Configuration Space for Motion Planning

RSS Lecture 9

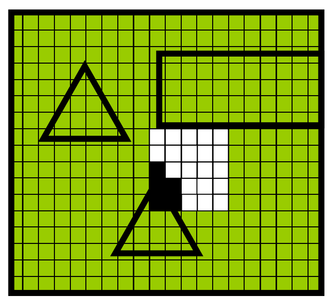
Wednesday, 5 March 2014

Prof. Seth Teller

Siegwart & Nourbahksh S 6.2

(Thanks to Nancy Amato, Rod Brooks, Vijay Kumar, and Daniela Rus for some of the figures)

Last Time



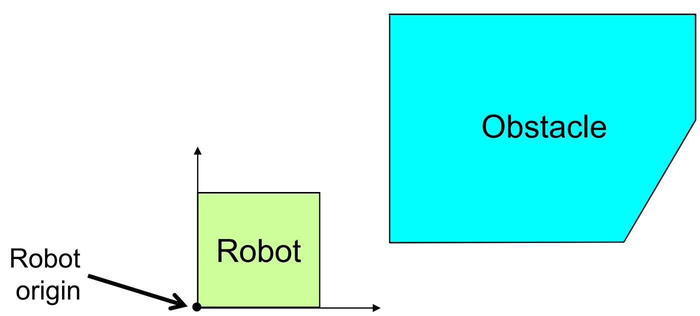
- Planning for point robots
  - Visibility graph method
  - Intermittent obstacle contact
- Ad hoc method of handling robots with positive area
  - Represent robot as a (2-DOF) disk
  - Discretize Cartesian space, conservatively
    But: some feasible paths
- Today: “configuration space” methods
  - Reason directly in a space with dimension = #DOFs
  - Transform there; solve problem; transform back

Today

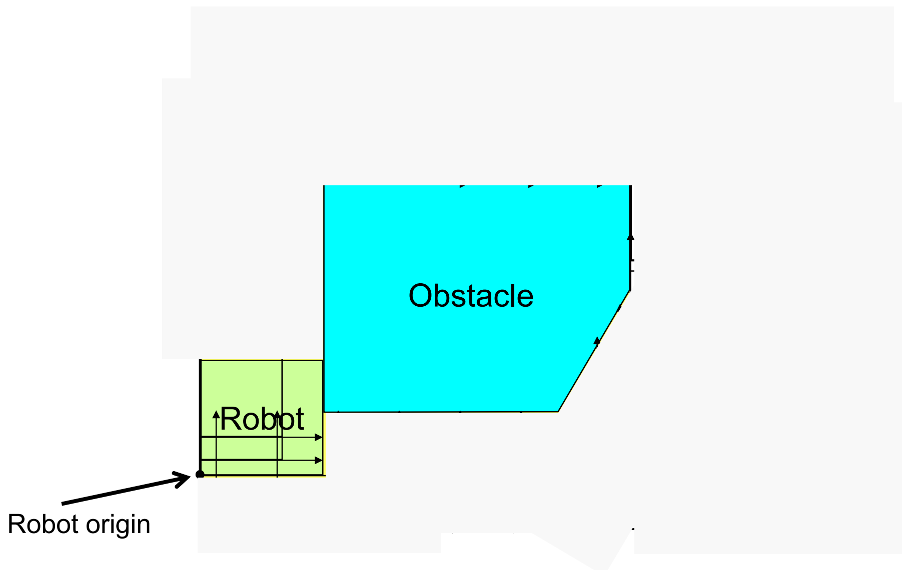
- Configuration space
  - Intuition
  - Preliminaries
    - Minkowski sums
    - Convexity, convex hulls
  - Definition
  - Construction
- Rigid (low-DOF) motion planning
  - Deterministic methods
- Articulated (high-DOF) motion planning
  - Randomized methods

Intuition

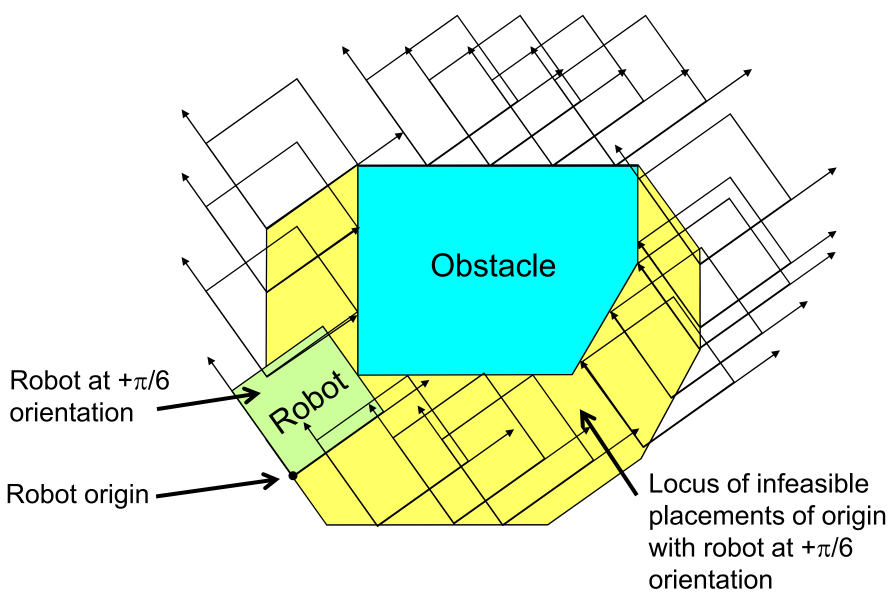
- Suppose robot can move only by translating in 2D
- How can it move in the presence of an *obstacle*?
  - Represent robot by its origin (how many DOFs?)
  - How to describe *infeasible placements* of robot origin?



Infeasibility Under Translation

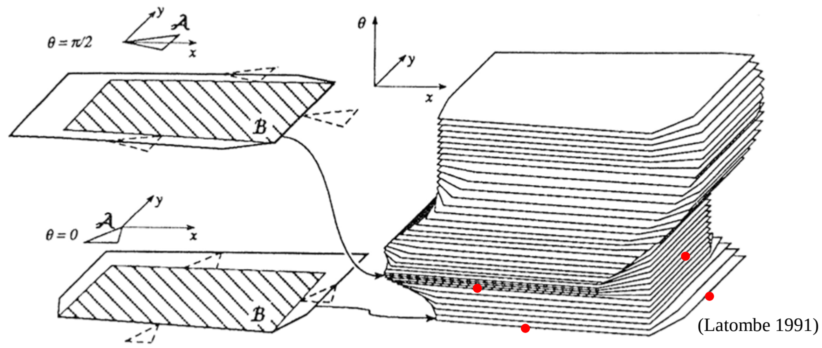


Infeasibility under 3-DOF Motion



Configuration Space

For a robot with *k* total motion DOFs, C-space is a coordinate system with *one dimension per DOF*



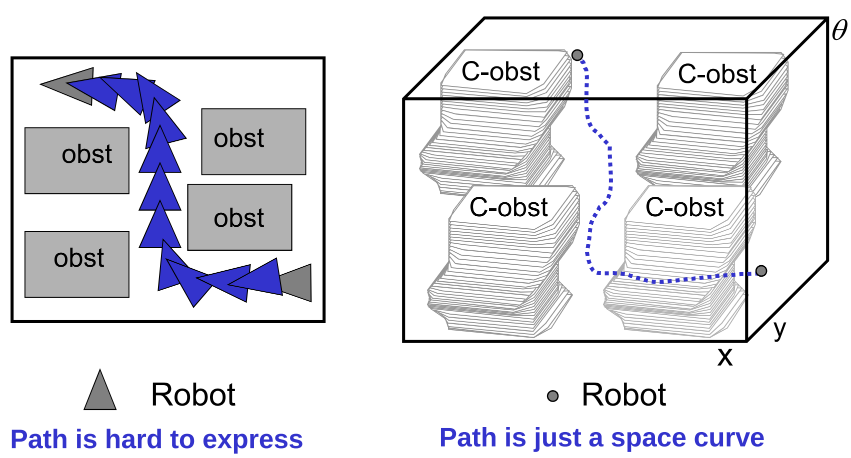
In C-space, a robot "pose" is simply

… and a workspace obstacle is a

Motion Planning Transformation

Workspace
(*x, y*)

C-space
(*x, y, θ*)



Configuration Space Idea

Robot geometry

Interaction (difficult to characterize)

Constraints due to obstacle geometry

Transformation to equivalent problem in higher dimension

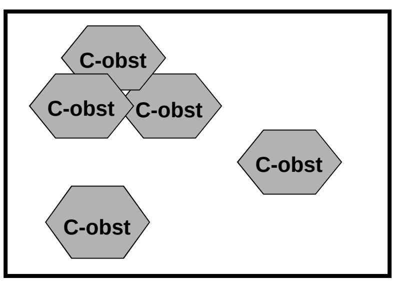
Transformation to equivalent problem in higher dimension

Point geometry

Interaction (simple to characterize)

Transformed constraints

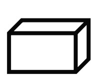
C-space Summary, Examples



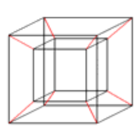
- Define space with one dimension per DOF of robot motion / pose
- Map robot to a *point* in this space
- C-space = *all* robot configurations
- C-obstacle = locus of *infeasible configurations* due to obstacle

Some example configuration spaces:

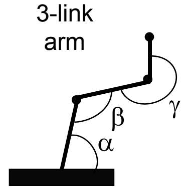
Translation + rotation in 2D



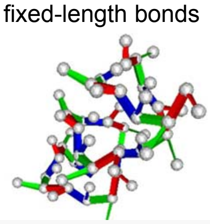
Translation + rotation in 3D



3-link arm

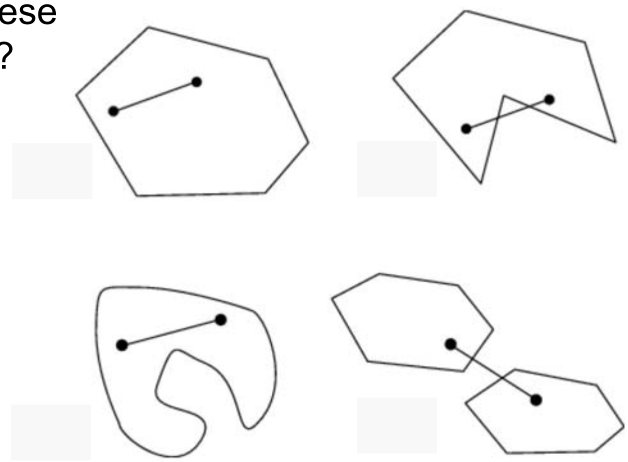


Molecule with *n* fixed-length bonds



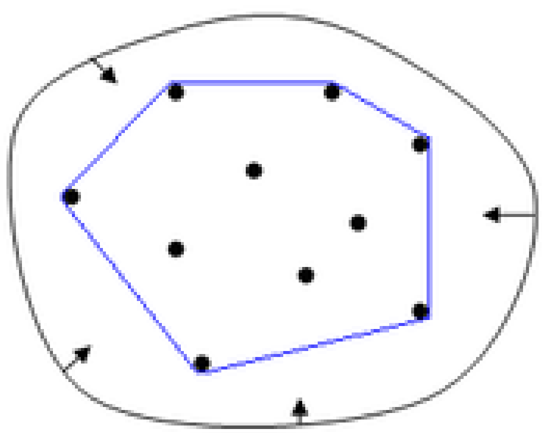
Convexity

- A set S is *convex* if and only if every line segment connecting two points in S is
- Which of these are convex?

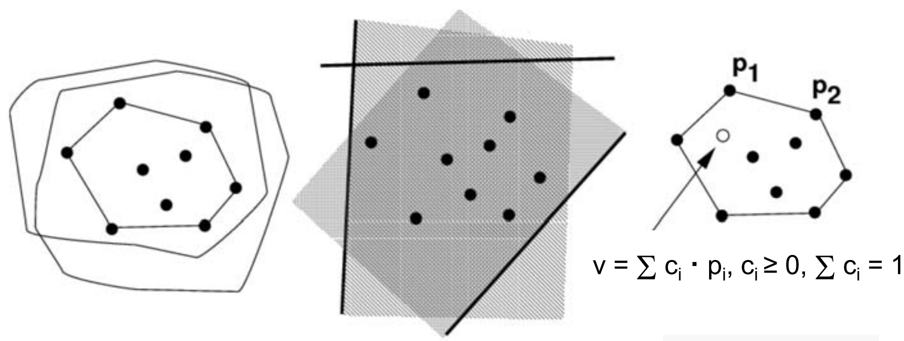


Convex Hull of a Set of Points

- Intuition: stretch a rubber band around point set



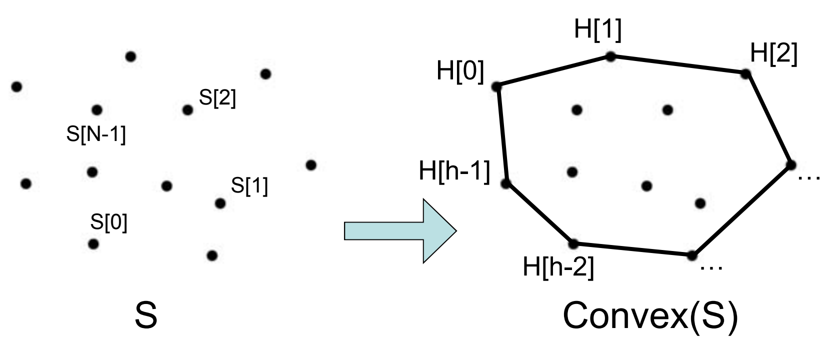
Convex Hull: Formal Definitions



- Which of these are constructive / algorithmic?

Computing 2D Convex Hull

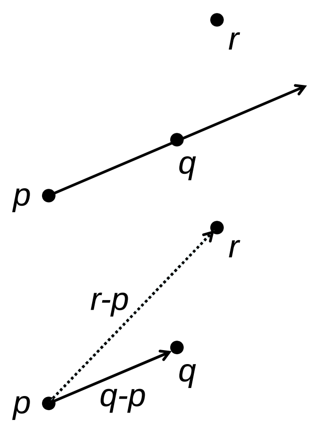
- Input: set S of N points (x_i, y_i) in 2D
- Output: polygonal boundary of convex hull of S



- How can Convex(S) be computed (efficiently)?

The Leftof Predicate

- Input: three points *p, q, r*
- Function Leftof (*p, q, r*) // argument order matters
- Output: 1 iff *r* is left of *directed line* $\overrightarrow{pq}$, otherwise -1



How to implement Leftof()?

1. Compute sign of determinant

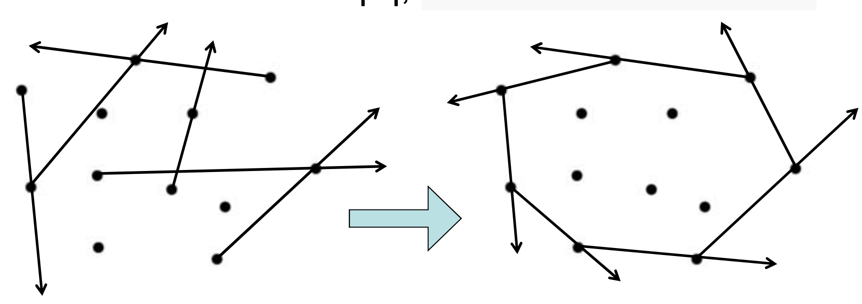
$$\begin{vmatrix} 1 & r_x & r_y \\ 1 & p_x & p_y \\ 1 & q_x & q_y \end{vmatrix}$$

2. Equivalently, find sign of *z* component of

Brute Force Solution

Identify point pairs that form edges of Convex(S)

I.e. for each pair p, q ∈ S, if ∀ r ∈ S – {p, q}, r lies left of the directed line $\overrightarrow{pq}$,

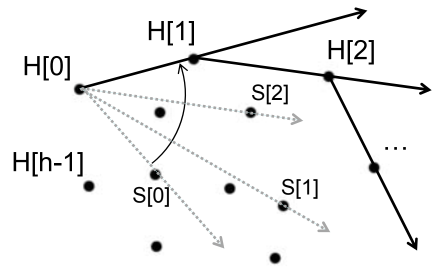


Running time for input of *n* points?

Can do better: O(n^2), O(n log n), O(nh), O(n log h) !

Jarvis March Algorithm

```
pivot = leftmost point in S;  i = 0   // leftmost point must be on convex hull
repeat
   H[i] = pivot          // store hull vertices in output point list H[i], 0 ≤ i < h
   endpoint = S[0]            // check candidate hull edge [pivot .. endpoint]
   for j from 1 to |S|-1
      if (Leftof (pivot, endpoint, S[j]))
         endpoint = S[j]
   pivot = endpoint; i++
until endpoint == H[0]
```

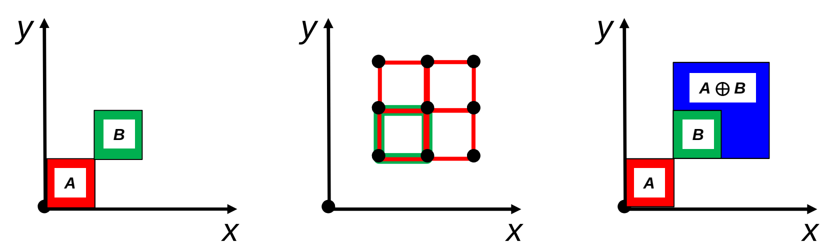


Outer loop runs times;
inner loop does work
Running time for input set of *n* points? "Output-sensitive" algorithm.

(With *h* =)

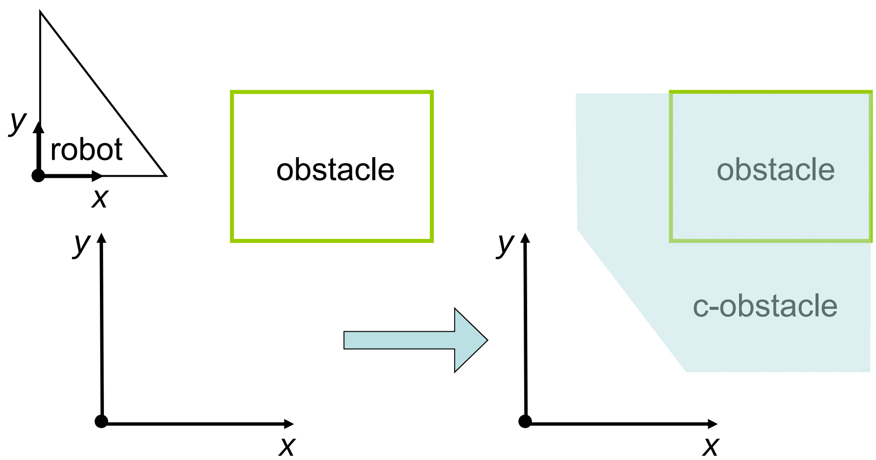
Minkowski Addition

- Given two sets $A,B \in R^d$, their *Minkowski sum*, denoted $A \oplus B$, is the set $\{ a + b \mid a \in A, b \in B \}$
  - Result of *adding* each element of *A* to each element of *B*
- If A & B convex, just add vertices & find convex hull:



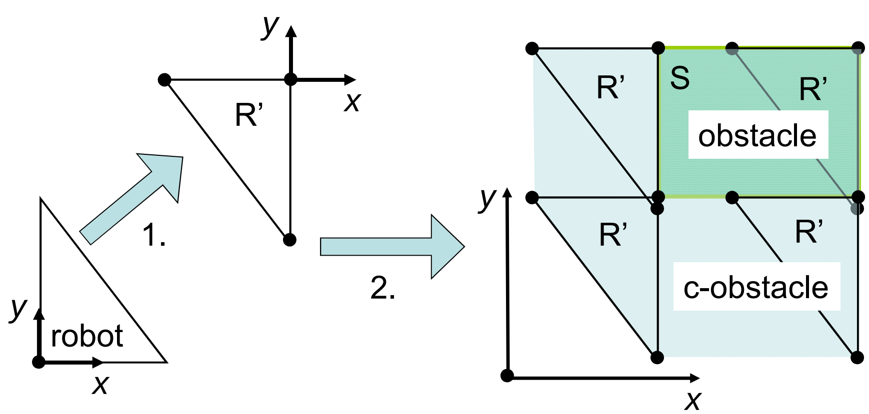
Computation of C-obstacles

- Inputs: robot polygon R and obstacle polygon S
- Output: c-space obstacle c-obstacle(S, R)



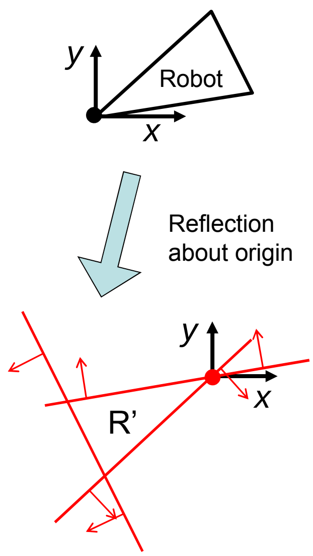
C-obstacle Computation

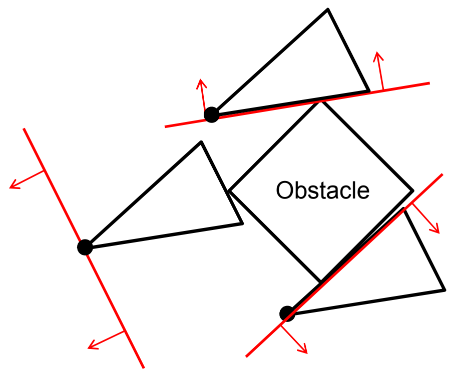
1. Reflect robot R about its origin to produce R’
2. Compute Minkowski sum of R’ and obstacle S



Sanity check: shaded region is infeasible for

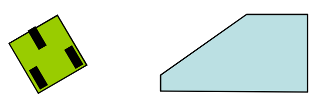
Why *Reflect* the Robot?

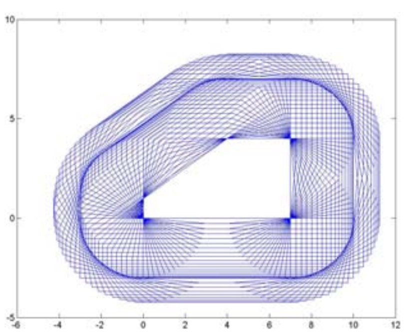


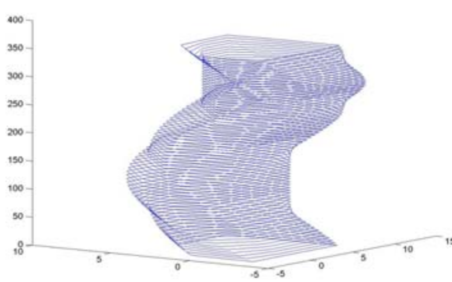


Shown above are contacts between robot *edges* and obstacle *vertices*. What happens when robot *vertices* contact obstacle *edges*?

C-obstacles with Rotations



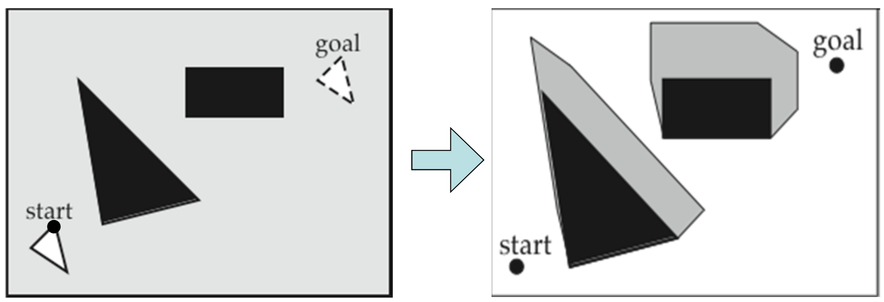




How do we compute this object?

Back to Motion Planning

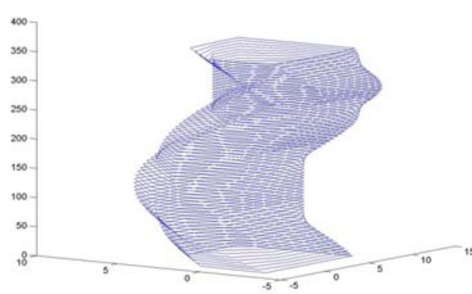
- Given robot and set of obstacles:
  - Compute C-space representation of obstacles
  - Find path from robot start pose to goal pose (point)



- Unfortunately, we have a rather serious problem:
  - We have constructed a representation of the *obstacles*
  - But we need to search a representation of the *freespace!*

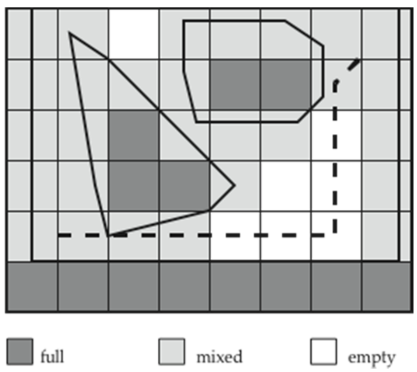
Computational Complexity

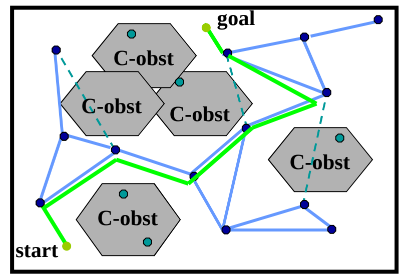
- The best deterministic motion planning algorithm known requires *exponential* time in the C-space dimension [Canny 1986]
- D goes up fast – already D=6 for a rigid body in 3-space; articulation adds many more DOFs
- Simple obstacles have complex C-obstacles
- Impractical to compute explicit representation of freespace for robot with many DOFs
- What to do?



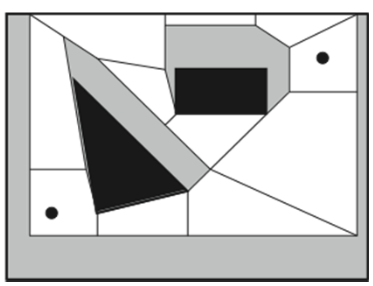
Strategies

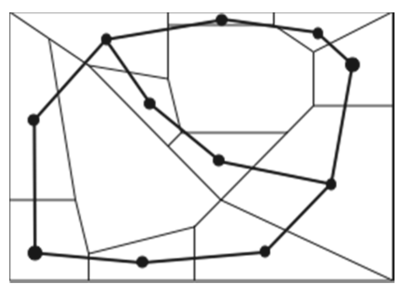
- Approximate: use regular subdivision of freespace
- Randomize: sample and evaluate C-space poses
- Sacrifice s for gains in

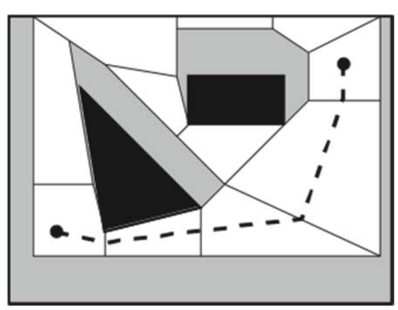




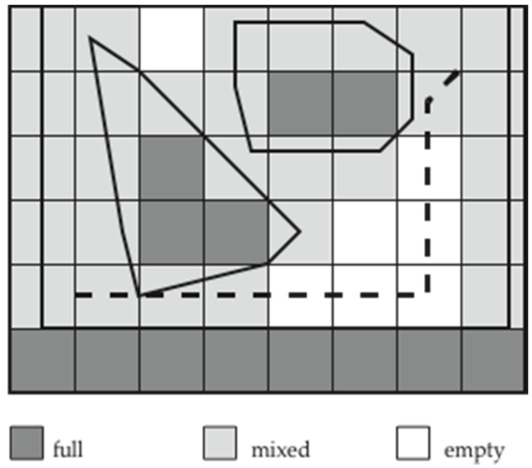
Example: Exact Decomposition







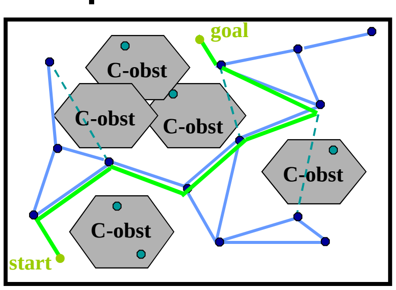
Approximate Cell Decomposition



- Advantage: recasts complex original problem as search within space of many, simpler motion plans

Probabilistic Road Maps for Motion Planning [Kavraki et al. 1996]

C-space



<u>Roadmap Construction (Pre-processing)</u>

1. Randomly generate robot configurations (nodes)
   - Discard invalid nodes (how?)
2. Connect pairs of nodes to form **roadmap edges**
   - Use simple, deterministic *local planner*
   - Discard invalid edges (how?)

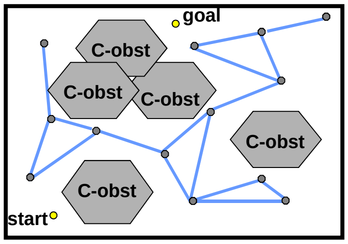
<u>Plan Generation (Query processing)</u>

1. Add *start* and *goal* poses into the roadmap
2. Find path from *start* to *goal* within roadmap
3. Generate a motion plan for each edge used

<u>Requires two primitive operations:</u>

1. Method for sampling C-Space points
2. Method for “validating” C-space points and edges

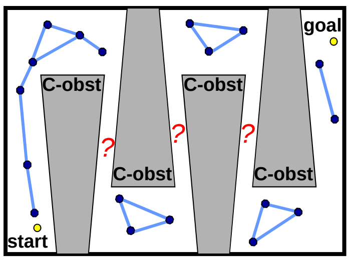
PRMs: Pros and Cons



Advantages

1. *Probabilistically complete*
2. Easily applied to high-dimensional C-spaces
3. Supports fast queries (w/ enough preprocessing)

Many success stories in which PRMs have been applied to problems previously thought intractable



Disadvantages

PRMs don't work well for some problems:

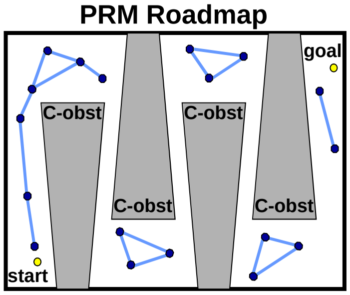
–
–

Sampling Around Obstacles: OBPRM [Amato et al. 1998]

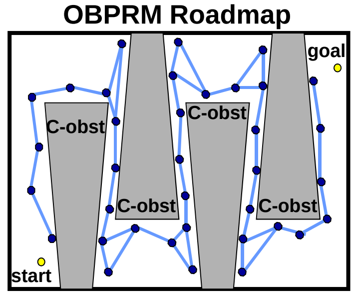
To navigate narrow passages we must *sample* inside them

Most PRM nodes placed where planning is easy, not where it's hard

PRM Roadmap



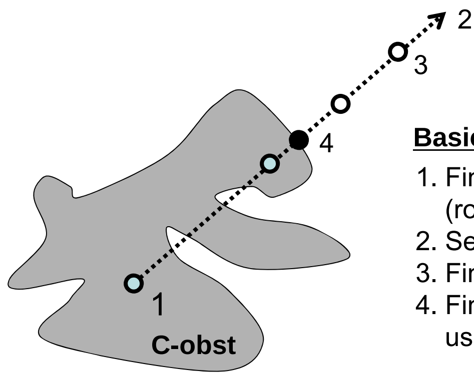
OBPRM Roadmap



Idea: Can we sample nodes near C-obstacle surfaces?

- We cannot explicitly construct the C-obstacles, but...
- We do have models of the (workspace) obstacles!

Finding Points on C-obstacles



Basic Idea (for workspace obstacle S)

1. Find a point in S's C-obstacle (robot placement colliding with S)
2. Select random direction in C-space
3. Find freespace point in that direction
4. Find boundary point between points using binary search (collision checks)

Note: we can use more sophisticated approaches to try to "cover" C-obstacle

Summary

- Introduced drastically simplifying transformation
  - Based on two useful geometric constructions
- Enables use of familiar techniques…
  - Discretization
  - Random sampling
  - Bisection
  - Graph search
- … To solve high-dimensional motion planning
- We'll use these ideas in Lab 6 (path planning)