KAUNAS UNIVERSITY OF TECHNOLOGY
INFORMATICS FACULTY

Farid Orujov

Indoor Positioning Methods and Techniques

Master's Degree Final Project

Supervisor
Assoc. prof. dr. Rytis Maskeliunas

KAUNAS, 2016

KAUNAS UNIVERSITY OF TECHNOLOGY
INFORMATICS FACULTY

Indoor Positioning Methods and Techniques

Master's Degree Final Project
Informatics (M4016M21)

Supervisor
(signature) Assoc. prof. dr. Rytis Maskeliunas
(date)
Reviewer
(signature) Dr. Kestutis Jankauskas
(date)

Project made by
(signature) Farid Orujov
(date)

KAUNAS, 2016

KAUNAS UNIVERSITY OF TECHNOLOGY

Informatics faculty

(Faculty)

Farid Orujov

(Student's name, surname)

Informatics (M4016M21)

(Title and code of study programme)

"Indoor Positiong metods and techics"

DECLARATION OF ACADEMIC INTEGRITY

0 ___ _____ ____

Kaunas

I confirm that the final project of mine, **Farid Orujov**, on the subject "Indoor Positiong metods and techics" is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarized from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by law) have been paid to anyone for any contribution to this thesis.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

_____ _____

(name and surname filled in by hand) *(signature)*

Farid, Orujov. Indoor Positioning Methods and Techniques. Master's Final Project / supervisor assoc. prof. dr. Rytis Maskeliunas; Faculty of informatics, Kaunas University of Technology.

Keywords: Indoor Positioning, iBeacons, RSSI.

Kaunas, 2016. 53 p.

SUMMARY

Nowadays, more and more people are using smartphones. The smartphones are indispensable helpers of people in their daily routine, including functions that helps find the location on the outdoor. Meanwhile, there is no specific techniques of indoor positioning.

This article made investigation of the indoor positioning algorithms based on signal strength received from the beacons. Several experiments have been done to determine the broadcast beacons in real conditions and it is essential to use the obtained data from a 3 meters range away from the beacons, in terms of the indoor positioning algorithm.

Moreover, comparative analysis of positioning algorithms was conducted based on criteria such as effectiveness and independence from the preliminary measurement. Algorithms of Proximity localization, Centroid localization, Weighted Centroid localization, Fingerprinting Localization and Trilateration were tasted experimentally.

At least 4 beacons are essential for the algorithms, provided that the size of the room is not more than 3-4 meters, otherwise the amount of beacons will have to be increased due to the fact that its distance range is limited to 3 meters.

The trilateration algorithm and algorithm of Weighted Centroid needs no preliminary measurement as the average deviation from the actual position is 0.97 and 1.01 respectively which is relatively better than other algorithmic values.

The fingerprinting localization algorithm requires preliminary measurement, and the average deviation from the actual position is equal to 0.65 meters. After experimentation, it became clear that the algorithm of fingerprinting localization is the most suitable for BLE beacons.

TABLE OF CONTENTS

ABBREVIATIONS AND NOTATIONS **7**
LIST OF TABLES **8**
LIST OF FIGURES **9**
1. INTRODUCTION **10**
1.1 Aim 10
1.2 Document Structure 10
2. ANALYTICAL PART **11**
2.1 Bluetooth Low Energy 11
2.2 iBeacon 11
2.3 Eddystone 13
2.4 Analysis of equipment 14
2.4.1 Hardware 14
2.4.2 Firmware 14
2.5 Algorithms 15
2.5.1 Model 15
2.5.2 Proximity Localization 15
2.5.3 Centroid Localization 16
2.5.4 Weighted Centroid Localization 16
2.5.5 Weight-Compensated Weighted Centroid Localization Based on RSSI 17
2.5.6 Fingerprinting Localization 18
2.5.6.1 Nearest Neighbor 19
2.5.6.2 k-Nearest Neighbors 20
2.5.7 Trilateration Localization 20
2.6 Conclusion 22
3. DEVELOPMENT PART **23**
3.1 Functional requirements 23
3.2 Non-Functional requirements 23
3.2.1 Solution operating environment 23
3.2.2.1 Hardware Requirements 23
3.2.3.2 Software Requirements 24
3.2.4 User Interfaces 24

3.3 Solution Architecture 25
3.3.1 Use case diagram 25
3.3.2 Class diagram 26
4. EXPERIMENTAL PART 28
4.1 Signal Strength Experiment 28
4.1.1 Aim 28
4.1.2 Setup and the progress of the experiment 28
4.1.3 Results 29
4.1.4 Conclusion 31
4.2 Indoor Positioning Experiment 31
4.1.1 Aim 31
4.2.2 Setup and the progress of the experiment 31
4.2.2.1 Experimental group № 1 32
4.2.2.2 Experimental group № 2 36
4.2.3 Results 38
4.2.4 Conclusion 39
5. CONCLUSIONS 40
REFERENCES 41
APPENDIX 42
1. The source code for the classes of indoor positioning algorithms 42
1.1 MainViewController.swift 42
1.2 ProximityViewController.swift 44
1.3 CentroidViewController.swift 45
1.4 TrilaterationViewController.swift 48
1.5 FingerprintingDefiningViewController.swift 49
2. The graphical representation of User Interface 52
3. Suitable for experiments smartphones 53
4. Specification of Estimote beacon 53

ABBREVIATIONS AND NOTATIONS

CSV	Comma-Separated Values
kNN	k-Nearest Neighbors
PDU	Protocol Data Unit
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
WCL	Weighted Centroid Localization
WCWCL-RSSI	Weight-Compensated Weighted Centroid Localization Based on RSSI

LIST OF TABLES

Table 1. Distance Function 19
Table 2. The average dependence of the signal from the rotation of the smartphone. 30
Table 3. The configuration parameters of BLE beacon 32
Table 4. The experimental groups 32
Table 5. The coordinates of beacons 33
Table 6. The group of experimental location coordinates 33
Table 7. The result of algorithms for thee beacons 34
Table 8. The result of algorithms for four beacons 35
Table 9. The coordinates of pre-planned location points 36

LIST OF FIGURES

Figure 1. The structure of the data packet in iBeacon protocol 12
Figure 2. The structure of the data packet in the Eddystone protocol 13
Figure 3. Model of the environment 15
Figure 4. The scheme of the Fingerprinting Localization algorithm 18
Figure 5. Trilateration 20
Figure 6. The Estimote beacon 24
Figure 7. Use case diagram 26
Figure 8. UML Class diagram 27
Figure 9. The experiment measuring the strength of Bluetooth beacon signal 28
Figure 10. The relationships between the distance and the strength of the signal 29
Figure 11. The average error in calculating the distance 30
Figure 12. Schematic representation of the room to test the algorithms of the experimental group 1 33
Figure 13. The result of the experimental group №1 34
Figure 14. The histogram of calculation error for three beacons 34
Figure 15. A histogram of calculation errors for 4 beacons 35
Figure 16. Schematic representation of the room to test the algorithms of the experimental group 2 36
Figure 17. The result of the experimental group №2 37
Figure 18. Histogram of calculation errors for Fingerprinting algorithm 37
Figure 19. The comparison of the error of indoor positioning algorithms 38
Figure 20. The comparison of the error of indoor positioning algorithms (in a corridor) 38

1. INTRODUCTION

Currently, there is increasing interest in the possibility of obtaining information about the location of an object. The range of services will expand significantly if user's location information can be provided. The location-based services refer to applications that depend on the user's location to provide services in various categories, including navigation and tracking. Unfortunately, the GPS technology does not specify a location close to walls, buildings, trees, buildings and subways, as the power of the GPS satellite's signal is weak, making it unusable for indoor GPS localization. It is popular to use Wi-Fi hotspots for detecting location in the room. However, given the fact that the walls are an obstacle that affects the signal Wi-Fi access points that data mechanism is not effective. In this case, the quantity and location of Wi-Fi access points are very important when using wireless technology, moreover such a solution is costly.

1.1 Aim

The aim of this research is to determine the possibilities of indoor positioning algorithms using BLE beacons. The objectives of the research are the following:

1. Investigate the range broadcasting of BLE beacon in a real environment.
2. Investigate which indoor positioning algorithms using Bluetooth Smart Beacons show the highest accuracy.
3. Investigate the effect of the amount of beacons on the accuracy.

1.2 Document Structure

This paper contains 5 chapters, references and appendices. The first chapter includes the introduction, lists the purpose and objectives of the work. The second chapter is devoted to the analysis of technologies, protocols, and algorithms. The third chapter describes the processes, functions and requirements for software development. The fourth chapter focuses on the experiments and results. The fifth chapter provides conclusions.

2. ANALYTICAL PART

2.1 Bluetooth Low Energy

The Bluetooth is a wireless standard that allows electronic devices to exchange data with each other. The hallmark is a short-range, low-cost, low-power solution that communicates using radio waves within the 2.4 GHz license free band. In 2001, researchers from the company Nokia started developing a new wireless technology, based on the Bluetooth standard, which would provide lower energy consumption and to retain compatibility with classic Bluetooth technology. The result of the development was published in 2004, the prototype was called Bluetooth Low End Extension [1]. After a prolonged period of time it has become standard.

Today, Bluetooth Low Energy standard became part of the Bluetooth 4.0 specification. There are 3 types of equipment:

- Bluetooth: standard which supports only classic mode;
- Bluetooth Smart Ready: standard which supports the "classic" mode and Low Energy mode;
- Bluetooth Smart: standard which supports only Low Energy mode.

Since 2012, manufacturers have begun to implement Bluetooth Smart Ready standard in their devices. Instance Bluetooth 4.0, particularly Bluetooth Smart Ready, has become supported on Apple devices starting with the iPhone 4S.

2.2 iBeacon

The iBeacon technology was developed by Apple company and presented to the public in 2013. This standard is an extension for the Bluetooth 4.0 standard. The Bluetooth 4.0 has two modes of communication:

- advertising: one-way discovery;
- connecting: two-way communication.

The mode of advertising is important for our purpose. The device transfers the data packets in the range from 20 milliseconds to 10 seconds. It is clear that the transmission frequency is directly proportional to the speed of device detection, but inversely to the battery life of the device.

The data packets may contain up to 47 bytes of information [2]:

- Preamble (1 byte)

- Access Address (4 bytes)
- Advertising channel PDU (2-39 bytes)
    - PDU Header (2 bytes)
    - PDU MAC address (6 bytes)
    - PDU Data (0-31 bytes)
        - iBeacon prefix (9 bytes)
        - Proximity UUID (16 bytes)
        - Major (2 bytes)
        - Minor (2 bytes)
        - TX power (2 bytes)
- CRC (3 bytes)

The aforementioned data packets have been visualized in Figure 1.

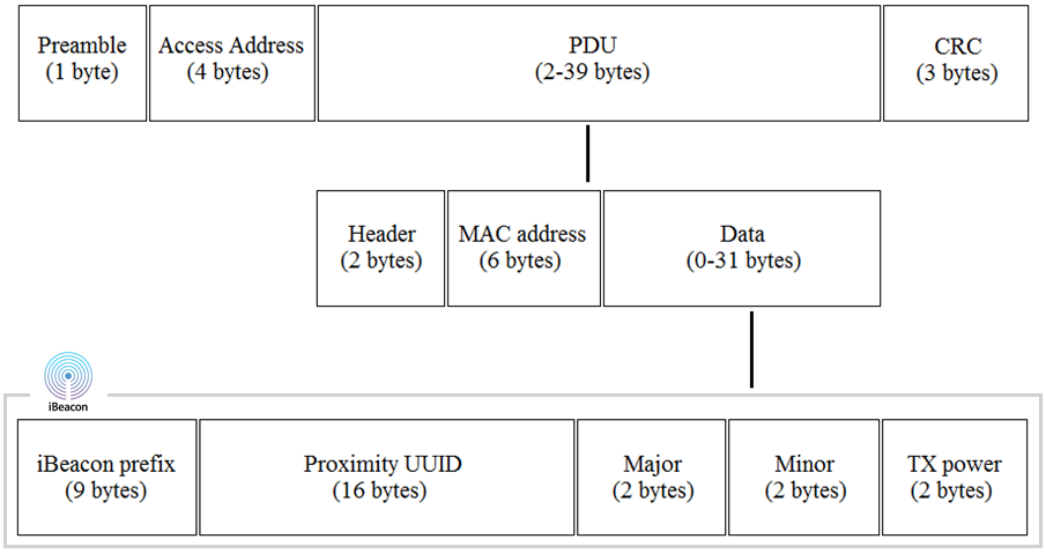


Figure 1. The structure of the data packet in iBeacon protocol

For example, the data received from the Estimote beacon have the following representation:

02 01 06 1A FF 4C 00 02 15 B9 40 7F 30 F5 F8 46 6E AF F9 25 55 6B 57 FE 6D 00 49 00 0A C5

In accordance with the format of iBeacon, a sequence of bytes is divided as follows:

iBeacon prefix	*02 01 06 1A FF 4C 00 02 15*
Proximity UUID	*B9 40 7F 30 F5 F8 46 6E AF F9 25 55 6B 57 FE 6D*

Major	*00 49*
Minor	*00 0A*
TX power	*C5*

The TX power is the strength of the signal measured at 1 meter from the iBeacon. This numeric indicator is required to determine the distance to the beacon. The algorithm for calculating a distance is based on data from the TX power (RSSI at a distance of 1 meter from iBeacon) and current RSSI (Received Signal Strength Indication). The iOS operating system takes care of the calculations and returns the result in meters. In this MA project, we will focus on data of Received Signal Strength Indicator.

2.3 Eddystone

The Google company has developed its own open format for communication with Bluetooth LE devices. This format is an alternative to the format which was presented by Apple. The composition of the data packets that are sent by BLE devices has been discussed above. The difference between the Eddystone and iBeacon is only a composition of PDU Data [3].

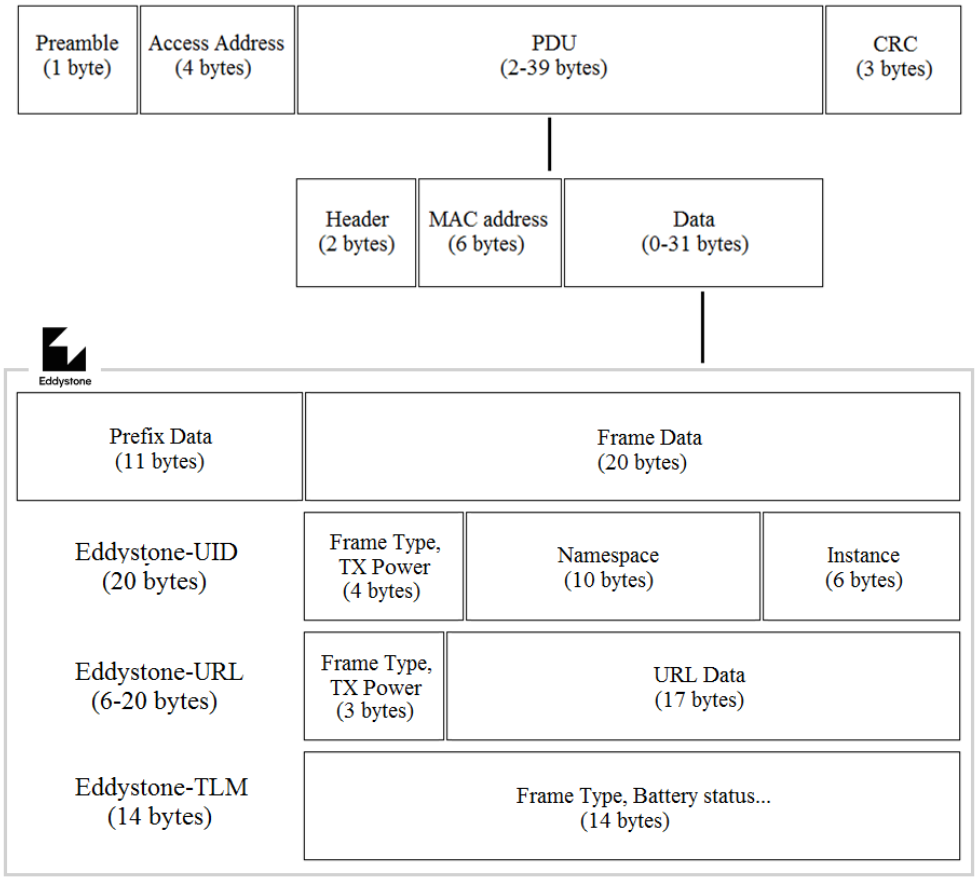


Figure 2. The structure of the data packet in the Eddystone protocol

The given standard has 3 modes:

- Eddystone-UID:
    - Namespace: The identifier which is used to separate beacons. It is an analogue of UUID which is used in standard iBeacon.
      Example: *EDD1EBEAC04E5DEFA017*
    - Instance: This identifier does the same job as Major & Minor in iBeacon.
      Example: *0BDB87539B67*
- Eddystone-URL: A compressed URL that has once been parsed and decompressed is directly used by the client.
- Eddystone-TLM: This data packet includes minor details like battery status, the temperature of beacon, etc.

2.4 Analysis of equipment

2.4.1 Hardware

The hardware consists of a microcontroller with a Bluetooth LE radio chip and a battery [4]. New radio chips are optimized for Bluetooth LE protocol. The radio chips that are used in beacons are mainly produced by two companies: Texas Instruments and Nordic Semiconductor. The beacons' manufacturers use coin cell batteries, alkaline AA batteries and an external power supply in production.

2.4.2 Firmware

Every beacon has an own firmware. It is the program code that is controlling the operation of the hardware. The firmware can monitor multiple parameters that affect the battery life.

- Transmit power (Tx) – The beacon transmits a signal of a certain power. The transmitted signal weakens with distance from the source. High transmission capacity means that the signal can travel long distances. The low transmission power means low power consumption, but affects propagation of the signal.
- Advertising Interval – The frequency with which iBeacon generates a signal. The interval of 100 ms means that the signal is distributed every 100 milliseconds. The value of the advertising interval to 500 ms means that the signal is broadcast only twice and as a result the battery power will be consumed less. The battery life increases as the advertising interval increases; however, the time of reaction of the receiver is reduced.

2.5 Algorithms

2.5.1 Model

This section describes the basic terms and model of the environment in which positioning algorithms are used. A positioned facility that receives the Bluetooth LE signals is called “agent”. In this case the "agent" means a smartphone. The model of environment includes several beacons and an agent. Without loss of generality, the space is regarded as a flat environment in which there are interferences from walls - floors, diverse signals, etc. Figure 3 illustrates an example of arrangement of the agent and the beacons in the plane, where B_i is i^{th} beacon, (X_i, Y_i) is the Cartesian coordinates of i^{th} beacon, P_i is the RSS from the i^{th} beacon, A is the agent, (X_A, Y_A) is the Cartesian coordinates of the agent, N is quantity of beacons.

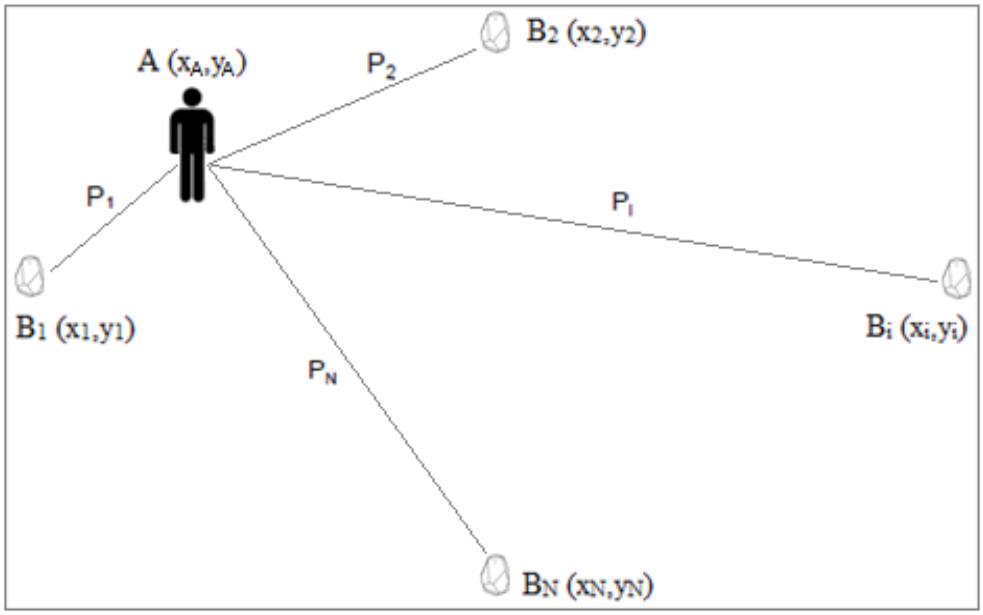


Figure 3. Model of the environment

The algorithms for detecting the location is the approach to solve the problems of determining the location of an agent-based power signals emitted by a few beacons. The algorithms described below are used to detect location of an agent in the room.

2.5.2 Proximity Localization

The proximity algorithm [5] is assigned to the agent that coordinates the beacon which emits the greatest power signal. The algorithm is the simplest, from a computational point of view. For instance, if four beacons are located in the room and the highest power signal P_1 has been received from B_1, then the agent is assigned coordinates of B_1 beacon.

The advantages of this algorithm include the ease of implementation due to the low computational complexity (O(N)) and the necessity to know only the location of the beacons. The obvious disadvantage is very low accuracy. This algorithm is useful as an initial approximation, the result of which can be used for a different algorithm.

2.5.3 Centroid Localization

The centroid algorithm [6] is a calculation of the geometric center of the plane figure formed by multiple beacons. In this case, the coordinates of the agent are calculated as a linear combination of the coordinates of the beacons. Location of the agent is determined by the following formulas:

$$\begin{cases} X_A = \frac{1}{N}\sum_{i=1}^{N} X_i \\ Y_A = \frac{1}{N}\sum_{i=1}^{N} Y_i \end{cases} \quad (1)$$

where X_A, Y_A – Cartesian coordinates of agent; X_i, Y_i is the Cartesian coordinates of i^{th} beacon; N is the number of beacons.

The advantages of this algorithm include ease of implementation, the complexity of computing is O(N) and the necessity to know only the location of the beacons. The obvious disadvantage is low accuracy. Since information about the power of the signal is not taken into account, consequently the error may reach the range of the signal broadcast by the beacon.

2.5.4 Weighted Centroid Localization

The weighted centroid algorithm [7] is an improved version of the previous algorithm by adding capacity in consideration of the received signals. Then the coordinates of the agent can be calculated as a linear combination of the coordinates of the beacons, based on signal power as a weight characteristic.

$$\begin{cases} X_A = \frac{\sum_{i=1}^{N} w_i \cdot X_i}{\sum_{j=1}^{N} w_j} \\ Y_A = \frac{\sum_{i=1}^{N} w_i \cdot Y_i}{\sum_{j=1}^{N} w_j} \end{cases} \qquad w_i = \frac{1}{d_i^g} \quad (2)$$

where X_A, Y_A – Cartesian coordinates of agent; X_i, Y_i is the Cartesian coordinates of i^{th} beacon; w_i -

weight characteristics; d_i - refers to the distance between agent and i^{th} beacon and g to the degree which determines the contribution of beacon; N is the number of beacons.

The iOS standard functions are used for getting d_i distance. In most cases, g is set to 1, although in each case, g requires different values due to different environmental conditions.

The advantages of this algorithm include ease of implementation and the need to know only the location of the beacons. The disadvantage is the dependence on the number of beacons simultaneously available to the agent. The more signals of known beacons the agent takes, the higher the accuracy of calculation of his location.

2.5.5 Weight-Compensated Weighted Centroid Localization Based on RSSI

The major improvement of WCL is presented in [8]. The method needs no calculation of distance, which makes it faster and more accurate than an original WCL method. In this case, the formula for calculating the weight uses only the signal strength RSSI. The characteristic of weight is calculated as follows:

$$w_i = \frac{\sqrt{\left(10^{\frac{P_i}{10}}\right)^g}}{\sum_{j=1}^{N} \sqrt{\left(10^{\frac{P_j}{10}}\right)^g}} \tag{3}$$

where X_A, Y_A – Cartesian coordinates of agent; X_i, Y_i is the Cartesian coordinates of i^{th} beacon; w_i - weight characteristics; P_i is the RSS from the i^{th} beacon and g to the degree which determines the contribution of beacon; N is the number of beacons.

Furthermore, the authors suggested to improve the weight characteristic by increasing the weight of the closest transmitter:

$$w_i' = w_i \cdot N^{2 \cdot w_i} \tag{4}$$

where w_i - weight characteristics; P_i is the RSS from the i^{th} beacon and g to the degree which determines the contribution of the beacon.

Determination of the agent's position is calculated using the formula (3) by replacing w_i with w_i'.

2.5.6 Fingerprinting Localization

The algorithm [5, 6, 9] approach is based on the spatial signature signal differentiation. The location of the agent is determined by comparing the currently measured signature signal power with signatures stored in a pre-formed as a database (Figure 4).

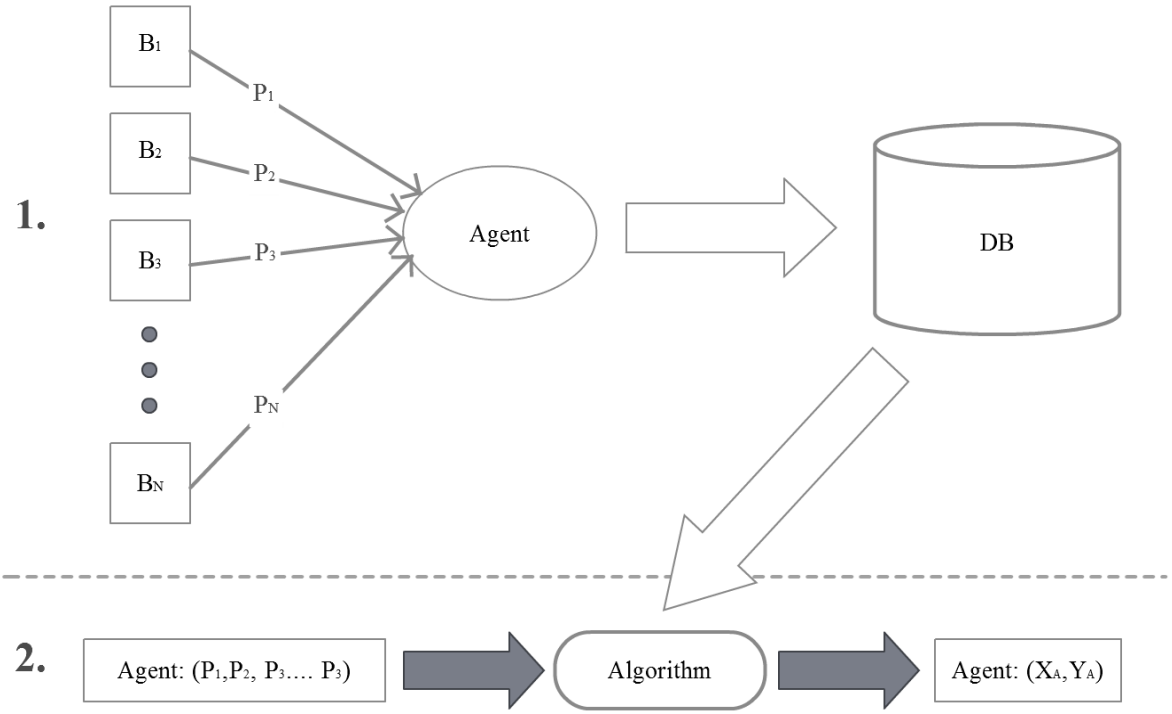


Figure 4. The scheme of the Fingerprinting Localization algorithm

The algorithm has 2 phases:

1. *The stage configuration environment.* At this stage, the power signals in pre-planned locations of all known active beacons are measured. The information collected is stored in a database with reference to the local or global coordinate space.
2. *Step positioning.* At this stage, the signal power measurements made over the agent are compared with the information stored in the database, by means of an algorithm. The algorithms of Nearest Neighbors and k-Nearest Neighbors are used in the paper [9].

The coordinates of the agent's location are obtained as a result of implementing these steps. If density of pre-planned location points is high, the algorithm returns accurate results. This implies a number of drawbacks: the need for a large amount of time for the configuration database; permanent

reconfiguration due to changes in the environment; high computational complexity of O (N • M), where M is the number of records in the database.

2.5.6.1 Nearest Neighbor

On stage, the determination of the position is required to find the nearest pre-planned locations point which is contained in the database. For this purpose, the data incoming in real time has to be compared with the data contained in the database by calculating metrics.

Table 1 illustrates several types of distance function.

Table 1. Distance Function

Euclidean	$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$	(5)
Mahalanobis	$\sum_{i=1}^{k}\lvert x_i - y_i\rvert$	(6)
Minkowski	$\left(\sum_{i=1}^{k}(\lvert x_i - y_i\rvert)^q\right)^{1/q}$	(7)

In this paper, formula (8) is used to find the Euclidean distances between the stored data and real-time data [10]:

$$Dis_i = \sqrt{\sum_{j=1}^{k}\left(P_{ij} - P'_{ij}\right)^2} \quad (8)$$

where i is i^{th} pre-planned locations point; P_{ij} is RSS from the i^{th} beacon in j^{th} pre-planned location point which stored into database; $P_{ij}{}'$ is real-time coming RSS from the i^{th} beacon in j^{th} pre-planned location point;

In the next step, one pre-planned location point is selected with the smallest Euclidean distance. The value of the coordinates of the pre-planned location points are assigned to the coordinates of the agent. The algorithm of k-Nearest Neighbors is used for choosing multitude pre-planned location points.

2.5.6.2 k-Nearest Neighbors

As mentioned above, NN is a special case of kNN when k = 1. The advantage of using multiple points is to improve the positioning accuracy. There is a possibility to use additional algorithms to approximate location.

The author [11] suggests using weighted centroid localization. He used k = 4. Coordinates of the agent are found by following the formulas:

$$\begin{cases} X_A = \sum_{i=1}^{k} w_i \cdot X_i \\ Y_A = \sum_{i=1}^{k} w_i \cdot Y_i \end{cases} \qquad w_i = \frac{Dis_{k-i}}{\sum_{j=1}^{k} Dis_j} \tag{9}$$

Where X_A, Y_A – Cartesian coordinates of agent; X_i, Y_i is the Cartesian coordinates of i^{th} beacon; k – number of pre-planned location point which have minimal Euclidean distances; w_i - weight characteristics; *Dis* - Euclidean distances.

2.5.7 Trilateration Localization

The trilateration algorithm [12] is based on a comparison of the distances from the 3 beacons to calculate the agent's location. The signal strengths of the beacons are decreasing exponentially, depending on distance between the transmitter and the receiver. Thus, this dependency can be considered as function of distance. The distance estimated by signal strength is presented as a circle with a radius around the beacon. The intersection of the broadcasting radiuses created by the three beacons provides a point or an area of receiver.

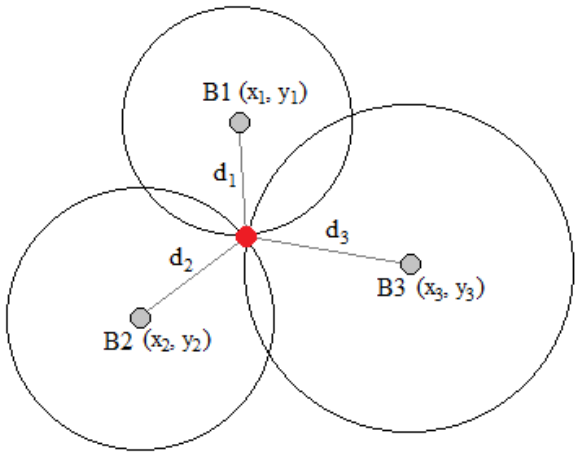


Figure 5. Trilateration

This model can be shown as such equation system [13]:

$$d_1^2 = (x - x_1)^2 + (y - y_1)^2 \tag{10}$$

$$d_2^2 = (x - x_2)^2 + (y - y_2)^2 \tag{11}$$

$$d_3^2 = (x - x_3)^2 + (y - y_3)^2 \tag{12}$$

Where x, y is coordinates of agent; x_1, x_2, x_3, y_1, y_2, y_3 is the coordinates of beacons; d_1, d_2, d_3 is the estimated distances.

This system of quadratic equations can be simplified by substituting equation 12 into equation 10 and 11, which will leave two linear equations:

$$2 \cdot (x_2 - x_1) \cdot x + 2 \cdot (y_2 - y_1) \cdot y = (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \tag{13}$$

$$2 \cdot (x_3 - x_1) \cdot x + 2 \cdot (y_3 - y_1) \cdot y = (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \tag{14}$$

The agent coordinates are found by solving equation 13 and equation 14, using Cramer's rule.

$$X_A = \frac{\begin{vmatrix} (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) & 2 \cdot (y_2 - y_1) \\ (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) & 2 \cdot (y_3 - y_1) \end{vmatrix}}{\begin{vmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \end{vmatrix}} \tag{15}$$

$$Y_A = \frac{\begin{vmatrix} 2 \cdot (x_2 - x_1) & (d_1^2 - d_2^2) - (x_1^2 - x_2^2) - (y_1^2 - y_2^2) \\ 2 \cdot (x_3 - x_1) & (d_1^2 - d_3^2) - (x_1^2 - x_3^2) - (y_1^2 - y_3^2) \end{vmatrix}}{\begin{vmatrix} 2 \cdot (x_2 - x_1) & 2 \cdot (y_2 - y_1) \\ 2 \cdot (x_3 - x_1) & 2 \cdot (y_3 - y_1) \end{vmatrix}} \tag{16}$$

The advantages of this algorithm are the low computational complexity and the necessity to know only the location of the beacons. The given algorithm is the most reliable, and its application include GPS and cellular networks.

2.6 Conclusion

The following conclusions can be drawn from the above analysis:

1. Currently, there are 2 protocols to communicate with the BLE beacons. The Eddystone protocol presented by Google offers more advanced features of user interaction, however this possibility is not suitable for our purposes. The iBeacon protocol that was developed earlier than Google's analog, has more practical material, which might be needed in the development stage.
2. As a conclusion, we can say that it is more expedient to use a battery with the BLE device for business solutions, due to easiness of installation and no need for establishing an additional power cable. However, the type of the BLE device is not very significant in the laboratory environment.
3. There are two types of indoor positioning algorithms.
    1. *Without the need for preliminary measurements*
        - Proximity
        - Centroid
        - Weighted Centroid
        - WCWCL-RSSI
        - Trilateration
    2. *Preliminary measurements are necessary*
        - Fingerprinting

Almost all the basic algorithms have low computational complexity.

3. DEVELOPMENT PART

3.1 Functional requirements

It is necessary to develop IOS application which provides testing indoor positioning algorithms. As a result of the algorithm, the system should return the Cartesian coordinates of the room. Each of the algorithms must be executed in real time without user intervention. The above-mentioned algorithms must be realized in a single application.

3.2 Non-functional requirements

The user interface must be intuitive and minimalistic. The system should work independently, without using the Internet. The mechanism of quick setup and deployment based on "just set it and forget it" should be developed in this solution. The efficiency of the system should be provided only by the presence of devices based on iOS operating system and a reasonable number of beacons.

3.2.1 Solution operating environment

In order to develop a system requires the presence of two major components:

- Physical
    - iOS device (iPhone, iPad, iPod)
    - Beacons
- Logic
    - iOS application

3.2.2.1 Hardware Requirements

Smartphones

The smartphones starting with iPhone 4s or newer and running on iOS 7 operating system or newer support the iBeacon technology. The Apple iPhone 5s was used in the development and testing of software and also in the experiments.

Beacons

The Estimote beacons are used in the solution (Figure 6). At least 4 beacons should be used in the experiments. As a result, 2 sets have to be purchased due to the fact that each kit contains 3 beacons.

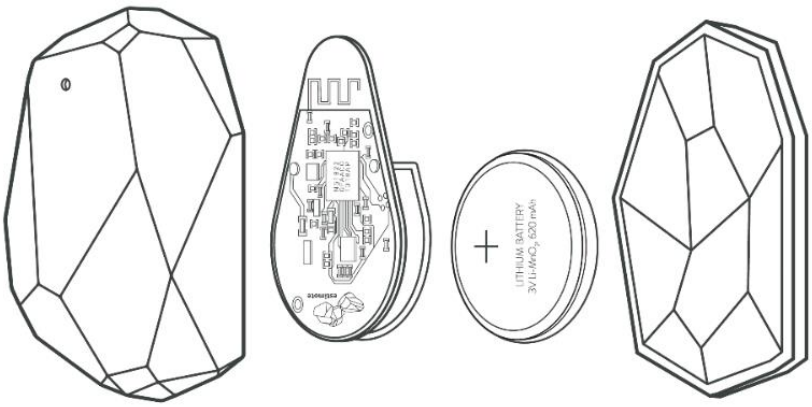


Figure 6. The Estimote beacon

3.2.3.2 Software Requirements

The application development is conducted in an environment xCode on computers that are based on the Macintosh operating system. The programming language chosen is Swift 2.0, it also is a relatively new programming language developed by Apple. It should be noted that smartphones and tablets have to be updated to iOS 7 or later versions of iOS as the iBeacon technology is available from the seventh version of the operating system.

The configuration of beacons is stored in RAM and the device's file system. The data are stored in the file system are text and use the CSV format.

3.2.4 User Interfaces

The application interface consists of 10 activities. The structure of the user interface:

- Main Screen
    - Proximity Localization Screen
    - Centroid Localization Screen
    - Weighted Centroid Localization Screen
    - Fingerprinting Localization Screen
        - Fingerprinting Localization Determination Screen
        - Configure Pre-Planned Location Point Screen
    - Trilateration Localization Screen
    - Settings Screen
        - Configure Beacon Screen

The *Main Screen* contains six buttons. Each button starts the transition to the next screen.

The *Proximity Localization Screen* contains four labels to display the results of range, the signal strength and position of a beacon. The algorithm's work is done on this screen and the result is displayed in the four labels.

The *Centroid Localization Screen* and the *Weighted Centroid Localization Screen* contain four labels and three switches. These toggle switches are used to change the Centroid algorithms, Weighted Centroid and WCWCL-RSSI.

The *Fingerprinting Localization Screen* contains three buttons. Two of the three buttons launch the transition to another screen. The function of the third button is to load the last saved configuration of the pre-planned location points.The *Fingerprinting Localization Determination Screen* contains a label to display the result of the algorithm and the button to pause the algorithm. The *Configure Pre-Planned Location Point Screen* contains two input fields, a button to generate pre-planned location points and a table of all the pre-planned location points.

The *Trilateration Localization Screen* contains a single label to display the result of the algorithm. The *Settings Screen* contains a table of all the beacons, which are in the scanner's field of visibility. The *Configure Beacon* Screen contains two input fields and a button to save the beacons coordinates.

3.3 Solution Architecture

The following algorithms must be modified:

- Centroid Localization
- Weighted Centroid Localization
- Weight-Compensated Weighted Centroid Localization Based on RSSI

Before using these indoor positioning methods, an algorithm of filtering distance should be used.

3.3.1 Use case diagram

The software functionalities are shown on UML diagram (figure 7).

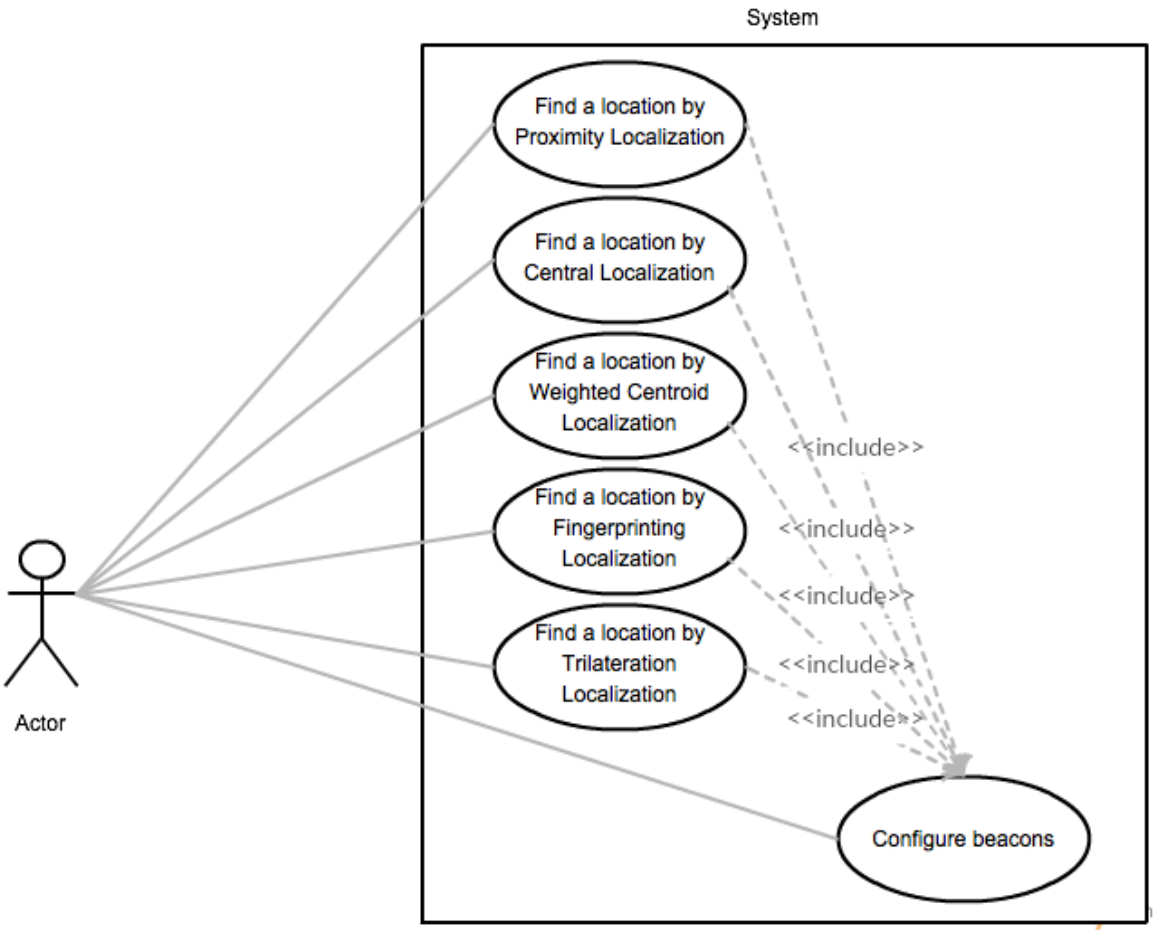


Figure 7. Use case diagram

The actor is to be understood as the system user. Also, the system should be understood as the iOS application. As described above, all indoor positioning algorithms should be implemented in the form of individual application functions.

The actor should preset the beacons before using the features in the system. By presetting of beacons is to be understood a series of simple actions. The user must specify the Cartesian coordinates of the beacons which have been installed in the room and if necessary, the user should make signal measurements to generate points.

3.3.2 Class diagram

Figure 8 illustrates the Class diagram of the mobile application. A more detailed diagram is provided in the appendix.

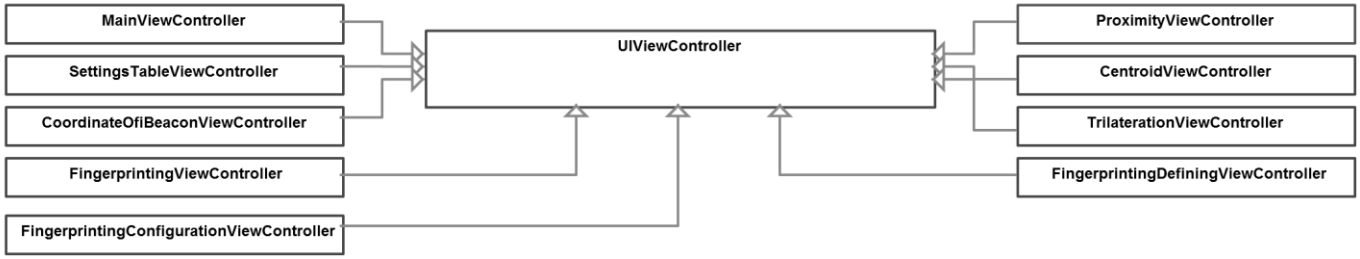


Figure 8. UML Class diagram

The third-party libraries are not used in the current solution, only a standard library of Swift 2.0 is used. The project consists of 9 classes and the storyboard.

The UIViewController is the class of graphical representation. Each class is the extension of UIViewController class.

4. EXPERIMENTAL PART

4.1 Signal Strength Experiment

This section describes an experiment conducted with the signal intensity, which uses the Proximity algorithm that was described in Chapter 2. The first part of this chapter describes the main purpose of the experiment, the second part contains detailed description of an experimental environment and the progress of the experiment. The third part shows the results obtained at the end of the experiment and the last section concludes the experiment.

4.1.1 Aim

The primary purpose of this experiment is to measure the signal strength of a Bluetooth beacon and effect of their changes. The objectives of the experiment clarify the following questions:

- What is the effect of the distance between the mobile device and the beacon on the signal strength?
- How does the orientation of mobile devices affect the signal strength?
- How big is the error in calculating the distance on the iOS platform?

4.1.2 Setup and the progress of the experiment

The testing was conducted in a variety of areas. All electronic devices which could affect the test results have been removed from the rooms. The Estimote beacon and the measuring tape were set on a wooden table (Figure 9). The experiment was conducted using the software description which has been given in Chapter 3.

Figure 9. The experiment measuring the strength of Bluetooth beacon signal

In the experiment used Proximity Location Algorithm. The values from the following parameters are collected:

- RSSI
- Accuracy

The initial distance between the smartphone and the beacon was taken no more than a few millimeters. Further measurements were carried out of remoteness from the beacon for every half a meter with rotation of the smartphone on its axis. The data collection lasted no more than 2 minutes.
For more accurate results, the experiment was repeatedly conducted using another Estimote beacon.

4.1.3 Results

Upon completion of these experiments, certain results have been obtained which are described in a few figures and tables below. Figure 10 depicts the relationships between the distance and the strength of the signal transmitted from the beacon. The data are average values of the results that have been obtained from the series of experiments conducted in different premises.

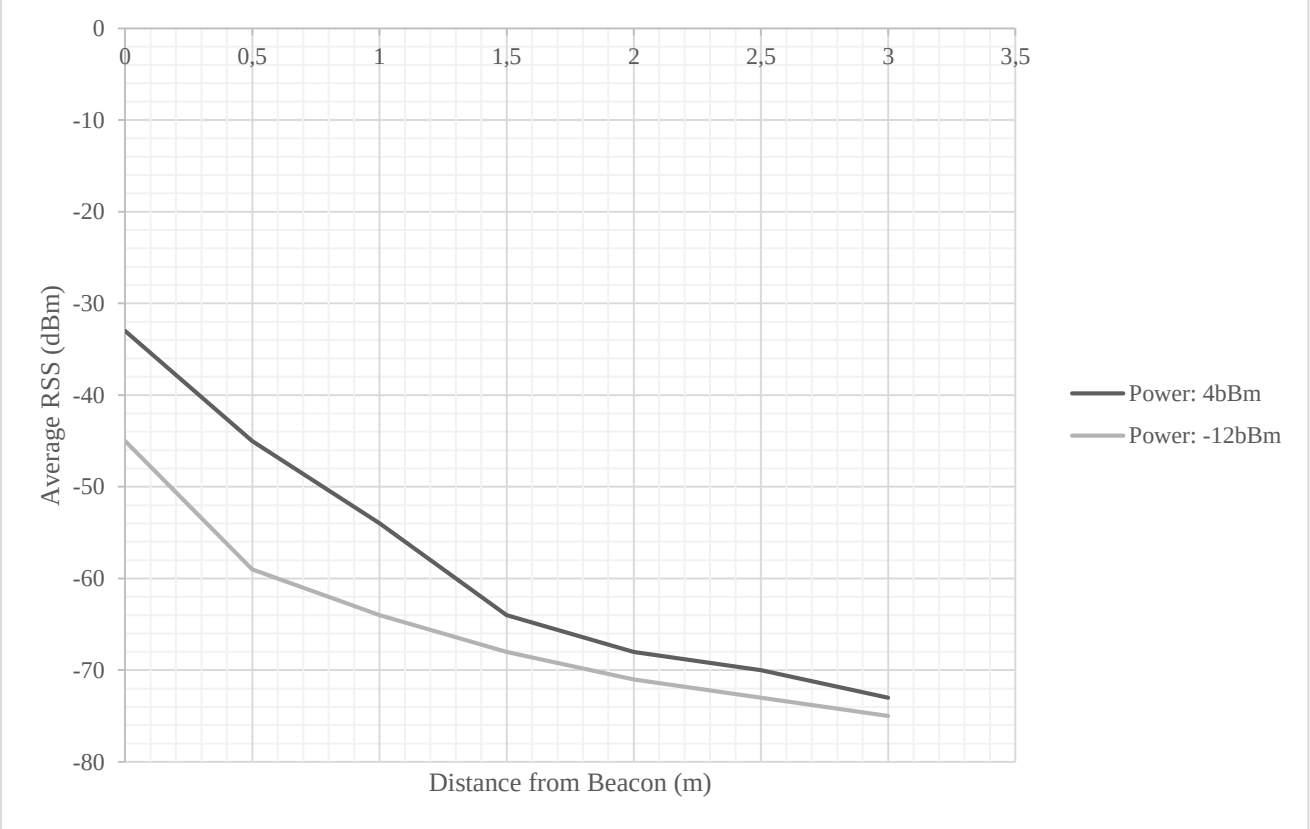


Figure 10. The relationships between the distance and the strength of the signal

The graph shows that at low power, the RSS values in the initial stages are very different, which may affect the results of the determination of the location.

With a distance of 0-1.5 meters, indications of signals on a smartphone, with different settings of transmit power, differ considerably, almost by -20 dBm. If the distance exceeds 1.5 meters, the difference in indications of signals start disappearing. With a distance of more than 3 meters, indications of signals tend to fluctuate from -77 to -80 dBm. It follows that, aiming at efficient performance of positioning

algorithms, the data obtained from beacons should only be used within a distance of 3 meters because if the mentioned distance is exceeded, the differences in data will disappear and the data becomes useless.

The manufacturer ensures that beacons can broadcast signals within a distance of 70 meters at their highest settings. However, the experiments have proved that under real conditions beacons can broadcast signals just within a distance of 10 meters. Nevertheless, rotation of the smartphone does not affect the RSS that is shown in Table 2.

Table 2. The average dependence of the signal from the rotation of the smartphone.

	0 m	0.5 m	1 m	1.5 m	2 m	2.5 m	3 m
0°	-34	-45	-52	-63	-67	-69	-73
90°	-33	-45	-54	-63	-68	-71	-73
180°	-35	-47	-53	-65	-68	-70	-72
270°	-34	-46	-53	-64	-67	-70	-73

Signal measurements taken within a distance of up to 3 meters prove that rotation of the smartphone does not have a marked influence on the strength of a signal.

Figure 11 shows the average error in calculating the distance between the iPhone 5s and the Estimote beacon. The calculations of the distance are performed by iOS operating system. However, the formula for computing the distance is not known.

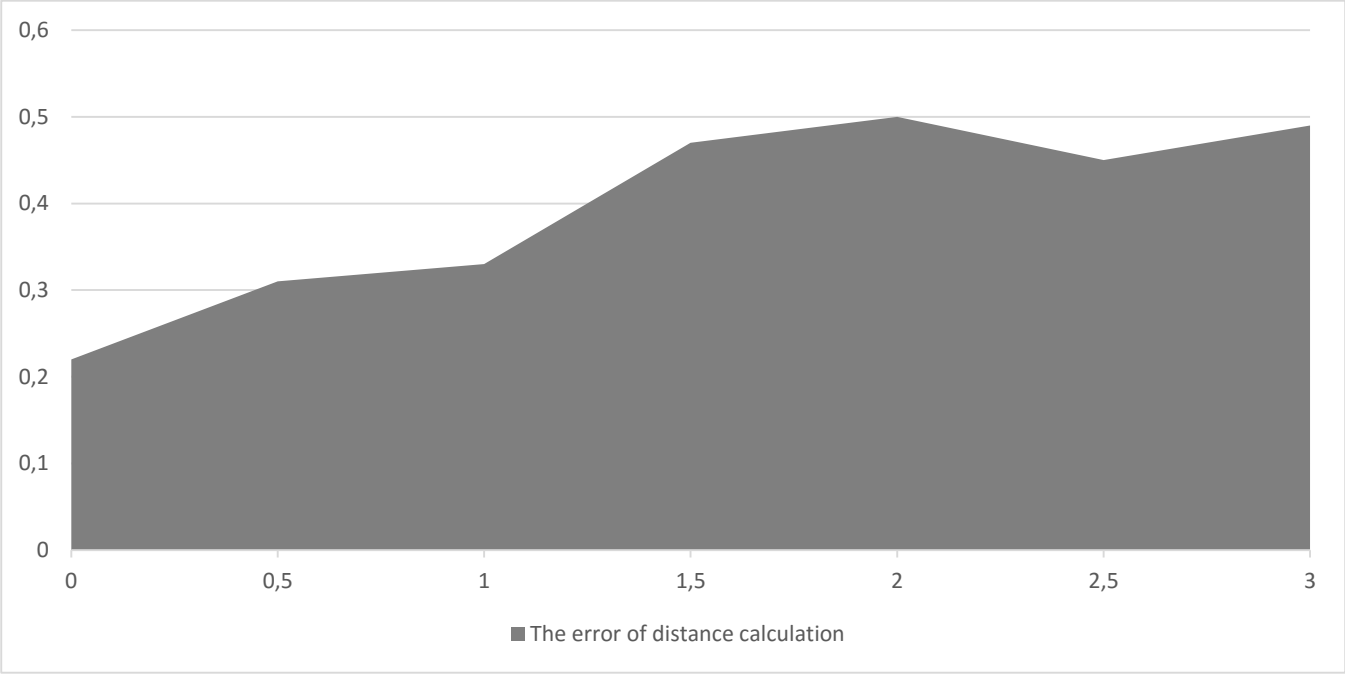


Figure 11. The average error in calculating the distance

Within a distance of 0-1 meter, the error of distance calculation is limited to the maximum of 0.3 meters; however, within a distance of 1-2.5 meters, the maximum error is 0.5 meter. It should be noted that current indicators fluctuate over a period of time.

In general, an error in the calculation is small. However, if an obstacle is located between a smartphone and a beacon, an error value will significantly grow.

4.1.4 Conclusion

It can be stated that:

- not only the distance affects the RSS, but also the power with which signal is generated;
- the rotation of a smartphone does not affect the RSS;
- in general, the error is small, if there are no obstacles in the signal path.

4.2 Indoor Positioning Experiment

This section describes the experiments which used algorithms to determine indoor location which have been described in Chapter 2.

The first part of this chapter describe the purpose of the experiment, the second description of the experimental environment and experimental models, as well as the progress of the experiment. The third section provides the results for each algorithm obtained at the end of the experiment.

4.1.1 Aim

The primary purpose of this experiment is to measure the signal strength Bluetooth beacon and effect of their changes. The objectives of the experiment are clarifying the following questions:

- What types of indoor positioning algorithms are suitable for which types of rooms?
- What is the minimum amount of BLE beacons should be used depending on the type of algorithm?
- What is the optimal place for the installation beacons to minimize the impact of obstacles?

4.2.2 Setup and the progress of the experiment

As mentioned earlier, testing was conducted in a variety of areas. For clarity, we describe one of the tested rooms. A specific environment of 4.64 by 4.64 in meters has been simulated in order to test the algorithms. By using the software provided by the manufacturer, we have set the maximum power and frequency signal of the beacon, based on the results of the previous experimentation.

Table 3. The configuration parameters of BLE beacon

Parameter Name	Value
Transmit power (Tx)	4 dBm
Advertising Interval	200 ms

All electronic devices which could affect the test results have been removed from the rooms. Also in this room, if possible, we collected objects that could reflect or absorb signals.

As noted above, there are two types of indoor positioning algorithms:

1. Which requires the use of preliminary measurements;
2. Which does not require the use of preliminary measurements.

Consequently, there is a need to test the algorithms in the different models with regard to their environment types.

Table 4. The experimental groups

The experimental group № 1	The experimental group № 2
Proximity Localization Centroid Localization Weighted Centroid Localization Weight-Compensated Weighted Centroid Localization Based on RSSI Trilateration Localization	Fingerprinting

4.2.2.1 Experimental group № 1

Figure 12 shows an experimental model of the room. This model uses four beacons for testing algorithms in experimental group № 1. There is a possibility of installing four or more beacons. It should be noted that the beacons are installed on each of the walls and at the same horizontal level.

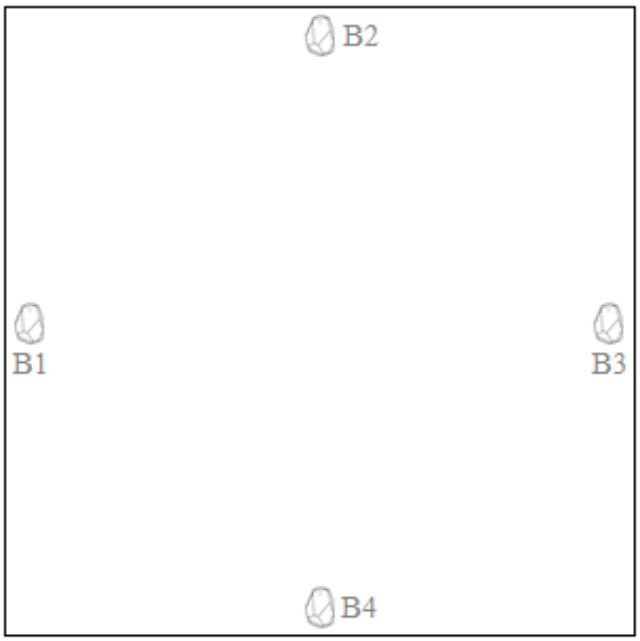


Figure 12. Schematic representation of the room to test the algorithms of the experimental group 1

The beacons are installed according to figure 12 and set the corresponding coordinates in the Cartesian coordinate system:

Table 5. The coordinates of beacons

Beacons	Coordinates (in meters)
B1	(0.00, 2.32)
B2	(2.32, 4.64)
B3	(4.64, 2.32)
B4	(2.32, 0.00)

During the experiments, the points that are indicated in Figure 13 are ought to determine the location. This points are divided to the three groups in Table 6:

Table 6. The group of experimental location coordinates

The name of groups	Real location coordinates
Red	(2.32, 2.32)
Green	(1.55, 3.87)
Blue	(3.87, 1.55)

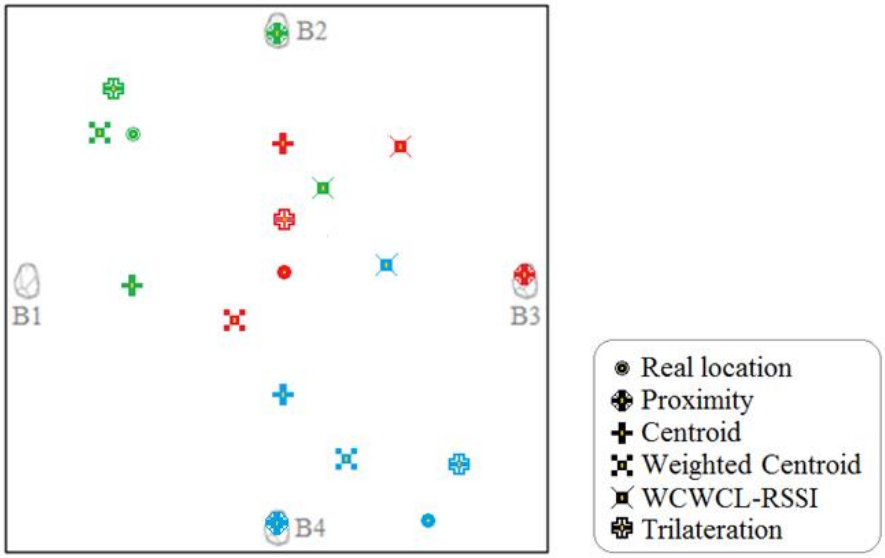


Figure 13. The result of the experimental group №1

Table 7 shows the result of algorithms:

Table 7. The result of algorithms for thee beacons

Color group name	Proximity	Centroid	Weighted Centroid	WCWCL-RSSI	Trilateration
Red	(4.64, 2.32)	(2.32, 3.09)	(1.32, 1.91)	(3.30, 3.93)	(2.32, 1.21)
Green	(2.32, 4.64)	(1.54, 2.32)	(1.40, 3.77)	(3.01, 3.42)	(1.84, 3.42)
Blue	(2.32, 0.00)	(2.31, 1.53)	(2.52, 2.43)	(3.33, 3.05)	(2.53 2.01)

Figure 14 shows calculation error for each algorithm in meters:

Figure 14. The histogram of calculation error for three beacons

The nearest three of four beacons have been used in the experiment for Weighted Centroid and WCWCL-RSSI. The Proximity Localization shows the highest value of error; it is 2.32 meters. The Trilateration localization shows the smallest error value; The error value is an average of 1 meter.

Table 8 shows the results have been obtained simultaneously using four beacons.

Table 8. The result of algorithms for four beacons

Color group name	Weighted Centroid	WCWCL-RSSI
Red	(1.31, 2.1)	(1.52, 3.44)
Green	(1.53, 3.24)	(3.20, 2.11)
Blue	(2.42, 1.63)	(3.33, 0.9)

Figure 15 shows the error of calculation in meters:

Figure 15. A histogram of calculation errors for 4 beacons

The experimental results showed that increase in the number of beacons does not affect the accuracy of determining the position in the experimental room, in contrast to the corridors. Probably, the reason is that the signals from the beacons have not fully covered the room. Therefore, an increase in the number of beacons will makes a slight improvement of the positioning accuracy.

As noted above, the experiments have been conducted not only in rooms, but also in the passageways. It should be noted that the algorithms of the Proximity Localization and Centroid Localization showed the worst result with any number of beacons. Moreover, none of the algorithms in the experimental group №1 have shown the expected result in long and narrow rooms like corridors.

The beacons for a given experimental group have been established vertically on the walls at the level of 1.5 meters from the floor. Given the specificity of these algorithms, the installation of beacons on the ceiling is an unreasonable step in contrast to the algorithms of the second experimental group.

4.2.2.2 Experimental group № 2

In this model, six beacons are used for testing Fingerprinting algorithm. The beacons have been installed on the walls as well as on the ceiling.

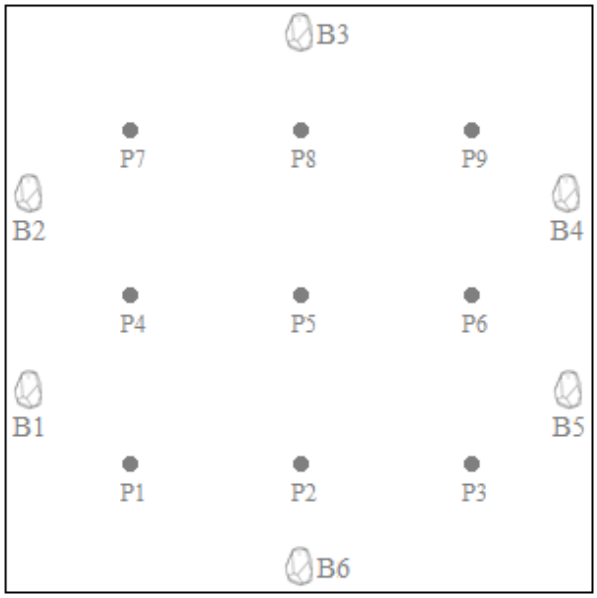


Figure 16. Schematic representation of the room to test the algorithms of the experimental group 2

The beacons are installed according to Figure 16. The RSS measurement were held for each points and corresponding coordinates were assigned as shown in Table 9:

Table 9. The coordinates of pre-planned location points

Fingerprint Points	Coordinates
P1	(1.16, 1.16)
P2	(2.32, 1.16)
P3	(3.48, 1.16)
P4	(1.16, 2.32)
P5	(2.32, 2.32)
P6	(3.48, 2.32)
P7	(1.16, 3.48)

P8	(2.32, 3.48)
P9	(3.48, 3.48)

During the experiments, the points that are indicated in Figure 17 have been committed attempts to determine the location.

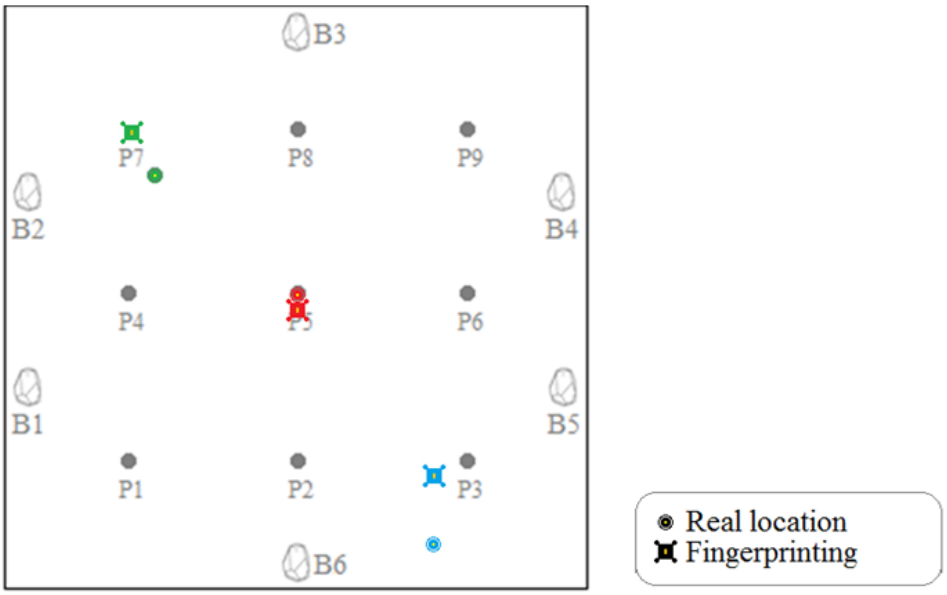


Figure 17. The result of the experimental group №2

Figure 18 shows a calculation error in meters:

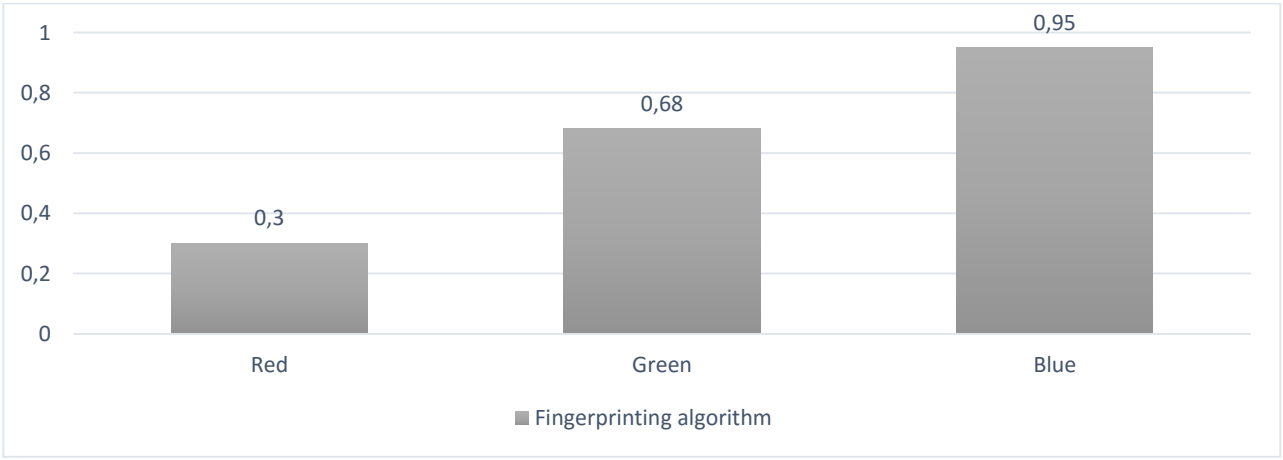


Figure 18. Histogram of calculation errors for Fingerprinting algorithm

The test results may be noted that the algorithm has shown good results not only in an ordinary room, but also in the passageways. Considering the specifications, the algorithms should not reduce the number of beacons, otherwise, it leads to errors in the approximation. During experimentation beacons were

installed on the walls as well as on the ceiling, the result in both cases can be specified as satisfactory, the error is less than a meter.

4.2.3 Results

Upon the completion of these experiments, results were obtained as described in the figure below. The duration of the testing conducted was 5 minutes for each algorithm and as a result has taken the average value of the error for each algorithm.

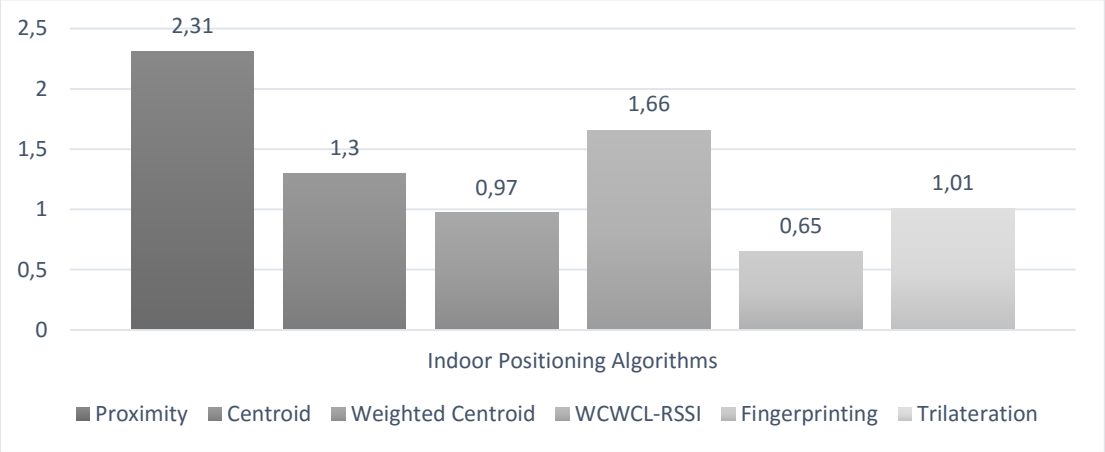


Figure 19. The comparison of the error of indoor positioning algorithms

As it can be seen from Figure 19, the Fingerprinting algorithm showed the highest accuracy in determining indoor location, an error which was 0.65 meters whereas the Proximity algorithm showed the worst result and the error was 2.31 meter. The results of this algorithm can be improved by reducing the parameter *k* in the kNN algorithm, the error increases due to the fact that it is not taken into account the distance from the beacon which the greatest signal strength.

Figure 9 presents the results of an experiment have been conducted in the corridors:

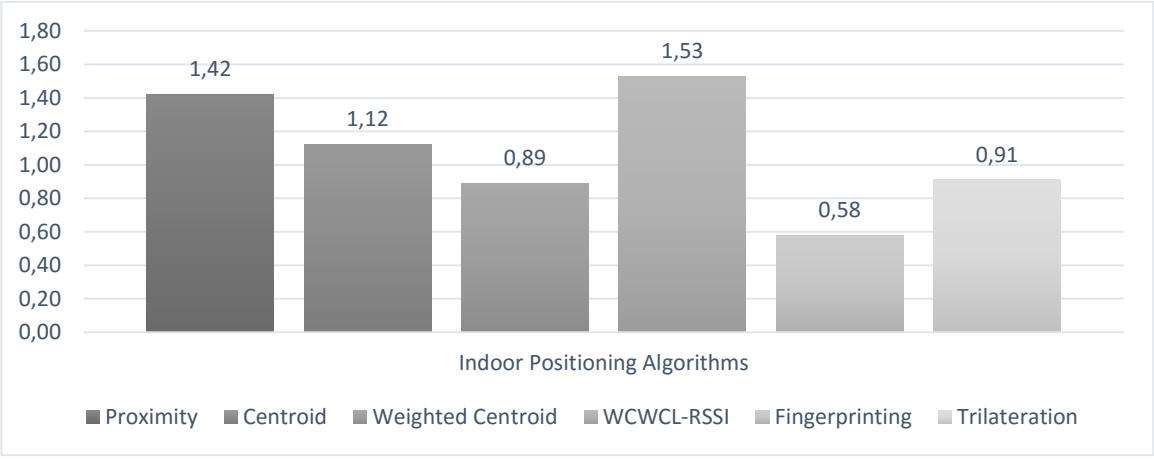


Figure 20. The comparison of the error of indoor positioning algorithms (in a corridor)

The experiments conducted in the corridors also showed the best result at the Fingerprinting algorithm. Although fairness it should be noted that the algorithms of Weighted Centroid and Trilateration also have shown good results. The remaining algorithms have a lack, when agent is moving, the result in coordinates is showing outside of the room. In avoidance errors of this type, we propose to create a map of the room and use room boundaries in computing.

4.2.4 Conclusion

It can be stated that:

- algorithms of the first group are not suitable for long narrow room in contrast to the second group of algorithms;
- the results of several experiments prove that four beacons are necessary but not sufficient to determine the location;
- the place for installation should be chosen where the beacon faces no obstacles: for the first group of algorithms, beacons may be installed on the walls and on cabinets, while for the second group, they may be installed on the ceiling.

5. CONCLUSIONS

Overall, these results can be evaluated as positive because the room has been fully covered by signal. The number of beacons was established from 3 to 6 and during testing it was found that the amount fully covers the room up to 5 meters. It can be stated that:

- The range of signal broadcast in the real world does not correspond to the range that has been declared by the manufacturer. It can be mentioned that the signal is practically damped by the wall. For an efficient operation of indoor positioning algorithms, the data obtained from beacons has to be from a 3 meters' range.
- According to my results, the Fingerprinting algorithm can be used as an indoor positioning algorithm using BLE beacons. The algorithm showed relatively high positioning accuracy that distinguishes from others, however, the main disadvantage is the phase of pre-configuration. The error of calculation is 0.67 meters.
- In general, we can say that the greater the number of beacons, the better the result of the positioning. From the experimental results, we can conclude the longer the wall, the more the beacons will be required for the correct operation of algorithms with consideration of the maximum range of broadcasting in real environment.

REFERENCES

[1] M. Honkanen, A. Lappetelainen, K. Kivekas, Low end extension for Bluetooth, Radio and Wireless Conference, 2004 IEEE, pp. 199 - 202

[2] Matthew S. Gast, Building Applications with iBeacon, O'Reilly Media, 2014, pp. 16-18

[3] Kontakt Knowledge Base, May 20th 2016. Available at: https://support.kontakt.io/hc/en-gb/articles/206853889-Beacon-profile-Eddystone/

[4] A Comprehensive Report by Aislelabs, May 20th 2016. Available at: http://www.aislelabs.com/reports/beacon-guide/#introbeacons

[5] Axel Küpper, Location-Based Services: Fundamentals and Operation, Wiley, 2005, pp. 365

[6] Kolodziej K.W., Hjelm J., Local Positioning Systems: LBS Applications and Services, CRC Press, 2006, pp. 445

[7] Shyi-Ching Liang, LunHao Liao and Yen-Chun Lee, Localization Algorithm based on Improved Weighted Centroid in Wireless Sensor Networks, Journal of Networks, Vol 9, No 1 (2014), pp. 183-189

[8] Quande Dong and Xu Xu, A Novel Weighted Centroid Localization Algorithm Based on RSSI for an Outdoor Environment, Journal of Communications, vol. 9, no. 3 (2014), pp. 279-285,

{9] Paramvir Bahl and Venkata N. Padmanabhan: RADAR: An in-building RF-based user location and tracking system. IEEE INFOCOM (2000) pp. 775-784

[10] Landu Jiang: A WLAN Fingerprinting Based Indoor Localization Technique, University of Nebraska-Lincoln, 2012, pp. 38

[11] Wenhuan Chi, Yuan Tian, Mznah Al-Rodhaan, Abdullah Al-Dhelaan and Yuanfeng Jin: A Revised Received Signal Strength Based Localization for Healthcare,IEEE Transactions on Consumer Electronics 55(3), 2009, pp.1295 - 1299

[12] Shchekotov M. Indoor Localization Method Based on Wi-Fi Trilateration Technique // Proceedings of the 16th Conference of Open Innovations Association FRUCT, Oulu, Finland, October 27-31, 2014, pp. 177-179

[13] Robert Jarvis. Arthur Mason: Indoor Positioning System - Department of Electrical and Computer. EE 4820. 2011, pp. 11-14

APPENDIX

1. The source code for the classes of indoor positioning algorithms

1.1 MainViewController.swift

```
import UIKit
struct settedBeacon {
    var Minor: NSNumber
    var X: Double
    var Y: Double
}
struct fingerPrintingBeacon {
    var averageRSSI:Int
    var averageAccuracies:Double
    var RSSI_Max:Int
    var RSSI_Min:Int
    var Accuracy_Max:Double
    var Accuracy_Min:Double
    var CountOfLoses:Int
}
struct fingerPrintingPoint {
    var X: Double
    var Y: Double
    var Beacons: [Int:fingerPrintingBeacon] = [:]
    var EuclideanDistance:Double
    var EuclideanDistance2:[Double]
    var RSSI: String // text for cell
}
protocol MainViewControllerDelegate: class {
    func getcoordinateOfiBeacons(_settedBeacons: [settedBeacon])
}
class MainViewController: UIViewController,MainViewControllerDelegate {
    var settedBeacons = [settedBeacon]();
    func getcoordinateOfiBeacons(_settedBeacons: [settedBeacon]){
        settedBeacons = _settedBeacons;
        print(settedBeacons.count)
    }
    @IBAction func clickOnProximityBtn(sender: AnyObject) {
        self.performSegueWithIdentifier("goToProximity", sender: self)
    }
    @IBAction func clickOnCentroid(sender: AnyObject) {
        self.performSegueWithIdentifier("goToCentroid", sender: self)
    }
    @IBAction func clickOnWCentroid(sender: AnyObject) {
        self.performSegueWithIdentifier("goToWCentroid", sender: self)
    }
    @IBAction func clickOnTrilateration(sender: AnyObject) {
        self.performSegueWithIdentifier("goToTrilateration", sender: self)
    }
    @IBAction func clickOnFingerprinting(sender: AnyObject) {
        self.performSegueWithIdentifier("goToFingerprinting", sender: self)
    }
    @IBAction func clickOnSettingsBtn(sender: AnyObject) {
        self.performSegueWithIdentifier("goToSettings", sender: self)
    }

    override func viewDidLoad() {
```

```
        super.viewDidLoad()
        let alert = UIAlertController(title: "Attention", message: "You have to
configure iBeacon before find your location", preferredStyle:
UIAlertControllerStyle.Alert)
        alert.addAction(UIAlertAction(title: "OK", style: UIAlertActionStyle.Default,
handler: { action in
            switch action.style{
            case .Default:
                print("default")
                self.performSegueWithIdentifier("goToSettings", sender: self)
            case .Cancel:
                print("cancel")
            case .Destructive:
                print("destructive")
            }
        }))
        self.presentViewController(alert, animated: true, completion: nil)
    }

    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        self.navigationItem.backBarButtonItem = UIBarButtonItem(title: "", style:
UIBarButtonItemStyle.Plain, target: self, action: nil)
        if (segue.identifier == "goToProximity") {
            let destination = segue.destinationViewController as!
ProximityViewController
            destination.settedBeacons = settedBeacons;

        }
        if (segue.identifier == "goToCentroid") {
            let destination = segue.destinationViewController as!
CentroidViewController
            destination.settedBeacons = settedBeacons;

        }
        if (segue.identifier == "goToWCentroid") {
            let destination = segue.destinationViewController as!
CentroidViewController
            destination.settedBeacons = settedBeacons;
            destination.weightedCentroidMode = true
        }
        if (segue.identifier == "goToTrilateration") {
            let destination = segue.destinationViewController as!
TrilaterationViewController
            destination.settedBeacons = settedBeacons;
        }
        if (segue.identifier == "goToSettings") {
            let destination = segue.destinationViewController as!
SettingsTableViewController
            destination.delegate = self
            destination.settedBeacons = settedBeacons;
        }
        if (segue.identifier == "goToFingerprinting") {
            let destination = segue.destinationViewController as!
FingerprintingViewController
            destination.settedBeacons = settedBeacons;
        }
    }
}
```

1.2 ProximityViewController.swift

```
import UIKit
import CoreLocation
class ProximityViewController: UIViewController, CLLocationManagerDelegate {
    @IBOutlet weak var proximity_lbl: UILabel!
    @IBOutlet weak var accuracy_lbl: UILabel!
    @IBOutlet weak var rssi_lbl: UILabel!
    @IBOutlet weak var coord_lbl: UILabel!
    var settedBeacons = [settedBeacon]()
    let locationManager = CLLocationManager()
    let region = CLBeaconRegion(proximityUUID: NSUUID(UUIDString: "B9407F30-F5F8-
466E-AFF9-25556B57FE6D")!, identifier: "Estimotes")
let colors = [
        19901: UIColor(red: 84/255, green: 77/255, blue: 160/255, alpha: 1),
        39081: UIColor(red: 142/255, green: 212/255, blue: 220/255, alpha: 1),
        6604: UIColor(red: 162/255, green: 213/255, blue: 181/255, alpha: 1),

        39588: UIColor(red: 84/255, green: 77/255, blue: 160/255, alpha: 1),
        44334: UIColor(red: 142/255, green: 212/255, blue: 220/255, alpha: 1),
        5270: UIColor(red: 162/255, green: 213/255, blue: 181/255, alpha: 1)
    ]

    override func viewDidLoad() {
        super.viewDidLoad()
        locationManager.delegate = self
        if (CLLocationManager.authorizationStatus() !=
CLAuthorizationStatus.AuthorizedWhenInUse) {
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.startRangingBeaconsInRegion(region)
    }

    func locationManager(manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], inRegion region: CLBeaconRegion) {
        var knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
        knownBeacons = beacons.filter{
            for settedBeacon in settedBeacons {
                if ($0.minor == (settedBeacon.Minor)){
                    return true
                }
            }
            return false
        }

        knownBeacons.sortInPlace { (element1, element2) -> Bool in
            return element1.rssi > element2.rssi
        }
        if (knownBeacons.count > 0) {
            let closestBeacon = knownBeacons[0] as CLBeacon
            self.view.backgroundColor = self.colors[closestBeacon.minor.integerValue]
            proximity_lbl.text = textForProximity(closestBeacon.proximity)
            accuracy_lbl.text = String(closestBeacon.accuracy)
            rssi_lbl.text = String(closestBeacon.rssi)

            let coord = settedBeacons.filter({$0.Minor == closestBeacon.minor}).first
            coord_lbl.text = "X: \(coord!.X) | Y: \(coord!.Y)"
        }
```

```
    }
    func textForProximity (proximity: CLProximity) -> String
    {
        switch (proximity) {
            case CLProximity.Far:
                    return "Far";
            case CLProximity.Near:
                    return "Near";
            case CLProximity.Immediate:
                    return "Immediate";
            default:
                    return "Unknown";
        }
    }
}
```

1.3 CentroidViewController.swift

```
import UIKit
import CoreLocation
class CentroidViewController: UIViewController, CLLocationManagerDelegate{
    @IBOutlet weak var coord_lbl: UILabel!
    @IBOutlet weak var mode_swch: UISwitch!
    @IBOutlet weak var weightedCentroidMode_swch: UISwitch!
    @IBOutlet weak var WCWCL_RSSIMode_swch: UISwitch!
    @IBAction func changeMore3Mode_swch(sender: UISwitch) {
        weightedCentroidParameter = 0.0
    }
    @IBAction func changeWCWCL_RSSIMode_swch(sender: UISwitch) {
        if (sender.on){
            self.title = "Weight-Compensated Weighted Centroid"
            WCWCL_RSSIMode = true
        }else{
            if (weightedCentroidMode){
                self.title = "Weighted Centroid"
            }else{
                self.title = "Centroid"
            }
            WCWCL_RSSIMode = false
        }
        weightedCentroidParameter = 0.0
    }
    @IBAction func changeWeightedCentroidMode(sender: UISwitch) {
        if (sender.on){
            self.title = "Weighted Centroid"
            weightedCentroidMode = true
            WCWCL_RSSIMode_swch.enabled = true
        }else{
            self.title = "Centroid"
            weightedCentroidMode = false
            WCWCL_RSSIMode_swch.setOn(false, animated: true)
            WCWCL_RSSIMode_swch.enabled = false
            WCWCL_RSSIMode = false
        }
        weightedCentroidParameter = 0.0
    }
    var weightedCentroidMode:Bool = false
    var WCWCL_RSSIMode:Bool = false
    var weightedCentroidParameter:Double = 0.0
```

```
    var settedBeacons = [settedBeacon]()
    let locationManager = CLLocationManager()
    let region = CLBeaconRegion(proximityUUID: NSUUID(UUIDString: "B9407F30-F5F8-
466E-AFF9-25556B57FE6D")!, identifier: "Estimotes")

    override func viewDidLoad() {
        super.viewDidLoad()
        locationManager.delegate = self
        if (CLLocationManager.authorizationStatus() !=
CLAuthorizationStatus.AuthorizedWhenInUse) {
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.startRangingBeaconsInRegion(region)
    }

    override func viewWillAppear(animated: Bool) {
        if (weightedCentroidMode){
            self.title = "Weighted Centroid"
        }
        weightedCentroidMode_swch.setOn(weightedCentroidMode, animated: true)
        WCWCL_RSSIMode_swch.enabled = weightedCentroidMode
    }

 func locationManager(manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], inRegion region: CLBeaconRegion) {
        var knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
        var userLocation: [Double]
        knownBeacons = beacons.filter{
            for settedBeacon in settedBeacons {
                if ($0.minor == (settedBeacon.Minor)){
                    return true
                }
            }
            return false
        }
        knownBeacons.sortInPlace { (element1, element2) -> Bool in
            return element1.rssi > element2.rssi
        }
        if (knownBeacons.count > 0) {
            if (weightedCentroidMode){
                userLocation = calculateWeightedCentroid(knownBeacons, more3:
mode_swch.on)
            }else{
                userLocation = calculateCentroid(knownBeacons, more3: mode_swch.on)
            }
            coord_lbl.text = "X: \(userLocation[0]) | Y: \(userLocation[1])"
        }
    }

    func calculateCentroid(beacons: [CLBeacon], more3:Bool) -> [Double]{
        var userLocationX = 0.0;
        var userLocationY = 0.0;
        var beaconsCount = beacons.count;
        if (!more3){
            beaconsCount = 3
        }
        for i in 0 ..< beaconsCount{
            let closestBeacon = beacons[i] as CLBeacon
            let coord = settedBeacons.filter({$0.Minor == closestBeacon.minor}).first
```

```
            userLocationX = userLocationX + coord!.X
            userLocationY = userLocationY + coord!.Y
        }
        userLocationX = Double(round(1000*( userLocationX/Double(beaconsCount)
))/1000)
        userLocationY = Double(round(1000*( userLocationY/Double(beaconsCount)
))/1000)
        return [userLocationX,userLocationY]
    }

    func calculateWeightedCentroid(beacons: [CLBeacon], more3:Bool) -> [Double]{
        var userLocationX = 0.0;
        var userLocationY = 0.0;
        var beaconsCount = beacons.count;
        if (!more3){
            beaconsCount = 3
        }
        for i in 0 ..< beaconsCount {
            let closestBeacon = beacons[i] as CLBeacon
            let coord = settedBeacons.filter({$0.Minor == closestBeacon.minor}).first
            let wCP =
calculateWeightedCentroidParameter(beacons,currentBeaconInArray: i, more3: more3)
            if((coord!.X * wCP).isNormal){
                userLocationX = userLocationX + coord!.X * wCP
            }
            if((coord!.Y * wCP).isNormal){
                userLocationY = userLocationY + coord!.Y * wCP
            }
        }
        userLocationX = Double(round(1000*( userLocationX ))/1000)
        userLocationY = Double(round(1000*( userLocationY ))/1000)
        return [userLocationX,userLocationY]
    }

    func calculateWeightedCentroidParameter(beacons: [CLBeacon],
currentBeaconInArray: Int, more3:Bool) -> Double {
        var count:Int = 0
        if (more3){
            count = beacons.count
        }else{
            count = 3
        }
        var currentWeightedCentroidParameter:Double = 0.0
        var returnWCP:Double = 0.0
        if (WCWCL_RSSIMode){
            currentWeightedCentroidParameter =
sqrt(pow(10,Double(beacons[currentBeaconInArray].rssi/10)))
            if (weightedCentroidParameter == 0.0){
                for j in 0 ..< count{
                    weightedCentroidParameter = weightedCentroidParameter +
sqrt(pow(10,Double(beacons[j].rssi/10)))
                }
            }
            returnWCP =
Double(currentWeightedCentroidParameter/weightedCentroidParameter) *
pow(Double(count),2 *
Double(currentWeightedCentroidParameter/weightedCentroidParameter))
        }else{
```

```
            currentWeightedCentroidParameter =
Double(1/beacons[currentBeaconInArray].accuracy)
            if (weightedCentroidParameter == 0.0){
                for j in 0 ..< count{
                    weightedCentroidParameter = weightedCentroidParameter +
Double(1/beacons[j].accuracy)
                }
            }
            returnWCP =
Double(currentWeightedCentroidParameter/weightedCentroidParameter)
        }
        return returnWCP
    }
}
```

1.4 TrilaterationViewController.swift

```
import UIKit
import CoreLocation
class TrilaterationViewController: UIViewController, CLLocationManagerDelegate {
    @IBOutlet weak var coord_lbl: UILabel!
    @IBOutlet weak var result_txtvw: UITextView!
    var settedBeacons = [settedBeacon]()
    let locationManager = CLLocationManager()
    let region = CLBeaconRegion(proximityUUID: NSUUID(UUIDString: "B9407F30-F5F8-
466E-AFF9-25556B57FE6D")!, identifier: "Estimotes")
    override func viewDidLoad() {
        super.viewDidLoad()
        locationManager.delegate = self
        if (CLLocationManager.authorizationStatus() !=
CLAuthorizationStatus.AuthorizedWhenInUse) {
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.startRangingBeaconsInRegion(region)
    }

    func locationManager(manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], inRegion region: CLBeaconRegion) {
        var knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
        var userLocation: [Double]
        knownBeacons = beacons.filter{
            for settedBeacon in settedBeacons {
                if ($0.minor == (settedBeacon.Minor)){
                    return true
                }
            }
            return false
        }
        knownBeacons.sortInPlace { (element1, element2) -> Bool in
            return element1.rssi > element2.rssi
        }
        if (knownBeacons.count > 0) {
            let coord1 = settedBeacons.filter({$0.Minor == (knownBeacons[0] as
CLBeacon).minor}).first
            let coord2 = settedBeacons.filter({$0.Minor == (knownBeacons[1] as
CLBeacon).minor}).first
            let coord3 = settedBeacons.filter({$0.Minor == (knownBeacons[2] as
CLBeacon).minor}).first
```

```
            if ((coord1 != nil) && (coord2 != nil) && (coord3 != nil)){
                userLocation = trilateration([coord1!.X,coord1!.Y], point2:
[coord2!.X,coord2!.Y], point3: [coord3!.X,coord3!.Y], r1: Double((beacons[0] as
CLBeacon).accuracy), r2: Double((beacons[1] as CLBeacon).accuracy), r3:
Double((beacons[2] as CLBeacon).accuracy));

                coord_lbl.text = "X: \(userLocation[0]) | Y: \(userLocation[1])"
            }
        }
    }

    func trilateration(point1: [Double], point2: [Double], point3: [Double],r1:
Double,r2: Double,r3: Double) -> [Double] {
        var resultPose: [Double] = [];
        let p2p1Distance = pow(pow(point2[0]-point1[0],2) + pow(point2[1]-
point1[1],2),0.5);
        let exx = (point2[0]-point1[0])/p2p1Distance;
        let exy = (point2[1]-point1[1])/p2p1Distance;
        let ix = exx*(point3[0]-point1[0]);
        let iy = exy*(point3[1]-point1[1]);
        let eyx = (point3[0]-point1[0]-ix*exx)/pow(pow(point3[0]-point1[0]-
(ix*exx),2) + pow(point3[1]-point1[1]-(iy*exy),2),0.5);
        let eyy = (point3[1]-point1[1]-iy*exy)/pow(pow(point3[0]-point1[0]-
(ix*exx),2) + pow(point3[1]-point1[1]-(iy*exy),2),0.5);
        let jx = eyx*(point3[0]-point1[0]);
        let jy = eyy*(point3[1]-point1[1]);
        let i = ix + iy
        let j = jx + jy
        let x = (pow(r1,2) - pow(r2,2) + pow(p2p1Distance,2)) / (2 * p2p1Distance);
        let y = (pow(r1,2) - pow(r3,2) + pow(i,2) + pow(j,2))/(2*j) - ix/j;
        var finalX = point1[0] + x*exx + y*eyx;
        var finalY = point1[1] + x*exy + y*eyy;
        finalX = Double(round(1000*( finalX ))/1000)
        finalY = Double(round(1000*( finalY ))/1000)
        result_txtvw.text! += "X: \(finalX) | Y: \(finalY) \n"
        resultPose.append(finalX);
        resultPose.append(finalY);
        return resultPose;
    }
}
```

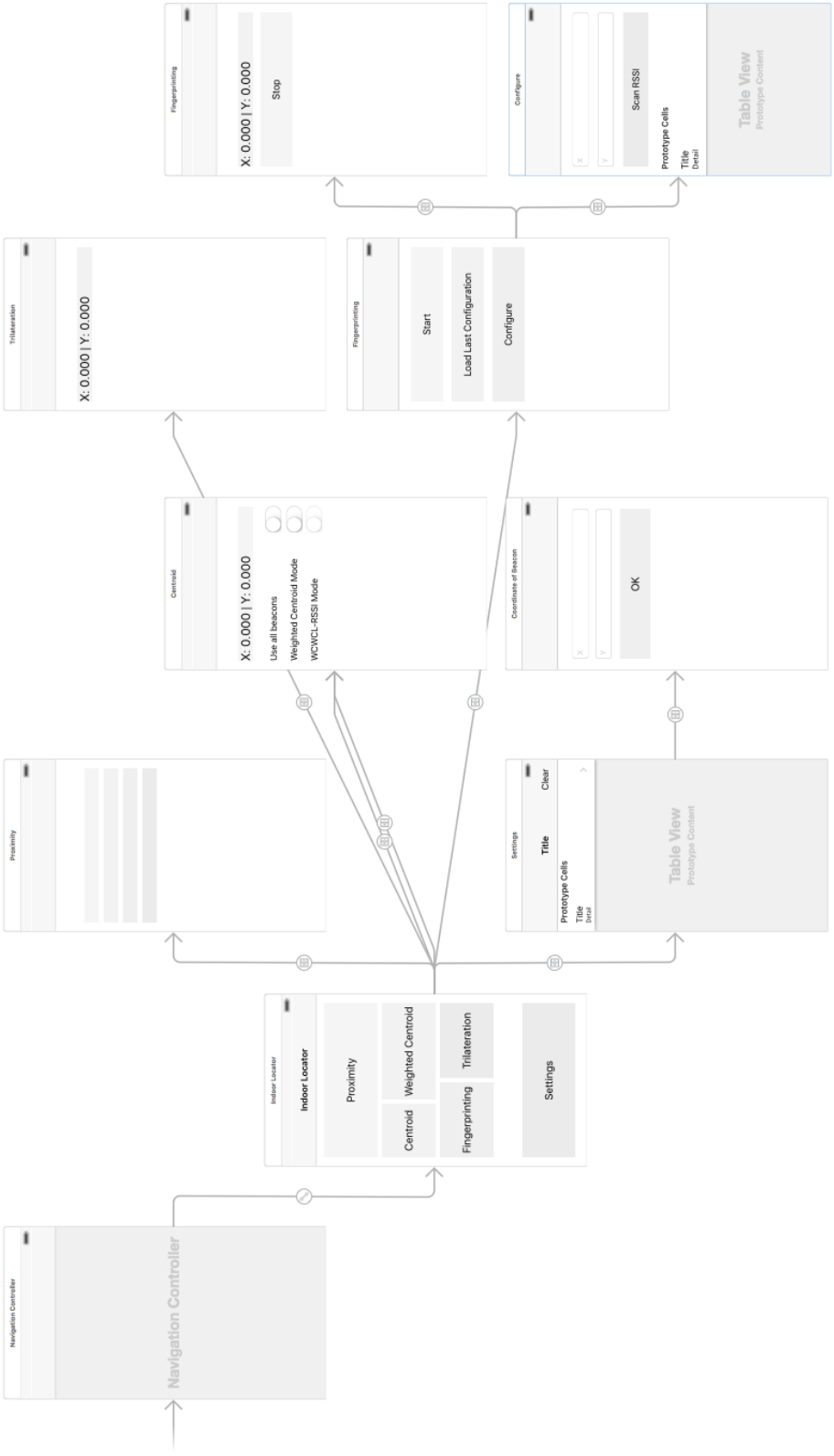
1.5 FingerprintingDefiningViewController.swift

```
import UIKit
import CoreLocation
import Foundation
class FingerprintingDefiningViewController: UIViewController,
CLLocationManagerDelegate {
    @IBOutlet weak var coord_lbl: UILabel!
    @IBOutlet weak var resultOfED_txtvw: UITextView!
    @IBAction func clickOnStop(sender: UIButton) {
        if (flagOfWorking){
            sender.setTitle("Start", forState: .Normal)
            locationManager.stopRangingBeaconsInRegion(region)
            flagOfWorking = false
        }else{
            sender.setTitle("Stop", forState: .Normal)
            locationManager.startRangingBeaconsInRegion(region)
            flagOfWorking = true
```

```
        }
    }

    var flagOfWorking:Bool = true
    var fingerPrintingPoints = [fingerPrintingPoint]()
    let locationManager = CLLocationManager()
    let region = CLBeaconRegion(proximityUUID: NSUUID(UUIDString: "B9407F30-F5F8-
466E-AFF9-25556B57FE6D")!, identifier: "Estimotes")
    override func viewDidLoad() {
        super.viewDidLoad()
        locationManager.delegate = self
        if (CLLocationManager.authorizationStatus() !=
CLAuthorizationStatus.AuthorizedWhenInUse) {
            locationManager.requestWhenInUseAuthorization()
        }
        locationManager.startRangingBeaconsInRegion(region)
    }

    func locationManager(manager: CLLocationManager, didRangeBeacons beacons:
[CLBeacon], inRegion region: CLBeaconRegion) {
        var knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
        var userLocation: [Double]
        if (fingerPrintingPoints.count > 0){
            knownBeacons.sortInPlace { (element1, element2) -> Bool in
                return element1.rssi > element2.rssi
            }
            if (knownBeacons.count > 0) {
                userLocation = FingerprintingDefine(knownBeacons)
                coord_lbl.text = "X: \(userLocation[0]) | Y: \(userLocation[1])"
            }
        }
    }

    func FingerprintingDefine(beacons: [CLBeacon]) -> [Double]{
        var min:fingerPrintingPoint = fingerPrintingPoint(X: -1.0, Y: -1.0, Beacons:
[:], EuclideanDistance: 1000000.0000, EuclideanDistance2: [], RSSI: "")
        var ED:Double = 0.0
        for j in 0 ..< fingerPrintingPoints.count{
            ED = 0.0
            fingerPrintingPoints[j].EuclideanDistance2 = []
            for i in 0 ..< beacons.count{

fingerPrintingPoints[j].EuclideanDistance2.append(abs(Double(fingerPrintingPoints[j].
Beacons[Int(beacons[i].minor)]!.averageRSSI) - Double(beacons[i].rssi)))
                ED = ED +
abs(Double(fingerPrintingPoints[j].Beacons[Int(beacons[i].minor)]!.averageRSSI) -
Double(beacons[i].rssi))
            }
            fingerPrintingPoints[j].EuclideanDistance = ED
            if (min.EuclideanDistance > fingerPrintingPoints[j].EuclideanDistance){
                min = fingerPrintingPoints[j]
            }
            var printString:String = " X: \(fingerPrintingPoints[j].X) Y:
\(fingerPrintingPoints[j].Y) - ED:\(fingerPrintingPoints[j].EuclideanDistance) \n"
            printString += "\(fingerPrintingPoints[j].EuclideanDistance2) \n"
            resultOfED_txtvw.text! = printString + resultOfED_txtvw.text!
        }
        fingerPrintingPoints.sortInPlace { (element1, element2) -> Bool in
            return element1.EuclideanDistance < element2.EuclideanDistance
```

```
        }
        var x:Double = 0.0
        var y:Double = 0.0
        if (fingerPrintingPoints.count>2){
            var w:Double = 0.0
            for j in 0 ..< 3{
                if (Double(fingerPrintingPoints[j].EuclideanDistance) ==
0.0){continue}
                w = w + 1/pow(Double(fingerPrintingPoints[j].EuclideanDistance),2)
            }
            for i in 0 ..< 3{
                x +=
((1/pow(Double(fingerPrintingPoints[i].EuclideanDistance),2))/w)*fingerPrintingPoints
[i].X
                y +=
((1/pow(Double(fingerPrintingPoints[i].EuclideanDistance),2))/w)*fingerPrintingPoints
[i].Y
            }
            x = Double(round(100*( x ))/100)
            y = Double(round(100*( y ))/100)
        }else{
            x = fingerPrintingPoints[0].X
            y = fingerPrintingPoints[0].Y
        }
        resultOfED_txtvw.text! = "----------------------------\n" +
resultOfED_txtvw.text!
        resultOfED_txtvw.text! = " X: \(min.X) Y: \(min.Y) -
ED:\(min.EuclideanDistance) \n" + resultOfED_txtvw.text!
        resultOfED_txtvw.text! = "----------------------------\n" +
resultOfED_txtvw.text!
        return [x,y]
    }
}
```

2. The graphical representation of User Interface



3. Suitable for experiments smartphones

<table>
<tr><th></th><th>Chip</th><th>RAM</th><th colspan="2">Bluetooth</th><th>Initial release iOS</th><th>Current supported iOS</th></tr>
<tr><td>iPhone 4s</td><td>Apple A5</td><td>512 MB LPDDR2 DRAM</td><td rowspan="3">4.0</td><td>Broadcom BCM4330</td><td>5.0</td><td rowspan="8">9.3.1</td></tr>
<tr><td>iPhone 5</td><td>Apple A6</td><td>1 GB LPDDR2 DRAM</td><td rowspan="2">Broadcom BCM4334</td><td>6.0</td></tr>
<tr><td>iPhone 5s</td><td>Apple A7</td><td rowspan="3">1 GB LPDDR3 DRAM</td><td>7.0</td></tr>
<tr><td>iPhone 6</td><td rowspan="2">Apple A8</td><td rowspan="4">4.2</td><td rowspan="2">Broadcom BCM4345</td><td rowspan="2">8.0</td></tr>
<tr><td>iPhone 6 Plus</td></tr>
<tr><td>iPhone 6s</td><td rowspan="3">Apple A9</td><td rowspan="3">2 GB LPDDR3 DRAM</td><td rowspan="2">Broadcom BCM4350</td><td rowspan="2">9.0</td></tr>
<tr><td>iPhone 6s Plus</td></tr>
<tr><td>iPhone SE</td><td>Broadcom BCM4345</td><td>9.3</td></tr>
</table>

4. Specification of Estimote beacon

	Name	Specification
Chip	Nordic Semiconductor nRF51822	CPU: 32-bit ARM Cortex M0 RAM: 16KB Flash Memory: 256KB Bluetooth LE: 2.4GHz
Battery	CR2450	620 mAh Medium form factor