Development of an Android APK for Bluetooth localization

Author: Carlos Carrera Carbó
Director: ING. BC. Lukas Vojtech
Prague, 2011/2012
General Index

1. Chapter 1: Introduction ...................................................... 2

2. Chapter 2: Bluetooth system .............................................. 5
   2.1 What is bluetooth? ..................................................... 5
   2.2 Communication ....................................................... 6
   2.3 Uses .................................................................. 7
   2.4 List of applications .................................................. 8

3. Chapter 3: Frequency hopping .......................................... 11
   3.1 What is freq. Hopping? .............................................. 11
   3.2 Technical considerations .......................................... 12
   3.3 FIR filter ................................................................ 13
   3.4 Order change .......................................................... 14

4. Chapter 4: RSSI ................................................................. 17

5. Chapter 5: GUI designing ................................................. 24
   5.1 First ideas and development ..................................... 24
   5.2 Screenshots and testing ............................................. 26

6. Chapter 6: Conclusion ........................................................ 30

7. Bibliography ..................................................................... 32
# Images and tables Index

## Chapter 1:
- Pic.1.1 Golf Player on a field ........................................ 2
- Pic.1.2 Smartphone Statistics ........................................ 4

## Chapter 2:
- Pic.2.1.1 Bluetooth board ........................................... 5
- Table 2.3.1 Power emitted and range .............................. 7
- Table 2.3.2 Bluetooth capacity ...................................... 8

## Chapter 3:
- Pic.3.3.1 FIR filter schematics example .......................... 13
- Pic.3.4.1 Filter order 10 ............................................... 14
- Pic.3.4.2 Filter order 21 ............................................... 14
- Pic.3.4.3 Filter order 51 ............................................... 15
- Pic.3.4.4 Filter order 71 ............................................... 15
- Pic.3.4.5 Filter order 81 ............................................... 15
- Pic.3.4.6 Filter order 113 ............................................. 15
- Pic.3.4.7 Filter parameters ........................................... 16

## Chapter 4:
- Pictures 4.1 to 4.24 Measurement results ........................ 18-20
- Table 4.1 Chosen ranges .............................................. 21
- Pic.4.25 Code ............................................................ 22
- Pic.4.26 Flux Diagram ................................................ 23
Chapter 5:

- Pictures 5.1.1 Logo ------------------------------------------------------------- 24
- Pictures 5.1.2 to 5.1.4 Scratches of the layout ----------------------------- 24-25
- Pictures 5.2.1 to 5.2.7 Screenshots ------------------------------------- 26-29
Abstract

Wireless sensor networks are becoming ubiquitous and their application areas are widening by the day. Localization algorithms play an important role in enhancing the utility of data collected by enabling sensors to determine the location from which each data packet is obtained. Localization can be done by implementing beacon based algorithms or signature based algorithms. Much of the research work in this area assumes received signal strength indicator (RSSI) as a parameter in their localization algorithms. Since RSSI is the key parameter we assume that it’s possible to make an application for Android phones to find Bluetooth devices (not only phones, but any other object with a Bluetooth emitter). The main idea is to make it for golf players who want to find balls in a field. The idea is to implement a FIR filter programmed in Android, as the other parts of the code (Bluetooth API and visual application), to get the best signal avoiding the multipath issues, and once we have the RSSI value of the Bluetooth signal, calculate the distance between both devices.
CHAPTER 1. Introduction

The aim of this project is to study and create an application (APK) for android phones, that using the Bluetooth API can measure the distance between two devices.

The final purpose of the application, is to design a golf ball finder. Installing a Bluetooth emitter inside the golf ball, and having in your phone the application, the user will be able to find the ball in a field, or at least know how far is it, making it easy to find it.

Most new players have always been annoyed by the loose of too many balls, especially in their first games, so this application will make their start in this amazing game a bit easier.

In order to know if our application can have a its own place in the APK’s market, we should make a cross-compared study, using the data from the numbers of golf player (golfers), and the number of Android phones users.

We usually see in golf a posh and “only-for-the-rich” sport, but in the last 20 years, that image has been changed a lot. Now the sport is more reachable to a more general public. Actually, According to Sports Marketing Surveys, there
are 61.1 million golfers worldwide; 6.9 million in Europe, 13.6 million in Asia, 1.7 million in Australasia, 1 million in South America and 500 000 in South Africa. There are 37.1 million in the US alone.

The second statistics we should have a look is the Android phones users.

According to the data from ABI Research, for each person that has a device with iOS system (iPhone, iPad, iPod), 2.4 has an Android device (smartphone or tablet). And the market studies reveal that this difference will be even larger around year 2016, when this difference should be 3.1 Android users for each iOS user. (We are comparing this two operating systems, as are the ones which has more users). To get a more accurate idea, in 2011 Android reach the 100 million activations worldwide.

Keeping on checking the android market status, 44% of all downloaded applications are for android, compared to the 31% that are for iOS systems. The number of applications downloaded are a key point in order to get our APK. As a developer we need to know how, when, and by how much upload and sell our APK, and of course, how to make it attractive.

Just for explaining the importance of the applications downloading, two numbers: last year, about 9000 million applications where downloaded (in all different operating systems), and this year 29000 million ones. Of course, this number will keep on increasing as years goes by.

If we have a look at both numbers, golfers and android users, we can say that a golf application can have its spot in the Android Market, and that it’s possible to have at least a little success.
Pic. 1.2 Smartpone operating system statistics[14]
CHAPTER 2. Bluetooth system

2.1 What is Bluetooth?

Bluetooth is a proprietary open wireless technology standard for exchanging data over short distances (using short wavelength radio transmissions in the ISM band from 2400-2480 MHz) from fixed and mobile devices, creating personal area networks (PANs) with high levels of security. Created by telecoms vendor Ericsson in 1994, it was originally conceived as a wireless alternative to RS-232 data cables. It can connect several devices, overcoming problems of synchronization.

Bluetooth is managed by the Bluetooth Special Interest Group, which has more than 15,000 member companies in the areas of telecommunication, computing, networking, and consumer electronics. The SIG oversees the development of the specification, manages the qualification program, and protects the trademarks. To be marketed as a Bluetooth device, it must be qualified to standards defined by the SIG. A network of patents is required to implement the technology and are only licensed to those qualifying devices; thus the protocol, whilst open, may be regarded as proprietary.

Pic. 2.1.1 Bluetooth board[15]
2.2 Communication and connection:

A master Bluetooth device can communicate with a maximum of seven devices in a piconet (an ad-hoc computer network using Bluetooth technology), though not all devices support this limit. The devices can switch roles, by agreement, and the slave can become the master (for example, a headset initiating a connection to a phone will necessarily begin as master, as initiator of the connection; but may subsequently prefer to be slave).

The Bluetooth Core Specification provides for the connection of two or more piconets to form a scatternet, in which certain devices simultaneously play the master role in one piconet and the slave role in another.

At any given time, data can be transferred between the master and one other device (except for the little-used broadcast mode). The master chooses which slave device to address; typically, it switches rapidly from one device to another in a round-robin fashion. Since it is the master that chooses which slave to address, whereas a slave is (in theory) supposed to listen in each receive slot, being a master is a lighter burden than being a slave. Being a master of seven slaves is possible; being a slave of more than one master is difficult. The specification is vague as to required behaviour in scatternets.

Many USB Bluetooth adapters or “dongles” are available, some of which also include an IrDA adapter. Older (pre-2003) Bluetooth dongles, however, have limited capabilities, offering only the Bluetooth Enumerator and a less-powerful Bluetooth Radio incarnation. Such devices can link computers with Bluetooth with a distance of 100 meters, but they do not offer as many services as modern adapters do.
2.3 Uses

Bluetooth is a standard wire-replacement communications protocol primarily designed for low power consumption, with a short range (power-class-dependent, but effective ranges vary in practice; see table below) based on low-cost transceiver microchips in each device. Because the devices use a radio (broadcast) communications system, they do not have to be in visual line of sight of each other, however a quasi optical wireless path must be viable.

<table>
<thead>
<tr>
<th>Class</th>
<th>Maximum permitted power</th>
<th>Range (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(mW)</td>
<td>(dBm)</td>
</tr>
<tr>
<td>Class 1</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Class 2</td>
<td>2.5</td>
<td>4</td>
</tr>
<tr>
<td>Class 3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.3.1 Power emitted and range[16]

The effective range varies due to propagation conditions, material coverage, production sample variations, antenna configurations and battery conditions. In most cases the effective range of class 2 devices is extended if they connect to a class 1 transceiver, compared to a pure class 2 network. This is accomplished by the higher sensitivity and transmission power of Class 1 devices.
<table>
<thead>
<tr>
<th>Version</th>
<th>Data rate</th>
<th>Maximum application throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1.2</td>
<td>1 Mbit/s</td>
<td>0.7 Mbit/s</td>
</tr>
<tr>
<td>Version 2.0 + EDR</td>
<td>3 Mbit/s</td>
<td>2.1 Mbit/s</td>
</tr>
<tr>
<td>Version 3.0 + HS</td>
<td>Bluetooth only for pairing devices (it uses 802.11 for transmitting)</td>
<td></td>
</tr>
<tr>
<td>Version 4.0</td>
<td>Bluetooth only for pairing devices (it’s “low energy bluetooth”)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3.2 Capacity[17]

While the Bluetooth Core Specification does mandate minimums for range, the range of the technology is application specific and is not limited. Manufacturers may tune their implementations to the range needed to support individual use cases.

2.4 List of applications

A typical Bluetooth mobile phone headset.

- Wireless control of and communication between a mobile phone and a handsfree headset. This was one of the earliest applications to become popular.

- Wireless control of and communication between a mobile phone and a Bluetooth compatible car stereo system

- Wireless Bluetooth headset and Intercom.
Wireless networking between PCs in a confined space and where little bandwidth is required.

Wireless communication with PC input and output devices, the most common being the mouse, keyboard and printer.

Transfer of files, contact details, calendar appointments, and reminders between devices with OBEX.

Replacement of previous wired RS-232 serial communications in test equipment, GPS receivers, medical equipment, bar code scanners, and traffic control devices.

For controls where infrared was often used.

For low bandwidth applications where higher USB bandwidth is not required and cable-free connection desired.

Sending small advertisements from Bluetooth-enabled advertising hoardings to other, discoverable, Bluetooth devices.

Wireless bridge between two Industrial Ethernet (e.g., PROFINET) networks.

Three seventh-generation game consoles, Nintendo's Wii and Sony's PlayStation 3 and PSP Go, use Bluetooth for their respective wireless controllers.

Dial-up internet access on personal computers or PDAs using a data-capable mobile phone as a wireless modem.

Short range transmission of health sensor data from medical devices to mobile phone, set-top box or dedicated telehealth devices.

Allowing a DECT phone to ring and answer calls on behalf of a nearby cell phone.
- Real-time location systems (RTLS), are used to track and identify the location of objects in real-time using “Nodes” or “tags” attached to, or embedded in the objects tracked, and “Readers” that receive and process the wireless signals from these tags to determine their locations.

- Personal security application on mobile phones for prevention of theft or loss of items. The protected item has a Bluetooth marker (e.g. a tag) that is in constant communication with the phone. If the connection is broken (the marker is out of range of the phone) then an alarm is raised. This can also be used as a man overboard alarm. A product using this technology has been available since 2009.
CHAPTER 3. Frequency hopping and FIR Filtering

3.1 What is Frequency-hopping?

Frequency-hopping spread spectrum (FHSS) is a method of transmitting radio signals by rapidly switching a carrier among many frequency channels, using a pseudorandom sequence known to both transmitter and receiver. It is utilized as a multiple access method in the frequency-hopping code division multiple access (FH-CDMA) scheme.

A spread-spectrum transmission offers three main advantages over a fixed-frequency transmission:

1. Spread-spectrum signals are highly resistant to narrowband interference. The process of re-collecting a spread signal spreads out the interfering signal, causing it to recede into the background.

2. Spread-spectrum signals are difficult to intercept. An FHSS signal simply appears as an increase in the background noise to a narrowband receiver. An eavesdropper would only be able to intercept the transmission if the pseudorandom sequence was known.

3. Spread-spectrum transmissions can share a frequency band with many types of conventional transmissions with minimal interference. The spread-spectrum signals add minimal noise to the narrow-frequency communications, and vice versa. As a result, bandwidth can be utilized more efficiently.
3.2 Technical Considerations

The overall bandwidth required for frequency hopping is much wider than that required to transmit the same information using only one carrier frequency. However, because transmission occurs only on a small portion of this bandwidth at any given time, the effective interference bandwidth is really the same. Whilst providing no extra protection against wideband thermal noise, the frequency-hopping approach does reduce the degradation caused by narrowband interferers.

One of the challenges of frequency-hopping systems is to synchronize the transmitter and receiver. One approach is to have a guarantee that the transmitter will use all the channels in a fixed period of time. The receiver can then find the transmitter by picking a random channel and listening for valid data on that channel. The transmitter’s data is identified by a special sequence of data that is unlikely to occur over the segment of data for this channel and the segment can have a checksum for integrity and further identification. The transmitter and receiver can use fixed tables of channel sequences so that once synchronized they can maintain communication by following the table. On each channel segment, the transmitter can send its current location in the table.

In the US, FCC part 15 on unlicensed system in the 900 MHz and 2.4 GHz bands permits more power than non-spread spectrum systems. Both frequency hopping and direct sequence systems can transmit at 1 Watt. The limit is increased from 1 milliwatt to 1 watt or a thousand times increase. The Federal Communications Commission (FCC) prescribes a minimum number of channels and a maximum dwell time for each channel.

In a real multipoint radio system, space allows multiple transmissions on the same frequency to be possible using multiple radios in a geographic area. This creates the possibility of system data rates that are higher than the Shannon limit for a single channel. Spread spectrum systems do not violate the Shannon limit. Spread spectrum systems rely on excess signal to noise ratios for sharing of spectrum. This property is also seen in MIMO and DSSS systems. Beam
steering and directional antennas also facilitate increased system performance by providing isolation between remote radios.

3.3 FIR Filter:

A finite impulse response (FIR) filter is a type of a signal processing filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which have internal feedback and may continue to respond indefinitely (usually decaying). The impulse response of an Nth-order discrete-time FIR filter (i.e. with a Kronecker delta impulse input) lasts for N+1 samples, and then dies to zero.

FIR filters can be discrete-time or continuous-time, and digital or analog.

Pic. 3.3.1 FIR filter example schematics[18]
3.4 How FIR filter change modifying its order:

In order to try different filters in our APK, we used Matlab software to design them. We decided to use a Hamming Window filter, because it’s the one that fits the best for our proposal. Although having the same parameters (sampling frequency, central frequency etc.), the most important parameter to try is the order of the filter. A higher order gives the filter more accuracy and a more narrowband, so we can pick less signals and get the one we want.

In the next pictures we can see how, with the same filter data (it filters from 2’399 Ghz to 2’401 Ghz, with a sampling frequency of 48 Ghz), the change in the order value modifies the filter. We have to think that, a bigger order also makes processing and computing a bit harder (more values to take in count).
Although we can see in the pictures that with a higher order we get a more accurate filter, we need to think also that a higher order makes processing of the signal more difficult. That worse processing makes our APK to be slower or get easier to bugs in the code.

So we have decided to use de Order 81 filter, as the improvement that has order 113 is not as remarkable to need all this new points for processing.

After creating the filter using matlab, we need to add it to our source code and make it work.
Picture below shows how the filter parameters are written in our source code:

```java
private float FIR_values[] = {
    (float)0.00359442519235900, (float)0.003656557575387599, (float)0.00466506449489721,
    (float)0.002972170797880599, (float)0.00210582940815290, (float)0.008124430703988665, (float)
    -0.00301859649334082, (float)-0.005392213141802124, (float)-0.00760466163973619,
    (float)-0.0106777813014297, (float)-0.02075550875660796, (float)-0.009480934247219521,
    (float)-0.00254622965927940, (float)0.00278305329767855, (float)0.00876252098225449,
    (float)0.000824662901450, (float)0.0238616742081911, (float)0.0254641957935988, (float)0.02
    (float)0.005025179287306, (float)0.0138768756642072, (float)0.00502183687394890, (float)-0.0
    (float)-0.0159022594092670, (float)-0.0257391133348965, (float)-0.0335929904635324, (float)-0.0
    (float)-0.0385680680215814, (float)-0.0366242951799914, (float)-0.0297324650121095, (float)-0.0
    (float)-0.0068225169895123920, (float)0.00081887973635401, (float)0.0203043182065146,
    (float)0.0409063420263713, (float)0.0449169819870154, (float)0.0449149819870154, (float)0.06
    (float)0.0318893948668687, (float)0.02030431628053164, (float)0.00691887973635401, (float)-0.0
    (float)-0.01374709895728661, (float)-0.0297324650121095, (float)-0.066242951795914, (float)-0.0
    (float)-0.0364574236076794, (float)-0.0335929904635324, (float)-0.0257391133348965, (float)-0.0
    (float)-0.00525493739387747, (float)0.00020183687394890, (float)0.0138768756642072, (float)
    0.024107187586781, (float)0.0254641957935988, (float)0.0238616742081811, (float)0.02
    (float)0.0148005819917329, (float)0.0087825209825244, (float)0.00278350329767855, (float)-0.0
    (float)-0.00672178666136796, (float)-0.00356034247219521, (float)-0.010753507610795, (float)
    -0.0035692893987206, (float)-0.00760466163973619, (float)-0.0053213151802124, (float)
    -0.0091745737341514, (float)0.008124430703988665, (float)0.0021058235081520, (float)
};
```

Pic 3.4.7. Filter Parameters written in the code.
CHAPTER 4. RSSI problems for measuring distance and a possible solution

As is proven in the project “Is RSSI a Reliable Parameter in Sensor Localization Algorithms – An Experimental Study”[1] made by the Department of Computer Science and Engineering, State University of New York at Buffalo, RSSI is not an accurate value to get a nice distance measurer.

A lot of universities, departments or research centers have tried to deal with this problem. As I said in the previous paragraph, it’s proven that it’s impossible to determine distance using RSSI, but we can give it a chance to measure at least, not accurate distances.

After thinking over this problem, and knowing that our project doesn’t need to be perfectly accurate, I decided to think it backwards.

Instead of measuring the distance using RSSI value, I decided to give each distance, a range of possible RSSI, so the APK will say the approximate distance, choosing in which range the value is.

To try this, I made an experiment using a previous APK that measures the RSSI value received. I tried in two different phones, so I can compare the results and make a better division of ranges.

For each distance (from 1 meter to 10m, we couldn’t get any signal closer), we wrote down 17 different values of RSSI, to later decide how the ranges will work.

These are the results (on left side, Samsung Galaxy S, on right Samsung Galaxy Nexus):
Taking a look at the graphics, we can realize that although not being completely lineal, it has some kind of linearity. In the graphics are not shown the awkward results (for example, a -81 value for 1 meter), so they are easier to understand and because they are useless when we want to mark some ranges.

We can have the doubt if adding more ranges (for example, every half meter) can be useful or not. In my personal opinion, more ranges will make the apk be more inexact because would be easier to step forward or backward in any range. As we don’t need a good accuracy, it’s better for the working of the program to be more simplified.
After comparing both results, the ranges we are going to use are the following (RSSI’s units are in dBm):

<table>
<thead>
<tr>
<th>RANGE</th>
<th>REAL DISTANCE</th>
<th>APK MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSI &gt; -45</td>
<td>0 meters</td>
<td>“0 meters”</td>
</tr>
<tr>
<td>-45 &gt; RSSI ≥ -51</td>
<td>1 meter</td>
<td>“1 meter”</td>
</tr>
<tr>
<td>-47 &gt; RSSI ≥ -51</td>
<td>2 meters</td>
<td>“2 meters”</td>
</tr>
<tr>
<td>-51 &gt; RSSI ≥ -54</td>
<td>3 meters</td>
<td>“3 meters”</td>
</tr>
<tr>
<td>-54 &gt; RSSI ≥ -58</td>
<td>4 meters</td>
<td>“4 meters”</td>
</tr>
<tr>
<td>-58 &gt; RSSI ≥ -61</td>
<td>5 meters</td>
<td>“5 meters”</td>
</tr>
<tr>
<td>-61 &gt; RSSI ≥ -68</td>
<td>6 meters</td>
<td>“6 meters”</td>
</tr>
<tr>
<td>-68 &gt; RSSI ≥ -71</td>
<td>7 meters</td>
<td>“7 meters”</td>
</tr>
<tr>
<td>-71 &gt; RSSI ≥ -76</td>
<td>8 meters</td>
<td>“8 meters”</td>
</tr>
<tr>
<td>-76 &gt; RSSI ≥ -80</td>
<td>9 meters</td>
<td>“9 meters”</td>
</tr>
<tr>
<td>-80 &gt; RSSI</td>
<td>10 meters</td>
<td>“10 meters or more”</td>
</tr>
</tbody>
</table>

Table 4.1. Chosen ranges

As seen in the table, we put together everything that goes further than 10 meters. That’s because we couldn’t measure anything further.

That happens because most modern smartphones still use class 2 Bluetooth emitters (explained in chapter 2). Probably, with future Bluetooth technology, we can get better distances. For example, class 1 devices should be able to reach even 100 meters.
To code this part, was such an easy algorithm as an “If” and “else if” class, that is the one in charge of choosing which range we are in each moment.

```java
private String calcularDistancia(){
    String distancia="";
    int value_short1;
    value_short1 = (int)value_short;
    if (value_short1 > -45){
        distancia= "0 meters";
    }else if (-45> value_short1 & & value_short1 >= -47)
    {
        distancia= "1 meter";
    }else if (-47> value_short1 & & value_short1 >= -51)
    {
        distancia= "2 meters";
    }else if (-51> value_short1 & & value_short1 >= -54)
    {
        distancia= "3 meters";
    }else if (-54> value_short1 & & value_short1 >= -58)
    {
        distancia= "4 meters";
    }else if (-58> value_short1 & & value_short1 >= -61)
    {
        distancia= "5 meters";
    }
    return distancia;
}
```

This picture shows the beginning of our algorithm that goes from 0 meters, to 10 meters or more.

If we have a look at the picture above, we need to explain the importance of “value_short1”. In the code, the RSSI value detected by the Bluetooth is stacked in “value_short”. In order to be able to use it, we kept it in a different value to be used in the algorithm, “value_short1”.

We also add the class to the screen display to show the new data (the approximate distance).
Flux diagram of the program:

The previous flux diagram shows how the program works once it’s initiated. First thing we need to do is connect to a device. With the Scan button we search the nearby area and collect the information about the Bluetooth devices.

Once we have found our desired device, we can select it and check the distance. This distance would be continuously updating as we move around.

As it’s explained before, the measurement it’s not accurate, and sometimes we can get a big error (for example, saying 1 meter instead of 8 meters). But most of the distances updates are meaningful, as are usually within our ranges.
CHAPTER 5. GUI designing:

5.1 First ideas and developments:

Every time a developer faces a new APK, GUI designing is a key part of it. A nice looking, easy interface, can determine the success or failure of any program.

We have decided to make a really simple graphical interface, and we tried to avoid any not necessary information that a real user won’t need (RSSI values, which are only useful from the developers point of view, the MAC address of the connected device, etc).

We also created a possible logo for our apk, which is as follows:

![Pic. 5.1.1 Logo](image)

The first menu our user will face, is a simple background where you can, clicking on the android's menu button, extract a submenu with two buttons, “Scan”, and “Make discoverable”. In future development, this background could be used for tips or extra information, as “Previously connected devices”, “Number of devices found” etc.

The screen looks as follows:

![Pic.5.1.2](image)

The bottom buttons won’t be shown from the beginning, as it’s need to press the Menu button to activate them.

“SCAN” is used to find Bluetooth devices around, and “Make discoverable” is for activating our own phone Bluetooth emitter.
Once we have press the “SCAN” button, we will see in our screen the different devices detected. In all of them we will be able to read the Name, MAC address and Static distance. Pressing on the one we want to target, we will jump into the next layout, which has more information about the device and it’s updatable.

This would be our last layout. For the users version, RSSI value is erased, as they won’t care how much power their phones received from the emitter. The name of the target device is obligatory, as we need to know whith what we are connected. If each ball has a different name (BALL1, BALL2 etc.), would be easier to find each of them if we can see their names.
5.2 Screenshots and testing:

As this pictures where just the preliminary ideas, it’s much better to show the real screenshots from the program working for real.

The pictures are taken from a Samsung Galaxy S, the one used for the field measurements to determine the ranges. In this model, the application works much better than any other we have tried in. That’s logical as everything is measured and written with the results of this phone.

The pictures below shows the actual application:

This is the first image the user will get. As I explained in the previous chapter, this image should be change to anything more useful (for example, guide for users, tips, copyright etc.). Is in this menu where we shall press the Android’s Menu button, which will make to appear the two buttons in next image.

This two buttons on the bottom part of the layout are used for activating our Bluetooth (“Make discoverable”), or to scan and connect to a new device. For connecting we will need first to find it scanning all devices nearby.
This picture shows the menu opened when we press “Connect to a device”. Here we should press “Scan for devices”, which will show us on the screen what devices and how far they are approximately (this distance value is not updatable, it’s static, if we want and updatable measurement, we should select one single device).

Picture 5.2.4 shows the first device located. We are looking for my Iphone 3gs (called “Iphone de Biko”), so we need to wait less than a second to get all devices around.
Once we have found our target device (devices were placed next to each other, so the static distance shown is “0 meters”), we should click on it to get constant control of the distance. Once we click on it, the screen should look as next picture, 5.2.6.

After selecting our device on previous screen, we will get more data (MAC address, RSSI value) and will be continuously updating as long as the device’s Bluetooth is activated.
This last screenshot shows a testing done in the airport (there are hundreds of Bluetooth devices there). With some of the found devices it’s impossible to get the name (it says null), but anyway measures it's distance. At that moment we found around 20 devices, although in the screenshot are only shown a few of them.
CHAPTER 6. Conclusion and future development

The first conclusion we get from this project, is that we can’t overwhelm the non reliability of RSSI as a factor in distance measurement.

Although this big problem, if the measurements we want to do doesn’t need to be accurate, we can try to make ranges with the RSSI values to get a approximately distance to the object.

This project is done and tried with one device working as a emitter (Iphone 3GS, which uses v2.1 with A2DP BT version). This means that with any different emitter, ranges should be recalculated to fit with the difference strength signal. This shouldn’t be a problem, as we want to calculate distance with Bluetooth equipped golf balls, which will use the same emitter. We’ll need only to recalculate it to fit with the new RSSI value.

With nowadays Bluetooth technology (at least the one used by most smartphones, which is class 2), the range of distance we can measure is really small. In our experiment we could only get signal to around 10 meters. But thinking in future developments and versions of Bluetooth, we can assure that this range will get wider and wider, being able to measure distances more useful in a golf field (from 0 to 100meters).

The name and address of the emitter should be shown on the screen (as it does now). It’s important to know if you are connected to your golf ball, or a field with different players and balls would be a chaos. This is also useful for a single player with different balls, being able to change the ball you are connected or looking for any of them.
A good idea is also to add a message saying if we are approaching or getting further of our target. As we are not triangulating, it’s difficult to know the direction where is the target. That can be annoying for the user, so pointing out at least if you are getting closer, can be a good clue to find the target ball easier.

After finishing the final version, I did some experiments on how the APK works. For close distances, for example 0 meters, the results are usually ok. That’s logical, as everything below the minimum range will be “0 meters”. From 1 to 4 meters results are usually good, with an approximate error of 33% (From 3 measurements, one will be wrong approx.). For longer distances error goes higher, but usually gives a quite good distance (for example, for 7 meters the APK can say 6 or 8, that it’s not a bad result.

From the previously explained, we can say that our project have achieved the expected result, although the problems that working with RSSI gave us.
Bibliography:


[13] Pic. 1.1, located in “Google Images”

[14] Pic.1.2, located in “Google Images”

[15] Pic.2.1.1 located in “Google Images”

[16&17] Tables 2.3.1 and 2.3.2, located in www.wikipedia.org

[18] Pic. 3.3.1, located in “Google Images”
APPENDIX:

In the rar. File attached to the thesis, there are the following data:
- Source code
- Layouts
- Excel files with the experiment results
- APK ready to install
- Project file to be charged in Eclipse or Netbeans.