Robot Motion Planning

Department of Electrical and Electronics Engineering
Spring 2005
Dr. Afşar Saranlı

Thanks to Professor Andrew W. Moore (Carnegie Mellon University) http://www.cs.cmu.edu/~awm/tutorials
Also: Artificial Intelligence: A Modern Approach, 2nd Ed., Russel & Norvig
Also: Robot Motion Planning, Jean-Claude Latombe (Kluwer 1990)

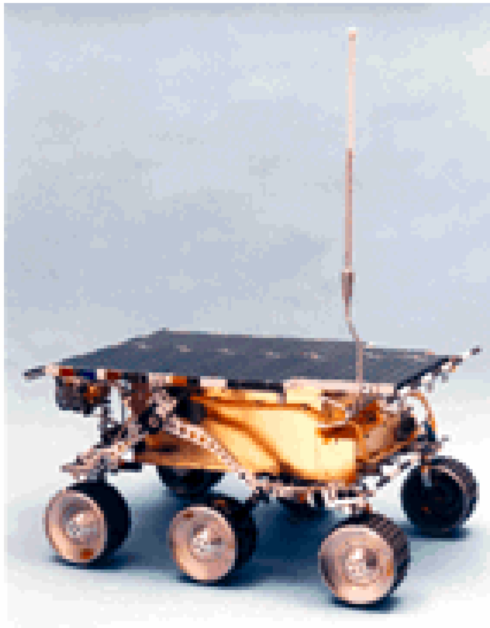
Outline

- Robotics Overview
- Spatial Reasoning
- Degrees of Freedom
- Robot Motion Planning
  - Configuration Space
  - Visibility Graph
  - Voronoi Diagrams
  - Cell Decomposition
  - Potential Methods
- Latombe Numerical Potential Field Method

Robotics

- Physically embodied agents
- *Sensors*
  (IR, range, touch, temp, cameras etc.)
- *Effectors*
  (Legs, wheels, joints, grippers, etc.)
- *Robot Programs* (AI)
  (localization, mapping, motion planning etc.)

Manipulators vs Mobile Robots



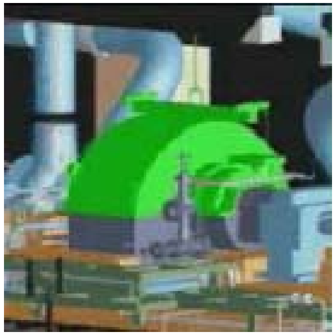
Robot Motion Planning

- ***Define the problem? Search space?***

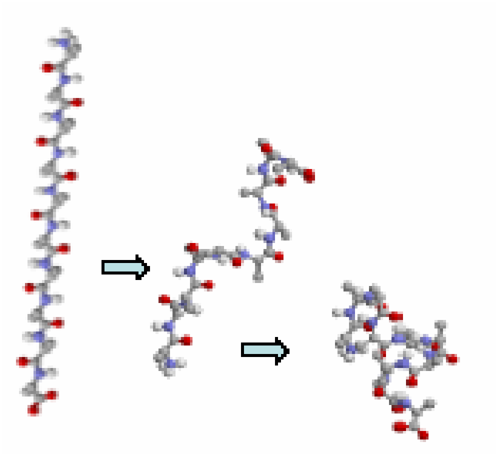
(Courtesy Howie Choset)

Robot Motion Planning

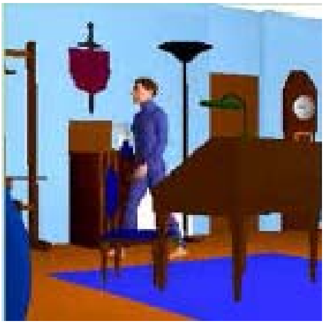
- ***Less obvious examples***



(Process Engineering / Design)



(Biology)



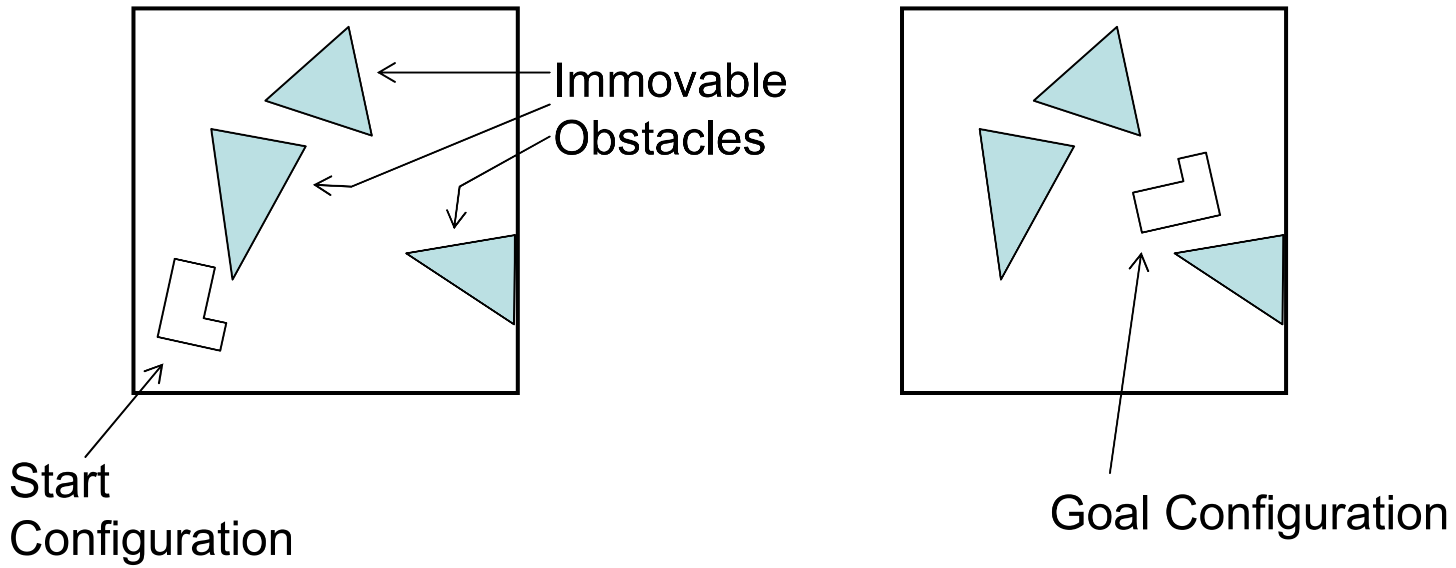
(Animation)

Think about Automated Reasoning

- We've already seen
  - State space search in discrete spaces
  - Reasoning with multiple agents
- Later (in this course) we'll see
  - Probabilistic Reasoning (e.g. with Markov Decision Processes)
- There is Reasoning with Constraints
- But *NOW* let's think about

SPATIAL REASONING

Spatial Reasoning

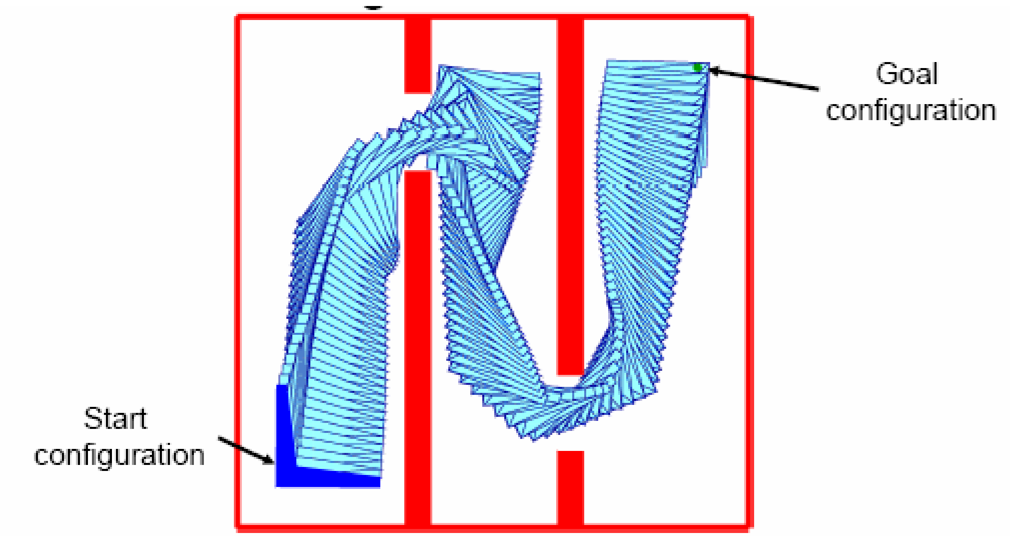


Can't we use our previous methods?

Discrete Search? – Not a discrete problem

Probabilistic? – Not really.

Robot Motion Planning vs Other Search

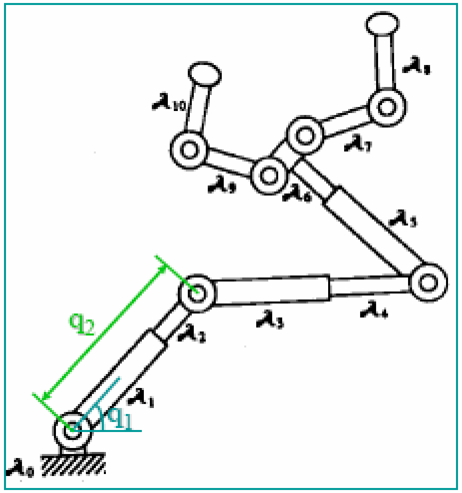


- *Discrete* or *Continuous*?
- *Deterministic* or *Stochastic*?
- What is the *search space dimension*?

Robots and Degrees of Freedom

For our purposes, a robot is:

A set of moving rigid objects called <u>LINKS</u> which are connected by <u>JOINTS</u>.



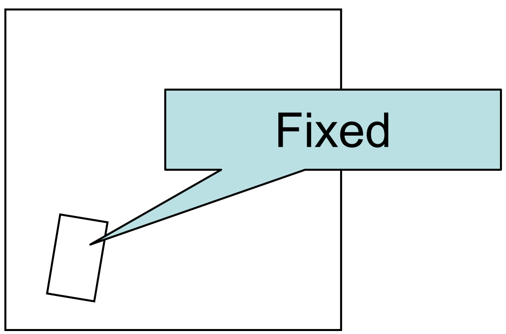
Typically, joints are *REVOLUTE* or *PRISMATIC*.

Such joints each give one *DEGREE OF FREEDOM*.

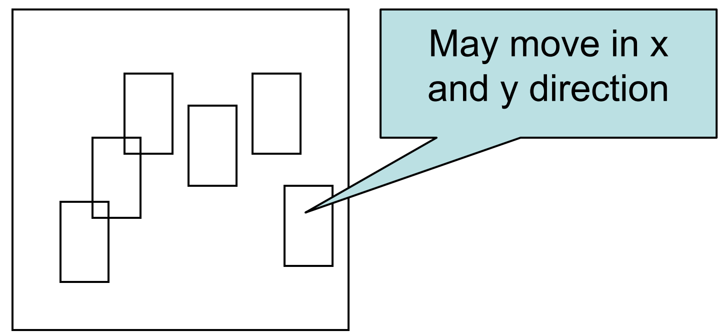
Given *p* DOFs, the configuration of the robot can be represented by *p* values $\boldsymbol{q} = (q_1\ q_2 \cdots q_p)$ where q_i is the angle or length of the *i*'th joint

Free Flying Polygons

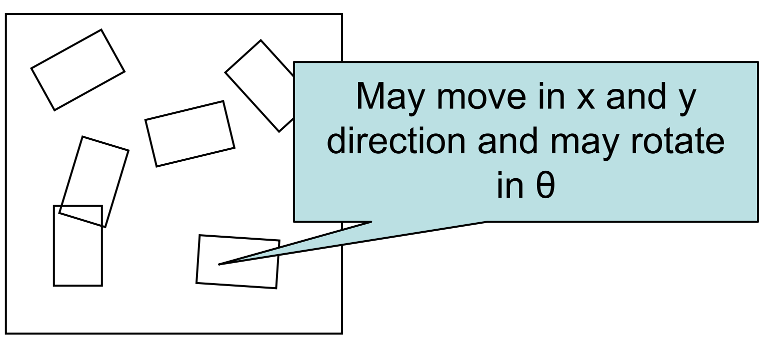
If part of the robot is fixed in the world, the joints are all the DOFs you're getting. But if the robot can be free-flying we get more DOFs.



0 DOFs



2 DOFs

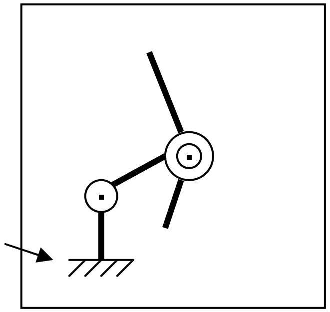


3 DOFs

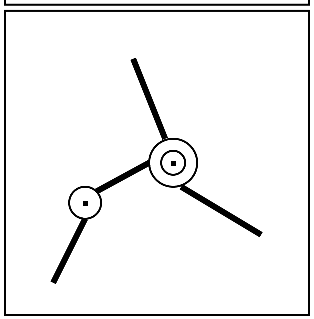
Question: How many DOFs for a polyhedron free flying in 3D space?

Other Examples

Fixed

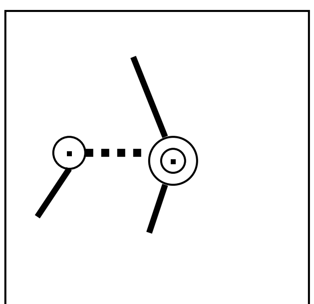


How many DOFs?



Free flying

How many DOFs?



Midline ■■■ must always be horizontal.

How many DOFs?

The configuration q has one real valued entry per DOF.

Robot Motion Planning

An important, interesting, spatial reasoning problem.

- Let *A* be a robot with *p* degrees of freedom, living in a 2-D or 3-D world.
- Let *B* be a set of obstacles in this 2-D or 3-D world.
- Call a configuration LEGAL if it neither intersects any obstacles nor self-intersects.
- Given an initial configuration q_{start} and a goal config q_{goal}, generate a continuous path of legal configurations between them, or report failure if no such path exists.

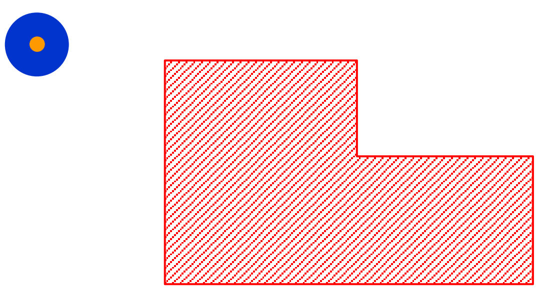
Configuration Space

- Is the *set of legal configurations* of the robot. It also defines the topology of continuous motions
- For rigid-object robots (no joints) there exists a transformation to the robot and obstacles that turns the robot into a single point.
- The *C-Space Transform*

C-Space Transform Examples

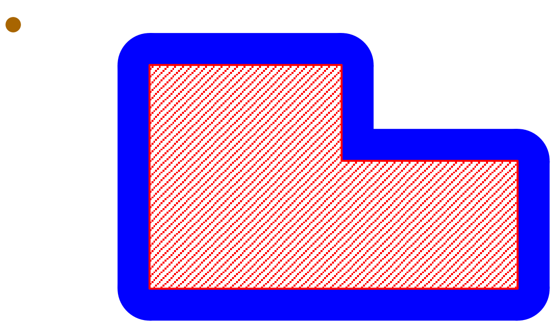
2-D World
2 DOFs

Where can I move this ***robot*** in the vicinity of this ***obstacle***?



…is equivalent to…

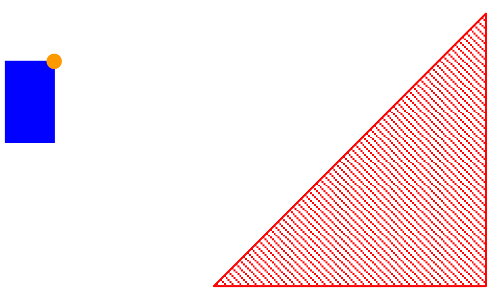
Where can I move this ***point*** in the vicinity of this ***expanded obstacle***?



C-Space Transform Examples

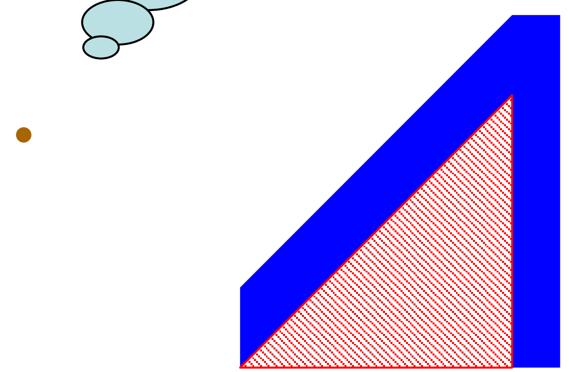
2-D World
2 DOFs

Where can I move this ***robot*** in the vicinity of this ***obstacle***?



…is equivalent to…

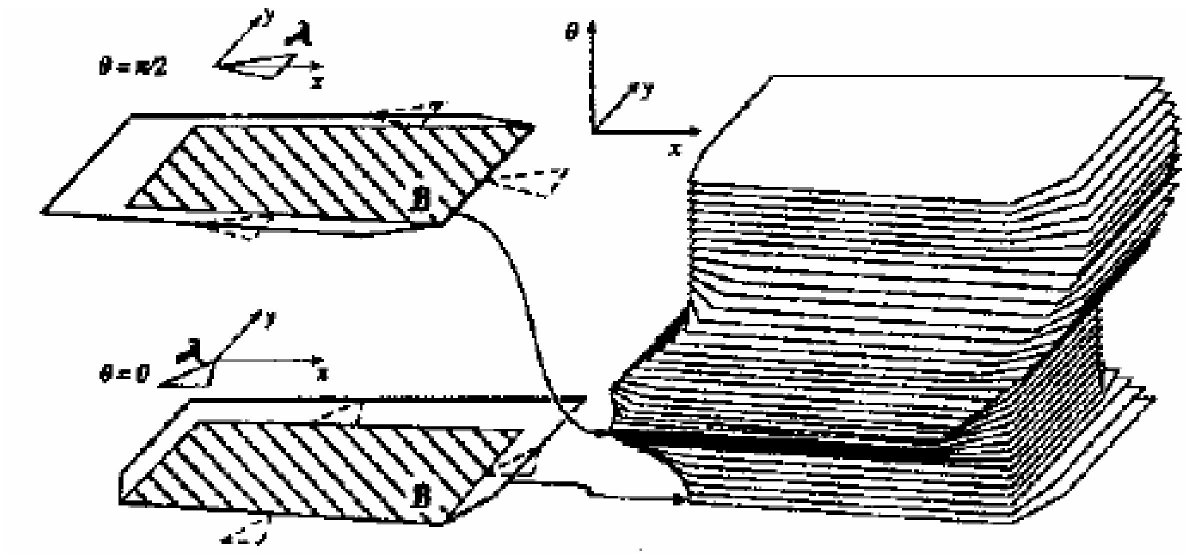
Where can I move this ***point*** in the vicinity of this ***expanded obstacle***?

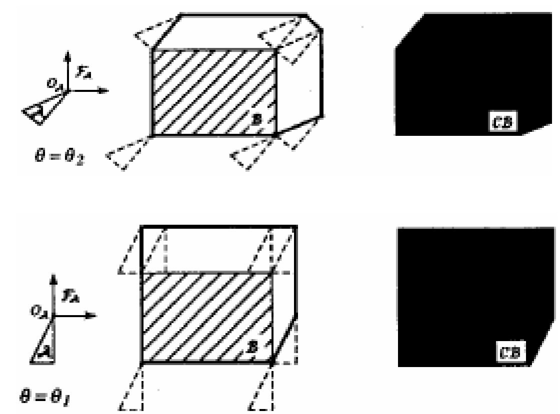


Assuming you're not allowed to rotate

C-Space Transform Examples

2-D World
3 DOFs





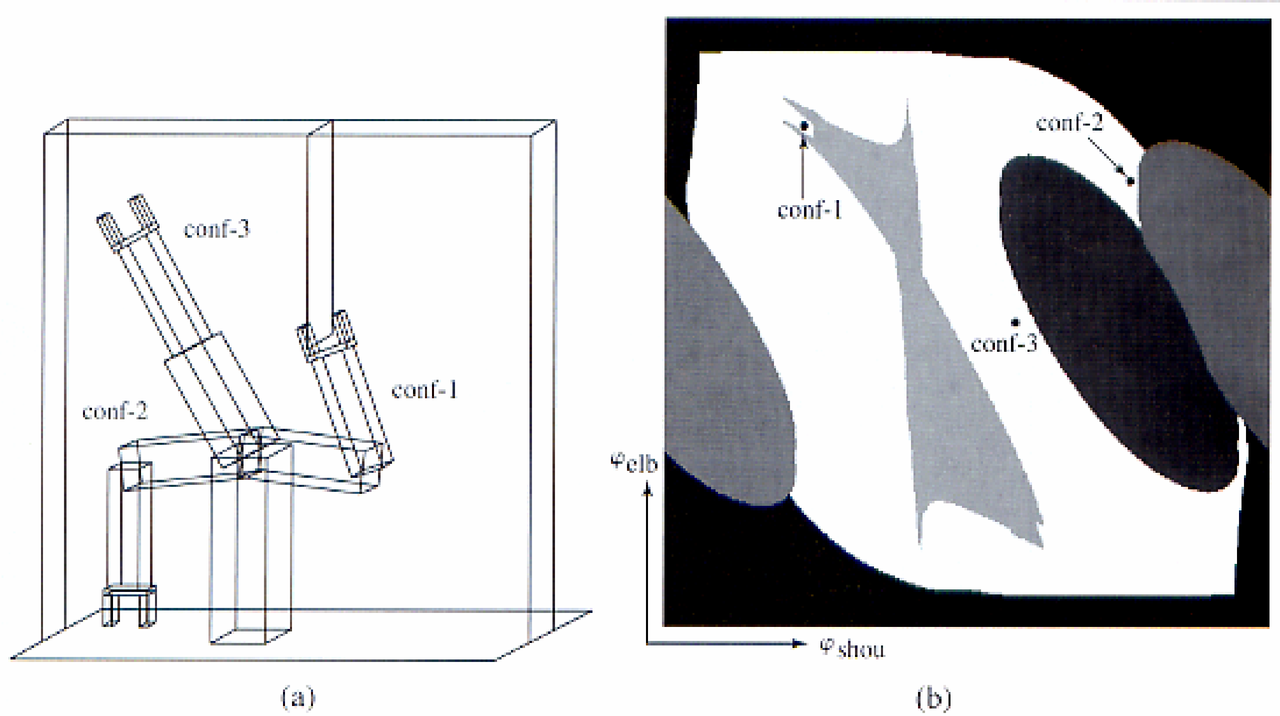
Examples from J.C. Latombe “Robot Motion Planning” (Kluwer 1990)

- We’ve turned the problem from “***Twist and turn this 2-D polygon past this other 2-D polygon***” into “***Find a path for this point in 3-D space past this weird 3-D obstacle***”.
- Why’s this transform useful?
- Because we can plan paths for points instead of polyhedra/polygons

Structure of C-Space

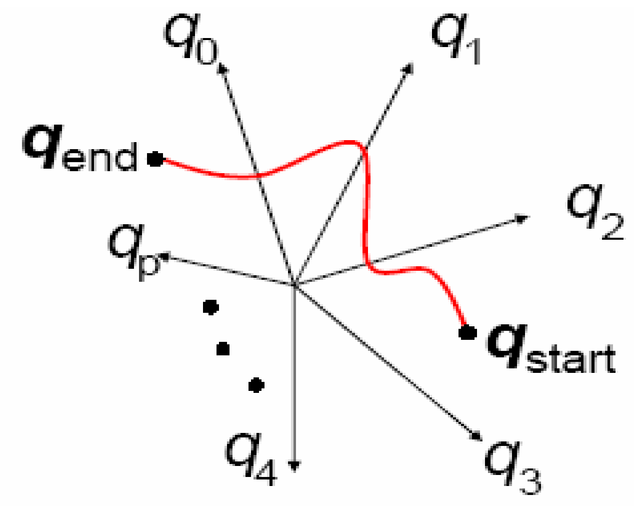
- Beware of the structure of C-Space:
- *Topology*
- The C-Space is not simple $\boldsymbol{R}^n$
- SO(2): Space of rotations in 2-D (Circle in the plane, θ=0 is the same as $\theta=2\pi$.)
- SO(3): Space of rotations in 3-D (Sphere in space)
- Etc.

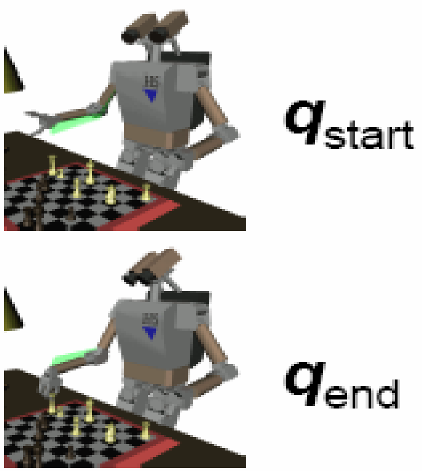
Other Cases of C-Spaces



- The obstacles in C-space can be very complex
- In all cases: The problem is reduced to finding the path of a point through C-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 ***q*** is legal if it does not cause the robot to intersect the obstacles
- Given start and goal configurations (***q***start and ***q***goal), find a continuous sequence of legal configurations from ***q***start to ***q***goal .
- Report failure if not path is found

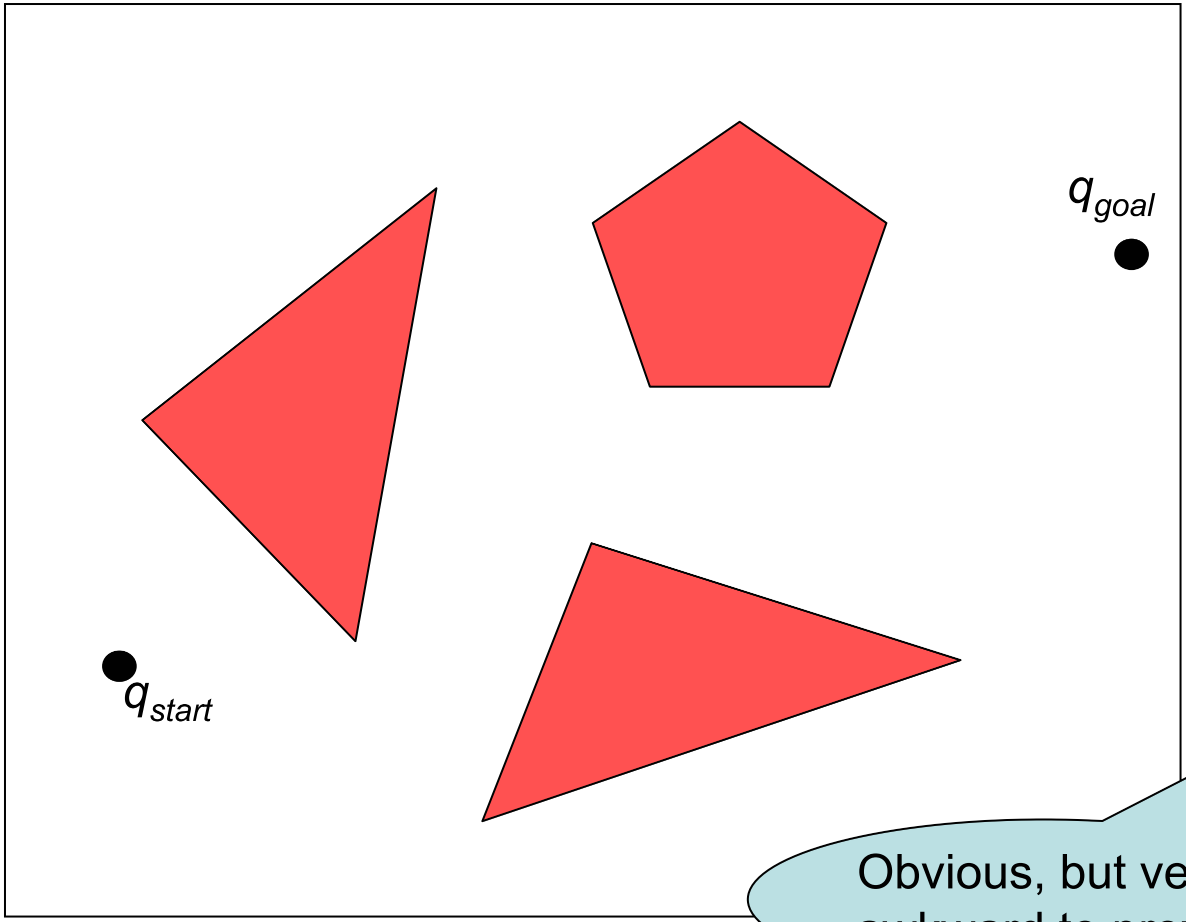
Motion Planning Research

…Has produced four kinds of algorithms.

The first is the Visibility Graph.

Visibility Graphs

Suppose someone gives you a C-SPACE with polygonal obstacles

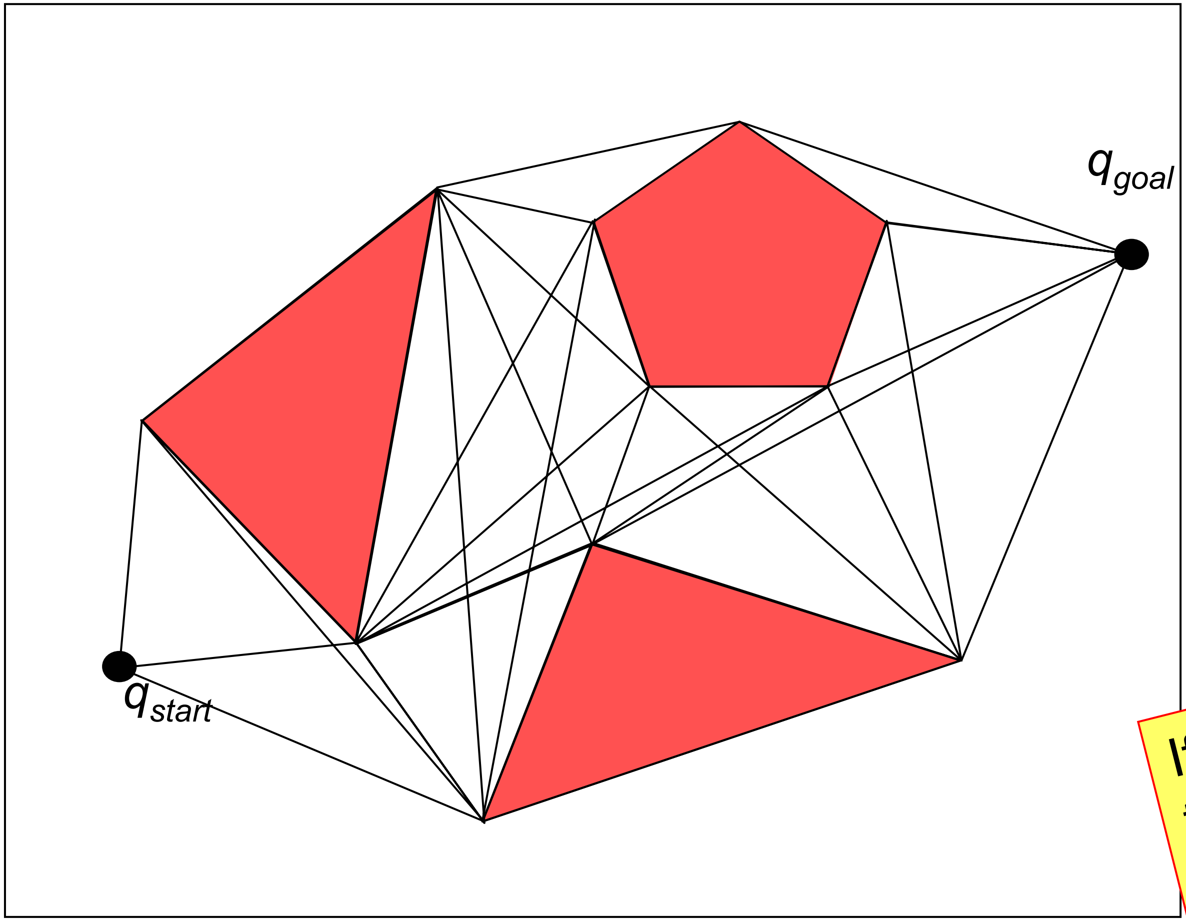


If there were no blocks, shortest path would be a straight line.

Else it must be a sequence of straight lines “shaving” corners of obstacles.

Obvious, but very awkward to prove

Visibility Graph Algorithm



1. Find all non-blocked lines between polygon vertices, start and goal.
2. Search the graph of these lines for the shortest path. (Guess best search algorithm?)

If there are n vertices, the easy algorithm is $O(n^3)$. Slightly tougher $O(n^2 \log n)$. $O(n^2)$ in theory.

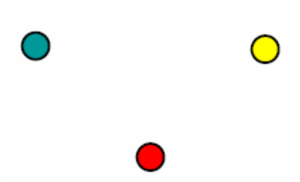
Visibility Graph Method - Complaints

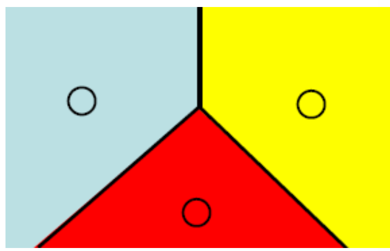
- Visibility graph method finds the shortest path.
- Bit it does so by skirting along and close to obstacles.
- Any error in control, or model of obstacle locations, and Bang! Collision with Obstacles!

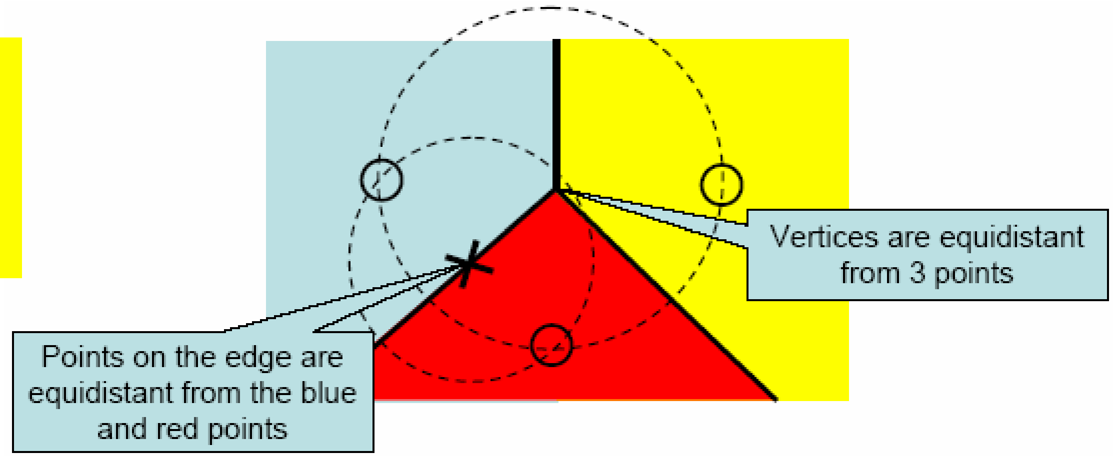
Who cares about optimality?

Perhaps we want to get a non-stupid path that steers as far from the obstacles as it can.

Voronoi Diagrams







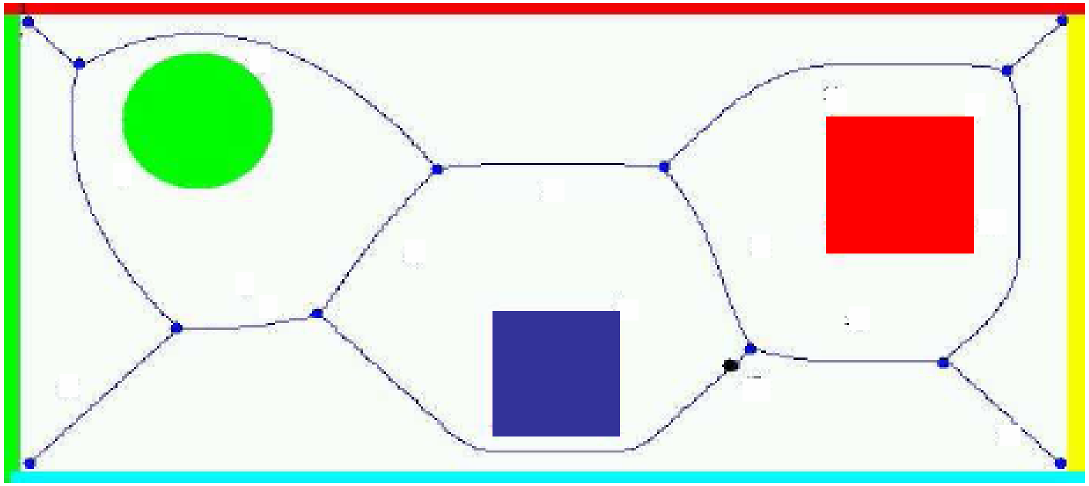
- Someone gives you some dots. Each with a different color.
- You color in the whole of 2-D space according to this rule:

  "The color of any given point equals the color of the nearest dot."

- The edges between your different regions are a ***Voronoi Diagram***.

Note: For *n* point in 2-D space the exact Voronoi diagram can be computed in time O(*n* log *n*).

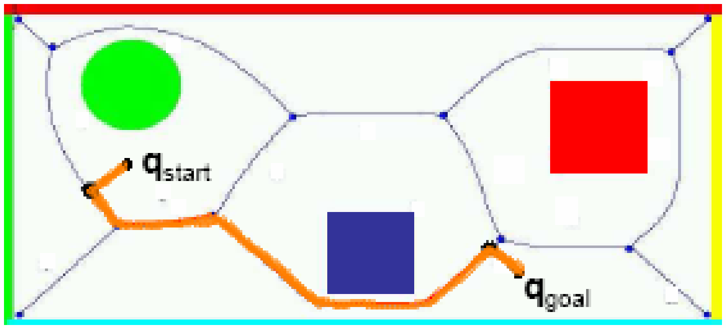
Voronoi Diagram for Polygons instead of Points



- Basic property: Points on the Voronoi Diagram are farthest (and hence safest) from the obstacles,
- Solution idea: Use the voronoi diagram edges instead of the visibility graph and search for a path!

Voronoi Diagram Methods for C-Space Motion Planning

- Compute the Voronoi Diagram of C-space.
- Compute shortest straightline path from start to any point on Voronoi Diagram.
- Compute shortest straightline path from goal to any point on Voronoi Diagram.
- Compute shortest path from start to goal along Voronoi Diagram.

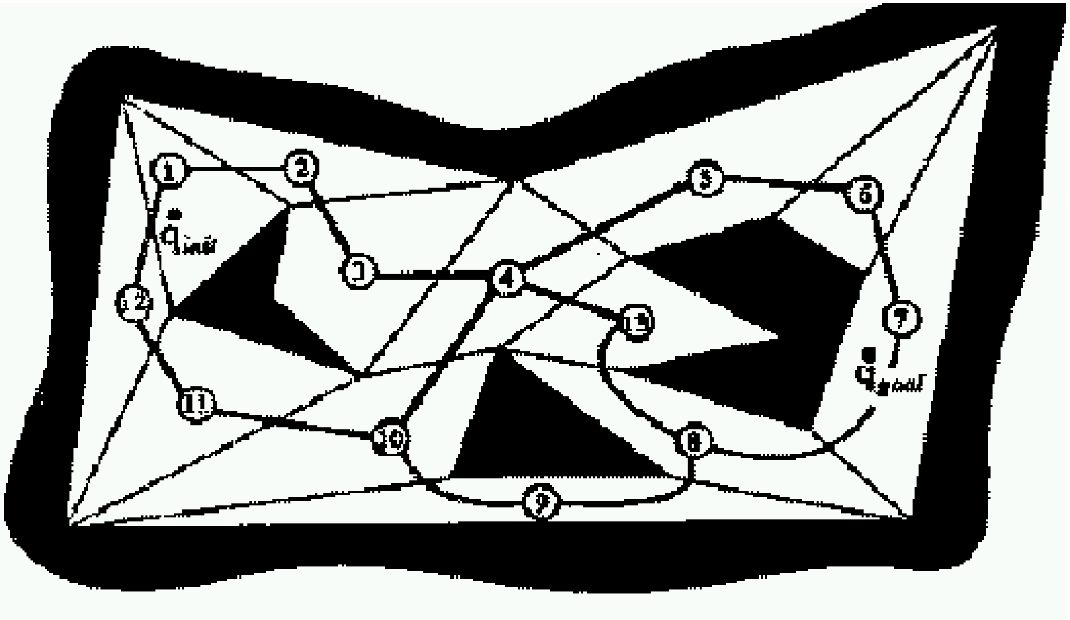


Voronoi Diagram Weaknesses

- Does not scale well to higher dimensional spaces,
- Difficult for arbitrary obstacle shapes in C-space (usually the case if converted from “work-space”)
- (However: Approximate algorithms exist)
- Can lead to paths that are too conservative,
- Can be unstable (small change in obstacle configuration may lead to large changes in voronoi diagram)

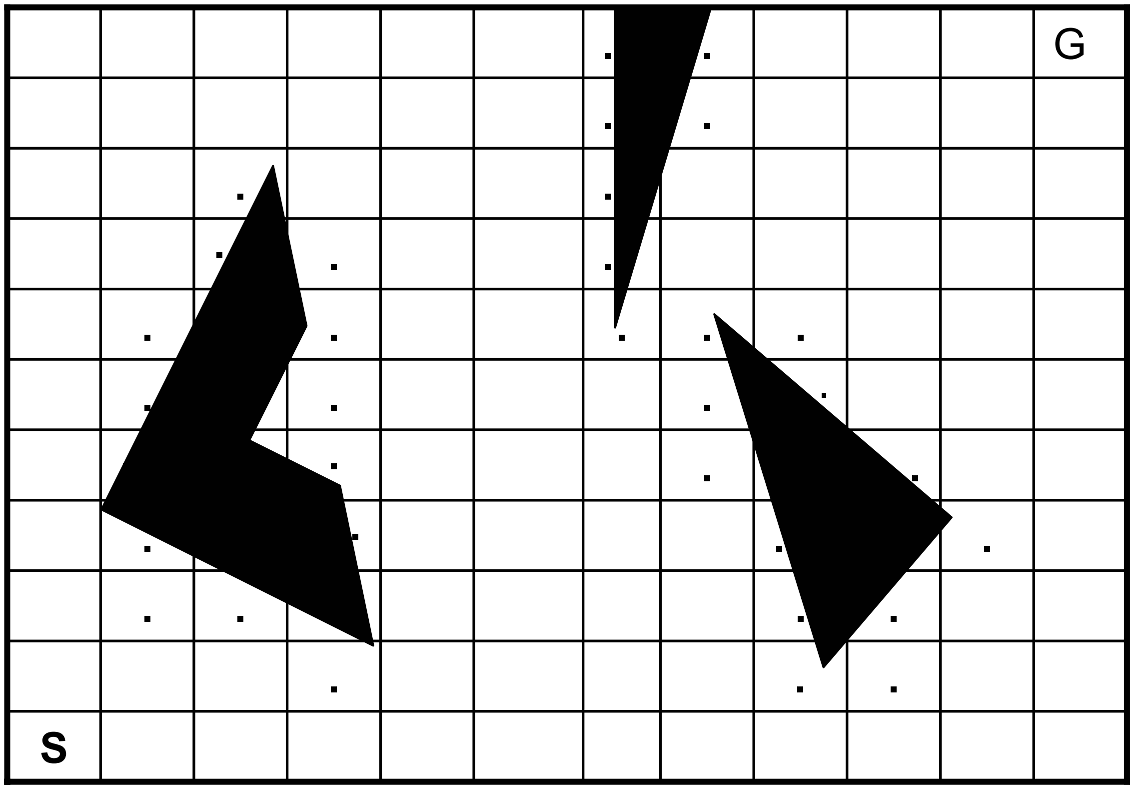
Cell Decomposition Methods

- Cell Decomp Method One: **Exact Decomposition**
- Break free space into convex exact polygons.



…But this is also impractical above 2-D or with non-polygons.

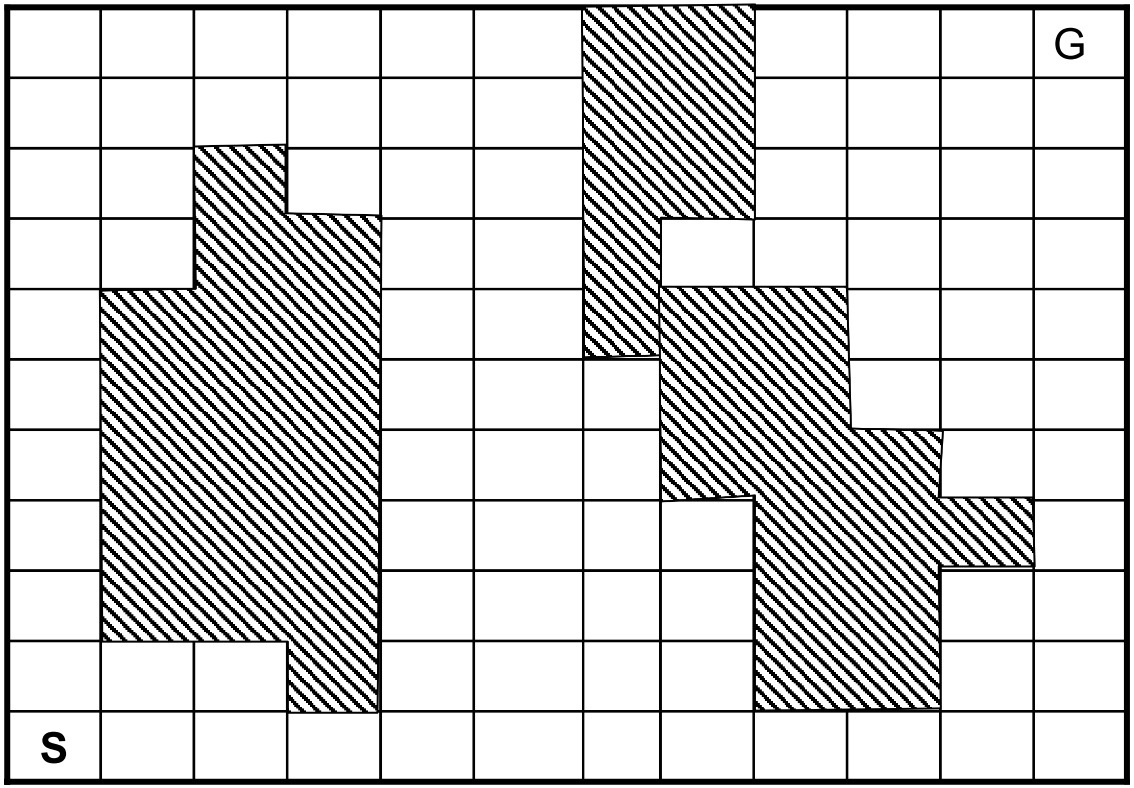
Approximate Cell Decomposition



- Lay down a grid
- Avoid any cell which intersects an obstacle
- Plan shortest path through other cells (e.g. with A*)

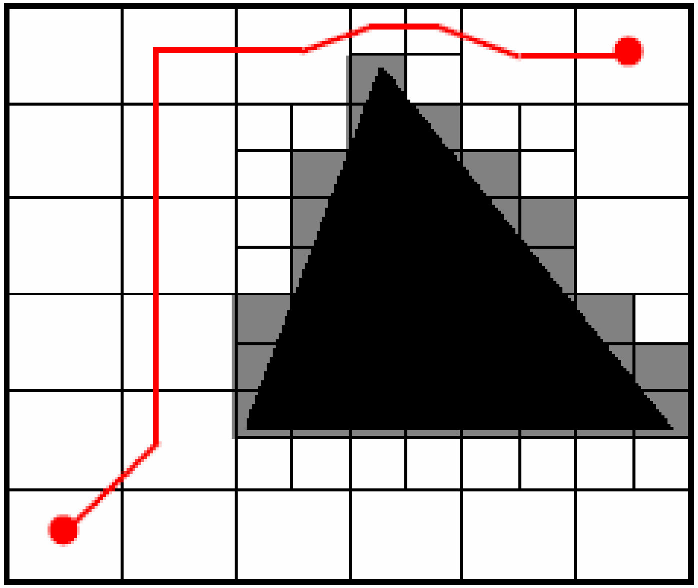
If no path exists, double the resolution and try again. Keep trying!!

Approximate Cell Decomposition

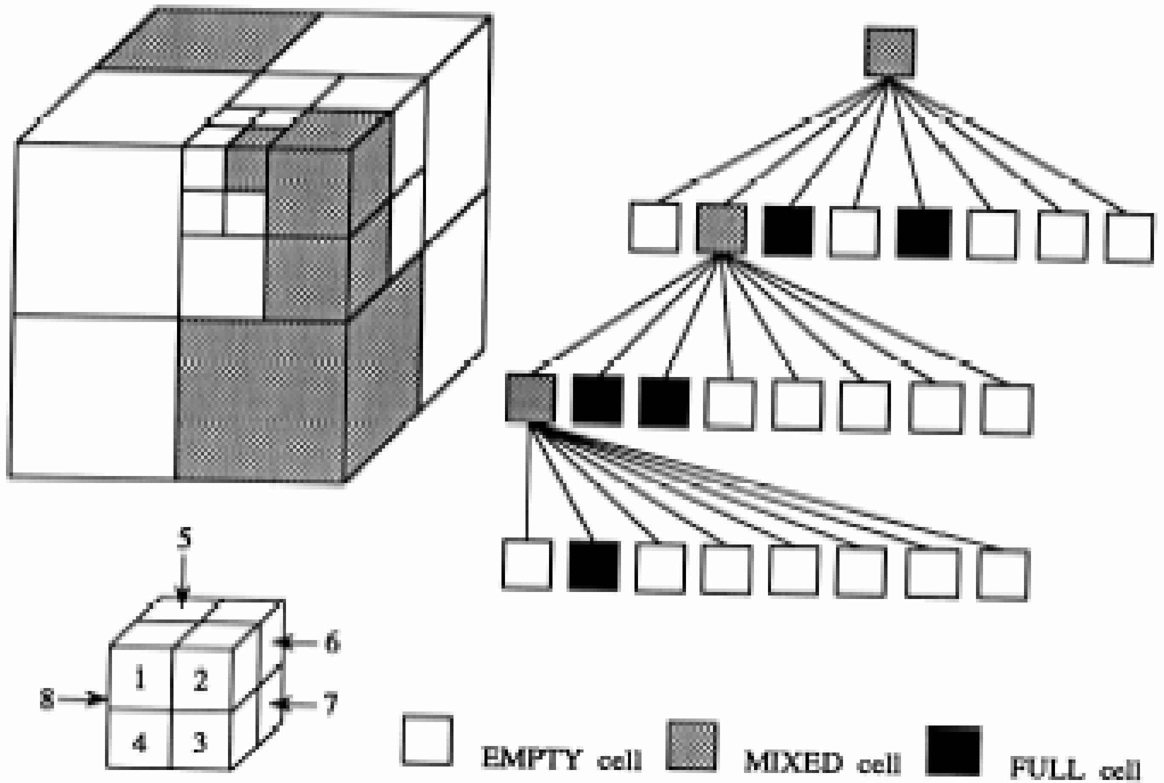


- Lay down a grid
- Avoid any cell which intersects an obstacle
- Plan shortest path through other cells (e.g. with A*)
- If no path exists, double the resolution and try again. Keep trying!!
- What are the problems?

Variable Resolution
"Approximate and Decompose"



Variable Resolution
"Approximate and Decompose"



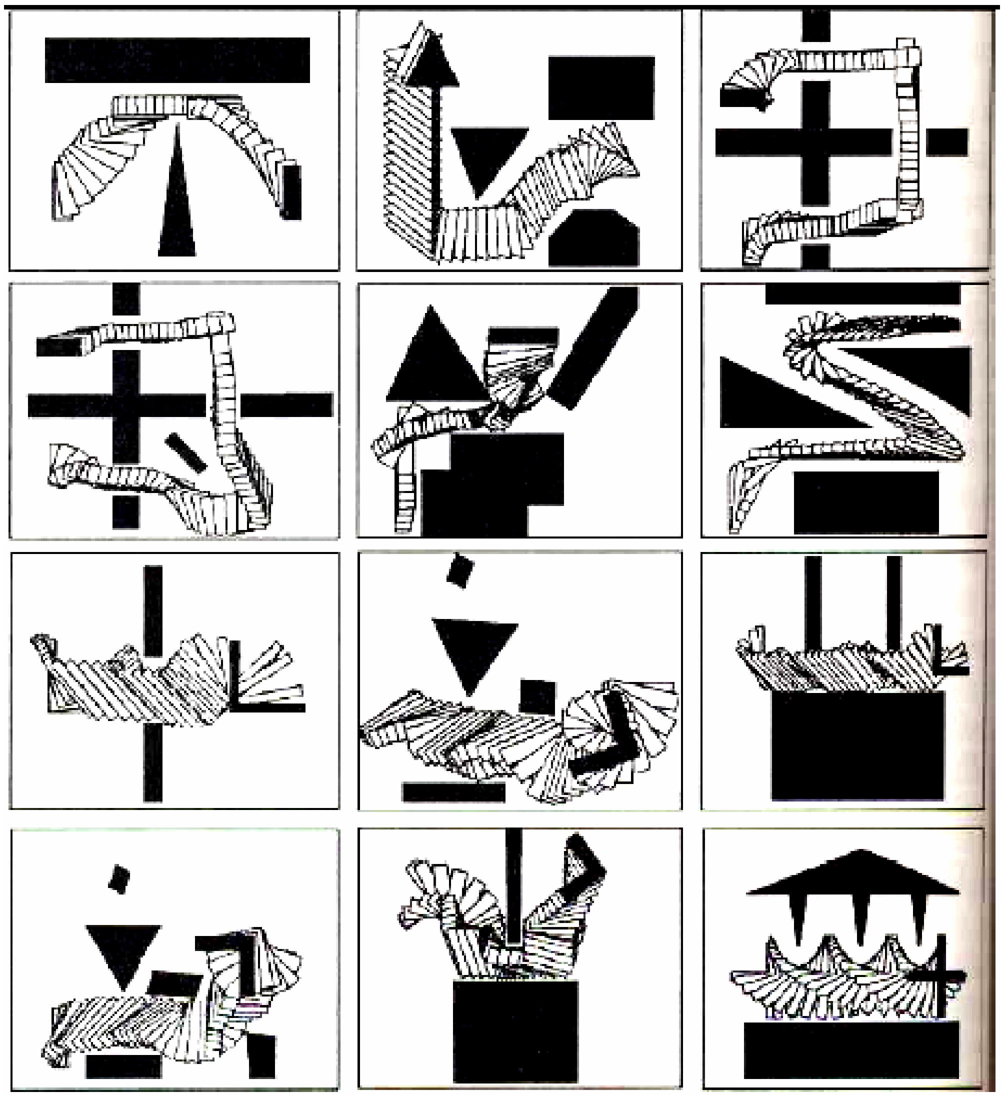
Approximate Cell Decomposition
The good and bad

- Not so many complaints. This is actually used in practical systems.

But

- Not exact (no notion of “best” path)
- Not complete: doesn’t know if problem actually unsolvable
- Still hopeless above a small number of dimensions?

Some Motion Examples



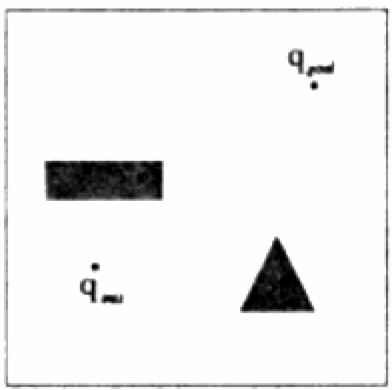
Examples from J.C. Latombe “Robot Motion Planning” (Kluwer 1990)

Potential Field Methods

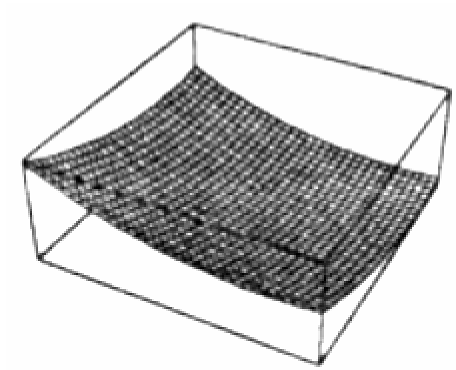
- Define a “potential function”…
- Should *decrease* as we approach goal,
- Should *sharply increase* as we approach an *obstacle,*
- Advantage: *Simple Motion Planner*!
  “Just follow the negative gradient of the potential function (steepest descent)”

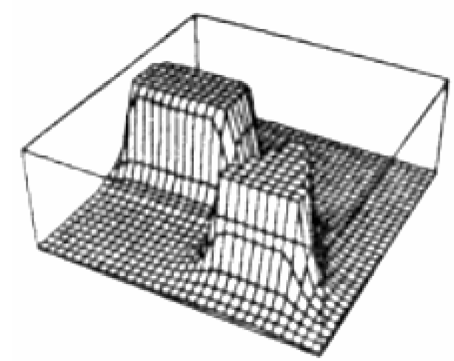
Potential Field Example

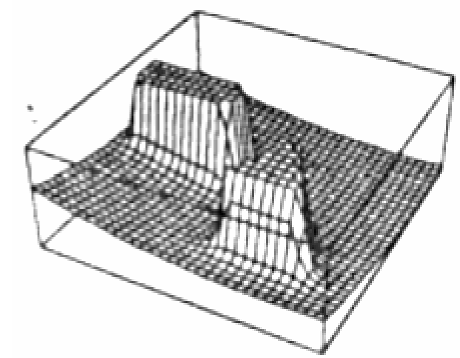
Given…



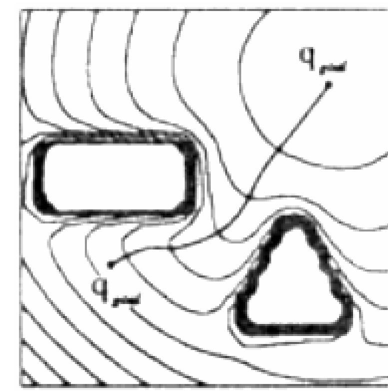
Define +

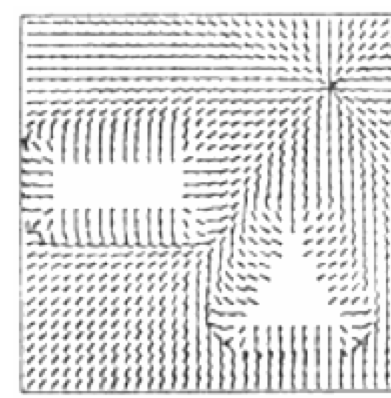




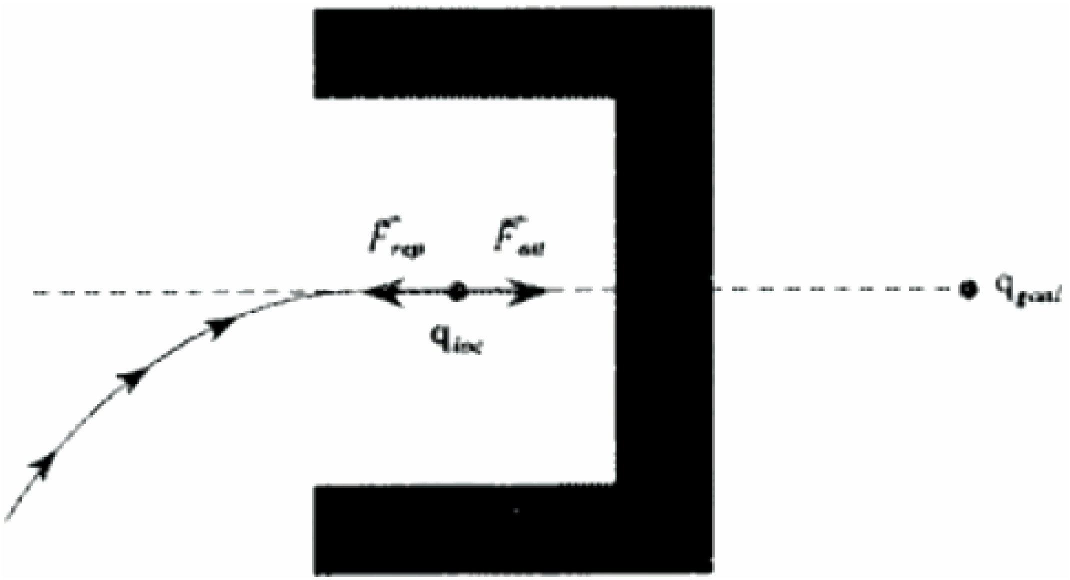


Follow steepest descent…



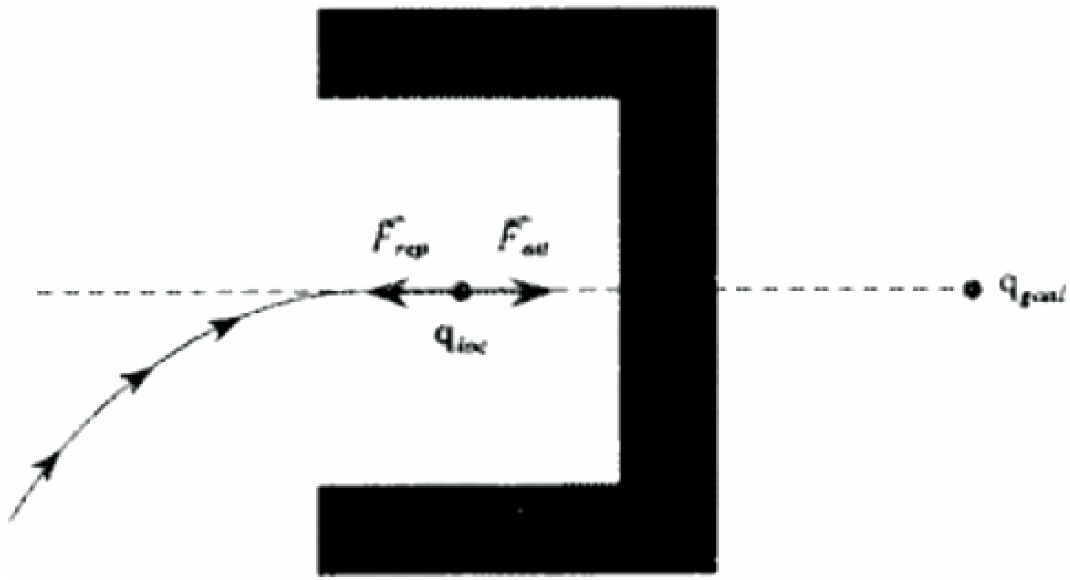


Potential Field Problems



- What is happening?
- How can we fix it?

Potential Field Problems



Solution I:

Use special local-minimum-free potential fields (Laplace equations can do this) – But very expensive to compute

Solution II:

When at a local minimum start doing some searching
- example soon

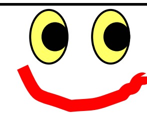
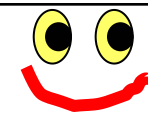
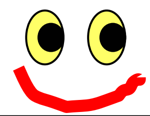
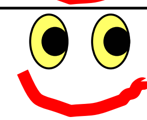
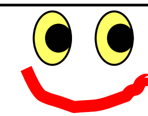
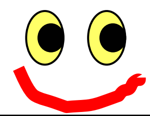
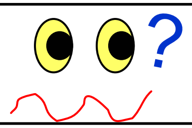
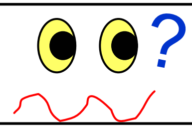
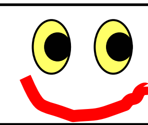
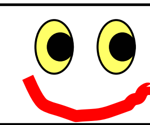
Comparison

	Potential Fields	Approx Cell Decomp	Voronoi	Visibility
Practical above 2 or 3 D?				
Practical above 8 D?				
Fast to Compute?				
Usable Online?				
Gives Optimal?				
Spots Impossibilities?				
Easy to Implement?				

Glimpse of State-of-the-Art

- Latombe's ***Numerical Potential Field Method***,
- Combines Cell Decomposition and Potential Fields
- Key insight: Compute an "optimal" potential field in world coordinate space (not config space)
- Define a C-space potential field in terms of world-space potential field

Comparison

	Potential Fields	Approx Cell Decomp	Voronoi	Visibility
Practical above 2 or 3 D?				
Practical above 8 D?				
Fast to Compute?				In 2-d
Usable Online?				
Gives Optimal?				In 2-d
Spots Impossibilities?				
Easy to Implement?