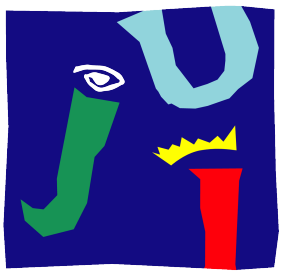


UNIVERSITAT JAUME I

ESCOLA SUPERIOR DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

MÁSTER EN INGENIERÍA INDUSTRIAL

Desarrollo de un sistema electrónico para la captación de imágenes de un solo píxel mediante métodos acústicos

Trabajo final de máster

Autor :
Marc Martí Sabaté

Director :
Ignacio Peñarrocha Alós

Coodirector :
Daniel Torrent Martí

Castellón, Julio 2019

Abstract

This project explains the implementation in an integrated and autonomous electronic device of an acoustic imaging system. The device belongs to the so called single pixel devices; that is to say, this device is able to reconstruct an image with spatial resolution using a single sensor or transducer. The key point of these techniques is the ability to modulate the source field and then recover the signal sequentially or by multiplexing in frequency (as it happens in this case). The image is finally reconstructed through a computational algorithm.

All along this project you will be able to discover the physics equations underneath the problem, the image reconstruction algorithm and its behavior, and its implementation in a real system and environment, which is the main part of the project. Considerations and restrictions of applying a mathematical model to the real world will appear and constraint the solution, forcing to take decisions such as the components selection.

Simulation results will be given and discussed, validating the reconstruction algorithm. Moreover, experimental measurements will be provided and will lead the discussion to potential mistakes and ways to improve the performance of the device.

Acknowledgements

I would like to express my gratitude to everyone who has helped in the realization of this project.

Firstly, I acknowledge the contribution of Daniel Torrent Martí. It is his initial idea what has turned into this project, and I am so grateful to him for giving me the opportunity of developing his idea.

I am also indebted to Ignacio Peñarrocha Alós, my academical supervisor at Universitat Jaume I. His advice in electronics and his point of view have been necessary to achieve the engineering part of the project.

Thanks are also due to José Martínez Sotoca and Vicente Javier Traver Roig, who were also involved in the project from the beginning, for all the information and advice about machine learning algorithms. I would like to thank in general all the members at GROC, where the project has taken place.

Contents

List of Figures

Abbreviations

ADC: Analog to Digital Converter
CAD: Computer-Aided Design
COB: Chip On Board
DAC: Digital to Analog Converter
DSP: Digital Signal Processing
FEM: Finite Element Method
FFT: Fast Fourier Transform
IDE: Integrated Development Environment
LFSR: Linear Feedback Shift Register
MSE: Mean Squared Error
MAE: Mean Absolute Error
MOSFET: Metal-oxide-semiconductor Field-effect Transistor
OS: Operating System
PCB: Printed Circuit Board
PLA: Polylactic Acid
PRNG: Pseudorandom Number Generator
PWM: Pulse Width Modulation
RAM: Random Access Memory
ReLU: Rectified Linear Unit
SAW: Surface Acoustic Wave
SPI: Serial Peripheral Interface
VAT: Value-Added Tax

The project follows the parts indicated by the norm UNE-EN 157001:2014, but their names have been translated. Therefore, the relationship between Spanish norm parts and English parts in the project are the following:

Report: Memoria
Appendices: Annexos
Specifications: Pliego de condiciones
Budget: Presupuesto

Part I

Report

Chapter 1

Objective

In these days, sensors are more sophisticated and include more information than they did twenty years ago. Moreover, even if they are more complex, there is a real need of getting more information with less resources, so as to cheapen production costs and appeal clients.

This project is framed in this context. Imaging is just a way of sensing with spatial resolution, and acoustic image already exists in fields such as medicine. The main objective of the project is to implement an acoustic imaging system based on single-pixel cameras in an embedded system, which will integrate all the steps; from emitting with a loudspeaker and receiving an acoustical signal with a microphone, to processing the signal, reconstructing the image and showing the image on a screen.

Instead of using an array of transducers, as done in sonar imaging systems, the key point of this system is that a two dimensional image is reconstructed using a single microphone, thanks to a structured plate that modulates the acoustic wave. Thus, this system is conceived to be compact and small, avoiding the usage of an array of sensors or huge processing units such as a computer. Its reduced size, low cost (compared with other acoustical imaging systems) and autonomy makes it more attractive than other imaging systems. Furthermore, as all the information is gathered in an embedded system, it is not difficult to communicate this unit with other computing systems, as intended nowadays with Industry 4.0.

Chapter 2

Scope

This project must be considered as a proof of concept of what can be achieved with this technology. Working in the audible frequency range will mean that the structured plate will define a spatial resolution of few centimeters. A 24 pixel image with a pixel distance of centimeters is a really low resolution image and it is not useful for a real application.

Nevertheless, proving that the solution is attainable for a low resolution system is valid to infer that a similar product is feasible working in ultrasound range and spatial resolution of micrometers or even nanometers. This scale is much more interesting for researchers. For example, it could work as a detector of molecule precipitation in a medium. For sure, the emitter and the receiver would be changed in order to work in ultrasound range, but the rest would be the same.

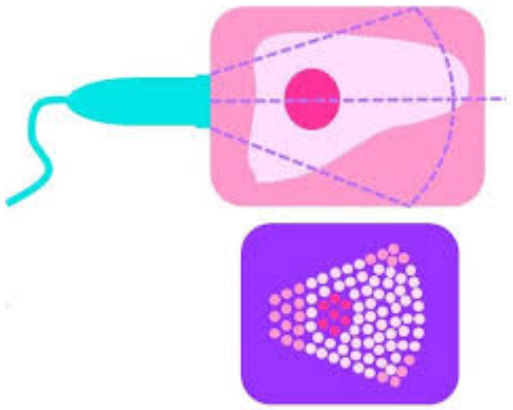
Not only in research work, but this system can also be interesting for other applications. In fact, most smartphones use SAW sensors in their screens, which use electronic properties of the components to detect pressure changes in the surface. The system in this project could be applied the same way as the SAW sensors, substituting them and also simplifying them. SAM (Scanning Acoustic Microscopy) is a really well-known technique that is used to detect cracks and defects in small electronic components. The technique presented here could also be useful in this field, cheapening the expensive microscopes that are used today.

Chapter 3

Background

Acoustical imaging is not a new topic. Acoustic waves have been used since the beginning of the XXieth century in order to get information about objects. The most popular application of acoustical imaging is possibly echography (sonography or medical ultrasonography). Despite this, neither echography is the only acoustic technique, nor biomedicine is the only application. There are several techniques, such as SAM (Scanning Acoustic Microscopy) or beamforming, that are applied in microscopy, underwater imaging, or failure analysis.

As for medical ultrasonography, it consists in a piezoelectric transducer (or array of transducers) that emits an acoustic signal. The waves travel into the body and come into focus at a desired depth. Depending on the characteristics of the material (reflectivity, scattering), the wave will suffer different phenomena. The same transducer acts as a pressure sensor (a piezoelectric transducer is reversible), recovering information about the scattered wave. Finally, the image is formed by computing the time it took to the echo to get back to the transducer ("time of flight"), and how strong is this echo. Thus, knowing the time, the spatial position can be stated, and the amplitude of the echo is the pixel's intensity. Echo's intensity is not the only parameter that is able to be imaged: there are several modes, and each mode images a different tissue property. Mode A is for amplitude, but it is possible to do brightness image (mode B), motion (mode M, video can be recovered) using different types of Doppler effect, or receiving the information in a different frequency that the emitted one (harmonic mode). Concerning spatial resolution (real distance between pixels), there are some parameters that fix it; the sampling rate and the frequency fix axial resolution, (if distance between two scatterers is lower than wavelength, the system will not be able to show two different points in the image). Parameters that affect lateral resolution are beam's width, frequency and scan density. Beam's width is usually a function of the transducer design (distance between piezoelectric cells). One single measurement allows to get a two dimensional image, however, several transducers are needed.



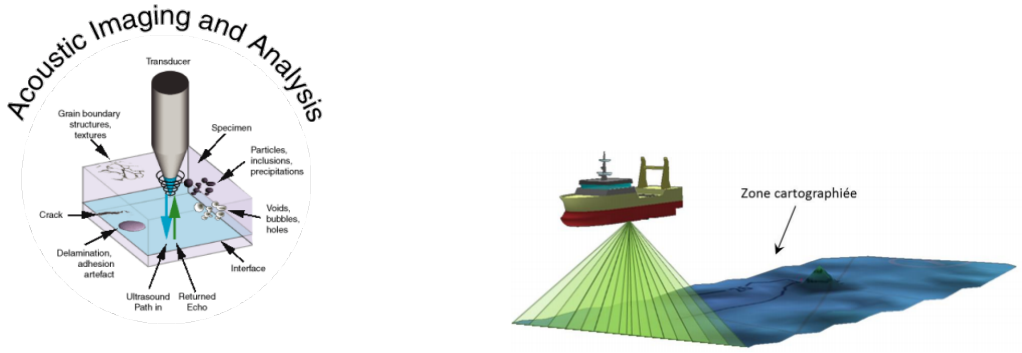
(a)

(b)

Figure 3.1: *Medical ultrasonography. At the left, its principle (mode B). At the right, an example of image. Both pictures have been taken from Asociación Española de Pediatría.*

The acoustic microscope, made up in 1936, is an electronic device that allows to do imaging of small objects using acoustic waves. The most popular technique with this microscope is SAM (Scanning Acoustic Microscopy). This technique focuses directly a sound wave from a transducer at a small point on a target object. Then, sound hitting the object is either scattered, absorbed, reflected or transmitted. Here, a transducer placed in a given position (whereas in medical ultrasonography it is the same transducer, here it can be another transducer placed in a different position) gets information about the scattered wave, recovering the presence of objects or boundaries. Again, "time of flight" gives the distance the sound has covered between emission and reception. After measuring once, the object is slightly moved, and the measurement is repeated, scanning the object. Contrast in the image is based either on the object's geometry or its material composition. One benefit of this technique is that the sound wave can be focused at different depths, not only producing surface images of the object, but also images from the inner part. As a limitation, it takes a long time to process the signal and also to measure, as it is a raster scan technique. Acoustic microscopy is not only important in the medical field, but also in the microelectronics industry. It is used to determine the quality of chips and electronic devices. Failure analysis is one of the most important applications, allowing the companies to save money by detecting bad products. Resolution in this technique is defined by the mechanical properties of the displacement system of the object. Only one (or maybe two, depending on the configuration) transducer is needed, but several measurements must be made in order to get a two dimensional image.

Concerning underwater acoustical imaging, sonar system is a well-known device to obtain images. Even if it was conceived for military purposes, such as ship or submarine detection, nowadays sonar systems are also used by researches to map the sea bottom, or identifying fish banks. Similar to the other techniques, piezoelectric transducers emit an acoustic wave, and they also receive the medium response. Time of flight is measured to find position in space, and the wave amplitude is related to the pixel's interference. There is an important difference between medical imaging and underwater imaging: the frequency range. Medical imaging needs ultrasound or even higher frequencies, while underwater imaging needs kilohertz or lower frequencies. Three dimensional images can be obtained by moving the emitter (usually moving the ship), or by using beamforming techniques. Resolution is fixed by the same parameters as for the medical ultrasonography (sampling rate and distance between transducers). One single measurement produces a two dimensional image; however, an array of transducers is needed.



(a) (b)

Figure 3.2: *At the left, scanning acoustic microscopy principle. At the right, a sonar imaging system.*

The proposed system changes dramatically the way acoustic images have been done up to these days. "Time of flight" and temporal amplitude are no longer regarded, and spectral content is the only parameter worth measuring. A structured plate acts as a sound wave modulator, changing the properties of the incident wave. Just as the acoustic wave is scattered and reflected inside the body in medical sonography, the sound wave interacts with the structured plate. Thus, when an object is placed just on top of some part of the structured plate, the response obtained in the microphone is changed. As each part of the structured plate modulates in a different way the sound wave, the difference between both signals (with object and without it) shows up which parts of the structured plate have been covered, getting a binary image of the surface.

Each pixel can be seen as a detected or not detected object in this position, which is much less information than in previous explained imaging techniques. However, there is a series of features that makes this system appealing compared to the others. It is a single-pixel system; this is a unique sensor system that is able to reconstruct a two dimensional image (both medical sonography and underwater imaging need an array of transducers to get two dimensional information). It is single-shot; provided that acoustic devices are broadband, this system profits to get all the spectral information in one measurement (SAM needs several measurements to cover the same section). Furthermore, the spatial resolution is no longer fixed by mechanical properties (SAM) or transducers' distance (underwater imaging), it is now fixed by the structured plate, which allows to get really interesting spatial resolutions.

Chapter 4

Standards

In this chapter, several documents related to the content of the report will be presented, so as to get an idea of what is involved in the project.

4.1 Standards

This section states the regulations that are followed in the project:

- **UNE-EN 157001:2014**: General criteria for the drawing-up of the documents which make up a technical project.
- **UNE-EN 60097:1996**: Grid systems for printed circuits.
- **UNE-EN 60249-1:1997**: Base materials for printed circuits. Part 1: test methods.
- **UNE-EN 62326-1:2004**: Printed boards. Part 1: generic specification.

4.2 Software

Depending on the part of the project, a different programming software has been used. For physical simulations and data set generation, MATLAB has been chosen. The machine learning algorithm has been implemented in Python, using Spyder's environment. The PCB design has been done using Eagle, from Autodesk company. Finally, the microcontroller programming environment is Arduino IDE.

MATLAB (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment. It allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages than M (programming language used by MATLAB).

Even if MATLAB is intended primarily for numerical computing, additional packages like Simulink add graphical multi-domain simulation and model-based design for dynamic and embedded systems or GUIDE, which allows to create interfaces between user and machine. Moreover, it has several toolboxes,

sets of functions put together following their application. For example, there exists signal processing toolbox, machine learning toolbox or optimization toolbox. Nowadays, MATLAB is largely used in various backgrounds of engineering, science and economics.

In the project, as the physical equations of the model have been established, raw MATLAB is enough to simulate its behavior. Thus, no other toolbox is needed.

Spyder is a scientific environment written in Python, designed by and for scientists, engineers and data analysts. It offers a combination of advanced editing, analysis, debugging and profiling functionalities. All code used by Spyder is open-source and completely free. Many toolboxes are available, and it is easy to get with and start using them. **Numpy** is the fundamental package for scientific computing with Python; **Matplotlib**, which is a Python 2D plotting library; **Keras** is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK or Theano.

In this project, an Artificial Neural Network is going to be implemented, using **TensorFlow** as the main package for neural networks. TensorFlow is in fact and end-to-end open source platform for machine learning. It is not exclusive for Python programming language, it can also be found for JavaScript. As said before, Keras can be run on top of TensorFlow, allowing to use both libraries with only installing one of them. Numpy and Matplotlib packages will also be used to implement the algorithm and visualize the results.

The **Arduino Software (IDE)** is an open-source C/C++ coding platform that makes it easy to write code and upload it to the board. It is able to run on Windows, Mac OS X and Linux. The environment is written in Java. With the help of third party cores, it is even possible to code and upload programs for other vendor development boards other than Arduino. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code requires two basic functions; starting the sketch and the main program loop (*setup()* and *loop()*).

EAGLE PCB Design Software, from Autodesk company, is a scriptable electronic design automation application that allows schematic capture, printed circuit board layout, autorouter and computer-aided manufacturing features. It stands for Easily Applicable Graphical Layout Editor. In this project, EAGLE has been used to design the PCB containing the electronics that will be needed to process the signal before entering the microcontroller.

Bibliography

[1] Gras microphone description. https://www.gras.dk/products/measurement-microphone-sets/constant-current-power-ccp/product/143-46be.

[2] Gras power supply description. http://www.gras.dk/products/product/222-12AL.html.

[3] Gras preamplifier description. https://www.gras.dk/products/product/204-26cb.html.

[4] Kemo l10 piezoelectric. https://www.kemo-electronic.de/en/Car/Speaker/L010-Piezo-Loudspeaker.php.

[5] Pla premium black wire. https://filament2print.com/es/pla-premium/745-pla-premium-negro.html.

[6] Why and how to cross validate a model? https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f. Accessed: 2019-05-25.

[7] Roberto Alejo Eleuterio. Análisis del error en redes neuronales: Corrección de los datos y distribuciones no balanceadas. *TDX (Tesis Doctorals en Xarxa)*, sep 2010.

[8] Léon; Brillouin. *Wave propagation in periodic structures; electric filters and crystal lattices*. 1946.

[9] B. Widrow et al. Adaptive "Adaline" neuron using chemical "memistors". Technical report, Stanford Electron. Labs, Stanford, CA, 1960.

[10] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 2017.

[11] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[12] Yan Huang, Wei Wang, Liang Wang, and Tieniu Tan. Multi-task deep neural network for multi-label learning. In *2013 IEEE International Conference on Image Processing, ICIP 2013 - Proceedings*, 2013.

[13] Halbert White Kurt hornik, Maxwell Stinchcombe. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

[14] J; Henderson D; Howard R; Hubbard W;Jackel L LeCun, Y; B; Denker. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1989.

[15] P. A. Martin. *Multiple Scattering: Interaction of Time-Harmonic Waves with N Obstacles*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2006.

[16] Andrew Maxwell, Runzhi Li, Bei Yang, Heng Weng, Aihua Ou, Huixiao Hong, Zhaoxian Zhou, Ping Gong, and Chaoyang Zhang. Deep learning architectures for multi-label classification of intelligent health risk prediction. *BMC Bioinformatics*, 2017.

[17] Min-Ling Zhang and Zhi-Hua Zhou. A k-nearest neighbor based algorithm for multi-label classification. In *2005 IEEE International Conference on Granular Computing*, 2005.

[18] Michael A. Nielsen. Neural networks and deep learning, 2018.

[19] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[20] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[21] Stuart Russel and Peter Norvig. *Artificial intelligence—a modern approach 3rd Edition.* 2012.

[22] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[23] Daniel Torrent. Acoustic anomalous reflectors based on diffraction grating engineering. *Physical Review B*, 2018.

[24] Hanazawa T. Hinton G. Shikano K. Lang K. J. Waibel, A. Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions*, 1989.

[25] P.J. Werbos. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.* PhD thesis, 1970.

[26] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, 1974.

[27] Min Ling Zhang. Ml-rbf: RBF Neural Networks for Multi-Label Learning. *Neural Processing Letters*, 2009.

[28] Min Ling Zhang and Zhi Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

Chapter 5

Design requirements

In this chapter, system design requirements are going to be discussed, in order to present what the system should be able to do and be able to compare it with the final product performance.

Design requirements can be divided in two different classes: those concerning hardware and those concerning software. The former will affect component selection, such as microcontroller, loudspeaker, microphone and so on, while the latter affects also the microcontroller and the code inside it.

On the one hand, **hardware requirements** are:

- System's **size** must be as small as possible. All system's electronics should be placed in a unique PCB, except for the microphone and the loudspeaker, which must be placed in the center and at a corner of the structured plate.
- **Consumption**. In order to get an autonomous system, its power supply should be sort of a battery. This implies that consumption must be reduced; otherwise, power supply size grows or even a transformer and rectifier is needed to plug-in the system.
- The **structured plate** should be a squared plate $20cm$ long. To get more details about the metastructure design, read **Appendix D**.
- System's cost should be lower than other acoustical imaging systems.
- The system is intended to do audible frequency imaging. Thus, system's **operating frequency** must be in the interval $f_i \in [0, 20]\, kHz$. All components must admit working in this bandwidth.

On the other hand, **software requirements** are:

- All signal processing must be done in the **microcontroller**. A computer can not be an element of the system.
- The system must be able to keep in its **memory** the trained machine learning algorithm, that is to say, all the layers with their weights that come from Python's algorithm.
- Software code should provide **noise generation**, as well as **reading microphone's signal**, signal processing (**Fourier transform**), **neural network prediction**, and **serial communication with a screen**.

- The neural network algorithm needs as input a double precision (32 bits) 997 value vector, representing spectral coefficients linearly distributed between $0Hz$ and $20kHz$.

Chapter 6

Solution analysis

In this chapter, different proposals will be discussed. Based on different criteria that will be exposed to the reader, an appropriate component selection will be done, and other solutions will be discussed in order to show benefits and drawbacks compared to the chosen one. As it will be seen during the chapter, there is no unique solution; there is always a trade off between system performances. Depending on the requirements' ranking, a solution will be considered the best one.

6.1 Hardware selection and design

6.1.1 Starting point: already selected material

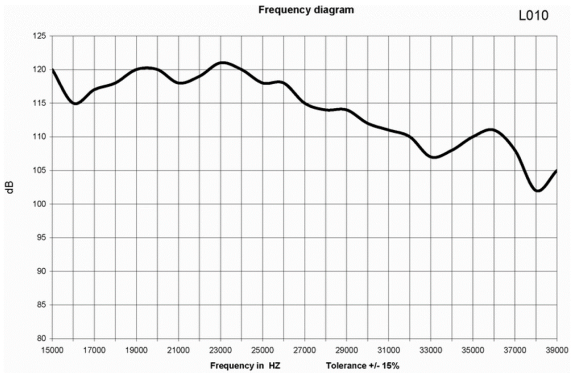
Before starting selecting components for the system, a recapitulation must be done in order to take into account those elements that have been already selected or that are available. Those being considered, the other elements should be selected according to them and their specifications.

The **microphone** is one of these already selected components. In fact, it is a GRAS 46BE 1/4″ CCP Free-field Standard Microphone Set [1]. It is a general acoustics diagnostics microphone with a frequency range going from $4Hz$ up to $80kHz$, a dynamic range from $35dB$ to $160dB$, and a sensitivity of $4mV/Pa$. This microphone has its own preamplifier, which is GRAS 26CB 1/4″ CCP Standard Preamplifier with Microdot Connector [3]. The preamplifier has, obviously, a larger frequency range than the microphone, otherwise it would cut some frequencies received by the microphone. Its frequency range goes from $2.5Hz$ up to $200kHz$, it produces $1.8\mu V$ noise, and a gain of $-0.35dB$. Finally, both the microphone and the preamplifier need a power source. This is accomplished by GRAS 12AL 1-Channel CCP Power Module with A-weighting filter [2]. This power supply acts as a current source, giving $4mA$ of constant current to the microphone and the preamplifier. Its voltage is varied to ensure the constant current, and the output signal is a differential voltage. This output signal has its own offset, and the gain is not well adapted to enter directly a microcontroller. Therefore, some signal processing will be done in order to fix this inconvenience.

Figure 6.1: *Selected microphone, preamplifier and power source.*

Another selected component is the **loudspeaker** that is going to be used. It is a Kemo L10, a piezoelectric loudspeaker from Kemo Electronic [4]. Its frequency range goes from $2Hz$ up to $60kHz$, and it accepts up to $16Vp-p$ tension on its terminals.

(a)



(b)

Figure 6.2: *Piezoelectric loudspeaker. The graph at the right is the magnitude Bode diagram.*

As the lowest component frequency is $60kHz$, the system should be able to work up to this frequency. However, one or two electronic circuits will be needed to process signal, and operational amplifier and transistor's cutoff frequency must be taken into account. Furthermore, selected microcontroller also affects in this working range, as the ADC clock defines the highest frequency the microcontroller is able to get. The idea in the project is to modulate the acoustic field up to $20kHz$; then, analogic components satisfy this condition, and the microcontroller will be chosen considering this constraint.

6.1.2 Microcontroller's selection

Several parameters must be considered before selecting a microcontroller for a given application. Some of them are: cost, consumption, input/output need, memory, word size, need for low level libraries

(peripheral configuration), programming graphics interface, learning difficulty or need for a board design. Next, each of this points are going to be discussed, considering application's need, but also features found in the market.

Cost

As for the cost, budget is always important when developing a project. Despite the fact that other components, as the microphone or its power supply, are much more expensive than common market microcontrollers, this component will be chosen minimizing the cost, but ensuring that the other features are accomplished. To get an idea of commercial prices, see **Table 6.1**. It must be said that, as a proof of concept, a **COB** will be chosen, instead of only the microcontroller. The reason is that it is simpler to use and to program.

Boards	Microcontroller	Operating Voltage/s (V)	Digital I/O Pins	PWM Enabled Pins	Analog I/O Pins	DC per I/O (mA)	Flash Memory (KB)	SRAM (KB)	EEPROM (KB)	Clock (MHz)	Length (mm)	Width (mm)	Cable	Native Network Support
Uno	ATmega328	5	14	6	6	20	32	2	1	16	68.6	53.4	USB A-B	None
Leonardo	ATmega32u4	5	20	7	12	40	32	2.5	1	16	68.6	53.3	micro-USB	None
Micro	ATmega32u4	5	20	7	12	40	32	2.5	1	16	48	18	micro-USB	None
Nano	ATmega328	5	22	6	8	40	32	2	0.51	16	45	18	mini-B USB	None
Mini	ATmega328	5	14		6	20	32	2	1	16	30	18	USB-Serial	None
Due	Atmel SAM3X8E ARM Cortex-M3 CPU	3.3	54	12	12	800	512	96	×	84	102	53.3	micro-USB	None
Mega	ATmega2560	5	54	15	16	20	256	8	4	16	102	53.3	USB A-B	None
M0	Atmel SAMD21	3.3	20	12	6	7	256	32	×	48	68.6	53.3	micro-USB	None
Yun Mini	ATmega32u4	3.3	20	7	12	40	32	2.5	1	400	71.1	23	micro-USB	Ethernet/Wifi
Uno Ethernet	ATmega328p	5	20	4	6	20	32	2	1	16	68.6	53.4	Ethernet	Ethernet
Tian	Atmel SAMD21	5	20	12	0	7	16000	64000	×	560	68.5	53	micro-USB	Ethernet/Wifi
Mega ADK	ATmega2560	5	54	15	16	40	256	8	4	16	102	53.3	USB A-B	None
M0 Pro	Atmel SAMD21	3.3	20	12	6	7	256	32	×	48	68.6	53.3	micro-USB	None
Industrial 101	ATmega32u4	5	7	2	4	40	16000	64000	1	400	51	42	micro-USB	Ethernet/Wifi
Uno Wifi	ATmega328	5	20	6	6	20	32	2	1	16	68.6	53.4	USB A-B	Wifi
Leonardo Ethernet	ATmega32u4	5	20	7	12	40	32	2.5	1	16	68.6	53.3	USB A-B	Ethernet
MKR1000	Atmel SAMD21	3.3	8	12	7	7	256	32	×	48	64.6	25	micro-USB	Wifi

Figure 6.3: *Arduino benchmark*

	Devices			
Platform	**Arduino**	**Propeller**	**Beagle Board**	**Raspberry Pi**
Software				
Operating System	None	None	Android, Linux, Windows CE, RISC OS	Linux, RISC OS
Dev. Environments/Toolkits	Arduino IDE, Eclipse	Propeller/Spin	Eclipse, Android ADK, Scratchbox	OpenEmbedded, QEMU, Scratchbox, Eclipse
Programming Language	Wirign-based (C++)	Spin/Propeller Assembly	Python, C, etc.	Python, C, possibly BASIC
Architecture	8 Bit	32 Bit	32 Bit	32 Bit
Hardware				
Processor	ATMEGA328	P8X32A-M44	TI DM3830 (ARM)	BCM2835 (ARM)
Speed	$16MHz$	$12MHz$	$720MHz$	$700MHz$
RAM	2 Kbyte	32 Kbyte	256 MB	256 MB
ROM	32 Kbyte	32 Kbyte	256 MB Flash	SD
I/O (various protocols)	14	32	22	8
ADC	6	None	internally used	internally used
USB	None	None	1x2.0	2x2.0
Audio	None	None	Stereo In/Out	Stereo out, In w/USB mic
Video	None	VGA,NTSC or PAL	DVI-D, S-Video	HDMI, NTSC or PAL
Misc.	Many shields available for added capability	8 processors for parallel tasking	SD/MMC, RS-232, JTAG, USB OTG, LCD	SD, 10/100 Ehternet, JTAG
Cost				
	29.95 $	49.99 $	199.95 $	35.00 $

Table 6.1: *Processor benchmark.*

Figure 6.3 makes a comparison between Arduino products, while **Table 6.1** compares Arduino Uno to other COB products, such as Raspberry Pi. Beagle Board or Raspberry Pi are not microcontrollers, they are microprocessors, much more powerful than a microcontroller, but, at the same time, more expensive and with many tools that will not be needed. For the purpose of the project, no OS is needed, so a microcontroller is preferred to a microprocessor. Furthermore, it is easier to learn and there is no need to spend time installing an OS. There are several microcontroller companies, but, as said before, a COB is preferred, as it is easier to start programming and connections are already prepared.

Word's width

Signal processing is a key point in the system. Microcontroller must be able to perform a Fourier transform of the temporal signal, as well as apply the different layers of the artificial neural network to provide a prediction of the image reconstruction. All neuron weights have been computed in *Python* using a *float32* resolution, that is to say, in floating point four byte representation. Therefore, the spectra should be computed with the same resolution, in order to entry the neural network. A word width of 32 bit is then compulsory to avoid losing accuracy. Nevertheless, another option is to work in fixed point representation until the entry of the neural network, and then cast into float point representation to make the reconstruction. This option can be necessary in case of a few RAM microcontroller or an analogical to digital converter that works in fixed point representation. Depending on the chosen microcontroller, one option or the other will be performed.

Input/Output

Concerning inputs and outputs, there are three main peripherals to control. First of all, the loudspeaker, in order to produce the incident acoustical wave. Thus, there should be one output pin. However, depending on the signal that is going to be emitted, a digital output pin or an analogical output pin will be needed. It is not common to find microcontrollers with analogical outputs; it is more usual to find PWM outputs, which play a similar role, but there are still differences between them. Digital outputs, instead, are easy to find, and almost every microcontroller has several of them. There is also a screen to be connected to the system so as to project the reconstructed image. If it is the case of a screen, a HDMI or VGA connection is needed. If the element to connect is a LCD screen, then serial communication is enough.

Finally, there is an analogical signal to read, which corresponds to the signal detected by the microphone. Then, the microcontroller must have an analogical input with its ADC. If there were no analogical input, an external ADC could be bought and connected between the microphone and the microcontroller. However, this would enhance the system's size, and also the price. Consequently, it is better to find a microcontroller with analogical inputs.

Consumption

Consumption is also important, considering that one of the project's goal is making an autonomous system. If consumption is low enough, a battery could be used as a power supply for the system. Elsewhere, a converter should be added and the system should be plugged in, reducing its autonomy. For example, Arduino's family is supplied with $5V$ or $3.3V$, which is easily feasible with small batteries. Operational amplifiers are not as kind; depending on the model, they can need symmetric or asymmetric power supply up to $\pm 15V$, which would mean using several batteries in series, or a little transformer with

some power electronics to control the voltage level.

Memory

Memory is another important parameter to manage. Microcontrollers are not powerful devices in terms of memory space. There are two main memories to consider: program memory and RAM. Common values for microcontroller's RAM are $2Kbytes$ or similar. Thus, what it is in memory during the execution must be managed carefully to avoid memory overflow.

First of all, neuron weights must not be in RAM. They should be saved either on program memory or in an external SD card. An estimation of Artificial Neural Network size is $8MB$. (Python's file is bigger, but its network structures are much more complex that what it is needed for just predicting; $8MB$ correspond to just the *float32* weights). There are no microcontrollers with such program memory space (common sizes are $256Kbytes$ or $512Kbytes$); an external memory must be used. Arduino has several shields available to plug directly; one of them being an external SD card memory.

As an input to the neural network, a 1000 value spectrum is needed. The system needs to compute a 1000 value Fourier transform (microcontroller algorithms for Fourier transforms as FFT are not as generalized as computer ones; input values must be a power of 2). A 1024 *float32* vector is big enough to overflow RAM from cheaper microcontrollers. For example, ATMEL 328P (Arduino Uno's microcontroller), has only $2KBytes$ of RAM, while the discussed vector is about $4Kbytes$. Furthermore, more than one vector is needed, and other auxiliar variables are used during the execution. There are two different solutions: the first one is to use a microcontroller with enough RAM to contain all the needed information. The other solution is to work as much as possible in fixed point representation, which is smaller in size, or to reduce data dimensions, which will decrease accuracy in the image reconstruction. Thus, there is a trade off between **data accuracy**, **memory space** and **image reconstruction accuracy**, and an analysis on those parameters is going to be performed. Microcontrollers such as ATMEGA 2560 (Arduino Mega's microcontroller) have $8KBytes$ of RAM, whereas microprocessors, have $512MB$ or even $1GB$, such as Raspberry Pi. Eventhough, as stated before, a microprocessor is too complex for the application's goal, and it is not worthy. The solution is still an efficient memory management. Once the proof of concept is done, the idea is to produce the system using the specific microcontroller found in the chosen COB in a small PCB.

Selection

After having a look at the most important parameters to take care of for the selection of the microcontroller, one of the models to be considered is Arduino Mega 2560 model. The operating voltage of this microcontroller is $5V$, as many Arduino models. It implies that it can be supplied using a simple battery. It has $256KB$ of program memory (Flash) and $8KB$ of RAM, which, as stated before, seems to be enough to implement the algorithm. It is $102mm$ long and $53.3mm$ wide, being bigger than other models such as Arduino Uno (all this data is found in **Figure 6.3**). Remember that system's reduced size is one of the designs requirements. However, using an Arduino Uno or Mini has some drawbacks, even if the size is smaller. For example these two models (in fact it is the same microcontroller for both COBs) have memory problems, given that they only have $2KB$ of RAM. Therefore, if they were chosen, working in fixed point variables would be compulsary. Furthermore, Arduino Mega 2560 admits most of the Arduino Uno shields, including the microSD shield that is needed to add an external memory to the system.

Concerning clock speed, which will be very important in combination with RAM to get all the information

coming from the microphone, Arduino Tian, Arduino M0, Arduino Yun Mini, Arduino M0 Pro and Arduino Industrial 101 offer a faster clock than Arduino Mega 2560. However, all these models are retired from market. Concerning prices, Arduino Uno costs around 20€, while Arduino Mega 2560 costs 35€. Others, like Arduino Nano cost also 20€, or Arduino Due, which costs 35€. Arduino Mega 2560 is a bit more expensive than Arduino Uno, but remember that the microphone and its power source are far more expensive. Then, the cost difference is no longer a problem, and other memory space is considered more important to decide which microcontroller suits better the system.

Concerning other types of systems, such as Raspberry PI or Beagle Board, it has been explained that these systems are much more complex than a microcontroller, and even if they have the required characteristics to develop the application, they have many tools that are not required, and require more knowledge and time to learn and start working with them. There are other families of microcontrollers' boards, such as Teensy. Their specifications are shown on **Table 6.2**:

Specification	**Teensy 2.0**	**Teensy 2.1**	**Teensy 3.0**	**Teensy 3.1**
Processor	ATMEGA32U4 8 bit AVR 16 MHz	AT90USB1286 8 bit AVR 16 MHz	MK20DX128 32 bit ARM Cortex-M4 48 MHz	MK20DX256 32 bit ARM Cortex-M4 72 MHz
Flash Memory (Bytes)	32256	130048	131072	262144
RAM Memory (Bytes)	2560	8192	16384	65536
EEPROM (Bytes)	1024	4096	2048	2048
I/O	25, 5 V	46, 5 V	34, 3.3 V	34, 3.3 V 5 V
Analog In	12	8	14	21
PWM	7	9	10	12
UART,I2C,SPI	1,1,1	1,1,1	3,1,1	3,2,1
Price	$16.00	$24.00	$19.00	$19.80

Table 6.2: *Teensy board benchmark.*

Teensy 3.1 seems to be really appropriate to solve many of the problems in the project. It has eight times more RAM than Arduino Mega 2560 ($64KB$ vs $8KB$), and its clock goes four times faster. It is able to measure faster and longer in time than Arduino Mega 2560, however, as it will be seen later in the report (see **Section 6.2.2**), even with Teensy 3.1 it is not possible to get the desired information in just one temporal measurement. Thus, there is no real difference n measuring between both microcontrollers. Teensy 3.1 is also cheaper, and it can be coded using Arduino IDE, the same open source environment as Arduino. However, there is more availability of shields for Arduino than for Teensy.

Regarding another important feature, which is the measuring speed, the ADC converter of ATMEGA2560 (Arduino Mega 2560 microcontroller) is said to be stabilized between 13 and $260\mu s$ (Section 26 in the ATMEGA2560 data sheet), which means that a value can be read in the analog input each $13\mu s$ as minimum. This establishes a maximum in the measuring rate of $77kHz$. This should be enough to recover information in the $0 - 20kHz$ bandwidth, but as it will be seen later, going faster gives better resolution on these frequencies. The ADC converter of MK20DX128 microcontroller (the microcontroller in Teensy 3.1), is however, faster than that in the ATMEGA2560. It is said to be stabilized between 1 and $50\mu s$ (see Section 6.6.1 in the MK20DX128 data sheet, where this parameter is called C_{rate} and it is measured in $Ksps$). Teensy 3.1 is then able to read parameters up to $1MHz$.

After all this discussion, it has been shown that Teensy 3.1 is a more interesting option. Nevertheless, all the experimental results and the code has been prepared for the Arduino Mega 2560, as it was available at the moment of the experimental part of the project. Repeating the measurements using Teensy 3.0 instead should not be a big problem: the programming language and its IDE are the same as for Arduino, and both LCD and microSD modules are compatibles. As a difference, the PCB that is going to be designed in the following sections should be redesigned to fit the shape and size of Teensy 3.1 instead of Arduino's ones.

6.1.3 External memory

An external memory is necessary to save some parameters and load others in the code. As seen in the previous section, microcontrollers does not have much program memory. The system should be able to manage all the weights in the reconstruction algorithm (Artificial Neural Network) and save images. It has been decided to use a microSD card as external memory, and communicate the microSD card with the microcontroller using SPI. This kind of data transmission is a communication standard used between electronic components.

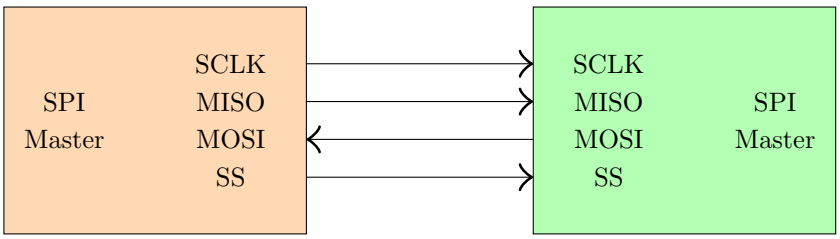


Figure 6.4: *SPI*

SPI communication, shown in **Figure 6.4**, includes a clock signal, an input data, an output data and a pin to chip select, connecting or disconnecting the components, allowing multiplexing. As it is a serial bus, it reduces the number of wires and integrated size. This also reduces the manufacturing cost. It is a synchronous communication protocol. Communication is done using four signals:

- **SCLK (Clock)**: signal used to synchronize. For each signal pulse, a bit is sent.
- **MOSI (Master Output Slave Input)**: it is the signal to send data from the Master to the Slave.
- **MISO (Master Input Slave Output)**: it is the signal to send data from the Slave to the Master.
- **Chip select**. This signal is used to select an Slave, or to tell the Slave to wake up.

The chosen socket to communicate the microSD card and the microcontroller is a Micro TF Card Memory Shield Module SPI Micro Storage Card Adapter for Arduino, as shown in **Figure 6.5**. This module has his own Arduino library, which includes the SPI communication, making easier to work with.

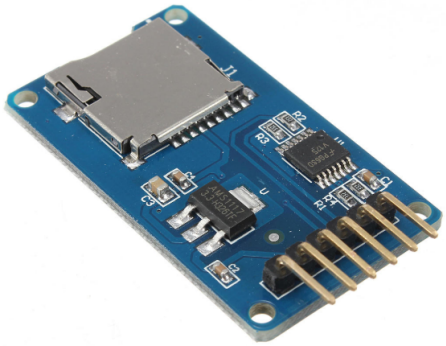


Figure 6.5: *MicroSD card adapter for Arduino.*

The pinout to connect this microSD shield to the Arduino Mega 2560 board is: MOSI is connected to pin 51, MISO is connected to pin 50, SCLK is connected to to pin 52 and CS is connected to pin 53.

6.1.4 LCD Display

The intended goal of the project includes showing the result in a screen, so as to tell the user the image that the system is reconstructing. Another serial communication has been chosen, this time with an LCD display. The chosen model is HD44780U 20×4 Dot Matrix Liquid Cristal Display Controller/Driver from Hitachi, available at Sparkfun Electronics.

Figure 6.6: 20×4 *LCD display.*

This LCD display is powered at $5V$; thus, it is enough to connect to the $+5V$ output pin from Arduino's. Serial communication between the LCD display and the Arduino board will be done using *Wire*, *LiquidCrystalI2C* and *LCD* libraries.

6.1.5 Power supply

The system is supposed to be autonomous. Thus, batteries will be prioritized against other power sources implying a connection to the electrical network. Arduino Mega 2560 can be powered with a

battery between $6V$ and $12V$, depending on the consumption of the system. Therefore, a consumption analysis should be done before selecting the proper power source for the system.

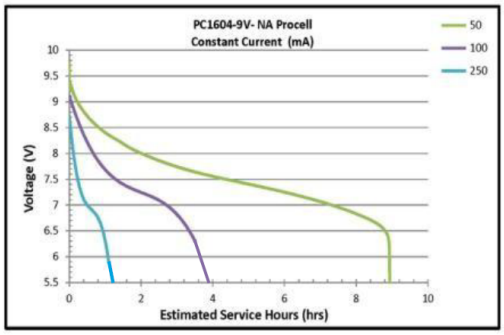
Summing up the project, there are few components that suppose a relevant consumption for the system: Arduino Mega 2560, microSD card, LCD screen and the microphone amplifier. From *The Power Consumption Database* on Internet, consumption values for the Arduino Mega 2560 have been obtained: the average consumption, measured at the USB plug using a $5.15V$ voltage during 507 minutes is $52.54mA$. Concerning microSD cards, it has been found that standard microSD cards consume between 5 and $30mA$, depending on the function or usage. The maximum is going to be considered in order to select a power source for the worst possible case. In the technical specifications of ATMEGA328P it can be found that the maximum current per output pin is $40mA$. However, as it can be seen in **Appendix A**, the current consumed by the loudspeaker is much lower than $40mA$ (in fact, it is $500nA$). The LM324N module used as an operational amplifier has also a low consumption: $700\mu A$. Finally, the LCD screen is said to consume between 20 and $40mA$, considering again the higher value for the project's purposes.

The total current consumed by the system is then $123.24mA$. Once this data is computed, the power source can be selected. It has been chosen to use a 9V battery, but there are several types: disposable (Alkaline, Zinc-carbon or Lithium) and rechargeable (NiCd, NiMH, Lithium polymer, Lithium-ion or Lithium iron phosphate). The most common type and also the cheapest one is the Alkaline battery. Its capacity is also the lowest one, but it has to be considered as a possible solution. Alkaline 9V batteries have a typical capacity of $550mAh$, and it is said that they can stand up to a $300mA$ drain current. Nevertheless, the battery capacity depends on the rate of discharge of the battery. Thus, as the current increases, the battery capacity decreases. Assuming that the capacity is still $550mAh$, the system consuming $123mA$ will provide 4 hours and 28 minutes of service. The battery capacity is of course lower than $550mAh$, but also the consumption is lower, as it has been considered the worst possible case, and not all the devices are working simultaneously in the system.

The chosen battery is a **PC1604**, which is a 9V 6LF22 Alkaline-Manganese Dioxide Battery from Duracell. This battery has the service curve shown in **Figure 6.7**. In this curve, the line that has to be followed is the purple one, given that the system's consumption is near $100mA$. As said before, Arduino Mega 2560 can be powered between $6V$ and $12V$. Then, it will be considered that the battery is able to power the system until the voltage level goes under $6V$, giving 3 hours and 45 minutes of service, which seems to be enough to make hundreds of measurements.

Another option is to use rechargeable batteries, which are more expensive, but also have higher capacities and they can be used more than once. The alkaline battery has been finally selected because it is the simplest and cheapest way to power the system.

(a)



(b)

Figure 6.7: *Duracell 9V battery with its discharge curve.*

6.1.6 Noise generation

Concerning the emitted signal, the goal is to transmit an electric signal to the piezoelectric so as to produce a **white acoustic noise** in the $0 - 20kHz$ spectral band. In signal processing, white noise is considered when the Fourier transform of a signal shows the same amplitude for all frequencies in the bandwidth. In other words, the signal is formed by all frequencies equally.

Generating such a signal is not an obvious process; common methods used in DSP use sequences of random numbers with spectral properties that approximate white noise. The quality of the signal depends on the used algorithm (PRNG algorithms). Most of the algorithms use an algebraic formula that depends on certain parameters, some of them fixed in the function, and others that can be chosen by the user (usually called *random seed*). The key point of a PRNG is to give a sequence of numbers that do not evidence an algebraic formula behind. They are state machines, because the following result in a sequence depends on the previous value. As can be easily deduced, there is a moment when the sequence starts again and values start repeating; the aim of these algorithms is to elongate as much as possible the sequence of numbers before repeating again their values. Some PRNG algorithms are LCG (Linear Congruential Generator), Marsenne Twister, PCG or Yarrow.

Despite the fact that other algorithms present better statistical properties, or are harder to predict, the only aspect that matters for the application is the spectral information they produce. Thus, even if it is one of the trivial PRNG algorithms, a LFSR algorithm has been chosen. LFSR algorithms are based on an initial N bit register. These bits are initialized with a given value (random seed), and then, a Boolean operation between some bits is produced, in order to generate a new bit. After that, this new bit enters the register by shifting all the bits one position. The new bit enters as the most weighted bit, while the less weighted bit is deleted. This process can be seen in **Figure 6.8**:

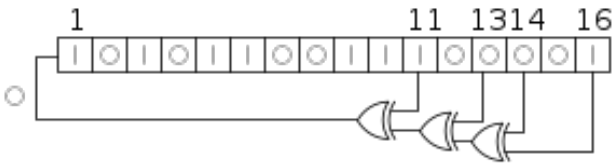


Figure 6.8: *16 bit Linear-Feedback Shift Register.*

In the example, bits $11, 13, 14$ and 16 are compared using XOR gates and the output from this Boolean operation is the new bit. In the figure:

$$bit = b_{11} \oplus b_{13} \oplus b_{14} \oplus b_{16} = 1 \oplus 0 \oplus 0 \oplus 1 = 0 \tag{6.1}$$

A 0 value is obtained. This method produces a sequence of $2^{16} - 1 = 65535$ different values, that is to say, it is the sequence's period (the period depends on the random seed used, but the maximum it can be achieved is 65535). This period and the microcontroller's clock frequency will define the repetition rate of the sequence. For example, imagine a clock frequency of $16MHz$ and suppose that emitting a number is only one clock cycle (it is not true, it depends on the lines of code used), then, each $\frac{1}{16 \cdot 10^6} \cdot 65535 = 0.0041s$, the same value will be emitted. This means that, in case of emitting PRNG more than $0.0041s$, the spectrum will have a peak in $\frac{1}{0.0041s} = 244Hz$, disturbing white noise distribution.

The PRNG algorithm has been chosen, but its frequency properties has not been shown:

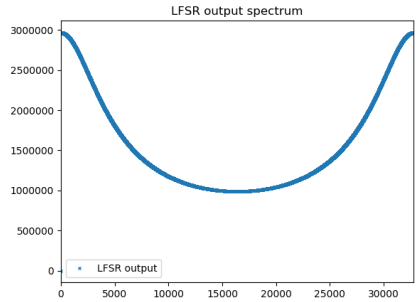


Figure 6.9: *LFSR output spectrum.*

As it can be seen in **Figure 6.9**, the LFSR register has a bias towards lower frequencies. 8003 numbers have been generated using a 16 bit register as before. If instead of representing the spectrum of the whole register, only few bits are considered, a different result is obtained.

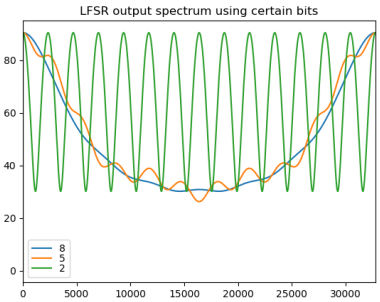


Figure 6.10: *LFSR output spectrum considering different amount of bits.*

The spectral properties do not go better. Instead, oscillations in the spectrum have appeared. Nevertheless, if only the last bit in the register is considered, all frequencies have the same amplitude: white noise distribution has been achieved.

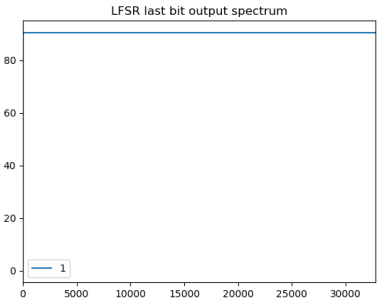
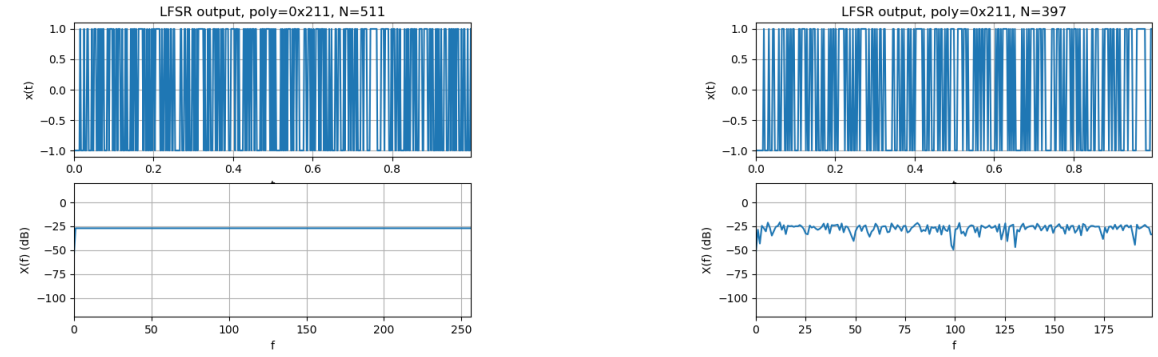


Figure 6.11: *LFSR output spectrum considering just the last bit.*

This spectrum also depends on the number of pseudo-random numbers generated. As shown in **Figure 6.12**, the less numbers are generated, the less flattened is the spectrum.



(a) (b)

Figure 6.12: *LFSR output spectrum with for different amount of samples.*

Furthermore, emitting the whole LFSR register would imply using an analogical output with a 16 bit DAC, which is really difficult to find in a microcontroller. Instead, using the properties of the last register bit, only a digital output is needed, as all the information is coded in two states. As for the repetition rate, the same problem as for the whole register appears: in case of emitting more than 65535 bit, a frequency peak should appear in the spectrum.

6.1.7 Analogical signal processing

Previously, a white noise generation algorithm has been established, and software inside the microcontroller has been prepared. However, many microcontrollers are supplied at $3.3V$ or $5V$, whereas the chosen loudspeaker works fine for $16Vp-p$ voltages. This basically means that a driver is needed to supply enough power to the loudspeaker if a loud sound wants to be produced. In **Figure 6.13** an electronic scheme is presented:

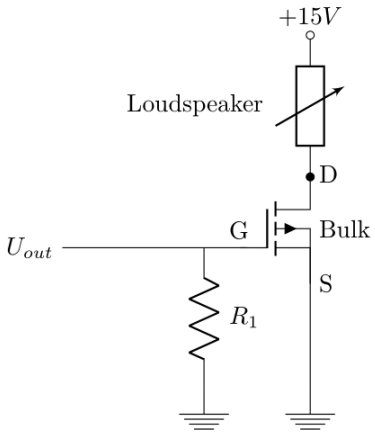


Figure 6.13: *Driver scheme.*

This is the most common driver used to supply a loudspeaker and command it using a microcontroller. For

more details, like resistor and MOSFET selection or a brief explanation, readers can refer to **Appendix A**. This electronic circuit has been considered as it is the easiest way to supply the maximum voltage to the loudspeaker. However, it also implies the use of a more complex battery. In the same Appendix, it has been explained that the loudspeaker can be fed directly with the Arduino's digital output. Then, even if the power is not the maximum available for the loudspeaker, there is no need to use any driver. In conclusion, it has been decided that supplying directly from the Arduino is a better solution than designing a driver with its own battery.

Loudspeaker is not the only peripheral that needs analogical signal processing to adapt to the needs of the components. In fact, the input signal, coming from the microphone, is not at all adapted to the microcontroller's analogical input. Remember that the output signal from the microphone goes through its power supply (GRAS 12AL), and a signal with an amplitude $0.2Vp-p$ and an offset is obtained. This offset is not a constant, given that it is due to the control the power supply does to the microphone. Microcontroller's input range goes from 0 to $5V$ with a 10 bit ADC. If the signal is connected directly, the signal will be coded using only 41 values, which means 4% of the input range. This is obviously unacceptable; audio signal must be processed before going to the microcontroller.

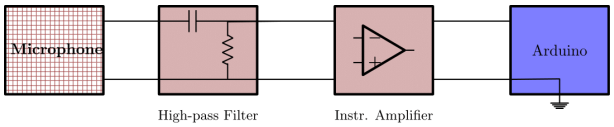


Figure 6.14: *Audio signal processing chain.*

In **Figure 6.14**, the block diagram between the microphone and the microcontroller is shown. First, as the output signal of the microphone a differential output (that is to say, it has two terminals), one of those will be considered as the ground, and it will be connected to the microcontroller's ground (in **Figure 6.14**, it is represented by the lower horizontal line). Applied signal processing can be divided in two different blocks; the first one is a high-pass filter, which aim is to eliminate the offset the original signal has, and then an amplifier, whose role is to set the signal in the range of the microcontroller's input range.

The high-pass filter has to stop continuous signal, but in any case it filters out signals between $5kHz$ and $15kHz$, as it is the modulated frequency band. Hence, it has been chosen to place the filter's cutoff frequency (f_0) between $10Hz$ and $100Hz$. Remember that cutoff frequency is the frequency that is attenuated by a $-3dB$ factor. For more design details, refer to **Appendix A**. **Figure 6.15** shows how the Arduino's input range would be used at each step on the processing chain. The final step (after the filter and the amplifier) has the same range for the signal and for the Arduino's input. Then, the maximal resolution for the signal is achieved. In this Appendix, two different amplifiers have been discussed and computed, to finally decide which one is better from the point of view of the project. The final electronic circuit is shown in the following diagram:

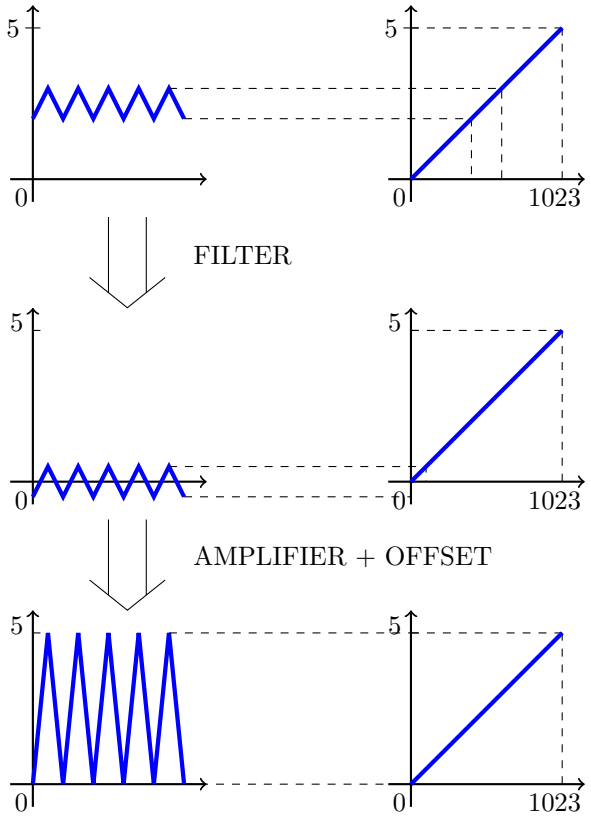


Figure 6.15: *Signal condition in every chain step.*

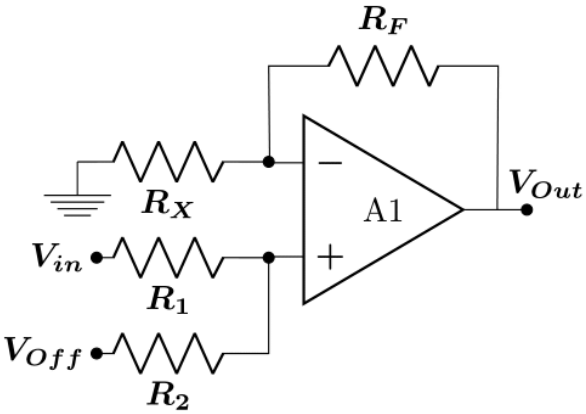


Figure 6.16: *Non inverting summing amplifier.*

Even if the other amplifier was a really popular and well-known option (the instrumental amplifier or three operational amplifier is widely used in industry), it implies using symmetric (or at least negative) voltage to supply the operational amplifier. Then, the system's power supply gets complicated, and a single battery cannot be used. By using this amplifier (**Figure 6.16**), the offset voltage is added before passing through the amplifier, thus allowing the voltage supply to be asymmetric. Furthermore, the offset voltage has been fixed to $3.3V$, which is a value that can be easily provided by Arduino. For more details on the values of the components, please refer to **Appendix A**.

6.1.8 PCB design

Once the electronic components have been selected, it is possible to design a PCB that integrates all the electronic devices and which size and connections are compatible with those from Arduino Mega 2560. In this way, an Arduino Mega 2560 shield has been conceived, to easily branch Arduino's connections with those from the shield, and to let other devices (microSD shield and LCD screen) connect simultaneously.

An Eagle template called *Practical Arduino Proto Shield Mega* has been downloaded. It can be found on Github, property of Jonathan Oxer and Marc Alexander. This template has the same size as a shield for Arduino Mega 2560 and has all the connections between Arduino Mega 2560 input/output pins prepared. In the middle zone, there is a space that can be used to prototype some circuits. Thus, this area has been used to implement the processing chain that has been discussed in the previous section. In **Figure 6.17**, the schematic can be seen. There, the central area has is the zone were the processing chain has been connected.

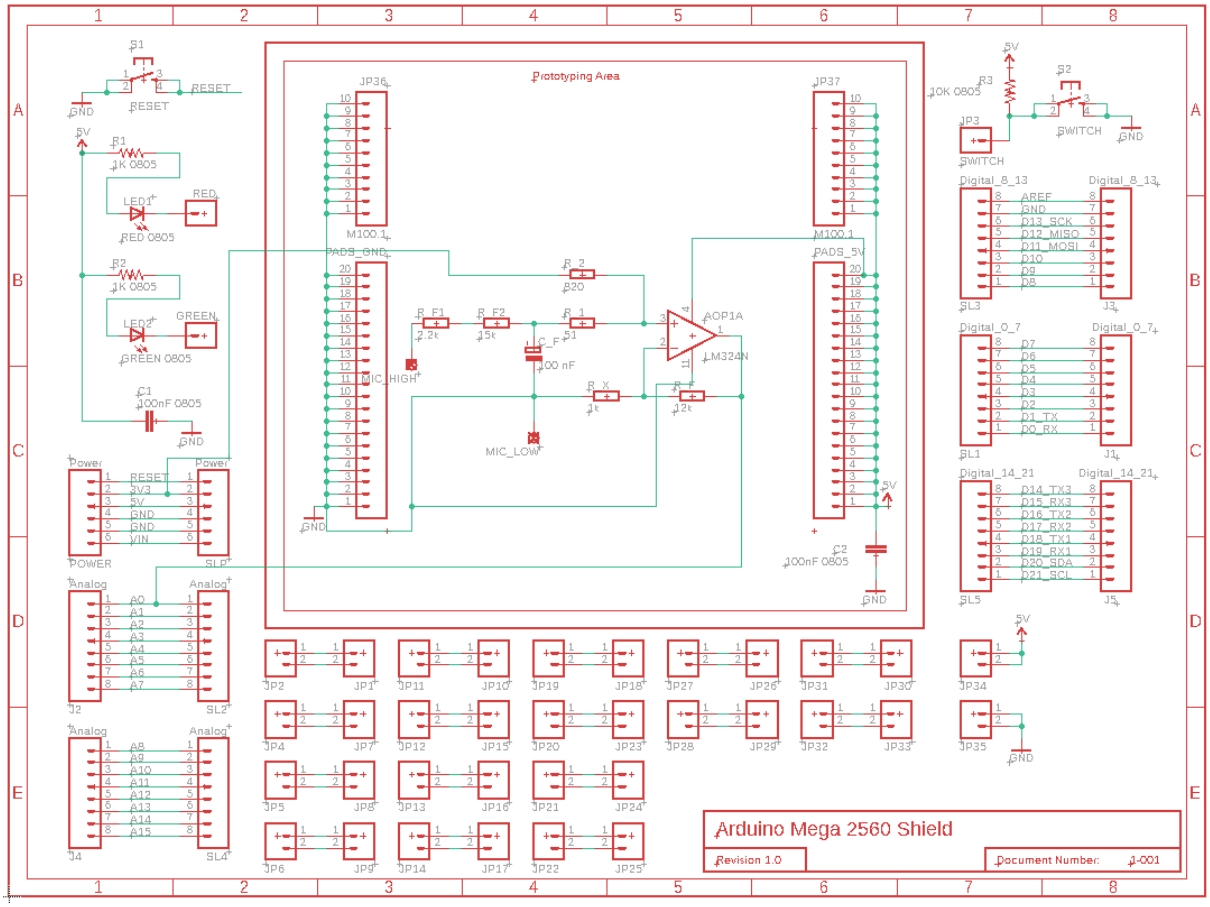
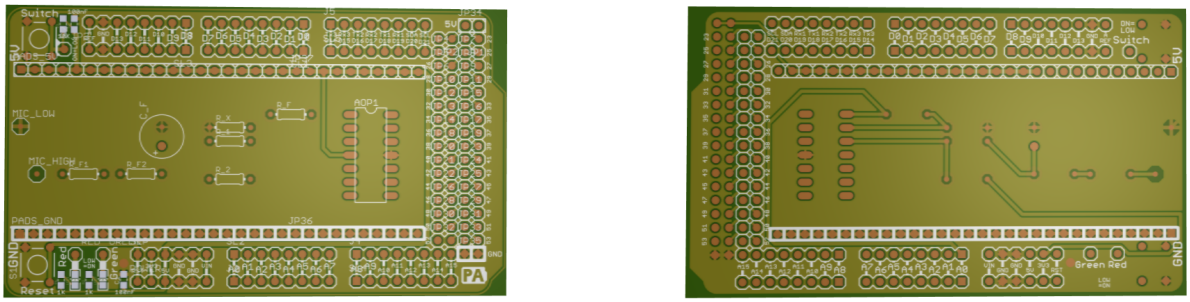


Figure 6.17: *Arduino Mega 2560 shield schematic. The area called Prototyping Area is where the processing chain is placed.*

After doing all the connections, and considering that the chosen template already used two different copper layers (top and bottom), the result is a two-layer PCB, shown in **Figure 6.18**. This figure has been obtained using **webGerber**, an online tool that gives a 3D view of Gerber files.



(a) (b)

Figure 6.18: *PCB board without the components. At left, the top side of the board. At right, the bottom part. Both sides have copper.*

6.2 Software's design

6.2.1 Microcontroller's ADC configuration

Once the signal is ready to enter the microcontroller, the ADC must be regarded. Some important features are how many bits does it use (voltage resolution) and at which frequency does it read.

Arduino's ADC usually have 10 bit resolution (in other words, they can differentiate $2^{10} = 1024$ voltage levels between 0 and $5V$). Concerning the sampling rate, it is an internal clock speed multiple. Arduino UNO's internal clock works at $16MHz$ (the same happens for Arduino MEGA 2560), and there is an internal value called prescaler that divides this frequency. In Arduino, it is fixed by default to 128. Moreover, each register refresh (or read) needs 13 clock cycles. Then, input signal is sampled at $f_s = \frac{16 \cdot 10^6}{128 \cdot 13} = 9615Hz$.

This sampling rate is largely not enough: following Nyquist criterion, the highest frequency that can be reconstructed given a sampling rate is: $f_{max} = f_s/2 = 4807Hz$. Remember that the acoustic signal is modulated by the structure and the object between $5kHz$ and $15kHz$. Sampling frequency must be able to get up to $20kHz$ information. Microcontroller's internal clock can not be changed, but as seen before, there are two factors that divide this rate: the prescaler and the reading cycles. The latter is not accessible and can not be changed, but the former is a configuration parameter. By going to low level programming and changing the prescaler, the acquisition sampling rate is able to increase. Nevertheless, there exists an important drawback. Experience from Arduino users has shown that as the acquisition sampling rate goes up, the accuracy of low weight bits in the register decreases. In fact, when this rate overcomes a certain limit, only the biggest 8 bits stay reliable, and then the signal is coded in 255 values only. This is an important trade off between voltage resolution and temporal resolution that will constraint the system.

Despite the fact that hardware reading has been solved, there is still an inconvenient that tights the sampling rate. The coded program in the microcontroller is supposed to emit and read both signals simultaneously during a certain time window. The emitting and receiving routine has been coded using an interruption; this way, periodic calls to the routine are ensured. The following code sets microcontroller's interruption configuration:

```
1  // set timer2 interrupt
   TCCR2A = 0; // set entire TCCR2A register to 0
3  TCCR2B = 0; // same for TCCR2B
   TCNT2 = 0; // initialize counter value to 0
5


7  // turn on CTC mode
   TCCR2A |= (1 << WGM21);
9  // Set CS21 bit for 8 prescaler
   TCCR2B |= (1 << CS21);
11 // enable timer compare interrupt
   // TIMSK2 |= (1 << 0CIE2A);
13 TIMSK2 &= B11111101; // Avoid interruptions
```

Arduino_code/conf.cpp *excerpt*

This excerpt of code is executed during Arduino's *setup()* function, and must be combined with ATMEGA2560 datasheet to be understood. First of all, the three first lines delete all previous content in timers configuration registers. Next, bit WGM21 in TCCR2A register is set. This corresponds to bit 1 in this register, and remember that the other bits are still at 0 value. Then, CTC mode (Clear Timer on Compare Match) is set. Different modes are available by changing this bit and WGM20 and WGM22, as seen in **Table 20-8** of the datasheet:

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of operation	TOP	Update or OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Table 6.3: *Waveform Generation Mode Bit Description.*

CTC mode has been chosen as the desired interruption code. In this mode, the OCR0A register is used to manipulate the counter resolution. The counter is cleared to zero when the counter value (TCNT0) matches the OCR0A. This mode allows greater control of the Compare Match output frequency. Thus, modifying both the prescaler and the OCR01 register, interruption call frequency can be controlled and, therefore, sampling frequency can be controlled.

After this, CS21 bit in TCCR2B register is set. This bit, together with CS20 and CS22 define the prescaler, as can be seen in **Table 20-9**:

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T2S}/ (No prescaling)
0	1	0	$clk_{T2S}/8$ (From prescaler)
0	0	0	$clk_{T2S}/32$ (From prescaler)
0	1	1	$clk_{T2S}/64$ (From prescaler)

Table 6.4: *Clock Select Bit Description.*

Finally, there is an instruction for avoiding interruptions. This instruction appears here because interruptions will be activated in the desired moment in the *loop()* function.

Next, measuring method will be presented, and changes in the Neural Network Algorithm will be discussed in order to adapt microcontroller's needs.

6.2.2 Measuring method

Remember from **Chapter 5** that the Artificial Neural Network's input is a 997 double precision vector representing spectral coefficients between $0Hz$ and $20kHz$. In Arduino, double precision and float have the same bit length (32 bits or 4 bytes), meaning that a single 997 value vector is $3.89kBytes$. Microcontroller's Fourier transform usually need two input vectors (Real value vector and Imaginary value vector): then, needed RAM doubles size. As seen in **Section 6.1.2**, microcontroller's RAM is not specially big, and two vectors this size almost fullfill Arduino MEGA2560 RAM.

Furthermore, Fourier Transform algorithm is also important. *arduinoFFT* library has been used, implying two important things: the former, arduinoFFT is an in-place Fast Fourier Transform algorithm; this means that the spectrum is computed and saved in the input vector, overwriting the temporal signal. Hence, the number of spectral coefficients obtained in the algorithm is the same as the temporal samples passed as input (it does not apply zero-padding techniques). The latter is that the applied FFT algorithm is a Radix-2 algorithm, meaning that it only accepts input vectors whose length is a power of 2. Considering this two remarks, the measurement vector might be 1024 value long, or 512. The first one implies more RAM used, and probable memory problems during computation. The second one seems to be more adequate. However, it also means that the Artificial Neural Network has to be solved using only 512 values as entry.

Accuracy error on the reconstructed image depending on the number of spectral coefficients passed as input has been characterized in order to obtain a trade off function between input size (which, as seen before, is related to RAM memory) and reconstruction accuracy. For more details on this analysis, refer to **Appendix C**.

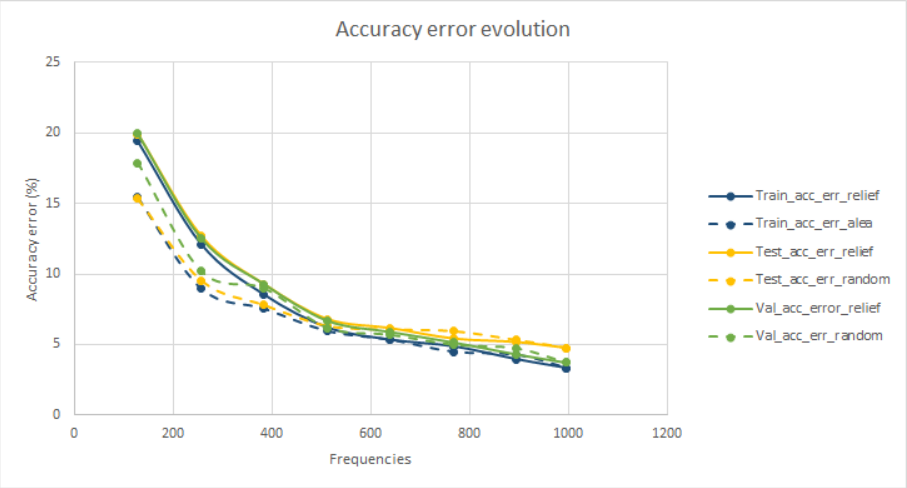


Figure 6.19: *Accuracy evolution depending on input size vector.*

Figure 6.19 shows how the accuracy error diminishes while the input vector size of the Artificial Neural Network grows. The three different colour lines represent the three data subsets (as explained in **Appendix C**). From this graph it can be outlined that there is no major difference between using 500 frequencies and 1000 frequencies in terms of Accuracy error. Descending to 256 frequencies implies an important increase of this metric. It must be said that this simulation has been done using a 100 holes problem, instead of the 24 holes problem solved here. The idea is to get the behavior of the system, and it has been considered that the behavior for these two features is the same for both problems. This is why in the figure above an accuracy error of 5% is achieved, while as it will be discussed later, the accuracy error decreases to less than 1%.

There is still another problem to solve: imagine that the microcontroller's ADC is able to get a sample with a sampling frequency of $200kHz$. Applying Nyquist criterion, the highest frequency it is able to solve is $100kHz$, which is higher than $20kHz$. As the frequency vector obtained is as long as the temporal vector, only 512 different frequencies will be obtained. Moreover, the spectrum is centred at 0 frequency and symmetric (this is a property derived from real signal Fourier transform). Then, only 256 coefficients are valid. The spectral resolution (distance between measured frequencies) is $\Delta f = \frac{f_s}{N_f} = \frac{200000}{512} = 390.625Hz$. Finally, computed spectral coefficients in the $[0-20]\,kHz$ band are

$$N_{coef} = \frac{f_{max}}{\Delta f} = \frac{20000}{390.625} = 51.2. \tag{6.2}$$

Obviously, 51 coefficients are not enough to be used as neural network's input. If the total number of samples is kept fixed (as seen, there is few freedom in this parameter) and the sampling rate is decreased, the number of spectral coefficients in the desired bandwidth increases. In spite of this result, which could easily make you think that decreasing sampling rate gives the solution to the measuring problem, there is another feature to take into account: temporal samples measured in a $20kHz$ cycle. This metric is simply stated as

$$res_{f_i} = \frac{f_s}{f_i} = \frac{200000}{20000} = 10. \tag{6.3}$$

Remember that a minimum of 2 samples is needed to determine a frequency. As long as sampling decreases, so does the resolution at $20kHz$, becoming a less reliable measurement. Up to this point, there is another trade off between frequency resolution and number of coefficients obtained.

The adopted solution is to measure several times changing the sampling frequency, Fourier transforming the data and saving those spectral coefficient in the desired bandwidth. As seen before, sampling frequency is controlled by the prescaler value and OCR0A register. Prescaler also affects the voltage resolution (as it decreases, lower ADC bits become less reliable); it has been fixed at 8 value, losing the two lower bits in the ADC register. OCR0A register also has its limits: it is upper-limited to 255 ticks (it is a 2 byte register), and it is low-limited by the duration of the inner code in the interruption. This seems logic; if the code inside the interruption routine is longer that the time between two interruption calls, the second interruption will be executed before finishing the code of the first call. The code inside the interruption is:

```
ISR(TIMER2_COMPA_vect){
    bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5));
    if(bitRead(lfsr,15)==1)
    {
        PORTH |= B00010000;
    }
    else
    {
        PORTH &= B00000000;
    }
    if(n<N){
        vec[n] = ADCH;
        n++;
    }
    lfsr = (lfsr >> 1) | (bit << 15); // lfsr actualization
}
```

Arduino_code/isr.cpp *excerpt*

This code excerpt shows intructions for computing the *LFSR* output bit, the output signal (***if/else*** *structure*), the reading (**ADCH** register), the *LFSR* update and finally the interruption counter update. This code is executed using 20 clock cycles, which means that OCR0A register is low-limited by this value. There is also the *if*$(n < N)$ structure, which works as a protection to avoid *vec*'s array overflow.

Microcontroller's available sampling frequencies have been measured experimentally, and results can be seen in **Table 6.5**:

OCR0A	Elapsed time $[\mu s]$ (512 calls)	Sampling frequency $[kHz]$
22	6060	84.5
24	6412	79.9
25	6668	76.8
26	6912	74.1
27	7160	71.5
28	7420	69.0
29	7676	66.7
30	7920	64.6

Table 6.5: *Experimental sampling frequency.*

Using these sampling rates, a MATLAB simulation has been done in order to get how many sampling

frequencies are needed to achieve 512 coefficients. For more details concerning this simulation, please refer to **Appendix A Section A.3**. The conclusion is that four measurements are enough to get all the spectral coefficients. The chosen sampling frequencies are the forth, fifth, sixth and seventh in **Table 6.5**. The reason of not choosing neither the first nor the second sampling rate is that, as seen in the microcontroller selection, ATMEGA2560's ADC can not stabilize values faster than $77kHz$.

How the measurement is managed and the sampling rate is changed is presented in the following excerpt:

```
uint16_t match[] = {22,24,25,26}; // ORC2A Values

void loop() {

  if(n==0 && not(processing) && fi<4)
  {
    processing = true;
    OCR2A = match[fi]; // = (16*10^6)/(200000*8) - 1 (must be <256)
    TCNT2 = 0; // initialize counter value to 0
    TIMSK2 |= (1 << OCIE2A); // Initialize interruptions
  }
  if(n>=N && processing && fi<4)
  {
    TIMSK2 &= B11111101; // Stop interruptions
    processing = false;
    fi++; // next sampling frequency
    n = 0; // first temporal sample

    // Fourier Transform and Save Coefficients
    // ...
  }
  // Other functions that can be executed in parallel
  // ...
}
```

Arduino_code/loop.cpp *excerpt*

The first *if* structure in the *loop()* function enables interruptions after changing the OCR2A register and taking the counter to 0 (TCNT2 register). The second *if* structure avoids interruptions during Fourier transform and saving time, as well as updates the values that will be needed for the following measurement. f_i variable seems to get to 4 and then this code is never executed again. Actually, what it is intended to do is to reset f_i once the image is reconstructed, in order to start the measurements for a new image, and get images periodically.

The Fourier transform is then computed. The chosen algorithm here is found in the *ArduinoFFT* library. It is a modulo 2 fast Fourier transform (FFT) algorithm. It is not the only solution that has been considered.

```
void arduinoFFT::Compute(double *vReal, double *vImag, uint16_t samples, uint8_t dir)
```

Arduino_code/arduinofft.cpp *excerpt*

This is the method used to compute the Fourier transform. It uses two arrays of *double* as input (*vReal* and *vImag*), which means that, considering $N = 512$ sample values, both arrays are $4kBytes$, implying that this method can not be used with some microcontrollers such as Arduino Uno. In case of using Arduino Uno, it has been considered another Fourier transform algorithm; instead of applying a floating point FFT, it computes a fixed point FFT. It is an algorithm that can be found in the *fix fft* library.

```
1 int fix_fft(char fr[], char fi[], int m, int inverse)
```

Arduino_code/fixfft.cpp *excerpt*

Here, in place of using two arrays of *double*, this function uses two arrays of *char*, which are 4 times less memory consuming. As it has been said in previous sections, these algorithms are *in-place* algorithms, and base 2. An *in-place* algorithm is a code that overwrites the input vector with the output information. It is a useful way to compute something without spending more memory than necessary. Nevertheless, it implies that the input information disappears. Base 2 algorithms are those who can only accept as input an array whose length is a power of 2. Even if the second algorithm consumes much less memory than the first one, it is also really constraining, as *char* only has 255 different states, and *double* has $4,295$ million states. Furthermore, the fixed point algorithm implies several castings to accommodate the arrays to go through the methods (the input array at the reconstruction algorithm needs to be a *double* array). Moreover, speed is also important. As has been explained in this section, four different measurements will be taking place in order to get 512 spectral coefficients. Between these measurements, Fourier transform will be computed and the coefficients will be saved. Then, the entire measuring time is not only the time to buffer the temporal measurements, but also the Fourier transform and the saving in the external microSD card. Time between measurements should be as short as possible, to avoid changes in the environment that could affect the measurement, such as temperature or humidity.

6.2.3 Rearranging coefficients

It has been explained in the previous section that four measurements are needed in order to obtain 512 different spectral coefficients. This is achieved by changing the sampling rate at each measurement. However, even if only the spectral coefficients that correspond to frequencies in the $0-20kHz$ bandwidth are saved in the external microSD card, they are no sorted to enter the reconstruction algorithm.

```
1     myFile = SD.open("INDICES.TXT");
      while (myFile){
3       ii = 0;
        while (ii<512)
5       {
          str_aux = myFile.readStringUntil('\n');
7         Indices[ii] = str_aux.toInt();
          ii++;
9       }
        myFile.close();
11    }

13    myFile = SD.open("MEASURE.TXT");
      while (myFile){
15      ii=0;
        while (ii<512)
17      {
          str_aux = myFile.readStringUntil(',');
19        Re[Indices[ii]] = str_aux.toFloat();
          ii++;
21      }
        myFile.close();
23    }
```

Arduino_code/sorting.cpp *excerpt*

This code excerpt is used to sort the coefficients to enter the reconstruction algorithm. All four measurements are saved in the same *txt* file, and there is also another *txt* file with the order of the coefficients. For more details on how this sorting is computed, please refer to **Appendix A Section A.3**. Communication with the external microSD card is done using the ***SD* Arduino library**, which uses serial communication between the board and the shield where the external memory is inserted. The *SD* Arduino library does not allow to open two different *txt* files at the same time. This is why this process must be done sequentially; first, sorting coefficients are charged in memory, and then the measurements are sorted based on these sorting coefficients.

After rearranging the array, one last step must be done before start reconstructing the image. The input array is not only the spectral information, but it is normalized by the void measurement. This void measurement is written in the external microSD card. Therefore, a new reading must be done and each spectral coefficient must be divided by its corresponding void coefficient.

The whole code can be seen in **Appendix E**. You will notice that the excerpts shown in this chapter are not exactly as the final code is. Excerpts have been modified to show the relevant lines of codes for the explanation in progress.

6.2.4 Reconstruction algorithm

Finally, the Arduino code gets to the prediction step or image reconstruction step. Here, two different approaches have been contrasted. On the one hand, a common least squares algorithm has been tested. On the other hand, a more sophisticated approach using machine learning algorithms has been implemented. For more information about the artificial neural network implemented, please refer to **Appendix C**. Moreover, an insight on the least squares algorithm and its relevance can be seen in **Appendix B**. Here, only the most important results and conclusions on which algorithm is going to be used will be discussed.

Both appendices show that both algorithms are capable of solving the inverse problem, even the linear regression shows a better performance on reconstructed images. In fact, the least mean squares algorithm plus a binary threshold on 0.5 shows a 0% accuracy error on reconstructed images, that is to say, it is able to reconstruct all images on the data sets.

The Artificial Neural Network does not offer such good performance. Using a two layer network (input layer plus output layer) as explained in the appendix, and training 100000 epochs on the data set gives the following result:

Data set	Accuracy error (%)	Perfect rec.	AENPC (%)
Train	0.23	36122/38180	4.27
Test	0.23	12054/12726	4.26
Validation	0.23	8497/8983	4.33

Table 6.6: *5x5 pixel image reconstruction results. AENPC stands for Accuracy Error for Non Perfect Cases, which is the Accuracy Error without taking into account those cases that have been completely well reconstructed.*

The accuracy error is really small, but it is not 0 as for the other algorithm. This could make you think that it is a better idea to solve the problem using the first algorithm, which is much more easy to understand and faster to develop. Nevertheless, remember that the 5×5 plate is not the final product, but it is a proof of concept. The idea is to produce structured plate's where the number of hole's is bigger (more pixels in the image), and the distance between them and their size is smaller. Exploring the behavior of the reconstruction algorithm over a more complex problem (for example a 10×10 plate)

has shown different performances. In this case, the accuracy error for the least squares algorithm is not 0. **Figure 6.20** shows the accuracy error per pixel:

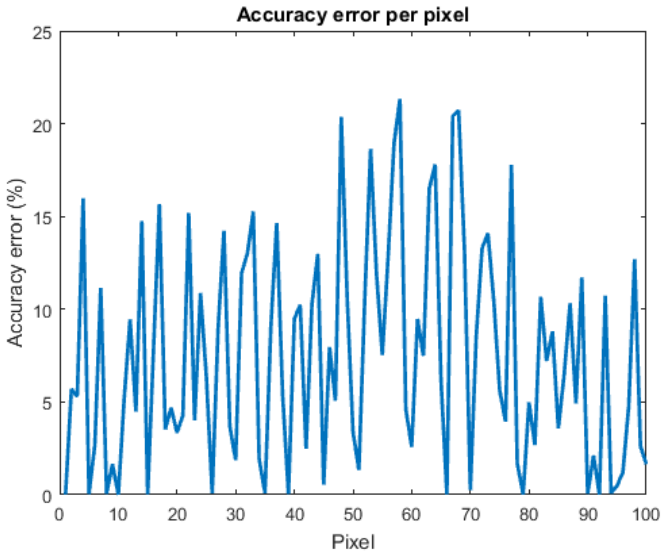


Figure 6.20: *Accuracy error per pixel using a least squares algorithm for the* 10×10 *case.*

These results are not bad, the probability of reconstructing the image is still high, but this time this algorithm does not assure that the result is going to be the right one. Concerning the Artificial Neural Network, the same network as for the 5×5 case has been used, just adapting the shapes and sizes of the layers (the output layer must have the same number of neurons as the number of pixels in the image). The results are as follows:

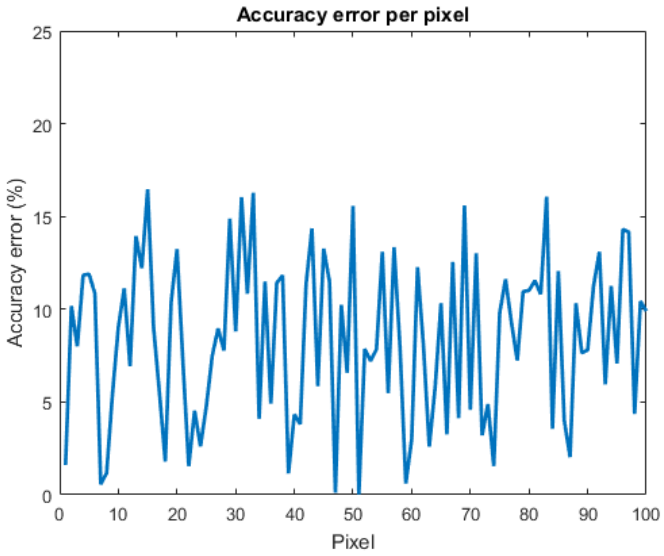


Figure 6.21: *Accuracy error per pixel using an artificial neural network for the* 10×10 *case.*

This performance seems better than the least squares algorithm. This is the reason why it has been decided to apply the machine learning algorithm instead of using a linear regression.

It must be outlined that the Artificial Neural Network model must be modified and adapted each time the number of holes in the plate is changed. The sizes and shapes depend on the plate's features. Therefore, a 10×10 problem can not be solved using an algorithm designed for a 5×5 problem.

Chapter 7

Results

7.1 System's hardware

This section summarizes the hardware selection that has been discussed during the previous chapter. The different selected components are:

- Microcontroller: Arduino Mega 2560
- Microphone: GRAS 46BE 1/4″ CCPFree-field Standard Microphone Set
- Loudspeaker: Kemo L10
- External memory: Micro TF Card MemoryShield Module SPI Micro Storage Card Adapter
- LCD display: HD44780U 20 × 4 Dot Matrix Liquid Cristal Display Controller/Driver

The components listed here are shown in **Figure 7.1**. Moreover, there are other components that have been selected to make the former ones work together:

- Microphone preamplifier: GRAS 26CB 1/4″ CCP Standard Preamplifier withMicrodot Connector
- Microphone power source: GRAS 12AL 1-Channel CCP Power Modulewith A-weighting filter
- Microphone processing chain: high-pass filter and amplifier
- System's power source: PC1604, a 9V 6LF22 Alkaline-Manganese Dioxide Battery from Duracell

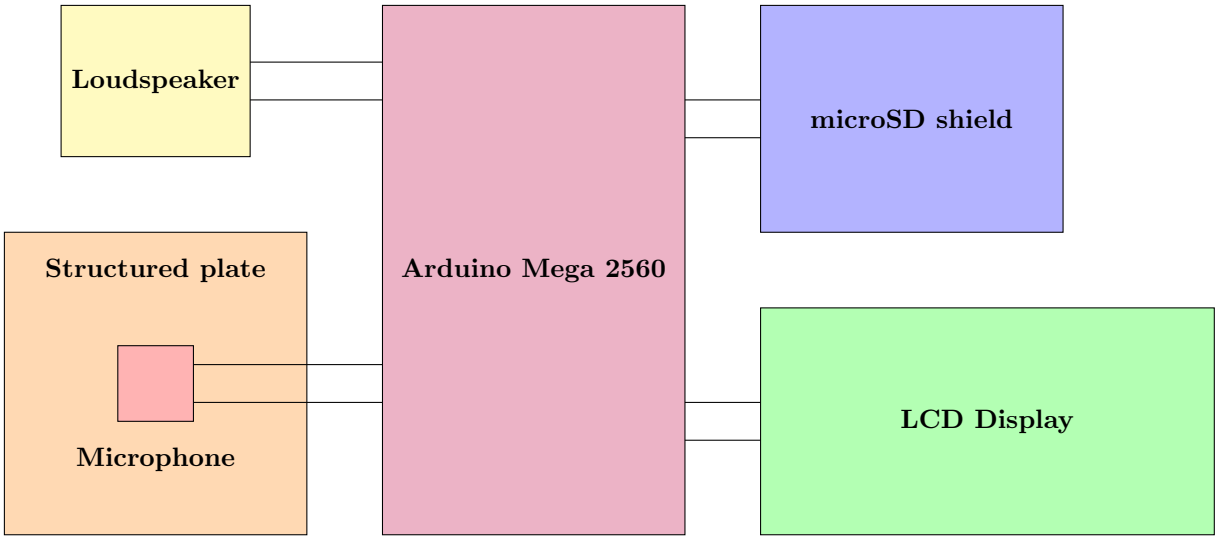


Figure 7.1: *System's block diagram*

7.2 System's software

Concerning the code developed for the system, some excerpts have been shown and explain in the previous chapter, and the whole code can be consulted in **Appendix E**. Here, a code flow diagram is shown in **Figure 7.2** to explain the behavior of the code.

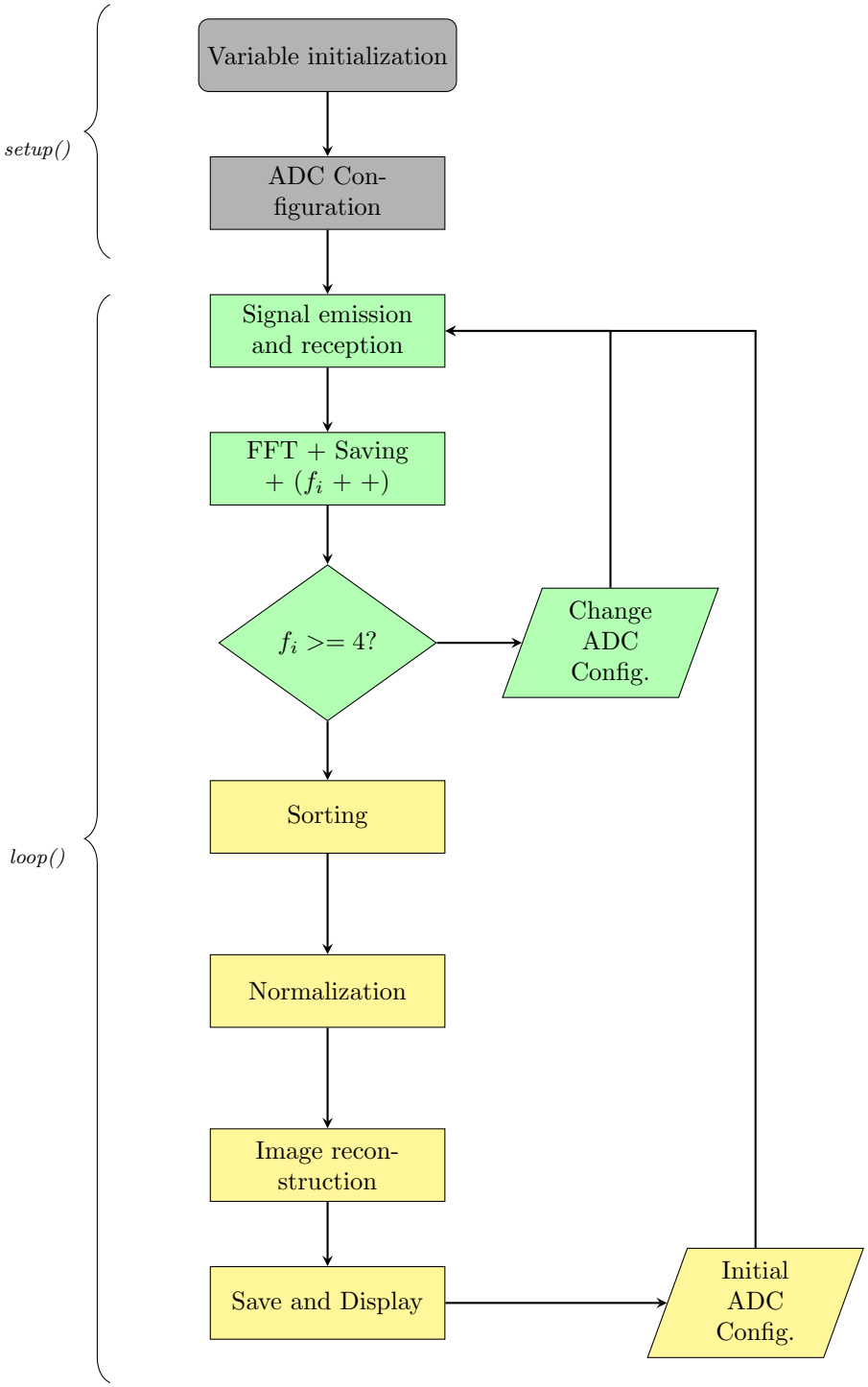


Figure 7.2: *Code's flow diagram.*

The above diagram shows the sequence of functions that take place inside the microcontroller. At the begining there are two processes in gray, corresponding to the *setup()* function. Then, the rest of the code belongs to the *loop()* function. However, there is also a difference between the green processes and the yellow one. The green processes belong to the measurement loop, that is repeated four times per image reconstructed. They are the processes in the code where elapsed time is more important, as it conditions measurement repeatability. The processes in yellow also belongs to the *loop()* function, but they do not need to be extremely fast, as their goal is to reconstruct and show the image with the obtained data.

7.3 System's behavior

Before start measuring and predicting images, there is still one important step to do; as explained before, measurements are normalized before passing through the reconstruction algorithm. This normalization implies the measurement of a *void spectrum*, in other words, measuring the structured plate frequency response when there is no hole covered.

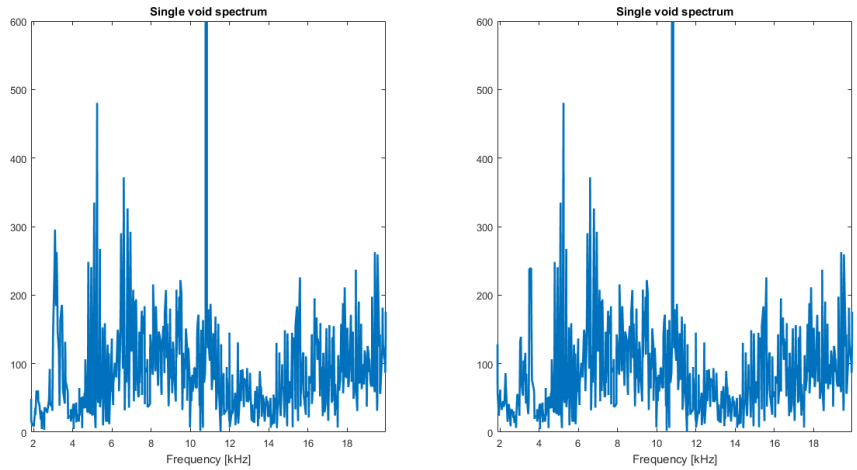


Figure 7.3: *Single void spectra.*

Figure 7.3 shows two different void spectra measured under the same conditions. At first sight they are much alike, even though there is a lot of noise in the measurement. However, the relative error is computed between these two measurements and a 9.67% error is found. Considering that this void spectrum is going to divide the mask measurement, a 10% variation is not good at all. Thereby, it has been decided to reduce experimental noise by taking several measurements and average them. It must be said that the average has been done in the spectral domain; otherwise, high frequency information would be lost (remember that an average is a low-pass filter). Two hundred measurements have been taken under the same conditions, and two different groups have been done, computing two independent mean spectra. The comparison between these two mean spectra is shown in **Figure 7.4**:

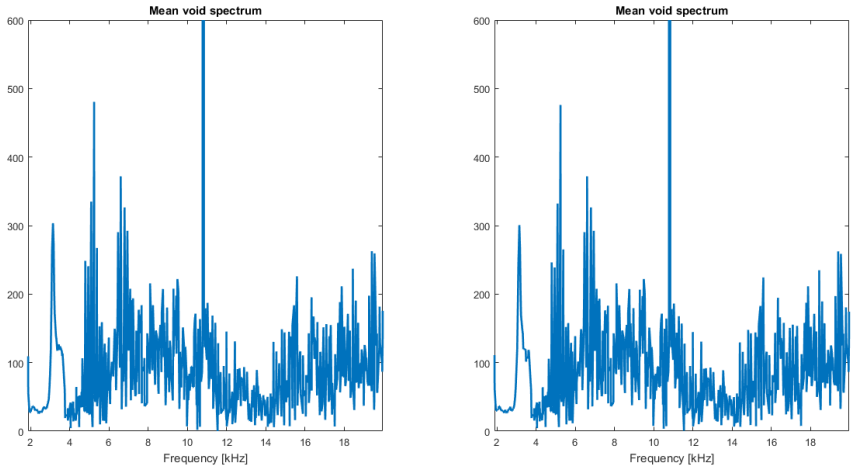


Figure 7.4: *Mean void spectra.*

One could think that there is not so much difference from the first figure. Despite the fact that the low frequencies are smoother than before, noise is still present in both spectra. Even though, when the relative error is computed, the real difference appears: this time there is only a 1.36% error between both means. The average has achieved its goal, and now the spectra are much more alike than before. This experience has been repeated for the case of covering some holes (applying a mask). It consists on a rectangular mask that can cover four different holes. The relative error in this case is 1.40%, which is the same order of magnitude as for the void spectra.

Now, the following step is to prove that the microphone detects a difference when holes are covered. Next figure allows a comparison between two mean spectra: one computed from void measurements, and the other one computed from mask measurements. The figure shows that both spectra are similar; there is still a lot of noise in the measurements and the shape of the curve is alike. However, there are some zones where differences can be noticed (marked in red).

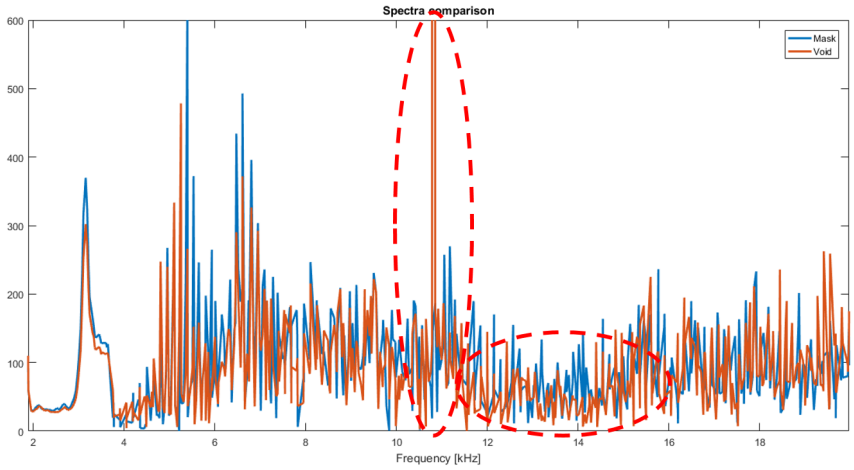


Figure 7.5: *Mean void spectrum and mean mask spectrum.*

The same relative error has been computed between those two signals, obtaining a 111.18% error. Compared to the error obtained between two mean void spectra, it seems clear that the microphone detects a difference when a mask is applied to the structured plate. Finally, **Figure 7.6** shows what would be the input to the image reconstruction algorithm; that is to say, the ratio between the mask measurement and the void measurement. In the left picture, one hundred void measurements and one hundred mask measurements have been used, whereas in the right picture, another one hundred void measurements and one hundred mask measurements have been used. It is important to outline that data from left picture and right picture are totally independent, and therefore, the figure is useful to draw some conclusions. Both figures present peaks at the same frequencies, except for the peak marked in red. Moreover, the order of magnitude of the signal is the same as synthetic signals used as training data.

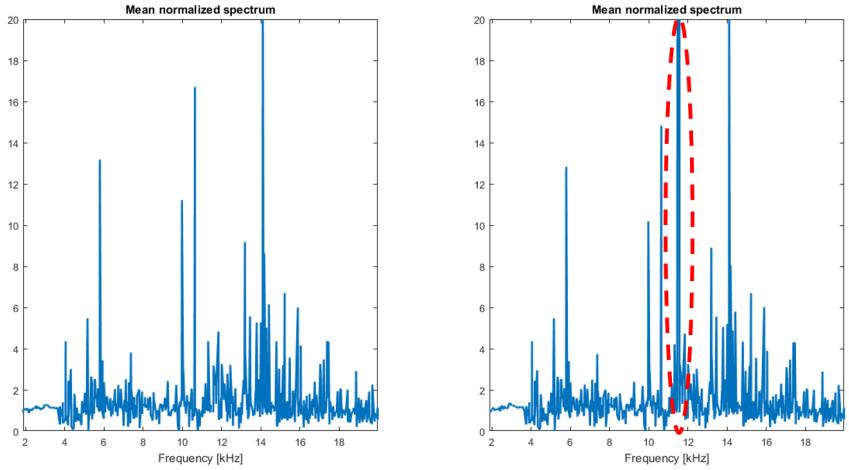
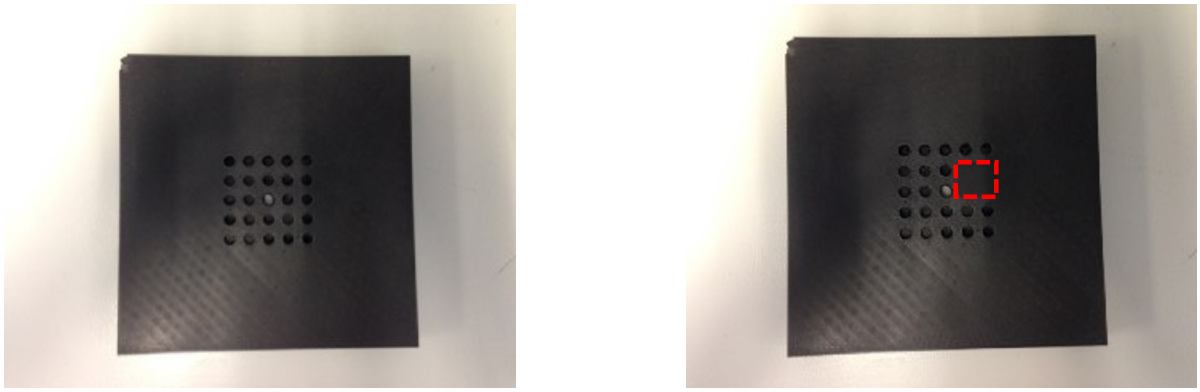


Figure 7.6: *Mean normalized spectra.*



(a) (b)

Figure 7.7: *At left, the structured plate with all its holes uncovered, ready to take the void measurement. At right, the structured plate with a mask applied, ready to take the mask measurement. In order to see better the mask applied, it has been marked in red.*

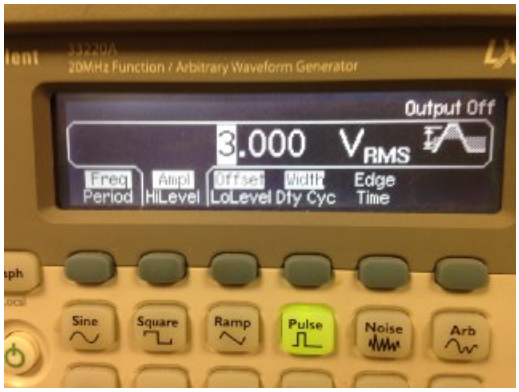
The relative error between both signals is 1.33%, that is to say, the same order as for the mean void measurements and the mean mask measurements. **Figure 7.7** shows the mask (image) that has been applied and measured.

Despite the fact that the normalized spectrum seems similar to synthetic training data, the Artificial Neural Network has been applied to this signal to try to reconstruct the image, and the prediction was not good at all.

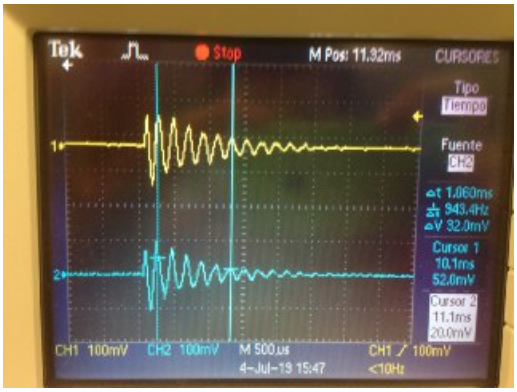
Given these prediction results, some measurements have been done in order to check if the microphone and the loudspeaker are working fine. First of all, a $5kHz$ sinusoidal signal has been created with a signal generator and emitted using the piezoelectric loudspeaker. Then, the signal has been received using two different microphones (the first one is the GRAS used in the project and the other one is a Bruel and Kjaer). The power source is also different: the GRAS microphone has been connected to the GRAS power source, while the Bruel has been connected to a Bruel power source (this one admits more than channel). The reconstructed signal has been visualized in the oscilloscope. It has been found that under the same conditions, the GRAS signal shows an amplitude of $V_{pp} = 138mV$ and the Bruel signal has an amplitude of $V_{pp} = 704mV$. This difference is even more striking because the sensitivity of the GRAS microphone is $4mV/Pa$ and the sensitivity of the Bruel microphone is $2.74mV/Pa$.

Next test has been the same as before but this time the GRAS power source has been replaced, and both microphones have been powered using the Bruel power source. This time, GRAS signal amplitude is $V_{pp} = 1.72V$ and Bruel signal amplitude is $V_{pp} = 0.872V$. Thus, the conclusion of these two tests is that the Bruel power supply amplifies the signal more than the GRAS power supply, but both microphones work fine for this signal.

But the emitted signal in the real system is not a sinusoidal signal, but it is a really fast squared signal. Then, the following signal has been emitted and received:



(a)



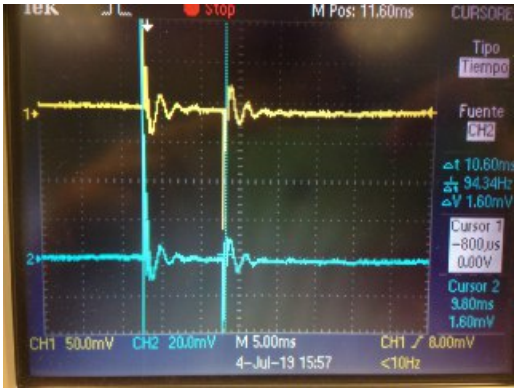
(b)

Figure 7.8: Squared signal using Kemo L10 as transducer.

Signal shown in **Figure 7.8** shows the response of the transducer to an step, that is to say, the step response of the piezoelectric. This response has a frequency of $5kHz$ and a time duration of $1ms$. Knowing that the emitted signal oscillates faster than $1ms$, the signal in the microphone will be blurred by the impulse response of the piezoelectric.

Finally, the same test has been performed using another loudspeaker, and the result is the following:

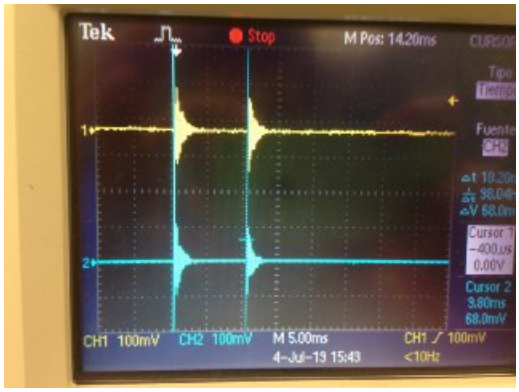
(a)



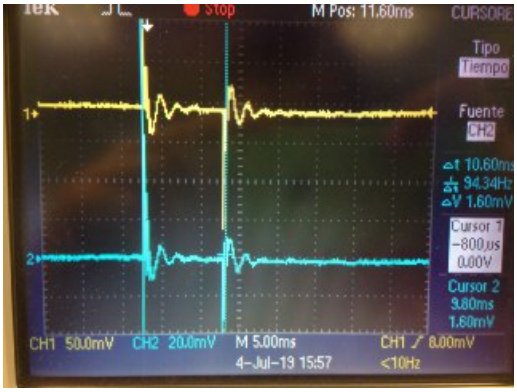
(b)

Figure 7.9: Squared signal using other transducer.

As seen in **Figure 7.9**, the step response of this transducer has less oscillations than the previous one. In order to compare both loudspeakers, take a look at **Figure 7.10**:



(a)



(b)

Figure 7.10: *Comparison between step responses.*

It be outlined that the left one is shorter in time but has more oscillations, whereas the right one is longer but has less oscillations.

Chapter 8

Conclusion

The implementation of a single, autonomous, integrated electronic device to reconstruct acoustic images with a single microphone has been done. A microcontroller has been programmed and all the peripherals have been included. Moreover, a theoretical model has been designed and it has been proved using synthetic (or simulated) data and a machine learning algorithm that a solution to the problem exists.

About the implementation and the experimental results, they are not good enough. The prediction does not present reliable results and, as shown in **Section 7**, the step response of the piezoelectric transducer used as a loudspeaker might not be appropriate to go as fast as it is intended in the project. Therefore, the loudspeaker should be replaced by one having a wider bandwidth (a shorter impulse response). Electronic restrictions to the problem may be more constraining than it was supposed at the beginning of the project. Moreover, experimental results have been done using an Arduino Mega 2560, while in **Section 6.1.2** a long discussion on which device should fit better the system has been done, and Teensy 3.1 has been considered even a better option than Arduino Mega 2560.

Electronics might not be the only thing to fix in order to reconstruct the image. From the recovered signals, it seems that there is a lot of noise in the system. A noise source study might be done in the future to filter them out. Furthermore, it could be possible to find a mismatch between theoretical model and real one. In **Appendix D** several assumptions have been made, and also some approximations that may be important in the case of the real system. The way this mismatch could be checked is by simulating the behavior of the system using a FEM software such as COMSOL Multiphysics to verify that the theoretical model matches the simulated one.

Once the measurements give good experimental results with the system, the proof of concept will be done, and the system is supposed to work both for a higher number of pixels and for a smaller scale. Thus, a new system will be designed, keeping the same algorithms and architecture, but this time focusing on ultrasound transducers, to go far beyond from audible frequencies, and to reduce pixel size to micrometers or even nanometers. It means that ultrasound emitters and receivers will be selected and some changes in the system stated here will be done. This is the real goal of this technology: to reconstruct acoustic image at nanoscale.

Chapter 9

Project's planning

This chapter shows how the planning of the project has been done and how it has ended. In fact, the work discussed and explained in the report corresponds to the *Task 4* of the Gantt diagram shown below. As said before, the implementation of a real and autonomous imaging system belongs to a more wide project, where the physics model (**Appendix D**) and the reconstruction algorithm (**Appendices B and C**). In the diagram, *Task 1* is the bibliographic research and study of both the physics model and the reconstruction algorithm. *Task 2* is the statement and simulation of the physics model in MATLAB, as well as the production of the data set that will be used to train and check the reconstruction algorithm. Third step, or *Task 3* is the development of the Artificial Neural Network model and its validation. *Task 4* is the implementation of the physical system, the aim of this report. Finally, *Task 5* is the writting of the report.

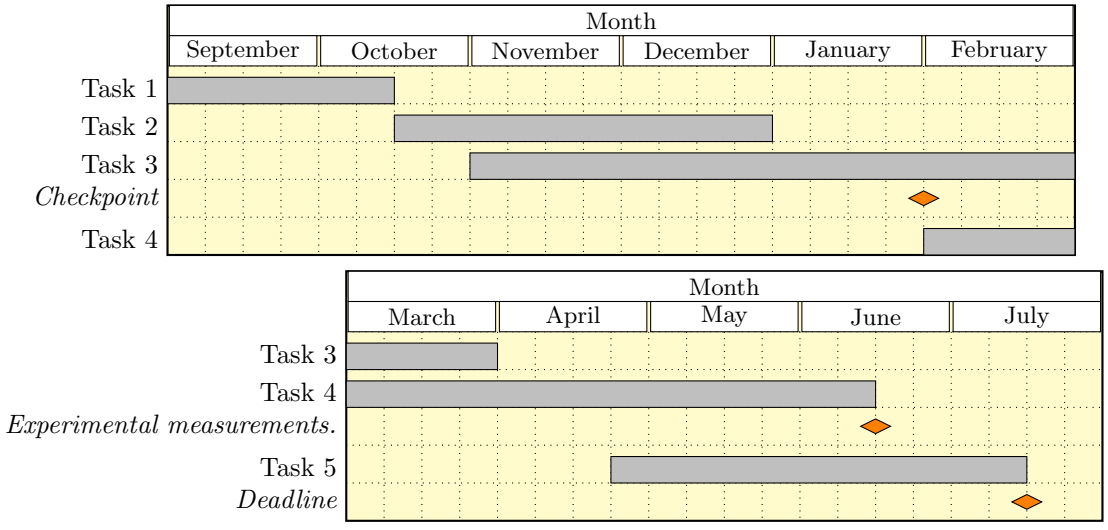


Figure 9.1: *Project Gantt's diagram*

Taking a look more in detail to the implementation part, that is to say, *Task 4*, it can be divided in the following subtasks:

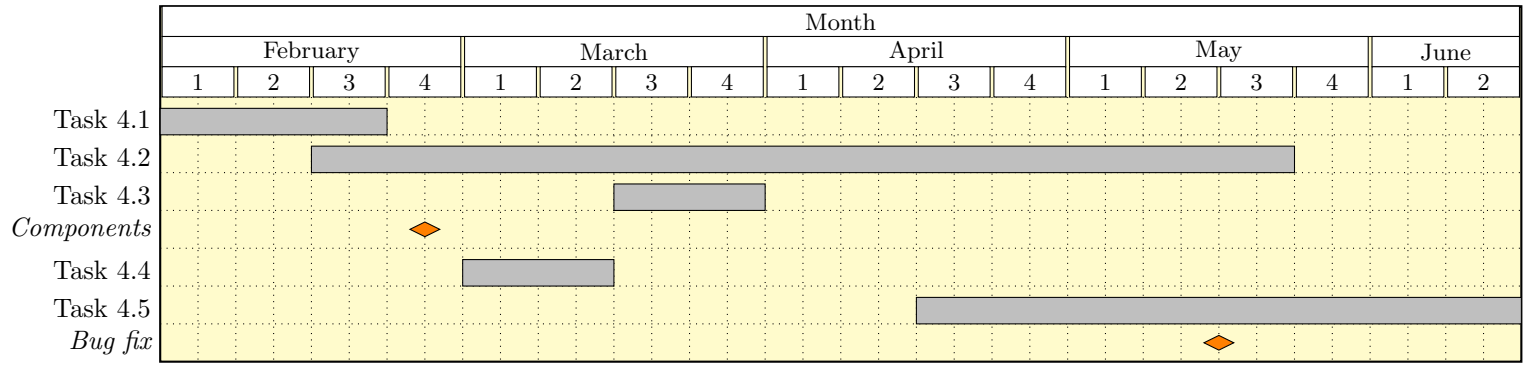


Figure 9.2: *Gantt's Task 4 detail.*

Task 4.1 is the component selection, where all the material needed to implement the system is selected and purchased. *Task* 4.2 is code development. This is the longest subtask, as it is really linked to the other tasks and will be modified in consequence. *Task* 4.3 is electronic circuit design. This includes both the conception and the implementation of the electronics. Thus, this subtask has experimental work. *Task* 4.4 is the plate's design and fabrication. Finally, *Task* 4.5 are all the experimental tests that are needed to check the performance of the code and its interaction with the hardware, before starting the experimental measurements that will give the results. Concerning the milestones, the first one is the estimate arrival of the selected components, and the second one is the date where all bugs in the code will be fixed.

The available resources in the project have been stated along the report, but are going to be summarized here, adding extra information about its availability.

Human resources are crucial for any kind of project. This project, as part of a university department research program, has few people working on them. Two groups were initially involved; GROC (Grup de Recerca d'Òptica de Castelló) UJI and INIT (Institut de Noves Tecnologies de la Imatge). Daniel Torrent Martí has been the GROC member involved in the physics part of the project, while José Martínez Sotoca and Vicente Javier Traver Roig have been involved in the reconstruction algorithm. They have been present in all the meetings the project has had, as well as Marc Martí Sabaté. Vicente Javier Traver has been on stay in Southampton the second semester of the academical year. In February, another person has been included in the project, supervising the implementation task. This person is Ignacio Peñarrocha Alós, academical supervisor of the project. Marc's availability in the project has not been the same in all the project. Until February, Marc has been working part time in the project. From February to June, he has been working full-time, and the last two months he has been working part time again.

Non human resources in the project have different natures. First of all, this project has been funded from INIT with 1000€. Two different computers have been used, one from the beginning of the project and *alien5* since February (for technical details please refer to Specification Part in **Chapter 11**). As for the software, MATLAB licences were available, and Spyder3 and Arduino IDE where installed in the computers (both softwares are open access). Moreover, the components discussed in **Section 6.1.1** where already available at the beginning of the project.

After having finished the project, a few comments can be told about the initial planning and the development of the project. Concerning *Task* 4, which is the main *core* of the project, *Task* 4.1 has suffered some delays. The selection of the microSD shield has been delayed since there were doubts about which microcontroller would fit better in the project. Also, there has been a delay in *Task* 4.4 because the first structured plate was deformed due to sun exposure. Returning to the hole project, *Task* 3, that is to

say, the reconstruction algorithm, has also suffered delays. Even though, the scope of the project has not been affected or changed.

Chapter 10

Order

The order of priority of the documents in the project is established by the norm UNE-EN 157001:2014, that is to say:

1. Specifications.
2. Budget.
3. Report.
4. Appendices.

Part II

Appendix

Appendix A

Electronic measurements

A.1 Loudspeaker adapting circuit

Before using a driver electronic circuit, loudspeaker supply features must be analyzed. Loudspeaker technical data gives an estimate for the maximal voltage it is able to accept: $16Vp-p$. However, there is neither information about current consumed by the piezoelectric nor information about its impedance.

Next figure shows the electronic circuit that has been used to get information about current and impedance:

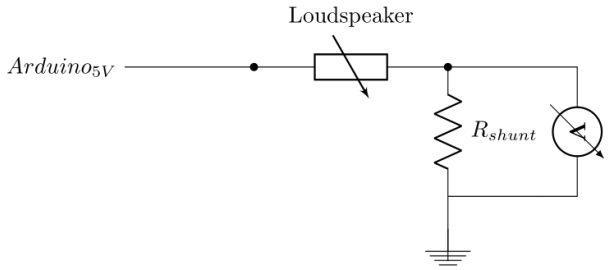


Figure A.1: *Shunt resistor set up.*

Several resistors have been tested until a difference in voltage in the voltmeter has been noticed. Finally, a $R_{shunt} = 47k\Omega$ resistor has been used. This is completely unusual; usually, shunt resistors are really small resistors, but as it will be seen in a moment, loudspeaker's impedance is huge, and then the current is so small that using a $R_{shunt} = 1\Omega$ does not allow to get any measurement in the voltmeter (the voltmeter is not able to measure under $0.001V$). Using this shunt resistor, the obtained voltage is $V_{R_{shunt}} = 0.023V$, meaning that the current going through the circuit is $I = 489nA$. Now, the loudspeaker impedance can be easily estimated, and the computed impedance is $Z_{loud} = 10.23M\Omega$. Finally, the current that would go through the circuit in case of replacing the shunt resistor by a wire is $I_{max} = 491nA$, which is a current that Arduino's digital output can provide.

Even if loudspeaker could be commanded directly using Arduino's digital output, when there is no selected output for the pin, the state of the output is called *high impedance*, and there is an uncertainty that can make the output oscillate and then undesired noise would be created. In order to avoid this phenomena,

an electronic driver can be used.

Figure 6.13 from **Section 6.1.7** in the report shows the simplest electronic circuit to perform a driver for the loudspeaker. The MOSFET's model is IRF520. U_{out} is the output voltage from the microcontroller, which varies between $0V$ and $5V$. This voltage will command the MOSFET transistor by changing V_{GS} voltage. Here, the MOSFET acts as a switch; its state will be either saturation, or cutoff. During saturation state, V_{DS} is almost 0, then, the loudspeaker has $15V$ between its terminals. During cutoff, there is no current going through the loudspeaker, and the voltage is 0. In order to command the MOSFET transistor, V_{GS} must be higher than V_{GSth} during saturation, and lower in the other case. As the MOSFET's source is directly connected to ground, $V_{GS} = V_G$. The command could be easily done by connecting the microcontroller's output directly to the MOSFET's gate, as $5V$ is high enough to activate the transistor. However, as said before, the "high impedance" state must be considered. Then, when Arduino's output pin does not have a value, MOSFET's gate is connected via R_1 to ground, forcing the voltage to be 0. R_1 resistor is big enough to ensure that the current is lower than the Arduino's pin limit ($20mA$). As gate's input current can be neglected:

$$I_{max} = \frac{V_{out}}{R_1} \tag{A.1}$$

By placing a $1M\Omega$ resistor, current is ensured to be under the limit. Loudspeaker's voltage during MOSFET's saturation is not exactly $15V$; there is always a little tension fall between Drain and Source in the MOSFET. In order to get an idea of the magnitude of this tension, the technical documentation gives some values, but it can also be checked experimentally, replacing the loudspeaker by a resistor and measuring the current that goes through it, as shown in **Figure A.2**.

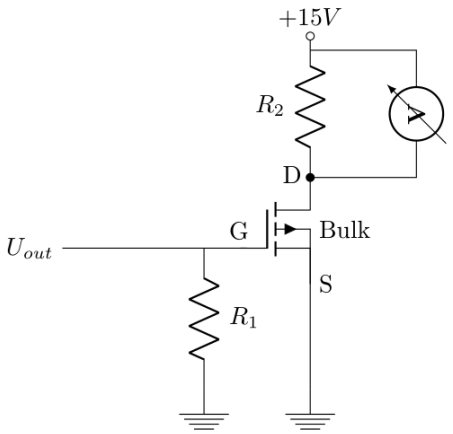


Figure A.2: *Electronic set up to measure current through the drain.*

Using a $R_2 = 2.2k\Omega$ resistor, a $6.7mA$ drain current has been found, meaning a voltage of $14.74V$ at loudspeaker terminals, and $V_{DSsat} = 0.26V$. Furthermore, using a voltage source instead of Arduino's output at U_{out}, V_{GSth} can be determined. It has been found that $V_{GSth} = 3.6V$.

The main drawback in this circuit is the fact that a $+15V$ asymmetric power supply is needed, which is three times the voltage needed for the microcontroller. This implies using a more complex power source, even needing a transformer and some power electronics. As said in the report, this idea has been discarded, according to the requirements of size, autonomy and price.

A.2 Microphone adapting circuit

Figure 6.14 in the report has shown that a processing chain is needed to adapt microphone's signal. The first step is the high-pass filter. As the content that is wanted to be eliminated is the continuous offset, it is enough to use a first order filter [1], using a resistor and a capacitor. The cutoff frequency $(-3dB)$ is placed between $10Hz$ and $100Hz$. Filter's expression is:

$$V_{out} = V_{in} \cdot \frac{j\omega R_f C_f}{1 + j\omega R_f C_f} \tag{A.2}$$

And f_0 is defined as $f_0 = \frac{1}{R_f C_f}$. Choosing a $100nF$ capacitor, resistor must be in the interval $[16-160]k\Omega$. Considering two series resistors $15k\Omega$ and $2.2k\Omega$, the cutoff frequency is $92.53Hz$.

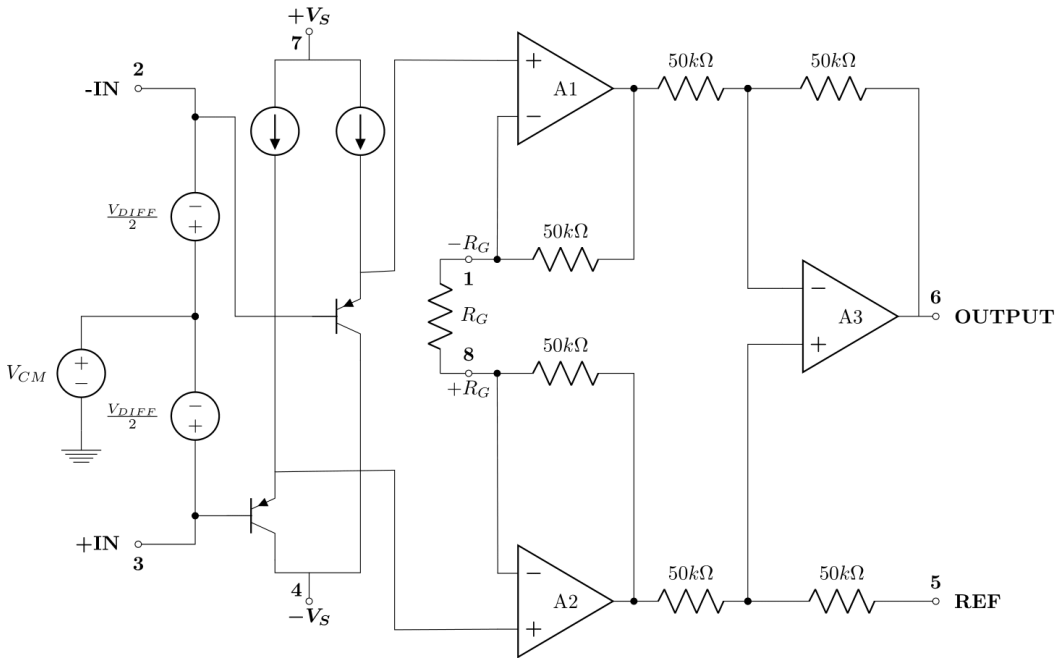


Figure A.3: *AD623AN inner diagram.*

Next, the instrumental amplifier used is AD623AN. It is the widely used amplifier made up with three operational amplifiers, as shown in **Figure A.3**. Its gain is defined as $G = 1 + \frac{100000}{R_G}$, begin R_G the resistor that fixes the gain, and it's up for the user to chose its value. Knowing that the input's amplitude is $0.1V$ and that the signal is going to be placed at an offset of $2.5V$, the output amplitude of the signal should be $2.5V$ to profit all the microcontroller's input range. This means an amplifier gain of $G = \frac{2.5}{0.1} = 250$, being then the resistor $R_G = 401.6\Omega$. The chosen resistor will be the immediately higher resistor in the market, in order to get a gain a bit lower (to avoid saturation in the microcontroller's ADC). This resistor is $R_G = 470\Omega$. The filter's output signal will be placed as the non inverting input of the instrumental amplifier, and the ground will be the inverting input. As the input impedance of the instrumental amplifier is directly the input impedance of an operational amplifier (thus, a really high

[1] Moreover, it is a passive filter, which gives less problems to implement than an active filter. An active filter ought to have a power source.

impedance, usually higher than $1M\Omega$), it does not disturb the previous filter, and the previous expression (equation (A.2)) is still valid.

This model of instrumental amplifier allows symmetric power supply with voltage values between $\pm 6V$. It also accepts asymmetric power supply up to $12V$. Asymmetric power supply has been chosen.The output signal will be limited to the supply voltage. In order to protect the microcontroller against overvoltage, the supply voltage will be limited to $5V$. Also, a $10k\Omega$ potentiometer will be placed in the *Offset* input to change the mean value of the signal and place it in the middle of the input's range. The potentiometer will be connected to $+5V$ and $0V$ to give a voltage in between those values. Theoretical expressions give a voltage to pass as input to the offset input; however it is easier and more useful to place a potentiometer and to look for the value experimentally.

Nevertheless, the input signal at point 2 in **Figure A.3** is positive and negative. If the power supply is asymmetric, the negative part of the input signal will not be processed, and thus, an important part of the information will be lost. The system is intended to be powered by a single battery, to make the system autonomous and small as possible. Therefore, this electronic circuit is not the best solution for the system.

Another solution is presented in the diagram in **Section 6.1.7, Figure 6.16** of the report. This time, the input signal is biased by an offset before passing through the operational amplifier, thus this amplifier is able to have asymmetric power supply without cutting the input signal. The circuit's function is

$$V_{Out} = (1 + \frac{R_F}{R_X}) \cdot (\frac{1}{R_1 + R_2}) \cdot (R_1 \cdot V_{Off} + R_2 \cdot V_{in}). \tag{A.3}$$

Knowing that V_{in} has an amplitude of $0.2V$ and no offset, the output signal must have an amplitude of $2.5V$ and a $2.5V$ offset and fixing the offset voltage at $3.3V$ (it has been fixed this way because Arduino is able to provide this voltage) the following system of equations can be established:

$$2.5 = (1 + \frac{R_F}{R_X}) \cdot (\frac{1}{R_1 + R_2}) \cdot (R_2 \cdot 0.2) \tag{A.4}$$

$$2.5 = (1 + \frac{R_F}{R_X}) \cdot (\frac{1}{R_1 + R_2}) \cdot (R_1 \cdot 3.3) \tag{A.5}$$

From this system of equations and using commercial resistor values, it has been found that using $R_F = 12k\Omega$, $R_X = 1k\Omega$, $R_1 = 51\Omega$ and $R_2 = 820\Omega$ gives a good performance: a 2.45 gain and a $2.51V$ offset. Then, this electronic circuit fits the requirements of the system: it achieves a signal covering the whole Arduino input range and it allows an asymmetric power supply ($5V$ and 0). This asymmetric power supply, again, protects the microcontroller against overvoltage, as the output signal cannot be higher than $5V$. As operational amplifier, a single supply operational amplifier must be used. LM741 does not allow single supply, it has to be symmetric power supply. The chosen model is **LM324**, even if it has four amplifiers inside, and only one is needed for the project's purpose, it allows single supply ($5V$ and $0V$).

A.3 MATLAB measurement simulation

As seen in **Section 6.2.2** of the report, one single measurement is not enough to obtain the desired frequency vector. Therefore, the following simulation has been done:

```
M = 0;
i = 0;
freq_acum = [];
moment=1;
fs_list = [84488, 79850, 76785, 74079, 71508, 69003, 66701, 64646];
while M<512

    fs = fs_list(i+1);
    N = 512;
    freq = -(fs)/2:(fs/N):(fs)/2-(fs/N);
    freq = freq(freq>=0);
    freq_coef = freq;
    freq_coef(freq_coef>20000)= NaN;
    freq_coef = (1:length(freq_coef)).*(~isnan(freq_coef));
    freq_coef = freq_coef(freq_coef>0);
    freq = freq(freq<=20000);

    measure.fs{i+1} = fs;
    freq_acum = [freq_acum,freq];
    K = [];
    for ii=1:length(freq_acum)

        for jj=ii+1:length(freq_acum)

            if freq_acum(ii)==freq_acum(jj)
                freq_acum(jj) = NaN;
                K = [K, (jj - M)];
            end
        end
    end

    for ii=1:length(K)
        freq_coef = freq_coef(~(freq_coef==K(ii)));
    end

    measure.fcoef{i+1} = freq_coef;
    measure.freq{i+1} = freq(freq_coef);
    freq_acum=sort(freq_acum(~isnan(freq_acum)));
    measure.Ncoef(i+1) = length(freq_acum) - M;
    M = length(freq_acum);
    i = i + 1;
end

measure.Nfs = i;
measure.M = M;

for jj=1:length(measure.freq)
    figure(9),plot(measure.freq{jj}, ones(1,length(measure.freq{jj})),'o'),hold on;
end

vec_aux = 1:length(freq_acum);
for ii = 1:measure.Nfs
    measure.ind_acum{ii} = [];
    for jj = 1:length(measure.freq{ii})

        measure.ind_acum{ii} = [measure.ind_acum{ii}, sum((freq_acum == measure.freq{ii}(
    jj)).*vec_aux)];
    end
end
```

Matlab_code/exp_measure.m *excerpt*

In this MATLAB excerpt, experimental sampling frequencies are sorted in descending order in an array. Then, for each sampling frequency, the frequency vector that would be obtained after FFT algorithm is computed, and those frequencies inside the desired bandwidth are kept, while the others are discarded. Then, frequencies are accumulated in a buffer, and repeated ones are deleted. At the end of the iteration, the number of elements in the buffer is computed and if it is lower than 512, the next sampling frequency is taken an another measurement is done. This code can be understood easily by looking **Figure A.4**.

Not only frequencies are saved to count the amount of spectral coefficients achieved. A whole structure is saved; it contains for each sampling frequency, the amount of coefficients saved, the frequencies, the frequency indices of saved coefficients, and even the indices of those coefficients in the final whole frequency vector. All these parameters will be necessary in order to manage the arrays in the microcontroller and save values in the SD card, as discussed in **Section 6.2.3** in the report.

Results from this simulations are the following: four sampling rates are needed (76.8, 74.1, 71.5 and $69.0kHz$). For the first sampling rate, 121 coefficients are obtained; 126, 130 and 135 for the others. The total amount is 512 coefficients.

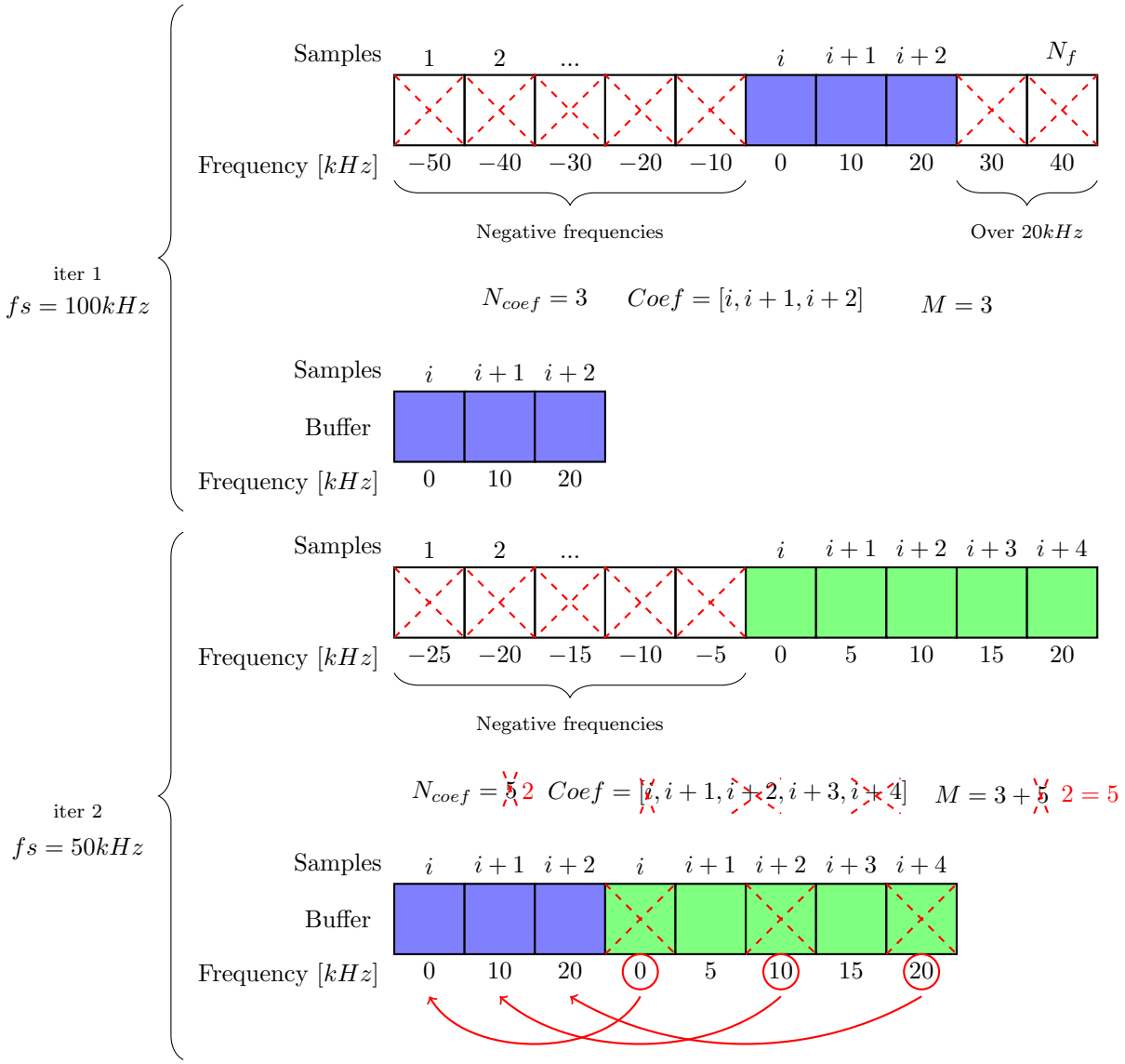


Figure A.4: *Algorithm graphic's explanation.*

Appendix B

Linear regression test

In this Appendix, the least squares algorithm as a reconstruction algorithm is going to be considered. First of all, consider one single measurement as the linear combination of its frequency components, such that

$$y_1 = \theta_1 \cdot f_1 + \theta_2 \cdot f_2 + \cdots + \theta_k \cdot f_k + a. \tag{B.1}$$

In this expression, y_1 means the first output of the plate, which can be either 1 or 0. f_i parameters are the spectral coefficients experimentally measured, and θ_i are the coefficients that link the input and the output. As these last coefficients are unknown, the aim is to solve them and to map spectral information with hole's presence. Now, suppose that instead of having one single observation, a N list of them is available, being N a higher number than the parameters to solve, k. Then, the equation above can be written as

$$\underbrace{\begin{bmatrix} y_1^1 \\ y_1^2 \\ y_1^3 \\ \vdots \\ y_1^N \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & f_1^1 & f_1^1 & \dots & f_k^1 \\ 1 & f_1^2 & f_2^2 & \dots & f_k^2 \\ 1 & f_1^3 & f_2^3 & \dots & f_k^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & f_1^N & f_2^N & \dots & f_k^N \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_{k+1} \end{bmatrix}}_{\mathbf{\Theta}}, \tag{B.2}$$

being Θ the array of parameters to solve, X the input and Y the output. Up to this moment, only one single hole has been considered, but this system can be extended to the other holes. Thus, the final expression is

$$\underbrace{\begin{bmatrix} y_1^1 & y_2^1 & y_3^1 & \dots & y_h^1 \\ y_1^2 & y_2^2 & y_3^2 & \dots & y_h^2 \\ y_1^3 & y_2^3 & y_3^3 & \dots & y_h^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_1^N & y_2^N & y_3^N & \dots & y_h^N \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & f_1^1 & f_1^1 & \dots & f_k^1 \\ 1 & f_1^2 & f_2^2 & \dots & f_k^2 \\ 1 & f_1^3 & f_2^3 & \dots & f_k^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & f_1^N & f_2^N & \dots & f_k^N \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \theta_1^1 & \theta_1^2 & \theta_1^3 & \dots & \theta_1^h \\ \theta_2^1 & \theta_2^2 & \theta_2^3 & \dots & \theta_2^h \\ \theta_3^1 & \theta_3^2 & \theta_3^3 & \dots & \theta_3^h \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \theta_{k+1}^1 & \theta_k^2 & \theta_k^3 & \dots & \theta_k^h \end{bmatrix}}_{\mathbf{\Theta}}. \tag{B.3}$$

Each column in Θ's matrix represents the parameters to estimate the output of each pixel of the image. As N is bigger than k, this matrix system is an overdetermined system of equations. The system is going to be solved using a least squares algorithm by means of the Moore-Penrose pseudoinverse matrix. Therefore, Θ can be found as $\Theta = (X^T X)^{-1} \cdot X^T \cdot Y$.

Using a 60000 observation data set with both input and output, 20000 observations have been used to solve and identify the parameters by means of the least squares algorithm, and the rest are used for validating the algorithm and computing an error. MAE error has been computed for each pixel, and it is shown in **Figure B.1**.

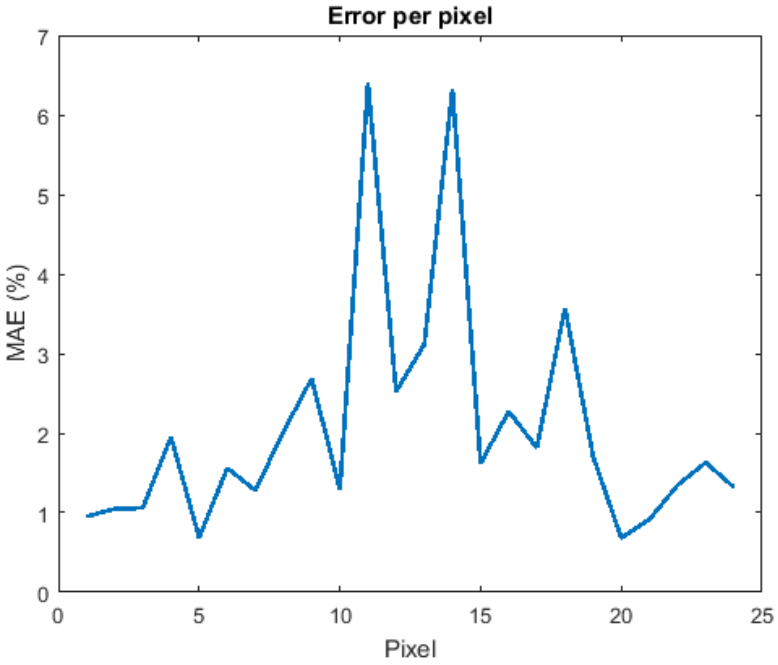


Figure B.1: *Accuracy evolution depending on input size vector.*

The results show that the least squares algorithm finds out a good result for the model. However, there are some pixels where the error is several times higher than for other pixels. Even if these results seem quite good, it has been decided to add a non linearity to improve the performance. As it is known, output values y_i^j are either 1 or 0. Nevertheless, the result of the linear regression is not a Boolean value, but a real one. By comparing each prediction to a threshold, the output can be converted into 0 and 1 values. A 0.5 threshold value for each pixel has been chosen. As a result, the error per pixel curve shown above becomes and straight line centered at zero, meaning that this time, the real output is always found.

A more deep insight analysis has been done on the data's nature. First of all, several simulations have been executed. A data cube has been obtained, where one dimension represents identification data size (N), the second one is the pixel evaluated (output y_i), the third one is the frequency (f_i), and the magnitudes are θ_i. Visualizating these data for a single pixel, an image as follows is obtained:

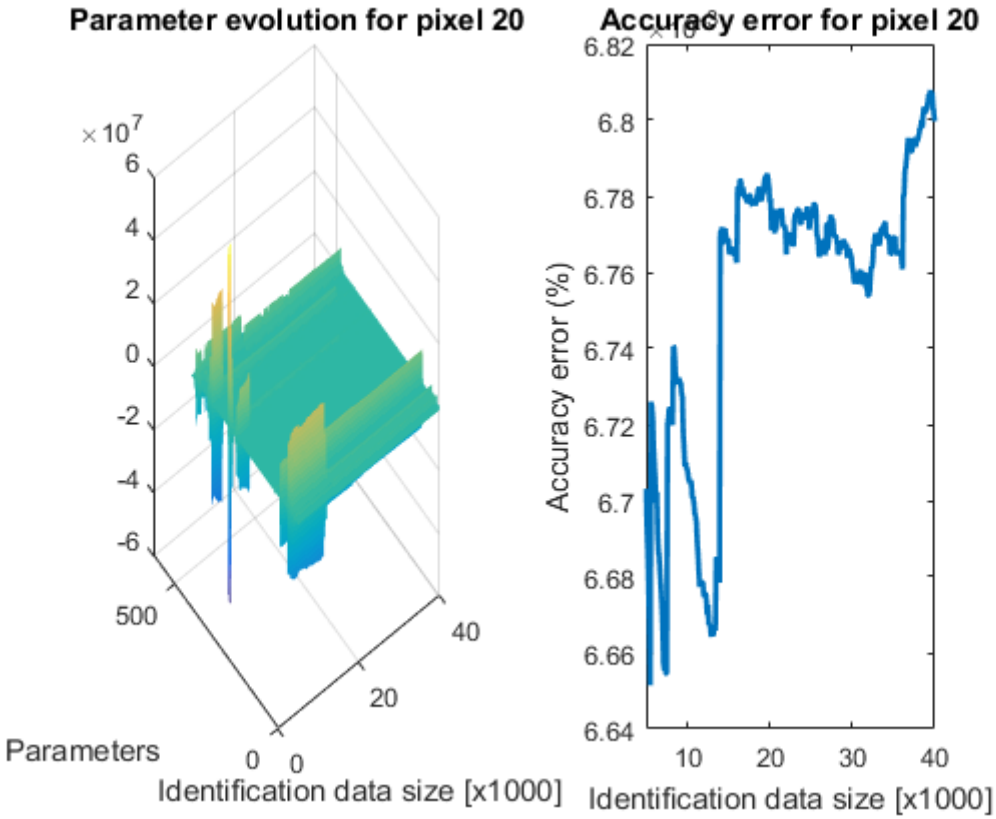


Figure B.2: *Accuracy evolution depending on input size vector.*

As it can be seen in **Figure B.2**, weights' magnitude is barely independent to the number of observations used for identifying the system. There are several parameters whose value is around 0 with independence of how big is the identification set. Moreover, the accuracy error is almost the same for every identification set size. Also, there is a very relevant conclusion that can be outlined from this figure. In fact, there are some frequencies whose coefficients are always 0; then, they are irrelevant for the system's identification. These frequencies have been searched in order to compare with the irrelevant frequencies for the other holes. A surprising result has been found: the irrelevant frequencies are the same for the identification of every hole, that is to say, these frequencies do not help solving the problem. Consequently, if the least squares algorithm is computed again eliminating the information concerning these frequencies, the same result should be retrieved. And, indeed, the same result is found.

Finally, the covariance matrix of the measurement matrix (X) is computed. It has been found that there is a high peak value in this matrix that shades the rest of the coefficients, making difficult to visually analyze it. Therefore, the correlation matrix has been computed. Remember that the correlation matrix is just a normalization of the covariance matrix, given by

$$R_{x,y} = \frac{S_x \cdot S_y}{S_{x,y}}, \tag{B.4}$$

where S_i is the variance of the i-th variable, and $S_{i,j}$ is the covariance between i-th and j-th variable.

Figure B.3 shows the absolute value of the correlation matrix. Correlation coefficients show the relationship between two different variables: they are near 1 when there is a direct relationship, nearby -1 when there is an inverse relationship, and 0 when there is no relationship. By showing the absolute value of those coefficients, the image focuses on whether there is a relationship between variables or not.

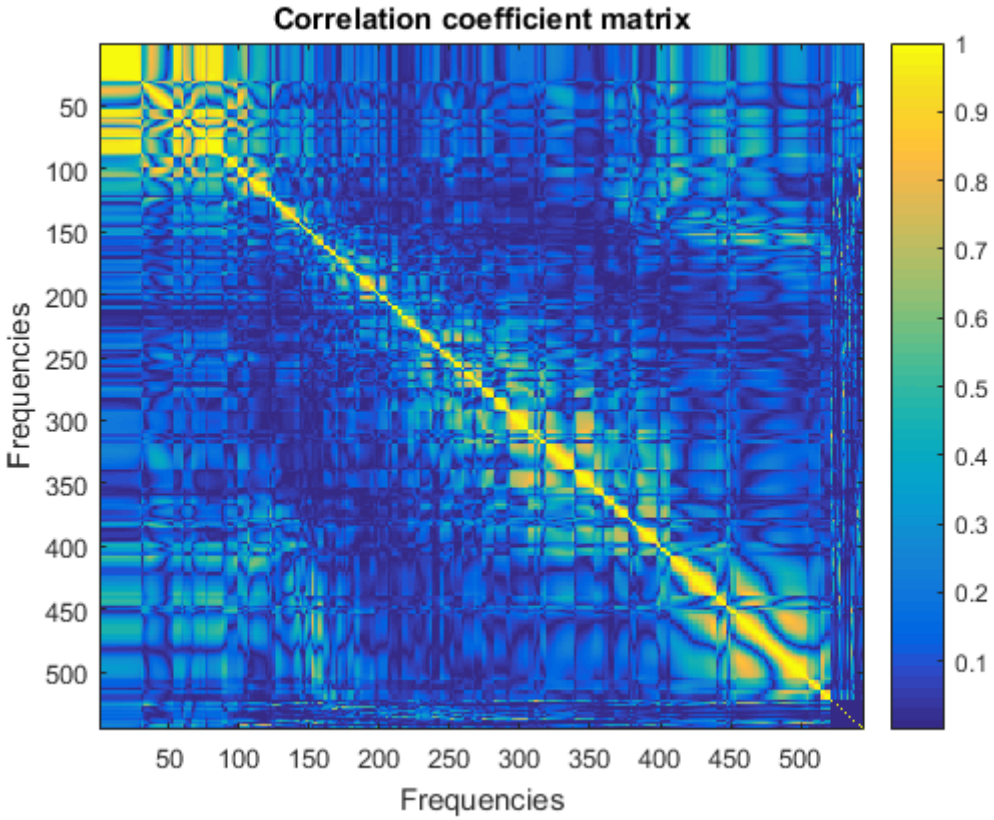


Figure B.3: *Correlation coefficient matrix.*

In **Figure B.3** there is the typical diagonal of 1 values, as every variable is directly correlated to itself. Out of this diagonal, however, there is no clear behavior of the frequencies. At low frequencies, it can be found that there is a group that behaves similar respect to the other frequencies. This means that information in this band can be reduced to a single frequency in the interval without losing information. Nevertheless, this approximations are little, and there is no clear band behavior in the data.

The linear regression algorithm starts to fail when the number of pixels starts to increase, that is to say, the complexity of the system increases. As shown in **Section 6.2.4** of the report, the error per pixel using the linear repression plus the 0.5 threshold has increased. Remember that for the 5×5 plate, the accuracy error per pixel was 0%; now, it is in average higher than 10%.

Appendix C

Artificial Neural Network

C.1 Neural networks: a historical approach

Deep Learning has suffered a meteoric rise over the last several years, characterized by drastic improvements over reigning approaches towards the hardest problems in Artificial Intelligence, massive investments from companies such as Google, and an exponential growth in research publications. However, before Deep Learning, there were several techniques in the topic of Machine Learning that lead to the actual state.

First of all, as Machine Learning is the framework where Artificial Neural Networks and afterwards Deep Learning were developed, it must be understood. Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959). [10] Russell and Norvig, in [21], define machine learning as a necessary part of Artificial Intelligence, the process by which the system learns to adapt to new circumstances and to detect and extrapolate patterns.

The most important idea involving Machine Learning is the generalization principle: it is not only important to learn the answer to data passed as training to the system, but it is also crucial to predict outputs from data never seen before. Thus, it is common to find data split into a training set and a test set of data. The main drawback of training algorithms is overfitting, which can be understood as the opposite of the generalization principle; the system can resolve training data with good accuracy, but it is not able to reach this performance for unseen data.

The first idea specifically related to Machine Learning methods appeared in 1957, published by Frank Rosenblatt as the **Perceptron** [19]. He conceived it as a simplified mathematical model of how the neurons in the brain operate; basically, it takes a set of binary inputs (nearby neurons), multiplies each input by a continuous valued weight (synapse strength), and thresholds the sum of these weighted inputs to output either a 1 or a 0 (simulating the fire of a neuron). Before this definition, Mcculoch and Pitts had shown that this mathematical model could model the basic logic functions (OR, AND and NOT). The breakthrough given by Rosenblatt was a way to make such artificial neurons learn, inspired by Hebb's work. Hebb introduced the idea that knowledge and learning occurred in the brain by the formation and change of synapses between neurons. Given a training set of input-output examples the Perceptron should learn a function by increasing the weights if the output is lower than the expected one, or inversely, decreasing the weights when the output is higher than it should be.

This procedure produces the following result: the weighted sum is just as a linear regression, followed

by a non-linear activation represented by the thresholding. When the output has a finite set of values it is fine to apply the threshold. Then, the problem can be seen not as a generation of continuous-valued output, but as a classification of a correct label. It is not possible for a single Perceptron to classify data with many categories. To achieve this goal, many Perceptrons are arranged in a layer, such that all receive the same input, and each of them is in charge of a different output of the function. Artificial Neural Networks are in fact formed by layers of Perceptrons. Up to this moment, only one layer existed.

Nevertheless, Artificial Neural Networks can be conceived with neurons that differ from the Perceptron. In 1960, Widrow and Hoff explored the option of use the weighted sum directly as the output, eliminating the thresholding function (ADALINE neuron) [9] . They found that the learning mechanism could be based on minimizing the error between the weighted sum and the output. The derivative of this error can be used to drive the error down and find the optimal weight values.

Despite of the increasing interest in this techniques to solve AI problems, the Perceptron had its limits. For example, it could not learn the boolean XOR function because this function is not linearly separable. These limitations frozen the research in this topic for several years.

The main idea of Artificial Neural Networks was to combine bunches of neurons to solve problems. Instead of using one simple output layer as the previous works, applying several hidden layers, as their output can work as an input for the following layer. The key point of using hidden layers is that they are able to find features within the data. Therefore, it was necessary to use these kind of architectures to solve problems such as face recognition. However, Rosenblatt's learning algorithm did not work for multiple layers.

The answer to this problem was stated in the early 60's, implemented as it is known today in 1970 by Linnainmaa [25]. The idea of the **backpropagation** algorithm is to propagate the error from the output layer to the hidden layers by means of the chain rule. Thus, the non-linear activation function of the hidden layers must be differentiable. Using this backpropagation algorithm, an optimization technique can be used to find the optimal weights to minimize the error.

As a consequence of the backpropagation algorithm and its application to multilayer neural networks, the topic became popular again. In 1989 another key point was published: *"Multilayer feedforward networks are universal approximators"* [13]. It was proofed that multilayer neural networks were able to implement any function. Also in 1989, Yann LeCun et al. demonstrated a real-world application of backpropagation in *"Backpropagation Applied to Handwritten Zip Code Recognition"* [14]. In this work, they stated the first example of convolutional layer, which would end up by defining Convolutional Neural Networks.

Up to this moment, all Artificial Neural Networks were trained using input-output pairs. This kind of training is known as **supervised learning**. In the 90's, the firsts ideas of **unsupervised learning** methods appeared; applied for example to encoders, clustering, Self Organizing Maps or Adaptative Resonance Theory. In the same decade, the third branch of machine learning was also studied with Artificial Neural Networks: **reinforcement learning**, which can be easily explained as learning to make good decisions. Whereas supervised learning tells what it should learn to output, reinforcement learning provides "rewards" as a by-product of making good decisions over time. Artificial Neural Networks started to be applied in many different fields, such as Electrical Engineering as adaptive filters or for identification and control of dynamical systems, or in robotics for autonomous land vehicles.

In 1989, Waibel introduced time-delay neural networks in order to solve speech recognition problems [24]. These networks were very similar to normal neural networks, except that each neuron processed only a subset of the input and had several sets of weights for different delays of the input data. Nevertheless, this kind of neural networks were surpassed by another approach in speech recognition problems: recurrent neural networks. The main feature of this architecture is the fact that the output of a layer is not only the

input of the following layer, but also the input of the precedent layer or even the same layer. Therefore, the problem of giving the network memory as to past inputs was solved. In this case, the backpropagation algorithm had to be fixed to work with this architecture; backpropagation through time. However, even this neural network did not succeed in training speech recognition with accuracy.

By the end of the decade, it was found that backpropagation algorithm did not work well for networks with a lot of layers, and the results were not as good as for a network with fewer layers. The reason is that backpropagation relies on finding the error at the output layer and successively splitting up blame for it for prior layers. When the number of layers is high, this algorithm ends up with either huge or tiny numbers and the resulting neural network does not work very well (*"vanishing of exploding gradient problem"*) [22].

A few years later, in 2006, Hinton published what it is considered as the first Deep Learning paper [11]. In this article, they found a fast algorithm that could train artificial neural networks with several layers by initializing their ways in a clever way rather than randomly. Other advancements that were made in the following years included the discovery of **dropout** as a way of training neural networks avoiding overfitting, and the study of non-linear activation functions performance. It was found that **ReLU function** is the best function in order to train the network without having the "vanishing gradient" problem.

Nowadays, big technological companies have engaged lots of resources in Deep Learning; fast processors, GPU parallel computation, huge data sets, and smart architecture and algorithms mix together to give the world better performance in problems that 20 years before were not affordable.

C.2 Need of a machine learning solution

In **Appendix D**, the physical equations of the problem are explained for the direct problem. However, the situation that is needed to be solved is the inverse problem. The main idea is to establish which holes in the structure are present or not in a given moment, understanding by presence the fact that there is an object over there that cancels the interaction with the acoustic field. In [23], it has been stated the procedure to get the spectral coefficients of the acoustic wave for a given composition of holes in the plate. However, these equations can not be inverted.

Covering a hole changes not only the spectral coefficients (R_α) and the lengths of the holes (L_α), but also intermediate variables that can not be easily measured. Some examples of these intermediate variables are the interaction term ($\chi_{\alpha\beta}$), the hole's filling fraction (f_α) or the frequency coefficients for a single hole (B_α). They are terms that vary depending on the covered holes in the structure. If all this information was available for each different image covering the plate, the inverse problem would not be difficult to solve. Despite this, recovering this information would mean a much more complex measurement system, and more measuring time per image, reducing recovery speed.

Machine learning algorithms, and more specifically Artificial Neural Networks, are tools that allow to find the solution of the problem without concerning about these variations in intermediate variables. By giving the network examples of spectral coefficients and the solution of covered holes, the network is supposed to get the nature of the problem and to reproduce it. Also in [23], a way of solving the problem the inverse problem has been developed: given some specific frequency coefficients, the distribution and length of the holes can be retrieved. Nevertheless, this method is still different of what is expected for the structure in this work. This inverse method works fine for plates with simple scattering structures (up to 4 or 5 different holes, periodic structures and cancelling modes (some $R_\alpha = 0$)).

C.3 Network's architecture

There are two main approaches inside supervised learning to solve problems by means of an artificial neural network: regression and classification. The former tries to find out which mathematical function follows the model, that is to say, it estimates a value using the input information. The latter focuses on literally finding the classes to which the data belongs. At first glance, one could think that the problem described in the previous chapter is a regression problem; physical equations are just mathematical functions that describe the behavior of a real situation.

An acoustic pulse is emitted by a loudspeaker; the sound wave travels the space and interacts with the modulated structure and the object, creating a new wave that is received on the microphone. However, this situation does not try to map any physical property of the object; just the presence or not of an object in a given position. Thus, the spatial distribution of the object on the structured is mathematically modelled by a Boolean matrix.

This consideration changes the way the problem is seen. It can be considered that for a given acoustic spectrum, the result is just a classification of either "pixels" are present or not. Therefore, the approach used in this work is a classification network.

Common classification networks consider that classes are exclusive (that is to say, only one output class is activated). They use activation functions that enhance this property, for example *softmax* activation function. In the case of modelling the problem as a simple classification problem, each combination of present pixels must be considered a different class. In other words, the presence of pixels 3 and 5 is seen by the neural network differently from the presence of pixel 3 and the presence of pixel 5. Consequently, for a N vs N pixels image, the number of classes is

$$N_{classes} = 2^{N^2} - 1. \tag{C.1}$$

The -1 term appears considering that the case of all pixels present (all holes uncovered) does not make any sense (it would mean that there is no object on the plate). For a 5 vs 5 image, there are 33 million classes, and for a 10 vs 10 image, this number grows up to $1.26{\cdot}10^{30}$. This amount of classes is unaffordable for a classic classification problem. Then, the idea is to reduce the number of classes to N^2; that is to say, to the number of pixels.

Consequently, this will mean that for a given spectrum, more than one class is active at the same time. This situation is known in neural networks vocabulary as a **multi-label learning** problem. There are two main approaches in this situation: an intuitive approach is to decompose it into multiple independent binary classification problems (one per category). However, this kind of method does not consider the correlations between the different labels of each instance and the expressive power of such a system can be weak. These methods transform a multi-label learning problem into a multiple single-label assignment as a binary classification problem. For example, an algorithm of this kind is the MT-DNN architecture (Multi-Task Deep Neural Network) [12] The second approach considers these relationships between labels. Several signal processing techniques have been designed, not only by means of neural networks, but also using other techniques, such as text categorization algorithms, decision trees and kernel methods ([17], [27]). The method chosen here is a neural network algorithm known as **BP-MLL** (Backpropagation for Multi-Label Learning). [28]

It is derived form the **backpropagation** algorithm (which can be first stated in [26], P.Werbos PhD thesis). Backpropagation algorithm is the most common training algorithm for fully-forward connected neural networks. Is is based on a gradient descent technique that minimizes a function (usually MSE). [7]

Backpropagation algorithm

At the heart of backpropagation is an expression for the partial derivative $\partial C/\partial w$ of the cost function C with respect to any weight w (or bias b) in the network [18]. This expression shows how quickly the cost changes when weights and biases are changed.

But before going deep insight backpropagation expressions, the algorithm used to compute the output must be shown (forward propagation). Weight notation is the following: w^l_{jk} will be used to denote the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.

As for network's biases and activations, the notation is: b^l_j is used for the bias of the j^{th} neuron in the l^{th} layer, whereas c^l_j is used for the activation of the j^{th} neuron in the l^{th} layer. The activation c^l_j of the j^{th} neuron in the l^{th} layer is related to the activations in the $(l-1)^{th}$ layer by

$$c^l_j = f(\sum_k w^l_{jk} c^{l-1}_k + b^l_j), \tag{C.2}$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer. Then, if matrix notation is introduced, there is a weight matrix w^l for each layer, l. Similarly, there is a bias vector, b^l and an activation vector c^l. Finally, f function can be vectorized. The notation $f(v)$ denotes an elementwise application of a function. Then, the equation above can be rewritten as

$$c^l = f(w^l c^{l-1} + b^l). \tag{C.3}$$

This expression shows how activation vectors are obtained: the weight matrix is applied to the activations, then bias vector is added, and finally the f function is applied. An intermediate variable is then introduced: $z^l = w^l c^{l-1} + b^l$. It is called the weighted input to the neurons in layer l. Thus, the previous equation can be written as $c^l = f(z^l)$.

The aim of backpropagation is to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network. An example of cost function is the quadratic cost function

$$C = \frac{1}{2n} \sum_x ||y(x) - c^L(x)||^2, \tag{C.4}$$

being n the total number of training examples, y is the desired output, L denotes the number of layers in the network, and c^L is the vector of activations output from the network when x is the input.

Two main assumptions concerning the cost function must be done. The first assumption is that the cost function can be written as an average over cost functions for individual training examples. The reason is that backpropagation computes partial derivative for single training example and then average over training examples. The second assumption is that the cost function can be written as a function of the outputs from the neural network, that is to say, $C = C(c^L)$. The quadratic cost expressed above accomplishes both requirements.

Now, the main backpropagation equations can be deduced. Backpropagation is about understanding how changing the weights and biases in a network changes the cost function. This means computing $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$. To compute those, another intermediate quantity is introduced, δ^l_j, the error in the j^{th} neuron in the l^{th} layer. The error is defined as

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}. \tag{C.5}$$

The first backpropagation equation is an equation for the error in the output layer (δ^L), given by

$$\delta_j^L = \frac{\partial C}{\partial c_j^L} f^{'}(z_j^L). \tag{C.6}$$

In this equation, the partial derivative term measures how fast the cost is changing as a function of the j^{th} output activation. The second term (f), measures how fast the activation function f is changing at z_j^L. Equation (C.6) is not in matrix-based form. It can be written in a matrix-based form as

$$\delta^L = \nabla_c C \odot f^{'}(z^L), \tag{C.7}$$

where $\odot$ stands for the Hadamard product.The second equation is an equation for the error δ^l in terms of the error in the next layer (δ^{l+1}). The expression is

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f^{'}(z^l). \tag{C.8}$$

Here, $(w^{l+1})^T$ is the transpose of the weight matrix for the $(l+1)^{th}$ layer. Applying the transpose weight matrix is intuitively the same as moving the error backward through the network. Taking the Hadamard product ($\odot$) with the activation function moves the error through the activation function in layer l. Combining (C.8) and (C.6), the error can be computed for any layer in the network.

Third equation is an equation for the rate of change of the cost with respect to any bias in the network, given by

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{C.9}$$

Finally, fourth equation is an equation for the rate of change of the cost with respect to any weight in the network, stated as

$$\frac{\partial C}{\partial w_{jk}^l} = c_k^{l-1} \delta_j^l. \tag{C.10}$$

This expression shows how to compute the partial derivatives $\partial C/\partial w_{jk}^l$ in terms of the quantities δ^l and c^{l-1}. A consequence of this equation is that weights output from low-activation neurons (c_k^{l-1} is small) learn slowly.

From (C.6) equation, it can be outlined that a weight in the final layer will learn slowly if the output neuron is either low activation or high activation. The four fundamental equations work for any activation function, not just the standard sigmoid function. All four fundamental equations are consequences of the chain rule from multivariable calculus.

Equation (C.6) can be obtained via

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial c_k^L}\frac{\partial c_k^L}{\partial z_j^L}. \tag{C.11}$$

The sum is over all neurons k in the output layer. As the output activation c_k^L depends only on the weighted input z_j^L when $k = j$, the term $\partial c_k^L/\partial z_j^L$ vanishes when $k \neq j$, leading

$$\delta_j^L = \frac{\partial C}{\partial c_j^L}\frac{\partial c_j^L}{\partial z_j^L}. \tag{C.12}$$

Recalling that $c_j^L = f(z_j^L)$,

$$\delta_j^L = \frac{\partial C}{\partial c_j^L} f'(z_j^L). \tag{C.13}$$

Which is exactly equation (C.6). Proving equation (C.8) needs to rewrite $\delta_j^l = \partial C/\partial z_j^l$ in terms of $\delta_k^{l+1} = \partial C/\partial z_k^{l+1}$,

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\delta_k^{l+1}, \tag{C.14}$$

given that $z_k^{l+1} = \sum_j w_{kj}^{l+1} f(z_j^l) + b_k^{l+1}$, the first term in the above equation is

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l). \tag{C.15}$$

Finally, substituting this development, equation (C.8) is obtained.

Equations (C.9) and (C.10) are also obtained by means of the chain rule. These four equations are combined with an optimization technique so as to modify the weights and biases reducing the cost function.

Gradient Descent Optimization

On top of these backpropagation algorithm there is an optimization technique, whose goal is to find weights and biases so that the output from the network approximates $y(x)$ for all training inputs x. This is done by minimizing the cost function defined before. A given cost function $C(v)$ is going to be minimized; a two variable example is taken for illustrating the algorithm, such as

$$C(v) = C(v_1, v_2). \tag{C.16}$$

Defining the gradient as the vector of partial derivatives, it is obtained

$$\nabla C = (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2})^T. \tag{C.17}$$

A change in the variable vector Δv implies a change in the cost function

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{C.18}$$

By setting $\Delta v = -\eta \nabla C$, the above equation becomes $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta ||\nabla C||^2$. Because $||\nabla C||^2 \geq 0$, this guarantees that $\Delta C \leq 0$, in other words, C will always decrease. Even if a two variable example has been used, gradient descent works fine for multivariate functions.

Algorithm's summary

These four fundamental backpropagation equations plus the gradient descent technique can be written in the form on a sequenced algorithm:

1. **Input x**: Set the corresponding activation a^l for the input layer.
2. Random low value **weight initialization**.
3. Random choice of **input feature**.
4. **Feedforward**: For each layer compute $z^l = w^l c^{l-1} + b^l$ and $c^l = f(z^l)$.
5. **Output error** δ^L: compute the vector $\delta^L = \nabla_c C \odot f^{'}(z^L)$.
6. **Backpropagate the error**: For each layer compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f^{'}(z^l)$.
7. **Output**: The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = c^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.
8. **Modify** layer weights using $\Delta v = -\eta \nabla C$.
9. **Get back to** second item and repeat.

Algorithm modification

In backpropagation algorithm, learning rate (η) is a critical feature for network's development. It determines the weight modification magnitude. If it is too small, convergence speed is too slow and probability of getting stacked in a local minimum increases. However, if the learning rate is too big, it can lead to instability (oscillations) inside loss function. There are several techniques to decrease this oscillations, such as changing of optimizer (not using simple gradient descent) or adding a momentum term in the weight modification step.

The chosen optimizer here is *Adagrad*, which adapts the learning rate to the parameters, performing smaller updates for parameters associated with frequently occurring features, and larger updates for parameters associated with infrequent features. It is well-suited for dealing with sparse data [1]. *Adagrad* uses a different learning rate for every parameter at every time step. [20] A learning rate of 0.002 has been chosen, as it has shown good performance experimentally. The upgrading equation for the *Adagrad* optimizer is

$$\Delta v_{t+1,i} = v_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla C, \tag{C.19}$$

[1] A variable with sparse data is one in which a relatively high percentage of the variable's cells are zero. Moreover, a variable is also called sparse when it can be expressed in some base where its representation is sparse.

where $G_t \in \mathbb{R}^{dxd}$ is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients. The main weakness of this technique is the accumulated term in the denominator of the upgrading parameter formula: learning rate eventually becomes infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

The activation function used between each layer of the neural net is a **hyperbolic tangent** function. This function is not the most commonly used for Deep Learning architectures (ReLU is the most popular, as it avoids "vanishing gradients"), but its properties allow to use the backpropagation algorithm.

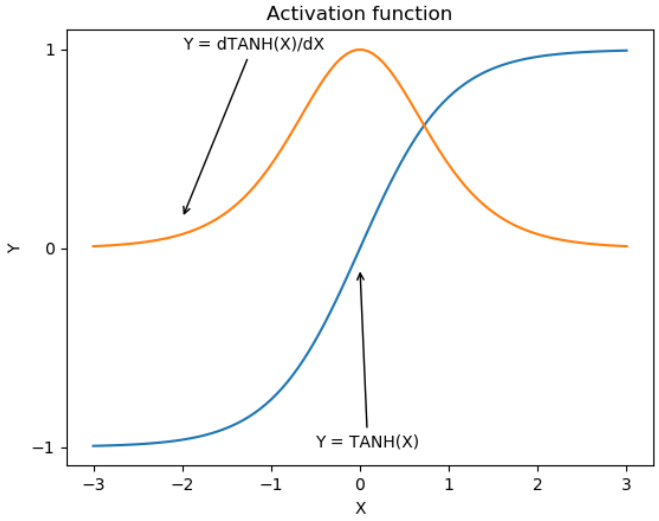


Figure C.1: *Activation function.*

Figure C.1 shows the representation of the activation function and its derivative. The latter is non zero except for big input values. This property is relevant for the backpropagation algorithm; most of the optimizers are based on the derivative of the error function (fifth item in backpropagation algorithm description). Big error inputs affect less than inputs with small errors, correcting weights in a better way.

Changes applied by Zhang [28] to implement multilabel learning are presented now. The common error functions used in most networks have been replaced by a new function which tries to capture the characteristics of multi-label learning (labels belonging to an instance should be ranked higher than those not belonging to). The error function considered here is defined by

$$E = \sum_{i=1}^{m} E_i = \sum_{i=1}^{m} \frac{1}{|Y_i| \cdot |\overline{Y}_i|} \cdot \sum_{(k,l) \in Y_i x \overline{Y}_i} exp(-(c_k^i + c_l^i)), \tag{C.20}$$

where Y_i is the label set associated to the training instance x_i, and c_j^i is the j-th output unit of the i-th training example, $\overline{Y}_i$ is the complementary set of Y_i, $|\cdot|$ measures the cardinality of a set and $f(x)$ is the activation function, which is set to be the hyperbolic tangent function,

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{C.21}$$

Finally, the term $c_k^i - c_l^i$ in equation (C.20) measures the difference between the outputs of the network on one label belonging to $x_i (k \in Y_i)$ and one label not belonging to it ($l \in \overline{Y}_i$). As this difference increases, the performance of the loss function is better. In addition, a negation of this difference (the output from a non-belonging label is higher than the output from a belonging label) is severely penalized by the exponential term.

The summation term in equation (C.20) takes into account the accumulated difference between outputs of any pair of labels (one belonging to x_i and the other one not belonging to it), and finally it is normalized by the total number of possible pairs (equation (C.20) denominator). As a consequence, correlations between different labels of x_i should get larger network outputs than those in $\overline{Y}_i$ (set of non-belonging labels of x_i). Minimization of this equation leads to output larger values for labels belonging to the training instance and smaller values for those not belonging to it. The error function is related to *ranking loss*, which evaluates the average fraction of label pairs that are reversely ordered for the instances (how many labels not belonging to x_i have higher outputs than labels belonging to x_i). Mathematically, it can be expressed as

$$rloss_S(f) = \frac{1}{p} \sum_{i=1}^{p} \frac{|D_i|}{|Y_i| \cdot |\overline{Y}_i|}, \tag{C.22}$$

being $D_i = \{(y_1, y_2) | f(x_i, y_1) \leq f(x_i, y_2), (y_1, y_2) \in Y_i x \overline{Y}_i\}$.

This new loss function is applied as the error function in the backpropagation algorithm. Here is the main difference respect to classical backpropagation equation development. Instead of using classic error functions, such as MSE or MAE, equation (C.20) is going to be used to derivate weight updating equations. After a mathematical development that can be consulted in [28] (equations from [11] to [16]), final parameter upgrading expression is

$$\Delta w^l = -\eta \frac{\partial E}{\partial w^l} = -\eta \frac{\partial E}{\partial z^l} \frac{\partial z^l}{\partial w^l} = \eta \delta^l \left[\frac{\partial (c^{l-1} w^l)}{\partial w^l} \right] = \eta \delta^l c^{l-1}. \tag{C.23}$$

The BP-MLL architecture also has a particularity in its final step: there is a linear **least squares** method applied in order to retrieve the thresholds that will be compared to the output of the net. Thus, this final step makes the final output a Boolean array. The explanation of the LMS method can be found at [28].

This LMS algorithm is used only in prediction for an unseen instance. The instance enters the network, and some outputs c_j are used for label ranking. The associated label set is determined then by means of these thresholds (there is a different threshold value for each label).

There is a difference between the solution stated by Zhang in his article and the applied solution in this work. This difference is subtle and the algorithm should work as well as Zhang's option. Labels in this project are classified as 1 whenever the hole is present and 0 when the hole is covered, instead of ± 1 as in Zhang's paper. There is no reason to obtain different results, as the condition of ranked labels is the same considering 0 values or -1. Another difference between Zhang's article and the algorithm applied here appears when training: Zhang's equation's have been deduced for **online training**, while the algorithm used here is trained using **mini-batch training**. As you can see, differences are little and they do not compromise the stability of the algorithm.

To sum up, the main characteristics of the net's architecture are:

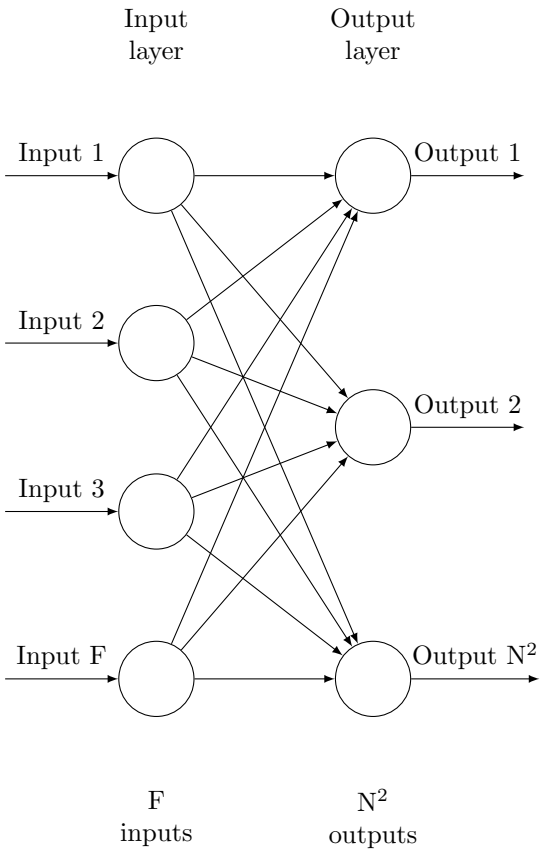
- Fully-forward connected layers.

- Backprogapation algorithm (in this case, Adagrad optimizer will be used, but it is not the only solution).
- Hyperbolic tangent activation function.
- Loss function allowing appropriate label ranking.
- Least Mean Squares algorithm to compute the final thresholds.

C.4 Hyper parameter discussion: choices

Before trying to reconstruct high resolution acoustic images, a smaller example will be presented, giving a proof that the situation has a solution. The example is based on a 5 vs 5 holed plate, with all its holes situated equispaced on two dimensions. The loudspeaker is supposed to be on one side, orientated towards the plate with an angle of 45º with the plate plane. Moreover, the microphone is supposed to be in the center of the plane, and at the same height ($z = 0$).

The neural net used in this case is that described before: a BP-MLL neural net. In the following figure there is an scheme of the architecture of the net:



The input layer obviously has the same size as the spectrum used as input. The output layer is the following one, which converts from the input vector to the multi-label classification with the same size as holes exist in the plate. In this first situation, F is equal to 997 frequencies and N^2 is equal to 24 holes/pixels.

A 60000 cases data set has been used, distributing them in three different parts: train, test and validation. Empirical knowledge recommends dividing these three parts in proportions such as Train: 70%, Test: 20%, Validation: 10%. Each one of them is used in different parts of the training procedure, as it can be seen in the following picture:

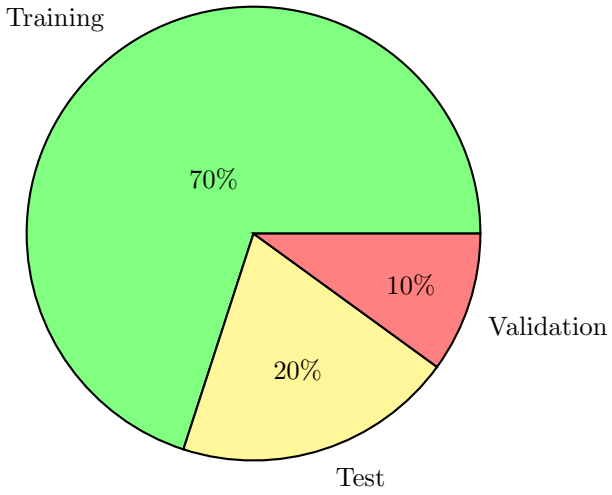


Figure C.2: *Data set distribution*

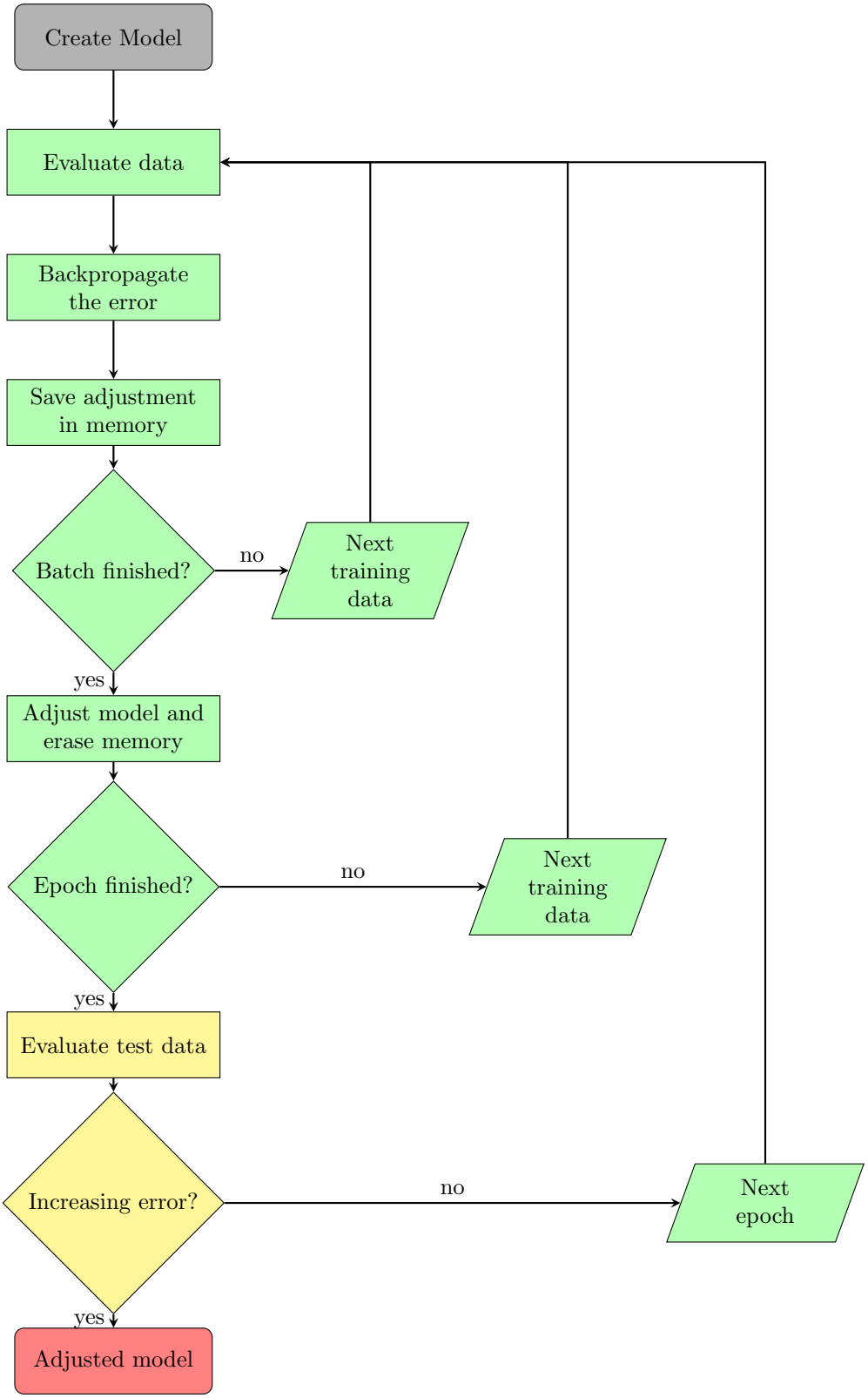


Figure C.3: *Training flowchart*

Training procedure has been established as follows: training data set is split into batches of size 1000 (this kind of training is known as **mini-batch training**). Neural weights are upgraded after all training cases in a batch have been propagated through the layers; each training sample has its own error value, and the mean error on the whole batch is used as the upgrading parameter. At each epoch, all training batches go through the neural net and upgrade its weights. Then, another epoch begins. It must be said that between epochs the training data set is rearranged and splitted again, in order to avoid certain behaviors due to data's entry order. Training does not finish when a given number of epochs has been accomplished; after each epoch, an error is measured using test data set (the network has never seen this data before). This error corresponds to the BP-MLL error function described before. Once the error function decreases slower than 1% in 5 consecutive epochs, the training procedure is stopped, and the network's model is saved with the weight values corresponding to the minimum error epoch. In **Figure C.2** it can be seen how the global data set is splitted to train the network, and in **Figure C.3**, each box is coloured with the colour code established in the pie chart, showing which data set is used at each step.

Once the net is trained, all three data sets are introduced in the network, and some results are predicted. As real images are known, they can be compared to predictions. The comparison is made using what is called as accuracy error. The idea can be seen in Figure C.4:

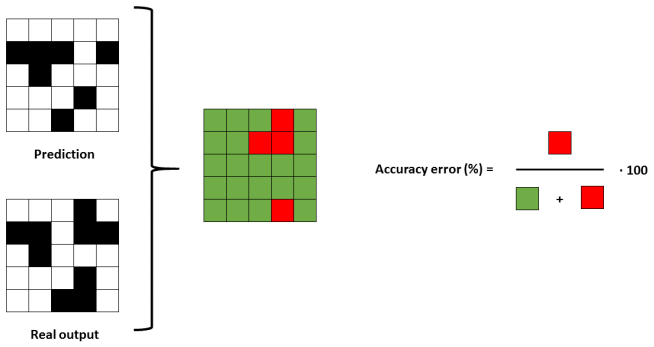


Figure C.4: *Accuracy error*

Accuracy is explained in [16] as a function of wright and wrong cases. Defining TP as true positive, or all cases were prediction is positive and real answer is also positive, TN as true negative, or all cases were prediction is negative and real answer is also negative, FP as false positive, or all cases were prediction is positive and real answer is negative, and FN as false negative, or all cases were prediction is negative and real answer is positive, accuracy can be obtained as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{C.24}$$

Therefore, accuracy error as stated before is just $1 - Accuracy$.

C.5 Results

First of all, it has been proofed that as BP-MLL loss function decreases, the image accuracy error decreases too. In order to show it, the following simulation has been done: the network has been trained a fixed number of epochs (2000) and, for several epochs, the network has been saved as the trained one and accuracy error has been computed. As this procedure is time consuming, accuracy error has been computed every 50 epochs, thus having 40 accuracy errors in the end. Results are shown here:

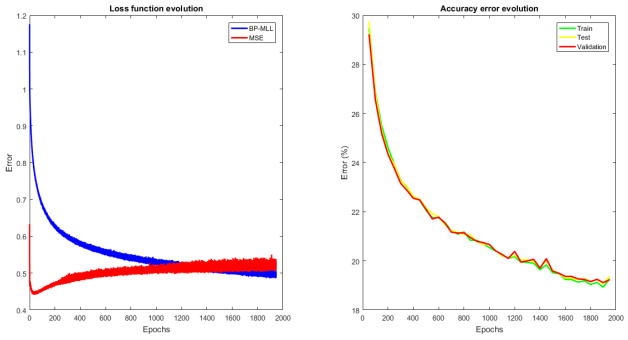


Figure C.5: *Accuracy error evolution during training.*

In **Figure C.5**, two different loss functions have been compared. Training has been applied using BP-MLL loss function, but MSE has also been obtained for each batch and epoch. As it can be inferred by looking both graphs, while BP-MLL loss function seems to improve image reconstruction accuracy, MSE shows a different behavior; first it decreases dramatically, and then starts increasing, getting stacked at some value. Consequently, BP-MLL loss function has been found to be the function which best models the problem. These results are obtained using 250000 cases data set of a 100 holes problem, a bigger problem than the actual one. Nevertheless, it has been assumed that the system's behavior is the same for both situations.

Results concerning the 24 holes situation are shown in the following table:

Data set	Accuracy error (%)	Perfect rec.	AENPC (%)
Train	0.23	36122/38180	4.27
Test	0.23	12054/12726	4.26
Validation	0.23	8497/8983	4.33

Table C.1: *5x5 pixel image reconstruction results. AENPC stands for Accuracy Error for Non Perfect Cases, which is the Accuracy Error without taking into account those cases that have been completely well reconstructed.*

From **Table C.1** and **Figure C.5**, it can be deduced that training has taken place properly, as the measured error function decreases during all the training procedure and the accuracy error of each data set is mostly the same. One of the main problems of training a neural network is overtraining or overfitting: it means that the model performs well on training data, but it does not generalize well for unseen cases. [10] This situation would be present in the error evolution if it had stopped decreasing at a point and then it had begun to increase slightly, or if there had been a big difference between data sets accuracy errors.

This results shown in **Table C.1** have been computed using cross validation technique. In this case,

the error value in the table is the mean value for four different training. Cross validation is one of the techniques used to test the effectiveness of a machine learning model. It is also a re-sampling procedure used to evaluate a model if the available data is limited. The cross-validation approach used here is based on the K-fold cross validation. It ensures that every observation from the original data set appears at least once in the training set. Basically, what has been done is to train several times the net, splitting the data sets (training, test and validation) rotating the observations. One given observation will have been in the training set, in the test set and in the validation set at the end of all training. The goal of this technique is to test the model's ability to predict new data not used before, preventing problems such as overfitting or selection bias [6].

C.6 Conclusion

Simulations in the precedent section have shown that image reconstruction can be done by means of the proposed artificial neural network. An accuracy error of 0.23% in a 24 pixel image means that in average, there is no pixel of the image wrong reconstructed. Cases where the image is not well reconstructed show an error of 4.3%, which means only one wrong pixel. Considering this situation, it can be assumed that the shape of the object will not be severely distorted, and thus, the problem can be considered to be solved.

However, it exists the possibility of decreasing this accuracy error: by fine-tuning the hyperparameters of the artificial neural network, or generating a bigger data set, as the data set used represents less than 1% of the possible cases. Naturally, there is a drawback in increasing the data set size, and it will be one of the main constraints when the whole system will be embedded in the microcontroller; memory space.

C.7 Beyond the problem: increasing matrix dimension

Treating the reconstruction of a 10x10 pixel image is different from solving the 5x5 image. In the precedent section, a 60000 cases data set has been created, out of $2^{25} - 1 = 33554431$ cases, thus representing a 0.20% of the possibilities. In this second situation, a 230000 cases data set has been created, out of $2^{100} - 1 = 1.2677 \cdot 10^{30}$ cases. This time, the data set represents a $1.8143 \cdot 10^{-26}$% of the possibilities. Here, the limitation appears from the computation time; generating this data set has cost more than a week, even working with processor parallelization. Getting up to 1 million cases would spent a month, and solving the problem for all possible cases takes longer than a year. Therefore, just with a 10x10 image, it can be seen that the machine learning approach is necessary to solve the problem.

Using the same model as for the 5x5 pixel image, the accuracy error obtained for the three data sets is around 20%, which is four times the error obtained for the 5x5 pixel situation. This result means that, in average, 1 over 5 pixels is wrong, and then, considering the 100 pixels image, there are 20 pixels wrong. This amount of wrong pixels can easily blur the shape of the object, so another method or technique must be found.

Data set	Accuracy error (%)	Perfect reconstruction	AENPC (%)
Train	19.28	0/69964	19.23
Test	19.37	0/19981	19.37
Validation	19.23	0/9990	19.23

Table C.2: *10x10 pixel number image reconstruction results*

Up to the moment, masks used were random patterns: for each hole, a random number between 0 and 1 was generated and compared to 0.5, generating a *true* or $false$ with 50% probability. Then, masks where random with 50% of hole's covered in average. Generating this kind of masks has proofed been useful, as they show great variability. However, this situation is not what it is found in reality. Objects usually have a well-defined shape, and it is one of the main important features. What has been proposed next is to change the data set used and to train with masks that simulates digits (instead of random masks as before), defining a new dictionary. This way, masks nature is changed, reducing the space of solutions and making the problem easier for the network. Some examples of these masks can be seen in **Figure C.6**.

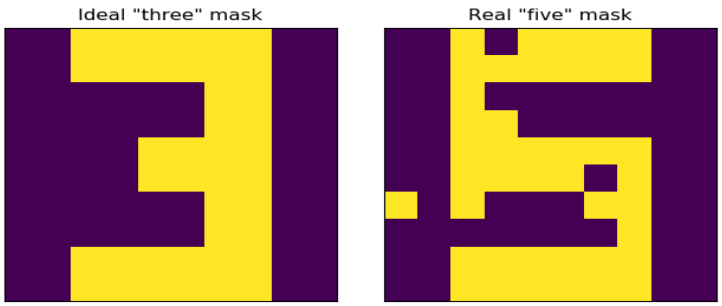


Figure C.6: *Masks examples. On the left, an ideal mask representing a "three". On the right, a real mask, where the "real" five has been blurred with a 5% uniform noise.*

Network's architecture has not been changed. Training and testing with these masks and spectra produces the following results:

Data set	Accuracy error (%)	Perfect reconstruction	AENPC (%)
Train	0.50	61030/69964	3.92
Test	0.49	17451/19981	3.88
Validation	0.48	8750/9990	3.84

Table C.3: *10x10 pixel number image reconstruction results*

From **Table C.3** it can be outlined that the neural network works fine. It is able to reconstruct more than 85% of the numbers without any wrong pixel. Problem's complexity has been reduced, and then the network is able to solve the problem giving good accuracy.

Possible solutions have been increased by adding displaced digits. The idea here is to validate whether the network is able to recognize the shape even if the position is not the same or not. These displaced masks have not been added to the training set; they are added only to the test set. Thus, the network is still trained using centered digits, and it is asked to solve displaced digits.

These displaced masks have also been blurred using the 5% uniform noise. Testing the trained model with these masks has shown that the net is not capable of generalize displacements on the masks. The results were not good, as it can be seen in **Figure C.7**.

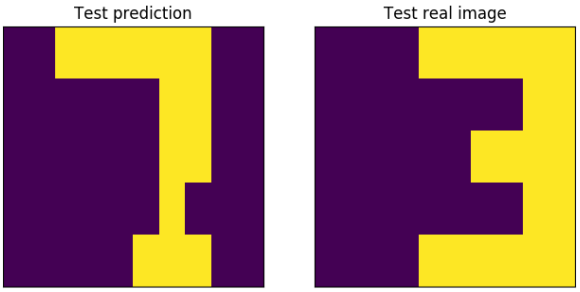


Figure C.7: *Displaced mask: network's reconstruction (left) and real image (right).*

Figure C.7 clearly shows that the neural network can not identify the displaced pattern, as all the training patterns were centered. A plausible hypothesis is that the network is trying to find the centered pattern with more active pixels in common with the displaced pattern. In order to get to this conclusion, it must be understood what does this displacement mean to the physical properties of the acoustic wave.

A certain pattern or mask covers certain holes and lets others uncovered. Each hole has its own resonance frequency. Moreover, each hole is surrounded by other holes with other resonance frequencies. It means that even if two different holes have close resonance frequencies, the system will not receive the same spectrum if either one is active or the other, as the surroundings are different, and the interference between sound coming from two holes will never be the same in two different spatial points. Therefore, whenever a mask or pattern is moved, those holes which where covered before are now uncovered, and the other way round.

The resonance frequencies acting in the centered mask are completely different from the ones that act for the displaced mask. Thus, both spectra are not forcefully related. In the figure below (**Figure C.8**), the explanation of this phenomena is shown:

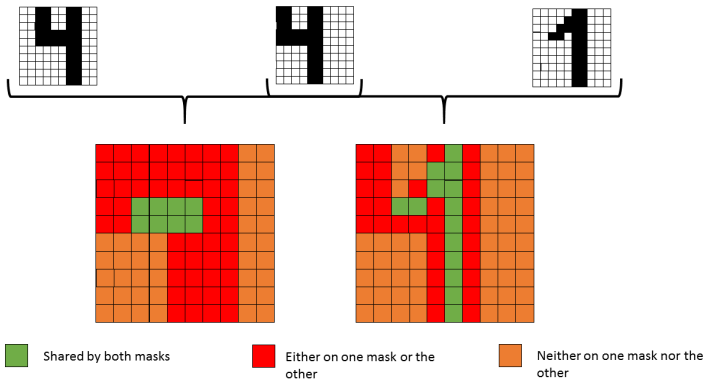


Figure C.8: *Resonators shared between different masks.*

Both *"four"* masks share little pixels. In addition, the pixels that appear neither on one nor the other are few. Covered hole's are as important as non covered hole's, that is to say, the relevant part of the figures above is the sum of the orange and the green part. Then, the *"displaced four"* mask and the *"one"* mask share more pixels than the other couple, and also the orange area is bigger. It can be said that the spectrum from the *"displaced"* mask will be more alike to the *"one"* mask rather to the *"centered four"* mask.

It has been shown the need to include displaced digits in the training data set so as to recover this kind of patterns. Then, after including them into the train, test and validation data set, the results are as follows:

Data set	**Accuracy error (%)**	**Perfect reconstruction**	**AENPC (%)**
Train	1.57	38489/52210	5.97
Test	1.37	15323/20583	5.35
Validation	1.34	7718/10293	5.34

Table C.4: *Results including displaced patterns in train data set*

There is a trade-off between including this kind of masks and getting better accuracy results: including them means an increase in the accuracy error. However, the network is now able to solve both kinds of pattern (centered and displaced). Notice that the number of train cases has decreased (as some new cases has been added to the data set, it should have increased). This choice has been done in order to balance the data set; if the percentage of displaced masks is largely smaller than the centered ones, the network will certainly have problems to solve the former ones.

Appendix D

Physics model

An acoustic image is going to be reconstructed using a loudspeaker, a microphone and a structured plate. Given several spectral coefficients, covered holes' positions in the structure are retrieved. Therefore, a robust physical model is needed.

In this Appendix, the physical model is going to be stated, as well as the background theory that supports it. The equations will solve the direct problem (that is to say, getting the spectral coefficients given a certain geometrical hole distribution), which is not the aim of this work. However, this direct solution is necessary to produce the simulations that will be in fact the observations used to train the artificial neural network by supervised learning. Furthermore, understanding the physics of the problem will provide deeper knowledge about the situation that is being explored and will help with future decision as choosing the network's architecture and hyperparameters.

First of all, multiple scattering theory (MST) will be introduced. Then, the real situation will be stated. Afterwards, physical equations to solve the problem will be presented and discussed, relating their properties with the real design properties of the plate. It will be seen that the model constraints real design parameters as well as real instruments constraint the simulations. Finally, MATLAB's code used to obtained learning cases will be discussed.

D.1 Introduction: Multiple scattering theory

Multiple scattering is not a new topic. It has been studied since the beginning of XXth century by many scientists as Lord Rayleigh, Heaviside, van de Hulst or Bohren. However, it is still a hard mathematical problem and several situations have no solution. Multiple scattering tries to define the interaction of fields with two or more obstacles [15]. For example, if there are several obstacles, the field scattered from one obstacle will induce further scattered fields from all the other obstacles, which will induce further scattered fields from all the other obstacles, and so on.

This phenomenon can be considered in various ways. Several studies have solved it by considering every obstacle completely independent from the others, that is to say, the scattered field by an obstacle is not affected by other obstacles. This hypothesis is valid only when the distance between obstacles is large enough. Nevertheless, there exists situation's where it can not be applied; for example, in atmospheric physics.

Multiple scattering theory is a versatile and fast simulation method, especially suitable for the analytical manipulation of the equations to obtain physical information that would be impossible with purely numerical methods. It is based on the expansion of the total field scattered by a cluster of N particles as a linear combination of the scattered field by each particle individually, and the solution of the problem consists in finding the coefficients of this linear combination. This is typically obtained after inversion of a $N \times N$ square matrix for point-like particles, but the size of the matrix can be larger if finite-size effects are taken into account.

The challenge in multiple scattering is to properly model the response of the scatterer (pillar, hole or sphere). The numerical simulation of phononic crystal plates is still limited to the Finite Element Method (FEM), which essentially provides of numerical experiments but does not allow to efficiently working on the underlying physics of the scattering of waves.

The fundamental hypothesis of this theory is that, given a cluster of point-like scatterers, the total field can be expressed as

$$\psi(\boldsymbol{r}) = \psi_0(\boldsymbol{r}) + \sum_{\beta} B_\beta G(\boldsymbol{r} - \boldsymbol{R}_\beta), \tag{D.1}$$

being ψ_0 the incident field, $\boldsymbol{r}$ is the point where the field is computed, $\boldsymbol{R}_\beta$ are the points of the scatterers, $G(\boldsymbol{r})$ the Green's function and the coefficients B_β are obtained from

$$\sum_{\beta} (t_\alpha^{-1}\delta_{\alpha\beta} - G(\boldsymbol{R}_\alpha - \boldsymbol{R}_\beta))B_\beta = \psi_0(\boldsymbol{R}_\beta), \tag{D.2}$$

which is a $N \times N$ system of equations. $\delta_{\alpha\beta}$ is Dirac's function, which is 1 only when $\alpha = \beta$. The quantity t_α is the scattering strength of each point-like scatterer and it is the only quantity that contains information about its physical properties. Using these equations, scatterers can be selected according their spatial distribution and their t_α coefficients. Then, B_α coefficients can be computed to finally obtain the field $\psi(\boldsymbol{r})$.

These two equations will be developed in the following section to give an insight on the physical phenomenon surrounding the structure.

D.2 Solution for acoustic waves

When an acoustic wave interacts with a periodic array of obstacles, it suffers from multiple scattering phenomenon. The holes in the N^2 hole cluster have length L_α, located at positions r_α. The incident wave is of unitary amplitude and its wave number is $\boldsymbol{k} = \boldsymbol{K} + q_0\hat{\boldsymbol{z}}$. A set of diffracted modes with reflection coefficients R_G will be excited, defining a pressure field

$$P = \sum_{G} (\delta_{G0} e^{iq_G z} + R_G e^{-iq_G z}) e^{i\boldsymbol{K}_G \cdot \boldsymbol{r}} \tag{D.3}$$

and a normal velocity field

$$v_n = \frac{iq_G}{k_b Z_b} \sum_{G} (\delta_{G0} e^{iq_G z} - R_G e^{-iq_G z}) e^{i\boldsymbol{K}_G \cdot \boldsymbol{r}}, \tag{D.4}$$

with $|\boldsymbol{K}+\boldsymbol{G}|^2+q_G^2 = w^2/c_b^2$ and with $\boldsymbol{G}$ being the set of all reciprocal lattice vectors [8]. These expressions are valid for the field outside the structure. The pressure field inside a hole, however, can be stated as

$$P = e^{i\boldsymbol{K}\cdot\boldsymbol{R}_\alpha} B_\alpha \frac{cosk_b(z-L_\alpha)}{sink_b L_\alpha}, \tag{D.5}$$

and the normal velocity field is given by

$$v_n = -\frac{e^{i\boldsymbol{K}\cdot\boldsymbol{R}_\alpha}}{Z_b} B_\alpha \frac{sink_b(z-L_\alpha)}{sink_b L_\alpha}. \tag{D.6}$$

Using a mode matching technique, both pairs of equations ((D.3), (D.4) and (D.5), (D.6)) are combined to give a solution to the acoustic field via the equations

$$\sum_G H_{\alpha G} e^{i\boldsymbol{G}\cdot\boldsymbol{R}_\alpha}(\delta_{G0} + \boldsymbol{R}_G) = B_\alpha cotk_b L_\alpha \tag{D.7}$$

$$\delta_{G0} - \boldsymbol{R}_G = -i\frac{k_b}{q_G}\sum_\beta f_\beta H_{\beta G} e^{-i\boldsymbol{G}\cdot\boldsymbol{R}_\beta} B_\beta. \tag{D.8}$$

Mode matching is also known as eigenmode expansion (**EME**) or bidirectional eigenmode propagation method (**BEP method**). It is a computational electrodynamics modelling technique, based on a eigenmode expansion, which is a linear frequency-domain method. It relies on the decomposition of the electromagnetic fields into a basis set of local eigenmodes that exists in the cross section of the device. Also, there are two boundary equations that help matching both fields. These two conditions can be seen in **Figure D.1**:

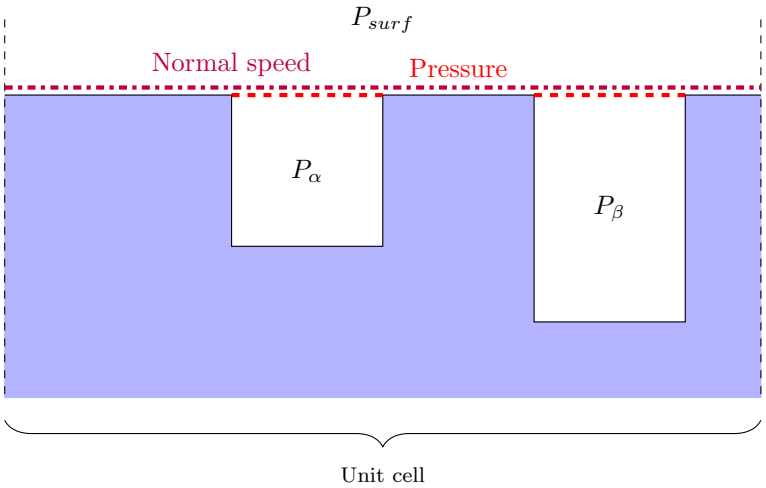


Figure D.1: *Boundary conditions*

the coupling factor is given by $H_{\alpha G} = \frac{1}{\Omega_h}\int\int_{\Omega_h} e^{i\boldsymbol{K}_G\cdot(\boldsymbol{r}-\boldsymbol{R}_\alpha)} d\Omega$ and the hole's filling fraction has been defined as $f_\alpha = \frac{\Omega_\alpha}{\Omega}$, with Ω and Ω_α being the areas of the unit cell and the hole α, respectively. The above system of equations allows to solve B_α coefficients from

$$\sum_{\beta} \left[\delta_{\alpha\beta} cot k_b L_\alpha - i\chi_{\alpha\beta}\right] B_\beta = 2H_{\alpha 0}, \tag{D.9}$$

where the interaction term $\chi_{\alpha\beta}$ is defined as

$$\chi_{\alpha\beta} = \sum_{G} \frac{k_b}{q_G} H_{\alpha G} H_{\beta G} f_\beta e^{-i\boldsymbol{G}\cdot\boldsymbol{R}_{\alpha\beta}}. \tag{D.10}$$

Then, knowing the geometry of the plate, B_α coefficients can be computed, and thus, reflection coefficients R_G can be computed, and the wave equation can be solved.

These equations allow to obtain reflected coefficients depending on the evaluated frequency. Then, the wave equation can be solved for any frequency. This idea will be used to obtain spectral information for a given geometric composition, supposing hole's covered and uncovered. Therefore, a geometric structure will be designed in order to get the information in a certain bandwidth, and spectra will be obtained using this structure and some binary masks (cancelling holes/resonators).

D.3 Plate's design

Once understood the theory surrounding a metastructure, next step is to design the geometry necessary to generate the desired problem. Requirements will be translated into mathematical equations first; then, the main parameters of the structure will be computed.

Analytic equations and computer simulations have always the advantage of not having real constraints. However, whenever real measurements are needed, some considerations need to be done. Real systems are characterized by parameters that are not always taken into account when the theoretical problem is solved. Therefore, in order to translate the equations into the real world, real limits must be considered.

The main issue with real measurements that will be released is related with electronic devices. Using Matlab, any wavelength can be simulated; nevertheless, real devices have their bandwidth limited. As it has been seen in the report, the available bandwidth goes from a few hertz to $22kHz$, that is to say, the audible bandwidth.

Multiple scattering theory also constrains the system in the following way. Each hole in the system has a main resonant frequency (or wavelength). Additionally, the scattering effect in one hole causes an effect in the nearby holes. To ensure that this interference is taking place, analysis wavelength must be longer than the distance between two consecutive holes, and shorter than several times the structure's length.

Hole's resonance frequency is in between $5kHz$ and $15kHz$, so as to be able to solve them with the microphone. Their wavelengths are: $\lambda = c/f = (2.27 - 6.80)cm$. Then, distance between holes should not be higher than $2.27cm$.

To sum up, requirements can be set as:

- Minimal frequency:
$$f_{min} > 100Hz \tag{D.11}$$
- Maximal frequency:
$$f_{max} < 22kHz \tag{D.12}$$

- Multiple scattering minimal length:

$$d < 2.27cm \tag{D.13}$$

It has been chosen to use the following geometric configuration: a certain squared plate of size $3L \times 3L$ with 25 holes distributed as shown in **Figure D.2**,

Central hole is not a resonator to be considered. It is supposed to be covered by the microphone, as the measurement point is there. Holes' depth has been chosen randomly, in a way that their resonance stays inside a given frequency interval $(5kHz - 15kHz)$ (the microphone used allows to measure up to this frequency without losing power). Each hole mainly resonates at a frequency related with its depth given

$$f_r = \frac{c}{4 \cdot L_i}. \tag{D.14}$$

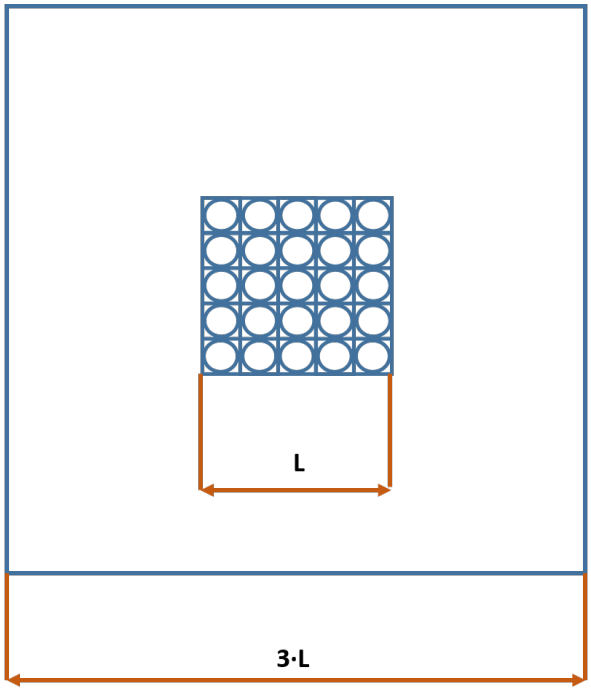


Figure D.2: *Plate's design*

Therefore, considering $c = 340m/s$ and frequency limits $(5kHz - 15kHz)$, holes' depth ought to be inside the interval

$$L_i \in \{0.57 - 1.70\}cm. \tag{D.15}$$

Distance between hole's has been chosen to be the mean between hole's lengths

$$d = \frac{L_{min} + L_{max}}{2} = 1.14cm. \tag{D.16}$$

Chosen d is smaller than the multiple scattering requirement.

There is a reason why a space of length L is left at each of the directions of the structure. Physical equations discussed before are valid for lattices, in other words, structures that are repeated several times in space. However, the structure in the project is finite. Instead of modelling the boundary effects that would appear considering a finite plate, the repeating cell in the structure has been enlarged with no hole space. Thus, there is ideally a $2 \cdot L$ space between two different hole structures. Multiple scattering theory depends on the distance between scatterers; it can be considered in this case that surrounding cells do not affect in terms of multiple scattering theory. Furthermore, as there is no sharp border next to the holes, there is no border effect to take into account. Despite this, the counterpart is that plate's surface is 9 times bigger than the real measurement surface. This condition constraints the size requirement stated in **Section 5** of the report.

As can be seen in **Figure D.2**, L is stated as $L = 15 \cdot d$, being d the net parameter. Whereas depths are completely independent for each hole, radium is the same for all of them. Radium can take values inside the interval $R_i \in (0 - d/2)$; a value of $R_i = 0.3 \cdot d$ has been chosen as it is considered experimentally to give good results. Small radii would diminish interaction between scatters, and big radii would make the whole structure a unique scatterer.

It is necessary for multiple scattering theory that the incident wavelength travels more than one scatterer in a single cycle. Considering the frequency of $f_{max} = 15kHz$, its wavelength is $\lambda_{min} = c/f_{max} = 2.27cm$. As $\lambda_{min}/d = 1.99 > 1$, a single cycle of this wave covers two holes. Using $f_{min} = 5kHz$, whose wavelength is $\lambda_{max} = c/f_{min} = 6.80cm$. Then, $\lambda_{max}/d = 5.97 > 1$, this wave covers more than five holes in a cycle.

Finally, dimensions have been defined. Holes' lengths are stated as in **Table D.1**:

6.17	15.59	13.70	10.41	16.06
10.88	14.65	11.82	8.52	6.64
9.94	12.29	0	8.05	9.47
14.17	7.11	12.76	9.00	13.23
17.00	16.53	11.35	5.70	7.58

Table D.1: *Holes' lengths (in mm).*

Hole's radium is $3.42mm$, except for the central hole, being $3.75mm$. Total plate length is $L = 171mm$, which is 15 times d. In the following section, a MATLAB code is presented translating the discussed physical equations in order to get a spectrum from a given geometrical distribution.

D.4 Exploring the code

Once the physical problem has been explained and understood, a MATLAB code simulating spectra coefficients in a given place for a given plate structure is shown:

```
% SCREEN_SQUARE.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
reda=1;
redb=1;
phi=pi/2;
Gmax=50;

rhob=1;
```

```
cb=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L0=reda; % Hole's distance
R0=0.3*reda; % Hole's radium
N=5; % Holes per row

Lscr=3*N*reda; % Plate's length

[ralpha,Ralpha,Lalpha]=square_cluster(N,R0,L0); % Plate's creation

Nh=length(Ralpha); % Number of holes

rhoalpha(1:Nh)=rhob; % Hole's density
calpha(1:Nh)=cb; % Hole's sound speed

sigmaL=0.3*reda;

Lalpha = round(linspace(0.0057,0.0170,Nh)./0.0114,4); % holes depth
Lalpha = Lalpha(randperm(Nh));

Ralpha=Ralpha./Lscr; % Normalization
Lalpha=Lalpha./Lscr; % Normalization
ralpha=ralpha./Lscr; % Normalization

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

uk=[1,0,1]/sqrt(2); % Propagation vector

z0=0; % z component of measurement point
Ns=1;

load('frequencies.mat'); % Frequency evaluation vector
nu = freq_acum/340*15*0.0114; % Frequency normalization
omega=2*pi*nu;

Nh=24;
Lalpha = Lalpha([1:12,14:25]); % Cancelling central hole
Ralpha = Ralpha([1:12,14:25]); % Cancelling central hole
ralpha = ralpha([1:12,14:25],:); % Cancelling central hole
rhoalpha = rhoalpha([1:12,14:25]); % Cancelling central hole
calpha = calpha([1:12,14:25]); % Cancelling central hole

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Void measurement (all holes present)

for nn=1:length(omega)
    [RG(nn,:),RGqG(nn,:),PhiR,Balpha,G,chiab,alphaG]=R_perforated_plate_backed(reda,redb,
    phi,Gmax,rhob,cb,Nh,Ralpha,Lalpha,ralpha,rhoalpha,calpha,omega(nn),uk);

    [r,psi0(nn,:,:)]=field_distribution(reda,redb,phi,omega(nn)./cb,z0,RG(nn,:),G,Ns); %
    Reflected field computation
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Ntests=60000; % Number of observations to compute

saveEvery = 1000;
fileX='X.mat';
fileY='Y.mat';
fileZ='Workspace.mat';
```

```
for mm=1:Ntests

    mask=randi([0 1],1,Nh); % Random chosen object to cover the plate
    Lp=Lalpha(mask>0);
    Rp=Ralpha(mask>0);
    rap=ralpha(mask>0,:);
    Np=length(Rp);

    parfor nn=1:length(omega)
        [RG(nn,:),RGqG(nn,:),PhiR,Balpha,G,chiab,alphaG]=R_perforated_plate_backed(reda,
    redb,phi,Gmax,rhob,cb,Np,Rp,Lp,rap,rhoalpha(1:Np),calpha(1:Np),omega(nn),uk);

        [r,psi(mm,nn)]=field_distribution(reda,redb,phi,omega(nn)./cb,z0,RG(nn,:),G,Ns);
    % Reflected field computation
    end

    OB(mm,:)=mask;
    psi(mm,:) = abs(psi(mm,:))./abs(transpose(psi0)); % Normalization respect to Void
    measurement

    if rem(mm,saveEvery)==0
        fprintf('saving at instance %d\n',mm)
        save('-v7',fileX,'OB');
        save('-v7',fileY,'psi');
        save('-v7',fileZ);
    end
end

save('-v7',fileX,'OB');
save('-v7',fileY,'psi');
save('-v7',fileZ);
```

Listing D.1: *screensquare.m*

This MATLAB code is the main code used to simulate and predict reflected frequency coefficients given a geometrical composition. As it can be seen, there are some subfunctions that are called in the code. The first one is *square cluster*, which is a function that creates the lattice and holes' locations in an squared distribution. After this, holes' depths are set. Both numbers, 0.0057 and 0.0170 correspond to normalized dimension of resonators at $5kHz$ and $15kHz$. Then, holes depths are rearranged permuting them randomly, and then they are normalized respect to the plate's length (that is to say, cell's size).

After stating all necessary constants and dimensions, the void measurement is done (void measurement is considered when all holes are present in the plate, meaning that there is no object covering them). Two subfunctions are called during this computation. The first one is *R perforated plate backed*, which can be seen in the following code excerpt.

```
function [RG,RGqG,PhiR,Balpha,KG,chiab,alphaG]=R_perforated_plate_backed(reda,redb,phi,
    Gmax,rhob,cb,Nh,Ralpha,Lalpha,ralpha,rhoalpha,calpha,omega,uk)

uvectors=direct_lattice(reda,redb,phi); % Direct lattice director vectors computation

[G,V,h0]=reciprocal_lattice(uvectors,Gmax); % Reciprocal lattice director vectors
    computation
G(:,3)=[];

Ac=V;

chiab = zeros(length(omega), Nh, Nh);
Mab = zeros(length(omega), Nh, Nh);

```

```
for nn=1:length(omega) % Para cada una de las frecuencias

    kb=omega(nn)/cb*uk;
    KG(:,1)=G(:,1)+kb(1); % Reflected wave vector = Reciprocal director vector + Incident wave vector
    KG(:,2)=G(:,2)+kb(2); % Reflected wave vector = Reciprocal director vector + Incident wave vector

    for alpha=1:Nh % For each hole
        Gralpha=KG(:,1)*ralpha(alpha,1)+KG(:,2)*ralpha(alpha,2);
        GRalpha=sqrt(KG(:,1).^2+KG(:,2).^2)*Ralpha(alpha); %
        alphaG(alpha,:)=2*exp(1i*Gralpha).*besselj(1,GRalpha+eps)./(eps+GRalpha); % HalphaG coefficient
    end

    Id=eye(Nh,Nh);
    IdG=eye(size(KG,1),size(KG,1));

    Ybeta=1i*omega(nn)./(rhoalpha.*calpha);
    fbeta=pi*Ralpha.^2/Ac;
    MG=sqrt(diag(KG*KG'))';
    qG=sqrt(omega(nn)^2/cb^2-MG.^2);
    YGb=1i*qG./rhob;

    for alpha=1:Nh
        for beta=1:Nh
            tchi=alphaG(alpha,:).*conj(alphaG(beta,:)).*fbeta(beta).*Ybeta(beta)./(eps+YGb); % Multiple scattering interaction between cavities
            chiab(nn,alpha,beta)=sum(tchi); % Multiple scattering interaction between cavities
            Mab(nn,alpha,beta)=Id(alpha,beta)*cot(omega(nn)*Lalpha(alpha)/calpha(alpha))-1i*chiab(nn,alpha,beta); % Article's equation (7)
        end
    end

    IM=squeeze(Mab(nn,:,:));

    Balpha(nn,:)=IM\(2*alphaG(:,h0)); % Balpha coefficient computation

    RG(nn,:)=IdG(:,h0)+1i*alphaG'*((Balpha(nn,:).').*fbeta(:).*Ybeta(:))./YGb.'; % Reflection coefficient computation. Article's equation (6)

    RGqG(nn,:)=real(qG).*abs(RG(nn,:)).^2./qG(h0); % Diffraction energy (Ig)

end

PhiR=sum(RGqG,2);
```

Matlab_code/R_perforated_plate_backed.m *excerpt*

This code is the most important one in all the simulation. This function computes the B_α coefficients and the reflection coefficients (R_α) using the equations discussed in **Section D.2**. This function also calls to two different functions: *direct lattice* and *reciprocal lattice*. These two functions compute the director vectors of both the direct lattice and the reciprocal lattice. The direct lattice is defined by the geometrical positions of the scatterers, defining the cell. The reciprocal lattice represents the Fourier transform of the direct lattice. It plays a fundamental role in most analytic studies of periodic structures, such as theory of diffraction. In the case of this project, reflected wave vectors are a combination of incident wave vectors and reciprocal lattice vectors. Line 38 in the code corresponds to **Equation D.9**. This equation helps finding B_α coefficients in line 44, and finally **Equation D.8** in line 46 computes the reflection coefficients.

The second subfunction called to compute the field is *field distribution*. This function computes the acoustic field in a given point for a given wavelength/frequency.

```
function [r,psi]=field_distribution(reda,redb,phi,kb,z,RG,G,npoints)

MG=sqrt(diag(G*G'))';

t=linspace(-0.5,0.5,npoints);
s=linspace(-0.5,0.5,npoints);
uvectors=direct_lattice(reda,redb,phi);

for LL=1:npoints
    for MM=1:npoints

        r(LL,MM,:)=t(LL)*uvectors(1).a+s(MM)*uvectors(2).a; % Vector going from coordinate's origin to the point where the field is computed
        x=r(LL,MM,1);
        y=r(LL,MM,2);
        qG=sqrt(kb^2-MG.^2);
        tpsi=RG(:).'.*exp(-1i*qG*z).*exp(1i*G(:,1)'*x).*exp(1i*G(:,2)'*y); % Reflected field at each hole
        psi(LL,MM)=sum(tpsi)./(reda*redb*sin(phi)); % Pressure field computed at the evaluation point

    end
end



 psi=psi+1;
```

Matlab_code/field_distribution.m *excerpt*

After having computed the acoustic field in the evaluation point for the void measurement, *screen square*'s code does a *loop* structure where a random mask is created and applied to the structured, vanishing some holes (vanishing is equal to covering in this context), as can be seen in lines 71 to 77 of the first code. Then, the acoustic field is computed for the geometrical distribution left, and this process is repeated until all the observations are computed.

Appendix E

Arduino code

E.1 Main code

```
# include "arduino_def.h"

//--------------------------------------------------------------------------------
// Global Variables
//--------------------------------------------------------------------------------

// Arrays
double Im[512];                     // Imaginary part of the signal
double Re[512];                     // Real part of the signal
int Indices[512];                   // Array with the rearranging coefficients
double c[24];                       // Output array of the algorithm
bool Output[24];                    // Boolean prediction (image)

// Processing variables
byte media=0;                       // Mean value used to normalize data

// ISR variables
volatile int n;                     // Samples counter
volatile byte vec[N];               // array of real values measured by the microphone
volatile uint16_t bit;              // Must be 16-bit to allow bit<<15 later in the code.
    Auxiliar variable
volatile uint16_t lfsr;             // 16-bit register to generate PRNG

// Flags
bool processing;                    // Flag that indicates if the ISR is active
int fi;                             // Flag that determines which step is
bool MODOVOID = true;               // Flag to change to Void Measurement mode

// Objects
arduinoFFT FFT = arduinoFFT();      // Create FFT object
File myFile;                        // Create File object
LiquidCrystal_I2C lcd(I2C_ADDR,2,1,0,4,5,6,7); // Create LCD object



//--------------------------------------------------------------------------------
// Main Routines
```

```
//-------------------------------------------------------------------------------

void setup() {

  Serial.begin(BAUDRATE);           // Initialize Serial Communication (once the program
    is finished, it won't be necessary)
  pinMode(loud,OUTPUT);             // Establish Loudspeaker's pin as output pin
  pinMode(lec,INPUT);               // Establish Microphone's pin as input pin
  pinMode(pin9,OUTPUT);             // Establish output tension
  digitalWrite(pin9,HIGH);
  pinMode(pinCS, OUTPUT);           // SPI communication

  // SD Card Initialization

  if (SD.begin())
  {
    Serial.println("SD card is ready to use.");
  } else
  {
    Serial.println("SD card initialization failed");
    return;
  }

  SD.remove("MEASURE.TXT");         // Erase previous measurement

  // LCD initialization and settings

  lcdConfig(lcd);

  // Settings

  startADC();
  setADCPrescaler(Prescaler);
  setVoltageReference(1);           // Set voltage reference by default --> set REFS0
  setADCres(Resolution);
  disDI();

  lfsr = start_state;               // Initialize the lfsr register

  measConfig();                     // Final ADC configuration

  n = 0;                            // Start the flag
  fi = 0;                           // Start the flag
  processing = false;               // Start the flag

  if (MODOVOID==true){
    SD.remove("VOID.TXT");
    SD.remove("MEAS.TXT");
  }
}


void loop() {

  if(n==0 && not(processing) && fi<4)
  {
    startMeas(fi);
  }
  if(n>=N && processing && fi<4)
  {
    finishMeas();
    fi++;
```

```
    n = 0;
100     media = mean(vec);
    for(int i=0;i<N;i++)
102     {
      Re[i] = (double)vec[i] - (double)media;
104     }
    memset((void *)Im, 0, 4*N);
106     fftStuff(Re, Im);
  }
108
  // Indexation and sorting
110
  if(fi==4 & processing==false)
112   {
    indSort();
114     fi=5;
  }
116
  // Normalization
118
  if (fi==5 & processing==false)
120   {
    normalization(MODOVOID, Re);
122   }

124   // Prediction

126   if (fi==6 & processing==false & not(MODOVOID))
  {
128     prediction(Re, c, Output);
  }
130
  // Saving results and printing
132
  if (fi==7 & processing==false & not(MODOVOID)){
134     saving(Output,lcd);

136     // Restart measurement

138     fi=0;
    n=0;
140   }
}
142


144
//-------------------------------------------------------------------------------
146 // Interruptions
//-------------------------------------------------------------------------------
148
ISR(TIMER2_COMPA_vect){
150   bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5)); // & 1u ;
  if(bitRead(lfsr,15)==1)            // Optimized digital write (PORTH,4)=PIN 7
152   {
    PORTH |= B00010000;
154   }
  else
156   {
    PORTH &= B00000000;
158   }
  if(n<N){                           // To avoid array overflow
160     vec[n] = ADCH;
    n++;
```

```
162   }

164   lfsr = (lfsr >> 1) | (bit << 15); // lfsr actualization
}
```

Arduino_code/arduino_def.ino *excerpt*

E.2 Header

```
//-------------------------------------------------------------------
// Prueba_microfono.h
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// Includes
//-------------------------------------------------------------------

# include <stdint.h>
# include <arduinoFFT.h>              // float-point FFT library

#include <SD.h>                       // Library to communicate with the microSD card
#include <SPI.h>

#include <LiquidCrystal_I2C.h>        // LCD Library
#include <Wire.h>
#include <LCD.h>

//-------------------------------------------------------------------
// Defines and Typedefs
//-------------------------------------------------------------------

# define CLR(x,y) (x&=(~(1<<y)))
# define SET(x,y) (x|=(1<<y))
# define PI 3.14159

// Defines for setting and clearing register bits
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#define BAUDRATE  9600  // Baud rate of UART in bps
#define I2C_ADDR  0x27  // i2c memory direction (obtained with I2C_scan)

//-------------------------------------------------------------------------------
// Global Constants
//-------------------------------------------------------------------------------

const uint16_t N=512;                   // Number of measurements
const int m=9;                          // Power of two that gives N
const int loud = 7;                     // Loudspeaker output pin
const char lec = "A0";                  // Microphone input pin
const int pin9 = 9;                     // Used as a voltage output
const uint8_t Prescaler = 8;            // ADC Clock Prescaler
const uint8_t Resolution = 8;           // 8 bit Analog Input Resolution
const uint16_t start_state = 0xACE1u;   // Any nonzero start state will work.
```

```
const int pinCS = 53;                       // microSD communication pin
const uint16_t match[] = {25,26,27,28};     // Value to introduce in the OCR2A register.
const int Ncoef[] = {121,126,130,135};      // Number of coefficients to save at each
    measurement.


//-------------------------------------------------------------------------------
// Function Prototypes
//-------------------------------------------------------------------------------

void startADC(void);
void stopADC(void);
void setADCPrescaler( uint8_t Prescaler );
void setVoltageReference( uint8_t reference );
void setADCres( uint8_t res );
void disDI(void);
void measConfig(void);
void lcdConfig(LiquidCrystal_I2C lcd);

byte mean(byte *a);
double mean(double *a);
void startMeas(int fi);
void finishMeas(void);
void fftStuff(double* Re, double* Im);
void indSort(void);
void normalization(bool MODOVOID, double*Re);
void prediction(double*Re, double*c, bool*Output);
void saving(bool*Output,LiquidCrystal_I2C lcd);

//-------------------------------------------------------------------------------
// Global Variables
//-------------------------------------------------------------------------------

extern volatile byte vec[N];                // array of real values measured by the
    microphone
extern double Re[512];
extern double Im[512];
extern int Indices[512];
extern bool Output[24];

extern arduinoFFT FFT;
extern double start;
extern LiquidCrystal_I2C;

extern volatile int n;                      // volatile is used to share variables between
    ISR and main program
extern int fi;                              // flag to define which step is the following in
     the process
extern bool processing;                     // flat to determine if the interruptions are
    available

extern volatile uint16_t lfsr;              // Initialize the lfsr register
extern volatile uint16_t bit;               // Must be 16-bit to allow bit<<15 later in the
    code. Auxiliar variable
extern File myFile;
```

Arduino_code/arduino_def.h *excerpt*

E.3 Functions

```
//-------------------------------------------------------------
// functions.h
//-------------------------------------------------------------

//-------------------------------------------------------------
// Includes
//-------------------------------------------------------------

#include "arduino_def.h"


byte mean(byte *a)
{
  double b=0;
  for(int i=0;i<sizeof(a);i++)
  {
    b += a[i];
  }
  byte c=b/(double)sizeof(a);
  return c;
}

double mean(double *a)
{
  double b=0;
  for(int i=0;i<sizeof(a);i++)
  {
    b += a[i];
  }
  double c=b/(double)sizeof(a);
  return c;
}

void startMeas(int fi)
{
    processing = true;
    OCR2A = match[fi];                  // = (16*10^6)/(200000*8) - 1 (must be <256)
    TCNT2 = 0;                          // initialize counter value to 0
    TIMSK2 |= (1 << OCIE2A);            // initialize interruptions
}

void finishMeas(void)
{
    TIMSK2 &= B11111101;                // Parar las interrupciones
    processing = false;
}

void fftStuff(double* Re, double* Im)
{
    FFT.Windowing(Re, N, FFT_WIN_TYP_HAMMING, FFT_FORWARD);  // Weight data
    FFT.Compute(Re, Im, N, FFT_FORWARD);                     // Compute FFT
    FFT.ComplexToMagnitude(Re, Im, N);                       // Compute magnitudes

    myFile = SD.open("MEASURE.TXT",FILE_WRITE);              // Save the read values
    while (myFile){
        if(fi==1)
        {
            for(int i=0;i<Ncoef[fi-1];i++)                   // Each sampling frequency
    has its own coefficients to save
```

```
59              {
                  myFile.print(Re[i]);
61                myFile.print(',');
                }
63          }
            else
65          {
                for(int i=1;i<Ncoef[fi-1];i++)
67              {
                  myFile.print(Re[i]);
69                myFile.print(',');
                }
71          }
            myFile.print('\n');
73          myFile.close();
        }
75  }

77  void indSort(void)
    {
79    String str_aux;
      myFile = SD.open("INDICES.TXT");                    // Load indices to
        rearrange the spectrum
81      while (myFile){
          int ii = 0;
83        while (ii<512)
          {
85          str_aux = myFile.readStringUntil(',');
            Indices[ii] = str_aux.toInt();
87          ii++;
          }
89        myFile.close();
        }
91
        myFile = SD.open("MEASURE.TXT");                  // Read the measure file
        and put the coefficients in their place
93      while (myFile){
          for(int ii=1; ii<513; ii++)
95        {
            str_aux = myFile.readStringUntil(',');
97          Re[Indices[ii]] = str_aux.toFloat();
          }
99        myFile.close();
        }
101
    }
103
    void normalization(bool MODOVOID, double*Re)
105 {
        String str_aux;
107     if(MODOVOID==true){
           myFile = SD.open("VOID.TXT",FILE_WRITE);
109     }
        else{
111        myFile = SD.open("VOID.TXT");
        }
113
        while (myFile){
115        int ii=0;
          while (ii<512)
117       {
            if(MODOVOID==1){                              // If the void mode is active
        , the spectrum will be saved in the void file
```

```
119             myFile.print(Re[ii]);
                if(ii<511){
121               myFile.print(',');
                }
123           }
              else{
125             str_aux = myFile.readStringUntil(',');          // If the void mode is not
      active, the void spectrum will be read and the measured spectrum will be normalized
                Re[ii] = Re[ii] / str_aux.toFloat();
127           }
              ii++;
129         }
            myFile.close();
131         fi = 6;
        }
133 }

135 void prediction(double*Re, double*c, bool*Output)
    {
137   String str_aux;
      myFile = SD.open("OUTPUT.TXT");                           // File containing the
        weights of the layers to be applied (512x24)
139   while (myFile){
        for(int ii=0;ii<24;ii++){
141       Im[ii]=0;
          for(int jj=0;jj<512;jj++){
143         str_aux = myFile.readStringUntil(',');
            c[ii] = c[ii] + Re[jj]*str_aux.toFloat();           // The output value of a
      neuron is the weighted sum of its inputs
145       }
        c[ii] = 2.0 / (1 + exp(-2.0*c[ii])) - 1.0;              // And the nonlinearity added
         by means of the activation function (tanh)
147     }
        myFile.close();
149   }

151   myFile = SD.open("THRESH.TXT");                           // File containing the
        thresholds to compare each of the outputs (24)
      while(myFile){
153     for(int ii=0; ii<24; ii++){
          str_aux = myFile.readStringUntil(',');
155       Output[ii] = c[ii] > str_aux.toFloat();
        }
157     myFile.close();
      }
159   fi=7;
    }
161
    void saving(bool*Output, LiquidCrystal_I2C lcd)
163 {
      myFile = SD.open("RESULT.TXT",FILE_WRITE);                // Predictions are saved in
        result file
165   while (myFile){
        for(int ii=0;ii<24;ii++){
167       myFile.print(Output[ii]);
          myFile.print(',');
169     }
        myFile.print('\n');
171     myFile.close();
      }
173
      lcd.home();
175   for(int ii=0;ii<24;ii++){
```

```
      lcd.setCursor(ii%6,ii/6);
177   if(Output[ii]==1){
        lcd.print('1');
179   }else{
        lcd.print('0');
181   }
    }
183 }
```

Arduino_code/functions.cpp *excerpt*

E.4 Settings

```
1  //-----------------------------------------------------------------
   // settings.h
3  //-----------------------------------------------------------------

5  //-----------------------------------------------------------------
   // Includes
7  //-----------------------------------------------------------------

9  #include "arduino_def.h"

11 //-----------------------------------------------------------------------------
   // Start elements
13 //-----------------------------------------------------------------------------
   void startADC(void)
15 {
     sbi(ADCSRA,ADATE);          // enable auto trigger
17 //   sbi(ADCSRA,ADIE)          // enable interrupts when measurement complete
     sbi(ADCSRA,ADEN);           // enable ADC
19   sbi(ADCSRA,ADSC);           // start ADC measurements
   }
21
   void stopADC(void)
23 {
     cbi(ADCSRA,ADEN);           // disable ADC
25 }

27 //-----------------------------------------------------------------------------
   // Set and modify ADC prescaler
29 //-----------------------------------------------------------------------------
   void setADCPrescaler( uint8_t Prescaler )
31 {
     switch (Prescaler)
33   {
       case 2:
35       cbi(ADCSRA,ADPS2);
         cbi(ADCSRA,ADPS1);
37       sbi(ADCSRA,ADPS0);
         break;
39     case 4:
         cbi(ADCSRA,ADPS2);
41       sbi(ADCSRA,ADPS1);
         cbi(ADCSRA,ADPS0);
43       break;
       case 8:
45       cbi(ADCSRA,ADPS2);
```

```
      sbi(ADCSRA,ADPS1);
47    sbi(ADCSRA,ADPS0);
      break;
49  case 16:
      sbi(ADCSRA,ADPS2);
51    cbi(ADCSRA,ADPS1);
      cbi(ADCSRA,ADPS0);
53    break;
  case 32:
55    sbi(ADCSRA,ADPS2);
      cbi(ADCSRA,ADPS1);
57    sbi(ADCSRA,ADPS0);
      break;
59  case 64:
      sbi(ADCSRA,ADPS2);
61    sbi(ADCSRA,ADPS1);
      cbi(ADCSRA,ADPS0);
63    break;
  case 128:
65    sbi(ADCSRA,ADPS2);
      sbi(ADCSRA,ADPS1);
67    sbi(ADCSRA,ADPS0);
      break;
69  default: // Set 128
      sbi(ADCSRA,ADPS2);
71    sbi(ADCSRA,ADPS1);
      sbi(ADCSRA,ADPS0);
73    break;
  }
75}

77//-------------------------------------------------------------------------
// Set and modify Voltage Reference
79//-------------------------------------------------------------------------
void setVoltageReference( uint8_t reference )
81{
  //  REFS1 REFS0 Voltage reference
83  //  0 0 AREF, Internal Vref turned off
  //  0 1 AVCC with external capacitor at AREF pin
85  //  1 0 Reserved
  //  1 1 Internal 1.1V Voltage Reference with external
87  //      capacitor at AREF pin
  switch (reference)
89  {
  case 0:
91    cbi(ADMUX,REFS1);
      cbi(ADMUX,REFS0);
93    break;
  case 2:
95    sbi(ADMUX,REFS1);
      sbi(ADMUX,REFS0);
97    break;
  case 1:
99  default:
      cbi(ADMUX,REFS1);
101   sbi(ADMUX,REFS0);
  }
103}

105//-------------------------------------------------------------------------
// Change ADC resolution
107//-------------------------------------------------------------------------
void setADCres( uint8_t res )
```

```
{
  switch (res)
  {
  case 8:
     sbi(ADMUX,ADLAR);      // left align ADC value to 8 bits from ADCH register
     break;
  case 10:
   default:
    cbi(ADMUX,ADLAR);
    break;
  }
}

//-------------------------------------------------------------------------------
// Digital Input Disable Register 0
//-------------------------------------------------------------------------------
void disDI( void )
{
  sbi(DIDR0,ADC5D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
  sbi(DIDR0,ADC4D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
  sbi(DIDR0,ADC3D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
  sbi(DIDR0,ADC2D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
  sbi(DIDR0,ADC1D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
  sbi(DIDR0,ADC0D);           // Set Digital bit of Analog Input to one logic to reduce
    noise
}

void measConfig( void )
{

                                // set timer2 interrupt
  TCCR2A = 0;                   // set entire TCCR2A register to 0
  TCCR2B = 0;                   // same for TCCR2B
  TCNT2 = 0;                    // initialize counter value to 0

  // turn on CTC mode
  TCCR2A |= (1 << WGM21);
  // Set CS21 bit for 8 prescaler
  TCCR2B |= (1 << CS21);
  // enable timer compare interrupt
  // TIMSK2 |= (1 << OCIE2A);
  TIMSK2 &= B11111101; // Parar las interrupciones
}

void lcdConfig(LiquidCrystal_I2C lcd)
{
  lcd.begin(20,4);              // Initialize LCD 20x4
  lcd.setBacklightPin(3,POSITIVE); // Backlight configuration
  lcd.setBacklight(HIGH);

  lcd.home();
}
```

Arduino_code/settings.cpp *excerpt*

Part III

Specifications

Chapter 11

Specifications

In this part of the document, the specifications of the project will be stated so as to be able to repeat the simulations and the measurements.

11.1 Simulations

The computer used to simulate the theoretical model and to compute the data that will be used as train data set is an Intel(R) Xeon(R) processor X5690 at 3.46GHz, with 24GB of RAM, Windows 8.1 Pro as OS and an ATI FirePro 2260 GPU. Needed time to compute a 60000 sample data set is 5 hours.

There is another computer involved in the project, where the Artificial Neural Network model has been created and trained. This computer is an Alienware Aurora R5 version 1.0.16. Its processor is an Intel(R) Core(TM) i7-6700K at 4GHz with 16GB of RAM, Ubuntu 16.04.6 as OS and a NVIDIA Corporation GP104 GeForce GTX1080 GPU. Needed time to compute the Artificial Neural Network with the 60000 sample data set is 8 hours.

11.2 Structured plate production

The structured plate with its holes, as seen in the report document, has been crafted using a 3D printer. The 3D printer is a BQ Witbox with a PLA premium black wire 1.75mm diameter [5]. Its printing temperature is 210^oC, but it starts softening at 60^oC. Then it is not recommended to let the crafted piece sun exposed. The plate has been design using a open source CAD software called **OpenSCAD**. The program used to print the plate was customized as follows:

Parameter	Value
Quality	
Layer Height	$0.05mm$
Shell	
Wall Thickness	$0.625mm$
Wall Line Count	2
Top/Bottom Thickness	$0.625mm$
Top Layers	8
Bottom Layers	8
Infill	
Infill Density	20%
Infill pattern	Tri-Hexagon
Infill Line Directions	[ ]
Material	
Printing Temperature	210^oC
Build Plate Temperature	55^oC
Build Plate Temperature Initial Layer	55^oC
Flow	70%
Enable Retraction	Yes
Retraction Distance	$4mm$
Speed	
Print speed	$70mm/s$
Top/Bottom Speed	$50mm/s$
Travel speed	$120mm/s$
Initial Layer Print Speed	$35mm/s$

Table 11.1: *3D printer main parameter set up.*

11.3 Software and versions

Depending on the part of the project, one software or another one has been used. For model's simulation and data set creation, **MATLAB** has been used, with version **R2016a**.

In case of microcontroller's code, **Arduino IDE** is the chosen software. In this case, version 1.6.12 has been used. Furthermore, in the Arduino's code, some libraries have been imported:

Library	Version
arduinoFFT	1.4.0
SD	1.0
SPI	1.0
LiquidCrystal I2C	1.0.7
LCD	1.0.7
Wire	1.0

Table 11.2: *Arduino library version.*

Finally, the Artificial Neural Network has been developed using Python 3.6.6 and the following libraries:

Library	Version
numpy	1.16.1
scipy	1.2.0
tensorflow-gpu	1.12.0
matplotlib	3.0.2
random	1.1.0

Table 11.3: *Python library version.*

11.4 Guidelines for the correct operation of the system

Once the system is all together and ready to be used, the following recommendations should be considered:

- Temperature is a critical parameter. Try to avoid plate's sun exposure. Its material (PLA) starts deforming around 60^{o}, and this temperature can be achieved just by sun exposure.
- MicroSD card must not be taken from the microSD socket while the system is powered, under risk of formatting the microSD card and losing all the information inside. Remember that the microSD not only contains the result of the image, but it also contains crucial information about void measurements and rearranging coefficients.
- Avoid placing the system in noisy environments. Even if dissipation should be enough to mask the noise coming from outside the system, the microphone is sensitive enough to be distorted, so it is recommended to place the set up two meters away from the noise source.
- Avoid placing the system in non stable surfaces. Any movement suffered by the structured plate affects the measurement, adding a phase term that has not been considered in the training. Thus, platform movements can lead to a mistake in the prediction.
- Before start measuring, voltage at the output of the amplifier must be controlled. An output voltage of $2.5V$ must be found, in order to cover the maximum range of the microcontroller's input.

11.5 Material specifications

11.5.1 Solder

Tin used to weld must satisfy the directive on the restriction of the use of certain hazardous substances in electrical and electronic equipment, or european directive RoHS. Therefore, the tin will not have any lead content, and *green products* will be used, which are exempt of it.

11.5.2 Electronic components

Given that this product is not dangerous for the user's security, electronic components will not be brought under rigorous quality analysis, but their correct performance will be ensured. It will be compulsory to check the performance of each component before using them to craft the system.

Part IV

Budget

Chapter 12

Budget

In this chapter project's budget will be discussed, and a general idea of the total cost of the system will be established. It must be said that this budget has been done considering the costs of the proof of concept, which is the aim of the project. Therefore, this budget does not represent the production cost of the system. Moreover, the microcontroller considered in the budget is Arduino Mega 2560, even if Teensy 3.1 has been told to be a better option. In order to estimate this, several changes should be considered, such as the fact of buying a single PCB instead of a series production, which lowers costs. Prices when batch size increases are lower than when buying single products. Thus, a scale factor must be taken into account.

The first table (**Table 12.1**) shows hardware budget; in other words, materials cost of the project:

Component	Price
Signal acquisition	**2308.1€**
Microphone and preamplifier	1629.00€
Microphone's power source	593.00€
PCB manufacturing	86.05€
Signal emission	8.79€
Loudspeaker Kemo L10	8.79€
Structure	0.40€
3D printing plate	0.40€
Power supply	1.67€
$9V$ alkaline battery (Duracell 6LF22)	1.67€
LCD display	19.95€
HD44780U 20x4 LCD display	19.95€
Microcontroller and external memory	43.70€
Arduino MEGA 2560	35.00€
Arduino microSD shield	5.61€
$1GB$ microSD card	3.09€
Total	2382.60€

Table 12.1: *Material's budget.*

From **Table 12.1** it can be outlined that the microphone and its power source are much more expensive

than the rest of the equipment. In fact, these two items represent the 93% of the material's budget. In this Table, VAT has already been considered (21%). PCB manufacturing price can be divided in cost of PCB (bare board), cost of electronic components, cost of PCB assembly and cost of testing. This division is shown in **Table 12.2**:

Component	Price
Bill of Materials	**1.05€**
Resistor $15k\Omega$ 1%	0.08€
Resistor $2.2k\Omega$ 1%	0.06€
Resistor $12k\Omega$ 1%	0.07€
Resistor $1k\Omega$ 1%	0.06€
Resistor 51Ω 1%	0.09€
Resistor 820Ω 1%	0.09€
Capacitor $100nF$	0.14€
LM324N	0.46€
PCB board	**35.00€**
PCB assembly	**40.00€**
Testing	**10.00€**
Total	**86.05€**

Table 12.2: *PCB cost development.*

Prices have been obtained from *www.pcbcart.com*. It has to be considered that only one PCB has been ordered; making the unity cost much more expensive than ordering several PCB. The PCB parameters to order it are the following ones:

- Material: FR4
- Layers: 2 Layer
- Material Details: Standard Tg 140C
- Part Number: SHIELD MEGA
- Board Type: Single Unit
- Board Size (width): $92.71mm$
- Board Size (height): $53.34mm$
- Quantity: 1
- Thickness: $1.2mm$
- Surface Finish: ENIG (Gold)
- Copper Weight (Finished): $35\mu m$
- Min. Tracing/Spacing: $0.20mm(8thou)$
- Min. Annular Ring: $0.30mm(12thou)$
- Smallest Holes: $0.40mm$
- Holes Numbers: 300-600

- Surface Mount: 1 side
- Soldermask: Both sides
- Peelable Soldermask: None
- Matt Color: None
- Silkscreen Legend: 2 sides
- Silkscreen Legend Color: Black
- Gold Fingers: No
- Slots in Board: No Slot in Board
- Testing: Yes
- UL Marking: No
- Date Code Marking: No
- Lead Time: 12 days
- Special Requirement Note: PCB is routed to outside line of Top layer

The non-material costs to take into account in the budget of the project are summarized in **Table 12.3**:

Component	Price
Software licenses and libraries	**2000.00€**
MATLAB R2016b	2000.00€
Spyder3	0.00€
Arduino IDE	0.00€
Computation hardware	**3701.11€**
Windows Intel computer	1428.89€
Ubuntu Alienware computer	2272.22€
Human resources cost	**19040.00€**
Salary	19040.00€
Total	**24741.11€**

Table 12.3: *Non-material budget.*

Salary from **Table 12.3** has been computed as follows: 17€per hour has been considered. Thus, a salary of 2720€for a full-time job in a month. Then, the total amount has been computed as $2720/2 \cdot 6 + 2720 \cdot 4 = 19040$€, where 6 is the number of part-time job months, and 4 is the number of full-time job months. Finally, the total budget of the project is **27124.71€**.