Bug Algorithms and Path Planning

- Discussion of term projects
- A brief overview of path planning
- Various "bug"-inspired (i.e., dumb) algorithms
- Path planning and some smarter algorithms

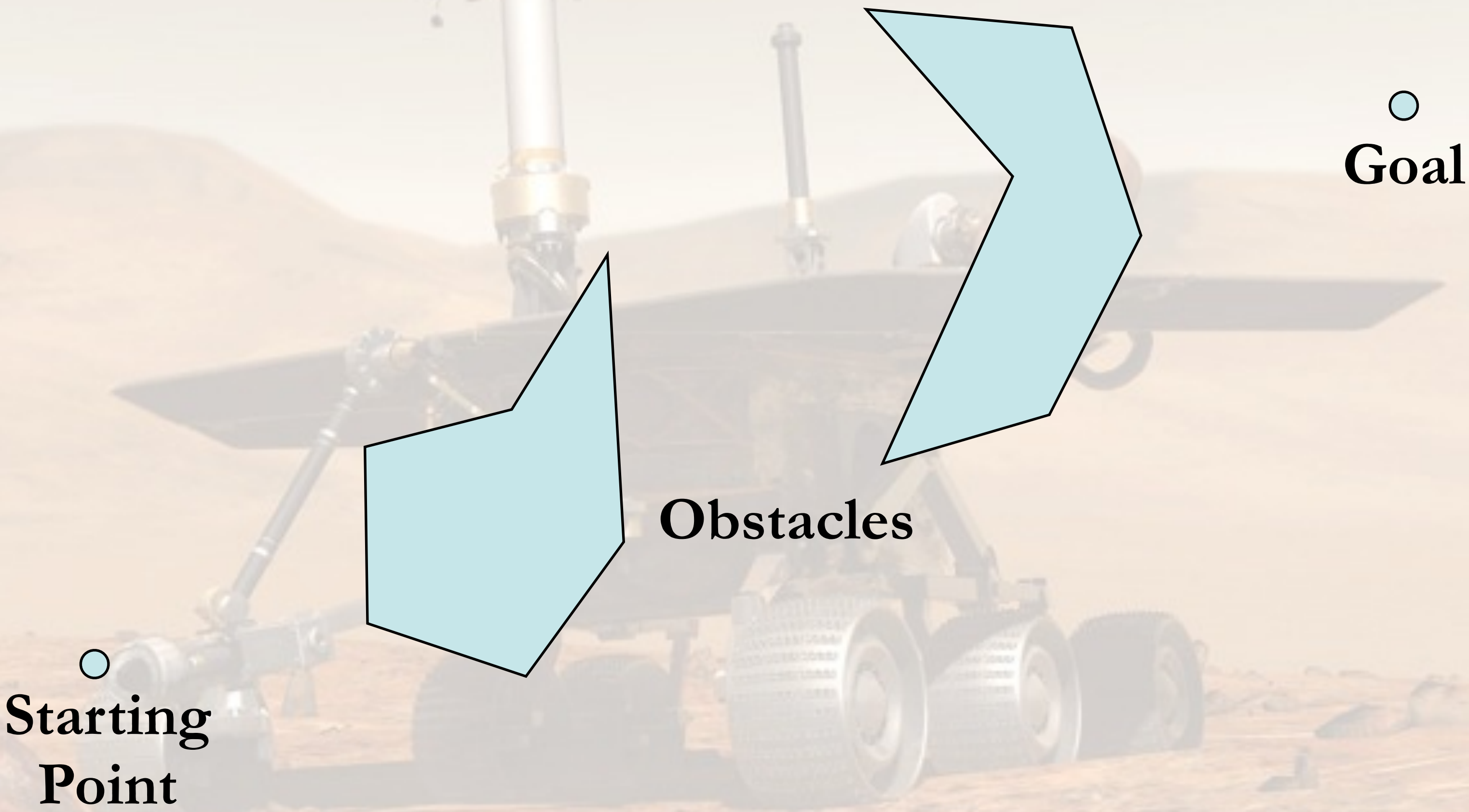
Term Design Projects

- Astronaut assistance rover
- Sample collection rover
- Minimum pressurized exploration rover
- Others by special request
- Details and top-level requirements are in slides for Lecture #01

What Can You Do? Trade Studies on...

- Number, size, placement of wheels
- Steering system
- Suspension system
- Motors and gears (coming up)
- Energetics (coming up)
- Static and dynamic stability
- Innovative solutions (legs? articulated suspensions?)

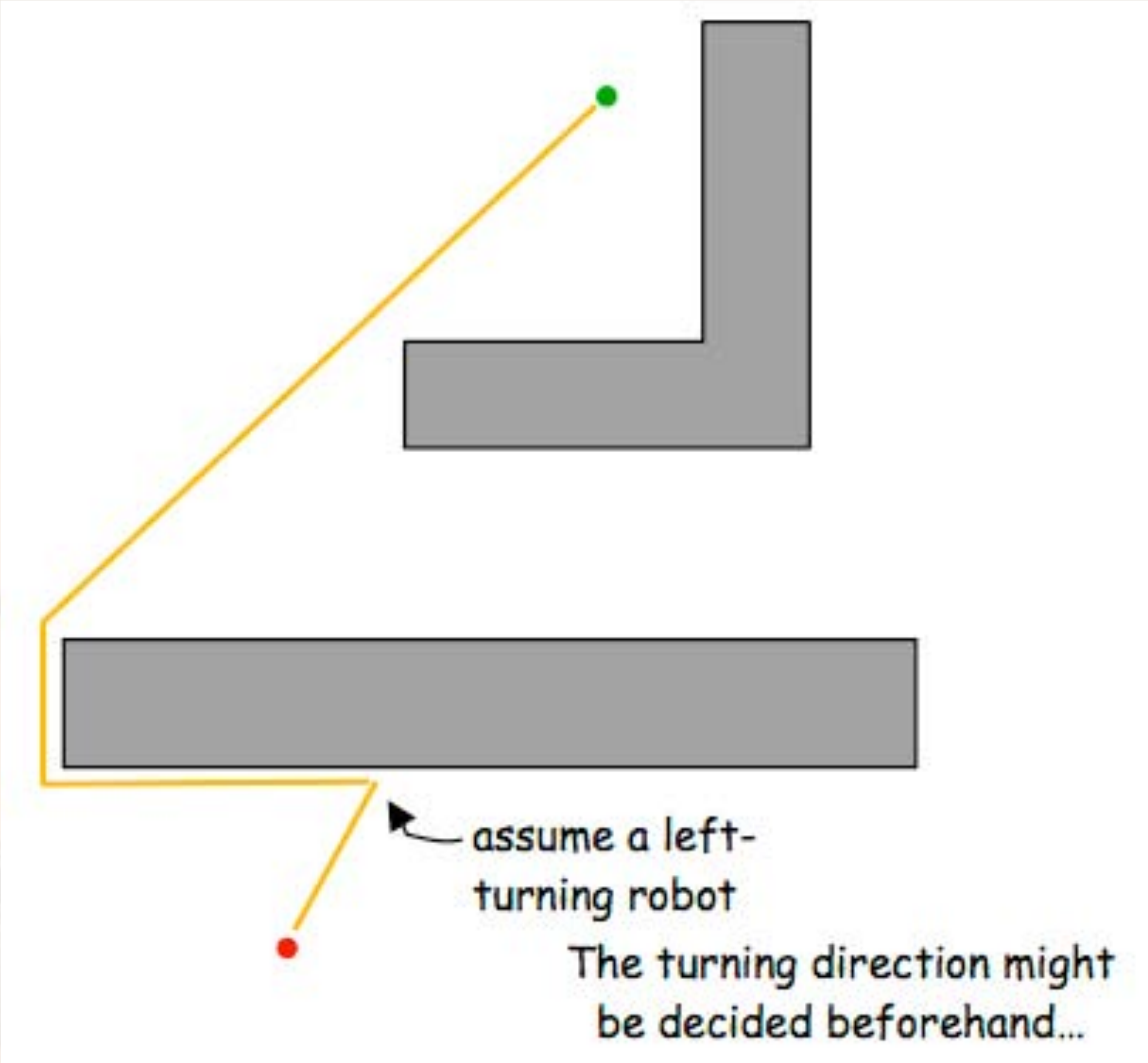
Path Planning with Obstacles



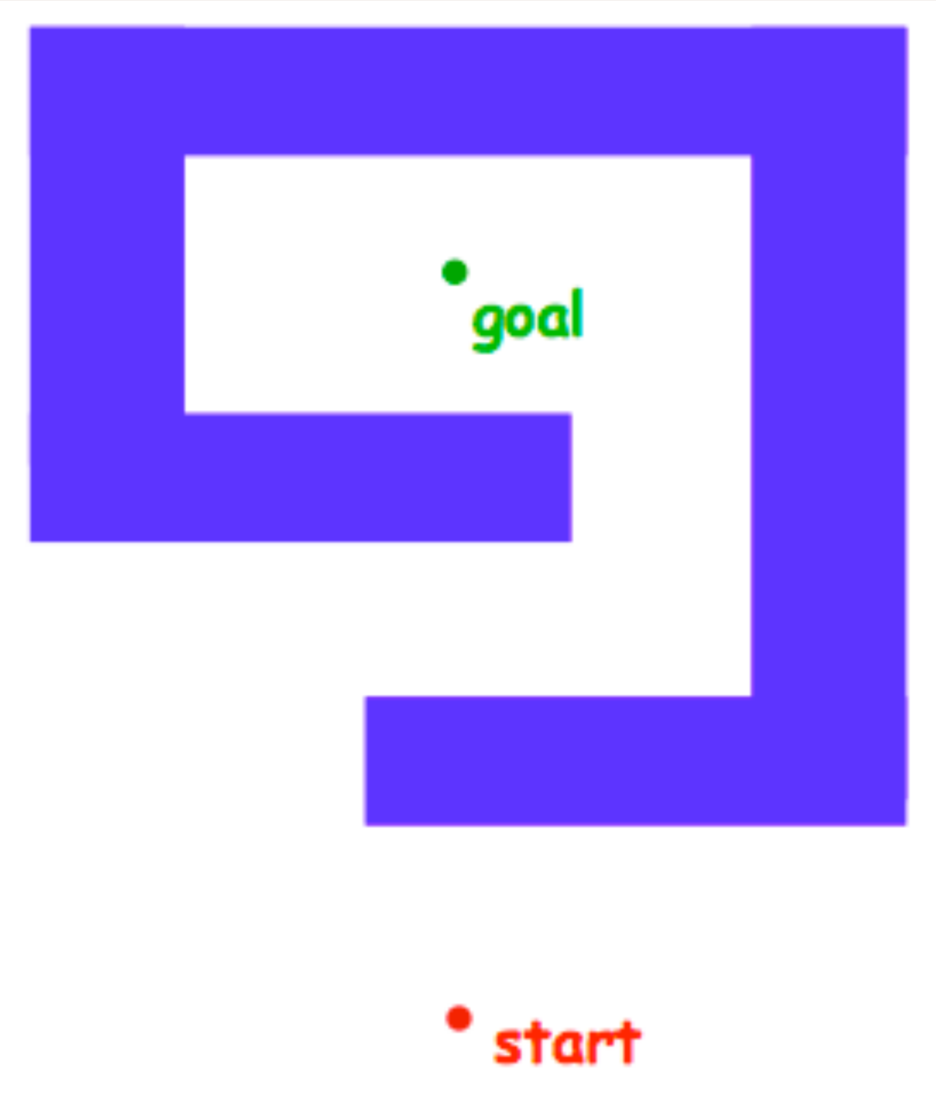
Path Planning with Bug Algorithms

- Loosely model path planning on insect capabilities
- Assumption is that rover knows its position, goal position, and can sense (at least locally) obstacles
- "Bug 0" algorithm:
  - Head towards goal
  - Follow obstacles until you can head to the goal again
  - Repeat until successful

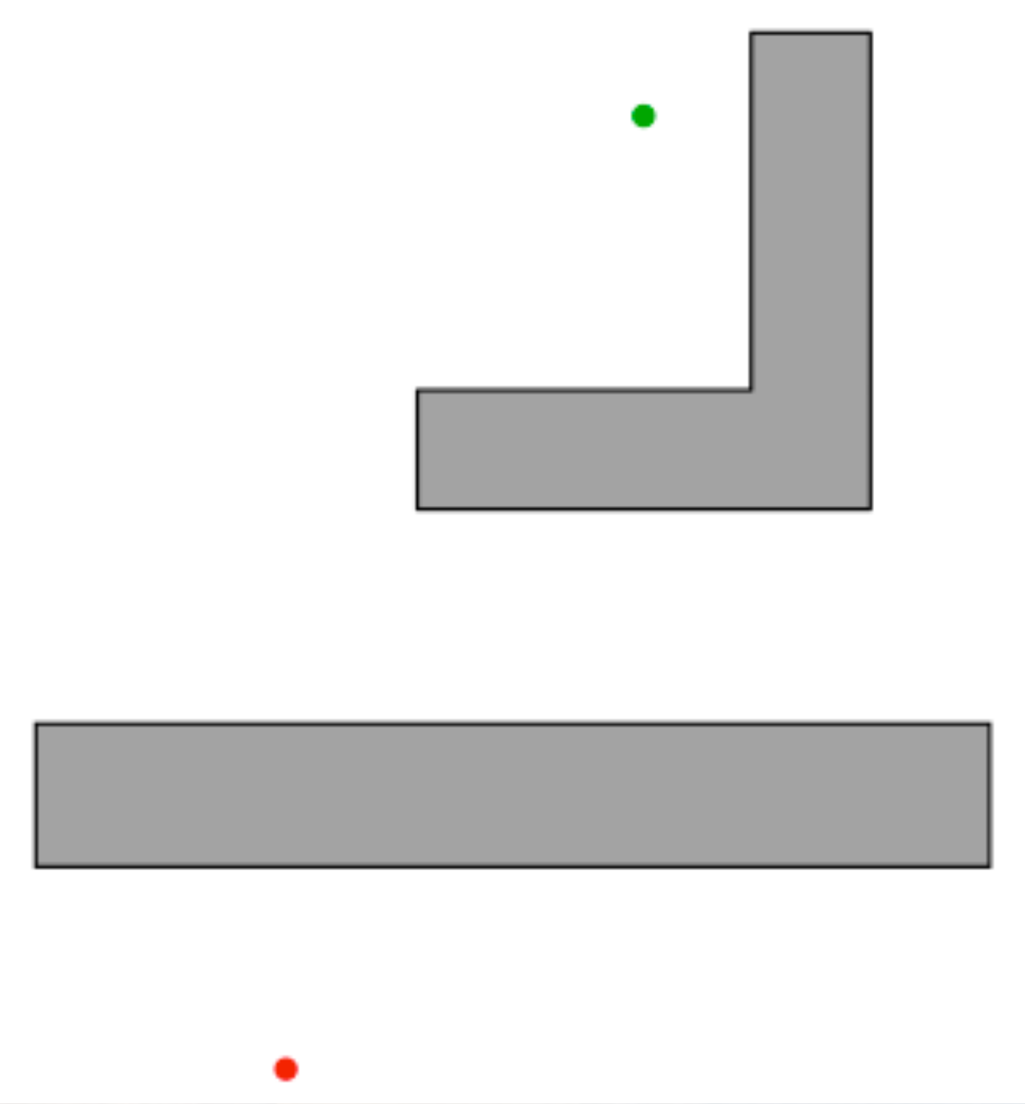
Basic Bug 0 Strategy



An Obstacle that Confounds Bug 0

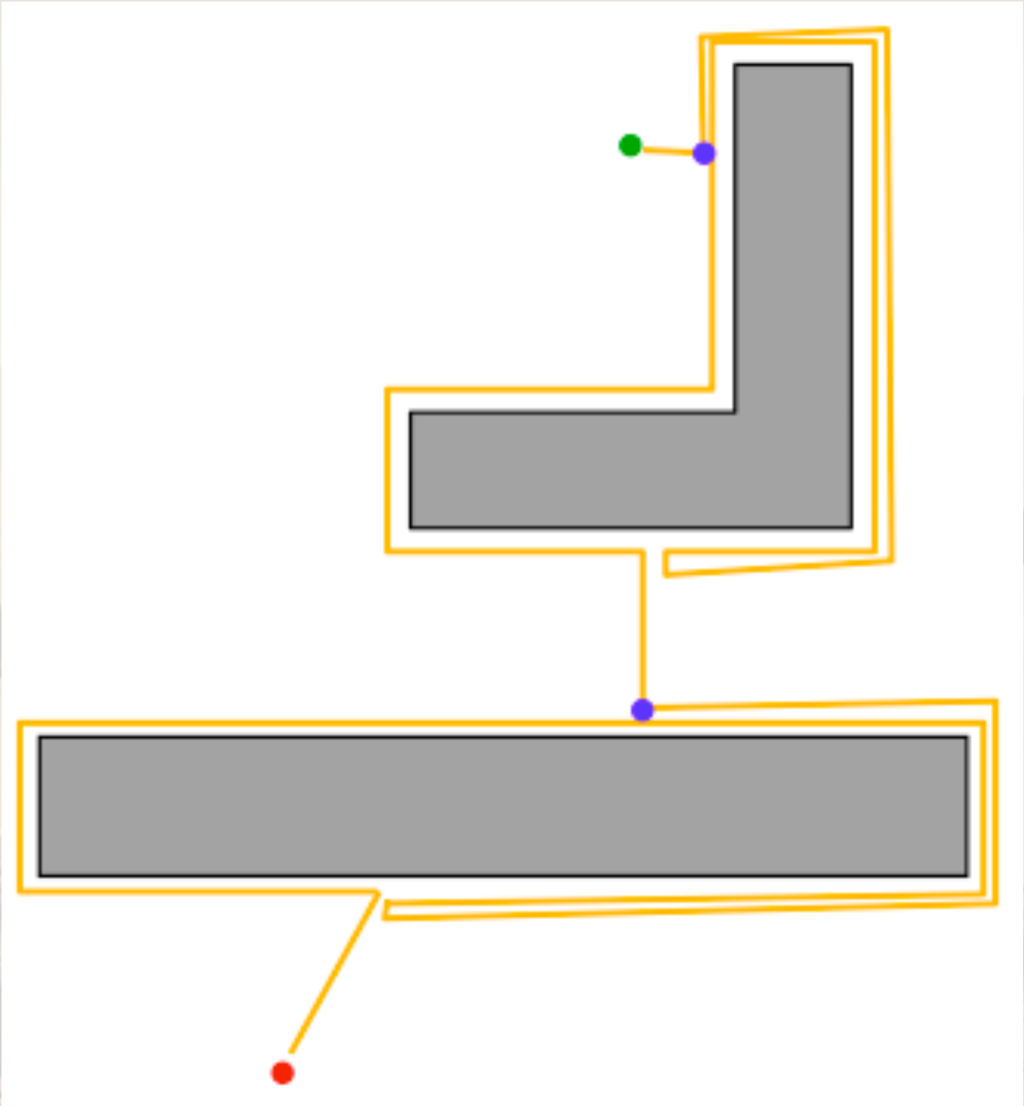


Improve Algorithm by Adding Memory

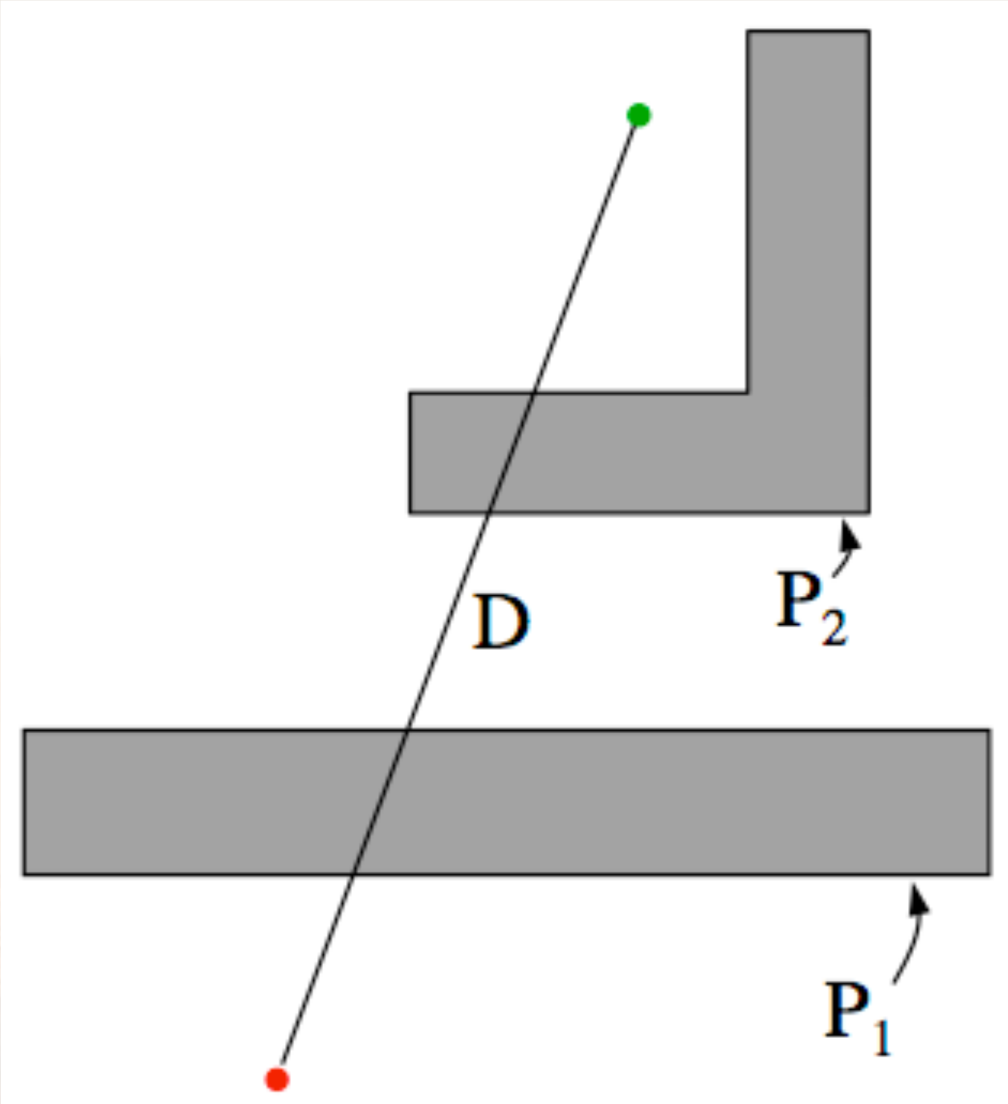


- Add memory of past locations
- When encountering an obstacle, circumnavigate and map it
- Then head to goal from point of closest approach
- "Bug 1" algorithm

Implementation of Bug 1

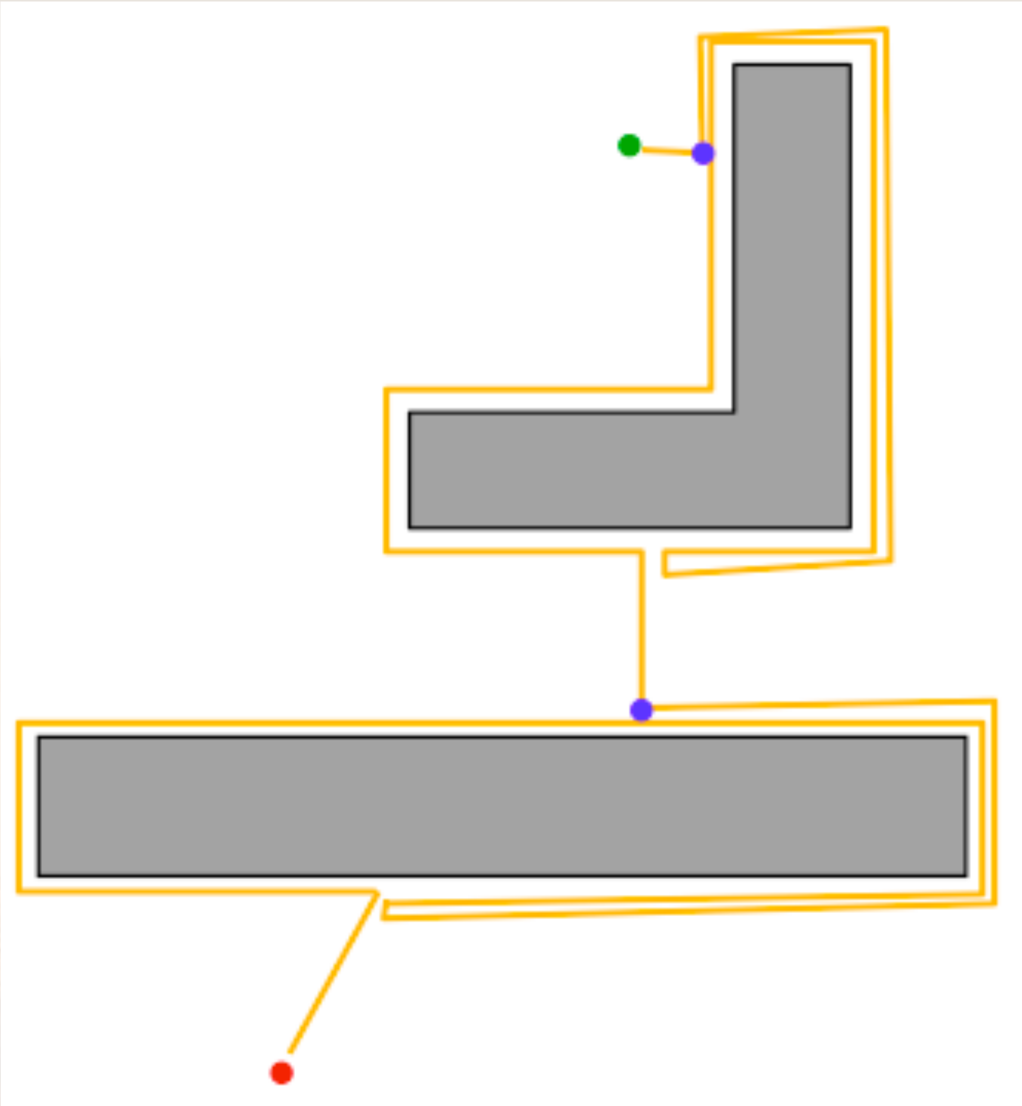


Bug 1 Path Bounds



- D=straight-line distance from start to goal
- P_i=perimeter of *i*th obstacle
- Lower Bound: shortest distance it could travel
- Upper Bound: longest distance it might travel

Bug 1 Upper and Lower Bounds



- Lower Bound:

$$D$$

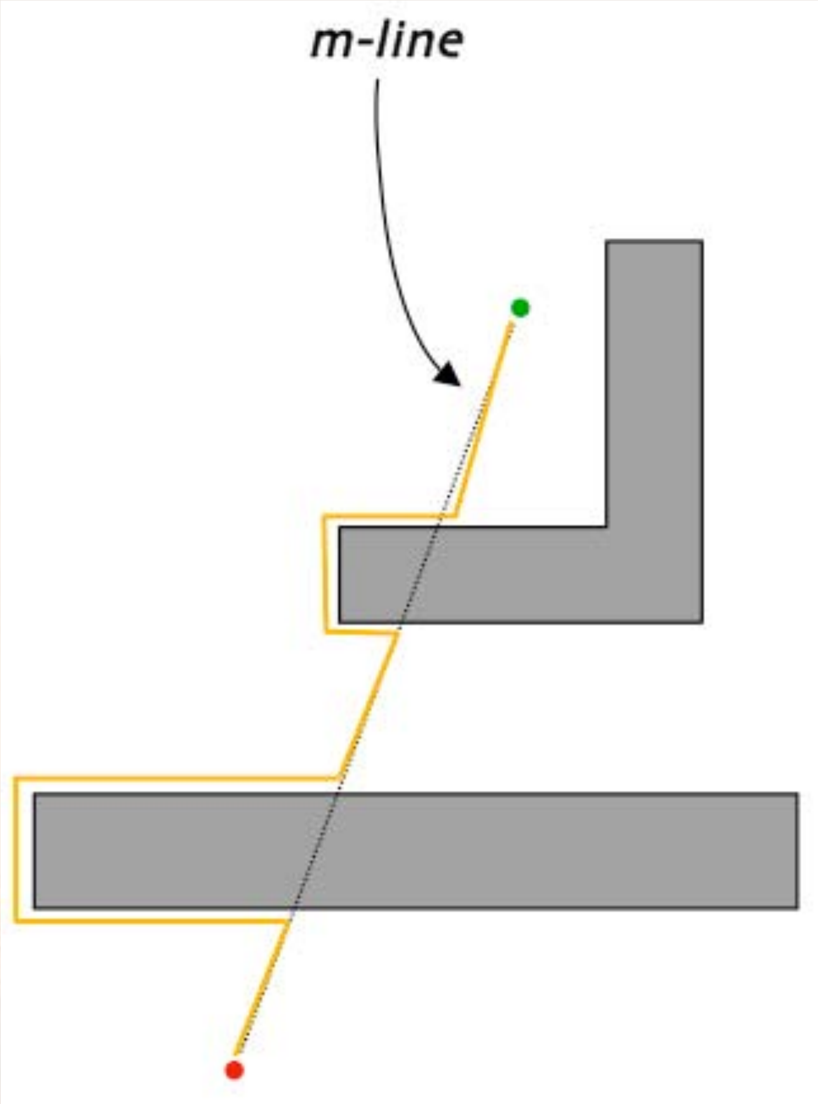
- Upper Bound:

$$D + 1.5 \sum_i P_i$$

Showing Bug 1 Completeness

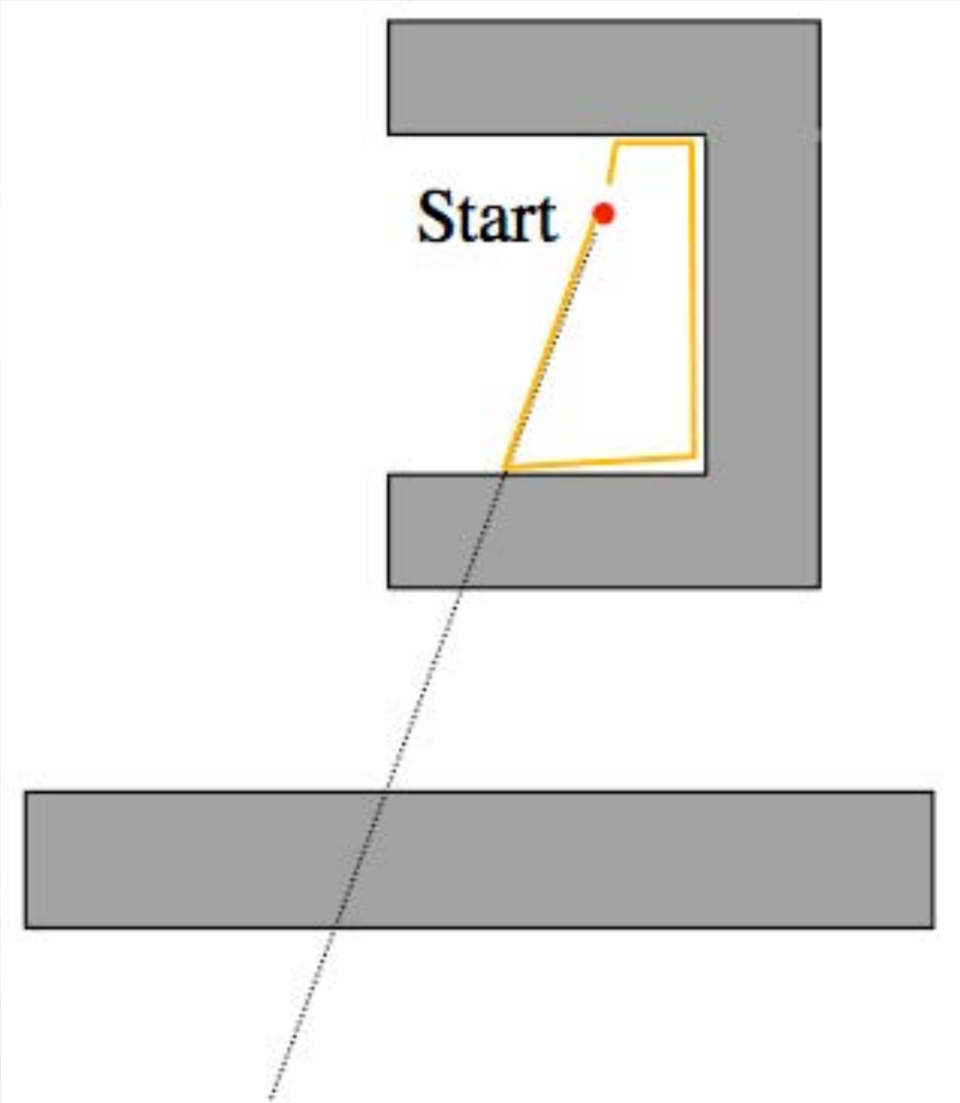
- An algorithm is *complete* if, in finite time, it finds a path if such a path exists, or terminates with failure if it does not
- Suppose Bug 1 were incomplete
  - Therefore, there is a path from start to goal
    - By assumption, it is finite length, and intersects obstacles a finite number of times
  - Bug 1 does not find the patch
    - Either it terminates incorrectly, or spends infinite time looking
    - Suppose it never terminates
      - Each leave point is closer than the corresponding hit point
      - Each hit point is closer than the previous leave point
      - There are a finite number of hit/leave pairs; after exhausting them, the robot will proceed to the goal and terminate
    - Suppose it terminates incorrectly - the closest point after a hit must be a leave
      - But the line must intersect objects an even number of times
      - There must be another intersection on the path closer to the object, but we must have passed this on the body, which contradicts definition of a leave point
- Therefore Bug 1 is complete

Bug 2 Algorithm



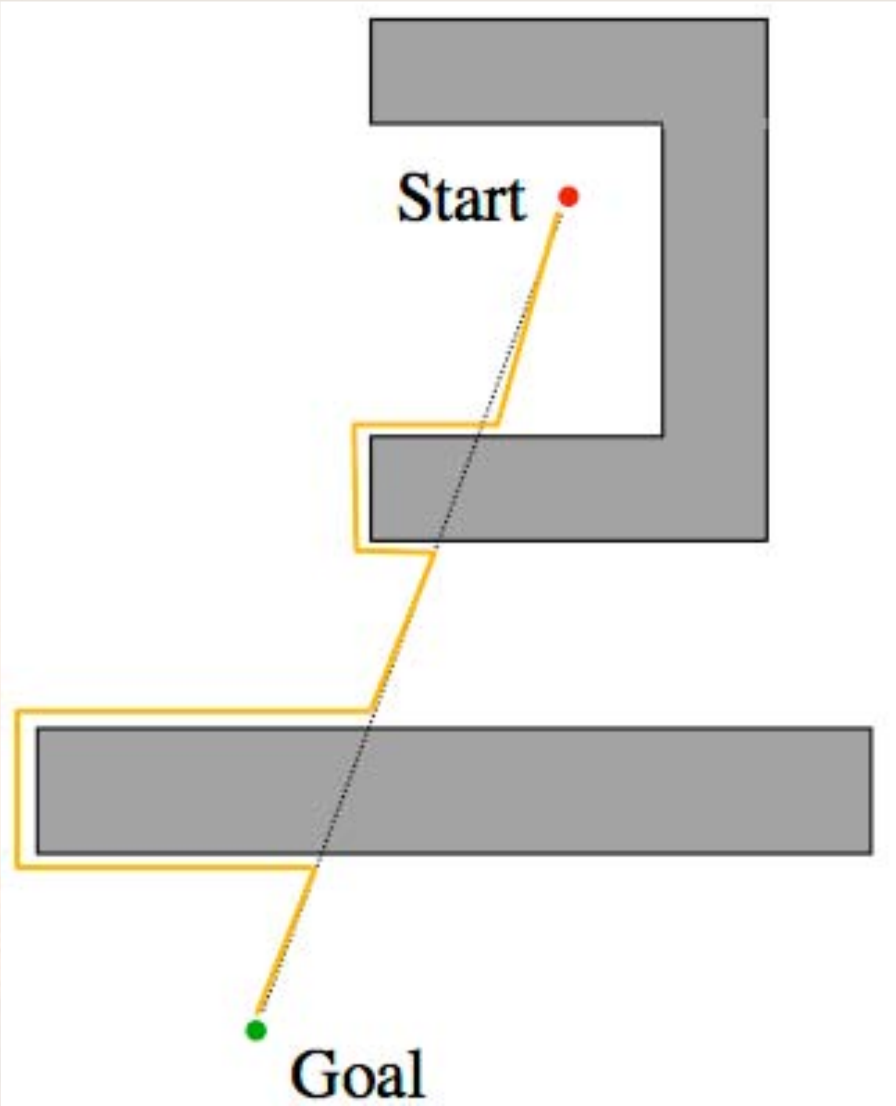
- Create an m-line connecting the starting and goal points
- Head toward goal on the m-line
- Upon encountering obstacle, follow it until you re-encounter the m-line
- Leave the obstacle and follow m-line toward goal

But This Bug 2 Doesn't Always Work



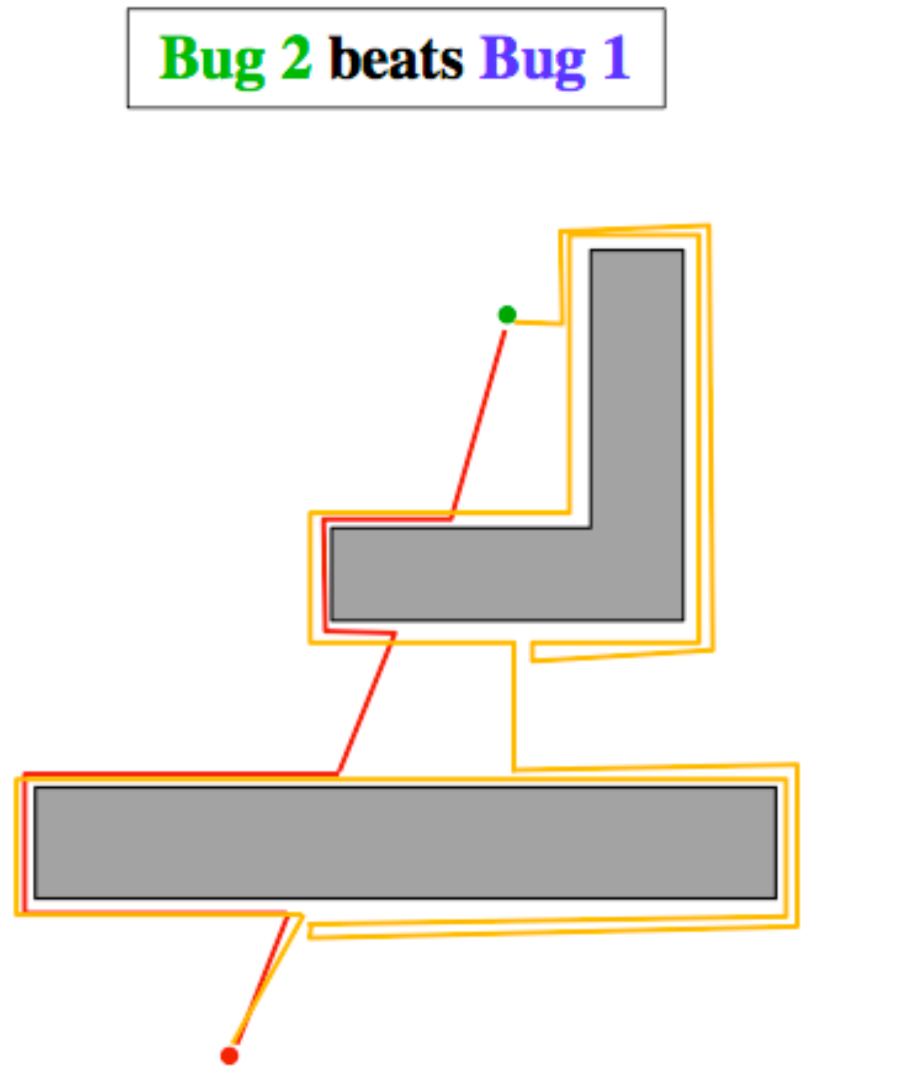
- In this case, re-encountering the m-line brings you back to the start
- Implicitly assuming a static strategy for encountering the obstacle ("always turn left")

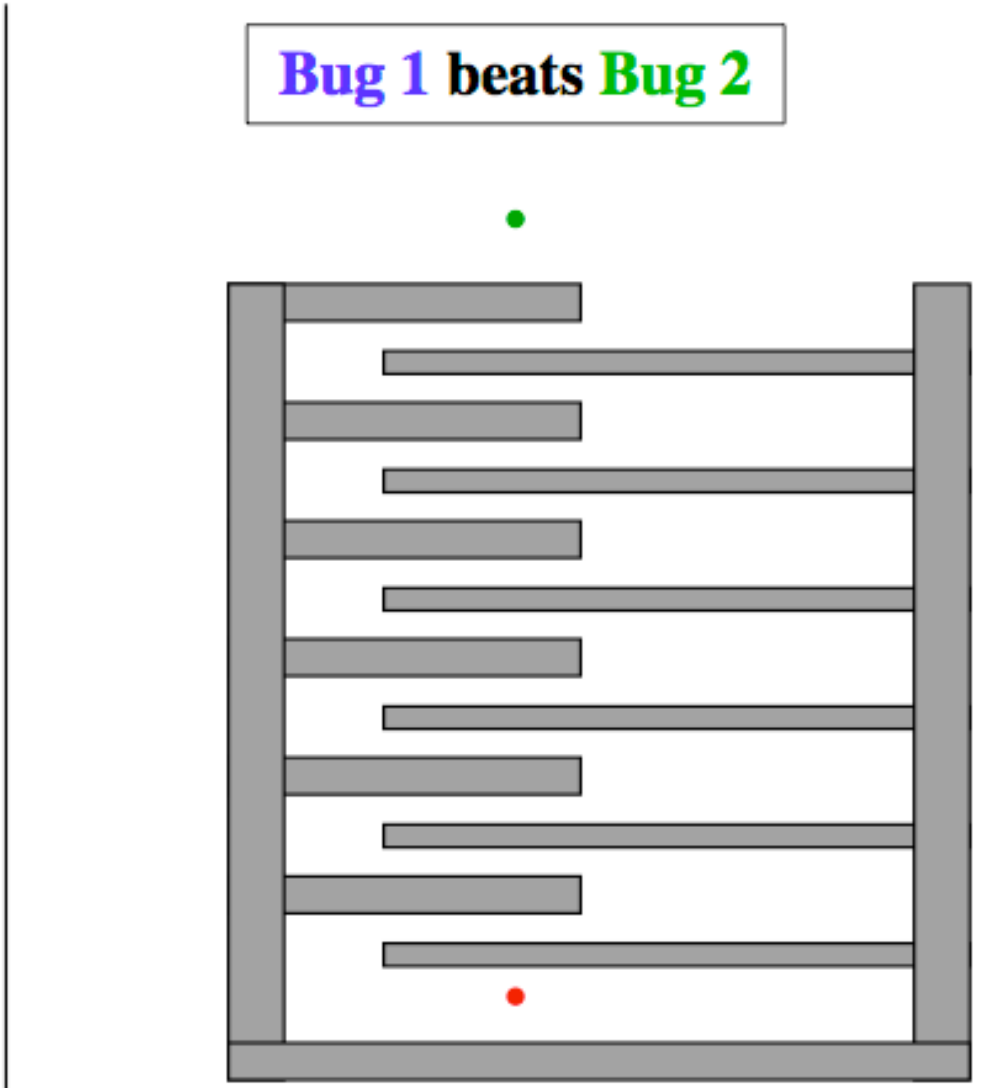
Bug 2 Algorithm



- Head toward the goal on the m-line
- If an obstacle is encountered, follow it until you encounter the m-line again closer to the goal
- Leave the obstacle and continue on m-line toward the goal

Comparison of Bug 1 and Bug 2

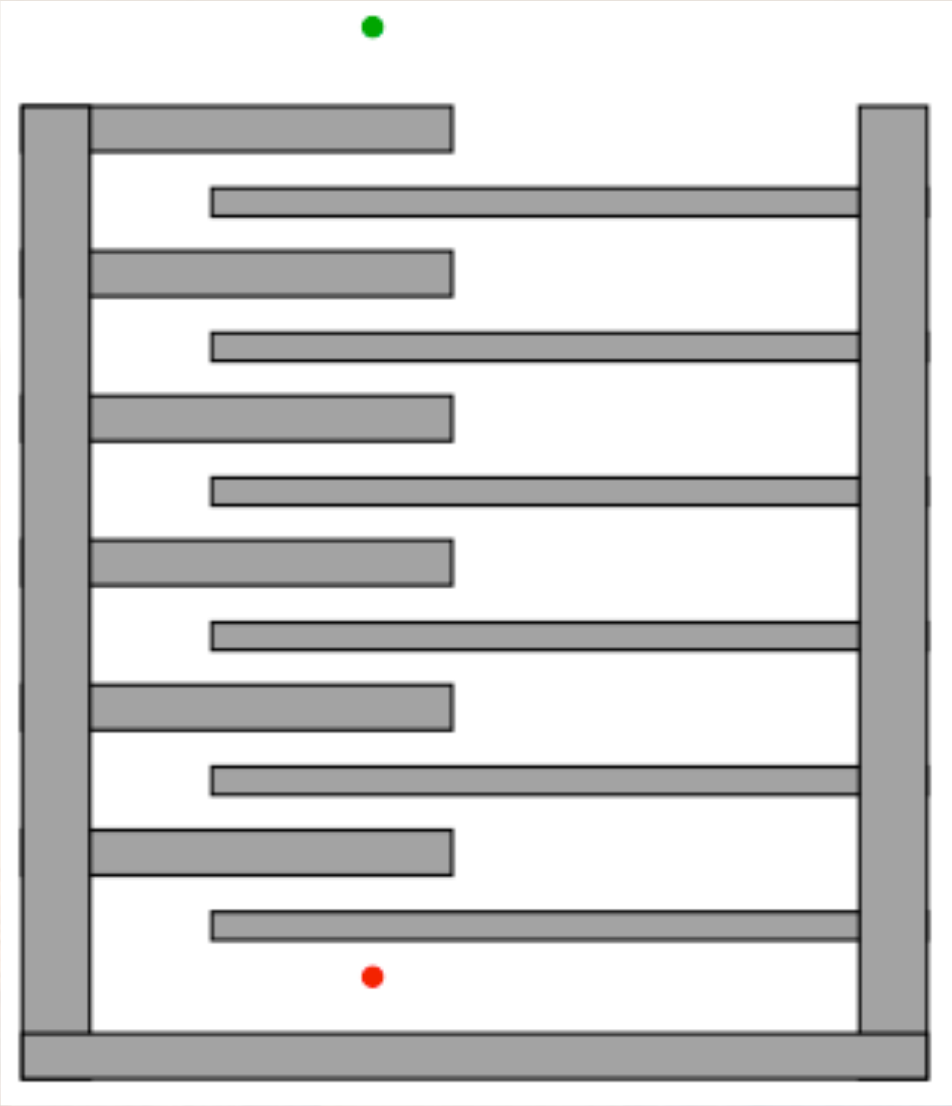




Bug 1 vs. Bug 2

- Bug 1 is an exhaustive search algorithm - it looks at all choices before commiting
- Bug 2 is a greedy algorithm - it takes the first opportunity that looks better
- In many cases, Bug 2 will outperform Bug 1, but
- Bug 1 has a more predictable performance overall

Bug 2 Upper and Lower Bounds



- Lower Bound:

$$D$$

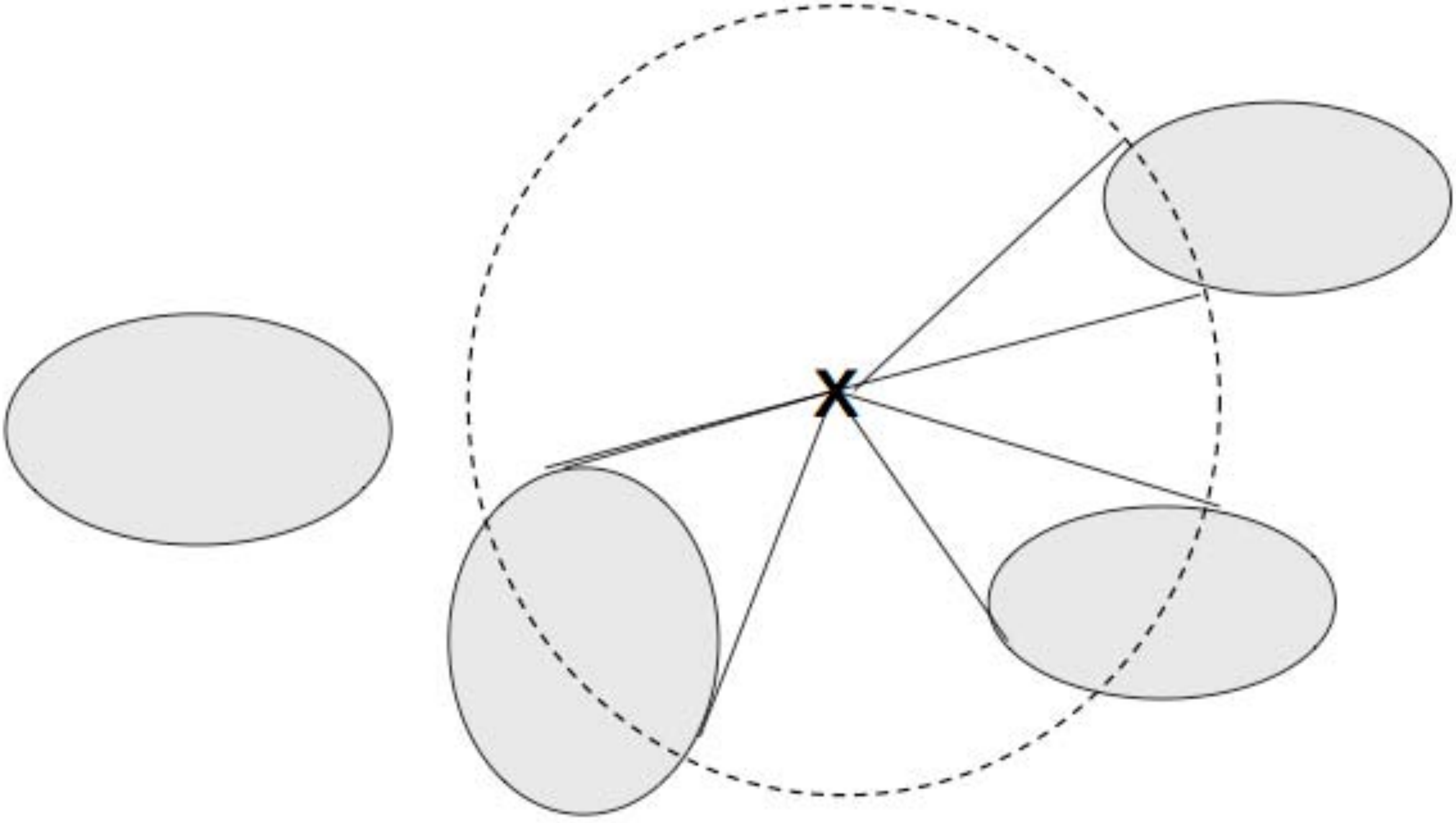
- Upper Bound:

$$D + \sum_i n_i \frac{P_i}{2}$$

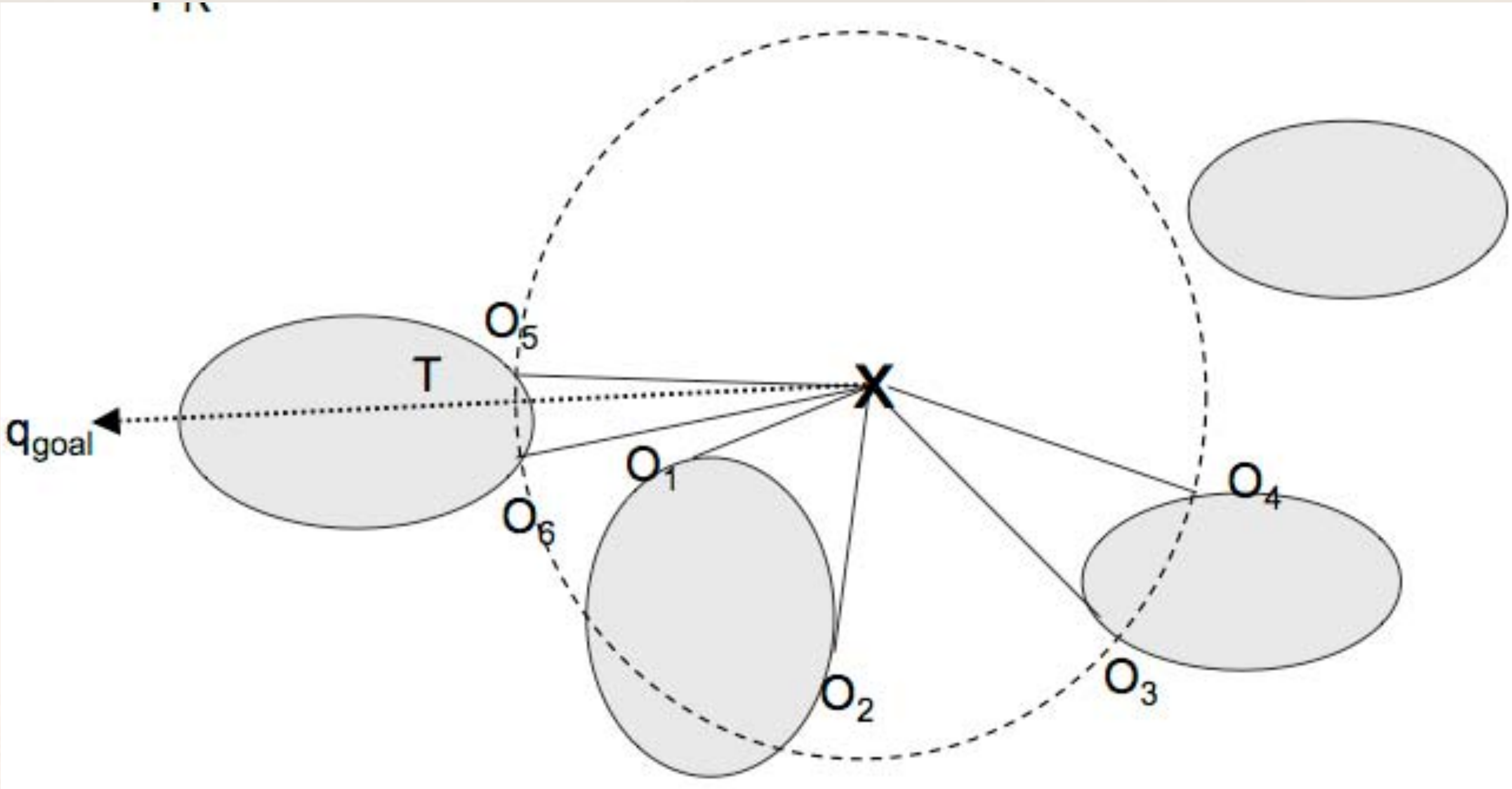
More Realistic Bug Algorithm

- Knowledge of
  - Goal point location (global beacons)
  - Wall following (contact sensors)
- Add a range sensor (with limited range and noise)
- Focus on finding endpoints of finite, continuous segments of obstacles

More Realistic Algorithm - Tangent Bug

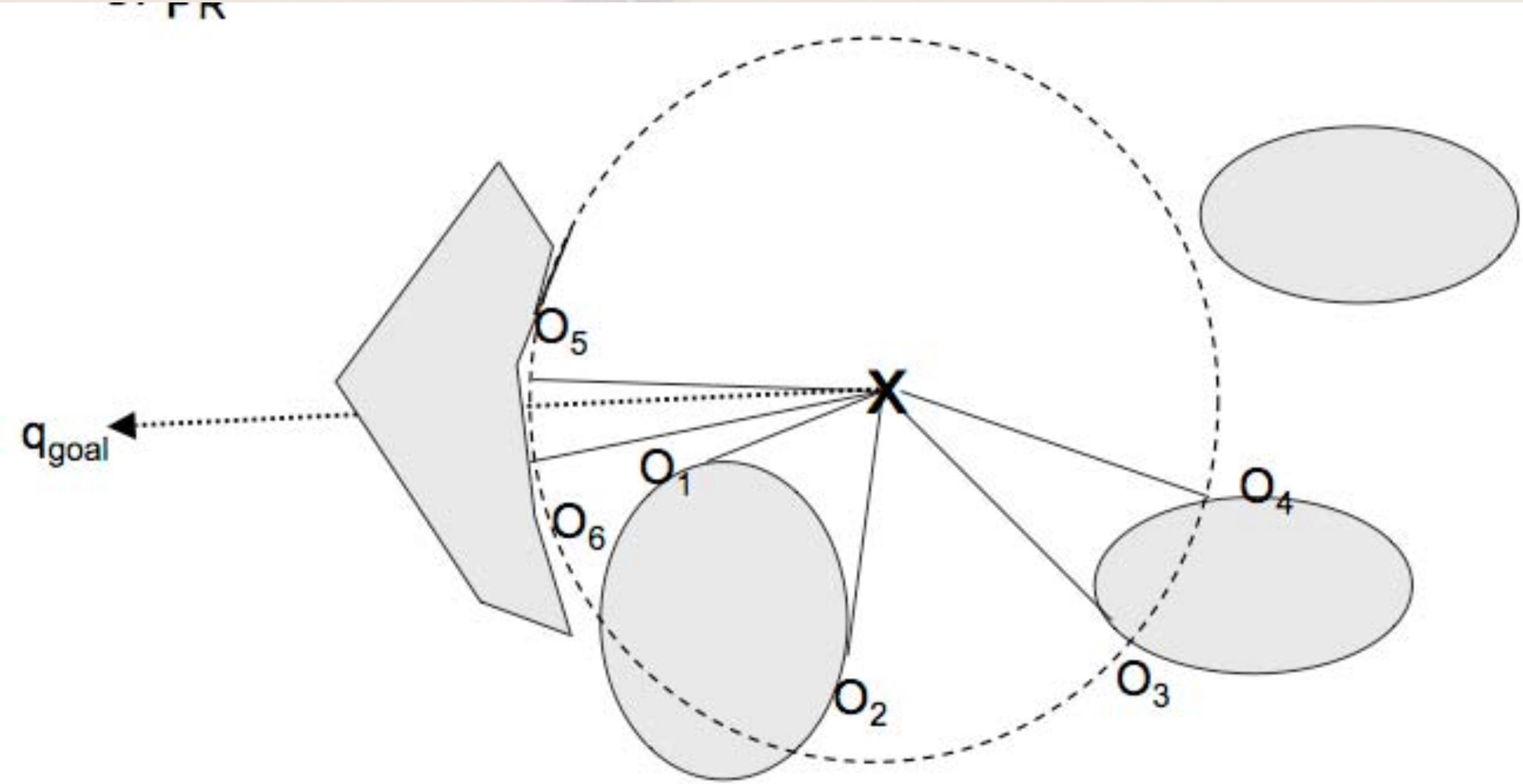


Implementation of Tangent Bug



Choose the target point O_i that minimizes $\widehat{xO_i} + \widehat{O_i q_{goal}}$

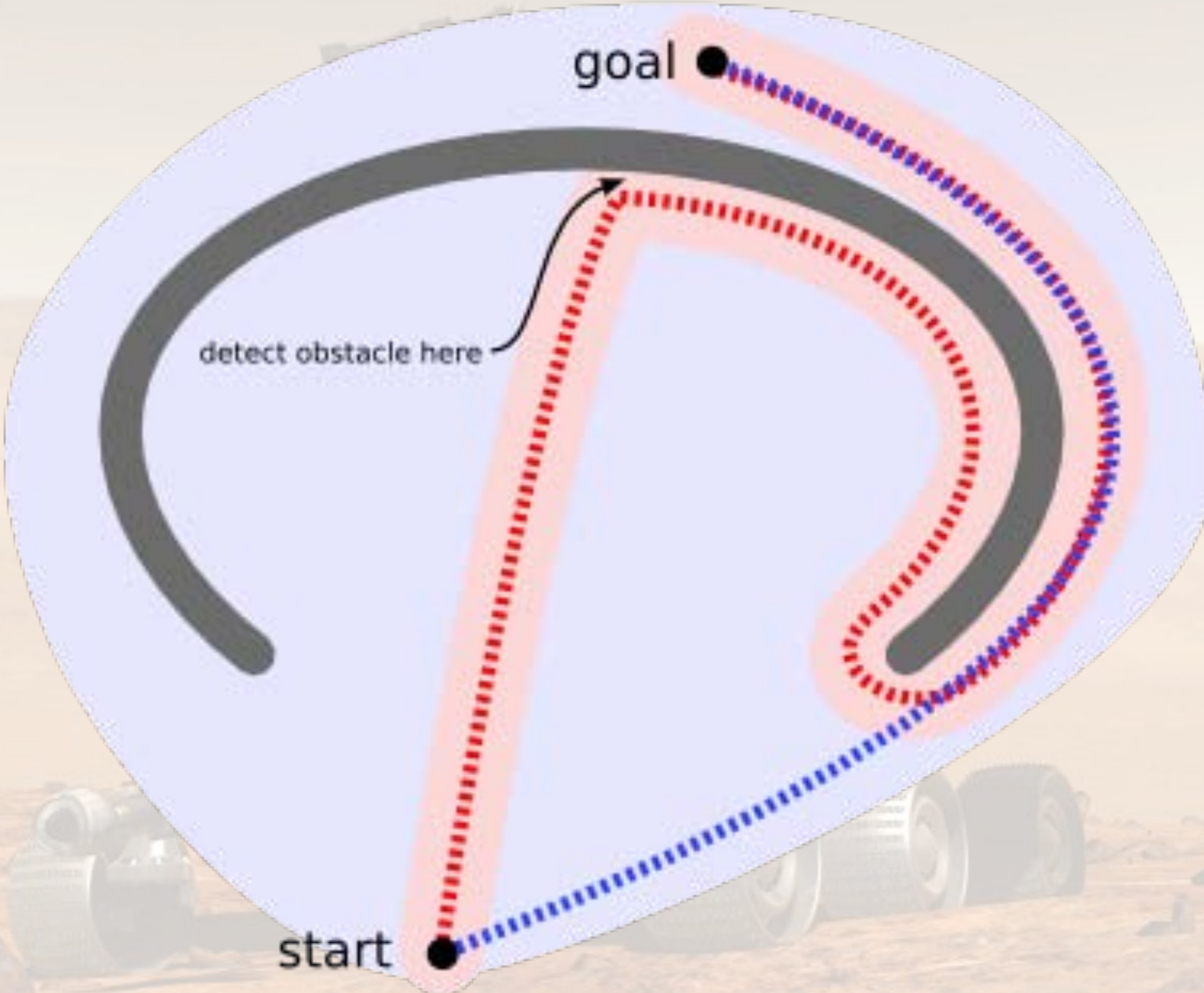
Encountering Extended Obstacles



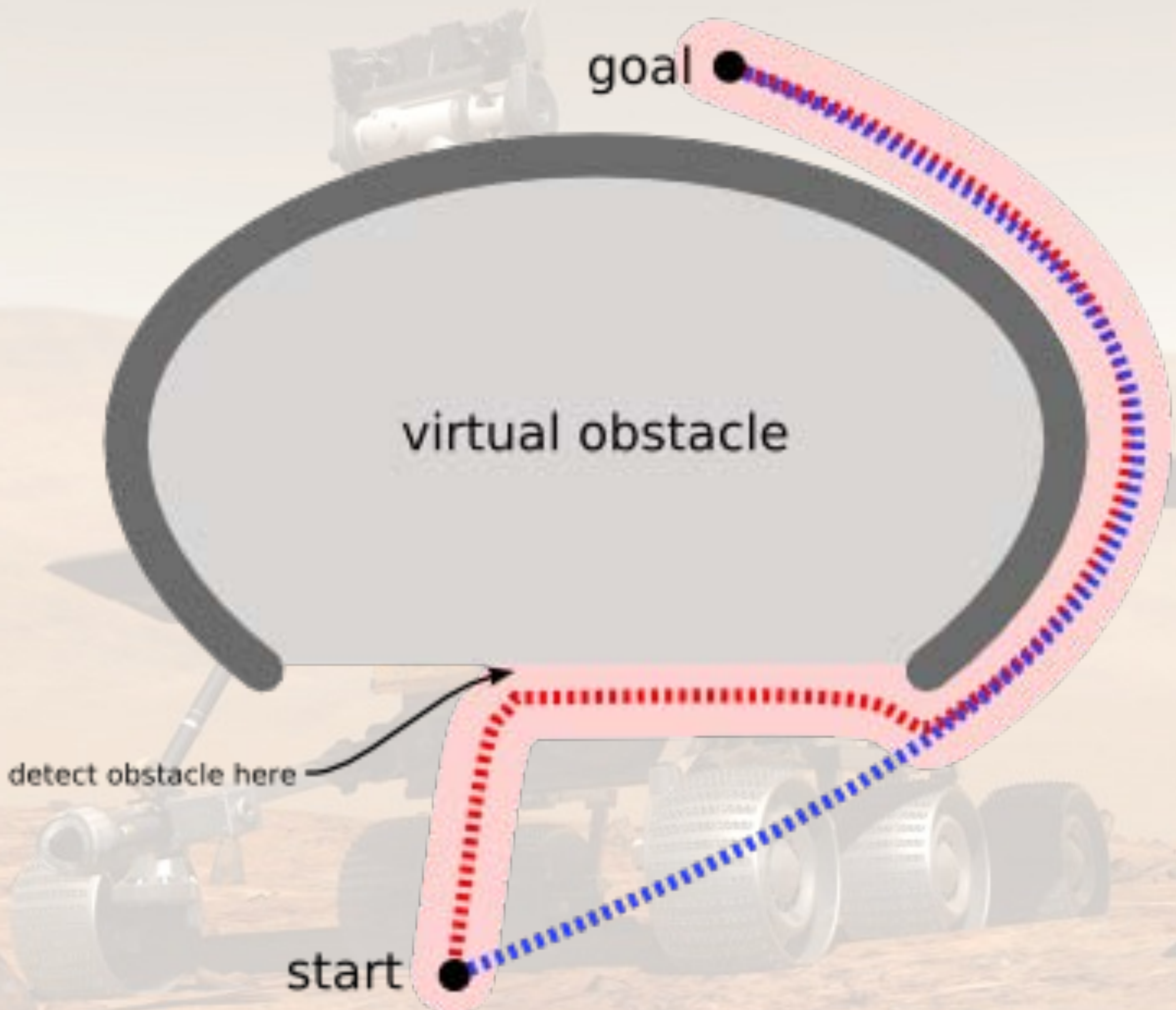
Path Planning

- How do we get to where we want to go?
- Gridded workspaces
- Formal search methods (e.g., Dijkstra)
- Heuristic search methods (e.g., Best-First)
- Hybrid search methods (A* and variants)

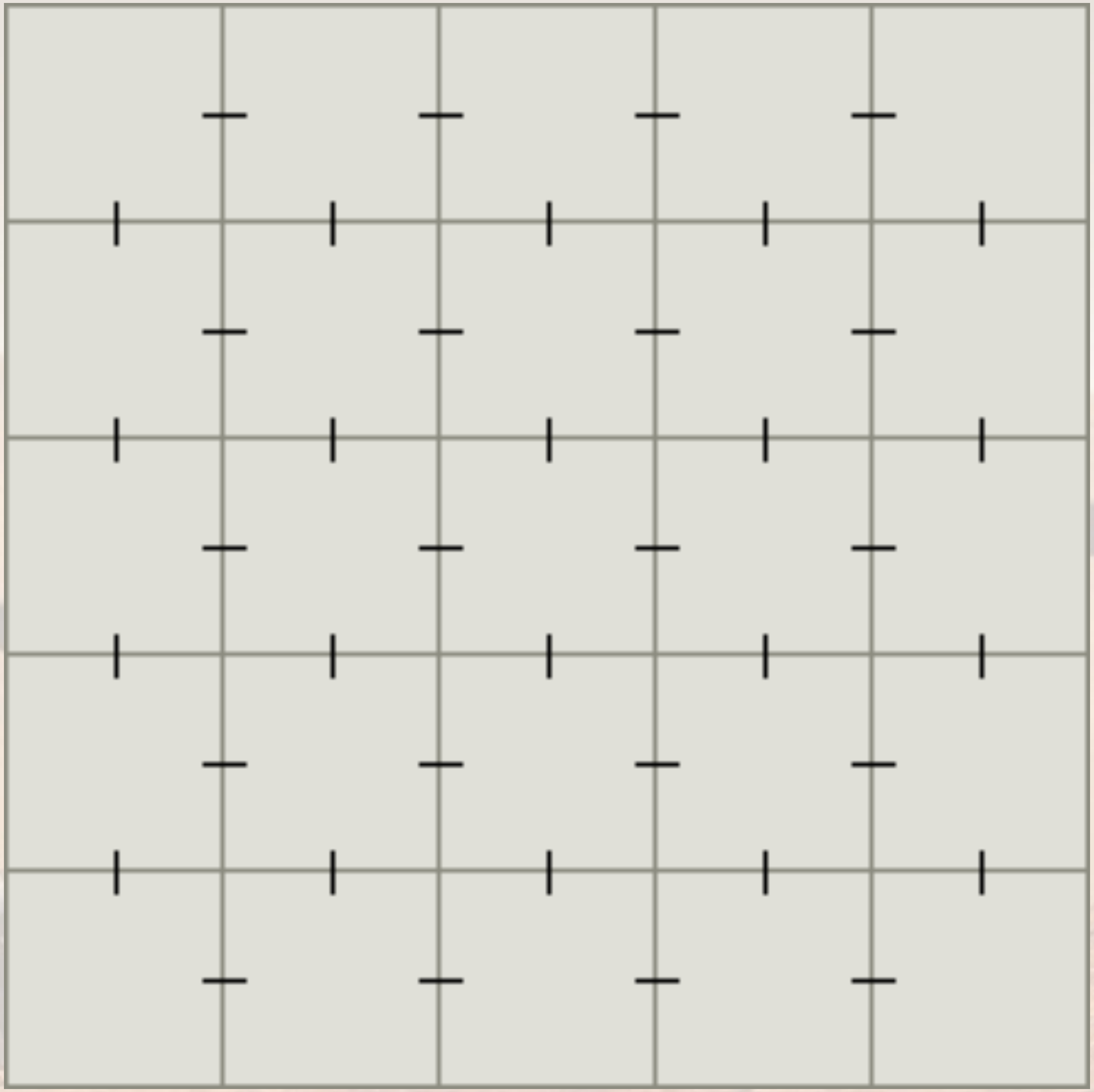
Path Planning - Potentials and Pitfalls



Avoid Entering Enclosed Spaces



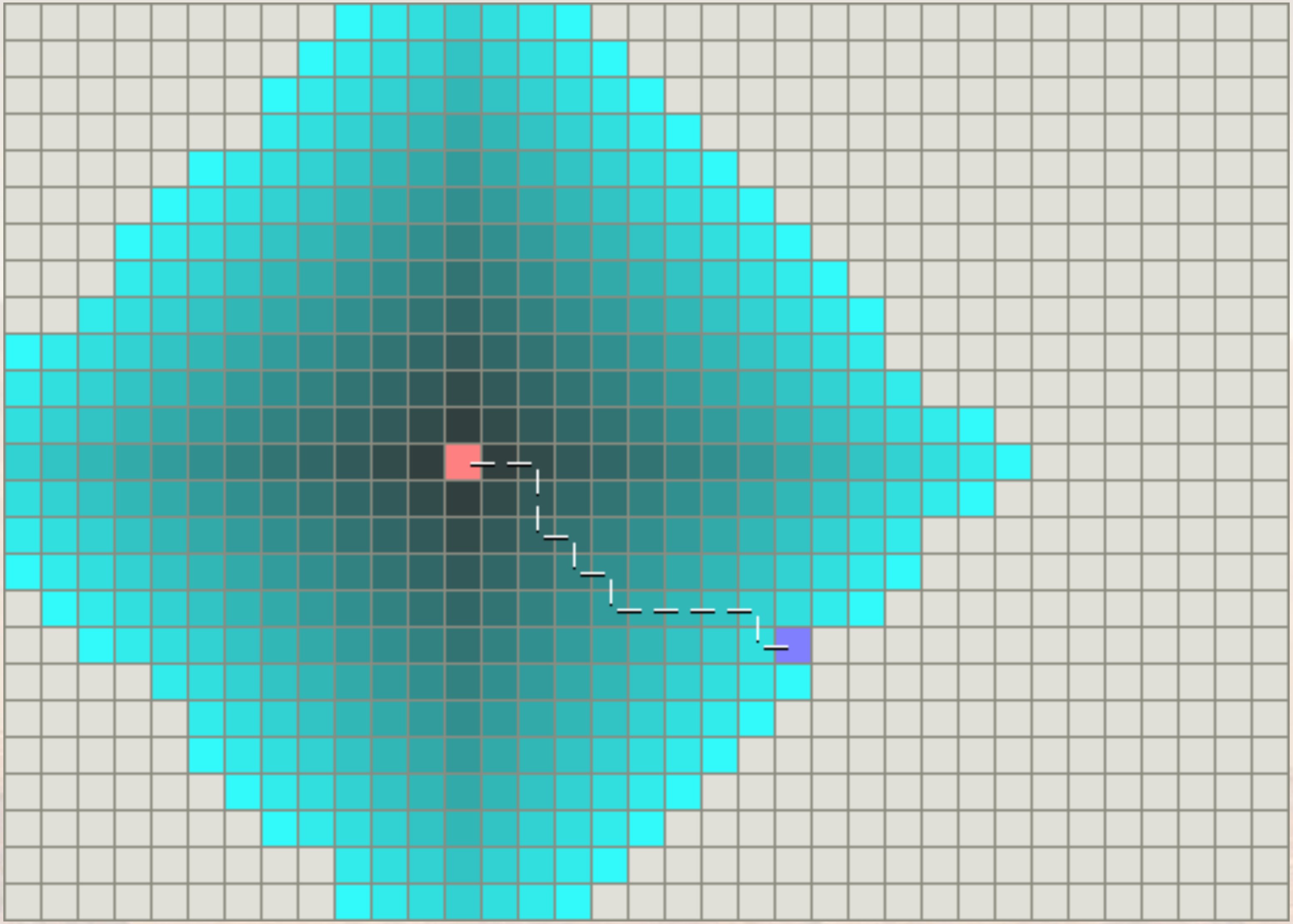
Convert (Planar) Space into Grid



Dijkstra's Algorithm

- Examine closest vertex not yet examined
- Add new cell's vertices to vertices not yet examined
- Expand outward from starting point until you reach the goal cell
- Guaranteed to find a shortest path (could be multiple equally short paths existing)...
- ...as long as no path elements have negative cost

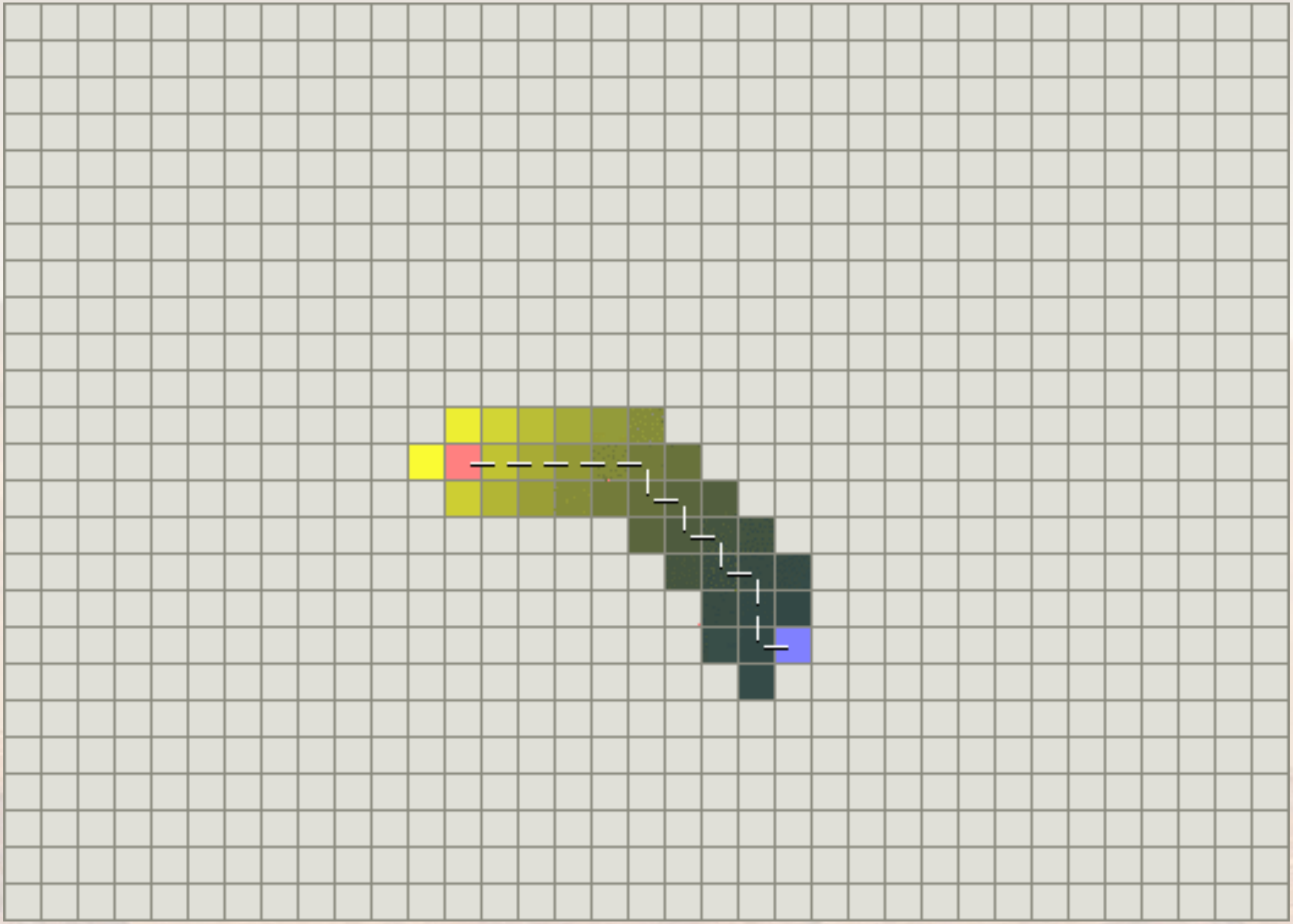
Dijkstra's Algorithm



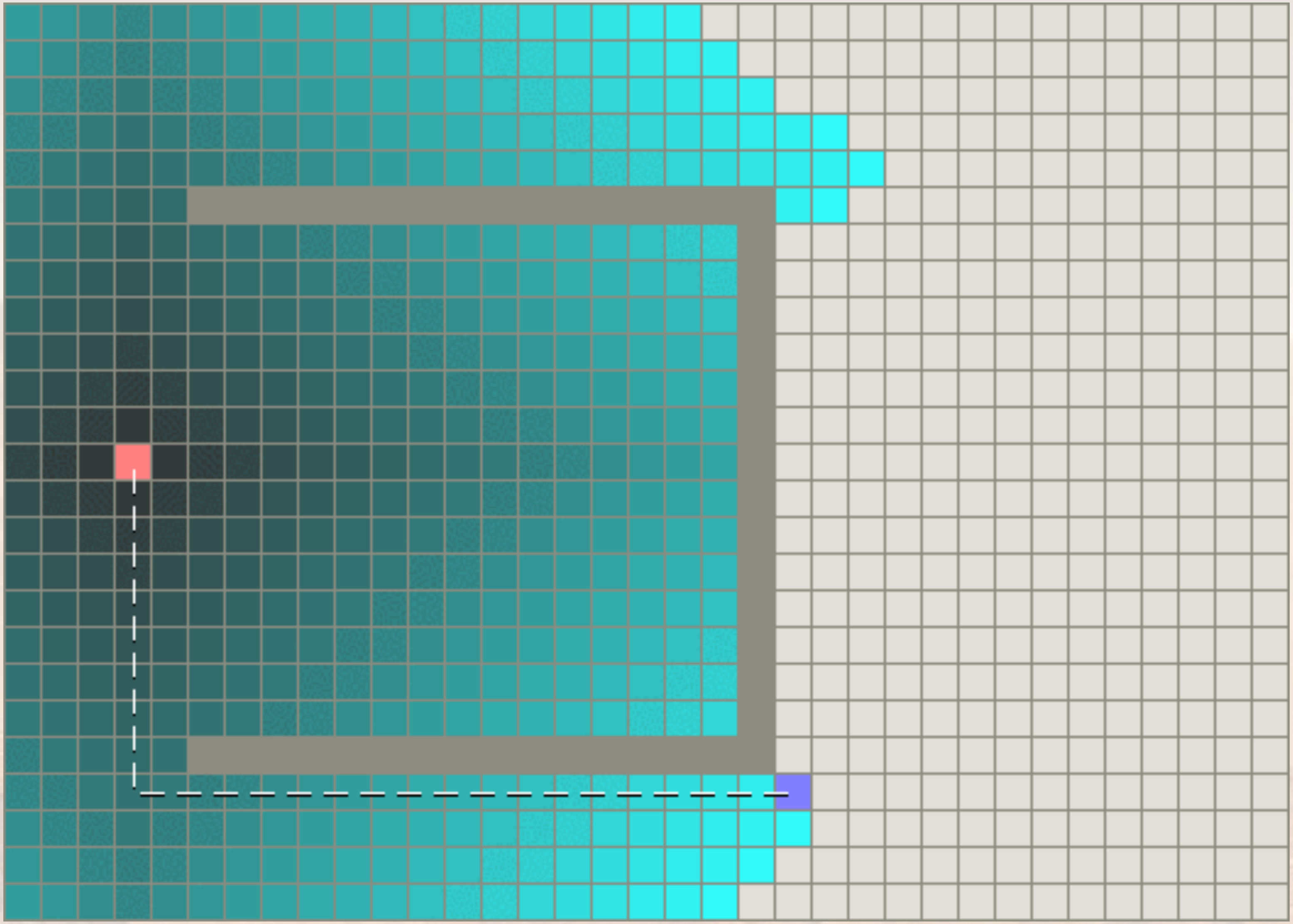
Greedy Best-First-Search Algorithm

- Assumes you have an estimate (“heuristic”) of how far any given element is from goal
- Continue to scan closest adjacent vertices to find closest estimated distance from goal
- Is not guaranteed to find a shortest path, but is faster than Dijkstra’s method

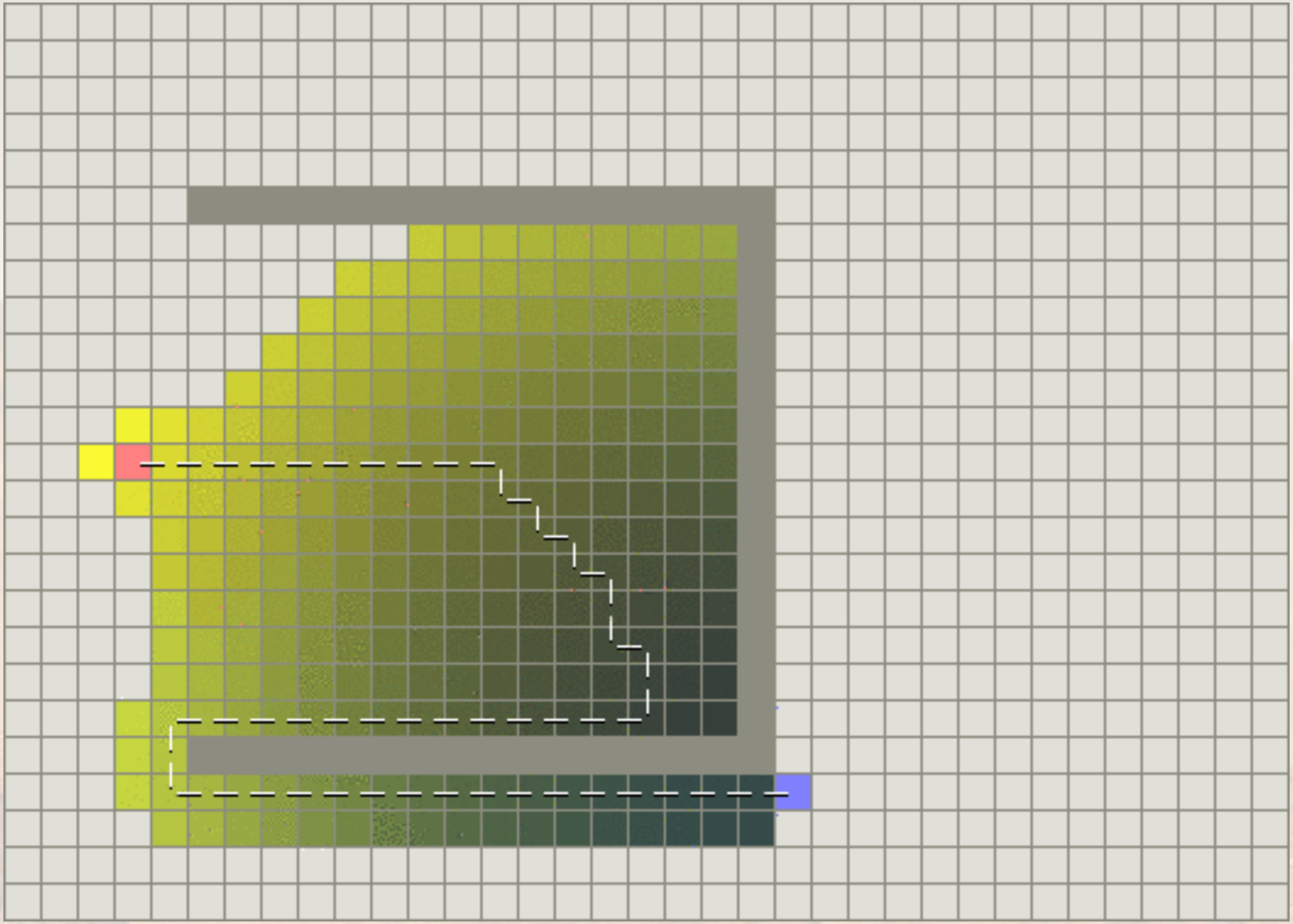
Greedy Best-First-Search Algorithm



Dijkstra's Method with Concave Obstacle



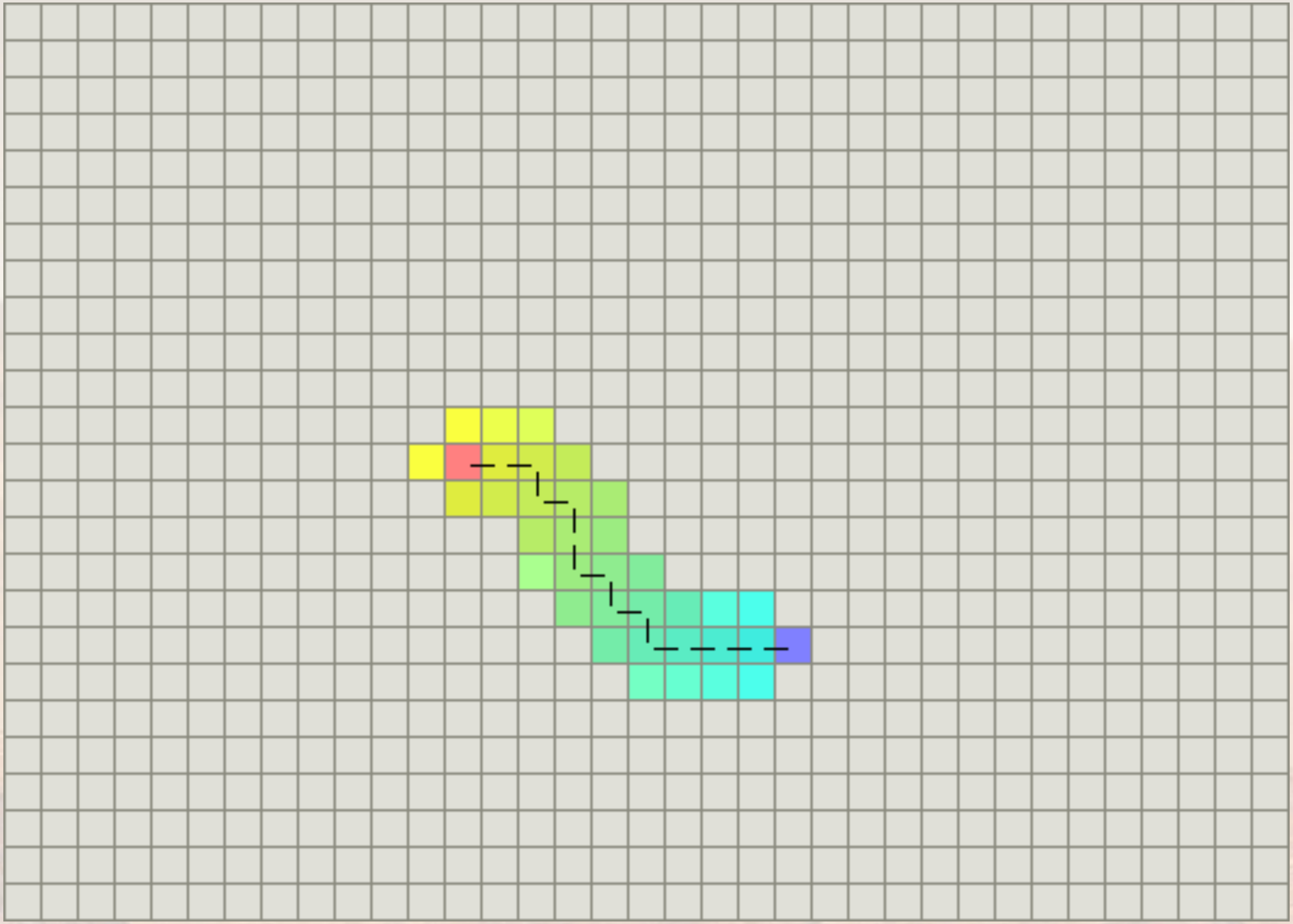
Best-First Search with Concave Obstacle



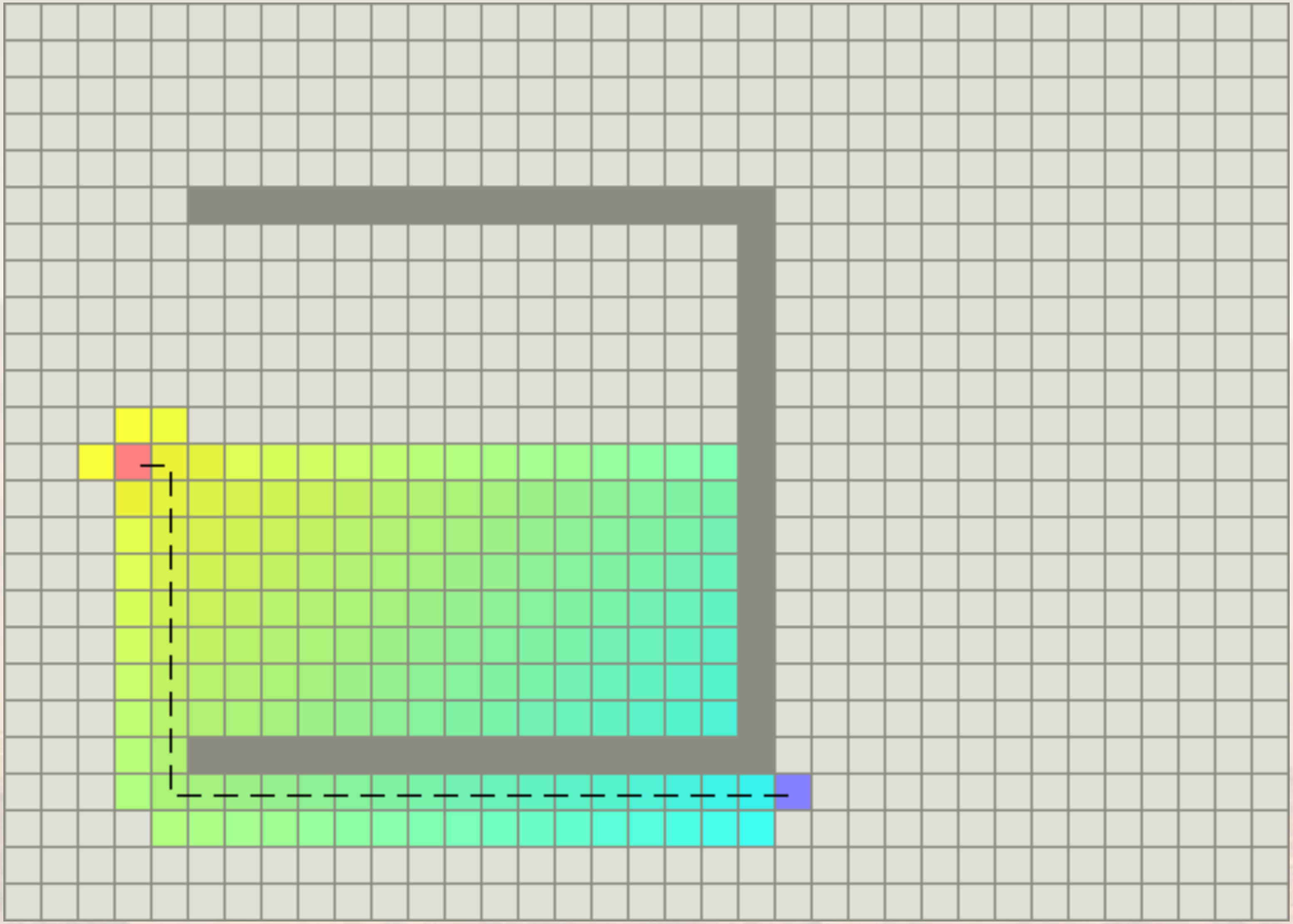
Comments on Concave Obstacle

- Dijkstra's method still produces shortest path, but a large area of the grid has to be searched
- Best-First method is quicker, but produces more inefficient path ("greedy" algorithm drives to goal even in presence of surrounding obstacle)
- Ideal approach would be to combine formal comprehensive (Dijkstra) and heuristic (Best-First) approaches
- A* - uses heuristic approach to finding path to goal while guaranteeing that it's a shortest path

Unconstrained A* Path Solution



A* Solution with Concave Obstacle



Implementation of A*

- g(n) is cost of the path from the starting point to any examined point on map
- h(n) is heuristic distance estimate from point on map to goal point
- Each loop searches for vertex (n) that minimizes $f(n)=g(n)+h(n)$

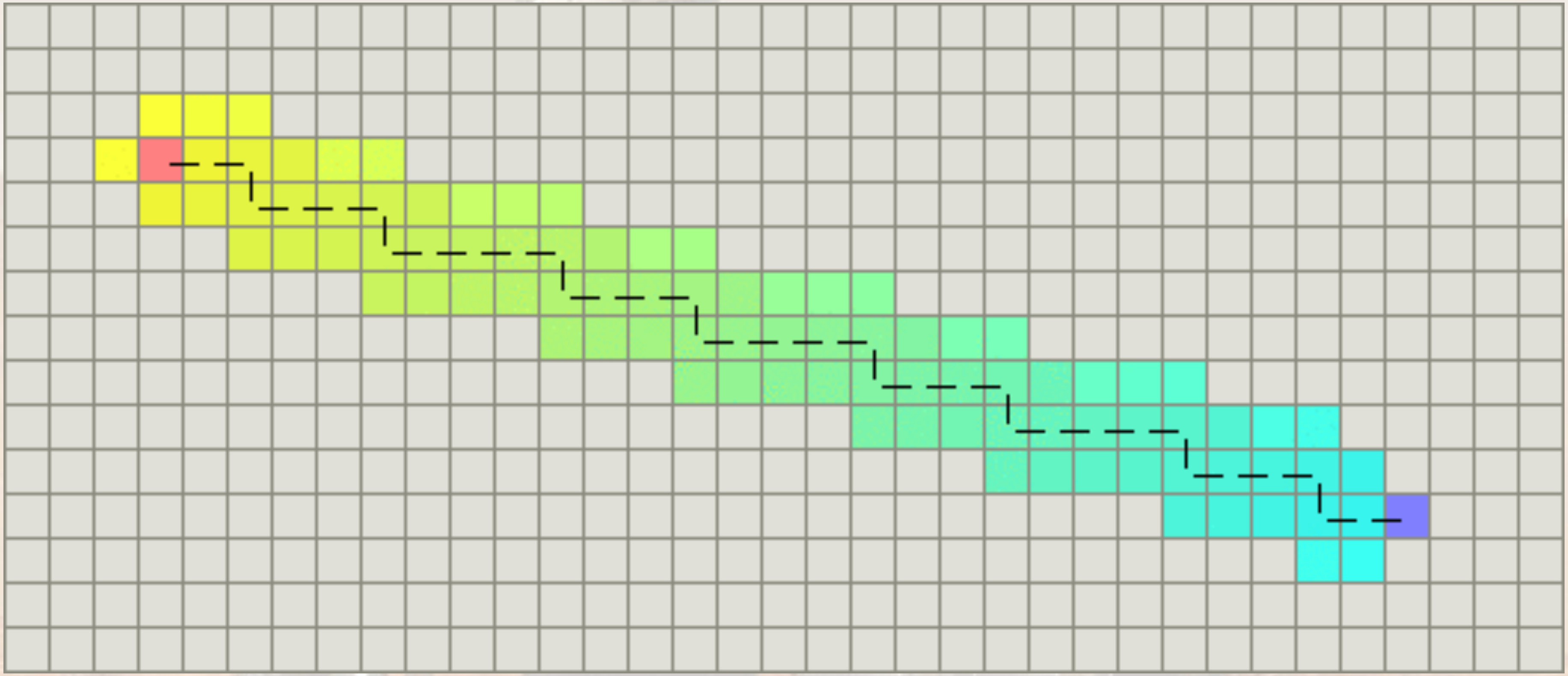
Effect of Heuristic Accuracy

- If h(n)=0, only g(n) is present and A* turns into Dijkstra's method, which is guaranteed to find a minimum
- If h(n) is smaller than actual distance ("admissible"), still guaranteed to find minimum, but the smaller h(n) is, the larger the search space and slower the search
- If h(n) is exact, get an exact answer that goes directly to the goal
- If h(n) is greater than real distance, no longer guaranteed to produce shortest path, but it runs faster
- If g(n)=0, only dependent on h(n) and turns into Best-First heuristic algorithm

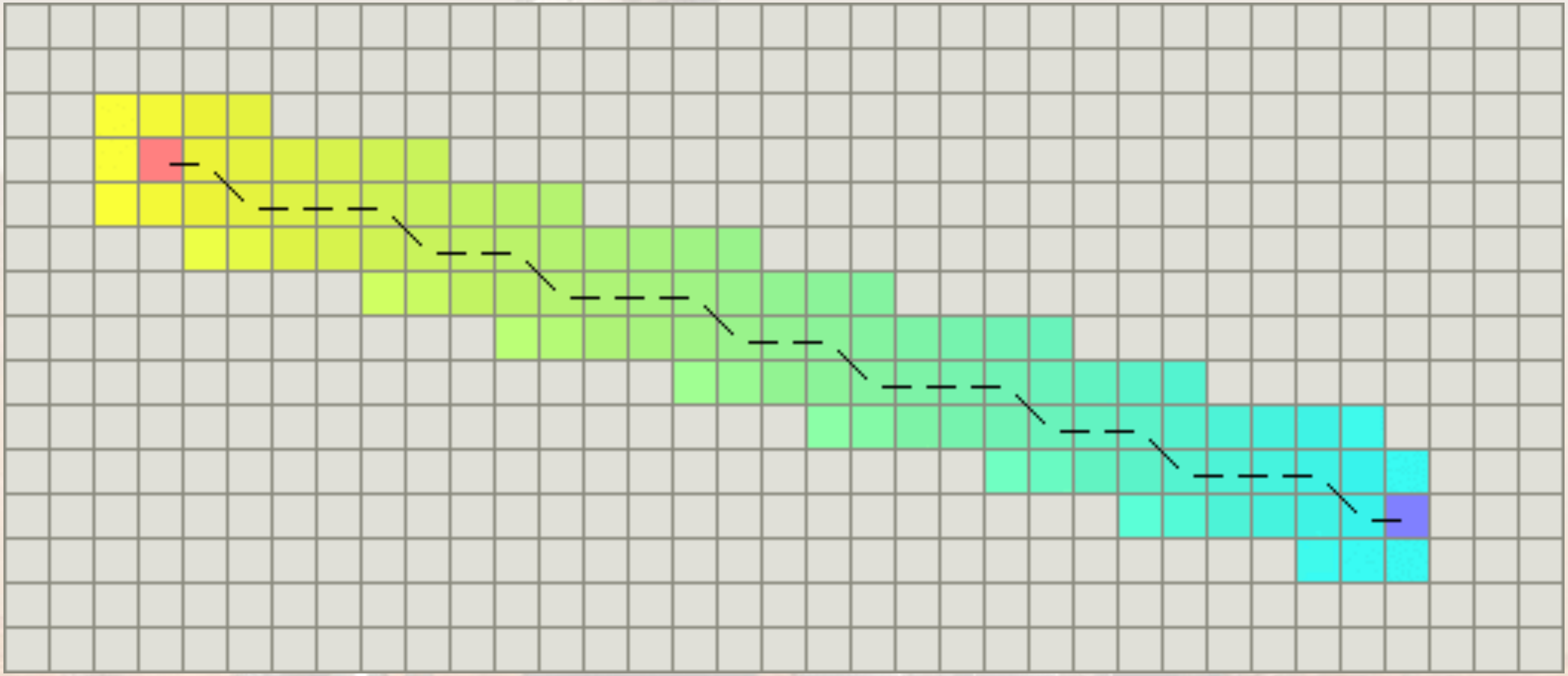
Insights into A* Path Planning

- You don't need a heuristic that's exact - you just need something that's close
- Non-admissible heuristics (h(n)>exact value) don't guarantee shortest path but do speed up solutions
- "Cost" of movement can be whatever metric you're most concerned about - e.g., slope or soil
- If flat area has movement cost of 1 and slopes have movement cost of 3, search will propagate three times as fast in flat land as in hilly areas
- g(n) and h(n) need to have the same units

Heuristic Estimation - Manhattan Distance



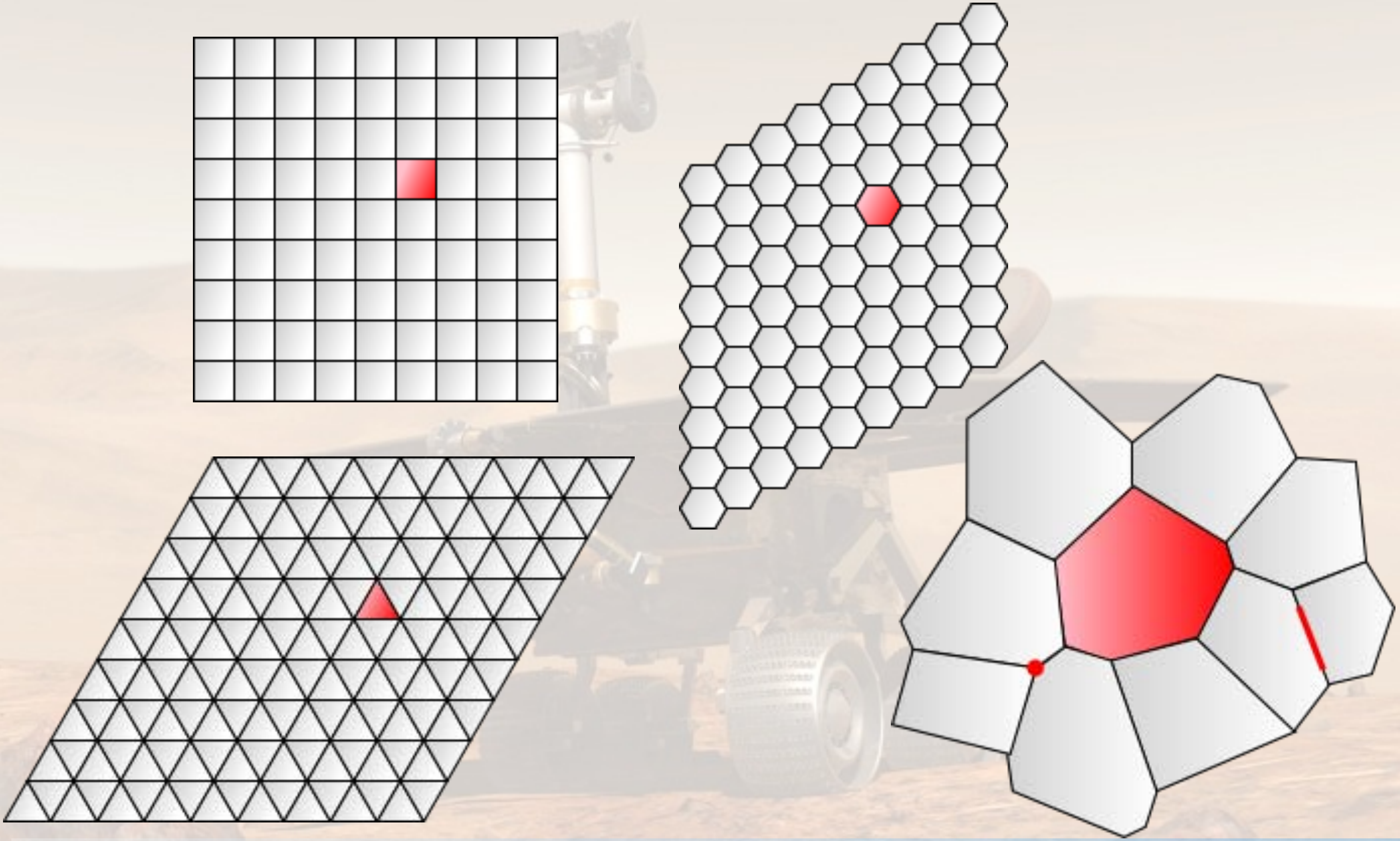
Heuristic Estimation - Chebyshev Distance



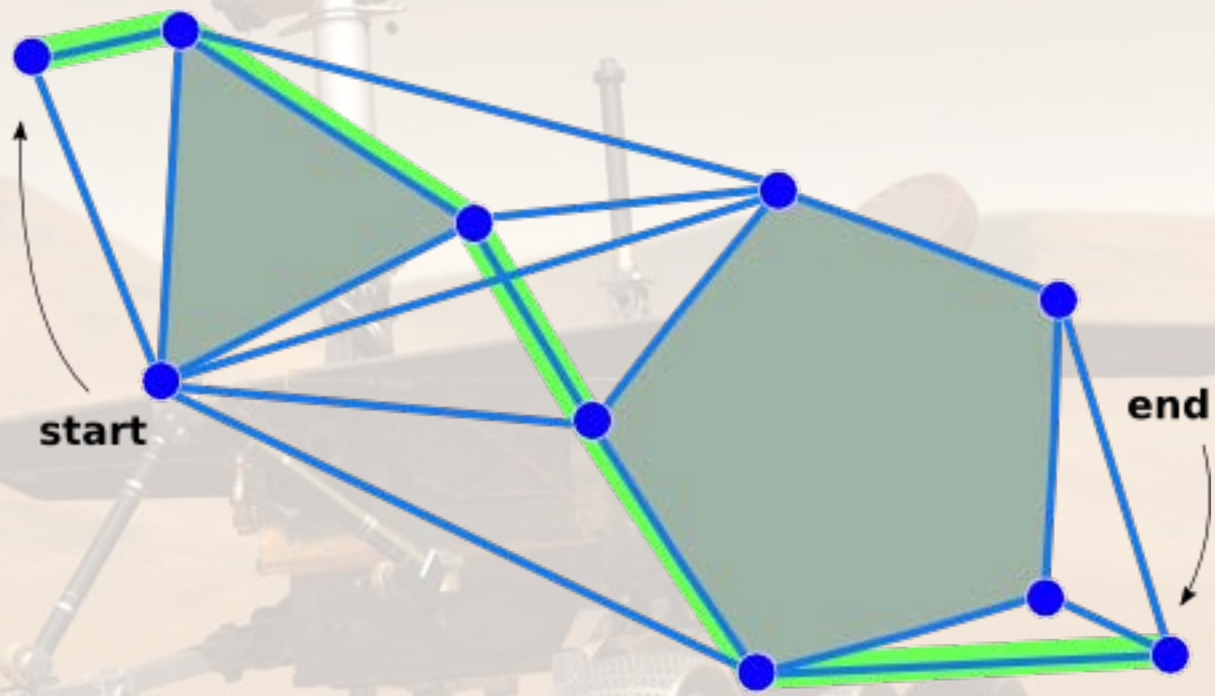
A* Variations

- Dynamic A* (“D*”)
  - A* works if you have perfect knowledge
  - D* allows for correcting knowledge errors efficiently
- Lifelong Planning A* (“LPA*”)
  - Useful when travel costs are changing
- Both approaches allow reuse of A* data, but require storage of all A* parameters
  - Storage requirements become prohibitive when moving obstacles are present

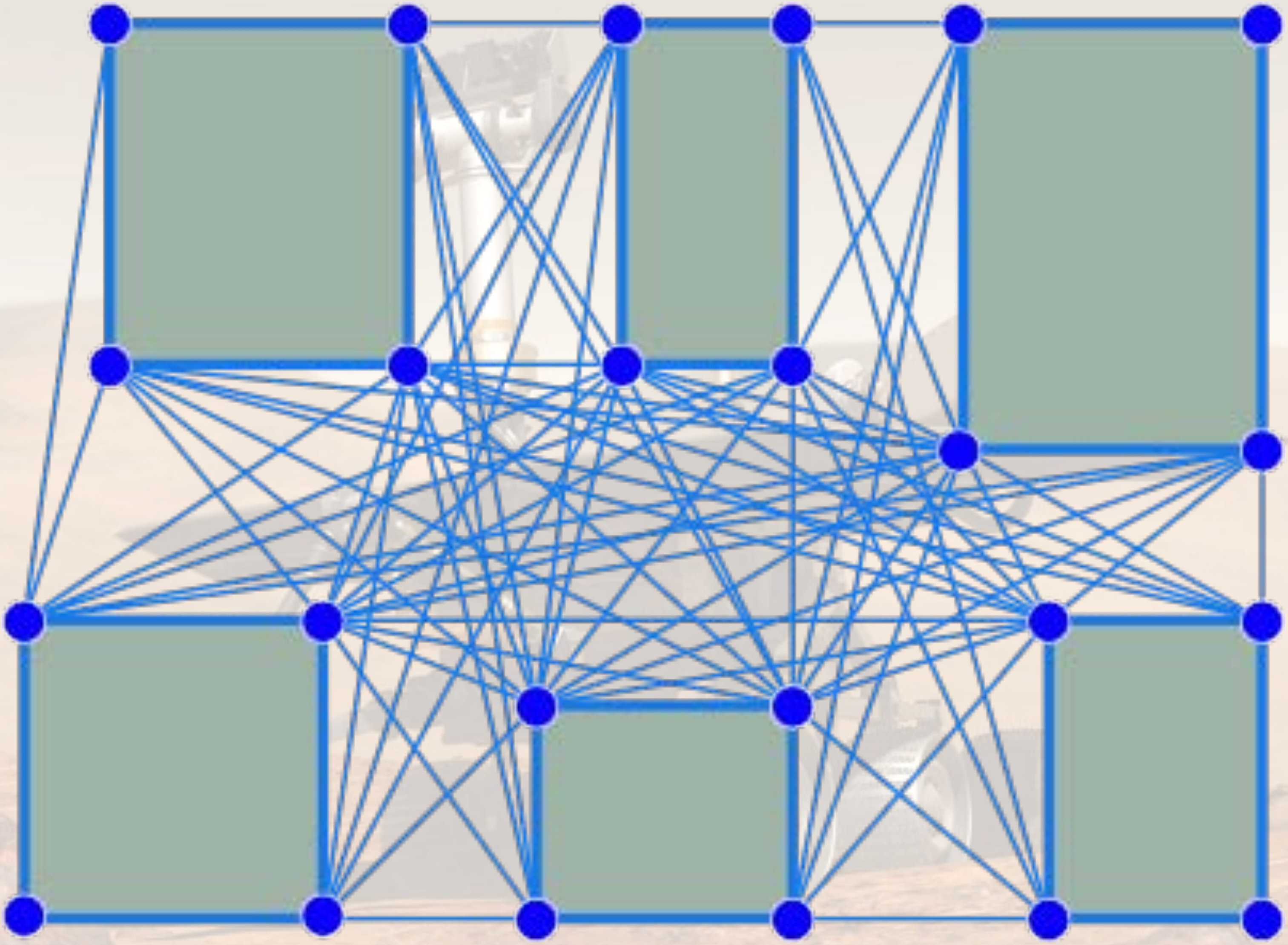
Grid Representations



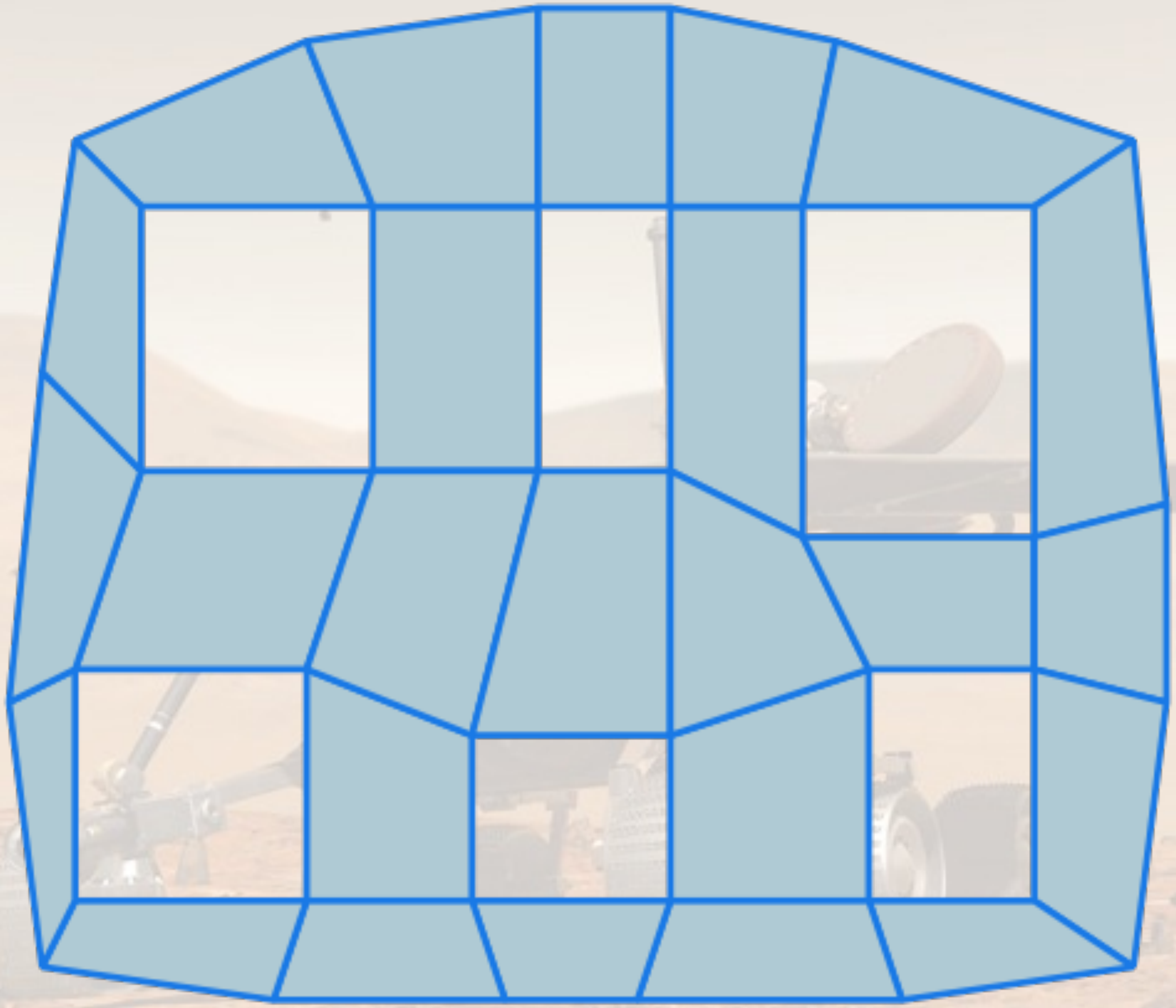
Polygonal Map Representations



Full Path Specification



Simplified Mesh Representation



Acknowledgments

Most of the material relating to path planning comes from Amit Patel from the Standford Computer Science department:

theory.stanford.edu/~amitp/GameProgramming/

Mapping

- Why do we map?
- Spatial decomposition
- Representing the robot
- Current challenges

Mapping

- Represent the environment around the robot
- Impacted by the robot position representation
- Relationships
  - Map precision must match application
  - Precision of features on map must match precision of robot's data (and hence sensor output)
  - Map complexity directly affects computational complexity and reasoning about localization and navigation
- Two basic approaches
  - Continuous
  - Decomposition (discretization)

Environment Representation

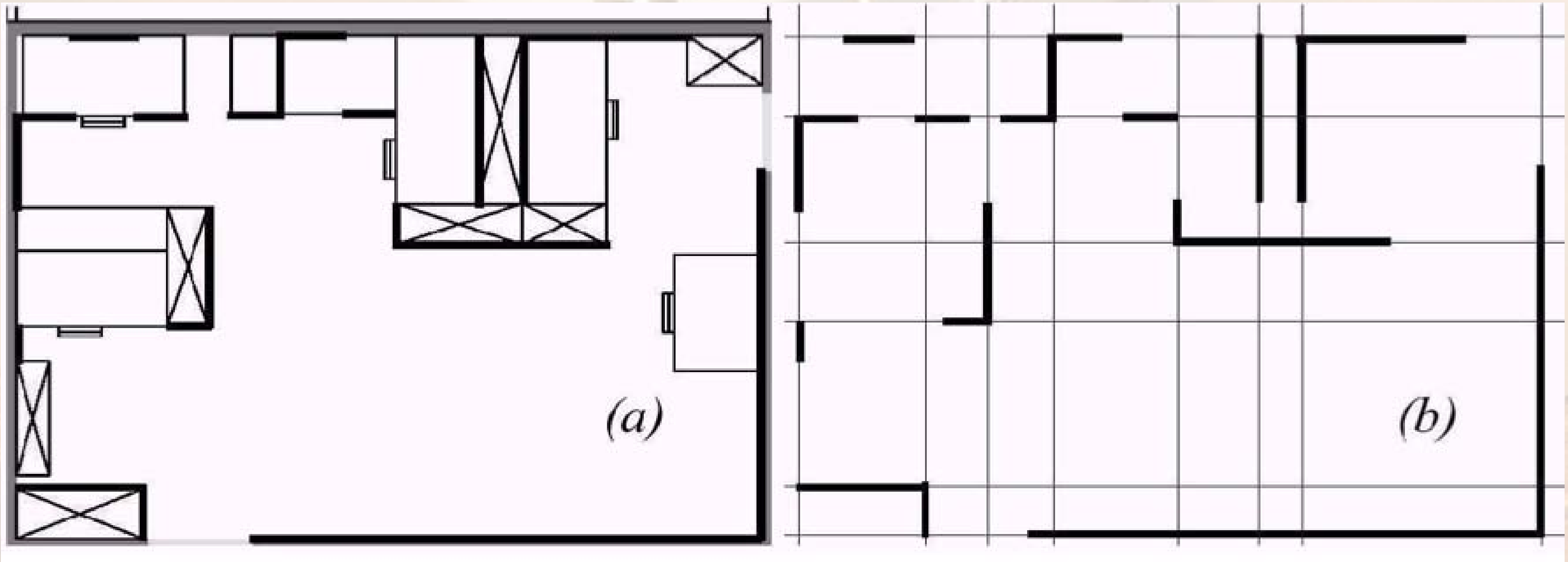
- Continuous metric - x, y, theta
- Discrete metric - metric grid
- Discrete topological - topological grid
- Environmental modeling
  - Raw sensor data - large volume, uses all acquired info
  - Low level features (e.g., lines, etc.) - medium volume, filters out useful info, still some ambiguities
  - High level features (e.g., doors, car) - low volume, few ambiguities, not necessarily enough information

Continuous Representation

- Exact decomposition of environment
- Closed-world assumption
  - Map models all objects
  - Any area of map without objects has no objects in corresponding environment
  - Map storage proportional to density of objects in environment
- Map abstraction and selective capture of features to ease computational burden

Continuous Representation

- Match map type with sensing device
  - e.g., for laser range finder, may represent map as a series of infinite lines
  - Fairly easy to fit laser range data to series of lines



Continuous Representation

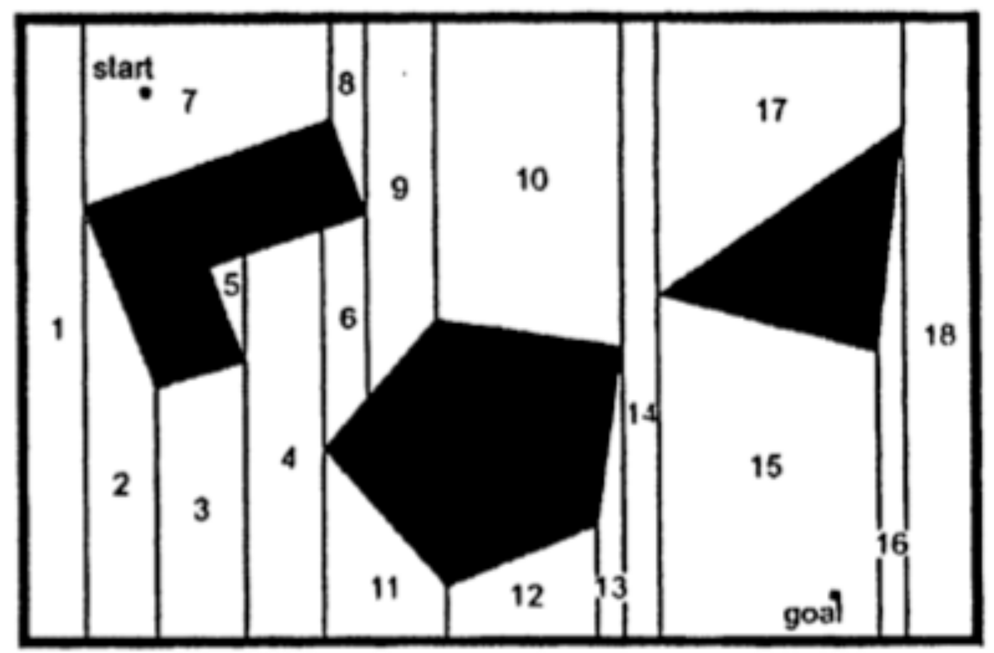
- In conjunction with position representation
  - Single hypothesis: extremely high accuracy possible
  - Multiple hypothesis: either
    - Depict as geometric shape
    - Depict as discrete set of possible positions
- Benefits of continuous representation
  - High accuracy possible
- Drawbacks
  - Can be computationally intensive
  - Typically only 2D

Decomposition

- Capture only the useful features of the world
- Computationally better for reasoning, particularly if the map is hierarchical

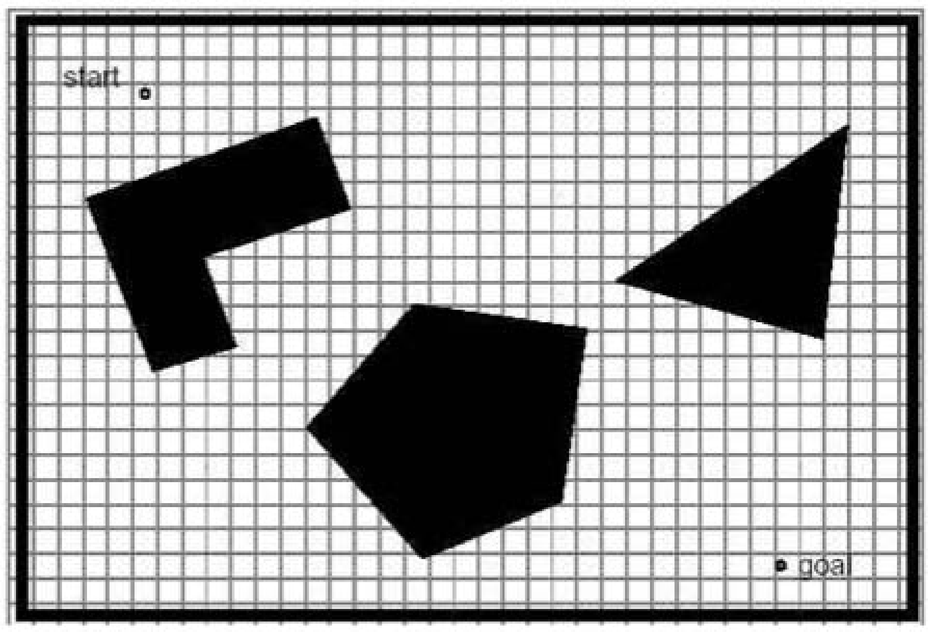
Exact Cell Decomposition

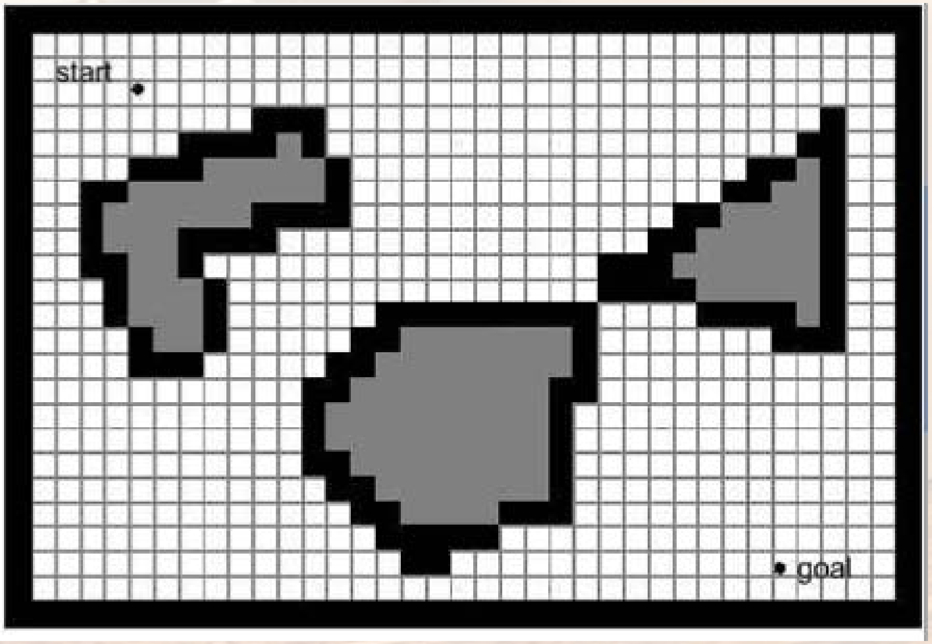
- Model empty areas with geometric shapes
- Can be extremely compact (18 nodes here)
- Assumption: robot position within each area of free space does not matter



Fixed Cell Decomposition

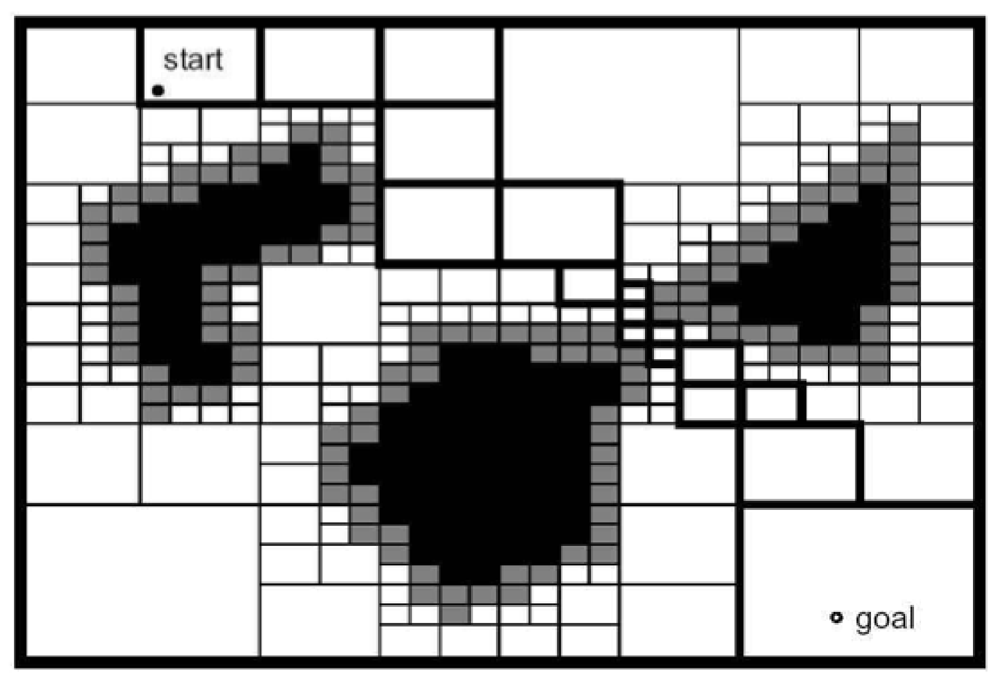
- Tesselate world - discrete approximation
- Each cell is either empty or full
- Inexact (note loss of passageway on right)





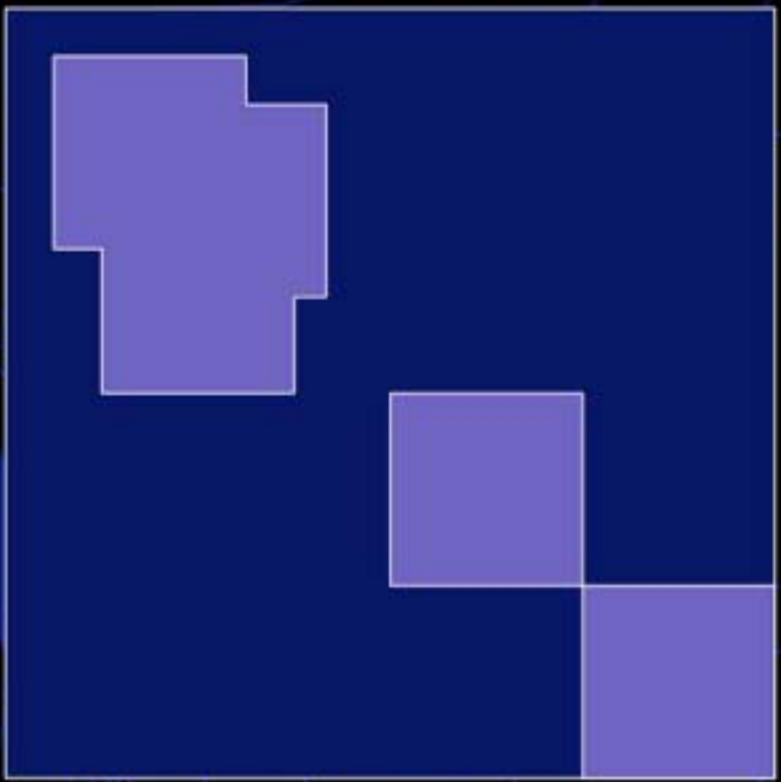
Adaptive Cell Decomposition

- Multiple types of adaptation: quadtree, BSP, etc.
- Recursively decompose until a cell is completely free or full
- Very space efficient compared to fixed cell



Quadtree Example

Space Representation

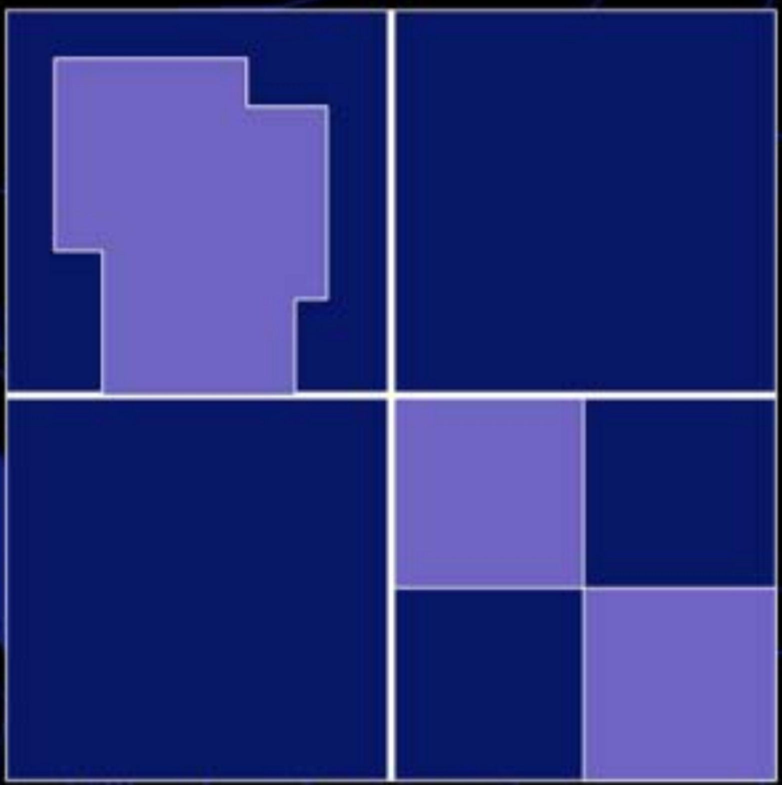


Equivalent quadtree

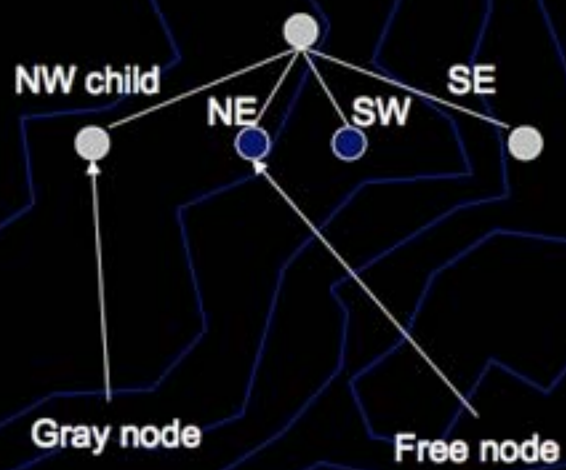
Russell Gayle, The University of North Carolina, Chapel Hill

Quadtree Example

Space Representation

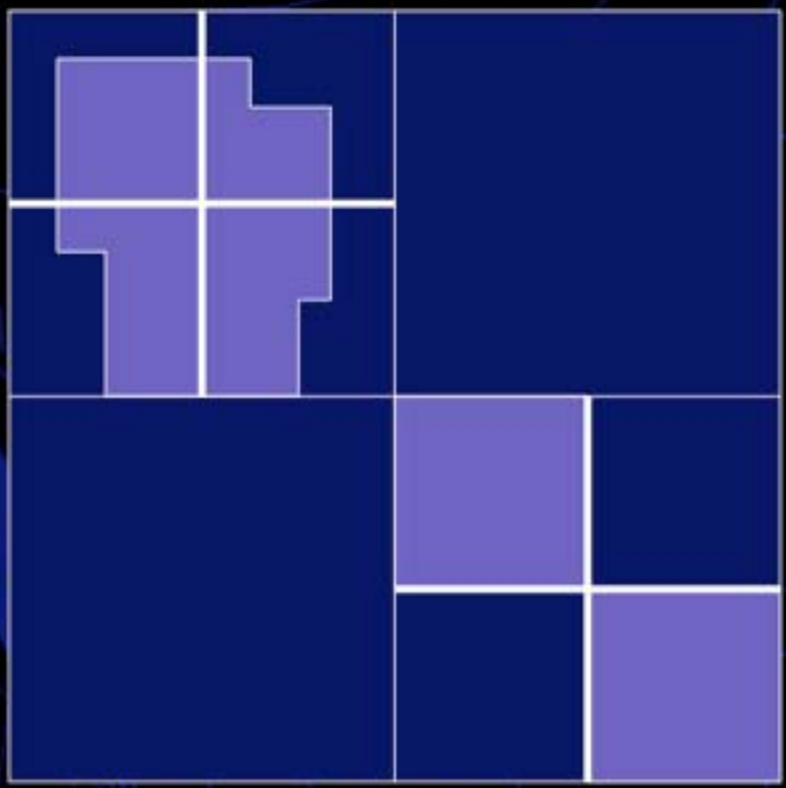


Equivalent quadtree

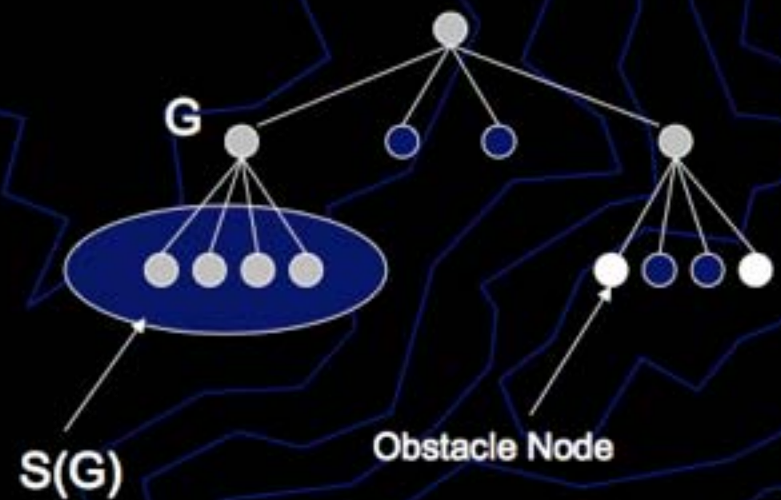


Quadtree Example

Space Representation

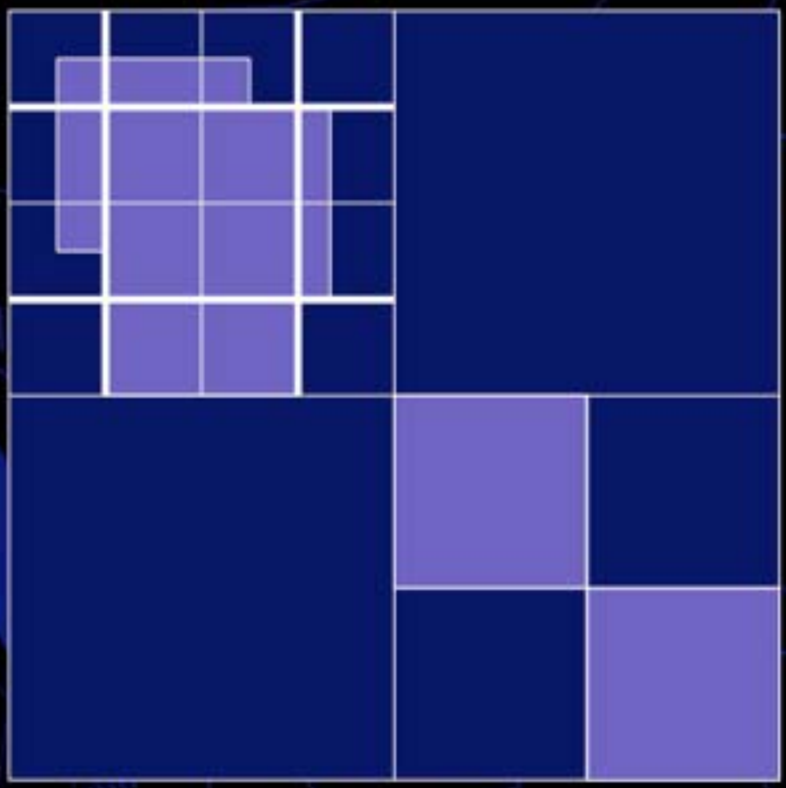


Equivalent quadtree

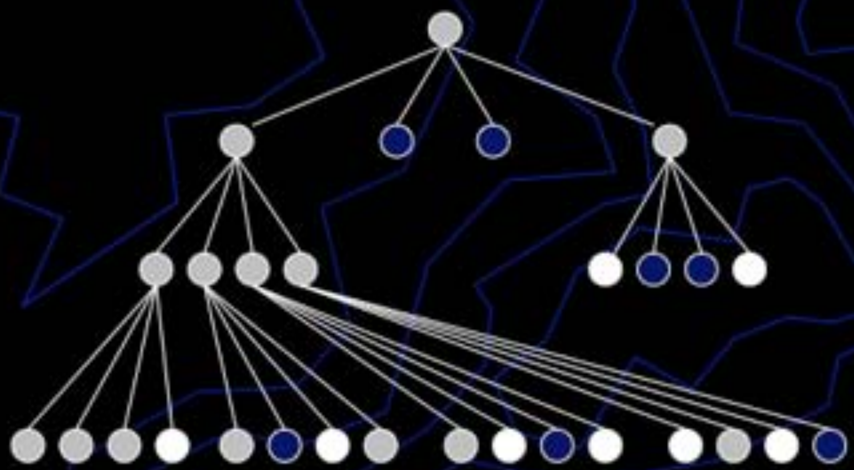


Quadtree Example

Space Representation



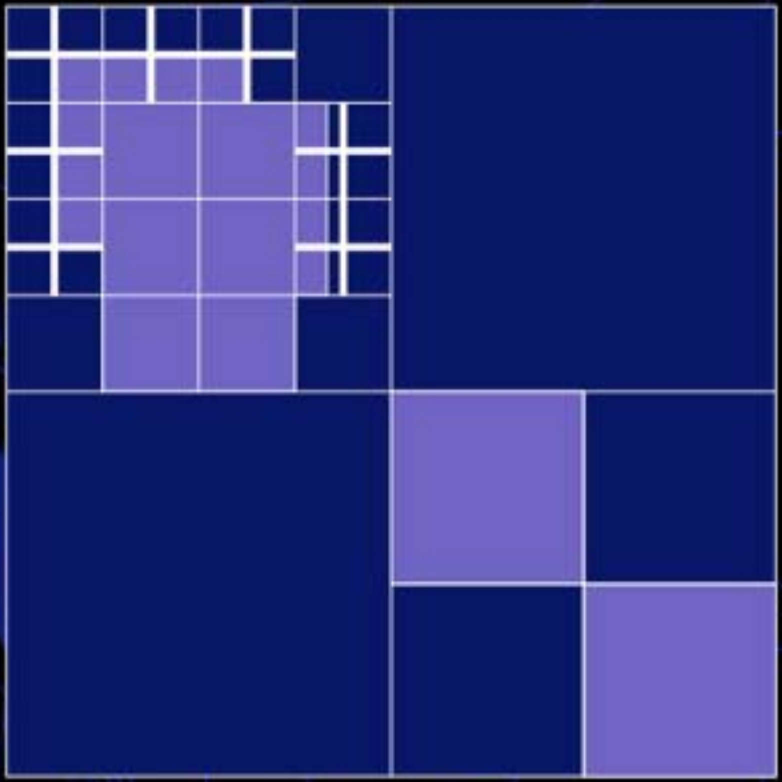
Equivalent quadtree



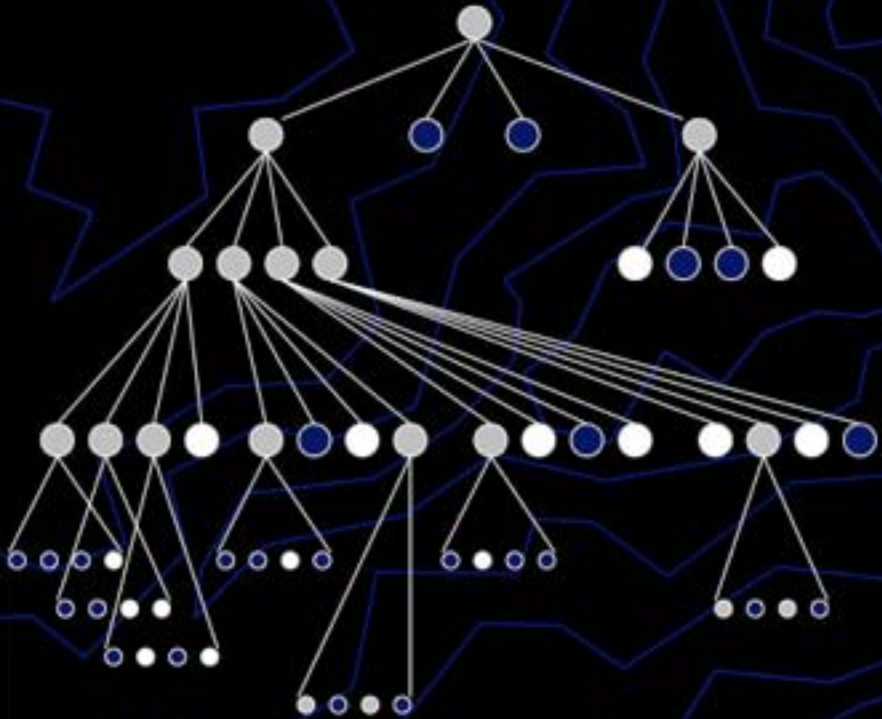
Each of these steps are examples of pruned quadtrees, or the space at different resolutions

Quadtree Example

Space Representation

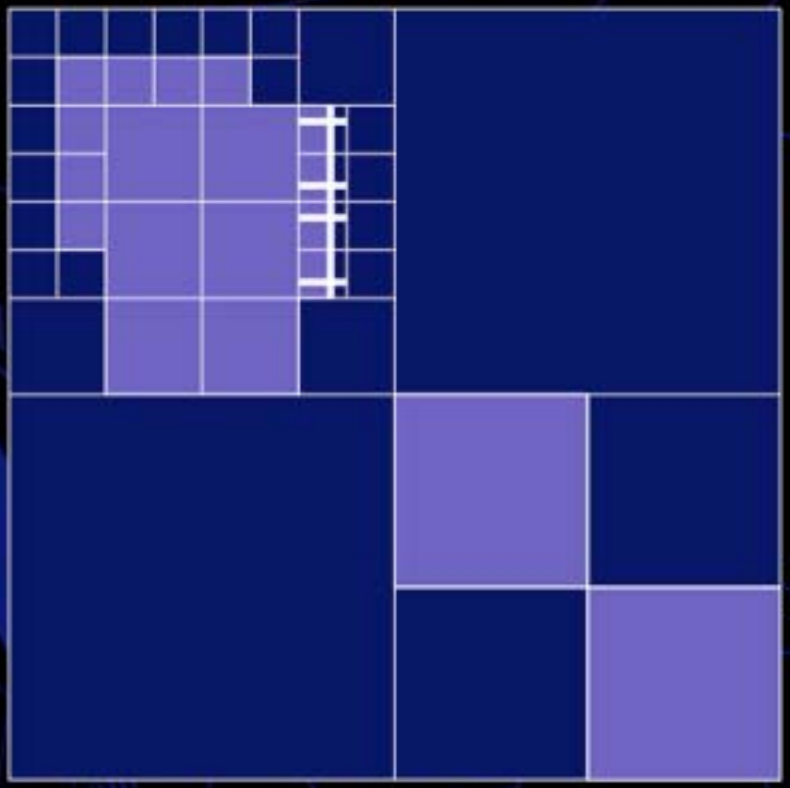


Equivalent quadtree

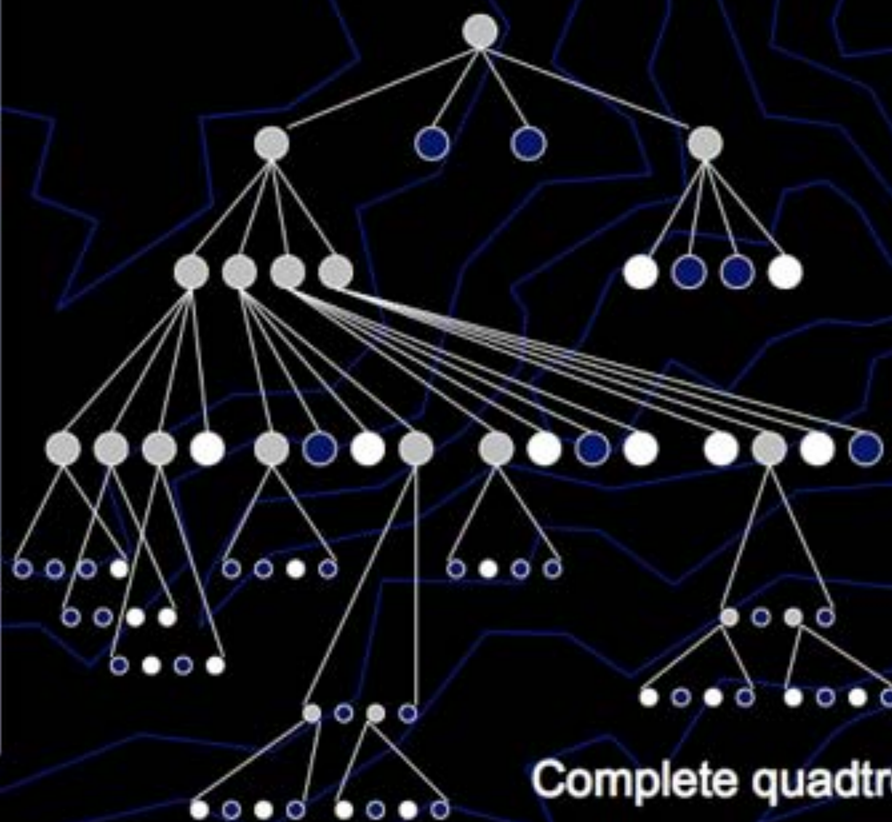


Quadtree Example

Space Representation



Equivalent quadtree



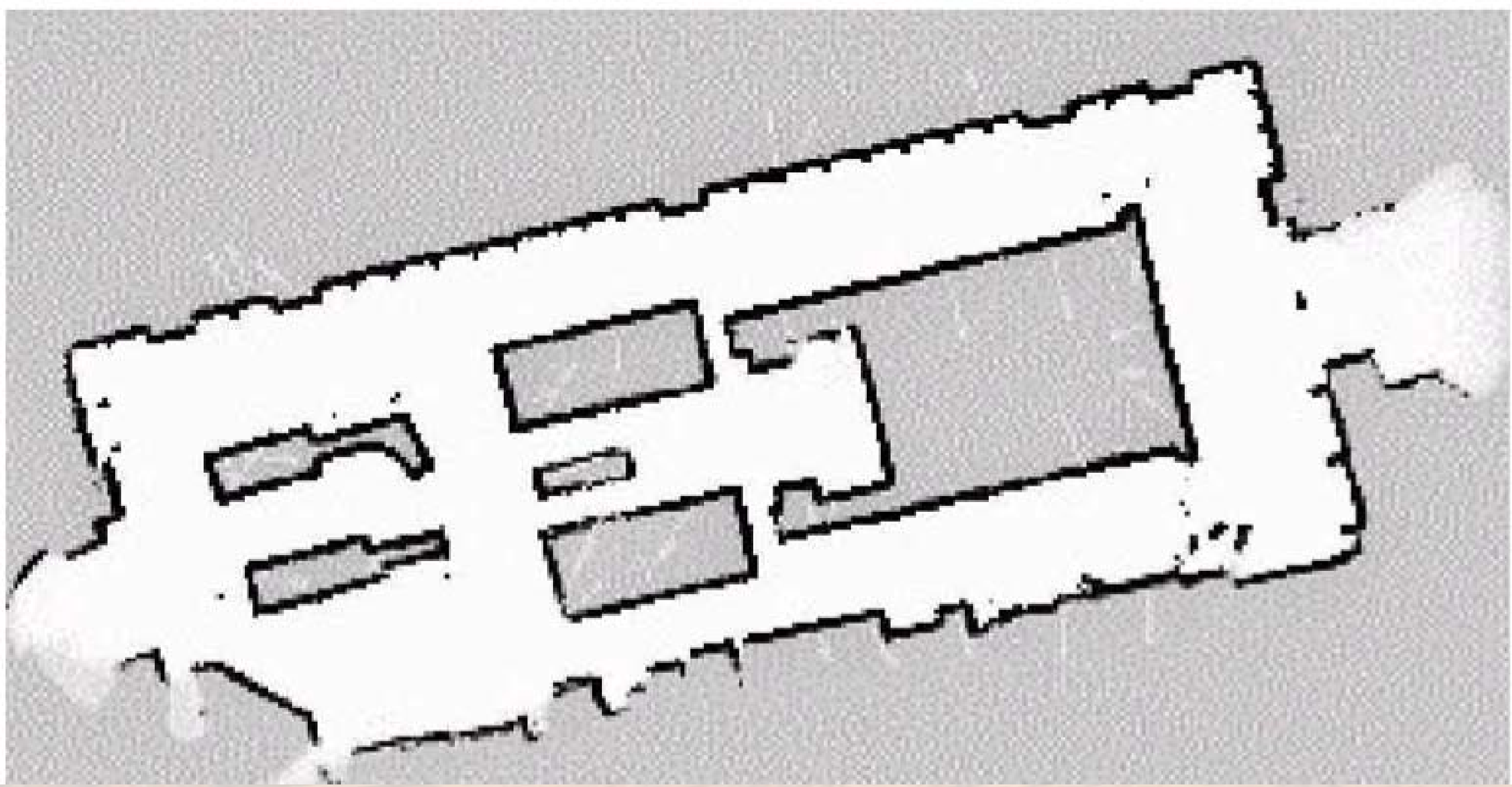
Complete quadtree

Occupancy Grid

- Typically fixed decomposition
- Each cell is either filled or free (set threshold for determining "filled")
- Particularly useful with range sensors
  - If sensor strikes something in cell, increment cell counter
  - If sensor strikes something beyond cell, decrement cell counter
  - By discounting cell values with time, can deal with moving obstacles
- Disadvantages
  - Map size a function of sizes of environment and cell
  - Imposes a priori geometric grid on world

Occupancy Grid

Darkness of cell proportional to counter value

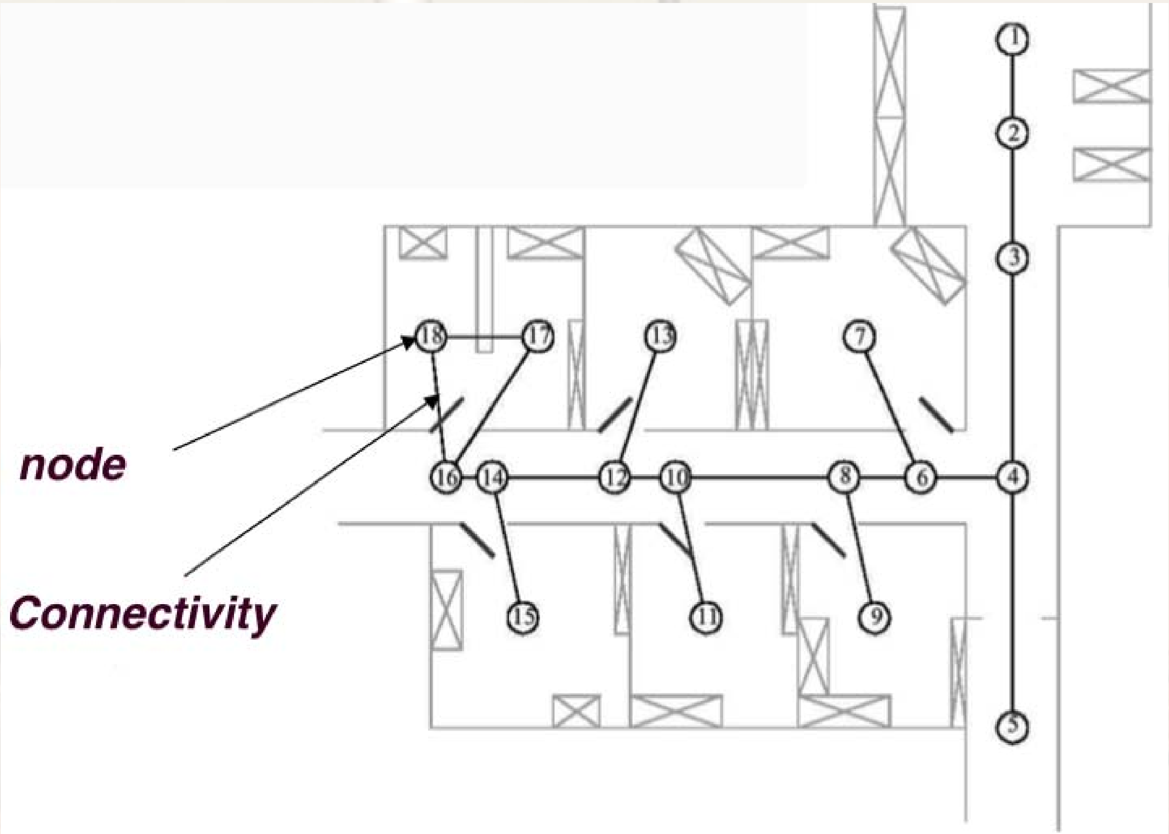


Topological Decomposition

- Use environment features most useful to robots
- Generates a graph specifying nodes and connectivity between them
  - Nodes not of fixed size; do not specify free space
  - Node is an area the robot can recognize its entry to and exit from

Topological Example

For this example, the robot must be able to detect intersections between halls, and between halls and rooms



Topological Decomposition

- To robustly navigate with a topological map a robot
  - Must be able to localize relative to nodes
  - Must be able to travel between nodes
- These constraints require the robot's sensors to be tuned to the particular topological decomposition
- Major advantage is ability to model non-geometric features (like artificial landmarks) that benefit localization

Map Updates: Occupancy Grids

- Occupancy grid
  - Each cell indicated probability of free space/occupied
  - Need method to update cell probabilities given sensor readings at time t
- Update methods
  - Sensor model
  - Bayesian
  - Dempster-Shafer

Representing the Robot

- How does the robot represent itself on the map?
- Point-robot assumption
  - Represent the robot as a point
  - Assume it is capable of omnidirectional motion
- Robot in reality is of nonzero size
  - Dilation of obstacles by robot's radius
  - Resulting objects are approximations
  - Leads to problems with obstacle avoidance

Current Challenges

- Real world is dynamic
- Perception is still very error-prone
  - Hard to extract useful information
  - Occlusion
- Traversal of open space
- How to build up topology
- This was all two-dimensional!
- Sensor fusion

Acknowledgements

- Thanks to Steven Roderick for originally developing this lecture
- “Introduction to Autonomous Mobile Robots” Siegwart and Nourbaksh
- “Mobile Robotics: A Practical Introduction” Nehmzow
- “Computational Principles of Mobile Robotics” Dudek and Jenkin
- “Introduction to AI Robotics” Murphy