UNIVERSITÉ MARIE & LOUIS PASTEUR

UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

Positioning systems : techniques and applications

Practice Work Presatation
International Mr-IoT

Advisor
Dr. Phillipe CANALDA

Student
MUCA UENDI
KHALIL Omar

EIPHI-BFC
GRADUATE SCHOOL
CROSS DISCIPLINARY SCIENCE AND TECHNOLOGY

27.03.2025

MUCA Uendi
KHALIL Omar

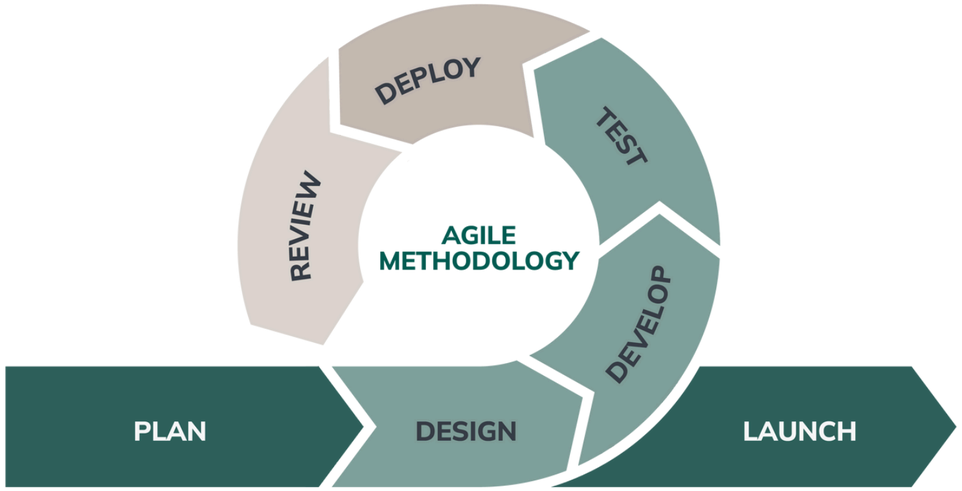
Master m1 Internet of Things

Table of Contents

Architecture and flow of the programs

The program is implemented in Python and follows the **Agile** methodology.

Agile methodology is a project management framework that breaks projects down into several dynamic phases, commonly known as sprints. Each team member had his task, evaluated with time.



Requirements:

1. **Programming language**:Python 3 or above as the programming language.
2. **Libraries**:NumPy and SciPy libraries and matplotlib ,seaborn,tabulate

3.IDEs: VS Code and google collab as the recommended IDE, but other IDEs can be used.

UNIVERSITÉ MARIE & LOUIS PASTEUR

UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

N-Lateration

The purpose of this program is to estimate the position of a receiver based on the distances to four reference points using the N-Lateration algorithm. The program is implemented in Python and follows the Agile methodology.

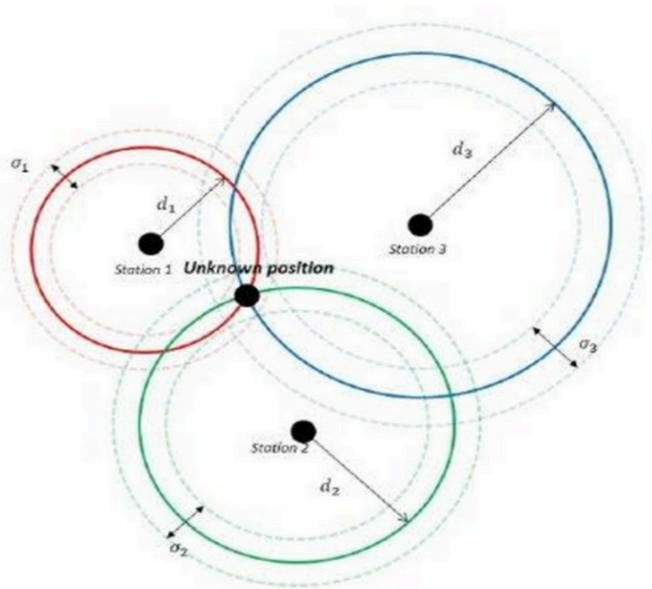
MUCA Uendi
KHALIL Omar

Master m1 Internet of Things

27.03.2025

Introduction to N-lateration

- N-Lateration is a positioning method used to estimate the coordinates of an unknown point based on measured distances from some known reference points like GPS satellites or WiFi access points .
- Trilateration (2D space) using three reference points to find a 2D coordinates .
- N+1 reference points -> N-dimensional space.



1. Radius equal to the distance between the access point and the receiver.
2. The estimated position intersection of the 3 spheres
3. Mathematically is the solution of nonlinear system of equations formed from coordinates and distance.

UNIVERSITÉ MARIE & LOUIS PASTEUR

UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

- **N-lateration implenetation**

1.Develop an **Object Class Diagram** to show system structure including attributes and methods

2.Implement two main algorithms:

- The **N-Lateration algorithm**
- The **Factory Design Pattern,** which structures and organizes the dataset for efficient handling.

3.Compare the computed position with a geometric resolution of the problem, validating accuracy through visual and analytical methods.

Input data set :

E_i	Center (of the AP, the satellite, etc.)	Distance to receiver (samrtphone)
i = 0	(0,5 ; 0,5 ; 0,5)	3
i = 1	(4 ; 0 ; 0)	2
i = 2	(4 ; 5 ; 5)	4,2
i = 3	(3 ; 3 ; 3)	2,5

The input data set includes the center of the four APs, the satellite, and the distance, and the receiver,and estimated position minimizes the sum of distances to all spheres.The implementation also incorporates key theoretical models:

1. **Celerity Model** (signal travel time for distance)
2. **Friis Attenuation Model**

$$(x-xi)^2+(y-yi)^2+(z-zi)^2=di^2$$

$$di=c*ti$$

UNIVERSITÉ MARIE & LOUIS PASTEUR

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

N-lateration Metrics used

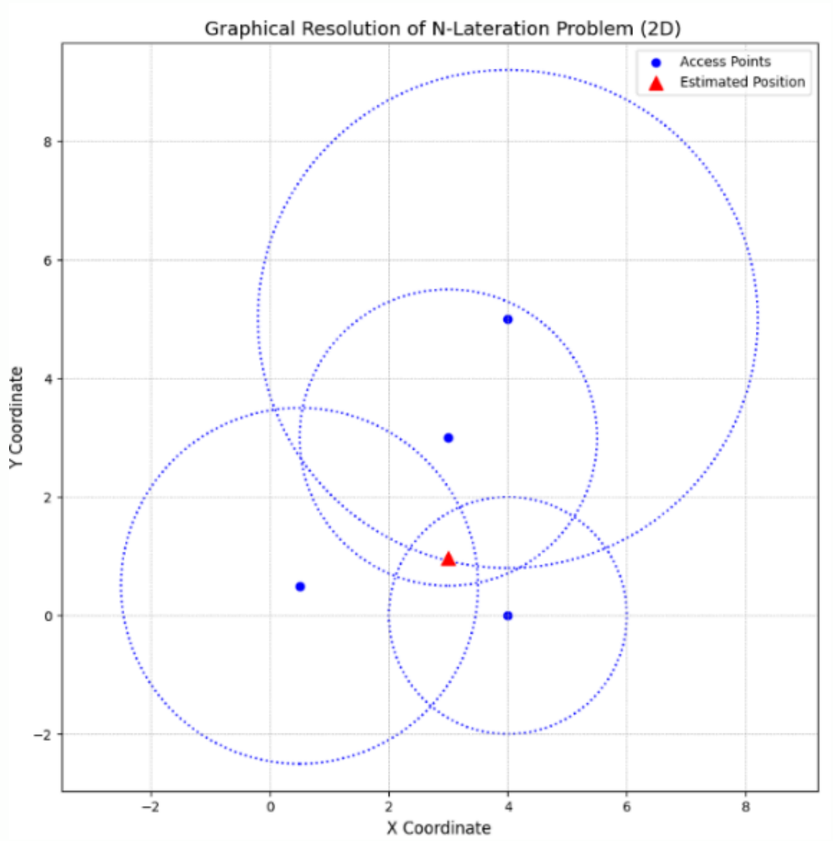
- Measures the straight-line distance between the receiver and each emitter in 3D space
- The algorithm starts the optimization using the average of all emitter coordinates as an initial guess for the receiver's position
- Calculates the difference between the estimated distances and the actual distances from each emitter
- Used a function called least squared to minimize errors to find best solution
- Visualize

Least squared method

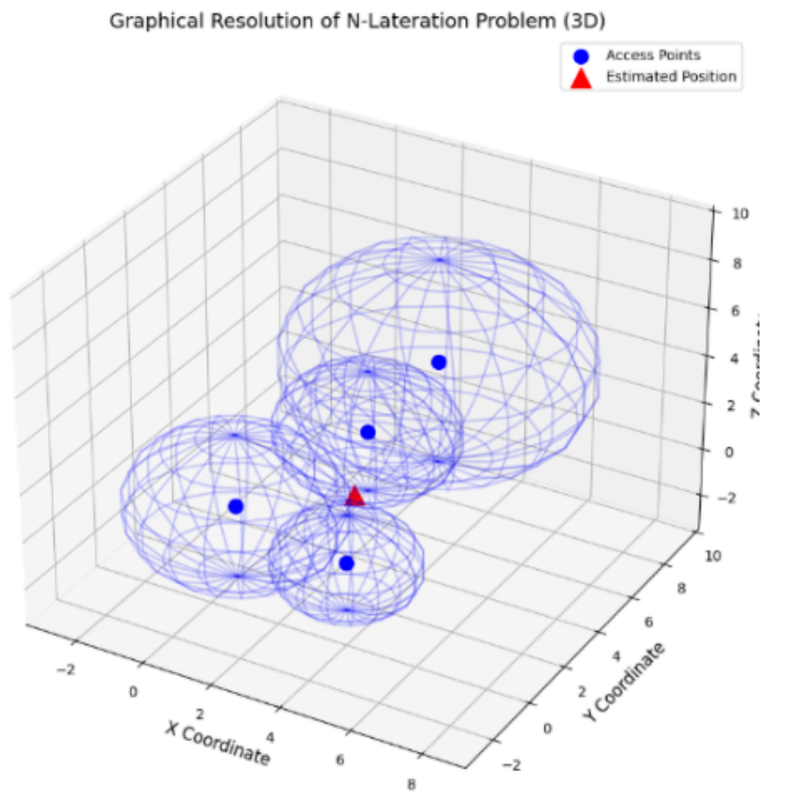
The least squares method adjusts the receiver's estimated position so that the difference between the measured distances and calculated distances to the emitters is as small as possible.

Graphical representation of problem

Graphical 2D resolution of the N-Lateration problem

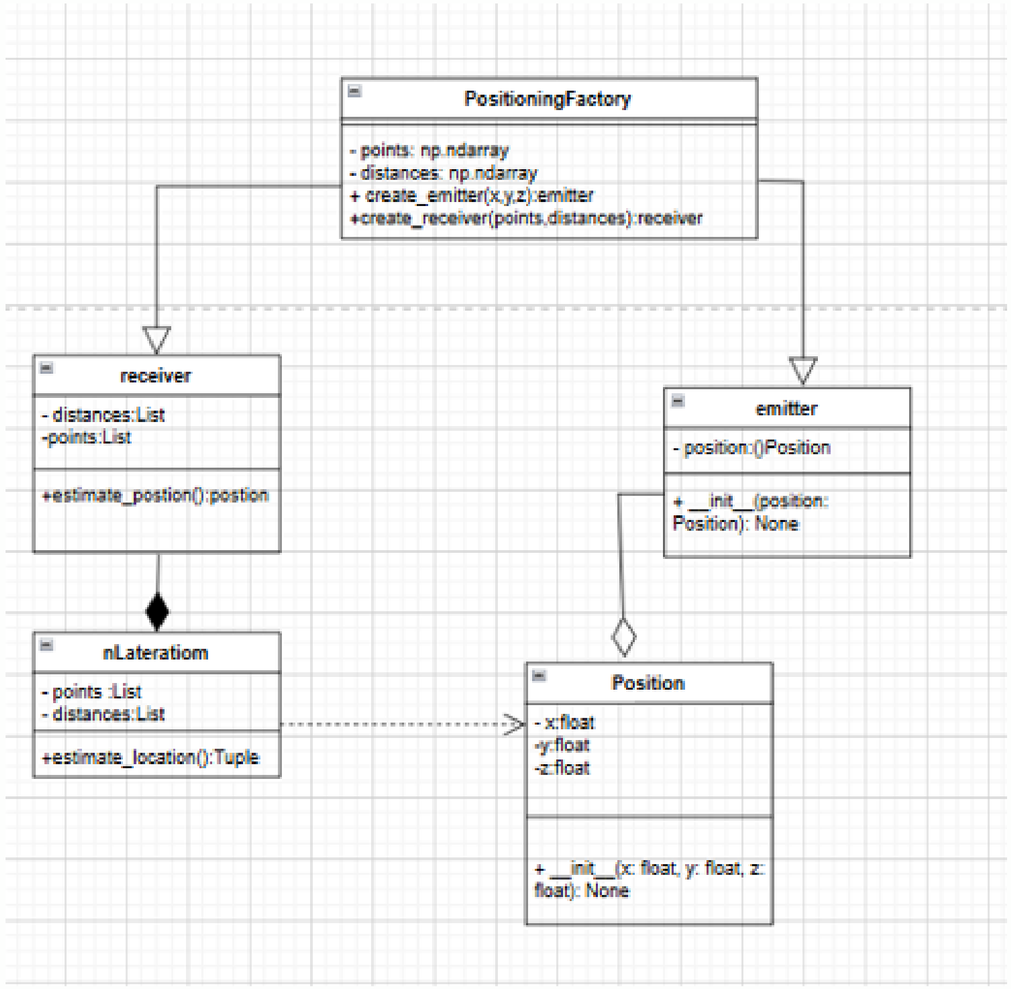


3D resolution of the N-Lateration problem



In theory, with perfect measurements, the circles would intersect at one exact point, giving the exact location of the receiver.But in practice we get a triangle shape intersection due to some reasons like environment effects (walls,people moving etc),interference,noise etc.The actual estimated coordinates are somewhere in the center of that triangle, computed numerically.

Class Diagram with Positioning Factory



The class diagram has **5 classes** where each is responsible for a specific thing in the N-Lateration algorithm.

1. **PositioningFactory**
2. **Emitter**
3. **Receiver**
4. **nLateration**
5. **Position**

Data Structure

1. Lists (list)
2. NumPy Arrays (np.array)
3. Dictionaries (dict)
4. Objects (Classes)
5. Tuples (tuple) : Used for immutable storage of emitter positions (x, y, z).

MUCA Uendi
KHALIL Omar

N-Lateration Pseudo-Code (Simplified)

START

Get user input for emitters and distances
IF less than 4 emitters → use default dataset

For each emitter:
Create Position(x, y, z)
Wrap in Emitter object

Create Receiver with points & distances
Receiver calls NLateration.estimate_location()
→ Initial guess = average of emitter positions
→ Minimize error using least squares
→ Return estimated Position(x, y, z)

Display estimated position

END

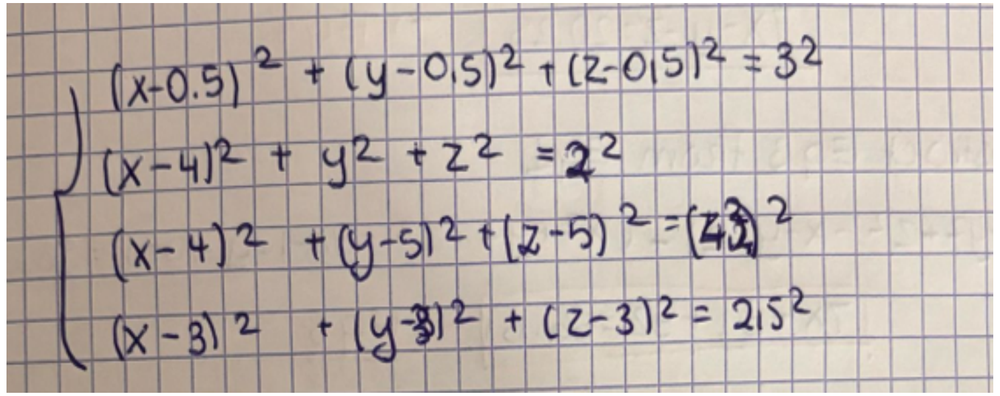
MUCA Uendi
KHALIL Omar

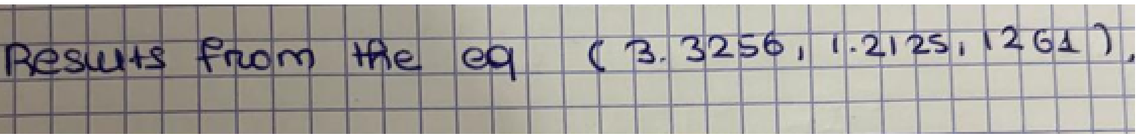
Results from test

```
Do you want to use the default dataset? (yes/no): yes

Estimated Position: (3.408832, 1.533518, 1.533465)
```

```
Do you want to use the default dataset? (yes/no): no

Enter emitter positions (x, y, z) and their distances. Enter 'done' to finish.
Enter emitter position (x y z distance) or 'done' to finish: 2 3 4 6
Enter emitter position (x y z distance) or 'done' to finish: 1 2 3 4
Enter emitter position (x y z distance) or 'done' to finish: 7 6 5 3
Enter emitter position (x y z distance) or 'done' to finish: 3 2 1  5
Enter emitter position (x y z distance) or 'done' to finish: 3 2 1 3
Enter emitter position (x y z distance) or 'done' to finish: done

 Estimated Position: (6.068087, 4.176065, 2.284856)
```

Test 1-We solved the system of equations also mathematically and we got this results:(3.3256,1.2125,1.2661)





- The program successfully estimates the position of the receiver using the N-Lateration algorithm.
- They vary few for due to this reason:

1. last_squares method in your code minimizes the total error iteratively
2. Min threshold criteria puted by least_squares solver
3. Depend on initial start.

Since we are solving an over-constrained system, the algorithm finds the best numerical approximation instead of a strict algebraic intersection.

Static-Fingerprint

The problem this code aims to solve is to estimate the position of a mobile terminal in a given square area based on a set of reference points (cells) with known locations and radio signal strengths

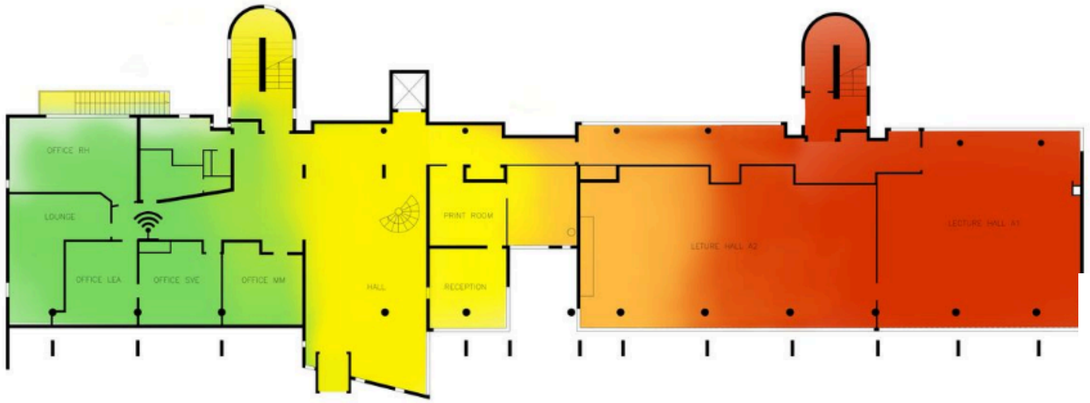
MUCA Uendi
KHALIL Omar

Master m1 Internet of Things

27.03.2025

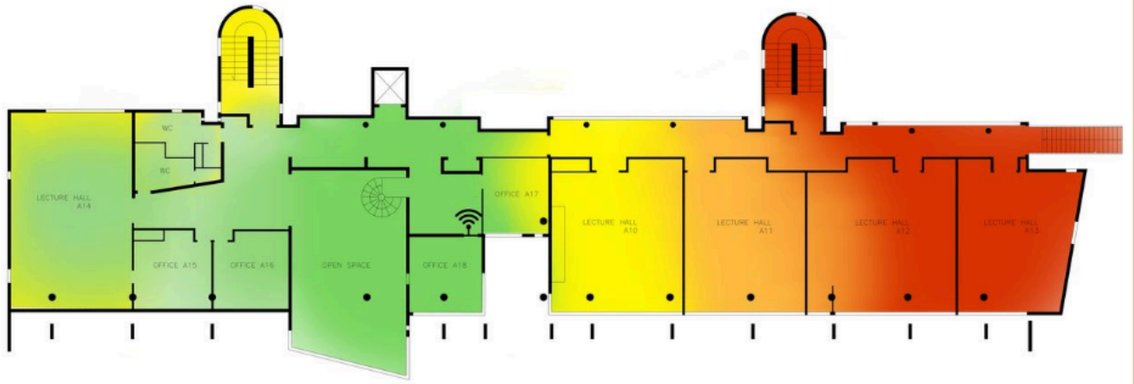
Introduction to FingerPrint

- The Fingerprint algorithm is a technique used in wireless communication systems to locate a mobile device based on the strength of the signals received from multiple Wi-Fi access points or cellular base stations. Classify and choose points based on **k-NN** algorithm->k=sqr(location_nr)



Ground floor-STGI BULDING A

First floor-STGI BULDING A



UNIVERSITÉ MARIE & LOUIS PASTEUR

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

Fingerprint-based localization operates in two main phases:

Cell i	Reference Location (x, y) (meters)	RSSI Vector [r1,r2,r3,r4]
0	(0,0)	[-38, -27, -54, -13]
1	(4,0)	[-74, -62, -48, -33]
2	(8,0)	[-13, -28, -12, -40]
3	(0,4)	[-34, -27, -38, -41]
4	(4,4)	[-46, -48, -72, -35]
5	(8,4)	[-45, -37, -20, -15]
6	(0,8)	[-17, -50, -44, -33]
7	(4,8)	[-27, -28, -32, -45]
8	(8,8)	[-30, -20, -60, -40]

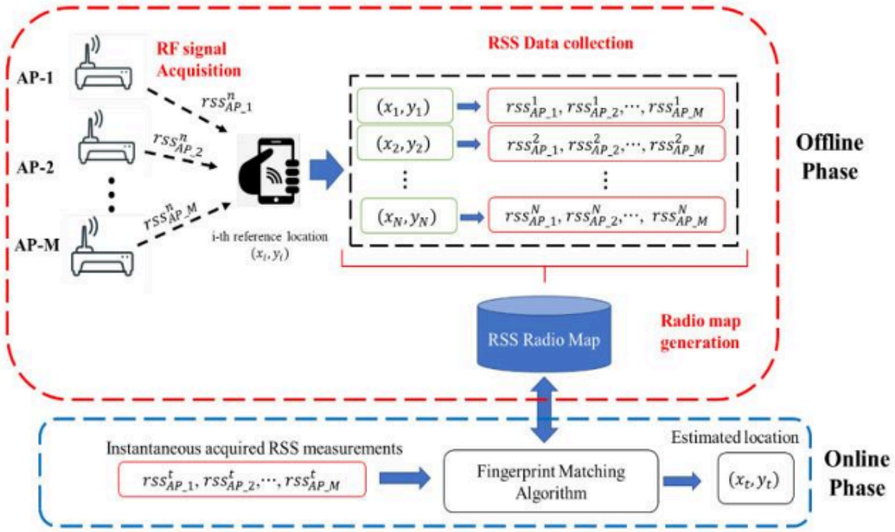
1.Offline Phase (Fingerprint Collection)

- Collect data manually and store them in database.
- The recorded dataset consists of: Reference Location(xi,yi)→RSSI Vector

2.Online Pahse(Postion Estimation)

- The system compares this new measurement with the stored fingerprint database and selects k-nearest reference points based on RSSI similarity.We use weighted average method.

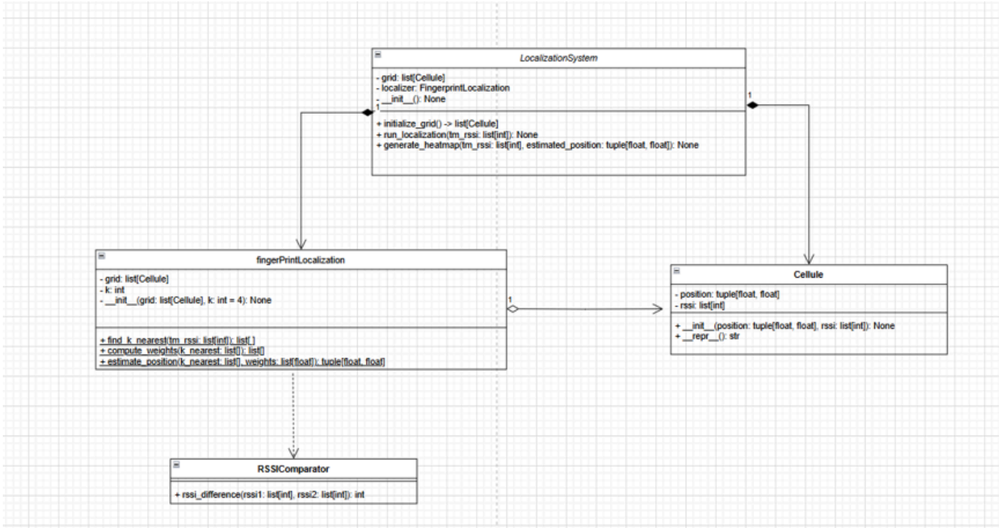
TM RSSI=[-26,-42,-13,-46]



Proposed Architecture

- The architecture of a static fingerprint system involves deploying a set of fixed APs or cell towers, performing offline fingerprinting to record the RSSI values at known locations, and using these values to estimate the location of a mobile device based on its observed RSSI values.
- The accuracy of the system depends on factors such as the density and placement of the APs or cell towers, the quality of the RSSI measurements, and the accuracy of the location estimation algorithm.
- Cellule Class ,RSSI-Based Fingerprinting Algorithm,Weighted ,Position Estimation,RSSICompator

Class Diagram



Data Structure

1. Lists (list)
2. NumPy Arrays (np.array)
3. Objects (Classes)
4. Tuples (tuple)

Metrics used for static fingerprint

- Offline:
  - Collect readings of RSSI from different locations for each AP to get vectors of measurements
- Online:
  - Get the user/receiver RSSI readings
  - Substitute them from our vectors
  - Get weights (one/difference) for each location
  - Using KNN choose the smallest 4 values, which are the closest to the receiver
  - Get total weight and standardize (weight of a location/total weight)
  - We multiply standardize weight of a loc with its coordinates to get finally the estimated location of the user

Fingerprint-Based Localization (KNN Pseudocode Simplified)

```
CLASS Cellule:
INIT(position, rssi)

CLASS RSSIComparator:
FUNCTION rssi_difference(r1, r2):
RETURN sum of absolute differences

Class FingerprintLocalization:
Function __init__(grid, k):
# grid: list of Cellule objects (known reference points)
# k: number of nearest neighbors to consider
self.grid = grid
self.k = k

Function find_k_nearest(tm_rssi):
# tm_rssi: RSSI values at the unknown location
distances = []

For each cell in self.grid:
diff = RSSIComparator.rssi_difference(tm_rssi, cell.rssi)
distances.append((diff, cell.position))
# Sort by smallest difference and return the top-k
distances.sort by diff
Return first k items in distances
Return (x, y)

Function compute_weights(k_nearest):
weights = []
For (diff, pos) in k_nearest:
# Avoid division by zero
weight = 1 / diff if diff != 0 else 1
weights.append(weight)
# Normalize weights so they sum up to 1
total_weight = sum(weights)
weights = [w / total_weight for w in weights]
Return weights
Function estimate_position(k_nearest, weights):
x, y = 0, 0
For i in range(len(k_nearest)):
(diff, position) = k_nearest[i]
x += position[0] * weights[i]
y += position[1] * weights[i]
CLASS LocalizationSystem:
INIT(): load reference grid and set k
FUNCTION run(tm_rssi):
k_nearest ← find_k_nearest(tm_rssi)
weights ← compute_weights(k_nearest)
position ← estimate_position(k_nearest, weights)
PRINT position
MAIN:
system ← LocalizationSystem()
tm_rssi ← current RSSI
system.run(tm_rssi)
```

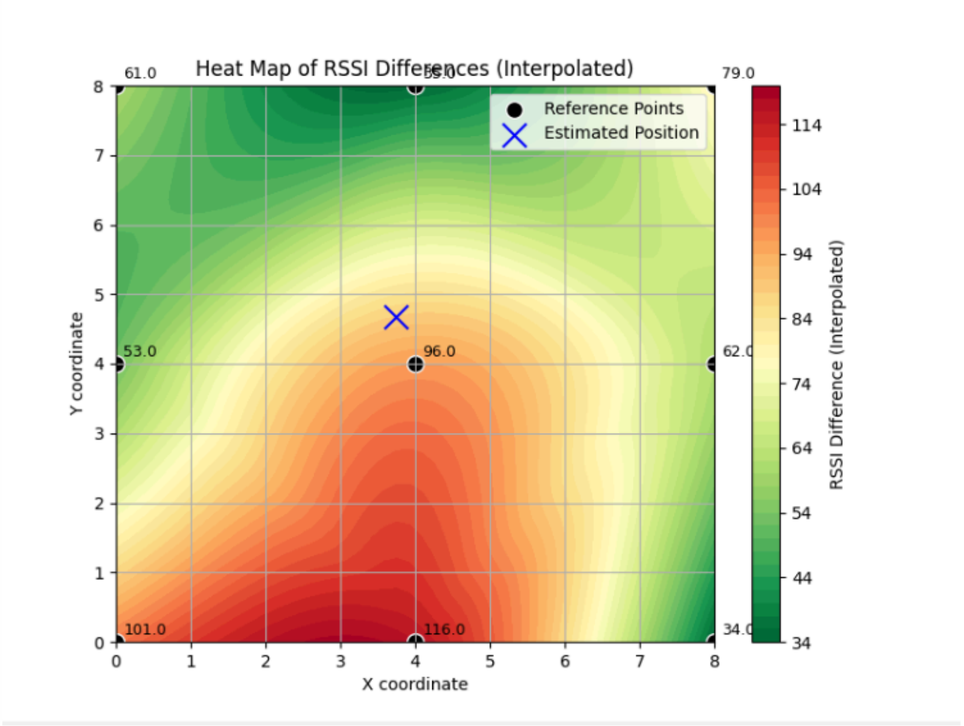
Results from test

Results:The code output provides an estimated position of (x = 3.75, y = 4.67), while the theoretical proof calculates a slightly different position of (x = 3.77, y = 4.66). This minor discrepancy may arise due to floating-point approximations, rounding differences, or numerical precision limitations in the computational approach.

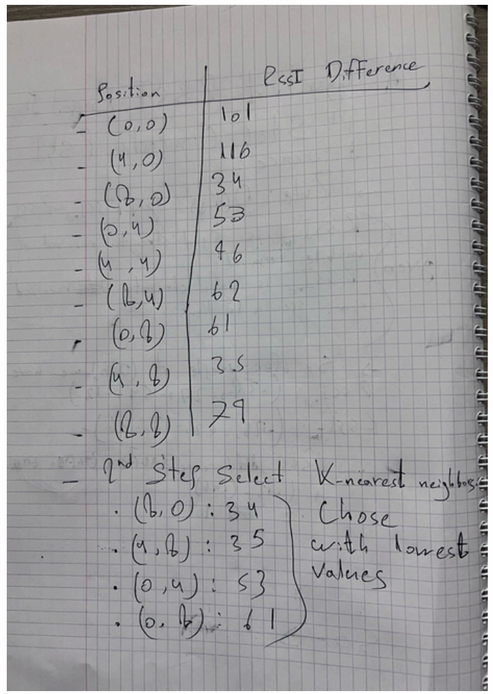
From rule k=3 or 4 as we have 8 location

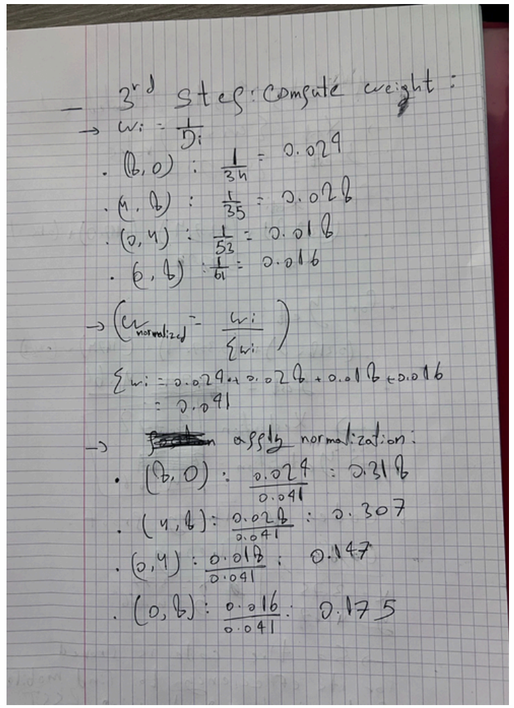
We got more accurate results and our data are not varying too much.

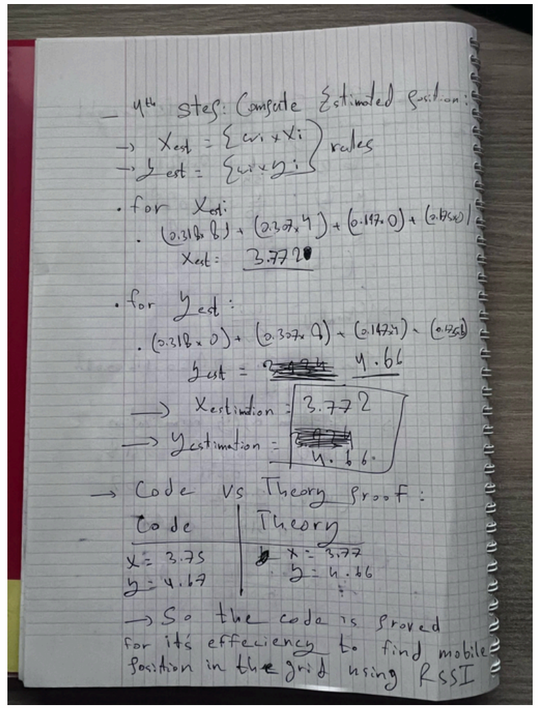
```
Enter number of Access Points (APs): 4
Enter number of reference locations: 9

Reference Location 0:
Enter x coordinate: 0
Enter y coordinate: 0
Enter RSSI values from 4 APs (comma-separated): -38,-27,-54,-13

Reference Location 1:
Enter x coordinate: 4
Enter y coordinate: 0
Enter RSSI values from 4 APs (comma-separated): -74,-62,-48,-33

Reference Location 2:
Enter x coordinate: 8
Enter y coordinate: 0
Enter RSSI values from 4 APs (comma-separated): -13,-28,-12,-40

Reference Location 3:
Enter x coordinate: 0
Enter y coordinate: 4
Enter RSSI values from 4 APs (comma-separated): -34,-27,-38,-41

Reference Location 4:
Enter x coordinate: 4
Enter y coordinate: 4
Enter RSSI values from 4 APs (comma-separated): -46,-48,-72,-35

Reference Location 5:
Enter x coordinate: 8
Enter y coordinate: 4
Enter RSSI values from 4 APs (comma-separated): -45,-37,-20,-15

Reference Location 6:
Enter x coordinate: 0
Enter y coordinate: 8
Enter RSSI values from 4 APs (comma-separated): -17,-50,-44,-33

Reference Location 7:
Enter x coordinate: 4
Enter y coordinate: 8
Enter RSSI values from 4 APs (comma-separated): -27,-28,-32,-45

Reference Location 8:
Enter x coordinate: 8
Enter y coordinate: 8
Enter RSSI values from 4 APs (comma-separated): -30,-20,-60,-40

--- Live RSSI Input ---
Enter RSSI values from APs (comma-separated): -26,-42,-13,-46

Estimated Position (RSSI Weighted): (3.75, 4.67)
```



Results from solving manually







Minor differences due to rounding are expected but do not affect overall accuracy, confirming the method's effectiveness for indoor localization.

Prediction with Hidden Markov Model

MUCA Uendi
KHALIL Omar

Master m1 Internet of Things

27.03.2025

- The main idea behind a Markov model is to model the probabilities of different states that a system can be in and the rates of transitions among them. The system depend only on its current state,not in prevoius states.
- Markov models are graphs with **nodes** as system **states** and edges as **transitions**. Each edge has a probability showing the chance of moving between states. These probabilities depend on transition rates, which reflect how quickly the system changes state.

Hidden Markov Model (HMM)

- HMMs are used to model the probability of system transiting from one hidden state to another but only is is being able to observe an output that is generated by current state.
- Hidden Markov model has two main components:the state transition model and the observation model.The state transition model describes the probability of transitioning from one hidden state to another, while the observation model describes the probability of generating an observable output given the current hidden state
- Mathematically, an HMM is represented as (S, V, A, B, π):

1. S (Hidden States): The possible internal states of the system.
2. V (Observable Outputs): The visible events or observations.
3. A (State Transition Matrix): Probabilities of moving between hidden states.
4. B (Observation Probability Matrix): Probabilities of different observations occurring in each hidden state.
5. π (Initial State Distribution): Probability of the system starting in each hidden state.

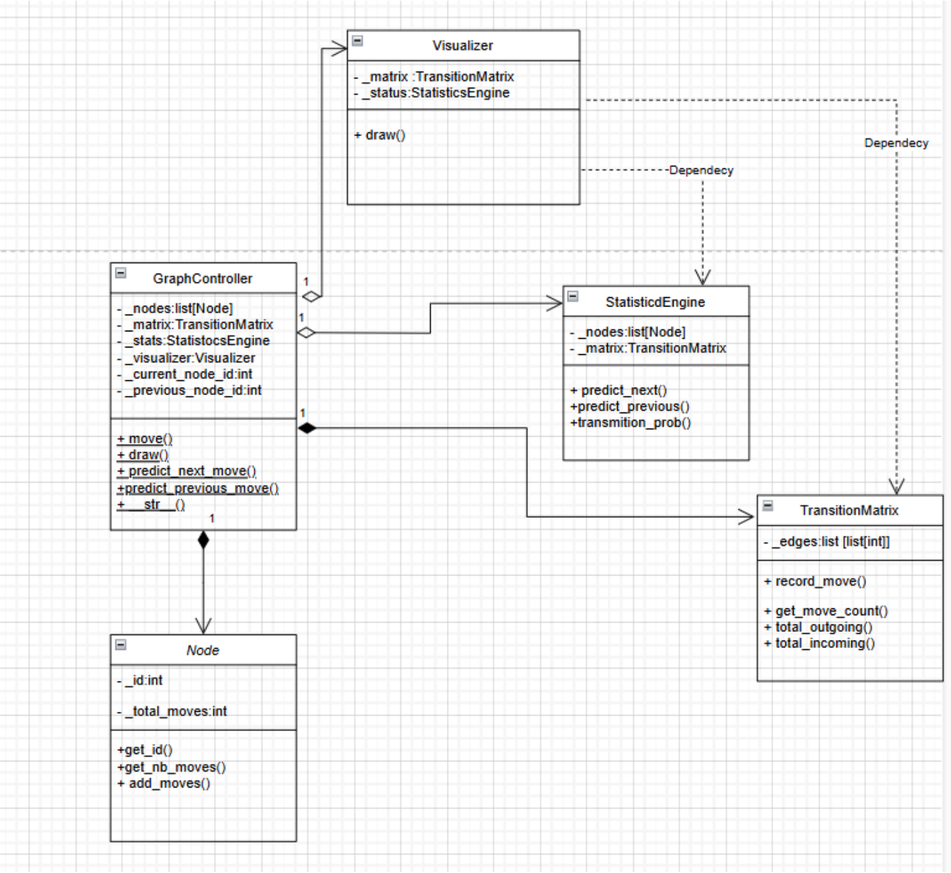
Introduction of Problem

- Hidden Markov Models (HMMs) are widely used for mobility prediction because they can effectively **analyze sequentia**l data and manage noisy sensor readings.
- The goal to track the user navigating through the site ,analyzing it's movement patterns and use this information to enhance visitors experience.By predicting future actions based on probability of past interactions so we can understand how visitors navigate through the website and use that information to we can optimize and improve user experience

Metrics used in HMM

- Starting, we tracked visiting data
- Then, we summed the visits for each node
- Summed the visits for each row and column
- Calculated probability based on: proba= Visits in a node/Total in row*100 for next move prediction
- Calculated probability based on: proba= Visits in a node/Total in column*100 for previous move prediction
- The next or previous predictions are chosen, if they the % of a certain node in its row/column is the highest

Class Diagram



The GraphController manages navigation by integrating five components: Node, GraphController, TransitionMatrix, StatisticEngine, and Visualizer. It uses Node to track page visits and TransitionMatrix to record transitions between nodes. The StatisticEngine analyzes transitions to predict likely next or previous moves. The Visualizer draws the graph using data from the TransitionMatrix and insights from the StatisticEngine. Both StatisticEngine and Visualizer depend on the TransitionMatrix, which acts as the central data source.

Data Structure

1.self._node:List[node]

2.self._edges->List[List[int]]

3.stats:List[float]

MUCA Uendi
KHALIL Omar

Fingerprint-Based Localization (KNN Pseudocode Simplified)

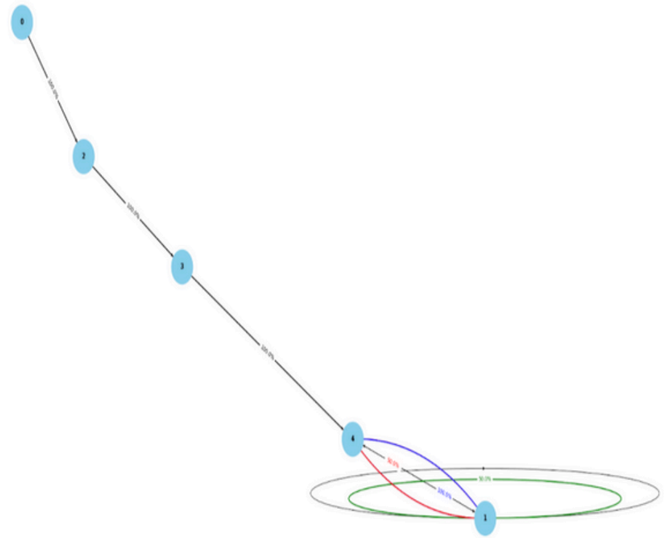
```
CLASS Node:
- INIT(id_num)
- get_id()
- get_nb_moves()
- add_move()
CLASS Graph:
- INIT(nb_nodes=5)
- move(next_node_id)
- node_stats(origin_id, destination_id)
- predict_next_move()
- predict_previous_move()
- draw_graph()
- __str__()  # returns a transition table
FUNCTION interactive_mode(graph):
WHILE True:
INPUT next_move
IF next_move == -1:
BREAK
ELIF next_move == 5:
 graph.draw_graph()
ELIF 0 <= next_move <= 4:
 graph.move(next_move)
PRINT graph
PRINT current node and previous node
CALL graph.predict_next_move()
CALL graph.predict_previous_move()
ELSE: PRINT "Invalid move"
```

```
FUNCTION random_mode(graph, moves=10):
FOR i IN range(moves):
random_id ← random between 0 and 4
graph.move(random_id)
PRINT graph
PRINT current node and previous node
CALL graph.predict_next_move()
CALL graph.predict_previous_move()

MAIN:
graph ← Graph(5)
PRINT "Select Mode: 1. Interactive  2. Random"
INPUT mode
IF mode == "1":
CALL interactive_mode(graph)
ELSE:
CALL random_mode(graph, moves=10)
```

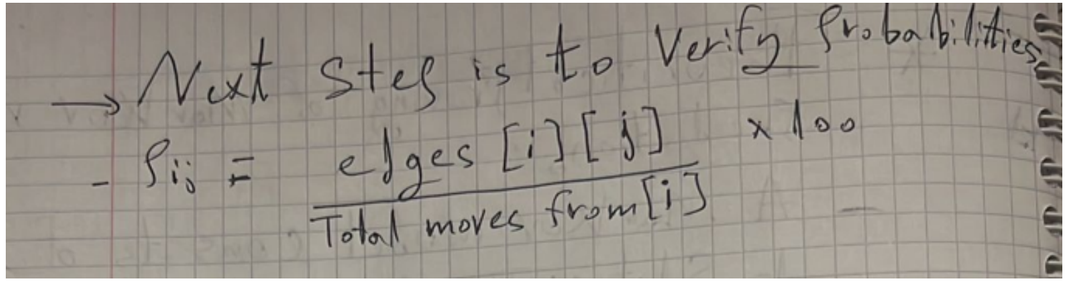
Results from the code execution

From\To	Page 0	Page 1	Page 2	Page 3	Page 4	Total Visits
Page 0	1 m, 50.00%	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	2 visits
Page 1	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	1 m, 50.00%	2 visits
Page 2	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 100.00%	0 m, 0.00%	1 visits
Page 3	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 100.00%	1 visits
Page 4	0 m, 0.00%	1 m, 100.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 visits
Total Visits	1 moves	2 moves	1 moves	1 moves	2 moves	7 visits

```
Current Node: 4
Original Previous Node: 1
Predicted Next Page: 1
Predicted Previous Page: 3
```



This table is using the same movement set we will use in the 'Theoretical proof' which is {0,2,3,4,1,1,4}., it shows the number of visits for each node and the predicted percentages of the next/previous move.



Legend:

- Blue and red line indicated bidirectional connection between nodes
- Green line indicates a loop
- Value shown is the predicted movement percentage

A Complex scenario:

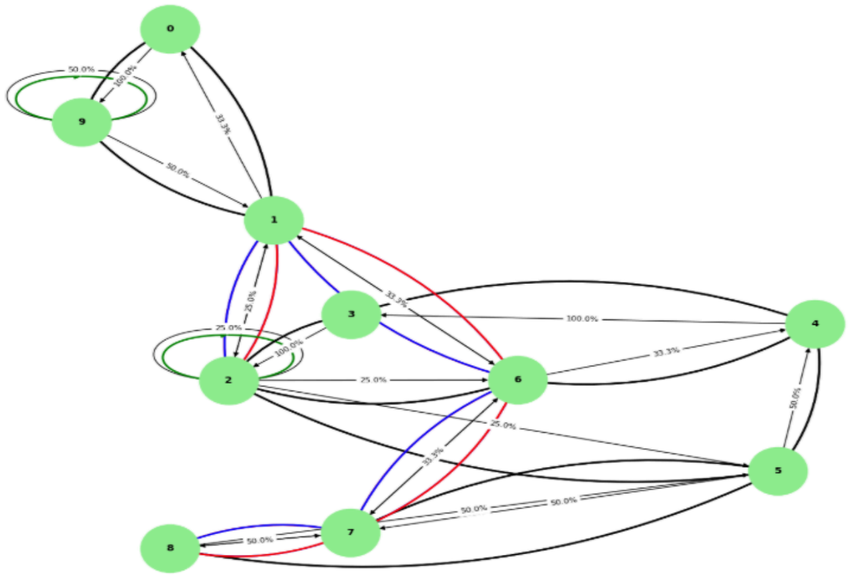
The scenario is that we have a building of 10 rooms. Track how many times the rooms are getting visited.

From\To	Room 0	Room 1	Room 2	Room 3	Room 4	Room 5	Room 6	Room 7	Room 8	Room 9	Total Visits
Room 0	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 100.00%	1 visits
Room 1	1 m, 33.33%	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	3 visits
Room 2	0 m, 0.00%	1 m, 25.00%	1 m, 25.00%	0 m, 0.00%	0 m, 0.00%	1 m, 25.00%	1 m, 25.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	4 visits
Room 3	0 m, 0.00%	0 m, 0.00%	2 m, 100.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	2 visits
Room 4	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	2 m, 100.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	2 visits
Room 5	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	2 visits
Room 6	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	0 m, 0.00%	3 visits
Room 7	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 33.33%	0 m, 0.00%	2 m, 66.67%	0 m, 0.00%	3 visits
Room 8	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	2 visits
Room 9	0 m, 0.00%	1 m, 50.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	0 m, 0.00%	1 m, 50.00%	2 visits
Total Visits	1 moves	3 moves	4 moves	2 moves	2 moves	2 moves	3 moves	3 moves	2 moves	2 moves	24 visits

```
Current Room: 0
Original Previous Room: 1
Predicted Next Room: 9
Predicted Previous Room: 1
```

The results obtained from the Markov Model prediction code and the theoretical manual calculations are identical, confirming that the implemented probability rule is correctly applied in the code. By using the transition probability formula:

percentage = (moves * 100) / total_moves.



In conclusion, mobility prediction with Hidden Markov Models is a useful and versatile technique for modeling and predicting the movement patterns of individuals or groups of people over time. HMMs allow for the incorporation of multiple sources of data, including geographic information, social network data, and sensor data, which can lead to more accurate and personalized models.

MUCA Uendi
KHALIL Omar

Conclusions:

- The programs successfully estimates the position of the receiver using the N-Lateration algorithm and Fingerprint and predicting succefully the next state with HMM.
- Compare the result provided by your implementation with a geometric resolution of the scenario provided and we find that our program gets the results in your proposed search area.
- The code execution time is few sec and just little more for Fingerprint as it generate RSSi map.

GITHUB REPO: https://github.com/UENDIMUCA/PositionningSystem

E-mail: uendi.umuca@edu.univ-fcomte.fr
E-mail: omar.khalil02@edu.univ-fcomte.fr

Thank you!