Introduction to Robotics

Motion Planning II

WS 2015 / 16

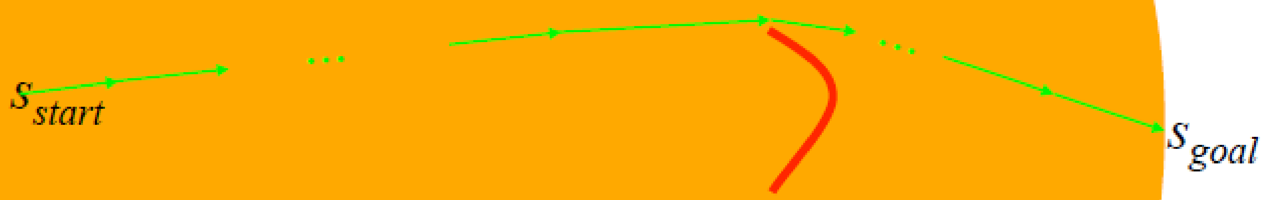
Prof. Dr. Daniel Göhring
Intelligent Systems and Robotics
Department of Computer Science
Freie Universität Berlin

Freie Universität Berlin

Heuristics in Heuristic Search

- Dijkstra's: expands states in the order of $f = g$ values

What are the states expanded?

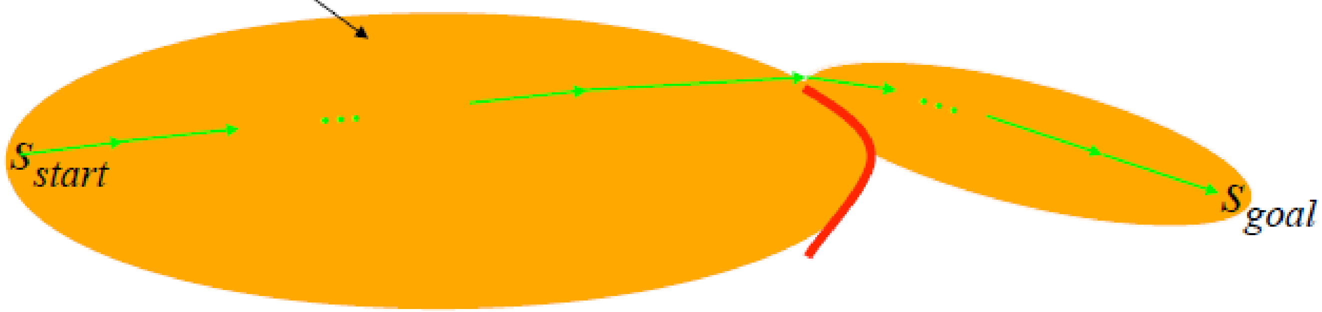


Slides: Maxim Likhachev

Heuristics in Heuristic Search

- A* Search: expands states in the order of $f = g+h$ values

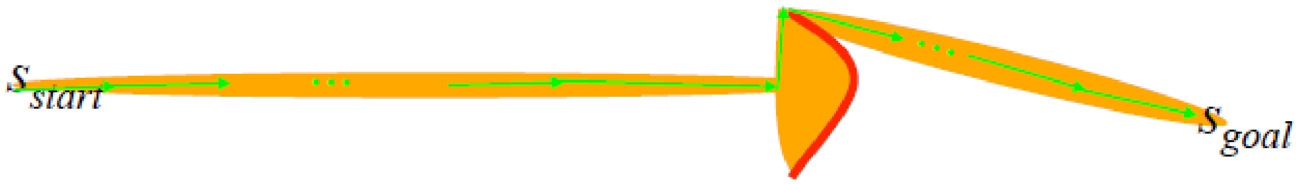
for high-D problems, this results in A being too slow and running out of memory*



Freie Universität Berlin

Heuristics in Heuristic Search

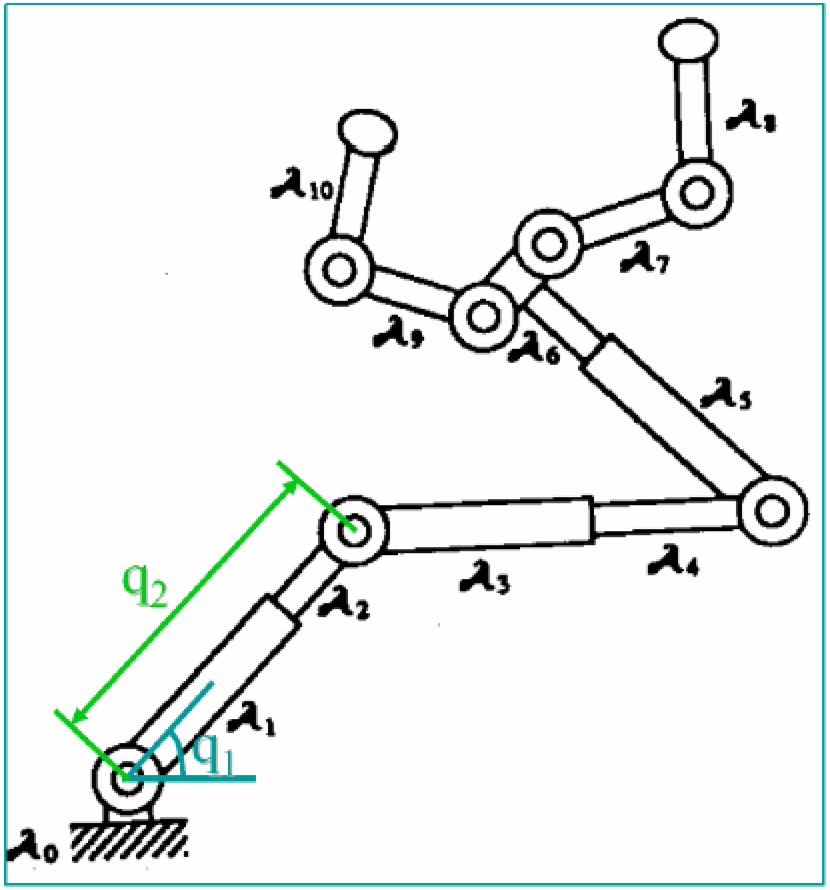
- Weighted A* Search: expands states in the order of $f = g + \varepsilon h$ values, $\varepsilon > 1$
- bias towards states that are closer to goal



Robot Motion Planning

- Search in geometric structures
- ***Spatial reasoning***
- Challenges:
  - Continuous state space
  - Large dimensional space
- Application of other search approaches like A*

Degrees of Freedom - Revisited

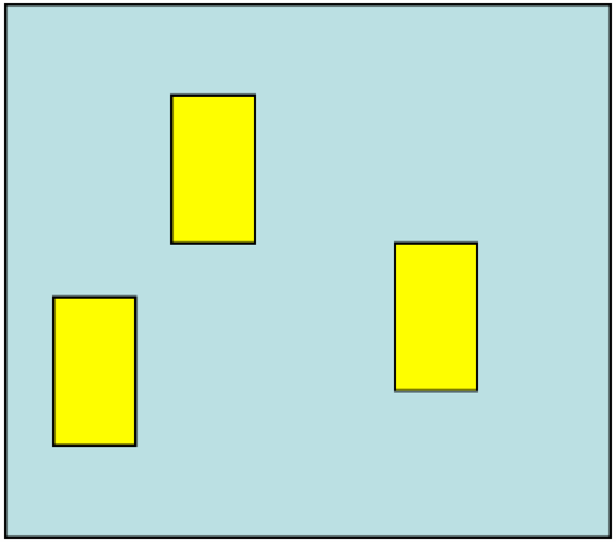


- The geometric configuration of a robot is defined by *p* degrees of freedom (DOF)
- Assuming *p* DOFs, the geometric configuration *A* of a robot is defined by *p* variables:

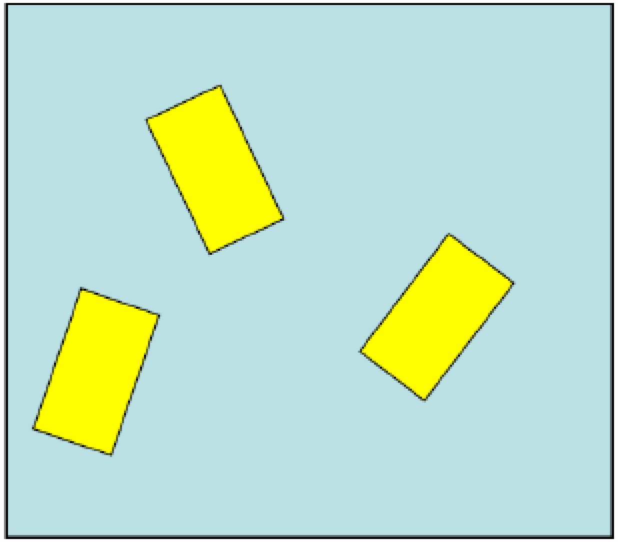
$A(\boldsymbol{q})$ with $\boldsymbol{q} = (q_1, \ldots, q_p)$

- Examples:
  - Prismatic (translational) DOF: q_i is the amount of translation in some direction
  - Rotational DOF: q_i is the amount of rotation about some axis

Examples



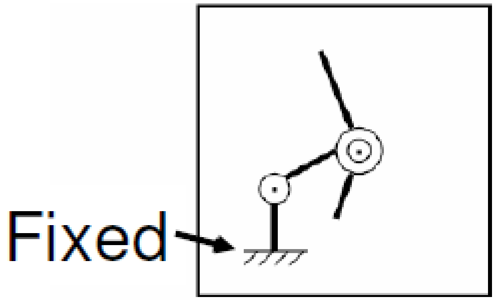
Allowed to move only in *x* and *y*: 2DOF



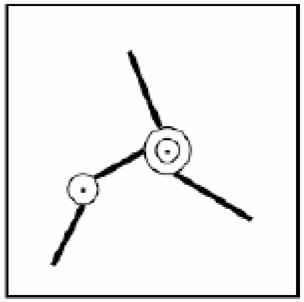
Allowed to move in *x* and y and to rotate: 3DOF (x,y,θ)

Freie Universität Berlin

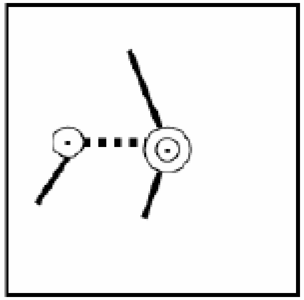
Examples



Fixed (attached at the base)

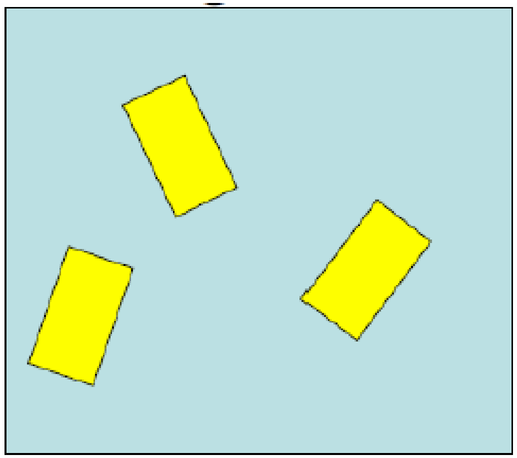


Free Flying



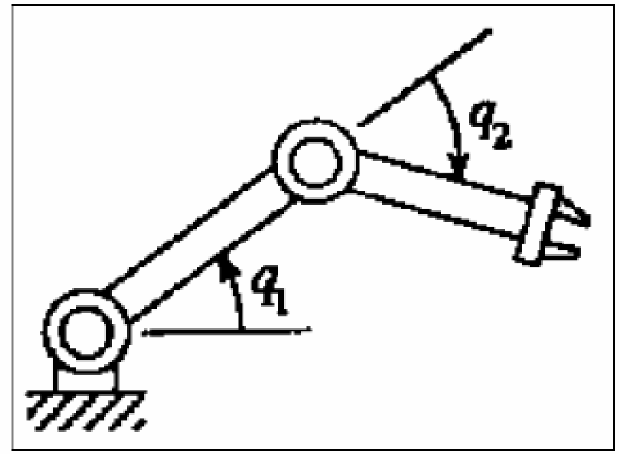
Fixed (the dashed line is constrained to be horizontal)

Configuration (C-Space)



$\boldsymbol{q} = (x, y, \theta)$

$\mathcal{C} = \Re^2 \times$ set of 2-D rotations

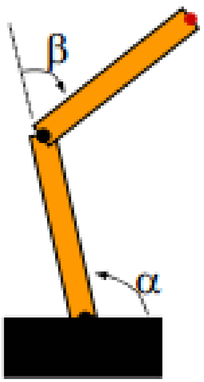


$\boldsymbol{q} = (q_1, q_2)$

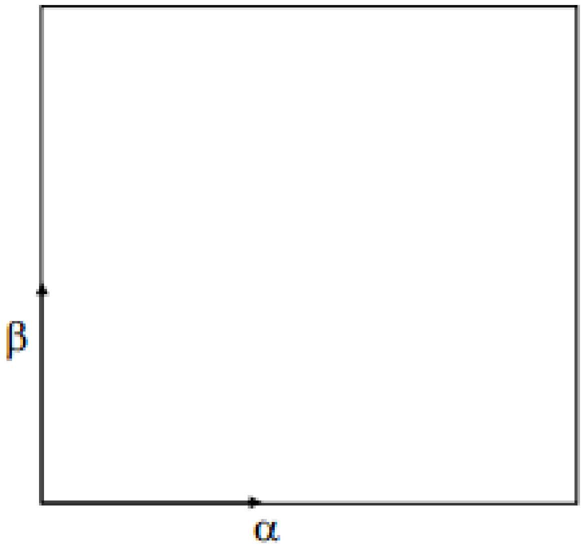
$\mathcal{C} =$ 2-D rotations x 2-D rotations

- Configuration space $\mathcal{C}$ = set of values of $\boldsymbol{q}$ corresponding to legal configurations of the robot
- Defines the set of possible parameters (the search space) and the set of allowed paths

Configuration Space



2R manipulator



Configuration space

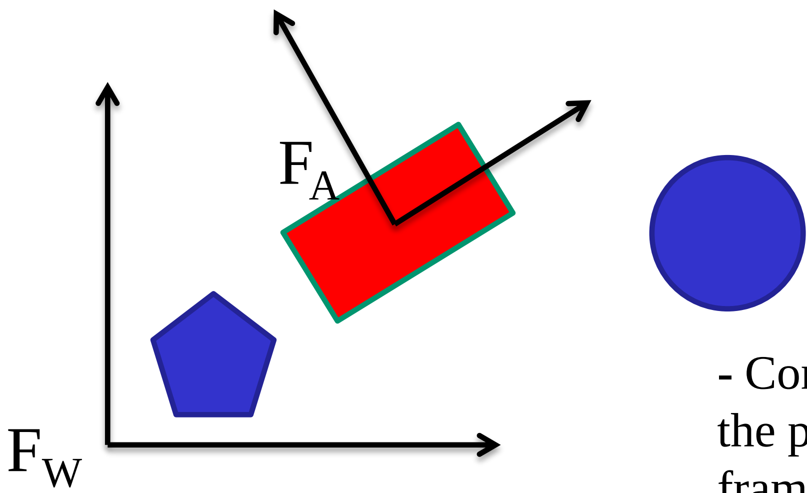
Motion Planning

- The world consists of
  - Obstacles:
    - already occupied space,
    - robot cannot go there
  - Free Space
    - Space which is not occupied
    - Robot „might“ go there
    - Determine if it can go there
  - Start position and goal position
- In a nutshell:

Find a collision-free trajectory under physical, geometrical and temporal constraints from start to goal.

World Frame and Robot Frame

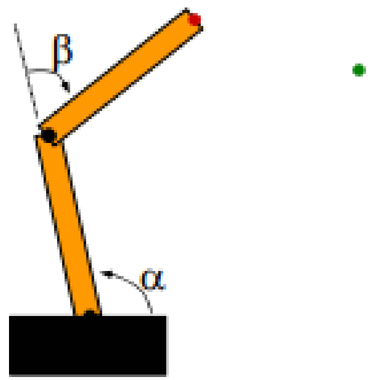
- A: single rigid object –(the robot)
- W: Euclidean space where A moves;
- W = R^2 or R^3
- B_1,…B_m: fixed rigid obstacles distributed in W



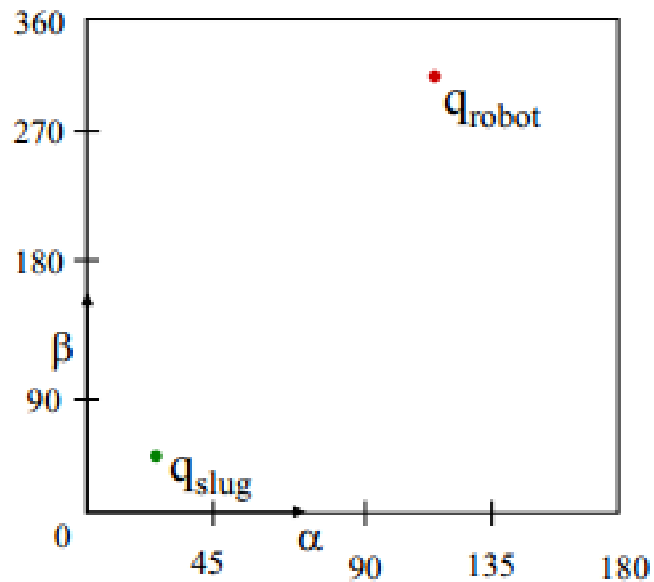
F_W: World frame
F_A: Robot frame, a moving frame rigidly associated with the robot

- Configuration q of A is a specification of the position and orientation of A in world frame F_W.
- Configuration space is the space of all possible robot configurations

Configuration Space



Two points in the robot's workspace

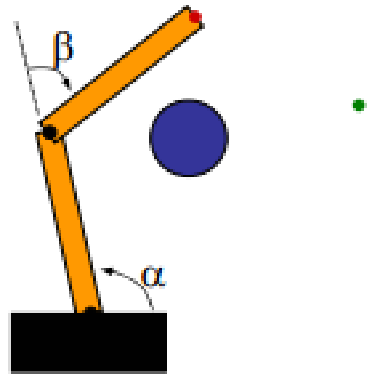


Torus

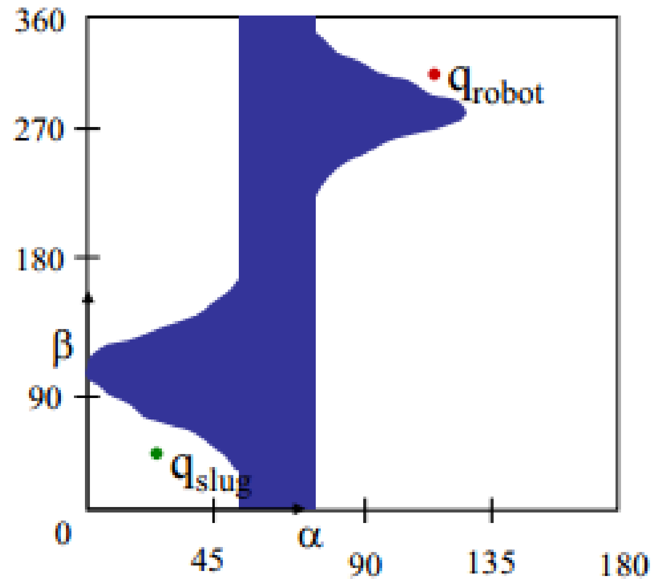
(wraps horizontally and vertically)

Objects in Configuration (or *C*)-Space

If the robot configuration is within the blue area, it will hit the obstacle



An obstacle in the robot's workspace

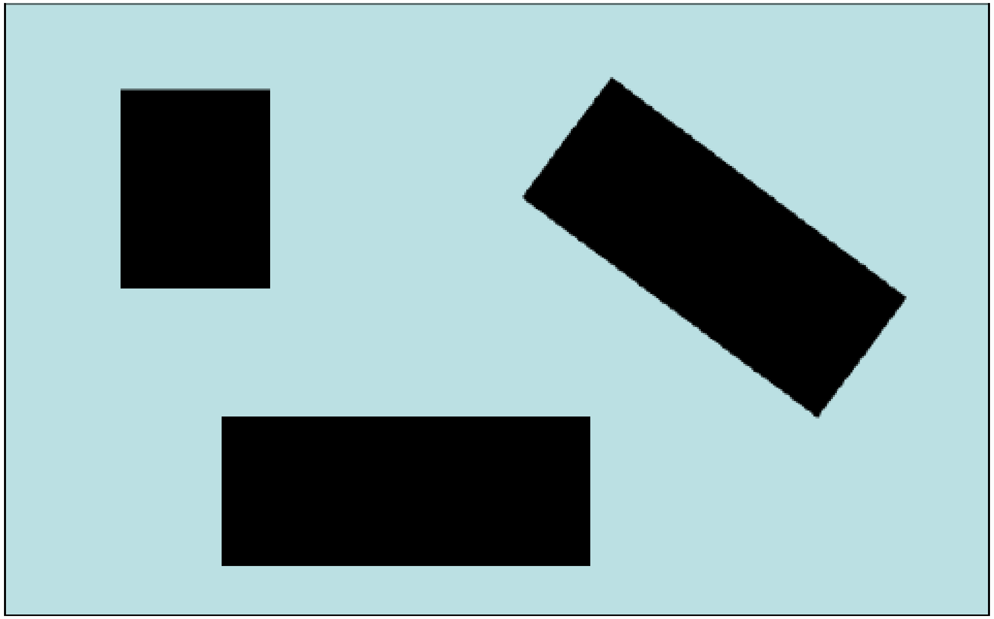


a "*C*-space" representation

What is dimension of the *C*-space of puma robot (6R)?

Visualization of high dimension *C*-space is difficult

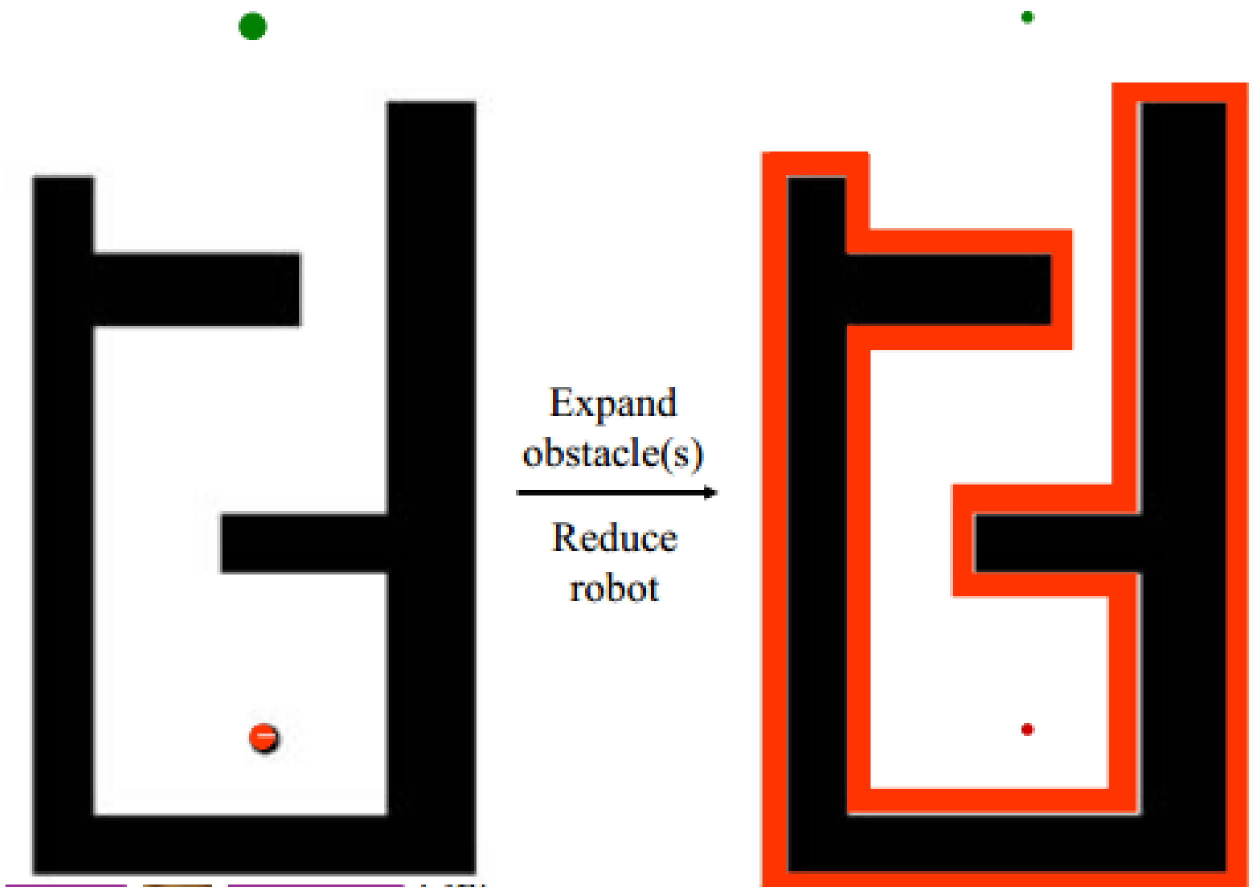
Free Space, Point Robot



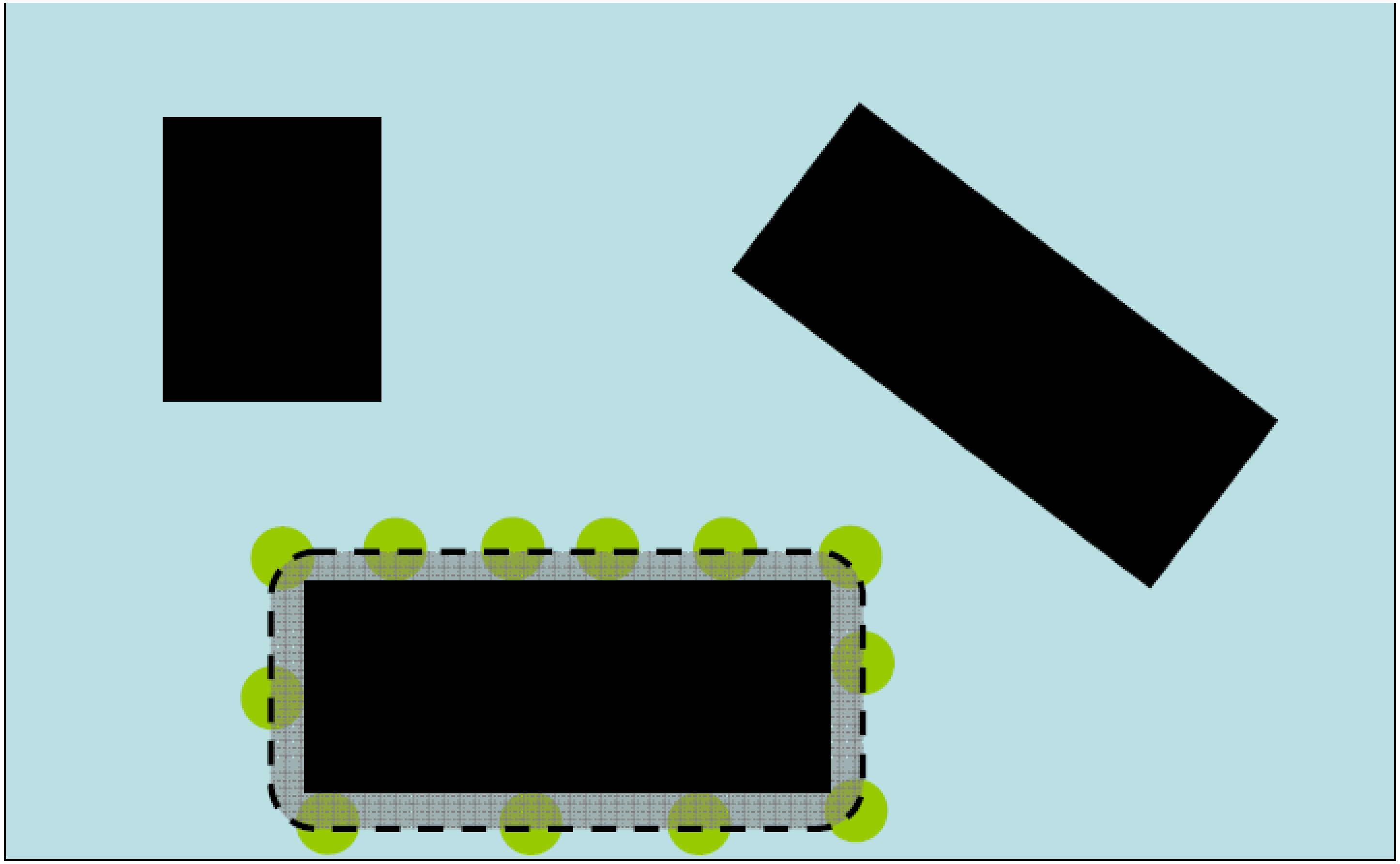
- $\mathcal{C}_{free}$ = {Set of parameters $\boldsymbol{q}$ for which $A(\boldsymbol{q})$ does not intersect obstacles}
- For a point robot in the 2-D plane: R^2 minus the obstacle regions

Freie Universität Berlin

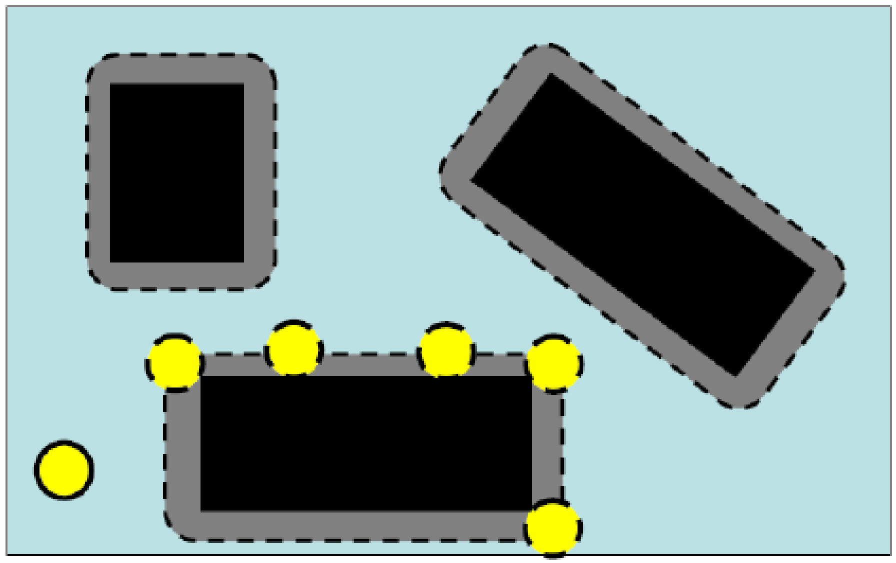
What if the Robot is not a Point



Free Space: Symmetric Robot

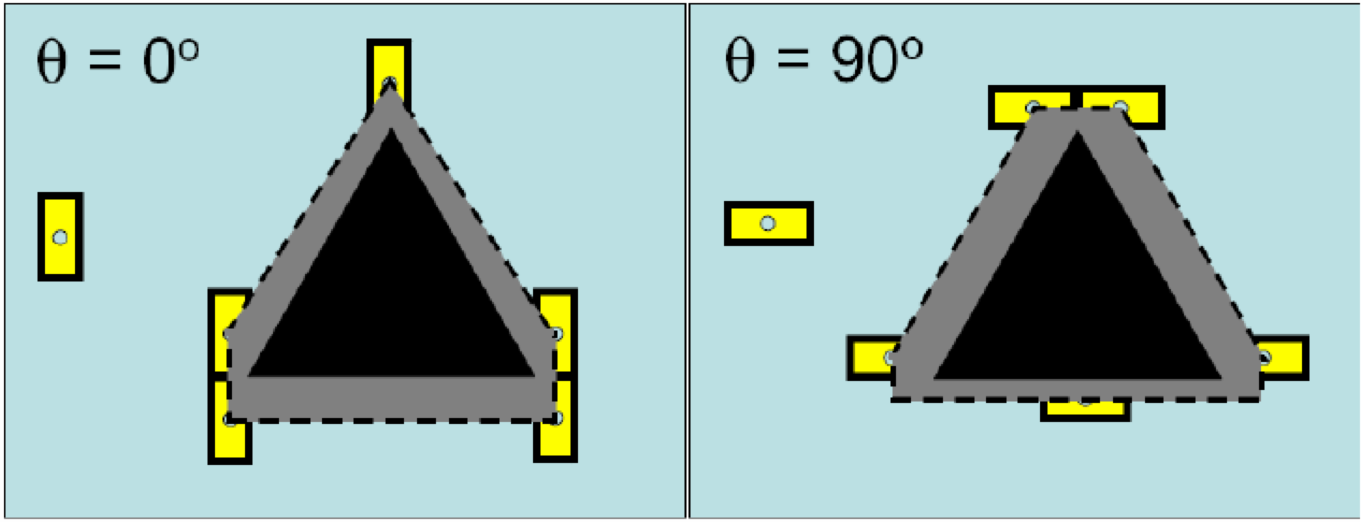


Free Space: Symmetric Robot



- We still have $\mathcal{C} = R^2$ because orientation does not matter
- Reduce the problem to a point robot by expanding the obstacles by the radius of the robot

Free Space: Non-Symmetric Robot



- The configuration space is now three-dimensional (x, y, θ)
- We need to apply a different obstacle expansion for each value of θ
- We still reduce the problem to a point robot by expanding the obstacles

Freie Universität Berlin

Free Space

From
Robot Motion Planning
J.C. Latombe

C-OBSTACLE REGION

$B_1, B_2, \ldots, B_m$ — obstacles

CB_i — C-obstacle

$\bigcup_{i=1}^{m} CB_i$ — C-obstacle region

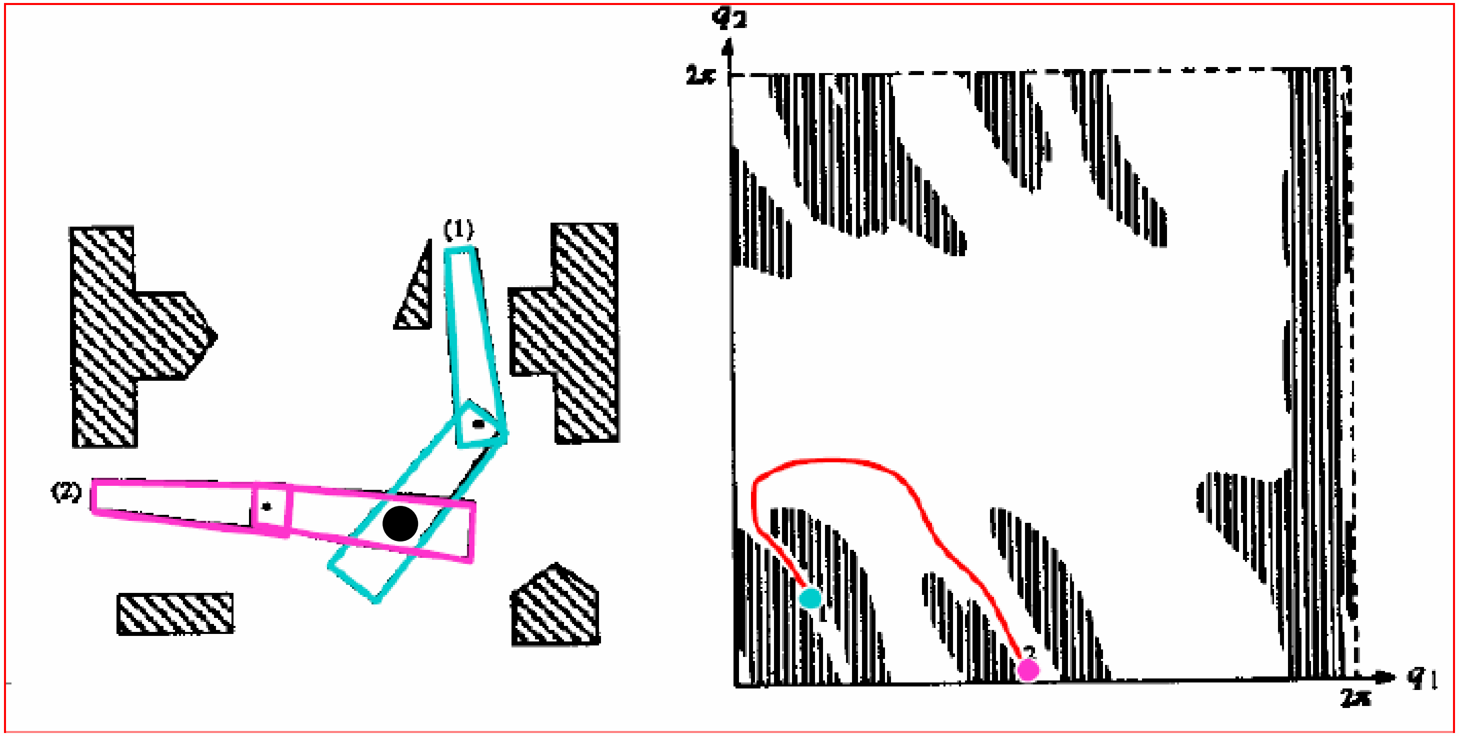
FREE SPACE

$$C_{free} = C \setminus \bigcup_{i=1}^{m} CB_i = \{q \in C : A(q) \cap \bigcup_{i=1}^{m} CB_i = \phi\}$$

Free configuration q iff $q \in C_{free}$

Freie Universität Berlin

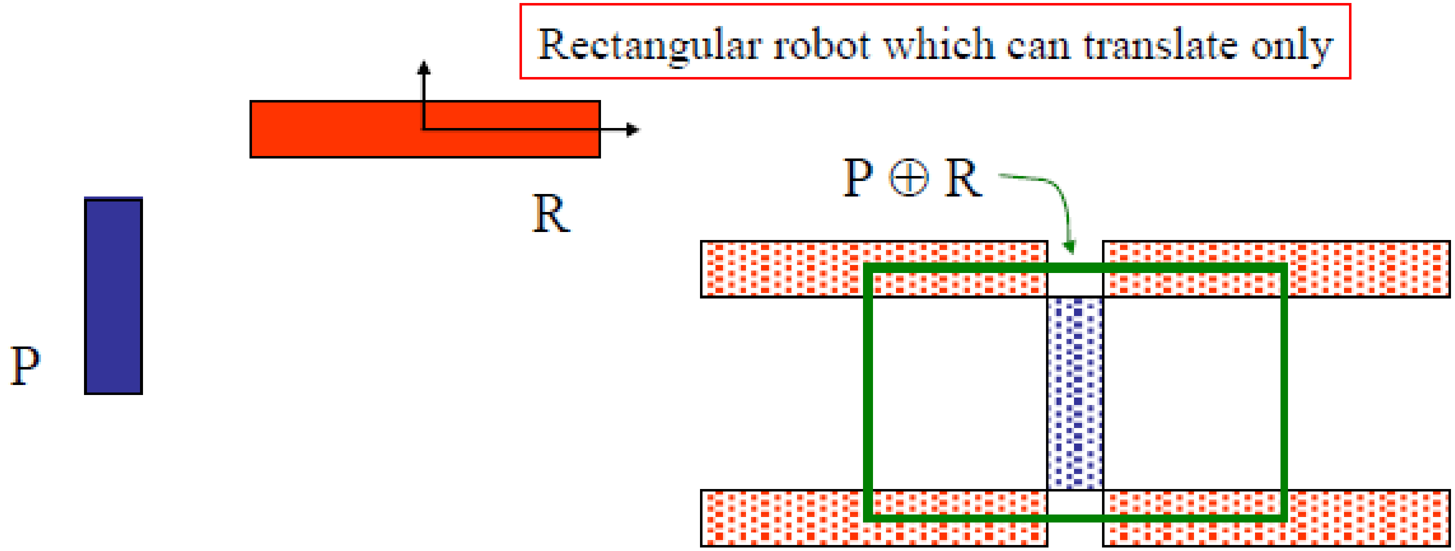
More Complex Configuration-Spaces



• In all cases: The problem is reduced to finding the path of a *point* through configuration space by "expanding the obstacles"

Minkowski Sum, Generating Obstacles in C-Space

This expansion of one planar shape by another is called the *Minkowski sum* $\oplus$



(Dilation operation) $$P \oplus R = \{\, p + r \mid p \in P \text{ and } r \in R \,\}$$

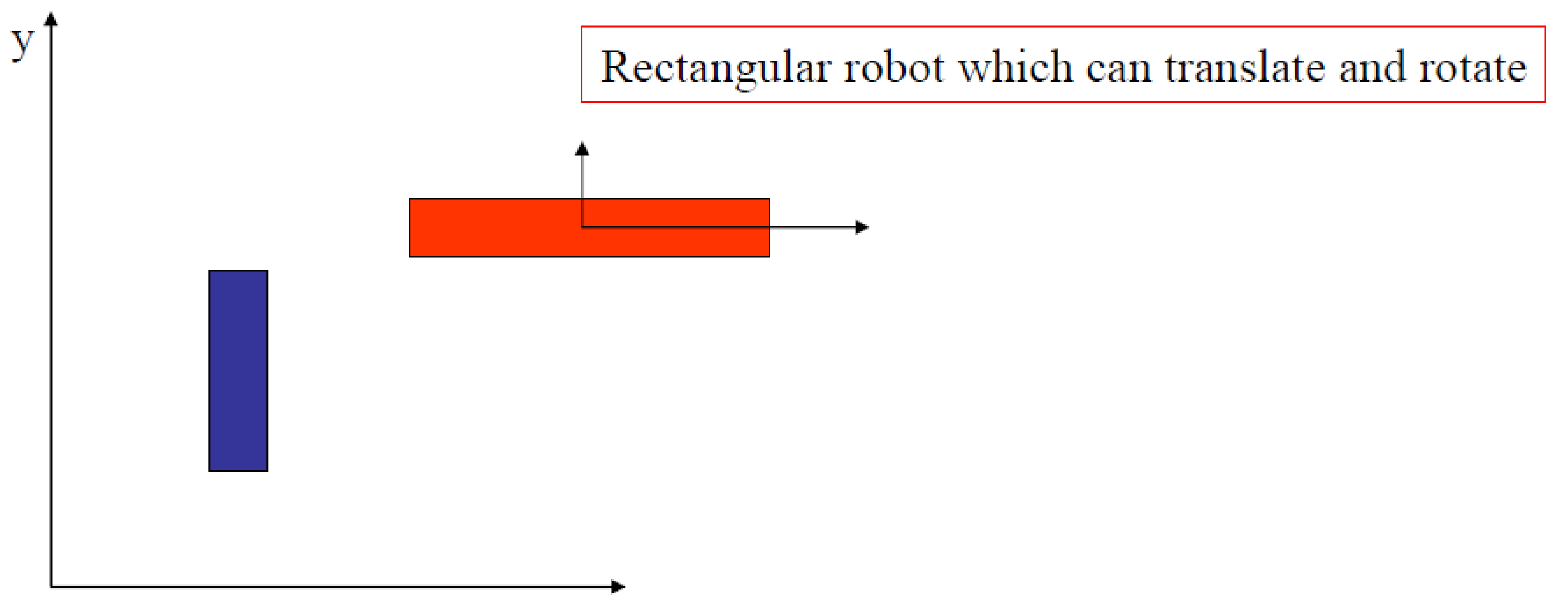
Used in robotics to ensure that there are free paths available.

Freie Universität Berlin

Additional Dimension

What would the $\mathcal{C}$-obstacle be if the rectangular robot (red) can ***translate*** *and* ***rotate*** in the plane.

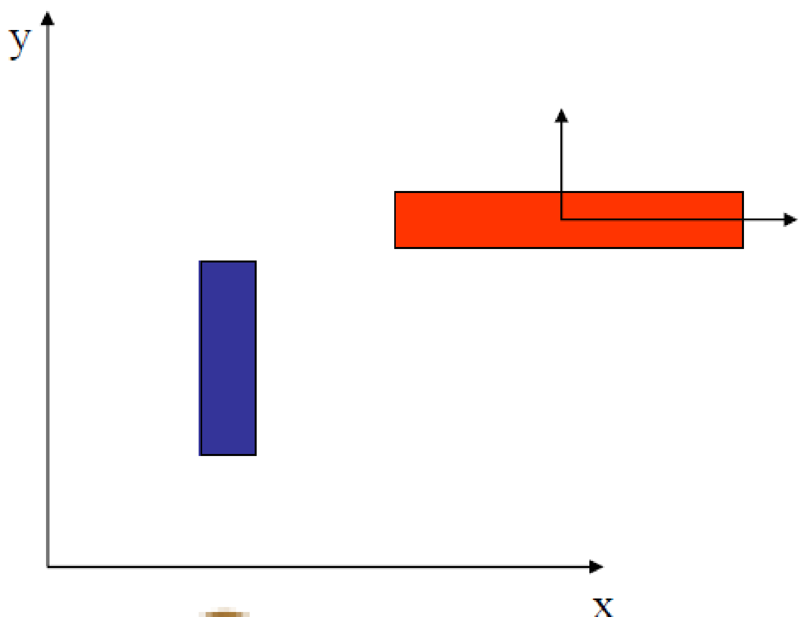
(The blue rectangle is an obstacle.)



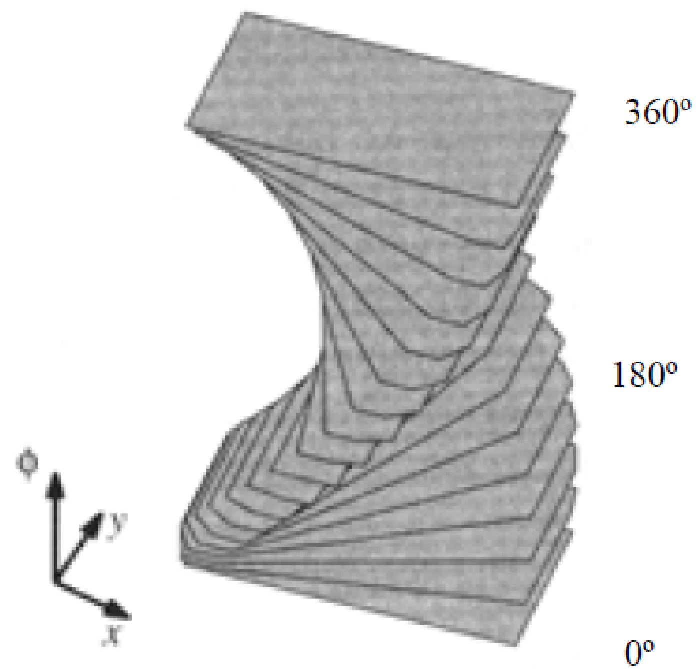
C-Obstacle in 3D

What would the $\mathcal{C}$-obstacle be if the rectangular robot (red) can ***translate** and **rotate*** in the plane.

(The blue rectangle is an obstacle.)



3-D
configuration space

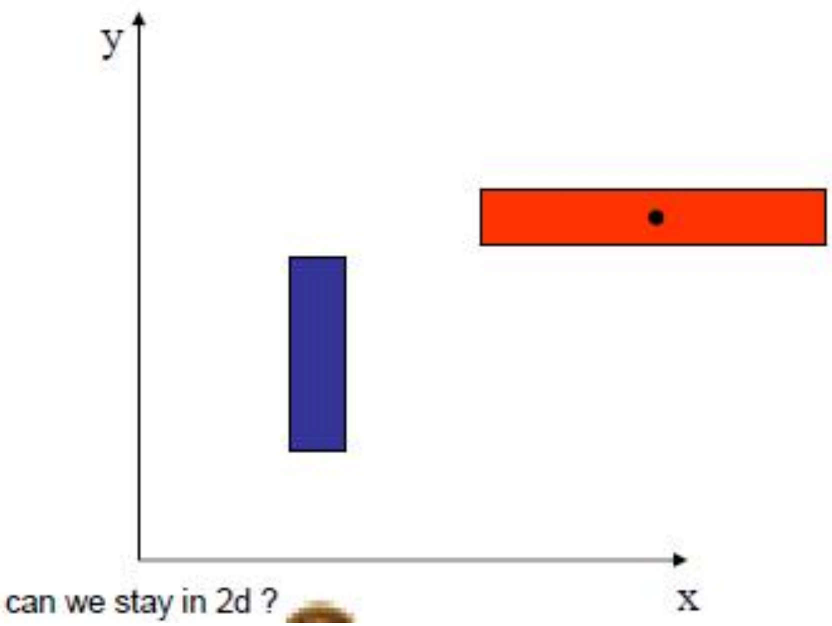


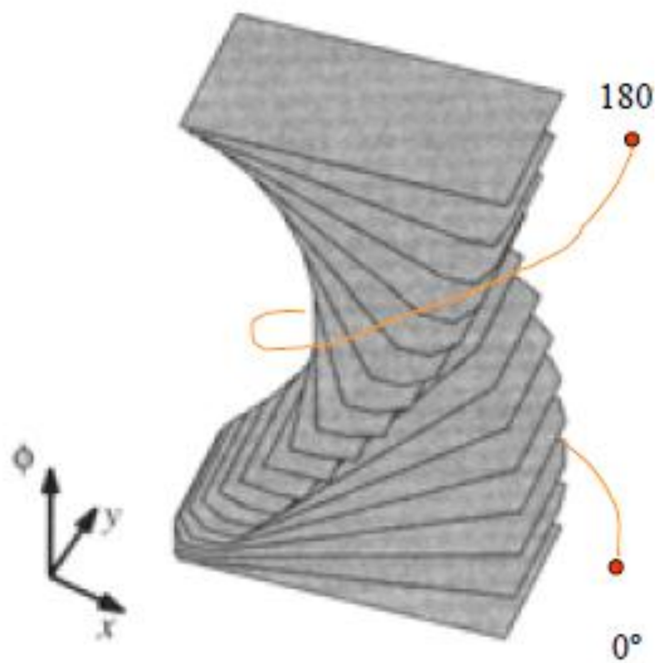
this is twisted...

C-Obstacle in 3D

What would the configuration space of a 3DOF rectangular robot (red) in this world look like?

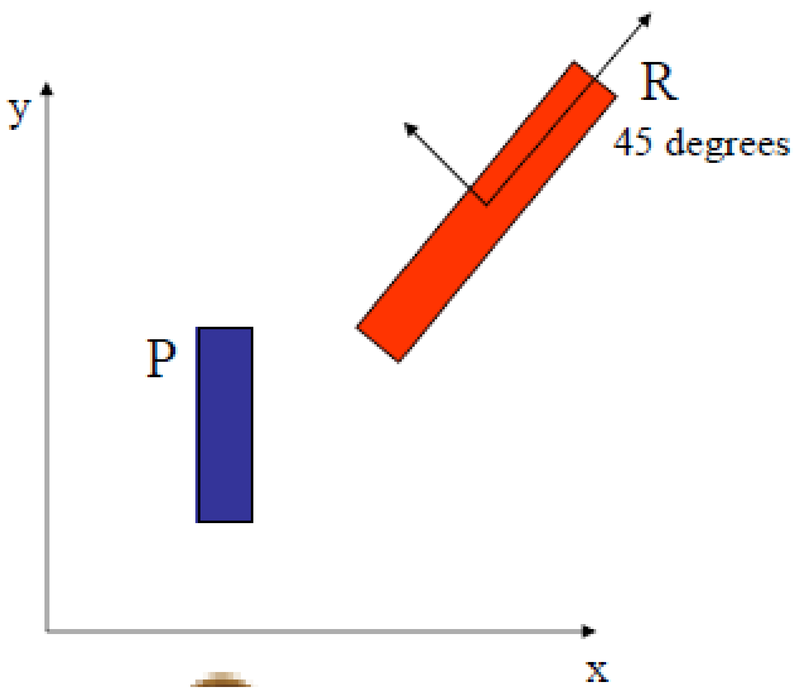
(The obstacle is blue.)

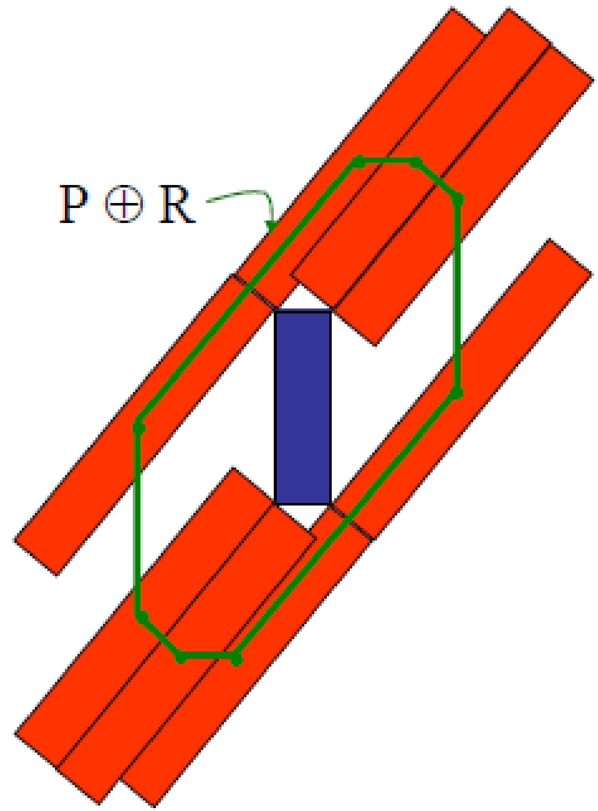




One Slice

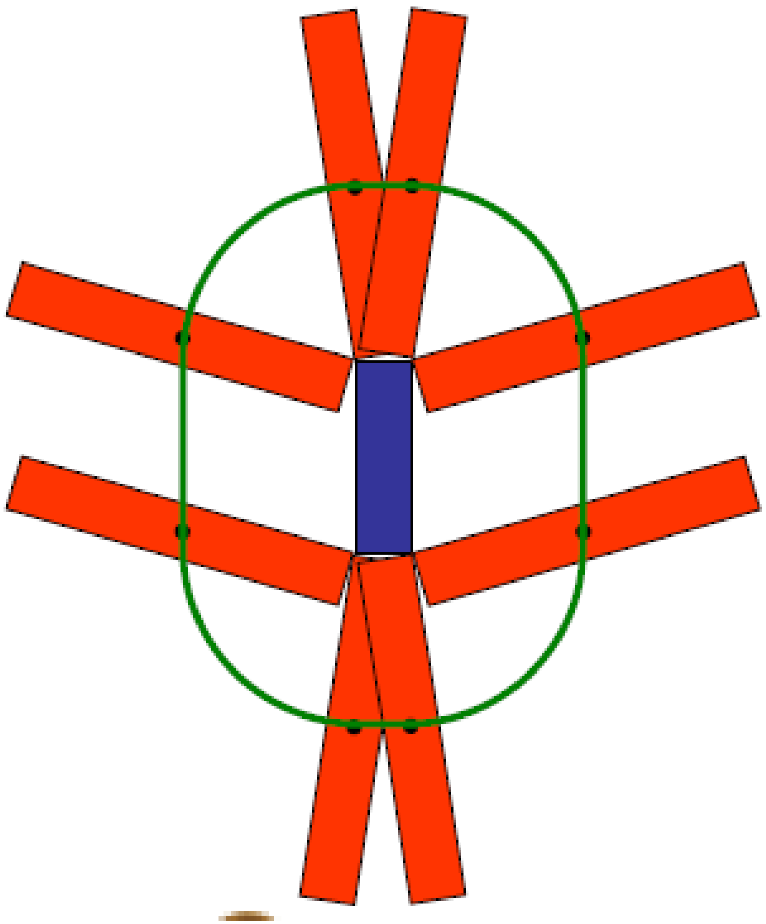
Taking one slice of the $\mathcal{C}$-obstacle in which the robot is rotated 45 degrees...

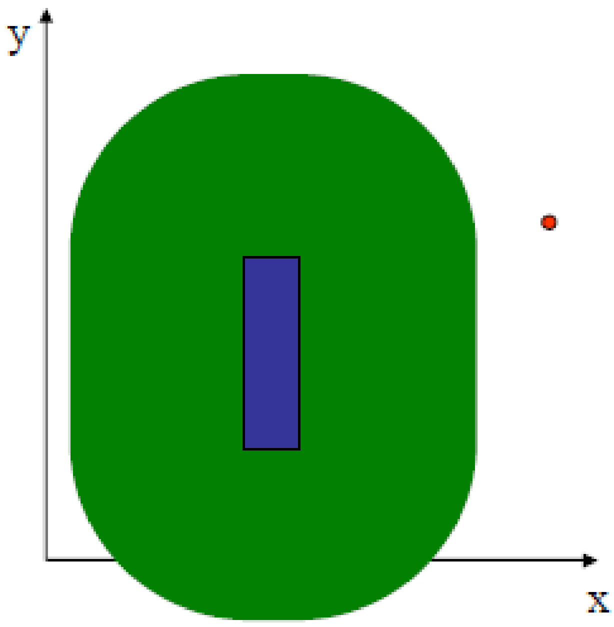




How many slices does P ⊕R have?

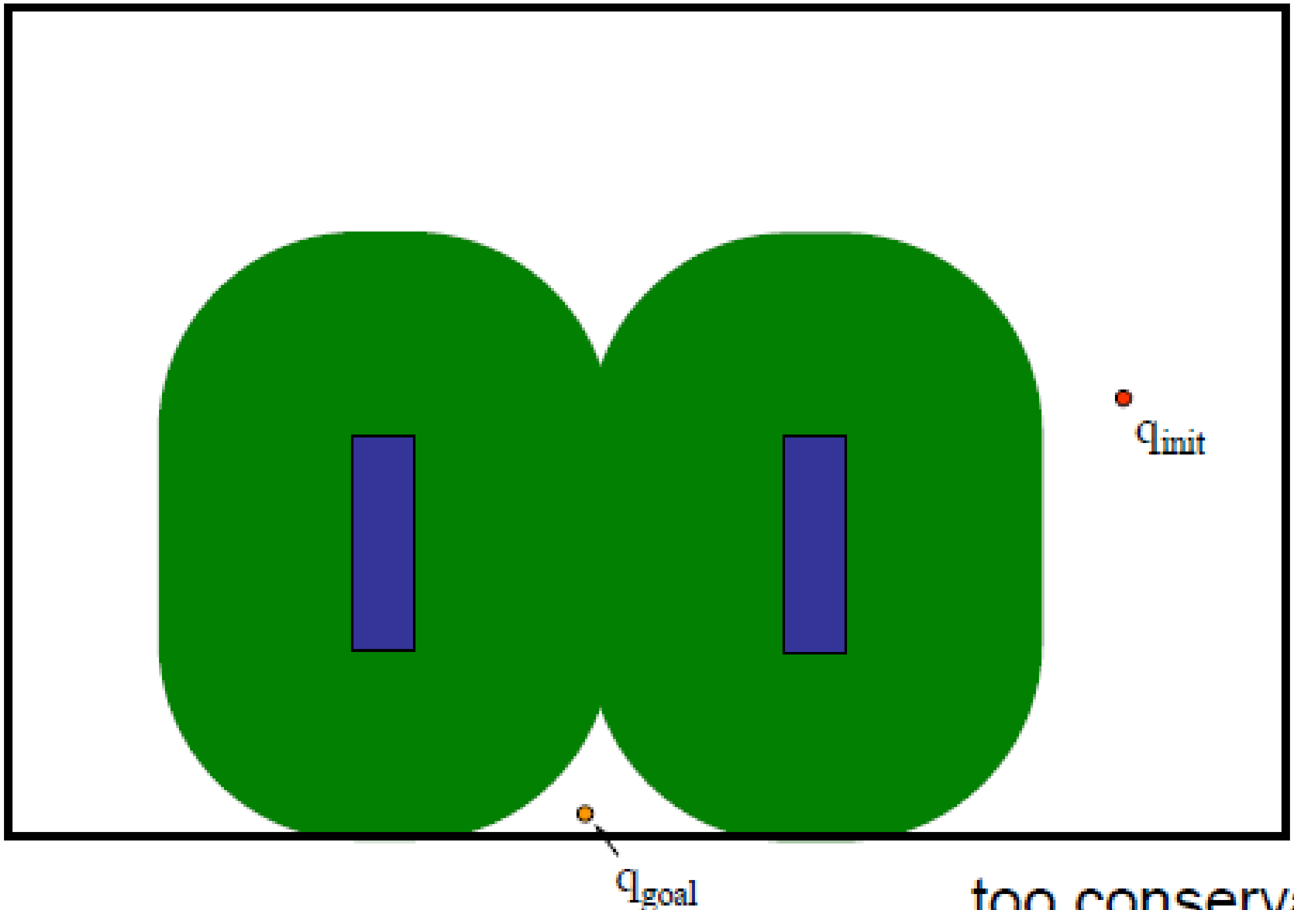
2D-Projection





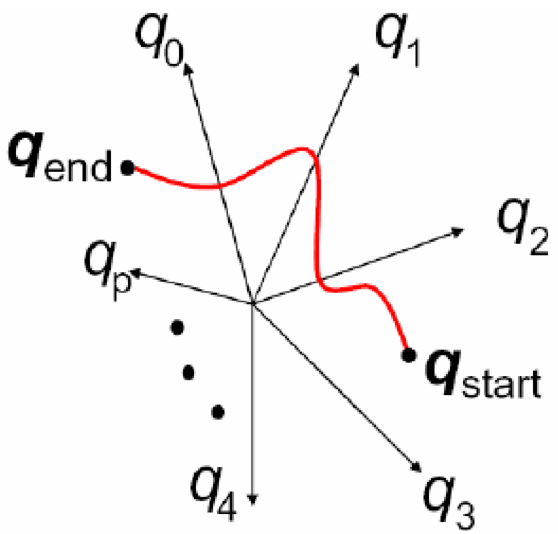
why not keep it this simple?

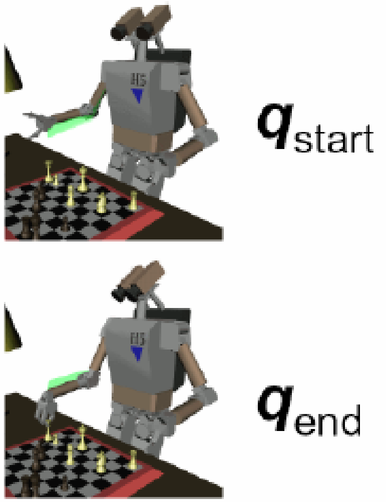
Projection Problems



too conservative!

Motion Planning Problem





- *A* = robot with *p* degrees of freedom in 2-D or 3-D
- *CB* = Set of obstacles
- A configuration $\boldsymbol{q}$ is legal if it does not cause the robot to intersect the obstacles
- Given start and goal configurations ($\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{goal}$), find a continuous sequence of legal configurations from $\boldsymbol{q}_{start}$ to $\boldsymbol{q}_{goal}$.
- Report failure if not path is found

Topics

- Visibility Graphs
  - Roadmaps
  - Voronoi
- Approximate Cell Decomposition

- Potential Fields
- Probabilistic Roadmaps (Sampling)

Freie Universität Berlin

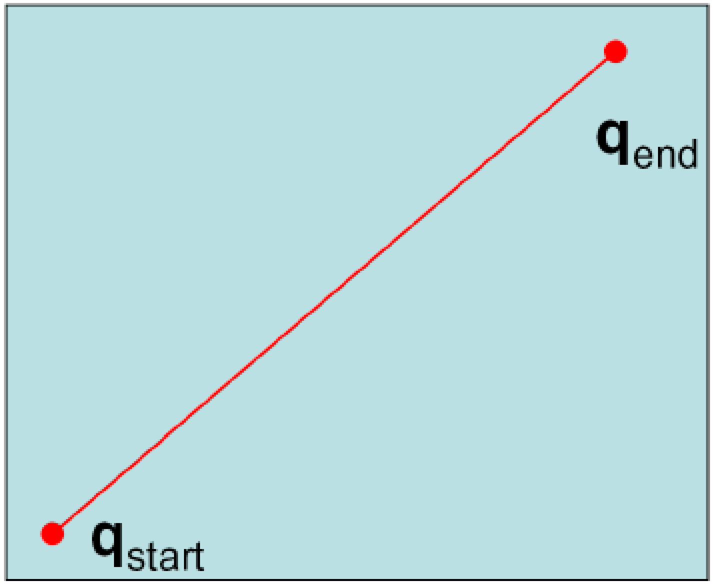
Topics

- Visibility Graphs
  - Roadmaps
  - Voronoi
- Approximate Cell Decomposition
- Potential Fields
- Probabilistic Roadmaps (Sampling)

In all cases:

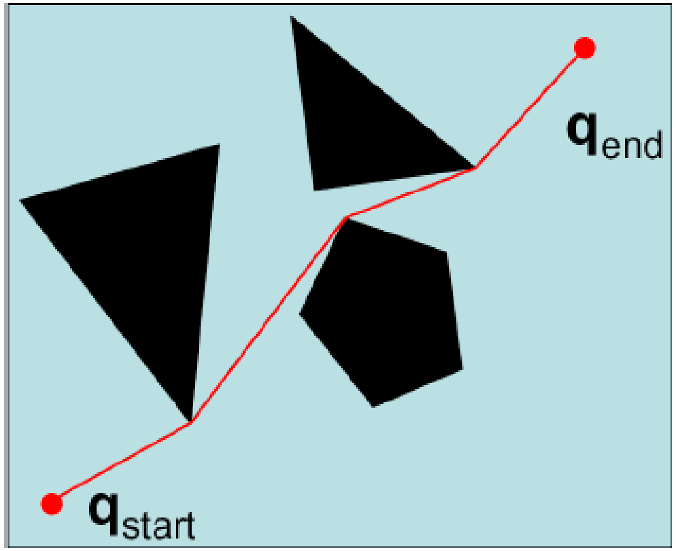
- Reduce the intractable problem in continuous C-space to a tractable problem in a discrete space Use all of the techniques we know (A*, stochastic search, etc.)

Visibility Graphs



In the absence of obstacles, the best path is the straight line between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$

Visibility Graphs



- Assuming polygonal obstacles: It looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles.
- Is this always true?

Roadmap Approaches, Visibility Graphs

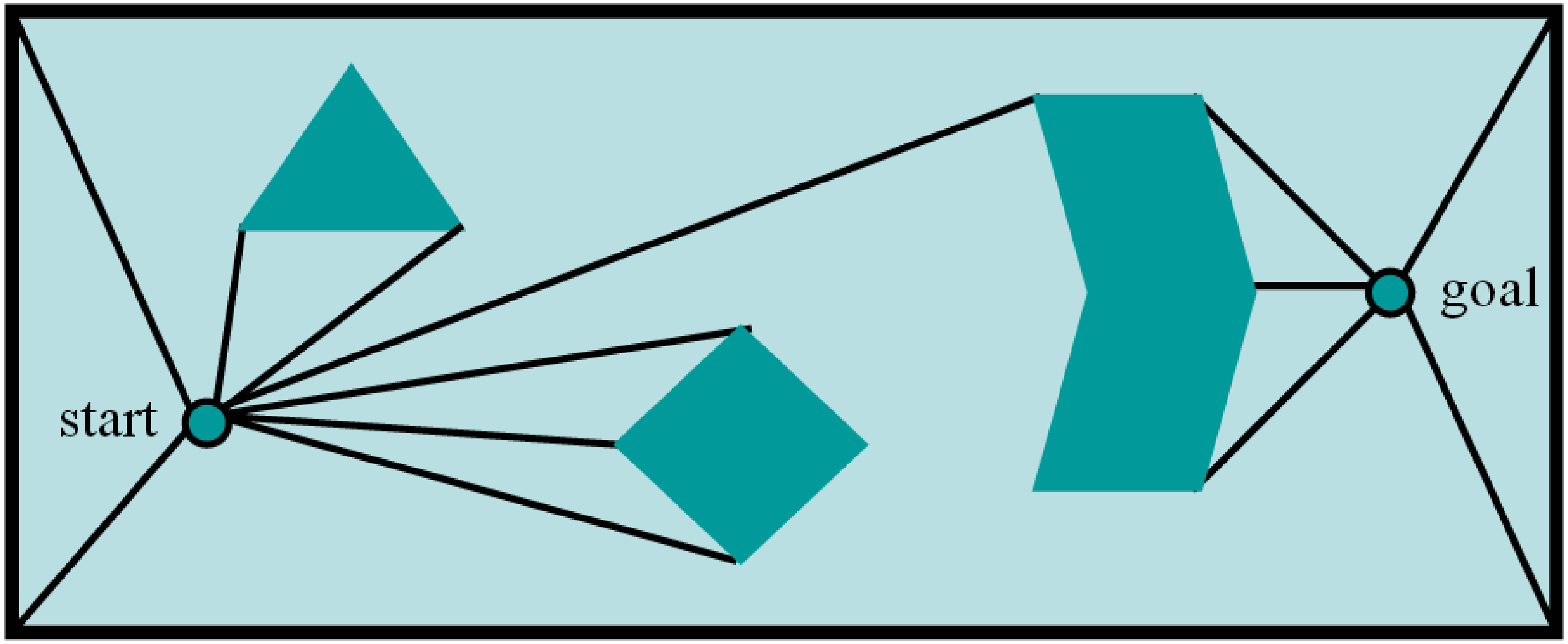
- used to find Euclidean shortest paths among a set of polygonal obstacles in the plane

- construct all of the line segments that connect vertices to one another (and that do not intersect the obstacles themselves)

- Formed by connecting all “visible” vertices, the start point and the end point, to each other
- For two points to be “visible” no obstacle can exist between them
- Paths exist on the perimeter of obstacles
- a graph is defined, converts the problem into graph search.
- Dijkstra’s algorithm, O(N^2), N = the number of vertices in C-space

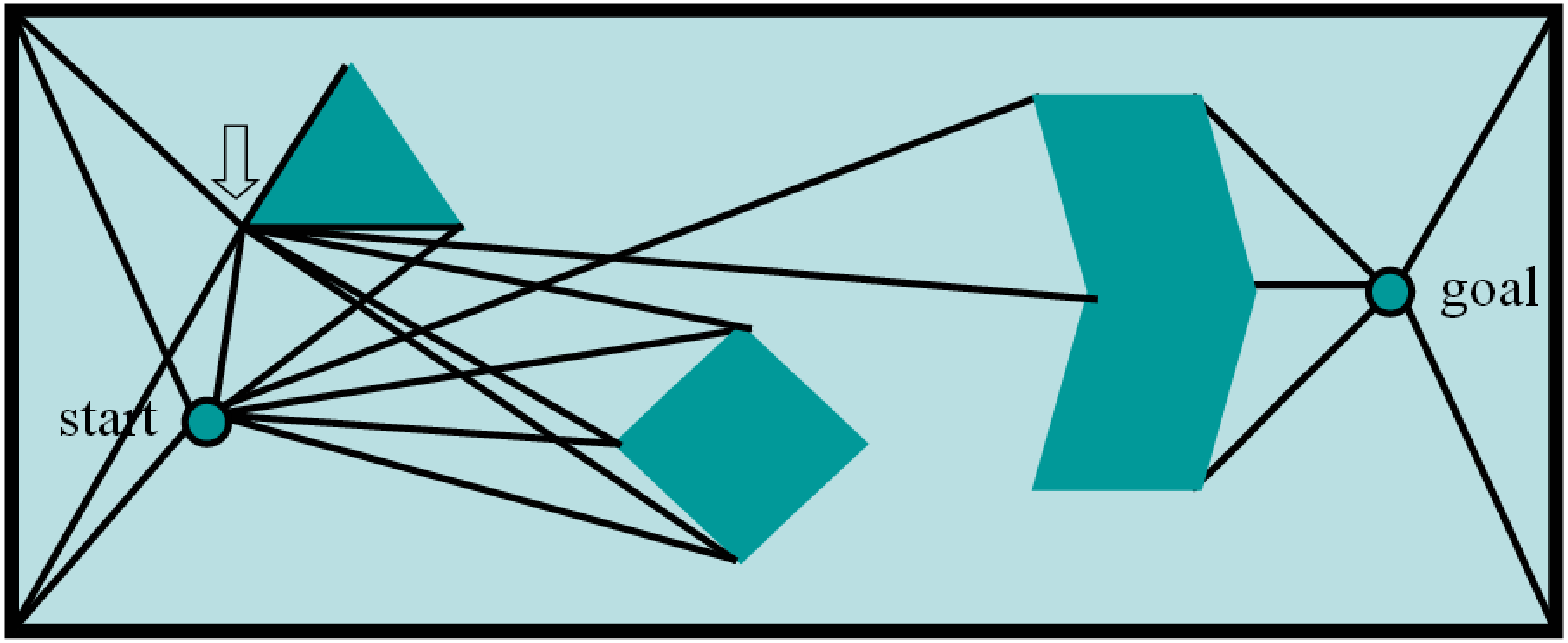
The Visibility Graph in Action (1)

- First, draw lines of sight from the start and goal to all "visible" vertices and corners of the world.



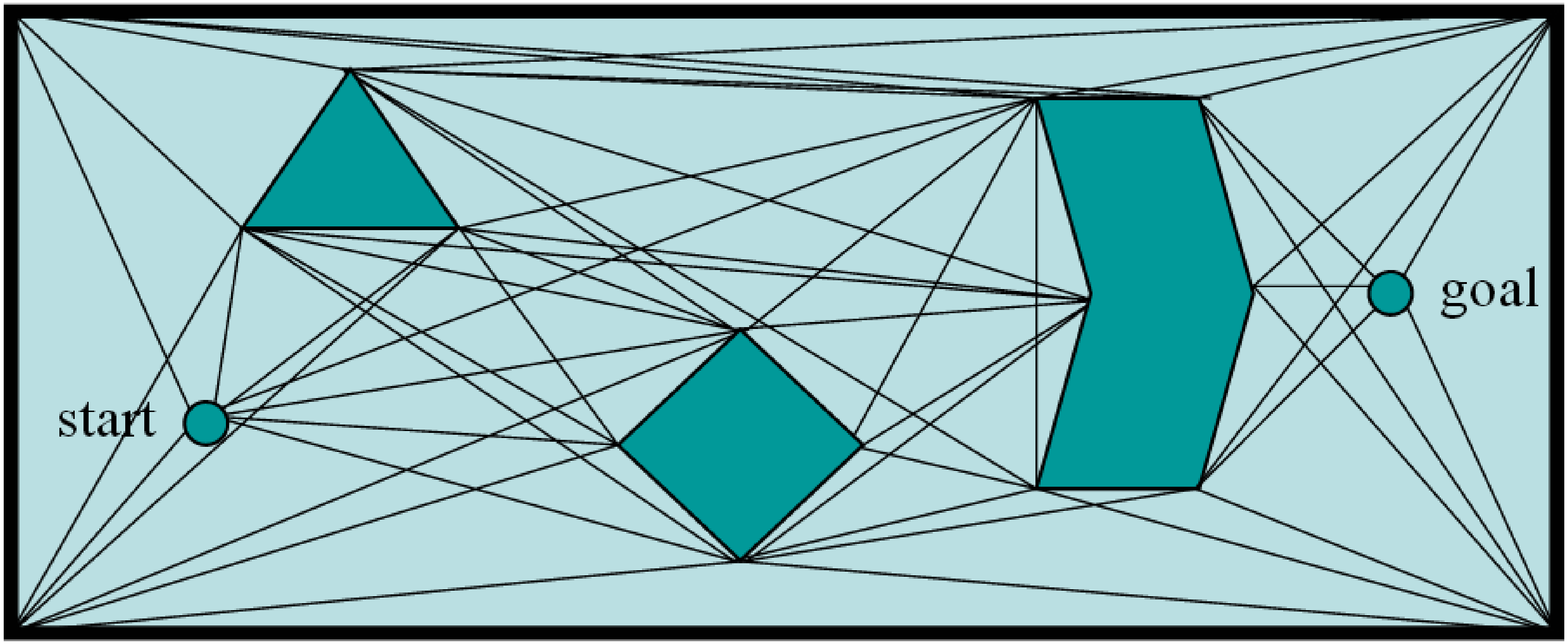
The Visibility Graph in Action (2)

- Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.



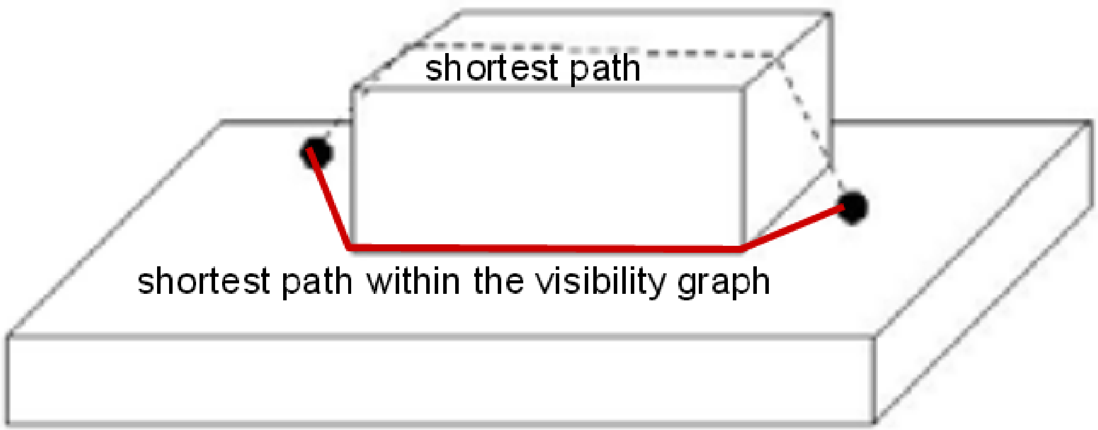
The Visibility Graph in Action (3)

- Repeat until done



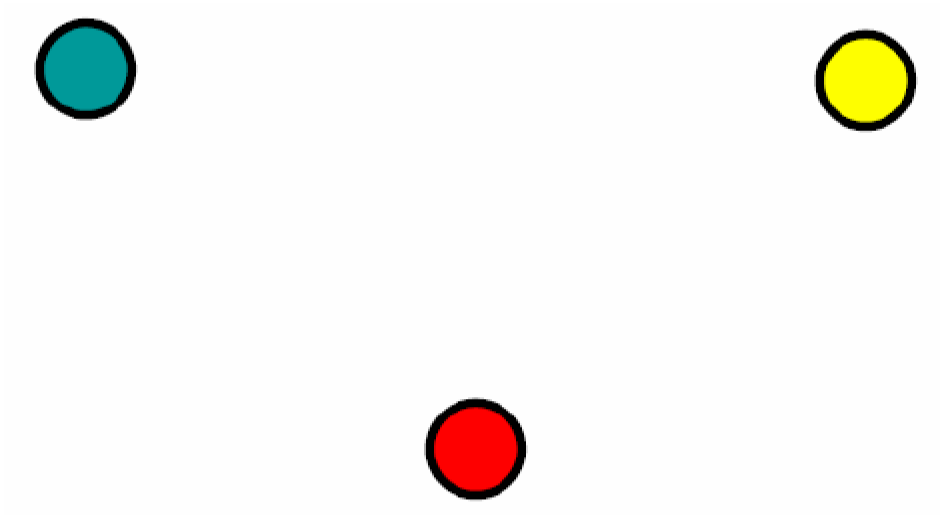
Optimality

- Visibility graphs do not preserve their optimality in higher dimensions



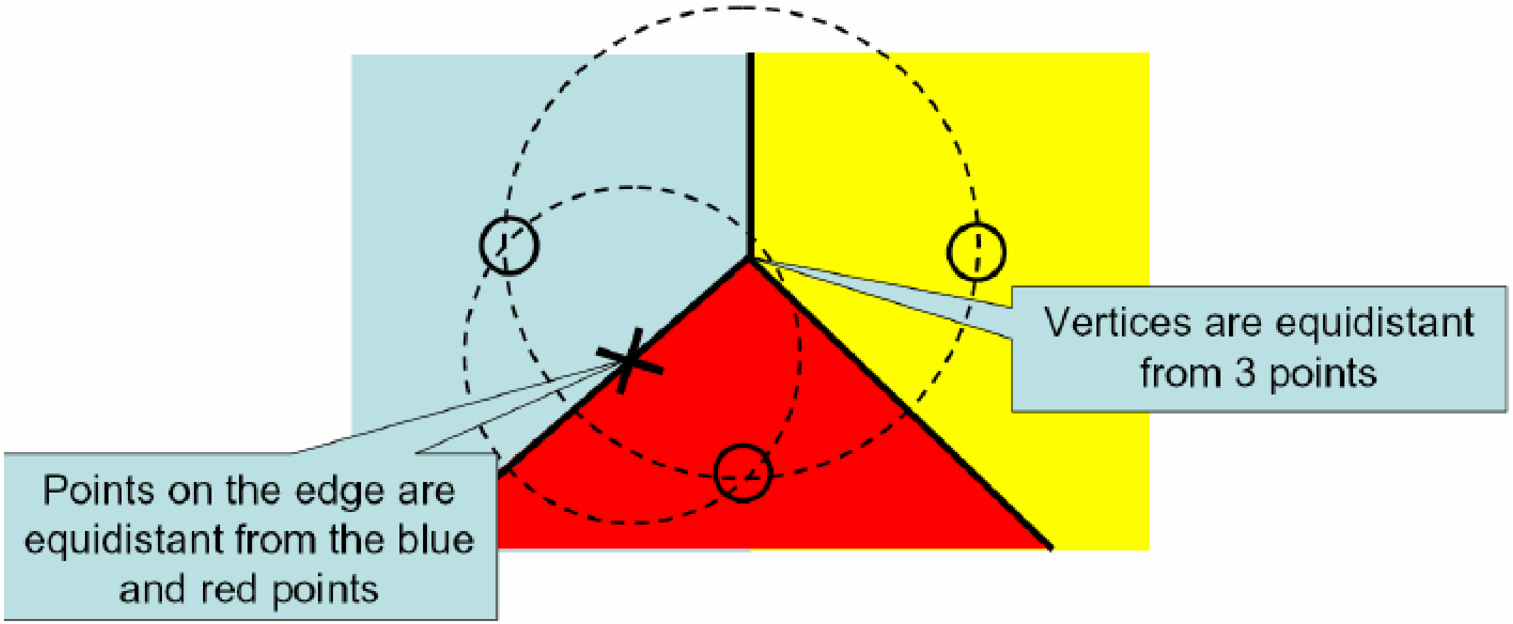
- In addition, the paths they find are “semi-free,” i.e. in contact with obstacles.
- No clearance, any execution error will lead to a collision
- Need other types of roadmaps for safer paths

Voronoi Diagrams



- Given a set of data points in the plane:
  – Color the entire plane such that the color of any point in the plane is the same as the color of its nearest neighbor

Voronoi Diagrams

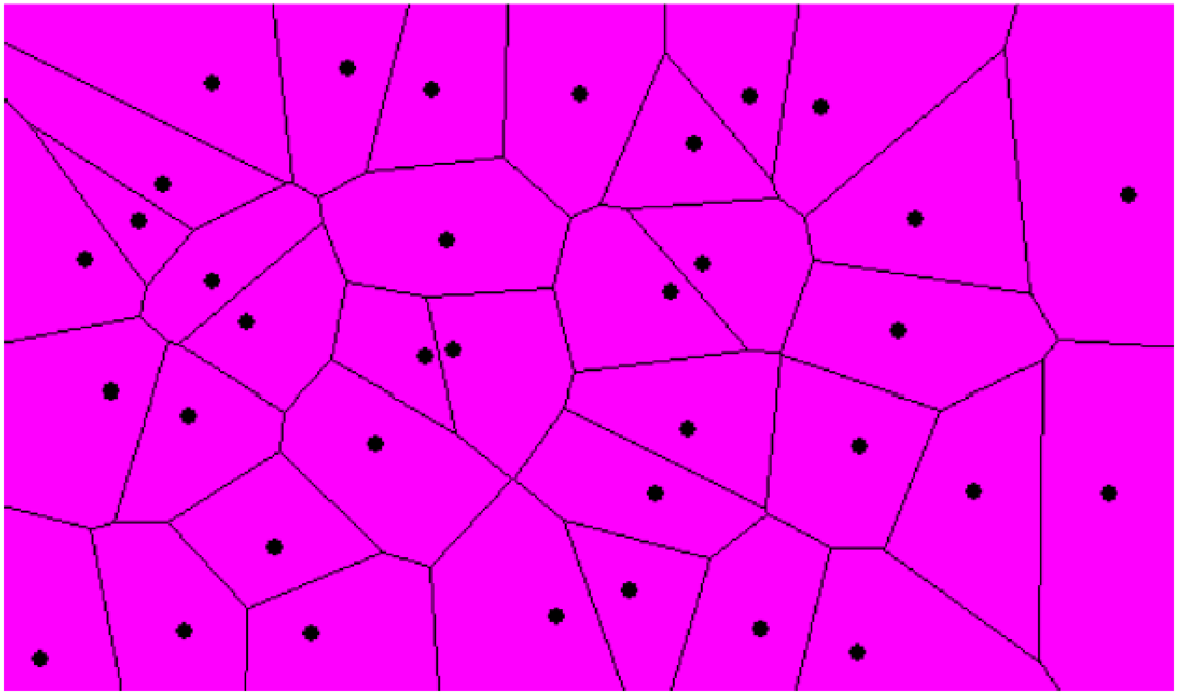


- Voronoi diagram = The set of line segments separating the regions corresponding to different colors

  - Line segment = points equidistant from 2 data points

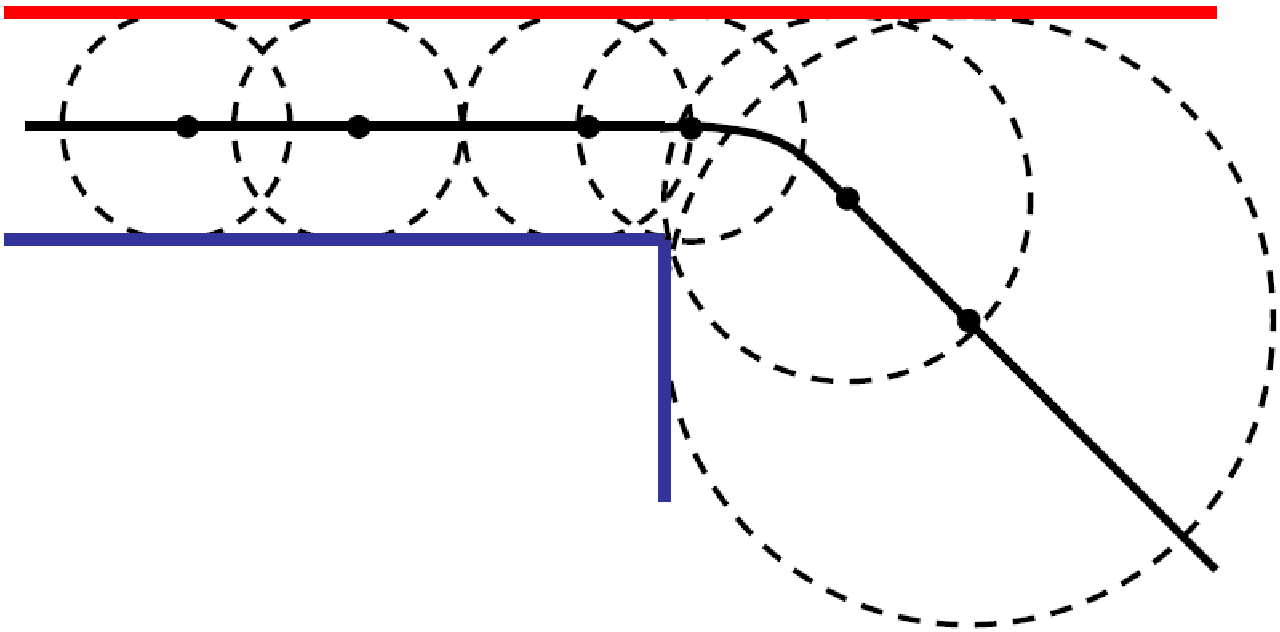
  - Vertices = points equidistant from > 2 data points

Voronoi Diagrams



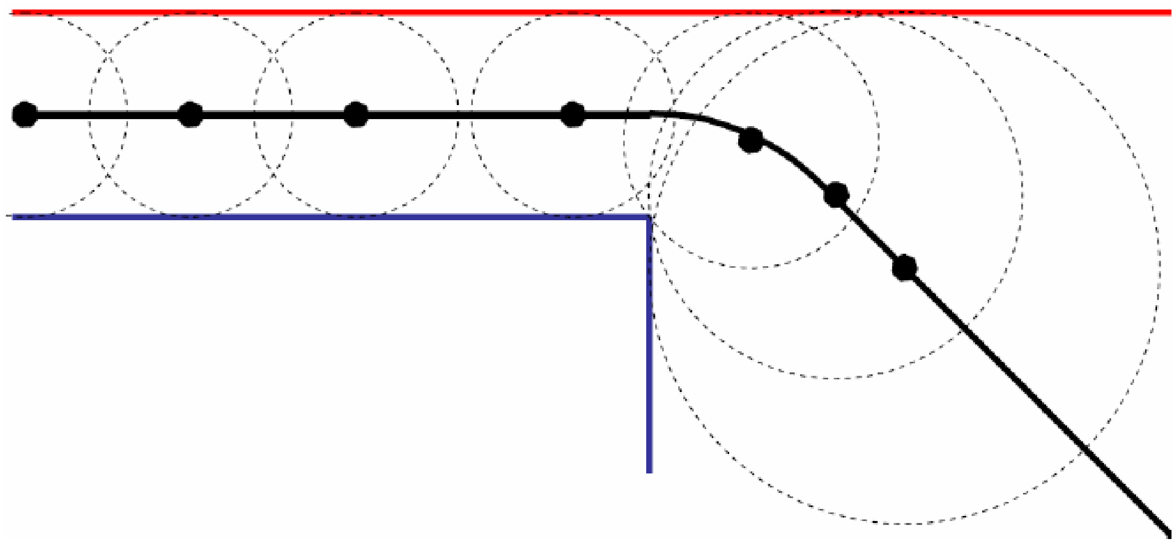
- Complexity (in the plane):
- O(*N* log *N*) time
- O(*N*) space

Voronoi Diagram



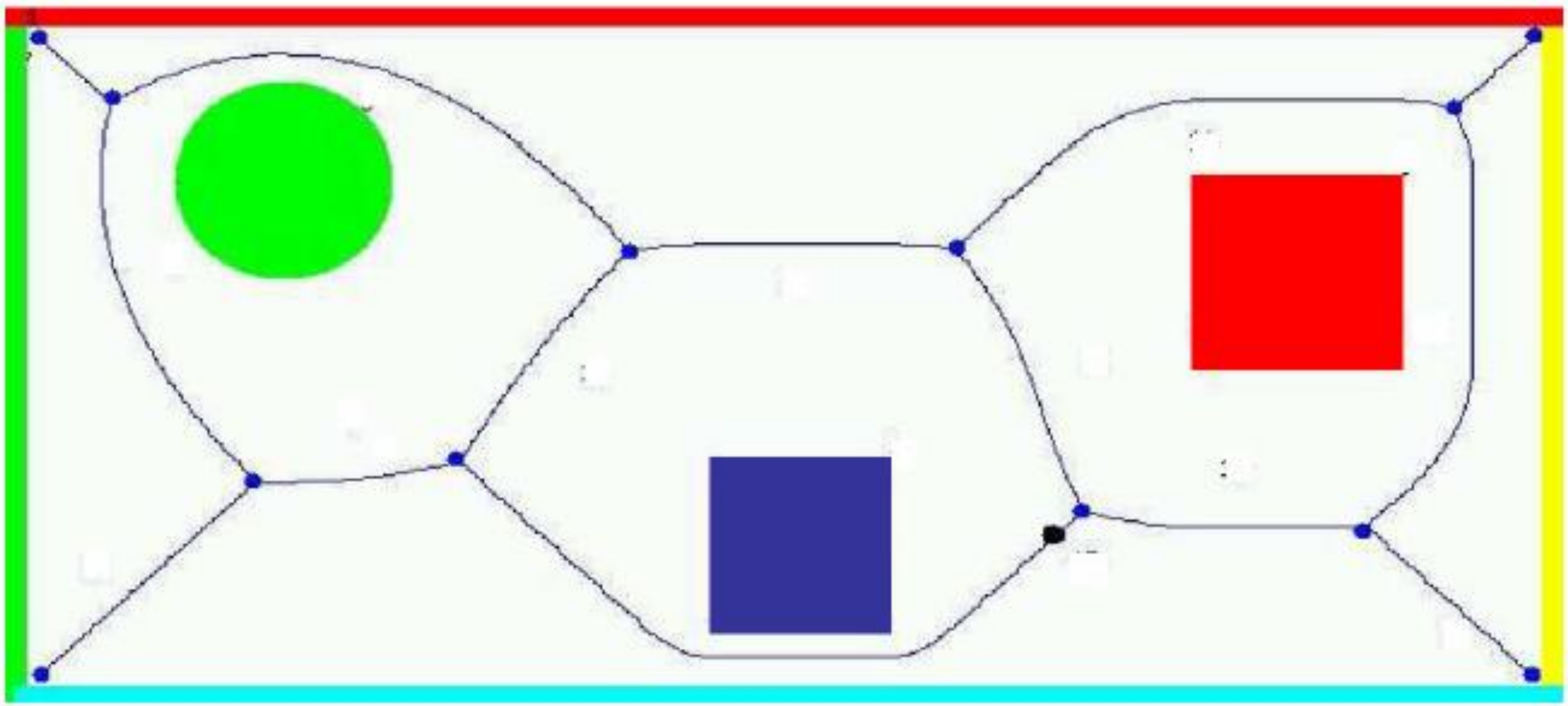
Freie Universität Berlin

Voronoi Diagrams: Beyond Points



- Edges are combinations of straight line segments and segments of quadratic curves
- Straight edges: Points equidistant from 2 lines
- Curved edges: Points equidistant from one corner and one line

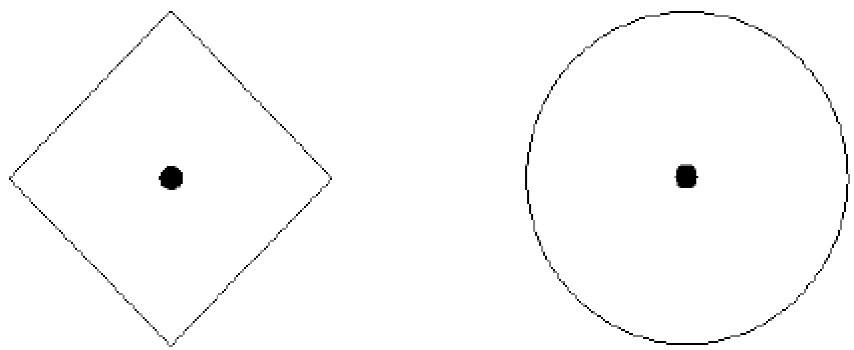
Voronoi Diagrams (Polygons)



Freie Universität Berlin

Voronoi Diagrams: Metric

- Many ways to measure distance; two are:
  - L_1 metric
    - (x,y) : |x| + |y| = const
  - L_2 metric
    - (x,y) : $x^2 + y^2$ = const



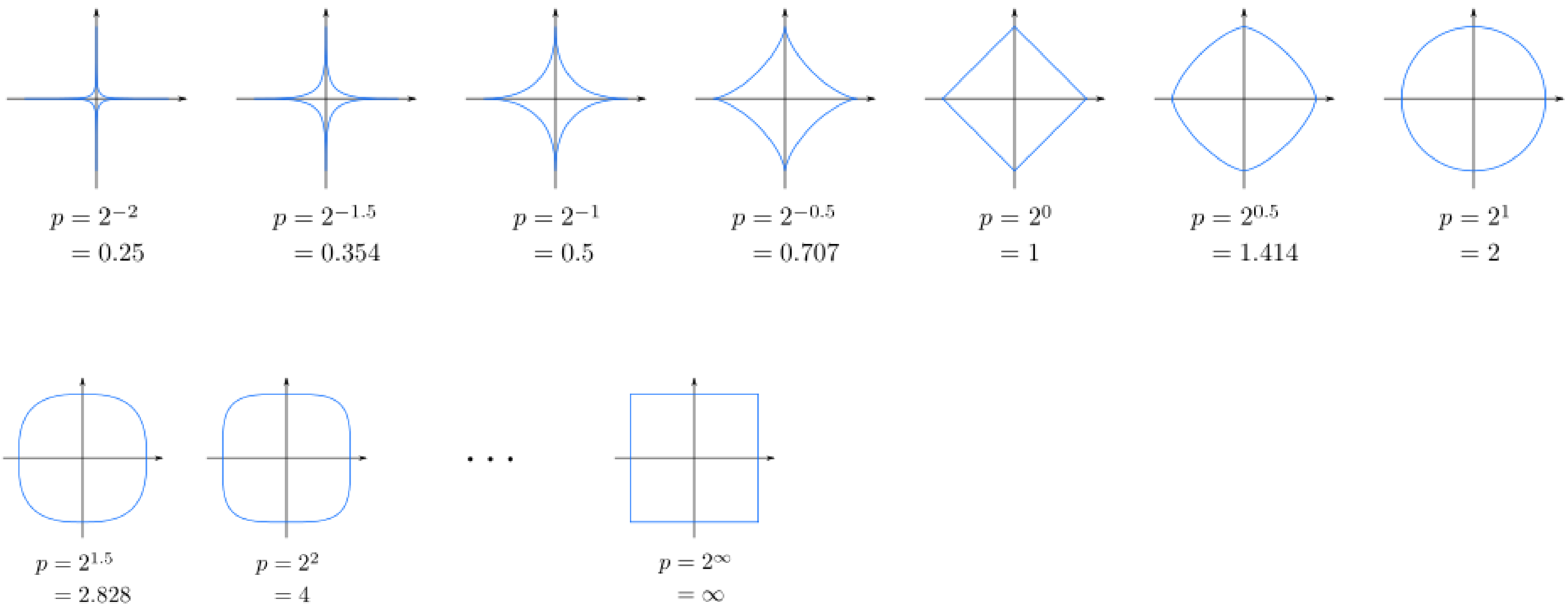
$\{(x,y) : |x| + |y| = \text{const}\}$ $\{(x,y) : x^2 + y^2 = \text{const}\}$

L1 L2

Minkowski Metric

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

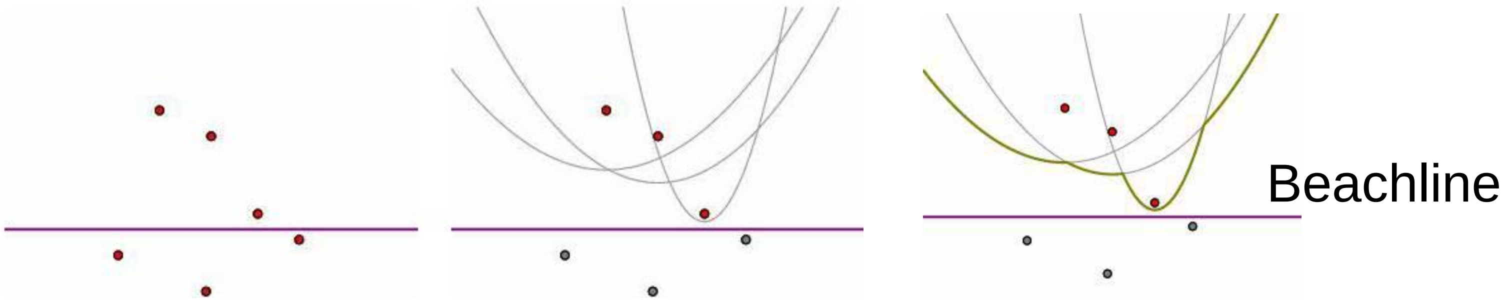
Source: Wikipedia



Voronoi Diagram: Construction (Roughly)

- For a Voronoi-Diagram in 2D: there are many options, e.g.: Fortune's Algorithm (a form of Sweepline Algorithm)

- Using O(n log n) time and
- O(n) space

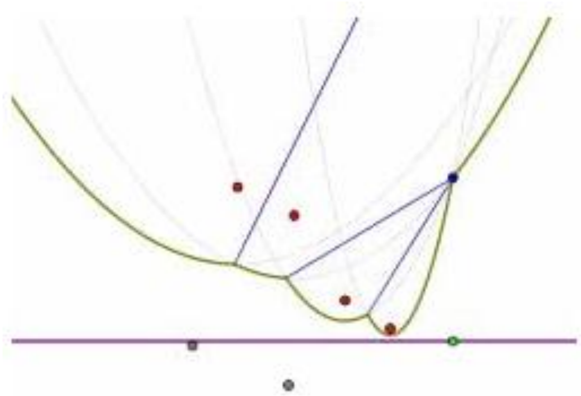


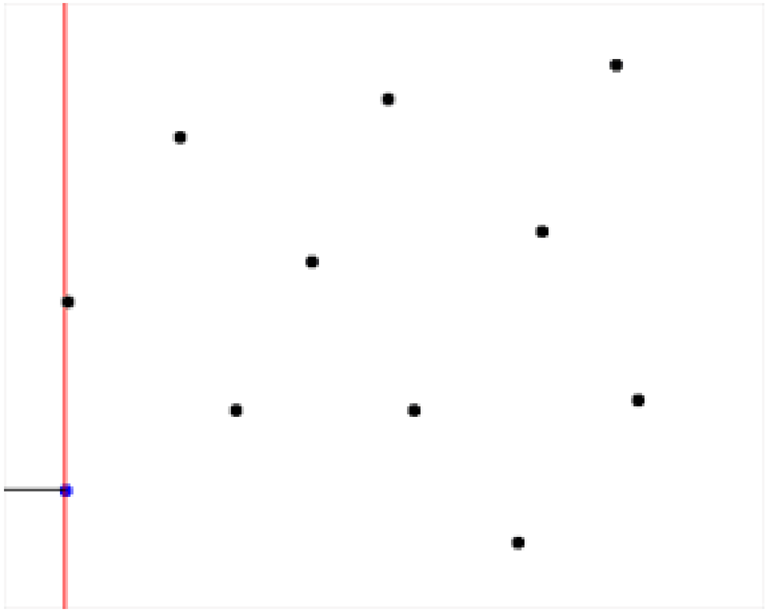
- Find intersections of 2 parabolas (breakpoints) and intersection points of three parabolas while sweeping down

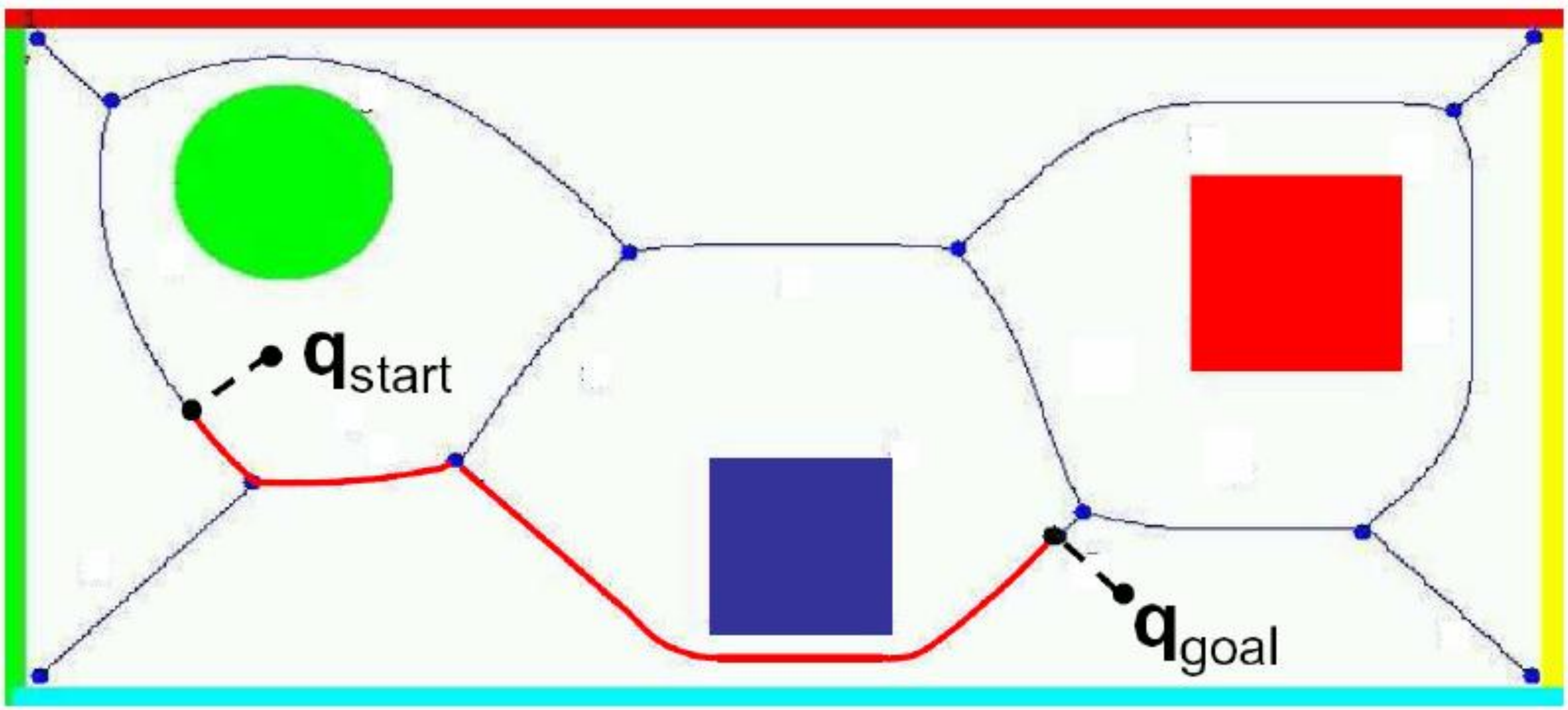
Voronoi Diagram: Construction (Roughly)

- For a Voronoi-Diagram in 2D: there are many options, e.g.: Fortune‘s Algorithm (a form of Sweepline Algorithm)

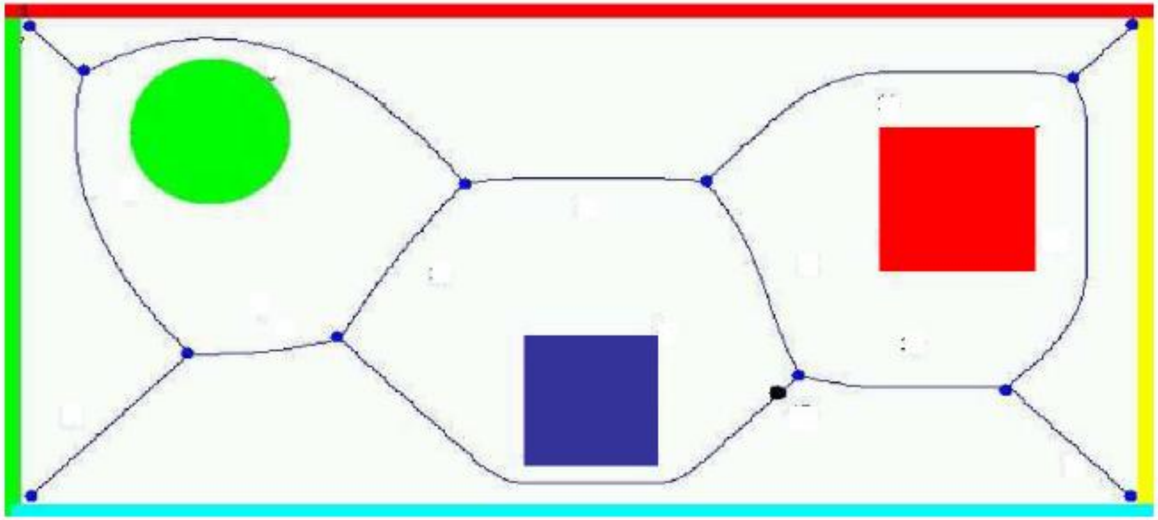
- Using O(n log n) time and
- O(n) space





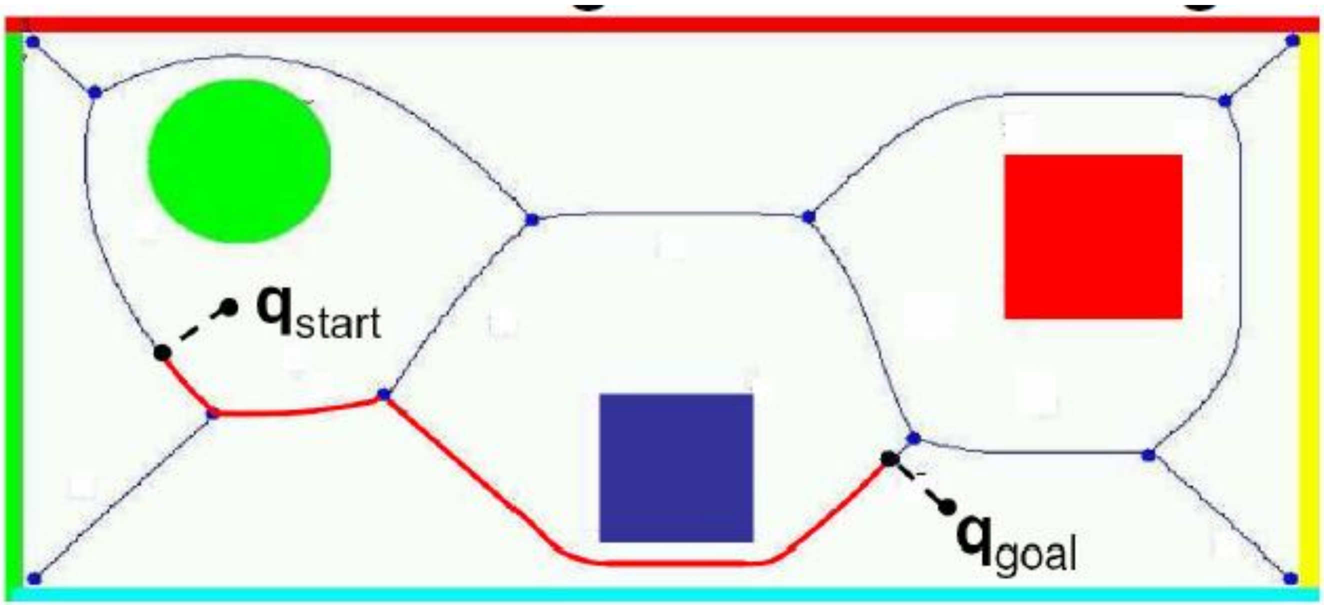

q_{start}
q_{goal}

Voronoi Diagram (Polygons)



- Key property: The points on the edges of the Voronoi diagram are the *furthest from the obstacles*
- Idea: Construct a path between $\mathbf{q_{start}}$ **and** $\mathbf{q_{goal}}$ **by** following edges on the Voronoi diagram
- (Use the Voronoi diagram as a roadmap graph instead of the visibility graph)

Voronoi-Diagram (Planning)



- Find the point **q^*_{start} of the Voronoi** diagram closest to **q_{start}**
- Find the point **q^*_{goal} of the Voronoi** diagram closest to **q_{goal}**
- Compute shortest path from **q^*_{start} to q^*_{goal} on the Voronoi diagram**

Voronoi: Weaknesses

- Difficult to compute in higher dimensions or nonpolygonal worlds
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic (“stay away from obstacles”) can lead to paths that are much too conservative
- Can be unstable → Small changes in obstacle configuration can lead to large changes in the diagram

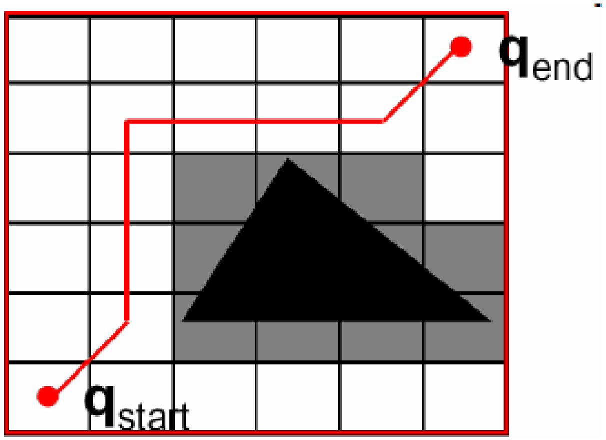
Topics

- Visibility Graphs
  - Roadmaps
  - Voronoi
- Approximate Cell Decomposition

Decompose the space into cells so that any path inside a cell is obstacle free

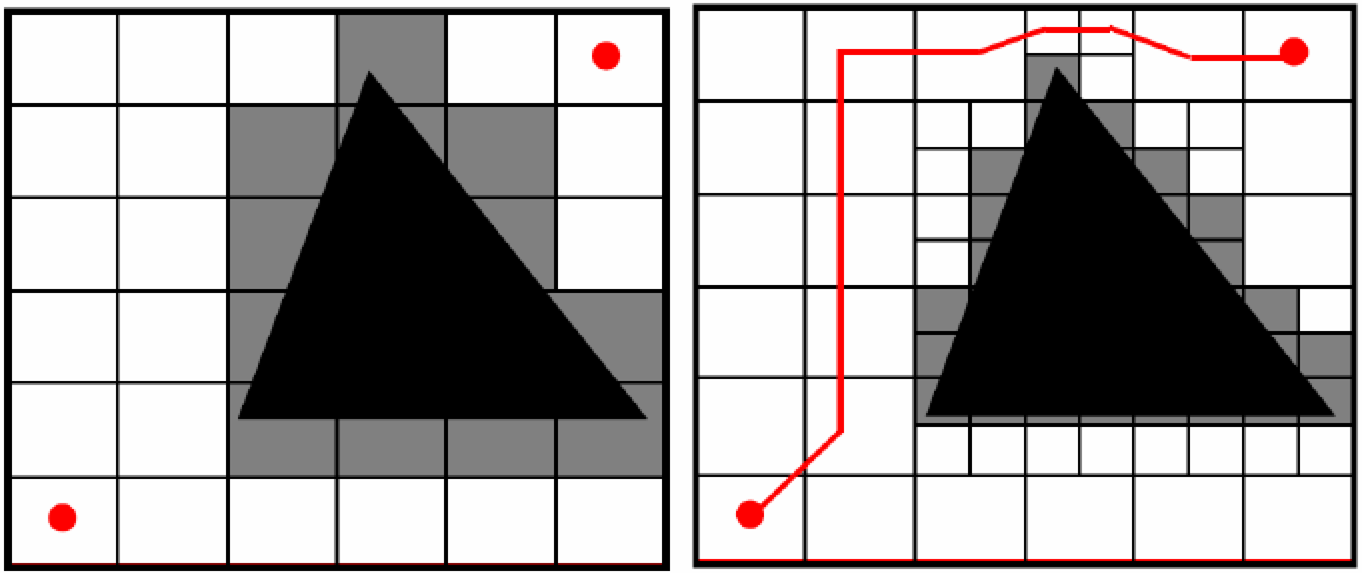
- Potential Fields
- Probabilistic Roadmaps (Sampling)
- Bug algorithms

Approximate Cell Decomposition



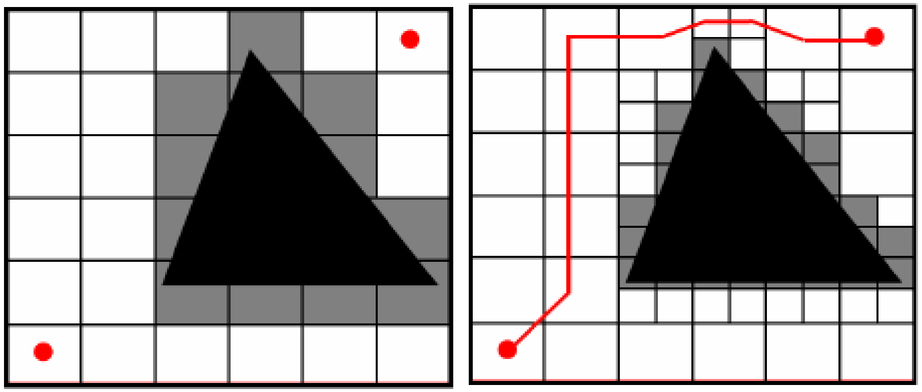
- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects C_{obs} as blocked
- Find path through remaining cells by using (for example) A* (e.g., use Euclidean distance as heuristic)
- Cannot be *complete as described so far. Why?*

Approximate Cell Decomposition



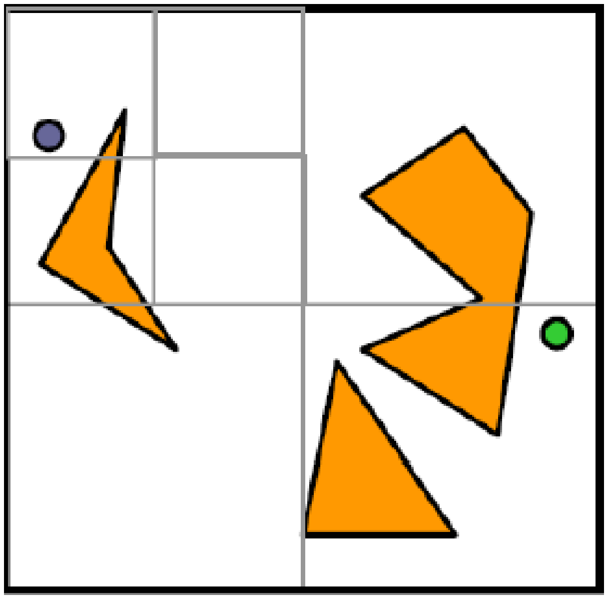
Freie Universität Berlin

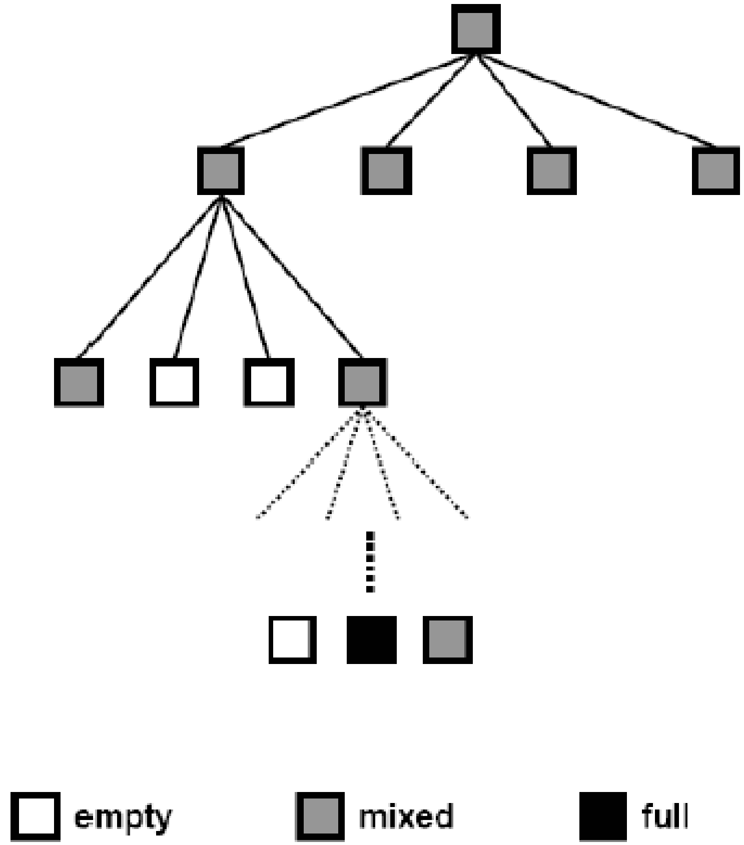
Approximate Cell Decomposition



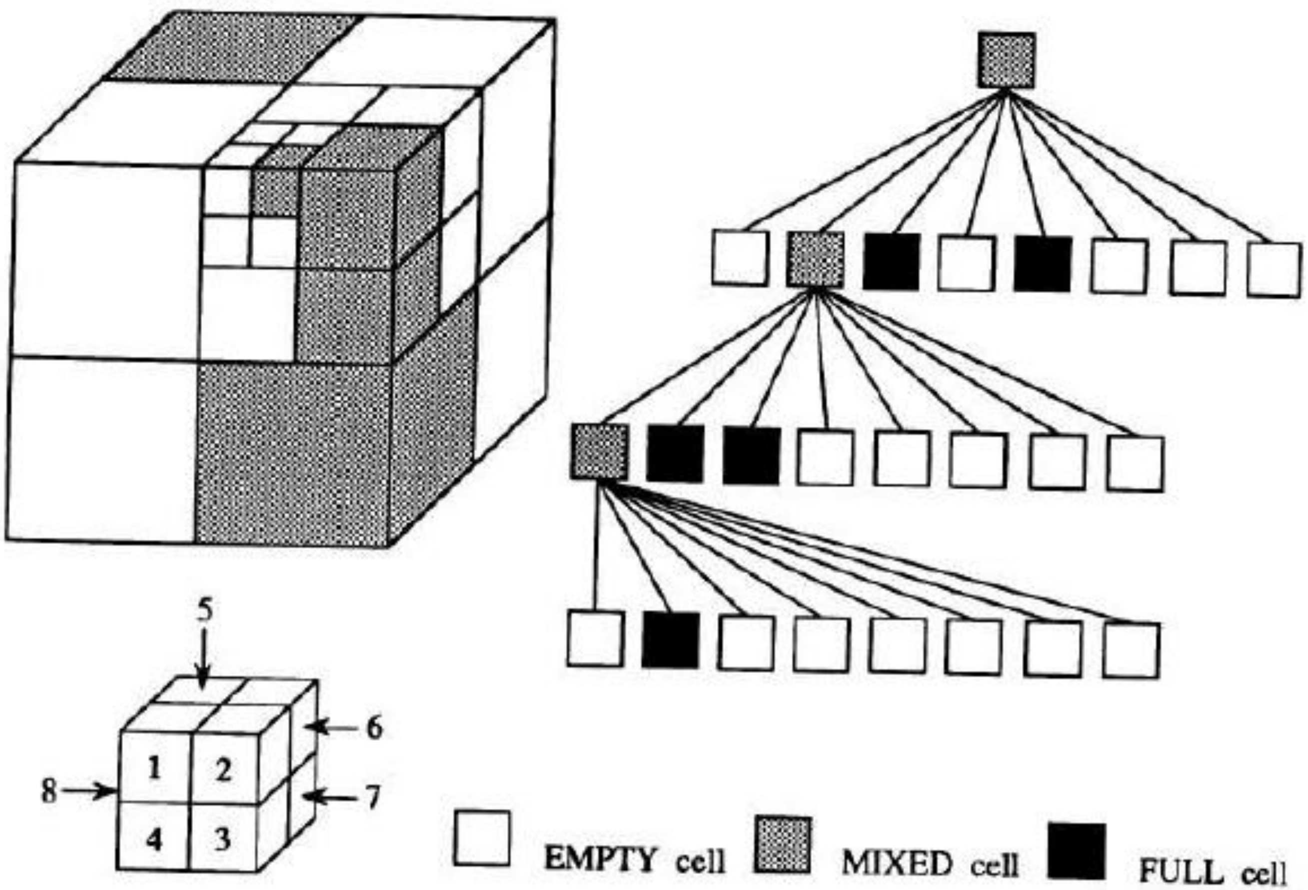
- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
  - Cells that are entirely contained in C_{obs} (*FULL) and*
  - Cells that partially intersect C_{obs} (*MIXED)*
- Try to find a path using the current set of cells
- If no path found:
  - Subdivide the *MIXED cells and try again with the new set of* cells

Quadtree Decomposition

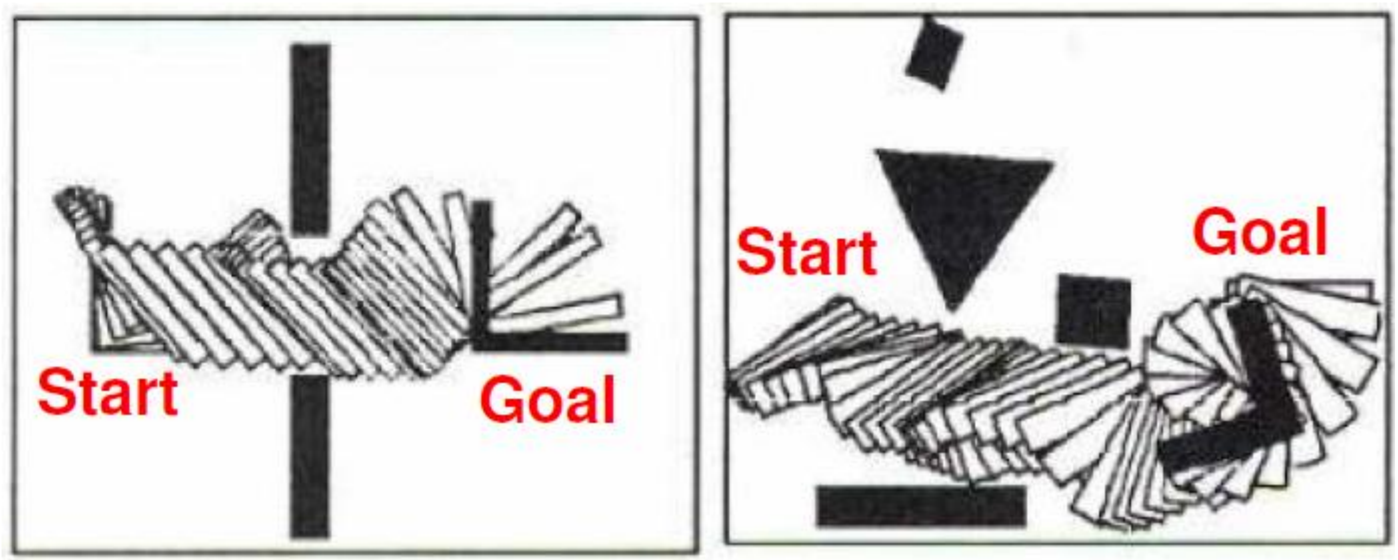




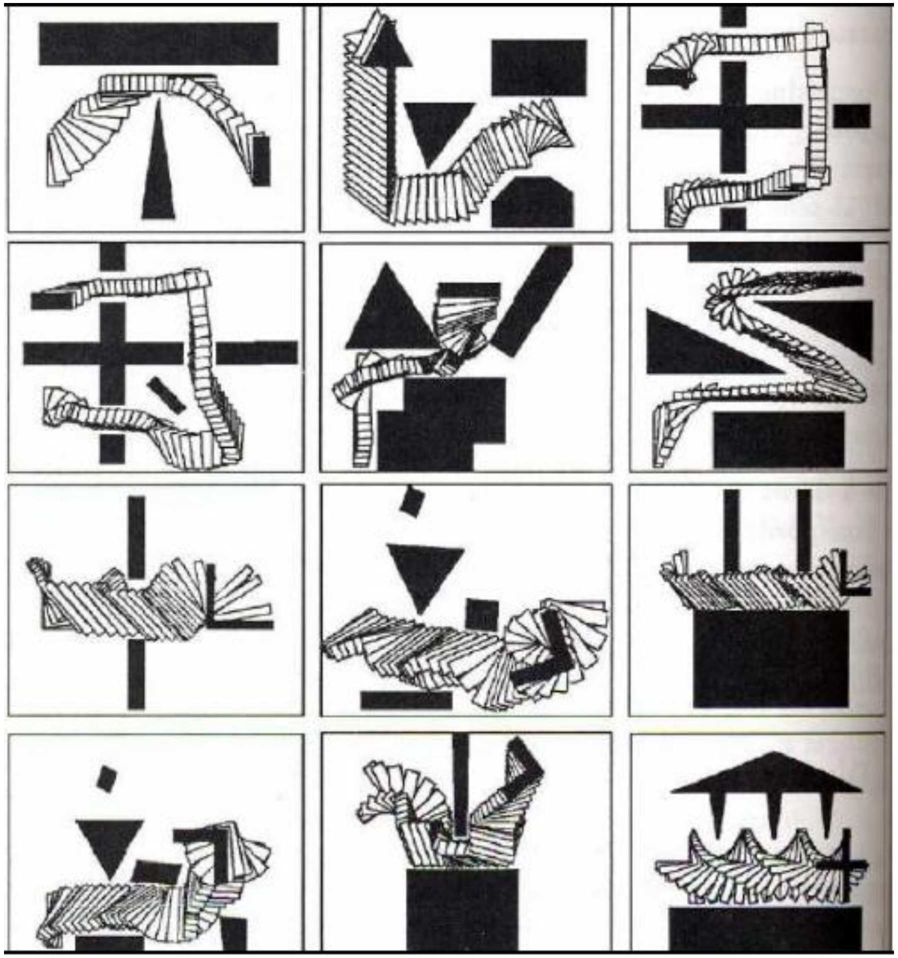
Octree Decomposition



Example



Examples

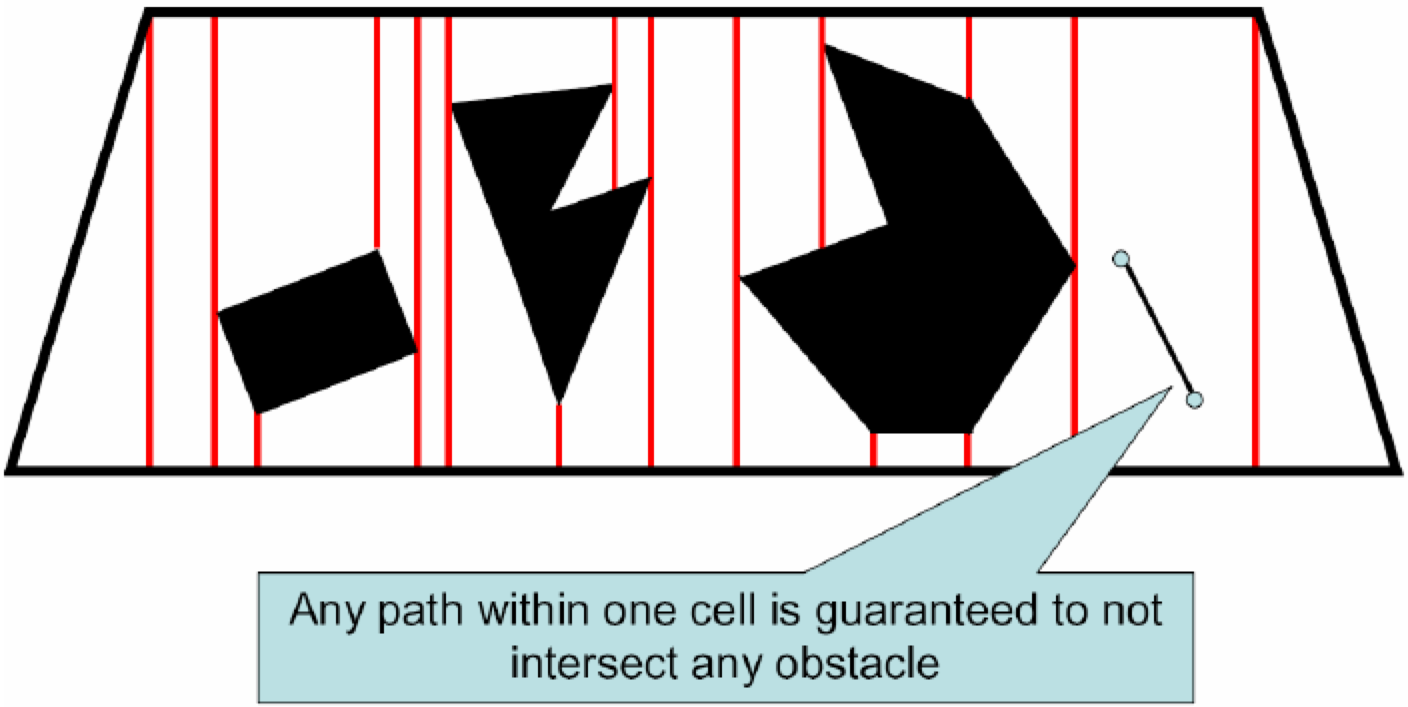


Approximate Cell Decomposition: Limitations

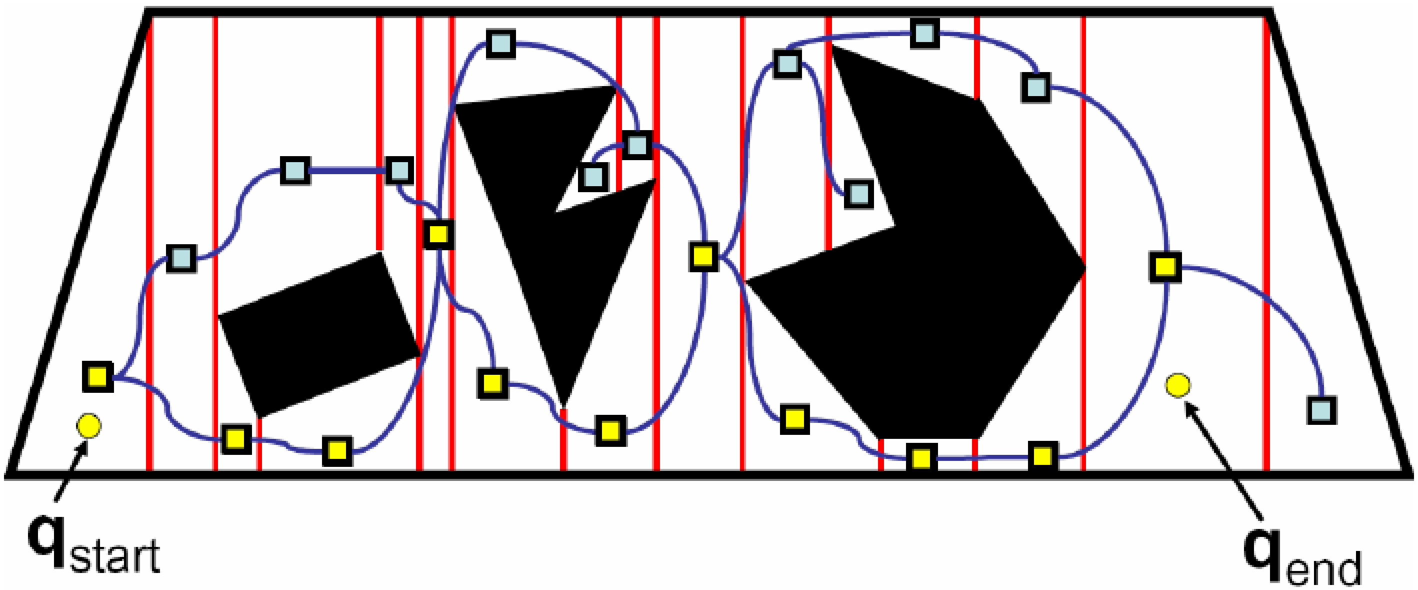
- Good:
  - Limited assumptions on obstacle configuration
  - Approach used in practice
  - Find obvious solutions quickly

- Bad:
  - No clear notion of optimality (“best” path)
  - Trade-off completeness/computation
  - Still difficult to use in high dimensions

Exact Cell Decomposition

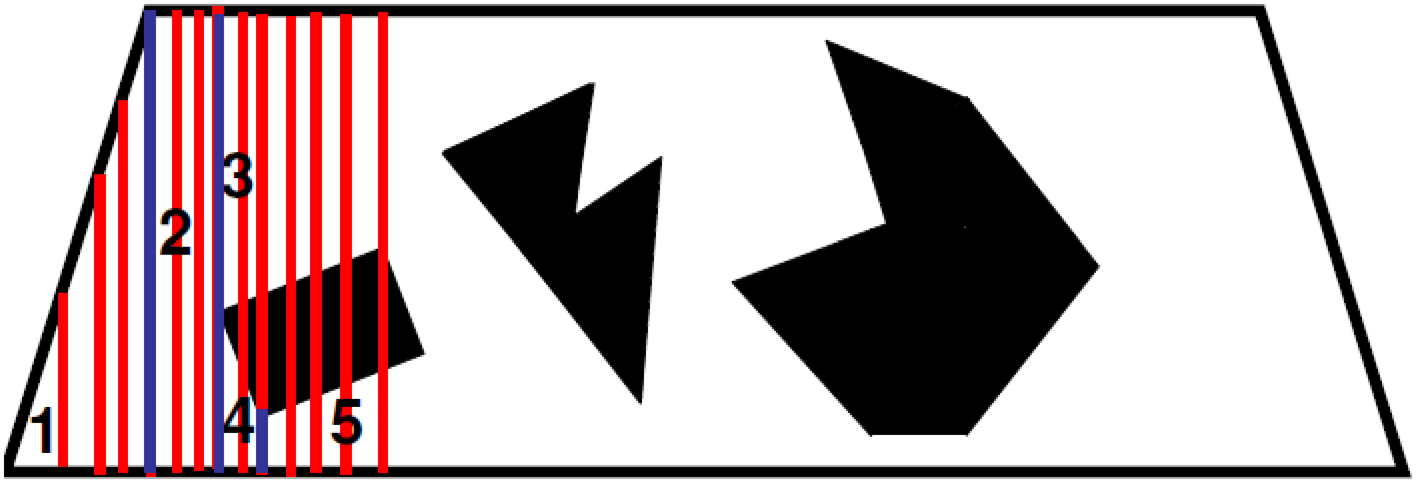


Exact Cell Decomposition



- The graph of cells defines a roadmap
- The graph can be used to find a path between any two configurations

Exact Cell Decomposition

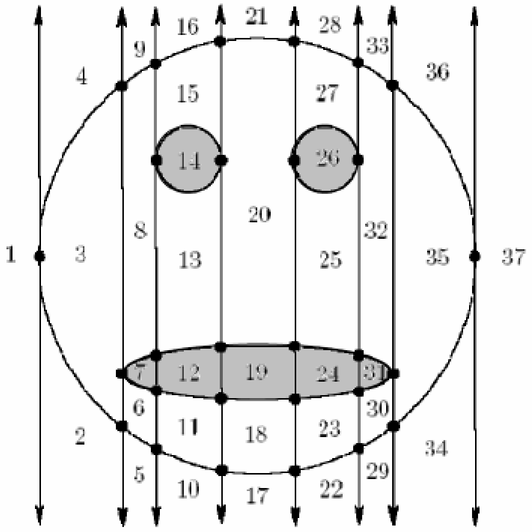


- Critical event: Create new cell
- Critical event: Split cell

Plane Sweep Algorithm

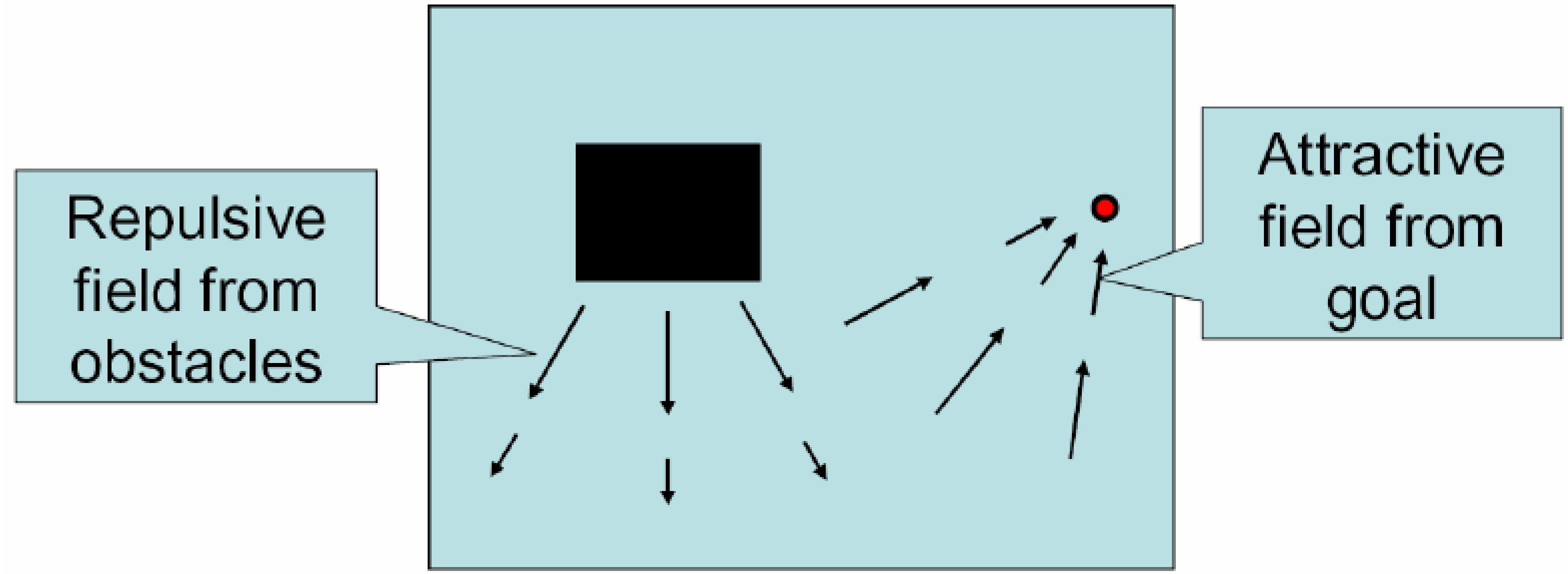
- Initialize current list of cells to empty
- Order the vertices of C_{obs} along the *x direction*
- For every vertex:
  - Construct the plane at the corresponding *x location*
  - Depending on the type of event:
    - Split a current cell into 2 new cells OR
    - Merge two of the current cells
  - Create a new cell

Exact Cell Decomposition



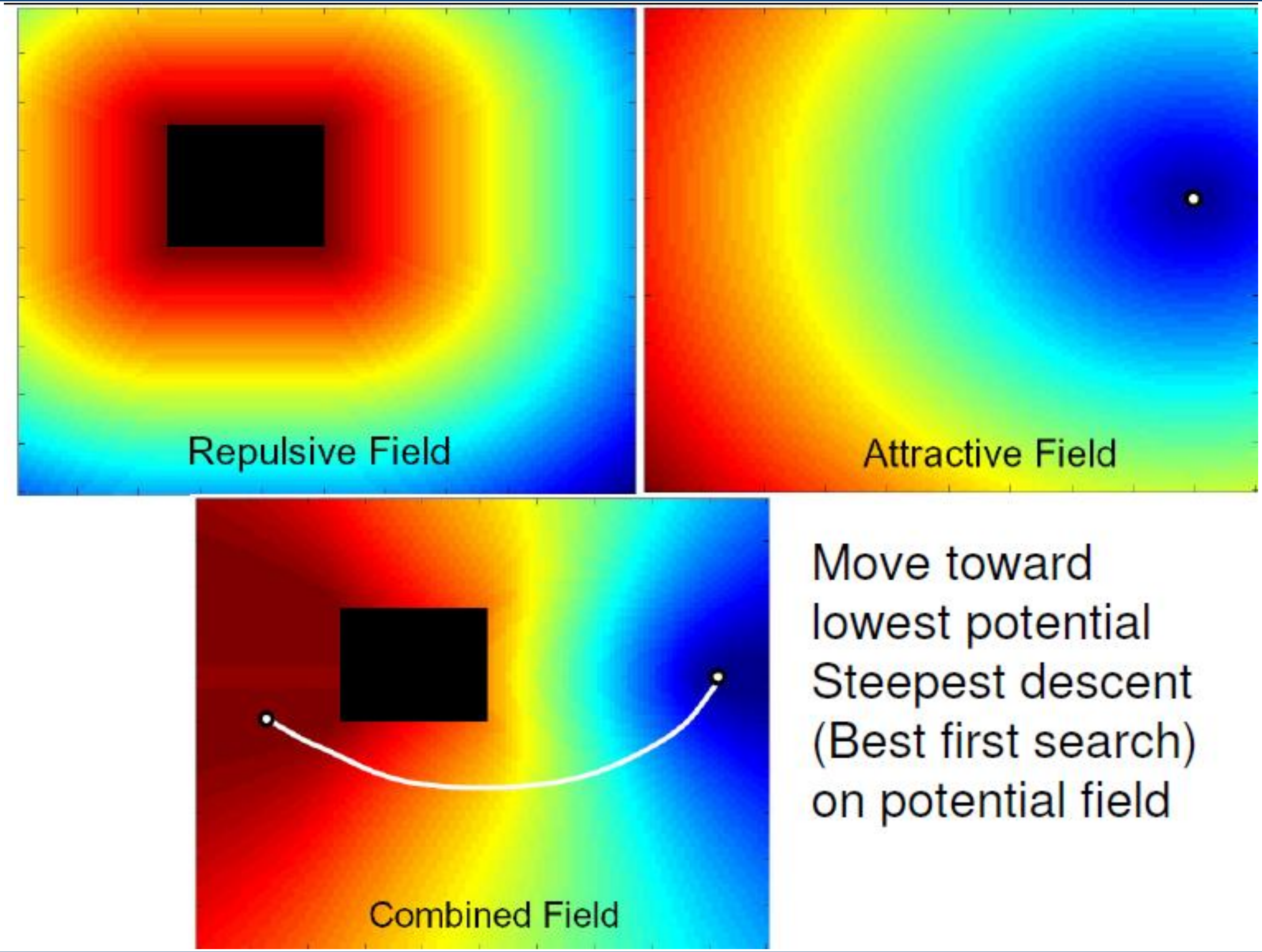
- A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries ("cylindrical cell decomposition")
- Provides exact solution completeness
- Expensive and difficult to implement in higher dimensions

Potential Fields



- Stay away from obstacles: Imagine that the obstacles are made of a material that generate a *repulsive field*
- Move closer to the goal: Imagine that the goal location is a particle that generates an *attractive* field

Potential Fields



Move toward lowest potential
Steepest descent (Best first search) on potential field

Freie Universität Berlin

Representation Potential Fields

$$U_g(\mathbf{q}) = d^2(\mathbf{q}, \mathbf{q}_{goal})$$

Distance to goal state

$$U_o(\mathbf{q}) = \frac{1}{d^2(\mathbf{q}, Obstacles)}$$

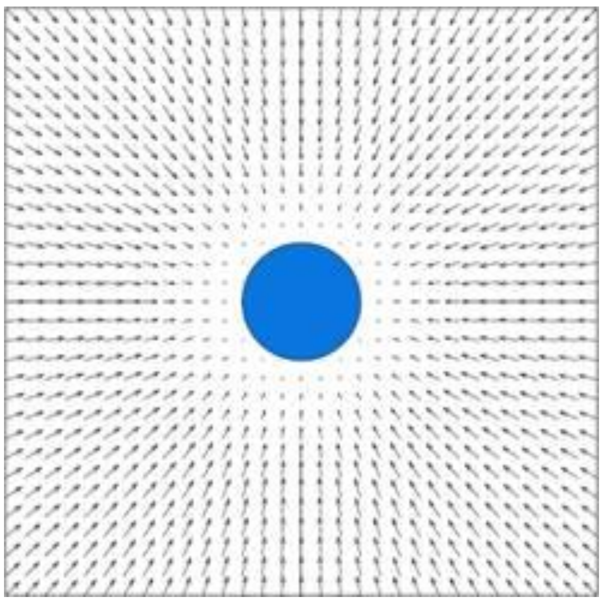
Distance to nearest obstacle point.
Note: Can be computed efficiently by using the *distance transform*

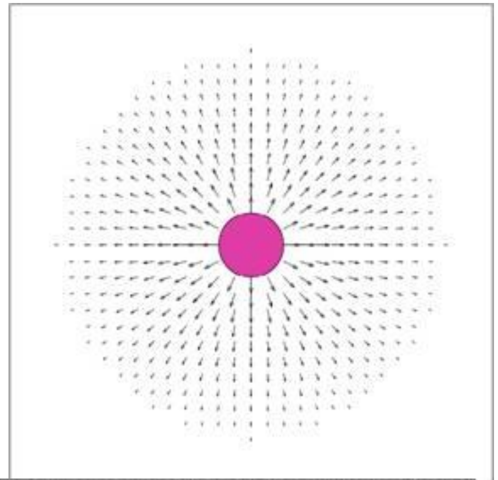
$$U(\mathbf{q}) = U_g(\mathbf{q}) + \lambda U_o(\mathbf{q})$$

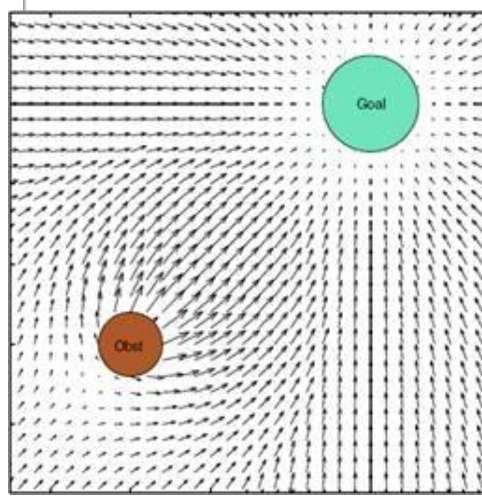
λ controls how far we stay from the obstacles

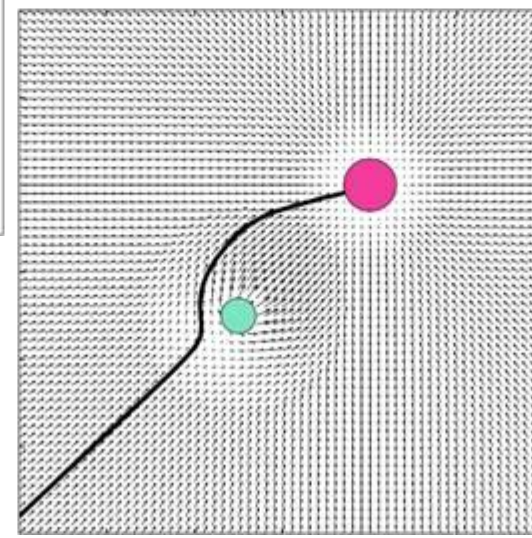
Potential Fields, Representation

- Closed Form as a math. formular,
- Gridbased







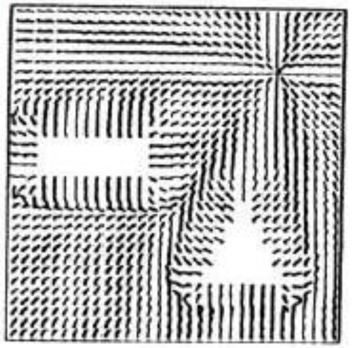


Freie Universität Berlin

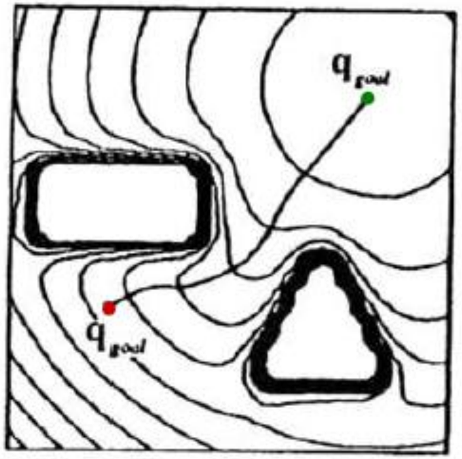
Potential field method

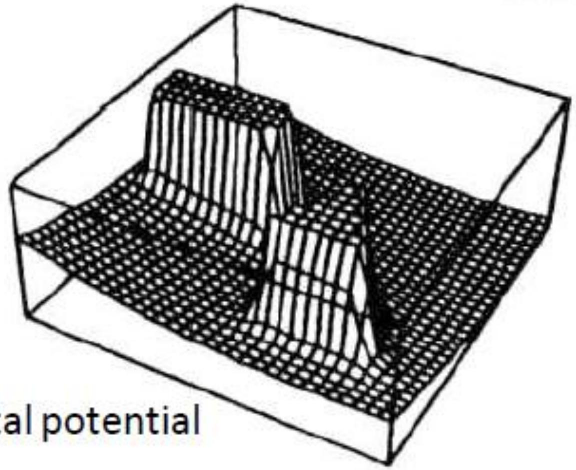
- After the total potential is obtained, generate force field (negative gradient)
- Let the sum of the forces control the robot.

Negative gradient



Equipotential contours

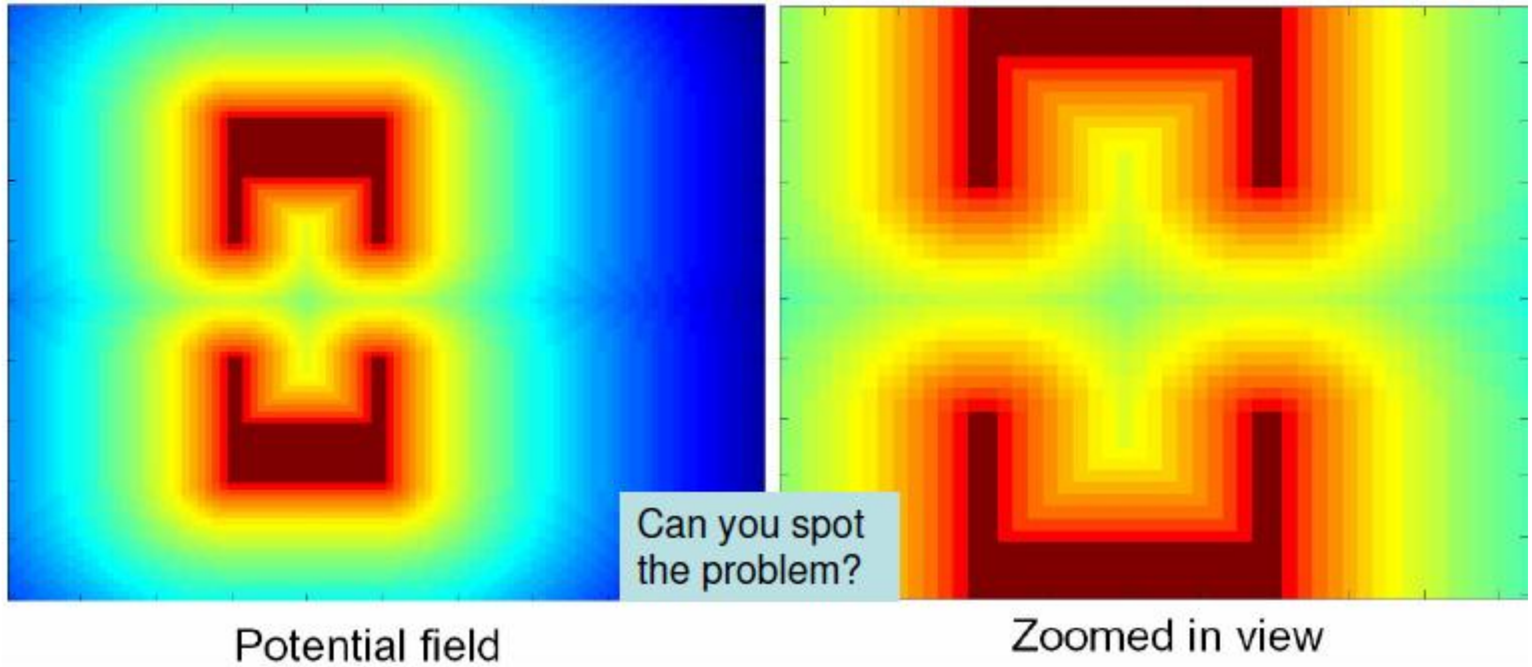




Total potential

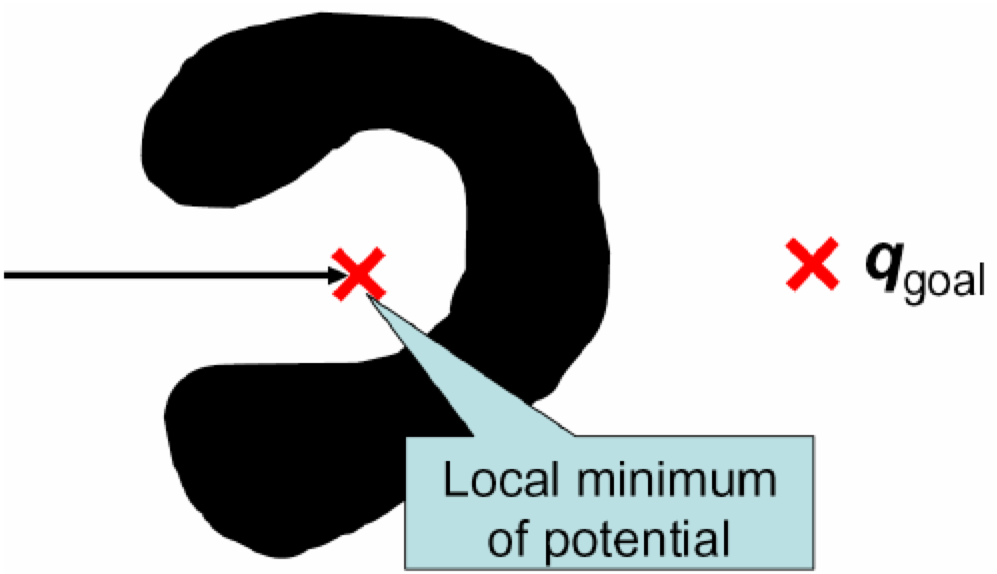
To a large extent, this is computable from sensor readings.

Potential Fields: Limitations



- Completeness
- Problems in higher dimensions

Local Minimum Problem



- Potential fields in general exhibit local minima
- Special case: Navigation function
  - $U(\boldsymbol{q}_{goal}) = 0$
  - For any ***q* different from q_{goal}, there exists a** neighbor ***q' such that U(q') < U(q)***

Getting out of Local Minima I

- Repeat
  - If $U(q)$ ***= 0 return Success***
  - If too many iterations return Failure
  - Else:
    - Find neighbor q_n ***of q with smallest*** $U(q_n)$
    - If $U(q_n) < U(q)$ ***OR*** q_n ***has not yet been*** visited
      - Move to q_n ($q \leftarrow q_n$)
      - Remember q_n

May take a long time to explore region “around” local minima

Getting out of Local Minima II

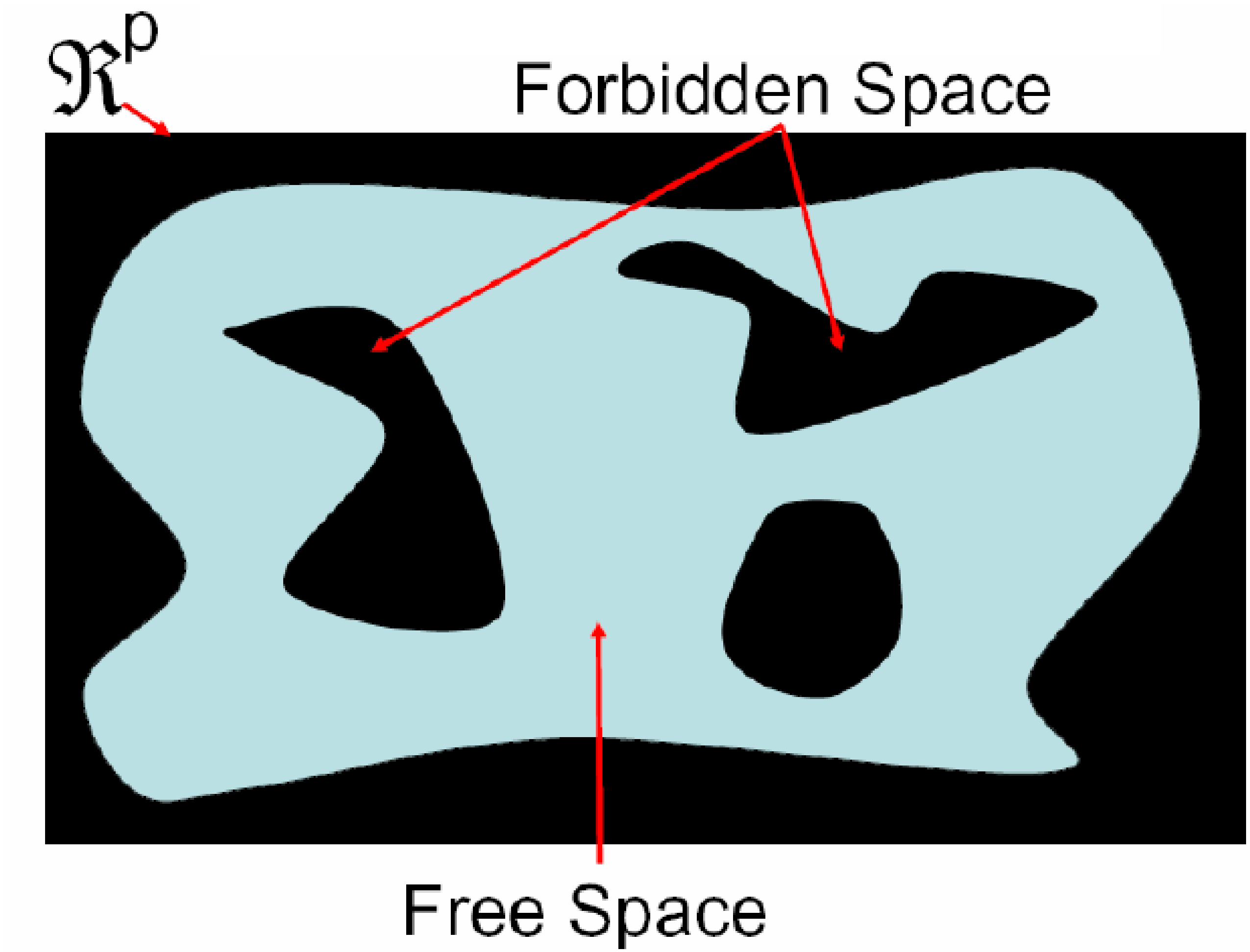
- Repeat
  - If $U(\mathbf{q}) = 0$ ***return Success***
  - If too many iterations return Failure
  - Else:
    - Find neighbor $\mathbf{q}_n$ ***of q with smallest*** $U(\mathbf{q}_n)$
    - If $U(\mathbf{q}_n) < U(\mathbf{q})$
      - Move to $\mathbf{q}_n$ $(\mathbf{q} \leftarrow \mathbf{q}_n)$
    - Else
      - Take a random walk for *T steps starting at* $\mathbf{q}_n$
      - Set ***q to the configuration reached at the end of*** the random walk

Topics

- Visibility Graphs
  - Roadmaps
  - Voronoi
- Approximate Cell Decomposition
- Potential Fields
- Probabilistic Roadmaps (Sampling)

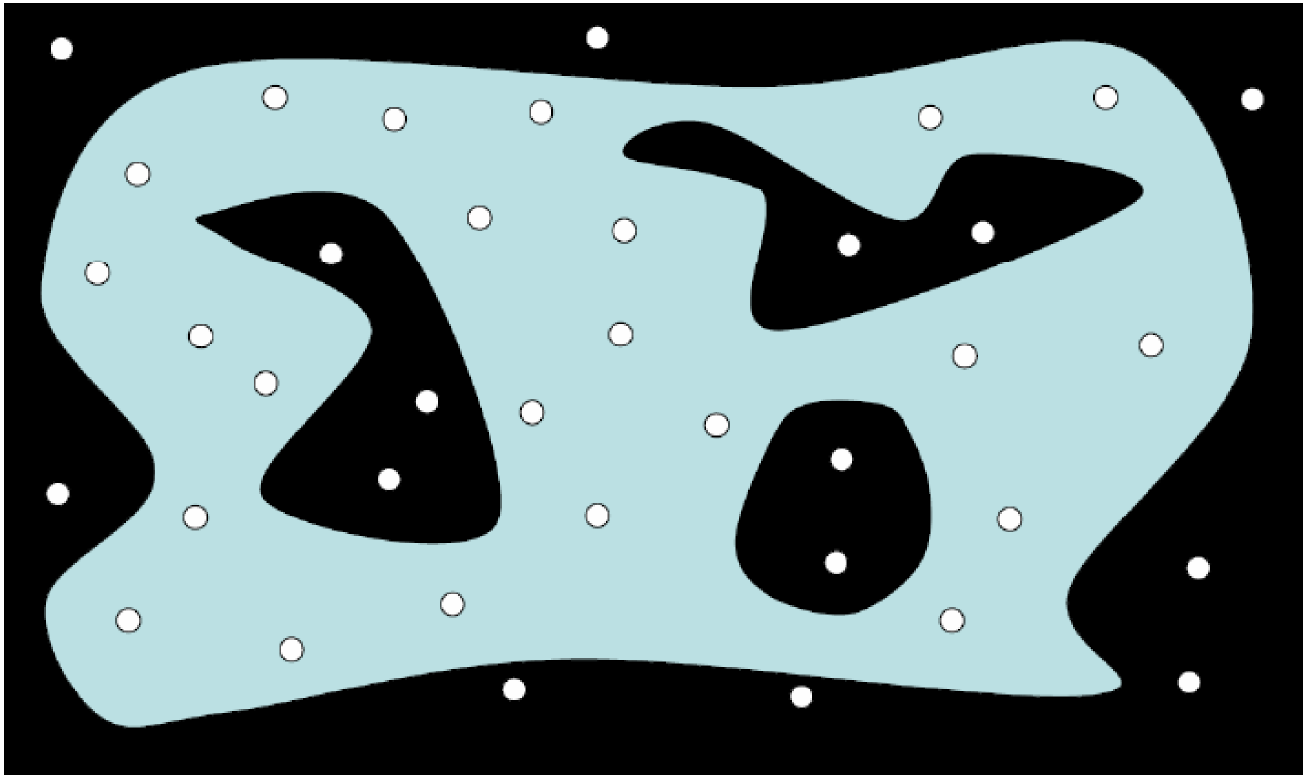
Completely describing and optimally exploring the C-space is too hard in high dimension + it is not necessary → Limit ourselves to finding a "good" sampling of the C-space

Sampling Techniques (Probabilistic Roadmaps)



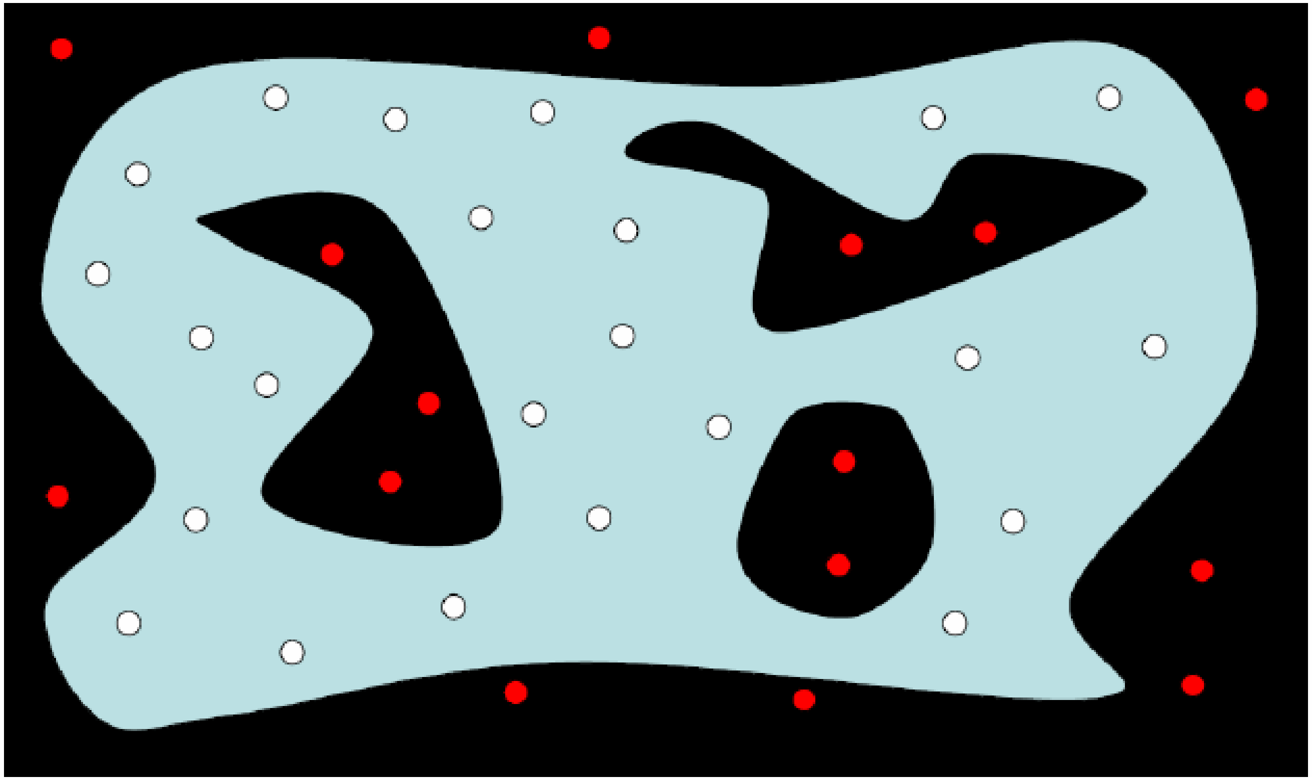
Sampling Techniques

Sample random locations



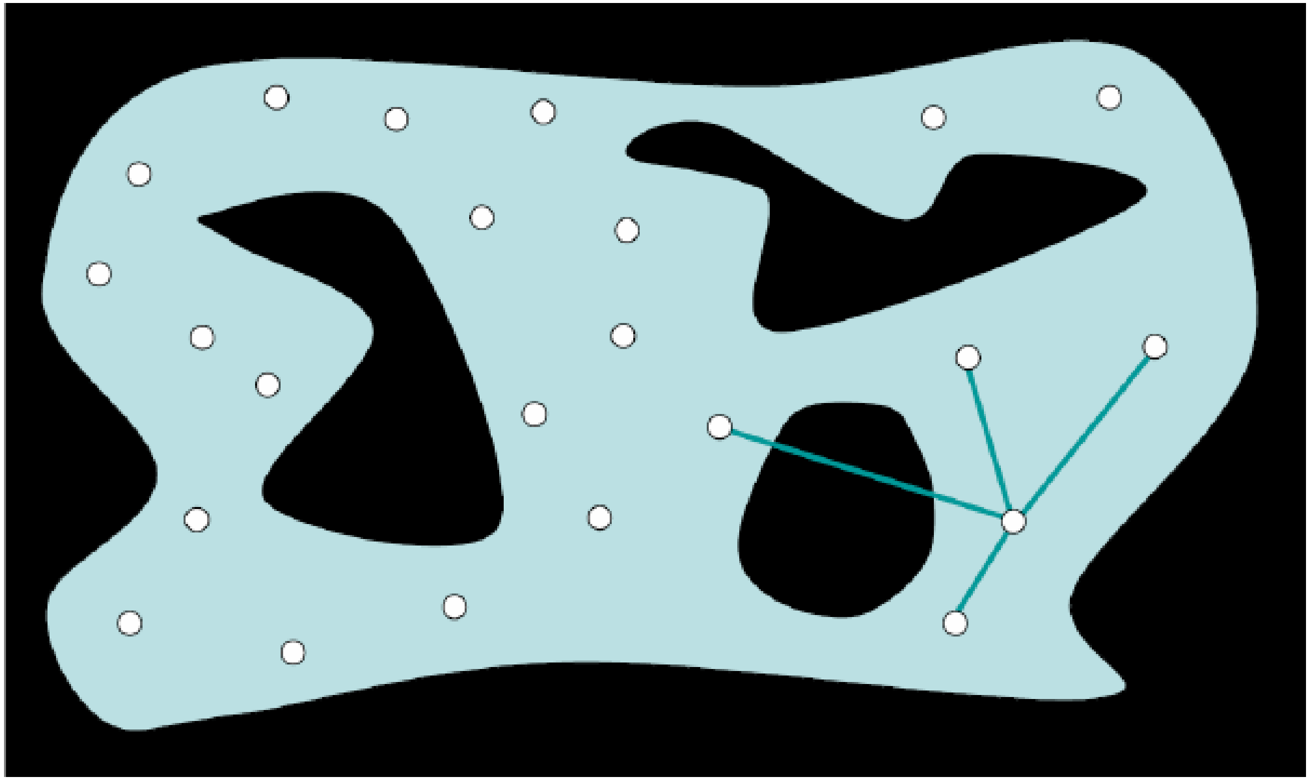
Sampling Techniques

Remove the samples in the forbidden regions



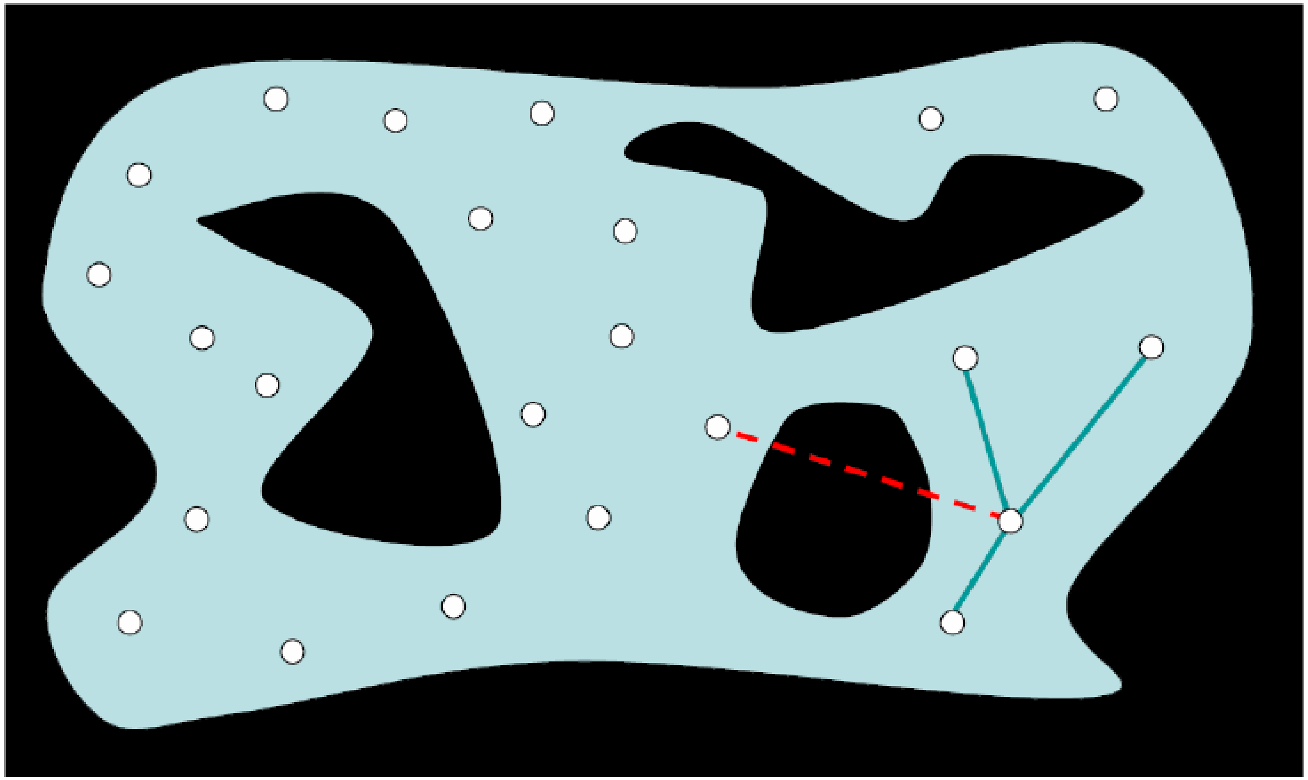
Sampling Techniques

Link each sample to its *K* nearest neighbors



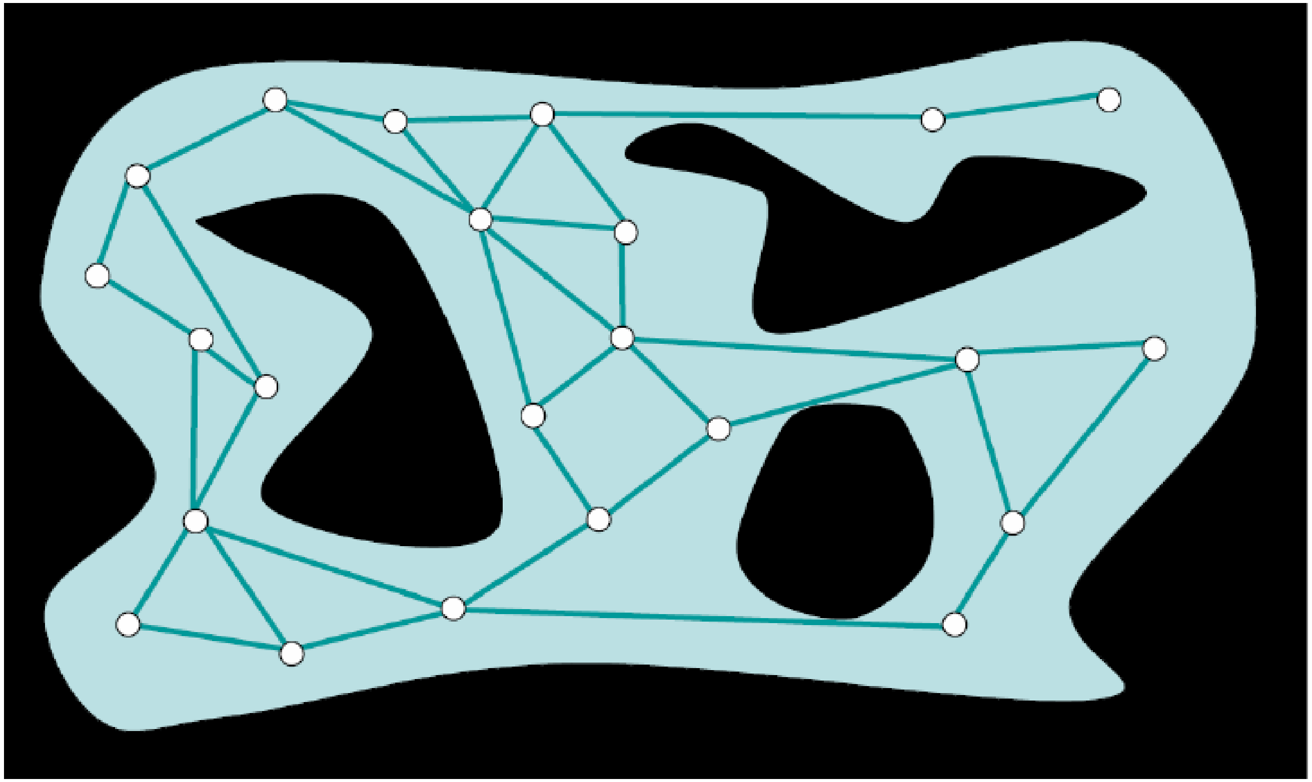
Sampling Techniques

Remove the links that cross forbidden regions



Sampling Techniques

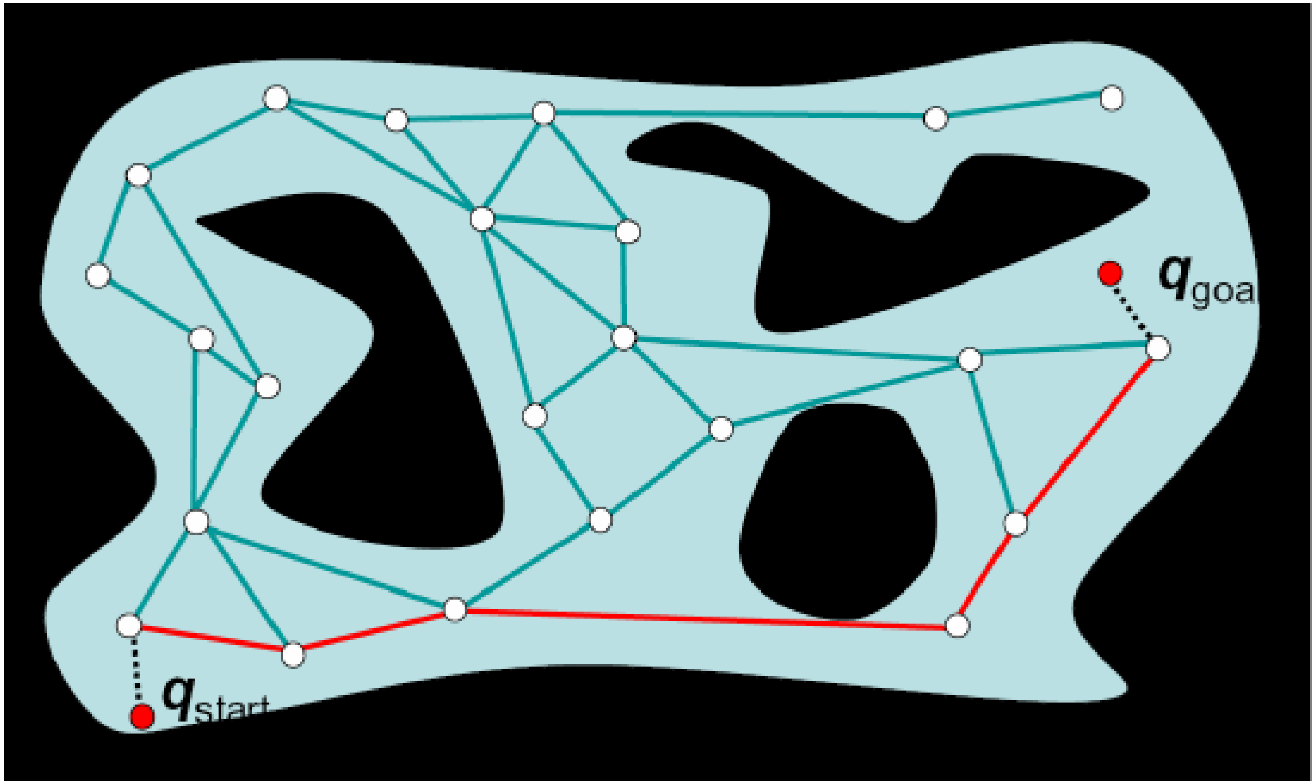
Remove the links that cross forbidden regions



The resulting graph is a *probabilistic roadmap (PRM)*

Sampling Techniques

Link the start and goal to the PRM and search using A*



Sampling Techniques

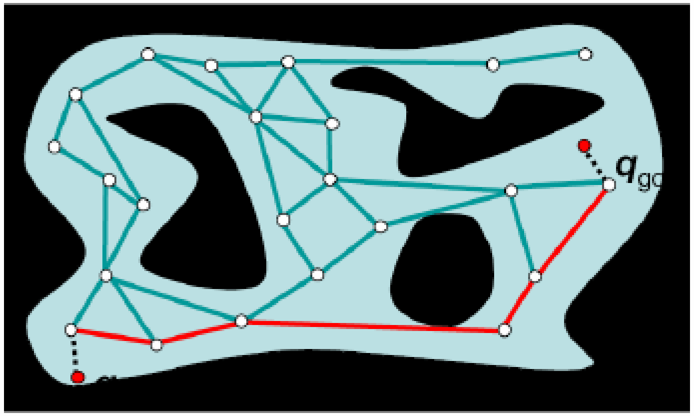
Continuous Space

↓

Discretization

↓

A* Search

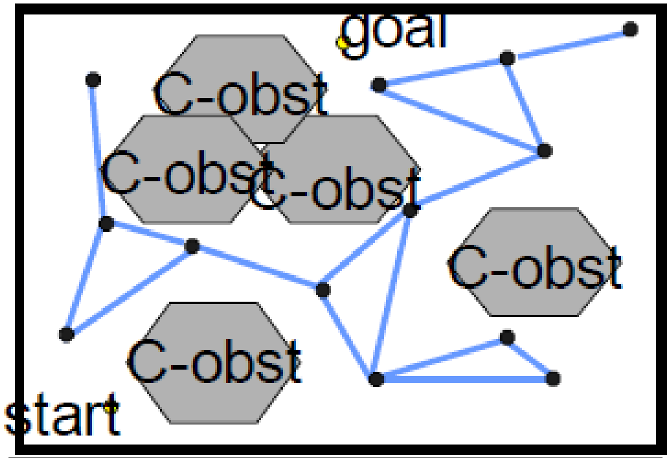


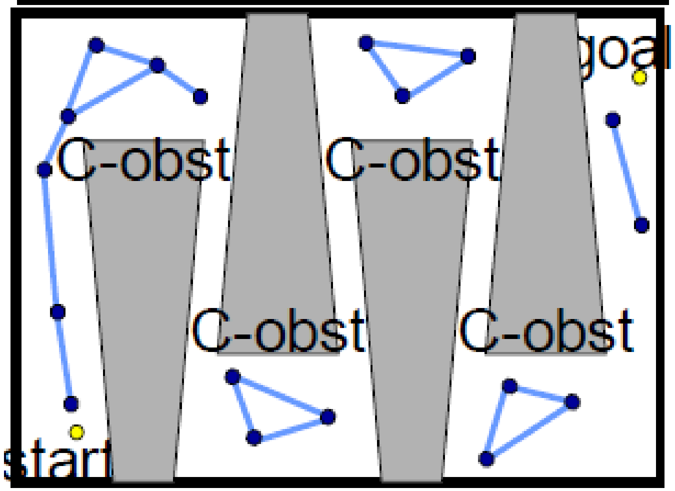
- “Good” sampling strategies are important:
  - Uniform sampling
  - Sample more near points with few neighbors
  - Sample more close to the obstacles
  - Use pre-computed sequence of samples

Sampling Techniques

- Remarkably, we can find a solution by using *relatively few randomly* sampled points.
- In most problems, a relatively small number of samples is sufficient to cover most of the feasible space with probability 1
- For a large class of problems:
  - Prob(finding a path) → 1 exponentially with the number of samples
- *But*, cannot detect that a path does not exist

Sampling Techniques (Probabilistic Roadmaps), Contd.





PRMs: The Good News

1. PRMs are probabilistically complete
2. PRMs apply easily to high-dimensional C-space
3. PRMs support fast queries w/ enough preprocessing

Many success stories where PRMs solve previously unsolved problems

PRMs: The Bad News

1. PRMs don't work as well for some problems:
– unlikely to sample nodes in narrow passages
– hard to sample/connect nodes on constraint surfaces

Literature

- J.C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.
- S. LaValle, Planning Algorithms. 2006. http://msl.cs.uiuc.edu/planning/

- Likhachev, ARA*, CMU

- Toussaint, Lecture Notes Robotics, 2011
- Dr. John (Jizhong) Xiao, City College New York

Literature, contd.

- (Limited) background in Russell&Norvig Chapter 25

- H. Choset et al., Principles of Robot Motion: Theory, Algorithms, and Implementations. 2006.

- Other demos/examples:
  - http://voronoi.sbp.ri.cmu.edu/~choset/
  - http://www.kuffner.org/james/research.html
  - http://msl.cs.uiuc.edu/rrt/