TD1-2-3 – Positionning
GNSS-Like system using Wi-Fi
Frederic Lassabe

Context

The goal of those TDs is to implement some algorithms used to localize a device using only Wi-Fi communications and get familiar with GNSS-like and fingerprinting notions and basic operations. Wi-Fi is a family of wireless networks based on the IEEE 802.11, that are used for wireless internet access using radio wave. It is not designed to provide location information. It is composed of wireless access point that broadcast signals and transmit data to mobile devices. A device (receiver) can scan it surrounding to identify access points (emitters) in its range. Access point and mobile devices are identified via their MAC address. The signal of an access point received by a device have a variable strength. This strength depends on the antenna power, gain, receiver antenna power and gain, the frequency and the inverse of the distance. In an ideal environment (no obstacles, without considering meteorology, no interferences, etc), it follows the Friis equation:

$$\frac{P_R}{P_T} = G_T G_R \left(\frac{\lambda}{4\pi R}\right)^2$$

P_R: Emitter power antenna (in Watt)
P_T: Receiver power antenna (in Watt)
G_T: Emitter power gain
G_R: Receiver power gain
λ: Wavelength of the signal $\left(\frac{c}{freq}\right)$
R: Distance between antennas (in m)

Beside the distance, we know every component, so theoretically we should be able to compute the distance using only this formula; but actually, it’s not accurate because of the environment properties (object, wave rebounding, interferences, etc.). Knowing distance between the device and several access point and their localization, we can multiliterate (=kind of triangulate) the position of the signal. So, the goal of those TDs is to implements solutions to this problem to get a better localization.

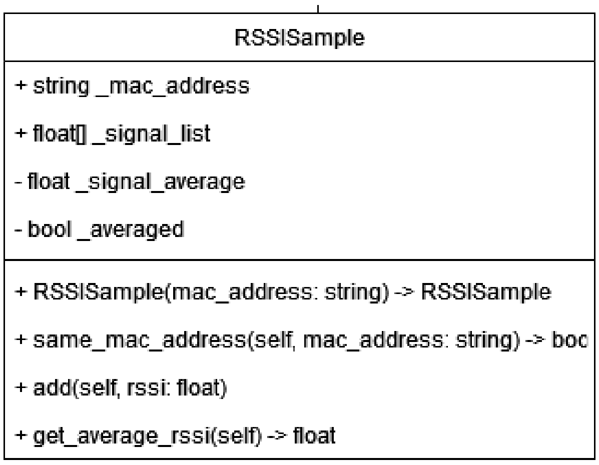
Table des matières

1 - Data structure and code organization

1.1 Objects and class

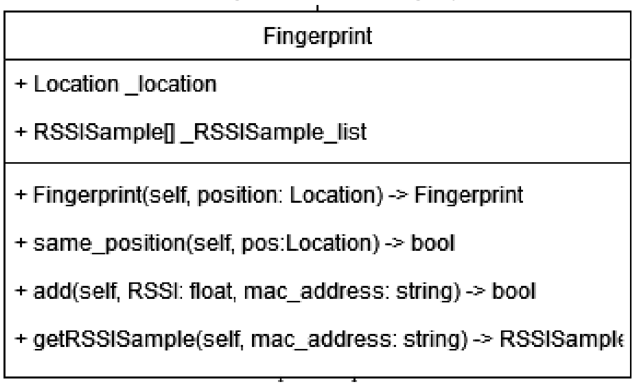
The elementary data is the Received Signal Strength Indication (RSSI), that corresponds to the power measured by the antenna in dBm (decibel-milliwatts). Because this value fluctuates a lot, it's recommended to take multiple measure and keep the mean value.

Each RSSI value are linked to an Access Point (AP) by it's MAC address. This address is unique and allow to differentiate signals from each AP. So, I created a class, called RSSISample, that store those signals, and handle the mean calculation.


RSSISample
+ string _mac_address
+ float[] _signal_list
- float _signal_average
- bool _averaged
+ RSSISample(mac_address: string) -> RSSISample
+ same_mac_address(self, mac_address: string) -> boo
+ add(self, rssi: float)
+ get_average_rssi(self) -> float

[UML Diagram of the RSSISample]

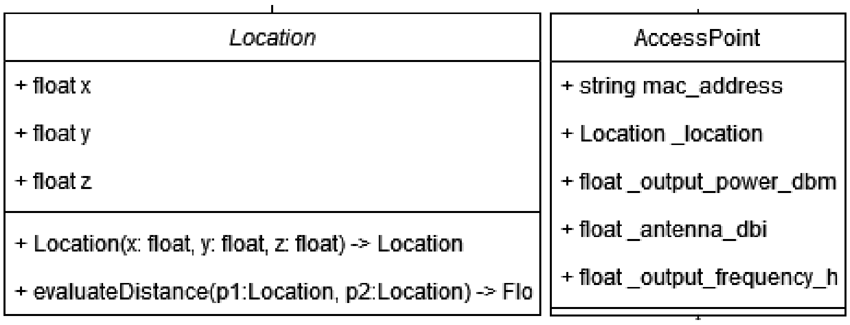
When the device scans its surroundings, it returns a list of mean RSSI values for each APs in range. This record set is called a Fingerprint. In our case, fingerprint is used for calibration, so we know their exact location. I created this second class, called Fingerprint that handle this data structure and handle some other functionality to populate the object and to quickly navigate in RSSISamples.


Fingerprint
+ Location _location
+ RSSISample[] _RSSISample_list
+ Fingerprint(self, position: Location) -> Fingerprint
+ same_position(self, pos:Location) -> bool
+ add(self, RSSI: float, mac_address: string) -> bool
+ getRSSISample(self, mac_address: string) -> RSSISampl

[UML Diagram of the Fingerprint]

I used two more classes to store data such as Location that provide information and tools for basic 3D space computation and storage, and AccessPoint that hold access point data (Location, power, gain, frequency, …)

[UML Diagrams of Location and AccessPoint]


Location
+ float x
+ float y
+ float z
+ Location(x: float, y: float, z: float) -> Location
+ evaluateDistance(p1:Location, p2:Location) -> Flo
AccessPoint
+ string mac_address
+ Location _location
+ float _output_power_dbm
+ float _antenna_dbi
+ float _output_frequency_h

Some algorithms I have implemented are made of two separate processes, the first one to initialize and calibrate and in a second time another process to estimate the location taking some parameters. So, for them I used some classes to hold calibrated values, this will be discussed later.

1.2 – CSV files

Primary data are stored in CSV Files. I implemented a parser that extract and process those datas. How fingerprints are represented in CSV file:

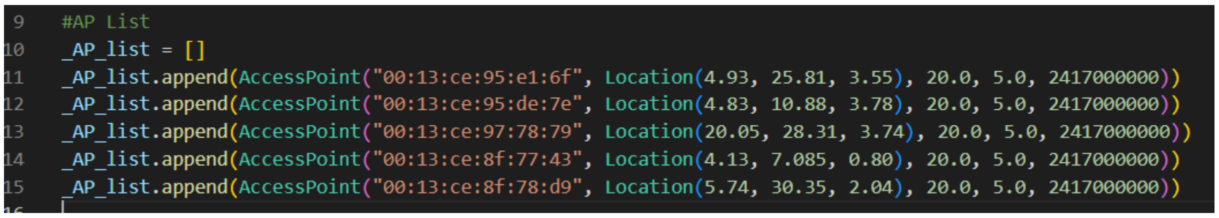
[Example for CSV File with headers]

Location										
x	y	z	Unused	MAC_1	$RSSI_1$	MAC_2	$RSSI_2$	…	MAC_n	$RSSI_n$
-4	4,8	9,6	#	00:13:ce:8f:77:43	-46	00:13:ce:8f:77:43	-61	…	00:13:ce:95:e1:6f	-73
5,4	-10	4,8	#	00:13:ce:8f:77:43	-69	00:13:ce:8f:77:43	-54	…	00:13:ce:8f:77:43	-66
7,9	4,7	-8	#	00:13:ce:95:e1:6f	-60	00:13:ce:8f:77:43	-50	…	00:13:ce:95:e1:6f	-54
…	…	…	…	…	…	…	…	…	…	…
9,3	-8	-5	#	00:13:ce:95:de:7e	-78	00:13:ce:8f:77:43	-78	…	00:13:ce:8f:78:d9	-57

The first three lines are used to store location data. The fourth one is useless for us. Then there is a list of RSSI values with their corresponding MAC address. This list doesn't have static size. Each line may have a different length. When there are multiple instances for the same MAC Address, we shall take the average of them. One fingerprint may appear multiples times, in this case, I merge fingerprint together.

Access points are given directly hard coded:

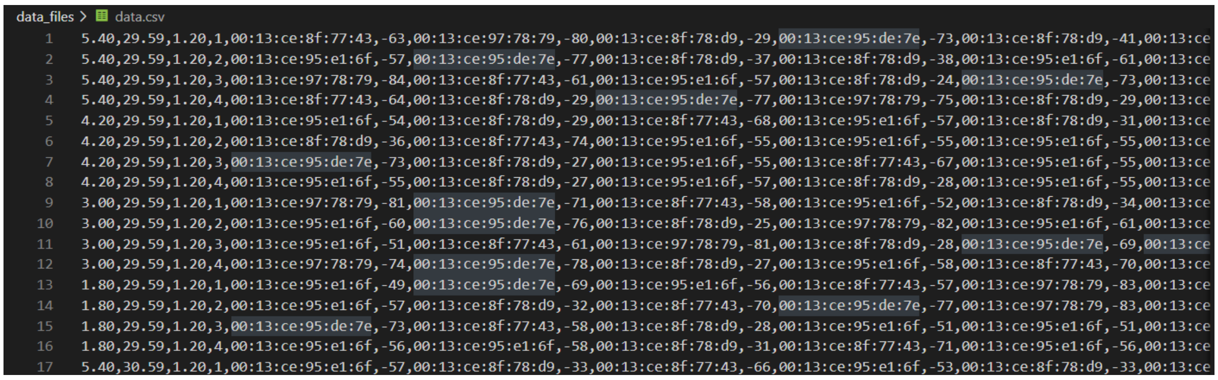
[Example of hard coded APs]



```
#AP List
_AP_list = []
_AP_list.append(AccessPoint("00:13:ce:95:e1:6f", Location(4.93, 25.81, 3.55), 20.0, 5.0, 2417000000))
_AP_list.append(AccessPoint("00:13:ce:95:de:7e", Location(4.83, 10.88, 3.78), 20.0, 5.0, 2417000000))
_AP_list.append(AccessPoint("00:13:ce:97:78:79", Location(20.05, 28.31, 3.74), 20.0, 5.0, 2417000000))
_AP_list.append(AccessPoint("00:13:ce:8f:77:43", Location(4.13, 7.085, 0.80), 20.0, 5.0, 2417000000))
_AP_list.append(AccessPoint("00:13:ce:8f:78:d9", Location(5.74, 30.35, 2.04), 20.0, 5.0, 2417000000))
```

Fingerprint to evaluate are given in another CSV file, like the precedent one. In this file, Location is given for information, and can be used to test if the algorithm works well.

[example of CSV File]



data_files > data.csv

```
5.40,29.59,1.20,1,00:13:ce:8f:77:43,-63,00:13:ce:97:78:79,-80,00:13:ce:8f:78:d9,-29,00:13:ce:95:de:7e,-73,00:13:ce:8f:78:d9,-41,00:13:ce
5.40,29.59,1.20,2,00:13:ce:95:e1:6f,-57,00:13:ce:95:de:7e,-77,00:13:ce:8f:78:d9,-37,00:13:ce:8f:78:d9,-38,00:13:ce:95:e1:6f,-61,00:13:ce
5.40,29.59,1.20,3,00:13:ce:97:78:79,-84,00:13:ce:8f:77:43,-61,00:13:ce:95:e1:6f,-57,00:13:ce:8f:78:d9,-24,00:13:ce:95:de:7e,-73,00:13:ce
5.40,29.59,1.20,4,00:13:ce:8f:77:43,-64,00:13:ce:8f:78:d9,-29,00:13:ce:95:de:7e,-77,00:13:ce:97:78:79,-75,00:13:ce:8f:78:d9,-29,00:13:ce
4.20,29.59,1.20,1,00:13:ce:95:e1:6f,-54,00:13:ce:8f:78:d9,-29,00:13:ce:8f:77:43,-68,00:13:ce:95:e1:6f,-57,00:13:ce:8f:78:d9,-31,00:13:ce
4.20,29.59,1.20,2,00:13:ce:8f:78:d9,-36,00:13:ce:8f:77:43,-74,00:13:ce:95:e1:6f,-55,00:13:ce:95:e1:6f,-55,00:13:ce:95:e1:6f,-55,00:13:ce
4.20,29.59,1.20,3,00:13:ce:95:de:7e,-73,00:13:ce:8f:78:d9,-27,00:13:ce:95:e1:6f,-55,00:13:ce:8f:77:43,-67,00:13:ce:95:e1:6f,-55,00:13:ce
4.20,29.59,1.20,4,00:13:ce:95:e1:6f,-55,00:13:ce:8f:78:d9,-27,00:13:ce:95:e1:6f,-57,00:13:ce:8f:78:d9,-28,00:13:ce:95:e1:6f,-55,00:13:ce
3.00,29.59,1.20,1,00:13:ce:97:78:79,-81,00:13:ce:95:de:7e,-71,00:13:ce:8f:77:43,-58,00:13:ce:95:e1:6f,-52,00:13:ce:8f:78:d9,-34,00:13:ce
3.00,29.59,1.20,2,00:13:ce:95:e1:6f,-60,00:13:ce:95:de:7e,-76,00:13:ce:8f:78:d9,-25,00:13:ce:97:78:79,-82,00:13:ce:95:e1:6f,-61,00:13:ce
3.00,29.59,1.20,3,00:13:ce:95:e1:6f,-51,00:13:ce:8f:77:43,-61,00:13:ce:97:78:79,-81,00:13:ce:8f:78:d9,-28,00:13:ce:95:de:7e,-69,00:13:ce
3.00,29.59,1.20,4,00:13:ce:97:78:79,-74,00:13:ce:95:de:7e,-78,00:13:ce:8f:78:d9,-27,00:13:ce:95:e1:6f,-58,00:13:ce:8f:77:43,-70,00:13:ce
1.80,29.59,1.20,1,00:13:ce:95:e1:6f,-49,00:13:ce:95:de:7e,-69,00:13:ce:95:e1:6f,-56,00:13:ce:8f:77:43,-57,00:13:ce:97:78:79,-83,00:13:ce
1.80,29.59,1.20,2,00:13:ce:95:e1:6f,-57,00:13:ce:8f:78:d9,-32,00:13:ce:8f:77:43,-70,00:13:ce:95:de:7e,-77,00:13:ce:97:78:79,-83,00:13:ce
1.80,29.59,1.20,3,00:13:ce:95:de:7e,-73,00:13:ce:8f:77:43,-58,00:13:ce:8f:78:d9,-28,00:13:ce:95:e1:6f,-51,00:13:ce:95:e1:6f,-51,00:13:ce
1.80,29.59,1.20,4,00:13:ce:95:e1:6f,-56,00:13:ce:95:e1:6f,-58,00:13:ce:8f:78:d9,-31,00:13:ce:8f:77:43,-71,00:13:ce:95:e1:6f,-56,00:13:ce
5.40,30.59,1.20,1,00:13:ce:95:e1:6f,-57,00:13:ce:8f:78:d9,-33,00:13:ce:8f:77:43,-66,00:13:ce:95:e1:6f,-53,00:13:ce:8f:78:d9,-33,00:13:ce
```

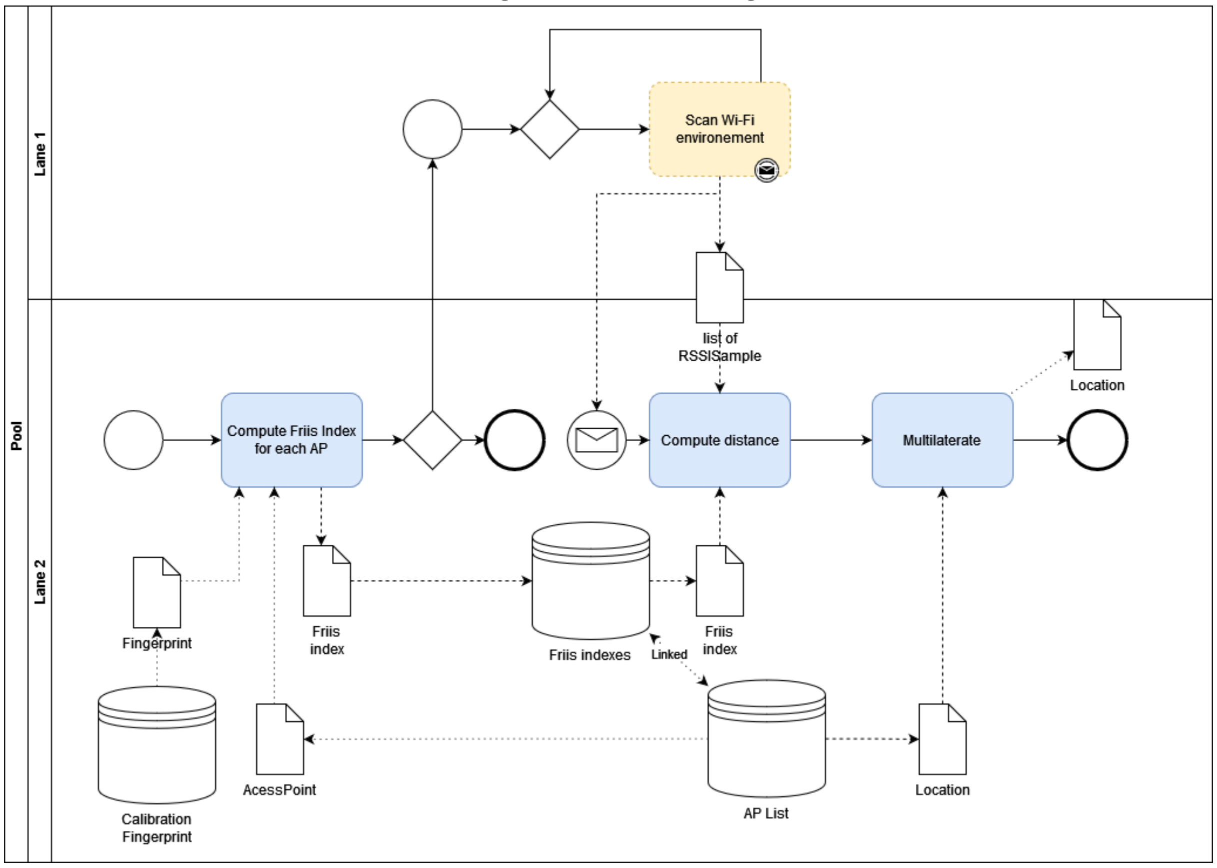
2 – FBCM Algorithm

The first algorithm use a modified version of the Friis formula:

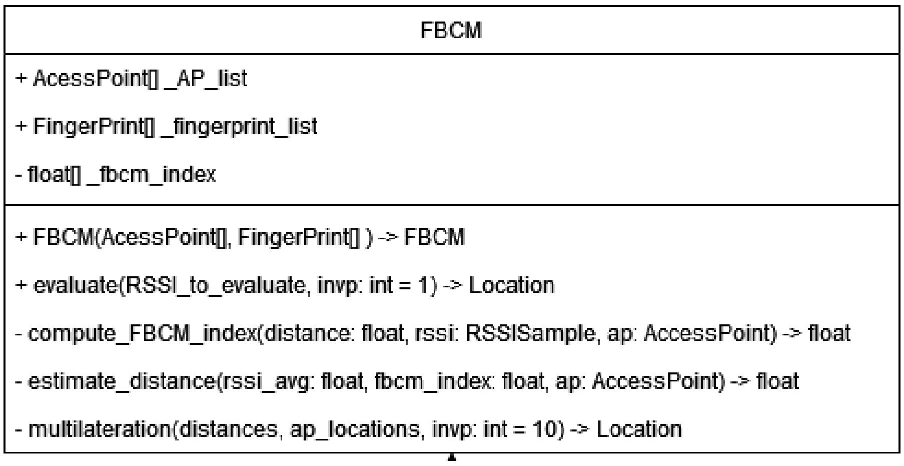
$$\frac{P_R}{P_T} = G_T G_R \left(\frac{\lambda}{4\pi}\right)^2 \frac{1}{d^i}$$

In this formula, d represents the distance between antennas, and i is the index. The index is a positive value, usually from 1 to 8, that are different for each AP. The goal of the algorithm is to use first calibration data (from which we know location and so distance between antennas) to compute this index for each AP (take the mean value when we have multiples RSSISample to calibrate one AP). And then in a second time, use this index stored in memory to compute the distance given a brand new RSSISample. We calculate it for several APs to be able to multiliterate the position of the set of RSSISample.

[BPMN Diagram of the FBCM Algorithm]



[UML Diagram of the FBCM]


FBCM

+ AcessPoint[] _AP_list

+ FingerPrint[] _fingerprint_list

- float[] _fbcm_index

+ FBCM(AcessPoint[], FingerPrint[]) -> FBCM

+ evaluate(RSSI_to_evaluate, invp: int = 1) -> Location

- compute_FBCM_index(distance: float, rssi: RSSISample, ap: AccessPoint) -> float

- estimate_distance(rssi_avg: float, fbcm_index: float, ap: AccessPoint) -> float

- multilateration(distances, ap_locations, invp: int = 10) -> Location

3 – Fingerprinting

The second approach is to store directly some fingerprints with their locations in a database. And then when we want to localize à device, one can compare his fingerprints to those which are stored in memory. In this TD we have implemented 3 kinds of matching algorithm:

- Simple matching
- Histogram matching
- Gauss matching

This approach is more precise, because it considers the topologies, and use a bigger calibration asset.

3.1 - Simple matching

Simple matching is a method that compare a fingerprint to locate directly to the dataset of calibration; and output the coordinate of the most similar fingerprint in this set. The comparison method used is the sum of difference between RSSIs with the same MAC address.

$$\sum \left|\text{RSSI}_{reference} - RSSI_{to_{compare}}\right|$$

I have decided to bound this difference by 100dBm (arbitrarily), and to add 100 to the sum for each RSSI that are not common for both fingerprints, in order to take into account, the case where fingerprints are far away. My algorithm output the location of the fingerprint with the lowest sum.

3.2 - Histogram matching by mac address

Histogram matching is quite similar to simple matching but uses probability instead of RSSI values. For each fingerprint, we convert RSSI values to normalized values.

Fingerprint example	
aa:bb:cc:dd:ee:ff	1,9
ff:ee:dd:cc:bb:aa	4,7
aa:aa:aa:aa:aa:aa	2,1

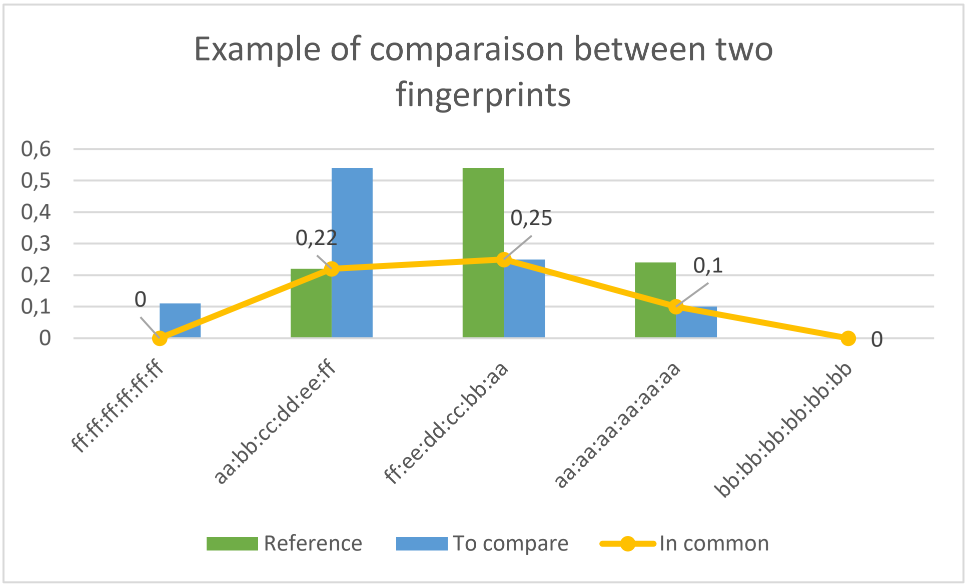
Fingerprint normalized	
Sum	8,7
aa:bb:cc:dd:ee:ff	1,9/sum = 0,22

ff:ee:dd:cc:bb:aa	4,7/sum = 0,54
aa:aa:aa:aa:aa:aa	2,1/sum = 0,24

Then, we compare fingerprint together by adding up part in common for each RSSISamples.

	Reference fingerprint	**Fingerprint to compare**	**In common**
ff:ff:ff:ff:ff:ff	0	0,11	0
aa:bb:cc:dd:ee:ff	0,22	0,54	0,22
ff:ee:dd:cc:bb:aa	0,54	0,25	0,25
aa:aa:aa:aa:aa:aa	0,24	0,1	0,1
bb:bb:bb:bb:bb:bb	0	0	0
		sum :	0,57

[Graph representing fingerprint comparison]

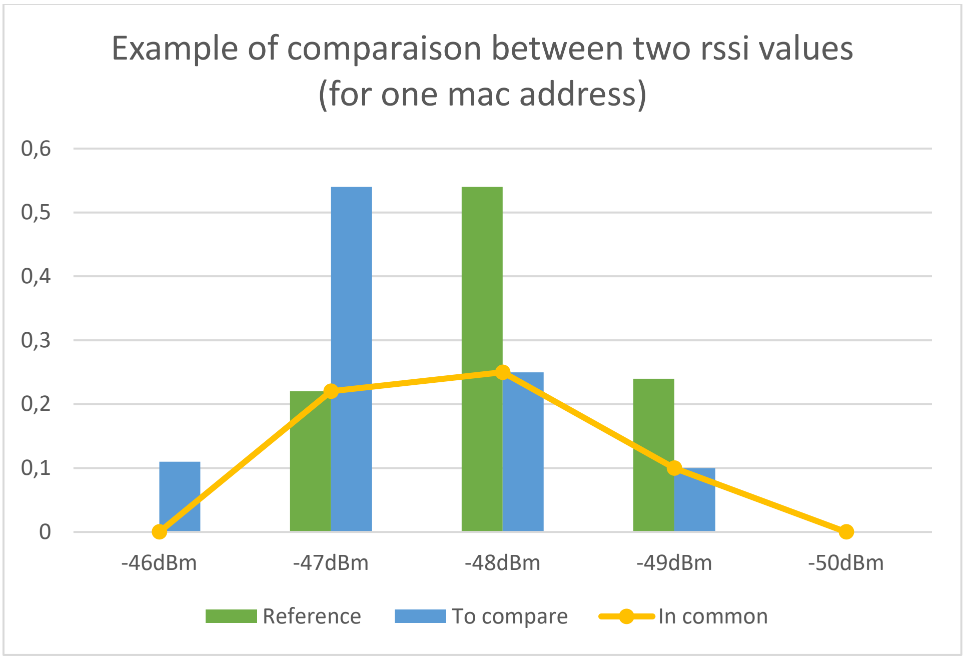


In this example, the sum is equal to 0,57 meaning that the probability of those two fingerprints to be at the same location is 57%. The algorithm output the location of the fingerprint with the higher probability.

3.3 - Histogram matching by values

This Histogram matching is the one that should have been implemented at first, comparing an histogram of data together

[Graph representing fingerprint comparison]



For this method, we then sum up all in common for each mac address and use this number as a score and we return the highest

3.3 - Gauss matching

Gauss matching uses a similar approach, but instead of using the exact normalized values, it uses recalculated ones that follows a gaussian distribution based on the characteristics of the fingerprint.

- At first the algorithm computes the average value of all RSSISamples.
- Then it computes its standard deviation.
- Finally, we use the formula to compute new values for each RSSISamples.

Formula:

$$newRSSIValue = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

x: The RSSIValue to convert

σ: The standard deviation

μ: The mean value

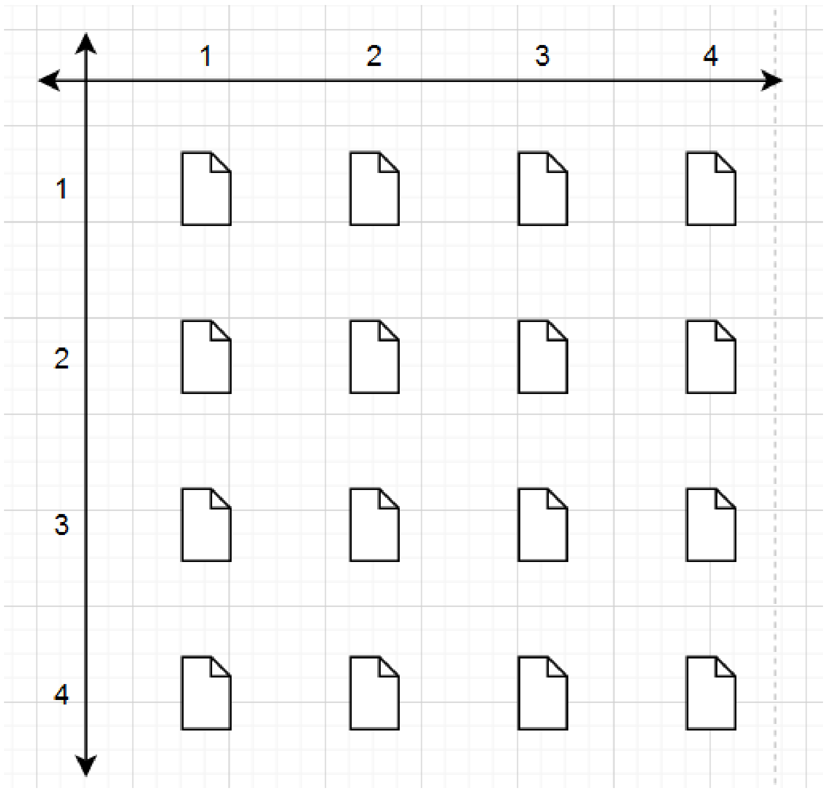
In order to get rid of side effects, we limit this formula to values that are included in

$$[\mu - 10 \; to \; \mu + 10]$$

Improvements ideas:

To improve the precision of this algorithm, I suggest using the K-nearest location with their probability in order to calculate an approximation of the position in between those locations and depending on their weights (weighted barycenter).

We start with the array of well-known fingerprints and their locations:



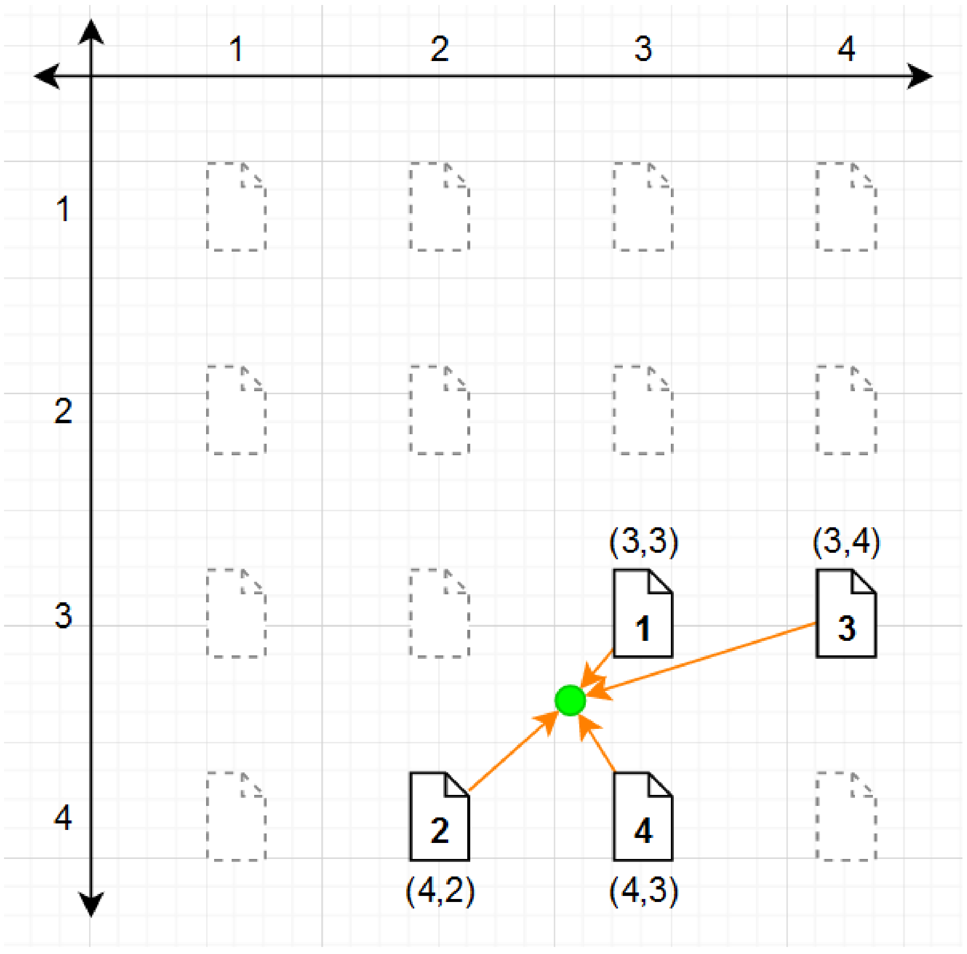
For clarity reasons, I choose to use 4x4 fingerprints perfectly aligned, but they could be more random.

First, we calculate the probability of being on a fingerprint, as done in the probabilistic approach; we get this array:

	1	2	3	4
1	0,25	0,18	0,13	0,09
2	0,24	0,13	0,6	0,41
3	0,32	0,53	0,89	0,73
4	0,4	0,89	0,62	0,56

Then we chose the K-nearest fingerprints (there in magenta the 4 nearest)

With this array, we are now able to compute the barycenter of the 4th nearest fingerprints as plotted under:



In green the estimation of the location in between those 4 fingerprints well known location.

Conclusion

Through this TP, I have discovered and manipulated some positioning methods, such as FBCM, and fingerprint matching methods. I also dealt with notions of multilateration, some signal behaviors and logarithmic calculation.

This TP also allows me to discover some challenges and problematic of those algorithms. I got familiar with notions of shifting, GDOP, time of calculus, calibration set size, precision, accuracy, etc. I also have the opportunity to enhance my algorithm to make it faster.