UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Diseño e implementación de un sincronizador OFDM implementado en USRP

TRABAJO DE TITULACIÓN

AUTORES: Herrera Bustamante, José Enrique

Rodríguez Ludeña, Vanessa Stefanía

DIRECTOR: Barragán Guerrero, Diego Orlando

LOJA- ECUADOR

2018

2018

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Magister.

Barragán Guerrero, Diego Orlando.

DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: Diseño e implementación de un sincronizador OFDM implementado en USRP, realizado por José Enrique Herrera Bustamante y Vanessa Stefanía Rodríguez Ludeña ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, enero de 2018

f) …….

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

Nosotros, José Enrique Herrera Bustamante con y Vanessa Stefanía Rodríguez Ludeña, declaramos ser autores del presente trabajo de titulación: Diseño e implementación de un sincronizador OFDM implementado en USRP, de la Titulación de Electrónica y Telecomunicaciones, siendo Diego Orlando Barragán Guerrero director del presente trabajo; y eximimos expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además, certificamos que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de nuestra exclusiva responsabilidad.

Adicionalmente declaramos conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: “Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad”

f. …….

Autor: José enrique Herrera Bustamante

Cédula: 1105165839

f. …….

Autor: Vanessa Stefanía Rodríguez Ludeña

Cédula: 1104445208

DEDICATORIA

Este trabajo lo dedico:

A mi madre quien me ha brindado su incondicional cariño con un increíble apoyo moral y anímico a lo largo de toda mi vida estudiantil y personal, ayudándome a salir adelante en los momentos más difíciles.

A mi padre que representa, para mí, un modelo a seguir de persona noble y respetable al haberme enseñado todos los valores y principios morales necesarios para ser un hombre de bien.

A mi hermana quien, a pesar de nuestras diferencias de pensamientos, siempre ha sido mi soporte y protección y ha sabido guiarme en la toma de difíciles decisiones que, en su momento, fueron muy importantes para mí.

A mis abuelos y demás familiares quienes me demostraron que la familia siempre es lo más importante anteponiendo su unión, ante todo; además, por haber puesto en mí una gran esperanza de triunfo y a quienes espero nunca defraudar.

A mis amigos y compañeros que, de una u otra forma, me ayudaron a crecer y superarme y con quienes espero seguir compartiendo momentos especiales en las próximas etapas de mi vida.

José Enrique

Dedico este trabajo principalmente a mis padres quienes a pesar de nuestra distancia física con sus consejos han sabido guiarme para culminar mi carrera profesional. A mis hermanas que siempre han estado junto a mí, brindándome su apoyo incondicional. A mi tío César por haberme acompañado durante mi trayecto estudiantil.

A mi familia en general, por compartir conmigo buenos y malos momentos. A Santiago, por acompañarme durante todo este arduo camino y compartir conmigo alegrías y fracasos. A mis profesores, gracias por su tiempo, por su apoyo, así como su sabiduría que me transmitieron en el desarrollo de mi formación profesional.

Vanessa Stefanía

AGRADECIMIENTOS

Por la realización de este trabajo, agradecemos:

A Dios por darnos fuerzas para superar obstáculos y dificultades a lo largo de toda nuestra vida.

Al tutor de nuestro trabajo de titulación, Mgtr. Diego Barragán, por la confianza puesta en nosotros, la paciencia durante el desarrollo del presente, y el apoyo brindado ante nuestras dudas y errores.

A la Universidad Técnica Particular de Loja y especialmente al Ing. Manuel Quiñones y al Mgtr. Francisco Sandoval, miembros de jurado, cuyas observaciones contribuyeron a mejorar la calidad de este trabajo; asimismo, por habernos facilitado el uso del espacio y los equipos del Laboratorio de Telecomunicaciones.

A todas las personas que ayudaron directa e indirectamente en la realización de este proyecto.

ÍNDICE DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En la actualidad, OFDM es el esquema de modulación más utilizado en tecnologías inalámbricas como Wi-Fi, LTE y WiMAX, debido a su capacidad de transmitir altas tasas de datos. Sin embargo, la elevada sensibilidad frente a los errores de sincronismo es su principal desventaja. Ante esto, se desarrolló algoritmos de sincronización en base al preámbulo del estándar IEEE802.11a, cuya simulación en Python/GNURadio e implementación con el USRP son detalladas en el presente trabajo.

El sincronismo fue dividido en tres etapas: detección del paquete, resuelta mediante la comparación de la autocorrelación y la potencia media de la señal, resultando en un pulso activo cuando el preámbulo es detectado; sincronismo de tiempo, donde se implementó la correlación cruzada con las secuencias STS y LTS, determinándose un mejor rendimiento con STS en base al análisis de la varianza de sus resultados; y sincronismo de frecuencia, corrigiéndose el CFO añadido por el canal, mediante la detección de fase de la señal y su posterior compensación. El desempeño de los algoritmos fue verificado con los resultados de la implementación con los USRPs.

PALABRAS CLAVES: OFDM, sincronismo, preámbulo, IEEE802.11a, detección de paquete, sincronismo de tiempo, CFO, Python, GNURadio, USRP.

ABSTRACT

Currently, OFDM is the most used modulation scheme in wireless technologies such as Wi-Fi, LTE y WiMAX, due to its capacity to transmit high rates of data. However, the high sensitive facing the synchronism errors is its principal drawback. Therefore, some synchronization algorithms were developed based on the preamble of IEEE802.11a standard, whose simulation in Python/GNURadio and USRP implementation are detailed in this document.

Synchronism was divided into three stages: packet detection, resolved by comparing the autocorrelation and the average power of the signal, resulting in an active pulse when the preamble is detected; time synchronism, where the cross correlation was implemented with the STS and LTS sequences, determining a better performance with LTS based on the analysis of the result variance; and the frequency synchronism, correcting the CFO added by the channel, by detecting the phase of the signal and its posterior compensation. The performance of the algorithms was verified with the results of the implementations with USRPs.

KEYWORDS: OFDM, synchronism, preamble, IEEE802.11a, packet detection, time synchronism, frequency synchronism, CFO, Python, GNURadio, USRP.

TERMINOLOGÍA

ADC: Analog to Digital Converter

AGC: Automatic Gain Control

AWGN: Additive White Gaussian Noise

BPSK: Binary Phase Shift Keying

CFO: Carrier Frequency Offset

CP: Cyclic Prefix

DAB: Digital Audio Broadcast

DAC: Digital to Analog Converter

DVB-T: Digital Video Broadcast Terrestrial

ETSI: European Telecommunications Standards Institute

FFT: Fast Fourier Transform

GI: Guard Interval

ICI: Inter Carrier Interference

IDFT: Inverse Discrete Fourier Transform

IEEE: Institute of Electrical and Electronics Engineers

IF: Intermediate Frequency

IFFT: Inverse Fast Fourier Transform

IP: Internet Protocol

ISI: Inter Symbol Interference

LTE: Long Term Evolution

LTS: Long Training Sequence

MIMO: Multiple Inputs Multiple Outputs

MCM: Multi-Carrier Modulation

NLOS: Non Line Of Sight

OFDM: Orthogonal Frequency Division Multiplexing

OTT: Out Of Tree

PDP: Power Delay Spread

PSK: Phase Shift Keying

QAM: Quadrature Amplitude Modulation

QPSK: Quadrature Phase Shift Keying

QA: Quality Assurance

RF: Radio Frequency

Rms: Root Mean Square

SCO: Sampling Clock Offset

SDR: Software Defined Radio

SNR: Signal to Noise Ratio

SRAM: Static Random Access Memory

STS: Short Training Sequence

SWIG: Simplified Wrapper and Interface Generator

UHD: USRP Hardware Driver

USRP: Universal Software Radio Peripheral

WiMAX: Worldwide Interoperability for Microwave Access

WMAN: Wireless Metropolitan Area Network

WLAN: Wireless Local Area Network

xDSL: X-Digital Subscriber Line

INTRODUCCIÓN

La técnica de multiplexación por división de frecuencias ortogonales, u OFDM, por sus siglas en inglés, es uno de los esquemas de modulación más utilizados en la actualidad, lo podemos encontrar en redes locales con tecnología Wi-Fi, en los hogares con TV Digital de los estándares *Digital Audio Broadcast* (DAB) y *Digital Video Broadcast – Terrestrial* (DVB-T), en telefonía móvil con el estándar *Long Term Evolution* (LTE) y en ambientes metropolitanos con tecnología *Worldwide Interoperability for Microwave Access* (WiMAX). Esta técnica se caracteriza por dividir el ancho de banda del canal en varias subportadoras ortogonales entre sí, siendo robusta tanto en canales con *Additive White Gaussian Noise* (AWGN) como en canales de múltiples trayectorias o Rayleigh. Una de las ventajas que presenta este esquema es la baja interferencia intersimbólica, un uso eficiente del espectro y una ecualización más sencilla, esto, en comparación a los sistemas monoportadora; por el contrario, su principal desventaja es la alta sensibilidad a errores de sincronismo que el receptor a través de diversas tareas debe corregir.

El sincronismo, como paso clave para la correcta recepción de los paquetes, está dividido en tres etapas: la primera es la detección del paquete que se resuelve con la comparación entre la autocorrelación de la señal y la potencia media de la misma. La segunda etapa es el sincronismo temporal que consiste en determinar el momento exacto de inicio de la información relevante para minimizar los efectos de *Inter Symbol Interference* (ISI). Hay varias formas de resolver esta etapa, siendo una de ellas la correlación cruzada de la señal recibida con los símbolos de las secuencias STS y/o LTS (*Short* y *Long Training Sequence*, respectivamente). Finalmente, la tercera etapa es el sincronismo de frecuencia, donde el alineamiento de la frecuencia entre portadoras del receptor y del trasmisor, evitando la *Inter Carrier Interference* (ICI), se consigue mediante la detección de la fase de la señal recibida haciendo uso de la correlación de la señal de entrada con la misma retrasada 16 muestras. Su resultado, en unidades angulares por tratarse de una fase angular, pasa a unidades de kHz, con lo cual se realiza la corrección del *Carrier Frequency Offset* (CFO) introducido por el canal inalámbrico.

En este trabajo se realiza el diseño, simulación e implementación de algoritmos que resuelven las tres etapas de sincronismo antes mencionadas: detección del paquete, sincronismo de tiempo y sincronismo de frecuencia. Las simulaciones se realizaron en el lenguaje de programación Python, con el objetivo de usar estos códigos para el diseño y creación de bloques propios en GNURadio, necesarios para la implementación de los algoritmos con la plataforma USRP N210 y demás equipos facilitados por el laboratorio de Telecomunicaciones

de la UTPL. Finalmente se compara el desempeño de cada algoritmo frente a canales AWGN reales y simulados, obteniéndose la varianza de los datos resultantes en la implementación.

La importancia de este trabajo es tener un mayor conocimiento sobre la sincronización de paquetes evitando así la pérdida de estos en todos los sistemas de comunicaciones inalámbricas que usan OFDM, especialmente los de la tecnología Wi-Fi con el estándar IEEE802.11 que se usa diariamente y cada vez con mayor frecuencia. Implementar los algoritmos de sincronización en un sistema SDR con los equipos USRP N210, implica un aporte significativo a la investigación de sistemas de radio en software dado que, en la actualidad, éstos se encuentran en auge por su bajo costo y eficiencia a la hora de simular sistemas de comunicaciones inalámbricas.

En resumen, el propósito de la implementación de los algoritmos de sincronismo es determinar su desempeño frente a diferentes escenarios de ruido y a los desplazamientos tanto en tiempo como en frecuencia, los cuales son añadidos por el canal en el trayecto de la señal entre el transmisor y el receptor.

El trabajo está organizado en cuatro capítulos divididos según la metodología usada y cumpliendo los objetivos planteados en el proyecto. En el capítulo uno, se da una corta introducción al modelo matemático del sistema OFDM, introduce los principales parámetros del estándar IEEE802.11a que se utilizó en los capítulos posteriores, los efectos que agregan los canales a los sistemas de comunicación inalámbricos, los errores de sincronismo y, por último, se resume todo el estudio del estado del arte que se realizó acerca de los algoritmos capaces de resolver dichos errores tanto en tiempo y frecuencia. En el capítulo dos se detallan las simulaciones de estos algoritmos con el lenguaje de programación Python y se describe el diseño y la creación de los bloques en GNURadio necesarios para un esquema de modulación OFDM basado en paquetes para el estándar IEEE802.11a. En el tercer capítulo se procede a la implementación de los algoritmos con la plataforma USRP N210 en diversos ambientes, se ilustran los flujogramas de bloques en GNURadio para la transmisión y recepción del preámbulo mediante los USRP y se aprecia las alteraciones originadas por el canal. En el capítulo cuatro se analizan los resultados, se valora el desempeño de todos los algoritmos implementados según su valor de varianza. Finalmente, se muestran las conclusiones y recomendaciones para futuras investigaciones.

CAPÍTULO I:

MARCO TEÓRICO

En esta sección se aborda los principios básicos de los sistemas OFDM, el modelo matemático, el concepto de ortogonalidad y portadoras múltiples, como se forma el prefijo cíclico y las ventajas que ofrece a está modulación. Se presenta también el estándar IEEE802.11a, sus principales características, parámetros y la formación del preámbulo. De la misma forma se explican los problemas de sincronismo que presenta OFDM y sus posibles soluciones con el diseño de algoritmos para la detección del paquete, el sincronismo de tiempo y el sincronismo de frecuencia haciendo uso de técnicas simples como son auto correlación y correlación cruzada.

1.1. OFDM

Las técnicas de modulación digitales pueden clasificarse ampliamente en dos categorías: modulación monoportadora, donde los datos se transmiten usando una portadora de radiofrecuencia que origina que los símbolos se solapen en el receptor cuando se produce una propagación multitrayecto, causando lo que se conoce como interferencia intersimbólica (ISI) [1]. En esta categoría el receptor no es capaz de demodular dicha señal sin el uso de un ecualizador, el cual introduce una alta complejidad en el sistema. La otra categoría es la modulación multiportadora (*Multi-Carrier* Modulation - MCM) donde los datos se transmiten modulando simultáneamente múltiples portadoras de radiofrecuencia, ocasionando la interferencia interportadora (ICI) [2].

La técnica de Multiplexación por División de Frecuencia Ortogonal (OFDM) es un caso especial de los esquemas de MCM, siendo ampliamente adoptada en muchos sistemas de comunicación de banda ancha y de alta velocidad de datos tales como DAB, DVB-T, comunicación de línea telefónica de alta velocidad como xDSL (*X-Digital Subscriber Line*), WLAN - IEEE 802.11 (*Wireless Area Network*) y WMAN - IEEE802.16 (*Wireless Metropolitan Area Network*). Mediante el uso de OFDM, estos sistemas de comunicación inalámbrica se esfuerzan por lograr una buena eficiencia espectral, altas velocidades de datos, comunicación robusta y complejidad computacional relativamente más baja [3].

1.1.1. Ortogonalidad y portadoras múltiples.

OFDM es un método de multiplexación donde diferentes canales denominados subportadoras utilizan una fracción del ancho de banda disponible para transportar los datos que están modulados en amplitud y fase. OFDM se caracteriza porque todas sus subportadoras son ortogonales entre sí, con lo que se logra un buen rendimiento espectral [4].

Los símbolos de datos se modulan en un número específico de subportadoras con una separación de frecuencia mínima requerida para mantener la ortogonalidad en el dominio del

tiempo, aunque los espectros de señal de las diferentes subportadoras se superponen en el dominio de la frecuencia, como se observa en la Figura 1.1.

La señal OFDM en banda base a la salida del transmisor es la concatenación de todos los símbolos OFDM dados por la secuencia de salida *Inverse Discrete Fourier Transform* (IDFT) de *N* puntos. La señal en banda base discreta en el tiempo del m-ésimo símbolo OFDM está dada por (1.1):

$$x(n) = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} X(m) e^{\frac{j2\pi nm}{N}}, \quad 0 \leq n \leq N-1 \tag{1.1}$$

Donde n es un índice de muestra en el dominio del tiempo, $X(m)$ son los símbolos transmitidos y modulados en la m-ésima subportadora y N es el número de subportadoras. Se utiliza el factor $1/\sqrt{N}$ para que sean mantenidas las condiciones del Teorema de Parseval, donde la energía de la señal en el dominio del tiempo es igual a la energía en el dominio de la frecuencia [5], [6].

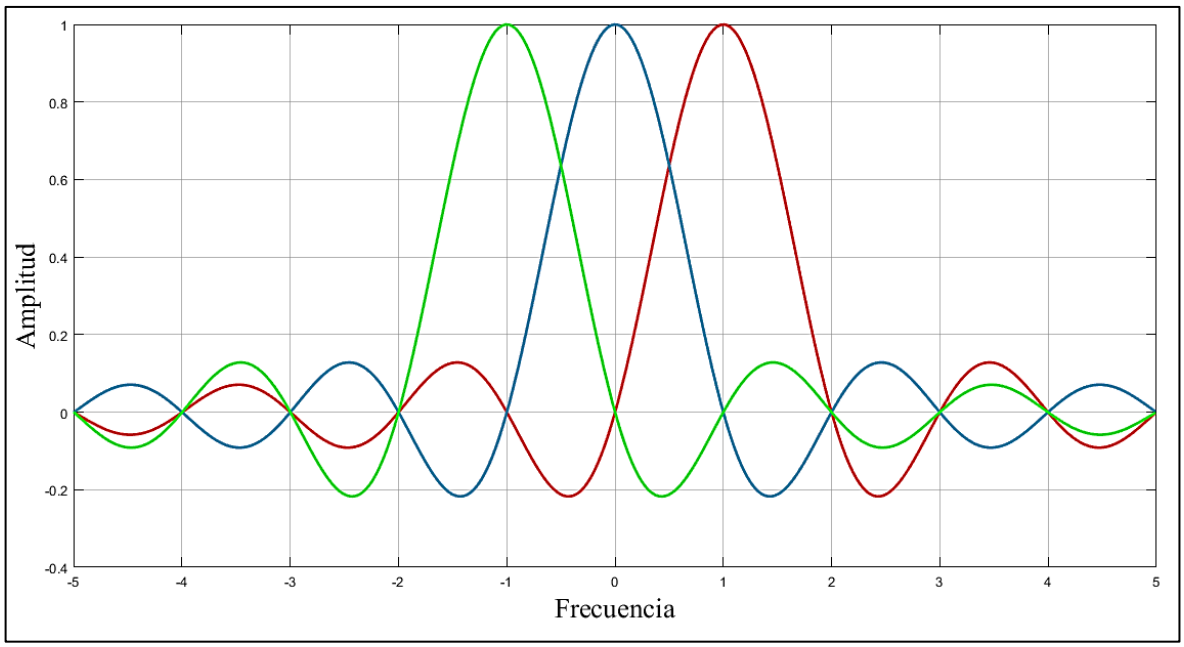
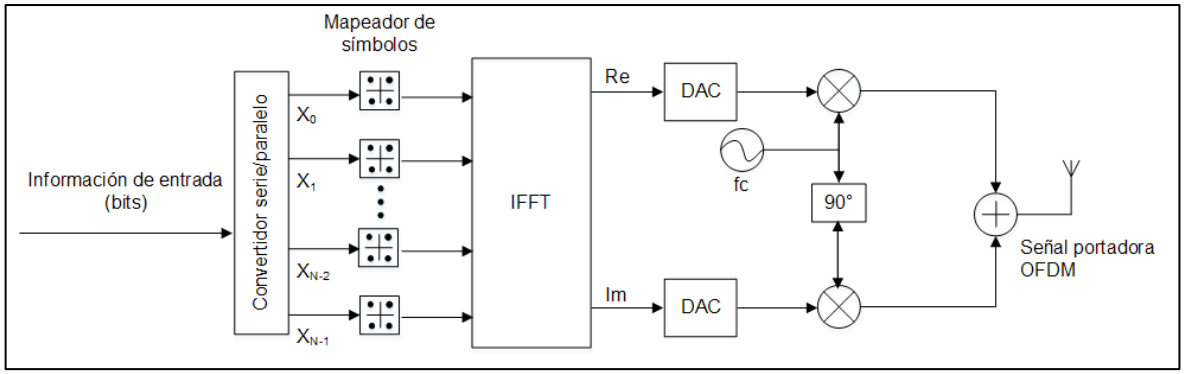


Figura 1.1. Ortogonalidad y portadoras múltiples de OFDM
Fuente: [5]
Elaboración: Autores

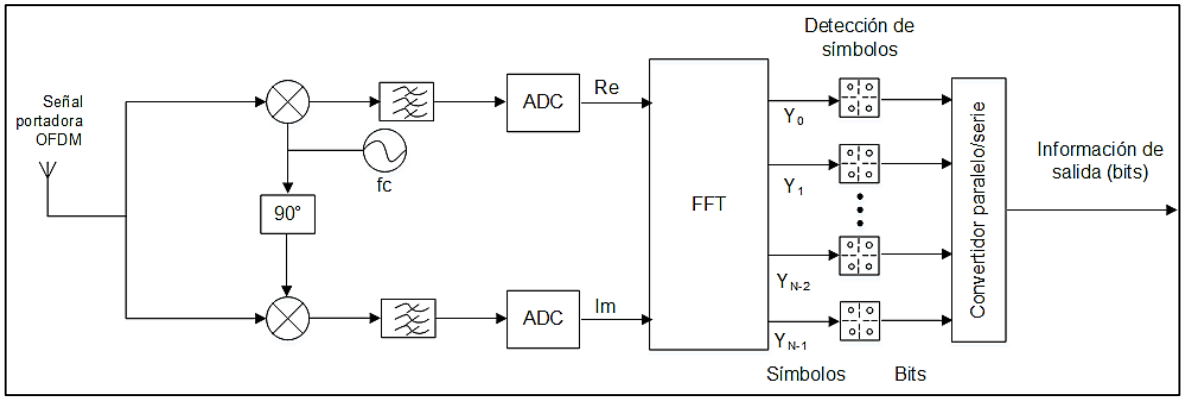
El diagrama básico de un sistema OFDM se muestra en la Figura 1.2, donde los datos en bits se dividen entre las subportadoras utilizando un convertidor de serie a paralelo y para cada subportadora se modula independientemente utilizando, en la mayoría de los casos, una modulación de amplitud de cuadratura (*Quadrature Amplitude Modulation* - QAM) o una modulación por desplazamiento de fase (*Phase Shift* Keying - PSK). A continuación, se crea la portadora OFDM con todas las subportadoras moduladas utilizando una transformada

inversa de Fourier (*Inverse Fast Fourier Transform* - IFFT) que calcula la señal de dominio de tiempo con todas las subportadoras para crear una única señal compleja de banda ancha que contiene todos los datos pertenecientes a todas las subportadoras.

El receptor separa la parte real de la imaginaria de las señales recibidas. Las señales pasan por el convertidor analógico - digital (*Analog to Digital Converter* - ADC) y, a continuación, las diferencias de frecuencia de las variables se calculan con un bloque de transformada de Fourier (*Fast Fourier Transform* - FFT). Este último producirá los diferentes flujos correspondientes a las subportadoras utilizados y la información en cada uno de ellos se demodula obteniendo así la señal transmitida [3].



(a)



(b)

Figura 1.2. Diagrama general de (a) transmisor y (b) receptor OFDM
Fuente: [3]
Elaboración: Autores

Una de las mayores ventajas de OFDM frente a sistemas monoportadoras, es su capacidad de hacer frente a severas condiciones de canal tales como interferencia de banda estrecha y atenuación selectiva de frecuencia debido al multitrayecto [1]. Sin embargo, introduce nuevos problemas al usar números elevados de subportadoras estrechas [3], el sistema se vuelve muy sensible a las desviaciones de tiempo y frecuencia y, por lo tanto, se necesita una sincronización precisa.

1.1.2. Prefijo cíclico

En entornos urbanos donde elementos como edificios o autos existen en mayores cantidades que en los medios rurales, y en entornos donde el emisor y/o el receptor se mueven a altas velocidades, el receptor de los sistemas de comunicación inalámbrica siempre recibirá un señales con diferentes cantidades de retraso respecto a la primera señal, denominado retardo de propagación, originando dos problemas consigo ICI e ISI [1].

La solución que OFDM propone para reducir el efecto de la propagación de trayectos múltiples es la adición de redundancia en la señal transmitida, con el elemento conocido como prefijo cíclico (*Cyclic Prefix* - CP) o intervalo de guarda (*Guard Interval* - GI) al comienzo de cada paquete [2], [3].

El objetivo del prefijo cíclico es proporcionar un tiempo de guarda entre los símbolos que serán descartados por el receptor y contendrán los restos no deseados de los ecos del símbolo anterior. El tiempo de guarda se forma al agregar, al principio del símbolo, un prefijo que contiene los últimos bits del mismo símbolo que se está enviando [3] como se muestra en la Figura 1.3. El GI también podría estar compuesto de ceros u otra señal arbitraria, pero la duplicación desde el final del símbolo OFDM preserva la periodicidad y simplifica la sincronización de tiempo. Al agregar el CP se reduce la velocidad de datos pero el sistema es menos vulnerable a los canales de trayectos múltiples en la mayoría de los casos [1].

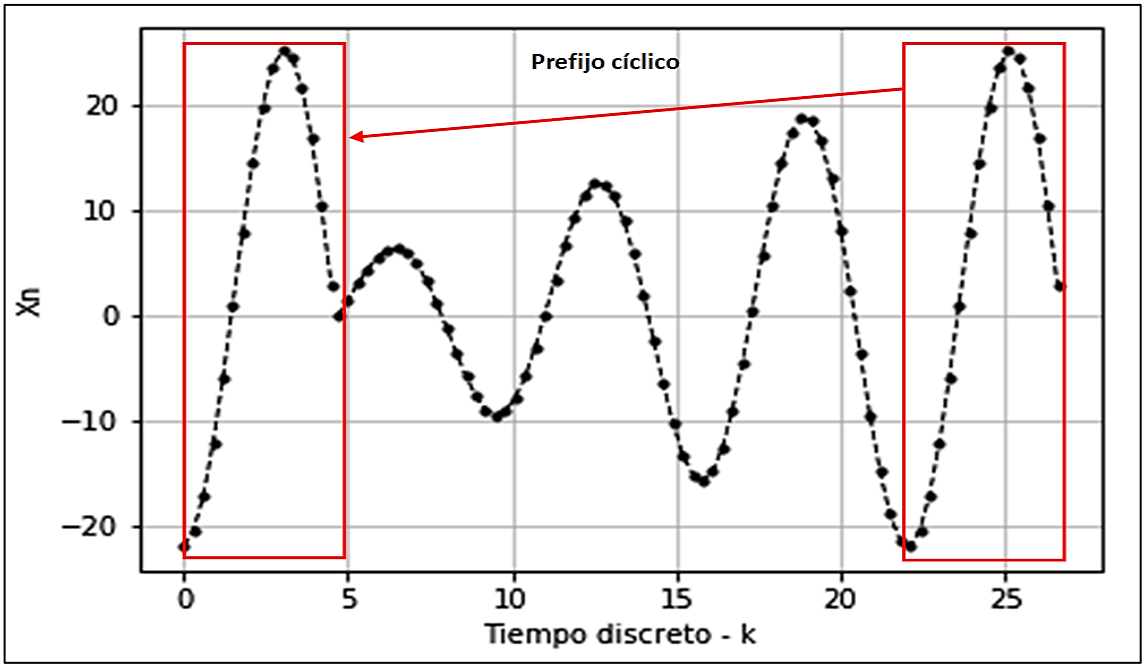


Figura 1.3. Adición de prefijo cíclico a una señal X_n
Fuente: [5]
Elaboración: Autores

1.2. Estándar IEEE802.11a.

IEEE802.11a es uno de los estándares de las redes inalámbricas de área local WLAN basado en IEEE802.11, publicado en 1997 por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) y conocido como Wi-Fi. El IEEE802.11a soporta velocidades de hasta 54 Mbps y usa OFDM como técnica de modulación a razón de su gran desempeño en canales altamente dispersivos, como escenarios *indoor*, donde este estándar es usado. Opera en la banda de 5.8 GHz donde existe menos interferencia por otras señales que en la banda de 2.4 GHz o estándar IEEE802.11b/g. Para facilitar la implementación de filtros y lograr una supresión de canal adyacente suficiente, solo se utilizan 52 subportadoras de los 64 puntos de la longitud de la ventana de la IFFT: 48 subportadoras llevan datos de usuario, 4 son tonos pilotos para el seguimiento de fase y los 12 restantes son tonos nulos. Cada símbolo tiene 80 muestras de largo, que da una duración de 4 μs con una frecuencia de muestreo de 20 MHz [1]. La Tabla 1.1 resume los parámetros más importantes del estándar [5], [7]

Tabla 1.1. Parámetros principales del estándar IEEE802.11a

Parámetro	Símbolo	Valor
Ancho de banda	W	20 MHz
Período de muestreo	T_s	50 *ns* (1 muestra)
Subportadoras en el canal / Longitud de FFT	N_{FFT}	64
Subportadora de datos	N_{datos}	48
Subportadora de pilotos	$N_{pilotos}$	4
Subportadoras en total	$N_{subportadoras}$	52
Frecuencia de espaciamiento entre subportadoras	$\Delta_F = 20\ MHz/64$	312.5 kHz
Duración del preámbulo	$T_{Preámbulo} = T_{FFT}/4$	16 μs
Periodo de FFT/IFFT	$T_{FFT} = 1/\Delta_F$	3.2 μs (64 muestras)
Duración de CP	T_{CP}	0.8 μs (16 muestras)
Duración de un símbolo OFDM	$T_{Simbolo}\ (T_{CP} + T_{FFT})$	4 μs (80 muestras)

Fuente: [5], [7]
Elaboración: Autores

1.2.1. Preámbulo.

El paquete OFDM está formado por el preámbulo o encabezado y los datos de carga útil. El preámbulo está conformado por dos conjuntos de símbolos: los cortos (STS) cuya longitud es de 16 muestras y los largos con longitud de 64 muestras (LTS) cada uno. Es decir, el preámbulo consiste en 10 símbolos cortos desde *t1* hasta *t10* con una duración total de 8 μs, seguidos de un intervalo de guarda compuesto por las 32 muestras finales del LTS y duración de 1.6 μs y dos símbolos largos que son *T1* y *T2* con 3.2 μs de duración, como se observa en

la Figura 1.4. La formación del preámbulo se especifica en [5], [8] y se detalla en el estándar IEEE802.11a [7].

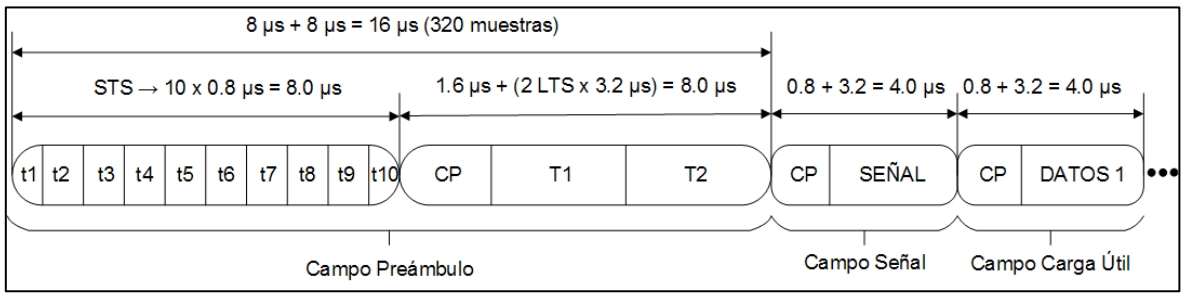


Figura 1.4. Estructura de secuencia de entrenamiento OFDM
Fuente: [5], [7], [8]
Elaboración: Autores

Un símbolo de entrenamiento corto (STS) consiste en 12 de las 52 subportadoras de OFDM, que son moduladas por los elementos de la secuencia S, dada por (1.2):

$$S_{-26,26} = \sqrt{\frac{13}{6}} * \{0,0,1+j,0,0,0,-1-j,0,0,0,1 + j,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1 + j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0,0,0,1 + j,0,0,0,1+j,0,0,0,1+j,0,0\} \quad (1.2)$$

La señal anterior es llevada al dominio del tiempo usando la IFFT, dando como resultado los STS transmitidos representados por (1.3):

$$r_{short}(t) = w_{Tshort}(t) \sum_{k=-N_{ST}/2}^{N_{ST/2}} S_k e^{kj2\pi\Delta f_t} \quad (1.3)$$

donde $T_{short} = 8\mu s$ es la duración total de STS, $N_{ST} = 52$ es el número de subportadoras y $\Delta f = 312.5\ kHz$ es el espaciamiento de frecuencia entre las subportadoras. El término $w(t)$ es una función de ventana de tiempo utilizada para suavizar la transición entre los símbolos OFDM y está dada por (1.4):

$$w_T(nT_S) = \begin{pmatrix} 1 & 1 \le n \le 79 \\ 0.5 & n = 0.80 \\ 0 & otros\ casos \end{pmatrix} \quad (1.4)$$

donde T_S es el período de muestreo (50 *ns* en 802.11a)

La señal en tiempo discreto se obtiene usando la IFFT de 64 puntos, generándose una señal de 64 muestras, pero como la señal en frecuencia es definida a través de 52 subportadoras es necesario completar esta señal para poder aplicar la IFFT. Es por ello que se agregan 11

ceros, 6 al principio y 5 al final de la Ecuación (1.2). Para generar STS, la señal a la cual se le aplica la IFFT debe ser repetida dos veces y media.

Un símbolo largo de entrenamiento consta de 53 subportadoras (incluido un valor cero en DC), que están moduladas por los elementos de la secuencia *L*, dada por (1.5):

$$L_{-26,26} = \begin{Bmatrix} 1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,1, \\ -1,1,-1,1,1,1,1,0,1,-1,-1,1,1,-1,1,-1,1,-1, \\ -1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,1,1 \end{Bmatrix} \quad (1.5)$$

Al aplicar la IFFT a (1.5) se genera un símbolo de entrenamiento OFDM largo de acuerdo con la Ecuación (1.6):

$$r_{LONG}(t) = w_{TLONG}(t) \sum_{k=-N_{ST}/2}^{N_{ST/2}} S_k e^{kj2\pi\Delta f(t-T_{GI2})} \quad (1.6)$$

donde $T_{LONG} = 8\mu s$ es la longitud total de LTS y $T_{GI2} = 1.6\mu s$ es la longitud del prefijo cíclico insertado antes del primer LTS. Del mismo modo los pasos descritos para STS también son realizados para LTS. Para generar un LTS completo se debe repetir el símbolo dos veces y añadir el prefijo cíclico.

El vector final del preámbulo se formaría con la Ecuación (1.7):

$$r_{PREAMBLE}(t) = r_{SHORT} + r_{LONG}(t - T_{SHORT}) \quad (1.7)$$

El preámbulo contiene información acerca del paquete que el receptor necesita, como su duración y su tasa de transmisión y también se usa para la sincronización. Los siete primeros símbolos STS se utilizan para el control automático de ganancia (*Automatic Gain Control* - AGC) y los tres últimos símbolos cortos para la estimación robusta del CFO y una sincronización de tiempo aproximada. Por su parte, el LTS se utiliza para la estimación leve del CFO y la evaluación del canal. Por último, para reducir la probabilidad de ISI, aumentar la robustez al desvanecimiento por trayectos múltiples y aumentar el rango de tolerancia para la sincronización de tiempo, se agrega un prefijo cíclico (CP) de 16 muestras a cada símbolo de datos en OFDM [5], [7].

1.3. Efectos de los canales en las comunicaciones inalámbricas

El rendimiento y la calidad de la comunicación inalámbrica dependen significativamente del tipo de canales que hay entre el transmisor y el receptor, por lo que es necesario un análisis de los comportamientos del canal, de sus efectos y los métodos para combatirlos. En la comunicación inalámbrica podría haber más de una ruta desde el transmisor al receptor, lo que se conoce como multitrayecto, producido por efectos como la difracción, reflexión,

dispersión y refracción de la señal en los objetos. El multitrayecto se clasifica en dos grupos: pequeña y gran escala. Un fenómeno a gran escala incluye efectos de pérdida de trayectoria, desvanecimiento por sombreado; mientras, un fenómeno a pequeña escala incluye desvanecimiento por múltiples trayectos [9].

- **Pérdida de trayectoria (*Path loss*):** representa el grado de atenuación de potencia de la señal a medida que aumenta la distancia entre el transmisor y el receptor.
- **Desvanecimiento por sombreado (*Shadowing*):** el desvanecimiento por sombra es una atenuación de la potencia media de la señal causada por los contornos del terreno (colinas altas o densas zonas urbanas) entre la antena del transmisor y la antena del receptor.
- **Multipath fading:** el desvanecimiento por multitrayecto se refiere al hecho de que la señal transmitida sigue muchos caminos diferentes antes de llegar al receptor. La calidad de la señal recibida depende de los siguientes parámetros característicos [10]:
    - **Delay spread o retraso de propagación***:* es el tiempo entre la primera señal recibida y la última. Se considera que la última ruta es -10 dB más baja que la ruta más fuerte.
    - **Power delay spread (PDP):** el perfil de retardo de potencia de un canal describe la distribución de tiempo de la potencia de la señal recibida cuando se transmite una onda de impulso a través del canal considerado.
    - **Retraso cuadrático medio (rms):** es una medida de la cantidad de dispersión de la señal en el tiempo y es un indicador importante para el canal inalámbrico. El rms puede variar de 20 a 50 ns para entornos de oficina pequeños, de 50 a 100 ns para edificios grandes y de 100 a 200 ns para entornos de fábrica [1].
    - **Doppler spread:** cuando el transmisor o el receptor están en movimiento, la señal recibida se dispersará en frecuencia como resultado del efecto Doppler. El desplazamiento de frecuencia está relacionado con la velocidad del movimiento y el ángulo entre la dirección de llegada de la señal y la dirección del movimiento***.***

Existe otro efecto que agrega el canal que es el ruido aditivo como ruidos electrónicos y térmicos que son inherentes a canales inalámbricos y a circuitos analógicos en RF. El ruido de parpadeo y el ruido de disparo también son fuentes importantes de ruido en los dispositivos electrónicos. Para simplificar, en un modelo de canal de banda base, todos los ruidos se combinan y modelan como un proceso aleatorio gaussiano complejo aditivo o AWGN.

El ruido AWGN escogido es tal que E_b/N_0 es 10 dB y es representada por (1.8)

$$y = x + sig * N \tag{1.8}$$

donde *x* es la señal de entrada, *y* es la señal de salida y *N* es el ruido de media cero y varianza unitaria. Dentro del factor sig, en la Ecuación (1.9), W es el ancho de banda de 312.5 kHz y R_d es la tasa de datos de 6 Mbps para el *Binary Phase Shift Keying* (BPSK) o 12 Mbps para el *Quadrature Phase Shift Keying* (QPSK) [11]:

$$sig = \sqrt{\frac{media(x^2) * W}{E_b/N_0 * R_d}} \tag{1.9}$$

El desplazamiento de frecuencia portadora (CFO) es introducido en la secuencia de entrada y se detallada en (1.10). El valor de cada desvío simula un caso con condiciones óptimas ($\Delta_f = 0\ kHz$), moderadas ($\Delta_f = 100\ kHz$) y severas ($\Delta_f = 200\ kHz$) [5].

$$r_n = z(t)e^{j2\pi\Delta f t}\big|_{t=nT_s} \tag{1.10}$$

Para modelar el desvanecimiento multitrayecto se utiliza el modelo *tapped-delay*, el cual usa los valores de retardo de propagación y atenuaciones tomados de las Tablas del Instituto Europeo de Normas de Telecomunicaciones (*European Telecommunications Standards Institute* - ETSI). Dentro de este estándar se destacan los modelos ETSI A (50 ns de retardo de propagación) y ETSI C (con 150 ns de retardo de propagación), debido al hecho de que son modelos consolidados en la literatura. El modelo A simula un ambiente interno de oficina típico sin línea de vista (*Non Line Of Sight* - NLOS) y 50 ns de rms. El modelo C corresponde a un ambiente de espacio abierto para condiciones NLOS y 150 de rms [5], [12], [13].

La señal transmitida o señal de entrada S_k y la señal de salida o señal recibida R_k pueden ser descritas tanto en el tiempo $R_k(t)$ como en frecuencia $R_k(f)$. La señal de entrada es afectada por el canal y por el ruido, como se muestra en (1.11) [11].

$$R_k = H_k S_k + W_k \tag{1.11}$$

Donde

$k = 0, \dots, N-1 \rightarrow N$ es el índice de subportadora

R_k: Muestras de preámbulo recibidas a la salida del bloque FFT.

S_k: Secuencia de entrenamiento del preámbulo conocida por el receptor.

H_k: Respuesta del canal en frecuencia.

W_k: Las muestras de ruido

La estimación del canal es esencial antes de la demodulación de las señales recibidas OFDM debido a la distorsión causada por el desvanecimiento selectivo de frecuencia y el

desvanecimiento variable en el tiempo. Las técnicas de estimación de canales [10] se pueden clasificar en dos categorías:

- **Técnicas de estimación de canal ciego:** estiman el estado del canal de información sin el conocimiento de la señal transmitida.
- **Técnicas de estimación de canales asistidas por datos:** la información conocida añadida a las señales transmitidas se usa para estimar la respuesta del canal. Pueden ser de dos tipos: canal asistido por símbolos piloto y la estimación del canal basado en símbolos de entrenamiento.

1.4. Sincronismo.

1.4.1. Problemas.

Los problemas de sincronismo resultan ser inevitables dentro de las comunicaciones digitales; comunicaciones en las cuales, la transmisión de información en banda base resulta imposible dado los recursos y limitaciones en equipos y entornos. Ante esto, las señales digitales discretas necesitan ser moduladas a señales continuas de altas frecuencias (RF), y posteriormente, demoduladas en el receptor para obtener su versión original [9], como se aprecia en la Figura 1.5.

De forma general, la modulación de una señal en banda base, consiste en la mezcla de dicha señal con otra generada por un oscilador local, dentro del transmisor, cuya frecuencia portadora va desde el orden de los kilohercios, y de esta manera la señal resultante puede pasar a ser transmitida al canal físico o inalámbrico. OFDM se caracteriza por emplear un oscilador de múltiples portadoras separadas lo suficiente para conseguir la ortogonalidad, donde las frecuencias de dichas portadoras son utilizadas para realizar el proceso de demodulación en el receptor.

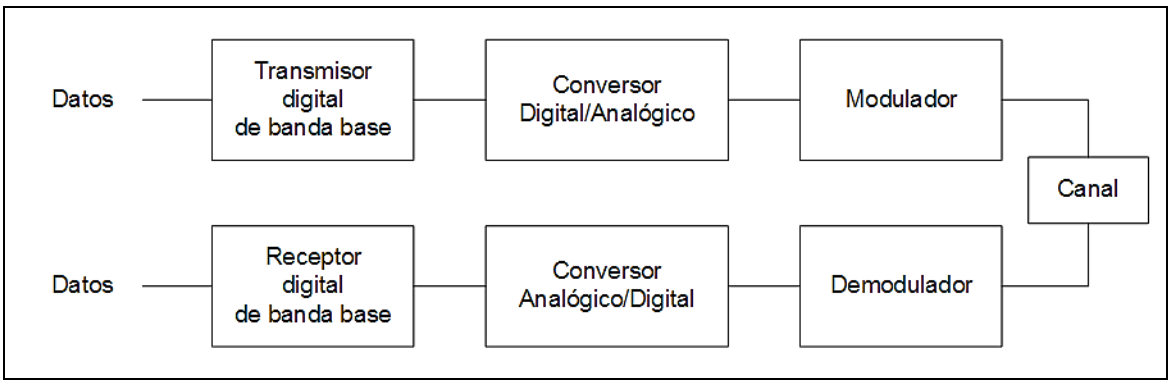


Figura 1.5. Procesamiento de una señal digital
Fuente: [9]
Elaboración: Autores

La demodulación, al igual que la modulación, emplea un oscilador local para la detección de la señal en banda base. El oscilador en el receptor necesita la misma frecuencia portadora del oscilador empleado en el transmisor permitiendo, así, detectar la señal de información de una forma efectiva, dándose lugar a la sincronización. Sin embargo, ambos osciladores no siempre se encuentran sincronizados provocándose errores en la recuperación de la señal original y es, precisamente eso, junto a la distorsión provocada por ruido, lo que provoca el surgimiento de los problemas de sincronismo en los sistemas OFDM.

Los problemas de sincronización son causados por los efectos del ruido del canal inalámbrico y aquellos adjudicados a canales multitrayecto. Ante esto, el receptor OFDM debe ser capaz de detectar el paquete entrante, mitigar cualquier desplazamiento de frecuencia de las subportadoras y lograr una sincronización precisa de tiempo respecto a la señal de reloj de muestreo [12]. Usualmente el receptor posee un oscilador controlado, del cual deriva las señales de oscilador local y de reloj, donde la falta de coincidencia del oscilador provoca un error de frecuencia/fase de portadora y un error de frecuencia/fase de reloj. Esto, debido a que el oscilador controlado no puede mantener una frecuencia/fase estable en su señal de salida y, además, sufre un ruido en fase de tiempo variable [9].

Las señales OFDM en banda base pueden verse alteradas por errores de sincronismo en términos de tiempo, frecuencia/fase y símbolos, como resultado de la presencia de interferencia, ruido y desvanecimiento introducidos por el canal. Tanto así, que pueden identificarse cuatro errores de sincronización [9]:

- ***Error de temporización del símbolo,*** $\boldsymbol{T_d}$. Modifica el límite del símbolo en el receptor en relación al límite real, en la forma de onda recibida.
- ***Desplazamiento de frecuencia de portadora (CFO),*** $\boldsymbol{\Delta f}$. Hace que la señal de banda base compleja recibida actúe a una frecuencia de $\boldsymbol{\Delta f}$.
- ***Error de fase de portadora,*** $\boldsymbol{\phi(t)}$**.** Provoca que una fase adicional sea introducida en la señal de banda base compleja recibida.
- ***Desplazamiento del reloj de muestreo (Sampling Clock Offset - SCO),*** $\boldsymbol{\delta}$**.** Produce un intervalo de muestreo de $(\mathbf{1}+\boldsymbol{\delta})\boldsymbol{T_s}$ en lugar de los $\boldsymbol{T_s}$ ideales para la señal de tiempo continuo recibida.

En sistemas que operan bajo el estándar 802.11a es muy corto el tiempo en el que la sincronización puede y debe ser realizada por el receptor. Al ser la información trasmitida en paquetes, estos pueden perderse en su totalidad lo que obligará al sistema a reenviar la información una y otra vez. Ante esto, una alta tasa de acierto es obligadamente requerida en los algoritmos de sincronismo usados en sistemas de transmisión de paquetes. Sin embargo, se busca un equilibrio respecto a su complejidad para evitar altos costos computacionales [5]

Una inapropiada sincronización de símbolo acarrea errores de bit afectando directamente la detección de la señal, degradando su desempeño, lo que se comprueba con el aparecimiento de las interferencias tanto entre símbolos como entre subportadoras [14].

1.4.2. Soluciones.

Para un sistema OFDM la sincronización en el receptor es un aspecto de gran importancia que exige ser llevado a cabo de manera óptima. Este tipo de sistemas resultan más susceptibles y vulnerables a los errores de sincronismo que aquellos sistemas de comunicación de portadora simple, dada la necesidad de ortogonalidad. La sincronización de una señal en este tipo de sistemas se traduce en encontrar el símbolo correcto en el instante preciso, y así también el desplazamiento de frecuencia que ha sido añadido a la señal recibida [15].

En sistemas basados en el estándar IEEE802.11a, la sincronización se facilita dada la estructura del paquete; al estar formado por un encabezado o preámbulo y a continuación por la información relevante que se está transmitiendo, la detección de éste último debe ser realizada inmediatamente después del preámbulo por parte del receptor OFDM. Esta detección depende totalmente de la eficiencia y rapidez del bloque de sincronismo, en el que los errores antes mencionados deben ser corregidos de manera precisa. Debido a que, al ser de gran tamaño y cantidad la información transmitida en paquetes, el receptor necesita realizar todo el proceso de sincronismo en un corto intervalo de tiempo; de no ser así, los paquetes se pierden y la información debe ser reenviada. Ante esto, uno de los requisitos para los algoritmos de sincronismo es poseer una alta tasa de acierto a expensas de un costo computacional lo más bajo posible [5].

1.5. Estado del arte de los algoritmos de sincronismo en sistemas OFDM.

Tras un repaso a las investigaciones realizadas en este campo [5], [6], [8], [15]–[17], se han propuesto ciertos algoritmos para resolver el problema de estimación de tiempo o errores de sincronismo de paquetes. En general, los métodos de sincronización en los sistemas OFDM se dividen en dos categorías: los *blind methods* o ‘métodos ciegos’ y aquellos que utilizan uno o más preámbulos al inicio de la trama (*data-aided*).

La mayoría de las soluciones prácticas basadas en un preámbulo para la sincronización de tramas y tiempos que pueden ser encontradas en la literatura, se basan en la autocorrelación de la señal recibida, tal como fue propuesto por Schmidl y Cox, quienes fueron los primeros en formular un algoritmo para la estimación de tiempo, donde se utilizó un preámbulo basado en dos patrones repetitivos: diez símbolos de entrenamiento cortos seguidos por dos símbolos de entrenamiento largos [15]. En [5] se presentó la implementación de algunos algoritmos de

sincronización para receptores OFDM en FPGA considerando los parámetros del preámbulo definido en el estándar IEEE802.11a, donde la detección del inicio del paquete fue realizada a través de técnicas de correlación, autocorrelación y correlación cruzada.

De forma similar, en el trabajo presentado por Zhou y Saito, la detección de la posición de inicio de símbolo fue realizada con las técnicas ya mencionadas, además de aplicar un método de sincronismo basado en la estructura del preámbulo, donde el umbral de detección se ajusta adaptativamente de acuerdo a la potencia de la señal. Dada su ligera complejidad, éste método realiza una sincronización rápida y precisa, incluso en escenarios con baja relación señal/ruido (*Signal to Noise* Ratio - SNR), gran desfase de frecuencia y/o desvanecimiento multitrayecto, resultando favorable su aplicación en sistemas con IEEE802.11a [8].

Por otro lado, ha sido de gran importancia el análisis del método de sincronización ante el desplazamiento de frecuencia de portadora (CFO), como lo hicieron Peng y Wen. El algoritmo propuesto se compuso de etapas de adquisición y de rastreo para la estimación y compensación del CFO. Tras los procesos de sincronismo y ecualización, la magnitud del vector de error fue usado para evaluar el rendimiento del sistema tanto en canales AWGN como en aquellos con multitrayecto [16].

1.5.1. Detección de paquete.

La detección de paquete puede ser considerada como la primera etapa del sincronismo. Para lograr un resultado eficiente se realiza la técnica de correlación de la señal recibida con una copia de la misma que posee un retraso de un símbolo (corto) de duración.

Esto se traduce en la Ecuación (1.12)

$$R(d) = \sum_{m=0}^{L-1} (r_{d+m}\, r_{d+m+L}^{*}) \tag{1.12}$$

Donde:

$R(d)$: Autocorrelación

r_{d+m}^{*}: Conjugada de la señal recibida

r_{d+m+L}: Señal recibida con retraso L

m: Factor de muestreo

L: Longitud de un símbolo (corto = 16 muestras)

Posteriormente, la potencia de la señal recibida es obtenida mediante la Ecuación (1.13)

$$P(d) = \sum_{m=0}^{L-1} |r_{d+m+L}|^2 \tag{1.13}$$

Esta potencia es usada para la comparación descrita en la Ecuación (1.14) para la detección del paquete como se describe en [15], donde se consideró que un paquete detectado ocurre cuando los valores obtenidos de dicha comparación sobrepasan un determinado umbral (1.15). Dicho umbral permite limitar la ocurrencia de errores en la detección, por lo que es escogido en función a los valores de potencia que se registran de la señal recibida [18].

$$M(d) = \frac{|R(d)|^2}{(P(d))^2} \tag{1.14}$$

$$M(d) > umbral \tag{1.15}$$

La operación de la Ecuación (1.14) puede ser simplificada a través del cálculo de $|R(d)_{Real}| + |R(d)_{Imag}|$ en reemplazo de $|R(d)|^2$; por tanto, la detección puede ser realizada mediante la comparación de $|R(d)|^2$ con la potencia media de la señal multiplicada por un valor de umbral específico [18], como se muestra en la Inecuación (1.16).

$$|R(d)|^2 > 0.5 * P(d) \tag{1.16}$$

1.5.2. Sincronismo de tiempo.

El grado de precisión en la detección del instante de tiempo en la que los símbolos OFDM inician y finalizan, determina el grado de eficiencia de un sincronismo temporal. En la literatura actual, se describe algoritmos de sincronismo en el tiempo en los cuales las técnicas de autocorrelación y correlación cruzada son utilizadas para determinar la posición correcta de los símbolos en el tiempo por parte del receptor; como por ejemplo en [19], donde Fort y otros autores realizan la comparación de las éstas técnicas, en función a su desempeño, complejidad computacional y de hardware.

La estructura del preámbulo, para el estándar IEEE802.11a, es conocido tanto en el transmisor como el receptor. Esto facilita la implementación de un algoritmo de sincronismo temporal basado en la técnica de correlación cruzada entre la señal recibida y los STS ya conocidos, encontrando posteriormente la posición del máximo valor de dicha operación [20].

El cálculo de la correlación cruzada es expresado en la Ecuación (1.17)

$$\Lambda(d) = \sum_{m=0}^{L-1} (c_m^* \, r_{d+m}) \tag{1.17}$$

Donde:

$\Lambda(d)$: Correlación cruzada

c_m^*: Compleja conjugada de las muestras del preámbulo

r_{d+m}: Señal recibida

m: Factor de muestreo

L: Longitud de un símbolo (16 muestras para STS y 64 muestras para LTS)

Como se explica en [5], ésta técnica involucra L multiplicaciones complejas para cada valor de salida, lo que representa un aumento en el costo computacional del bloque de sincronismo; sin embargo, la detección es más precisa respecto a la técnica de autocorrelación.

El cálculo del valor máximo en éste ámbito, se realiza a través de la Ecuación (1.18), utilizando un detector de pico para los símbolos de la secuencia de entrenamiento larga o LTS.

$$d_{xc\,max} = \arg\ max\ (|\Lambda(d)|) \tag{1.18}$$

Generalmente, el uso de los símbolos LTS es preferido ante el uso de STS, dado que, en este último, son detectados 16 picos, mientras que en LTS solamente se detectan 2, lo que reduce la carga computacional del algoritmo.

1.5.3. Sincronismo de frecuencia.

El sincronismo en frecuencia tiene como objetivo detectar, corregir y compensar el desplazamiento de frecuencia de portadora (CFO) y el error de fase de portadora ϕ(t); ambos errores originados por la presencia de ICI [9], producida por el efecto Doppler del canal, cuya repercusión se traduce en la destrucción de la ortogonalidad entre las subportadoras.

El efecto Doppler desencadena una variación de la señal en el dominio del tiempo, la cual está estrechamente relacionada con el movimiento del transmisor o receptor, que se extiende al dominio de la frecuencia [21]. El desplazamiento en frecuencia entre la señal transmitida y la receptada posee, consecuentemente, una diferencia de fase. Aquella diferencia entre ambas señales es proporcional al desplazamiento en frecuencia, y viceversa, como se aprecia en la Ecuación (1.19).

$$\phi = 2\pi t f_{\Delta} \tag{1.19}$$

Desafortunadamente, la fase de señal en el dominio de la frecuencia también está influenciada por la fase del canal. Para deshacerse del efecto de canal durante la estimación de compensación de temporización se puede calcular la diferencia de fase entre subportadoras adyacentes, dado que las subportadoras adyacentes suelen sufrir un desvanecimiento de canal similar y que se elimina la fase de canal común [21].

La diferencia de fase es calculada mediante la comparación de las señales discretas del transmisor y del receptor. Al ser proporcionales, la diferencia de fase con el desplazamiento

de frecuencia, se puede encontrar uno de los valores para determinar el otro; esto, gracias a la Ecuación (1.20) que relaciona dichos términos.

$$\Delta_f = \frac{\phi}{2\pi * n * T_s} \tag{1.20}$$

La señal transmitida definida en la Ecuación (1.21), al pasar por el canal, se ve afectada por la introducción de un determinado desplazamiento en frecuencia, llegando al receptor con una frecuencia de portadora diferente, lo que se refleja en la Ecuación (1.22)

$$x_n = s_n * exp(j2\pi f_{tx} n T_s) \tag{1.21}$$

$$r_n = s_n * exp(j2\pi \Delta_f n T_s) \tag{1.22}$$

Donde:

x_n: Señal transmitida

r_n: Señal recibida

s_n: Símbolos cortos de entrenamiento

f_{tx}: frecuencia de portadora en transmisor

n: Número de muestra

T_s: Tiempo de muestreo, $T_s = \frac{1}{f_s}$

Δ_f: Diferencia de frecuencias de portadoras de Tx y Rx, $\Delta_f = f_{tx} - f_{rx}$

Su comparación es realizada mediante la técnica de autocorrelación expresada en la Ecuación (1.23) a partir de una variación de (1.12), donde L = 16 debido a que se utiliza la secuencia corta de entrenamiento

$$R(d) = \sum_{m=0}^{L-1} (r_{d+m}\, r_{d+m+L}^{*}) \tag{1.23}$$

$$R(d) = \sum_{m=0}^{L-1} \left[s_{d+m}\, exp\left(j2\pi\Delta_f (d+m) T_s\right)\right]\left[s_{d+m+L}\, exp\left(j2\pi\Delta_f (d+m+L) T_s\right)\right]^{*} \tag{1.24}$$

$$R(d) = exp\left(-j2\pi\Delta_f L T_s\right) \sum_{m=0}^{L-1} (s_{d+m}\, s_{d+m+L}^{*}) \tag{1.25}$$

El resultado de esta auto correlación es proporcional a la diferencia de fase, con lo que se puede establecer la Ecuación (1.26)

$$\phi \approx \arg[R(d)] \tag{1.26}$$

Tal como se mencionó anteriormente, referente a la proporcionalidad entre un valor y otro, el desplazamiento en frecuencia es calculado en función a ϕ, reemplazando dicho termino en la Ecuación (1.27 o 1.28)

$$\Delta_{\mathrm{f}}=\frac{\arg[R(d)]}{2\pi * 16 * T_s} \tag{1.27}$$

$$\Delta_{\mathrm{f}}=\frac{\arg[R(d)]}{2\pi * 16 * \frac{1}{f_s}} \tag{1.28}$$

Una vez obtenido el valor de Δ_f, se realiza la compensación de este desplazamiento a través de la multiplicación de la señal recibida por una exponencial compleja negativa cuya frecuencia es el valor del Δ_f calculado, como se aprecia en la Ecuación (1.29)

$$s_d = r_d \; exp(-j2\pi\Delta_{\mathrm{f}} n T_s) \tag{1.29}$$

CAPÍTULO II:

DISEÑO Y SIMULACIÓN DE ALGORITMOS

El desarrollo de un algoritmo involucra dos etapas: la simulación y la implementación. Las simulaciones permiten realizar un sin número de pruebas dentro de escenarios virtuales controlados, dando al usuario la oportunidad de establecer y controlar las variables dentro de los mismos. La proximidad de los resultados de simulación ante aquellos obtenidos a partir de las pruebas de implementación, dependen de la robustez tanto del algoritmo como del programa de simulación y de las condiciones de los escenarios tales como el ruido ficticio del canal.

Este capítulo describe el diseño y simulación de los algoritmos de sincronización en el lenguaje de programación Python y la herramienta GNURadio. En la Sección 2.1 se determinan los recursos y las limitaciones presentes en el desarrollo de las simulaciones, además de un breve resumen de las herramientas utilizadas y su contribución al desarrollo de los algoritmos.

En la Sección 2.2 se detalla la construcción del preámbulo, las simulaciones del ruido y el CFO añadidos por el canal junto con el diseño de los algoritmos de sincronización (detección del paquete, sincronismo de tiempo y sincronismo de frecuencia) desarrollados en Python. Finalmente, en la Sección 2.3, se precisa las pautas a seguir en el diseño de los bloques que realizan las tareas de sincronización de la señal recibida, desarrollados en GNURadio.

2.1. Recursos y limitaciones.

Los recursos usados en el diseño y simulación de los algoritmos de sincronismo fueron Python y GNURadio, ya que permitieron el uso de la herramienta USRP para la implementación de dichos algoritmos con mayor facilidad. Los recursos usados durante las simulaciones fueron proporcionados por el laboratorio de Telecomunicaciones de La Universidad Técnica Particular de Loja. En la Tabla 2.1 se especifican las características de los recursos usados

Tabla 2.1. Detalles de los recursos usados

Recurso	Detalles
Computador portátil	Lenovo ThinkPad L430
Ubuntu	Versión 14.04 LTS
Python	Versión 2.7.13
Spyder	Versión 3.1.13
GNURadio	Versión 3.7.10.2
UHD	Versión 3.11

Fuente: Autores
Elaboración: Autores

Las limitaciones encontradas a lo largo de las simulaciones fueron: el limitado conocimiento previo del lenguaje de programación Python, la instalación del software necesario, el manejo

de Spyder y la definición de la función AWGN. Gracias al conocimiento adquirido en MATLAB y las Tablas de conversión de comandos MATLAB – Python [22] se superó la limitación. El canal AWGN no tiene un comando específico en Python como lo tiene MATLAB, por lo que para su simulación se tuvo que definir la función *awgn*.

Al principio, una de las limitaciones en la creación de bloques GNURadio fue fijar el tamaño del vector *Source*, lo que ocasionó una variación de los valores del vector con cada repetición, obteniendo como resultado un vector de salida diferente al simulado en Python. Posteriormente, esto fue corregido con el apropiado manejo de los valores de entrada de cada bloque de GNURadio.

2.1.1. Python

La simulación de los algoritmos de sincronismo se realizó usando el ambiente Spyder, un potente entorno de desarrollo interactivo para el lenguaje Python con funciones avanzadas de edición, pruebas interactivas, depuración e introspección y un entorno informático numérico gracias al soporte de IPython (intérprete interactivo mejorado y/o terminal de Python) junto a librerías de funciones populares como NumPy para álgebra lineal, SciPy para procesamiento de señal e imagen y Matplotlib para trazado interactivo 2D / 3D [23].

Python es un lenguaje de programación que permite trabajar más rápido e integrar los sistemas de manera más efectiva. Es fácil de aprender; usado en muchas aplicaciones y lo más importante para nuestro trabajo es que posee una licencia de código abierto que es compatible con la Licencia pública general de GNURadio. Los tipos de datos y los comandos usados de Python para las simulaciones se plasman en las Tablas 2.2 y 2.3.

Tabla 2.2. Tipos de datos usados en Python

Tipo	Clase	Notas
str	Cadena	Inmutable
list	Secuencia	Mutable, puede contener objetos de diversos tipos
int	Número entero	Precisión fija, convertido en *long* en caso de *overflow*.
long	Número entero	Precisión arbitraria
float	Número decimal	Coma flotante de doble precisión
complex	Número complejo	Parte real y parte imaginaria *j*.
bool	Booleano	Valor booleano verdadero o falso

Fuente: [24]
Elaboración: Autores

Tabla 2.3. Comandos de Python usados en las simulaciones

Comando	Uso	Comando	Uso
hstack	Concatena dos vectores	fft	Transformada de Fourier del vector X
arange	Crea secuencias	ifft	Transformada inversa de Fourier del vector X
zeros	Crea un vector de ceros	fft.shift	Reordena el vector X en orden creciente de frecuencia.
array	Crea un vector	conj	Calcula el conjugado de X
sqrt	Raíz cuadrada	abs	Calcula el absoluto de X
txt.open	Abre un documento .txt	plot	Dibuja una señal
txt.write	Escribir en un .txt	subplot	Muestra varias imágenes en una sola Figura
txt.close	Cerrar un documento .txt	amax	Muestra el valor del punto máximo de una señal
angle	Regresa el ángulo de un argumento complejo	argmax	Muestra la posición del punto máximo de una señal
xcorr	Calcula la correlación de dos vectores	random.rand	Crea una matriz de la forma dada y rellena con muestras aleatorias de una distribución uniforme
scipy.fromfile	Realiza operaciones con los datos de un archivo externo	variance	Calcula la varianza de un conjunto de datos

Fuente: [24]
Elaboración: Autores

2.1.2. SDR - GNURadio

Software Defined Radio o SDR es una tecnología donde las señales portadoras que se modulan y transmiten por un canal son definidas mediante software. En la Figura 2.1 se muestra el concepto de SDR, las formas de onda son generadas como señales digitales que se convierten de digital a analógico a través de un convertidor digital a analógico (*Digital to Analog Converter* – DAC) de banda ancha y con la posibilidad de subir la señal de IF a RF. De la misma manera, el receptor emplea un ADC de banda ancha que captura todos los canales del nodo del software radio. El ADC digitalizará la señal y la pasará al procesador de banda base para otros procesos; demodulación, codificación de canal, codificación de fuente, etc.

SDR se aplica a una amplia gama de áreas dentro de la industria inalámbrica, actuando como una herramienta relativamente nueva que ha estado creciendo gracias a las ventajas que ofrece como, por ejemplo:

- Reutilización de hardware

- Menor costo de implementación
- Compatibilidad con dispositivos heredados
- Actualizaciones sencillas en modulación, codificación de control de errores y protocolos de enlace de datos.
- Recibir y transmitir nuevos protocolos de comunicaciones simplemente mediante la actualización de software sobre el hardware existente, evitando incurrir en el cambio total de la infraestructura de comunicaciones que ya se encuentra implementada.

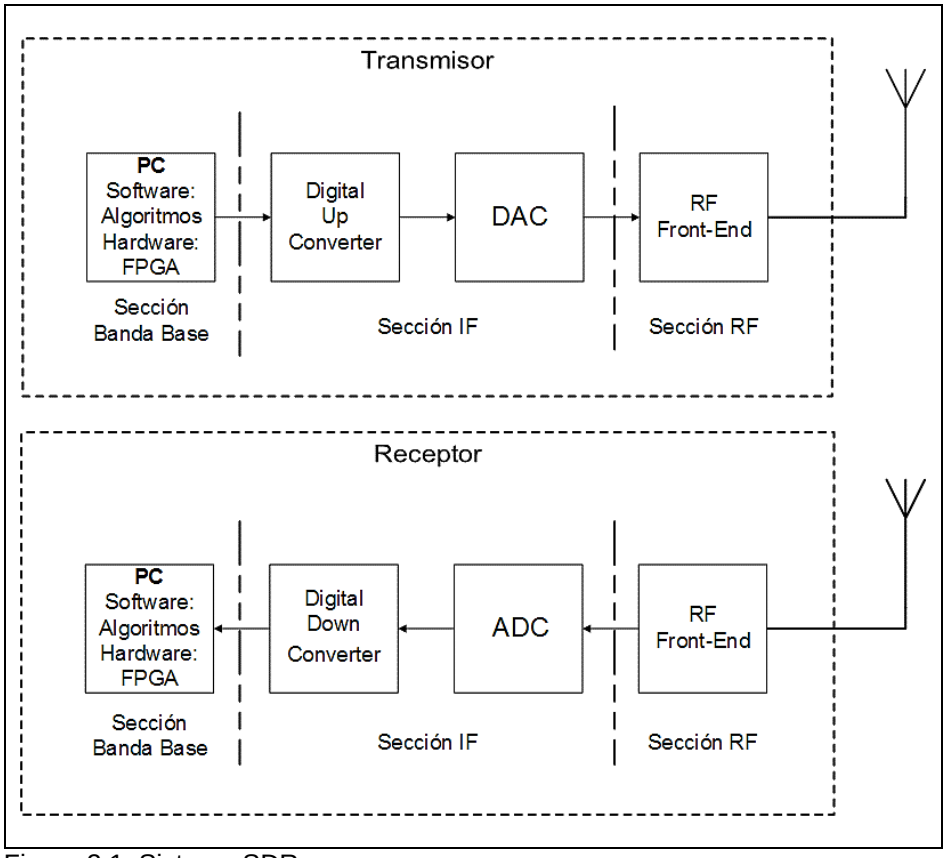


Figura 2.1. Sistema SDR
Fuente: [10], [11], [25]
Elaboración: Autores

Por otro lado, la herramienta GNURadio es un software libre formado por un conjunto de archivos y aplicaciones agrupados en librerías, que permiten manipular señales mediante procesado digital para realizar el diseño de sistemas SDR. GNURadio funciona en los sistemas operativos Linux, Mac OS y Windows y puede ser usado conjuntamente con hardware externo como, por ejemplo, los dispositivos USRP, para implementar físicamente el sistema SDR, o bien, sin incluir hardware, para trabajar en un entorno de simulación [26].

La construcción de bloques propios de procesamiento (*Out Of Tree* - OTT) se lleva a cabo mediante la programación en alto nivel, usualmente Python o C++. Sin embargo, existe la posibilidad de utilizar una interfaz gráfica, llamada GNURadio Companion (GRC), que permite

realizar diseños completos sin escribir líneas de código, simplificando así el nivel de complejidad.

Para construir un sistema SDR con GNURadio se debe crear una especie de grafo, donde los vértices o nodos son los bloques de procesado de señal y las aristas o arcos que unen esos nodos representan el flujo de datos entre ellos. En cualquier aplicación de GNURadio siempre ha de haber una fuente, un bloque donde se realice el proceso y un sumidero. Las fuentes (*Sources*) están caracterizadas por tener un solo puerto de salida como, por ejemplo: un fichero, un micrófono o un dispositivo USRP. Los sumideros (*Sinks*), tienen un solo puerto de entrada como puede ser un fichero, la tarjeta de sonido, un dispositivo USRP, visualizadores gráficos. Por último, son necesarios también los bloques de procesado de señal como son filtros, amplificadores, moduladores, operadores lógicos [25].

Los tipos de datos que se manejan son: *byte* (1 byte de datos), *short* (2 bytes de datos), *int* (4 bytes de datos), *float* (4 bytes de datos para números en punto flotante) y *complex* (8 bytes de datos, un float para cada componente). Su clasificación se presenta en la Figura 2.2.

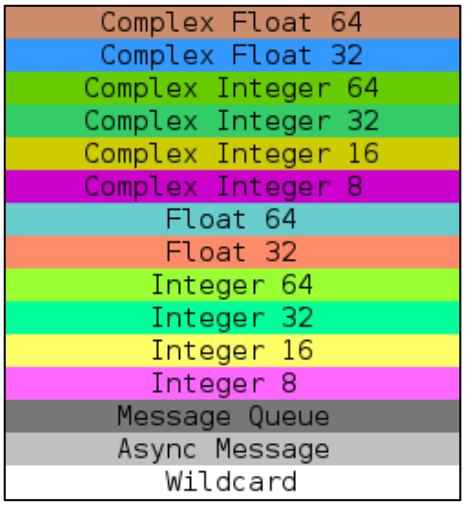


Figura 2.2. Clasificación de tipo de datos de GNURadio
Fuente: [25]
Elaboración: [25]

Para acceder a la herramienta, simplemente hay que llamarla desde un terminal de Linux escribiendo el comando *gnuradio-companion.* La creación de los bloques personalizados o módulos OOT, que no figuran dentro los bloques preinstalados de GRC, se realiza de manera rápida y sencilla a través de la herramienta *gr_modtool* de la terminal de comandos del sistema operativo, brindando opciones al programador que van desde el establecimiento del nombre del bloque hasta la activación de las pruebas de calidad o *QA Test* (*Quality Assurance*) [27].

En la Figura 2.3 se muestran las interacciones entre las diferentes capas o niveles de la arquitectura *GNURadio*. Los bloques de procesado de la señal se implementan en C++ y son convertidos a Python mediante el programa SWIG.

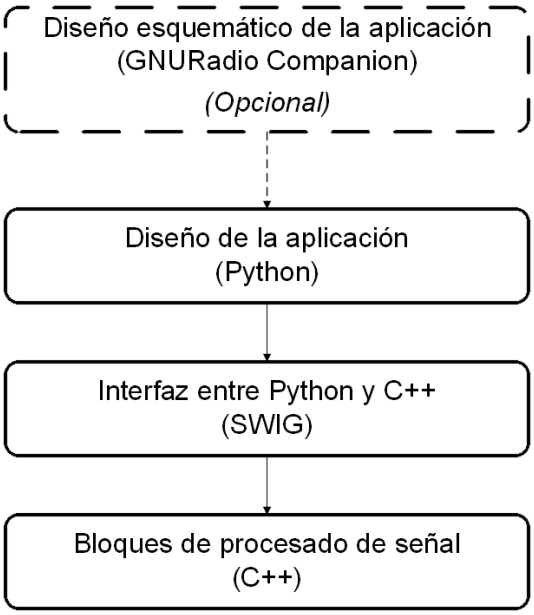


Figura 2.3. Arquitectura GNURadio
Fuente: [25]
Elaboración: Autores

2.2. Simulación en entorno Python

2.2.1. Generación del preámbulo

En base a los parámetros establecidos en el estándar IEEE802.11a [7], se realizó la construcción del preámbulo en el entorno Python. Dentro de toda la etapa de simulaciones, el preámbulo obtenido sirve de base para la ejecución y pruebas de los diferentes algoritmos de sincronismo que se han planteado en el presente trabajo. El desarrollo del preámbulo puede ser apreciado en el Anexo A, cuyo código fue desarrollado en el entorno Python.

En las Tablas 2.4 y 2.5 se presentan las 16 y 64 muestras de un símbolo STS y LTS, respectivamente, que fueron concatenadas conforme se estableció en la Sección 1.2.1 para generar el preámbulo.

Tabla 2.4. Muestras del símbolo STS

N	Muestra	N	Muestra
1	0.046+0.046i	9	0.046+0.046i
2	-0.132+0.02i	10	0.002-0.132i
3	-0.014-0.079i	11	-0.079-0.014i
4	0.143-0.013i	12	-0.013+0.143i
5	0.092+0.000i	13	0.000+0.092i
6	0.143-0.013i	14	-0.013+0.143i
7	-0.014-0.079i	15	-0.079-0.014i
8	-0.132+0.002i	16	0.002-0.132i

Fuente: [7]
Elaboración: Autores

Tabla 2.5. Muestras del símbolo LTS

N	Muestra	N	Muestra	N	Muestra	N	Muestra
1	0.156+0.000i	17	0.063-0.063i	33	-0.156+0.000i	49	0.063+0.063i
2	-0.005-0.120i	18	0.037+0.098i	34	0.012-0.098i	50	0.119+0.004i
3	0.040-0.111i	19	-0.057+0.039i	35	0.092-0.106i	51	-0.023-0.161i
4	0.097+0.083i	20	-0.131+0.065i	36	-0.092-0.115i	52	0.059+0.015i
5	0.021+0.028i	21	0.082+0.092i	37	-0.003-0.054i	53	0.025+0.059i
6	0.060-0.088i	22	0.070+0.014i	38	0.075+0.074i	54	-0.137+0.047i
7	-0.115-0.055i	23	-0.060+0.081i	39	-0.127+0.021i	55	0.001+0.115i
8	-0.038-0.106i	24	-0.057-0.022i	40	-0.122+0.017i	56	0.053-0.004i
9	0.098-0.026i	25	-0.035-0.151i	41	-0.035+0.151i	57	0.098+0.026i
10	0.053+0.004i	26	-0.122-0.017i	42	-0.057+0.022i	58	-0.038+0.106i
11	0.001-0.115i	27	-0.127-0.021i	43	-0.060-0.081i	59	-0.115+0.055i
12	-0.137-0.047i	28	0.075-0.074i	44	0.070-0.014i	60	0.060+0.088i
13	0.025-0.059i	29	-0.003+0.054i	45	0.082-0.092i	61	0.021-0.028i
14	0.059-0.015i	30	-0.092+0.115i	46	-0.131-0.065i	62	0.097-0.083i
15	-0.023+0.161i	31	0.092+0.106i	47	-0.057-0.039i	63	0.040+0.111i
16	0.119-0.004i	32	0.012+0.098i	48	0.037-0.098i	64	-0.005+0.120i

Fuente: [7]
Elaboración: Autores

El preámbulo se generó siguiendo los procesos del diagrama de flujo de la Figura 2.4

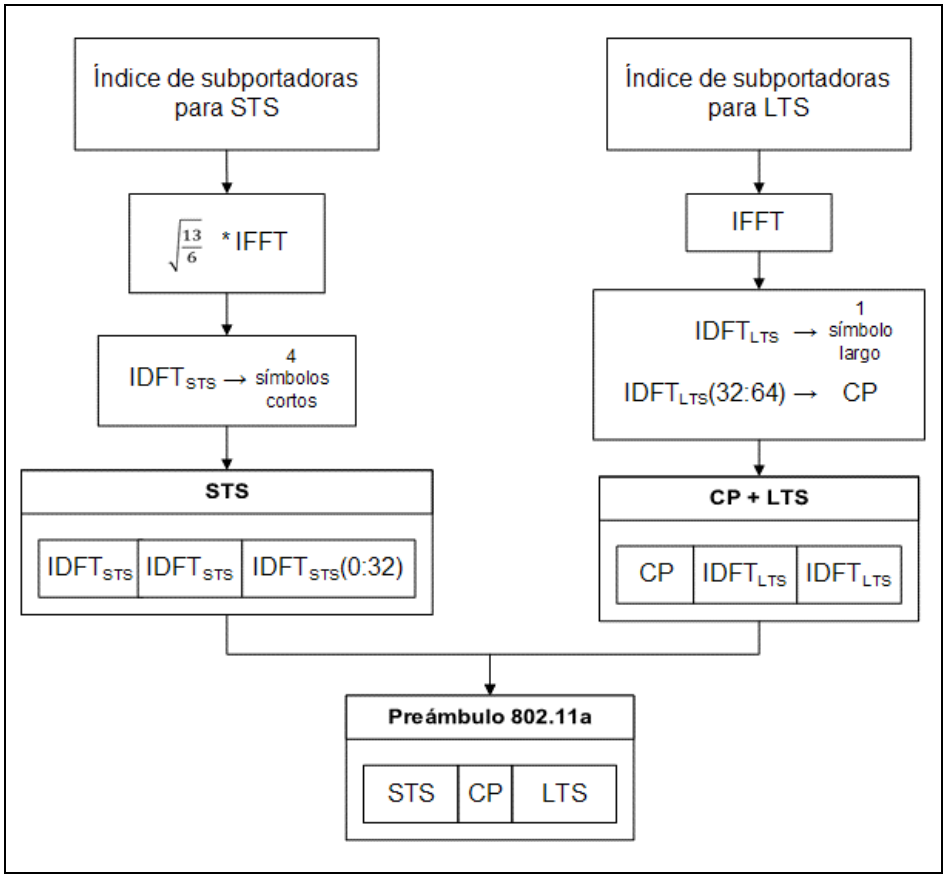


Figura 2.4. Flujograma de la generación del preámbulo IEEE802.11a
Fuente: Autores
Elaboración: Autores

2.2.2. Simulación del canal

Para lograr resultados precisos y con mayor apego a las situaciones reales que se dan en el entorno, la representación del comportamiento del canal es esencial. El modelo de canal implementado para evaluar el rendimiento de los próximos diseños, es denominado *tapped-delay* e incorpora los efectos de desvanecimiento por múltiple trayectoria de los modelos ETSI A y ETSI C [13], [28], desvío de frecuencia (CFO) y ruido gaussiano blanco (AWGN).

El canal seleccionado para las simulaciones de este documento se generó usando la técnica de estimación de canal asistida basada en símbolos de entrenamiento, bajo las siguientes condiciones: CFO de 0, 100 o 200 kHz y ruido AWGN de 10 dB, siguiendo el esquema de la Figura 2.5.

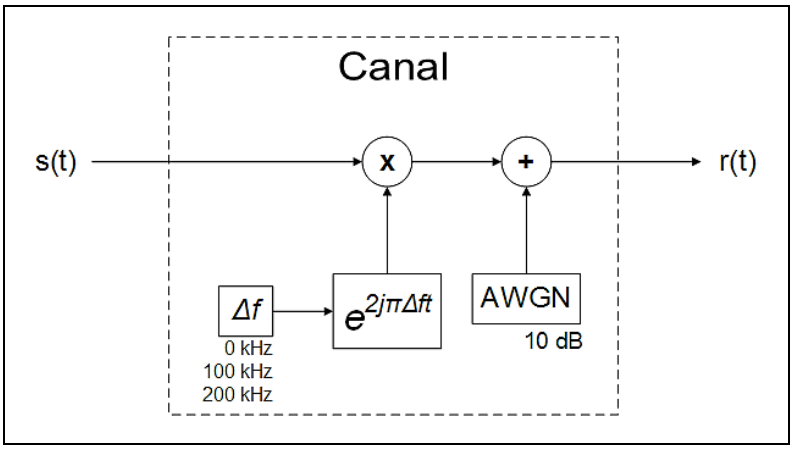


Figura 2.5. Modelo del canal
Fuente: Autores
Elaboración: Autores

Durante la simulación del canal surgió una limitación. En MATLAB, existe el comando *awgn(x, snr, 'measured')* que hace posible añadir ruido AWGN a una señal, donde *x* es el vector de la señal entrante, *snr* es de tipo escalar y especifica la relación señal/ruido por muestra, en dB, y *'measured'* establece que la potencia de la señal será medida antes de agregar el ruido; un ruido complejo en el caso de que la señal sea del mismo carácter [29]. Al ser una función propia del programa, resulta muy sencilla su aplicación. Sin embargo, no se da el mismo caso en Python, donde no existe una función propia que realice el mismo trabajo, por lo que se consideró conveniente utilizar una función personalizada.

La función de AWGN, expresada en Python como *awgn(input_signal, snr_dB, rate)* [30], se formó con los parámetros: *input_signal*, la señal a la que se le agrega el ruido, *snr_dB*, el valor de SNR del canal (10 dB en este trabajo) y *rate*, que tomó el valor de 0.5 en las simulaciones, por tratarse del caso de BPSK [11]. El desplazamiento en frecuencia se agregó al multiplicar la señal transmitida por la Ecuación (1.10) que representa significativamente la consecuencia del canal inalámbrico. La función desarrollada en el entorno Spyder, puede ser apreciada en el Anexo A, dentro de la simulación del algoritmo de generación de preámbulo IEEE802.11a.

2.2.3. Detección de paquete

La primera etapa del sincronismo es la detección del paquete. Este primer algoritmo detecta la presencia de la trama basándose en una operación de comparación entre la autocorrelación normalizada y la potencia recibida como se detalla en la Sección 1.5.1.

El algoritmo se formó con tres secciones de código, iniciando con la generación del preámbulo de valores complejos dentro de un arreglo y la introducción tanto del CFO como del ruido AWGN, siguiendo con las operaciones de autocorrelación y obtención de la potencia (ambas dentro de arreglos de la misma longitud), la ejecución del filtro CIC en ambos casos, y finalizando con la relación de ambos arreglos con la Inecuación (1.16). Su diagrama puede ser apreciado en la Figura 2.6; se generó el preámbulo como se detalla en la Sección 2.2.1, se le introdujo el valor de CFO de 100 kHz y se le adicionó ruido AWGN.

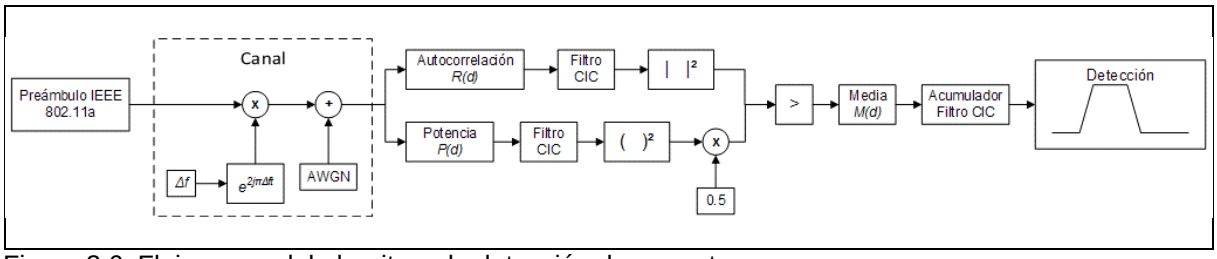


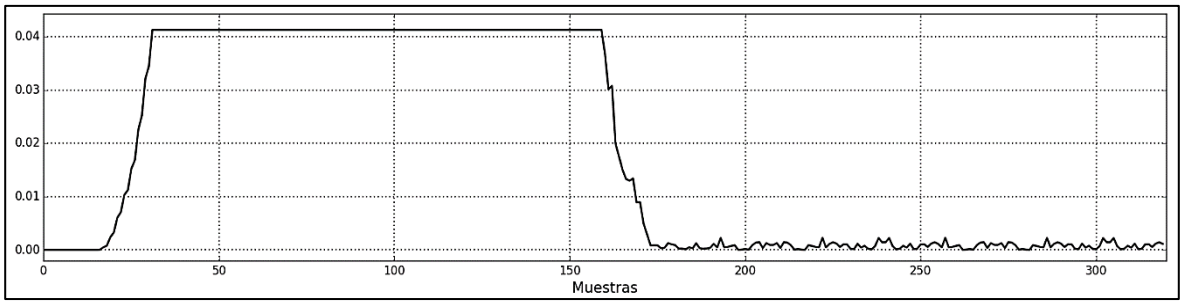
Figura 2.6. Flujograma del algoritmo de detección de paquete
Fuente: Autores
Elaboración: Autores

Posteriormente, se realizó las operaciones de autocorrelación normalizada de la señal entrante con la misma retrasada 16 muestras y el cálculo de la potencia media. Sus salidas, pasando por filtros CIC, se muestran en las Figuras 2.7(a), 2.7(b) y 2.7(c). Asimismo, estas operaciones se aplicaron al preámbulo en presencia de ruido AWGN y CFO, cuyo resultado se muestra en la Figura 2.7(d).

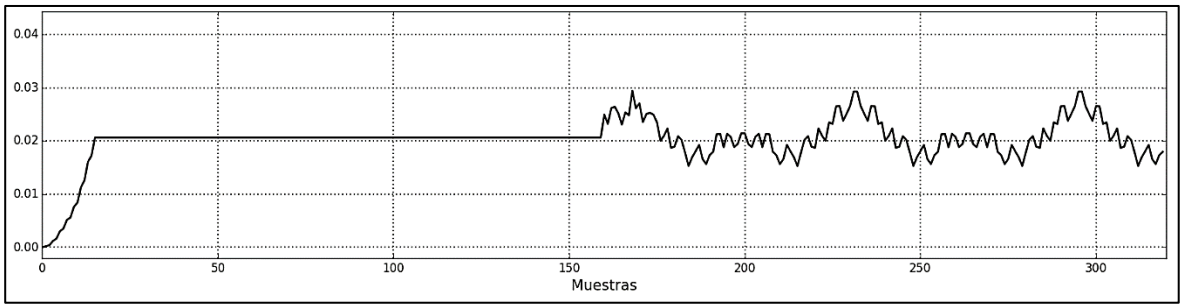
Finalmente, se comparó la autocorrelación con la potencia media de la señal, cuyo resultado pasó a través de un filtro para evitar la presencia de picos espurias provocados por el ruido del canal.

La detección es representada con un arreglo de tipo flotante de valores binarios a lo largo del tamaño del preámbulo; presentado el valor de 1 durante 135 muestras cuando la trama es detectada, mientras que los valores restantes son iguales a cero, con lo que se obtiene una salida gráfica de tipo escalón como se muestra en la Figura 2.8(a). El filtrado de la detección es efectuado por medio de un atrasador de 16 muestras [5], aplicado a la salida del comparador de la Ecuación (1.16).

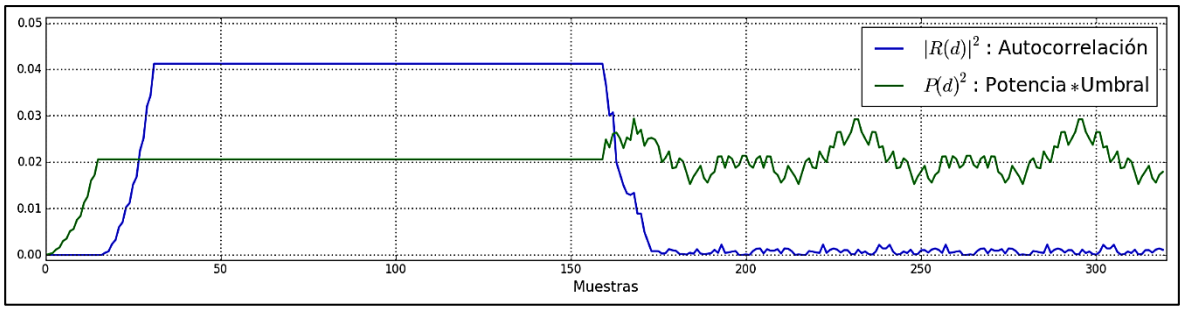
La elaboración del algoritmo de detección dentro del entorno Spyder se muestra en el Anexo B, en conjunto con sus correspondientes gráficas de salida..



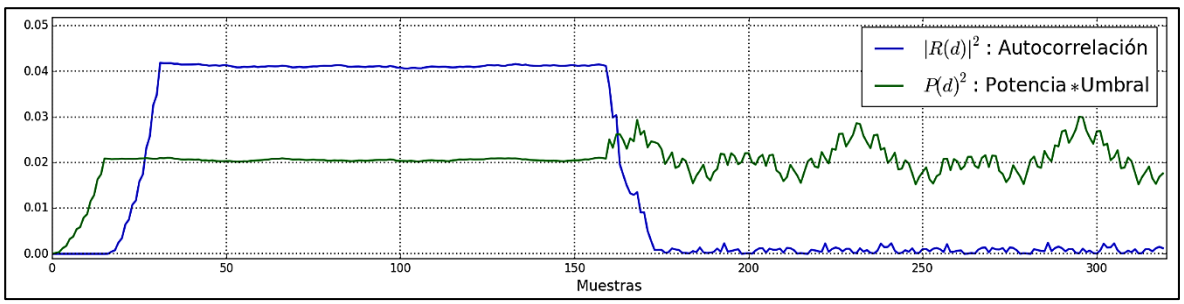
(a)



(b)



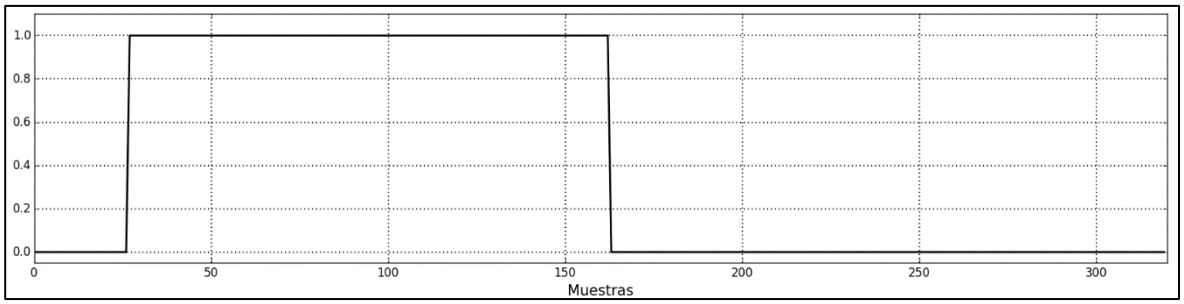
(c)



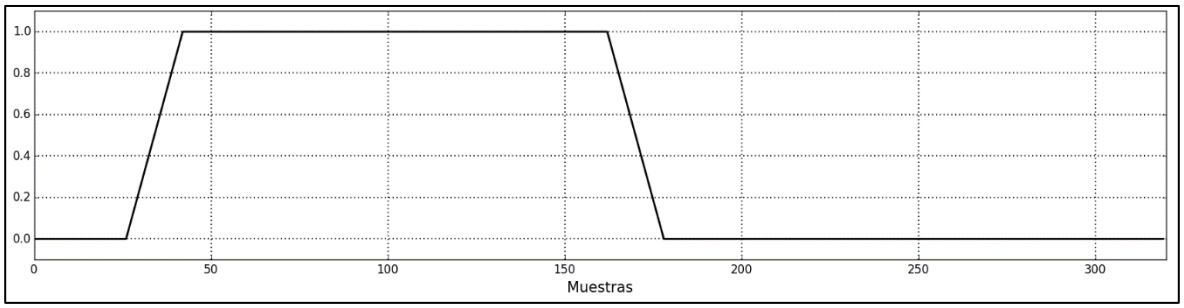
(d)

Figura 2.7. (a) Autocorrelación normalizada $|R(d)|^2$, (b) Potencia media, (c) Comparativa y (d) Comparativa con presencia de ruido AWGN y CFO
Fuente: Autores
Elaboración: Autores

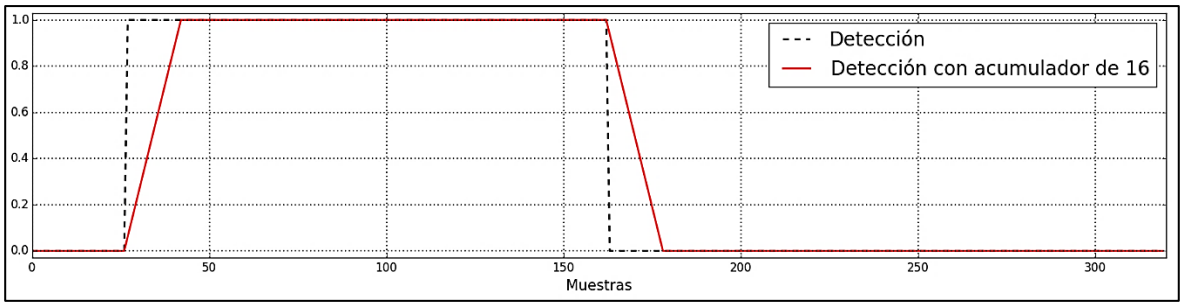
El resultado obtenido es también un arreglo de tipo flotante con valores que forman una señal trapezoidal, cuya gráfica se muestra en la Figura 2.8(b). La diferencia entre ambas salidas puede apreciarse en la Figura 2.8(c), donde se evidencia que los escalones o *mesetas* de ambas señales inician y finalizan en diferentes muestras. En la Figura 2.8(d) fueron agregados los efectos del canal, donde se evidenció una diferencia muy escasa respecto a lo obtenido en la Figura 2.8(c).



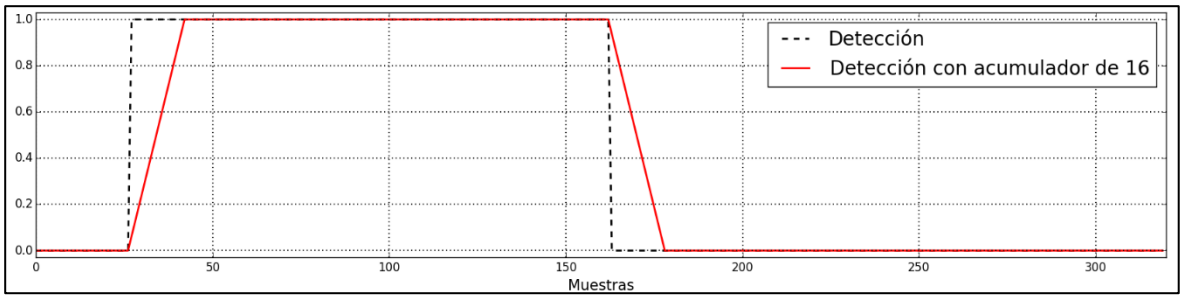
(a)



(b)



(c)



(d)

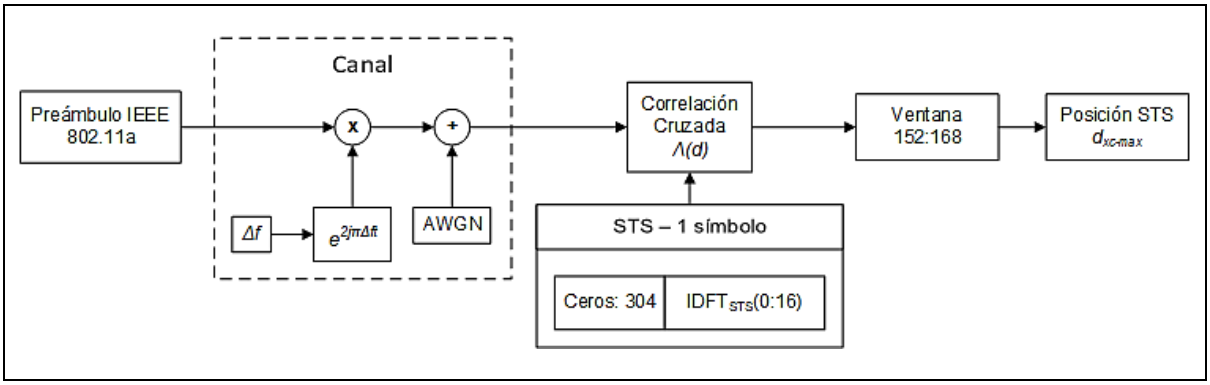
Figura 2.8. (a) Detección $|M(d)|^2$, (b) Detección con acumulador de 16 muestras, (c) Comparativa y (d) Comparativa con presencia de ruido AWGN y CFO
Fuente: Autores
Elaboración: Autores

Una vez detectada la presencia del paquete, los algoritmos de sincronismo de tiempo y de frecuencia pudieron ser puestos en marcha.

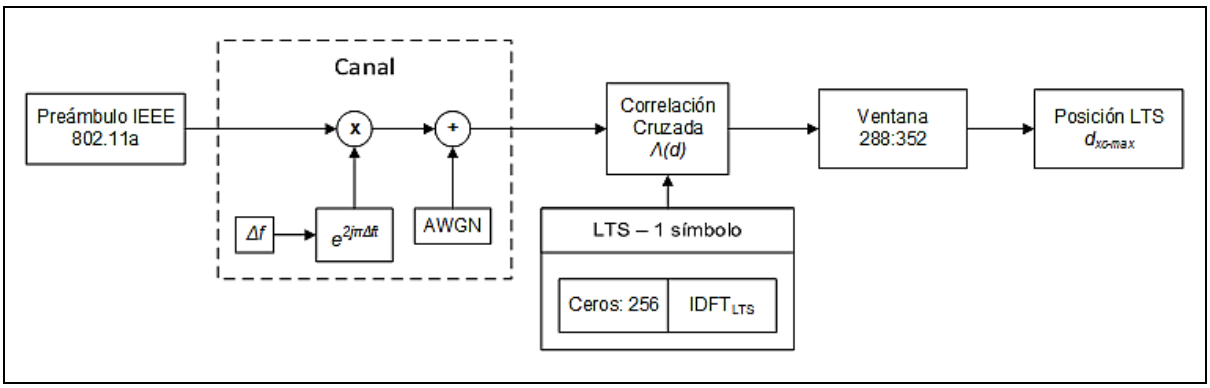
2.2.4. Sincronismo de tiempo

En tiempo, el objetivo es encontrar la posición de la muestra donde inicia el paquete, o uno de los fragmentos que lo componen. Como se conoce de antemano, el paquete es precedido por un preámbulo cuya estructura es prestablecida; razón por la que, en el receptor, la

detección de un determinado fragmento del preámbulo puede ser efectuada de forma precisa. Este tipo de sincronismo se realizó siguiendo las pautas que se detallan en la Sección 1.5.2, y detección se compuso de dos etapas: la correlación cruzada de la señal con un símbolo corto y la correlación cruzada con un símbolo largo.



(a)



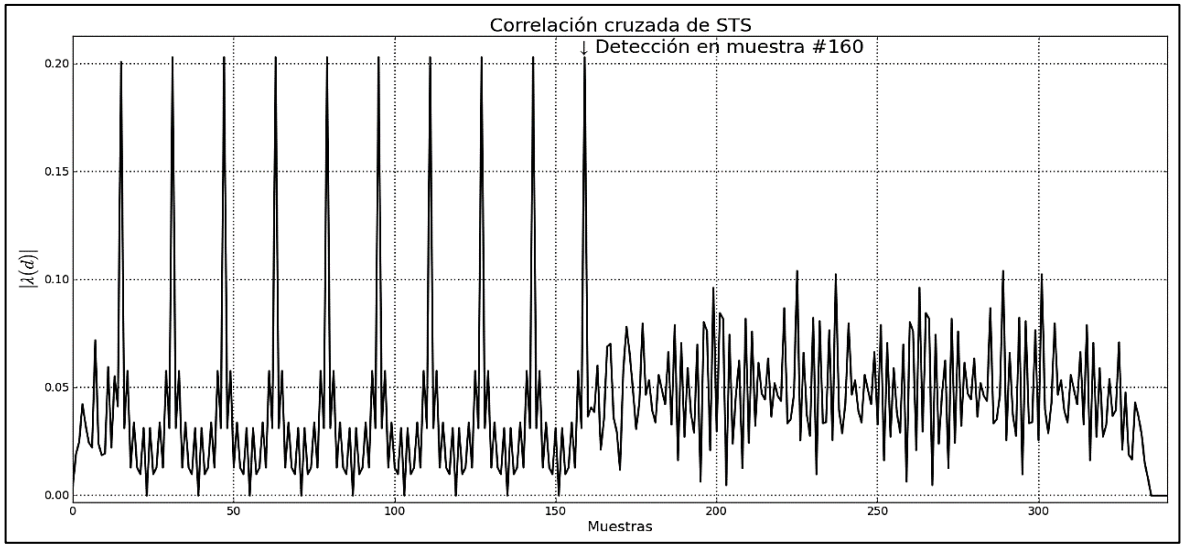
(b)

Figura 2.9. Flujograma de los algoritmos de Sincronismo de tiempo usando la secuencia (a) STS y (b) LTS
Fuente: Autores
Elaboración: Autores

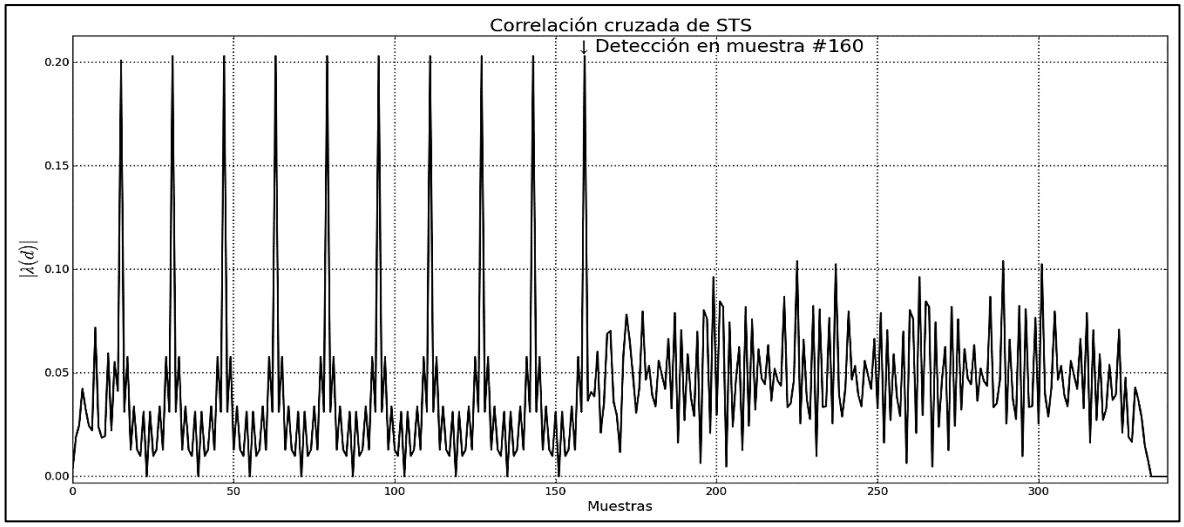
Se consideró la detección del símbolo final de ambas secuencias de entrenamiento en vista de que dichas posiciones representan el inicio del IG, en el caso de la secuencia corta, y el inicio de la información relevante, en el caso de la secuencia larga. Con la detección de una de estas posiciones, se deduce la existencia y duración de algún retraso temporal de la señal.

En la Figura 2.9 se muestran los pasos seguidos en el diseño del algoritmo, al igual que en el algoritmo de detección, aquí se usó el preámbulo como señal de entrada y se le agregó los efectos de AWGN y CFO. Al tener un preámbulo ya conocido por el receptor se usó la correlación cruzada para estimar el atraso temporal. La correlación cruzada multiplica la señal que llega con el ruido del canal por el símbolo original transmitido, es decir la señal entrante y un arreglo de 320 muestras conformado por un arreglo de ceros y un símbolo corto o un símbolo largo, según sea el caso (304 ceros y un símbolo corto de 16 muestras, o 256 ceros y un símbolo largo de 64 muestras).

La función de correlación cruzada entrega, en su salida, una señal con picos posicionados en la última muestra de cada uno de los símbolos, por lo que, en la señal resultante, la posición de los símbolos es localizada mediante la extracción del valor máximo dentro una determinada ventana. Como se conoce de antemano, tanto la STS como la LTS tienen una duración de 160 muestras, y juntas 320 muestras, por lo que los picos de sus últimos símbolos se encontrarán en las muestras 160 y 320, respectivamente.



(a)



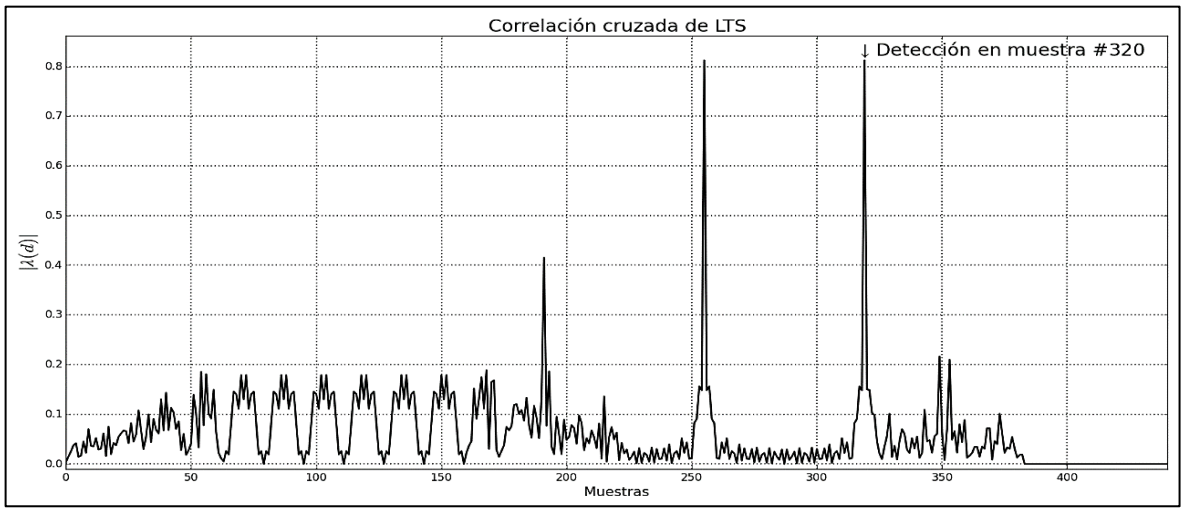
(b)

Figura 2.10. Señal obtenida al aplicar el algoritmo de correlación cruzada para (a) STS y para (b) STS con presencia de ruido AWGN y CFO
Fuente: Autores
Elaboración: Autores

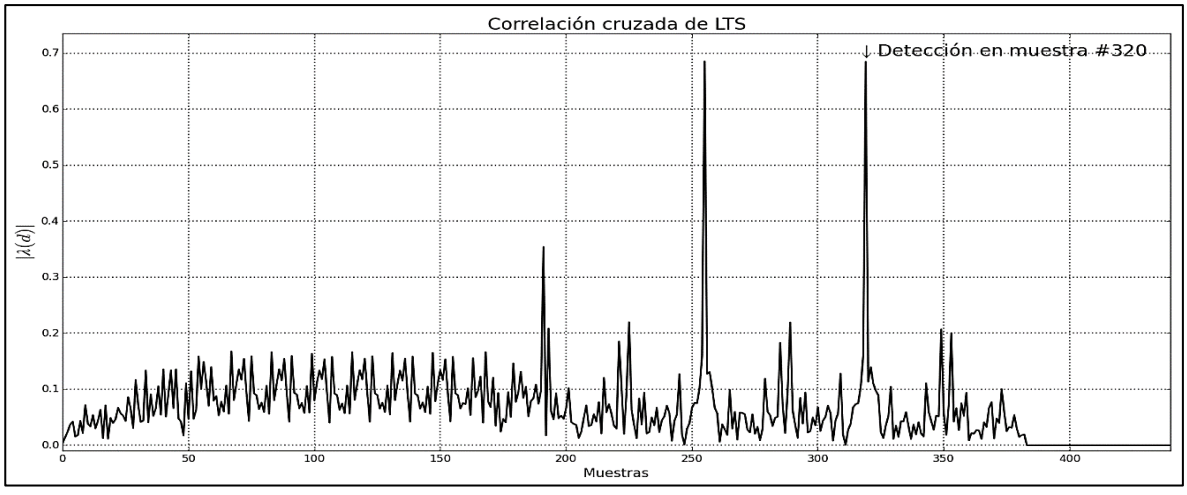
La ventana utilizada se generó con la duración de cada símbolo de las secuencias de entrenamiento, estableciéndose el límite inferior con la mitad de muestras antes de la posición esperada del símbolo y el límite superior con la otra mitad después de la posición; es decir, en el caso de STS, el símbolo tiene una duración de 16 muestras, por lo que los límites de la

ventana usada fueron 152 y 168. De la misma forma, en el caso de LTS, con un símbolo de 64 muestras, se usó una ventana comprendida entre las posiciones 288 y 352.

El resultado obtenido de las simulaciones de los algoritmos de correlación cruzada se observa en las Figuras 2.10(a) y 2.11(a), donde se presenta la posición detectada del último símbolo en STS y LTS, respectivamente. Al ser simulaciones controladas, las posiciones detectadas fueron las esperadas; es decir, para la secuencia STS la detección se ubicó en la muestra 160, y para la secuencia LTS se ubicó en la muestra 320. Al agregar ruido AWGN y CFO al preámbulo generado, se puede apreciar en las Figuras 2.10(b) y 2.11(b) que las salidas de las correlaciones cruzadas, en ambos casos, no sufrieron alteraciones notables, por lo que la detección de las posiciones esperadas se efectuó sin errores.



(a)



(b)

Figura 2.11. Señal obtenida al aplicar el algoritmo de correlación cruzada para (a) LTS y para (b) LTS con ruido AWGN y CFO
Fuente: Autores
Elaboración: Autores

2.2.5. Sincronismo de frecuencia

El sincronismo de frecuencia permite la compensación de los errores por desplazamiento de frecuencia portadora en base a la detección de la fase de la señal recibida, dado que se busca eliminar su variación angular frente a la señal transmitida. Con este antecedente, el desarrollo del algoritmo fue realizado como se explica en la Sección 1.5.3, donde se puntualizó que, mediante el uso de la autocorrelación de la señal se calculó la variación de fase Φ y a través de las Ecuaciones (1.20), (1.26) y (1.27) se obtuvo un aproximado en frecuencia del CFO.

En teoría, se entiende que la introducción del CFO en la señal por parte del canal inalámbrico equivale a la multiplicación del preámbulo generado por una exponencial compleja. Para revertirlo y/o compensarlo, en el receptor, se hace uso del mismo preámbulo como un arreglo de valores complejos multiplicado por una exponencial compleja (Ecuación (1.22)) de signo opuesto a la exponencial usada en el canal (Ecuación (1.21)). Si el parámetro Δf resulta ser igual o muy similar en ambas exponenciales, la diferencia de fase es compensada y el preámbulo puede ser sincronizado.

Para comprobar la precisión de la sincronización del preámbulo en las pruebas realizadas al compensar el desfase en el receptor, no se agregó el efecto del ruido AWGN simulado, por lo que el esquema de éste cambió respecto al que se había estado utilizando. La configuración del algoritmo de sincronismo de frecuencia puede ser apreciada en la Figura 2.12.

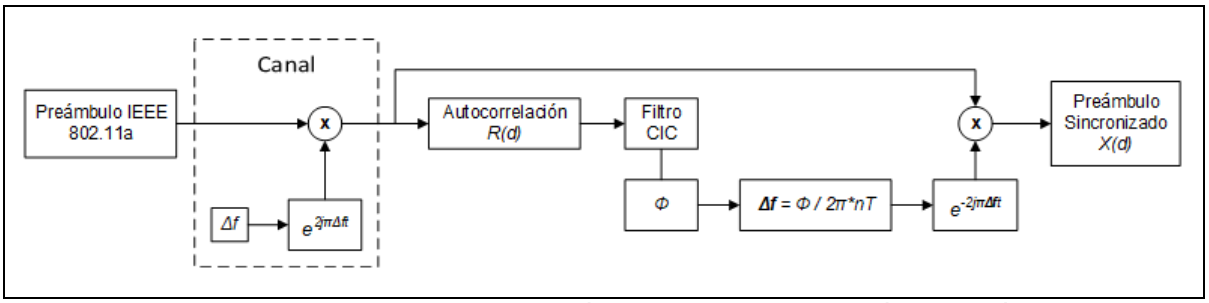
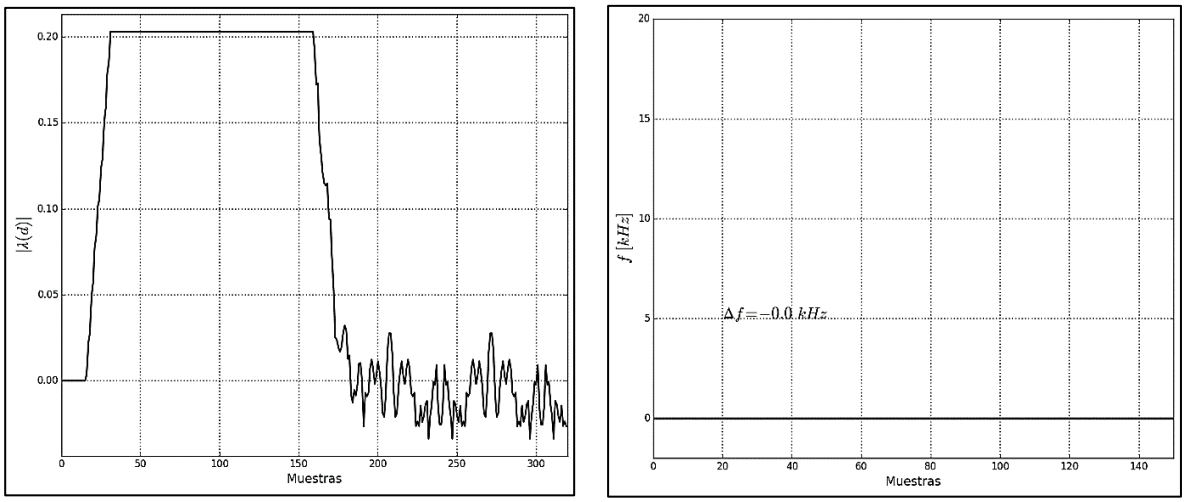


Figura 2.12. Flujograma del algoritmo de Detección de CFO y sincronización del preámbulo
Fuente: Autores
Elaboración: Autores

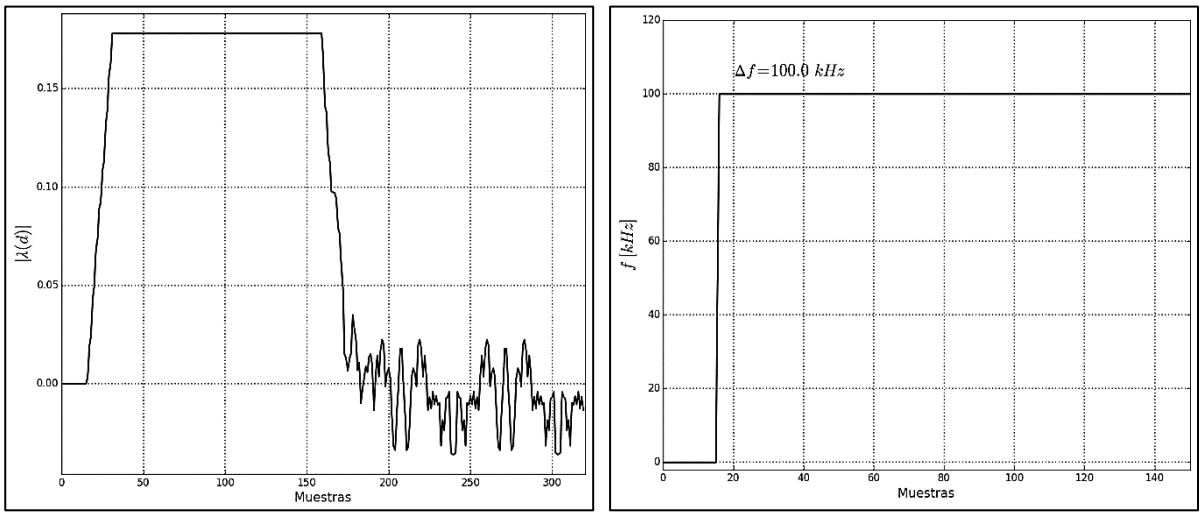
La extracción de la fase de la señal recibida se hizo por medio del cálculo del ángulo de uno de los valores de la meseta que presenta la autocorrelación, y mediante el uso de la Ecuación (1.28) se encontró el valor del CFO en unidades de kHz.

La primera prueba realizada se registró un CFO de 0 kHz, como se aprecia en la Figura 2.13. Las pruebas posteriores fueron realizadas con la introducción de determinados valores de CFO en la señal de entrada. Las Figuras 2.14 y 2.15, ilustran los resultados obtenidos del algoritmo, con valores de CFO de 100 kHz y 200 kHz, respectivamente, donde se obtuvo, en ambos casos, detecciones precisas de cada CFO.



(a) (b)

Figura 2.13. Detección de CFO con 0 kHz: (a) Autocorrelación y (b) Desplazamiento detectado
Fuente: Autores
Elaboración: Autores

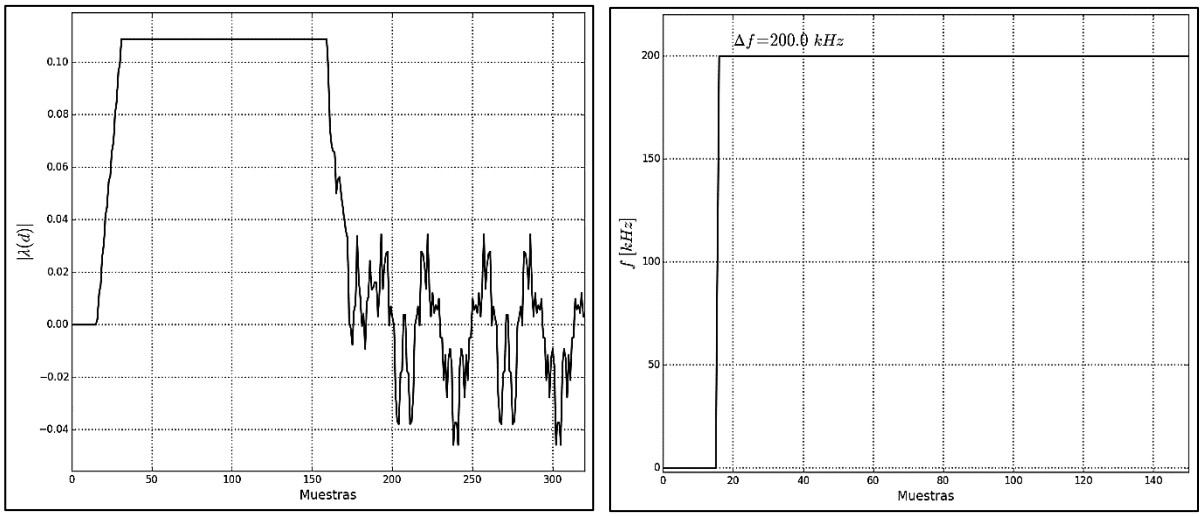


(a) (b)

Figura 2.14. Detección de CFO con 100 kHz: (a) Autocorrelación y (b) Desplazamiento detectado
Fuente: Autores
Elaboración: Autores

La variación de los valores obtenidos en la autocorrelación, respecto a los valores con un CFO de 0 kHz, se incrementó notablemente. Los puntos pertenecientes a la meseta tuvieron una amplitud mayor con valores bajos de CFO, mientras que, con valores altos, su amplitud se vió disminuida.

El desplazamiento de frecuencia obtenido e ilustrado en los literales (b) de las Figuras 2.14 y 2.15, se presenta como un valor constante a partir de cierta muestra y mantiene su magnitud hasta la última muestra de la secuencia STS. Mientras que en la Figura 2.13(b) se observa una señal de amplitud constante igual a cero, debido a que no fue detectado ningún desplazamiento de frecuencia.



(a) (b)

Figura 2.15. Detección de CFO con 200 kHz: (a) Autocorrelación y (b) Desplazamiento detectado
Fuente: Autores
Elaboración: Autores

2.3. Diseño de bloques GNURadio.

Una vez realizadas las simulaciones de los algoritmos de sincronismo en el lenguaje Python dentro del entorno Spyder, se procedió al diseño de los bloques en GNURadio basándonos en los códigos ya obtenidos en la Sección 2.2. Al existir diferentes etapas de sincronismo se diseñó un bloque propio para cada una de ellas; cada bloque de procesamiento ejecutó diferentes tareas sobre el preámbulo recibido. De esta manera, se creó un bloque con la operación de detección de paquete, un bloque para la sincronización de tiempo usando tanto la secuencia STS y otro bloque usando la secuencia LTS y finalmente un bloque para el sincronismo de frecuencia.

El preámbulo generado en Python como se explica en la Sección 2.2.1 se almacenó en un archivo de texto, con extensión .txt, el cual se usó para la creación del bloque *Source* que fue enviado repetidas veces. Los bloques creados fueron incluidos en determinados *GRCs*, que son los archivos generados en GNURadio, y de manera similar a la sección anterior en Python, se creó archivos para cada etapa de sincronismo (dos archivos para sincronismo de tiempo).

Los GRCs desarrollados compartieron ciertas similitudes en cuanto a los bloques que los componen. Cada *GRC* fue elaborado con bloques constantes como el de la variable *samp_rate* en el que se determina la tasa de muestreo utilizada que, por tratarse del estándar IEEE802.11a, se fijó en 20 MHz. Otro bloque constante es el denominado *Options* en el cual el estilo del GUI fue elegido. Estos bloques, junto al mencionado bloque *Source* donde se generó y repitió determinadas veces el vector del preámbulo, fueron agregados en todas las etapas de sincronismo.

2.3.1. Detección de paquete

El flujograma de bloques en GNURadio que se muestra en la Figura 2.16 se formó con el bloque fuente o *Vector Source*, el bloque *Detector De Trama* y el bloque de salida o *Scope Sink*. El bloque detector de trama se diseñó siguiendo el mismo proceso de la Figura 2.6, donde el tipo de datos de entrada del bloque se estableció en *Complex Float 32* de color azul y el tipo de datos en la salida en *Float 32* de color naranja (ver Figura 2.2). Su desarrollo puede ser apreciado en el Anexo C en conjunto con el código en formato *.xml*, necesario para su inserción dentro del entorno de GNURadio.

El bloque es de tipo síncrono, lo cual implica tener el mismo número de elementos de entrada que de salida; es decir, en este caso, 320 símbolos generados en el preámbulo de entrada y 320 símbolos que se producen a la salida tras pasar por las operaciones de autocorrelación, filtrado y detección.

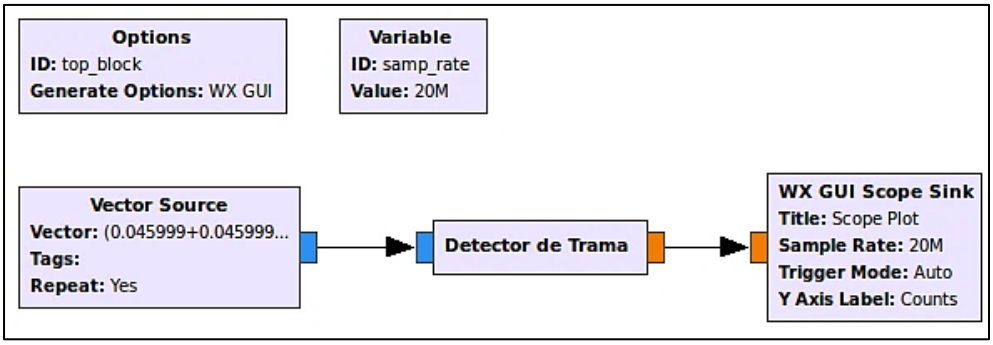


Figura 2.16. Flujograma de bloques para la Detección de paquete en GNURadio
Fuente: Autores
Elaboración: Autores

La salida del flujograma de detección es el vector *y2n_sm*, el cual representa la presencia de la trama usando un acumulador de 16 muestras. En la Figura 2.17 se observa varias repeticiones del vector de salida *y2n_sm*, el cual tiene 320 muestras que equivalen a 16 µs.

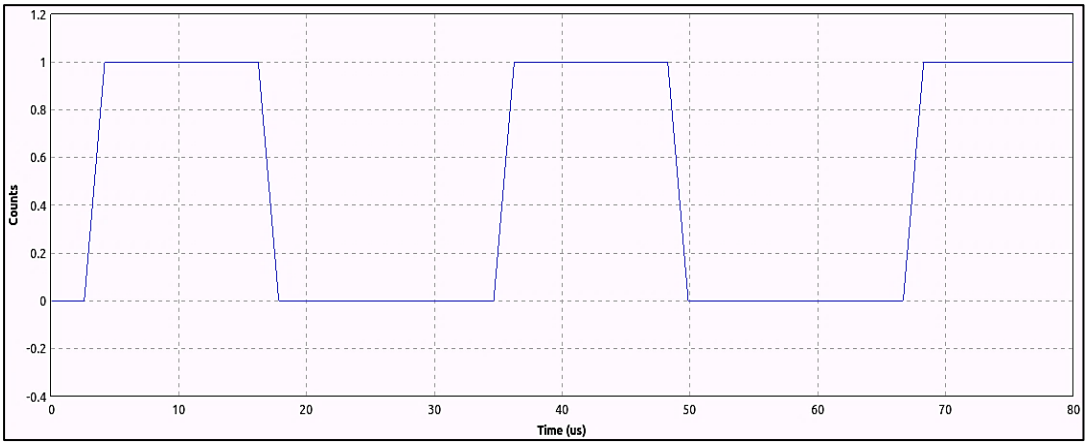
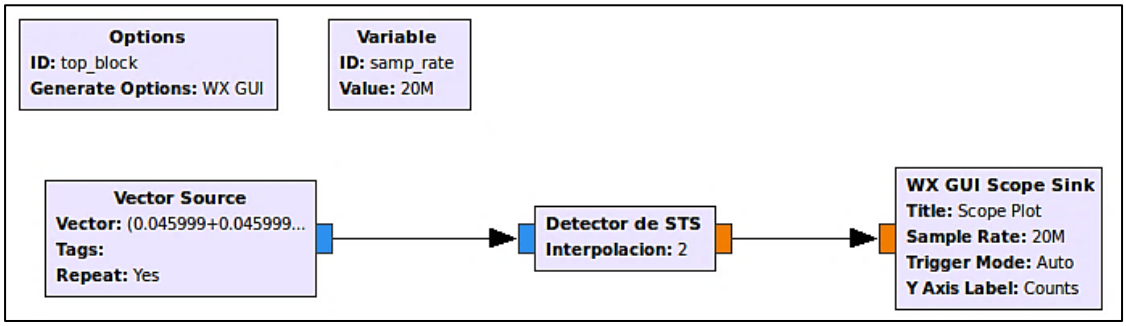


Figura 2.17. Salida del flujograma de bloques de la Detección de paquete en GNURadio.
Fuente: Autores
Elaboración: Autores

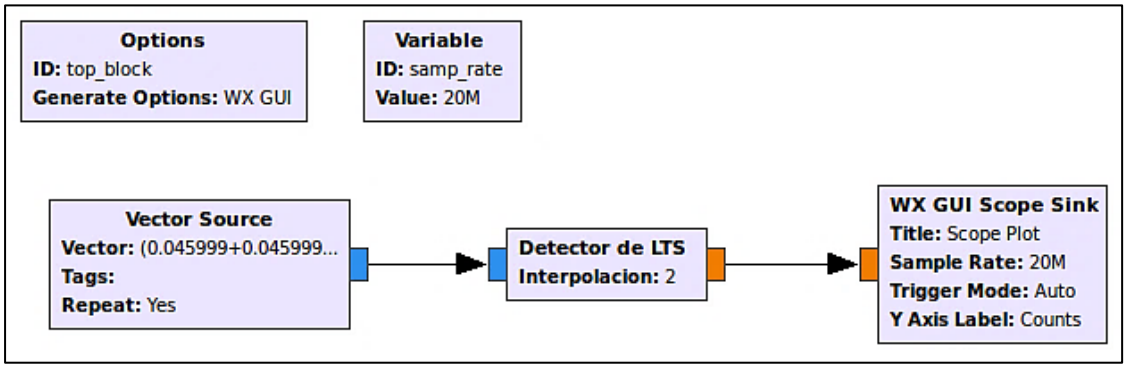
2.3.2. Sincronismo de tiempo

El procedimiento para la creación del flujograma de bloques de sincronismo de tiempo es el mismo tanto para la secuencia STS como para la secuencia LTS. Al igual que en la detección del paquete, el flujograma se diseñó con el bloque generador del preámbulo (*Source*), el bloque de procesamiento, donde el sincronismo de tiempo es realizado y el bloque de representación de la correlación cruzada en el tiempo (*Sink*). La configuración del *GRC* es presentado en la Figura 2.18, tanto para (a) STS, como para (b) LTS.

Los bloques *Detector de STS* y *Detector de LTS* fueron diseñados conforme a lo expuesto en la Figura 2.9, cada uno cuenta con una entrada de 320 valores y una salida de 640 muestras. Por esta razón, y en vista que el tamaño del vector de salida puede ser considerado como un múltiplo del tamaño del vector de entrada, se creó un bloque de tipo interpolador, con un valor de 2 para su interpolación.



(a)



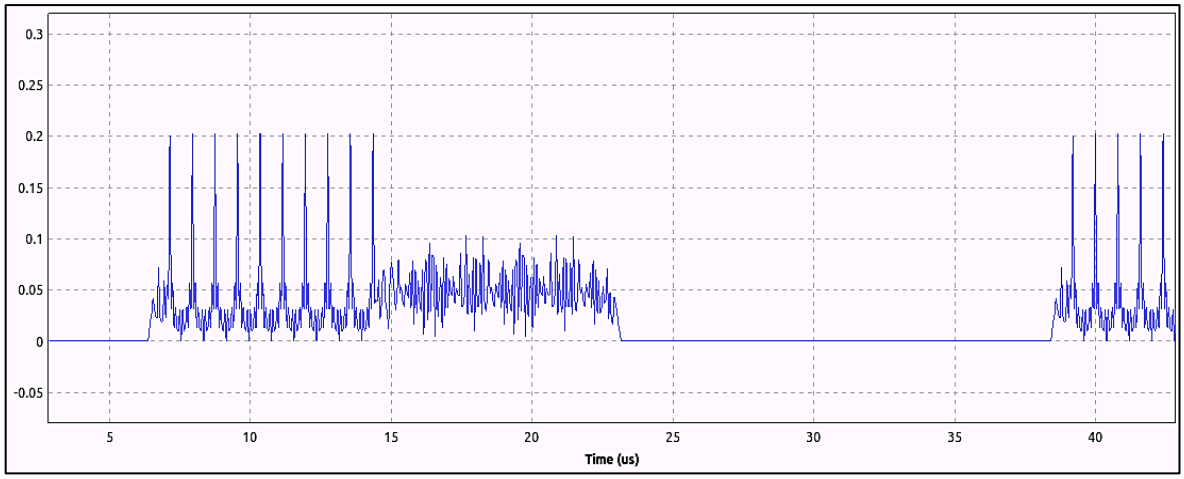
(b)

Figura 2.18. Flujograma de bloques para el Sincronismo de tiempo en GNURadio con (a) STS y (b) LTS
Fuente: Autores
Elaboración: Autores

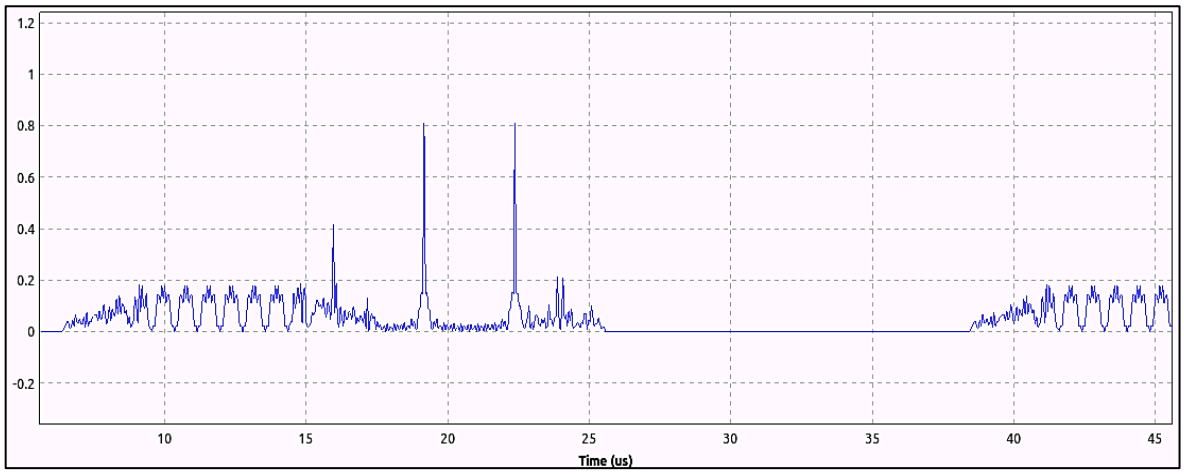
No es necesario aplicar ambos algoritmos de sincronismo simultáneamente, generalmente, se escoge un solo tipo, siendo LTS el más usado en la literatura en vista de que son solamente dos símbolos largos frente a los 10 símbolos cortos.

La salida de los bloques que simulan los algoritmos de sincronismo de tiempo se observa en la Figura 2.19 para (a) STS y (b) LTS. El tamaño del vector de salida en ambos flujogramas fue de 640 muestras, lo que representa una duración de 32 µs. En la respuesta de la secuencia

STS se obtuvieron 10 picos que representan el final de cada símbolo corto. Para la detección, se estableció la misma ventana de 16 muestras, usada en la Sección 2.2.4, aplicada a la posición del último pico (muestra 160). De la misma manera, para LTS, en la señal de salida de 640 muestras con 32 µs de duración, debe ser detectado uno de los 3 picos obtenidos; el primero marca el final del GI (muestra 32), mientras que los dos picos restantes representan el final del primer y segundo símbolo de LTS (muestras 256 y 320, respectivamente). Ante esto, en el diseño del bloque se estableció una ventana de 64 muestras de longitud alrededor del último pico para calcular el máximo valor en esa ventana (muestra 320).



(a)



(b)

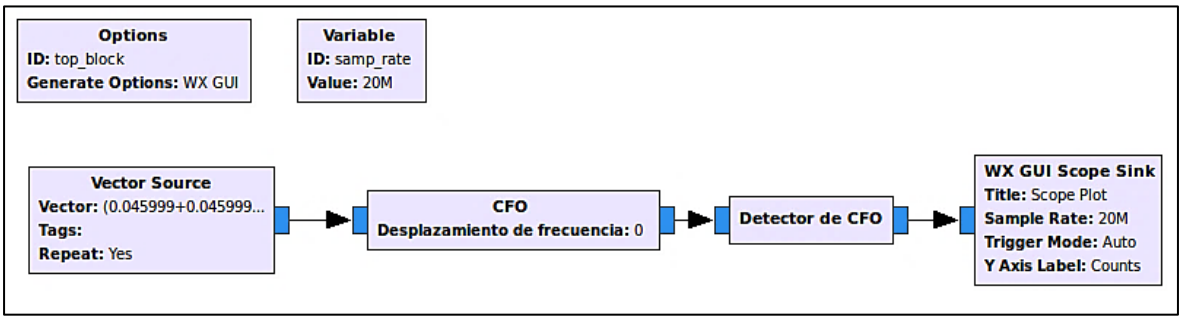
Figura 2.19. Respuesta en tiempo en GNURadio del Sincronizador de tiempo con (a) STS y (b) LTS
Fuente: Autores
Elaboración: Autores

2.3.3. Sincronismo de frecuencia

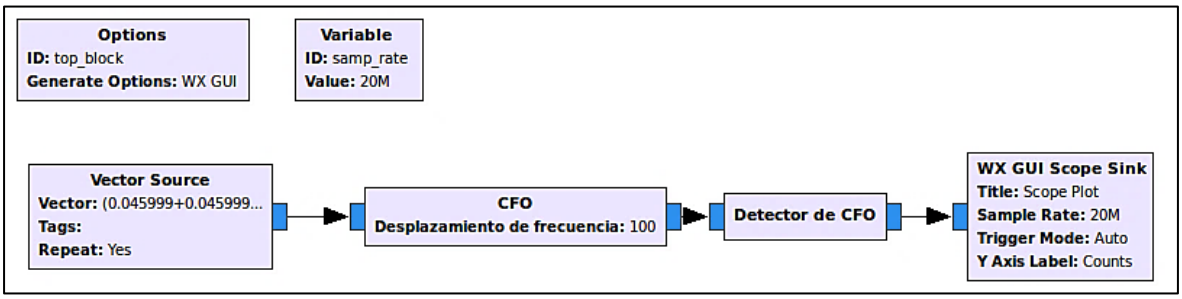
Finalmente, para la última etapa de sincronismo se crea el flujograma de bloques de la Figura 2.20 con valores de CFO iguales a (a) 0 kHz (b) 100 kHz y (c) 200 kHz. Tal como en el proceso de creación de los flujogramas anteriores, éste se diseñó con un bloque *Source*, un bloque

que añade el CFO al preámbulo generado para efectos de simulación de canal, un bloque donde éste es detectado y corregido y un bloque *Sink.*

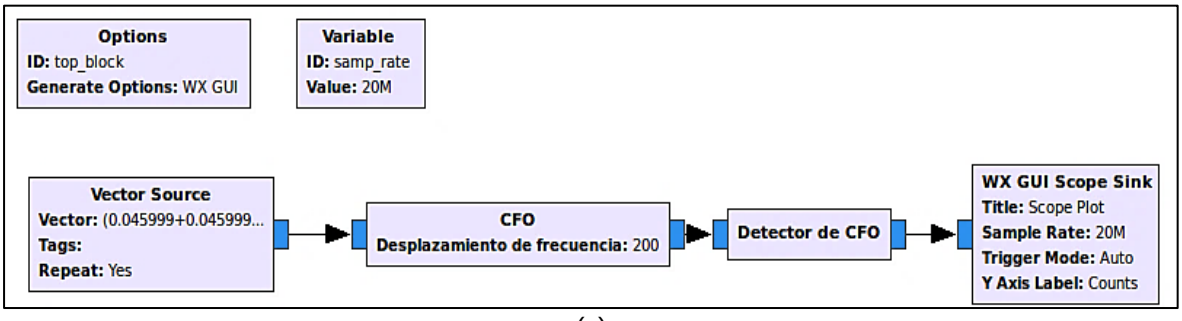
El bloque denominado *CFO* se configuró para que el valor del desplazamiento de frecuencia pueda ser establecido manualmente. Mientras que, el bloque *Detector de CFO* realiza el proceso inverso al bloque anterior, al detectar el valor de CFO introducido y compensando su error mediante el uso de la exponencial compleja como se explica en la Sección 2.2.5. y siguiendo el proceso de la Figura 2.12. Ambos bloques de tipo *sync* (síncrono) se diseñaron con entradas y salidas de tipo *Complex Float 32* y 320 muestras de longitud.



(a)



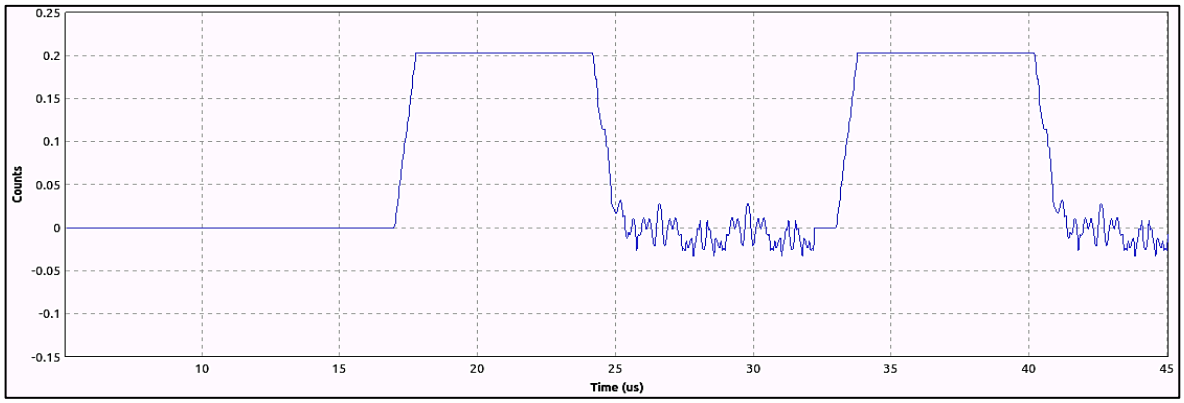
(b)



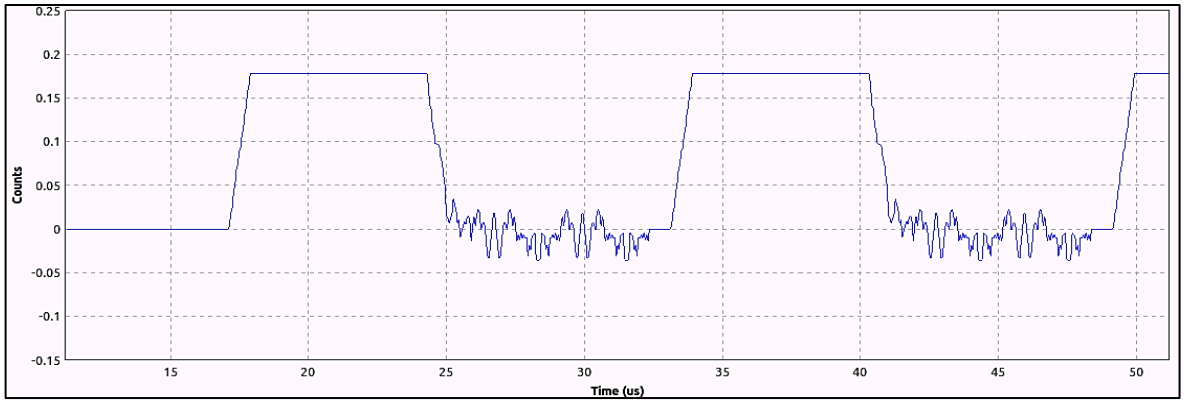
(c)

Figura 2.20. Diagrama de bloques del Sincronizador de frecuencia (a) Con 0 kHz, (b) Con 100 kHz y (c) Con 200 kHz
Fuente: Autores
Elaboración: Autores

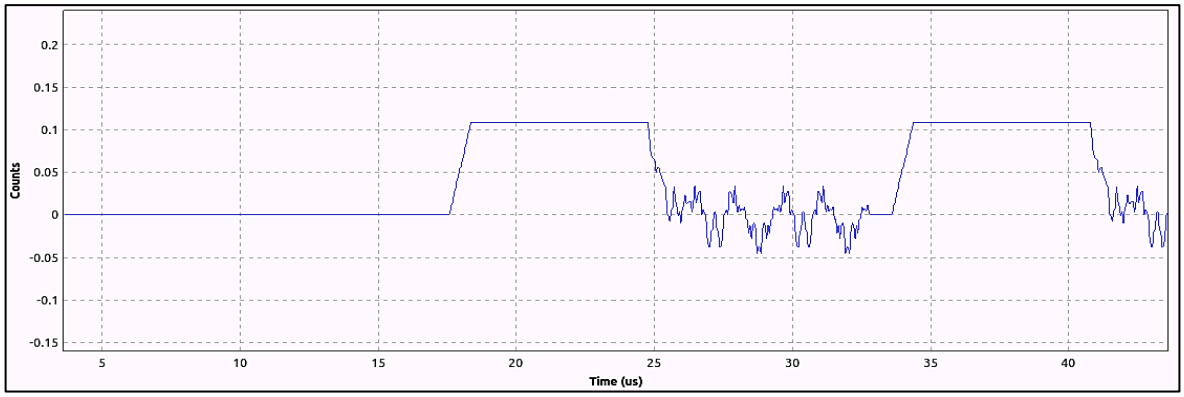
En la Figura 2.21 se observa la salida de cada flujograma de bloques para los tres CFO agregados (0, 100 y 200 kHz), esta salida se produce al correlacionar la señal de entrada con la misma retrasada 16 muestras, denominada *y2n*.



(a)



(b)



(c)

Figura 2.21. Respuesta en tiempo en GNURadio del Sincronizador de frecuencia con (a) 0 kHz, (b) 100 kHz y (c) 200 kHz
Fuente: Autores
Elaboración: Autores

CAPÍTULO III:

IMPLEMENTACIÓN CON EL USRP

En este capítulo se realizó la implementación de los flujogramas de bloques diseñados en GNURadio en el capítulo 2, utilizando los equipos USRPs. En la Sección 3.1 se especifican las características del USRP y de las antenas usadas, mientras que en la Sección 3.2 se muestran los *GRC* creados para las pruebas de los algoritmos de sincronización y las señales de salida de cada uno de ellos registradas en el receptor.

3.1. USRP

Universal Software Radio Peripheral (USRP) es un equipo comercializado por *Ettus Research,* que permite el desarrollo de SDR con las herramientas GNURadio, LabVIEW y Simulink, además es una plataforma muy flexible y puede ser utilizada para implementar aplicaciones en tiempo real [31].

El USRP está formado por una placa base o placa madre (*motherboard*), una placa hija (*daugtherboard*) o placas secundarias, que pueden transmitir y/o recibir señales en diferentes rangos de frecuencias (0 a 5.9 GHz), cada una de las ranuras se encuentra etiquetada como TXA, RXA, TXB, RXB. La placa base incluye una FPGA, una memoria incorporada (*Static Random Access Memory* - SRAM), una memoria extraíble (tarjeta SD), los conversores ADC/DAC, la alimentación y la conexión vía Ethernet. Además, la placa madre ejerce de intermediario en el tratamiento de la señal entre el host (PC) y la placa hija. Sus funciones principales son: la conversión de la señal del dominio analógico de frecuencia intermedia (*Intermediate Frequency* - IF) al dominio digital y viceversa, y el ajuste de la tasa binaria del sistema. Las placas secundarias pueden intercambiarse fácilmente, lo que permite al USRP trabajar en varias frecuencias. La *daugtherboard* también es la encargada de trasladar de IF (banda de base) a RF (radiofrecuencia), y viceversa, la señal de interés (ver Figura 3.1)

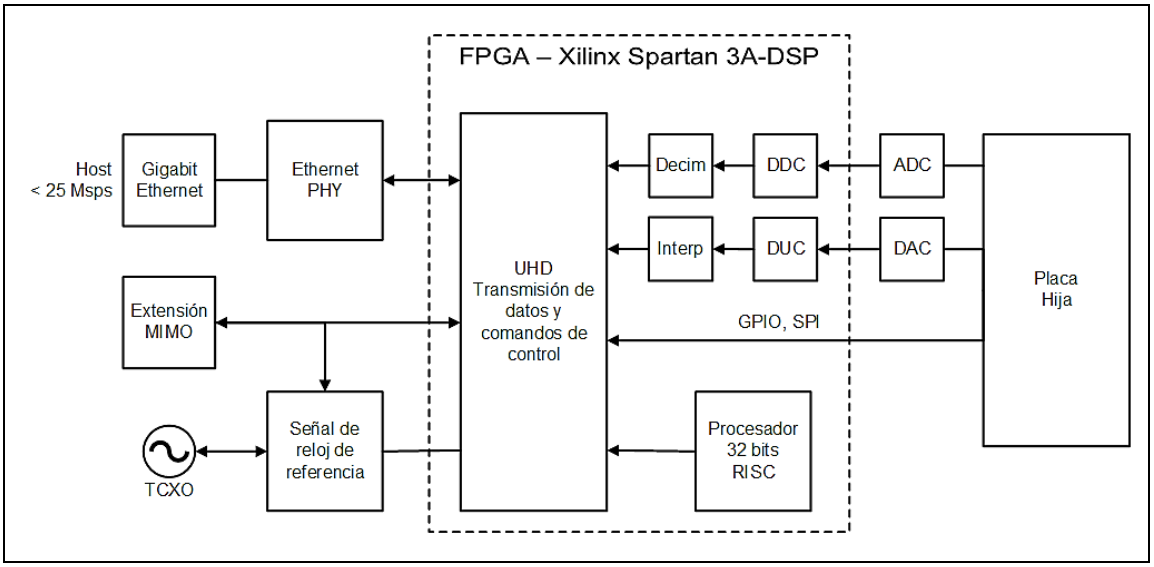


Figura 3.1. Arquitectura del USRP N210
Fuente: [10], [32]
Elaboración: Autores

La primera vez que se utiliza el dispositivo USRP, el firmware de USRP y la imagen de FPGA deben descargarse en la memoria. Luego, el dispositivo puede corresponder con cualquier controlador de hardware USRP (*USRP Hardware Driver* - UHD) instalado por computadora.

En este documento, utilizamos los dispositivos Ettus USRP N210. La placa RF utilizada en nuestro banco de pruebas es la placa secundaria de banda ancha WBX, que opera dentro de un amplio rango frecuencias y trabaja bajo el rango típico de potencia de transmisión (ver Tabla 3.1 donde se especifican las características de un USRP N210 [31]. Además, mediante el puerto de expansión, es posible implementar una configuración 2x2 MIMO (*Multiple Inputs Multiple Outputs*). La comunicación entre el USRP y el PC se realiza a través del protocolo *Gigabit Ethernet*.

Tabla 3.1. Parámetros característicos del USRP N210

Característica	Unidad	Valor
Interfaz Gigabit Ethernet	Mbps	25
FPGA	-	Spartan 3A-DSP 3400
SRAM	MByte	1
Rango de frecuencias	MHz	50 – 2200
Frecuencia de muestreo de 2 ADCs	MS/s	100
Resolución ADC	Bits	14
Frecuencia de muestreo de 2 DACs	MS/s	400
Resolución DAC	Bits	16 bits
Ancho de banda de RF instantáneo	MHz	25 / 16 bits
Consumo de energía [Voltaje]	V	6
Consumo de energía [Amperaje]	mA	1.3
Potencia de salida	dBm	15 - 20
Receptor de la figura de ruido	dB	5
Dimensiones (l x w x h)	cm	22 x 16 x 5
Peso	kg	1.2

Fuente: [31]
Elaboración: Autores

En el panel frontal del dispositivo (Figura 3.2) se encuentran: 6 LEDS que informan sobre el estado del dispositivo, 4 conectores SMA (RF1 y RF2) para los dipolos (Tx/Rx), el puerto de expansión MIMO, el puerto Gigabit Ethernet y el conector de la fuente de alimentación.

En la Tabla 3.2 se detallan algunas de las características de las antenas WSS016 (Figura 3.3) utilizadas en el presente trabajo.

Figura 3.2. Panel frontal del USRP N210
Fuente: Autores
Elaboración: Autores

Tabla 3.2. Parámetros característicos de la antena WSS016

Característica	Valor
Modelo	WSS016
Tipo de antena	Dipolo omnidireccional
Polarización	Vertical (lineal)
Tipo de radiación	Toroidal
Color	Negro
Perfil de la antena	236.5 mm
Frecuencia de operación	824 – 960 – 1710 – 1990 MHz
Impedancia	50 Ohm nominal
Conector	SMA (M)
Ganancia	3 dBi

Fuente: [33]
Elaboración: Autores

Figura 3.3. Perfil de antena WSS016
Fuente: [33]
Elaboración: Autores

3.2. Pruebas realizadas con el USRP N210

El sistema implementado para realizar las pruebas de los algoritmos de sincronismo mediante la utilización de los USRP se resume en el diagrama esquemático ilustrado en la Figura 3.4. El transmisor estuvo compuesto por una computadora portátil LENOVO con sistema operativo Ubuntu 10.4 y puerto de red compatible con Gigabit Ethernet, un módulo USRP N210 con una placa hija WBX; y una antena omnidireccional WSS016. El receptor, de forma similar, se formó con los mismos dispositivos que el transmisor, como se observa en la Figura 3.5.

Cada uno de los dispositivos USRP N210 tiene una dirección IP (*Internet Protocol*) predeterminada por defecto. Durante las pruebas realizadas en este proyecto, la direcciones IP del USRP fueron: 192.168.10.2/24 en el transmisor y 192.168.10.3/24 en el receptor.

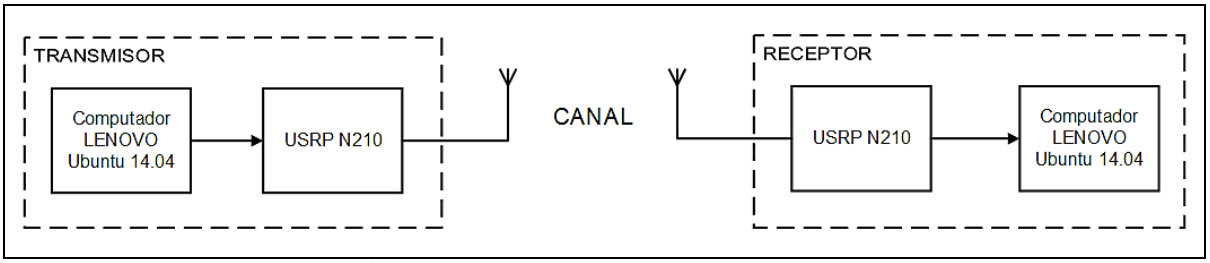


Figura 3.4. Diagrama esquemático del sistema empleado en las pruebas de los algoritmos con los USRPs
Fuente: Autores
Elaboración: Autores

Figura 3.5. Conexión del sistema para las pruebas de los algoritmos con USRPs
Fuente: Autores
Elaboración: Autores

3.2.1. Transmisor

Los algoritmos de sincronismo implementados se aplicaron a las diferentes secciones del preámbulo del estándar IEEE802.11a, por lo que solamente fue necesaria la transmisión de dicho preámbulo, dejando de lado la información relevante que lo proceda. El preámbulo, como fue explicado en la Sección 1.2.1 y construido en la Sección 2.2.1, es transmitido de forma repetitiva de un USRP a otro, dando lugar a la aplicación de los algoritmos de sincronismo en cada una de las repeticiones de la transmisión.

El transmisor fue elaborado bajo las especificaciones ya mencionadas en la Sección 2.3 y su diagrama se ilustra en la Figura 3.6, donde un bloque fuente (Source) generó el vector preámbulo, un bloque preinstalado de GNURadio llamado *UHD: USRP Sink* realizó el enlace de comunicación con el USRP, y un bloque de interfaz *WX GUI Scope Sink* permitió graficar la respuesta temporal de la señal del vector de salida hacia el USRP. El bloque *UHD: USRP Sink* estableció la conexión con el dispositivo al ingresar su dirección IP (TX: 192.168.10.2). Además, se fijó la frecuencia de transmisión en 900 MHz y el valor de la ganancia del transmisor (30 dB).

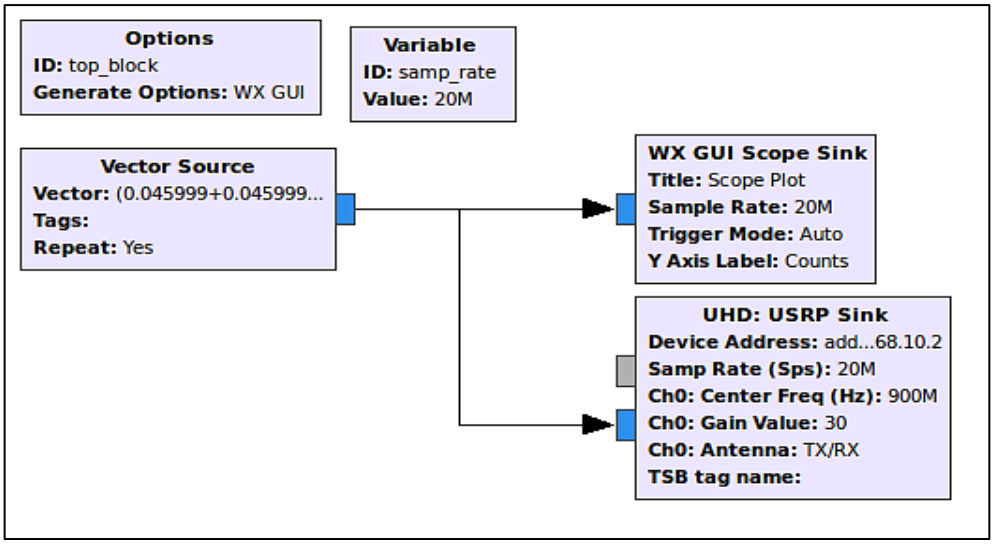


Figura 3.6. Flujograma de bloques del transmisor implementado con el USRP
Fuente: Autores
Elaboración: Autores

3.2.2. Receptor

De manera similar a lo realizado en el transmisor, el receptor se creó con tres bloques con tres distintas funciones. El primero fue *UHD: USRP Source* en donde los datos fueron recibidos por el USRP y pasaron a formar el vector de entrada que fue procesado en los siguientes bloques. En este bloque se establece la dirección IP del USRP receptor (192.168.10.3), la frecuencia (900 MHz) y la ganancia (30 dB) en la que está receptando los datos. El siguiente bloque utilizado que se observa en la Figura 3.7 es *WX GUI Scope Sink* también usado en el transmisor y que hizo uso del bloque de variable *samp_rate* con el valor de frecuencia de muestreo (20 MHz).

En los archivos *GRC* se incluyeron posteriormente cada uno de los bloques que realizaban el proceso de sincronismo: detección de paquete, sincronismo de tiempo y sincronismo de frecuencia.

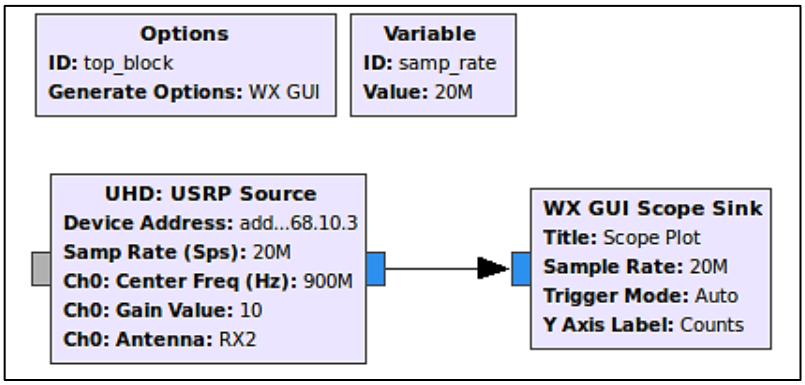


Figura 3.7. Flujograma de bloques del receptor implementado con el USRP
Fuente: Autores
Elaboración: Autores

3.2.2.1. Detección de paquete

El preámbulo enviado a través del transmisor fue receptado y procesado por el mismo bloque Detector de Trama ya utilizado en la Sección 2.3.1, como se observa en la Figura 3.8. La respuesta temporal de la señal de salida se ilustra en la Figura 3.9, en donde se observa la presencia de escalones cuando la detección del paquete ha sido efectiva en cada repetición del preámbulo.

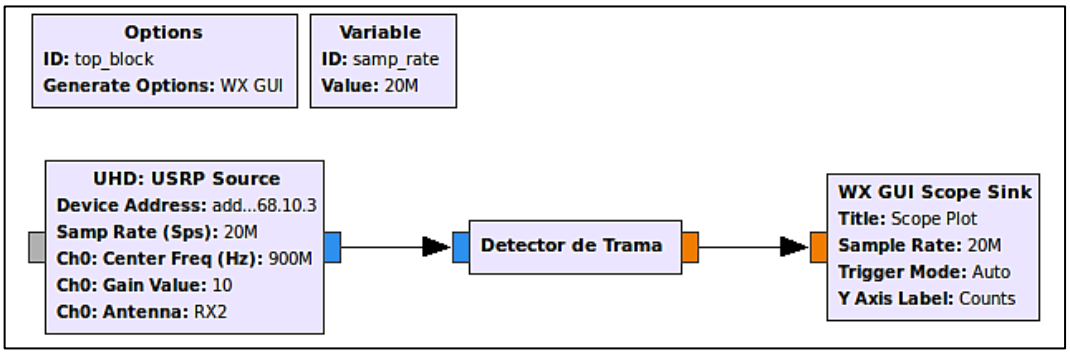


Figura 3.8. Flujograma de bloques del algoritmo de Detección de paquete, implementado con los USRPs.
Fuente: Autores
Elaboración: Autores

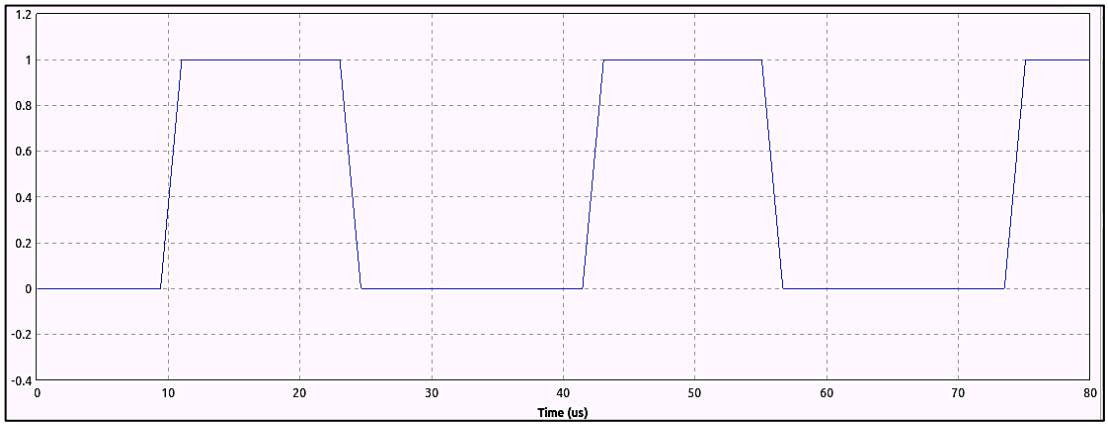


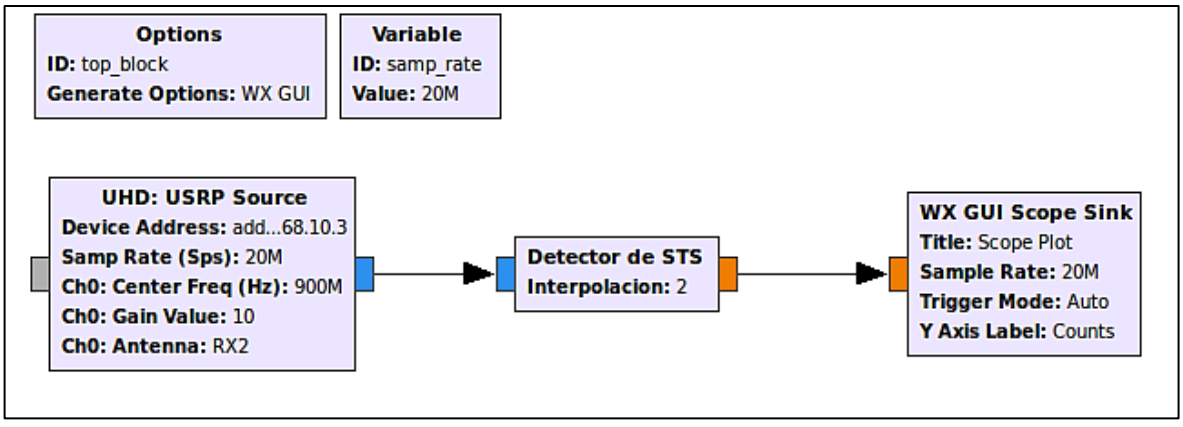
Figura 3.9. Salida del flujograma de bloques del Detector de trama.
Fuente: Autores
Elaboración: Autores

Lo representado en la Figura 3.9 guarda una alta relación con lo expuesto anteriormente en la Figura 2.8(b) y la Figura 2.17, demostrando que no existe diferencia entre los resultados obtenidos de las simulaciones y aquellos obtenidos en la implementación, a pesar de los efectos reales del canal inalámbrico utilizado.

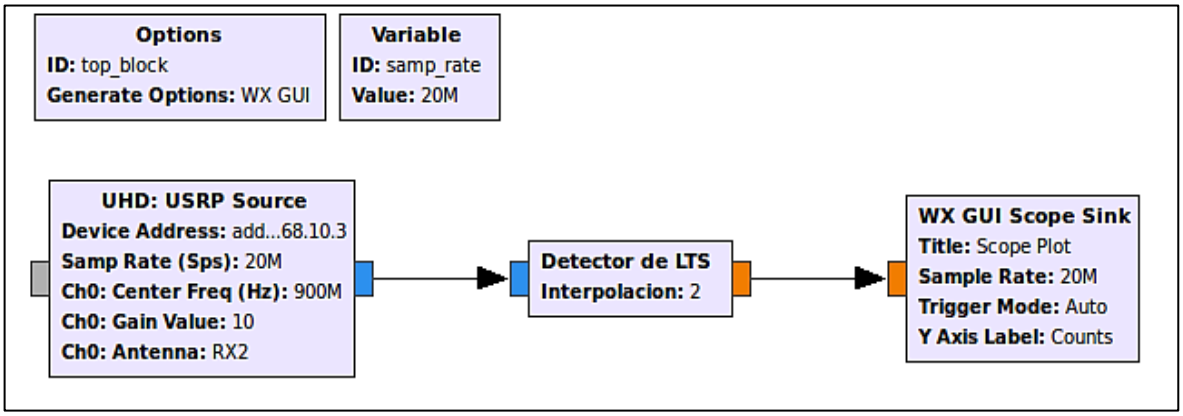
3.2.2.2. Sincronismo de tiempo

En el receptor implementado al inicio de ésta sección, se agregó los bloques *Detector de STS* y *Detector de LTS* desarrollados y aplicados en la Sección 2.3.2. El preámbulo transmitido, al pasar por los bloques detectores, fue procesado y la posición detectada del último símbolo es almacenada dentro de un archivo de texto por cada una de las repeticiones transmitidas

La Figuras 3.10 ilustra la configuración de los algoritmos de sincronización de tiempo en GNURadio, implementados en USRP, donde el preámbulo recibido pasó por el bloque *Detector de STS* y *Detector de* LTS, detectando la posición o muestra en la que finalizó el segundo símbolo corto y largo, respectivamente.



(a)



(b)

Figura 3.10. Flujograma de bloques de los algoritmos de Sincronismo de tiempo, implementados con USRPs con (a) STS y (b) LTS
Fuente: Autores
Elaboración: Autores

Al igual que lo realizado en la Detección de Paquete, se cambió el bloque generador del preámbulo por el bloque de recepción del USRP; en la Figura 3.11 se muestra la respuesta en el tiempo de la señal que se obtuvo luego de la ejecución de los algoritmos de correlación cruzada.

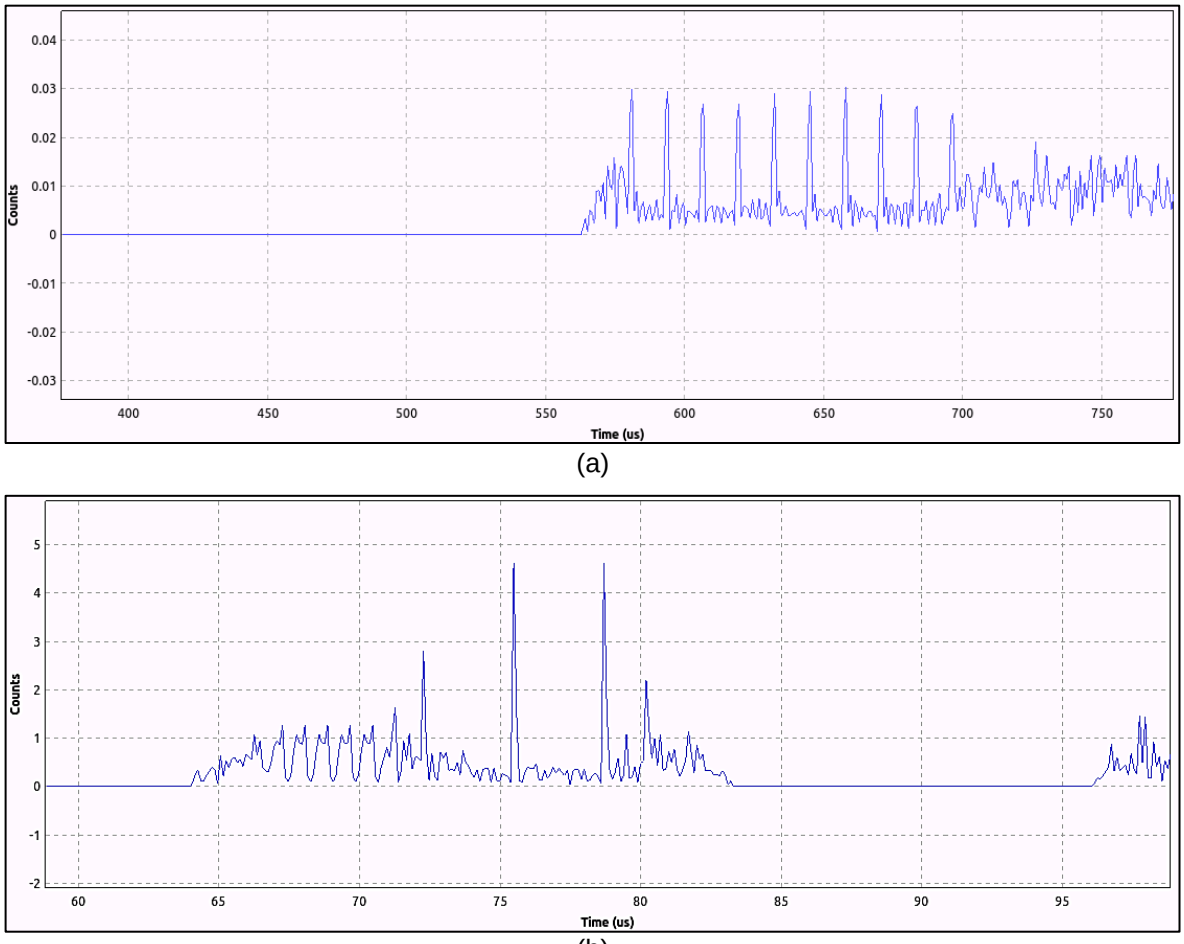


Figura 3.11. Respuesta temporal de los flujogramas de Sincronismo de tiempo con (a) STS y (b) LTS
Fuente: Autores
Elaboración: Autores

Como se aprecia en estas figuras, los efectos del canal inalámbrico fueron más evidentes. Los picos de los símbolos cortos y de los largos fueron detectables en casi todas las repeticiones, sin embargo, se dio el caso de la detección de falsos picos a razón de altas amplitudes de los picos espurias en el receptor.

3.2.2.3. Sincronismo de frecuencia

La última etapa de sincronismo consistió en detectar el desplazamiento de frecuencia que el canal introdujo al transmitir el preámbulo. En la Figura 3.12 se muestra el flujograma de bloques que permitió detectar este desplazamiento. De manera similar a las secciones anteriores, el flujograma se formó con tres bloques, el *UHD: USRP Source,* el bloque *Detector de CFO* y el bloque *WX GUI Scope Sink.* Este último se usó para obtener la respuesta

temporal de la autocorrelación efectuada para el cálculo del CFO (Sección 1.5.3) como salida del bloque *Detector de CFO* (Figura 3.13).

El valor máximo de la meseta de esta señal fue empleado en el cálculo del desplazamiento de fase y, por consiguiente, también el desplazamiento de frecuencia, añadido en cada repetición del preámbulo. El valor obtenido en la detección de CFO, fue enviado a un archivo de texto, el cual era actualizado en cada repetición del preámbulo. Los datos obtenidos en este punto sirvieron para obtener la varianza, expuesta en el siguiente capítulo.

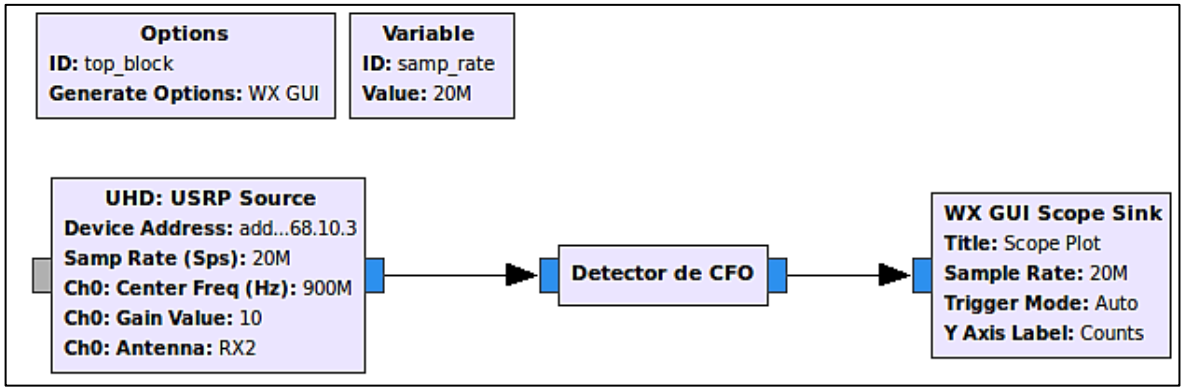


Figura 3.12. Flujograma de bloques del algoritmo de sincronismo de frecuencia, implementado con el USRP
Fuente: Autores
Elaboración: Autores

En la Figura 3.13 se ilustra la respuesta en el tiempo de la autocorrelación cuando los dos dispositivos USRP se colocaron a una distancia de 30 centímetros (Figura 3.5), lo que equivaldría a un ambiente óptimo según lo expuesto en la Sección 1.3; mientras que en la Figura 3.14 se presenta la respuesta obtenida cuando los dispositivos se ubicaron a 6 metros de distancia (ambiente moderado), uno respecto al otro (Figura 3.15).

Al comparar las señales de salida del algoritmo simulado dentro de Spyder (Figuras 2.13, 2.14 y 2.15) y dentro de GNURadio (Figura 2.21), con la señal obtenida en el último caso al implementar el algoritmo con los USRPs en las Figuras 3.13 y 3.14, se puede apreciar grandes diferencias a razón de que las simulaciones son entornos controlados, mientras que las implementaciones con los equipos no pueden ser controladas en relación a los efectos que agrega el canal, mismos que ocasionaron variaciones en la magnitud y duración de la meseta respecto a los resultados obtenidos en las simulaciones; representando una detección del desplazamiento de frecuencia diferente en cada una de las repeticiones. Esta diferencia se volvió más notable dentro de la implementación del algoritmo con una separación de 6 metros entre transmisor y receptor.

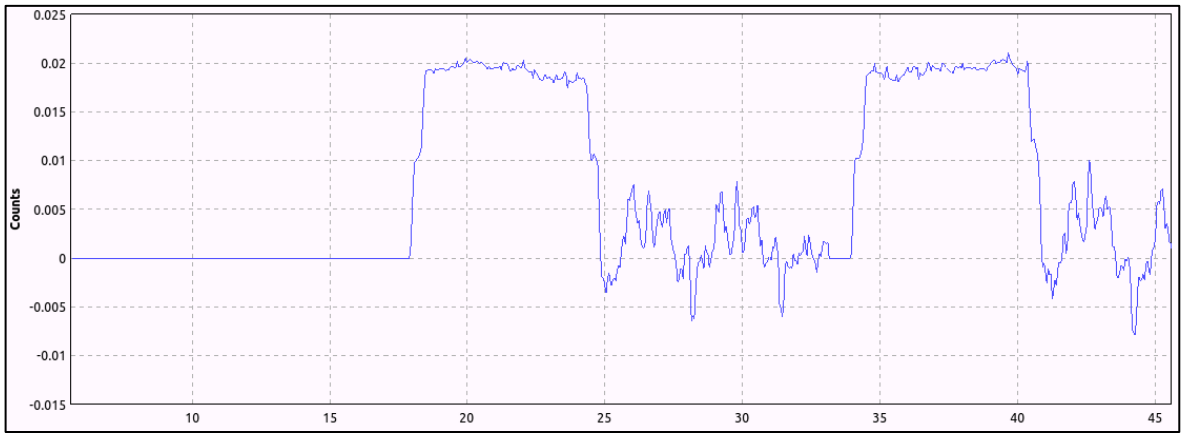


Figura 3.13. Respuesta temporal de la autocorrelación usada para la detección de CFO, implementada con los USRPs a 30 centímetros de separación.
Fuente: Autores
Elaboración: Autores

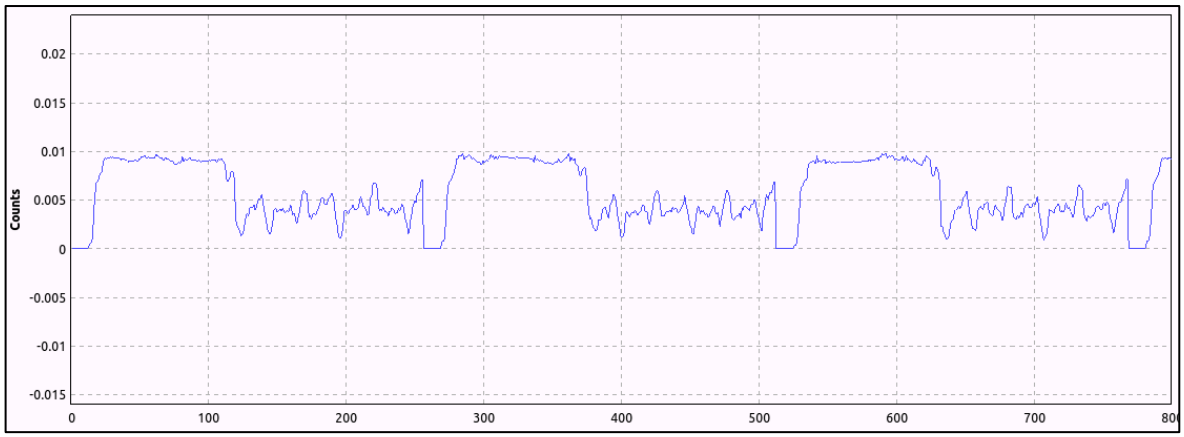


Figura 3.14. Respuesta temporal de la autocorrelación usada para la detección de CFO, implementada con los USRPs a 6 metros de separación.
Fuente: Autores
Elaboración: Autores

Una de las diferencias más marcadas entre los dos casos, es el comportamiento del ruido y su efecto en la señal de autocorrelación, haciéndose más evidente en el segundo caso. La amplitud de las señales demuestra este efecto, siendo un valor de 0.02 para la amplitud de la meseta de la autocorrelación en el primer caso y una reducción a la mitad (0.01) en la amplitud de la meseta del segundo.

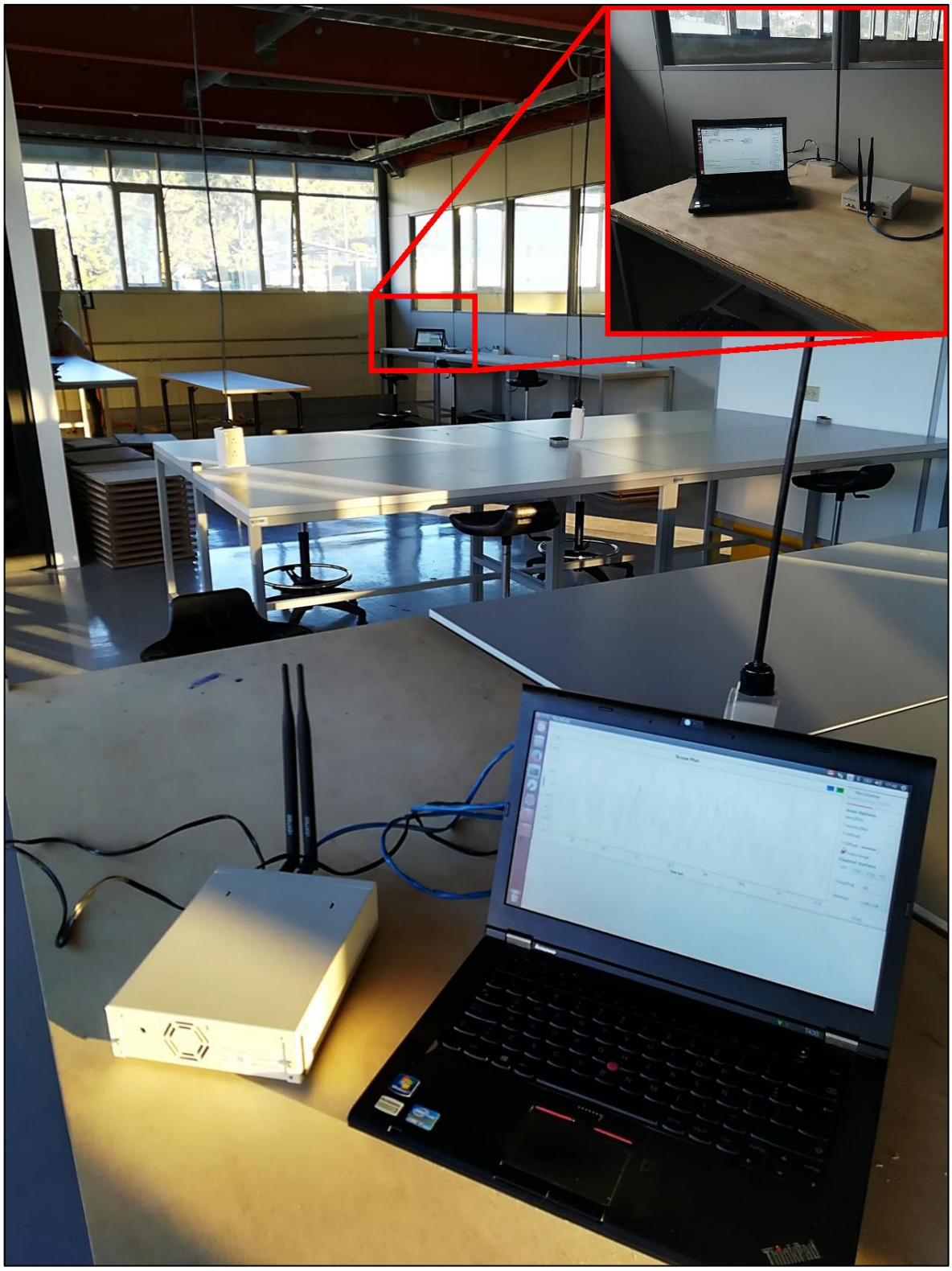


Figura 3.15. Colocación de dispositivos con una separación de 6 metros entre los USRPs.
Fuente: Autores
Elaboración: Autores

CAPÍTULO IV:

ANÁLISIS DE RESULTADOS

En este capítulo se analiza los resultados obtenidos al realizar la implementación de cada uno de los algoritmos de sincronismo haciendo uso de los USRPs. La respuesta del algoritmo de detección de paquete fue fácil de interpretar dado que su objetivo consistió en confirmar la presencia de datos en el receptor. Casos totalmente diferentes se dieron en la implementación de los algoritmos de sincronismo de tiempo y de frecuencia. La precisión de estos algoritmos fue comprobada por medio de una medida de dispersión denominada varianza, que consistió en determinar la variación de los resultados obtenidos de los algoritmos en cada repetición enviada por el USRP, respecto a la media del conjunto de todos los resultados de dichas repeticiones.

La varianza de una serie de muestras exhibe el desempeño del algoritmo para soportar errores en la recepción provocados por el canal. Dado un determinado número de muestras como resultado de las repeticiones de los algoritmos, la varianza pudo ser determinada por medio de Ecuación 4.1, la cual, a su vez, hace uso de la media del conjunto total de datos que serán analizados.

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{N} \tag{4.1}$$

Donde

σ^2 = Varianza del conjunto de muestras

$i = 0, \dots, n \rightarrow n$ es el número de muestras del conjunto

x_i = Dato o muestra en la posición $i = 0, \dots, N-1$

$\bar{x}$ = Media del conjunto de muestras

N = Tamaño del conjunto de muestras

El cálculo de la varianza para los sincronismos de tiempo y de frecuencia se realizó en Spyder con lenguaje de programación Python, usando el comando *variance(data, xbar=None).* El argumento *data* es la muestra de datos y *xbar* es la media del conjunto de datos, la cual es calculada automáticamente si no es proporcionada por el usuario.

Posteriormente, se calculó el valor de la desviación estándar a través de la Ecuación 4.2 y en base a los resultados anteriormente obtenidos de la varianza. Con este resultado, se evidenció la relación de la estimación respecto a los valores esperados en cada uno de los casos analizados.

$$\sigma = \sqrt{\sigma^2} \tag{4.2}$$

Donde

σ = Desviación Estándar del conjunto de muestras

σ^2 = Varianza del conjunto de muestras

Cada vez que se ejecutó los *GRC* implementados en el receptor de la Sección 3.2.2 las posiciones detectadas de STS y LTS, además del valor de CFO, se almacenaron en archivos de texto individuales con los datos en formato binario. Estos archivos fueron cargados en Python por medio del comando *numpy.loadtxt*, convirtiendo los datos binarios en datos de tipo *int16* para las posiciones STS y LTS y *float32* en el caso de los valores de CFO, acción por la cual el cálculo de la varianza pudo ser realizada con el tipo de datos correcto.

4.1. Detección de paquete

El primer paso de la sincronización OFDM es la detección del paquete. La estructura del algoritmo desarrollado para cumplir esta función fue detallada en los capítulos anteriores.

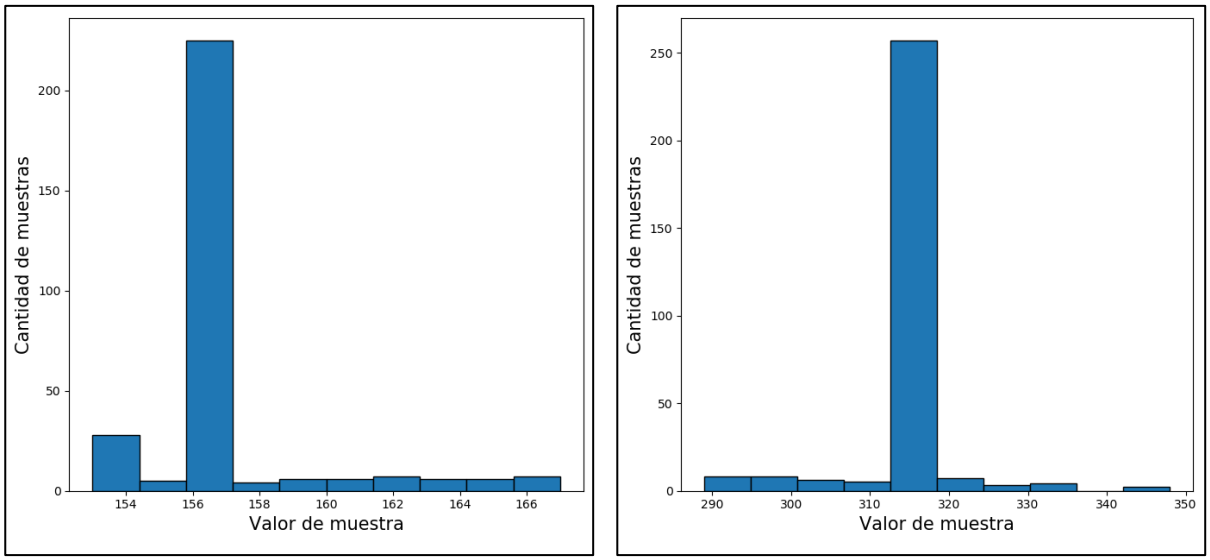
Se puede observar que no existen diferencias entre la simulación en Python (Figura 2.8 (b)) y la simulación en GNURadio (Figura 2.17). De la misma manera al agregar el ruido implementando el algoritmo con el equipo USRP, la señal de salida (Figura 3.9) se mantiene. En las tres figuras mencionadas se observa que el preámbulo transmitido fue detectado en el receptor en cada una de las repeticiones realizadas por el transmisor, siendo una señal activa cuando al menos dos símbolos cortos sucesivos de la secuencia STS son detectados.

4.2. Sincronismo de tiempo

El resultado de la implementación del algoritmo de sincronismo de tiempo con el USRP es la estimación de la posición o muestra en la que finaliza el último símbolo de la STS o el segundo símbolo de la LTS. Según el estándar IEEE802.11a dichas posiciones, sin agregar ruido a la transmisión, deben situarse en las muestras 160 para STS y 320 para LTS; no obstante, al implementarlo en el USRP se agrega ruido por parte del canal provocando un desplazamiento de la posición en cada repetición.

El conocer la posición afectada por el desplazamiento permitió deducir la magnitud de éste y, así, determinar el tiempo exacto en el que termina el preámbulo y empieza la información relevante. El preámbulo se envió repetidas veces y al ser receptado se procesó en los bloques *Detector STS* y *Detector LTS*, en estos bloques se especificó que los valores de las posiciones detectadas sean almacenados en un archivo de texto en cada repetición. En la Figura 4.1 se muestra el histograma de las posiciones obtenidas en el receptor al aplicar el algoritmo de la correlación cruzada con (a) STS y con (b) LTS, luego de transmitir el preámbulo 300 veces.

Los datos del archivo de extensión .txt fueron convertidos a valores de tipo float y finalmente se calculó la varianza de la estimación de todas las posiciones tomadas, resumiendo los resultados en la Tabla 4.1.



(a) (b)

Figura 4.1. Histograma de la posición obtenida al aplicar el algoritmo de correlación cruzada con (a) STS y (b) LTS, al implementarlo con el USRP
Fuente: Autores
Elaboración: Autores

La magnitud de la varianza determina cuán alejada se encuentra la estimación respecto a la muestra esperada. Es decir, una varianza de 6.59, descrita en la Tabla 4.1, significa que la estimación está a 2.567 muestras por encima o por debajo de 160, que es el valor esperado.

Tabla 4.1. Valores de varianza de la posición obtenida al aplicar los algoritmos de sincronismo de tiempo

Algoritmo	Repeticiones	Varianza [σ^2]	Desviación Estándar [$\sqrt{\sigma^2}$]
Correlación cruzada con STS	300	6.589	2.567
Correlación cruzada con LTS	300	45.04	6.711

Fuente: Autores
Elaboración: Autores

Valores de varianza relacionados a estos algoritmos de correlación cruzada fueron encontrados en [5]. En dicho trabajo, el autor realiza determinadas simulaciones implementadas con el hardware FPGA Spartan-3 de Xilinx. La varianza obtenida bajo las mismas condiciones, es decir, con SNR de 10 dB, para STS fue de 2.60 y de 7.28 para LTS, a consecuencia de tratarse de simulaciones controladas, además de reconocer las mejores prestaciones que posee el equipo utilizado al ejecutar dichas pruebas de forma paralela dentro del software de Xilinx, en contraste con el tipo de ejecución que posee GNURadio que es de tipo lineal a través de flujogramas.

4.3. Sincronismo de frecuencia

Al implementar el algoritmo de sincronismo de frecuencia en el USRP se determinó el desplazamiento de la frecuencia que se agregó en cada repetición del preámbulo transmitido.

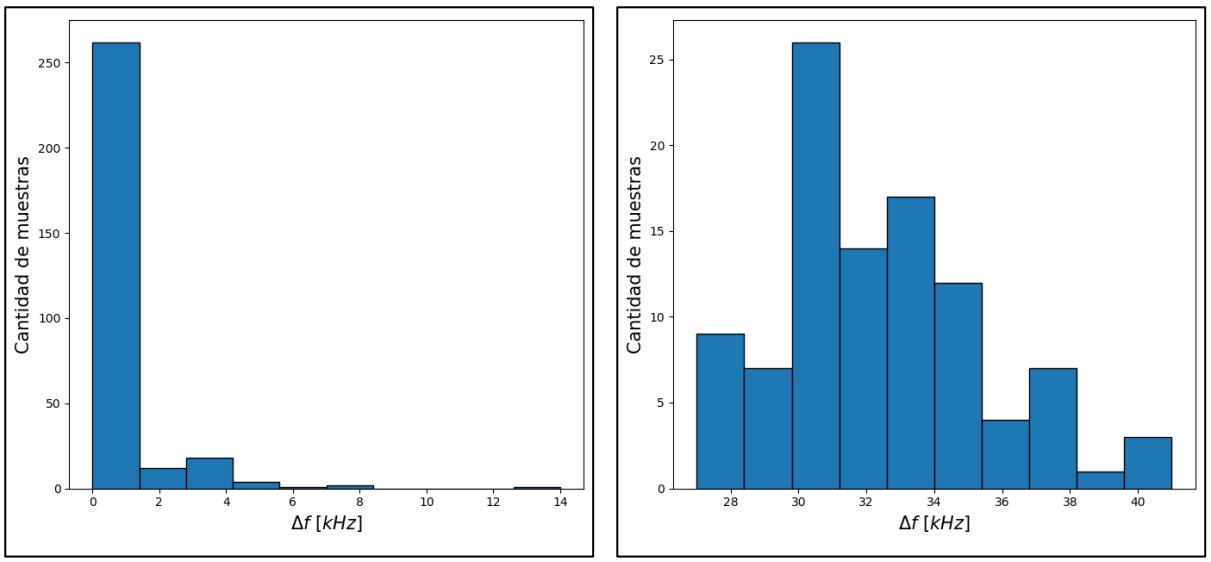
Este desplazamiento registrado se almacenó en un archivo de texto cuyos valores fueron convertidos a tipo *float* mediante Python y la varianza del conjunto fue calculada.

Se realizó dos pruebas del algoritmo, el primero fue con los USRP separados 30 cm (Figura 3.5) y la segunda con los USRP separados 6 m y con objetos entre ellos que actuaron como interferencias (Figura 3.15). En la Figura 4.2 se muestran los histogramas de las estimaciones del desplazamiento de frecuencia que se obtuvieron al transmitir el preámbulo 300 veces.

Tabla 4.2. Varianza del desplazamiento de frecuencia obtenida al aplicar el algoritmo de sincronismo de frecuencia

Algoritmo	Distancia [m]	Repeticiones	Varianza [σ^2]	Desviación Estándar [$\sqrt{\sigma^2}$]
Detector CFO	0.30	300	2.200	1.483
Detector CFO	6.00	300	9.161	3.027

Fuente: Autores
Elaboración: Autores



(a) (b)

Figura 4.2 Histograma del desplazamiento de frecuencia, como resultado de la implementación con USRP a una separación de (a) 30 cm y (b) 6 m
Fuente: Autores
Elaboración: Autores

Los valores de la varianza de estas dos pruebas que se resumen en la Tabla 4.2. representan qué tan alejados estuvieron estos resultados respecto al desplazamiento de frecuencia esperado. En el caso en el que los USRP estuvieron separados 30 cm, equivalente a un ambiente óptimo según lo explicado en la Sección 1.3, el valor esperado de desplazamiento de frecuencia fue 0 kHz, y con una varianza obtenida de 2.20 se tradujo en que los valores de desplazamiento se distanciaron en un estimado de $\sqrt{2.20}$ kHz del valor esperado. De igual manera en un ambiente moderado, el valor teórico de desplazamiento de frecuencia es 100 kHz; sin embargo, la media de los valores resultantes de la implementación fue 32, cuando los USRPs estuvieron separados 6 m. La varianza de 9.16, obtenida en este caso, representa

que los valores de CFO no guardaron una tendencia hacia los 100 kHz como se esperaba, envés de ello, se registró una tendencia marcada hacia los 30 kHz, debido a que las interferencias y la separación entre transmisor y receptor influyen directamente al CFO introducido en las transmisiones inalámbricas.

El ambiente severo (valor teórico esperado de $\Delta_f = 200\ kHz$) no fue simulado debido a la falta de herramientas necesarias para elevar el valor de CFO en el espacio que tiene el laboratorio, tales como usuarios con dispositivos utilizando la misma banda de frecuencia y/o un mayor número de obstáculos entre el Tx y el Rx.

CONCLUSIONES

En este trabajo de titulación se presentó las simulaciones realizadas en el lenguaje Python de las tres etapas de sincronismo para OFDM, la detección del paquete, el sincronismo de tiempo y el sincronismo de frecuencia, basándose en los parámetros definidos en el estándar IEEE802.11a. Los algoritmos fueron implementados en un sistema SDR, usando el software de GNURadio y los dispositivos hardware USRP N210. Finalmente, se evaluó la precisión de estos algoritmos en el entorno del Laboratorio de Telecomunicaciones de la UTPL, ubicando el receptor a una distancia de 30 cm y 6 m frente al transmisor. El uso de estos equipos en conjunto con el SDR de GNURadio facilitó enteramente la implementación de los algoritmos de sincronismo gracias a la característica de crear bloques propios OOT bajo nuestros parámetros específicos, con la posibilidad de ser modificados en cualquier instante.

La primera etapa del sincronismo es la detección del paquete, efectuada al correlacionar la señal de entrada (preámbulo) con la misma señal retrasada 16 muestras, retraso que equivale a la longitud de un símbolo de la STS. La amplitud de la señal de autocorrelación fue contrastada con la de la potencia media multiplicada por un umbral o límite. El valor escogido del umbral fue de 0.5 en base a la literatura analizada con el fin de minimizar la detección de falsos positivos y así evitar fallas en la detección de paquetes válidos. Cuando la señal autocorrelacionada fue mayor a la potencia media en, al menos, 32 de sus muestras equivalentes a dos símbolos STS, su resultado fue una señal activa de tipo escalón evidenciando la detección del preámbulo en el receptor.

Para resolver la segunda etapa del sincronismo (sincronismo de tiempo), se simularon e implementaron dos algoritmos: correlación cruzada con la STS y correlación cruzada con la LTS. El primero se correlacionó el preámbulo con un vector formado por un símbolo de STS (16 muestras) y 304 ceros de relleno (320 muestras en total). La señal resultante mostró 10 picos que representaron el final de cada símbolo de la STS del preámbulo. El segundo algoritmo se correlacionó el preámbulo con un vector formado por un símbolo LTS (64 muestras) y 256 ceros de relleno. Como resultado se obtuvo 3 picos que indican el final del prefijo cíclico, y de los dos símbolos LTS, respectivamente. Al implementar estos dos algoritmos, se transmitió el preámbulo 300 veces consecutivas y se rastreó la posición de los picos finales mediante el uso de ventanas de 16 muestras de longitud para la STS y de 64 muestras para la LTS. Los efectos del canal inalámbrico causaron una variación de dichas posiciones en cada una de las repeticiones. La varianza de los resultados obtenidos fue de 6.59 en la STS y de 45 en la LTS. Por tanto, se concluye que el algoritmo que mejor desempeño tiene es el de correlación cruzada con la STS, ya que tiene menor varianza que el algoritmo de correlación cruzada con la LTS; además, el uso de los aventanados reduce el tiempo del proceso de detección al evitar el análisis de cada uno de los picos dentro de las

secuencias de entrenamiento, enfocándose solamente en aquellos dentro de los límites de la ventana.

Finalmente, se realizó la última etapa del sincronismo, el desplazamiento de frecuencia, al determinar la fase del valor más alto en la meseta obtenida de la correlación de la señal de entrada con la misma retrasada 16 muestras. El efecto del canal inalámbrico se derivó en la introducción de CFO representada como la multiplicación de la señal transmitida por una exponencial compleja. Su eliminación se logró al multiplicar la señal recibida (con CFO) por una exponencial compleja similar a la mencionada anteriormente, pero de símbolo opuesto. El desempeño de este algoritmo se evaluó de dos maneras, la primera fue ubicando el transmisor y receptor (USRPs) a 30 cm de separación obteniendo una varianza de 2.2 y a 6 m con una varianza de 9.16 en 300 repeticiones. Se concluyó que entre mayor es la separación entre los elementos del sistema, mayor es el valor detectado de CFO en el receptor. Las condiciones óptimas del entorno provocaron un desplazamiento de frecuencia cercana a 0 kHz; mientras que, el aumento de la distancia entre los dispositivos USRPs elevó los valores de CFO obtenidos en cada una de las repeticiones realizadas.

RECOMENDACIONES

Basándonos en los resultados obtenidos en este trabajo, se recomienda que en futuras investigaciones:

- Se instalen las versiones compatibles entre GNURadio y UHD para evitar problemas en la compilación e instalación de los bloques OOT creados por medio de la terminal de Ubuntu.
- Se establezca un límite en las repeticiones enviadas del preámbulo en GNURadio ya que, de momento, los bloques *Source* no lo permiten. Con ello, se logrará obtener valores resultantes de la detección de CFO, la detección de la posición correcta en la correlación cruzada de STS y de LTS que no difieran tanto unos de otros y, por lo tanto, obtener una varianza reducida.
- Conociendo ya el valor de la posición del pico en el que finaliza el último símbolo de la STS y/o el último símbolo de la LTS, se calcule el desplazamiento temporal T_d respecto al valor esperado de las muestras y éste sea corregido.
- Se pruebe los algoritmos aquí desarrollados, en ambientes con interferencias considerables y el preámbulo sea transmitido junto a datos relevantes para el usuario.
- Las tres etapas de sincronismo sean desarrolladas en un solo bloque OOT y éste sea implementado posteriormente en los USRPs.
- Este tipo de implementación puede ser fácilmente adaptada a cualquier tipo de sincronismo de un sistema multiportadora que sea basado en uso de preámbulo y autocorrelación, con lo cual los resultados de esta investigación sirven como punto de partida para sistemas inalámbricos tipo 5G.

BIBLIOGRAFÍA

[1] C. El Hajjar, “Synchronization Algorithms for OFDM Systems (IEEE802.11a, DVB-T) : Analysis, Simulation, Optimization and Implementation Aspects, Synchronisationsalgorithmen für OFDM Systeme (IEEE802.11a, DVB-T) : Analyse, Simulation, Optimierung und Implementierungsaspekte”, 2007.

[2] S. Leonidakis, “Software-Defined Radio Impementation of OFDM”, Technical University Of Crete, 2014.

[3] M. Gadhiok, “Symbol timing synchronization for OFDM-based WLAN systems”, Thesis, Rice University, 2008.

[4] M. M. Boter, “Design and Implementation of an OFDM-based Communication System for the GNURadio Platform”, Master Thesis, Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Stuttgart, Germany, 2009.

[5] D. O. Barragán, “Implementação em FPGA de algoritmos de sincronismo para OFDM”, Dissertação de Mestrado, Universidade Estadual de Campinas, 2013.

[6] C. Geetha Priya y M. Suganthi, “Two symbol timing estimation methods using Barker and Kasami sequence as preamble for OFDM-based WLAN systems”, *Signal Process.*, vol. 90, núm. 7, pp. 2177–2189, jul. 2010.

[7] IEEE Computer Society, LAN/MAN Standards Committee, Institute of Electrical and Electronics Engineers, y IEEE-SA Standards Board, *Supplement to 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 : High-speed physical layer in the 5 GHz band*. New York, N.Y., USA: Institute of Electrical and Electronics Engineers, 1999.

[8] L. Zhou y M. Saito, “A new symbol timing synchronization for OFDM based WLANs”, en *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754)*, 2004, vol. 2, p. 1210–1214 Vol.2.

[9] T.-D. Chiueh y P.-Y. Tsai, *OFDM Baseband Receiver Design for Wireless Communications*. Wiley Publishing, 2007.

[10] D. Nguyen, “Implementation of OFDM systems using GNURadio and USRP”, *Univ. Wollongong Thesis Collect. 1954-2016*, ene. 2013.

[11] J. L. Sanfuentes, “Software defined radio design for synchronization of 802.11 A receiver”, Monterey, California. Naval Postgraduate School, 2007.

[12] C. E. Kennedy, D. J. Dechene, y A. Shami, “Design, implementation, and evaluation of a field-programmable gate array-based wireless local area network synchronizer”, *Wirel. Commun. Mob. Comput.*, vol. 13, núm. 12, pp. 1082–1094, ago. 2013.

[13] J. Medbo y P. Schramm, “Channel Models for HIPERLAN/2 in Different Indoor Scenarios”, en *Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on*, 1998, vol. 1, pp. 125–129.

[14] M.-H. Hsieh y C.-H. Wei, “A low-complexity frame synchronization and frequency offset compensation scheme for OFDM systems over fading channels”, *IEEE Trans. Veh. Technol.*, vol. 48, núm. 5, pp. 1596–1609, sep. 1999.

[15] T. M. Schmidl y D. C. Cox, “Robust frequency and timing synchronization for OFDM”, *IEEE Trans. Commun.*, vol. 45, núm. 12, pp. 1613–1621, dic. 1997.

[16] C.-S. Peng y K.-A. Wen, “Synchronization for carrier frequency offset in wireless LAN 802.11a system”, en *The 5th International Symposium on Wireless Personal Multimedia Communications*, 2002, vol. 3, pp. 1083–1087 vol.3.

[17] M. Cho, Y. Jung, y J. Kim, “Symbol timing synchronization for IEEE 802.11n WLAN systems”, en *2009 First Asian Himalayas International Conference on Internet*, 2009, pp. 1–6.

[18] M. J. Canet, F. Vicedo, V. Almenar, J. Valls, y E. R. D. Lima, “A common FPGA based synchronizer architecture for Hiperlan/2 and IEEE 802.11a WLAN systems”, en *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754)*, 2004, vol. 1, p. 531–535 Vol.1.

[19] A. Fort, J. W. Weijers, V. Derudder, W. Eberle, y A. Bourdoux, “A performance and complexity comparison of auto-correlation and cross-correlation for OFDM burst synchronization”, en *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP ’03)*, 2003, vol. 2, p. II-341-4 vol.2.

[20] J. Xie, Y. Ding, S. Yang, y L. Qi, “FPGA implementation of frame synchronization and symbol timing synchronization based on OFDM system for IEEE 802.11a”, en *2010 International Symposium on Intelligent Signal Processing and Communication Systems*, 2010, pp. 1–4.

[21] Y. S. Cho, J. Kim, W. Y. Yang, y C. G. Kang, *MIMO-OFDM Wireless Communications with MATLAB*. Wiley Publishing, 2010.

[22] Vidar Bronken Gundersen, “MATLAB commands in numerical Python (NumPy)”. Mathesaurus, 11-sep-2007.

[23] P. Raybaut, *Spyder*. United States: Spyder Project Contributors, 2017.

[24] Python Software Fundation, *Python 2.7.13 documentation*. Python Software Fundation, 2017.

[25] I. Perejil, “Implementación de terminales de Radio Cognitiva en la banda de TV”, Proyecto Final de Carrera, Universitat Politécnica de Catalunya, Barcelona, Catalunya, 2016.

[26] The GNURadio Foundation, Inc, *GNURadio*. Global: The GNURadio Foundation, Inc, 2017.

[27] The GNURadio Foundation, Inc, “Guided Tutorial GNURadio in Python”, *Guided Tutorial GNURadio in Python*, 07-oct-2017. [En línea]. Disponible en: https://goo.gl/3KQfpA. [Consultado: 22-nov-2017].

[28] S. Hares, H. Yanikomeroglu, y B. Hashem, “A relaying algorithm for multihop TDMA TDD networks using diversity”, en *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, 2003, vol. 3, pp. 1939–1943.

[29] MathWorks, *MATLAB*. MathWorks, Inc, 2017.

[30] Program Creek, “Python: Addditive White Gaussian Noise (AWGN) Channel”, *Python scipy.signal.resample Examples*, 28-abr-2017. [En línea]. Disponible en: https://goo.gl/6esVMd. [Consultado: 28-mar-2017].

[31] Ettus Research LLC, “USRP™ N200/N210 Networked Series”. Ettus Research Editorial, 14-sep-2012.

[32] A. Fathy, K. Hamdna-Allah, M. Gamal, y M. Taha, “Implementation of a wireless OFDM system using USRP 2 and N210 kits”, Graduation Project, 2012.

[33] Ettus Research LLC, “WSS016 Antenna Specifications”. Wanshih Electronic, 25-abr-2011.

ANEXOS

ANEXO A

Código en lenguaje Python del algoritmo de generación del preámbulo IEEE802.11a, elaborado en el entorno Spyder.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 13 11:37:51 2017

@author: José Herrera & Vanessa Rodríguez

Referencia: “FPGA Implementation Of Synchronization Algorithms For OFDM”, Diego Barragán
(2013)
"""

# -*- coding: utf-8 -*-

from IPython import get_ipython
def __reset__(): get_ipython().magic('reset -sf')

import numpy as np # Librería para operaciones algebráicas
import matplotlib.pyplot as plt # Librería para presentación de gráficas

fsMHz = 20 # Frecuencia de muestreo
nFFTSize = 64 # Tamaño de la FFT

# Cada bit de símbolo (de a1 hasta a52) es asignado al índice de las portadoras [-26 a -1,
1 a 26]
subcarrierIndex = np.hstack([np.arange(-26,0, dtype=int),np.arange(1,27, dtype=int)])

# Elementos para secuencia STS
inputFFTShortPreamble = np.hstack([np.zeros((8), dtype=np.complex),np.array([1+1j,0,0,0,-
1-1j,0,0,0,\
                                    1+1j,0,0,0,-1-1j,0,0,0,-1-1j,0,0,0,1+1j,0,0,0,\
                                    0,0,0,0,-1-1j,0,0,0,-1-1j,0,0,0,1+1j,0,0,0,\
                                    1+1j,0,0,0,1+1j,0,0,0,1+1j],
dtype=np.complex),np.zeros((7), dtype=np.complex)])

# Elementos para secuencia LTS
inputFFTLongPreamble = np.hstack([np.zeros((6), dtype=float),np.array([1,\
                                  1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,\
                                  1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,1,-1,1,-1,\
                                  1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,\
                                  1,1], dtype=float),np.zeros((5), dtype=float)])

# Cálculo de la IFFT
# IFFT de STS
inputiFFT = (np.sqrt(13.0/6.0) * np.fft.fftshift(inputFFTShortPreamble))
outputiFFT = np.fft.ifft(inputiFFT,nFFTSize) # Secuencia de 64 muestras

# IFFT de LTS
inputiFFT = np.fft.fftshift(inputFFTLongPreamble)
outputiFFT_L = np.fft.ifft(inputiFFT,nFFTSize) # Secuencia de 64 muestras

# Concatenación de múltiples símbolos
# Secuencia Corta STS (64+64+32=160)
outputShortPreamble = np.hstack([outputiFFT,outputiFFT,outputiFFT[0:32]])
# Prefijo Cíclico + 2 Secuencias Largas LTS (32+64+64=160)
outputLongPreamble = np.hstack([outputiFFT_L[32:64],outputiFFT_L,outputiFFT_L])
# Preámbulo 802.11a = Secuencia Corta + Prefijo Cíclico + Secuencia Larga (160+160=320)
preamble_802p11a = np.hstack([outputShortPreamble,outputLongPreamble]) # Canal Ideal

# Introducción de desvío de frecuencia CFO
fdeltakHz = 100.0
fsMHz = 20.0 # AB del canal
```

```
exp = np.exp(1j * 2.0 * np.pi * fdeltakHz * 1e3 * np.arange(0,len(preamble_802p11a)) /
(fsMHz * 1e6))

# Señal transmitida + CFO
preamble_802p11a1 = preamble_802p11a * exp

# Simulación de ruido de canal AWGN
def awgn(input_signal, snr_dB, rate):

    avg_energy = sum(input_signal * input_signal) / len(input_signal)
    snr_linear = 10.0 ** (snr_dB / 10.0)
    noise_variance = avg_energy / (2 * rate * snr_linear)

    if input_signal.dtype is complex:
        noise = np.sqrt(noise_variance) * np.random.rand(len(input_signal)) * (1+1j)
    else:
        noise = np.sqrt(2.0 * noise_variance) * np.random.rand(len(input_signal))

    output_signal = input_signal + noise

    return output_signal

# Señal transmitida + AWGN
preamble_802p11a2 = awgn(preamble_802p11a, 100.0, 0.0000005)

# Señal transmitida + AWGN + CFO
preamble_802p11a3 = awgn(preamble_802p11a1, 100.0, 0.0000005)
```

ANEXO B

Código en lenguaje Python del algoritmo de detección de paquete, elaborado en el entorno Spyder.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 11 17:16:35 2017

@author: José Herrera & Vanessa Rodríguez

Referencia: “FPGA Implementation Of Synchronization Algorithms For OFDM”, Diego Barragán (2013)
"""

from IPython import get_ipython
def __reset__(): get_ipython().magic('reset -sf')

import numpy as np # Librería para operaciones algebráicas
import matplotlib.pyplot as plt # Librería para presentación de gráficas

fsMHz = 20 # Frecuencia de muestreo
nFFTSize = 64 # Tamaño de la FFT

# Cada bit de símbolo (de a1 hasta a52) es asignado al índice de las portadoras [-26 a -1, 1 a 26]
subcarrierIndex = np.hstack([np.arange(-26,0, dtype=int),np.arange(1,27, dtype=int)])

inputFFTShortPreamble = np.hstack([np.zeros((8), dtype=np.complex),np.array([1+1j,0,0,0,-1-1j,0,0,0,\
                                    1+1j,0,0,0,-1-1j,0,0,0,-1-1j,0,0,0,1+1j,0,0,0,\
                                    0,0,0,0,-1-1j,0,0,0,-1-1j,0,0,0,1+1j,0,0,0,\
                                    1+1j,0,0,0,1+1j,0,0,0,1+1j],
dtype=np.complex),np.zeros((7), dtype=np.complex)])

inputFFTLongPreamble = np.hstack([np.zeros((6), dtype=float),np.array([1,\
                                   1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,1,-1,-1,1,\
                                   1,-1,1,-1,1,1,1,1,0,1,-1,-1,1,1,-1,1,-1,\
                                   1,-1,-1,-1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,\
                                   1,1], dtype=float),np.zeros((5), dtype=float)])

# Cálculo de la IFFT
# IFFT de preamble_802p11a
inputiFFT = (np.sqrt(13.0/6.0) * np.fft.fftshift(inputFFTShortPreamble))
outputiFFT = np.fft.ifft(inputiFFT,nFFTSize) # Secuencia de 64 muestras

# IFFT de LTS
inputiFFT = np.fft.fftshift(inputFFTLongPreamble)
outputiFFT_L = np.fft.ifft(inputiFFT,nFFTSize) # Secuencia de 64 muestras

# Concatenación de múltiples símbolos
# Secuencia Corta preamble_802p11a (64+64+32=160)
outputShortPreamble = np.hstack([outputiFFT,outputiFFT,outputiFFT[0:32]])
# Prefijo Cíclico + 2 Secuencias Largas LTS (32+64+64=160)
outputLongPreamble = np.hstack([outputiFFT_L[32:64],outputiFFT_L,outputiFFT_L])
# Preámbulo 802.11a = Secuencia Corta + Prefijo Cíclico + Secuencia Larga (160+32+128=320)
preamble_802p11a = np.hstack([outputShortPreamble,outputLongPreamble]) # En Canal Ideal

# Introducción de desvío de frecuencia CFO
fdeltakHz = 100.0
fsMHz = 20.0 # AB del canal
exp = np.exp(1j * 2.0 * np.pi * fdeltakHz * 1e3 * np.arange(0,len(preamble_802p11a)) / (fsMHz * 1e6))

# Señal transmitida + CFO
```

```
preamble_802p11a1 = preamble_802p11a * exp

# Simulación de ruido de canal AWGN
def awgn(input_signal, snr_dB, rate=1.0):

    avg_energy = sum(input_signal * input_signal) / len(input_signal)
    snr_linear = 10.0 ** (snr_dB / 10.0)
    noise_variance = avg_energy / (2 * rate * snr_linear)

    if input_signal.dtype is complex:
        noise = np.sqrt(noise_variance) * np.random.rand(len(input_signal)) * (1+1j)
    else:
        noise = np.sqrt(2.0 * noise_variance) * np.random.rand(len(input_signal))

    output_signal = input_signal + noise

    return output_signal

# Señal transmitida + AWGN
preamble_802p11a2 = awgn(preamble_802p11a, 10.0, 1.0)

# Señal transmitida + AWGN + CFO
preamble_802p11a3 = awgn(preamble_802p11a1, 10.0, 1.0)

# Retraso
delay = 16

# Auto-Correlación
acc = 0
ac2 = np.array([], dtype=np.complex)
for k in range(0,(len(preamble_802p11a3)-delay)):
    ind1 = k
    ind2 = k + delay
    acc = preamble_802p11a3[ind1] * np.conj(preamble_802p11a3[ind2])
    ac2 = np.hstack([ac2,acc])
ac2 = np.hstack([np.zeros((delay), dtype=np.complex),ac2])

# Filtro CIC
N = delay
delayBuffer = np.zeros((N), dtype=np.complex)
intOut = 0
xn = ac2
y2n = np.array([0]*len(xn), dtype=np.complex)
for ii in range(0,len(xn)):
    # Convolución
    combOut = xn[ii] - delayBuffer[-1]
    delayBuffer[1:] = delayBuffer[0:-1]
    delayBuffer[0] = xn[ii]
    # Integrador
    intOut = intOut + combOut
    y2n[ii] = intOut

# Potencia de la señal
acc = 0
ac2_e = np.array([], dtype=np.complex)
for s in range(0,len(preamble_802p11a3)):
    ind1 = s
    acc = preamble_802p11a3[ind1] * np.conj(preamble_802p11a3[ind1])
    ac2_e = np.hstack([ac2_e,acc])
ac2_e = ac2_e[0:]

# Filtro CIC
N = delay
delayBuffer1 = np.zeros((N), dtype=np.complex)
intOut1 = 0
xn = ac2_e
```

```
y2n_e = np.array([0]*len(xn), dtype=np.complex)
for ii in range(0,len(xn)):
    # Convolución
    combOut1 = xn[ii] - delayBuffer1[-1]
    delayBuffer1[1:] = delayBuffer1[0:-1]
    delayBuffer1[0] = xn[ii]
    # Integrador
    intOut1 = intOut1 + combOut1
    y2n_e[ii] = intOut1

# Parámetros
auto_corr = abs(y2n) ** 2
potencia = 0.5 * (y2n_e) ** 2

# Detección en función del umbral
pdm = np.array([0]*len(xn), dtype=float)
for v in range(0, len(auto_corr)):
    if auto_corr[v] >= potencia[v]:
        pdm[v] = 1.0
    else:
        pdm[v] = 0.0

# Acumulador
N = 16
delayBuffer = np.zeros((N), dtype=int)
intOut = 0
xn = pdm
y2n_sm = np.array([0]*len(xn), dtype=float)
for ii in range(0,len(xn)):
    # Convolución
    combOut = xn[ii] - delayBuffer[-1]
    delayBuffer[1:] = delayBuffer[0:-1]
    delayBuffer[0] = xn[ii]
    # Integrador
    intOut = intOut + combOut
    y2n_sm[ii] = intOut
y2n_sm = y2n_sm / delay

# Figura 2.7
plt.close('all')
plt.figure(1)

plt.subplot(211)
plt.plot(auto_corr, 'k', linewidth=2)
plt.xlim(0, len(auto_corr))
plt.ylim(-0.002, np.amax(auto_corr)+0.003)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Autocorrelación $|R(d)|^2$', fontsize=15)

plt.subplot(212)
plt.plot(potencia, 'k', linewidth=2)
plt.xlim(0, len(potencia))
plt.ylim(-0.002, np.amax(auto_corr)+0.003)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Potencia $(0.5*P(d))^2$', fontsize=15)

plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=0.5)

# Figura 2.8
plt.figure(2)

plt.subplot(211)
plt.plot(pdm, 'k', linewidth=2)
plt.xlim(0, len(pdm))
```

```
plt.ylim(-0.05, np.amax(pdm)+0.1)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Detección $M(d)=|R(d)|^2/P(d)^2$', fontsize=15)

plt.subplot(212)
plt.plot(y2n_sm, 'k', linewidth=2)
plt.xlim(0, len(y2n_sm))
plt.ylim(-0.1, np.amax(y2n_sm)+0.1)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Detección $M(d)$ con acumulador de 16 muestras', fontsize=15)

plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=0.5)

# Comparación
plt.figure(3)

plt.subplot(211)
aut, = plt.plot(auto_corr, 'b', linewidth=2)
pot, = plt.plot(potencia, 'g', linewidth=2)
plt.legend([aut, pot], [u"$|R(d)|^2$: Autocorrelación", u"$P(d)^2$: Potencia$*$Umbral"],
fontsize=15)
plt.xlim(0, len(auto_corr))
plt.ylim(-0.001, np.amax(auto_corr)+0.02)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Autocorrelación y Potencia', fontsize=15)

plt.subplot(212)
pdm, = plt.plot(pdm, 'k--', linewidth=2)
det, = plt.plot(y2n_sm, 'r', linewidth=2)
plt.legend([pdm, det], [u"Detección", u"Detección con acumulador de 16"], fontsize=15)
plt.xlim(0, len(auto_corr))
plt.ylim(-0.03, 1.03)
plt.grid(True, linewidth=1.5)
plt.xlabel('Muestras', fontsize=15)
plt.title(u'Detección de Paquete', fontsize=15)

plt.tight_layout(pad=0.5, w_pad=0.5, h_pad=0.5)

#Mostrar gráficas
plt.show()
```

ANEXO C

Código en lenguaje Python del algoritmo de detección de paquete, elaborado en el entorno GNURadio.

```
#!/usr/bin/env python

"""
Created on Mon Sep 20 09:14:35 2017

@author: José Herrera & Vanessa Rodríguez

"""


import numpy as np # Libreria para operaciones algebraicas
from gnuradio import gr # Libreria para ejecutar aplicaciones de GNU Radio

# Declaracion del tipo de bloque (sync) y del nombre bloque

class Frame_Detect_cf(gr.sync_block):

    # Inicializacion
    def __init__(self):
        gr.sync_block.__init__(self,
            name="Frame_Detect_cf",
            in_sig=[np.complex64],# Tipo de datos entrantes
            out_sig=[np.float32]) # Tipo de datos salientes

    # Desarrollo de operaciones
    def work(self, input_items, output_items):
        in0 = input_items[0][0:320] # Entrada de 320 valores
        out = output_items[0][0:320] # Salida de 320 valores

        print '-------------------------'
      preamble_802p11a = in0
      delay = 16 # Retraso

      # Auto-Correlacion
      acc = 0
      ac2 = np.array([], dtype=np.complex)
      for k in range(0,(len(preamble_802p11a)-delay)):
          ind1 = k
          ind2 = k + delay
          acc = preamble_802p11a[ind1] * np.conj(preamble_802p11a[ind2])
          ac2 = np.hstack([ac2,acc])
      ac2 = np.hstack([np.zeros((delay), dtype=np.complex),ac2])

      # Filtro CIC
      N = delay
      delayBuffer = np.zeros((N), dtype=np.complex)
      intOut = 0
      xn = ac2
      y2n = np.array([0]*len(xn), dtype=np.complex)
      for ii in range(0,len(xn)):
          # Convolucion
          combOut = xn[ii] - delayBuffer[-1]
          delayBuffer[1:] = delayBuffer[0:-1]
          delayBuffer[0] = xn[ii]
          # Integrador
          intOut = intOut + combOut
          y2n[ii] = intOut

      # Potencia de la senal
      acc = 0
```

```
        ac2_e = np.array([], dtype=np.complex)
        for s in range(0,len(preamble_802p11a)):
            ind1 = s
            acc = preamble_802p11a[ind1] * np.conj(preamble_802p11a[ind1])
            ac2_e = np.hstack([ac2_e,acc])
        ac2_e = ac2_e[0:]

        # Filtro CIC
        N = delay
        delayBuffer1 = np.zeros((N), dtype=np.complex)
        intOut1 = 0
        xn = ac2_e
        y2n_e = np.array([0]*len(xn), dtype=np.complex)
        for ii in range(0,len(xn)):
            # Convolucion
            combOut1 = xn[ii] - delayBuffer1[-1]
            delayBuffer1[1:] = delayBuffer1[0:-1]
            delayBuffer1[0] = xn[ii]
            # Integrador
            intOut1 = intOut1 + combOut1
            y2n_e[ii] = intOut1

        # Parametros
        auto_corr = abs(y2n) ** 2
        potencia = 0.5 * (y2n_e) ** 2

        # Deteccion en funcion del umbral
        pdm = np.array([0]*len(xn), dtype=int)

        for v in range(0,len(auto_corr)):
            if auto_corr[v] > potencia[v]:
                pdm[v] = 1
            else:
                pdm[v] = 0

        # Acumulador
        N = delay
        delayBuffer = np.zeros((N), dtype=int)
        intOut = 0
        xn = pdm
        y2n_sm = np.array([0]*len(xn), dtype=float)
        for ii in range(0,len(xn)):
            # Convolucion
            combOut = xn[ii] - delayBuffer[-1]
            delayBuffer[1:] = delayBuffer[0:-1]
            delayBuffer[0] = xn[ii]
            # Integrador
            intOut = intOut + combOut
            y2n_sm[ii] = intOut
        y2n_sm = y2n_sm / delay

    # Datos salientes del bloque
    out[:] = y2n_sm
    return len(output_items[0][0:len(in0)])
```

Código en lenguaje Python del algoritmo de detección de paquete, elaborado en el lenguaje *xml.*

```
<?xml version="1.0"?>
<block>
  <name>Detector de Trama</name>
  <key>Tesis_Frame_Detect_cf</key>
  <category>[Tesis]</category>
  <import>import Tesis</import>
  <make>Tesis.Frame_Detect_cf()</make>
```

```
  <!-- Make one 'param' node for every Parameter you want settable from the GUI.
       Sub-nodes:
       * name
       * key (makes the value accessible as $keyname, e.g. in the make node)
       * type
  <param>
    <name>...</name>
    <key>...</key>
    <type>...</type>
  </param>
-->
  <!-- Make one 'sink' node per input. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <sink>
    <name>in</name>
    <type>complex</type>
  </sink>

  <!-- Make one 'source' node per output. Sub-nodes:
       * name (an identifier for the GUI)
       * type
       * vlen
       * optional (set to 1 for optional inputs) -->
  <source>
    <name>out</name>
    <type>float</type>
  </source>
</block>
```