CARLOS III UNIVERSITY OF MADRID

SUPERIOR POLYTECHNIC

BACHELOR IN AUDIOVISUAL SYSTEMS ENGINEERING

FINAL YEAR PROJECT

Simulation tool implementing centralized and distributed algorithms for tracking acoustic targets

Author: **Álvaro de la Serna Martínez**
Supervisor: **Dr. Jerónimo Arenas García**

Madrid, 2014

Acknowledgements

This document is the conclusion of a year-long process of research and work. I would like to express my sincere gratitude to my Project supervisor, Dr. Jerónimo Arenas, for the continuous support of my research, for his patience, motivation and knowledge. His guidance has helped me throughout the time of research and writing of this document. I could not have imagined having a better advisor and manager for my Final Year Project and I hope he is satisfied with my work.

I would also like to thank my family, specially my parents, for their constant support throughout my life and for motivating me during these tough University years. This document is dedicated to them.

Last but not least, I would like to thank Ana González-Lahore, for her patience at all times and for always bringing a smile to my face.

To my parents

Universidad Carlos III de Madrid

Abstract

The goal of this document is the implementation of a software tool for the simulation of the acoustic tracking problem over a wireless sensor network working in a centralized or distributed manner. Its Graphical User Interface (GUI) allows the user to configure the parameters associated to the diffusion adaptive algorithms implemented in the simulation tool, in order to offer a visual representation of the behavior of a real sensor network working with those settings. For illustration we ran several simulations, which allowed us to visualize the performance of different network configurations. The results obtained with the implemented simulation tool show it can be very helpful to study the audio target tracking problem and ultimately for the design of sensor networks that can guarantee certain performance criteria.

Moreover, we have developed the code for the implementation of a real acoustic-tracking sensor network working in a centralized manner, using ©Libelium's Waspmote™ sensor boards as the network nodes and using ©Libelium's Meshlium-Xtreme™ as central node.

Contents

List of Figures

This document is structured in seven sections: The first one is a general introduction, in which basic concepts and terminology of the problem at hand will be explained; the second section presents and develops the acoustic source localization problem, along with its *centralized* and *distributed* solutions; the third section presents the software simulation tool and its corresponding Graphical User Interface, specially implemented for the simulation of this problem; the fourth one is a collection of simulations run with the said software tool; the fifth one presents the implementation of the diffusion algorithms over a real wireless sensor network; the sixth section presents the reached conclusions; and the last section is an estimation of the budget needed for a real implementation of this Final Year Project.

1 Introduction

This Final Year Project deals with the location and tracking of noisy targets using a wireless network of sensors and distributed processing. In this chapter the motivations and contributions of the Final Year Project are discussed, which essentially are the following:

- Implementation of a simulation tool for the comparison of the centralized and distributed solutions to the above problem.
- Implementation of the diffusion algorithms over a real wireless sensor network.

Prior to that discussion, we will present some basic concepts in order to give the reader some background in the matter that will help fully understand the remaining sections of the Project. These concepts span different fields, such as *Signal Processing*, *adaptive filters*, some basic *Acoustics* concepts, as well as an introduction to the *acoustic source localization* problem and the differences between *centralized* and *distributed* networks.

1.1 General introduction to Signal Processing

Human beings interact with the world around them by *reacting* to stimuli. These stimuli are captured by the senses and processed by the brain. The human brain is a very powerful tool, which stores, analyzes, processes and classifies information and, at the same time, is constantly learning and *adapting* the way it processes that information relying on previous results. By doing so, humans are capable of making decisions based on (for example) the risks and costs associated to its execution. In the particular case of the sense of hearing, which is nothing else than a way of processing information, humans can know their relative position to the source, in a way that they can know if someone is speaking close to them or if the speaker is far away, if the speaker is behind or in front of them, etc. There are many more examples where the human brain currently surpasses the abilities of computers of drawing conclusions and making decisions in real-time situations, e.g. knowing how many people there are in a room with a simple visual exploration or, in the case of the sense of hearing, recognizing sound patterns and identifying someone/something by the sound it makes. A good example would be what is known as the "Cocktail party effect", which basically means to be able to focus one's auditory attention on a particular stimulus while filtering out a range of other stimuli, the same way that someone at a party can focus on a single conversation in a noisy room [1].

Nevertheless, there are cases in which we need computers to analyze certain phenomena in a more efficient way than our brain, performing estimation and decision tasks. These scenarios emerge when the considered variables are not easy to understand, when our senses are incapable of determining the stimuli (e.g. radio signals), when the required degree of precision surpasses human possibilities, when the amount of information to be processed exceeds the human capacity, or simply when the working environment is harmful for humans (e.g. measuring the pressure of a boiler or the sound power of a plane turbine). It is in these type of cases when **Signal Processing** becomes a very powerful (and useful) tool.

Telecommunication systems are a perfect example of the robustness of artificial computing versus a human brain. In these, the analysis of the information carried by the signals would be impossible for a human brain. Considering not only the rate at which the information is transmitted and processed but also all the effects the signals might have suffered during transmission (attenuation, dispersion, diffraction, etc.), one can imagine how difficult it would be for a human to process all that data, while artificial processors can work at extremely high rates using processing tools, e.g. filters, estimators, decision makers, etc., that use mathematical logic and precise operations to perform the required tasks, along with robust algorithms to deal with data loss problems (or distortion) on transmission.

Everything said up to this point applies to Signal Processing in general. Depending on the nature of the data at hand, one can work using *Analog* or *Digital*

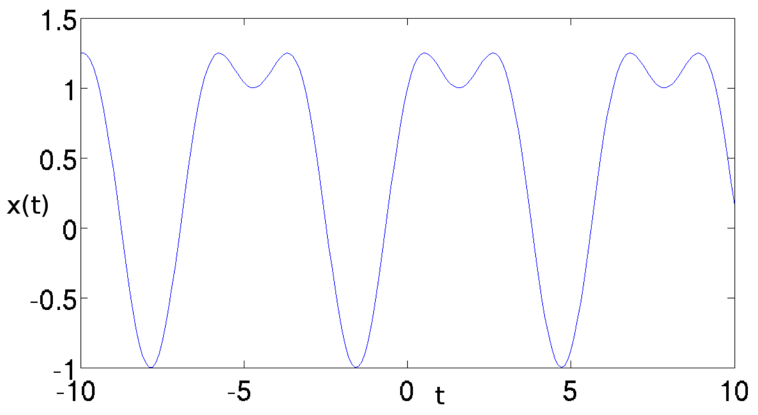


Figure 1: Graphical representation of a continuous-time signal

Signal Processing.

1.1.1 Introduction to Analog Signal Processing

Analog Signal Processing (ASP) is any signal processing conducted on analog signals by analog means. The term *analog* indicates that a magnitude is mathematically represented as a set of continuous values, typically representing the voltage, electric current, or electric charge around components in the electronic devices (e.g. crossover filters in loudspeakers). Common analog processing elements include capacitors, resistors, inductors and transistors. Analog signals are often depicted graphically as shown in Figure 1.

An analog signal is often described by a function of time, say, $x(t)$. In many cases, it is easier to analyse the behaviour of the signal by studying its *frequency response*, i.e. $X(f)$, which is the *Fourier Transform* of $x(t)$. This type of analysis is based on the *impulse response* of the signal when it is passing through the system. While any signal can be used in ASP, there are many types of signals that are used very frequently [2]:

- **Sinusoids:** Sinusoids are the building block of analog signal processing. All real world signals can be represented as an infinite sum of sinusoidal functions using Fourier Transform.
- **Impulse:** An impulse (*Dirac delta* function, see Figure 2) is defined as a signal

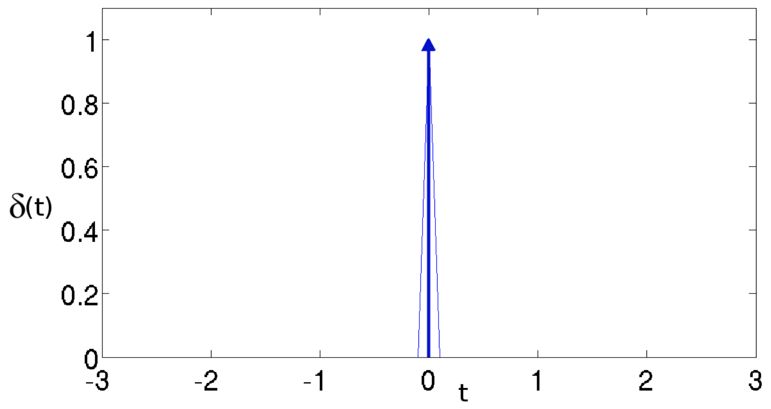


Figure 2: Graphical representation of the Dirac delta function

that has an infinite magnitude and an infinitesimally narrow width with a unit area under it. It is not possible in reality to generate such a signal, but it can be sufficiently approximated with a large amplitude, narrow pulse, to produce the theoretical impulse response in a network to a high degree of accuracy. The impulse response is said to *define* the system because all possible frequencies are represented in the input.

- **Step:** A unit step function, also called the *Heaviside* step function, is a signal that takes value zero before zero and a value of one after this point. The step response (i.e. output of the system if the input is the step function) shows how a system responds to a sudden input, similar to turning on a switch.

By *sampling* an analog signal, i.e. periodically measuring the value of the analog signal every T seconds, a *digital* signal can be obtained. For an accurate representation of a signal $x(t)$ by its time samples $x(nT)$, two conditions must be met:

1. The signal $x(t)$ must be *bandlimited*, that is, its frequency spectrum must be limited to contain frequencies up to some maximum frequency, say f_{max}, and no frequencies beyond that (see Figure 3).

2. The sampling rate f_s must be chosen to be at least **twice** the maximum frequency f_{max}, i.e.

$$f_s \geq 2f_{max} \tag{1}$$

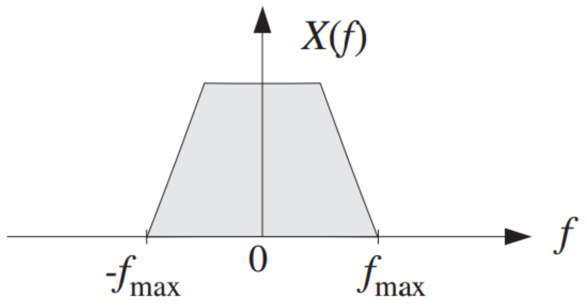


Figure 3: Representation of the spectrum of a bandlimited signal

The *minimum* sampling rate, $f_s = 2f_{max}$, is called the *Nyquist rate*, and the value $\frac{f_s}{2}$ is called the *Nyquist frequency*, which also defines the *cut-off frequencies* of the low pass analog filters that are required in *Digital Signal Processing* operations [3].

Working at a sampling rate equal to the *Nyquist rate* guarantees that the frequency replicas do not overlap and therefore the sampled continuous signal can be recovered.

1.1.2 Introduction to Digital Signal Processing

The term *digital* refers to discrete (discontinuous) values. We can retrieve discrete values from an analog (continuous) signal by *sampling* it, which translates into a reduction of a continuous signal by taking values from it only at certain moments $t = nT$, with $n = 1, 2..., N$; being $T \leq \frac{1}{f_{max}} = \frac{2}{f_s}$ the sampling period associated to the *Nyquist frequency* of the signal.

Discrete-time signals can be obtained by sampling a continuous-time signal, or they may be generated directly by some discrete-time process. Whatever the origin of the discrete-time signals, discrete-time processing systems offer great flexibility of implementation and can be used to simulate analog systems or, more importantly, to realize signal transformations that cannot be implemented with continuous-time hardware. Thus, discrete-time representations of signals are often desirable when sophisticated and flexible signal processing is required [2].

Discrete-time signals are represented mathematically as sequences of numbers.

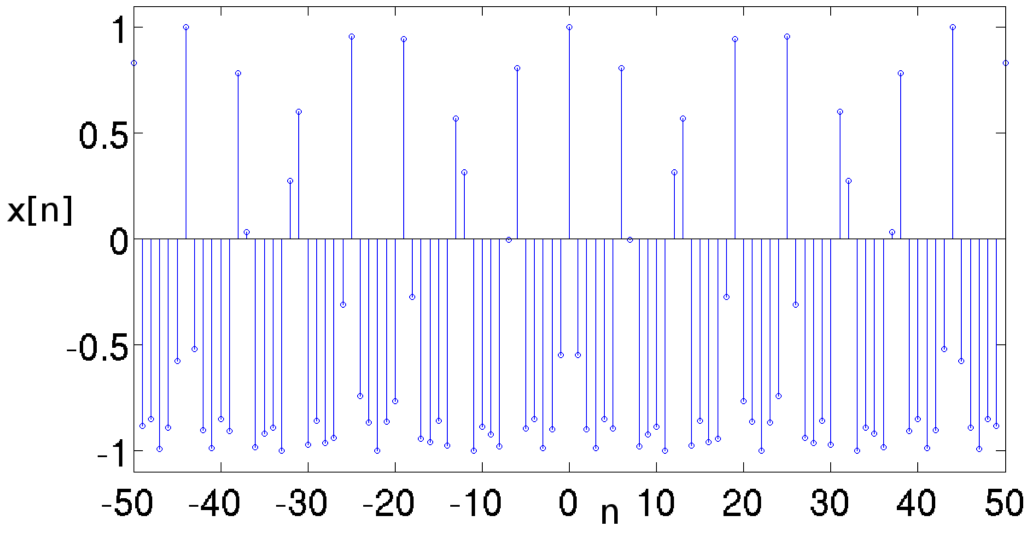


Figure 4: Graphical representation of a discrete-time signal

A sequence of numbers x, in which the nth number in the sequence is denoted $x[n]$ is formally written as:

$$x = \{x[n]\} \tag{2}$$

where $n \in (-\infty, \infty)$ is an integer. These sequences often come from the sampling method mentioned in the previous point, and they are often depicted graphically as shown in Figure 4.

There are multiple ways of manipulating a sequence, some of the simplest are *scaling*, i.e. multiplying each element of the signal by a complex number α, and *delay*. A sequence $y[n]$ is said to be a delayed version of sequence $x[n]$ if

$$y[n] = x[n - n_0] \tag{3}$$

where n_0 is an integer.

As in the analog case, the main sequences used in Digital Signal Processing are *sinusoids*, the *unit sample sequence* and the *unit step sequence*:

- **Sinusoids:** as in the analog case, any signal can be represented as an infinite sum of sinusoidal functions using Fourier Transform.

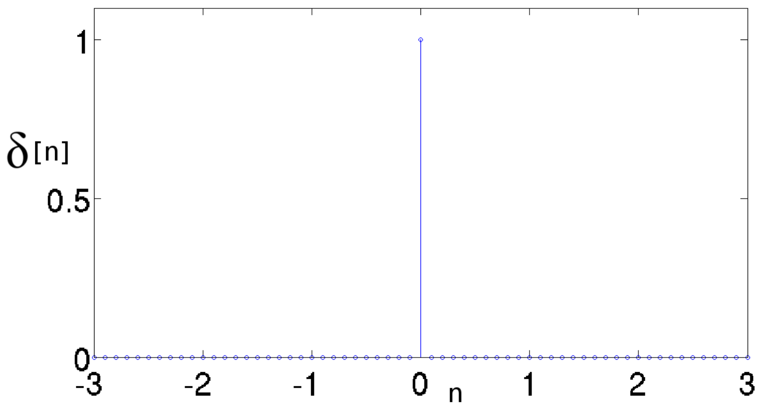


Figure 5: Unit sample sequence

- **Unit sample sequence:** it is the basic sequence in discussing the theory of discrete-time signals and systems (see Figure 5), which responds to the following expression:

$$\delta[n] = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

- **Unit step sequence:** it is a signal that has a magnitude of zero before zero and a magnitude of one after zero (see Figure 6), and is given by:

$$u[n] = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

It is easy to deduce that the unit step is related to the impulse (unit sample) by:

$$u[n] = \sum_{k=0}^{\infty} \delta[n-k] \tag{6}$$

that is, the unit step sequence can be related to the impulse in terms of a sum of delayed impulses.

With this in mind, any arbitrary sequence $x[n]$ can be represented as a sum of *scaled*, *delayed* impulses. In the general case:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \tag{7}$$

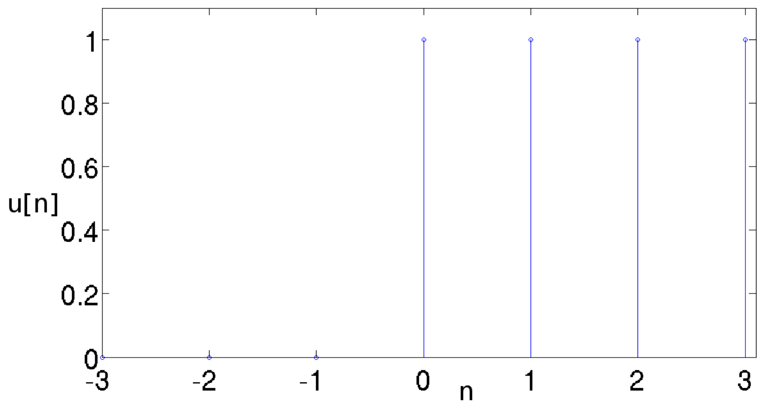


Figure 6: Unit step sequence

Throughout the Project we will work with discrete signals of acoustic origin, i.e. obtained from sampling the atmospheric pressure level. The objective will be to estimate the trajectory of a noisy target, which will require the use of more advanced signal processing than just using scaling or delay. Specifically, the signals will feed a type FIR (Finite Impulse Response) signal filter, with impulse response $p[n]$. Given that the design is performed with the objective of minimizing an error signal and that the response of the filter may vary in time, we will resort to adaptive filtering algorithms, which are briefly presented in the next section.

1.2 Introduction to adaptive filters

An adaptive filter is a computational tool that attempts to model the relationship between two signals in real time and in an iterative manner [4]. Compared to classical digital filters, working with adaptive filters presents the following advantages:

- They can complete some signal processing tasks that traditional digital filters cannot, e.g. one can use adaptive filters to remove time-varying noise from a signal.
- They can complete real-time or online modelling tasks. Typically, adaptive filters are useful when one performs real-time or online signal processing applications where the desired filter response varies over time, e.g. the acoustic source localization and tracking problem.

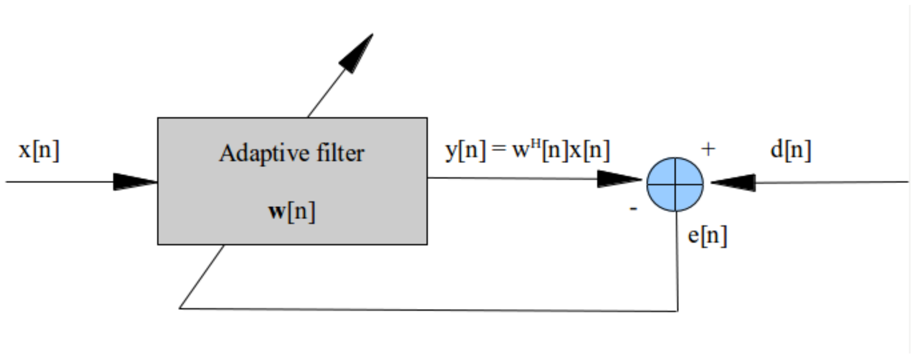


Figure 7: Schematic of an adaptive filter

Adaptive filtering with a FIR structure is carried out in the following way (see Figure 7): An input signal $x[n]$ is fed into a system which computes the output signal $y[n]$ at time n:

$$y[n] = \mathbf{w}^H[n]\mathbf{x}[n] \tag{8}$$

where the H superscript denotes conjugate transposition and $\mathbf{w}[n]$ represents the vector of weights of the filter, which control the behavior of the filter, at sample time n. Finally, $\mathbf{x}[n]$ represents a vector of *samples* of the signal:

$$\mathbf{x}[n] = [x[n], x[n-1], ..., x[n-N+1]] \tag{9}$$

where N represents the length of the vector of weights of the filter.

This output signal is then compared to $d[n]$, the *desired response signal*, by calculating the difference between them at time n:

$$e[n] = d[n] - y[n] \tag{10}$$

which is called the *error signal*. This error signal is then fed into the system in a way that modifies or *adapts* the parameters of the filter from time n to time $n+1$ in a specific manner. The weights of the filter can be learnt and adapted using different algorithms, e.g. the *least-mean-squares* or the *recursive least squares* algorithms, among others [4]. In this Project we will just pay attention to the Least Mean Square Algorithm, that we describe next.

The Least Mean Square Algorithm

Least-mean-squares (LMS from now on) algorithms are a class of adaptive filters used to simulate a certain filter by finding the filter coefficients, $\mathbf{w}$, that *minimize* the least-mean-squares of the error signal (10), i.e. they minimize the mean of the quadratic error:

$$J[n] = \mathbb{E}\{e^2[n]\} \tag{11}$$

where $J[n]$ denotes the cost associated to the estimation error.

The LMS algorithm for a filter of order N (N being the number of weights of the filter) can be performed in the following way: we want to obtain the filter coefficients $\hat{\mathbf{w}}[n]$ that minimize the error function (11). This is carried out updating the weights of the filter using *steepest descent*.
Applying steepest descent means to take the partial derivatives of (11) with respect to the individual entries of the filter coefficient (weight) vector:

$$\nabla_{\hat{\mathbf{w}}}\mathbb{E}\{e^*[n]e[n]\} = 2\mathbb{E}\{\nabla_{\hat{\mathbf{w}}}\left(e[n]\right)e^*[n]\} = -2\mathbb{E}\{e^*[n]e[n]\} \tag{12}$$

where ∇ denotes the gradient operator. Now, the gradient of $e^2[n]$ is a vector which points towards the steepest ascent of the squared error. In order to find its minimum we need to take a step in the opposite direction of the gradient, which results in the *steepest descent* algorithm:

$$\hat{\mathbf{w}}[n+1] = \hat{\mathbf{w}}[n] - \frac{\mu}{2}\nabla_{\hat{\mathbf{w}}}\mathbb{E}\{e^*[n]e[n]\} = \hat{\mathbf{w}} + \mu\mathbb{E}\{e^*[n]e[n]\} \tag{13}$$

Finally, the LMS is obtained as an stochastic approximation of the rule above, using an instantaneous estimation of the expectation. Thus, approximating the expectation $\mathbb{E}\{x[n]e^*[n]\} \approx x[n]e^*[n]$, the LMS update rule is obtained:

$$\hat{\mathbf{w}}[n+1] = \hat{\mathbf{w}}[n] + \mu e^*[n]\mathbf{x}[n] \tag{14}$$

Using this notation, the boldface variables correspond to vectors, the asterisk denotes conjugation and μ is the step size or *adaptation constant*, which represents the *speed* at which the algorithm converges. In this sense, the higher the value of μ, the faster the algorithm converges, but also the higher the residual error becomes. One has to choose μ correctly for the algorithm to be convergent, otherwise the algorithm would not converge fast enough or would not converge at all (if μ is set too high the algorithm may diverge).

The strength of this algorithm relies on its simplicity: it does not require estimations of the correlation functions, or keeping a lot of variables in memory, as other algorithms do.

1.3 Introduction to Acoustics

Acoustics is the science that deals with the study of all mechanical waves in gases, liquids and solids including vibration, sound, ultrasound and infrasound [5]. Its application is present in almost all aspects of modern society, being noise control industries the most obvious one.

If we consider its original definition, acoustics is simply the study of small pressure waves in air which can be detected by the human ear (*sound*) and it is a part of *fluid dynamics*. Fluid dynamics present a major problem: motion equations are non-linear. This implies that an exact general solution of these equations is not available. Acoustics is a first order approximation in which non-linear effects are neglected. In classical acoustics the generation of sound is considered to be a boundary condition problem (i.e. the sound generated by a loudspeaker)[5].

The study of acoustics revolves around the generation, propagation and reception of mechanical waves and vibrations. For the generation and reception parts of the process, which are of special importance, one needs to dive into the *electroacoustics* field, which is a branch of acoustical engineering that deals with the design of headphones, microphones, loudspeakers, sound systems, sound reproducing and recording; but the most important part of the process relies in the propagation of the sound wave.

1.3.1 Wave propagation: pressure levels

In fluids such as air and water, sound waves propagate as disturbances in the ambient pressure level. While this disturbance is usually small, it is still noticeable to the human ear. The smallest sound that a person can hear, known as the threshold of hearing, is nine orders of magnitude smaller than the ambient pressure [5]. This great variance in the sound levels a human ear can detect means that human hearing does not have a flat *spectral sensitivity* (frequency response) relative to frequency versus amplitude. Instead, human ears present a *logarithmic* frequency response. Humans do not perceive low- and high-frequency sounds as well as they perceive sounds near 2,000 Hz, as shown in the equal-loudness contour (see Figure 8). An equal-loudness contour is a measure of sound pressure (dB SPL), over the frequency spectrum, for which a listener perceives a constant loudness when presented with pure steady tones. The unit of measurement for loudness levels is the *phon*, and is arrived at by reference to equal-loudness contours. By definition, two sine waves of differing frequencies are

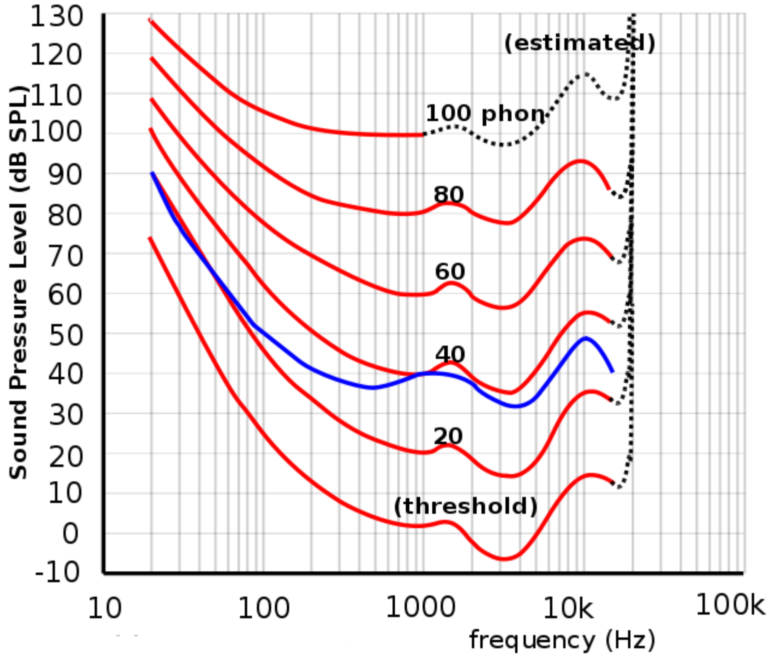


Figure 8: Equal-loudness contours (in red) (from ISO 226:2003 revision). It represents the sound pressures for which a listener perceives a constant loudness when presented with pure steady tones. The original ISO standard for humans is represented in blue (≈40 phons)

said to have equal-loudness level measured in phons if they are perceived as equally loud by the average young person without significant hearing impairment [6, p.276].

The loudness of the disturbances detected by the human ear is called the **sound pressure level** (SPL) and is measured on a logarithmic scale in decibels, dB, following Equation (15) below. Technically, the sound pressure level (or sound level) L_p is a measure of the *effective* sound pressure of a sound relative to a reference value.

The effective value of the sound pressure would be p_{rms}, the reference sound pressure value would be $p_{ref} = 20\ \mu\text{Pa}$ (rms), which corresponds to the threshold of hearing at 1 kHz for a typical human ear [5], so that the SPL can be calculated as follows:

$$L_p = 10 \cdot log_{10}\left(\frac{p_{rms}^2}{p_{ref}^2}\right) = 20 \cdot log_{10}\left(\frac{p_{rms}}{p_{ref}}\right) dB \tag{15}$$

The *rms* acronym stands for *root mean square* and it is the square root of the arithmetic mean (average) of the squares of the original values:

$$x_{rms} = \sqrt{\frac{1}{N} \sum_{n=n'}^{n'+N-1} x^2[n]} \tag{16}$$

where $n = n', n'+1, ..., n'+N-1$ is the sampling window size, i.e., the number of samples we are working with. Note that we are working with discrete values, otherwise this expression would respond to the integral over the sampling time interval of the measured signal squared.

When working with a sound pressure level, the *distance* between the emitter and the receptor is a crucial variable, because sound pressure levels are *inversely* proportional to distance. This means that the value of the SPL suffers an attenuation proportional to the distance travelled by the sound wave. The measured SPL can be calculated using the following equation with $r_1 = 1$ m, which is the standard reference distance:

$$L_{p_2} = L_{p_1} + 20 \cdot log_{10}\left(\frac{r_1}{r_2}\right) dB \tag{17}$$

where:

- L_{p_2} is the measured SPL at distance r_2.
- L_{p_1} is the reference SPL value, measured at distance r_1.
- r_1 is the reference distance, that is, the distance at which we can ensure the source emits a SPL value equal to L_{p_1}.
- r_2 is the distance between the source and the receiver.

Note that the SPL suffers an attenuation proportional to the square of the travelled distance r_2.

This SPL propagation model will be essential for some of the acoustic source tracking algorithms that we will describe and implement in this Project.

1.3.2 Wave frequency

In the field of engineering, sound pressure levels are frequently analyzed as a function of frequency, partly because this is how our ears interpret sound. What we experience as *higher pitched* or *lower pitched* sounds are pressure vibrations having a higher or lower number of cycles per second (*frequency*). Normally, acoustic signals are sampled in time and then presented in more meaningful forms such as octave bands or time frequency plots. Both these methods are used to analyze sound and better understand the acoustic phenomenon. Analytic instruments such as the spectrum analyzer facilitate visualization and measurement of acoustic signals and their properties.

The entire frequency spectrum can be divided into three sections: audio, ultrasonic, and infrasonic. The *audio* range falls between 20 Hz and 20,000 Hz. This range is very important because its frequencies can be detected by the human ear. This range has a number of applications, including speech communication and music. Moreover, it is the frequency range we are going to work with in the acoustic source localization problem.

The *ultrasonic* range refers to the very high frequencies: 20,000 Hz and higher. This range has shorter wavelengths which allow better resolution in imaging technologies. Medical applications such as ultrasonography and elastography rely on the ultrasonic frequency range.

On the other end of the spectrum, the lowest frequencies are known as the *infrasonic* range. These frequencies can be used to study geological phenomena such as earthquakes.

Since the goal of this Project is to implement an acoustic-based localization algorithm we will be working with acoustic waveforms whose frequencies fall in the *audio* spectrum.

We will be working with noise sensors, which are a type of transducer that convert sound pressure waves into electrical signals and those electrical signals into digital values, which will be the input data to the algorithms.

1.4 Introduction to Acoustic Source Localization

Sound localization is the task of determining the *direction* and *distance* to a source with the only help of the sounds it makes. Getting these two parameters allows an accurate localization of a noisy source, e.g. a vehicle in motion or a person speaking, which is crucial for a certain number of applications [7].

Despite comprising several methods which have been studied for several years, acoustic localization systems are very difficult to find outside research papers and military applications. This offers endless possibilities for researchers, who can add acoustic-source localization algorithms to *image-based* systems, e.g. video tracking and image processing, to study the improvement of the system in terms of localization efficiency. This acoustic addition to tracking systems makes sense in scenarios in which visual localization presents limitations in angle, reach and/or brightness of the image. The acoustic-visual combination would be more efficient in the way that the acoustic localization system can "alert" the visual localization system of the direction from which the target is coming, making its tracking easier and faster. This example illustrates how each system complements the weaknesses of the other: *visual* tracking helps in the determination of the height at which the source is moving, since 3-D estimation in the acoustic system is not necessary (if we know the direction from which a sound is coming, we cannot really know how its position varies in altitude unless we are able to see it); and *acoustic* tracking helps in the determination of the direction from which the source is approaching in the cases the visual system, i.e. a camera, is not pointing in the direction of the source, thus not recognizing it.

There have been several approaches to the solution of this problem, which one could group into two big families: Solutions based the calculation of the *Time Delay of Arrival* (TDOA) and solutions based on the measured signal pressure level, inversely proportional to the square of the distance (see Equation (17)).

The first family of solutions has been widely used for the following reasons [8]:

- Such systems are conceptually simple.
- They are reasonably effective in moderately reverberant environments.
- Their low computational complexity makes them well-suited to real-time implementation with several sensors.

Some authors propose a two-microphone implementation which tries to simulate the way our ears locate the sound source. Others prefer a three or more microphones array arranged in different ways. In all of them, the *modus operandi* consists in first estimating the delay between the signals received in every sensor (microphone) and afterwards calculating the *Direction of Arrival* (DOA) of the sound. These models can present errors in their estimation, mostly due to phase errors in the retrieval of information and synchronization errors [7, 8, 9, 10]. It is important to note that all the solutions to the acoustic source localization problem present good behaviour in two dimensions. There is no need for a three-dimensional solution because one cannot exactly know how the position of the source varies in altitude without visual aid. Nevertheless, an N-dimensional solution to this problem can be found in [11] for outdoor sound localization applications.

With respect to the second family, some approaches [12, 13] rely on the robustness of a sensor network working either in a *centralized* or *distributed* manner, which is the approach taken for study in this Project. In these, every element of the network (*node* from now on) has a built-in noise sensor. By taking samples of the sound signal, it makes an estimation of the position of the source at time instant n, shares it with its *neighbours* and, using its neighbors' estimations, adapts its result and calculations to estimate the final position. A node is said to be a neighbor of another node inside a network if they are separated a certain distance, smaller than a given threshold, thus making their communication possible.

In the case of the *centralized* solution (see Figure 9), a central node is responsible of retrieving the data measured by all the nodes in the network and combining them into a final estimate. In the case of the acoustic source localization problem, each node in the network takes a sample measurement of the SPL at time instant n, makes a first estimate of the position of the source, makes an estimation of the possible estimation errors and sends both values along the network until it reaches the central node, which will be responsible for retrieving the estimations of all the nodes in the network and perform the necessary calculations to give the final estimation of the position of the source at time n.

In the *distributed* scenario (see Figure 10), each node takes a sample measurement at time instant n, makes a first estimate of the position of the source and makes an estimation of the possible estimation errors. Then it exchanges those estimations with its neighbors in order to obtain a *joint* estimation of the position at time n, thus resulting in as many estimated trajectories as nodes are in the network.

Diffusion methods endow networks with powerful adaptation abilities that en-

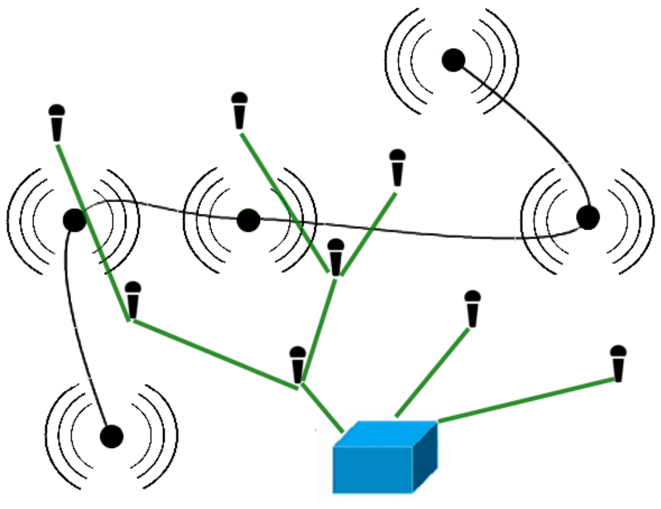


Figure 9: Scheme of a centralized sensor network. In this scenario, all sensors (microphones) will send their measured SPL value at time instant n to the central node (blue box), which will be responsible of performing the estimation of the position of the source at time n.

able the individual nodes to continue learning even when the cost function changes with time (the cost function is still MSE (11) but since our acoustic target is in motion, its position, and therefore the cost function, changes with time). In the diffusion approach, information is processed locally and *simultaneously* at all nodes and the processed data are diffused through a real-time sharing mechanism that ripples through the network continuously [13].

Figures 9 and 10 show an example of a centralized and a distributed sensor network respectively, in which a noisy source is moving along the depicted path. These are the two main scenarios to be studied and compared along the following sections. A diffusion sensor-network-based solution to the acoustic source localization problem presents several advantages and disadvantages, which will be discussed in the next sections.

1.4.1 Pros & Cons of *Centralized* networks

A centralized system is one in which a central controller exercises control over the lower-level components of the system directly or through the use of a power

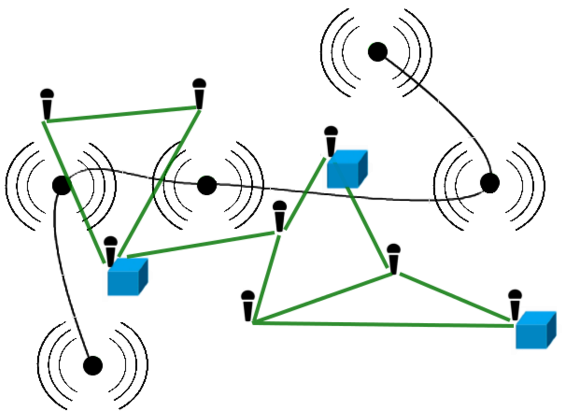


Figure 10: Scheme of a distributed sensor network. In this scenario, there are some nodes in the network capable of performing the necessary estimations (computational nodes, in blue), which will receive the measured SPL values sent by its neighbors (sensors closer to them than a certain distance) at time n. In the distributed configuration we will end up having as many position estimations as computational nodes exist in the network. Finally we can obtain a *joint* estimation by combining all possible estimations.

hierarchy (such as instructing a middle level component to instruct a lower level component). The complex behaviour exhibited by this system is thus the result of the central controller's "control" over lower level components in the system, including the active supervision of the lower level components.

The term *centralized* is used for diffusion networks that rely on a single (*central*) node of the network to perform the computations. This architecture model presents several advantages, like higher computational speed, easy maintenance (simpler programming code, which allows to easily track each node and locate all possible errors along the network) and provides greater control to the person managing the network.

However, the drawback of a fully centralized control system is that everything must be routed back to a single control device. This *rendezvous point* could cause slower communications (large networks can cause congestion at the central node because all nodes are trying to communicate with it at the same time), and is a potential single point of failure for the network. Moreover, the routing must be automatic. If a network node crashes the whole network has to be re-routed. This translates into a lack of flexibility when changes in the network need to be made [17].

1.4.2 Pros & Cons of *Distributed* networks

A distributed (or *decentralized*) system is a system in which lower level components operate on local information to accomplish global goals. In other words, a distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. This definition leads to the following especially significant characteristics of distributed systems: concurrency of components, lack of a global clock and independent failures of components [18].

In contrast to centralized control, networks that distribute its intelligence and computation resources among the constituent nodes allow local devices to make decisions on their own, without communicating with a "master" device. This speeds response times and eliminates the single point of failure. Moreover, *distributed* networks offer a slightly slower computational speed (since the nodes have to be constantly communicating with each other and have to process the information obtained by their neighbors) but they are more robust against node failures. In the distributed scenario, the network does not stop working if a single node crashes, but if the number of nodes in the network is too large, this kind of network architecture can get saturated if the communications are not programmed correctly (some nodes might end up receiving larger amount of data than others, resulting in local bottlenecks). Moreover, because all decisions are made at the nodes themselves, the (human) manager loses control and visibility into the network. Worse, intelligent devices capable of making their own decisions require more processing power, and greater upkeep, in order to handle the additional costs in gathering, maintaining and updating the data. So the cost of a fully distributed network is usually greater than that of a fully centralized network [17].

Considering the paragraphs above, one could summarize the advantages of distributed systems over centralized ones in a short list:

1. *Incremental growth*: Computing power can be added in small increments.
2. *Reliability*: If a single node crashes, the system as a whole can still survive.
3. *Speed*: A distributed system can have more total computing power than a centralized one. For example, imagine we have 10.000 CPU chips, each running at 50 MIPS (Million Instructions Per Second). It is not possible to build a 500.000 MIPS single processor since it would require 0.002 nanosecond instruction cycles. Therefore we obtain enhanced performance through load distributing.

On the other hand, choosing distributed systems over centralized ones can present the following disadvantages:

1. *Network security*: Network managers have to be careful with the encryption of the channel used in the communication to prevent errors and to strengthen them against the threat of being exposed by hackers, because in a distributed system information is transmitted using *messages* which carry the information instead of just sharing the measured data as in the centralized case.

2. *Networking*: If the network gets saturated then problems with transmission will surface.

3. *Troubleshooting*: Troubleshooting and diagnosing problems in a distributed system can become very difficult, because the analysis may require connecting to remote nodes or inspecting communications between nodes.

1.5 Contributions of the Project

In the previous section we have described how centralized and distributed networks work and we have highlighted their respective strengths and weaknesses. The motivation of the Project is to develop a software simulation tool that simulates a sensor network working on an acoustic target tracking problem either in a centralized or distributed way, which allows to compare the results obtained by diffusion networks and to compare them to the optimal theoretical result of the centralized solution. With it, we want to study the different behaviours of the system depending on the number of nodes in the network and depending on the relative distances between the nodes and the trajectory of the acoustic source, this last one to determine if the model adjusts to "reality", in the sense that noisy sources that are far from the sensors (human ears if the sensor were a person) are harder to locate.

Moreover, when working with distributed networks, we will test how the coverage range of the nodes affects the accuracy of the algorithm. It will be shown that increasing the coverage range of the nodes leads to more nodes in the network becoming neighbors and, consequently, the accuracy of the algorithm increases drastically.

In order to compare the performance of both types of solutions we have developed a MATLAB®-based software for their simulation and comparison, in which the network configuration parameters can be set using a graphical user interface (GUI). The software allows to manage some simulation parameters, such as:

- The trajectory of the noisy source, i.e. the one that the network will estimate.
- The size and topology of the network.
- Some parameters of the diffusion algorithms, e.g. the step size.

Such software has been validated by a number of experiments in which the behavior of both solutions is analized.

Moreover, using hardware resources of the research group, a software implementation of a diffusion-based distributed strategy has been developed for ©Libelium's wireless sensor motes.

2 Acoustic target tracking

In this chapter we present the algorithms for target localization and tracking over sensor networks. We start by introducing the problem, notation and cost function. Then, we derive algorithms for on-line tracking using distributed and centralized processing in sensor networks. These algorithms are based on measurements of the acoustic pressure and time delay at each node, the latter normally not being available in this application. For completeness, we conclude the chapter by presenting some alternatives for estimating such time delay.

2.1 Problem statement

We consider a network of wireless nodes with fixed positions, $\mathbf{s}_k$, over a coverage area (see Figure 11). Each node uses one omnidirectional noise sensor (i.e., a noise sensor that can "hear" noise from virtually any direction) to obtain the sound pressure level (*SPL*) and an antenna to communicate with other nodes in the network.

We consider a time-varying trajectory with coordinates $x[n]$, n being the iteration number (associated with discrete time). The target emits with a constant energy over this trajectory, and it is therefore characterized by a constant, L_{p_s}, which is the SPL measured at 1 meter. The objective is to estimate the position, $\mathbf{w} = [x, y]^T$,of the target with a sensor network either in a centralized or distributed manner. The idea is that each network node k has a microphone that measures the acoustic pressure, L_{p_k}, and has access to the propagation time required for the acoustic wave to get from the source to node k, $t_k[n]$.

The value of the SPL received by every noise sensor depends on its *directivity pattern* and on the *relative distances* of every node to each point of the trajectory of the acoustic source. In section 1.3.1 we presented the equation that returns the value of the SPL of the acoustic waveform after it has travelled a given distance (see Equation 17). In addition, acoustic waves are very sensitive to noise. Here, the term *noise* refers to any disturbance in the original signal, e.g. echoes, obstacles in the path, additional sounds, etc. In order to simulate information losses due to attenuation, diffusion, noise and presence of obstacles [12], some corrections have to be added to (17), so the measured SPL value at node k at time instant n, $L_{p_k}[n]$, can

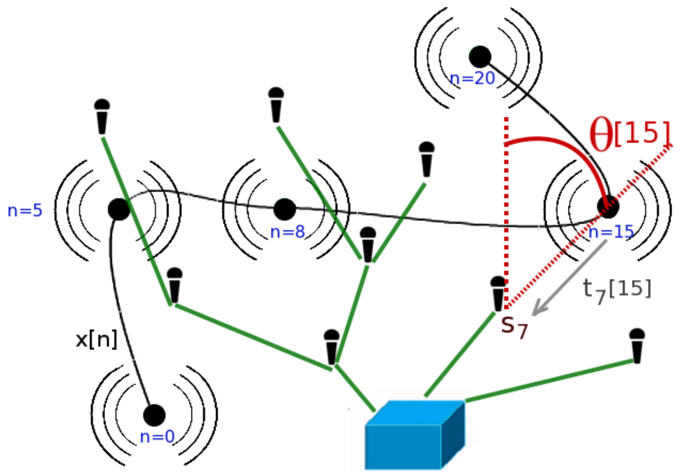


Figure 11: Relative position between the noisy source and a certain sensor at time instant n=15 in a centralized scenario. Here, $t_7[15]$ and $\theta[15]$ denote the signal propagation time from the source to node s_7 and the direction of arrival of the signal to node s_7, respectively, at time $n = 15$.

be expressed as:

$$L_{p_k}[n] = L_{p_s} + g_k\left(\theta[n]\right) + 10 \cdot log\left(\frac{d_0}{d_k[n]}\right)^{\alpha} + n_k[n] \qquad (18)$$

where:

- $L_{p_k}[n]$ is the sound pressure level measured by sensor k at time n, in dB.
- L_{p_s} is the sound pressure level of the acoustic source, in dB, measured at a distance $d_0 = 1$ m.
- $g_k\left(\theta\right)[n]$ is the gain of sensor k at time n, in dB, due to its directivity pattern. Since in this Project we assume omnidirectional microphones we will have $g_k\left(\theta\right)[n] = 0$ dB $\forall \theta, n$
- $d_k[n]$ is the Euclidean distance between node k and the acoustic source at time n.
- α is the attenuation exponent corresponding to the log-distance path loss model.

- $n_k[n]$ is a Gaussian noise at time instant n.

As previously said, we also assume that each node has access to a variable $t_k[n]$ which is associated to the propagation time of the acoustic wave between the target and the node at time instant n, i.e.:

$$t_k[n] = \frac{d_k[n]}{v} \tag{19}$$

where $d_k[n]$ is the real distance between the target and node k.

Thus, considering a network of N nodes, the objective is to find the coordinates of the source that minimize the hybrid cost function over $\mathbf{w}$ [12]:

$$J(\mathbf{w}) = \sum_{k=1}^{N} \Big((1-\eta) \cdot J_k^p(\mathbf{w}) + \eta \cdot \nu \cdot J_k^t(\mathbf{w}) \Big) \tag{20}$$

which denotes the sum of the local costs over all the nodes in the network, and where:

- $\mathbf{w} = [x, y]^T$ are the coordinates of the estimated position of the source.
- $J_k^p(\mathbf{w})$ is the local cost associated with node k related to the measured SPL.
- $J_k^t(\mathbf{w})$ is the local cost associated with node k related to time interval measurements and communications.
- $\eta \in$[0,1] represents the weights of the SPL and time interval cost terms in the localization of the source, i.e. how much importance we assign to each error term.
- Variable ν is used to make the value of $J_k^t(\mathbf{w})$ be approximately in the same numerical range as $J_k^p(\mathbf{w})$.

Considering the following grouping of terms in (18):

$$h_k[n] = L_{p_s} + g_k\left(\theta[n]\right) + 10\alpha log(d_0) \tag{21}$$

the local cost functions can be defined as [12]:

$$J_k^p(\mathbf{w}) = \mathbb{E}\left[\left| L_{p_k}[n] + 10 \cdot \alpha \cdot log\left(\parallel \mathbf{w}[n] - \mathbf{s}_k \parallel\right) - h_k[n] \right|^2 \right] \tag{22}$$

$$J_k^t(\mathbf{w}) = \mathbb{E}\left[\left| t_k[n] - \frac{\parallel \mathbf{w}[n] - \mathbf{s}_k \parallel}{v} \right|^2 \right] \tag{23}$$

where:

- $v \approx 340$ m/s is the speed of sound (we are working with acoustic waves).
- $\mathbf{w}[n]$ is the estimated coordinates of the noisy source at time instant n.
- $\mathbf{s}_k$ represents the coordinates of sensor k.
- $t_k[n]$ denotes the measured propagation time needed for the acoustic signal to reach sensor k (see Equation (19)).
- $h_k[n]$ is a variable which depends on the directivity of sensor k at time n due to the relative position between node k and the trajectory (see Figure 11) and is common to all nodes, but only when working with omnidirectional microphones [12], as we will explain shortly.

Equation (22) corresponds to the mean squared difference between the SPL value measured at node k considering the acoustic wave has travelled the corresponding distance between node k and the estimated coordinates of the target, L_{p_k}, and the original SPL emitted by the target, L_{p_s}. This cost term takes value 0 if the measured SPL value is equal to the theoretical value we would obtain after substituting the estimated distance, $\parallel \mathbf{w}[n] - \mathbf{s}_k \parallel$, into Equation (22).

In the same way, Equation (23) corresponds to the squared difference between the real propagation time of the acoustic wave from the target to node k, $t_k[n]$ and the theoretical propagation time from the estimated coordinates of the target, $\mathbf{w}[n]$, to node k, $\mathbf{s}_k$, at time instant n. This expression takes value 0 if the measured SPT value is equal to the theoretical value we would obtain substituting the estimated distance, $\parallel \mathbf{w}[n] - \mathbf{s}_k \parallel$, into Equation (23).

Simplifications

Working with directivity patterns can result in very complicated propagation models. For the purpose of this Project we will be working under the assumption that the source and the sensors are **omnidirectional**. This means that the source emits with constant level in every direction and the sensors are capable of detecting sound coming from any direction, so an important simplification can be made to (18):

- $g_k(\theta)[n] = 0 \; \forall \; k, n$, due to the *omnidirectionality* of the noise sensor, i.e. every sensor is designed to get the same SPL value independently of the direction from where the sound is coming.

This implies that the $h_k[n]$ variable becomes constant:

$$h_k[n] = L_{p_s} + 10\alpha log(d_0) \tag{24}$$

Since our reference distance $d_0 = 1$, the logarithmic term takes value 0 and:

$$h_k[n] = h = L_{p_s} \tag{25}$$

This results in a simpler expression for the local cost function associated to the measured SPL:

$$J_k^p(\mathbf{w}) = \mathbb{E}\left[\left|L_{p_k}[n] + 10 \cdot \alpha \cdot log\left(\| \mathbf{w}[n] - \mathbf{s}_k \|\right) - L_{p_s}\right|^2\right] \tag{26}$$

The calculation of the local costs associated to time interval measurements and communications (see Equation (23)) requires that the network nodes have access to the value of $t_k[n]$. This might be possible in e.g., phone localization in mobile networks [12] under the assumption that the phone and all nodes are synchronized and a timestamp is included in the transmitted packages. However, this is not realistic in an acoustic localization situation. In practice, the only possibility would be to estimate $t_k[n]$ using approaches based on TDOA measurements either sharing information between the measured waves in different nodes or implementing two (or more) microphones at each node. This has also been suggested in [12]. In this document this possibility is briefly described at the end of the chapter for completeness. However, for simplicity, in the simulations and the implemented software we will assume that $t_k[n]$ is known at each node. In order to use a more or less realistic model we generate such measurements with the theoretical model (19), to which we add a noisy term which accounts for error terms of different nature, e.g., multi path, measurement noise, etc.

$$t_k[n] = \frac{d_k[n]}{v} + n_t[n] \tag{27}$$

In order to optimize the hybrid cost (20) associated to the tracking of the acoustic source, two adaptive algorithms have been tested over the wireless sensor network: the *centralized* and the *distributed* LMS algorithms.

2.2 Centralized LMS

In the centralized least-mean-squares algorithm, every node in the network contributes to *jointly* estimate the acoustic source position at time n. In this scenario, all $L_{p_k}[n]$ and/or $t_k[n]$ measurements are sent to a central node, which is in charge of minimizing the hybrid cost (20). If a stochastic gradient scheme is used, since all the nodes work as a *whole*, the gradient of (20) can be calculated and used by the central node as [12]:

$$\nabla_w J(\mathbf{w}) = \sum_{k=1}^{N} \left((1-\eta)\, \nabla_w J_k^p + \eta\nu \nabla_w J_k^t \right) \tag{28}$$

where:

$$\nabla_w J_k^p = \frac{20\alpha}{\ln 10} \mathbb{E} \left\{ \frac{\mathbf{w} - \mathbf{s}_k}{\| \mathbf{w} - \mathbf{s}_k \|^2} e_k^p[n] \right\} \tag{29}$$

$$\nabla_w J_k^t = -\frac{2}{v} \mathbb{E} \left\{ \frac{\mathbf{w} - \mathbf{s}_k}{\| \mathbf{w} - \mathbf{s}_k \|} e_k^t[n] \right\} \tag{30}$$

with error functions:

$$e_k^p[n] = L_{p_k}[n] + 10\alpha log\left(\| \mathbf{w} - \mathbf{s}_k \| \right) - h \tag{31}$$

$$e_k^t[n] = t_k[n] - \frac{\| \mathbf{w} - \mathbf{s}_k \|}{v} \tag{32}$$

where $t_k[n]$ denotes the *signal propagation time* (SPT) from the source to node k (19) and $h = L_{p_s}$.

In [12], the *steepest descent* algorithm (see section 1.2) for the minimization of (20) takes the form:

$$\mathbf{w}_i[n] = \mathbf{w}_{i-1}[n] - \mu_{\text{cent}} \nabla_w J(\mathbf{w}_{i-1}[n]) \tag{33}$$

where $\mu_{\text{cent}} > 0$ is the step size, and $\mathbf{w}_i[n]$ is the estimate of the position of the acoustic source at iteration i.

This iterative approach may cause problems when the trajectory approaches a node in the network. That is because the gradients (29) and (30) become very large in that case. This problem can be lightened by multiplying $\nabla_w J_k^p$ by $ln\left(\frac{1}{2}\right) \cdot \| w - s_k \|^2$ and scaling $\nabla_w J_k^t$ with $v \cdot \frac{\|w-s_k\|}{2}$. Doing so, and approximating the gradient (28) with the instantaneous data at time n, we arrive to the **centralized LMS algorithm** for the tracking of an acoustic source in motion [12]:

$$\hat{\nabla}_w J_k(\mathbf{w}[n-1]) = \Big(\alpha(1-\eta)e_k^p[n] - \nu\eta e_k^t[n]\Big) \cdot (\mathbf{w}[n-1] - \mathbf{s}_k) \tag{34}$$

$$\mathbf{w}[n] = \mathbf{w}[n-1] - \mu_{\text{cent}} \sum_{k=1}^{N} \hat{\nabla}_w J_k(\mathbf{w}[n-1]) \tag{35}$$

where errors $e_k^p[n]$ and $e_k^t[n]$ are evaluated using (31) and (32) at $\mathbf{w} = \mathbf{w}[n-1]$ and where:

- α is the attenuation exponent corresponding to the log-distance path loss model (see Equation (18))
- $\eta \in [0,1]$ represents how much importance we assign to each error term.
- ν is used to make the value of $e_k^t[n]$ be approximately in the same numerical range as $e_k^p[n]$.
- μ_{cent} is the step size of the centralized LMS algorithm.

2.3 Distributed LMS

In the distributed scenario the estimation of the position of the acoustic source is performed in two steps: a local estimation at every node in the network and a merger of the estimations of each node's neighbors into a final estimation of the position. This means that in the distributed scenario the network nodes no longer exchange their $L_{p_k}[n]$ nor $t_k[n]$ measurements, but instead they use them to update a local estimate, $\boldsymbol{\psi}_{k,n}$, and it is this intermediate estimation which is interchanged among the neighbor nodes in order to obtain a final estimation of the position, $\mathbf{w}_k[n]$. As mentioned in the previous section, a node is said to be a *neighbor* of another node inside the network if they are separated a certain distance, smaller than a given threshold r. This threshold value depends on the transmission power of the node.

In [12], they point out the impossibility of the nodes to respond quickly in real-time situations if they were to use the alternating directions method of multipliers in which the global cost (20) is decoupled and written as a group of local constrained optimization problems, and they propose an alternative algorithm in which minimizing the global cost (20) can be accomplished by solving the following unconstrained local optimization problems for $k \in 1, .., N$, being N the number of nodes in the network:

$$\min_{\mathbf{w}} \left\{ \sum_{l \in N_k} c_{l,k} \left[(1-\eta) J_k^p(\mathbf{w}) + \eta \nu J_k^t(\mathbf{w}) \right] + \sum_{l \in N_k \setminus \{k\}} g_{l,k} \parallel \mathbf{w} - \boldsymbol{\psi}_l \parallel^2 \right\} \tag{36}$$

where N_k denotes the set of neighbors of node k (including k itself), $\boldsymbol{\psi}_l$ is a local variable that represents the global parameter at node l (its intermediate estimate of the position) and $N_k \setminus \{k\}$ denotes the set N_k excluding node k. In this formulation, $\{g_{l,k}\}$ are non-negative parameters and $\{c_{l,k}\}$ denote non-negative entries of a right-stochastic matrix C satisfying:

$$\begin{cases} c_{l,k} = 0 & \text{if } l \notin N_k \\ \sum c_{l,k} = 1 & \text{if } l \in N_k \end{cases} \tag{37}$$

Each $c_{l,k}$ represents a weight value that node k assigns to information arriving from its neighbor l [13].

Following the arguments in [13, pp.4-7] and performing a similar normalization of the gradient as in the centralized algorithm, [12] arrives to the following **normalized distributed LMS algorithm** for minimizing the hybrid cost (20) in a distributed and adaptive manner:

$$\boldsymbol{\psi}_k[n] = \mathbf{w}_k[n-1] - \mu_{\text{dist},k} \sum_{l \in N_k} c_{l,k} \hat{\nabla}_w J_l(\mathbf{w}_k[n-1]) \tag{38}$$

$$\mathbf{w}_k[n] = \sum_{l \in N_k} a_{l,k} \boldsymbol{\psi}_l[n] \tag{39}$$

In this algorithm, $\mu_{\text{dist},k} > 0$ is the step size at node k, $\boldsymbol{\psi}_{k,n}$ denotes an intermediate estimate at node k of the value of the position at iteration n, $\mathbf{w}_{k,n}$ is the estimation at node k of the position of the source at iteration n after merging the local estimation with the estimations received from the neighbors, and the gradient takes the form:

$$\hat{\nabla}_w J_l(\mathbf{w}_k[n-1]) = \left(\alpha(1-\eta) e_l^p[n] - \nu \eta e_l^t[n] \right) (\mathbf{w}_k[n-1] - \mathbf{s}_l) \tag{40}$$

where $e_l^p[n]$ and $e_l^t[n]$ are evaluated using (31) and (32) at $\mathbf{w} = \mathbf{w}_{k,n-1}$.

Moreover, the $\{g_{l,k}\}$ parameters are replaced with the $\{a_{l,k}\}$ coefficients [12], which are non-negative entries of a left-stochastic matrix $A \in \mathbb{R}^{NxN}$ that satisfy:

$$\begin{cases} a_{l,k} = 0 & \text{if } l \notin N_k \\ \sum a_{l,k} = 1 & \text{if } l \in N_k \end{cases} \tag{41}$$

In addition, we make parameters $a_{l,k}$ in (39) constant and equal to the inverse of the number of neighbors of node k, to make the nodes' neighbors equally relevant in the computations.

Just for completeness, these $c_{l,k}$ and $a_{l,k}$ parameters can be obtained using adaptive methods in order to find an optimal value for each node instead of taking constant values. The interested reader can consult [14, 15, 16] for a more insightful development of these methods.

As said at the beginning of this section, the distributed LMS algorithm is performed in two steps: the first one is an adaptation step in which node k updates its intermediate estimation from $\mathbf{w}_k[n-1]$ to $\boldsymbol{\psi}_k[n]$ using the measured data $L_{p_l}[n], t_l[n]$ (all other nodes in the network are performing a similar step and generating their intermediate estimate $\boldsymbol{\psi}_l[n]$) and the second one is a combination step, in which node k combines its intermediate state $\boldsymbol{\psi}_k[n]$ with those of its neighbors to obtain $\mathbf{w}_k[n]$. This is the reason this type of algorithm is referred as *Adapt-then-Combine* (ATC). Finally, the step size parameters $\mu_{\text{dist},k}$, according to [13], can assume constant values ($\mu_{\text{dist},k} = \mu_{\text{dist}}$), which is critical to endow the network with continuous adaptation and learning abilities (otherwise, when step sizes die out, the network stops learning). Actually, constant step sizes also empower networks with tracking abilities, in which case the algorithms can track time changes in the optimal estimated trajectory [13].

For simplification we will consider the $c_{l,k}$ terms in (38) to be:

$$\begin{cases} c_{l,k} = 1 & \text{if } l = k \\ c_{l,k} = 0 & \text{otherwise} \end{cases} \tag{42}$$

In the following sections of this Project, Equations (38) and (39) will be referred as **diffusion ATC** algorithm instead of *distributed LMS* algorithm. Considering the paragraphs above and denoting the number of neighbors of a node as n_k, the diffusion ATC algorithm results in the following equations:

$$\boldsymbol{\psi}_k[n] = \mathbf{w}_k[n-1] - \mu_{\text{dist}} \hat{\nabla}_w J_k(\mathbf{w}_k[n-1]) \tag{43}$$

$$\mathbf{w}_k[n] = \frac{1}{n_k} \sum_{l \in N_k} \boldsymbol{\psi}_l[n] \tag{44}$$

2.4 TDOA techniques

In previous sections we assumed that $t_k[n]$ measurements are known at each node. In practice, this is rarely the case and we can proceed by:

a) Ignoring the signal propagation time (SPT) term in the cost function (i.e., setting $\eta = 0$ in the hybrid cost function).

b) Estimating the SPT through some TDOA technique.

In this section we present the basics of these techniques and also include references to some previous works that try to locate the target based only on the correlation among the received acoustic signals. These approaches rely on the availability of at least two different measured signals and, by computing the cross-correlation between them, they obtain the TDOA between the signals. Here, we will consider that these two signals are acquired with two microphones connected to the same node, although extensions that use microphones from different nodes could also be considered.

2.4.1 Estimation of the TDOA

The objective is to find which is the difference in propagation time between the acoustic source and the two microphones. TDOA methods are based on selecting the delay that maximizes the correlation between the signals registered at the two microphones, given by:

$$r_1[n] = x[n - D[n]] + n_1[n] \tag{45}$$
$$r_2[n] = x[n] + n_2[n] \tag{46}$$

where:

- $x[n]$ denotes the source signal.
- $r_i[n]$ is the received signal at the i^{th} sensor.
- $n_i[n]$ is the additive noise signal at the i^{th} sensor.
- $D[n]$ is the value of the delay at time instant n.

It is assumed that $x[n]$, $n_1[n]$ and $n_2[n]$ are mutually uncorrelated.

The target is to find the value of the delay, $D[n]$, that minimizes the error function, $e[n]$, which can be defined as the difference between r_1 and r_2, i.e.:

$$e[n] = r_1[n] - r_2[n] = x\left[n - D[n]\right] - x[n] + n_T[n] \tag{47}$$

where $n_T[n]$ is a noise term denoting the difference between $n_1[n]$ and $n_2[n]$.

In [9] they obtain the value of the delay at time n using an LMS estimation model based on the *autocorrelation* function of $r_2[n]$ and the *correlation* between r_1[n] and $r_2[n]$. The said LMS solution gives the estimated delay value, τ, between the two received signals, but may not be intuitive enough for the inexperienced reader. Using a graphical representation, this algorithm “looks for” the value k that maximizes the cross-correlation between r_1 and r_2 (see Figure 12), that is:

$$\max_k \left(\mathbb{E}\{\mathbf{r_1}[n]\mathbf{r_2}^T[n]\}\right) = \max_k \left(\mathbb{E}\{\mathbf{r_1}[n]\mathbf{r_1}^T[n-k]\}\right) \tag{48}$$

To do so, they start by taking a *window* (finite set) of input samples, thus working with $r_1[n]$ and $r_2[n]$ as vectors. From this, the following expressions can be defined:

$$R_{12}[n] = \mathbb{E}\{\mathbf{r}_1[n]\mathbf{r}_2^T[n]\} \tag{49}$$
$$R_{22}[n] = \mathbb{E}\{\mathbf{r}_2[n]\mathbf{r}_2^T[n]\} \tag{50}$$

where $R_{12}[n]$ denotes the *correlation* between $\mathbf{r}_1[n]$ and $\mathbf{r}_2[n]$ and $R_{22}[n]$ is the *auto-correlation* function of $\mathbf{r}_2[n]$.

The said LMS model requires an adaptive filter to estimate the delay. Now, at every time instant n, the filter weight vector minimizing $\mathbb{E}\{e^2[n]\}$ can be obtained as:

$$\hat{H}_d[n] = R_{22}^{-1}[n]R_{12}[n] \tag{51}$$

where sub-index d is related to the *delay* we want to estimate. Then, they obtain the least-squares solution using a similar expression to (14):

$$\hat{H}_d[n+1] = \hat{H}_d[n] + \mu_\tau \mathbf{e}[n]\mathbf{r}_2[n] \tag{52}$$

where $e[n]$ represents the difference between $\mathbf{r}_1[n]$ and $\mathbf{r}_2[n]$ and μ_τ is the step size.

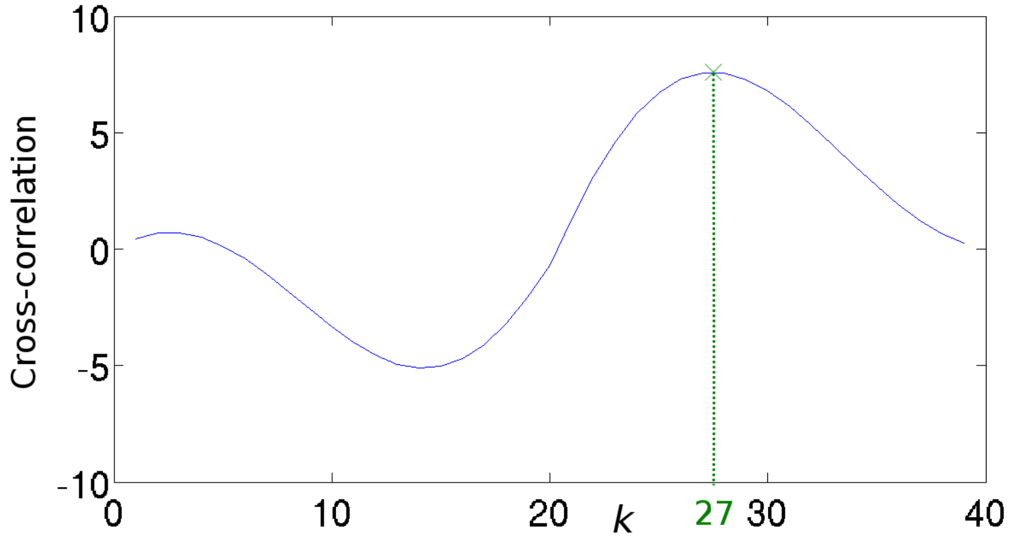


Figure 12: TDOA estimation from the correlation between r_1 and r_2. In this case the delay $\tau = k = 27$ μs

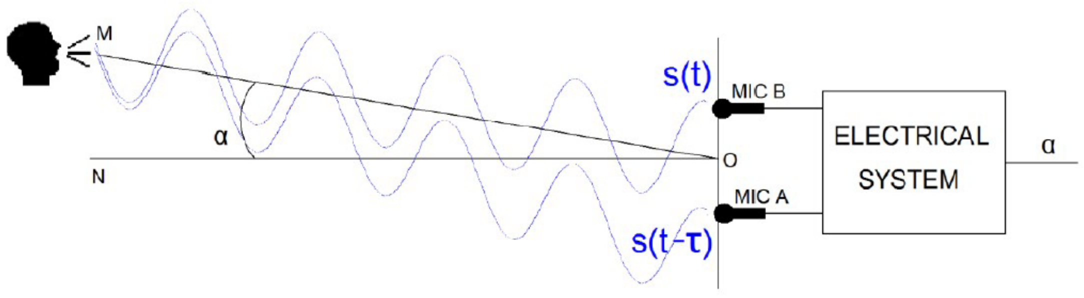


Figure 13: Two-microphone model, taken from [7]. This figure illustrates how the acoustic wave coming from source M reaches microphones A and B, being τ the time delay of arrival (TDOA) between them and α the direction of arrival of the wave.

2.4.2 Estimation of the angle of arrival (DOA)

The objective is to find which is the direction of arrival of the acoustic wave after we have determined its TDOA.

Considering an acoustic source M located in front of two omnidirectional microphones (see Figure 13), the goal is to determine the *direction of arrival* (DOA) of its sounds. The microphones are placed in a fixed position and separated by a certain distance and then the origin of coordinates of the system, O, is set in the middle of them. Considering the orthogonal line to the microphone axis at the origin, $\vec{ON}$, the angle α is defined by the separation angle between this line and line $\vec{OM}$. From now on, the term *direction of arrival* (DOA) refers to the angle α where the noisy source is located [7].

This system presents an obvious drawback: it cannot locate sounds coming from behind the microphones. Since the microphones are omnidirectional (i.e. the microphone can "hear" from all directions with the same sensitivity), they will sense the same sound pressure level regardless of its DOA, which becomes problematic when the sound is coming from behind them because they would output the same values as if the sound were in front of them (see Figure 14).

Considering the case in which the acoustic source is located "in front of" the microphones, once both signals are captured, they are processed to estimate this time delay. Then, with the help of trigonometric calculus, the angle α is returned.

Taking a look at Figure 15 we can state the following: considering that the coordinates of the source M are given by (x,y) and the coordinates of microphones A and B are (x_A,y_A) and (x_B,y_B) respectively, it is clear that the sound wave reaches microphone B before it can reach microphone A, so there exists a *delay*, τ, in the signal retrieved at microphone A. Segment AB' represents the distance travelled by the signal during that delay.

Considering this and having Figure 15 in mind, the following equation can be derived (taken from [7]):

$$AB' = AM - B'M = AM - BM \tag{53}$$

where AM and BM are expressed as (see Figure 15):

$$AM = \sqrt{(x_A - x)^2 + (y_A - y)^2} \tag{54}$$

$$BM = \sqrt{(x_B - x)^2 + (y_B - y)^2} \tag{55}$$

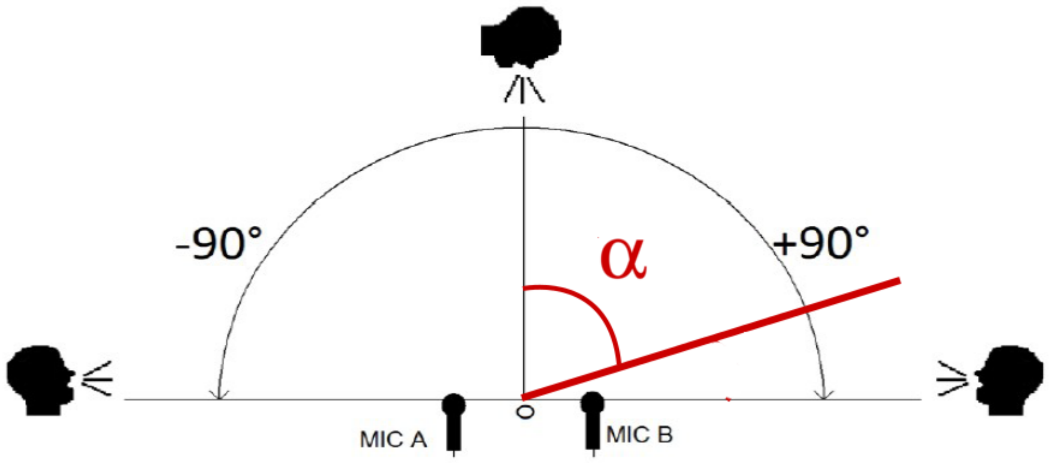


Figure 14: Coverage angles of the two-microphone model, taken from [7]. The value of α denotes the direction of arrival of the acoustic wave: a negative value of α means the sound is coming from the left, a positive value means the sound is coming from the right, and the value 0 denotes the source is in front of the microphone pair.

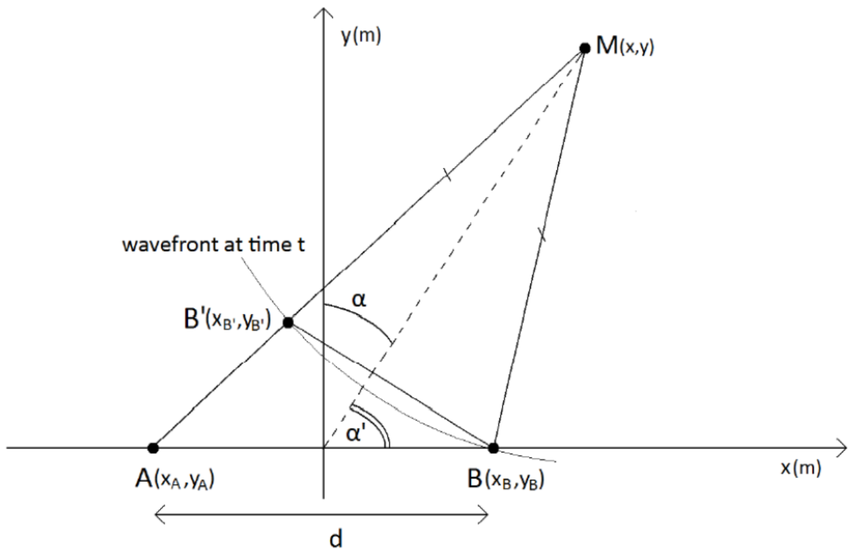


Figure 15: Calculation of α, taken from [7]. It can be seen that the acoustic wave reaches microphone B before microphone A, so there exists a delay in the signal retrieved at microphone A. Therefore, segment AB' represents the distance travelled by the signal during that delay.

In order to get rid of the square roots, Equation (53) is squared. Now, since the two microphones have fixed positions the following statements apply (see Figure 15):

$$x_A = -x_B \tag{56}$$

$$y_A = y_B = 0 \tag{57}$$

which simplifies the previous equations and, after several calculations and term reordering, leads to:

$$y = \pm\sqrt{\frac{AB'^2}{4} - x_B^2 + x^2 \cdot \left(\frac{4 \cdot x_B^2}{AB'^2} - 1\right)} \tag{58}$$

Since both the values of x_B and AB' remain unchanged regardless of the direction, Equation (58) represents all the possible positions of the source M, given a certain delay [7]. But this presents a problem: for a certain delay, Equation (58) is not defined for all values of x, only for those which make the expression inside the square root positive, that is:

$$x \geq \sqrt{-\frac{AB'^2 \cdot (AB'^2 - 4x_B^2)}{4 \cdot (4x_B^2 - AB'^2)}} \tag{59}$$

In order to better understand this, Figure 16 represents Equation (58), which takes positive values from a certain value of x. Considering that the signal travels at the speed of sound, $v \approx 340$ m/s, the distance $AB' = \tau \cdot v$. The value of τ is obtained following the steps in section 2.4.1.

It can be seen in Figure 16 that Equation (58) has a hyperbolic evolution until a certain point and then becomes linear. Only taking the linear part, first its slope must be obtained and then its arctangent in order to get angle α', which is the angle formed by the x-axis and $y(x)$ in Figure 15:

$$\alpha' = \tan^{-1}\left(\frac{dy(x)}{dx}\right) = \tan^{-1}\left(\frac{2x\sqrt{\frac{(AB'^2-4x_B^2)(AB'^2-4x^2)}{AB'^2}}}{AB'^2 - 4x^2}\right) \tag{60}$$

Since α is the angle formed by the y-axis and $y(x)$, we do the following:

$$\alpha = \begin{cases} 90 - \alpha' & \text{if } \alpha' \geq 0 \\ 90 + \alpha' & \text{if } \alpha' < 0 \end{cases} \tag{61}$$

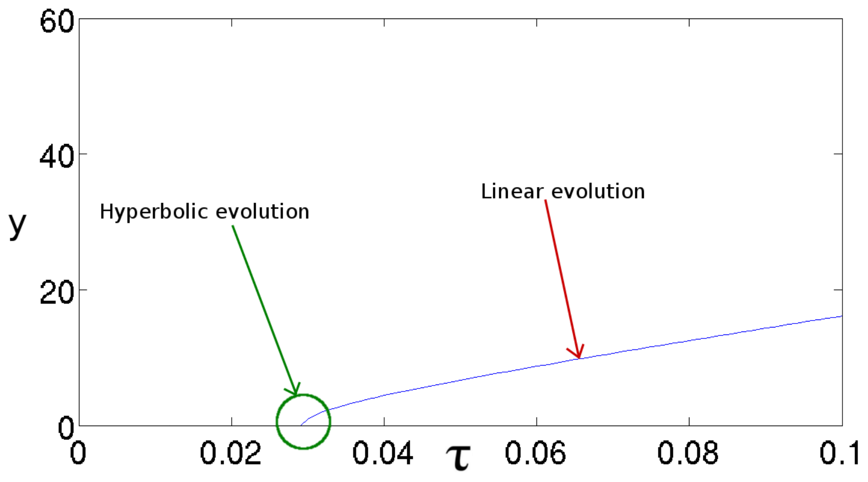


Figure 16: Possible positions of the acoustic source according to Equation (58), given a certain delay for x = 5

Note that the negative values of α' correspond to the cases in which the signal arrives first to microphone A than to microphone B.

In the two-microphone scenario the problems that may appear are related to temporal or spatial *aliasing*. The first one can be easily solved using *Nyquist* theorem. The second one can take place in the following case: Having in mind Figure 15, distance $B'A$ depends on the distance between microphones, d. If this distance increases up to d', the distance $B'A$ increases too. It is clear that the delay has the same behavior.

The problems appear when the microphones are too separated (see Figure 17). In this case, the delay can increase until a maximum value of $\lambda/2$ (λ is the acoustic wave length). If that occurs, two situations may arise: the detected delay is wrong or the delay is not detected at all. The delay is the time it takes for the signal to traverse $B'A$. In this case, $B'A$ is equal to a whole wavelength, so the signals captured by both microphones are equal and no delay is detected. This is obviously untrue, since the delay exists.

The case in which the detected delay is not the real one takes place when $\lambda/2 < B'A < \lambda$ (see Figure 18). This means that the microphones must be separated at least a distance larger than the acoustic wave length λ (or smaller than $\lambda/2$ but then they would be too close to each other).

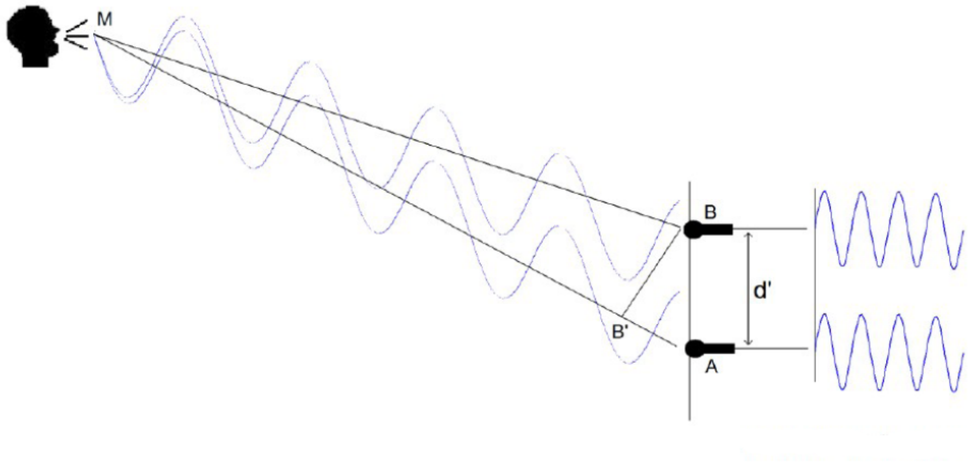


Figure 17: Delay not detected, taken from [7]. In this case, the delay is equal to the wave length λ, so the signals captured by both microphones are the same and no delay is detected.

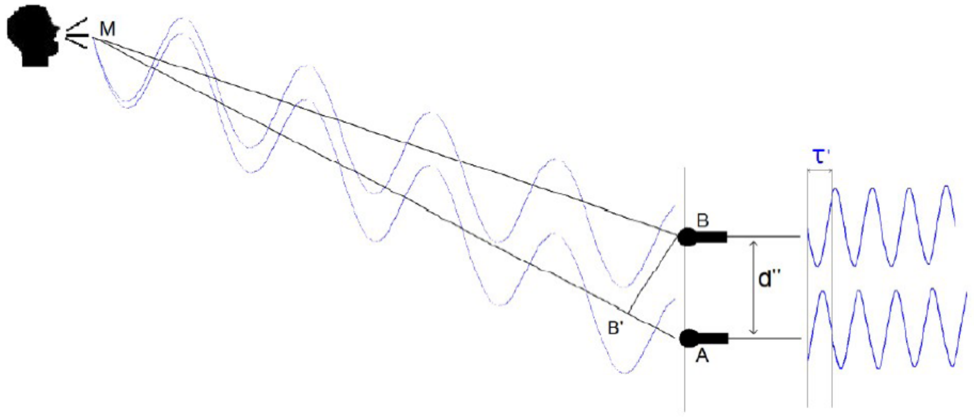


Figure 18: Wrong delay is detected, taken from [7]. In this case $\lambda/2 <$ B'A $< \lambda$ and the detected delay results in a smaller value than the real one.

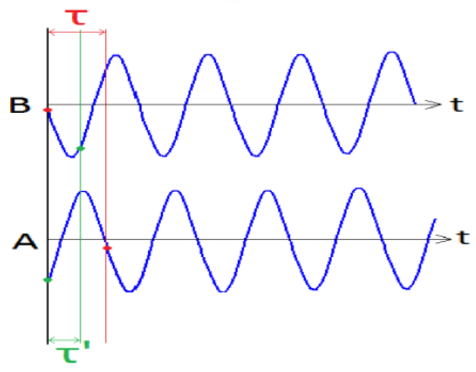


Figure 19: Delay smaller than the optimal, taken from [7]. The real value of the delay is the one coloured in red, whereas the green one corresponds to the detected value of the delay.

Taking a closer look (see Figure 19), the real delay τ is the one coloured in red. In Figure 18 we can see that the source is closer to MIC B so the signal reaches MIC B before MIC A. However, if these signals are inserted into a system whose aim is to obtain the delay, the system would return the green coloured delay, τ'. Actually τ' is the only delay smaller than $\lambda/2$, thus it will be mistakenly identified as the existing delay. So when two signals like those are captured, it would seem that the speaker is closer to MIC A than to MIC B, hence the delay obtained is false. The condition that must be fulfilled in order to avoid spatial aliasing is:

$$B'A \leq \frac{\lambda}{2} \tag{62}$$

To obtain the maximum distance between microphones, it is necessary to find the minimum value for λ. That value depends on the source of the sound, since every source may work on a different frequency range:

$$d \leq \frac{\lambda}{2};\ \ \lambda_{min} = \frac{v}{f_{max}};\ \ d \leq \frac{v}{2 \cdot f_{max}} \tag{63}$$

where v is the speed of sound.

But from a practical point of view this value causes difficulties. Actually, the accuracy when placing the microphones was not assured: the smaller the distance d, the higher would be the impact of a possible error of placement. Furthermore, a higher value of d leads to a higher number of delay samples which leads to a higher precision [7], so we usually pick a higher value for d.

This approach to the solution to the acoustic source localization problem can be extended to any number of microphones, but it is very computationally expensive and not suitable for real-time applications. The interested reader can consult [8, 9, 10, 11], in which they also estimate the position of an acoustic source via the estimation of the TDOA of the sound waves it generates, for a more extensive analysis of the use of the cross-correlation between any number of signals to estimate the position of the acoustic source.

2.5 Conclusions

In this chapter we have presented the centralized and distributed LMS algorithms for source localization and tracking of acoustic sources. These methods are based on the availability of two kinds of measurements: one based on the registered acoustic pressure, and a second one based on the propagation time. The availability of the latter is difficult in practice and we can either neglect the corresponding term (i.e. setting $\eta = 0$) or we can exploit results based on TDOA techniques. In the centralized case the central node is responsible for estimating the position of the source using the $L_{p_k}[n]$ and $t_k[n]$ measurements sent from *every* node in the network, whereas in the distributed scenario each node uses that pair of values to update an intermediate state, $\boldsymbol{\psi}_{k,n}$, which will then interchange with its neighbors in order to merge those intermediate states into a final estimation of the position of the target.

In the following section we present a simulation tool that allows to examine the properties of the centralized and distributed algorithms, while allowing also to assign different importances to the SPT and SPL terms of the hybrid cost function (20).

3 The Simulation Tool

The implementation of a simulation tool for centralized and distributed audio target localization is the main goal of this Final Year Project. In previous sections we have presented the *centralized LMS* and *diffusion ATC* algorithms for the localization and tracking of noisy targets. In order to get a visual representation of their respective behaviors we have developed a simulation tool implementing the said algorithms and a Graphical User Interface (GUI) to help the user build and configure the network parameters that will define the tracking process.

The implemented simulation tool allows the user to configure several parameters of the acoustic source localization and tracking problem in order to simulate the behavior of the algorithms under those conditions. It needed to fulfill certain requirements, listed below.

3.1 Requirements

The functionality of the simulation tool is given by the supervisor of the Project, who has defined the following *criteria*:

1. Both the software and the GUI must be implemented in MATLAB®, a very powerful matrix-oriented mathematical processing tool.

2. The software must implement a function for each diffusion algorithm, which have to be robust regardless of the network configuration.

3. The GUI must be simple enough to follow, since it will serve as the network-configuring tool for the simulations.

4. In that configuration the user must be able to define:

   - Topology of the network:
     - Number of nodes in the network.
     - Behavior of the network, i.e., centralized or distributed.
     - Reach of the nodes in the distributed scenario. Consequently, this also establishes the number of neighbors of each node.

- Real trajectory followed by the target, i.e., the one which will be estimated by the algorithms.
- Step sizes for both algorithms, μ_{cent}, μ_{dist}.
- Parameter η.
- Parameters associated to the acoustic source:
  - SPL measured at $d_0 = 1$ m, L_{p_s}.
  - Standard deviation of the noise associated to propagation time, σ_{n_t}.
  - The attenuation exponent corresponding to the log-distance path loss model, α.
- Number of runs to average the estimation error.

5. The simulation must be real-time oriented, that is, it must represent a noisy source in motion and its trajectory estimation by the sensor network at every time instant n.

6. The simulation must show the estimated trajectories of two algorithms simultaneously.

7. Finally, a graphical representation of the errors in the estimation must be provided in order to compare the chosen configurations.

3.2 Design of the Graphical User Interface

The GUI has been designed to guide the user through all the steps needed for configuring a sensor network, which will be working in either a centralized or distributed manner in order to estimate the trajectory followed by an acoustic source in motion, as well as to show the accuracy of the estimation performed by such network:

1. **Introduction:** In this window (Figure 20) the GUI greets the user and lets him know the Demo will guide him/her through the process.

2. **Node placement:** In the windows corresponding to Figures 21 and 22, the GUI informs the user of the way of placing the nodes in the network and lets him introduce the total amount of nodes in the network. The user will then place the said number of nodes in the network by clicking on an empty axes.

After the user has placed the last node, the GUI saves the selected coordinates in the axes into an array of coordinates, $s = \{s_1 = [x_1, y_1], ..., s_N = [x_N, y_N]\}$.

We limited the total number of nodes in the network to a minimum of 4 and a maximum of 10 for computational and user-experience reasons. Since the tool has to wait for the user to manually place each node in the network, working with a large number of nodes can be tedious for the user and, moreover, would lead to very slow simulations.

3. **Virtual trajectory generation:** We need to *track* a trajectory using the noise sensors. In order to do that the user is asked to provide such trajectory. In the window corresponding to Figure 23 the GUI informs the user about the need of a virtual trajectory and gives instructions for its generation. Then the GUI displays the final simulation environment, which contains the network nodes and the virtual trajectory (see Figure 24).

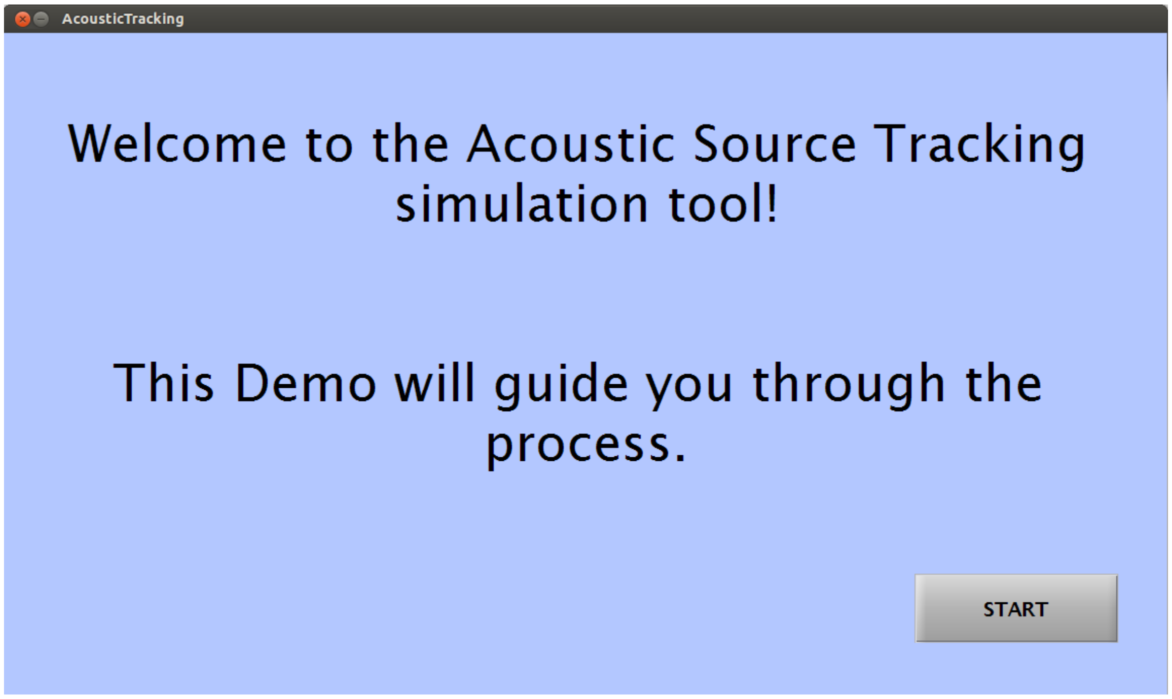
   For the generation of the trajectory the user is asked to select ten points. After those points have been selected we perform a *spline* approximation of the resultant curved trajectory using the MATLAB® function *EvaluateCardinal2DAtNplusOneValues*, which is available at `www.mathworks.com`. It is important to highlight that there exist the same amount of points between each selected point. This means that the points selected by the user will be equidistant in time, i.e., it will take the same time to get from one point to the next regardless of the distance between them. This allows the user to simulate changes in the speed of the acoustic source, by increasing the distance between the selected points.

4. **Algorithm description:** Once the simulation *setup* is finished, the GUI displays the window corresponding to Figure 25, in which a brief explanation of the algorithms is displayed.

5. **Simulation setup and display of results:** In the window corresponding to Figure 26 the user will be able to configure the parameters for the two algorithms that will perform the estimation. Such parameters are the ones listed in the previous section. If the user chooses to perform one of the estimations by using the *diffusion ATC* algorithm, the user can select the coverage range of each node in the network. By doing so, each node will compare that range with its relative distances to the remaining nodes in order to identify its *neighbors* and the neighbor groups are displayed in a small axes.

   Figure 27 shows the simulation taking place. It first plots the trajectories estimated by each algorithm on a single run, and then calculates each algorithm's *mean-square error* averaged over the specified number of iterations, which are then plotted (see Figure 28).

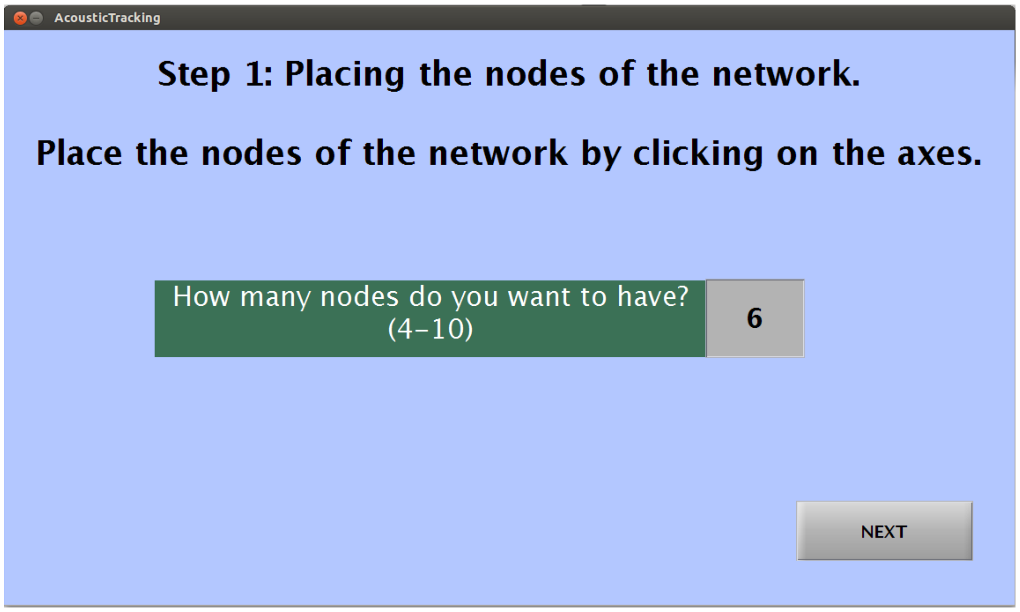
The possibility of setting all these parameters provides the user complete control of the stage on which to run the simulations, as well as the visual illustration of the results. In this way, the user can set the same network, working under two different parameter configurations, to compare the achieved performance in terms of MSE.



Welcome to the Acoustic Source Tracking simulation tool!

This Demo will guide you through the process.

Figure 20: Greeting GUI window


AcousticTracking

Step 1: Placing the nodes of the network.

Place the nodes of the network by clicking on the axes.

How many nodes do you want to have? (4–10) 6

Figure 21: Node placement information and selection of the total number of nodes in the network

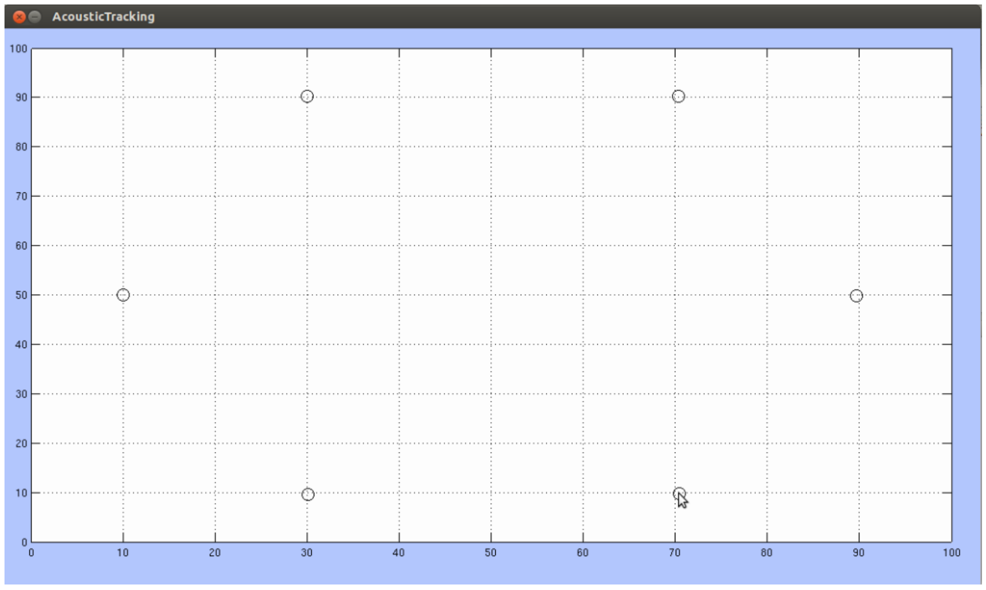
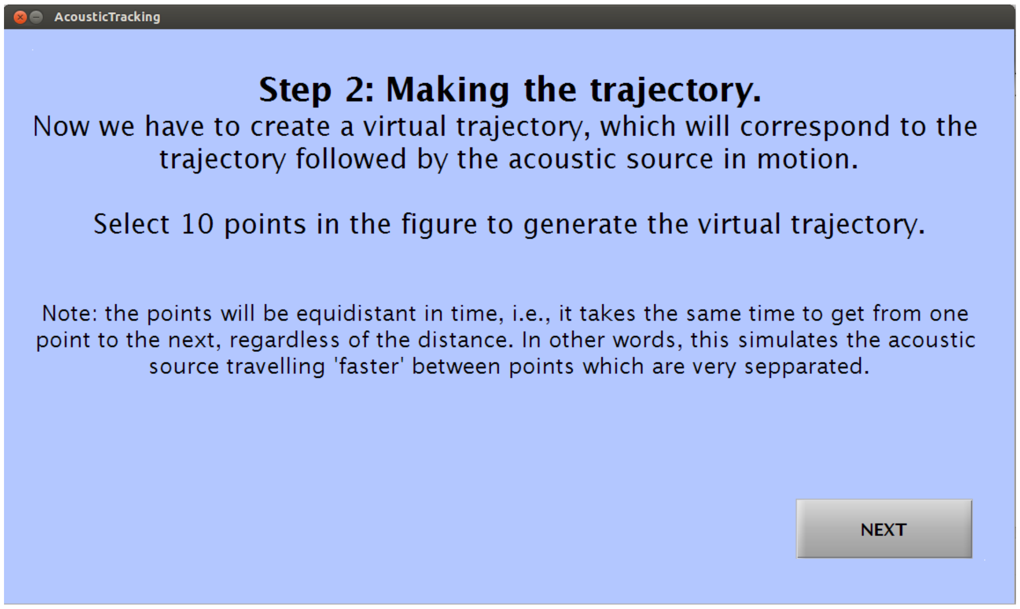


Figure 22: Node placement. This is performed by clicking on the axes as many times as number of nodes the user wants to have in the network



Step 2: Making the trajectory.

Now we have to create a virtual trajectory, which will correspond to the trajectory followed by the acoustic source in motion.

Select 10 points in the figure to generate the virtual trajectory.

Note: the points will be equidistant in time, i.e., it takes the same time to get from one point to the next, regardless of the distance. In other words, this simulates the acoustic source travelling 'faster' between points which are very sepparated.

Figure 23: Trajectory generation: how to generate the *real* trajectory and side-note explanation of the meaning behind the points being equidistant in time

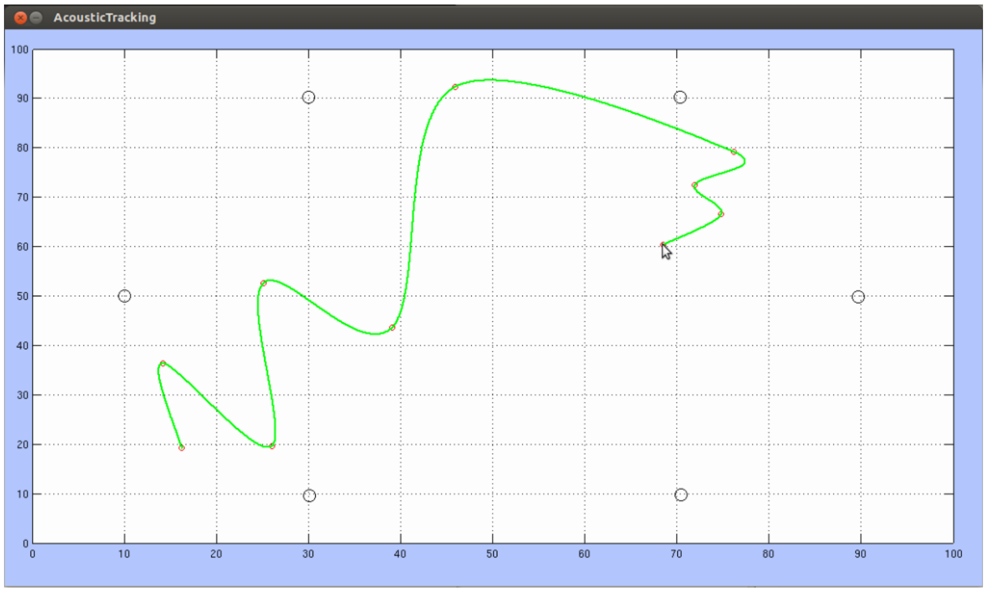
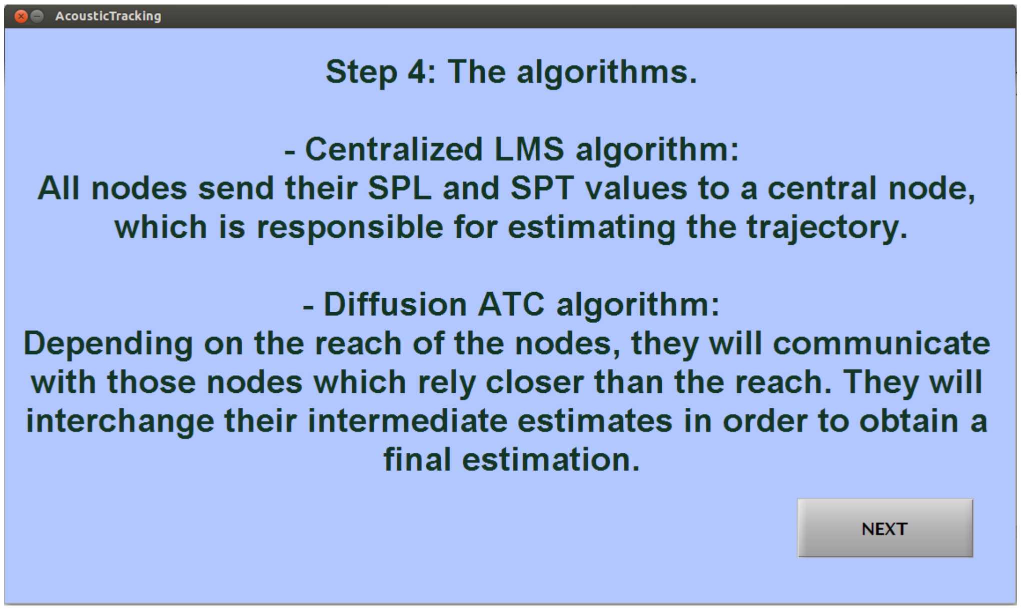


Figure 24: Trajectory generation: trajectory generated by the user



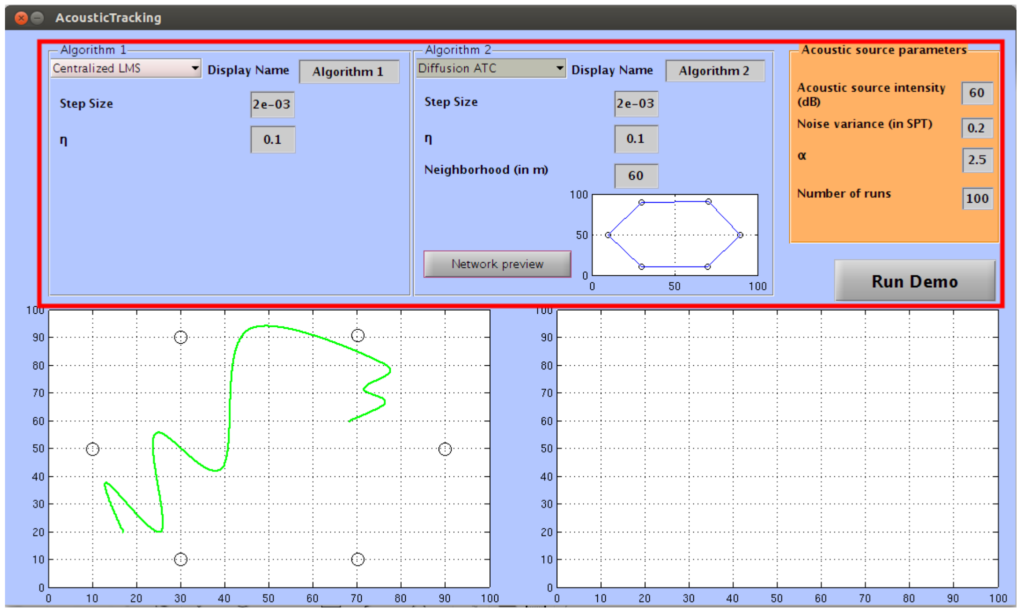
AcousticTracking

Step 4: The algorithms.

- Centralized LMS algorithm:
All nodes send their SPL and SPT values to a central node, which is responsible for estimating the trajectory.

- Diffusion ATC algorithm:
Depending on the reach of the nodes, they will communicate with those nodes which rely closer than the reach. They will interchange their intermediate estimates in order to obtain a final estimation.

Figure 25: Brief explanation of the *Centralized LMS* and *Difussion ATC* algorithms



AcousticTracking

Algorithm 1: Centralized LMS — Display Name: Algorithm 1
Step Size: 2e-03
η: 0.1

Algorithm 2: Diffusion ATC — Display Name: Algorithm 2
Step Size: 2e-03
η: 0.1
Neighborhood (in m): 60

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

Figure 26: Configuration of the parameters of each algorithm.

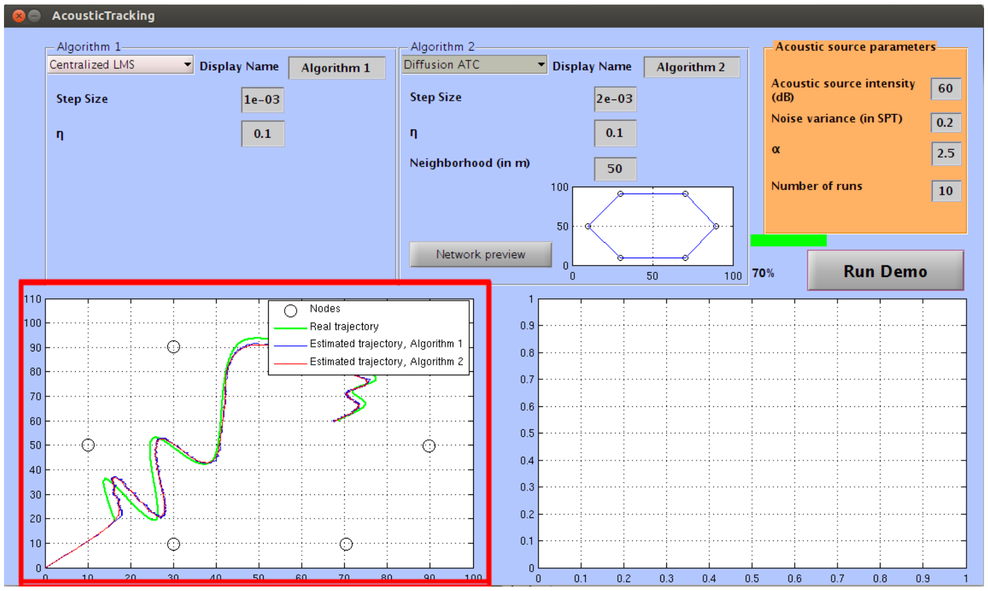


Figure 27: Simulation running: The estimated trajectories are plotted in the lower-left axes and the MSE of each algorithm are being calculated, averaged over 10 iterations (number of runs).

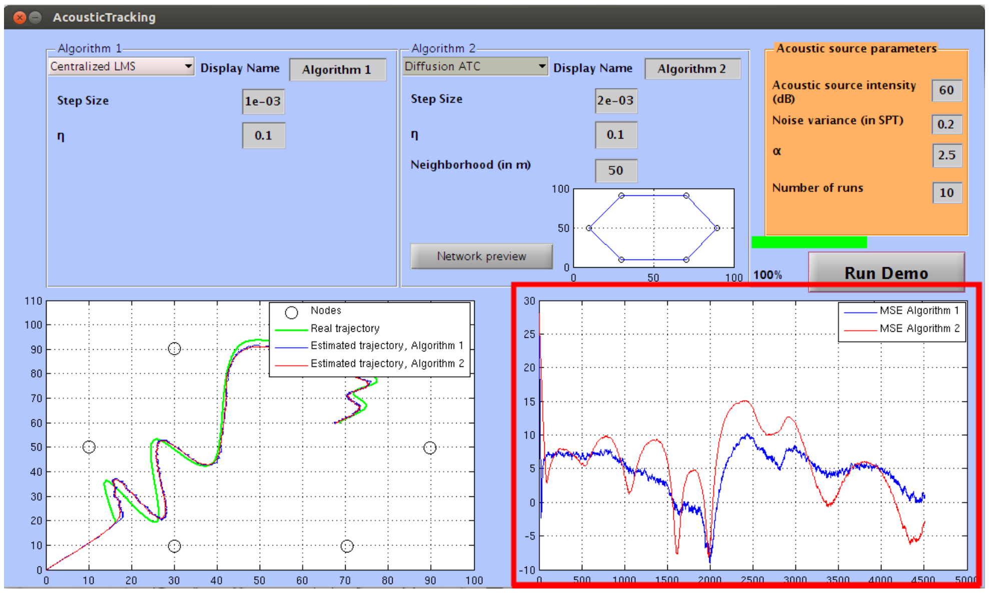


Figure 28: End of the simulation: Once the final MSEs of both algorithms have been obtained, they are plotted in the lower-right axes.

3.3 Design of the functionalities of the simulation tool

In this section we will describe the processes taking place in the background of the simulation tool, i.e., all the calculations and functions used throughout the simulation process, which is illustrated in the following diagram:

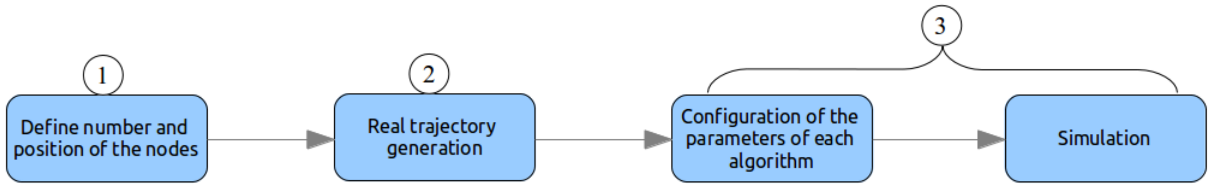

1
Define number and position of the nodes
2
Real trajectory generation
3
Configuration of the parameters of each algorithm
Simulation

Figure 29: Main blocks of the simulation tool.

The processes corresponding to blocks 1 and 2 are very straightforward:

1. An editable field allows the user to manually select the total number of nodes in the network. That value is stored internally in a variable, N, after pressing the 'NEXT' button.

2. In order to place the nodes in the network, we call the MATLAB® *ginput(N)* function and wait until the user has finished selecting the points. Once the user has placed the last node we store the coordinates of all the nodes in an array of points, $ND = \{s_k = [x_k, y_k]\}$, with $k = 1, ..., N$.

3. After the user has been informed of the way of generating the virtual trajectory, we do the following:

   (a) We call the MATLAB® *ginput(N)* function with $N = 10$ (the user is asked to select 10 points) and wait until the user has selected the last point.

   (b) After that, we perform a *spline* approximation of the resultant curved trajectory using the MATLAB® function *EvaluateCardinal2DAtNplusOneValues*, available at `www.mathworks.com`.

   (c) Finally we display the resultant virtual trajectory and store it as a global variable, called *trajectory*, into an array of length 4500.

After the node placement and trajectory generation steps the GUI shows a window with several editable fields and a single button (see Figure 26). This window corresponds to block 3 and includes the fields for configuring the algorithms: step sizes, η, reach of the nodes in the case of the Diffusion ATC algorithm; and the parameters corresponding to the acoustic source: sound pressure level measured at 1 meter (i.e., the intensity of the source), variance of the noise term associated to the signal propagation time, the attenuation exponent corresponding to the log-distance path loss model, α, and the number of runs to average the estimation error.

A block diagram illustrating the simulation process is depicted in Figure 30. Every time the user presses the 'Run demo' button a sequence of steps is triggered:

1. **Reset the plots:** The GUI first clears the contents of the plots, leaving just the nodes and the real trajectory.

2. **Generation of the SPL and SPT values:** In this step we calculate the respective SPL and SPT values that would be measured at each node following Equations (17) and (19) respectively.

   These equations are based on the real distance between every point in the trajectory and each node in the network, so we need to obtain those distances. Once we have obtained the distances we are able to calculate the theoretical SPL and SPT values by calling a function for each node in the network, called *gen_pressure*, which takes as input values the SPL value emitted by the source, L_{p_s}, a vector of distances from the node to every point of the trajectory, $\mathbf{d}$, the reference distance, d_0, and parameter α. The latter is obtained from the editable field labeled as 'α' in the GUI. The SPT values are obtained directly by dividing the vector of distances $\mathbf{d}$ by the speed of sound.

3. **Loop:** First, the *nruns* variable is set to the value appearing in the editable field labeled as 'Number of runs'. Now, for every run we do the following:

   (a) **Generation of the real SPL and SPT values:** The real SPL values are obtained by adding random noise to the calculated theoretical SPL values, following Equation (18). Meanwhile, the real SPT values are obtained by adding random noise to the calculated theoretical SPT values. The standard deviation of the noise in propagation time is selected as the squared root of the value contained in the editable field labeled as 'Noise variance (in SPT)'

   (b) **Run the algorithms:** The algorithms that will perform the estimation have been selected by the user in the top *popupmenus*.

   Depending on the chosen algorithms we call the following functions:

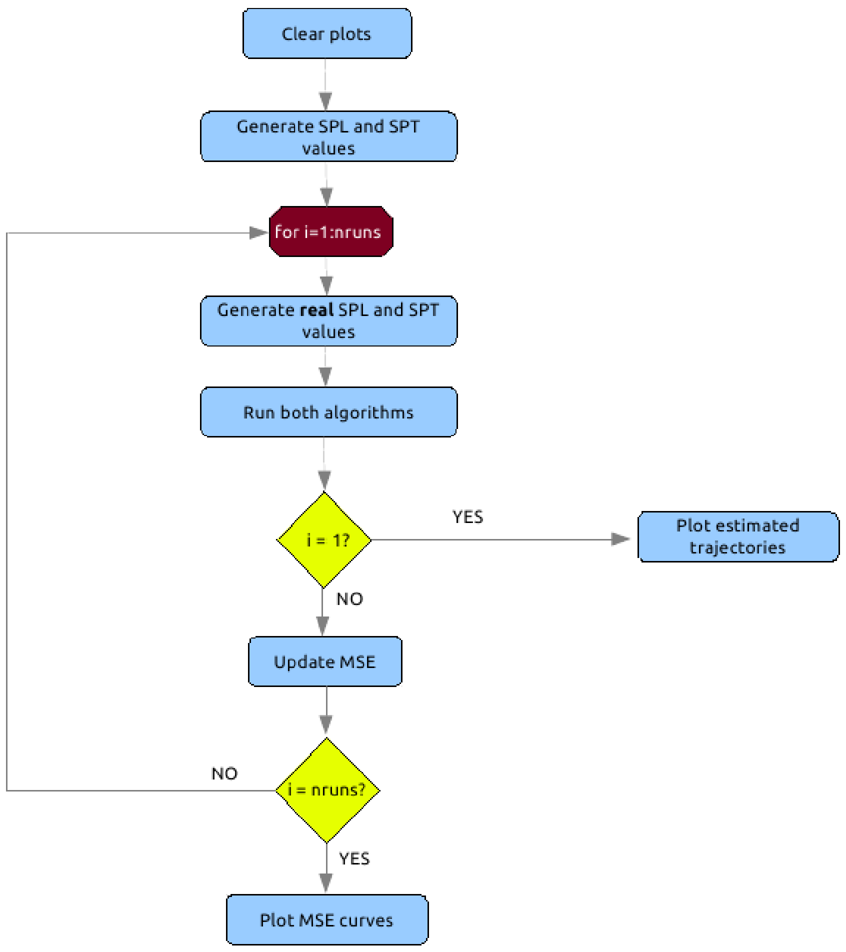


Figure 30: Flowchart illustrating the simulation process.

- `w = centralized((ND,alpha,mu,etha,v,SPL,SPT,h)):`
  This function runs the Centralized LMS algorithm. It takes as input parameters the ones mentioned previously and it returns the estimated trajectory, **w**.
- `w = difusion(ND,reach,alpha,mu,v,etha,SPL,SPT,h):`
  This function runs the Diffusion ATC algorithm. It takes as input parameters the ones mentioned previously, along with the reach of the nodes, and returns the estimated trajectory, **w**, which corresponds to the *mean* of the N estimated trajectories, N being the number of nodes in the network.

(c) **Plotting the estimated trajectories:** In the first run we plot the trajectories estimated by each algorithm, $\mathbf{w}_1$ and $\mathbf{w}_2$, in the lower left axes. We only perform this step once because we want to illustrate the *real* behavior of the algorithm, i.e. the typical aspect of the estimated trajectory in each run of the algorithm. The average over the number of runs of all the estimations would be a much less informative result.

We want to highlight that in the case of the Diffusion ATC algorithm we plot a trajectory which corresponds to the *average* of the N estimated trajectories, N being the number of nodes in the network. Remember that in the distributed case *each* node in the network performs an estimation of the position based on the *intermediate states* of its neighbors (see Section 2.3 for a complete explanation of this).

(d) **Update the value of the MSE:** The calculation of the MSE is performed by averaging the squared error in the estimation over the total number of runs. In every run we add different noise to the theoretical SPL and SPT values in order to simulate how real noise affects the estimation, and we calculate the squared Euclidean distance between each point of the estimated trajectory and the corresponding point in the trajectory, i.e., $\| \mathbf{w} - \textit{trajectory} \|^2$

4. Once the loop has finished, we plot the MSE on the lower-right axes. Specifically, we plot a curve corresponding to:

$$10\log_{10}\left(\frac{1}{nruns}\sum_{i=1}^{nruns} MSE_i\right)$$

4 Simulations

In the previous section we have presented the aspect and functionalities of the implemented simulation tool. In this section we will show its utility, i.e., we will run some simulations under different conditions in order to illustrate how the simulation tool can be used to analyze the influence of different algorithm parameters, as well as to compare the centralized and distributed solutions.

We will illustrate the behavior of the algorithms when we modify the following settings:

- Step size, μ.
- Parameter η.
- Centralized vs. Distributed solutions.
- Topology of the network in distributed solutions (reach of the nodes).
- Variance of the noise associated to the signal propagation time.
- Parameter α.

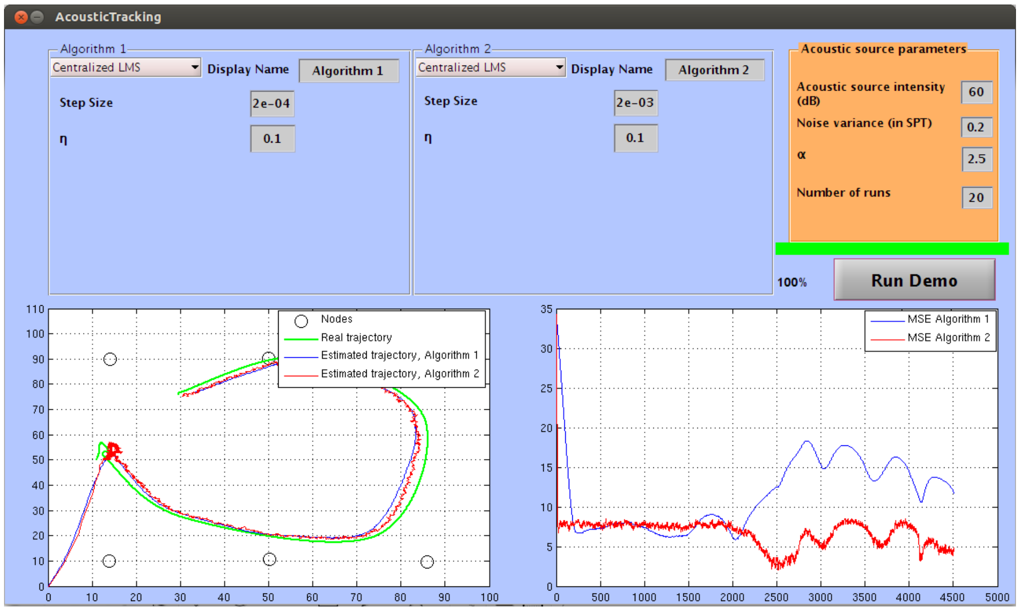
We should emphasize that our goal when providing these simulation results is not to provide a complete evaluation of the algorithms and their properties, but just to illustrate the functionality of the implemented simulation tool, and how it can be very helpful to study the audio target tracking problem and ultimately for the design of sensor networks that can guarantee certain performance criteria.

4.1 Effect of the value of the step size

The step size of the algorithms represents the *speed* at which each algorithm converges.

We simulated an acoustic source emitting a SPL value of 60 dB, performing a time-varying trajectory over a wireless sensor network formed by six sensors, working in a centralized manner. The trajectory has been generated in a way that at the beginning its points are close to each other, i.e., points in which the acoustic source is moving slower, and from a point on they become very distant. For the first algorithm we set the step-size value to $\mu_1 = 0,0002$ and, for the second algorithm, to $\mu_2 = 0,002$. The remaining parameters are equal for both algorithms and have been adjusted as indicated in Figure 31.

Figure 31 illustrates how for small values of μ, the algorithm takes longer to converge but still presents a better behavior than the one with the higher step size, i.e., the one which converges faster, until the moment the source accelerates. From this moment on the algorithm with the smallest step-size value will not be able to track the optimal solution fast enough, whereas the algorithm with the highest step-size value will be able to adapt to those fast variations.


AcousticTracking

Algorithm 1: Centralized LMS — Display Name: Algorithm 1 — Step Size: 2e-04 — η: 0.1

Algorithm 2: Centralized LMS — Display Name: Algorithm 2 — Step Size: 2e-03 — η: 0.1

Acoustic source parameters: Acoustic source intensity (dB): 60 — Noise variance (in SPT): 0.2 — α: 2.5 — Number of runs: 20

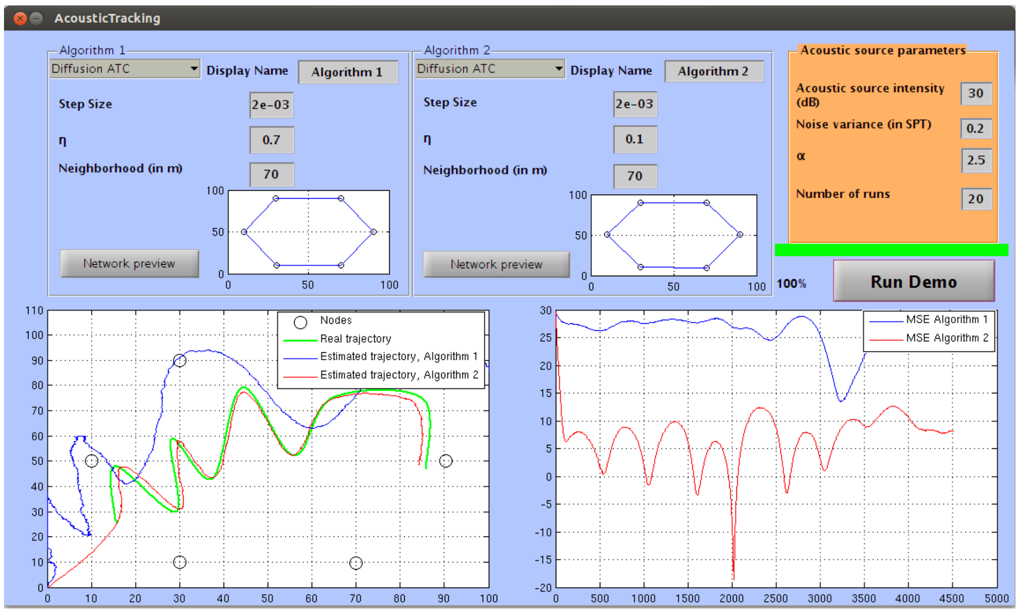
100%

Figure 31: Effect of the Step-size value on the estimation of the trajectory.

4.2 Effect of parameter η

Parameter η denotes how much importance we assign to each error term in the hybrid cost function (20).

To illustrate its relevance we ran a simulation in which an acoustic source emitting with a SPL of 30 dB makes a time-varying trajectory over a wireless sensor network formed by 6 sensors working in a distributed manner, i.e., running the Diffusion ATC algorithm. In order to compare the effects of varying parameter η, we set the reach of the nodes and their respective step sizes to the same value, $r = 70$ m and $\mu = 10^{-3}$, and assigned a value of $\eta_1 = 0.7$ for the first algorithm and a value of $\eta_2 = 0.1$ for the second one. Looking at Figure 32 it is clear that the algorithm running with the smallest value of η gives a better estimation of the trajectory and lower MSE values. This means that it is not advisable to give excessive weight to the SPT term, although the optimal value of θ from a practical point of view may vary depending on the emission level of the source and the variance of the noise associated to the SPT.



Algorithm 1
Diffusion ATC
Display Name: Algorithm 1
Step Size: 2e-03
η: 0.7
Neighborhood (in m): 70

Algorithm 2
Diffusion ATC
Display Name: Algorithm 2
Step Size: 2e-03
η: 0.1
Neighborhood (in m): 70

Acoustic source parameters
Acoustic source intensity (dB): 30
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 20

100%

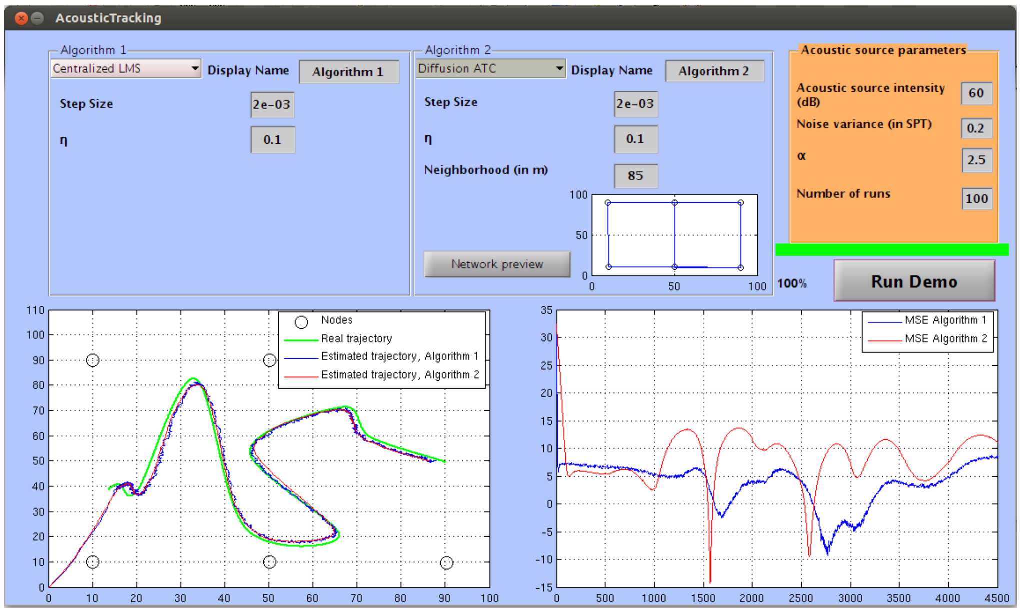
Figure 32: Effects of parameter η in the estimation of the trajectory.

4.3 Centralized vs Distributed algorithms

In this section, we will compare the behavior of both Centralized LMS and Diffusion ATC algorithms when operating under the same conditions.

For the simulations we set a network of 6 nodes. We selected the Centralized LMS as the first algorithm and the Diffusion ATC as the second, with their respective parameters set as shown in Figure 33. It can be seen that under these conditions the distributed solution gives almost as good estimations as in the centralized case.

For a better illustration we performed 2 additional simulations under the same network configuration: in the first one we decreased the step-size value in both algorithms (see Figure 34) and in the second one we set parameter η to 0.5 in both algorithms (see Figure 35). It can be easily seen in Figure 34 that decreasing the step-size value allows the centralized scenario to better adapt to the changes in the trajectory, whereas the distributed scenario is not able to adapt fast enough. Figure 35 shows how increasing the weight associated to the SPT term under these conditions does not affect the estimation results in a positive way.


AcousticTracking

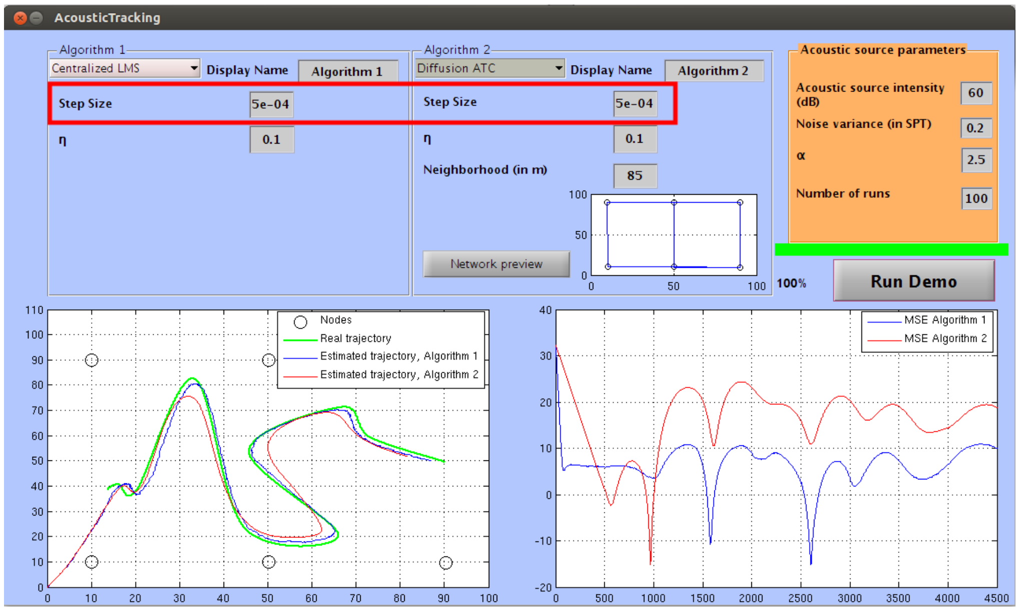
Algorithm 1
Centralized LMS
Display Name: Algorithm 1
Step Size: 2e-03
η: 0.1

Algorithm 2
Diffusion ATC
Display Name: Algorithm 2
Step Size: 2e-03
η: 0.1
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

100%

Figure 33: Centralized vs Distributed scenario.

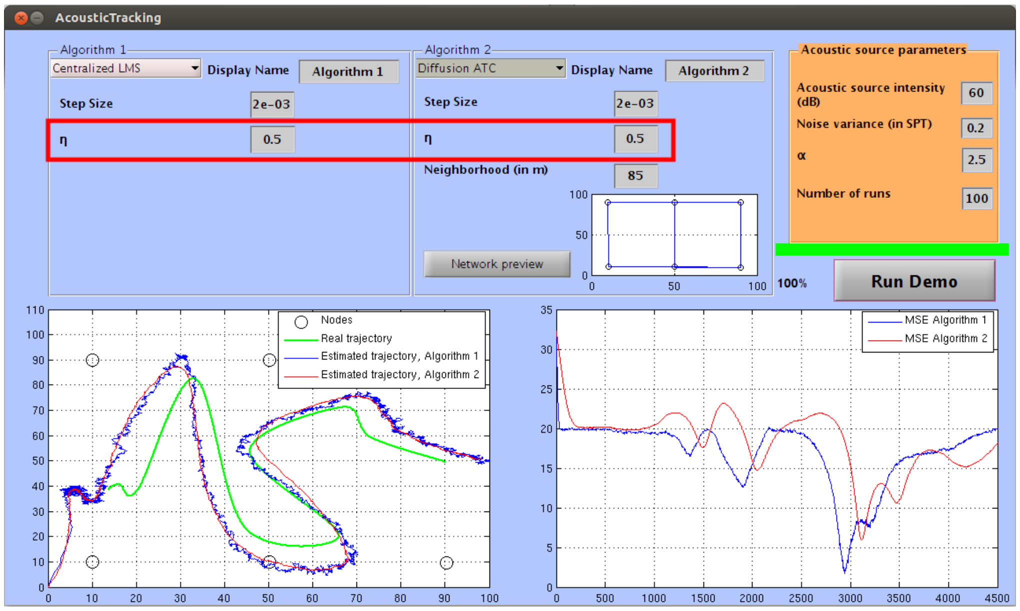

AcousticTracking

Algorithm 1: Centralized LMS
Display Name: Algorithm 1
Step Size: 5e−04
η: 0.1

Algorithm 2: Diffusion ATC
Display Name: Algorithm 2
Step Size: 5e−04
η: 0.1
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

Figure 34: Centralized vs Distributed scenario: effect of decreasing the step-size value.


AcousticTracking

Algorithm 1: Centralized LMS
Display Name: Algorithm 1
Step Size: 2e−03
η: 0.5

Algorithm 2: Diffusion ATC
Display Name: Algorithm 2
Step Size: 2e−03
η: 0.5
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

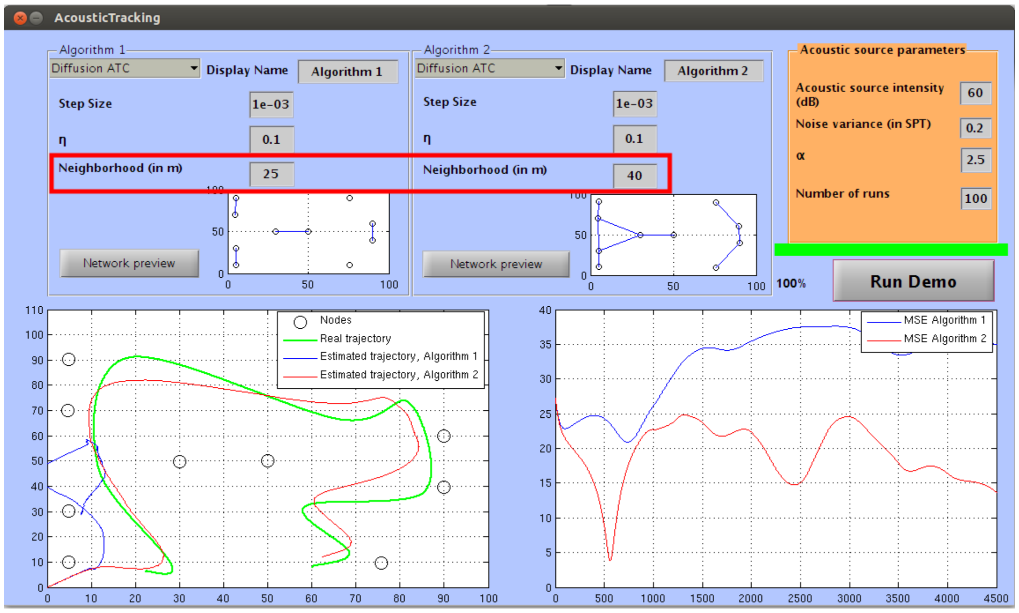
Figure 35: Centralized vs Distributed scenario: effect of increasing parameter η.

4.4 Effect of the reach of the nodes in distributed solutions

In the networks which operate in a distributed manner, the reach of each network node defines its neighbors. An increase in the reach, i.e., an increase in the number of neighbors, means that each node now has access to more information, thus making the Diffusion ATC algorithm more accurate. To illustrate this we simulated a network of ten nodes working in a distributed manner. We assigned each algorithm a node reach of $r_1 = 25$ m and $r_2 = 40$ m respectively. We chose the same step size and η parameter values for both Diffusion ATC algorithms in order for the simulation to be completely *neighbor-dependant*, i.e., the accuracy of the algorithms will depend on the number of neighbors each node has.

Figure 36 illustrates this scenario. It can easily be seen how an increase on the reach of the nodes results in an increase of the number of neighbors of each node and, as a result, it increases the accuracy of the algorithm, thus obtaining a better estimation of the trajectory. Moreover, this Figure illustrates the importance of each node having *at least* two neighbors in the network. As said in Section 2.4.2, working with a pair of omnidirectional microphones gives poor results, we need at least three to obtain a good estimation of the trajectory. This also applies to isolated nodes, i.e. those which do not have neighbors, because they cannot properly adapt their estimations since they do not have access to any network information.

As a final illustration, for the simulation shown in Figure 37 we set $r_1 = 100$ m. In this case each node in the network is neghbor of the rest, thus being able to access the information of the *whole* network, which results in a more accurate estimation of the trajectory.



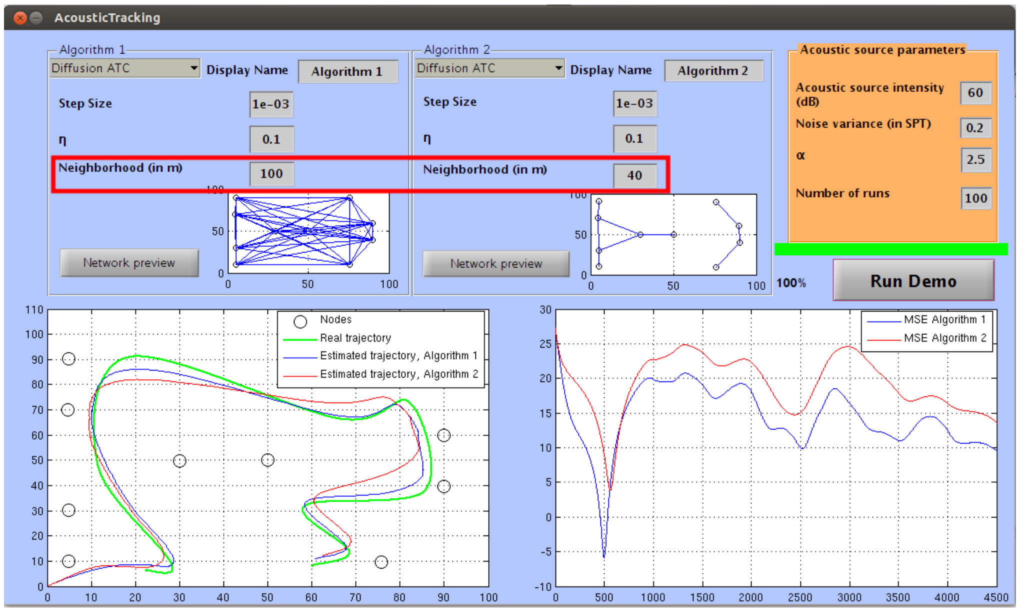
Algorithm 1
Diffusion ATC
Display Name: Algorithm 1
Step Size: 1e−03
η: 0.1
Neighborhood (in m): 25

Algorithm 2
Diffusion ATC
Display Name: Algorithm 2
Step Size: 1e−03
η: 0.1
Neighborhood (in m): 40

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

100%

Figure 36: Illustration of the dependency of the Diffusion ATC algorithm on the reach of the nodes: result of setting a small reach value.



Algorithm 1
Diffusion ATC
Display Name: Algorithm 1
Step Size: 1e−03
η: 0.1
Neighborhood (in m): 100

Algorithm 2
Diffusion ATC
Display Name: Algorithm 2
Step Size: 1e−03
η: 0.1
Neighborhood (in m): 40

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 100

100%

Figure 37: Illustration of the dependency of the Diffusion ATC algorithm on the reach of the nodes: result of setting a high reach value.

4.5 Effect of the variation of the acoustic parameters

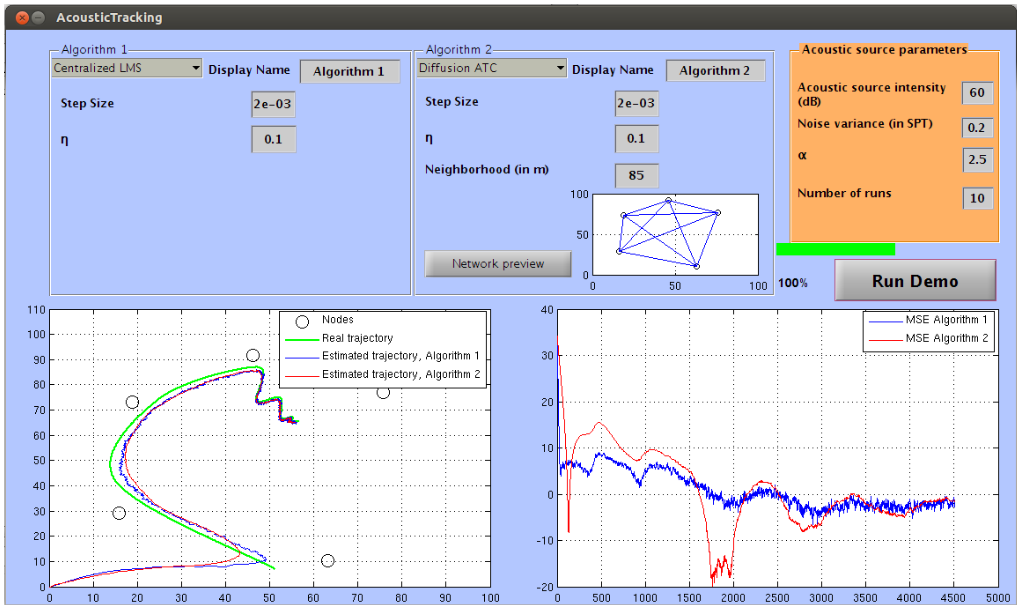
In this section we want to illustrate the effect of varying the different parameters associated with the acoustic source. Since the GUI does not allow the user to set different values of the acoustic parameters to each algorithm, we performed the simulations under the same network configuration and trajectory, but with variations in the acoustic parameters, in order to get a visual representation of these effects.

4.5.1 Effect of the value of the variance of the noise associated to the SPT

The value we assign to the noise term corresponding to the signal propagation time will determine how much we alter the measured SPT values. It is our way of simulating losses due to attenuation, diffusion, noise and presence of obstacles.

For this simulation we used a network of five nodes. We chose the Centralized LMS and Diffusion ATC algorithms as the first and second algorithms respectively. Both algorithms have the same step size and η parameter values. Figure 38 shows a simulation with the noise variance parameter set to 0.2.

On the other hand, Figure 39 illustrates the effect of increasing the variance value to 5. It can be seen that increasing the value of the SPT noise variance makes both algorithms obtain very similar estimations of the trajectory. Moreover, it can be seen that we obtain a worse behavior of the algorithms in terms of MSE. This is because the SPT term in the hybrid cost function (see Equation (20)) is noisier than the SPL one.


AcousticTracking

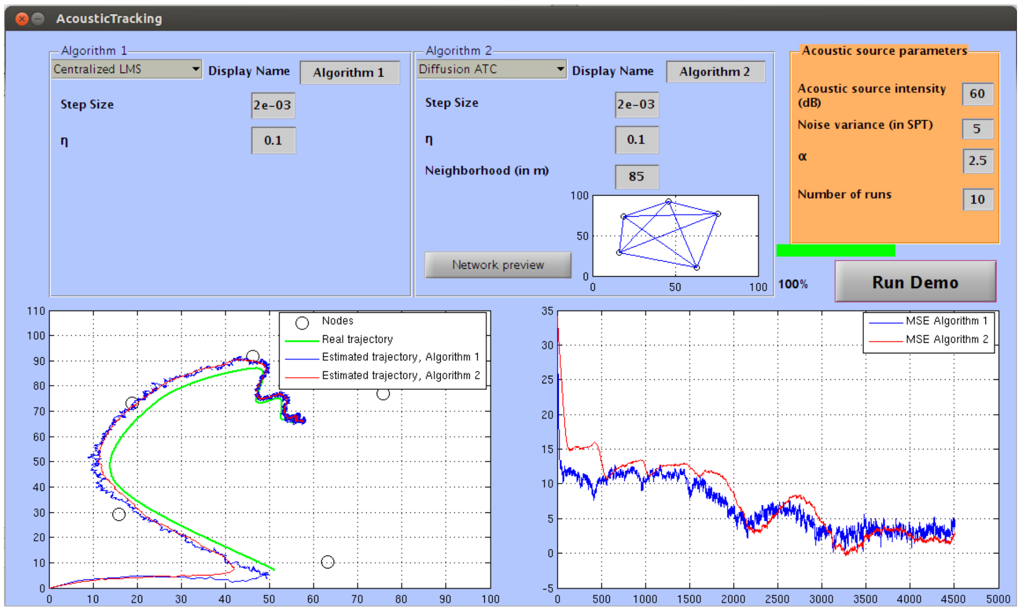
Algorithm 1: Centralized LMS — Display Name: Algorithm 1
Step Size: 2e−03
η: 0.1

Algorithm 2: Diffusion ATC — Display Name: Algorithm 2
Step Size: 2e−03
η: 0.1
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 2.5
Number of runs: 10

100%

Figure 38: Effect of the value of the variance of the noise associated to the SPT. Simulation ran with variance value 0.2.


AcousticTracking

Algorithm 1: Centralized LMS — Display Name: Algorithm 1
Step Size: 2e−03
η: 0.1

Algorithm 2: Diffusion ATC — Display Name: Algorithm 2
Step Size: 2e−03
η: 0.1
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 5
α: 2.5
Number of runs: 10

100%

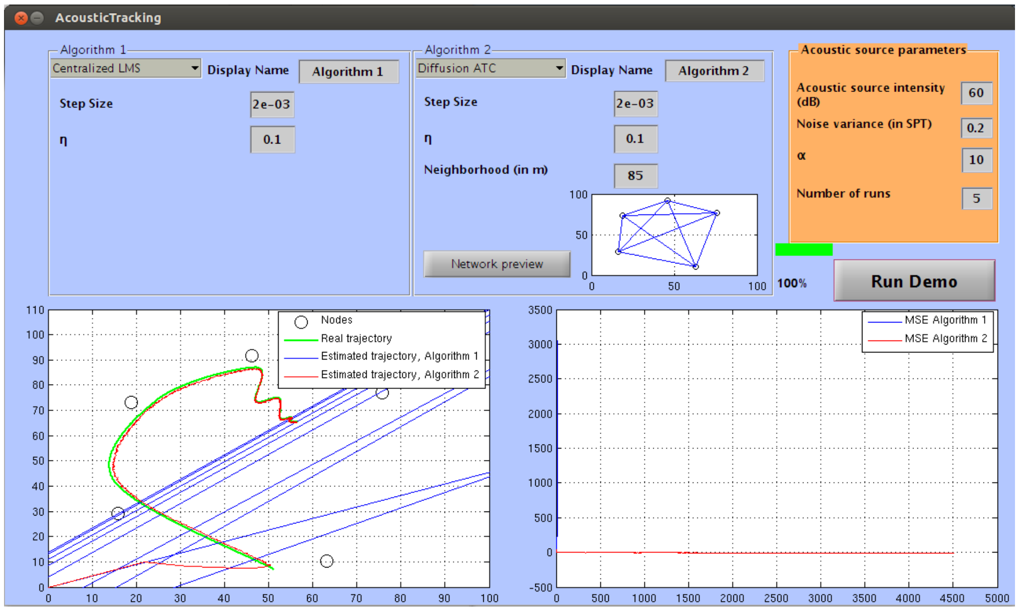
Figure 39: Effect of the value of the variance of the noise associated to the SPT. Simulation ran with variance value 5.

4.5.2 Effect of parameter α

Parameter α is the attenuation exponent corresponding to the log-distance path loss model.

For the illustration of the effect of this parameter we used the same network configuration as in the previous case, but for the first simulation we set parameter α to a very high value and, for the second one, to a very low value, leaving the remaining parameters unchanged. It can be seen in Figures 40 and 41 that, using this configuration, increasing the value of α is beneficial for the Diffusion ATC algorithm, whereas the Centralized LMS one tends to diverge.

For the simulation corresponding to Figure 41 we set parameter α to 0.1. From the absence of estimated trajectories in the axes we can assume that the estimated trajectories rely too far from the real one to even fit the GUI axes. It is obvious that working under these conditions does not give any significant results.


AcousticTracking

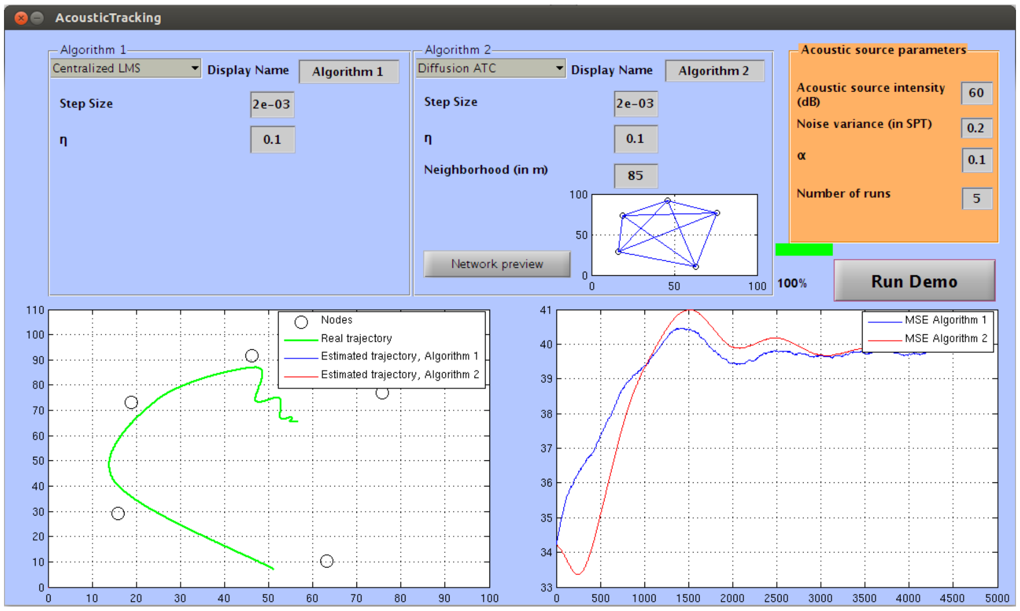
Algorithm 1
Centralized LMS
Display Name: Algorithm 1
Step Size: 2e-03
η: 0.1

Algorithm 2
Diffusion ATC
Display Name: Algorithm 2
Step Size: 2e-03
η: 0.1
Neighborhood (in m): 85

Acoustic source parameters
Acoustic source intensity (dB): 60
Noise variance (in SPT): 0.2
α: 10
Number of runs: 5

100%

Figure 40: Effect of parameter α when it is set to 10.



AcousticTracking

Algorithm 1: Centralized LMS — Display Name: Algorithm 1

Step Size: 2e−03

η: 0.1

Algorithm 2: Diffusion ATC — Display Name: Algorithm 2

Step Size: 2e−03

η: 0.1

Neighborhood (in m): 85

Acoustic source parameters

Acoustic source intensity (dB): 60

Noise variance (in SPT): 0.2

α: 0.1

Number of runs: 5

Figure 41: Effect of parameter α when it is set to 0.1.

We want to highlight that the values of α chosen for these simulations do not make physical sense. This term usually takes value $\alpha \approx 2$. In the theoretical, lossless, noiseless case, setting parameter $\alpha = 2$ in Equation (18) results in Equation (17) with $r_1 = d_0$.

5 Implementation of the diffusion algorithms over a real wireless sensor network

This section is dedicated to the implementation of the *Centralized LMS* diffusion algorithm in a real network of wireless noise sensors manufactured by ©Libelium company. It begins with a description of the technical characteristics of this company's sensor boards and the noise sensor that will be used for building the network in which we will implement the adaptive algorithms for the acoustic source localization.

Prior to that, we want to state that the reason for choosing ©Libelium's products for the implementation is that they follow the *Open Source* philosophy, as other companies such as Arduino™ or Raspberry™, providing detailed documentation and an on-line forum in which they offer support for developers. Moreover, they provide code examples of every functionality of every device they manufacture, making it relatively intuitive to make *ad-hoc* combinations.

5.1 Used Hardware

In this section we give a brief description of the technical characteristics of the hardware used in the implementation, which was provided by the University research group.

5.1.1 ©Libelium's Waspmote™

Waspmote™ is ©Libelium's advanced mote for Wireless Sensor Networks. It is an open source wireless sensor platform specially focused on the implementation of low-consumption modes to allow the sensor nodes (*motes*) to be completely autonomous and battery powered, offering a variable lifetime between 1 and 5 years depending on the duty cycle.

The omnidirectional *noise sensor* needed for the sound pressure level measurements is built directly on an additional board called **SensorCities**™, which needs to be connected on top of the Waspmote™ to work [21].

Communication between motes is achieved via radio using *XBee*™ radio antenna (see Figure 42).

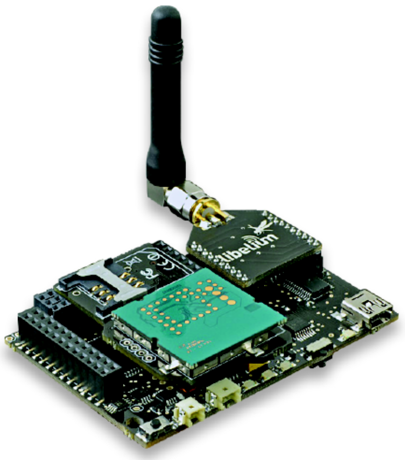


Figure 42: Waspmote™ with XBee™ radio antenna, taken from [20]

For the network implementation, the *XBee™-802.15.4* model is chosen, which has a transmission power consumption of 1 mW and a 500 m range of coverage [20].

5.1.2 ©Libelium's Meshlium-Xtreme™

As central node we used ©Libelium's *Meshlium-Xtreme™*, shown in Figure 43. It is a Linux router which works as the Gateway of the Waspmote™ Sensor Networks, which is connected to the sensor network as shown in Figure 44.

This device has a built-in database in which the values measured by the network sensors are stored. These values can be accessed either from a web interface or by sending messages to the network.

Figure 43: Meshlium-Xtreme™, taken from [22]

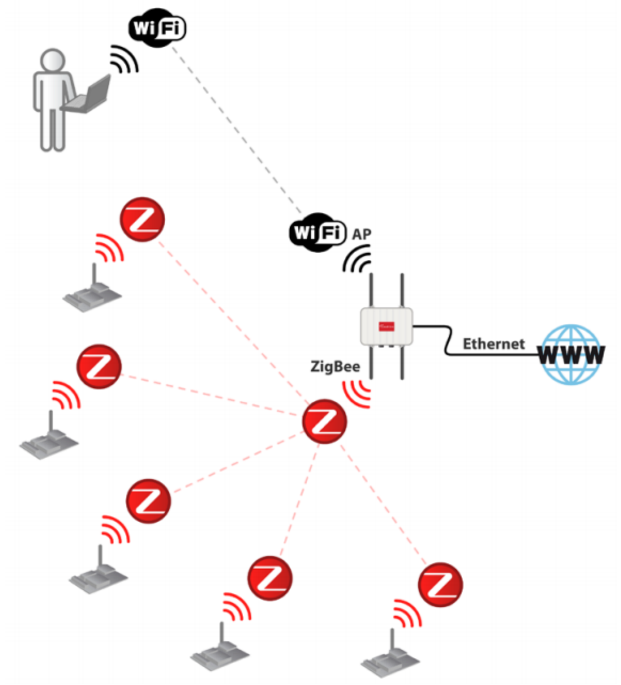


Figure 44: Wireless sensor network using Waspmote™ and Meshlium-Xtreme™, taken from [22]

5.2 Implementation of the code

This subsection focuses mainly on the implementation of the wireless sensor network in a real environment. The resulting scenario will be used as a reference for the final part of the Project: the comparison between the theoretical model and the real wireless network; which is still a work in progress. For the implementation to be successful we need to program the nodes following the desired network configuration. For simplification reasons we chose to implement a wireless noise-sensor network working in a centralized manner.

Waspmote™ devices are programmed in C^{++} using an SDE (Software Development Environment) provided by ©Libelium, which allows the user to upload the code to the motes. When programming for these devices one has to take into account that the motes operate in a *cyclical* manner. This means that there is a `loop()` function which contains the sequence of operations that the node needs to perform. Taking advantage of this cyclical behavior we defined the following sequences inside the loop:

1. **Initialization of the constant parameters:** During the `setup()` block of the execution, prior to the `loop()`, we initialize parameters h, α and η.

2. **Receive the previous estimation of the position:** In order to being able to calculate the error terms associated to the cost function in the Centralized LMS algorithm (see Section 2.2) each node needs to have access to the previous estimation of the position of the source. This is performed by the central node, which broadcasts each new estimation to the whole network after it has been computed. We start the process with a previous position estimation equal to $\mathbf{w}[0] = [0, 0]^T$.

3. **Measurement of the SPL:** Each node reads the SPL value measured by the noise sensor, which is given in dBA (acoustic decibels). The dBA units are used to denote the use of different weighting filters, used to approximate the noise sensor intensity measurement to the human ear's response to sound, in SPL.

4. **Calculation of the error terms and local gradient values:** For this calculation we must take into account that in a real scenario we will not have access to the *real* trajectory followed by the acoustic source, thus lacking the real distances of each node to every point in the trajectory. Therefore, the error terms are calculated using Equation (31) at each node. Since we only take the local error terms corresponding to SPL measurements, the local gradients are calculated using Equation (34) setting parameter η to zero.

5. **Sending the gradient values to the central node:** In order to obtain a *joint* estimation of the trajectory, each node sends its local gradient to the central node, which is responsible of computing the estimation of the position of the source following Equation (35). This is computationally easier than sending the SPL and SPT measurements directly to the central node, because while Meshlium™ is a very powerful router, it does not perform “complex” mathematical operations. Implementing such operations in the device might result in unwanted communication delays.

6 Conclusions and further work

In this section we will summarize the main conclusions reached along the document, most of which have already been discussed at the end of each of the previous sections, along with some possible future lines of work and research.

6.1 Conclusions

In this document we have dealt with the implementation of a software tool for the simulation of the acoustic localization and tracking problem using a wireless sensor network that works either in a centralized or distributed manner, which allows the user to compare the results obtained by diffusion networks working in a distributed manner and to compare them to the optimal theoretical result of the centralized solution. This tool allows the user to visualize the effect of altering different parameters over the same sensor network in order to simulate the behavior of a real wireless sensor network working with those settings, as well as the average MSE values associated to that configuration.

In order to give the inexperienced reader the needed theoretical background to understand both the terminology and the functionalities of this simulation tool, we covered the basics of Signal Processing, adaptive algorithms, Acoustics and the *Acoustic Source Localization* problem. Moreover, we defined the *hybrid cost function* associated with this problem, formed by the combination of the local costs associated to the SPL and SPT measurements obtained by each node in the network. We also saw the way of computing those error terms and the corresponding gradients and adaptation steps for the minimization of the said cost function using an adaptive diffusion algorithm: the *Centralized LMS* and *Diffusion ATC* algorithms; which estimate the coordinates $\mathbf{w}[n] = [x, y]^T$ that minimize the said cost function.

The implemented simulation tool guides the user through a number of steps that configure the simulation parameters:

- Setting the total number and placement of the nodes in the network.
- Generating the “real” trajectory to be estimated by the algorithms.
- Choosing the algorithms and setting the parameters associated to each one:

  - Step size, μ.
  - The SPT weight parameter, η.

- Setting the parameters associated to the acoustic source:
  - SPL measured at 1 m, L_{p_s}.
  - The variance of the noise associated to the SPT.
  - The value of the loss-path model exponent α.

- Setting the number of runs for the calculation of the average MSE.

For illustration reasons we ran several simulations in order to exemplify the functionality of the simulation tool. These simulations showed the results of modifying the parameters mentioned above, as well as a visual comparison between the different settings.

The results obtained with the implemented simulation tool show it can be very helpful to study the audio target tracking problem and ultimately for the design of sensor networks that can guarantee certain performance criteria.

We also implemented a real wireless sensor network using hardware manufactured by ©Libelium. This on-going work was not defined initially as one of the objectives of the project. Nevertheless, an initial implementation has been carried out, and this piece of work is currently under validation.

6.2 Possible improvements and future lines of work

The implemented simulation tool offers great functionalities. Nevertheless, there is still room for future development that would enhance the system and increase its value. After being tested by some classmates and colleagues from the research group, we saw that the implemented tool lacks two functionalities that would make the user experience much better:

1. Allow the user to configure different acoustic parameters for each scenario. This would be very helpful for comparing the same network settings using different acoustic parameters for each algorithm, instead of having to run additional simulations to obtain new results.

2. Allow a method to re-configure the number and placement of the nodes in the network, as well as to create a new trajectory, instead of having to re-run the simulation tool.

Regarding the implementation of the real wireless sensor network, there is still a lot of testing and data gathering left to do in order to be able to give proper results and conclusions, but it looks promising so far.

7 Estimated budget

This section is dedicated to the breakdown of the estimated budget needed for the development of this Project. We will differentiate between the implementation of the simulation tool and the implementation of the real wireless sensor network.

7.1 Budget for the implementation of the simulation tool

For the calculation of this budget we need:

- The price of a MATLAB® license.
- The average salary of a MATLAB® developer.
- The amortized cost of the computer.

The price of a MATLAB® license can be obtained from the MathWorks® web page: `http://www.mathworks.es`. Depending on the target use of the program, the price of a license can go from €35 to €2000. For the calculation of this budget we chose the "Home" license, which is €105.

The value of the average salary of a MATLAB® developer was obtained from ITJobsWatch (`http://www.itjobswatch.co.uk/`, June 17, 2014) and it was £45000 a year, which correspond to €56338.70 a year. Considering that it took 3 months to develop and test the simulation tool, we estimate a total of:

$$\frac{56338.70}{12} \cdot 3 = €14084.67$$

corresponding to the salary of the MATLAB® developer.

The amortized cost of the computer can be calculated as its depreciation. The straight-line method of calculating depreciation would be to divide the initial cost, i.e. its price, by the computer's "useful life". Since MATLAB® can run on most of today's computers, we chose to perform this calculation using a €1568.52 ($2129) Dell Precision T3610 computer (`http://www.pcmag.com/article2/0,2817,2372609,00.asp`, June 20, 2014), with an estimated useful life of 8 years, thus obtaining an amortized cost of:

$$\frac{€1568.52}{8\text{ years}} \cdot \frac{1\text{ year}}{12\text{ months}} \cdot 3\text{ months} = €49.01$$

Considering the values mentioned above, the approximate budget needed for the implementation of the simulation tool is:

Item	No. of items	Price (€)
MATLAB license	1	105
Salary of the MATLAB developer	1	14084.67
Amortized cost	1	49.01
Total		**14238.68**

7.2 Budget for the implementation of the real wireless sensor network

For the calculation of this budget we need:

- The price of ©Libelium's Waspmote™.
- The price of ©Libelium's SensorCities™ sensor board and noise sensor.
- The price of ©Libelium's Meshlium-Xtreme™.
- The average salary of a C^{++} developer.
- The amortized cost of the computer.

The respective prices of ©Libelium's Waspmote™, XBee™ and Meshlium-Xtreme™ can be obtained from ©Libelium's web page, `http://www.libelium.com/`, by either downloading the full product catalog or by choosing an official distributor and checking the prices. We chose to download the product catalog and found the following prices:

- Waspmote Digimesh-PRO SMA 2 DBI (XBee antenna included): €155.
- Meshlium-Xtreme Digimesh-PRO-AP: €690.

For the prices of the SensorCities™ sensor board and noise sensor we went to CookingHacks web site, `http://www.cooking-hacks.com` and found the following prices:

- Waspmote Smart Cities sensor board: €120.
- Noise sensor: €20.

The value of the average salary of a C^{++} developer was obtained from ITJobsWatch (`http://www.itjobswatch.co.uk/`, June 17, 2014) and it was £55000, which corresponds to €68858.42. Considering that the adaptation of the MATLAB® code to C^{++} for the implementation of the Centralized LMS algorithm took a month to complete, we estimate a total of:

$$\frac{68858.42}{12} = €5738.20$$

corresponding to the salary of the C^{++} developer.

The amortized cost of the computer can be calculated in the same way as in the previous case:

$$\frac{€1568.52}{8 \text{ years}} \cdot \frac{1 \text{ year}}{12 \text{ months}} \cdot 1 \text{ month} = €16.34$$

Taking into account that in order to obtain a good performance from the sensor network we need *at least* four nodes in the network, we will estimate the budget needed for a wireless sensor network formed by 10 nodes in order to cover a large area. Using the above values and assuming the physical installation of the nodes costs €300 we estimate a total budget for implementing the real sensor network of:

Item	No. of items	Price (€)
Waspmote Digimesh-PRO SMA 2 DBI	10	155
Meshlium-Xtreme Digimesh-PRO-AP	1	690
Waspmote Smart Cities sensor board	10	120
Noise sensor	10	20
Salary of the C++ developer	1	5738.2
Amortized cost	1	16.34
Installation of the physical components	1	300
Total		**9694.54**

References

[1] Adelbert W. Bronkhorst (2000), “The Cocktail Party Phenomenon: A Review of Research on Speech Intelligibility in Multiple-Talker Conditions”. Acta Acustica united with Acustica, Vol. 86, pp. 117-128.

[2] Oppenheim, Alan V., Ronald W. Schafer and John R. Buck (1999), “Discrete-time Signal Processing”. Englewood Cliffs, NJ: Prentice Hall.

[3] Orfanidis, Sophocles J. (1996), “Introduction to Signal Processing”. Englewood Cliffs, NJ: Prentice Hall.

[4] Madisetti, V., and Douglas B. Williams (1998), “Chapter 18: Introduction to Adaptive Filters”. *The Digital Signal Processing Handbook*. Boca Raton, FL: CRC.

[5] S.W. Rienstra, A. Hirschberg (2013), “An Introduction to Acoustics”, Extended and revised edition of IWDE 92-06.

[6] Eargle, John (1994), “Part Five: Psychoacustical Data.” *Electroacoustical Reference Data*. New York: Van Nostrand Reinhold.

[7] Carlos Fernandez Scola, Maria Dolores Bolaños Ortega (2010), “Direction of arrival estimation. A two microphones approach”. Masters Thesis, Blekinge Institute of Technology.

[8] Ashok Kumar Tellakula (2007), “Acoustic Source Localization Using Time Delay Estimation”. Masters Thesis, Supercomputer Education and Research Centre of the Indian Institute of Science.

[9] D.H.Youn, Nasir Ahmed and G.Clifford Carter (1982), “On Using the LMS Algorithm for Time Delay Estimation”. IEEE Transactions on Acoustics, Speech and Signal Processing Magazine, vol. ASSP-30, no. 5, pp. 798-800.

[10] Michal Mandlik, Vladimir Brazda (2011), “Sound source location method”. Pertner’s Contacts Magazine. Number 5, Volume VI, pp.197-204.

[11] Pourmohammad and Ahadi (2013), “N-dimensional N-microphone sound source localization”. EURASIP Journal on Audio, Speech, and Music Processing, vol. 27.

[12] R. Abdolee, S. Saur, B. Champagne, and A.H. Sayed (2013), “Diffusion LMS localization and tracking algorithm for wireless cellular networks”, in Proc. ICASSP, pp.4598-4602.

[13] J. Chen and A.H. Sayed (2011), “Diffusion Adaptation Strategies for Distributed Optimization and Learning over Networks”, presented at CoRR.

[14] N. Takahashi, I. Yamada, and A.H. Sayed (2010), "Diffusion least-mean squares with adaptive combiners: formulation and performance analysis", presented at IEEE Transactions on Signal Processing, pp.4795-4810.

[15] A.H. Sayed (2012), "Diffusion Adaptation over Networks", presented at CoRR.

[16] Fernandez-Bes, Jesus ; Azpicueta-Ruiz, Luis A. ; Silva, Magno T.M. ; Arenas-Garcia, Jeronimo (2013), “Improved least-squares-based combiners for diffusion networks”. IEEE Proceedings of the Tenth International Symposium on Wireless Communication Systems (ISWCS 2013).

[17] Ricard Heeks (1999), “Centralized vs. Decentralised Management of Public Information Systems: *A Core-Periphery Solution*”. Information Systems for Public Sector Management. Working Paper Series, no. 7, pp. 4-14.

[18] G. Coulouris, J. Dollimore, and T. Kindberg (2002), “Distributed systems - concepts and designs (3. ed.)”, presented at International Computer Science Series, pp.1-772.

[19] Yong-Eun Kim, Dong-Hyun Su, Chang-Ha Jeon, Jae-Kyung Lee, Kyung-Ju Cho and Jin-Gyun Chung (2011), “Sound Source Localization Method Using Region Selection”, Advances in Sound Localization, Dr. Pawel Strumillo (Ed.), ISBN: 978-953-307-224-1, InTech, Available from: http://www.intechopen.com/books/advances-in-sound-localization/sound-source-localization-method-using- region-selection.

[20] ©Libelium Comunicaciones Distribuidas S.L. (2013), “Waspmote Datasheet”. Document version: v4.4, pp. 2-3.

[21] ©Libelium Comunicaciones Distribuidas S.L. (2012), “SmartCities technical guide”. Document version: v0.5, p. 26.

[22] ©Libelium Comunicaciones Distribuidas S.L. (2013), “Meshlium Xtreme Datasheet”. Document version: v4.2.