A MIMO MODELING FRAMEWORK USING A SOFTWARE DEFINED RADIO PARADIGM

by

Angel Camelo Vásquez

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

University of Puerto Rico
Mayagüez Campus

2012

Approved by:

Nayda Santiago, Ph.D. Member, Graduate Committee	Date
Kejie Lu, Ph.D. Member, Graduate Committee	Date
Domingo Rodríguez, Ph.D. President, Graduate Committee	Date
Mrs. Isabel Rios, MBA Representative, Graduate Studies	Date
Pedro I. Rivera Vega, Ph.D. Chairperson of the Department	Date

Abstract of Thesis Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of Master of Science

A MIMO MODELING FRAMEWORK USING A SOFTWARE DEFINED RADIO PARADIGM

By

Angel Camelo Vásquez

July 2012

Chair: Domingo Rodríguez
Major Department: Electrical and Computer Engineering

This thesis presents the theory, design, and implementation of a computational framework for modeling MIMO (Multiple Input Multiple Output) systems and simulate them under the *Software Defined Radio* paradigm. Likewise, it presents a set of tools for this type of simulation, such as certain class of time frequency representation tools: Short Time Fourier Transform, Ambiguity Function, Wigner, and Choi-Williams. This thesis also describes interfaces to the computational framework SIRLAB: webSIRLAB and SIRDroid. Finally, modeling examples are presented for the GNUradio software.

Resumen de Tesis Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestro en Ciencias

UN ENTORNO DE TRABAJO PARA MODELAMIENTO MIMO UTILIZANDO EL PARADIGMA DE RADIO DEFINIDO POR SOFTWARE

Por

Angel Camelo Vásquez

Julio 2012

Consejero: Domingo Rodríguez
Departamento: Ingeniería Eléctrica y Computadoras

Esta tesis presenta la teoría, el diseño y la implementación de un simulador para modelamiento de sistemas MIMO (Multiple Input Multiple Output) para ser utilizado bajo el paradigma de *Radio Definido por Software.* Así mismo, presenta un conjunto de herramientas asociadas a este tipo de simulación, como lo son ciertas clases de representaciones en tiempo frecuencia: Short Time Fourier Transform, Ambiguity Function, Wigner y Choi-Williams. Esta tesis también describe algunas interfaces para el marco computacional SIRLAB: webSIRLAB y SIRDroid. Finalmente, se muestran algunos ejemplos que son presentados integrando todo en el software de GNUradio.

Acknowledgments

I want to express my most sincere gratitude and acknowledgement to my advisor, Dr. Domingo Rodriguez for his patient guidance, encouragement and excellent advice throughout the course of study. Also, I am grateful to my thesis committee members, Dr. Nayda Santiago and Dr. Kejie Lu for their review and helpful criticism. Thanks to all the Professors during the three years of study: Dr. Miguel Vélez-Reyes, Dr. Vidya Manian, Dr. Fernando Vega, and Dr. Pedro Vásquez. I also express my gratitude to the Electrical Engineering Department, Puerto Rico Seismic Network, and R&D Staff. Last but not least, I take this opportunity to express my profound gratitude to my family and my friends for their support during this work. Also, to all the people who have been involved directly of indirectly during my studies at Mayaguez, Puerto Rico.

Table of Contents

List of Tables

List of Figures

Abbreviations

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
CPU	Central Processing Unit
CR	Cognitive Radio
DCM	Delay Convolution Modulation
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
EMEDS	Extended Method of Exact Doppler Spread
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPU	Graphic Unit Processing
GRC	GNUradio Companion
IDE	Integrated Development Interface
LCD	Liquid Crystal Display
MCD	Modulation Convolution Delay
MIMO	Multiple Input Multiple Output
MMEA	Modified Method of Equal Areas
MPS	Multipath Simulator
OFDM	Orthogonal Frequency Division Multiplexing
SCTM	Scatter Centre Target Model
SDR	Software Defined Radio
SIMD	Single Instruction Multiple Data
SIRLAB	Signal Representation Laboratory
SISO	Single Input Single Output
SNR	Signal to Noise Ratio
SPMD	Single Program Multiple Data
STFT	Short Time Fourier Transform
USRP	Universal Software Radio Peripheral
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

In a model of communication as shown in Figure 1–1, there are three main parts, the transmitter, receiver and the channel, the latter where there is no control of its operation. It is a limited resource, subject to multiple variables that determine the amount of information that can be transmitted, ie its capacity. One way to improve that capacity is to use a MIMO system. MIMO (Multiple input, Multiple Output) systems are leading research in radar and communications, due to their flexibility and utilization of the channel. A MIMO system uses multiple antennas at both transmission and reception in order to increase data rates. MIMO systems has been implemented in the 3G, WiMAX, and 802.11n standards[1].

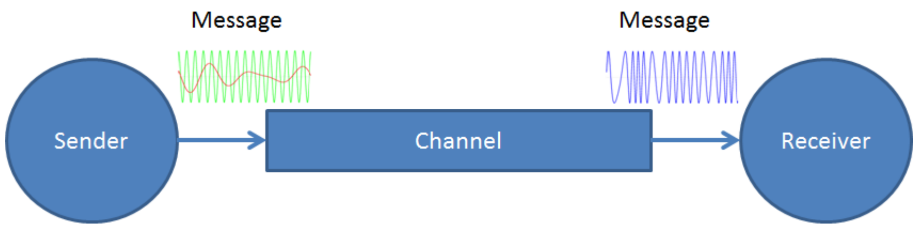


Figure 1–1: Communication Model

Applications in communication and radar systems have been using MIMO technology for a number of years, in communications algorithms such as beamforming, spatial multiplexing, and noise power[2]. In radar systems, MIMO applications have been useful for object detection[3].

On the other hand, Software Defined Radio (SDR) has also been widely used in communication systems in recent years. SDR is defined as a communication system technology where the radio components are implemented in software instead of physical analog devices[4].

The use of SDR includes signal processing applications in coding, modulation, acoustic data transmission, sonar, and others[5]. However, the use of SDR technology for MIMO systems is an emerging trend[6]. A framework for SDR simulations of MIMO systems may be a potential contribution.

One of the most successful implementations of SDR is GNU radio. GNU radio is an open source, extensible, and modular implementation of SDR. GNU Radio uses a WYSIWYG approach to editing the data flow. In order to extend *GNU Radio* to MIMO systems, four basic functions need to be implemented: STFT, DCM channel, MCD channel, and ambiguity function.

A Master's thesis in engineering sciences may include three contributions or axes in their development: A theoretical axis, where new theories are proposed. An application axis, where these theories are implemented; and a technological axis, where technologies are used at the existing time and extended beyond their original capabilities. This thesis aims to contribute in these three areas as follows: firstly, a theoretical contribution is provided by developing and proposing the mathematical formulations for various time-frequency representations, as well as modeling of MIMO channels. Secondly, these formulations are implemented as a software prototype. Finally, existing technologies are improved by extending the DSP development platform and laboratory *GNU Radio* signal representations for SIRLAB beyond their original capabilities.

In the following pages we define the background information of SDR, MIMO, and the proposed extensions to *GNU Radio*.

1.2 Literature Review

The work presented by M. Patzold, et al., deals with the modeling, analysis, and simulation of MIMO forms of mobile-mobile communications [7]. It tackles the problem of how the channel is modeled between mobile-mobile units under the assumption that both the transmitter and the receiver are surrounded by an infinite number of scatterers.

The theory behind this work is centered in the concepts of stochastic and deterministic models. There is a reference model derived from a two-ring scattering model, where both the transmitter and the receiver are moving. It is extended by a stochastic simulation model using only finite numbers of scatterers. The stochastic model is extended into a deterministic simulation model where all the parameters are fixed. Various methods of parameter computation are presented. They include: Extended Method of Exact Doppler Spread (EMEDS) and Modified Method of Equal Areas (MMEA).

This work is considered important for this thesis for the following reason: It provides both stochastic and deterministic models for a MIMO system. These deterministic models are more significant in the context of computational simulation. It also provides a few methods to compute the parameters involved in the simulation.

In the work presented by P. Carlos, et al., the problem addressed is how to determine the effect of indoor environment modeling precision on MIMO channel characterization when dealing with a 3D raytracing with electromagnetic wave propagation [8]. There are specific tools proposed to solve this problem. First, there is a characterization of a MIMO channel using a channel transfer matrix H, a correlation measurement, and a capacity measurement. Two different indoor environments are set and studied. The propagation parameters are set. Finally, the antenna number and spacing are changed in order to detect the variation in characteristic parameters. This work is significant since it establishes a method to perform a channel characterization. Also, a simulation setting would be necessary for it.

In the work described by S.H. Zhou, et al., the main problem is how to detect a target using and statistical approximation [9]. It involves the theory in the field of detection. The detection presents two approach. One approach is the target present hypotheses. The other one is the target absent hypotheses. The target present hypotheses uses the concept that any echo signal will have an statistical distribution. This statistical distribution can be used if, for example, an array of antennas are placed to receive a signal, then those signal would have an statistical correlation showing the presence of the target. This antenna array is a MIMO radar.

The tools used by S.H. Zhou are a mathematical model that comprises the signal model for diversity MIMO radar. The first tool is the Round-Shaped Scatter Centre Target Model (SCTM). A large object can be divided into many small size objects. Another tool is presented as the configuration of a diversity MIMO radar, where, there are M separated stations.

That work is related with our thesis in the sense that there are an approximation in the use of signal and an antenna array to detect objects. Our formulation includes the time-frequency representations where a target is a change into the environment, resulting into a change in the signal representation.

In the work described by Z. Jindong, et al., the problem to solve is how to use the ambiguity function to characterize the range and spatial properties of MIMO radars [10]. To solve this, several tools are presented. First, the ambiguity function is introduced as a fundamental tool for radars. This function is then extended to the MIMO radar. Finally, range and spatial resolutions are obtained.

Another important tool for the work presented by Z. Jindong is the simulation of several waveforms and its corresponding ambiguity functions. The predictions are then validated. This work is related to our thesis as we are considering not only the ambiguity function, but other time-frequency representations.

In the work developed by P. Hallbjorner, et al., the problem is how to make measurements over a multipath simulator [11]. It is a study where such simulator is used to characterize dual antenna performance. The tool presented in this work is a multipath simulator (MPS) comprising 16 antennas that simulates a 2D signal environment. This work presents measurements useful to the reconstruction of a software based simulator for such paths.

The problem addressed by M. Dickens, et al., is how to build a software radio with commercial over-the-shelf components. Such device includes a general purpose processor on a small-form-factor motherboard, radio hardware, touchscreen and LCD, and audio speakers. It also includes an internal battery. The tools used for the proposed solution involve both hardware and software level. The hardware is a single computational piece with the elements mentioned above. At the software level a Linux operating system is used. Both, this thesis and the work done by M. Dickens use *GNU Radio* for the DSP level operations.

L. Garcia Reis, et al., introduce the Software Defined Radio approach [4]. The problem formulation is: what a software defined radio can do?. To solve this, all the characteristics of the SDR are presented. A computational tool is introduced: *GNU Radio* as the main component in a SDR architecture.

This article is important for this thesis because it shows how the *GNU Radio* software is becoming the de-facto software in the modeling and implementation of SDRs; also, its GPL license is compatible with the objectives of this thesis. This explains why we are choosing *GNU Radio* as the development platform for the proposed framework.

1.3 Summary of Following Chapters

This thesis document is presented in the following manner: In Chapter 2, the theoretical background is introduced. It includes mathematical descriptions and formulas used in the investigation. Chapter 3 presents SIRLAB, its current implementations and its extensions for web and Android platforms. Chapter 4 introduces the *Software Radio Paradigm* and how it has been used to build the proposed framework. Conclusions and future work are presented in chapter 5.

Chapter 2

Theoretical Formulation

2.1 Mathematical Preliminaries

2.1.1 Abstract Algebra

First, we introduce some fundamental concepts regarding signal algebra [12]

Definition 1. *A non empty set G is said to be a group if there is a binary operation denoted by $*$, such that it satisfies the following properties:*

1. $a * b \in G, \forall a, b \in G$. *This is the closure property.*
2. $a * (b * c) = (a * b) * c$. *Associative property.*
3. $\exists e \in G$ *such that* $\forall a \in G, a * e = e * a = a$. e *is the identity.*
4. $\forall a \in G, \exists b \in G$ *such that* $a * b = e$. *Inverse existence within the group.*

If also, $\forall a, b \in G$ we have $a * b = b * a$, that group is said to be commutative or abelian.

A set H is a subgroup if $H \subseteq G$ and G is a group.. From this definition, every group G has two trivial subgroups: Identity $\{e\}$, and itself G.

Definition 2. *Let R be a non empty set. R is a ring if there are two binary operations $+$ and $\cdot$, and the following properties are satisfied:*

1. $a + b \in R$. *Closure for* $+$.
2. $a + b = b + a$. $+$ *is commutative.*
3. $(a + b) + c = a + (b + c)$. $+$ *is associative.*
4. $\exists 0 \in R$ *such that* $\forall a \in R, a + 0 = 0 + a = a$. *Zero element.*

5. $\forall a \in R, \exists (-a) \in R$ *such that* $a + (-a) = 0$*. Additive inverse.*

6. $a \cdot b \in R$*. Closure for* $\cdot$*.*

7. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$*. Associative for* $\cdot$*.*

8. $a \cdot (b + c) = a \cdot b + a \cdot c$ *and* $(a + b) \cdot c = a \cdot c + b \cdot c$*. Distributive.*

If also $\exists 1 \in R$ (exists an unique 1 element) such that $\forall a \in R, a \cdot 1 = 1 \cdot a = a$, R is a ring with a unit.

Definition 3. *A field* F *is a commutative ring with a unit such that* $\forall a \in F, \exists b$ *(*b *is unique) such that* $a \cdot b = b \cdot a = 1$*.*

Definition 4. *A non empty set* V *is a vector space over a field* F *if* V *is an abelian group over a binary operation* $+$*, and* $\forall \alpha \in F \forall v, w \in V$*,* $\alpha v \in V$ *is defined with the following properties:*

1. $\alpha(v + w) = \alpha v + \alpha w$*,* $\alpha \in F, v, w \in V$*.*

2. $(\alpha + \beta)v = \alpha v + \beta v$*,* $\alpha, \beta \in F, v \in V$*.*

3. $\alpha(\beta v) = (\alpha\beta)v$*,* $\alpha, \beta \in F, v \in V$*.*

4. $1v = v$*,* $v \in V$*.*

A set W is a subspace if $W \subseteq V$ and W is a vector space

Definition 5. *Let be* V, W *vector spaces over a field* F*. A linear transform or homomorphism from* V *to* W *is a function* $T : V \to W$ *such that:*

1. $T(v_1 + v_2) = T(v_1) + T(v_2)$*,* $\forall v_1, v_2 \in V$*.*

2. $T(\alpha v_1) = \alpha T(v_1)$*,* $\forall v_1 \in V, \forall \alpha \in F$*.*

The set of all the homomorphisms from V to W is symbolized by $\mathrm{Hom}(V, W)$. This set is a vector space with two operations:

1. $(T_1 + T_2)(v) = T_1 v + T_2 v$, $\forall T_1, T_2 \in \mathrm{Hom}(V, W)$ and $v \in V$.
2. $(\alpha T)v = \alpha(T(v))$, $\forall T \in \mathrm{Hom}(V, W)$, $v \in V$ and $\alpha \in F$.

Definition 6. *An algebra over a field F is a vector space V over F with an additional operation called vector operation where:*

$$\begin{aligned} \cdot : \quad V \times V &\rightarrow \quad V \\ (v_0, v_1) &\mapsto v = v_0 \cdot v_1 \end{aligned}$$

Such new operation must satisfy the following properties:

1. $v_0 \cdot (v_1 \cdot v_2) = (v_0 \cdot v_1) \cdot v_2$, $\forall v_0, v_1, v_2 \in V$.
2. $v_0 \cdot (v_1 + v_2) = v_0 \cdot v_1 \cdot v_2$, $\forall v_0, v_1, v_2 \in V$.
3. $(v_0 + v_1) \cdot v_2 = v_0 \cdot v_1 \cdot v_2$, $\forall v_0, v_1, v_2 \in V$.
4. $\alpha(v_0 \cdot v_1) = (\alpha v_0) \cdot v_1 = v_0 \cdot (\alpha v_1)$, $\forall v_0, v_1 \in V; \alpha \in F$.

If $\exists 1 \in V$ such that $1 \cdot v = v \cdot 1 = v, \forall v \in V$, then V is said to be an algebra with unit over F and 1 is called the identity.

An algebra V is commutative if $v_0 \cdot v_1 = v_1 \cdot v_0$.

The set of all the homomorphisms in a vector space $Hom(V, V)$ is an algebra with identity and its operation is the function composition.

Definition 7. *The function composition is defined as*

$$\begin{aligned} \circ : \quad Hom(V, V) \times Hom(V, V) &\rightarrow Hom(V, V) \\ (T_1, T_2) &\mapsto \quad T, \end{aligned}$$

where

$$(T_1 \circ T_2)v = T_1(T_2(v)), T_1, T_2 \in Hom(V, V). \tag{2.1}$$

2.1.2 Signals

Definition 8. *An unidimensional complex finite signal is a function*

$$\begin{aligned} x : \quad \mathbb{Z}_N &\rightarrow \quad \mathbb{C} \\ n &\mapsto x[n]. \end{aligned}$$

If the signal energy is defined as:

$$E(x) = \sum_{n=0}^{N-1} x[n]x^*[n]. \tag{2.2}$$

A finite signal or finite numeric sequence can be seen as a vector:

$$x = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix}.$$

The space of the unidimensional signals is denoted by $l^2(\mathbb{Z}_N)$.

Finally, we characterize the set of the unidimensional signals giving the following statements:

1. $l^2(\mathbb{Z}_N)$ is an abelian group V under the cyclic convolution operation. The identity element for this group is the signal $[1, 0, \ldots, 0]$.
2. $l^2(\mathbb{Z}_N)$ is an abelian group W under the Hadamard product operation. The identity element for this group is the signal $[1, \ldots, 1]$.
3. There is a group homomorphism between the groups V and W. The DFT maps each element in V into an element W.
4. Adding the + operation as usual to the groups described in 1 and 2 makes a ring. Its zero element in both cases is the signal $[0, .., 0]$.
5. The rings obtained in 4 are commutative under the + operation, each one. This results into a field, each one.
6. From 5 one can extend the $l^2(\mathbb{Z}_N)$ set and define it as a vector space over the Field (C). Such vector space can have the cyclic convolution operation, the Hadamard operation, or the usual vector sum operation. The Fourier matrix $F_N \in \mathrm{Hom}(V, W)$.

2.1.3 Kronecker Products

The Kronecker product (or tensor product) properties have been studied deeply. In this work, Kronecker products signal algebra is used to analyze and formulate computational algorithms for time-frequency representations, following the work of J.P. Soto Quiros[13]. We came to the conclusion that the Kronecker product is a powerful tool for parallel algorithm formulations in different platforms: SIMD, vectorial and mixed architectures [14]. The Kronecker product is defined as follows:

Definition 9. *Let A_{n_1,n_2} and B_{m_1,m_2} be two arbitrary matrices of dimension $n_1 \times n_2$ and $m_1 \times m_2$, respectively.*

$$A_{N_1,N_2} = \begin{bmatrix} a_{0,0} & \cdots & a_{0,n_2-1} \\ \vdots & \ddots & \vdots \\ a_{n1-1,0} & \cdots & a_{n_1-1,n_2-1} \end{bmatrix},$$

$$B_{M_1,M_2} = \begin{bmatrix} b_{0,0} & \cdots & b_{0,m_2-1} \\ \vdots & \ddots & \vdots \\ b_{m1-1,0} & \cdots & b_{m_1-1,m_2-1} \end{bmatrix}.$$

The Kronecker product of A and B, $C = (A \otimes B)$ is defined as the $N_1 \times M_1$ by $N_2 \times M_2$ matrix given by

$$C = (A_{N_1,N_2} \otimes B_{M_1,M_2}) = \begin{bmatrix} a_{0,0}B & \cdots & a_{0,n_2-1}B \\ \vdots & \ddots & \vdots \\ a_{n1-1,0}B & \cdots & a_{n_1-1,n_2-1}B \end{bmatrix}.$$

The Kronecker product $A \otimes B$ is computed by substituting any element a_{n_1,n_2} of matrix A with the product $a_{n_1,n_2}B$. So, we can consider the Kronecker product as a matrix decomposition because a very large matrix can be decomposed as the product of two smaller matrices.

The Kronecker product has the following properties:

Associative Property

Let A, B, and C three matrices, then

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C. \tag{2.3}$$

Distributive over matrix multiplication property

Let A, B, and C three matrices, then

$$(A \otimes B)(C \otimes D) = (AC \otimes BD). \tag{2.4}$$

Inversion Property

Let A and B two non-singular matrices, so $C = (A \otimes B)$, then

$$C^{-1} = (A \otimes B)^{-1} = \left(A^{-1} \otimes B^{-1}\right). \tag{2.5}$$

Transposition Property

Let t denote matrix transpose, then

$$(A \otimes B)^{t} = \left(A^{t} \otimes B^{t}\right). \tag{2.6}$$

2.1.4 Discrete Fourier Transform

The Discrete Fourier Transform is defined as the mapping

$$\begin{aligned} \mathrm{DFT}: \quad l^2(\mathbb{Z}_N) &\rightarrow l^2(\mathbb{Z}_N) \\ x &\mapsto X, \end{aligned}$$

such that

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{kn}{N}}, k \in Z_N. \tag{2.7}$$

An algorithm to perform this calculation is described in the **Algorithm** 1

The complexity of this algorithm is given by two nested loops with N iterations each one. This contains the principal multiplication. It gives a complexity of $O(n^2)$.

Algorithm 1 Discrete Fourier Transform

```
Input: Signal $x[n] \in l^2(\mathbb{Z}_N)$.
Output: Signal $X[k] \in l^2(\mathbb{Z}_N)$.
1: $M \leftarrow \lfloor \frac{N}{w} \rfloor$
2: for $k = 0 \to N-1$ do
3:     $X[k] \leftarrow 0$
4:     for $j = 0 \to N-1$ do
5:         $X[k] \leftarrow X[k] + x[n]e^{-j2\pi\frac{kn}{N}}$
6:     end for
7: end for
```

A lower complexity algorithm is given by the FFT algorithm. The most common and fundamental one was discovered by J.W Cooley and John Tukey. The algorithm[15] described in **Algorithm** 2 has an $O(n\log(n))$ complexity. This algorithm requires the length of x be a power of two. However, the speed increase is significantly better. Taking a 1024 length signal, the traditional DFT would take 1048576 multiplications. The FFT would need 20480 multiplications which is a considerable speed-up. For this reason, the FFT is the most important numerical algorithm, given the multiple applications of the DFT. This result is fundamental in the implementation of algorithms discussed in 2.4.

Algorithm 2 Fast Fourier Transform ditfft2(x,N,s)

```
Input: Signal $x[n] \in l^2(\mathbb{Z}_N)$, $N \in \mathbb{Z}$, $s \in \mathbb{Z}$.
Output: Signal $X[k] \in l^2(\mathbb{Z}_N)$.
1: if $N = 1$ then then
2:     $X[0] \leftarrow x[0]$                                  ▷ trivial size-1 DFT base case
3: else
4:     $X[0 : \frac{N}{2} - 1] \leftarrow$ ditfft2($x[n \in 2\mathbb{Z}], \frac{N}{2}, 2s$)   ▷ DFT of $(x[0], x[2s], x[4s], ...)$
5:     $X[\frac{N}{2} : N-1] \leftarrow$ ditfft2($x[n \in 2\mathbb{Z}+1], \frac{N}{2}, 2s$)   ▷ DFT of
   $(x[s], x[s+2s], x[s+4s], ...)$
6:     for $k = 0 \to \frac{N}{2} - 1$ do          ▷ combine DFTs of two halves into full DFT:
7:         $t \leftarrow X[k]$
8:         $X[k] \leftarrow t + e^{-i2\pi\frac{k}{N}}X[k + \frac{N}{2}]$
9:         $X[k + \frac{N}{2}] \leftarrow t - e^{-i2\pi\frac{k}{N}}X[k + \frac{N}{2}]$
10:    end for
11: end if
```

2.2 Definitions

2.2.1 MIMO System

MIMO refers to Multiple Input Multiple Output. It is the ability to have multiple antennas at both the transmission and reception as shown in Figure 2–1 where a 4×4 MIMO system is given as an example. Actually, is a field with great opportunities for research and its results are implemented in today's common products such as 3G cellular, WiMax, and 802.11n.

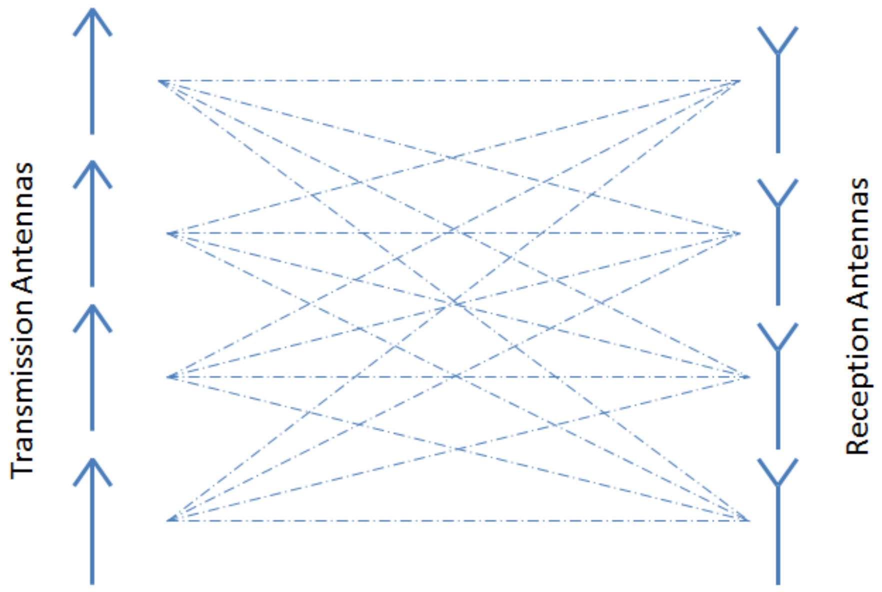


Figure 2–1: 4x4 MIMO Model

Spatial diversity improves performance by using multiple antennas at the reception and combining them optimally. This reduces the noise power in the signal and thus increase the signal to noise ratio (SNR).

In an optimal combiner, signal $y_i(t)$ is modulated as $x_i(t) \in C$, multiplying by predefined weights $w_i \in C$ getting the signal $S_0 \in C$ [16] .

The signal at the reception is composed of the desired signal, noise and interference. If $x_d(t)$ is the desired signal, x_n is the noise, and $x_j(t)$ are the interference signals, then x is

$$x = x_d + x_n + \sum_{j=0}^{L-1} x_j. \tag{2.8}$$

Assuming that both the noise and interference are uncorrelated, then the correlation matrix is:[17]

$$\mathbf{R_{nn}} = \sigma^2\mathbf{I} + \sum_{j=0}^{L-1} E\left[u_j^* \cdot u_j^T\right]. \tag{2.9}$$

Although the ideal goal is to eliminate the noise, in practice, it is only necessary to reduce it enough so the SNR increases and the desired signal can be recovered [18]. In this way, the weights $w_i(t)$ can be obtained in a more simple way using an adaptive algorithm.

In a system with an antenna at the reception and N transmission antennas can be used to suppress an optimal combiner $M-1$ interference. This is assuming an ideal case without noise.

The channel capacity significantly improved using this method in more realistic environments. In a Rayleigh fading transmission system with N simultaneous users, $N-1$ interferers can be canceled with $K+N$ antennas.

Multipath occurs when the signal has multiple propagation paths, that is reflected in the objects in its path. This improves the channel capacity as well. A good example is BLAST, a digital communication system with Rayleigh fading characteristics of simultaneous N users where the receiver knows the channel parameters [19]. If using a $N \times N$ MIMO system, N transmitters and N receivers capacity increases linearly with N. Assuming AWGN (Additive White Gaussian Noise) channel is approximated by the matrix H.

$$H_{mn} = Normal(0, \frac{1}{\sqrt{2}}) + Normal(0, \frac{1}{\sqrt{2}})\jmath, \tag{2.10}$$

where $\jmath$ is the imaginary unit.

Thus, H is distributed Chi-square with two degrees of freedom. The lower bound is

$$C > \sum_{k=1}^{n} log_2\left[1 + \frac{\rho}{n}\chi_{2k}^2\right] b/s/Hz, \tag{2.11}$$

where $\rho = \frac{P}{N}$ is the signal to noise ratio with P the average signal power at the receiver and N the average noise power at the receiver.

As experimental data, a 6×6 MIMO system has a capacity of $300b$/s/Hz.

The simplest model of a MIMO system is $y = Hx + N$. The receiver can not always know the coefficients. If H is diagonalizable then you can get

$$H = U\Sigma V^*, \tag{2.12}$$

so

$$U^{-1}y = \Sigma(V^*x) + N. \tag{2.13}$$

That is, you can preprocess the signal to send and postprocessing after receiving it, knowing in advance some approximation non-singular H.

2.2.2 Modeling Framework

A software framework is an abstraction in which software providing generic functionality can be selectively changed by user code. It is a collection of software libraries providing a defined application programming interface.

Scientific Modeling is the process of generating abstract, conceptual, graphical and mathematical models. This offers a growing collection of methods, techniques and theory about all kinds of specialized scientific modeling.

A modeling framework is a software framework that is intended to be used in scientific modeling. A simulation is the implementation of a model. A simulation brings a model to life and shows how a particular object or phenomenon will behave. Such a simulation can be useful for testing, analysis, or training in those cases where real-world systems or concepts can be represented by models. As every modeling framework is an implementation of a model, then it must be able to perform simulations.

2.2.3 Paradigm

The word paradigm has been used in science to describe a pattern. It is defined as a philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated.

In this context, a paradigm with radars and communications is the analog one. Under this scope, the laws are defined by the laws of electromagnetism. The paradigm used in this thesis uses the laws used in the context of Digital Signal Processing.

2.2.4 Software Defined Radio

Now we explain what is software defined radio[20]. Here is a traditional radio in that it does what it typically makes a traditional radio, send and receive signals. Not like a traditional radio because several components that were previously done in hardware, are now computationally implemented in software. In conclusion, the software defined radio is to bring the software closer to the antenna.

A traditional radio system comprises both a transmitter and a receiver. The stages in the transmission and reception are as follows. First the source coding / decoding, channel coding then / decoding, after baseband modulation / demodulation and finally the upconverter / downconverter. In a traditional radio, software reaches the source decoding. The next steps are implemented in hardware. With software-defined radio, software has been extended to the baseband modulator, providing great flexibility in the implementation of these two intermediate stages as shown in Figure 2–2. It is in this domain where monitoring is particularly important channel, that is where we need a time frequency representation.

It then noted that SDR is possible not only to the capacity of current computers to achieve the speed required to perform these tasks, but also the possibility to implement in an ASIC or a FPGA that code. So from the programming, we can

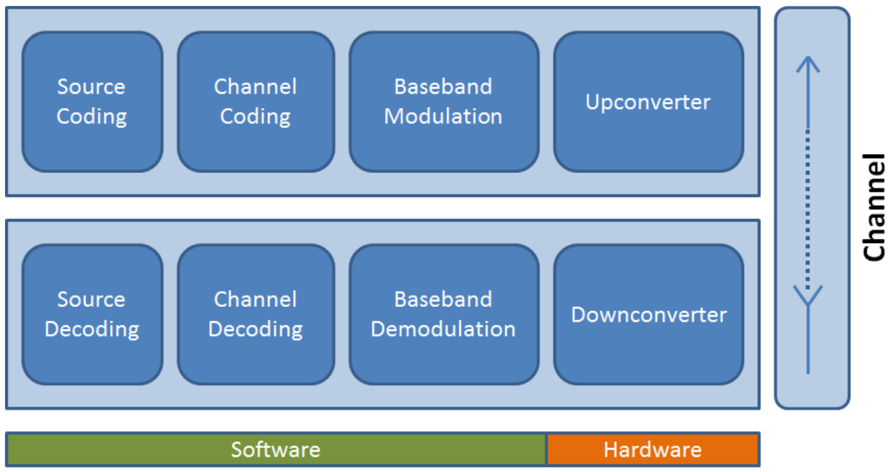


Figure 2–2: Software Defined Radio Paradigm

implement the software defined radios in those devices. The radio devices that have been implemented range from AM receivers to digital communication systems and radar systems [5].

A list of popular SDR products is described in Table 2–1. The decision to choose *GNU Radio* was because its flexibility in terms of licensing. It is subject to a GPL license where we could access to the code and learn and modify it as needed. Without any affiliation with an external company, we found this license compatible with the objectives of this thesis, as it was required to share and redistribute our results.

The support among the GNU community is very good and we found solutions to the problems we encountered relatively quick as we advanced with the project. Finally, the compatibility offered with the USRP (Universal Software Radio Peripheral) bring us the possibility to interact with electromagnetic signals.

Software	Language	License
GNU Radio	Python, C++	GPL
SoRa	C++	EULA
Simulink	Matlab	EULA

Table 2–1: Software Defined Radio

2.3 Parallel Computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously [21]. These calculations can be performed in different processor circuitry, different cores, or even different computation units separated across the world. This range of granularity levels requires mechanisms for interprocess communications. According to [22], there are 4 types of parallel computers:

1. Shared Memory Multiprocessor System: this computer consists of several processors sharing a common memory address space. Using this approach, a dataset can be loaded into a memory segment and request each processor to compute over a fraction of that data. Depending on the operating system, there can be threads which are light processes sharing a large amount of data. The modern operating systems can manage such light processes and assign them to a set of processors to be executed.
2. Message Passing Multicomputer: This multicomputer is created by connecting several different computers by an interconnection network. The interconnection network is responsible for sending and receiving the data and results for each computer in the network. A process can be divided into multiple subprocesses with its corresponding amount of data and send it to a computer in the network. A limiting issue for this approach is the speed of the network, specially if there is a high amount of data involved.
3. Distributed Shared Memory: this is a combination of the last two types. It is a shared memory multiprocessor system in the way of a single address space where each process can access. It is a message passing multicomputer where the memory and the datasets have to be sent from its location to the processor requiring it.
4. SIMD and MIMD: Single and Multiple Instruction, Multiple Data is an early approach to parallel large data sets where n elements are loaded into a single

vector and then applied an instruction at once. The circuitry of the modern processors permits do that using large registers and wide memory buses. Many assembly instructions are provided to achieve this at user level.

The speedup factor of a process executed in parallel is defined as:

$$S(n) = \frac{t_s}{t_p}. \tag{2.14}$$

Where t_s is the execution time using one processor and t_p is the execution time using a multiprocessor with n processors. From this equation one can evaluate the maximum speed up assuming an ideal and completely parallelizable algorithm. Then such speed is:

$$S(n) = \frac{t_s}{t_s/n} = n. \tag{2.15}$$

However, a normal process can not be parallelized to this limit, this is due to:

1. Time where the processor is not doing useful work to the algorithm.
2. Communication latency with memory or network passing data.
3. Segment codes not parallelizable
4. Extra work on each thread

The efficiency is a measure that indicates how well the algorithm is being parallelized, comparing it to the ideal case. The maximum efficiency is 100% for the ideal case. It is defined as:

$$E = \frac{S(n)}{n} \times 100\%. \tag{2.16}$$

This efficiency is measured empirically. An analytical way to calculate the speed up factor and the efficiency was given by Gene Amdahl in its famous law, defined as:

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} = \frac{n}{1+(n-1)f}, \tag{2.17}$$

where $f \in [0,1]$ is the fraction that can not be parallelized into concurrent tasks. This imposes a limit to the speed up factor with infinite processors:

$$\lim_{n \to \infty} S(n) = \frac{1}{f}. \tag{2.18}$$

2.3.1 Parallel Programming Software

Several programming languages and Application Programming Interfaces (API) have been released to perform parallel computing. A representative list include, but is not limited to: MPI, OpenMP, OpenCL and MOSIX. Message Passing Interface is a standardized and portable message passing system to be used in a Message Passing Multicomputer[23], it is used in a wide range of scientific computing application, such as weather simulation [24] . The standard defines the syntax and semantics of a core of library routines useful for write programs in Fortran or C. OpenMP is an application programing interface to write programs in a shared memory multiprocessor[25]. It consists of several compiler directives, library routines and environmental variables. OpenMP is widely supported by the majority of C, C++ and Fortran compilers such as gcc and Intel. Another API for parallel programming is OpenCL. It is an extension of OpenGL (A standard for computer graphics) oriented to take advantage of the SIMD and MIMD capabilities offered by graphic cards[26]. It is cross platform and is supported by the commercial GPU hardware providers. MOSIX is a UNIX kernel extension to build a distributed shared memory computers [27]. For purposes of this thesis, the API selected for parallelizing algorithms is OpenMP. This selection was done due to the cross compatibility offered by its developers, great support among the GNU users and easiness to program a parallel algorithm with a few directive lines.

2.3.2 Kuck's Model

Kuck's model [28], as seen in Figure 2–3 is a machine model which shows hierarchical processing elements, local memory, shared memory, and its network paths into a single image. Its parameters are:

- P_0^i: Processor i Speed.
- M_0^i: Processor's i Memory Amount.
- SMN$_1$: Shared Memory 1 Speed i Speed.
- SM$_i$: Shared Memory i Amount.
- $N_{0.5}^0$: Interprocessor Network Speed.

This model enables us to compare analytically different architectures and make predictions about the efficiency and speed of such architectures [29].

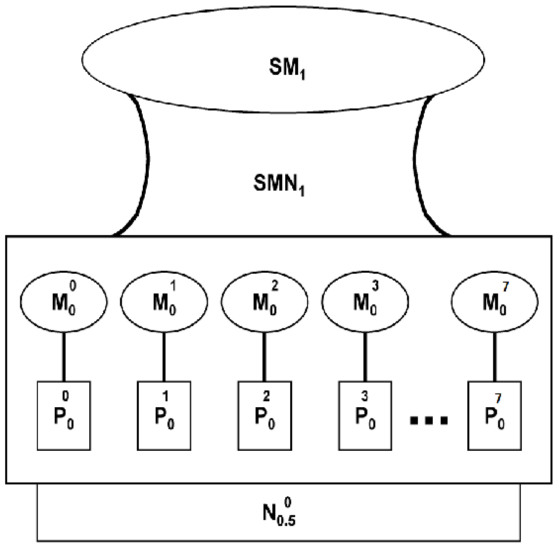


Figure 2–3: Kuck's Model

2.3.3 Kronecker Products in DSP Parallel Algorithms

One of the reasons for using Kronecker product in DSP algorithms formulation is the fact there is a clear relation between the structural distribution of a Kronecker matrix and the viability of a very efficient algorithmic implementation in terms of a specific computational architecture.

Let C_n a Kronecker matrix given by:

$$C_n = (I_s \otimes B_r) = \begin{bmatrix} B_r & & & & \\ & B_r & & & \\ & & B_r & & \\ & & & \ddots & \\ & & & & B_r \end{bmatrix},$$

where I_s is the identity matrix of order s.

In most DSP algorithms (including the matrix DFT implementations) is necessary to compute some matrix multiplications, for instance, $Y_n = (C_n)\, X_n$, where X_n is the input data vector. As C_n has a special structure, making a multiplication in regular form would be inefficient. It would be better to take advantage of the symmetry of $(I_s \otimes B_r)$ by splitting the vector X_n into s subvectors x_i of length r and performing the matrix product $(B_r)\, x_i$ on each case. So, for

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } B_2 = \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix},$$

then, $Y_6 = C_6 X_6 = (I_3 \otimes B_2)\, X_6$:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} & 0 & 0 & 0 & 0 \\ b_{1,0} & b_{1,1} & 0 & 0 & 0 & 0 \\ 0 & 0 & b_{0,0} & b_{0,1} & 0 & 0 \\ 0 & 0 & b_{1,0} & b_{1,1} & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{0,0} & b_{0,1} \\ 0 & 0 & 0 & 0 & b_{1,0} & b_{1,1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

If we think in the Single Program Multiple Data (SPMD) paradigm, easily we can see the relationship among this approach and the referred paradigm. This product can be performed in three independent processors at the same time. Splitting the input vector X_6 into 3 sub-vectors of length 2 and performing the matrix product $(B_2)\,X_2$ on each processor. So,

In Processor 0:

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix},$$

in Processor 1:

$$\begin{pmatrix} y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix},$$

in Processor 2:

$$\begin{pmatrix} y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix} \begin{pmatrix} x_4 \\ x_5 \end{pmatrix}.$$

This way, the Kronecker matrices of the form $I_s \otimes B_r$ are associated with the SPMD algorithm implementations. This approach would need s processors ($Processor_0$, $Processor_1 \cdots Processor_{s-1}$) and each processor would process a sub-vector of X with r elements.

2.4 Time-Frequency Representations

2.4.1 Short Time Fourier Transform Distribution

The STFT is a continuous function [30] S:

$$\begin{array}{rcl} S: \; L^2(\mathbb{R}) \times L^2(\mathbb{R}) & \rightarrow & L^2(\mathbb{R}^2) \\ (x, w) & \mapsto & S_{x,w}, \end{array}$$

such that:

$$S_{x,w}(t,w) = \int x(t)w(t-\tau)e^{-j2\pi wt}dt. \tag{2.19}$$

Its definition for the discrete case is

$$\begin{aligned} S: \quad l^2(\mathbb{Z}_N) \times l^2(\mathbb{Z}_M) &\rightarrow l^2(\mathbb{Z}_N \times \mathbb{Z}_M) \\ (x,w) &\mapsto S_{x,w}, \end{aligned}$$

such that:

$$S_{x,w}[k,m] = \sum_{n \in \mathbb{Z}_{\mathbb{N}}} x[n]w[n-m]e^{-j2\pi\frac{kn}{N}}. \tag{2.20}$$

Now, the Cyclic Short Time Fourier Transform (CSTFT) is defined as the mapping:

$$\begin{aligned} S: \quad l^2(\mathbb{R}) \times l^2(\mathbb{Z}_{\mathbb{N}}) &\rightarrow l^2({\mathbb{Z}_{\mathbb{N}}}^2) \\ (x,w) &\mapsto S_{x,w}, \end{aligned}$$

such that:

$$S_{x,w}[k,m] = \sum_{n=0}^{N-1} x[n]w[\langle n-mR\rangle_N]e^{-j2\pi\frac{kn}{N}}, \tag{2.21}$$

where $R \in \mathbb{N}$ is the displacement constant. The parameter m is limited by $m \leqslant P = \lceil\frac{N}{R}\rceil$. The algorithm is described in the **Algorithm** 3.

Algorithm 3 Short Time Fourier Transform

```
Input:  Signal $x[n] \in l^2(\mathbb{Z}_N)$, Window width $w > 0 \in \mathbb{Z}$, Zero-padding $P > 0 \in \mathbb{Z}$.
Output:  Matrix $s[M][P]$
1: $M \leftarrow \lfloor\frac{N}{w}\rfloor$
2: for $i = 0 \rightarrow M-1$ do
3:     $wn \in \mathbb{Z}^N \leftarrow 0$
4:     $y \in \mathbb{Z}^P \leftarrow 0$
5:     for $j = 0 \rightarrow w-1$ do
6:         $wn[j + w \cdot i] \leftarrow 1$
7:     end for
8:     for $j = 0 \rightarrow N-1$ do
9:         $y[j] \leftarrow x[j] \cdot wn[j]$
10:    end for
11:    $s[i] \leftarrow \mathrm{dft}(y)$
12: end for
```

To evaluate the complexity of this algorithm, the number of cycles on the outer loop is a fraction of N. There is a number of N multiplications in the second inner loop. This algorithm takes advantage of the FFT algorithm to calculate the DFT which has a complexity of $O(n\log(n))$. Taking this into consideration, this algorithm has a complexity of $O(n^2\log(n))$. The outer loop is parallelizable.

2.4.2 Ambiguity Function Distribution

The Ambiguity function is a mapping [31]

$$\begin{aligned} A: \quad L^2(\mathbb{R}) \times L^2(\mathbb{R}) \quad &\rightarrow \quad L^2(\mathbb{R}^2) \\ (f,g) \quad &\mapsto \quad A_{f,g}, \end{aligned}$$

such that:

$$A_{f,g}(\tau, f) = \int f(t) g^*(t-\tau) e^{-j2\pi f t} dt. \tag{2.22}$$

The discrete Ambiguity transform is the map

$$\begin{aligned} A: \quad l^2(Z_N) \times l^2(Z_N) \quad &\rightarrow \quad l^2(Z_N \times Z_N) \\ (f,g) \quad &\mapsto \quad A_{f,g}, \end{aligned}$$

where,

$$A_{f,g}[m,k] = \sum_{n \in Z_N} f[n]\, g^*[\langle n+m \rangle_N]\, e^{-j2\pi \frac{kn}{N}}. \tag{2.23}$$

Where $f[t]$ is the broadcast signal, $g[t]$ is the return signal, $g^*[t]$ the complex conjugate of the return signal, m is the delay between the broadcasting of the signal and the returning time that the signal is detected at the receiver, k represents the Doppler frequency, $\langle n+m \rangle_N$ is the module N operation of $n+m$ addition and Z_N is an integer group in the interval $[0, N-1]$.

Then, if we take into consideration the Kronecker product properties, we can write the Ambiguity Function $A'_{x,y} \in \mathbb{C}^{N^2 \times 1}$ as:

$$A'_{f,g} = [I_N \otimes (P \odot F_N^*)]\, v, \tag{2.24}$$

where $\odot$ is the Hadamard product, and $P \in \mathbb{C}^{N\times N}$:

$$P[\tau, v] = \begin{cases} e^{j2\pi\langle mvt\rangle_N} & N \text{ Odd}, 2m \equiv 1 \text{mod} N \\ e^{j2\pi\langle vt\rangle_N} & N \text{ Even}, \langle vt\rangle_N < \frac{N}{2} \\ e^{j2\pi(\langle vt\rangle_N - N)} & N \text{ Even}, \langle vt\rangle_N \geqslant \frac{N}{2}, \end{cases} \tag{2.25}$$

with $v \in \mathbb{C}^{N^2\times 1}$

$$\begin{bmatrix} v_0 \\ \vdots \\ v_{N-1} \end{bmatrix}, \tag{2.26}$$

and $v_i = [y[0], y[1], \ldots, y[N-1]]^T$ such that $y[n] = x[\langle n+j\rangle_N]x^*[n]$ for $j, n \in \mathbb{Z}_N$.

Calculating the Kronecker product we have:

$$A'_{x,y} = \begin{bmatrix} (P \odot F_N^*)v_0 \\ \vdots \\ (P \odot F_N^*)v_{N-1} \end{bmatrix}. \tag{2.27}$$

Reorganizing into $A_{x,y} \in \mathbb{C}^{N\times N}$ such that

$$A_{x,y} = [(P \odot F_N^*)v_0 \mid (P \odot F_N^*)v_1 \mid \ldots \mid (P \odot F_N^*)v_{N-1}]. \tag{2.28}$$

The algorithm to perform this function is described on **Algorithm** 4.

Algorithm 4 Ambiguity Function

```
Input:  Signal $x[n] \in l^2(\mathbb{Z}_N)$, $y[n] \in l^2(\mathbb{Z}_N)$
Output:  Matrix $A[N][N]$
1: for $i = 0 \to N-1$ do
2:     for $j = 0 \to N-1$ do
3:         $y1[j] \leftarrow \text{conj}(y[\text{mod}(j+i, N)])$
4:     end for
5:     for $j = 0 \to N-1$ do
6:         $z[j] \leftarrow x[j] \cdot y1[j]$
7:     end for
8:     $A[i] \leftarrow \text{dft}(z)$
9: end for
```

To evaluate the complexity of this algorithm, the number of cycles on the outer loop is N. There is a number of N multiplications in the inner loops. This algorithm takes advantage of the FFT algorithm to calculate the DFT which has a complexity of $O(n\log(n)$. Taking this into consideration, this algorithm has a complexity of $O(n^2\log(n))$. The outer loop is parallelizable.

2.4.3 Wigner Distribution

The discrete Wigner distribution is a mapping [32]

$$\begin{array}{rccc} W_x : & L^2(\mathbb{Z}_N) & \to & l^2(\mathbb{Z}_N \times \mathbb{Z}_N) \\ & (x) & \mapsto & W_x, \end{array}$$

such that:

$$W_x[n,k] = \frac{1}{N} \sum_{\tau=0}^{N-1} \sum_{v=0}^{N-1} \sum_{l=0}^{N-1} \rho_N e^{j2\pi vl} x[\langle l+\tau \rangle_N] x^*[l] e^{-j\frac{2\pi}{N}(nv+k\tau)}. \tag{2.29}$$

We can redefine the Wigner distribution as a linear operator W_x. First, we define the matrix $W'_x \in \mathbb{C}^{N^2\times 1}$ such that

$$\begin{aligned} W'_x &= \tfrac{1}{N}(F_N \otimes F_N) A'_x \\ &= \tfrac{1}{N}(F_N \otimes F_N)[I_N \otimes (P \circ F_N^*)]v. \end{aligned} \tag{2.30}$$

Resolving the Kronecker expression

$$W'_x = \frac{1}{N} \begin{bmatrix} \mathscr{F}_0 A'_x \\ \dots \\ \mathscr{F}_0 A'_x \end{bmatrix}, \tag{2.31}$$

where $\mathscr{F}_m \in \mathbb{C}^{N\times N^2}$

$$\mathscr{F}_m = [w_N^{m\cdot 0} \cdot F_N \mid w_N^{m\cdot 1} \cdot F_N \mid \dots \mid w_N^{m\cdot (N-1)} \cdot F_N], \tag{2.32}$$

with $w_N = e^{-j\frac{2\pi}{N}}$ and $m \in \mathbb{Z}_N$.

Finally, we get the matrix representation for the Wigner distribution

$$W_x = \frac{1}{N}[\mathscr{F}_0 A'_x \mid \mathscr{F}_1 A'_x \mid \ldots \mid \mathscr{F}_{N-1} A'_x]. \tag{2.33}$$

Algorithm 5 describes a procedure to obtain the Wigner distribution

Algorithm 5 Wigner

```
Input:  Signal $x[n] \in l^2(\mathbb{Z}_N)$, $y[n] \in l^2(\mathbb{Z}_N)$
Output:  Matrix $W[N][N]$
 1: for $i = 0 \to N-1$ do
 2:     for $j = 0 \to N-1$ do
 3:         $y1[j] \leftarrow \mathrm{conj}(y[\mathrm{mod}(j+i, N)])$
 4:     end for
 5:     for $j = 0 \to N-1$ do
 6:         $z[j] \leftarrow x[j] \cdot y1[j]$
 7:     end for
 8:     $A[i] \leftarrow \mathrm{dft}(z)$
 9: end for
10: $W1 \leftarrow \mathrm{dft2}(A)$
11: $W \leftarrow W1^T$
```

To evaluate the complexity of this algorithm, the number of cycles on the outer loop is N. There is a number of N operations in the inner loops. This algorithm takes advantage of the FFT algorithm to calculate the DFT which has a complexity of $O(n\log(n)$. Finally, the bidimensional FFT has a complexity $O(n^2\log(n)))$. Taking this into consideration, this algorithm has a complexity of $O(n^2\log(n))$. The outer loop is parallelizable.

2.4.4 Choi-Williams Distribution

The Choi-Williams function is defined as the mapping [33]

$$\begin{array}{rccc} C_x: & L^2(\mathbb{R}) & \to & L^2(\mathbb{R}^2) \\ & (x) & \mapsto & C_x, \end{array}$$

such that:

$$C_x(t,f) = \int_{-\infty}^{\infty} A_x(\eta,\tau)\Phi(\eta,\tau)e^{j2\pi(\eta t-\tau f)}d\eta d\tau, \tag{2.34}$$

where the kernel is defined as:

$$\Phi(\eta, \tau) = e^{-\alpha(\eta\tau)^2}, \alpha \in \mathbb{C}. \tag{2.35}$$

Its definition for the discrete case is a mapping

$$\begin{array}{rccc} C_x : & l^2(\mathbb{Z}_N) & \rightarrow & l^2(\mathbb{Z}_N \times \mathbb{Z}_N) \\ & (x) & \mapsto & C_x, \end{array}$$

such that:

$$C_x[t, f] = \sum_{m=0}^{N-1} \sum_{k=0}^{N-1} A_x[m, k]\Phi[m, k]e^{j\frac{2\pi}{N}(kt-mf)}, \tag{2.36}$$

where the kernel is defined as:

$$\Phi[m, k] = e^{-\alpha(mk)^2}. \tag{2.37}$$

Its matrix representation is a matrix $C_x \in \mathbb{C}^{N^2 \times N^2}$ such that:

$$C_x = [\mathscr{F}_0(A_x' \odot \Phi) \mid \ldots \mid \mathscr{F}_{N-1}(A_x' \odot \Phi)], \tag{2.38}$$

where $\Phi \in \mathbb{C}^{N^2 \times 1}$, $\Phi_m[k] = \Phi[m, k]$,

$$\Phi = \begin{bmatrix} \phi_0 \\ \vdots \\ \phi_{N-1} \end{bmatrix}, \tag{2.39}$$

and $\mathscr{F}_l \in \mathbb{C}^{N \times N^2}$ such that

$$\mathscr{F}_l = \left[w_N^{l \cdot 0} F_N \mid \ldots \mid w_N^{l \cdot (N-1)} F_N \right]. \tag{2.40}$$

Algorithm 6 summarizes the procedure to obtain the Choi-Williams distribution.

To evaluate the complexity of this algorithm, the number of cycles on the outer loop is N. There is a number of N operations in the inner loops. This algorithm

Algorithm 6 Choi-Williams

```
Input:  Signal $x[n] \in l^2(\mathbb{Z}_N)$, $y[n] \in l^2(\mathbb{Z}_N)$, $\alpha$
Output:  Matrix $W[N][N]$
 1: for $i = 0 \to N-1$ do
 2:     for $j = 0 \to N-1$ do
 3:         $y1[j] \leftarrow y[\text{mod}(j+i, N)]$
 4:     end for
 5:     for $j = 0 \to N-1$ do
 6:         $z[j] \leftarrow x[j] \cdot \text{conj}(y1[j])$
 7:     end for
 8:     $A[i] \leftarrow \text{dft}(z)$
 9:     for $j = 0 \to N-1$ do
10:         $A[i][j] \leftarrow A[i][j] \cdot e^{-\alpha(i \cdot j)^2}$
11:     end for
12: end for
13: $W1 \leftarrow \text{dft2}(A)$
14: $W \leftarrow W1^T$
```

takes advantage of the FFT algorithm to calculate the DFT which has a complexity of $O(n\log(n)$. The conjugate operations is a $O(n^2)$ algorithm. Finally, the bidimensional FFT has a complexity $O(n^2\log(n)))$. Taking this into consideration, this algorithm has a complexity of $O(n^2\log(n))$. The outer loop is parallelizable.

2.5 MIMO Channel Modeling

Let be M the number of input antennas, N the number of output antennas. A MIMO channel may be modeled as a linear system

$$y = Hx, \tag{2.41}$$

where $x = [x_0 \mid x_1 \mid \ldots \mid x_{M-1}]$, $y = [y_0 \mid y_1 \mid \ldots \mid y_{N-1}]$; $x_i, y_j \in l^2(\mathbb{Z}_D)$. Using this approach, every pair transmitter/receiver is modeled as a single operator. Thus, the output at each antenna is the sum:

$$y_a = \sum_{b=0}^{M-1} H_{a,b}(x_b). \tag{2.42}$$

The operator $H_{a,b}$ is modeled as a SISO (Single Input Single Output). Two SISO models are presented as a operator composition. Each model has 3 stages: Modulation, Convolution, and Delay.

Assuming that: $x_b(t), t \in \mathbb{R}, b \in \mathbb{Z}_M$ is a continuous signal and a digital signal $x_b(n \cdot T_s) \to x_b[n], n \in \mathbb{Z}_D$, $T_s = 1$ and $D \gg M \times N$ our two models are:

2.5.1 Modulation-Convolution-Delay (MCD)

The first model is the MCD model whose result is a signal $y_{a,b} \in l^2(\mathbb{Z}_D)$:

$$y_{a,b} = g_{a,b} \circledast_D \delta_{m_{a,b}}, \tag{2.43}$$

where $g_{a,b} \in l^2(\mathbb{Z}_D)$

$$g_{a,b} = T_{h_{a,b}}\{f_{a,b}\}, \tag{2.44}$$

where $f_{a,b} \in l^2(\mathbb{Z}_D)$

$$f_{a,b} = x_b \odot_D X_{k_{a,b}}. \tag{2.45}$$

Composing these 3 equations give us the following equation

$$y_{a,b} = (T_{h_{a,b}}\{x_b \odot_D X_{k_{a,b}}\}) \circledast_D \delta_{m_{a,b}}. \tag{2.46}$$

2.5.2 Delay-Convolution-Modulation (DCM)

The second model is the DCM model whose result is a signal $y_{a,b} \in l^2(\mathbb{Z}_D)$:

$$y_{a,b} = g_{a,b} \odot_D X_{k_{a,b}}, \tag{2.47}$$

where $g_{a,b} \in l^2(\mathbb{Z}_D)$

$$g_{a,b} = T_{h_{a,b}}\{f_{a,b}\}, \tag{2.48}$$

where $f_{a,b} \in l^2(\mathbb{Z}_D)$

$$f_{a,b} = x_b \circledast_D \delta_{m_{a,b}}. \tag{2.49}$$

Composing these 3 equations give us the following equation

$$y_{a,b} = (T_{h_{a,b}}\{x_b \circledast_D \delta_{m_{a,b}}\}) \odot_D X_{k_{a,b}}. \tag{2.50}$$

Finally, we have two general equations for a MIMO system:

$$y_a = \sum_{b=0}^{M-1} (T_{h_{a,b}}\{x_b \odot_D X_{k_{a,b}}\}) \circledast_D \delta_{m_{a,b}}, \tag{2.51}$$

$$y_a = \sum_{b=0}^{M-1} (T_{h_{a,b}}\{x_b \circledast_D \delta_{m_{a,b}}\}) \odot_D X_{k_{a,b}}. \tag{2.52}$$

Where 2.51 is the MCD model received at the antenna a, and 2.52 is the DCM model received at the antenna a.

Chapter 3

Time-Frequency Signal Representation Tools

Several Time-Frequency toolboxes were considered. Its main characteristics are shown in Table 3–1:

Software	Developer	Language
TFTB	Rice Univ	Matlab, Octave
TFSA	Univ of Queensland	Matlab
DiscreteTFDs	Jeff O' Neill	Matlab
SIRLAB	AIPLAB	C

Table 3–1: Time-Frequency Toolboxes

Several criteria were used to select the main Time Frequency Representation tools:

1. Speed
2. GNU/Linux compatible
3. Free source
4. Documentation

SIRLAB was chosen as the main Time-Frequency Signal Representation tool. It is written in C which is faster than Matlab code generally speaking. Also is compatible with GNU/Linux, is open source and well documented. Other approaches only works with Matlab. The toolbox from Rice University will be used for benchmarking and comparison.

SIRLAB gives a skeleton to the development and testing of new algorithms for signal processing operators. It was developed at the AIPLAB at the University of Puerto Rico, Mayaguez campus. It is specially suitable for the development of signal representation algorithms such as the Short Time Fourier Transform, its main example. Seen as a framework, SIRLAB provides a tool to use time-frequency operators in other works at the AIPLAB. Such tool is implemented as various functions for the input - processing - output application of an algorithm. That functions are grouped into several libraries:

1. Inputlib: reads parameters from a parameters file and from a .wav file
2. Processlib: signal management .wav segment reads, zeropadding and colormap
3. Aritmetlib: basic arithmetic operations, Hadamard, normalization and maximum
4. Outputlib: Graphical elements for output frames.

SIRLAB is based in OpenCV and FFTW open source libraries. As seen on the Figure 3–1 the architecture is very simple:

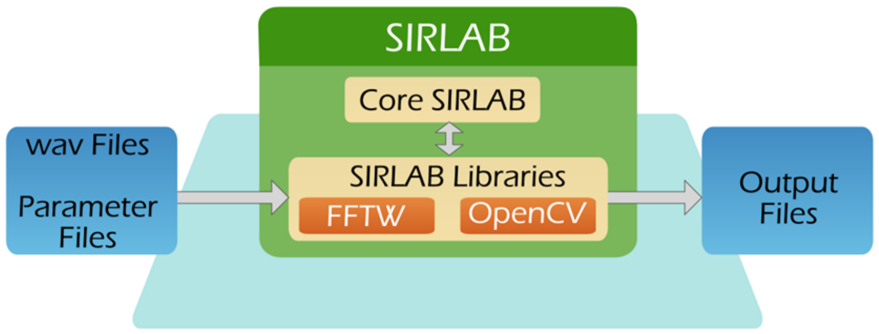


Figure 3–1: SIRLAB Architecture

1. A wav file and a parameters file are the input to SIRLAB
2. SIRLAB process the wav file according to the received parameters and using its own libraries, FFTW, and OpenCV.
3. Finally, produces a set of output files, which are stored in local storage.

A typical implementation with SIRLAB is the computation of the Short Time Fourier Transform and the Ambiguity Function. These functions are written in C. However, several challenges with SIRLAB are:

- Hard to install: it requires programming and compilation abilities. In many systems there is not all the libraries installed. An entry level user would avoid the technical effort involved with the installation.
- Powerful machine: the computational requirement for time frequency analysis turns SIRLAB not suitable for netbooks, smartphones, and tablets.
- Hard to test: creation and modification of a parameter file is not as straightforward as it seems.

Keeping all this in mind we propose a set of interfaces for SIRLAB, they are webSIRLAB and SIRDroid.

3.1 webSIRLAB

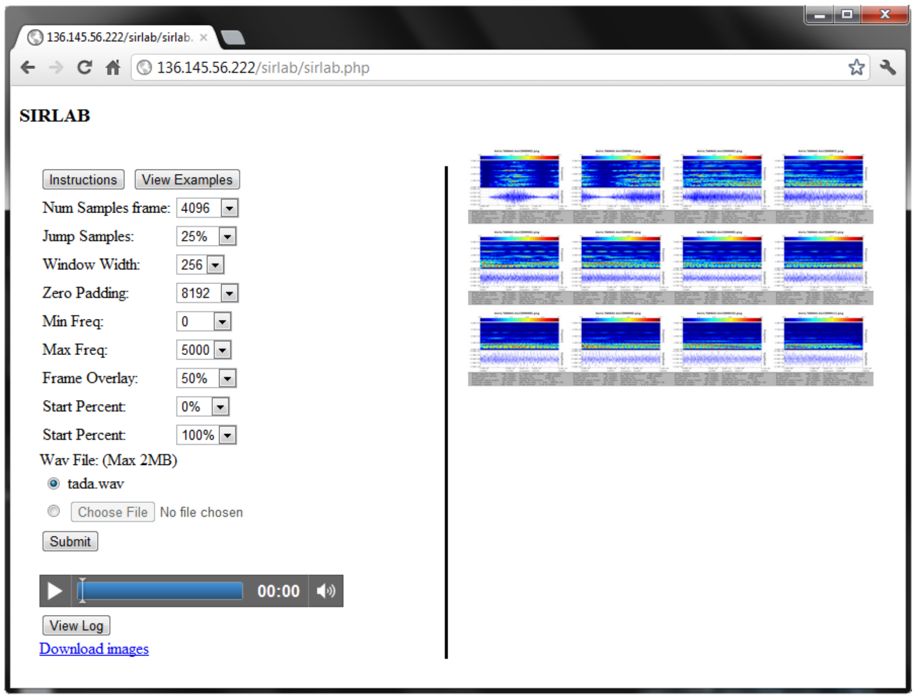
webSIRLAB is a web interface for SIRLAB where its advantages are:

1. No install: a new user only has to type one web address. It is recommended to have a set of wav files or a program for recording that ones.
2. Client Server Architecture: The client only has to make every requirement while the server is processing the data.

The Figure 3–2 shows the typical webSIRLAB page. At the left side there is a data form where the parameters are adjusted. Initially it takes the default values. At the right side the results are displayed in a image grid. Clicking on any image pops up a frame containing the zoomed image as seen on Figure 3–3.

The architecture for webSIRLAB is shown on Figure 3–4. It explanation is as follows:

1. The user fills the form, uploads a wav file, and submits.


136.145.56.222/sirlab/sirlab.php

SIRLAB

Instructions | View Examples

Num Samples frame:	4096
Jump Samples:	25%
Window Width:	256
Zero Padding:	8192
Min Freq:	0
Max Freq:	5000
Frame Overlay:	50%
Start Percent:	0%
Start Percent:	100%

Wav File: (Max 2MB)

● tada.wav

○ Choose File No file chosen

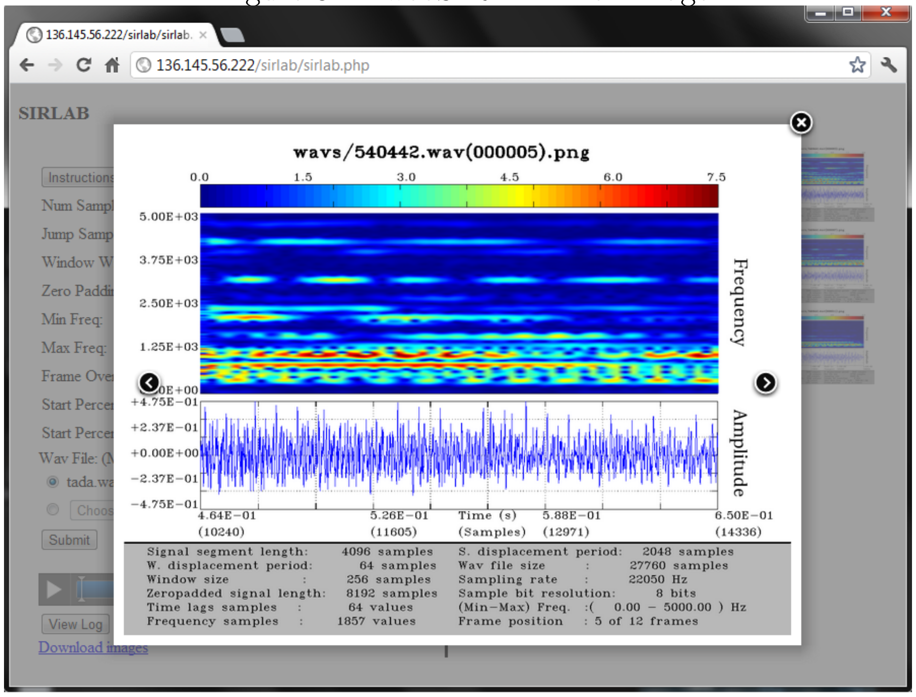
Submit

00:00

View Log

Download images

Figure 3–2: webSIRLAB Main Page



```
Signal segment length:    4096 samples   S. displacement period:   2048 samples
W. displacement period:     64 samples   Wav file size        :   27760 samples
Window size           :    256 samples   Sampling rate        :   22050 Hz
Zeropadded signal length:  8192 samples  Sample bit resolution:       8 bits
Time lags samples     :     64 values    (Min−Max) Freq.  :(   0.00 − 5000.00 ) Hz
Frequency samples     :   1857 values    Frame position    : 5 of 12 frames
```

Figure 3–3: Frame at webSIRLAB

2. webSIRLAB takes the parameters and the wav file and executes SIRLAB, producing a short log and, if no errors, the time frequency images.
3. The user's browser retrieves the images from the server.

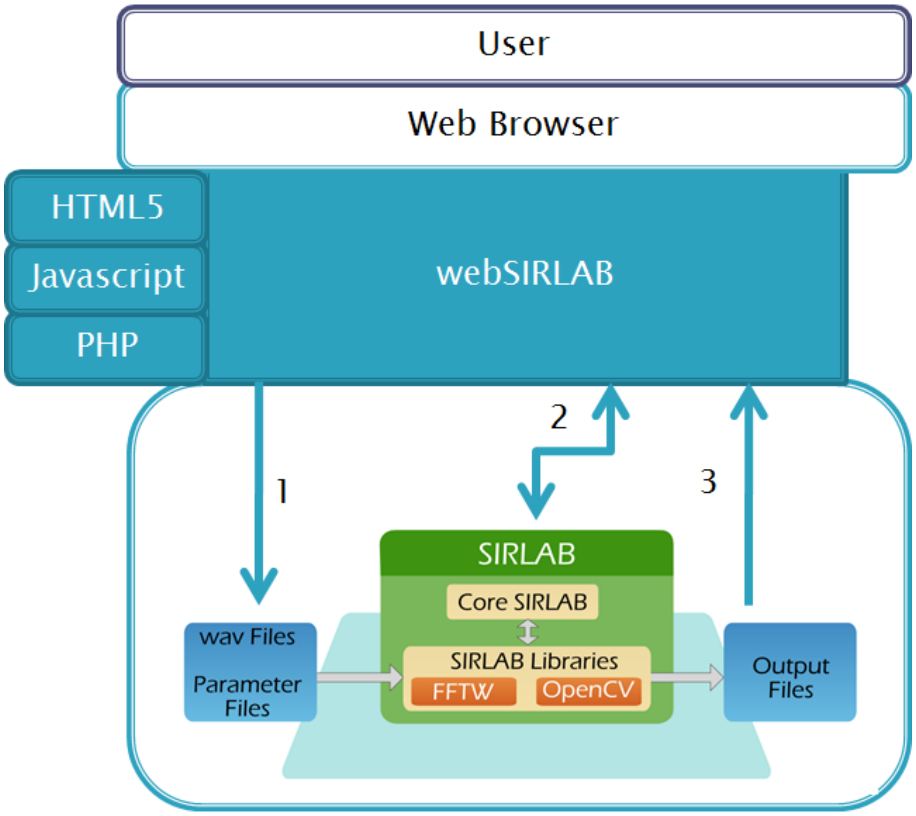


Figure 3–4: webSIRLAB Architecture

4. The images are shown in the browser.

In order to take advantages of the capacities of SIRLAB, webSIRLAB has a set of variables that can be modified in the web form. The Figure 3–5 shows the complete wav file signal to be processed, and two boxes -A and B- containing a detailed description of such signal, these details that are contained in Box A, are explained in the Figure 3–6 while the Box C is explained in the Figure 3–7. Each number in the figures represents a parameter. There are:

1. Num Samples Frame: number of samples that each frame contains and represents in its STFT
2. Frame Overlay: is a percentage of 'Num Samples Frame' and tells webSIRLAB how much every frame advances, being 0 not advance and 100 a full advance without overlap.

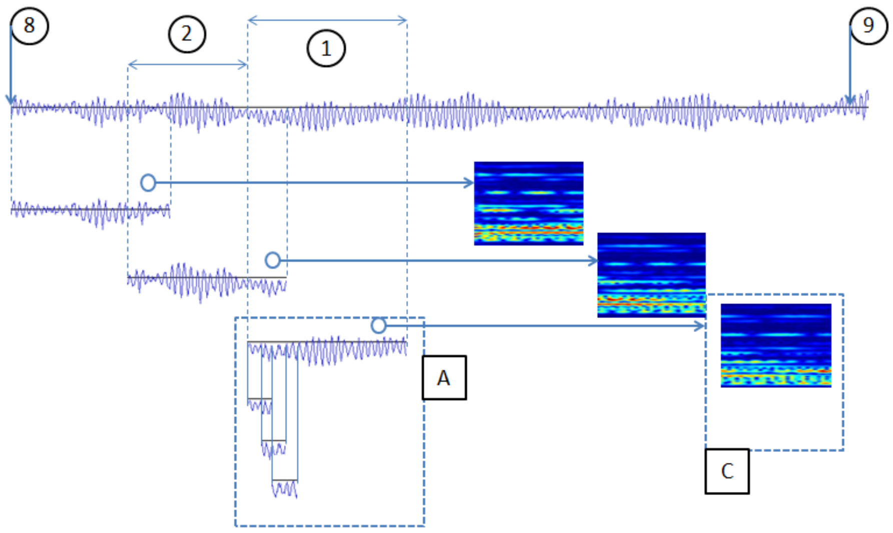


Figure 3–5: webSIRLAB Frame Parameters

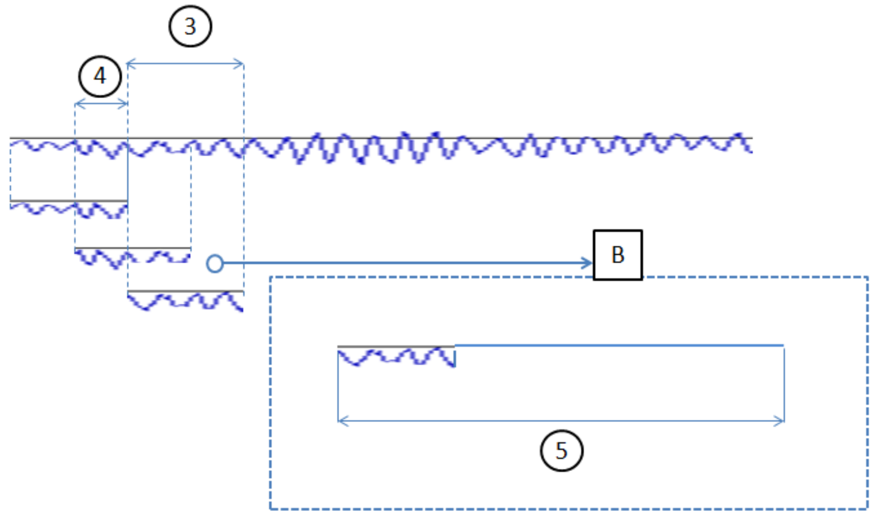


Figure 3–6: webSIRLAB Window Parameters

3. Window width: number of samples that are taken in a single window, a shorter number gives better temporal resolution. A greater one gives better spectral resolution.
4. Jump Samples: is a percentage of 'Window Width' used to define how much the window advances, being 0 not advance and 100 a full advance without overlap.

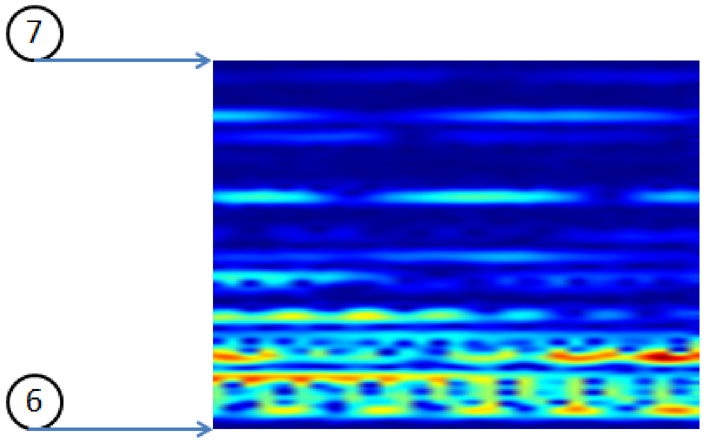


Figure 3–7: webSIRLAB Presentation Parameters

5. Zeropadding: zeroes are appended to the window to complete this number of samples, usually a power of two. This signal will be operated by the DFT.
6. Min Freq: is the minimal frequency that will be displayed on the result.
7. Max Freq: is the maximal frequency that will be displayed on the result.
8. Start Percent: the starting percentage at which the complete wav file signal will be processed.
9. End Percent: the ending percentage at which the complete wav file signal will be processed.

3.2 SIRDroid

SIRDroid is an Android interface for SIRLAB. In this lab we saw the importance of new mobile technologies. Noting that devices such as tablets and smartphones have every day a greater market share, and operating system android is more popular in this segment, it was decided to implement an application that could bring SIR-LAB to these devices[34]. Because these devices do not have the processing power required by SIRLAB, takes on particular significance the client-server architecture given earlier. SIRDroid can record environmental data, send them to process and receive in seconds The main objective with SIRDroid was to improve the end-user experience with the STFT on a tablet. Such devices give us several advantages such as:

- Mobility. Long life batteries and improved portability with lightweight components.
- Touch screen. Natural user interaction enables us to design an intuitive interface.
- Connectivity. Internet access with cellular data or wi-fi hotspot can connect to a server which process data.
- Multimedia. Microphone is used to record real time data while speakers reproduces it.

Taking this into consideration. It was necessary to develop an application to access the capabilities of webSIRLAB from a Tablet with Android. Android is an operating system for mobile devices such as smartphones and tablets. It is developed by the Open Handset Alliance. Its main features are:

- Open source: Free to use and modify as needed. The developer community is large and very active. It is well documented.
- Best-selling platform: The platform will be supported and extensively used in the near future.
- Solid platform: Linux is a stable and robust kernel.

SIRDroid is written in java targeting an android OS 3.2. Its architecture is shown in Figure 3–8, and described as follows:

1. The microphone (Mic) records environmental acoustic data.
2. The client Android mobile device creates a wav file and sends it through the Internet to a remote webSIRLAB server.
3. The Remote webSIRLAB server process the wav file with a set of standard parameters and sends the resulting images to the client.

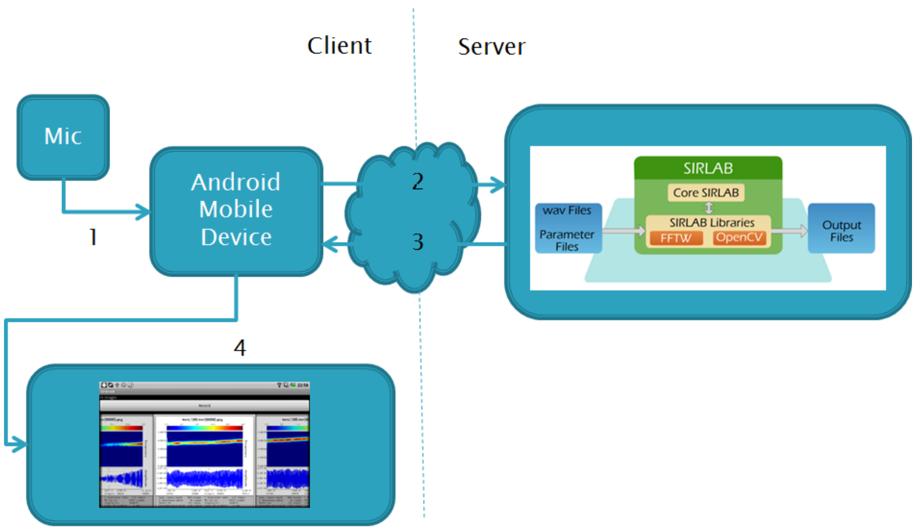


Figure 3–8: SIRDroid Architecture

4. The client receives the images and displays them in its screen as seen on the Figure 3–9.

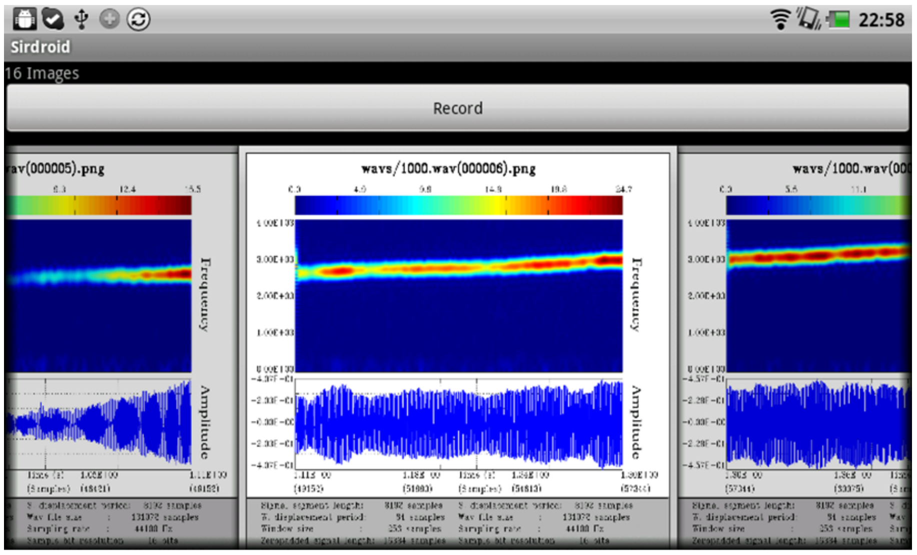


Figure 3–9: SIRDroid Result

3.2.1 SIRDroid Implementation

As mentioned before, the implementation was done into the Java programming language. Its main characteristic is a Object Oriented Programming language in which everything can be modeled as an object with a set of properties and methods. Among other features java has:

1. General Purpose: Java is suitable for a wide range of applications that goes from data management to digital signal processing.
2. Concurrent: multitasking programming, parallel execution.
3. Class Based. Robust and scalable.

Keeping this in mind, the SIRDroid application was designed as a set of classes with certain roles. The class diagram can be seen in Figure 3–10.

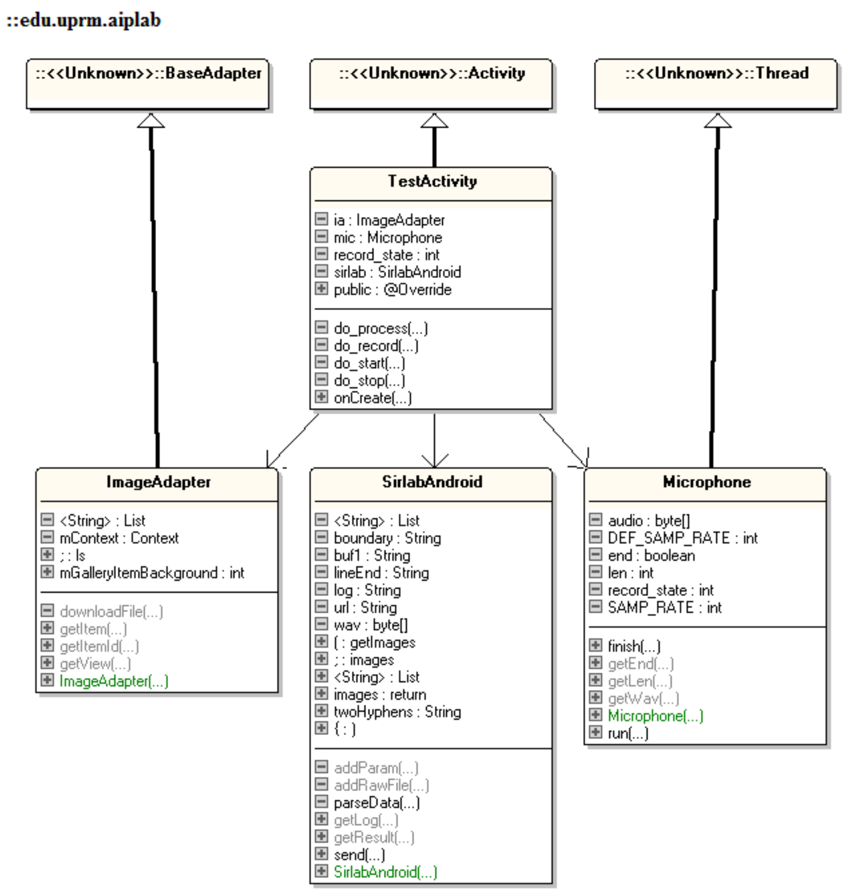


Figure 3–10: SIRDroid Class Diagram

1. ImageAdapter. Display the images.
2. Microphone. Manages the internal microphone and creates the wav file signal.
3. SirlabAndroid. Manages the connection with a webSIRLAB server.
4. TestActivity. Main class, synchronizes the application.

3.3 Installation

SIRLAB and SIRDroid(Server) are packed together in a single deb package. The minimum system requirements are:

- Processor: All fully compatible 686 (or newer) instruction set processors, 500 MHz
- Hard Drive space (for software): 4 MB
- Hard Drive space (for signals) : 16 GB
- RAM: 1024 MB

Recommended Minimum Configuration (for Optimal Performance):

- Processor: All fully compatible 786 (or or newer) instruction set processors, 2 GHz
- Hard Drive space (for software and temporary files): 32 GB
- RAM: 4 GB

Supported Operating Systems:

- Ubuntu Precise Pangolin 12.04(LTS)
- Debian Wheezy.

webSIRLAB will not install under older operating system versions. This is due to the new support to OpenCV libraries.

Suggested additional software:

- Apache2
- PHP5

Chapter 4

Software Implementation

4.1 GNU Radio

According to its developer[35], *GNU Radio* is a free and open-source software development toolkit. Its main characteristics are:

- Signal processing blocks.
- Can be used with readily low cost external RF hardware.
- Can be used as a simulation environment.
- Widely supported among hobbyist, academic, and commercial environments.
- *GNU Radio* was originally created for digital signal processing.

GNU Radio applications are primarily written using Python programming language. The critical performance signal processing is implemented in C++. The developer is able to implement real-time, high-throughput, signal processing paths in a simple to use development environment called GNU Radio Companion.

The main architecture of *GNU Radio* can be seen on the Figure 4–1. *GNU Radio* has several modules. These modules have blocks. All the blocks are 'glued' together with the Python language.

GNU Radio Companion (GRC) is a Simulink like graphical tool to design signal processing flow graphs. A description of its windows is shown in the Figure 4–2

- Menu bar: Is the region of the application interface where the menus are displayed.

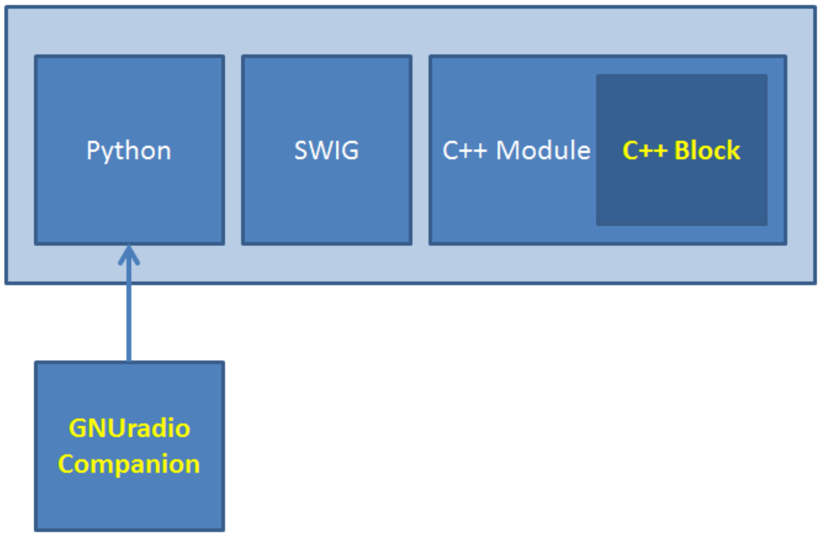
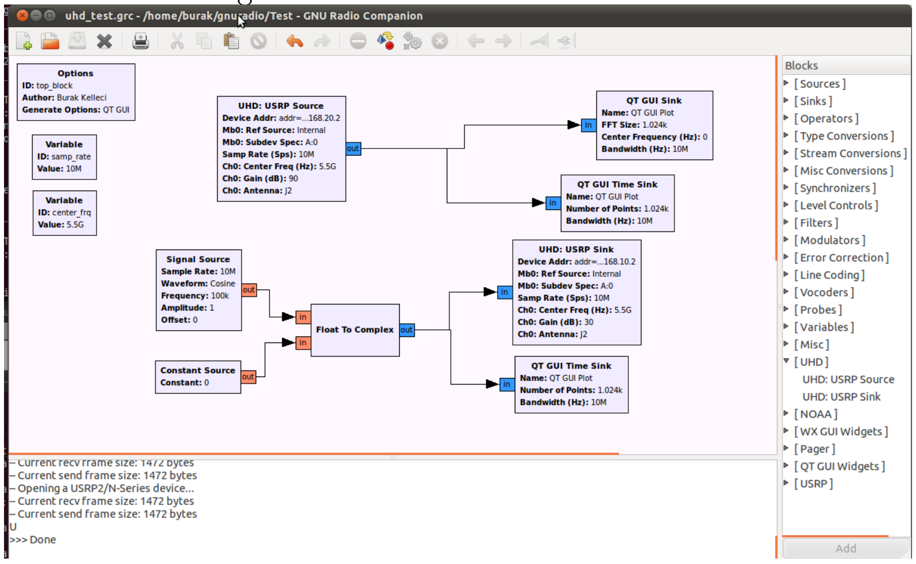


Figure 4–1: GNU Radio Architecture



Options
ID: top_block
Author: Burak Kelleci
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 10M

Variable
ID: center_frq
Value: 5.5G

UHD: USRP Source
Device Addr: addr=...168.20.2
Mb0: Ref Source: Internal
Mb0: Subdev Spec: A:0
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 5.5G
Ch0: Gain (dB): 90
Ch0: Antenna: J2

QT GUI Sink
Name: QT GUI Plot
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 10M

QT GUI Time Sink
Name: QT GUI Plot
Number of Points: 1.024k
Bandwidth (Hz): 10M

Signal Source
Sample Rate: 10M
Waveform: Cosine
Frequency: 100k
Amplitude: 1
Offset: 0

Constant Source
Constant: 0

Float To Complex

UHD: USRP Sink
Device Addr: addr=...168.10.2
Mb0: Ref Source: Internal
Mb0: Subdev Spec: A:0
Samp Rate (Sps): 10M
Ch0: Center Freq (Hz): 5.5G
Ch0: Gain (dB): 30
Ch0: Antenna: J2

QT GUI Time Sink
Name: QT GUI Plot
Number of Points: 1.024k
Bandwidth (Hz): 10M

```
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
-- Opening a USRP2/N-Series device...
-- Current recv frame size: 1472 bytes
-- Current send frame size: 1472 bytes
U
>>> Done
```

Figure 4–2: GNU Radio Companion

- Tools bar: It has also buttons to save, load, compile, and execute the system.
- Working area: It contains the system design. The blocks are drag and dropped into this area and its connections are defined.
- Blocks: It contains a list of all the blocks available. A block can be selected and inserted into the working area. Custom and programmed blocks are also included here.

4.1.1 Creating Custom Blocks

New custom blocks can be created by connecting existing blocks together. There are two special block that are intended to simulate a generic input or a generic output. Connecting such elements as a source or a sink can model more complex systems. These new blocks can be used on other blocks. The Figure 4–3 provides an example of how such thing can be done.

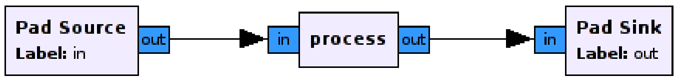


Figure 4–3: GNU Radio Custom Block

4.1.2 Programing Custom Blocks

Sometimes there is no way to make a block with the available tools at gnuradio-companion. However it is possible to write a new block and integrate it as a new library into *GNU Radio*. By this way there is more control in the algorithmic behavior. This is done using the C++ programming language and extending the abstract class *gr_block* and implementing the inherited function general_work(). The class diagram can be seen on the Figure 4–4.

After extending the *gr_block class*, it is needed to make it available to the Python interpreter and the GRC editor. This is done following the next steps:

1. Add a python definition to the module python definition.
2. Add a XML file describing the inputs and outputs to GRC

In a GNU environment, the automatic building program 'make' will perform the compilation.

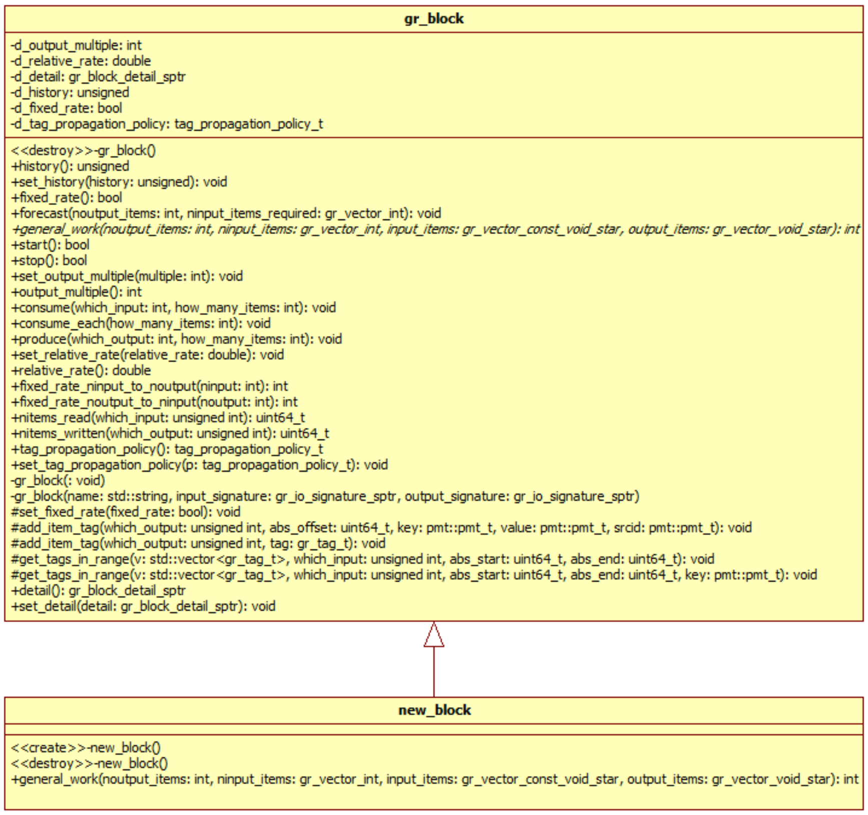

gr_block
-d_output_multiple: int
-d_relative_rate: double
-d_detail: gr_block_detail_sptr
-d_history: unsigned
-d_fixed_rate: bool
-d_tag_propagation_policy: tag_propagation_policy_t
<<destroy>>-gr_block()
+history(): unsigned
+set_history(history: unsigned): void
+fixed_rate(): bool
+forecast(noutput_items: int, ninput_items_required: gr_vector_int): void
+general_work(noutput_items: int, ninput_items: gr_vector_int, input_items: gr_vector_const_void_star, output_items: gr_vector_void_star): int
+start(): bool
+stop(): bool
+set_output_multiple(multiple: int): void
+output_multiple(): int
+consume(which_input: int, how_many_items: int): void
+consume_each(how_many_items: int): void
+produce(which_output: int, how_many_items: int): void
+set_relative_rate(relative_rate: double): void
+relative_rate(): double
+fixed_rate_ninput_to_noutput(ninput: int): int
+fixed_rate_noutput_to_ninput(noutput: int): int
+nitems_read(which_input: unsigned int): uint64_t
+nitems_written(which_output: unsigned int): uint64_t
+tag_propagation_policy(): tag_propagation_policy_t
+set_tag_propagation_policy(p: tag_propagation_policy_t): void
-gr_block(: void)
-gr_block(name: std::string, input_signature: gr_io_signature_sptr, output_signature: gr_io_signature_sptr)
#set_fixed_rate(fixed_rate: bool): void
#add_item_tag(which_output: unsigned int, abs_offset: uint64_t, key: pmt::pmt_t, value: pmt::pmt_t, srcid: pmt::pmt_t): void
#add_item_tag(which_output: unsigned int, tag: gr_tag_t): void
#get_tags_in_range(v: std::vector<gr_tag_t>, which_input: unsigned int, abs_start: uint64_t, abs_end: uint64_t): void
#get_tags_in_range(v: std::vector<gr_tag_t>, which_input: unsigned int, abs_start: uint64_t, abs_end: uint64_t, key: pmt::pmt_t): void
+detail(): gr_block_detail_sptr
+set_detail(detail: gr_block_detail_sptr): void
new_block
<<create>>-new_block()
<<destroy>>-new_block()
+general_work(noutput_items: int, ninput_items: gr_vector_int, input_items: gr_vector_const_void_star, output_items: gr_vector_void_star): int

Figure 4–4: Block Class Diagram

4.2 Time Frequency Analysis

4.2.1 Short Time Fourier Transform

As discussed in subsection 2.4.1 the Short Time Fourier Transform can be seen as a linear operator with Kronecker products. The current implementation was done in SIRLAB and was fully ported to the *GNU Radio* environment using the technique of programming a new block. As explained in section 3.1, like the original SIRLAB version, the version has multiple configurable parameters. Likewise has an input signal. This signal can be any source of *GNU Radio*. The output of this algorithm is a screen where the Short Time Fourier Transform can be seen in real time as the input signal is read. The block and its typical use can be seen in the Figure 4–5.

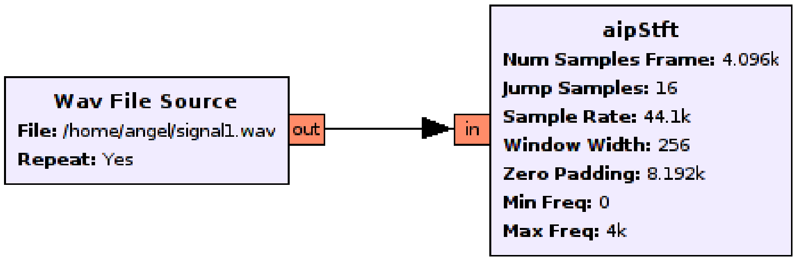


Figure 4–5: Short Time Fourier Transform Block

4.2.2 Ambiguity Function

The main formulation for the Ambiguity Function was discussed in subsection 2.4.2. This block was already implemented on SIRLAB and was ported to *GNU Radio* programming a new block. The parameters for this block are as follows:

1. Sample Rate: The rate at which the input signals are sampled.
2. Min freq: The minimum frequency required at the Time-Frequency representation.
3. Max freq: The maximum frequency required at the Time-Frequency representation.
4. Numsamplesframe: The number of samples that each frame uses for its calculation.

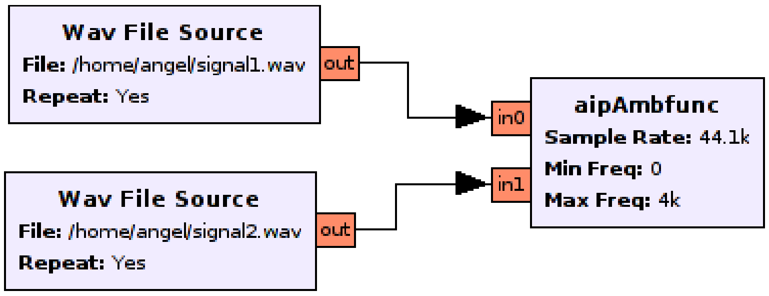


Figure 4–6: Ambiguity Function Block

4.2.3 Wigner Distribution

The main formulation for the Wigner Distribution was discussed in subsection 2.4.3. This block was implemented into *GNU Radio* programming a new block. The parameters for this block are as follows:

1. Sample Rate: The rate at witch the input signals are sampled.
2. Min freq: The minimum frequency required at the Time-Frequency representation.
3. Max freq: The maximum frequency required at the Time-Frequency representation.
4. Numsamplesframe: The number of samples that each frame uses for its calculation.

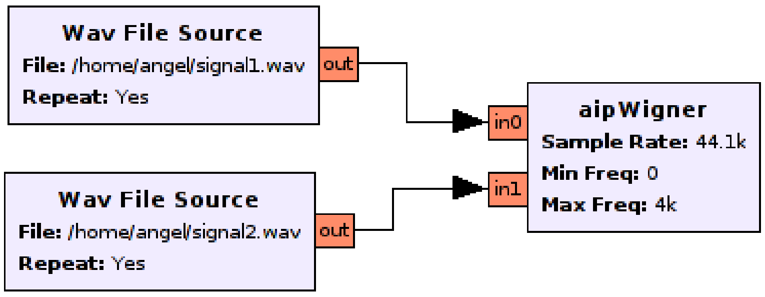


Figure 4–7: Wigner Distribution Block

4.2.4 Choi-Williams

The main formulation for the Choi-Williams Distribution was discussed in subsection 2.4.4. This block was implemented into *GNU Radio* programming a new block. The parameters for this block are as follows:

1. Sample Rate: The rate at witch the input signals are sampled.
2. Min freq: The minimum frequency required at the Time-Frequency representation.
3. Max freq: The maximum frequency required at the Time-Frequency representation.

4. Numsamplesframe: The number of samples that each frame uses for its calculation.

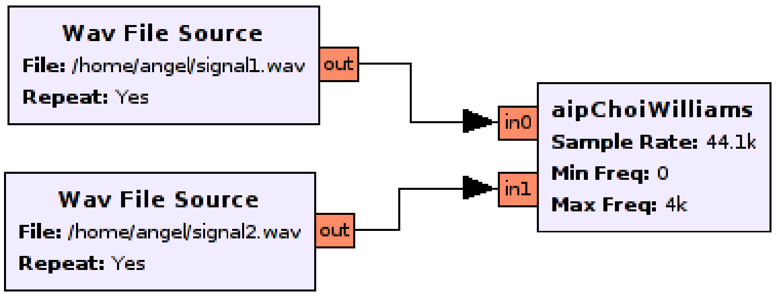


Figure 4–8: Choi-Williams Block

4.3 MIMO Analysis

4.3.1 SISO Block

The most basic MIMO system is the 1×1 or SISO system. It has 1 transmitter element and 1 receiver element. Two MIMO channels were presented in section 2.2.1. This is an implementation of such a model.

The two MIMO channel models presented above can be implemented under the paradigm of SDR with gnuradio-companion IDE. Each of the combination transmitter / receiver can be seen as a SISO model in which happen 3 stages: modulation, convolution and delay for first case. In the second case there is delay, convolution and modulation. This implementation would require only 3 blocks as shown in Figure 4–9.

4.3.2 4x4 MIMO Example

It implements the 4×4 MIMO channel model assuming that each pair transmitter / receiver is a SISO channel. This results in 16 SISO channels each with its respective convolution kernel, vector modulation and delay. Figure 4–10 shows how to carry out this process for a MCD model.

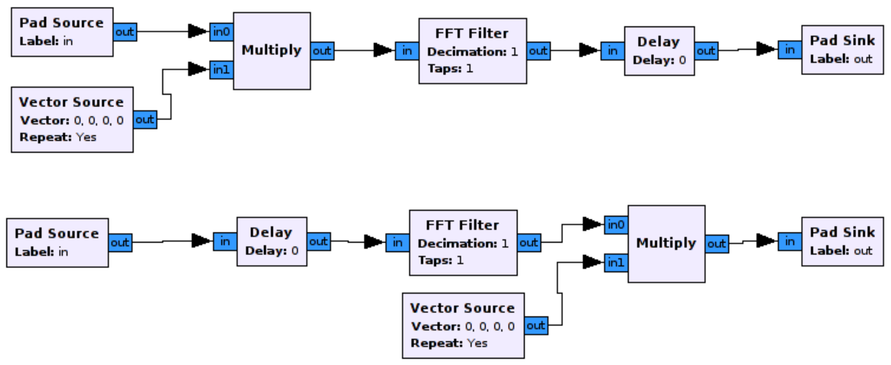


Figure 4–9: MCD and DCM SISO Model

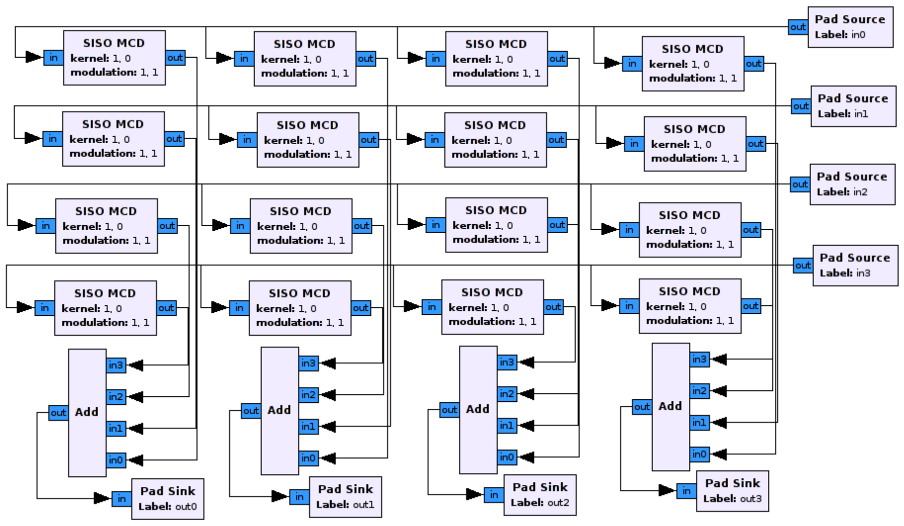


Figure 4–10: 4x4 MIMO Model

Finally the new block generated from an application can be used by connecting their respective Inputs and Outputs as shown in Figure 4–11

4.4 Applications and Testing

4.4.1 Applications

The first test we did with the STFT was in a SISO channel. In previous projects at this laboratory, the short time Fourier transform was used for the characterization of species in bioacoustics. It has also been used in SONAR applications.

In this example of short time Fourier transform, *GNU Radio* was used to generate a set of tones and the STFT to monitor that channel. There we can inspect the tones generated at 440, 880 and 1000 kHz generated at different time, the diagram

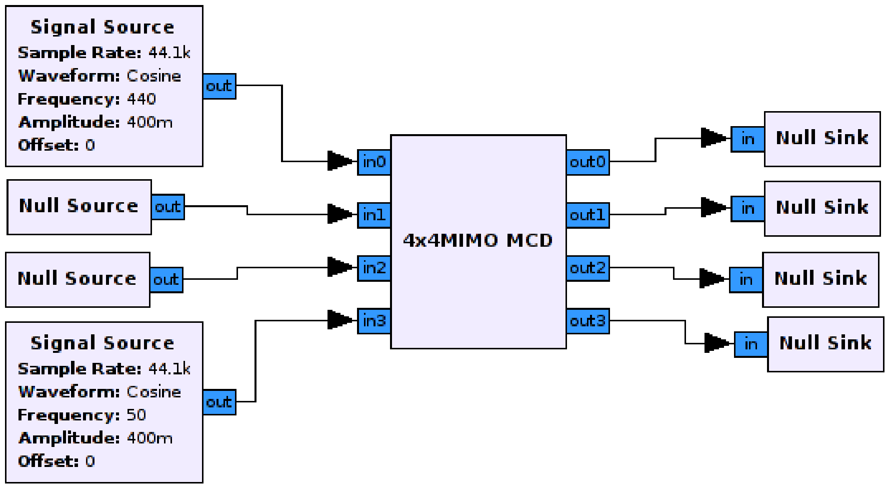


Figure 4–11: MIMO Block

to achieve this can be seen on Figure 4–12. The results are shown in Figure 4–13. There, the tones generated enter at different time.

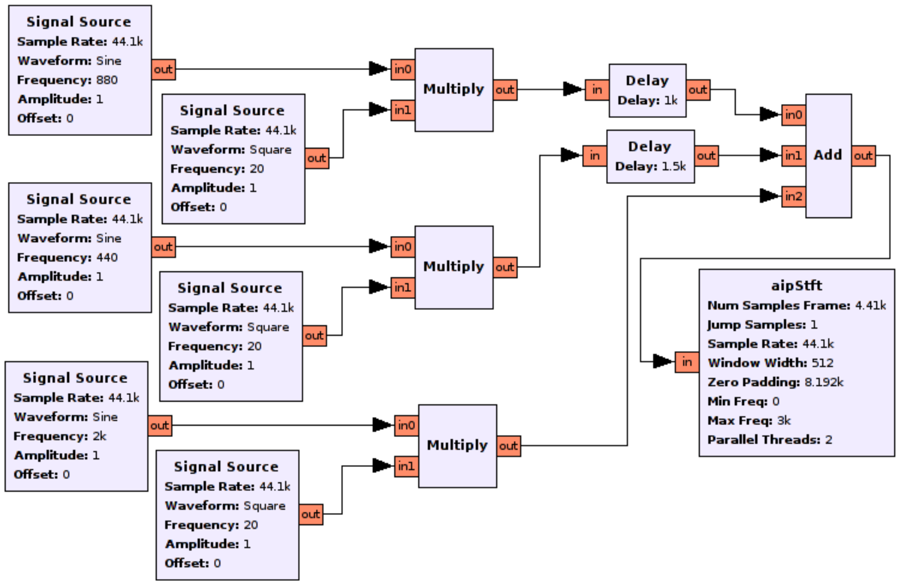
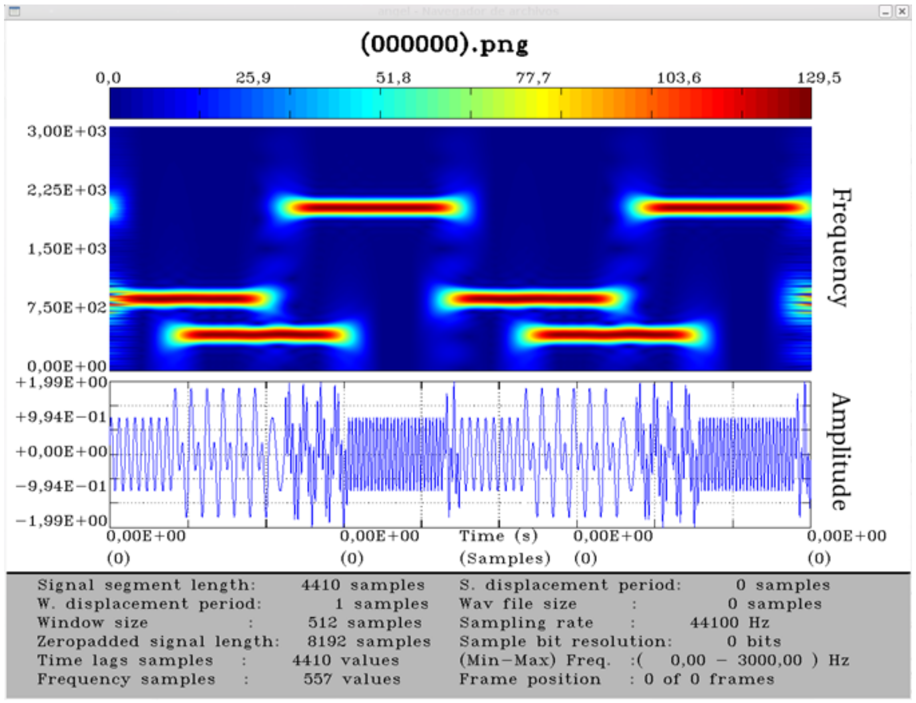


Figure 4–12: STFT Cognitive Radio Diagram

The ambiguity function is generally used in radar and sonar applications. The design of signals to fed a transmitter in these applications is done using the ambiguity function. A test signal and its auto ambiguity function should have an easy-to-discover maximum. In the example provided with the diagram 4–14, a sine signal is generated as a source, and was rescued a complete cycle. This signal has two



```
Signal segment length:      4410 samples     S. displacement period:      0 samples
W. displacement period:        1 samples     Wav file size       :        0 samples
Window size           :      512 samples     Sampling rate       :    44100 Hz
Zeropadded signal length:   8192 samples     Sample bit resolution:       0 bits
Time lags samples     :     4410 values      (Min−Max) Freq.  :(   0,00 − 3000,00 ) Hz
Frequency samples     :      557 values      Frame position    : 0 of 0 frames
```

Figure 4–13: STFT Cognitive Radio Result

copies that are sent to the ports at the ambiguity function block. The result shown in Figure 4–15 left. Another test was done similarly to obtain a square signal and its ambiguity function, which is shown in Figure 4–15 right.

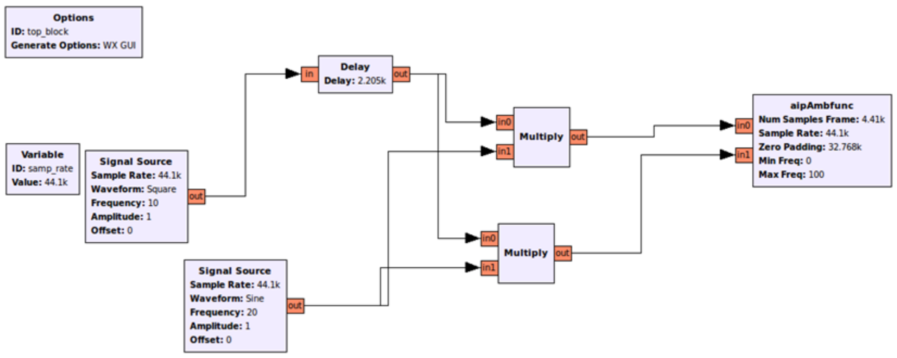
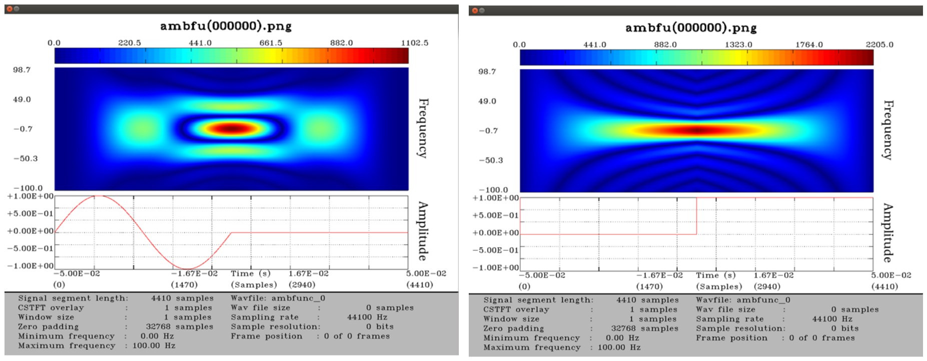


Figure 4–14: Auto Ambiguity Function Diagram

Another application for the ambiguity function is provided as an example in the figure 4–16.There an array of two microphones are set to register environmental acoustic signals. Each signal is connected directly to the ambiguity function block. Two test where performed sending a tone from different places. The ambiguity function measures the phase between the two captured signals as can be seen in

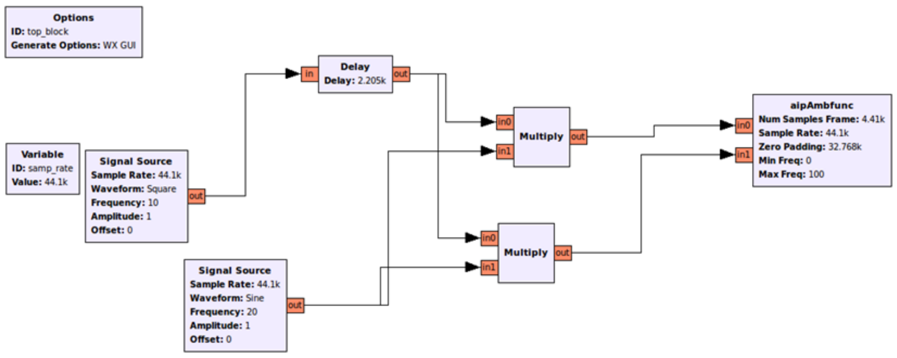


```
Signal segment length:     4410 samples   Wavfile: ambfunc_0
CSTFT overlay         :       1 samples   Wav file size      :          0 samples
Window size           :       1 samples   Sampling rate      :      44100 Hz
Zero padding          :   32768 samples   Sample resolution:            0 bits
Minimum frequency     :   0.00 Hz         Frame position    : 0 of 0 frames
Maximum frequency     : 100.00 Hz
```

```
Signal segment length:     4410 samples   Wavfile: ambfunc_0
CSTFT overlay         :       1 samples   Wav file size      :          0 samples
Window size           :       1 samples   Sampling rate      :      44100 Hz
Zero padding          :   32768 samples   Sample resolution:            0 bits
Minimum frequency     :   0.00 Hz         Frame position    : 0 of 0 frames
Maximum frequency     : 100.00 Hz
```

Figure 4–15: Auto Ambiguity Function

Figure 4–17. The graphic in the left side shows that the signals have no phase. The right one shows that the signals have phase. The graphics are different, and can be used to detect at which angle or direction the tone comes from. This is also a basic example in the use of the ambiguity function in MIMO systems.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 44.1k

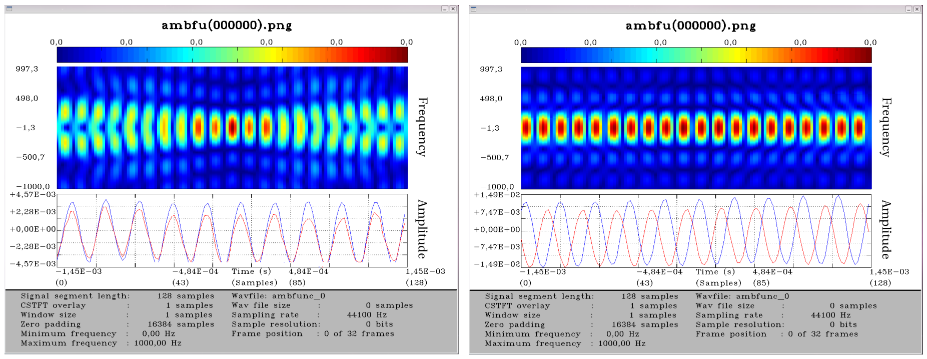
Signal Source
Sample Rate: 44.1k
Waveform: Square
Frequency: 10
Amplitude: 1
Offset: 0

Signal Source
Sample Rate: 44.1k
Waveform: Sine
Frequency: 20
Amplitude: 1
Offset: 0

Delay
Delay: 2.205k

aipAmbfunc
Num Samples Frame: 4.41k
Sample Rate: 44.1k
Zero Padding: 32.768k
Min Freq: 0
Max Freq: 100

Figure 4–16: Phase Detection Diagram

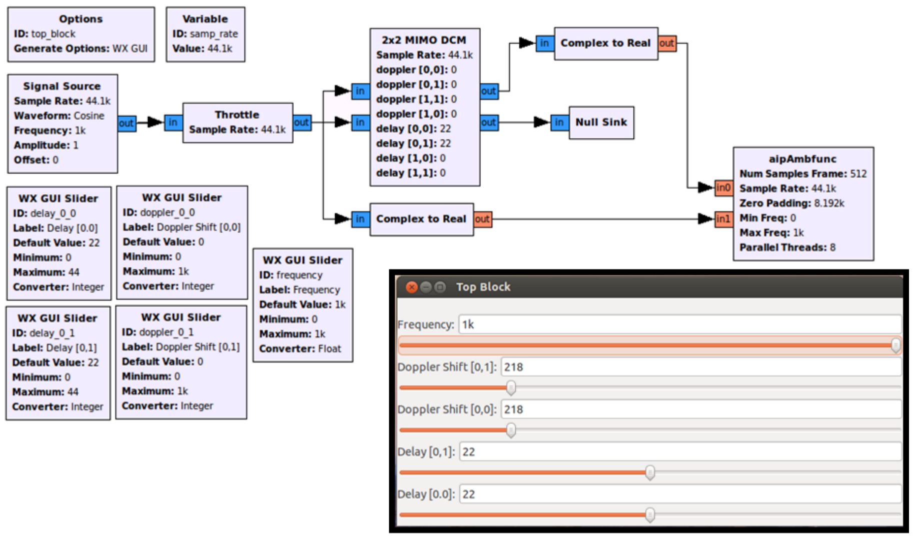


```
Signal segment length:      128 samples   Wavfile: ambfunc_0
CSTFT overlay         :       1 samples   Wav file size      :          0 samples
Window size           :       1 samples   Sampling rate      :      44100 Hz
Zero padding          :   16384 samples   Sample resolution:            0 bits
Minimum frequency     :   0,00 Hz         Frame position    : 0 of 32 frames
Maximum frequency     : 1000,00 Hz
```

```
Signal segment length:      128 samples   Wavfile: ambfunc_0
CSTFT overlay         :       1 samples   Wav file size      :          0 samples
Window size           :       1 samples   Sampling rate      :      44100 Hz
Zero padding          :   16384 samples   Sample resolution:            0 bits
Minimum frequency     :   0,00 Hz         Frame position    : 0 of 32 frames
Maximum frequency     : 1000,00 Hz
```

Figure 4–17: Phase Detection Result

We then put together time-frequency representations using a 2x2 MIMO DCM and configure its doppler frequency and delay for the SISO channels that are connected to the first output antenna. Then an ambiguity function is used to compare the input and output signals on the channel. The diagram to achieve this can be shown in Figure 4–18. The resulting ambiguity function is shown in Figure 4–19.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 44.1k

Signal Source
Sample Rate: 44.1k
Waveform: Cosine
Frequency: 1k
Amplitude: 1
Offset: 0

Throttle
Sample Rate: 44.1k

2x2 MIMO DCM
Sample Rate: 44.1k
doppler [0,0]: 0
doppler [0,1]: 0
doppler [1,1]: 0
doppler [1,0]: 0
delay [0,0]: 22
delay [0,1]: 22
delay [1,0]: 0
delay [1,1]: 0

Complex to Real

Null Sink

Complex to Real

alpAmbfunc
Num Samples Frame: 512
Sample Rate: 44.1k
Zero Padding: 8.192k
Min Freq: 0
Max Freq: 1k
Parallel Threads: 8

WX GUI Slider
ID: delay_0_0
Label: Delay [0,0]
Default Value: 22
Minimum: 0
Maximum: 44
Converter: Integer

WX GUI Slider
ID: doppler_0_0
Label: Doppler Shift [0,0]
Default Value: 0
Minimum: 0
Maximum: 1k
Converter: Integer

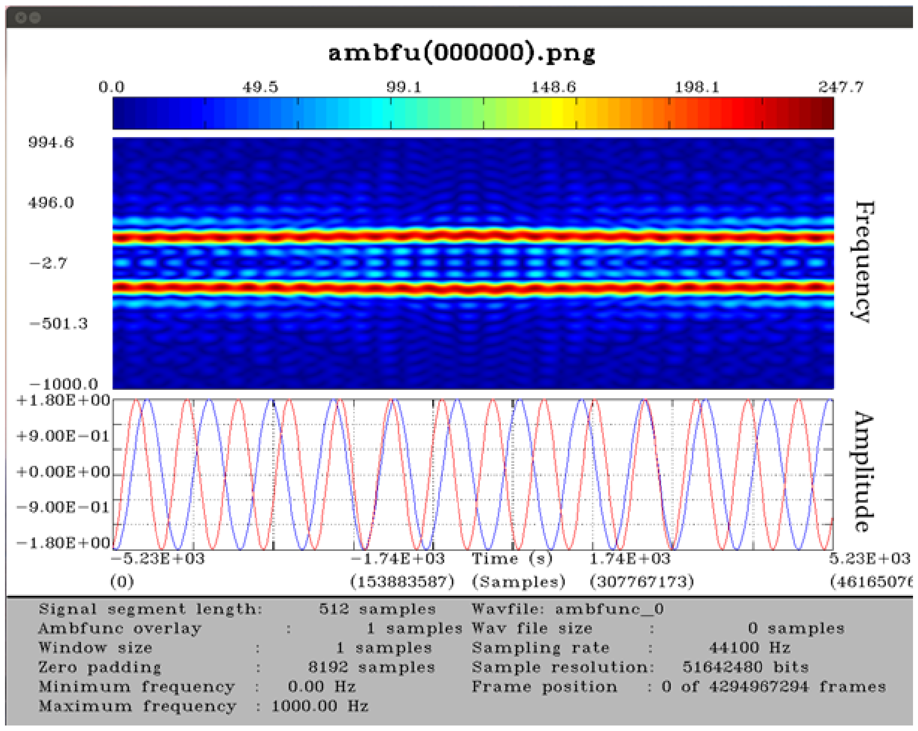
WX GUI Slider
ID: frequency
Label: Frequency
Default Value: 1k
Minimum: 0
Maximum: 1k
Converter: Float

WX GUI Slider
ID: delay_0_1
Label: Delay [0,1]
Default Value: 22
Minimum: 0
Maximum: 44
Converter: Integer

WX GUI Slider
ID: doppler_0_1
Label: Doppler Shift [0,1]
Default Value: 0
Minimum: 0
Maximum: 1k
Converter: Integer

Top Block
Frequency: 1k
Doppler Shift [0,1]: 218
Doppler Shift [0,0]: 218
Delay [0,1]: 22
Delay [0,0]: 22

Figure 4–18: MIMO Ambiguity Function Diagram



```
Signal segment length:      512 samples   Wavfile: ambfunc_0
Ambfunc overlay         :     1 samples   Wav file size      :        0 samples
Window size             :     1 samples   Sampling rate      :     44100 Hz
Zero padding            :  8192 samples   Sample resolution:   51642480 bits
Minimum frequency       :   0.00 Hz       Frame position     : 0 of 4294967294 frames
Maximum frequency       : 1000.00 Hz
```

Figure 4–19: MIMO Ambiguity Function Result

4.4.2 Algorithm Complexity

To prove the algorithms complexity we test each time-frequency block with different sizes: 512, 1024, 2048, 4096, and 8192. Each execution was done using one thread. The Figure 4–20 shows the complexity of the algorithms developed. The graph gives the measure in milliseconds for each of the algorithms. On the X axis are the sizes used and the corresponding execution time in milliseconds on the y axis. We can see the trend for all algorithms is $N^2\log(N)$, as had been predicted.

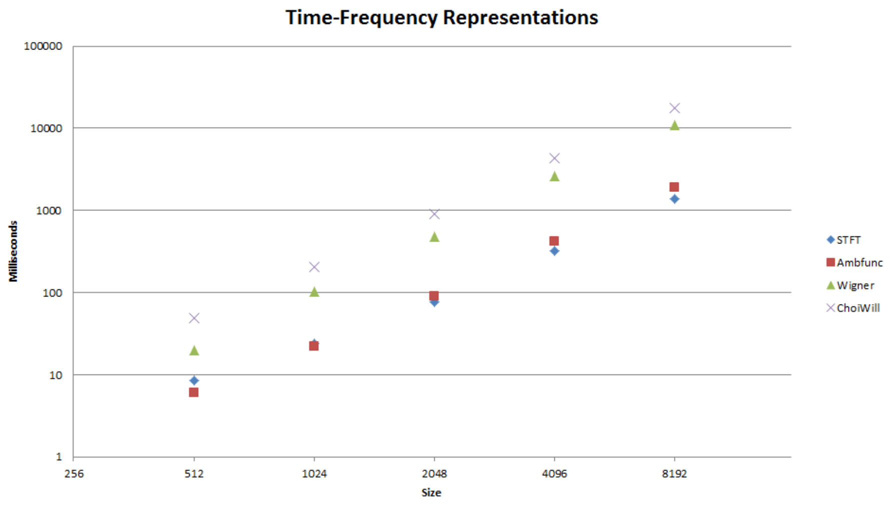


Figure 4–20: Execution Time, Milliseconds

4.4.3 Parallel Results

The time took to perform the Short Time Fourier Transform was measured changing both the size of each signal fragment, and the number of threads used for parallelization. We test the application-level parallelization on two systems. The first testing system was a dual core pc with the following figures of merit:

- CPU: 5850 Core2Duo 2.16GHz 4MB Cache, 2 cores, 2 threads
- Memory: 4GB DDR2 667MHz Dual Channel
- OS: GNU/Linux Debian 5.0 Kernel 2.6.32

The Kuck Model for this particular architecture can be seen on Figure 4–21 Where

- $P^i_{[0,1]}$: 2.16GHz

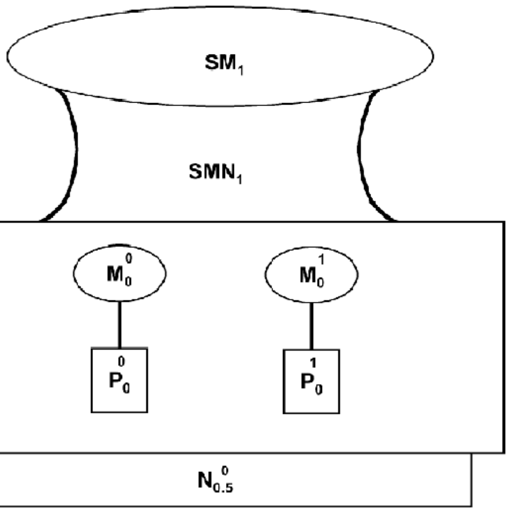


Figure 4–21: Intel T5850 Kuck Model

- M_0^i: 64KB
- SMN_1: N/A
- SMN_2: 5333.33 MB/s
- SM_1: 2MB
- SM_2: 4GB
- $N_{0.5}^0$: N/A

The resulting time for this system is shown in Figure 4–22.

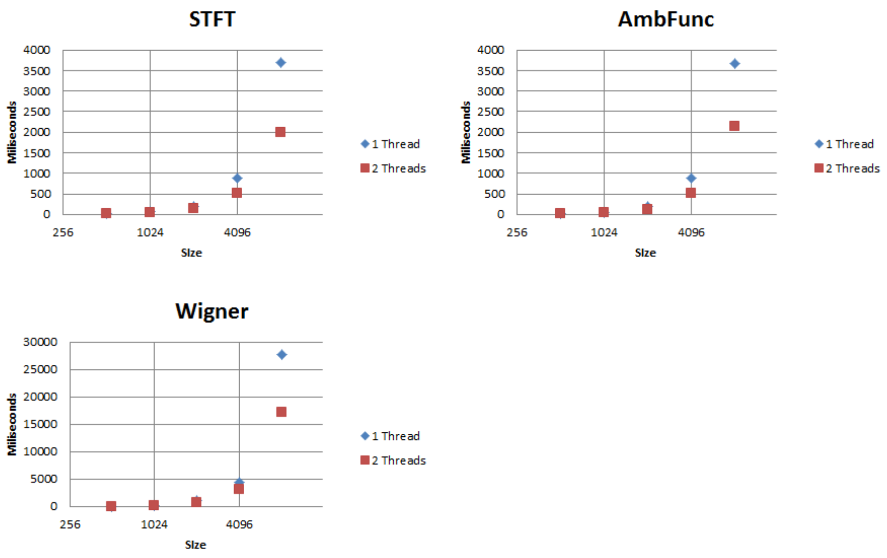


Figure 4–22: Intel T5850 Time

The second testing system is represented by the following figures of merit:

- CPU: 2630 Core i7 2.00GHz Nominal, 2.9GHz turbo, 6MB Cache, 4 cores, 8 threads
- Memory: 6GB DDR3 1333MHz Dual Channel
- OS: GNU/Linux Ubuntu 12.04 Kernel 3.2

The resulting time and speed up for this system are shown in Figure 4–23 and Figure 4–24 respectively.

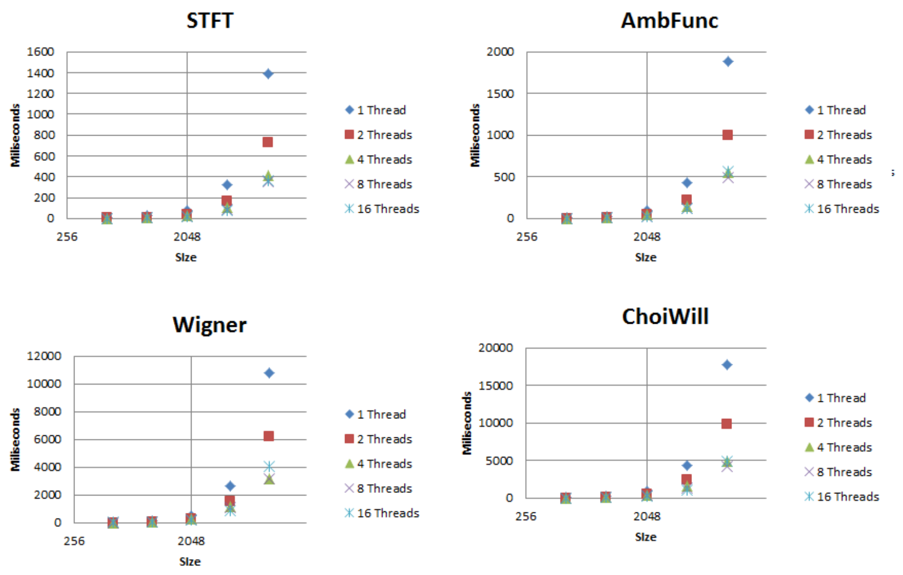


Figure 4–23: Intel 2630QM Time

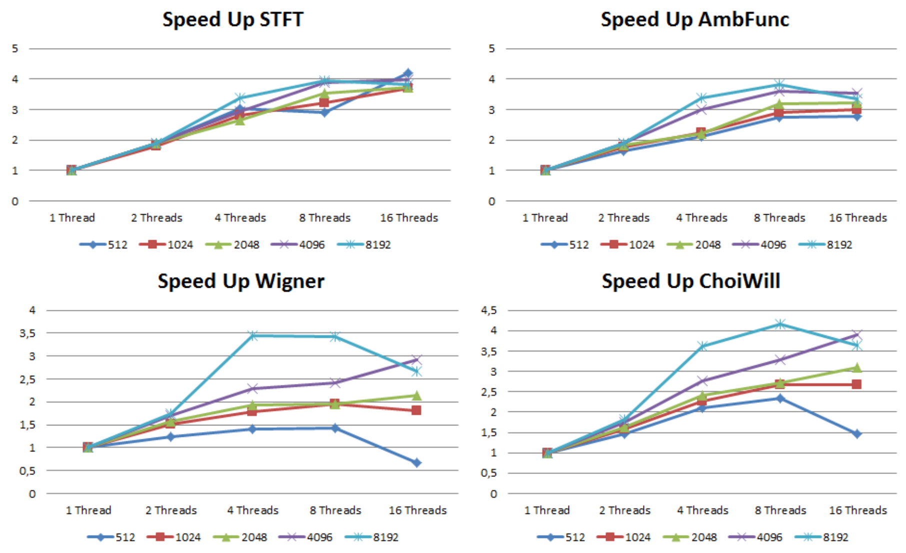


Figure 4–24: Intel 2630QM Speed Up

Generally speaking, the optimal performance is achieved using as many threads as cores. However a good recommendation is to use two threads per core. Using more than two threads per core have disadvantages given the additional effort that the operating system has to do managing the not-running processes.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this research work, a set of time frequency signal representations were implemented as part of a framework for MIMO simulations. Also, all the interfaces for such framework were defined and how to access them are explained. A benchmark was executed in order to measure speed and realtime capabilities.

A web interface was provided to perform Short Time Fourier Transform over a wav file. This enables the user to create time frequency representations on recorded real data. Another interface for realtime was developed for Android devices such as tablets and smartphones.

A base MIMO System was developed using two channel models: MCD (Modulation Convolution Delay) and DCM (Delay Convolution Modulation). An example of how to construct MIMO systems were provided along with a benchmark. The interfaces for such simulation were provided. A synthetic data was designed for test the system.

Finally, a MIMO modeling framework was designed and developed using time frequency representation tools and a MIMO channel simulation. Such framework is capable of performing at real data speed using a state-of-the-art computational hardware. Also the developed framework is highly compatible with the current signal interfaces such as wav files, data files, audio interfaces, network interfaces and USRP devices.

Such framework is useful to verify new theory and models for MIMO scattering channels in the electromagnetic and acoustic domains.

5.2 Future Work

- Develop input signals that can be fed to the framework in order to simulate various channel models. This can be done numerically with a computational environment such as Matlab. Also can be synthesized from a real data.
- Integrate with a DSP signal generator or a FPGA. This can be achieved using the framework's ability to connect read from the computer's sound system. A cable can be connected from the DSP/FPGA output to the computer input. The reverse case can be done connecting the computer output to the DSP/FPGA input.
- Integrate with the NETSIG computer module [36]. Actually the webSIRLAB interface is being adapted. With the framework ability to transport data remotely over a network this is useful to perform simulations remotely in a cloud computing way.
- Create additional simulations with high order MIMO. The modularity of the framework and the block oriented development makes this no difficult.

References

[1] IEEE Computer Society. Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac)and physical layer (phy) specifications amendment 5: Enhancements for higher throughput. *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)*, pages 1 –565, 29 2009. doi: 10.1109/IEEESTD.2009.5307322.

[2] A.I. Sulyman and M. Hefnawi. Performance evaluation of capacity-aware mimo beamforming schemes in ofdm-sdma systems. *Communications, IEEE Transactions on*, 58(1):79 –83, january 2010. ISSN 0090-6778. doi: 10.1109/TCOMM.2010.01.080111.

[3] D.R. Fuhrmann, J.P. Browning, and M. Rangaswamy. Signaling strategies for the hybrid mimo phased-array radar. *Selected Topics in Signal Processing, IEEE Journal of*, 4(1):66 –78, feb. 2010. ISSN 1932-4553. doi: 10.1109/JSTSP.2009.2038968.

[4] A. Luiz Garcia Reis, A.F. Barros, K. Gusso Lenzi, L.G. Pedroso Meloni, and S.E. Barbin. Introduction to the software-defined radio approach. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 10(1):1156 –1161, jan. 2012. ISSN 1548-0992. doi: 10.1109/TLA.2012.6142453.

[5] L. Williams and M.R. Inggs. Low cost networked radar and sonar using open source hardware and software. In *Radar Systems, 2007 IET International Conference on*, pages 1 –5, oct. 2007.

[6] A. Gupta, A. Forenza, and Jr. Heath, R.W. Rapid mimo-ofdm software defined radio system prototyping. In *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, pages 182 – 187, oct. 2004. doi: 10.1109/SIPS.2004.1363046.

[7] M. Patzold, B.O. Hogstad, and N. Youssef. Modeling, analysis, and simulation of mimo mobile-to-mobile fading channels. *Wireless Communications, IEEE Transactions on*, 7(2):510 –520, february 2008. ISSN 1536-1276. doi: 10.1109/TWC.2008.05913.

[8] P. Carlos, P. Yannis, V. Rodolphe, and C. Pierre. Sensitivity of the mimo channel characterization to the modeling of the environment. *Antennas and Propagation, IEEE Transactions on*, 57(4):1218 –1227, april 2009. ISSN 0018-926X. doi: 10.1109/TAP.2009.2015791.

[9] S.H. Zhou and H.W. Liu. Target statistical correlation characteristic for spatial-frequency jointly diversity multiple-input multiple-output radar. *Radar, Sonar Navigation, IET*, 5(6):638 –649, july 2011. ISSN 1751-8784. doi: 10.1049/iet-rsn.2010.0153.

[10] Z. Jindong, W. Kerang, and Z. Xiaohua. Spatial-dependent waveform design for colocated uniform linear array multiple-input multiple-output radar. *Radar, Sonar Navigation, IET*, 5(5):545 –550, june 2011. ISSN 1751-8784. doi: 10.1049/iet-rsn.2009.0297.

[11] P. Hallbjorner, J.D. Sanchez-Heredia, P. Lindberg, A.M. Martinez-Gonzalez, and T. Bolin. Multipath simulator measurements of handset dual antenna performance with limited number of signal paths. *Antennas and Propagation, IEEE Transactions on*, 60(2):682 –688, feb. 2012. ISSN 0018-926X. doi: 10.1109/TAP.2011.2173451.

[12] Mario Paredes. Domingo Rodrigues. Jorge Villamizar-Morales. La estructura algebraica del espacio de seales unidimensionales. *Revista Integracion, UIS*, 23 (2):15 –39, 2005.

[13] J.P. Soto Quiros. *"Computational Framework for Harmonic Treatment of Bidimensional Representations"*. 2011.

[14] J. Granata, M. Conner, and R. Tolimieri. Recursive fast algorithm and the role of the tensor product. *Signal Processing, IEEE Transactions on*, 40(12):2921 –2930, dec 1992. ISSN 1053-587X. doi: 10.1109/78.175736.

[15] Steven Johnson and Matteo Frigo. Implementing ffts in practice. *Connexions*, sep 2009.

[16] J. Winters. Optimum combining in digital mobile radio with cochannel interference. *Selected Areas in Communications, IEEE Journal on*, 2(4):528 – 539, jul 1984. ISSN 0733-8716. doi: 10.1109/JSAC.1984.1146095.

[17] J.H. Winters and M.J. Gans. The range increase of adaptive versus phased arrays in mobile radio systems. In *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, volume 1, pages 109 –115 vol.1, oct-2 nov 1994. doi: 10.1109/ACSSC.1994.471427.

[18] J. Winters. Optimum combining for indoor radio systems with multiple users. *Communications, IEEE Transactions on*, 35(11):1222 – 1230, nov 1987. ISSN 0090-6778. doi: 10.1109/TCOM.1987.1096697.

[19] Gerard. J. Foschini. Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. *Bell Laboratories Technical Journal*, page 41 59, oct 1996.

[20] J. Mitola. The software radio architecture. *Communications Magazine, IEEE*, 33(5):26 –38, may 1995. ISSN 0163-6804. doi: 10.1109/35.393001.

[21] G. S. Almasi and A. Gottlieb. *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1989. ISBN 0-8053-0177-1.

[22] Barry Wilkinson and Michael Allen. *"Parallel Programing"*. Prentice Hall, 1999.

[23] A. Skjellum, N.E. Doss, and P.V. Bangalore. Writing libraries in mpi. In *Scalable Parallel Libraries Conference, 1993., Proceedings of the*, pages 166 – 173, oct 1993. doi: 10.1109/SPLC.1993.365570.

[24] E.M. Khaneghah, S.L. Mirtaheri, and M. Sharifi. Evaluating the effect of inter process communication efficiency on high performance distributed scientific computing. In *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*, volume 1, pages 366 –372, dec. 2008. doi: 10.1109/EUC.2008.11.

[25] J.C. Moreira, E. Miguez, C. Vilacha, and A.F. Otero. Parallelization of an optimal power flow with a multicore symmetric shared memory computer. In *Environment and Electrical Engineering (EEEIC), 2011 10th International Conference on*, pages 1 –4, may 2011. doi: 10.1109/EEEIC.2011.5874767.

[26] S. Antão and L. Sousa. Exploiting simd extensions for linear image processing with opencl. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 425 –430, oct. 2010. doi: 10.1109/ICCD.2010.5647672.

[27] M. Hakim, J. Jais, and S. Salwa. Mosix: Implementation, trend and benchmark in malaysia. In *Information Technology, 2008. ITSim 2008. International Symposium on*, volume 3, pages 1 –6, aug. 2008. doi: 10.1109/ITSIM.2008.4632069.

[28] David J Kuck. *"High Performance Computing"*. Oxford University Press, 1996.

[29] Robert J. Voigt Linday B.H. May. Queueing theory modeling of a cpu-gpu system. May 2010. URL `http://www.ll.mit.edu/HPEC/agendas/proc10/Day1/PA10_May_abstract.pdf`.

[30] L. Cohen. *Time-Frequency Analysis*. Prentice Hall PTR, 1995.

[31] L. Auslander and R. Tolimieri. Characterizing the radar ambiguity functions. *Information Theory, IEEE Transactions on*, 30(6):832 – 836, nov 1984. ISSN 0018-9448. doi: 10.1109/TIT.1984.1056980.

[32] E. Wigner. On the quantum correction for thermodynamic equilibrium. *Phys. Rev.*, 40:749–759, Jun 1932. doi: 10.1103/PhysRev.40.749. URL `http://link.aps.org/doi/10.1103/PhysRev.40.749`.

[33] W.J. Williams H. Choi. Improved time-frequency representation of multicomponent signals using exponential kernels. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(6):862 –871, 1989.

[34] MobiLens comScore. April 2012 u.s. mobile subscriber market share, comscore reports, 2012. URL `http://www.comscore.com/Press_Events/Press_Releases/2012/6/comScore_Reports_April_2012_U.S._Mobile_Subscriber_Market_Share`. [Online; accessed 17-July-2012].

[35] GNU Radio. Gnu radio, main page, 2012. URL `http://gnuradio.org`. [Online; accessed 17-July-2012].

[36] D. Rodriguez, K. Lu, and C. Aceros. Sirlab-netsig integration for environmental surveillance monitoring in wireless mesh sensor networks. In *Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on*, pages 1 –4, feb. 2011. doi: 10.1109/LASCAS.2011.5750270.

[37] H.M. Lugo-Cordero, R.K. Guha, Kejie Lu, and D. Rodriguez. Secure service distribution for versatile service-oriented wireless mesh networks. In *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, pages 88 –94, oct. 2011. doi: 10.1109/MALWARE.2011.6112331.

APPENDICES

Appendix A

Basic Block Structure

```
class aip_example;

typedef boost::shared_ptr<gr_null_sink> gr_null_sink_sptr;


aip_example_sptr

aip_make_example (size_t sizeof_stream_item);


class aip_example : public gr_sync_block

{

    friend aip_example_sptr aip_make_example (size_t sizeof_stream_item);

    aip_example (size_t sizeof_stream_item);


 public:


    int work (int noutput_items,

        gr_vector_const_void_star &input_items,

        gr_vector_void_star &output_items);


};
```

Appendix B

Internal Loop Example

```
for (j = 0; j <= ltimemax; j++)
{
        // Multiply the signal by window. Return in window.
    haddamard_sig(datahd, datatx1, datarx1, numsamplesframe);
        // Do zeropadding to increase spectral resolution.
    zeropaddingf(datainpad, zeropadding, datahd, numsamplesframe);
        // Execute the plan
    fftw_execute( plan1); // FFT of the padded time signal.
        // Time shift of a signal.
    shift_signal(datatx1, jumpsamples, numsamplesframe, 1);
        // Shift the FFT result
    fft_shift(datainpad, dataout, zeropadding);
        // Write and shift time domain
    write2matstft(zeropadding, datainpad, 1, kfreqmin, kfreqmax,
        matstft, (j+numsamplesframe/2)%numsamplesframe, &maximoimag);
}
```

Appendix C

Generated Code

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: Top Block
# Generated: Tue May 12 00:34:46 2012
##################################################

from gnuradio import audio
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.gr import firdes
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
import aip
import wx

class top_block(grc_wxgui.top_block_gui):
```

```
def __init__(self):
    grc_wxgui.top_block_gui.__init__(self, title="Top Block")

    ##################################################
    # Variables
    ##################################################
    self.samp_rate = samp_rate = 44100

    ##################################################
    # Blocks
    ##################################################
    self.gr_vector_source_x_1 = gr.vector_source_f((1,0,0,0), True, 1)
    self.gr_interp_fir_filter_xxx_0 = gr.interp_fir_filter_fff(250, (1, ))
    self.audio_source_0 = audio.source(samp_rate, "", True)
    self.audio_sink_0 = audio.sink(samp_rate, "", True)
    self.ambfunc_0 = aip.ambfunc("ambfunc_0",44100,1,1,1024,0,10000,1024)

    ##################################################
    # Connections
    ##################################################
    self.connect((self.gr_vector_source_x_1, 0),
        (self.gr_interp_fir_filter_xxx_0, 0))
    self.connect((self.gr_interp_fir_filter_xxx_0, 0),
        (self.audio_sink_0, 0))
    self.connect((self.audio_source_0, 0),
        (self.ambfunc_0, 1))
    self.connect((self.gr_interp_fir_filter_xxx_0, 0),
                (self.ambfunc_0, 0))

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate

if __name__ == '__main__':
    parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
    (options, args) = parser.parse_args()
    tb = top_block()
    tb.Run(True)
```

Appendix D

VESO-Mesh Integration

VESO is the acronym for Versatile Service Oriented. It is implemented by a Wireless Mesh Sensor Network (WMSN). VESO mesh introduces the concept of a Service Oriented Routing Algorithm (SORA). SORA provides service orientation at the network layer, allowing integration of services. As explained by [37], SORA utilizes a DNS service to map the location of resources into an IP address.

In order to integrate the results given in this thesis with the VESO platform we are proposing the use of such results in this way: Take advantage of the protocols developed for the VESO mesh, to perform signal sensing (acquisition), signal communication (conveying) and signal processing(treatment), a special block has to be written into the proposed framework. This special block should implement the VESO mesh SORA protocols in order to decode the signal. Then that signal is processed by a workflow into *GNU Radio*. After, the results can be passed to a encoder block, that also implements the protocol for VESO mesh, sending the signal. There each master sensor node (MSN) in the network can perform the operations required for this signal. An example for such approach can be seen in Figure D–1 where we have such special blocks.

A set of computing devices can instantiate the proposed model, integrating a mesh as shown in Figure D–2. Then, the signals are acquired by a client and distributed over the mesh.

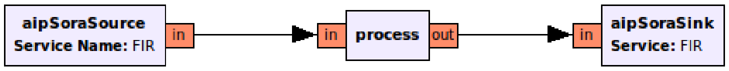


Figure D–1: SORA Block

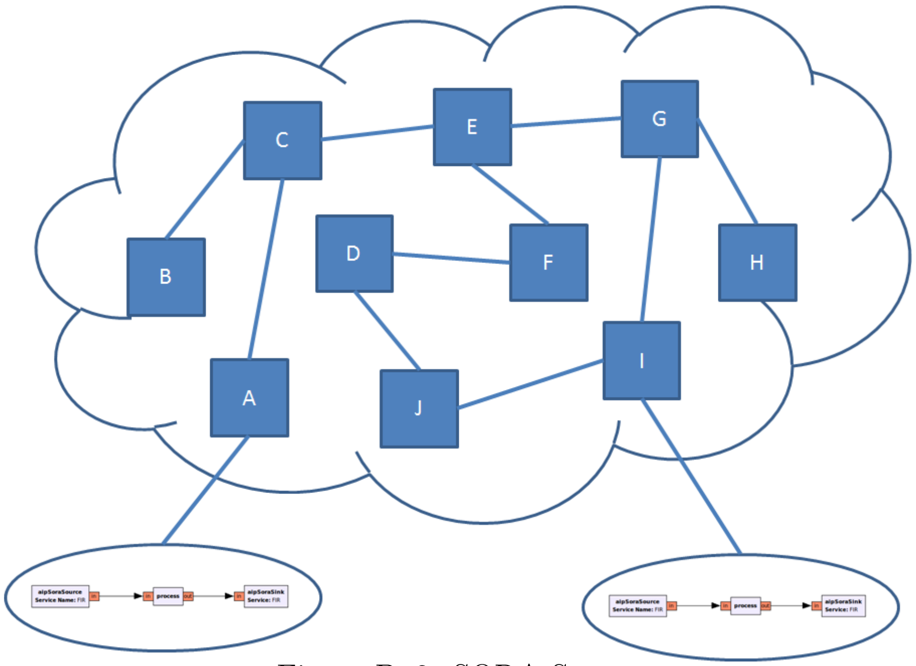


Figure D–2: SORA System

Appendix E

Installation

E.1 webSIRLAB, SIRDroid(Server)

A deb package has been provided to install webSIRLAB and SIRDroid into a computer. It needs an Ubuntu 12.04 operating system. Also administrative privileges. Assuming you open a terminal and are placed in the directory where the deb package is copied, then proceed to install typing the command:

```
pc@user:~/\$ sudo dpkg -i stftweb_2.0-1_amd.deb
```

The server is then accessible typing `localhost/sirlab` in a web browser.

E.2 SIRDroid(Client)

In order to install SIRDroid in a tablet, an apk package is provided. After copy such file into the device filesystem, proceed with the installation taping that file. Android will ask to accept the required privileges. If third-party apps is not enabled, Android will prompt you to modify the setting.

E.3 SIRDroid(Source)

For this, a zip file is provided containing the source code for SIRDroid. You must have an eclipse SDK with the Android plugin. After import the zip package into the workspace, create an Android device and run the application on them. More precise instructions on how to install the Android's SDK can be found in the next url: `http://developer.android.com/sdk/installing/index.html`.

E.4 aiplab-gnuradio

A tar-gz file is provided with the source code of the blocks developed and containing each time-frequency representation. GNU Radio and webSIRLAB must be installed prior to install aiplab-gnuradio. Assuming you are in the next directory where aiplab-gnuradio is copied, the installation proceeds typing this to a terminal:

```
pc@user:~/\$ tar -zxvf aiplab-gnuradio.tar.gz
pc@user:~/\$ cd aiplab-gnuradio
pc@user:~/aiplab-gnuradio\$ ./bootstrap
pc@user:~/aiplab-gnuradio\$ ./configure
pc@user:~/aiplab-gnuradio\$ make
pc@user:~/aiplab-gnuradio\$ sudo make install
```

After installing aiplab-gnuradio, gnuradio-companion shows the time-frequency representation blocks in the category *aip*.