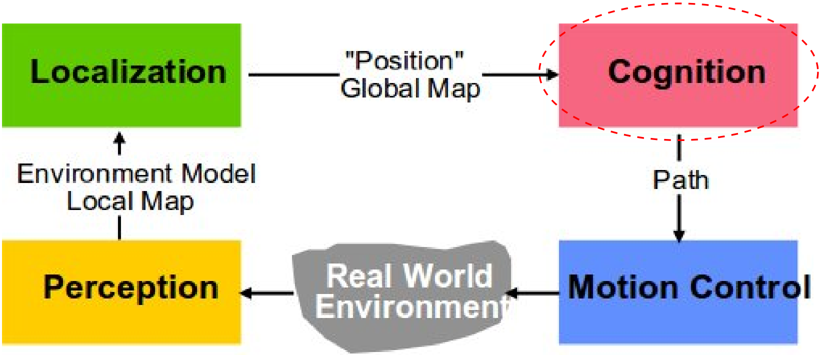
Robotics and Autonomous Systems

Lecture 12: Navigation

Terry Payne

Department of Computer Science
University of Liverpool

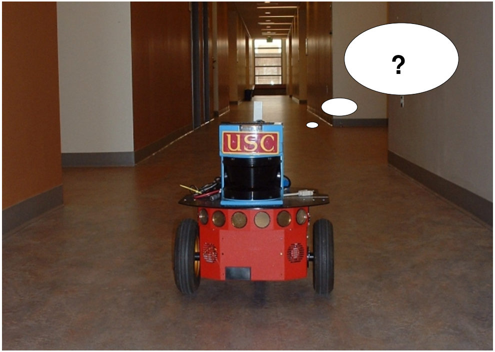
Today



- Navigation — how the robots gets around the world.

Navigation

- We started this course with three questions:



  - Where am I ?
  - Where am I going ?
  - How do I get there ?
- We are now at a point where we can answer the last two.

What counts as navigation

- Navigation is concerned with how a robot gets around the world.
    - So what is new?
- Assume that the robot:
    - Knows where it is.
    - Knows where it wants to go.
- Concerned with getting from one place to another.

What counts as navigation

- Distinguish two kinds of navigation
  - Global navigation
  - Local navigation

What counts as navigation

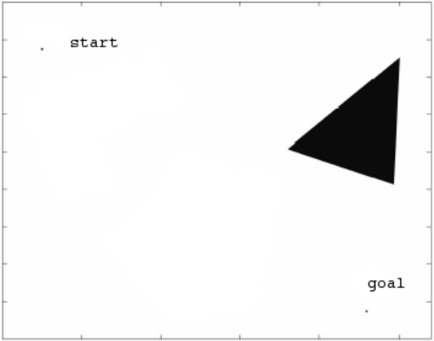
- Global navigation is about deciding how to get from some start point to a goal point.
- The robot plans in some sense.
- We will look at methods for path planning.
- In short, the robot comes up with a “plan”.
  - A sequence of way points
- We’ll look at a couple of different methods that are appropriate for different map representations.
  - Remember them?

What counts as navigation

- Local navigation is about obstacle avoidance.
  - If there are objects in the way, make sure you don't hit them.
- Range of different approaches depending on what kind of information we have about the world.
  - Depends on sensors

What counts as navigation

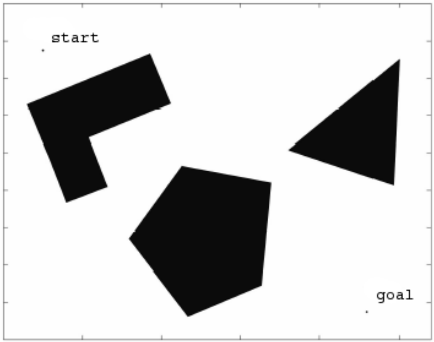
- One way to think about the difference between the two is in terms of the relationship between the robot's `start` point and the `goal` point.



- If there is a clear line of sight between the start point and the goal then we are into obstacle avoidance.
  - Just avoiding some debris that isn't on the map

What counts as navigation

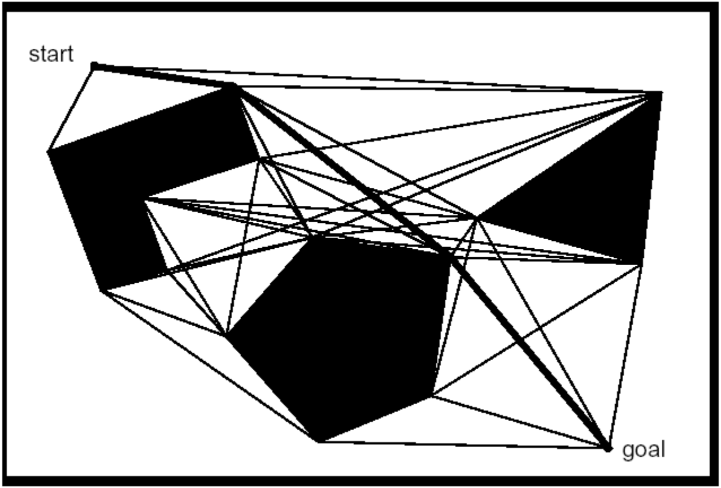
- However, if there is no line of sight from `start` to `goal`:

  then we have to find a path.
- Typically path segments will be between two points between which there is a line of sight.
  - We call these waypoints

Visibility graph

- Direct implementation of line-of-sight.



- Connect up all the vertices in the map.

Visibility graph

- Given the line segments, we can find the shortest path from `start` to `goal`.
  - We'll talk about this later.
- Can then translate the path into a series of waypoints.
  - Waypoints are the end points of the line segments.
- Given the visibility graph above, there is an obvious problem with using the lines as a guide for where the robot should go.
- Problem.

Visibility graph

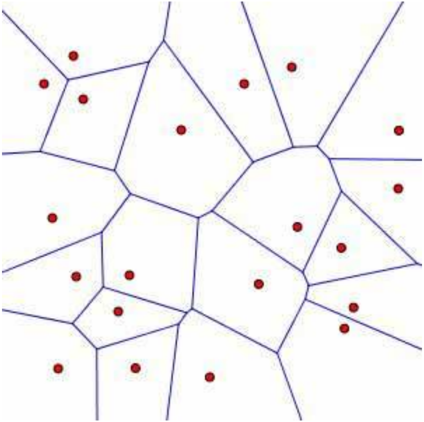
- Given the line segments, we can find the shortest path from `start` to `goal`.
  - We'll talk about this later.
- Can then translate the path into a series of waypoints.
  - Waypoints are the end points of the line segments.
- Given the visibility graph above, there is an obvious problem with using the lines as a guide for where the robot should go.
- No room for the robot.

Visibility graph

- Routes at the moment run arbitrarily close to the vertices of objects.
  - Problems with collisions
- Fix this by expanding objects by enough that the robot will still clear them.
  - More than half the diameter of the robot.
- Still not a good solution.

Voronoi diagram

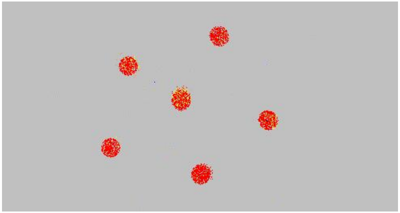
- A Voronoi diagram is a way to divide up a plane (a map).



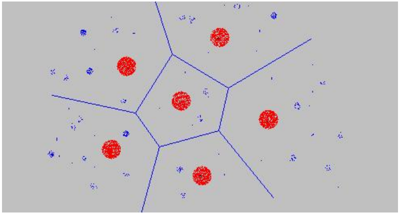
- Given a set of points P, a Voronoi diagram is a set of polygons such that the points inside each polygon are closer to one member of P than any other.

Voronoi diagram

- Here the points in P are big red dots.

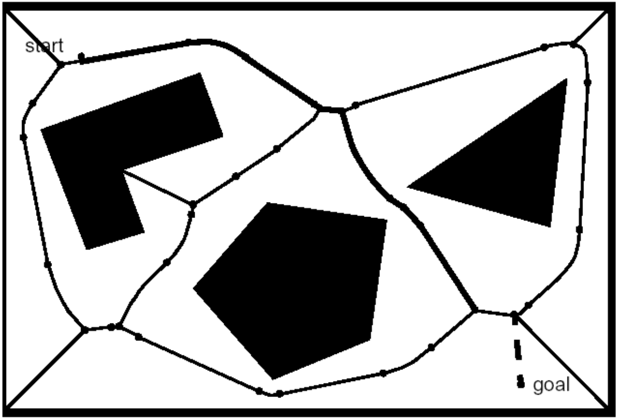


- The polygons then contain all points closer to one red dot than another.



Voronoi diagram

- Can extend this to cases where *P* is a set of objects.



- Treat the line segments exactly like the edges in the visibility graph.

Voronoi diagram

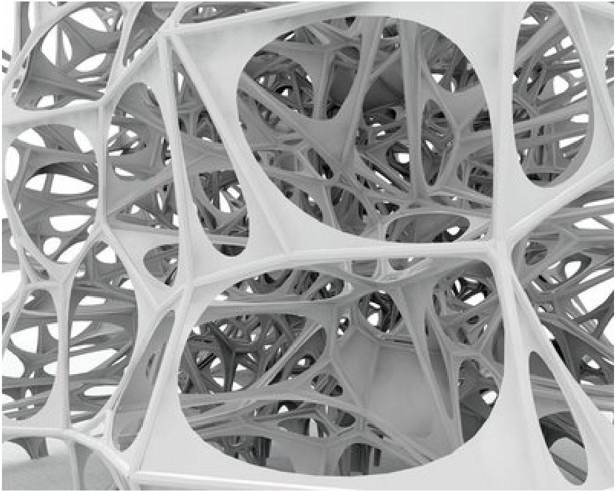
- The lines are not necessarily lines of sight
  - As above they may bend.
- However, they are object free, and so can be followed just like lines of sight can.

Voronoi diagram

- Voronoi diagrams also have a nice property in terms of path-following
  - That is when you get the robot to follow the "plan".
- A robot that is maximising its distance from objects will follow the lines in the Voronoi diagram.
  The standard kind of thing to do to follow corridors etc.
- Means that we can again reduce the path to a set of waypoints.
  - Head to the next waypoint while maximising distance from objects.

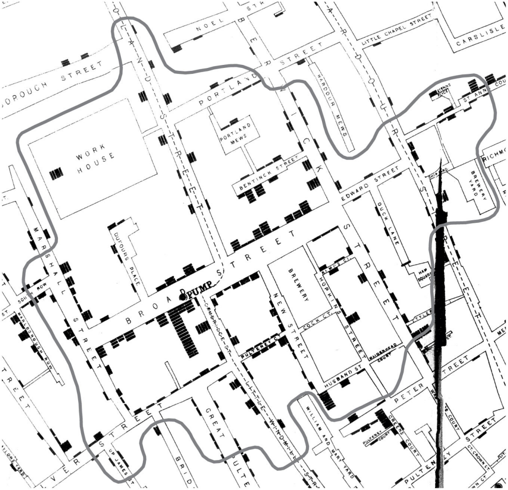
Asides

- Voronoi diagrams work in 3D also:



Asides

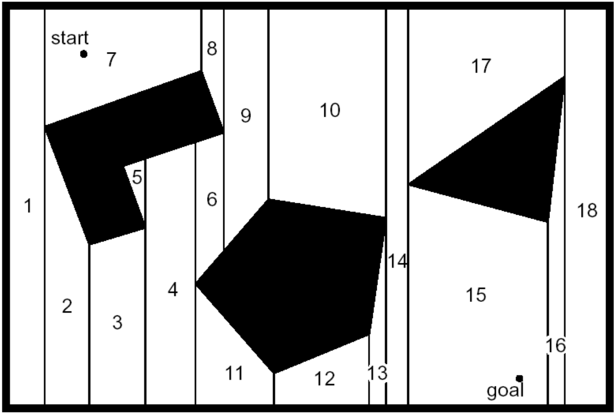
- They were also famously used by John Snow to identify the source of the 1854 cholera epidemic in London



- They were also famously used by John Snow to identify the source of the 1854 cholera epidemic in London

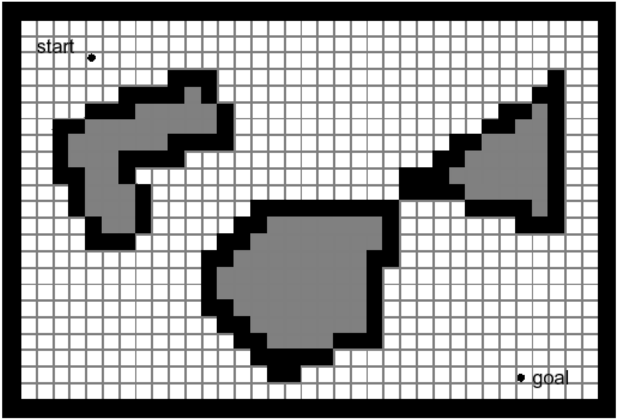
Cell-based maps

- Last time we saw a variety of different cell-based maps.



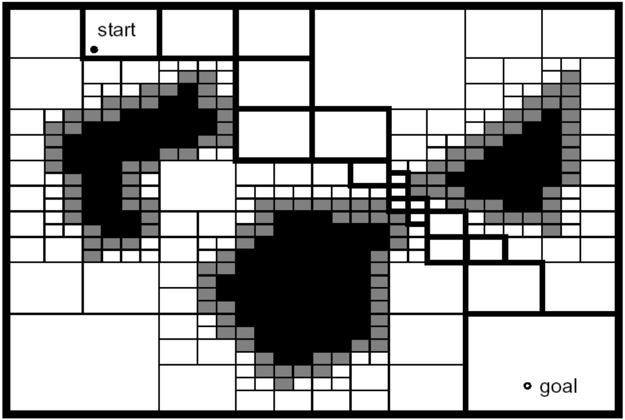
- Exact cell decomposition

Cell-based maps



- Fixed cell decomposition

Cell-based maps



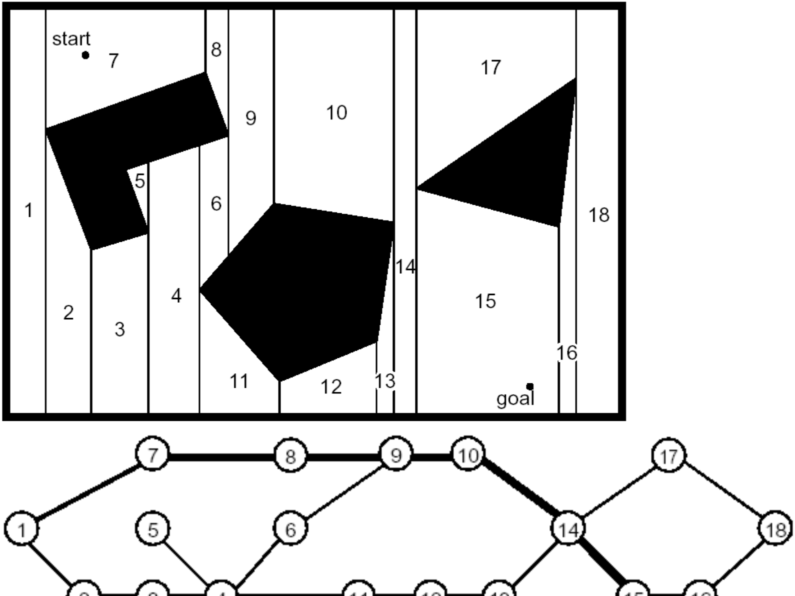
- Adaptive cell decomposition.

Cell-based maps

- Given the maps, we still want to figure out a sequence of line segments.
- Not quite so straightforward for cell-based maps.
- We will look at two general approaches to do path-finding:
  - Explicit search of a connectivity graph.
  - Wavefront planning
- These are really the same thing in different guises.

Connectivity graph

- Identify which cells are next to which other cells.



Connectivity graph

- The question is how to figure out a path from the graph.
- When the graph is complex, we need to use search techniques.
- This is also the case for the connectivity graphs we get automatically from the visibility graph or Voronoi diagram approaches.
- Standard approaches to search:
  - Depth first
  - Breadth first
  - A*
- Plus there are robotics-specific approaches like D*.

Search

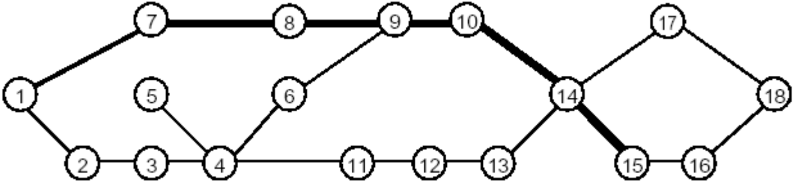
- A general algorithm for search is:

```
agenda = initial node;
while agenda not empty do{
   state <- node from agenda;
   new nodes = nodes connected to state;
   if goal in new nodes
   then {
          return solution;
      }
   add new nodes to agenda;
}
```

- Note that this doesn't generate a set of waypoints, it just looks for the goal state.

Search

- Let's think about how this would work on the connectivity graph:



Search

- To use the algorithm we need to decide how to do the selection in

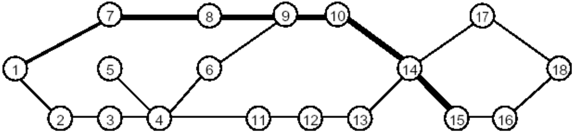
```
state <- node from agenda;
```

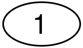
  and how to do the addition in:

```
add new nodes to agenda;
```

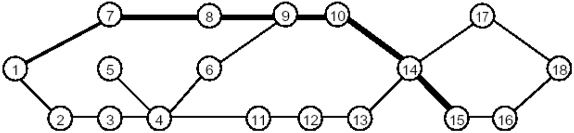
- Depth-first search:
  - Takes the first node on the agenda;
  - Adds new nodes to the front of the agenda.
- Leads to a search that explores "vertically".

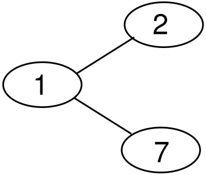
Depth-first search



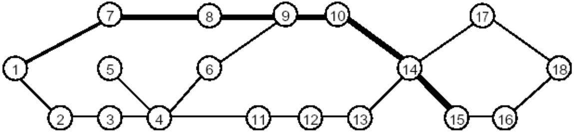


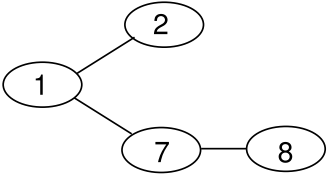
Depth-first search



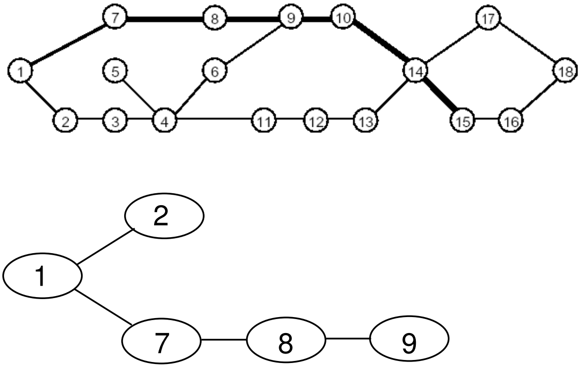


Depth-first search

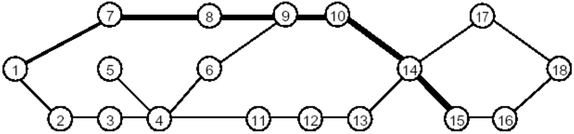


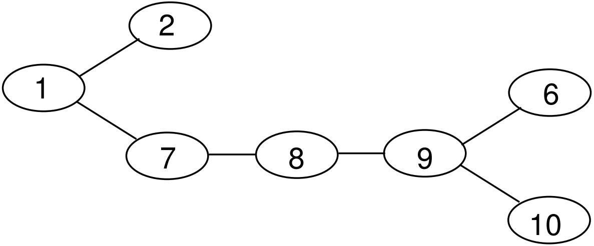


Depth-first search



Depth-first search

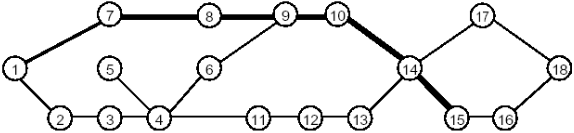


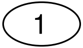


Breadth-first search

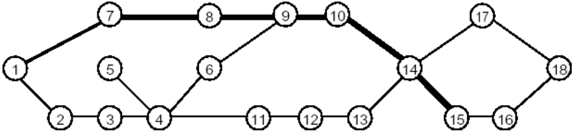
- Breadth-first search
  - Takes the first node on the agenda;
  - Adds new nodes to the back of the agenda.
- Explores all the nodes at one “level” before looking at the next level.

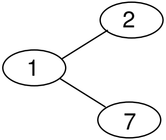
Breadth-first search



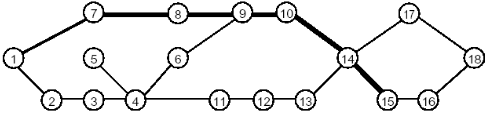


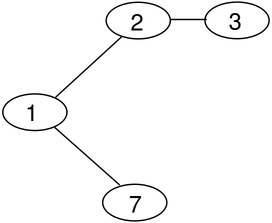
Breadth-first search



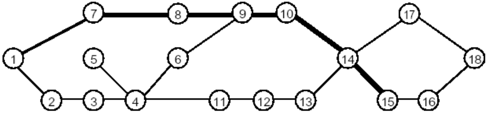


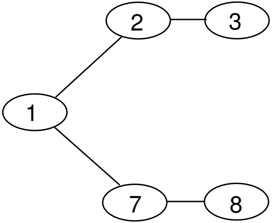
Breadth-first search



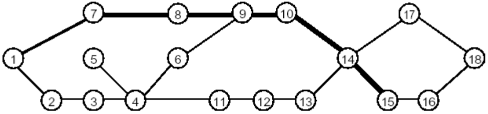


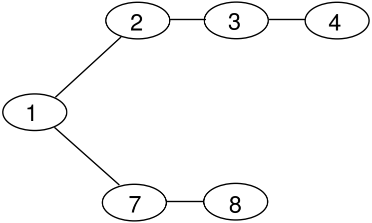
Breadth-first search



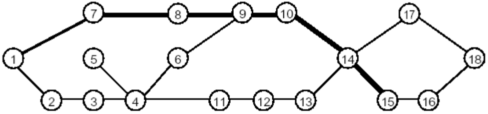


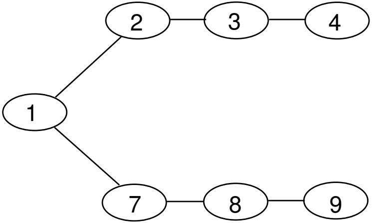
Breadth-first search



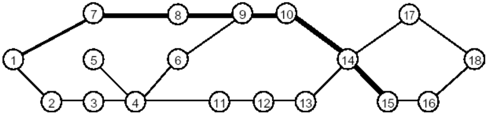


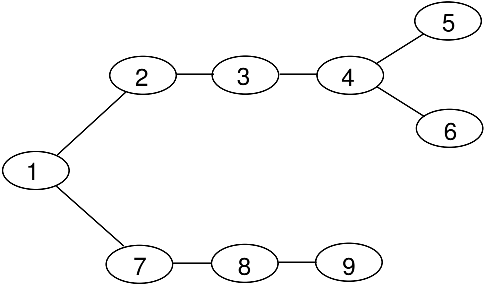
Breadth-first search



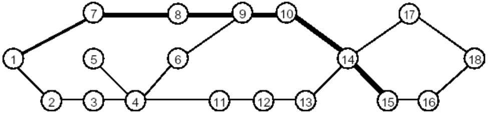


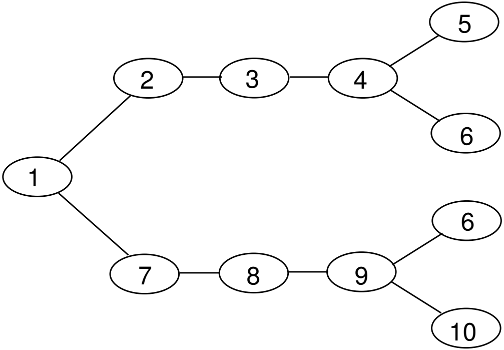
Breadth-first search





Breadth-first search





A* search

- A* search focuses the search by giving each node a pair of weights:
  - How far it is from the start; and
  - How close it is to the goal.
- The cost of the node is then the sum of the weights.
- We pick from the agenda by choosing the node with the lowest cost. (Choosing like this means we don't have to worry about what order we put nodes onto the agenda).
- Generalization of Dijkstra's algorithm.

A* search

- In some domains we have to design clever functions to determine what "far" is.
- In robotics we can just use Euclidean or Manhattan distance between points:
  - Euclidean distance

$$d^e_{s,g} = \sqrt{(x_g - x_s)^2 + (y_g - y_s)^2}$$

  - Manhattan distance

$$d^m_{s,g} = |(x_g - x_s)| + |(y_g - y_s)|$$

- Of course the distance to the goal may be an underestimate
  - may be no route through (common in Manhattan)

but it turns out that this is a good thing for A*.

A* search


Church
Cartney
Rogers
Bloomingdale Road
Street
Amory
74
73
72
71
70
69
68
67
66
65
64
63
St

A* search

D* search

- Often in robotics we need to replan
- D* is a version of A* that keeps track of the search that led to a plan and just fixes the bits that need to be fixed.
  - Dynamic A*
- Quicker than replanning from scratch.
  - Usually have to replan from the robot to the goal and the only change is near the robot.
  - That is where the robot senses failure.

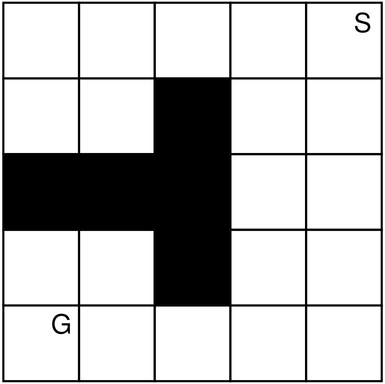
Waypoints

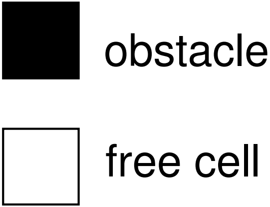
- In all these approaches we have to extract the waypoints after we find the goal.
- First we identify the sequence of cells.
  - As we search we can build a plan for each node we visit.
  - The plan for each node is the route to its parent plus the step to the node.
  - When we get to the goal we have the plan.
- Then we build a waypoint from each grid cell.
  - Typically the center of gravity of the cell.

Wavefront planning

- Also known as Grassfire, Wildfire or NF1.
- Essentially breadth-first search in a convenient form for application to grid-based maps.
- Works like this:
  1. Start at the cell containing the `goal` and label it 0.
  2. Take every unlabelled cell that is next to a cell labelled n and label it $n+1$.
  3. Repeat until the cell containing the `start` is labelled.
- Then read the sequence of cells to traverse by following the labels down from the `start`.

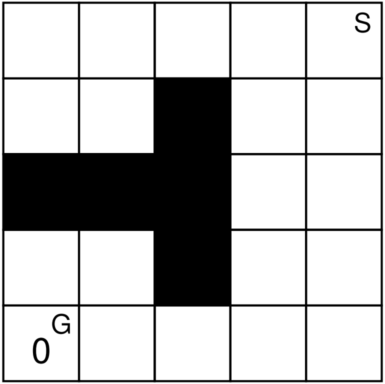
- Here's an example:

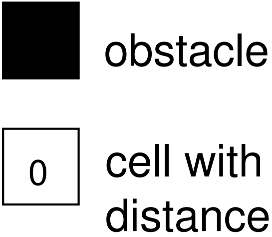




Wavefront planning

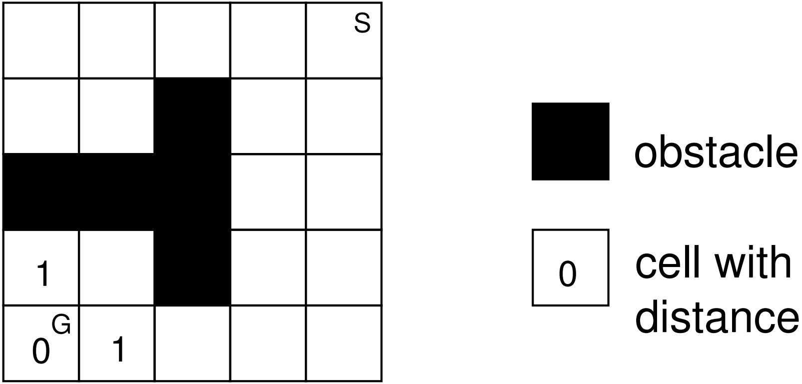
- Here's an example:





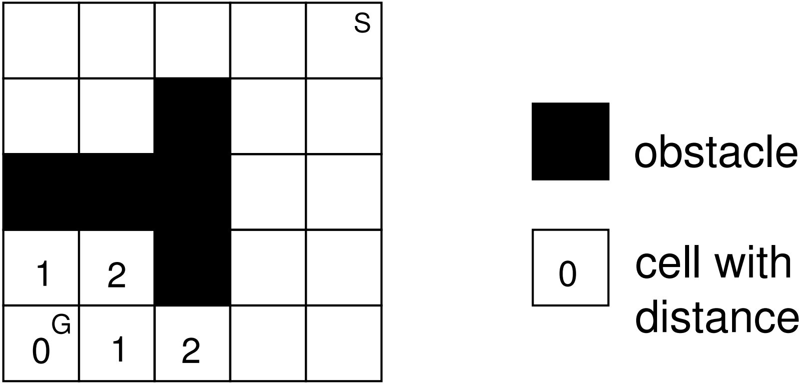
Wavefront planning

- Here's an example:



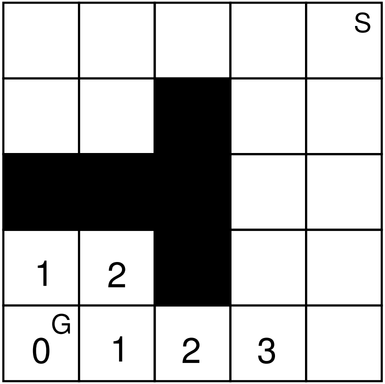
Wavefront planning

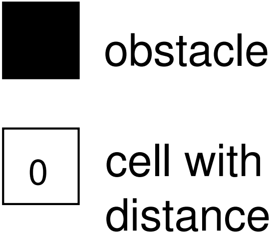
- Here's an example:



Wavefront planning

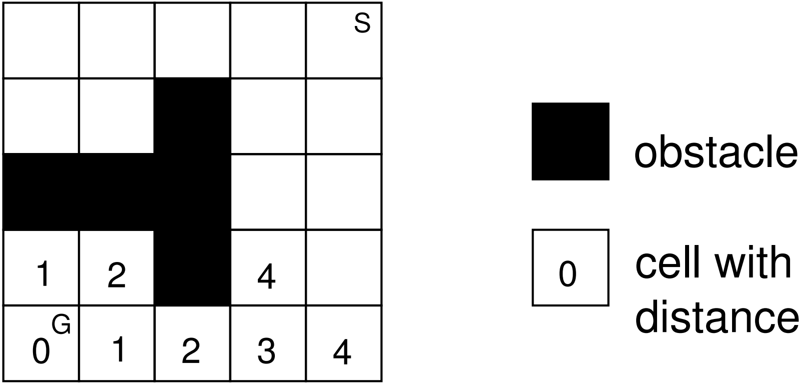
- Here's an example:





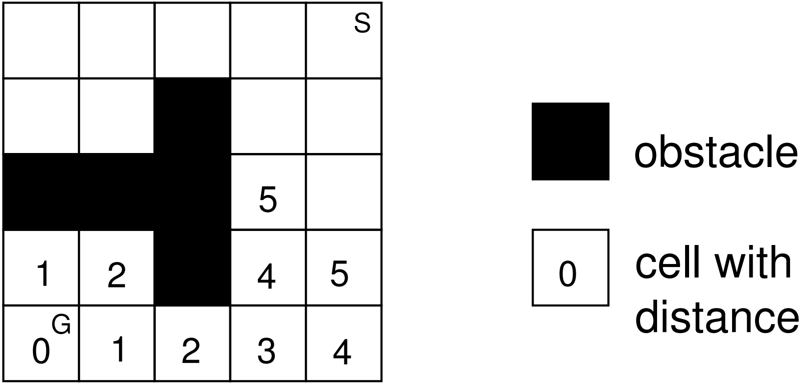
Wavefront planning

- Here's an example:



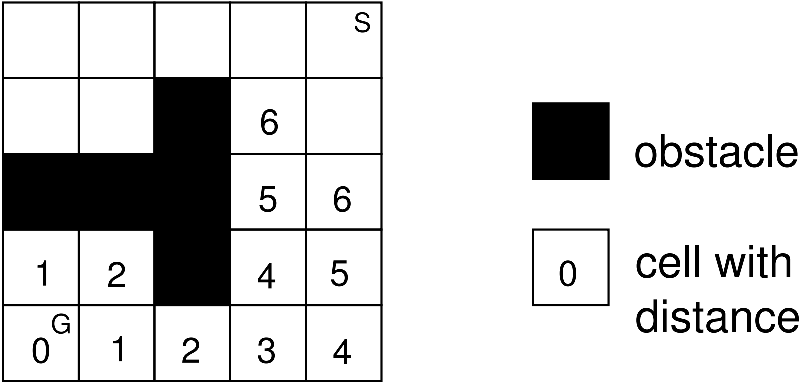
Wavefront planning

- Here's an example:

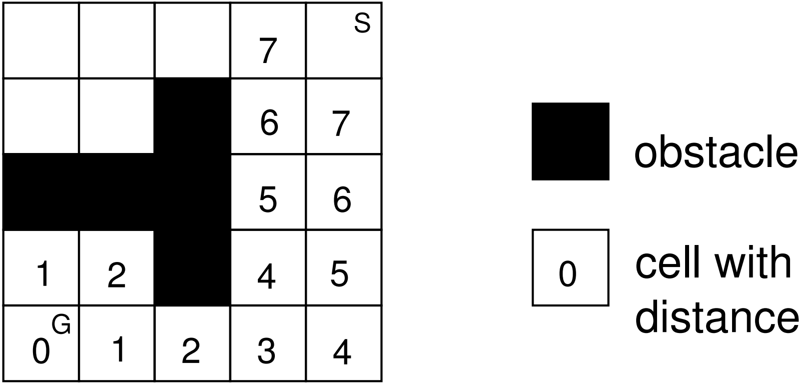


Wavefront planning

- Here's an example:

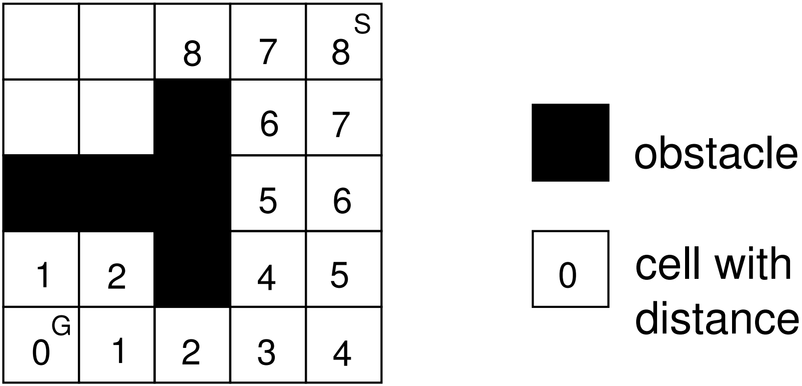


- Here's an example:



Wavefront planning

- Here's an example:

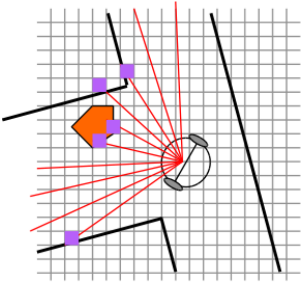


Wavefront planning

- Works especially well with occupancy grids, where the obstacles are already factored into the map.

Vector field histogram

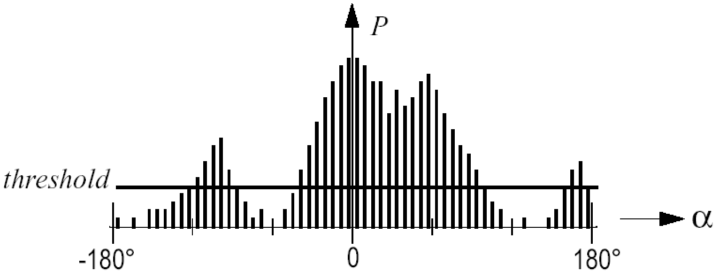
- Approach that uses sensor readings to tell the robot how to avoid obstacles.



- Representing the area around the robot as a grid, compute the probability that any square has an obstacle.
    - Robot-centric grid.
- Provides a local map to decide how the robot should move.

Vector field histogram

- The local map is reduced to a 1 DOF histogram.
  - Probability of occupancy:

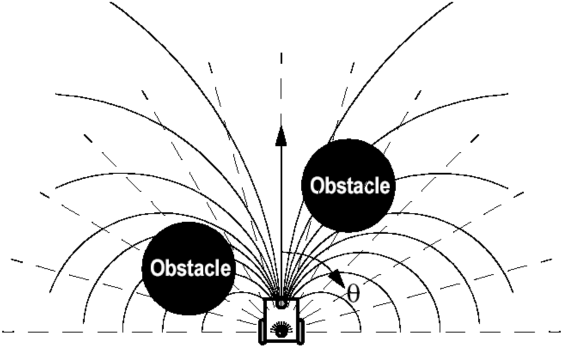


- Then compute the steering angle for the best gap.
- "Best" selected using function G which combines:

$$G = a.\ \texttt{target-direction} + b.\ \texttt{wheel-orientation}$$
$$+ c.\ \texttt{previous-direction}$$

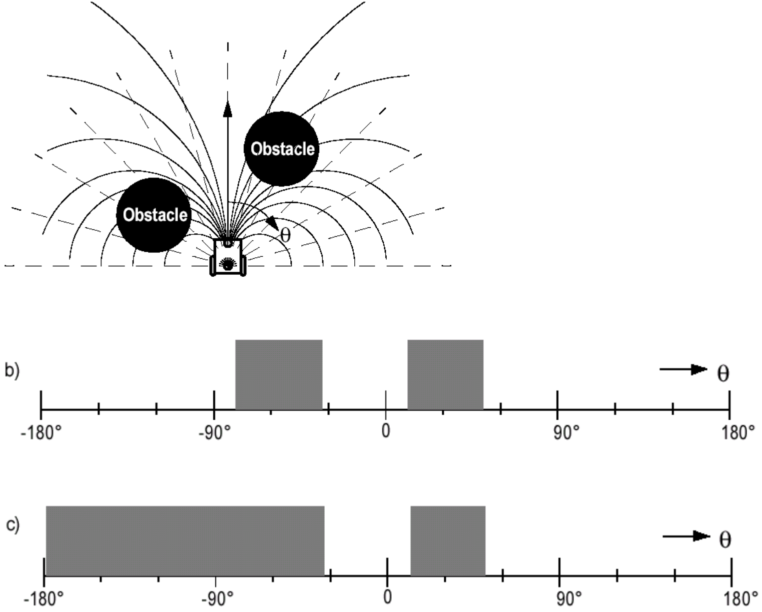
VFH+

- An issue with VFH is that it doesn't take account of how the robot can really move.



- The best gap could be one that the robot has to stop and do some complex maneuver to go through.

VFH+



VFH+

- VFH+ in action.

- `http://www.youtube.com/watch?v=84tPPOUjvSA`

Summary

- In this lecture we looked at issues to do with navigation.
  - Global navigation is about finding a path.
  - Local navigation is about avoiding obstacles.
- We looked at several examples of both.