**1 IMPLEMENTATION**

**1.1 Used technologies**

1.1.1 MongoDB

MongoDB is a document-oriented database, which is classified as a NoSQL database. Such a database provides a storage and query mechanism that does not require to follow an established relational scheme. NoSQL databases are geared to working with very large and very varied data sets, which are frequently updated. Their architectures vary and are delimited in 4 primary classifications: document databases, graph databases, key-value databases, wide column stores.

Document-oriented database encapsulate and encode data in standard formats or encodings (XML, YAML or JSON). Use of document databases has increased due to the use of [JavaScript](https://www.theserverside.com/definition/JavaScript) and JSON, a data format that has gained wide reputation among web application developers.

In my implementation, the database is composed of JSON type documents, where the correspondence between the location and the fingerprint is made. I chose to use this type of database because of its storage flexibility. A fingerprint can be stored as being composed of only the AP signals it intercepts, 0 values for APs from which no signal is intercepted should not be stored. An example of document can be noticed in Figure 5.

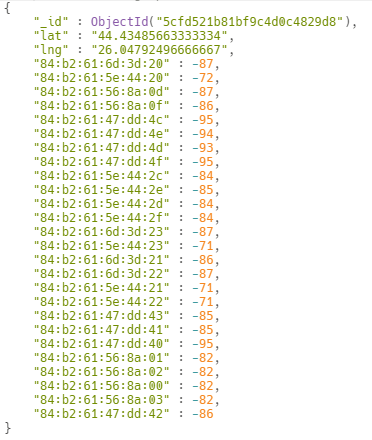


Figure 5: MongoDB entry

1.1.2 Node.js

As stated on the official site, Node.js is an asynchronous, event-driven engine where the application makes a request and then continues working on other useful tasks rather than stalling while it waits for a response. On completion of the requested task, the application is informed of the results via a callback (or a promise or Observable. This enables large numbers of operations to be performed in parallel – essential when scaling applications.

MongoDB was also designed to be used asynchronously, so it works well with Node.js applications. This and the fact that Node.js work very easily with JSON type data, made me choose this configuration (as shown in Figure 6) to build and query the MongoDB database. Also, in the case of an indoor localization application development, it can use HTTP (GET, POST) methods to communicate with the Node.js server in order to query the database.

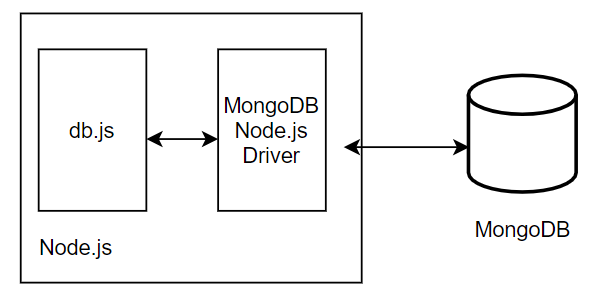


Figure 6: Node.js - MongoDB architecture

1.1.3 Java

Java allows you to create modular programs and reusable code. It was designed to be easier to write, compile, debug and learn than other programming languages. It has the ability to move easily from one computer system to another.

Because of Java’s object-orientation, ease of use and cross-platform capabilities, I chose it for the effective implementation of the offline algorithm.

1.1.4 Octave

Octave is a programming language designed to solve complex numerical computations. Octave programs are made up of scripts. The syntax is very similar to MATLAB, based on matrices and provides many functions for matrix operations. For example, Octave provides an implementation for MDS, which is the main reason I chose to use this programming language.

I also opted for the implementation in Octave of both the clustering algorithm (k-mean) and the algorithm of betweenness centrality, due to the ease of the matrix operations.

**1.2 Stress-free floor plan**

The floor plan is represented by a diagram viewed from above, where the relations between rooms, walls or other spaces can be seen. The plan used for this implementation can be seen in Figure 7.

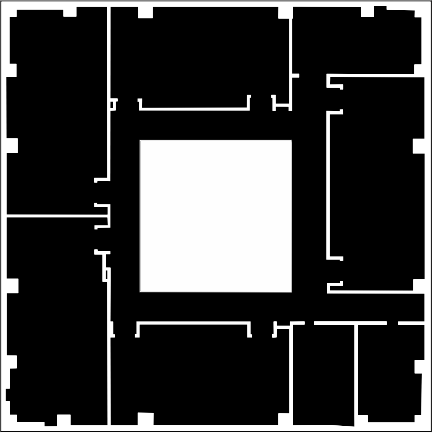


Figure 7: Floor plan

If two locations are extracted from the floor plan, the distance between them is not necessarily the geographical distance between them, because there are different obstacles such as the walls, so another option should be used.

Over the floor plan append a mesh of grids, which are the locations of interest, as shown in Figure 8. The length between grids may be between 1 and 3 meters, according to the article [10]. For longer or smaller lengths, accuracy may suffer. In my implementation I use l = 1m. Further, the distances between each grid are calculated to obtain the matrix D = [dij], where dij is the distance from point i to point j. Due to the fact that I do not use the number of steps as in the original version, the distance between locations is calculated using the following algorithm:

* For each location that is in the same room (the corridor is seen as a room), assign the distance to the neighbors (for the neighbors from the top, bottom, left, right is assigned a distance of 1m, and for neighbors diagonally is assigned the distance 2m).
* At the next step the door locations must be set as part of both the corresponding room and the corridor, and then proceed as in the previous step, assigning distances to the neighbors.
* At the last step is applied the Floyd-Warshall [14] algorithm to compose the shortest paths between locations. It takes O(n3) running time and n is the number of grids. This step results in matrix D.

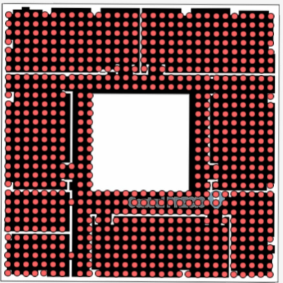


Figure 8: Floor plan with grid locations

The matrix D obtained is then used as input for the Y = cmdscale (D) function of the Octave (cmdscale function is an implementation of MDS), which forms a p-dimension Cartesian space. For p = 2, the stress-free plan obtained is shown in Figure 9.

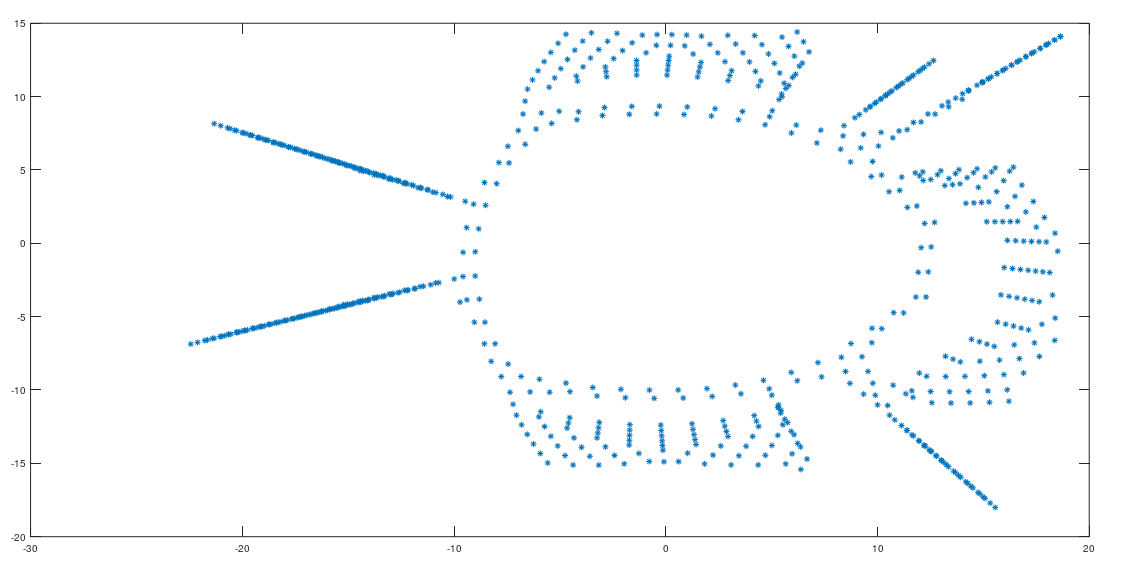


Figure 10: Stress-free floor plan

**1.3 Fingerprint Space**

The fingerprint collection is realised using the application [12], with the default locations where they can be gathered, as in Figure 4. The fingerprints are collected from the meter to the meter, and from each room at a time. The application returns an xml file for each camera where the fingerprints were extracted.

For the fingerprints collected from the same room (the corridor is considered a room), the distance between them is calculated using the coordinates of each point. Then, the door setting is made to realise the connection between the rooms. After the distance between the doors and the fingerprints collected from neighboring points has been calculated, the Floyd-Warshall algorithm is applied to the distance matrix D’ to complete the matrix with the smallest distance between any two fingerprints collected. After fingerprint collection, we have a set of fingerprints F = {fi, i = 1 ...n} (n is the number of records) and a distance matrix D’ = [d’ij] , both of which are essential for constructing the fingerprint space.

Due to the fact that some areas on the floor were not accessible, the grids in those areas were removed and the fingerprints were not collected. Accessible areas from the floor can be seen in Figure 4.

As in the case of stress-free floor plan, MDS receives as input the distance matrix D’ and maps all fingerprints into a p-dimensional Cartesian space. Fingerprint space and Stress-free floor plan after the elimination of inaccessible areas, are shown in Figure 11.

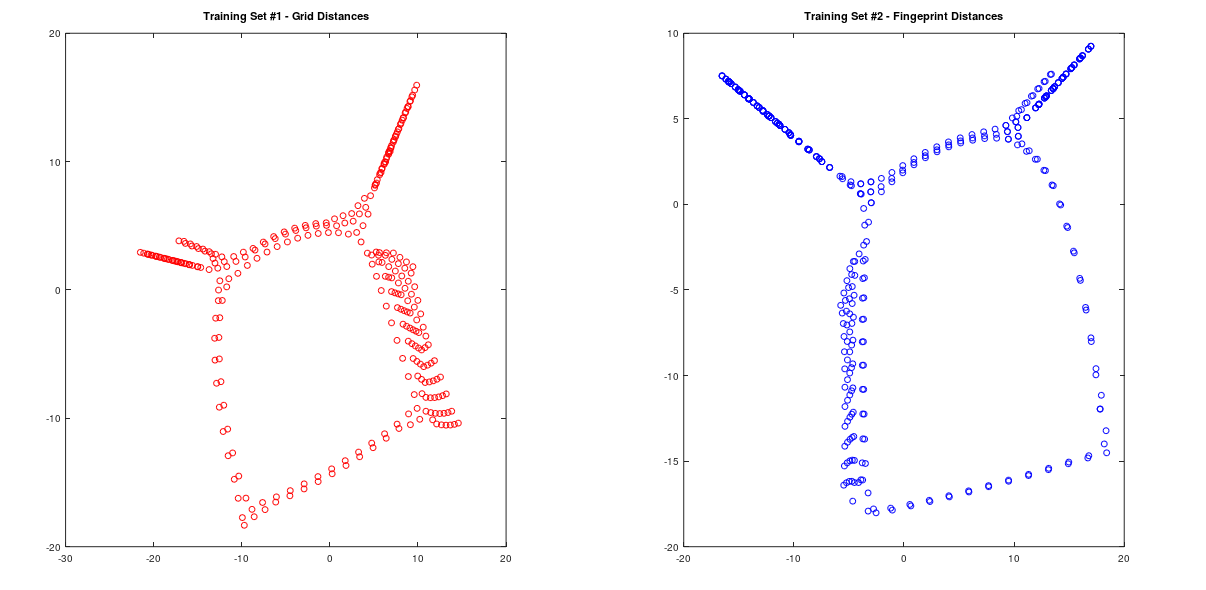


Figure 11: Stress-free floor plan and Fingerprint Space

**1.4 Mapping**

Mapping of fingerprints with real locations is possible due to spatial similarities between stress-free floor plan and fingerprint space.

The first step is corridor recognition, which is connecting all the rooms between them. Moving from one room to another necessarily involves going through the corridor. In graph theory, this means that the fingerprints that are part of the corridor have high centrality values. Implementation uses an betweenness centrality [15] algorithm to recognize the fingerprints that are part of the corridor. Having a graph of the type G = (V, E), the betweenness centrality of the vertex v V is defined as:

B(v) = st(v)st, vst, where st is number of shortest paths from s to t, and st(v) is the number of shortest paths from s to t that pass the vertex v.

Before calculating the fingerprints betweenness centrality, we build Minimum Spanning Tree (MST) to connect all fingerprints through a single path. Now we can calculate the betweenness centrality for MST fingerprints and differentiate the fingerprints in the corridor from those in the rooms as follows: the fingerprints are sorted in descending order by the values obtained from betweenness centrality and knowing that the number of fingerprints collected in the corridor is c, means that the first c fingerprints of the sorted ones are part of the corridor. The fingerprints that are part of the corridor have their betweenness larger than 3500. Let Fc denote the set of fingerprints that are collected from corridors.

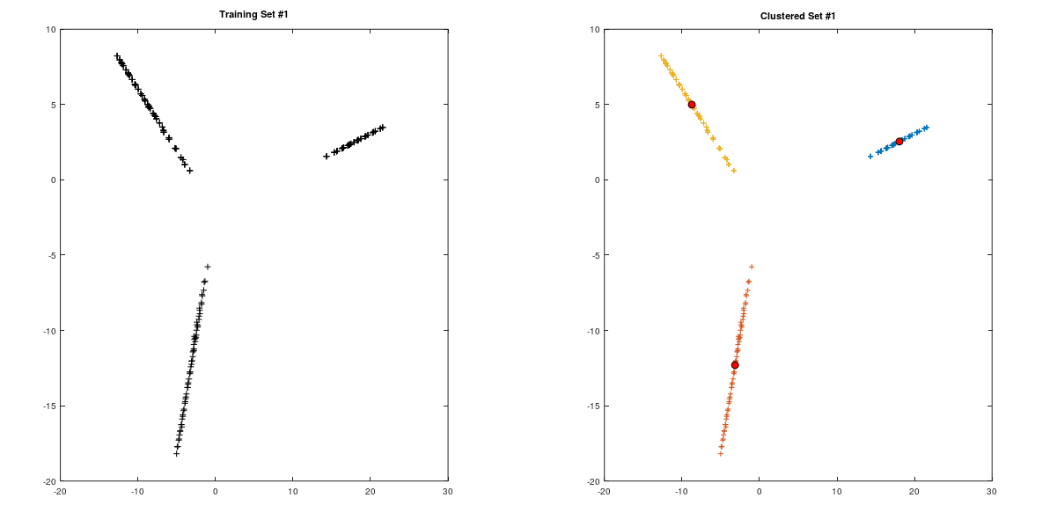


Figure 12: Room recognition(K-means clustering)

The next step is the room recognition. Once the fingerprints belonging to the corridor are removed F -Fc, we obtain k clusters. The k-means algorithm is used to identify these clusters, k being the number of rooms in the floor plan, as shown in Figure 12. At this point we know what fingerprints are from the same room, but we do not know the specific room they belong to.

Further, the doors must be mapped to link the corridor with the rooms and to identify the rooms. The doors are located, finding the closest pairs of fingerprints (fi, fj), fi being in the corridor, and fj being inside the room. Having FD and PD sets, where FD is the set of fingerprints identified as doors, and PD is the set of sample locations identified as doors, we can map the doors in the fingerprint space and the stress-free floor plan by calculating the distances between them and identify the similarities in the distances.

In the last step, with the fingerprints representing the doors mapped to the actual location, the rooms can be further mapped. Using doors as reference points, the fingerprints and sampled locations that are part of the same room are sorted in ascending order by the distance to the door and then are mapped.

**1.5 Querying algorithm**

Building and querying the databases is done in Node.js. To construct databases, objects are created as in Figure 5 and entered into the database as documents. At this point, site-survey database and LiFS database are created.

The query algorithm is based on a tolerance. Let f be the fingerprint to be queried. Initially, the tolerance is 0, and the algorithm query for a fingerprint that is comprised between the values f + tolerance and f - tolerance. The tolerance is incremented until the algorithm outputs a location. If more than one location is returned, we will pick the one that is closest to the previous location returned. The same query is done for both databases, and then the estimated locations of each are compared.