UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Smartphone Relative Positioning using Phone Sensors

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Nèstor Bonjorn López

In partial fulfilment

of the requirements for the degree in

BACHELOR'S DEGREE IN TELECOMMUNICATIONS SYSTEMS ENGINEERING

Advisor: Dr. Ilker Seyfettin Demirkol

Barcelona, June 2016

Abstract

A relative positioning system that enables two Android smartphone pedestrian users to find each other without the need of GPS availability is presented. Moreover, the smartphone user will also be able to track his movement in real time without the GPS constraint. The methods proposed involve both radio and acoustic waves emission and reception, as well as the mastery of several embedded phone sensors usage. Obtained results show difficulties to accomplish the main objective of the project, due to the radio waves power fluctuation in urban environments. However, the self-movement tracking and a high-accuracy ranging system for short distances based on acoustic signals emission work properly.

Resum

En aquesta tesi es proposa un sistema de posicionament relatiu que permet a dos usuaris d'smartphones Android que vagin a peu trobar-se l'un a l'altre sense la necessitat de cobertura GPS. A més a més, l'usuari també podrà seguir el seu moviment en temps real sense la limitació del GPS. Els mètodes proposats involucren l'emissió i la recepció tant d'ones electromagnètiques com acústiques, a més d'un clar domini dels sensors del mòbil. Els resultats obtinguts demostren dificultats per assolir el principal objectiu del projecte, degut a la fluctuació de la potència de les ones electromagnètiques en ambients urbans. Tot i això, la monitorització de la pròpia trajectòria i un acurat sistema de mesura de distàncies curtes basat en l'emissió de senyals acústics funcionen adequadament.

Resumen

En esta tesis se propone un sistema de posicionamiento relativo que permite a dos usuarios de smartphones Android que vayan a pie encontrarse el uno al otro sin la necesidad de cobertura GPS. Además, el usuario también podrá monitorizar su movimiento en tiempo real sin la limitación del GPS. Los métodos propuestos involucran la emisión y recepción tanto de ondas electromagnéticas como acústicas, además de un claro domino de los sensores del móvil. Los resultados obtenidos demuestran dificultades para lograr el principal objetivo del proyecto, debido a la fluctuación de la potencia de las ondas electromagnéticas en ambientes urbanos. Aún así, la monitorización de la propia trayectoria y un preciso sistema de medida de distancias cortas basado en la emisión de señales acústicas funcionan adecuadamente.

Acknowledgements

I would like to thank Dr. Ilker Demirkol for believing, from the very beginning, in my capabilities to carry out this thesis. Moreover, his advices and supervision during the whole project have been of great help.

I would also like to thank Guillermo Ortas for being a great co-developer of this collaborative project.

Finally, I would also like to thank Anna and my parents for letting me borrow their smartphones for my experiments, as well as for their unconditional support.

Revision history and approval record

Revision	Date	Purpose
0	15/05/2016	Document creation
1	23/06/2016	Document revision
2	25/06/2016	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Nèstor Bonjorn	nestorbonjorn@gmail.com
Ilker Demirkol	ilker.demirkol@entel.upc.edu

Written by:		Reviewed and approved by:	
Date	20/06/2016	Date	23/06/2016
Name	Nèstor Bonjorn	Name	Ilker Demirkol
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

1.1. Statement of purpose

The main objective of this project is to create an Android application that allows any Android smartphone user to find another smartphone (or any other device supporting Wi-Fi Direct[1] communication) without the need of Global Positioning System (GPS) availability or cellular network coverage. This enables its usage even in challenging scenarios, such as indoor or isolated ones.

In order to find each other, a relative positioning system approach has been proposed. This approach makes use of self-movement tracking and distance estimations based on radio waves' received power. However, for close distances, the received signal power-based estimations are substituted for a high-accuracy ranging approach based on the time of flight (ToF) of acoustic waves.

Moreover, the ToF-based approach enables different use cases of the application, such as the measurement of small objects' sizes.

1.2. Requirements and specifications

The Android application must give information to its user about the direction of a remote Wi-Fi Direct device relative to himself, as well as distance estimations between them. Both devices are required to have the Wi-Fi Direct connectivity available –which does not mean an established connection–; once this is done, the user must choose the device that he intends to find, according to a list of all the available devices' names. Moreover, in order to get and update the distance and direction measurements, the user must walk carrying his smartphone pointing towards his direction of displacement.

The close-proximity ranging approach, i.e. the acoustic waves-based one, does require an established Wi-Fi Direct connection, so in this case, the other device has to accept the connection. However, the received signal power-based approach does not require any connection; therefore, it works even without the acceptance of the other device.

The methods used in order to find the remote device work for distances lower than 250 meters, while the high-accuracy ranging approach for close distances works up to 1.3 meters.

Finally, the application also permits the user to track his movement in real time regardless of any connectivity.

1.3. Project background

This thesis has been performed within a collaborative project (called RADIUS) that involves two people: Guillermo Ortas and myself. However, all the methods and procedures stated in this report are from my own work.

This project is not a continuation of any previous one, thus it starts from the scratch. However, the main project initial ideas were provided by the supervisor Dr. Ilker Demirkol. From these initial ideas along with some research inputs we have developed the whole project.

[1] Wi-Fi standard that enables devices to connect with each other without requiring a wireless access point.

The work stated in this report follows three main methods: pedestrian dead reckoning, relative positioning and acoustic distance, explained in the sections 3, 4 and 5 of this report, respectively. Both the former and the latter are based on several journal papers ideas but following original algorithms and code, since they were not available in the consulted data. Regarding the relative positioning approach, as well as the integration of all the methods in a pleasant Android application, a completely original work is performed.

1.4. Work Plan, milestones and Gantt Diagram

Project: Radius	Responsible: Both	WP ref: 1
Major constituent: Research		Sheet 1 of 9
Short description: Research on similar projects and mobile apps.		Planned start date: 08/02/2016 Planned end date: 23/05/2016
		Start event: Project start End event: -
Internal task T1: Paper research about the usage of smartphones' sensors in other projects	**Internal task T2:** App research and evaluation	

Project: Radius	Responsible: Both	WP ref: 2
Major constituent: Develop Test App		Sheet 2 of 9
Short description: Test app development block. We develop an app that reads all of the smartphone's sensors.		Planned start date: 15/02/2016 Planned end date: 29/02/2016
		Start event: Project start End event: -
Internal task T1: Environment installation: Android Studio **Internal task T2:** Interface design **Internal task T3:** Android Language learning	**Internal task T4:** Code writing **Internal task T5:** Testing	

Project: Radius	Responsible: **Nèstor**	WP ref: 3
Major constituent: User movement monitoring		Sheet 3 of 9
Short description: Monitor the user's movement in order to know the source of the changes in the distance estimations between two smartphones (i.e. how much has the user moved between two distance estimations). This allows calculating the relative direction, and hence the relative position, between the two devices.		Planned start date: 22/02/2016 Planned end date: 01/04/2016
		Start event: Test app developing End event: -
Internal task T1: Parameter acquisition **Internal task T2:** Orientation calculation	**Internal task T4:** Simulation **Internal task T5:** Testing	

Internal task T3: Pedestrian Dead Reckoning	

Project: Radius	Responsible: **Guillermo**	WP ref: 4
Major constituent: Wi-Fi Direct		Sheet 4 of 9
Short description: Design a communications protocol between two devices for the Wi-Fi Direct connectivity		Planned start date: 29/02/2016 Planned end date: 20/05/2016
		Start event: Test app developing End event: Project end
Internal task T1: Research on papers and projects that have already used this connectivity **Internal task T2:** Design communications protocol	**Internal task T3:** Testing **Internal task T4:** Chat implementation	

Project: Radius	Responsible: **Nèstor**	WP ref: 5
Major constituent: Kalman Filter		Sheet 5 of 9
Short description: Implement the Kalman Filter (signal processing) to improve the computed estimations.		Planned start date: 26/03/2016 Planned end date: 09/05/2016
		Start event: - End event: -
Internal task T1: Research on papers and projects that have already used this filter for other applications **Internal task T2:** Simulation	**Internal task T3:** Design and developing **Internal task T4:** Testing	

Project: Radius	Responsible: **Nèstor** (sound ToF) **Guillermo** (RSSI, Bluetooth)	WP ref: 6
Major constituent: Distance estimation		Sheet 6 of 9
Short description: Estimate the distance between two devices using different systems.		Planned start date: 26/03/2016 Planned end date: 03/06/2016
		Start event: - End event: -
Internal task T1: Wi-Fi Direct RSSI **Internal task T2:** Bluetooth RSSI	**Internal task T3:** Audio ToF	

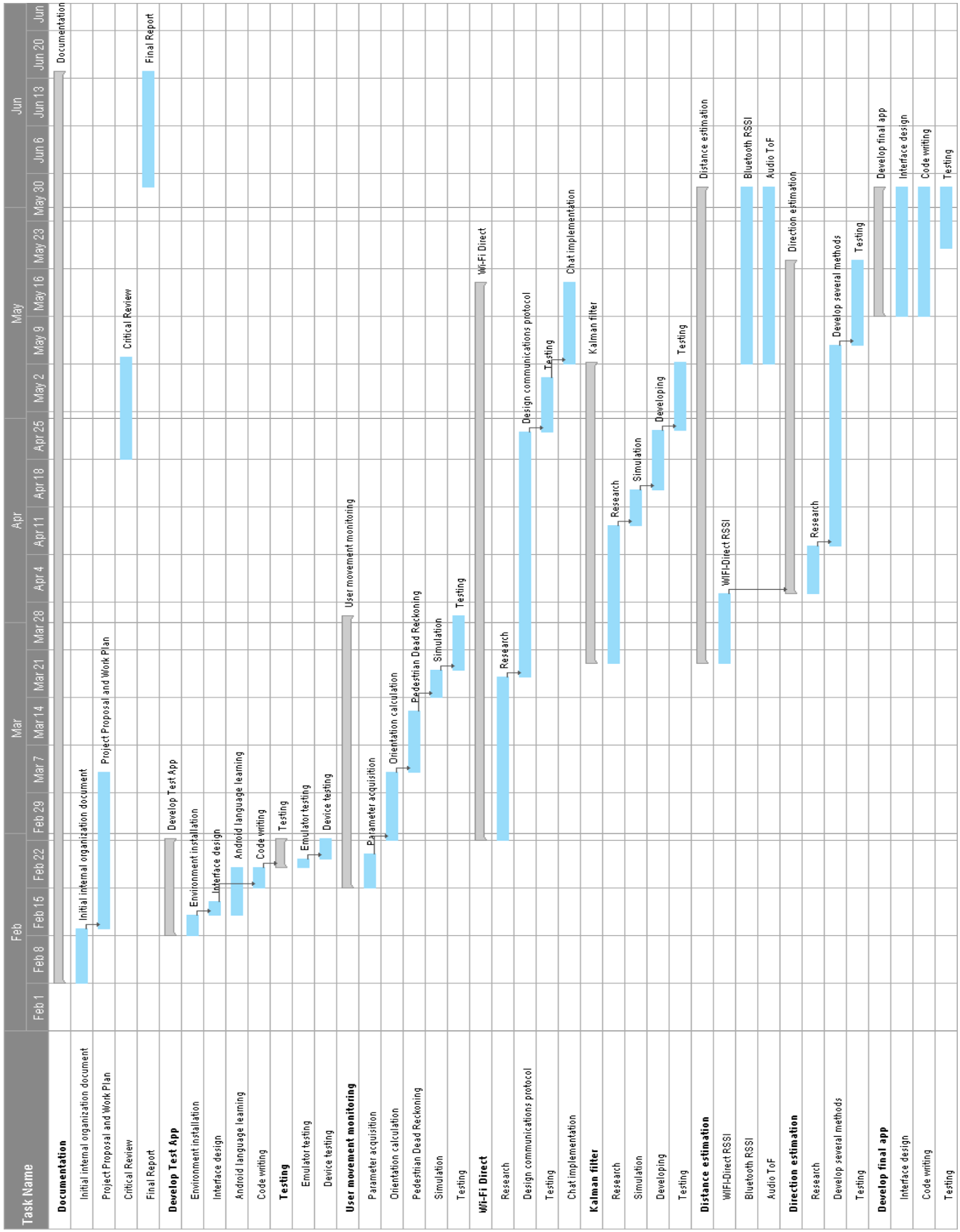
Project: Radius	Responsible: **Nèstor**	WP ref: 7
Major constituent: Direction estimation		Sheet 7 of 9

Short description: Estimate the relative direction between two devices using different approaches.	Planned start date: 05/04/2016 Planned end date: 24/05/2016 Start event: Distance estimation End event: -
Internal task T1: Research **Internal task T2:** Develop several methods	**Internal task T3:** Testing

Project: Radius	Responsible: **Guillermo**	WP ref: 8
Major constituent: Bluetooth		Sheet 8 of 9
Short description: Design a communications protocol between two devices using Bluetooth. The objective is to get RSSI values with a higher rate to get a better average and thus, a better distance estimation.		Planned start date: 10/05/2016 Planned end date: 13/06/2016 Start event: Project Critical Review delivery End event: Project end
Internal task T1: Research on papers and projects that have already used this connectivity **Internal task T2:** Design communications protocol		**Internal task T3:** Testing

Project: Radius	Responsible: Both	WP ref: 9
Major constituent: Develop final app		Sheet 9 of 9
Short description: Final app development block: it includes all previous calculations		Planned start date: 10/05/2016 Planned end date: 13/06/2016 Start event: - End event: Project end
Internal task T1: Interface design **Internal task T2:** Code writing		**Internal task T3:** Testing

WP#	Task#	Short title	Milestone / deliverable	Date (week)
3	4	Test Android App	Basic sensor reading Android app	26/02/2016
4	2	Wi-Fi Direct protocol	Wi-Fi Direct protocol design	29/04/2016
5	2	Filter design	Kalman Filter implementation	29/04/2016
6	1	RSSI distance	RSSI distance estimation	05/04/2016
-	-	CDR	Critical design review	09/05/2016
7	-	Direction	Direction estimation	16/05/2016
4	4	Chat	Wi-Fi-Direct chat implementation	20/05/2016
6	3	Sound distance	Sound distance estimation	03/06/2016
8	2	Android App	Final Android app	13/06/2016
-	-	FR	Final report	27/06/2016


Task Name
Feb
Mar
Apr
May
Jun
Feb 1
Feb 8
Feb 15
Feb 22
Feb 29
Mar 7
Mar 14
Mar 21
Mar 28
Apr 4
Apr 11
Apr 18
Apr 25
May 2
May 9
May 16
May 23
May 30
Jun 6
Jun 13
Jun 20
Jun
1 Documentation
2 Initial internal organization document
3 Project Proposal and Work Plan
4 Critical Review
5 Final Report
6 Develop Test App
7 Environment installation
8 Interface design
9 Android language learning
10 Code writing
11 Testing
12 Emulator testing
13 Device testing
14 User movement monitoring
15 Parameter acquisition
16 Orientation calculation
17 Pedestrian Dead Reckoning
18 Simulation
19 Testing
20 Wi-Fi Direct
21 Research
22 Design communications protocol
23 Testing
24 Chat implementation
25 Kalman filter
26 Research
27 Simulation
28 Developing
29 Testing
30 Distance estimation
31 WIFI-Direct RSSI
32 Bluetooth RSSI
33 Audio ToF
34 Direction estimation
35 Research
36 Develop several methods
37 Testing
38 Develop final app
39 Interface design
40 Code writing
41 Testing

1.5. Incidences

The most remarkable incidence during the development of the project has been the impossibility of achieving a high rate of received signal strength indicator (RSSI) measurements. With this high rate, it would be possible to consider the average of some samples instead of a single one before processing it to obtain a distance estimation; consequently, the RSSI measurement's error would diminish. However, the best rate achieved is about one measurement per second; thus, the use of averaging would reduce the dynamism of the application.

Without this RSSI averaging, the distance estimations are really inaccurate, so the initial idea of exactly locating a remote device turns into approximated estimations that allow the smartphone user to find the remote device by updating the estimations while walking.

Regarding the user movement monitoring, one of the methods initially proposed also had to be discarded. We thought that by integrating twice the accelerometer values of the 3-axis accelerometer embedded in smartphones, we could obtain a good monitoring of the user movement. However, doing some research, we soon realised that this would accumulate too much measurement error and would become unfeasible in large distances. Therefore, we opted for a pedestrian dead reckoning based on algorithms for both step detection and step length estimation, as well as some orientation methods.

2. State of the art of the technology used or applied in this thesis

Currently, the problem of the smartphone's relative distance or location is not too well resolved. It is true that there are plenty of mobile applications that try to fix the position of a remote smartphone from your own device but they either lack precision or just do not work in many challenging situations. One of the most famous "Friend locator application" is *Find My Friends*, which is only useful for situations where you and your friend have a good GPS signal. Likewise, in the current market, we can find other similar applications that also have the GPS availability and precision limitations already mentioned.

On the other hand, there are a few applications ensuring that they work without GPS or Internet like *Friends Tracker*. The truth is that this app in particular requires SMS coverage and cellular network accessibility (e.g., travelling abroad without roaming would hinder this option), so it also has availability restrictions. Moreover, it is not able to reach a better precision than GPS and its interface is both unpleasant and difficult to handle.

In the application that concerns this thesis, a true GPS-less phone locator that works in challenging situations within a range of 250 meters is developed.

3. Pedestrian Dead Reckoning

In the last few decades, the human movement monitoring has been a popular topic of research. *A priori*, there are several methods to achieve this tracking but not all of them work equally. For example, the GPS tracking could be useful in some outdoor scenarios but never in indoor ones such as in shopping malls or office buildings. It would neither work in some urban environments, e.g., among buildings with very high altitudes blocking line-of-sight links to GPS satellites. In these scenarios, the multipath propagation would severely affect the position estimations, making it impossible to reach a great precision in short distances. However, there are other technologies that allow both an outdoor and indoor tracking that are based on sensors measurements from the device itself.

The 3-axis accelerometers embedded in current mobile devices could allow for a user positioning monitoring based on a double integration of the accelerometer readings. Nevertheless, the smartphones' accelerometers are not as accurate as needed in this double integration method, so the measurement errors would be too difficult to handle.

On the other hand, although it is only useful for a user that keeps walking, the pedestrian dead reckoning method (PDR) is the best way to track the user walking movement, which is the scenario that concerns this thesis. This method does not need such a measurement precision as the previously mentioned one and works properly in most environments.

In this chapter, the methodologies that involve the PDR method are presented.

3.1. Fundamentals

The PDR is based on three particular methods: step detection, step size estimation and heading calculation. The combination of these three methods will allow the user to track his movement in real time.

The relative position of the user p_k is updated every time a new step is detected. When this happens, the application estimates a step size l_k for this particular k^{th} step, and looks for the current device heading θ_k. After this, it simply applies

$$p_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix} = \begin{bmatrix} x_{k-1} \\ y_{k-1} \end{bmatrix} + l_k * \begin{bmatrix} \sin\theta_k \\ \cos\theta_k \end{bmatrix} \tag{3.1}$$

in order to obtain the new relative position, taking into account that the heading is always relative to magnetic North and the positive x and y axes point to East and North respectively.

To acquire a good performance, the user will have to hold his phone pointing towards his direction of displacement.

3.2. Step detection

3.2.1. Accelerometer

3.2.1.1. Reading accelerometer values

This project is made in a smartphone context; specifically, in an Android environment. The software used is Android Studio, which is the official integrated development environment

(IDE) for Android app development, based on IntelliJ IDEA. In this context, it is not difficult to obtain sensor readings from the embedded hardware of a smartphone.

First of all, it is necessary to get an instance of the *SensorManager* Java class, by calling `Context.getSystemService()` with the argument *SENSOR_SERVICE*. After that, a *SensorEventListener* for the desired sensor and at the desired frequency rate has to be registered. Then, whenever new sensor data is available, the program calls the method *onSensorChanged* for the specific *SensorEvent*[2]. Finally, overriding this method, the sensor values can be obtained.

In the case of the accelerometer, three values are obtained for each *SensorEvent* instance, which belong to three different axes relative to the smartphone body's coordinates shown in *Figure 1*. Moreover, the Android API can also directly provide a linear acceleration, that is, a software-based sensor that provides the acceleration along each device axis excluding gravity.

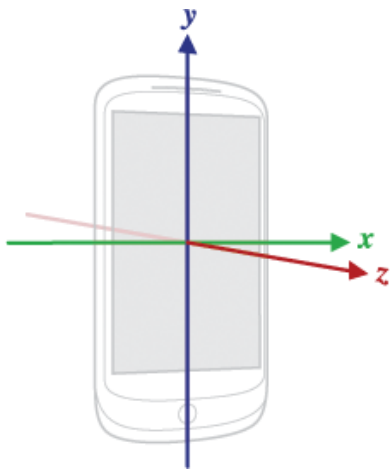


Figure 1. Coordinate system (relative to a device) that's used by the Android Sensor API [1].

3.2.1.2. Global coordinate system acceleration

The accelerometer readings in the device's coordinate system, however, are not useful enough for the step detection method because they depend on how the user carries the device. In order to get a robust method that works in every situation, a global coordinate system (GCS) acceleration must be achieved. This means that, whatever the device's inclination, the z axis will always be pointing towards the sky and perpendicular to the ground, while x and y axes point to East and North respectively.

These GCS acceleration measurements cannot be obtained directly from any of the Android sensors, so the methodology proposed includes the acquisition of a rotation matrix R that represents the rotation of the device with respect to the GCS. One of the properties of this specific matrix is that its inverse allows the measurements for a coordinate system conversion from the device coordinate system to the GCS, following the equations (3.2) and (3.3):

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R * \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.2}$$

[2] *SensorEvent* is a Java class that represents a sensor event and holds information about it

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R^{-1} * \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}, \tag{3.3}$$

where $\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$ is a vector of some measures in local coordinate system, and $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ the same in GCS.

The *SensorManager* class provides a method called *getRotationMatrix* that computes the rotation matrix, as well as an inclination matrix, based on accelerometer and magnetometer data. Thus, after the acquisition of both the accelerometer and magnetometer sensor measurements, the rotation matrix will be obtained by calling the *getRotationMatrix* method with the specified parameters.

After that, the methods from the *Matrix* Java class will be used to obtain the inverse of the rotation matrix, as well as to multiply this matrix by the desired vector in order to perform the coordinates' conversion, as shown in equation (3.3). The linear acceleration vector will be directly chosen in order to perform the conversion. This yields to three axis acceleration measurements relative to the Earth and without the influence of gravity. In this case, the new z axis values tracking will be enough to detect the user steps as will be explained in the next section.

3.2.2. Peak detection

3.2.2.1. Low-pass filtering

When a person walks, the impact of the foot with the ground causes a certain variance of the device's acceleration. In this way, the acceleration measurements are characterized by the reach of high and low peaks on each step. However, the values obtained from the embedded accelerometer of a smartphone are very noisy and this causes unexpected peaks.

In order to reduce the noisy values obtained, a low-pass filtering is applied. The filter consists on a simply sampling average that smooths the noisy acceleration measurements. This allows an easier step detection based on high and low peak detection. The equation (3.4) represents the low-pass filtering, where x_k represents the noisy measurements, and y_k the average-filtered output.

$$y_k = \frac{1}{8}\sum_{k-7}^{k} x_k \tag{3.4}$$

3.2.2.2. Local maximums and minimums algorithm

Now, a method for peak detection is required. It is very easy to find an absolute maximum or minimum given an array of values, but it is not so easy to find local maximums and minimums. The algorithm proposed accomplishes this objective in real-time, i.e. as the sensor values arrive, and is explained below, as well as showed in the appendices of this document.

The algorithm is divided regarding the search of a maximum or a minimum. After finding one of them, the algorithm keeps searching the other. Starting with a maximum searching, when a

sensor measurement has been found, the algorithm compares if the actual value is higher than the highest value found. In this case the maximum value is updated, as well as its index, i.e. its number of sample. The process of the maximum updating continues until the current value is lower than the maximum found minus a certain threshold. In this precise instant, the last maximum obtained is considered as one local maximum, i.e. a high peak has been detected. Then, a minimum searching process begins in an analogous way. However, before that happens, it resets the value of the minimum value to the current value.

In this case, the algorithm considers a minimum value as one of the local minimums when the current value is higher than the minimum value found plus the threshold mentioned before. After that, it resets the maximum value to the current value.

The algorithm will keep working while the sensor values keep arriving and it will consider that a step has been done when a pair of one local maximum and minimum has been found.

3.3. Step size estimation

3.3.1. Static method

Some PDR methods use a static step size for the updating of the position in each step. In this case, the constant K that represents the size is set experimentally, so it is independent of the measures obtained while walking. In a simple way, this value can be determined equally for every user (3.5):

$$l_k = K \tag{3.5}$$

In a more sophisticated way, it can change depending on the user's height (3.6):

$$l_k = K' * height\ . \tag{3.6}$$

For both methods, if the person changes its velocity during his walking, the step size remains constant. Therefore, the static methods are not accurate enough.

3.3.2. Dynamic method

A dynamic step length method tries to avoid this situation. This means that the current step length will be estimated according to the acceleration measurements represented by A in equation (3.7):

$$l_k = f(A) \tag{3.7}$$

There are not perfect mathematic formulas that determine the step length of a person according to the acceleration measurements obtained from a device that he carries. However, there are good approximations, although they also depend on experimental constants.

Several papers have proposed different dynamic methods in order to estimate the step length of a person as accurately as possible. The model proposed by [2] and validated by [3] is used in this project. In this way, the step size will be determined through the following equation:

$$l_k = K * \sqrt[4]{a_k^{max} - a_k^{min}} \tag{3.8}$$

where K is an experimental constant set to 0.52, and a_k^{max} and a_k^{min} are the maximum and minimum acceleration values, respectively, that represent the kth step.

3.4. Device orientation calculation

A priori, there are two methods to estimate the orientation and inclination of a smartphone. Through the Android API, we can directly obtain the orientation from the accelerometer and magnetometer sensors. Another way is to make use of the gyroscope sensor measurements, which indicate the velocity of rotation of the device in three local axes.

However, the best way is to make a combination of these two methods in order to get information with less uncertainty than if the sources had been used individually. This concept is called Sensor Fusion.

3.4.1. Accelerometer and magnetometer

The straightest way to obtain the orientation of an Android smartphone is by means of the *getOrientation* method from the *SensorManager* class. The parameter needed to use the *getOrientation* method is the rotation matrix obtained with the *getRotationMatrix* method, with the accelerometer and magnetometer sensor values, as explained in the section 3.2.1.2 of this report.

The result is a three-dimensional vector containing the values of azimuth, pitch and roll in GCS. All three angles are positive in the counter-clockwise direction and are shown in *Figure 2*, as well as described below:

- Azimuth: rotation around the $-z$ axis, i.e. the opposite direction of z axis.
- Pitch: rotation around the $-x$ axis, i.e. the opposite direction of x axis.
- Roll: rotation around the y axis.

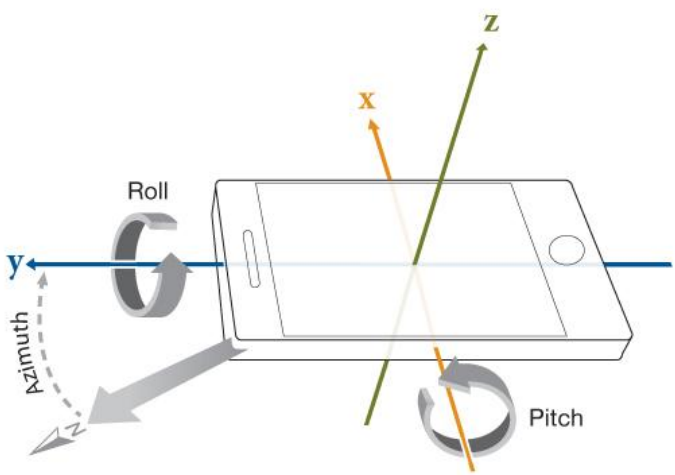


Figure 2. Smartphone's azimuth, pitch and roll angles [4].

3.4.2. Gyroscope values integration

Actually, it is impossible to obtain an orientation in GCS with only the gyroscope sensor. What can be measured by this sensor are changes of orientation, i.e. a relative orientation.

The gyroscope sensor's data do not represent angles of rotation, but they represent the rotation speed of the device in the three device's coordinates. However, integrating the data acquired, a rotation change can be obtained, although the accomplishment of this method is not trivial in Android platform. The method followed in this work is based on the Android Developers page proposed code in [5], and will be explained in the following lines:

In order to perform the discrete integral, every angular speed acquired from the gyroscope has to be multiplied by its corresponding delta time, obtaining a delta rotation. Both the angular speed and the delta time are obtained from the *SensorEvent* class information. The former is directly obtained from the sensor's data of every gyroscope *SensorEvent*; however, the latter cannot be directly obtained.

Firstly, the event time-stamp, i.e. the time when a specific event takes place, is obtained. In this way, the program stores the timestamp of each gyroscope event and then, by subtracting the last gyroscope event timestamp, the delta time for each event is acquired. Since the timestamps are stored in nanoseconds, a conversion from nanoseconds to seconds must be done before proceeding.

Now, it is time to express the rotation change; however, there are many ways to do that, although not all of them work properly in every situation.

One of the most famous methods is the Euler angles, which represent a rotation as a sequence of three elemental rotations starting from a known orientation. These elemental rotations are the rotations in three different axes, typically denoted as yaw, pitch and roll. However, the pitch, i.e. the rotation around the x axis, cannot go up to 90 degrees in this method because, in this situation, the definitions of yaw and roll become ambiguous. Therefore, if an application without any constraint is needed, it may be necessary to change the Euler angles definitions on the fly, which becomes too complex. Still, if you constraint your application, Euler angles work fine.

Another way to express the rotation angles is within a rotation matrix. This matrix contains nine numbers that fully represent the rotation of a three-dimensional object.

Finally, there is other method based on quaternion vectors. This four-element vector can represent every rotation of an object in a three-dimensional Euclidean space with only four numbers and without any constraints.

$$q = \begin{bmatrix} x * \sin\left(\frac{\theta}{2}\right) \\ y * \sin\left(\frac{\theta}{2}\right) \\ \mathrm{z} * \sin\left(\frac{\theta}{2}\right) \\ \cos\left(\frac{\theta}{2}\right) \end{bmatrix} \tag{3.9}$$

In equation (3.9), the definition of a quaternion that represents a rotation is shown, where x, y and z represent the normalized axes where the rotation has taken place, and θ represents the rotation magnitude, as shown in *Figure 3*.

The method proposed in this thesis makes use of a quaternion vector in order to represent every delta rotation, by then transforming it into a rotation matrix, applying the rotation to the current orientation, and finally converting the new orientation into azimuth, pitch and roll angles. All this is further explained in the following lines:

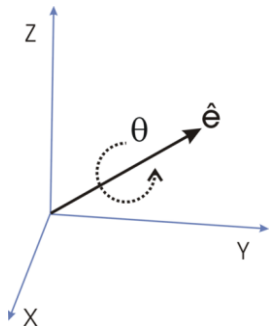


Figure 3. Quaternion rotation representation [6].

First of all, the speed magnitude $|v|$ is calculated within the equation (3.10)

$$|v| = \sqrt{{v_{yaw}}^2 + {v_{pitch}}^2 + {v_{roll}}^2} \tag{3.10}$$

where v_{yaw}, v_{pitch} and v_{roll} are the speed rotation values directly obtained from the gyroscope sensor's data.

Now, the magnitude of the rotation angle θ can be obtained following the equation (3.11)

$$\theta = |v| * \Delta t \tag{3.11}$$

where Δt is the delta time mentioned before, i.e. the time passed between two gyroscope event timestamps.

The x, y and z parameters from equation (3.9) can be obtained by normalizing the gyroscope sensor's data, i.e.:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{|v|} \begin{bmatrix} v_{yaw} \\ v_{pitch} \\ v_{roll} \end{bmatrix} \tag{3.12}$$

Now, both the rotation normalized axes and the rotation magnitude are obtained, so the quaternion is already computable. Thus, a rotation matrix can be calculated from it through the *getRotationMatrixFromVector* method from the *SensorManager* class, which can transform a rotation vector, i.e. the quaternion obtained, into a rotation matrix of nine elements. This rotation matrix stores the delta rotation performed in each gyroscope event so it will be called *gyroDeltaMatrix* (ΔG in equation (3.13)).

The multiplication of two rotation matrices yields another rotation matrix whose application to a point effects the same rotation as the sequential application of the two original rotation matrices [7]. So, in order to obtain a rotation matrix that represents the current device orientation, the *gyroDeltaMatrix* has to be multiplied by the rotation matrix that represents the last orientation (or rotation). This last matrix will be called *gyroMatrix* (G in equation (3.13)).

$$[G_k] = [G_{k-1}][\Delta G_k] \tag{3.13}$$

In order to do that, a *MatrixMultiplication* method is created and the operation order shown in (3.13) must be followed. However, the first time that we access to the gyroscope sensor event the *gyroMatrix* does not exist yet, thus, it should be initialized. For this reason, the rotation matrix obtained from the accelerometer and magnetometer sensors is used as the initial *gyroMatrix* G_0.

Finally, after all the calculations have been done, the *gyroMatrix* has to be transformed into azimuth, pitch and roll angles, and this is made by the *getOrientation* method from the Android API, as it is done for the accelerometer and magnetometer orientation; however, in this case, the parameter passed to the method will be the *gyroMatrix*.

3.4.3. Sensor Fusion

3.4.3.1. Qualitative explanation

Both accelerometer/magnetometer and gyroscope orientation are reached. In order to get benefit from both methods, a sensor fusion method has to be performed.

The downside of the accelerometer/magnetometer orientation is that their data are very noisy, especially the output from the magnetic field sensor. Moreover, if there is an abrupt change in the device orientation, some spikes appear in the data captured by these sensors. The application of a low-pass filtering would reduce the noise and the spikes but, at the same time, would decrease the dynamic response of the output, i.e. the fast changes in orientation would be slowly captured by the output of the filter.

On the other hand, the gyroscope sensor is far more accurate and has good dynamic response; i.e. very short response time. However, its coordinate system reference is always local, which means that it needs an external source to situate the obtained gyroscope's orientation in the world's coordinate system.

Another drawback of this method is a popular phenomenon called drift. The origin of this is the integration performed to obtain the computed rotations, which leads the integrated gyroscope data to accumulate white noise during the reading. Consequently, the gyroscope-based orientation begins to drift, i.e. to move away from the correct value, although the device is not moving.

However, the accelerometer/magnetometer orientation has a stable output because no integration is performed, so it does not suffer any drift. Therefore, the output from the sensor fusion needed would be mostly gyroscope data but with the accelerometer/magnetometer usage to correct drift, as well as to set up a GCS for the orientation.

Figure 4 represents the sensor data fusion that will be followed in this thesis, via a complementary filter, i.e. a filter that manages both high-pass and low-pass filters simultaneously.

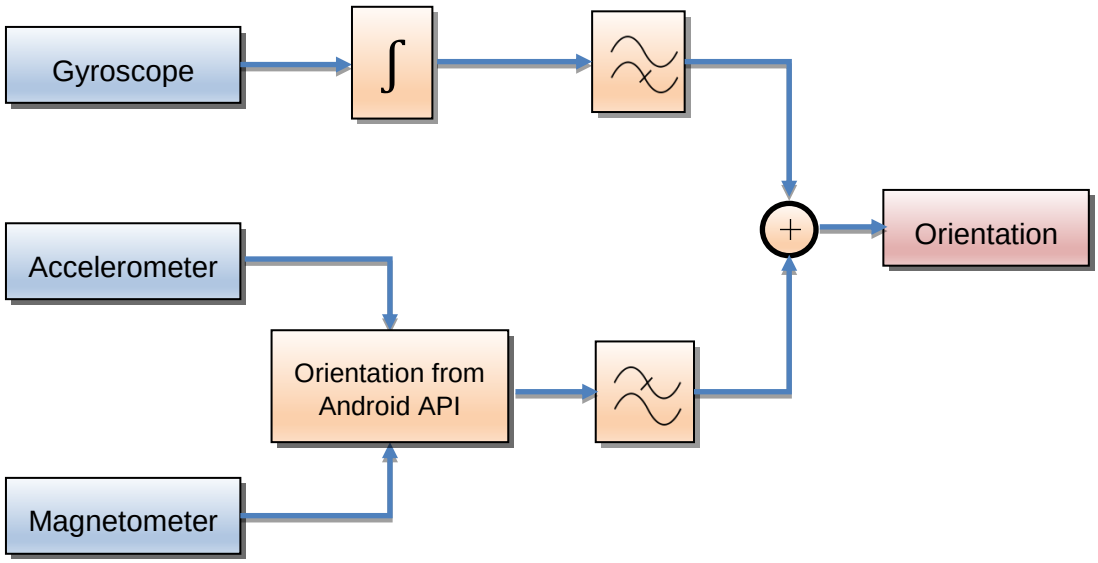


Figure 4. Block diagram of the Sensor Fusion method via complementary Filter

3.4.3.2. Simplified implementation

This Sensor Fusion method is implemented in the following way:

Firstly, a summary of the section 3.4.2 is represented in the following simplified Java code line:

```
gyroOrient = gyroOrient + dT * gyroRotation;
```

where `gyroOrient` is the orientation calculated from the gyroscope sensor, `gyroRotation` the speed vector acquired from the gyroscope, and `dT` the time elapsed between the last and the current gyroscope event.

The low-pass filtering of the accelerometer/magnetometer orientation is represented in the following Java code line:

```
accMagOrient = k * accMagOrient + (1 - k) * newAccMagOrient;
```

where `accMagOrient` is the orientation estimation calculated from the accelerometer and magnetometer sensors, `newAccMagOrient` the new orientation measure from these same sensors, and `k` a coefficient that controls how slowly the new orientation measures are introduced in the current orientation estimation.

The term `k * accMagOrient` can be seen as the high-pass component of the filter, although in the above code this definition does not make sense. However, replacing this term by a weighted output from the gyroscope orientation leads to high-pass filtering the gyroscope data.

Therefore, the code line that will represent the complementary filter and that will be executed in a loop is:

```
orientation = k * gyroOrient + (1 - k) * newAccMagOrient;
```

In order to complete the Sensor Fusion proposed, the gyroscope orientation has to be overwritten by the output of the complementary filter every time the above line is executed. This leads to correct the gyroscope drift mentioned before.

In this way, the following two simplified code lines represents the full proposed Sensor Fusion method:

```
orientation = k * gyroOrient + (1 - k) * newAccMagOrient;
gyroOrient = orientation;
```

The constant k defines the time constant τ of the filter in the following way:

$$\tau = \frac{k * dt}{1 - k} \tag{3.14}$$

where dt is the operation time interval; i.e. the time elapsed between two executions of the above code lines.

τ sets the time from which the low-pass component signal will be considered. Likewise, only the high-pass component signals shorter than this time constant will affect the output.

Finally, the constant k can be set according to the desired constant τ, as shown in equation (3.15):

$$k = \frac{\tau}{\tau + dt} \tag{3.15}$$

3.4.3.3. Gyroscope overwriting

The code lines about the complementary filter written above work well with the correct syntax in Java platform and do not require further explanation, except for one of them; the one used to overwrite the gyroscope orientation from the output of the complementary filter.

As it is explained in section 3.4.2, the gyroscope delta rotations are performed by rotation matrices multiplication, so the orientation vector from the sensor fusion output has to be converted into a rotation matrix. This can be made following the definitions of coordinate rotations [3] from [7]. Regarding every axis separately, the three coordinate rotations are represented in the following way:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$
$$R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & -\cos\beta \end{bmatrix} \tag{3.16}$$
$$R_z = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & -\cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[3] A coordinate rotation is a rotation about a single coordinate axis

where α, β and γ are pitch, roll and yaw (or azimuth) angles respectively.

A single rotation matrix that can represent three rotations in the three Euclidean angles can be obtained by multiplying the above three matrices in the correct order as shown in the following equation:

$$R = R_z R_x R_y \tag{3.17}$$

This yields to the general rotation matrix that will finally represent the orientation output:

$$R = \begin{bmatrix} \cos\gamma\cos\beta - \sin\alpha\sin\beta\sin\gamma & \sin\gamma\cos\alpha & \cos\gamma\sin\beta + \sin\gamma\sin\alpha\cos\beta \\ -\cos\beta\sin\gamma - \sin\alpha\sin\beta\cos\gamma & -\cos\gamma\sin\alpha & \cos\gamma\sin\alpha\cos\beta - \sin\gamma\sin\beta \\ -\cos\alpha\sin\beta & -\sin\alpha & \cos\alpha\cos\beta \end{bmatrix} \tag{3.18}$$

3.5. Position updating

In order to update the new position some issues have to be considered.

In the Java code three *timerTasks*[4] are created so as to perform the PDR method. In this case, the three of them will be executed every 50 ms.

The first one is in charge of calculating the acceleration in GCS in the three axes, and will be firstly executed one second after the creation of the application.

The second one is in charge of performing the sensor fusion method between the accelerometer/magnetometer-based and gyroscope-based orientation, obtaining the fused orientation. It is executed at the same time than the previous one.

Finally, the last one performs the position updating by means of both the GCS acceleration and the fused orientation obtained from the other two timers. In this case, this timer will be executed 40 ms after the others for the first time, in order to leave time for the previous calculations.

The third timer will also be the one in charge of detecting the user steps from the accelerometer measurements, and only after a step is detected the current fused orientation will be used in the position updating, following the equation (3.1).

This timer also creates a graphic interface where the user will be able to visualize his pedestrian movements in real-time. The interface is based on the *androidplot* library and will consist on a 2-D dynamic plot. As new positions are calculated, the plot will update its appearance, showing the current position as well as the previous ones. A blue dot will represent the calculated positions, and a yellow line will join these dots. Since this work is based on relative positions, the first position will be considered as the [0, 0] point, which will be in the centre of the plot, and the others will be relative to that.

Finally, the y and x axes of the plot will always be in meters and will always represent the magnetic North and East, respectively.

[4] The *timerTask* is a Java class that represents a task to run at a specified time.

4. Relative position estimation

The main purpose of this project –and specifically of the methods explained in this section– is to determine the relative position between two smartphone devices, considering that one is moving (henceforth A) while the other one remains static (henceforth B).

In order to design a successful method, two different approaches are previously required: A's movement tracking and the relative distance measurement between A and B. Integrating these two concepts, i.e. assigning different distances measurements to different positions of A, a relative position between the two devices can be estimated.

The theory claims that with three distances measured from three different positions that are not in the same straight line, a relative position can be directly obtained. This process is called trilateration, and makes use of the geometry of circles and triangles. In practice, however, this is not so easy, due to all the possible measurement errors.

The user's movement tracking, which is performed by means of PDR, allows the estimation of your current position with respect to your previous ones quite accurately, at least for short user displacements. However, the distances measurements will be significantly less accurate.

The distance estimation method will be based on radio waves' RSSI measurements from a Wi-Fi Direct communication between two devices. It is true that RSSI-based estimations do not allow for a high estimation accuracy, but they do allow for an easy approximation of the distance. Other methods like the ToF of radio waves have been discarded due to its impossible implementation in the software environment that concerns this project, which is an Android smartphone app. Nevertheless, all the Wi-Fi Direct issues and methodology are not explained in this report, where it is directly considered that RSSI-based distances are available.

As it is said, RSSI does not enable high distance precision, since the radio waves can be disturbed by many factors. This perturbation could cause that the power received is lower than the expected from a certain distance, making it impossible for the receiver to obtain a good distance measurement. Therefore, the initial idea of exactly determining the relative position of a remote device resolves into a simpler approach: to dynamically calculate and update the relative direction and distance estimations while walking towards the current estimated direction, in order to finally meet the remote user.

4.1. Heuristic method

In this method, the purpose is to dynamically update a relative direction which makes A and B get closer, if A follows this direction. In order to accomplish that, a weighting system for different directions is implemented; however, before proceeding, a coordinate system has to be defined for both the self-position of A and the estimated position of B. This consists of a relative 2-D plane in which the x and y axes point to East and North, respectively. In the first steps, the origin of this coordinate system is set to the start position of A; however, this reference position might be actualized during the walking, as will be explained later. Finally, polar coordinates will be considered in order to situate in the plane all the relative positions.

4.1.1. Weighting system

During the walking of A, a weight is assigned for each direction walked relative to the reference position with an associated distance measurement, following the equation (4.1)

$$w_k = \frac{(d_0^{A-B} - d_k^{A-B})}{\Delta\rho}, \tag{4.1}$$

where d_0^{A-B} is the distance between A and B measured in the reference position, d_k^{A-B} the distance between A and B measured in the kth step position and $\Delta\rho$ the current rho coordinate of A's position.

If the direction walked was the same direction of B and the measured distance was correct, the weight would be 1 and the user would know that he is walking in the correct direction. If the direction walked was the opposite direction of B, i.e. $\rho_A = \rho_B + 180º$, the weight would be -1 and the user would know that he has to walk exactly towards the opposite direction. For other weights, the closer to 1 in absolute value they are, the closer the real relative direction and the current angular coordinate of A will be –considering for negative weights the current direction plus 180 º. Therefore, this method allows for determining the goodness of different directions.

As previously mentioned, the relative distance estimations are very noisy, therefore it is not attempted to estimate the exact relative direction of the remote device basing on the weights. However, a heuristic method implementation based on this direction weighting system, as well as some filtering, will be done in order to improve the overall process.

4.1.2. Kalman Filter

Obviously, this method is not called heuristic for the Kalman Filter implementation. However, this filter helps in the final performance of the method.

This filter is a set of equations that operate recursively in order to minimize the mean square error of an estimation. It is mainly used in navigation systems but has a wide variety of use cases. In this work, a simplified discrete implementation of it is used, since it will be used as an estimator of a random constant instead of a stochastic process. This is because A does not have *a priori* knowledge of the movement of B, so it will be considered that B remains static.

The Kalman filter tries to estimate the state $x \in \Re^n$ of a process governed by the linear stochastic difference equation represented in (4.2) with a measurement $z \in \Re^n$ represented in (4.3).

$$x_k = F_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1} \tag{4.2}$$

$$z_k = Hx_k + v_k \tag{4.3}$$

The n x n matrix F relates the state at the previous time step to the state at the current step. The n x 1 matrix G relates the optional control input $u \in \Re^n$ to the state x. The m x n matrix H relates the state to the measurement z_k. Finally, the random variables w_k and v_k, which are independent of each other, represent the process and measurement noise respectively, which is assumed to be white and with normal probability distribution.

Now, $\hat{x}_k^- \in \Re^n$ is defined as the state estimation at step k without the knowledge of z_k

$$\hat{x}_k^- = F_{k-1}\hat{x}_{k-1} + G_{k-1}u_{k-1}\,, \tag{4.4}$$

and $\hat{x}_k \in \Re^n$ as the state estimation at step k given the measurement z_k. Therefore, *a priori* and *a posteriori* estimate errors can be defined as

$$e_k^- \equiv x_k - \hat{x}_k^- \text{ , and}$$

$$e_k \equiv x_k - \hat{x}_k\,,$$

respectively.

Deriving the equations of the Kalman Filter, the equation that represents the *a posteriori* state estimate $\hat{x}_k$ is obtained. It consists of a linear combination of the *a priori* estimate $\hat{x}_k^-$ and a weighted difference between the actual measurement z_k and a measurement prediction $H\hat{x}_k^-$ as shown in the equation (4.5).

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{4.5}$$

The n x m matrix K in (4.5) is calculated to be the gain that minimizes the *a posteriori* error covariance

$$P_k = E[e_k e_k^T]. \tag{4.6}$$

Likewise, the *a priori* estimate error covariance is

$$P_k^- = E[e_k^- e_k^{-T}]. \tag{4.7}$$

In order to calculate the optimal K, the expression from (4.5) has to be substituted into the above definition of e_k, to then perform (4.7), take the derivative of the trace of the result with respect to K, set the result equal to zero and finally solve for K. One of the resulting forms of this K is given by

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R} \tag{4.8}$$

where R is the measurement error covariance.

The K computed in equation (4.8) can be understood as: the lower the measurement error covariance R is, the more the measurement z_k is trusted.

At this point, the *a priori* estimate error covariance has to be redefined in equation (4.9), since it is needed in order to calculate K_k in equation (4.8), and cannot be obtained from equation (4.7) because the errors are unknown.

$$P_k^- = FP_{k-1}F^T + Q \tag{4.9}$$

Q from equation (4.9) corresponds to the process noise covariance and F is the same than in equation (4.2).

Note that the *a priori* error covariance P_k^- in the kth step depends on the *a posteriori* error covariance from the $(k-1)^{th}$ step. Therefore, the *a posteriori* error covariance also has to be redefined:

$$P_k = (I - K_k H)P_k^- \tag{4.10}$$

Now, all the required equations have been exposed and they can be divided into two groups: time update equations (prediction) and measurement update equations (correction). The former are in charge of projecting forward –in time– the current state and error covariance estimates in order to obtain the *a priori* estimates for the next time step. The latter are responsible for incorporating the new measurement into the *a priori* estimate in order to obtain a better estimation. Before proceeding in stating the final equations, the specific case of this work has to be considered.

For this case, the state x is the relative direction of the device B and the measurement z is an estimated direction. Since the state is considered as static, F will be the identity matrix. Moreover, there will not be any control input and H will also equal the identity matrix because the measurements are directly from the state. Therefore, the final equations that will be implemented in the Android context are the following ones:

Time update equations	**Measurement update equations**
State projection $\hat{x}_k^- = \hat{x}_{k-1}^-$	Kalman Gain computation $K_k = P_k^-(P_k^- + R)^{-1}$
Error covariance projection $P_k^- = P_{k-1} + Q$	Estimation updating (with measurement z_k) $\hat{x}_k = \hat{x}_k^- + K(z_k - \hat{x}_k^-)$
	Error covariance updating $P_k = (I - K_k)P_k^-$

Table 1. Discrete Kalman Filter equations

Following these equations, the Kalman Filter allows the improving of the estimation in real time and without too much computational cost, since it satisfies the Markov property, i.e. the calculations of the future state are made regarding only the current state and data and not all the previous accumulated ones.

The implemented Java equations of the filter, as well as its initialization and a MATLAB simulation of it, are shown in the appendices of this document.

4.1.3. Development

4.1.3.1. Direction

As said before, this method will follow some heuristic implementations. The first one refers to how the direction estimations of A are considered as measurements for the Kalman Filter, even though, actually, they are not real direction estimations.

Firstly, the directions that have an assigned weight in absolute value lower than 0.5 are discarded. The other directions are considered as direction measurements; however, depending on their value, they can be considered as more than one measurement in order to provide a specific estimation with more or less credibility than others, since it is not the same a direction weighted with 0.5 than a direction weighted with 1.

Theoretically, the weight cannot be higher than one; however, in practice it certainly can, because of all the possible measurement errors. This does not mean that the directions that correspond to a weight higher than one are discarded; quite the contrary: the higher it is, the more credible this direction will be. In this way, five different stages are considered according to the weight value: from 0.5 to 0.8, from 0.8 to 1.5, from 1.5 to 4, from 4 to 9 and from 9 to 15. For these five stages the measurement will be considered once, twice, thrice, four times and five times, respectively.

For the negative value weights, the method will perform in an analogous way, considering in this case the current opposite weighted direction instead of directly the current one.

It is considered a weight limit of 15 both for positive and negative values. For positive values, this high weight would probably mean that the distance measured in the reference position d_0^{A-B} was made in a zone with bad coverage while the measurement in the current position is in a good coverage zone. An example scenario would be two users that had had a wall or a building between them in the reference position measurement, and now they have good visibility. Therefore, they would have passed from a Non-line-of-sight (NLOS) environment to a Line-of-sight (LOS) one. This yields to update the reference position as the current position of A.

On the other hand, weights lower than -15 would probably mean that the users have passed from a LOS to NLOS environment. In this case, however, the reference position is not updated and the current direction is simply discarded.

In order to improve the precision of the direction estimations during the walking, the reference position will be also updated when the estimated distance has become one fifth of the estimated distance in the reference position.

In any case in which the reference position is updated, the Kalman Filter will be reinitialized.

4.1.3.2. Position

Once direction estimation is obtained, an approximate relative position can be calculated. The most straightforward way is to consider the distance estimation from the reference position, since the relative direction is always in regard to this position. In this way the remote position of B p_k^B would be

$$p_k^B = \begin{bmatrix} x_k^B \\ y_k^B \end{bmatrix} = \begin{bmatrix} x_0^A \\ y_0^A \end{bmatrix} + d_0^{A-B} * \begin{bmatrix} \cos\theta^{A-B} \\ \sin\theta^{A-B} \end{bmatrix}, \tag{4.11}$$

where x_k^B and y_k^B are the Cartesian coordinates of B's position in the kth step, x_0^A and y_0^A the Cartesian coordinates of A's reference position, d_0^{A-B} the distance measurement between the two devices in the reference position, and θ^{A-B} the relative direction between the two devices –regarding the reference position.

However, as A and B get closer the absolute error of the distance measurement diminishes. Therefore, it would be better to consider the closer distance measurements rather than the reference position one. In order to accomplish that, a method based on *Figure 5* representation is performed.

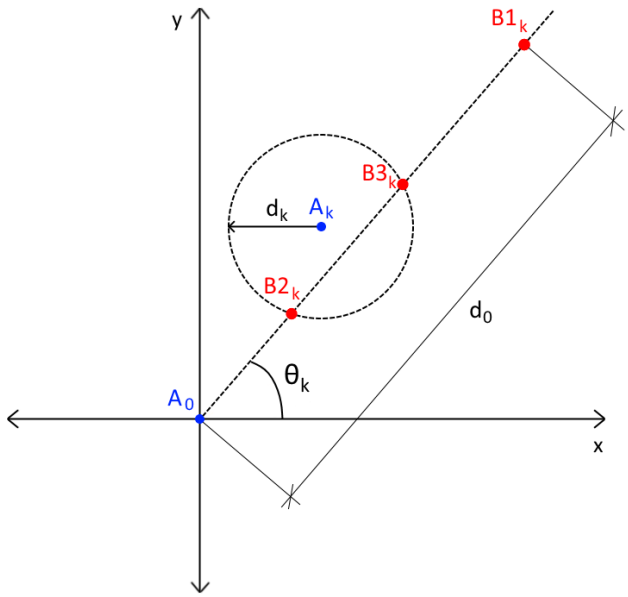


Figure 5. Improvement of position estimation representation

In *Figure 5*, A_0 represents the reference position of A, A_k the position of A in the kth step, d_0 the distance measurement between A and B in the reference position, d_k the distance measurement between A and B in the kth step, θ_k the estimated direction in the kth step, $B1_k$ the estimated position of B in the kth step following the estimation represented in equation (4.11), and $B2_k$ and $B3_k$ two estimated positions of B in the kth step following the method explained right after.

In order to obtain $B2_k$ and $B3_k$ estimations, an intersection between a circumference and a straight line has to be performed as shown in *Figure 5*. The circumference is represented as

$$\left(x - x_k^A\right)^2 - \left(y - y_k^A\right)^2 = d_k \,, \tag{4.12}$$

where x_k^A and y_k^A represent the position of A in the kth step, and the straight line as

$$y = \tan\theta_k * x \,. \tag{4.13}$$

Therefore, the x values that represent the two intersection points are defined by

$$\left(x - x_k^A\right)^2 - \left(\tan\theta_k * x - y_k^A\right)^2 = d_k \,. \tag{4.14}$$

Solving for x and substituting in (4.13) the two intersection points are acquired. The furthest point will be the chosen one, since it is considered that the user A has not surpassed B yet.

Therefore, this point will be the one representing the remote device in the 2-D dynamic plot detailed in the section 3.5 of this report.

5. Acoustic distance

In this section, a high-accuracy software-based acoustic ranging solution is presented. Typically, the ranging methods are achieved through measuring the ToF of acoustic or radio signals. In this case, the distance between two devices is the product of the ToF and the signal speed –speed of sound for acoustic signals and speed of light for radio signals. Obviously, it is easier to perform this method with acoustic signals, since their velocity is much lower than that of radio signals; however, at the same time, the range is smaller. In this thesis, the acoustic ranging is used to perform a high-accuracy distance estimation between two smartphone's users when they are close enough to hear each other by emitting acoustic signals. Likewise, it can also be useful for other kind of use cases, as simply calculating the length of an object.

The biggest problem of the ToF methods is the precision of these time measurements, which are often taken by timestamps' recordings of local devices' clocks at the moment the signal is emitted or received. In this case, there could be three uncertainty factors that would lead to measurement inaccuracies: the clock skew between the two devices involved in the process, the misalignment between the sender timestamp and the current acoustic emission, and the delay between the sound arriving and the current recognition of it at the receiver.

The former can be solved with internet access, by considering the same remote clock. However, without internet coverage this would not be possible. The latter two uncertainties could be caused by several intrinsic factors and are impossible to be sufficiently minimized. Therefore, the timestamp-based approach is discarded.

For this project, the method to achieve the acoustic ranging resides on sample counting instead of timestamps difference, as it is performed in [8].

5.1. Sample counting method

In this method, two devices emit an acoustic signal and each of them records both its own and the remote's signals. With a good detection approach, both signals will be detected and the amount of recording samples between them will be easily calculated. Then, this can be transformed into time measurements, following the equation (5.1):

$$t = \frac{N_2 - N_1}{f_s} \tag{5.1}$$

where t is the time elapsed between the two signals recorded in a single device, N_2 and N_1 the samples in which the signals are detected, and f_s the sample frequency of the recording.

If this equation is calculated in both devices, and the sounds emitted from each device do not interfere between them, i.e. they do not share space-time, the distance measurement between the two devices is calculated by the following equation:

$$d = \frac{t_1 - t_2}{2} * v \tag{5.2}$$

where, t_1 is the time calculated from (5.1) in the device that first sends the sound, t_2 the time calculated in the other device, and v the speed of sound. Therefore, in order to calculate the distance in a device, calculations from the other device are needed; thus, a communication between them will be compulsory. This communication will be also based on the Wi-Fi Direct protocol but, as mentioned before, all these issues are not explained in this report.

If the detection has been done properly, the distance measurement granularity is limited only to the sound sampling rate. In this way, the accuracy would be:

$$e = \frac{1}{f_s} * v \tag{5.3}$$

where e is the maximum distance error, f_s the sampling frequency and v the speed of sound. With a f_s of 44100 Hz and considering a sound speed of 340 m/s, the maximum error is only about 8 mm.

In order to perform this detection in an optimal way, a chirp signal is chosen, due to its wide spectrum and, hence, good detectability. Moreover, a Matched Filter is implemented at the receiver, as it will be explained later.

5.2. Signal generation

The Android API does not have any method to directly play a specific acoustic sound as, for example, a simple pure tone or a chirp signal. Therefore, the desired signal has to be firstly generated.

As it is previously said, the chosen acoustic sound is a chirp signal, specifically a linear up-chirp. This signal is characterised by an instantaneous frequency that increases linearly with time, following the equation (5.4):

$$f(t) = f_0 + kt \tag{5.4}$$

where f_0 is the starting and lowest frequency, and k is the rate of frequency increase or chirp rate that can be defined as (5.5):

$$k = \frac{f_1 - f_0}{T} \tag{5.5}$$

where f_1 is the final frequency and T the time taken to sweep from f_0 to f_1.

Once the chirp main characteristics are defined, it can be generated in a Java context. However, in this digital context the previous analogous definitions cannot be directly applied and the continuous time domain has to be transformed into a discrete one. In this way, the generated signals will have a finite number of samples.

Firstly, an array of the desired number of samples has to be created. Then, the values for each sample are calculated as shown in equations (5.6) and (5.7):

$$c_k = \sin\left(2 * \pi * f_k * \frac{k}{f_s}\right) \tag{5.6}$$

Where c_k is the chirp signal value in the kth sample, k the current sample, f_k the instant chirp frequency for each sample – defined in (5.7) – and f_s the sample rate.

$$f_k = f_0 + k * \frac{f_1 - f_0}{N - 1} \tag{5.7}$$

In equation (5.7), f_0 and f_1 follow the same definitions as in the equations (5.4) and (5.5), and N is the number of samples of the signal.

The parameters selected for the chirp signal generation are:

$$f_0 = 13\ kHz$$

$$f_1 = 18\ kHz$$

$$f_s = 44.1\ kHz$$

$$N = 1500\ samples$$

These parameters allow for a signal duration of 34 ms approximately.

The selection of the initial and final chirp frequencies are made based on human listening capabilities, since it is preferable than the emitted sounds are practically inaudible. Moreover, the bigger the difference between them is, the more bandwidth the signal will have, and this will allow for an easier detection.

On the other hand, the sample rate is set bearing in mind the Nyquist–Shannon sampling theorem. This theorem guarantees that if a signal contains no frequency higher than B hertz it can be sampled and then perfectly reconstructed –by means of interpolation–, whenever the sample rate is higher than 2B (5.8):

$$f_s > 2B \tag{5.8}$$

Therefore, f_s has to be higher than 36 kHz, that is the highest frequency of the chosen chirp signal multiplied by two. However, the Android smartphones do not allow every sample rate, so the final parameter will be set to 44.1 kHz, which is allowed by them.

Finally, the selection of N, i.e. the number of samples, is made following an agreement between the time of processing and the detectability of the signal, because the larger the signal is, the easier to be detected by a receiver it becomes, but at the same time, the longer the process of detecting will be.

5.3. Audio playing

In order to play a sound in Android, the AudioTrack class is used. As it is said in the Android developers' reference page, this class allows streaming of Pulse Code Modulation[5] (PCM) audio buffers to the audio sink for playback. This is achieved by the method `write(byte[], int, int)`, which push the data to the AudioTrack object. Then, the `play()` method is only required in order to finally play the sound.

[5] Method to digitally represent sampled analog signals.

Before that, the AudioTrack object has to be initialized, and this will be made with the following class constructor: `AudioTrack(int streamType, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes, int mode)`.

- The `streamType` selected is the default audio stream for music playback, i.e. `AudioManager.STREAM_MUSIC`.
- The `sampleRateInHz` parameter is the sample rate defined in the previous section (5.2).
- The `channelConfig` is `AudioFormat.CHANNEL_OUT_MONO`, since only one audio output channel is required.
- The `audioFormat` parameter is `AudioFormat.ENCODING_PCM_16BIT`. This format will allow a 16-bit PCM encoding, which is the largest one available for the AudioTrack class. This encoding allows 2^{16}, i.e. 65536, different levels for the digital representation of a signal. Specifically, the available values will be from -32768 to 32767.
- The `bufferSizeInBytes` parameter is the number of samples that will be written to the AudioTrack object and subsequently played. Since the signal is generated as short (16 bits) values, and the AudioTrack object will be written in bytes (8 bits), the length of the AudioTrack buffer will be the original number of samples multiplied by 2 –two bytes per sample.
- The `mode` chosen is `AudioTrack.MODE_STATIC`, since it is the mode recommended when dealing with short sounds that fit in the memory and that need to be played with the smallest latency possible.

Since the 16-bit PCM is the chosen encoding, the generated signal will be set as short[6] type, as well as set with values from -32768 to 32767. However, in order to "push" the data into the AudioTrack object by the `write(byte[], int, int)` method, an array of bytes is required. Therefore, each short type sample will be converted into two byte[7] type. This will be made by the following two code lines executed in a loop:

```
byteValues[i++] = (byte) (shortValues & 0x00ff);
byteValues[i++] = (byte) ((shortValues & 0xff00) >>> 8);
```

where `byteValues` and `shortValues` are the arrays that store the byte and short values of the signal respectively. On the other hand, `&` is the bitwise AND operator[8] and `>>>` the zero fill right shift operator[9]. In this way, the first line will copy the lowest 8 bits of the `shortValues` to the `byteValues`, while in the second line the 8 bits copied will be the highest ones of `shortValues`. This follows the little-endian format, which is the required for the data representation in the AudioTrack object. In this format, the least significant byte is stored at the first location and the most significant byte in the last one.

Finally, the chirp signal will be played by executing the two following code lines:

```
audioTrack.write(byteValues, 0, byteValues.length);
audioTrack.play();
```

[6] The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
[7] The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
[8] This operator copies a bit to the result if it exists in both operands
[9] The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

5.4. Audio recording

For the audio recording, an AudioRecord class object will be used. As it is said in the Android developers' reference page, this class manages the audio resources for Java applications to record audio from the audio input hardware of the platform. In order to acquire the data from the AudioRecord object, the `read(short[] audioData, int offsetInShorts, int sizeInShorts)` method will be used.

Before that, the AudioRecord object has to be initialized, and this will be made with the following class constructor: `AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes)`.

- The `audioSource` selected is the microphone audio source `MediaRecorder.AudioSource.MIC`.
- The `sampleRateInHz` parameter is the same than in the AudioTrack object initialization.
- The `channelConfig` is the `AudioFormat.CHANNEL_IN_MONO`, since only one audio input channel is required.
- The `audioFormat` is the same than in the AudioTrack object initialization.
- The `bufferSizeInBytes` is the desired number of samples of the recording.

After the AudioRecord object initialization is done, the following two code lines are executed in order to commence the recording:

```
audioRecord.startRecording();
audioRecord.read(receivedByteValues, 0, receivedByteValues.length);
```

where the `receivedByteValues` parameter will store the recorded signal in bytes following the 16-bit PCM modulation.

Finally, in order to get an array of the same type as the one generated for the playing, the `receivedByteValues` will be transformed into a `short` data type array by means of the `ByteBuffer` class methods.

5.5. Matched filter

The acoustic range is the main drawback of the proposed acoustic distance method. For that reason, some filtering is done at the receiver in order to improve the detection process.

Firstly, the block diagram of *Figure 6*, which represents a filter implementation, will be defined by equations (5.9) and (5.10), where x[k] is the input of the filter, h[k] the filter impulse response, y[n] the output of the filter, and X(f), H(f) and Y(f) its respective frequency-domain functions.

Figure 6. Block diagram of a filter implementation

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{5.9}$$

$$Y(f) = X(f) * H(f) \tag{5.10}$$

Now, the emitting and receiving process of the proposed acoustic method can be schematised as shown in *Figure 7*, where s[n] represents the emitted sound, i.e. the chirp signal, w[n] an Additive White Gaussian Noise (AWGN), h[n] the impulse response of the filter and y[n] the output signal of the filter:

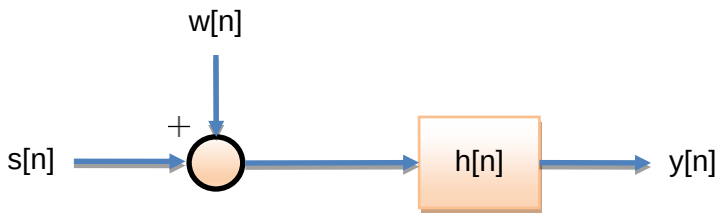


Figure 7. Block diagram of the acoustic distance estimation process

In the above block diagram, y[n] can be defined in the following way:

$$y[n] = s_o[n] + w_o[n] \tag{5.11}$$

where $s_o[n]$ and $w_o[n]$ are the filtered signal and noise parts of the input of the filter, respectively.

Now, the optimal impulse response for the filter is calculated in order to maximize the Signal to Noise Ratio (SNR) of the detected peak, which is defined in the following equation:

$$\frac{\hat{S}}{N} \triangleq \frac{|s_o[k_m]|^2}{N} \tag{5.12}$$

where $\frac{\hat{S}}{N}$ is the peak SNR, $s_o[k_m]$ the received sample in which the sound arrives and N the noise power.

Firstly, $s_o[k_m]$ and N will be expressed regarding the filter response in equations (5.13) and (5.14):

$$\begin{aligned} s_o[k_m] &= \mathcal{F}^{-1}\{S(f) * H(f)\}\,|_{k=k_m} \\ &= \int_{-\infty}^{\infty} S(f) * H(f) * e^{jwk_m}\delta f \end{aligned} \tag{5.13}$$

$$\begin{aligned} N &= E\{|w_o[k]|^2\} \\ &= \int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \delta f \end{aligned} \tag{5.14}$$

where $\mathcal{F}^{-1}$ represents the inverse Fourier transform operator and $E\{\}$ the expectation operator. Therefore, substituting (5.13) and (5.14) in (5.12):

$$\frac{\hat{S}}{N} = \frac{\left|\int_{-\infty}^{\infty} S(f) * H(f) * e^{jwk_m}\delta f\right|^2}{\int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \delta f} \tag{5.15}$$

Now, the Cauchy–Schwarz inequality will be applied in order to find the maximum value of $\frac{\hat{S}}{N}$, as well as the value of $H(f)$ in order to obtain it. The inequality is represented in the following equation:

$$|\langle A, B\rangle|^2 \leq \langle A, A\rangle * \langle B, B\rangle \,, \tag{5.16}$$

where $\langle \cdot, \cdot \rangle$ is the inner product.

Moreover, '$\leq$' from equation (5.16) becomes '$=$' if

$$A = K * B \,, \tag{5.17}$$

so

$$|\langle A, B\rangle|^2 = \langle A, A\rangle * \langle B, B\rangle \mid_{A=K*B} . \tag{5.18}$$

The Cauchy–Schwarz inequality can also be rewritten as:

$$\left|\int_{-\infty}^{\infty} A * B^* \, dx\right|^2 \leq \int_{-\infty}^{\infty} |A|^2 dx \int_{-\infty}^{\infty} |B|^2 dx \,. \tag{5.19}$$

Now, defining A and B^* as

$$A = H(f) * \sqrt{\phi_{nn}(f)} \tag{5.20}$$

$$B^* = \frac{S(f) * e^{jwk_m}}{\sqrt{\phi_{nn}(f)}} \tag{5.21}$$

and applying the Cauchy-Schwarz inequality

$$\begin{aligned}
\frac{\hat{S}}{N} &= \frac{\left|\int_{-\infty}^{\infty} A * B^* \, \delta f\right|^2}{\int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \delta f} \\
&\leq \frac{\int_{-\infty}^{\infty} |A|^2 \, \delta f \int_{-\infty}^{\infty} |B|^2 \, \delta f}{\int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \delta f} \\
&\leq \frac{\int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \, \delta f \int_{-\infty}^{\infty} \left|\frac{S(f) * e^{jwk_m}}{\sqrt{\phi_{nn}(f)}}\right|^2 \delta f}{\int_{-\infty}^{\infty} \phi_{nn}(f) * |H(f)|^2 \delta f} \\
&\leq \int_{-\infty}^{\infty} \left|\frac{S(f) * e^{jwk_m}}{\sqrt{\phi_{nn}(f)}}\right|^2 \delta f \\
&\leq \int_{-\infty}^{\infty} \frac{|S(f)|^2}{\phi_{nn}(f)} \, \delta f
\end{aligned} \tag{5.22}$$

the optimum $\frac{\hat{S}}{N}$ value is obtained: $\int_{-\infty}^{\infty} \frac{|S(f)|^2}{\phi_{nn}(f)} \, \delta f$.

Therefore, the greater the bandwidth of the signal is, the greater the peak SNR will be.

Finally, in order to obtain the optimum $\frac{\hat{S}}{N}$ in the system, the condition represented in the equation (5.17) must be met, so

$$H_{opt}(f) * \sqrt{\phi_{nn}(f)} = K * \left(\frac{S(f) * e^{jwk_m}}{\sqrt{\phi_{nn}(f)}} \right)^*$$

$$H_{opt}(f) = K * \frac{S^*(f) * e^{-jwk_m}}{\phi_{nn}(f)}. \tag{5.23}$$

This H_{opt} can be expressed in the time domain as:

$$h_{opt}[n] = K * s^*[k_m - n]\,. \tag{5.24}$$

This impulse response of the filter can perfectly be designed since the same signal is always emitted and the receiver knows it.

Setting $K = 1$ and $k_m = 0$ for commodity, the final block diagram is shown in *Figure 8*.

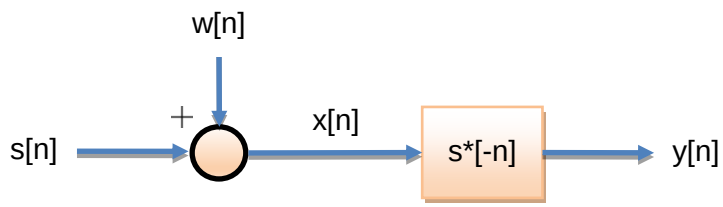


Figure 8. Block diagram of the Matched Filter process followed in this work

In *Figure 8*, s[n] represents the chirp signal generated and played, w[n] an AWGN, x[n] the chirp signal with AWGN, i.e. the chirp signal recorded, s*[-n] the filter impulse response, which is the conjugated time-reversed version of the generated chirp signal, and y[n] the matched output from the filter, represented in the equations (5.25) to (5.28).

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]s^*[n+k] \tag{5.25}$$

$$y[n] = \sum_{k=-\infty}^{\infty} (s[k] + w[k]) * s^*[n+k] \tag{5.26}$$

$$y[n] = \sum_{k=-\infty}^{\infty} s[k]s^*[n+k] + w[k]s^*[n+k] \tag{5.27}$$

$$y[n] = \sum_{k=-\infty}^{\infty} R_{ss}[-n] + w[k]s^*[n+k] \tag{5.28}$$

6. Results

6.1. PDR

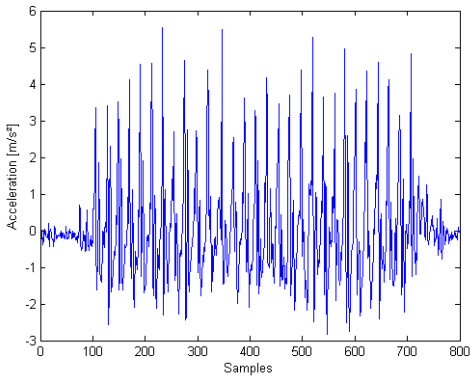


Figure 9. Z-axis acceleration measurements (29 steps)

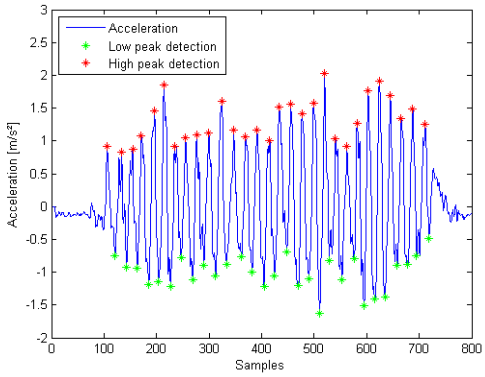


Figure 10. Step detection (29 steps)

In *Figure 9*, the acceleration measurements in the z axis (in GCS) for a walk of 29 steps are represented.

In *Figure 10*, the low-pass filtered acceleration measurements shown in *Figure 9* are represented. Moreover, the algorithm that detects local peaks is performed, obtaining the red and green asterisks for every maximum and minimum detected, respectively. Each pair of consecutive high and low peak detections represents a single step. Thus, as seen in *Figure 10*, all the 29 steps have been correctly detected.

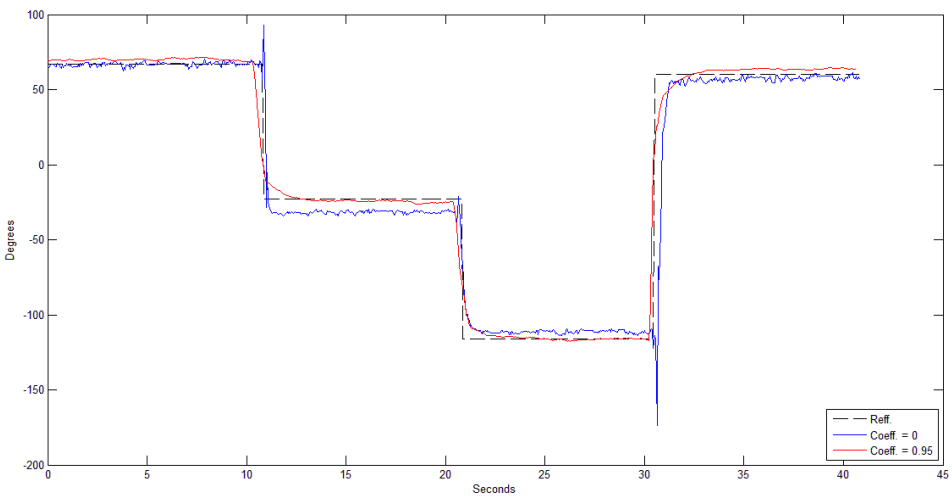


Figure 11. Orientation estimation test

In *Figure 11*, it is shown a comparison between the orientation from the accelerometer/magnetometer sensors alone (coefficient of the complementary filter equal to zero), and the orientation from the sensor fusion method with a coefficient of 0.95, which is the coefficient chosen. It is clearly seen that the sensor fusion method allows the orientation measurements to not have undesired spikes, as well as to be smoother. Moreover, it also allows a better dynamic response than the application of a low-pass filter to the accelerometer/magnetometer measurements.

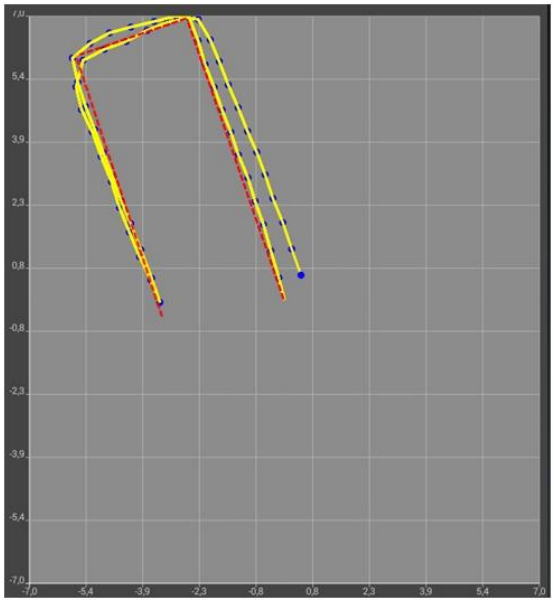


Figure 12. PDR performance

In *Figure 12*, the walking trajectory of a user is represented following all the methods explained in the section 3 of this report. This 2-D plot is what the user can see in the application during his walking –with the exception of the red dashed line, which represents the real walking of the user and is drawn for a comparative purpose between the real and the estimated trajectory. In this case, after 54 steps and returning to the initial position, the accumulated estimation error is about 95 cm.

In order to quantify the performance of this method, five repetitions of the trial represented in *Figure 12*, but only in one way, have been tested.

Figure 13 illustrates the five trials, and *Table* 2 shows the overall performance of the method.

	Steps done	Steps detected	Distance travelled	Real final position	Estimated final position	Accumulated absolute error	Accumulated relative error
Trial 1	28	28	17.3 m	[-3.4,-0.4] m	[-3.55, -0.22] m	0.23 m	1.35 %
Trial 2	28	28	17.3 m	[-3.4,-0.4] m	[-3.21,-0.04] m	0.41 m	2.35 %
Trial 3	28	27	17.3 m	[-3.4,-0.4] m	[-3.66,-0.13] m	0.37 m	2.16 %
Trial 4	28	28	17.3 m	[-3.4,-0.4] m	[-4.67,-0.09] m	1.3 m	7.55 %
Trial 5	28	28	17.3 m	[-3.4,-0.4] m	[-3.89,-0.76] m	0.6 m	3.5 %

Table 2. PDR performance

Considering the above table, the relative error is less than 5 % of the distance travelled in most cases, and always less than 10 %.

On the other hand, 139 out of 140 steps walked by the user throughout the five trials have been correctly detected. This represents more than 99 % of accuracy in the step detection.

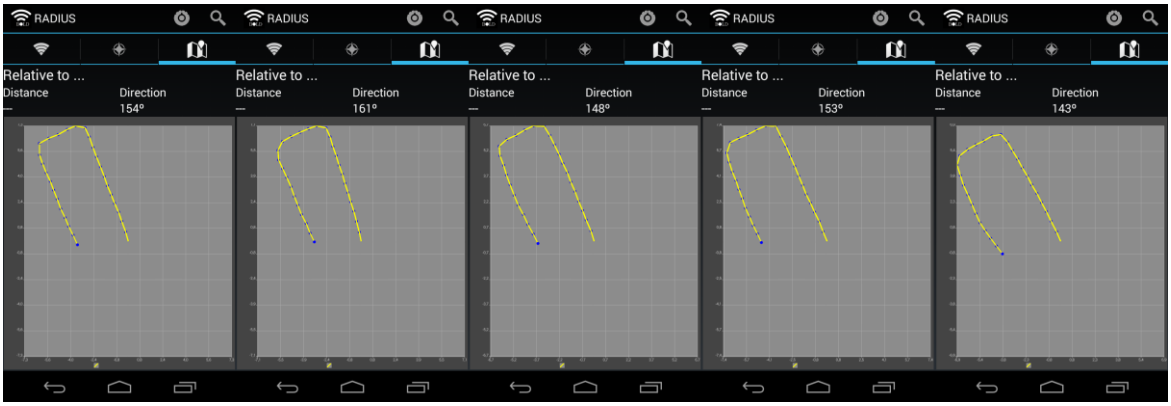


Figure 13. Five PDR trials in a specific trajectory

6.2. Relative positioning

It is very difficult to quantify the performance of this method, since it is very variable and it can be analysed in many ways. However, it can be said that, overall, the method does not work as expected.

In a straight approaching of A towards B, the method usually works well. In this case, the application indicates that user A is walking in the correct direction, and the distance estimation is updated recursively while walking as shown in *Figure 14*, where the red points represent all the estimated positions of B and the yellow line the trajectory travelled by A.

In the test shown in *Figure* 14, the estimated distance in the reference position between both of them was about 20 meters, while the real distance was about 50 meters. However, this does not affect the approaching between them, since by following the estimated direction both users finally meet. Moreover, as A approaches B, the distance estimation –and hence the relative position estimation– becomes more accurate.

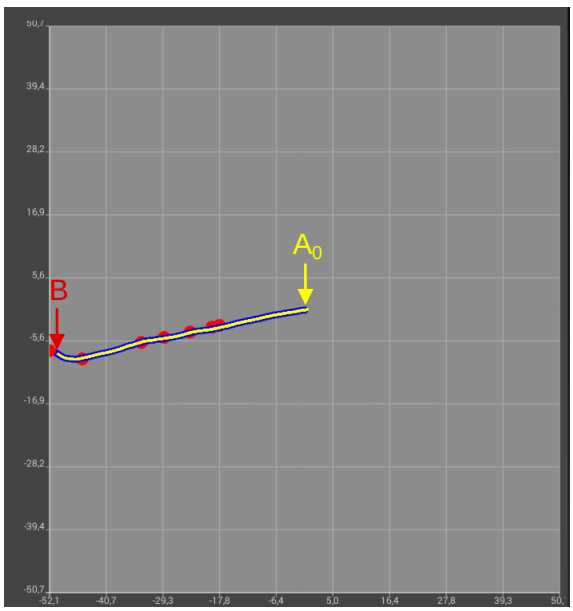


Figure 14. Relative positioning test in a straight walk towards the remote device (the arrow A_0 represents the initial position of A and the arrow B the real position of B during the entire walk).

On the other hand, if A starts walking in a random direction the method usually goes wrong. However, there have been some trials in which the method clearly enables both users to find each other, as shown in *Figure 15*.

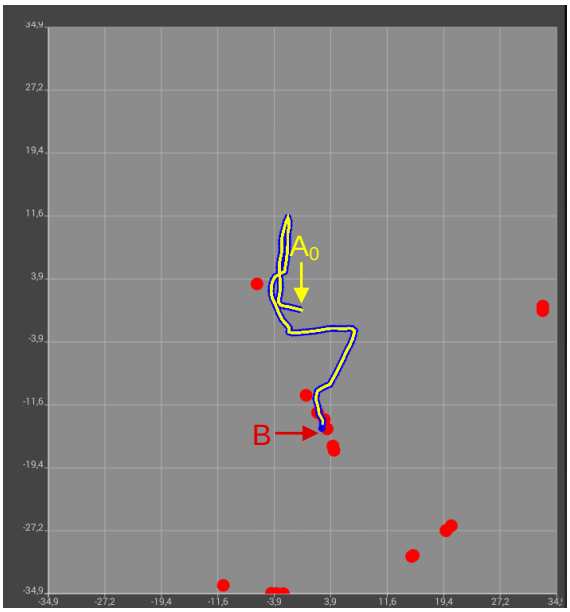


Figure 15. Relative positioning test performing properly (the arrow A_0 represents the initial position of A and the arrow B the real position of B during the entire walk).

In the above figure, the user A walks following the estimated relative direction, and after 113 steps he gets B in a distance shorter than 1 meter.

6.3. Acoustic distance method

In this method, the choosing of the chirp frequency is critical, since depending on the smartphone it will be properly reproduced and heard, or not. In the experiments done, three smartphones have been tested: Google Nexus 5, Motorola Moto G (2nd generation) and BQ Aquaris E4.5. While the microphones and speakers of the first two are able to receive and reproduce any frequency between 0 and 22 kHz, the latter's are unable to hear frequencies higher than 8 kHz. Moreover, this one also has difficulties to reproduce high frequencies like 17 kHz or higher. This yields to finally set the chirp frequency between the audible frequencies 3 kHz and 7 kHz, instead of the ones proposed in the section 5.2 of this report.

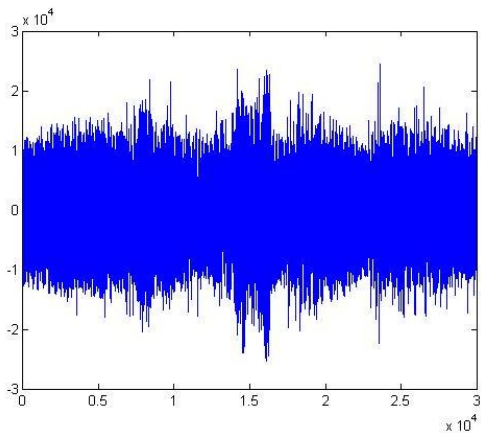


Figure 16. Recording during the acoustic distance process

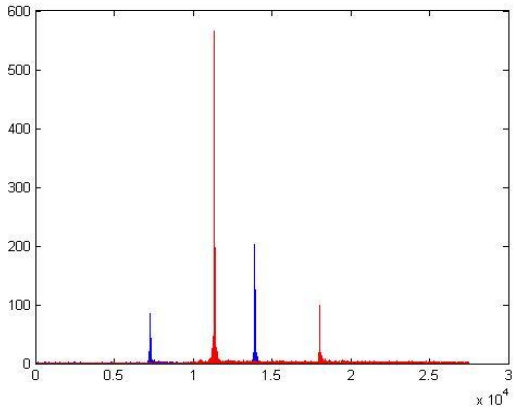


Figure 17. Matched filtered recording in both devices

In *Figure 16*, the recording of the whole acoustic distance process is represented. The recording input consists of two chirp signals in a noisy environment; one emitted from a remote device and the other from the own device. Applying the Matched Filter at both devices, the plot shown in *Figure 17* is obtained, where the blue line represents the filtered recording in one device (the one represented in the recording of *Figure 16*) and the red line the filtered recording in the other device. The difference in the distance between the two highest peaks in each device yields to a distance estimation between them, as it is explained in the section 5.1 of this report.

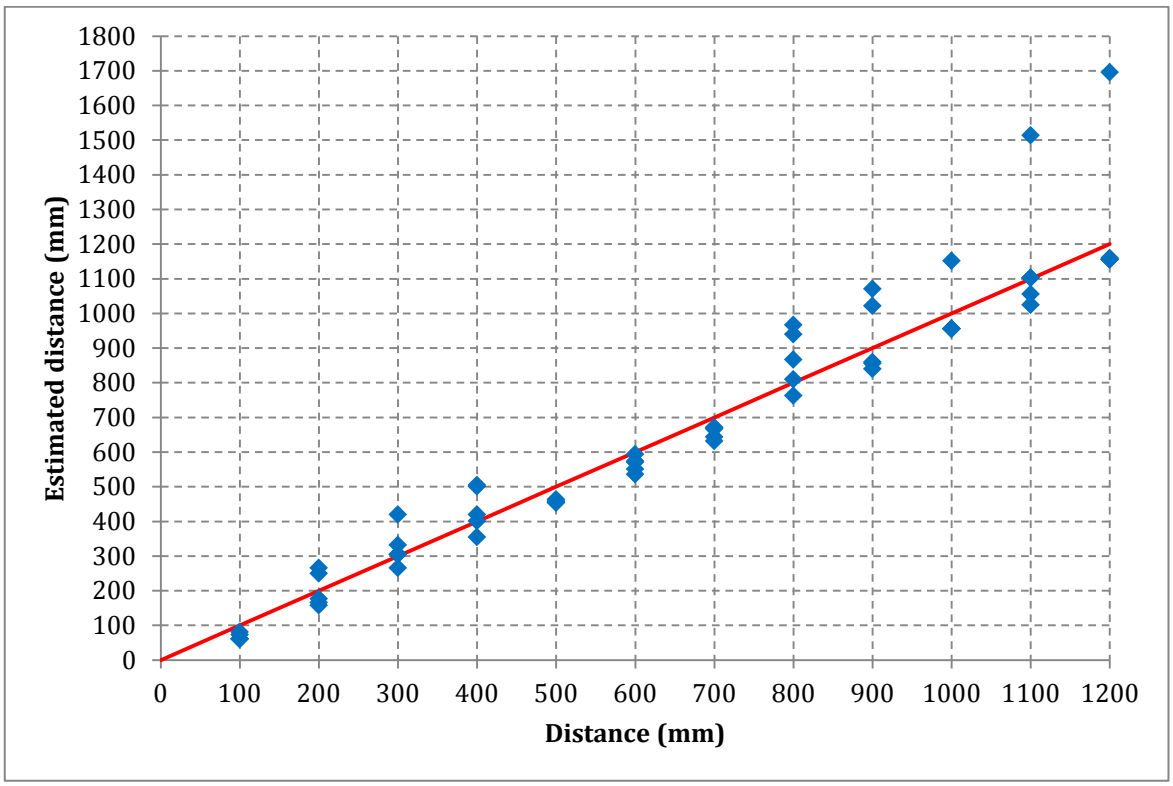


Figure 18. Estimated distance against real distance

In *Figure 18* the acoustic distance method performance is represented, taking five samples for each distance measurement. The red line represents the real distance, while the blue diamonds represent the estimated distances in each measurement. Beyond 1.2 meters, the method starts failing in most trials.

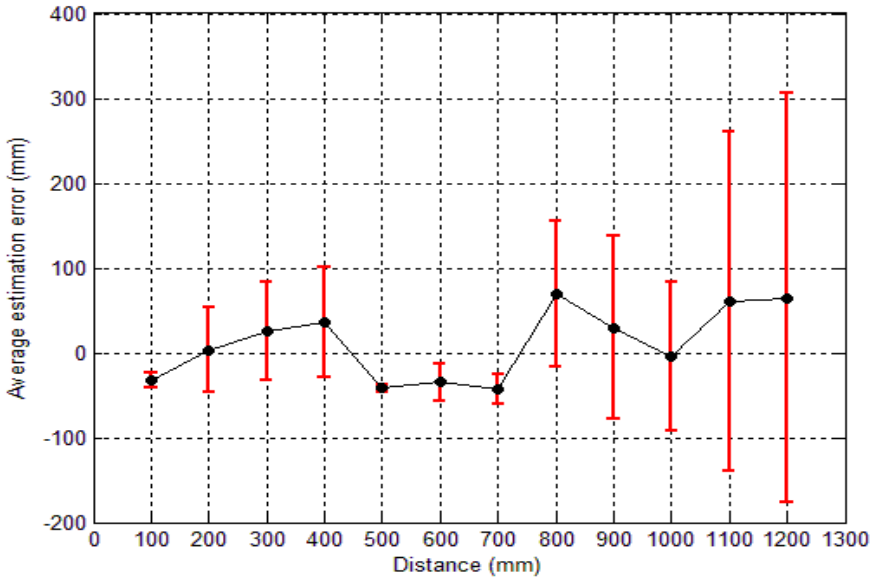


Figure 19. Absolute error mean and standard deviation

In *Figure 19*, the average absolute error (black points) and the standard deviation (vertical red lines) from the five measurements shown in *Figure 18* are represented.

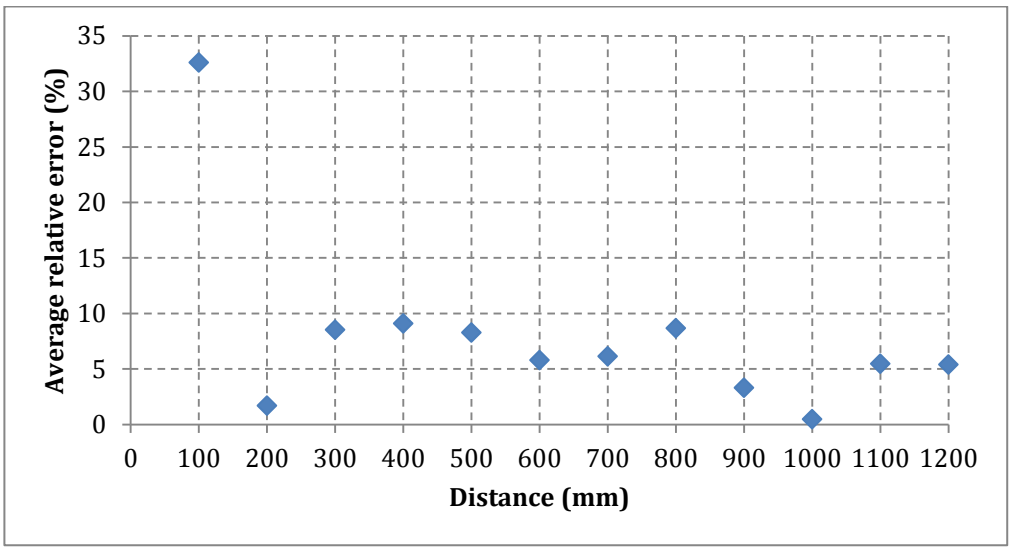


Figure 20. Average relative error

In *Figure 20*, the average relative error of the five distance measurements done for each distance is represented. It is clearly seen that between 0.2 and 1.2 meters, the five sampling average relative error is less than 10 % for every distance measurement.

7. Budget

Personnel	Time	Cost/hour	Total cost
Junior Engineer	720 h	8 €	5760 €
Project Supervisor	10 h	18 €	180 €
Total	730 h		5940 €

Table 3. Personnel costs

Software	Cost
Android Studio	- €
MATLAB	105 €
Total	105 €

Table 4. Software costs

Cost activities	Cost
Personnel	5940 €
Software	105 €
TOTAL	**6045 €**

Table 5. Total costs

8. Conclusions and future development

Clearly, the results that represent the main objective of this thesis are not good enough. However, we were aware of the difficulty of this project, so it cannot be considered as a failure, quite the contrary; it can be the beginning of a further research on this topic that may allow a proper performance in the future.

Some of the methods performed in this thesis do work properly, like the pedestrian dead reckoning approach, that allows a smartphone user to track his pedestrian movement in a two dimensional plane and in real time, even in indoor scenarios. This could be improved so that the user movement could be tracked in three dimensions, using the suitable accelerometer and gyroscope data, as well as the barometer sensor embedded in some current smartphones.

Regarding the RSSI measurements, we realised how bad they are in order to estimate distances based on them. For this reason, all the tried methods for estimating a relative position between two devices by means of the RSSI measurements show poor results. Maybe a better implementation of the Kalman filter or the use of different signal processing methods like the particle filter would improve the overall performance. Undoubtedly, it is a great challenge to handle the RSSI fluctuation in order to obtain useful information about the environment from it.

Concerning the acoustic distance estimations, a good measurement precision that will allow the measurement of little objects or distances has been reached. However, this method is not able to be useful in the users approaching method, since it has a very limited range. The matched filter designed for the detection of the chirp signals seems to work properly, since the detected peaks power for short distances are usually at least a hundred times the power of the detected noise, even in very noisy scenarios. Nevertheless, for distances longer than two meters, it seems that the smartphone's microphones do not record the chirp signal. In this case, it is impossible to detect it even with the best signal processing at the receiver.

Finally, I think that this degree thesis has been very useful to complete my bachelor's degree education, since it makes me face real problems and learn autonomously. For instance, before starting this project, I did not have any prior knowledge in Android development, and now I am able to develop full Android applications of many types. Moreover, I learned to deal with real time information as well as to manage multiple sensors' data at once, and I experimented with real processing approaches learned theoretically during the degree.

Bibliography

[1] Android Developers, API Guides, Sensors Overview, Sensor Coordinate System. [Online] Available: *http://developer.android.com/guide/topics/sensors/sensors_overview.html [Accessed: 15* May 2016*]*

[2] *H. Weinberg, "Using the ADX1202 in pedometer and personal navigation applications," 2002.*

[3] *D. Alvarez, R. C. González, A. López, and J. C. Alvarez, "Comparison of step length estimators from wearable accelerometer devices," in Proc. IEEE EMBS, Aug. 2006, pp. 5964–5967.*

[4] MathWorks Mobile Sensor Connectivity Team, 2013. [Online] Available: http://www.mathworks.com/ matlabcentral/mlc-downloads/downloads/submissions/40876/versions/8/screenshot.jpg [Accessed: 16 May 2016]

[5] Android Developers, Sensor Event. [Online] Avaliable:. *http://developer.android.com/intl/es/reference/android/hardware/SensorEvent.html* [Accessed: 16 May 2016]

[6] DF Malan (Own work), Public Domain. [Online] Available: *https://commons.wikimedia.org/w/index.php?curid=1354297* [Accessed: 16 May 2016]

[7] James Diebel. "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors". October 2006, Standford University.

[8] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. Beepbeep: A high accuracy acoustic ranging system using cots mobile devices. In Conference on Embedded Networked Sensor Systems (Sensys), 2007.

Appendices

REAL-TIME LOCAL PEAKS DETECTION ALGORITHM:

```
public void localPeakDetect(Float z) {
    current = z;
    if (current > max) {
        max = current;
        maxpos = index;
    }
    if (current < min) {
        min = current;
        minpos = index;
    }

    if (lookForMax) {
        if (current < (max - threshold)) {
            maxValues.add(max);
            maxIndex.add(maxpos);

            min = actual;
            minpos = index;

            numMaxs++;

            lookForMax = false;
        }
    } else {
        if (current > (min + threshold)) {
            minValues.add(min);
            minIndex.add(minpos);

            max = current;
            maxpos = index;

            numMins++;

            lookForMax = true;
        }
    }
    index++;
}
```

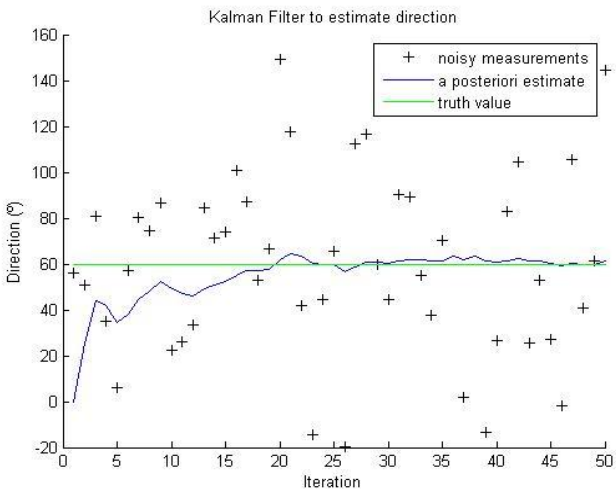
KALMAN FILTER INITIALIZATION:

```
public void initKalmanFilter() {
    kalmanQ = 0.00001f;
    kalmanR = 0.001f;
    kalmanP = 1.f;
    firstKalman = true;
}
```

KALMAN FILTER ALGORITHM:

```
public float kalmanFilter(float direction) {
    if (firstKalman) {
        xhat = direction;
        firstKalman = false;
        return xhat;
    }
    // time update
    xhatminus = xhat;
    Pminus = kalmanP + kalmanQ;

    // -179° <--> 179° transition
    measurementInnovation = direction-xhatminus;
    if (measurementInnovation > 180.f) {
        measurementInnovation -= 360.f;
    }
    else if (measurementInnovation < -180.f) {
        measurementInnovation += 360.f;
    }

    // measurement update
    kalmanK = Pminus / (Pminus + kalmanR);
    xhat = xhatminus + kalmanK * (measurementInnovation);
    kalmanP = (1 - kalmanK) * Pminus;

    // -179° <--> 179° transition
    if (xhat > 180.f) {
        xhat -= 360.f;
    }
    else if (xhat < -180.f) {
        xhat += 360.f;
    }

    return xhat;
}
```

KALMAN FILTER MATLAB SIMULATION:



Glossary

GPS: Global Positioning System

ToF: Time of Flight

RSSI: Received Signal Strength Indicator

PDR: Pedestrian Dead Reckoning

IDE: Integrated Development Environment

GCS: Global Coordinate System

NLOS: Non-line-of-sight

LOS: Line-of-sight

AWGN: Additive White Gaussian Noise

SNR: Signal to Noise Ratio