Introduction to Game Programming and Robotics

Unit # 12

Acknowledgement

- Most of the examples/material presented in this presentation is taken from the "Introduction to AI Robotics" book written by Robin Murphy

Navigation

- Navigation remains one of the most challenging functions to perform, in part because it involves practically everything about AI robotics: sensing, acting, planning, architectures, hardware, computational efficiencies, and problem solving.
- Reactive robots have provided behaviors for moving about the world without collisions, but navigation is more purposeful and requires deliberation.
- In particular, a robot needs to be able to plan how to get a particular location.
- Two different categories of techniques have emerged: *topological navigation* and *metric navigation*, also known as *qualitative navigation* and *quantitative navigation*, respectively.

Four Questions

- The functions of navigation can be expressed by four questions:
- **Where am I going?** This is usually determined by a human or a mission planner.
- Roboticists generally do not include this as part of navigation, assuming that the robot has been directed to a particular goal.
- **What's the best way there?** This is the problem of path planning, and is the area of navigation which has received the most attention.

Four Questions (Cont'd)

- **Where have I been?** As a robot explores new environments, it may be part of its mission to map the environment.
- But even if the robot is operating in the same environment (e.g., delivering mail in an office building), it may improve its performance by noting changes. A new wall may be erected, furniture rearranged, paintings moved or replaced.
- **Where am I?** In order to follow a path or build a map, the robot has to know where it is; this is referred to as localization.
- Localization can be relative to a local environment (e.g., the robot is in the center of the room), in topological coordinates (e.g., in Room 311), or in absolute coordinates (e.g., latitude, longitude, altitude).

Sensor Uncertainty

- The assumption that sensors would give an accurate representation of the world is often not true.
- Sensors are always noisy and have vulnerabilities.
- Therefore, a robot has to operate in the presence of uncertainty.
- In the Reactive Paradigm, the way in which the sensors were coupled with the actuators accepted this uncertainty.
- If the sonar or IR returned an incorrect range reading, the robot may appear to start to avoid an imaginary obstacle.

Sensor Uncertainty (Cont'd)

- However, the process of moving often eliminated the source of the noisy data, and soon the robot was back to doing the right thing.
- Uncertainty becomes more serious when dealing with map making and localization; therefore a new wave of techniques has been developed to smooth over sensor noise and ascertain the correct state of the world.

Spatial Memory (Cont'd)

- Spatial memory takes two forms: *route (qualitative) and layout (metric/quantitative)* representations.
- Route representations express space in terms of the connections between landmarks.
- An example of a route representation is when a person gives directions propositionally (as a list): "go out of the parking lot, and turn left onto Fowler Drive. Look for the museum on the right, and turn left at next traffic light."
- Route representations also tend to supply orientation cues: "out of the parking lot" (versus being contained in it), "turn left," "on the right."
- These orientation cues are egocentric, in that they assume the agent is following the directions at each step.

Spatial Memory (Cont'd)

- Layout representations are the opposite of route representations.
- When a person gives directions by drawing a map, the map is a layout representation.
- The major differences between layout and route representations are the viewpoint and utility.
- A layout representation is essentially a bird's-eye view of the world.
- The layout is orientation and position independent. Layout representations can be used to generate a route representation, but this doesn't necessarily work the other way.

Relational Methods in Route Navigation

- *Topological, route,* or *qualitative navigation* is often viewed as being more simple and natural for a behavior-based robot.
- Relational techniques are the most popular, and can be thought of as giving the robot an abbreviated, "connect the dots" graph-style of spatial memory.

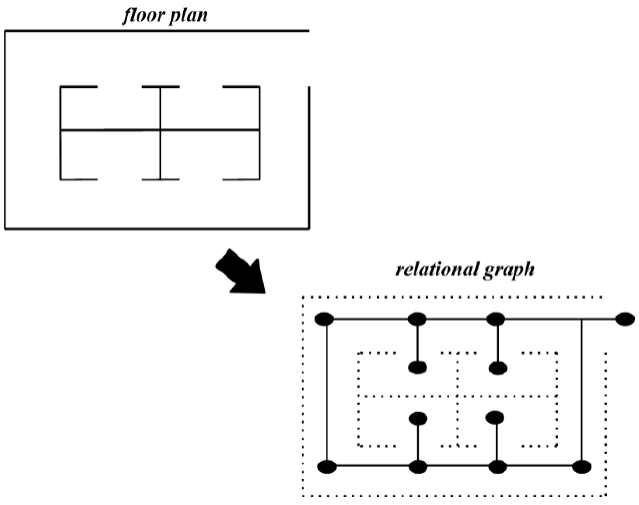
Landmark

- Topological navigation depends on the presence of landmarks.
- A *landmark* is one or more perceptually distinctive features of interest on an object or locale of interest.
- Landmarks are used in most aspects of navigation.
- If a robot finds a landmark in the world and that landmark appears on a map, then the robot is localized with respect to the map.

Relational Methods

- Relational methods represent the world as a graph or network of nodes and edges. Nodes represent landmarks or goals.
- Edges represent a navigable path between two nodes, in effect that two nodes have a spatial relationship.

Relational Graph



Metric Navigation

- *Metric path planning,* or *quantitative navigation,* is the opposite of topological navigation.
- As with topological methods, the objective is to determine a path to a specified goal.
- The major philosophical difference is that metric methods generally favor techniques which produce an optimal, according to some measure of best, while qualitative methods seem content to produce a route with identifiable landmarks or gateways.
- Another difference is that metric paths usually decompose the path into subgoals consisting of *waypoints.*
- These waypoints are most often a fixed location or coordinate (x,y).

Reactive vs. Deliberative

- The ability to produce and compare all possible paths also assumes that the planning has access to a pre-exisiting (or *a priori)* map of the world.
- Equally as important, it assumes that the map is accurate and up to date.
- As such, metric methods are compatible with deliberation, while qualitative methods work well with more reactive systems.

Metric Path Planners

- Metric path planners have two components: the *representation* (data structure) and the *algorithm.*
- Path planners first partition the world into a structure amenable for path planning.
- The intent of any representation is to represent only the salient features, or the relevant configuration of navigationally relevant objects in the space of interest; hence the term configuration space.

A* Search Algorithm

- The A* search algorithm is the classic method for computing optimal paths for holonomic robots.
- It is derived from the A search method.
- In order to explain how A*works, A search will be first presented with an example, then A*.
- Both assume a metric map, where the location of each node is known in absolute coordinates, and the graph edges represent whether it is possible to traverse between those nodes.

A Search Algorithm

- The A search algorithm produces an optimal path by starting at the initial node and then working through the graph to the goal node.
- It generates the optimal path incrementally; each update, it considers the nodes that could be added to the path and picks the best one.
- It picks the “right” node to add to the path every time it expands the path (the "right node” is more formally known as the plausible move).

A Search Algorithm (Cont'd)

- The heart of the method is the formula (or evaluation function) for measuring the plausibility of a node:

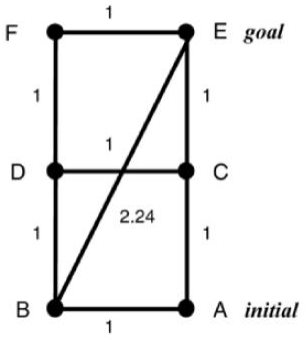
$$f(n) = g(n) + h(n)$$

where:

- $f(n)$ measures how good the move to node n is
- $g(n)$ measures the cost of getting to node n from the initial node. Since A expands from the initial node outward, this is just the distance of the path generated so far plus the distance of the edge to node n
- $h(n)$ is the cheapest cost of getting from n to goal

A Search Algorithm (Cont'd)

- Assume that a Cspace representation yielded the graph.
- The A search algorithm begins at node A and creates a tree-like structure to determine which are the possible nodes it can consider adding to its path.
- There are only two nodes to choose from: B and C.



A Search Algorithm (Cont'd)

- In order to determine which node is the right node to add, the A search algorithm evaluates the plausibility of B and C by looking at the edges.
- The plausibility of B as the next move is
- f(B) = g(B) + h(B) = 1 + 2.4 = 3.4
- Where g(B) is the cost of going from A to B, and h(B) is the cost to get from B to the goal E.

A Search Algorithm (Cont'd)

- The plausibility of C is
- f(C) = g(C) + h(C) = 1 + 1 = 2.0
- Since f(B) > f(C), the path should go from A to C.
- But this assumes that h(n) was known at every node. This meant that the algorithm had to recurse in order to find the correct value of h(n). This leads to visiting all the nodes.

A* Search Algorithm

- A search is guaranteed to produce the optimal path because it compares all possible paths to each other.
- A* search takes an interesting approach to reducing the amount of paths that have to be generated and compared: it compares the possible paths to the best possible path, even if there isn't a path in the real world that goes that way.
- The algorithm estimates h rather than checks to see if there is actually a path segment that can get to the goal in that distance.
- The estimate can then be used to determine which nodes are the most promising, and which paths have no chance of reaching the goal better than other candidates and should be pruned from the search.

A* Search Algorithm (Cont'd)

- Under A* the evaluation function becomes

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

- Where the * means that the functions are estimates of the values that would have been plugged into the A search evaluation.

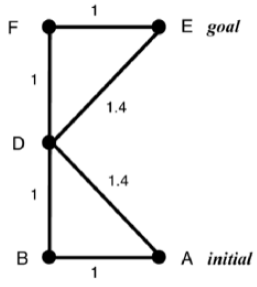
A* Search Algorithm (Cont'd)

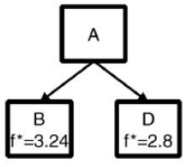
- In path planning, g*(n) is the same as g(n): the cost of getting from the initial node to n, which is know through the incremental build of the path. h*(n) is the real difference.
- How can we be sure that the estimate will be accurate enough that we don't end up choosing a path which isn't truly optimal?
- This can be done by making sure that h(n) will never be smaller than h*(n).
- The restriction that h*(n) <= h(n) is called the admissibility condition.
- Since h*(n) is an estimate, it is also called a heuristic function, since it uses a rule of thumb to decide which is the best node to consider.

A* Search Algorithm (Cont'd)

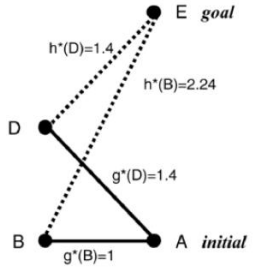
- Fortunately, there is a natural heuristic function for estimating the cost from n to the goal: the Euclidean (straight line) distance.
- Recall that the
- locations of each node are known independently of the edges. Therefore, it is straightforward to compute the straight line distance between two nodes.
- The straight line distance is always the shortest path between two points, barring curvature of the earth, etc.
- Since the real path can never be shorter than that, the admissibility condition is satisfied.

A* Search Algorithm (Cont'd)





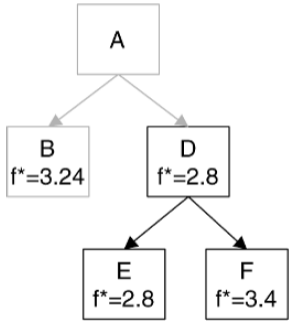
a.

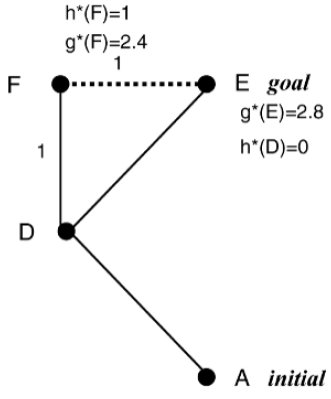


b.

$$f^*(B) = g^*(B) + h^*(B) = 1 + 2.24 = 3.24$$
$$f^*(D) = g^*(D) + h^*(D) = 1.4 + 1.4 = 2.8$$

A* Search Algorithm (Cont'd)

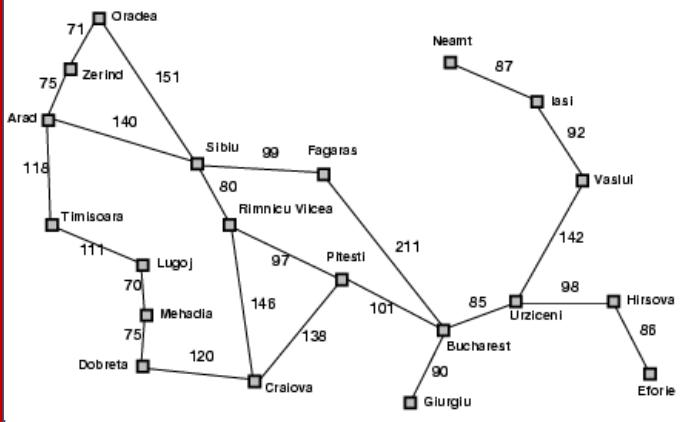




d.

$$f^*(E) = g^*(E) + h^*(E) = 2.8 + 0 = 2.8$$
$$f^*(F) = g^*(F) + h^*(F) = 2.4 + 1.0 = 3.4$$

A* using Romania Example (Russell & Norvig)

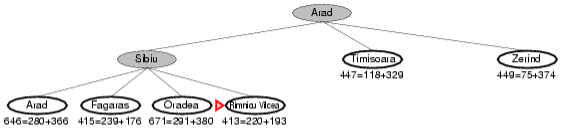


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

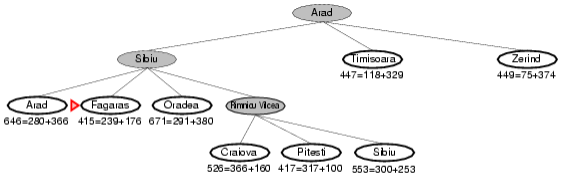
A* search example

A* search example

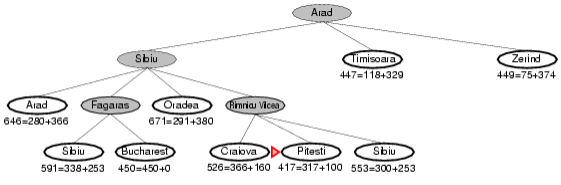
A* search example



A* search example



A* search example



A* search example