Faculdade de Engenharia da Universidade do Porto

FEUP

Co-location of Mobile Devices with Audio and Wireless Interfaces

Filipe Moreira Guedes

Dissertação submetida no Âmbito do

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Major de Telecomunicações

Orientador: Ricardo Santos Morla (Professor Doutor)

Porto, Julho de 2008

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2007/2008

A Dissertação intitulada

"Co-location of Mobile Devices with Audio and Wireless Interfaces"

foi aprovada em provas realizadas 10/Julho/2008

o júri

Presidente Professor Doutor Pedro Henrique Henriques Guedes de Oliveira
Professor Catedrático da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Carlos Manuel Robalo Lisboa Bento
Professor Auxiliar da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Professor Doutor Ricardo Santos Morla
Professor Auxiliar Convidado da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Filipe Hermínio Pacheco Moreira Guedes

Faculdade de Engenharia da Universidade do Porto

Abstract

A location mechanism capable of determining the co-location of mobile devices provides valuable information to context-aware applications. Different techniques and technologies may be used to determine location information. They often require specific infrastructures or specialized hardware. As mobile computing devices are commonly equipped with audio and 802.11 interfaces, a co-location system using these interfaces represents a low-cost and easy to deploy solution.

This dissertation discusses adopted solutions for the development and evaluation of a co-location system for mobile devices with audio and 802.11 interfaces. In particular, a co-location system based on audio and 802.11 information has been developed, that uses data acquired by mobile devices to determine their co-location, providing a correct *near* or *far* classification in scenarios of interest.

Resumo

Um mecanismo de localização capaz de determinar a co-localização de dispositivos móveis fornece informação de valor a aplicações sensíveis ao contexto. Diferentes técnicas e tecnologias podem ser utilizadas para determinar a localização. Frequentemente, estas requerem infrastruturas específicas ou *hardware* especializado. Os dispositivos de computação móveis estão, usualmente, equipados com interfaces áudio e *802.11*, portanto, um sistema de co-localização que use estas interfaces representa uma solução de baixo custo e de fácil instalação.

Esta dissertação discute as soluções adoptadas para o desenvolvimento e avaliação de um sistema de co-localização para dispositivos móveis com interfaces áudio e *802.11*. Em particular, foi desenvolvido um sistema de co-localização baseado em informação áudio e *802.11*, que usa dados recolhidos por dispositivos móveis para determinar a sua co-localização, fornecendo uma classificação correcta de *near* (perto) ou *far* (longe) em cenários de interesse.

Acknowledgements

I am very grateful to Prof. Ricardo Morla, whose useful and clear-minded appreciations regarding this work helped me to achieve the objectives we proposed.

A great word of thanks goes to my friend and colleague Pedro Correia, who helped me to perform the tests used to evaluate this work. His patience revealed to be limitless, in the same manner his friendship had already revealed to be priceless.

I would also like to thank all my friends that helped me maintain my motivation, with their supporting words and gestures.

Finally, for keeping me strongly determined with their love and tender support — and, at times, their extreme patience! —, especially during the most difficult moments, I leave here my words of great gratitude to my parents, my brother Manuel Pedro, and Joana. Thank you all!

The Author

Contents

List of Figures

List of Tables

List of Code Listings

Acronyms

AP	Access Point
FIFO	First In First Out
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
LOS	Line-of-Sight
PDA	Personal Digital Assistant
RF	Radio Frequency
TDOA	Time difference of arrival

Chapter 1

Introduction

Today's computing paradigms are ever changing, becoming more efficient and more able to serve users in the accomplishment of their everyday tasks. One of such emerging paradigms is ubiquitous computing, a model in which computing is available throughout the physical environment, integrated in common objects, from mobile phones to microwave ovens, and present in everyday activities [1]. Not confined to the interaction of the user with a single device, ubiquitous computing, also referred to as pervasive computing, stands opposed to desktop computing as a mobile, interactive, widely available while ideally seamless and context-aware computing paradigm.

Context-aware computing is, in fact, a particular case of ubiquitous computing in which applications running on mobile devices take advantage of the information gathered from the surrounding environment to provide services and information to the user [2, 3, 4].

Context and contextual information include location, nearby objects and people, conditions — such as lighting, noise or network connectivity — and changes in these aspects. Generally, there are three important aspects of context: where you are, who you are with and what resources are nearby [4]. Using such information, context-aware applications can provide valuable services, particularly in a mobile computing environment.

Schilit *et al.* [4] identify four categories of context-aware applications:

1. *Proximate Selection*: user interface technique where nearby objects of interest are emphasized or otherwise made easier to choose;

2. *Automatic Contextual Reconfiguration*: adding new components, removing existing components, altering connections between components, in response to context changes;

3. *Contextual Information and Commands*: offer commands or services based on available context;

4. *Context-Triggered Actions*: automatically execute commands based on available context, i.e., specify how a context-aware system adapts to context changes.

It becomes obvious that these context aspects and context-aware applications require the determination of location and, particularly with mobile computing, the co-location of devices, i.e., where devices are located relatively to each other. One can say that location information is significantly important for a context-aware application. Recent research focused on the study and development of location and positioning computing systems capable of determining the location of mobile devices, reflecting the relevance of such contextual information.

1.1 Characterization of the problem

An automatic location system may use different techniques for location-sensing [5] (e.g. triangulation or proximity) and be implemented using different technologies (e.g. GPS or ultrasounds). Complexity may vary from one technique to another and specific technologies represent specific hardware requirements.

To help with the characterization and evaluation of location systems, Hightower *et al.* [6] elaborated a taxonomy, generally independent of used techniques and technologies, that focuses on several properties of such systems:

- physical position and symbolic location;
- absolute versus relative;
- localized location computation;
- accuracy and precision;
- scale;
- recognition;
- cost;
- limitations.

These properties can be used to classify a location system and, during development, to help choose implementation solutions. For example, choosing to use commonly available hardware can help reduce costs, but limitations may arise from this, as such hardware may not be suitable to accurate location determination or to highly scalable systems. Another example of how these properties may help choose implementation solutions is the choice between localized location computation and centralized location computation. Localized location computation requires a certain level of computing capabilities from the

devices but, opposed to centralized location computation, is less dependent of specific infrastructures and their capabilities.

Regarding the use of commonly available hardware, as mobile computing devices — such as PDAs and laptop computers, which are widely used equipments — are commonly equipped with audio and IEEE 802.11 interfaces, a location system for mobile devices using these interfaces would not represent an additional cost with hardware — for the user or the manufacturer — and could become a widely spread solution for an automatic location system, consequently promoting the development of context-aware applications.

The addressed problem in this dissertation is the determination of the co-location of audio and 802.11 capable mobile devices, i.e., the determination of the relative location of mobile devices using their audio and 802.11 capabilities.

1.2 Objectives

The objective of this work is to understand how a *co-location system* for mobile devices can be developed that allows a device to determine if it is far or close to other devices of interest using audio and 802.11 information, requiring little or no calibration, and using embedded audio and 802.11 interfaces commonly available in mobile devices to avoid the need for any kind of specific external hardware.

Other sub-objectives of this work include: the development of algorithms to classify the co-location of devices and to process audio and 802.11 information; the development of the underlying distributed system which allows devices to exchange information needed by such algorithms; and the evaluation of the *co-location system* in different scenarios.

1.3 Methodology

Work was divided in three major phases. Phase one focused on the problem of audio-based co-location determination and the development of an algorithm capable of doing so. During this phase, the audio-based co-location algorithm was developed using MATLAB. Audio data was acquired in simple scenarios and used offline to adjust, during development, and evaluate, in a final stage, the algorithm.

Phase two focused on the development of a distributed system that implements the audio-based co-location algorithm in C# and all the components which integrate the *co-location system*. Mechanisms for exchanging data were developed and online tests were performed in predefined scenarios to evaluate the system.

The third phase integrates 802.11 data to understand how results from the previous phases can be improved. During this phase, an 802.11-based algorithm and a final co-location classification algorithm (using the output of the audio-based and the 802.11-based algorithms) were developed. Further online tests, in predefined scenarios, were conducted to evaluate the system.

1.4 Structure

This dissertation discusses adopted solutions for the development and evaluation of a *co-location system* for mobile devices. The proposed system uses audio and IEEE 802.11 information to classify the co-location of mobile devices.

This dissertation is structured into the following chapters: chapter 1 is this introduction; chapter 2 presents related work in the area of context-aware applications and positioning systems; chapter 3 presents our approach to the problem and discusses the proposed solution for the *co-location system*; in chapter 4, test results of the proposed solution are presented and evaluated; chapter 5 concludes this dissertation, discussing the contribution of this work and proposing possible options of future work on this problem.

Chapter 2

Related Work

Since the emergence of the ubiquitous computing paradigm, research has been done regarding context-aware applications and the determination of the location of mobile devices.

This chapter discusses some of this research work, presenting context-aware applications and location systems of interest.

2.1 Context-Aware Systems and Applications

As the work discussed in this dissertation will represent a direct impact in systems using context-aware applications, it is important to identify these applications and understand features they provide.

Teleporting [7] is a good example of a context-aware application which uses location information. Obtaining the user location from the Active Badge [8] system, described in section 2.2, *Teleporting* allows existing and familiar applications to follow the user while he moves around by mapping the user interface onto nearby resources (e.g. a desktop computer). For an application like *Teleporting*, the ability to determine the user location relative to nearby resources is a key feature.

The ParcTab [4, 9] system, described in figure 2.1, one of the first mobile computing systems to explore context-awareness, features a similar application: a virtual whiteboard that supports multi-user drawing. People in the same room can easily collaborate using this virtual whiteboard. It can persist from meeting to meeting, become active only when a specific group is meeting and even follow people from room to room, performing *automatic contextual reconfiguration*.

ParcTab features other interesting applications. A voting application allows users to select ballots which are referenced at a particular location — an example of *proximate*

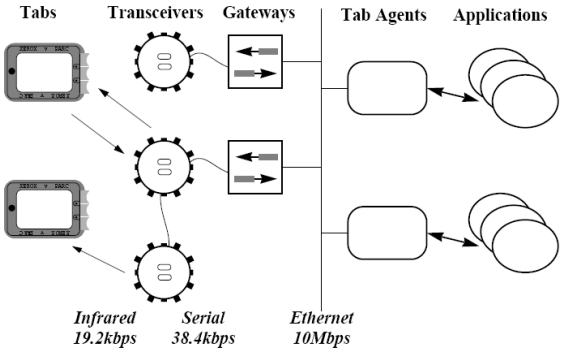


Figure 2.1 – ParcTab system components. Source: The ParcTab Mobile Computing System [9].

selection. There is also a very powerful implementation of Contextual Reminders. A message pops up according to "when, where, who and what" is with you, i.e., the application will pop up when different situations — or contexts — occur, such as "next time I'm near the coffee machine" or "as soon as I leave the meeting room" — this is a *context-triggered action*.

Another context-aware application in the ParcTab system is a location browser that views a "location-based filesystem". The browser changes the displayed directory according to the user's location, i.e., context determines what information is displayed — this is an example of *contextual information*. These location directories may contain information about the location and its occupants, and allow for temporary information to be added, such as notes, instructions or warnings.

Based on the Active Badge system, an automatic call forwarding mechanism, using a PBX with a digital interface designed for computer-integrated telephony, forwards incoming calls to the destination user's nearest phone [8]. This may seem obsolete nowadays with the massive use of GSM phones, but in a scenario were such devices are not allowed (for security reasons, for example) this may prove to be a useful feature.

The context-aware system proposed by Schmidt *et al.* [10] was tested using an application that selects the profile of a GSM phone according to context (profiles are a set of configurations ranging from ringtones to call filtering). In a noisy environment, the phone must ring loudly so it can be heard. In the library, the phone must be silent, so other people are not disturbed. During a meeting, the user may want to filter calls and allow only urgent ones to come through. Figure 2.2 shows a GSM phone connected to a prototype of the sensor board, which is equipped with various sensors, such as accelerometers, photodiodes and temperature sensors, chosen to mimic human senses.

Similarly to ParcTab and Active Badge, the Active Bat [11] system has been tested with some context-aware applications. These include a teleporting application similar to *Teleporting*, some simple location-based games and a large number of user-written small applications, ranging from alerts for people returning to the office to fresh coffee

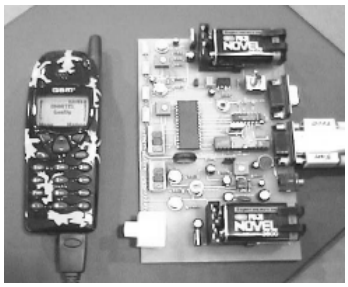


Figure 2.2 – GSM phone connected to a sensor board. Source: Advanced Interaction in Context [10].

notifications.

Developers of the *Relate* system (discussed in section 2.2) prototyped a *spatial file transfer* application [12]. Figure 2.3 shows the user interface: the left panel is a file browser, showing the local file system; the right panel shows the spatial arrangement of nearby devices. This application represents an easy way to share files with users who are in the same room, during a meeting, for example. As the application shows the spatial arrangement of devices, the user can easily identify the destination device — e.g., if he only wants to send a file to the person next to him.

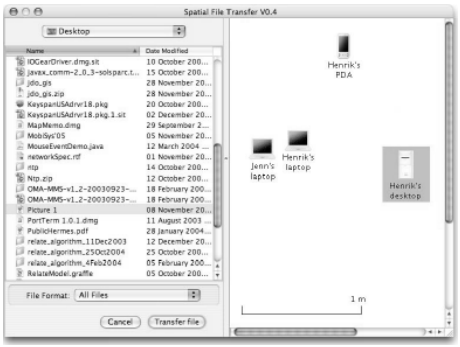


Figure 2.3 – Spatial file transfer application of the *Relate* system. Source: A Relative Positioning System for Co-located Mobile Devices [12].

As we have seen so far, there are different types of applications that can use contextual information, providing new and better services to users. Being an area with several years of research and with the recent and rapid emergence of mobile computing devices, it would be expected that context-aware applications were widely available and routinely used. However, several issues remain to solve before it can be that way. The location determination problem is one of the most relevant, and can severely limit the pervasion of context-aware applications in our daily lives.

2.2 Positioning and Location Systems

As context-aware applications, in particular location-aware applications, need location information, this problem has been addressed by several people in the last few years, taking different approaches. Location-sensing techniques range from triangulation, which determines the position of objects using lateration or angulation, to proximity, which measures the nearness of objects to a known position.

One of the first works in the area is the Active Badge [8], an indoor badge sensing system, developed at the Olivetti Research Laboratory (which later became the AT&T Laboratories Cambridge and was shut down in 2002). Each person wears an infrared emitting badge and a sensor network, depicted in figure 2.4, detects the badges, sending the data to a server. This system uses *centralized location determination*, as the information about the absolute location of people wearing the badge is centralized. Using diffuse infrared cellular proximity, badges did not work well in locations exposed to direct sunlight or other infrared sources.

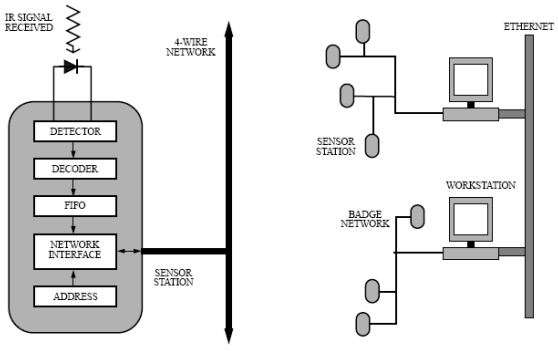


Figure 2.4 – The Active Badge sensor and telemetry network. Source: The Active Badge Location System [8].

Later, the Active Bat [11] location system was developed. This system uses triangulation, by means of an ultrasound time difference of arrival (TDOA) lateration technique, providing better accuracy than the Active Badge. Bat tags are attached to devices and carried by personnel. Figure 2.5 shows one of these tags. They emit ultrasonic pulses which are detected by a grid of receivers, placed in well-determined positions. Each receiver records the time elapsed between pulse emission and reception, allowing the calculation of the Bat position using a multilateration algorithm. An Active Bat can be located to within 9 cm of its true position for 95% of the cases.

Using the same positioning principle as the Active Bat system, the DOLPHIN [13] system consists of several distributed sensor nodes with radio-frequency (RF) and ultrasound capability. Each node has a one-chip CPU used to calculate the position of the nodes, using measured TDOA of ultrasonic pulses. Positioning of sensor nodes requires

Figure 2.5 – A Bat tag. Source: The Anatomy of a Context-Aware Application [11].

less manual configuration than the Active Bat system. Results show a 15 cm accuracy. Figure 2.6 illustrates the DOLPHIN system.

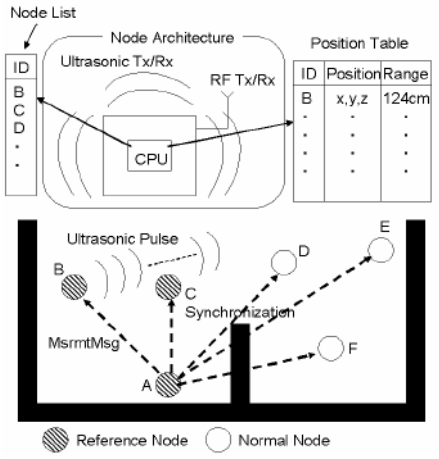


Figure 2.6 – The DOLPHIN system. Source: DOLPHIN: An Autonomous Indoor Positioning System in Ubiquitous Computing Environment [13].

So far, presented systems require a specific fixed-sensor infrastructure. Two major disadvantages arise from this: 1) they are costly and 2) they have limited scalability.

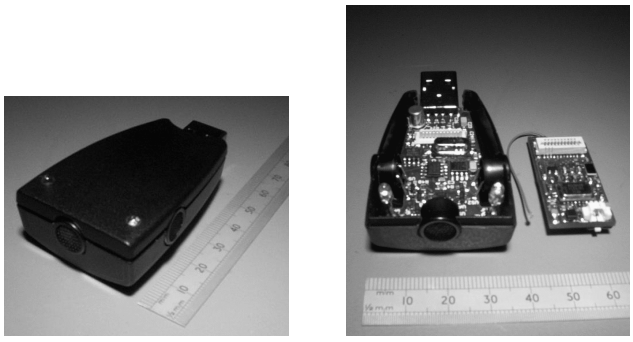
The RADAR [14] system is an in-building RF-based user location and tracking system that uses 802.11 access points (AP) signal strength information to determine user location. RADAR is based on empirical signal strength measurements as well as theoretically computed measurements, using a simple signal propagation model. It compares the signal strength measured in real-time with a measurements data set, empirically collected and theorically computed. Experimental results show that RADAR is able to estimate the location of an user with a median resolution of 2 to 3 meters.

Other 802.11-based location systems, like the ones proposed by Haeberlen *et al.* [15] and by Jin *et al.* [16], also use signal strength information to locate mobile devices. Generally, they compare real-time signal strength information detected by mobile devices with a set of reference data, previously collected. This comparison is done using different mathematical approaches, usually based on statistical and probabilistic models.

These beacon-based systems using signals from 802.11 APs to determine a device's absolute location can represent easily deployable solutions: 802.11 coverage is an increasing reality in many environments and most mobile devices are released with a built-in 802.11 interface. Usually, these systems compare realtime measurements with a database (or map) of reference points with known 802.11 fingerprints. Building such maps requires previous data acquisition tasks, which are time-consuming and often have to be repeated whenever there is a change in the number or location of APs.

To overcome these issues, self-mapping systems are being developed, i.e., systems that are able to build and update a radio map on-the-fly. LaMarca *et al.* [17] introduced a graph algorithm for self-mapping, using an initial set of data from which to begin building a map using data from users, with good results. In fact, with a small uniformly distributed set of data the system produced a map that estimates user location as well as a wardriving[1] database.

The *Relate* [12] system provides relative location information to co-located devices using ultrasound technology for location-sensing and a non-802.11 network based on RF for peer-to-peer communication. Depending on line-of-sight (LOS) conditions, accuracy ranges from 7 cm (good LOS) to 9 cm (limited LOS) with 90% confidence. When a device is moved, the system can update its estimated co-location within seconds, allowing for a fast convergence after a scenario change. The *Relate* system overcomes the dependence on any external infrastructure, however, it requires specific hardware — an ultrasound USB dongle, depicted in figure 2.7 — in each node.



(a) External view. **(b)** Internal view.

Figure 2.7 – The *Relate* hardware: USB dongle. In sub-figure b), note the sensing components (left) and microcontroller (right) circuit boards. Source: adapted from A Relative Positioning System for Co-located Mobile Devices [12].

The NearMe Wireless Proximity Server [18] computes lists of people and things that are nearby, comparing clients' lists of 802.11 APs and respective signal strengths. Using a client-server architecture, the NearMe system collects data from registered clients and

[1] "Wardriving is the act of searching for Wi-Fi wireless networks by a person in a moving vehicle using such items as a laptop or a PDA", *in* http://en.wikipedia.org/wiki/Wardriving, last viewed on 2008/06/29.

responds to queries about nearby people. All the information is centralized and no calibration is required in any of the nodes. The main work of NearMe is performed by the server and all clients have to do is register with the server, report 802.11 signatures and query for nearby people.

Using audible sound to locate objects, Beep [19] is an indoor positioning system, based on 3D multilateration algorithms, developed at the University of California. Figure 2.8 shows an overview of this system. Consisting of a set of acoustic sensors connected to a central server through a wireless network, Beep does not require the user to carry any kind of specific hardware. Using mobile devices' audio support, location computation is controlled by the user, i.e., position is computed on-demand. This guaranties privacy and saves power by avoiding constant communication between devices.

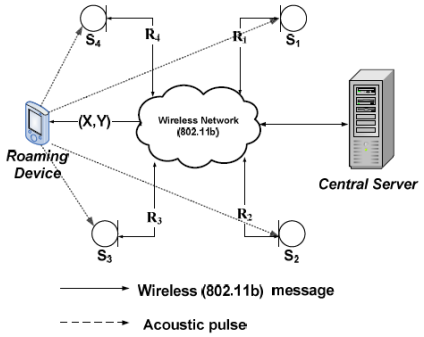


Figure 2.8 – Overview of Beep. Source: Beep: 3D Indoor Positioning Using Audible Sound [19].

Scott *et al.* [20] present a working prototype of an audio location system, using the time-of-flight of audio, involving the use of microphones in the environment. On top of this system, a 3D user interface was implemented, based on human sounds (such as finger clicking), allowing for a new experience of human-computer interaction. Figure 2.9 shows the architecture of this interface.

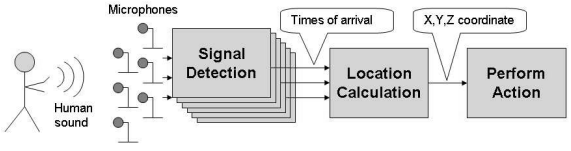


Figure 2.9 – Audio location-based 3D interface system architecture. Source: Audio Location: Accurate Low-Cost Location Sensing [20].

Chapter 3

An Audio/802.11-based Co-location System

In this chapter we propose a *co-location system* that uses the level of audio activity in the environment and the number of common APs detected to determine the co-location of devices, using a simple *near/far* classification. Co-location is computed locally by each device and required information is exchanged over the network. A software was developed to implement the necessary data acquisition, communication, and co-location computing features.

The following sections discuss and analyze the implementation details of the *co-location system*, focusing on the initial requirements used as guidelines, system architecture, and implementation.

3.1 Requirements

To determine the requirements of the *co-location system*, one may begin with identifying who or what will use it and under which conditions. Context-aware applications, like the ones discussed in section 2.1, are good examples of this, as they may use co-location information to serve users.

An application like ParcTab's virtual whiteboard needs to identify people who are in the same room. In a meeting this could be people around a table, who are gathered in a few meters radius. Hotdesking/teleporting applications may need to identify people with a better resolution. Imagine you want to display an application in a co-worker's desktop computer. If there's another computer nearby, the system will need a few centimeters accuracy to differentiate them and display your application on the right computer. Proximate selection applications need less accuracy. If the idea is to emphasize near objects

and make them easier to select, one can adopt a low resolution definition of *near*. So, context-aware applications may have different kinds of accuracy requirements.

Another important requirement is convergence, i.e., how long it takes for the system to detect a context change and act accordingly, determining the new co-location of devices. Applications like Contextual Reminders will require rapid convergence: if I want to be reminded to turn off the coffee machine as I pass near it, the system must warn me when I'm still moving towards it and before I'm moving away. When using the Teleporting system convergence time may not be so critical as the user may be accustomed to long operating system booting times. As with accuracy, there can be different convergence requirements.

Convergence and accuracy requirements may vary not only with the type of application being used, as we have seen, but also with the environment: the number of mobile devices present or the number of different users. How fast and how accurate does our system need to be? In order to support different context-aware applications, the answer is as fast and accurate as possible.

However, in a first approach to the problem, development and implementation used rather modest accuracy and convergence values as guidelines. In fact, the *co-location system* discussed in this dissertation gives a qualitative classification for the co-location of devices: *near* or *far*. Convergence was not evaluated but, during development, algorithms used and programming and communications solutions were chosen taking into account the time it may take the system to classify the co-location.

3.2 Architecture

System architecture is illustrated in figure 3.1 and can be divided in three major components:

1. *Data acquisition*: acquires data from audio and 802.11 interfaces;
2. *Inter-node communication*: requests data from other devices and replies to requests from other devices, over the network;
3. *Co-location classification*: uses acquired and received data to classify the co-location of devices, providing this information to users or context-aware applications.

The data acquisition component continuously collects data needed by the other two components, keeping in memory a First-In, First-Out (FIFO) buffer with the last ten seconds of audio and a buffer with current 802.11 information. This way data is readily available.

The inter-node communication component periodically sends data requests to other devices and processes incoming messages, sending replies to data requests from other devices and sending received data to the co-location classification component.

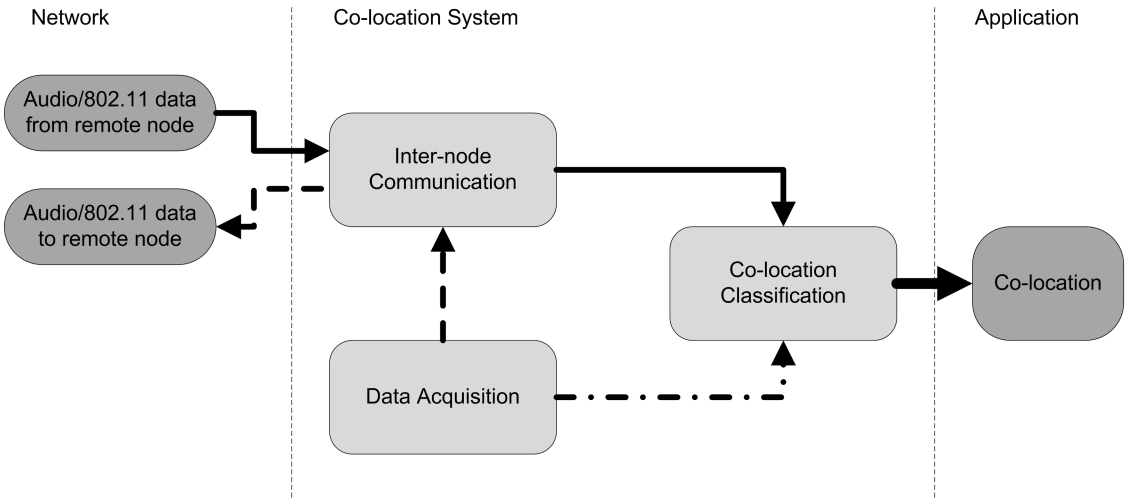


Figure 3.1 – System architecture overview, emphasizing relations between its three major components. This figure also shows the relations between the system, data sources (network) and applications using co-location information

The co-location classification component is the key component of the whole system. It receives data from the other two components and uses three algorithms to determine the co-location of devices:

1. an audio-based algorithm, which computes co-location based in audio information only;

2. an 802.11-based algorithm, which computes co-location based in 802.11 information only;

3. and a final co-location classification algorithm, which uses the output of the previous two algorithms to classify the co-location of the devices.

The following sections of this chapter discuss the implementation of this architecture. In section 3.3 we describe the offline development of the audio-based co-location algorithm which integrates the co-location classification component. A MATLAB implementation of the algorithm is presented.

Section 3.4 discusses the implementation of the three components which integrate the *co-location system*. The audio-based co-location algorithm is implemented in C#. At this stage of development, this is the only algorithm which integrates the co-location classification component.

Section 3.5 describes the implementation of the 802.11 extension. This includes the 802.11-based algorithm and the final co-location classification algorithm, thus completing the co-location classification component.

3.3 Audio-based co-location algorithm

Implementation of the audio-based co-location algorithm was done using an offline approach. Two mobile devices (a PDA and a laptop computer) were used to acquire audio data which was then used to test an implementation of the algorithm in MATLAB.

All audio was collected in PCM format, using a sampling frequency of 11025 Hz, with a resolution of 16 bits per sample. Each pair of audio data sets was then manually synchronized using Audacity (an open-source audio processing software) and imported into MATLAB.

The audio-based co-location algorithm uses a simple approach to decide about the co-location of devices. Sampled values are converted into their absolute values, and made to vary between 0 and 1. Each data set is divided in 100 ms windows and the average value of each window is calculated and stored in a vector — $\overline{x}$ for device 1 (local node) and $\overline{y}$ for device 2 (remote node). These vectors are then input into an algorithm which outputs a g_i value for each pair $(\overline{x}_i, \overline{y}_i)$. The number of elements in these vectors is equal to the number of windows.

Determination of each g_i value, depicted in figure 3.2, is as follows: if both $\overline{x}_i$ and $\overline{y}_i$ are below the activity threshold, m_0, g_i is 0, meaning *blind zone*; if they are both equal to or above m_0, g_i is 2, meaning *near*; otherwise, g_i is -1, meaning *far*. For example, if all $\overline{x}_i$ are above m_0 and all $\overline{y}_i$ are below m_0 then all g_i values will be equal to -1.

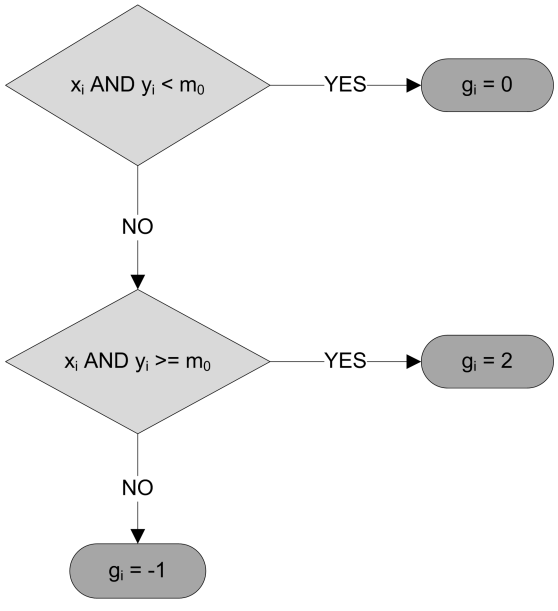


Figure 3.2 – Audio-based algorithm: vector g determination.

At this stage of development, several data sets (collected in different scenarios) were used to find a good value for the activity threshold. Using $m_0 = 3 \times 10^{-3}$ seemed to produce satisfactory results; nonetheless this was an arbitrary choice.

The algorithm then inputs vector g into a *memory function* that outputs a vector p. Each p_i value holds the memory of the evolution of g over time. This is done as shown in the equation

$$p(i) = p(i-1) - 0,05 \times p(i-1) + g(i) \ , \tag{3.1}$$

where $p(i)$ obeys the following conditions:

- $p(0) = g(0)$;
- if $p(i) > 10$ then $p(i) = 10$;
- if $p(i) < -10$ then $p(i) = 10$.

The values chosen for g_i (-1, 0, and 2) cause p_i to decrease, stabilize towards 0, or increase, respectively. The choice of giving twice as much weight to a near event than to a far event was arbitrary and originated in our initial experiments with this algorithm.

Finally, the last value of p (which holds the memory of all values and is $p(N)$ for a N windows division) is evaluated and co-location is classified according to these conditions:

1. if $p(N) = 0$ then no classification is possible and the output is *blind zone*;
2. if $p(N) > 0$ then classification is *near*;
3. if $p(N) < 0$ then classification is *far*.

These classification conditions are rather coarse, but were adopted to simplify the algorithm in this first approach to audio-based co-location determination.

Code listing 3.1 shows the MATLAB implementation of the audio-based co-location algorithm. Audio data collected by devices 1 and 2 is imported into vectors `x` and `y`, respectively. In lines 1 and 2, they are converted into their absolute values. Code in line 7 calculates the number of windows needed, which is $W = N/L$, where N is the length of both vectors (they must be of the same length) and L is the window length ($L = 1100$). The `fix` function truncates the result of this operation, so that W is an integer. In lines 11 and 12, the average value of window `i` is calculated: `xm` represents $\overline{x}$, and `ym` represents $\overline{y}$. Variable `g` represents the g_i value, for each iteration `i`, and its value is determined in the code block between lines 15 and 23. Each value p_i of the memory function output vector p, represented by variable `p`, is calculated in the code block between lines 30 and 34. Code in lines 36 and 37 prepares the window lower and upper limits for the next iteration.

Once this algorithm was tested, the development of the complete *co-location system* began.

Code Listing 3.1 – MATLAB implementation of the audio-based co-location algorithm.

```
x = abs(x);
y = abs(y);

window_init = 1;
window_end = L;

W = fix(N/L);

for i = 1:W

    xm(i) = sum(x(window_init:window_end))/L;
    ym(i) = sum(y(window_init:window_end))/L;


    if ( xm(i) < m0 && ym(i) < m0)
        g = 0;
    else
        if ( xm(i) >= m0 && ym(i) >= m0)
                g = 2;
        else
                g = -1;
        end
    end

    if (i == 1) p(i) = g;
    else
        p(i) = p(i-1) - 0.05*p(i-1) + g;
    end

    if (p(i) > 10) p(i) = 10;
    else
        if (p(i) < -10) p(i) = -10;
        end
    end

    window_init = window_init + L;
    window_end = window_end + L;

end
```

3.4 Co-location system

To implement, and test, the *co-location system*, a console-based software application was developed, using the C# programming language.

As seen in section 3.2, the architecture of the *co-location system* presents three major components: the inter-node communication, data acquisition, and co-location classification components.

The inter-node communication component uses TCP/IP sockets to send and receive messages over the network. Two classes are used for this:

1. class `tcpListener`, which listens to incoming connections and receives incoming messages;
2. class `tcpSender`, which sends messages to the remote node.

Implementation uses two functions to handle incoming messages. Function `tListener` receives incoming messages using an instance of `tcpListener` and sends them to another function — `processIncomingMsg`. Function `processIncomingMsg` processes received messages. Incoming audio and 802.11 data is sent to the co-location classification component. Incoming data requests are handled as follows: 1) after identifying the type of request, data is obtained from the data acquisition component; 2) an instance of the `tcpSender` class is used to send the requested data to the remote node.

Data requests are made using only one function — `tRequestInfo`. This function sends periodic (every 10 seconds) requests of audio and 802.11 data to the remote node, using an instance of the `tcpSender` class.

The data acquisition component, as the name suggests, acquires data using the audio and 802.11 interfaces. Audio data is collected in the PCM format, with a sampling frequency of 11025 Hz and a resolution of 16 bits per sample. Therefore, the storage of t seconds of audio data requires n bytes of memory, as shown in the following equation:

$$n = \frac{t \times f_s \times k}{8} \ , \tag{3.2}$$

where f_s is the sampling frequency, in Hz, and k is the number of bits per sample. The system maintains the last 10 seconds (approximately) of audio in memory, so the used FIFO buffer size is 222200 bytes.

To avoid the transmission of large amounts of data over the network, and consequent bandwidth or network delay problems, nodes do not exchange the actual audio data they acquire. Instead, they exchange the window average values vector corresponding to the acquired audio data. Thus, the audio-based co-location algorithm only computes this vector for local node's audio data and the data acquisition component provides the window average values vector to the inter-node communication component.

The audio-based co-location algorithm, described in section 3.3, was the first algorithm to integrate the co-location classification component. Code listing 3.2 shows the C#

implementation of the algorithm. Variable `avgOther` corresponds to vector $\overline{y}$ and contains data that the remote node sent over the network. Variable `avgOwn` corresponds to vector $\overline{x}$ and is calculated using two functions: 1) the first, function `getSamples`, converts data in the FIFO buffer to an array of floating point values; 2) the second, function `getAverage`, converts values in the array to their absolute values, and computes the average value of each window of 1100 values. Variable `g` corresponds to value g_i — calculated in the code block between lines 3 and 5 — and the array `results` corresponds to vector p — each p_i value is calculated in the code block between lines 7 and 11.

Code Listing 3.2 – C# implementation of the audio-based co-location algorithm

```
1  for (i = 0; i < avgOwn.Length; i++)
2  {
3    if ((avgOwn[i] < m0) && (avgOther[i] < m0)) g = 0;
4    else if ((avgOwn[i] >= m0) && (avgOther[i] >= m0)) g = 2;
5    else g = -1;
6
7    if (i == 0) results[i] = g;
8    else results[i] = results[i - 1] - results[i - 1] * (float)0.05 + g;
9
10   if (results[i] > 10) results[i] = 10;
11   else if (results[i] < -10) results[i] = -10;
12 }
13 return results[results.Length - 1];
```

Figure 3.3 shows an overview of this implementation of the co-location classification component, which, at this stage, only uses audio information to determine the co-location of devices.

3.5 802.11 extension

The implementation of the *co-location system* as described in section 3.4 was completed with the development and integration of an 802.11 extension. This integrates the co-location classification component of the *co-location system* and uses acquired 802.11 information to provide additional co-location information and improve the behaviour of the system in some scenarios.

This extension requires the acquisition of 802.11 data. The data acquisition component was modified to provide this feature. 802.11 acquired data offers information ranging from the interface name to RSSI values, but only detected APs' MAC addresses are stored and used.

The 802.11 extension includes two new algorithms into the co-location classification component: an 802.11-based co-location algorithm and a final co-location classification algorithm. Figure 3.4 shows the inner architecture of the co-location classification component with the 802.11 extension, focusing on the relation between its three algorithms.

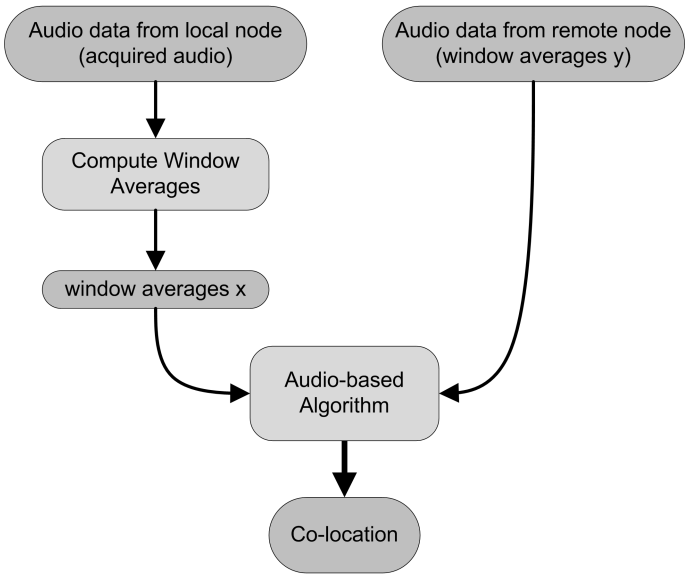


Figure 3.3 – Co-location classification component: audio-based algorithm, emphasizing data input (the data acquisition component provides data from the local node and the inter-node communication component provides data from the remote node) and co-location output. The "compute window averages" block represents operations done by functions `getSamples` and `getAverage`.

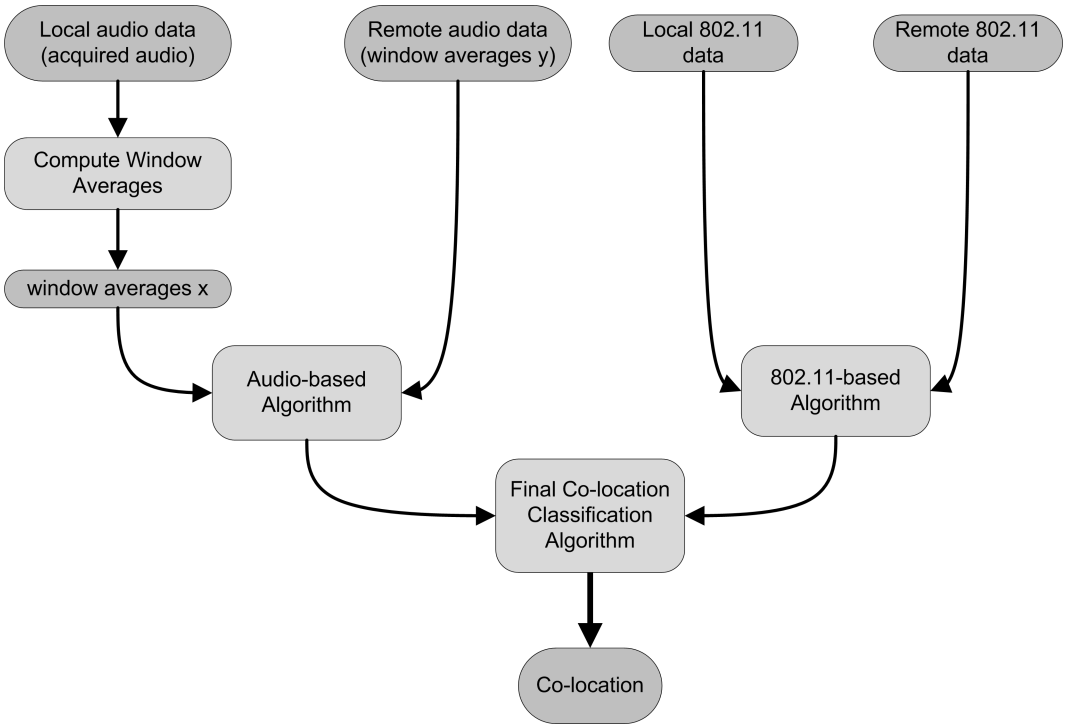


Figure 3.4 – Co-location classification component: audio-based and 802.11-based algorithms, and final co-location classification algorithm. Note the relation between the algorithms, as well as the data flow within the component.

The 802.11-based algorithm uses MAC addresses of detected APs to classify the co-location of devices, using a very simple approach: if devices detect zero APs in common, their co-location is classified as *far*; otherwise, their co-location is classified as *near*.

The final co-location classification algorithm uses the output of the audio-based and the 802.11-based co-location algorithms to decide about co-location, and provides the final output of the co-location classification component. This is, also, a very simple algorithm. If the output of the 802.11-based co-location algorithm is *far*, then the final output is *far*. Otherwise, the final output is equal to the audio-based co-location algorithm.

Code listing 3.3 shows the C# implementation of the final co-location classification algorithm. Variables `_wifiResult` and `_audioResult` contain the output of the 802.11-based and the audio-based algorithms, respectively. In line 3, the code verifies the output of the 802.11-based algorithm. If it is not *far*, it proceeds and in the code block between lines 7 and 9 it classifies the co-location according to the output of the audio-based algorithm.

Code Listing 3.3 – C# implementation of the final co-location classification algorithm, included in function `ComputeLocation`. Condition in the first line guaranties that the algorithm only proceeds if both audio-based and 802.11-based algorithms have produced results.

```
1  if (_wifiResult != -100 && _audioResult != -100)
2  {
3    if (_wifiResult == 0) coLocation = "Far";
4
5    else
6    {
7      if (_audioResult == 0) coLocation = "Blind Zone";
8      else if (_audioResult > 0) coLocation = "Near";
9      else if (_audioResult < 0) coLocation = "Far";
10   }
11 }
```

With this 802.11 extension, the implementation of the *audio/802.11-based co-location system* is complete. In chapter 4, improvements introduced with this extension are evaluated and analyzed.

Chapter 4

Evaluation

Test results are presented and analyzed in this chapter. First tests focused on the audio-based co-location algorithm and were conducted using an offline approach, in which the algorithm was implemented using MATLAB and input data was previously acquired. After the offline tests, an online approach was used to test the *co-location system*. Two configurations were used: the first one only uses the audio-based co-location algorithm to compute the co-location of devices; the second also uses the 802.11 information to complement the audio data, i.e., uses the complete co-location classification component, with the audio-based co-location algorithm and the 802.11 extension.

In all tests, two different mobile devices were used. For the offline approach, audio data sets were collected using a laptop computer and a PDA. For the online approach, and due to software limitations on the PDA, two laptop computers were used. Co-location is classified as *near* or *far*; a *blind zone* classification is given in situations where the detected audio activity in both devices is below a threshold.

Audio sampled values vary between -1 and 1. In figures showing collected audio data, each point represents the average value of the absolute value of samples in an 1100 samples window (approximately 100 ms).

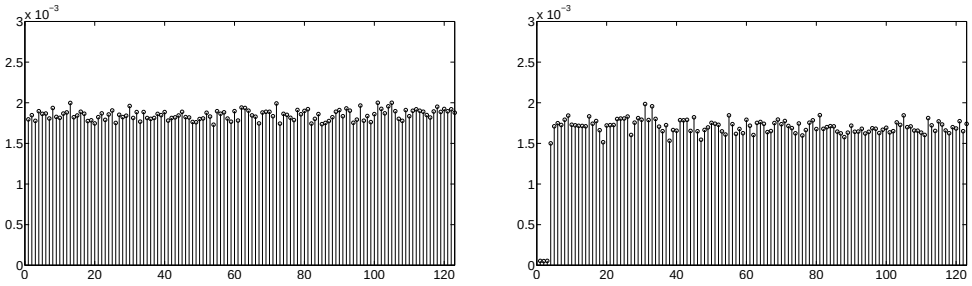
In figures showing audio algorithm output, each point represents the output for a given window. Values vary between -10 and 10, where -10 is *far*, 10 is *near* and 0 is *blind zone*. The dashed line represents the audio activity threshold, $m_0 = 3 \times 10^{-3}$.

4.1 Offline Audio-based Algorithm

4.1.1 Test scenarios description

To test the audio-based co-location algorithm used in the *co-location system*, some audio data sets were collected in different scenarios for using later with the MATLAB implementation of the algorithm.

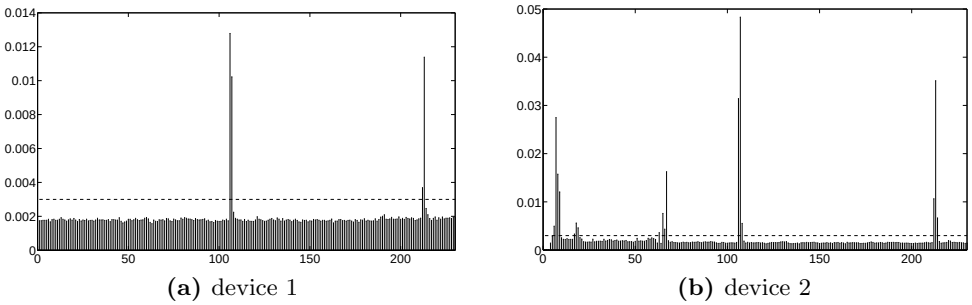
In scenario A1, devices were located near to each other, in the same room, on top of the same desk. There were no loud sound events and only background noise was recorded — mainly computer cooling fans. Figure 4.1 shows the window average values captured on both devices. We can observe that the window average values are always below the activity threshold $m_0 = 3 \times 10^{-3}$, and according to our audio algorithm expect results to be in the *blind zone*.



(a) device 1 **(b)** device 2

Figure 4.1 – Collected audio data for scenario A1.

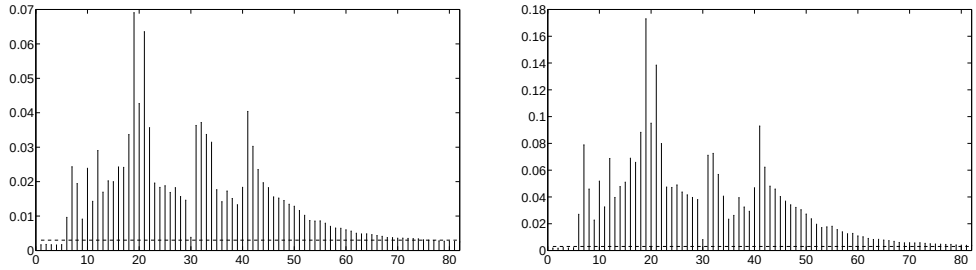
In scenario A2, audio samples were collected in the same room, where the devices were located in opposite corners. There was some background noise and several louder events took place. Figure 4.2 shows the data that corresponds to this scenario. The dashed line is the 3×10^{-3} activity threshold. We can observe that device 1 detected two events above the threshold and device 2 detected five. Expected results include three situations: a) a *near* classification when both devices detected an event at the same time; b) a *far* classification when a device detects an event the other does not; c) a *blind zone* classification whenever both devices detect audio below the activity threshold.



(a) device 1 **(b)** device 2

Figure 4.2 – Collected audio data for scenario A2.

In scenario A3, devices were placed on top of a desk, in the same room. A guitar string was plucked. Other than that the room was quiet, in similar conditions to scenario A1. Figure 4.3 shows window average values are mainly above the activity threshold. In this scenario, test results should classify the co-location as *near*.



(a) device 1 **(b)** device 2

Figure 4.3 – Collected audio data for scenario A3.

4.1.2 Results

The results for these scenarios were obtained by feeding the collected data as input to the audio algorithm on MATLAB. Figures 4.4 to 4.6 show the algorithm output for each scenario throughout the period in which the audio data was collected.

4.1.3 Analysis

Figure 4.4 shows results for scenario A1 are as expected: with no activity detected by either of the devices, the algorithm output is 0 for all windows, so classification is *blind zone*.

In scenario A2, a sequence of five sound events took place. Audio activity was below the threshold at all times except during these events. The first three events were only detected by device 2. Therefore, the algorithm classified the co-location as *far* tending towards the *blind zone* some time after each event. For example, we can observe that the output of the algorithm after event 1 started tending towards the blind zero but was immediately reinforced by event 2 and later by event 3. Both devices detected the last two events. As these are *near* events, event 4 forced the slowly decaying output of the algorithm after event 3 to go up towards near. Event 5 was much later than event 4 and, as such, the reinforcement noticed between events 1, 2, and 3 was not noticed between events 4 and 5. This results are as expected because of the memory property of the algorithm (eq. 3.1).

Results for scenario A3 are, also, as expected. In fact, most of the audio recorded on both devices was above the activity threshold, which means that two nearly located mobile devices would classify, using this algorithm, their co-location as *near* for most of the time. We also notice that the output of the algorithm is capped at +/- 10, which explains why the values on figure 4.6 don't grow further given that there is simultaneous activity in both devices, nor decay towards the blind zone.

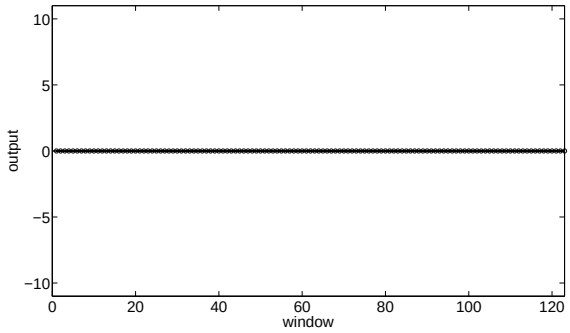


Figure 4.4 – Audio-based algorithm output for scenario A1.

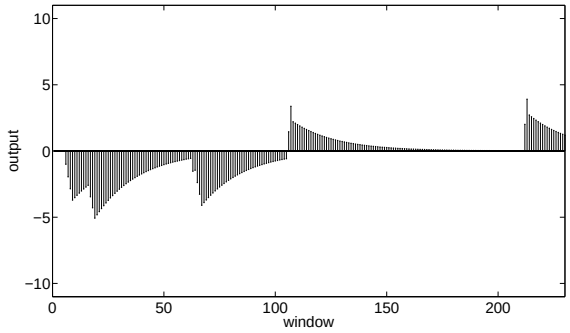


Figure 4.5 – Audio-based algorithm output for scenario A2.

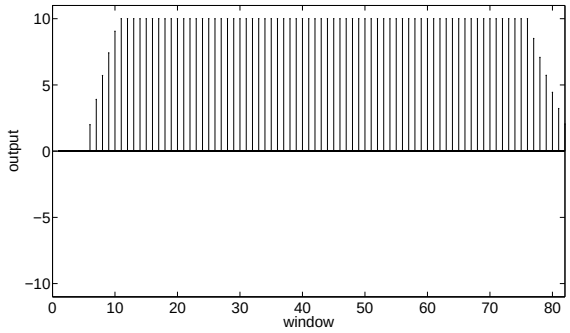


Figure 4.6 – Audio-based algorithm output for scenario A3.

4.2 Online Audio-based Algorithm

4.2.1 Test scenarios description

Five different indoor scenarios were used for these online tests. Figures 4.7 to 4.11 show the data (window average values) that nodes collected and exchanged in these scenarios.

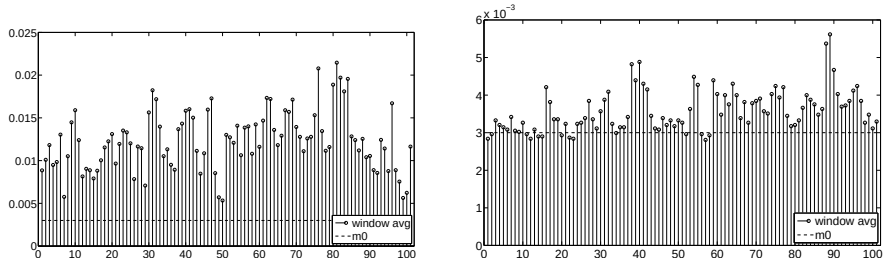
In scenario B1, both devices were in the same room, on top of the same desk, next to each other. There was music playing loud enough for the window average values to be mainly above the activity threshold. Expected result for this scenario is a correct classification of the devices' co-location — in this case, *near*.

In scenario B2, both devices were far from each other, in different rooms. There was music playing in both rooms. Two different songs were used and the rooms were on opposite edges of the floor, making it impossible to hear what was going on in the other room. As the algorithm only takes into account the level of audio activity and the music in each room was loud enough to cause both devices to detect audio activity, the expected result is an incorrect classification of the devices' co-location — *near* instead of *far*.

For scenario B3, devices were near each other, just as in scenario B1. Microphones of both devices were turned off by setting their volume to zero in the operating system. There were people talking in the room. With no audio being recorded, the expected result is a *blind zone* classification.

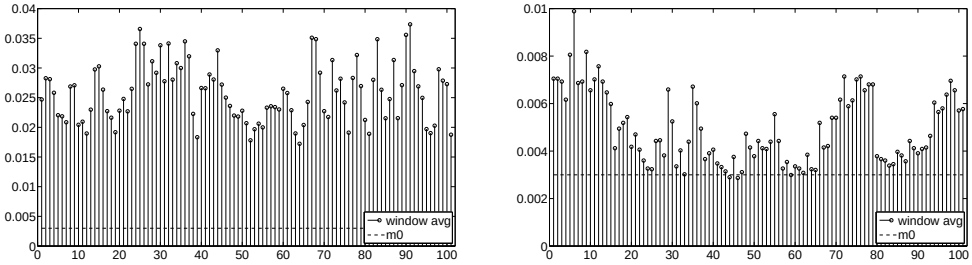
Scenarios B4 and B5 are similar to scenario B2 regarding the physical location of the devices. In scenario B4 there was music playing in the room where device 1 was located and the room where device 2 was located was quiet (similar to offline scenario A1). In scenario B5 these sound conditions were inverted; the room where device 1 was located was quiet and music was playing in the room where device 2 was located. For these two scenarios, the expected result is a correct classification of the devices' co-location — in this case, *far*.

Calibration of the devices for all test scenarios was as follows: for device 1, $fm_1 = 1$; for device 2, $fm_2 = 0.09$. This calibration, and the need for it, is discussed in section 4.2.4.



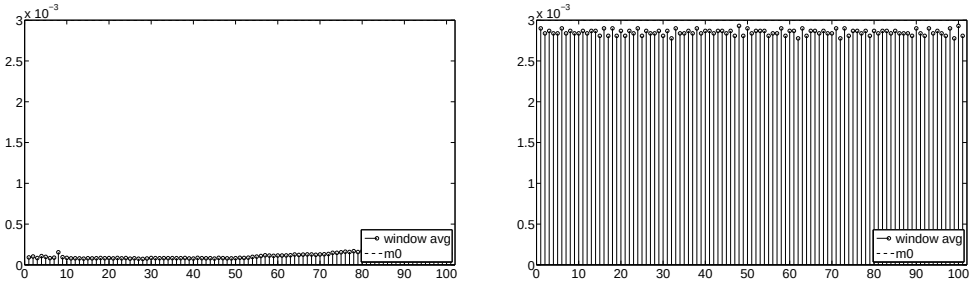
(a) device 1 **(b)** device 2

Figure 4.7 – Collected audio data for scenario B1.



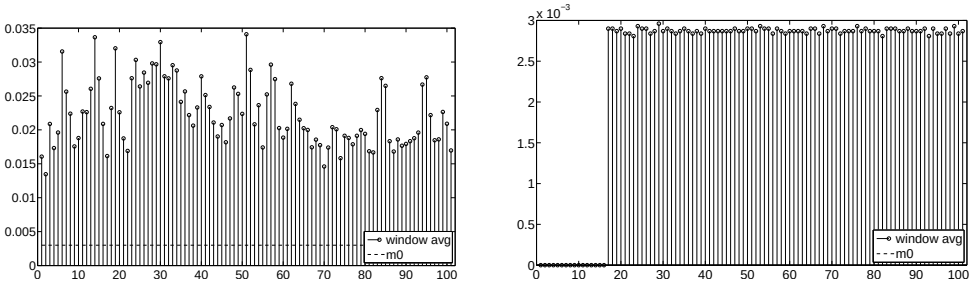
(a) device 1 **(b)** device 2

Figure 4.8 – Collected audio data for scenario B2.



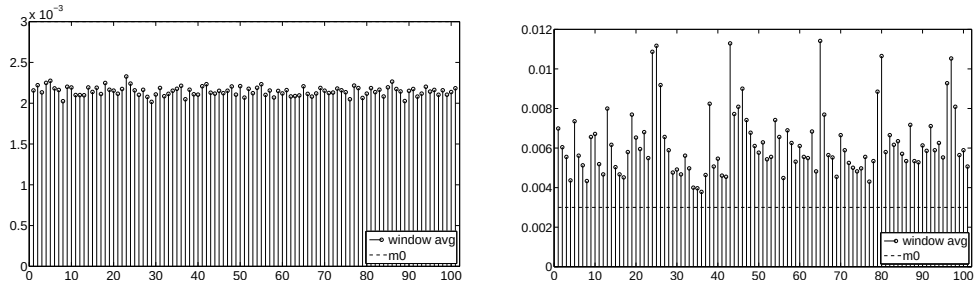
(a) device 1 **(b)** device 2

Figure 4.9 – Collected audio data for scenario B3.



(a) device 1 **(b)** device 2

Figure 4.10 – Collected audio data for scenario B4.



(a) device 1 **(b)** device 2

Figure 4.11 – Collected audio data for scenario B5.

4.2.2 Results

Obtained results are summarized in table 4.1.

Table 4.1 – Online audio test results.

Scenario	Conditions		Co-location	Expected Result	Audio algorithm output
	Device 1	Device 2			
B1	Noise	Noise	Near	Near	Near
B2	Noise	Noise	Far	Near	Near
B3	Microphone off		-	Blind Zone	Blind Zone
B4	Noise	Silence	Far	Far	Far
B5	Silence	Noise	Far	Far	Far

Figures 4.12 to 4.16 show the complete audio algorithm output for the different online audio test scenarios.

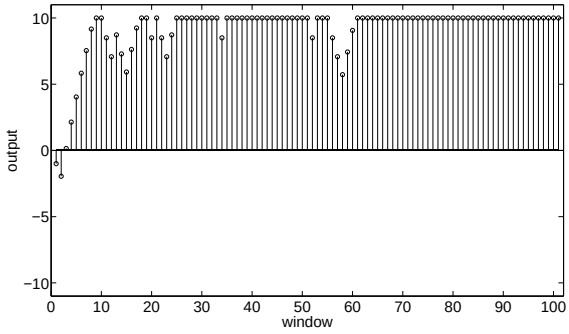


Figure 4.12 – Audio-based algorithm output for scenario B1.

4.2.3 Analysis

As shown in table 4.1, results for online audio algorithm tests in scenarios B1 to B5 are as expected.

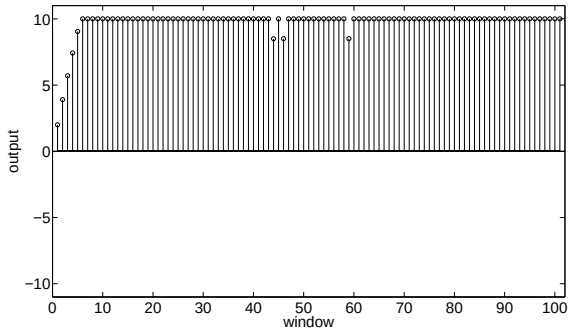


Figure 4.13 – Audio-based algorithm output for scenario B2.

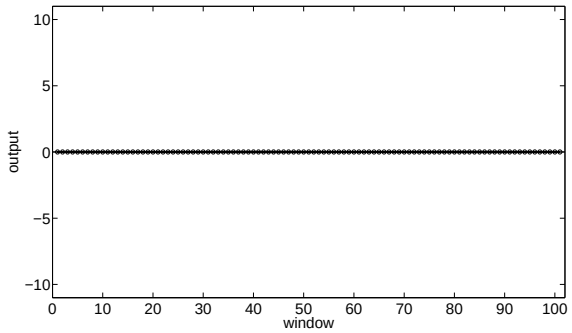


Figure 4.14 – Audio-based algorithm output for scenario B3.

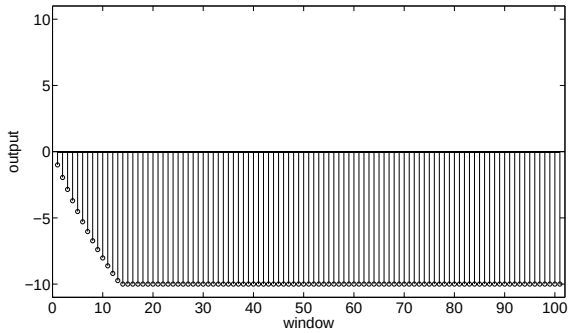


Figure 4.15 – Audio-based algorithm output for scenario B4.

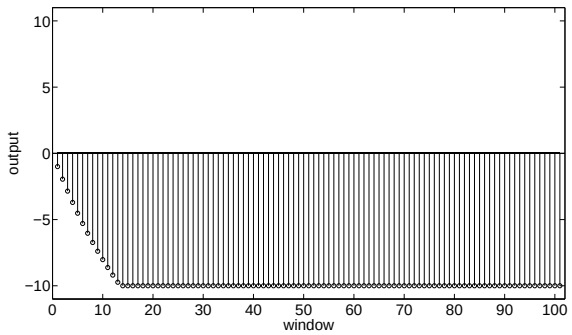


Figure 4.16 – Audio-based algorithm output for scenario B5.

In scenario B1, devices were located nearby in an environment with audio activity above the threshold. In figure 4.12 it is possible to observe that the algorithm output for nearly every window is 10. The last window value, which contains the memory of previous values, is 10 and, therefore, the system classified the co-location as *near*.

In scenario B2, devices were far from each other and detected audio activity above the threshold. In figure 4.13 it is possible to observe that the algorithm output for nearly every window is 10. The last window value is 10 and, therefore, the system incorrectly classified the co-location as *near*. This is an expected limitation of the implemented audio algorithm which comes from the fact that the decision is based on the level of audio activity.

In scenario B3, the volume of both devices' microphone was set to zero. In figure 4.14 it is possible to observe that the algorithm output of every window is 0. Consequently, the system output is *blind zone* as there was no sufficient information to conclude about the co-location of the devices.

In scenario B4, devices were far from each other. Device 1 detected audio activity above the threshold while device 2 did not. In figure 4.15 it is possible to observe that the algorithm output for nearly every window is -10. The last window value is -10 and the system correctly classified the co-location as *far*.

In scenario B5, devices were also far from each other. Audio characteristics of the environment were inverted in relation to scenario B4: device 1 detected no audio activity above the threshold while device 2 did. In figure 4.15 it is possible to observe that the algorithm output for nearly every window is -10. The last window value is -10 and the system correctly classified the co-location as *far*.

4.2.4 System calibration

During some preliminary tests, data collected with device 2 was always above the activity threshold, even in a completely silent environment. This was due to the high sensitivity of the microphone. To overcome this, as seen in section 3.4, a calibration mechanism was implemented where collected values are attenuated by multiplication by a factor fm — with $fm \in [0; 1]$.

For the two devices, the following calibration procedure was used:

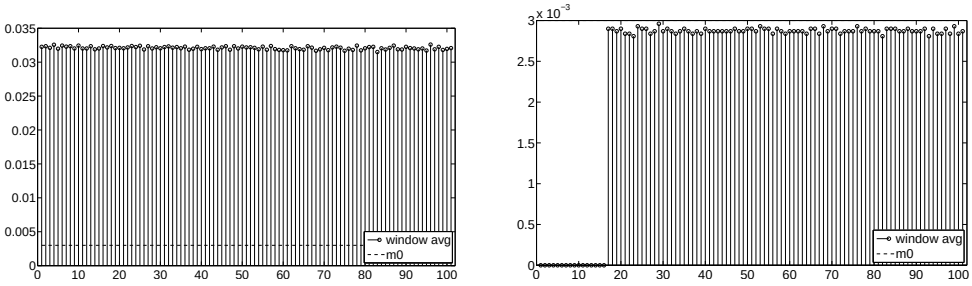
1. Devices were place in an environment with low background noise — an open space working environment —, next to each other.

2. Data was collected and analyzed to see if any of the devices detected audio activity above the threshold.

3. For each of the devices, a value for fm was determined, in order to attenuate collected data to levels below the threshold.

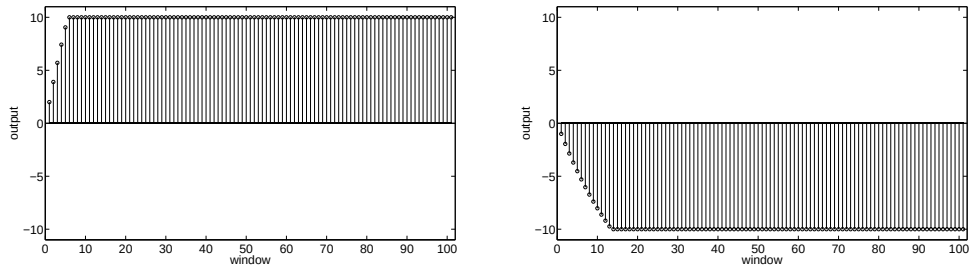
Determined values were $fm_1 = 1$, for device 1 — which means no calibration was in fact necessary for this device —, and $fm_2 = 0.09$, for device 2.

Test results with and without this calibration show it contributes towards better results. Figure 4.17 shows data collected by device 2, in scenario B4, with and without calibration. Figure 4.17a clearly shows values well above the activity threshold, while figure 4.17b shows values below the threshold. Comparison of results with and without calibration is shown in figure 4.18, where figure 4.18a shows no calibration lead to an incorrect decision of *near* — last window value is 10 —, and figure 4.18b shows calibration lead to a correct decision of *far* — last windows value is -10.



(a) no calibration **(b)** $fm_2 = 0.09$

Figure 4.17 – Data collected by device 2, with and without calibration, in scenario B4.



(a) no calibration **(b)** $fm_1 = 1$, $fm_2 = 0.09$

Figure 4.18 – Algorithm output comparison, with and without calibration, for scenario B4.

4.3 Online Audio-based algorithm and 802.11 extension

4.3.1 Test scenarios description

Test scenarios C2 and C3, used for online audio/802.11-based *co-location system* tests, are the same as scenarios B2 and B3, respectively. These scenarios are described in section 4.2.1. Audio calibration is the same as for tests discussed in section 4.2: $fm_1 = 1$, $fm_2 = 0.09$. Figures 4.19 and 4.20 show the window average values captured on both devices. Tables 4.2 and 4.3 show the MAC addresses of APs that each device detected.

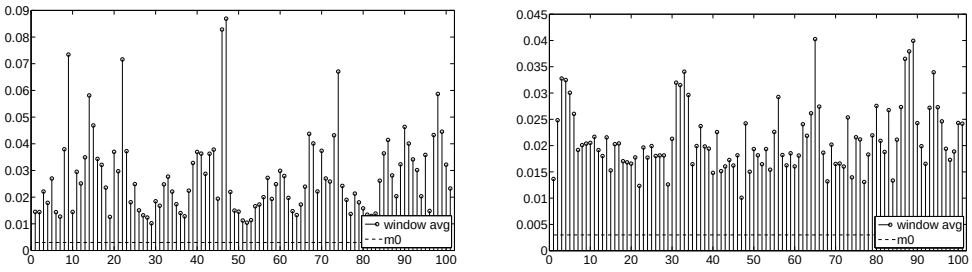
With the integration of the 802.11 extension into the system, expected results for these scenarios are altered. Both devices were located so that they can't detect any APs in common: a co-location classification of *far* is then expected.

Table 4.2 – MAC addresses of detected APs in scenario C2.

MAC addresses of detected APs	
Device 1	Device 2
00:11:20:8d:ed:30	00:0f:90:8a:96:a0
00:16:9c:48:59:e0	00:19:e7:26:4b:70
00:0e:d7:c2:23:00	00:0e:d7:c2:03:10
00:12:01:b5:80:c0	00:0e:d7:c1:f9:70
00:16:9c:48:4a:60	00:17:df:a8:a8:30
	00:0e:83:90:10:50
	00:0e:d7:c2:07:60
	00:0e:d7:b0:f5:00
	00:17:df:a8:a2:e0
	00:17:df:a8:a0:10
	00:0e:d7:b1:36:70

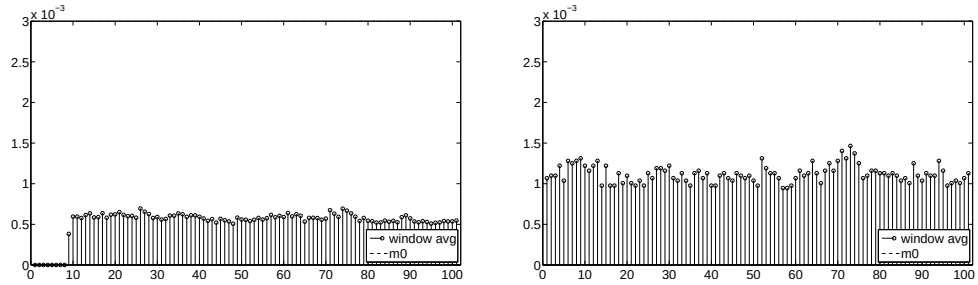
Table 4.3 – MAC addresses of detected APs in scenario C3.

MAC addresses of detected APs	
Device 1	Device 2
00:0e:d7:c2:23:00	00:0f:90:8a:96:a0
00:11:20:8d:ed:30	00:17:df:a8:a0:10
00:16:9c:48:59:e0	00:17:df:a8:a8:30
	00:0e:83:90:10:50
	00:0e:d7:c2:07:60
	00:0e:d7:c1:f9:70
	00:17:df:a8:a2:e0
	00:19:e7:26:4b:70



(a) device 1 **(b)** device 2

Figure 4.19 – Collected audio data for scenario C2.



(a) device 1 **(b)** device 2

Figure 4.20 – Collected audio data for scenario C3.

4.3.2 Results

Table 4.4 summarizes results obtained during online audio and 802.11 tests.

Table 4.4 – Online audio and 802.11 test results.

Scenario	Co-location	Audio algorithm output	Common APs	Expected decision	Decision
C2	Far	Near	0	Far	Far
C3	Far	Blind zone	0	Far	Far

Figures 4.21 and 4.22 show the complete audio-based co-location algorithm output for these tests.

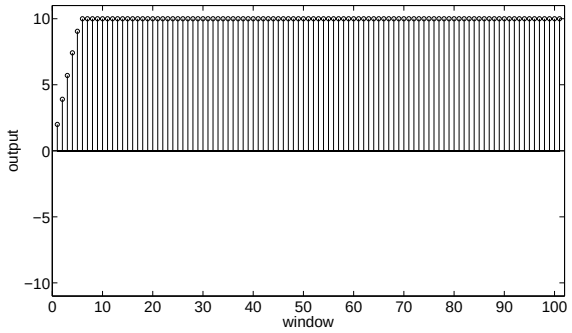


Figure 4.21 – Audio-based algorithm output for scenario C2.

4.3.3 Analysis

Results obtained using the audio-based algorithm and complementary 802.11 information were as expected. When a device's AP MAC address list has no addresses in common with the other device's list, the system always classifies the co-location as *far*. This overcomes the limitation results of scenario B2, emphasized in section 4.2, and allows for the correct classification of the co-location of two devices in distinct noisy environments, as

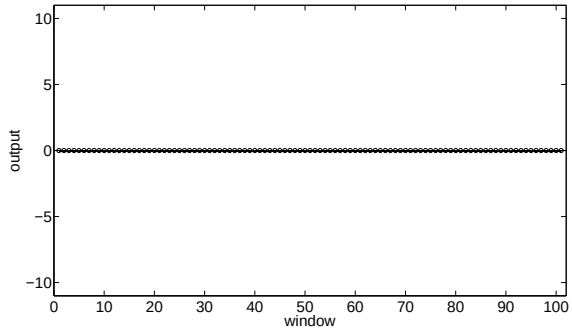


Figure 4.22 – Audio-based algorithm output for scenario C3.

results from scenario C2 show. The audio algorithm classified the devices' co-location as *near*, as shown in figure 4.21, but as there were no common AP MAC addresses this classification is corrected and the final decision is *far*.

Results of test scenario C3 show this configuration can also overcome the *blind zone* limitation in a *far* situation where no common APs exist in the devices' list. Figure 4.22 shows the audio-based algorithm classified the co-location as being in the *blind zone* but AP MAC addresses information allows the system to decide for a *far* classification.

So, using simple 802.11 information such as AP MAC addresses allows for a better behaviour of the system in some *far* situations, overcoming certain limitations the system using only the audio algorithm revealed.

Chapter 5

Conclusions and Future Work

In this dissertation we have attempted to describe, with reasonable detail, the work that led to the implementation and evaluation of a *co-location system* for mobile devices with audio and wireless interfaces.

This chapter discusses the contribution of this work and obtained results, points out some limitations of the current implementation of the *co-location system*, and proposes solutions to these limitations and enhancements to the *co-location system* that may be addressed in future work.

5.1 Contribution and Results

This work contributes with a possible solution for an easy to deploy, low-cost and effective audio/802.11-based *co-location system*. The implemented *co-location system* uses audio and 802.11 interfaces commonly available in mobile devices, and a distributed system to exchange information, to determine the co-location of mobile devices.

In the majority of the test scenarios, the *co-location system*, using an audio-based algorithm, an 802.11-based algorithm and a final co-location classification algorithm, can correctly classify the co-location of two mobile devices. These results show that there's good potential in using audio and 802.11 information to determine the co-location of devices.

5.2 Future Work

Although satisfying results were obtained with this work, some possibilities of improvement can be explored. There are some limitations that need to be solved and possible enhancements, to include in future versions, may be suggested.

First of all, this implementation of the *co-location system* only supports two devices at the same time. A future version needs to improve this, and allow for the support of a large number of mobile devices, so that the *co-location system* can be successfully deployed in a real-life environment.

Developed co-location algorithms can, also, be the object of future work. The audio-based co-location algorithm only uses audio activity levels to determine the co-location. Although effective in some scenarios, it leads to incorrect classifications in others. Also, the algorithm only uses the last value of the memory function output, p, to classify co-location. Perhaps more information could be extracted if the variation of $p(i)$ values was taken into account. The audio-based co-location algorithm can be improved to overcome these limitations.

The 802.11-based co-location algorithm is used in a limited way. In fact, its contribution basically helps to overcome some limitations of the audio-based algorithm. Further development of this algorithm could lead to a more thorough use of 802.11 information in the classification of the co-location.

An interesting enhancement in future versions of the *co-location system* would be the implementation of an automatic calibration mechanism. It would make the system work almost out-of-the-box, and easier to use.

In general, using the same architecture we used, the *co-location system* can be enhanced with the improvement of its audio-based and 802.11-based algorithms, the improvement of its capacity (adding support to a large number of devices at the same time), and the development of an automatic calibration mechanism. With such enhancements, the *co-location system* could gain in accuracy, scalability, and reliability, thus becoming a better solution for determining the *co-location of mobile devices with audio and wireless interfaces*.

References

[1] M. Weiser. Hot topics-ubiquitous computing. *Computer*, 26(10):71–72, Oct 1993.

[2] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[3] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the* 1^{st} *international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[4] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pages 85–90, 1994.

[5] Jeffrey Hightower and Gaetano Boriello. Location sensing techniques. Technical Report UW-CSE-01-07-01, University of Washington, Computer Science and Engineering Department, July 2001.

[6] Jeffrey Hightower and Gaetano Borriello. A survey and taxonomy of location systems for ubiquitous computing. Technical Report UW-CSE 01-08-03, University of Washington, Computer Science and Enginnering, August 2001.

[7] Frazer Bennett, Tristan Richardson, and Andy Harter. Teleporting - making applications mobile. In *Proceedins of IEEE Workshop on Mobile Computing Systems and Applications*, pages 82–84, 1994.

[8] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 40(1):91–102, 1992.

[9] Bill Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34–39, 1993.

[10] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC '99: Proceedings of the* 1^{st} *international symposium on Handheld and Ubiquitous Computing*, pages 89–101. Springer-Verlag, 1999.

[11] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *Proceedings of the* 5^{th} *Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 1999)*, pages 59–68, 1999.

[12] Mikes Hazas, Christian Kray, Hans Gellersen, Henoc Agbota, Gerd Kortuem, and Albert Krohn. A relative positioning system for co-located mobile devices. In *Proceedings of MobiSys '05: The* 3^{rd} *International Conference on Mobile Systems, Applications, and Services*, pages 177–190, June 2005.

[13] Yasuhiro Fukuju, Masateru Minami, Hiroyuki Morikawa, and Tomonori Aoyama. DOLPHIN: An autonomous indoor positioning system in ubiquitous computing environment. *IEEE Workshop on Software Technologies for Future Embedded Systems*, pages 53–56, 2003.

[14] V.N. Bahl, P.; Padmanabhan. RADAR: an in-building RF-based user location and tracking system. *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2:775–784 vol.2, 2000.

[15] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *MobiCom '04: Proceedings of the* 10^{th} *annual international conference on Mobile computing and networking*, pages 70–84. ACM Press, 2004.

[16] Ming-Hui Jin, Eric Hsiao-Kuang Wu, Yun-Bin Liao, and Hui-Chun Liao. 802.11-based positioning system for context aware applications. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, 2:929–933 Vol.2, Dec. 2003.

[17] Anthony LaMarca, Jeff Hightower, Ian Smith, and Sunny Consolvo. Self-Mapping in 802.11 Location Systems. In *UbiComp 2005: Proceedings of the* 7^{th} *International Conference on Ubiquitous Computing*, pages 87–104, 2005.

[18] John Krumm and Ken Hinckley. The NearMe wireless proximity server. In *Proceedings of UbiComp 2004: The* 6^{th} *International Conference on Ubiquitous Computing*, pages 283–300, September 2004.

[19] Atri Mandal, Cristina V. Lopes, Tony Givargis, Amir Haghighat, Raja Jurdak, and Pierre Baldi. Beep: 3D indoor positioning using audible sound. *IEEE Consumer Communications and Networking Conference (CCNC'05)*, 2005.

[20] James Scott and Boris Dragovic. Audio location: accurate low-cost location sensing. In *Proceedings of The* 3^{rd} *International Conference on Pervasive Computing*, pages 1–18, 2005.