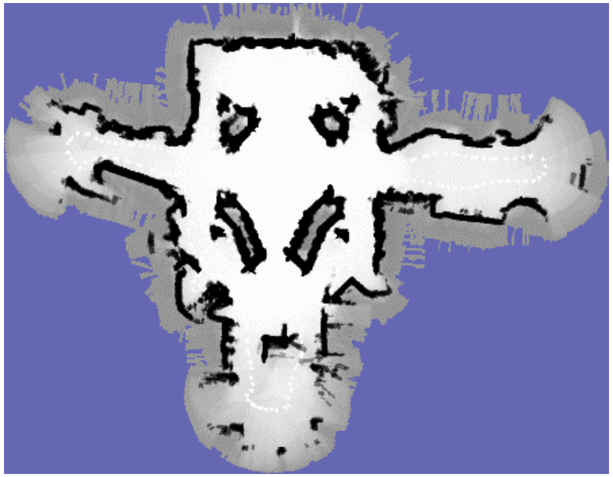
Navigation and Metric Path Planning

November 7, 2002

Class Meeting 22

Minerva tour guide robot (CMU):
Gave tours in Smithsonian's National Museum of History



Example of Minerva's occupancy map used for navigation

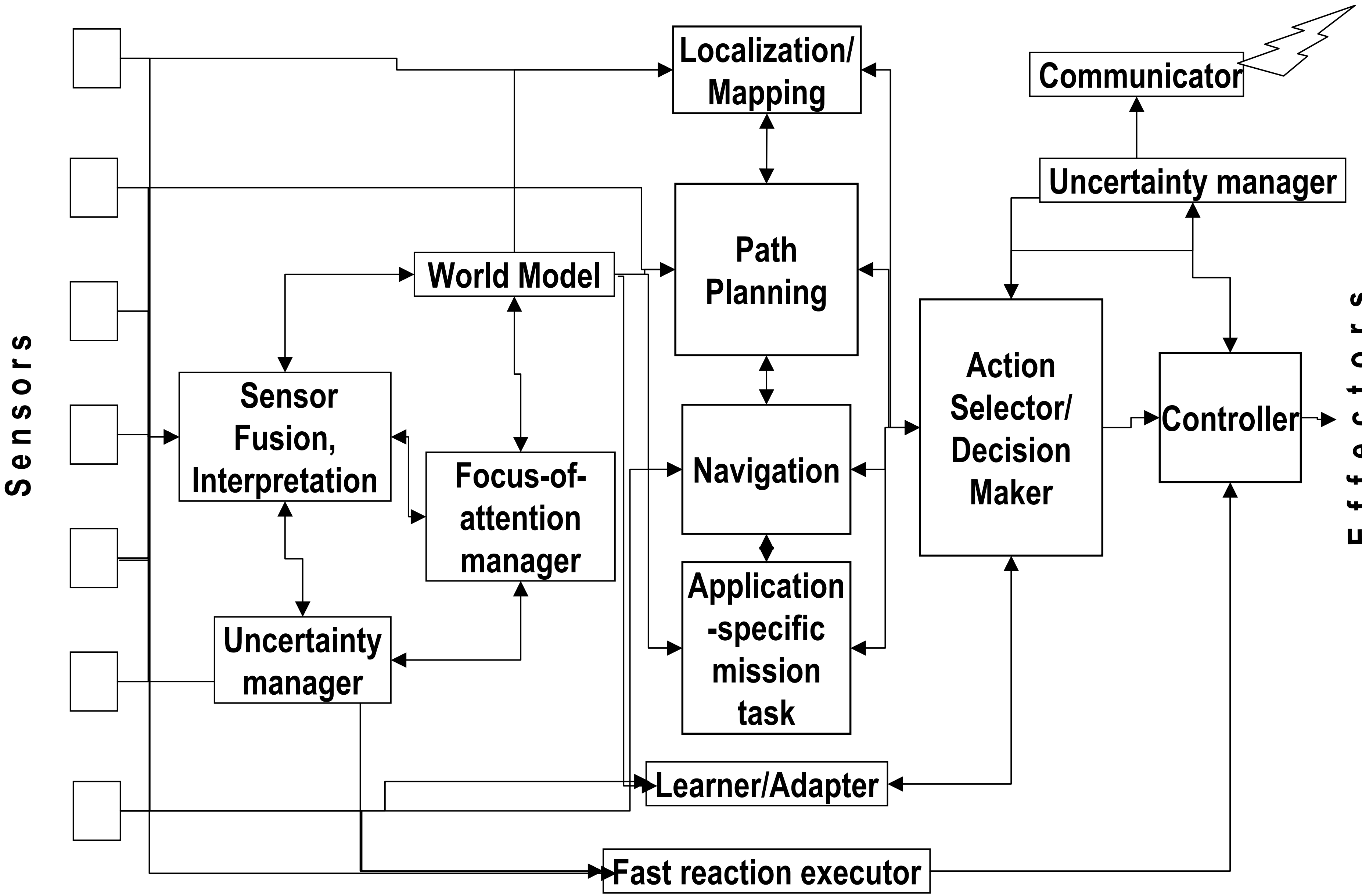
Announcements

- Remember:
  - Assignment #5: Due next class meeting (Tuesday, November 12)
- Extra credit assignment:
  - To be handed out next Tuesday, due 1 week later
  - Will not involve programming

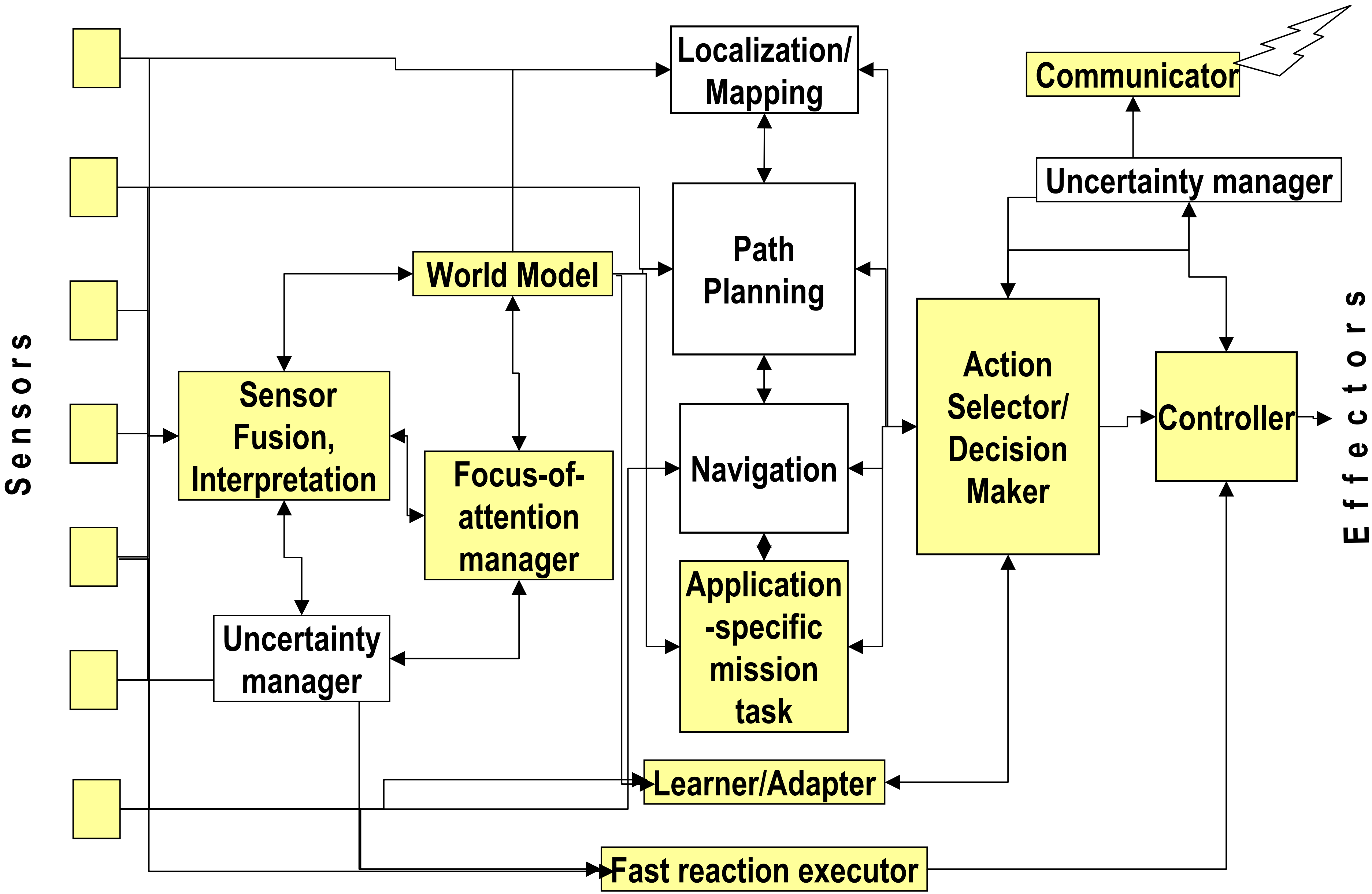
Objectives

- Understand techniques for metric path planning:
  - Configuration space
  - Meadow maps
  - Generalized Voronoi graphs
  - Grids
  - Quadtrees
  - Graph-based planners: A*
  - Wavefront-based planners

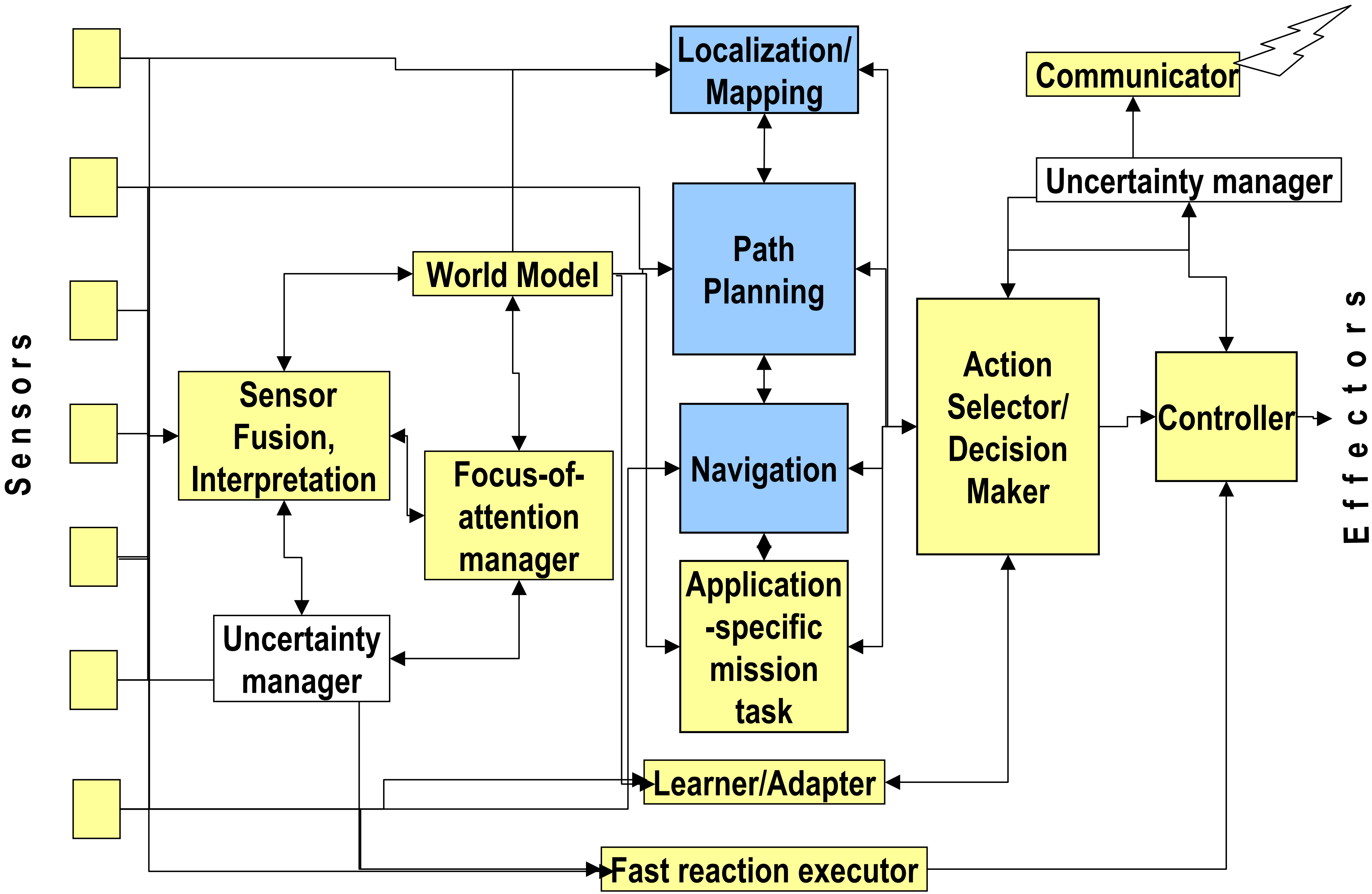
Remember Our Hypothetical Intelligent Mobile Robot (from Early in the Semester)



Our studies so far have investigated…



The rest of the semester we will investigate…



Introduction to Navigation

- Navigation is fundamental ability in autonomous mobile robotics

- Primary functions of navigation:
  - Where am I going?
    - Usually defined by human operator or mission planner
  - What's the best way to get there?
    - Path planning: qualitative and quantitative
  - Where have I been?
    - Map making
  - Where am I?
    - Localization: relative or absolute

Introduction to Navigation (con’t.)

- Navigation is a fundamental robotics problem because it involves almost everything about AI robotics:
  - Sensing
  - Acting
  - Planning
  - Architectures
  - Hardware
  - Computational efficiencies
  - Problem solving

Introduction to Navigation (con't.)

- Path Planning Research – goes back to 1970s
- Lots of approaches: proper choice depends upon ecological niche
- Criteria for Evaluating Path Planners:
  - Complexity
  - Sufficiently represents the terrain
  - Sufficiently represents the physical limitations of the robot platform
  - Compatible with the reactive layer
  - Supports corrections to the map and re-planning

Intro. (con't.): Impact of Sensor Uncertainty

- Early path planning research (in simulation) assumed:
  - Sensors give an accurate representation of world
  - Robot ability to localize

- But, as we've learned, these assumptions aren't true

- Therefore, robot has to operate in presence of uncertainty

- Result: new techniques for dealing with sensor noise in localization, map building, and path planning

Intro. (con't.): Spatial Memory

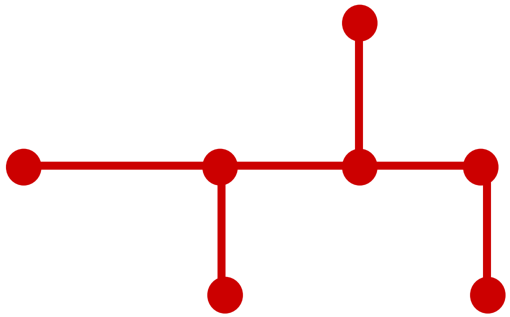
- Spatial memory:
  - World representation used by robot
  - Provides methods and data structures for processing and storing information derived from sensors
  - Organized to support methods that extract relevant expectations about a navigational task

Intro. (con't.): Spatial Memory (con't.)

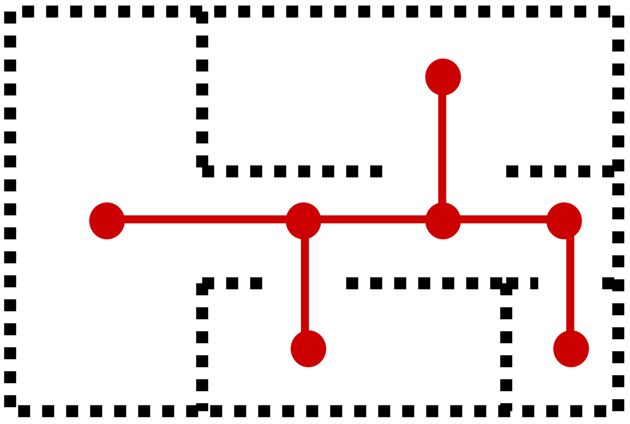
- Four basic functions of Spatial memory:
  - Attention: What features, landmarks to look for next?
  - Reasoning: E.g., can I fit through that door?
  - Path Planning: What is the best way through this building?
  - Information collection: What does this place look like? Have I ever seen it before? What has changed since I was here before?

Intro. (con't.): Spatial Memory (con't.)

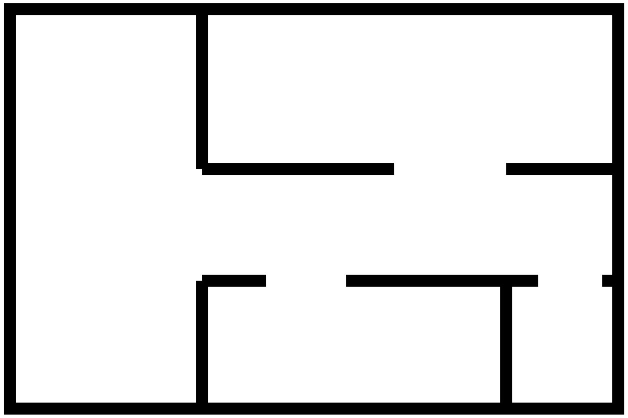
- Examples of two forms of Spatial memory:
  - Qualitative (route):



derived from:



  - Quantitative (metric or layout):



Intro. (con't.): Spatial Memory (con't.)

- Two forms of Spatial memory:
  - Qualitative (route):
    - Express space in terms of connections between landmarks
    - Dependent upon perspective of the robot
    - Orientation clues are egocentric
    - Usually cannot be used to generate quantitative (metric/layout) representations
  - Quantitative (metric or layout):
    - Express space in terms of physical distances of travel
    - Bird's eye view of the world
    - Not dependent upon the perspective of the robot
    - Independent of orientation and position of robot
    - Can be used to generate qualitative (route) representations

Intro. (con't.): Spatial Memory (con't.)

- Questions regarding spatial memory:
  - How accurately and efficiently does robot need to navigate?
  - Is navigation time-critical, or is a slightly sub-optimal route acceptable?
  - What are the characteristics of the environment?
  - Are there landmarks to provide orientation cues?
  - Are distances known accurately?
  - What are the sources of information about the environment that specify terrains, surface properties, obstacles, etc.?
  - What are the properties of the available sensors in the environment?

Metric Path Planning

- Objective: determine a path to a specified goal
- Metric methods:
  - Tend to favor techniques that produce an optimal path
  - Usually decompose path into subgoals called waypoints
- Two components to metric methods for path planning:
  - Representation (i.e., data structure)
  - Algorithm

Configuration Space

- Configuration Space (abbreviated: “Cspace”):
  - Data structure that allows robot to specify position and orientation of objects and robot in the environment
  - “Good Cspace”: Reduces # of dimensions that a planner has to deal with
  - Typically, for indoor mobile robots:
    - Assume 2 DOF for representation
    - Assume robot is round, so that orientation doesn’t matter
    - Assumes robot is holonomic (i.e., it can turn in place)
      - (Although there is much research dealing with path planning in non-holonomic robots)
  - Typically represents “occupied” and “free” space
    - “Occupied” → object is in that space
    - “Free” → space where robot is free to move without hitting any modeled object

Object Growing

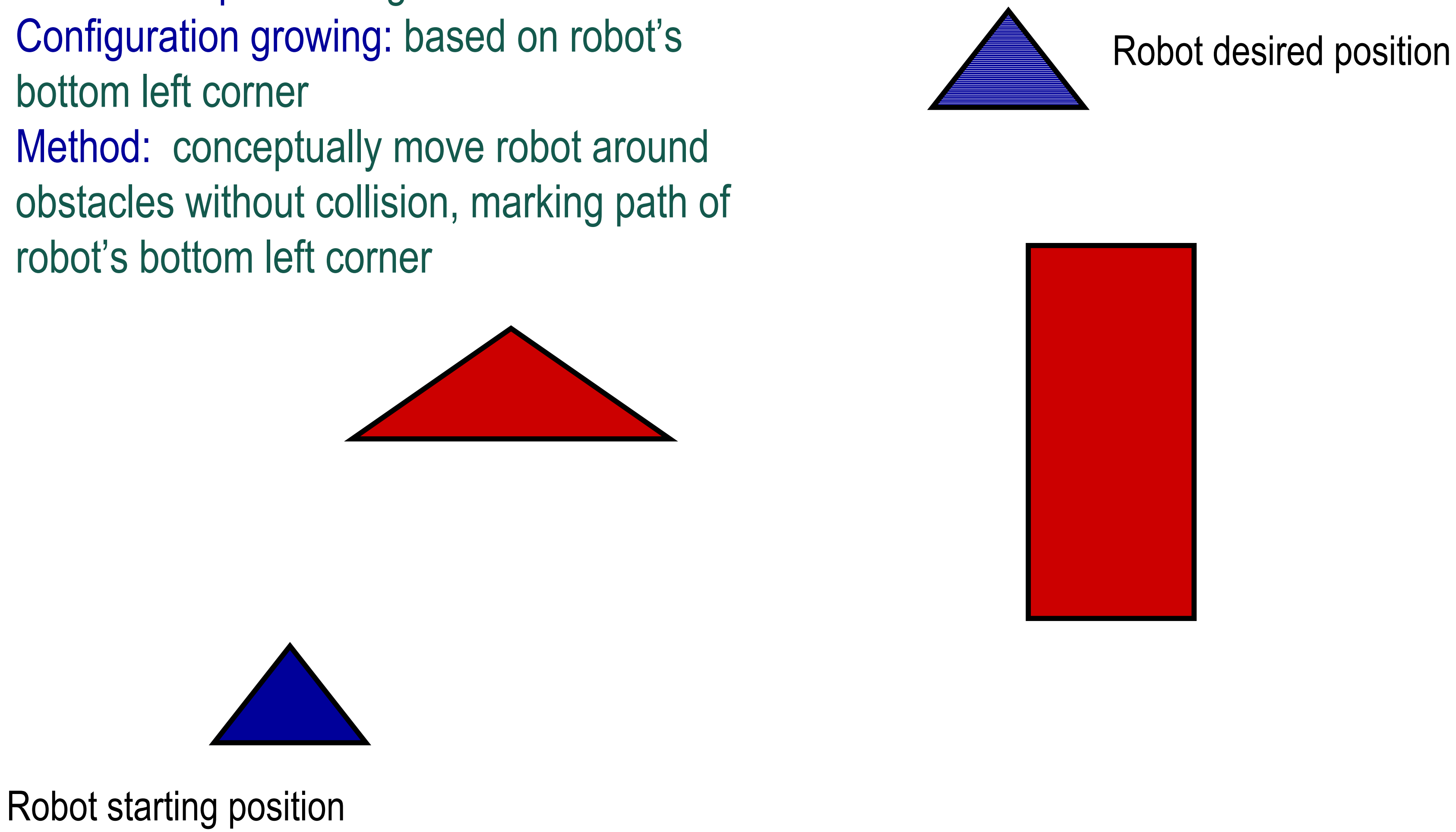
- Since we assume robot is round, we can “grow” objects by the width of the robot and then consider the robot to be a point

- Greatly simplifies path planning

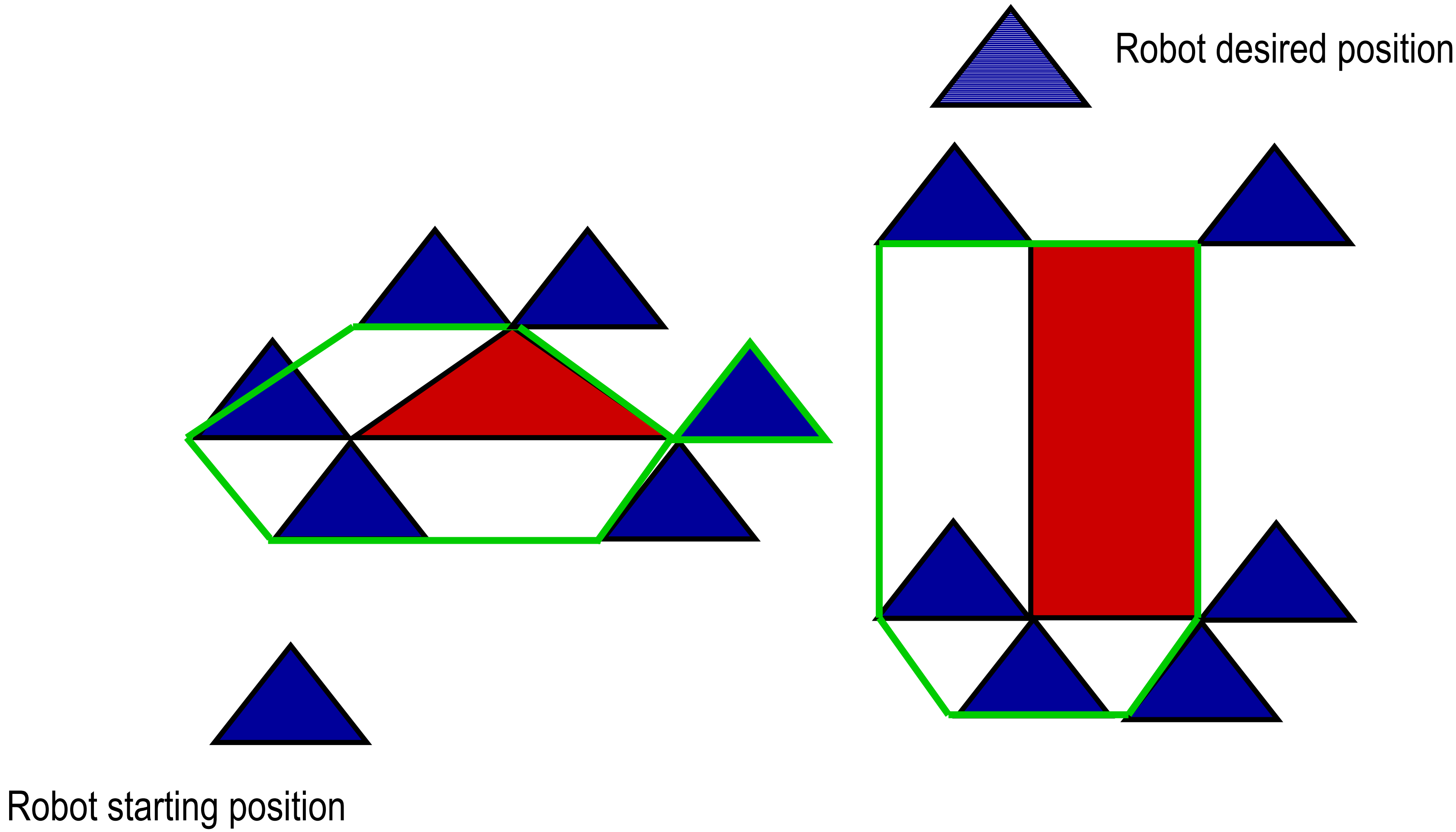
- New representation of objects typically called “configuration space object”

Method for Object Growing

- In this example: Triangular robot
- Configuration growing: based on robot's bottom left corner
- Method: conceptually move robot around obstacles without collision, marking path of robot's bottom left corner

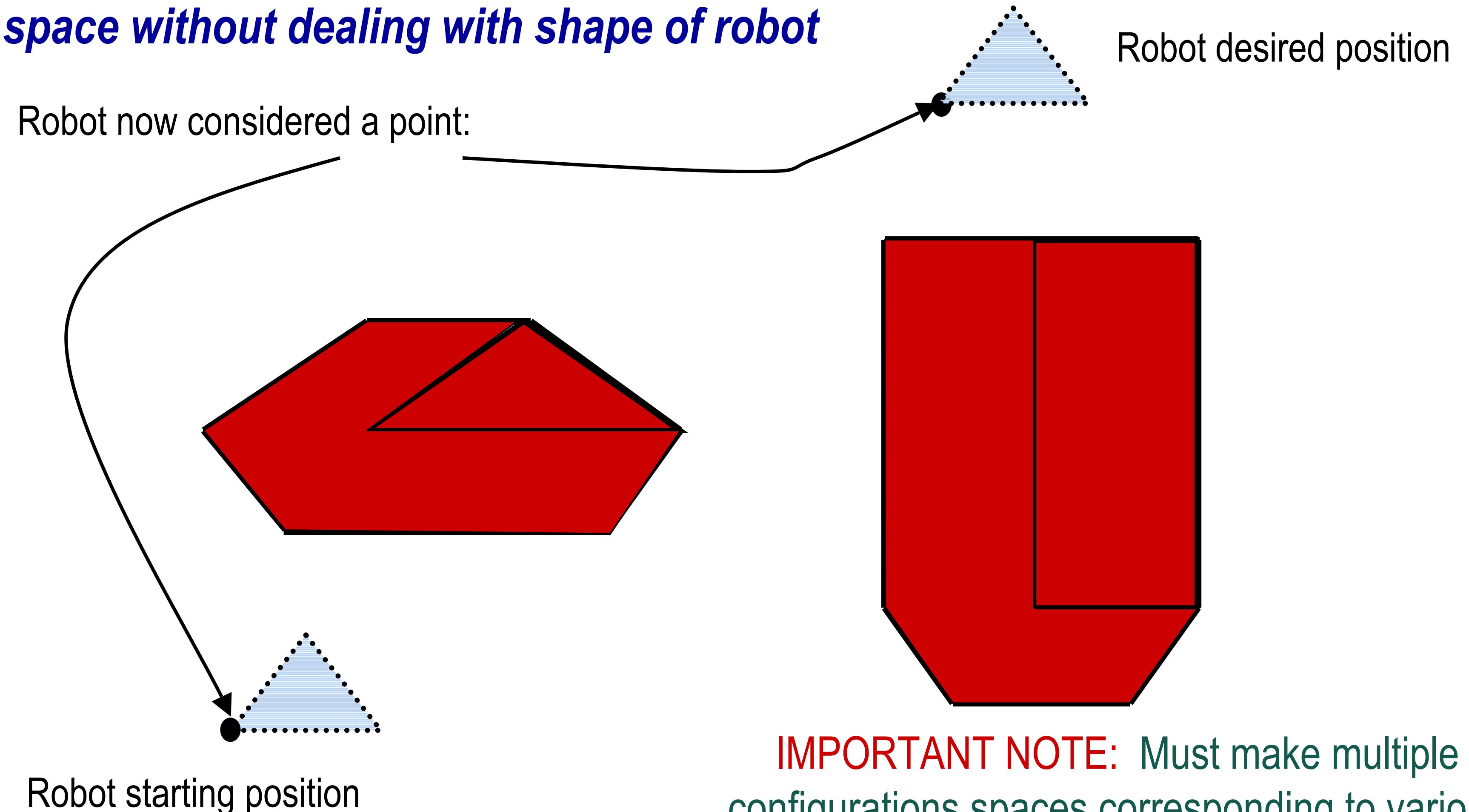


Method for Object Growing



Result of Object Growing: New Configuration Space

Can now plan path of point through this space without dealing with shape of robot



IMPORTANT NOTE: Must make multiple configurations spaces corresponding to various degrees of rotations for moving objects. Then, generalize search to move from space to space

Examples of Cspace Representations

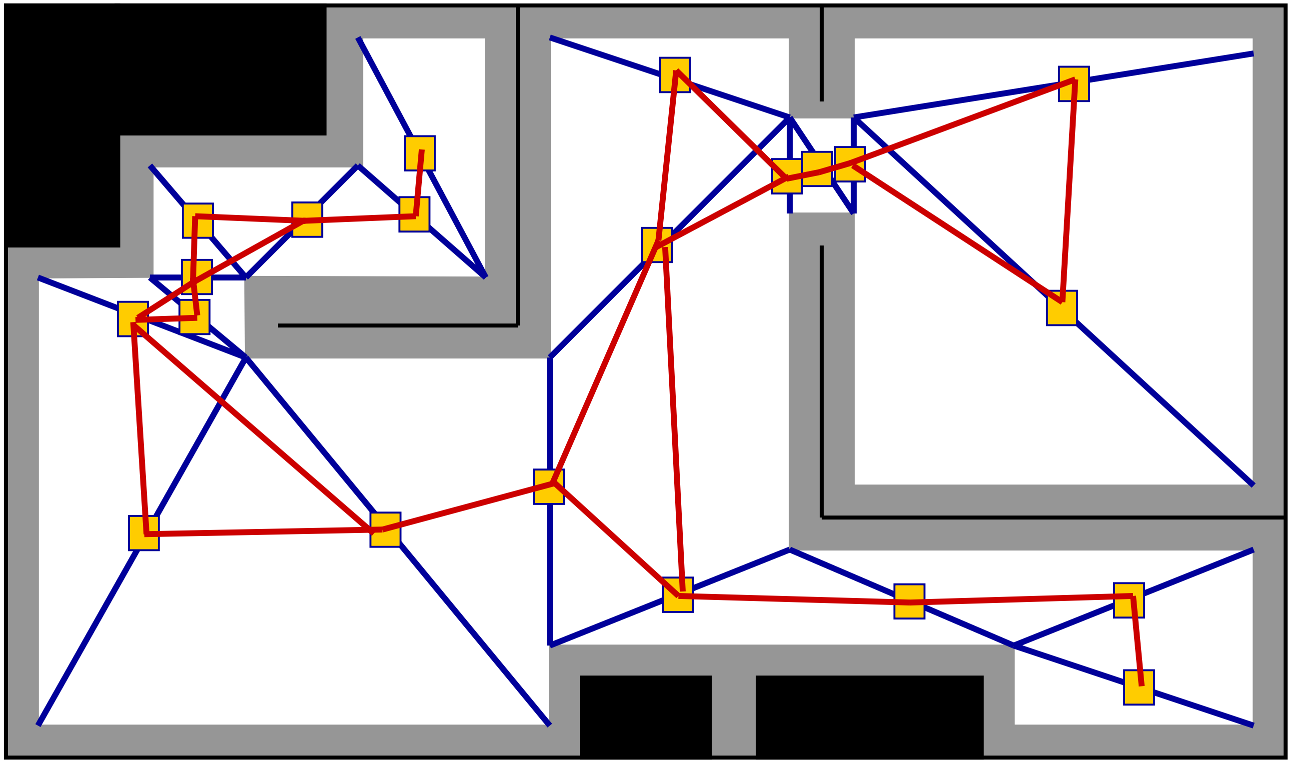
- Voronoi diagrams
- Regular grids
- Quadtrees/octtrees
- Vertex graphs
- Hybrid free space/vertex graphs (meadow map)

Meadow Maps (Hybrid Vertex-graph Free-space)

- Transform space into convex polygons
  - Polygons represent safe regions for robot to traverse
- Important property of convex polygons:
  - If robot starts on perimeter and goes in a straight line to any other point on the perimeter, it will not go outside the polygon
- Path planning:
  - Involves selecting the best series of polygons to transit through

Example Meadow Map

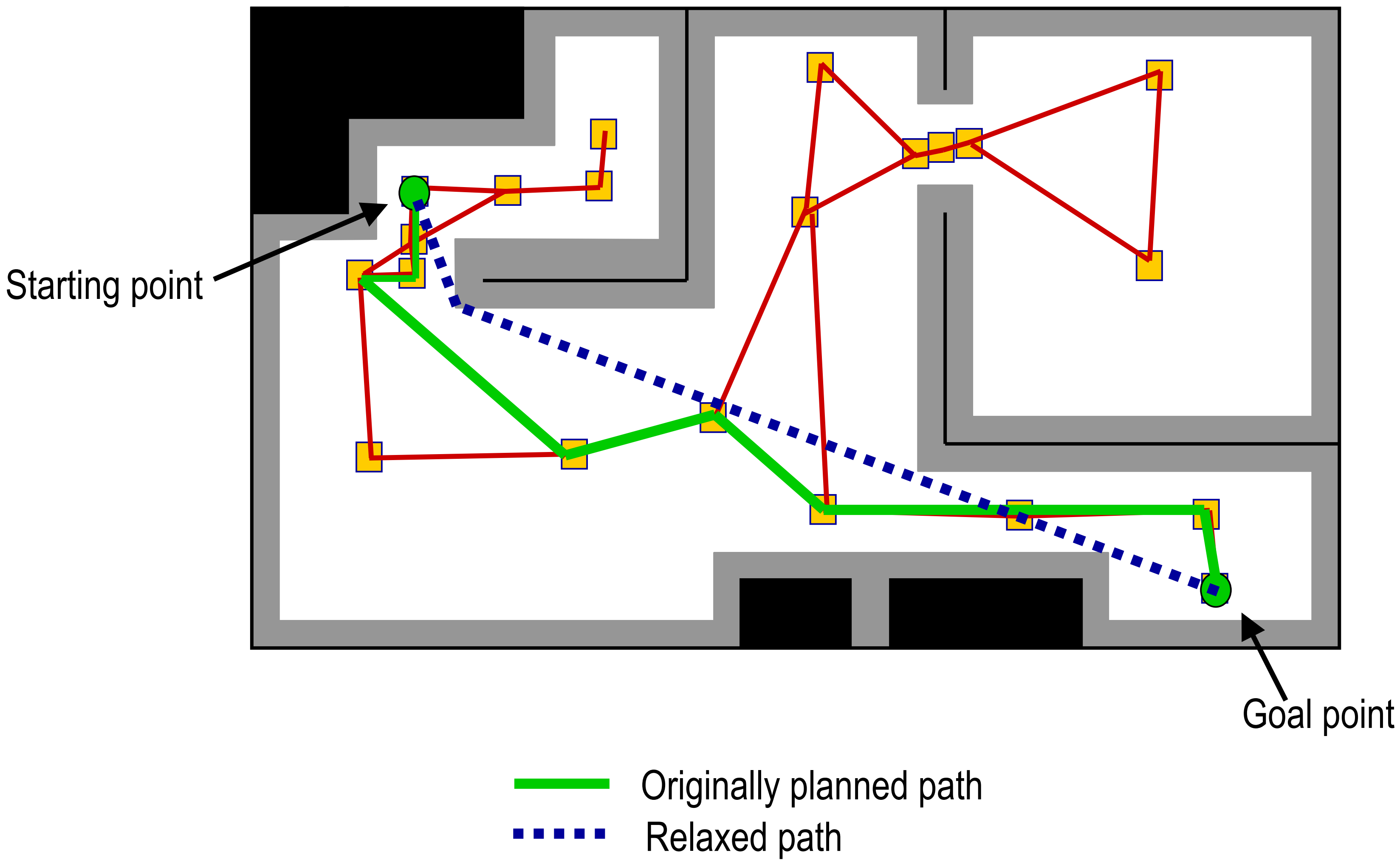
1. Grow objects
2. Construct convex polygons
3. Mark midpoints; these become graph nodes for path planner
4. Path planner plans path based upon new graph



Path Relaxation

- Disadvantage of Meadow Map:
  - Resulting path is jagged
- Solution: path relaxation
  - Technique for smoothing jagged paths resulting from any discretization of space
- Approach:
  - Imagine path is a string
  - Imagine pulling on both ends of the string to tighten it
  - This removes most of “kinks” in path

Example of Path Relaxation



Limited Usefulness of Meadow Maps

- Three problems with meadow maps:
  - Technique to generate polygons is computationally complex
  - Uses artifacts of the map to determine polygon boundaries, rather than things that can be sensed
  - Unclear how to update or repair diagrams as robot discovers differences between *a priori* map and the real world

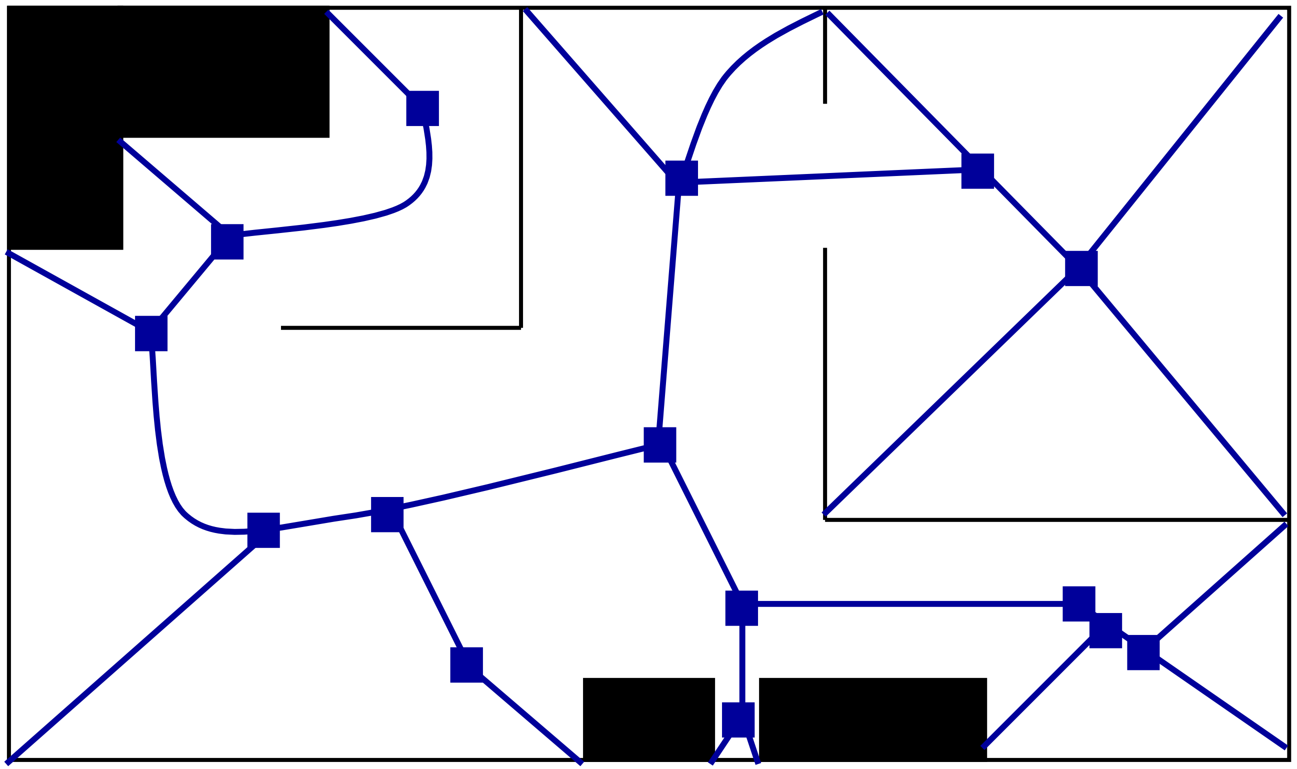
Generalized Voronoi Diagrams (GVGs)

- GVGs:
  - Popular mechanism for representing Cspace and generating a graph
  - Can be constructed as robot enters new environment

- Basic GVG approach:
  - Generate a Voronoi edge, which is equidistant from all points
  - Point where Voronoi edge meets is called a Voronoi vertex
  - Note: vertices often have physical correspondence to aspects of environment that can be sensed
  - If robot follows Voronoi edge, it won’t collide with any modeled obstacles → don’t need to grow obstacle boundaries

Example Generalized Voronoi Graph

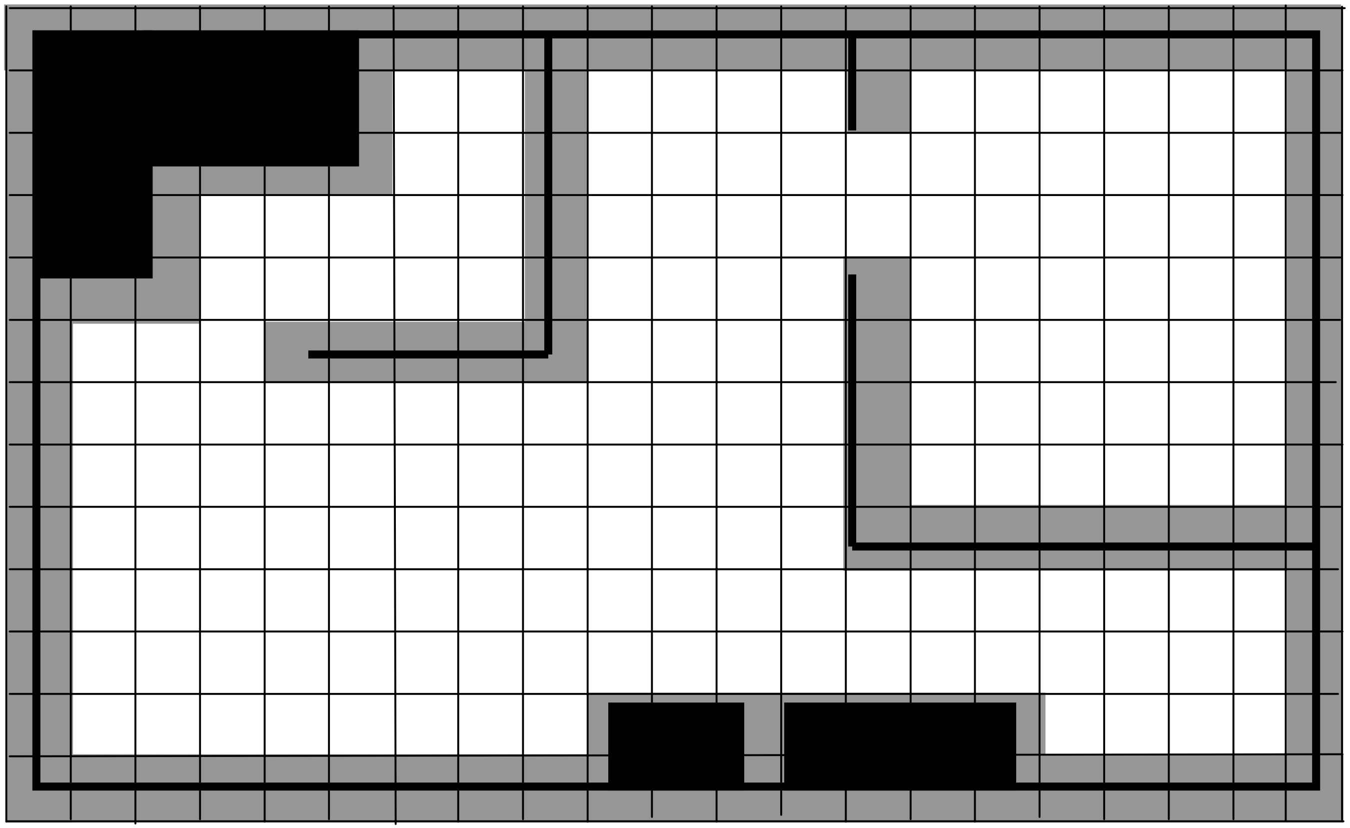
- (NOTE: This is only an approximate, hand-drawn graph to give the basic idea)



Regular Grids / Occupancy Grids

- Superimposes a 2D Cartesian grid on the world space
- If there is any object in the area contained by a grid element, that element is marked as occupied
- Center of each element in grid becomes a node, leading to highly connected graph
- Grids are either considered 4-connected or 8-connected

Example of Regular Grid / Occupancy Grid



Disadvantages of Regular Grids

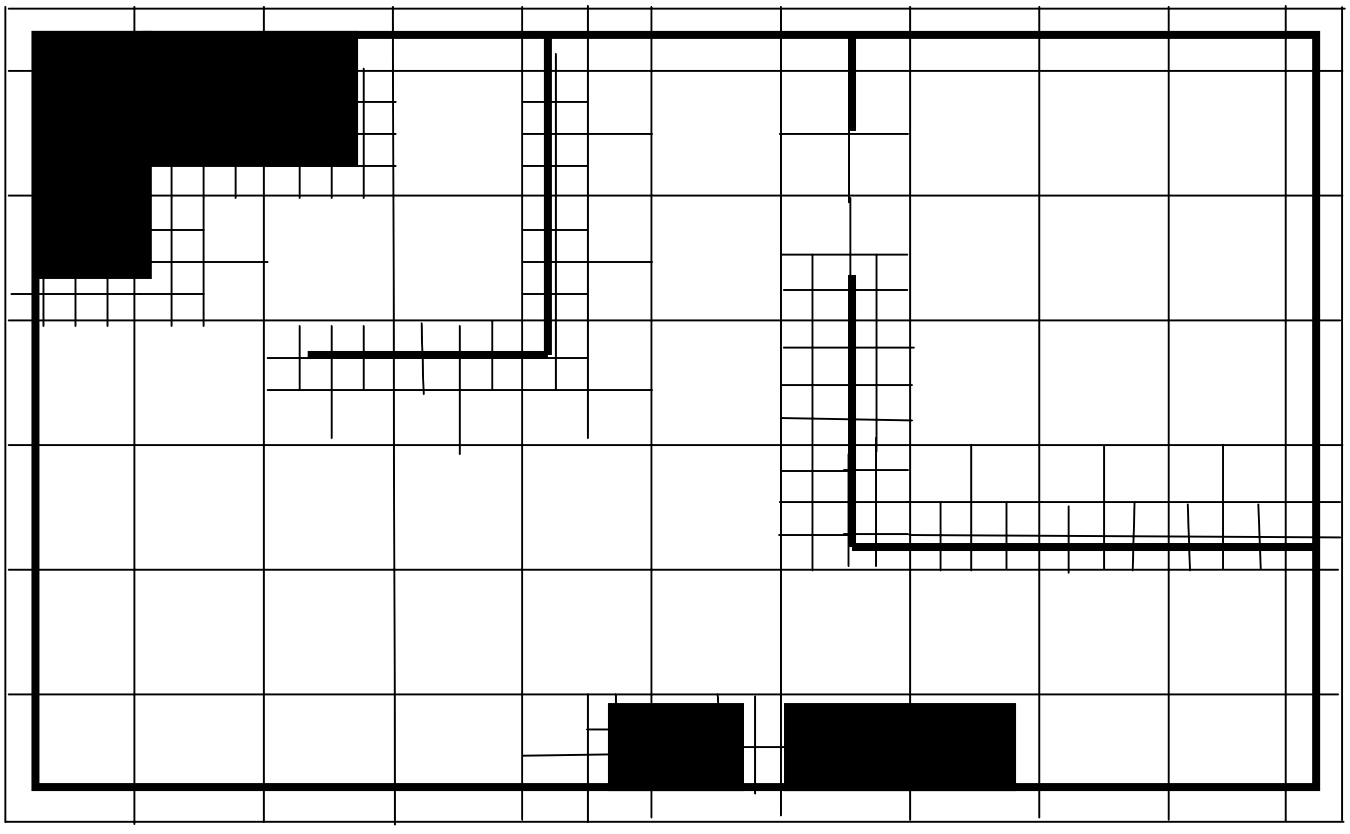
- Digitization bias:
  - If object falls into even small portion of grid element, the whole element is marked as occupied
  - Leads to wasted space
    - Solution: use fine-grained grids (4-6 inches)
    - But, this leads to high storage cost and high # nodes for path planner to consider
- Partial solution to wasted space: Quadtrees

Quadtrees

- Representation starts with large area (e.g., 8x8 inches)
- If object falls into part of grid, but not all of grid, space is subdivided into for smaller grids
- If object doesn’t fit into sub-element, continue recursive subdivision
- 3D version of Quadtree – called an Octree.

Example Quadtree Representation

(Not all cells are subdivided as in an actual quadtree representation (too much work for a drawing by hand!, but this gives basic idea)



Graph Based Planners

- Finding path between initial node and goal node can be done using graph search algorithms

- Graph search algorithms: found in networks, routing problems, etc.

- However, many graph search algorithms require visiting each node in graph to determine shortest path
  - Computationally tractable for sparsely connected graph (e.g., Voronoi diagram)
  - Computationally expensive for highly connected graph (e.g., regular grid)

- Therefore, interest is in “branch and bound” search
  - Prunes off paths that aren’t optimal

- Classic approach: A* search algorithm
  - Frequently used for holonomic robots

“A” Search Algorithm

- “A” search:
  - Produces optimal path
  - Starts at initial node and works way to goal node
  - Generates optimal path incrementally
  - Each update: considers nodes that could be added to the path, and selects best one
  - Evaluation function:

    $f(n) = g(n) + h(n)$

    where:

    $f(n)$ measures how good the move to node n is

    $g(n)$ measures cost of getting to node n from initial node

    $h(n)$ is the cheapest cost of getting from n to goal
- Problem: assumes you know $h(n)$ for all nodes, which means you have to visit all nodes to recurse in order to find $h(n)$

A* Search Algorithm

- A* (read "A star") search:
  - Reduces number of paths to be explored
  - No need to explore a path if it cannot be a good path
  - Estimates $h(n)$, even if no actual path available
  - Use this estimate to prune out paths that cannot be good
  - A* Evaluation function:

    $$f^*(n) = g^*(n) + h^*(n)$$

    where:

    * means these are estimates
  - In path planning, $g^*(n)$ is equivalent to $g(n)$

- How to estimate $h(n)$?

Estimating $h(n)$

- Must ensure that $h^*(n)$ is never greater than $h(n)$
  (NOTE: Error in book, page 362)

- Admissibility condition:
  - Must always underestimate remaining cost to reach goal

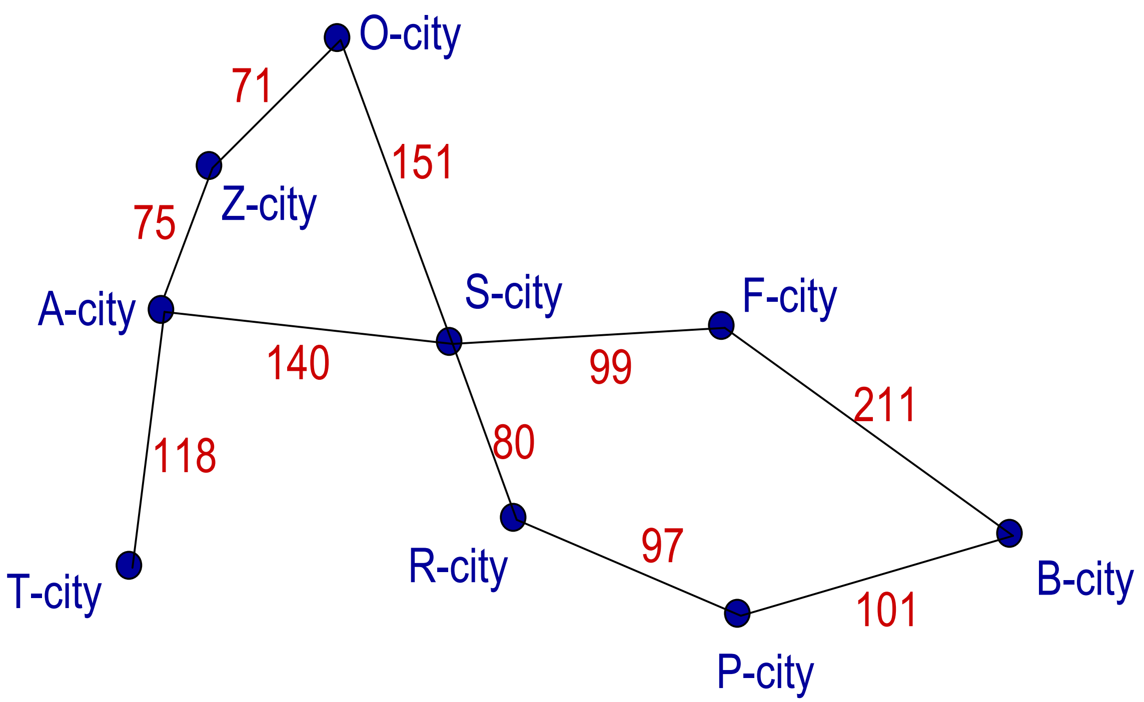
- Easy way to estimate:
  - Use Euclidian (straight line) distance
  - Straight line will always be shortest path
  - Actual path may be longer, but admissibility condition still holds

Example of A*

Compute optimal path from A-city to B-city

Straight-line distance to B-city from:

- A-city: 366
- B-city: 0
- F-city: 178
- O-city: 380
- P-city: 98
- R-city: 193
- S-city: 253
- T-city: 329
- Z-city: 374



Method for Example

- Expand each node from A-city, computing $f^*(n) = g^*(n) + h^*(n)$
- (Example worked on board)

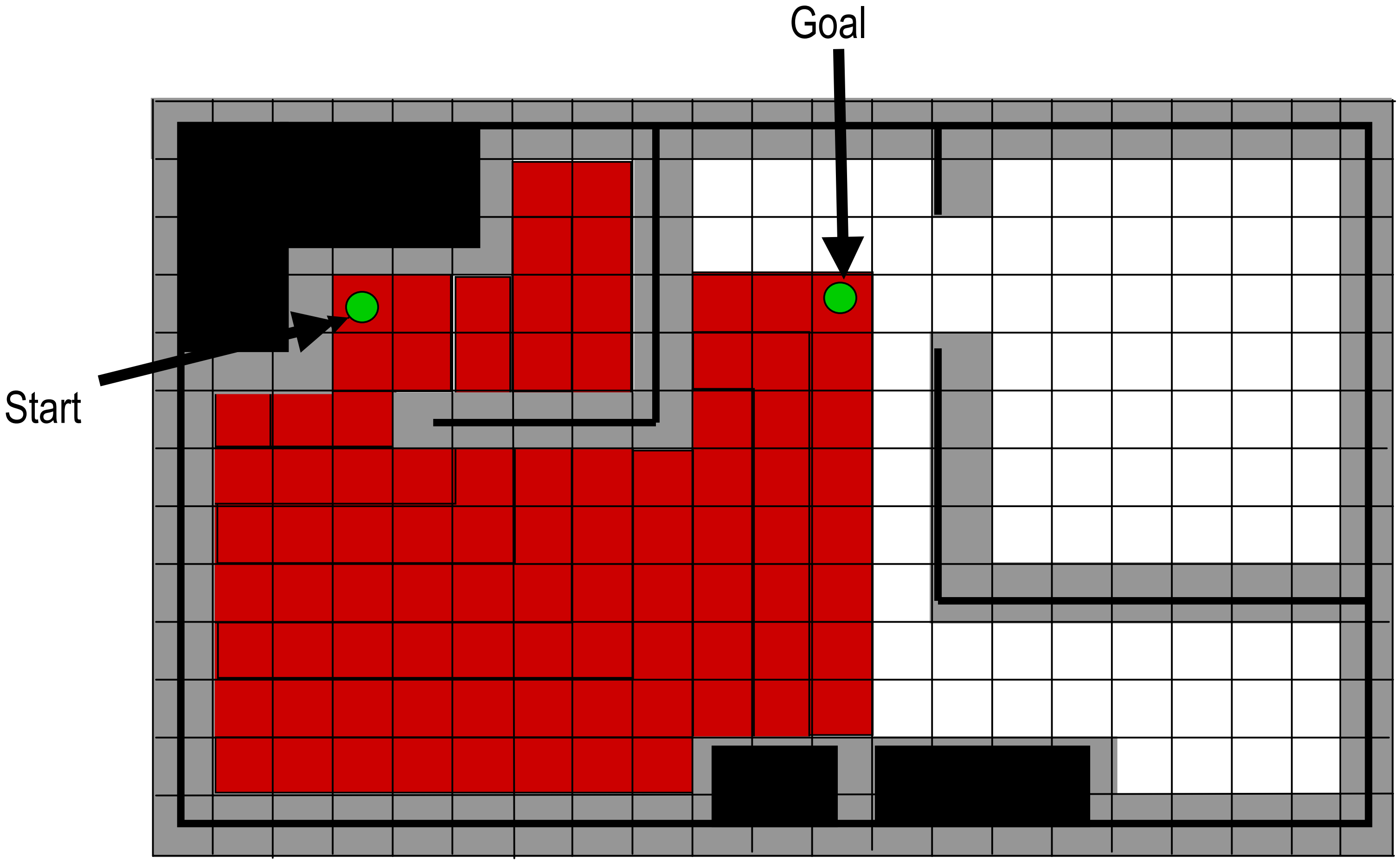
Pros and Cons of A* Search/Path Planner

- Advantage:
  - Can be used with any Cspace representation that can be transformed into a graph
- Limitation:
  - Hard to use for path planning when there are factors to consider other than distance (e.g., rocky terrain, sand, etc.)

Wavefront-Based Path Planners

- Well-suited for grid representations
- General idea: consider Cspace to be conductive material with heat radiating out from initial node to goal node
- If there is a path, heat will eventually reach goal node
- Nice side effect: optimal path from all grid elements to the goal can be computed
- Result: map that looks like a potential field

Example of Wavefront Planning



Wavefront Propagation Can Handle Different Terrains

- Obstacle: zero conductivity
- Open space: infinite conductivity
- Undesirable terrains (e.g., rocky areas): low conductivity, having effect of a high-cost path

Summary of Metric Path Planning

- Converts world space to a configuration space
- Use obstacle growing to enable representation of robot as a point
- Cspace representations exploit interesting geometric properties of the environment
- Representations can be converted to graphs
- A* works well with Voronoi diagrams, since they produce sparse graphs
- Wavefront planners work well with regular grids
- Metric path planning tends to be computationally expensive
- Limitation of popular path planners: assume holonomic robots

Preview of Next Class

- Topological Path Planning