

Potential Field Methods

Acknowledgement: Parts of these course notes are based on notes from courses given by Jean-Claude Latombe at Stanford University (and Chapter 7 in his text *Robot Motion Planning*, Kluwer, 1991), O. Burchan Bayazit at Washington University in St. Louis. Seth Hutchinson at the University of Illinois at Urbana-Champaign, and Leo Joskowicz at Hebrew University.

Potential Field Methods

Basic Idea:

- robot is represented by a point in C-space
 - treat robot like particle under the influence of an **artificial potential field U**
 - U is constructed to reflect (locally) the structure of the free C-space (hence called 'local' methods)
 - originally proposed by Khatib for on-line collision avoidance for a robot with proximity sensors
-

Motion planning is an iterative process

1. compute the artificial force $\vec{F}(\mathbf{q}) = -\nabla U(\mathbf{q})$ at current configuration
 2. take a small step in the direction indicated by this force
 3. repeat until reach goal configuration (or get stuck)
-

Note:

- major problem: local minima (most potential field methods are incomplete)
- advantages: speed
- RPP, a randomized potential field method proposed by Barraquand and Latombe for path planning, can be applied to robots with many dof

The Potential Field (translation only)

Assumption: \mathcal{A} translates freely in $\mathcal{W} = \mathbb{R}^2$ or \mathbb{R}^3 at fixed orientation (so $\mathcal{C} = \mathcal{W}$)

The Potential Function: $\mathbf{U} : \mathcal{C}_{free} \longrightarrow \mathbb{R}^1$

- want robot to be *attracted* to goal and *repelled* from obstacles
 - **attractive potential** $\mathbf{U}_{att}(\mathbf{q})$ associated with \mathbf{q}_{goal}
 - **repulsive potential** $\mathbf{U}_{rep}(\mathbf{q})$ associated with \mathcal{CB}
 - $\mathbf{U}(\mathbf{q}) = \mathbf{U}_{att}(\mathbf{q}) + \mathbf{U}_{rep}(\mathbf{q})$
- $\mathbf{U}(\mathbf{q})$ must be differentiable for every $\mathbf{q} \in \mathcal{C}_{free}$

The Field of Artificial Forces: $\vec{F}(\mathbf{q}) = -\nabla\mathbf{U}(\mathbf{q})$

- $\nabla\mathbf{U}(\mathbf{q})$ denotes gradient of \mathbf{U} at \mathbf{q} , i.e., $\nabla\mathbf{U}(\mathbf{q})$ is a vector that 'points' in the direction of 'fastest change' of \mathbf{U} at configuration \mathbf{q}
- e.g., if $\mathcal{W} = \mathbb{R}^2$, then $\mathbf{q} = (x, y)$ and

$$\nabla\mathbf{U}(\mathbf{q}) = \begin{bmatrix} \frac{\partial\mathbf{U}}{\partial x} \\ \frac{\partial\mathbf{U}}{\partial y} \end{bmatrix}$$

- $|\nabla\mathbf{U}(\mathbf{q})| = \sqrt{\left(\frac{\partial\mathbf{U}}{\partial x}\right)^2 + \left(\frac{\partial\mathbf{U}}{\partial y}\right)^2}$ is the magnitude of the rate of change
- $\vec{F}(\mathbf{q}) = -\nabla\mathbf{U}_{att}(\mathbf{q}) - \nabla\mathbf{U}_{rep}(\mathbf{q})$

The Attractive Potential

Basic Idea: $U_{att}(\mathbf{q})$ should **increase** as \mathbf{q} moves **away from** \mathbf{q}_{goal} (like potential energy increases as you move away from earth's surface)

Naive Idea: $U_{att}(\mathbf{q})$ is linear function of distance from \mathbf{q} to \mathbf{q}_{goal}

- $U_{att}(\mathbf{q})$ does increase as move away from \mathbf{q}_{goal}
- but $-\nabla U_{att}$ has constant magnitude so it doesn't help us get to the goal

Better Idea: $U_{att}(\mathbf{q})$ is a 'parabolic well'

- $U_{att}(\mathbf{q}) = \frac{1}{2}\xi\rho_{goal}^2(\mathbf{q})$, where
 - $\rho_{goal}(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_{goal}\|$, i.e., Euclidean distance
 - ξ is some positive constant scaling factor
- unique minimum at \mathbf{q}_{goal} , i.e., $U_{att}(\mathbf{q}_{goal}) = 0$
- $U_{att}(\mathbf{q})$ differentiable for all \mathbf{q}

$$\begin{aligned}\vec{F}_{att}(\mathbf{q}) = -\nabla U_{att}(\mathbf{q}) &= -\nabla \frac{1}{2}\xi\rho_{goal}^2(\mathbf{q}) \\ &= -\frac{1}{2}\xi\nabla\rho_{goal}^2(\mathbf{q}) \\ &= -\frac{1}{2}\xi(2\rho_{goal}(\mathbf{q}))\nabla\rho_{goal}(\mathbf{q})\end{aligned}$$

The Gradient $\nabla \rho_{goal}(\mathbf{q})$

Recall: $\rho_{goal}(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_{goal}\| = (\sum_i (x_i - x_{g_i})^2)^{1/2}$,
 where $\mathbf{q} = (x_1, \dots, x_n)$ and $\mathbf{q}_{goal} = (x_{g_1}, \dots, x_{g_n})$

$$\begin{aligned}
 \nabla \rho_{goal}(\mathbf{q}) &= \nabla \left(\sum_i (x_i - x_{g_i})^2 \right)^{1/2} \\
 &= \frac{1}{2} \left(\sum_i (x_i - x_{g_i})^2 \right)^{-1/2} \nabla \left(\sum_i (x_i - x_{g_i})^2 \right) \\
 &= \frac{1}{2} \left(\sum_i (x_i - x_{g_i})^2 \right)^{-1/2} (2(x_1 - x_{g_1}), \dots, 2(x_n - x_{g_n})) \\
 &= \frac{(x_1, \dots, x_n) - (x_{g_1}, \dots, x_{g_n})}{(\sum_i (x_i - x_{g_i})^2)^{1/2}} \\
 &= \frac{\mathbf{q} - \mathbf{q}_{goal}}{\|\mathbf{q} - \mathbf{q}_{goal}\|} = \frac{\mathbf{q} - \mathbf{q}_{goal}}{\rho_{goal}(\mathbf{q})}
 \end{aligned}$$

So, $-\nabla \rho_{goal}(\mathbf{q})$ is a unit vector directed toward \mathbf{q}_{goal} from \mathbf{q}

Thus, since $-\nabla \mathbf{U}_{att}(\mathbf{q}) = -\frac{1}{2}\xi(2\rho_{goal}(\mathbf{q}))\nabla \rho_{goal}(\mathbf{q})$, we get:

$$\vec{F}_{att}(\mathbf{q}) = -\nabla \mathbf{U}_{att}(\mathbf{q}) = -\xi(\mathbf{q} - \mathbf{q}_{goal})$$

- $\vec{F}_{att}(\mathbf{q})$ is a vector directed toward \mathbf{q}_{goal} with magnitude linearly related to the distance from \mathbf{q} to \mathbf{q}_{goal}
- $\vec{F}_{att}(\mathbf{q})$ converges linearly to zero as \mathbf{q} approaches \mathbf{q}_{goal} – good for stability
- $\vec{F}_{att}(\mathbf{q})$ grows without bound as \mathbf{q} moves away from \mathbf{q}_{goal} – not so good

Conic Well Attractive Potential

Idea: Use a 'conic well' to keep $\vec{F}_{att}(\mathbf{q})$ bounded

- $\mathbf{U}_{att}(\mathbf{q}) = \xi \rho_{goal}(\mathbf{q})$
- $\vec{F}_{att}(\mathbf{q}) = -\nabla \mathbf{U}_{att}(\mathbf{q}) = -\xi \frac{(\mathbf{q} - \mathbf{q}_{goal})}{\|\mathbf{q} - \mathbf{q}_{goal}\|}$
- $\vec{F}_{att}(\mathbf{q})$ is a unit vector (constant magnitude) directed towards \mathbf{q}_{goal} everywhere except $\mathbf{q} = \mathbf{q}_{goal}$
- \mathbf{U}_{att} is singular at the goal – not stable (might cause oscillations)

Better (?) Idea: A hybrid method with parabolic and conic wells

$$\mathbf{U}_{att}(\mathbf{q}) = \begin{cases} \frac{1}{2}\xi \rho_{goal}^2(\mathbf{q}) & \text{if } \rho_{goal}(\mathbf{q}) \leq d \\ d\xi \rho_{goal}(\mathbf{q}) & \text{if } \rho_{goal}(\mathbf{q}) > d \end{cases}$$

and

$$\vec{F}_{att}(\mathbf{q}) = \begin{cases} -\xi(\mathbf{q} - \mathbf{q}_{goal}) & \text{if } \|\mathbf{q} - \mathbf{q}_{goal}\| \leq d \\ -d\xi \frac{(\mathbf{q} - \mathbf{q}_{goal})}{\|\mathbf{q} - \mathbf{q}_{goal}\|} & \text{if } \|\mathbf{q} - \mathbf{q}_{goal}\| > d \end{cases}$$

The Repulsive Potential

Basic Idea: \mathcal{A} should be repelled from obstacles

- never want to let \mathcal{A} 'hit' an obstacle
- if \mathcal{A} is far from obstacle, don't want obstacle to affect \mathcal{A} 's motion

One Choice for \mathbf{U}_{rep} :

$$\mathbf{U}_{rep}(\mathbf{q}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0}\right) & \text{if } \rho(\mathbf{q}) \leq \rho_0 \\ 0 & \text{if } \rho(\mathbf{q}) > \rho_0 \end{cases}$$

where

- $\rho(\mathbf{q})$ is minimum distance from \mathcal{CB} to \mathbf{q} , i.e., $\rho(\mathbf{q}) = \min_{\mathbf{q}' \in \mathcal{CB}} \|\mathbf{q} - \mathbf{q}'\|$
- η is a positive scaling factor
- ρ_0 is a positive constant – *distance of influence*

So, as \mathbf{q} approaches \mathcal{CB} , $\mathbf{U}_{rep}(\mathbf{q})$ approaches ∞

The Repulsive Force $\vec{F}_{rep}(\mathbf{q}) = -\nabla U_{rep}(\mathbf{q})$ for convex \mathcal{CB}

(unrealistic) Assumption: \mathcal{CB} is a single convex region

$$\begin{aligned}
 \vec{F}_{rep}(\mathbf{q}) &= -\nabla U_{rep}(\mathbf{q}) \\
 &= -\nabla \left(\frac{1}{2} \eta \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right)^2 \right) \\
 &= -\frac{1}{2} \eta \nabla \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right)^2 \\
 &= -\eta \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right) \nabla \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right) \\
 &= -\eta \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right) (-1) \left(\frac{1}{\rho^2(\mathbf{q})} \right) \nabla \rho(\mathbf{q}) \\
 &= \eta \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right) \left(\frac{1}{\rho^2(\mathbf{q})} \right) \nabla \rho(\mathbf{q})
 \end{aligned}$$

Let \mathbf{q}_c be *unique* configuration in \mathcal{CB} closest to \mathbf{q} , i.e., $\rho(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_c\|$

Then, $\nabla \rho(\mathbf{q})$ is unit vector directed away from \mathcal{CB} along the line passing through \mathbf{q}_c and \mathbf{q}

$$\nabla \rho(\mathbf{q}) = \frac{\mathbf{q} - \mathbf{q}_c}{\|\mathbf{q} - \mathbf{q}_c\|}$$

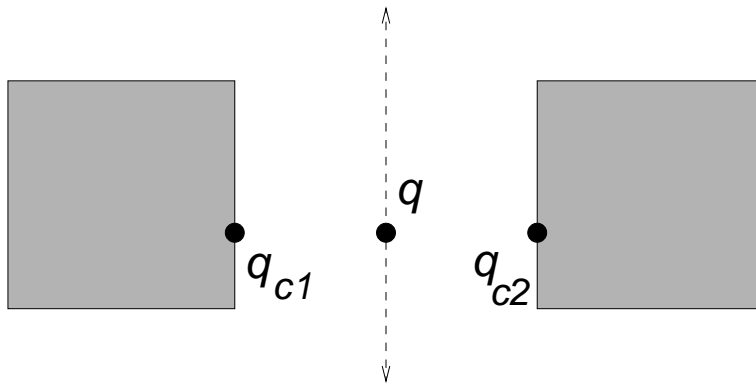
so

$$\vec{F}_{rep}(\mathbf{q}) = \eta \left(\frac{1}{\rho(\mathbf{q})} - \frac{1}{\rho_0} \right) \left(\frac{1}{\rho^2(\mathbf{q})} \right) \frac{\mathbf{q} - \mathbf{q}_c}{\|\mathbf{q} - \mathbf{q}_c\|}$$

The Repulsive Force for non-convex \mathcal{CB}

If \mathcal{CB} is not convex, $\rho(\mathbf{q})$ is differentiable everywhere except for at configurations \mathbf{q} which have more than one closest point \mathbf{q}_c in \mathcal{CB}

In general, the set of closest points \mathbf{q}_c to \mathbf{q} is $n - 1$ -dimensional (where n is the dimension of \mathcal{C})



Note: $\vec{F}_{rep}(\mathbf{q})$ exists on both sides of this line, but points in different directions (towards line) and could result in paths that oscillate

Usual Approach: Break \mathcal{CB} (or \mathcal{B}) into convex pieces

- associate repulsive field with each convex piece
- final repulsive field is the sum
- potential trouble that several small \mathcal{CB}_i may combine to generate a repulsive force greater than would be produced by a single larger obstacle
 - can weight fields according to size of \mathcal{CB}_i

Notes on Repulsive Fields

on designing \mathbf{U}_{rep}

- can select different η and ρ_0 for each obstacle region – ρ_0 small for \mathcal{CB}_i close to goal (or else repulsive force may keep us from ever reaching goal)
- if $\mathbf{U}_{rep}(\mathbf{q}_{goal}) \neq 0$, then global minimum of $\mathbf{U}(\mathbf{q})$ is generally not at \mathbf{q}_{goal}

on computing \mathbf{U}_{rep}

- pretty easy if \mathcal{CB} is polygonal or polyhedral
- really hard for arbitrary shaped \mathcal{CB}
- can try to break \mathcal{CB} into convex pieces (not necessary polyhedral) – then can use iterative, numerical methods to find closest boundary points

Gradient Descent Potential Guided Planning

Using a potential field (attractive and repulsive) for path planning...

GRADIENT DESCENT PLANNING

input: \mathbf{q}_{init} , \mathbf{q}_{goal} , $\mathbf{U}(\mathbf{q}) = \mathbf{U}_{att}(\mathbf{q}) + \mathbf{U}_{rep}(\mathbf{q})$, and $\vec{F}(\mathbf{q}) = -\nabla\mathbf{U}(\mathbf{q})$

output: a path connecting \mathbf{q}_{init} and \mathbf{q}_{goal}

1. let $\mathbf{q}_0 = \mathbf{q}_{init}$, $i = 0$
 2. if $\mathbf{q}_i \neq \mathbf{q}_{goal}$
 then $\mathbf{q}_{i+1} = \mathbf{q}_i + \delta_i \frac{\vec{F}(\mathbf{q})}{\|\vec{F}(\mathbf{q})\|}$ {take a step of size δ_i in direction $\vec{F}(\mathbf{q})$ }
 else stop
 3. set $i = i + 1$ and goto step 2
-

Notes/Difficulties/Issues:

- originally proposed and well-suited for on-line planning where obstacles are 'sensed' during motion execution [Khatib 86], [Koditschek 89]
- also called 'Steepest Descent' or 'Depth-First' Planning
- local minima are a major problem – recognizing and escaping ...
 – heuristics for escaping [Donald 84, Donald 87]
- step size δ_i
 - δ_i should be small enough so that no collision is possible when moving along straight-line segment $\mathbf{q}_i, \mathbf{q}_{i+1}$ in C-space, e.g., set δ_i smaller than minimum (current) distance to \mathcal{CB}
 - δ_i shouldn't let us overshoot goal
- how to evaluate $\rho(\mathbf{q})$ and $\nabla\rho(\mathbf{q})$ which appear in the equations for $\vec{F}(\mathbf{q})$, i.e., in finding the closest point of \mathcal{CB} to current configuration \mathbf{q}

Better Search Methods

Problem: Gradient descent methods are fast, and pretty simple . . . , but they have problems getting stuck in local minima

Other Search Methods

- **BEST-FIRST SEARCH:** deal with local minima by 'filling' them up
- **NUMERICAL NAVIGATION FUNCTIONS:** design potential functions that are 'guaranteed' to get us to the goal (in general not possible)
- **RANDOM WALK METHODS:** when get stuck, take a random walk and continue (not complete, but often work very well)

Best-First Search Planning

Idea: use an AI-type search – place a grid over \mathcal{C} and use a heuristic to search the grid

Note: we implicitly used a similar discretization of \mathcal{C} in depth-first planning via our step size δ_i

THE GRID \mathcal{GC}

Definition: Two grid points \mathbf{q} and \mathbf{q}' are **p -neighbors** if they differ by exactly one 'grid increment' in at most p coordinates

- (x, y, z) is a 1-neighbor of $(x, y, z \pm 1)$, $(x, y \pm 1, z)$ and $(x \pm 1, y, z)$
- (x, y, z) is a 2-neighbor of all its 1-neighbors and $(x, y \pm 1, z \pm 1)$, $(x \pm 1, y \pm 1, z)$, and $(x \pm 1, y, z \pm 1)$
- (x, y, z) is a 3-neighbor of all its 1 and 2-neighbors and $(x \pm 1, y \pm 1, z \pm 1)$

Of course, each component is discretized appropriately, e.g., θ

In practice, consider only 1- or 2-neighbors (simply called neighbors)

Conditions the grid should satisfy

- both \mathbf{q}_{init} and \mathbf{q}_{goal} are grid points of \mathcal{GC}
- if two grid points in \mathcal{C}_{free} are neighbors, then the straight-line connecting them also lies in \mathcal{C}_{free} (means grid must be 'fine' enough)
- \mathcal{GC} is a bounded rectangloid

Best-First Planning: Basic Idea

- iteratively construct a tree T of configurations in \mathcal{GC}
- \mathbf{q}_{init} is root of T , and add nodes to T as we 'visit' them in our search
- at each iteration, examine neighbors of 'most promising' leaf in current tree (lowest potential)
 - add 'promising' unvisited neighbors as children of the current leaf (potential less than large threshold M)
 - store leaves in priority queue ($OPEN$) with priority being potential (so can find most promising leaf fast)
- stop when reach \mathbf{q}_{goal} (a neighbor of current leaf) or there are no more unexplored nodes reachable from leaves of T
- if \mathbf{q}_{goal} is reached, the (reverse) path is identified by following leaf to root (\mathbf{q}_{goal} to \mathbf{q}_{init}) path in T

Best-First Planning Algorithm

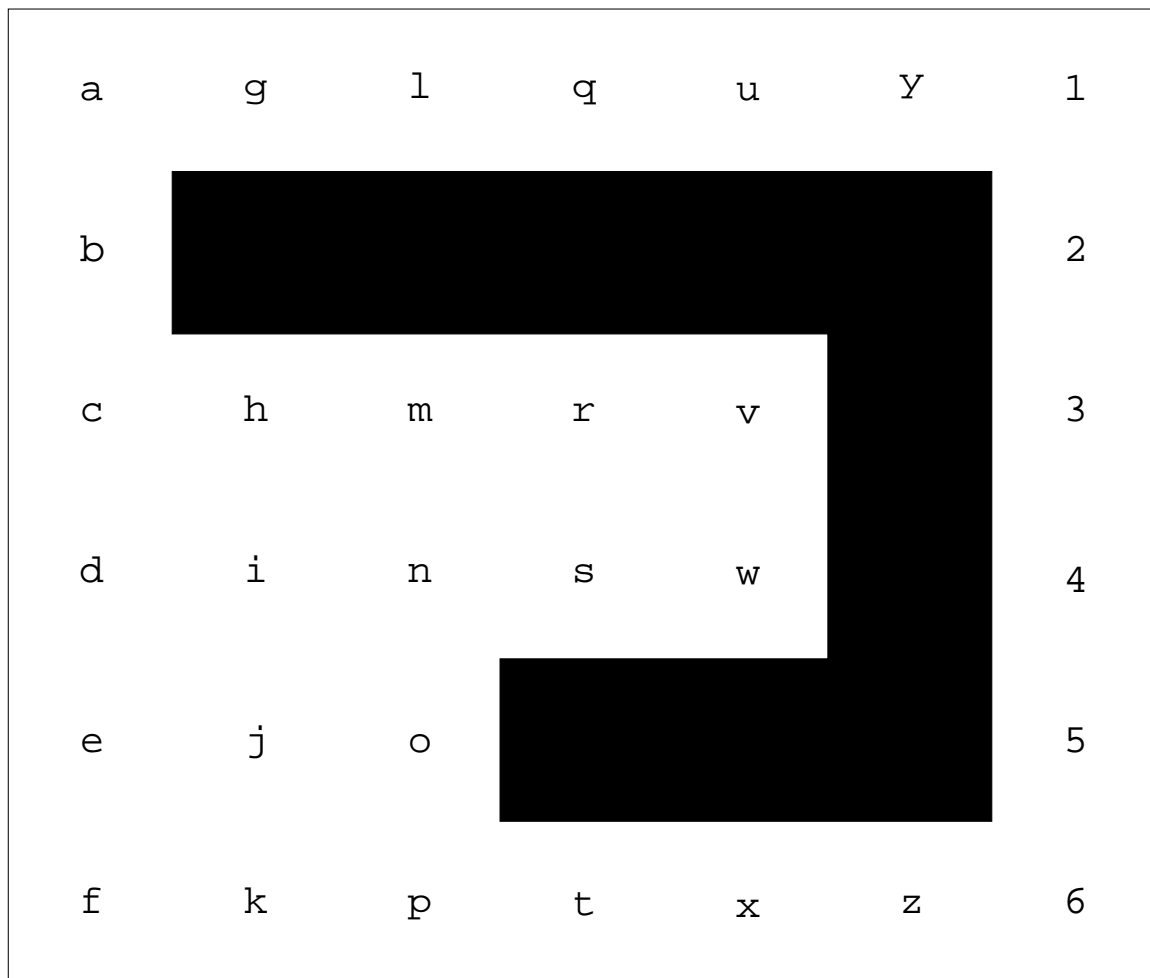
ALGORITHM: BEST-FIRST PLANNING

1. create T with root \mathbf{q}_{init}
insert \mathbf{q}_{init} in empty priority queue OPEN
 $success := False$
 2. **while** (not-empty(OPEN) and $success == False$)
 $\mathbf{q} := \text{first}(\text{OPEN})$
 for each $\mathbf{q}' \in \text{neighbor}(\mathbf{q})$
 if ($U(\mathbf{q}') < M$ and \mathbf{q}' not in T)
 make \mathbf{q}' a child of \mathbf{q} in T (pointer to \mathbf{q})
 insert(\mathbf{q}' , OPEN)
 if ($\mathbf{q}' == \mathbf{q}_{goal}$) **then** $success := True$
 3. **if** ($success == True$)
 then return path (trace pointers from \mathbf{q}_{goal} to \mathbf{q}_{init} in T)
 else return *failure*
-

intuition:

- follow steepest descent until we can go no further
- next, try another promising path (fill in/visit nodes around local minima)

Example: Best-First Planning



Remarks about Best-First Planning

Good Points

- **always** finds a solution if one exists
- if represent OPEN and T by balanced trees, then takes only logarithmic time to consider/process a new node \mathbf{q}

Practicality

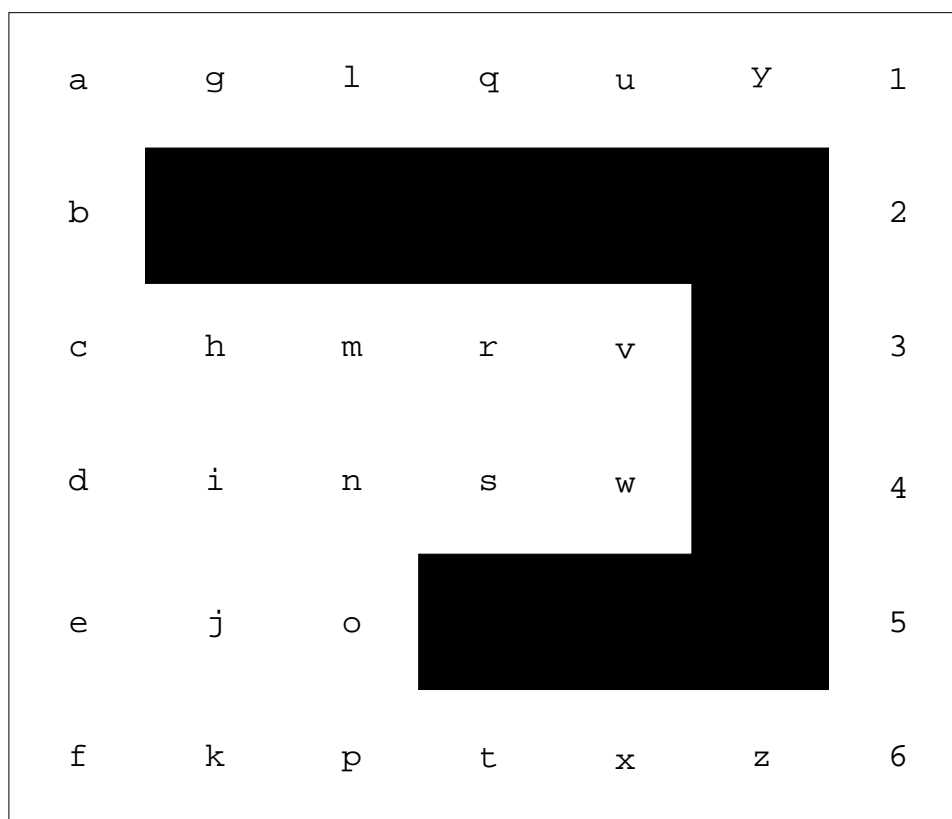
- time/space explode as number of neighbors of a node increases (when dimension m of \mathcal{C} rises and/or p increases for p -neighbors)
- works very well for $m \leq 3$...

Navigation Functions

Question: Can we design a potential function which only has one local minimum at \mathbf{q}_{goal} ? (called a **global navigation function**)

- unfortunately, not ... [Koditschek 1987]
 - if $\mathcal{C} = \mathbb{R}^2$, and if there are q disjoint C-obstacles (each homeomorphic to a closed unit disc – no 'holes'), then any potential function \mathbf{U} has at least q saddle points (unstable equilibrium)
- in principle, we could make an 'almost global navigation function' which work except for a small set of points (of measure zero) which are 'unstable' equilibrium configurations (goals) – unfortunately, this is a pain to construct
- a more practical alternative is a **numerical navigation function** which is defined at points on a grid

NF1: A Simple Numerical Navigation Function



NF1: A SIMPLE NUMERICAL NAVIGATION

1. put a grid \mathcal{GC} on C-space
2. label each grid point in \mathcal{GC} with a 0 if it is in \mathcal{C}_{free} and 1 otherwise
3. use L^1 distance (Manhattan distance) to \mathbf{q}_{goal} as navigation function \mathbf{U}
4. compute L^1 distance from \mathbf{q}_{init} to \mathbf{q}_{goal} by **wavefront propagation**
 - (a) set $\mathbf{U}(\mathbf{q}_{goal}) = 0$
 - (b) set $\mathbf{U}(\mathbf{q}') = 1$ for each 1-neighbor of the goal
 - (c) set $\mathbf{U}(\mathbf{q}') = 2$ for each 1-neighbor of nodes labelled in (b)
 - ...
 - (d) **stop** when label \mathbf{q}_{init} or no more nodes to label

NF2: Another Numerical Navigation Function

numerical navigation function NF1:

- is simple and finds paths with shortest L^1 distance
- produces paths that 'graze' C-obstacles
 - not ideal for path planning (can be improved with 'smoothing')

numerical navigation function NF2:

- finds paths that are as far as possible from C-obstacles
- still simple, but paths don't have shortest L^1 distance

The navigation function NF2 is computed in three steps:

1. compute a $(m - 1)$ -dimensional subset \mathcal{S} of \mathcal{GC}_{free} (a **skeleton**)
 - contains points with 'equal' L^1 distance from C-obstacles
 - similar to Voronoi Diagram in two-dimensions
2. compute the values of the potential function \mathbf{U} for all configurations in \mathcal{S}
3. compute \mathbf{U} for the rest of the configurations in \mathcal{GC}_{free}

The paths found will have:

- a portion connecting \mathbf{q}_{init} to the skeleton \mathcal{S}
- a portion connecting \mathbf{q}_{goal} to the skeleton \mathcal{S}
- a portion in the skeleton \mathcal{S} between these two connection points

i.e., the skeleton \mathcal{S} functions as a sort of roadmap (not 1-dimensional though except when \mathcal{C} is two-dimensional)

NF2: computing the skeleton \mathcal{S}

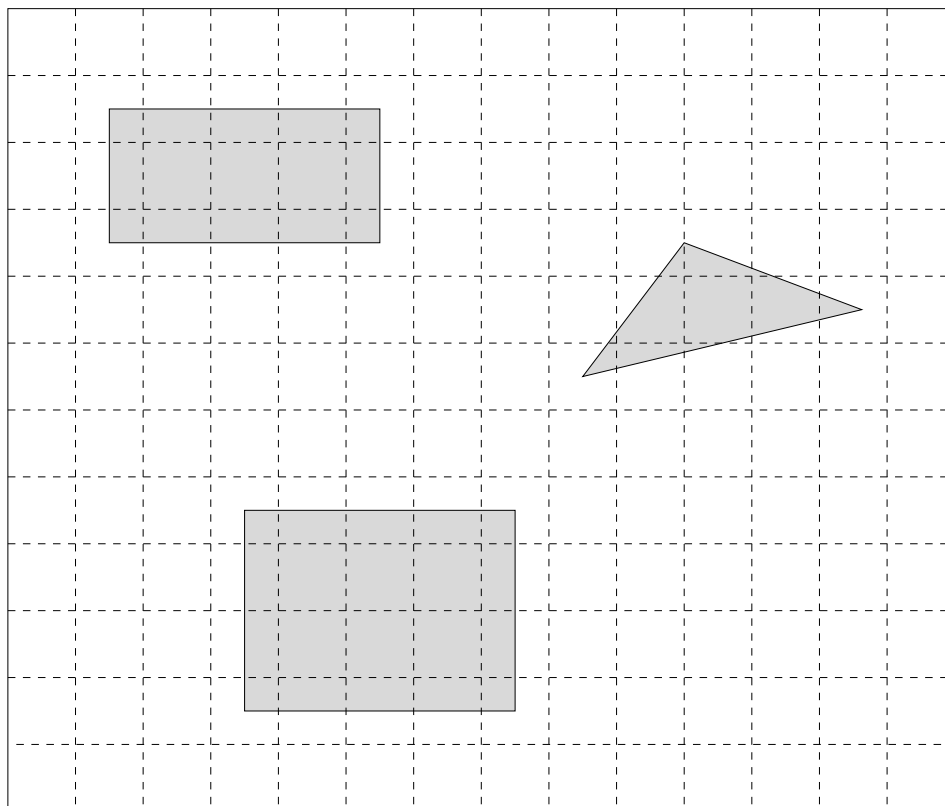
PROCEDURE: COMPUTE SKELETON

```

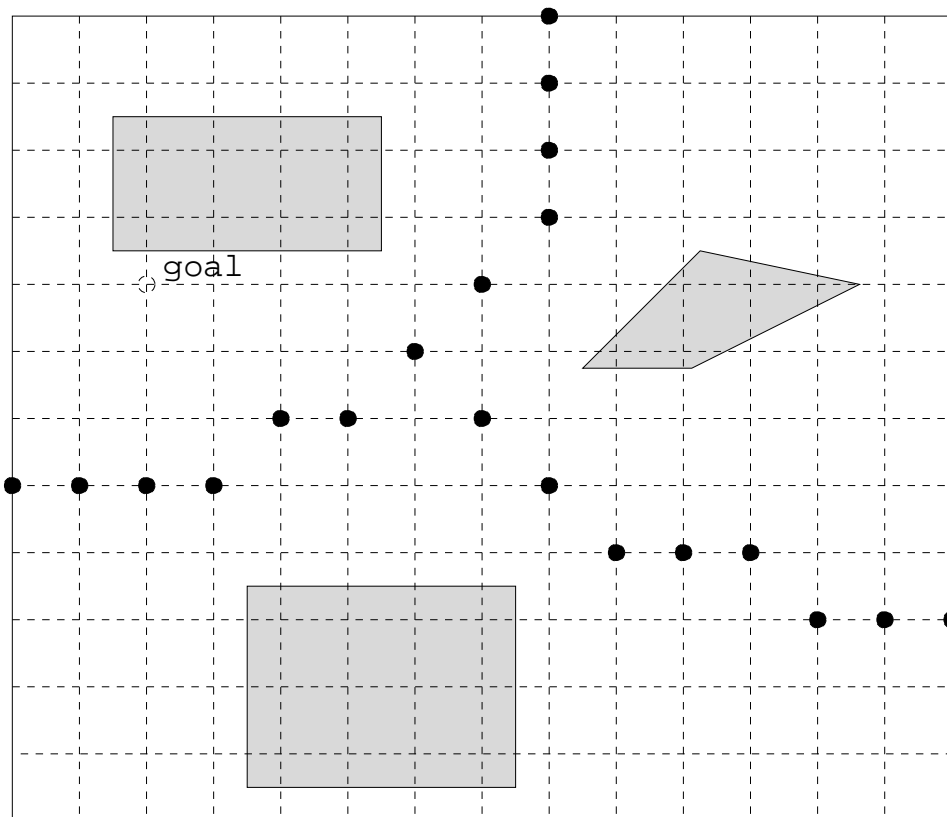
{* initialization *}
  for every  $\mathbf{q} \in \mathcal{GC}_{free}$ 
    set  $U(\mathbf{q}) := M$  { $M$  is very large number}
     $L_i := \text{empty}$ ,  $i = 0, 1, \dots$ , { $L_i$ 's are list of configs}
{* find boundary configs for C-obstacles *}
  for every  $\mathbf{q} \in \mathcal{GC} - \mathcal{GC}_{free}$ 
    if  $\mathbf{q}$  has a 1-neighbor  $\mathbf{q}'$  in  $\mathcal{GC}_{free}$  { $q$  is boundary config }
       $d_1(\mathbf{q}) := 0$  {distance from C-obstacle}
       $O(\mathbf{q}) := \mathbf{q}$  {config where wave 'started' }
      insert  $\mathbf{q}$  at end of  $L_0$  {configs dist 0 from C-obst }
    end for
  end for
{* find where waves from boundary nodes meet}
  for  $i = 0, 1, \dots$ , until  $L_i$  is empty
    for every  $\mathbf{q} \in L_i$  {configs  $q$  dist  $i$  from C-obst}
      for every 1-neighbor  $\mathbf{q}'$  of  $\mathbf{q}$  in  $\mathcal{GC}_{free}$ 
        if  $d_1(\mathbf{q}') = M$  { $q'$  not visited yet}
           $d_1(\mathbf{q}') := d_1(\mathbf{q}) + 1$  {distance from C-obstacle}
           $O(\mathbf{q}') := O(\mathbf{q})$  {config where wave 'started' }
          insert  $\mathbf{q}'$  at end of  $L_{i+1}$  {configs dist  $i+1$  from C-obst }
        else { $q'$  and  $q$  are in meeting waves }
          if  $D1(O(\mathbf{q}'), O(\mathbf{q})) > \alpha$  {waves started far enough apart}
            insert  $\mathbf{q}'$  in  $\mathcal{S}$  (if  $q$  not in  $\mathcal{S}$ )
          end for
        end for
      end for
    end for
  end for
end for

```

NF2: computing the skeleton \mathcal{S}



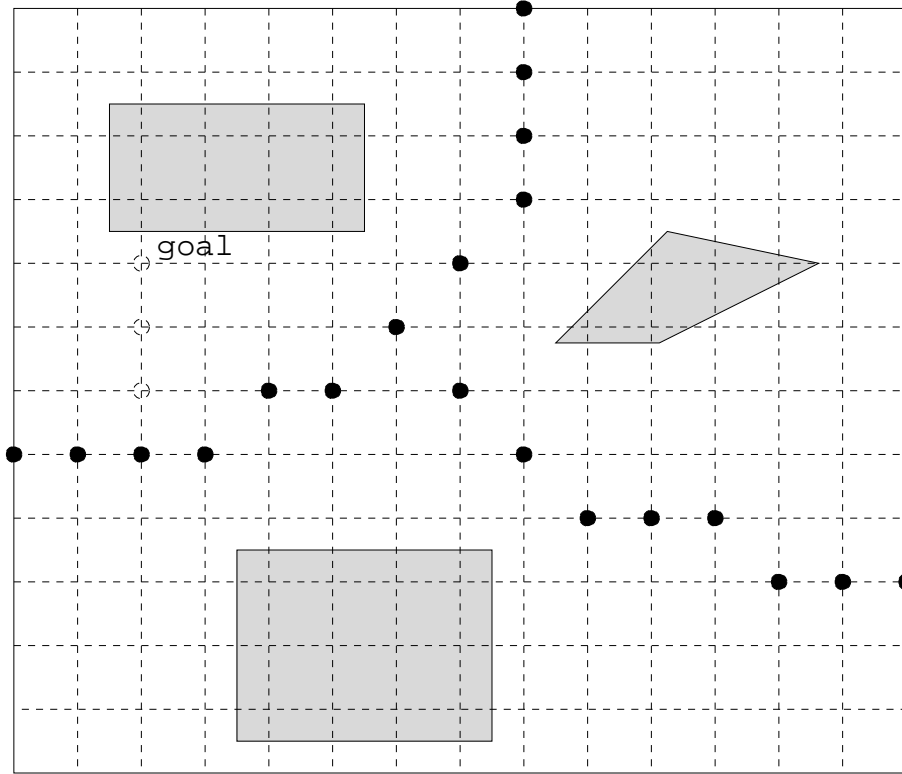
NF2: Computing U in the Skeleton



1. connect \mathbf{q}_{goal} to \mathcal{S} by path σ following steepest ascent of d_1 in \mathcal{GC}_{free}
– add σ to the skeleton \mathcal{S}
2. use **wavefront expansion** to compute \mathbf{U} **restricted** to \mathcal{S}
 - (a) set $\mathbf{U}(\mathbf{q}_{goal}) := 0$ and insert in list Q
 - (b) **while** Q is not empty, pick first $\mathbf{q} \in Q$
 - for every** unvisited m -neighbor of \mathbf{q} in \mathcal{S}
 - set $\mathbf{U}(\mathbf{q}') = \mathbf{U}(\mathbf{q}) + 1$
 - add \mathbf{q}' to end of Q
 - end for**
 - end while**

(now all configs in \mathcal{S} accessible from \mathbf{q}_{goal} have \mathbf{U} values)

NF2: Computing U in the rest of \mathcal{GC}_{free}



Let L_0 be the set of configurations in \mathcal{S}' accessible from \mathbf{q}_{goal} , i.e., all configurations whose potential U has already been computed

We now compute the potential $U(\mathbf{q}')$ for all configurations in $\mathcal{GC}_{free} - \mathcal{S}$ that are accessible from \mathbf{q}_{goal} by **wavefront expansion**:

```

for  $i = 0, 1, \dots$ , until  $L_i$  is empty
  for every  $\mathbf{q} \in L_i$ 
    for every 1-neighbor  $\mathbf{q}'$  of  $\mathbf{q}$  in  $\mathcal{GC}_{free}$ 
      if  $U(\mathbf{q}') = M$  {  $U(\mathbf{q}')$  not computed yet }
         $U(\mathbf{q}') := U(\mathbf{q}) + 1$ 
        insert  $\mathbf{q}'$  at end of  $L_{i+1}$ 
      end for
    end for
  end for
end for

```

NF2: Summary

The potential function \mathbf{U} computed by NF2:

- has valleys at skeleton \mathcal{S}
- generates no stable equilibrium **except** \mathbf{q}_{goal} (this is good)
- however, usually not continuous and don't want to explicitly compute its gradient
- a best-first search finds paths that connect \mathbf{q}_{init} to \mathcal{S} , then follow \mathcal{S} (far from C-obstacles), and finally leave \mathcal{S} to connect to \mathbf{q}_{goal}
- (slightly) more complex to construct than NF1: $O(n + s' \log s')$ where n is number of configs in \mathcal{GC} (may be very large) and s' is the number of configs in \mathcal{S} plus path from \mathbf{q}_{goal} to \mathcal{S}

Choosing resolution of C-space grid

- any discretization gives up completeness . . .
- larger resolution means smaller search space and faster methods
- smaller resolution means closer to complete (recall notion of resolution-complete)
- hierarchical solutions may be best choice, i.e., start with a coarse grid and refine as needed

Numerical Navigation Functions for Higher-Dimensional \mathcal{C}

Problem: the size of the grid \mathcal{GC} is too big for higher-dimensional \mathcal{C} , i.e., practical only for $m \leq 3$

- size of \mathcal{GC} increases **exponentially** with the dimension m of \mathcal{C}
- both NF1 and NF2 require time and space exponential in m

But never fear, there is a (heuristic) way out ...

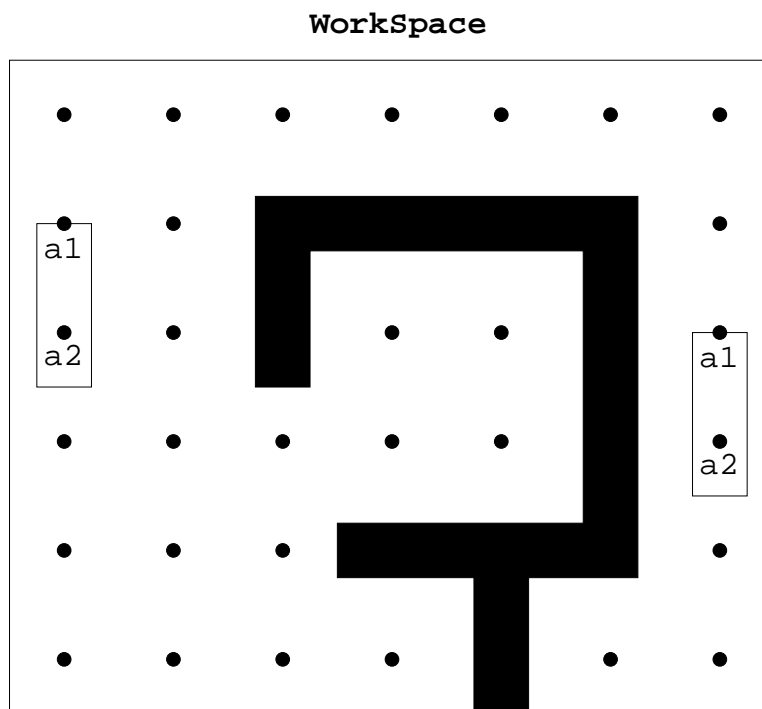
NUMERICAL NAVIGATION GENERAL APPROACH

1. put a grid on the **workspace** \mathcal{W}
2. select **control points** on the robot \mathcal{A} (multiple 'reference' points)
3. compute **workspace potentials** for each control point of the robot
 - treat each control point as a 'point robot' in a 'C-space' (the workspace)
 - use NF1 or NF2 to compute these workspace potentials
 - a C-space potential for configuration \mathbf{q} is constructed from the workspace potentials for all the control points
4. *Search:* use favorite... best-first, etc

Notes:

- Steps 1 and 2 performed off-line, before searching
- don't compute C-space potentials for all grid points, only those which search 'touches'

Computing Workspace Potentials



Let \mathcal{GW} be a regular grid of points thrown on \mathcal{A} 's **workspace**

- a point in \mathcal{GW} is labeled '1' if it lies in \mathcal{B} (or boundary), and '0' otherwise
- the set of points labeled '0' is called \mathcal{GW}_{free}

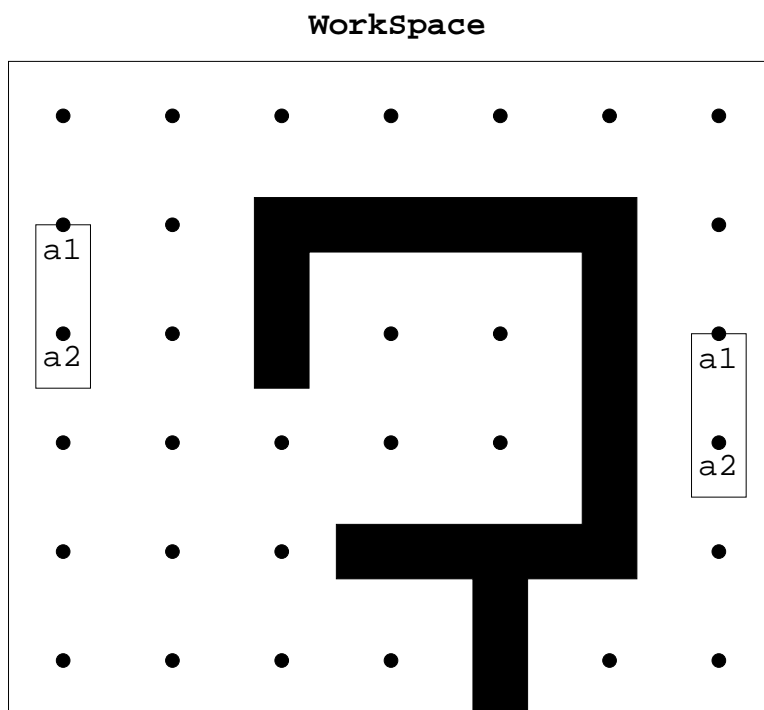
Let $a_j, j = 1, \dots, Q$ be the **control points** selected on \mathcal{A}

- a **workspace potential** function V_j is associated with each a_j
- V_j is computed by NF1 or NF2 over \mathcal{GW}_{free} , i.e., the point a_j is the 'point robot', its 'C-space' is \mathcal{W} , and its 'C-space' grid is \mathcal{GW}_{free} (analogous to \mathcal{GC}_{free})

Computing C-space Potentials from Workspace Potentials

The C-space potential $\mathbf{U}(\mathbf{q})$ is computed using an **arbitration function** G :

$$\mathbf{U}(\mathbf{q}) = G(\mathbf{V}_1(a_1(\mathbf{q})), \mathbf{V}_2(a_2(\mathbf{q})), \dots, \mathbf{V}_Q(a_Q(\mathbf{q})))$$



- only compute $\mathbf{U}(\mathbf{q})$ as needed (not for all of \mathcal{GC})
- to compute $\mathbf{V}_j(a_j(\mathbf{q}))$ we
 - compute the *position* of a_j in configuration \mathbf{q}
 - find the closest grid point x in \mathcal{GW} to this position
 - **lookup** the value of $\mathbf{V}_j(x)$ in pre-computed table
- \mathbf{U} is positive or null over \mathcal{C}_{free} and can have multiple goal configs if not enough control points
- need resolutions of \mathcal{GW} and \mathcal{GC} related appropriately

Arbitration Functions

Choice 1: $U(\mathbf{q}) = \sum_{j=1}^{j=Q} V_j(a_j(\mathbf{q}))$

- weight all control points equally
- increases 'conflicts' and creates numerous local minima

Choice 2: $U(\mathbf{q}) = \min_{j=1}^{j=Q} V_j(a_j(\mathbf{q})) + \epsilon \max_{j=1}^{j=Q} V_j(a_j(\mathbf{q}))$

- ϵ is small constant (e.g., .1)
- 1st term favors control point in \mathcal{A} that is closest to **its** goal position
- 2nd term makes \mathcal{A} rotate (a little) towards goal
- usually generates fewer local minima than choice 1
- however, may create 'deep' local minima if not enough space to rotate when get close to goal position

Choice 3: $U(\mathbf{q}) = \max_{j=1}^{j=Q} V_j(a_j(\mathbf{q}))$

- favors control point that is farthest from **its** goal position
- increases 'competition' among control points and therefore increases the number of local minima
- experimentally, these local minima will have smaller attraction domains

NF1 vs NF2 for Workspace Potentials

Generally, NF2 creates better workspace potentials than NF1

- keeps control points (and thus robot) away from obstacles
- leaves more space to maneuver (and rotate)

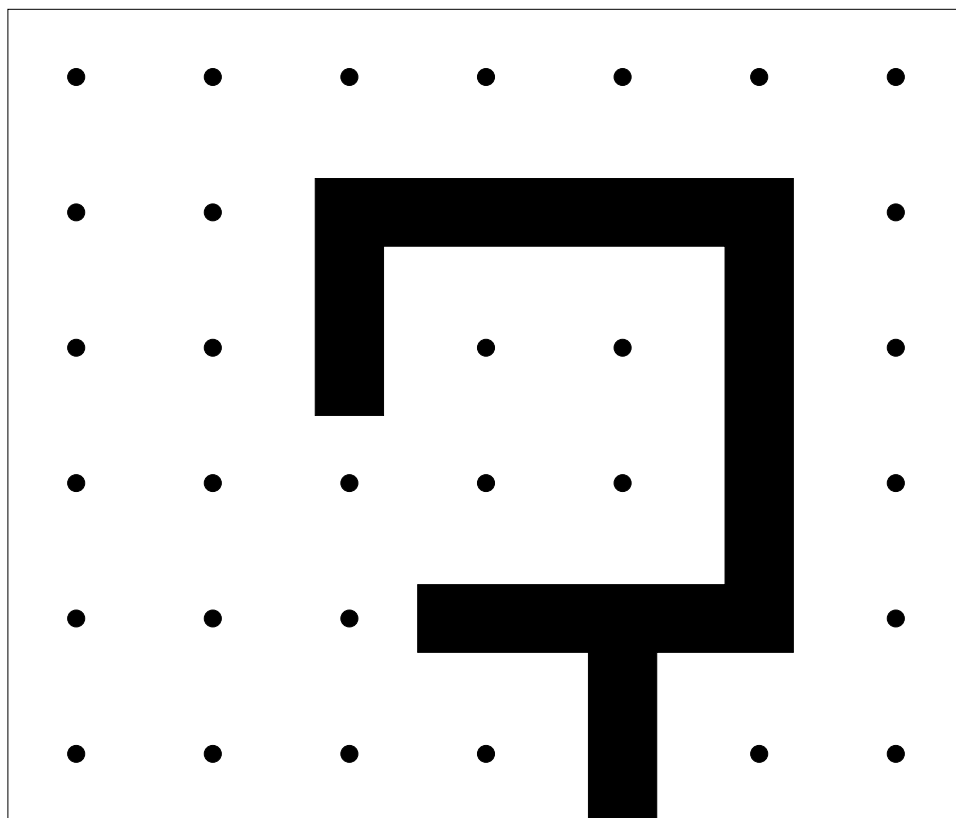
Note when using NF2:

- the skeleton \mathcal{S} is the same for all control points (depends only on obstacles)
- the skeleton \mathcal{S}' augmented with goal config is different for each control point
- so the final potentials in \mathcal{GW} computed for each control point will be different

Escaping Local Minima – Random Motions

Overview [Barraquand and Latombe, 1989]

1. start at \mathbf{q}_{init} and use best-first search to follow the gradient of $\mathbf{U}(\mathbf{q})$ until reach a local minimum \mathbf{q}_{loc} (called **gradient motion**)
2. execute a **random walk** starting at \mathbf{q}_{loc} and ending at some new configuration \mathbf{q}' (random walk approximates Brownian motion)
3. execute another gradient motion starting \mathbf{q}' (end of random walk)
4. repeat until reach \mathbf{q}_{goal}



Note: this strategy incrementally builds a connectivity graph of the local minimum (each gradient motion terminates at a local minimum)

Gradient Motion

Ideally, at each step in our best-first search we would like to check all m -neighbors of current configuration \mathbf{q} , but ...

- $2m = 2\binom{m}{1}$ 1-neighbors
- $2m^2 = 2m + 2^2\binom{m}{2}$ 2-neighbors
- $2m^2 + 2^3\binom{m}{3}$ 3-neighbors
- ...
- $3^m - 1 = \sum_{i=1}^m 2^i \binom{m}{i}$ m -neighbors

Thus, for high-dimensional \mathcal{C} , there are way too many m -neighbors

- e.g., $m = 10$ gives $3^{10} - 1 = 59048$ m -neighbors!

One option would be to restrict the neighbors to say k -neighbors, where k is small, eg. $k = 3$ – however, this does not work well in practice

Randomly Selecting Neighbors to Process:

1. randomly select a neighbor $\mathbf{q}' \in \text{Neighbors}(\mathbf{q})$
2. if $\mathbf{U}(\mathbf{q}') < \mathbf{U}(\mathbf{q})$, then use \mathbf{q}' as next step
3. if $\mathbf{U}(\mathbf{q}') \geq \mathbf{U}(\mathbf{q})$ or $\mathbf{q}' \in \mathcal{CB}$, then randomly select another $\mathbf{q}' \in \text{Neighbors}(\mathbf{q})$
4. if don't find a suitable \mathbf{q}' after some fixed number K iterations, then consider \mathbf{q} to be a local minima (q_{loc}) and execute a random motion

Random Motions

So we are now sitting at a local minimum $\mathbf{q} = (x_1, x_2, \dots, x_m)$

We take a **random step** to a new configuration $\mathbf{q}' = (x'_1, x'_2, \dots, x'_m)$

- let Δ_i denote the increment on each axis (them are m)
- randomly set each x'_i to be (i) x_i , (ii) $x_i + \Delta_i$, or (iii) $x_i - \Delta_i$

Note: we will need to check \mathbf{q}' for collision

A **random walk** is a sequence of t random steps

- if t is too small we may not escape from the local minimum
- if t is too big we may loose information that we've discovered about \mathcal{C} (since we may walk out of it...)
- Barraquand and Latombe suggest letting t be a random variable with expected value L/δ^2 , where
 - L is length of 'bounding' box of \mathcal{GW}
 - δ is length of grid increment in \mathcal{GW}

Searching using Random Motions

Basic Best-First Search Strategy

- use best-first search from \mathbf{q}_{init} until reach local minimum \mathbf{q}_{loc}
- generate the successors (using random walk) of the pending local minimum with smallest potential value
 - limit the number of random motions started from any local minimum to some fixed value K
- repeat the process with the 'best' terminus of a random motion

Remarks:

- may reach same local minimum multiple times (hard to detect)
- may waste time exploring 'nested' local minimum wells (comes from best-first nature of search)

Searching Alternative 1

Alternative 1: add a 'depth-first' component to the best-first strategy

- when reach a local minimum \mathbf{q}_{loc} , generate a successor \mathbf{q}' using a random walk
- immediately perform a gradient motion from this successor \mathbf{q}' to new local minimum \mathbf{q}'_{loc}
- if $U(\mathbf{q}'_{loc}) < U(\mathbf{q}_{loc})$, then generate the successors of \mathbf{q}'_{loc} and continue in same way
- if $U(\mathbf{q}'_{loc}) \geq U(\mathbf{q}_{loc})$, then 'forget' \mathbf{q}'_{loc} and generate another successor \mathbf{q}'' of \mathbf{q}_{loc} and execute a gradient motion from \mathbf{q}''
 - generate at most K successor for any \mathbf{q}_{loc}
 - if none of these K successors leads to a 'better' local minima, then we're at a dead end and search continues from most recent local minimum whose K successors have not yet been searched

Remarks:

- generally performs better than simple best-first strategy
- may still waste time exploring 'nested' local minimum wells
- may have trouble escaping from a 'low' local minimum

RPP: Barraquand and Latombe

Idea: Always keep track of path from \mathbf{q}_{init} to current configuration, and when get 'stuck' randomly pick some configuration on this path to continue with

PROCEDURE RPP

```

 $\tau \leftarrow$  BEST-FIRST-PATH ( $\mathbf{q}_{init}$ );  $\mathbf{q}_{loc} \leftarrow$  LAST( $\tau$ );
while  $\mathbf{q}_{loc} \neq \mathbf{q}_{goal}$  do
  ESCAPE  $\leftarrow$  false;
  for  $i = 1$  to  $K$  until ESCAPE do
     $t \leftarrow$  RANDOM-TIME;
     $\tau_i \leftarrow$  RANDOM-PATH( $\mathbf{q}_{loc}, t$ );  $\mathbf{q}_{rand} \leftarrow$  LAST( $\tau_i$ );
     $\tau_i \leftarrow$  CONCAT( $\tau_i$ , BEST-FIRST-PATH( $\mathbf{q}_{rand}$ ));
     $\mathbf{q}'_{loc} \leftarrow$  LAST( $\tau_i$ );
    if  $\mathbf{U}(\mathbf{q}'_{loc}) < \mathbf{U}(\mathbf{q}_{loc})$  then
      ESCAPE  $\leftarrow$  true;
       $\tau \leftarrow$  CONCAT( $\tau, \tau_i$ );
  if  $\neg$  ESCAPE then
     $\tau \leftarrow$  BACKTRACK( $\tau, \tau_1, \dots, \tau_K$ );
     $\mathbf{q}_{back} \leftarrow$  LAST( $\tau$ );
     $\tau \leftarrow$  CONCAT( $\tau$ , BEST-FIRST-PATH( $\mathbf{q}_{back}$ )) ;
 $\mathbf{q}_{loc} \leftarrow$  LAST( $\tau$ );

```

Remarks:

- this search has been shown to work well for many problems
- can be used with high-dimensional \mathcal{C}
- does not work well for some types of problems though

Path Smoothing

Note: Paths obtained using random motions may be long and 'jerky'

- need to **smooth** them after we are done
- actually needed for many methods, including probabilistic roadmap methods (PRMs)
- lots of known, standard techniques – not a problem

Completeness of a Planner

Definitions:

- a **complete planner** is one that correctly returns a path when one exists and declares that no path exists otherwise
- a planner is **resolution complete** if it correctly returns a path when one exists *at the discretization level chosen* and returns failure otherwise
Sometimes, one also requires that the discretization can be set fine enough so that if a solution will be found if one exists
- a planner is **probabilistically complete** if, whenever a path exists, the probability that it does not find one converges to zero as computation time increases

and some other oft used terms...

- a planner is often said to be **heuristic** to imply that it is incomplete (but there are heuristics that preserve completeness)
- a planner is often said to be **exact** if it is complete and uses algebraic representations of objects (exact is often opposed to “heuristic”)
Note that algebraic representations are no more “exact” than, say, bitmap representations (no representation of the real world is totally accurate)
- **local** and **global planners**: “local” is often used for planners with minimal preprocessing whereas “global” planners fully precompute a representation of \mathcal{C}_{free} ’s connectivity (many planners between these extremes)

Moral

So far everything we've seen suggests:

- we should use complete planners only for special cases and for low-dimensional C-space (2 or 3)
- we should consider trading a limited amount of completeness for a major gain in computational efficiency
 - probabilistic completeness
 - resolution completeness

Moral: *What we probably want are heuristic, possibly incomplete, but practically efficient planners*