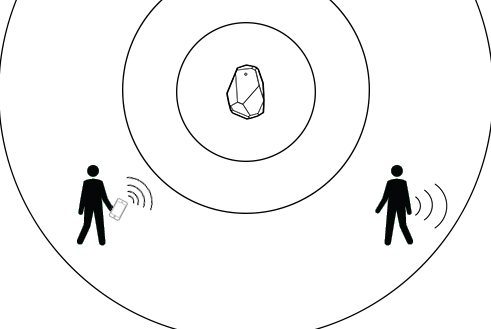
Rapport de développement mobile

**Track Me Inside**



## Table des matières

[Table des matières 2](#_Toc473059924)

[Introduction 3](#_Toc473059925)

[But 3](#_Toc473059926)

[Contraintes 3](#_Toc473059927)

[Déroulement 3](#_Toc473059928)

[Estimote SDK 4](#_Toc473059929)

[Choix entre Ranging et Monitoring 4](#_Toc473059930)

[Différences entre les SDK Estimote pour Android et Apple 5](#_Toc473059931)

[Technologie d’émission 6](#_Toc473059932)

[Bluetooth LE (BLE) 6](#_Toc473059933)

[IBeacon 6](#_Toc473059934)

[Inférer la position de l’utilisateur 7](#_Toc473059935)

[Dériver la distance du signal du balise 7](#_Toc473059936)

[Complications 8](#_Toc473059937)

[Trilatération vs triangulation 9](#_Toc473059938)

[Implémentation de la trilatération 9](#_Toc473059939)

[Utilisation d’un deuxième capteur 10](#_Toc473059940)

[Protocole de calibrage 10](#_Toc473059941)

[Persistance du calibrage 11](#_Toc473059942)

[Tests de la précision de notre application 11](#_Toc473059943)

[Détermination de la distance entre une balise et le téléphone 12](#_Toc473059944)

[Détermination de la position de l’utilisateur 12](#_Toc473059945)

[Améliorations 12](#_Toc473059946)

[Economie de la batterie des beacons 12](#_Toc473059947)

[Conclusion 13](#_Toc473059948)

[Références 14](#_Toc473059949)

## Introduction

Dans le cadre du projet de semestre au sein du cours de développement mobile de Madame A. Rizzoti, nous avons choisi de faire un projet sur Android en utilisant des balises de localisation Estimote. Ce projet a pour but d’apprendre à développer avec le langage Android ainsi que d’apprendre le fonctionnement des balises Estimote.

## But

L’application Android « Track Me Inside » doit permettre le chargement et l’affichage de données sur le téléphone de l’utilisateur lorsque celui-ci se trouve à un endroit spécifique à l’intérieur d’une salle de classe.

### Contraintes

* L’application doit être développée sur Android ;
* L’application doit permettre de localiser le téléphone de l’utilisateur à l’aide de plusieurs balises ;
* La localisation doit être précise dans toute la salle de classe ce qui exclut l’utilisation naturelle du SDK Android de Estimote qui se base sur la proximité de l’utilisateur à une zone définie par une ou plusieurs balises ;
* Ces balises seront représentées des Proximity Beacons de l’entreprise Estimote ;
* L’application doit utiliser au moins deux capteurs différents ;
* L’application doit permettre la persistance de certaines données.

### Déroulement

Premièrement, nous allons présenter les classes du Estimote Android SDK que nous avons implémentées ainsi que celles que nous avons considérées.  
Deuxièmement, nous détaillerons le fonctionnement du protocole Bluetooth.  
Troisièmement, nous expliquerons comment nous avons calculé la position de l’utilisateur à partir de plusieurs balises.  
Finalement, nous parlons des améliorations possibles et ferons une conclusion.

## Estimote SDK

Ce SDK s’adresse à toute la gamme de produit de Estimote. Cela comprend les Proximity Beacons, Location Beacons, Long Range Location Beacons, Stickers, Mirror. Nous nous intéressons uniquement aux Proximity Beacon car ils sont les plus performants.

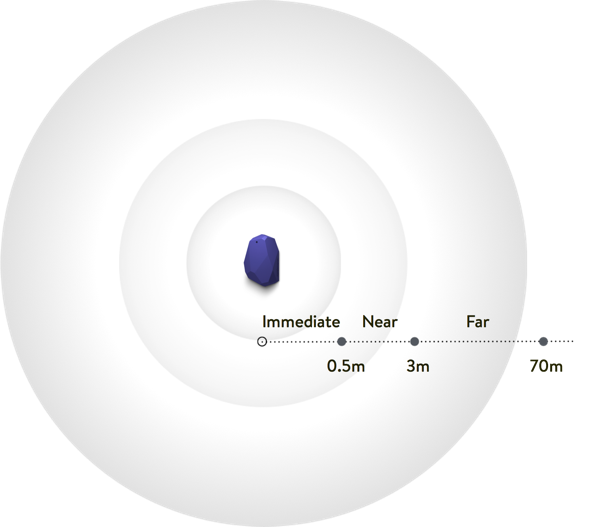
* La classe *Beacon* représente une balise Estimote.
  + Les méthodes permettant de déterminer l’identité d’une balise par le biais de son *UUID*, *minor* et *major* sont respectivement *getProximityUUID ()*, *getMinor ()*, *getMajor ()*.
  + La méthode *getRSSI ()* renvoie un Indicateur de puissance du signal reçu. C'est la puissance du signal de la balise au moment où le téléphone reçoit ce signal.
  + La méthode *getMesuredPower ()* renvoie une constante en lecture seule étalonnée en usine qui indique quel est le RSSI attendu à une distance de 1 mètre de la balise. Combiné avec RSSI, il permet d'estimer la distance entre le dispositif et la balise. Nous appellerons cette valeur « **txPower** ».
* La classe *Utils* propose quelques méthodes pratiques :
  + La méthode *computeAccuracy (Beacon beacon)* retourne la distance entre le beacon et le téléphone en mètre à partir du RSSI et de la puissance d’émission mesurée de la balise (txPower).
    - Il n’est pas possible de donner le txPower en argument de la méthode ni de modifier cette valeur dans l’objet *beacon*. Cependant nous pouvons passer par le constructeur de la classe *beacon* pour spécifier le txPower.
* La classe *Utils.Proximity* propose quatre zones pour estimer la distance à une balise. Ces quatre zones sont représentées par un Enum qui est composé de :
  + IMMEDIATE, NEAR, FAR, UNKNOWN
* La classe *BeaconManager* permet de monitorer une zone.
  + Il existe deux classes dérivées qui implémente des listeners donnant une indication sur la proximité de l’utilisateur à un beacon ou plusieurs : *MonitoringListener* et *RangingListener*

### Choix entre Ranging et Monitoring

Le SDK détecte et interagit avec les balises de deux façons :

* Monitoring : actions déclenchées à l'entrée et à la sortie de la région. Fonctionne si l'application est en cours d'exécution, suspendue ou tuée (si l'application ne s'exécute pas quand une entrée / sortie arrive, l’OS le lancera en arrière-plan pendant quelques secondes pour gérer l'événement)
* Ranging : actions déclenchées en fonction de la proximité d'une balise ; Ne fonctionne que lorsque l'application est en cours d'exécution (par exemple, elle s'affiche à l'écran ou s'exécute en arrière-plan en réponse à un événement de surveillance, etc.)

Ranging retourne une **liste de balises dans la portée**, avec une proximité estimée de chacun d'eux.



Nous avons choisi la classe *RangingListener* car elle fonctionne même quand l’utilisateur est dans la zone. L’information renvoyée par la classe *RangingListener,* c’est-à-dire une liste de *Utils.Proximity* n’est d’aucune utilité pour notre application. Nous avons choisi d’inférer la distance entre le téléphone et les beacons à partir du RSSI.

### Différences entre les SDK Estimote pour Android et Apple

Le SDK indoor location développer pour iOS emploie une tonne de techniques complexes pour prédire où l'utilisateur est. Estimote est en train d’adapter ses algorithmes pour Android mais c’est un processus qui prend du temps en raison de l'immense variation des appareils Android. Chaque modèle possède une antenne Bluetooth et un hardware différent (Différents matériels Bluetooth, différentes positions des antennes, différents facteurs de forme et matériaux de fabrication, etc.).

## Technologie d’émission

### Bluetooth LE (BLE)

Le protocole Bluetooth Smart aussi appelé Bluetooth LE ou Bluetooth 4.0 offre des améliorations par rapport à sa technologie Bluetooth prédécesseur. Le protocole fonctionne autour de la fréquence radio 2,4 GHz et est utilisé pour des courtes portées.

Bluetooth Low-Energy vise à fournir des communications RF (Radio Fréquence) à faible consommation d'énergie à court terme, c'est-à-dire en dessous de 100 mètres de couverture du signal. Le protocole Bluetooth LE met l'accent sur les économies d'énergie plutôt que sur le débit de données ou sur la modulation.

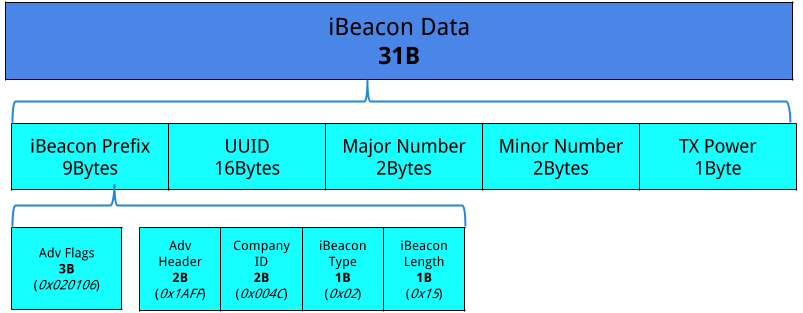
Selon le cas, le Bluetooth LE peut utiliser jusqu’à 1% de batterie par rapport à la norme Bluetooth. Par contre le débit de données pour descendre au tiers de la technologie Bluetooth.

### IBeacon

IBeacon est un système de positionnement en intérieur développé par Apple fonctionnant sur Bluetooth 4.0. Ce protocole est utilisable par IOS, OS X 10.9+ et Android 4.3+. Il diffuse quatre informations :

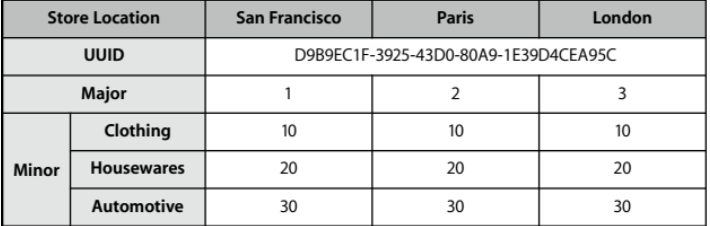
* Un *UUID*: identifie la balise par une chaine alphanumérique définit par l’utilisateur  
  de la forme « \*\*\*\*\*\*\*\*-\*\*\*\*-\*\*\*\*-\*\*\*\*-\*\*\*\*\*\*\*\*\*\*\*\* ».
* Un *Major*: valeur identifiant un sous-ensemble de balises dans un grand groupe.
* Un *mineur*: valeur identifiant une balise spécifique.
* Un niveau de puissance TX indiquant la force du signal à un mètre de l'appareil. Cette donnée est étalonnée par Apple pour être précis pour ses propres produits.

L’application lit l'UUID, le majeur et le mineur et les utilise pour déterminer des informations sur la balise par le biais d’une base de données ; La balise elle-même ne contient aucune information descriptive.



#### Exemple de l’utilisation du protocole iBeacon

Voici l’exemple d’un magasin de détail national. L'UUID représente le propriétaire des balises, ici l’entreprise. Le Major définit chaque emplacement du magasin. En utilisant cette valeur, l'application sera en mesure de déterminer facilement dans quel magasin l'utilisateur est actuellement. Le Minor définit une section spécifique dans chaque magasin. Cela permet à l'application de déterminer la section du magasin dans laquelle l'utilisateur se trouve.



L’image ci-dessus montre un exemple de mise en œuvre du protocole iBeacon

## Inférer la position de l’utilisateur

Les estimations de distance aux balises ne seront jamais super fiables. Ils sont basés sur la force du signal qui arrive au téléphone, mais le problème est que ce signal peut parcourir des chemins multiples (ligne droite, ou rebondir sur des murs, etc.), ou être absorbé sur le chemin (par exemple, par d'autres personnes). Même la façon dont le téléphone est tenu par l’utilisateur a un impact sur la précision (passer du mode portrait en le tenant d'une main au mode paysage à deux mains). Le téléphone ne peut pas savoir si la puissance du signal a baissé parce que quelqu’un fait obstacle ou parce que la distance a augmenté.

La distance et la puissance perçue suivent une relation quadratique inverse. Si la distance à la balise double, la force du signal est divisée par quatre. Cela diminue drastiquement la précision de l’estimation de la distance à mesure que la distance augmente.

Cette puissance du signal reçu est appelée **Received Signal Strength Indication** ou **RSSI.**

### Dériver la distance du signal du balise

Il existe une grande variété de formules qui lient le RSSI à la distance. La plus populaire est :  
RSSI (dBm) = - 10n \* log10(d) + txPower.  
  
La constante de propagation, aussi appelée l'exposant de perte de trajet, est donnée par « n ». Elle varie de 2,7 à 4,3 selon l’humidité du milieu. (Le vide a n = 2 pour référence).

Cette formule ne nous a pas donné de résultat satisfaisant.

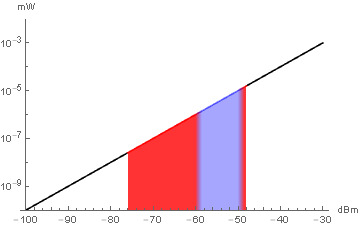
Ensuite, nous nous sommes basé sur une formule qui effectue une régression de puissance sur la base de mesures effectuées sur un dispositif spécifique. Cette formule est :  
distance [m] = A \* (RSSI / txPower)B + C avec A, B et C étant des constantes.

La calibration de ces constantes a été faite pour le Nexus 4 dont les valeurs sont :  
A = 0.42093, B = 6.9476 et C = 0.54992.

### Complications

Nous avons remarqué qu’il y avait des différences significatives entre le RSSI des deux modèles de beacons utilisés. En effet, la puissance d’émission des trois « Proximity beacon » ainsi que des trois « Location beacons » est réglée sur +4dBm cependant la puissance lue à un mètre est différente d’un modèle à l’autre. La médiane n’est pas la même, donc le calibrage correspondant aux deux modèles sera différent. En plus, lecture des RSSI des « Proximity beacons » fournissent des valeurs avec un faible écart-type alors que ceux des « Location beacons » possèdent un écart-type très grands.

Le graphique suivant illustre ces différences à un mètre d’un beacon.



= Proximity Beacon

= Location Beacon

Afin de palier à ce problème nous avons utilisé uniquement les « Proximity beacons » lors des tests de notre application. Il serait possible de modifier la puissance d’émission des « Location beacons » rendre la puissance du signal à un mètre soit similaire à l’autre modèle mais cela entrainerait des erreurs qui rendrait nos tests de précisions encore plus difficile à interpréter.

Une autre alternative serait de modifier le txPower de chaque modèle en fonction d’un calibrage unique. Notre expérience montre que la mise à jour de propriété d’une balise au travers du protocole Bluetooth est lente est peu fiable. Cette solution est par conséquent inadaptée pour une application qui doit être réactive.

Les deux alternatives ci-dessus n’ont pas été explorées car la variance des « Location beacons » est un problème insoluble à notre connaissance.

Nous avons également observé que le signal des « Proximity beacons » étaient moins absorbé par un obstacle. Quand une personne se place entre un beacon et le téléphone, le RSSI fluctue peu. Dans un cas identique, le signal d’un « Location beacon » est complétement absorbé.

### Trilatération vs triangulation

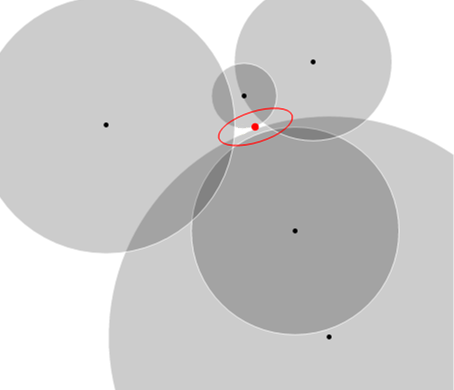
Afin de déterminer la position de l’utilisateur, l’application scan les balises à proximité. Quand trois balises sont captées et leur puissance d’émission est suffisamment haute, il est temps de mettre ces données ensemble. A cette fin, il existe deux méthodes principales : la triangulation et la trilatération.

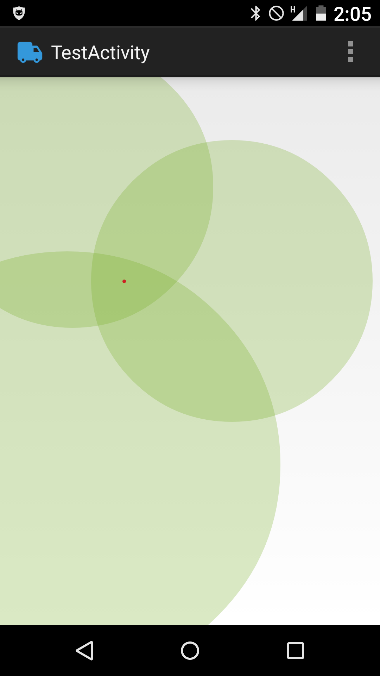
La triangulation est le processus de détermination de l'emplacement d'un point en mesurant les angles entre celui-ci et des points connus.

Afin de pouvoir mesurer les angles d’arrivée du signal de réception (angle of arrival 'AOA') il nous faut des antennes spécifiques. Malheureusement, les smartphones actuels ne possèdent pas ces antennes pour la technologie Bluetooth ou Wifi.

Nous avons donc opté pour la trilatération. Elle consiste à mesurer la distance entre téléphone et trois beacons puis de construire trois cercles dont le centre est défini par la positions des beacons et le rayon par la distance mesurée. La position du téléphone sera donnée par l’intersection de ces cercles.

### Implémentation de la trilatération





Méthode 2

Méthode 1

Nous avons testé deux méthodes :

La première méthode utilise une interpolation linéaire en deux dimensions. Trois cercles sont générés à partir de la position de chaque balise et de sa distance avec l’utilisateur. Le résultat est une donnée par le centre de la zone à équidistance des trois cercles. Le résultat est précis dans le cas où le téléphone se trouve dans la surface définit par les trois disques calculés. Cette méthode n’est donc pas adaptée à notre problème.

L’avantage de cette méthode se trouve dans le fait qu’elle ne nécessite pas que les trois cercles se touchent ou se coupent.

La deuxième méthode calcule la zone d’équidistance entre les balises. Cette méthode ne donne pas de solution précise mais elle est beaucoup plus réaliste. De plus, elle donne peu d'importance si la distance du beacon au téléphone est grande en attribuant à chaque balise un poids inversement proportionnée au carré de sa distance avec le téléphone. En effet, plus la distance est élevée, plus elle est sujette à des imprécisions car la puissance du signal diminue de façon quadratique lorsque la distance diminue. Ainsi un petit changement de puissance perçue, chose courante, aura beaucoup plus d'impact pour une longue distance plutôt que pour une courte distance.

Nous avons effectué des tests unitaires sur la fonction de trilatération et les résultats montre que la deuxième méthode est plus consistante, plus cohérente mais moins précise lorsque l’intersection est parfaite.

## Utilisation d’un deuxième capteur

Nous utilisons un 2e capteur pour réduire le bruit lors de l’estimation de la position et pour des économies de batterie.

Lorsque l’utilisateur se déplace, la puissance perçue est beaucoup précise donc nous avons utilisé l’accéléromètre pour prévenir l’utilisateur de la perte de précision du calcul de sa position.

Afin de détecter les mouvements de l’utilisateur, nous calculons la norme du vecteur résultant de l’addition des vecteurs d’accélération sur les trois axes. Si cette valeur dépasse un seuil donné, un indicateur visuel averti l’utilisateur que l’approximation de la position est biaisée en affichant un indicateur visuel rouge et le « ranging » est arrêté. Quand celui-ci est à nouveau stationnaire pendant deux secondes, l’indicateur s’affiche en vert et le « ranging » est redémarré.

Ce processus d’arrêt puis de redémarrage du « ranging » permet également des économies de batterie. En effet le processus de « ranging » à un intervalle très court est coûteux en énergie.

## Protocole de calibrage

Le téléphone est posé sur une surface à niveau et la balise est placée à un mètre de celui-ci dans la direction du haut du téléphone. Aucun obstacle ne doit être placé entre les deux appareils. Pendant une période d’environ 5 secondes, le téléphone lit le RSSI de la balise. À la fin du temps imparti, nous effectuons une moyenne tronquée sur ces valeurs. C’est-à-dire que le 10% des plus grandes valeurs et le 10% des plus petites valeurs sont ignorées avant de prendre la moyenne des valeurs restantes.

L’image ci-dessous illustre la façon dont le calibrage est effectué.

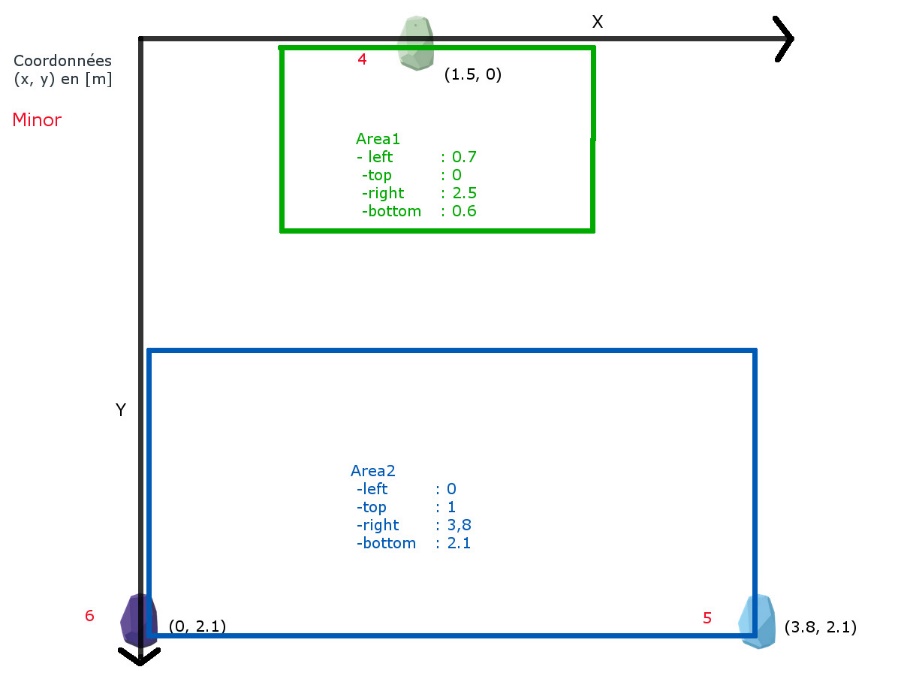


### Persistance du calibrage

La constante de calibrage est sauvegardée dans le cache de l’application. La première fois que l’utilisateur ouvre l’application, la fenêtre de calibrage sera automatiquement lancée. Une fois cette opération effectuée, cette fenêtre sera accessible par le biais d’un onglet.

## Tests de la précision de notre application

Les tests suivants ont été effectués dans la salle de classe 320 du campus Arc2 en utilisant la configuration ci-dessous :



Les balises se trouvent à la même hauteur et émettent à +4dBm avec un intervalle de 100ms. Nous avons utilisé un Nexus 4 avec un intervalle de scan de 200ms.

### Détermination de la distance entre une balise et le téléphone

En dessous de 3 mètres de distance, la distance peut varier de +/- 1 mètre. À des distances plus élevées, la variation augmente. En dessus de 3 mètres la précision fournie par notre application peut varier de +/- 2 mètres.

### Détermination de la position de l’utilisateur

Nous avons obtenu une précision de +/- un mètre sur les deux axes. Cela signifie que si le téléphone se trouve au centre d’une zone de contenu carrée de 3 mètres de côté, le chargement de données se fera correctement à chaque fois.

## Améliorations

Voici quelques points que nous pourrions investiguer afin d’augmenter la précision du processus du calcul de la position de l’utilisateur :

* Utiliser un modèle de trilatération basée sur R3. Cela ralentirait les calculs, mais augmenteraient la précision globale.
* La distance peut être inférée à partir du moment où le signal est arrivé au téléphone (time of arrival « TOA »). Cependant, cette technique a besoin d'une synchronisation d'horloge précise entre le téléphone et les beacons. Nous n’avons pas pu utiliser cette technique, car les beacons ne peuvent pas transmettre le moment d’émission d’un signal.
* Le RSSI dépend de l’orientation de l’antenne de réception. Nous pourrions développer un algorithme pour moduler le RSSI en fonction de l’orientation du téléphone en utilisant les mesure du gyroscope intégré. Cette technique est utilisée dans le Estimote Indoor Location SDK pour iOS.
* L’utilisation d’une antenne externe permettrait de contourner les problèmes liés à la variation de hardware ou d’antenne des téléphones sous Android. Si cette antenne permettait de mesurer l’angle d’arrivée du signal, il serait possible de calculer la position à l’aide de la triangulation.
* Implémenter un algorithme de détection et de suppression des signaux rebonds.

### Economie de la batterie des beacons

Lors du déploiement d'un réseau de capteurs Bluetooth il est important de garder à l'esprit que chaque nœud a une quantité limitée d'énergie. La batterie des beacons peut tenir jusqu’à deux ans en utilisant les paramètres par défaut et que la gestion intelligente de la batterie est activée.

Afin d’augmenter la précision de l’application nous avons modifié certains paramètres de base de nos balises :

* La puissance d’émission : La valeur par défaut est de -12 dBm et peut varier de -30 dBm à +4 dBm. Nous avons choisi +4dBm.
* L’intervalle d’émission : La valeur par défaut est de 950ms et peut varier de 100ms à 10s. Nous avons choisi 100ms.

Nous avons néanmoins utilisé deux fonctionnalités pour économiser la batterie des balises Estimote :

* « Smart Power Mode » : Le mode intelligent utilise des algorithmes adaptatifs avancés pour optimiser automatiquement la puissance d’émission. S'il n'y a personne à portée, la balise émet moins souvent pour économiser la batterie. Lorsqu'un utilisateur est détecté, la balise émet avec un débit normal.
* « Flip to Sleep » : La balise cesse d’émettre lorsque qu’elle est sur le dos.

## Conclusion

La précision de la localisation du téléphone, objectif principal de ce projet, est correcte pour des distances inférieures à 3 mètres d’un Beacon. Il est à noter que ces résultats sont néanmoins obtenus dans des conditions quasi parfaites.

Il reste certes des pistes d’amélioration que nous pouvons suivre, en particulier une meilleure estimation de la distance qui sépare le téléphone des Beacons. Ceci est cependant compliqué, car l’estimation actuelle s’effectue sur la base de la puissance perçue et cette dernière est très variable.

Estimote a prouvé que cet objectif était atteignable en utilisant le protocole IBeacon avec des appareils Apple. Les algorithmes qui composent le Estimote Indoor SDK pour IOS sont le fruit de la mise en commun de plusieurs thèses par une équipe d’experts. Il est donc difficile de juger de l’écart qui sépare nos résultats de ceux du Indoor SDK.

## Références

* Estimote Indoor SDK on Android:
  + <https://community.estimote.com/hc/en-us/articles/203909356-Is-Indoor-Location-SDK-available-on-Android->
  + <https://forums.estimote.com/t/android-and-ios-indoor-location/4633>
* iBeacon : <https://developer.mbed.org/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/>
* Calcule de la distance :<https://forums.estimote.com/t/determine-accurate-distance-of-signal/2858/4>
  + <https://altbeacon.github.io/android-beacon-library/distance-calculations.html>
  + <http://www.eecs.ucf.edu/seniordesign/fa2014sp2015/g14/docs/sd_1_report.pdf>
  + <http://electronics.stackexchange.com/questions/83354/calculate-distance-from-rssi>
  + <http://www.slideshare.net/ChrisThomson2/factors-effecting-positional-accuracy-of-ibeacons>
* Informations sur les beacons : <https://community.estimote.com/hc/en-us/articles/201636913-What-are-Broadcasting-Power-RSSI-and-other-characteristics-of-beacon-s-signal->
* Estimote Android SDK : <http://estimote.github.io/Android-SDK/JavaDocs/>
* SensorEvent : <https://developer.android.com/reference/android/hardware/SensorEvent.html#values>
* Accéléromètre : <https://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--mobile-22125>
* Image : <http://stackoverflow.com/questions/29047902/how-to-add-an-image-to-the-drawable-folder-in-android-studio>
* Options de stockage : <https://developer.android.com/guide/topics/data/data-storage.html>
* Estimote « ranging » : <https://developer.android.com/guide/topics/data/data-storage.html>
* Tests unitaires locaux : <https://developer.android.com/training/testing/unit-testing/local-unit-tests.html>
* Moyenne tronquée (sans les extrêmes) : <https://fr.wikipedia.org/wiki/Moyenne_tronqu%C3%A9e>
* Triangulation : <https://www.researchgate.net/post/How_to_detect_the_object_as_well_as_its_distance_using_tringulation>
* Théorie sur la trilatération :
  + <https://en.wikipedia.org/wiki/Trilateration>
  + <https://arxiv.org/ftp/arxiv/papers/1312/1312.6829.pdf>
* Trilatération (méthode 1) : <http://stackoverflow.com/questions/30336278/multi-point-trilateration-algorithm-in-java>
* Trilatération (méthode 2) : <https://github.com/Navigine/Indoor-navigation-algorithms/blob/master/navigation/trilateteration/src/trilateration.cpp>
* Résolution du problème de fréquence du ranging : <https://github.com/Estimote/Android-SDK/issues/183>
* Résolution du problème de redémarrage de l’activité lors d’une rotation : <http://stackoverflow.com/questions/456211/activity-restart-on-rotation-android>