**Bluetooth 4.0 Indoor Positioning System**

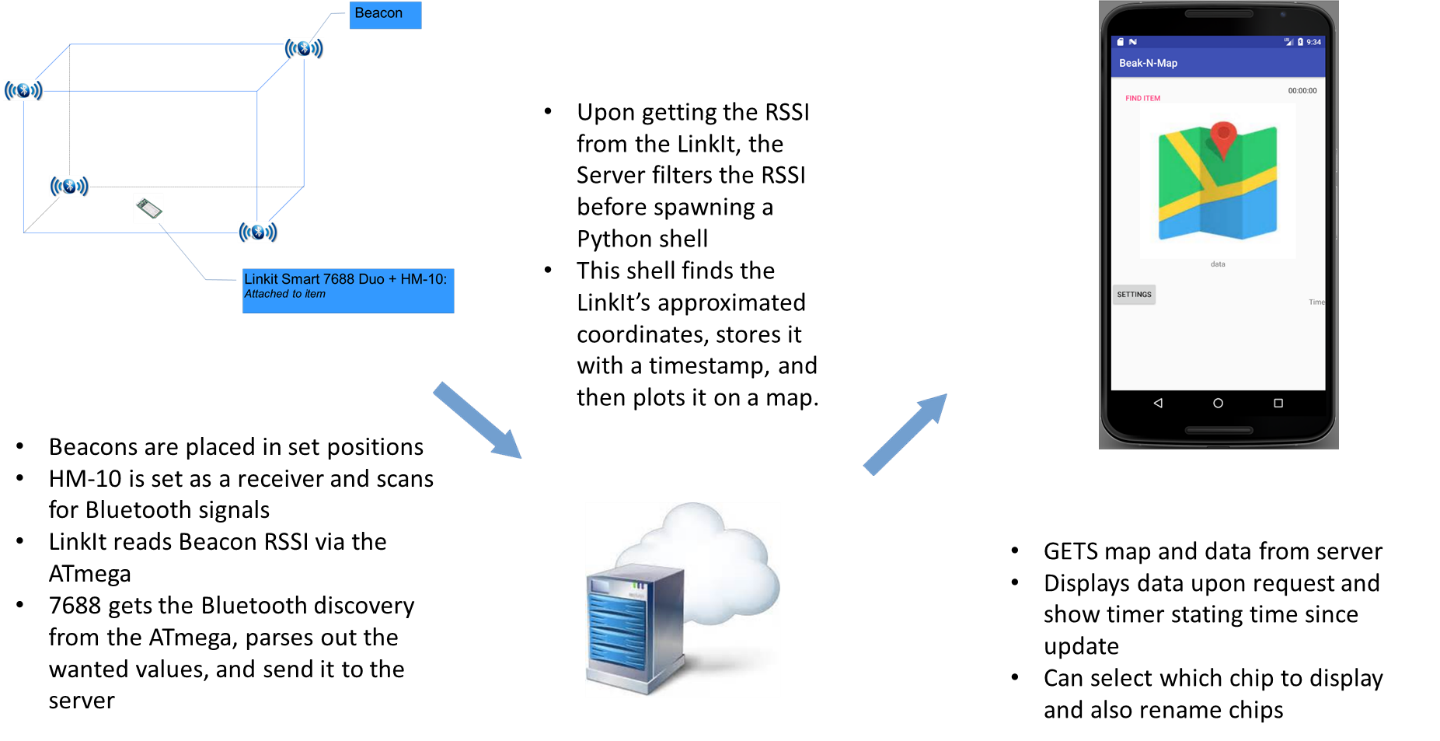
Version: 0.4.2

**Goals:**

Develop an Indoor Positioning System that will allow someone to track the position of an object from their phone.

**Changelog:**

|  |  |
| --- | --- |
| 0.1.0 | 3 LinkIts are to be placed around a room. Using HM-10s, the LinkIts will look for BLE signals. A beacon will be attached to the object that should be tracked. The LinkIts will read the relative RSSI of the beacon and send it to a server. The server will calculate the object’s location using triangulation and return a xy-coordinate in the Cartesian plane with one corner of the room designated as the origin. This data will then be stored until the next coordinate is returned. An Android app will be able to display said data. |
| 0.2.0 | The LinkIt is attached to object; 4 beacons will be placed in corners of the room such that no more than 3 beacons are co-planar. This will enable 3D positioning. Trilateration is to be used instead of triangulation. Calculations are now done in the LinkIt. Note: Placing the Beacons in the corners streamlines the data flow. |
| 0.2.1 | Server will now also generate a map with the object location plotted and attach a time stamp of when the data was obtained. App will be able to pull the data. |
| 0.3.0 | RME Minimization is now to be used instead of trilateration. Calculations are moved to server. LinkIt is now only responsible for gathering RSSIs |
| 0.3.1 | RSSI values are processed according to the Android Beacon Library Standard before RME Minimization. App provides the option to “rename” LinkIts as well as choose whether to display specific objects |
| 0.3.2 | RSSI-to-meter conversion now done according to the Android Beacon Library Standard. Optimized code to be room-specific by limiting ranges of possible values. |
| 0.4.0 | Beacons are now HM-10s calibrated as broadcasters |
| 0.4.1 | Uses a Kalman Filter to Stabilize RSSI Values |
| 0.4.2 | Android App can emulate LinkIt’s Behavior (Finds own Location) |
| 0.4.3 | (In development) Server can store data from mutiple LinkIts at the same time. See README. |

**Overview:**

**Figure1: Data Flow of the IPS.** Further information can be found below and in the README files.

*Beacons:* Bluetooth Low Energy Hardware Transmitters; they broadcast their identifiers along with their Received Signal Strength Index (RSSI). The 4 beacons are placed in 4 corners of the room such that they are not all coplanar as pictured above. The MAC address of each beacon and their location (in a Cartesian coordinate plane) must be noted down as this will be used to identify each individual beacon. Each beacon will also have to be calibrated individually as difference exist between each one, even if only slightly.

*LinkIt:* Used for prototyping purposes. An HM-10 is attached to it to give the chip Bluetooth capabilities.

*ATmega32U4:* MCU; Compatible with Arduino Yun Sketches. Sends discovery commands (“AT+DISI?”) to the HM-10, which will respond with a string containing all the BLE devices it has discovered in the format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OK+DISC | FactoryID(8) | UUID(32) | Major(4) Minor(4) Measured Power(2) | MAC(12) | RSSI(4) |

**Figure 2: BLE Signal Format.** This is one signal out of the many discovered by the DISI commands. The entire discovery string begins with OK+DISIS and ends with OK+DISCE. The number in each box indicates how many bytes each subsection is in size. Each box indicates a segment of the signal separated by a semicolon.

After collecting the discovery string, the ATmega32U4 sends it to the MT7688AN. This process is then repeated as long as the LinkIt has power.

*MT7688AN:* MPU; Runs OpenWrt, a stripped-down Linux distribution. Holds a python program that runs upon boot. Upon receiving the discovery string, it identifies the RSSI it requires based on the MAC address and puts it into an ordered list. Once it has at least one sample of each beacon’s RSSI, the list is formatted into a dictionary and from there, converted to JSON and POSTed onto the Server.

*Server:* Simple Node.js Server with GET and POST functions, it stores a blank map, the object’s coordinates (labeled with the time the data was found), and a map with the corresponding coordinate indicated.

*Requests:* GET returns the data that the server has stored. POST receives 4 RSSI, filters it, and then spawns a Python-Shell to process it.

*Filter (Kalman):* The RSSIs are ran through this filter to remove noise from the RSSI. One weakness of this filter is its susceptibility to sudden changes. This is compensated for by only using the past 20 values.

*Python-Shell:* Runs a python program, taking in the RSSIs as an argument and returning anything the program prints as a return argument. The program converts the RSSI to meters by using beacon specific equations and then limits the meters to values possible within the room. Using these distances, the code uses minimization of the root mean error (RME) to locate the most probable location of the objects. That is to say, based on an initial guess (set to default to the room’s center except for when one of the beacons indicate an extremely small distance), the distances from the guessed point to the beacon are compared to the distance obtained from the RSSI. This data is then used to calculated a RME. Based on the RME a new guess is set whereupon the process repeat until the RME is minimized. Once the coordinate is obtained, a timestamp (in the form of seconds since epoch) is added and the whole data set is stored on the server. Using the coordinate, the corresponding position on a map is also marked.

*App:* Upon pressing a button, the most recent map and coordinate from the server is displayed. A chronograph is also started, indicating the time since the information was refreshed. The app also supports a setting screen which allows you to choose which LinkIts are to be displayed and also give the user the ability to rename the LinkIt form the default “chip#”. Alternatively, the phone can collect the Bluetooth signals in the place of the LinkIt and post them onto the server before getting the calculated data from the server.

**Materials:**

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Quantity | Cost (RMB/unit) | Notes |
| [HM-10 Bluetooth Module](https://item.taobao.com/item.htm?spm=a230r.1.14.7.ebb2eb2kUkE1q&id=522935314627&ns=1&abbucket=18" \l "detail) | 5 | 16.8 | * 1 as receiver, 4 as beacons(broadcaster) |
| Portable Power Source | 5 | Provided | * Using improvised power sources * HM-10s have high power consumption, use rechargeable or wired power sources |
|  |
| [Beacons](http://www.seekcy.com/M1.html) | 4 | 44 | * Optional substitute for HM-10s |
| Linklt Smart 7688 Duo | 1 | Provided |  |
| USB-micro USB Power Cable | 1 | Provided |  |
| Dupont Line | 12 | Provided | * Female-Female |
| Coordinate Map of Leedian Office | 1 | Provided | * 2D Floor plan |
| * Must be to scale |
|  |
| TTL to USB Converter | 1 | Provided | * May require more to power HM-10 with USB Power Packs |
| Total Price |  | 84 | * 176 RMB for Beacons Option |

**Table 1: Final Materials used in IPS Prototyping.** Hardware from previous versions not listed.

**Hardware Set Up:**

*HM-10:*

HM-10s can be easily set up by using the Serial Monitor of the Arduino IDE. Follow the instructions listed in *HM-10 as iBeacon* [1]. To quickly determine if you are connected to the HM-10, send the “AT” command; the HM-10 should respond with “OK”.

*Beacons:* These can be set up using the instructions in *HM-10 as iBeacon*. Note that you should not enable auto-sleep if you wish to use the Beacon repeatedly.

*Receivers:* Set the receiver with the following commands. Reference the HM-10 user guide for more options. [2]

1. AT
2. AT+MODE1
3. AT+ROLE1
4. AT+IMME1

*Beacons (Seekcy):*

An alternate to the HM-10s; they have more accuracy than the HM-10s, but are harder to configure. All configurations are done in Seekcy’s proprietary app, called “Seekcy Beacon.”

*LinkIt:*

The LinkIt can be set up using the Quick Start Guide[3]. Note that you will require a computer with WiFi capabilities and a way to power the LinkIt. There are times when it is difficult to ssh into the LinkIt; at this time, check the strength of the Wifi the LinkIt is connected to – it may be that it is too weak. Additionally, sometime corrupted files will appear in ROOT. The only known solution to this is to factory reset the board by pressing the WiFi button for 20 seconds, so be sure to keep backups of everything put onto the LinkIt.

In order to make a program run on boot, follow the follow commands after logging into the LinkIt Linux OS:

$cd /

$mkdir robot

>#Put code you wish to run in directory (i.e. test.py)

$cd /

$cd etc

$vim rc.local

>#Insert the following lines:

#!/bin.sh -e

python /robot/test.py

exit 0

*Calibration:* A good overview can be found in the Android Beacon Library documentation [4][5]

*Data Gathering:* Programs written for gathering sample data can be found in the Testing directory, details are noted in the README

*Calculating Constants:* Perform a power regression with the ratio between the RSSI and the dBm at 1 meter as the independent and the distance (in meters) the RSSI was taken at as the dependent. Following this, find the difference between the calculated distance and the actual distance; this will be your intercept. [5][6]

**Conclusions:**

***Table 2: System Accuracy Breakdown.*** *Only the most recent test values are listed*

The majority of the error in this system comes from the natural fluctuation of Bluetooth signals as seen below in Table 2. Beacons producers, including Radius Networks and the Android Beacon library strongly insist beacons and BLE only be used for proximity measurements and not precise coordinate finding as in this project. This is due to the fact the BLE signals are easily affected by the environment. Line-of-sight is not enough to guarantee stable RSSI measurements, with surrounding objects greatly influencing the acuity of the signal. There are other methods of location finding with Bluetooth, but they require additional hardware. Another more stable wave type for indoor positioning would be Ultra-Wide Band(UWB), which has been reported to give distance estimate with an accuracy of up to 10 cm. In addition, there are other methods of filtering besides Kalman, which can be attempted. In this project, only a limited version of the extended Kalman filter was used. Other considerations to make are the errors that come from converting from RSSI to meters, as this can be avoided by directly using the RSSI or other properties of the signal to determine distance or location. Finally, the functions used in this locator are optimized for 4 beacons. By making use of other hardware or more beacons, the accuracy of the location pinpointing should increase.

|  |  |
| --- | --- |
| Component | Accuracy & Precision |
| RSSI Scanning Precision (no obstacles) | * Low Precision   + Seekcy: Average Std Dev 2.391937     - This can produce ~2m of error in lower RSSI (~-80) and around 1m at lower (~-40)   + HM-10: Average Std Dev 3.776246794 * Accuracy: Dependent on Environmental factors |
| RSSI Fluctuation Compensator | * Testing Done with HM-10 * Unfiltered Data   + Std Dev 2.402831 * Average of last 20 data points (removing top & bottom 10%)   + Android Beacon Library Method   + Std. Dev 1.21705 * Limited Kalman Filter   + Std. Dev 1.1421047 |
| RSSI to Distance Conversion | * Seekcy: Average +/- 0.7060m (min 0.009 max 4.6) * HM-10: Average +/- 0.9595m (min 0.004 max 5.2)   + Possible to get lower error, but would reduce robustness * Precision: Algorithm |
| Distance to Coordinates Accuracy/Precision | * Accuracy (Assuming Accurate RSSI Values):   + Avg: +/- 0.327   + Min: +/- 0.004 m   + Max: +/- 0.785 m * Precision: to an order of 14 |
| Factors that greatly influence accuracy | * RSSI fluctuation   + RSSI's are restricted to whole numbers * Poor Calibration Formula (RSSI -meter) * Natural characteristic of a Bluetooth signal   + Any obstruction will weaken signals   + Objects located nearby will also scatter the signal |

**Appendix:**

**Hardware:**

Some hardware that has potential but not attempted for use as a locator were Arduino, RaspberryPi, and utilizing a cell phone’s native hardware through an app.

*Arduino:* Lacks onboard WiFi and Bluetooth.

*RaspberryPi:* This was not used simply because we lacked the hardware to run it.

*Cell Phone:* This is currently in development as of 2017/09/05. Considering the only functionality needed is to gather BLE signals and post the resulting data onto a server, this has promise as a way to track people within a building. However, actual testing has yet to be done due to lack of a Android phone – Bluetooth is not supported within the Android Studios Emulator and a Bluetooth dongle is needed to run a Bluetooth compatible Android emulator. [11]

*LinkIt:* Not a recommended development platform; only one out of the three LinkIts we had access to worked. Its onboard Wi-Fi connectivity is weak and also very temperamental, working one moment and not at all the next; there was no clear solution to this issue. Corrupted files also appeared in the root directory with no explanation and seemed to cause the LinkIt to reboot continually, making it impossible to ssh into the board; the only way to remove these was to factory reset the board. The ATmega32U4 lacks the processing power and memory for anything but the simplest of operations, not being able to “read a string and store 4 parts of it in an array”, as was attempted in this project. The MT7688AN lacks the ability to use the numpy and scipy python modules. Due to poor documentation and lack of userbase, the best reasoning I could derive was that OpenWrt, the Linux distribution on the MT7688AN, was lacking a certain compiler needed for the modules.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Beacons** | **RSSI Fluctuation w/ Obstacles (Office Setting)** | **RSSI Fluctuation w/o Obstacles (Hallway)** | **Configuration** | **Connection Stability** | **Power** | **Cost (RMB)** |
| Seecky | ~ +/- 20 dBm  Unreliable readings past 1 meter away from receiver | ~ +/- 15 dBm | Requires proprietary app (Seekcy Beacon)  Issues setting Major/Minor  MAC address will change unprompted at times | Casing strongly affects signal transmission  Often loses connection after 3-4 meters | Requires large coin batteries  Supposed to last 3+ years at 500 ms 0dBm | ~40 |
| HM-10 | ~ +/- 20 dBm  Unreliable readings past 5 meters away from receiver | ~ +/- 10 dBm | Easy to configure with AT-Commands  Requires TTL to USB computer and Arduino IDE  More reliable and versatile | Fairly Stable | Requires TTL to USB converter  Easy to switch out power source  Much higher energy consumption\* | ~20-40 |
| Ebeoo | ~ +/- 30 dBm  Signal is often not found | ~ +/- 10 dBm | Requires proprietary app  Unreliable (Changes in app do not always reflect changes in the beacon) | Often Unstable  Loses signal  at short ranges  Sometimes require reboot to regain signal connection | Power source built in, hard to change | ~130 |
| \*\*Xbee (hypothetical) | N/A | Claims 40 meter range outdoors with line of sight | Uses AT-Command  More versatile (more txPowers to choose from) | Assumed to have fairly stable connection | Linked to Arduino Power | ~180-200 |

**Table3: Hardware Testing Results.** All Beacons were tested with a third-party BLE scanner. Most BLE Beacons will likely have similar performance as the main source of error is the signal itself.

\* drained Seekcy batteries to half in 3 days of testing, ~27 hrs

\*\*Suggested Hardware, but needs to be tested. It is likely more precise in terms of object positioning, but figures are taken from product specifications.

**Bluetooth:**

Note that Bluetooth and BLE are not recommend for precise location-finding.

*HM-10:* A module useful for giving Bluetooth functionality to a development board without on-board Bluetooth. It uses BLE and is capable of being adjusted to suit multiple purposes, given that there is access to the Arduino IDE and a TTL-USB converter. Notably, it can be set as either a broadcaster or receiver. However, it consumes a large amount of power, so is recommended for use either with auto-sleep on or a device with a steady power supply.

*PyBluez:* A python module used for searching for and establishing Bluetooth connections. It seems to only be useable when the underlaying OS is aware of the machine’s Bluetooth capabilities. Has many dependencies, so not recommended for use on a LinkIt. Additionally, BLE support is only available on Linux machines. As the steps to download PyBluez are a little troublesome, they are included below.

*Mraa Uart Lib:* While this is the correct library to use when reading and writing serial data, it is unneeded between the ATmega32U4 and MT7688AN chipsets on the LinkIt and ATmega32U4 uses Arduino’s SoftwareSerial.h.

**RSSI stabilization methods:**

All methods were tested by being run on a sample of the 20 most recent RSI values.

*Moving Average:* Commonly used as a way to smooth out data, able to function similarly to a low-pass filter.

*Simple:* Average of previous n-data, as n increases, so does the delay in adjusting to a new RSSI range. Best method out of all types of moving average. A good balance between RSSI stabilization and time delay is seems to be at n=7. However, this value may differ as, the speed of data gathering of this system is fairly slow.

*Cumulative:* Average of all past data points; inappropriate for RSSI stabilization as old values have no bearing in what current RSSI values are.

*Weighted:* A somewhat better system than cumulative, where the most recent values are given more weight. However, this cause large skews in data when the RSSI jumps, as it often does due to interference.

*Kalman Filter:* An algorithm that keeps track of the estimated state of the system and the uncertainty of the estimate to adjust for statistical noise and other uncertainties such as signal interference. However, it is susceptible to sudden changes in input without extra data; to mitigate this effect, the Kalman filter is run on a stored set or the most recent 20 RSSI values This is currently what is implemented in the system and provides the lowest standard deviation out of all listed methods.

*Android Beacon Library Method:* The method used by the Open-Source Android Beacon Library for establish calculation constants. Takes a moving average of the past 20 values while ignoring the top and bottom 10%. Its performance is slightly worse than the Kalman Filter, but much easier to implement.

**RSSI to distance:**

*Calibration Equation:* This method is outlined in the Android Beacon Library [4]; sample RSSI data from varying distances is gathered and then fit against a power regression to form an equation, which is then used to map RSSI to distance. Tends to generate an average error of 0.7-0.9 meter.

*Heatmaps:* A method of hard coding particular ranges to RSSI to distances. Difficult to do so due to the whole number nature of RSSI signals and the ranges of RSSI which share close proximity while corresponding to a comparatively large range of distances. Due to the large amount of overlap of RSSI, it is not possible to write an accurate method for this.

*FOA:* Frequency of Arrival; makes use of the Doppler effect to find distances, often used with TDOA, however, the observation points must be in relative motion and requires a large amount of data transfer.

**Positioning Algorithms:**

*AOA/TDOA:* Angle of Arrival/Time Delay of Arrival; Determines the angle of which a signal arrives by measuring time delay of the signal at an antennae array and then uses this data to triangulate the signal origin. We do not have access to an antennae array and our current hardware setup does not allow for this method, so this could not be implemented. Additionally, TDOA requires the time difference between the signal departure and arrival.

*TOA:* Calculates distance based on the absolute time of arrival and signal velocity and then works in junction with other receivers to find a position circle where the signal origin is. Requires synchronized base stations, so could not be implemented.

*Triangulation:* Determines a point by forming triangles to it from a known point.As we were not able to get angle measurements, this method could2 not be used.

*Trilateration:* Draws circles with radii equal to the RSSI-converted distances and reports the intersection as the location of the desired object. Very accurate under the assumption the distances are accurate. However, this method is not robust enough to handle the wild fluctuations in the RSSI value, often return non-valid values. [8]

*Minimization:* Method outlined in the report; in simple terms, minimizes the error between the guessed distances and RSSI-converted distances to pin down a coordinate. [9]

**Signals:**

*WiFi:*  Accuracy is too low for our purposes. The current record is 1.96 meters by Anyplace, which combines WiFi signaling with pathfinding to achieve this.

*GPS:* Global Positioning Systems; has the same issue of limited accuracy, with error averaging between 1 and 5 meters. Additionally, they lose much of their accuracy in areas with large building or when indoors, often losing signal entirely and are commonly limited to 2-dimensions. While use of satellites is inappropriate, another method is with pseudolites, where earth-based devices perform the same function a satellite would. This method is similar to our most recent prototype, bar the different signal used.

*Bluetooth/BLE:* BLE is similar to Bluetooth with a few key differences. It has a reduced range of 250 meters, faster in making connection and has lower power consumption. However, it is incapable of transmitting anywhere near the amount of data Bluetooth but this is inconsequential, as in our IPS, we require a long-lasting system that only needs to transmit RSSI and nothing else. It is note to have an accuracy of about 5 meters on average and a little over a meter in clear conditions.

*NFC/RFID:* Near Frequency Communication/Radio Frequency ID; these have a short range of 20cm and 25 meters respective. However, they are most useful for identifying that an object has passed nearby, and not for calculating a precise position.

*UWB:* Ultra-Wide Band; a high bandwidth and short pulsed waveform not effected by line-of-sight and the ability to reduce the effect of multipath, when signals bounce off of nearby objects. Distance accuracy has been noted to be within 10 cm with clear conditions. This is a better option than BLE, but is noticeable more expensive, and it is difficult to find UWB beacons that do not already have a infrastructure attached to them. Companies like Pozyx and Estimote already use UWB with good results in their IPS systems. [10]

**Techniques:**

*Pathfinding:* A method that makes use of the fact that the object being tracked is moving and uses onboard sensors to attempt to correct for it’s location. Also, it will logically limit where the object could be based on physical constraints. This is unsuited as our tracker should be stationary for the majority of the time.

*Crowd Sourcing:* This technique is used by Tile to supplement item locator. This method works by using and phone nearby with the Tile App enable. When app detects any Tile in close enough proximity, even those belonging to someone else, it stores the phone’s current location as that Tile’s location. This is not an avenue available to use as it depends on a large existing userbase.

Resources:

[1] HM-10 as Beacon

http://www.blueluminance.com/HM-10-as-iBeacon.pdf

[2] HM-10 User Manual (TinySine)

[http://www.tinyosshop.com/datasheet/Tinysine%20Serial%20Bluetooth4%20user%20manual.pdf](http://www.tinyosshop.com/datasheet/Tinysine Serial Bluetooth4 user manual.pdf)

[3] MediaTek LinkIt Smart 7688 Quick Start Guide

<https://www.cnx-software.com/2015/12/01/quick-start-guide-for-linkit-smart-7688-board/>

[4] Android Beacon Library Distance Calculations 1

https://altbeacon.github.io/android-beacon-library/distance-calculations.html

[5] Android Beacon Library Distance Calculations 2

https://altbeacon.github.io/android-beacon-library/distance-calculations2.html

[6] Real Statistics Using Excel

http://www.real-statistics.com/regression/power-regression/

[7] Kalman Filters Explained - Removing Noise from RSSI Signals

https://www.wouterbulten.nl/blog/tech/kalman-filters-explained-removing-noise-from-rssi-signals/

[8] Trilateration using 3 latitude and longitude points, and 3 distances

https://gis.stackexchange.com/questions/66/trilateration-using-3-latitude-and-longitude-points-and-3-distances/415#415

[9] Positioning and Trilateration

<http://www.alanzucconi.com/2017/03/13/positioning-and-trilateration/>

[10] Ultra Wideband Indoor Positioning Technologies

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4883398/

[11] Emulate Android and BLE Hardware

chrislarson.me/blog/emulate-android-and-bluetooth-le-hardware.html