Birhanu Hailemariam Laekemariam

RSSI-based Indoor Localization System

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Thesis
7 May 2012

Author(s) Title Number of Pages Date	Birhanu Hailemariam Laekemariam RSSI-based indoor localization system 35 pages + 2 appendices 7 May 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Information Systems
Instructor(s)	Olli Alm, Research Teacher and Project Manager Olli Hämäläinen, Senior Lecturer

This bachelor's thesis describes a research towards the system that is capable of locating a target object in indoor environment using the existing Wireless Fidelity (WiFi) infrastructures in the environment. Location aware computing service implemented in this project plays a significant role in the areas of navigation, security, market analysis, social behavior and activity studies, healthcare and entertainment. Furthermore, pragmatic predication of the upcoming movements and activities could be derived by extracting and analyzing the previously recorded movement data.

The sensor reading used for the location estimation is the Received Signal Strength Indication (RSSI) measured from the Access Points (APs). RSSI sample maps were collected from each room and common area in the building from which a database index containing an associative array was built. The associative array is composed of a collection of 'key-value' pairs; where the name of the room or common area in which the RSSI sample map was collected makes the 'key' pair and the RSSI values on the sample map make the 'value' pair. The location estimation process was carried out using similarity query techniques that is capable of finding the most similar vector index in the database index to the query vector built from the RSSI request of the target object.

The current version of the localization system is capable of locating a target object with room level accuracy with few random error results. In addition, the localization system records the target object's location-related information to a log file, from which movement and indoor activity analysis was able to be derived. Since the overall design and implementation of the localization system is intended to follow a framework-oriented approach, the final version of the product could be adapted to a new environment in accordance with the latest specifications and requirements.

Keywords	Location estimation, RSSI Similarity Match

Contents

Abbreviations and Terms

AP	Access Point
API	Application Programming Interface
Bluetooth LE	Bluetooth Low Energy
BSSID	Basic Service Set Identifier
CCU	Cell Controller Unit
dBm	Power Ratio in Decibels
GHz	Giga Hertz
GPS	Global Positioning System
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineering
IPCS	Indoor Positioning and Communication system
ISM	Industrial Scientific and Medical
ISO/IEC	International Organization for Standardization and the International Electro-technical Commission
JSON	JavaScript Object Notation
Kbps	Kilo Bits Per Second
KML	Keyhole Markup Language
m/s	Meters per Second
MVC	Model View Controller
OFDM	Orthogonal Frequency-Division Multiplexing
PDA	Personal Digital Assistant
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SDK	Standard Development Kit
SVG	Scalable Vector Graphics
UI	User Interface
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

1 Introduction

Location and context-aware technologies play a critical role in identifying and studying user behavior and interest on the basis of the user's activity and interactive relationships with other individuals or objects within a frame of time. Since, most ordinary individuals spend their time and many of their daily life activities are taking place in indoor environments, indoor localization systems are the ultimate technology chosen to solve the problem of locating and identifying the indoor objects in indoor environments. However, the ability to accurately localize and track an indoor object using existing GPS technology fails to a limited performance, due to the absence of a direct view to three or more satellites at the same time and signal disturbance caused by the metallic materials used to fasten the structure of the building.

The indoor localization system developed in this project utilizes wireless sensor nodes such as access points (APs) or any other sensor modalities that are capable of providing received signal strength indication (RSSI), which is used to determine the location of a target object or visitor in indoor environment. A client equipped with a smartphone device scans the latest RSSI values from surrounding active APs, and sends an HTTP Post request containing the RSSI values to the server. Accordingly, the algorithmic software on the server, receives the RSSI request, generates the estimated location, records the location and profile related information to a log file, and finally returns the estimated location result to the requesting target object. In addition, movement and indoor activity analysis was conducted using the data on the log file, which can contribute great share in studying individual behavior and interest, social relations and activities, marketing, business planning and other research areas.

2 Localization Technologies and Techniques

2.1 Wireless Network Standards

Wireless local area networks (WLANs) allow electronic devices to connect with each other using radio waves over the network. The technology possesses a broad range of communications and security protocols that will probably continue emerging over the next few years as technology advances. With each of the new WLAN communication standard iteration, performance and functionality have been enhanced, which has encouraged the evolvement of the standard into a real alternative to the wired Ethernet for primary corporate access. Consequently, different standards for different purposes with different performance ability and overall operating specifications have been introduced to the communication world over the last years.

2.1.1 DASH7

DASH7 is a wireless networking sensor technology that follows the global standard of ISO/IEC 18000-7 and utilizes the 433.92 MHz frequency which is license-free and accessible world widely. The technology offers a strong signal level that is capable of penetrating through concrete walls and liquid materials, but also, has the ability to transmit and receive signal data over a long range, which is scalable up to 0.25 - 2 km. In addition, good signal bending properties or fewer blind spots, optimal location granularity (approx. 1 meter) and low power consumption are positive peculiarities of the protocol. Furthermore, extendable features such as multi-hop, encryption, sensors and IPv6 make it efficient and secured. [1.]

An Apache licensed open source firmware called Open Tag provides developers a C-based API to develop internal (embedded) DASH7 applications, primarily microcontrollers. In order to offer an interfacing layer with embedded HW components such as microcontrollers, badge tags and all kinds of smart sensors, Open Tag firmware includes Python/C client tool in its implementation. [2.]

2.1.2 WLAN

A Wireless LAN networking protocol connects two or more devices using wireless distribution methods, typically spread-spectrum or Orthogonal Frequency-Division Multiplexing (OFDM) radio, and usually provides a connection through an access point to the wider internet. The protocol is based on a cellular architecture in which the system is subdivided into cells called Base Service Set (BSS), in which each cell is controlled by a base station called access point (AP).

There are several wireless LAN solutions currently available in the field of communications technology, with varying levels of standardization and interoperability. Two of the standards that are currently leading the industry are HomeRF and (IEEE** 802.11b). Of these two, IEEE 802.11 technology has wider industry support and it is targeted to solve enterprise, home and even public 'hot spot' wireless LAN needs. [3, 2.]

IEEE 802.11b or WiFi, which is its marketing name, is one of the supplements of the initial 802.11 standard and operates at 2.4 GHz bandwidth, but also includes 5.4 and 11 Mbs data rates in addition to the initial 1 and 2 Mbs. It is used in a point-to-multipoint configuration, in which an access point communicates via an omnidirectional antenna with one or more nomadic or mobile clients that are located in a coverage area around the access point. A typical wireless access point using 802.11b or 802.11g supplements with a stock antenna might have a range of 30 meters indoors at a data rate of 11 Mbs and 90 m outdoors at a data rate of 1 Mbs. However, outdoor ranges can be improved to many kilometers through the use of high gain directional antennas at the router and remote devices. [3, 2.]

Home RF is an open industry specification that defines how electronic devices such as PCs, cordless phones and other peripherals share and communicate voice, data and streaming media in and around the home. HomeRF-compliant products operate in 2.4 GHz frequency band and utilize frequency-hopping spread spectrum RF technology for secure and robust wireless communications with data rates up to 1 Mbps. Unlike WiFi, HomeRF has quality of service support for streaming media and is the only wireless LAN to integrate voice. [3, 4.]

2.1.3 Bluetooth Low Energy

Bluetooth is a wireless technology standard for exchanging data over a short distance between Bluetooth enabled electronic devices using short-wavelength radio transmissions in the ISM band in 2.4 GHz. The range may vary depending on the class of the radios used in the implementation; class 2 radios, the most commonly found in mobile devices have a range of 10 meters; class 1 radios, which are used in industrial use cases have a range of 100 meters; and class 3 radios have a range of up to 1 meter. Bluetooth technology is designed to have very low power consumption. For instance, class 2 radios use 2.5 milliwatts of power. [4.]

Bluetooth Low Energy (Bluetooth LE) wireless technology is the latest feature of v4.0 of the Bluetooth Core Specification, enhanced to enable new functionalities and applications for Bluetooth smart devices. The technology operates in the same spectrum range of 2.4 GHz as Classic Bluetooth technology but uses a different set of channels, by having 40 channels on 2 MHz wide-ranging instead of Bluetooth classics' 79 channels on 1 MHz. The operation range for Bluetooth LE varies between 10 to 100 meters distances and it has 1 Mbps data rate. As a result, the technology is not optimized for transferring files over a node, except for sending small chunks of data (exposed state). [5.]

Table 1 illustrates the technical specifications of Classic Bluetooth technology (Bluetooth 3.0) and Bluetooth LE technology (Bluetooth 4.0). Major specifications, which are significant for making a choice of one of the standards over the other such as radio frequency, coverage range and data rate are discussed.

Table 1. High level comparison between Classic Bluetooth technology and Bluetooth Low Energy technical specification. Table adopted from Classic Bluetooth vs. Bluetooth Low Energy (2011) [6].

Technical Specification	**Classic Bluetooth technology**	**Bluetooth Low Energy technology**
Radio frequency	2.4GHz	2.4GHz
Distance/Range	~10-100 m	~10-100 m
Data rate	1-3 Mbps	1 Mbps
Application throughput	0.7-2.1 Mbps	0.26 Mpbs
Latency (from a non-connected state)	100+ ms	<6 ms
Security	56 to 128 bit	128-bit AES
Network topology	Scatternet	Star-bus
Power consumption	1 as the reference	0.01 to 0.5 (depending on use case)
Peak current consumption	< 30 mA	< 20 mA
Primary use case	Mobile phones, headsets, stereo audio automotive, PCs, etc.	Mobile phones, gaming, sport & fitness, medical, industrial, home electronics, PCs, etc.

As the specifications of classic and low energy Bluetooth devices are presented in table 1, Classic Bluetooth technology is a relatively high power standard, which provides a high data transmission rate and can carry data over longer distances. On the contrary, the Bluetooth LE technology consumes considerably less energy and has a lower transmission rate and carries data over very short distances. It is apparent that the later version could be used to log continuous data from different monitoring devices to a remote server.

Bluetooth LE technology is designed with two equally important implementation alternatives: single-mode chips and dual-mode chips. Small power-sensitive devices such as tokens, watches and sports sensors based on single-mode BLE implementation enjoy the low-power consumption advantages enabled for highly integrated and compact devices such as ultra-low power idle mode operation, simple device discovery, and reliable point-to-multipoint data transfer with advance power-save and encryption functionality. In the case of dual-mode implementation, Bluetooth LE functionality is integrated into Classic Bluetooth circuitry. Subsequently, Bluetooth LE's architecture shares Classic Bluetooth technology radio and antenna, enhancing currently chips with the new low energy stack, and enhancing the development of Classic Bluetooth devices with new capabilities. [7.]

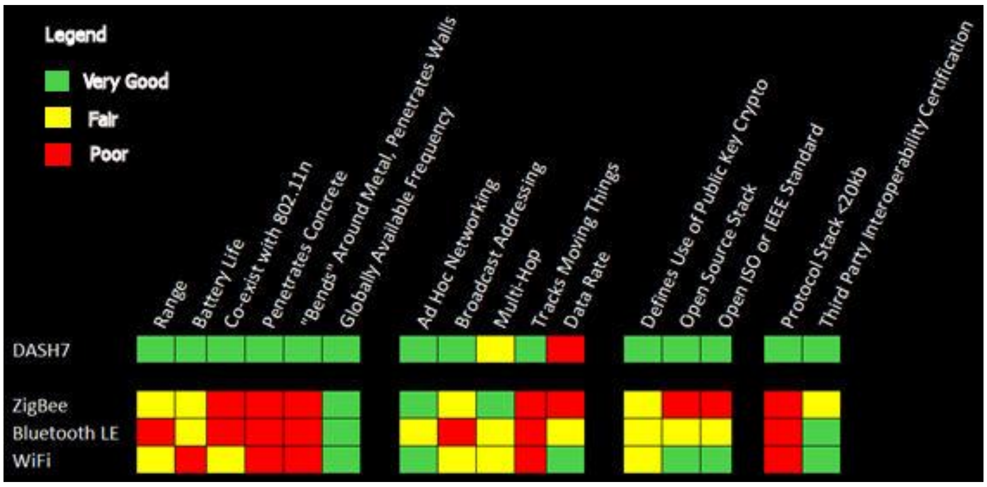
2.1.4 ZigBee

ZigBee specification is a combination of HomeRF Lite and the 802.15.4 specification. The specification operates in the 2.4GHz(ISM) radio band, the same as the 802.11b standard, Bluetooth, Microwaves and some other devices. It is capable of connecting 255 devices per network. The specification supports data transmission rates of up to 250 Kbps at a range of up to 30 meters. ZigBee's technology is slower than that of 802.11b (11 Mbps) and Bluetooth (1 Mps) but it consumes significantly less power. [8.]

The technology builds upon the physical layer and medium access control defined in IEEE standard 802.15.4 (2003 version) for low-rate WPANs. The specification goes on to complete the standard by adding four main components: network layer, application layer, ZigBee device objects (ZDOs) and manufacturer-defined application objects which allow for customization and favor total integration. [8.]

ZigBee looks rather like Bluetooth but is simpler, has a lower data rate and spends most of its time snoozing. This characteristics means that a node on a ZigBee network should be able to run for six months to two years on just two AA batteries. The operational range of ZigBee is 10-75 meters compared to 10-100 meters for Bluetooth (without a power amplifier). ZigBee sits below Bluetooth in terms of data rate. The data rate of ZigBee is 250 kbps at 2.4GHz, 40 kbps at 915 MHz and 20 Kbps at 868 MHz whereas that of Bluetooth is 1 Mbps. When ZigBee node is powered down, it can wake up and get a packet in around 15 milliseconds whereas a Bluetooth device would take around three seconds to wake up and respond. [9, 3.]

Figure 1 illustrates feature comparisons of the four wireless sensor protocols called DASH7, ZigBee, Bluetooth LE and WiFi on the bases of individual technical capability and relative ease of usability. The names of the protocols are aligned in vertical order with their respective feature characteristics aligned horizontally. Three colors, green, yellow and red, tell the level of optimality from very good to poor consequently for each protocol.



Legend

🟩 Very Good

🟨 Fair

🟥 Poor

	Range	Battery Life	Co-exist with 802.11n	Penetrates Concrete	"Bends" Around Metal, Penetrates Walls	Globally Available Frequency	Ad Hoc Networking	Broadcast Addressing	Multi-Hop	Tracks Moving Things	Data Rate	Defines Use of Public Key Crypto	Open Source Stack	Open ISO or IEEE Standard	Protocol Stack <20kb	Third Party Interoperability Certification
DASH7	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟨	🟩	🟥	🟩	🟩	🟩	🟩	🟩
ZigBee	🟨	🟨	🟥	🟥	🟥	🟩	🟩	🟨	🟩	🟥	🟥	🟨	🟥	🟥	🟥	🟨
Bluetooth LE	🟥	🟨	🟥	🟥	🟥	🟩	🟨	🟥	🟨	🟥	🟨	🟨	🟨	🟨	🟥	🟩
WiFi	🟨	🟥	🟨	🟥	🟥	🟩	🟩	🟨	🟨	🟥	🟩	🟨	🟩	🟩	🟥	🟩

Figure 1. DASH7 feature comparison with other similar technologies. Picture modified from what is DASH7 technology (2009) [1].

As shown in figure 1, DASH7 wins in most of the specified characteristics, having a long range of communication coverage, low power consumption and better tracking abilities of moving things, except for the Data Rate category, as a higher frequency level is needed to deliver large bandwidths or higher data-rates. Nevertheless, WiFi technology has a better data rate capability among the other three technologies, since it is optimized for the purpose of data exchanging between electronic devices in a computer network rather than for lactating and tracking a moving object in indoor environment. ZigBee and Bluetooth technologies have a resemblance specification in most of the listed features which the other two technologies do not. This indicates that both technologies can be the second alternative choices next to the DASH7 technology, for developing a system that is expected to locate and track a moving object.

2.2 Location Estimation Approaches

2.2.1 Triangulation Estimation

Triangulation estimation is a trigonometric approach of determining an unknown location based on two angles and a distance between them. In sensor networks, two reference nodes are required to be located on a horizontal baseline for *X-axis* and two sensor nodes are located on a vertical baseline for *Y-axis*. The distance d, between the two references nodes on the baseline can be measured in a preliminary stage and stored in memory. The two angles α_1 and α_2 are measured between the baseline and the line formed by the reference node and target node as shown in figure 2. [10, 234.]

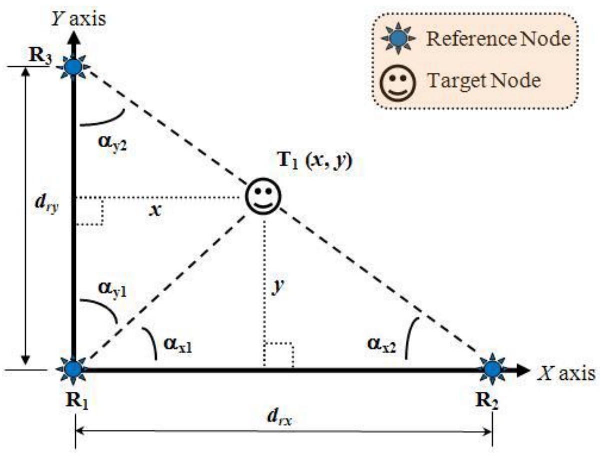


Figure 2. Triangulation estimation of a target node, using three nodes. Picture modified from Emerging communications for wireless sensor networks (2011) [10, 235].

As shown in figure 2, reference nodes R_1 and R_2 form the baseline of *X-axis*. Reference node R_1 can be reused to form the baseline of Y-axis together with reference node R_3. A target node T_1 moves freely around the area. Based on basic triangulation, the location coordinate (*x, y*) of T_1 can be determined by using the combination of R_1 and R_3 to find *x*, and the combination of R_1 and R_2 to find *y* [10, 235].

$$x = \frac{d_{ry}\sin(\alpha_{y1})\sin(\alpha_{y2})}{\sin(\alpha_{y1} + \alpha_{y2})}$$

$$y = \frac{d_{rx}\sin(\alpha_{x1})\sin(\alpha_{x2})}{\sin(\alpha_{x1} + \alpha_{x2})}$$

GPS satellites rotate around the earth twice a day in a very precise orbit and transmit signal information to earth. A user with a GPS receiver takes this information and use triangulation to calculate the exact location of the user. Essentially, the GPS receiver compares the time a signal was transmitted by a satellite with the time it was received. The time difference tells the GPS receiver how far away the satellite is. As shown in figure 3, the GPS receiver must be locked onto the signal of at least three satellites to calculate a 2D position (latitude and longitude) and track movement. With four or more satellites in view, the receiver can determine the user's 3D position (latitude, longitude and altitude). Once the user's position has been determined, the GPS unit calculates other information, such as speed, bearing, track, trip distance to destination, sunrise and sunset time and more [11].

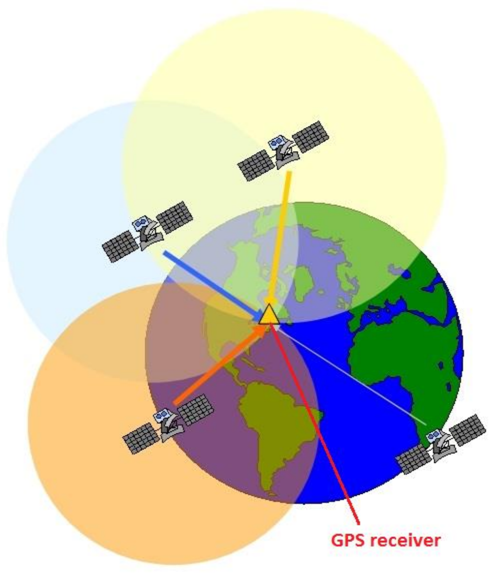


Figure 3. Intersection point of four satellites determining GPS receiver's location on the earth surface. Picture modified from GPS – How it works (2012) [12].

Figure 3 shows, the intersection point of four GPS satellites locating the exact position of a GPS receiver on the earth surface. The Triangulation method, which determines the location of an object by identifying the intersection point of three or more satellites by measuring the angles on the point from a known reference point at either end of a fixed baseline, is applied in the localization process to locate the estimated position of the GPS receivers marked by a yellow triangle on the picture.

The role of the satellites is to continuously report the time very accurately. Time information travels in the form of a radio wave down from the satellite to the GPS receiver. Because of this fact it is possible to know the time required for the radio wave to travel from the satellite to earth. Radio waves travel at a constant speed of light (~ 2.99 * 10^8 m/s): the distance that the wave has to travel can easily be calculated by using the formula $\text{speed} = \frac{\text{distance}}{\text{time}}$. When one distance is known, the GPS receiver must be located on the surface of a sphere with the satellite at the center and with a radius equal to this distance. With two distances, the location must be on a circle represents the intersection between the two spheres. With three distances known, two points are possible of which one will be far out in space thus able to be eliminated, the other one will be on earth's surface. Distance from four or more GPS satellites will intersect at just one point. [13.]

2.2.2 RSSI Similarity Matching Method

RSSI-based location estimation method is evaluated using the Received Signal Strength Indicator (RSSI) from the sensor nodes installed in the indoor environment. Regarding network infrastructure implementations, different wireless network sensors such as DASH7, ZigBee, Bluetooth or WLAN can be configured to operate as a transmitter node and a connection point between hardware components installed in the environment. A set of receivers that includes Radio Frequency Identification Device (RFID) tags, Bluetooth smart devices and special Indoor Positioning and Communication System (IPCS) are deployed throughout the coverage area to measures the RSSI values from the sensor nodes. The RSSI values from the receivers are sent to the server, which hosts the algorithmic software that is capable of generating the target object's estimated location using the RSSI values.

The RSSI-based indoor localization system applied in this project uses RSSI similarity matching approach by collecting an RSSI sample maps, building a database index and finding a matching RSSI from the database index. A human operator performs a site survey by measuring the RSSI values manually from each room and common area in the indoor environment. These RSSI values are recorded to a file by giving the name of the room or common area as associative file-name, and sorted in a folder that makes up the RSSI sample map collection. Then, a database index is built for the purpose of containing the vector index objects that represent the RSSI sample maps, and improving the speed of data retrieval operation while operating the location estimation process.

Location estimation process is done by performing similarity matching operation between the vector indexes in the database index and the query vector built from the RSSI request of the target object. In other words, the reference name of the vector index having the most similar element values to element values of the query vector will be selected as the name of the location where the target object is estimated to be.

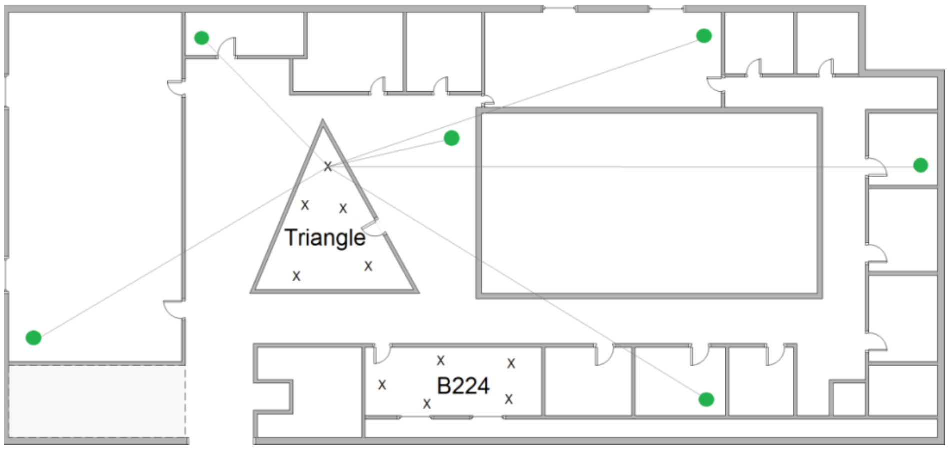


Figure 4. Example of sample collection points inside a room.

Figure 4 illustrates five sample points from where the RSSI sample maps were collected in a room called 'Triangle'. The collected RSSI sample maps contain the RSSI values measured from the signal level of the surrounding APs that are visible to the Triangle room. These RSSI sample maps represent the specific location (i.e room Triangle) in which the collection of the RSSI sample maps took place. The collection of the RSSI sample maps forms a 'key-value' pair data representation, so that the RSSI values on each sample map are associated with the name of the RSSI sample maps (i.e. Triangle-1, Triangle-2, … ,Triangle-5).

The sample maps representing the rooms from which they were collected are given an association name that corresponds with the name of the room (e.g. 'B214') or common area (i.e. 'Corridor B205'). This associated room or common area name is defined as a 'key' pair to the RSSI values on the sample map which are the 'value' pair in a 'key-value' pair oriented data representation. Thus, when a request containing the surrounding RSSI values is sent by the target object or visitor, the name of the RSSI sample map with the most matching RSSI values will be obtained as the name the estimated location.

Apart from being technically less complex and hence less expensive than using the uniquely installed signal nods such as the DASH7 transceiver and Bluetooth radio wave transmitters, the other advantage of using the RSSI method is the ability to use existing wireless infrastructure (e.g. WLAN APs as single nodes). However, the accuracy level of the RSSI levels has been put in jeopardy due to the noisy nature of

AP's signals caused by the multipath contributions having occurred in indoor scenarios, such as signal absorber or reflector building materials, humidity level and people blockage. In addition, most of the existing wireless devices (e.g. WLAN access points, ZigBee nodes) are not designed for the purpose of measuring accurate RSSIs.

2.2.3 Aligning Local Positions with Global Coordinate Systems

Associating the local positioning system with GPS or other global positioning systems is useful for many reasons, as a system with combined features have diversified functionalities and operates more optimally. A local positioning system is needed to provide location information within the coverage area in which the GPS signal level does not reach or where the signal is weak to be tracked, such as urban canyons or indoor places. On the other hand, local positioning systems are needed to be integrated with GPS coordinate systems in order to give a full tracking coverage and consistent route record of the target object to be tracked.

By aligning the floor plan image of a building on geospatial software that support a Keyhole Markup Language (KML) notation such as Google earth and Google Maps at the exact surface area where the actual building lays, geographical data such as geo coordinates, geologic tilting and heading altitude can be generated from any point on the surface area. By doing so, a building and its rooms which are dot spot relative to the global map scale can be expressed in terms of a coordinate system.

A practical example is given in this paragraph. While a user equipped with a positioning system that has a combined implementation of both the global positioning system and local positioning system navigates outdoors, the integrated positioning system can shift its service to the local positioning service automatically at the moment when a building or an indoor environment is found that is aligned on the surrounding geographic coordinates. Thus, the system starts to offer an indoor positioning service by pin-pointing and tracking the movement route of the user inside the building.

Alignment of local positions with global coordinate systems plays a significant role also in the areas of persisting series (consistent) trace routes of the target object. For example, when the target object or visitor hangs around in one of the building units inside a compound premise and changes the location to other buildings within the same compound, the integrated system can obtain a consistent total record of the target object's trace routes and indoor activities by switching the integrated system's 'local' and 'global positioning' services back and forth.

Additionally, keeping a consistent movement track of an indoor object that moves in both indoor and outdoor environments plays a significant role in the area of security related concerns. A typical example would be, the location of a stole piece of art with a patched RFID tag can be tracked and located both inside the museum as well as outside the museum, in the case of the local security systems fail to alert right on the spot.

3 RSSI-based Localization System Description

3.1 Requirements

3.1.1 Functional Requirements

Any smartphone device, Bluetooth device, RFID tag or special IPCS tag that is capable of sending signal information captured from the surrounding area to the server can be linked to the target object or visitor that is going to be tracked. In the case of the smartphone device, the device first scans lists of available Basic Service Identifiers (BSSIDs), Secured Set Identifiers (SSIDs) and their signal levels in decibels (dBm), and sends these data to the server over a network. In respect to the RFID tag, signal transceivers transmit a signal to the tag and read the data from the integrated circuit (IC) of the tag through their antenna; these transceivers also forward the data to the system server running RFID software or RFID middleware. In the IPCS tag situation, IPCS nodes which also act as locating beckons form a network connection to the server trough the Cell Controller Unit (CCU) in order to transfer data between the IPCS tags and the system server.

The server application is responsible for handling requests and responses between the server and the visitors equipped with one of the tags mentioned in the previous paragraph or any other electronic devices such as smart phone and Bluetooth smart watch. The Controller component of this server application accepts clients' request containing location related information as input stream, parses the information, and forward to the Business Logic component of the application. The Business logic component comprehends an algorithmic implementation that processes location related data sent from the visitor's device, and generates the estimated location of the visitor in question (i.e. room name or common area name).

The Controller component of the server application then retrieves the result generated by the Business Logic (i.e. room name or common area name), adds the visitor's tag ID (device name) and timestamp from the memory, and finally appends all the data to a log file. The data from the log file will later be used for analytical procedures related with movement and activity concerns. If the visitor itself is also interested in monitoring real-time movement route or location of its own, the Controller component forwards the location information to the visitor. A smart phone client application with a graphical User Interface (UI) can be implemented and deployed on the visitor's device in order to show real-time location, movement trace route and other location-related information.

The log analyzer server application is responsible for consuming the log data produced by the other server application mentioned in the previous paragraph and makes movement analysis of the target object or visitor by manipulating the data on the log file. This server application is also deployable on the server and has a time-based job scheduler configuration file (e.g. cron job scheduler in Unix-like Operating System) in order to execute its functions at a specific time interval.

3.1.2 Non-Functional Requirements

A request from the client to the server is adjusted to be in a given time interval of few seconds, which minimizes the power loss occurring while sending and receiving data back and forth at a high frequency rate. Devices such as Bluetooth LE, Radio-Frequency Identification (RFID) and a special IPCS tag fulfills the need of the power efficiency requirement, since their energy consumption is drastically lower than that of other similar devices, such as smart phones, PADs and tablets.

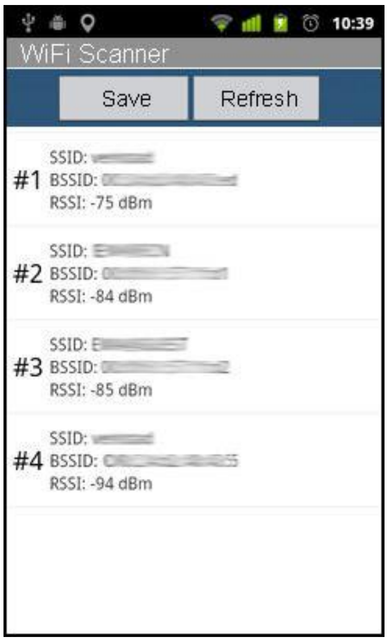
The server application is expected to handle constant requests from several clients frequently at instant time, and also instantly appends resultant data (e.g. estimated location) to a file system. Therefore, the response time of the server needs to be optimized within milliseconds range. Another point where the applicability of rapid server response time would be significantly important is when a visitor itself or another third party needs to monitor the real-time movement or indoor activities on a client UI. In this case, the server has to respond the resultant data (i.e. the estimated location) instantaneously to the client in order to visualize real-time movement and indoor activity of the visitor synchronically on the client UI.

As mentioned in the previous section, instant location estimation responses, a visitor's tag ID (device name) and timestamp will be appended to a log file. Thus, following this kind of approach makes monitoring the server applications’ performance very easy, as errors from the applications and the hosting server itself can be observed immediately from the log file. In addition, the methodology from the previous sentence makes the system fault tolerant by avoiding data loss risk which can occur while system crash or service inconvenience happens.

3.2 RSSI Sample Mapping

An Android application implemented on top of a WiFiManager Application Programming Interface (API) provided by the Android Standard Development Kit (SDK), scans lists of active APs, and retrieves their BSSID (address of the AP), SSIDs (configured network names) and levels (detected signal strengths in dBm). These scanning results are saved to a file by giving similar file name to the name of the room or common area, where the scanning takes place. This technique imposes a 'key-value' pair data representation by forming an association between the scanned results (i.e. RSSI values) and the room or common area, where the RSSI values are collected. The formation of the association between the RSSI values and the room or common area, where these RSSI values are collected allows retrieving the name of the room or common area (corresponds to the name of the file) be passing the matching RSSI values as a 'key' pair.

Following the same scanning procedure, manual measurement of the RSSI values from each room and common area such as the lobby and corridors was carried out in order to build the RSSI sample maps collection containing RSSI values measured from each room and common area. The sample maps collection is then stored in a packaged folder and retrieved by the Positioning Service (explained in chapter 3.3.1) in order to create a database index that improves the speed of data retrieval operation. Indexing the data model gives the Positioning Service an efficient performance hit while performing similarity matching operation between the query vector built from the RSSI request of the visitor and the vector indexes in the database index built from the RSSI sample maps.


WiFi Scanner

Save Refresh

#1 SSID: [redacted]
BSSID: [redacted]
RSSI: -75 dBm

#2 SSID: [redacted]
BSSID: [redacted]
RSSI: -84 dBm

#3 SSID: [redacted]
BSSID: [redacted]
RSSI: -85 dBm

#4 SSID: [redacted]
BSSID: [redacted]
RSSI: -94 dBm

Figure 5. List of reachable access points, stored by signal strength.

As shown in figure 5, the WiFiScanner Android application scans the surrounding WiFi network and displays, the SSID, BSSID and RSSI of each visible AP. This signal information is saved to the file by giving an association file name which is similar with the specific name of the surrounding area, where the scanning take place (e.g. B224) or common area name (e.g. corridor, lobby and terrace).

3.3 System Architecture of the Localization System

Logical architecture of the system defines exchanges between system components and system functional processes. The Application server contains two different server applications called the Positioning Service and Movement Analyzer Service that are designed and developed to give two different services for both the system users and system administrators. The movement data analyzed by the Movement Analyzer Service is stored in a document-based database called CouchDB. The mobile device client application, which is part of the remote implementation of the system, communicates with other components on the server over a network using the HTTP request/response protocol.

Both the Positioning and the Movement Analyzer Services utilize a log file for appending the resultant data and consuming the recorded data. The Positioning Service appends the data to the log file containing profile and location-related information of the target object or visitor which includes device name or tag ID, room or common area name and the response timestamp. The Movement Analyzer Service consumes the log file to derive movement analysis results of the target object or indoor visitor, establishes a connection with the database and persist the analyzed results to database.

The system level software architecture shown in figure 5 illustrates the structural design and core components of the system developed. It also shows the procedural request/response process flow between the remote client, the server and the storage database. In addition, it depicts the main functionalities undergone in both the Positioning and Movement Analyzer Services and the responsible objects expected to handle the defined task.

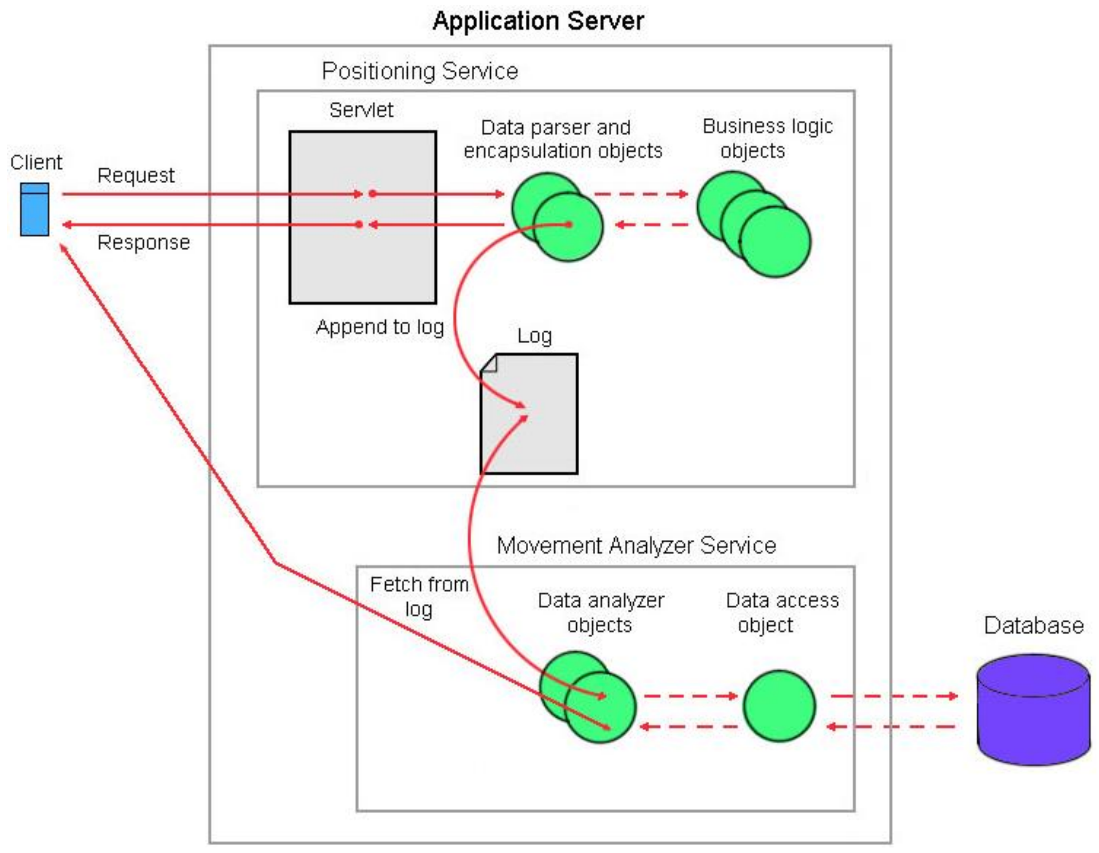


Figure 5. System level architecture of the localization system.

As it can be observed in figure 5, the logical architecture of the system is structured in three main parts: the client, the server and the database. The client section comprises a mobile device application that scans the surrounding available signal levels and sends them to the server including the device ID over a network. The Application Server contains two applications: the Positioning Service and Movement Analyzer Service. The Positioning Service is responsible for laying out a communication interface to the client, accepting client requests, performing location aware algorithmic operation, appending information to a log file and finally returning the estimated location response back to the client. The Movement Analyzer Service, on the other hand, is subjected to consume the log data, extract relevant content, draw route or movement analysis and finally record the analyzed data to the database.

3.3.1 Positioning Service

The Positioning Service is a Java-based web application, which comprehends a 'Similarity Matching' algorithmic implementation and which was deployed on the Apache Tomcat web container. The main objective of this service is to generate real-time location estimation of the target object (indoor visitor) by performing similarity matching techniques between the query vector built from the signal level dBm values of the visitor's RSSI request and the vector indexes built from the RSSI sample maps in the database index.

To paragraph briefly discusses how the vector matching operation works. First, when a request from the target object containing RSSI values arrives, the Positioning Service builds a vector object representation for the RSSI request, which means dBm signal level values on the RSSI request will be assigned to be the element values of the built vector object. Second, the Positioning Service searches for the matching vector object from the database index that has the most similar element values as the vector object built from the target object's RSSI request. Subsequently, the servlet component of the Positioning Service responds the reference name of the resultant vector object which corresponds to the room or common area name of the requesting object as an estimated location name.

As mentioned in a book called Similarity Search, similarity queries are defined explicitly or implicitly by a query object q and a constraint on the form of and extent of proximity required, typically expressed as a distance. The response to a query returns all objects which satisfy the selection conditions, presumed to be those objects close to the given query objects. [14, 15.]

There are different type similarity query schemes: the Range Query, Nearest Neighbor Query, Reverse Nearest Neighbor Query, Similarity Join, Combinations of Queries and Complex Similarity Queries. The *similarity range query R(q, r)* is probably the most common type of similarity query*.* The query is specified by a query object $q \in D$, with a query radius r as the distance constraint. The query retrieves all objects found within distance r of q, formally. [14, 15-19.]:

$$R(q,r) = \{o \in X, d(o,q) \leq r\}$$

Individual objects in the response can be ranked according to their distance with respect to q. The query object q need not exist in the collection $X \subseteq D$ to be searched, and the only restriction on q is that it belongs to the metric domain *D*. When the search radius is zero, the range query *R(q, 0)* is called a *point query* or *exact match*.
In this case, an identical copy (or copies) of the query object q is the thing what is being looked for. [14, 15].

3.3.2 Android Client Application

The Android client application corresponds with the View section in the MVC structure of the system and it is meant to be installed on the target object (indoor visitor) smart phone device. When the 'Start Locating' button is pressed, it starts to periodically scan visible RSSIs from surrounding active WiFi networks or any other signals from signal transmitter nodes, and sends an RSSI request containing the scanned signal results to the server over a network. On the flip side of the story, the Positioning Service server application explained in the previous section receives the RSSI request, performs similarity matching operation to generate estimated location of the target object or visitor, and forwards the estimated location back to the target object. The client application then receives the response from the server, associates this response contextually with the graphical design component of the client application, and visualizes the real-time location and movement trace of the requesting visitor on the UI.

3.3.3 Movement Analyzer Service

The Movement Analyze Service comprises similar functionality as any 'Log Analyzer' software does, that is consuming and parsing the log data produced by other service applications. Similarly, the Movement Analyzer Service consumes the log file generated by the Positioning Service server application, and attempts to determine the movement analysis and indoor activity of the visitor. The first point of interest in implementing this service was subjected to draw the movement analysis of the target object or a visitor, which includes the time spent or residing duration in a specific area, average time spent per room (common area), total number of visited rooms (common areas), starting time of the visit, ending time of the visit and the total time spent for the entire visit.

4 Implementation of Server Services and Android client Application

4.1 Positioning Service Implementation

Structural design of the Positioning Service server application follows the Model View Controller (MVC) design pattern, which splits intersections between users and application into three layers: the Model (business logic), the View (presentation) and the Controller (persistence), according to the tasks and types of resources shared by each component in the application. This separation of concerns facilitated the independent development, testing and maintenance of each role, and also made the overall development process significantly smoother.

The *UpdateLocationServlet* class, which is designed to play a Controller role in the MVC structure, is implemented on the top of the Servlet API of the J2EE specification, and it is responsible for handling data exchanges between the client (i.e the target object's or visitor's) device and responsible objects in the Business Logic layer. A request from the target object's device is sent in the form of a serialized *OutputStream* object over a network using the HTTP request/response protocol. The *doPost* method implemented in the *UpdateLocationServlet* class receives the incoming request in the form of an *InputStream* object, and passes it to the *streamDataParser* method that parses and converts the data into a *String* type object. Another method in the *UpdateLocationServlet* class called *dataHandler*, takes the *String* value and extracts the content into profile-related and location-related sub category. Finally, the *dataHandler* method forwards the profile-related data to the object of *VisitorEncapsulation* type and the location-related to object of *SignalDataParser* type respectively.

The business logic layer is designed to comprehend the *WLANSimilarityMatch* class, which is implemented on top of the OBSearch similarity index search API. In order to give a summarized description of how the vector similarity matching operation performed. First, it is necessary to start with how the query vectors and vector index objects are created, and what the things are that make up the element values of both vector objects. As shown in appendix 1, when an instance of the *WLANSimilarityMatch* class is initialized, it creates a database index containing the vector indexes built from the RSSI sample maps by assigning signal level dBm values recorded on the RSSI sample maps to the element value of the vector indexes. Unique BSSIDs that are available within the indoor environment but not visible to room or common area where the RSSI sample maps are collected from, are given a zero (0) element value while building the vector object representation of the RSSI sample maps. This procedure also helps to indicate that the not-found BSSIDs were not visible in the room or common area while collecting the RSSI sample maps. The name of the RSSI sample map is given to the vector object as a reference or 'key' pair name to imply that this vector represents the RSSI values found in the specific locations where the RSSI sample map was collected.

The query vector object is also created in the same procedure as the vector indexes explained earlier, except that this time the signal level dBm values which are assigned to the vector elements' value are obtained from the real-time RSSI request sent by the target object or visitor as *InputStream* object.

The sequence diagram shown in figure 6 shows components of the system involved in the location estimation process and the sequential message exchanges undertaken between system components in a sequential time order. By taking a system-level approach, only the top most essentials and fundamental parts of the whole system are taken into account to be included in the sequence diagram design.

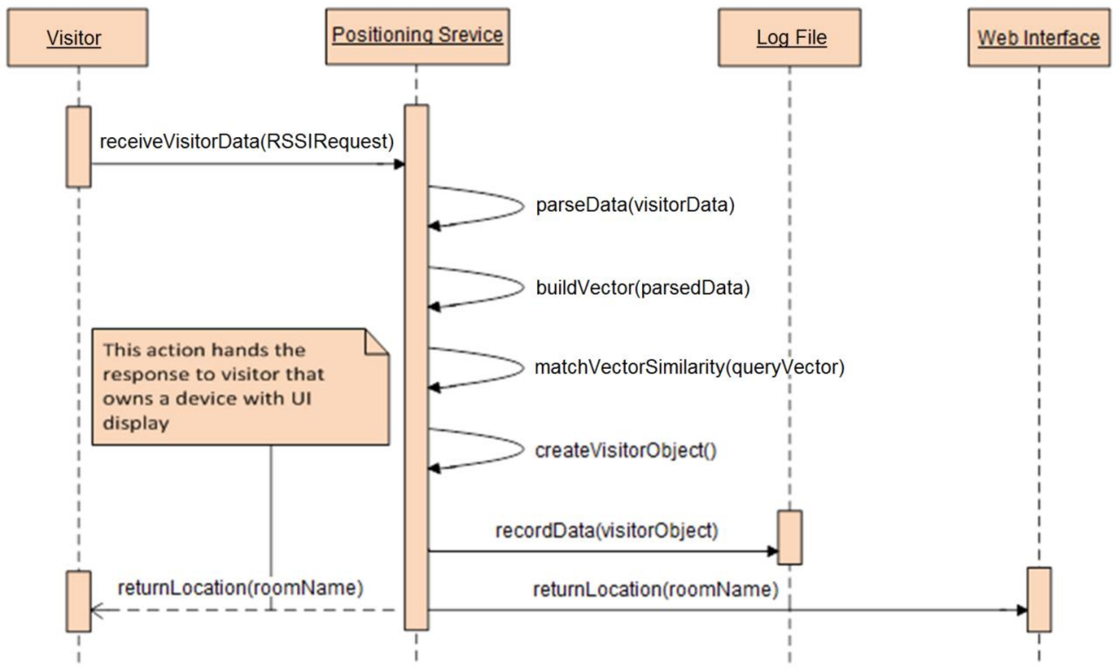

Visitor
Positioning Srevice
Log File
Web Interface
receiveVisitorData(RSSIRequest)
parseData(visitorData)
buildVector(parsedData)
This action hands the response to visitor that owns a device with UI display
matchVectorSimilarity(queryVector)
createVisitorObject()
recordData(visitorObject)
returnLocation(roomName)
returnLocation(roomName)

Figure 6. System-level sequence diagram of the Positing service.

As shown in figure 6, an actor identified as a Visitor invokes the *takeVisitorData* method from the servlet component of the Positioning Service server application by passing the signal-related information (RSSI) values found from the surrounding area. Another method called *doParsing* parses, which is implemented in the Business Logic component of the Positing Service, takes the RSSI request's data and extracts the location-related information (i.e. signal level dBm values). Then, the *buildVector* method builds a vector object representation having vector element values equivalent with the signal levels dBm values. The *matchSimilarity* method takes the query vector object built from the RSSI request, and executes similarity matching operation against the vector indexes built from the RSSI sample maps in the database index. The *createVisitorMethod* composes location and profile related information of the visitor by creating an object of *Visitor* type, which is handed to the *recordData* method that appends in the object's data to a Log File.

The *recordData* method records the estimated location, tagID or device name of the visitor and the response timestamp to a log file. The primary use of this log file is to make movement analysis and indoor activity related studies of the indoor object or visitor, the second to monitor the system performance and other internal server errors caused by software faults. Finally, the *returnLocation* method returns the estimated location which is the reference name of the matching vector object that is returned by the *matchSimilarity* method.

4.2 Android Client Application Implementation

The Android client application was implemented on top of the WiFiManager API provided by the Android SDK, which offers a complete support for managing all aspects of WiFi connectivity. The WiFiMagager object was obtained to the current context via the *getSystemServices* method call. Once the *WiFiManager* object was obtained, the *getScanResults* method was called, which returns results of the latest AP scan, containing the address of the access point (BSSID), the network name (SSID), the frequency in MHz and (frequency) and the detected signal level in dBm (level).

Listing 1 illustrates a *WiFiScanReciver* Java class extending a *BroadcastReciever* super class that is registered by an outer *Activity* class as a broadcast receiver to be invoked by the system when new WiFi scan results are available. The *WiFiScanReciever* class gets the callback via the inherited *onRecieve* method and obtains the new scanning results from the intent that activated it. The scan results that include the BSSID, SSID and signal level are parsed by the *LogParser* class to eliminate duplicate BSSIDs and SSIDs and they are stored into an object of *String* type. Finally, the *String* object containing the parsed scanning results will be handed as a parameter value to the *excutePost* method (listed in listing 2) that is responsible for opening a network connection and sending the data of the *String* to the server in the form of an *OutputStream* object.

```
private WifiManager wifi = (WifiManager) getSysteSerice(Context.
          WIFI_SERVICE);

class WifiScanReceiver extends BroadcastReceiver {
    public void onReceive(Context c, Intent intent) {

            StringBuilder sb = new StringBuilder();
            List<ScanResult> wifiList = main
                    Wifi.getScanResults();

            for (ScanResult result : wifiList) {
               sb.append("" +result.BSSID+ "/" +result.SSID+ "/"
                    +result.frequency+ "/" +result.level+ "\n");
            }

            LogParser logParser = new LogParser();
            logParser.parse(sb.toString());
            requestContent = deviceName + "\n" + logParser.getR
                    esult();
          }
}
```

Listing 1. Java code implementation of the WiFiScanReciever class.

Listing 2 illustrates the *excutePost* method which is implemented in the same outer class that also implements the *WiFiScanReciver* inner class (listed in listing 1). The *excutePost* method takes a *String* object (explained in the paragraph prior to listing 1) as a parameter value, opens a network connection using the *HTTPURLConnection* object to the server hosting the Positioning Service, and sends an HTTP Post request containing binary data representation of the *String* object by calling the *writeBytes* method of the Data*OutputSream* object.

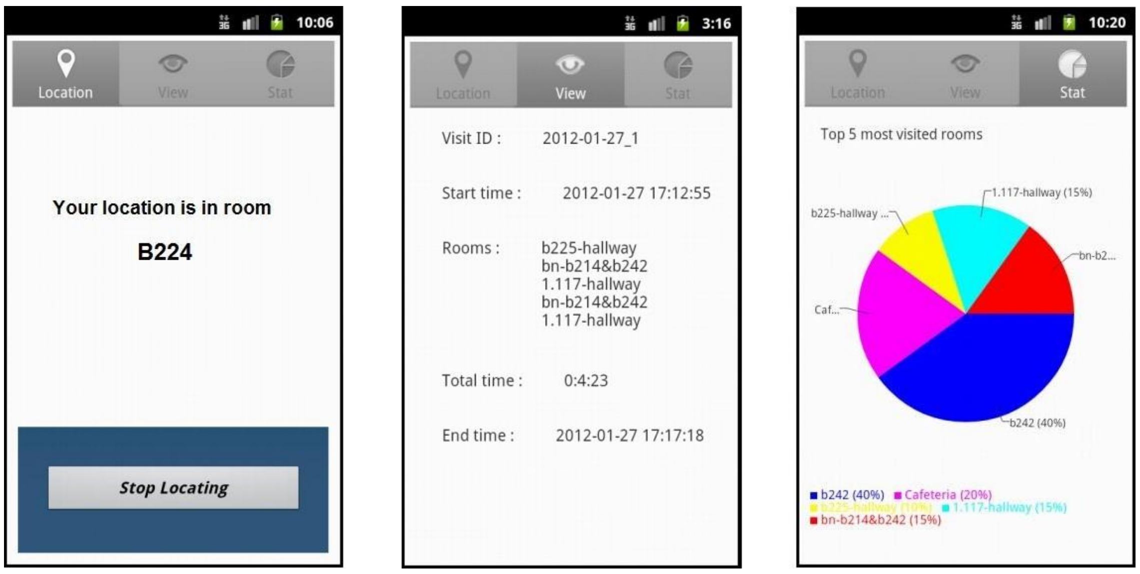
If connection with the server has been established successfully, the *getResponseCode* method called on the reference object of the *HttpURLConnection* type returns the 'HTTP_OK' response code, and the server receives the request as an *InputStream* object. As a result, the server is also able to return the response over the opened connection, and the getInputStream method on the client application accepts the response and forwards it to the responsible component, which handles graphical interpretation and UI visualization aspects. For the time being, the client application only displays the name of the location in a simple text format as shown on the screen shot image of the application UI (left side) in figure 7, but a more dynamic UI is in the process of the development.

```
public String executePost(String requestContent) {
        String serverResponse = null;
        try {
            // Create connection
            String targetURL = "http://hostingserver:port/RSS
                  ISimilarityWeb/update-location";
            URL url = new URL(targetURL);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("POST");
            // Send request
            DataOutputStream wr = new DataOutputStream(connec
                 tion.getOutputStream());
            wr.writeBytes(requestContent);
            wr.flush();
            wr.close();

            if (connection.getResponseCode() == HttpURLCon
                 nection.HTTP_OK) {
                 // Get Response
                 InputStream is = connection.getInputStream();
                 BufferedReader rd = new BufferedReader(new
                       InputStreamReader(is));
                 String line;
                 StringBuffer response = new StringBuffer();
                 while ((line = rd.readLine()) != null) {
                       response.append(line);
                       response.append('\r');
                 }
                 rd.close();
                 is.close();
                 serverResponse = response.toString();
             } else {
                   serverResponse = "Connection error: "
                            + connection.getResponseCode();
             }
        } catch (Exception e) {
              Log.d(TAG, e.toString());
              return null;
        }
        return serverResponse;
}
```

Listing 2. Java code implementation of the executePost method.

Figure 7 shows a screen shot image of the Android client application displaying the real-time server response of the visitor's current location (on the left side) and overall visit statistics (on the right side). The current location of the visitor is displayed on the Location tab of the application when the visitor clicks on the 'Start Locating' button. The Stat tab displays a pie chart diagram, which demonstrates the overall visit statistics data from the database that is generated by the Movement analysis service.



Your location is in room

B224

Visit ID : 2012-01-27_1

Start time : 2012-01-27 17:12:55

Rooms : b225-hallway
bn-b214&b242
1.117-hallway
bn-b214&b242
1.117-hallway

Total time : 0:4:23

End time : 2012-01-27 17:17:18

Top 5 most visited rooms

■ b242 (40%) ■ Cafeteria (20%)
■ b225-hallway (10%) ■ 1.117-hallway (15%)
■ bn-b214&b242 (15%)

Figure 7. Real-time location and overall visit statics on the Android client application.

The Location tab on the Android client application, shown on the left hand side picture of figure 7, displays real-time estimate location of the target visitor when the 'Start Locating' button is clicked. At the moment this tab presents a simple text label telling the current location name of the visitor in the form of “Your location is in room X”, where the value of X is substituted with the exact room or area name returned by the Positing Service. In future plans, the interface of this tab is intended to be redesigned and modified with a new interface look which has reach dynamic UI features on the top of the indoor environment floor plan graphics.

The View tab shown in the middle picture presents textual information of the analyzed movement data that is derived by the Movement Analyzer Service and stored in the database. The analyzed movement data contains a visit ID, visit start time, last five visited rooms, total time spent during the whole visit and visit end time. This view gives an easy to read general and collective movement information of the specific visitor in identically nested grid view.

The Stat tab on the right hand side picture illustrates a pie-chart diagram that displays a statistical database data telling the time spent or duration proportion shared between top five most visited rooms and the rest of the visited rooms. When the user clicks on the Stat tab, an HTTP Get request is sent to the database including the name (ID) of the device on the request body, which will be used as a 'key' pair in order to select the analyzed movement data of the requesting visitor from the table in a key-value pair oriented database. Then, the database returns a JavaScript Object Notation (JSON) document response, containing the analyzed movement data and profile information of the selected visitor. As shown in appendix 2, the JSON document response retrieved from the server parsed and extracted into Java objects on the client side of the Android application, which will then be displayed on the View and Stat tab of the application UI.

4.3 Movement Analyzer Service Implementation

The time spent or residing duration was able to be derived by calculating the time difference from subsequent location timestamp to the preceding location timestamp. As described in the last paragraph of the preceding section, a single line in the log file consists of a device name or tag ID, current location (location or room name) and the response timestamp generated by the Positioning Service. Thus, subtracting the timestamp value on the subsequent line from the timestamp value on the preceding line both lines having the same device name or tag ID gave the time spent or residing duration of the visitor identified by the same device name or tag ID inside the room appended on the previous line.

The visit start time and end time were determined by taking the timestamp value from the first ever appended line and the last appended line of having the same device name or ID. The total visit time spent during the whole visit was computed by summing up the time spent or the duration values of each visited location (room). In addition, average time spent in each room was calculated by dividing the total visit time by the total number of visited rooms.

Figure 8 shows a screen shot image of the client web application visualizing the time distribution of all the visited exhibitions of the specific visitor selected from the dropdown menu. The visits have taken place in one of the museum's weekend events, which also served as an experiment trails for testing the Position service. The test lasted for about two days that is equivalent to 16 hours and was categorized as a single visit event for the specific visitor identified by visit ID.

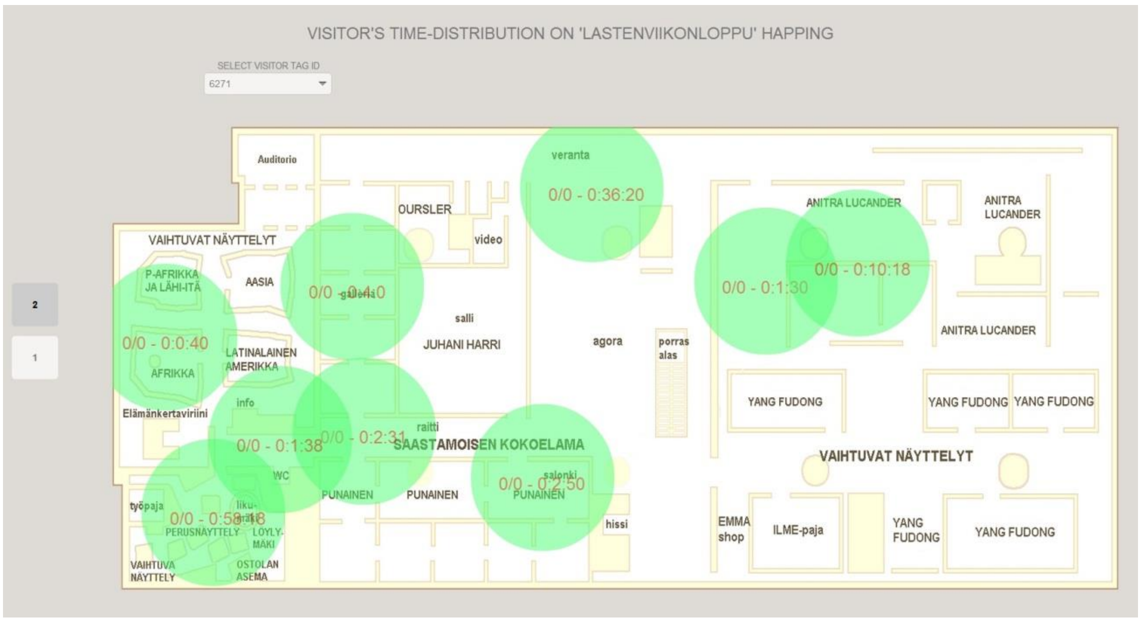


Figure 8. Visitor 6271 time-distributions among the visited exhibitions.

Figure 8 illustrates the visited exhibitions and the amount of the time spent in each exhibition for the specific visitor selected from the dropdown menu that is identified with a 6271 tag ID. The light green color circles cover the square radius area of the estimated locations (i.e. exhibition sections) visited by the selected visitor. The duration or time spent of each visited exhibition sections is marked with a red color text, written in the form of 'Year/day HH:mm:ss' in the middle of the circles. A Scalable Vector Graphics (SVG) JavaScript library has been used for drawing the circle shapes and texts dynamically when the dropdown selection is completed.

To give a brief explanation on how the client web application works, when a visitor ID is selected from the dropdown menu, a direct Uniform Resource Locator (URL) path request is sent to the database server appending the tag ID number as a URL parameter value. Then, the server returns a JSON response containing the analyzed movement data of the selected visitor, which comprises the time spent or duration taken in each visited room. The returned sever response is evaluated by the JavaScript interpreter function and handed to the method called *drawSVGShape* as an *ArrayList* object type parameter value. Finally, the *drawSVGShape* method iterates through the *ArrayList* elements, and draws the SVG circle shapes at the exact pixel coordinates on the floor plan image of the building.

5 Positioning Service Result Analysis

Experimentation of the position-estimation system was conducted in the school building environment (Metropolia University of Applied Sciences, Leppävaara unit), using the already installed WiFi networks. The RSSI sample maps were manually collected from each room and common area in the building at an average sample point of 3 to 4, using the WiFi Scanner Android application illustrated in figure 5. Overall the result of the Positioning Service was satisfactory with a room level accuracy standard, except for the few random errors that occurred due to various reasons mentioned in the next paragraph affecting the RSSI measurement values on both the RSSI sample maps and the target object's RSSI request.

One of the most significant reasons that could be mentioned in the first place is the signal propagation behavior of the APs. In indoor environment RSSI values are highly uncertain and fluctuating all the time, for example, due to interior building materials (e.g. glass walls), people's presence, room temperature and humidity level. In digital communication, the purpose of the received power is to avoid burst error and ensure high bit-error rate communication. However, the level change of RSSI is not important as long as it is maintained within the safety region. Therefore, when the RSSI method is used for location estimating purposes, the level of the accuracy falls in jeopardy, because the estimation process is directly based on the RSSI value.

Hence, different solutions were considered in the implementation of the positioning system in order to minimize the expected errors. The first solution includes a collection of the RSSI sample maps from several sample points within the collection area (i.e. room or common area), which helped to decrease the probability of choosing the wrong RSSI sample map as the estimated location result of the visitor. As a result, the Positioning Service (explained in section 3.3.1) has less probability to generate ambiguous location estimation results while performing its similarity matching operation. The other error reducing solution considered in the future plan includes improving the signal quality of the RSSI using standards that are optimized for the location estimation solutions, such as DASH7, IPCS nodes and Bluetooth LE.

Figure 9 illustrates a comparison of the estimated route with the actual route of the target object taken during the 35 second experimentation walk while testing the Positioning Service server application (explained in section 3.3.1). The location of the experiment was chosen deliberately so that the possibility of ubiquitous results could be large enough due to the closer nature of the nearby room and the glass materials used for portioning the interior walls.

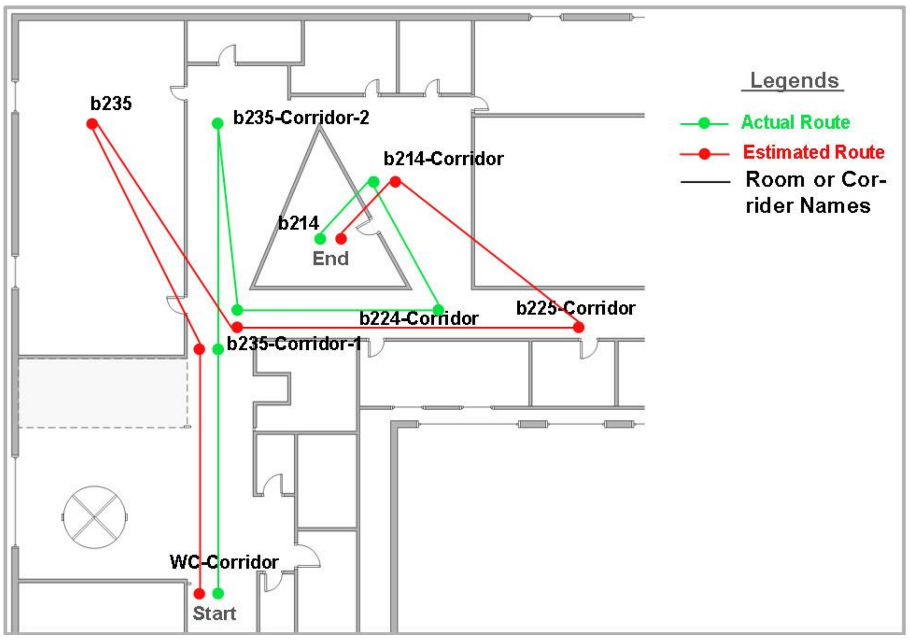


Figure 9. Result of actual and estimated routes.

As shown by the experiment results, random errors occurred at two points on the whole walking route between the starting point (WC-Corridor) and the ending point (room b214). In both error cases, the Positioning Service gave the nearest room name as a result of the estimated location name. At the third requesting point, the Positioning Service generates the 'b235' room name as the estimated location name, while the actual location of the visitor was in b235-Corridor. Similarly, while the actual location of the visitor was in b224-Corridor at the fifth requesting point, the Position Service generates the b225-Corrider location name which is the next corridor from the b224-Corridor.

6 Conclusions

In this bachelor's thesis, developing an indoor localization system using the RSSI similarity matching approach has been discussed comprehensively. The system uses RSSI values sent by a target object or visitor, and performs 'similarity matching' operation in order to determine the location of the target object or visitor. An indoor visitor having the Android client application installed on an Android-powered device can request its current location by sending an RSSI request containing signal level values found from the surrounding active APs. Estimating the real-time location of a target object or visitor with room level accuracy has been successful as can be observed based on the experiment results. Typical indoor environment groups, which would benefit a huge gain from using the indoor localization system, include social service providers and recreational areas such as medical institutions, shopping malls, hotels, airports, academic institutions and museums.

The system uses existing WLAN networks, which are widely available in most of the indoor environments and supported by most mobile devices such as smart phones, RFID tags, Personal Digital Assistant (PDAs) and tablets. However, WLAN network standards are not optimized for locating or tracking an indoor object, rather for communication purposes by providing higher data rates that allow devices to exchange data within a smaller coverage area. Thus, the level of the location estimation accuracy has been influenced with few random errors. In order to develop an indoor localization system with high-quality precision level, protocols such as DASH7 and Bluetooth LE that are optimized for localization purposes have to be used instead of the WiFi protocol. In addition, better consideration of physical and environmental conditions such as humidity level, room temperature and interior building materials that could affect the signal propagation behavior would play a significant role in optimizing the localization system.

References

1 The DASH7 Alliance. What is DASH7 technology [online]. California, USA: The DASH7 Alliance; 22 August 2009.
URL: http://www.dash7.org/index.php?option=com_content&view=article&id=11&Itemid=13. Accessed 22 December 2011.

2 Norair JP. Open tag DASH7 stack overview [online]. California, USA: Blackbird Technology; 16 December 2009.
URL: http://www.dash7.org/OpenTag%20Webinar.pdf. Accessed 2 January 2012.

3 Chandermouli V. A detailed study on wireless LAN technologies [online]. Texas, USA: The University of Texas.
URL: http://www.uta.edu/oit/policy/ns/docs/wireless-paper-vijay.pdf. Accessed 4 January 2012.

4 Bluetooth Special Interest Group. Bluetooth wireless technology [online]. Kirkland, WA, USA: Bluetooth Special Interest Group.
URL: https://www.bluetooth.org/Building/overview.htm. Accessed 5 January 2012.

5 Decuir J. Bluetooth 4.0: Low energy [online]. Cambridge, UK: Cambridge Silicon Radio SR plc; 2010. URL: http://chapters.comsoc.org/vancouver/BTLER3.pdf. Accessed 3 February 2012.

6 Bluegiga Technologies. Bluetooth low energy [online]. Espoo, Finland: Bluegiga Technologies; 15 May 2011.
URL: http://www.glynstore.com/content/docs/bluegiga/BLE_getting_started.pdf. Accessed 6 Feburary 2012.

7 Cassaniti D. A multi-hop 6Lo WPAN wireless sensor network for waste management optimization [online]. Padua, Italy: University of Padua; 2012.
URL: http://tesi.cab.unipd.it/37629/1/Dario_Cassaniti_-_Tesi_Magistrale_-_Master_Thesis_2011.pdf. Accessed 4 February 2012.

8 Gohn B. ZigBee enables modern eco-friendly home networks [online]. Boston, USA: Ember Corporation; March 2008.
URL: http://www.smallformfactors.com/articles/id/?3040. Accessed 08 Feburary 2012.

9 Coleri S. ZigBee/IEEE 802.15.4 summary [online]. California, USA: Berkeley University of California; 10, September 2004.
URL: http://pages.cs.wisc.edu/~suman/courses/838/papers/zigbee.pdf. Accessed 8 Feburary 2012.

10 Pu C-C, Pu C-H, Lee H-J. Emerging communications for wireless sensor networks. China: InTech; February, 2006.

11 Garmin Ltd. What is GPS? [online]. Kansas, USA: Garmin Ltd; 2011.
URL: http://www8.garmin.com/aboutGPS/. Accessed 7 Feburary 2012.

12 Geneq inc. GPS – How it works [online]. Quebec, Canada: Geneq Inc. URL: http://www.sxbluegps.com/gps-error-Budget.html. Accessed 07 January 2012.

13 ESO. How does GPS work? [online]. Munich, Germany: ESO; 02, January 2012. URL: http://www.eso.org/public/outreach/eduoff/seaspace/navigation/navgps/na v gps-3.html. Accessed 08 February 2012.

14 Zezula P, Amato G, Dohnal V, Batko M. Similarity search. USA: Springer Science + Business Media, Inc; 2006.

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```java
package fi.metropolia.spagu.web;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Properties;
import java.util.Random;

import net.obsearch.ambient.bdb.AmbientBDBJe;
import net.obsearch.example.vectors.VectorsDemo;
import net.obsearch.exception.IllegalIdException;
import net.obsearch.exception.NotFrozenException;
import net.obsearch.exception.OBException;
import net.obsearch.exception.OBStorageException;
import net.obsearch.exception.PivotsUnavailableException;
import net.obsearch.index.ghs.impl.Sketch64Int;
import net.obsearch.pivots.AcceptAll;
import net.obsearch.pivots.rf04.RF04PivotSelectorInt;
import net.obsearch.result.OBPriorityQueueInt;
import net.obsearch.result.OBResultInt;

import net.obsearch.index.bucket.impl.*;
import net.obsearch.index.IndexInt;
import net.obsearch.index.ghs.impl.*;
import net.obsearch.exception.*;

import org.apache.log4j.Level;
import org.apache.log4j.Logger;

import sun.reflect.ReflectionFactory.GetReflectionFactoryAction;

import draft.ManhattanIntVector;

public class WLANSimilarityMatch {
    static ResponseParser responseParser = new ResponseParser();
    //random generator
    final static Random r = new Random();
    /** Query count.           */
    final static int QUERY_SIZE = 5;
    /** Dimension of the vectors.        */
    public static int VEC_SIZE = 115;
    final static File INDEX_FOLDER = new
          File("c://data/bench/spagu/sim_db" + File.separator
                   +"wlanDB");
```

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```
/** LOAD OBSEARCH PARAMS
* @throws IOException*/
Public static void init() throws IOException {
      File f = new File("obsearch.properties");
      // Debug
      if(!f.exists()) {
             System.err.println("Init failed, file does not
                    exist"+f.getAbsolutePath());
             System.exit(0);
      }
      Properties props = new Properties();
      try {
             props.load(new FileInputStream(f));
      } catch (IOException e) {
             System.err.println("error loading properties");
      }
}
/** FOR INDEXING VECTORS
* @return  */
public static ManhattanIntVector generateIntVector() {
      int[] data = new int[VEC_SIZE];
      int i = 0;
      while (i < data.length) {
             data[i] = r.nextInt(1000);
             i++;
      }
      return new ManhattanIntVector(data);
}

private static SignalParser signalParser;
private static Sketch64Int<ManhattanIntVector> index;
private AmbientBDBJe<ManhattanIntVector, Sketch64Int<Manhat
      tanIntVector>> a;
// At the indexing is done every time from the files, this
// will be removed if better solution is invented
private HashMap<ArrayList<Integer>, String> samples;
private static HashMap<String, String> samplesStr;
private static boolean accesspointsLoaded = false;

public WLANSimilarityMatch() {
      try {
          // Create the pivot selection strategy
          RF04PivotSelectorInt<ManhattanIntVector> sel = new
           RF04PivotSelectorInt<ManhattanIntVector>(new Ac
                ceptAll<ManhattanIntVector>());
          sel.setDataSample(100);
          //TYPE OF THE INDEX, ABSTRACT
          index = new ketch64Int<ManhattanIntVector>(Manhatt
                anIntVector.class, sel, 64);
          // error expected
          index.setExpectedError(VEC_SIZE);
```

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```
                // small if you are planning to insert a lot of ob
                      jects!
                index.setSampleSize(100);
                // Probability of returning an error within 1.40
                      times the real distance
                //  (measured in standard deviations)  (3 meansa
                      prob.of 0.99)
                index.setKAlpha(3);
                // select the ks that the user will call.
                // This example will only be called with k=1
                index.setMaxK(new int[]{1,3,5,10});
                // little optimization that can help if your ob
                      jects are of the same size.
                index.setFixedRecord(true);
                index.setFixedRecord(VEC_SIZE);
                // Create the ambient that will store the index's
                      data. (NOTE: folder name is hardcoded)
                //CONCRETE, PHYSICAL INDEX
                a = new AmbientBDBJe<ManhattanIntVector, ketch64In
                      t<ManhattanIntVector>>(index, INDEX_FOLDER);

                signalParser = new SignalParser();

                if(signalParser.loadAccesspoints()) {
                    accesspointsLoaded = true;
                }
            } catch (FileNotFoundException e) {
                      e.printStackTrace();
            } catch (OBStorageException e) {
                      e.printStackTrace();
            } catch (NotFrozenException e) {
                      e.printStackTrace();
            } catch (IllegalAccessException e) {
                      e.printStackTrace();
            } catch (InstantiationException e) {
                      e.printStackTrace();
            } catch (OBException e) {
                      e.printStackTrace();
            } catch (IOException e) {
                      e.printStackTrace();
            }
    }

    /** Lets keep this separate, we call this when we need to re
            index the data  */
    public void doIndex() {
            try {
                signalParser.parseAccessPoints();
                samples = signalParser.parseSignals();
                samplesStr = this.generateStrSamples(samples);
                WlanSimilarity.VEC_SIZE = samples.size();
                int DB_SIZE = samples.size();
```

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```
                // Loop for each vector
                for(ArrayList<Integer> sample: samples.keySet()) {
                      int [] temp = new int[sample.size()];
                      int pointer = 0;

                      for(Integer val : sample) {
                           temp[pointer] = val;
                           pointer++;
                      }
                      ManhattanIntVector item = new Manhatt
                           anIntVector(temp, samples.get(sample));
                      index.insert(item);
                }
                a.freeze();
          } catch (FileNotFoundException e) {
                e.printStackTrace();
          } catch (IOException e) {
                e.printStackTrace();
          } catch (OBStorageException e) {
                e.printStackTrace();
          } catch (OBException e) {
                e.printStackTrace();
          } catch (IllegalAccessException e) {
                e.printStackTrace();
          } catch (InstantiationException e) {
                e.printStackTrace();
          } catch (PivotsUnavailableException e) {
                e.printStackTrace();
          }
    }

    private HashMap<String, String> generateStrSamples(HashMap
          <ArrayList<Integer>, String> samples2) {
          HashMap<String,String> result = new ashMap<String,
                String>();
                for(ArrayList<Integer> i: samples2.keySet()) {
                      System.out.println(i == null);
                      result.put(i.toString(), samples2.get(i));
                }
                return result;
    }

    public List<String> getRooms(String str) throws FileNot
          FoundException, IllegalIdException, IllegalAccessExcep
                tion, InstantiationException, OBException {
          if(!accesspointsLoaded) {
                System.err.println("ACCESS POINTS NOT LOADED!");
                return null;
          }
```

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```
        ArrayList<Integer> signal = signalParser.parseSigna
              l(str);
        int [] temp = new int[signal.size()];
        int pointer = 0;
        // For each vector
        for(Integer val: signal) {
              temp[pointer] = val;
              pointer++;
        }
        List<String> roomnamesWithDistances = getSimilar
              (temp);

        return roomnamesWithDistances;
  }
  /** Input: HashMap<String, Integer> --> where String is
  * AP, integer is signal strength */
  public static List<String> getSimilar(int [] vector)
        throws IllegalIdException, IllegalAccessException,
               InstantiationException, OBException {
        List<String> rooms = new ArrayList<String>();
        //query range: from how wide area the items will be
        // retrieved
        int range = 10000;
        // now we can match some objects!
        final int RESULTSETSIZE = 1;
        RF04PivotSelectorInt<ManhattanIntVector> sel = new
        RF04PivotSelectorInt<ManhattanIntVector>(new
               AcceptAll<ManhattanIntVector>());
        sel.setDataSample(100);
        // Reset the stats counter
        index.resetStats();

        long start = System.currentTimeMillis();
        List<OBPriorityQueueInt<ManhattanIntVector>> que
               ryResults = new
        ArrayList<OBPriorityQueueInt<ManhattanIntVector>>(
               QUERY_SIZE);
        List<ManhattanIntVector> resultset = new ArrayLi
               st<ManhattanIntVector>(RESULTSETSIZE);
        int i = 0;
        // HERE WE NOW LOOP THROUGH 10 BEST MATCHES, ONLY
        // FIRST ONE IS RETRIEVED
        while(i < RESULTSETSIZE){
               ManhattanIntVector q = new ManhattanIntVec
                      tor(vector);
               // Query the index with k=1
               OBPriorityQueueInt<ManhattanIntVector> queue =
                      new OBPriorityQueue
                             Int<ManhattanIntVector>(10);
               //Perform a query with a large range and // k=1
               index.searchOB(q, range , queue);
```

Appendix 1: Main Functionalities of WLANSimilarityMatch.java

```
                    queryResults.add(queue);

                    for(OBResultInt<ManhattanIntVector> f : qu
                           eue.getSortedElements()){
                    // Check that the id makes sense
                    assert index.getObject(f.getId()).equals(
                           f.getObject());
                    assert f.getDistance() <= range;
                           resultset.add(f.getObject());
                    }
                    i++;
            }
            // print the results of the set of queries.
            long elapsed = System.currentTimeMillis() - start;
            // Get(0) value in the first index of the ar
                    raylist
            return rooms;
    }
```

Appendix 2: CouchDBDAO.java implementation of the Android client application

```java
package fi.metropolia.spagu.indoorlocalizion.data;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.LinkedHashMap;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;


public class CouchDBDAO {

   private static String TAG = "CouchDBDAO";
   private Visitor visitor = new Visitor();
   private String deviceName = android.os.build.MODEL;

   public String excuteHTTPRequest() {
          HttpClient httpclient = new DefaultHttpClient();
          HttpGet get = new HttpGet("http://hostingserver.port/
              couchdb/location/_design/location/_view/visitor?ke
                   y=%22" +deviceName+ "%22");

          HttpResponse response = null;
          InputStream instream = null;
          try {
               response = httpclient.execute(get);
               HttpEntity entity = response.getEntity();
               instream = entity.getContent();
          } catch (ClientProtocolException e) {
               e.printStackTrace();
          } catch (IOException e) {
               e.printStackTrace();
          } catch (IllegalStateException e) {
               e.printStackTrace();
          }
```

Appendix 2: CouchDBDAO.java implementation of the Android client application

```
            BufferedReader reader = new BufferedReader(new InutStr
                      eamReader(instream));
            String strdata = null;
            StringBuffer JSONResponse = new StringBuffer();

            try {
                 while( (strdata = reader.readLine()) != null) {
                      JSONResponse.append(strdata);
                      JSONResponse.append('\r');
                 }
            } catch (IOException e) {
                 Log.d(TAG, e.toString());
            }
                 return JSONResponse.toString();
     }

     public ArrayList<InstantLocation> parseJSONObject(String
          JSONstr) {

          ArrayList<InstantLocation> instLocList = new ArayList<I
                 nstantLocation>();

          try {
               JSONObject json = new JSONObject(JSONstr);
               json =  json.getJSONArray("rows").getJSONObject(0.g
                      etJSONObject("value");
               JSONArray route = (JSONArray) json.getJSONArray("Ro
                      ute");

             for (int i = 0; i < route.length(); i++) {
                 InstantLocation instLoc = new InstantLocation();
                 JSONObject childJSONObj = route.getJSONObject(i);

                 String roomName = childJSONObj.getString("room");
                 long duration = childJSONObj.getLong("duration");

                 instLoc.setRoomName(roomName);
                 instLoc.setDuration(duration);

                 instLocList.add(instLoc);
             }
          } catch (JSONException e) {
            Log.d(TAG, e.toString());
          }
            return instLocList;
     }
}
```