Robot Path Planning

Peter K. Allen
Department of Computer Science
Columbia University

Reading Reference

Principles of Robot Motion

Theory, Algorithms, and Implementations

By

Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki and Sebastian Thrun, MIT Press

https://mitpress.mit.edu/books/principles-robot-motion

Robot Path Planning

Overview:

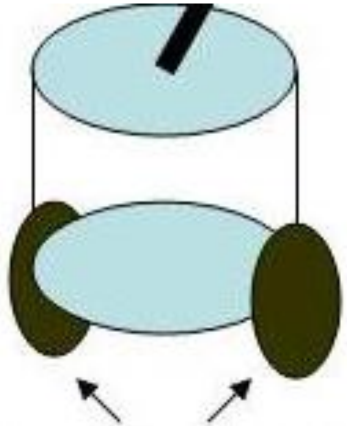
1. Visibility Graphs
2. Voronoi Graphs
3. Potential Fields
4. Sampling-Based Planners
   - PRM: Probabilistic Roadmap Methods
   - RRTs: Rapidly-exploring Random Trees

Robot Path Planning

Things to Consider:

- Spatial reasoning/understanding: robots can have many dimensions in space, obstacles can be complicated
- Global Planning: Do we know the environment *apriori ?*
- Online Local Planning: is environment *dynamic?* Unknown or moving obstacles? Can we compute path "on-the fly"?
- Besides collision-free, should a path be optimal in time, energy or safety?
- Computing exact "safe" paths is provably computationally expensive in 3D – "piano movers" problem
- Kinematic, dynamic, and temporal reasoning may also be required

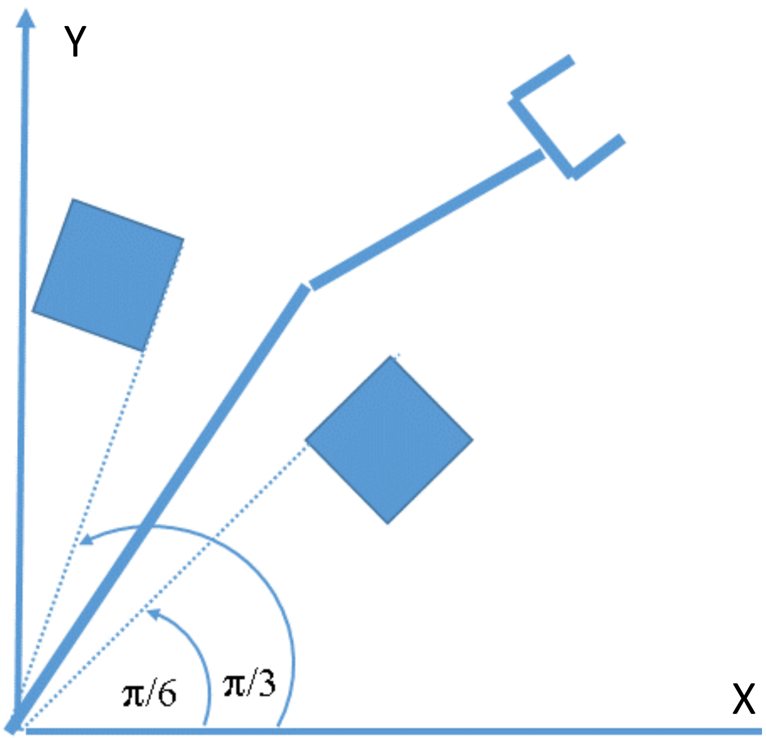
Configuration Space of a Robot

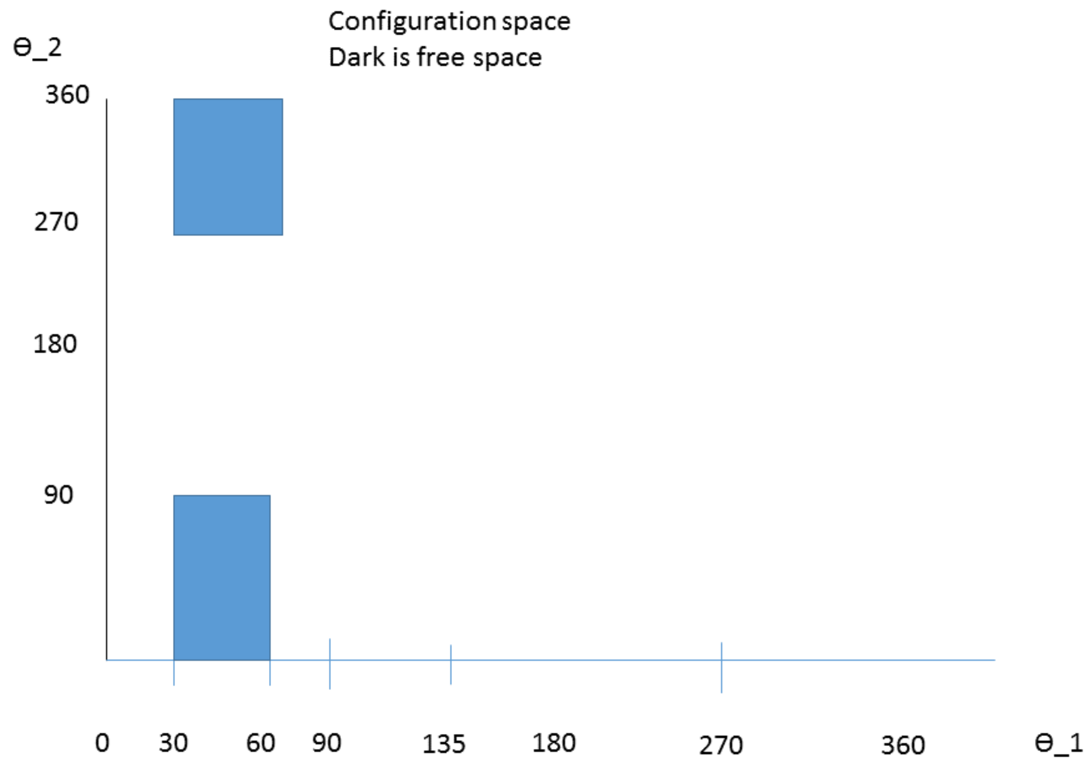


Mobile Base with 2 wheel differential drive

- Configuration Space (C-Space) : Set of parameters that completely describes the robots' state
- Mobile base has 3 Degrees-of-Freedom (DOFs)
- It can translate in the the plane (X,Y) and rotate (Θ)
- C-Space is allowable values of (X,Y,Θ)

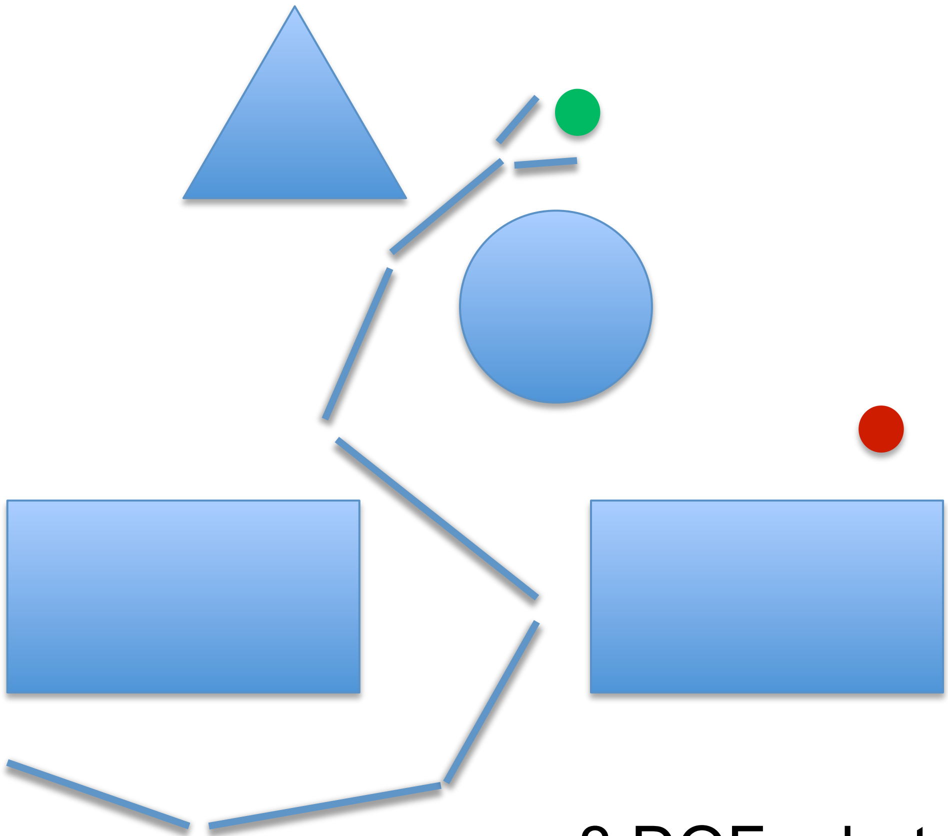
Configuration Space: C-Space





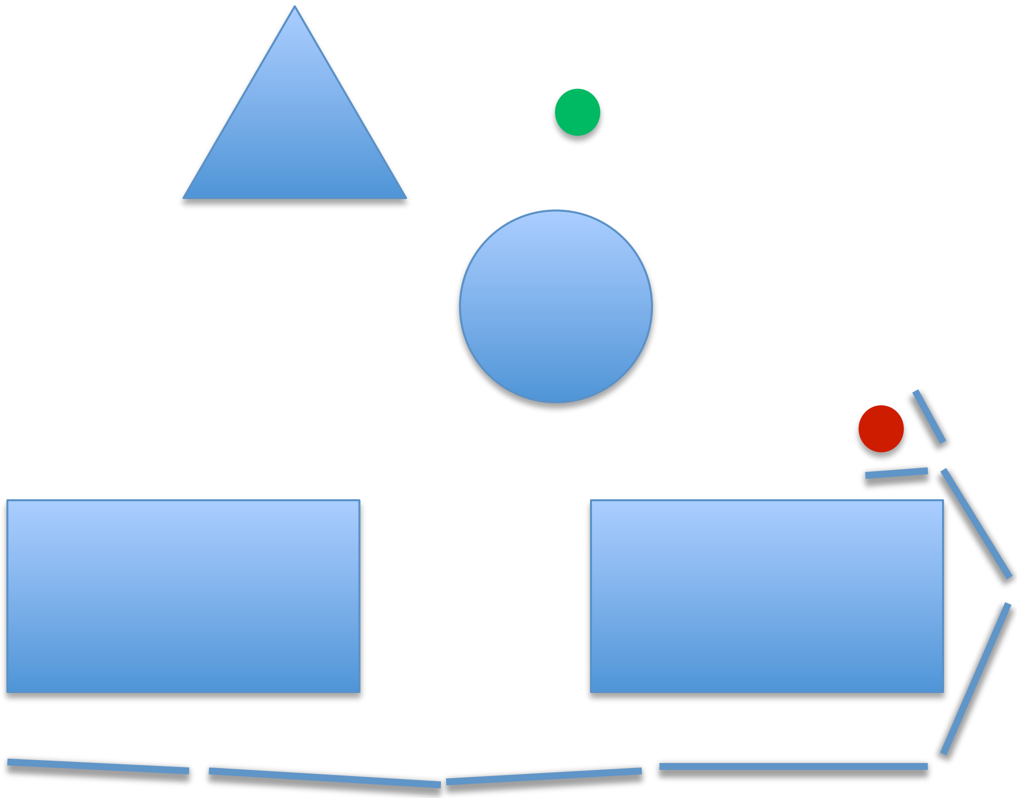
- 2-DOF robot: joints ϴ_1, ϴ_2 are the robot's C-Space
- C-Free: values of ϴ_1, ϴ_2 where robot is NOT in collision
- C-Free = C-Space – C-Obstacles

Path Planning in Higher Dimensions



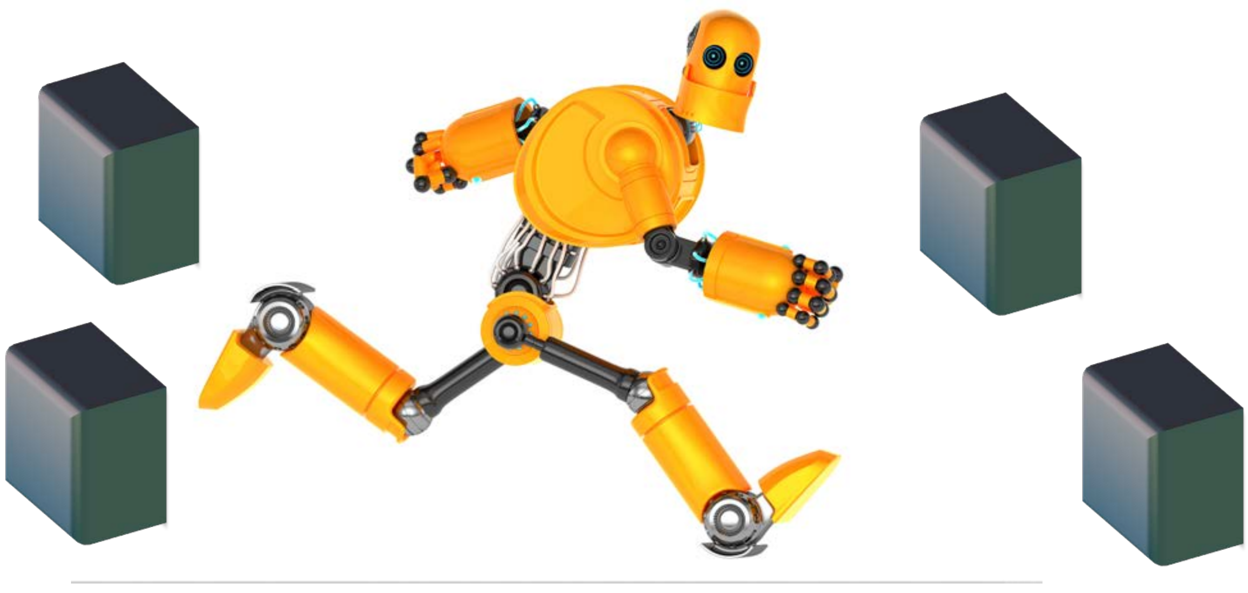
- 8 DOF robot arm
- Plan collision free path to pick up red ball

Path Planning in Higher Dimensions



- 8 DOF robot arm
- Plan collision free path to pick up red ball

Path Planning in Higher Dimensions



- Humanoid robot has MANY DOFs
- Anthropomorphic Humanoid: Typically >20 joints:
  - 2-6 DOF arms, 2-4 DOF legs, 3 DOF head, 4 DOF torso, plus up to 20 DOF per multi-fingered hand!
- Exact geometric/spatial reasoning difficult
- Complex, cluttered environments also add difficulty

Robot Path Planning

Overview:

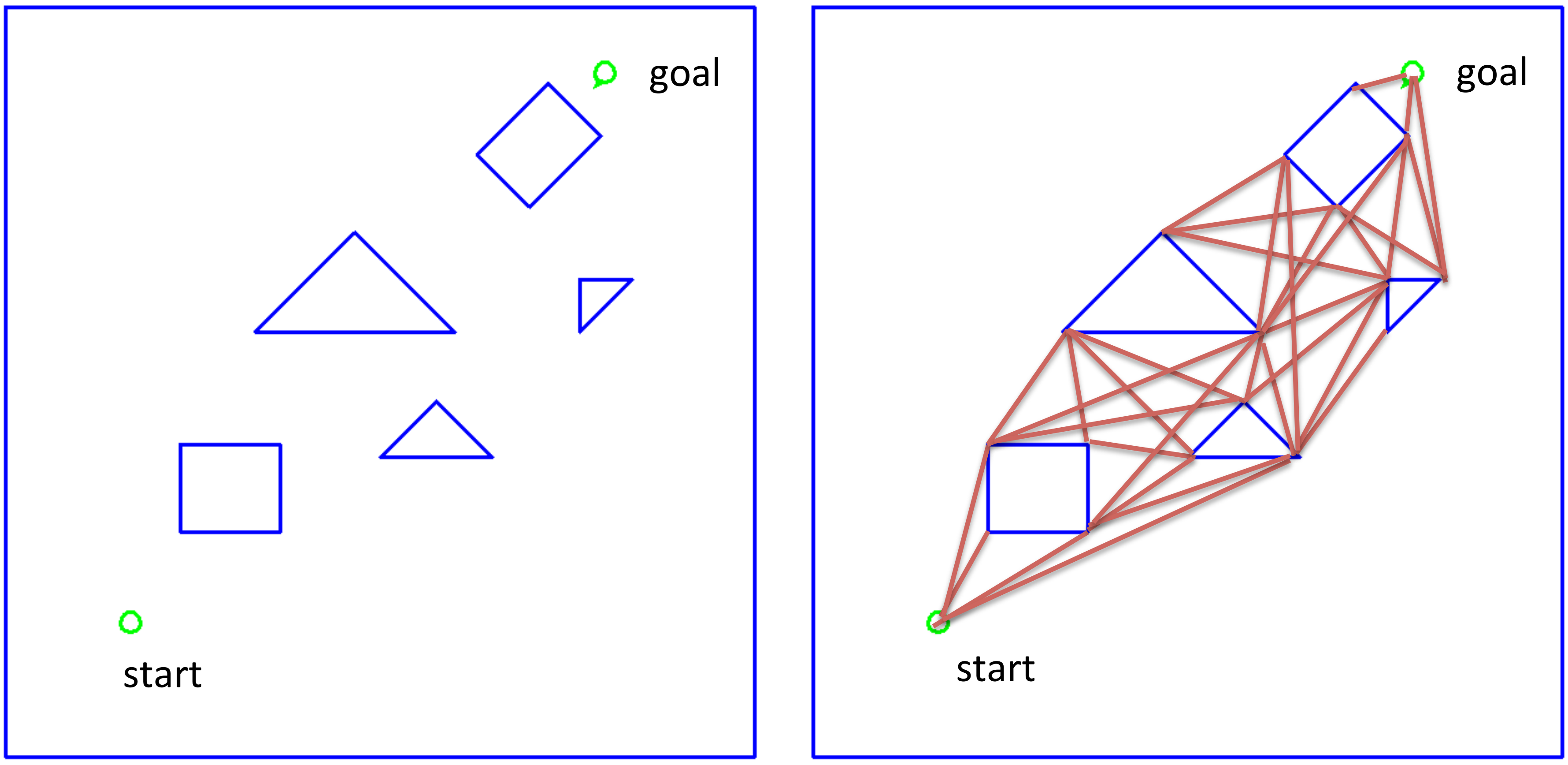
1. Visibility Graphs
2. Voronoi Graphs
3. Potential Fields
4. Sampling-Based Planners
   - PRM: Probabilistic Roadmap Methods
   - RRTs: Rapidly-exploring Random Trees

Visibility Graph

How does a Mobile Robot get from A to B?

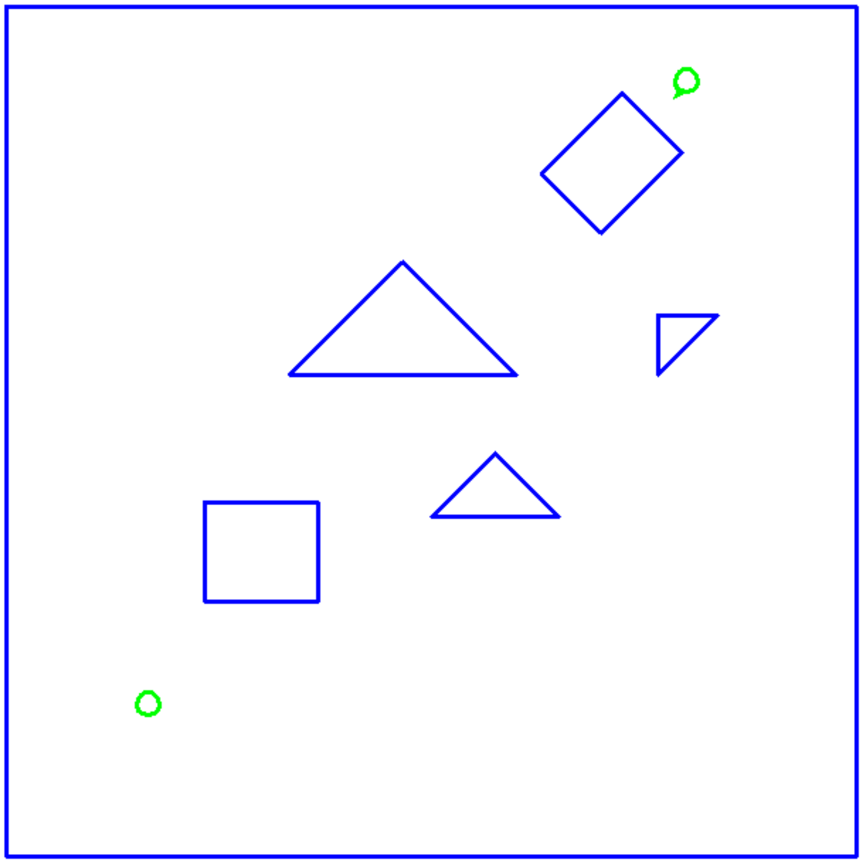
- Assume robot is a point in 2-D planar space
- Assume obstacles are 2-D polygons
- Create a Visibility Graph:
  - Nodes are start point, goal point, vertices of obstacles
  - Connect all nodes which are “visible” – straight line un-obstructed path between any 2 nodes
  - Includes all edges of polygonal obstacles
- Use A* to search for path from start to goal

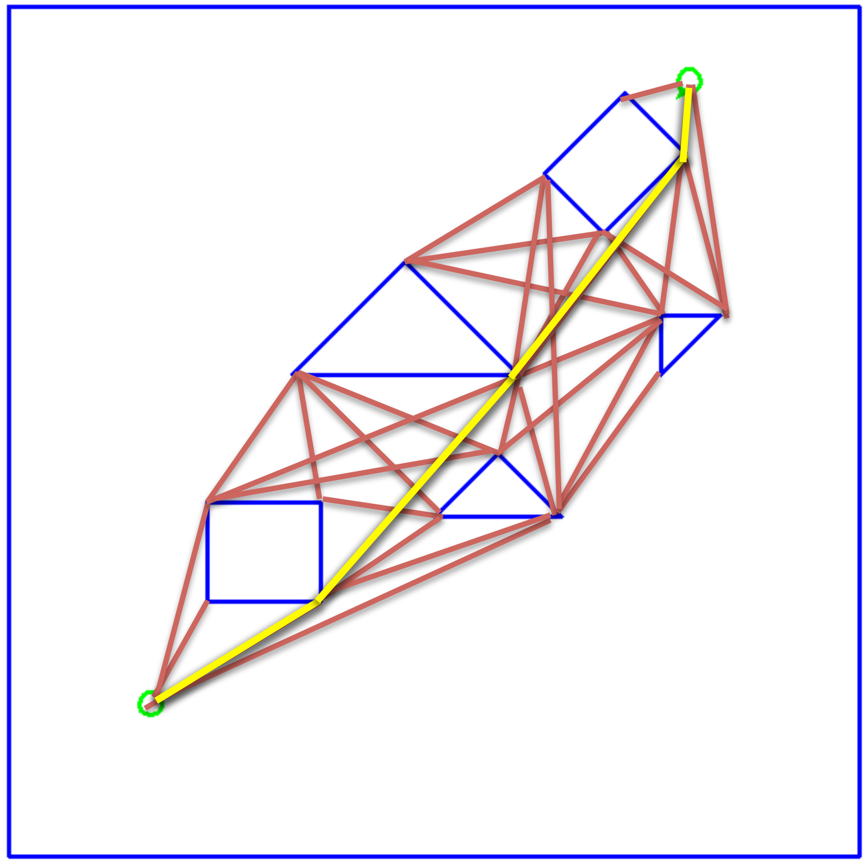
Visibility Graph - VGRAPH



- Start, goal, vertices of obstacles are graph nodes
- Edges are “visible” connections between nodes, including obstacle edges

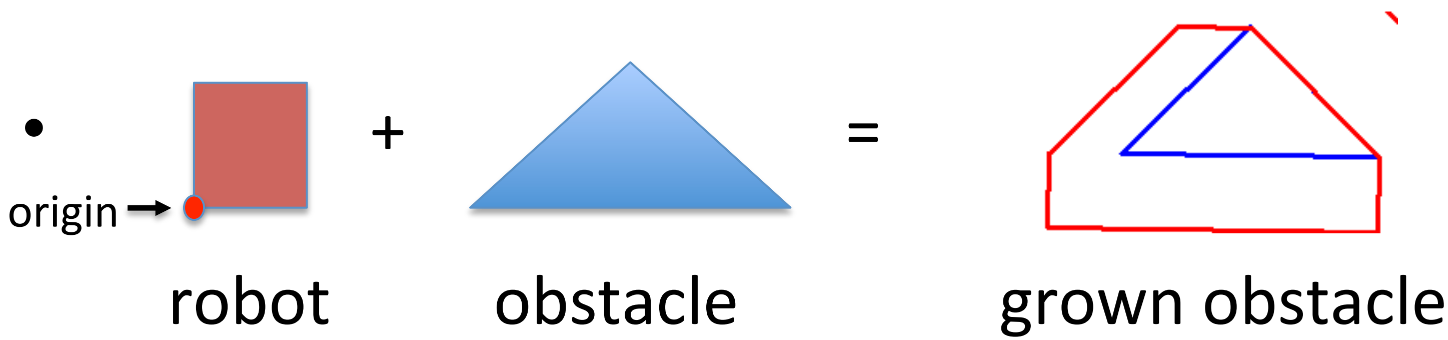
Visibility Graph - VGRAPH





- A* search for shortest path via visible vertices

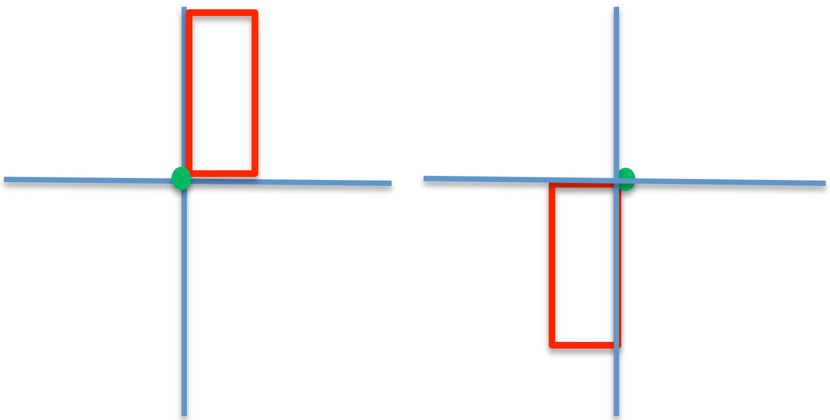
VGRAPH: Grown Obstacles

- VGRAPH algorithm assumes point robot
- What if robot has mass, size?
- Solution: expand each obstacle by size of the robot – create Grown Obstacle Set
- 

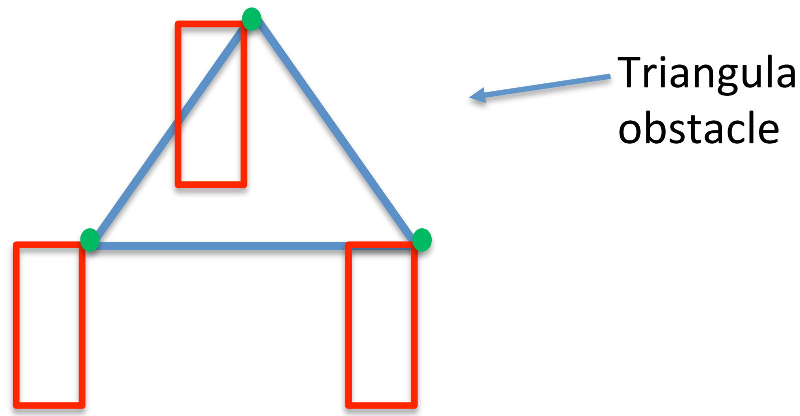
- This effectively “shrinks” the robot back to a point
- Graph search of the VGRAPH will now find shortest path if one exists using grown obstacle set

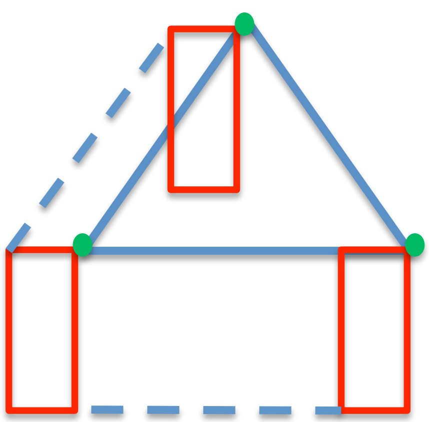
VGRAPH: Growing Obstacles

Reflect robot about X, Y axes

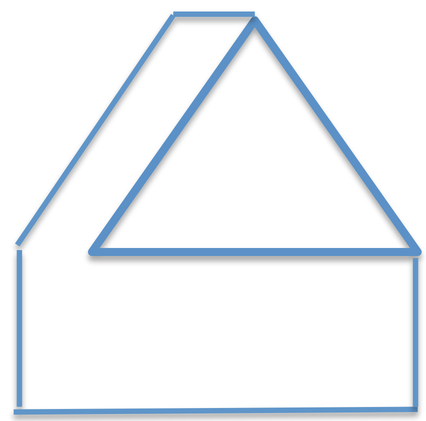


Add reflected robot vertices to each obstacle vertex



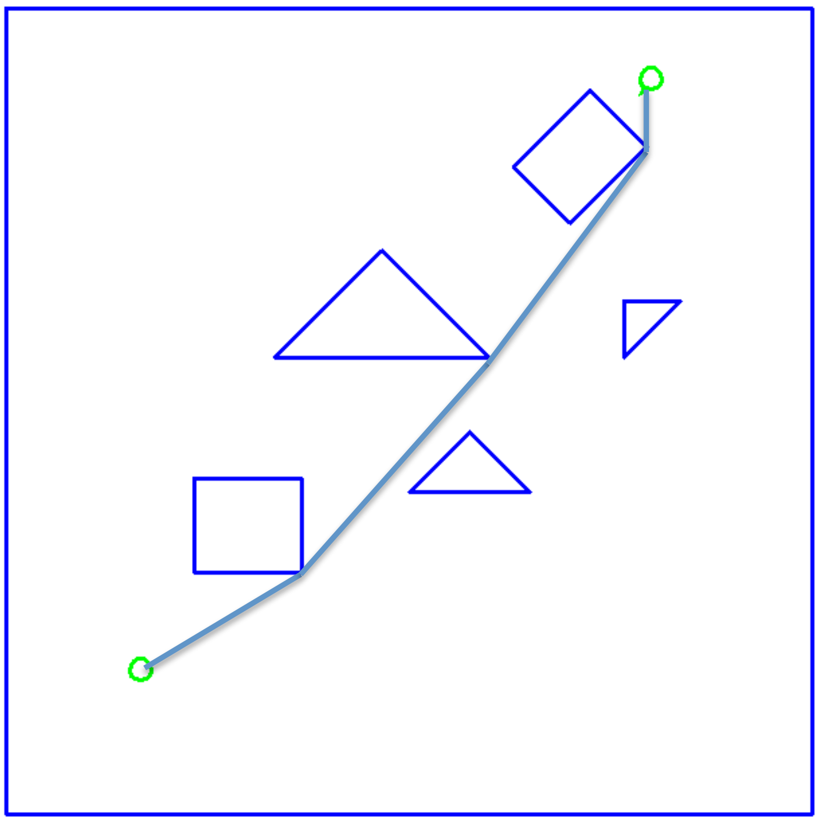


Compute convex hull of vertices

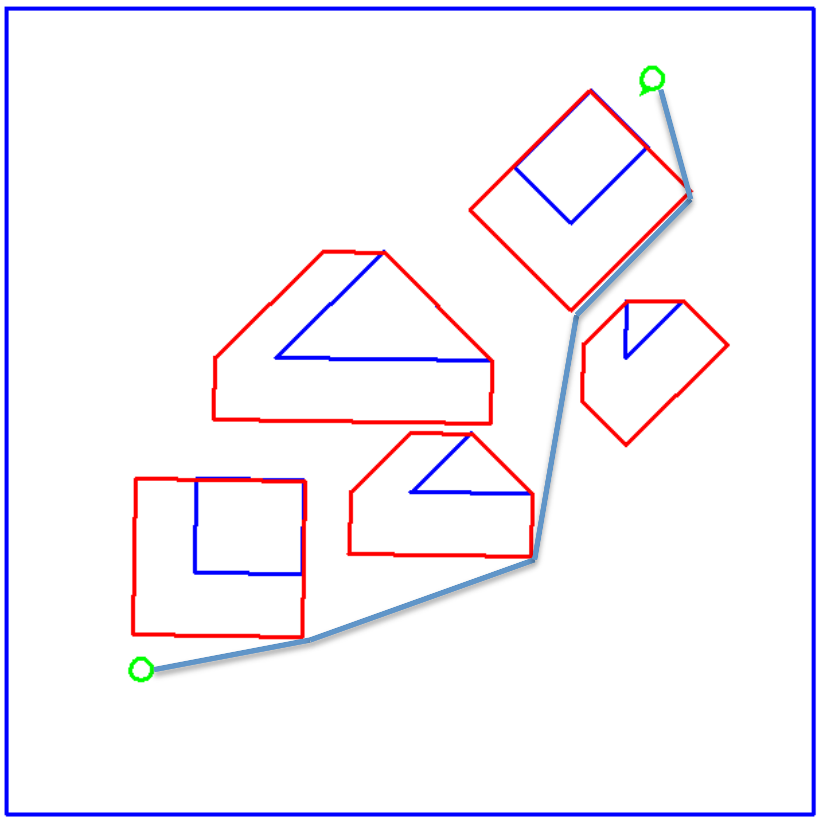


Convex hull is grown obstacle

VGRAPH: Grown Obstacles



Point Robot Path

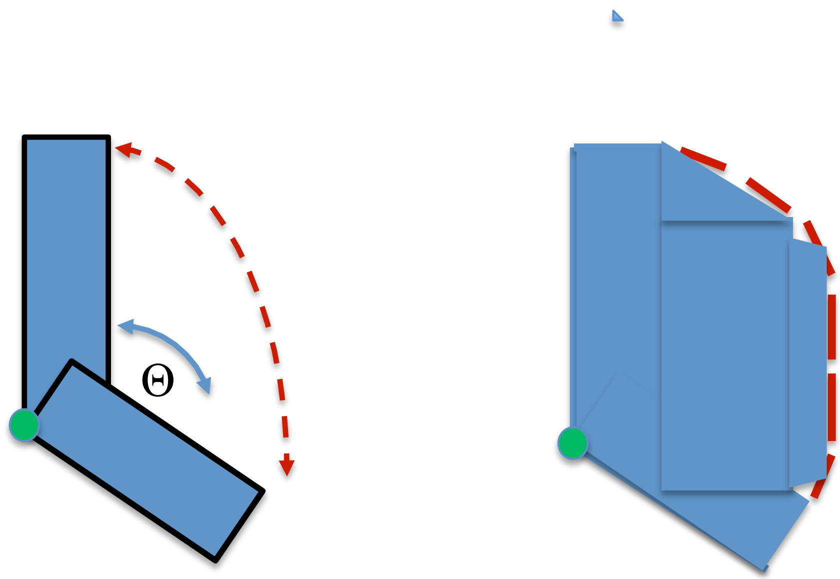


Path after growing obstacles with square robot

VGRAPH Extensions

- Rotation: Mobile Robot can rotate
- Solution:
  - Grow obstacles by size that includes all rotations
  - Over-conservative. Some paths will be missed
  - Create multiple VGRAPHS for different rotations
  - Find regions in graphs where rotation is safe, then move from one VGRAPH mapping to another
- Non-convex obstacles/robots: any concave polygon can be modeled as set of convex polygons

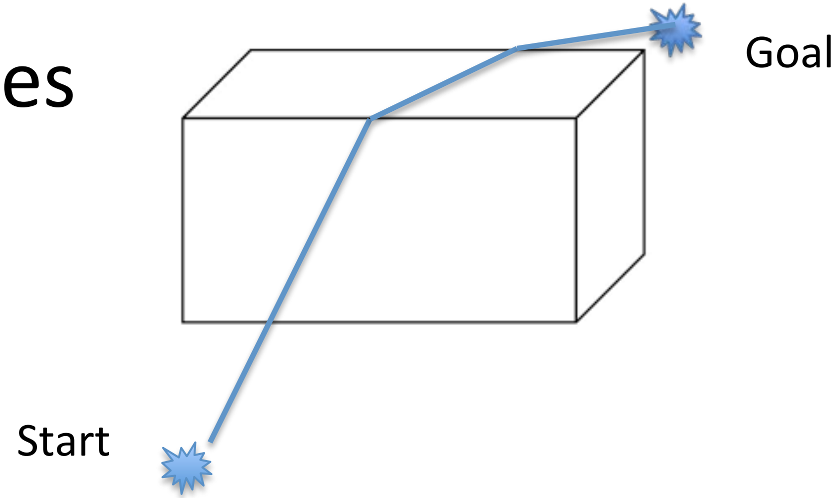
VGRAPH: Rotations



Left: Rectangular robot that can rotate
Right: Polygon that approximates all rotations
Polygon is over-conservative, will miss legal paths

VGRAPH Summary

- Guaranteed to give shortest path in 2D
- Path is dangerously close to obstacles – no room for error
- Does not scale well to 3D. Shortest path in 3D is not via vertices:
- Growing obstacles
  is difficult in 3D



Robot Path Planning

Overview:

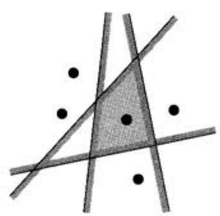
1. Visibility Graphs
2. Voronoi Graphs
3. Potential Fields
4. Sampling-Based Planners
   - PRM: Probabilistic Roadmap Methods
   - RRTs: Rapidly-exploring Random Trees

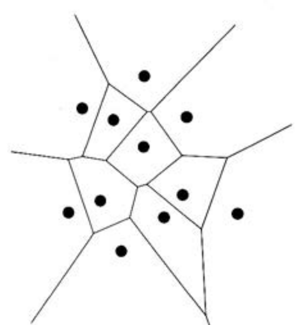
Voronoi Path Planning

- Find paths that are not close to obstacles, but in fact as far away as possible from obstacles.

- This will create a maximal safe path, in that we never come closer to obstacles than we need.

- Voronoi Diagram in the plane. Let P = {p_i}, set of points in the plane, called *sites.* Voronoi diagram is the sub-division of the plane into N distinct cells, one for each *site*.

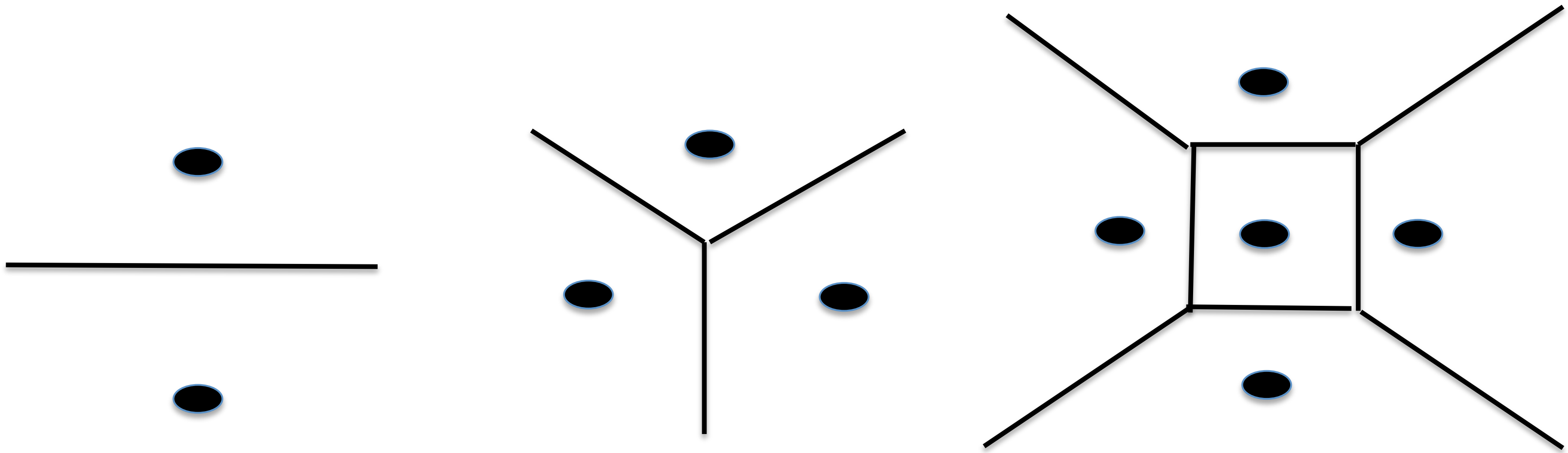




- Cell has property that a point q corresponds to a site p_i iff:

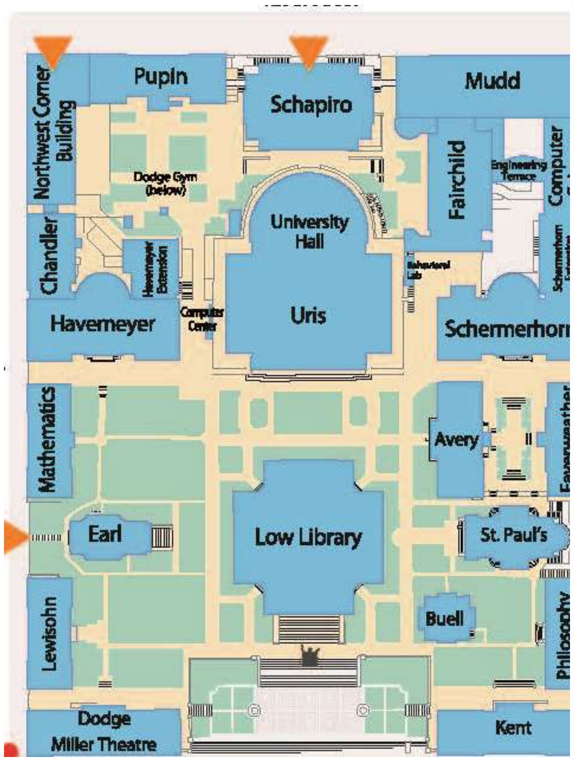
  $\text{dist}(q, p_i) < \text{dist}(q, p_j)$ for all $p_j \in P$, $j \neq i$

Voronoi Graph



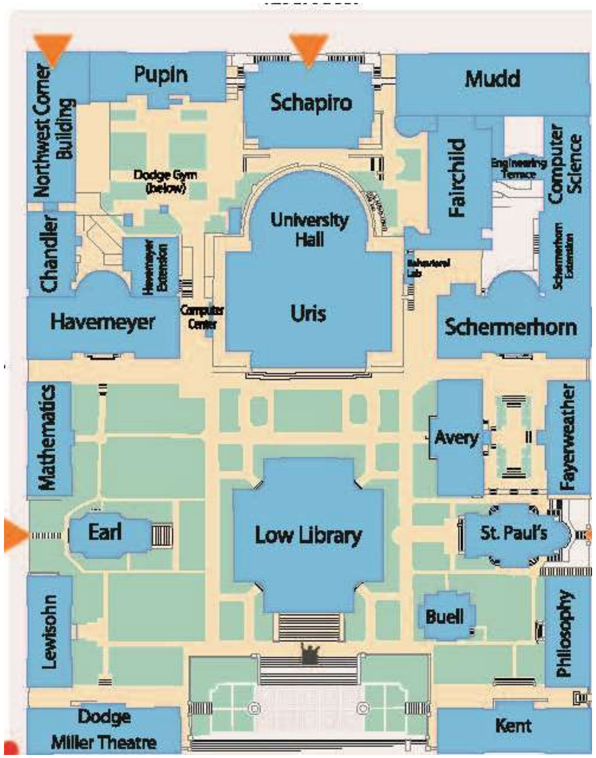
- Intuitively: Edges and vertices are intersections of perpendicular bi-sectors of point-pairs
- Edges are equidistant from 2 points
- Vertices are equidistant from 3 points
- Online demo: http://alexbeutel.com/webgl/voronoi.html

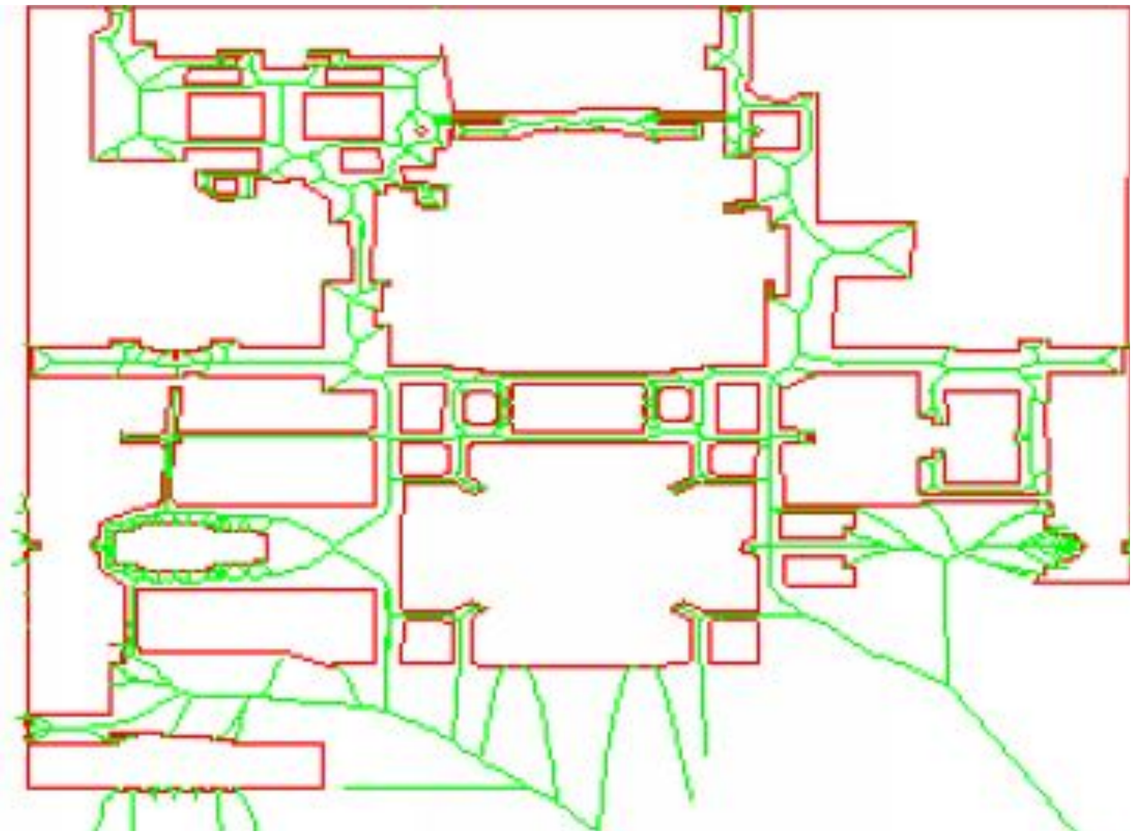
Voronoi Path on Columbia Campus



To find a path:

Voronoi Path on Columbia Campus

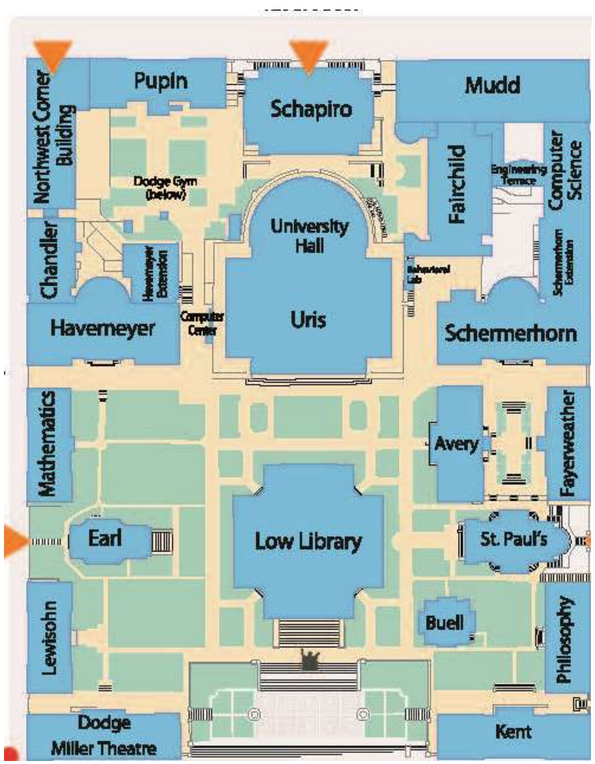


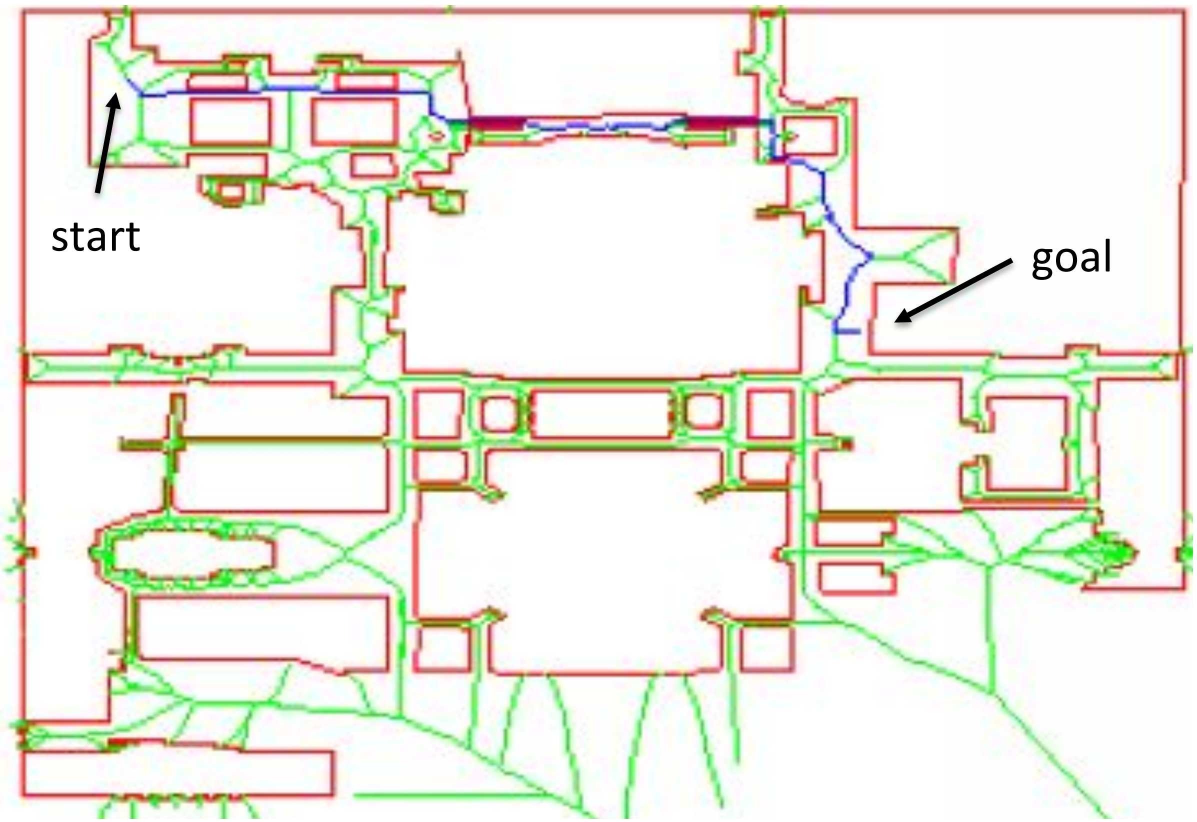


To find a path:

- Create Voronoi graph - O(N log N) complexity in the plane
- Connect q_start, q goal to graph – local search
- Compute shortest path from q_start to q_goal (A* search)

Voronoi Path on Columbia Campus





To find a path:

- Create Voronoi graph - O(N log N) complexity in the plane
- Connect q_start, q_goal to graph – local search
- Compute shortest path from q_start to q_goal using A*

Voronoi Graph Summary

- Difficult to compute in higher dimensions, 3-D obstacles add complexity
- Paths can be overly conservative – don’t always have to be far away from obstacles.
- Voronoi is fragile – small environment change can significantly change the graph

Robot Path Planning

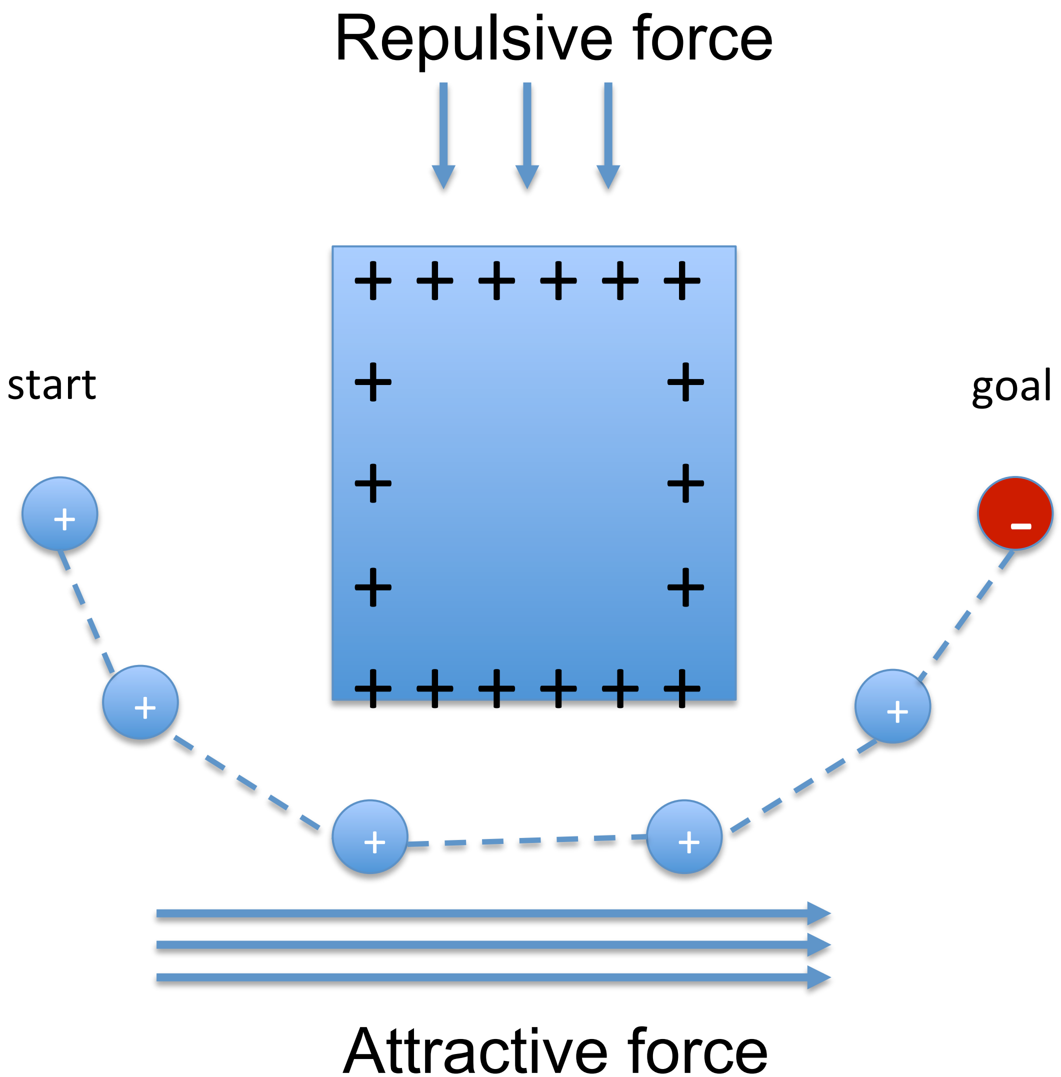
Overview:

1. Visibility Graphs
2. Voronoi Graphs
3. Potential Fields
4. Sampling-Based Planners
   - PRM: Probabilistic Roadmap Methods
   - RRTs: Rapidly-exploring Random Trees

Potential Field Path Planning

- Simple idea: Have robot “attracted” to the goal and “repelled” from the obstacles
- Think of robot as a positively charged particle moving towards negatively charged goal – attractive force
- Obstacles have same charge as robot – repelling force
- States far away from goal have large potential energy, goal state has zero potential energy
- Path of robot is from state of high energy to low (zero) energy at the goal
- Think of the planning space as an elevated surface, and the robot is a marble rolling “downhill” towards the goal

Potential Field Path Planning



Potential Field Path Planning

Attractive Energy: Distance to goal
Highly attractive farther away
Goes to zero at goal

$$F_{att}(q) = d^2(q, q_{goal})$$

Repelling Energy:
Inverse of distance to obstacles
Goes to zero as we move away

$$F_{rep}(q) = \frac{1}{d^2(q, Obstacles)}$$

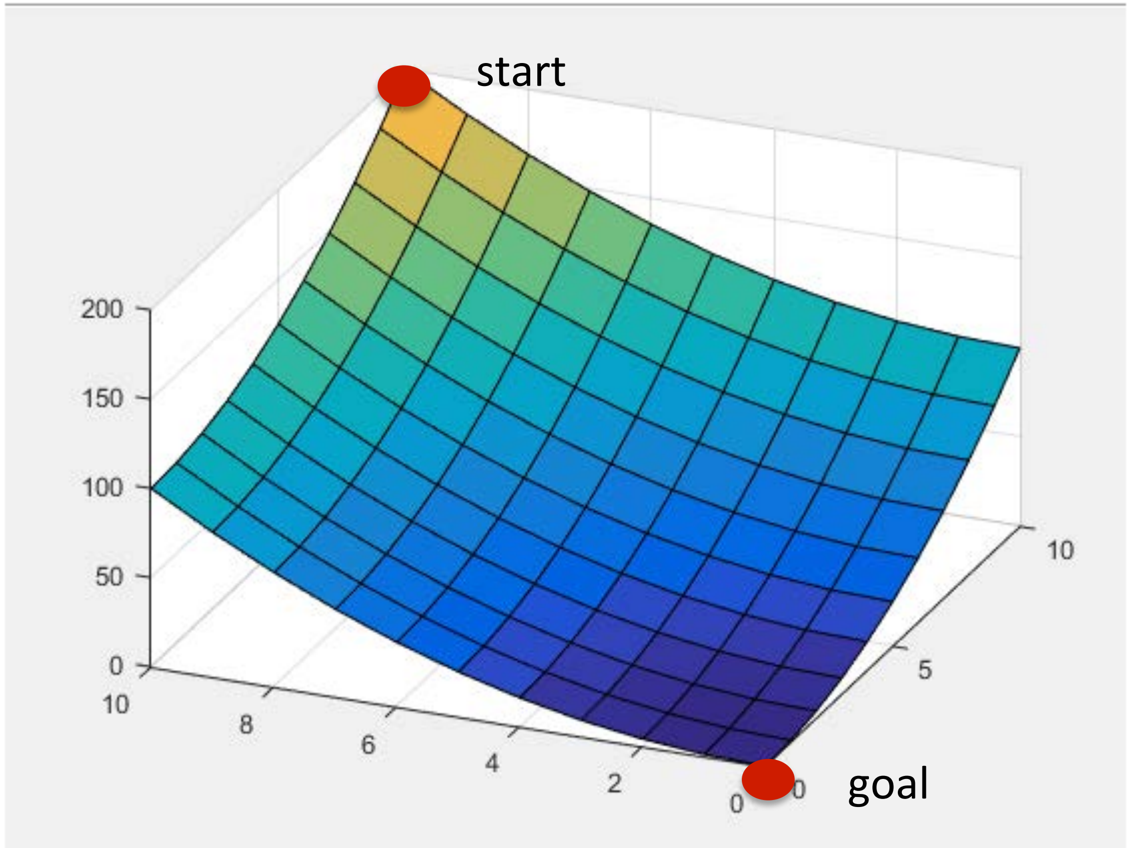
Potential Function:
Sum of energy acting on robot
α weights + and - forces

$$F(q) = F_{att}(q) + \alpha F_{rep}(q)$$

Robot moves along
negative gradient of F(q)

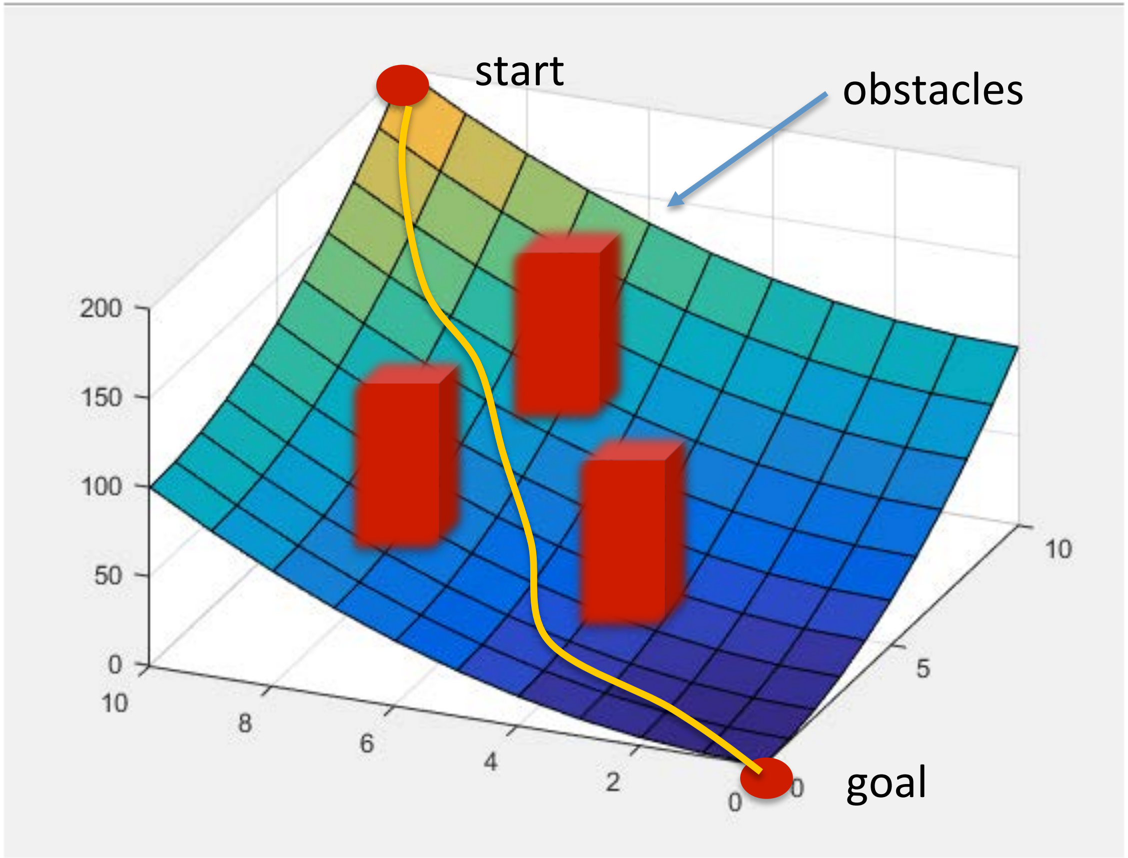
$$-\nabla F(q)$$

Potential Field



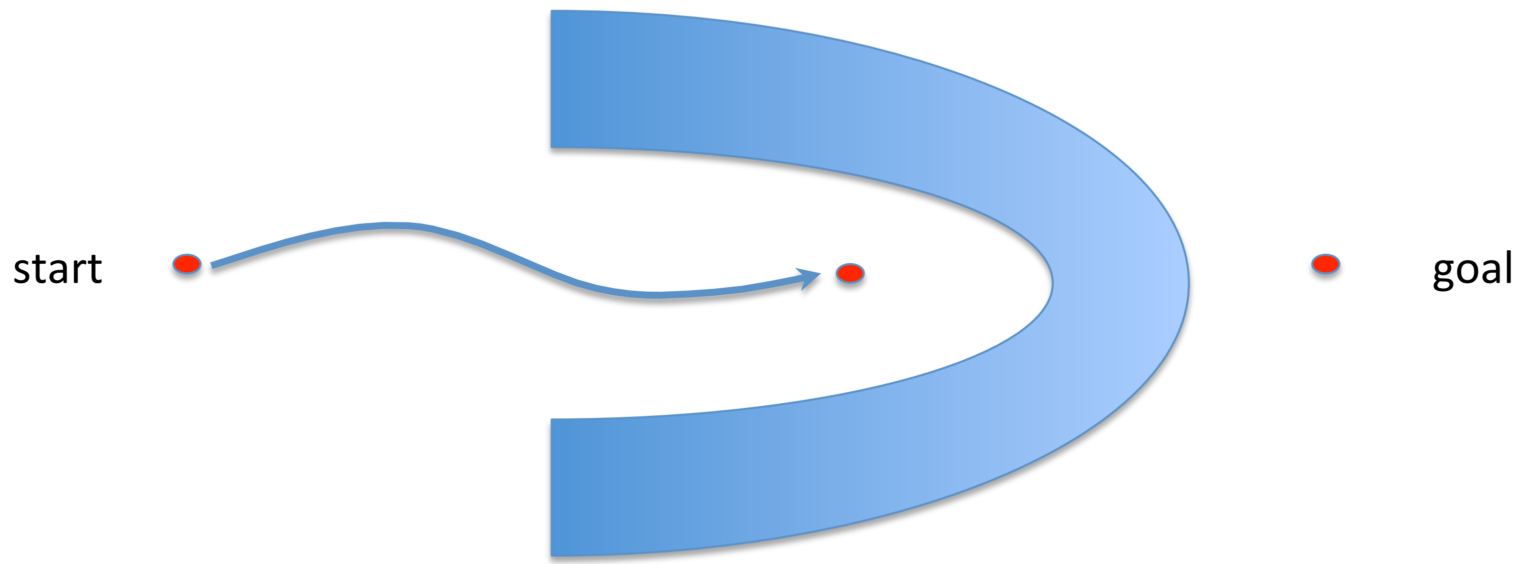
- Attractive Potential Function is distance from goal
- High energy away from goal, Zero at goal
- Path is negative gradient, largest change in energy

Potential Field



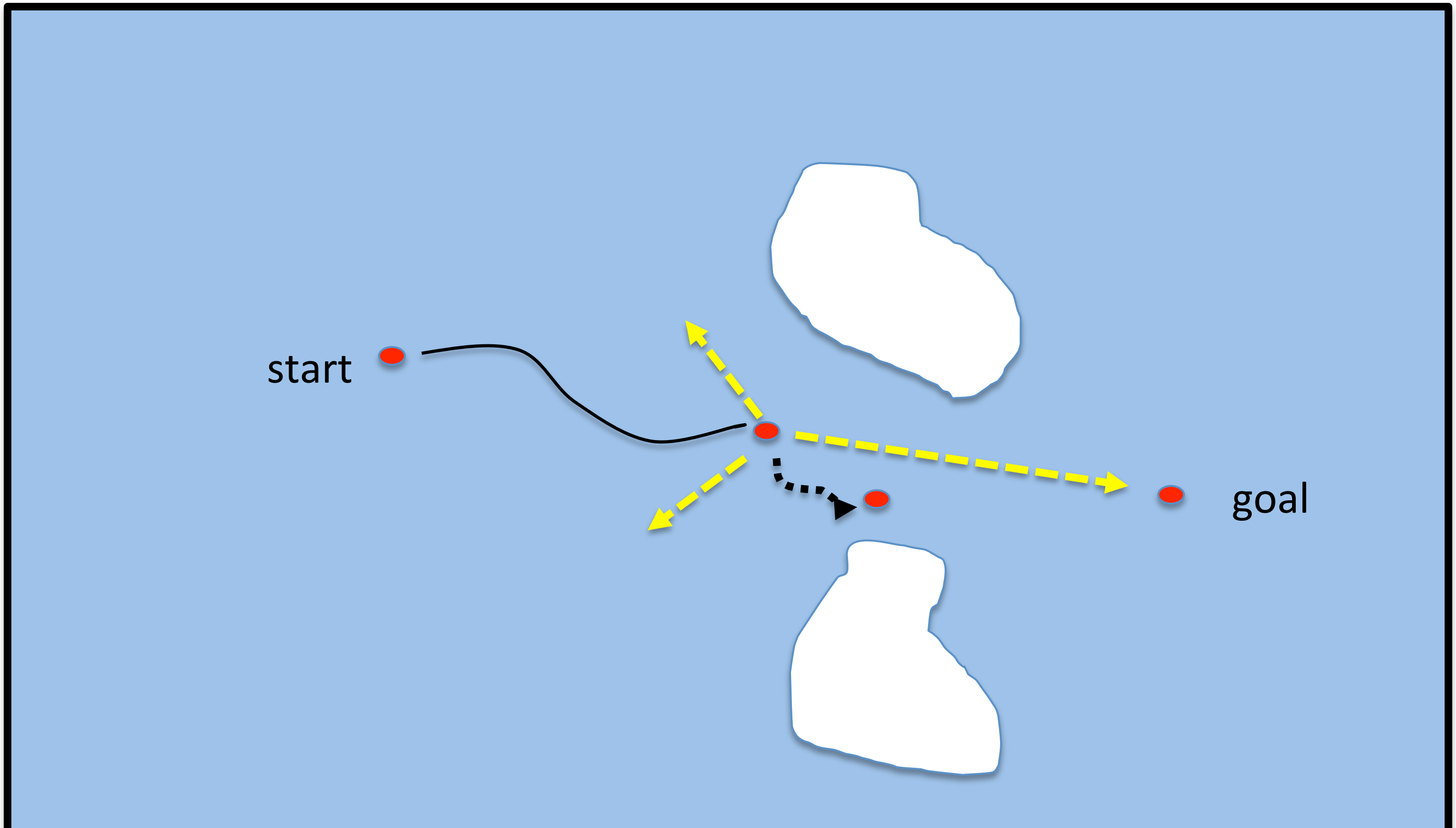
- Obstacles create high energy barriers
- Gradient descent follows energy minimization path to goal

Potential Field Limitations



Local minimum:
attractive force (goal) = repulsive force (obstacles)

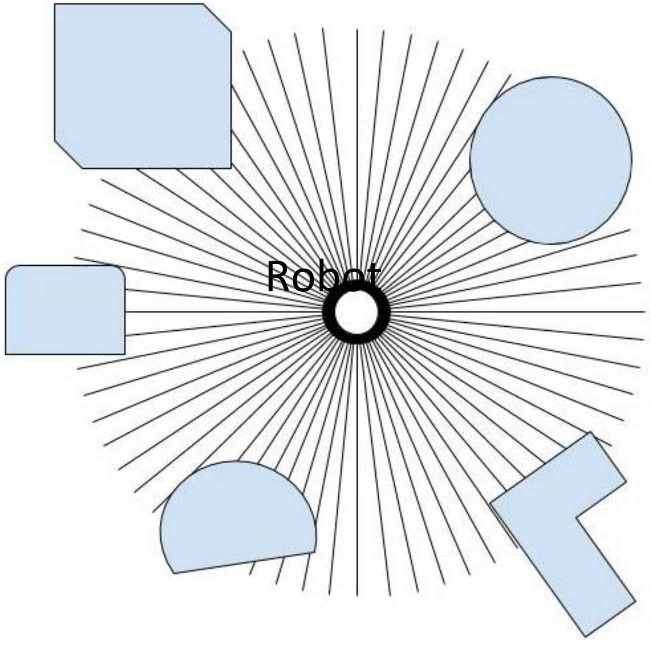
Potential Field Methods



Local minimum: attractive force = repulsive force
Solution: Take a random walk – perturb out of minima
Need to remember where you have been!

Potential Fields Summary

- More than just a path planner: Provides simple control function to move robot: gradient descent
- Allows robot to move from wherever it finds itself
- Can get trapped in local minima
- Can be used as online, local method:
  - As robot encounters new obstacles \
    compute the Potential Function onlir
  - Laser/sonar scans give online
    distance to obstacles



Robot Path Planning

Overview:

1. Visibility Graphs
2. Voronoi Graphs
3. Potential Fields
4. 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).

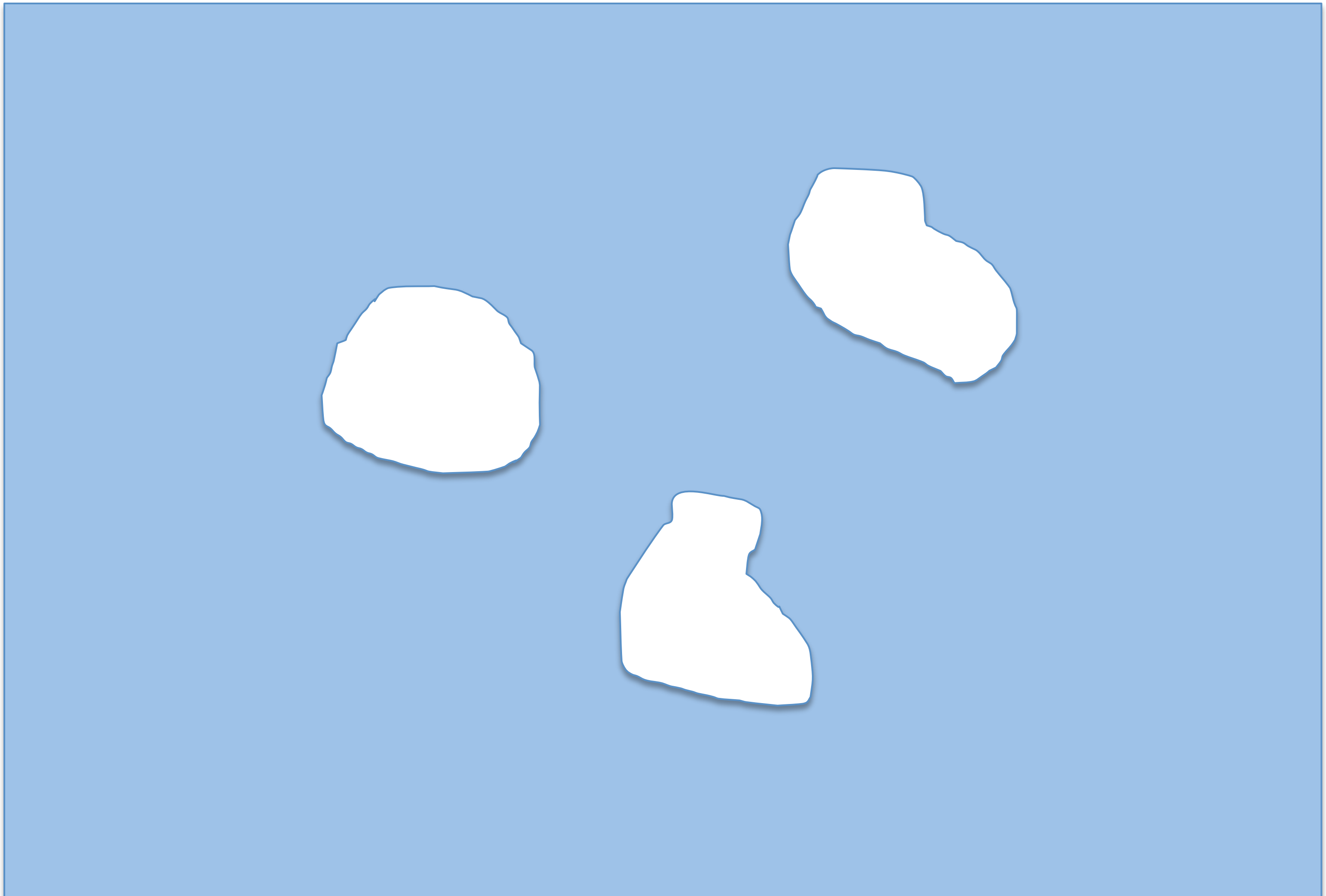
Probabilistic Roadmap Planner - PRM

- 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

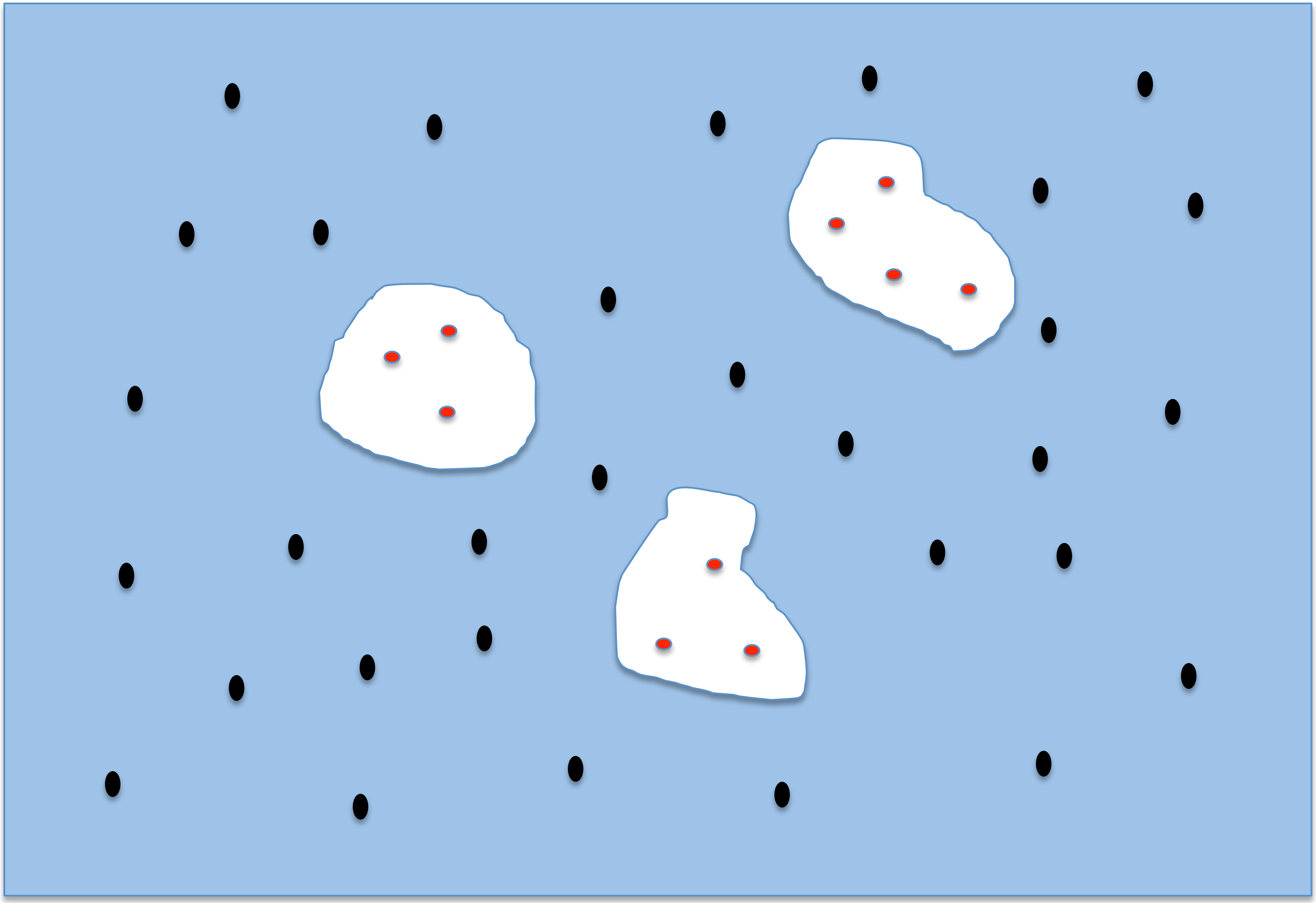
- Initially empty roadmap Graph G
- A robot configuration q is randomly chosen
- If q→Q-Free (collision free configuration) then add to G
- Repeat until N vertices chosen
- For each vertex q, select k nearest neighbors
- Local planner tries to connect q to neighbor q'
- If connect successful (i.e. collision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

PRM



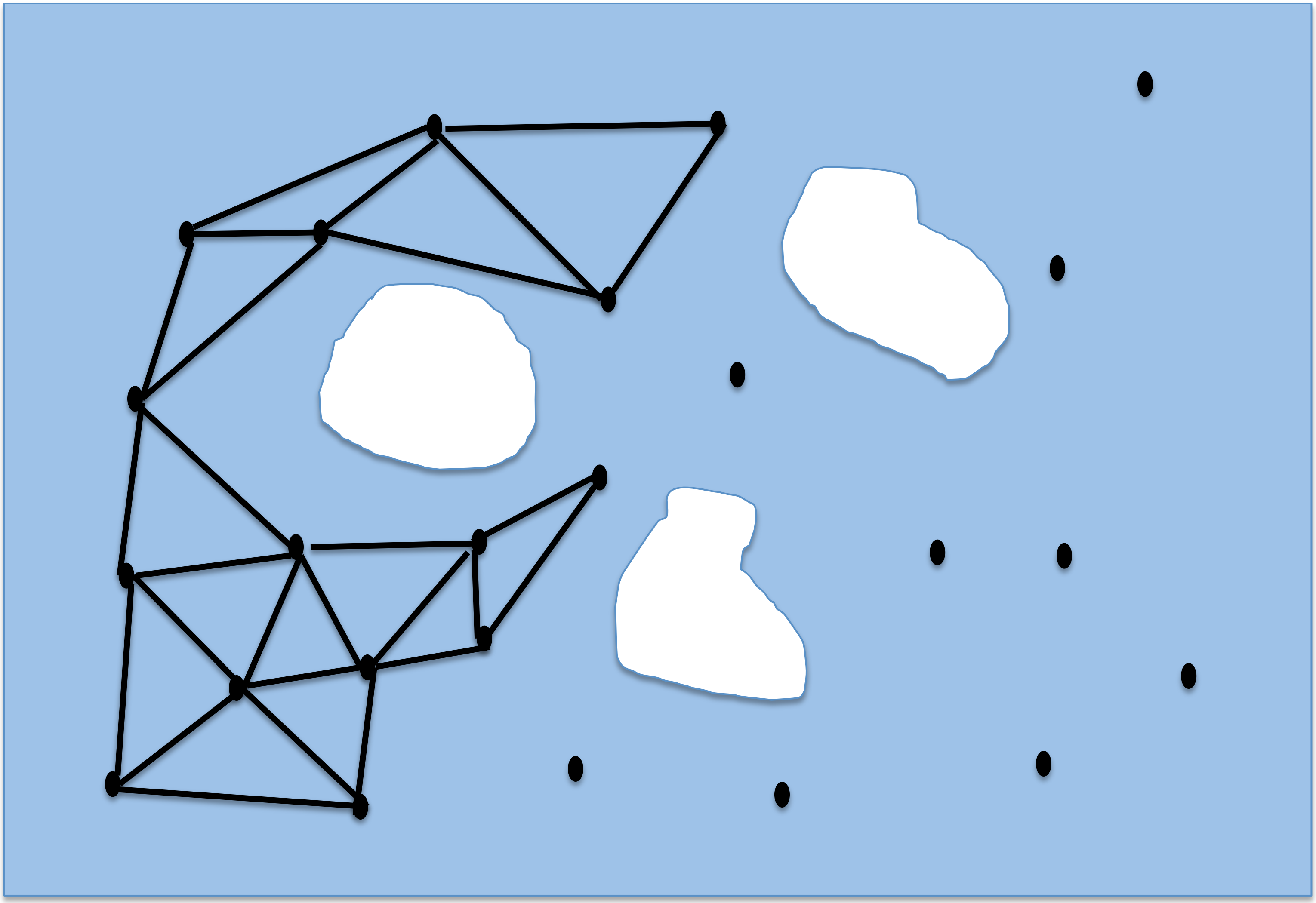
2D planar environment with obstacles

PRM



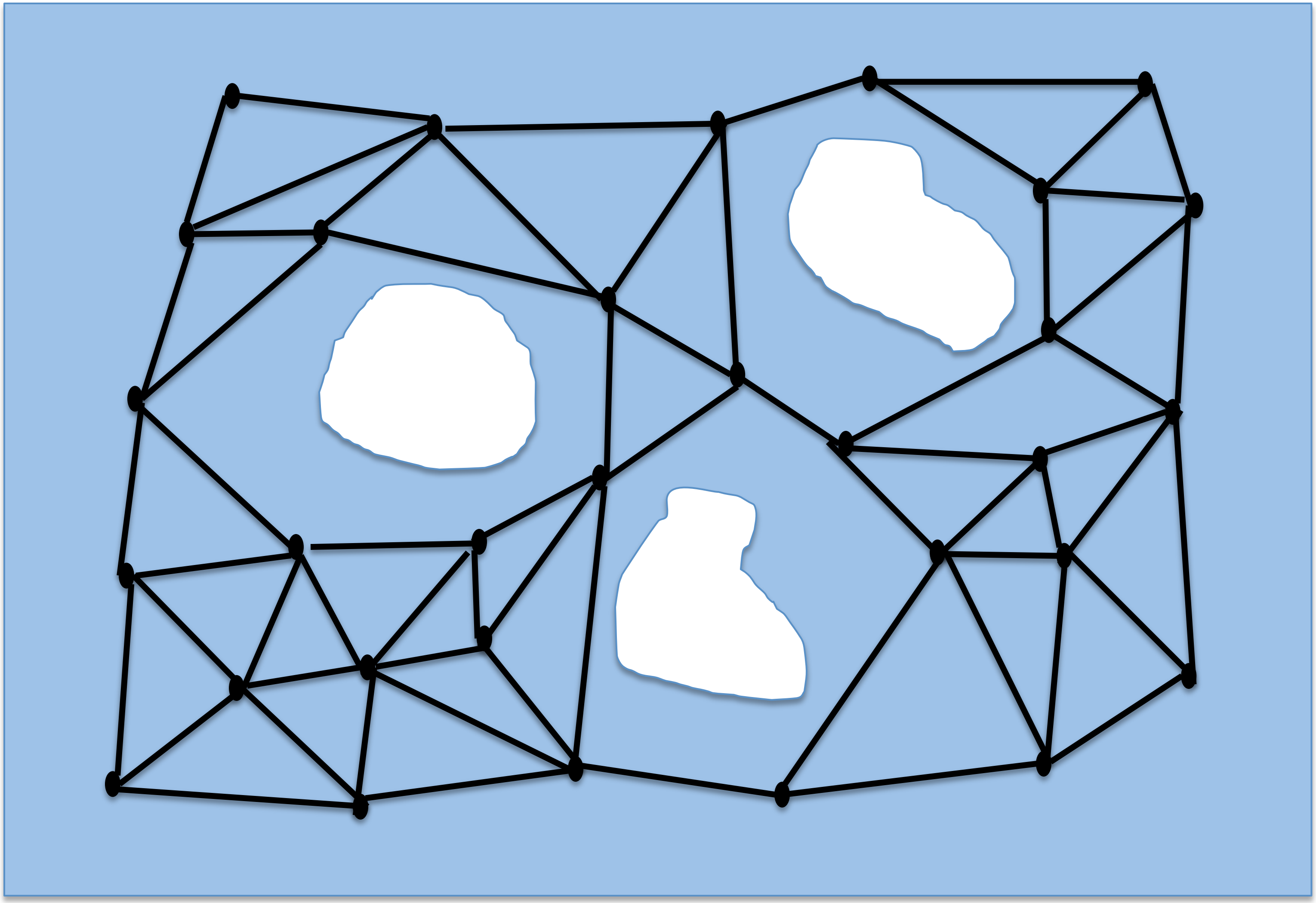
1. Randomly sample C-Space for N collision-free configurations

PRM



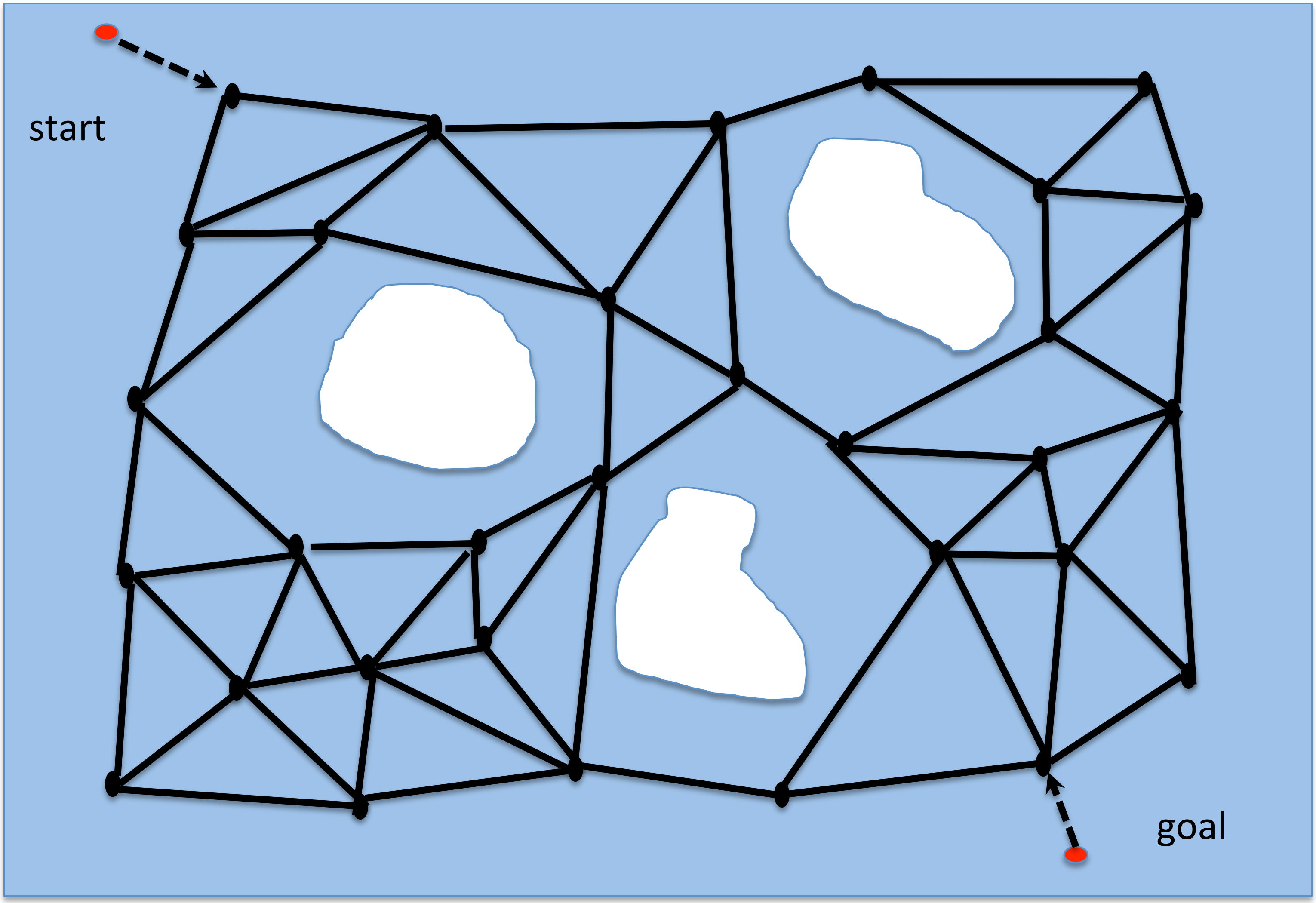
2. Link each vertex in Q-Free with K nearest neighbors

PRM



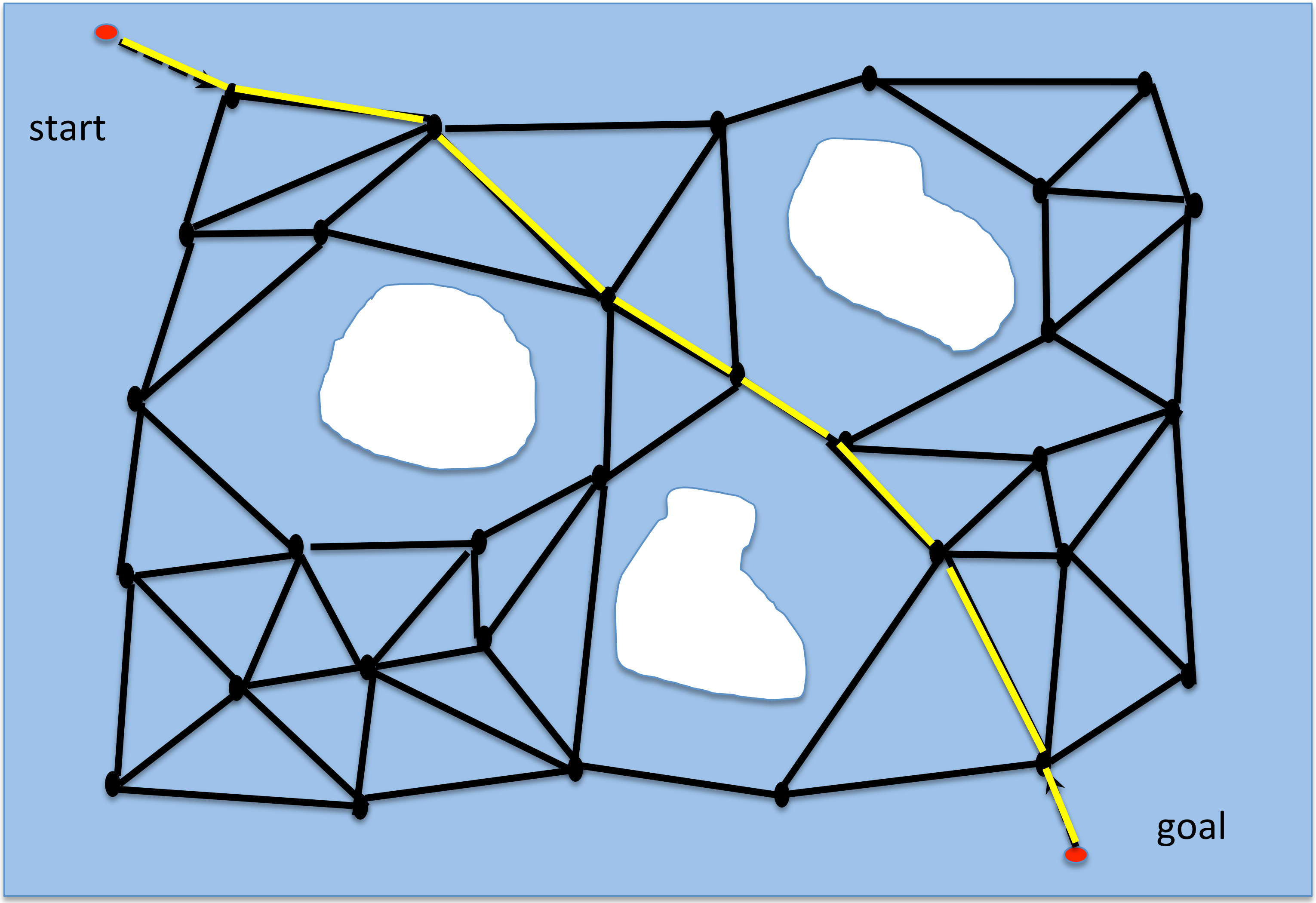
2. Link each vertex in Q-Free with K nearest neighbors

PRM



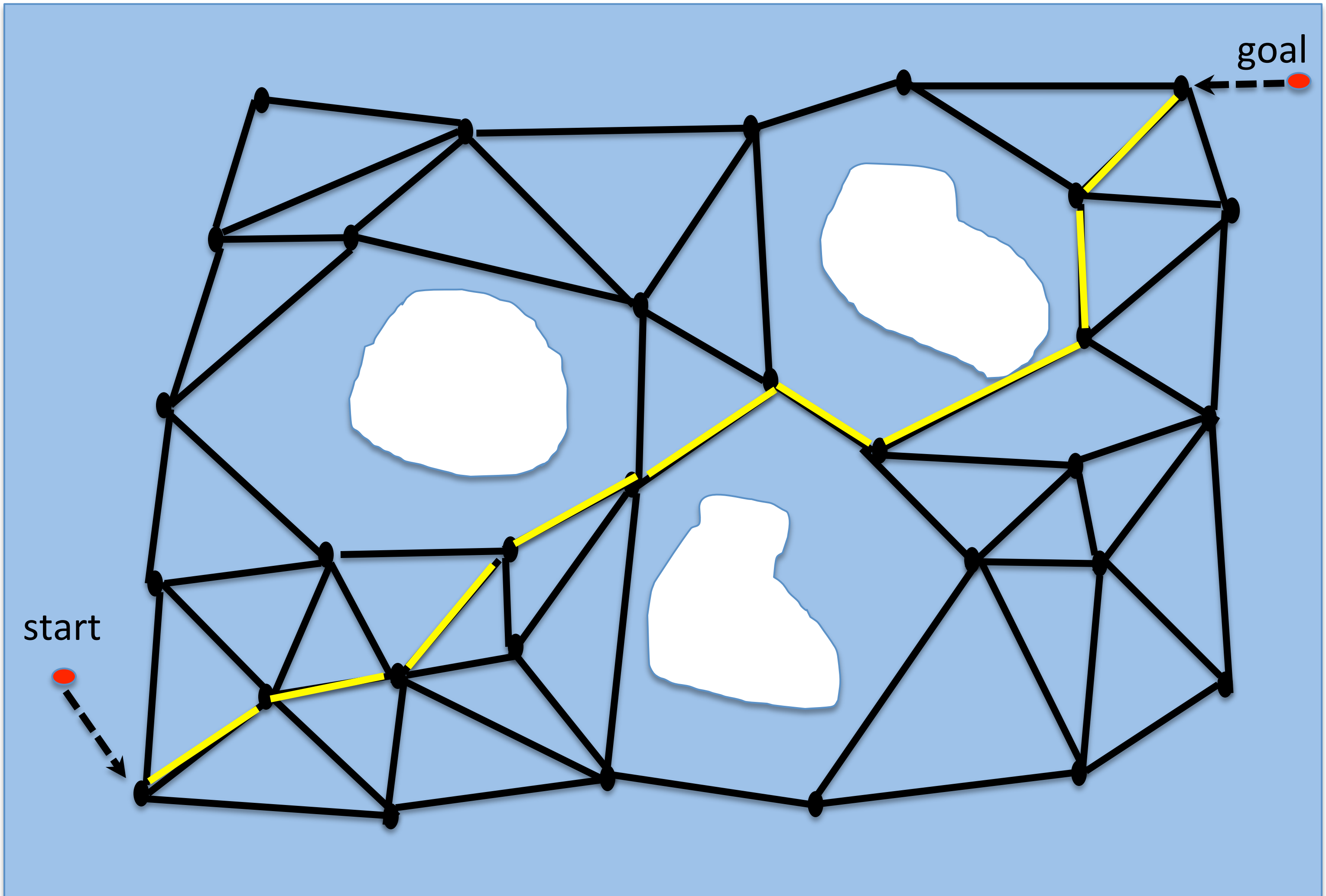
3. Connect start and goal to nearest node in roadmap

PRM



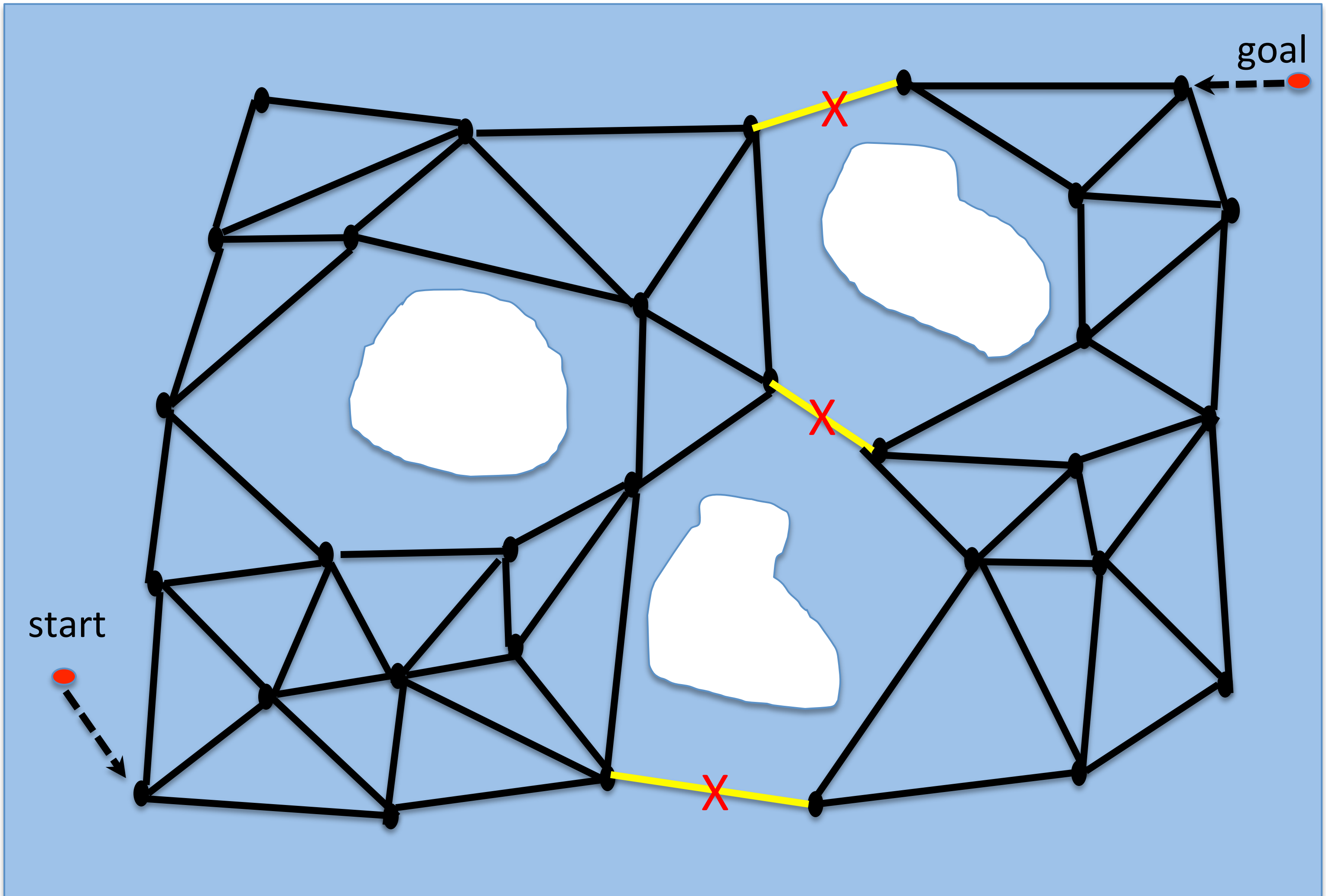
4. Graph Search for shortest path

PRM



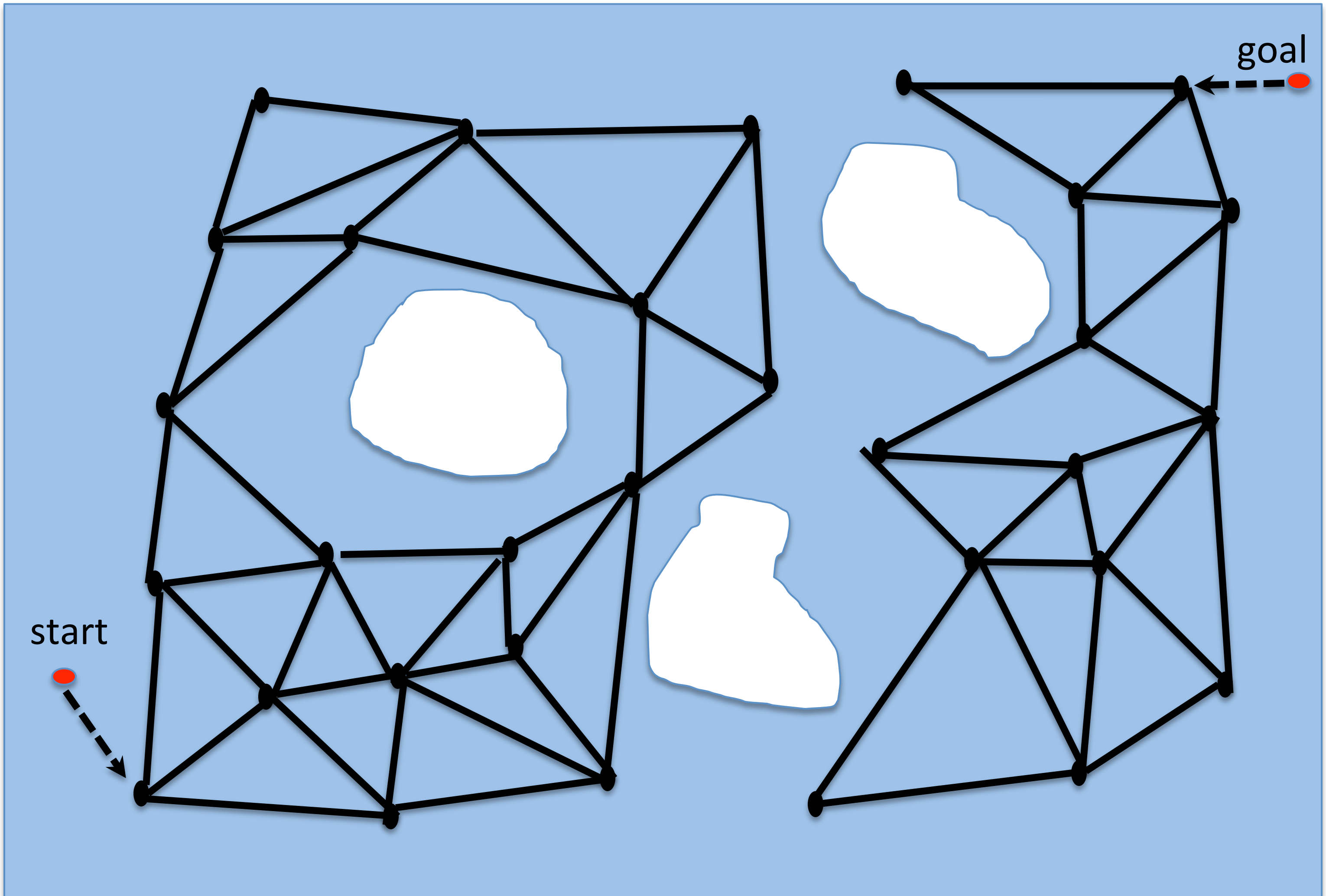
Handles multiple queries-once on roadmap, finds a path

PRM



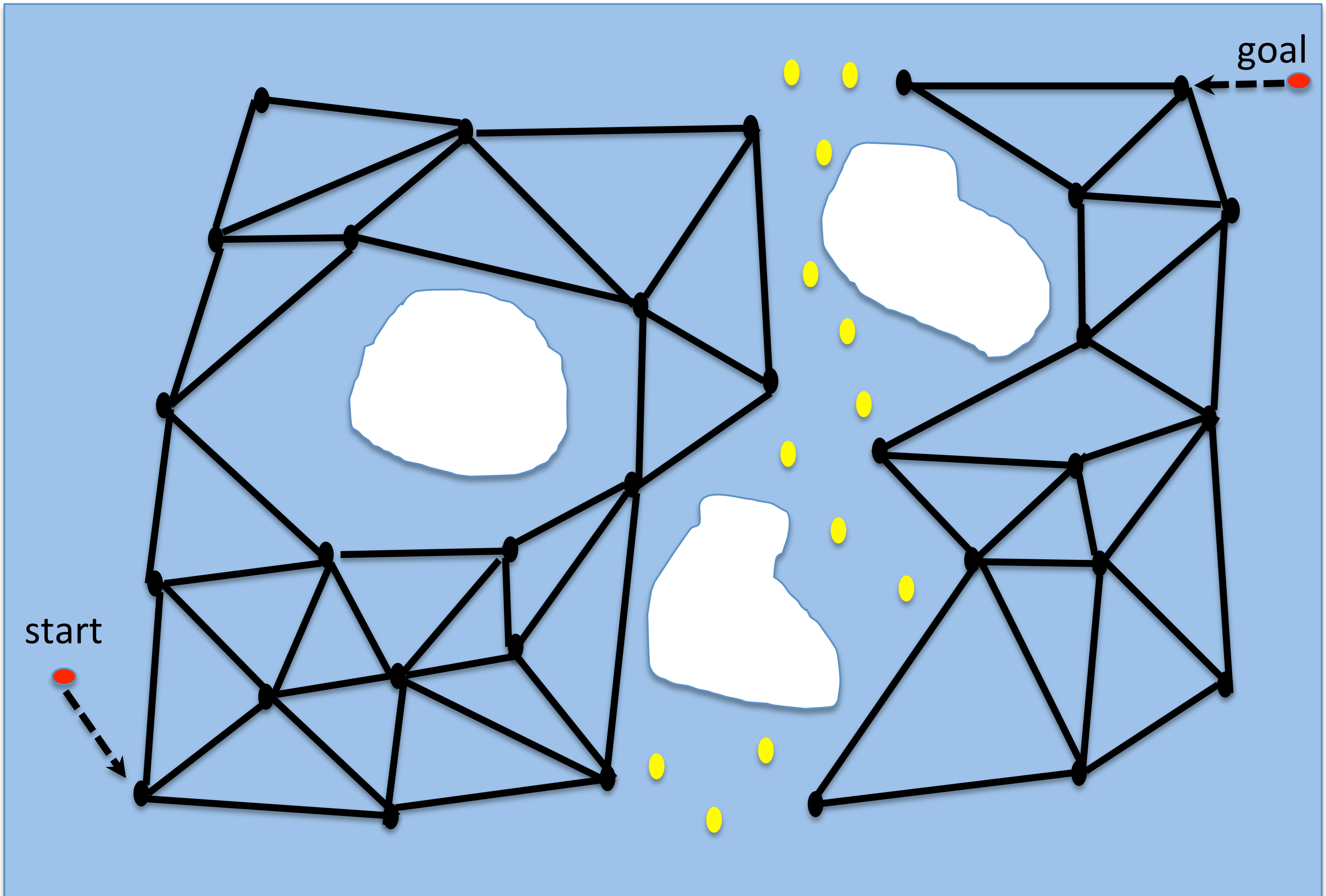
Problem: Graph may not be fully connected!

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

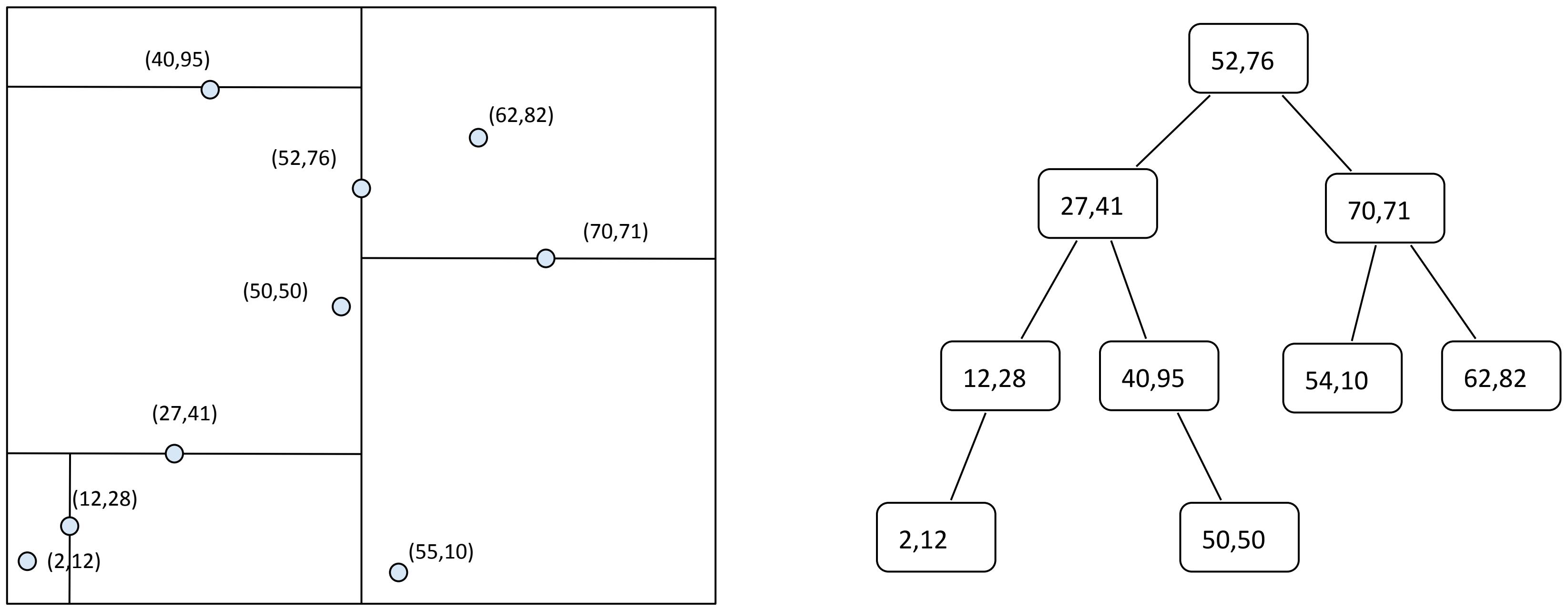
PRM Planner Details

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

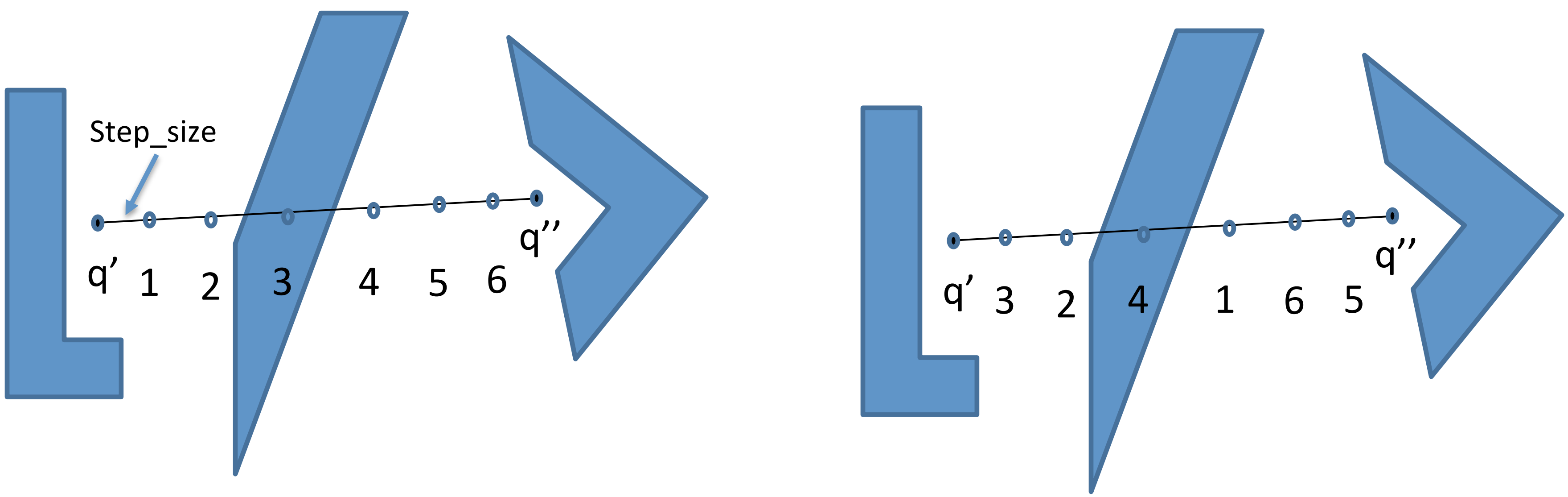
K-D Tree in 2-D



- Root of tree splits data along X dimension
- Successive levels of tree alternate X and Y splits

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

q1

q

q2

- 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:

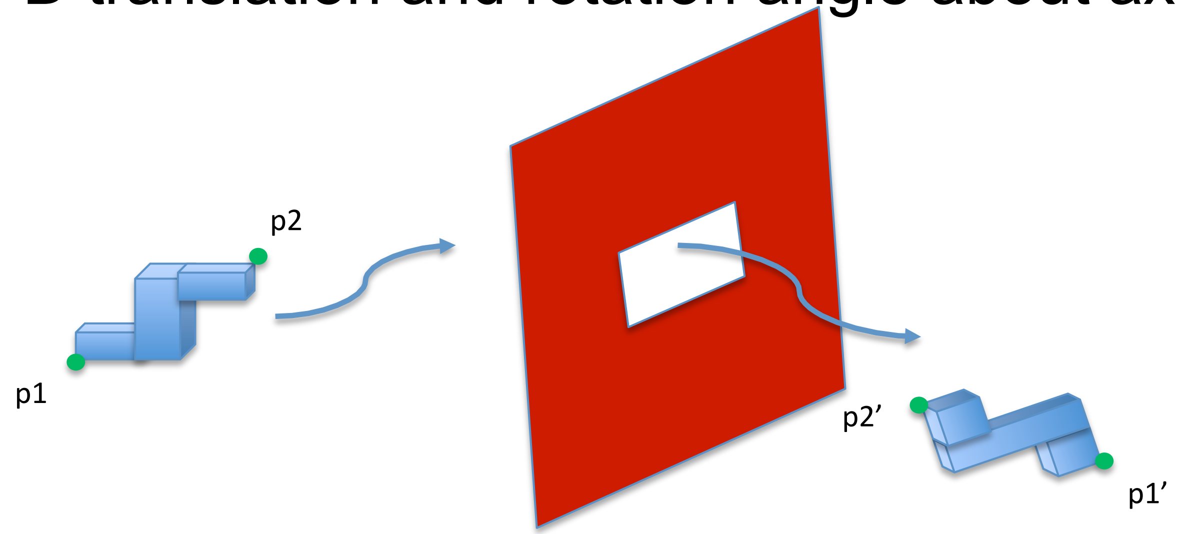
$$\mathbf{d(q,q') = \ || sample(q) - sample(q')||}$$

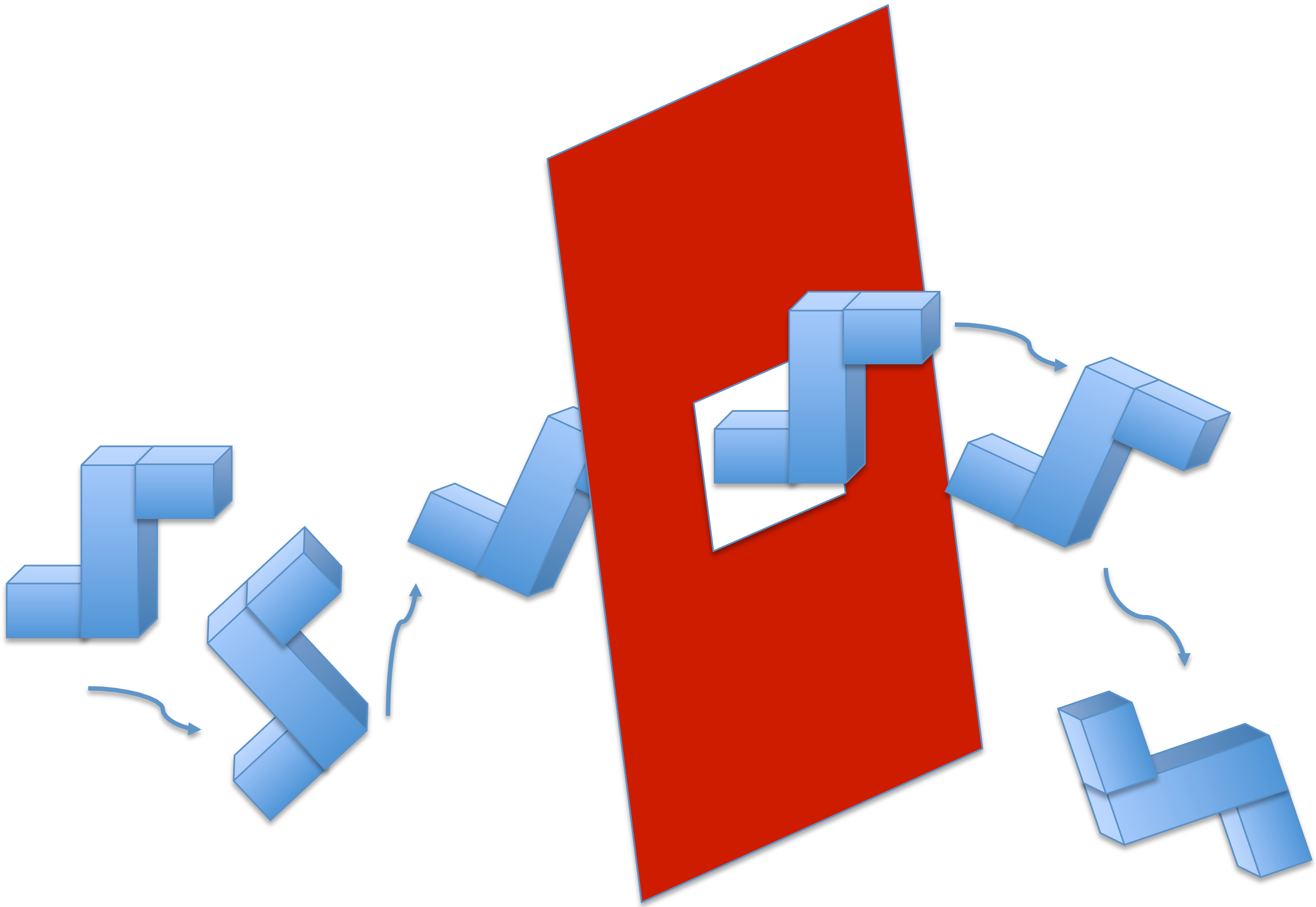
- Rigid robot: just choose 2 points of maximal extent as samples

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

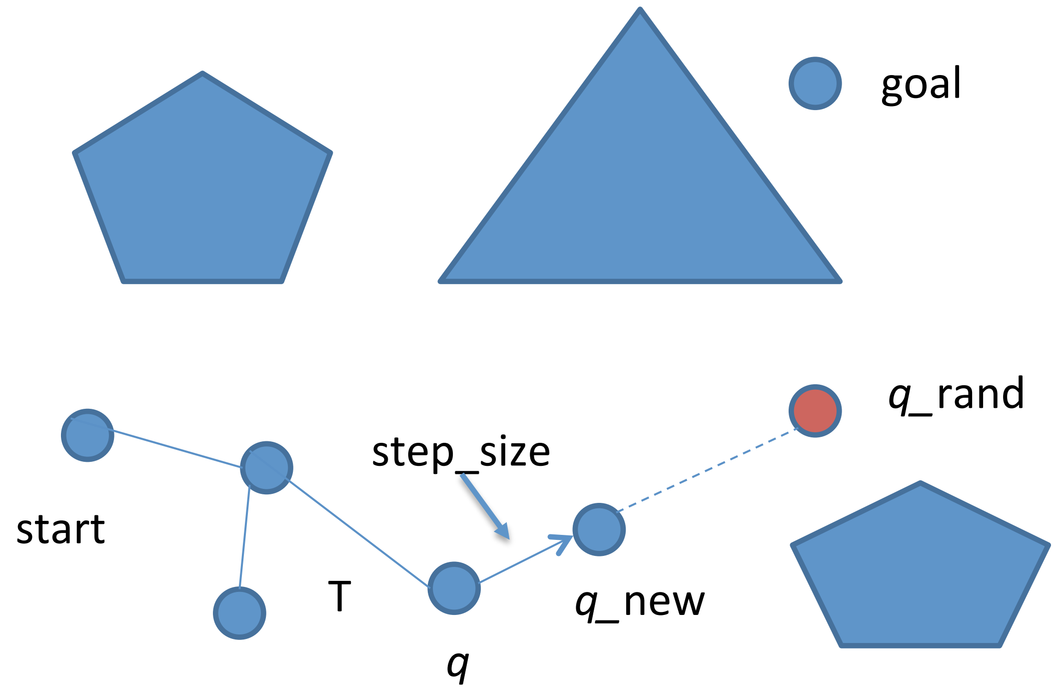




RRT: Rapidly-exploring Random Trees

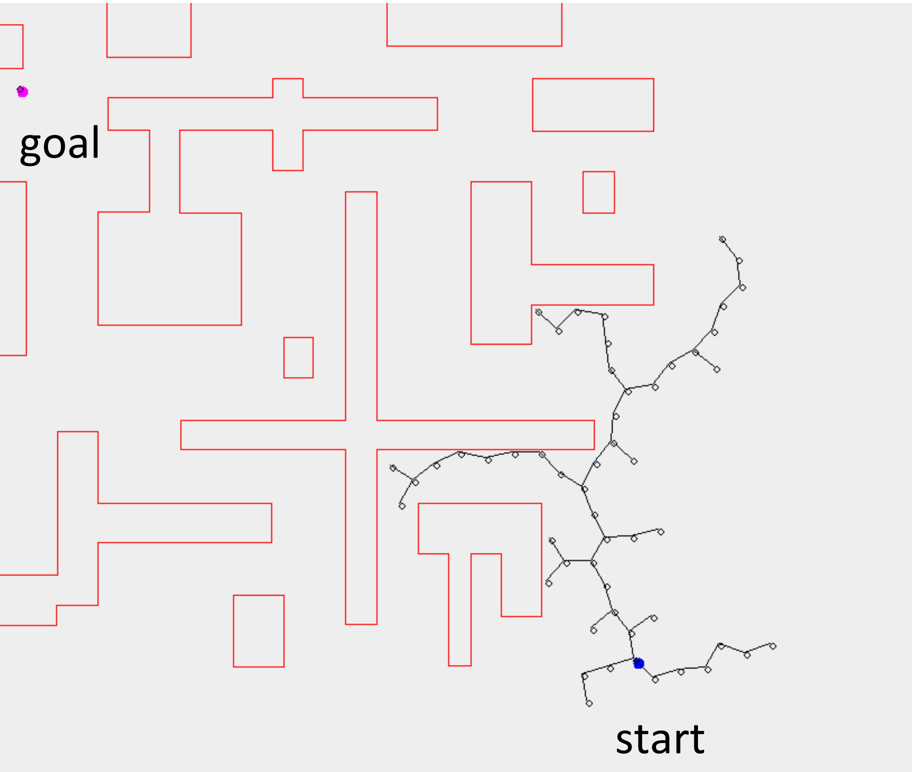
- Single query planner to get from config A to config B
- Randomly sample Q-Free for path from q_start to q_goal, growing a tree towards goal
- Can use 2 trees, rooted at q_start and q_goal.
- As trees grow, the eventually share a common node, and are merged into a path

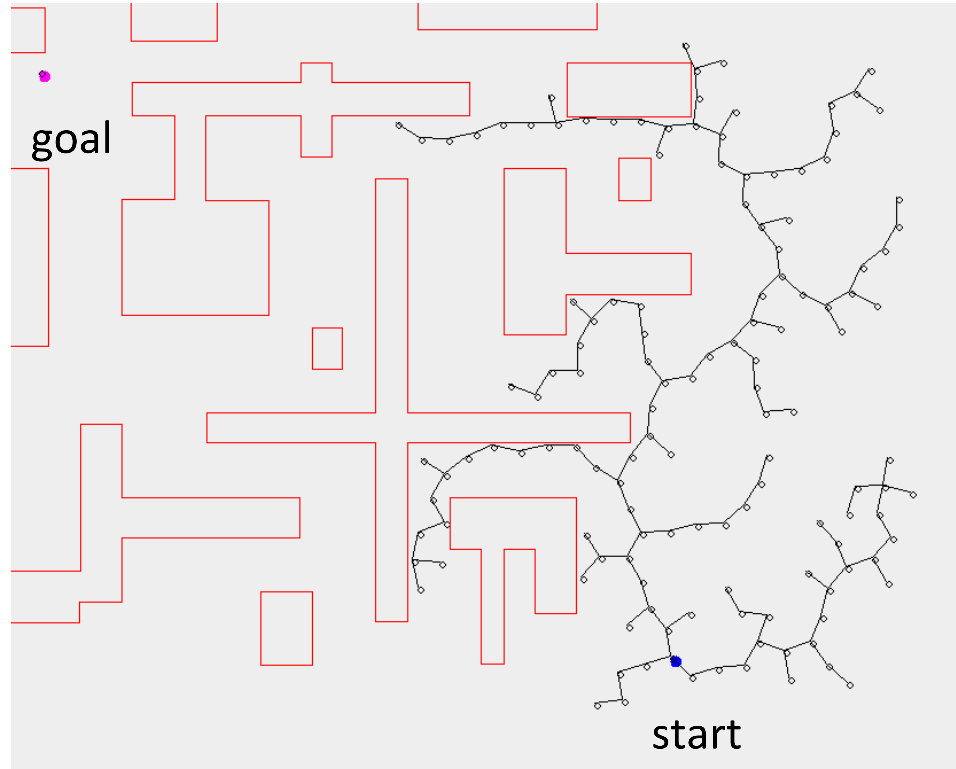
RRT: Build Tree Algorithm



- 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…

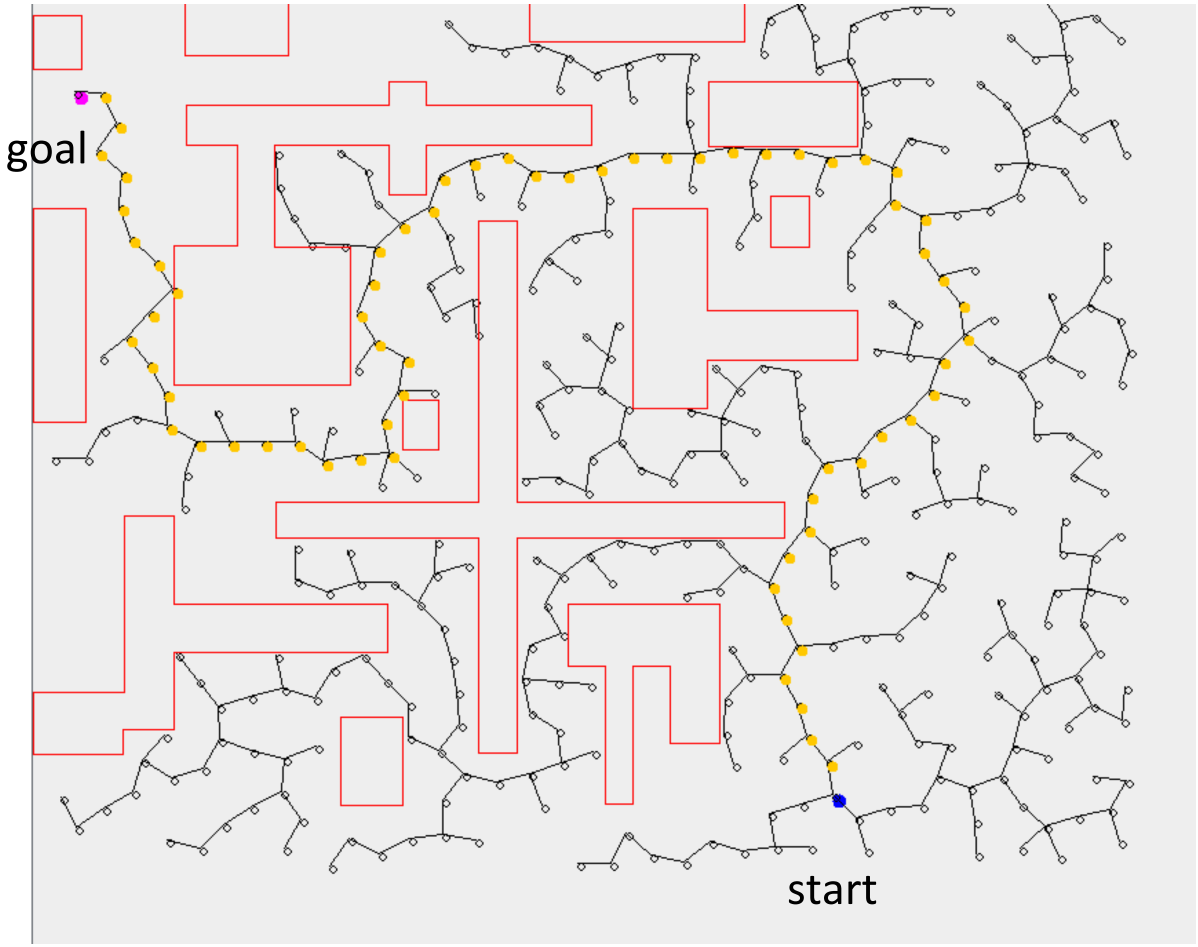
RRTs





- 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

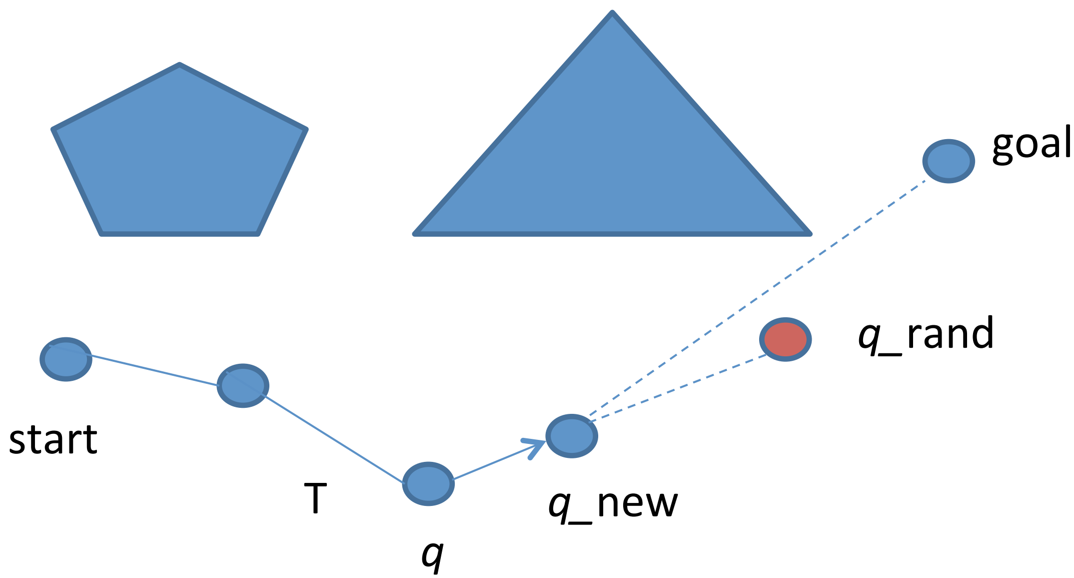
RRTs



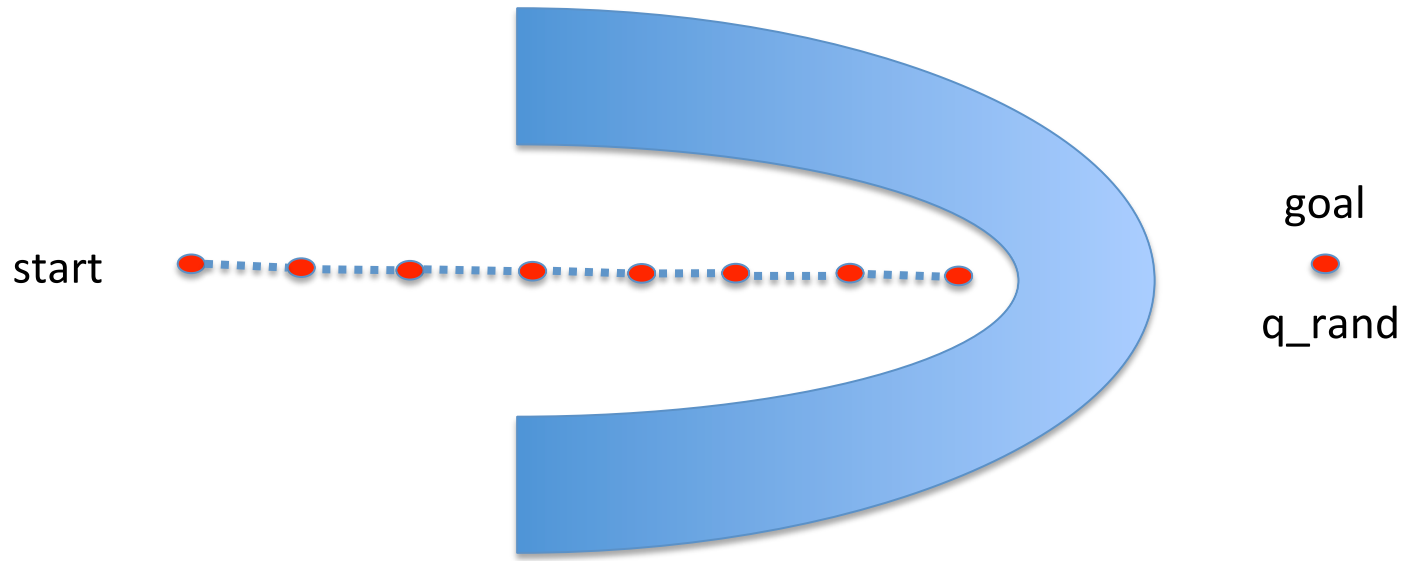
- Once tree reaches the goal, we have a path
- Path is not optimal in any sense
- Path can be different each time - stochastic
- Scales to higher dimensions

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

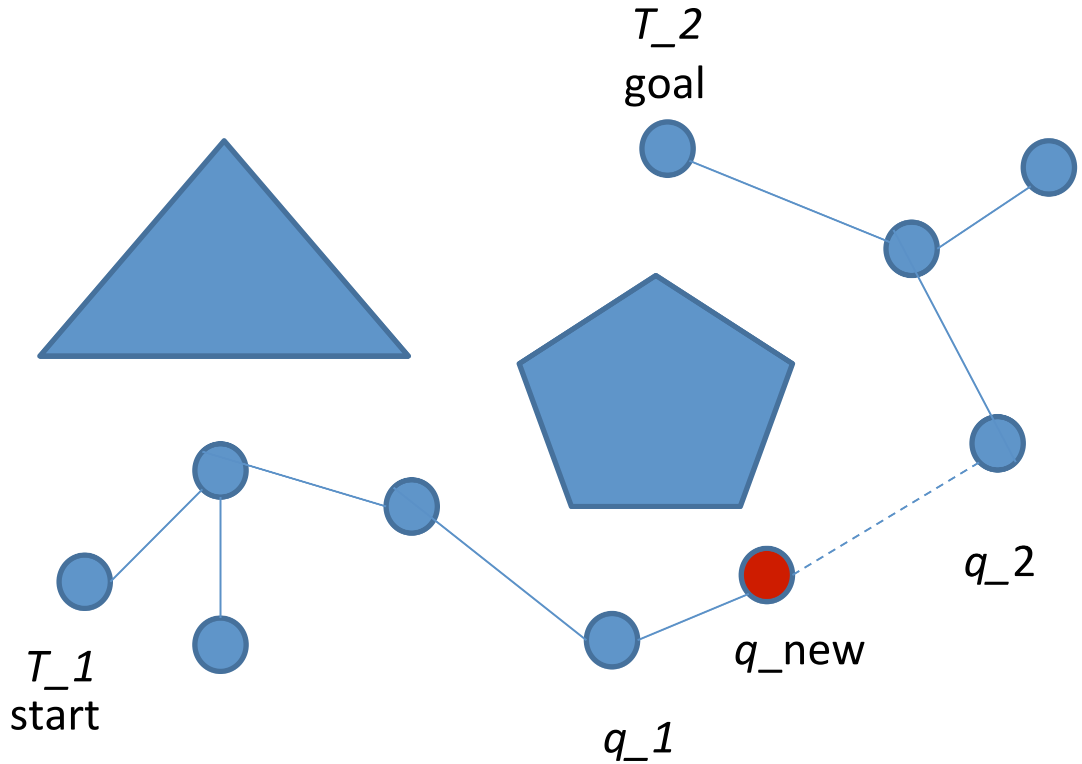
BiDirectional RRT

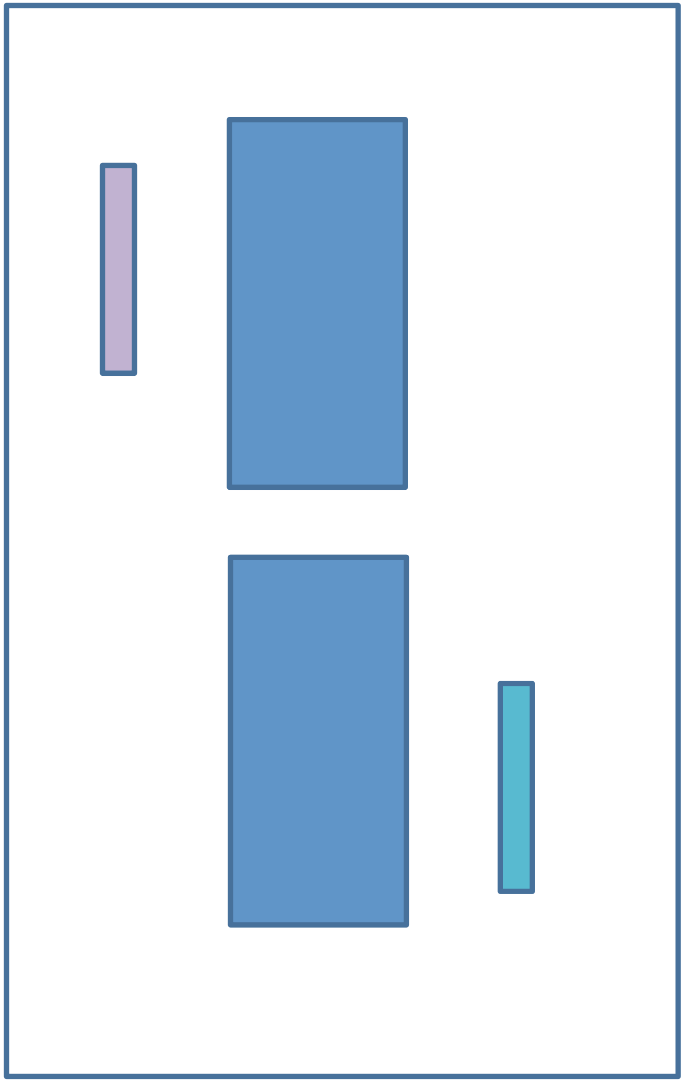
Use 2 trees (T_1, T_2) one rooted at start, one at goal

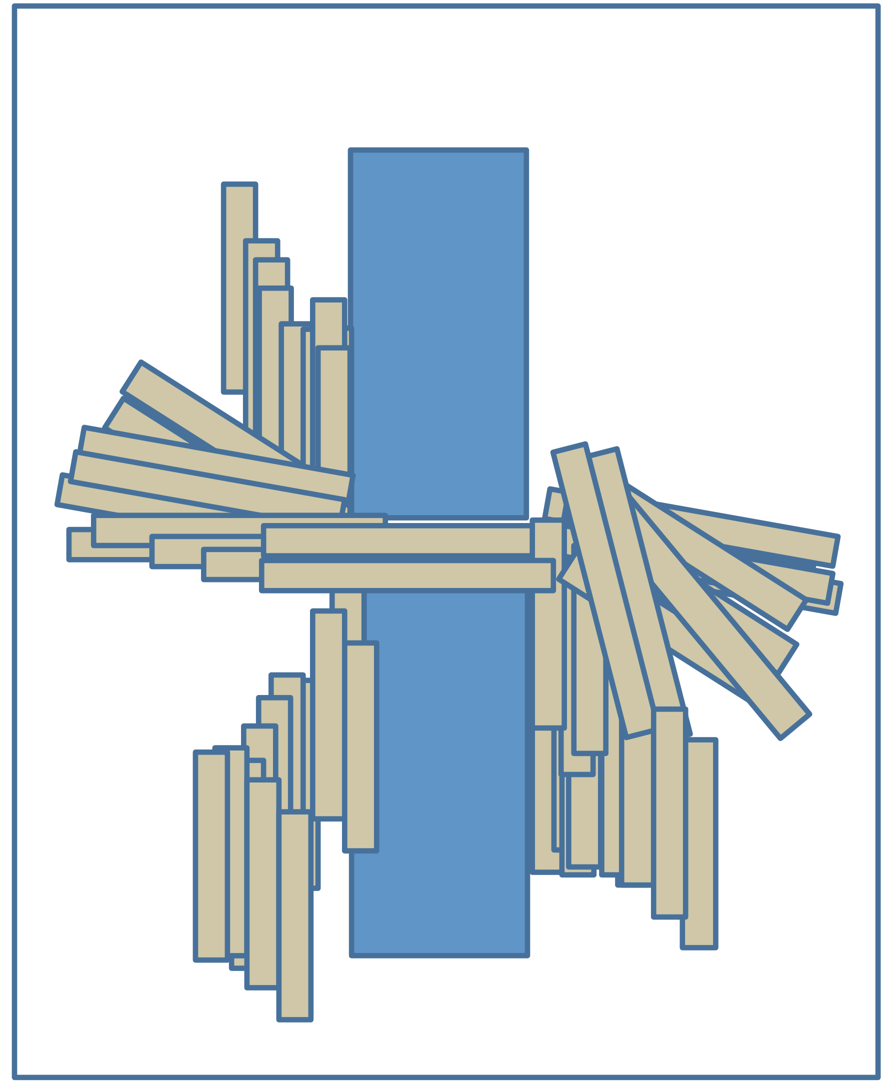
To connect the trees (and form a path):

- Expand tree T_1 randomly, add node q_new
- Expand T_2 towards q_new
  - If tree T_2 connects to q_new, path formed
    else add a q_new for tree T_2
- Now expand T_1 to q_new in tree T_2
- Keep swapping T_1 and T_2 for expansion towards the other tree until they meet

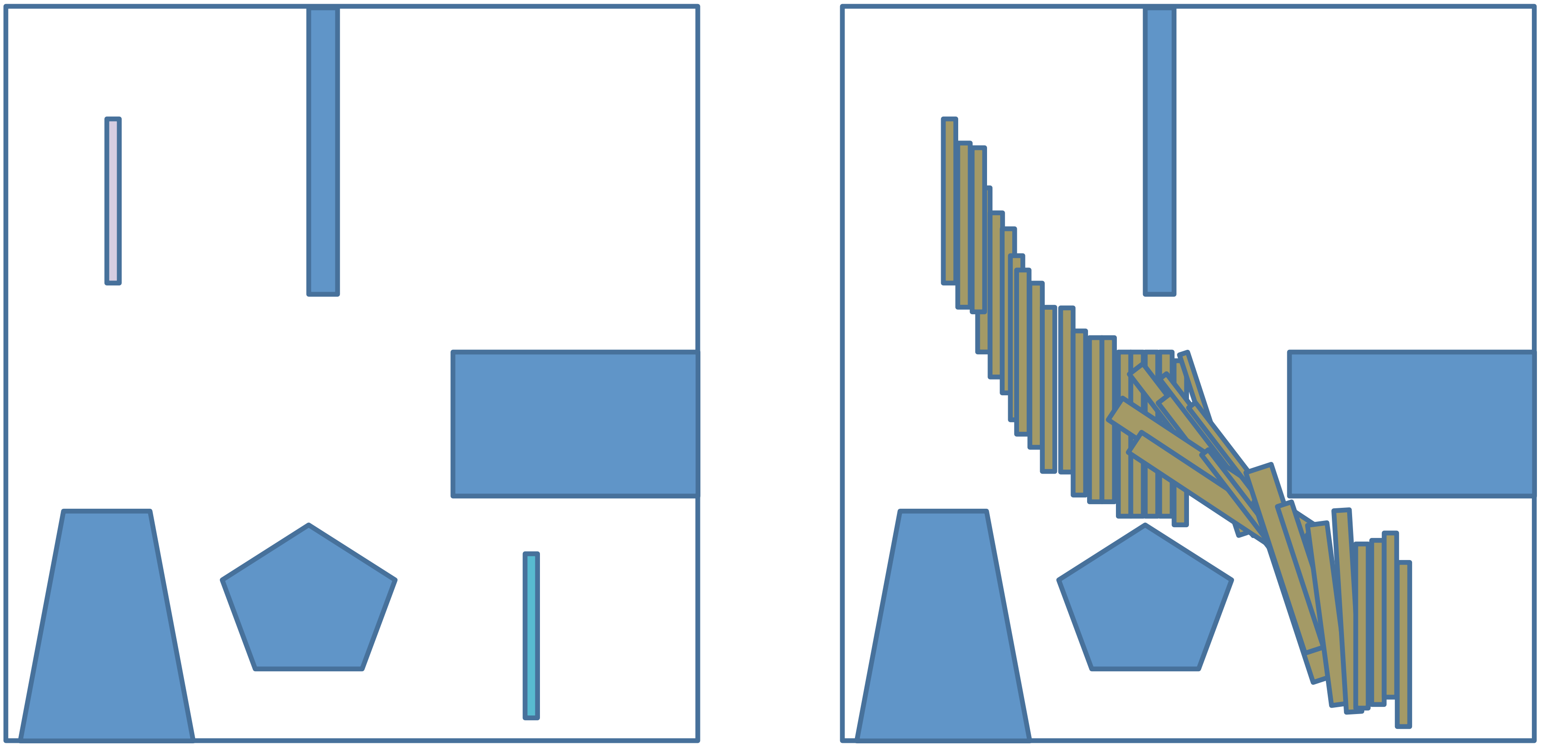
BiDirectional RRT





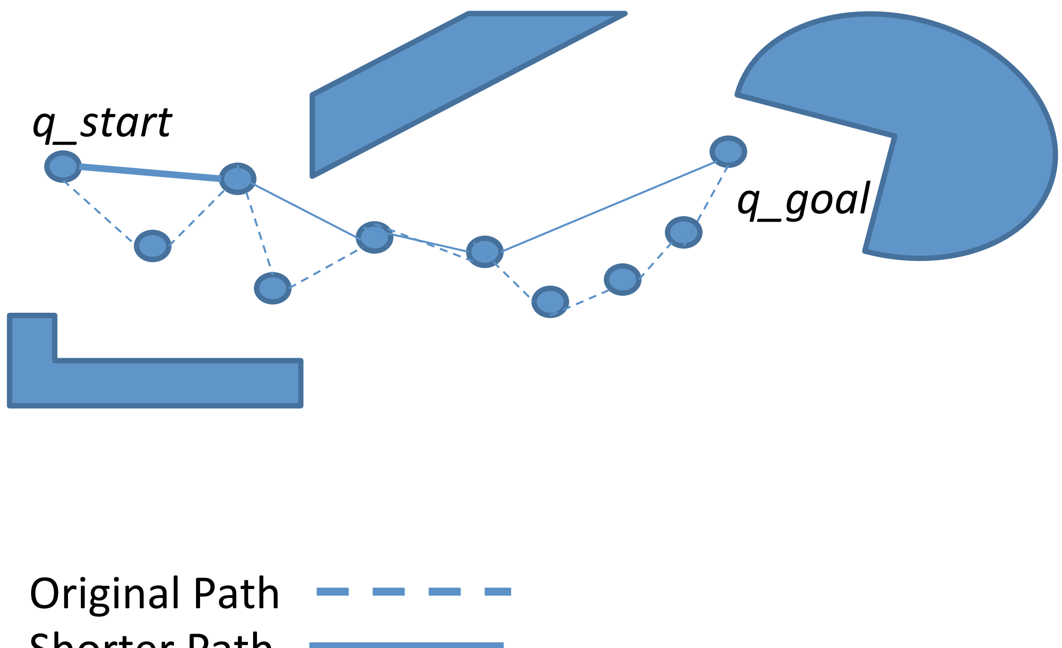


Time-lapse paths



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.



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