Telmo Ivani Costa Carvalho Barros

Licenciado em Ciências de Engenharia Eletrotécnica e de Computadores

Data Acquisition Channel for an E-Nose

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Nuno Paulino, Auxiliary Professor,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Data Acquisition Channel
for an Electronic Nose

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

This dissertation represents more than just a piece of paper for me. It is the culmination of six amazing years where I grew not only as a student, but also as a person. This project would not be possible without the guidance of my mentor, Professor Nuno Paulino as well as the help of my colleagues at the Chemical Department, Gonçalo Santos and Ana Carolina Pádua, from the Faculty of Sciences and Technology of the Nova University of Lisbon. They have always demonstrated openness to any question I could've had and supported me to pursue this ordeal even when I felt a little disorientated.

Once again, I would like to thank my advisor. Professor Nuno Paulino had an immensely great amount of patience and always showed availability to help me.

To Professor Helena Fino, I would also like to thank because she showed me, unintentionally, that it is possible to grow up career wise and still maintain her own simplicity and humbleness.

I would also like to thank Professor Rui Tavares for his constant good sense of humour, something that tends to be forgotten once the workload increases.

To my great group of friends, I want to say thank you so much. You are definitely one of the main reasons I'm achieving so much.

I would also love to thank my mother and sister for being my foundation. Everything good I do is with them on my mind first.

Last, but not least, I'd love to thank my father. I can't show him the result of his excellent education on me, but I know this would have made him proud.

Abstract

Nowadays, electronic devices are ubiquitous. This spread across all scientific fields has allowed for greater development of solutions for problems once thought impossible, or impractical, to solve. Most specifically, it leads to a constant cost reduction in previously expensive technology, to a point where university students can acquire these devices and tinker with them, even finding applications that were not initially thought for the device. As a result, we began the development of this e-nose system, since it became increasingly important to know and understand the constituents and source of a chemical sample.

This dissertation arises from the need to integrate the computational power inherent to a low-cost microcontroller unit, to identify chemical substances and their characteristics. With the main goal set, it was possible to formulate the guidelines to achieve a successful data acquisition channel for an e-nose system implementation with reduced costs, size and fast execution. The starting point consisted in understanding how previous implementations of this system work, as well as its most basic components. After researching and studying e-nose technologies, I was able to define two separate segments for a data acquisition channel: acquisition & communication module (hardware) and a data pre-processing module (software). By using a conductive polymer sensor with a PSoC 5LP microcontroller unit, it was possible to detect changes in the sensor response according to the chemical sample being tested, acquire this signal with the Delta Sigma ADC and, straight away, send this data to the computer where it would be pre-processed.

This stage was implemented in Python and with its extensive signal processing libraries. Its main goal was to receive and store the data so that it could be use on a future pattern recognition algorithm to reach the final e-nose conclusion. Since this displayed to the user, a GUI was developed, and the resulting acquisitions could be displayed in several plots, as a way of verifying a successful operation.

In the end, it was a success, since after performing acquisition, communication and displaying the results, subtle differences were visible on the plots displayed.

Keywords: ADC, e-nose, microcontroller

Resumo

Atualmente, os dispositivos eletrónicos são omnipresentes. Esta disseminação nos diversos campos científicos permitiu um maior desenvolvimento de soluções para problemas que antes eram considerados impossíveis ou impraticáveis de resolver. Mais especificamente, leva a uma constante redução do custo da tecnologia, ao ponto de estudantes universitários poderem adquirir estes dispositivos e trabalhar com eles, até mesmo encontrar soluções que não foram inicialmente pensadas para o dispositivo. Como resultado, iniciamos o desenvolvimento deste sistema nariz eletrónico, uma vez que se tornou cada vez mais importante conhecer e compreender os constituintes e a fonte de uma amostra química.

Esta dissertação surge da necessidade de integrar o poder computacional inerente a um microcontrolador de baixo custo, para identificar substâncias químicas e as suas características. Com o objetivo principal traçado, foi possível formular as diretrizes para obter um canal de aquisição de dados bem-sucedido para a implementação de um sistema de nariz eletrónico com custos e tamanho reduzidos e execução rápida. O ponto de partida consistiu em compreender implementações anteriores deste sistema, bem como quais os componentes fundamentais. Depois de pesquisar e estudar as tecnologias do nariz eletrónico, consegui definir dois segmentos para um canal de aquisição de dados: módulo de aquisição e comunicação (hardware) e um módulo de pré-processamento de dados (software). Utilizando um sensor de *conductive polymer* com um microcontrolador PSOC 5LP, foi possível detetar alterações na resposta do sensor de acordo com a amostra química, adquirir este sinal com o ADC Delta Sigma e, seguidamente, enviar estes dados para o computador onde seriam pré-processados.

Esta fase foi implementada em Python e com as suas extensas bibliotecas de processamento de sinais. O principal objetivo consistiu em receber e armazenar os dados para que pudessem ser utilizados num futuro algoritmo de reconhecimento de padrões para chegar à conclusão final do nariz eletrónico. Sendo este módulo exibido ao utilizador, foi desenvolvida uma GUI e as aquisições resultantes são exibidas em vários gráficos, como forma de verificar que o processo foi efetuado com sucesso.

No final, os diversos passos - aquisição, comunicação e exibição dos resultados – foram atingidos e é possível denotar diferenças nos gráficos exibidos.

Palavras-chave: ADC, Nariz Eletrónico, microcontrolador

Contents

List of Figures xiii

List of Tables xv

Acronyms xvii

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Goals . . . 2
- 1.2 Thesis Outline . . . 5

2 State-of-the-Art 7
- 2.1 Electronic Nose . . . 7
- 2.2 Electronic nose structure . . . 10
  - 2.2.1 Sensors . . . 10
  - 2.2.2 Data Acquisition Channel . . . 16
- 2.3 System-on-Chip . . . 19
  - 2.3.1 Identification Software . . . 27
- 2.4 Electronic nose past implementations . . . 27
  - 2.4.1 Identification of cannabis-based drugs . . . 28
  - 2.4.2 Identification of spoiled beef . . . 29
  - 2.4.3 Healthcare application . . . 31

3 Implementation 35
- 3.1 Data Acquisition and Transmission . . . 36
- 3.2 Data Pre-processing . . . 47
- 3.3 Transfer Function . . . 53
- 3.4 Data acquisition channel flowchart . . . 53

4 Results 57
- 4.1 Plot results . . . 58

5 Conclusion and Future Work 65
- 5.1 Final Remarks . . . 65

5.2 Future Developments 65

Bibliography **67**

A Microcontroller Configuration Code **71**

B USB configuration code **77**

C Start Screen of the Application code **81**

D Start Screen of the Application code **87**

E Data Acquisition Implementation code **93**

F Data Analysis Implementation code **99**

List of Figures

1.1 e-nose . . . 2
1.2 rc circuit . . . 2
1.3 rc circuit charging step response . . . 3
1.4 rc circuit discharging step response . . . 3

2.1 Cross-section of the skull, showing the location of the olfactory epithelium, sensory neurons, cribriform plate, olfactory bulb, and some central connections [14] . . . 8
2.2 Comparison between a mammalian and E-nose systems . . . 9
2.3 Principle of sensors used in E-nose sensing [8] . . . 11
2.4 Photograph of the metal electrodes coated with the conductive polymer . . . 13
2.5 Conductive polymer structure . . . 14
2.6 Conductive Polymer Conductivity . . . 14
2.7 Sample-and-hold circuit . . . 16
2.8 Sample-and-hold time diagram . . . 17
2.9 SAR ADC . . . 17
2.10 ΔΣ ADC diagram . . . 18
2.11 Arduino UNO . . . 21
2.12 Block diagram of the ATmega328 . . . 21
2.13 MSP430 board . . . 22
2.14 MSP430 block diagram . . . 22
2.15 FRDM-KE06Z . . . 23
2.16 FRDM-KE06Z block diagram . . . 23
2.17 PSoC 5LP . . . 24
2.18 PSoC 5LP architecture . . . 25
2.19 Electrical conductance of TGS842sensor towards exposures to studied drugs[6] 29
2.20 A typical raw signal|response of an electronic nose's sensor[13] . . . 30
2.21 Overview of Optical e-nose (the side view of the chamber only shows 2 sensors out of 4)[4] . . . 32
2.22 Sensor response to CH4 gas with different concentrations starts from 2.5 ppm to 100 ppm in two wavelengths 7.88 μm (left) & 3.4 μm (right).[4] . . . 32

3.1 Conductive Polymer based sensor . . . 36

3.2 Conductive Polymer Structure . . . 36
3.3 Conductive Polymer sensor response to rectangular wave . . . 38
3.4 RC filter schematic and breadboard implementation . . . 39
3.5 DelSig ADC component . . . 40
3.6 DelSig ADC component configuration . . . 41
3.7 DAC component . . . 42
3.8 DAC component configuration . . . 42
3.9 DMA component . . . 43
3.10 DMA component configuration . . . 43
3.11 Ping Pong DMA schematic . . . 44
3.12 USBFS component . . . 44
3.13 USBFS component configuration . . . 45
3.14 Complete data acquisition circuit . . . 46
3.15 Application start screen: console and GUI . . . 48
3.16 Data acquisition GUI window . . . 49
3.17 Comparing the output of 6 plots . . . 50
3.18 E-nose system database . . . 51
3.19 E-nose test bench . . . 51
3.20 E-nose test bench . . . 52
3.21 Flowchart of this e-nose project . . . 54

4.1 RC circuit Python simulation . . . 58
4.2 RC circuit breadboard output . . . 58
4.3 Sensor baseline response to a 1 V signal . . . 59
4.4 Sensor baseline frequency response . . . 59
4.5 Sensor exposed to chemical compound response to a 1 V signal . . . 60
4.6 Sensor exposed to chemical compound frequency response . . . 60
4.7 Sensor exposed to chemical compound response to a 100 mV signal . . . 61
4.8 Sensor exposed to chemical compound frequency response . . . 61
4.9 Sensor exposed to chemical compound response to a 10 mV signal . . . 62
4.10 Sensor exposed to chemical compound frequency response . . . 62
4.11 Sensor output responses compared . . . 63
4.12 Sensor frequency responses compared . . . 63

List of Tables

2.1 Commercially available e-noses [17] 19
2.2 RISC and CISC architectures 20
2.3 Microcontroller comparison 26

3.1 Components used for the RC circuit 39

Acronyms

ADC	Analog to Digital Converter.
ANN	Artificial Neural Network.
API	Application Programming Interface.
CAN	Controller Area Network bus.
CISC	Complex Instruction Set Computing.
CP	Conductive Polymer.
CRC	Cyclic Redundancy Check.
DAC	Digital to Analog Converter.
DC	Direct-Current.
DMA	Direct Memory Access.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
FFT	Fast Fourier Transform.
FRAM	Ferroelectric RAM.
GPIO	General Purpose Input Output.
GUI	Graphical User Interface.
HOMO	Highest Occupied Molecular Orbital.
I/O	Input/Output.
I2C	Inter IC.
ICSP	In-Circuit Serial Programming.
ISA	Instruction Set Architecture.
K-NN	K- Nearest Neighbour.

KB	Kilobyte.
LSB	Least Significant Bit.
LUMO	Lowest Unoccupied Molecular Orbital.
MCU	Microcontroller Unit.
MHz	Megahertz.
MOS	Metal-Oxide Semiconductor.
MSB	Most Significant Bit.
PC	Personal Computer.
PCA	Principal Component Analysis.
PMC	Power Management Controller.
PSoC	Programmable Sytem-On-Chip.
PWM	Pulse Width Modulator.
RAM	Random Access Memory.
RC	Resistor-Capacitor.
RISC	Reduced Instruction Set Computing.
RTC	Real Time Counter.
SAR	Successive Approximation Register.
SFDR	Spurious-Free Dynamic Range.
SNR	Signal to Noise Ratio.
SoC	System-on-a-Chip.
SPI	Serial Peripheral Inter.
SRAM	Static RAM.
UART	Universal Asynchronous Receiver-Transmitter.
USB	Universal Serial Bus.
VQFN	Very Thin Quad Flat No-lead.

Chapter 1

Introduction

1.1 Motivation

The first way of contact with the surrounding environment is through sensations resulting from interactions with several different sources and the nerve cells in our bodies. Therefore, the human body is considered a remarkable sensing device because it is capable of detecting changes on the surrounding environment and acting upon them. In other words, it can conclude whether something is good/beneficial or bad/dangerous. A particular case of this complete system is the human nose. It can determine the source of an odour based on particles captured by the receptors inside itself.

With the development of technology and increased understanding of the human body, it became possible implementing its sensory capabilities with electronic devices. For this project, the focus will be on the smell sensation and its implementation through e-noses. Essentially, an e-nose is a device built to emulate the behaviour of a mammalian nose. It accomplishes this feat by acquiring and analysing the characteristic response resulting for the detected odour. This allows for a great deal of improvement on human lives since it can be used for medical diagnostics [5], food quality [19], among others.

Usually, the architecture of an e-nose consists in two major segments. A sensor array, with a dynamic range wide enough to sense a number of diverse chemical compounds and an electronic interface capable of acquiring the incoming signal and examining its response to provide an answer regarding its source. The e-nose architecture is depicted on figure 1.1.

Figure 1.1: e-nose

1.1.1 Goals

This thesis' goal is the development of a data acquisition channel. This system can be divided in two major sections - sensor array and pattern recognition software. The former is implemented with a conductive polymer whose electrical characteristics change when exposed to different gases. The signal interface block has to apply an electrical signal (either current or voltage) to the sensor array and then measure the resulting electrical signal (either voltage or current) resulting from the sensor. The signal identification software has to analyse the measured electrical signal and determine the gas that was applied to the sensor.

Also, since the sensor being used is a conductive polymer [16], its response is known beforehand [1] . It's similar to that of a RC circuit present on figure 1.2, which is characterized by its time constant: $\tau = RC$. In this case, the data-acquisition channel implementation for an e-nose will revolve around extracting the response's transfer function and, indirectly, discovering the value of τ.

$$V_c = V_{in}(1 - e^{t/\tau}) \tag{1.1}$$

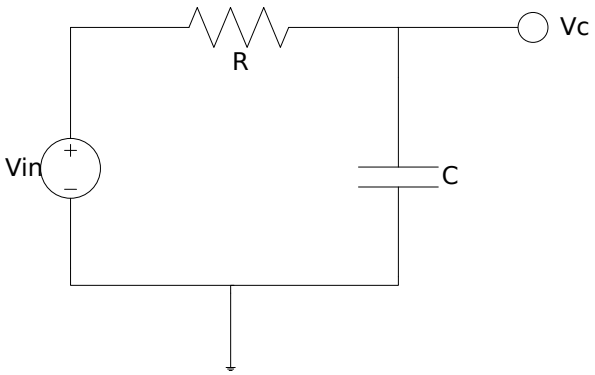

Figure 1.2: rc circuit

The expected sensor response when subject to a Heaviside signal is showcased on figure 1.3. After an initial charging phase, and when the sensor is no longer under the effect of a Heaviside signal, its output changes to the one present on figure 1.4, similar to the output registered when measuring a discharging capacitor.

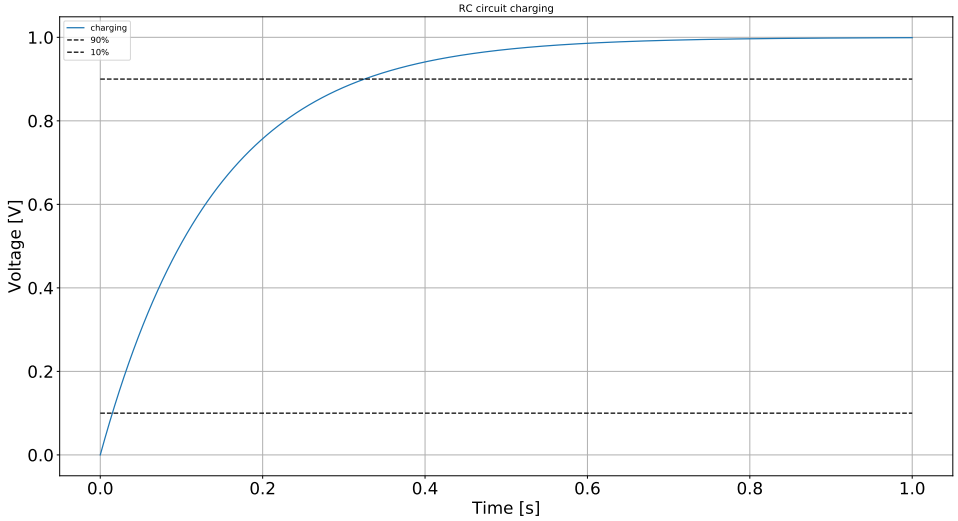

Figure 1.3: rc circuit charging step response

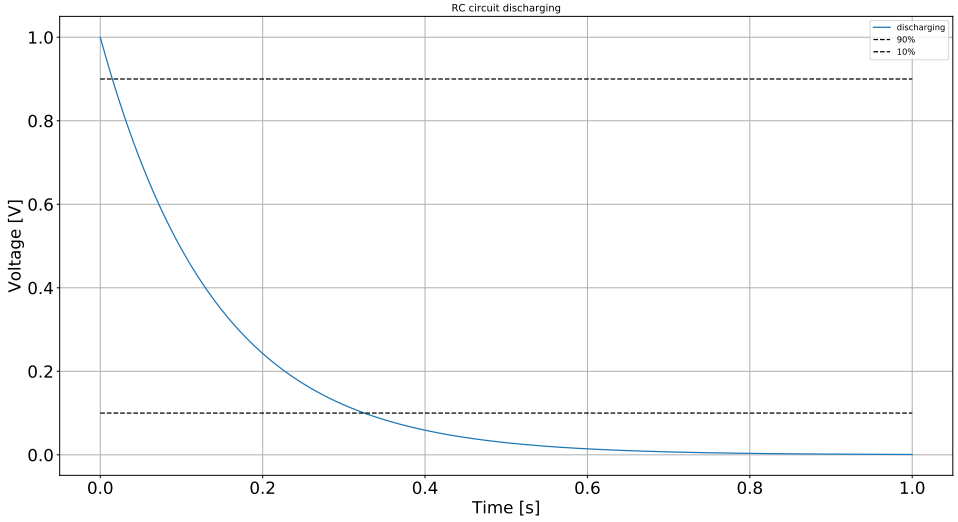

Figure 1.4: rc circuit discharging step response

As mentioned before, this system is possible because there is a known characteristic that differentiates every sample's response, but this signal needs to be converted from analog to digital, so that a computer can process it. This conversion step is achieved with a SoC. It'll be programmed to convert each analog sample acquired and transferring it to the computer. Essentially, a SoC is a device that is designed around a microcontroller chip, with memory and other peripherals (ex.: ADC, DAC, UART and USB). The introduction of this technology allowed for a faster development of solutions regarding data acquisition from the analog domain and its analysis, all in a small package.

In this case, the SoC of choice to implement the first stage was the PSoC 5LP because it has the necessary components to achieve a data acquisition channel: a DAC - to generate the signal that is applied to the sensor -, and ADC - Samples the analog signal according to a predefined sampling frequency. This allows a computer to process the incoming signal that is sent by the microcontroller through a communication protocol. After acquisition, data is transferred to a computer and this ends the first stage of the project. Afterwards, the acquired data goes through a pre-processing algorithm, programmed with Python. This final step is responsible for extracting the Bode plot from the incoming signal and use this result to obtain the mathematical transfer function characteristic of the sample being studied.

1.2 Thesis Outline

This dissertation is divided in 6 chapters which represent the flow of work during the development of this project.

The first chapter, as mentioned before, introduced the subject, as well as where it fits in the technological development that resulted from the improvement of SoC and sensor technology and data processing field. Also possible due to continuous steps towards more integrated circuits, where components such as DAC and ADC are embedded on the boards, instead of being peripherals.

On chapter two, the study of technologies that allowed this solution to be developed are presented in a deeper and more technical level, as well as giving real world examples of where a data acquisition channel is used, mainly regarding *Electronic noses* technologies. This is also the starting point of the dissertation, since it is the theoretical knowledge necessary to move forward with this project.

Following the theory and study of previous data acquisition channel solutions, is chapter 3. On this segment, the interface between sensor and data processing software is developed, according to certain specifications and constraints. It's a critical element because data processing is heavily dependent on the accuracy of the signal acquired by the programmed SoC. This chapter also covers the development of a software responsible for receiving data sent from the SoC. Adding to this, the program is also designed to perform a certain degree of data processing with the goal of obtaining an equation corresponding to the previously acquired signal, which will be used to perform the identification of future samples.

Chapter four contains the results obtained as well as comments regarding them.

Finally, chapter five presents the final remarks about the project. Its outcome is discussed and ideas for future improvements are mentioned for a possible iteration on this data acquisition channel.

Chapter 2

State-of-the-Art

The goal of this chapter is to introduce previous technology used to developed electronic nose solutions. In the first place, Its important to define clearly what constitutes an electronic nose and, afterwards, dive into the main components responsible for its implementation.

Even though the fundamental basis for the development of this technology has been established for some time, constant development on the microelectronics and signal processing fields, has allowed more efficient and robust solutions to arise, in addition too the improvement on sensor technology.

2.1 Electronic Nose

As mentioned before, this electronic device is created to replicate the behaviour of a biological nose, most specifically, the human olfactory system. The idea of its implementation was introduced by Dodd and Persaud, with the goal of recognizing characteristic odours such as fruity, grassy, earthy, malty, among others, based on the unique response of gas sensors when exposed to several substances. In other words, capturing the "flavour fingerprint" or determining the odour's source.[1].

To understand previous implementations of e-noses, it is essential to have an idea of how a mammalian nose can identify and its importance to mammals in general. According to [14], chemical sensing systems are essential to every living organism. They exist because chemical compounds have a profound influence on our lifestyle and, from simple bacteria to more complex animals it can be noted that these compounds can determine the way the organism lives. These influence different aspects of life such as feeding, detection of harmful conditions, sexual behaviour, among others. That is why, more complex organisms developed a sensing system, based on smell that could help distinguish good

from bad odour sources.

In this case, the focus is on the nasal cavity. It is responsible for detecting and processing information incoming from the detection of molecules coming from the odour. These are captured by the receptors inside a nasal cavity.

The incoming signals result from the interaction of the chemical compounds with the receptors in the olfactory epithelium. When detected these induce an electrical signal, that goes through the bulb and, finally, reaches the brain 2.1.

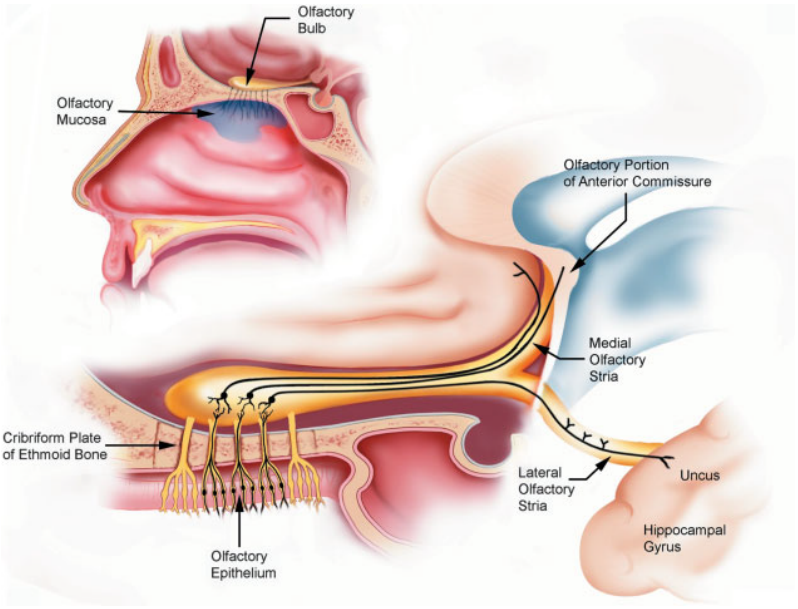

Figure 2.1: Cross-section of the skull, showing the location of the olfactory epithelium, sensory neurons, cribriform plate, olfactory bulb, and some central connections [14]

The complexity of this system depends on how sensitive it is. There is an estimation that only 2% of volatile compounds available in a single sniff will reach the olfactory receptors, and around 40 molecules of some mercaptans are sufficient to understand the structure of an odour [21]. Humans detect the presence of volatile organic compounds because their receptors have certain thresholds. Detection of molecules happens when these thresholds are overcome and, subsequently, captured by the aforementioned receptors..

Based on the previous description of a mammalian nose, it is easier to establish a comparison with its electronic counterpart. This is showcased on figure 2.2 :

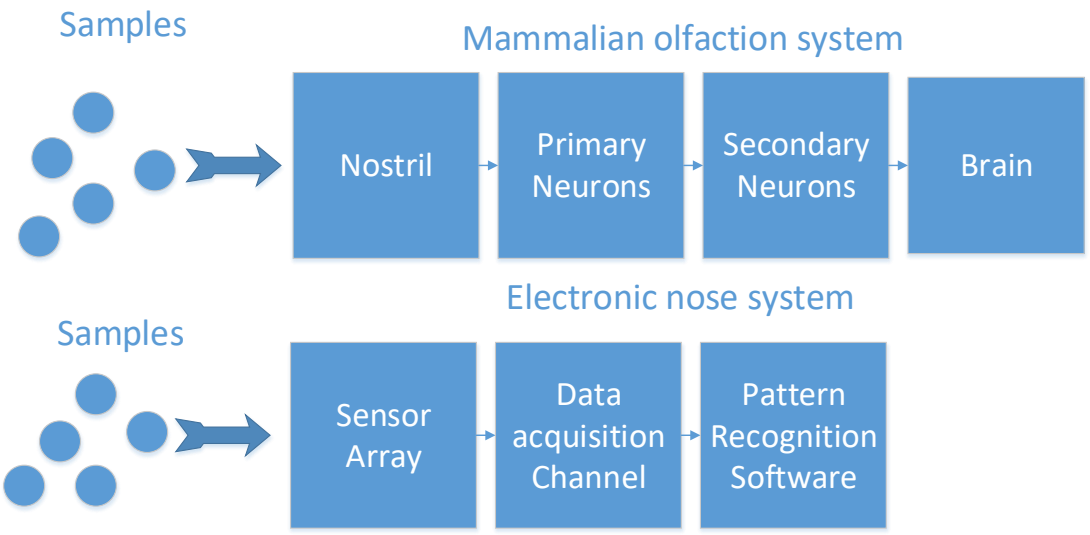

Figure 2.2: Comparison between a mammalian and E-nose systems

2.2 Electronic nose structure

The structure of an electronic nose has not suffered many changes over the years of its study and development, so it is possible to outline clearly which individual segments, when assembled, become an electronic nose system. According to the literature, it has the following structure:

- **Multi-sensor array | Sensors**
  Fundamental segment of the e-nose. It is how the chemical sample is detected. Several implementations are based on sensors that suffer a reversible physical and/or chemical change in the sensing material, when exposed to a substance. These changes can reveal themselves in the form of electrical properties, such as conductance. [7];

- **Data acquisition channel**
  As mentioned before, the electrical characteristics of the sensor suffer a deviation from their usual operating point when exposed to an odour. These changes must be captured and sent to a unit that can process it. This is the responsibility of a data acquisition channel. It must be able to sample incoming signals from the analog medium [5] and transmit it to the next stage, so that the signal can be analysed, and a decision can be reached;

- **Identification Software**
  As the name suggests, this sub block of the e-nose is responsible for reaching a conclusion regarding the odour's source, based on the input signal detected by the sensor. In other words, a pattern recognition software [6].

2.2.1 Sensors

The device responsible for detecting the odours being tested by the electronic nose is the sensor. Generally, an e-nose is implemented using an array of sensors [6], since it allows the detection of a wide range of odours, akin to a human nose. For this reason, sensors can be considered one of the most important segments of an e-nose. The correct implementation of an e-nose is heavily dependant on understanding sensor technology and how can it contribute for the accuracy of this system's readings. According to [25], sensor is a device that converts a physical measure into an electrical signal. As such, sensors represent part of the interface between physical world and its digital counterpart. Regarding an e-nose, the goal is to study chemical compounds and their "fingerprints", therefore, sensors used for these implementations are, according to [8], usually the following :

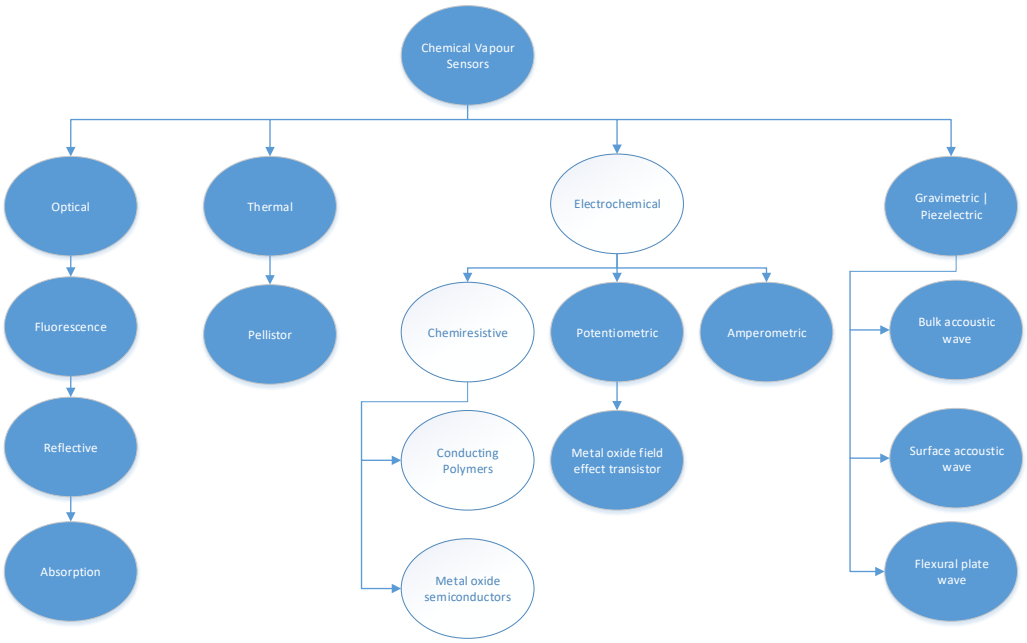

Figure 2.3: Principle of sensors used in E-nose sensing [8]

From the previous diagram, there are four main options regarding sensors for this specific application. Based on [8], each has the following description:

- Optical sensor
  The added value provided by an optical sensor rests in its capacity to simultaneously collect information regarding intensity and wavelength of the signal. It supports a range of techniques - absorbance, reflectance, fluorescence, refractive index and colorimetry. Currently, fluorescence has been the most explored technique. Despite this, there also have been developments on absorption and colorimetry techniques by Di Natale et al. [3] and Ralow and Suslick [15].

- Thermal sensor
  Even though this subtype has not been widely used, there are some examples of its implementation. It allows the detection of heat signatures - changes in temperature - of different samples based on their catalytic oxidation [2]. Usually, this is achieved with a design based on a catalytic surface, platinum, palladium and a heater to maintain sensor operating temperature. One benefit of this sensor is that humidity only slightly affects its performance. Major drawbacks are that it must operate at high temperatures – greater power consumption - and its high susceptibility to contamination from sulphur compounds.

- Electrochemical sensor
  As the name implies, the response of an electrochemical sensor depends on its conductance, when exposed to a chemical sample. This deviation is measured as a function of analyte concentration in an electrochemical cell when current flows through the electrode. This behaviour is caused by red-ox reactions of a catalyst on the electrode's surface. These sensors have a smaller dynamic range since they are sensitive to specific compounds.

- Gravimetric | Piezoelectric sensors
  Piezoelectricity was predicted and demonstrated by the Curie brothers in 1880 [12]. It is a phenomenon where a certain amount of mechanical stress translates to an electric current. This effect is results from crystal inside the sensor that generate electrical dipoles. With it, the inverse effect is also possible - applying a current that changes its mechanical properties. Nowadays, exist different variations of the Piezoelectric senor - bulk acoustic wave (BAW), surface acoustic wave (SAW), flexural plate wave (FPW) and shear horizontal acoustic plate mode (SHAPM), among others. From these options, the most used are BAW and SAW, however in recent years FPW have been used.

From the previously described sensor types, the most frequently used in the implementation of electronic noses are the electrochemical, more specifically, the chemiresistive sensors [26] [18] [6] [24]. The measured characteristic is their current|voltage oscillations. Depending on the sample being tested, the sensors resistance deviates from its default value, which originates a specific response.

Conductive Polymer As a chemiresistor, the conductive polymer's functionality is based on a deviation in its conductance. The particles responsible for this sensing capability are conductive polymer composites, such as polypyrrole and carbon black. [8]. This sensor type also has been found to have a wide sensitivity range of organic vapours and can operate at ambient temperatures with sensitivities of 0.1– 100 ppm - another reason for being widely used for electronic nose implementations [8].

Metal Oxide Semiconductors Similar to conductive polymers, MOS also suffer a deviation in their conductance, when exposed to a gaseous sample. Firstly showed to be gas sensitive in 1962 [8]. Besides a common main structure with a ceramic substrate coated by a heated wire resistor, they are also coated by one of two metal oxide semiconducting film types. On one hand, the metal oxide can be p-type - mainly cobalt or nickel - reducing compounds or of the n-type - mainly tin dioxide, zinc oxide, titanium dioxide, or iron (III) oxide - which corresponds to oxidizing compounds. Have been studied and applied in very diverse fields such as food analysis, medical diagnosis, air contaminants and explosives detection [26].

The sensor of choice for this data acquisition channel is the conductive polymer, from the electrochemical type and it can be seen on figure 2.4.

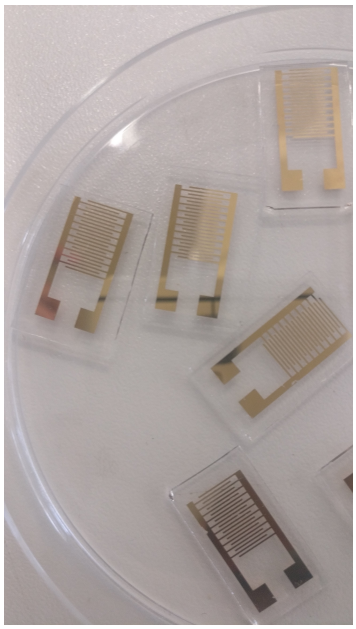

Figure 2.4: Photograph of the metal electrodes coated with the conductive polymer

These are desirable for electronic applications since they are conductive, flexible, functional and tunable. This happens due to their favourable electronic structure, which can be explained based on the energy band theory. It helps understanding the reason behind the conductor, insulator and semiconductor designations given to certain materials. In short, a band gap is the energy difference between the valence and conduction bands of a material. When the valence band overlaps the conduction band, the valence electrons are free to move and propagate to the conduction band. In the case of insulators, the band gap is too large for electrons to cross it. Conductors have a short band gap which allows an easy flow of electrons. Semiconductors sit in the middle because their band gaps are sufficiently short only to allow electrons to cross to the conduction band upon excitation, leaving behind a hole. This allows both hole and electron charge to move, which generates the current flow.

From the perspective of chemists, the electronic characteristic of conductive polymers that favours their wide usage is the presence of conjugated single and double bonds along the polymer skeleton. While the single bond contains a localized strong θ-bond, the double bonds include a localized θ-bond and a weaker localized π-bond.

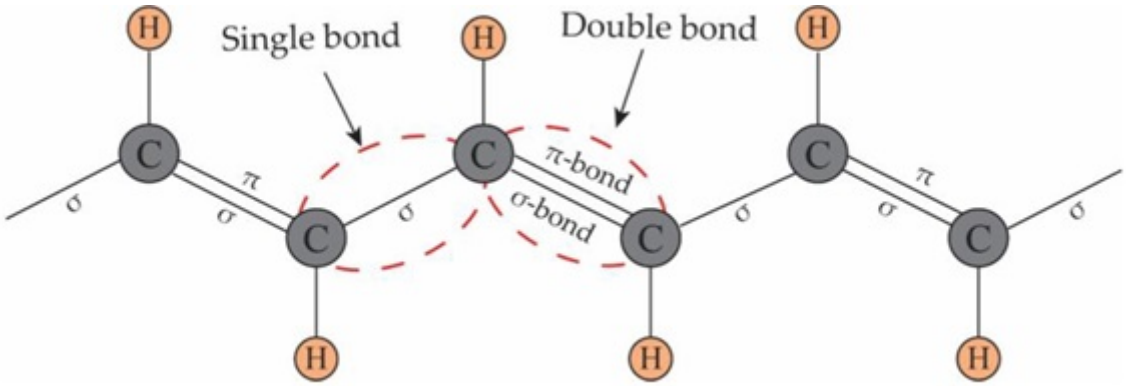

Figure 2.5: Conductive polymer structure

The electric flow results from the constant transfer of π-bonds that are between consecutive carbon atoms. The resulting effect will cause electrons in the double bonds to move along the carbon chain (the p_z-orbitals in the chain of π-bonds overlap continuously and the electrons in the π-bonds move along the carbon skeleton). Even though this creates a degree of current flow, it does not justify the widespread usage of conductive polymers. The breakthrough that caused their increased presence and predominance was achieved by Shirakawa, Heeger, and MacDiarmid, discovered that, by applying a halogen dopant would remove an electron from a delocalized bonding arrangement creating a hole. This hole will be filled by a neighboring electron, which will generate another hole on the previous position the electron was at, allowing charge to flow through the polymer chain. This doping effect can be seen on figure 2.6.

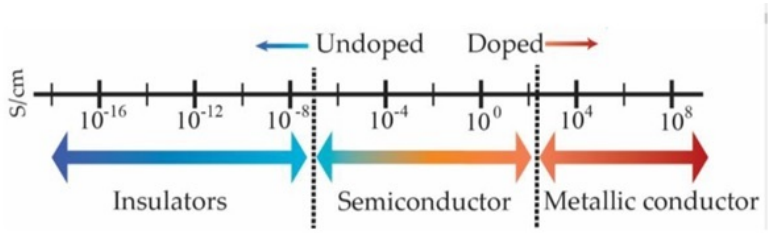

Figure 2.6: Conductive Polymer Conductivity

Later developments found that different methods of doping have achieved higher conductivities by conductive polymers. It has also been noted that, without any doping agent, a conductive polymer behaves as an insulator, but upon doping, their conductivity can

change from insulating to metallic. This behaviour results from dopants undergoing redox processes where transferred charges lead to subsequent charge carriers being formed. Therefore, a dopant does not only withdraw electrons but also adds them to the backbone of the conductive polymer. An explanation for this effect is that, when doped, electrons undergo a transfer from the highest occupied molecular orbital (HOMO) of the valence band (oxidation) or transferred to the lowest unoccupied molecular orbital (LUMO) of the conduction band (reduction)[10].

2.2.2 Data Acquisition Channel

This segment introduces the concept of a data acquisition channel and which technology is necessary for its development. When a system that interfaces with the analog domain, it must be capable of converting signals - analog - coming from sensors to digital format. This interface is accomplished with an ADC - analog-to-digital converter [9] [26]. As the name implies, an analog-to-digital converter is a peripheral that converts analog signals in a defined range to the corresponding digital outputs [20].
Usually, an ADC is considered to have the following components:

- Operational amplifier
- Sample and Hold circuit
- ADC

Operational Amplifier Ever since applications started interacting with real world signals, there is a need to use an interface that will act as a medium between the digital and analog domains. This interface is the ADC and it needs another electronic component that can drive it and amplify the source signal to its full analog input range. This component is the operational amplifier - OpAmp. Besides ensuring optimum signal-to-noise ratio (SNR) and spurious-free dynamic range (SFDR), it is also capable of converting signals from single-ended to differential. Its major drawback is the amount of noise it introduces on the circuit, since it is an active component. This can, ultimately, degrade conversion performance.

Sample and Hold The behaviour of a sample-&-hold circuit results from the current flow that charges the capacitor to a voltage equal to V_{IN} through R_{ADC}. Therefore, the complete circuit consists of an internal charging resistance and a hold capacitor - C_{ADC}-, bundled together by an electronically operated switch. As soon as the ADC conversion starts, the electronic switch is closed and a charging current flows through the circuit. The measure of how long this switch is closed is called sampling frequency - f_{ADC} -, and the equivalent time period is calculated by $t_{ADC} = \frac{1}{f_{ADC}}$, which can be seen on figure 2.7.

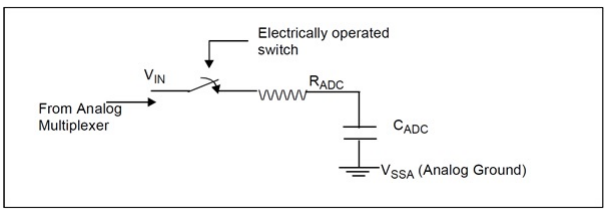

Figure 2.7: Sample-and-hold circuit

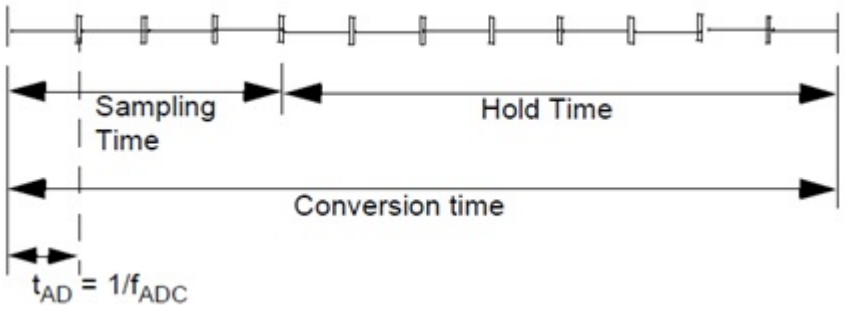

Figure 2.8: Sample-and-hold time diagram

Regardless of type, this block must be implemented with the Nyquist-Shannon theorem (cfr. Shannon theorem) in mind, which states that the sampling frequency must be equal, or greater, than twice the carrier's frequency, in order to eliminate all the frequency components higher than the Nyquist frequency (equation 2.1).

$$f_s \geq 2f_c \tag{2.1}$$

There are two main techniques for analog-to-digital converters: Successive-approximation-register (SAR) and delta-sigma (DelSig).

Successive approximation Register (SAR) ADC

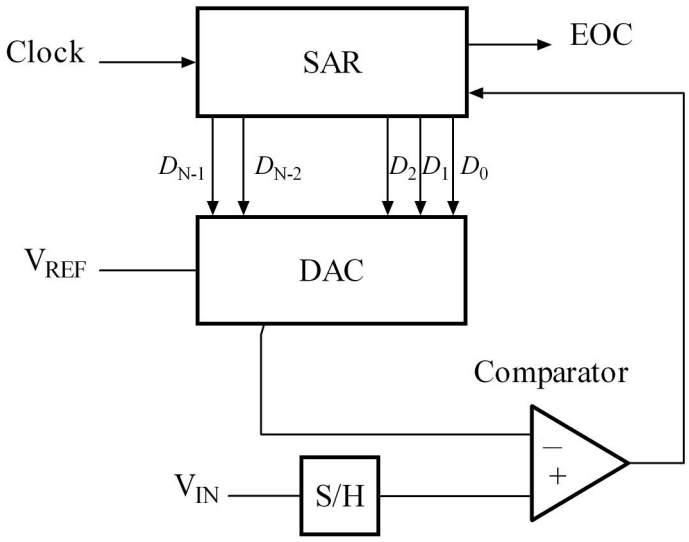

Figure 2.9: SAR ADC

Successive-approximation-register (SAR) analog-to-digital converter (ADC) is a technique for data conversion that is based on a binary search algorithm. It allows the ADC sample rate to be a fraction of the internal circuitry's speed, even if it is running at several megahertz (MHz). Applications that benefit from the use of this technique fall into the

medium-to-high resolution (8-16 bits) and a sample rate of 5 Mega samples per second (Msps). It is also widely used due to its low power consumption and small form factor. Thus, implementations such as pen digitizers and portable/battery-powered instruments systems can benefit from this technique.

Many iterations of the SAR ADC have been developed and implemented. Nevertheless, its fundamental architecture continues to be simple - figure 2.9. After holding the V_{IN} input voltage, it proceeds to compare this value with a reference - $V_{REF}/2$, where V_{REF} is the voltage provided to the ADC. Whether it is above or below the reference will cause the ADC output to shift between HIGH and LOW logic values, conversely 1 or 0. This is the binary search algorithm. It is executed from the MSB of the N-bit register until it reaches the LSB and, when finished, the N-bit digital word is available in the register.

Delta-Sigma($\Delta\Sigma$) ADC

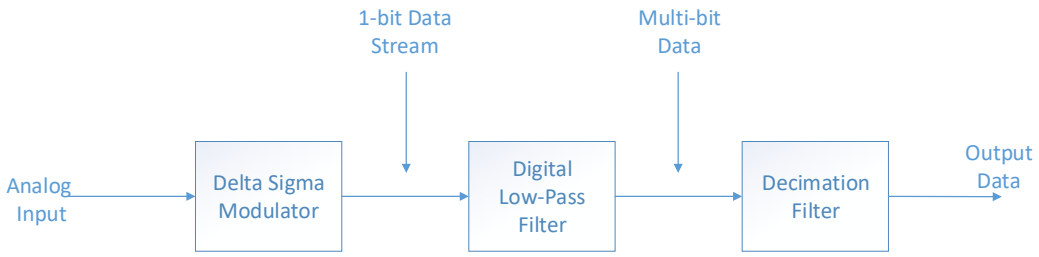

Figure 2.10: $\Delta\Sigma$ ADC diagram

Similarly to the SAR ADC, a delta-sigma (DelSig) ADC is capable of converting signals from the analog to the digital domain. Its design consists in approximately three-quarters digital and one-quarter analog.

The conversion process is based in oversampling the input signal over time, so that it can average the output of each sample. Therefore, the sampling rate is several times faster than the digital results at the output ports. In order to succeed with this implementation, the input signal needs to be somewhat slow (DC to several megahertz).

The $\Delta\Sigma$ converter's primary internal cells are the $\Delta\Sigma$ modulator and the digital/decimation filter. The first one samples the input signal at a very high rate into a 1-bit stream. The digital/decimation filter then takes this sampled data and converts it into a high-resolution, slower digital code. While most converters have one sample rate, the $\Delta\Sigma$ converter has two—the input sampling rate (f_s) and the output data rate (f_D).

2.3 System-on-Chip

Nowadays chips house several μ-processors together with components that used to be peripherals. These electronic devices are called System-on-Chip and, as the name suggests, these house a CPU and several other components in one chip, such as the DAC, ADC, USB, RAM, among others. With the current rate of technological development, the price of an SoC has been decreasing and their computing power has been increasing. This enables a bigger audience to tinker with the products and, regarding e-noses, allowed students to develop their own electronic nose solutions[11] instead of buying the more expensive commercial options, such as the ones displayed on table 2.1.

Table 2.1: Commercially available e-noses [17]

Model	Manufacturer	Technology
ChemPro100	Environics	ion mobility spectrometry
Air Quality Module	AltraSens	2 MOX sensors
FOX 4000	Alpha-Mos	18 MOX sensors (or QMB/CP)
oNose	Illumina	fluorescence sensorssbead array
ZNose 7100	Electronic Sensor Technology	GC and SAW

This technology is fundamental for the idea of a portable and still affordable e-nose. It is a compact integrated circuit designed to govern a specific operation in an embedded system. Due to the increase in integration of components on chips it is possible to develop a data acquisition channel with these since they can sample the sensor input and transmit the data to another device (most frequently a PC) where the identification procedure is executed. Microcontrollers used to be based on one of two methodologies: CISC or RISC. The former stands for complex instruction set computing and the latter means reduced instruction set computing.

The differences between each architecture type are outlined on table 2.2

Table 2.2: RISC and CISC architectures

CISC	**RISC**
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design: the ISA does as much as possible using hardware circuitry	Software-centric design: High-level compilers take on most of the burden of coding many software steps from the programmer
Most efficient use of RAM than RISC	Heavy use of RAM
Complex and variable length instructions	Simple, standardized instructions
May support microcode (microprogramming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

With continuous development of microcontroller technology, it became increasingly difficult to draw the line that distinguishes between CISC and RISC MCU.

There are a variety of Microcontrollers available and choosing one comes down to its price, size, peripherals and capabilities. After careful consideration, and with goal a portable, inexpensive and relatively powerful electronic nose, the following IC are good options for the development of an e-nose project:

- Arduino Uno from Microchip
- PSoC 5LP from Cypress Semiconductors
- MSP430 from Texas Instruments
- LPCXpresso Boards from NXP Semiconductors

Arduino Uno

Based on the ATmega328, by Microchip - implemented with RISC technology - Arduino Uno has 14 digital I/O, operates at 16 MHz (crystal quartz), USB connection, a ICSP header (allows reprogramming of the board without unplugging it from the computer) and a reboot button. It has 328 kB of flash memory, 2 kB of SRAM and 1 kB of EEPROM. The peripherals present on this board are 23 general I/O lines, a RTC (real time counter), three flexible timers/counters with comparators and a PWM, I2C, a 6 channel 10 bits ADC, a Watchdog timer with internal oscillator, a series SPI port and 6 software defined power saving modes.

Figure 2.11: Arduino UNO

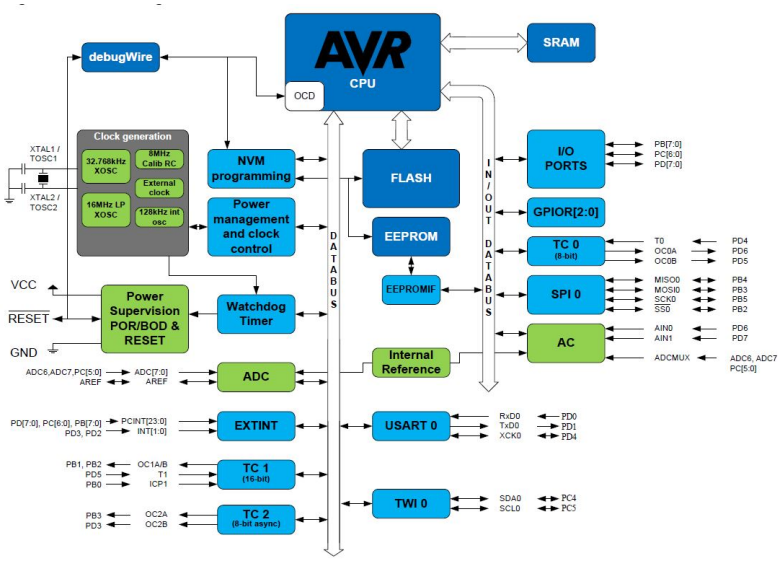

Figure 2.12: Block diagram of the ATmega328

Texas Instruments

The MSP430FR2433 device belongs to the MSP430™ family, which is part of the low-cost, with sensory capabilities and measuring MCU. The architecture implementation, as well as the FRAM and the integrated peripherals, with low-power modes and a VQFN (4mm x 4mm) , allow it to operate for large periods of time, even when used for sensing applications. The fast write speeds, flexibility and low-power are a result of the FRAM technology, which is a junction of RAM with the non volatility of Flash.

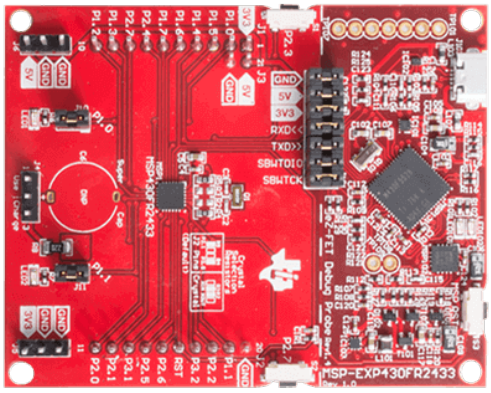

Figure 2.13: MSP430 board

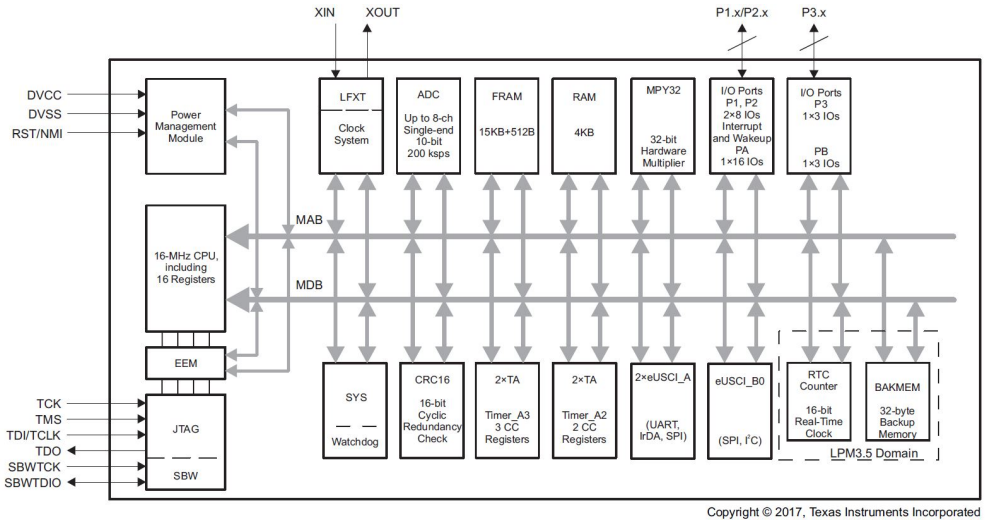

Figure 2.14: MSP430 block diagram

Freedom Development Board

NXO Semiconductors offers many devices that belong to the MCU universe. FRDM-KE06Z is part of the freedom development boards microcontrollers and its implementation is based on the Arm® Cortex®-M0+ processor. This microcontroller operates at 48 MHz, with 128 kB of Flash memory, 16 kB of RAM and 256 kB of EEPROM. It supports signals which amplitude varies between 2.7 V and 5.5 V, with -40ºC to 105ºC as working temperatures. As peripherals, it has a 12-bit SAR ADC, a 6-bit DAC, a Power Management Module (PMC), 71 GPIO pins, a programmable CRC (cyclic redundancy check module), a PWM, a series interface, a UART, a I2C and a CAN.

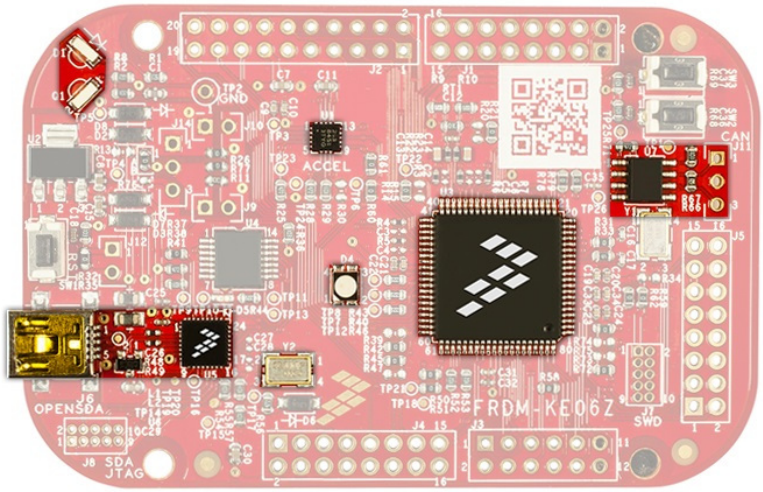

Figure 2.15: FRDM-KE06Z

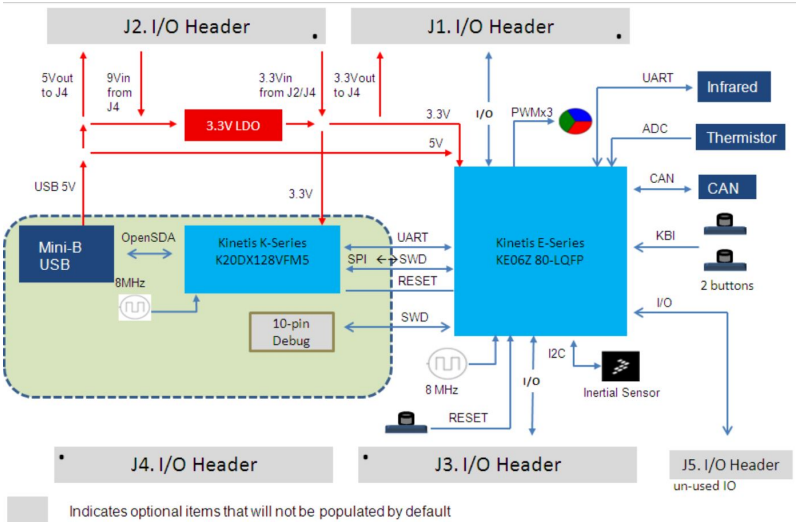

Figure 2.16: FRDM-KE06Z block diagram

PSoC 5LP

The PSoC 5LP is a microcontroller developed by Cypress Semiconductors. It comes equipped with a 32-bit ARM Cortex-M3, which operates at a maximum of 80 MHz, as well as with a 24 channel DMA controller. It has 256 kB of flash memory, 64 kB of RAM and 2 kB of EEPROM. This IC contains peripherals that cover both analog and digital domains, such as the Delta-Sigma ADC (8- to 20-bits), comparators, amplifiers, CapSense®support (to a maximum of 62 sensors), 62 GPIO pins. Its digital components include Timers, Counter and a PWM. Regarding communication, the available options are I2C, USB, CAN, among others. Its working temperature range goes from -40ºC to 85ºC and its voltage operating range is between 1.71 V to 5.5 V.

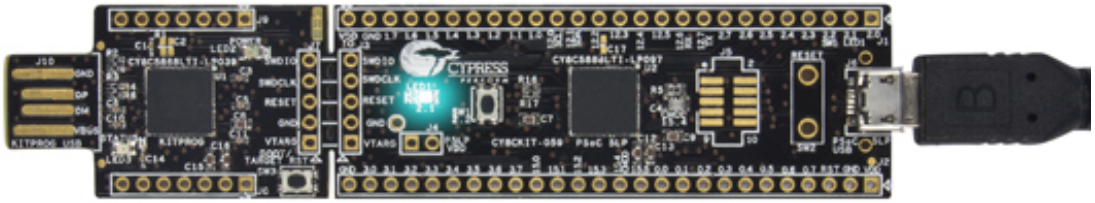

Figure 2.17: PSoC 5LP

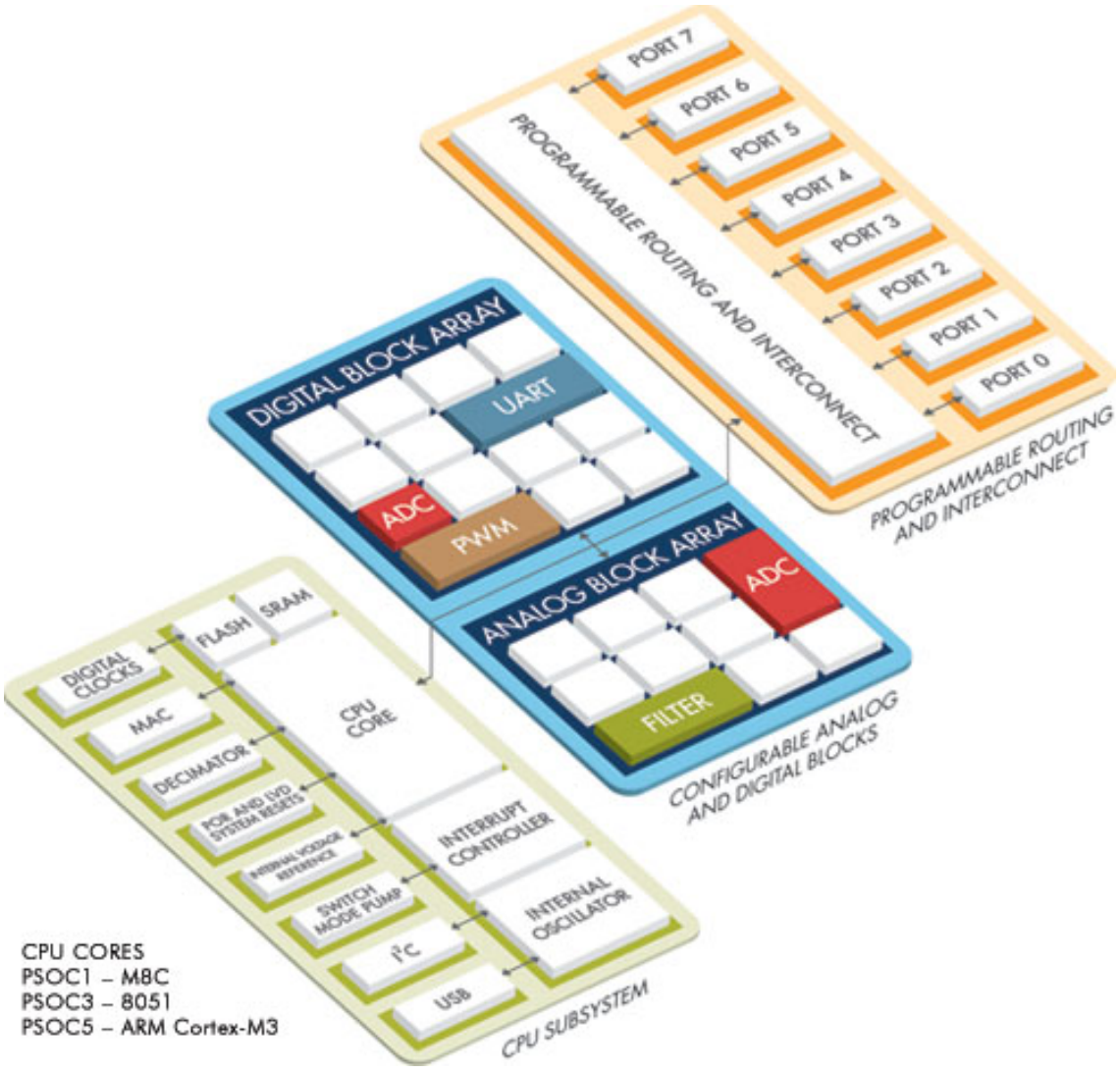

Figure 2.18: PSoC 5LP architecture

Table 2.3: Microcontroller comparison

Device	**Clock frequency (MHz)**	**GPIO**	**Flash memory (kB)**	**RAM (kB)**	**ADC**	**Other**
Arduino Uno	16	14	328	2	10-bit	USB, PWM, I2C
PSoC 5LP	80	62	256	64	8-20 bits Delta-Sigma	24-channel DMA controller, Capsense, PWM, Counter, Timer
MSP430	48	17-19	128	16	16-channel 12-bit SAR	6-bits DAC, two 8-bit I,Timer, UART, SPI, I2C, CAN
FRDM-KE06Z	48	71	128	16	12-bits SAR	6-bits DAC, PWM, UART, CAN, I2C

As it can be seen from table 2.3, these SoC are very similar in what they offer. To achieve a low-cost and portable electronic nose, it is mandatory that the unit chosen can offer great value, for a relatively low price. The chosen one was the PSoC 5LP because it fills the previous requirement. While it offers a very capable processor and an amount of memory only short of the Arduino UNO, it has the desired ADC for this application - $\Delta\Sigma$ ADC - and its price is the lowest - 10$ while price of the rest is between 13$ - 20$. It also has the lowest required operating voltage at 1.71 V, which is good for a possible low-power application. Another important factor on the decision of developing this project with the PSoC 5LP is its friendly development software GUI. It allows the user to design the entire circuit by simply dragging and dropping the components available on the board and programming based on its simple API.

2.3.1 Identification Software

The last part of the electronic nose system is the algorithm responsible for sample identification. From start, to finish, the sensors are exposed to a chemical sample that induces a change in their baseline response. This is captured by the data acquisition system, which acts as an interface between the analog and digital domains, being also capable of transmitting the data to another device that executes a software, based on an algorithm that can identify the sample origin.

Typically, pattern recognition software is based on classification algorithms such as K-Nearest Neighbour or Artificial Neural Networks [13, 22].

- **k-NN: k-Nearest Neighbour**

  In overview, the NN classification algorithm is very intuitive. After having well defined classes, new samples that need to be classified are designated to a class of their nearest neighbours. To increase the efficacy of this method, it became more useful to consider a certain amount of neighbours - k. Although it is meant to be a generalized way of classifying incoming samples, the k variable can be fine tuned to be responsive and more exact for the classification of certain samples.

- **ANN: Artificial Neural Network**

  This algorithm is based on the behaviour of a human brain. According to [23], it consists on more than 10 billion interconnected neurons responsible for receiving, processing and transmitting information through biochemical reactions. A neuron is made up of 3 parts: dendrite, where the nucleus is located; axon, a fibre that extends from the dendrite and which binds to other neurons through synaptic terminals or synapses. Thus, an ANN has artificial neurons as its constituent elements. The synapses of these neurons are represented by a mathematical model where weights represent the effect of each input and the non-linear characteristic of the neurons is represented by a transfer function. The neuron pulse is then calculated as the weighted sum of the input signals, obtained by a transfer function. The ability to represent the desired behaviour by the artificial neuron is achieved by adjusting the weights according to the chosen algorithm.

2.4 Electronic nose past implementations

The three technologies - sensors, microcontrollers and classification algorithms - presented above, have all been used on the development of past e-nose solutions in a diverse number of areas, such as identification of cannabis-based drugs [6], identification of spoiled beef [13] and, also for healthcare applications [4] .

2.4.1 Identification of cannabis-based drugs

Since it became the main producer of cannabis, Morocco struggled against the rise of illegal drugs smuggling in 2003. The main supplier of this substance was the European market, which led to an increase in market turnover of Moroccan cannabis. Also in 2003, this turnover was estimated to be around 12 billion US$.

One method chosen to control this issue was random searches at the border, using trained dogs. Another solution was using specialized equipments, but these were bulky and expensive, thus complicating their widespread use.

Ideally, the solution would involve a device that is portable, affordable and still able to execute the systematic process of sampling volatiles released by the person, or their luggage, followed by an analysis. The goal of finding this solution is what led to the development of an electronic nose system that focuses on detecting drugs, especially cannabis derivatives.

The first stage of this portable e-nose consisted on a sensor array with the following devices: TGS 8XX (with XX= 15, 21, 22, 24, 25 and 42). The peculiarity of these sensors is that, even though they have a widespread selectivity for gaseous substance, they still have a degree of affinity towards a specific gas or compound. The idea revolved around overlapping the sensitivities of each sensor to reach a decision on whether it had traces of cannabis derivatives.

The housing of this e-nose system also contained a relative humidity and a temperature sensors, to monitor the conditions of the experiment. Regarding the data acquisition segment, it was developed around a microcontroller - PIC16F877 – which was programmed in assembler language, via a serial RS232 communication port. Since the last step of the process revolved around analysing digital data, the incoming sensor signal had to be converted and this was achieved with a 10-bit resolution ADC. On the computer side, the data acquisition interface was implemented with LabVIEW©(National Instruments Inc., Austin, TX, USA). The pattern recognition software, responsible for sample identification, was developed using MATALB 7.0.1, from MathWorks Inc.

To test the efficacy of the developed electronic nose, the samples used - snuff tobacco, tobacco leaves, cannabis plants (leaves), cannabis buds and hashish - had to be in line with what was normally found during border searches. These were obtained from the court of justice (Meknes, Morocco).

After an initial phase of exposing the e-nose to the substance, its response sharply rose from its initial state, then it stabilized around 10 minutes after the begging of the procedure diue to sensor saturation.

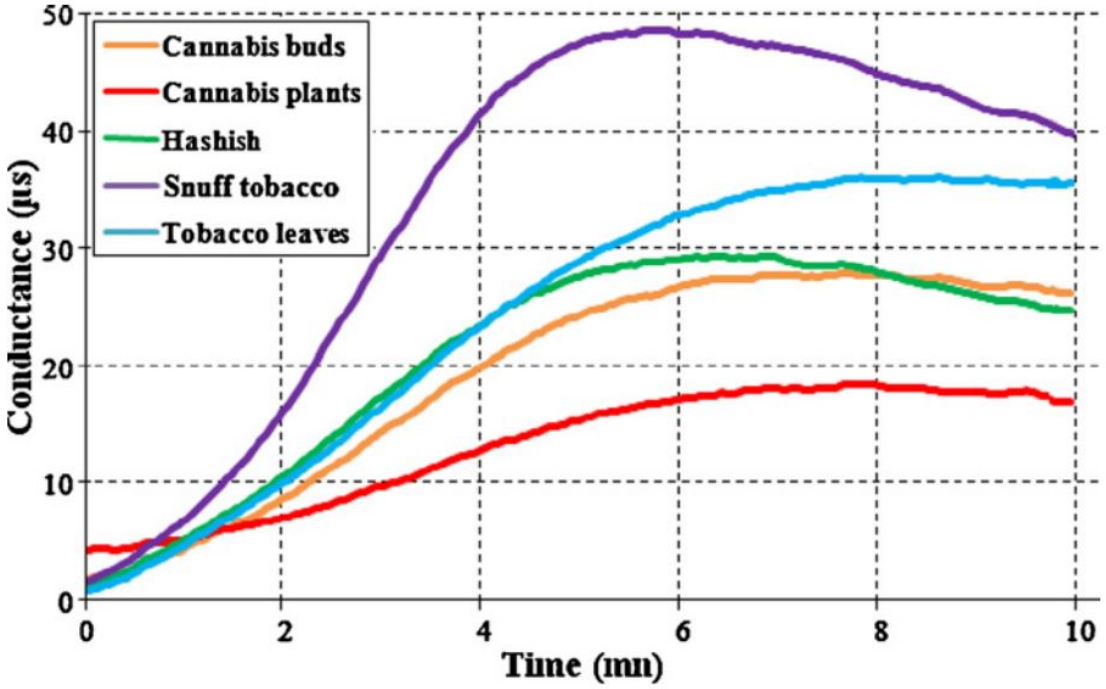

Figure 2.19: Electrical conductance of TGS842sensor towards exposures to studied drugs[6]

2.4.2 Identification of spoiled beef

Food contamination is a major issue regarding the health of a population. Therefore, it is mandatory to have a process that can constantly examine its condition.

On this study, the goal was to develop a means that could facilitate and accelerate how food analysis is performed, to avoid the consumption of spoiled meat products. For this reason, it is imperative to develop sensing techniques that could prevent situations, like food poisoning. Many of the so far developed methods are limited. They have only been applied in laboratories, are destructive, slow or convenient for field scale use.

The sensing technique implemented on this study considered the metabolic reactions present on food products that produce metabolites in the form of gas, solid and liquids. The micro-organisms would produce these substances and, by studying the headspace of the food product, its possible to determine the quality of a given food product. The sample used to develop this e-nose implementation was 100g of a beef strip loin (M. Longissimus lumborum) from different carcasses, obtained after a chilling period of 24 h. Each sample was further divided into two samples of 50g each and packaged in a way that emulated the environment they would have in a grocery store. Besides packaging, also the temperature levels were taken into account to produce an environment similar to that of a grocery store – between 10ºC and 4ºC, where 10ºC was meant to expedite meat spoilage and 4ºC is the typical temperature for meat conservation on grocery stores.

For this research project, the chosen e-nose system was the Cyranose-320TM electronic

nose (Cyrano Sciences, Pasadena, CA, USA). It consisted of an array with 32 conductive polymer sensors, and each was fine tuned to detect a specific chemical sample. As previously mentioned, the response of a conductive polymer is characterized by a deviation in its conductance and it depends on the level of exposure to the chemical being tested. After the sensing phase, the output is stored in a format supported by MS Excel for further analysis.

The typical response obtained by the e-nose during this study is showcased on figure 2.20

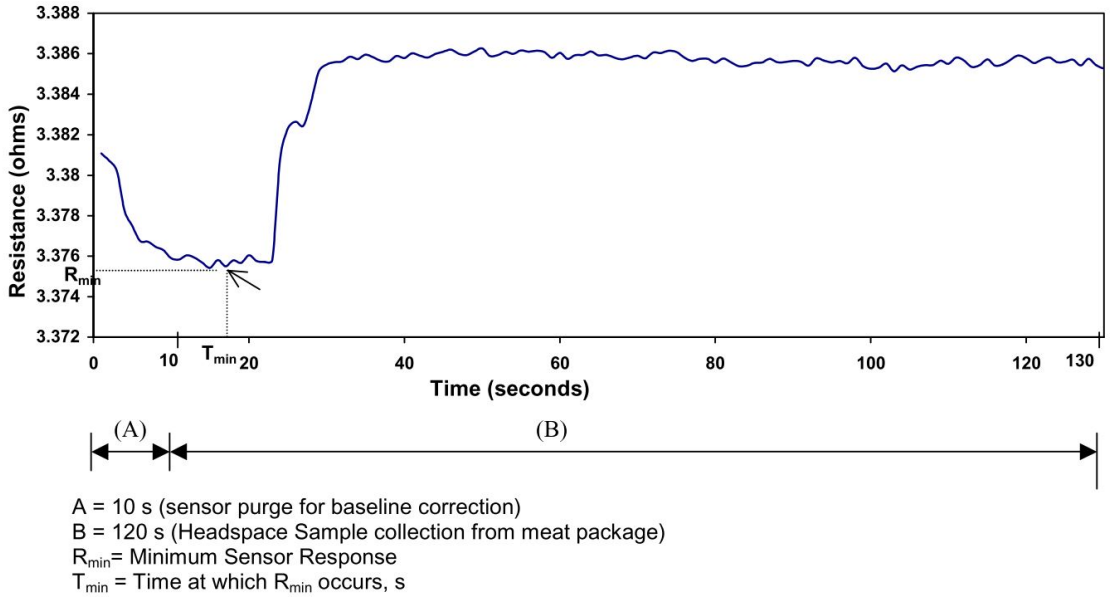

Figure 2.20: A typical raw signal|response of an electronic nose's sensor[13]

2.4.3 Healthcare application

The idea of implementing an e-nose in healthcare is related not only with the affordable price point, but also their non-invasiveness when performing a diagnose.

Regarding the structure of this implementation [4], it consisted on an array of high emissivity (EMIRS200) thermal infrared emitters aligned to four different optical tunable infrared detectors. This array was inside a chamber, which had within itself temperature and humidity sensors. They are responsible for monitoring the chamber's environment. This optical system is tunable. It has micro-machined Fabry-Pérot-Interferometer structures on top that employ a couple of parallel reflectors. This way, it is possible to tune the central wavelength received by the sensors. This optical system is controlled by an Arduino Edison via a touch screen.

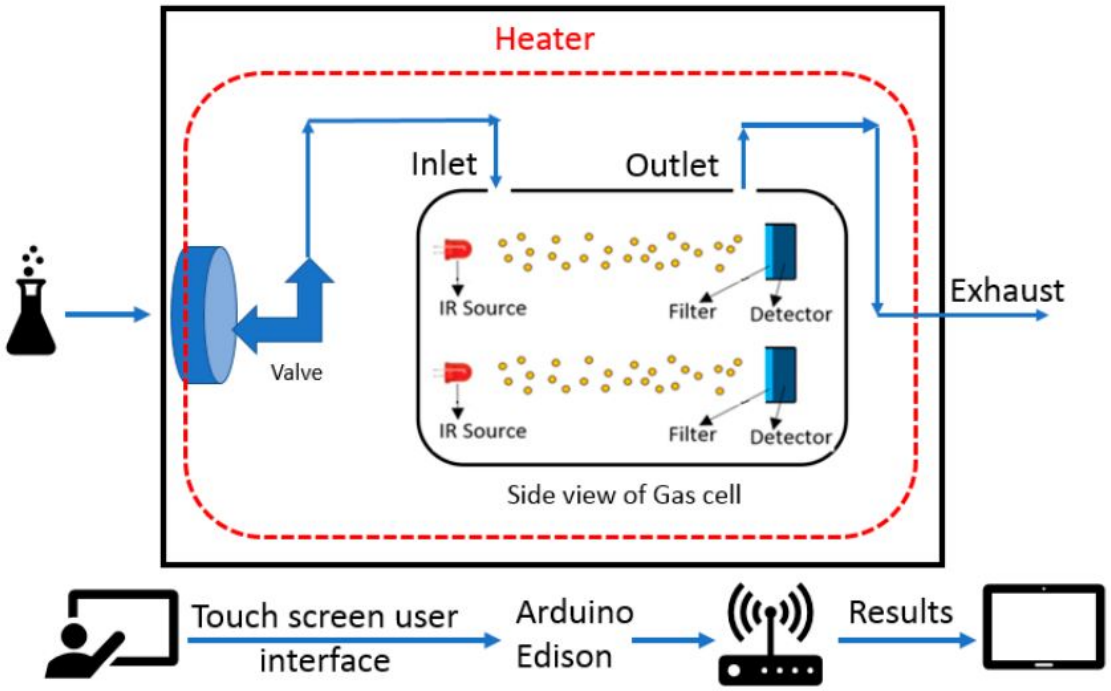

Figure 2.21: Overview of Optical e-nose (the side view of the chamber only shows 2 sensors out of 4)[4]

At the time, this system could perform up to 10 frequency readings per optical detector, which virtually meant that the array system had 33 sensors, sensitive enough to read between 3.1μm and 10 μm.

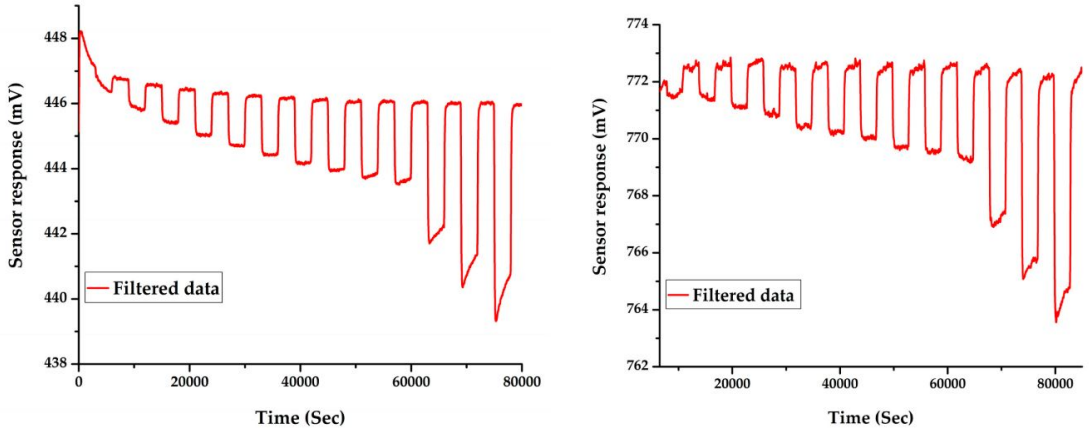

Figure 2.22: Sensor response to CH4 gas with different concentrations starts from 2.5 ppm to 100 ppm in two wavelengths 7.88 μm (left) & 3.4 μm (right).[4]

The results shown below are from tests performed on simple gas types, such as methane or ethanol. In the end, this e-nose system managed to show a level of sensitivity below

1ppm, for methane. Since many samples are not simple but a complex mixture of different substances, it should be able to examine these as well. This system managed to separate complex vapours such as acetone and ethanol.

Summary This chapter introduced the technology that is going to be developed and discussed on the remaining chapters of this thesis. According to previously mentioned sources and with the development of technology, an electronic nose has seen an increase in its presence in our daily lives. An integral part of it is the data acquisition channel, responsible for capturing the sensor response and processing it in a way that allows a computer software to execute a pattern recognition software successfully. There have been several iterations but more recent solutions have implemented this technology resorting to a microcontroller unit and an array of sensors that gives it a wide dynamic range of odours it can detect.

Chapter 3

Implementation

This chapter comprises the development and implementation of the data acquisition channel. It states which methodologies and choices were taken into account during each phase, as well as some explanations why a particular solution did not end up in the final prototype. Moreover, this section can be divided in two major areas: data acquisition, which also includes the transmission of an acquired signal, and data pre-processing, executed after the previous step is finalized. These two blocks were programmed in two different languages - C and Python. For this reason, it was important to decide which transmission protocol guaranteed data reception with low error.

3.1 Data Acquisition and Transmission

Essentially, the data acquisition segment implementation for this system revolves around selecting the correct sensor for our study and conversion of the sensor's response to digital format. This interface must allow, among other conditions, conversion of the signal with sufficient resolution to distinguish each specific sample response, possibility of applying different signals to the sensor, a configurable ADC, a reliable transmission protocol and to have a small form factor.

The sensor type chosen for this electronic nose implementation is the *conductive polymer*. On figures 3.1 and 3.2 is displayed the sensor appearance and structure, respectively. By applying a signal to it, a baseline response is obtained and, due to its chemiresistor properties - conductive polymer composites that react differently, depending on the sample -, by applying the same signal, a new response is obtained since the conductance was affected by the gas sample being studied.

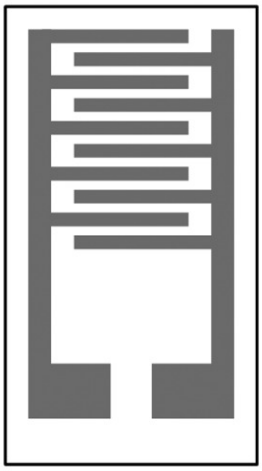

Figure 3.1: Conductive Polymer based sensor

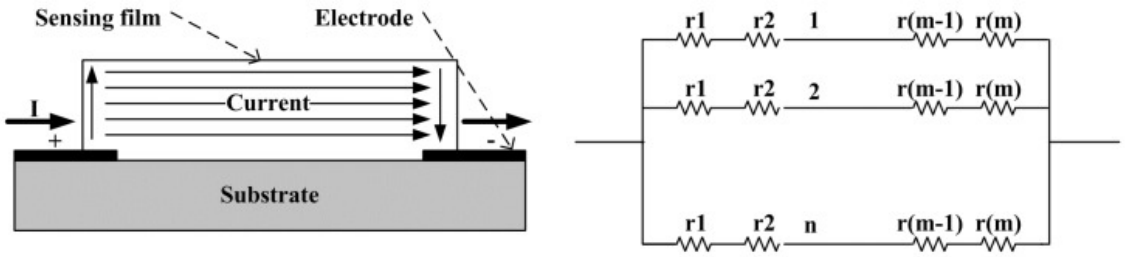

Figure 3.2: Conductive Polymer Structure

Based on previous studies [16], it was possible to determine which type of signal can achieve a better response from the Conductive Polymer. The resulting wave was of triangular type - amplitude: 80 mV_{pp}; frequency: 2 KHz. This way, a clear and distinguishable

response was obtained, as well as low levels of corrosion from the sensor composites.

Nevertheless, the approach chosen for this dissertation was different. At first, a rectangular wave is applied to the sensor during 30 seconds, purging the sensor with air and to discover the resulting response, displayed on figure 3.3. After, the sensor is exposed to the chemical substance for 30 seconds, achieving saturation around 20 seconds into the exposure period, the moment when the input signal is applied and, at the same time, the sampling procedure starts. It was clear that the sensor behaved exactly like a *low-pass* filter circuit. While the wave was rising and falling, there is a noticeable curve, resembling a capacitor charging and discharging, obtained at the sensor's output.

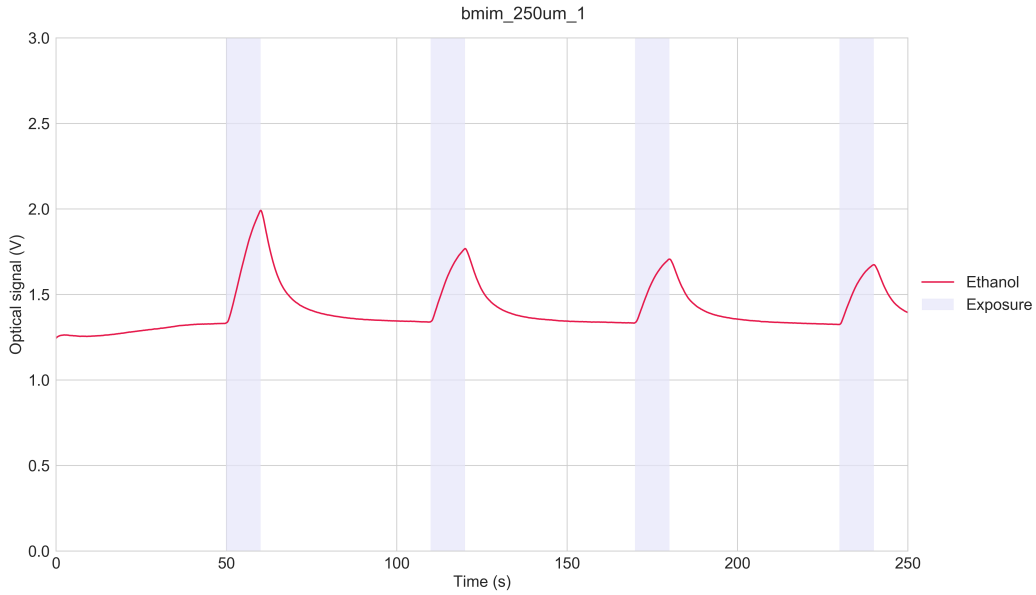

Figure 3.3: Conductive Polymer sensor response to rectangular wave

A conductive polymer sensor has a response that resembles the one obtained when measuring the output voltage signal from a capacitor. This behaviour is known as pseudo-capacitance - storage of electricity in an electrochemical capacitor - and is originated by a very fast sequence of reversible faradaic redox processes on the surface of the electrode. Therefore, a rc circuit (figure 3.4) was designed with the goal of fine tuning the data acquisition channel parameters before turning to the sensor – which wasn't completely characterized - that was going to be used in the final implementation.

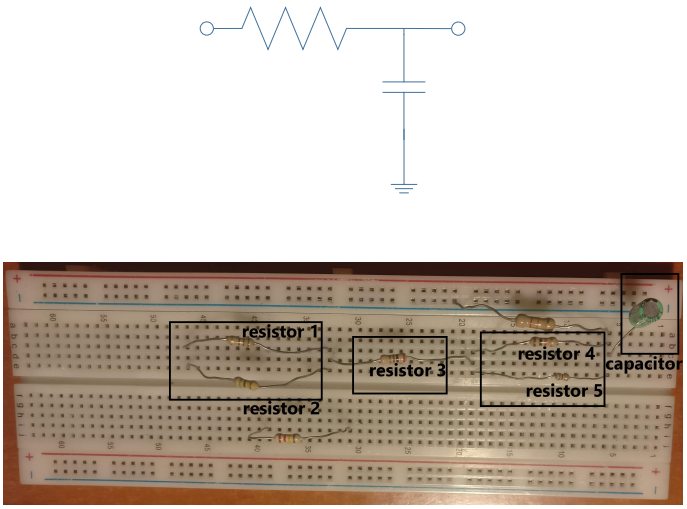

Figure 3.4: RC filter schematic and breadboard implementation

The components selected to implement the RC circuit used for testing are described on table 3.1. These where arranged in a way that allowed for a response that smoothly transitioned from rising to steady phase. As a result, resistors 1 and 2 are in parallel and equivalent to 303 Ω; resistors 4 and 5, also in parallel, are equivalent to 909 Ω. This translates in a final equivalent resistance of 1.4 kΩ.

Table 3.1: Components used for the RC circuit

Component	**Value**
Capacitor	10 μF
Resistor 1	510Ω
Resistor 2	750Ω
Resistor 3	200Ω
Resistor 4	10 kΩ
Resistor 5	1 kΩ
Resistor 6	82 kΩ

The obtained circuit response resulted from applying a signal at one end and acquiring the output at another. This is executed by an integrated circuit - PSoC 5LP®

(Programmable System-on-Chip) by *Cypress Semiconductors*. Its architecture includes the necessary components that allow both signal generation and data-acquisition - DAC (Digital-to-Analog converter) and ADC (Analog-to-Digital converter). For the acquisition interface, the choice was the Delta-Sigma ADC. This, opposed to the SAR (Successive Approximation Register) ADC, is better at filtering the incoming signal and can convert samples at higher resolution leves, which is ideal for an electronic nose, a system that relies heavily on little differences of incoming signals.

There is also the need to use a device with the capability of transferring the acquired data to a second device. It will be responsible for performing the necessary pre-processing on the acquired data, feeding it to a follow-up stage - identification algorithm. Thus, the choice of PSOC 5LP® . This device has several options to execute data transmission between itself and a computer, therefore, becoming the necessary interface between sensor and pattern recognition software.

On figure 3.5 it is present the ADC component. It is responsible for sampling the incoming signal response from the sensor. Its configuration revolved around establishing the resolution of each sample and the frequency conversion value. In this case, the goal was to have WORD long samples and a a rate of conversion as high as possible. On figure

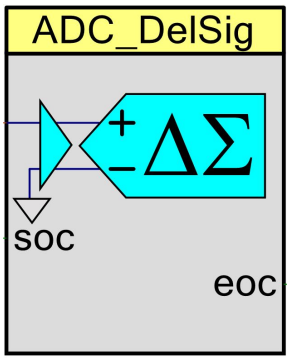

Figure 3.5: DelSig ADC component

3.6, are present the available options to configure the Delta-sigma ADC. One condition to obtain a good frequency representation using the FFT algorithm states that the wave must be periodic. Thus, the incoming signal had to be periodic. This presented an issue because not the first goal when configuring the ADC was choosing the maximum value for its sampling frequency. With the new requirement, a new value had to be calculated.

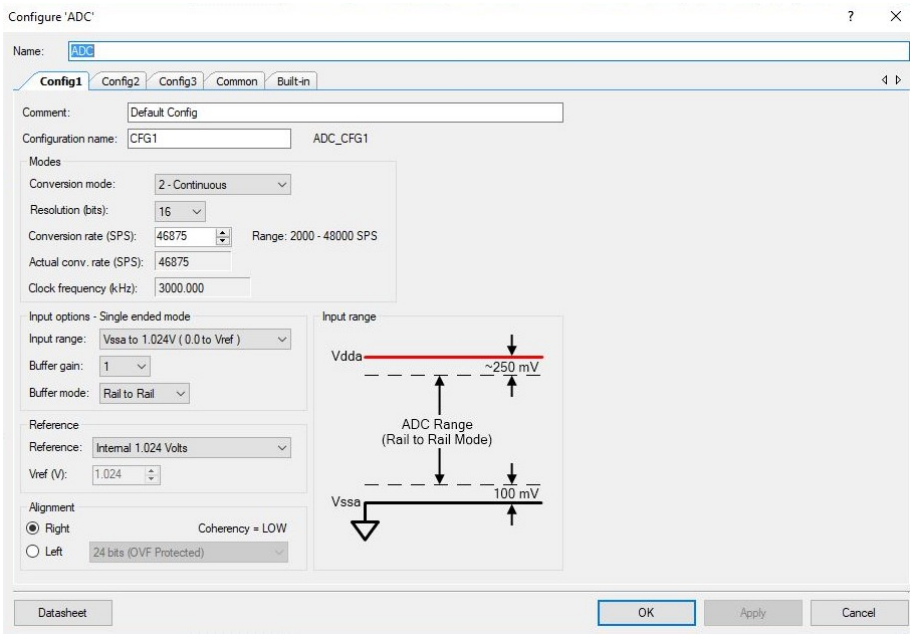

Figure 3.6: DelSig ADC component configuration

The DAC component showcased on figure 3.7, and its configuration on figure 3.8, is used to apply a heaveside wave to the sensor to obtain its response, whether it is exposed to a chemical sample or simply its baseline state. This allows a better study of the response on the frequency domain since it covers the whole spectrum. The three different values for each wave were chosen to test how would the response change according to an bigger or smaller amplitude value. The reason why it is only 3 DAC is because this is a limitation of the microcontroller.

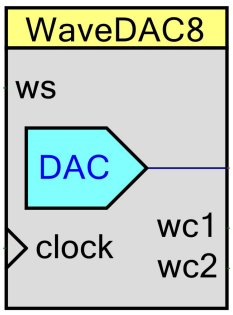

Figure 3.7: DAC component

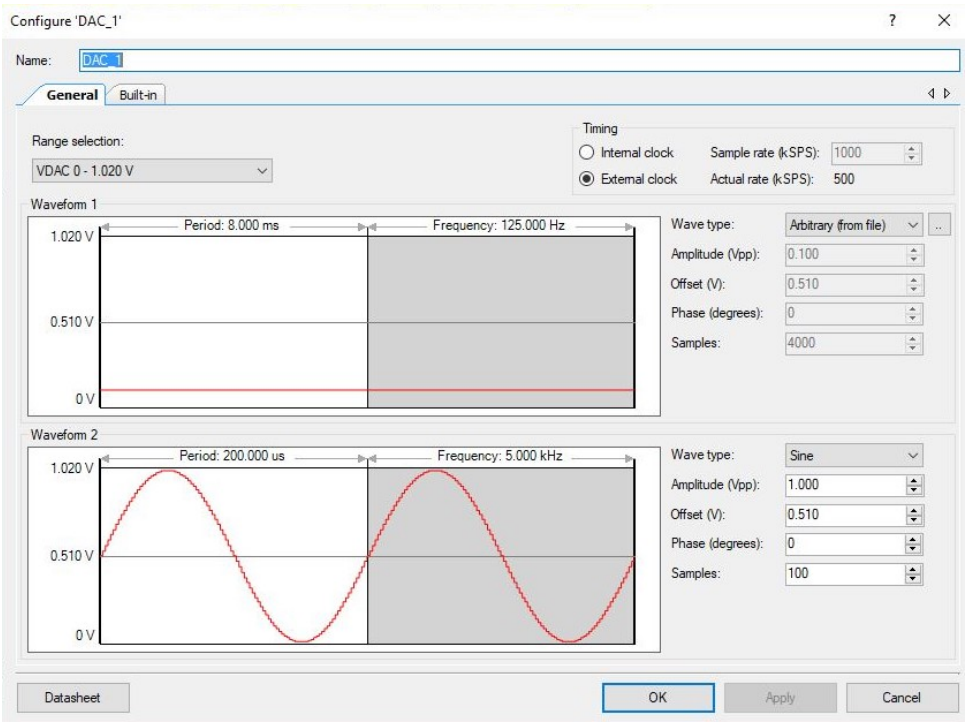

Figure 3.8: DAC component configuration

In such implementations, it is desirable to have some degree of flexibility. In its architecture, the PSoC 5LP® allows a maximum of three active DAC components and one DMA - direct memory access, shown on figure 3.9. This last component is crucial for this project because its functionality overcomes one limitation of the device. Before, the number of samples sent to the computer program was limited by the memory size of the microcontroller, since it was responsible for sampling, buffering data and transmitting it. This meant that the acquisition window was limited and the number of samples acquired was not be enough to feed the identification algorithm.

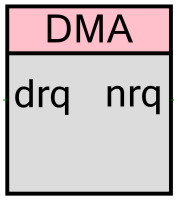

Figure 3.9: DMA component

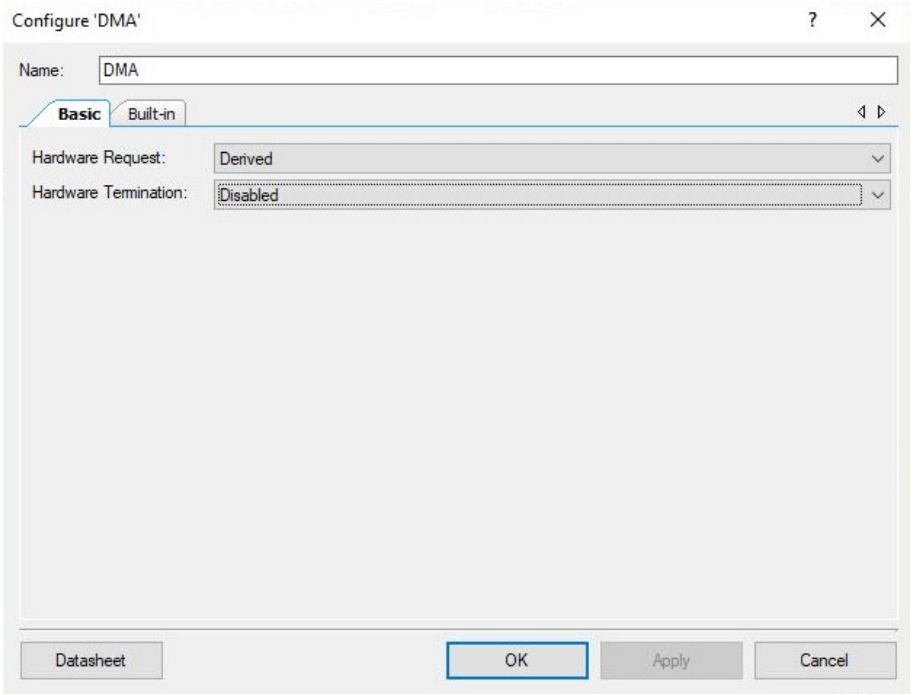

Figure 3.10: DMA component configuration

Thus, the DMA allows separation of the data acquisition process and its subsequent transmission by allocating two arrays, and while one is being used to store data from the sensor, another is being accessed to send data to the computer. This alternative executes until the desired number of samples is achieved. This iterative process is called *Ping-pong DMA* and is showcased on figure 3.11.

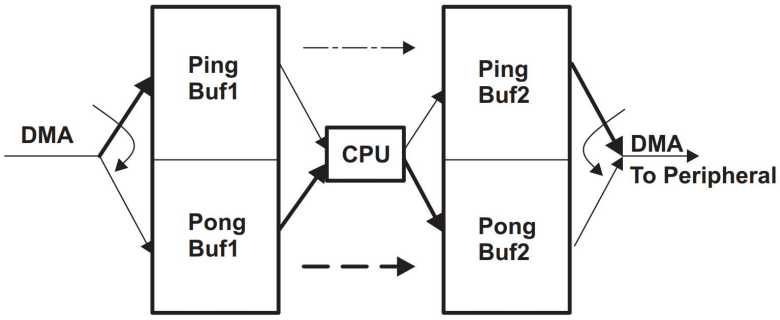

Figure 3.11: Ping Pong DMA schematic

Regarding the data transmission to a computer, this board has several options and the one chosen, mainly due to its speed and throughput is the Universal Serial Bus Full Speed, or USBFS, component. This way, after detecting and acquiring from the sensor, data is sent to a computer that has a program which supports the format sent by the board. Since the ADC was configured to acquire 2-byte samples and the USBFS component could only send values of 1-byte size, each sample acquired is divided in a little endian format and, when received on the computer there is a need convert each value back to 2-Byte format.

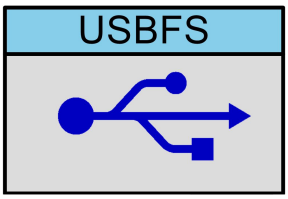

Figure 3.12: USBFS component

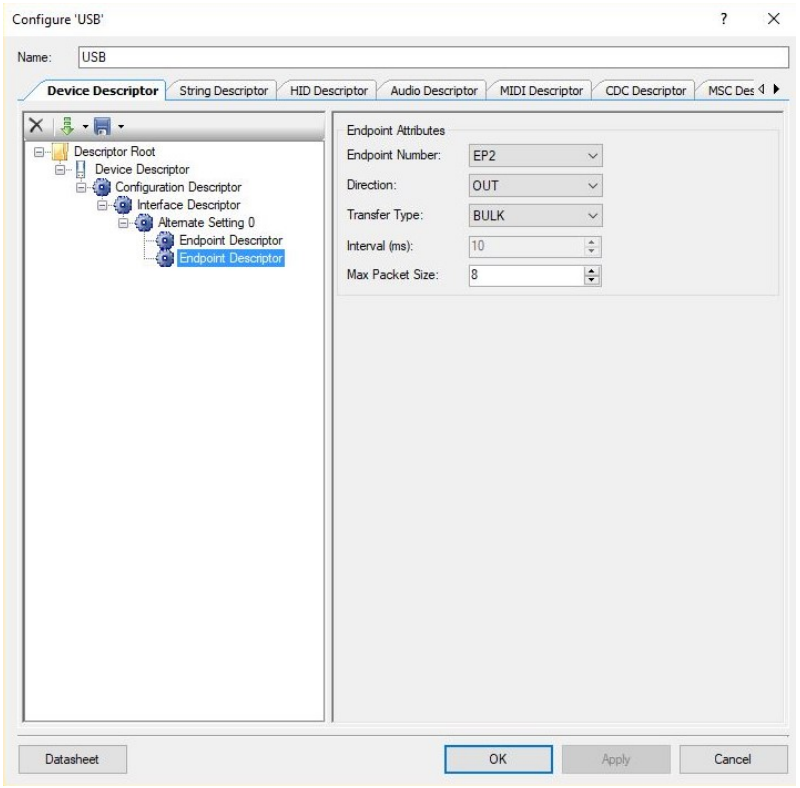

Figure 3.13: USBFS component configuration

This way, after configuring each aforementioned component and adding one auxiliary - Multiplexer, which gives the user options regarding the input signal -, the final circuit designed on the PSoC 5LP® to guarantee a successful data acquisition process is showcased on figure 3.14.

Ideally, the data acquisition process would start at the same time as the wave generator. This was not the case in the beginning since some issues with synchronization revealed a different challenge to tackle. This meant that, after one acquisition took place, the resulting signal would not start from 0. As a corrective measure, a new clock component was added to the circuit to guarantee that both DAC and ADC started their executions at the same time.

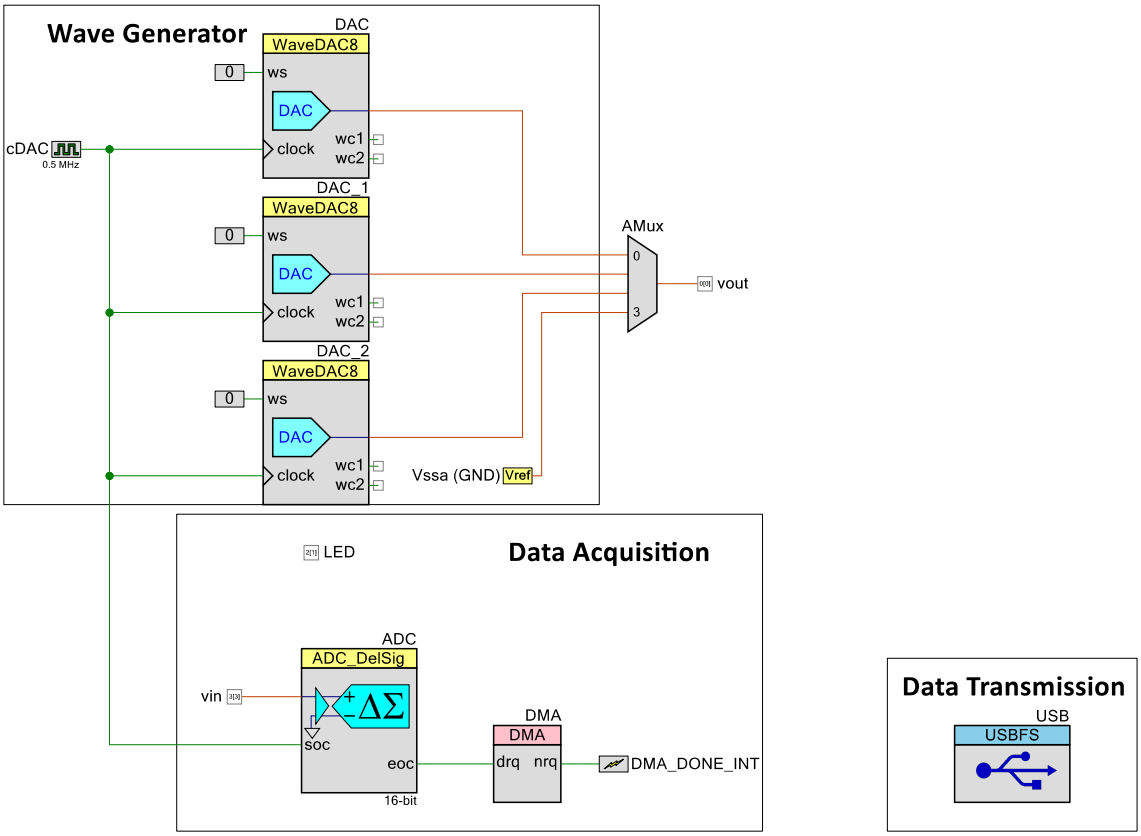

Figure 3.14: Complete data acquisition circuit

3.2 Data Pre-processing

Until this point of the project, every measurement performed required an Oscilloscope, which allowed visualizing the sensor response. Thus, it is necessary to also implement a computer program capable of receiving data sent by the microcontroller and display it afterwards. As mentioned before, the protocol chosen to send data from the PSoC 5LP® is USB. It allows a throughput of 64 bytes of data per transmission, where each sample is BYTE size.

Since each sample acquired by the ADC is a WORD, when sent to a computer, they were converted to Little Endian format. This meant that the software had to be able to receive the data and also convert two consecutive 1-byte long samples in their WORD original representation.

The programming language chosen to develop this segment was Python. This choice was mainly due to its extensive data processing libraries and also because it is open-source.This increases the value of the project since one of the goals is to reduce the cost of development of electronic nose and by using an open sourced language, there is no added cost to it. Also, there are many helpful guides and tutorials which would not exist if it was a closed platform.

In all its entirety, the program can be divided in the following operations:

- Data reception
- Data storage
- Data display
- Data comparison

Every bullet point written above represents one building block of the software developed to ensure the data acquisition procedure executed correctly. At first, the program output was showcased on a terminal, and user interaction was also through this method. It was neither practical nor aesthetically appealing. The solution adopted was the development of a graphical user interface on top of the program. It was implemented using the Qt for Python library which made it easier and more intuitive to use.

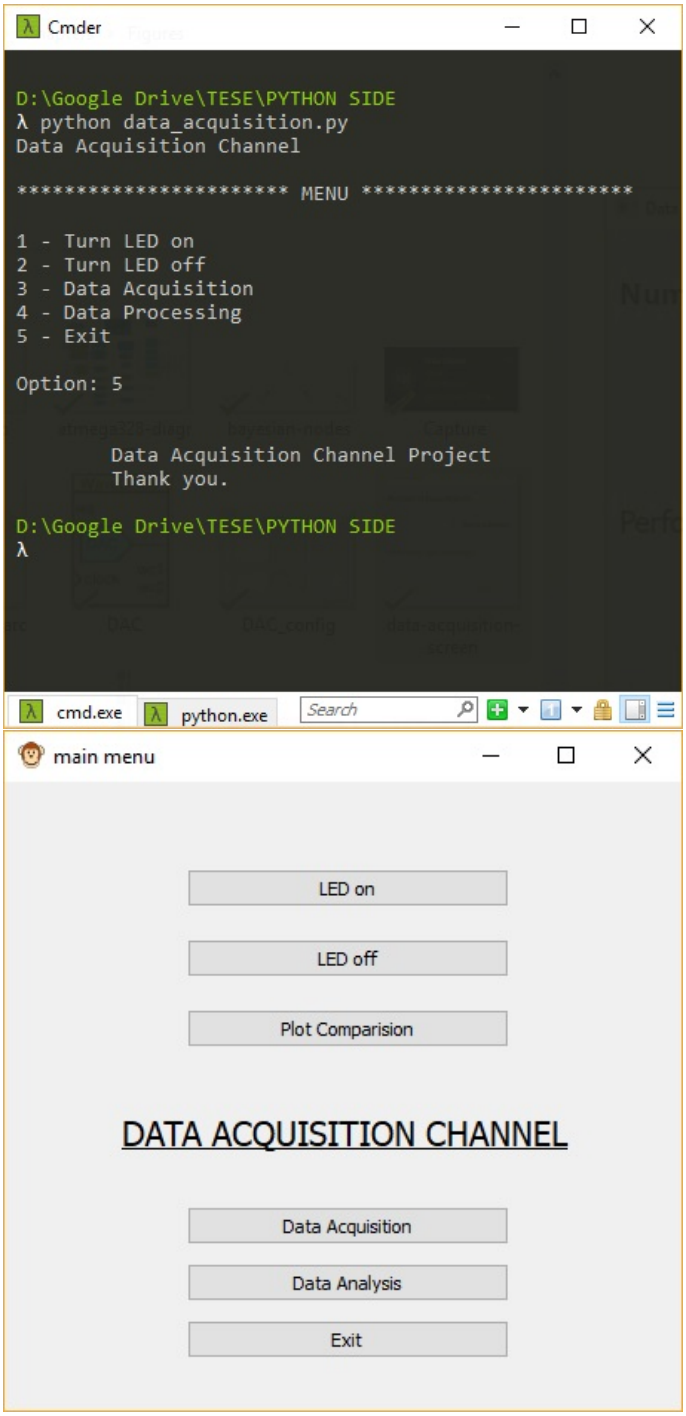

Figure 3.15: Application start screen: console and GUI

As a next step,the user is presented with a number of options. If there is a new sample that is going to be studied, by clicking the *Data Acquisition* button, the user is taken to a new window where the number of cycles and input signal (out of 3) can be chosen.

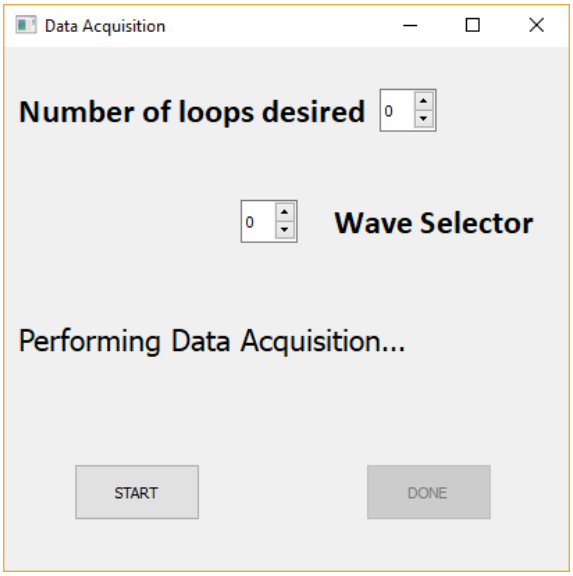

Figure 3.16: Data acquisition GUI window

After finishing the previous step, the user is given the option to either see the acquired sample or any other that might be on a database, composed of previous files saved after their own acquisition processes. Here is where a small degree of preprocessing is applied. At this point, it is desirable that every signal acquired has the same output as a rc filter.

Compared to the approach on [16], this thesis goes further. Instead of only obtaining the voltage output from the sensor, it also takes into account the frequency response to determine the sample origin. This characteristic is calculated from applying the FFT algorithm to the voltage output. This resulting curve is then object of further analysis as to obtain the equation responsible for its graphical representation and this is what will be used to classify each sample, their characteristic transfer function equations.

Since some acquisitions tend to start at an undesired point for this study, it was necessary the implementation of a correction algorithm that would only allow the curve that resembled the desired output to be stored in a file that, in the future, will be sent to the identification algorithm.

There's also the possibility to compare a maximum of 6 different sample plots. Even though in some cases, the differences might be very small for the human eye to notice, it seemed interesting to have this option where it's possible to have several curves in one plot.

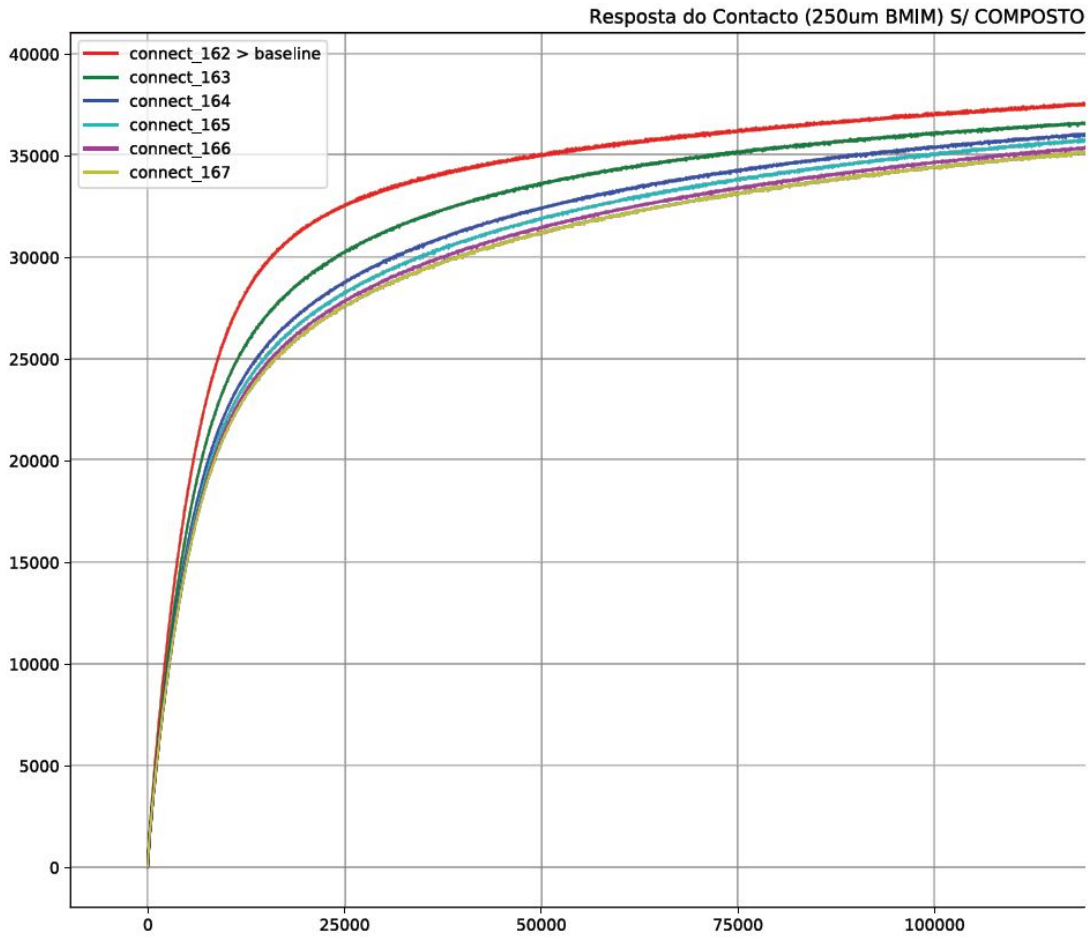

Figure 3.17: Comparing the output of 6 plots

All of the information gathered after each acquisition is saved on a CSV file that is accessed whenever the user wishes to display the results. This database stores the date and time of acquisition execution as well as the name given by the user.

1		Sample Name
2	D:/DEVELOPMENT/University Thesis/Data Processing/connect_1	24/05/2018 06:56
3	D:/DEVELOPMENT/University Thesis/Data Processing/connect_2	24/05/2018 06:56
4	D:/DEVELOPMENT/University Thesis/Data Processing/connect_4	24/05/2018 09:18
5	D:/DEVELOPMENT/University Thesis/Data Processing/connect_5	24/05/2018 09:20
6	D:/DEVELOPMENT/University Thesis/Data Processing/connect_6	24/05/2018 09:20
7	D:/DEVELOPMENT/University Thesis/Data Processing/connect_7	24/05/2018 09:48
8	D:/DEVELOPMENT/University Thesis/Data Processing/connect_8	24/05/2018 09:50
9	D:/DEVELOPMENT/University Thesis/Data Processing/connect_9	24/05/2018 09:51
10	D:/DEVELOPMENT/University Thesis/Data Processing/connect_10	24/05/2018 09:52
11	D:/DEVELOPMENT/University Thesis/Data Processing/connect_11	24/05/2018 09:53
12	D:/DEVELOPMENT/University Thesis/Data Processing/connect_12	24/05/2018 10:01
13	D:/DEVELOPMENT/University Thesis/Data Processing/connect_13	24/05/2018 10:05

Figure 3.18: E-nose system database

On figures 3.19 and 3.20 is presented the complete e-nose test bench:

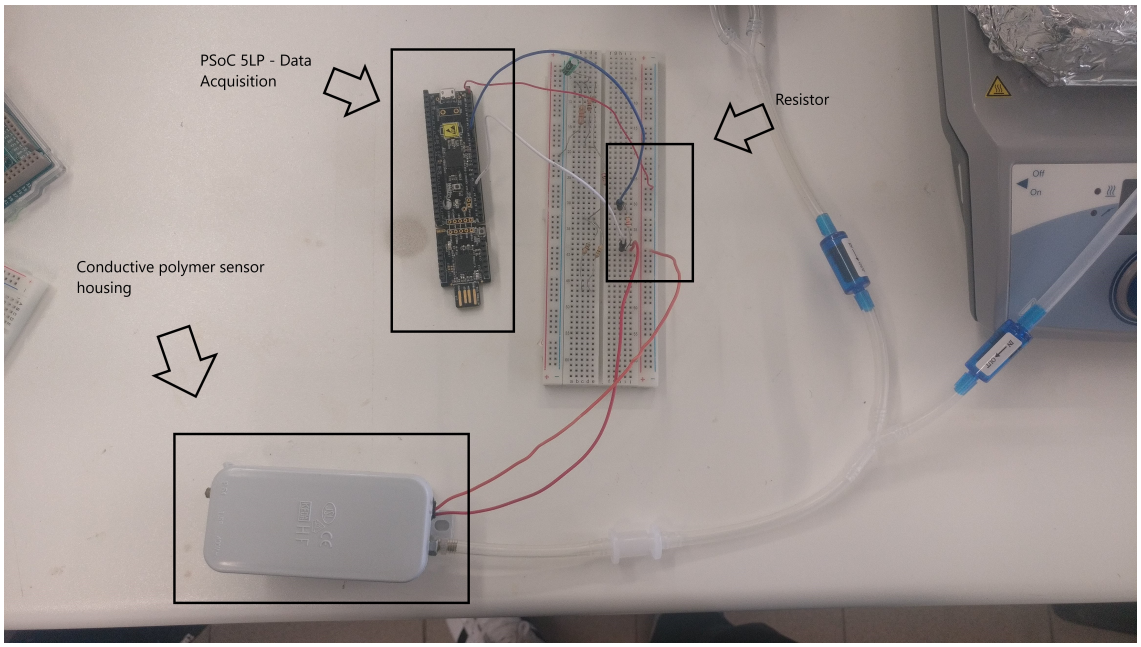

Figure 3.19: E-nose test bench

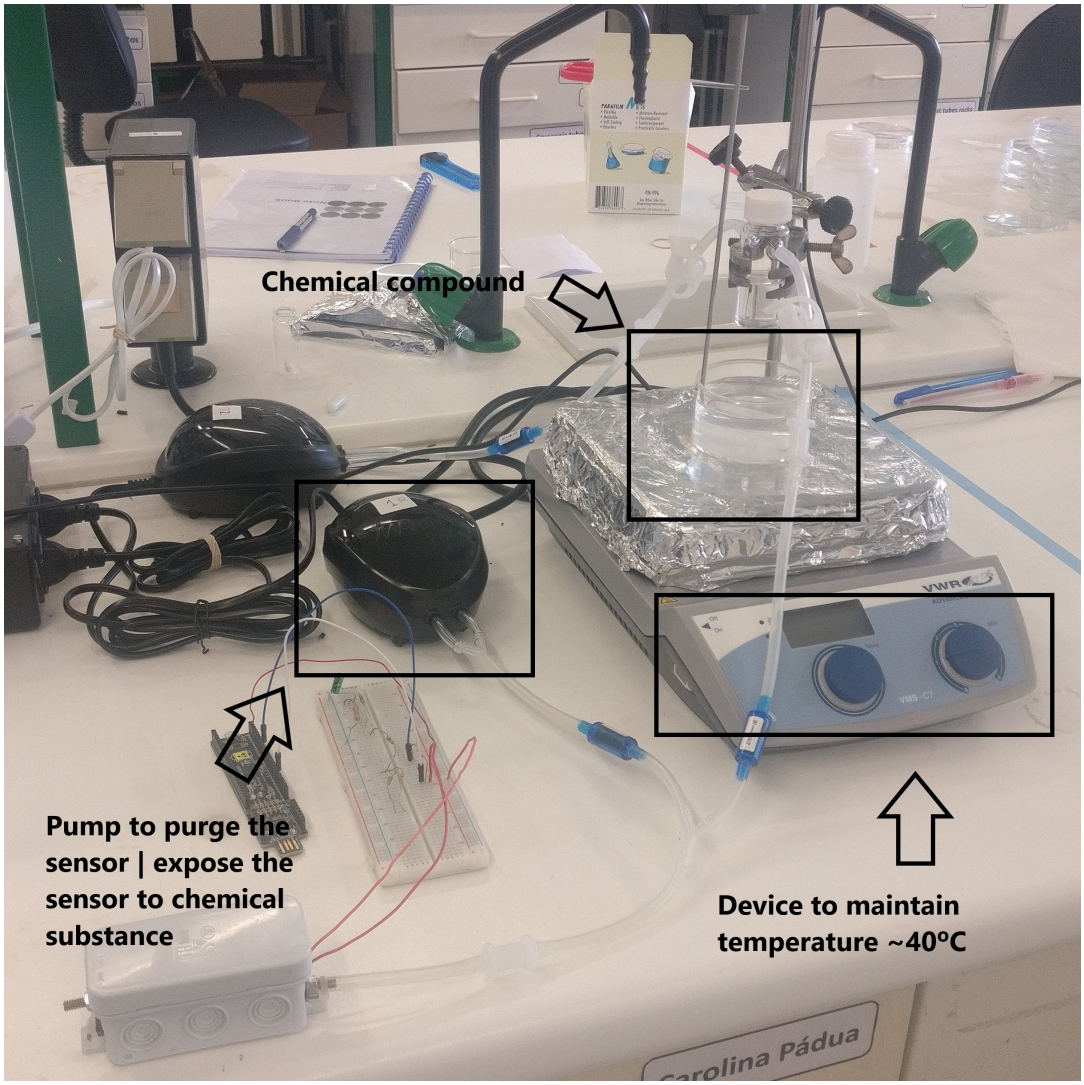

Figure 3.20: E-nose test bench

3.3 Transfer Function

The main goal of this thesis is the development of a data acquisition channel for an electronic nose. The implementation of this solution revolved around programming a microcontroller that would become the interface between the sensor signal and its digitalization for the computer to process it.

Since the expected response of the sensor, when a step signal is applied to it, is known beforehand to be similar to the equation of a RC circuit, its possible to say that each sample studied will have the transfer function showcased by equation 3.1

$$H(t) = (1 - e^{\frac{t}{\tau}}) \tag{3.1}$$

where, t is the temporal variable and τ is the unique characteristic associated with each response obtained, also called, the time constant.
Therefore, the last step of this project consists in finding the value of τ. The process to obtain its value is called system identification and it calculates the value of τ by figuring out what is the relationship between the input - step signal - and the output from the sensor.

To maintain some consistency, the chosen programming language to implement this segment was also Python and the desired tool to achieve it is the Python Control Systems Library.

3.4 Data acquisition channel flowchart

After the implementation of this project, it is possible to outline its working flow, when executed. This is showcased on figure 3.21.

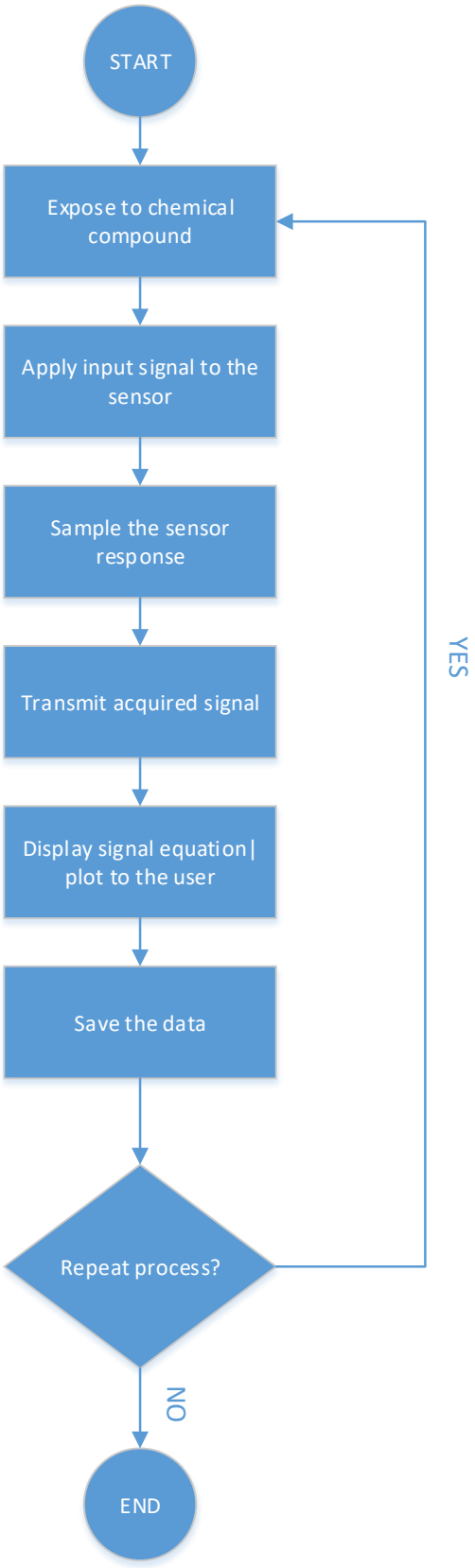

Figure 3.21: Flowchart of this e-nose project

Summary Chapter 3 revealed the tools and ideas for the data acquisition channel implementation. Based on the information gathered and written in chapter 2, the implementation of the data acquisition channel was divided into two segments - hardware and software. The former was programmed with the PSoC 5LP and its user-friendly interface, which guaranteed a successful acquisition of the sensor response and managed to send the samples to a computer, with some flexibility in user choices. The front-end of the project was written in Python and, at first, it was developed on the command line. After a guaranteed successful communication between PSoC 5LP and pc, the next step was the development of a GUI. On it, the user managed to access every task regarding data pre-processing that was implemented.

Chapter 4

Results

This section will present the results obtained after the implementation described on the previous chapter. To understand if the response was heavily influenced by the amplitude of the input signal, this sensor was tested with three different signals amplitude (1 V, 100 mV and 10 mV). Afterwards, the extraction of a transfer function from each resulting plot is discussed.

4.1 Plot results

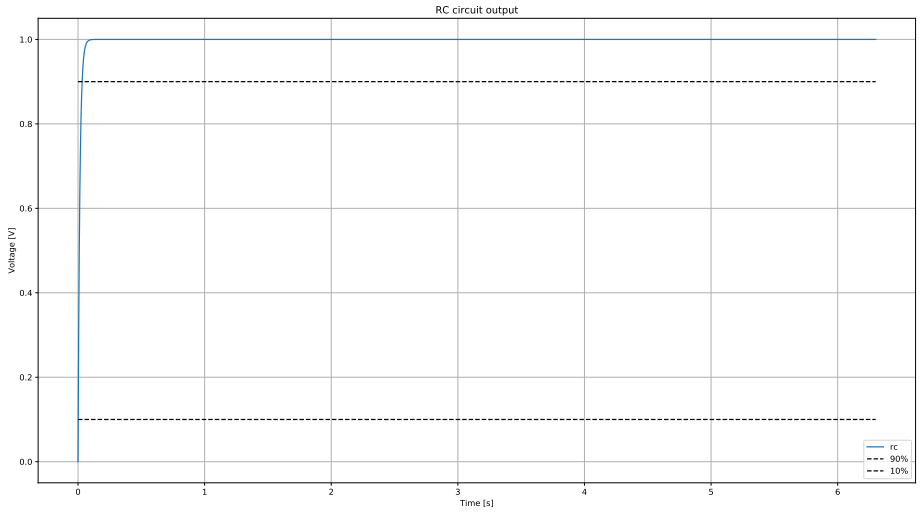

Figure 4.1: RC circuit Python simulation

This output is a representation of the ideal RC step response wave obtained with Python. The values of the resistors and capacitor were established identically as the values they have on the breadboard implementation.

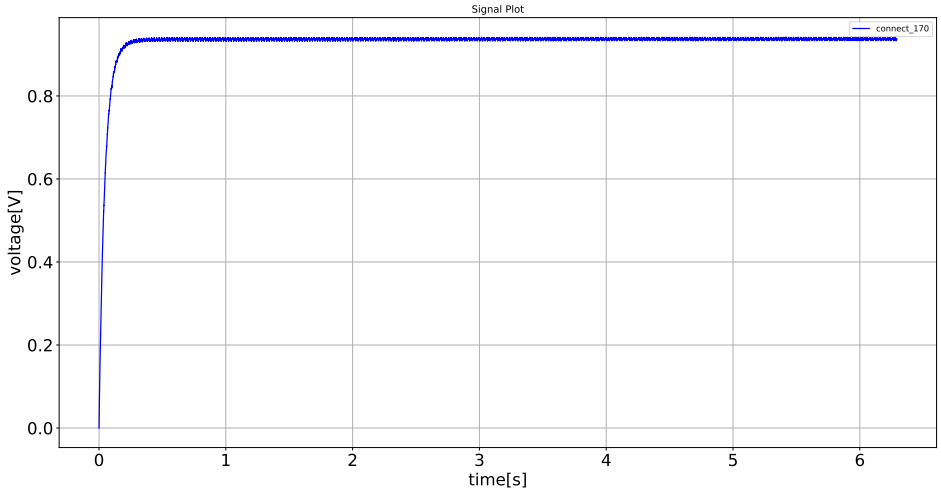

Figure 4.2: RC circuit breadboard output

On figure 4.2 it is possible to see the resulting output from the RC circuit after applying at its input a 1V dc signal. Comparing both figures - figure 4.1 (Python) and figure

4.2 (Breadboard), it can be said that the responses are very similar and, therefore, the data acquisition circuit is working as expected.

Both figures below are a representation of the sensor baseline response - not subject to any chemical compound - when a 1 V dc signal is applied to it. On figure 4.3, it is represented the voltage output and, as it was mentioned on the *state-of-the-art*, it is quite similar to the response of a rc circuit. After executing the FFT algorithm mentioned in the Implementation chapter, the obtained response is shown on figure 4.4.

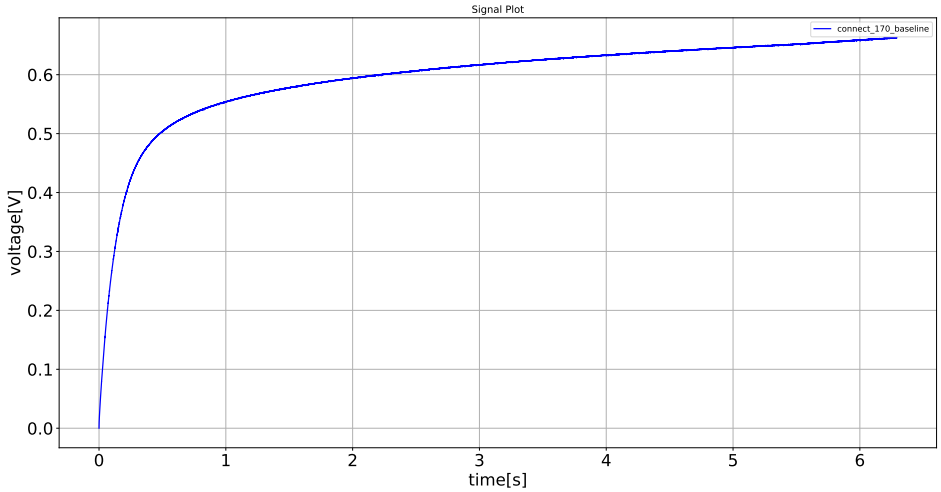

Figure 4.3: Sensor baseline response to a 1 V signal

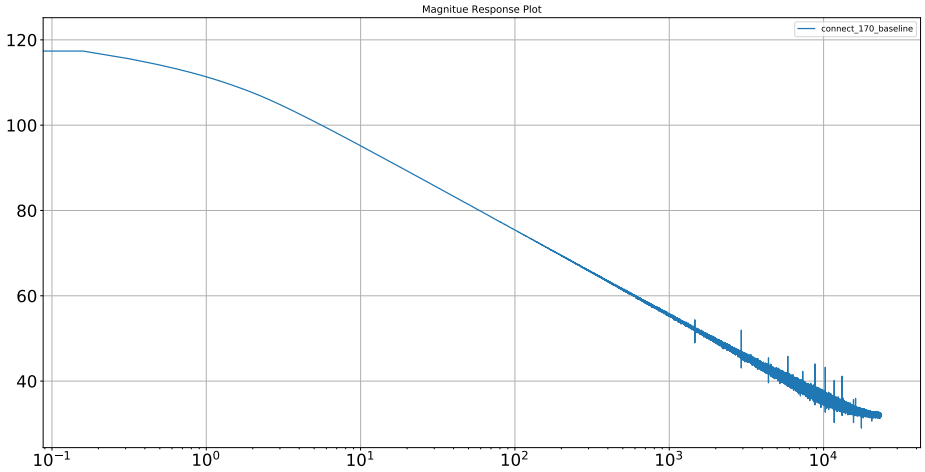

Figure 4.4: Sensor baseline frequency response

The following illustrations showcase the response of the sensor when it's properties have changed. This change is attributed to the chemical sample being tested as it alters the conductance of the Polymer, resulting in deviations on the curves. While on this state, three different signals were applied to it: 1 V, 100 mV and 10 mV. It is noticeable on the latter two that a heavy influence of noise is present because the signal amplitude is smaller.

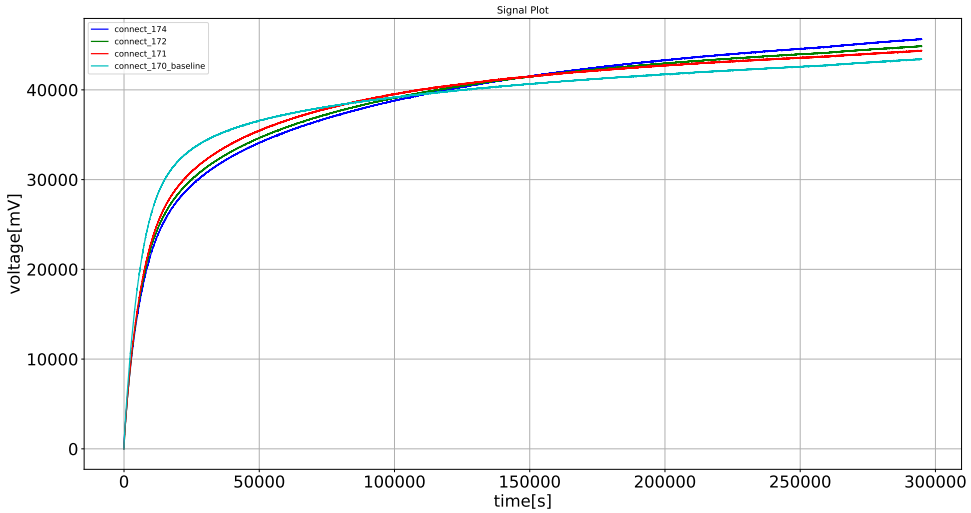

Figure 4.5: Sensor exposed to chemical compound response to a 1 V signal

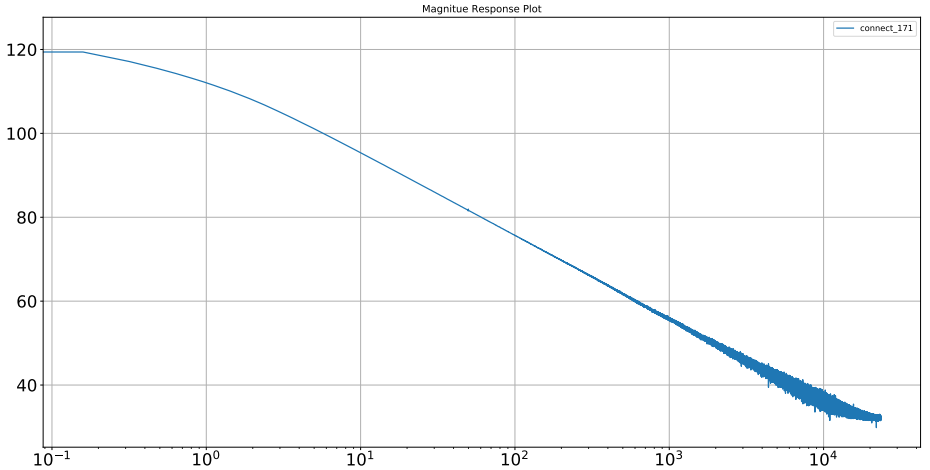

Figure 4.6: Sensor exposed to chemical compound frequency response

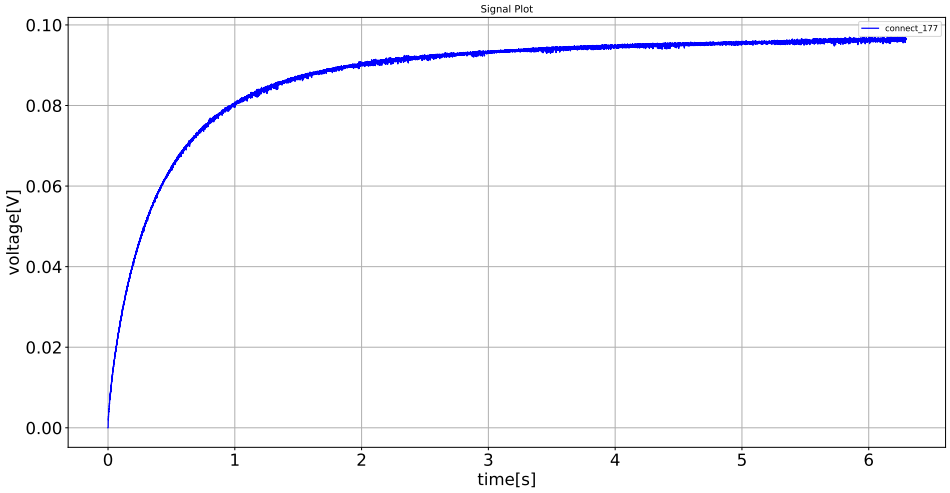

Figure 4.7: Sensor exposed to chemical compound response to a 100 mV signal

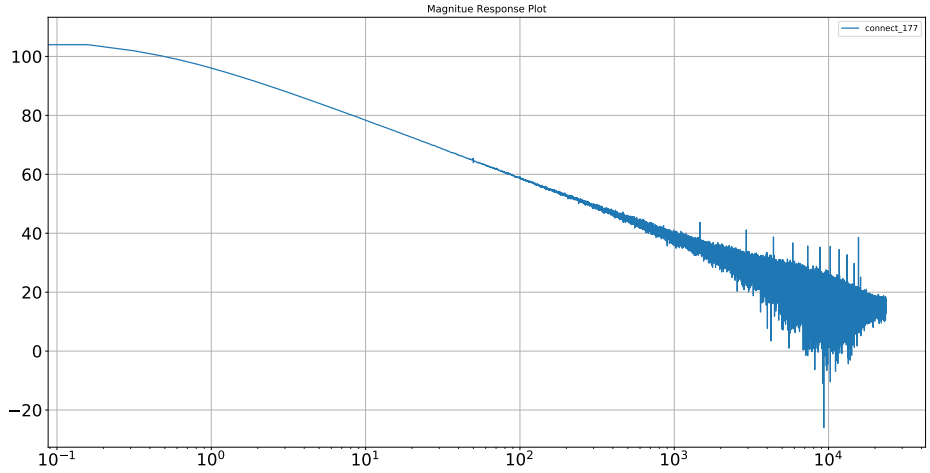

Figure 4.8: Sensor exposed to chemical compound frequency response

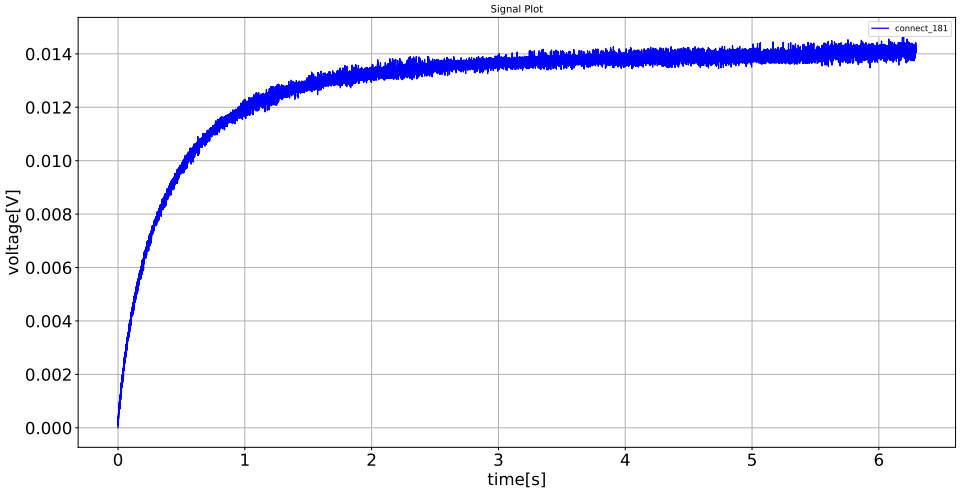

Figure 4.9: Sensor exposed to chemical compound response to a 10 mV signal

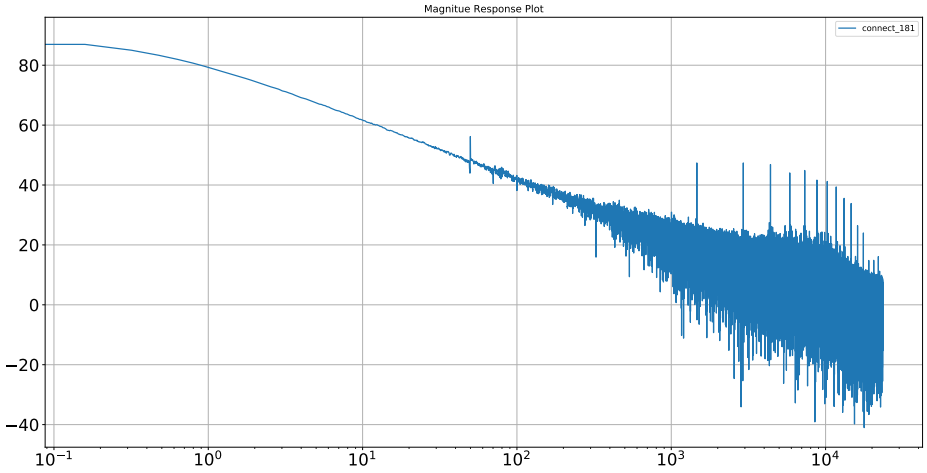

Figure 4.10: Sensor exposed to chemical compound frequency response

The goal of this application is to notice differences in sensor behaviour according to the chemical sample being tested. On figure 4.11, these deviations are clearly defined. While every curve, besides the baseline, should, ideally, be on top of each other, the deviation is not drastic. On the other hand, there is a clear distinction between the baseline curve and the others, which showcases that the data acquisition channel is capturing, successfully, the differences in sensor response.

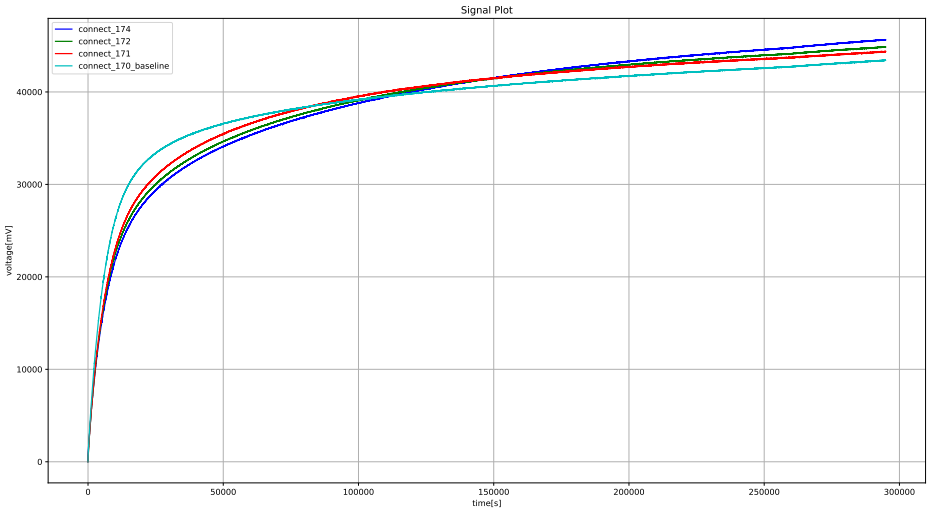

Figure 4.11: Sensor output responses compared

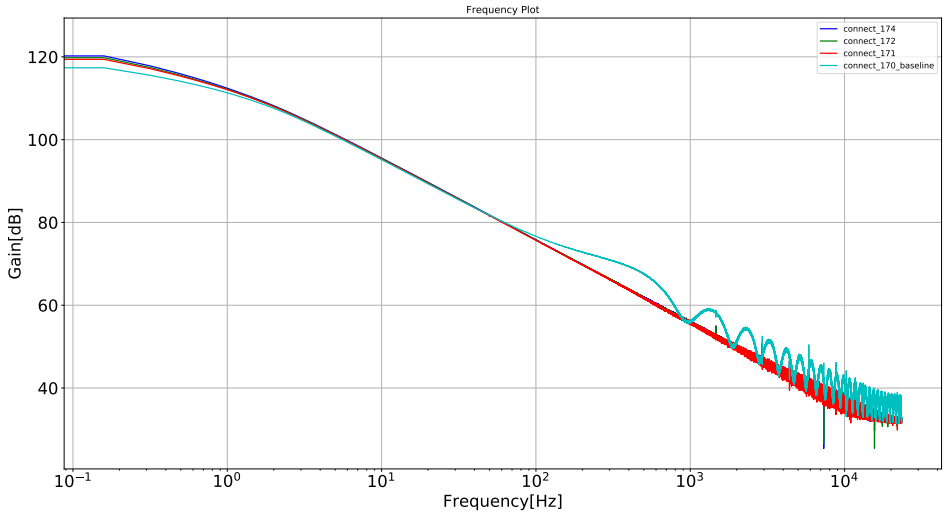

Figure 4.12: Sensor frequency responses compared

Chapter 5

Conclusion and Future Work

5.1 Final Remarks

This implementation of a data acquisition channel for an electronic nose is only possible due to the technology development verified on recent years in fields such as micro controllers, sensors and data processing libraries available for open source programming languages, such as Python. The structure of it is divided in three main blocks: sensor array, electronic board and computer software. The sensor array suffers a deviation from its normal response to incoming signals based on whether it is being exposed to a fluid or not. After it, the electronic board is capable of sampling this incoming signal and send it to another device with software capable of pre-processing and storing the data. Based on the requirements initially stated for the development of this dissertation, it can be considered successful. Data acquisition and subsequent transmission was achieved with a degree of flexibility that allows further developments on top of this implementation possible. Regarding the final step of identifying between different gases, it was was not possible to achieve this goal.

5.2 Future Developments

Due to the increasingly fast paced rate of technology development, it is possible to keep improving this e-nose implementation. I propose a study of a possible integration of another microcontroller that could communicate via wireless to a smartphone, since these have the necessary computing power to perform the identification process. It would increase the portability factor. Also, as possible future improvements, it could be implemented a machine learning algorithm that could, based on previous data, calculate the optimal frequency value for the ADC.

Bibliography

[1] K. Arshak, E. Moore, G. M. Lyons, J. Harris, and S. Clifford. "A review of gas sensors employed in electronic nose applications." In: *Sensor Review* 24.2 (2004), pp. 181–198. ISSN: 02602288. DOI: 10.1108/02602280410525977. URL: www.emeraldinsight.com/0260-2288.htm.

[2] H. Debéda, D. Rebière, J. Pistré, and F. Ménill. "Thick film pellistor array with a neural network post-treatment." In: *Sensors and Actuators: B. Chemical* 27.1-3 (1995), pp. 297–300. ISSN: 09254005. DOI: 10.1016/0925-4005(94)01605-H.

[3] C. Di Natale, D. Salimbeni, R. Paolesse, A. Macagnano, and A. D'Amico. "Porphyrins-based opto-electronic nose for volatile compounds detection." In: *Sensors and Actuators, B: Chemical* 65.1 (2000), pp. 220–226. ISSN: 09254005. DOI: 10.1016/S0925-4005(99)00316-0.

[4] S. Esfahani and J. A. Covington. "Low Cost Optical Electronic Nose for Biomedical Applications." In: *Proceedings* 1.4 (2017), p. 589. ISSN: 2504-3900. DOI: 10.3390/proceedings1040589. URL: http://www.mdpi.com/2504-3900/1/4/589.

[5] J. W. Gardner, H. W. Shin, and E. L. Hines. "Electronic nose system to diagnose illness." In: *Sensors and Actuators, B: Chemical* 70.1-3 (2000), pp. 19–24. ISSN: 09254005. DOI: 10.1016/S0925-4005(00)00548-7.

[6] Z. Haddi, A. Amari, H. Alami, N. El Bari, E. Llobet, and B. Bouchikhi. "A portable electronic nose system for the identification of cannabis-based drugs." In: *Sensors and Actuators, B: Chemical* 155.2 (2011), pp. 456–463. ISSN: 09254005. DOI: 10.1016/j.snb.2010.12.047. URL: http://dx.doi.org/10.1016/j.snb.2010.12.047.

[7] G. Harsányi. "Polymer films in sensor applications: a review of present uses and future possibilities." In: *Sensor Review* 20.2 (2000), pp. 98–105. ISSN: 0260-2288. DOI: 10.1108/02602280010319169. URL: https://www.emeraldinsight.com/doi/10.1108/02602280010319169.

[8] D. James, S. M. Scott, Z. Ali, and W. T. O'Hare. "Chemical sensors for electronic nose systems." In: *Microchimica Acta* 149.1-2 (2005), pp. 1–17. ISSN: 00263672. DOI: 10.1007/s00604-004-0291-6.

[9] X. Jiang, P. Jia, R. Luo, B. Deng, S. Duan, and J. Yan. "A novel electronic nose learning technique based on active learning: EQBC-RBFNN." In: *Sensors and Actuators, B: Chemical* 249.May (2017), pp. 533–541. ISSN: 09254005. DOI: 10.1016/j.snb.2017.04.072. URL: http://dx.doi.org/10.1016/j.snb.2017.04.072.

[10] T.-H. Le, Y. Kim, and H. Yoon. "Electrical and Electrochemical Properties of Conducting Polymers." In: *Polymers* 9.4 (2017), p. 150. ISSN: 2073-4360. DOI: 10.3390/polym9040150. URL: http://www.mdpi.com/2073-4360/9/4/150.

[11] M. Macias Macias, J. E. Agudo, A. Garcia Manso, C. J. avier Garcia Orellana, H. M. anuel Gonzalez Velasco, and R. Gallardo Caballero. "A compact and low cost electronic nose for aroma detection." In: *Sensors (Basel, Switzerland)* 13.5 (2013), pp. 5528–5541. ISSN: 14248220. DOI: 10.3390/s130505528.

[12] A. Manbachi and R. S. Cobbold. "Development and application of piezoelectric materials for ultrasound generation and detection." In: *Ultrasound* 19.4 (2011), pp. 187–196. ISSN: 1742271X. DOI: 10.1258/ult.2011.011027. arXiv: arXiv:1011.1669v3.

[13] S. Panigrahi, S. Balasubramanian, H. Gu, C. Logue, and M. Marchello. "Neural-network-integrated electronic nose system for identification of spoiled beef." In: *LWT - Food Science and Technology* 39.2 (2006), pp. 135–145. ISSN: 00236438. DOI: 10.1016/j.lwt.2005.01.002.

[14] T. Pearce, S. Schiffman, H. Nagle, and J. Gardner, eds. *Handbook of Machine Olfaction: Electronic Nose Technology*. Wiley-VCH, 2002, p. 633. ISBN: 047186109X.

[15] N. A. Rakow and K. S. Suslick. "A colorimetric sensor array\nfor odour visualization." In: *Nature* 406.August (2000), pp. 710–713. URL: http://www.nature.com/nature/journal/v406/n6797/pdf/406710a0.pdf.

[16] R. T. da Rocha, I. G. R. Gutz, and C. L. do Lago. "A Low-Cost and High-Performance Conductivity Meter." In: *Journal of Chemical Education* 74.5 (1997), p. 572. ISSN: 0021-9584. DOI: 10.1021/ed074p572. URL: http://pubs.acs.org/doi/abs/10.1021/ed074p572.

[17] F. Röck, N. Barsan, and U. Weimar. "Electronic nose: Current status and future trends." In: *Chemical Reviews* 108.2 (2008), pp. 705–725. ISSN: 00092665. DOI: 10.1021/cr068121q. URL: http://pubs3.acs.org/acs/journals/doilookup?in{_}doi=10.1021/cr068121q{\%}5Cnhttp://pubs.acs.org/doi/pdf/10.1021/cr068121q.

[18] M. S. Freund and N. S. Lewis. "A chemically diverse conducting polymer-based "electronic nose"." In: *Proceedings of the National Academy of Sciences of the United States of America* 92.7 (1995), pp. 2652–2656. ISSN: 0027-8424. DOI: 10.1073/pnas.92.7.2652. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=42276{\&}tool=pmcentrez{\&}rendertype=abstract.

[19] S. Saevels, J. Lammertyn, A. Z. Berna, E. A. Veraverbeke, C. Di Natale, and B. M. Nicolaï. "An electronic nose and a mass spectrometry-based electronic nose for assessing apple quality during shelf life." In: *Postharvest Biology and Technology* 31.1 (2004), pp. 9–19. ISSN: 09255214. DOI: 10.1016/S0925-5214(03)00129-7.

[20] STMicroelectronics. "Application Note: Understanding and minimising ADC conversion errors." In: (2003), pp. 1–42. URL: http://www.st.com/web/en/resource/technical/document/application{_}note/CD00004444.pdf.

[21] M. Stuiver. "Biophysics of the sense of smell." In: *Excelsior* (1958), p. 99.

[22] K.-T. Tang, S.-W. Chiu, C.-H. Pan, H.-Y. Hsieh, Y.-S. Liang, and S.-C. Liu. "Development of a Portable Electronic Nose System for the Detection and Classification of Fruity Odors." In: *Sensors* 10.10 (2010), pp. 9179–9193. ISSN: 1424-8220. DOI: 10.3390/s101009179. URL: http://www.mdpi.com/1424-8220/10/10/9179/.

[23] M. Titterington. "Neural networks." In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.1 (2010), pp. 1–8. ISSN: 19395108. DOI: 10.1002/wics.50. arXiv: arXiv:1411.3159v1.

[24] a.D Wilson and M. Baietto. "Applications and advances in electronic-nose technologies." In: *Sensors* 9.7 (2009), pp. 5099–5148. ISSN: 1424-8220. DOI: 10.3390/s90705099. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3274163{\&}tool=pmcentrez{\&}rendertype=abstract.

[25] J. S. Wilson, ed. *Sensor Technology Handbook*. 2005, p. 691. ISBN: 0750677295. DOI: 10.1016/B978-075067729-5/50054-9. URL: http://books.google.com/books?id=hPPM8G4kI0wC{\&}pgis=1.

[26] L. Zhang, F. Tian, and G. Pei. "A novel sensor selection using pattern recognition in electronic nose." In: *Measurement* 54 (2014), pp. 31–39. ISSN: 02632241. DOI: 10.1016/j.measurement.2014.04.005. URL: http://www.sciencedirect.com/science/article/pii/S0263224114001559.

Appendix A

Microcontroller Configuration Code

```
#include "project.h"

/* Defines for DMA */
#define DMA_BYTES_PER_BURST 2
#define DMA_REQUEST_PER_BURST 1
#define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_DST_BASE (CYDEV_SRAM_BASE)


/*Constants that store the memory size for the number of samples acquired*/
enum{
    U16_BUFFER_SIZE = 32,
    U8_BUFFER_SIZE  = 2*U16_BUFFER_SIZE,
};
/*Memory Address for storing acquired samples*/
union Buffer{
    uint16 buffer16[U16_BUFFER_SIZE];
    uint8 buffer8[U8_BUFFER_SIZE];
}buffer_m, buffer_t;

/* Variable declarations for DMA and other variables */
uint8 DMA_Chan;
uint8 DMA_TD[2];
uint8 input[1] = {};
uint8 ws = 1, ciclos = 0;
uint8 handover_flag = 0, buffer_flag = 1;
int cnt = 0, wave_selector = 1, counter =0 ;
/* pointer to transmission buffer */
uint8 * pBuffer = &buffer_m.buffer8[0];


```

```
32 /*Initialization of DMA prototype*/
33 void config_dma();
34 /*Initialization of COMPONENTS prototype*/
35 void init_components();
36 /*Read input from the user*/
37 void read_usb_input();
38 /*transfer the acquired samples to the PC*/
39 void data_transfer();
40 /* transfer the acquired samples to the PC */
41 void data_transfer_continuous();
42 /*select the wave to output*/
43 void wave_select();
44 /*select the buffer to transmit*/
45 void buffer_pointer();
46 /*reset components for following data transfers and acquisitions*/
47 void reset_components(int param);
48 /* Stop components */
49 void stop_components();
50 /*clear the buffers after each transmission*/
51 void clear_buffer();
52
53
54 int main(){
55     /* DMA channel initialization */
56     config_dma();
57
58     CyGlobalIntEnable;
59
60     /* USBUART initialization */
61     USB_Start(0, USB_5V_OPERATION); //OUT EP: 2; IN EP: 1
62     while(!USB_bGetConfiguration()){}
63     USB_EnableOutEP(2);
64
65     /* Startup the basic components*/
66     init_components();
67
68     /* The ADC Interrupt is disabled because
69     the system doesn't need the interrupt */
70     ADC_IRQ_Disable();
71
72     for(;;){
73         read_usb_input();
74         switch(input[0]){
75             case '1': LED_Write(1);USB_EnableOutEP(2); break;
76             case '2': LED_Write(0);USB_EnableOutEP(2); break;
77             case '3': {
78                             read_usb_input();
79                             ciclos = input[0];
80                             ciclos = ciclos - 48;
81                             LED_Write(1);
```

```
82                              while(counter<ciclos){
83                                          data_transfer();
84                                          USB_EnableOutEP(2);
85                                          clear_buffer();
86                                          counter++;
87                              }
88                              LED_Write(0);
89                              counter = 0;
90                              break;
91              }
92              default:  USB_EnableOutEP(2); break;
93          }
94      }
95  }
96
97  void config_dma(){
98      /* DMA Configuration for DMA */
99      DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
100             HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));
101     DMA_TD[0] = CyDmaTdAllocate();
102     DMA_TD[1] = CyDmaTdAllocate();
103     CyDmaTdSetConfiguration(DMA_TD[0], 64, DMA_TD[1],
104             DMA__TD_TERMOUT_EN | CY_DMA_TD_INC_DST_ADR);
105     CyDmaTdSetConfiguration(DMA_TD[1], 64, DMA_TD[0],
106             DMA__TD_TERMOUT_EN | CY_DMA_TD_INC_DST_ADR);
107     CyDmaTdSetAddress(DMA_TD[0], LO16((uint32)ADC_DEC_SAMP_PTR),
108         LO16((uint32)&buffer_m.buffer16[0]));
109     CyDmaTdSetAddress(DMA_TD[1], LO16((uint32)ADC_DEC_SAMP_PTR),
110         LO16((uint32)&buffer_t.buffer16[0]));
111     CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);
112     CyDmaChEnable(DMA_Chan, 1);
113 }
114
115 void init_components(){
116     /* Start the Multiplexer */
117     AMux_Start();
118     /*Start Multiplexer output at 0V*/
119     AMux_Select(3);
120     /* Start ADC */
121     ADC_Start();
122     /* Start isr connected to DMA nrq signal */
123     DMA_DONE_INT_Start();
124 }
125
126 void read_usb_input(){
127     while(USB_GetEPState(2) == USB_OUT_BUFFER_EMPTY){}
128     USB_ReadOutEP(2, input, 1);
129     USB_EnableOutEP(2);
130 }
131
```

```
132 void buffer_pointer(){
133     if(buffer_flag == 1){
134         pBuffer = &buffer_m.buffer8[0];
135         buffer_flag = 2;
136     }
137     else{
138         pBuffer = &buffer_t.buffer8[0];
139         buffer_flag = 1;
140       }
141 }
142 
143 void data_transfer(){
144     read_usb_input();
145     ws = input[0];
146     ws = ws - 48;
147     wave_select(ws);
148 
149     for(;;){
150         if(handover_flag){
151             handover_flag = 0;
152             buffer_pointer();
153             while(USB_GetEPState(1) != USB_IN_BUFFER_EMPTY){}
154             USB_LoadInEP(1, pBuffer, 64);
155             cnt++;
156             if(cnt == 9216){
157                 cnt = 0;
158                 stop_components();
159                 break;
160              }
161         }
162     }
163 }
164 
165 void wave_select(int param){
166     CyDmaChEnable(DMA_Chan, 1);
167     if(param == 1){
168         AMux_Select(0);
169         DAC_Start();
170     }
171     else if(param == 2){
172         AMux_Select(1);
173         DAC_1_Start();
174     }
175     else if(param == 3){
176         AMux_Select(2);
177         DAC_2_Start();
178     }
179     else
180         return;
181 
```

```
182     cDAC_Start();
183 }
184 
185 void reset_components(int param){
186     CyDmaChDisable(DMA_Chan);
187     CyDmaChEnable(DMA_Chan, 1);
188     ADC_StopConvert();
189     cDAC_StopBlock();
190     LED_Write(0);
191     wave_select(param);
192 }
193 
194 void stop_components(){
195     cDAC_StopBlock();
196     AMux_Select(3);
197     DAC_Stop();
198     DAC_1_Stop();
199     DAC_2_Stop();
200     ADC_StopConvert();
201 
202     *pBuffer = 0;
203     CyDmaChDisable(DMA_Chan);
204 }
205 
206 void clear_buffer(){
207     for (uint8 i = 0; i < U16_BUFFER_SIZE; ++i){
208         buffer_m.buffer16[i] = 0;
209         buffer_t.buffer16[i] = 0;
210     }
211 }
212 /* [] END OF FILE */
```

Appendix B

USB configuration code

```
1  import os
2  import usb.core
3  import usb.util
4  import sys
5  import numpy as np
6
7
8  class USBDevice:
9
10     def __init__(self, *args, **kwargs):
11     self.filename = "connect_0"
12     self.OUT = None
13     self.IN = None
14     self.connect_usb()
15
16     def connect_usb(self):
17         """
18         Function responsible for assigning the IN and OUT endpoints
19         :return:
20         """
21         # search for our device by product and vendor ID
22         dev = usb.core.find(idVendor=0x4B4, idProduct=0xF232)
23
24         # raise error if device is not found
25         if dev is None:
26             raise ValueError('Device not found')
27
28         # set the active configuration (basically, start the device)
29         dev.set_configuration()
30
31         # get interface 0, alternate setting 0
```

```
32          intf = dev.get_active_configuration()[(0, 0)]
33
34          # find the first (and in this case only)
35          # OUT endpoint in our interface
36              epOut = usb.util.find_descriptor(
37              intf,
38              custom_match= \
39              lambda e: \
40              usb.util.endpoint_direction(e.bEndpointAddress) == \
41              usb.util.ENDPOINT_OUT)
42
43          # find the first (and in this case only)
44          #IN endpoint in our interface
45              epIn = usb.util.find_descriptor(
46              intf,
47              custom_match= \
48              lambda e: \
49              usb.util.endpoint_direction(e.bEndpointAddress) == \
50              usb.util.ENDPOINT_IN)
51
52          # make sure our endpoints were found
53          assert epOut is not None
54          assert epIn is not None
55
56          self.OUT= epOut
57          self.IN = epIn
58
59      def close_device(self, exitval=0):
60          """
61          Function responsible for disconnecting
62          the device from the Python interface
63          :param exitval
64          :return:
65          """
66          print("Exiting...")
67          usb.util.dispose_resources
68          sys.exit(exitval)
69
70      def save_usb_data(self, sample_array):
71          """
72          Function responsible for saving the transferred data to a text file
73          :return: bool value confirming whether it saved successfully or not
74          """
75          self.filename = self.check_file_path()
76          try:
77              f = open(self.filename, 'w')
78          except:
79              print("Can't␣open␣file")
80          self.filename = input("Write␣the␣desired␣file␣name:␣")
81          f = open(self.filename, 'w')
```

```
82
83      for element in sample_array:
84          f.write(str(element) + "\n")
85          f.close()
86          print("Data Saved on file %s" %self.filename)
87
88      def transfer_data(self, loop_count, wave_selector):
89          """
90          Function responsible for the data transfer between the PSoC and the PC
91          :param loop_count: option to choose
92          how many data transfer are desired
93          :param wave_selector: option to choose
94          which wave output is desired from the PSoC (1V, 100mV, 10mV)
95          :return: array filled with the acquired samples from the PSoC
96          """
97          for i in range(938*loop_count):
98              try:
99                  data = self.IN.read(64)
100             except:
101                 print("Can't receive data from the PSoC")
102         self.close_device()
103         if i == 0:
104             aux = data
105         else:
106             aux = np.append(aux, data)
107
108         self.save_usb_data(data)
109
110     def check_file_path(self):
111         """
112         :return: new file name
113         """
114         current_loop = 0
115         while os.path.exists("connect_%s" % current_loop):
116             current_loop += 1
117
118         self.filename = ("connect_%s" % current_loop)
119
120         return self.filename
121
122
123 def main():
124     usb = USBDevice()
125
126
127 if __name__ == "__main__":
128     main()
```

Appendix C

Start Screen of the Application code

```
from PyQt4 import QtCore, QtGui
from menu import Ui_mainMenu
import sys


try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8

    def _translate(context, text, disambig):
        return QtGui.QApplication.\
            translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.\
            translate(context, text, disambig)


class Ui_mainWindow(object):
    def __init__(self, object):
        self.setupUi(object)

    def setupUi(self, mainWindow):
        mainWindow.setObjectName(_fromUtf8("mainWindow"))
        mainWindow.resize(798, 617)

```

```
32          icon = QtGui.QIcon()
33          icon.addPixmap(QtGui.QPixmap(_fromUtf8("monkey_on_32x32.png")),\
34               QtGui.QIcon.Normal, QtGui.QIcon.Off)
35
36          mainWindow.setWindowIcon(icon)
37          mainWindow.setStatusTip(_fromUtf8(""))
38
39          self.centralwidget = QtGui.QWidget(mainWindow)
40          self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
41
42          self.verticalLayoutWidget = QtGui.QWidget(self.centralwidget)
43          self.verticalLayoutWidget.setGeometry(QtCore.QRect(130, 50, 581, 221))
44          self.verticalLayoutWidget.setObjectName(\
45              _fromUtf8("verticalLayoutWidget"))
46
47          self.verticalLayout = QtGui.QVBoxLayout(self.verticalLayoutWidget)
48          self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
49
50          self.label = QtGui.QLabel(self.verticalLayoutWidget)
51
52          font = QtGui.QFont()
53          font.setPointSize(24)
54
55          self.label.setFont(font)
56          self.label.setAlignment(QtCore.Qt.AlignCenter)
57          self.label.setWordWrap(True)
58          self.label.setObjectName(_fromUtf8("label"))
59
60          self.verticalLayout.addWidget(self.label)
61
62          self.startButton = QtGui.QPushButton(self.centralwidget)
63          self.startButton.setGeometry(QtCore.QRect(290, 390, 261, 91))
64          self.startButton.setIconSize(QtCore.QSize(32, 16))
65          self.startButton.setObjectName(_fromUtf8("startButton"))
66
67          self.comboBox = QtGui.QComboBox(self.centralwidget)
68          self.comboBox.setGeometry(QtCore.QRect(710, 540, 69, 22))
69          self.comboBox.setObjectName(_fromUtf8("comboBox"))
70          self.comboBox.addItem(_fromUtf8(""))
71          self.comboBox.addItem(_fromUtf8(""))
72          self.comboBox.addItem(_fromUtf8(""))
73          self.comboBox.addItem(_fromUtf8(""))
74          self.comboBox.addItem(_fromUtf8(""))
75          self.comboBox.addItem(_fromUtf8(""))
76          self.comboBox.addItem(_fromUtf8(""))
77
78          mainWindow.setCentralWidget(self.centralwidget)
79
80          self.statusbar = QtGui.QStatusBar(mainWindow)
81          self.statusbar.setObjectName(_fromUtf8("statusbar"))
```

```
82
83          mainWindow.setStatusBar(self.statusbar)
84          self.menubar = QtGui.QMenuBar(mainWindow)
85          self.menubar.setGeometry(QtCore.QRect(0, 0, 798, 21))
86          self.menubar.setObjectName(_fromUtf8("menubar"))
87
88          self.menuFile = QtGui.QMenu(self.menubar)
89          self.menuFile.setObjectName(_fromUtf8("menuFile"))
90
91          self.menuHelp = QtGui.QMenu(self.menubar)
92          self.menuHelp.setObjectName(_fromUtf8("menuHelp"))
93
94          mainWindow.setMenuBar(self.menubar)
95
96          self.actionOpen_File = QtGui.QAction(mainWindow)
97          self.actionOpen_File.setObjectName(_fromUtf8("actionOpen_File"))
98
99          self.actionSave = QtGui.QAction(mainWindow)
100         self.actionSave.setObjectName(_fromUtf8("actionSave"))
101
102         self.actionClose = QtGui.QAction(mainWindow)
103         self.actionClose.setObjectName(_fromUtf8("actionClose"))
104
105         self.actionAbout = QtGui.QAction(mainWindow)
106         self.actionAbout.setObjectName(_fromUtf8("actionAbout"))
107
108         self.actionMotif = QtGui.QAction(mainWindow)
109         self.actionMotif.setObjectName(_fromUtf8("actionMotif"))
110
111         self.actionWindows = QtGui.QAction(mainWindow)
112         self.actionWindows.setObjectName(_fromUtf8("actionWindows"))
113
114         self.actionCde = QtGui.QAction(mainWindow)
115         self.actionCde.setObjectName(_fromUtf8("actionCde"))
116
117         self.actionPlastique = QtGui.QAction(mainWindow)
118         self.actionPlastique.setObjectName(_fromUtf8("actionPlastique"))
119
120         self.actionCleanlooks = QtGui.QAction(mainWindow)
121         self.actionCleanlooks.setObjectName(_fromUtf8("actionCleanlooks"))
122
123         self.actionWindowsvista = QtGui.QAction(mainWindow)
124         self.actionWindowsvista.setObjectName(_fromUtf8("actionWindowsvista"))
125
126         self.menuFile.addAction(self.actionOpen_File)
127         self.menuFile.addAction(self.actionSave)
128         self.menuFile.addSeparator()
129         self.menuFile.addAction(self.actionClose)
130
131         self.menuHelp.addAction(self.actionAbout)
```

```
132
133         self.menubar.addAction(self.menuFile.menuAction())
134         self.menubar.addAction(self.menuHelp.menuAction())
135
136         self.retranslateUi(mainWindow)
137
138         QtCore.QMetaObject.connectSlotsByName(mainWindow)
139
140     def retranslateUi(self, mainWindow):
141         mainWindow.setWindowTitle(_translate("mainWindow", \
142             "Data␣Acquisition␣Channel␣for␣an␣E-Nose", None))
143
144         self.label.setText(_translate("mainWindow",\
145              "Data␣Acquisition␣Channel␣for␣an␣Electronic␣Nose", None))
146
147         self.startButton.setStatusTip(_translate("mainWindow",\
148              "Press␣to␣go␣to␣the␣main␣menu", None))
149
150         self.startButton.setText(_translate("mainWindow", "START", None))
151         self.startButton.clicked.connect(self.open_menu)
152
153         self.comboBox.setItemText(0, _translate("mainWindow",\
154              "cde", None))
155         self.comboBox.setItemText(1, _translate("mainWindow",\
156              "Cleanlooks", None))
157         self.comboBox.setItemText(2, _translate("mainWindow",\
158              "motif", None))
159         self.comboBox.setItemText(3, _translate("mainWindow",\
160              "Plastique", None))
161         self.comboBox.setItemText(4, _translate("mainWindow",\
162              "windowsvista", None))
163         self.comboBox.setItemText(5, _translate("mainWindow",\
164              "Windows", None))
165         self.comboBox.setItemText(6, _translate("mainWindow",\
166              "fusion", None))
167         self.comboBox.activated[str].connect(self.style_choice)
168
169         self.menuFile.setTitle(_translate("mainWindow", "File", None))
170
171         self.menuHelp.setTitle(_translate("mainWindow", "Help", None))
172
173         self.actionOpen_File.setText(_translate("mainWindow", "Open", None))
174
175         self.actionSave.setText(_translate("mainWindow", "Save", None))
176
177         self.actionClose.setText(_translate("mainWindow", "Exit", None))
178
179         self.actionAbout.setText(_translate("mainWindow", "About", None))
180
181
```

```
182     def open_menu(self):
183         self.window = QtGui.QMainWindow()
184         self.ui = Ui_mainMenu()
185         self.ui.setupUi(self.window)
186         self.window.show()
187         mainWindow.hide()
188 
189     def style_choice(self, text):
190         # self.label.setText(text)
191         QtGui.QApplication.setStyle(QtGui.QStyleFactory.create(text))
192 
193 
194 if __name__ == "__main__":
195 
196     app = QtGui.QApplication(sys.argv)
197     mainWindow = QtGui.QMainWindow()
198     ui = Ui_mainWindow(mainWindow)
199     # ui.setupUi(mainWindow)
200     mainWindow.show()
201     sys.exit(app.exec_())
```

APPENDIX D

Start Screen of the Application code

```
from PyQt4 import QtCore, QtGui
from data_acquisition import Ui_data_acquisition
from thesis_GUI_data_analysis import Ui_data_analysis
from plot_several import PlotSeveralSamples
from usb_device import USBDevice
import sys

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text,\
             disambig,\
              _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context,\
             text, disambig)


class Ui_mainMenu(object):
    def setupUi(self, mainMenu):
        self.dataOpID = 0

        self.mainMenu = mainMenu
        self.mainMenu.setObjectName(_fromUtf8("mainMenu"))
```

```
32          self.mainMenu.resize(405, 376)
33
34          icon = QtGui.QIcon()
35
36          icon.addPixmap(QtGui.QPixmap(_fromUtf8(\
37              "monkey_on_32x32.png")),\
38               QtGui.QIcon.Normal,\
39                QtGui.QIcon.Off)
40
41          self.mainMenu.setWindowIcon(icon)
42
43          self.verticalLayoutWidget = QtGui.QWidget(self.mainMenu)
44          self.verticalLayoutWidget.setGeometry(QtCore.QRect(100, 30, 211, 151))
45          self.verticalLayoutWidget.setObjectName(\
46              _fromUtf8("verticalLayoutWidget"))
47
48          self.verticalLayout = QtGui.QVBoxLayout(self.verticalLayoutWidget)
49          self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
50
51          self.pushButton_2 = QtGui.QPushButton(self.verticalLayoutWidget)
52          self.pushButton_2.setObjectName(_fromUtf8("pushButton_2"))
53
54          self.verticalLayout.addWidget(self.pushButton_2)
55
56          self.pushButton_3 = QtGui.QPushButton(self.verticalLayoutWidget)
57          self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
58
59          self.verticalLayout.addWidget(self.pushButton_3)
60
61          self.pushButton = QtGui.QPushButton(self.verticalLayoutWidget)
62          self.pushButton.setObjectName(_fromUtf8("pushButton"))
63
64          self.verticalLayout.addWidget(self.pushButton)
65
66          self.verticalLayoutWidget_2 = QtGui.QWidget(self.mainMenu)
67          self.verticalLayoutWidget_2.setGeometry(QtCore.QRect(\
68              100, 240, 211, 121))
69          self.verticalLayoutWidget_2.setObjectName(\
70              _fromUtf8("verticalLayoutWidget_2"))
71
72          self.verticalLayout_2 = QtGui.QVBoxLayout(self.verticalLayoutWidget_2)
73          self.verticalLayout_2.setObjectName(_fromUtf8("verticalLayout_2"))
74
75          self.pushButton_5 = QtGui.QPushButton(self.verticalLayoutWidget_2)
76          self.pushButton_5.setObjectName(_fromUtf8("pushButton_5"))
77
78          self.verticalLayout_2.addWidget(self.pushButton_5)
79
80          self.pushButton_6 = QtGui.QPushButton(self.verticalLayoutWidget_2)
81          self.pushButton_6.setObjectName(_fromUtf8("pushButton_6"))
```

```
82
83          self.verticalLayout_2.addWidget(self.pushButton_6)
84
85          self.pushButton_4 = QtGui.QPushButton(self.verticalLayoutWidget_2)
86          self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
87
88          self.verticalLayout_2.addWidget(self.pushButton_4)
89
90          self.label = QtGui.QLabel(self.mainMenu)
91          self.label.setGeometry(QtCore.QRect(70, 200, 291, 21))
92
93          font = QtGui.QFont()
94          font.setPointSize(15)
95          font.setItalic(False)
96          font.setUnderline(True)
97          font.setStrikeOut(False)
98          font.setKerning(True)
99          font.setStyleStrategy(QtGui.QFont.PreferDefault)
100
101         self.label.setFont(font)
102         self.label.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))
103         self.label.setObjectName(_fromUtf8("label"))
104
105         self.usb = USBDevice()
106         #self.usb.connect_usb()
107
108         self.retranslateUi(self.mainMenu)
109         QtCore.QMetaObject.connectSlotsByName(self.mainMenu)
110
111     def retranslateUi(self, mainMenu):
112         mainMenu.setWindowTitle(_translate("mainMenu",\
113               "main␣menu",\
114                None))
115
116         self.pushButton_2.setText(_translate("mainMenu", "LED␣on", None))
117         self.pushButton_2.clicked.connect(self.turnOnLED)
118
119         self.pushButton_3.setText(_translate("mainMenu", "LED␣off", None))
120         self.pushButton_3.clicked.connect(self.turnOffLED)
121
122         self.pushButton.setText(_translate("mainMenu",\
123               "Plot␣Comparision",\
124                None))
125         self.pushButton.clicked.connect(self.plot_comparision)
126
127         self.pushButton_5.setText(_translate("mainMenu",\
128               "Data␣Acquisition",\
129                None))
130         self.pushButton_5.clicked.connect(self.data_acquisition)
131
```

```
132         self.pushButton_6.setText(_translate("mainMenu",\
133              "Data␣Analysis", None))
134         self.pushButton_6.clicked.connect(self.data_analysis)
135 
136         self.pushButton_4.setText(_translate("mainMenu", "Exit", None))
137         self.pushButton_4.clicked.connect(self.end_application)
138 
139         self.label.setStatusTip(_translate("mainMenu",\
140              "Data␣Acquisition␣Channel",\
141                None))
142 
143         self.label.setText(_translate("mainMenu", \
144             "DATA␣ACQUISITION␣CHANNEL",\
145             None))
146 
147     def turnOnLED(self):
148         self.usb.OUT.write('1')
149 
150     def turnOffLED(self):
151         self.usb.OUT.write('2')
152 
153     def data_acquisition(self):
154         self.window = QtGui.QMainWindow()
155         self.ui = Ui_data_acquisition()
156         self.ui.setupUi(self.window, self.usb)
157         self.window.show()
158         # self.mainMenu.hide()
159 
160     def plot_comparision(self):
161         self.PlotS = PlotSeveralSamples()
162 
163     def data_analysis(self):
164         self.ui = Ui_data_analysis()
165 
166     def end_application(self):
167         choice = QtGui.QMessageBox.question(self.mainMenu,\
168              'Exit',\
169              "Do␣you␣want␣to␣terminate␣the␣application?",\
170                  QtGui.QMessageBox.Yes|QtGui.QMessageBox.No)
171         if choice == QtGui.QMessageBox.Yes:
172             self.usb.close_device()
173             sys.exit()
174         else:
175             pass
176 
177 
178 if __name__ == "__main__":
179     app = QtGui.QApplication(sys.argv)
180     mainMenu = QtGui.QWidget()
181     ui = Ui_mainMenu()
```

```
182     ui.setupUi(mainMenu)
183     mainMenu.show()
184     sys.exit(app.exec_())
```

APPENDIX E

Data Acquisition Implementation code

```
from PyQt4 import QtCore, QtGui
from usb_device import USBDevice
import numpy as np
import time
import sys

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8

    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)

except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)


class Ui_data_acquisition(object):

    def setupUi(self, data_acquisition, USB):
        self.data_acq = data_acquisition
        data_acquisition.setObjectName(_fromUtf8("data_acquisition"))
        data_acquisition.resize(406, 377)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(_fromUtf8(
```

```
32              "../../.designer/backup/monkey_on_32x32.png")),
33               QtGui.QIcon.Normal,
34               QtGui.QIcon.Off)
35          data_acquisition.setWindowIcon(icon)
36          self.label = QtGui.QLabel(data_acquisition)
37          self.label.setGeometry(QtCore.QRect(10, 190, 291, 41))
38
39
40          font = QtGui.QFont()
41          font.setPointSize(16)
42
43          self.label.setFont(font)
44          self.label.setObjectName(_fromUtf8("label"))
45
46          self.btn_start = QtGui.QPushButton(data_acquisition)
47          self.btn_start.setGeometry(QtCore.QRect(50, 300, 91, 41))
48          self.btn_start.setObjectName(_fromUtf8("btn_start"))
49
50          self.number_loops = QtGui.QSpinBox(data_acquisition)
51          self.number_loops.setGeometry(QtCore.QRect(270, 30, 41, 31))
52          self.number_loops.setObjectName(_fromUtf8("number_loops"))
53
54          self.label_2 = QtGui.QLabel(data_acquisition)
55          self.label_2.setGeometry(QtCore.QRect(10, 30, 251, 31))
56
57          font = QtGui.QFont()
58          font.setFamily(_fromUtf8("Calibri"))
59          font.setPointSize(18)
60          font.setBold(True)
61          font.setWeight(75)
62
63          self.label_2.setFont(font)
64          self.label_2.setObjectName(_fromUtf8("label_2"))
65
66          self.btn_back = QtGui.QPushButton(data_acquisition)
67          self.btn_back.setGeometry(QtCore.QRect(260, 300, 91, 41))
68          self.btn_back.setObjectName(_fromUtf8("btn_back"))
69
70          self.label_3 = QtGui.QLabel(data_acquisition)
71          self.label_3.setGeometry(QtCore.QRect(230, 110, 151, 31))
72
73          font = QtGui.QFont()
74          font.setFamily(_fromUtf8("Calibri"))
75          font.setPointSize(18)
76          font.setBold(True)
77          font.setWeight(75)
78
79          self.label_3.setFont(font)
80          self.label_3.setAlignment(QtCore.Qt.AlignRight |
81               QtCore.Qt.AlignTrailing |
```

```
82              QtCore.Qt.AlignVCenter)
83          self.label_3.setObjectName(_fromUtf8("label_3"))
84
85          self.wave_selector = QtGui.QSpinBox(data_acquisition)
86          self.wave_selector.setGeometry(QtCore.QRect(170, 110, 41, 31))
87          self.wave_selector.setObjectName(_fromUtf8("wave_selector"))
88
89          self.usb = USB
90
91          self.retranslateUi(data_acquisition)
92          QtCore.QMetaObject.connectSlotsByName(data_acquisition)
93
94      def retranslateUi(self, data_acquisition):
95          data_acquisition.setWindowTitle(_translate("data_acquisition",
96              "Data␣Acquisition",
97              None))
98          self.label.setText(_translate("data_acquisition",
99              "Performing␣Data␣Acquisition...",
100             None))
101
102         self.btn_start.setStatusTip(_translate("data_acquisition",
103             "Start␣data␣acquisition␣process",
104             None))
105         self.btn_start.setText(_translate("data_acquisition",
106             "START",
107             None))
108         self.btn_start.clicked.connect(self.data_transfer)
109
110         self.label_2.setStatusTip(_translate("data_acquisition",
111           "Select␣how␣many␣times␣you␣wish␣to␣run␣the␣data␣acquisition␣phase",
112           None))
113         self.label_2.setText(_translate("data_acquisition",
114             "Number␣of␣loops␣desired",
115             None))
116
117         self.btn_back.setStatusTip(_translate("data_acquisition",
118             "Back␣to␣the␣main␣menu",
119             None))
120         self.btn_back.setText(_translate("data_acquisition",
121             "DONE", None))
122         self.btn_back.setEnabled(False)
123         self.btn_back.clicked.connect(self.close_window)
124
125         self.label_3.setStatusTip(_translate("data_acquisition",
126             "Select␣which␣wave␣you␣wish␣to␣output",
127             None))
128         self.label_3.setText(_translate("data_acquisition",
129             "Wave␣Selector",
130             None))
131
```

```
132
133     def data_transfer(self):  # loop_count, wave_selector
134         # self.btn_start.setEnabled(False)
135         var = 0  # represent the number of times this loop will run
136         bytes_to_read = 64
137
138         # how many times in a row does the user wish to run the acquisition
139         auxiliar = self.number_loops.value()
140         loops = str(auxiliar)
141
142         # wave_selector: 1 - 1V | 2 - 100mV | 3 - 10mV
143         val = self.wave_selector.value()
144         op = str(val)
145
146         # value to start data acquisition
147         self.usb.OUT.write('3')
148
149         self.usb.OUT.write(loops)
150
151         time.sleep(0.25)
152
153         while var < auxiliar:
154             self.usb.OUT.write(op)
155             for i in range(9216):
156                 try:
157                     data = self.usb.IN.read(bytes_to_read)
158
159                 except:
160                     print("Can't receive data from the PSoC")
161                     self.usb.close_device()
162
163                 if i == 0:
164                     aux = data
165                 else:
166                     aux = np.append(aux, data)
167
168             self.usb.save_usb_data(aux)
169             print(str(var))
170             var += 1
171             data = None
172             time.sleep(6)
173
174         self.btn_start.setEnabled(True)
175         self.btn_back.setEnabled(True)
176
177     def updateProgressBar(self, val):
178         self.progressBar.setValue(val)
179
180     def close_window(self):
181         self.data_acq.close()
```

```
182
183
184 if __name__ == "__main__":
185
186     app = QtGui.QApplication(sys.argv)
187     data_acquisition = QtGui.QWidget()
188     ui = Ui_data_acquisition()
189     ui.setupUi(data_acquisition)
190     data_acquisition.show()
191     sys.exit(app.exec_())
```

Appendix F

Data Analysis Implementation code

```
1  import sys
2  import os
3  from PyQt4 import QtGui, QtCore
4  from scipy.signal import get_window
5  import matplotlib.pyplot as plt
6  import numpy as np
7  import scipy as sci
8  import pandas as pd
9  import datetime
10 from matplotlib.figure import Figure
11 from matplotlib.backends.backend_qt4agg
12     import FigureCanvasQTAgg as FigureCanvas
13 from matplotlib.backends.backend_qt4agg
14     import NavigationToolbar2QT as NavigationToolbar
15
16
17 try:
18     _fromUtf8 = QtCore.QString.fromUtf8
19 except AttributeError:
20     def _fromUtf8(s):
21         return s
22
23 try:
24     _encoding = QtGui.QApplication.UnicodeUTF8
25
26 def _translate(context, text, disambig):
27     return QtGui.QApplication.translate(context, text, disambig, _encoding)
28 except AttributeError:
29     def _translate(context, text, disambig):
30         return QtGui.QApplication.translate(context, text, disambig)
31
```

```
32
33 class Ui_data_analysis(QtGui.QWidget):
34     def __init__(self):
35         super(Ui_data_analysis, self).__init__()
36         self.fileDict = {}  # used for the plot_several_samples case
37         self.plotColor = ['b','g','r','c','m','y','k']
38         self.plotLabel=""
39         self.aux = 0
40         self.buffer = None
41         self.signal = None
42         self.wave = None
43         self.time = None
44         self.FFT = None
45         self.phase = None
46         self.ff = None
47         self.NP = None
48         self.SIZE = 0
49         self.window = None
50         self.freqs = None
51         self.w = None
52         self.sample_name = None
53         self.x = None
54         self.y = None
55         self.run()
56         self.initUI()
57
58     def initUI(self):
59         self.setGeometry(600, 300, 1000, 600)
60         self.center()
61         self.setWindowTitle("Data Analysis")
62         icon = QtGui.QIcon()
63         icon.addPixmap(QtGui.QPixmap(_fromUtf8("monkey_on_32x32.png")),
64             QtGui.QIcon.Normal, QtGui.QIcon.Off)
65         self.setWindowIcon(icon)
66         grid = QtGui.QGridLayout()
67         self.setLayout(grid)
68
69         btn1 = QtGui.QPushButton('Signal', self)
70         btn1.resize(btn1.sizeHint())
71         btn1.clicked.connect(self.plot_signal)
72         grid.addWidget(btn1, 2, 0)
73
74         btn2 = QtGui.QPushButton('Frequency Response', self)
75         btn2.resize(btn1.sizeHint())
76         btn2.clicked.connect(self.plot_frequency)
77         grid.addWidget(btn2, 2, 1)
78
79         self.figure = plt.figure(figsize=(15, 5))
80         plt.rc('axes', labelsize=20)  # fontsize of the x and y labels
81         plt.rc('xtick', labelsize=20)  # fontsize of the tick labels
```

```
82          plt.rc('ytick', labelsize=20)  # fontsize of the tick labels
83
84          self.canvas = FigureCanvas(self.figure)
85          self.toolbar = NavigationToolbar(self.canvas, self)
86          grid.addWidget(self.canvas, 1, 0, 1, 2)
87          grid.addWidget(self.toolbar, 0, 0, 1, 2)
88          self.show()
89
90      def run(self):
91          self.load_data()
92          self.plot_processing()
93          self.obtain_axis_values()
94          self.compute_fft()
95          self.save_plots()
96
97      def plot_signal(self):
98          plt.cla()
99          ax = self.figure.add_subplot(111)
100         ax.plot(self.time, self.wave, 'b', label=self.sample_name)
101
102         ax.set_title('Time␣Response␣Plot')
103         plt.ylabel('voltage[V]')
104         plt.xlabel('time[s]')
105
106         plt.legend()
107         plt.grid(True)
108         self.canvas.draw()
109
110     def plot_frequency(self):
111         plt.cla()
112         ax1 = self.figure.add_subplot(111)
113         plt.semilogx(self.x, self.y, label='{}'.format(self.sample_name))
114         plt.legend()
115         plt.grid(True)
116         ax1.set_title('Magnitue␣Response␣Plot')
117         plt.ylabel('gain[dB]')
118         plt.xlabel('frequency[rad/s]')
119
120         #ax2 = self.figure.add_subplot(212)
121         #plt.semilogx(self.phase, label='{}'.format(self.sample_name))
122         #plt.legend()
123         #plt.grid(True)
124         #ax2.set_title('Phase Response Plot')
125
126         self.canvas.draw()
127
128     def plot_bh(self):
129         plt.plot(self.freqs, 20 * np.log10(np.abs(self.w)), label=self.window)
130         plt.title('Blackman␣Window')
131         plt.ylabel("amplitude")
```

```
132         plt.xlabel('frequency bin #')
133         plt.legend()
134 
135     def center(self):  # inspect this method later
136         qr = self.frameGeometry()
137         cp = QtGui.QDesktopWidget().availableGeometry().center()
138         qr.moveCenter(cp)
139         self.move(qr.topLeft())
140 
141     def load_data(self):
142         filename = QtGui.QFileDialog.getOpenFileName(self, 'Open File')
143         self.extract_data(filename)
144 
145     def compute_fft(self):
146         self.FFT = sci.fft(self.signal)  # buffer
147         self.NP = len(self.signal)  # buffer
148         Fs = 46875  # sampling frequency > initial: 25000
149         self.ff = np.arange(0, (self.NP - 1)) / (self.NP - 1) * Fs
150         self.phase = np.angle(self.FFT, deg=True)
151         self.x = self.ff[:int((len(self.ff) - 1) / 2)]  # x axis of fft
152         self.y = 20 * sci.log10(abs(
153             self.FFT[1:int(len(self.FFT) / 2)] / np.sqrt(self.NP)))
154 
155     def compute_bh(self):  # compute blackman harris window
156         # I have no idea about the number that multiplies with NP
157         x = sci.linspace(start=0.0, stop=self.NP * 0.00004, num=self.NP)
158         for self.window in ['blackman']:  # 'hamming'
159         n = self.SIZE
160         self.w = np.fft.rfft(self.signal * get_window(self.window,
161             self.signal.size), self.signal.size)
162         self.freqs = np.fft.rfftfreq(self.signal.size, d=x[1] - x[0])
163 
164     def save_plots(self):
165         # SAVE USING PANDAS BUT WITH concat METHOD
166         pd.concat([pd.DataFrame(data=self.signal, columns=['SIGNAL']),
167         pd.DataFrame(data=self.y, columns=["FREQUENCY RESPONSE"]),
168         pd.DataFrame(data=self.x, columns=["FREQUENCY VALUES"])],
169         axis=1).to_csv('connect_1.csv', index=False)
170 
171         # csv file with samples and their date|hour of acquisition
172         valDate = datetime.datetime.now()
173         samples = pd.DataFrame(data=[self.sample_name])
174         sampleTime = pd.DataFrame(data=[valDate])#, columns=["Time"])
175         zumba = pd.concat([samples, sampleTime], axis=1)
176         with open("myDatabase.csv", 'a') as f:
177         zumba.to_csv(f, index=False, header=False)
178 
179     def plot_processing(self):
180         i = 500
181         while i >= 0:
```

```
182         res = self.buffer[i - 1] - self.buffer[i]
183         if  res > 0 and abs(res) > 100:
184         self.signal = self.buffer[i:]
185         return
186         i = i - 1
187 
188     def closeEvent(self, event):
189         print("User has clicked the red x on the main window")
190         event.accept()
191 
192     def extract_data(self, filename):
193         f = open(filename, 'r')
194         self.sample_name = filename.split('/')[-1]
195 
196         # READ EVERY LINE FROM THE FILE AND STORE IT ON AN ARRAY
197         alist = [line.rstrip() for line in open(filename)]
198 
199         # real size of the signal being analyzed
200         self.SIZE = int(len(alist) / 2)
201         self.buffer = np.ones(self.SIZE)
202 
203         # conversion of high and low bytes to a 16-bits value
204         i = 0
205         j = 0
206         while i <= len(alist) - 1:
207         var1 = int(alist[i])
208         var2 = var1 + (int(alist[i + 1]) * 256)
209 
210         self.buffer[j] = var2
211         j = j + 1
212         i = i + 2
213         f.close()
214         # TO CUT-OFF DC VALUE
215         # self.signal = self.buffer - np.mean(self.buffer)
216 
217     def obtain_axis_values(self):
218         for value in self.signal:
219             aux = value / 65536
220             self.wave = np.append(self.wave, aux)
221 
222         seconds = len(self.signal)/46875
223         self.time = np.linspace(start=0.0, num=len(self.wave), stop=seconds)
224 
225 
226 def main():
227     app = QtGui.QApplication(sys.argv)
228     display = Ui_data_analysis()
229     sys.exit(app.exec_())
230 
231 
```

```
232 if __name__ == '__main__':
233     main()
```