SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

TESI DI LAUREA DI PRIMO LIVELLO

Automatic recognition of networks based on energy detection

Laureando
Luca Milani

Relatore
Prof.ssa Maria-Gabriella Di Benedetto

ANNO ACCADEMICO 2012/2013

Non credete a ciò che arriva senza sacrificio.
Non fidatevi, è un'illusione.
R.B.

SAPIENZA
UNIVERSITÀ DI ROMA

cost
EUROPEAN COOPERATION IN SCIENCE AND TECHNOLOGY

IC0902

TRINITY
COLLEGE
DUBLIN

CTVR the telecommunications research centre

Candidato: Luca Milani
Matricola: 1410986
Relatore: Prof.ssa Maria-Gabriella Di Benedetto

Part of this thesis has been done in CTVR / The Telecommunications Research Centre, "Trinity College", Dublin 2, Ireland during the month of "Short Term Scientific Mission" (STSM) "COST Action IC0902" under the supervisions of the Professor Luiz DaSilva and PhD student Paolo Di Francesco.

First I really would like to thank my advisor Professor Maria-Gabriella Di Benedetto for giving me this opportunity and for the support provided during this months. I would like to sincerely thank all of the members of CTVR who I met and interacted with during my stay at Trinity College of Dublin. In particular, I would like to thank Paolo Di Francesco for his patience and Professor Luiz DaSilva for accepting me in his research group, it was a wonderful experience that made me grow and it will remember forever.

Un ringraziamento speciale va a tutta la mia famiglia che mi ha supportato e incoraggiato verso questo primo traguardo. Cercherò sempre di ripagare la loro fiducia con impegno, dedizione ed educazione, nello studio e soprattutto nella vita. Ringrazio anche tutte quelle persone che hanno lasciato un segno indelebile in me, che mi hanno fatto crescere e mi hanno reso migliore.

Novembre 2013

ABSTRACT

English Version
This thesis outlines the work undertaken by the candidate, Luca Milani, at the "ACTS" lab, Department of Information, Electronics and Telecommunication Engineering (DIET), "Sapienza, University of Rome", Italy under the supervision of the Professor Maria-Gabriella Di Benedetto and during the month of "Short Term Scientific Mission" (STSM) "COST Action IC0902" at "Trinity College of Dublin" (Ireland) under the supervisions of the Professor Luiz DaSilva and PhD student Paolo Di Francesco.

The thesis project consists in designing a network recognition system in the 2.4 GHz ISM band based on MAC sub-layer feature. Through a simple implementation of an energy detector, it is possible to scan the ISM band in order to extract the frame pattern in time domain. This plain process is able to link a particular energy profile to one expected network. The final goal is to capture wireless signals (in particular Wi-Fi and Bluetooth) and to use the energy profile pattern for automatic recognition of the networks. The work is a part of AIR-AWARE Project, initiative of the ACTS lab that involves a large group of students and researchers.
The thesis is articulated in seven sections and two appendixes: I start with the work summary and purpose in the first chapter, then I continue with the networks standards (chapter 2) and the energy detection theory (chapter 3). Chapters 4 and 5 regard the devices used in this work, respectively the Universal Software Radio Peripheral (USRP) and the spectrum analyzer. Finally chapters 6 and 7 discuss about core researches and experiments with future work references.
In this work I considered three different scenarios: as a first experiment, I captured the data when only the wifi is active in the environment. As a second experiment I captured only Bluetooth signals. Finally I tried to recognize both Bluetooth and Wifi signals in a mixture scenario where a network was predominant on the other.

Italian Version

Questa tesi descrive il lavoro svolto dal candidato, Luca Milani, presso il laboratorio "ACTS", "Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni (DIET)" , de "La Sapienza, Università di Roma", sotto la supervisione della Professoressa Maria-Gabriella Di Benedetto e durante il mese di "Short Term Scientific Mission" nell'ambito del progetto "COST Action IC0902" al "Trinity College" di Dublino (Irlanda) sotto le supervisioni del professore Luiz DaSilva e del dottorando Paolo Di Francesco.

Il progetto di tesi consiste nella progettazione di un sistema di riconoscimento di rete nella banda ISM a 2,4 GHz basato sul substrato MAC. Attraverso una semplice implementazione di un rivelatore di energia, è possibile la scansione della banda ISM al fine di estrarre un diagramma energetico nel dominio temporale. Questo tipo di processo è in grado di collegare un particolare profilo di energia ad una specifica tecnologia wireless. L'obiettivo finale è quello di catturare i segnali senza fili (in particolare connessioni Wi-Fi e Bluetooth) e di utilizzare il modello appena esposto per il riconoscimento automatico delle reti. L'opera fa parte del progetto AIR-AWARE, iniziativa del laboratorio ACTS che coinvolge un folto gruppo di studenti e ricercatori. La tesi si articola in sette sezioni e due appendici: il primo capitolo espone il progetto e gli obiettivi per poi continuare con gli standard delle reti wireless considerate (capitolo 2) e la teoria dell'energy detection (capitolo 3) . I capitoli 4 e 5 riguardano i dispositivi utilizzati in questo lavoro, rispettivamente l'Universal Software Radio Peripheral (USRP) e l'analizzatore di spettro. Infine i capitoli 6 e 7 discutono delle ricerche e degli esperimenti svolti, integrando delle idee riguardo futuri studi. In questo lavoro vengono considerati tre diversi scenari: come primo esperimento, le catture vengono eseguite quando solo il Wi-Fi è attivo nell'ambiente. In un secondo esperimento vengono considerati solo i segnali Bluetooth. Infine viene tentato il riconoscimento ibrido sia di segnali Bluetooth che di segnali Wi-Fi, in uno scenario in cui una rete predomina sull'altra.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Preamble

Nowadays a large number of devices use to connect to networks using radio waves, and this number of devices is continuously growing. In general it is more and more likely that other devices are operating in a certain frequency band where the cognitive radio wants to work. In order not to interfere, or to exploit the unused frequency ranges, or just to be aware of the radio environment in which it is set, a cognitive radio has to discover if other wireless networks are active in that moment in that place. The presented work aims to evaluate this issue by proposing a method for automatic recognition and classification of wireless technologies. The considered frequency band is the Industrial Scientific and Medical (ISM) 2.4 GHz band and the considered networks are Bluetooth IEEE 802.15.1 and Wi-Fi IEEE 802.11.
ISM 2.4 GHz band is also used by many wireless mice and keyboards, cordless Wi-Fi phones and also by cameras for security closed-circuit TVs. Moreover, common interference at 2.4 GHz band comes from microwave ovens and DECT cordless phones (operating at 1.9 GHz); these can compromise the quality of the radio link of the other technologies, and they should be considered by the cognitive radio recognition system. The classification is a fundamental aspect for a cognitive radio device because it may be the initial step to be able to construe the surrounding environments.
The proposed approach consists of exploiting features of the Media Access Control (MAC) sub-layer of the various wireless technologies. Every network has its own particular MAC behavior, as defined in the standard that describes each of them. Through the study of these standards, a peculiar MAC behavior can be identified for each type of network, and a recognition and classification process can be carried out by evaluating these features. In

particular, all it need is a time-domain frame diagram. This diagram shows the presence versus absence of a frame in every instant with no attention to the content of these frames. In fact, the packet informations are not relevant for the scope of this recognition, the only important thing is the frame pattern, a medium for revealing the technology that is currently in use, and thus the networks recognition. This means that there can be a maximum or minimum duration for certain types of frames, or even a fixed duration. The same rules can be determined for the silence gaps that fall between the frames. Moreover the standard may specify a regular and predetermined transmission of a frame (usually these are control frames needed to ensure the correct system functionalities), or the transmission of acknowledgment frames after the reception of data frames. All these rules are specific for every single technology, i.e. each different network may present a MAC behaviour that is proper and peculiar of that technology.
In order to obtain the time-domain frame exchange diagram, all it need is a simple device: an energy detector (ED). Using the energy detection theory, the ED can compute the short-term energy that is present on the air interface. After defining a threshold value, all the consecutive short-term energy values that are higher than the threshold can be considered as frames. In this way, the frame diagram can be formed using energy detection [1].

1.2 AIR-AWARE Project

The work undertaken is a part of AIR-AWARE Project, the goal of this project is to implement a classification method based on information coming from protocol layers above the physical one (PHY). In particular, the objective is to recognize MAC sub-layer specific features for each of the ISM 2.4GHz band operating technologies. The AIR-AWARE module (Fig.1.1) was conceived as a generic device that provides information about the energy level in the whole ISM Band bandwidth. Through it we will be able to define presence or absence of a frame in real time. A software process analyzes the frame pattern provided by the energy detector and it should be included to the module.

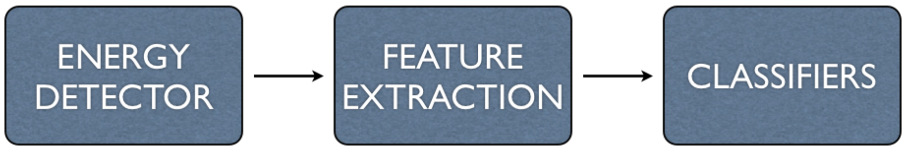


Fig. 1.1: AIR-AWARE MODULE.

1.3 Overview

The thesis is articulated in seven sections and two appendixes: I start with the work summary and purpose in the first chapter, then I continue with the network standards (chapter 2) and the energy detection theory (chapter 3). Chapters 4 and 5 regard the devices used in this work, respectively the Universal Software Radio Peripheral (USRP) and the spectrum analyzer. Finally chapters 6 and 7 discuss about core researches and experiments with future work references.
In this work I considered three different scenarios: as a first experiment, I captured the data when only the wifi is active in the environment. As a second experiment I captured only Bluetooth signals. Finally I tried to recognize both Bluetooth and Wifi signals in a mixture scenario where a network was predominant on the other.

CHAPTER 2

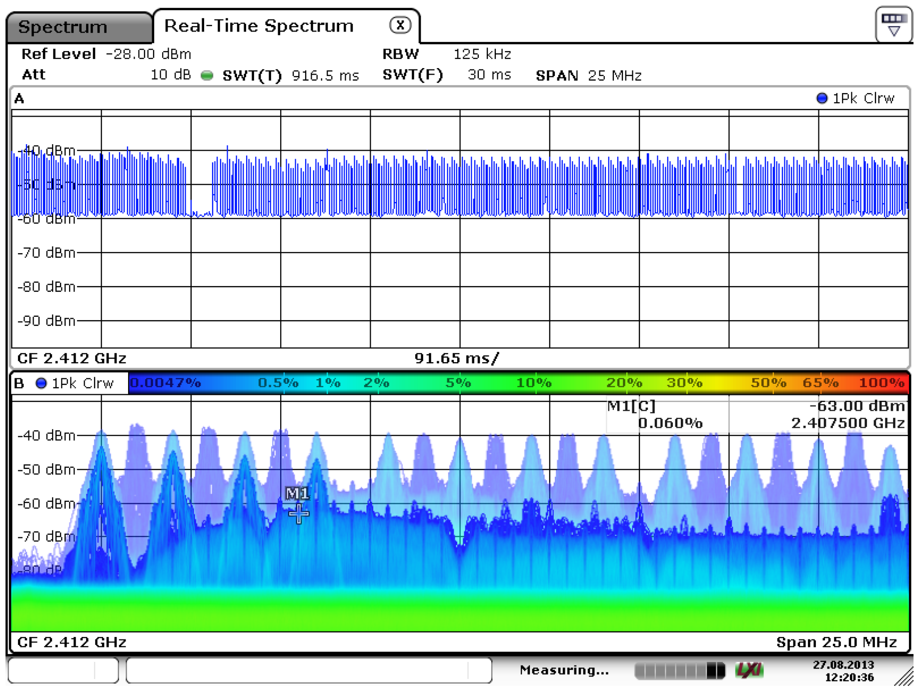
NETWORK STANDARDS

As first, this chapter starts with a general description of the considered two standards (IEEE 802.15.1 and IEEE 802.11), in both of them it is possible to identify some main features that enclose many types of transmissions or different versions. In some cases it will be simple to extract a feature directly from the standard, while in other cases it will be useful to investigate the captures in order to note a particular behavior.

2.1 The Bluetooth technology IEEE 802.15.1

Fig. 2.1: Bluetooth logo.

The Bluetooth technology (Fig. 2.1) is defined in the IEEE Standard 802.15.1 [3], that describes the specifications for the MAC and PHY layers, and it is used for Wireless Personal Area Networks (WPANs). This technology is available in quite every wireless device, such as cellular phones, laptops and netbooks, and for this reason it is very common to find an active Bluetooth device in many places. Bluetooth devices can communicate in the context of a piconet, that can be composed by 2-8 devices, all synchronized to a common clock and all sharing the same hopping sequence. In the piconet there is one device called master and the other devices are called slaves (up to 7). The master is the centre of the topology, that is to say that every slave communicates directly only with the master; in this way a communication between two slaves always passes through the master. The band used is the whole ISM 2.4 GHz band: from 2.4 to 2.4835 GHz. In fact the bandwidth of the signal is of 1 MHz, but the whole band is exploited by using the Frequency Hopping Spread Spectrum (FHSS) technique (Fig. 2.2 - 2.3).



Ref Level -28.00 dBm RBW 125 kHz
Att 10 dB SWT(T) 916.5 ms SWT(F) 30 ms SPAN 25 MHz

CF 2.412 GHz 91.65 ms/

M1[C] 0.060% -63.00 dBm 2.407500 GHz

CF 2.412 GHz Span 25.0 MHz

Date: 27.AUG.2013 12:20:36

Fig. 2.2: Bluetooth FHSS technique.

The ISM band is therefore divided into 79 channels of 1 MHz each. A combination of Gaussian Frequency Shift Keying (GFSK) and Phase Shift Keying modulation is used. Note that we took as reference the IEEE Standard 802.15.1 - 2005, that is the last IEEE available standard and that describes the version 1.2 of Bluetooth and some features of the version 2.0.

The version 2.0 of the Bluetooth Core Specification was released in 2004. The main difference is the introduction of an Enhanced Data Rate (EDR) for faster data transfer. The nominal rate of EDR is about 3 Mbit/s, although the practical data transfer rate is 2.1 Mbit/s. EDR uses a combination of GFSK and Phase Shift Keying modulation (PSK) with two variants, $\pi/4$-DQPSK and 8DPSK. EDR can provide a lower power consumption through a re-

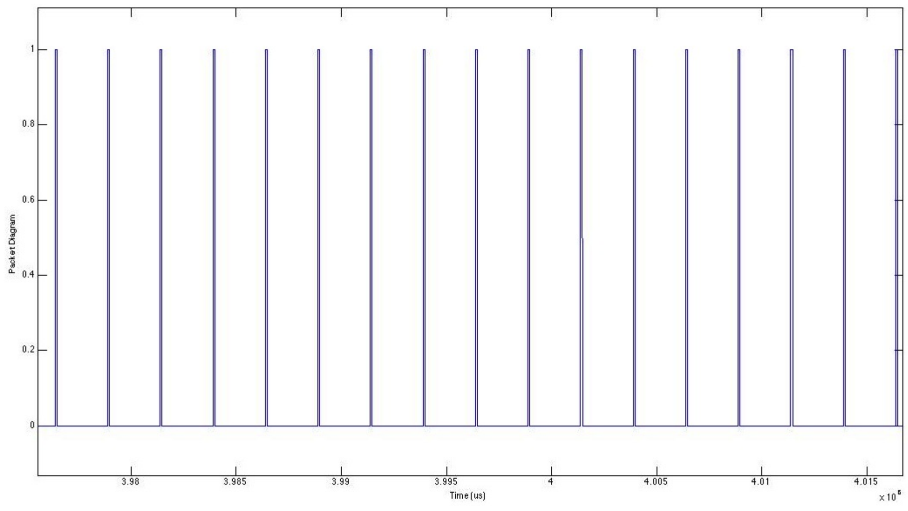


Fig. 2.3: FHSS technique in a simplified Short Energy Diagram.

duced duty cycle. The specification is published as "Bluetooth v2.0 + EDR" which implies that EDR is an optional feature. Aside from EDR, there are other minor improvements to the 2.0 specification, and products may claim compliance to "Bluetooth v2.0" without supporting the higher data rate.

Very important for the our scope is the division of the time axis into time slots. A packet can last an odd number of time slots; in particular, there can be 1-time slot packets, 3-time slots packets and 5-time slots packets. A communication between the master device and a slave device is usually composed by alternate packets (one from master and one from slave), since each device waits for a "return packet" (at least an acknowledgment) after sending a packet. Following these rules, imposed by the Standard, it is clear that a Bluetooth MAC packet exchange pattern is characterized by packets that start every time slot duration, or at multiples of this value, if considering the multi-slot packets. Furthermore, many acknowledgment packets are expected; the "NULL" packet is the one used for acknowledgment, and it has a fixed length of 126 bits, that corresponds to a fixed duration of 126 μs considering the bit rate of 1 Mb/s.

The other packets have also minimum and maximum durations, imposed by the Standard. It was decided to focus attention on the NULL packet and the packet length. In fact, the packet length during a transmission appears sufficiently concentrated around the maximum value admitted by the standard (in particular for the version 2.0). Three values will be predominant that correspond to the maximum size of the packages 1, 3 and 5 slots provided by the Standard. In particular the following table summarizes type and duration of all possible data packets.

Bit Rate:	1 Mb/s (ver 1.2)	2.1 Mb/s (ver 2.0)
NULL(ACK):	126 μs	60 μs
OneSlotFrame:	126 to 366 μs	174 μs (max value)
ThreeSlotFrame:	1250 to 1622 μs	772 μs (max value)
FiveSlotFrame:	2500 to 2870 μs	1367 μs (max value)

It is important to note that a Bluetooth communication system is dimensioned considering a bandwidth of 1 MHz in a single instant. By using an Energy Detector, the hopping sequence is unknown, and therefore it is impossible to know to which channel to be tuned to in every instant. In this condition, a simple way to catch the energy of all the packets that the devices send and receive is to sense the entire ISM 2.4 GHz band, i.e. all the 79 channels; by doing this, however, the noise power will be much higher, and this must be taken into account in the phase of determination of the threshold for the high versus low energy value. A possible alternative is to sense a lower bandwidth, in order to decrease the sensed noise power. In this way, however, all the packets sent in channels outside the sensed band are not caught. Considering that the "choice" to use a single channel has a uniform probability density, i.e. in mean there are no channels that are chosen more than others, sensing a lower bandwidth can still be a good trade-off between considered bandwidth and "packet loss" (in sensing term) [1].

2.2 The Wi-Fi technology IEEE 802.11

Fig. 2.4: Wi-Fi logo.

Wi-Fi is a popular technology that allows an electronic device to exchange data or connect to the internet wirelessly using radio waves (Fig. 2.4). Many devices can use Wi-Fi, e.g. personal computers, video-game consoles, smartphones, some digital cameras, tablet computers and digital audio players. These can connect to a network resource such as the Internet via a wireless network access point. Such an access point (or hotspot) has a range of about 20 meters indoors and a greater range outdoors. Hotspot coverage can comprise an area as small as a single room with walls that block radio waves, or as large as many square miles achieved by using multiple overlapping access points.

The Wi-Fi technology is defined in the IEEE Standard 802.11; in particular the reference standard taken into account is the revised version of 2007 [2]. There are different types of physical layers, each of them with a different used band, modulation transmission rates and coding; this results in different 802.11 Standard version (802.11a, b, c, d, e, f, g, h, i, j, k, n, p, r, s, v, w, y). The 802.11 b / g / n versions are considered in this work. A Wi-Fi system consists basically in an Access Point (AP) to which single client devices are connected, and that gives access to a wider network (usually Internet); in this way a Wireless Local Area Network (WLAN) is created. The physical layer of a Wi-Fi network is different depending on the Standard version, of course, but obviously even for the supported bitrate, whose value can be variable. For example, 802.11b uses the ISM 2.4 GHz band, with Direct Sequence Spread Spectrum (DSSS); possible bitrates are 1, 2, 5.5 and 11 Mb/s.

The modulations used are the following:

- Differential Binary Phase Shift Keying (DBPSK) for a bitrate of 1 Mb/s;
- Differential Quadrature Phase Shift Keying (DQPSK) for a bitrate of 2 Mb/s;
- Code Complementary Keying (CCK) for a bitrate of 5.5 and 11 Mb/s.

Different InterFrame Spaces (IFSs) are also defined. In particular, relevant for the purpose of the project, is the Short InterFrame Space (SIFS), the shortest of the IFSs. It is important for us because it is used before the transmission of an acknowledgment (ACK) packet or a CTS packet. It is defined as the time duration between the end of the last symbol of the previous packet and the beginning of the first symbol of the following packet (Fig. 2.5). Since the data-ACK packet exchange appears to be effectively really used, based on real traffic analysis in a scenario with medium to high traffic, the SIFS, among the different IFSs, is the most likely to occur. This is very important because it has a nominal value of 10 μs.

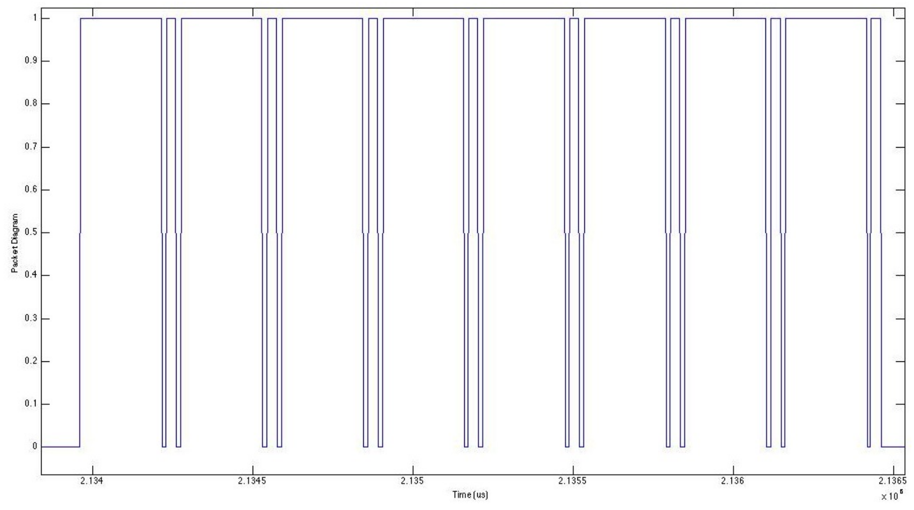


Fig. 2.5: Example of Wi-Fi packet diagram.

This value of 10 μs is important in this context because it is a silence gap value that occurs very often in a Wi-Fi transmission and it is peculiar of this technology, i.e. it characterizes this type of network. Thanks to this peculiarity, it can be a good candidate for being a feature [1].

CHAPTER 3

ENERGY DETECTION THEORY

The most common methods for the recognition of signals use to leverage the physical layer characteristics of the technologies. These approaches allow to detect the type of modulation employed through measuring certain parameters of the received signal, but they often require a specific ad-hoc technology to work. In order to evaluate the problem, we have chosen to focus on higher standard characteristics than physical. Just above the physical layer in fact, there is the MAC level, where each wireless technology defines the structure of the packages.

This chapter aims to present the spectrum sensing method used to scan the captures. The energy detection theory was used because it provides a clean result in obtaining the Short Energy Diagram (SED) in time, thanks to this diagram I was able to analyze to presence vs the absence of the frames. The energy detector is known as a suboptimal detector, which can be applied to detect unknown signals as it does not require a prior knowledge on the transmitted waveform as the optimal detector does. The Fig. 6.7 shows what is the SED, the first 3D plot provides the Energy versus both the time and the frequency while we are interested to visualize the energy versus time considering the whole bandwidth that we have. Conceptually we only need a rotation of the figure with a compaction of all frequency contributions.

For this reason the next sections present a mathematical review of the Energy Detection Theory and a synthesis about the Fast Fourier Transform used for this purpose.

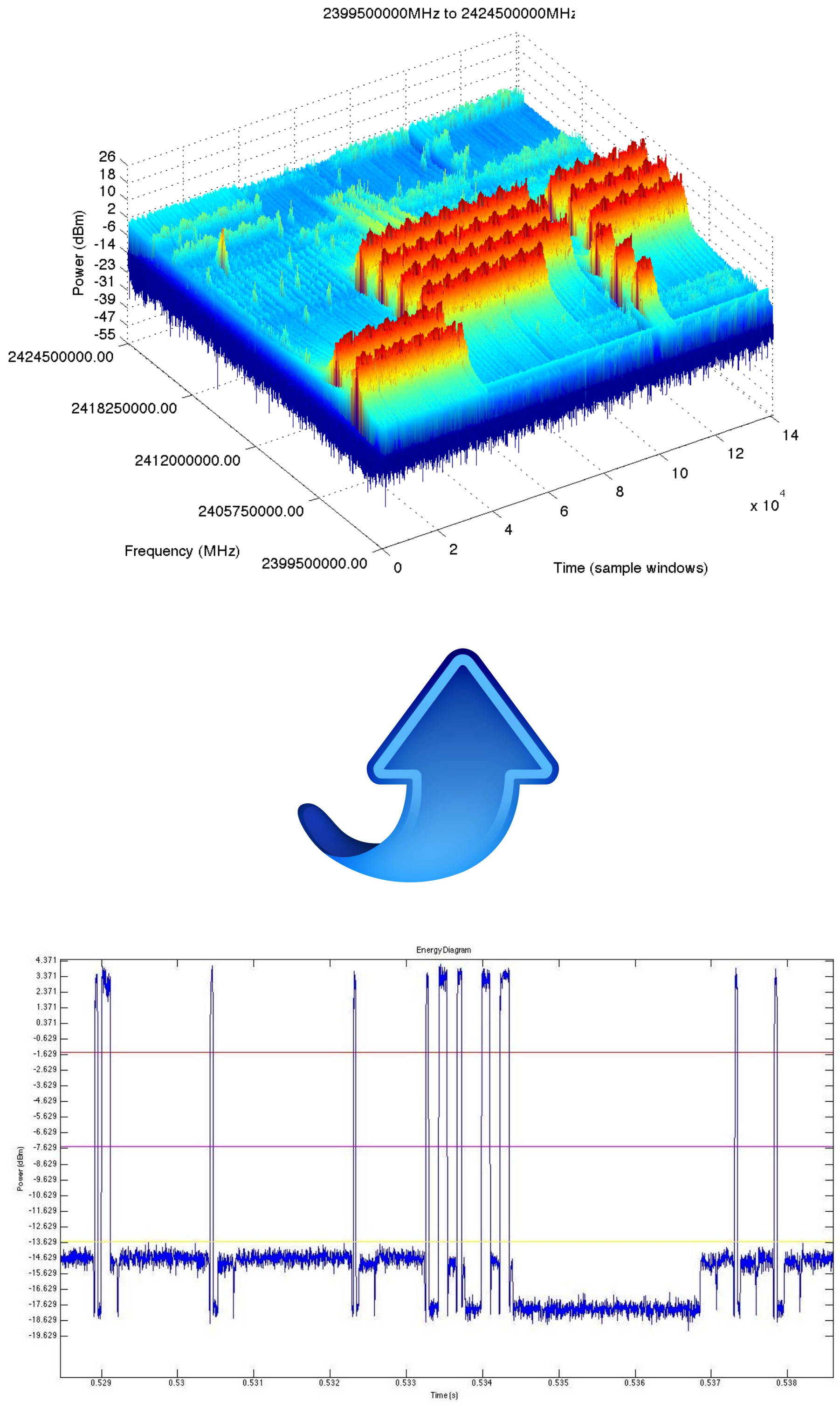


Fig. 3.1: 3D Energy Diagram.

3.1 Neymar-Person approach

The Energy Detector (ED) is a spectrum sensing method of analysis in the time domain that allows to obtain estimates on the utilization of spectrum in a very short time, being based on simple operations carried out in the time domain [4]. The ED is well known as a result of decision theory to the detection problem of random signals immersed in Additive White Gaussian Noise (AWGN). This approach, based on hypothesis testing, allows to solve problems of binary decision (presence or absence of valid signal) through the observation of received samples.

Thus ED is a non-coherent detection method that uses the energy of the received signal to determine the presence of primary signals. This simple method is able to gather spectrum- occupancy information quickly. However, its sensing capability is vulnerable to noise.

It is assumed a set of signal samples statistically independent:

$$(x[0],\ x[1],\ \dots,\ x[N-1]) \tag{3.1}$$

On these data, it applies a function $T(x)$ that is able to highlight a parameter for discriminating the decision. The determination of the function $T(x)$ and the decision threshold are the core issues of this approach. In order to detect the valid signal immersed in AWGN, the sampled received signal, $X[n]$ will have two hypotheses as follows:

$$H_1 :\ Y[n] = X[n] + W[n] \qquad Valid\ signal \tag{3.2}$$

$$H_0 :\ Y[n] = W[n] \qquad Noise\ floor \tag{3.3}$$

$$n = 1,\ 2,\ \dots,\ N \tag{3.4}$$

Where N is the number of samples. The noise $W[n]$ is assumed to be additive white Gaussian (AWGN) with zero mean and variance σ_w^2. $X[n]$ is the useful signal and it is assumed to be a random Gaussian process with zero mean and variance σ_x^2.
Hence it's possible to formulate the hypothesis as follows:

$$H_1 : \mathcal{N}(0, \sigma_x^2 + \sigma_w^2) \tag{3.5}$$

$$H_0 : \mathcal{N}(0, \sigma_w^2) \tag{3.6}$$

The signal detection through the Neymar-Person (NP) approach consists in a comparison between $\mathcal{L}(x)$ and the threshold γ:

$$\mathcal{L}(x) = \frac{p(x; H_1)}{p(x; H_0)} > \gamma \tag{3.7}$$

where $p(x; H_1)$ and $p(x; H_0)$ are the probability density functions associated to x when H_1 or H_0 is true. After choosing the decision of a certain false alarm probability (FA), the calculation of the threshold using NP passes through the evaluation of the following integral:

$$P_{FA} = \int\limits_{x:\mathcal{L}(x)>\gamma} p(x; H_0)\, dx > \gamma \tag{3.8}$$

Thus $\mathcal{L}(x)$ becomes:

$$\mathcal{L}(x) = \frac{\dfrac{1}{[2\pi(\sigma_x^2+\sigma_w^2)]^{\frac{N}{2}}} \; e^{-\dfrac{1}{2(\sigma_x^2+\sigma_w^2)} \sum_{n=0}^{N-1} x^2[n]}}{\dfrac{1}{(2\pi\sigma_w^2)^{\frac{N}{2}}} \; e^{-\dfrac{1}{2\sigma_w^2} \sum_{n=0}^{N-1} x^2[n]}} \tag{3.9}$$

After making the logarithm of the numerator and denominator and collecting the n-independent terms, the test function $\mathcal{T}(x)$ may be expressed as follows:

$$\mathcal{T}(x) = \sum_{n=0}^{N-1} x^2[n] \;\; > \; \gamma' \tag{3.10}$$

This version of ED calculates the signal samples energy only in a window time of length N. If the signal is present (H_1), $\mathcal{T}(x)$ should be above the threshold, instead if the signal is absent (H_0) the function should be under the threshold. Interesting enough is to operate as follows:

$$\mathcal{T}'(x) = \frac{1}{N}\mathcal{T}(x) = \frac{1}{N}\sum_{n=0}^{N-1} x^2[n] \tag{3.11}$$

The performance of the NP approach can be extracted from the probabilities of correct decision (P_D) and false alarm (P_{FA}). In fact:

$$\frac{\mathcal{T}(x)}{\sigma_x^2+\sigma_w^2} \sim \chi_N^2 \qquad\qquad H_1 \; case \tag{3.12}$$

$$\frac{\mathcal{T}(x)}{\sigma_w^2} \sim \chi_N^2 \qquad\qquad H_0\ case \tag{3.13}$$

In both cases the sum of variable squares gives a χ^2 (Chi-squared) distribution.

$$\mathcal{X}_\nu(x) = \frac{1}{2^{\frac{\nu}{2}}\Gamma(\frac{\nu}{2})}\, x^{\frac{\nu}{2}-1}\, e^{-\frac{x}{2}} \qquad\qquad with\ \ x > 0 \tag{3.14}$$

The probabilities become:

$$P_{FA} = P_r\{\mathcal{T}(x) > \gamma'; H_0\} = Q_{\chi_N^2}\left(\frac{\gamma'}{\sigma^2}\right) \tag{3.15}$$

$$P_D = P_r\{\mathcal{T}(x) > \gamma'; H_1\} = Q_{\chi_N^2}\left(\frac{\gamma'}{\sigma^2 + \sigma_s^2}\right) \tag{3.16}$$

Dividing the argument of Q (P_D) by the noise variance σ^2 and defining $\gamma'' = \gamma'/\sigma^2$, P_D becomes:

$$P_D = P_r\{\mathcal{T}(x) > \gamma'; H_1\} = Q_{\chi_N^2}\left(\frac{\gamma''}{\frac{\sigma_s^2}{\sigma^2} + 1}\right) \tag{3.17}$$

$$where\ \ SNR = \frac{\sigma_s^2}{\sigma^2}$$

Thanks to a large N and to the central limit theorem, the sum of the variable squares tends to assume a Gaussian distribution:

$$\mathcal{T}(x) \sim \mathcal{N}(N\sigma_w^2, 2N\sigma_w^4) \qquad\qquad H_1\ case \tag{3.18}$$

$$\mathcal{T}(x) \sim \mathcal{N}(N(\sigma_w^2 + \sigma_x^2), 2N(\sigma_w^2 + \sigma_x^2)^2) \qquad\qquad H_2\ case \tag{3.19}$$

Thus:

$$P_{FA} = Q\left(\frac{\gamma - N\sigma_w^2}{\sqrt{2N\sigma^4}}\right) \tag{3.20}$$

$$P_D = Q\left(\frac{\gamma - N(\sigma_w^2 + \sigma_x^2)}{\sqrt{2N(\sigma_w^2 + \sigma_x^2)^2}}\right) \tag{3.21}$$

For a certain value of the threshold, once set N and the signal variances, the previous formulas provide the so-called ROC (Receiver Operating Curve) that characterizes the performance of receivers. Later in this work it will be discussed the choices of the thresholds (chapter 6).
Using the formula of the energy detector just shown, it was decided to define the function of short-term energy (E_N).

$$E_N(v) = \sum_{i=1}^{N} |v_i|^2 \cdot T_s \tag{3.22}$$

Where N is the time length to calculate the energy, v_i is the i voltage sample inside the consider time window and T_s is the sample period.

The energy detector is known as a suboptimal detector, which can be applied to detect unknown signals as it does not require a prior knowledge on the transmitted waveform as the optimal detector does. Fig. 3.2 depicts block-diagram of an energy detector. The ADC is used to convert the received signal to the digital domain and it corresponds to two different devices in this work. Then the square magnitude of the digitized signal is calculated by using the Fast Fourier Transform (FFT) and magnitude square function. To make the measurement more accurate, N numbers of samples is taken and the average value of the samples is used to make the decision whether signal is present or not by comparing it with the threshold [5] [6].

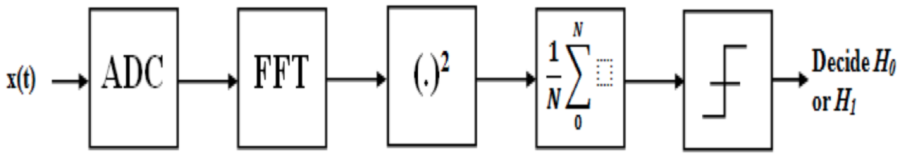


Fig. 3.2: Block diagram of an energy detector.

3.2 Fast Fourier Transform (FFT)

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) and its inverse. A Fourier transform converts time (or space) to frequency and vice versa; an FFT rapidly computes such transformations. An FFT computes the DFT and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster.

Let $x_0, ..., x_{N-1}$ be complex numbers. The DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi \frac{n}{N}} \qquad k = 0, \dots, N-1. \tag{3.23}$$

Evaluating this definition directly requires N^2 operations: there are N outputs X_k, and each output requires a sum of N terms. An FFT is any method to compute the same results in $N \log N$ operations. To illustrate the savings of an FFT, consider the count of complex multiplications and additions. Evaluating the DFT's sums directly involves N^2 complex multiplications and $N(N-1)$ complex additions. The well-known radix-2 Cooley-Tukey algorithm, for N a power of 2, can compute the same result with only $(N/2)\log_2(N)$ complex multiplications (ignoring simplifications of multiplications by 1 and similar) and $N \log_2(N)$ complex additions. In practice, actual performance on modern computers is usually dominated by factors other than the speed of arithmetic operations and the analysis is a complicated subject, but the overall improvement from N^2 to $N \log N$ remains [7].

3.2.1 Cooley-Tukey algorithm

By far the most commonly used FFT is the Cooley-Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2, along with N multiplications by complex roots of unity traditionally called twiddle factors. The most well-known use of the Cooley-Tukey algorithm is to divide the transform into two pieces of size $N/2$ at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general. Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley-Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT.

CHAPTER 4

SOFTWARE DEFINED RADIO

This chapter introduces the Software Defined Radio (SDR) and the Cognitive Radio purpose in detailed way. In particular it describes the Universal Software Radio Peripheral (USRP) and two kind of software (GNU Radio and Iris) able to implement a radio through the USRP. This instrument will be one of the main devices used in this work in order to capture the signal in the environment.

4.1 Definition

Software Radio attempts to create a flexible platform by using software instead of traditional hardware to perform operations on signals. The ideal SDR platform would use as little hardware as possible and let software deal with all of the processing. An ideal receiver might have just an antenna connected to an ADC. Samples would then be read from the ADC and software would handle all signal processing. There are many benefits to using software to process signals. Software is quick to compile and load, which gives iterative development a much higher cycle rate. Researchers and developers who are looking to experiment with different filter approaches can quickly prototype solutions and have real world results using SDR. Software that is developed for signal processing can easily be shared and adapted by others, allowing new work to re-use existing code. This code re-use is easily demonstrated by the GNU Radio community which is described in detail later. Using commodity Personal Computers (PCs) is also advantageous because platform developers can take advantage of the high performance to economy ratio. Fast super-scalar processors with multiple cores are becoming commodity resources. This make high power computing resources available at a good price. Although specific signal processing hardware may be faster, a general purpose software solution has a place in many radio applications. SDR has advantages to manufacturers because they only need to produce and support one hardware platform which can cut development costs. For service providers using SDR, they can quickly update and change their network with little hardware change. End users of SDR products will be able to communicate efficiently and have the most up to date features without having to buy a new hardware platform each time. The flexibility and reconfigurability of SDRs make them prime targets for military and disaster response applications. In these scenarios, an existing communication infrastructure may be damaged or not even exist at all. Having a communication device that can adapt to changing communication conditions would allow for better response and coordination [9].

4.1.1 Current uses of SDR

- Military uses
- Civilian disaster uses
- Amateur radios
- Home uses

4.1.2 Future uses of SDR

- Cell phones
- Wi-Fi
- Entertainment distribution
- Public Safety
- Broadcasting
- Digital Radio
- Digital Television

4.2 Cognitive Radio

The ability to reconfigure a communication network based on environmental conditions is the key to the field of Cognitive Radio. SDRs give the flexibility to change the underlying communication blocks used, while cognitive radios evaluate the operating environment and initiate the changes in the radio to maximize the goals of the radio. Of particular interest and one of the goals of the Cognitive Radio community is dynamic spectrum access. With more wireless technologies being developed, the available spectrum is becoming more and more limited. However the usage of the spectrum varies greatly with time and location. It is of great interest to primary users and secondary users of the spectrum on how to effectively share these underutilized bands. However, analysis of some of those bands show major under utilization. This suggest sample opportunity for new devices to make use of the under utilized spectrum as long as they do not interfere with the primary users of the spectrum.

There are two main approaches:

- **Spectrum Overlay**: dynamic spectrum access by secondary spectrum users that exploits spatial and temporal spectral opportunities in a non-interfering manner.
- **Spectrum Underlay**: a type of secondary spectrum access where radiated power limits, power spectral density limits, or modulation requirements on secondary transmissions protect primary users from interference.

4.3 UNIVERSAL SOFTWARE RADIO PERIPHERAL

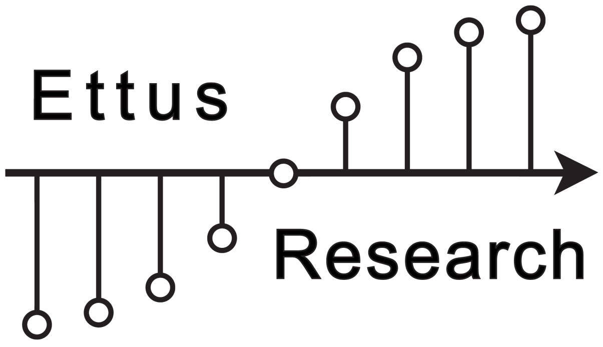


Fig. 4.1: Ettus Research logo.

The Universal Software Radio Peripheral (USRP) products are computer-hosted software radios [8].
The USRP product family includes a variety of models that use a similar architecture. A motherboard provides the following subsystems: clock generation and synchronization, FPGA, ADCs, DACs, host processor interface, and power regulation. These are the basic components that are required for baseband processing of signals. A modular front-end, called a daughterboard, is used for analog operations such as up/down - conversion, filtering,

and other signal conditioning. This modularity permits the USRP to serve applications that operate between DC and 6 GHz. USRPs connect to a host computer through a high-speed USB or Gigabit Ethernet link, which the host-based software uses to control the USRP hardware and transmit/receive data. The USRP family was designed for accessibility, and many of the products are open source. The board schematics for select USRP models are freely available for download and all USRP products are controlled with the open source UHD driver.

The original version of the USRP, developed by Matt Ettus (Fig. 4.1), was meant to be a simple and flexible platform for software radios. It contains ADCs, DACs, and a FPGA which can be programmed to do some signal processing and is used to implement the Digital Down Converter (DDC) block. The USRP also contains a USB2 controller that is used to send and receive samples from the computer. Fig. 4.2 shows a high level representation of the USRP.

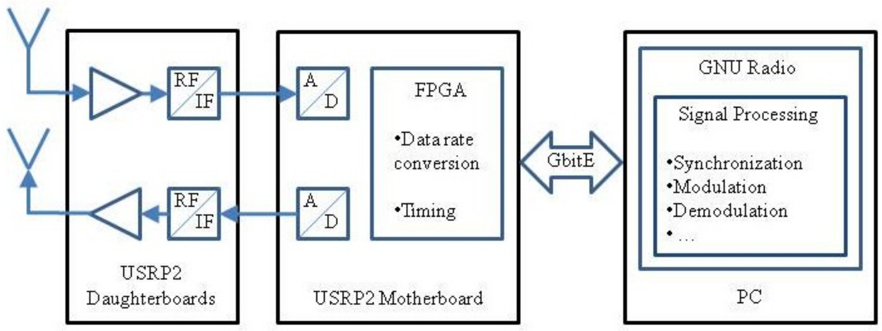


Fig. 4.2: USRP2 block diagram.

Daughterboards act as a Radio-Frequency (RF) front-end capable of tuning to different bands depending on the Daughterboard used. A daughterboard down converts a signal to an Intermediate Frequency (IF) that the ADC can handle. There are a variety of daughterboards for different bands, such as the XCVR2450 for frequencies 2.4-2.5GHz and 4.9-5.9GHz. On-board ADCs are 12-bit Analog Devices ADCs with a sample rate of 64 MS/s. The DACs are 14-bit and operate at 128 MS/s. The complex samples are composed of 16-bit In-phase and 16-bit Quadrature samples for a total of 4 bytes per sample. These samples must be transferred to the computer over the USB2 link for further processing by the host computer. The USB2 specification has a maximum rate of 60 MB/s. However, in real world tests this maximum is not achieved for sustained transfers. The USRP is limited to transferring at most 32 MB/s over the USB2 link. This means we must decimate the signal by at least a factor of 8 in order to keep a constant stream of samples. Decimation is achieved by DDCs implemented in the

FPGA. These DDCs center the signal at 0 and perform sample decimation. This decimation rate can be controlled by initialization parameters when interfacing with the USRP using software. A minimum decimation of 8 of 64 MS/s gives us 8 MS/s (complex samples), which gives us an equivalent sampling window of 8 MHz, i.e. less than 8 bluetooth channels.
This limitation led to choosing the USRP2 for multi-channel decoding [9].

Fig. 4.3: USRP N210.

The USRP2 released by Matt Ettus is a more powerful version of the USRP. The design is similar to the USRP and is compatible with the original daughterboards. Changes include using ADCs capable of 14-bit 100 MS/s, DACs capable of 16- bit 400 MS/s, and a Gigabit Ethernet (GigE) link instead of USB2. GigE has a maximum transfer rate of 125 MB/s which is equivalent to about 30 MS/s. Thus, the USRP2 has a sampling window maximum of 25 MHz when using a decimation rate of 4. This sampling window is wide enough to cover about 22 consecutive IEEE 802.15.1 channels in the 2.4 GHz band. Although there are 16 total channels in the ISM band, using the USRP2 is a good start to seeing the power of SDR. As technology improves, a SDR capable of sampling the entire band of bluetooth channels is a possibility.

In this work I used the USRP N210 (Fig. 4.3) with a XCVR2450 daughterboard mounted inside (Fig. 4.4). It is a high-performance transceiver intended for operation 2.4 GHz and 5.9 GHz range. In particular I'll observe the 2.4 GHz ISM band in order to monitor as much as possible Wifi and Bluetooth channels. The adopted antenna was a dual band 2.400-2.483 GHz and 4.9-5.8 GHz vertical antenna, with a gain of 3 dBi in the lower band.

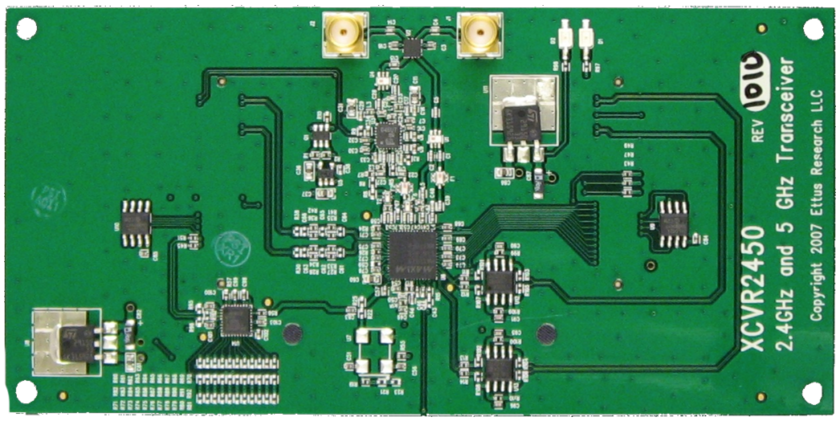

XCVR2450
2.4GHz and 5 GHz Transceiver
Copyright 2007 Ettus Research LLC
REV

Fig. 4.4: Daughterboard XCVR24250.

4.4 GNU Radio

GNU Radio [10] is an open-source effort to create software that uses a minimal hardware platform to implement a radio (Fig. 4.5). The project aims to make SDRs easy to program and accessible to a larger group of users. The community is active and new developments are constantly being released.

GNU Radio

Fig. 4.5: GNU Radio logo.

A block based programming model is taken with GNU Radio. Different signal processing blocks can be connected together in a signal processing pipeline. These blocks can be composed of other blocks in a recursive fashion. The ability to easily reuse existing code and swap blocks makes creating an application with GNU Radio quick. Since processing logic is implemented in software, to test changes a recompile is all that is needed. Blocks can be written in C++ or Python. Normally C++ provides the simpler block functionality and Python is used to compose and glue blocks together. In order to interface the C++ blocks with Python, a C++ wrapper and interface generator called SWIG is used. Using the combination of Python and C++ allows programmers to create logic that runs at the speed of compiled code, while easily connecting these blocks with a scriptable language. Recently GNU Radio has also added Multi-Core support and is able to thread different blocks onto different processing units, taking advantage of the growing number of multi-core computers. The community is constantly improving and adding features to make development of projects using GNU Radio even easier. Currently GNU Radio operates on data in a stream. Certain applications would be better suited if processed in a packetized manner. This is one of the next areas of development for GNU Radio. Having the software architecture to do signal processing is not useful without a way to get real-world samples of the signal. Some users have tried with some success to use the Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) built in to their sound cards. However the performance and capabilities of these components severely limited the potential projects. For this reason the GNU Radio community also created an open hardware platform to interface with the world.

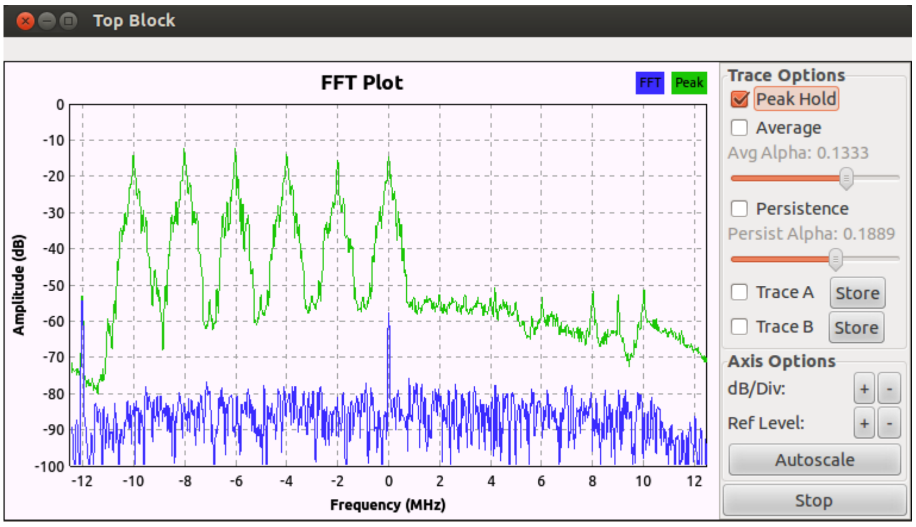


Fig. 4.6: Visualizing Bluetooth FHSS technique through GRC.

GNU Radio performs all the signal processing. You can use it to write applications to receive data out of digital streams or to push data into digital streams, which is then transmitted using hardware. GNU Radio has filters, channel codes, synchronization elements, equalizers, demodulators, vocoders, decoders, and many other implemented blocks which are typically found in radio systems (Fig. 4.6). More importantly, it includes a method of connecting these blocks and then manages how data is passed from one block to another. Extending GNU Radio is also quite easy; if you find a specific block that is missing, you can quickly create and add it.

Since GNU Radio is software, it can only handle digital data. Usually, complex baseband samples are the input data type for receivers and the output data type for transmitters. Analog hardware is then used to shift the signal to the desired centre frequency. That requirement aside, any data type can be passed from one block to another - be it bits, bytes, vectors, bursts or more complex data types.

4.4.1 GNU Radio Companion

The acquisition process with the USRP is developed through the GNU-Radio Software, in particular we used GNU Radio Companion (GRC) [11]. GRC is a graphical tool for creating signal flow graphs and generating flow-graph source code, it is very intuitive and powerful thanks to the large community that every day improves the project.
It allows users to construct a graphical representation of a flow-graph by instantiating and inter-connecting blocks from the GNU Radio libraries. Ever since its invention, GRC has attracted a lot of interest from academia, research industry and hobbyist people; while at the same time, GRC has significantly reduced the development-cycle time by easing flow-graph and GUI generation and has greatly helped GNU Radio users in their understanding of the underlying GNU Radio framework/architecture by visualizing the underlying structure of interconnected DSP blocks.

4.4.2 Extending GNU Radio: Block Structure

Obviously it is possible to extend GNU Radio through writing personal blocks that can be used both in GNU Radio and in GRC like a stand-alone component to connect with the others. GNU Radio provides a useful wiki website, where there are all the information about installing and extending the software [12]. This section presents just an overview of a block structure and it underlines the powerful and the simplicity in designing.

The work function

To implement processing, the user must write a "work" routine that reads inputs, processes, and writes outputs.
An example work function implementing an adder in c++:

```
int work(int noutput_items,
         gr_vector_const_void_star &input_items,
         gr_vector_void_star &output_items)
{
  //cast buffers
  const float* in0 = reinterpret_cast<const float *>(
     input_items[0]);
  const float* in1 = reinterpret_cast<const float *>(
     input_items[1]);
  float* out = reinterpret_cast<float *>(output_items
     [0]);

  //process data
  for(size_t i = 0; i < noutput_items; i++) {
    out[i] = in0[i] + in1[i];
  }

  //return produced
  return noutput_items;
}
```

Parameter definitions:

- **noutput_items:** total number of items in each output buffer.
- **input_items:** vector of input buffers, where each element corresponds to an input port.
- **output_items:** vector of output buffers, where each element corresponds to an output port.

Some observations:

- Each buffer must be cast from a void* pointer into a usable data type.
- The number of items in each input buffer is implied by noutput_items
- The number of items produced is returned, this can be less than noutput_items

IO signatures

When creating a block, the user must communicate the following to the block:

- The number of input ports
- The number of output ports
- The item size of each port

An IO signature describes the number of ports a block may have and the size of each item in bytes. Each block has 2 IO signatures: an input signature, and an output signature.
Some example signatures in C++:

```
-- A block with 2 inputs and 1 output --

gr_sync_block("my_adder", gr_make_io_signature(2, 2,
    sizeof(float)), gr_make_io_signature(1, 1, sizeof(
    float)))
```

```

-- A block with no inputs and 1 output --

gr_sync_block("my␣source", gr_make_io_signature(0, 0,
    0), gr_make_io_signature(1, 1, sizeof(float)))

-- A block with 2 inputs (float and double) and 1
    output --

std::vector<int> input_sizes;
input_sizes.push_back(sizeof(float));
input_sizes.push_back(sizeof(double));

gr_sync_block("my␣block", gr_make_io_signaturev(2, 2,
    input_sizes), gr_make_io_signature(1, 1, sizeof(
    float)))
```

Some observations:

- Use gr_make_io_signature for blocks where all ports are homogenous in size.
- Use gr_make_io_signaturev for blocks that have heterogeneous port sizes.
- The first two parameters are min and max number of ports, this allows blocks to have a selectable number of ports at runtime.

4.4.3 Block types

To take advantage of the gnuradio framework, users will create various blocks to implement the desired data processing. There are several types of blocks to choose from:

- Synchronous Blocks (1:1)
- Decimation Blocks (N:1)
- Interpolation Blocks (1:M)
- General Blocks (N:M)

Synchronous Block

The sync block allows users to write blocks that consume and produce an equal number of items per port. A sync block may have any number of inputs or outputs. When a sync block has zero inputs, its called a source.

When a sync block has zero outputs, its called a sink. An example sync block in c++:

```
#include <gr_sync_block.h>

class my_sync_block : public gr_sync_block
{
public:
  my_sync_block(...) :
    gr_sync_block("my␣block",
                  gr_make_io_signature(1, 1, sizeof(
                     int32_t)),
                  gr_make_io_signature(1, 1, sizeof(
                     int32_t)))
  {
    //constructor stuff
  }

  int work(int noutput_items,
           gr_vector_const_void_star &input_items,
           gr_vector_void_star &output_items)
  {
    //work stuff...
    return noutput_items;
  }
};
```

Some observations:

- noutput_items is the length in items of all input and output buffers
- an input signature of gr_make_io_signature(0, 0, 0) makes this a source block
- an output signature of gr_make_io_signature(0, 0, 0) makes this a sink block

Decimation Block

The decimation block is another type of fixed rate block where the number of input items is a fixed multiple of the number of output items. An example decimation block in c++:

```
#include <gr_sync_decimator.h>

class my_decim_block : public gr_sync_decimator
{
```

```
public:
  my_decim_block(...) :
    gr_sync_decimator("my␣decim␣block",
                      in_sig,
                      out_sig,
                      decimation)
  {
    //constructor stuff
  }

  //work function here...
};
```

Some observations:

- The gr_sync_decimator constructor takes a 4th parameter, the decimation factor.
- The user should assume that the number of input items = noutput_items*decimation.

Interpolation Block

The interpolation block is another type of fixed rate block where the number of output items is a fixed multiple of the number of input items. An example interpolation block in c++:

```
#include <gr_sync_interpolator.h>

class my_interp_block : public gr_sync_interpolator
{
public:
  my_interp_block(...) :
    gr_sync_interpolator("my␣interp␣block",
                         in_sig,
                         out_sig,
                         interpolation)
  {
    //constructor stuff
  }

  //work function here...
};
```

Some observations:

- The gr_sync_interpolator constructor takes a 4th parameter, the interpolation factor.

- The user should assume that the number of input items = noutput_items/interpolation.

Basic Block

The basic block provides no relation between the number of input items and the number of output items. All other blocks are just simplifications of the basic block. Users should choose to inherit from basic block when the other blocks are not suitable.
The adder revisited as a basic block in c++:

```
#include <gr_block.h>

class my_basic_block : public gr_block
{
public:
  my_basic_adder_block(...) :
    gr_block("another␣adder␣block",
             in_sig,
             out_sig)
  {
    //constructor stuff
  }

  int general_work(int noutput_items,
                   gr_vector_int &ninput_items,
                   gr_vector_const_void_star &
                      input_items,
                   gr_vector_void_star &output_items)
  {
    //cast buffers
    const float* in0 = reinterpret_cast<const float
        *>(input_items[0]);
    const float* in1 = reinterpret_cast<const float
        *>(input_items[1]);
    float* out = reinterpret_cast<float *>(
        output_items[0]);

    //process data
    for(size_t i = 0; i < noutput_items; i++) {
      out[i] = in0[i] + in1[i];
    }

    //consume the inputs
    this->consume(0, noutput_items); //consume port 0
        input
    this->consume(1, noutput_items); //consume port 1
        input
    //this->consume_each(noutput_items); //or shortcut
         to consume on all inputs

    //return produced
    return noutput_items;
  }
};
```

Some observations:

- This class overloads the general_work() method, not work().
- The general work has a parameter: ninput_items.
- ninput_items is a vector describing the length of each input buffer.
- Before return, general_work must manually consume the used inputs.
- The number of items in the input buffers is assumed to be noutput_items.
- Users may alter this behaviour by overloading the forecast() method.

Obviously there are many other types of blocks, here I reported just the structure and some examples of the most famous blocks. For any additional informations it is possible to consult the bibliography.

4.5 Iris

Iris is a open-source software totally developed at CTVR / The Telecommunications Research Centre, "Trinity College", Dublin 2, Ireland. I had the pleasure to know this software during my STSM and I report just a short description of this and of its potentialities. In the bibliography it can find many links and additional materials about it [13] [14] [15].

Iris is a software architecture for building highly reconfigurable radio networks. It has formed the basis for a wide range of dynamic spectrum access and cognitive radio demonstration systems presented at a number of international conferences between 2007 and 2010. These systems have been developed using heterogeneous processing platforms including general-purpose processors, field-programmable gate arrays and the Cell Broadband Engine. Focusing on runtime reconfiguration, Iris offers support for all layers of the network stack and provides a platform for the development of not only reconfigurable point-to-point radio links but complete networks of cognitive radios.

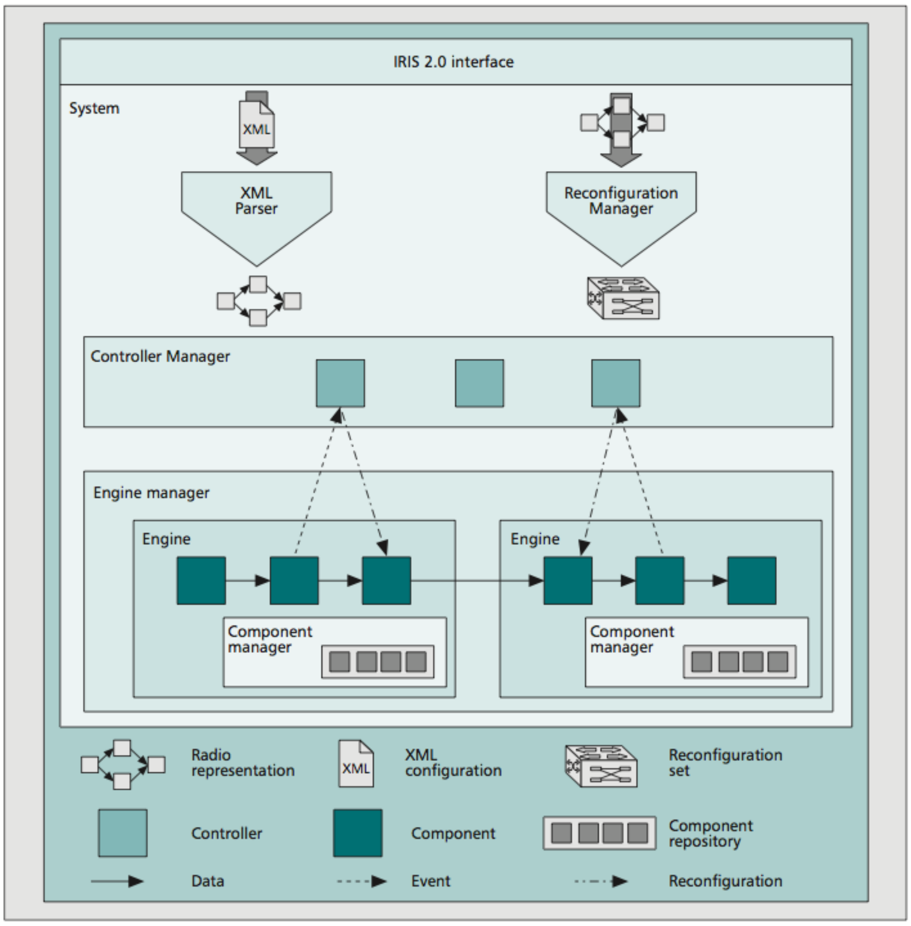


Fig. 4.7: Structure of Iris.

4.5.1 Reconfiguration

In contrast with software architectures such as GNU Radio, Iris is designed specifically to support maximum reconfigurability while the radio is running. This permits a network node to carry out reconfigurations seamlessly in response to observed changes in the operating environment. This reconfigurability is realized through a number of mechanisms that were built into the Iris architecture. The first of these is the component parameter. When implementing an Iris component, the designer can choose to expose a number of parameters. While the radio is running, these parameters can be dynamically reconfigured to adjust the operation of the component. Another powerful way in which component parameters can be used is to implement data-flow switching. A switch component can be used to direct the flow of data in the system to one of two or more signal processing branches. A parameter exposed on the component specifies the active output port to which data is written, and thus the active signal processing branch. This approach can be used, for example, in a receiver with independent receive chains for demodulation, and for signal detection and classification. When searching for a signal of interest, the flow of data is directed to the processing chain for signal detection and classification. When a signal is detected and classified, the switch component is reconfigured to direct the flow of data to the demodulation chain for data extraction.

In addition to the mechanisms provided to support parametric and structural reconfiguration, the Iris architecture provides specific support for the triggers which determine when these reconfigurations must take place. The first type of trigger that can occur within a radio is internal and typically takes place in response to a change detected by one of the radio components. In order to support such a trigger, Iris provides support for component events. These events are specified by the component designer and may be triggered by the component at any time. The elements of the Iris architecture with responsibility for listening for and responding to these events are the controllers. Controllers have a global view of the running radio and are capable of reconfiguring all aspects of it in response to component events. The Controller Manager is illustrated in Fig. 4.7, and is responsible for loading and managing the life cycle of controllers within Iris. In addition to reconfiguring component parameters, controllers can adjust the structure of a running radio by inserting and removing components. Like components, controllers are implemented in portable C++ and are loaded into a radio according to the XML configuration file. A key advantage of using controllers in this way is the avoidance of inter-component dependencies. By ensuring that components remain independent of one another, maximum reusability can be achieved. Controllers may consist of simple reconfiguration responses to predefined events or they may be much more complex entities, listening for multiple events, monitoring the state of the overall radio, reconfiguring many different aspects of it when necessary, and learning over time. In this way a controller can be used to implement a cognitive engine which drives the operation of the entire radio [16].

4.5.2 Iris Engines

One of the motivations behind the design of the Iris architecture was to move from experimenting with simple point-to-point links toward larger networks of cognitive radio nodes. The support provided for not just the PHY layer but also higher layers of the network stack is something that differentiates Iris from other architectures such as GNU Radio. By examining the constituent parts of a node within a cognitive network, we can identify a number of domains, each with different requirements for reconfiguration, datapassing, and execution. Implementing an architecture that caters for just one of those domains will lead to reduced efficiency and greater development challenges. Within the Iris architecture, the concept of a modular domain engine is used. The Iris engine encapsulates one or more components of the overall data flow graph of the node and defines the datapassing, execution, and reconfiguration semantics for those components. A radio implemented in Iris may consist of one or more of these engines. We define three domains within a cognitive network node and provide an engine for each. These domains are the scheduled PHY, the flexible PHY, and the network stack [16].

4.5.3 Simple Radio implementation

I report a simple radio code in xml composed by only two blocks, Iris doesn't provide a friendly Graphical User Interface (GUI), anyway the code designing is very intuitive:

```
<?xml version="1.0" encoding="utf-8" ?>

<softwareradio name="Radio1">

  <engine name="phyengine1" class="phyengine">

    <component name="usrprx1" class="usrprx">
      <parameter name="frequency" value="2412000000"/>
      <parameter name="rate" value="2500000"/>
      <port name="output1" class="output"/>
    </component>

    <component name="filerawwriter1" class="
       filerawwriter">
      <parameter name="filename" value="out.m"/>
      <port name="input1" class="input"/>
    </component>

  </engine>

  <link source="usrprx1.output1" sink="filerawwriter1.
     input1" />

</softwareradio>
```

At first it need to specify the used engine (in this case the "phyengine") and all the used blocks. In this example we have just an UHD block (it permits us to link the URSP with the software architecture) and a file raw writer that saves in a file the row In-Phase & In-Quadrature data coming from the USRP. Inside the block statement it is possible to set all the parameters, such as the center frequency, the sample rate, the gain, the file name, etc. At the end of the code we need to state the links between the blocks, in this case just the connection between the USRP source and the File sink.

CHAPTER 5

SPECTRUM ANALYZER

The second main instrument used in this work is the Spectrum Analyzer. The chapter 5 starts with a general description of the device and then it particularly talks about the used instrument, the Rohde&Schwartz FSVR 7 underling the used mode and the employed software.

5.1 Generality

A spectrum analyzer measures the magnitude of an input signal versus frequency within the full frequency range of the instrument. The primary use is to measure the power of the spectrum of known and unknown signals. It is important to note that the input signal is always electrical, so an appropriate transducer is needed to consider all kind of signals (acoustic, optics, etc.). By analyzing the spectra of electrical signals, dominant frequency, power, distortion, bandwidth and other several components, can be observed that are not easily detectable in time domain waveforms [17]. The classical mode of the instrument has frequency on the horizontal axis and the amplitude displayed on the vertical axis. Generally there are two types of spectrum analyzer, dictated by the methods used to obtain the spectrum of the signal:

- A **swept-tuned** spectrum analyzer uses a superheterodyne receiver to down-converter a portion of the input signal spectrum to the center frequency of the band-pass filter. With this type of architecture, the voltage-controlled oscillator is swept through a range of frequencies, enabling the consideration of the full frequency range of the instrument.
- A **FFT** (Fast Fourier Transform) spectrum analyzer computes the discrete Fourier transform (DFT), an efficient mathematical process that

transforms a waveform into the components of its frequency spectrum. Some spectrum analyzer, such a real-time spectrum analyzers, use a **hybrid** technique where the signal is first processed using super-heterodyne techniques and then analyzed using FFT techniques.

5.2 FFT based Spectrum Analyzer

The modern digital spectrum analyzers usually use the FFT technique. With a FFT based spectrum analyzer, the frequency resolution is $\Delta f = \frac{1}{T}$, the inverse to the time T over which the waveform is measured and transformed. Obviously it is necessary to sample the input signal with a sampling frequency that is at least twice the bandwidth of the signal, due to the Nyquist theorem. This can place considerable demands on the required analog-to-digital converter (ADC) and processing power for the Fourier transform, making FFT based spectrum analyzers limited in frequency range.

5.3 Realtime FFT: Hybrid approach

Most modern spectrum analyzers are now almost exclusively Hybrid Superheterodyne-FFT based. With increasing computing power, the newest analyzer generation was equipped with FFT filters for narrow bandwidths. Multiple narrowband FFTs were concatenated to a trace representing the selected frequency span. As the computing time for the FFTs was small compared to the settling time for narrow RBW filters, the FFT method provided a great speed advantage over the traditional sweep method. The latest spectrum analyzer generation makes excessive use of the FFT method for narrow resolution bandwidths. In addition, it introduces complex digital filters for wideband resolution filters, providing a factor of 25 in sweep speed increase, compared to earlier analog implementations. The measurement speed has increased dramatically, but one property has survived all evolution steps: it's impossible to detect signals between the end of one sweep and the start of the next one. This gap in data acquisition (Fig. 5.1), called "blind time", has decreased with each new spectrum analyzer generation, but it is still present.

Because real-time analysis means not to loose any information, thanks to the high resolution of ADCs available today, it's possible to combine these wideband ADCs with FFT algorithms implemented in delicate hardware (e.g. an FPGA).

The key-concepts are:

- Parallel sampling and FFT calculation: the data acquisition continues while the FFTs are performed (Fig. 5.2).

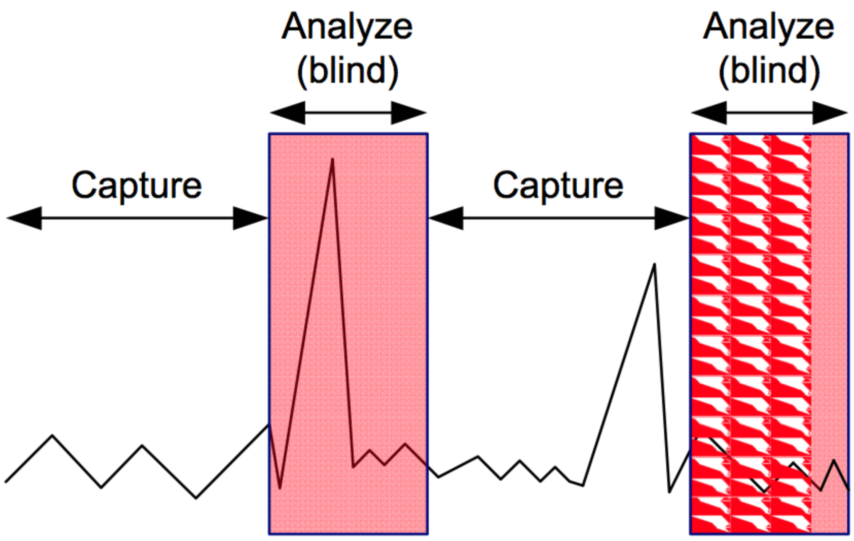


Fig. 5.1: Captures and blind time.

- Fast processing of FFT algorithms: The computation speed must be high enough to avoid that "stacks" of unprocessed data are being built up. Slow FFT computation will result in an overflow of the capture memory and a subsequent data loss (so a new blind time).

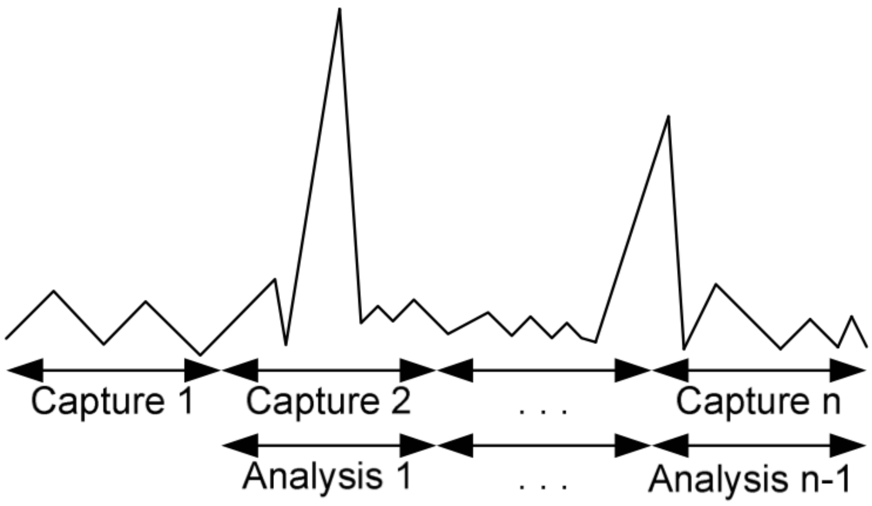


Fig. 5.2: Parallel capture

5.4 Rohde&Schwartz FSVR 7

Fig. 5.3: Rohde&Schwartz FSVR 7 Spectrum Analyzer.

The R&S FSVR 7 (Fig. 5.3) is a Real Time Spectrum Analyzer, operating in the frequency range of 10 Hz to 7GHz with a 40MHz maximum bandwidth [19]. The R&S FSVR 7 spectrum analyzer is based on the last approach described. As already stated earlier, the critical point behind real-time analysis is to run data acquisition and data processing in parallel. To achieve this, the digital backend of the R&S FSVR is equipped with a chain of powerful ASICs and FPGAs in combination with a large memory for captured data. This combination allows the instrument to process the data in several stages in a pipeline architecture. The last stage of the pipeline is the CPU, which reads the pre- processed data, applies the necessary scaling information and displays the resulting curve on the screen.

Fig. 5.4 shows the signal flow diagram from the A/D converter (ADC) to the display unit. The ADC is operated at a constant sampling rate of 128 MHz. The ADC streams raw data into the resampler and digital downconverter, which convert the input signal into a digital baseband, whose bandwidth is equal to the selected frequency span, and whose sampling rate fulfills the Nyquist criterion for this bandwidth. The ratio between complex baseband sample rate and selected frequency span is 1.2, meaning that e.g. a 40 MHz span is sampled with 50 complex MSamples per second. For smaller bandwidths, the sampling rate is automatically reduced. The sampling rate determines the number of samples which are available for analysis. After resampling, the data stream is transformed into the frequency domain by means of an FFT. Each FFT consists of 1024 so called bins or data points. The

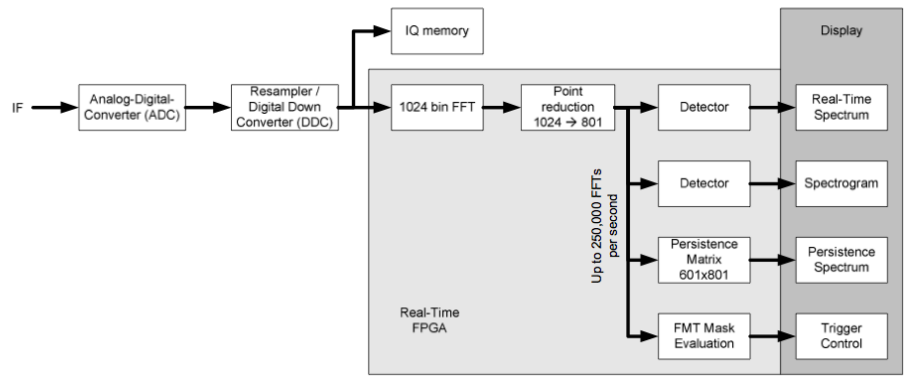

IQ memory
Display
IF
Analog-Digital-Converter (ADC)
Resampler / Digital Down Converter (DDC)
1024 bin FFT
Point reduction 1024 → 801
Detector
Real-Time Spectrum
Detector
Spectrogram
Up to 250,000 FFTs per second
Persistence Matrix 601x801
Persistence Spectrum
Real-Time FPGA
FMT Mask Evaluation
Trigger Control

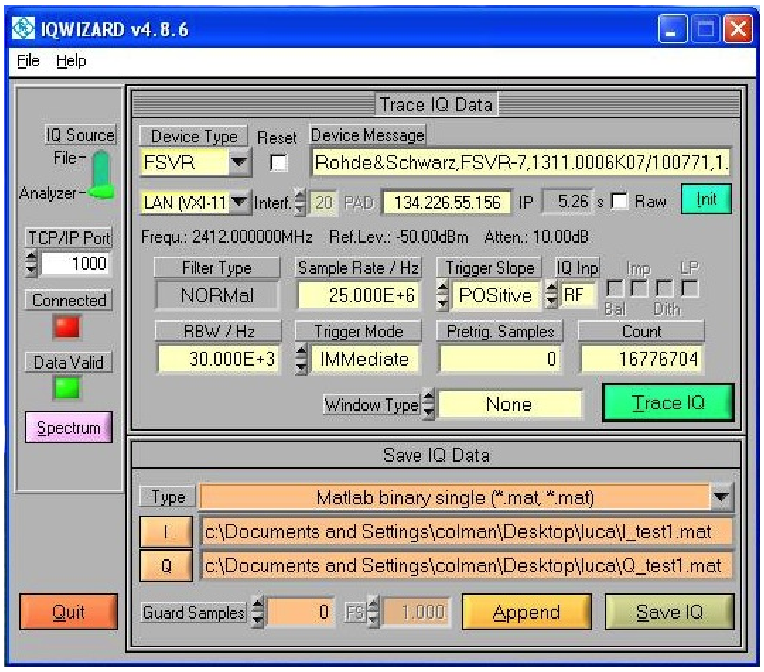
Fig. 5.4: FSVR7 signal flow diagram.

FPGA running the FFT algorithms delivers up to 250,000 FFTs per second. In parallel the resampled baseband data is written into the I/Q memory for additional offline (non real-time) post-processing, like e.g. zooming into a captured region or reading out the I/Q samples via LAN or GPIB. Note that the I/Q memory is implemented as a circular buffer which means that once the memory is full, the oldest samples will be overwritten [18].

5.5 R&S I&Q Wizard Software

IQWizard is a software tool for loading IQ data files in various formats or measuring IQ signals with the Spectrum Analyzer. The IQ data may be stored various file formats for further processing with signal analysis, simulation and generation tools such as MATLAB. It's possible to connect the computer running IQWizard directly to the instrument with a GPIB or LAN cable or establish the connection via Ethernet switch connected to DHCP server.

In this work the remote mode was used in order to capture the I&Q data in MATLAB format (Fig. 5.5 - 5.6). More details about the parameters and the purposes can be found in chapter 6.


IQWIZARD v4.8.6

File Help

Trace IQ Data

IQ Source File– Analyzer–

TCP/IP Port 1000

Connected

Data Valid

Spectrum

Device Type FSVR

Reset

Device Message Rohde&Schwarz,FSVR-7,1311.0006K07/100771,1.

LAN (VXI-11 Interf. 20 PAD 134.226.55.156 IP 5.26 s Raw Init

Frequ.: 2412.000000MHz Ref.Lev.: -50.00dBm Atten.: 10.00dB

Filter Type NORMal

Sample Rate / Hz 25.000E+6

Trigger Slope POSitive

IQ Inp RF

Imp LP Bal Dith

RBW / Hz 30.000E+3

Trigger Mode IMMediate

Pretrig. Samples 0

Count 16776704

Window Type None

Trace IQ

Save IQ Data

Type Matlab binary single (*.mat, *.mat)

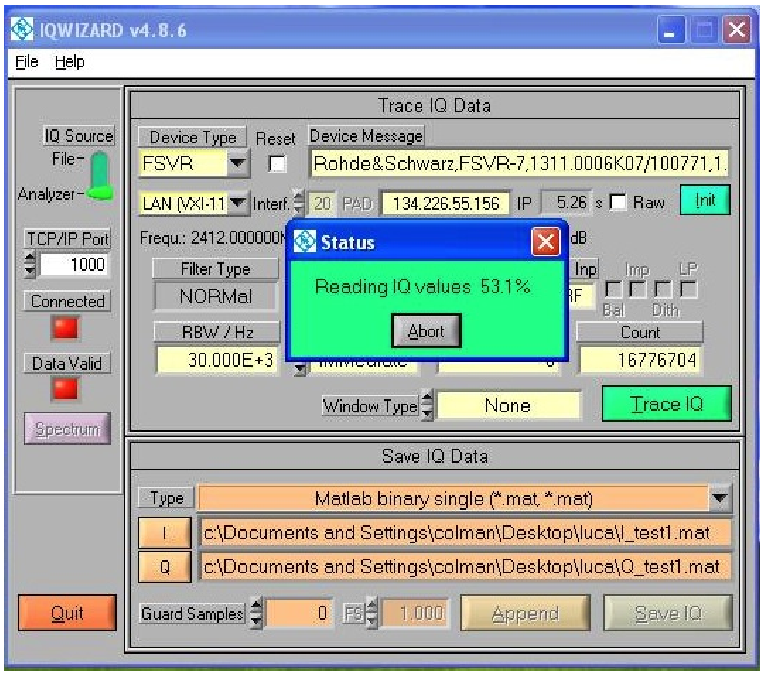
I c:\Documents and Settings\colman\Desktop\luca\I_test1.mat

Q c:\Documents and Settings\colman\Desktop\luca\Q_test1.mat

Guard Samples 0 FS 1.000 Append Save IQ

Quit

Fig. 5.5: R&S I&Q Wizard Software.


IQWIZARD v4.8.6

File Help

Trace IQ Data

IQ Source File– Analyzer–

TCP/IP Port 1000

Connected

Data Valid

Spectrum

Device Type FSVR

Reset

Device Message Rohde&Schwarz,FSVR-7,1311.0006K07/100771,1.

LAN (VXI-11 Interf. 20 PAD 134.226.55.156 IP 5.26 s Raw Init

Frequ.: 2412.000000M dB

Filter Type NORMal

RBW / Hz 30.000E+3

Imp LP Bal Dith

Count 16776704

Status

Reading IQ values 53.1%

Abort

Window Type None

Trace IQ

Save IQ Data

Type Matlab binary single (*.mat, *.mat)

I c:\Documents and Settings\colman\Desktop\luca\I_test1.mat

Q c:\Documents and Settings\colman\Desktop\luca\Q_test1.mat

Guard Samples 0 FS 1.000 Append Save IQ

Quit

Fig. 5.6: R&S I&Q Wizard Software during a capture.

CHAPTER 6

CORE RESEARCH

The experiments and captures have been done in CTVR / The Telecommunications Research Centre, "Trinity College", Dublin 2, Ireland, under the supervision of PhD student Paolo Di Francesco, and in the ACTS lab, Department of Information, Electronics and Telecommunication Engineering (DIET), "Sapienza, University of Rome", Italy.

Preamble and work summary

This work aims to deal with this issue by proposing a method for automatic recognition and classification of wireless technologies. The considered frequency band is the Industrial Scientific and Medical (ISM) 2.4 GHz band. The proposed approach consists of exploiting features of the Media Access Control (MAC) sub-layer of the various wireless technologies. Every network has its own particular MAC behavior, as defined in the standard that describes each of them. Through the study of these standards, a peculiar MAC behavior can be identified for each type of network, and a recognition and classification process can be carried out by evaluating these features. In particular, all it need is a time-domain frame diagram. This diagram shows the presence versus absence of a frame in every instant with no attention to the content of these frames. In fact, the packet informations are not relevant for the scope of this recognition, the only important thing is the frame pattern, a medium for revealing the technology that is currently in use, and thus the networks recognition. This means that there can be a maximum or minimum duration for certain types of frames, or even a fixed duration. The same rules can be determined for the silence gaps that fall between the frames. Moreover the standard may specify a regular and predetermined transmission of a frame (usually these are control frames needed to ensure

the correct system functionalities), or the transmission of acknowledgment frames after the reception of data frames. All these rules are specific for every single technology, i.e. each different network may present a MAC behaviour that is proper and peculiar of that technology.
In order to obtain the time-domain frame exchange diagram, all it need is a simple device: an energy detector(ED). Using the energy detection theory, the ED can compute the short-term energy that is present on the air interface. After defining a threshold value, all the consecutive short-term energy values that are higher than the threshold can be considered as frames. In this way, the frame diagram can be formed using energy detection [1].

I first capture the I and Q data signals through two different devices and then I'll process the files with the same MATLAB script to analyze the captures and to extract the features (Fig. 6.1). In this work we consider only three different scenarios: as a first experiment, I'll capture the data when only the wifi is active in the environment. As a second experiment I'll capture only Bluetooth signals. Finally I'll try to recognize both Bluetooth and Wifi signals in a mixture scenario where a network will be predominant on the other.

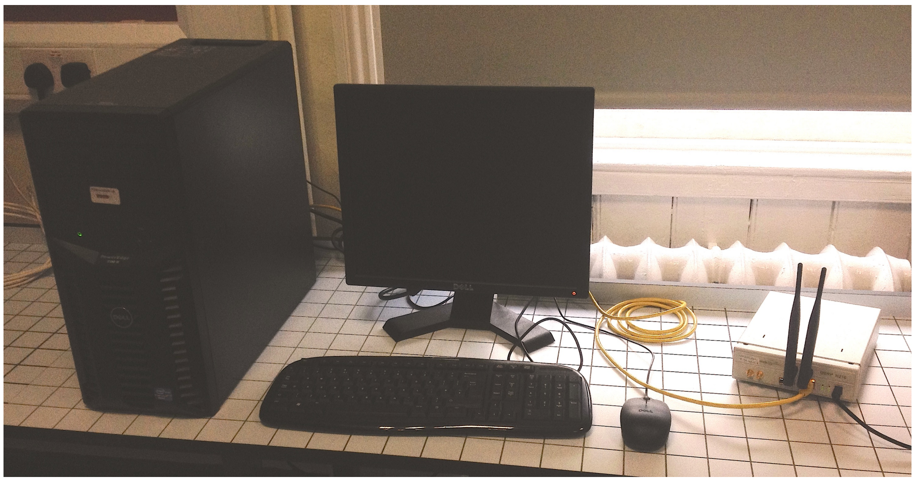


Fig. 6.1: Workstation used for the experiments.

6.1 The Universal Software Radio Peripheral

The first step is the acquisition of the signals through the Universal Software Radio Peripheral (USRP), in particular I used the USRP N210 (Fig. 6.2).

This hardware has recently gained growing attention by the research community given its low cost and open source vision. It is fundamental to

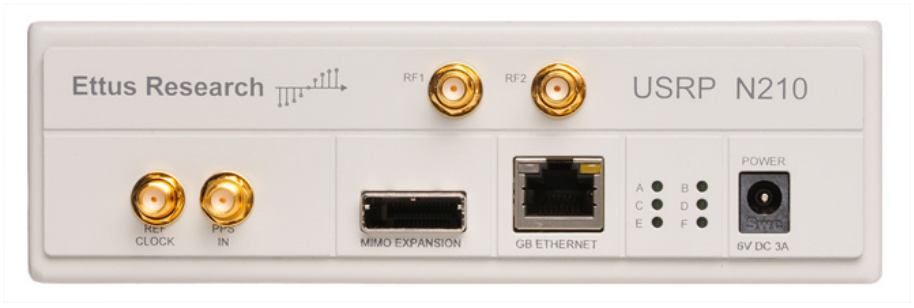


Fig. 6.2: USRP N210.

focus the attention in the communication interface between hardware and software, USRP2 uses the Gigabit Ethernet (GigE) link. GigE has a maximum transfer rate of 125 MB/s which is equivalent to about 30 MS/s. Thus, the USRP2 has a sampling window maximum of 25 MHz when using a decimation rate of 4. This sampling window is wide enough to cover about 22 consecutive IEEE 802.15.1 channels in the 2.4 GHz band. Although there are 16 total channels in the ISM band, using the USRP2 is a good start to seeing the power of SDR. As technology improves, a SDR capable of sampling the entire band of bluetooth channels is a possibility.
I used the USRP N210 with a XCVR2450 daughterboard mounted inside. It is a high-performance transceiver intended for operation 2.4 GHz and 5.9 GHz range. In particular I'll observe the 2.4 GHz ISM band in order to monitor as much as possible Wifi and Bluetooth channels. The adopted antenna was a dual band 2.400-2.483 GHz and 4.9-5.8 GHz vertical antenna, with a gain of 3 dBi in the lower band.

6.2 GNU Radio software

The acquisition process with the USRP is developed through the GNU Radio Software Defined Radio (SDR) platform [10], in particular we used GNU Radio Companion (GRC).

GRC (Fig. 6.3) is a graphical tool for creating signal flow graphs and generating flow-graph source code, it is very intuitive and powerful thanks to the large community that every day improves the project. GNU Radio is an open-source effort to create software that uses a minimal hardware platform to implement a radio (in this case the USRP). The project aims to make SDRs easy to program and accessible to a larger group of users. A block based programming model is taken with GNU Radio. Different signal processing blocks can be connected together in a signal processing pipeline. These blocks can be composed of other blocks in a recursive fashion. The ability to easily reuse existing code and swap blocks makes creating an

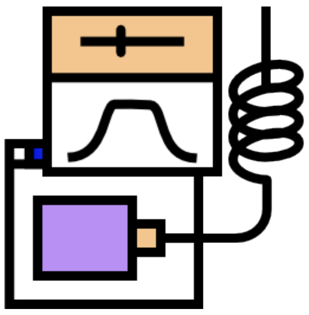


Fig. 6.3: GNU Radio Companion logo.

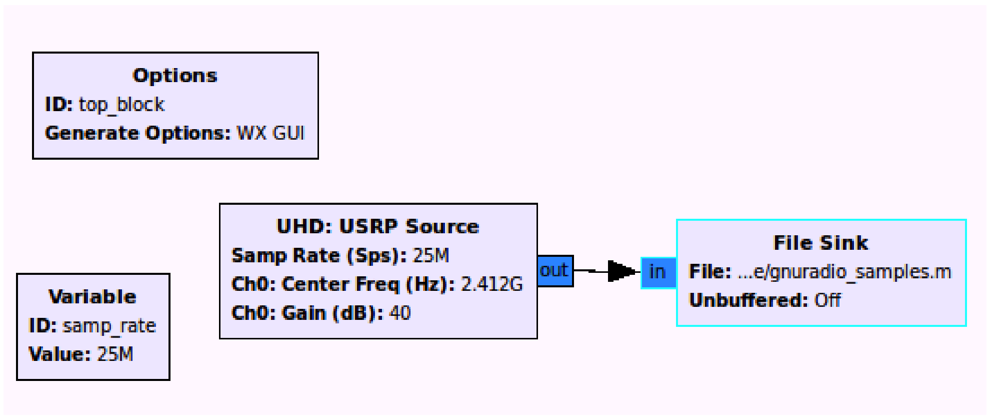
application with GNU Radio quick. Since processing logic is implemented in software, to test changes a recompile is all that is needed. Thanks to GRC we are able to develop a radio simply connecting the C++ process blocks through a useful GUI.

6.2.1 Flowgraph implementation

However, the main problem found is the sample rate, in fact I must use the higher sample rate possible for the USRP (25 MS/s) to obtain the maximum bandwidth (about 25MHz). This could create some problem during the capture because the limit of the Ethernet cable and the speed of the Hard Drive are not able to sustain the high stream of data coming from the USRP. Those limitations could generate overflow events hence falsify the measures. In order to minimize the overflows I developed a very simple flow graph in GRC (Fig. 6.4) composed by the UHD source (a interface block between the hardware and the software, i.e. between the USRP and GNU Radio) then connected to a File Sync Block. In this way I can save a I&Q raw data file of several seconds without leaks of information (no overflows) that it can be load in a further software to do an offline analysis.

6.3 R&S FSVR Real-Time Spectrum Analyzer

The second device used in the experiments is the Rohde&Schwarz FSVR7 Spectrum Analyzer [4], this instrument is more accurate and more expensive than the USRPs. It would be useful to compare the results in order to verify that the measures done with the USRP were right. I used the device in remote mode (Fig. 6.5) by a DELL workstation running WindowsXP (Intel Core Duo CPU 6600, 2.40 GHz, 2 GB of RAM), because this is the only way to exploit the spectrum analyzer as an I&Q recorder. We made use of a software provided by R&S called I&Q Wizard, a simple tool for loading I&Q signals.



Options
ID: top_block
Generate Options: WX GUI

UHD: USRP Source
Samp Rate (Sps): 25M
Ch0: Center Freq (Hz): 2.412G
Ch0: Gain (dB): 40

Variable
ID: samp_rate
Value: 25M

File Sink
File: ...e/gnuradio_samples.m
Unbuffered: Off

Fig. 6.4: GRC flowgraph.

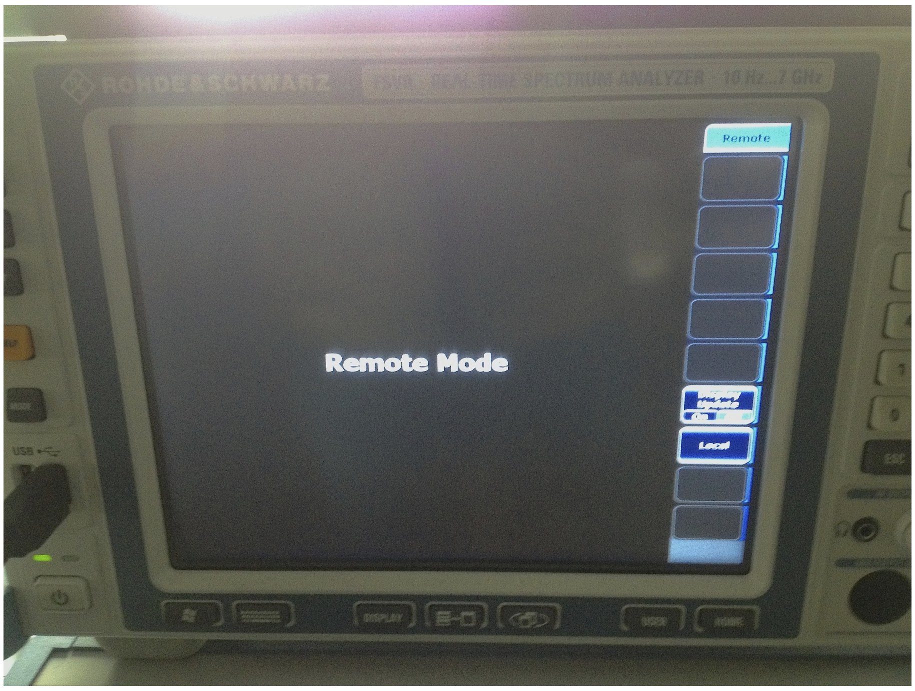


Fig. 6.5: Spectrum Analyzer in remote mode.

6.4 Acquisition parameters

I have set the two used instruments with equal acquisition parameters, there are only two difference: the first is that the output format of the Spectrum Analyzer is composed by two different raw data (for I and Q) that I have to combine subsequently through an ad-hoc instruction, differently to the USRP that has an output format based on a single matrix but with the necessity of a script conversion provided by the Ettus Research. The second difference is the time window length of the captures. Unfortunately I was able to capture only 16776704 samples with the spectrum analyzer. This is a limitation caused by the remote mode and by the nature of the instrument that it was not designed as an I&Q recorder. However the samples are enough to elaborate the signals, considering a sample rate of 25 MS/s we obtained a capture of 16776704 / 25e6 = 0,67106816 sec, a good window of time if we consider that we are looking for energy profiles in the order of microseconds.

Center freq:	4.412	GHz
Sample rate:	25	MS/s
Gain:	40	dB

I set the sample rate to the maximum to obtain a bandwidth of about 25 MHz (both the USRP and the Spectrum Analyzer admit 25 MS/s at most). The choice of setting the center frequency at that value is due to a maximization of the considered channels, in fact I was able to monitor the range of frequency between 2399.5 MHz and 2424.5 MHz. If I refer to the Bluetooth case, considering that the first channel is at 2.402 GHz and one channel has a width of about 1 MHz, I can observe about 22 Bluetooth channels (with a guard band of 2.5 MHz to eliminate the non-ideality of the filters).

6.5 The MATLAB script

I chose MATLAB to elaborate the data captures from the considered instruments. In particular, I used the MATLAB script "read_complex_binary.m" contained in the folder "/gnuradio-core/src/utils/" in every Ubuntu system with GNURadio installed to load the files coming from USRP. Regarding the Spectrum Analyzer I used the simple instruction "complex()" in MATLAB to combine the separate I&Q data in a single data file. In order to process the data file with MATLAB I used a laptop MacBook Pro with Processor 2,7 GHz Intel Core i7 and Memory 8 GB 1333 MHz DDR3 running Mac OS X 10.8.5.

The main structure of the code is reported at the end of this work (appendix B).

The first task of the script is to calculate the Fast Fourier Transform (FFT) of the signals. The only important aspect to consider is the choice of the FFT size. In order to explain this issue we have to recall some important MAC sub-layer features of the WiFi, especially the Short InterFrame Space (SIFS). For the considered IEEE standards the SIFS duration is 10 μs and in general it does not deviate from a mean value of 9 and 12 μs; moreover the timing of the Bluetooth networks is above 100 μs. So I need a time resolution capable to appreciate variations of few microseconds. The "new sample rate" after the FFT is the real Sample rate divided by the FFT size, choosing 64 Samples (S) as FFT Size I could obtain a clear resolution to have a considerable precision.

$$Sample\,Window = \frac{FFTsize}{SampleRate} = \frac{64\,S}{25\;10^6 S/s} = 2.56\;\mu s$$

6.5.1 Analysis based on Energy Detection

The energy detection theory was used because it provides a clean result in obtaining the Short Energy Diagram (SED) in time, thanks to this diagram I was able to analyze to presence vs the absence of the frames. The energy detector is known as a suboptimal detector, which can be applied to detect unknown signals as it does not require a prior knowledge on the transmitted waveform as the optimal detector does. As shown in Fig. 6.6, the 3D plot provides the Energy versus both the time and the frequency while we are interested to visualize the energy versus time considering the whole bandwidth that we have. Conceptually we only need a rotation of the figure with a compaction of all frequency contributions.

6.5.2 Short Energy Diagram

Once I made all the premises, I can start to analyze the signals through the use of the Short Energy Diagram (SED) in time. I obtained a SED (Fig. 6.7), simply calculating the mean of the samples at the same time for all frequencies considered, i.e. the average on the total 25 MHz bandwidth.

However, the diagram in this way is not readable and useful to our goal. Thus I must consider some thresholds to locate the areas of energy and to ignore any unwanted signals and the noise floor. Three different thresholds have been set (Fig.1), they are based on the minimum and maximum power received in reference to the Short Energy Diagram.

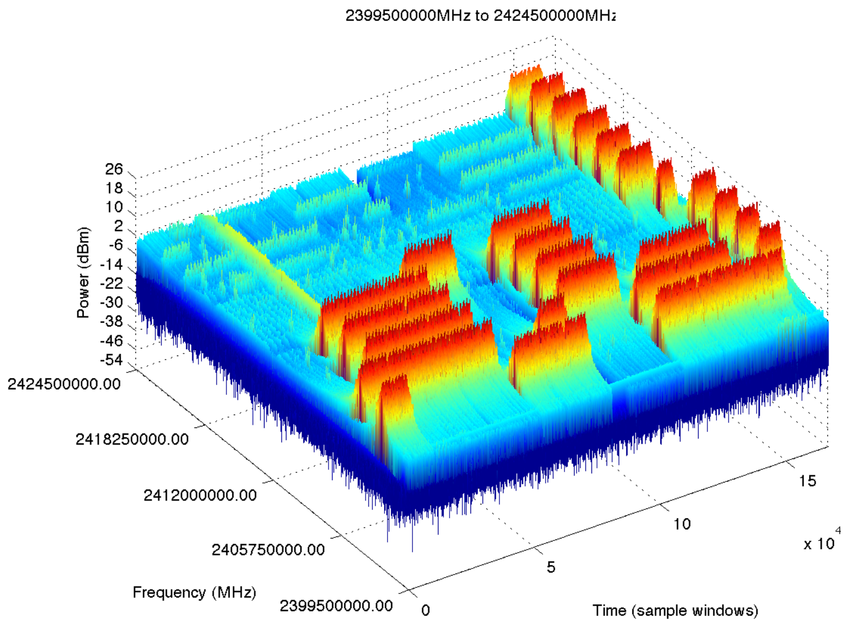


Fig. 6.6: 3D Energy Diagram.

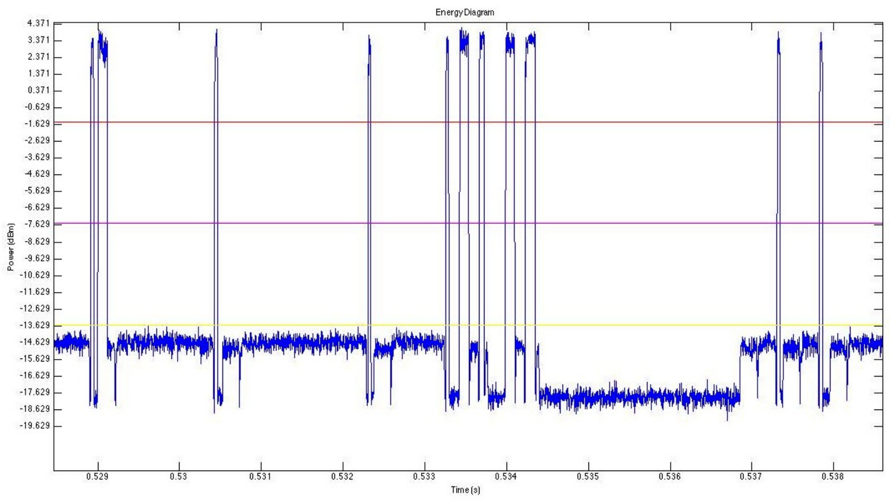


Fig. 6.7: Short Energy Diagram with thresholds.

$$PowerRange = PR = MaxPower - MinPower$$

$$LowerThreshold = \frac{PR}{4}$$

$$MediumThreshold = \frac{PR}{2}$$

$$HigherThreshold = \frac{3\,PR}{4}$$

In general, the medium threshold should be above of about 10 dB from the noise floor. For this reason we'll look at way to maintain only the medium and higher levels. Interesting enough is the plot of the SED with the only presence of three values (Fig. 6.8). This is could be useful to have a general vision of the distribution of power in time.

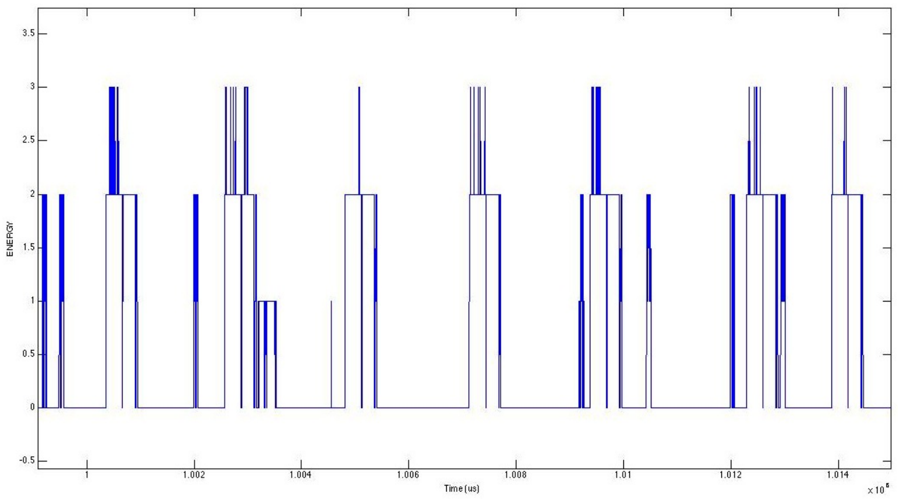


Fig. 6.8: Simplified Energy Diagram.

As mentioned above, there is a better way to work, indeed we are not concerned to discover the power or others peculiarities of the signals. The only important thing that we have to know it is the presence or the absence of a frame in relation to time. In order to do that, I converted the matrix to a new matrix where all the medium and higher samples corresponding to '1' (presence of frame) and all the lower and null samples corresponding to '0' (absence of frame). This processing also adds a good improvement to robustness of analysis; recalling an FFT size of 64 S, it is normal that the noise floor should be wider and more intrusive than a bigger FFT size (for example 1024 S). The reason has been identified in the average process: a bigger FFT size gives good result in the estimation of the noise floor, but it gives the same result in the recognition of the useful power for this project. This because more than half samples are located near the noise floor and the mean is able to minimize the fluctuations if there are many contributions to the sum. If I compare two SED (64 S and 1024 S FFT size respectively, Fig. 6.9 and 6.10) the difference is evident, in fact several measurements on real captures pointed out many problems in considering the lower threshold. Hence the choice to consider only the medium and higher thresholds because they highlighted a clear behavior also with a lower FFT size.

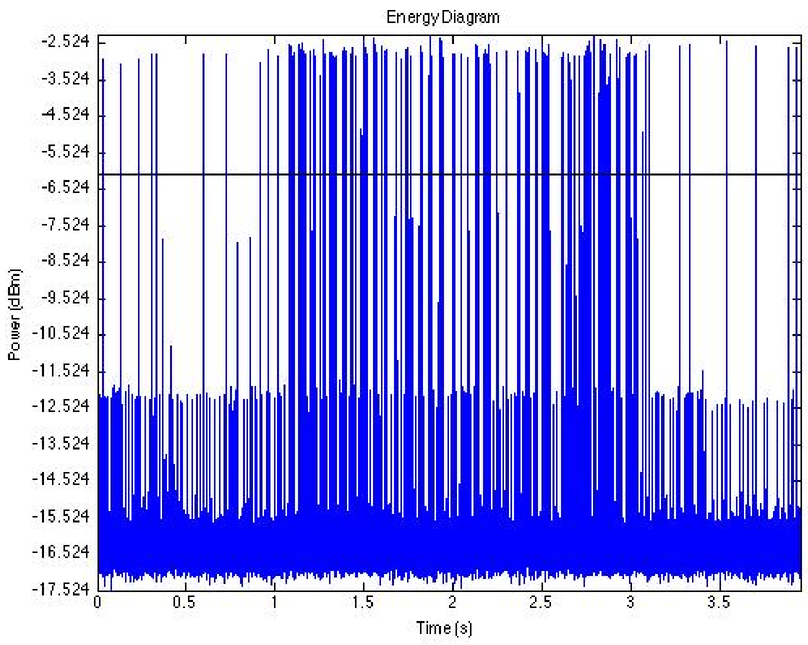


Fig. 6.9: Short Energy Diagram with FFT size of 64 S.

6.5.3 Frame Diagram

Now I can plot the frame diagram (Fig. 6.11), the central reference of the analysis. The main goal of the project is the extraction of the MAC features and the frame diagram in time is the best solution to discover and check these peculiarities. At first I observed that in general the frame diagram assumes a relevant periodic trend; thanks to the correction scripts and the threshold discriminations, we can note some regular patterns during a particular bluetooth or wifi transmission. Because we are looking for a system that should give us automatic informations about the networks, we have to work on some scripts designed to scan the entire frame matrix.

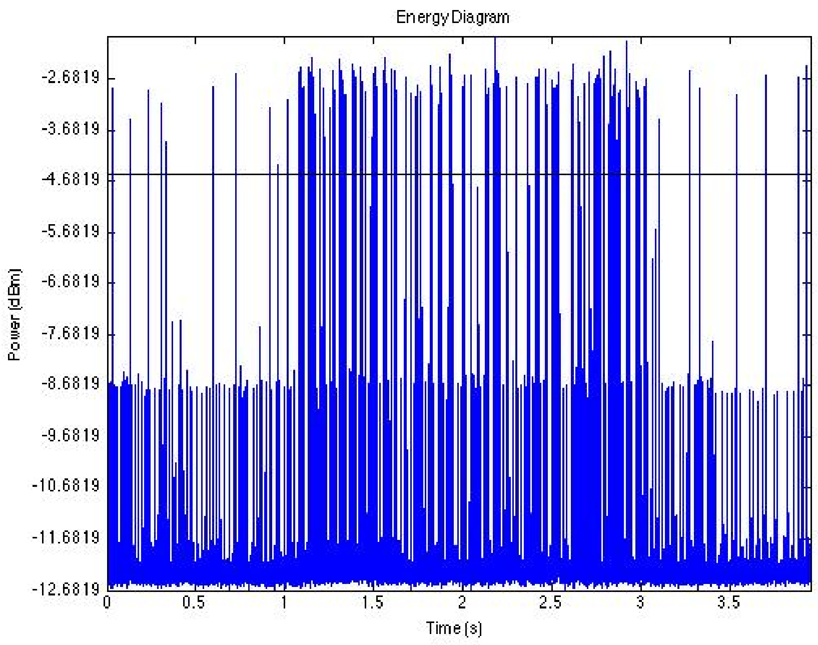


Fig. 6.10: Short Energy Diagram with FFT size of 1024 S.

6.5.4 Correction Script

I adopted two different types of correction designed to improve the accuracy of the frame pattern. The first revision is made before any recognition scripts and it consists in a correction of four anomalous cases: 101 - 010 - 1001 - 0110 (where '1' means a frame present and '0' means an not-frame present). Considering a sample window of 2.56 μs, it may be the presence of wrong information due to overflow events or threshold mistake. In this way I'm able to correct one or two potential wrong samples, corresponding to a time of 2.56 μs and 5.12 μs. Actually the corresponding times of one-two samples are shorter, it is reasonable not to assume the transition precisely at start of the sample window but, using a probabilistic view, it could be right to assume as an hypothesis the transition at middle of sample window. So, the time correction would become 1.28 - 2.56 μs. A second correction of three-four samples has been adopted in the code, however it only concerns the bluetooth recognition. The corresponding time of the second one are 5.12 - 7.68 μs and they are too long to elaborate a Wifi connection with a SIFS of only 10 μs. Otherwise, the bluetooth case trades energy profiles above to 50 μs, compatible with the revision just exposed.

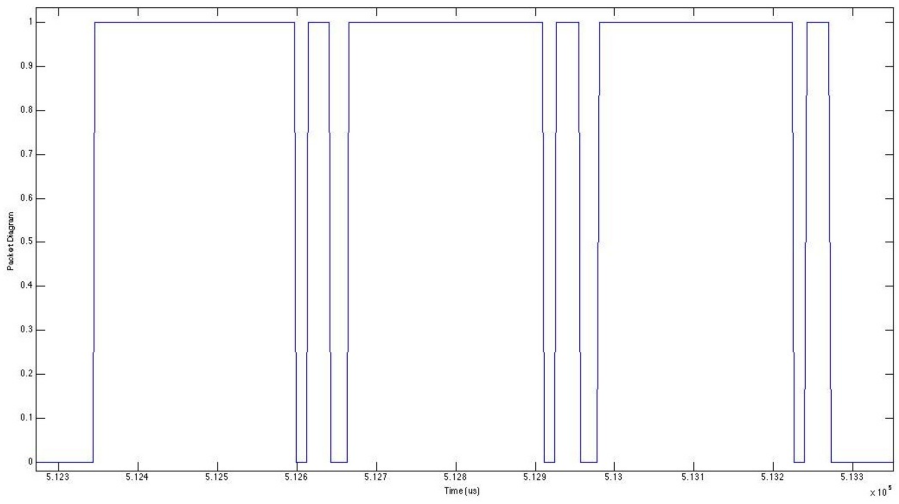
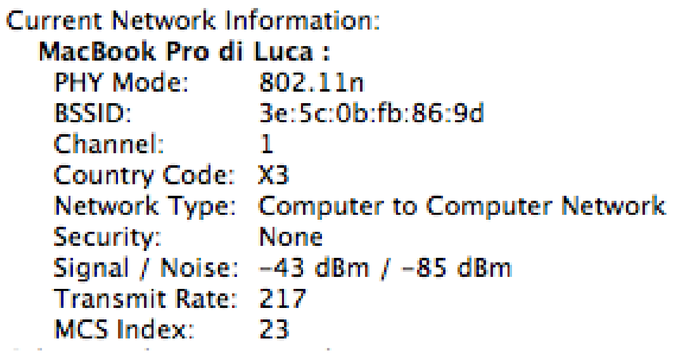


Fig. 6.11: Example of frame diagram (WiFi transmission).

6.6 Result: The WiFi Case (IEEE 802.11 b / g / n)

The most captures have been done using a MacBook Pro and iPad mini, in particular I created a computer-to-computer network to ensure that the standard type used was b, g or n and also to choose the channel. Regarding the channel I always set the channel one (Fig. 6.12) because it was the only one totally included in the 25 MHz of bandwidth considered.

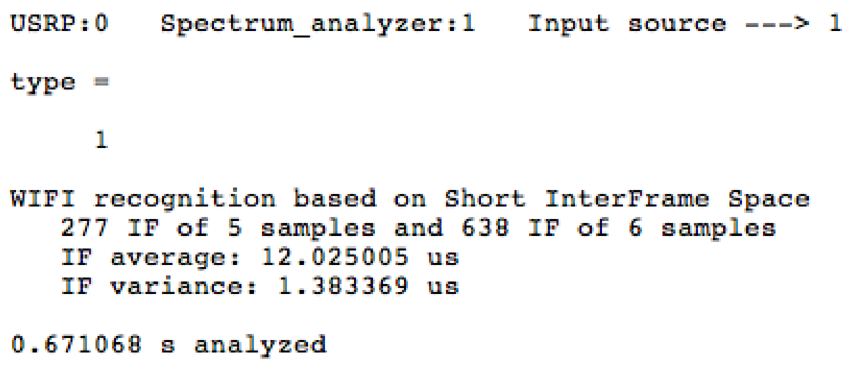


```
Current Network Information:
   MacBook Pro di Luca :
     PHY Mode:       802.11n
     BSSID:          3e:5c:0b:fb:86:9d
     Channel:        1
     Country Code:   X3
     Network Type:   Computer to Computer Network
     Security:       None
     Signal / Noise: -43 dBm / -85 dBm
     Transmit Rate:  217
     MCS Index:      23
```

Fig. 6.12: Wi-Fi network informations.

6.6.1 First analysis

The physical layer of a Wi-Fi network is different depending on the standard version [2], of course, but obviously even for the supported bit rate, whose value can be variable. I considered the SIFS as the start feature to recognize a WiFi network in the air. It is defined as the time duration between the end of the last symbol of the previous frame and the beginning of the first symbol of the following frame and it has a nominal value of 10 μs for the standard considered. I have done this verification through the use of a simple script based of a fast analysis of the entire data matrix. The idea behind is to find at first a possible data frame and then immediately verify the length of the InterFrame before the acknowledgment. We only considered frames composed at least by 90 samples as a valid data frames, i.e. the threshold of one valid frame has set to about 230 μs. I often obtained great result just by this simple process, I report an example of MATLAB print generated by the automatic script (Fig. 6.13).



```
USRP:0   Spectrum_analyzer:1   Input source ---> 1

type =

     1

WIFI recognition based on Short InterFrame Space
   277 IF of 5 samples and 638 IF of 6 samples
   IF average: 12.025005 us
   IF variance: 1.383369 us

0.671068 s analyzed
```

Fig. 6.13: Example of Script Stamp (WiFi transmission).

6.6.2 Second analysis

In addition to this first check, I tried to start a specific analysis based on the length of the data frames. In fact, sometimes it's difficult to be sure that the network recognized is WiFi only with InterFrame inspection. Therefore, every time the script finds a valid IF, it saves the duration of the previous frame in a new array. In this way I'm able to plot the time distribution of the found frames and the average of their duration. An interesting instrument to plot the frame length is the BoxPlot (Fig. 6.14).

Box plot

In descriptive statistics, a box plot is a convenient way of graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points.

Box plots display differences between populations without making any assumptions of the underlying statistical distribution: they are non-parametric. The spacings between the different parts of the box help indicate the degree of dispersion (spread) and skewness in the data, and identify outliers. In addition to the points themselves, they allow one to visually estimate various L-estimators, notably the interquartile range, midhinge, range, mid-range, and trimean. Boxplots can be drawn either horizontally or vertically.

The BoxPlot is a quick way of examining one or more sets of data graphically. Boxplots may seem more primitive than a histogram or kernel density estimate but they do have some advantages. They take up less space and are therefore particularly useful for comparing distributions between several groups or sets of data. Choice of number and width of bins techniques can heavily influence the appearance of a histogram, and choice of bandwidth can heavily influence the appearance of a kernel density estimate. As looking at a statistical distribution is more intuitive than looking at a boxplot, comparing the boxplot against the probability density function (theoretical histogram) for a normal N(0,σ^2) distribution may be a useful tool for understanding the BoxPlot [21].

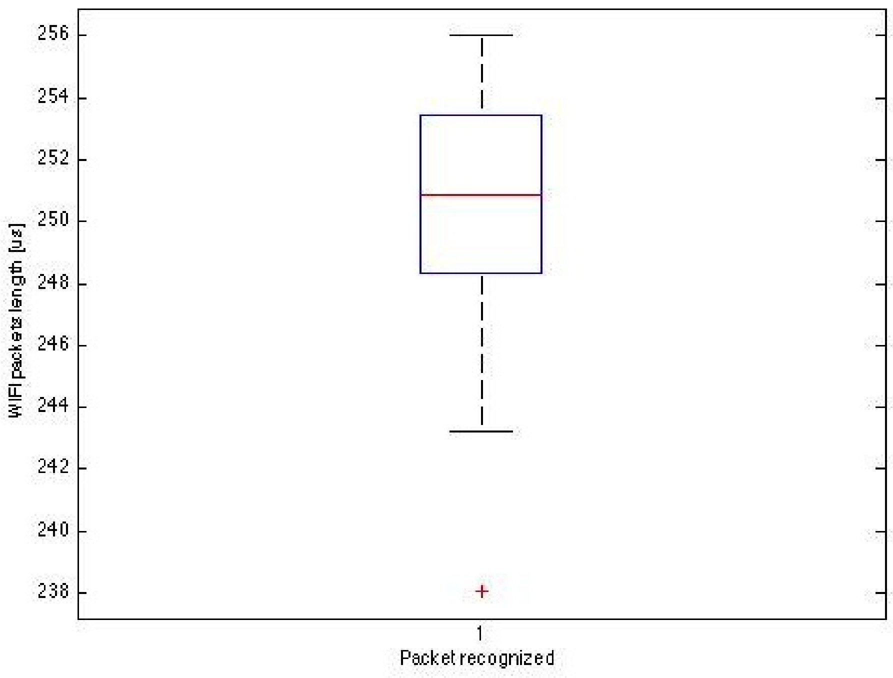


Fig. 6.14: BoxPlot of recognized WiFi frames.

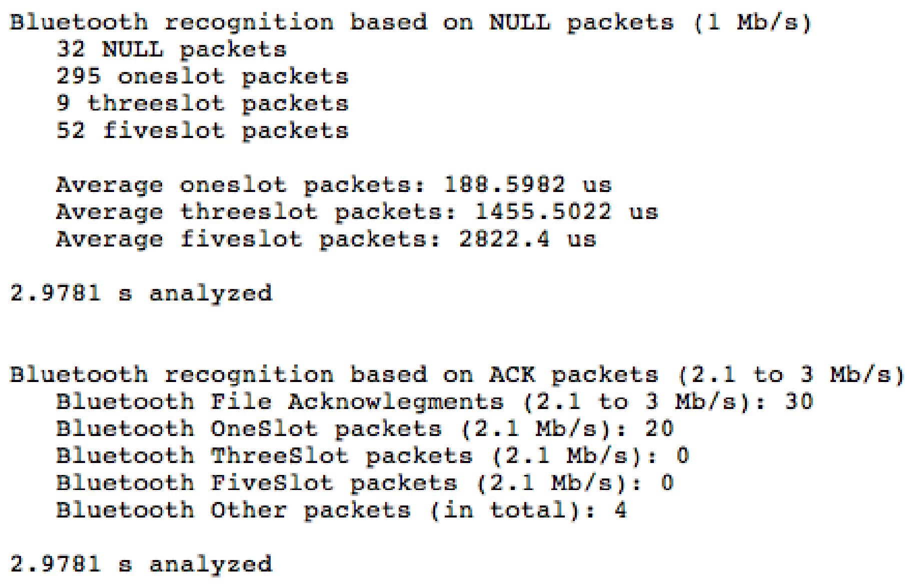
As just shown, the length of the WiFi frames during a transmission tends to be fixed, in particular it is about 250 - 255 μs if I consider a data transfer or a music streaming. It is possible to mark this result in all captures done from both USRP and Spectrum Analyzer. The boxes in the graphics don't extend to a lot of values but they remain around the expected value and the outliers may be caused by noise overlaps that corrupt the decision process at start.

6.7 Result: The Bluetooth Case (IEEE 802.15.1)

Regarding the Bluetooth recognition, I choose to take as reference the length of the frames. I considered the standard IEEE 802.15.1 [3] with the bluetooth versions 1.2 and 2.0; the main difference between the versions is the bit rate, respectively 1 Mb/s and 2.1 Mb/s considering the practical data transfer. A frame can last an odd number of time slots; in particular, there can be 1-time slot frames, 3-time slots frames and 5-time slots frames. A communication between the master device and a slave device is usually composed by alternate frames, since each device waits for a "return frame" (at least an acknowledgment) after sending a frame. Following these rules, imposed by the standard, it is clear that a Bluetooth MAC frame exchange pattern is characterized by frames that start every time slot duration, or at multiples of this value, if considering the multi-slot frames. Furthermore, many acknowledgment frames are expected; the NULL frame is the one used for acknowledgment, and it has a fixed length of 126 bits, that corresponds to a fixed duration of 126 μs considering the bit rate of 1 Mb/s. The other frames have also minimum and maximum durations, imposed by the standard. I report a table of considered values:

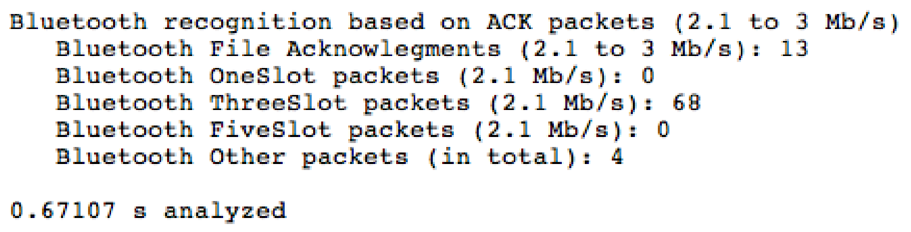
Bit Rate:	1 Mb/s	2.1 Mb/s
NULL(ACK):	126 μs	60 μs
OneSlotFrame:	126 to 366 μs	174 μs (max value)
ThreeSlotFrame:	1250 to 1622 μs	772 μs (max value)
FiveSlotFrame:	2500 to 2870 μs	1367 μs (max value)

As a first peculiarity, I checked the NULL frames, in fact they are the only ones able to ensure a probable presence of bluetooth networks. The time range of the OneSlot frames, for example, is too wide and other type of energy profiles could be assigned to this, especially if there are WiFi networks during the captures. Considering the fixed value of the NULLs, this approach appears robust and only after that, it may make sense to consider the longer frames. I took a lot of captures of bluetooth signals (v2.0), in particular I used a scenario where a smartphone Nokia N70 communicated to a MacBook Pro and a second scenario with a data transfer between two Nokia N70, both in transferring a file of about 5 MB. I also considered two different situation: the transition time between the device pairing and the data transfer, and the window time in the middle of the transfer. In the standard chosen, there are two different bit rate that produce two different length groups of frame: several captures and experiments have revealed that the bluetooth version v2.0 uses a prevalent bit rate of 1 Mb/s during the pairing and a bit rate of 2.1 Mb/s during the data transfer. For this reason I did two different script in order to recognize the features in both cases.



```
Bluetooth recognition based on NULL packets (1 Mb/s)
   32 NULL packets
   295 oneslot packets
   9 threeslot packets
   52 fiveslot packets

   Average oneslot packets: 188.5982 us
   Average threeslot packets: 1455.5022 us
   Average fiveslot packets: 2822.4 us

2.9781 s analyzed


Bluetooth recognition based on ACK packets (2.1 to 3 Mb/s)
   Bluetooth File Acknowlegments (2.1 to 3 Mb/s): 30
   Bluetooth OneSlot packets (2.1 Mb/s): 20
   Bluetooth ThreeSlot packets (2.1 Mb/s): 0
   Bluetooth FiveSlot packets (2.1 Mb/s): 0
   Bluetooth Other packets (in total): 4

2.9781 s analyzed
```

Fig. 6.15: USRP example: Transition time between pairing and transfer.



```
Bluetooth recognition based on ACK packets (2.1 to 3 Mb/s)
   Bluetooth File Acknowlegments (2.1 to 3 Mb/s): 13
   Bluetooth OneSlot packets (2.1 Mb/s): 0
   Bluetooth ThreeSlot packets (2.1 Mb/s): 68
   Bluetooth FiveSlot packets (2.1 Mb/s): 0
   Bluetooth Other packets (in total): 4

0.67107 s analyzed
```

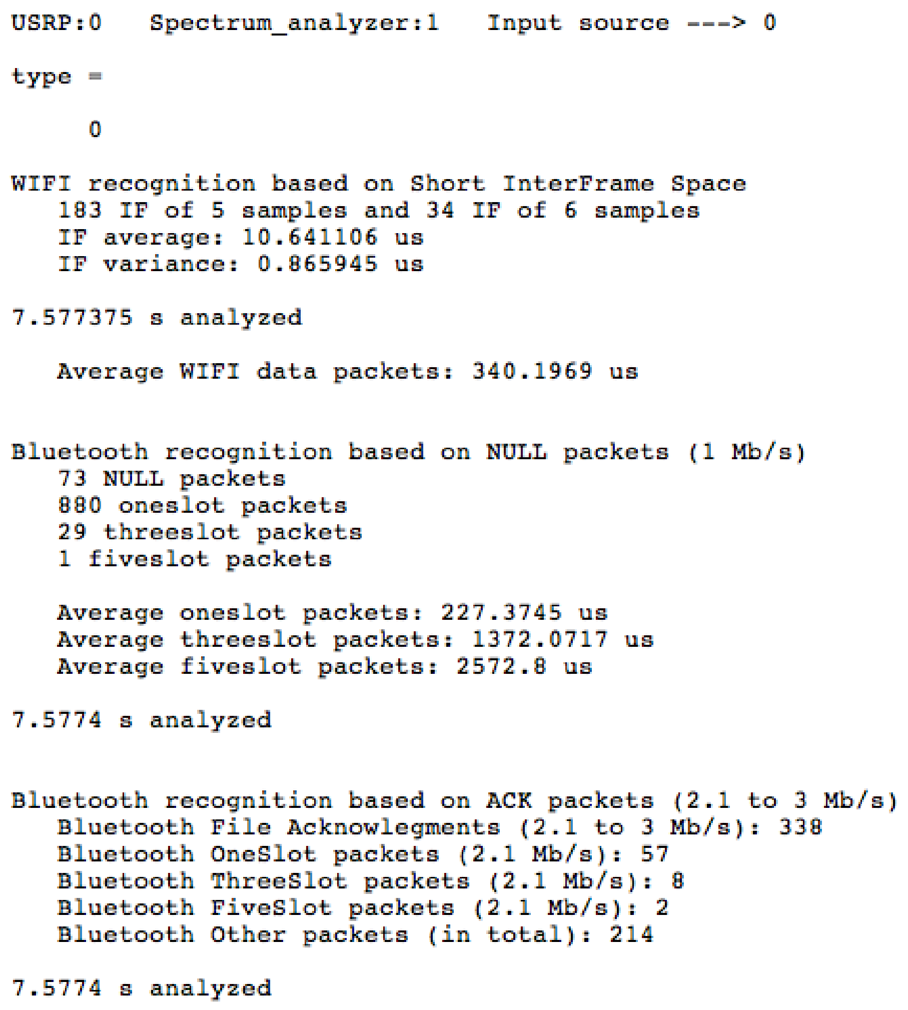
Fig. 6.16: Spectrum analyzer example: data transfer.

Thanks to this two examples (Fig. 6.15 and Fig. 6.16), it is possible to underline that, during the transition time, the script recognized both peculiarities, instead during the data transfer it found no acknowledgment of 1 Mb/s but only of others.
Finally it may be interesting a brief analysis of the frame lengths during the transfer: an important feature is that the frames, when it's possible, tend to assume the maximum length admitted by the standard. For this reason,

regarding to 2.1 Mb/s instead of considering all the range of value for the "slot" frames, I choose to locate only the maximum length frames. Besides the devices usually employ the same type of frame during a transmission, as it's possible to look in the examples (Fig. 6.15 OneSlot frames and Fig. 6.16 ThreeSlot frames). Referring to 1 Mb/s I tried to consider all the range of frames because I didn't note a particular frame behavior during the pairing.

6.8 Result: The mixed scenario

As described before, the last scenario considered it was the captures of both WiFi and Bluetooth signals. I obtained the best result in capturing a predominant bluetooth transition time with a WiFi pairing in background (about 40 cm far to the bluetooth devices). I report in the (Fig. 6.17) the whole script stamp: the results are not bad, however I found some problems when the networks operate at the same distance because the WiFi tends to dominate against the Bluetooth. The latter usually tries to change operative frequency in order to maximize the transfer, so the script is not able to recognize the features.



```
USRP:0   Spectrum_analyzer:1   Input source ---> 0

type =

     0

WIFI recognition based on Short InterFrame Space
   183 IF of 5 samples and 34 IF of 6 samples
   IF average: 10.641106 us
   IF variance: 0.865945 us

7.577375 s analyzed

   Average WIFI data packets: 340.1969 us


Bluetooth recognition based on NULL packets (1 Mb/s)
   73 NULL packets
   880 oneslot packets
   29 threeslot packets
   1 fiveslot packets

   Average oneslot packets: 227.3745 us
   Average threeslot packets: 1372.0717 us
   Average fiveslot packets: 2572.8 us

7.5774 s analyzed


Bluetooth recognition based on ACK packets (2.1 to 3 Mb/s)
   Bluetooth File Acknowlegments (2.1 to 3 Mb/s): 338
   Bluetooth OneSlot packets (2.1 Mb/s): 57
   Bluetooth ThreeSlot packets (2.1 Mb/s): 8
   Bluetooth FiveSlot packets (2.1 Mb/s): 2
   Bluetooth Other packets (in total): 214

7.5774 s analyzed
```

Fig. 6.17: USRP example: mixed scenario.

CHAPTER 7

CONCLUSION & FUTURE WORKS

While the WiFi results appear enough robust in the considered cases, the bluetooth case should still improve with other peculiarities to be extracted from the standard. In particular, interesting enough could be the investigation in timing of slot and in the silence gap between particular types of frames. It also may extend the argument considering the other standards and the other versions of both networks considered, always starting with common features and continuing with particular features in order to maximize the efficiency of the script.

Another branch for the future works may be the implementation of the energy detection process totally in real time. This would allow a faster analysis because the computational process off-line would become just an analysis of the matrix to extract the features without any kind of demanding operations. In this approach the captures should be smaller but more frequent in order to create a sort of "pipeline" process between GNU Radio and MATLAB. Unfortunately there are some limitations in using that approach, in fact the sample rate problem appeals again: the flow graph of energy detection in GRC should become more complex and the overflow events may be increasingly present. So, the only way to avoid the matter is to decrease the Sample Rate with the hope of better performance of devices in the future. In the appendix A, I report a first implementation using GRC, this could pave the way to move some processes up to now offline to a real-time analysis.

BIBLIOGRAPHY

[1] M.-G. Di Benedetto and S. Boldrini, *Towards Cognitive Networking: Automatic Wireless Network Recognition Based on MAC Feature Detection, In: H. Venkataraman, G.-M. Muntean, Cognitive Radio and its Application for Next Generation Cellular and Wireless Networks, Volume: 116,* DOI: 10.1007/978-94-007-1827-2_9, Springer, 2012, 239-257.

[2] IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999), pp C1-1184, June 12 2007.

[3] IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements-Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs), IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002), pp 0_1-580, 2005.

[4] S. M. Kay, *Fundamentals of Statistical Signal Processing. Vol. II: Detection Theory,* Prentice-Hall, Upper Saddle River, NJ, 1998.

[5] Sergio Benco, *Riconoscimento automatico di reti Bluetooth attraverso la piattaforma SDR Universal Software Radio Peripheral,* http://acts.ing.uniroma1.it/Archivio_tesi/benco/benco_tesi.pdf.

[6] ICWMC 2011 : The Seventh International Conference on Wireless and Mobile Communications, *Spectrum Sensing Measurement using GNU Radio and USRP Software Radio Platform,* IARIA, 2011.

[7] http://en.wikipedia.org/wiki/Fast_Fourier_transform.

[8] http://www.ettus.com/product/details/UN210-KIT

[9] Leslie Choong, *Multi-Channel IEEE 802.15.4 Packet Capture Using Software Defined Radio,* UCLA Networked & Embedded Sensing Lab, 03 2009, Pages 1-20. University of California, Los Angeles.

[10] http://gnuradio.org/redmine/projects/gnuradio/wiki

[11] http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

[12] http://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide

[13] http://www.crew-project.eu/iris

[14] http://www.softwareradiosystems.com/redmine/projects/iris

[15] http://www.softwareradiosystems.com/redmine/projects/iris/wiki/Wiki

[16] Sutton, P.D. ; Lotze, J. ; Lahlou, H. ; Fahmy, S.A. ; Nolan, K.E. ; Ozgul, B. ; Rondeau, T.W. ; Noguera, J. ; Doyle, L.E. , *Iris: an architecture for cognitive radio networking testbeds* Communications Magazine, IEEE (Volume: 48 , Issue: 9), Sept. 2010, pages: 114 - 122.

[17] http://en.wikipedia.org/wiki/Spectrum_analyzer/

[18] http://www.rohdeschwarz.de/file_15450/1EF77_0E.pdf

[19] http://www.rohde-schwarz.com/en/product/fsv-productstartpage_63493-10098.html

[20] Rozeha A. Rashid, M. Adib Sarijari, N. Fisal, S., K. S. Yusof, N. Hija Mahalin, *Spectrum Sensing Measurement using GNU Radio and USRP Software Radio Platform*, Faculty of Electrical Engineering, Universiti Teknologi, Malaysia.

[21] http://en.wikipedia.org/wiki/Box_plot

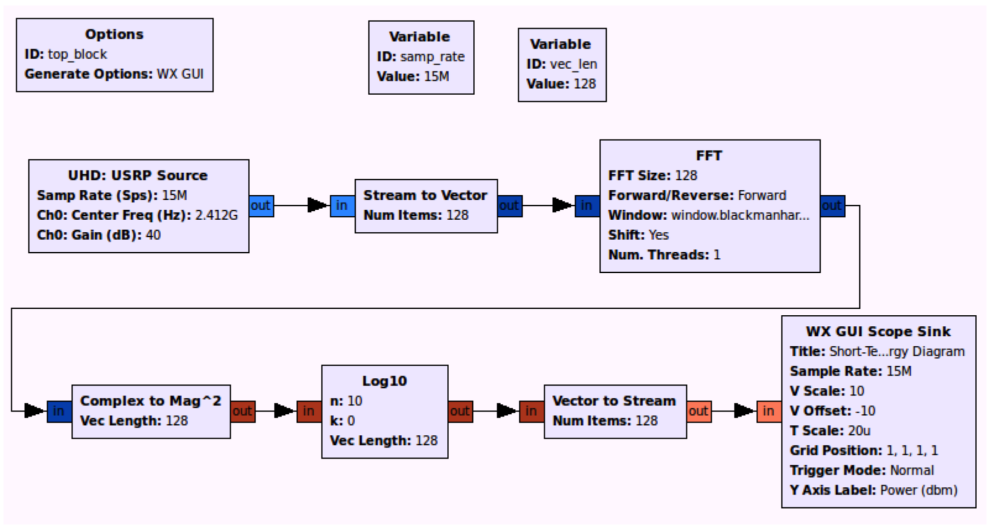
APPENDIX A

REAL TIME ENERGY DETECTOR

This appendix regards an implementation totally in GRC of an energy detection of the ISM 2.4 GHz band.

The sample rate considered is 15 MS/s and it was the best solution not to obtain too many overflows running the software, for more informations about this aspect it's possible to consult the chapter 6, in particular the section 6.2.1.
The flow graph is reported in Fig. A.1, the basic idea is the same used in the MATLAB script presented in this report and it was founded again on the energy detection theory. The data stream coming from the USRP source (a interface block between the hardware and the software, i.e. between the USRP and GNU Radio) is converted in a vector of 128 items by a block called *gr.stream_to_vector*. Its task is to take a stream of items as its input and convert it into a stream of blocks containing nitems_per_block as its output. In this work, nitems_per_block is equal to the size of our FFT which is 128. Then, this signal will be pushed into the GNU radio FFT block, in this process the input of the signal processing block is in complex vector type and the output is also a complex vector. In the FFT block, windowing technique is used to optimize the FFT result. Windowing is a technique used to shape the time portion of the sampled signal. This is to minimize edge effects that will result in spectral leakage in the FFT spectrum and increases the spectral resolution the frequency domain result. The complex output of the FFT block will then be connected to the complex magnitude block named *gr.complex_to_mag*^{2}. This block takes a complex number as its input and gives the squared magnitude (in float format) of this number

as output. Then, the result of this block will be converted from the ADC value to the dBm units by using *gr.nlog10_ff* block [20]. Finally, the vector is converted again to a stream to visualize the Short Energy Diagram in time domain thanks to the use of the "WX GUI Scope Sink" (Fig. A.2), the mean instruction on the vector items is included in the conversion block.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 15M

Variable
ID: vec_len
Value: 128

UHD: USRP Source
Samp Rate (Sps): 15M
Ch0: Center Freq (Hz): 2.412G
Ch0: Gain (dB): 40

Stream to Vector
Num Items: 128

FFT
FFT Size: 128
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Complex to Mag^2
Vec Length: 128

Log10
n: 10
k: 0
Vec Length: 128

Vector to Stream
Num Items: 128

WX GUI Scope Sink
Title: Short-Te...rgy Diagram
Sample Rate: 15M
V Scale: 10
V Offset: -10
T Scale: 20u
Grid Position: 1, 1, 1, 1
Trigger Mode: Normal
Y Axis Label: Power (dbm)

Fig. A.1: Energy detector flow graph in GRC.

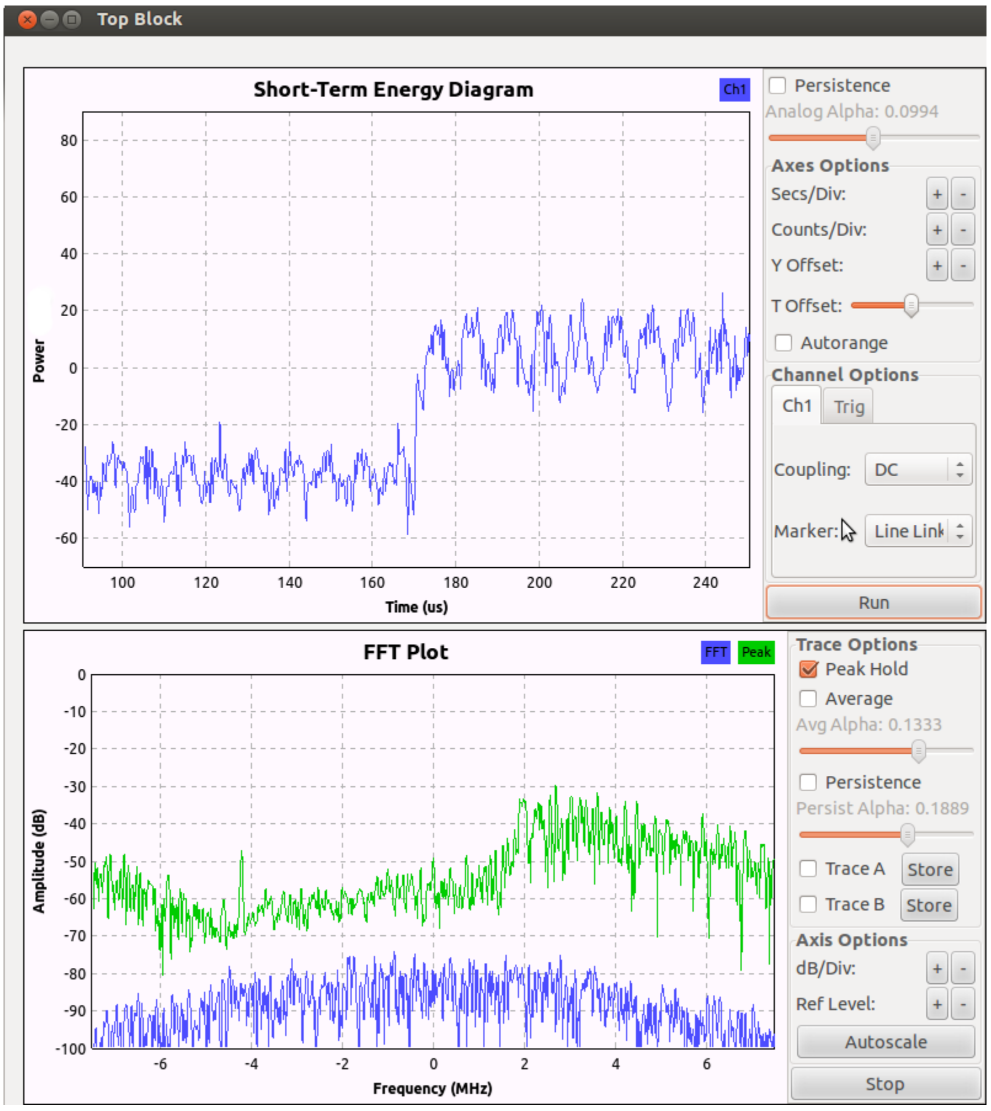


Fig. A.2: Real Time SED.

APPENDIX B

MATLAB CODE

I report the main structure of the MATLAB code used in this work to obtain the Short-Term Energy Diagram and to scan the captures coming from both USRP and Spectrum Analyzer in order to extract the MAC features for the automatic recognition of networks.

```
%% LUCA MILANI

%% Screen clening
clear all;
clc;
fclose('all');
close all hidden;

%% Source type
type = input('USRP:0␣␣␣Spectrum_analyzer:1␣␣␣Input␣
    source␣--->␣')

%USRP case
if (type == 0)

    basepath = '/Users/LucaMilani/Desktop/captures/';
        %reference folder
    filename = ['gnuradio_samplesBLUETOOTH0409.m'];
           %filename

    signal = read_complex_binary ([basepath filename])
        ;    %convertion script
```

```

%Spectrum_analyzer case
elseif (type == 1)

    basepath = '/Users/LucaMilani/Desktop/
       spectrum_captures/';   %reference folder
    filename = ['blue3'];       %filename
    ifile = [basepath filename '_I.mat'];
    qfile = [basepath filename '_Q.mat'];

    load(ifile);
    load(qfile);
    signal = complex(I,Q);       %complex data
    clear I;
    clear Q;
end

%% frequency range -- center freq: 2.412GHz ,
    BANDWIDTH 25MHz
startfreq = 2399500000;
endfreq = 2424500000;

%% FFT window size and Sample Rate
fftSize = 64;
sampleRate = 25e6;

%% Start the timer
tic;

%% Calculate the number of non-overlapping windows
len = length(signal);
numtraces = floor(len / fftSize);

%% Truncate the capture array
samples = signal(1:numtraces*fftSize);

%% Clear some memory
clear signal;

%% Convert for PSD calculations
samples = reshape(samples,fftSize,numtraces);       %%
    fftSize*numtraces matrix whose elements are taken
    column-wise from samples

```

```
%% Calculate PSDs
captures = 10*log10(fftshift(abs(fft(double(samples)))
    ,1));

clear samples;      %clear some memory

%% Start elaboration
%truncation of spectrum_analyzer array to eliminate a
    peak at the end
if (type == 0)
    captures2 = mean(captures);
elseif (type == 1)
    captures22 = mean(captures);
    for n= 1:261900
        captures2(n)=captures22(n);
    end
    clear captures22;
end

%clear some memory
clear captures;

%% Create a time matrix based on the sample window
time = [1:length(captures2)];
time = (time.* fftSize)./sampleRate;

%% Plot the Short-term Energy Diagram
figure;
plot (time,captures2);
title('Short-term Energy Diagram');

%% Calculate the power range and the thresholds
maxpower = max(max(captures2));
minpower = min(min(captures2));
powerrange = abs(maxpower - minpower);
cmin = minpower + powerrange*0.5;
cmax = maxpower;

%setting the axis scales and labels
axis ( [ 0, ((numtraces*fftSize)/sampleRate), minpower
    , maxpower]);
set(get(gca,'XLabel'),'String','Time (s)');
set(get(gca,'YLabel'),'String','Power (dBm)');

```

```
%setting the power ticks
ybounds = ylim;
set(gca, 'YTick', ybounds(1):ybounds(2) );

%calculating the threshold based on the power range
threshold1 = abs((maxpower-minpower)/4);
threshold2 = abs((maxpower-minpower)/4)*2;
threshold3 = abs((maxpower-minpower)/4)*3;

%% Plot the threshold lines on the Energy Diagram
xlim = get(gca,'xlim'); %Get x range
hold on;
plot([xlim(1) xlim(2)],[(minpower+threshold3) (
    minpower+threshold3)],'r');
plot([xlim(1) xlim(2)],[(minpower+threshold2) (
    minpower+threshold2)],'m');
plot([xlim(1) xlim(2)],[(minpower+threshold1) (
    minpower+threshold1)],'y');
hold off;

%% Make a new matrix to analyze the distribution of
    the power
captures3 = [];

%three different levels corresponding to the
    thresholds
for n=1:length(captures2)
    if ( captures2(n)>(minpower+threshold1) &&
        captures2(n)<(minpower+threshold2) )
         captures3(n) = 1;
    elseif ( captures2(n)>(minpower+threshold2) &&
        captures2(n)<(minpower+threshold3) )
         captures3(n) = 2;
    elseif ( captures2(n)>(minpower+threshold3) )
         captures3(n) = 3;
    else
         captures3(n) = 0;
    end
end

clear captures2;    %clear some memory

time = time.*10^6;   %time convertion in us

```

```
%% New Energy dinstribution diagram in time
figure(2);
plot(time,captures3);

%setting axis scales and labels
axis ( [ 0, (((numtraces*fftSize)/sampleRate)*10^6),
    -0.5, 3.5]);
set(get(gca,'XLabel'),'String','Time␣(us)');
set(get(gca,'YLabel'),'String','ENERGY');

%% Make a new matrix that considers only the higher
    thresholds
for n=1 : length(captures3)
    if ( captures3(n) > 1 )
        captures3(n) = 1;
    else
        captures3(n) = 0;
    end
end

%% Correction of 2.56*2= 5.12 s corresponding to 2
    samples

for n=1 : (length(captures3)-3)
    if ( (captures3(n) == 1) && (captures3(n+1) == 0)
       && (captures3(n+2) == 1) )
        captures3(n+1) = 1;
    elseif ( (captures3(n) == 1) && (captures3(n+1) ==
        0) && (captures3(n+2) == 0) && (captures3(n+3)
        == 1) )
        captures3(n+1) = 1;
        captures3(n+2) = 1;
    end
end

%% Inverse correction 2.56*2= 5.12 s corresponding to
    2 samples

for n=1 : (length(captures3)-3)
    if ( (captures3(n) == 0) && (captures3(n+1) == 1)
       && (captures3(n+2) == 0) )
        captures3(n+1) = 0;
    elseif ( (captures3(n) == 0) && (captures3(n+1) ==
        1) && (captures3(n+2) == 1) && (captures3(n+3)
```

```
            == 0) )
            captures3(n+1) = 0;
            captures3(n+2) = 0;
    end
end

%% Packet diagram Plot

figure(3);
plot(time,captures3);

%New cursor mode to obtain a better resolution in time
dcmObj = datacursormode;  %# Turn on data cursors and
    return the
                            %#    data cursor mode object
set(dcmObj,'updateFcn',@updateFcn);  %# Set the data
    cursor mode object update

%setting axis scales and labels
axis ( [ 0, (((numtraces*fftSize)/25e6)*10^6), -0.2,
    1.2]);
set(get(gca,'XLabel'),'String','Time␣(us)');
set(get(gca,'YLabel'),'String','Packet␣Diagram');


%% WIFI automatic recognition
%every samples of 2,56us with fftSize=64
%Null space considered of 6 samples ----> 15,36 us
    maximum
%however we can consider the first and the last
    samples as a single contribution
%average ----> 12,8 us
%an accepted InterFrame must be a size of 5 or 6
    samples.

i = 0;        %counter variable
five_IF=0;  %InterFrame of 5 samples
six_IF=0;   %InterFrame of 6 samples
z = 0;        %WIFIpackets counter
wifi_packets = []; %WIFIpackets matrix
for n=1 : length(captures3)
    if ( captures3(n)==1 )
        i=i+1;
```

```
%90 samples is the threshold to be ensure that before the IF we had a packet. About 230 us
    elseif ( captures3(n)==0 && i<=90 )
        i=0;
    elseif ( captures3(n)==0 && i>90 )
%Dicrimination between the 5IFs and the 6IFs
%after every right InterFrame we save the duration of the packet before in a matrix
        if ((captures3(n+1)==0)&&(captures3(n+2)==0)&&(captures3(n+3)==0)&&(captures3(n+4)==0)&&(captures3(n+5)==0)&&(captures3(n+6)==1))
            six_IF=six_IF+1;
            z = z+1;
            wifi_packets(z) = i;
        elseif ((captures3(n+1)==0)&&(captures3(n+2)==0)&&(captures3(n+3)==0)&&(captures3(n+4)==0)&&(captures3(n+5)==1))
            five_IF=five_IF+1;
            z = z+1;
            wifi_packets(z) = i;
        end

        i=0;
    end
end

%Convertion samples to time (us)
wifi_packets = wifi_packets.*2.56;

%% SIFS recognition stamp
if ((five_IF + six_IF)> 0)
    fprintf ('WIFI recognition based on Short InterFrame Space\n');
    fprintf ('   %d IF of 5 samples and %d IF of 6 samples\n', five_IF, six_IF);
    % 5 samples ----> 2.56*4 = 10.24us
    % 6 samples ----> 2.56*5 = 12.80us
    %% calculating the average and the variance of the IFs
    average_IF = ((five_IF)*(10.24) + (six_IF)*(12.80)) / (five_IF + six_IF);
    var_IF = ((five_IF*(10.24-average_IF)^2) + (six_IF*(12.80-average_IF)^2)) / (five_IF + six_IF);
    fprintf ('␣␣␣IF␣average:␣%f␣us␣\n␣␣␣IF␣variance:␣%
       f␣us\n\n', average_IF, var_IF);
    fprintf ('%f␣s␣analyzed\n\n', len/sampleRate)

    %% BoxPlot of the WIFI packet lengths
    figure(4);
    boxplot(wifi_packets);
    %setting axis scales and labels
    set(get(gca,'XLabel'),'String','Packet␣recognized'
       );
    set(get(gca,'YLabel'),'String','WIFI␣packets␣
       length␣[us]');

    %% Mean WIFI packets length
    disp(['␣␣␣Average␣WIFI␣data␣packets:␣' num2str(
       mean(wifi_packets)) '␣us']);
    %disp(['   Variance WIFI data packets: ' num2str(
       var(wifi_packets)) ' us']);
end


%% Bluetooth packets recognition

%% another correction of 3/4 samples.
%Possible because the length of the Bluetooth packets
    and acknowledgements is longer
for n=1 : (length(captures3)-5)
    if ( (captures3(n)==1)&&(captures3(n+1)==0)&&(
       captures3(n+2)==0)&&(captures3(n+3)==0)&&(
       captures3(n+4)==1) )
        captures3(n+1) = 1;
        captures3(n+2) = 1;
        captures3(n+3) = 1;
    elseif ((captures3(n)==1)&&(captures3(n+1)==0)&&(
       captures3(n+2)==0)&&(captures3(n+3)==0)&&(
       captures3(n+4)==0)&&(captures3(n+5)==1))
        captures3(n+1) = 1;
        captures3(n+2) = 1;
        captures3(n+3) = 1;
        captures3(n+4) = 1;
    end
end

%inverse correction of 3/4 samples.
```

```
for n=1 : (length(captures3)-5)
    if ( (captures3(n)==0)&&(captures3(n+1)==1)&&(
        captures3(n+2)==1)&&(captures3(n+3)==1)&&(
        captures3(n+4)==0) )
         captures3(n+1) = 0;
         captures3(n+2) = 0;
         captures3(n+3) = 0;
    elseif ((captures3(n)==0)&&(captures3(n+1)==1)&&(
        captures3(n+2)==1)&&(captures3(n+3)==1)&&(
        captures3(n+4)==1)&&(captures3(n+5)==0))
         captures3(n+1) = 0;
         captures3(n+2) = 0;
         captures3(n+3) = 0;
         captures3(n+4) = 0;
    end
end

%% Inter arrival Checking
% (not active)
% i = 0;
% inter = 0;
% for n=1 : length(captures3)
%     if ( captures3(n) == 0 )
%         i=i+1;
%     elseif ( captures3(n)==1 && i>242 && i<247 )
%         i=0;
%         inter = inter + 1;
%     else
%         i=0;
%     end
% end

%% NULL, oneslot, threeslot and fiveslot recognize (@
   1 MS/s)

i = 0;
NULL = 0;
oneslot = 0;
threeslot = 0;
fiveslot = 0;
one_matrix = [];
three_matrix = [];
five_matrix = [];
for n=1 : length(captures3)
```

```
    if ( captures3(n)==1 )
        i=i+1;
    %%NULL 126us ----> 49_50_51 samples
    elseif ( captures3(n)==0 && i<52 && i>48 )
        i=0;
        NULL = NULL + 1;
    %%oneslot 126us to 366us -----> 143_144_145 samples
    elseif ( captures3(n)==0 && i<146 && i>48 )
        oneslot = oneslot + 1;
        one_matrix(oneslot) = i;
        i=0;
    %%threeslot 1250 to 1622us
    elseif ( captures3(n)==0 && i<636 && i>486 )
        threeslot = threeslot + 1;
        three_matrix(threeslot) = i;
        i=0;
    %%fiveslot 2500 to 2870us
    elseif ( captures3(n)==0 && i<1124 && i>974 )
        fiveslot = fiveslot + 1;
        five_matrix(fiveslot) = i;
        i=0;
    else
        i=0;
    end
end

%conversion samples to time
one_matrix = one_matrix.*2.56;
three_matrix = three_matrix.*2.56;
five_matrix = five_matrix.*2.56;

%% NULL recognition stamp
if (NULL > 0)
    fprintf('\n\n');
    fprintf('Bluetooth␣recognition␣based␣on␣NULL␣packets␣(1␣Mb/s)\n');
    fprintf('␣␣␣%d␣NULL␣packets\n␣␣␣%d␣oneslot␣packets\n', NULL, oneslot);
    fprintf('␣␣␣%d␣threeslot␣packets\n␣␣␣%d␣fiveslot␣packets\n\n', threeslot, fiveslot);
    disp(['␣␣␣Average␣oneslot␣packets:␣' num2str(mean(one_matrix)) '␣us']);
    disp(['   Average threeslot packets: ' num2str(mean(three_matrix)) ' us']);
    disp(['   Average fiveslot packets: ' num2str(mean(five_matrix)) ' us']);
    fprintf('\n');
    disp([ num2str(len/sampleRate) ' s analyzed']);
    fprintf('\n');
end

%% BoxPlot of oneslot range packets to analyze the length distribution
%and to underline any difference from the WIFI BoxPlot
figure(5);
boxplot(one_matrix);
set(get(gca,'XLabel'),'String','One slot range [126 to 366 us]');
set(get(gca,'YLabel'),'String','Packet duration (us)');

%% Check of File Transmission packet duration
blueACK=0;
blueONE=0;
blueTHREE=0;
blueFIVE=0;
blue1=0;
blue2=0;
blue3=0;
for n=1 : length(captures3)
    if ( captures3(n)==1 )
        i=i+1;
    %%ACK type 48.6us to 60 us about ----> 19-24 samples
    % 2.1 to 3 MS/s
    elseif ( captures3(n)==0 && i<26 && i>17 )
        i=0;
        blueACK = blueACK + 1;
    %%oneslot type 174us ----> 68 samples
    % 2.1 MS/s
    elseif ( captures3(n)==0 && i<71 && i>65 )
        i=0;
        blueONE = blueONE + 1;
    %%threeslot type 770.56/773.12us ------> 301-302 samples
    % 2.1 MS/s
```

```
        elseif ( captures3(n)==0 && i>298 && i<305 )
            blueTHREE = blueTHREE + 1;
            i=0;
        %%fiveslot type 1366/1367us ------> 533-534
            samples
        % 2.1 MS/s
        elseif ( captures3(n)==0 && i>530 && i<537 )
            blueFIVE = blueFIVE + 1;
            i=0;
        %%first type 307.2us / 309.8us ----> 119
            _120_121_122 samples
        elseif ( captures3(n)==0 && i<123 && i>118 )
            i=0;
            blue1 = blue1 + 1;
        %%second type 811.5us -----> 317 samples
        elseif ( captures3(n)==0 && i>315 && i<319 )
            blue2 = blue2 + 1;
            i=0;
        %%third type 924.16us ------> 361 samples
        elseif ( captures3(n)==0 && i>359 && i<363 )
            blue3 = blue3 + 1;
            i=0;
        else
            i=0;
        end
end

%% File packets stamp
if ( blueACK > 0 )
        fprintf('\n');
        fprintf('Bluetooth␣recognition␣based␣on␣ACK␣
            packets␣(2.1␣to␣3␣Mb/s)\n');
        disp(['␣␣␣Bluetooth␣File␣Acknowlegments␣(2.1␣to␣3␣
            Mb/s):␣' num2str(blueACK) ]);
        disp(['␣␣␣Bluetooth␣OneSlot␣packets␣(2.1␣Mb/s):␣'
            num2str(blueONE)]);
        disp(['␣␣␣Bluetooth␣ThreeSlot␣packets␣(2.1␣Mb/s):␣
            ' num2str(blueTHREE)]);
        disp(['␣␣␣Bluetooth␣FiveSlot␣packets␣(2.1␣Mb/s):␣'
            num2str(blueFIVE)]);
        disp(['␣␣␣Bluetooth␣Other␣packets␣(in␣total):␣'
            num2str(blue1+blue2+blue3)]);
        fprintf('\n');
        disp([ num2str(len/sampleRate) '␣s␣analyzed']);
```

```
    fprintf('\n\n');
end




%% Report stats
timetaken = toc;
str = ['Processing took ' num2str(timetaken) ' seconds
    for file: ' filename];
disp(str)
```

ABBREVIATIONS AND ACRONYMS

ISM	Industrial Scientific and Medical
USRP	Universal Software Radio Peripheral
WPAN	Wireless Personal Area Network
FHSS	Frequency Hopping Spread Spectrum
GFSK	Gaussian Frequency Shift Keying
EDR	Enhanced Data Rate
PSK	Phase Shift Keying
AP	Access Point
DSSS	Direct Sequence Spread Spectrum
DBPSK	Differential Binary Phase Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
CCK	Code Complementary Keying
IFS	InterFrame Spaces
SIFS	Short InterFrame Space
ACK	Acknowledgment
ED	Energy Detector
SED	Short-Term Energy Diagram
FFT	Fast Fourier Transform
AWGN	Additive White Gaussian Noise
NP	Neymar-Person
FA	False Alarm probability
ROC	Receiver Operating Curve
ADC	Analog-to-Digital Converter
DFT	Discrete Fourier transform
SDR	Software Defined Radio
DDC	Digital Down Converter
USB	Universal Serial Bus
FPGA	Field Programmable Gate Array
DAC	Digital-to-Analog Converter
RF	Radio-Frequency

IF	Intermediate Frequency
GigE	Gigabit Ethernet
GRC	GNU Radio Companion
DSP	Digital Signal Processing
GUI	Graphical User Interface
ASIC	Application Specific Integrated Circuit