The Drunken Sailor’s Challenge

CS7001 mini-project by Arya Irani
with Tucker Balch

September 29, 2003
Introduction

The idea is to design an agent to navigate a two-dimensional field of obstacles given a compass that points very roughly towards the direction of the goal, and information about any obstacles that may be immediately adjacent to the agent. Specifically, the field is an 80x23 unit grid and for each step the agent is provided an approximate heading to the goal: one of eight compass directions (northwest, north, northeast, east, etc.), and an array specifying, for each of those eight compass directions, whether or not an obstacle or the goal itself is immediately adjacent to the agent. Armed with this information, the agent chooses one of the eight directions to move in. Moving north, south, east, or west is a distance of one unit; moving northwest, northeast, southwest, or southeast is a distance of $\sqrt{2}$ units.

In mobile systems, computing resources such as processor time and memory tend to be at a premium, so it’s important to use them efficiently. On the other hand, if an agent that uses a more complex algorithm and has a larger memory footprint results in a significant savings in terms of the number of steps (moving generally requires energy) or the amount of time required to reach the goal, that’s something to consider too.

A* Search with Bitmap-based Goal Estimation

The first challenge was to implement an agent which would always take the optimal path to the goal, given currently and previously available information.
In this implementation, the map is represented internally as a 160x46 array, twice the size in each direction as the field the agent is navigating. The doubled size is due to the fact that agent doesn’t know its location with respect to the field and so it should be able to plan to move an entire field-length in any direction. An N x M field would require a 2N x 2M array, so that’s something that should be considered in selecting an algorithm to be used in a real situation.

**Goal Estimation**

A* is a popular algorithm for finding the optimal path between a given start node and goal node in a connected graph, but it isn’t very useful if you don’t know which node is the goal node. (In this implementation, each unoccupied unit on the grid is considered a distinct node. Each node is connected bidirectionally to each of its eight grid neighbors, provided they are not occupied by some obstacle.)

The goal node is estimated as the average position of all nodes which have **not** been proven to **not** be the goal. Before the agent is given its first approximate heading, **any** of the nodes could potentially be the goal. When the agent is first given an approximate heading toward the goal, it can disqualify all the nodes which aren’t in that general direction as being potential goals. It considers all nodes within 45 degrees in either direction of the approximate goal heading (90 degrees total) to be in the general direction of the approximate goal heading. With each subsequent step, it can further narrow the set of potential goals nodes by taking the intersection of the previous set of potential goal nodes and the set of nodes in the general direction of the goal heading, and recalculate the average position of the remaining potential goal nodes to arrive at a new goal node estimate. In Figure 1, the red square represents the agent on its own internal
map, the yellow area represents the set of potential goal nodes, and the green square marks the average of the potential goal nodes. The goal heading here is pointing northwest, so everything from west to north is potentially the goal node. In Figure 2, the agent has moved a little ways westward and the goal heading has changed to point northward. The blue area represents nodes that are in the general direction of the goal heading, but which are no longer considered potential goal nodes, they were disqualified at or prior to the time shown in Figure 1.

![Figure 1](image1.png)  ![Figure 2](image2.png)

**A* Heuristic**

If you don’t know what the A* is, there’s a good article[^2] about it at ai-depot.com, which I referred to in implementing A* for this project. One of the important issue with A* is that for each node you need to be able to estimate the remaining distance to the goal. Furthermore, your estimate must be less than or equal to the actual distance along the graph to the goal from the node in question; if your estimate is an overestimate, there is a possibility that the path A* finds to the goal is not the best one.

That being said, in this agent’s implementation, the estimate is the straight-line distance to the average potential goal node, but the search stops as soon as any potential goal node is encountered along the path to the average goal node. However, I would argue that optimality won’t be undermined in this case because a) that nearest potential goal node may be the actual goal node, and b) it’s the nearest potential goal node so it’s
on the way anyway, and the agent is only taking one step at a time and our average estimated goal is being revised with each step. The advantage of stopping early is we have a pretty good approximation for the next step, and you’ve only had to search a smaller graph.

Notes

This considers all nodes within 45 degrees in either direction of the approximate goal heading (90 degrees total) to be in the general direction of the approximate goal heading. Earlier drafts of this implementation considered all nodes within 22.5 degrees in either direction of the approximate goal heading to be in the general direction of the approximate goal heading (45 degrees total). It turned out the be the case that 22.5 on either side was too restrictive, possibly due to differences in the methods by which the angles are calculated within the agent and within the framework providing the direction to the agent. Sometimes the intersection of the previous set of potential goal nodes and the set of nodes in the direction of the approximate goal heading was the empty set, and there was no goal at all. A* does not perform well under these circumstances.

Video visualizations of this algorithm in operation will be available at http://home.cc.gatech.edu/arya/9.

A less-complex agent

The second implementation was an attempt to design an agent which still always be able to reach the goal when a path exists, but using a constant amount of memory (not related to the size of the world) at the expense of a potentially having to travel a longer path.
The Algorithm

The agent starts by trying to move in the direction of the approximate goal heading. It continues to move in the direction of the approximate goal heading until it encounters an obstacle. When it encounters an obstacle, it stores the approximate ship heading at the time it encountered an obstacle. Then it makes a random choice as to whether to try to circumnavigate the obstacle by moving clockwise around the obstacle or counterclockwise. While it is trying to circumnavigate the obstacle, it tracks its current position with respect to its position at the time it originally encountered the obstacle. (Note that the agent does not need to know its actual location at the time it encountered the obstacle, only the relative direction it’s traveled since then. It continues to move in a circle hugging the edge of the barrier until it finds itself located in the general direction of the goal heading at the time of encountering the obstacle with respect to its position at the time of encountering the obstacle. Simple, right? See Figure 3.
Notes

Earlier drafts of this implementation always either tried to circumnavigate an obstacle by moving clockwise, or always by moving counterclockwise. Sometimes this would lead to an infinitely long, looping path. It was due to this problem that the decision was made to randomly decide whether to move clockwise or move counterclockwise upon reaching each obstacle. The hope is that one of the random decisions will eventually allow the agent to progress towards the goal, and this has been the case with the test maps used so far.

Video visualizations of this algorithm in operation will be available at http://home.cc.gatech.edu/arya/9.

Future work

With more time I would like to try a few other designs. For example an agent that used A* for search but represented obstacles with vectors instead of a bitmap might provide an optimal path with lower memory requirements.

Video visualizations of the A* with bitmap-based goal estimation algorithm in action are available at http://home.cc.gatech.edu/arya/9. Because the second algorithm does not have an internal representation of the world map, it is less trivial to generate these videos. However, they would be helpful in understanding the algorithm, and will be available at the same URL soon.
References