

Projet de semestre : *Géolocalisation dans un bus par iBeacon*



Auteur : Da Silva Andrade David

Classe : ITI-Jour 3^{ème} année 2014-2015

1 Table des matières

2	INTRODUCTION	3
3	LES MODULES IBEACONS	3
3.1	INTRODUCTION	3
3.2	LE BLUETOOTH 4.0 LE	3
3.3	LES SPECIFICATIONS DES IBEACONS	5
4	LES MODULES ESTIMOTE	6
4.1	LEURS CARACTERISTIQUES	6
4.2	LE SDK	8
5	APPLICATION N° 1 : LOCALISATION ET CONFIGURATION DES IBEACONS	10
5.1	PRESENTATION	10
5.2	L'INTERFACE ET SON UTILISATION.....	10
5.3	LES SPECIFICATIONS DU CODE.....	11
6	APPLICATION N° 2 : ACQUISITION DES BEACONS DANS DES FICHIERS CSV	12
6.1	PRESENTATION	12
6.2	LE MODELE DE STOCKAGE DES INFORMATIONS.....	12
6.3	L'INTERFACE ET SON UTILISATION.....	13
6.4	LES SPECIFICATIONS DU CODE.....	14
6.4.1	<i>Création et écriture du fichier csv</i>	<i>14</i>
6.4.2	<i>Le GPS.....</i>	<i>14</i>
7	CAPTURES DES DONNEES DANS UN BUS TPG	15
7.1	PRESENTATION	15
7.2	LES DONNEES SOUS FORME DE GRAPHIQUES	16
7.2.1	<i>Trajet n°1.....</i>	<i>16</i>
7.2.1.1	Mesure du RSSI depuis le téléphone n°1	16
7.2.1.2	Mesure de distance depuis le téléphone n°1	16
7.2.1.3	Mesure du RSSI depuis le téléphone n°2	17
7.2.1.4	Mesure de distance depuis le téléphone n°2	17
7.2.2	<i>Trajet n°2.....</i>	<i>18</i>
7.2.2.1	Mesure du RSSI depuis le téléphone n°1	18
7.2.2.2	Mesure de distance depuis le téléphone n°1	18
7.2.2.3	Mesure du RSSI depuis le téléphone n°2	19
7.2.2.4	Mesure de distance depuis le téléphone n°2	19
7.3	ANALYSE DES DONNEES	20
7.4	AMELIORATIONS POSSIBLES	21
8	APPLICATION N° 3 : SERVICE DE COLLECTE DES DONNEES ET SIMULATION	23
8.1	PRESENTATION	23
8.2	L'INTERFACE ET SON UTILISATION.....	24
8.3	SPECIFICATIONS DU CODE.....	25
8.3.1	<i>Le service Android.....</i>	<i>25</i>
8.3.2	<i>Définition d'un bus.....</i>	<i>26</i>
8.3.3	<i>Localisation.....</i>	<i>27</i>
9	LES PROBLEMES RENCONTRES.....	28
10	CONCLUSION	28
11	SOURCES	29
12	ANNEXES	29

2 Introduction

Le but de ce projet était de tester la capacité de localisation d'une personne dans un bus. Celle-ci est équipée d'un téléphone portable et le bus est lui équipé de modules iBeacons.

Pour réaliser mon projet, j'avais à ma disposition 4 Estimote iBeacons afin d'effectuer mes tests. Pour localiser les gens via leurs téléphones j'ai dû programmer des applications Android, celles-ci se comptent au nombre de 3. La première me permet de simplement scanner et programmer les iBeacons selon mon envie. La deuxième a pour but l'acquisition de données émises par les iBeacons scannés. Et finalement, la troisième est essentiellement une application de démos, elle me permet de simuler des bus en fonctions du positionnement par rapport aux iBeacons qui m'entourent.

Le projet peut être divisé en plusieurs étapes :

1. Étude des iBeacons de manière générale
2. Étude des modules Estimote iBeacons
3. Configuration des iBeacons via une application programmée
4. Programmation d'une application d'acquisition des iBeacons
5. Acquisitions en situation réelle (dans un bus des TPG)
6. Analyse des résultats des acquisitions
7. Développement d'un algorithme de lissage des acquisitions
8. Création d'un service Android implémentant cet algorithme
9. Création d'une application de démo utilisant ce service

3 Les modules iBeacons

3.1 Introduction

Les iBeacons sont comme leur nom l'indique des « balises », ils sont essentiellement utilisés pour le positionnement en intérieur. Afin d'effectuer ce positionnement, ils vont envoyer à intervalle un paquet de données permettant de les identifier.

Ils utilisent tous la technologie Bluetooth 4.0 LE, ce qui permet d'avoir de très faibles consommations et ainsi augmenter la durée de vie (plusieurs années dans certains cas) de ces balises.

3.2 Le Bluetooth 4.0 LE

Le Bluetooth 4.0 LE (Low Energy) ou connu aussi sous le nom de Bluetooth Smart, est une technologie sans fil gérée par le « Bluetooth Special Interest Group ». Il a été conçu pour de très faibles consommations d'énergie.

Le Bluetooth Smart a été initialement créé par Nokia sous le nom Wibree en 2006. En 2010, il a été fusionné dans la norme Bluetooth avec la sortie du Bluetooth version 4.0.

Il existe 3 types d'appareils Bluetooth :

- **Bluetooth:** supporte uniquement le Bluetooth "classique"
- **Bluetooth Smart Ready:** support le mode « classique » et le mode « low energy »
- **Bluetooth Smart:** supporte uniquement le mode « low energy »

Les Beacons sont uniquement Bluetooth Smart.

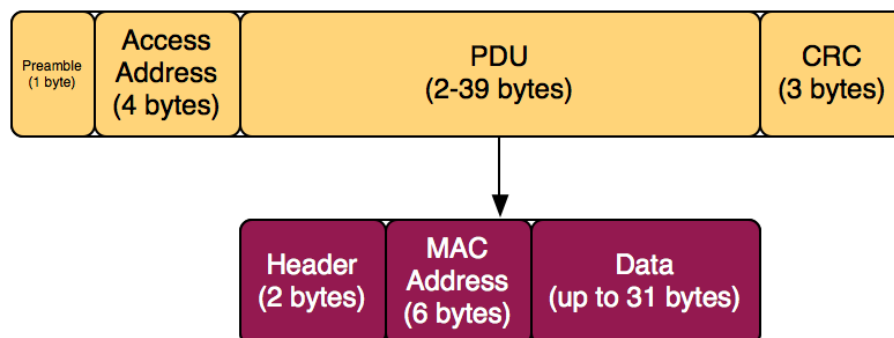
La technologie Bluetooth Smart fonctionne dans la même gamme de fréquences que la technologie Bluetooth « normale » (bande de fréquence ISM, de 2 400 à 2 483,5 MHz), mais les canaux utilisés ne sont pas les mêmes. Le Bluetooth Smart utilise deux canaux de 40 MHz chacun, contrairement au Bluetooth « normal » qui utilise 79 canaux de 1 MHz. Le débit maximum disponible est de 1 Mbit/s et la puissance d'émission maximale est de 10 mW.

Lors de l'utilisation du Bluetooth 4.0 LE avec les iBeacons deux modes sont utilisés par les iBeacons :

- Advertising
- Connecting

Le mode « Advertising » est un mécanisme de découverte unidirectionnel. Un Beacon qui désire être « découvert » envoie un paquet de données à un intervalle de temps pouvant varier de 20ms à 10 secondes. Le paquet envoyé est composé au maximum de 47 bytes :

- 1 byte de préambule
- bytes d'adresse
- 2-39 bytes d'unités de données de protocole (PDU)
- bytes de CRC



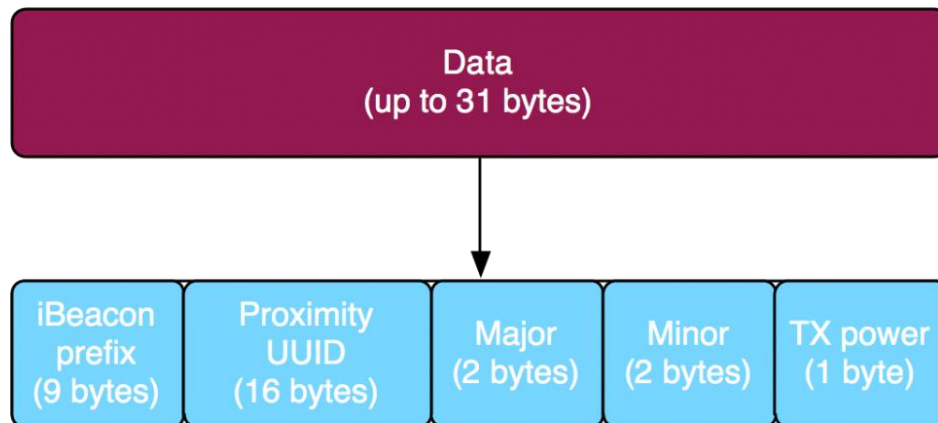
Le champ « Access Address » est toujours identique lorsqu'il s'agit du mode « Advertising », sa valeur sera de 0x8E89BED6. Lors de l'établissement d'une connexion, ce champ a une différente valeur pour chaque connexion en cours.

Pour utiliser le mode « Connecting », il faut qu'un appareil Bluetooth (un téléphone portable par exemple) soit en mode « Scanner », celui-ci ne fait qu'écouter ce qui se passe et capte tous les messages des modules en mode advertising. Une fois un appareil découvert il passe en mode « Initiator » qui va se connecter sur le Beacon qui passera en mode « Connecting ». Lorsque le mode « Connecting » est opérationnel, des données peuvent être envoyées dans les deux sens sur la connexion. Dans le cadre des modules Estimote utilisés, ce mode permet la configuration des modules ainsi que la lecture de leurs paramètres ou capteurs.

Tous les Beacons n'ont pas de mode « Connecting », ce mode est plutôt utilisé sur des Beacons qui peuvent être configurés ou bien s'ils disposent de données à envoyer à un autre appareil (par exemple, des valeurs de capteurs).

3.3 Les spécifications des iBeacons

Tous les iBeacons sont des Beacons, mais tous les Beacons ne sont pas des iBeacons. Comment peut-on définir que notre Beacon est un iBeacon ? Cette différence se trouve uniquement dans le format des données « DATA » contenues dans le PDU. Le format est défini par Apple et est composé comme ceci :



La signification de chacun de ces éléments est la suivante :

iBeacon prefix : ce champ est fixe pour tous les iBeacons du même modèle, celui-ci permet d'identifier le fabricant ainsi que les configurations Bluetooth supportées par le iBeacon. Ce champ est codé en « <len> <type> <data> », voici deux liens permettant de connaître les paramètres des flags ainsi que les codes correspondant aux fabricants :

<https://www.bluetooth.org/en-us/specification/assigned-numbers/generic-access-profile>
<https://www.bluetooth.org/en-us/specification/assigned-numbers/company-identifiers>

Proximity UUID : est un identifiant permettant de distinguer vos Beacons des autres. Prenons l'exemple d'une société de transport en commun qui désire utiliser ces iBeacons, l'entreprise aura un UUID pour tous ses iBeacons. Si ses concurrents utilisent également des iBeacons, ils auront eux aussi un UUID uniquement pour leur entreprise.

Major : est un identifiant permettant la séparation des iBeacons par « groupes ». Par exemple, toujours dans le cadre d'une compagnie de transport en commun, ça sera un identifiant pour chaque bus.

Minor : est un identifiant pour séparer individuellement chaque iBeacon. Par exemple, chaque iBeacon contenu dans le bus décrit par le Major.

TX power : correspond à la distance mesurée à 1m du iBeacon, cette valeur est fixe et est calibrée par le fabricant. Pour convertir cette valeur en dBm il suffit de faire un complément à 2. Cette valeur est très importante, elle permet de mettre une distance sur l'iBeacon en connaissant le TX power et en ayant le RSSI (Received Signal Strength Indication) qui est retourné par le module Bluetooth utilisé lors de la réception du paquet « advertising ».

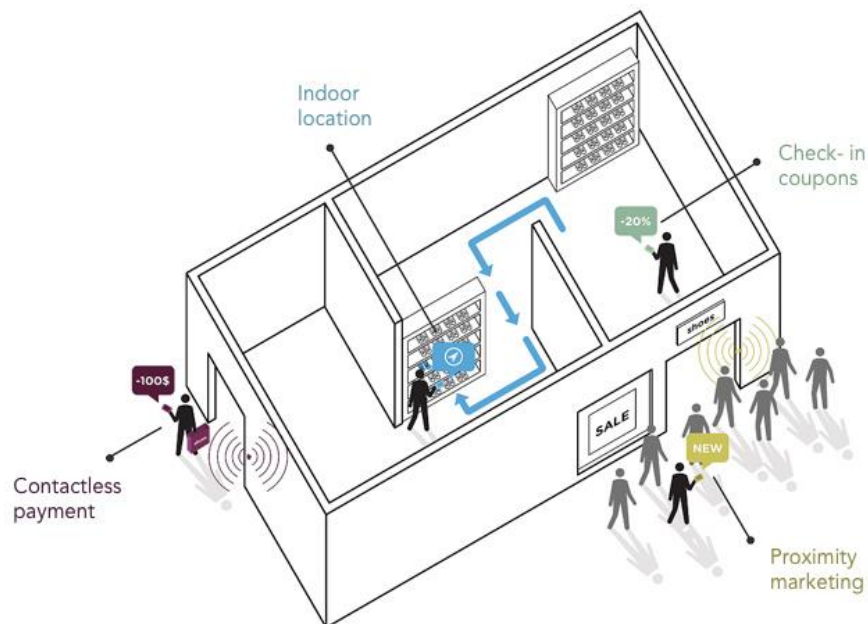
Voici un exemple de ces champs avec des valeurs

```
02 01 06 1A FF 4C 00 02 15: iBeacon prefix (fixe)
B9 40 7F 30 F5 F8 46 6E AF F9 25 55 6B 57 FE 6D: proximity UUID (Exemple
avec Estimote UUID)
00 49: major
00 0A: minor
C5: TX power en complément à 2: 0xC5 = > -59 dBm
```

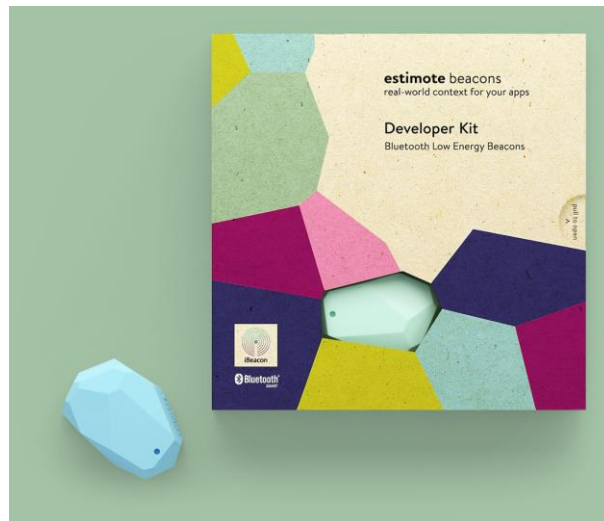
4 Les modules Estimote

4.1 Leurs caractéristiques

Les modules iBeacons proposés par Estimote sont essentiellement orientés pour la publicité, par exemple, vous vous baladez dans un magasin et lorsque vous arrivez vers le stand des chaussures une notification apparaît sur votre téléphone vous proposant une réduction sur l'un des éléments de cette zone. Une autre application est celle de la localisation dans un magasin, par exemple une personne qui cherche le rayon avec les chaussures va être guidée à travers le magasin afin d'arriver à ce rayon. Voici une image proposée par Estimote pour illustrer cela :



Il existe deux types de modules chez Estimote, le premier est celui que j'ai dans le cadre de mon projet et se nomme « Estimote Beacons », les voici en image :



Ils sont vendus par packs de 3 pour le prix de 99.99\$. Ils sont composés d'un processeur ARM® Cortex M0, ils sont également composés d'un accéléromètre ainsi qu'un capteur de température.

Si on garde leurs configurations initiales (en terme de puissance et fréquence d'émission) ils ont une autonomie de 3 ans, celle-ci peut être prolongée jusqu'à 5+ ans en modifiant leurs paramètres.

Leur portée est d'environ 70m, mais celle-ci dépend essentiellement de l'environnement dans lequel le Beacon est utilisé.

Comme présentée précédemment la norme « iBeacon » ne permet pas d'envoyer des données autres que le UUID, Major, Minor ainsi que la puissance à 1m du Beacon. Ce qui signifie que si on veut recevoir les données de l'accéléromètre ou bien du capteur de température une connexion Bluetooth classique est nécessaire.

Estimote propose également des « Stickers » qui ne sont pas des iBeacon, mais de simples Beacons, car ils n'utilisent pas des identifiants UUID, minor et major. Les voici en image :



Leur batterie n'est supposée tenir que 1 an et leur portée est de 10m avec la configuration par défaut. Néanmoins ils sont beaucoup plus petits que les Estimote iBeacons et ils coûtent moins cher (10 stickers coûtent 99.99\$). Comme la norme iBeacon n'est pas utilisée, les

données broadcastées peuvent directement contenir les valeurs des capteurs ainsi que d'autres informations, en voici la liste :

- sticker type (par exemple réfrigérateur, chien, ordinateur, etc.)
- sticker ID (valeur numérique)
- orientation dans l'espace
- déplacement en X, Y, Z
- température
- puissance d'émission à 1m (utilisé pour calculer la distance)
- niveau de la batterie
- version du firmware

Les stickers sont plutôt adaptés à la localisation d'objets particuliers qui sont susceptibles d'être déplacés car ils sont peu encombrants, alors que les Estimote Beacons sont adaptés à des positions fixes.

4.2 Le SDK

Estimote fournit un SDK permettant la détection des Beacons ainsi que leur configuration.

Le SDK est disponible pour Android et iOS, néanmoins le SDK pour iOS est celui qui a les plus de fonctionnalités implémentées à l'heure actuelle comme par exemple la connexion authentifiée sur les iBeacons pour la configuration.

Le SDK apporte un lot de classes très utiles :

- **Beacon** : ceci est une classe qui permet la description complète d'un iBeacon (UUID, minor, major, etc...).
- **BeaconService** : ce service tourne en arrière-plan et permet de récupérer tous les iBeacons. Il est accessible via la classe BeaconManager. Celui-ci doit être autorisé dans le manifeste de l'application via cette entrée :

```
<!-- Estimote service responsible for scanning beacons. -->
<service
    android:name="com.estimote.sdk.service.BeaconService"
    android:exported="false" />
```

- **BeaconManager** : cette classe implémente un « setRangingListener » qui permet la mise en place d'un écouteur sur la découverte d'iBeacons, voici un code illustrant cela :

```
// Configure BeaconManager.
beaconManager = new BeaconManager(MainActivity.this);
beaconManager.setRangingListener(new BeaconManager.RangingListener() {
    @Override
    public void onBeaconsDiscovered(Region region, final List<Beacon> beacons) {
        // Note that results are not delivered on UI thread.
        runOnUiThread(new Runnable() {
            public void run() {
                getActionBar().setSubtitle("Found beacons: " + beacons.size());
            }
        });
    }
});
});
```


Toutes les 1 seconde environ (temps minimum pour le « ServiceBeacon »), la méthode « onBeaconsDiscovered » est appelée automatiquement avec en paramètre une liste de Beacons se nommant « beacons » qui contient tous les iBeacons détectés lors du dernier scan. Elle crée ensuite un thread dans lequel l'utilisateur est libre d'utiliser la liste comme bon lui semble.

Pour lancer le scan, il faut tout d'abord activer le service avec la méthode « startRanging » à laquelle on passe en argument la région qu'on désire scanner. Cette région est composée d'un UUID, d'un major et d'un minor. Si on désire scanner n'importe quel iBeacon, on peut tout mettre à « null » comme dans l'exemple ci-dessous :

```
Region ALL_ESTIMATE_BEACONS_REGION = new Region("Reg1", null, null, null);  
beaconManager.startRanging(ALL_ESTIMATE_BEACONS_REGION);
```

Une fois le scan lancé il tourne en permanence. Si on désire l'interrompre, on utilise la méthode « stopRanging » en lui passant à nouveau la région :

```
beaconManager.stopRanging(ALL_ESTIMATE_BEACONS_REGION);
```

- **Utils.computeAccuracy** : cette méthode n'est disponible que sur Android et elle permet l'estimation de la distance entre le téléphone et l'iBeacon. Néanmoins cette valeur est beaucoup moins précise sur Android que sur iOS. Ceci est principalement dû à l'implémentation de la mesure du RSSI sur Android qui n'est pas aussi efficace que celle sur iOS. Ceci est expliqué par Estimote sur leur blog officiel :

<https://community.estimote.com/hc/en-us/articles/202028913-Why-Beacons-are-less-responsive-on-Android-than-on-iOS>

La raison pour laquelle cette méthode n'est pas implémentée sur le SDK pour iOS c'est simplement qu'Apple fournit directement une librairie qui permet la location au sein de son système. Celle-ci se nomme iOS CoreLocation, plus d'informations à ce sujet peuvent être trouvées dans ce topique sur stackoverflow :

<https://stackoverflow.com/questions/21338031/radius-networks-ibeacon-ranging-fluctuation>

Voici les liens pour télécharger le SDK Android et iOS ainsi que des exemples d'utilisation:

<https://github.com/Estimote/Android-SDK>

<https://github.com/Estimote/iOS-SDK>

5 Application n° 1 : localisation et configuration des iBeacons

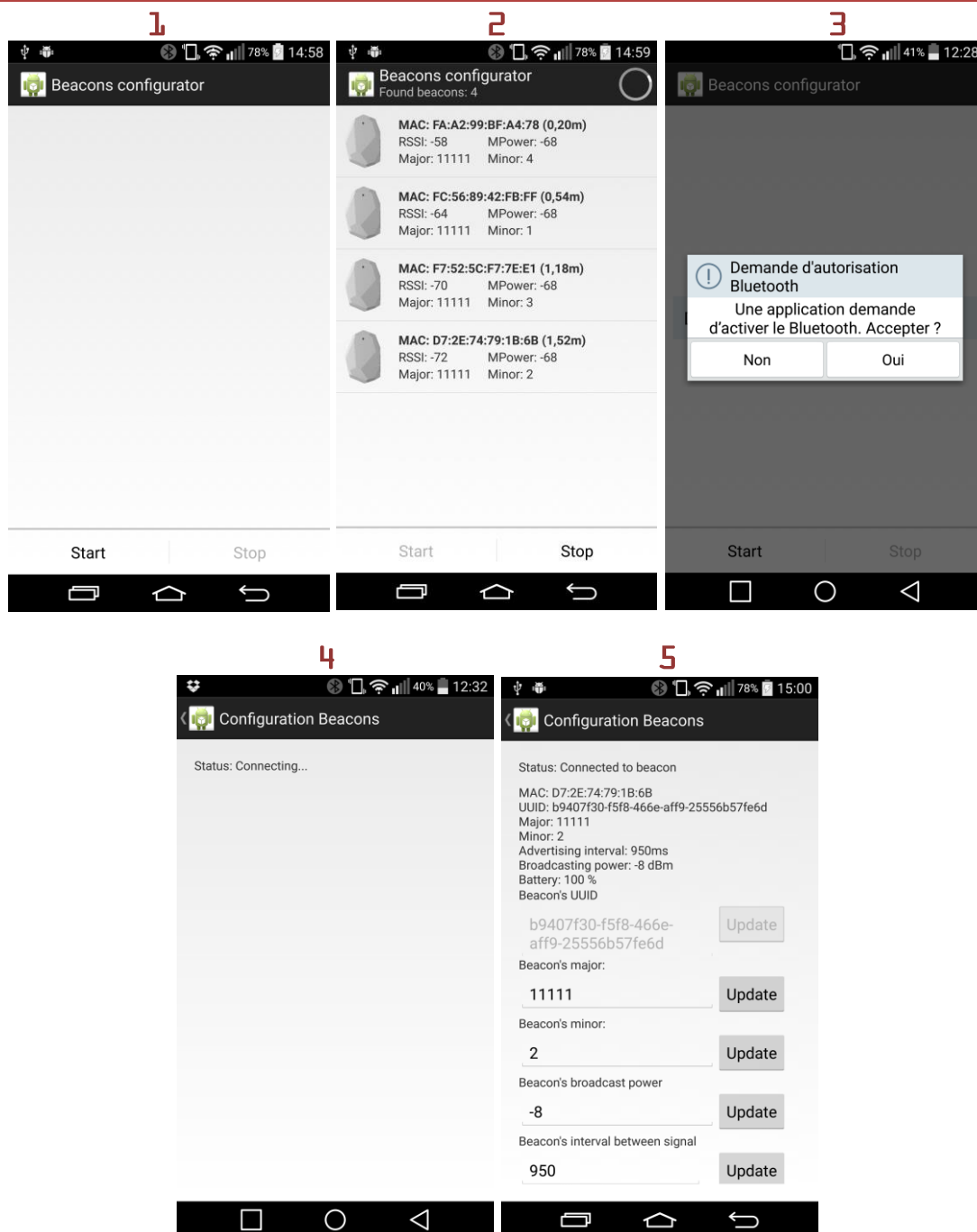
5.1 Présentation

Cette application permet de lister tous les iBeacons détectés par le téléphone sous forme d'une liste. Si l'utilisateur appuie sur l'un iBeacon une connexion est initialisée vers celui-ci est une nouvelle activité (page) apparaît permettant à l'utilisateur d'éditer tous les paramètres du iBeacon (Major, Minor, broadcast power, etc....).

Cette application repose sur une démo proposée par Estimote avec l'ajout de nouvelles fonctionnalités. Les démos peuvent être retrouvées à cette adresse :

<https://github.com/Estimote/Android-SDK/tree/master/Demos>

5.2 L'interface et son utilisation



1. La vue de l'application au lancement de celle-ci. Pour lancer le scan de tous les iBeacons à proximité il suffit d'appuyer sur le bouton « Start »
2. Ce message est affiché uniquement lorsque l'on appuie sur le bouton « Start » et que le Bluetooth sur l'appareil n'est pas activé. Si l'utilisateur refuse d'activer le Bluetooth, le scan ne se lance pas et l'application retourne à l'affichage numéro 1. Si le Bluetooth est déjà activé, cet affichage n'apparaîtra pas.
3. Le scan est en cours, tous les iBeacons détectés sont affichés au centre dans l'ordre du plus proche au plus éloigné. Si on désire modifier un paramètre d'un iBeacon, il suffit d'appuyer sur celui-ci.
4. Si l'utilisateur appuie sur un iBeacon présent dans la liste, l'application tentera de se connecter à celui-ci. Si la connexion échoue, l'utilisateur se voit revenir à l'affichage numéro 3, mais si celle-ci est un succès l'affichage passera au numéro 5.
5. Dans cette page l'utilisateur peut directement modifier les valeurs des paramètres de l'iBeacon. Pour cela il lui suffit de rentrer les valeurs dans les champs prévus à cet effet puis d'appuyer sur le bouton « Update » correspondant. Si les valeurs sont correctes et si l'envoi de celles-ci est correctement effectué, un « Toast » s'affichera avec un message de confirmation.

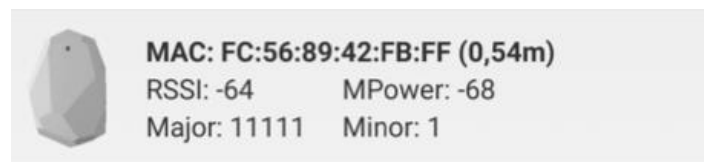
5.3 Les spécifications du code

Cette application repose une démo proposée par Estimote avec ensuite l'ajout de nouvelle fonctionnalité, ce code peut être retrouvé à cette adresse :

<https://github.com/Estimote/Android-SDK/tree/master/Demos>

J'ai essentiellement repris 2 classes de la démo d'Estimote:

- **LeDeviceListAdapter** : cette classe me permet d'afficher une liste d'iBeacons sous forme d'éléments graphiques qui sont placés dans une « ListView ». La structure des éléments de la liste est contenue dans un fichier nommé « device_item.xml ». Voici le résultat d'un élément dans cette liste :



- **ConfigActivity** : cette activité est celle appelée lorsque l'on désire modifier les valeurs d'un iBeacon qui est affiché dans la liste de l'activité principale. Elle contient principalement des méthodes qui sont appelées lorsque l'on appuie sur les boutons pour actualiser les données sur le périphérique.

Ce code m'a essentiellement permis de comprendre le fonctionnement du SDK. J'ai simplement rajouté quelques nouvelles options par rapport à la démo originale, deux boutons (start et stop) ainsi que plus de valeurs modifiables dans la fenêtre de configuration (la démo ne proposait que la modification du minor).

6 Application n° 2 : acquisition des Beacons dans des fichiers CSV

6.1 Présentation

Cette application a pour but la capture des données provenant des iBeacons détectés, les données choisies sont les suivantes :

- Minor
- RSSI (puissance du signal reçu sur le téléphone)
- Temps depuis le début de la capture (en ms)
- Distance en mètre fournie par le SDK Estimote
- Longitude GPS
- Latitude GPS

6.2 Le modèle de stockage des informations

Ces données sont stockées dans un fichier csv sous le format suivant :

Minor ;RSSI;Temps_ms;Distance_m;Longitude;Latitude


Ce qui nous donne ceci une fois des valeurs récoltées:

```
2;-59;387942;0.25;46.2131815;6.1306105
2;-66;389011;0.74;46.2131815;6.1306105
2;-58;390095;0.2;46.2131815;6.1306105
2;-61;391145;0.33;46.2128429;6.1314002
2;-66;392235;0.74;46.2128429;6.1314002
```

Le nom du fichier est choisi selon ce modèle :

beacon_UUID_Major_Minor.csv

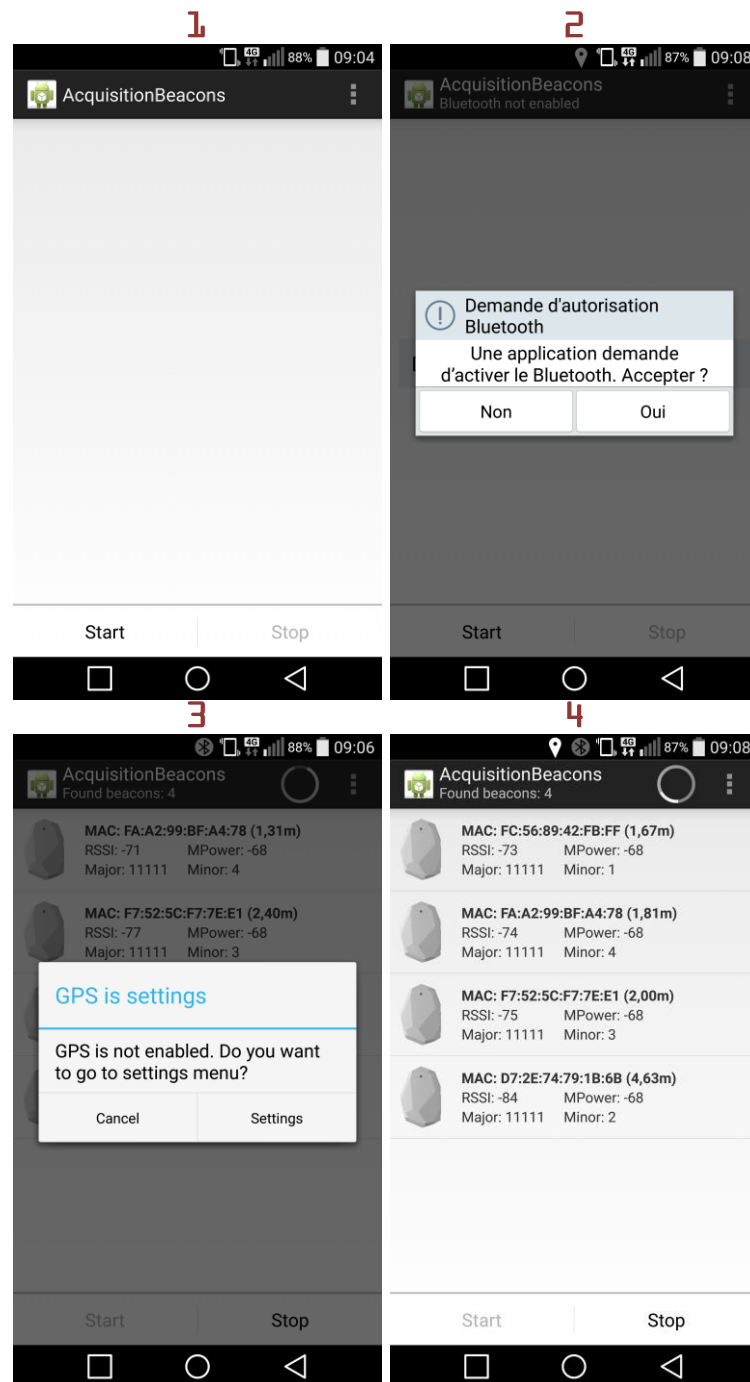
Ce qui nous donne ceci :



beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_1.csv
beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_2.csv
beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_3.csv
beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_4.csv

Ces fichiers sont ensuite stockés dans un dossier contenant la date et l'heure du lancement de la capture.

6.3 L'interface et son utilisation



- 1) La vue de l'application au lancement de celle-ci. Pour lancer l'acquisition de tous les iBeacons à proximité il suffit d'appuyer sur le bouton « Start »
- 2) Ce message est affiché uniquement lorsque l'on appuie sur le bouton « Start » et que le Bluetooth sur l'appareil n'est pas activé. Si l'utilisateur refuse d'activer le Bluetooth, le scan ne se lance pas et l'application retourne à l'affichage numéro 1. Si le Bluetooth est déjà activé, cet affichage n'apparaîtra pas.
- 3) Ce message est affiché uniquement lorsque l'on appuie sur le bouton « Start » et que le GPS sur l'appareil n'est pas activé. Si l'utilisateur refuse d'activer le GPS, l'acquisition se

poursuit, mais les données GPS seront à « 0.0 ». Si le GPS est déjà activé, cet affichage n'apparaîtra pas.

- 4) Le scan est en cours, tous les iBeacons détectés sont directement sauvegardés dans les fichiers csv dans un répertoire sur le téléphone. Le chemin par défaut du répertoire est normalement « /storage/emulated/0/BEACONS ».

6.4 Les spécifications du code

6.4.1 Création et écriture du fichier csv

À chaque fois qu'un scan est effectué, les iBeacons détectés sont enregistrés dans leurs fichiers respectifs. Pour cela j'ai écrit une classe nommée « SaveBeaconOnFile » permettant l'enregistrement de ces données via une méthode nommée « write » prenant en paramètre une collection de « Beacon » ainsi que l'objet de type GPSTracker.

Avant de lancer la première mesure, il faut appeler la méthode « startCapture » qui permet d'enregistrer le temps auquel la capture a commencé.

L'écriture sur Android est similaire à une simple application Java sur PC. Par exemple, on peut utiliser un objet de type « PrintWriter » avec sa méthode « println ». Ce qui change c'est le chemin d'accès au répertoire, sur Android le chemin d'accès des répertoires est stocké dans des variables d'environnement, en voici la liste ainsi que les méthodes permettant de les récupérer :

<http://developer.android.com/reference/android/os/Environment.html>

Dans mon cas j'ai utilisé la méthode « getExternalStorageDirectory() » pour trouver le chemin du répertoire de sauvegarde. Le mot « External » ne signifie pas forcément qu'il s'agit d'une carte SD sur le téléphone, cela peut être le cas lorsque celle-ci est connectée, mais si ce n'est pas le cas c'est simplement un répertoire qui est émulé sur la mémoire flash du téléphone. Par exemple, mon téléphone ne dispose pas de carte SD, le répertoire pointé par « getExternalStorageDirectory() » est donc « /storage/emulated/0/ ».

6.4.2 Le GPS

Pour créer ma classe GPSTracker, j'ai essentiellement suivi ce tutoriel qui me permet de simplement implémenter le GPS dans mon application :

<http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/>

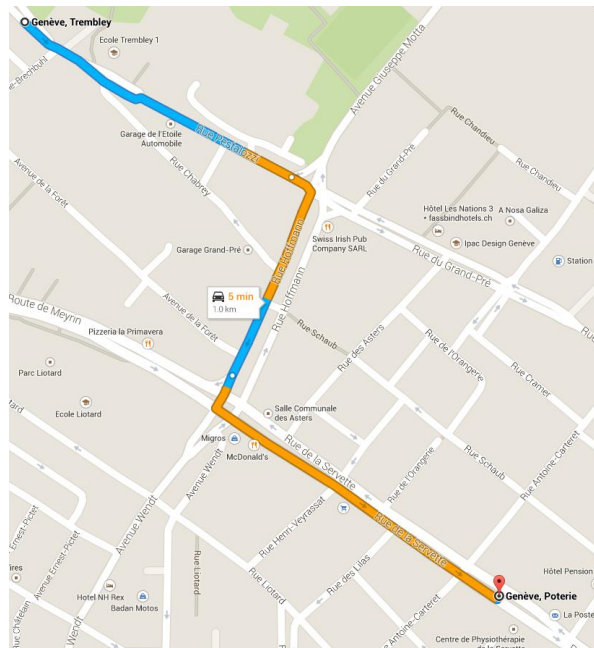
Cette classe contient une suite de méthodes très utiles dans mon projet :

- canGetLocation() : test si le GPS est activé sur le téléphone, si ce n'est pas le cas demande à l'utilisateur de l'activer.
- getLocation() : actualise les données GPS. Cette méthode doit être appelée avant la lecture des valeurs.
- getLatitude() : retourne la latitude lors de la dernière actualisation des valeurs.
- getLongitude() : retourne la longitude lors de la dernière actualisation des valeurs.

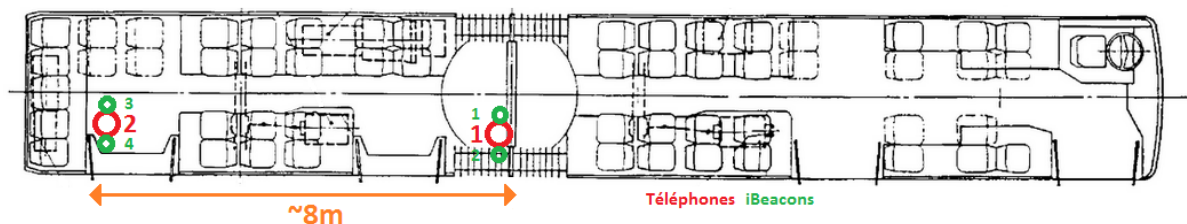
7 Captures des données dans un bus TPG

7.1 Présentation

En reprenant l'application numéro 2 je me suis rendu dans un bus TPG avec un collègue afin d'effectuer des acquisitions en situations réelles. Nous avons pris le bus 3 depuis l'arrêt Poterie jusqu'à l'arrêt Trembley. Nous avons ensuite effectué le chemin inverse dans la même ligne, voici un dessin du chemin effectué :

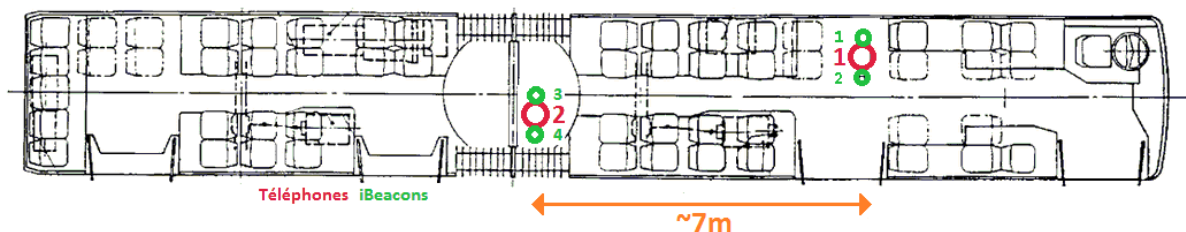


Nous avons tous les deux un téléphone portable ainsi que 2 iBeacons. Voici un schéma représentant notre position lors du premier trajet :



Le téléphone numéro 1 avait à côté de lui les iBeacons avec le minor à 1 et à 2. Le deuxième téléphone était lui avec le minor 3 et 4.

Pour le deuxième trajet nous avons changé de place dans le bus :

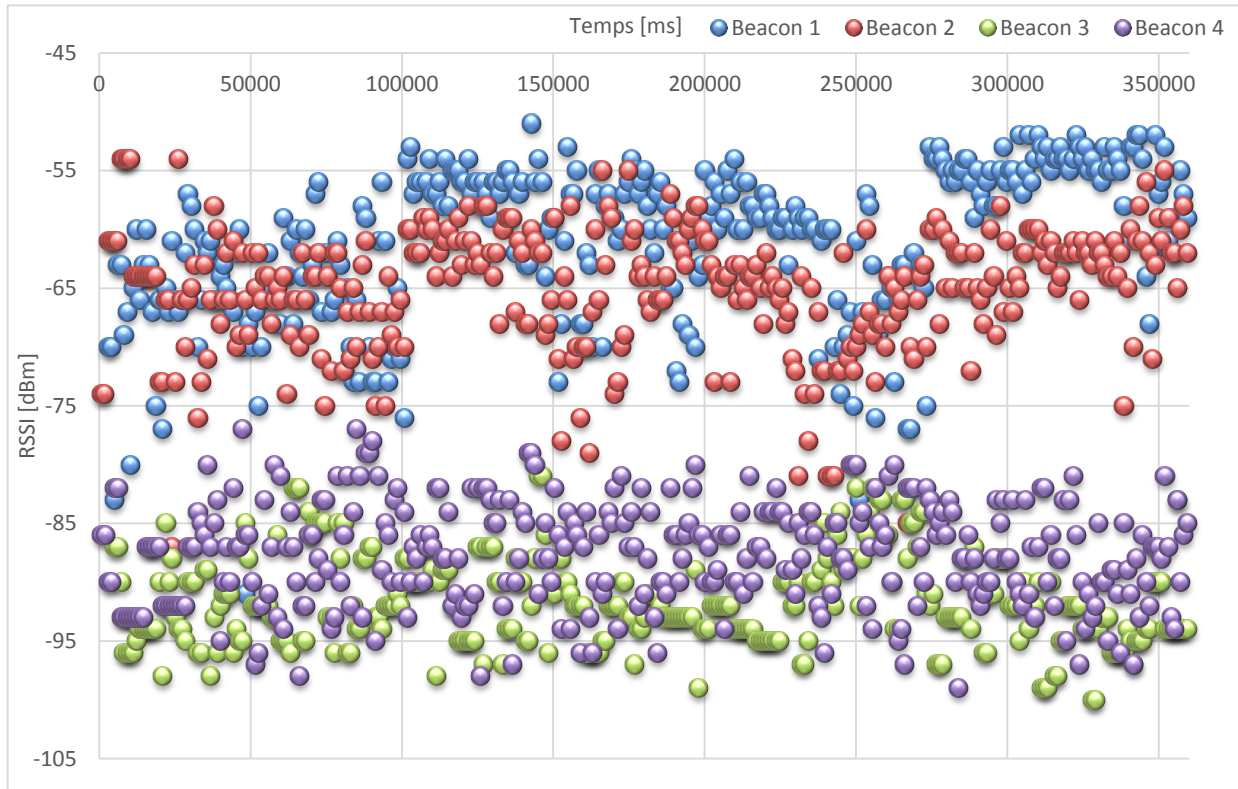


Les distances entre les deux téléphones sont estimées, car il était impossible de les mesurer sur place. Je me suis donc aidé des schémas du bus ainsi que sa taille totale (18.7m) afin d'estimer la distance entre les deux téléphones.

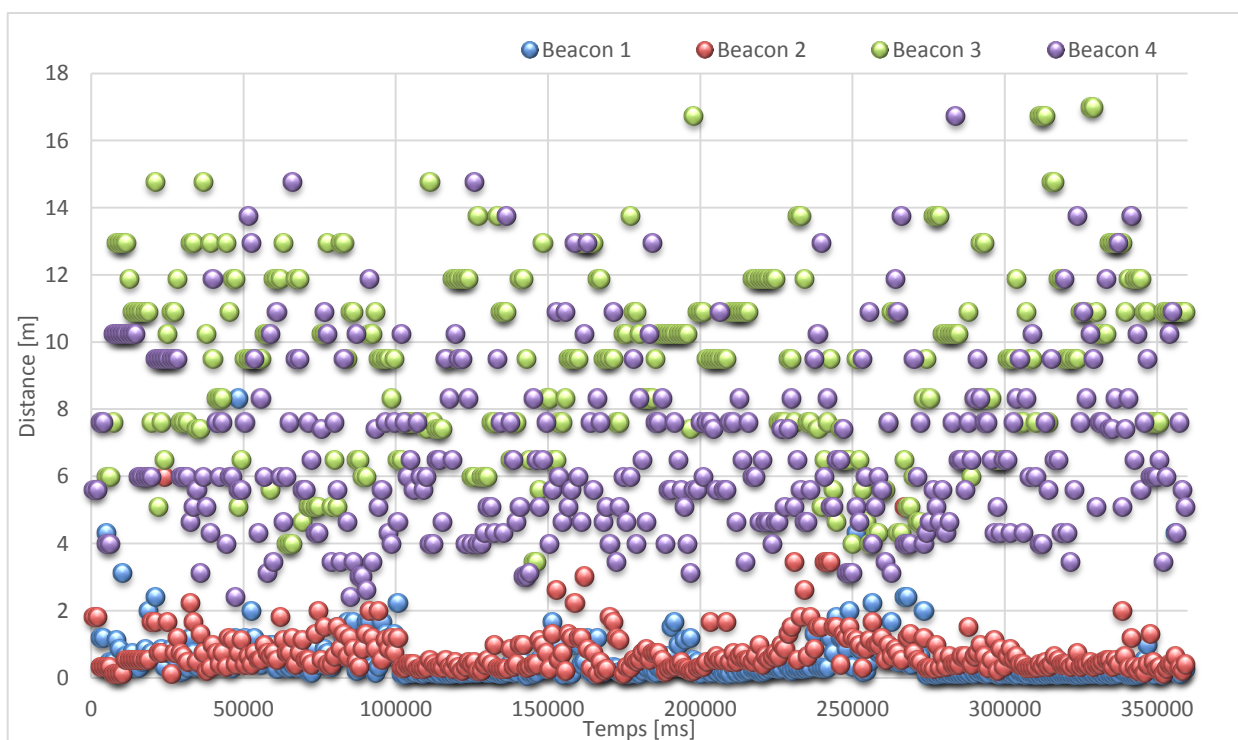
7.2 Les données sous forme de graphiques

7.2.1 Trajet n°1

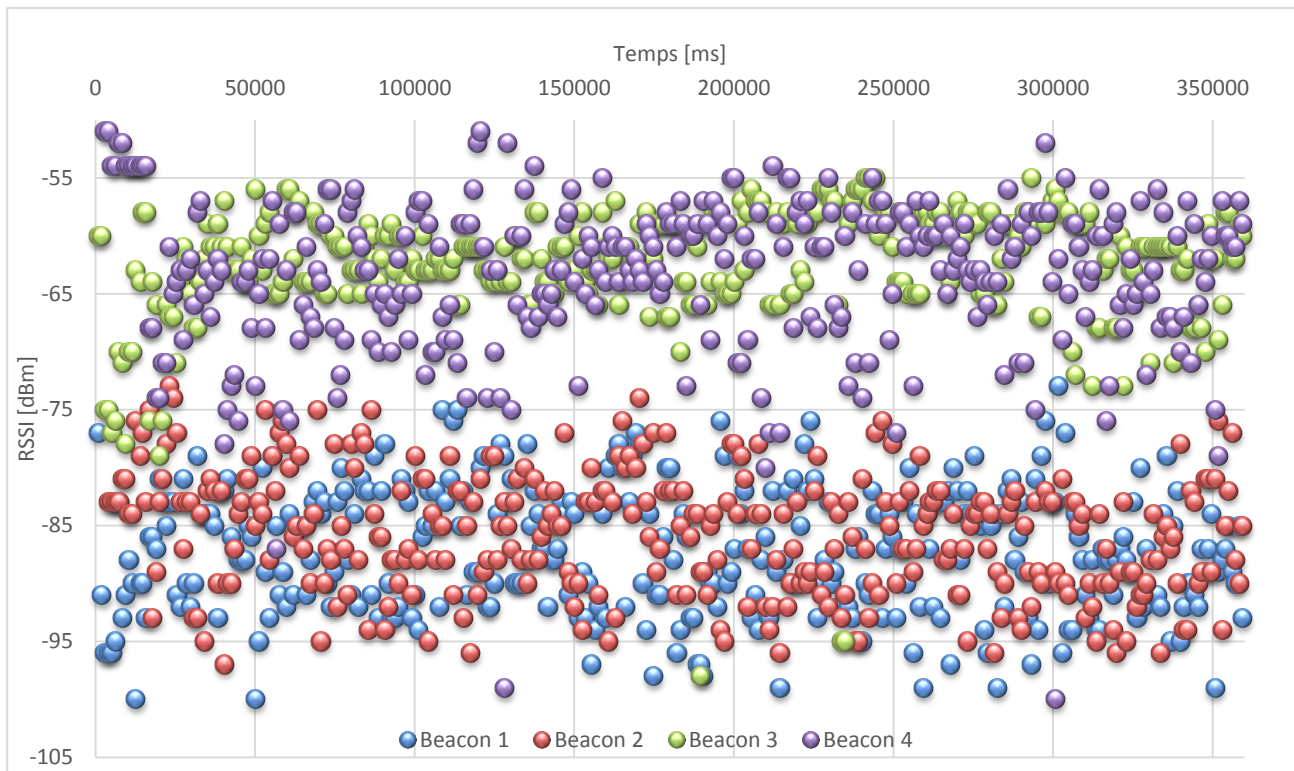
7.2.1.1 Mesure du RSSI depuis le téléphone n°1



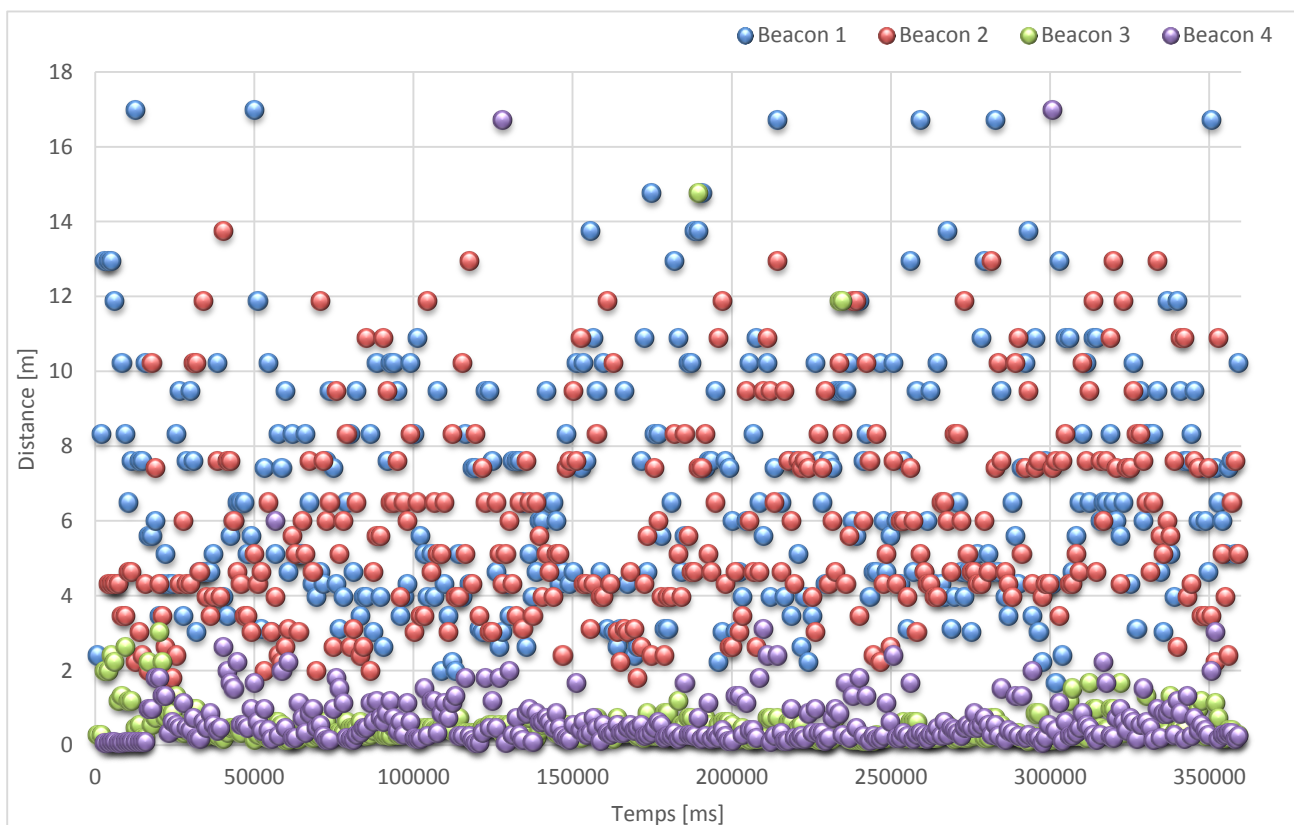
7.2.1.2 Mesure de distance depuis le téléphone n°1



7.2.1.3 Mesure du RSSI depuis le téléphone n°2

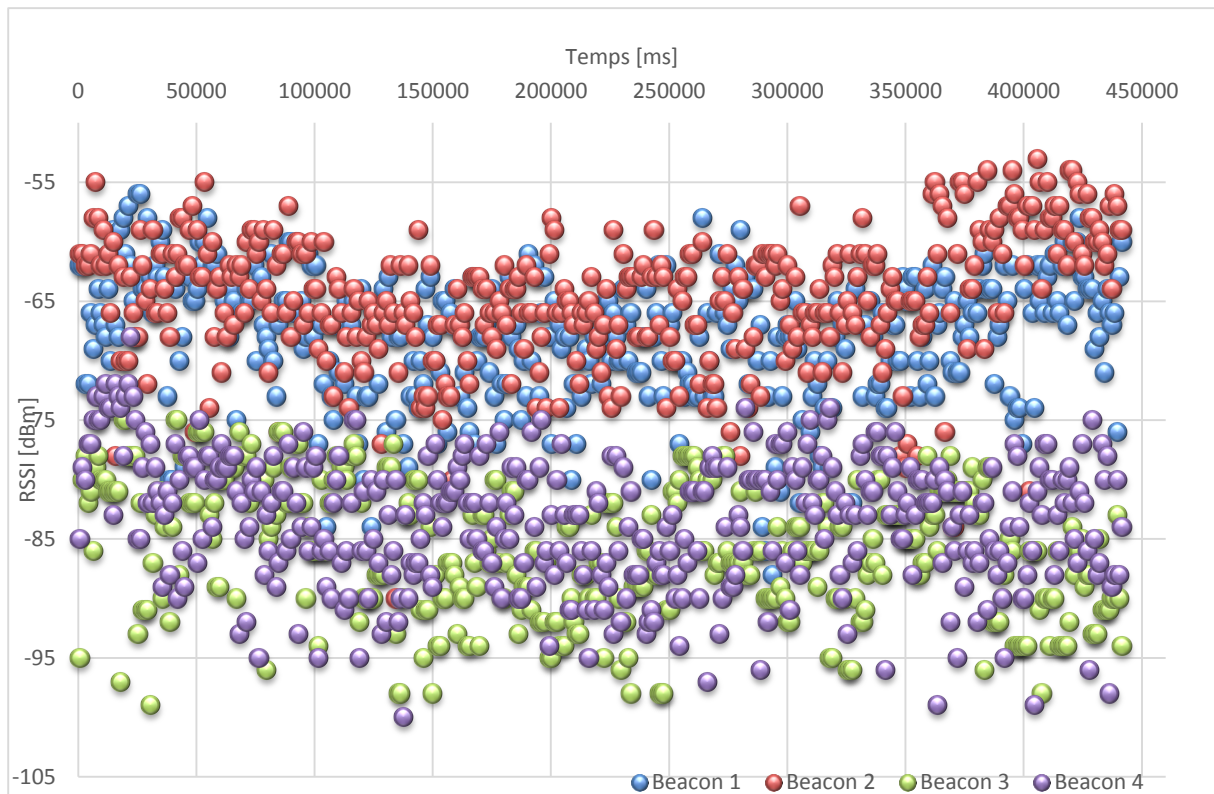


7.2.1.4 Mesure de distance depuis le téléphone n°2

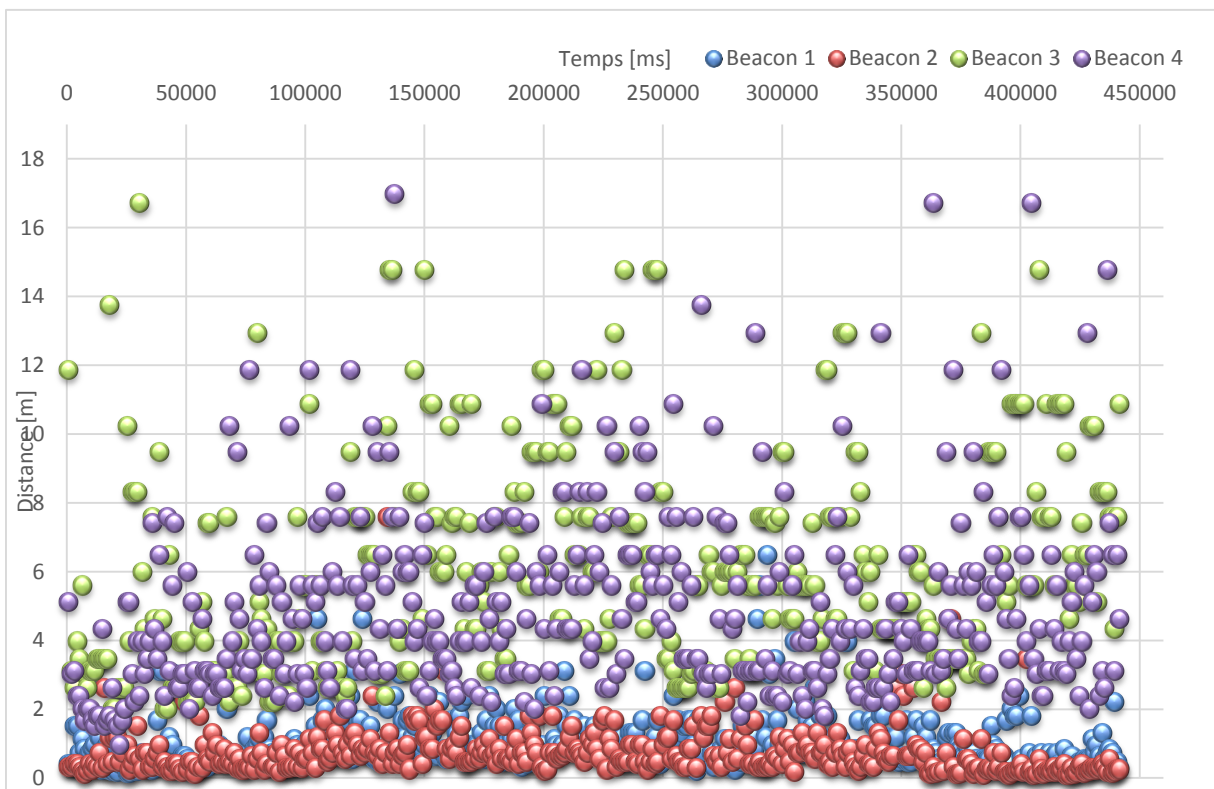


7.2.2 Trajet n°2

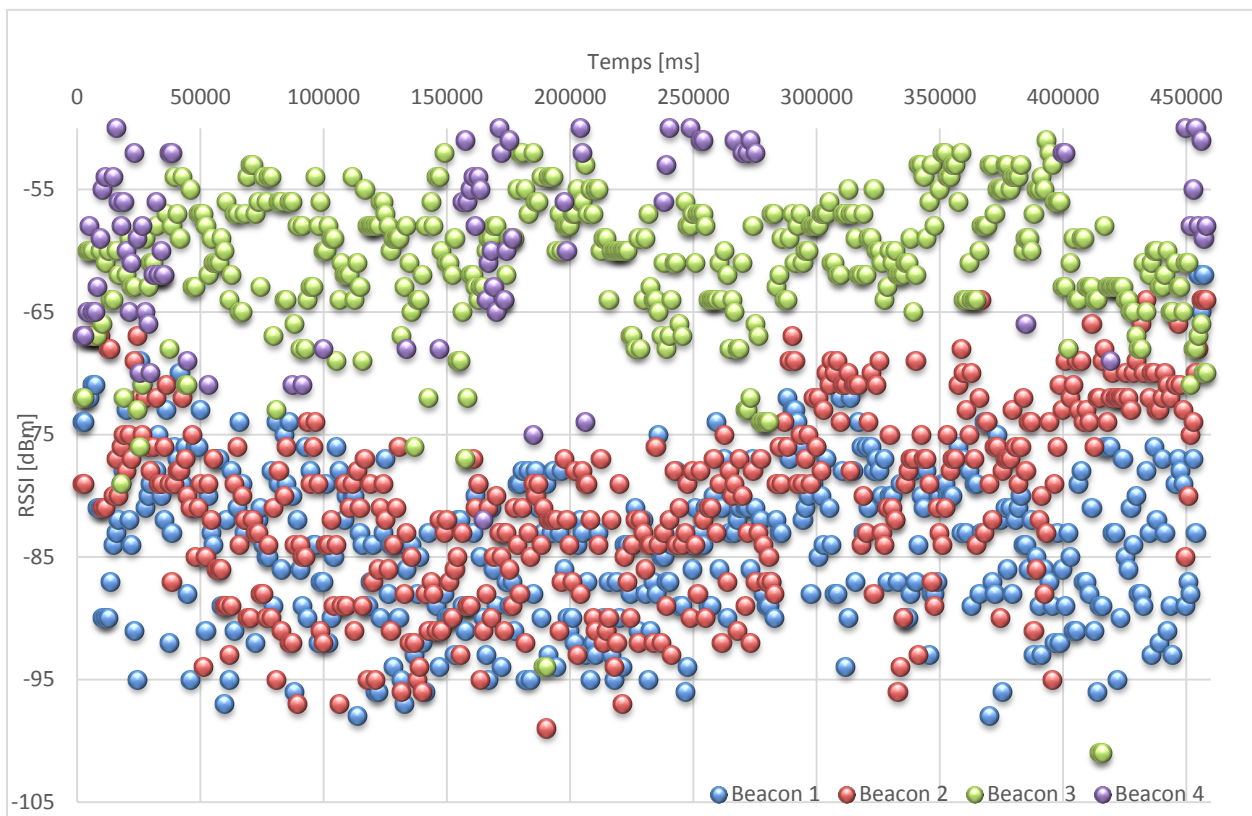
7.2.2.1 Mesure du RSSI depuis le téléphone n°1



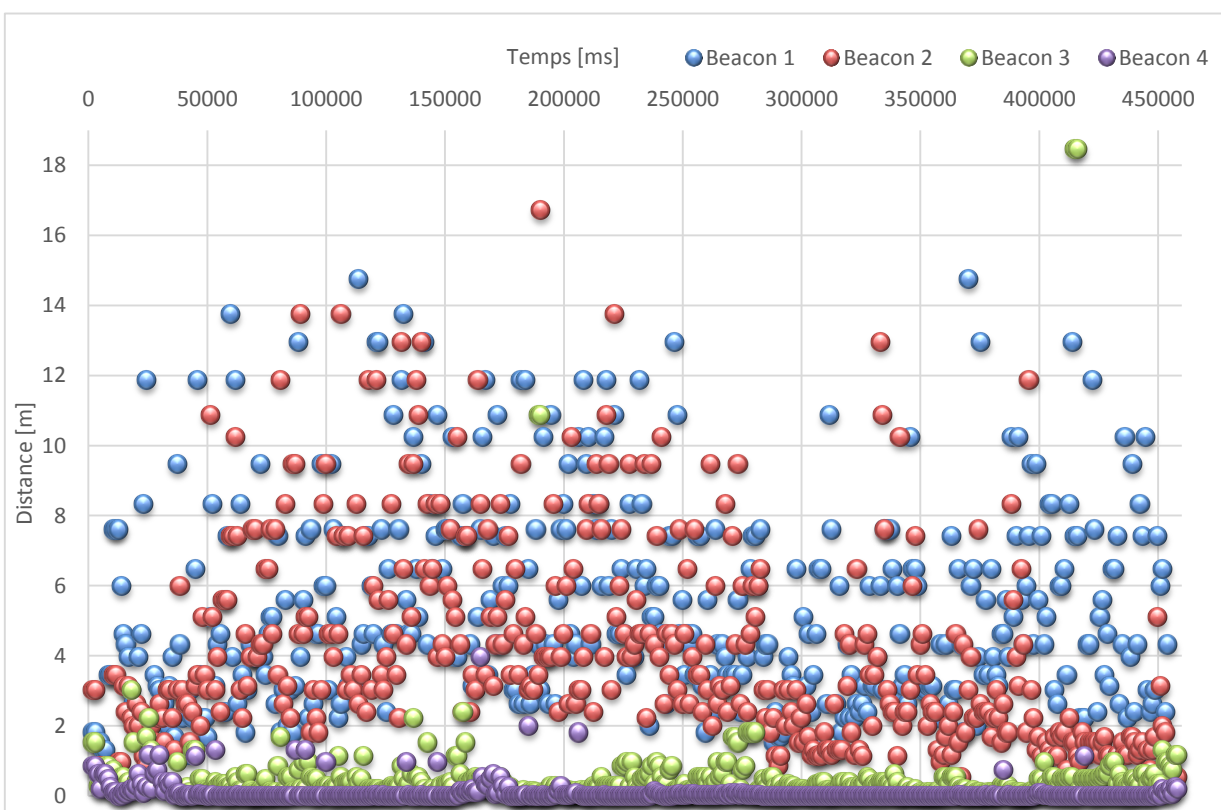
7.2.2.2 Mesure de distance depuis le téléphone n°1



7.2.2.3 Mesure du RSSI depuis le téléphone n°2



7.2.2.4 Mesure de distance depuis le téléphone n°2



7.3 Analyse des données

L'une des premières choses qu'on constate en regardant les graphiques c'est l'inconsistance des données malgré le fait que la distance réelle (à l'intérieur du bus) ne change pas. Avant de faire les relevés, j'avais déjà quelques hypothèses sur la variation des valeurs reçues sur les téléphones, les voici :

1. Les gens à l'intérieur du bus. En effet, lorsque les mesures ont été effectuées c'était à l'heure de pointe, le bus était bondé et les gens peuvent très facilement atténuer le signal. Ceci est essentiellement dû au fait qu'un corps humain est composé en moyenne à 65% de cet élément. Estimote nous dit même que sur son site Internet que le signal peut être diffracté, interféré ou absorbé par l'eau (y compris le corps humain), voici le lien expliquant cela :

<http://estimote.com/api/>

2. Les autres signaux présents dans l'environnement. Par autres signaux je parle essentiellement de ceux à 2.4Ghz, comme les Wifi qui sont très présents en ville.
3. La précision du RSSI sur les téléphones Android. Comme expliqué précédemment dans ce rapport, la mesure du RSSI sur Android n'est pas aussi précise que sur un appareil Apple, ceci est même avancé comme explication par le constructeur Estimote pour la fluctuation des données :

<https://community.estimote.com/hc/en-us/articles/202028913-Why-Beacons-are-less-responsive-on-Android-than-on-iOS>

Si maintenant on reprend les hypothèses avec les données des captures réalisées :

1. Cette hypothèse reste tout à fait possible, pour pouvoir analyser plus profondément cette hypothèse l'idéal serait de faire le même trajet, mais cette fois-ci dans un bus vide et ensuite d'analyser les données obtenues.
2. L'hypothèse des autres signaux est possible, mais néanmoins peu probable, simplement à cause du fait que les lignes de bus empruntées se trouvent essentiellement au milieu de grandes rues où les réseaux wifi arrivent déjà très atténués. Néanmoins, pour pouvoir complètement ignorer cette théorie ça serait d'effectuer un parcours dans un bus dans un endroit sans ce type d'interférence, une route de campagne par exemple.
3. Cette hypothèse est difficilement contestable avec les résultats obtenus, simplement à cause du fait qu'aucun des deux téléphones ne tournait sur iOS. Cela serait très intéressant de faire les mêmes trajets, mais cette fois-ci avec un téléphone iOS et l'autre sur Android pour simplement pouvoir comparer les résultats. Malheureusement, mes connaissances en programmation mobiles sont actuellement limitées à Android, mais cela serait une expérience intéressante à réaliser.

7.4 Améliorations possibles

On constate dans les mesures que ce qui dérange le plus ce sont les points qui font des « sauts ». Par exemple, lors de la mesure de distance depuis le téléphone n°2 on a 2 points qui se retrouvent à 18.5m alors qu'ils se trouvent en réalité à moins d'un mètre. J'ai donc décidé de créer une fenêtre médiane, cela signifie que pour une mesure au temps 1 je prends les valeurs de 1 jusqu'à « w », « w » étant la taille de la fenêtre. J'applique ensuite la médiane sur ces valeurs afin de n'avoir qu'un seul point. Je n'ai pas opté pour la moyenne, car les variations des « sauts » seraient beaucoup trop importantes sur le résultat.

J'ai donc simulé ceci grâce à Matlab avec le code suivant :

```
clear all
close all

% Read all files
beacon1 = dlmread('./beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_1.csv', ';');
beacon2 = dlmread('./beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_2.csv', ';');
beacon3 = dlmread('./beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_3.csv', ';');
beacon4 = dlmread('./beacon_b9407f30-f5f8-466e-aff9-25556b57fe6d_11111_4.csv', ';');

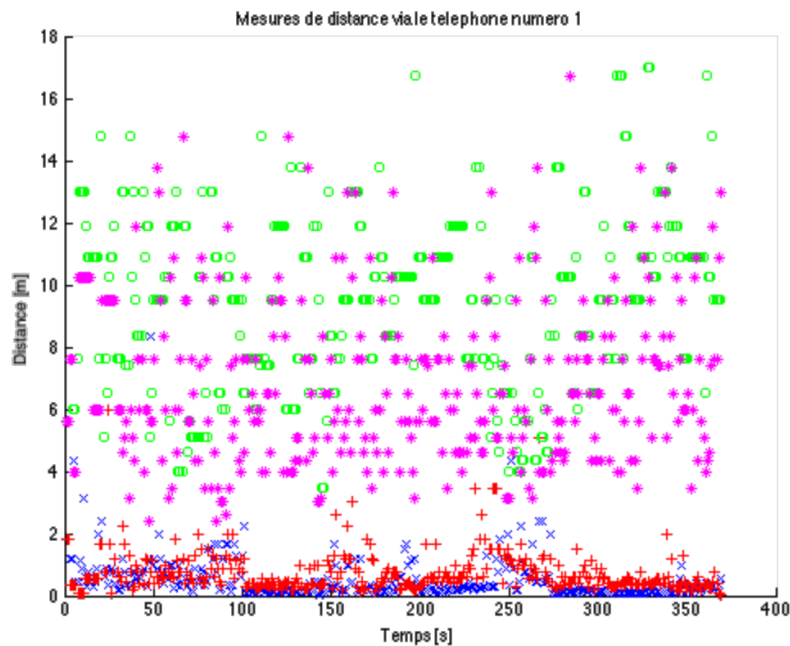
% Plot pre filter
figure
hold on
title('Mesures de distance via le telephone numero 1')
xlabel('Temps [s]')
ylabel('Distance [m]')
plot(beacon1(:,3)/1000,beacon1(:,4), 'bx') % plot distances
plot(beacon2(:,3)/1000,beacon2(:,4), 'r+') % plot distances
plot(beacon3(:,3)/1000,beacon3(:,4), 'go') % plot distances
plot(beacon4(:,3)/1000,beacon4(:,4), 'm*') % plot distances
hold off

% Window's size
w_size = 20

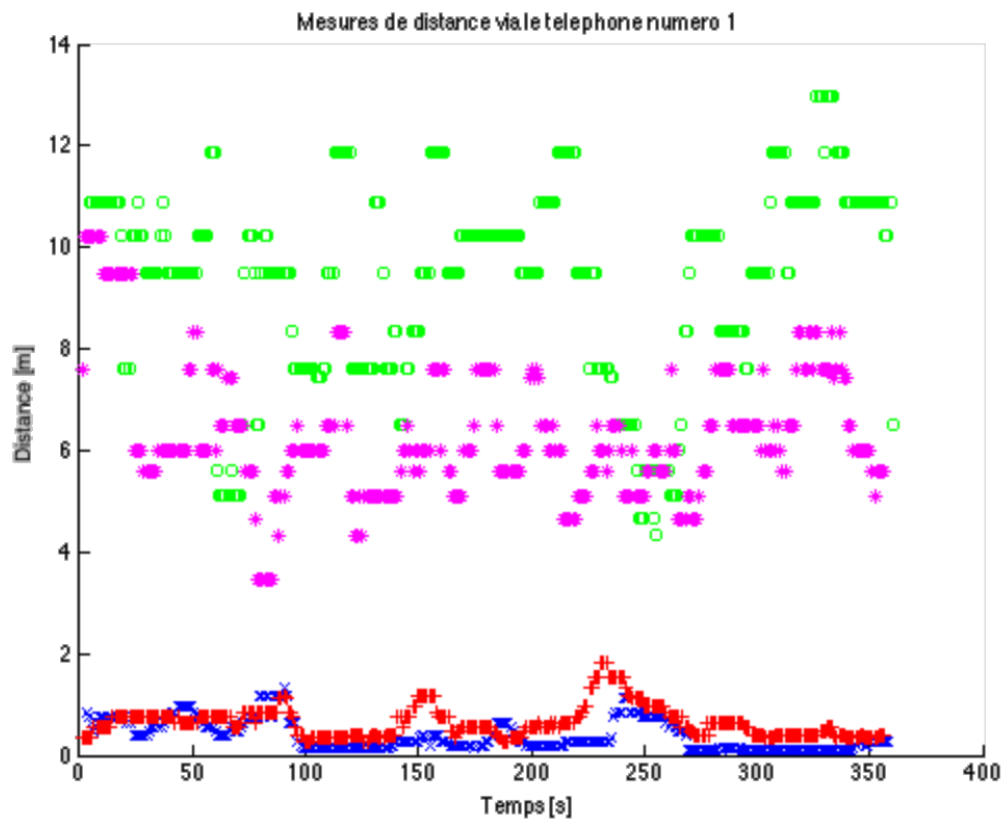
for i = 1 : size(beacon1, 1)-w_size
    beacon1(i,4) = median(beacon1([i:i+w_size],4));
end
for i = 1 : size(beacon2, 1)-w_size
    beacon2(i,4) = median(beacon2([i:i+w_size],4));
end
for i = 1 : size(beacon3, 1)-w_size
    beacon3(i,4) = median(beacon3([i:i+w_size],4));
end
for i = 1 : size(beacon4, 1)-w_size
    beacon4(i,4) = median(beacon4([i:i+w_size],4));
end

% Plot post filter
figure
hold on
title('Mesures de distance via le telephone numero 1')
xlabel('Temps [s]')
ylabel('Distance [m]')
plot(beacon1(1:length(beacon1)-w_size,3)/1000,beacon1(1:length(beacon1)-w_size,4), 'bx')
% plot distances
plot(beacon2(1:length(beacon1)-w_size,3)/1000,beacon2(1:length(beacon1)-w_size,4), 'r+')
% plot distances
plot(beacon3(1:length(beacon1)-w_size,3)/1000,beacon3(1:length(beacon1)-w_size,4), 'go')
% plot distances
plot(beacon4(1:length(beacon1)-w_size,3)/1000,beacon4(1:length(beacon1)-w_size,4), 'm*')
% plot distances
hold off
```

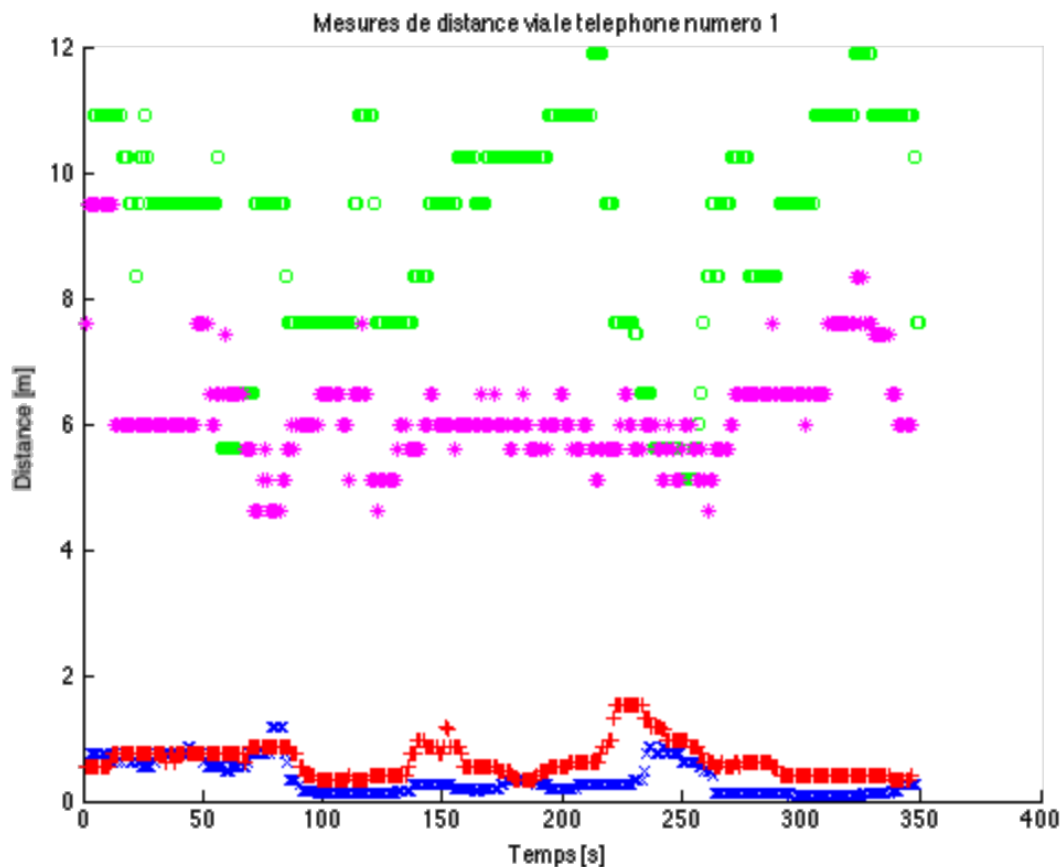
J'ai appliqué ce code sur les mesures effectuées lors du premier trajet pour la mesure de la distance sur le téléphone n°1. L'exécution du code me retourne deux graphiques, le premier est le résultat sans traitement des données :



Voici le résultat avec les données filtrées par une fenêtre de 10 données :



Voici le résultat avec les données filtrées par une fenêtre de 20 données :



On constate une nette amélioration des valeurs avec beaucoup moins de sauts qui rendaient la localisation extrêmement difficile auparavant.

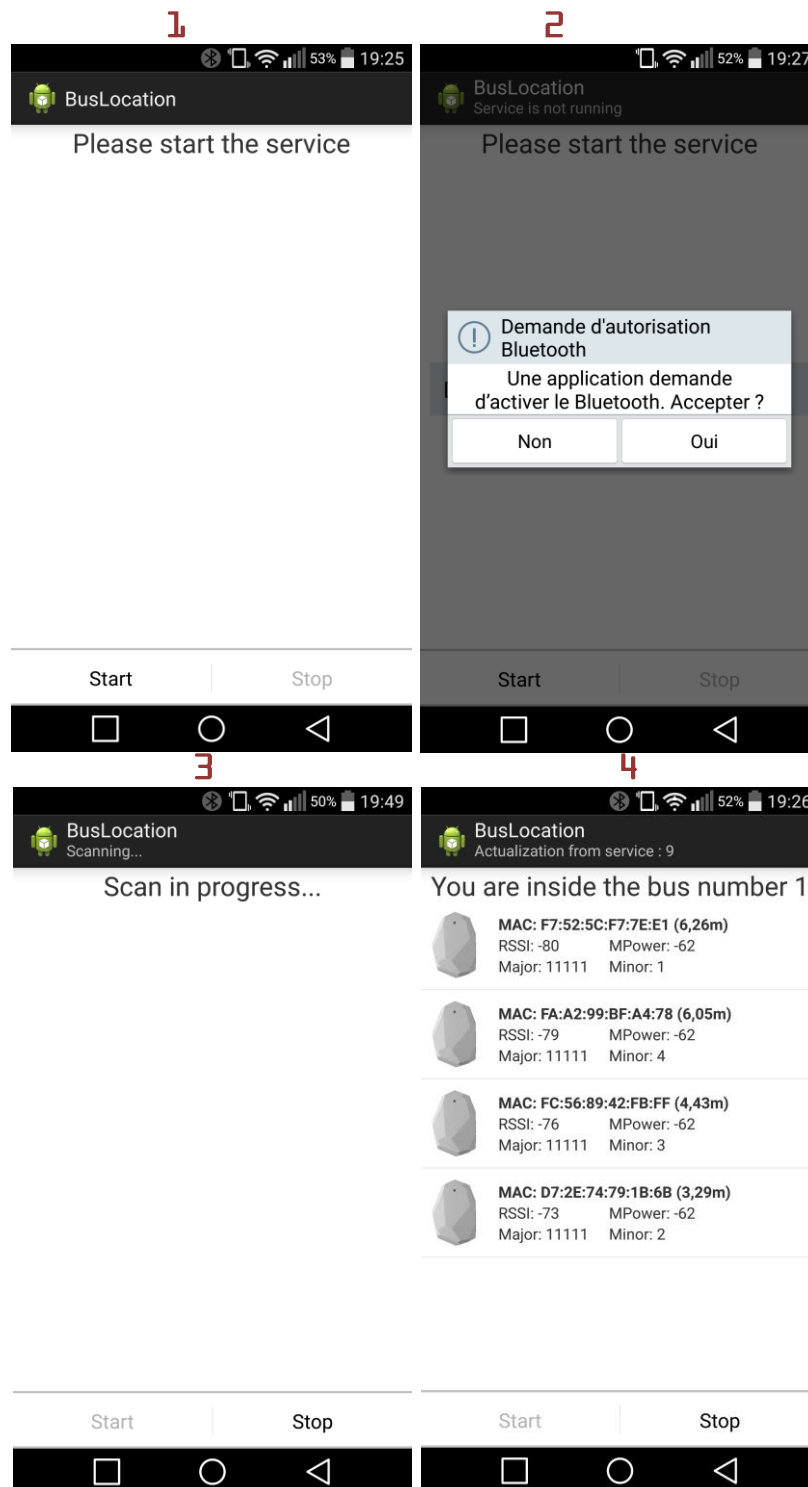
8 Application n° 3 : service de collecte des données et simulation

8.1 Présentation

Cette application a pour but la localisation dans un bus équipé de deux iBeacons. Celle-ci implémente un service qui applique le « lissage » médian exposé au chapitre précédent. Afin de faire en sorte que l'utilisateur n'a pas besoin de toujours avoir son téléphone déverrouillé, j'ai décidé de créer ce lissage sous forme de service Android, celui-ci tourne tant que le processus de l'application principale est actif.

L'application implémente néanmoins un algorithme très simple afin de savoir dans quel bus on se trouve.

8.2 L'interface et son utilisation



1. Au lancement de l'application, le service n'est pas lancé, pour ce faire il suffit d'appuyer sur le bouton « Start ».
2. Ce message est affiché uniquement lorsque l'on appuie sur le bouton « Start » et que le Bluetooth sur l'appareil n'est pas activé. Si l'utilisateur refuse d'activer le Bluetooth, le service ne se lance pas et l'application retourne à l'affichage numéro 1. Si le Bluetooth est déjà activé, cet affichage n'apparaîtra pas.

3. Quand le scan est en cours, cela signifie que le service est en train de récupérer suffisamment de données pour pouvoir effectuer le lissage des valeurs. Le service par défaut est configuré à 30 mesures, cela signifie qu'il faut attendre environ 30 secondes avant d'avoir l'affichage numéro 4.
4. Une fois que les iBeacons s'affichent, cela signifie que le service a suffisamment de données pour pouvoir effectuer la médiane. Les iBeacons affichés sont la médiane des 30 dernières mesures effectuées par le service.
La « TextView » affiche dans quel bus l'utilisateur se trouve en fonction des bus stockés dans l'application.

8.3 Spécifications du code

8.3.1 Le service Android

Sur Android un service est simplement une activité qui n'a pas d'interface accessible à l'utilisateur. Il existe plusieurs types de service, dans mon cas j'ai opté pour un service de type « broadcast », cela signifie que lorsque le service souhaite communiquer c'est lui qui va émettre à n'importe quel client qui est « bind » sur ce service. Une autre implémentation possible aurait été l'activité principale vienne interroger le service toutes les x millisecondes.

Le service est créé au lancement de l'application il n'est cependant pas lancé. Il ne faut pas oublier de déclarer son service dans le manifeste XML, dans mon cas cela ressemble à ceci :

```
<service android:name="com.hepia.BusLocation.BroadcastService" />
```

Mon service s'occupe donc de récupérer les données iBeacons qui entourent le téléphone, c'est donc un service qui implémente un service. À chaque fois qu'un scan est achevé, je stocke les iBeacons capturés dans un objet de type « Queue ». Lorsque le nombre d'iBeacons présent dans la queue atteint la taille de la fenêtre, je peux commencer à trier tous mes iBeacons. Ceux-ci sont extraits de la queue pour être placés dans une liste. À ce moment-là, tous les Beacons sont mélangés, pour m'aider à trouver tous les Beacons identiques (avec le même UUID, major et minor) j'ai simplement créé une clé avec leurs points communs et j'ai inséré ceci dans un dictionnaire contenant une liste de Beacons. Une fois mon dictionnaire rempli avec tous mes iBeacons il me suffit de les extraire clé par clé et d'appliquer un algorithme de tri sur la valeur du RSSI, dans mon cas j'ai simplement implémenté un algorithme quick sort. Une fois les RSSI il suffit d'extraire l'iBeacon se trouvant au milieu de la liste pour récupérer l'élément médian, cet élément est ensuite ajouté à une liste se nommant « broadcastList » contenant un élément médian pour chaque iBeacon.

Cette méthode n'est sûrement pas la plus optimisée (surtout la partie avec le dictionnaire) mais cela m'a permis de facilement trier les iBeacons selon leurs RSSI.

Voici le code qui me permet de faire tout ça :

```
buffer.add(beacons); // Add latest beacons
buffer.remove(); // Remove oldests beacons

// Create a dictionary
Map<String, List<Beacon>> allBeacons = new HashMap<String, List<Beacon>>();
for (List<Beacon> listBeacs : buffer) {
    for (Beacon beac : listBeacs) {
        String key = beac.getProximityUUID() + "_" + beac.getMajor() + "_"
            + beac.getMinor();
```

```
// One key with multiple values, in this case multiple beacons
// If the already exist
if (allBeacons.containsKey(key))
    allBeacons.get(key).add(beac);
else { // Else, create the entry
    List<Beacon> tmpList = new ArrayList<Beacon>();
    tmpList.add(beac);
    allBeacons.put(key, tmpList);
}
}

broadcastList = new ArrayList<Beacon>();
for (String key : allBeacons.keySet()) {
    // Order list
    List<Beacon> orderedList = quickSortByRSSI(allBeacons.get(key));
    // Get the median
    broadcastList.add(orderedList.get((int) (orderedList.size() / 2)));
}
```

Une fois notre « broadcastList » crée avec toutes les médianes, il suffit de l'envoyer via un broadcast à l'application principale. Pour ce faire, une méthode se nommant « sendUpdatesToUI » est appelée toutes les secondes, voici son contenu :

```
if (buffer.size() >= WINDOW_SIZE) {
    intent.putExtra("counter", String.valueOf(++counter));
    intent.putParcelableArrayListExtra("beacons", broadcastList);
    sendBroadcast(intent);
}
handler.postDelayed(this, 1000); // Recall sendUpdatesToUI after 1 seconds
```

La variable « counter » permet simplement de savoir combien d'envois ont été effectués par le service.

Il faut savoir que pour envoyer un objet par un service il faut obligatoirement qu'il implémente le type « Parcelable », heureusement le type « Beacon » implémenté par Estimote implémente déjà cela.

Le gros avantage de ce service c'est essentiellement qu'il puisse continuer à tourner même si le téléphone est verrouillé.

8.3.2 Définition d'un bus

Afin de pouvoir simuler des bus, j'ai dû créer une classe permettant de définir les caractéristiques de ceux-ci. La classe se base sur ces attributs :

- String lineNumber : le numéro de ligne que dessert le bus
- String City : la ville dans laquelle le bus est actuellement
- String ID : un ID unique pour chaque bus permettant de les différencier.
- ArrayList<Beacon> beaconsFrontToBack : la liste qui contient tous les iBeacons qui équipent le bus. Ceux-ci sont définis de l'avant du bus vers l'arrière.
- Int distanceBetweenBeacons : contient la distance en mètres qui sépare les iBeacons définis plus tôt.

Tous ces attributs sont passés à la création d'un objet de type « Bus ».

Dans une version améliorée de l'application, cette classe pourrait récupérer les données directement depuis une base de données contenant toutes informations des bus afin de les stocker ensuite dans des objets de type « Bus ».

Voici un exemple de définition d'un bus :

```
ArrayList<Beacon> bus1BeaconsFrontToBack = new ArrayList<Beacon>();  
  
bus1BeaconsFrontToBack.add(new Beacon("b9407f30-f5f8-466e-aff9-25556b57fe6d",  
                                         "Beacon1", "FC:56:89:42:FB:FF", 11111, 1, -68, -1));  
bus1BeaconsFrontToBack.add(new Beacon("b9407f30-f5f8-466e-aff9-25556b57fe6d",  
                                         "Beacon2", "D7:2E:74:79:1B:6B", 11111, 2, -68, -1));  
  
Bus bus1 = new Bus("10", "Genève", "1", bus1BeaconsFrontToBack, 10);
```

8.3.3 Localisation

Mon algorithme de localisation est plutôt simple, tout d'abord, j'élimine tous les bus dont leurs iBeacons ne sont pas détectés, pour cela une méthode « isThisBusDetected » a été implémentée à la classe Bus à laquelle on lui passe une liste contenant tous les iBeacons détectés. À l'intérieur de cette méthode, je teste également si la distance de ces iBeacons est inférieure à 150% de la distance entre les iBeacons, ce qui me permet d'éliminer le Bus si celui-ci se trouve tout simplement à une trop grande distance du téléphone.

Une fois la liste de tous les bus détectés complète, il me suffit de trouver lequel est le plus proche du téléphone. Pour cela, j'additionne les distances des deux iBeacons dans chaque bus et je cherche la valeur la plus petite dans la liste de bus, une fois ce bus trouvé j'estime que c'est le bus dans lequel l'utilisateur se trouve.

Cet algorithme est loin d'être parfait, surtout si deux bus se trouvent être l'un à côté de l'autre.

9 Les problèmes rencontrés

La plupart des problèmes rencontrés ont plutôt été orientés logiciel, ceci était essentiellement dû au fait que mes connaissances en programmation sur Android étaient limitées. Malgré cela l'aide apportée par la communauté Android ainsi que le blog Estimote m'ont permis de facilement et rapidement résoudre mes mauvais pas.

La partie qui m'a pris le plus de temps a été la création du service, simplement à cause du fait qu'il existe beaucoup de façon de communiquer entre le service et l'application. Certaines de ces communications sont assez complexes et pas forcément adaptées à mon utilisation. Au final j'ai opté pour le service en mode broadcast.

Un autre problème rencontré a été le fait que mon téléphone est passé en version Android 5.0, ceci a eu pour effet de faire crasher toutes mes applications développées. Pour résoudre ce problème, j'ai simplement du changé le SDK proposé par Estimote avec leur nouvelle version, néanmoins j'ai mis plusieurs heures pour trouver d'où venait le problème et comment le résoudre.

10 Conclusion

En conclusion, il est possible de localiser quelqu'un dans un bus, mais cette localisation a ses limites. Il ne faut pas s'attendre à avoir une précision de l'ordre du centimètre, mais plutôt de 2 à 3 mètres près selon l'environnement. Le service créé est tout à fait opérationnel, néanmoins l'algorithme de détection du bus peut être facilement amélioré.

D'un point de vue plus personnel, j'ai beaucoup apprécié ce projet, il m'a permis d'apprendre de nouvelles choses sur la technologie Bluetooth ainsi que perfectionner mes compétences en programmation Android.

11 Sources

Estimote knowledge base

<https://community.estimote.com/hc/en-us>

Android GPS, Location Manager Tutorial

<http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/>

Bluetooth 4.0: An introduction to Bluetooth Low Energy—Part I

http://www.eetimes.com/document.asp?doc_id=1278927

Bluetooth 4.0: Low Energy Joe Decuir, Standards Architect, CSR plc; IEEE Region 6 NW Area chair

<http://chapters.comsoc.org/vancouver/BTLER3.pdf>

How do iBeacons work?

<http://www.warski.org/blog/2014/01/how-ibeacons-work/>

What makes an iBeacon an iBeacon

<http://glimwormbeacons.com/learn/what-makes-an-ibeacon-an-ibeacon/>

What's the difference between Beacons and iBeacon™?

<http://blog.fosbury.co/whats-the-difference-between-beacons-and-ibeacon%E2%84%A2>

Radius Networks' ibeacon ranging fluctuation

<https://stackoverflow.com/questions/21338031/radius-networks-ibeacon-ranging-fluctuation>

What is the iBeacon Bluetooth Profile

<https://stackoverflow.com/questions/18906988/what-is-the-ibeacon-bluetooth-profile>

The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs

<http://www.aislelabs.com/reports/beacon-guide/>

Energy Analysis of Neighbor Discovery in Bluetooth Low Energy Networks

<http://research.nokia.com/sites/default/files/public/TR-EnergyAnalysis-BLE.pdf>

12 Annexes

Toutes les annexes peuvent être téléchargées sur GitHub à l'adresse suivante :

<https://github.com/Healjin/iBeaconsGeolocation>

Voici le contenu de ce répertoire :

- Les trois applications Android citées dans ce document
- Le rapport en format PDF
- Les captures des deux trajets réalisées avec l'application n°2
- Un script Matlab simulant le traitement du service de l'application n°3