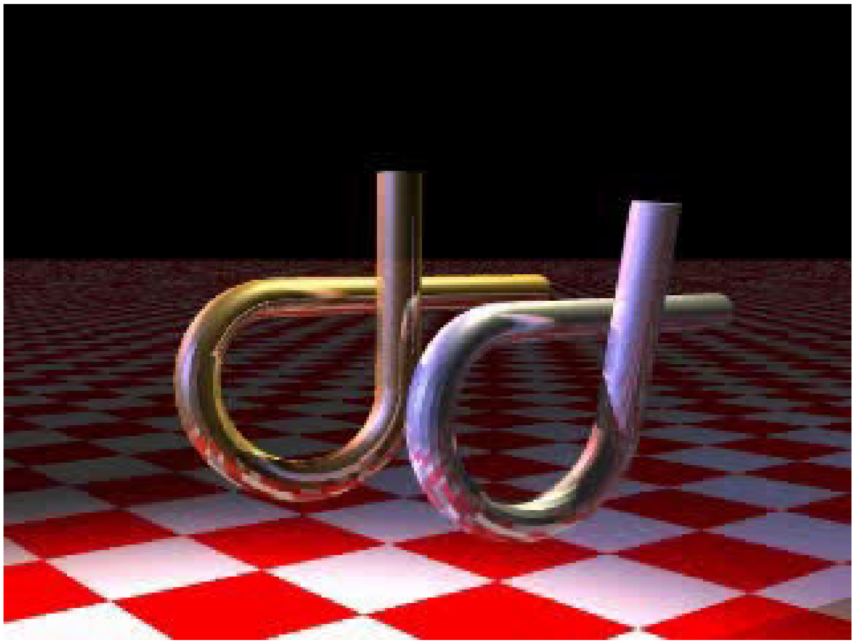
Robot Motion Planning

Movies/demos provided by James Kuffner and Howie Choset +
Examples from J.C. latombe's book (references on the last page)

Example from Howie Choset



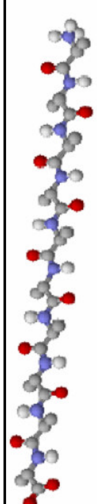
Example from James Kuffner

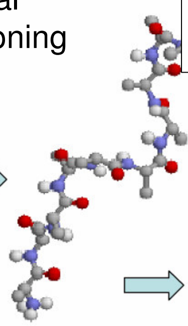
Example from Howie Choset

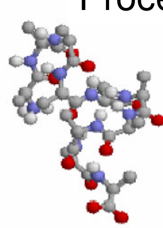
Robot Motion Planning

- Application of earlier search approaches (A*, stochastic search, etc.)
- Search in geometric structures
- ***Spatial reasoning***
- Challenges:
  - Continuous state space
  - Large dimensional space

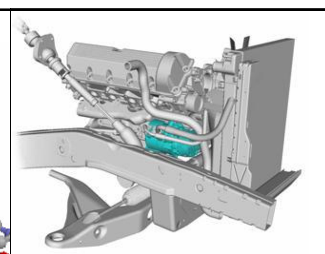
Robotics is only (a small) one of many applications of spatial reasoning

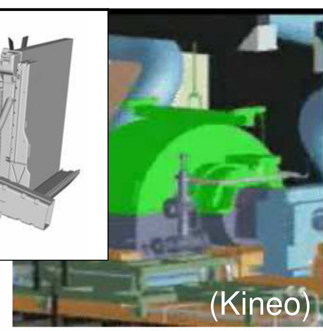






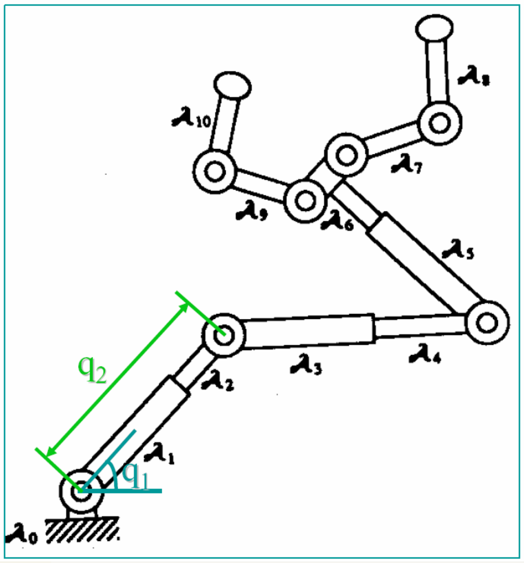
Biology





Process Engineering/Design

Degrees of Freedom

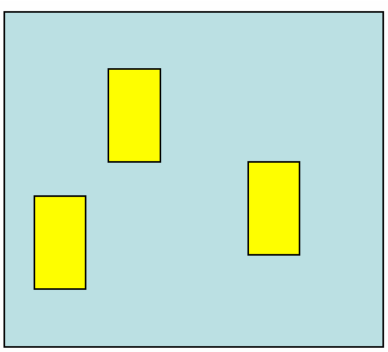


- The geometric configuration of a robot is defined by *p* degrees of freedom (DOF)
- Assuming *p* DOFs, the geometric configuration *A* of a robot is defined by *p* variables:

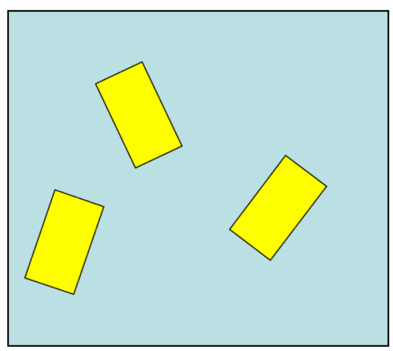
$A(\boldsymbol{q})$ with $\boldsymbol{q} = (q_1,\ldots,q_p)$

- 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



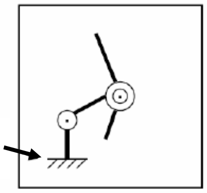
Allowed to move only in *x* and *y*: 2DOF



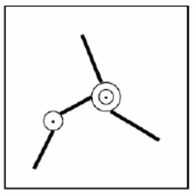
Allowed to move in *x* and y and to rotate: 3DOF (x,y,θ)

Examples

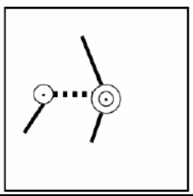
Fixed



Fixed (attached at the base)

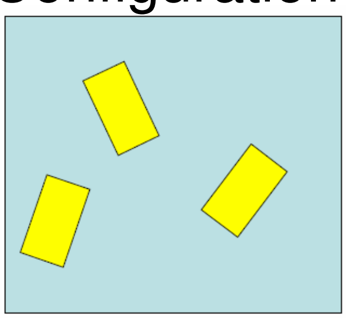


Free Flying



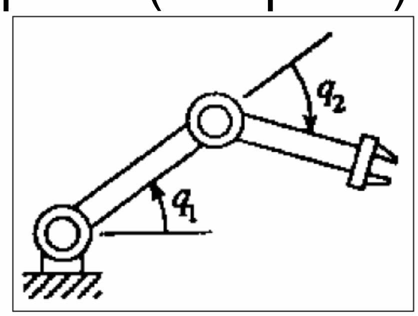
Fixed (the dashed line is constrained to be horizontal)

Configuration Space (C-Space)



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

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

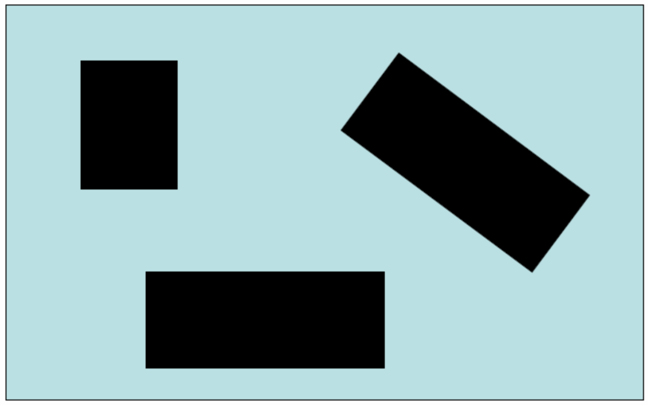


$\boldsymbol{q} = (q_1,q_2)$

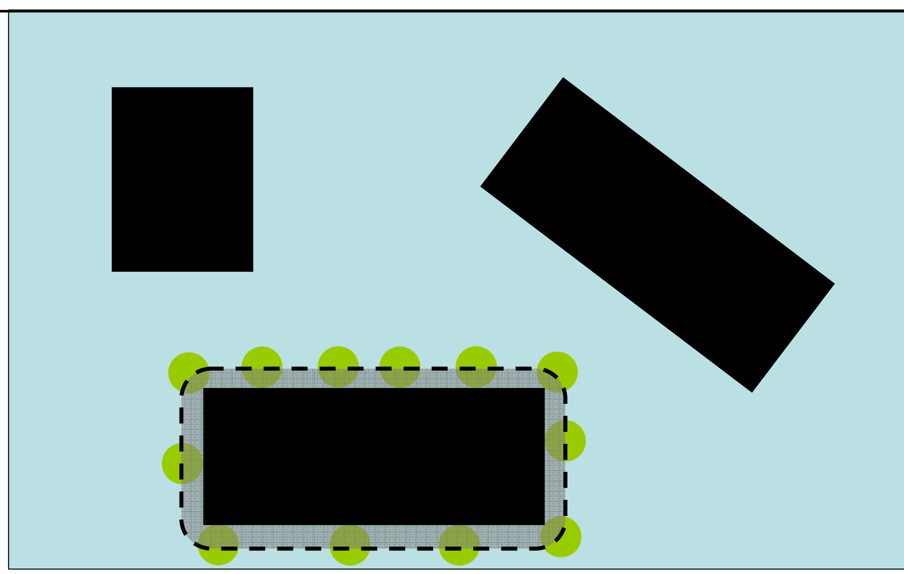
$\mathcal{C} =$ 2-D rotations x 2-D rotations

- 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

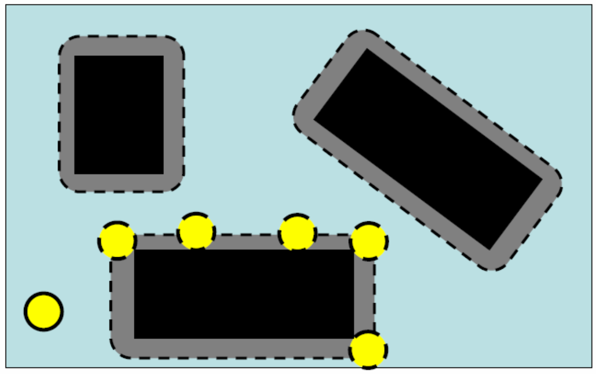
Free Space: Point Robot



- $\mathcal{C}_{free}$ = {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

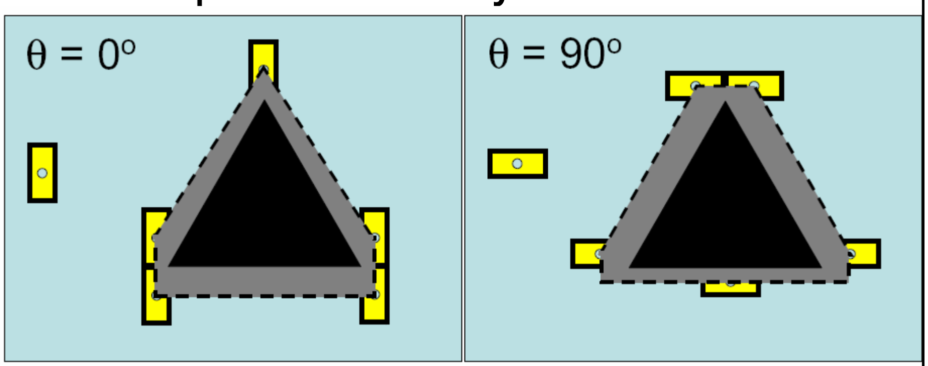


Free Space: Symmetric Robot

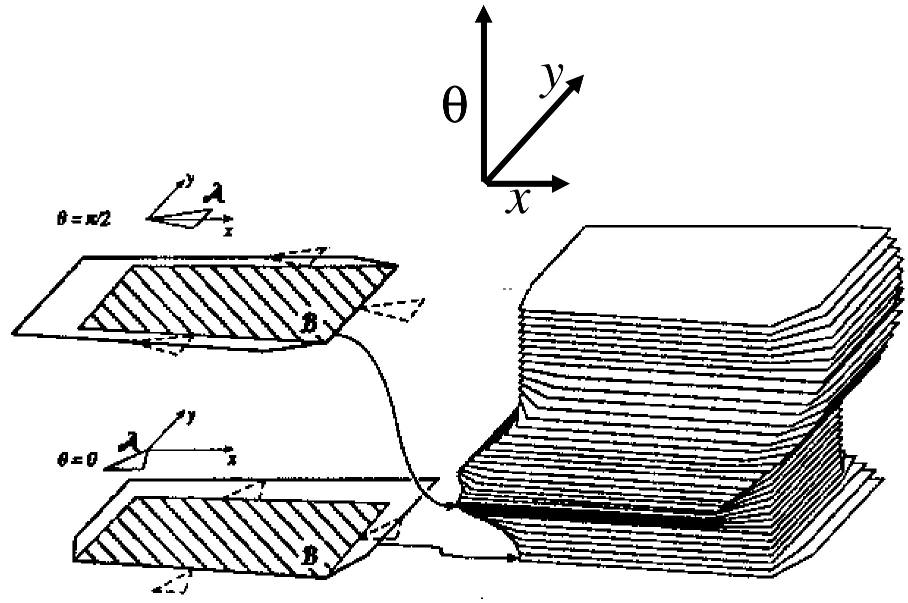


- We still have $\mathcal{C} = R^2$ because orientation does not matter
- Reduce the problem to a point robot by expanding the obstacles by the radius of the robot

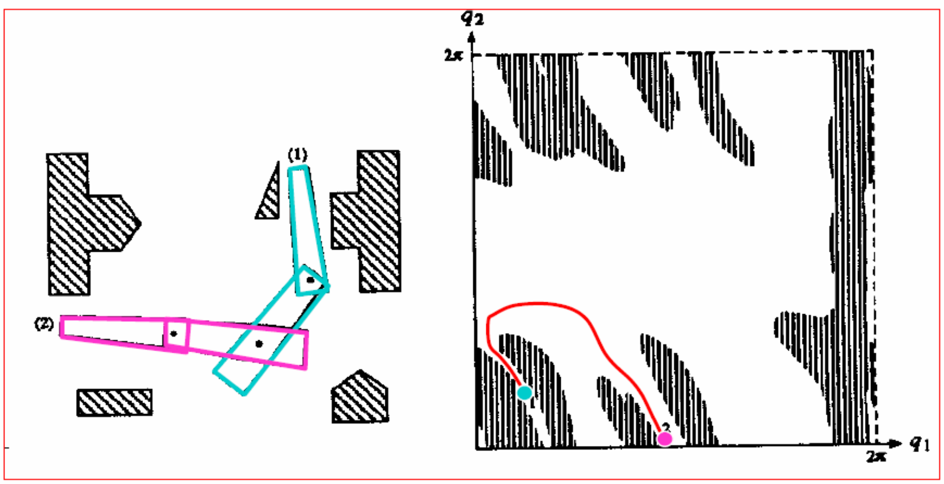
Free Space: Non-Symmetric Robot



- 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

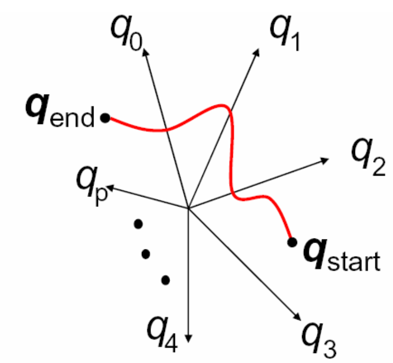


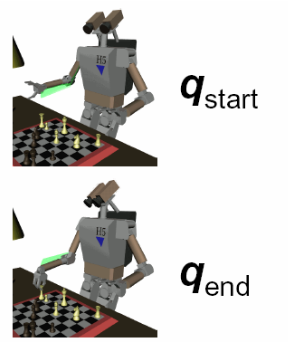
More Complex C-Spaces



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

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

Any Formal Guarantees? Generic Piano Movers Problem

- Formal Result (but not terribly useful for practical algorithms):
  - p: Dimension of $\mathcal{C}$
  - m: Number of polynomials describing $\mathcal{C}_{free}$
  - d: Max degree of the polynomials
- A path (if it exists) can be found in time *exponential* in p and *polynomial* in m and d

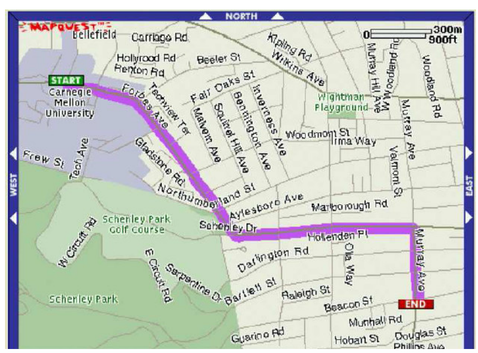
[From J. Canny. "The Complexity of Robot Motion Planning Plans". MIT Ph.D. Dissertation. 1987]

Approaches

- Basic approaches:
  - Roadmaps
    - Visibility graphs
    - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Extensions
  - Sampling Techniques
  - On-line algorithms

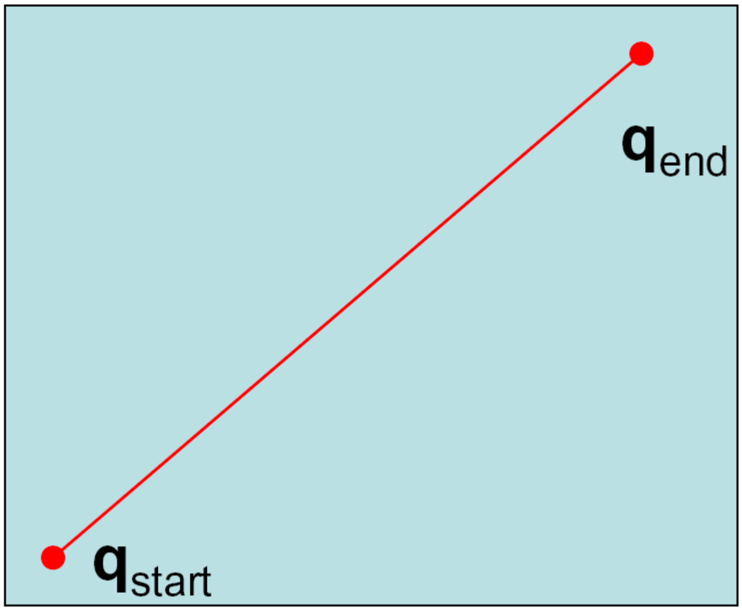
In all cases: Reduce the intractable problem in continuous C-space to a tractable problem in a discrete space → Use all of the techniques we know (A*, stochastic search, etc.)

Roadmaps

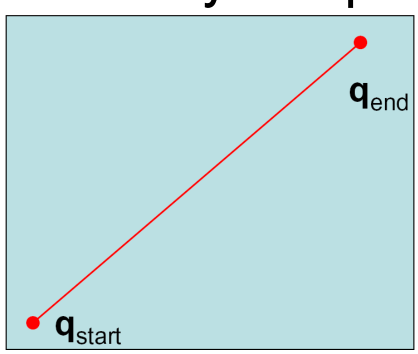


- General idea:
  - Avoid searching the entire space
  - Pre-compute a (hopefully small) graph (the roadmap) such that staying on the "roads" is guaranteed to avoid the obstacles
  - Find a path between $\boldsymbol{q}_{start}$ and $\boldsymbol{q}_{goal}$ by using the roadmap

Visibility Graphs

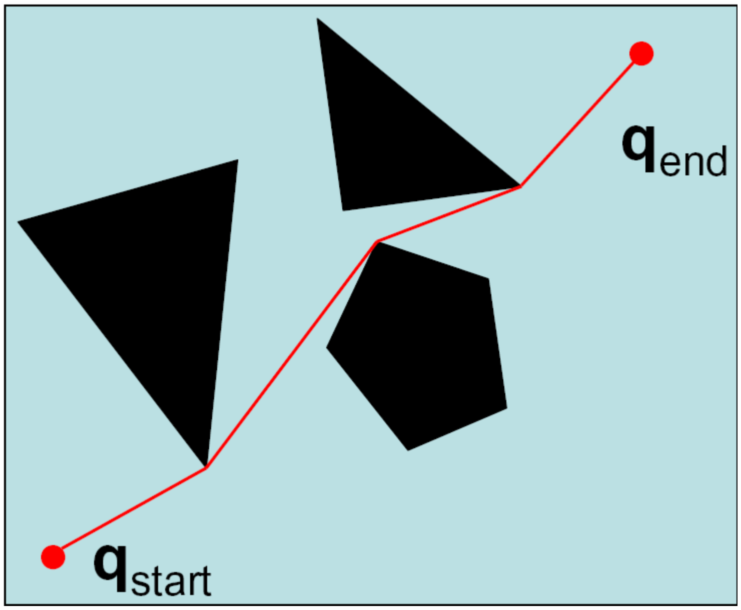


Visibility Graphs

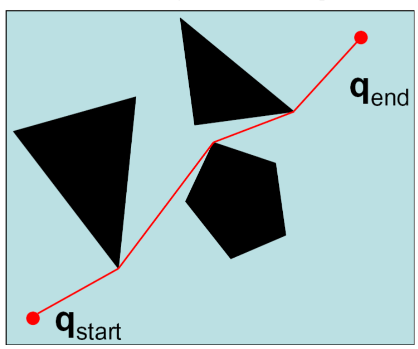


In the absence of obstacles, the best path is the straight line between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$

Visibility Graphs

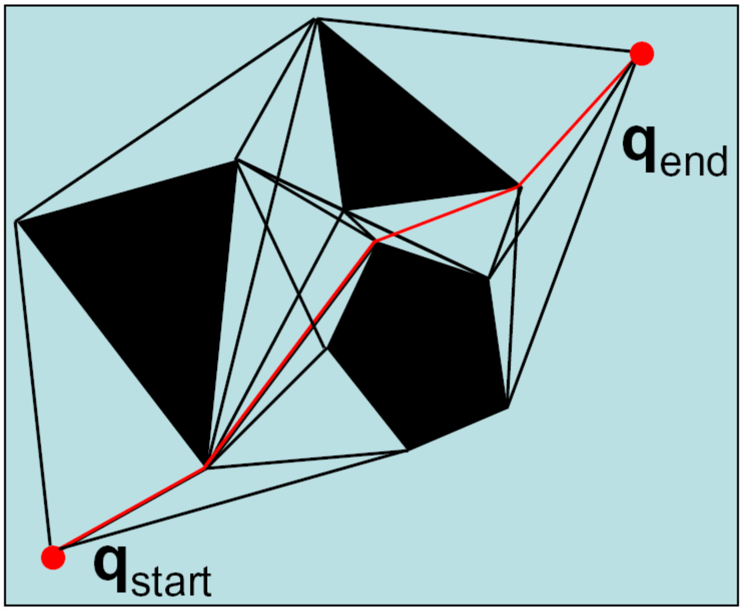


Visibility Graphs

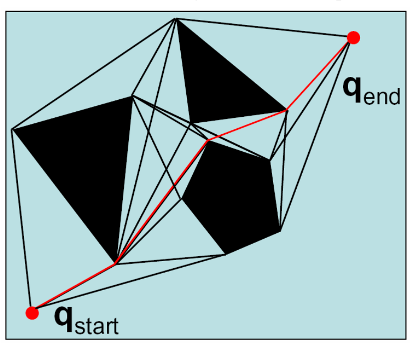


- Assuming polygonal obstacles: It looks like the shortest path is a sequence of straight lines joining the vertices of the obstacles.
- Is this always true?

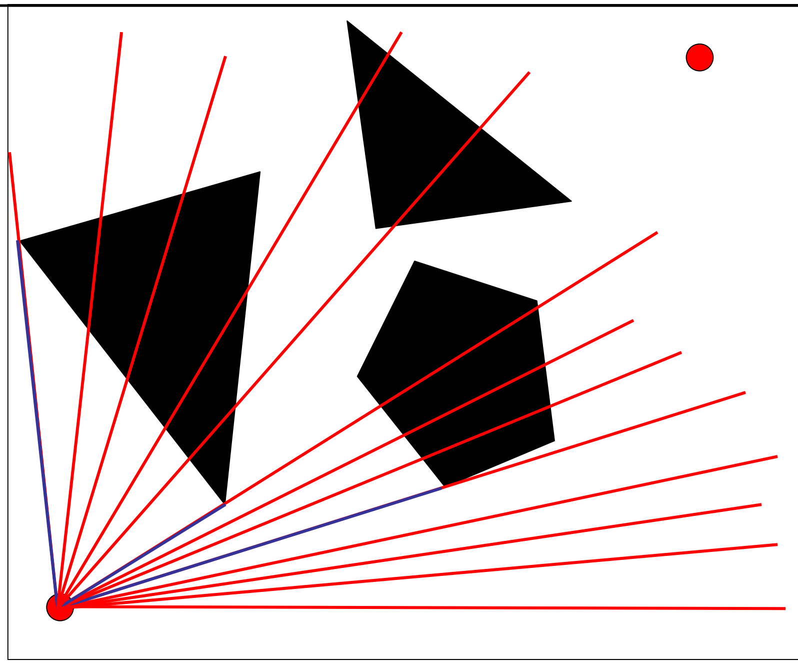
Visibility Graphs



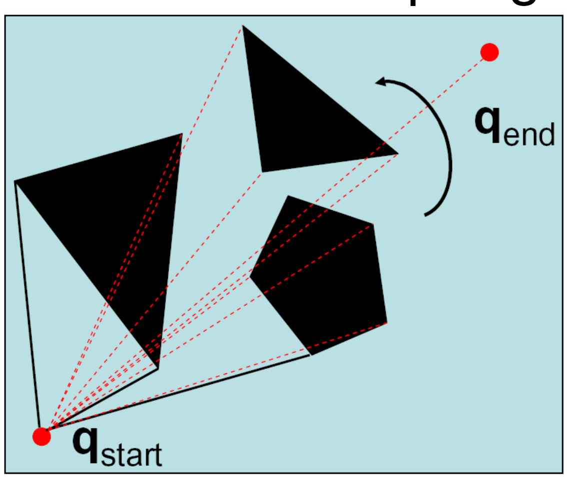
Visibility Graphs



- Visibility graph *G* = set of unblocked lines between vertices of the obstacles + $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$
- A node *P* is linked to a node *P'* if *P'* is visible from *P*
- Solution = Shortest path in the visibility graph

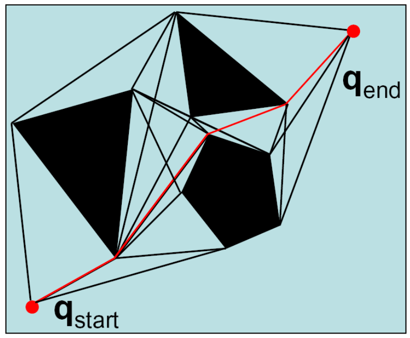


Construction: Sweep Algorithm



- Sweep a line originating at each vertex
- Record those lines that end at visible vertices

Complexity

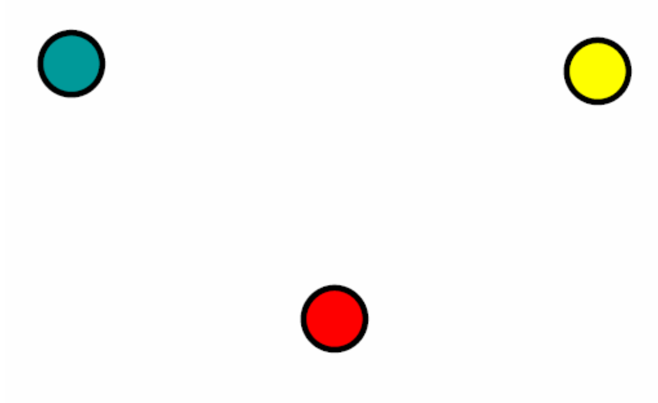


- N = total number of vertices of the obstacle polygons
- Naïve: $O(N^3)$
- Sweep: $O(N^2 \log N)$
- Optimal: $O(N^2)$

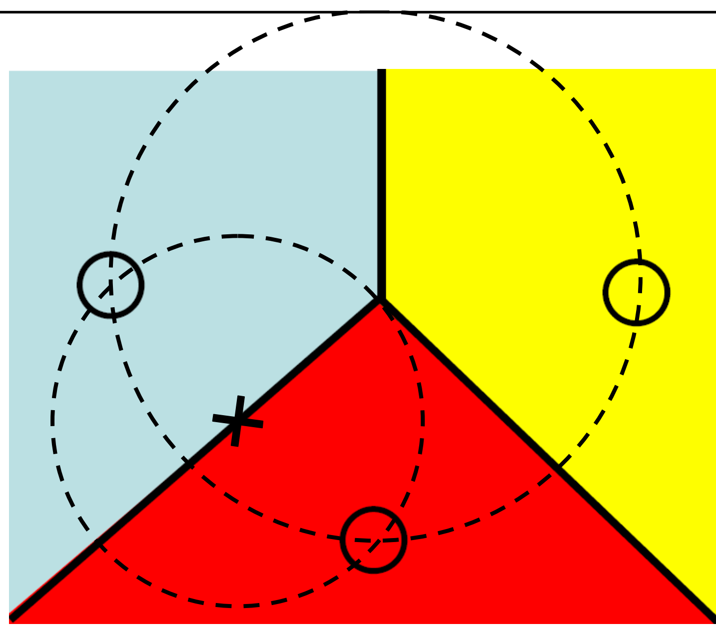
Visibility Graphs: Weaknesses

- Shortest path but:
  - Tries to stay as close as possible to obstacles
  - Any execution error will lead to a collision
  - Complicated in >> 2 dimensions
- We may not care about strict optimality so long as we find a safe path. Staying away from obstacles is more important than finding the shortest path
- Need to define other types of “roadmaps”

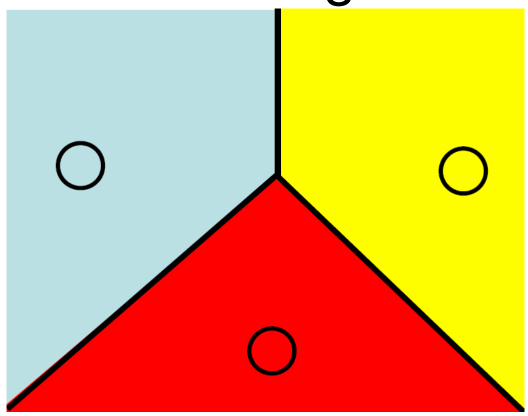
Voronoi Diagrams



- 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

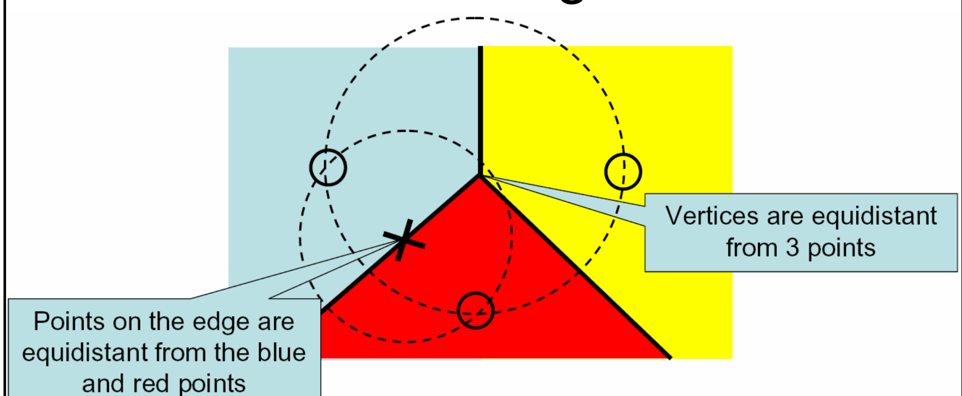


Voronoi Diagrams



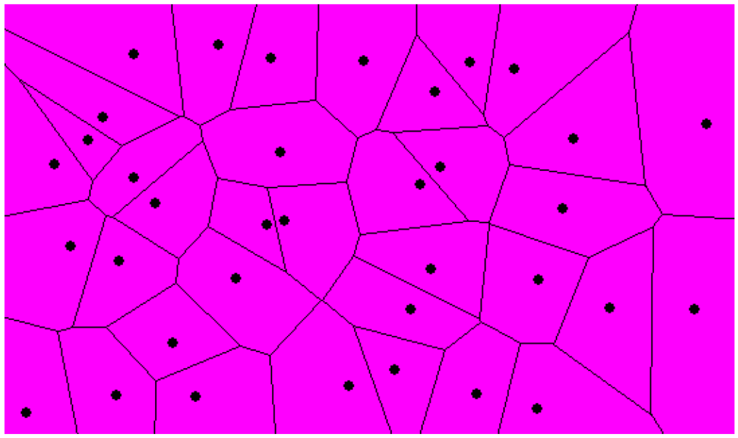
- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points

Voronoi Diagrams



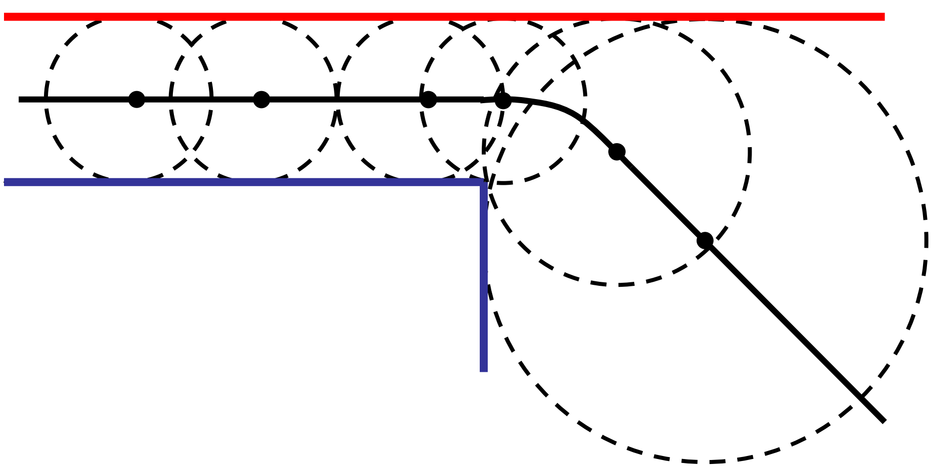
- Voronoi diagram = The set of line segments separating the regions corresponding to different colors
  - Line segment = points equidistant from 2 data points
  - Vertices = points equidistant from > 2 data points

Voronoi Diagrams

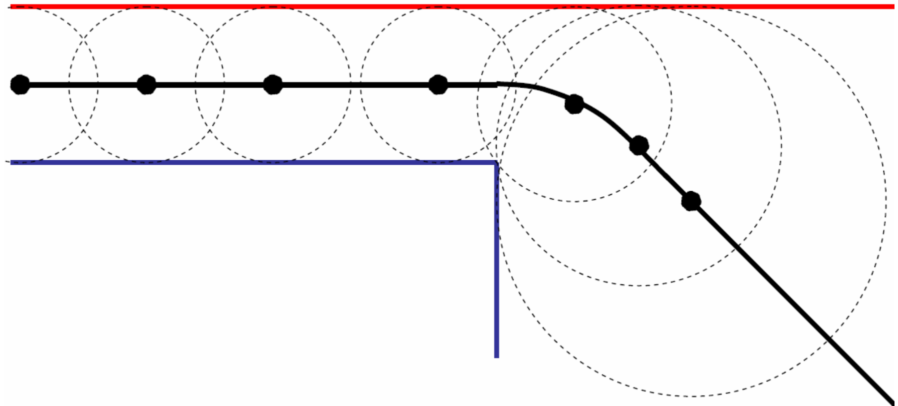


- Complexity (in the plane):
- O(N log N) time
- O(N) space

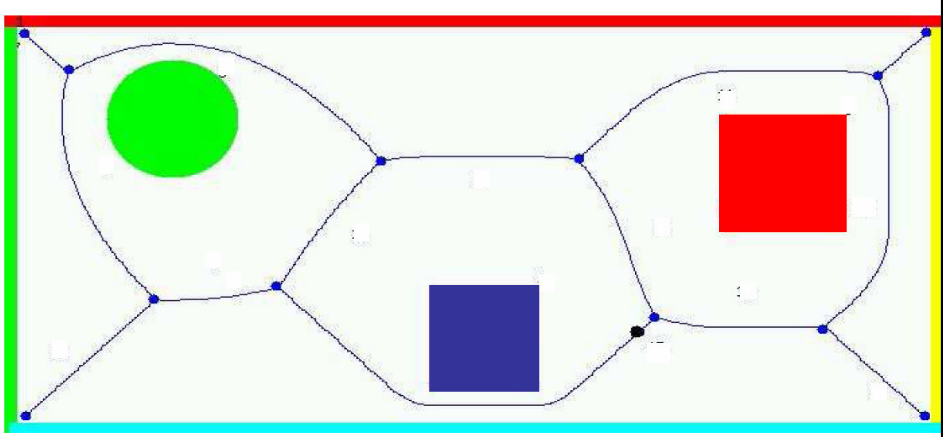
(See for example http://www.cs.cornell.edu/Info/People/chew/Delaunay.html for an interactive demo)

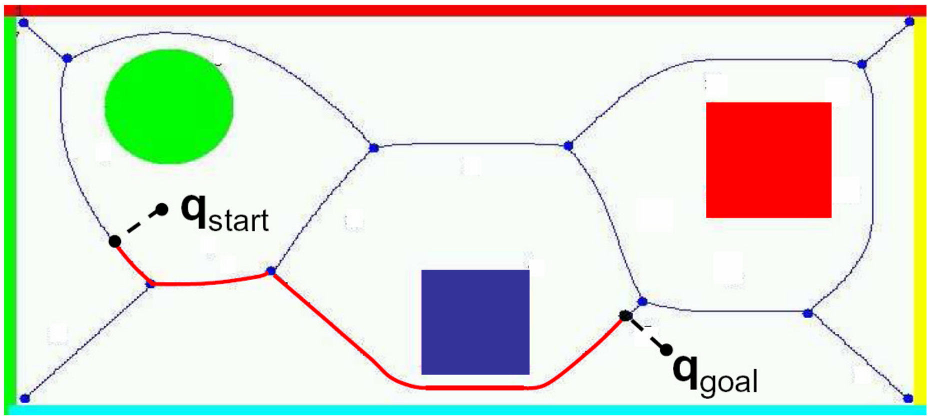


Voronoi Diagrams: Beyond Points

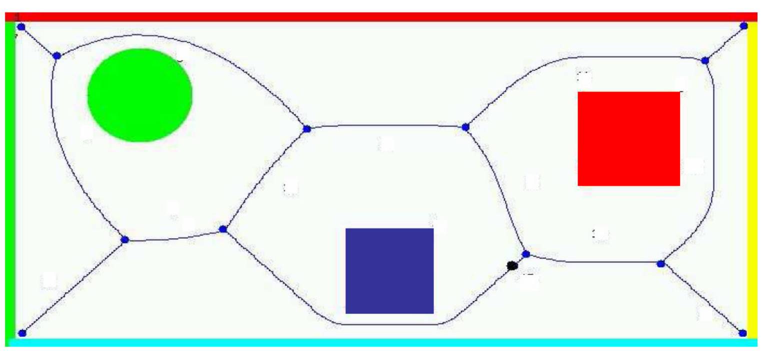


- Edges are combinations of straight line segments and segments of quadratic curves
- Straight edges: Points equidistant from 2 lines
- Curved edges: Points equidistant from one corner and one line



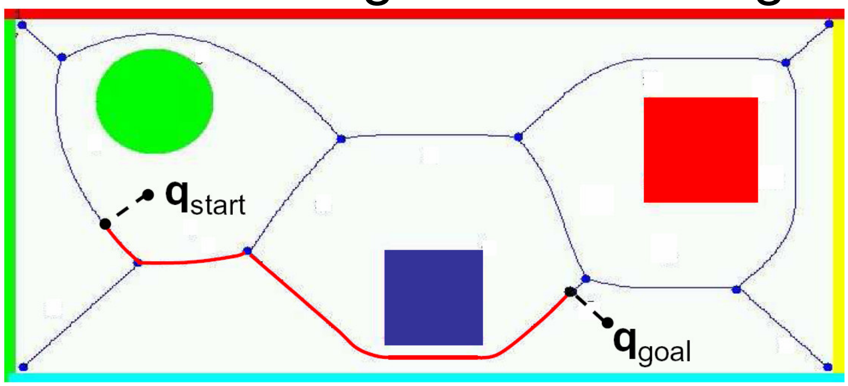


Voronoi Diagrams (Polygons)



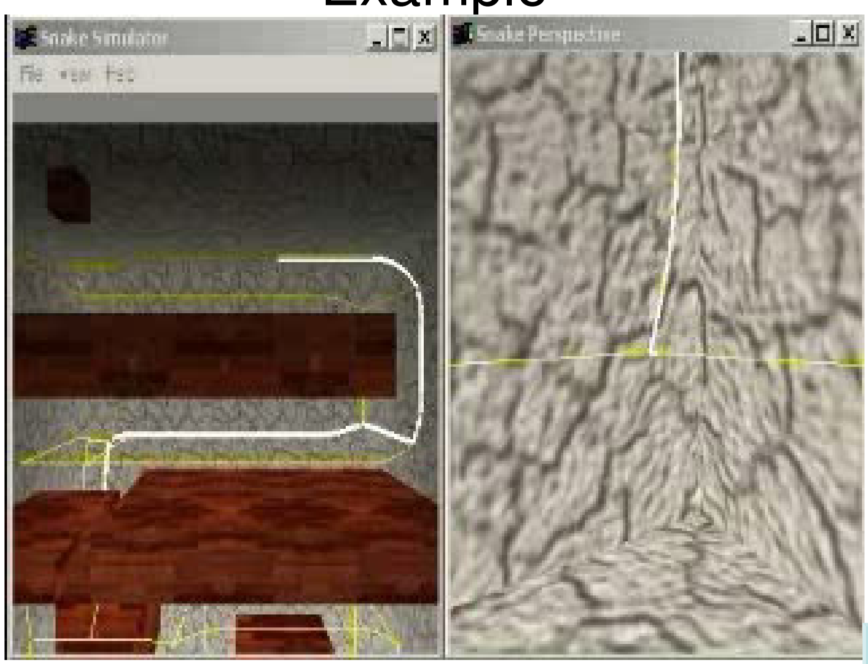
- Key property: The points on the edges of the Voronoi diagram are the *furthest* from the obstacles
- Idea: Construct a path between $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ by following edges on the Voronoi diagram
- (Use the Voronoi diagram as a roadmap graph instead of the visibility graph)

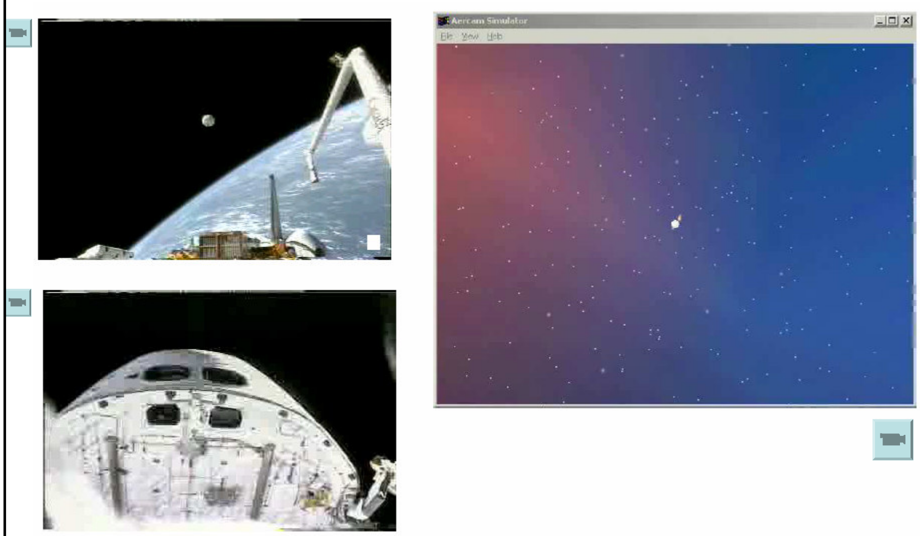
Voronoi Diagrams: Planning



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

Example





Courtesy Howie Choset and http://www.aercam.org

Voronoi: Weaknesses

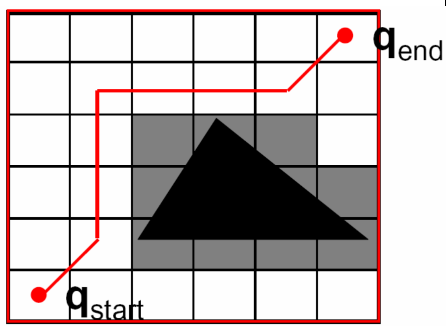
- Difficult to compute in higher dimensions or nonpolygonal worlds
- Approximate algorithms exist
- Use of Voronoi is not necessarily the best heuristic (“stay away from obstacles”) Can lead to paths that are much too conservative
- Can be unstable → Small changes in obstacle configuration can lead to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

Approaches

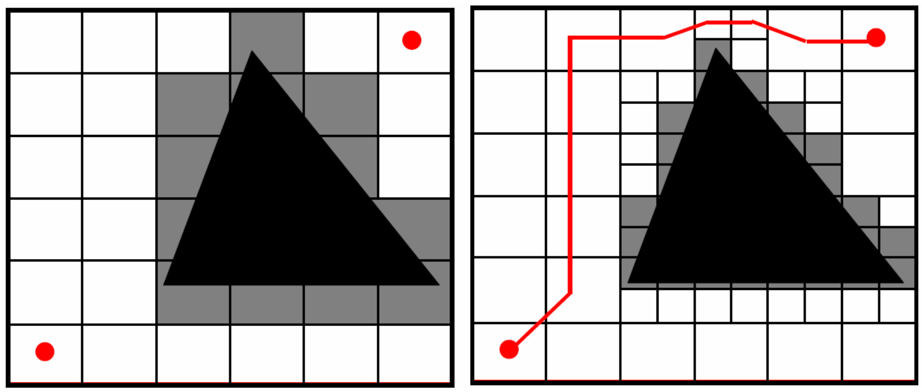
- Basic approaches:
  - Roadmaps
    - Visibility graphs
    - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Extensions
  - Sampling Techniques
  - On-line algorithms

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

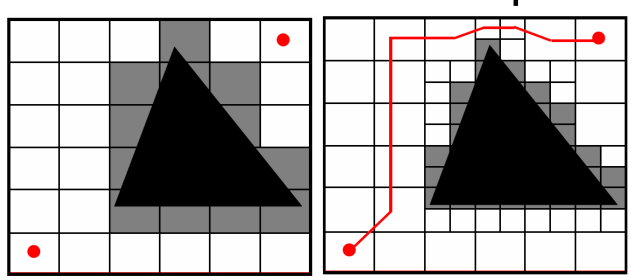
Approximate Cell Decomposition



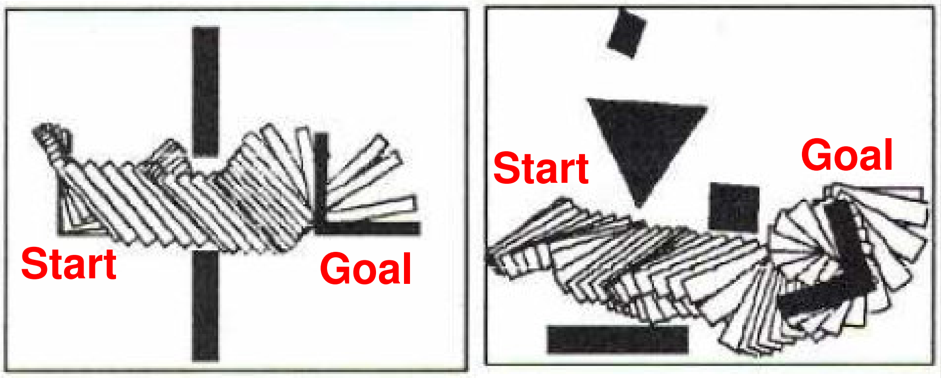
- Define a discrete grid in C-Space
- Mark any cell of the grid that intersects $\mathcal{C}_{obs}$ as blocked
- Find path through remaining cells by using (for example) A* (e.g., use Euclidean distance as heuristic)
- Cannot be *complete* as described so far. Why?

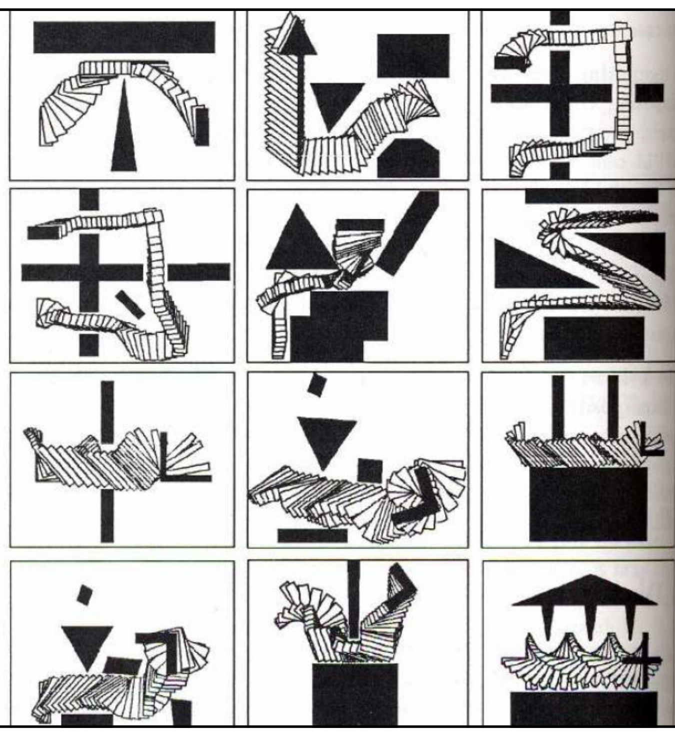


Approximate Cell Decomposition



- Cannot find a path in this case even though one exists
- Solution:
- Distinguish between
  - Cells that are entirely contained in $\mathcal{C}_{obs}$ (*FULL*) and
  - Cells that partially intersect $\mathcal{C}_{obs}$ (*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

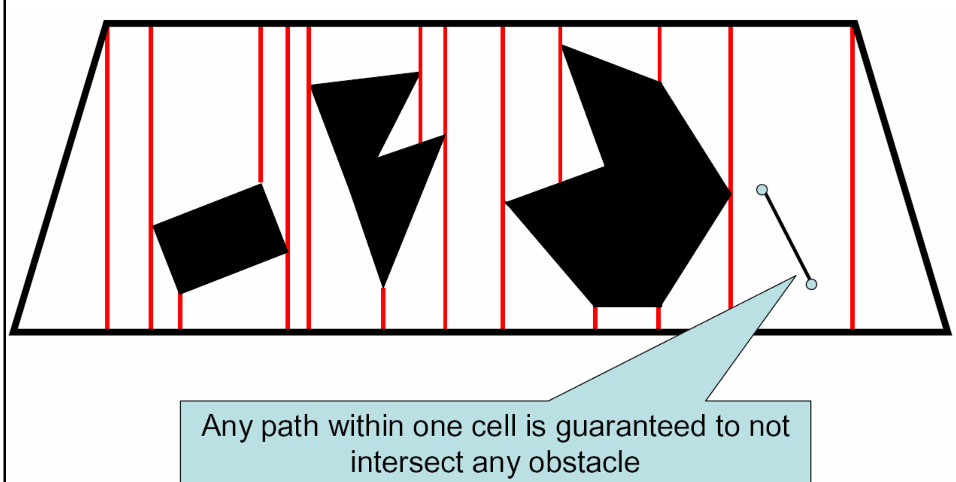

Start
Goal
Start
Goal



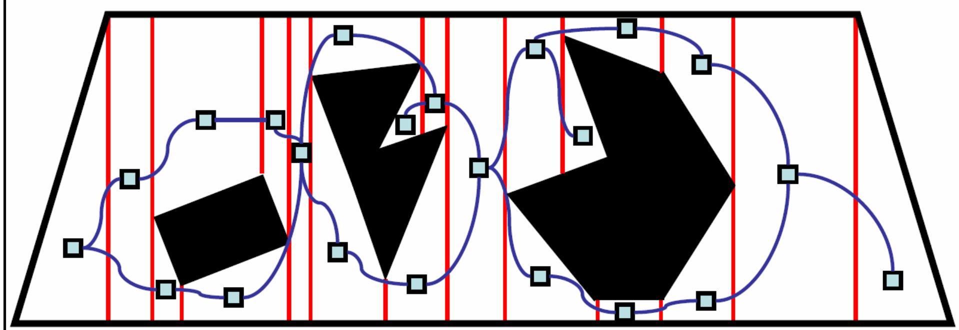
Approximate Cell Decomposition: Limitations

- 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

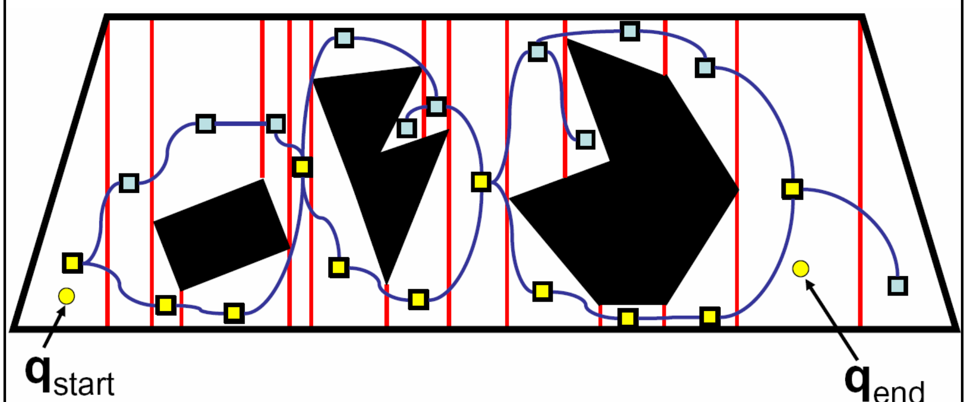


Exact Cell Decomposition

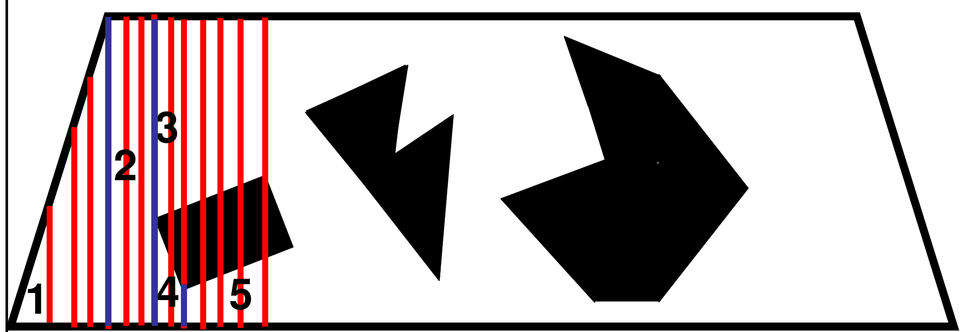


- The graph of cells defines a roadmap

Exact Cell Decomposition



- The graph can be used to find a path between any two configurations



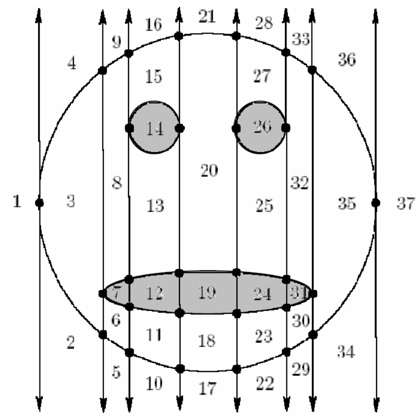
Critical event:
Create new cell

Critical event:
Split cell

Plane Sweep algorithm

- Initialize current list of cells to empty
- Order the vertices of $\mathcal{C}_{obs}$ 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
- Complexity (in 2-D):
  - Time: $O(N \log N)$
  - Space: $O(N)$

Exact Cell Decomposition

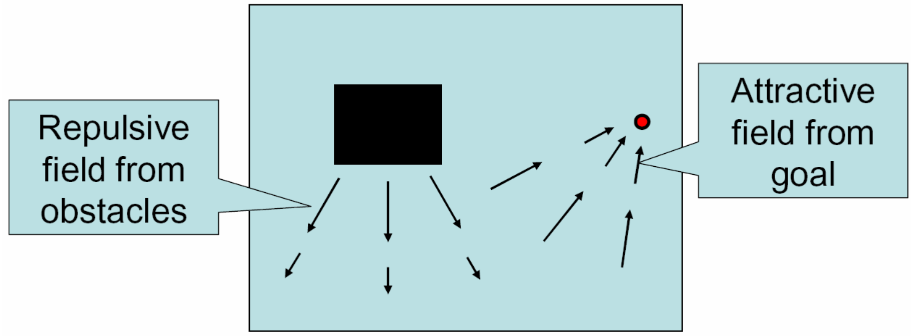


- A version of exact cell decomposition can be extended to higher dimensions and non-polygonal boundaries ("cylindrical cell decomposition")
- Provides exact solution → completeness
- Expensive and difficult to implement in higher dimensions

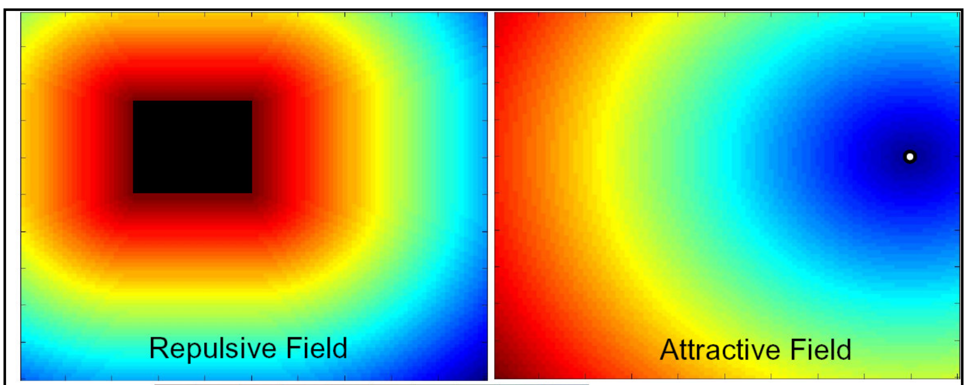
Approaches

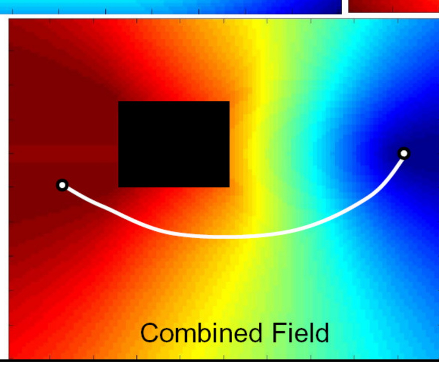
- Basic approaches:
  - Roadmaps
    - Visibility graphs
    - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Extensions
  - Sampling Techniques
  - On-line algorithms

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





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

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

Distance to goal state

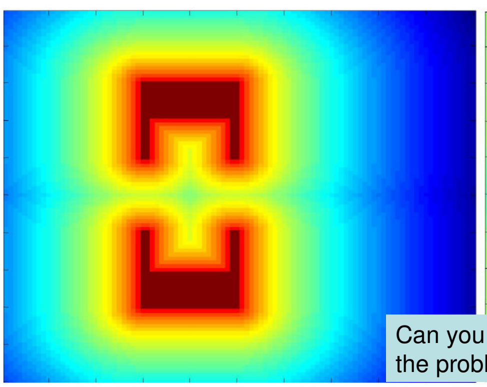
$$U_o(\mathbf{q}) = \frac{1}{d^2(\mathbf{q}, Obstacles)}$$

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

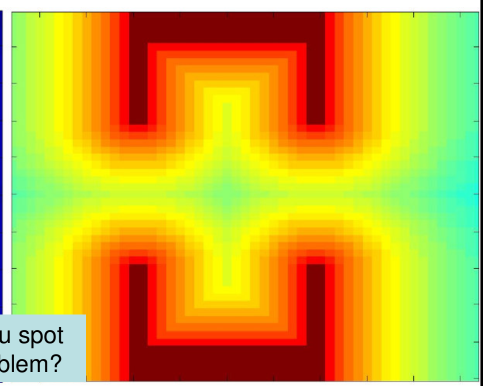
$$U(\mathbf{q}) = U_g(\mathbf{q}) + \lambda U_o(\mathbf{q})$$

λ controls how far we stay from the obstacles

Potential Fields: Limitations



Potential field

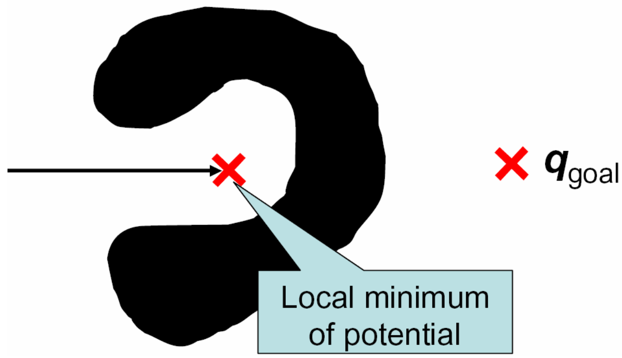


Zoomed in view

Can you spot the problem?

- 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 $\boldsymbol{q}$ different from $\boldsymbol{q}_{goal}$, there exists a neighbor $\boldsymbol{q}$' such that $U(\boldsymbol{q}') < U(\boldsymbol{q})$

Getting out of Local Minima I

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

May take a long time to explore region "around" local minima

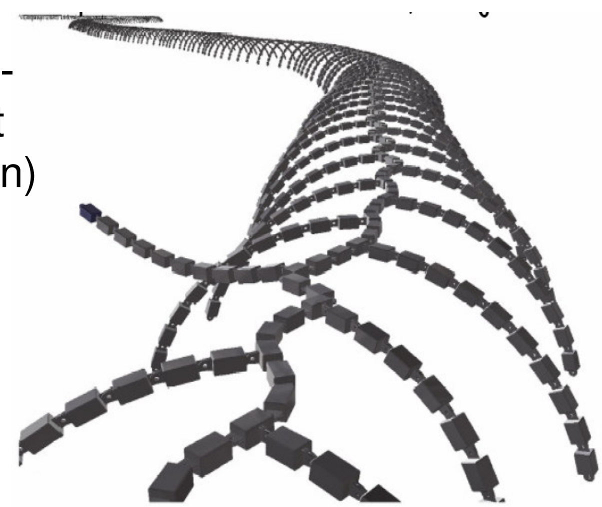
Getting out of Local Minima II

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

Similar to stochastic search and simulated annealing: We escape local minima faster

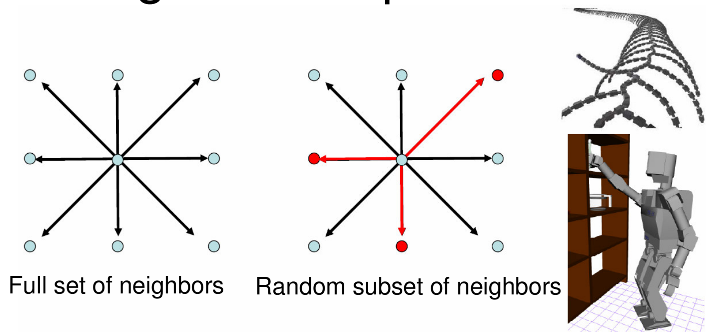
Large C-Space Dimension

Millipede-like robot (S. Redon)



~13,000 DOFs !!!

Dealing with C-Space Dimension

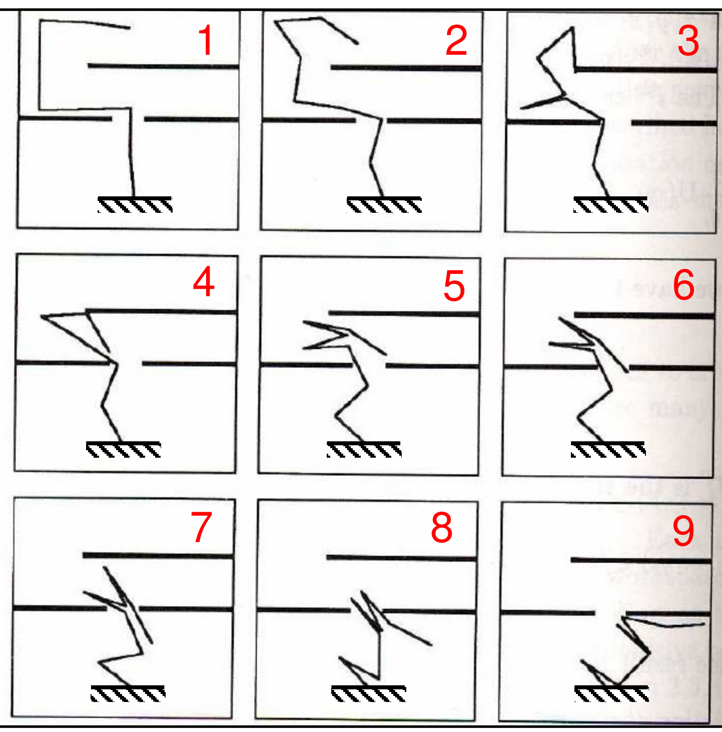


- We should evaluate all the neighbors of the current state, but:
- Size of neighborhood grows exponentially with dimension
- Very expensive in high dimension

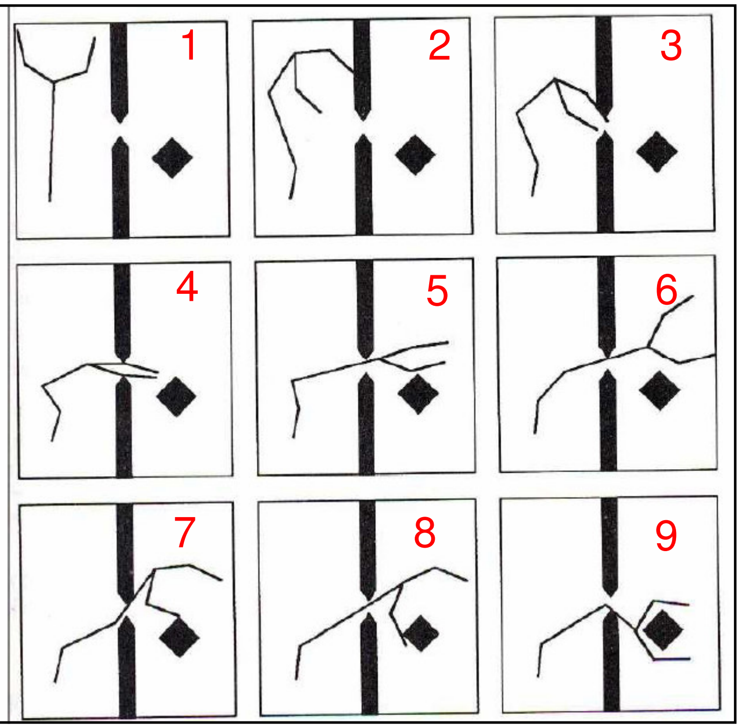
Solution:

- Evaluate only a random subset of *K* of the neighbors
- Move to the lowest potential neighbor

8 DOFs



10 DOFs

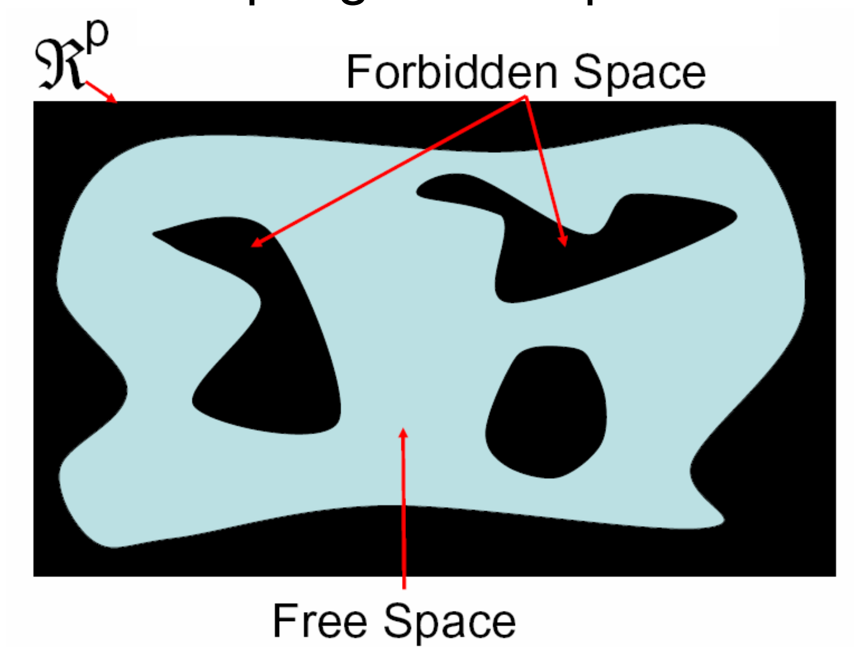


Approaches

- Basic approaches:
  - Roadmaps
    - Visibility graphs
    - Voronoi diagrams
  - Cell decomposition
  - Potential fields
- Extensions
  - Sampling Techniques
  - On-line algorithms

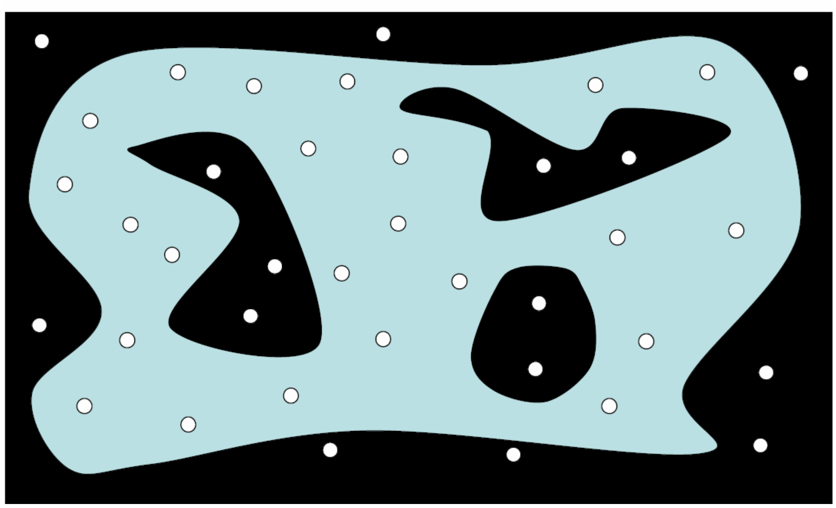
Completely describing and optimally exploring the C-space is too hard in high dimension + it is not necessary →
Limit ourselves to finding a “good” sampling of the C-space

Sampling Techniques



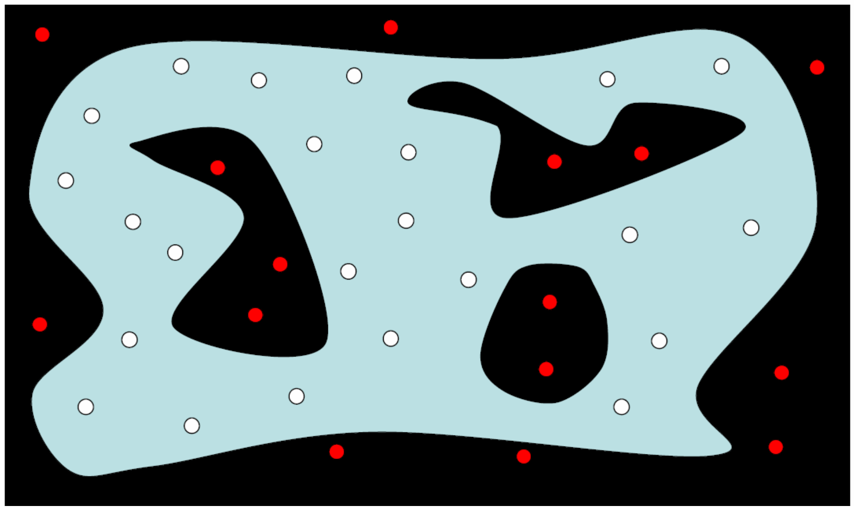
Sampling Techniques

Sample random locations



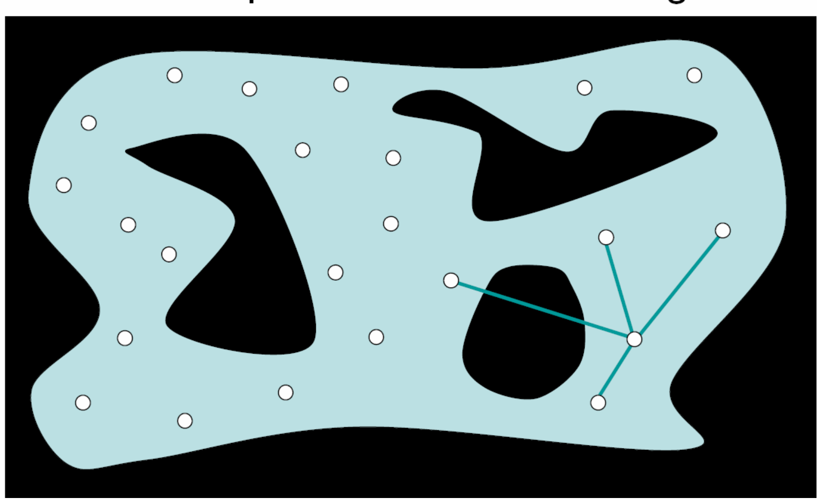
Sampling Techniques

Remove the samples in the forbidden regions



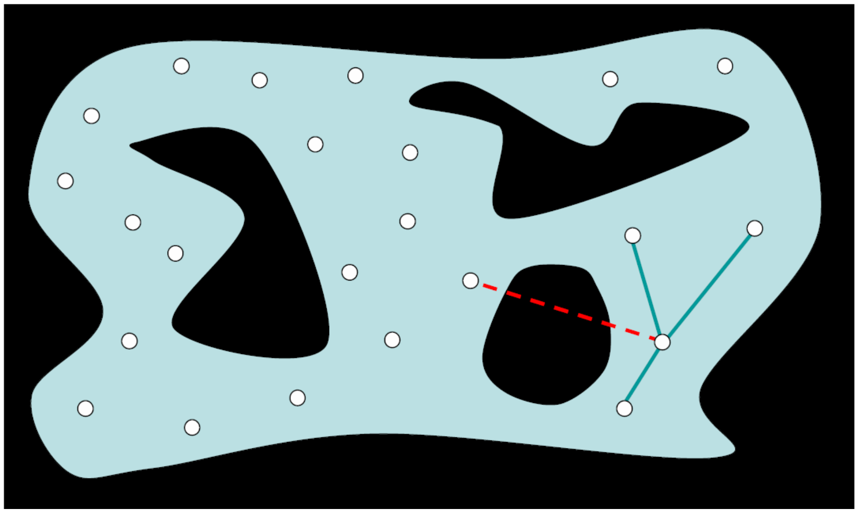
Sampling Techniques

Link each sample to its *K* nearest neighbors



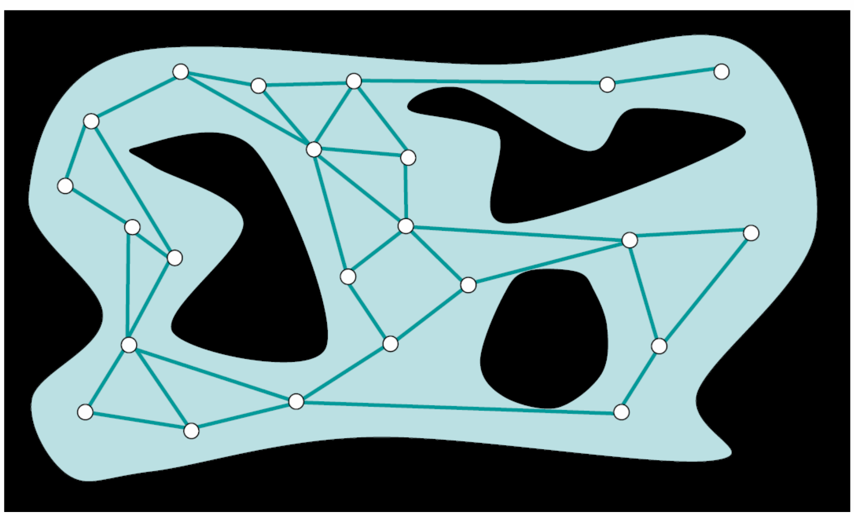
Sampling Techniques

Remove the links that cross forbidden regions



Sampling Techniques

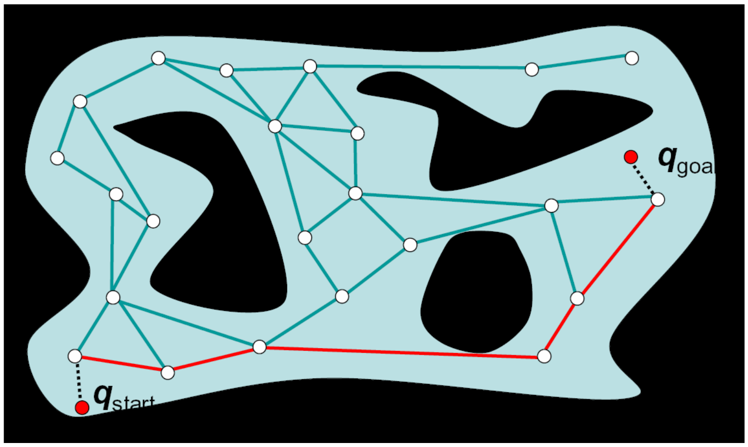
Remove the links that cross forbidden regions



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

Sampling Techniques

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

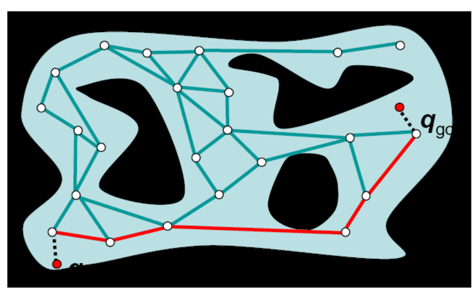


Sampling Techniques

Continuous Space

Discretization

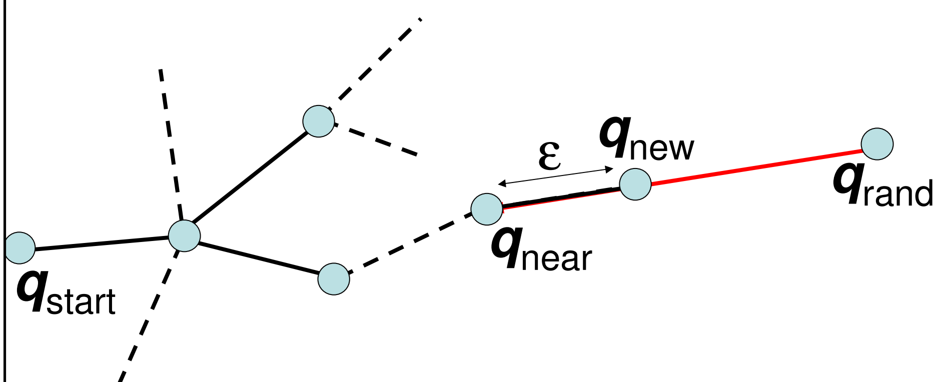
A* Search



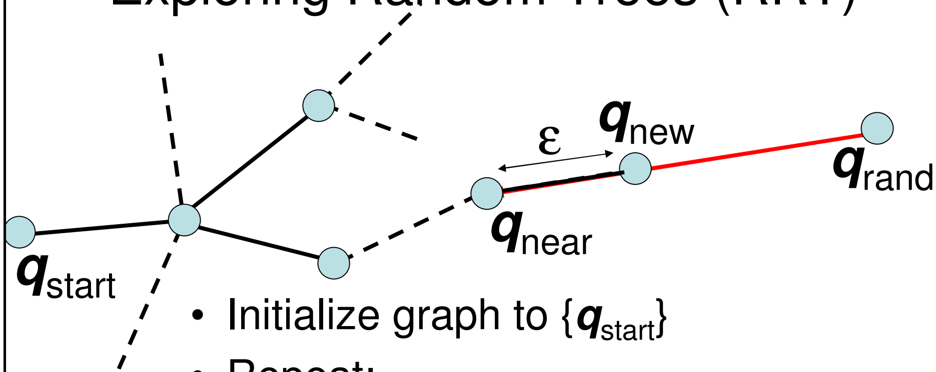
- “Good” sampling strategies are important:
  - Uniform sampling
  - Sample more near points with few neighbors
  - Sample more close to the obstacles
  - Use pre-computed sequence of samples

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

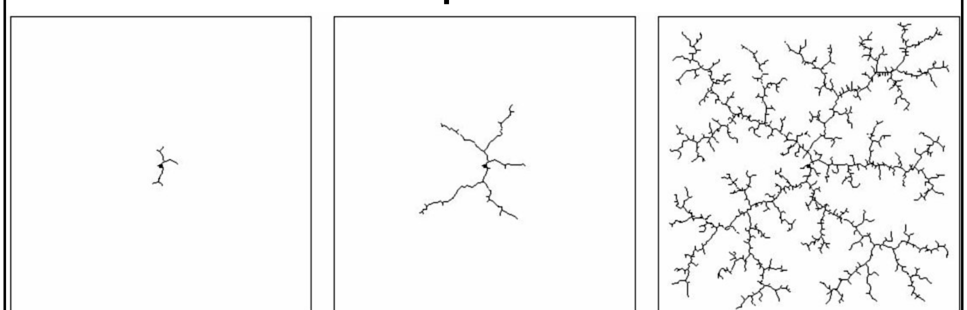


Even More Radical: Rapidly Exploring Random Trees (RRT)

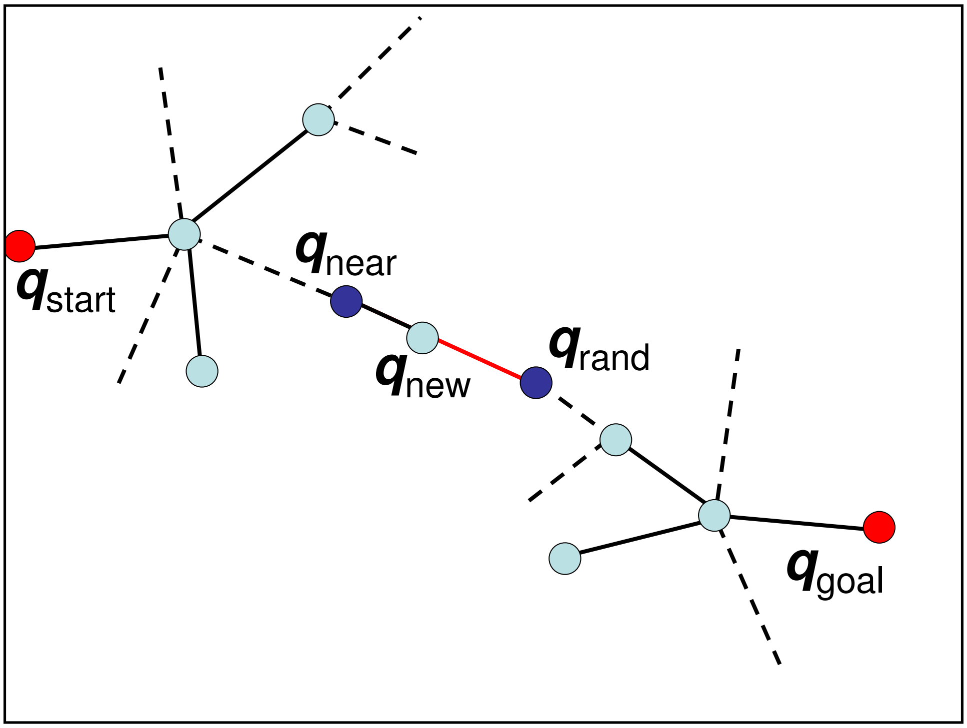


- Initialize graph to $\{q_{start}\}$
- Repeat:
  - Select random new sample q_{target}
  - Find closest node q_{near} to q_{target}
  - Create edge (q_{near}, q_{new}) if no collisions

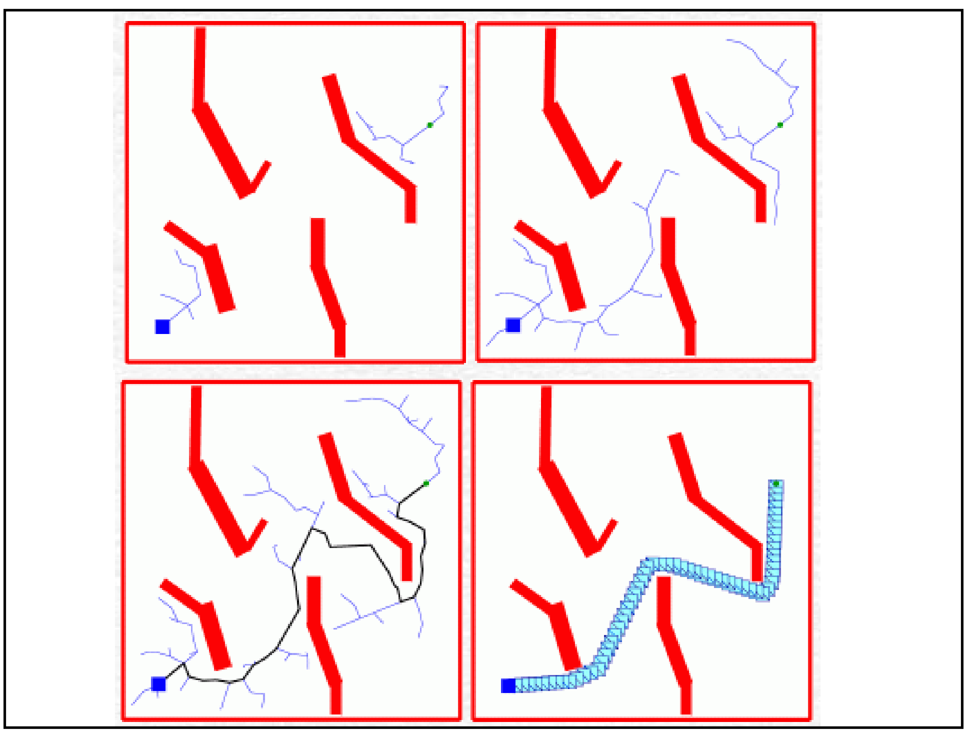
Properties

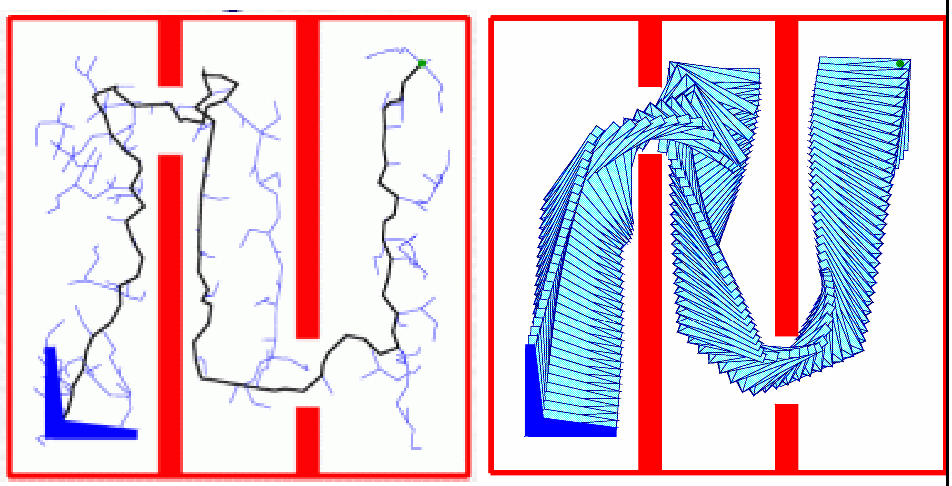


- Tends to explore the space rapidly in all directions
- Does not require extensive pre-processing
- Single query/multiple query problems
- Needs only collision detection test → No need to represent/pre-compute the entire C-space

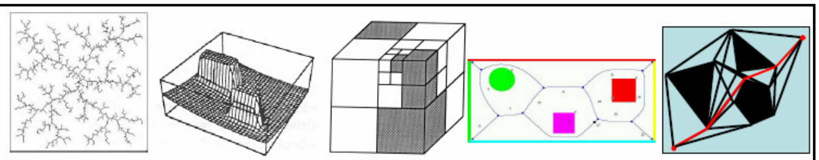


q_{start}
q_{near}
q_{new}
q_{rand}
q_{goal}

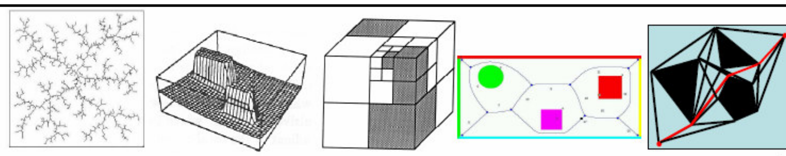




From Kuffner et al.



	Sampling	Potential Fields	Approx. Cell Decomposition	Voronoi	Visibility
Practical in ~2-D or 3-D	Y	Y	Y	Y	Y
Practical in >> 2-D or 3-D	Y	Y (using randomized version)	??	N	N
Fast	Y	Y	Y	In low dim.	In 2-D
Online Extensions	Y	Y	??	??	N
Complete?	Probabilistically complete	Probabilistically-resolution complete	Resolution-Complete	Y	Y



	Sampling	Potential Fields	Approx. Cell Decomposition	Voronoi	Visibility
Practical in ~2-D or 3-D	Y	Y	Y	Y	Y
Practical in >> 2-D or 3-D	Y	Y (using randomized version)	??	N	N
Fast	Y	Y	Y	In low dim.	In 2-D
Online Extensions					N
Complete?	Probabilistically complete	Probabilistically-resolution complete	Resolution-Complete	Y	Y

More exact/Complete

Faster/More practical in high dim.

- (Limited) background in Russell&Norvig Chapter 25
- Two main books:
  - J-C. Latombe. Robot Motion Planning. Kluwer. 1991.
  - S. Lavalle. Planning Algorithms. 2006. http://msl.cs.uiuc.edu/planning/
  - 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/