MASTER THESIS

TITLE: Test and Measurement under SDR framework

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Gerard Romero Lorenzo

DIRECTOR: Antoni Gelonch Bosch

DATE: October 24th 2017
In times when telecommunications are one of the cardinal points of our society, engineers people who study them are very important to be able to see and analyse the signals which make up them. That is why experts, in this broad field, need a tool capable of providing a vision of all the signals which they are sending or receiving.

In this project, we tried to implement a graphical interface tool able to give visibility to all the signals received by any telecommunication device. Making the user itself able to interact with it, in such a way he has the absolute control of what he wants to do. In addition, this tool is designed to support all those users who use a framework based on Software Defined Radio.

In this project, first, it will be explained what is exactly the Software Defined Radio technology, including its benefits and giving examples of application where it is used. Also, there is a collection of frameworks that are based on this technology, explaining in more detail ALOE Middleware.

Next, we will try to explain in detail how the software is built, what is the data flow, the graphs in which it is composed and the functionality they have.

And finally, the entire implementation will be incorporated into the Software Defined Radio technology, especially, within the ALOE middleware where real-time tests will be sent by sending audio signals and LTE signals.
INDEX

INTRODUCTION ................................................................................................................. 1

CHAPTER 1. SOFTWARE DEFINED RADIO ........................................................................ 2

1.1. Benefits provided SDR Technology ........................................................................... 2
1.2. Applications of SDR Technology ............................................................................. 3
  1.3.1. GNURADIO ........................................................................................................... 3
  1.3.2. REDHAWK ........................................................................................................... 4
  1.3.3. RUSTRADIO ....................................................................................................... 4
  1.3.4. ALOE ................................................................................................................... 4
    1.3.4.1. Layers and architecture ................................................................................. 5
    1.3.4.2. Multi-Processing platform ............................................................................. 6
    1.3.4.3. Topology ......................................................................................................... 6

CHAPTER 2. GRAPH USER INTERFACE .......................................................................... 8

2.1. Objective & Scope ..................................................................................................... 8
2.2. Graphs ..................................................................................................................... 8
  2.2.1. Time graph .......................................................................................................... 9
    2.2.1.1. Lineal and Logarithmic scale ....................................................................... 9
    2.2.1.2. Inphase and Quadrature mode ................................................................. 9
    2.2.1.3. Module & Phase mode ................................................................................ 10
    2.2.1.4. Average function ...................................................................................... 10
    2.2.1.5. Block function ............................................................................................ 10
  2.2.2. Frequency graph ............................................................................................... 11
    2.2.2.1. FFT Representation ................................................................................... 11
    2.2.2.2. dB function ............................................................................................... 12
  2.2.3. Spectrogram graph ............................................................................................. 12
  2.2.4. Constellation graph .......................................................................................... 13
  2.2.5. ALOE's graphs .................................................................................................. 13
    2.2.5.1. Block Diagram graph ................................................................................. 13
    2.2.5.2. ALOE Execution Time's graph ............................................................... 14
  2.3. Functionalities ......................................................................................................... 14
    2.3.1. Graph's selector ............................................................................................. 14
CHAPTER 3. RESULTS

3.1. Graphic representation of a sinusoidal signal
   3.1.1. Model A
   3.1.2. Model B
   3.1.3. Performance

3.2. Graphic representation of an audio signal

3.3. Graphic representation of an LTE signal

CHAPTER 4. CONCLUSIONS

CHAPTER 5. FUTURE WORK

BIBLIOGRAPHY

ANNEX

Annex 1: SRSGUI - Class diagram
FIGURE LIST

Fig. 1-1: ALOE Processing layers ................................................................. 5
Fig. 1-2: ALOE Architecture ....................................................................... 5
Fig. 1-3: Multiprocess graphic description .................................................. 6
Fig. 1-4: ALOE modules organization ......................................................... 7
Fig. 2-1: Graph’s options .............................................................................. 9
Fig. 2-2: Scale graph .................................................................................. 9
Fig. 2-3: Average selector .......................................................................... 10
Fig. 2-4: Block selector .............................................................................. 11
Fig. 2-5: dB selector .................................................................................. 12
Fig. 2-6: Spectrogram functionalities .......................................................... 13
Fig. 2-7: Example ALOE Block diagram .................................................... 13
Fig. 2-8: Example ALOE Execution Time’s graph ....................................... 14
Fig. 2-9: Options graph selector ................................................................. 15
Fig. 2-10: Screenshot Button ..................................................................... 15
Fig. 2-11: Drag Button ................................................................................ 16
Fig. 2-12: Zoom Button ............................................................................. 16
Fig. 2-13: Auto scale Button ..................................................................... 16
Fig. 2-14: Options legend position ............................................................. 16
Fig. 2-15: Changing Title and Axis Label .................................................... 17
Fig. 2-16: Colour graph selector ................................................................. 17
Fig. 2-17: Linestyle graph selector .............................................................. 17
Fig. 2-18: Scattering format graph selector ............................................... 18
Fig. 2-19: Background colour selector ....................................................... 18
Fig. 2-20: Composition of SRSGUI Library ................................................ 19
Fig. 2-21: SRSGUI Directory Structure ...................................................... 20
Fig. 2-22: Workflow SRSGUI ..................................................................... 22
Fig. 2-23: ALOE modules ....................................................................... 24
Fig. 2-24: Full ALOE modules ................................................................. 25
Fig. 3-1: Time signal model A. On the right lineal scale, on the left logarithmic scale ........................................................................ 27
Fig. 3-2: Time averaging with $\mu=0.75$ ...................................................... 27
Fig. 3-3: Signal time with 2 blocks ............................................................. 27
Fig. 3-4: Frequency I/Q Lineal Scale .......................................................... 28
Fig. 3-5: Frequency I/Q Logarithmic Scale .................................................. 28
Fig. 3-6: (a) FFT signal module lineal. (b) FFT signal module logarithmic. (c) FFT Phase signal ................................................................. 28
Fig. 3-7: Function ’dB’. Top is $10\log(|x|)$, button is $20\log(|x|)$ .................. 29
Fig. 3-8: Frequency signal with different FFT Window ................................ 29
Fig. 3-9: Spectrogram signal model A ....................................................... 30
Fig. 3-10: Signal in time domain ............................................................... 30
Fig. 3-11: Constellation Signal ................................................................. 31
Fig. 3-12: Signal in Frequency domain ..................................................... 31
Fig. 3-13: Spectrogram model B ............................................................... 32
Fig. 3-14: Audio application ..................................................................... 33
Fig. 3-15: Time Audio with 20 blocks samples .......................................... 33
Fig. 3-16: Frequency plane: real tone 6kHz captured by microphone ........ 34
Fig. 3-17: Frequency plane: real tone 10kHz captured by microphone ........ 34
Fig. 3-18: Audio constellation .................................................................................. 35
Fig. 3-19: Sweep frequencies Spectrogram .......................................................... 35
Fig. 3-20: LTE signal - Time plane .......................................................................... 35
Fig. 3-21: LTE Signal - Frequency plane – I/Q .................................................... 36
Fig. 3-22: LTE Signal - Frequency plane from 0 to fs - Module and Phase .......... 36
Fig. 3-23: LTE Signal - Frequency plane from -fs/2 to fs/2 – Module ............. 36
Fig. 3-23: LTE signal – Power .................................................................................. 36
Fig. 3-24: LTE Signal – Spectrogram – from 0 to fs ........................................... 37
Fig. 3-24: LTE Signal – Spectrogram – from -fs/2 to fs/2 .................................... 37
Fig. Annex - 1-1: SRSGUI Class Diagram.............................................................. 41

TABLE LIST

Table 3-1: Performance.............................................................................................. 32
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOE</td>
<td>Abstraction Layer and Operating Environment</td>
</tr>
<tr>
<td>DDC</td>
<td>Digital Down Converter</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>Fo</td>
<td>Frequency carrier</td>
</tr>
<tr>
<td>Fs</td>
<td>Frequency sampling</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>SDR</td>
<td>Software Defined Radio</td>
</tr>
<tr>
<td>TS</td>
<td>Time Slot</td>
</tr>
<tr>
<td>USRP</td>
<td>Universal Software Radio Peripheral</td>
</tr>
</tbody>
</table>
INTRODUCTION

The present project consists on the development of a system able to measure captured signals by wireless reception devices under a Software Defined Radio (SDR) implementation. In addition to capturing them, the system will be able to display the signals graphically and provide to the users the option of interacting with the software to determine what actions can do on the captured signal.

The main objective of this project is to be able to capture the working conditions of the wireless signals thanks to the implemented software. Provide different functionalities so the user himself knows that he is watching always and can adapt to the software to his own needs.

To explain the development to create such a system, the study of SDR technology will be undertaken, explaining its interiorities and giving examples of the frameworks that use it. The explanation of the different frameworks will focus on the so-called ALOE middleware, providing technical details.

Secondly, we will define the needs which for software is as important as the one developed in this study, explaining the global needs that have led to it, how it has been developed and what functionalities provides the tool in both modes: standalone and within a framework of SDR.

And finally, an exhaustive analysis will be done to see the tool performance in different environments, as can be LTE or voice signal.
CHAPTER 1. SOFTWARE DEFINED RADIO

The concept of Radio Defined Software is defined as a radio communication system in which most or all of its physical layer functions are defined by software replacing all the hardware part of a communication system (mixers, filters, modulators / demodulators, detectors, etc ...).

The substitution of the hardware elements by software makes it possible to process such quantities of data never thought. This phenomenon is produced by the fact that all this amount of data is delivered directly to the general-proposed processor leaving aside the electronic circuits which have all the hardware part.

The great impact that this type of technological systems is assuming makes one believe it will be one of the most dominant in the world of telecommunications, due to an architecture as flexible as it is profitable. The limitations caused by the hardware are more than palpable: difficulty in replacing physical elements, the price of them can become very high, interference, distortions, etc ... Meanwhile, SDR systems eliminate much of the physical elements and consequently decrease significantly prices, reduce considerably interferences through digital processing of data and increase the efficiency of the overall system.

1.1. Benefits provided SDR Technology

The benefits derived from this technology are marked, not only economically, but also at a technical level. A software-defined radio can be flexible enough to avoid the "limited spectrum" assumptions of previous kinds of radios, in one or more ways including the following benefits.

Spread spectrum and ultrawideband techniques allow several transmitters to transmit in the same places on the same frequency, with very little interference. Typically, they are combined with one or more error detection and correction techniques to fix all the errors caused by that interferences.

It is possible to create a signal reception system able to work with the antennas together and to avoid mutual interferences between them using unused frequencies. In addition to cooperation among antennas, operating frequencies and transmitting power can be adjusted for the whole set of antennas to avoid interferences into the own system but also to other wireless systems ones.

Another benefit to implement SDR Technology is Dynamic transmitter power adjustment. With this technic, the system can reduce transmission power to minimum when the system considers it is necessary, therefore, it can reduce interferences to other signals in the moment when it is not send anything or send a very low signal. Besides, it can increase the durability of batteries in portable devices.
Finally, using SDR technology, a wireless mesh network could be created by increasing the capacity of the network and reducing the necessary powers of the nodes. As a consequence, nodes transmit only the nearby nodes to distribute the data.

The features and benefits of this system make it a great attraction for the great technologies like 5G, IoT and sensor networks.

1.2. Applications of SDR Technology

There is a wide variety of applications where SDR systems can extract much of their potential. It is so flexible that it can come up with applications of all possible classes. Here are some examples:

- Amateur Radio.
- Radio Astronomy.
- 4G LTE infrastructure.
- Global Navigation Satellite Systems

1.3. Frameworks

In order to SDR devices to be able to operate from any telecom device, you need a software layer able to provide such functionality. However, before developing software, a framework should be created which provides low-level interface functions.

Next, we will see some examples of frameworks capable of implementing Software Defined Radio technology. The explanation will focus on ALOE middleware, SDR Framework used in the implementation of the current software project.

1.3.1. GNURADIO

GNU Radio is open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems. [1]
1.3.2. REDHAWK

REDHAWK is a software-defined radio (SDR) framework designed to support the development, deployment, and management of real-time software radio applications. To support the design and development of software applications, REDHAWK provides tools that allow development and testing of software modules called "Components" and composition of Components into "Waveform Applications" that can be seamlessly deployed on a single computer or multiple network-enabled computers.

The REDHAWK integrated development environment (IDE) provides tools to support development of REDHAWK software. The development and deployment of REDHAWK Applications are aided by graphical editors and drag-and-drop Waveform construction. The IDE allows users to interact with and control multiple running REDHAWK instances and applications. [2]

1.3.3. RUSTRADIO

Rust Radio is a SDR and signal processing framework with an emphasis on simple processing block implementations.

Unlike other SDR frameworks where blocks take and return discrete buffers of samples, Rust Radio's processing blocks operate on iterators. This allows each processing block to view the incoming samples as a continuous stream, and the actual buffering of samples is abstracted away, which leads to much shorter and simpler block implementations. Other benefits of iterator-based processing blocks are:

- Blocks can be used independently. No other part of the system is needed to use a particular block, which makes testing and code re-use easy. Feel free to run blocks on vectors, or lists, or anything that can be iterated over.
- Rust's generics allow us to process anything that our blocks will allow. This means that we can use floating point, fixed point, integer, or even arbitrary-precision numbers to represent our samples. [3]

1.3.4. ALOE

ALOE (Abstraction Layer and Operating Environment) is a software that acts as a bridge between an operating system or databases and applications or MIDDLEWARE, which facilitates the development of code in multi-platform.
1.3.4.1. **Layers and architecture**

This radio software has four processing layers, each of which is shown in the image below.

![ALOE Processing layers](image1)

**Fig. 1-1:** ALOE Processing layers

The Hardware layer is a group of Processors (Processing Elements or PE) that are interconnected.

The ALOE layer is where the Software is introduced and works separately on each computer in which it is connected.

The application layer is the layer where the programmed waveforms are executed. It is composed of all the modules that have been built and all the applications that interconnect the modules in order to create a processing chain.

And finally, we have the abstract application layer where all the modules, that have been programmed, are managed and provided with the necessary and/or available resources.

Conditioned by the implementation, ALOE uses some components and libraries like those that are shown in the following image:

![ALOE Architecture](image2)

**Fig. 1-2:** ALOE Architecture
In the API layer, ALOE uses the libraries to use their objects and thus facilitate the processing of the signal, and also uses the Daemons Software. This Software is a group of programs in order to be able to maintain a correct execution of ALOE.

1.3.4.2. *Multi-Processing platform*

ALOE has a great advantage over other software destined to radio communication and it has a multi-process platform. You can split applications into different processes (modules) and each one of the PE (Processor Elements) performs a different process from the application. In order to be able to get a good process, the PEs must be timed in time and have to process the data in an interval called Time Slot (TS), i.e., all the application modules, that are running must, be executed in a Time Slot.

Despite having the great advantage of multiprocessing, ALOE has an important disadvantage. This disadvantage is the latency of the signal, i.e., the total time when you pass all the data blocks during the entire process until you reach the exit can be very high if the application has a large number of modules.

![Multiprocess graphic description](image)

**Fig. 1-3:** Multiprocess graphic description

In an ALOE application we can have different types of modules and not all of them need the same resources in order to process the signal. For this reason, the same software has a Scheduling mechanism of the available resources. As a consequence, the system can give more resources to those modules that need them at certain times.

1.3.4.3. *Topology*

ALOE is distributed in modules able to interconnect each other to create applications. All modules have the same structure, and it is composed of an initialization function of the variables, a main function where the input and output interfaces are selected and a stop function. All modules have configurated files that determine the number of input and output interfaces and the data types they...
carry, other files to be able to include the necessary libraries for their execution, and settings configuration internal files.

**Fig. 1-4:** ALOE modules organization
CHAPTER 2. GRAPH USER INTERFACE

Signals in telecommunications are the real element that carries the information from one point to another. To know what information it takes with it, it is very important to analyse it, to modulate it for its full understanding. It is for this reason that graphic software has been designed to provide users with a tool capable of analysing and shaping the signals to their liking in order to extract information in the most effective way.

2.1. Objective & Scope

The main objective of this project is to implement a graphic interface to provide the user with the ability to see the signal captured by the receiver elements, and to be able to analyse the signals in the temporal and the frequency planes.

At first, the scope of this graphical interface was simply to show the typical graphs that are used in telecommunications, for example, the temporal and frequencial graphs, its averaged and the constellation. But as we saw the great potential of this interface and the few computing resources that are used to execute it, significant improvements were introduced both for the user's interaction with the graphs and by the number of graphs that can be visualized.

As a result, this graphical interface was restructured into two blocks with the same functionalities. On the one hand, it has the block with signals only with real part, from now MODEL A, oriented solely and exclusively to the academic use, and the second block with complex signals, from now MODEL B, oriented to the most part of investigation and capture real signals by USRP, DSP, etc.

2.2. Graphs

For this software, graphics are the most important and most outstanding element. They provide a clear and concise vision of how the signal is received, and can moulded to the user's taste thanks to the multiple options provided by the graphical interface.
To put in situation that can be represented under this software, we will provide a description of all developed graphs and their peculiarities.

### 2.2.1. Time graph

This type of graph shows the temporal signal as it arrives from the receiving elements. In model A only the real part will be shown, while in model B we have the signal with the real part and the imaginary part.

#### 2.2.1.1. Lineal and Logarithmic scale

In the graphical interface you can choose what kind of visualization you want on the vertical axis. There are two options: Linear or Logarithmic.

![Scale graph](image)

**Fig. 2-2:** Scale graph

By default, the linear option is selected, but it can be easily changed in the SCALE section.

#### 2.2.1.2. Inphase and Quadrature mode

This case is only in model B where you receive a complex signal. In this graph mode you can display the Inphase and Quadrature components of a complex signal, i.e. the real and imaginary part.

You can choose between linear and imaginary scales depending of the user’s necessities.
2.2.1.3. Module & Phase mode

This case is only in model B where you receive a complex signal. An important representation of the FFT has been with the module and phase. The module will be represented in positive axes and represents the following formula.

\[ |z| = \sqrt{Re^2 + Im^2} \quad (2.1) \]

Meanwhile, the phase will be always contained between \(-\pi\) and \(\pi\) and represents the following formula.

\[ \phi = \text{Arg } z = \text{atan2}(Im, Re) \quad (2.2) \]

In addition to this special feature, it can be represented either in linear or logarithmic scale depending on the previous state that has been selected.

2.2.1.4. Average function

In this mode the temporal signal can be viewed as a histogram of each points of the signals which have been plotted. With this mechanism you can represent a % of the signal memory. Thanks to a \(\mu\) value, we choose how much memory we want to graph. The interface gives users a mechanism to change that value online.

\[ x[n] = \mu * x_{\text{actual}}[n] + (1 - \mu) * x_{\text{anteriores}}[n] \quad (2.3) \]

In addition to this special feature, it can be represented either in linear or logarithmic scale depending on the previous state that has been selected.

2.2.1.5. Block function

To finalize the set of temporal graphs, the signal can be represented in block windows, i.e., you can get to represent blocks of data memory to see how the signal advances in a certain period of time by emulating a temporal evolution. The interface gives users a mechanism to change that value online.
Each block is determined by the amount of data entering each sequence by filling the graph from left to right.

In addition to this special feature, it can be represented either in linear scale or log scale depending on the previous state that has been selected.

2.2.2. Frequency graph

This type of graph shows the frequency representation of the signal captured by the receiver elements of the system. At all times, users will be able to choose at runtime the samples window which they want to apply for the realization of the FFT. The system limits the value to a number less than the total number of signal samples because that would entail an erroneous representation.

The value of the window will determine the number of points which the frequency signal represents. Knowing in this representation always the signal goes from 0 to sampling frequency, if users make the FFT with a small value, the graph reduce its signal resolution.

In both models A and B, the ‘Inphase and Quadrature’ and ‘Module and Phase’ components are represented because the FFT always deal with complex numbers.

The graphs of the sections ‘2.2.1.1 Lineal and Logarithmic scale’, ‘2.2.1.2 Inphase and Quadrature mode’, ‘2.2.1.3 Module & Phase mode’ and ‘ 2.2.1.4 Average function’ are generic graphs that are bought both by the temporary plane and by the frequency plane. Obviously, the information presented will be different but the type of implementation is the same, for this reason, they will not be explained again.

2.2.2.1. FFT Representation

Here, users can interact with the frequency signal by changing the way of representing it on the graph. Starting from the base that the signal always repeats each ‘fs’, the proportional graphical interface 3 different options:
• You can represent the signal from \(-fs/2\) to \(fs/2\) leaving the signal centred at frequency 0Hz

• The signal from 0Hz to \(fs\) can be represented by viewing the full spectrum.

• And I could represent only the part of 0Hz to \(fs/2\) by visualizing half. The latter can be used because in many cases the signal is symmetrical and you only want to see the first part to avoid confusion.

2.2.2.2. **dB function**

With this mode, the power and the voltage in dB can be represented if the respective signals are received. For this, there are two options that mark this mode of representation:

• If the signal we receive is power, the user has the option to dial the subtype '10 \* log10 (x)'. Here the graph represents dBW.

• If the signal we receive is voltage, the user has the option to dial the subtype '20 \* log10 (x)'. Here the graph represents dBμ.

The system is not able to say by itself the signal to be represented, this is why the interface gives to users all the autonomy to decide what they want to represent.

![dB[x]

Fig. 2-5: dB selector

In addition, it can be represented with the Average function, in turn, select the type of scale (linear or logarithmic) that you want to represent.

2.2.3. **Spectrogram graph**

This graphic representation is the most striking because the frequency component is represented as a function of time. You can see how the amplitude of the frequency in the passage of time gives a false sense of 3D. This refers to the frequency amplitude being represented in the graph pixel using a colour map, whereby each frequency value will have associated a coloured pixel. This representation is added by each Time slot with the next value of the vertical axis.

For this type of graph, the interface provides two runtime functionalities so users can interact with it. In it you can change the sample frequency of the frequency
signal, you can select the window to create the Fast Fourier Transform, besides, you can change the upper limit of the graph so which you can see as many TimeSlots as you want.

![ FFT interface screenshot ](image)

**Fig. 2-6: Spectrogram functionalities**

As in the section ‘2.2.2.1 FFT Representation’, you can also choose the type of representation of the frequency signal.

### 2.2.4. Constellation graph

This type of graph is only available in the model B because it represents the phase and the quadrature of the signal in the same graph and different axis. In the model A would not make sense because it represents only one dimension and this would cause misunderstandings.

### 2.2.5. ALOE’s graphs

The following graphs are not related to the signals we capture from the receiver elements of the system. These are series of visual representations related to the ALOE framework discussed in **CHAPTER 1**.

Inside the interface, it has reserved a section to incorporate information of the structure of the application.

#### 2.2.5.1. Block Diagram graph

In this case, a graph is not actually plotted. This section represents a ‘.png’ image of the modules which are used by the ALOE application to be executed.

![ Block Diagram example ](image)

**Fig. 2-7: Example ALOE Block diagram**
2.2.5.2. **ALOE Execution Time's graph**

Finally, this graphic representation details the execution time of the modules which are used for the execution of the ALOE application. These modules are the same as those in the previous section.

It updates the time in each Time Slot and gives us an exact idea of the modules that consume more resources of the system.

![Module Time ALOE](image)

**Fig. 2-8:** Example ALOE Execution Time's graph

### 2.3. Functionalities

In this type of software, one of the keys is the interaction of the user with the graph type that is being at any moment. It is very important to represent correctly the signals that are being received. For this, it has implemented some features that give the users a feeling of being completely dominating all the software. The features give a great autonomy so that they can decide what can be shown and, above all, how to display them. In addition, each of them is global, i.e., for all the graphs of the interface, giving freedom movement and maintaining to them when they change from one representation to another.

Next, you will explain each of the features.

#### 2.3.1. **Graph's selector**

Main and most significant functionality in this software is the ability to choose and change on-line the type of graphic at any time of the execution. This can be achieved with the right-side panel, which can choose from a limited variety of type of graphs that will give the user full control of the interface. In addition, it is possible to choose the sampling frequency, the % of memory that is wanted to
maintain in the average, the Time Slots that are wanted to represent in the Spectrogram and number of FFT window which can use to make FFT.

![Graph user interface](image)

**Fig. 2-9:** Options graph selector

### 2.3.2. Screenshot

As its own word indicates, with this functionality you can create a graph screenshot of the exact moment the button is pressed, creating a ‘.png’ image in any folder that is chosen from the system.

Another peculiarity of model B is separate ‘.png’ images will be created (Graph 1 and 2). This is because you do not want always to extract both graphs together for exploitation and the library wants to give to the user the freedom to choose separately.

![Screenshot Button](image)

**Fig. 2-10:** Screenshot Button

### 2.3.3. Drag

This functionality gives the user the ability to drag the graph anywhere, keeping the graph centred on its values.
2.3.4. **Zoom**

With this function, the user can zoom the desired graphic at any time and in two different ways by selecting the button. The first one is to choose the desired area with the mouse, selecting with the left button the first point and dragging it until the second point, creating the rectangular zone to graph. While the second way is to use the mouse scroll.

![Fig. 2-11: Drag Button](image)

2.3.5. **Auto scale**

The user can re-scale the vertical axis and the horizontal axis adapting to the limits of the signal itself at any time by pressing the button.

![Fig. 2-12: Zoom Button](image)

2.3.6. **Legend position**

All graphics must have a legend which indicates what it is plotting. A special feature, that has the legend, is it can be moved in the position that the user wants. By selecting it with the left button and subsequently pressing the right button, the user can choose the position of the legend in 5 different positions: in the upper right, left and center, and the bottom right and left.

![Fig. 2-13: Auto scale Button](image)

![Fig. 2-14: Options legend position](image)
2.3.7. **Axis and Title text**

A very practical functionality is will be mentioned in this section. The title of a graph and the text of the axis label are very important in a graph because they represent what it means. Therefore, it is why you want to give to users the freedom to change it at run time without having to manipulate the source code every time you want to change it.

![Fig. 2-15: Changing Title and Axis Label](image)

2.3.8. **Signal format**

This functionality gives to the users a perception of completely mastering how you wants to represent the signal within the graph. By selecting the signal, firstly, clicking the left button, secondly, users can do three actions on the displayed signal:

a) First one you would be able to change the colour by the different options that exist.

![Fig. 2-16: Colour graph selector](image)

b) The second you would be to change the representation with which it is drawn, i.e., the style of line (linear, impulses, steps or without line between points).

![Fig. 2-17: Linestyle graph selector](image)

c) And finally, you would have the representation of the value in the graph (circle, triangles, crosses, etc).
2.3.9. Background colour

This is a secondary function which gives the user the ability to change the background colour at runtime. Without having anything selected on the graph, clicking the right button and selecting the section ‘Change colour background’, you can get to change the colour among 7 different colours.

2.4. Implementation

To achieve an implementation capable of being able to adapt to any SDR system, it has been done a C++ library. For this, the existing SRSGUI has been started and improved to adapt it to the needs of the project.
Fig. 2-20: Composition of SRSGUI Library

The composition of the new bookcase SRSGUI adapted exclusively for this project is appreciated in the image. This library is nourished by the QT graphic libraries, QCustomPlot and the FFTW3 library.

On the one hand, the QT graphical library is an object-oriented cross-platform framework widely used to develop software programs that use graphical user interface. It is developed as free and open source software and uses the C++ programming language natively.

The plotting library QCustomPlot is a Qt C++ widget\(^1\) for plotting and data visualization. It focuses on making good looking, publication quality 2D plots, graphs and charts, as well as offering high performance for real time visualization applications and producing high quality plots for other media.

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of discrete cosine / sine transforms or DCT / DST). FFTW, which is free software, should be the FFT library of choice for most applications because it has the fastest resolution process in the computing discrete Fourier transform (DFT).

2.4.1. Implementation of the SRSGUI library

This section aims to explain how the SRSGUI static library implementation has been performed in a Linux environment. Here will show how the library has been structured in C++, and how is the workflow followed by the library, explained step by step.

\(^1\) Widgets are the primary elements for creating user interfaces in Qt. Widgets can display data and status information, receive user input, and provide a container for other widgets that should be grouped together. A widget that is not embedded in a parent widget is called a window.
2.4.1.1. Structure SRSGUI

This is a static library which is copied to our system when it is compiled at first and creates an executable. Once you do this, the library main code can be deleted because system has save the executable into the root folder and the program will still work. In order to compile the large number of '.cpp' software codes and their '.h' headers, the Makefiles have been created. Thanks to them, you save time to compile each of the codes from within the software.

Fig. 2-21: SRSGUI Directory Structure

In the previous image, it was possible to appreciate the structure of the files which has been implemented in the library. It has been structured in 4 folders:

- 'src': In this folder are all the '.cpp' computing codes of the library. This folder is structured in two files: 'srsgui.cpp' and 'srsgui++.cpp', and 3 main folders: 'common', 'realplot' and 'complexplot'.
Files 'srsgui.cpp' and 'srsgui++.cpp' contain the functions to create the general application using threats, and they are made first than the initialization of the interfaces. Each is used as they are called from C or C++ code.

The folder 'realplot' and 'complexplot' contain the implementations of the graphic interfaces of the library (model A and B). The elements that make up the graphic part using the QT graphic library, have their headers in the same folder as their '.cpp' codes. This is because they are headers of the graphical part, but they are not of the system in general.

Finally, the 'common' folder contains: the global functions of the graphical interfaces (Event.cpp), the external graphic library (QCustomplot) and the own widget (QCPplot and doubleQCPplot) created to display the signals received and interact with users.

- 'include': This is where most of the library headers reside. Thanks to them, they can interconnect '.cpp' codes and transfer data between them. Also, in these headers is where you create the classes of objects.

The 'srsgui.h' and 'srsgui++.h' headers have the relations between the main code, based on c or c++ and the calling function of the library; and the main functions of the graphical interfaces, for example, initializations of objects or data entry.

- 'images': In this folder reside the images of the buttons that compose the software.

- 'test': Here there are the test programs to see everything works correctly. These have been very useful when to implement and calibrate the graphical interface.

In addition to the structure, also we have implemented a class diagram where we can see how the elements are interconnected with each other. This can be seen in section 'Annex 1: SRSGUI Class Diagram', and gives us an overview of: how the library has been implemented, what variables are used and what functions are implemented in each of the classes.

2.4.1.2. Workflow SRSGUI

It is very important to know what is the workflow of an application because it tells us how each of the interfaces is structured, how they are done, what is the exact order of the applications and, above all, how the information flows. Next, we can see how is the workflow of our GUI application.
In order to explain the whole workflow, it has to start from the beginning, i.e., from the code created by any user who calls the functions of the library to the interaction of the user with the graphical interface. In this case, the implementation has been based on being able to choose a source code based on C or C++. 

In the whole library, each header (.h) has a file (.cpp) which implements the code of its functions, initializes its variables and, also, includes the relations with the other headers of the library.

Inside the main code (code.c and code.cpp), firstly, the SRSGUI library is initialized and then, the functions of different models (Real or Complex). If the main code is in C language, it is necessary a call to plot_real or plot_complex
headers which later will be linked with headers Realplot.h and Complexplot.h. If the main code is in C++ language, directly it can be linked with headers Realplot.h and Complexplot.

As mentioned above, it can be seen in Fig. 2-2 how the C model has the intermediate pass with the functions of the plot_real.h and plot_complex.h header because it is necessary to make a previous step from C to C++ code indicate this ‘c external’. This option allows sharing the code between compilers in binary form and if it is not entered, the compiler will manipulate the information in an erroneous way giving compilation errors and being able to block the application. Once we link to the functions of the 'Realplot' and 'Complexplot' header, the C++ app is created by creating objects from the interfaces.

Firstly, you create an object called 'Wrapper' which encompasses the generic part of the interface and then adds the so-called main Widgets of the application (RealWidget and ComplexWidget). Next, you create the events (files Events) to be able to transfer the information data between the C++ application and the Widgets. For the creation of the main widgets, two extra widgets have been developed: QCPplot and doubleQCPplot.

- QCPplot contains a graphic based on the QCustomPlot library and the basic functionalities explained in ‘2.3 Functionalities’. It is used for the representation of unit graphs as they can be the spectrogram or the constellation.

- doubleQCPplot contains two representations of previous widget to be able to represent two graphs in the same instant of time, and also, to have independence on the functionalities. It is used for the representation of complex signals which have real and imaginary parts.

We used the graphic library extruded QCustomPlot because it is very flexible, very simple, computationally it weighs very little and can adapt to each of the desired functionalities.

Much of the project implementation has been based on the construction of the four Widgets (RealWidget, ComplexWidget, QCPplot and doubleQCPplot) because these represent the whole of the graphical interface. Within each of them has been implemented each and every one of the functionalities and graphs described in sections ‘2.2 Graphs’ and ‘2.3 Functionalities’. To know what functions and objects have been used during the implementation of the library can be seen in the 'Annex 1: SRSGUI - Class diagram'.

2.4.2. Integrated in ALOE

The purpose of this section is to introduce the SRSGUI library into the ALOE middleware. For this, it has been decided to create a special module and adapt it so it can be integrated according to the application which is made.
In ALOE, each module is divided into a process of initialization of the variables and a process of execution and signal processing. This is done to segment the tasks and give all the necessary resources to the execution part.

The ALOE modules which have been created to receive and graph the signal are two:

![ALOE modules diagram](image)

**Fig. 2.23**: ALOE modules

In the module ‘RX Data’ we can put any structure of reception of data and even a system of creation of signal like can be LTE. In the project implementation, this module can become a USRP using an FPGA or the JACK Audio receiver of the laptop itself also a data file.

- In the case of the USRP, the signal is picked up by the Digital Down Converter (DDC), then the signal from the carrier frequency (fo) to the Baseband is transferred. Next, the Analogic Digital Converter is applied to pass the analogic signal to discrete and have it sampled in digital format.

- In the case of JACK, the signal is picked up through the microphones of the laptop, the signal is conditioned and transferred to the Microcontroller which applies the Analogic Digital Converter to pass the analogic signal to discrete and have it sampled in digital format.

- In case of data file, the systems read a block of data and transfer them to the next module.

Finally, we have the module 'GRAPH' which includes the part of the SRSGUI library. It has been decided to create a single module to be able to squeeze to the maximum the functionalities of the ALOE middleware. With ALOE you can get to call as many times as you want this module, i.e., it can be connected to the output of all modules if necessary. This provides great flexibility to the software to be able to visualize that it is being transmitted between modules.


Fig. 2-24: Full ALOE modules

2.5. Detected Problems

Detected problems during the implementation of the library have been several. Firstly, it was detected that in order to be able to implement a C implementation, it was necessary to create too many interconnections between classes in order to avoid any compilation and execution problems.

Secondly, it has had to be created duplicate QCPplot objects for model A and doubleQCPplot objects for model B to produce logarithmic graphs. When attempting to join them to one, the logarithmic axis suffered a processing error and printed the negative axes.

Another problem has been seen in the creation of the complex widget (one graphic for Inphase and another for Quadrature). In this case an extra Widget had to be created in order to plot two QCPplot objects separately on the same interface.

And finally, it was not possible to elaborate a graph in 3D for the spectrogram. Initially, it was considered such graphic would be convenient to implement in 3D, but the external library QCustomPlot does not elaborate this type of graphs. We have not considered using the rest of external libraries because they use to a large number of resources computations that cause the system to crash.
CHAPTER 3. RESULTS

In this chapter we will represent all the results obtained in our implementation tests, both in model A and model B. To structure the information has been considered to divide the chapter into 3 sections. The first referenced to the tests of model A and model B sending a sinusoidal signal, the second referenced to model B representing the voice signal, and finally, the third is referenced to model B representing a LTE signal.

3.1. Graphic representation of a sinusoidal signal

In this first representation of results, a digital sinusoidal periodic signal has been generated so to be able to see how the graphical interface acts on a typical signal in telecommunications:

\[ x[n] = A \cdot \cos\left(2\pi \cdot \left(\frac{f_o}{f_s}\right) \cdot n\right) + j \cdot B \cdot \sin\left(2\pi \cdot \left(\frac{f_o}{f_s}\right) \cdot n\right) \]  \hspace{1cm} (3.1)

In these cases, a test has been made based on the tests of the SRSGUI library, so the ALOE middleware is not used.

3.1.1. Model A

As explained in section ", the peculiarity of model A is the input of signals with values belonging to the real family and it will be given the actions determined by the user. For this reason, a digital sinusoidal signal of 1024 samples was generated with the following main parameters: A = 1, B = 0, fs = 48KHz and fo = 2KHz, with the above formula being as it follows:

\[ x[n] = \cos\left(2\pi \cdot \left(\frac{2000}{48000}\right) \cdot n\right) \]  \hspace{1cm} (3.2)

To see how the graphical interface acts when a Re \{z\} signal is input, a representation of each state will be shown, at the end a global analysis will be done. First, the results will be displayed in the time plane and then shown in the frequency plane.
As can be seen, the graphs agree with the expected results on the time plane. In the images we can see how a temporal signal of 1024 samples is plotted in each frame, it is possible to change in both scales linear and logarithmic, we can represent the time course of the signal in 'n' blocks of 1024 samples and an averaging of the signal.

It should be noted the logarithmic scale always has positive values, so, in the ‘Fig. 3-1’ it is difficult to see the correct function of the graph.
Test and Measurement under SDR framework

The frequency plane will then be displayed.

**Fig. 3-4:** Frequency I/Q Lineal Scale

**Fig. 3-5:** Frequency I/Q Logarithmic Scale

**Fig. 3-6:** (a) FFT signal module lineal. (b) FFT signal module logarithmic. (c) FFT Phase signal (with grey background)

In the previous images we can see how the function is correct. Even having the signal only with the part Re \{z\} of the complete signal, IQ components are generated in the frequency plane. In this plane you can more easily appreciate the correct function of the signal, thanks to the graph type ‘Module and Phase’ where we see two peaks centred at \( f_0 = 2 \text{KHz} \) and \( f_s - f_0 = 46 \text{KH} \). Besides, we have to take in account the correct enclose phase between -PI and PI. Also we
can see the colour functionalities of the graph interface, in colour lines and colour background.

Also, you can see the correct implementation of ‘dB’ function in the followings images. There you can see how the signal takes the Module part and applies the corresponding formula, or $10\log(|x|)$ or $20\log(|x|)$.

\textbf{Fig. 3-7:} Function ‘dB’. Top is $10\log(|x|)$, button is $20\log(|x|)$

To create a frequent signal, temporary signal must be applied to a FFT with a specific samples window. By reducing this window, the signal is losing resolution because it captures fewer samples of the time signal. Now we are going to see the correct execution.

\textbf{Fig. 3-8:} Frequency signal with different FFT Window
Finally, we can see the spectrogram of the received signal. In it you can see how the frequency signal, bounded from 0 to the sampling frequency (X axis), is added each time the signal is input to print it. This graph sweeps bottom-top, starting from the bottom when it reaches the upper limit.

![Spectrogram](image1)

**Fig. 3-9: Spectrogram signal model A**

### 3.1.2. Model B

Unlike the previous model, in model B, complex signals must be introduced and it will be given the actions determined by the user. For this reason, a digital sinusoidal signal of 1024 samples has been generated with the following main parameters: $A = 1$, $B = 1$, $f_s = 48\text{KHz}$ and $f_o = 2\text{KHz}$, with the above formula being as it follows:

$$x[n] = \cos\left(2\pi \left(\frac{2000}{48000}\right) * n\right) + j * \sin\left(2\pi \left(\frac{2000}{48000}\right) * n\right) \quad (3.3)$$

To see how the graphical interface acts before entering a complex signal, a representation of each state will be shown and, at the end, a global analysis will be done. First, the results will be displayed in the time plane and then shown in the frequency plane.

![Time Domain](image2)

**Fig. 3-10: Signal in time domain**
Unlike the previous model we can observe the IQ components in constellation. In addition they can be added the modes of temporal representation of 'n' blocks of 1024 samples and / or effect the average of the signal.

Fig. 3-11: Constellation Signal

The frequency plane will then be displayed.

Fig. 3-12: Signal in Frequency domain

In the previous images we can see how the function is correct. The IQ components are generated in the frequency plane and the module and phase components where we see a peak centred at fo = 2KHz. Unlike the previous model, where only the real part of the signal was sent here we transmitted both the real and the imaginary part. When doing the FFT the peak at fs-fo = 46KH is eliminated due to the cancellation by the complex sinus.
Finally, we can see the spectrogram of the received signal. In it, it is possible to see how all frequency signals, bounded from 0 to the sampling frequency (X axis), are added every time the FFT is joined to the graph. This graph sweeps bottom-top, starting from the bottom when it reaches the upper limit.

### 3.1.3. Performance

In terms of performance, the model A is much more fluid than model B in terms of the representation of the time plane. This is due to the non-inclusion of the imaginary part, which causes the resources to be focused on the graph representing the real part.

Whereas in the other parts, the performance is very similar between the two models. This is because the operation is exactly the same: they have the same graphs and have the same particular and global functionalities.

To see which of the two models is more efficient, it is going to perform a test based on the CPU consumption on an Intel Core I7 computer. This test consists on comparing the consumption between the two modules when we reduce the time between data entry.

<table>
<thead>
<tr>
<th>Time Slot</th>
<th>Model A</th>
<th>Model B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ms</td>
<td>9%</td>
<td>10%</td>
</tr>
<tr>
<td>100us</td>
<td>15%</td>
<td>18%</td>
</tr>
<tr>
<td>10us</td>
<td>50%</td>
<td>54%</td>
</tr>
<tr>
<td>1us</td>
<td>64%</td>
<td>70%</td>
</tr>
</tbody>
</table>

**Table 3-1: Performance**
3.2. **Graphic representation of an audio signal**

In this section we will use the ALOE middleware to receive signals from the laptop microphone. Thanks to one of the interface graph, you can see how the application is composed and what ALOE modules it uses.

![Diagram of ALOE middleware](image)

**Fig. 3-14:** Audio application

Once we execute the ALOE application, we can observe the signal captured by the microphone in the time plane.

![Time Audio with 20 blocks samples](image)

**Fig. 3-15:** Time Audio with 20 blocks samples

Real and imaginary components (IQ) and Module and Phase can be seen in the lineal scale. In addition to applying correctly the averaging and a block chart which it can better appreciate the time travel in real time.

Thanks to the frequency components you can see the good function of the interface. In them you can see as sending a signal at $f_0 = 6\text{KHz}$ and $f_0 = 10\text{KHz}$ are shown perfectly in the graphs.
Fig. 3-16: Frequency plane: real tone 6KHz captured by microphone

Fig. 3-17: Frequency plane: real tone 10KHz captured by microphone

We can also see how the constellation of the signal is executed correctly. It shows a diagonal indicating that the real and imaginary component is inverted.
Thanks to the spectrogram we can see how the frequency signal evolves over time. The next picture shows what happens when a sweep frequency has been send.

3.3. Graphic representation of an LTE signal

In this section will represent LTE signals with BW = 1.92MHz previously captured by a USRP. The values of the signals were stored in a file, which is to be read sequentially through an application executed in the ALOE middleware. Then you are going to show the modules of the application.
Once the application is executed, the results can be observed both in the time plane and in the frequency plane.

**Fig. 3-21:** LTE Signal - Frequency plane – I/Q

**Fig. 3-22:** LTE Signal - Frequency plane from 0 to fs - Module and Phase

**Fig. 3-23:** LTE Signal - Frequency plane from -fs/2 to fs/2 – Module

**Fig. 3-24:** LTE signal – Power
Fig. 3-25: LTE Signal – Spectrogram – from 0 to fs

Fig. 3-26: LTE Signal – Spectrogram – from -fs/2 to fs/2
CHAPTER 4. CONCLUSIONS

As it has been seen in the whole project, in order to implement a good graphic interface, first it is necessary to have a clear objective, in our case to print the signal in multiple conditions, and later to add functionalities that make the user able to master by complete the whole system, in this case graphic colours, graphic selector, etc.

In addition, you have to create a good hierarchical structure of the directories of the library, header and codes so they can be organized in such a way that the system is as clean as effective. Without forgetting the test codes to be able to test all the functions implemented.

Another very important point in this type of project is to be clear the workflow which must follow the data from its beginning to its end. In addition to knowing what steps to take each time, they must know how to do them in the most efficient way without wasting unnecessary resources.

And the most important point to keep in mind in order to achieve a brilliant graphical interface is: to implement such secondary functionalities that make the user feel total control of the system. This gives users the freedom to be able to do and undo as they like, and perform actions on the signals represented. The person who is using it should always keep in mind that the interface adapts to their own needs and have a minimum of prior knowledge of information on the things to be represented at.
CHAPTER 5. FUTURE WORK

As seen in the results obtained, the SRSGUI graphic library is a very powerful tool that gives users flexibility and autonomy to interact with the signal and adapt it to their needs.

It could expand the characteristics of the library and make it even more powerful and versatile. Thus, it is why it is going to list some possible improvements to implement them from now on.

• You could further optimize the code and give the system more processing speed.

• In the spectrogram the following improvements could be made. The first would be to put the time axis to see how it evolves over time. This improvement would link with the next one which would be able to move the spectrum in the temporal axis, something similar to the graph of the blocks. Finally, it could be implemented in 3D thanks to the graphical Qwt library.

• You could implement a filtering mechanism giving users the ability to choose which one to apply. In addition, users could introduce a file with the filter so that the system could read and apply it in real time.

• Also you could improve the graphical interface and put a main menu with different options like 'Help' or load old settings.
BIBLIOGRAPHY

[1] https://www.gnuradio.org/about/
Annex 1: SRSGUI - Class diagram

Fig. Annex - 1-1: SRSGUI Class Diagram