UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

SMARTPHONE RELATIVE DISTANCE BASED ON RF SIGNAL LEVEL

RADIUS

Guillermo Ortas Delgado

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

Advisor:

ILKER SEYFETTIN DEMIRKOL

In partial fulfilment of the requirements for the degree in

TELECOMMUNICATION SYSTEMS ENGINEERING

Barcelona, June 2016

Abstract

"Show me the location of another phone relative to mine". This is a collaborative degree thesis project between my colleague Nèstor Bonjorn and myself that aims to locate a peer relative to the user. The point of this work is to do so without drawing upon an Internet connection or a GPS signal. With this original idea in mind the challenge was set: to compute the distance and direction of a peer through an Android application using every useful embedded sensor of the users' smartphones (magnetometer, accelerometer, gyroscope, light sensor, barometer…) to this goal, their radio chips and some signal processing.

The result of the project is an Android application that is able to accurately track, draw and record the path taken by a user by means of sensor fusion, compute the distance and direction between peers using RSSI-based approximations with limited resolution, provide a chat-like service to exchange messages and calculate very precise distances using audio signals' time of flight.

Resum

"Mostra'm la ubicació d'un altre telèfon respecte al meu". Aquest és un projecte de fi de grau col·laboratiu entre el meu company Nèstor Bonjorn i jo que tracta de trobar la posició d'un altre relatiu a l'usuari. La gràcia d'aquest treball és aconseguir-ho sense recórrer a una connexió a internet o a un senyal GPS. Amb aquesta idea interioritzada el repte estava sobre la taula: calcular la posició i direcció d'un usuari distant a través d'una aplicació per Android que faci servir qualsevol sensor integrat al telèfon (magnetòmetre, acceleròmetre, giroscopi, sensor de llum, baròmetre...) que ajudi a aconseguir-ho, els seus chips de ràdio i una mica de processament de senyal.

El resultat del projecte és una aplicació per Android que és capaç de seguir i dibuixar la trajectòria d'un usuari de forma precisa i enregistrar-la per mitjà de fusió de sensors, calcular la distància i direcció entre parells d'usuaris fent servir aproximacions basades en RSSI amb precisió limitada, proporcionar un servei d'intercanvi de missatges i calcular distàncies mitjançant temps de vol de senyals d'àudio amb gran resolució.

Resumen

"Muéstrame la ubicación de otro teléfono respecto al mío". Este es un proyecto de fin de carrera colaborativo entre mi compañero Nèstor Bonjorn y yo que trata sobre encontrar la posición de otro respecto al usuario. La gracia de este trabajo es lograrlo sin recurrir a una conexión a internet o una señal GPS. Con esta idea original en mente, el reto estaba sobre la mesa: calcular la distancia y dirección de un usuario distante a través de una aplicación para Android que use cualquier sensor integrado en el teléfono (magnetómetro, acelerómetro, giroscopio, sensor de luz, barómetro…) que ayude a conseguirlo, sus chips de radio y un poco de procesado de señal.

El resultado del proyecto es una aplicación para Android que es capaz de rastrear, dibujar y registrar la ruta del usuario de forma precisa por fusión de sensores, calcular la distancia y dirección entre dos usuarios utilizando aproximaciones basadas en RSSI con precisión limitada, proporcionar un servicio de intercambio de mensajes y calcular distancias con gran resolución midiendo el tiempo de vuelo de señales de audio.

Acknowledgements and dedication

First and foremost I want to thank my friend Nèstor, whom of course this project wouldn't had been possible without. We worked closely and continually helped and supplemented each other with the other part of this joint work. It's always been a pleasure to work with him.

I also want to thank my advisor Ilker Demirkol for giving us the opportunity to work in this amazing topic, for his advice and support and for the workplace he gave us during the semester to develop the idea.

Last but not least I also have to thank my family and friends that gave me support during the way and helped me explore, develop and widen the original idea, with special mention to Marta, who was tremendously supportive all along and who drew the logo of the project for me.

Revision history and approval record

Revision	Date	Purpose
0	27/05/2016	Document creation
1	06/06/2016	Document revision
2	17/06/2016	Document revision
3	25/06/2016	Document revision

DOCUMENT DISTRIBUTION LIST

Name	E-mail
Guillermo Ortas	guilleortas@gmail.com
Nèstor Bonjorn	nestorbonjorn@gmail.com
Ilker Demirkol	ilker.demirkol@entel.upc.edu

Written by:		Reviewed and approved by:	
Date	27/05/2016	Date	25/06/2016
Name	Guillermo Ortas	Name	Ilker Demirkol
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

1. Introduction

The mobile era takes connectivity to the next level, society and especially young people live in a connected and always-on state that leads them to think that a mobile device is rendered useless without cell service or access to the Internet. But would you really be completely isolated from the rest of the world in that case?

The original idea was proposed by the thesis advisor, Ilker Demirkol, and it consisted on locating users relative to each other via their smartphones. When the thesis was being planned, it was proposed to do so without the need of a GPS signal, cell service or Internet access, endowing it with a new dimension of possible applications.

This is a new and independent project, so it is not based upon the foundation of any other and it is not part of a department or company research, so it was developed from the ground up. The project has been given the name RADIUS.

1.1. Statement of purpose and goals

The distinguishing point of this work is exactly that, the positioning is done using only the embedded sensors of the users' smartphones and any other vastly available components in consumer smartphones that are useful to this objective. The main drawback of this approach is that the range is limited to local vicinity, this issue will be analysed and quantified later on.

This idea would be great for applications of short range but with low visibility, with no GPS signal available (tall buildings or indoor scenarios) or in a situation where it is simply not preferred due to battery consumption or other reasons. Trying to find someone in a mall or in a social event where there is a large density of people or in a scenario in which the cellular networks become saturated are some cases of use. Another possible application could aim to locate victims or missing people in a natural catastrophe or in other emergency situations with no cellular service. A more different approach would be to help blind people navigate the city, help them identify objects or get in touch with volunteers or other unseeing pedestrians around.

These are the thesis main goals, they will be expanded and described in detail in the work plan:

i. Establish a communication between users using Wi-Fi Direct or Bluetooth.
ii. Acquire RSSI values from the communication link.
iii. Compute the distance to the remote user and estimate its direction based on said computations.
iv. Create a messaging functionality to help the users meet each other.
v. Achieve all previous objectives in the conditions described above, as if the smartphones were located in an anechoic chamber and do everything in the context of an Android application that is easy to use on the go.

1.2. Requirements and specifications

The final product should be in the form of an Android app that is able to track the user's movement and trajectory and to give a relative distance and direction estimations of a distant user in an understandable and user friendly interface.

The application should not require more than an initial interaction to be set up and the user should not need be required to input any more information after that in order to use it. The application has to be installed in all the involved phones but it should not need to be running necessarily in the foreground at the same time, it should even work if the smartphone locked.

These requirements also enable other functionalities not directly related to finding a peer, like measuring the length of an object or tracing the route taken in a museum or a shop. This last application is a marketing strategy currently being implemented in clothing companies amongst others[1], it will be expanded in the review of literature.

The performance specifications include a maximum orientation resolution of 15º, a maximum distance resolution of 3 m and, for logistical reasons, the application should not exceed 10 MB in size. These values were chosen because it was determined that they were enough so that the task of finding a peer in the proposed scenarios can be easily carried out.

1.3. Methods and procedures

To start with, the peers must be able to communicate with one another to exchange information about their relative location. In this project two alternatives are considered to achieve this: Wi-Fi Direct and Bluetooth.

Wi-Fi Direct is a novel Wi-Fi Alliance standardised wireless communications protocol that enables devices to connect with each other without requiring a physical wireless access point[2]. It can be used to exchange data between devices or to browse the Internet at typical Wi-Fi speeds, even with more than one device simultaneously[3,4]. Even though the vast majority of modern smartphones support Wi-Fi Direct, only one of the Wi-Fi devices involved in the connection is required to be compliant with the

[1] Philipp Schloter, Hamid Aghajan. "Wireless RFID Networks for Real-Time Customer Relationship Management". Department of Electrical Engineering, Stanford University and Detecon, Inc. [Online] Available: http://wsnl2.stanford.edu/papers/wireless_RFID_networks_for_RTCRM_Final_V06.pdf. [Accessed: 18 June 2016].
[2] Wi-Fi Alliance. "Wi-Fi Direct". Discover Wi-Fi. [Online] Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct. [Accessed: 13 June 2016].
[3] Wi-Fi Alliance. "How fast is Wi-Fi Direct?" Knowledge centre, FAQ. [Online] Available: http://www.wi-fi.org/knowledge-center/faq/how-fast-is-wi-fi-direct. [Accessed: 13 June 2016].
[4] Wi-Fi Alliance. "Can a network based on devices certified under the Wi-Fi Direct program cross connect to an infrastructure network for internet connectivity?" Knowledge centre, FAQ. [Online] Available: https://web.archive.org/web/20130403025145/http://www.wi-fi.org/knowledge-center/faq/can-network-based-devices-certified-under-wi-fi-direct-program-cross-connect. [Accessed: 9 December 2013].

standard to establish a peer-to-peer connection, so compatibility is not a problem at all.

Although Bluetooth is currently ubiquitous in consumer electronics and it has recently been greatly updated and upgraded, the range and data transfer capabilities it offers cannot compete with the newer Wi-Fi Direct standard, as it is not intended to cover the needs of high performance networking or larger distances. A performance comparison is done in the project development section.

1.3.1. Input sources and use

- Wi-Fi and Bluetooth signals RSSI for distance estimation.
- Speaker and microphone for ToF distance computation.
- Three axis accelerometer for obtaining velocity and movement by means of integration.
- Magnetometer for obtaining the Earth's magnetic field as a frame of reference as well as the pitch and roll of the device.
- Three axis gyroscope for acquiring the rotation speed in any direction. By combining it with the accelerometer and magnetometer a better estimation can be given of the total user movement and its tracking is made more accurate.

Some input candidates have been disregarded for the lack of presence in the market of the components they use or for their infeasibility of implementation:

- ToF-based distance computation using RF signals. Timescale too fast to be correctly measured in a mobile device, studied in the project methodology.
- Magnetometer for distance approximation using Earth's magnetic field measurements correlation. Earth's magnetic field varies too little in the range of our interest —as seen as seen in the image below— and it's also dependent on altitude and time.

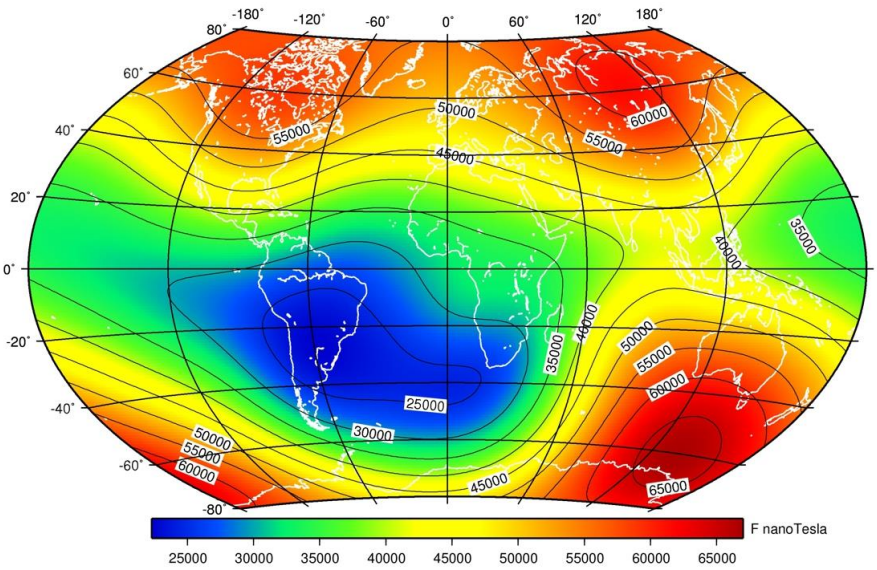

Figure 1: Map of the total Earth's magnetic field intensity in 2015. British Geological Survey.

- Barometer for speed of sound adjustment using the atmospheric pressure and peer positioning in multiple levels. Component too scarce to rely upon.
- Speaker and microphone for getting a signal strength-based distance estimation. Strongly limited by ambient noise.
- Light sensor in conjunction with the smartphone flashlight to give a distance estimate in controlled conditions. Strongly limited by ambient light.
- Pattern recognition using the camera to measure the distance to the objective. Limited to camera resolution and LoS only.

Once a method is chosen to get a distance measurement or estimation, the basic procedure is as follows:

1. Communication establishment between peers.
2. Distance acquisition.
3. Device movement in any direction.
4. Acquire distance again.
5. Compute relative position and direction.
6. Repeat steps 3, 4 and 5 in loop.

1.4. Work plan and assignment of tasks

The work plan has been broken down into individual work packages, since this is a collaborative project, the responsible of each block is specified.

1.4.1. Work packages

Project: RADIUS	Responsible: Both	WP ref: 1
Major constituent: Research and benchmarking		Sheet 1 of 8
Short description: Research on similar projects and mobile apps that carry out a similar task. Continuous block.		Planned start date: 08/02/2016 Planned end date: 01/06/2016
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 application		Sheet 2 of 8

Short description: Test app development block. It consists of developing an app that reads all of the smartphone's sensors. It also serves the purpose of getting familiarised with the Android programing environment.	Planned start date: 15/02/2016 Planned end date: 29/02/2016
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: **Guillermo**	WP ref: 3
Major constituent: Wi-Fi Direct communications		Sheet 3 of 8
Short description: Design a communications protocol between two devices based on Wi-Fi Direct connectivity. It includes string and command passing.		Planned start date: 02/03/2016 Planned end date: 09/05/2016
Internal task T1: Research on papers and projects that have already used this connectivity **Internal task T2:** Design communications protocol through an app, code writing and interface design **Internal task T3:** Testing		

Project: RADIUS	Responsible: **Guillermo**	WP ref: 4
Major constituent: Bluetooth communications		Sheet 4 of 8
Short description: Design a communications protocol between two devices using Bluetooth and compare its performance to the Wi-Fi Direct counterpart.		Planned start date: 05/05/2016 Planned end date: 27/05/2016
Internal task T1: Research on papers and projects that have already used this connectivity		

Internal task T2: Design communications protocol through an app, code writing and interface design **Internal task T3:** Testing

Project: RADIUS	Responsible: **Nèstor**	WP ref: 5
Major constituent: Kalman Filter		Sheet 5 of 8
Short description: Implement the Kalman Filter (signal processing) to improve the computed estimations.		Planned start date: 02/03/2016 Planned end date: 27/05/2016
Internal task T1: Research on papers and projects that have already used this filter for other applications **Internal task T2:** Design **Internal task T3:** Simulation **Internal task T4:** Testing		

Project: RADIUS	Responsible: **Guillermo** (RSSI) **Nèstor** (sound ToF)	WP ref: 6
Major constituent: Distance estimation		Sheet 6 of 8
Short description: Compute the distance between two devices using different systems: RSSI and sound waves time of flight (ToF). Estimate the direction based on the distance computation.		Planned start date: 14/03/2016 Planned end date: 27/05/2016
Internal task T1: Using RSSI **Internal task T2:** Using ToF **Internal task T3:** Using camera		

Project: RADIUS	Responsible: **Nèstor**	WP ref: 7
Major constituent: User movement monitoring		Sheet 7 of 8

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 to calculate the relative direction, and hence the relative position, between the two devices.	Planned start date: 23/02/2016 Planned end date: 18/04/2016
Internal task T1: Parameter acquisition **Internal task T2:** Computation	

Project: RADIUS	Responsible: Both	WP ref: 8
Major constituent: Develop final app		Sheet 8 of 8
Short description: Final app development block: it includes all previous calculations		Planned start date: 09/05/2016 Planned end date: 13/06/2016
Internal task T1: Interface design **Internal task T2:** Code writing **Internal task T3:** Testing		

1.4.2. Milestones

WP#	Task#	Short title	Milestone / deliverable	Date
2	4	Code writing	Basic sensor reading Android app	26/02/2016
3	1	Communications protocol	Wi-Fi Direct protocol design	25/04/2016
4	1	Communications protocol	Bluetooth protocol design	27/05/2016
5	2	Processing/tracking	Kalman Filter implementation	25/04/2016
-	-	CDR	Critical Design Review	09/05/2016
6	-	RSSI/sound ToF	Distance and direction estimation	03/05/2016

8	2	Code writing	Final Android app	09/05/2016
-	-	FR	Final Report	27/06/2016
-	-	Thesis defence	Thesis defence	15/07/2016

1.4.3. Work breakdown structure

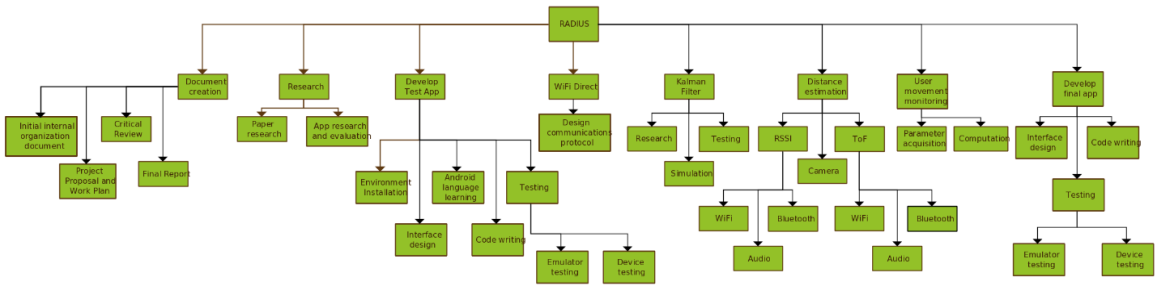

Figure 2: Work breakdown structure (opens in a web explorer)

1.4.4. Gantt diagram

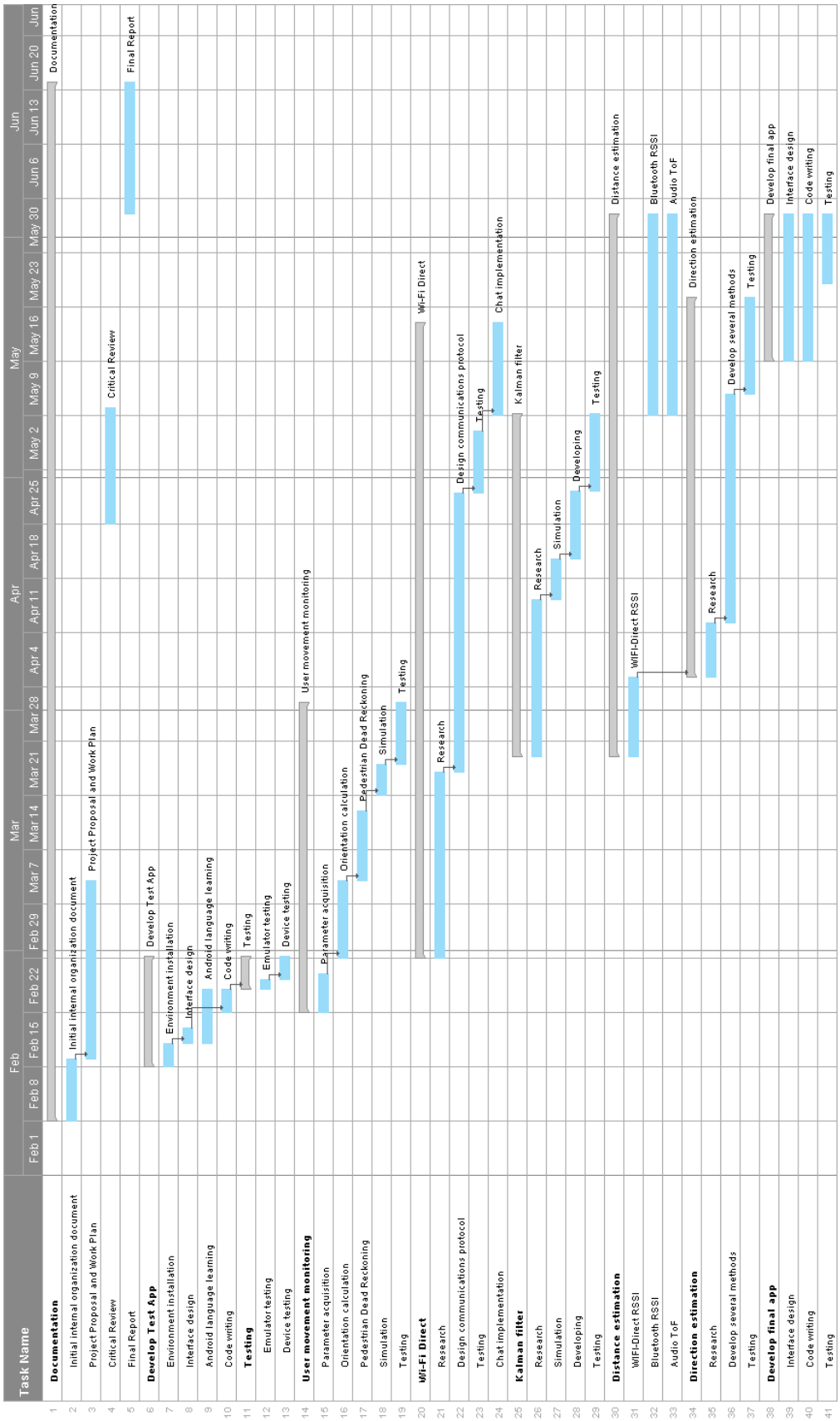

Figure 3: Gantt diagram

1.5. Deviations and incidences

The work plan developed smoothly and without any incidences during the project until the RSSI acquiring part. The original idea was to constantly update RSSI values at a rate high enough to counter the effects of shadowing and the varying nature of the RF channel. This is possible and in fact very easy to achieve using regular Wi-Fi connectivity, but not using Wi-Fi Direct, due to the way Google implements the manager for this kind of communication. They argue that these readings would not be reliable and so, RSSI values can only be obtained when a scan is performed.

A scan takes about one to five seconds, which is too slow to rely on a distance calculated in this way. An alternative was then devised: the use of Bluetooth in the same way to achieve a higher rate of RSSI samples. Bluetooth was not the first option because of its shorter range and slower transfer speeds (a detailed comparison is made in the methodology section), but it promised a better distance estimations at this point. It was found that Bluetooth only gives RSSI information when a scan is performed as well.

At this stage, finding a faster way of doing a scan was the priority. A command-base procedure was discovered. A scan can be done in this way in less than a second, which is a much shorter and definite time, making the initial objective feasible. However, it presents a large drawback: only users with access to the system partition of their device can use it. This special access is obtained with a process called rooting, which only some power users do. That means this way of estimating the distance is not available to the general public.

ToF-based distance estimation, though, is available to everyone and can potentially offer a much better precision and reliability. However, it presents the challenge of being able to measure times in the order of nanoseconds for the distance range of interest without specialised hardware.

2. State of the art

2.1. Similar applications and projects

Currently, there are a lot of "friend locator" apps that claim to work without using GPS or Internet access, just by using mobile networks, and they do. This problem was solved by Google in 2007 when an update was introduced to Google Maps that gave it the ability to compute the user's location by triangulating between cell towers[5]. Nowadays this technology has been polished and the accuracy has been upgraded to the point of being able to find a friend in close range thanks to the improved precision that fixed Wi-Fi hotspot identification and Bluetooth device recognition enables.

There are also some apps that enable communication without using cell service either. These apps offer chat and file transfer functionalities but their scope is limited to just that, with no interest in the location of the users. In this project this idea is taken one step further. Its global objective is to locate a phone using no mobile networks at all (no cell service or GPS signal). As stated in the introduction, only the embedded sensors and communications chips of a modern smartphone are used to accomplish this.

2.2. Current research

There is a project currently being developed called Estimote that is able to precisely locate users in real time using Bluetooth. The main drawback is that it only works in known indoor spaces and requires specialised hardware to be set up beforehand (iBeacons), so it's not very practical for the use case explored in this thesis.

As commented in the introduction, retailers are interested in following the path taken by clients in their shops. This strategy delivers important information about their stores and how they should arrange them to maximize sales or how to promote a specific product giving it more visibility. This is the kind of application Estimote would apply best to. Others use RFID tags[6,7] instead of iBeacons in a very similar approach to help locate customers in a retail store.

There have been quite a lot of paper publications lately that aim to locate users in the conditions tackled by this thesis, but in an indoor scenario, not in the exterior. This is to be expected, as a GPS signal is almost always available outdoors, unless it's an area

[5] Kate Greene. "Finding Yourself without GPS". MIT Technology Review, 2007. [Online] Available: https://www.technologyreview.com/s/409138/finding-yourself-without-gps/. [Accessed: 13 June 2016].
[6] Marin Vukovic, Ignac Lovrek, Hrvoje Kraljevic. "Discovering shoppers' journey in retail environment by using RFID". University of Zagreb, Faculty of Electrical Engineering and Computing. [Online] Available: https://bib.irb.hr/datoteka/596198.FAIA243-08571.pdf [Accessed 18 June 2016].
[7] In-Chul Jung, Young S. Kwon. "Grocery Customer Behavior Analysis using RFID-based Shopping Paths Data". World Academy of Science, Engineering and Technology, 2011. [Online] Available: http://waset.org/publications/13460/grocery-customer-behavior-analysis-using-rfid-based-shopping-paths-data. [Accessed 18 June 2016].

without coverage or it is not desired to use GPS for power consumption issues or for any other reasons.

Some of these research publications state that their position estimation system relies on mobile sensor fusion (the same RADUIS uses to track the users' movements) and others on pre-installed hardware in the room that emit signals of interest. Examples of the first kind are "Indoor localization without infrastructure using the acoustic background spectrum" and "Virtual Compass: Relative Positioning To Sense Mobile Social Interactions". Some of the second kind are "Ultrasonic Time Synchronization and Ranging on Smartphones" and "Discovering shoppers' journey in retail environment by using RFID". These papers are listed in the bibliography at the end of this report, except the last one, which is the one referenced in Footnote 6.

3. Methodology

In this section, the developed communications protocol (establishment, commands…) and the distance computation procedures will be explained.

3.1. Wi-Fi Direct communications protocol description

3.1.1. Parts involved

There are several software parts that make up the application developed for this project, the ones that intervene in the communications protocol will be described next. All technical words are explained in the glossary at the end of this report.

3.1.1.1. WiFiDirectActivity

The activity that uses Google's Wi-Fi Direct API to discover and connect with nearby peers.

3.1.1.2. DeviceDetailFragment

A fragment that manages the interaction with a particular peer, it's the component that establishes the connection and manages incoming message transfer requests.

3.1.1.3. DeviceListFragment

A fragment that shows discovered peers and their status and that passes the interaction events to the WiFiDirectActivity for being handled.

3.1.1.4. resultHandler

The component that manages the received commands form other peers, it runs and/or calls the corresponding methods to execute the desired action. It is not a native Android handler.

3.1.1.5. MessageTransferService

A service that manages outgoing message transfer requests.

3.1.1.6. WiFiDirectBroadcastReceiver

A BroadcastReceiver that listens to Wi-Fi Direct events (status changes) and reports them.

3.1.2. Connection establishment

The first step is to perform a scan, which delivers a list of the current nearby peers in the form of WifiP2pDevice objects. When a user taps on the connect button of a peer, the DeviceDetailFragment sets up a WifiP2pConfig object and tells the WifiP2pManager to connect both peers through the WiFiDirectActivity and a group owner is decided, all of this is mainly done by the Wi-Fi Direct API itself.

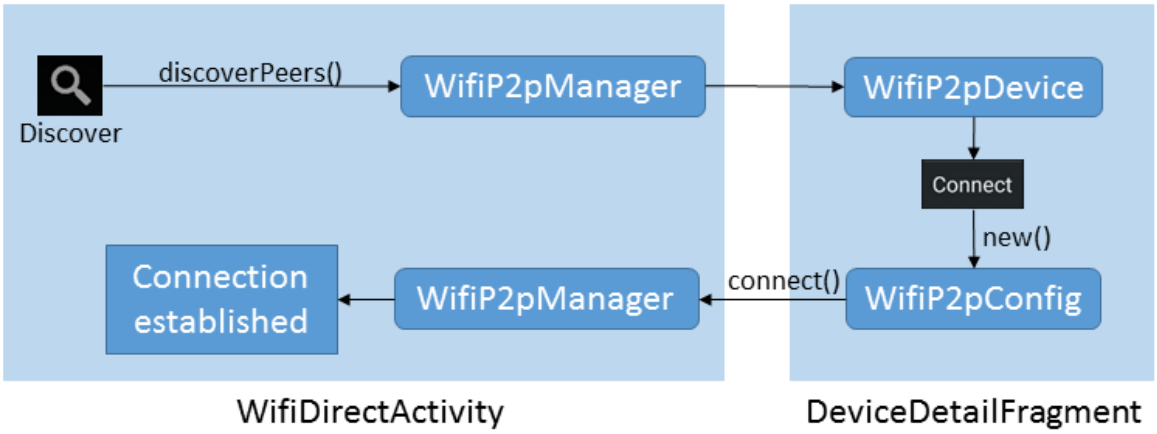

Figure 4: Connection establishment diagram

When the connection is established, the group owner IP address —which is known at this point— is saved for later use, then the client sends a dummy text message to the server for it to register its address as well. Finally, a MessageAsyncTask is started in both sides of the communication. This looping background task allows bidirectional communication between peers through TCP sockets.

3.1.3. String and command passing

When a user wants to send a message, a MessageTransferService is created and started. This service opens a socket that connects to the receiving peer, which is accepted by the MessageAsyncTask running in the background of it. When an incoming message is received, the MessageAsyncTask delivers it to the resultHandler and restarts itself to listen for new messages.

The resultHandler then analyses the message header and determines whether it's simply string of text to be displayed or is a command to perform some other action. It triggers tasks like the audio ToF-based distance measurement process or requests information to the other peer. The core methods and code that establishes the connection and enables exchange of information is attached in the appendix.

3.2. RSSI-based computation

3.2.1. Bluetooth vs Wi-Fi Direct: speed and range comparison

Both of these technologies can be used to obtain RSSI values for distance calculation, so the first thing to do is to choose one of them or work with both in harmony.

As stated in the introduction, Wi-Fi Direct uses the same hardware as regular Wi-Fi, which means that the same speeds can be achieved. Both Wi-Fi Direct and Bluetooth speeds vary depending on the standard they use. For this comparison, the 802.11n Wi-Fi standard and the 4.0 version of Bluetooth will be regarded.

802.11n offers a maximum net data rate of up to 600 Mbps, while 4.0 Bluetooth can reach a peak transmission speed of 24 Mbps, a much lower link speed. The maximum Wi-Fi Direct range of almost 200 m puts it ahead of Bluetooth, with only 60 m of

range[8,9]. For this application, range is very important and should be prioritised, which means it's a decisive factor in this case, meaning that the clear winner here is Wi-Fi Direct, as it outperforms Bluetooth in both fields.

Bluetooth was only considered to be an option when it was found that by using Wi-Fi Direct, RSSI values could only be obtained by doing a scan, and not ad lib in any moment and with the desired polling rate as initially though. This is important to iron out the unstable nature of the RF channel. However, Bluetooth presented the same drawback, so the result of the comparison remained unchanged.

Both standards are supported by almost all current smartphones made in 2010 and later, as both technologies were presented around that year. The quoted data rates and ranges are theoretical figures, but serve the purpose of a qualitative comparison. Security and power consumption are excluded from this benchmarking. Range results from real-life Wi-Fi Direct tests will be exposed in the results section.

3.2.2. Value acquisition procedure

To begin with, a scan must be performed, there's no need for the other peer to accept an invitation or be connected at all to do so. As explained in the connection establishment headland, the scanning action returns a list of WifiP2pDevice objects, which contain the RSSI value of each one, although somewhat hidden, obtaining a RSSI reading is not made easy by the Wi-Fi Direct API. The signal level figure is buried in the return statement of the `toString()` method called on the desired WifiP2pDevice object. This information can then be extracted with some fiddling and used to estimate the distance.

3.2.2.1. Alternative method

Having met the problem that doing a scan takes more time than the required to get a good enough RSSI sampling rate to achieve a reliable and realistic distance conversion, another way of performing it that takes less time was researched. It exists the possibility of going under the API to do a scan in a faster way that can achieve exactly that objective: allow a higher RSSI value polling rate that would increase the robustness of the algorithm.

It is a command-based solution that uses a cross-platform network entity called wpa_supplicant that is executed in the device itself. It implements an interface that can be used to perform a Wi-Fi Direct scan. The ADB Shell is used to run the following commands in the smartphone from a computer. To set up the supplicant, a configuration file must be created to grant the proper rights, which is done by setting

[8] IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput. IEEE Standard 802.11n-2009.
[9] IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN). IEEE Standard 802.15.1-2005.

the update_config flag to 1. After that, the daemon is started and a tool called wpa_cli is called to be able to execute the wpa_supplicant commands[10].

The command P2P_FIND looks for peers, this is the equivalent of performing a scan though the API, the P2P_STOP_FIND finishes the discovery. Then, all of the information of the discovered peers can be accessed by the P2P_PEER command, which returns the same information as the `toString()` method of a WifiP2pDevice object. This is the command that returns the RSSI value in dBm under the tag "level". In the screenshot below, a RSSI value of -36 dBm was captured.

These commands are the ones that are ultimately executed through the API, that is the reason why this is a faster way of doing the same thing, because it bypasses all of the above layers.

[10] ArchLinux. "WPA supplicant". ArchLinux Wiki. [Online]. Available: https://wiki.archlinux.org/index.php/WPA_supplicant. [Accessed 4 June 2016].

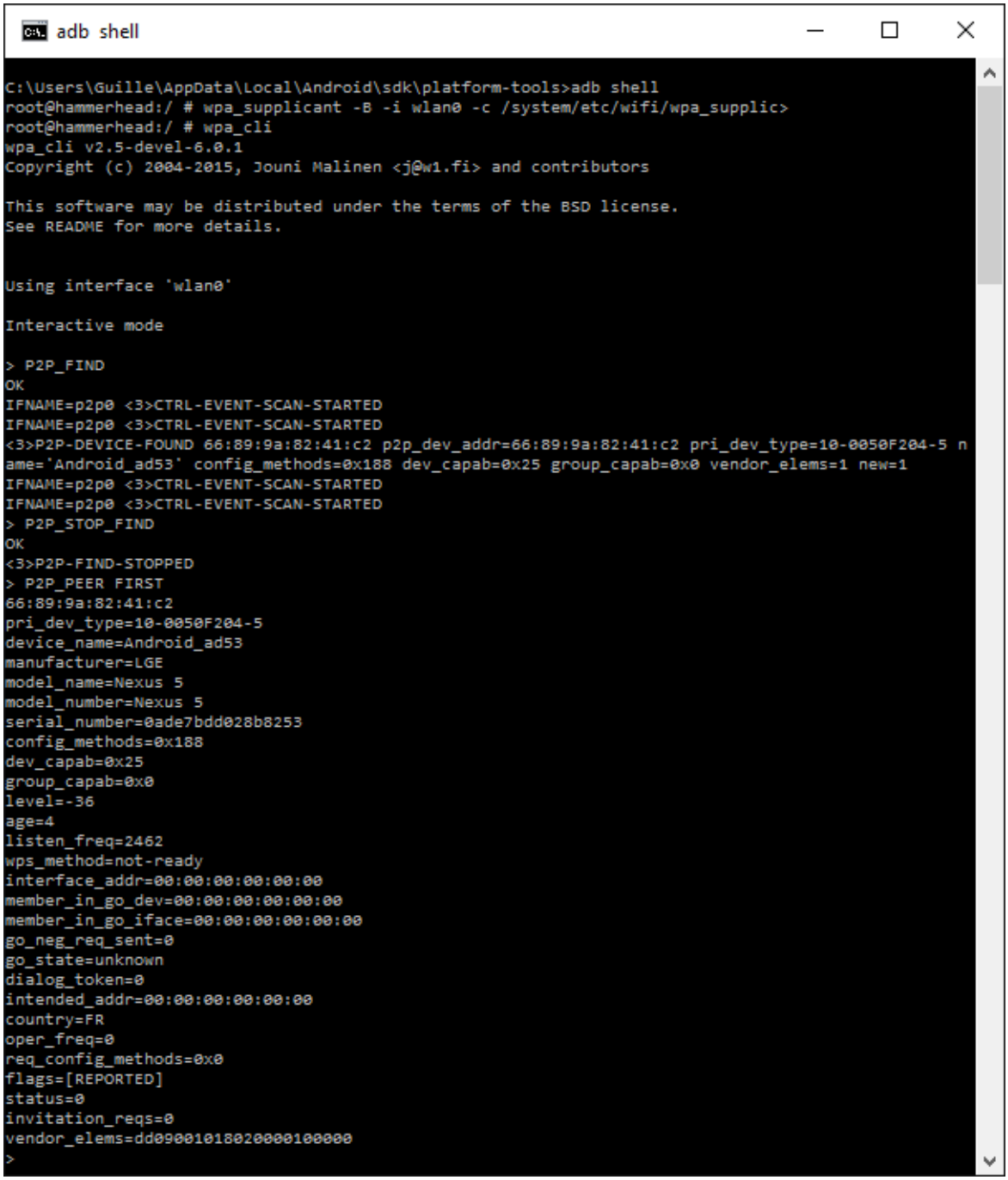

Figure 5: Wi-Fi Direct scan performed using wpa_supplicant

The use of this supplicant, though, requires root privileges on the Android device that wants to run it. As explained in the introduction, root access is beyond the reach of a regular user —at whom this application is targeted—, so this alternative RSSI reading method is immediately disregarded by this disadvantage as it is not possible to be overcome.

3.2.3. Distance conversion

The log-distance path loss model is used to assess the Wi-Fi Direct signal attenuation and translate RSSI values into distance magnitudes, which is given by the following expression:

$$P_r(d) = P_r(d_0) - 10\alpha \log_{10}\left(\frac{d}{d_0}\right) + X_{\sigma_r} \qquad (3.1)$$

$$with \quad d_0 = 1\ m, \quad P_r(d_0) = -37.6\ dBm, \quad E\left[X_{\sigma_r}\right] = 0, \quad \sigma_r = 6\ dB \quad and \quad \alpha = 2.3$$

The received signal strength is equivalent to the received power $P_r(d)$ and X_{σ_r} is a random Gaussian variable of null mean and σ_r standard deviation. The reference d_0 was set to 1 m and the RSSI was measured 30 times at that distance to get a mean $P_r(d_0)$ value of -37.6 dBm to calibrate the algorithm. These measurements were done in a large furnished indoor location using two LG Nexus 5 as reference devices, so the constants α and σ_r were chosen accordingly[11].

Since the RSSI reading rate is not very high, there is an algorithm built into the application that dismisses "unlucky" values using a weight system in order to provide a better distance approximation. When a value —or series of values— is unexpectedly low, it is not taken into account towards the computation, as it could have been induced by shadowing or by the user standing in an area of lower visibility. This algorithm and the logic behind the direction approximation computations and their code implementations are outside the scope of this report, as it is part of Nèstor's.

3.3. ToF-based computation

3.3.1. Previous study

To begin with, a study was made to determine the feasibility of measuring the time of flight of RF waves. Since the two smartphones involved in a measure are not required to be connected to the Internet for the application to work, they cannot be synchronized with a time server, which means that a computation based on a timestamp is not possible.

[11] Theodore. S. Rappaport. *Wireless communications: principles and practices*, 2nd ed. Prentice-Hall, 2002.

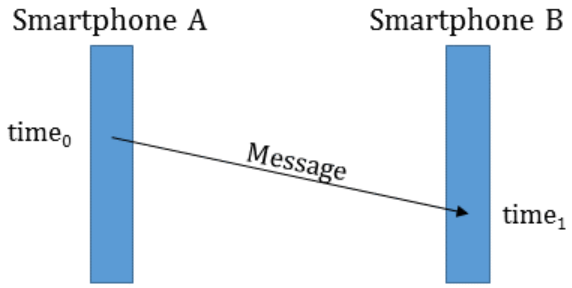

Figure 6: Timestamp-reading ToF using synchronized devices

If both devices had Internet access, they could use the timestamp-reading strategy, the time the message was sent could be read from it in the receiving smartphone and subtracted from the current time, resulting in the time it took the signal to travel:

$$ToF = time_1 - time_0 \tag{3.2}$$

Timestamp-reading is the simplest way of measuring RF ToF, which is not possible in this application.

Time of flight has to be calculated using a reply strategy, in which the devices do not need to be synchronized. In order for this idea to work, the processing delay must be perfectly known and controlled. The processing delay is defined as $time_2 - time_1$.

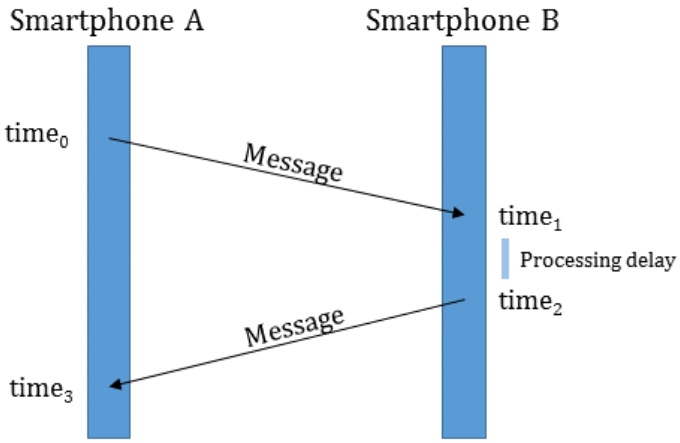

Figure 7: Message reply ToF using unsynchronized devices

With this strategy, the time of flight is calculated by this expression:

$$ToF = \frac{time_3 - time_0 - processing\ delay}{2} \tag{3.3}$$

At this point the next step is to check if the delay can be controlled with enough precision to reach a reasonable distance granularity. Ideally, the delay would be null,

but that is not possible. A known time in both sides of the communication is then set as the delay time, following this diagram:

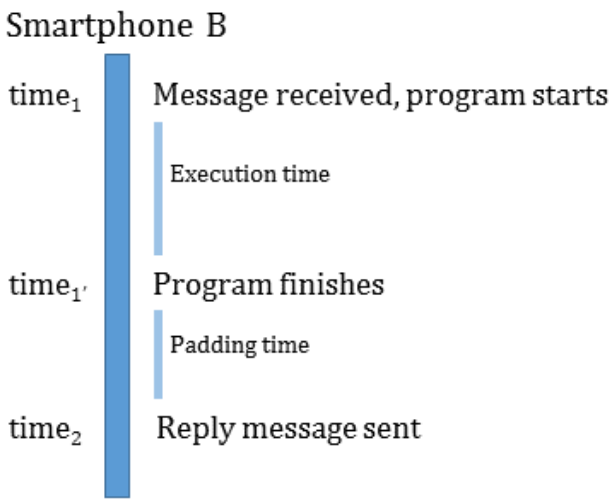

Figure 8: Message reply ToF delay control

The total delay time, $time_2 - time_1$, can be set to a known value if the padding time is adaptively set to $time_2 - time_{1'}$. The delay time has to be chosen to be greater than the execution time but not needlessly big, a compromise has to be made.

Following this strategy, a preliminary simulation using Python language was made to check the viability, and then a more refined one using Java.

3.3.1.1. Python delay time control simulation

Python is a high-level, interpreted programming language that is fast and easy to get a general idea of a problem approach, great for this kind of use cases. That does not mean it's not a serious programming language, as it is suitable to build large scale programs and it is widely used as so.

The program follows the basic idea depicted in Figure 8 that dynamically sets the padding time to establish the total delay time to the desired value. A Monte Carlo simulation was written using the `time.clock()` method to measure time. It is the preferred choice for Python code benchmarking as it offers a resolution of less than a microsecond[12], which is good enough for a first approach and evaluation.

A desired delay time was set to 100 ms and a random loop that takes 10 to 20 ms to finish was used as a dummy load to represent real computations. The simulation was repeated $N = 10000$ times and the errors in real vs desired delay time were measured and stored in a vector x, the following histogram represents the distribution of said vector.

[12] Python Software Foundation. The Python Standard Library, Generic Operating System Services. [Online]. Available: https://docs.python.org/2/library/time.html#time.clock [Accessed: 18 June 2016]

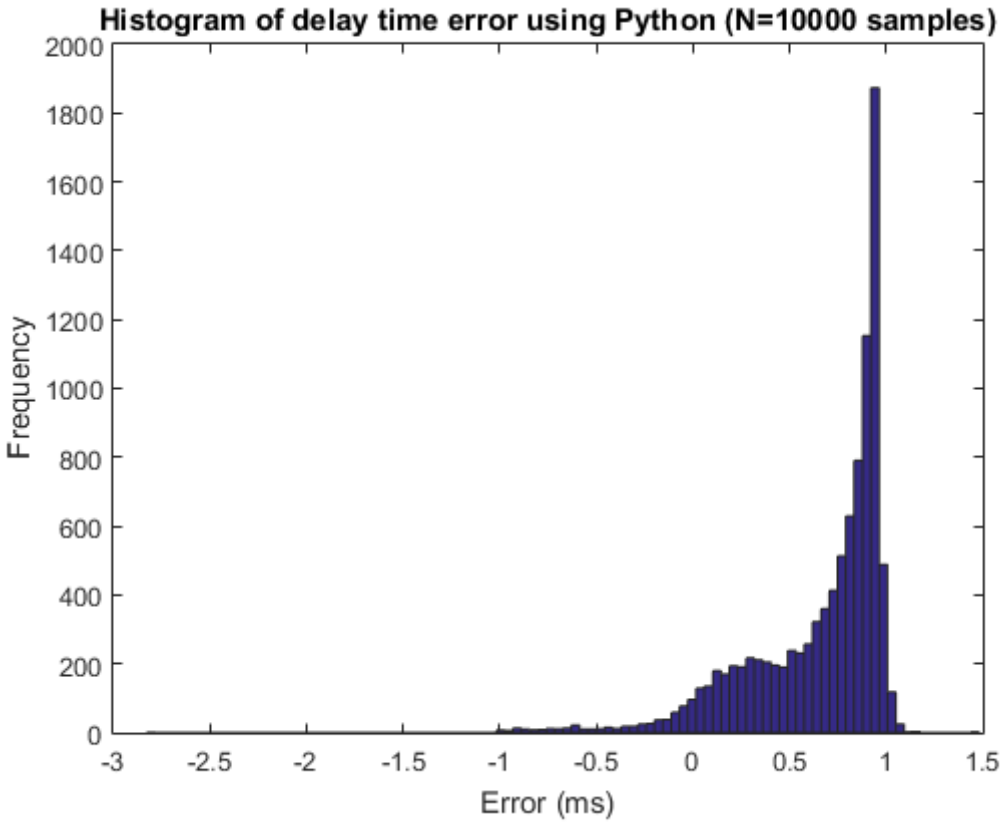

Figure 9: Histogram of delay time error using Pyhton (N=10000 samples)

Negative error times are possible, they just mean that the total processing took less time than the established value. There's a large spike near 1 ms that lowers when approaching zero, there's also a small bulge around 300 µs. The mean error is 655.6 µs and the distribution has a standard deviation of 353.3 µs, computed as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|x_i - \mu|^2} \qquad with \qquad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad (3.4)$$

These results show that the error is too big to be handled. In the best case, this method would give a time resolution of µ, which corresponds to a distance of almost 200 km at the speed of light and that is obviously unacceptable.

To get a distance resolution of 30 cm, a maxim error of a nanosecond is required. The same study using Java language is done next to see if it brings any improvements.

3.3.1.2. Java delay time control simulation

The next step is to analyse the same problem using Java programming language, which is basically the same as the Android language in which a hypothetical solution would be running on. This gives the results more verisimilitude. The program follows the same logic described in the Python section, but in this case the method used to record times is `System.nanoTime()`, which is more precise than the Python counterpart,

but still not great for a real implementation. Nevertheless, it offers a good idea of the viability, which is the purpose of this study. After running the simulation another 10000 times, these results emerge:

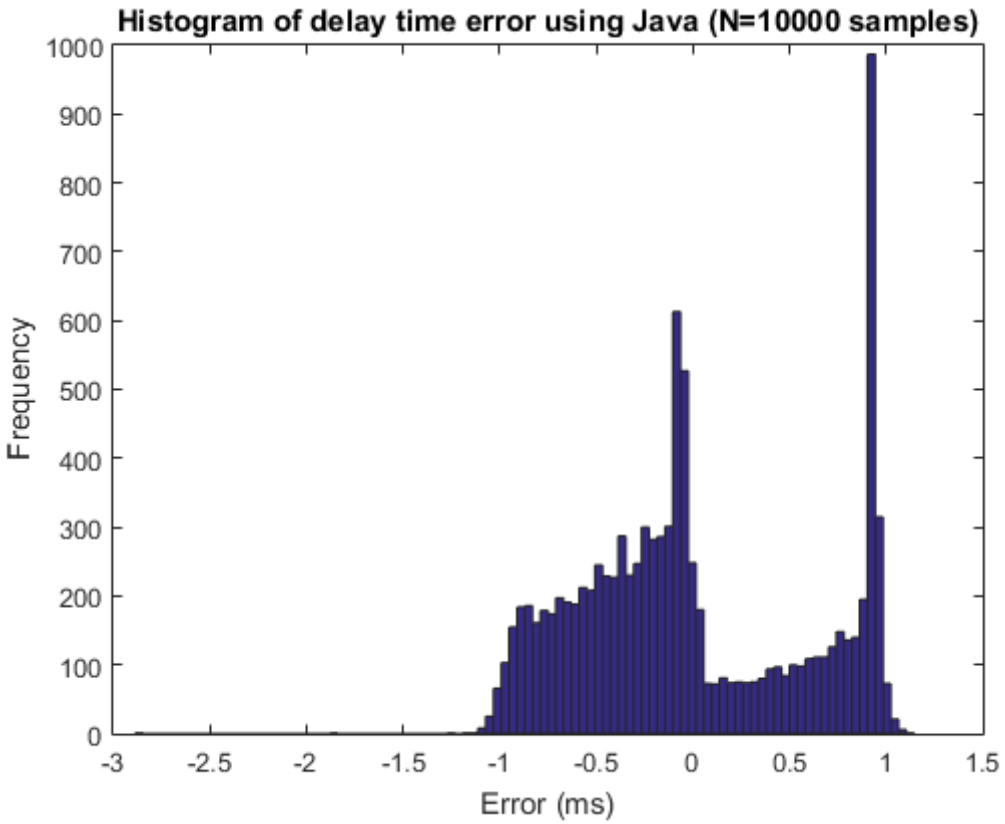

Figure 10: Histogram of delay time error using Java (N=10000 samples)

Two big spikes can be observed around 1ms and zero, with quite a lot of executions to the left of these values. This distribution has a mean of -3.262 µs and a standard deviation of 599.6 µs. That means the error in the delay time using Java language is much lower, but presents more dispersion. This solution would deliver a much better distance resolution (just under 1 km) but would be less predictable.

The code used for both simulations can be found in the appendix.

3.3.1.3. Conclusion

It's been proved that measuring RF waves' time of flight in these conditions is not a real option for this application. The Java implementation of the delay time control delivers a resolution that could be useful in other use cases, but not in this one, as 1 km is out of the Wi-Fi Direct range anyhow. Instead, measuring audio waves' time of flight is proposed, this will be explained in the following section.

3.3.2. Audio ToF protocol description

An acoustic ranging solution is proposed. Both smartphones emit a known signal in a short period of time, which is recorded and processed to find the instants at which the own signal and the remote one have been recorded. The audio samples between these events are counted and translated into time through the sampling frequency, this information is then passed to the other device. With both times available, a simple equation gives the distance between smartphones. The following diagram represents the process:

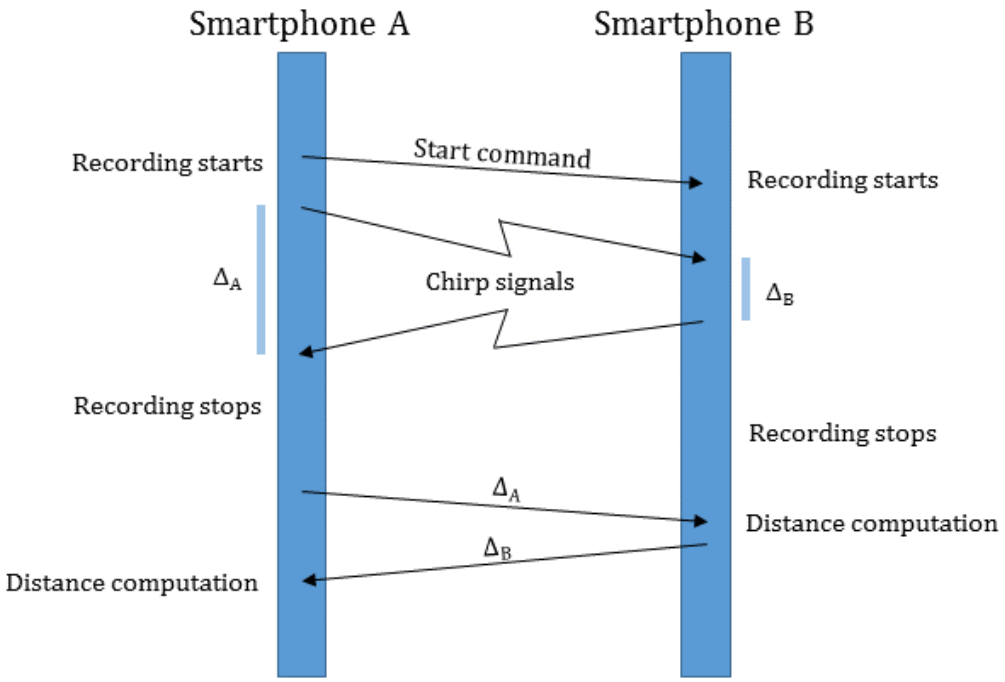

Figure 11: Audio ToF protocol

Note that there's not a particular order in which the signals have to be emitted, there is also no synchronization needed using this method and the processing delay does not alter the results, so there's no need to control it. The time of flight is computed in this way:

$$ToF = \frac{\left||\Delta_A| - |\Delta_B|\right|}{2} \qquad (3.5)$$

The absolute value of the times is taken because there's no guarantee of which signal will be sent first. The recording time is set in the protocol implementation to a fixed value low enough to be easily processed but sufficiently long to include the sent and received signals. This is a master-slave protocol, but it can be started in either device, so once the process is initialised by one of the devices, the other one shall not start it until it has been finished completely, that is why the app doesn't allow the user to start it again if the process is running.

The emitted audio wave is a chirp signal, which has great correlation properties to be easily identified even with background noise, which is typical of the situations this project applies to. The maximum theoretical distance resolution of this method is determined by the sampling frequency of the microphones, which is considered 44100 Hz, the industry standard:

$$\Delta d = \frac{c_s}{f_s} = 7.78\,mm \qquad with \begin{cases} c_s = 343.2\,m/s \\ f_s = 44100\,Hz \end{cases} \qquad (3.6)$$

The conditions of the used speed of sound are sea-level atmospheric pressure and 20ºC. The technical aspects of signal processing and detection and the code implementation of them are outside the scope of this report

4. Results

After making two training apps, doing a lot of research, performing lots of simulations, modelling signals, and specially committing ourselves and dedicating a lot of time, these are the results of the thesis.

At the start of this project two apps were created to get familiarised with the Android environment and mobile development in general. One of them is a "phone tester", which reads the values of the most important sensors and prints their values. That was an exercise to get to know basic APIs and work with values behind the scenes. The second app is a speed cube stopwatch to measure times of Rubik's cube solves. This was a more user-oriented app with more emphasis on the interface and ease of use. That doesn't mean the innards are any less complex, as it manages memory, stores values and computes some statistics.

During the development of the project, a Bluetooth communications protocol was created to explore this option. The result is quite similar to the Wi-Fi Direct counterpart, but it isn't as mature because the idea was dropped in few days. It doesn't implement commands or distance estimation capabilities, but it has a chat interface to communicate between users that could be ported to the Wi-Fi Direct client.

The final RADIUS app is able to perform a number of tasks:

- Track, print and record the trajectory of the user.
- Establish a communication between users using Wi-Fi Direct.
- Compute the distance to the remote user and estimate its direction based on the acquired RSSI values from the communication link (with an alternative root method).
- Ability to send and receive text messages.
- Ability to measure distances using time of flight of audio waves.

The objectives set in the statement of purpose seem to have been accomplished, and some others have been added as well. For illustrative purposes, screenshots of the RADIUS app can be found in the appendix of this report.

Performance of RSSI-based distance estimations will be analysed next. Audio ToF-based distance estimations and user tracking performance, benchmarking and analysis are not part of the scope of this report, inasmuch as Nèstor is the responsible of these parts.

4.1. Performance benchmarking of distance estimations

To start with, the maximum range of operation is measured. The range was determined to be a little bit over 190 m within LoS, and 60 m with obstacles in the way. These are mean values obtained by repeating the experiment ten times in a fairly open outdoor scenario with two LG Nexus 5 devices. Distances were physically measured and maximum range is defined as the distance at which communication is lost due to poor signal reception.

Next, the verisimilitude of the log-distance model is assessed to check if it adjusts to real signal propagation behaviour. Mean RSSI measured values are compared to the log-distance model described in equation *(4.1)* with its calibration constant and parameters, which is plotted with a range of one standard deviation. RSSI values were obtained in the conditions described above and within LoS.

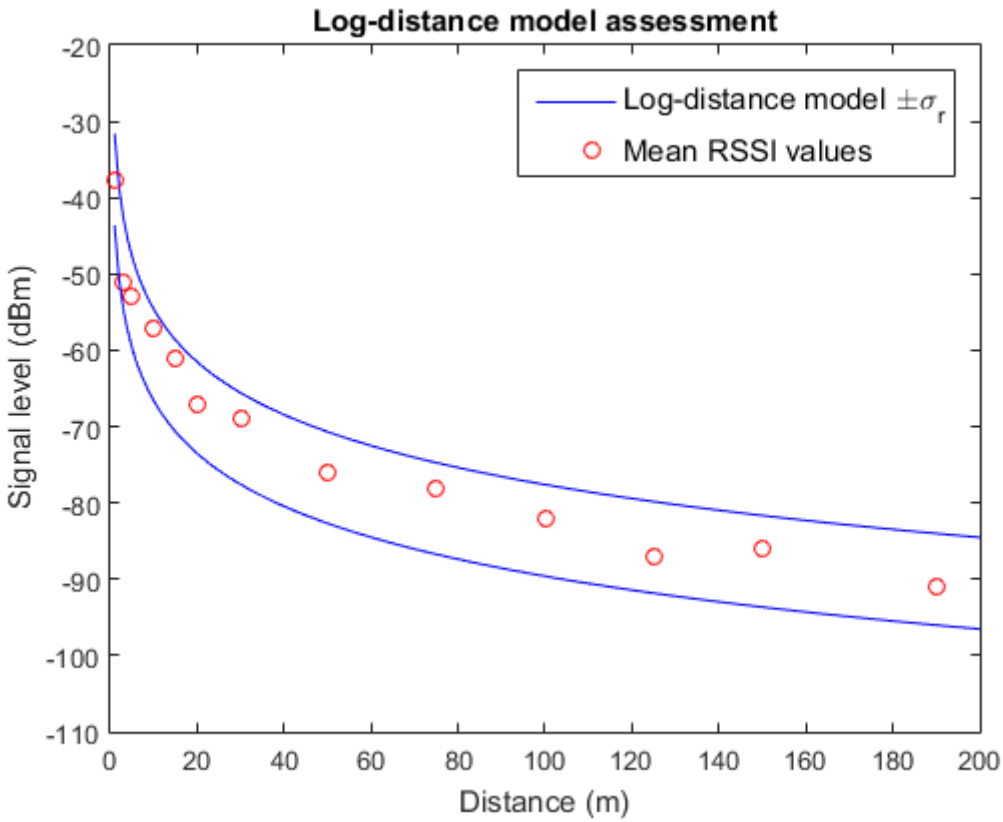

Figure 12: Log-distance model assessment

RSSI measurements fall nicely in the expected range. Values in the lower distances tend to move around a little bit more, but in general lines it seems the model is properly adjusted and that it can be trusted. These results look quite promising, but it should be remembered that when the app is running, it can't deliver results as accurate because shadowing and noise are not being cancelled out by averaging over several readings like in this case, so computations are a bit less accurate.

The same data is visualised then as the error in real versus estimated distance to gain more insight into the results:

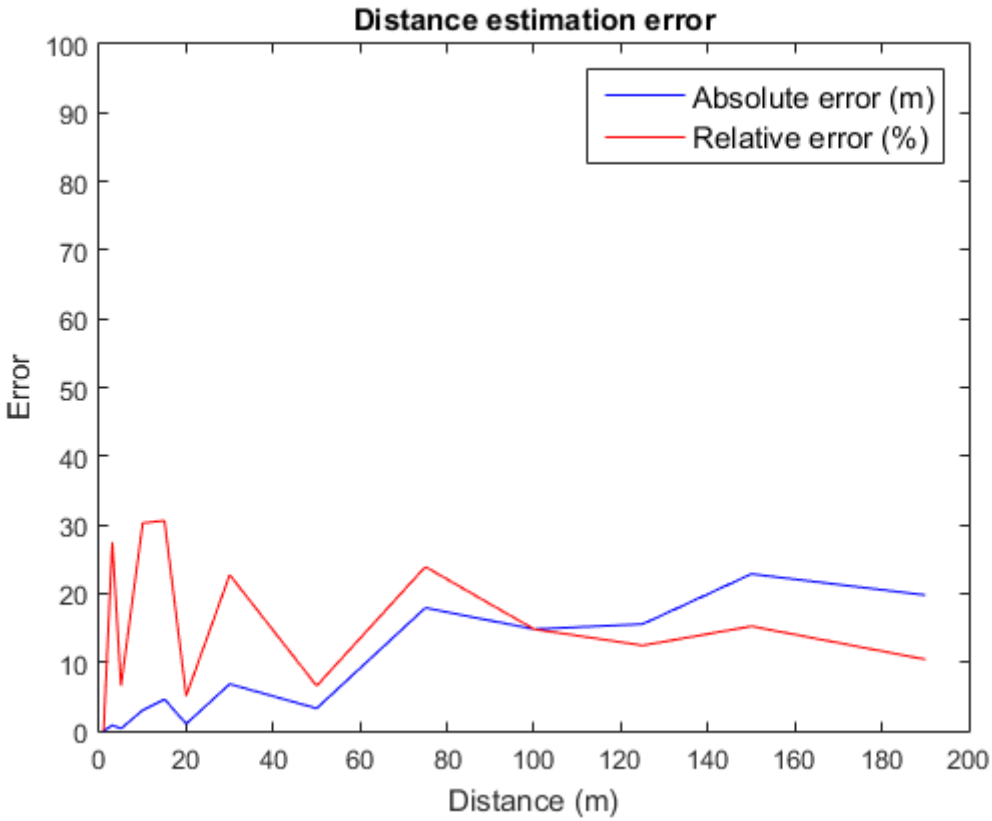

Figure 13: Distance estimation error

This plot shows that the absolute error grows with the distance, which is to be expected, but when taking a look at the relative error, the estimations seem to be better in the longer distances. This is due to the RSSI values in the shorter range being a little off the model, as seen in Figure 12.

The maximum relative error is about 30% at 10 and 15 metres, which corresponds to an error of 3 and 4.5 m respectively, the model also presents other local maxima of around 20% in the medium range. The minimum relative error is around 5% and it's produced in the short to medium and long distances, when the model best fits the real propagation behaviour.

5. Budget

This thesis does not have a physical prototype as a final product, so an estimation of the number of hours dedicated to it are evaluated at the cost of a junior engineer salary, as well as the licenses of the used software and the devices for developing and testing the applications.

Concept	Description	Units	Price	Total
Matlab R2015a	Academic License	1	0 €	0 €
Android Studio 2.1.1	Apache License version 2.0	1	0 €	0 €
Development devices	LG Nexus 5	2	300,00 €	600,00 €
Salary	Estimated work hours	400	8,00 €	3.200,00 €
			GRAND TOTAL	**3.800,00 €**

6. Conclusions

This was a very ambitious project to which a lot of hours have been dedicated, more than it could seem at first sight. The Android application alone has taken over two months of dedicated work to be thought-out and written. Even then, a number of features could not be implemented due to lack of time, they are listed below as possible upgrades.

The results of the project could have been a little bit better, but considering the point we have reached taking into account that we started from scratch, we are both very pleased. This work can be continued and be further developed to deliver better performance and incorporate new features.

6.1. Possible upgrades and future development

More and more smartphones are including pressure and temperature sensors in their specification list lately. Taking advantage of this, an algorithm can be implemented to adjust the speed of sound to the exact conditions of the environment in the moment of execution. This would make the reading so precise that the final result would be bottlenecked by the sampling frequency of the microphones.

Also, the ability to give this project a third dimension of depth can be implemented by using a barometer. A typical pressure sensor can resolve an altitude difference of less than a metre[13], making the application able to tell the difference between floors.

The same idea of reading the environment conditions to adjust the models the program uses can be applied to the propagation model of RF waves to give a better RSSI to distance equivalence. The light sensor and the microphone can be useful to determine if the user is located outdoors or indoors and how big the space is in that case.

Furthermore, a chat interface (like the Bluetooth-based one commented in the results) and an image or general file transfer service can be added to the application to take the user experience to the next level. Another very simple thing to do is to add more functionality through commands. The framework is already set, so only an idea is necessary to be implemented right away.

Moreover, a multi-hop implementation could be considered. The idea is that a message can be sent to a destination which is out of the reach of the original sender. This could be achieved by broadcasting a copy of the message to all available peers in range. Eventually, the message will reach its destination as long as the chain doesn't break. In which case, the message would have been transmitted further away than it would have in the regular way in any case. But once again there's a problem with the Android API: when a Wi-Fi Direct connection is established, a group owner is decided. That

[13] BOSCH Sensortec. "BMP280 Digital Pressure Sensor". Datasheet. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf. [Accessed 21 June 2016].

particular device cannot act as a client of another group at the same time[14]. This means that the current implementation of Wi-Fi Direct in Android forces every message to pass through the group owner. And not only that, a device cannot be connected to more than one group as a client, so this would derail the whole strategy. To sum up: a multi-hop implementation can be done using Wi-Fi Direct, but cannot be easily done with Android devices as its API doesn't allow to do so.

A paper that tackles exactly this problem was recently published and it seems they are able to achieve multi-hop networking using Wi-Fi Direct in Android, so there is still hope for this idea.

[14] Android Developer Reference. "WifiP2pGroup". android.net.wifi.p2p. [Online]. Available: https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pGroup.html. [Accessed 21 June 2016].

7. Appendices

7.1. RADIUS application screenshots

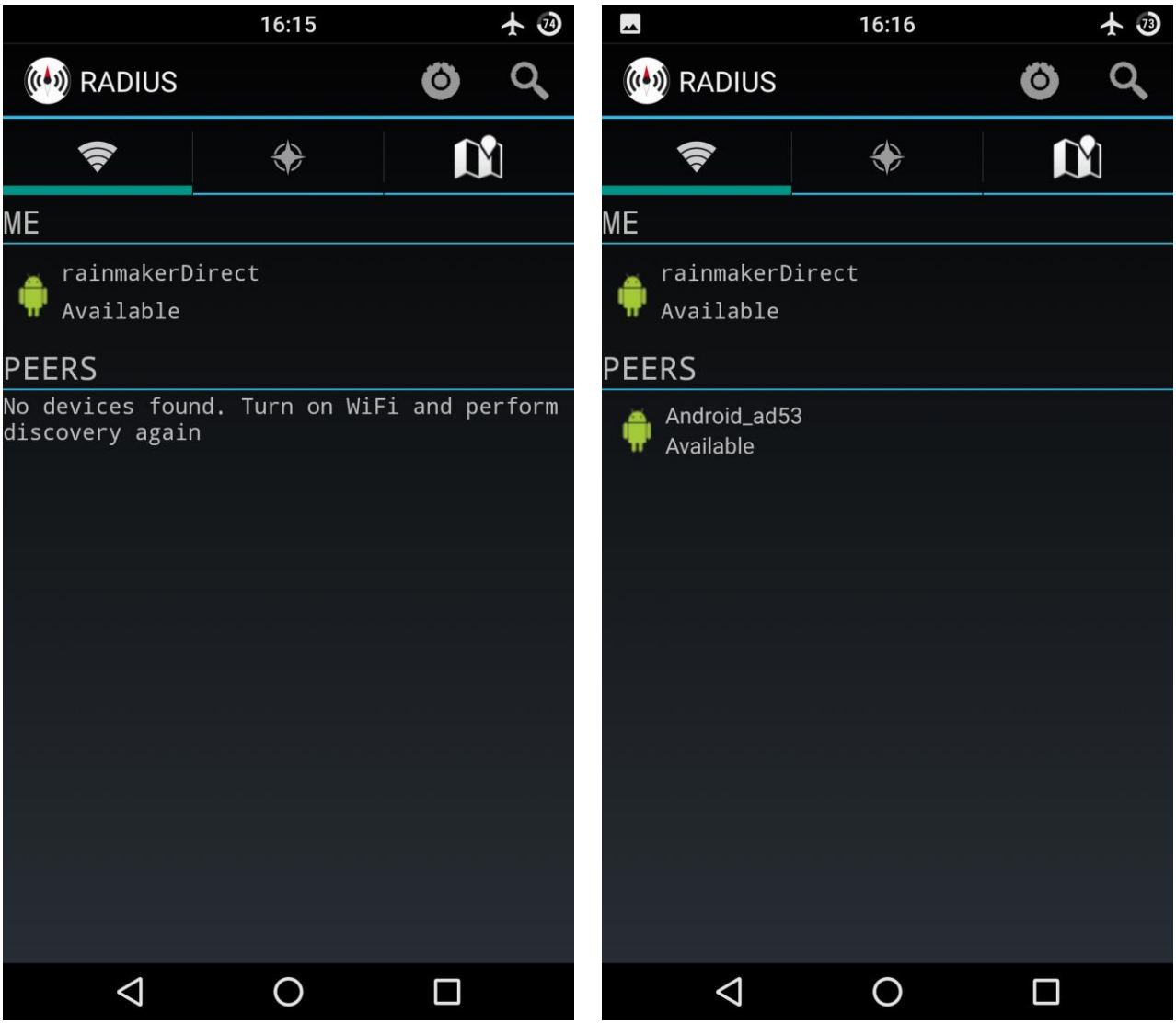

Figure 14: RADIUS app screenshots, (a) shows the welcome screen, (b) shows the results of a peer discovery.

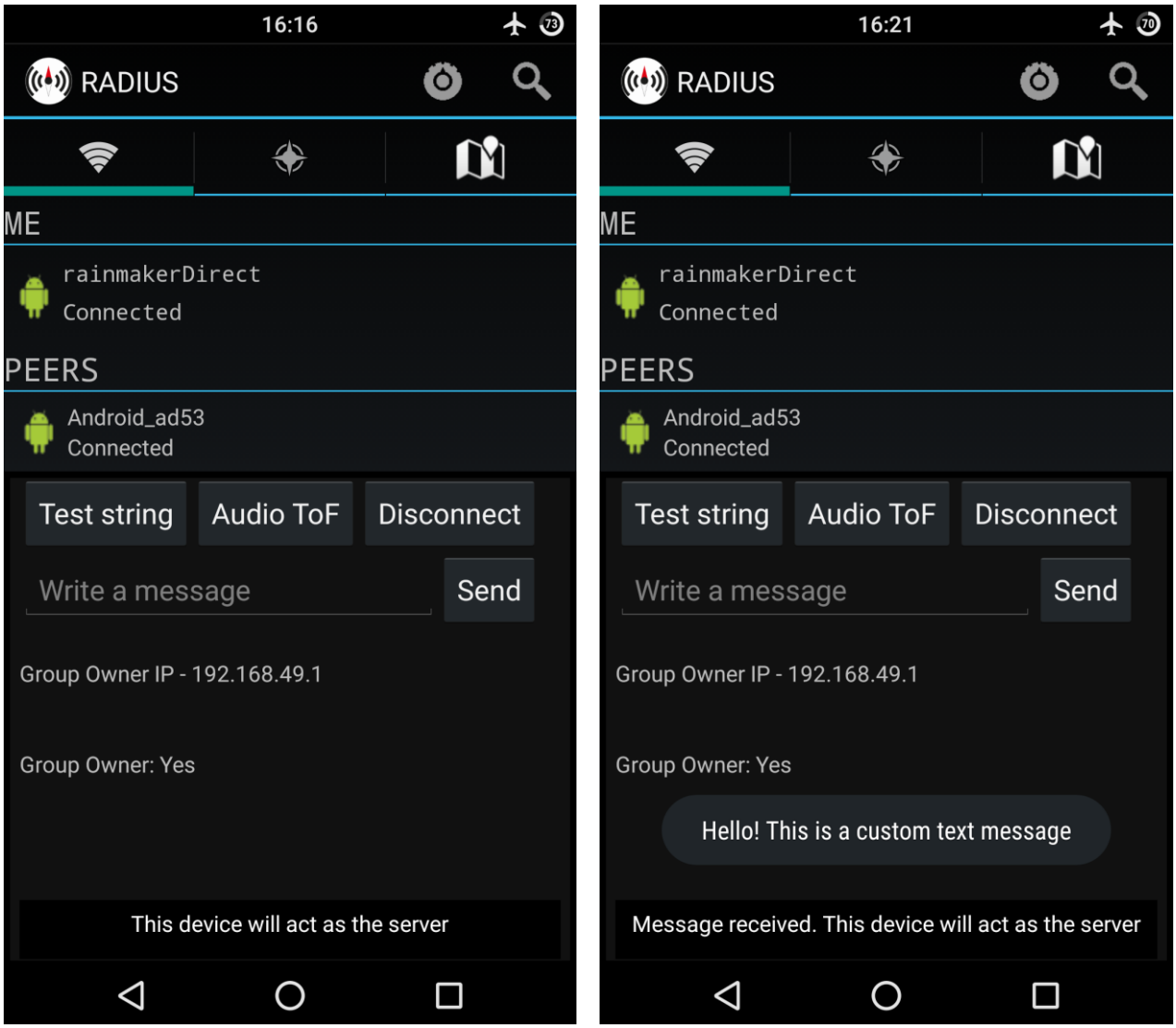

Figure 15: RADIUS app screenshots, (a) shows the state of an established connection, (b) shows the toast notification of a received text message.

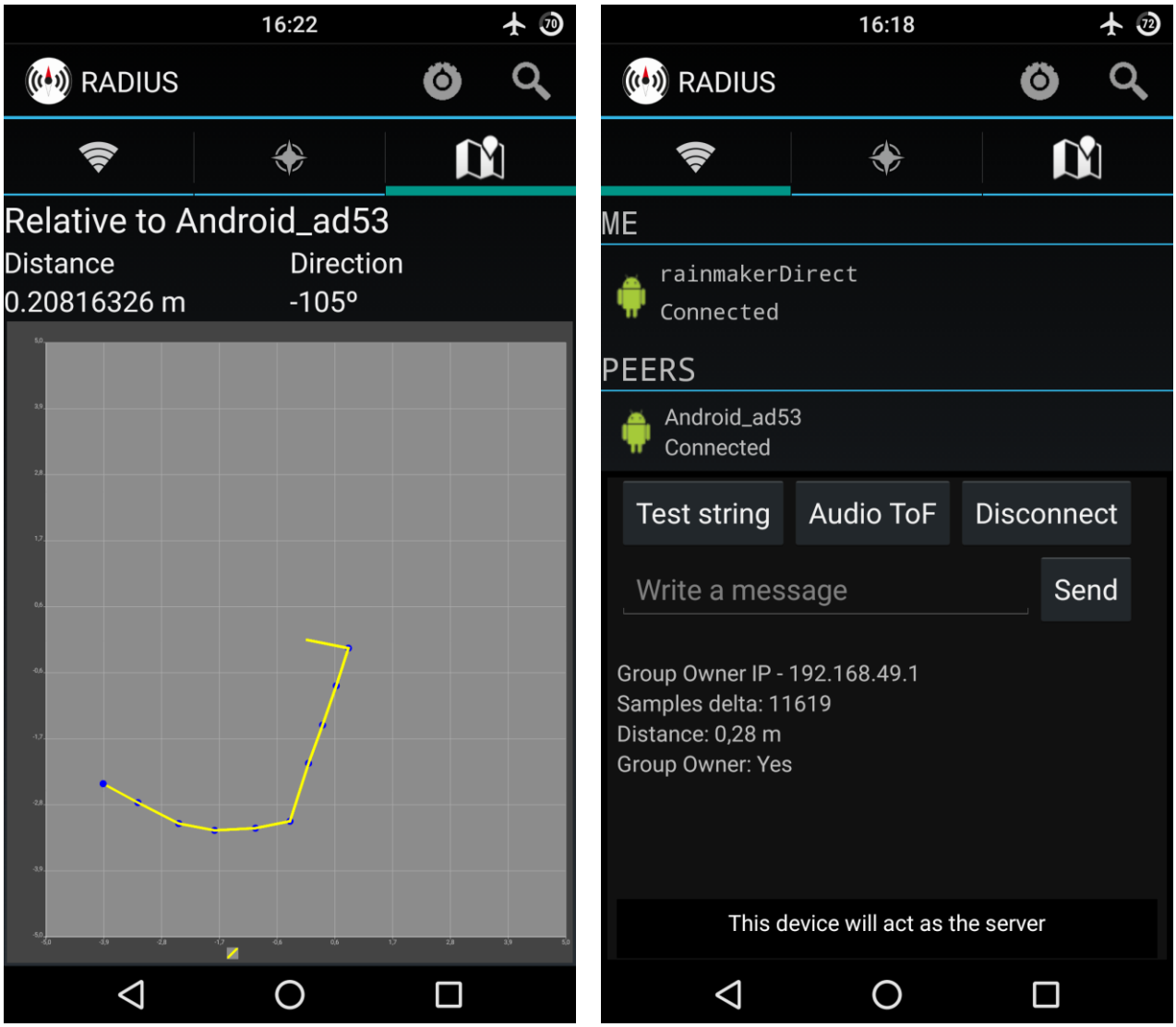

Figure 16: RADUIS app screenshots, (a) shows the trajectory taken by the user to the distant peer with its current relative distance and direction, (b) shows the result of an audio ToF-based distance computation.

7.2. Communications protocol code

7.2.1. Discover button callback

```
public boolean discover() {
    if (!isWifiP2pEnabled) {
        Toast.makeText(WiFiDirectActivity.this,
R.string.p2p_off_warning,
                Toast.LENGTH_SHORT).show();
        return true;
    }
    final DeviceListFragment fragment = (DeviceListFragment)
getFragmentManager()
            .findFragmentById(R.id.frag_list);
    fragment.onInitiateDiscovery();
    manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
```

```
        @Override
        public void onSuccess() {
            Toast.makeText(WiFiDirectActivity.this, "Discovery
Initiated",
                    Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onFailure(int reasonCode) {
            Toast.makeText(WiFiDirectActivity.this, "Discovery Failed: "
+ reasonCode,
                    Toast.LENGTH_SHORT).show();
        }
    });
    return true;
}
```

The discoverPeers method of WifiP2pManager is part of the API.

7.2.2. Connect button callback

```
mContentView.findViewById(R.id.btn_connect).setOnClickListener(new
View.OnClickListener() {

    @Override
    public void onClick(View v) {
        WifiP2pConfig config = new WifiP2pConfig();
        config.deviceAddress = device.deviceAddress;
        config.wps.setup = WpsInfo.PBC;
        if (progressDialog != null && progressDialog.isShowing()) {
            progressDialog.dismiss();
        }
        progressDialog = ProgressDialog.show(getActivity(), "Connecting
to: " + device.deviceName,
                "Address: " + device.deviceAddress + "\nPress back to
cancel", true, true
                , new DialogInterface.OnCancelListener() {

                    @Override
                    public void onCancel(DialogInterface dialog) {
                        ((DeviceActionListener)
getActivity()).cancelDisconnect();
                    }
                }
        );
        ((DeviceActionListener) getActivity()).connect(config);

    }
});



public void connect(WifiP2pConfig config) {
    manager.connect(channel, config, new ActionListener() {

        @Override
        public void onSuccess() {
```

```
        }

        @Override
        public void onFailure(int reason) {
            Toast.makeText(WiFiDirectActivity.this, "Connect failed.
Retry.",
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

The connect method of WifiP2pManager is part of the API.

7.2.3. Connection established callback

```
public void onConnectionInfoAvailable(final WifiP2pInfo info) {
        if (progressDialog != null && progressDialog.isShowing()) {
            progressDialog.dismiss();
        }
        this.info = info;
        this.getView().setVisibility(View.VISIBLE);

        TextView view = (TextView)
mContentView.findViewById(R.id.group_owner);
        view.setText(getResources().getString(R.string.group_owner_text)
+ ((info.isGroupOwner == true) ? getResources().getString(R.string.yes)
: getResources().getString(R.string.no)));

        view = (TextView) mContentView.findViewById(R.id.device_info);
        view.setText("Group Owner IP - " +
info.groupOwnerAddress.getHostAddress());
        serverIP = info.groupOwnerAddress.getHostAddress();

        if (info.groupFormed && info.isGroupOwner) {
            isClient = false;
            ((TextView)
mContentView.findViewById(R.id.status_text)).setText(getResources().getS
tring(R.string.server_text));

        } else if (info.groupFormed) {
            isClient = true;
            ((TextView)
mContentView.findViewById(R.id.status_text)).setText(getResources().getS
tring(R.string.client_text));
        }

        if (isClient) { sendMessage(COMMAND_DUMMY); }

        new MessageAsyncTask(getActivity(),
mContentView.findViewById(R.id.status_text)).executeOnExecutor(AsyncTask
.THREAD_POOL_EXECUTOR);mContentView.findViewById(R.id.btn_send_string).s
etVisibility(View.VISIBLE);

mContentView.findViewById(R.id.btn_send_custom_message).setVisibility(Vi
ew.VISIBLE);
        customMessageText.setVisibility(View.VISIBLE);

mContentView.findViewById(R.id.btn_send_command).setVisibility(View.VISI
BLE);
```

```
mContentView.findViewById(R.id.btn_connect).setVisibility(View.GONE);
    }
```

7.2.4. MessageAsyncTask

```
public static class MessageAsyncTask extends AsyncTask<Void, Void,
String> {

        private Context context;
        private TextView statusText;

        public MessageAsyncTask(Context context, View statusText) {
            this.context = context;
            this.statusText = (TextView) statusText;
        }

        @Override
        protected String doInBackground(Void... params) {
            try {
                ServerSocket serverSocket = new
ServerSocket(SERVER_PORT);
                Log.d(WiFiDirectActivity.TAG, "Server socket opened");
                Socket client = serverSocket.accept();

                if (DeviceDetailFragment.isClient()) {
                    Log.d(WiFiDirectActivity.TAG, "Server InetAddress -
" + DeviceDetailFragment.serverIP);
                } else {
                    DeviceDetailFragment.clientIP =
client.getInetAddress().getHostAddress();
                    Log.d(WiFiDirectActivity.TAG, "Client InetAddress -
" + client.getInetAddress());
                }
                Log.d(WiFiDirectActivity.TAG, "Connection done");
                DataInputStream inputstream = new
DataInputStream(client.getInputStream());
                Log.d(WiFiDirectActivity.TAG, "Receiving message...");
                String receivedMessage = inputstream.readUTF();
                Log.d(WiFiDirectActivity.TAG, "Received message - " +
receivedMessage);
                serverSocket.close();
                return receivedMessage;
            } catch (IOException e) {
                Log.e(WiFiDirectActivity.TAG, e.getMessage());
                return null;
            }
        }

        @Override
        protected void onPostExecute(String result) {
            if (result != null) {
                new resultHandler(context,
statusText).handleResult(result);
            }
            new MessageAsyncTask(context,
statusText).executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
        }
```

```
        @Override
        protected void onPreExecute() {
        }
    }
```

7.2.5. Send message callback

```
String messageToSend = customMessageText.getText().toString();
sendMessage(messageToSend);
customMessageText.setText(R.string.empty);
```

```
    public void sendMessage(String messageToSend) {
        Log.d(WiFiDirectActivity.TAG, "Sending message");
        Intent serviceIntent = new Intent(getActivity(),
MessageTransferService.class);

serviceIntent.setAction(MessageTransferService.ACTION_SEND_MESSAGE);
        String addr;
        if (isClient) {
            addr = info.groupOwnerAddress.getHostAddress();
        } else {
            addr = clientIP;
        }

serviceIntent.putExtra(MessageTransferService.EXTRAS_GROUP_OWNER_ADDRESS
, addr);

serviceIntent.putExtra(MessageTransferService.EXTRAS_GROUP_OWNER_PORT,
SERVER_PORT);

serviceIntent.putExtra(MessageTransferService.EXTRAS_MESSAGE_TO_SEND,
messageToSend);
        getActivity().startService(serviceIntent);
```

```
public class MessageTransferService extends IntentService {

    public static final String ACTION_SEND_MESSAGE =
"com.example.android.wifidirect.SEND_MESSAGE";
    public static final String EXTRAS_GROUP_OWNER_ADDRESS = "go_host";
    public static final String EXTRAS_GROUP_OWNER_PORT = "go_port";
    public static final String EXTRAS_MESSAGE_TO_SEND = "Mensaje por
defecto, algo va mal";
    private static final int SOCKET_TIMEOUT = 5000;

    public MessageTransferService(String name) {
        super(name);
    }

    public MessageTransferService() {
        super("MessageTransferService");
    }
```

```
    @Override
    protected void onHandleIntent(Intent intent) {

        if (intent.getAction().equals(ACTION_SEND_MESSAGE)) {
            String host =
intent.getExtras().getString(EXTRAS_GROUP_OWNER_ADDRESS);
            Socket socket = new Socket();
            int port =
intent.getExtras().getInt(EXTRAS_GROUP_OWNER_PORT);
            DataOutputStream stream;

            try {
                Log.d(WiFiDirectActivity.TAG, "Opening client socket");
                socket.bind(null);
                socket.connect((new InetSocketAddress(host, port)),
SOCKET_TIMEOUT);
                stream = new DataOutputStream(socket.getOutputStream());
                Log.d(WiFiDirectActivity.TAG, "Client socket is
connected: " + socket.isConnected());

                String messageToSend =
intent.getExtras().getString(EXTRAS_MESSAGE_TO_SEND);

                stream.writeUTF(messageToSend);
                Log.d(WiFiDirectActivity.TAG, "Message sent");
            } catch (IOException e) {
                Log.e(WiFiDirectActivity.TAG, e.getMessage());
            } finally {
                if (socket != null) {
                    if (socket.isConnected()) {
                        try {
                            socket.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }

        }
    }
}
```

7.3. Python delay time control simulation code

```
import time
import random

objectiveTime = 0.1  # 100 ms
results = []
N = 5000

for i in range(N):
    # Program starts
    time0 = time.clock()

    # Dummy load
    L = []
```

```
    for j in range(random.randint(50000, 100000)):
        L.append(random.random())

    # Program ends
    time1 = time.clock()

    # print "Execution time:"
    # print time1-time0
    time.sleep(objectiveTime-(time1-time0))
    time2 = time.clock()
    # print "Total time:"
    # print time2-time0

    # Returns the error in microseconds
    results.append((time2-time0-objectiveTime)/1e-6)

text_file = open("OutputPyhton.txt", "w")
for item in results:
    text_file.write("%s\n" % item)
text_file.close()
```

7.4. Java delay time control simulation code

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class ToFTestJava {

    static int numberOfLoops = 5000;
    static int loopObjectiveTime = 100000000; //100 ms
    static long[] results = new long[numberOfLoops];

    public static void main(String[] args) {

        for (int i = 0; i < numberOfLoops; i++) {
            results[i] = measureOneLoop();
        }

        try {
            write("OutputJava.txt", results);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

    }

    public static long measureOneLoop() {
        //Program starts
        long time0 = System.nanoTime();

        //Dummy load (between 10 and 20ms)
        int[] foo = new int[1000000];
        Random random = new Random();
        for (int i = 0; i < (random.nextInt(900000) + 100000); i++) {
            foo[i] = random.nextInt();
        }
```

```
        //Program ends
        long time1 = System.nanoTime();

        //System.out.print("Execution time (ms):");
        long execTime = time1 - time0;
        //System.out.println(execTime/1000000);

        try {
            TimeUnit.NANOSECONDS.sleep(loopObjectiveTime - execTime);
        } catch (InterruptedException ex) {
            System.out.println(ex.getMessage());
        }

        long time2 = System.nanoTime();
        //System.out.print("Total time (ms):");
        //System.out.println((time2-time0)/1000000);

        //System.out.print("Error (us):");
        //System.out.println(((time2-time0) - loopObjectiveTime)/1000);

        //Returns error in microseconds
        return ((time2 - time0) - loopObjectiveTime) / 1000;
    }

    public static void write(String filename, long[] x) throws Exception
{
        BufferedWriter outputWriter = null;
        outputWriter = new BufferedWriter(new FileWriter(filename));
        for (int i = 0; i < x.length; i++) {
            outputWriter.write(Long.toString(x[i]));
            outputWriter.newLine();
        }
        outputWriter.flush();
        outputWriter.close();
    }
}
```

Bibliography

References are cited in the footnote of each corresponding page, the consulted bibliography is listed below:

C. Funai, C. Tapparello, W. Heinzelman. "Supporting Multi-hop Device-to-Device Networks through Wi-Fi Direct Multi-group Networking". *Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY, USA.*

N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, A. T. Campbell. "A Survey of Mobile Phone Sensing". *Department of Computer Science, Dartmouth College, Hanover, NH, USA.*

J. Liu, R. Chen, L. Pei, R. Guinness, H. Kuusniemi. "A Hybrid Smartphone Indoor Positioning Solution for Mobile LBS". *Department of Navigation and Positioning, Finnish Geodetic Institute, Geodeetinrinne 2, Masala 02431, Finland.*

D. Ayllón, H. Sánchez-Hevia, R. Gil-Pita, M. Rosa-Zurera. "Indoor Blind Localization of Smartphones By Means Of Sensor Data Fusion". *Department of Signal Theory and Communications, University of Alcala, Spain.*

P. Lazik, N. Rajagopal, B. Sinopoli, A.Rowe. "Ultrasonic Time Synchronization and Ranging on Smartphones". *Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA.*

N. Banerjee, S. Agarwal, P. Bahl, R. Chandra, A. Wolman, M. Corner. "Virtual Compass: Relative Positioning To Sense Mobile Social Interactions". *University of Arkansas Fayetteville, Microsoft Research Redmond, and University of Massachusetts Amherst, USA.*

Y. Gu, A. Lo, I. Niemegeers. "A survey of indoor positioning systems for wireless personal networks". Communications Surveys Tutorials, IEEE, Vol 11, 2009.

Aleksey Shipilëv. "Nanotrusting the Nanotime". *Blog post*, 2014. [Online] Available: http://shipilev.net/blog/2014/nanotrusting-nanotime/. [Accessed: 21 June 2016].

Vincent Gao. "Proximity and RSSI". *What's new with blue?* 2016. [Online] Available: http://blog.bluetooth.com/proximity-and-rssi/. [Accessed: 21 June 2016].

Doxygen. "wpa_supplicant control interface". *wpa_supplicant / hostapd*, 2015. [Online] Available: https://w1.fi/wpa_supplicant/devel/ctrl_iface_page.html [Accessed: 21 June 2016].

Google Inc. "Package Index". *Android Developer Reference*. [Online]. Available: https://developer.android.com/reference/packages.html. [Accessed 21 June 2016].

K. A. Li, T. Y. Sohn, S. Huang, W. G. Griswold. "PeopleTones: A System for the Detection and Notification of Buddy Proximity on Mobile Phones". Department

of Computer Science and Engineering, University of California, San Diego, CA, USA.

Patrick Lazik and Anthony Rowe. Indoor pseudo-ranging of mobile devices using ultrasonic chirps. *In Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys 2012, New York, NY, USA.

Stephen P. Tarzia, Peter A. Dinda, Robert P. Dick, and Gokhan Memik. Indoor localization without infrastructure using the acoustic background spectrum. *In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys 2011, New York, NY, USA.*

H. Lu, W. Pan, N. D. Lane, T. Choudhury, A. T. Campbell. "SoundSense: Scalable Sound Sensing for People-Centric Applications on Mobile Phones". *Department of Computer Science, Dartmouth College, Hanover, NH, USA.*

Glossary and specific terms

RSSI: Received Signal Strength Indication.

ToF: Time of Flight.

LoS: Line of Sight.

RF: Radio Frequency.

API: Application Programming Interface. It's a set of methods, tools and protocols used to build an application or software.

Activity: It's a class that focuses on a task that the user would like to do, interacting with it and creating the user interface.

Fragment: It's a piece of an application's user interface that makes part of the activity.

Service: It's a component that serves the purpose of executing tasks that are longer than normal and do not require interaction with the user.

BroadcastReceiver: It's a base class that receives broadcast messages.

AsyncTask: It's a class that enables to perform background tasks without the need of threads or handlers.

ADB Shell: Android Debug Bridge. It provides a UNIX command-line interface shell able to run commands on a connected device.

Root: It's the name of the user account that has access to all commands and files on a Linux operating system by default. In the Android context, it refers to the process of gaining said privileges.

Monte Carlo simulation: It's a type of computational algorithm that is commonly used in the analysis of random nature problems. It relies on the repetition and sampling of a process to extract a trend and help predict its behaviour and analyse the results.