorithm, can get stuck in local minima
   - Idea: Use q_goal as q_rand just some of the time
   - Moves tree towards goal every now and then
   - Just 5% bias towards goal can improve performance



RRT: Too Much Bias



If q_rand == q_goal all the time:

- Greedily tries to reach goal
- Gets trapped
- Randomness is needed to search the space

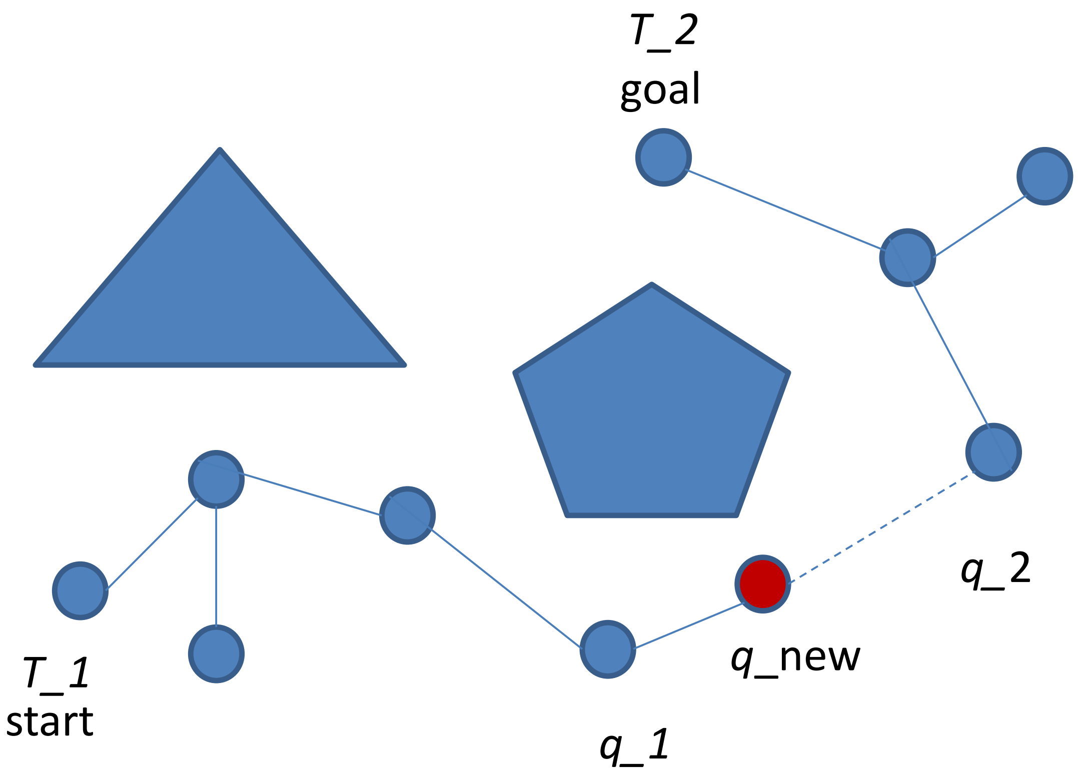
BiDirectional RRT

Use 2 trees (T_1, T_2) one rooted at start, one at goal

To connect the trees (and form a path):

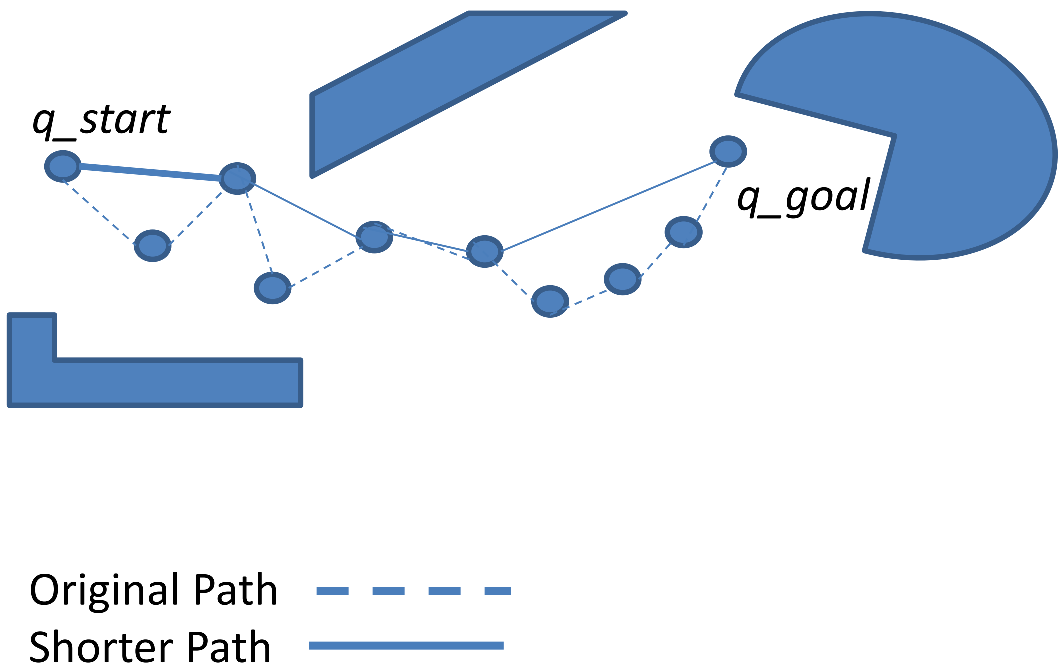
- Expand tree T_1 randomly, add node q_new_1
- Expand T_2 towards q_new_1
  - If tree T_2 connects to q_new_1, path formed, done! else add a q_new_2 for tree T_2
- Now Swap trees T1, T2 and repeat the process

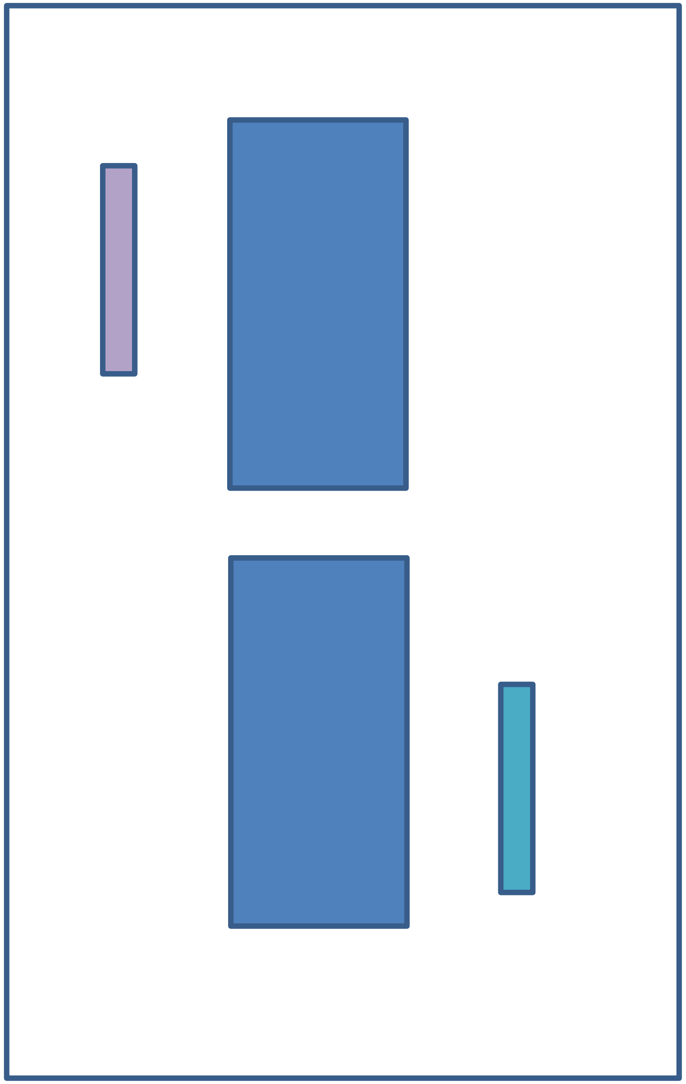
BiDirectional RRT

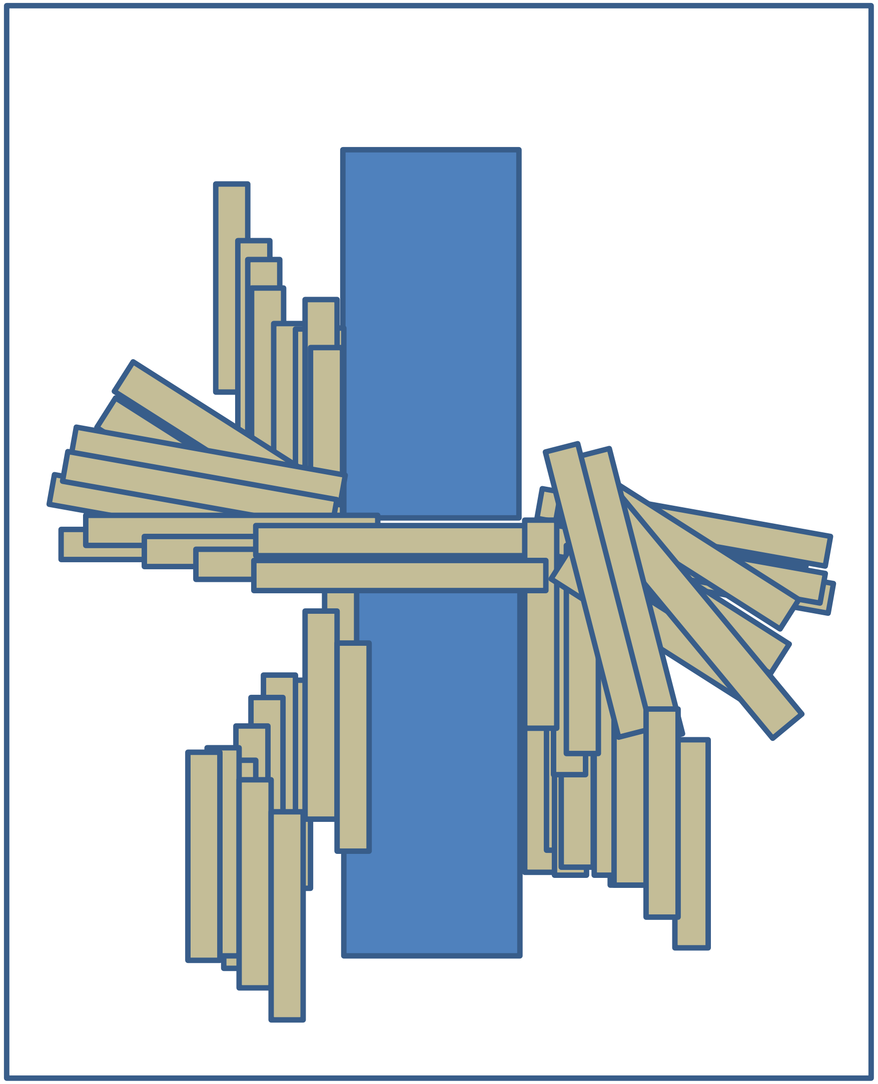


Optimizing Paths

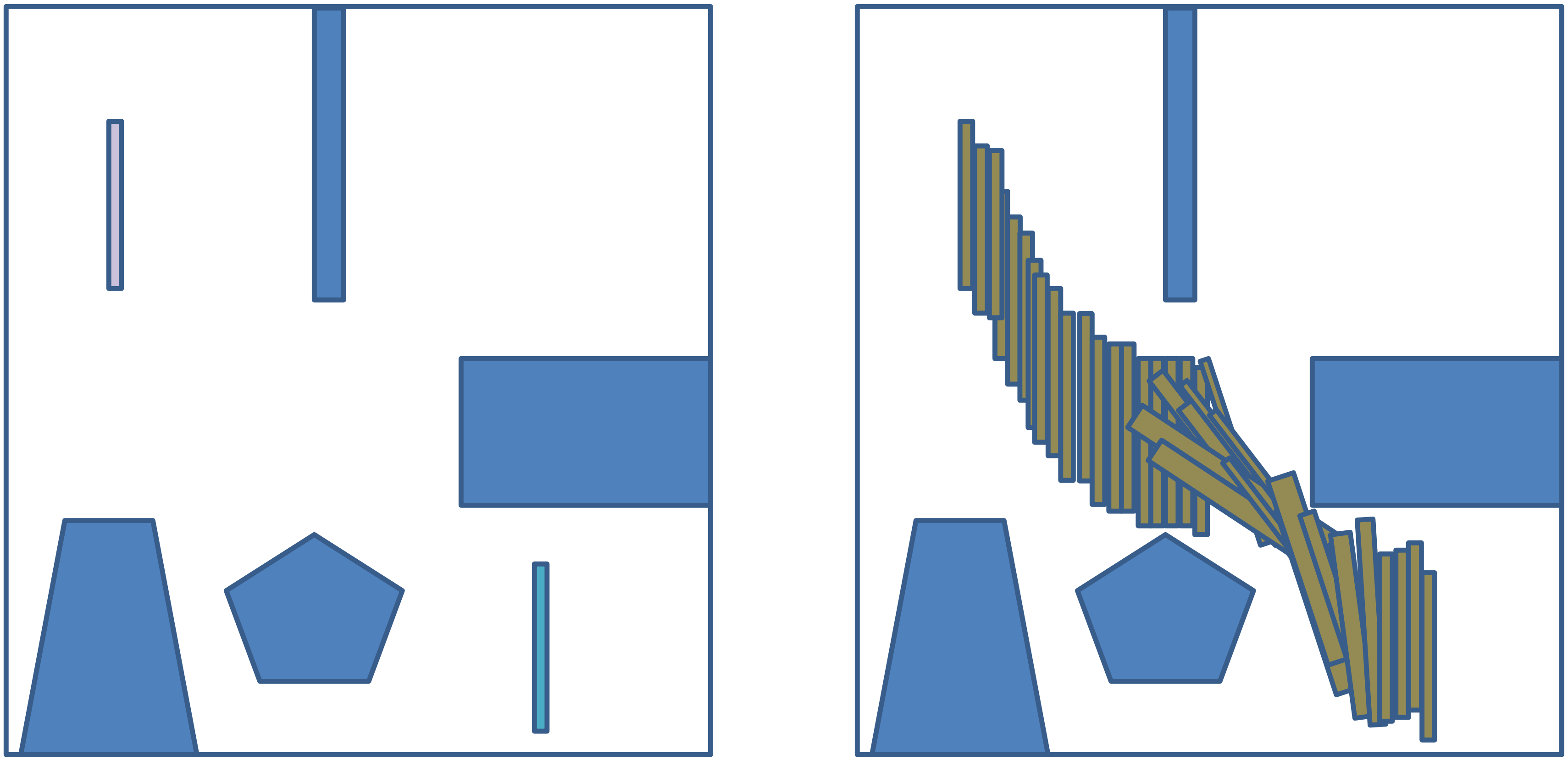
- Try connecting non-adjacent configurations
- Choose q_1 and q_2 randomly, try to connect.
- Greedy approach: try connecting points q_0, q_1, …q_n to q_goal.







Time-lapse paths



RRT Summary

- Efficient way to form goal-directed search without explicit computation of C-Free
- Scales to higher dimensions – multi-DOF robots
- Performance is related to local planner
- step-size is an important parameter
- nearest-neighbor computation can slow performance
- Kinodynamic Planning: Can also include velocity and other constraints in building trees
- Website: http://msl.cs.uiuc.edu/rrt

Path Planning Summary

- Many methods to choose from
- Depends on dimensionality of C-Space, application
- Tradeoffs: computation time, accuracy, optimality, safety
- Most methods are purely kinematic:
  - Plans do not incorporate dynamics
  - A kinematic path for a bi-ped humanoid robot may not be realizable if robot falls or isn't stable
  - Solution: find kinematic paths between KNOWN stable robot configurations
  - Can add dynamics stabilizer to the resulting kinematic path to insure stability
- Paths may not be smooth in Cartesian space – especially true with sampling-based methods