Motion Planning I

RSS Lecture 10

Monday, 11 Mar 2013

Prof. Seth Teller

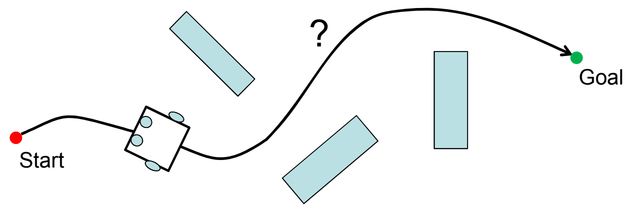
Siegwart & Nourbakhsh Ch. 6

Today

- Problem statement
- Motion planning module in context
- Bug algorithms for point robots in the plane
- Motion planning as search
  - Visibility graph algorithm
  - Discretization and A*
  - Potential Field method

Motion Planning Intuition

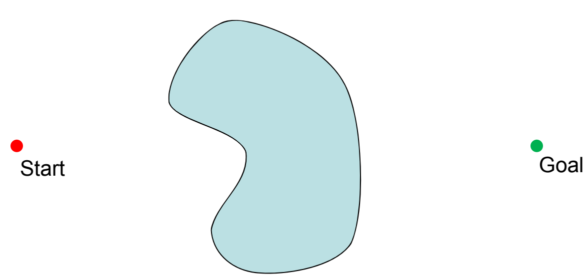
- What series of motions will get robot to goal?



- Are there cases in which no such motion exists?

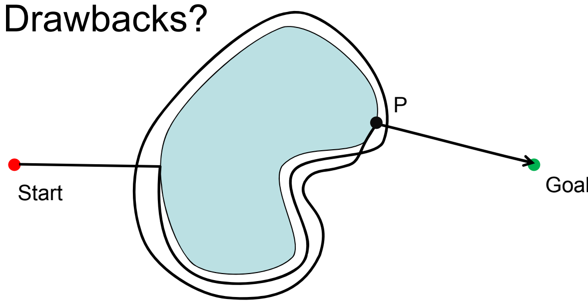
Bug Algorithm (for Point Robots)

- Simple algorithm based on four assumptions:
  - Perfect knowledge of direction and distance to goal
  - Ability to distinguish freespace from obstacle contact
  - Ability to move along an arbitrary obstacle boundary
  - Ability to detect whenever a location is revisited
- Which assumptions are strong? Weak?



Bug Motion Planning Algorithm

- Repeatedly advance toward goal
- Upon encountering an obstacle:
  - Completely circumnavigate it counter-clockwise, then depart from point P that minimizes distance to goal
- Advantages? Drawbacks?



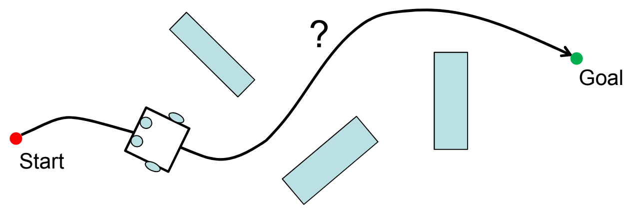
- Many variants: Bug2, BugDist, BugTangent…

Complete Motion Planning

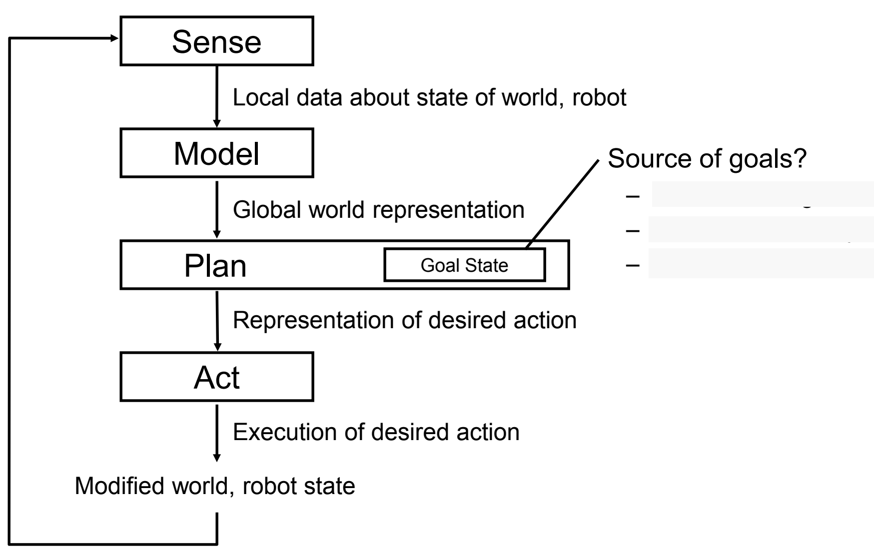
- Formal statement of motion planning problem:
  - Compute a collision-free path for a rigid or articulated moving object among static (or dynamic) obstacles
- Ideally we desire a “complete” motion planner:
  - If a solution exists, planner is guaranteed to return it
  - Otherwise, planner indicates that no solution exists
- CMP is known to be computationally intractable
  - In general it requires exponential running time in the number of DOFs (articulation, # of obstacles etc.)
  - … Even with access to perfect, global information!

Planning Under Uncertainty

- How can robot move from starting configuration to a goal configuration despite *uncertainty:*
  - Imperfect prior knowledge
  - Imperfect perception
  - Imperfect reasoning
  - Imperfect execution
  - Imperfect prediction

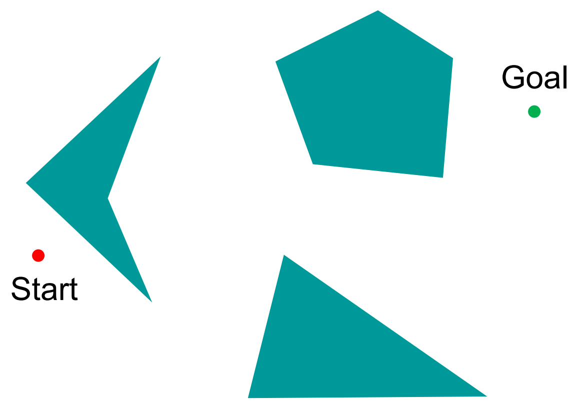


Deliberative Architecture



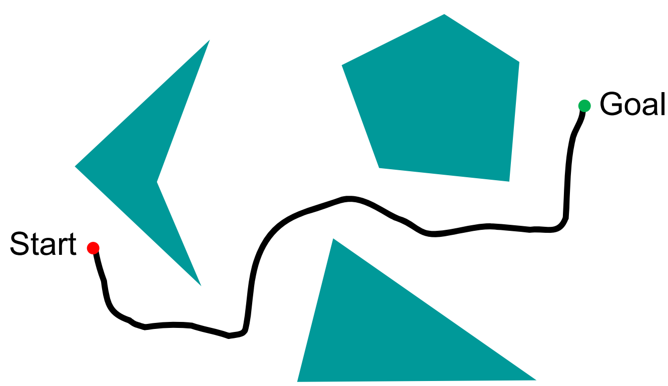
Off-Line Motion Planning

- Today, we'll make some strong assumptions:
  - Robot has perfect map of start, obstacles, goal
  - Robot can localize itself globally with no error



Motion Planning Intuition

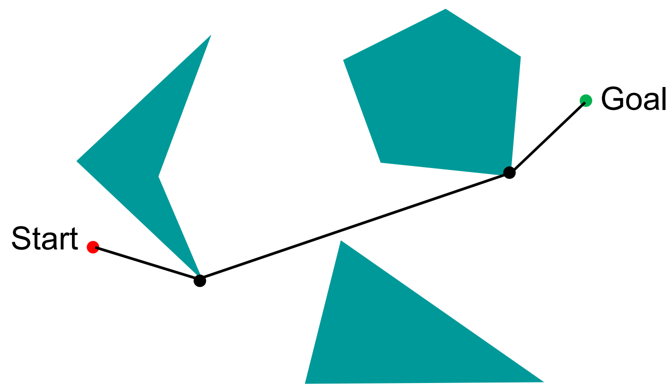
- We want robot to stay far from obstacles



… But we don't yet have a suitable representation of freespace to work with

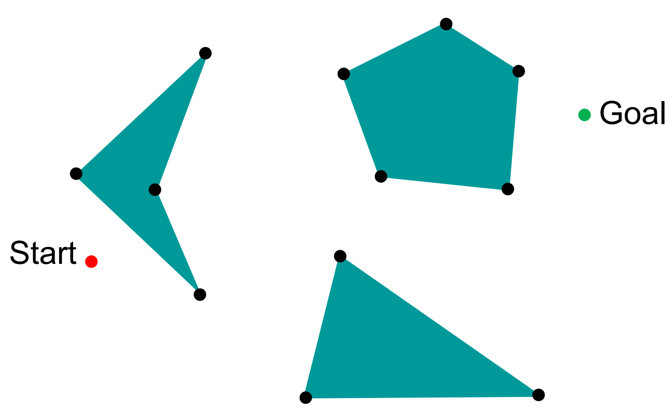
Observation

- If there exists a collision-free path from start to goal, then there exists a piecewise-linear path involving only start, goal and obstacle vertices



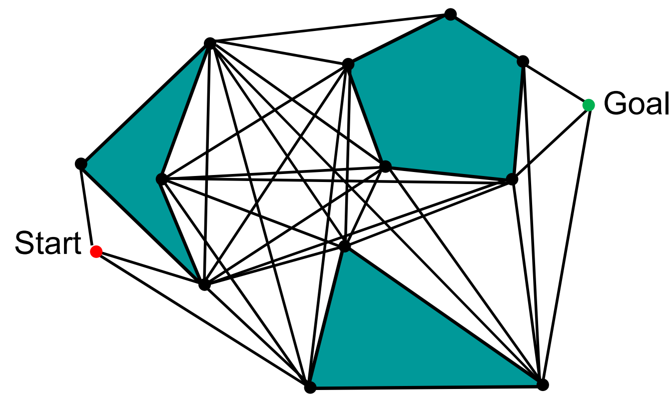
Visibility Graph Algorithm

- Construct graph G = (V, E)
  - V = {obstacle vertices} ∪ {Start, Goal}



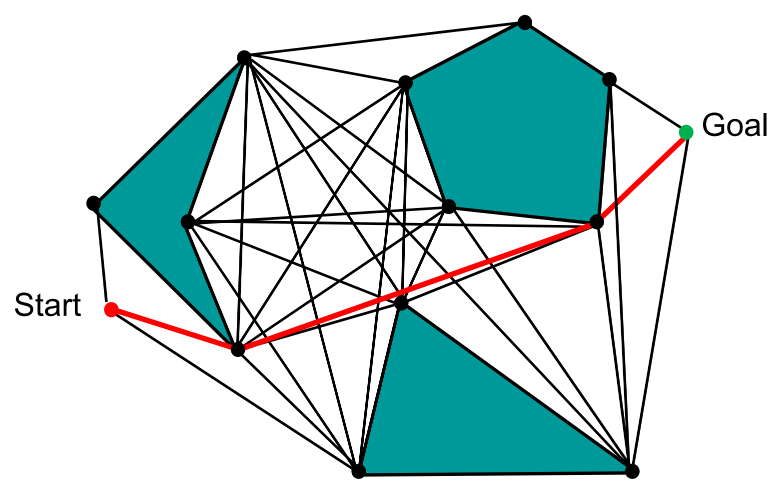
Visibility Graph Algorithm

- Construct graph G = (V, E)
  - V = {obstacle vertices} ∪ {Start, Goal}
  - E = edges (v_i, v_j) disjoint from obstacle interiors



Find Shortest Path in Graph G

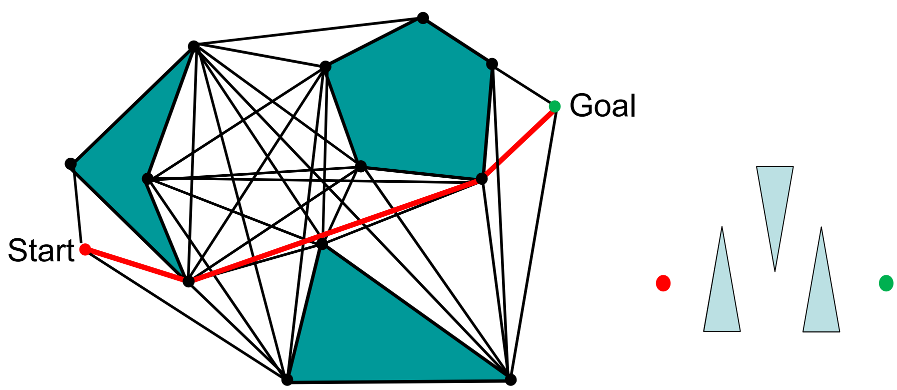
- Use rooted at start vertex



Algorithm
Single-source Shortest Path

```
function (G, w, s)                 // Graph G, weights w, source s
  for each vertex v in V[G]         // Initialize d[ ], previous, S, and Q
    d[v] := ∞                       // Vertex v is not yet reached
    previous[v] := undefined        //   … so there's no path to it yet
  d[s] := 0                         // Source reachable with zero cost
  S := empty set                    // Set of vertices reached so far
  Q := set of all vertices          // Set of candidate vertices
  while Q is not an empty set       // While unreached vertices
    u := vtx v in Q with            // O(n) search or Fibonacci heap
    S := S union {u}                // Vertex u reached
    for each edge (u, v)            // For each neighbor v of u
      if                            // If lower-cost path to v exists via u
        d[v] := d[u] + w(u,v)       //   … update cost to v
        previous[v] := u            //   … and update path record
```

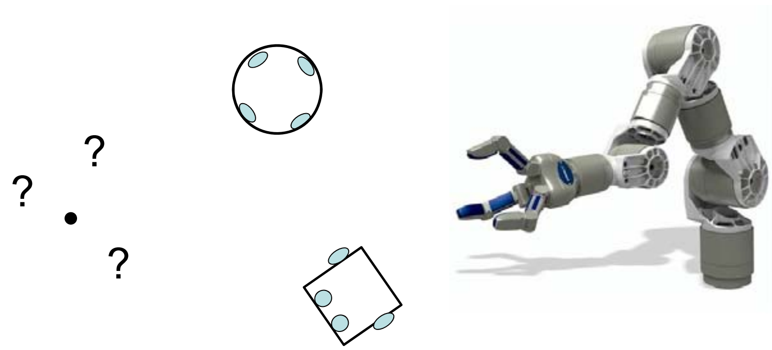
Application of Shortest-Path



- What do we use as edge weights?
- Memory usage?
- Running time?
- What major assumption have we made about the robot?
- Does this algorithm extend naturally to polyhedra in 3D?

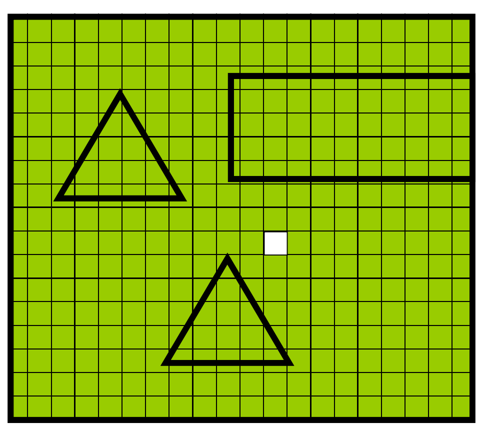
A *Point* Robot?

- Can't fit the robot into a zero-area point …
  – Today we'll address robot extent via *discretization*
  – Next time we'll see a much more elegant method



Discretizing Polygonal Obstacles

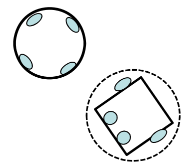
- How should we discretize freespace into a grid?
  – Is this just like rendering polygons in graphics?



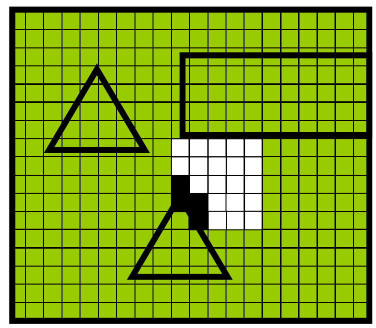
  – To avoid collisions, we must account for !

Discretizing Polygonal Obstacles

- For today, assume robot is a disk with radius R
  - Then for planning purposes, robot has only 2 DOFs (why?)

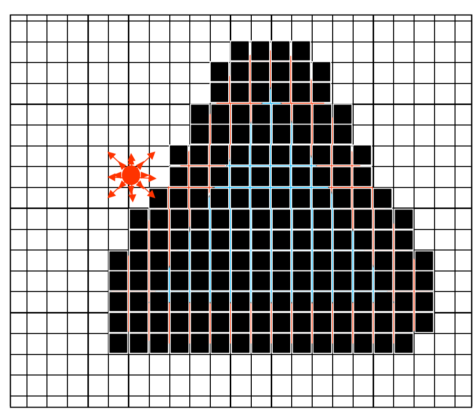


- Then a grid cell represents freespace if:
  - It does not overlap with any obstacle
  - It lies further than R from all obstacle edges
- Algorithm:
  - Pick any grid cell that is known to lie in freespace
  - Do a breadth-first search (or "flood-fill") from the start cell
  - As each cell is visited by the search, compute the minimum distance d to any obstacle edge
  - If d > R, label cell "free" and recurse; otherwise label cell "occupied"
  - Once fill is complete, label any unreached cells as "occupied"



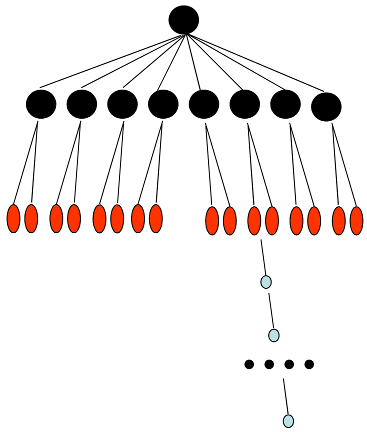
Planning in Discrete State Space

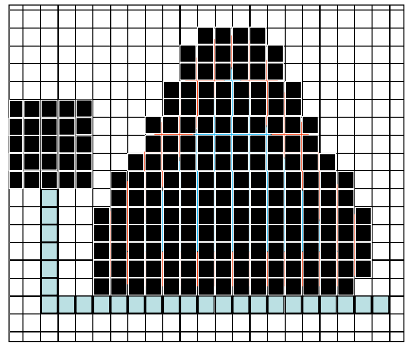
- Cartesian space



- Actions take robot from one state to another
- Objective: find a minimum-cost path from the start state to the goal state

Planning as Tree Search

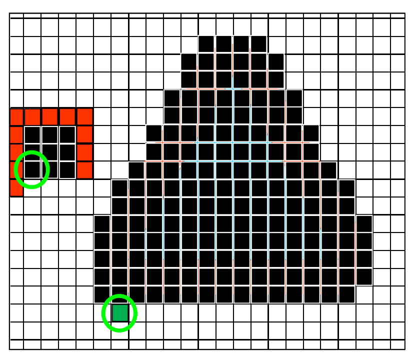




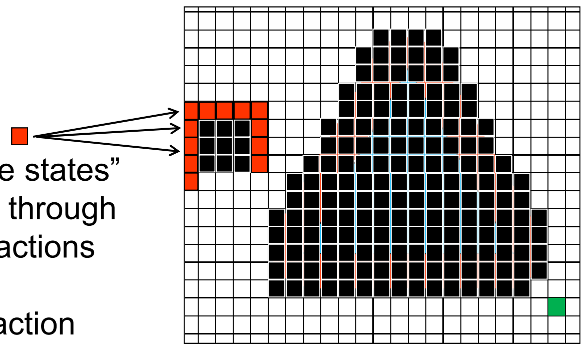
… How can such searching be made *effective* and *efficient*?

Move Generation

- Which state-action pair to consider next?
- Shallowest next
  - Aka: Breadth-first search
  - Guarantees shortest path
  - But: storage-intensive
- Deepest next
  - Aka: Depth-first search
  - Can use minimal storage
  - But: no optimality guarantee



Informed Search – A*



"Candidate states" reachable through available actions

… *which* action should robot take?

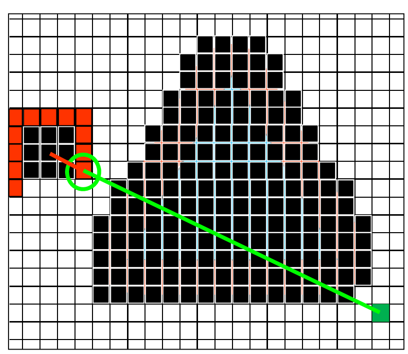
Informed Search – A*

- Use domain knowledge to bias search order
- Favor actions that might get closer to the goal
- Each state gets assigned an approximate cost

$f(x) = c(x) + h(x)$

c(x) = cost incurred from start state to graph node x

h(x) = estimated cost from node x to goal, aka "heuristic" cost

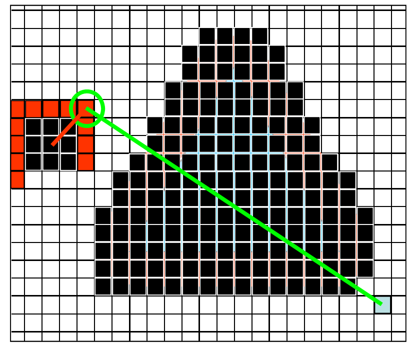


- Example:
  - **c(x) = 3**, **h(x)** = ||x - goal|| = sqrt(8^2+18^2) = **19.7**, so **f(x) = 22.7**

Informed Search – A*

- Each state x is assigned an *approximate cost* f:

$$f(x) = c(x) + h(x)$$



c(x) = cost incurred from start state to graph node x

h(x) = estimated cost from node x to goal, aka "heuristic" cost

- Choose the candidate state with the

- Cost for another example candidate action is higher:
  - c(x) = 4, h(x) = ||x-goal|| = sqrt(11^2+18^2) = 21.1, so **f(x)=25.1**

How to Construct Heuristics

- The more closely h(x) approximates the true cost to the goal, h*(x), the more efficient the search will be* …

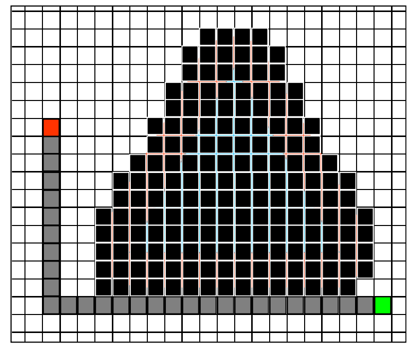
BUT:

- In order for A* to find the *optimal* path, it must be the case that
- Why? Suppose this was not the case. Then the search would
- Such an h is called an "admissible" heuristic

*There is a subtle design tradeoff involved here – what is it?

A Problem with Plans

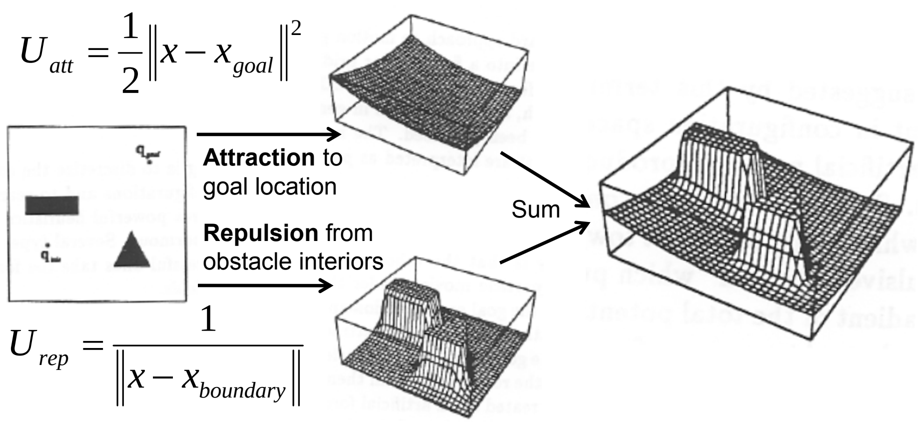
- We have a plan that gets us from the start ■ to the goal ■
- But… what happens if we *depart* from the plan?
  - We can *replan*, or:
  - We can maintain a *policy,* i.e. a data structure that can produce a plan given *any* start location



Potential Field Method

- Real-time collision avoidance method [Khatib 1986]
- Construct scalar potential field throughout freespace

$$U_{att} = \frac{1}{2}\left\| x - x_{goal} \right\|^2$$

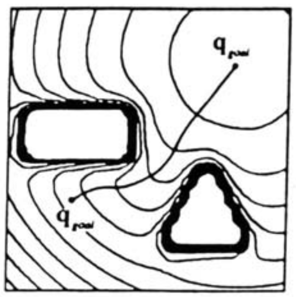


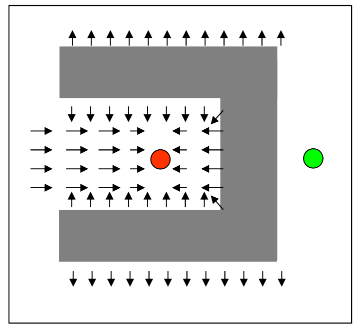
$$U_{rep} = \frac{1}{\left\| x - x_{boundary} \right\|}$$

- Robot moves along ______ of potential field

Ideal Potential Field

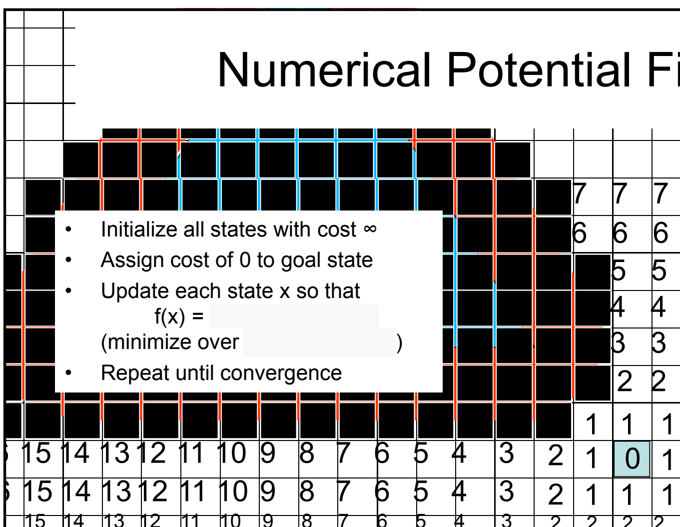
- We want to construct the potential field so that it:
  - Is nearly infinite close to obstacles
  - Has a global minimum at the goal (so no local minima)
  - Is smooth everywhere
  - Does scalar method achieve this?





If only life were so easy…

Numerical Potential Field



- Initialize all states with cost ∞
- Assign cost of 0 to goal state
- Update each state x so that
  f(x) =
  (minimize over)
- Repeat until convergence

Numbers shown are for an obstacle-induced cost of ∞, and a goal-induced cost of 1 unit per grid cell (can also make it costly to approach obstacles)

To plan from any node, simply

Completeness

- Recall our definition of complete MP
  - Is the visibility graph algorithm complete?
  - Are the potential field algorithms complete?

Recap: Design Decisions

- How is your map described? This will have an impact on the state space for your planner
  - Is it a list of polygons?
  - Is it a grid map?
- What are you trying to optimize?
  - Minimum ? How?
  - Minimum How?
  - Minimum ? How?
- What kind of search should you use?
  - Can you formulate a reasonably good heuristic?
  - If so, then A* can be a good choice
- Physical intuition can yield useful algorithms
  - Potential field methods