Name: Chakradhar Reddy Punur
NetID: crp190

PROJECT 1: WHERE ARE YOU?

1) Optimality Strategy

Design Choices and Justification:

The Optimal Strategy models localization as a belief-space search problem, where each state represents all possible bot locations on the ship grid. Finding the quickest series of actions that lowers this belief to a single cell which is the bot's actual position is the aim.

Because A* search ensures the smallest number of movements when directed by an admissible heuristic, I utilized it to investigate the belief states.

Heuristic Design:

The heuristic estimates the spread of possible positions:

h(L) = (max(x) - min(x)) + (max(y) - min(y))

This is the Manhattan distance between the farthest possible cells, representing the minimum steps required to converge the bot's uncertain positions into a single location.

Admissible: It guarantees that A* finds the best answer by never overestimating the actual number of steps required to collapse all options.

Consistent: The optimality guarantee of A* is maintained when each movement lowers or maintains the heuristic value.

Distance Precomputation:

All shortest path distances between open cells are precomputed using BFS in order to further minimize performance.

A lookup table called dist_from_to[(start, end)] contains these, allowing constant-time access to distances while searching and significantly lowering the cost of repeated computations.

Why It's Optimal:

f = g + h, where g is the number of moves made thus far and h is the lower constraint on the amount of uncertainty left, is the non-decreasing order in which states are expanded by A*.

Since the heuristic is consistent and admissible, the globally optimal solution with the fewest movements is always the first belief state to reach a single point.

Performance and Limitations:

The A*-based optimal technique works well for ship sizes up to D = 10, achieving results in roughly 1.1 seconds with 20 moves to localize the bot. But when the ship size goes above D = 10, the calculation becomes very memory-intensive and slow.

This is due to the exponential growth of the state space; every belief state in A* denotes a potential set of bot locations. The program will run out of memory and take an infinite amount of time for larger grids since there are an infinite number of such combinations.

Despite its ability to minimize the number of calculations, the precomputed distance table is unable to counteract the combinatorial explosion of conceivable states.

2) Efficiency Strategy

In order to (heuristically) localize the bot to a single location on the ship, the efficiency strategy rapidly computes a series of motions, putting computational speed ahead of the solution's optimality or even guaranteed accuracy.

Essentially, the bot's potential positions are modeled as a probabilistic belief distribution over the ship's open cells. This distribution is then repeatedly updated by basic, repetitive movements until the belief focuses on a single cell.

a) Probabilistic Belief Representation:

A consistent probability distribution is used to initialize the ship's open cells. For effective manipulation, this is kept in a numPy array that matches the ship's grid shape. The belief is "shifted" in the desired direction following each move attempt using np.roll with the entire array normalized to add to 1 and the probabilities in blocked cells set to zero.

Without explicitly listing sets of locations (as in the set-based L of the baseline technique), it effectively tracks every conceivable bot position. We can measure uncertainty and move toward a single high-probability cell by employing probabilities. Vectorized operations are made possible by NumPy arrays, which speed up updates even for massive grids.

Sacrifices for speed:

Unlike theexact set-based approach ofthe baseline's move function, which precisely tracks discrete positions, this probabilistic model introduces approximation errors. For instance, normalization after shifts can lead to floating-point drift, and low-probability cells might be prematurely diminished, potentially localizing to a suboptimal or incorrect cell if the sequence isn't perfect.

Array operations are O(D^2) per update (where D is ship size), but highly optimized in NumPy, allowing sub-second computation even for

large D. This is much faster than BFS in the baseline, which could be O(D^4) in worst-case pathfinding over all positions.

b) Cyclic Movement Pattern:

Moves are selected in a fixed, repeating cycle: UP, DOWN, LEFT, RIGHT. This is a simple round-robin without any adaptive decision-making based on the current belief state.

The cycle ensures exploration in all directions, simulating a "wall-following" or oscillatory behavior that bounces the bot (and its belief) around the ship.

Sacrifices for Speed:

This is highly suboptimal in terms of move count—the sequence is often much longer than necessary. It doesn't plan ahead or choose moves to maximize information gain.

Unlike the baseline (which computes shortest paths dynamically) or the optimal strategy (which uses A* with heuristics), this ignores the current belief to pick the "best" next move. I sacrificed intelligent exploration for simplicity.

Move selection is O(1) per step, avoiding any search or optimization loops. This makes the entire algorithm run in O(steps * D^2), where steps is bounded, enabling it to scale to very large ships upto sizes of 100 in seconds.

c) Artificial Probability Boosting for Convergence:

After each move, I identified the cell with the current maximum belief, added a fixed boost (0.05) to it, and renormalized it. This stops when any cell's belief exceeds 0.9.

Natural belief updates via shifts can be slow to converge in large or loopy ships, as probabilities diffuse evenly. The boost acts as a greedy

"attraction" mechanism, accelerating concentration toward the most likely cell (often a dead end or corner). The 0.9 threshold balances early stopping with reasonable confidence, and 0.05 was empirically tuned to avoid over-boosting too quickly.

Sacrifices for Speed:

It's not probabilistically sound and can force incorrect localization. It might be possible that two cells can have similar probabilities and this boosting method can favor one cell which could result in an inaccurate location.It sacrifices accuracy and optimality for forced termination.

Boosting is O(D^2) for finding the max, but it's trivial compared to planning. It ensures the loop terminates quickly, preventing infinite loops in hard cases.

3) Performance of Optimality Strategy

The plot shows that for small ships (D ≤ 10), the optimal strategy achieves fewer moves when compared to the baseline strategy, confirming its correctness and optimality. As the ship size increases up to D = 10, the baseline strategy requires up to ~70 moves, while the optimal strategy localizes in about 20 moves on average. This shows that A* optimal strategy works only for small ships.

However, beyond D = 10, the A* approach becomes computationally infeasible, as it runs out of memory due to the exponential growth in the state space.

4) Performance of Efficiency Strategy

The runtime graph shows that the baseline strategy’s runtime grows rapidly with increasing ship size, due to repeated random exploration and lack of informed guidance.

The efficient strategy scales much better because its runtime grows slowly and remains stable even as D increases, since it uses probabilistic updates to focus search on likely target regions.

Both graphs show that the efficient strategy achieves localization at a much faster time when compared to the baseline strategy as the ship size increases.

5) Question 5

Algorithmic Approach:

a) Optimal Strategy (A* Search):

In this strategy, A* search is used over all the possible bot locations.

Each move (UP, DOWN, LEFT, RIGHT) transitions the state set.

The Manhattan distance heuristic estimates how far we are from full localization (i.e., one possible position).

This guarantees an optimal sequence of moves but quickly becomes infeasible as the ship size increases because the number of possible sets grows exponentially with the grid size.

b) Efficient Strategy (Probabilistic Localization):

Instead of searching through all possible sets, this strategy maintains a belief distribution which is a probability assigned to each open cell representing where the bot might be.

It updates this belief after each move using the environment structure.

The movement decisions are made greedily to reduce entropy (uncertainty) in the belief.

This makes it much faster, scaling efficiently even for ships as large as 100, though it does not guarantee optimal moves.

Trends:

For small ships (D ≤ 15), the A* strategy finds the fewest possible moves and demonstrates correctness.

As D increases, A* quickly becomes intractable, requiring excessive memory and computation time due to the combinatorial explosion in possible location sets.

The Efficient strategy, however, remains stable and fast even for large D, though it sometimes requires slightly more moves to localize the bot.

Why This Problem Is Hard:

a) **Optimality Hardness:**

The quantity of potential belief states increases exponentially with the size of the grid. Each open cell may represent a potential bot position or may not.

Determining the shortest series of actions that distinctly identifies the bot's position requires navigating a huge state space, making it costly in terms of computation.

b) **Efficiency Hardness:**

Even efficiently estimating localization is difficult because movements can produce non-deterministic outcomes which is dependent on obstacles.

The strategy must deduce global positioning data from local actions without explicit feedback.

For expansive grids, it is costly to uphold or revise the belief distribution.