Bachelor in Computer Science and Engineering

Artificial Intelligence 2024-2025
Grupo 89

Final practice

“Heuristic Search in Radars”

Jorge Adrian Saghin Dudulea – 100522257
Denis Loren Moldovan – 100522240
Ignacio Cortina de Antonio – 100522372

Índice

I Introduction **2**

1. Explanation of the problem 2

2. Explanation of the approach 2

II Explanation of the system **3**

1. Mathematical modelling of the search problem 3

1.1. Initial and Goal States 4

1.2. Heuristics designed 4

1.3. Map and search space generation 5

1.4. Search graph generation 6

2. Logical modelling of the system 7

III Experiments **9**

IV Use of AI **10**

1. Where AI Was Used 10

2. How AI Facilitated the Project 10

3. Verification Corrections 10

V Conclusion **12**

1. Mathematical modelling: 12

2. Technical implementation: 14

3. Testing: 14

4. Challenges 14

Parte I

Introduction

1. Explanation of the problem

In the context of the actual aerial surveillance and military operations, the stealth technology has become a key element to successfully complete high risk missions, minimizing detection probability. Despite having advanced radar systems, they still lack some features that can be exploited through the correct strategy. This project faces the problem of the planification of certain routes for our spying aircraft, whose main objective is to fly over a series of critical zones without being detected by the ground radars in the region.

The main goal consists on designing and implementing a software able to generate a probabilistic map from the physical characteristics of multiple radars, using as a starting point the radar equation. With this map and thanks to heuristic search algorithms, the system needs to calculate the best route allowing the stealth plane visit a series of predefined points of interest (POIs), reducing as much as possible the high risk areas, where radars are located.

For modeling the problem, a bidimensional grid explains the geographical area of interest, assigning at each cell some specific value, which represents the possibility of being detected by one or multiple radars. This possibility is obtained through a gaussian distribution, then, it is normalized in order to obtain probabilities. The resulting map gets transformed into a directed and weighted graph in where every node represents a valid cell and each of the edges an allowed movement between adjacent cells, the weight of the graph is equivalent to the detection cost of the goal cell,

The final system integrates the detection map generation, the construction of the graph and the application of algorithms for the route scheduling. The solution is validated by different experimental scenarios with varying radar distributions and parameters.

2. Explanation of the approach

The different scenarios and distributions of the map have been tested with the pycharm tool, pybuilder, which has helped us thanks to its modular and declarative results, another pycharm tool used has been pylint, which has been very useful in the structuration of the code, ensuring compliance with coding standards and proper code structuration. For the collective code manipulation, a private version control (gitea) locally hosted, has been used, in order to ease the code sharing, updating and such.

Parte II

Explanation of the system

1. Mathematical modelling of the search problem

1.0.1. State Space:

Let the state space S be defined as:

$$S = \{(y,x) \in \mathbb{Z}^2 | 0 \leq y < H, 0 \leq x < W, \Psi^*(y,x) \leq \tau\}$$

where:

- $H \times W$: Dimensions of the grid
- $\Psi^* : \mathbb{Z}^2 \to [\varepsilon, 1]$: Scaled detection probability function
- $\tau \in (0,1]$: Detection tolerance threshold

1.0.2. Operator Set:

The action set A defines valid transitions between states:

$$A = \{Up, Down, Left, Right\}$$

Each action $a \in A$ maps to a movement vector:

$$\delta(a) = \begin{cases} (-1,0) & \text{if } a = Up \\ (1,0) & \text{if } a = Down \\ (0,-1) & \text{if } a = Left \\ (0,1) & \text{if } a = Right \end{cases}$$

1.0.3. Transaction Function:

$$T : S \times A \to \mathrm{S} \cup \{\emptyset\}$$

$$T((y,x),a) = \begin{cases} (y',x') & \text{if } (y',x') \in S \text{ where } (y',x') = (y,x) + \delta(a) \\ \emptyset & \text{otherwise} \end{cases}$$

1.0.4. Cost function:

$$c((y_1,x_1),(y_2,x_2) = \Psi^*(y_2,x_2)$$

1.1. Initial and Goal States

1.1.1. Initial State:

$$s_0 = (y_0,x_0) \text{ where } (lat_0,lon_0) \mapsto (y_0,x_0)$$

1.1.2. Goal State:

For a sequence of POIs $\{p_1, ..., p_k\}$:

$$s_{goal} = \{(y_i,x_i)\}_{i=1}^{k} \text{ where each } p_i \mapsto (y_i,x_i)$$

1.1.3. Solution Characteristics:

A solution is a path $\pi = [s_0,s_1, ...,s_n]$ where:

1. $s_n \in s_{goal}$
2. $\forall i, \exists a \in A$ such that $s_{i+1} = T(s_i,a)$
3. Total costs $C(\pi) = \sum_{i=0}^{n-1} c(s_i,s_{i+1})$ is minimized

1.2. Heuristics designed

1.2.1. Euclidean distance:

$$h_1((y,x),(y_{goal},x_{goal})) = \sqrt{(y-y_{goal})^2+(x-x_{goal})^2} \cdot \varepsilon$$

Admissibility Proof:

1. Actual path cost between adjacent cells $\geq \varepsilon$
2. Straight-line distance is the minimum possible path length
3. Thus $h_1 \leq$ actual cost (underestimates)

1.2.2. Manhattan distance:

$$h_2((y,x),(y_{goal},x_{goal})) = (|y-y_{goal}|+|x-x_{goal}|) \cdot \varepsilon$$

Admissibility Proof:

1. Manhattan distance $\geq$ Euclidean distance
2. Each move costs $\geq \varepsilon$
3. Therefore $h_2 \leq$ actual cost (underestimates)

1.3. Map and search space generation

1.3.1. Implementation of the map

Let the rectangular area delimited by geodetic coordinates $(\text{lat}_0, \text{long}_0)$ $(\text{lat}_1, \text{long}_1)$ be devided into a grid $H \times W$. Each cell represents a unique point in where

- The latitude and longitude intervals are calculated as

$$\Delta\, lat = \frac{lat_0 - lat_1}{H-1} \quad \Delta\, long = \frac{long_0 - long_1}{W-1}$$

- Each cell (i, j) is mapped to a geographic coordinates:

$$lat_i = lat_1 + i \cdot \Delta lat, lon_j = long_0 + j \cdot \Delta lon$$

For each radar $r_k \in \{1, 2, ..., N_r\}$, located in the coordinates (lat_k, lon_k), we compute:

- The maximum detection range (R_{max}) using the provided rada equation.

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

- The Euclidean distance (d) from each grid cell to the radar using the conversion factor $K = 111{,}000$ (meters per degree)

$$d = K \cdot \sqrt{(lat_i - lat_k)^2 + (lon_1 - lon_k)^2}$$

- If $d \leq R$, compute the detection possibility using the 2D Gaussian function

$$\Psi_k^*(i,j) = \frac{e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}$$

Where

- $x = (lat_i, lon_j)$
- $\mu = (lat_k, lon_k)$
- Σ is the covariance matrix, typically diagonal (can be fixed for simplification)

- if $d > R_{max}$, set $\Psi_k^*(i,j) = 0$
- Then for each cell, compute the maximum detection possibility from all radars:

$$\Psi^*(i,j) = max_k \Psi_k^*(i,j)$$

The raw detection possibilities $\Psi^*(i,j)$ are scaled to the interval $[\varepsilon, 1]$ using the modified Min-Max normalization:

$$\Psi^*_{\text{scaled}}(i,j) = \left(\frac{\Psi^*(i,j) - \Psi^*_{\text{mín}}}{\Psi^*_{\text{máx}} - \Psi^*_{\text{mín}}}\right) \cdot (1-\varepsilon) + \varepsilon$$

where:

- $\Psi_{\text{mín}}$ and $\Psi_{\text{máx}}$ are the minimum and maximum non-zero detection values in the map.
- ε is a small positive constant to ensure no cell has cost 0 (e.g., $\varepsilon = 0{,}001$)

The resulting detection map is stored in memory as a matrix:

$$M \in \mathbb{R}^{H \times W}, \quad M[i][j] = \Psi^*_{\text{scaled}}(i,j)$$

1.4. Search graph generation

Given the detection map M, we construct a directed weighted graph $G = (V, E)$, where:

1.4.1. Vertices

Each vertex $v_{i,j} \in V$ corresponds to a cell (i, j) such that:

$$M[i][j] \leq \text{detection threshold}$$

Only these vertices are considered traversable by the spy plane.

1.4.2. Edges

An edge exists between vertex $v_{i,j}$ and its valid neighbors $v_{i',j'}$ if:

$$(i', j') \in \{(i+1, j), (i-1, j), (i, j+1), (i, j-1)\}$$

and both cells satisfy:

$$M[i][j] \leq \text{threshold}, \quad M[i'][j'] \leq \text{threshold}$$

For each such edge:

$$e_{(i,j)\to(i',j')} \in E \quad \text{with weight} \quad w = M[i'][j']$$

That is, the cost of moving from a cell to a neighbor is the detection cost of the **destination** cell, reflecting the risk of exposure.

2. Logical modelling of the system

As mentioned in the introduction, the project was composed to fit the pybuilder integration and pylint code smells detection, and the folder structure is available in the README.md, along with every other aspect on how to run the source code, bundled inside the zip file.

As for the intial code, the map representation was modified to represent a more realistic “heat map” of the radar detection boundaries, and the path, indicating the POIs and the end of the path with different colors.

Moreover, a map cache has been implemented to avoid recomputing the whole map over and over again if the tests are run multiple times with different tolerances or heuristics. They have been integrated by storing the hash key of the whole data as file name and storing them inside a folder in the component’s module.

In addition, when the tests are ran, the maps are computed but not shown to avoid cluttering on the user screen.

On the other hand, when the code detected that a POI was inside a no-fly zone of a certain radar, the program raises an error and advises the user to increase the tolerance or change the respective POI. Other possible implementation was to ignore completely the POI and compute the path to the rest of possible POIs, but we thought that a real mission cannot be completed if the whole set of points cannot be visited.

Parte III

Experiments

The following test cases that are implemented and tested using the pybuilder module, where created taking into account all the probable cases where the program could fail, safeguarding from incorrect input and raising the appropiate errors.

id	description	input	expected_error
TC-001	Tolerance set to 0 (impossible)	`scenario_0` `0.0`	`ValueError: Tolerance must be greater than 1e-4`
TC-002	Negative tolerance value	`scenario_1` `-0.1`	`ValueError: Tolerance must be greater than 1e-4`
TC-003	Tolerance > 1.0	`scenario_2` `1.1`	`ValueError: Tolerance must be between 1e-4 and 1`
TC-004	Non-existent scenario	`scenario_99` `0.5`	`KeyError: Scenario 'scenario_99' not found`
TC-005	Missing tolerance argument	`scenario_3`	`TypeError: Tolerance must be numeric`
TC-007	Isolated POI (no connecting path)	`scenario_5` `0.3`	`RuntimeError: Pathfinding aborted due to invalid path segment`
TC-008	Malformed coordinate input	`scenario_0_corrupt` `0.5`	`ValueError: Invalid POI coordinates`
TC-009	All POIs in high-detection areas	`scenario_6` `0.001`	`RuntimeError: Empty graph - all nodes exceed tolerance`
TC-010	Single POI (no path needed)	`scenario_0_single` `0.5`	`ValueError: At least 2 POIs required for pathfinding`
TC-011	Non-numeric tolerance	`scenario_2` `high`	`TypeError: Tolerance must be numeric`

In order to set up the environment and run the test cases, follow the README.md from the root of the zip file.

Parte IV

Use of AI

During the development of this project, generative AI was used to support certain phases of our work. Its usage was strictly limited to:

- **Conceptual understanding** (e.g., radar equations, Gaussian distributions, heuristic admissibility).
- **Debugging assistance** (e.g., troubleshooting code, understanding libraries).

1. Where AI Was Used

- **Radar Equation & Gaussian Distributions**: AI helped clarify the mathematical foundations, including the radar equation (1) and the construction of a multivariate Gaussian distribution (2). It also assisted in defining conditions for admissible heuristics.
- **Python Libraries** (NetworkX, NumPy): We used AI (e.g., DeepSeek) to better understand functions like DiGraph() in NetworkX, as official documentation sometimes lacked detailed examples.
- **Debugging Support**: AI provided alternative approaches when debugging issues in grid generation and probability scaling.

2. How AI Facilitated the Project

- **Efficiency**: AI summarized technical papers on radar detection models, speeding up research.
- **Library Explanations**: Some Python functions had sparse documentation, but AI supplemented with practical examples and breakdowns.
- **Validation**: Our heuristic admissibility proofs were cross-checked against AI-generated examples to ensure correctness.

3. Verification Corrections

All AI-provided information was manually verified:

- Formulas (e.g., radar equation) were cross-referenced with academic sources.
- Heuristics were tested in edge cases (see Experiments section).
- Code suggestions were adapted to fit our implementation.

Parte V

Conclusion

In this project we have successfully implemented a heuristic search system for a stealth airplane navigating through a radar-monitored airspace without being detected while visiting some points of interest. This was achieved by modelling radar detection using multivariate Gaussian distributions and an A* algorithm with admissible heuristics.

1. Mathematical modelling:

- We formalized the problem by using grid-based states, radar equations and probabilistic radar detection thresholds.
- We designed two admissible heuristics (Euclidean and Manhattan distance) for optimal pathfinding.

The Euclidean heuristic calculates straight-line distance and is ideal for open areas with few obstacles, providing shorter, smoother paths that minimize detection exposure when diagonal movement is allowed, while the Manhattan heuristic, which sums horizontal and vertical distances, works better for grid-based movement in dense radar environments, as it avoids underestimating costs around obstacles and is computationally faster for large grids.

See *Fig.1* for the outputs on scenario 9, the first being the one with the Euclidean heuristic, and the second the Manhattan one.

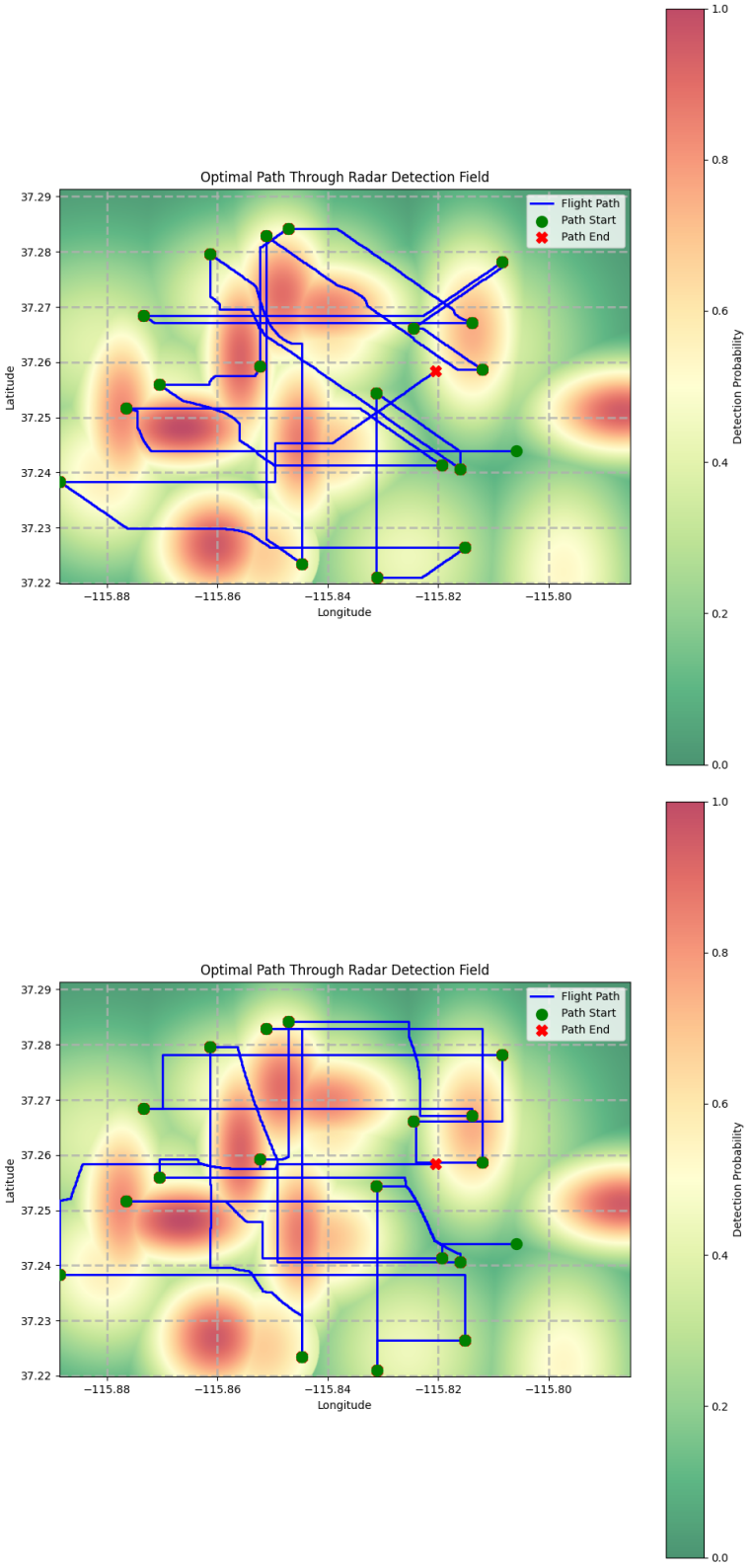

Fig. 1: *Paths graphs*

2. Technical implementation:

- Built a probability map in Python using NumPy and integrated with Networkx for graph-based search.
- Implemented edge cases too.

3. Testing:

- We tested different scenarios with different POIs and tolerance to ensure good performance.

4. Challenges

Some of the challenges that arose during the development of the project were:

- The conversion from geodetic(degrees) to Cartesian(meters) coordinates result in approximation errors, especially in large sized areas.
- Ensure that heuristics never overestimated costs required careful testing, specially in edge cases.
- Large grids result in slower pathfinding. For improving this, we should implement more modifications in the future.

In conclusion, this project highlights the importance of heuristic search in stealth aviation, validating theoretical modelling and practical implementation. The tests show that even if the aircraft's technology is not perfect, it is capable of exploiting radars' weaknesses through intelligently planned path mapping.