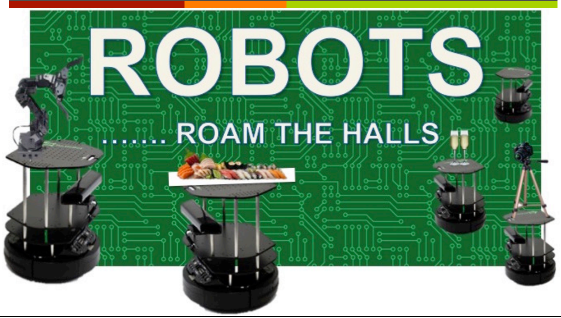
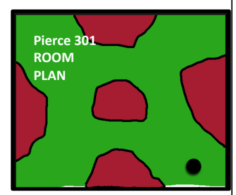
CS 189: Autonomous Robot Systems

Spring 2018, Fridays 1-4pm, Pierce 301



Pset 4: Scavenger Hunt!

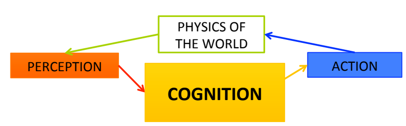
- Part (a): Due next Friday
  - Recognize all three objects
- Part (b) Due Friday after Spring Break
  - Competition!
  - Wandering Rough Space Map: Similar corridor widths to last time, but some "side arm" areas.
- Competition (full class time)
  - First round: 5 robots and teams
  - Second round: 5 robots and teams
  - *You only need to come for your round*



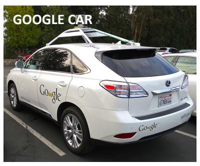
Agenda

- Lecture: Navigation I: Path Planning
- Demo Time: Pset 3b Follower, Pierce 301
- Upcoming:
  - Following week: Pset 4a (scavenger, part 1)
- References: [see note on Piazza]
  - "Introduction to AI Robotics", chapter 9 and 10, Robin Murphy, 2000.
  - "Intro to Autonomous Mobile Robots", chapter 5.5 , 6.1-2, Seigwart et al, 2004
  - "Robot Motion Planning", Lecture Notes, Choset and others (CMU 16-735)

What Does it Mean to be Autonomous?



Today: Robots Navigating the World



Scenarios

- *Hospital Helper (e.g. Diligent, Tugs)*
- *Office security or mail-delivery (e.g. Cobal, Savioke)*
- *Tour Guide robot in a museum (Minerva)*
- *Autonomous Car with GPS and Nav system*

Biological analogies:
Humans, bees and ants, migrating birds, herds

Today: Robots Navigating the World

Second Part of CS189: High-level reasoning
From finite state machines to complex representation and memory

- Path Planning: *How to I get to my Goal?*
- Localization: *Where am I?*
- Mapping: *Where have I been?*
- Exploration: *Where haven't I been?*

What is Path Planning?

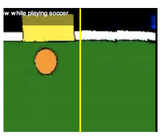
- **Simple Question:** *How do I get to my Goal?*
- **Not a simple answer!**
  - Can you see your goal?
    Do you have a map?
    Are obstacles unknown or dynamic?
  - Does it matter how fast you get there?
    Does it matter how smooth the path is ?
  - How much compute power do you have?
    How precise is your motion control?
- Path Planning is best thought of as a Collection of Algorithms
  - You have to match the method to the *"ecological niche"*
    - Environment, Success metrics, Robot capability.

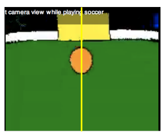
Types of Path Planning Approaches

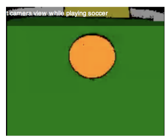
- Reminder of the Basics
  - Visual homing (Purely local sensing and feedback control)
  - Inverse Kinematics (Turn-move-turn to get from A to B)
- Bug-based Path Planning (mostly-local without a map)
  - Robots can see the Goal (direction and distance)
  - But there are unknown obstacles in the way (No map)
- Metric (A*) Path Planning (global with a map)
  - Assumes that you have a *map (distance or graph)* and you know where you and the goal are located in it.
  - Path is represented as a of series of waypoints

Basics: Visual Homing

- Purely Reactive Navigation
  - Measure Visual (x,y) Position of Goal
  - Move to bring goal to Visual Center
  - Proportional Control (if you see the goal), Random walk (if you don't)

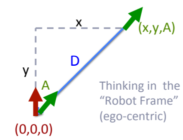






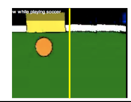
Basics: Inverse Kinematics

- Getting from Here to Point B
  - Popular Option: Turn-Move-Turn *[Lecture: Autonomy 1]*
  - Non-holonomic constraints; Infinite possible paths
  - No obstacles (like in visual homing example)



Path Planned is: Turn A then Move D

Turn A = atan2 (x/y) = W x duration
Move D = sqrt (x^2 + y^2) = L x duration
(Turn again, to end in new orientation)



Example: Line up Ball & Goal

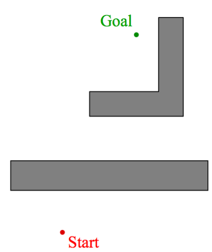
Bug-based Path Planning

What if the Robot has obstacles in the way?
- Always have Goal direction and/or distance (Global)
- But No Map: Only local knowledge of environment (Local)
- *Example Scenario:*
  - *Robot knows GPS location of goal, but unknown buildings in the way*
  - *Indoor robot see goal location, but furniture in the way.*

"Bug" Algorithms depend on simple but provable behaviors!
- Don't need to build a map
- *Simple Computation: Visual Homing + Wall-following + Odometry*

- Very intuitive class of algorithms – but surprisingly powerful

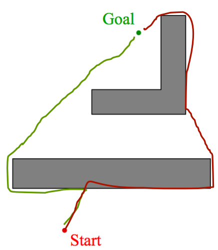
Basic Idea: Bug o



- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

Adapted from Choset 16-735

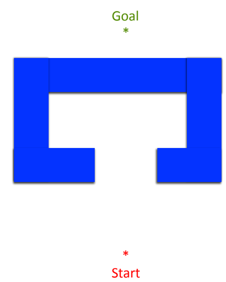
Basic Idea: Bug 0



- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

Adapted from Choset 16-735

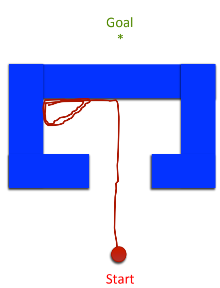
What map will foil Bug 0?



- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

Adapted from Choset 16-735

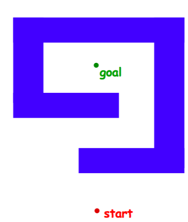
What map will foil Bug 0?



- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

Adapted from Choset 16-735

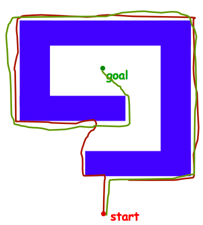
What map will foil Bug 0?



- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

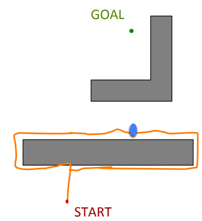
Adapted from Choset 16-735

What map will foil Bug 0?



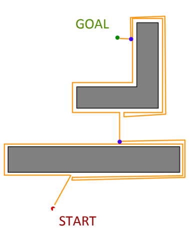
- Robot
  - Known direction to goal
  - Wall-following
- Bug 0 Algorithm
  - Head towards goal
  - If obstructed, follow obstacle wall until you can head towards goal again.
  - Continue

A Better Bug: Bug 1



- Robot
  - Known direction to goal
  - Wall-following
  - Measure distance to goal
  - Odometry with encoders
- Bug 1 Algorithm
  - Head towards goal
  - If obstructed, circumnavigate the obstacle and remember the point P on the perimeter that is closest to the goal
  - Return to that closest point and continue.

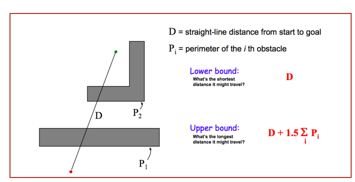
A Better Bug: Bug 1



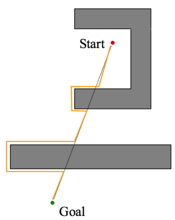
- Robot
  - Known direction to goal
  - Wall-following
  - Measure distance to goal
  - Odometry with encoders
- Bug 1 Algorithm
  - Head towards goal
  - If obstructed, circumnavigate the obstacle and remember the point P on the perimeter that is closest to the goal
  - Return to that closest point and continue.

What map will foil Bug 1?

- **None!**
  - *Any reasonable world* (finite number of obstacles with finite perimeter)
  - Analysis: It is possible to bound worst and best case trajectories
  - Discussion: What do you think are the pros and cons of this approach?

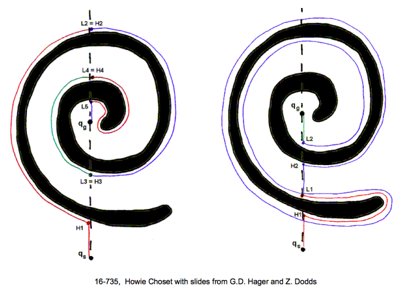


An Alternative: Bug 2



- Robot
  - Known direction to goal
  - Wall-following
  - Measure distance to goal
  - Odometry with encoders *or* orientation to goal
- **M-line**
  - **Line from the start to goal**
- Bug 2 Algorithm
  - Head toward goal on the m-line
  - If an obstacle is in the way, follow it until you encounter the m-line again and you are closer to the goal.
  - Leave the obstacle and continue toward the goal

Some Fun Examples: Bug2



Adapted from Choset 16-735

Many Types of Bug Algorithms!

- Recent Variant: i-Bug (intensity-Bug, Lavalle etc al)
  - Proved that you can exit an obstacle at the first point "closer" to the goal (don't need to keep track of m-line)
- Attractive for many reasons
  - Simplicity of implementation and robot assumptions, ability to deal with unknown and dynamic environments, and *the analogy to ant behavior.*

Open question: Do ants (bugs) use the bug algorithms?

Many Types of Bug Algorithms!

Types of Path Planning Approaches

- Reminder of the Basics
  - Visual homing (Purely local sensing and feedback control)
  - Inverse Kinematics (Turn-move-turn to get from A to B)
- Bug-based Path Planning (mostly-local without a map)
  - Robots can see the Goal (direction and distance)
  - But there are unknown obstacles in the way (No map)
- Metric (A*) Path Planning (global with a map)
  - Assumes that you have a *map (distance or graph)* and you know where you and the goal are located in it.
  - Path is represented as a of series of waypoints

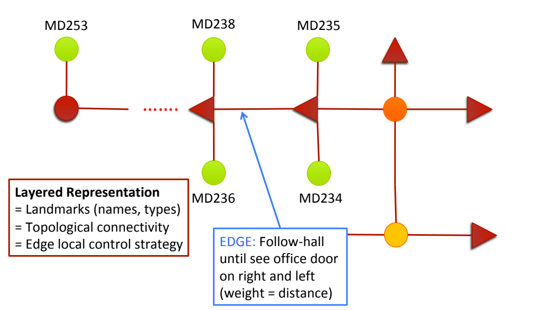
Metric/Global Path Planning

- What if the Robot has Full Knowledge
  - A map of the environment and robot + goal's locations
  - Goal: Find a "optimal" path (typically distance but other possibilities)
  - We will focus on robots, but it's a general problem *(think Google maps)*
- Two Components
  - Map Representation ("graph"):
    - Feature based maps (office numbers, landmarks)
    - Grid based maps (cartesian, quadtrees)
    - Polygonal maps (geometric decompositions)
  - Path Finding Algorithms:
    - Shortest-Path Graph Algorithms (Breadth-First-Search, A* Algorithm)

Map Representation: Feature based

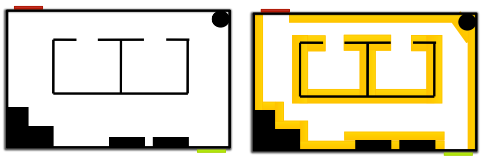
- Also known as a Topological or Landmark-based Map
  - Features your robot can recognize:
    - Includes **both natural** landmarks (corner, doorway, hallways) and **artificial** ones (office door numbers; or robot-friendly tags)
  - Gateways are landmarks that represent decisions (e.g. intersection)
  - Distinguishable places are unique landmarks
- World is a graph that connects landmarks
  - Edges represent **actual motion:** how to get from landmark A to landmark B
    Usually visual/reactive navigation is possible along an edge
  - Edges can also keep **extra attributes**: distance, time it takes, etc.
- Google Maps are topological maps for humans (e.g. turn at intersection)
  - *Caveat: Much less easy to construct topological maps for robots!*

Example: Maxwell-Dworkin



Map Representation: Grid based

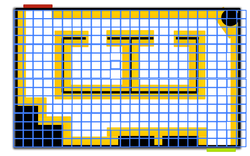
- Ignore any notion of Features
- Instead, Convert the map into a grid-graph
  - Step 1: Grow the boundaries (by robot size)
  - Step 2: Overlay a grid



Adapted from Murphy 2000

Map Representation: Grid based

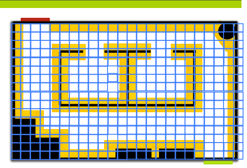
- **Basic: An Occupancy Matrix**
  - Problem:
    - How do you choose the "resolution" of the grid?
    - Too small – computationally expensive, jagged paths
    - Too big – might miss paths



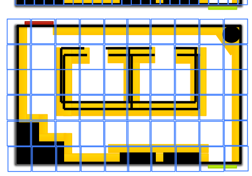
Note: Occupancy Grids will be more useful later, when the robot is responsible for making the map!

Map Representation: Grid based

- Basic: An occupancy matrix
  - Problem:
    - How do you choose the "resolution" of the grid?
    - Too small – computationally expensive, jagged paths
    - Too big – might miss paths

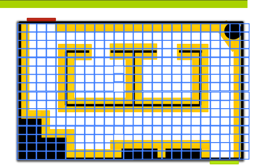


- Quadtree
    - Create a grid recursively!
    - Start with very coarse grid;
    - Then for each grid section, if there is an obstacles, refine.
    - *Outcome: Captures large open spaces as a single big grid point*

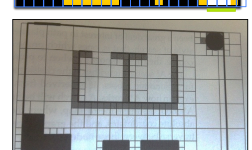


Map Representation: Grid based

- Basic: An occupancy matrix
  - Problem:
    - How do you choose the "resolution" of the grid?
    - Too small – computationally expensive, jagged paths
    - Too big – might miss paths



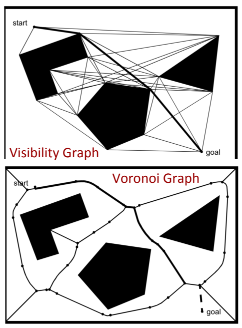
- Quadtree
    - Create a grid recursively!
    - Start with very coarse grid;
    - Then for each grid section, if there is an obstacles, refine.
    - *Outcome: Captures large open spaces as a single big grid point*

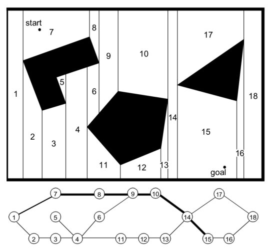


Murphy 2000

More Map Representations

From "Introduction to Autonomous Mobile Robots", Chapter 5 and 6, Seigwart and Nourbaksh, 2004.





Cell Decomposition Map

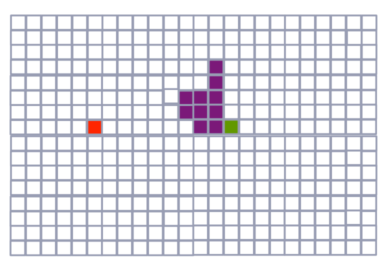
Metric/Global Path Planning

- What if the Robot has Full Knowledge
  - A map of the environment and robot + goal's locations
  - Goal: Find a "optimal" path (typically distance but other possibilities)
  - We will focus on robots, but it's a general problem *(think Google maps)*
- Two Components
  - Map Representation ("graph"):
    - Feature based maps (office numbers, landmarks)
    - Grid based maps (cartesian, quadtrees)
    - Polygonal maps (geometric decompositions)
  - **Path Finding Algorithms:**
    - **Shortest-Path Graph Algorithms (Breadth-First-Search, A* Algorithm)**

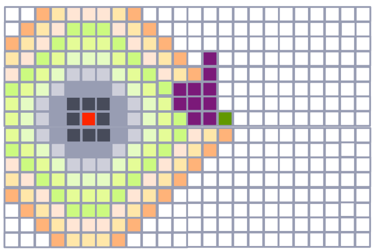
Path Finding Algorithms

- All Map Representations are a weighted "graph"
  - Nice part is that you only need to do this once (amortize computation)
- Algorithm: Compute shortest paths in the graph
  - Path is represented by a series of waypoints
  - Single Path Search Algorithms: Find shortest path A to B
    - Breadth-First-Search (simple graphs); Dijkstra's (weighted)
    - A* search for large graphs (BFS + Heuristic)
  - Gradient Path Algorithms: Find *all paths* towards B
    - E.g. Fixed Basestation: BFS, Dijkstra's, Wavefront algorithms, etc

Breadth-First Search

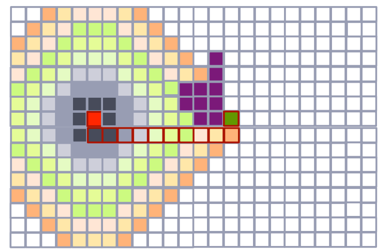


Breadth-First Search



Breadth-First Search

Note that bug 2 ("m-line") would have worked well in this case too!
If few obstacles, then bug is good enough



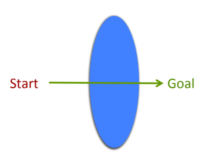
A* Algorithm

A* Algorithm
Similar to BFS but choose next node to expand based on two things
1. Distance from start (like BFS)
2. Expected distance from goal (H)

"H" is the *heuristic*. The theory shows that so long as the heuristic is "optimistic" then A* returns the best path.

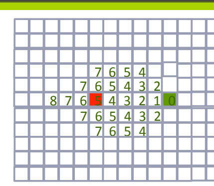
Key point:
Average behavior can be awesome!

For maps,
H = straight-line distance is a good heuristic

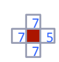


How A* works

How BFS would
Explore the space

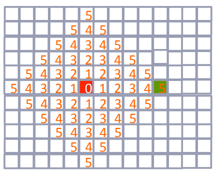


Manhattan distance to Green
(easy to compute directly)
(no obstacles considered)

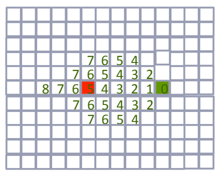


A* criteria = BFS+Manhattan

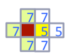
How A* works



How BFS would
Explore the space

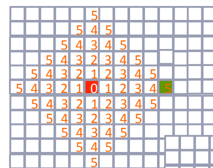


Manhattan distance to Green
(easy to compute directly)
(no obstacles considered)

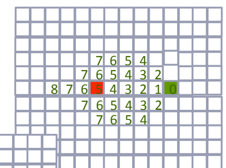


A* criteria = BFS+Manhattan

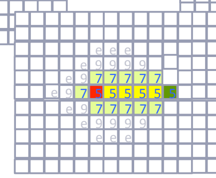
How A* works



How BFS would
Explore the space



Manhattan distance to Green
(easy to compute directly)
(no obstacles considered)



A* criteria = BFS+Manhattan

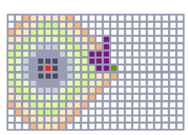
A* Algorithm

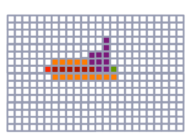
A* Algorithm
Similar to BFS but choose next node to expand based on two things
1. Distance from start (like BFS)
2. Expected distance from goal (H)

"H" is the *heuristic*. The theory shows that so long as the heuristic is "optimistic" then A* returns the best path.

Key point:
Average behavior can be awesome!

For maps,
H= straight-line distance is a good heuristic





A* is a "general" graph search (AI, game tree, orbitz, etc); see Murphy 2000 chapter 10 for more details

Case Studies and AAAI Competitions

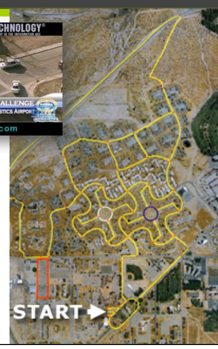
Given a "map" of the environment with some landmarks.

Given initial position (not pose) and final goal

Unknown obstacles might be introduced

AAAI 1992 and 1994 Mobile robot competitions [Murphy 2000]

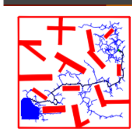
DARPA Urban Challenge (2007)



Final Thoughts

- Robot systems must combine many ideas
  - Interleave bug like navigation with serious path planning
  - High-level maps and low-level primitives
    - e.g. collision avoidance, feature recognition, etc
  - Ecological niche matters!
    - E.g. Robot soccer is very different from a mail-delivery robot.
- Cool New Methods
  - RRT: Rapidly exploring Random Trees
  - Combining with Probabilistic localization

RRT solves hard problems



High-dimensional Spaces
Complex movement constraints

(Parallel parking is hard, but now imagine parking a car with three hitched trailers!)

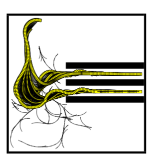


Figure 13: *A nonholonomic planning problem that involves a car pulling three trailers. There are seven degrees of freedom.*

RRT

- Sample
  - Pick some random points
  - Based on voronoi areas
    - Bias towards open spaces)
    - Bias towards goal, if one exists
- Extend
  - Connect the new point to your old path by seeing how close your robot can get to that point
    - *extend using actual (complex) dynamics model of the robot*