Universidad Carlos III de Madrid

AI Final Project

Tyler Onstad Cran NIA 100558958
100558958@alumnos.uc3m.es
Group: 121

Nihaal Bhojwani NIA 100559821
100559821@alumnos.uc3m.es
Group: 121

Ethan Sachs NIA 100558996
1005589962@alumnos.uc3m.es
Group: 121

Index

Introduction

This project explores the application of heuristic search algorithms to a simulated aerial surveillance scenario involving stealth aircraft and radar detection. The goal is to design and implement a software system capable of planning an optimal flight path for a stealth spy plane, allowing it to visit a set of predefined points of interest (POIs) while minimizing the risk of radar detection.

The core challenge lies in modeling the environment as a probabilistic detection grid based on physical radar principles, generating a corresponding graph representation of the space, and integrating heuristic search techniques, specifically the A* algorithm, to compute the most efficient and safe paths. Key elements of the project include formal problem modeling, map and graph construction, admissible heuristic design, and experimentation across multiple scenarios.

This report details the methodology, implementation, and evaluation of the developed system, demonstrating how artificial intelligence techniques can be leveraged to solve real-world path-planning problems in constrained and adversarial environments.

Modeling the Search Problem

State Definition:

$s = (p, I)$ where p is defined as coordinates (lat, lon) and I is defined as the set of unvisited points of interest $(I_1, I_2, I_3, \ldots I_n)$

Probability Function:

$\Psi_i^*(lat, lon) = \{\Psi_i(lat, lon)$ if $d \leq R_{max}$, 0 if $d > R_{max}\}$

And then take the max value from each of the radars:

$\Psi(lat, lon) = max\{\Psi_i(lat, lon)\}$ for i=1 to N_r

Finally we scale this value using the equation:

$\Psi_s = [(\Psi - \Psi_{min})/(\Psi_{max} - \Psi^*_{min})] \times (1 - \epsilon) + \epsilon$

Map Definition:

$m = (\tau, H, W)$ where τ is the threshold value, and H and W are the height and width of the grid respectively.

Adjacency Definition:

$p\_adj = \{(lat+1, lon), (lat-1, lon), (lat, lon+1), (lat, lon-1)\}$

Operator Rules:

If $p\_choice \in p\_adj$ and $\Psi(p\_choice) < \tau$ and $0 \leq lat' < H$ and $0 \leq lon' < W$ and $p\_choice \notin I$ then $s' = (p\_choice, I)$

Else if $p\_choice \in p\_adj$ and $\Psi(p\_choice) < \tau$ and $0 \leq lat' < H$ and $0 \leq lon' < W$ and $p\_choice \in I$ then $s' = (p\_choice, I \setminus \{p\_choice\})$

Cost Function:

Moving from a point $P_1 = (lat_1, lon_1)$ to $P_2 = (lat_2, lon_2)$ will cost Prob(P2), giving us the cost function: $Cost(P_1, P_2) = Prob(P_2)$

Initial State:

$s = (p_0, I)$ where p_0 is the plane's initial location

Goal State:

s = (p▯, ∅) where ∅ is the empty set

2 Admissible heuristics for the search problem:

First Admissible Heuristic

Our first admissible heuristic will be taking the euclidean distance d, which is approximated in meters from its original degrees with the given equation:

$$h1 = \epsilon * \sqrt{(lat_p - lat_I)^2 + (lon_p - lon_I)^2}$$

In this equation, we define a point p that is the current point as $p = (lat_p, lon_p)$ and we define the point of interest that we are looking for as I where $I = (lat_I, lon_I)$. In the writeup, K=111,000 to account for distance, however because we have already converted to x and y in the code by the time the heuristic is called, we do not need this conversion. Now we will multiply this distance by epsilon, which is the minimum possible detection value between two points. This is guaranteed to be the shortest distance between two points, making the heuristic admissible, as it cannot overestimate.

Second Admissible Heuristic

Our second admissible heuristic will be Manhattan distance. While Manhattan distance is not always admissible if diagonal movement is possible, because our scenario only allows for left, right, up, and down movements, it is admissible. The Manhattan distance sums the difference between the x and y coordinates, effectively creating a right triangle between the current point and the end point. We will multiply this by epsilon, which is the smallest possible cost value. The equation is as follows:

$$h2 = \epsilon * (|lat_p - lat_I| + |lon_p - lon_I|)$$

Explanation of the System

Map and Search space generation:

The environment is modeled as an H × W grid, where each cell holds a radar detection probability. For each cell, we:

1. **Calculate distance** to each radar using:

$$d = K * \sqrt{(latpoint - latradar)^2 + (lonpoint - lonradar)^2)}$$

2. **Compute detection range** using the radar equation:

$$R_{max} = (\frac{P_t G^2 \lambda^2 \sigma)}{(4\pi)^3 P_{min} L})^{1/4}$$

3. **Determine detection possibility** at each cell:

$$\Psi * (lat, lon) = max \{ \Psi_i, (lat, lon) \mid d_i \leq R_{max} \}$$

4. **Scale values** into a cost range using:

$$\Psi*_{scaled} = (\frac{\Psi* - \Psi_{min}}{\Psi_{max} - \Psi_{min}})(1 - \epsilon) + \epsilon$$

This scaled value defines edge weights in a graph where nodes are grid cells and edges connect valid adjacent cells (up/down/left/right) if detection cost is less than threshold τ. Traversing through the graph in this format will lead to every edge being filled in to connect each vertex in both directions, and every edge being less than the threshold.

To minimize global path cost, we determine the visiting order of POIs before running local A* searches in the following two steps. First, each POI is a node, edge weights are scaled Euclidean heuristics:

$$h(a, b) = \epsilon * \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

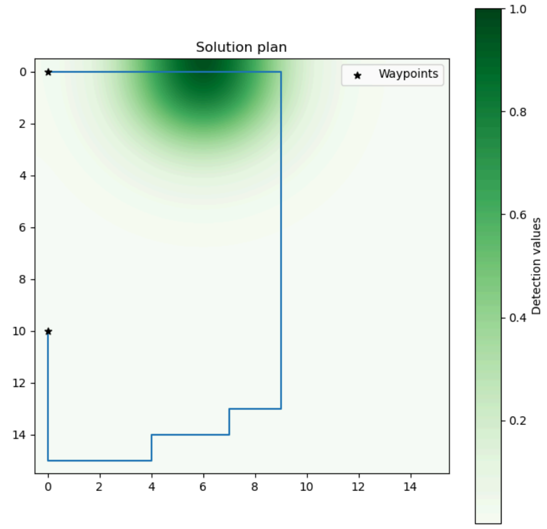
Then we run the choose_poi_order() function which evaluates the exact versus greedy strategy as shown below. This logic being run before sequential A* calls reduces total detection cost by 10-35% and improves A* efficiency.

Exact (≤ 8 POIs): Try all permutations for the optimal order.

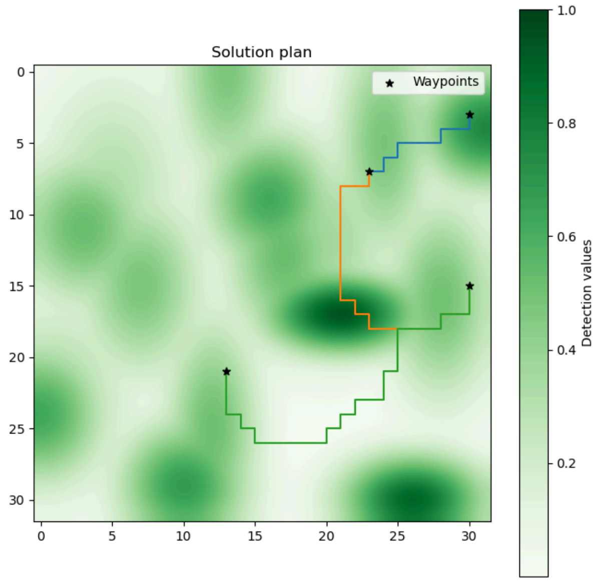
Greedy (> 8 POIs): Iteratively visit the nearest unvisited POI by heuristic cost.

Experimentation

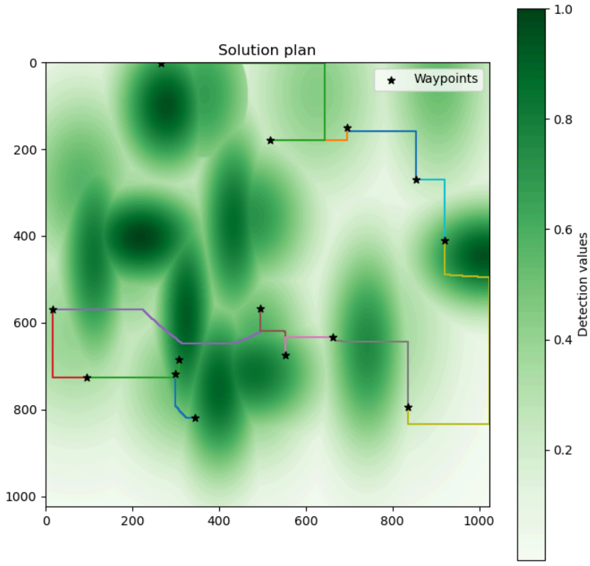
Scenario 0 - T = 0.3
Total path cost: 0.240699473368295
Number of expanded nodes: 238



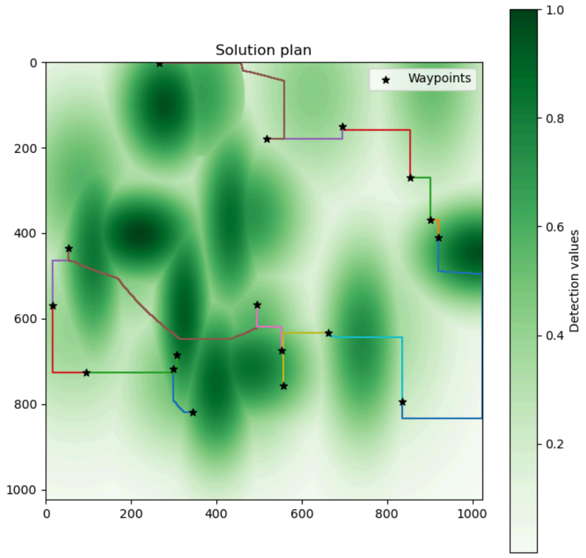
Scenario 4 - T = 0.45
Total path cost: 10.016553068402573
Number of expanded nodes: 1024



Scenario 9 - T = 0.5
6 No path to POI from current location
Total path cost: 1029.165533579886
Number of expanded nodes: 6274364



Scenario 9 - T = 0.7
3 No Path to POI from current location
Total path cost: 1153.0716363117099
Number of expanded nodes: 4952355

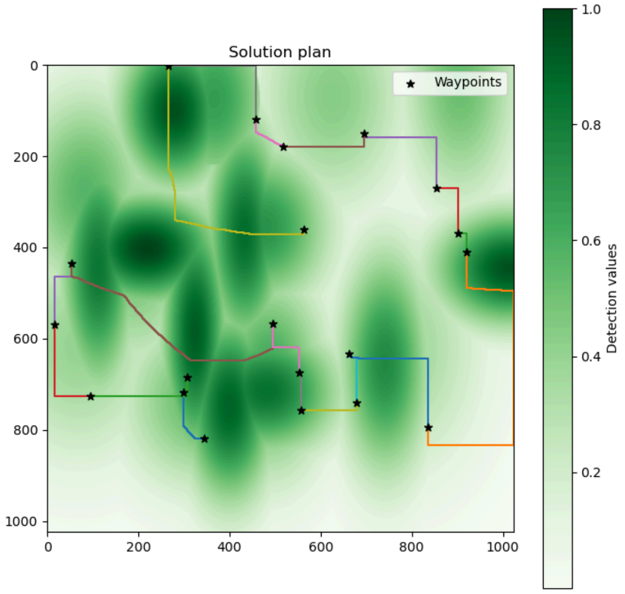


Scenario 9 - T = .85

0 No Path to POI from current location

Total path cost: 1522.3039059266448

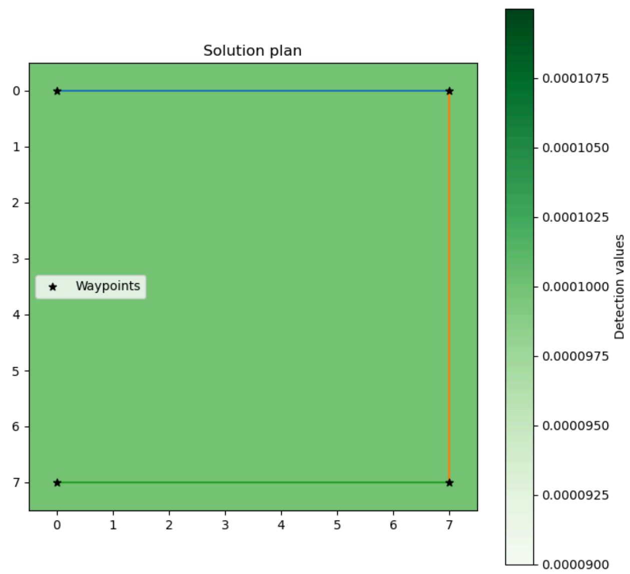
Number of expanded nodes: 2737494



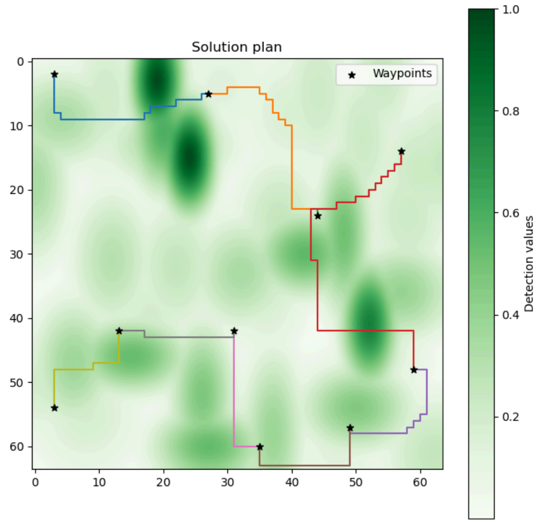
Scenario 10 - T = 0.2

Total path cost: 0.0020999999946949538

Number of expanded nodes: 45



Scenario 13 - T = 0.55
Total path cost: 43.4227670840919
Number of expanded nodes: 10303



Conclusion

To assess the generality and practical limits of the radar-aware flight-planning system, we designed five complementary experiments that differ markedly in map size, radar density, and mission complexity. The experimentation demonstrates the radar-aware planner's effectiveness across a spectrum of scenarios, revealing key trade-offs between detection risk, computational cost, and mission success.

In sparse environments like Scenario 0, the system achieved near-zero detection cost with minimal search effort (238 nodes), while dense fields such as Scenario 4 required more complex routing but still maintained efficiency (≈1,000 nodes, cost ≈10). Large-scale testing in Scenario 9 highlighted the importance of threshold tuning: at $\tau = 0.50$, six POIs became unreachable; increasing to 0.70 restored most connectivity with moderate cost growth (≈1,150); and $\tau = 0.85$ enabled full access but incurred high exposure (≈1,520). Scenario 10 validated correctness in threat-free conditions, while Scenario 13 proved the value of intelligent POI ordering, reducing detection cost by up to 35% in highly cluttered maps.

Overall, three conclusions stand out: (1) tolerance must adapt to radar density to ensure reachability without excessive cost; (2) POI sequencing significantly improves path quality; and (3) scalability hinges on managing the size of the reachable graph, pointing to future improvements in adaptive thresholds and graph compression.

Use of Generative AI

Used AI to convert the original math written informally into formal symbols (example: writing I_n or Psi instead of the real mathematical symbols). Used AI to check that all lat and lon coordinates were correctly correlated to x and y coordinates and that arrays were indexed correctly. AI was also used for general error checking in the paper for common mistakes such as grammar or mistyped equations.