



Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA



Proyecto Final de Carrera Ingeniería Superior de Telecomunicaciones

**“CONTRIBUCIÓN AL DISEÑO DE GNSS-SDR.
UN RECEPTOR GNSS DE CÓDIGO ABIERTO”**



Autor: Luis Esteve Elfau

Director de Proyecto: Dr. Carles Fernández Prades

Barcelona, Julio 2013

**CONTRIBUCIÓN AL DISEÑO DE GNSS-SDR.
UN RECEPTOR GNSS DE CÓDIGO ABIERTO.**

RESUMEN

Este proyecto presenta GNSS-SDR, un receptor SDR (Radio Definida por Software) de código abierto para sistemas GNSS (Sistemas de Navegación Global por Satélite).

La falta de capacidad de reconfiguración de la mayoría de las plataformas receptoras comerciales y la llegada de nuevos sistemas y señales de radionavegación convierten a los receptores software en un marco atractivo para el diseño de nuevas arquitecturas y algoritmos de procesado de señales. Con el objetivo de explorar el potencial de este escenario futuro con una pluralidad de nuevas señales y bandas de frecuencias disponibles para su utilización, este proyecto presenta las contribuciones al diseño de la arquitectura de software, y ofrece detalles sobre la implementación, de un receptor GNSS multibanda y multisistema.

El resultado es un banco de pruebas para el procesado de señales GNSS que permite cualquier tipo de personalización, incluyendo la intercambiabilidad de las fuentes de señal, los algoritmos de procesado de señal, la interoperabilidad con otros sistemas, formatos standard de salida de datos, y la oferta de interfaces de todas las señales intermedias, parámetros y variables. La liberación del código fuente bajo la licencia GNU General Public License (GPL) asegura usabilidad, la inspección y la mejora continua por parte de la comunidad de investigación, ya que permite la discusión basada en el código tangible y el análisis de los resultados obtenidos con señales reales. El código fuente se complementa con un ecosistema de desarrollo, que consiste en un sitio web (<http://gnss-sdr.org>), así como un sistema de control de revisiones, instrucciones para los usuarios y desarrolladores, y herramientas de comunicación.

Se presenta en profundidad el diseño de los bloques iniciales del Plano de Procesado de Señal del receptor: el acondicionador de señal, el bloque de adquisición y el bloque canal del receptor, así como la extensión de la funcionalidad de los bloques de adquisición y *tracking* del receptor GNSS-SDR para poder realizar el seguimiento de las nuevas señales Galileo E1 disponibles.

En cada apartado se ofrece un análisis teórico, detalles de la implementación de cada bloque y su posterior testeo para corroborar los cálculos realizados, tanto con señales generadas sintéticamente como con señales reales provenientes de los satélites en el espacio.

RESUM

Aquest projecte presenta GNSS-SDR, un receptor SDR (Ràdio Definida per Software) de codi obert per a sistemes GNSS (Sistemes de Navegació Global per Satèl·lit).

La manca de capacitat de reconfiguració de la majoria de les plataformes receptors comercials i l'arribada de nous sistemes i senyals de radionavegació converteixen els receptors software en un marc atractiu per al disseny de noves arquitectures i algoritmes de processat de senyals. Amb l'objectiu d'explorar el potencial d'aquest escenari futur amb una pluralitat de nous senyals i bandes de freqüències disponibles per ser utilitzades, aquest projecte presenta les contribucions al disseny de l'arquitectura del software, i ofereix detalls sobre la implementació, d'un receptor GNSS multibanda i multisistema.

El resultat és un banc de proves per al processament de senyals GNSS que permet qualsevol tipus de personalització, incloent la intercanviabilitat de les fonts de senyal, els algorismes de processament de senyal, la interoperabilitat amb d'altres sistemes, formats estàndard de sortida de dades, i l'oferta d'interfícies de totes les senyals intermèdies, paràmetres i variables. L'alliberament del codi font sota la llicència GNU General Public License (GPL) assegura la usabilitat, la inspecció i la millora contínua per part de la comunitat de recerca, ja que permet la discussió basada en el codi tangible i l'anàlisi dels resultats obtinguts amb senyals reals. El codi font es complementa amb un ecosistema de desenvolupament, que consisteix en un lloc web (<http://gnss-sdr.org>), així com un sistema de control de revisions, instruccions per als usuaris i desenvolupadors, i eines de comunicació.

Es presenta en profunditat el disseny dels blocs inicials del Pla de Processament del Senyal del receptor: l'acondicionador de senyal, el bloc d'adquisició i el bloc canal del receptor, així com l'extensió de la funcionalitat dels blocs d'adquisició i *tracking* del receptor GNSS-SDR per poder fer el seguiment de les noves senyals Galileu E1 disponibles.

En cada apartat s'ofereix una anàlisi teòrica, detalls de la implementació de cada bloc i el seu posterior testeig per corroborar els càlculs realitzats, tant amb senyals generats sintèticament com amb senyals reals provinents dels satèl·lits que s'hi troben a l'espai.

ABSTRACT

This project introduces GNSS-SDR, an open source Global Navigation Satellite System software-defined receiver.

The lack of reconfigurability of current commercial-of-the-shelf receivers and the advent of new radionavigation signals and systems make software receivers an appealing approach to design new architectures and signal processing algorithms. With the aim of exploring the full potential of this forthcoming scenario with a plurality of new signal structures and frequency bands available for positioning, this paper describes the software architecture design and provides details about its implementation, targeting a multiband, multisystem GNSS receiver.

The result is a testbed for GNSS signal processing that allows any kind of customization, including interchangeability of signal sources, signal processing algorithms, interoperability with other systems, output formats, and the offering of interfaces to all the intermediate signals, parameters and variables. The source code release under the GNU General Public License (GPL) secures practical usability, inspection, and continuous improvement by the research community, allowing the discussion based on tangible code and the analysis of results obtained with real signals. The source code is complemented by a development ecosystem, consisting of a website (<http://gnss-sdr.org>), as well as a revision control system, instructions for users and developers, and communication tools.

The project shows in detail the design of the initial blocks of the Signal Processing Plane of the receiver: signal conditioner, the acquisition block and the receiver channel, the project also extends the functionality of the acquisition and tracking modules of the GNSS-SDR receiver to track the new Galileo E1 signals available.

Each section provides a theoretical analysis, implementation details of each block and subsequent testing to confirm the calculations with both synthetically generated signals and with real signals from satellites in space.

“Porque lo mejor de nuestra vida aún está por ocurrir.”

Fernando Alfaro

AGRADECIMIENTOS

En primer lugar, querría expresar mis agradecimientos al director de este proyecto, el Dr. Carles Fernández Prades, por su fe en mí (más de la que yo he tenido en mí mismo algunas veces), por su paciencia infinita y por su saber hacer a la hora de dirigirme en este proyecto, sin él no lo habría acabado.

Este proyecto es parte de GNSS-SDR, un proyecto global en el que han trabajado (y trabajarán) muchas más personas. Gracias a ellos hemos empezado a construir algo de lo que me enorgullezco. En especial mis más sinceros agradecimientos al Dr. Javier Arribas, porque haber trabajado codo con codo con él ha sido una experiencia muy enriquecedora, al Dr Pau Closas, por su predisposición a ayudarme en los temas teóricos, y al Ingeniero Informático Carlos Avilés, ya que fue él quien empezó a poner los primeros granitos de arena en este proyecto.

Querría agradecer también a los miembros del tribunal (Dr. Juan Antonio Fernández Rubio, Dr. Josep Vidal y Dr. Josep Altet) por su predisposición y por el esfuerzo y el tiempo dedicados en leer y valorar el trabajo que he realizado.

No querría olvidarme del Dr. Ferran Marquès Acosta, que siempre me ha demostrado ser una gran persona, y siempre me ha alentado a finalizar lo que un día empecé con él.

Este proyecto se lo quiero dedicar a mis padres, porque ellos me han dado la educación que tengo y sé que por fin estarán orgullosos de haber finalizado esa labor que culmina con la presentación de este trabajo. Gracias por todo lo que me habéis dado.

Pero este proyecto no se podría haber terminado sin la ayuda de muchas otras personas a las que quiero agradecer todo su apoyo brindándoles unas pocas líneas.

En primer lugar, a la persona que me ha tenido que aguantar en estos últimos meses de estrés y de horarios intempestivos, a esa persona que se ha convertido en lo más importante en mi vida, gracias Elena.

A mi hermana Pati, y a David, que siempre me ofrecen un punto de vista diferente y me hacen ver que la vida se puede mirar desde muchos otros prismas.

A Maite, por todo lo que hemos pasado, ¡y lo que nos queda!, por ser como eres y aportar siempre tu granito de arena cuando más te he necesitado.

A Manel, por estar siempre ahí, y a Paco, por llevar tantos años aguantándome, y al resto de mis compañeros y amigos de Epsilon (Cali, Marc, Carles, Àlex, ...) por los momentos que hemos vivido.

A mis primos (Ana Belén, Edzard, Mamen, Sheila, Óscar, Diana, Jesús, Ana, David, Carmen, Marta y los pequeñajos) y a mis tíos (Mari Carmen, Pascual, Jose, Loli, Carlos, Montse, José y Pepe), sin olvidarme de mis abuelos (que en paz descansen) porque me habéis hecho apreciar lo que es para mí mi familia.

A mis amigos del Clot (Gus, Lina, Isra, Rocío, Fer, Pep, Marc, Marta, Jordi, Bego, Rafa, Imma, Jordi y Elena), porque con vosotros todo es muy fácil y gracias a

ello uno se puede olvidar de lo dura que es la vida a veces. A mis amigos de Cubelles (Rubén, Dan, Gastón y Manel), porque sois un punto de apoyo importante en mi vida. A los del grupo (Javi J., Javi H. y Ramón), porque gracias a vosotros he podido hacer realidad un sueño que tenía desde pequeño: subir a un escenario y tocar. A la peña Humpfrey (Edu, Bingen, Cristian, Bea, Lonely, Gemma, Triqui, Bea, Xavi y Thais) por enseñarme otra forma de ver la vida. A mis compañeros de la facultad, en especial a Over, por tantas noches sin dormir. Y a Javi, porque lo que se creó durante la infancia no lo podrá romper nadie, ni la distancia ni la enfermedad, muchos ánimos.

Han sido muchos meses, años, en los que compaginar el proyecto con el trabajo, los estudios, aficiones y vida familiar se me ha hecho muy cuesta arriba, en los que pensaba que nunca lograría acabarlo. Pero por fin puedo decir que lo hice, y el esfuerzo a merecido la pena.

Gracias a todos los que me habéis ayudado en mayor o menor medida, no sabéis el peso que me quito de encima.

Va por todos vosotros.

ÍNDICE

AGRADECIMIENTOS	i
ÍNDICE	iii
Lista de acrónimos	vii
Lista de figuras.....	xi
Lista de tablas	xvi
1 INTRODUCCIÓN	1
1.1 Bibliografía	4
2 DESCRIPCIÓN DEL SISTEMA GNSS-SDR	7
2.1 Resumen	7
2.2 Funcionamiento de un receptor GNSS genérico	7
2.3 Características de GNSS-SDR.....	8
2.4 Paradigmas del diseño de GNSS-SDR	9
2.4.1 Entradas y salidas.....	10
2.4.2 Arquitectura del receptor.....	12
2.4.3 Validación (testeo) de los bloques.....	17
2.4.4 Otras características	19
2.5 Detalles de la implementación.....	20
2.5.1 Plano de Control	20
2.5.2 Plano de Procesado de Señal.....	23
2.6 Bibliografía	30
3 GNSS-SDR: SIGNAL CONDITIONER	34
3.1 Resumen	34
3.2 Introducción	34
3.3 Resampler.....	35

3.3.1	<i>Resampling</i> mediante diezmado	36
3.3.2	Interpolación	46
3.3.3	Remuestreo mediante combinación de diezmado e interpolación: <i>rational resampling</i>	57
3.3.4	Bloques analizados	60
3.3.5	Tests y Resultados	61
3.4	Input Filter	70
3.4.1	Bloques analizados	70
3.5	Unit Test	71
3.6	Conclusiones	72
3.7	Bibliografía	72
4	GNSS-SDR: ACQUISITION	76
4.1	Resumen	76
4.2	Introducción	76
4.2.1	Tipos de arranque del algoritmo de adquisición	76
4.2.2	Modelo de señal	77
4.2.3	Planteamiento general del algoritmo de adquisición	80
4.3	Dimensiones y resolución del espacio de búsqueda (<i>grid</i>)	81
4.4	Decisión sobre la presencia o ausencia de un satélite	82
4.5	Formas de recorrer el espacio de búsqueda	83
4.5.1	Serial Search Acquisition	84
4.5.2	Parallel Frequency Space Search Acquisition	85
4.5.3	Parallel Code Phase Search Acquisition	88
4.6	Diseño del estadístico de prueba	91
4.6.1	Estimadores de Máxima Verosimilitud	94
4.6.2	Función Test de la Adquisición	97
4.7	Cálculo del umbral de decisión	99
4.7.1	Medidas de la fiabilidad de la adquisición: probabilidades de detección, falsa alarma y detección fallida	99

4.7.2 Cálculo de las probabilidades de detección, falsa alarma y detección fallida para una celda.....	100
4.7.3 Caracterización del estadístico de prueba para una celda.	101
4.7.4 Extensión de los resultados al espacio de búsqueda	106
4.8 Bloques diseñados.....	114
4.8.1 <i>AcquisitionInterface</i>	114
4.8.2 Bloques que realizan la implementación del algoritmo <i>Parallel Phase Search Acquisition</i> para la señal GPS L1 C/A.....	116
4.9 Pruebas realizadas y resultados obtenidos.....	120
4.9.1 Test unitarios y de integración para la adquisición	120
4.9.2 Pruebas para verificar los algoritmos implementados y su calidad	
121	
4.10 Conclusiones y futuros trabajos.....	141
4.11 Bibliografía	142
5 GNSS-SDR: CHANNEL	144
5.1 Resumen	144
5.2 Arquitectura y funcionalidades del canal de un receptor GNSS.....	144
5.3 Implementación de GNSS-SDR Channel.....	146
5.3.1 <i>ChannelInterface</i>	146
5.3.2 <i>Channel</i>	147
5.3.3 <i>GpsL1CaChannelFsm</i>	151
5.4 Conclusiones y futuros trabajos	156
5.5 Bibliografía	156
6 EXPERIMENTOS CON SEÑAL REAL	158
6.1 Resumen	158
6.2 Experimento 1: Posicionamiento en tiempo real con la USRP y un ordenador de gama baja.	158
6.2.1 Hardware utilizado	158
6.2.2 Configuración del Software GNSS-SDR.....	160
6.2.3 Resultados obtenidos.....	162

6.3 Experimento 2: Posicionamiento en tiempo real con SiGE GN3S y un ordenador de gama media.	165
6.3.1 Hardware utilizado	165
6.3.2 Configuración del Software GNSS-SDR.....	166
6.3.3 Resultados obtenidos.....	168
6.4 Conclusiones.	171
6.5 Bibliografía	172
7 EXTENSIÓN A LA SEÑAL GALILEO E1.....	173
7.1 Resumen.	173
7.2 Modelo de señal.....	174
7.3 Implementación.....	176
7.3.1 Galileo E1 Acquisition	176
7.3.2 Galileo E1 Tracking.....	180
7.4 Resultados experimentales con señales reales.....	183
7.5 Conclusiones y futuros trabajos	185
7.6 Bibliografía	186
8 CONCLUSIONES Y FUTUROS TRABAJOS	188

Lista de acrónimos

A

- A/D: Analog to Digital
- ADC: Analog Digital Conversion
- AGC: Automatic Gain Control
- API: Application Programming Interface
- ASIC: Application Specific Integrated Circuit
- AWGN: Additive White Gaussian Noise

B

- BASS: Block Adjustment of Synchronizing Signal
- BOC: Binary Offset Carrier
- BPSK: Binary Phase Shift Keying

C

- CBOC: Composite Binary Offset Carrier
- CD: Compact Disc
- CDF: *Cumulative Distribution Function*
- CDMA: Code Division Multiple Access
- CFAR: Constant False Alarm Rate
- CN₀: Carrier to Noise Ratio
- COTS: Commercial Off-The-Shelf
- CPLD: Complex Programmable Logic Device
- CPU: Central Processing Unit
- CSR: Cambridge Silicon Radio
- CTTC: *Centre Tecnològic de Telecomunicacions de Catalunya*

D

- D/A: Digital to Analog
- DFT: Discrete Fourier Transform

- DLL: Delay Lock-Loop
- DS-CDMA: Direct-Sequence Code Division Multiple Access
- DVB-T: Digital Video Broadcasting – Terrestrial
- DVD: Digital Versatile Disc

E

- EGNOS: European Geostationary Navigation Overlay Service

F

- FAF: Federal Armed Forces
- FEC: Forward Error Correction
- FIR: Finite Impulse Response
- FLL: Frequency LockLoop
- FFT: Fast Fourier Transform
- FSM: Finite State Machine

G

- GLONASS: Globalnaya Navigatsionnaya Sputnikovaya Sistema o Global Navigation Satellite System
- GLRT: Generalized Likelihood Ratio Test
- GNSS: Global Navigation Satellite System
- GNSS-SDR: Global Navigation Satellite System – Software Defined Radio
- GNU:GNU is Not Unix
- GPL: General Public License
- GPS: Global Positioning System
- GPSTk: GPS Toolkit
- GPU: *Graphics Processing Unit*
- GSA: European GNSS Agency
- GSNRx: GNSS Software Navigation Receiver

H

- HTML: HyperText Markup Language
- I**
- IC: Integrated Circuit
 - IEEE: Institute of Electrical and Electronics Engineers
 - IF: Intermediate Frequency
 - IIR: Infinite Impulse Response
 - ION: Institute Of Navigation
 - iOS: iPhone OS
 - IOV: In-Orbit Validation
 - ISMB: Istituto Superiore Mario Boella
 - ISO: International Organization for Standardization

K

- KML: Keyhole Markup Language

L

LNA: Low-Noise Amplifier

M

- MF: Matched Filter
- ML: Maximum Likelihood
- MLE: Maximum Likelihood Estimator
- MSAS: Multi-functional Satellite Augmentation System
- MSE: Minimum Square Error
- MUTEX: Mutual Exclusion

N

- NMEA: National Marine Electronics Association

O

- OGC: Open Geospatial Consortium

P

- PC: Personal Computer
- PDF: *Probability Density Function*
- PGA: Programmable Gain Amplifier
- PLAN: Position, Location And Navigation Group
- PLL: Phase LockLoop
- PRN: Pseudo Random Noise
- PRS: Public Regulated Service
- PVT: Position, Velocity, and Time

R

- RF: Radio Frequency
- RINEX: Receiver Independent Exchange Format
- ROC: Receiver Operating Characteristic
- RTF: *Rich Text Format*

S

- SBAS: Satellite Based Augmentation System
- SDR: Software Defined Radio
- SIMD: Single–Input Multiple–Data
- SNR: Signal to Noise Ratio
- SNV: Squared Signal-to-Noise Variance
- SR: Software Radio
- SVN: Satellite Vehicle Number
- SVN: Subversion

T

- TTFF: Time To First Fix

U

- UML: *Unified Modeling Language*
- UPC: *Universitat Politècnica de Catalunya*

- USB: Universal Serial Bus
- USRP: Universal Software Radio Peripheral

V

- VCO: Voltage-Controlled Oscillator
- VOLK: Vector-Optimized Library of Kernels

W

- WAAS: Wide Area Augmentation System

X

- XML: eXtensible Markup Language

Lista de figuras

Figura 2-1: Principio básico del posicionamiento GNSS. Conocida la posición de cuatro satélites (SVNi) y de la distancia de cada satélite respecto al punto de medición (pi) se obtiene la posición de éste (en tres dimensiones) mediante trilateración. Figura extraída de [2-1].8

Figura 2-2: Esquema general del receptor GNSS-SDR. Cada módulo acepta múltiples implementaciones, que pueden ser seleccionadas por el usuario mediante un fichero de configuración.....9

Figura 2-3: Cabezales de radiofrecuencia utilizados con éxito en el software GNSS-SDR. a) USRP V1, b) SiGe GN3S Sampler v2, y c) receptor DVB-T basado en el chipset Realtek RTL2832U.11

Figura 2-4: Jerarquía de clases para el Plano de Procesado de Señal.....23

Figura 2-5: Interfaz general para los bloques de procesado de señal.....24

Figura 2-6: Interfaz general para los bloques de procesado de señal.....25

Figura 2-7: Jerarquía de clases para el módulo de Adquisición. Las clases adaptadoras AcqA y AcqB encapsulan el bloque de adquisición propio en una interfaz compatible con AcquisitionInterface. Los bloques AcqA y AcqB GNU Radio blocks son diferentes implementaciones de la adquisición. El procesado de señal de la adquisición se realiza en estos bloques. Se pueden usar bloques de GNU Radio existentes o implementar unos nuevos.....27

Figura 3-1: El bloque Signal Conditioner con sus entradas y salidas.....35

Figura 3-2: Sistema diezmador.....37

Figura 3-3: Secuencias $x_{old}[n]$, $t[n]$ y $v[n]$ con factor de diezmado $D=3$. (figura extraída de [3-2] y modificada).....	38
Figura 3-4: Transformadas de Fourier de las secuencias $x_{old}[n]$, $v[n]$ y $x_{new}[n]$ con $D=3$	39
Figura 3-5: Proceso de filtrado para evitar la distorsión por <i>aliasing</i> en el diezmado (el LPF se muestra en líneas discontinuas).	40
Figura 3-6: El proceso completo de diezmado con un filtro paso bajo.	40
Figura 3-7: Respuesta frecuencial de un filtro paso bajo real.	41
Figura 3-8: Análisis frecuencial del método de enventanado. Figura extraída de [3-7]	43
Figura 3-9: Interpolación por vecindad. En rojo los puntos originales y en azul la función interpolada. Imagen extraída de [3-19].	46
Figura 3-10: Interpolación Lineal. Imagen extraída de [3-19].....	47
Figura 3-11: Interpolación Polinómica. Imagen extraída de [3-19].	48
Figura 3-12: Relación entre interpolación y diezmado para $L=3$. (figura extraída y modificada de [3-2])	50
Figura 3-13: Transformadas de Fourier del proceso de interpolación con $L=3$. En línea discontinua se muestra el filtro paso bajo ideal de ganancia L	51
Figura 3-14: El proceso completo de interpolación con un filtro paso bajo.....	52
Figura 3-15: (a)Filtro FIR simétrico de $S=13$ coeficientes diseñado por enventanado y (b) señal $v[n]$ generada mediante zero padding con $L=3$ a la que se aplica el filtro.....	54
Figura 3-16: Aplicación del filtro FIR $h[n]$ sobre la señal $v[n]$ para calcular las muestras (a) $x_{new}[7]$, (b) $x_{new}[8]$ y (c) $x_{new}[9]$	55
Figura 3-17: Los coeficientes del filtro FIR agrupados para implementar el banco de filtros polifásico.....	57
Figura 3-18: (a) Combinación de los procesos de diezmado e interpolación y (b) esquema final del “ <i>rational resampler</i> ” con las frecuencias de muestreo en cada punto del esquema.	58
Figura 3-19: Espectros de las señales involucradas en el proceso de remuestreo fraccional con $L=4$ y $D=3$, haciendo hincapié en la evolución de la frecuencia de muestreo en cada caso.	60
Figura 3-20: Remuestreo con <i>gr_rational_resampler</i> , ventana de Hanning y banda de transición del filtro del 10%.	64
Figura 3-21: Remuestreo con <i>gr_rational_resampler</i> , ventana de Hanning y banda de transición del filtro del 20%, donde el error es apreciable a simple vista.	65

Figura 3-22: Remuestreo con <i>gr_pfb_arb_resampler</i> , ventana de Harris y banda de transición del filtro del 10%.	65
Figura 3-23: Remuestreo con <i>direct_resampler_conditioner</i> donde se puede apreciar el error producido por el remuestreo por vecindad.	66
Figura 4-1 : Propiedades de las correlaciones de los códigos PRN. Izquierda: autocorrelación del código PRN 1. Derecha: correlación cruzadas de los códigos PRN 1 y 2. Figura extraída de [4-1].	79
Figura 4-2 : Grid bidimensional de la búsqueda de un código CA. Figura extraída de [4-5].	80
Figura 4-3: Diagrama de bloques de la <i>Serial Search Acquisition</i> .	85
Figura 4-4: Densidad espectral de una señal GPS a una IF de 9.548 MHz multiplicada por una secuencia PRN generada localmente. (a) Cuando se multiplica por un código PRN perfectamente alineado, la salida presenta un pico en la frecuencia portadora. (b) Cuando se multiplica por un código no alineado, la salida no presenta picos. Figura extraída de [4-1].	86
Figura 4-5: Diagrama de bloques del algoritmo <i>Parallel Frequency Space Search Acquisition</i> .	86
Figura 4-6: Diagrama de bloques de la Parallel Code Phase Search Acquisition.	90
Figura 4-7: 2 representaciones de las PDFs real y aproximadas del estadístico de prueba (sin normalizar por el factor $2N$) formuladas en la expresión (4-95) para todo el espacio de búsqueda en la hipótesis H_0 cuando el estadístico de prueba normalizado de una celda sigue una distribución chi-cuadrado con 2 grados de libertad. En a) puede observarse la forma general de ambas PDFs, mientras que b) muestra un zoom sobre la zona de convergencia de ambas gráficas, que se da para valores del umbral entorno a $3-4 \times 10^{-3}$.	111
Figura 4-8: Representación del espacio de búsqueda del estadístico $ R_{xd} ^2$ a) para el satélite con PRN 1 (no presente) b) para el satélite con PRN 11 (presente).	123
Figura 4-9: Histograma y PDF teórica para los estadísticos de prueba TH_0 y TH_1 para diversos valores de CNO: a) 35 dBHz, b) 38 dBHz, c) 41 dBHz, d) 44 dBHz, e) 47 dBHz y f) 50 dBHz.	126
Figura 4-10: Representación de los histogramas reales y de la PDF teórica del estadístico de prueba TH_0 para diferentes valores de CN0.	128
Figura 4-11: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 35 dBHz.	130
Figura 4-12: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 38 dBHz.	130
Figura 4-13: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 41 dBHz.	131

Figura 4-14: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 44 dBHz.....	131
Figura 4-15: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 47 dBHz.....	132
Figura 4-16: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN0 de 50 dBHz.....	132
Figura 4-17: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para CN0 = 35 dBHz	134
Figura 4-18: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para CN0 = 38 dBHz	134
Figura 4-19: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para CN0 = 41 dBHz	135
Figura 4-20: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para CN0 = 44 dBHz	135
Figura 4-21: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para CN0 = 47 dBHz	136
Figura 4-22: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 35 dBHz.....	137
Figura 4-23: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 38 dBHz.....	138
Figura 4-24: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 41 dBHz.....	138
Figura 4-25: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 44 dBHz.....	139
Figura 4-26: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 47 dBHz.....	139
Figura 4-27: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una CN0 = 44 dBHz y diversas frecuencias de muestreo (2, 4 y 8 Msps).....	140

Figura 5-1: Arquitectura del bloque Channel de GNSS-SDR que encapsula a los bloques Acquisition, <i>Tracking</i> y Telemetry Decoder y puede replicarse fácilmente. Además incluye una cola concurrente de mensajes para comunicarse con los bloques de adquisición y <i>tracking</i>	145
Figura 5-2: Diagrama de estados de la máquina de estados finitos implementada en el canal.....	153
Figura 6-1: Antena Novatel GPS-600 (imagen izquierda) y Cabezal USRP equipada con la placa DBSRX (imagen derecha) utilizados en el experimento 1.....	159
Figura 6-2: Equipo completo utilizado en el experimento 1.....	159
Figura 6-3: Survey Window del software VisualGPS que representa gráficamente las medidas de posición, así como los promedios de estas medidas, para el experimento 1	162
Figura 6-4: Azimuth&Elevation Window, Signal Quality Window y Navigation Window del software VisualGPS que representan la información de elevación y azimuth de los satélites, su SNR y las medidas de la última posición respectivamente para las señales del experimento 1.....	163
Figura 6-5: Representación en Google Earth de los datos de la posición guardada en el fichero .kml para el experimento 1.....	164
Figura 6-6: Equipo completo utilizado en el experimento 2.....	165
Figura 6-7: Survey Window del software VisualGPS que representa gráficamente las medidas de posición, así como los promedios de estas medidas, para el experimento 2	169
Figura 6-8: Azimuth&Elevation Window, Signal Quality Window y Navigation Window del software VisualGPS que representan la información de elevación y azimuth de los satélites, su SNR y las medidas de la última posición respectivamente para las señales del experimento 2.....	170
Figura 6-9: Representación en Google Earth de los datos de la posición guardada en el fichero .kml para el experimento 2.....	171
Figura 7-1: Los 4 satélites Galileo en órbita han dado su primer posicionamiento positivo el 12 de marzo de 2013. Figura extraída de [7-1]	173
Figura 7-2: Un período de la sub-portadora CBOC para a) la componente de señal E1-B, y b) la componente de la señal E1-C. Figura extraída de [7-5].	175
Figura 7-3: $ R_{xd}(f=f_d, T) ^2$ normalizada para diversas formas de onda de la señal local $d[n]$ y diferentes frecuencias de muestreo.	178
Figura 7-4: Algoritmo 1, algoritmo que implementa la adquisición mediante el método <i>Parallel Code Phase Search</i> (PCPS) para el software GNSS-SDR.....	179
Figura 7-5: Algoritmo 2, algoritmo de <i>Tracking</i> para señales Galileo E1 utilizado en GNSS-SDR.....	182
Figura 7-6: Equipo utilizado para realizar las pruebas de adquisición y <i>tracking</i> de las señales Galileo E1B y E1C.....	183

Figura 7-7: Estadístico GLRT para el algoritmo de adquisición *Parallel Code Phase Search* y una configuración con $f_{IN}=4$ Msps, un margen frecuencial que va de -5 kHz a 5 kHz con unos pasos de 250 Hz, utilizando como réplica local la señal sinBOC E1B correspondiente al satélite de PRN 11. 184

Figura 7-8: Evolución de las muestras de la correlación VE, E, P, L y VL en una ejecución del bloque de *tracking GalileoE1DIIPII\Ve\m\Tracking*. 185

Lista de tablas

Tabla 3-1: Resultados de las medidas de calidad de los *resamplers* analizados. 63

Tabla 3-2: Datos técnicos de la generación de la señal GPS utilizada en los test de remuestreo..... 67

Tabla 3-3: Medidas de tiempo de ejecución de los *resamplers* analizados..... 69

1 INTRODUCCIÓN

La localización se ha convertido en una característica implícita no sólo en los teléfonos móviles de gama media y alta, sino también en otros dispositivos portátiles como cámaras digitales y consolas portátiles de videojuegos. Este despliegue masivo de receptores GNSS requiere de un alto nivel de integración a bajo coste, un tamaño reducido y un consumo de energía mínimo, cosa que ha llevado a los principales fabricantes de circuitos integrados (IC) para GPS, como Qualcomm Inc., Broadcom Corporation, Cambridge Silicon Radio (CSR, que se fusionó con SiRF en 2009), Texas Instruments Inc., ST Microelectronics, u-blox AG, Maxim o MediaTek a ofrecer soluciones en un solo chip de fácil integración en dispositivos multifunción. Por lo tanto, el cabezal de radiofrecuencia (RF front-end) y el procesado en banda base se implementan conjuntamente en circuitos integrados monolíticos, pequeñas cajas negras que dejan al usuario sin posibilidad de interactuar o modificar la arquitectura interna o la algoritmia del software receptor. Este enfoque es conveniente para servicios y aplicaciones basados en la localización, ya que tanto usuarios como desarrolladores están interesados solamente en el uso de la información de localización (a veces aprovechando información complementaria obtenida a través proveedores de redes inalámbricas), pero no en la forma en que ha sido obtenida la posición. Algunos ejemplos de esta abstracción se pueden encontrar en las interfaces de programación de aplicaciones (API) de dos de los grandes sistemas operativos para dispositivos móviles. El iOS de Apple ofrece un *framework* con objetos que incorporan las coordenadas geográficas y la altitud de la localización del dispositivo junto con valores que indican la exactitud de las medidas, cuándo se realizaron, y la información sobre la velocidad y el rumbo del dispositivo en movimiento. En una situación similar se encuentra Android, que ofrece un paquete de localización que contiene clases con métodos cuyos nombres son muy descriptivos, tales como `getLatitude()`, `getLongitude()`, `getAltitude()`, `getSpeed()`, `getAccuracy()` y así sucesivamente. Esta capa de abstracción simplifica mucho el trabajo del desarrollador de aplicaciones, pero por el contrario no permite observar o modificar cualquier aspecto interno del receptor.

Por otro lado, el lanzamiento de una serie de nuevos sistemas GNSS (Galileo, COMPASS), la modernización de los ya existentes (GPS L2C y L5, GLONASS L3OC) y el despliegue de sistemas de aumentación (tanto los basados en satélites, tales como WAAS en los EE.UU., EGNOS en Europa y MSAS en Japón, como los basados en sistemas terrestres, tales como el posicionamiento mediante WiFi y los sistemas GNSS asistidos por redes celulares) proporcionan un marco sin precedentes para los diseñadores de nuevos receptores [1-1]. En los próximos años, una multitud de nuevas señales, sistemas y bandas de frecuencia estarán disponibles para uso civil, y su explotación requerirá de un rediseño a conciencia de la arquitectura del receptor y de los algoritmos internos que lo componen.

De esta forma las nuevas señales disponibles plantean el desafío de diseñar receptores multisistema y multibanda, abordando temas como la realización de contramedidas para paliar los efectos de las interferencias, el posicionamiento de alta precisión enfocado al mercado masivo, el GNSS asistido y la hibridación con otras tecnologías.

Además de ser cajas negras ocultas bajo una capa de abstracción, la mayoría de circuitos integrados para GPS disponibles en el mercado están limitados en términos de configuración, flexibilidad y capacidad de actualización. Este hecho ha provocado que la mayoría de diseñadores de receptores se encaminen hacia el paradigma del software radio (SR), en el cual un cabezal de radiofrecuencia analógico realiza la conversión de la señal GNSS a frecuencia intermedia (o directamente a banda base) antes de la conversión analógico digital (ADC). Todo el procesado restante, tanto de señal como de datos, incluyendo la hibridación con otros sistemas, se define en el dominio del software. Este enfoque proporciona a los diseñadores un alto grado de flexibilidad, permitiendo el pleno acceso a cualquier punto de la cadena de los sistemas que componen el receptor y la posibilidad de realizar las modificaciones pertinentes en dichos puntos.

La última década ha sido testigo de una rápida evolución de los receptores de software GNSS. Desde el primer receptor software (Servicio Estándar de Posicionamiento GPS) que se describe en [1-2], donde se introdujo el concepto de muestreo paso banda (también llamado aliasing intencional), varios trabajos han profundizado sobre aspectos de la arquitectura e implementación de este tipo de receptores [1-3], [1-4], [1-5], [1-6], [1-7], [1-8], [1-9], [1-10] y [1-11]. Los libros de texto [1-12] y [1-13] ayudaron a concienciar a la comunidad sobre los grandes beneficios proporcionados por los receptores software con respecto al enfoque tradicional orientado al hardware, proporcionando implementaciones en Matlab de receptores GPS completos, y en [1-14] y [1-15] se presentan discusiones sobre diseños de arquitecturas de alto nivel. En [1-16] (en cuya realización participa el autor), se presenta un análisis de los patrones de diseño del software y su aplicación a los receptores software GNSS.

Hoy en día, hay soluciones disponibles tanto a nivel académico como comercial que, por lo general, no sólo incluyen soluciones de programación, sino también incluyen el desarrollo de cabezales de radiofrecuencia dedicados. Como ejemplos, podemos mencionar GSNRx (Receptor Software de Navegación GNSS [1-14]), desarrollado por el Grupo de Posición, Localización y Navegación (PLAN) de la Universidad de Calgary, el ipexSR, un receptor software multi-frecuencia (GPS C/A y L2C, EGNOS y GIOVE-A E1-E5a) desarrollado por el Instituto de Geodesia y Navegación en la Universidad de Munich FAF [1-17] y [1-18], o N-Gene, un receptor de software totalmente desarrollado por el Istituto Superiore Mario Boella (ISMB) y el Politecnico di Torino que es capaz de procesar en tiempo real señales de radiodifusión de GPS y Galileo en las bandas L1/E1, así como demodular las correcciones diferenciales transmitidas por el sistema EGNOS. Este receptor es capaz de procesar en tiempo real más de 12 canales, con una frecuencia de muestreo de aproximadamente 17.05 MHz con 8 bits por muestra [1-19].

En este contexto, este proyecto presenta un receptor GNSS definido por software (llamado así GNSS-SDR) de código abierto publicado bajo licencia GNU General Public License (GPL), hecho que garantiza la libertad de modificar, compartir y utilizar el código con cualquier propósito. Esto facilita la usabilidad, inspección y continua mejora del código por parte de la comunidad de investigación, ya que permite

discutir en base a código tangible así como analizar resultados obtenidos a partir de señales reales. Por lo tanto, pretende ser también un marco de prueba de algoritmos y una herramienta educativa, ya que se le permite a todo el mundo consultar libremente el código fuente, ver cómo se implementa el receptor y contribuir a su desarrollo a través de la aportación de mejoras, la corrección de errores, así como la adición de nuevas características.

El diseño del receptor GNSS-SDR se centra en el procesado de la señal, entendido como el procesado que se da entre el ADC y el cálculo de los observables de código y fase, incluyendo la demodulación del mensaje de navegación. Se omite, intencionadamente, el procesado de datos, entendido como el cálculo de la localización a partir de los observables y el mensaje de navegación, ya que para ello ya existen una serie de librerías y aplicaciones altamente testeadas (algunas de ellas de código abierto, como GPSTk [1-20] y [1-21]).

El objetivo principal del autor de este proyecto es realizar una contribución al desarrollo del receptor GNSS-SDR, un receptor GNSS definido por software al que se pretende dotar de las siguientes características:

- Capacidad para trabajar en diversas bandas de frecuencia y con señales de diversos sistemas, es decir, que sea multibanda y multisistema.
- Capacidad de trabajar en tiempo real o en postprocesado mediante archivos de señal muestreada.
- Capacidad de proporcionar interfaces para una gran variedad de cabezales de radiofrecuencia disponibles en el mercado, así como para aquellos que puedan ser diseñados por los usuarios, a través de los buses USB y Ethernet
- Capacidad de ser completamente personalizable: incluyendo la intercambiabilidad de las fuentes de señal, de los algoritmos de procesado de señal, asegurar la interoperabilidad con otros sistemas, ofrecer diferentes formatos de salida, así como ofrecer interfaces para todas las señales, parámetros y variables intermedias.
- Capacidad para ejecutarse en una gran variedad de plataformas hardware y de sistemas operativos.

Concretamente el autor pretende realizar las siguientes aportaciones:

- Diseño y testeo del bloque de acondicionamiento de la señal.
- Diseño y testeo del bloque de adquisición.
- Diseño y testeo del bloque canal.
- Realizar la extensión de la funcionalidad del receptor para realizar la adquisición y *tracking* de señales Galileo E1.

El resto del proyecto se estructura de la siguiente manera: en el Capítulo 2 se realiza una descripción general del receptor GNSS-SDR, se esbozan las principales características y objetivos del receptor, se identifican las posibles fuentes de señal, se realiza una descripción de la arquitectura general del receptor y de los patrones de

diseño utilizados y se analiza cuáles son los formatos más útiles para la salida de datos. Los tres siguientes capítulos abordan el diseño de varios bloques realizados por el autor para el receptor GNSS-SDR. Cada capítulo consta de un desarrollo teórico, un apartado de detalles de la implementación, así como una serie de pruebas y conclusiones y de su propia bibliografía. De esta forma, en el Capítulo 3 se analiza el bloque de acondicionamiento de la señal (GNSS-SDR Signal Conditioner) haciendo hincapié en el proceso de remuestreo y la manera en que se ha implementado en el receptor con el fin de conseguir que el sistema funcione en tiempo real. El Capítulo 4 aborda el proceso de la adquisición del sistema GNSS-SDR, profundizando en la forma de calcular los umbrales de decisión que maximizan la probabilidad de detección para una determinada probabilidad de falsa alarma. Posteriormente, en el Capítulo 5 se aborda el diseño del bloque canal, especialmente la máquina de estados finitos (FSM) que controla la lógica a implementar en este bloque. En Capítulo 6 se presentan dos experimentos con señal real para verificar la integración de los bloques diseñados en el receptor completo. El Capítulo 7 consiste en una extensión del trabajo realizado a la señal Galileo E1. Este capítulo es un resumen del trabajo desarrollado por el autor durante el Google Summer of Code 2012 (GSOC 2012) [1-22], por el cuál ha sido becado, y que ha sido presentado en la conferencia GNU Radio 2012 [1-23], dicha presentación puede verse en [1-24]. Finalmente el Capítulo 8 cierra el proyecto recopilando todas las conclusiones presentadas en todos los capítulos anteriores y planteando futuras tareas a desarrollar.

1.1 Bibliografía

[1-1] C. Fernández-Prades, L. Lo Presti, and E. Falleti, "Satellite radiolocation from GPS to GNSS and beyond: Novel technologies and applications for civil massmarket," Proceedings of the IEEE, Nov. 2011, In Press. DOI: 10.1109/JPROC.2011.2158032.

[1-2] D. Akos, A Software Radio Approach to Global Navigation Satellite System Receiver Design, Ph.D. thesis, College of Engineering and Technology, Ohio University, Aug. 1997.

[1-3] K. Krumvieda, P. Madhani, C. Cloman, E. Olson, J. Thomas, P. Axelrad, and W. Kober, "A complete IF software GPS receiver: A tutorial about the details," in Proc. of the 14th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GPS'01), Salt Lake City, UT, Sept 2001, pp. 789–829.

[1-4] V. Chakravarthy, J. Tsui, D. Lin, and J. Schamus, "Software GPS receiver," GPS Solutions, vol. 5, no. 2, pp. 63–70, Oct. 2001.

[1-5] B. M. Ledvina, S. P. Powell, P. M. Kintner, and M. L. Psiaki, "A 12-channel real-time GPS L1 software receiver," in Proc. of the National Technical Meeting of the Institute of Navigation (ION NTM'03), Anaheim, CA, Jan. 2003, pp. 767–782.

[1-6] G. W. Heckler and J. L. Garrison, "SIMD correlator library for GNSS software receivers," GPS Solutions, vol. 10, no. 4, pp. 269–276, Nov. 2006.

[1-7] H. Hurskainen, J. Raasakka, T. Ahonen, and J. Nurmi, "Multicore software-defined radio architecture for GNSS receiver signal processing," EURASIP Journal on Embedded Systems, vol. 2009, 2009, Article ID 543720.

[1-8] T. E. Humphreys, J. A. Bhatti, T. Pany, B. M. Ledvina, and B. W. O'Hanlon, "Exploiting multicore technology in software-defined GNSS receivers," in Proc. of the 22nd International Meeting of the Satellite Division of The Institute of Navigation (ION GNSS'09), Savannah, GA, Sept. 2009, pp. 326–338.

[1-9] A. Mitelman, J. Almqvist, R. Håkanson, D. Karlsson, F. Lindström, T. Renström, C. Ståhlberg, and J. Tidd, "Testing software receivers," GPS World, vol. 20, no. 12, pp. 28–34, Dec. 2009.

[1-10] T. Hobiger, T. Gotoh, J. Amagai, Y. Koyama, and T. Kondo, "A GPU based real-time GPS software receiver," GPS Solutions, vol. 14, no. 2, pp. 207–216, Mar. 2010.

[1-11] X. Li and D. Akos, "Implementation and performance of clock steering in a software GPS L1 single frequency receiver," Navigation: Journal of The Institute of Navigation, vol. 57, no. 1, pp. 69–85, Spring 2010.

[1-12] J. Bao-Yen Tsui, Fundamentals of Global Positioning System Receivers. A Software Approach, John Wiley & Sons, Inc., Hoboken, NJ, 2nd edition, 2005.

[1-13] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, A Software-Defined GPS and Galileo Receiver. A Single-Frequency Approach, Applied and Numerical Harmonic Analysis. Birkhäuser, Boston, MA, 2007.

[1-14] M. G. Petovello, C. O'Driscoll, G. Lachapelle, D. Borio, and H. Murtaza, "Architecture and benefits of an advanced GNSS software receiver," Positioning, vol. 1, no. 1, pp. 66–78, 2009.

[1-15] F. Principe, G. Bacci, F. Giannetti, and M. Luise, "Software-defined radio technologies for GNSS receivers: A tutorial approach to a simple design and implementation," International Journal of Navigation and Observation, vol. 2011, pp. 1–27, 2011, Article ID 979815, DOI:10.1155/2011/979815.

[1-16] C. Fernández-Prades, C. Avilés, L. Esteve, J. Arribas, and P. Closas, "Design patterns for GNSS software receivers," in Proc. of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010), ESTEC, Noordwijk, The Netherlands, Dec. 2010, DOI:10.1109/NAVITEC.2010.5707981.

[1-17] M. Anghileri, T. Pany, D. Sanromà-Güixens, J.-H.Won, A. Sicramaz-Ayaz, C. Stöber, I. Krämer, D. Dötterböck, G. W. Hein, and B. Eissfeller, "Performance evaluation of a multi-frequency GPS/Galileo/SBAS software receiver," in Proc. of the 20th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS'07), Fort Worth, TX, Sept. 2007, pp. 2749–2761.

[1-18] C. Stöber, M. Anghileri, A. Sicramaz Ayaz, D. Dötterböck, I. Krämer, V. Kroppand J.-H. Won, B. Eissfeller, D. Sanromà-Güixens, and T. Pany, "ipexSR: A real-time multi-frequency software GNSS receiver," in Proc. of the 52nd International Symposium ELMAR-2010, Zadar, Croatia, Sept. 2010, pp. 407–416.

[1-19] M. Fantino, A. Molino, and M. Nicola, “N–Gene GNSS receiver: Benefits of software radio in navigation,” in Proc. of the European Navigation Conference - Global Navigation Satellite Systems (ENCNNS), Napoles, Italy, May 2009.

[1-20] B. W. Tolman, R. B. Harris, T. Gaussiran, D. Munton, J. Little, R. Mach, S. Nelsen, B. Renfro, and D. Schlossberg, “The GPS toolkit - open source GPS software,” in Proc. of the 17th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS'04), Long Beach, CA, 21–24 Sept. 2004, pp. 2044–2053.

[1-21] D. Salazar, M. Hernández-Pajares, J. M. Juan, and J. Sanz, “GNSS data management and processing with the GPSTk,” GPS Solutions, vol. 14, no. 3, pp. 293–299, June 2010.

[1-22] “Google Summer of Code 2012”, <http://www.google-melange.com/gsoc/homepage/google/gsoc2012>. Comprobado: 25 de junio de 2013.

[1-23] “GNU Radio Conference 2012”, <http://www.trondeau.com/gnu-radio-conference-2012/>. Comprobado: 25 de junio de 2013.

[1-24] “Processing Galileo signals with GNSS-SDR”, http://www.youtube.com/watch?v=ajh2_xBCZSM. Comprobado: 25 de junio de 2013.

2 DESCRIPCIÓN DEL SISTEMA GNSS-SDR

2.1 Resumen

En este capítulo se hace una descripción completa del receptor GNSS-SDR. El capítulo se estructura de la siguiente forma: en el apartado 2.2 se explica el funcionamiento de un receptor GNSS genérico, en el apartado 2.3 se definen las características del software diseñado, en el apartado 2.4 se abordan en detalle los paradigmas del diseño del receptor: las entradas y salidas del mismo, el lenguaje de programación escogido y la arquitectura del receptor. Finalmente, en el último apartado, el 2.5, se hace una descripción funcional de los bloques de GNSS-SDR haciendo hincapié en los dos planos en que se divide el receptor: el Plano de Control y el plano de Procesado de Señal.

2.2 Funcionamiento de un receptor GNSS genérico

Los sistemas globales de navegación por satélite (GNSS) permiten determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una determinada precisión. Dentro de estos sistemas se encuentran el GPS americano, el GLONASS ruso (ambos en estado operativo), el Galileo europeo y el COMPASS chino (estos últimos en fase de desarrollo).

Los sistemas GNSS funcionan mediante una red de satélites en órbita sobre el globo terráqueo, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para tal fin debe detectar y sincronizar señales provenientes de un mínimo de tres satélites de la red, de los que recibe unas transmisiones que contienen, entre otros datos, el identificador y la hora del reloj de cada uno de ellos. En base a estas señales, el aparato sincroniza el reloj del receptor GNSS y calcula el tiempo que tardan en llegar las señales al equipo. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, mediante un algoritmo de trilaterización se obtiene la posición absoluta o coordenadas reales del punto de medición sobre la superficie de la Tierra mediante el valor de las coordenadas de longitud y latitud (dos dimensiones). Dichas coordenadas pueden venir expresadas en grados, minutos y/o segundos o en las unidades de medición utilizadas en otros sistemas geodésicos. La captación de cuatro o más satélites facilita, además, la altura del receptor con respecto al nivel del mar (tres dimensiones), tal y como se puede ver en la Figura 2-1. También se

consigue una exactitud extrema en el reloj del receptor GNSS, similar a la de los relojes atómicos que llevan a bordo cada uno de los satélites.

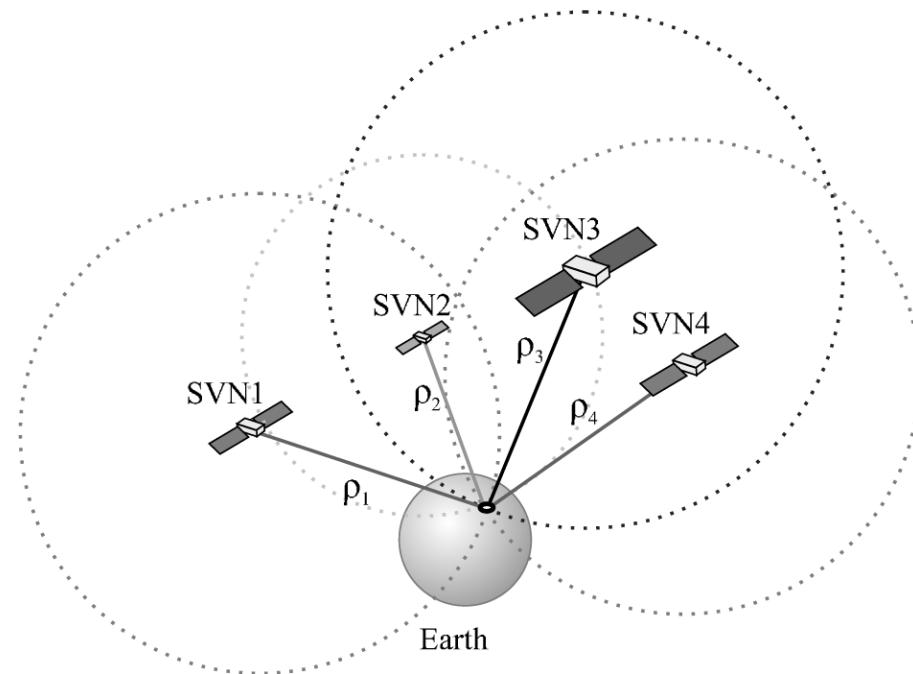


Figura 2-1: Principio básico del posicionamiento GNSS. Conocida la posición de cuatro satélites (SVN_i) y de la distancia de cada satélite respecto al punto de medición (ρ_i) se obtiene la posición de éste (en tres dimensiones) mediante trilateración. Figura extraída de [2-1].

2.3 Características de GNSS-SDR

El receptor propuesto proporciona una interfaz adecuada para diferentes tipos de cabezales de radiofrecuencia e implementa toda la cadena de recepción hasta la solución de navegación. Su diseño permite cualquier tipo de personalización, incluyendo la elección de diversas fuentes de señal, la posibilidad de cambiar los algoritmos internos de procesado de la misma, la interoperabilidad con otros sistemas, diversos formatos de salida, y además ofrece interfaces para todas las señales, parámetros y variables intermedias. El objetivo es escribir un código eficiente y altamente reutilizable, fácil de leer y mantener, con pocos errores, y así producir ejecutables optimizados para una gran variedad de plataformas hardware y de sistemas operativos. En ese sentido, el reto consiste en definir un equilibrio razonable entre el nivel de abstracción y el rendimiento del software. El receptor se ejecuta en un ordenador personal y proporciona interfaces para una gran variedad de cabezales de radiofrecuencia disponibles en el mercado, así como para aquellos que puedan ser diseñados por los usuarios, a través de los buses USB y Ethernet. Esta versatilidad se da gracias a la adaptación de las frecuencias de muestreo, frecuencias intermedias y resoluciones de las muestras de los algoritmos de procesado. Las pruebas de testeo se realizan en dos ámbitos, por un lado se realiza la validación funcional sistemática de cada bloque de software (al realizar un desarrollo de software basado en el testeo y mediante el uso de pruebas unitarias, *unit testing*, como métodos de verificación y validación), y por otro lado se realiza la validación experimental del receptor completo utilizando señales reales y sintéticas.

El diagrama general de bloques del receptor se muestra en la Figura 2-2. El software se compone de un Plano de Control a cargo de la gestión de todo el receptor y de las interacciones con el sistema operativo y con las aplicaciones externas, y de un Plano de Procesado de Señal que se encarga de extraer la información a partir de las muestras de la señal.

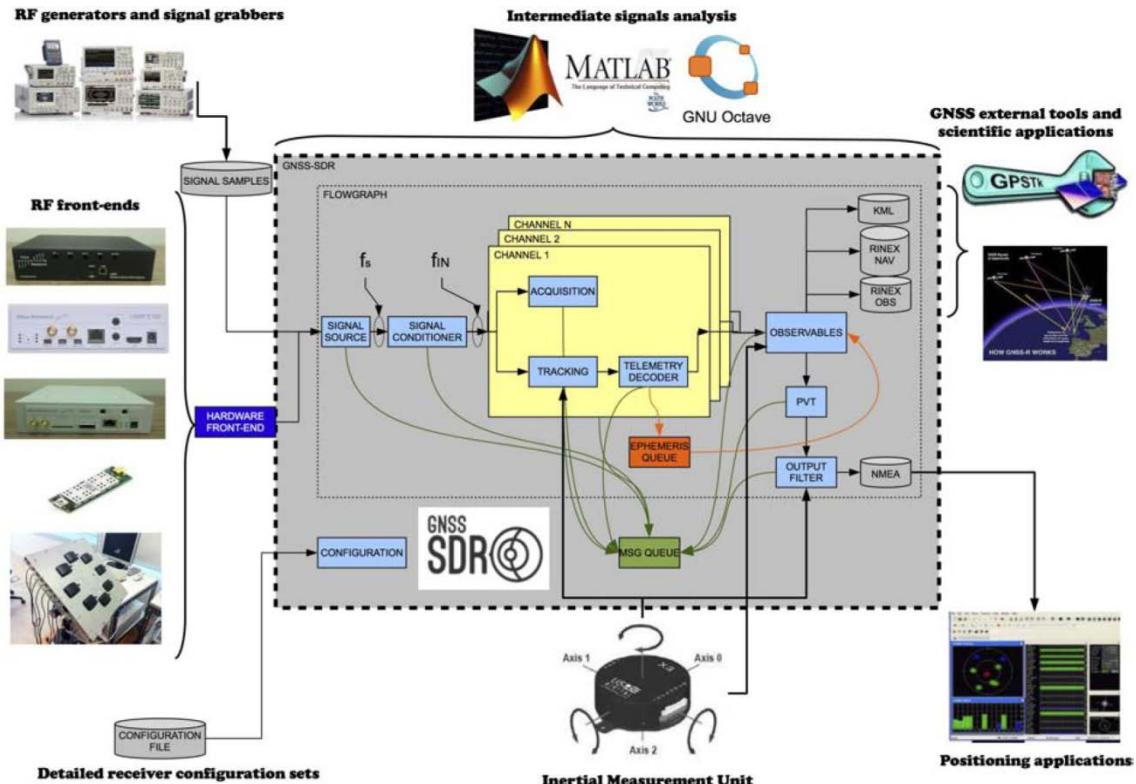


Figura 2-2: Esquema general del receptor GNSS-SDR. Cada módulo acepta múltiples implementaciones, que pueden ser seleccionadas por el usuario mediante un fichero de configuración.

Los aspectos clave del diseño del receptor, así como los detalles más relevantes de su implementación, se desarrollan en los dos apartados siguientes.

2.4 Paradigmas del diseño de GNSS-SDR

En este apartado se abordan aspectos claves del diseño del receptor: las entradas y salidas de las que debe disponer, la arquitectura de software elegida (que aborda temas cruciales como el lenguaje de programación utilizado, el uso de patrones de diseño y del framework GNURadio), el método de validación (testeo) de los bloques diseñados (mediante googletest) y otras características relevantes que propician que GNSS-SDR sea un proyecto bien estructurado y atractivo para futuros desarrolladores.

2.4.1 Entradas y salidas

2.4.1.1 Entradas

Una característica atractiva de un receptor software es la posibilidad de trabajar en tiempo real con señales reales, cuando el procesador es lo suficientemente rápido, o en modo *offline* (post-procesado) utilizando muestras de señal almacenadas en un archivo, cuando la complejidad de la aplicación impide un procesado en tiempo real. Estas señales pueden también ser creadas por generadores de señal sintética con el fin de realizar experimentos con parámetros controlados.

Idealmente, un receptor software debe llevar a cabo la digitalización (ADC) después de la antena. Sin embargo, debido a las limitaciones tecnológicas, todavía existe la necesidad de utilizar un cabezal de radiofrecuencia para realizar la amplificación y la conversión (bajada) en frecuencia antes del ADC. También se necesita una interfaz entre la salida del ADC y el PC (o cualquier otro procesador de propósito general) en el que se ejecute el receptor. Esta "parte hardware" del receptor puede ser implementada con componentes comerciales de uso genérico (COTS - *Commercial Off-The-Shelf*) o aprovechando circuitos integrados existentes para aplicaciones específicas de radiofrecuencia. Los circuitos más modernos ofrecen completos cabezales de radiofrecuencia para GNSS, incluyendo un amplificador de bajo nivel de ruido (LNA) y un mezclador, seguido por un filtro para eliminar las frecuencias imágenes, un amplificador de ganancia programable (PGA), un oscilador controlado por tensión (VCO), un sintetizador de frecuencias, un oscilador de cristal, y un ADC multinivel (generalmente de hasta 3 bits) equipado con el control automático de ganancia (AGC) del sistema.

Existen varias tarjetas para la adquisición de señales disponibles en el mercado. Por ejemplo, la Universal Software Radio Peripheral (USRP) [2-2] (ver Figura 2-3a) es una familia de hardware controlado por ordenador de propósito general dedicado a software radio que equipada con una placa hija DBSRX [2-3] se puede utilizar como cabezal de radiofrecuencia para receptores GNSS. Otras soluciones más específicas, de menor coste, se componen generalmente de una antena, un circuito integrado que incluye el cabezal de RF, un dispositivo complejo de lógica programable (CPLD), que organiza los bits de la muestra en bytes, y un microcontrolador USB 2.0. Este es el caso de la SiGe GN3S Sampler v2 (ver Figura 2-3b), basada en el circuito integrado SiGe 4120 GPS [2-4], que proporciona un flujo de datos con una frecuencia de muestreo de 16.3676 MHz y un ancho de banda de hasta 4.04 MHz, el Primo de NSL (basado en el circuito integrado MAX2769 de Maxim [2-5], que permite seleccionar un ancho de banda de 2.5, 4.2, 8, o 18 MHz y tiene una frecuencia de muestreo de hasta 40 MHz), el Primo II (de las mismas características, pero de doble banda, con un par de circuitos MAX2769), y el NavPort-III de IFEN, un cabezal de radiofrecuencia capaz de trabajar simultáneamente en cuatro bandas de frecuencia, con un ancho de banda de 13 MHz cada una [2-6].

Por último remarcar que recientemente se han llevado a cabo experimentos que permiten utilizar receptores USB de señal digital de televisión DVB-T basados en el chipset Realtek RTL2832U [2-7] (ver Figura 2-3c) como cabezales de radiofrecuencia para Software Defined Radio (SDR) obteniendo con éxito posicionamiento de señales GPS [2-8], cosa que los convierte en la alternativa más barata para realizar trabajos en este campo.

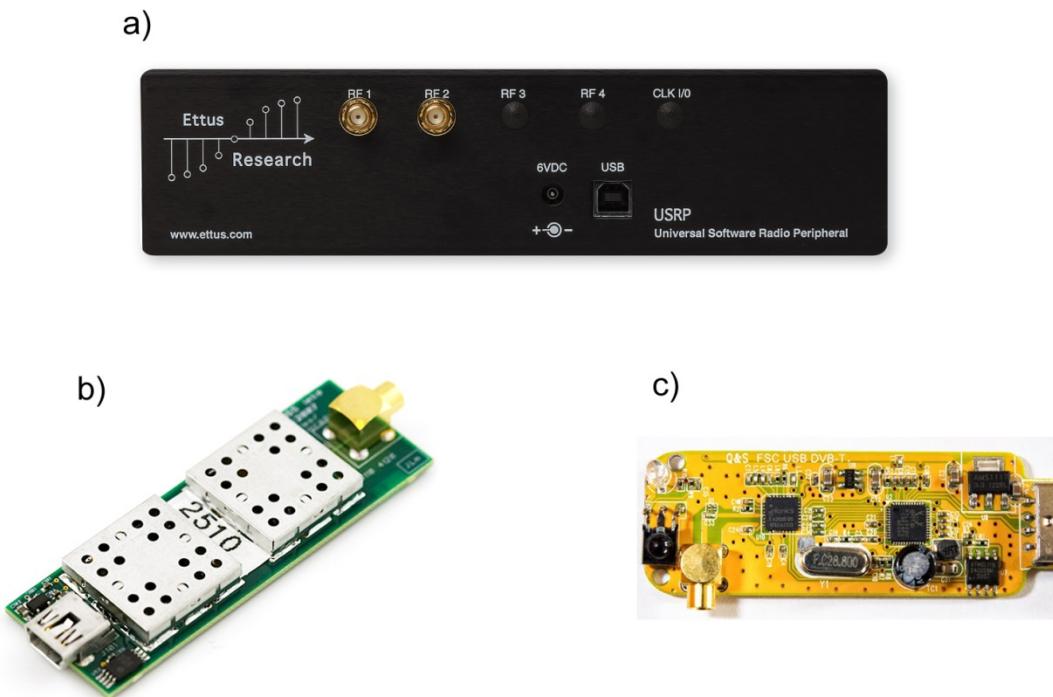


Figura 2-3: Cabezales de radiofrecuencia utilizados con éxito en el software GNSS-SDR. a) USRP V1, b) SiGe GN3S Sampler v2, y c) receptor DVB-T basado en el chipset Realtek RTL2832U.

En resumen, hemos identificado dos requisitos principales para el receptor software en cuanto a las fuentes de señal:

- Debe permitir el procesado en tiempo real (cuando sea posible) así como el funcionamiento *offline* (sin conexión).
- Debe ser capaz de utilizar una gran variedad de fuentes de señal (tanto desde archivos, como desde diversos cabezales de radiofrecuencia).

2.4.1.2 Salidas

La mayoría del software geodésico para procesado de datos GNSS utiliza un conjunto bien definido de observables:

- medidas de código, los pseudorangos, que se obtienen de la diferencia entre el tiempo de recepción (expresado en el marco de tiempo del receptor) y el tiempo de transmisión (expresado en el marco de tiempo del satélite) de la señal de un satélite en concreto,
- medidas de la fase de la portadora (que en realidad son medidas comparativas entre la portadora de la señal recibida del satélite y una frecuencia de referencia generada por el receptor), y

- el tiempo de observación, que es la lectura del reloj del receptor en el momento de la validación de la fase de la portadora y/o las mediciones del código.

El formato RINEX (*Receiver Independent Exchange Format*) es un formato de intercambio de datos para sistemas de navegación por satélite, que incluye tanto los observables como la información contenida en el mensaje de navegación emitido por los satélites. La versión más común en la actualidad es la 2.10, que permite el almacenamiento de las medidas de pseudorango, fase de la portadora y Doppler para los sistemas GPS o GLONASS, junto con los datos de sistemas de aumentación basados en satélites (SBAS) EGNOS y WAAS, simultáneamente. La necesidad de mejorar el tratamiento de los archivos de datos en el caso de los ficheros que contienen datos de *tracking* de más de un sistema de satélites, cada uno con diferentes tipos de observables, da lugar a importantes modificaciones en la estructura de dichos datos. En el momento de la escritura de este proyecto, la nueva versión disponible es la 3.01 [2-9], que incluye formatos definidos para las nuevas señales GPS, GLONASS, Galileo y SBAS. Por lo tanto, los ficheros RINEX, tanto de observables como de navegación son uno de los formatos de salida que se proponen para el receptor GNSS-SDR.

Los receptores comerciales por lo general no permiten el acceso a la información intermedia, sino que ofrecen acceso directo a la solución de posición, velocidad y tiempo (PVT) y a los datos del almanaque. NMEA (*National Marine Electronics Association*) 0183 [2-10] es el formato de salida estándar para los receptores GPS comerciales, y hay una gran cantidad de software compatible con capacidad para el tratamiento y visualización de dicha información. GNSS-SDR proporciona este formato de salida para el bloque PVT.

Por último, otro formato interesante para exportar los datos geográficos es KML (*Keyhole Markup Language*), directamente importable por aplicaciones software como Google Earth, Google Maps, y Google Maps para móviles. KML es un estándar abierto llamado oficialmente OpenGIS KML Encoding Standard (OGC KML) [2-11], y es mantenido por el consorcio Open Geospatial Consortium Inc. (OGC).

En resumen, hemos encontrado útiles los siguientes formatos de salida de datos:

- RINEX.
- NMEA.
- KML.

2.4.2 Arquitectura del receptor

La naturaleza intrínseca de un receptor GNSS impone algunos requisitos en el diseño de la arquitectura del software receptor. Dado que la composición de las señales GNSS recibidas cambia con el tiempo (ya que en un principio, sólo algunos satélites son visibles, y conforme avanza el tiempo, algunos de estos satélites dejan de ser visibles y por el contrario aparecen otros nuevos), algunos canales perderán el seguimiento de sus señales mientras que por otro lado se deberán instanciar canales

nuevos para procesar las señales que aparezcan. Esto significa que el receptor debe ser capaz de activar y desactivar los canales de forma dinámica, y además necesita detectar estos cambios en tiempo de ejecución. Además, existe la necesidad de diseñar el mecanismo de comunicación entre los bloques del Plano de Procesado de Señal y el Plano de Control.

Por otro lado, la arquitectura del software debe lidiar con algunos criterios de diseño que a veces pueden ser antitéticos, tales como la flexibilidad frente a la robustez, o la eficiencia contra la portabilidad. Con el objetivo en mente de alcanzar el procesado en tiempo real, la eficiencia debe ser la característica prioritaria en aquellos bloques que trabajan a altas velocidades (principalmente, el acondicionamiento de señal, la supresión del Doppler, y la correlación), mientras que otros bloques que trabajan a velocidad media (el seguimiento y extracción de parámetros de navegación) o baja (generación de medidas y la solución de navegación) se pueden implementar basándose en la robustez y la fiabilidad.

Con el objetivo de satisfacer todas estas necesidades se explican a continuación algunos aspectos básicos en el diseño del receptor como son: el lenguaje de programación elegido (C++), el uso del *framework* de GNU Radio para el manejo del flujo de muestras entre los bloques del Plano de Procesado de Señal y el uso de diversos patrones de diseño con el fin de utilizar soluciones bien diseñadas y altamente testeadas a problemas comunes, evitando así lo que comúnmente se conoce como la “reinvención de la rueda”.

2.4.2.1 Lenguaje de programación: C++

El lenguaje de programación elegido es C++. La razón de esta elección no sólo se basa en el hecho de que C++ sea un lenguaje dominante, aunque gracias a ello es mucho más fácil reclutar a programadores experimentados y además dispone de una serie de librerías optimizadas así como un amplio abanico de compiladores avanzados. Otra razón de peso es el hecho de que a pesar de que permite la programación a bajo nivel (*close to the metal programming*), respondiendo así al criterio de eficiencia, C++ también añade capas de abstracción que hacen posible el uso de plantillas o *templates* (funciones y clases parametrizables que permiten trabajar con tipos de datos abstractos, especificándose a posteriori qué tipos de datos son los que se quieren usar cuando sean necesarios). Las plantillas son utilizadas por el compilador para generar código fuente temporal, que es fusionado y compilado con el resto del código fuente. Este mecanismo, conocido como polimorfismo estático, además de ser una forma de pre-evaluación de parte del código en tiempo de compilación en lugar de en tiempo de ejecución, también repercute en un código máquina más optimizado, ejecutables más pequeños, menores tiempos de ejecución, y en menores requisitos de memoria, evitando la sobrecarga que provoca el polimorfismo en tiempo de ejecución. Por otra parte, una nueva versión del lenguaje, llamado C++11, ha sido recientemente aprobado como estándar ISO (Organización Internacional de Estandarización) [2-10]. De esta manera se asegura que en las próximas décadas, los programas diseñados conforme al standard C++ actual podrán ser ejecutados con modificaciones mínimas, tal y como sucede hoy en día con programas antiguos escritos en el mismo lenguaje. Además, también asegura la portabilidad, así como la disponibilidad de compiladores compatibles. Esta nueva versión proporciona mejoras para la escritura de código concurrente (por ejemplo, para procesadores multinúcleo) a través de tipos de datos seguros, punteros inteligentes (*smart pointers*), nuevas formas de manipulación de objetos optimizadas en cuanto a memoria, y una gran cantidad de nuevas características tanto en el núcleo del

lenguaje como en las librerías, que son muy adecuadas para las aplicaciones de software radio.

2.4.2.2 GNURadio

La arquitectura del Plano de Procesado de Señal se basa principalmente en GNU Radio [2-13], un marco (*framework*) bien establecido que proporciona la gestión en tiempo de ejecución del procesado de señales así como bloques de procesado adecuados para implementar aplicaciones de software radio. Los *frameworks* son un caso especial de librerías de software, son abstracciones de código reutilizables encapsuladas en una API bien definida, sin embargo, contienen algunas de las características clave que los distinguen de las librerías normales: el flujo general de control del programa no está determinado por la aplicación que llama al *framework*, sino por el mismo *framework*; y puede ser ampliado por el usuario añadiendo código que proporcione una funcionalidad específica. Los *frameworks* facilitan el desarrollo de software permitiendo así a los diseñadores y programadores dedicar su tiempo a cumplir con los requisitos del software en lugar de preocuparse de detalles de más bajo nivel del estándar, reduciendo así el tiempo de desarrollo.

La jerarquía de clases de GNU Radio impone una arquitectura de procesos concurrentes (*threads*) por bloque que permite una asignación estratégica de cada procesador a cada proceso de forma automática en procesadores multinúcleo (*scheduling* de procesos automático), ocultando toda la complejidad detrás de una API simple y robusta. Se utiliza la memoria compartida para gestionar de manera eficiente el flujo de datos entre los bloques, y ofrece un gran conjunto de bloques bien programados que proporcionan implementaciones para las tareas más comunes de procesado de señal. En cambio, GNU Radio no proporciona ningún método estándar de mecanismo de control sobre los bloques.

El usuario puede construir un receptor mediante la creación de un diagrama de flujo (*flowgraph*) donde los nodos son los bloques de procesado de señal y las líneas representan el flujo de datos entre ellos. Conceptualmente, los bloques procesan flujos infinitos de datos que fluyen de sus puertos de entrada hacia sus puertos de salida. Los atributos de los bloques incluyen el número de puertos de entrada y salida que tienen, así como el tipo de datos que fluyen a través de cada uno de ellos. Una vez que se conectan y forman un diagrama de flujo (*flowgraph*), la aplicación se puede ejecutar y los datos se colocan en el flujo. Mientras haya datos que procesar, los procesos concurrentes del programa ejecutan el código de los diferentes bloques.

2.4.2.3 Patrones de diseño

El concepto de patrón de diseño de software se introdujo por primera vez en [2-14], un libro que rápidamente se convirtió en una referencia fundamental sobre el tema. Los patrones de diseño son descripciones de soluciones a problemas de software comunes que surgen en contextos diferentes, capturando estructuras recurrentes en todos ellos y la dinámica entre los diferentes participantes del software con el fin de facilitar la reutilización de diseños exitosos y altamente testeados. Seguir un patrón ayuda a resolver los conflictos que se producen a veces entre aspectos principales del diseño como son la flexibilidad, la extensibilidad, la fiabilidad, la escalabilidad y la eficiencia. No son recetas de código, pero dan soluciones generalizadas a los problemas que ocurren comúnmente, mostrando relaciones e interacciones entre clases y/u objetos pero sin especificar la instanciación final en una aplicación en concreto.

A continuación se proporcionan descripciones de algunos patrones de diseño utilizados en el receptor GNSS-SDR.

Un principio clave del diseño reutilizable orientado a objetos es el siguiente: “Programa para una interfaz, no para una implementación” [2-14]. Este principio es aplicable sobre todo a las relaciones de dependencia que tienen que ser utilizadas cuidadosamente en una aplicación compleja. Es fácil agregar una dependencia a una clase, sin embargo, el caso inverso no es tan sencillo y librarse de una dependencia no deseada puede convertirse en un trabajo de refactorización complicado o, peor aún, en un impedimento para que el usuario vuelva a utilizar el código en otro contexto. Aislar la interfaz de la aplicación provoca que la implementación pueda variar, y es una relación de dependencia saludable. Este enfoque no sólo ofrece flexibilidad al desarrollador, sino que también separa la parte más valiosa, el diseño, de la aplicación, y permite a los clientes desvincularse de la implementación. De hecho, una clase abstracta proporciona una mayor flexibilidad sobre todo cuando la clase evoluciona. De esta forma se pueden agregar nuevos comportamientos y funcionalidades sin afectar a los clientes. En GNSS-SDR, todos los bloques del plano de procesado de señal constan de una interfaz y una implementación.

Otra característica relevante de un receptor GNSS es que algunos bloques de procesado de señal procesan el flujo de datos de entrada, aplicando en paralelo el mismo conjunto de operaciones, tales como la adquisición o el *tracking* de la señal de los diferentes satélites. Por lo tanto, es deseable una estructura que mejore la capacidad del receptor (es decir, el número de satélites que se pueden seguir simultáneamente) mediante la réplica de unidades arquitectónicas, facilitando así un procesado de datos en paralelo eficiente. La solución sigue el patrón de diseño Arquitectura de Canal (*Channel Arquitecture pattern*) [2-15], que agrupa todos las funciones de procesado de señal relacionadas con un solo satélite en un subsistema canal. Un canal puede ser considerado como un tubo que transforma los datos de entrada de forma secuencial en datos de salida, cambiando en algunos casos la velocidad de dichos datos. El patrón de Arquitectura de Canal se adapta bien a la transformación secuencial de datos de un estado o forma a otro diferente. Esto simplifica los algoritmos, que se pueden descomponer fácilmente en una serie de pasos que operan sobre elementos aislados de la secuencia de datos. Además, se pueden añadir varias instancias del subsistema canal y así ampliar el número de satélites procesados. De esta forma la arquitectura se puede adaptar fácilmente para manejar múltiples elementos del flujo de datos en paralelo, incluso aunque estén en diferentes etapas de procesado.

Como se ha comentado con anterioridad, existe la necesidad de diseñar el mecanismo de comunicación entre los bloques del Plano de Procesado de Señal y el Plano de Control ya que GNURadio no proporciona ningún método estándar de mecanismo de control sobre los bloques. El patrón de diseño Cola de Mensajes (*Message Queueing pattern*) [2-15] es muy flexible y puede adaptarse a casi cualquier método de control que se necesite implementar en el receptor. En este sentido existe un proceso de control (*control thread*) que se ejecuta en paralelo al diagrama de flujo del receptor (*flowgraph*), y que es el encargado de recibir las notificaciones que desencadenan cambios en la aplicación. Algunas de estas notificaciones se envían directamente desde los bloques de procesado. Por ejemplo, un bloque de adquisición, que termina su ejecución y detecta la señal de un satélite, enviará una notificación que indica su éxito al *control thread* a través de la cola de mensajes. El *control thread*, entonces cambiará la configuración interna del canal y pasará los resultados del proceso de adquisición (es decir, el identificador del satélite detectado, y las

estimaciones aproximadas del retardo de código y la frecuencia Doppler) al bloque de *tracking*. Dado que GNSS-SDR es una aplicación multiproceso, el *control thread* recibirá mensajes de muchas fuentes, probablemente, al mismo tiempo. El mecanismo para enviar y leer estos mensajes debe garantizar que sólo un proceso lee o escribe los datos compartidos a la vez, esta condición es conocida como seguridad en procesos que se ejecutan de forma concurrente (*thread-safety*). Así pues, el *control thread* implementa una cola de mensajes concurrente (*thread-safe*), y comparte una instancia de esta cola con todos los módulos que deban enviar mensajes.

Por último, una de las características más atractivas de un receptor software es la posibilidad de intercambiar algoritmos (por ejemplo, distintas implementaciones para la adquisición de la señal y el *tracking*) y observar su impacto en todo el sistema, o establecer comparaciones entre dichos algoritmos. Además, estos algoritmos se deberían poder seleccionar en tiempo de ejecución. Este tipo de problema puede ser resuelto mediante el patrón Estrategia (*Strategy pattern*) [2-14], que define una familia de algoritmos, los encapsula, y los hace intercambiables, permitiendo que los algoritmos varíen independientemente de los clientes que los utilizan. La complejidad en la instanciación de objetos (por ejemplo, implementaciones concretas de la adquisición y el *tracking*) cuando sólo se conocen sus interfaces abstractas se puede abordar a través del patrón Método de Fábrica (*Factory Method*) [2-14]. Este patrón consiste en la encapsulación de los procesos involucrados en la creación de objetos, definiendo una interfaz para crear un objeto, pero dejando que las subclases decidan qué clase instanciar. El patrón Método de Fábrica permite a una clase diferir la instanciación a sus subclases. Este enfoque elimina la necesidad de ligar al código clases específicas de la aplicación, proporciona enlaces a las subclases (lo que hace más flexible la creación de objetos dentro de una clase a través del método de fábrica que creando directamente el objeto), y conecta jerarquías de clases paralelas (localizando qué clases se pueden conectar conjuntamente).

En resumen, en este apartado se han identificado los requisitos y características en términos de arquitectura de software:

- Consideración de C++ como un lenguaje de programación conveniente para la implementación de GNSS-SDR.
- El receptor software debe permitir la parallelización a través de múltiples procesos concurrentes (*threads*).
- Se debe realizar la reconfiguración en tiempo de ejecución.
- Se requiere un mecanismo concurrente de comunicación entre los módulos de procesado, para tal fin se ha escogido el *framework* de GNU Radio.
- El desacoplamiento entre las interfaces y las implementaciones es crucial para una implementación reutilizable.
- Es deseable poder agregar nuevos algoritmos y mejorar las implementaciones existentes sin romper nada.

2.4.3 Validación (testeo) de los bloques

Es un hecho ampliamente aceptado que una metodología basada en el testeo ofrece valiosos beneficios para el desarrollo del software: facilita los cambios en el software, simplifica la integración, automatiza la documentación, ayuda a separar la interfaz de la aplicación, aumenta la productividad de los desarrolladores, y desempeña un papel central en el proceso de garantía de calidad del software. En esta metodología, los test se escriben antes que la función que se quiere probar, contrariamente a lo que se suele hacer en un enfoque basado en las características del software a diseñar. Esta metodología ofrece dos ventajas principales: ayuda a garantizar que la aplicación a desarrollar posteriormente esté preparada para ser testeada, ya que los desarrolladores deben considerar la forma de testear la aplicación desde el principio, en lugar de preocuparse de ello más tarde, y también asegura que se escriban los test para cada función, al contrario de lo que suele ocurrir cuando se planifica que dichos test sean escritos a posteriori, ya que muchas veces se acaban omitiendo por falta de tiempo.

Hay diferentes niveles de test, desde la simple comprobación de la funcionalidad de una sección específica de código hasta la verificación de que la aplicación en su totalidad se integra bien con sistemas de terceros. Podemos establecer la siguiente clasificación:

- **Test unitarios (*unit testing*):** Entendiendo una unidad como la parte más pequeña comprobable de una aplicación, un test unitario es un trozo de código desarrollado con el único objetivo de verificar que una rutina o función de nuestro software está funcionando según esperamos. En definitiva, lo que un test unitario pretende es tener trozos de código que se encargarán de testear por separado y de forma independiente cada uno de los métodos de las distintas clases que compongan el programa desarrollado, para que cuando se haga alguna modificación en el código, posibles errores derivados de esa modificación puedan ser identificados y corregidos de manera rápida y eficaz. Se componen de tres fases: i) llevar a cabo la planificación de los test (planificar el enfoque general, los recursos y el esquema del test; determinar las características que quieren ser comprobadas; y perfeccionar el plan general), ii) obtener el conjunto de test (diseñar el conjunto de test; implementar el plan perfeccionado), y iii) evaluar el test unitario (ejecutar los procedimientos del test; comprobar su finalización, y evaluar el resultado del test).
- **Test de integración (*integration testing*):** Fase del testeo del software en la que los módulos individuales se combinan y se testean como un grupo, después de los test unitarios y antes de los test de sistema. Un test de integración toma como entrada los módulos que han superado los test unitarios, los agrupa en grandes agregados, les aplica test definidos en un plan de test de integración, y proporciona como salida el sistema integrado listo para los test de sistema.
- **Test de sistema (*system testing*):** son test que deben llevarse a cabo en un sistema completo e integrado para evaluar el nivel del cumplimiento de sus requisitos específicos. Estos test no deben exigir ningún conocimiento del diseño interior del código o de la lógica utilizada, e incluyen pruebas de rendimiento utilizando herramientas de *profiling* (análisis de rendimiento) y

logging (registro de actividad en sistemas) de vital importancia en aplicaciones específicas cuyo objetivo es trabajar en tiempo real.

- **Test de alto nivel (*higher-level testing*):** Los comentarios de usuarios y la revisión colectiva son herramientas poderosas en la producción de software de alta calidad. Liberar el código fuente bajo la licencia GPL permite la inspección no sólo de las características del software, sino también de cómo se han implementado.

Para los test unitarios y de integración de GNSS-SDR se utiliza el marco de test de C++ de Google (*Google C++ Testing Framework*) [2-16], una librería que se ocupa de toda la infraestructura de los test, dejando que el desarrollador se centre en el contenido de los mismos.

Con el fin de verificar el comportamiento del código a testear, Google Test permite escribir aserciones (*assertions*), que son declaraciones que comprueban si una condición es verdadera. El resultado de una *assertion* puede ser éxito (*success*), error no fatal (*nonfatal error*) o error fatal (*fatal error*). Si se produce un error fatal, se aborta la ejecución del test en curso, de lo contrario el programa continúa ejecutándose normalmente. Si un test falla o tiene una aserción fallida, entonces el resultado global del test es error, de lo contrario el resultado es éxito. Las aserciones de Google Test son macros que se asemejan a las llamadas a funciones: la forma de testear una clase o función es realizar aserciones sobre su comportamiento. Cuando una aserción falla, Google Test imprime el número de línea del archivo de la aserción que ha fallado, junto con un mensaje de error. Este mensaje de error puede ser personalizado por el autor del test.

Un caso de prueba (*test case*) contiene uno o varios test. Los test normalmente se agrupan en los casos de prueba que reflejan la estructura del código del test. Cuando hay que realizar múltiples test en un caso de prueba que comparten objetos y subrutinas comunes, se pueden poner en una clase denominada accesorio de prueba (*test fixture*). El propósito de un accesorio de prueba es asegurar que hay un entorno bien conocido y fijo en el que se ejecutan los test de modo que los resultados son repetibles.

En el desarrollo de este proyecto el autor ha realizado más de 40 test diferentes cuyos resultados pueden observarse al ejecutar el archivo */install/runtests*.

En primer lugar se han realizado test de instanciación para todos las implementaciones de los bloques de GNSS-SDR, dichos test consisten en comprobar que cuando se instancia cada bloque del software, la clase Método de Fábrica (*GNSSBlockFactory*) devuelve una interfaz del bloque con la implementación que se especifica en el fichero de configuración para tal efecto. Dichos test (un total de 24) se encuentran en el archivo */src/tests/gnss_block/gnss_block_factory_test.cc*.

Además, se han implementado test unitarios para los bloques de remuestreo, adquisición y *tracking* diseñados durante la realización de este proyecto. Por último se han realizado test de integración y de sistema de los bloques mencionados validando el funcionamiento del sistema completo con la inclusión de dichos bloques en el sistema global, y se han obtenido resultados del correcto funcionamiento a través de otras herramientas tales como Matlab. Estos test se explican con más detalles en los capítulos siguientes dedicados a cada uno de los bloques diseñados.

2.4.4 Otras características

- **Desarrollo del ecosistema.** La infraestructura para la gestión de proyectos, el desarrollo de código y una comunicación eficaz entre los usuarios y desarrolladores constituyen un aspecto clave en los proyectos de software. El sitio web debe estar bien diseñado en términos de usabilidad, funcionalidad y capacidad de ampliación, que garantice al usuario una experiencia agradable y atractiva. Para ello se ha utilizado Drupal [2-17] como sistema de gestión de contenidos. El resultado se puede encontrar en <http://gnss-sdr.org>. Para la gestión del código, es esencial una aplicación que automatice el proceso de mantener el historial de anotaciones del proyecto, y que permita la reversión de los cambios de código, el control de cambios, y el seguimiento de errores. Se ha utilizado el servicio proporcionado por SourceForge [2-18], que permite el acceso al código usando dos sistemas principales de control de versiones como Subversion [2-19] y git [2-20], junto con una lista de distribución que facilita la comunicación entre los usuarios y desarrolladores.
- **Documentación.** Este punto es de suma importancia para los usuarios, desarrolladores, testers, arquitectos de software y estudiantes. Para escribir la documentación del software se ha utilizado Doxygen [2-21], una herramienta de referencia en este campo. La documentación está escrita en el código C++, en forma de comentarios, por lo que es relativamente fácil de generar (ya que puede ser escrita junto al código fuente) y actualizar. Doxygen escanea el código, extrae la documentación y la vuelca en formato HTML, LaTeX, RTF o XML, enlazando la documentación y el código, de modo que el lector de un documento puede explorar el código actual. También genera automáticamente gráficos de dependencia, diagramas de herencia, y diagramas de colaboración. Además de la documentación del código, la página web del proyecto ofrece instrucciones detalladas sobre la instalación, uso, el tipo de codificación e información general sobre el programa.
- **Herramientas de construcción de ejecutables (*Building tools*).** El proceso de construcción de ejecutables debe ser de fácil mantenimiento y portable a diferentes sistemas operativos y arquitecturas hardware. Si no está bien pensado, este proceso puede provocar que, en un futuro, se tenga que dedicar más tiempo a realizar ajustes del sistema de compilación en lugar de lo verdaderamente importante, que es el desarrollo del código fuente. Con este propósito se ha utilizado CMake [2-22], una herramienta que se encarga de compilar las fuentes con las opciones adecuadas, creando bibliotecas estáticas y compartidas, y construyendo ejecutables, usando variedad de compiladores en los sistemas operativos más comunes. Además CMake proporciona muchas funciones integradas que pueden ser combinados para producir configuraciones de construcción arbitrarias, tales como versiones de depuración (*debug*) y liberación (*release*) o variantes de un solo *thread* o *multithread*, traduciendo de forma automática las propiedades solicitadas en parámetros (*flags*) de la línea de comandos para invocar los componentes de los *toolsets* apropiados como compiladores y enlazadores (*linkers*). En cuanto a la compilación, se ha utilizado la colección de compiladores de GNU (GCC) [2-23]. Comentar también que CMake es un sistema de construcción de

ejecutables que ha ido ganando impulso en los últimos años, convirtiéndose en el estándar de facto para las herramientas de construcción de ejecutables. CMake proporciona muy buenas características (como la posibilidad de descargar automáticamente las dependencias y un fácil manejo de los bloques opcionales), y está disponible en un gran número de plataformas.

- **Registro (*Logging*).** El *logging* es de suma importancia para realizar la depuración y seguimiento del software diseñado. Además, permite al desarrollador realizar estadísticas interesantes a partir de los registros obtenidos en la ejecución, y obtener patrones de uso. En GNSS-SDR, las funciones de *logging* están a cargo de Google Logging Library (*google-glog*) [2-24], una biblioteca que implementa el *logging* a nivel de aplicación. Proporciona APIs simples pero poderosas para diversos eventos de logging en el programa. Los mensajes se pueden registrar según varios niveles, y los usuarios pueden controlar el comportamiento del *logging* desde la línea de comandos, pudiendo establecer condiciones, abortar el programa de *logging* cuando las condiciones previstas no se cumplan, e introducir sus propios niveles detallados de *logging*.
- **Análisis del rendimiento (*Profiling*).** Para el análisis del rendimiento se ha utilizado Google Performance Tools (*google-perf-tools*) [2-25], una colección de asignadores de memoria *multithread* de alto rendimiento, además de algunas herramientas de análisis de rendimiento, que permiten automatizar el proceso de *profiling*, identificar cuellos de botella computacional, y ayudar a los desarrolladores a focalizar sus esfuerzos de optimización.

2.5 Detalles de la implementación

El proceso principal de GNSS-SDR inicializa la librería de *logging*, procesa los *flags* de la línea de comandos, proporcionados por el usuario y crea una instancia de un objeto *ControlThread*. El constructor de dicho objeto lee el fichero de configuración, crea una cola de control y crea un *flowgraph* acorde con la configuración. Entonces, el proceso principal del programa llama al método *run()* del objeto instanciado, este método conecta el *flowgraph* y empieza a ejecutarlo. A partir de este momento, hasta que se recibe un mensaje de detención, lee los mensajes de control enviados por los módulos del receptor a través de una cola concurrente y los procesa. Por último, cuando se recibe un mensaje de detención, el proceso principal ejecuta el destructor del objeto *ControlThread*, que libera la memoria, y realiza las tareas necesarias para salir limpiamente del programa.

Como se muestra en la Figura 2-2, el receptor de software se divide en un Plano de Control y un Plano de Procesado de Señal. En las siguientes secciones se proporcionan detalles acerca de su implementación.

2.5.1 Plano de Control

El Plano de Control está a cargo de la creación del *flowgraph* de acuerdo a lo estipulado en el fichero de configuración, así como de la gestión de los módulos. La configuración permite a los usuarios definir de una manera sencilla un receptor personalizado especificando el tipo de *flowgraph* (tipo de fuente de señal, el número

de canales, los algoritmos que se utilizarán para cada canal y cada módulo, las estrategias para la selección de los satélites, el formato de salida, etc.). Puesto que es difícil prever, en términos de configuración, cómo serán las implementaciones de los módulos en el futuro, se ha utilizado un método muy simple que se puede ampliar sin que esto provoque un impacto importante en el código. La forma de conseguirlo consiste simplemente en mapear los nombres de las variables de los módulos con los nombres de los parámetros de configuración.

Las propiedades se pasan entre los bloques del programa usando la clase *ConfigurationInterface*. Hay dos implementaciones de esta interfaz: *FileConfiguration* e *InMemoryConfiguration*. *FileConfiguration* lee las propiedades (que son parejas compuestas por el nombre de la propiedad y su valor) de un archivo y las almacena internamente. *InMemoryConfiguration* no lee desde archivo, sino que permanece vacía después de su instanciación y asigna valores y nombres a las propiedades mediante el método *set_property*. *FileConfiguration* está destinado a ser utilizado en la aplicación GNSS-SDR, mientras que *InMemoryConfiguration* está destinado a ser utilizado en los tests para evitar la dependencia de ficheros en el sistema de archivos. Las clases que necesiten leer parámetros de configuración reciben instancias de *ConfigurationInterface* de donde obtendrán los valores de dichas propiedades. A modo de ejemplo, los parámetros relacionados para el bloque *SignalSource* deberían tener este aspecto:

```
SignalSource.parameter1=value1
```

```
SignalSource.parameter2=value2
```

El nombre de estos parámetros (*parameter1*, *parameter2*, ...) puede ser cualquier cosa pero hay una palabra reservada: *implementation*. Este parámetro indica en su valor el nombre de la clase que tiene que ser instanciada por el Método de Fábrica para desempeñar ese papel. Por ejemplo, si queremos usar la implementación *File_Signal_Source* para el módulo *SignalSource*, la línea correspondiente en el fichero de configuración sería:

```
SignalSource.implementation= File_Signal_Source
```

Ya que la configuración es sólo un conjunto de nombres de propiedades y valores sin significado o sintaxis, el sistema es muy versátil y de fácil ampliación. La adición de nuevas propiedades al sistema sólo implica modificaciones en las clases que hacen uso de estas propiedades. Además, los archivos de configuración no se comparan con una sintaxis estricta por lo que siempre se encuentran en un estado correcto (siempre y cuando contenga parejas de nombres de propiedades y valores en formato INI¹).

¹ Un archivo INI es un archivo de texto de 8-bits en el que cada propiedad tiene un nombre y un valor, de la forma *nombre = valor*. Se distinguen entre mayúsculas y minúsculas, y no pueden utilizar espacios. El punto y coma (;) indica el comienzo de un comentario, todo lo que hay entre el punto y coma y el final de la línea se ignora.

De hecho, la aplicación define una clase de acceso sencilla para buscar las parejas de valores de configuración y se los pasa a una clase que sigue el patrón Método de Fábrica (*Factory Method*) llamada *GNSSBlockFactory*. Este factory decide, de acuerdo con los parámetros de configuración, qué clase tiene que ser instanciada y los parámetros que se le pasarán al constructor. De esta forma, el *factory* encapsula la complejidad de la instanciación de los bloques. Con este enfoque, el hecho de añadir un nuevo bloque que requiere nuevos parámetros será tan simple como añadir la clase del bloque y modificar el *factory* para que sea capaz de instanciarla. Este bajo acoplamiento entre las implementaciones de los bloques y la sintaxis de la configuración permite extender sobremanera las capacidades de la aplicación. También permite producir receptores totalmente personalizados, por ejemplo, un banco de pruebas para algoritmos de adquisición, así como colocar observadores en cualquier punto de la cadena del receptor.

La clase *GNSSFlowgraph* es la responsable de preparar el diagrama de bloques de acuerdo con la configuración, ejecutarlo, modificarlo en tiempo de ejecución y detenerlo. Los bloques se identifican por su rol. Esta clase sabe qué roles tiene instanciar y la forma de conectarlos para configurar el diagrama genérico que se muestra en la Figura 2-2. Se basa en la configuración para obtener las instancias correctas de las funciones que necesita y entonces aplica las conexiones entre los bloques de GNURadio para que el diagrama esté listo para ser iniciado. La complejidad relacionada con la gestión de los bloques y el flujo de datos es manejada por la clase de GNURadio *gr_top_block* [2-26]. *GNSSFlowgraph* encapsula la instancia del *gr_top_block* para poder aprovechar mejor las prestaciones de *GNSSBlockFactory*, del sistema de configuración y de los bloques de procesado. Esta clase es también responsable de la aplicación de los cambios en la configuración del *flowgraph* durante el tiempo de ejecución, ya que reconfigura dinámicamente los canales, pudiendo escoger la estrategia para la selección de los satélites. Esta estrategia puede ir desde una búsqueda secuencial a través de todos los identificadores (ID) de los satélites a planteamientos mucho más inteligentes que determinen cuáles son los satélites que más probablemente estén a la vista a partir de estimaciones aproximadas de la posición del receptor con el fin de evitar la búsqueda de satélites que se encuentran en el otro lado de la Tierra.

Esta clase codifica internamente acciones a realizar en el diagrama. Estas acciones se identifican por números enteros sencillos. *GNSSFlowgraph* ofrece un método que recibe un entero que codifica una acción, y este método provoca la acción representada por el número entero. Las acciones pueden ir desde cambiar las variables internas de los bloques a modificar completamente el diagrama construido mediante la activación/desactivación de bloques. El número y la complejidad de las acciones están tan sólo limitados por el número de enteros disponible para realizar la codificación. Este enfoque encapsula la complejidad de la preparación de un diagrama completo con todos los bloques necesarios instanciados y conectados. También se hace un buen uso del sistema de configuración y de *GNSSBlockFactory*, que mantiene el código limpio y fácil de entender. Y además permite actualizar el conjunto de acciones que deben realizarse en el diagrama con bastante facilidad.

La clase *ControlThread* es la responsable de instanciar a *GNSSFlowgraph* y pasarle la configuración necesaria. Una vez que se define el *flowgraph* y se conectan sus bloques, éste empieza a procesar el flujo de datos entrantes. El objeto *ControlThread* pasa a ser en ese momento el encargado de leer la cola de control y de procesar todos los mensajes enviados por los bloques de procesado a través de la cola de control concurrente.

2.5.2 Plano de Procesado de Señal

Un aspecto clave en el diseño de software orientado a objetos es la jerarquía de clases, tal y como se muestra en la Figura 2-4. La notación utilizada es la siguiente: se utiliza una versión muy simplificada del Lenguaje de Modelado Unificado (UML), un lenguaje de modelado estandarizado de propósito general en el campo de la ingeniería del software orientado a objetos. En este proyecto, las clases se describen como rectángulos con dos secciones: la parte superior incluye el nombre de la clase, y la parte inferior los métodos de la clase. Una flecha punteada desde la ClaseA a otra ClaseB representa una relación de dependencia. Esta relación significa simplemente que la ClaseA de alguna forma depende de la ClaseB. En C++ esto casi siempre se materializa en un `# include`. La herencia permite modelar las relaciones "es una" y "es como", lo que permite reutilizar fácilmente tanto el código como los datos existentes. Cuando una ClaseA hereda de otra ClaseB, se dice que la ClaseA es una subclase (o clase hija) de la ClaseB, y que la ClaseB es la superclase (o clase padre) de la ClaseA. La notación de modelado UML para la herencia es una línea con una punta de flecha que apunta desde la subclase a la superclase.

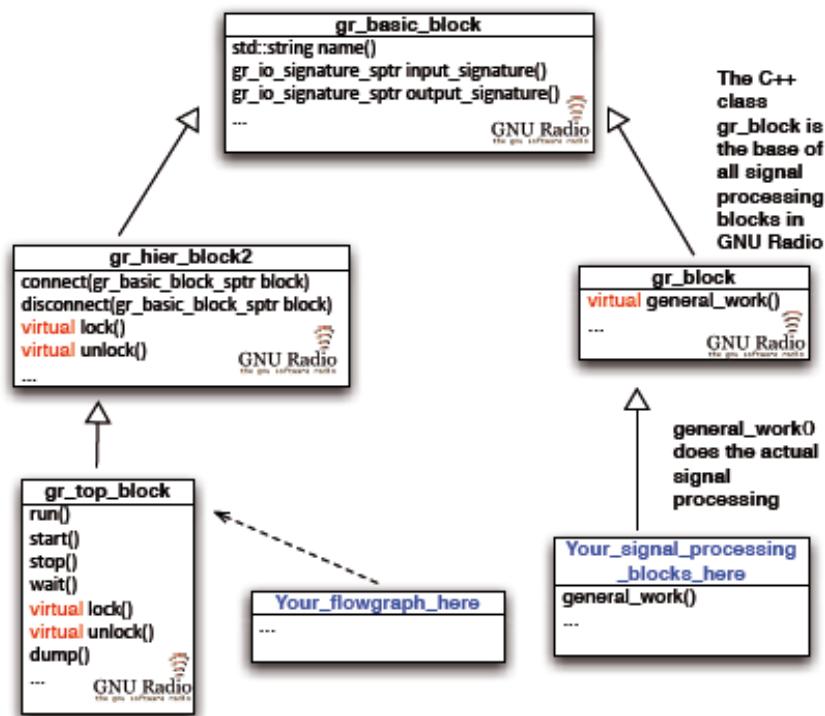


Figura 2-4: Jerarquía de clases para el Plano de Procesado de Señal.

Como se muestra en la Figura 2-4, `gr_basic_block` [2-27] es la clase base abstracta de la que heredan todos los bloques de procesado de señal, es una abstracción simple de una entidad que tiene un nombre y un conjunto de entradas y salidas. Nunca se instancia directamente, sino que ésta es la clase padre abstracta tanto de `gr_hier_block2` [2-28], que es un contenedor recursivo que agrega o quita bloques jerárquicos o de procesado del diagrama interno, como de `gr_block` [2-29],

que es la clase base abstracta para todos los bloques de procesado. Un flujo de procesado de señal se construye mediante la creación de un árbol de bloques jerárquicos, que en todos los niveles puede contener nodos terminales que en la práctica implementan las aplicaciones de procesado de señal.

La clase *gr_top_block* es el bloque jerárquico de mayor alto nivel que representa un *flowgraph*. En él se definen las funciones de GNU Radio que se utilizan durante la ejecución del programa: *run ()*, *start ()*, *stop ()*, *wait ()*, etc.

Tal y como se muestra en la Figura 2-5, la subclase llamada *GNSSBlockInterface* es la interfaz común para todos los módulos de GNSS-SDR. En él se definen los métodos virtuales puros, que deben ser implementados por las clases derivadas. Las clases que contienen métodos virtuales puros se denominan clases abstractas y no pueden ser instanciadas directamente. Además, una subclase de una clase abstracta sólo puede ser instanciada directamente si todos los métodos virtuales puros heredados han sido implementados por esa clase o una clase padre de la misma.

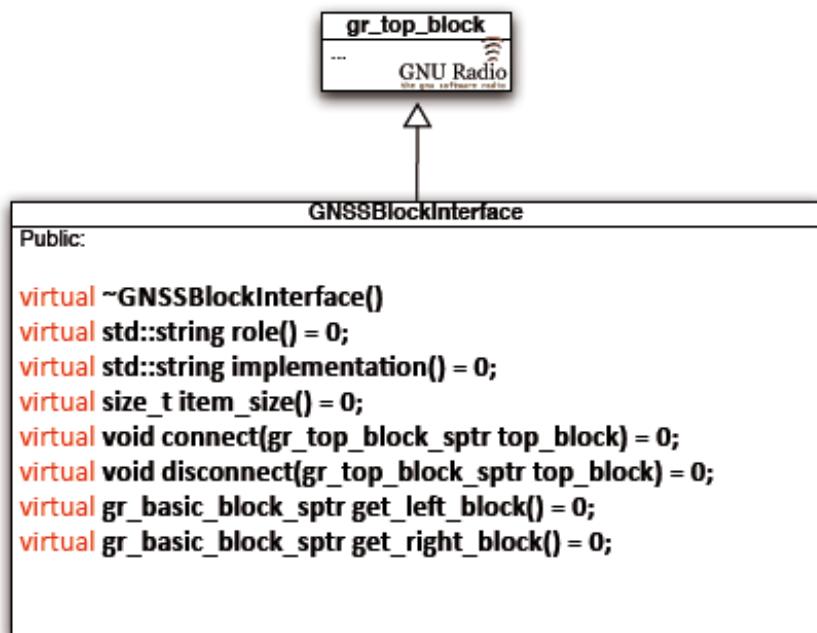


Figura 2-5: Interfaz general para los bloques de procesado de señal.

A partir de las subclases de *GNSSBlockInterface* definimos las interfaces de los bloques del receptor mostrados en la Figura 2-2. Esta jerarquía, que se muestra en la Figura 2-6, establece la definición de los diferentes algoritmos e implementaciones, que serán instanciados de acuerdo al fichero de configuración. Esta estrategia permite múltiples implementaciones que comparten una interfaz común, consiguiendo el objetivo de disociar las interfaces de las implementaciones, ya que define una familia de algoritmos, encapsula cada uno de ellos, y los hace intercambiables. Por lo tanto, se habilita que el algoritmo varíe independientemente del programa que lo utiliza.

Por último, como se puede ver en la Figura 2-6 entre las interfaces y los bloques que se encargan de realizar la implementación de los algoritmos de procesado, y que heredan directamente de la clase de GNURadio *gr_block*, existe una

clase intermedia denominada adaptador que se encarga de encapsular los bloques de GNURadio y adapta sus entradas y salidas para que puedan ser conectados al *flowgraph* de GNSS-SDR. De esta forma, si se quiere utilizar algún bloque del repositorio de GNU Radio, tan sólo hay que escribir el adaptador correspondiente que implemente los métodos de la interfaz de GNSS-SDR a la que se corresponda el bloque que se quiere utilizar.

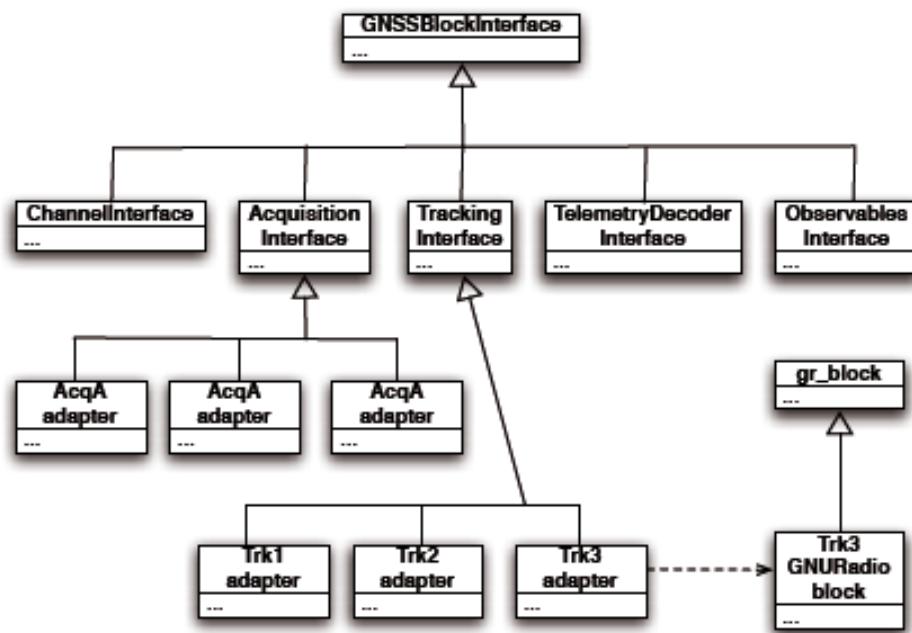


Figura 2-6: Interfaz general para los bloques de procesado de señal.

En los apartados siguientes, se hace una descripción inicial de las subclases de `GNSSBlockInterface`. Dichas subclases son, como ya se ha comentado, interfaces abstractas que difieren su instanciación a sus propias subclases.

2.5.2.1 Signal Source

El módulo fuente de señal (*Signal Source*) está a cargo de la ejecución del controlador de hardware, es decir, la parte del código que se comunica con el cabezal de RF y recibe las muestras procedentes del ADC. Esta comunicación se realiza generalmente a través de un puerto USB o del bus Ethernet. Dado que el procesado de los datos en tiempo real requiere de una ejecución altamente optimizada de todo el receptor, este módulo también permite la lectura de las muestras de un archivo almacenado en un disco duro en aquellas aplicaciones sin limitaciones de tiempo de procesado. Los parámetros relevantes de las muestras son la frecuencia intermedia (o las componentes banda base I & Q), la frecuencia de muestreo y el número de bits por muestra, que deben ser especificados por el usuario en el archivo de configuración.

Este módulo también realiza la adaptación de profundidad de los bits, ya que la mayoría de los cabezales de RF existentes proporcionan muestras cuantificadas con 2 o 3 bits, mientras que las operaciones dentro del procesador se llevan a cabo

mediante palabras de 32 o 64 bits, dependiendo de su arquitectura. Aunque existen implementaciones de los procesos computacionales más intensivos (sobre todo de la correlación) que utilizan arquitecturas y tipos de datos específicos en pro de la eficiencia (por ejemplo los correladores basados en *Single-Input Multiple-Data* (SIMD) presentados en [2-30], explotando un subconjunto específico de las instrucciones en lenguaje ensamblador de los procesadores de Intel que permiten la computación en paralelo) este enfoque es específico para un tipo de procesador y da como resultado un código no portable. Por este motivo en GNSS-SDR se propone mantener las muestras obtenidas de la señal en tipos de datos estándar y dejar que el compilador seleccione la mejor versión existente en las librerías del sistema de las rutinas necesarias para un procesador determinado (implementadas a partir de SIMD o cualquier otra tecnología específica del procesador utilizado). La capa de abstracción que realiza esta selección entre diferentes implementaciones de la más eficiente en el procesador concreto donde se está ejecutando se gestiona mediante la librería Volk, comentada en el Capítulo 4).

En el momento de la escritura de este proyecto GNSS-SDR cuenta con cuatro implementaciones diferentes de *Signal Source*: *File_Signal_Source*, que permite leer las muestras volcadas en un fichero de datos; *UHD_Signal_Source*, que es una implementación del Universal Hardware Driver de GnuRadio [2-31] que permite utilizar la familia de periféricos USRP [2-2], con la placa hija DBSRX [2-3] como cabezal de radiofrecuencia; *GN3S_Signal_Source*, que permite utilizar el dispositivo SiGe GN3S Sampler v2, basado en el circuito integrado SiGe 4120 GPS [2-4]; y *Rtlsdr_Signal_Source*, que permite utilizar receptores USB de señal digital de televisión DVB-T basados en el chipset Realtek RTL2832U [2-7] como cabezal de RF en GNSS-SDR (ver Figura 2-3).

2.5.2.2 *Signal Conditioner*

El acondicionador de señal (*Signal Conditioner*) es el encargado de adaptar la señal procedente del *Signal Source* a un formato acorde para ser utilizado por el resto de bloques del receptor, este bloque actúa como una fachada entre la fuente de señal y la sincronización de los canales, proporcionando una interfaz simplificada para la señal de entrada. En el caso de cabezales de radiofrecuencia multibanda, este módulo se encarga de proveer un flujo de datos separados para cada banda.

Este bloque se explica en profundidad en el Capítulo 3 de este proyecto debido a que el autor del mismo se ha encargado del diseño de su implementación.

2.5.2.3 *Channel*

Un canal (*Channel*) encapsula todo el procesado de señal dedicado a un solo satélite. Por lo tanto, es un gran objeto compuesto que encapsula los bloques de adquisición, *tracking* y decodificación. Al ser un objeto compuesto, que puede ser tratado como una entidad única, esto hace que se pueda replicar fácilmente. Dado que el número de canales es seleccionable por el usuario en el archivo de configuración, este enfoque ayuda a mejorar la escalabilidad así como el mantenimiento del receptor.

Este bloque se explica con más detalle en el Capítulo 5 de este proyecto debido a que el autor del mismo se ha encargado del diseño de su implementación.

2.5.2.4 Acquisition

La primera tarea de un receptor GNSS consiste en detectar la presencia o ausencia de los satélites a la vista. Esto se hace mediante el proceso de adquisición, que también proporciona una estimación aproximada de dos de los parámetros de la señal: el desplazamiento frecuencial f_{d_i} respecto a la frecuencia intermedia (IF), y un retardo temporal τ_i que permite al receptor crear un código local sincronizado con el código de la señal de entrada.

`AcquisitionInterface` es la interfaz común para todos los algoritmos de adquisición y sus correspondientes implementaciones. La interfaz del algoritmo, que puede variar en función del uso que se haga de la información externa al receptor, como en el caso de los sistemas GNSS asistidos, se define en una clase denominada adaptador (ver la Figura 2-7). Estos adaptadores encapsulan la interfaz de los bloques de GNU Radio en una interfaz compatible con `AcquisitionInterface`. Esto permite el uso de bloques de GNU Radio ya existentes derivados de `gr_block`, y asegura que las nuevas implementaciones que se desarrollen también serán reutilizables en otras aplicaciones basadas en GNU Radio. Además, añade otra capa de abstracción, ya que cada algoritmo de adquisición puede tener diferentes implementaciones (por ejemplo, utilizando diferentes librerías dependiendo del sistema de las señales a adquirir). De esta manera, las implementaciones pueden ser mejoradas sucesivamente sin tener ningún impacto ni en la interfaz del algoritmo, ni la interfaz general de la adquisición.

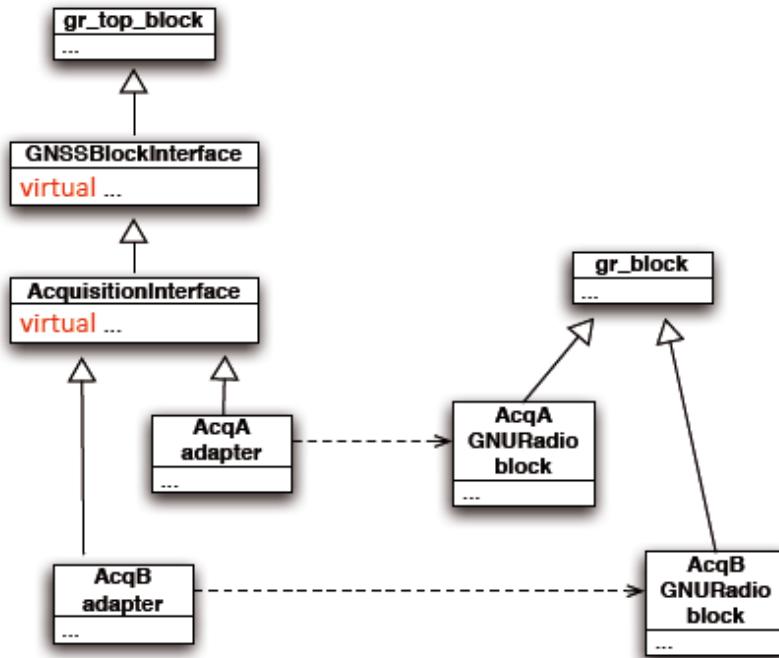


Figura 2-7: Jerarquía de clases para el módulo de Adquisición. Las clases adaptadoras AcqA y AcqB encapsulan el bloque de adquisición propio en una interfaz compatible con `AcquisitionInterface`. Los bloques AcqA y AcqB GNU Radio blocks son diferentes implementaciones de la adquisición. El procesado de señal de la adquisición se realiza en estos bloques. Se pueden usar bloques de GNU Radio existentes o implementar unos nuevos.

Los métodos de adquisición basados en software pueden consistir en una búsqueda en serie (*serial search acquisition*), una búsqueda frecuencial en paralelo basada en la Transformada Rápida de Fourier (FFT) [2-1] (que pueden ser implementados con librerías especializadas y muy optimizadas como la FFTW [2-32]), usando integración coherente o no coherente [2-33], o incluso utilizando señales que pertenezcan a diferentes bandas de frecuencia [2-34].

Este bloque se explica en profundidad en el Capítulo 4 de este proyecto, haciendo hincapié en la formulación matemática desarrollada por el autor para el cálculo del umbral de decisión del algoritmo de adquisición implementado.

2.5.2.5 Tracking

Cuando un satélite se declara presente, los parámetros estimados por el módulo de adquisición se pasan al módulo de *tracking* del receptor, lo que da lugar a la segunda etapa de la unidad de procesado de señales, cuyo objetivo es realizar una búsqueda local de estimaciones más precisas tanto del retardo de código como de la frecuencia y fase de la portadora, así como realizar un seguimiento de sus eventuales variaciones. Los algoritmos de *tracking* que realizan el seguimiento del retardo de código incluyen el Lazo de Seguimiento del Retardo (*Delay Lock Loop*, DLL) y sus múltiples variantes, así como detectores de dicho seguimiento (detectores de *lock*) que decidan si el DLL está enganchado al retardo de código de la señal recibida o se ha producido una pérdida en dicho seguimiento. Por otro lado los algoritmos denominados Lazo de Seguimiento de Fase (*Phase Lock Loop*, PLL) y Lazo de Seguimiento de Frecuencia (*Frequency Lock Loop*, FLL) son los métodos habituales para el *tracking* de la portadora (además, también abarcan diferentes discriminadores para el lazo de fase así como diferentes detectores de *lock* para la portadora [2-35]). Dado que el tiempo que transcurre desde que se produce una adquisición positiva hasta el inicio del *tracking* no se conoce, se ha implementado un contador con el fin de estimar en qué muestra de la secuencia del código se inicia el *tracking* después de un período de tiempo determinado.

Una vez más, una jerarquía de clases que consiste en una clase *TrackingInterface*, las clases adaptadoras y sus subclases donde se implementan los diversos algoritmos, proporciona una manera eficiente de probar diferentes enfoques para el *tracking*, con pleno acceso a los parámetros que se deseen evaluar. Además de los lazos de seguimiento clásicos, esta arquitectura da la posibilidad de testear otros enfoques, como el método de ajuste de bloques de la señal de sincronización (*block adjustment of synchronizing signal method*, *BASS method*) [2-33], estrategias basadas en combinaciones de señales en diferentes bandas de frecuencia [2-36] o esquemas de hibridación con señales de medidas inerciales [2-37].

2.5.2.6 Decodificación del mensaje de navegación (*Telemetry Decoder*)

La mayoría de los enlaces de señales GNSS son moduladas por un mensaje de navegación que contiene el momento en que se transmitió el mensaje, los parámetros orbitales de los satélites (también conocidos como efemérides) y un almanaque (información sobre el estado general del sistema, órbita aproximada de todos los satélites de la red, así como los datos relacionados con la corrección de errores). Los bits de los datos de navegación se estructuran en palabras, éstas en páginas, después en subtramas, éstas en marcos (*frames*) y finalmente en supertramas. A veces, los bits que corresponden a un solo parámetro se distribuyen

en diferentes palabras, y los valores extraídos de diferentes frames son necesarios para la correcta decodificación del mensaje de navegación. Algunas palabras se utilizan sólo para la sincronización, otras para el control de errores mientras que otras contienen información real. También existen mecanismos de control de errores que van desde sistemas de control de paridad hasta sistemas de codificación de corrección preventiva de errores (*Forward Error Correction*, FEC), dependiendo del sistema. Toda esta complejidad de decodificación es administrada por una máquina de estados finitos implementada con la librería Boost.Statechart [2-38]. Los detalles sobre la estructura del mensaje de navegación se presentan en los documentos de especificaciones relacionadas con cada sistema: véase [2-39] para GPS L1 C / A y L2C, [2-40] para GPS L1C, [2-41] para GPS L5, [2-42] para GLONASS, [2-43] de Galileo y [2-44] para SBAS. La jerarquía de clases de estas implementaciones sigue la misma estructura que la arquitectura ya presentada para la adquisición y el *tracking*: una clase llamada *TelemetryDecoderInterface* proporciona una interfaz única para decodificar el mensaje de navegación de los diferentes sistemas. La subclase adaptador encapsula las diferentes implementaciones.

2.5.2.7 Observables

Los sistemas GNSS ofrecen diferentes tipos de observables. Los más utilizados son los observables de código, también llamados pseudorangos. El término pseudo viene del hecho de que en el receptor el error del reloj es desconocido y por lo tanto la medida no es un observable puro. Las aplicaciones de alta precisión también utilizan los observables de la fase de la portadora, que se basan en la medición de la diferencia entre la fase de la portadora transmitida por los satélites GNSS y la fase de la portadora generada en el receptor. Ambos observables se calculan a partir de las salidas del módulo de *tracking* y del módulo de decodificación del mensaje de navegación.

Este módulo recoge todos los datos aportados por todos los canales en estado de *tracking*, alinea todos los datos recibidos en un conjunto coherente de datos, y calcula los observables. Los detalles matemáticos de dicho procedimiento se pueden encontrar en [2-45]. El formato de salida de estos datos es en forma de archivos RINEX, tal como se describe en la Sección 2.4.1.2. Los archivos RINEX allanan el camino para conseguir un posicionamiento de alta precisión, ya que pueden ser utilizados directamente por aplicaciones y librerías que implementan los algoritmos correspondientes para tal fin.

2.5.2.8 PVT

Aunque el procesado de datos para la obtención de soluciones PVT de alta precisión se aleja del objetivo inicial de GNSS-SDR, se ofrece un módulo que permite calcular una solución de mínimos cuadrados simple y se permite que en un futuro se puedan implementar métodos de posicionamiento más sofisticados. La integración con las librerías y herramientas de software que son capaces de tratar con datos de varias constelaciones como GPSTk o Glab [2-46] constituyen una solución viable para conseguir un receptor GNSS de alto rendimiento, completamente personalizable.

2.6 Bibliografía

[2-1] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S. H. Jensen, “A Software-Defined GPS and Galileo Receiver. A Single-Frequency Approach”, 1^a edición, Boston: Birkhäuser, noviembre de 2006. Capítulo 5: GNSS Receiver Operation Overview. pp. 73.

[2-2] Ettus Research, “USRP1”, <https://www.ettus.com/product/details/USRPPKG>, Comprobado: 9 de junio de 2013.

[2-3] Ettus Research, “Product Detail: DBSRX2 800-2300 MHz Rx”, <http://www.ettus.com/product/details/DBSRX2>, Comprobado: 9 de junio de 2013.

[2-4] SiGe Semiconductor, “SE4120L GNSS receiver IC datasheet”, May 26, 2009, Ottawa, ON, Canada.

[2-5] Maxim Integrated Products, Sunnyvale, CA, MAX2769 Universal GPS Receiver, June 2010, Ref. 19-0791; Rev 2.

[2-6] N. Falk, T. Hartmann, H. Kern, B. Riedl, T. Pany, R. Wolf, and J. Winkel, “SX-NSR 2.0 A multifrequency and multi-sensor software receiver with a quad-band RF front end”, in Proc. of the 23rd International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2010), Portland, OR, Sept. 2010, pp. 1395–1401.

[2-7] Realtek Semiconductor Corp., “RTL2832U. DVB-T COFDM Demodulator + USB 2.0”, <http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PNid=22&PFid=35&Level=4&Conn=3&ProdID=257>. Comprobado: 30 de enero de 2013.

[2-8] GNSS-SDR, “GNSS-SDR operation with a Realtek RTL2832U USB dongle DVB-T receiver”, <http://gnss-sdr.org/documentation/gnss-sdr-operation-realtek-rtl2832u-usb-dongle-dvb-t-receiver>. Comprobado: 30 de enero de 2013.

[2-9] W. Gurtner and L. Estey, RINEX. The Receiver Independent Exchange Format Version 3.01, June 2009.

[2-10] National Marine Electronics Association, “NMEA 0183 Standard”, http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp, Comprobado: 31 de enero de 2013.

[2-11] Open Geospatial Consortium Inc., “OGC KML”, Version: 2.2.0, abril de 2008.

[2-12] ISO/IEC 14882:2011(E) Programming Languages - C++, Third Edition, 2011.

[2-13] “GNU Radio”, <http://gnuradio.org>. Comprobado: 31 de enero de 2013..

[2-14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley, Upper Saddle River, NJ, 1995.

[2-15] B. P. Douglass, "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", Addison Wesley, Upper Saddle River, NJ, 2002.

[2-16] "Google C++ Testing Framework", <http://code.google.com/p/googletest/>, Retrieved: 7 de febrero de 2013.

[2-17] "Drupal", <http://drupal.org>, Comprobado: 11 de Febrero de 2013.

[2-18] "SourceForge", <http://sourceforge.net/>, Comprobado: 11 de Febrero de 2013.

[2-19] "Apache Subversion", <http://subversion.apache.org>, Comprobado: 11 de Febrero de 2013.

[2-20] "git. The fast version control system", <http://gitscm.com>, Comprobado: 11 de Febrero de 2013.

[2-21] "Doxygen", <http://www.stack.nl/~dimitri/doxygen/>, Comprobado: 11 de Febrero de 2013.

[2-22] "CMake", <http://www.cmake.org/>, Comprobado: 11 de Febrero de 2013.

[2-23] "GCC, the GNU Compiler Collection", <http://gcc.gnu.org>, Comprobado: 11 de Febrero de 2013.

[2-24] "Google logging library (google-glog). Logging library for C++," <http://code.google.com/p/google-glog/>, Comprobado: 11 de Febrero de 2013.

[2-25] "Google Performance Tools (google-perf-tools)," <http://code.google.com/p/google-perf-tools/>, Comprobado: 11 de Febrero de 2013.

[2-26] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Top Block and Hierarchical Block Base Classes. Classes. gr_top_block" http://gnuradio.org/doc/doxygen/classgr_top_block.html. Comprobado: 3 de junio de 2013.

[2-27] "GNU Radio 3.6.5 C++ API. Modules. Implementation details. Classes. gr_basic_block" http://gnuradio.org/doc/doxygen/classgr_basic_block.html. Comprobado: 3 de junio de 2013.

[2-28] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Top Block and Hierarchical Block Base Classes. Classes. gr_hier_block2" http://gnuradio.org/doc/doxygen/classgr_hier_block2.html. Comprobado: 3 de junio de 2013.

[2-29] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Base Classes for GR Blocks. Classes. gr_block" http://gnuradio.org/doc/doxygen/classgr_block.html. Comprobado: 3 de junio de 2013.

[2-30] G. W. Heckler and J. L. Garrison, "SIMD correlator library for GNSS software receivers", GPS Solutions, vol. 10, no. 4, pp. 269–276, Nov. 2006.

[2-31] Ettus Research, “USRP Hardware Driver software (UHD)”, <http://ettus-apps.sourcerepo.com/redmine/ettus/projects/uhd/wiki>, Comprobado: 12 de Febrero de 2013.

[2-32] M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” Proceedings of the IEEE, vol. 93, no. 2, pp. 216–231, 2005, Special issue on “Program Generation, Optimization, and Platform Adaptation”.

[2-33] J. Bao-Yen Tsui, Fundamentals of Global Positioning System Receivers. A Software Approach, John Wiley & Sons, Inc., Hoboken, NJ, 2nd edition, 2005.

[2-34] C. Gernot, K. O’Keefe, and G. Lachapelle, “Assessing three new GPS combined L1/L2C acquisition methods,” IEEE Transactions of Aerospace and Electronic Systems, vol. 3, no. 47, pp. 2239–2247, July 2011, DOI: 10.1109/TAES.2011.5937297.

[2-35] E. D. Kaplan and C. J. Hegarty, Eds., Understanding GPS. Principles and Applications, Artech House, Norwood, MA, 2nd edition, 2006.

[2-36] D. Megahed, C. O’Driscoll, and G. Lachapelle, “Performance evaluation of combined L1/L5 Kalman filterbased tracking versus standalone L1/L5 tracking in challenging environments,” in Proc. of the 22nd International Meeting of the Satellite Division of The Institute of Navigation (ION GNSS), Savannah, GA, Sept. 2009, pp. 2591–2601.

[2-37] C. Fernández-Prades, P. Closas, and J. Vilà-Valls, “Nonlinear filtering for ultra-tight GNSS/INS integration,” in Proc. of the IEEE International Conference on Communications (ICC 2010), Cape Town (South Africa), May 2010, pp. 1–5, DOI: 10.1109/ICC.2010.5502037.

[2-38] “The Boost Statechart Library,” www.boost.org/doc/libs/release/libs/statechart/doc/index.html, Comprobado: 19 de Junio de 2013.

[2-39] Science Applications International Corporation, *Interface Specification IS-GPS-200 Revision E. Navstar GPS Space Segment/Navigation User Interfaces*, El Segundo, CA, June 8, 2010.

[2-40] Science Applications International Corporation, *Interface Specification IS-GPS-800 Revision A. Navstar GPS Space Segment/User Segment L1C Interfaces*, El Segundo, CA, June 8, 2010.

[2-41] Science Applications International Corporation, *Interface Specification IS-GPS-705 Revision A. Navstar GPS Space Segment/User Segment L5 Interfaces*, El Segundo, CA, June 8, 2010.

[2-42] Russian Institute of Space Device Engineering, Moscow, Russia, *Global Navigation Satellite System GLONASS. Interface Control Document. Navigational radiosignal in bands L1, L2*, 2008, Version 5.1 downloadable from <http://www.glonass-ianc.rsa.ru>.

[2-43] European Union, *European GNSS (Galileo) Open Service. Signal In Space Interface Control Document*. Ref: OS SIS ICD, Issue 1, February 2010.

[2-44] RTCA, Washington, DC, *Minimum Operational Performance Standards for Global Positioning System/Wide Area Augmentation System Airborne Equipment*, DO-229D, Dec. 13 2006.

[2-45] A. Leick, *GPS Satellite Surveying*, John Wiley & Sons, Inc., Hoboken, NJ, 3rd edition, 2004.

[2-46] M. Hernández-Pajares, J. M. Juan, J. Sanz, P. Ramos- Bosch, A. Rovira-García, D. Salazar, J. Ventura-Traveset, C. López-Echazarreta, and G. Hein, “The ESA/UPC GNSS-Lab Tool (gLAB),” in Proc. of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC’2010), ESTEC, Noordwijk, The Netherlands, Dec. 2010, DOI: 10.1109/NAVITEC.2010.5708032.

3 GNSS-SDR: SIGNAL CONDITIONER

3.1 Resumen

En este capítulo se analizará el bloque Acondicionador de Señal (*Signal Conditioner*) del sistema GNSS-SDR. En primer lugar se explica brevemente los sub-bloques que lo componen y la funcionalidad de cada uno de ellos. En los siguientes apartados se explican con mayor detalle dos de estos bloques. Concretamente en el apartado 3.3, se explica el bloque que lleva a cabo el remuestreo (*Resampler*), mediante un análisis teórico de los procesos implicados (diezmado, interpolación y filtrado) y la realización de varios test para comparar los diferentes bloques existentes y el diseñado para GNSS-SDR. En el apartado 3.4 se hace hincapié en la necesidad de añadir un bloque de filtrado que también sea capaz de realizar una translación frecuencia (*InputFilter*) y se explica la adaptación de 2 bloques de GNURadio para tal fin, aportando también resultados de la aplicación de los mismos en un test con el receptor GNSS-SDR completo. Cierra este capítulo un apartado con las conclusiones obtenidas del trabajo realizado.

3.2 Introducción

Uno de los requerimientos de la implementación del sistema GNSS-SDR es que idealmente debería funcionar con señales provenientes de diversos cabezales de RF así como con ficheros de datos guardados en disco. Dichas señales tienen ciertos parámetros (la frecuencia intermedia IF, el número de bits por muestra y la frecuencia de muestreo f_s) que varían dependiendo de la fuente que las ha capturado. De esta forma, si se desea independizar el receptor de dichos parámetros, en primer lugar se debe implementar un bloque que confiera uniformidad a los datos de entrada. Tal y como se muestra en la Figura 3-1 este bloque recibe el nombre de Acondicionador de Señal (*Signal Conditioner*). Este bloque es un objeto compuesto que encapsula a otros tres bloques denominados *Data Type Adapter*, *Input Filter* y *Resampler*. La clase *Signal_Conditioner* es un adaptador que encapsula a los 3 bloques mencionados y no tiene ninguna relación de dependencia con ningún bloque análogo que herede del *gr_block* de GNU Radio ya que no lleva a cabo ningún procesado de señal, tan solo conecta y encapsula a estos 3 bloques.

Las diferentes funciones que realiza cada bloque son, en el orden en que se llevan a cabo:

- La conversión del tipo de datos en que se guardan las muestras de la señal de entrada al tipo de datos que se utiliza en los bloques del sistema GNSS-SDR, que se lleva a cabo en el bloque *Data Type Adapter*.

- Bajada en frecuencia y filtrado de la señal de entrada, que se lleva a cabo en el bloque *InputFilter*.
- Reconversión de la frecuencia de muestreo de la señal de entrada a una fijada por el sistema, que se lleva a cabo en el bloque *Resampler*.

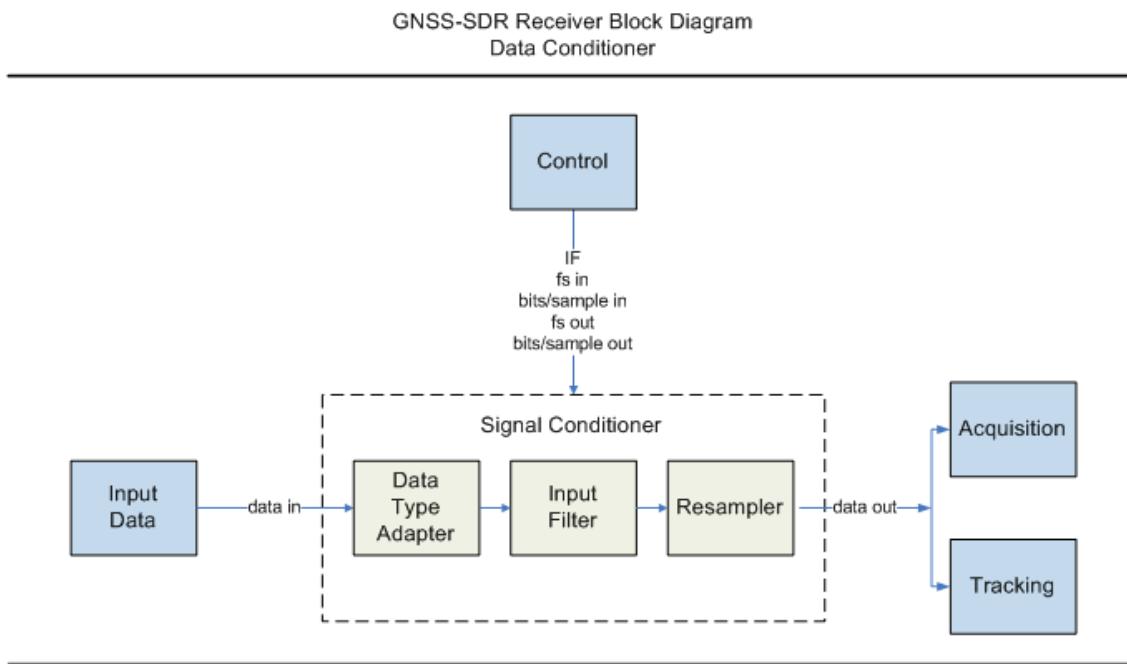


Figura 3-1: El bloque Signal Conditioner con sus entradas y salidas

En el momento de la escritura de este proyecto GNSS-SDR dispone de una implementación para el bloque *Data Type Adapter* denominada *Ishort_To_Complex* que permite convertir las muestras I&Q guardadas por el cabezal de radiofrecuencia en formato *interleaved short* al formato complejo de GNU Radio *gr_complex* (que es una encapsulación del formato de la librería std de C++ *std::complex< float >*) utilizado por los bloques de procesado de señal de GNSS-SDR. La decisión de poner este bloque en primer lugar responde al hecho de que de esta forma, los siguientes bloques que forman parte del *Signal Conditioner* puedan ser implementados con bloques de GNU Radio, ya que los datos ya se han adaptado al formato utilizado por los bloques de este repositorio. Para el bloque *InputFilter* se dispone de las implementaciones *Fir_Filter* y *Freq_Xlating_Fir_Filter*. Por último, para el bloque *Resampler* se dispone de la implementación *Direct_Resampler*. Las implementaciones de estos dos últimos bloques se explican en los siguientes sub-apartados ya que han sido desarrolladas por el autor de este proyecto.

3.3 Resampler

En este apartado se estudia en detalle la teoría e implementación del tercer bloque mencionado, el *resampler*.

Tal y como se define en [3-1] el *resampling* (o remuestreo) es el “proceso mediante el cual se cambia la frecuencia de muestreo de una señal una vez ésta ya ha sido muestreada”. Para definir dicho proceso consideremos una señal continua $x_c(t)$ que ha sido muestreada a una frecuencia de muestreo $f_{old} = \frac{1}{T_{old}}$ (la f_{s-in} del esquema del *Signal Conditioner* de la Figura 3-1) así como sus muestras discretas $x_{old}[n] = x_c(n \cdot T_{old})$. El *resampling* es necesario cuando necesitamos obtener muestras $x_{new}[n] = x_c(n \cdot T_{new})$ a una nueva frecuencia de muestro $f_{new} = \frac{1}{T_{new}}$ (la f_{s-out} del *Signal Conditioner*) y no disponemos de la señal continua $x_c(t)$ para poder muestrearla a la frecuencia deseada.

Una alternativa al *resampling* para poder resolver el problema que se plantea consiste en realizar una conversión digital-analógica de la señal $x_{old}[n]$ para obtener una reconstrucción de la señal continua y posteriormente volver a muestrearla a la frecuencia deseada. En la práctica esta solución se suele descartar debido a la distorsión que ocasionan los conversores D/A y A/D, haciendo del proceso de *resampling* la vía óptima para llevar a cabo el cambio de la frecuencia de muestreo de la señal.

La frecuencia de muestreo puede cambiar de dos formas: aumentando (mediante el denominado proceso de interpolación que consiste en estimar nuevas muestras intermedias) o disminuyendo (el denominado proceso de diezmado que consiste en eliminar las muestras sobrantes). Se empieza analizando este último debido a su mayor simplicidad.

3.3.1 Resampling mediante diezmado

3.3.1.1 Teoría del diezmado

Este apartado, junto con el apartado 3.3.2.2, es una adaptación libre de [3-2], pero se incluyen en el proyecto para comodidad del lector y hacer de este proyecto un trabajo autocontenido.

Según [3-2] el diezmado es un procedimiento que consiste en la eliminación regular de muestras de una secuencia según una determinada relación de diezmado, que en este caso se denominará D (siendo D un número natural). De esta forma a partir de la secuencia $x_{old}[n]$ se puede diezmar por un factor D reteniendo las muestras múltiplos de D y descartando el resto:

$$x_{new}[n] = x_{old}[Dn] \quad (3-1)$$

El bloque que realiza la operación de diezmado se representa en la Figura 3-2.

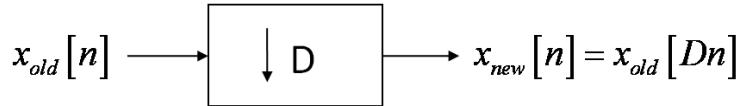


Figura 3-2: Sistema diezmador.

Para analizar el proceso de diezmado es conveniente introducir la siguiente secuencia (ver Figura 3-3):

$$v[n] = x_{old}[n]t[n] = \begin{cases} x_{old}[n] & n = 0, \pm D, \pm 2D \\ 0 & resto \end{cases} \quad (3-2)$$

donde $t[n]$ es un tren de impulsos con periodo D definido por:

$$t[n] = \sum_{r=-\infty}^{\infty} \delta[n+rD] = \frac{1}{D} \sum_{k=0}^{D-1} e^{j \frac{2\pi kn}{D}} \quad (3-3)$$

y cuya transforma de Fourier puede escribirse como:

$$T(f) = \sum_{n=-\infty}^{\infty} t[n] e^{-j2\pi fn} = \frac{1}{D} \sum_{k=0}^{D-1} \sum_{n=-\infty}^{\infty} e^{-j2\pi \left(f - \frac{k}{D}\right)n} = \frac{1}{D} \sum_{k=0}^{D-1} \delta\left(f - \frac{k}{D}\right) \quad (3-4)$$

Si se define la convolución en un periodo de las transformadas de Fourier $X_1(f)$ y $X_2(f)$ como:

$$X_1(f) * X_2(f) = \int_{-\frac{1}{2}}^{\frac{1}{2}} X_1(\lambda) X_2(f - \lambda) d\lambda \quad (3-5)$$

y se aplican las ecuaciones (3-2) y (3-4) junto con la propiedad de transformada de un producto, se puede calcular la transformada de Fourier de $v[n]$ de la siguiente forma:

$$V(f) = X_{old}(f) * T(f) = X_{old}(f) * \frac{1}{D} \sum_{k=0}^{D-1} \delta\left(f - \frac{k}{D}\right) = \frac{1}{D} \sum_{i=0}^{D-1} X_{old}\left(f - \frac{k}{D}\right) \quad (3-6)$$

Como puede observarse en la Figura 3-3 la secuencia $v[n]$ conserva los valores de las muestras de $x_{old}[n]$ que formarán parte de $x_{new}[n]$ mientras que anula las muestras que se descartarán.

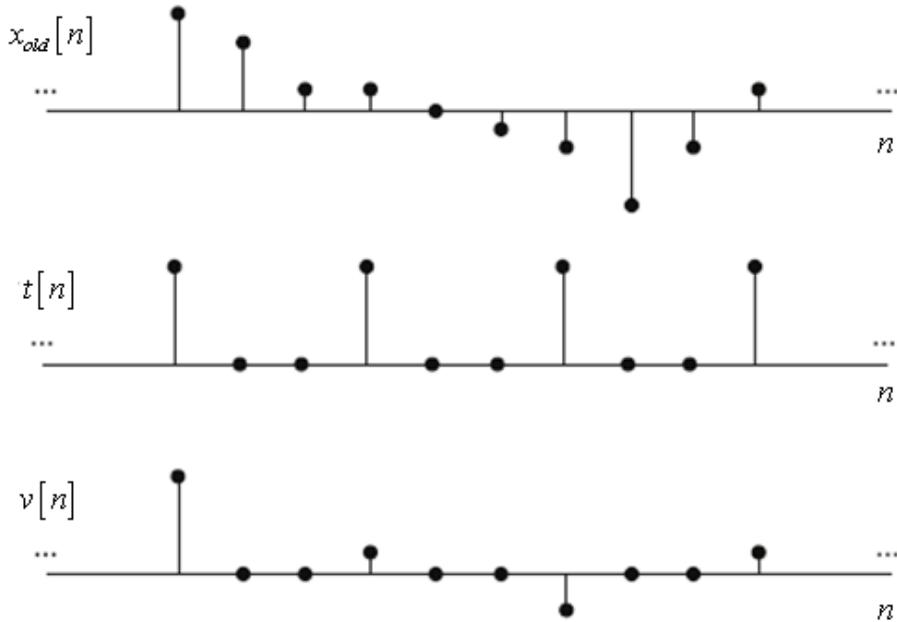


Figura 3-3: Secuencias $x_{old}[n]$, $t[n]$ y $v[n]$ con factor de diezmado $D=3$. (figura extraída de [3-2] y modificada)

La expresión de $x_{new}[n]$ queda por tanto:

$$x_{new}[n] = x_{old}[Dn] = v[Dn] \quad (3-7)$$

y su transformada de Fourier en función de la secuencia auxiliar $v[n]$:

$$X_{new}(f) = \sum_{n=-\infty}^{\infty} x_{new}[n] e^{-jD2\pi fn} = \sum_{n=-\infty}^{\infty} v[Dn] e^{-jD2\pi fn} = \sum_{m=-\infty}^{\infty} v[m] e^{-j2\pi f \frac{m}{D}} = V\left(\frac{f}{D}\right) \quad (3-8)$$

Aplicando la ecuación (3-6) a la ecuación (3-8) se puede expresar la transformada de Fourier de la señal resultante en función de la transformada de la señal de entrada:

$$X_{new}(f) = V\left(\frac{f}{D}\right) = \frac{1}{D} \sum_{k=0}^{D-1} X_{old}\left(\frac{f-k}{D}\right) \quad (3-9)$$

La interpretación de esta ecuación se puede apreciar en la Figura 3-4 donde se puede ver un ejemplo de los espectros de las señales anteriores con $D=3$. La variable del eje de abscisas es la frecuencia discreta $f = \frac{F}{f_s}$, donde F es la

frecuencia analógica (en Hz) y f_s la frecuencia de muestreo (que en las 2 primeras gráficas se corresponde con f_{old} y en la tercera se corresponde con $f_{new} = \frac{f_{old}}{D}$). De esta forma, $f=1$ corresponde a la frecuencia analógica f_{old} y f_{new} respectivamente.

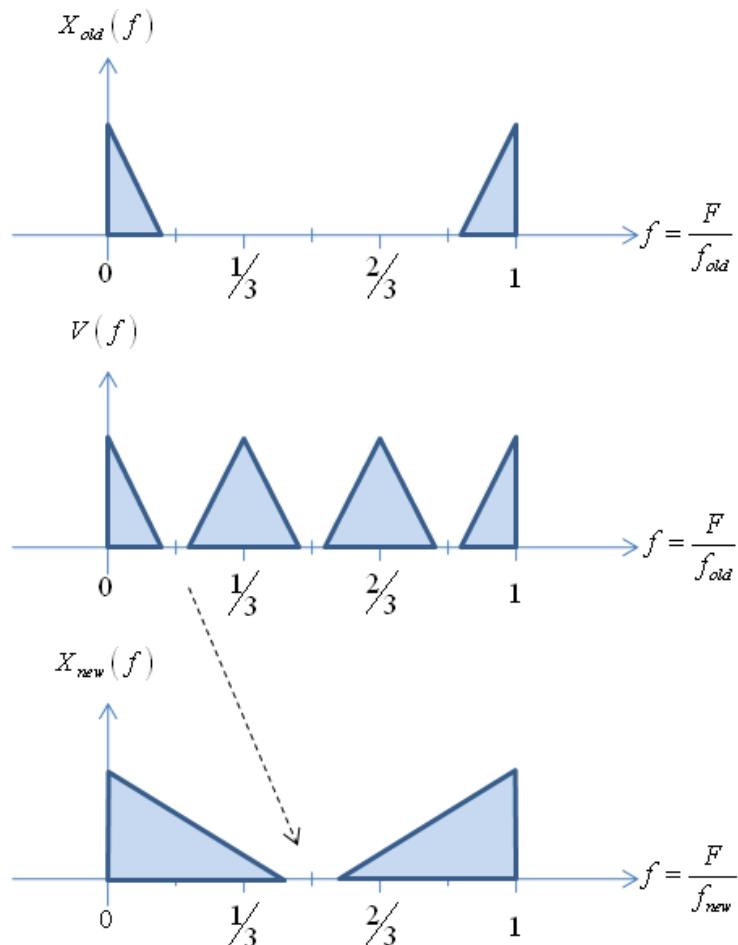


Figura 3-4: Transformadas de Fourier de las secuencias $x_{old}[n]$, $v[n]$ y $x_{new}[n]$ con $D=3$.

Como puede observarse en la Figura 3-4, el diezmado produce una expansión del espectro al representarlo en función de la frecuencia discreta debido a que, aparte de periodificarse, éste queda escalado inversamente por el factor de diezmado (otra forma de interpretarlo es que estamos cambiando la variable del eje de abscisas y por tanto estamos escalando dicho eje). De esta forma todas las componentes frecuenciales de la señal de entrada que se encuentren más allá de $f = \frac{1}{D}$ se asignan a frecuencias más allá de $f=1$ en el espectro de la señal diezmada. Este fenómeno puede producir solapamiento frecuencial (*aliasing*) de espectros y, por tanto, distorsión. Para no tener problemas de *aliasing* la nueva frecuencia de muestreo debe seguir cumpliendo el criterio de Nyquist ([3-3] y [3-4]), es decir $f_{new} > 2B$, siendo B el ancho de banda de la señal original en Hz. En caso de no cumplirse la condición de

Nyquist para la nueva frecuencia de muestreo y con el fin de evitar dicha distorsión la señal de entrada al diezmador debe ser filtrada con un filtro paso bajo cuya frecuencia de corte es $f_{stop} = \frac{1}{2D}$, que corresponde a una frecuencia analógica de $F_{stop} = \frac{f_{old}}{2D} = \frac{f_{new}}{2}$ tal y como se puede ver de forma ideal en la Figura 3-5 con $D = 3$.

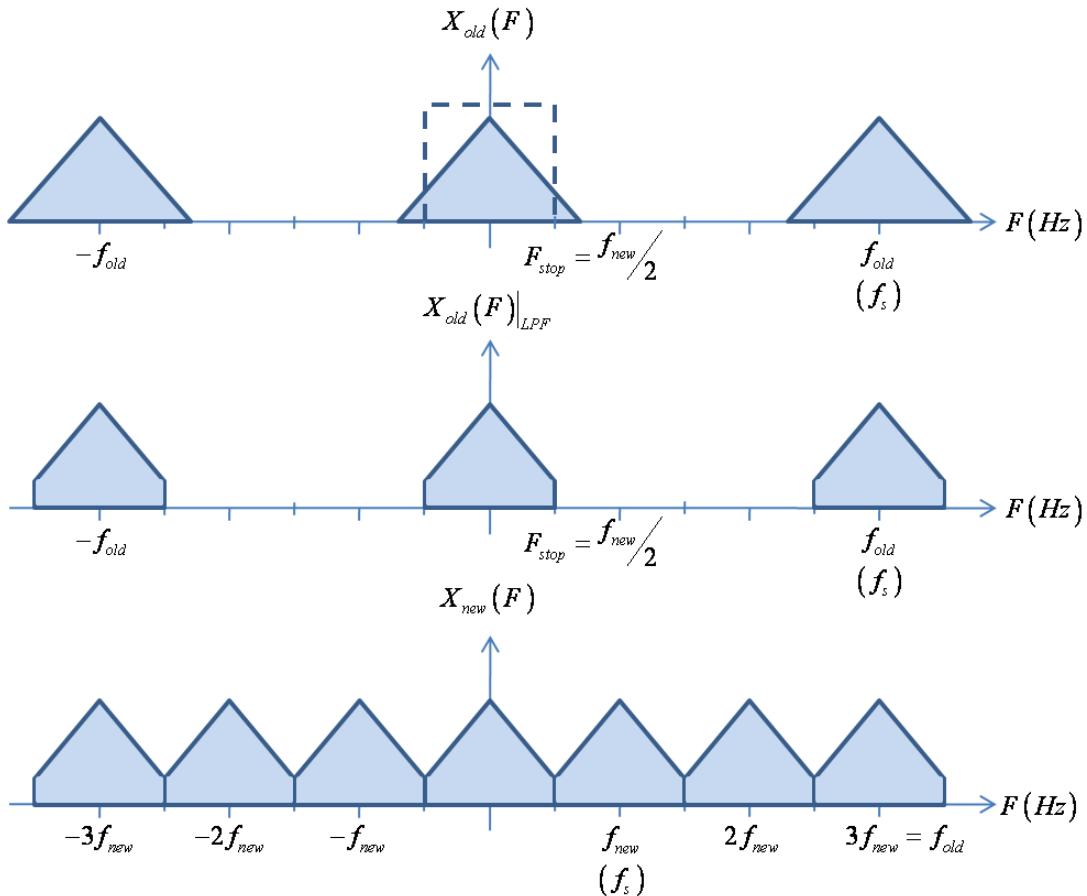


Figura 3-5: Proceso de filtrado para evitar la distorsión por *aliasing* en el diezmado (el LPF se muestra en líneas discontinuas).

El esquema definitivo del diezmador tal y como se muestra en la Figura 3-6 incorpora un filtro paso bajo (LPF) cuya banda atenuada (*stop band*) debe iniciarse en $F_{stop} = \frac{f_{old}}{2D} = \frac{f_{new}}{2}$.

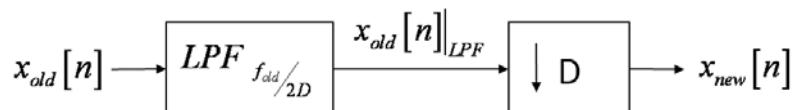


Figura 3-6: El proceso completo de diezmado con un filtro paso bajo.

3.3.1.2 Tipos de filtrado

En el caso ideal el filtro paso bajo que eliminaría por completo todas las frecuencias del espectro no deseadas tal y como se muestra en la Figura 3-5 se corresponde con la función sinc:

$$h_I(n) = \frac{1}{D} \operatorname{sinc}\left(\frac{n}{D}\right) \rightarrow H_I(f) = \begin{cases} 1 & \text{si } 0 \leq |f| < 1/2D \\ 0 & \text{resto} \end{cases} \quad (3-10)$$

El filtro de la ecuación (3-10) es un filtro IIR no causal. En la práctica, tal y como se indica en [3-5], se suele utilizar una implementación mediante filtro FIR debido fundamentalmente a la característica de fase lineal que posee su respuesta impulsional.

Dos de los aspectos más importantes al realizar el diseño de la aproximación FIR del filtro paso bajo ideal son: el número de coeficientes a utilizar en el filtro y el diseño de dichos coeficientes.

El problema de utilizar un filtro FIR es el compromiso entre la duración frecuencial de la banda de transición del filtro Δf (ver la Figura 3-7) y el número de etapas (coeficientes) S del mismo.

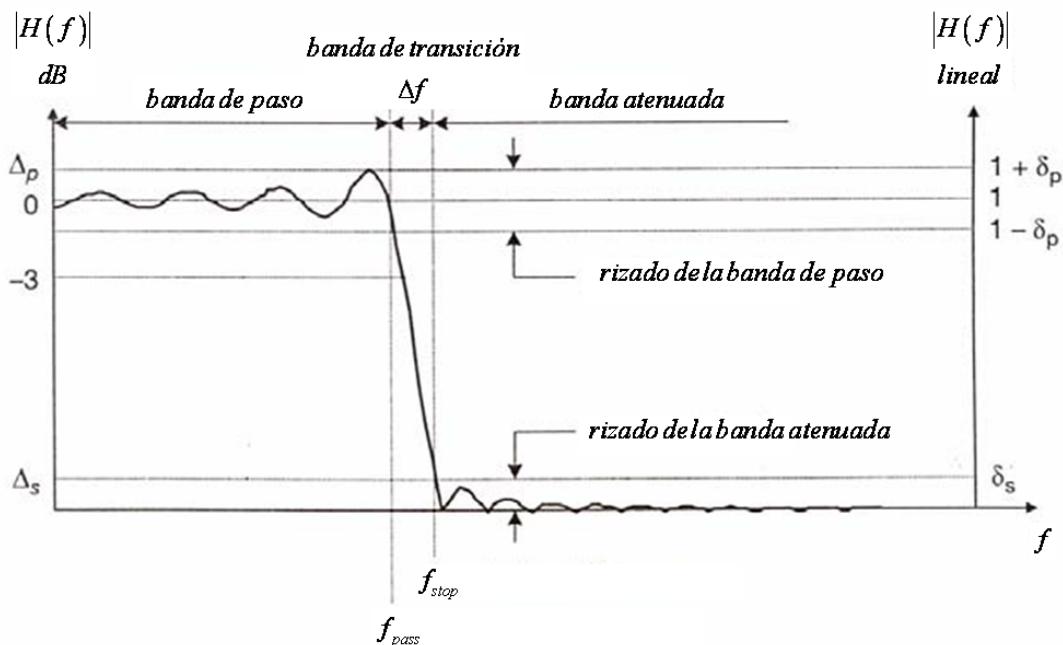


Figura 3-7: Respuesta frecuencial de un filtro paso bajo real.

Se ha demostrado (en [3-5] y [3-6]) que el número de coeficientes S de un filtro FIR paso bajo expresado en su forma directa (mediante amplificadores y retardos) es proporcional al ratio entre la frecuencia de muestreo original f_{old} y la banda de paso del filtro Δf . Concretamente:

$$S = k \cdot \frac{f_{old}}{\Delta f} \quad (3-11)$$

donde $2 < k < 4$ es un factor que depende del nivel de rizado que se pueda tolerar en la banda de paso (δ_p) y en la banda atenuada (δ_s) del filtro paso bajo.

Cuando se aplican factores de diezmado altos ($D > 10$) podrían surgir problemas si se necesita una banda de transición del filtro muy abrupta ya que, según lo visto en la ecuación (3-11), implicaría realizar un filtro con muchos coeficientes. En estos casos, tal y como se explica en [3-1], la solución pasa por dividir el proceso de diezmado en 2 o más etapas reduciendo así el número de coeficientes a utilizar en los filtros.

En cuanto al diseño de los coeficientes del filtro FIR, existen múltiples técnicas, entre las que destacan las cuatro siguientes: 1) enventanado de la respuesta impulsional, 2) muestreo de la respuesta frecuencial, 3) diseño por aproximación de Chebyshev y 4) la aproximación por mínimos cuadrados. La mayoría de ellas están bien documentadas en la mayoría de libros de procesado de señal [3-7] o en otros documentos [3-8].

En el primer caso (método de enventanado), el método consiste en que dadas las especificaciones de un filtro ideal y su respuesta impulsional en frecuencia $H_I(f)$, se construye el filtro FIR enventanando la respuesta impulsional del filtro en el dominio temporal, es decir, se obtiene la respuesta impulsional del filtro $h_I[n]$ antitransformando la respuesta frecuencial y se multiplica ésta por una ventana $w[n]$, que sea una función simétrica en un intervalo entorno al origen y de duración finita, con el objetivo de acotar la respuesta impulsional del filtro ideal.

$$h[n] = h_I[n] \cdot w[n] \quad (3-12)$$

Como la respuesta impulsional del filtro ideal y de la ventana son simétricas respecto al punto central el filtro tendrá fase lineal.

Posteriormente se retraza dicho producto con el fin de convertir la respuesta impulsional del filtro en causal.

Dadas las propiedades de la transformada de Fourier, esta multiplicación produce una convolución de los espectros del filtro ideal $H_I(f)$ con la respuesta frecuencial de la ventana $W(f)$ que provoca la aparición de la banda de transición del filtro así como un rizado en la banda de paso y en la banda atenuada (ver Figura 3-8).

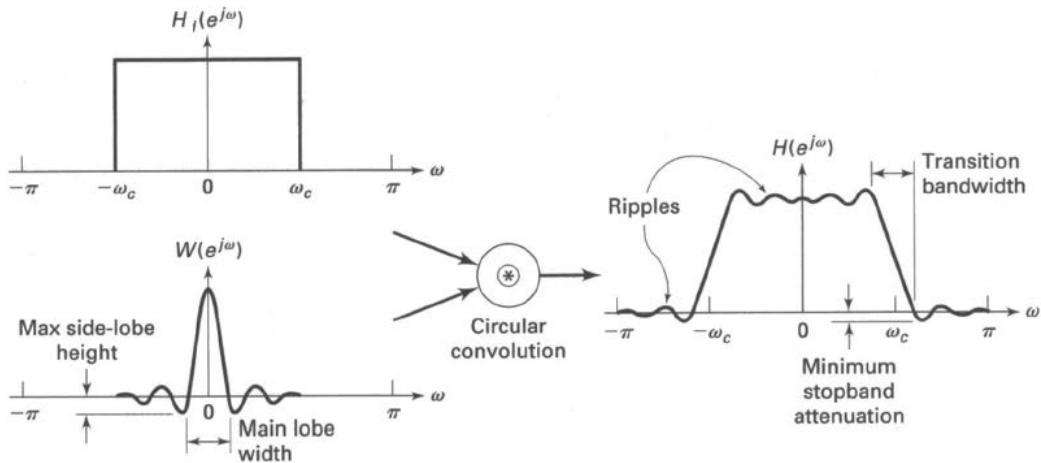


Figura 3-8: Análisis frecuencial del método de enventanado. Figura extraída de [3-7]

Existen numerosos tipos de ventanas que se utilizan dependiendo de la característica que se quiera mejorar (selectividad de la banda de transición, atenuación en la frecuencia de corte, rizado de las bandas de paso y atenuada, ...). Las más utilizadas son la ventana rectangular, la de Hanning, la de Hamming, la de Blackman, la de Blackman-Harris y la de Kaiser.

Así pues, el enventanado de la respuesta impulsional es una técnica sencilla para el diseño de filtros FIR, aunque una de las posibles dificultades del diseño puede recaer en la obtención de la respuesta impulsional del filtro ideal $h_I[n]$ dado que como las especificaciones de dicho filtro se dan en el dominio frecuencial puede que el cálculo de la transformada inversa de $H_I(f)$ no sea trivial.

En Matlab el método de enventanado se encuentra implementado en la función `fir1()` [3-9].

El segundo de los métodos comentados (muestreo en frecuencia) consiste en obtener muestras de la respuesta frecuencial del filtro ideal $H_I(f)$ y mediante el uso de la DFT inversa obtener la respuesta impulsional del filtro FIR. Por simplicidad el muestreo se suele realizar en S frecuencias equiespaciadas en el intervalo $[0,1]$.

De esta forma se define:

$$H[k] = H_I(f) \Big|_{f=f_k=\frac{k}{S}} \quad \text{para } k = 0, 1, \dots, S-1 \quad (3-13)$$

y la respuesta impulsional se obtiene de realizar la DFT inversa:

$$h[n] = DFT^{-1}\{H[k]\} = \frac{1}{S} \sum_{k=0}^{S-1} H[k] e^{j \frac{1}{S} kn} \quad \text{para } n = 0, 1, \dots, S-1 \quad (3-14)$$

Una de las ventajas de este método es que si la DFT se realiza sobre un número de muestras que sea potencia de 2 se puede implementar de forma eficiente mediante el algoritmo de la FFT.

Otra ventaja que ofrece este método es que la respuesta frecuencial del filtro obtenido $H(f)$ es la misma que la del filtro ideal $H_I(f)$ en los puntos muestreados, y por tanto el error de la aproximación (diferencia entre el filtro ideal y la aproximación) es nula en estos puntos. El gran inconveniente es que se carece de control de la respuesta del filtro fuera de estos puntos y por tanto del error que se comete. De esta forma, el error de la aproximación depende de la respuesta impulsional del filtro ideal y al ser éste mayor en aquellas zonas en las que se produce una transición abrupta esto provoca que el error sea superior en los límites de las bandas que dentro de ellas.

En Matlab el método de muestreo frecuencial se encuentra implementado en la función `fir2()` [3-10].

Los dos métodos anteriores son sencillos de implementar pero tienen una serie de desventajas, ya que no se pueden elegir los valores de las frecuencias que delimitan las bandas de paso f_{pass} y la banda atenuada f_{stop} de forma precisa y además no se puede diseñar el rizado de la banda de paso δ_p y el de la banda atenuada δ_s de forma independiente, distribuyéndose además este rizado de forma no uniforme dentro de las bandas.

El tercer método comentado (diseño por aproximación de Chebyshev) es conocido también como método de diseño de filtros óptimos de rizado constante. En él se explota la característica de que si el error se distribuye uniformemente podemos diseñar filtros con menor orden que verifiquen las especificaciones.

El criterio que se utiliza para el diseño del filtro (planteado como una aproximación de Chebyshev) es óptimo en el sentido de que el error de la aproximación (entre el filtro ideal y el real) se reparte uniformemente en cada banda (la banda de paso y la banda atenuada) minimizando el error máximo en cada una de ellas. El filtro resultante presenta, pues, rizado en ambas bandas (de ahí el apelativo de equiripple que se le suele asignar).

Si definimos la siguiente función de error:

$$E(f) = W(f) \cdot (|H_I(f)| - |H(f)|) \quad (3-15)$$

donde $H(f)$ es la respuesta del filtro a diseñar, $H_I(f)$ es la respuesta del filtro ideal y $W(f)$ es una función de pesos para especificar el error permitido en cada banda, el objetivo es encontrar los coeficientes del filtro $h[n]$ que minimizan el valor de $E(f)$ en toda la banda, permitiendo un valor máximo específico del error determinado por δ_p y δ_s a través de la función $W(f)$.

$$\min_{h[n]} \left[\max_f |E(f)| \right] \quad (3-16)$$

El diseño se reduce a un problema de optimización que consiste en fijar 4 de los 5 parámetros del filtro ($f_{pass}, f_{stop}, \delta_p, \delta_s$ y S) y optimizar el parámetro restante, Parks y McClellan proponen una solución al problema en [3-11] basándose en el algoritmo de Remez [3-12].

La gran ventaja de este método es que permite un gran control del filtro en cuanto a frecuencias, ganancias y longitud. El gran problema es que no existe una forma fácil de optimizar el filtro respecto a su longitud S , aunque existen aproximaciones como la de Hermann [3-13] y la de Kaiser [3-14].

En Matlab existen 2 funciones para llevar a cabo la aproximación de Chebyshev: la función `firpmord()` [3-15] nos da una estimación del orden del filtro dependiendo de sus requisitos, y la función `firpm()` [3-16] calcula los coeficientes del filtro.

El último método comentado (aproximación por mínimos cuadrados, *Least Squares*) consiste en una variante del método anterior.

En este caso en lugar de minimizar el valor absoluto del error $|E(f)|$ como en (3-16) se minimiza el error cuadrático medio $\|E(f)\|_2$ sobre el intervalo de frecuencias deseado, donde:

$$\|E(f)\|_2 = \sqrt{2 \int_0^{\frac{\pi}{2}} W(f) \cdot (|H_I(f)| - |H(f)|)^2 df} \quad (3-17)$$

Este método se describe en [3-17] y se implementa en Matlab a través de la función `firls()` [3-18].

Por último cabe comentar que el diezmado es un proceso que no es invariante en el tiempo, ya que si retrasamos una muestra la señal de entrada obtendremos una salida completamente diferente.

Siguiendo con el ejemplo de $D=3$, sea la secuencia original $x_{old}[n] = x_{old}[0], x_{old}[1], x_{old}[2], x_{old}[3], \dots$, con un diezmador de $D=3$ se obtiene una secuencia de salida $x_{new}[n] = x_{old}[0], x_{old}[3], x_{old}[6], \dots$. Al aplicar al diezmador la señal retardada una muestra $x_{old}^*[n] = x_{old}[1], x_{old}[2], x_{old}[3], x_{old}[4], \dots$ a la salida se obtiene $x_{new}^*[n] = x_{old}[1], x_{old}[4], x_{old}[7], \dots$ que no es una versión retardada de $x_{new}[n]$, con lo que queda demostrado que el sistema no es invariante con el tiempo.

3.3.2 Interpolación

3.3.2.1 Teoría matemática de la interpolación.

Desde un punto de vista matemático (concretamente en el campo del análisis numérico) la interpolación consiste en la construcción de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.

En ingeniería es frecuente disponer de un cierto número de puntos obtenidos mediante el muestreo de una función analógica o a partir de un experimento e intentar construir una función que los ajuste.

En ambos casos el problema se reduce a encontrar, a partir de n parejas de puntos (x_k, y_k) , una función f que verifique:

$$f(x_k) = y_k \quad k = 1, 2, \dots, n \quad (3-18)$$

denominada función interpolante de dichos puntos. Existen diversas formas de construir la función interpolante, algunos de ellos se analizan en los siguientes subapartados.

3.3.2.1.1 Interpolación por vecindad

Es el caso más sencillo y consiste en mantener el mismo valor de los puntos originales en un entorno de los mismos. Por tanto la función interpolante es constante a intervalos regulares tal y como se muestra en la Figura 3-9.

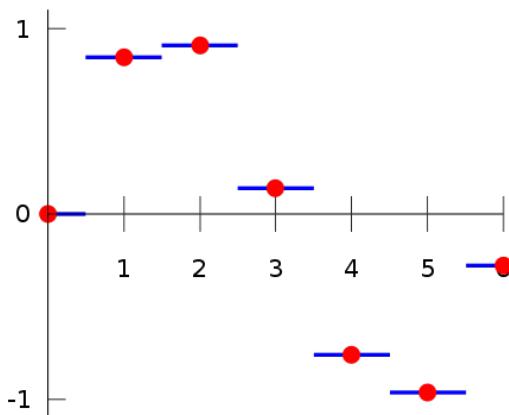


Figura 3-9: Interpolación por vecindad. En rojo los puntos originales y en azul la función interpolada. Imagen extraída de [3-19].

Este tipo de interpolación es muy sencilla de implementar y se utiliza en aplicaciones en tiempo real de Rendering 3D. En la Sección 3.3.4 veremos como en el software GNSS-SDR se ha implementado un *resampler* basado en este tipo de interpolación con muy buenos resultados en cuanto a tiempo de procesado y en cuanto a rendimiento global del receptor.

3.3.2.1.2 Interpolación Lineal

La interpolación lineal consiste en unir los puntos consecutivos mediante rectas tal y como se puede ver en la Figura 3-10.

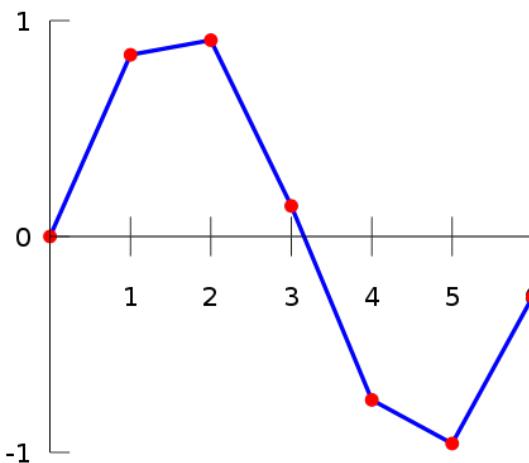


Figura 3-10: Interpolación Lineal. Imagen extraída de [3-19].

Dados 2 puntos (x_0, y_0) y (x_1, y_1) la ecuación de la recta que pasa por ambos puntos viene dada por:

$$y = \frac{y_1 - y_0}{x_1 - x_0} \cdot (x - x_0) + y_0 \quad (3-19)$$

En el caso de tener un conjunto de n puntos, la interpolación lineal consiste en utilizar como función interpolante para cada pareja de puntos consecutivos la ecuación de la recta (3-19) que pasa por dicha pareja de puntos.

La interpolación lineal se utiliza frecuentemente en análisis numérico y en la generación de gráficos por ordenador ya que es una característica habilitada en la mayoría de procesadores gráficos debido a su bajo coste de implementación.

Por último, comentar que la interpolación por vecindad, así como la interpolación lineal son un caso concreto de la interpolación por Splines que se verá en el apartado 3.3.2.1.4 .

3.3.2.1.3 Interpolación Polinómica

Tal y como se explica en [3-20] la interpolación polinómica se basa en el siguiente teorema:

“Si x_0, x_1, \dots, x_n son $(n+1)$ puntos diferentes y f es la función que nos da los valores para esos puntos, entonces existe un único polinomio P de grado como máximo n con la propiedad de que

$$P(x_k) = a_0 + a_1 \cdot x_k + a_2 \cdot x_k^2 + \dots + a_n \cdot x_k^n = f(x_k) \quad \text{para } k = 0, 1, \dots, n \quad (3-20)$$

O lo que es lo mismo, dados $(n+1)$ puntos existe un polinomio de grado n que pase por todos ellos.

Una vez obtenido el polinomio se puede generar cualquier nueva muestra simplemente evaluándolo. Esta operación se puede llevar a cabo de forma eficiente mediante el algoritmo de Horner [3-21] puesto que tan sólo realiza n sumas y n multiplicaciones para evaluar un polinomio de grado n , mientras que en la forma monomial se requieren de n sumas y $\frac{n^2+n}{2}$ multiplicaciones (si las potencias se calculan como multiplicaciones).

Los resultados de aplicar la interpolación polinómica a un conjunto de puntos puede observarse en la Figura 3-11.

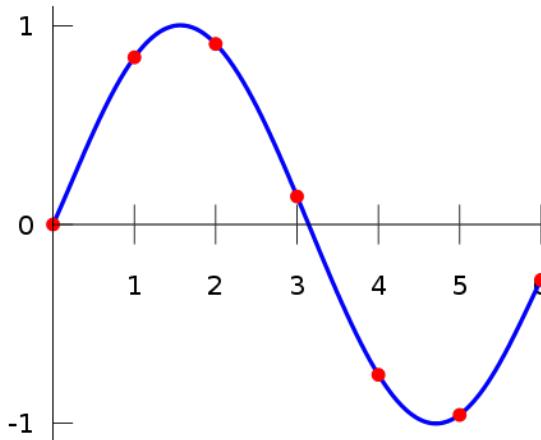


Figura 3-11: Interpolación Polinómica. Imagen extraída de [3-19].

Para la obtención del polinomio interpolador existen varios métodos, entre los que destacan los métodos algebraicos basados en la matriz de Vandermonde. Dado el sistema de ecuaciones definido por (3-20), sea y_k el valor de $f(x_k)$ podemos representar las $(n+1)$ ecuaciones de forma matricial:

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \cdots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \Rightarrow \boxed{\underline{X} \cdot \underline{a} = \underline{y}} \quad (3-21)$$

donde \underline{X} es la denominada matriz de Vandermonde. Para encontrar el polinomio interpolador se debe resolver el sistema anterior y aislar el vector \underline{a} ,

invirtiendo dicha matriz ($\underline{a} = \underline{\underline{X}}^{-1} \cdot \underline{y}$). El cálculo de la matriz inversa puede ser arduo en el caso de que el orden n del polinomio sea elevado, es por este motivo que se han llevado a cabo varios métodos que permiten reducir el número de operaciones en un orden, de $O(n^3)$ operaciones del método de eliminación de Gauss-Jordan a $O(n^2)$ de los métodos explicados en [3-22], [3-23] y [3-24] que se basan en las simetrías de la estructura de la matriz de Vandermonde.

A parte de los algoritmos basados en la matriz de Vandermonde, existen otros métodos para hallar el polinomio interpolador entre los que destacan el método de las diferencias divididas de Newton, el método de interpolación polinómica de Lagrange y el método de interpolación polinómica de Hermite.

3.3.2.1.4 Interpolación por splines

Dado que encontrar el polinomio interpolador cuando se dispone de una secuencia elevada de puntos puede llevar a una tarea de cálculo desmesurada, es una opción bastante frecuente el descomponer toda la secuencia de puntos en segmentos y aplicar la interpolación polinómica a cada segmento. Al polinomio interpolador de cada segmento se le llama *spline*.

Ésta es la opción más común en procesado de señal debido a su bajo coste de implementación y a sus altas prestaciones. El principal problema del que adolece la interpolación por *splines* es que al unir dos *splines* consecutivos pueden aparecer problemas de derivabilidad en los extremos de los intervalos debido a que por estos puntos (los extremos) deben pasar los 2 *splines*. En caso de que la derivabilidad sea una condición necesaria existen otros tipos de interpolación no polinómica (interpolación racional [3-25], interpolación trigonométrica [3-26], mediante wavelets [3-27], etc.).

Por último cabe remarcar que los 2 tipos de interpolación comentados inicialmente son casos particulares de la interpolación mediante *splines*.

La interpolación por vecindad explicada en el apartado 3.3.2.1.1 se corresponde con un *spline* de orden 0 en el que el spline inicial es

$$P(x_0) = a_0 = y_0 \quad (3-22)$$

De esta forma se utiliza un solo punto x_k cuyo valor y_k se repite en todo el segmento hasta el siguiente punto x_{k+1} . En este caso en los extremos de los *splines* no existe derivabilidad, y de hecho ni siquiera continuidad (tal y como se ve en la Figura 3-9).

De la misma forma, la interpolación lineal explicada en el apartado 3.3.2.1.2 se corresponde con un *spline* de orden 1 en el que el spline inicial que se construye con los 2 primeros puntos (x_0, y_0) y (x_1, y_1) se obtendría a partir del siguiente sistema

$$\begin{cases} P(x_0) = a_0 + a_1 \cdot x_0 = y_0 \\ P(x_1) = a_0 + a_1 \cdot x_1 = y_1 \end{cases} \quad (3-23)$$

Y cuya solución, tal y como se muestra en (3-19), viene dada por:

$$a_0 = -x_0 \cdot \frac{y_1 - y_0}{x_1 - x_0} + y_0 = \frac{x_1 y_0 - x_0 y_1}{x_1 - x_0} \quad y \quad a_1 = \frac{y_1 - y_0}{x_1 - x_0} \quad (3-24)$$

Una vez obtenido el *spline* inicial, se calcula el *spline* siguiente a partir de la pareja de puntos (x_1, y_1) y (x_2, y_2) , y así sucesivamente. Este tipo de *splines* de orden 1 ofrece continuidad en los extremos, pero no derivabilidad (tal y como se puede apreciar en la Figura 3-10).

En el siguiente apartado se verá como la interpolación por *splines* puede interpretarse como un problema de filtrado, así como las opciones para implementarlo.

3.3.2.2 La interpolación aplicada al *resampling*

Desde el punto de vista del procesado digital de la señal la interpolación es la operación inversa al diezmado. La interpolación consiste en crear una nueva señal intercalando L muestras entre las muestras existentes de la señal de entrada. La nueva señal $x_{new}[n]$ equivale a haber remuestreado la señal original con una frecuencia de muestreo L veces superior, $f_{new} = L \cdot f_{old}$.

La relación entre las muestras de entrada y de salida para $L=3$ puede verse en la Figura 3-12 donde $x_{new}[n]$ es la señal ya interpolada, en ella las muestras en blanco son las nuevas muestras creadas de forma artificial y las muestras rellenas de negro corresponden a las de la señal original $x_{old}[n]$.

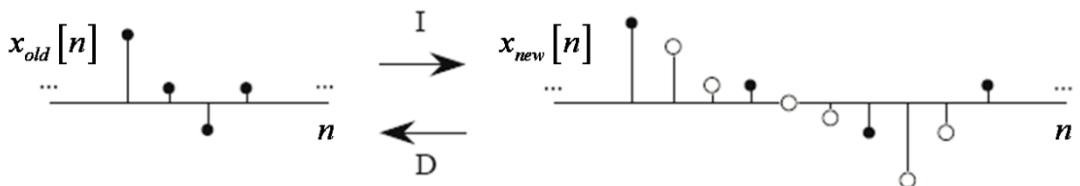


Figura 3-12: Relación entre interpolación y diezmado para $L=3$. (figura extraída y modificada de [3-2])

En la interpolación se hace necesaria la definición de una secuencia intermedia $v[n]$ que se crea intercalando $L-1$ muestras nulas entre cada dos muestras de la secuencia de entrada $x_{old}[n]$, este proceso se conoce como *zero padding*. Dicha secuencia intermedia se define como:

$$v[n] = \begin{cases} x_{old}\left[\frac{n}{L}\right] & n = 0, \pm L, \pm 2L, \dots \\ 0 & \text{resto} \end{cases} \quad (3-25)$$

y su transformada de Fourier puede expresarse como:

$$V(f) = \sum_{n=-\infty}^{\infty} v[n] e^{-j2\pi fn} = \sum_{n=-\infty}^{\infty} x\left[\frac{n}{L}\right] e^{-j2\pi fn} = \sum_{m=-\infty}^{\infty} x[m] e^{-j2\pi fLm} = X_{old}(Lf) \quad (3-26)$$

que expresa una compresión del eje frecuencial discreto provocando la aparición de réplicas comprimidas del espectro de $X_{old}(f)$ en las frecuencias múltiples de $\frac{1}{L}$ tal y como puede apreciarse en la Figura 3-13.

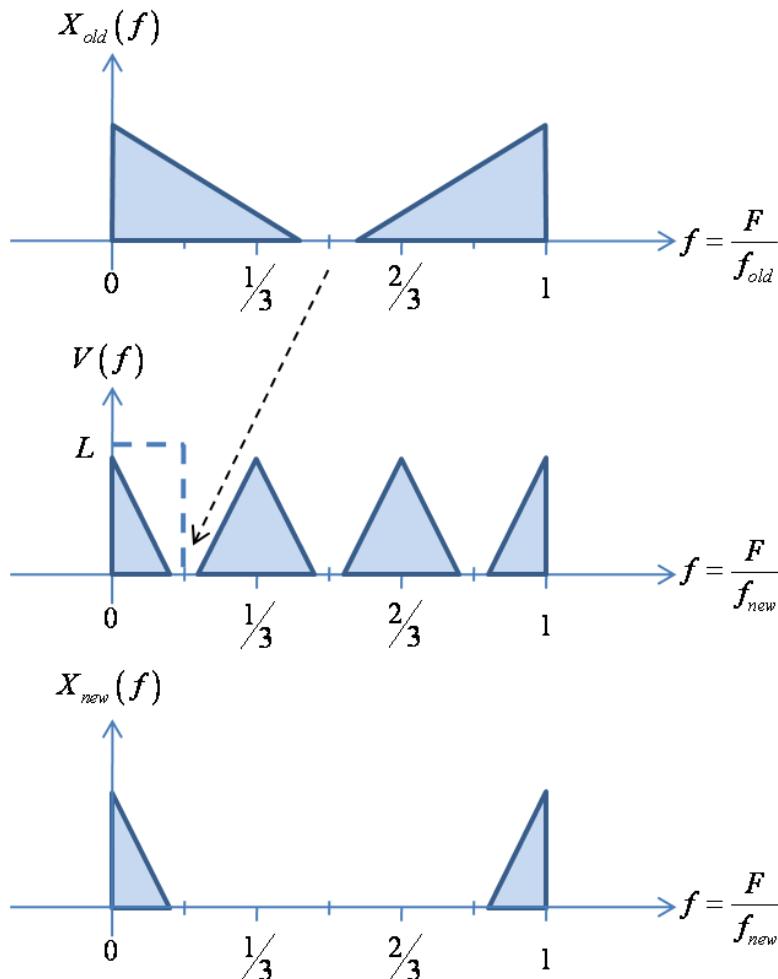


Figura 3-13: Transformadas de Fourier del proceso de interpolación con $L=3$. En línea discontinua se muestra el filtro paso bajo ideal de ganancia L .

Para obtener la secuencia interpolada se debe filtrar la secuencia intermedia $v[n]$ mediante un filtro paso bajo (denominado filtro interpolador) que elimine las réplicas comprimidas espectralmente de la transformada de $x_{old}[n]$ centradas en los múltiplos de $\frac{1}{L}$. Este filtro debería tener una frecuencia de corte de $f_{stop} = \frac{1}{2L}$ que corresponde a una frecuencia analógica de $F_{stop} = \frac{f_{old}}{2} = \frac{f_{new}}{2L}$ y una ganancia igual a L en la banda de paso

Así pues, el esquema definitivo de un interpolador es el que aparece en la Figura 3-14.

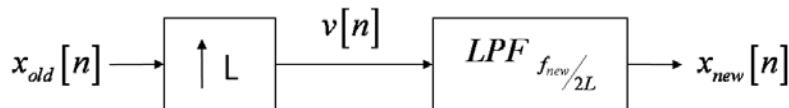


Figura 3-14: El proceso completo de interpolación con un filtro paso bajo.

Al igual que en el caso del diezmado, el problema de la interpolación se reduce a un ejercicio de diseño de un filtrado paso bajo.

En el caso ideal el filtro interpolador que eliminaría por completo todas las réplicas del espectro no deseadas tal y como se muestra en la Figura 3-13 se corresponde con una función sinc:

$$h(n) = \text{sinc}\left(\frac{n}{L}\right) \rightarrow H(f) = \begin{cases} L & \text{si } 0 \leq |f| < \frac{1}{L} \\ 0 & \text{resto} \end{cases} \quad (3-27)$$

El filtro de la ecuación (3-27) es un filtro IIR no causal. En la práctica sólo se puede aproximar el comportamiento de dicho filtro paso bajo, con lo que toda la precisión del proceso de interpolación recae en el diseño de la respuesta frecuencial del filtro.

La solución más habitual es aproximar el comportamiento del filtro ideal mediante un filtro FIR de fase lineal debido a la facilidad de su implementación.

Existen multitud de formas de aproximar dicho comportamiento, las cuales ya se han estudiado en el apartado dedicado al diezmado (punto 3.3.1.2).

Es de interés comentar que algunas de las aproximaciones que se pueden llevar a cabo del filtro paso bajo se corresponden con los métodos matemáticos comentados en el apartado anterior.

De esta forma, la interpolación polinómica mediante un *spline* de orden cero (interpolación por vecindad) comentada en los puntos 3.3.2.1.1 y 3.3.2.1.4 se

corresponde con la implementación de un pulso rectangular de L muestras, cuya formulación en el caso causal se puede ver en (3-28).

$$h(n) = \begin{cases} 1 & \text{si } 0 \leq n \leq L-1 \\ 0 & \text{resto} \end{cases} \rightarrow H(f) = \frac{\sin(L\pi f)}{\sin(\pi f)} \cdot e^{-j\pi f(L-1)} \quad (3-28)$$

En el caso de la interpolación polinómica mediante un *spline* de orden 1 (interpolación lineal) comentada en los puntos 3.3.2.1.2 y 3.3.2.1.4 el filtro interpolador se correspondería con un pulso triangular de $2L-1$ muestras tal y como se puede ver en (3-29).

$$h(n) = \begin{cases} 1 - \frac{|n|}{L} & \text{si } 0 \leq |n| \leq L-1 \\ 0 & \text{resto} \end{cases} \rightarrow H(f) = \frac{\sin^2(L\pi f)}{\sin^2(\pi f)} \quad (3-29)$$

El problema de este filtro es que es no causal, con lo que la salida de su implementación práctica estará retardada $L-1$ muestras con respecto a la del filtro no causal.

Desde el punto de vista frecuencial, el gran problema del que adolecen ambos filtros interpoladores (el de orden 0 y el de orden 1) es que debido a la amplitud de sus lóbulos secundarios no se consigue atenuar suficientemente las réplicas del espectro con lo que la interpolación tendrá un error mayor que el conseguido con los métodos descritos en el apartado 3.3.1.2. Aun así, la gran ventaja que tienen estos métodos (sobretodo el interpolador de orden 0) es su simplicidad de implementación y la velocidad de procesado.

Por último, nos detendremos en el análisis de la implementación de la estructura del filtro interpolador.

3.3.2.3 Filtros polifásicos

Si el filtro FIR a implementar tiene S etapas, veremos que no es necesario realizar S multiplicaciones para generar cada una de las muestras de la nueva señal $x_{new}[n]$ (muestras en blanco de la Figura 3-12) tal y como supondría una implementación directa del filtro.

Volviendo al ejemplo de interpolación con $L=3$, supongamos que hemos decidido realizar un filtro interpolador con $S=13$ coeficientes tal y como se muestra en la Figura 3-15a, en la que se observa un filtro FIR generado por el método de enventanado con ventana rectangular. El número de coeficientes elegido es impar ya que es la estructura óptima utilizada en los filtros interpoladores tal y como se explica en [3-28].

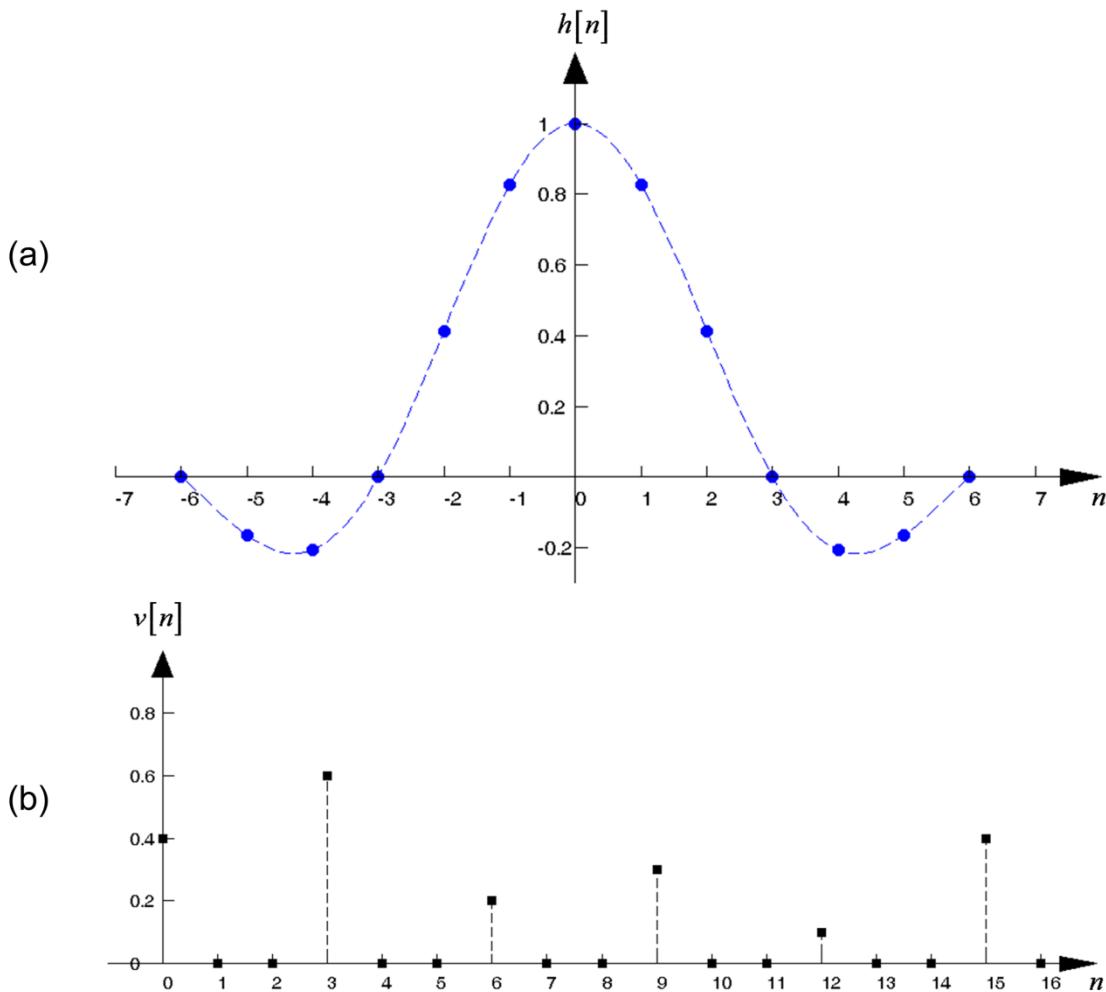


Figura 3-15: (a)Filtro FIR simétrico de $S=13$ coeficientes diseñado por enventanado y (b) señal $v[n]$ generada mediante zero padding con $L=3$ a la que se aplica el filtro.

El filtrado del espectro de $V(f)$ para generar la señal interpolada se corresponde en el dominio temporal con la convolución de la señal $v[n]$ con el filtro interpolador $h[n]$:

$$x_{new}[n] = v[n] * h[n] = \sum_{k=0}^{S-1} v[n-k]h[k] = \sum_{k=n-\frac{S-1}{2}}^{n+\frac{S-1}{2}} v[k]h[n-k] \quad (3-30)$$

De la fórmula de la convolución (3-30) se desprende inicialmente que para generar cada nueva muestra de la señal interpolada $x_{new}[n]$ serían necesarias S multiplicaciones, pero si se analiza la estructura de las señales implicadas se concluye que no es necesario, como se mostrará a continuación.

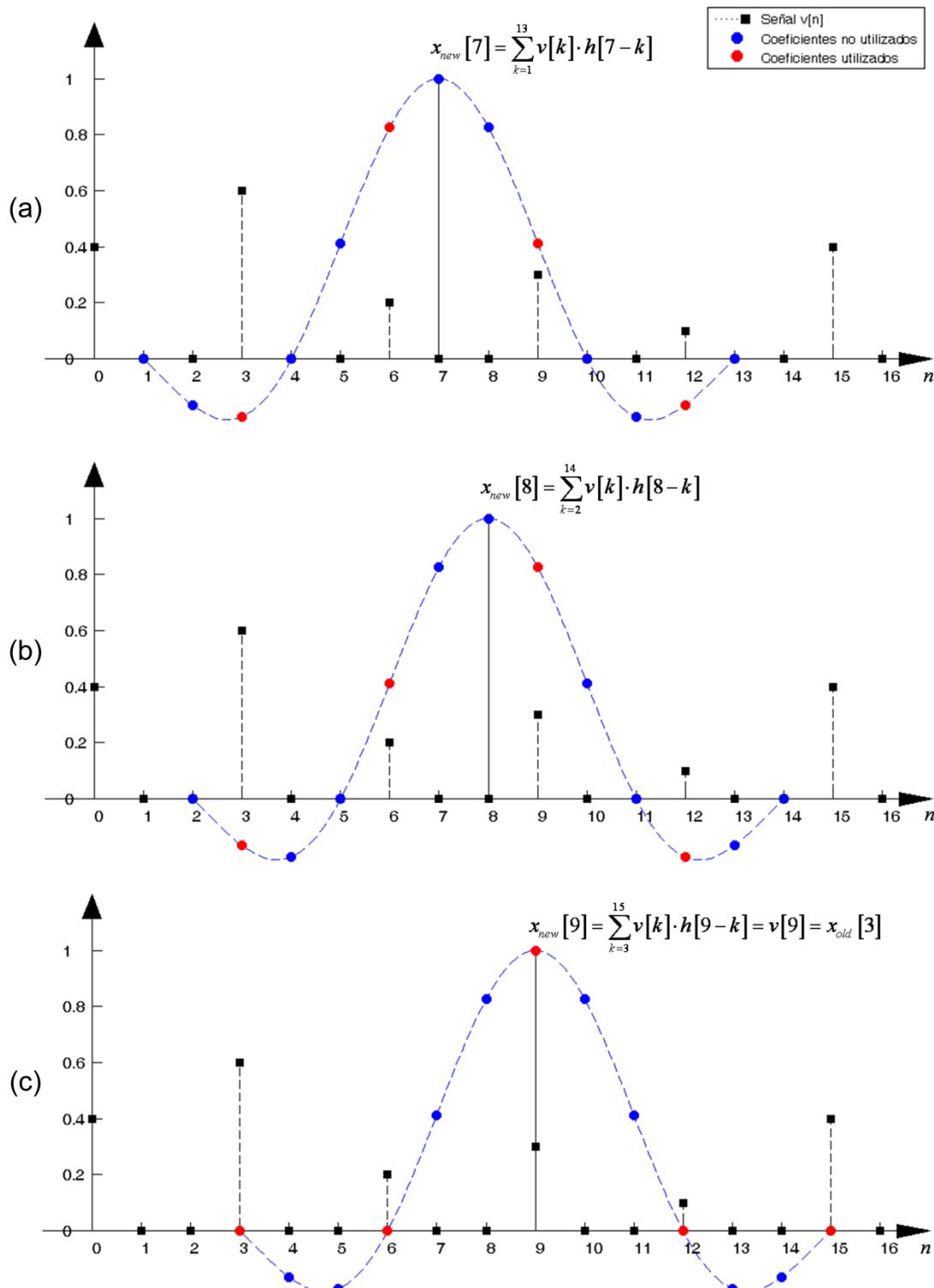


Figura 3-16: Aplicación del filtro FIR $h[n]$ sobre la señal $v[n]$ para calcular las muestras (a) $x_{new}[7]$, (b) $x_{new}[8]$ y (c) $x_{new}[9]$.

Analizando las gráficas de la Figura 3-16, que muestran cómo se aplica la respuesta impulsional de nuestro filtro interpolador a la señal $v[n]$ de la Figura 3-15b para generar las muestras $x_{new}[7]$ (en la Figura 3-16a), $x_{new}[8]$ (en la Figura 3-16b) y $x_{new}[9]$ (en la Figura 3-16c), se puede ver que no son necesarios todos los coeficientes del filtro en la generación de cada nueva muestra, tan sólo son necesarios los coeficientes marcados en rojo en cada caso, ya que el resto se multiplican por cero y no se tienen por qué implementar dichas multiplicaciones. De esta forma se reducen así el número de multiplicaciones ya que se necesitan 4 multiplicaciones para generar $x_{new}[7]$, el mismo número para generar $x_{new}[8]$ y a priori 5 multiplicaciones para generar $x_{new}[9]$ (aunque en realidad tan sólo se necesita una multiplicación debido a que de los cinco coeficientes del filtro implicados en la generación de $x_{new}[9]$ cuatro son nulos).

De esta forma se muestra que el número de multiplicaciones distintas de cero para cada nueva muestra $x_{new}[n]$ con un filtro interpolador FIR simétrico de orden impar no tiene una forma cerrada en función del número de coeficientes S y del factor de interpolación L en cada caso. Aun así se puede establecer que el número de multiplicaciones C_L será aproximadamente de:

$$C_L \approx \frac{S}{L} \text{ multiplicaciones/muestra de salida} \quad (3-31)$$

con lo que se reduce el coste computacional en un factor L respecto a la implementación tradicional del filtro FIR en forma directa.

Por otro lado se puede observar que el patrón se vuelve a repetir para la obtención de $x_{new}[10]$, $x_{new}[11]$ y $x_{new}[12]$, ya que en su generación se necesitarán los mismos coeficientes del filtro que se usaron para generar $x_{new}[7]$, $x_{new}[8]$ y $x_{new}[9]$ respectivamente. Como el patrón se repite cada $L=3$ muestras se pueden separar los $S=13$ coeficientes del filtro de la Figura 3-15a en 3 grupos tal y como se muestra en la Figura 3-17.

De esta forma, los coeficientes marcados con:

- se utilizan para generar las muestras de $x_{new}[n]$ con $n = 0, 3, 6, \dots, 3k$
- se utilizan para generar las muestras de $x_{new}[n]$ con $n = 1, 4, 7, \dots, 3k + 1$
- ◆ se utilizan para generar las muestras de $x_{new}[n]$ con $n = 2, 5, 8, \dots, 3k + 2$

Este tipo de implementación en la que se guardan los S coeficientes del filtro FIR original agrupados en L subfiltros de aproximadamente $\frac{S}{L}$ coeficientes que se

van aplicando alternadamente a lo largo del tiempo es denominada “filtrado FIR variante en el tiempo” o “filtrado polifásico” y fue introducido en [3-29].

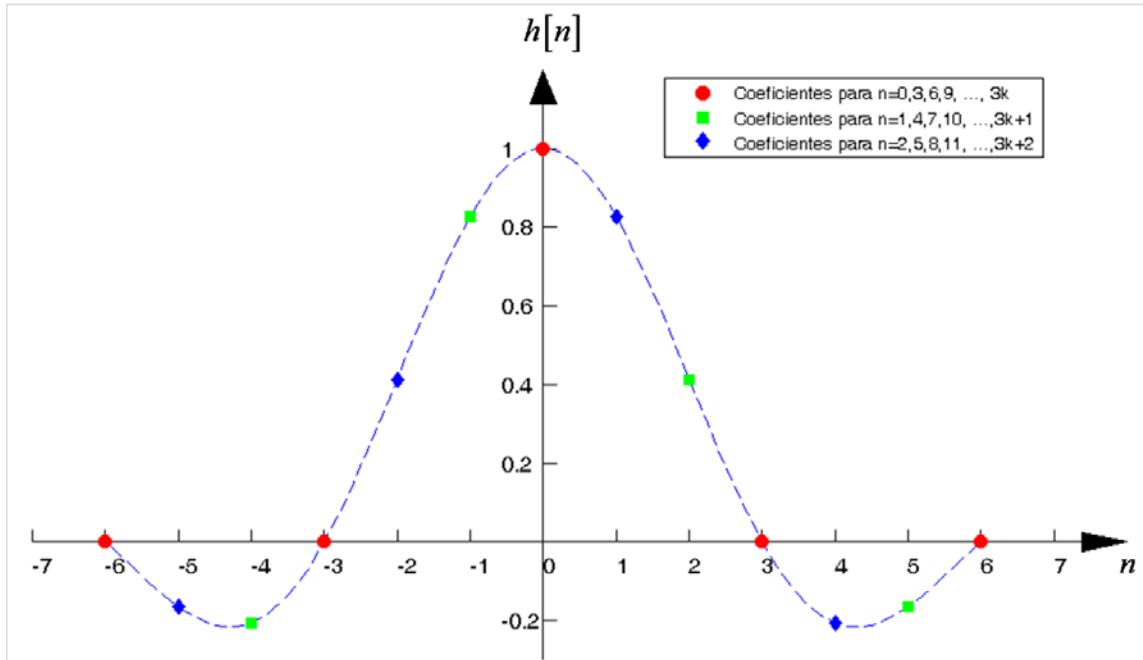


Figura 3-17: Los coeficientes del filtro FIR agrupados para implementar el banco de filtros polifásico.

Un último apunte sobre el filtro implementado es que, como se puede ver en la Figura 3-16c, cuando se calculan las muestras de $x_{new}[n]$ para $n = 0, 3, 6, \dots, 3k$ los coeficientes del filtro $h[n]$ que se utilizan (los coeficientes marcado con un círculo rojo en la Figura 3-17) valen todos 0 excepto el coeficiente central, cuyo valor es la unidad, provocando que la salida del filtro en los instantes de tiempo múltiplos de L tenga el mismo valor que las muestras originales:

$$x_{new}[n] = v[n] = x_{old}\left[\frac{n}{L}\right] \quad \text{para } n = 0, L, 2L, \dots, k \cdot L \quad (3-32)$$

3.3.3 Remuestreo mediante combinación de diezmado e interpolación: *rational resampling*

En los apartados anteriores se ha visto como los procesos de diezmado e interpolación sirven para reducir o aumentar el número de muestras de la señal original por un factor entero (D o L respectivamente), dando lugar a una señal remuestreada con una relación con respecto a la frecuencia de muestreo original de $f_{new} = \frac{f_{old}}{D}$ en el caso del diezmado y de $f_{new} = L \cdot f_{old}$ en el caso de la interpolación.

Este apartado muestra cómo se puede cambiar la frecuencia de muestreo por un factor racional $\frac{L}{D}$ realizando una interpolación por un factor L seguida de un diezmado por un factor D tal y como se muestra en la Figura 3-18a, dando lugar a una relación final entre frecuencias de $f_{new} = \frac{L}{D} \cdot f_{old}$.

Por ejemplo si se desea pasar de una frecuencia de muestreo original (f_{old}) de 48 kilomuestras por segundo (ksps) a otra frecuencia de muestreo (f_{new}) de 44,1 ksps, tal y como se realiza en los test del apartado 3.3.5, la relación entre ambas frecuencias es de $\frac{f_{new}}{f_{old}} = \frac{44,1}{48} = 0,91875$, que se puede llevar a cabo con una interpolación de $L=147$ muestras seguida de un diezmado de $D=160$ muestras ya que de la división de ambos factores se obtiene la misma relación $\frac{L}{D} = \frac{147}{160} = 0,91875$.

Comentar que si la relación entre frecuencias $\frac{L}{D}$ es menor que la unidad (como el caso del ejemplo) se está llevando a cabo un diezmado fraccional, mientras que si el factor es mayor que uno se denomina interpolación fraccional.

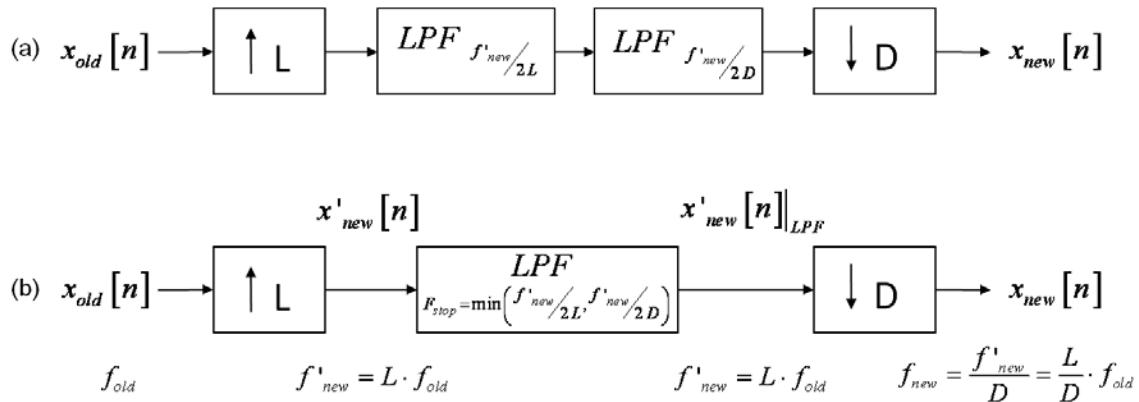


Figura 3-18: (a) Combinación de los procesos de diezmado e interpolación y (b) esquema final del “*rational resampler*” con las frecuencias de muestreo en cada punto del esquema.

El esquema final del remuestreador fraccional (*rational* o *fractional resampler*) lo podemos ver en la Figura 3-18b. En dicha figura podemos observar que al encadenar dos filtros paso bajo consecutivos es necesario solamente implementar el más restrictivo de los dos. Como se puede observar en la figura, ambos filtros se implementan después de la interpolación por L muestras, por lo que la frecuencia de muestreo a la que trabajan es $f'_{new} = L \cdot f_{old}$, como la frecuencia de corte del filtro interpolador es de $f'_{new}/2L$ y la del filtro antialiasing del diezmado es de $f'_{new}/2D$ (ambas expresadas en Hz), el filtro resultante deberá tener como frecuencia de corte el mínimo entre ambas cantidades:

$$F_{stop} = \min\left(\frac{f'_{new}}{2L}, \frac{f'_{new}}{2D}\right) = \min\left(\frac{f_{old}}{2}, \frac{L}{D} \cdot \frac{f_{old}}{2}\right) \quad (3-33)$$

De esta forma, depende de la relación entre las frecuencias de muestreo original y final $\frac{L}{D}$ el hecho de que se implemente el filtro interpolador o el filtro antialiasing del diezmado.

Por ejemplo, para una relación de $\frac{L}{D} = \frac{4}{3} = 1, \hat{3}$ (por tanto, interpolación) la frecuencia de corte del filtro será la del filtro interpolador $\left(F_{stop} = \frac{f_{old}}{2}\right)$, tal y como se observa en los espectros de la Figura 3-19. En dicha gráfica se aprecia que el filtro debe atenuar suficientemente las réplicas del espectro en las frecuencias múltiples de f_{old} con el fin de no introducir niveles de ruido elevados cuando se realice el diezmado.

En la práctica el coste computacional de cambiar la frecuencia de muestreo por un factor $\frac{L}{D}$ es menor que el producido por la suma de ambos procesos debido a la implementación de un único filtro tal y como se ha explicado anteriormente.

Concretamente, el número de multiplicaciones necesarias ($C_{L/D}$) para un filtro de S coeficientes queda reducida en un factor L si se implementa mediante un filtro polifásico, y además, como el diezmador descarta $D-1$ muestras de cada D , podemos reducir el número de multiplicaciones en un factor D quedando el número de multiplicaciones en:

$$C_{L/D} \approx \frac{S}{LD} \text{ multiplicaciones/muestra de salida} \quad (3-34)$$

Finalmente mencionar que existen multitud de textos donde se profundiza en algunos de los aspectos más complejos de la teoría del remuestreo, concretamente: la elección del número etapas del filtro en [3-5] y [3-30], el diseño de filtros FIR óptimos en [3-5] y [3-31] y en qué casos es mejor utilizar un filtro IIR en [3-31].

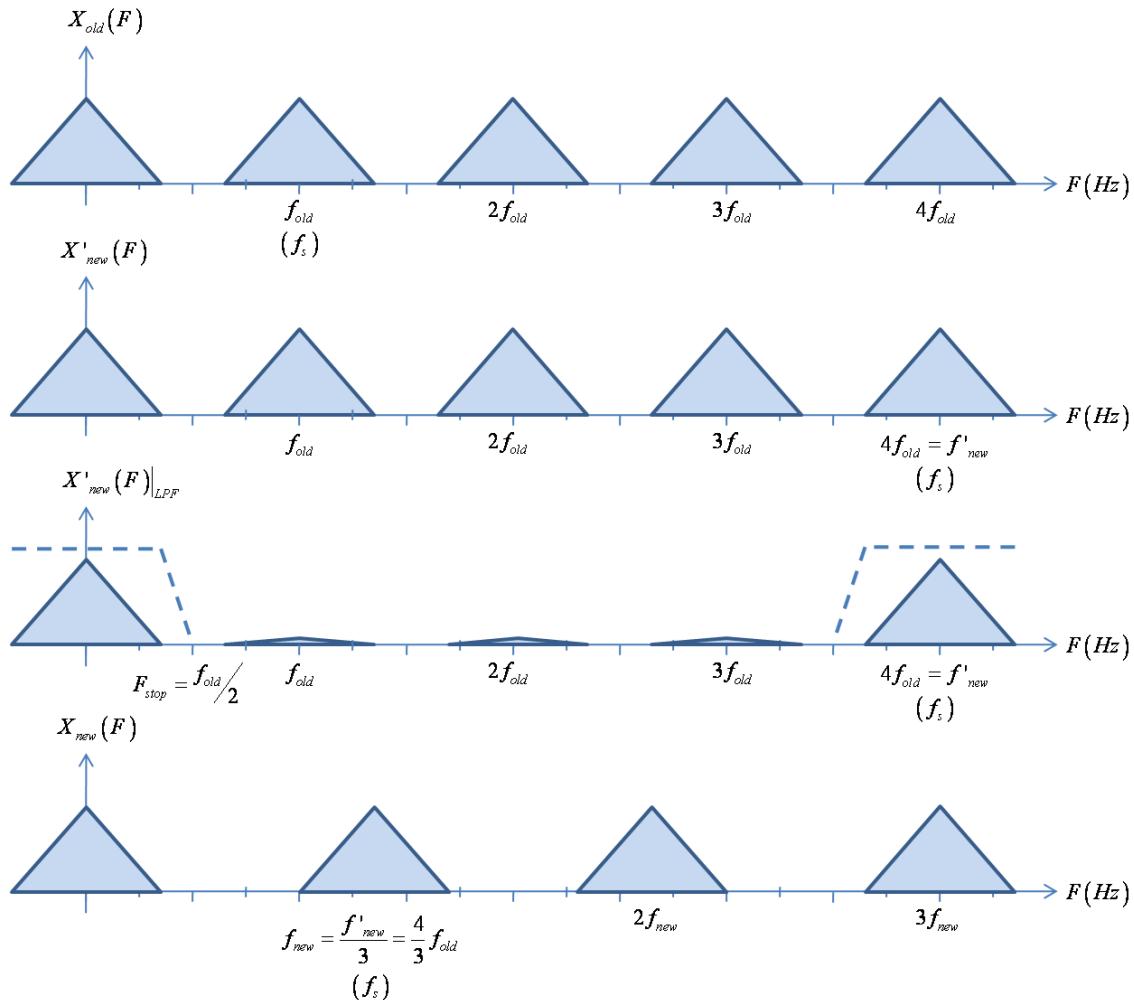


Figura 3-19: Espectros de las señales involucradas en el proceso de remuestreo fraccional con $L=4$ y $D=3$, haciendo hincapié en la evolución de la frecuencia de muestreo en cada caso.

3.3.4 Bloques analizados

En este apartado se resumen las características de los 3 bloques de *resampler* testeados:

`gr_rational_resampler_ccf [3-32]`

`gr_pbf_arb_resampler_ccf [3-34]`

`direct_resampler_conditioner [3-35]`

El primer bloque (`gr_rational_resampler`) es un bloque de GNU Radio que implementa un remuestreo fraccional (*rational* o *fractional resampling*) como el estudiado en el apartado 3.3.3.

Dicho bloque tiene como parámetros de entrada los factores de interpolación (L) y diezmado (D), que deben ser 2 números enteros, y un vector con los

coeficientes (TAPS) del filtro paso bajo (filtro interpolador/antialiasing). Los coeficientes del filtro se han generado mediante el método de enventanado explicado anteriormente en el apartado 3.3.1.2 a través de la función *gr_firdes* [3-33] de GNU Radio. El filtro se implementa mediante un filtro polifásico como los comentados en el apartado 3.3.2.3.

El segundo bloque (*pfb_arb_resampler*) es también un bloque de GNURadio de reciente implementación (se añadió en la release 3.3.0). Implementa un *resampler* arbitrario (real) mediante un banco de filtros polifásico y una segunda etapa de interpolación lineal.

Sus parámetros de entrada son: el ratio entre la frecuencia de muestreo inicial y la final (que puede ser cualquier magnitud real positiva), un vector con los TAPS del filtro paso bajo (generado con la misma función que en el apartado anterior) y un valor entero que es el número de filtros que implementará el banco de filtros polifásico y que a su vez es el valor de interpolación (L) del *resampler*. En la documentación del bloque se aconseja asignar un valor de 32 para dicho parámetro así como la utilización de la ventana de Blackman-Harris en el diseño del filtro paso bajo.

El bloque, a partir del ratio deseado y del factor de interpolación obtiene un factor de diezmado aproximado (dado que éste tiene que ser un valor entero) y para conseguir el ratio deseado entre las frecuencias de entrada y de salida realiza una interpolación lineal entre los valores de salida de 2 filtros consecutivos.

El tercer bloque (*direct_resampler_conditioner*) es un bloque que se ha diseñado para GNSS-SDR a partir de una implementación [3-36] existente en el software GPS-SDR [3-37].

El bloque realiza un remuestreo directo, es decir: en caso de que el factor de *resampling* sea menor que la unidad (diezmado) descarta aquellas muestras no deseadas hasta conseguir el ratio deseado y en caso de que el factor de *resampling* sea mayor que la unidad realiza una interpolación por *splines* de orden 0 (interpolación por vecindad). El bloque tan sólo necesita las frecuencias de muestreo de entrada (fs_{in}) y de salida (fs_{out}) que se le deben pasar a través del fichero de configuración de GNSS-SDR.

Aunque es previsible que la calidad del remuestreo sea inferior a la de los otros dos bloques a analizar es de suponer que su tiempo de procesado será mucho menor, característica deseable con el fin de conseguir que el software GNSS-SDR pueda procesar señales GNSS en tiempo real.

3.3.5 Tests y Resultados

La comparativa entre los 3 bloques comentados en el apartado anterior se ha llevado a cabo en base a 2 tipos de medidas: la calidad de la señal a la salida del bloque y el tiempo de procesado de dichos bloques.

3.3.5.1 Medidas de calidad

Las medidas de calidad que se toman se realizan comparando la señal obtenida a la salida de cada uno de los bloques analizados con la salida obtenida con

un bloque de referencia. Dicho bloque de referencia es el proporcionado por la función `resample()` [3-38] de Matlab.

La elección del *resampler* de Matlab como bloque de referencia es debido a que implementa el filtro paso bajo mediante un filtro polifásico (como los 2 primeros bloques que queremos analizar) pero además si no se le proporcionan los coeficientes del filtro, éstos se calculan internamente con la función `firls()` [3-39], que proporciona los coeficientes de un filtro FIR de fase lineal que implementa la solución de menor error cuadrático medio con respecto al filtro ideal (la aproximación por mínimos cuadrados explicada en el apartado 3.3.1.2).

Como señal de test se han generado 10240 muestras de una señal sinusoidal de frecuencia 1 kHz muestreada a 48 ksps (mediante Matlab). Posteriormente se le han añadido 40 muestras nulas al principio para poder detectar donde empieza la señal ya que en algunos bloques, debido a los retardos, se hace necesario sincronizar las señales, dando lugar a un total de 10280 muestras (equivalentes a 0.214 segundos de señal) y se han procesado con la función `resample()` de Matlab con unos factores de interpolación y diezmado de 147 y 160 respectivamente para obtener la misma señal a la salida con una frecuencia de muestreo de 44,1 ksps. Éste es un proceso de reconversión habitual en sistemas de audio, ya que los sistemas profesionales, así como el audio de los DVD, trabajan a 48 ksps mientras que el soporte de audio CD utiliza una frecuencia de muestreo de 44,1 ksps. Otro motivo por el cual se han elegido estos valores es porque proporcionan un ratio de conversión no entero, pero sí racional.

Posteriormente se ha procesado la misma señal original (de 48 ksps) con los 3 bloques comentados en el apartado anterior. Para los 2 *resamplers* de GNU Radio (los 2 primeros) se han analizado varios casos: se ha utilizado la ventana de Hanning en ambos *resamplers* y además la de Blackman-Harris para el `gr_pfb_arb_resampler` tal y como se aconseja en la documentación del bloque. En cada caso se ha analizado el comportamiento con una banda de transición para el filtro paso bajo del 10% y del 20% de la frecuencia de corte de dicho filtro. En cada uno de los casos se han tomado 3 medidas:

El error máximo (3-35), que es la mayor diferencia absoluta entre las muestras de la señal de referencia y la procesada por el bloque a analizar.

$$\text{Max Error} = \max |x_i - \hat{x}_i| \quad (3-35)$$

El sesgo o bias (3-36), definido como el valor absoluto de la media aritmética de las diferencias entre la señal de referencia y la estimada.

$$Bias = \left| \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i) \right| \quad (3-36)$$

Y el error cuadrático medio (MSE) (3-38), definido como la media aritmética del cuadrado de las diferencias entre la señal de referencia y la estimada por los bloques a analizar.

$$MSE = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2 \quad (3-37)$$

De esta última magnitud se dan también los valores en dB calculados como:

$$MSE(dB) = 10 \log_{10} \left(\frac{1}{N} \sum_{i=1}^N (x_i - \hat{x}_i)^2 \right) \quad (3-38)$$

Los resultados de las 3 medidas se pueden observar en la Tabla 3-1.

	Max Error	Bias	MSE	MSE (dB)	TAPS
gr_rational_resampling_hanning_window (transición LPF 10 %)	0,013612	0,000005	0,000077	-41,11	9921
gr_rational_resampling_hanning_window (transición LPF 20 %)	0,066750	0,000011	0,001719	-27,65	4961
gr_pfb_arb_resampling_hanning_window (transición LPF 10 %)	0,015763	0,000007	0,000106	-39,75	2159
gr_pfb_arb_resampling_hanning_window (transición LPF 20 %)	0,068453	0,000048	0,001804	-27,44	1081
gr_pfb_arb_resampling_harris_window (transición LPF 10 %)	0,015192	0,000009	0,000098	-40,11	15325
gr_pfb_arb_resampling_harris_window (transición LPF 20 %)	0,013946	0,000007	0,000082	-40,89	7663
direct_resampler_conditioner	0,130526	0,000032	0,002862	-25,43	

Tabla 3-1: Resultados de las medidas de calidad de los resamplers analizados.

En los resultados podemos observar que los bloques que incorporan el filtro antialiasing polifásico ofrecen mejores prestaciones (sobretodo en MSE) que el *direct_resampler_conditioner*. Esto es debido a que en este último bloque no se lleva a cabo ningún tipo de filtrado, o mejor dicho, realiza un filtrado de orden 0, que frecuencialmente se corresponde con una sinc (tal y como se vio en la ecuación (3-28)) y cuyos lóbulos secundarios provocan este aumento en el error.

El caso en el que se han obtenido las mejores prestaciones es el del *gr_rational_resampler* con ventana de Hanning y una banda de transición del 10%, seguido de cerca por el *gr_pfb_arb_resampler* con ventana de Blackman-Harris para ambos porcentajes de la banda de transición del filtro (10% y 20%), pero en los 3 casos el número de coeficientes para el filtro paso bajo es elevado, lo que hace suponer un tiempo de procesado excesivo.

Destacar que el bloque *gr_rational_resampler* es el que tiene las mejores prestaciones cuando se utiliza la ventana de Hanning. Este resultado era de esperar ya que la relación entre las frecuencias de muestreo inicial y final es fraccional ($\frac{147}{160}$), y este bloque ha sido diseñado específicamente para estos casos, a diferencia del *gr_pfb_arb_resampler* cuyo objetivo principal es poder realizar remuestreo con ratios reales. Además, como en el *gr_pfb_arb_resampler* se ha impuesto que el factor de interpolación fuera de 32 (valor que produce un descenso del número de TAPS del filtro paso bajo), el diezmado aproximado que obtiene es de 34, con lo que para conseguir el ratio deseado entre frecuencias debe realizar una interpolación lineal entre muestras consecutivas de las salidas de los filtros del banco de filtros polifásico.

Comentar también, la mejora en las prestaciones del *gr_pfb_arb_resampler* al utilizar la ventana de Harris con respecto a las del mismo *resampler* con la ventana Hanning tal y como se apuntaba en la teoría, cosa que se ve reflejada en la práctica en una disminución del MSE pero a costa de un aumento considerable en el número de coeficientes del filtro.

Finalmente comentar que aunque el *direct_resampler_conditioner* es el que obtiene las peores prestaciones éstas no están lejos de las que se obtienen con los dos *resamplers* polifásicos con ventana de Hanning y un porcentaje del 20% de la banda de transición del filtro, en concreto puede observarse que sólo se pierden unos 2 dB en la medida del MSE.

En las Figuras Figura 3-20, Figura 3-21, Figura 3-22 y Figura 3-23 puede observarse una representación gráfica de las señales implicadas en los test de remuestreo. En concreto en la Figura 3-23 pueden observarse los efectos en la señal remuestreada debido al muestreo por vecindad que lleva a cabo el *direct_resampler_conditioner* (muestras de color verde) frente a la remuestreo por mínimos cuadrados que lleva a cabo Matlab (muestras de color rojo) y ya a simple vista puede verse el error que se produce, que es mucho mayor que los casos de las Figuras Figura 3-20 y Figura 3-22, en la que el filtro polifásico tiene una banda de transición del 10%, pero no está tan lejos del error que se puede apreciar en la Figura 3-21 correspondiente al *gr_rational_resampler* con ventana de Hanning y banda de transición del 20%.

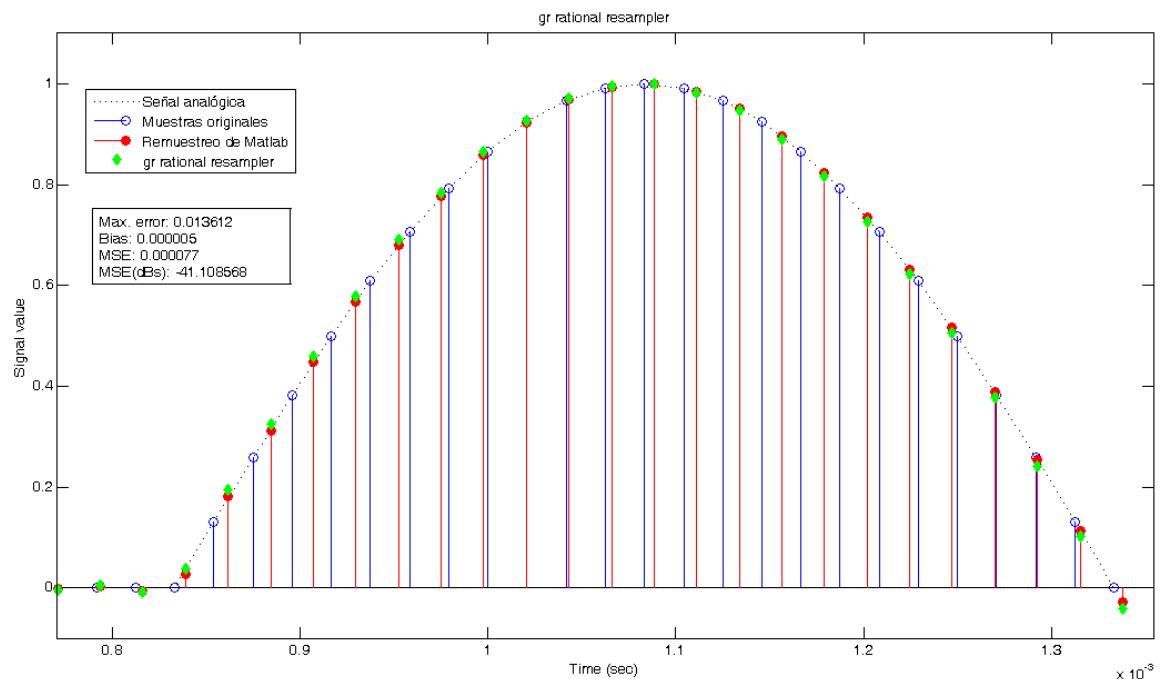


Figura 3-20: Remuestreo con *gr_rational_resampler*, ventana de Hanning y banda de transición del 10%.

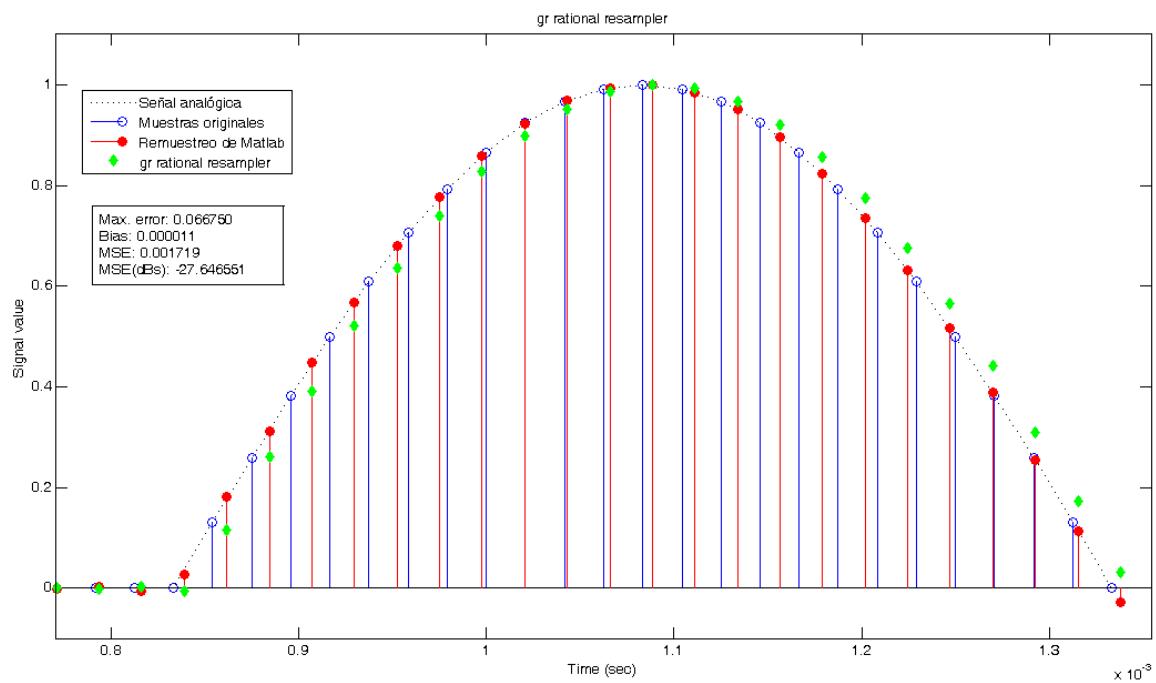


Figura 3-21: Remuestreo con *gr_rational_resampler*, ventana de Hanning y banda de transición del filtro del 20%, donde el error es apreciable a simple vista.

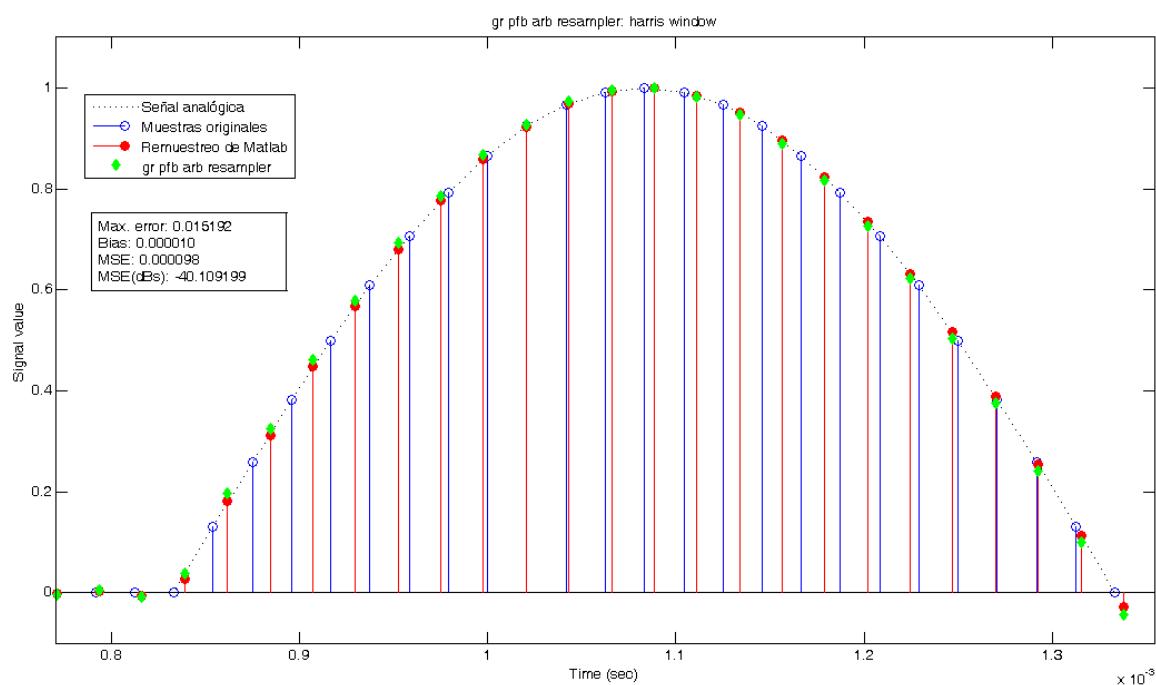


Figura 3-22: Remuestreo con *gr_pfb_arb_resampler*, ventana de Harris y banda de transición del filtro del 10%.

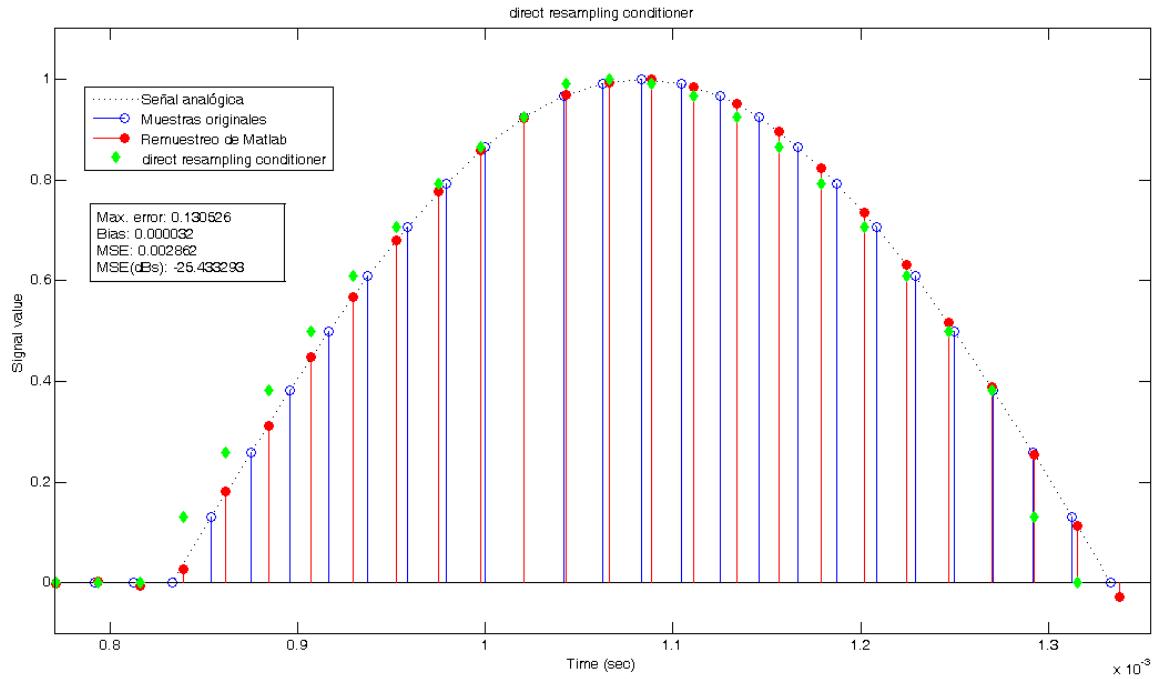


Figura 3-23: Remuestreo con *direct_resampler_conditioner* donde se puede apreciar el error producido por el remuestreo por vecindad.

3.3.5.2 Medidas de tiempo de procesado

Para las medidas de tiempo de los diferentes bloques se ha utilizado la función *time* de la Shell de Linux. Dicha función devuelve 3 tiempos referentes a la ejecución del programa: user time, system time y real time.

User time es la cantidad de tiempo que el programa pasa en ‘modo usuario’. También puede entenderse como la cantidad de tiempo que la CPU pasa realizando alguna acción para nuestro programa.

System time es la cantidad de tiempo que el programa pasa en ‘modo kernel’. Equivalentemente el *system time* es la cantidad de tiempo que la CPU gasta haciendo llamadas de sistema al kernel en nombre de nuestro programa.

Real time (también conocido como *elapsed time* o *wall time*) es el tiempo total desde que empieza la ejecución del programa hasta que termina. Mientras GNSS-SDR está ejecutándose hay otros programas que también lo están haciendo en background y que comparten los recursos (entre ellos la CPU) influyendo en que el tiempo de ejecución sea mayor. Este tiempo se tiene en cuenta en el *real time* pero no así en el *user* ni en el *system time*.

De esta forma, el tiempo de CPU viene dado por la suma del *user time* más el *system time*:

$$\text{CPU time} = \text{user time} + \text{system time} \quad (3-39)$$

Para llevar a cabo los tests se ha instanciado un objeto de una subclase de *gr_top_block* de GNU Radio [3-40] que implementa cada uno de los *resamplers* analizados. En dicho *top_block* se ha conectado un bloque *gr_file_source* [3-41] (que es el encargado de leer los datos del fichero de la señal a remuestrear generada sintéticamente) al bloque *resampler* a analizar, y la salida de éste a un bloque *null_sink* [3-42] (que no tiene más función que la de hacer de bloque terminal de los datos).

Así pues, hay que prever que en los resultados obtenidos, a parte del tiempo de procesado del *resampler* también se encuentran contabilizados los tiempos de instancia y conexión de los bloques GNU Radio así como los tiempos de lectura de fichero. En un intento de modelar de forma más fiel el tiempo de procesado se ha realizado un test en el que se ha conectado directamente el *file_sink* al *null_sink* con el fin de modelar el tiempo de lectura de fichero y restárselo a los tiempos obtenidos en los diferentes tests. De esta forma, los tiempos de proceso estimados se definen según

$$\text{estimated time} = \text{user time} + \text{system time} - \text{real time (of reading file)} \quad (3-40)$$

Capture parameters:
-Signal Type: GPS L1 C/A
-Satellites in view: 04, 13, 20, 23, 25, 27
-Position: Hawai
-IF: 0 MHz (baseband signal)
-Fs: 8 MHz
-Sample size: GNURadio complex = (32 bits float) + (32 bits float)i -> 8 Bytes per complex sample
-Sample multiplexing: I(4 Bytes),Q(4 Bytes),I,Q
-Number of samples: 64000000
-Capture length: 8 seconds

USRP parameters:
-RF Daughter board: DBSRX
-RF frequency: 1.57542 GHz
-RF gain: 0 dB (minimum gain)
-USRP Decimating factor= 8
-USRP sample size = 16 bits per channel, 2 Channels (DBSRX produces a baseband signal)
-Capture utility: gnuradio-3.2.2\gr-usrp\apps\usrp_rx_cfile.cc

Agilent GPS generator parameters:
-Carrier Frequency: 1.57542 GHz
-Carrier power: -50 dBm
-GPS Personality modulation on
- Scenario: Hawai
- Settings: Default

Tabla 3-2: Datos técnicos de la generación de la señal GPS utilizada en los test de remuestreo.

Para la realización de los tests de tiempo de procesado se han utilizado dos ficheros de señal correspondientes a 2 segundos y 8 segundos de señal GPS generada sintéticamente mediante un generador Agilent que posee una aplicación mediante la cual se puede generar un escenario GPS completo de hasta ocho satélites con datos de telemetría. La señal se ha capturado mediante una USRP a una frecuencia de muestreo de 8 Msps. Los datos técnicos de la captura pueden observarse en la Tabla 3-2.

Los test se han llevado a cabo para los diferentes escenarios del apartado anterior: los tres tipos de remuestreo con diferentes bandas de transición para el filtro antialiasing (10% y 20%) en los *resamplers* polifásicos y diferentes ventanas para el *gr_pfb_arb_resampler*. Como frecuencia de salida del *resampler* se han utilizado 2,048 Msps y 2,046 Msps (que es múltiplo de la frecuencia de chip de la señal GPS, 1,023 kHz).

Cada medida se ha llevado a cabo cien veces y en la Tabla 3-3 se presentan un promedio sobre esas cien realizaciones de cada una de las medidas.

En los resultados se pueden apreciar algunos datos interesantes. El dato más relevante es que el único *resampler* capaz de trabajar en el sistema GNSS-SDR en tiempo real es el *direct_resampler_conditioner*. Concretamente, si nos fijamos en el tiempo estimado de procesado (datos marcados en color rojo) podemos ver que tanto para la señal de 2 segundos como para la de 8 segundos su tiempo de procesado (0,49 y 1,74 segundos respectivamente) es muy inferior a la duración de la señal a procesar. El segundo bloque que produce un tiempo de procesado menor es el *gr_rational_resampler* con ventana de Hanning y banda de transición del filtro del 20%, aun así, el tiempo de procesado es muy cercano a la duración de la señal (entre 1,31 y 1,94 segundos para la señal de 2 segundos y 5,31 segundos para la señal de 8 segundos) aspecto que dificulta su inclusión en un sistema en tiempo real debido al poco tiempo de procesado que quedaría para el resto de bloques.

En el resto de bloques analizados el tiempo de procesado es superior a la duración de la señal, cosa que hace completamente inviable su inclusión en un sistema en tiempo real.

El segundo aspecto a comentar es que en los filtros polifásicos, el número de coeficientes del filtro depende de dos factores: de la magnitud de los factores de interpolación y muestreo (contra mayores son mayor es el número de coeficientes) y la banda de transición del filtro (concretamente puede apreciarse que cuando se pasa de un 20% a un 10% el número de coeficientes prácticamente se dobla en todos los casos). Este hecho es importante porque influye en el tiempo de procesado tal y como era de esperar (a mayor número de coeficientes mayor tiempo de procesado), aunque la relación no es directa y se ve influenciada por otros aspectos. Así pues, se puede observar que a igual número de coeficientes y de factores de interpolación, el *gr_pfb_arb_resampler* es mucho más lento que el *gr_rational_resampler* (12,13 segundos frente a 5,08 al procesar ambos la misma señal de 8 segundos con una banda de transición del 20%), esto puede ser debido a la interpolación lineal que lleva a posteriori el *gr_pfb_arb_resampler* tal y como se ha comentado en el apartado 3.3.4.

testing 2 seconds of data	user	system	real	estimated	fs_in (Hz)	fs_out (Hz)	interp	diez	TAPS LPF	LPF fs (MHz)	LPF fc (MHz)	LPF tx BW (MHz)
direct_resampler_conditioner_2s_2046	0,03	0,91	0,94	0,49	8000000	2046000						
direct_resampler_conditioner_2s_2048	0,03	0,91	0,94	0,49	8000000	2048000						
gr_rational_resampler_hann_2s_2046_10	1,31	3,45	4,79	4,31	8000000	2046000	1023	4000	248001	8184	1,023	0,1023
gr_rational_resampler_hann_2s_2046_20	0,61	1,77	2,41	1,94	8000000	2046000	1023	4000	124001	8184	1,023	0,2046
gr_rational_resampler_hann_2s_2048_10	0,37	2,71	3,09	2,64	8000000	2048000	32	125	7751	256	1,024	0,1024
gr_rational_resampler_hann_2s_2048_20	0,33	1,42	1,77	1,31	8000000	2048000	32	125	3875	256	1,024	0,2048
gr_pbf_arb_resampler_hann_2s_2046_10	2,30	2,64	4,98	4,49	8000000	2046000	32		7759	256	1,023	0,1023
gr_pbf_arb_resampler_hann_2s_2046_20	1,63	1,85	3,52	3,05	8000000	2046000	32		3879	256	1,023	0,2046
gr_pbf_arb_resampler_hann_2s_2048_10	2,31	2,64	4,98	4,51	8000000	2048000	32		7751	256	1,024	0,1024
gr_pbf_arb_resampler_hann_2s_2048_20	1,66	1,87	3,53	3,09	8000000	2048000	32		3875	256	1,024	0,2048
gr_pbf_arb_resampler_harris_2s_2046_10	19,66	3,24	23,15	22,46	8000000	2046000	32		55055	256	1,023	0,1023
gr_pbf_arb_resampler_harris_2s_2046_20	9,54	2,78	12,40	11,89	8000000	2046000	32		27527	256	1,023	0,2046
gr_pbf_arb_resampler_harris_2s_2048_10	19,68	3,27	23,26	22,52	8000000	2048000	32		55001	256	1,024	0,1024
gr_pbf_arb_resampler_harris_2s_2048_20	9,59	2,80	12,45	11,95	8000000	2048000	32		27501	256	1,024	0,2048
times_dc_test0 (read file)	0,01	0,43	0,44									

testing 8 seconds of data	user	system	real	estimated	fs_in (Hz)	fs_out (Hz)	interp	diez	TAPS LPF	LPF fs (MHz)	LPF fc (MHz)	LPF tr BW (MHz)
direct_resampler_conditioner_8s_2048	0,07	3,59	3,67	1,74	8000000	2048000						
gr_rational_resampler_hann_8s_2048_10	1,36	10,94	12,43	10,39	8000000	2048000	32	125	7751	256	1,024	0,1024
gr_rational_resampler_hann_8s_2048_20	1,23	5,77	7,03	5,08	8000000	2048000	32	125	3875	256	1,024	0,2048
gr_pbf_arb_resampler_hann_8s_2048_10	9,16	11,70	19,80	18,94	8000000	2048000	32		7751	256	1,024	0,1024
gr_pbf_arb_resampler_hann_8s_2048_20	6,67	7,38	14,13	12,13	8000000	2048000	32		3875	256	1,024	0,2048
gr_pbf_arb_resampler_harris_8s_2048_10	69,77	20,44	91,75	88,30	8000000	2048000	32		55001	256	1,024	0,1024
gr_pbf_arb_resampler_harris_8s_2048_20	28,07	20,81	49,16	46,97	8000000	2048000	32		27501	256	1,024	0,2048
times_dc_test0 (read file)	0,01	1,90	1,92									

Tabla 3-3: Medidas de tiempo de ejecución de los resamplers analizados.

3.4 Input Filter

El bloque *InputFilter* debe encargarse del filtrado de la señal de entrada (cuando sea necesario) así como de una posible bajada en frecuencia cuando la señal de entrada se encuentra a una frecuencia intermedia (IF) conocida.

Como se ha comentado al inicio de este capítulo, las señales que entran al bloque acondicionador de señal (Signal Conditioner) provenientes del ADC del cabezal de RF pueden estar demoduladas a banda base o encontrarse moduladas aún a una frecuencia intermedia (IF). Si esta frecuencia es conocida sería deseable eliminarla antes de entregar la señal al bloque de adquisición. Se hace necesaria por tanto la inclusión de un bloque en el acondicionador de señal que realice esta tarea. El bloque recibe el nombre de *InputFilter* debe incluir por tanto una bajada en frecuencia así como un filtro paso bajo que elimine las altas frecuencias no deseadas. Por otro lado, dado que cabe la posibilidad de que el filtrado implícito que realiza el remuestreador implementado (el *direct_resampler_conditioner* explicado en el apartado 3.3.4) sea insuficiente para el correcto funcionamiento del sistema completo bajo ciertas condiciones, sería deseable que el bloque *InputFilter* pudiera filtrar la señal (incluso si esta ya se encuentra demodulada a banda base) con el fin de mejorar las prestaciones del sistema global.

3.4.1 Bloques analizados

Para la realización de estas funciones se han diseñado dos implementaciones denominadas *Fir_Filter* (bloque que realiza un filtrado de la señal) y *Freq_Xlating_Fir_Filter* (que realiza una bajada en frecuencia y un filtrado). Dado que existen bloques en el repositorio de GNURadio que ofrecen las prestaciones necesarias para realizar las funciones que se desea implementar y además han sido altamente testeados, se ha decidido encapsular mediante sendos adaptadores los bloques existentes en dicho repositorio.

3.4.1.1 FIR Filter (*FirFilter*)

La clase *Fir_Filter* es un adaptador que encapsula al bloque de GNURadio *fir_filter_ccf* [3-45], dicha clase implementa un filtro FIR con entrada y salida complejas a partir de un vector con los coeficientes del filtro (TAPS). Para el cálculo de dichos coeficientes se ha utilizado la función *pm_remez* (también del repositorio de GNURadio [3-46]) que calcula la respuesta impulsional óptima del filtro según el método por aproximación de Chebyshev (también conocido como método de Parks-McClellan) explicado en el apartado 3.3.1.2 utilizando el algoritmo de Remez.

De esta forma, las opciones que se pueden modificar desde el fichero de configuración son:

- Número de coeficientes del filtro (*InputFilter.number_of_taps*).
- Número de bandas del filtro (*InputFilter.number_of_bands*).
- La frecuencia inicial y final de cada banda (*InputFilter.band1_begin*, *InputFilter.band1_end*, ...).

- La amplitud deseada en el inicio y el final de cada banda (*InputFilter.ampl1_begin*, *InputFilter.ampl1_end*, ...).
- El error aplicado a cada banda (*InputFilter.band1_error*, ...)
- El tipo de filtro (*InputFilter.filter_type*): "bandpass" (respuesta plana en bandas. Este es el valor por defecto), "hilbert" (para filtros de fase lineal con simetría impar) o "differentiator" (Respuesta en bandas proporcional a la frecuencia).
- *Grid density* (*InputFilter.grid_density*) determina la precisión con la que se implementará el filtro. El valor mínimo (valor por defecto) es 16, valores más altos, por el contrario, ralentizan el cálculo del filtro.

3.4.1.2 Frequency Translating FIR Filter (*Freq_Xlating_Fir_Filter*)

La clase *Freq_Xlating_Fir_Filter* es un adaptador que encapsula al bloque de GNURadio *freq_xlating_fir_filter_ccf* [3-47], dicha clase combina eficientemente una traslación de frecuencia, por lo general una bajada en frecuencia (*down-conversion*), con un filtro FIR (típicamente de paso bajo) y un diezmado. Es ideal para un filtro selector de canal y se puede utilizar de forma eficiente para seleccionar una señal de banda estrecha de una señal de entrada de banda ancha. Para el cálculo de los coeficientes del filtro FIR se ha vuelto a utilizar la función *pm_remez al igual que en la clase Fir_Filter*.

Las características que se pueden modificar a través del fichero de configuración son las mismas que para la clase *Fir_Filter* más las siguientes:

- La frecuencia intermedia (*InputFilter.IF*) que será desplazada a frecuencia 0.
- El factor de diezmado a aplicar (*InputFilter.decimation*) que debe ser un número entero (el valor por defecto es 1).

Pruebas realizadas con el sistema completo GNSS-SDR

Se han realizado varios test para comprobar la utilidad de los bloques en un sistema completo. De esta forma se han realizado 3 experimentos.

En primer lugar se ha utilizado el sistema completo GNSS-SDR para obtener posición en tiempo real. Para ello

3.5 Unit Test

Un test unitario (*Unit Test*) es un trozo de código desarrollado con el único objetivo de verificar que una rutina o función de nuestro código está funcionando según esperamos.

En definitiva, lo que un *Unit Test* pretende es tener trozos de código que se encargarán de testear por separado y de forma independiente cada uno de los métodos de las distintas clases que compongan el programa desarrollado, para que cuando se haga alguna modificación en el código, posibles errores derivados de esa modificación puedan ser identificados y corregidos de manera rápida y eficaz.

Por último comentar que se ha implementado un unit test en base a las simulaciones realizadas sobre los 3 *resamplers* con el fin de verificar la validez del diseño de futuros *resamplers*. De esta forma se han definido unos umbrales respecto a las 3 medidas de calidad analizadas en el apartado 3.3.5.1 con el fin de que cualquier *resampler* que se incorpore al software GNSS-SDR asegure una funcionalidad básica.

3.6 Conclusiones

Se ha podido ver que el *resampler* implementado (*direct_resampler_conditioner*) cumple con los requerimientos para los que fue diseñado. Así se ha visto que si bien es la opción de peor calidad ésta no dista mucho de la de algunos *resamplers* polifásicos, pero por el contrario, su tiempo de procesado es muy inferior al del resto de soluciones analizadas por lo que su viabilidad para el sistema GNSS-SDR con el fin de poder analizar señales GNSS en tiempo real es total.

3.7 Bibliografía

[3-1] R.G. Lyons, "Understanding Digital Signal Processing", Pearson Education, 1st edition, pp 303-317, noviembre de 1996.

[3-2] J.B. Mariño Acebal, F. Vallverdú Bayés, J.A. Rodríguez Fonollosa, A. Moreno Bilbao, "Tratamiento digital de la señal. Una introducción experimental", Edicions UPC, pp 111-119, 2^a edición, 1996

[3-3] H. Nyquist, "Certain topics in telegraph transmission theory", Transactions of the AIEE, vol. 47, pp 617-644, abril de 1928.

[3-4] C. E. Shannon, "Communication in the presence of noise," Proc. Institute of Radio Engineers, vol. 37, no.1, pp 10-21, enero de 1949.

[3-5] R.E. Crochiere, L.R. Rabiner , "Optimum FIR Digital Implementations for Decimation, Interpolation, and Narrow Band Filtering", IEEE Transactions on Acoustics, Speech and Signal Processing. Vol. ASSP-23, No. 5, octubre de 1975.

[3-6] M. G. Ballanger, "Computation Rate and Storage Estimation in Multirate Digital Filtering with Half-Band Filters", IEEE Transactions on Acoustics, Speech and Signal Processing. Vol. ASSP-25, No. 4, agosto de 1977.

[3-7] A.V. Oppenheim, R.W. Schafer, "Discrete-Time Signal Processing", Prentice Hall, pp 439-540, 3^a edición, agosto de 2009.

[3-8] L. R. Rabiner, "Techniques for designing finite-duration impulse response digital filters," IEEE Transactions on Communications Technology, Vol. COM-19, No. 2, pp. 188-195, abril de 1971.

[3-9] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Window-based finite impulse response filter design" <http://www.mathworks.com/help/toolbox/signal/fir1.html>, Comprobado: 12 de marzo de 2012.

[3-10] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Frequency sampling-based finite impulse response filter design" <http://www.mathworks.com/help/toolbox/signal/fir2.html>, Comprobado: 12 de marzo de 2012.

[3-11] T.W. Parks, J.H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase", IEEE Transactions on Circuit Theory. Vol. CT-19, No. 2, marzo de 1972.

[3-12] E. Ya. Remez "Sur le calcul effectif des polynômes d'approximation de Tschebyscheff." C. P. Paris, 337-340, 1934.

[3-13] O. Herrmann, L.R. Rabiner, y D.S.K. Chan, "Practical Design Rules for Optimum Finite Impulse Response Low-pass Digital Filters," Bell Syst. Tech. J., vol. 52, pp. 769-799, julio-agosto de 1973.

[3-14] J.F. Kaiser and K. Steiglitz, "Design of FIR Filters with Flatness Constraints," Proc. 1983 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Boston, Mass., pp. 197-200, abril de 1983.

[3-15] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Parks-McClellan optimal FIR filter order estimation" <http://www.mathworks.com/help/toolbox/signal/firpmord.html>, Comprobado: 12 de marzo de 2012.

[3-16] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Parks-McClellan optimal FIR filter design" <http://www.mathworks.com/help/toolbox/signal/firpm.html>, Comprobado: 12 de marzo de 2012.

[3-17] T. W. Parks, C. S. Burrus, "Digital Filter Design", John Wiley & Sons, pp. 54-83, 1987.

[3-18] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Least square linear-phase FIR filter design" <http://www.mathworks.com/help/toolbox/dsp/ref/firls.html>, Comprobado: 12 de marzo de 2012.

[3-19] Wikipedia, the free encyclopedia. Interpolation. <http://en.wikipedia.org/wiki/Interpolation>, Comprobado: 12 de marzo de 2012.

[3-20] R.L. Burden, J.D. Faires, "Numerical Analysis", Brooks Cole, 9 edition pp 106-117, agosto de 2010.

[3-21] W.G. Horner. "A new method of solving numerical equations of all orders, by continuous approximation.", Philosophical Transactions of the Royal Society of London, vol. 109, pp. 308-335, julio de 1819

[3-22] Å . Björck, V. Pereyra. "Solution of Vandermonde Systems of Equations". Mathematics of Computation (American Mathematical Society) en 1970.

[3-23] N. J. Higham. "Fast Solution of Vandermonde-Like Systems Involving Orthogonal Polynomials". IMA Journal of Numerical Analysis 8 pp: 473–486 en 1988.

[3-24] D. Calvetti, L. Reichel. "Fast Inversion of Vanderomnde-Like Matrices Involving Orthogonal Polynomials". BIT Numerical Mathematics 33 pp. 473–484 en 1993.

[3-25] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery "Section 3.4. Rational Function Interpolation and Extrapolation", Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press 2007.

[3-26] T.N.T. GOODMAN, A. SHARMA "Trigonometric Interpolation", Proceedings of the Edinburgh Mathematical Society (Series 2), 35: pp. 457-472, 1992.

[3-27] I. Daubechies, "Ten Lectures on Wavelets", Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[3-28] R.W. Schafer, L.R. Rabiner "A Digital Signal Processing Approach to Interpolation", Proceedings of the IEEE, Vol. 61, No. 6, junio de 1973.

[3-29] M. Bellanger, G. Bonnerot, M. Coudreuse "Digital filtering by polyphase network: Application to sample rate alteration and filter banks", , IEEE Transactions on Acoustics, Speech and Signal Processing. Vol. 24, pp. 109-114 abril de 1976.

[3-30] R.E. Crochiere, L.R. Rabiner , "Decimation and Interpolation of Digital Signals – A Tutorial Review", Proceedings of IEEE, Vol. 69, No. 3, marzo de 1981

[3-31] R.E. Crochiere, L.R. Rabiner , "Further Considerations in the Design of Decimators and Interpolators", IEEE Transactions on Acoustics, Speech and Signal Processing. Vol. ASSP-24, No. 4, agosto de 1976

[3-32] GNU Radio. GNU Radio C++ Signal Processing Blocks. Filters. Rational Resampler.
http://gnuradio.org/doc/doxygen/classgr_rational_resampler_base_ccf.html, Comprobado: 26 de marzo de 2012.

[3-33] GNU Radio. Digital Filter Design. Firdes.
http://gnuradio.org/doc/doxygen/classgr_firdes.html, Comprobado: 26 de marzo de 2012.

[3-34] GNU Radio. GNU Radio C++ Signal Processing Blocks. Filters. PFB ARB Resampler.
http://gnuradio.org/redmine/projects/gnuradio/repository/revisions/master/entry/gnuradio-core/src/lib/filter/gr_pfb_arb_resampler_ccf.h, Comprobado: 26 de marzo de 2012.

[3-35] GNSS-SDR. GNSS-SDR C++ Blocks. Algorithms. Conditioners. Direct Resampler.
http://sourceforge.net/p/gnss-sdr/code/339/tree/trunk/src/algorithms/resampler/adapters/direct_resampler_conditioner.h, Comprobado: 21 de febrero de 2013.

[3-36] GPS-SDR. Blocks. GPS Source. Resample_USRP_V1.
https://github.com/gps-sdr/gps-sdr/blob/master/objects/gps_source.h, Comprobado: 21 de febrero de 2013.

[3-37] GPS-SDR. <http://www.gps-sdr.com:8080/>, actualmente en desuso, repositorio <https://github.com/gps-sdr>, Comprobado: 21 de febrero de 2013.

[3-38] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, "Change sampling rate by rational factor"

<http://www.mathworks.es/help/toolbox/signal/ref/resample.html>, Comprobado: 26 de marzo de 2012.

[3-39] The MathWorks Inc., MATLAB R2011a Documentation, Signal Processing Toolbox, “Least square linear-phase FIR filter design” <http://www.mathworks.es/help/toolbox/signal/ref/firls.html>, Comprobado: 26 de marzo de 2012.

[3-40] GNU Radio. GNU Radio C++ Signal Processing Blocks. Top Block and Hierarchical Block Base Classes. http://gnuradio.org/doc/doxygen/classgr_top_block.html, Comprobado: 26 de marzo de 2012.

[3-41] GNU Radio. GNU Radio C++ Signal Processing Blocks. Signal Sources. File Source. http://gnuradio.org/doc/doxygen/classgr_file_source.html, Comprobado: 26 de marzo de 2012. , Comprobado: 26 de marzo de 2012.

[3-42] GNU Radio. GNU Radio C++ Signal Processing Blocks. Signal Sinks. Null Sink. http://gnuradio.org/doc/doxygen/classgr_null_sink.html, Comprobado: 26 de marzo de 2012. , Comprobado: 26 de marzo de 2012.

[3-43] Agilent Technologies. “Agilent E4438C ESG Vector Signal Generator datasheet”, diciembre 2006 <http://cp.literature.agilent.com/litweb/pdf/5988-4039EN.pdf>, Comprobado: 26 de marzo de 2012.

[3-44] Agilent Technologies. “Agilent Option 409 {GPS} Personality for the {E4438C ESG} Vector Signal Generator”, octubre 2007 <http://cp.literature.agilent.com/litweb/pdf/5988-6256EN.pdf>, Comprobado: 26 de marzo de 2012.

[3-45] GNU Radio. GNU Radio C++ Signal Processing Blocks. Filters. Classes. FIR Filter. http://gnuradio.org/doc/doxygen/classgr_1_1filter_1_1fir_filter_ccf.html. Comprobado: 26 de Febrero de 2013.

[3-46] GNU Radio. GNU Radio C++ Signal Processing Blocks. Filters. Functions. Parks-McClellan FIR filter design using Remez algorithm. http://gnuradio.org/doc/doxygen/group_filter_design.html#qaf13aef1a29a91a0bfc06adeca11dd021. Comprobado: 26 de Febrero de 2013.

[3-47] GNU Radio. GNU Radio C++ Signal Processing Blocks. Filters. Classes. Freq Xlating FIR Filter. http://gnuradio.org/doc/doxygen/classgr_1_1filter_1_1freq_xlating_fir_filter_ccf.html. Comprobado: 26 de Febrero de 2013.

4 GNSS-SDR: ACQUISITION

4.1 Resumen

En este capítulo se analizará el bloque de adquisición de señal (*Acquisition*) que incluyen los canales del sistema GNSS-SDR. En primer lugar se realiza una introducción al concepto de adquisición. Posteriormente se describen tres tipos diferentes de adquisición, haciéndolo de forma más exhaustiva para el método implementado (*Parallel Code Phase Search Acquisition*). A continuación se profundiza en el cálculo teórico del umbral de detección, entrando en detalle en la caracterización de los estadísticos de prueba para las dos hipótesis analizadas, obteniendo sus funciones de densidad de probabilidad (PDF) teóricas y las diferentes probabilidades que definen el rendimiento del algoritmo de adquisición. Posteriormente se explican los bloques de adquisición implementados en GNSS-SDR. Por último se analizan los test utilizados a fin de validar los resultados teóricos y se exponen las conclusiones obtenidas.

4.2 Introducción

Tal y como se describe en [4-1] el propósito del proceso de adquisición de cualquier sistema GNSS es determinar qué satélites son visibles por el receptor y realizar una primera estimación de los denominados parámetros de sincronización, que se corresponden con el retardo o fase de código (*code phase*) τ_i y la frecuencia portadora de la señal que llega a la entrada del bloque canal f_{d_i} , que puede tratarse de la frecuencia intermedia IF más la denominada frecuencia Doppler, o tan sólo la frecuencia Doppler si la IF ha sido eliminada en el bloque acondicionador de señal explicado en el capítulo anterior. Esta estimación se refina posteriormente en el bloque de *Tracking*, y permite desmodular los bits de la señal mensaje de navegación que emiten los satélites. Cuando se detectan cuatro o más satélites, y el receptor obtiene los datos de sus efemérides, el sistema puede resolver la ecuación que determina la posición.

4.2.1 Tipos de arranque del algoritmo de adquisición

El tiempo transcurrido entre que se inicia el proceso de adquisición y el cálculo de la primera posición se conoce como *Time To First Fix* (TTFF), y su promedio se utiliza frecuentemente como una figura de mérito del receptor.

En vista de lo explicado en los párrafos anteriores, la primera operación que debe realizar el bloque de adquisición es determinar qué satélites son visibles. Esta

lista teórica de satélites visibles se elabora en función de la información a priori disponible en el receptor. Según [4-2] se pueden dar tres situaciones diferentes:

- *Cold Start* (Arranque en frío): Cuando el receptor no tiene información sobre su posición ni el almanaque de satélites, el procedimiento de adquisición se debe aplicar a todos los posibles satélites en el espacio. Cada canal del receptor se asigna a un satélite, y si la señal de dicho satélite no se encuentra después de un período de observación determinado se cambia el canal a otro código de satélite. Si se encuentra la señal, la información se pasa al algoritmo de *tracking*. Este procedimiento también se conoce como "sky search" (búsqueda en el cielo).
- *Warm Start* (Arranque tibio): Cuando el receptor dispone de su ubicación aproximada, la hora aproximada del día, y una emisión del almanaque registrada recientemente, puede llevar a cabo una estimación de los satélites que son visibles. Esto reduce aproximadamente a la mitad el número total de satélites que se deben buscar respecto a un arranque en frío y por tanto acelera el proceso de adquisición. En este caso el receptor sólo necesita la actualización de las efemérides y de las señales temporales. Los receptores modernos tienen una pequeña batería y una memoria interna donde se almacenan los datos del último posicionamiento positivo que se ha llevado a cabo. Cuando se vuelve a arrancar el sistema estos datos se utilizan como punto de partida de la adquisición.
- *Hot Start* (Arranque en caliente): Cuando el receptor está siguiendo a un satélite y la línea de visibilidad con el satélite se ve interrumpida por un período corto de tiempo (al conducir a través de un túnel, bajo los árboles en un bosque, ...) pero los datos de las efemérides y el almanaque siguen siendo válidos, no es necesario obtener los datos de las efemérides de nuevo y la posición se puede volver a calcular después de realizar simplemente una adquisición de la señal. Este caso se conoce comúnmente como re-adquisición.

En el software GNSS-SDR se han implementado hasta la fecha dos tipos de arranque, el arranque en frío que se ejecuta al iniciar el software y el arranque en caliente, que se implementa en el proceso de re-adquisición en la máquina de estados que controla el bloque canal cuando el proceso de *tracking* se pierde, tal y como se verá en el Capítulo 5. Además, en el momento de la escritura de este proyecto se está llevando a cabo la implementación de una adquisición asistida, que permitirá al receptor conectarse a un servidor de internet que le proporcione la lista de satélites visibles por el mismo.

4.2.2 Modelo de señal

Para entender el funcionamiento de la adquisición así como los diferentes métodos que se pueden implementar para llevarla a cabo se hace necesario establecer un modelo para la señal que llega al canal proporcionada por el acondicionador de señal (*Signal Conditioner*) estudiado en el Capítulo 3. Considerando que a la antena del receptor han llegado las señales provenientes de M satélites, el modelo de señal banda base presente en la entrada del bloque de adquisición puede tener la siguiente forma simplificada tal y como se utiliza en [4-2]:

$$\begin{aligned}
 x(t_m) &= \sum_{i=1}^M a_i \sum_{l=-\infty}^{\infty} B_i(l) s_i(t_m - \tau_i - lT_b) e^{j2\pi f_{d_i} t_m} + n(t_m) = \\
 &= \sum_{i=1}^M a_i \sum_{l=-\infty}^{\infty} B_i(l) \sum_{u=0}^{N_c-1} \sum_{k=0}^{L_c-1} c_k^i p_i(t_m - \tau_i - kT_c - uT_{PRN} - lT_b) e^{j2\pi f_{d_i} t_m} + n(t_m)
 \end{aligned} \tag{4-1}$$

Donde:

- $t_m = mT_s$ siendo m una variable discreta y $f_s = 1/T_s$ la frecuencia de muestreo.
- a_i es la amplitud compleja del satélite i -ésimo, y se asume constante durante el tiempo de adquisición. De esta forma, la potencia del satélite i -ésimo se puede definir como $P_i = |a_i|^2$, donde $|\bullet|$ es el operador módulo.
- $B_i(l)$ son los bits de navegación emitidos por el satélite i -ésimo de valores $\{\pm 1\}$, y T_b el periodo de dichos bits (20 milisegundos en el caso de la señal GPS L1 C/A).
- $s_i(t_m - \tau_i) e^{j2\pi f_i t_m}$ es la señal GNSS banda base compleja DS-CDMA del satélite i -ésimo recibido. En este apartado consideramos la modulación BPSK para la señal DS-CDMA en la que $s_i(t) = \sum_{u=0}^{N_c-1} \sum_{k=0}^{L_c-1} c_k^i p_i(t - kT_c - uT_{PRN})$, donde c_k^i es el código de ensanchamiento (spreading code chips) o código PRN (*Pseudo Random Noise*) propio de cada satélite de valores $\{\pm 1\}$, $p_i(t)$ es el denominado pulso conformador de duración T_c , y T_c es el periodo de chip ($1/1.023 \mu s$ en el caso de la señal GPS L1 C/A). El código PRN se repite cada L_c chips, formando así las palabras código (*codewords*) que expanden a su vez el periodo de bit al repetirse $N_c = \frac{T_b}{T_{PRN}}$ veces durante un periodo de bit. En el caso de la señal GPS L1 C/A $p_i(t)$ es un pulso rectangular y los valores que se dan son de $L_c = 1023$ chips por codeword y $N_c = 20$ codewords por bit [4-3].
- f_{d_i} y τ_i son el desplazamiento Doppler y el retardo de código para el satélite i -ésimo respectivamente, denominados parámetros de sincronización.
- $n(t)$ es la señal de ruido. Dicho ruido es un proceso complejo, estacionario, circularmente simétrico y AWGN (*Additive White Gaussian Noise*) de media nula y varianza σ^2 , el cual modela al ruido térmico.

En pro de la simplicidad del modelo no se incluyen el efecto multicamino ni el de señales interferentes.

Como se ha comentado al inicio de este apartado el objetivo de la adquisición es obtener una estimación inicial tanto del retardo de código τ_i como del desplazamiento frecuencial f_{d_i} con el fin de eliminarlos de la señal para poder desmodular los bits del mensaje de navegación gracias a un posterior refinamiento de esta estimación. Para conseguir esta estimación se aprovechan las propiedades de los códigos PRN de la señal DS-CDMA. En [4-4] se explica que dichos códigos cumplen que la autocorrelación de un código tiene un máximo mucho mayor que el resto de valores de la misma y que el de todos los valores de la correlación cruzada entre los códigos de 2 satélites distintos. Es decir, sean c^k y c^i los códigos PRN de 2 satélites diferentes se cumple que:

$$R_{kk}(m) = \sum_{l=0}^{1022} c_l^k c_{l+m}^k = \begin{cases} 1023 & \text{si } m = 0 \\ < 65 & \text{si } |m| \geq 1 \end{cases} \quad y \quad r_{ik}(m) = \sum_{l=0}^{1022} c_l^i c_{l+m}^k < 65 \quad \forall m \quad (4-2)$$

Gráficamente podemos observar estas propiedades en la figura Figura 4-1.

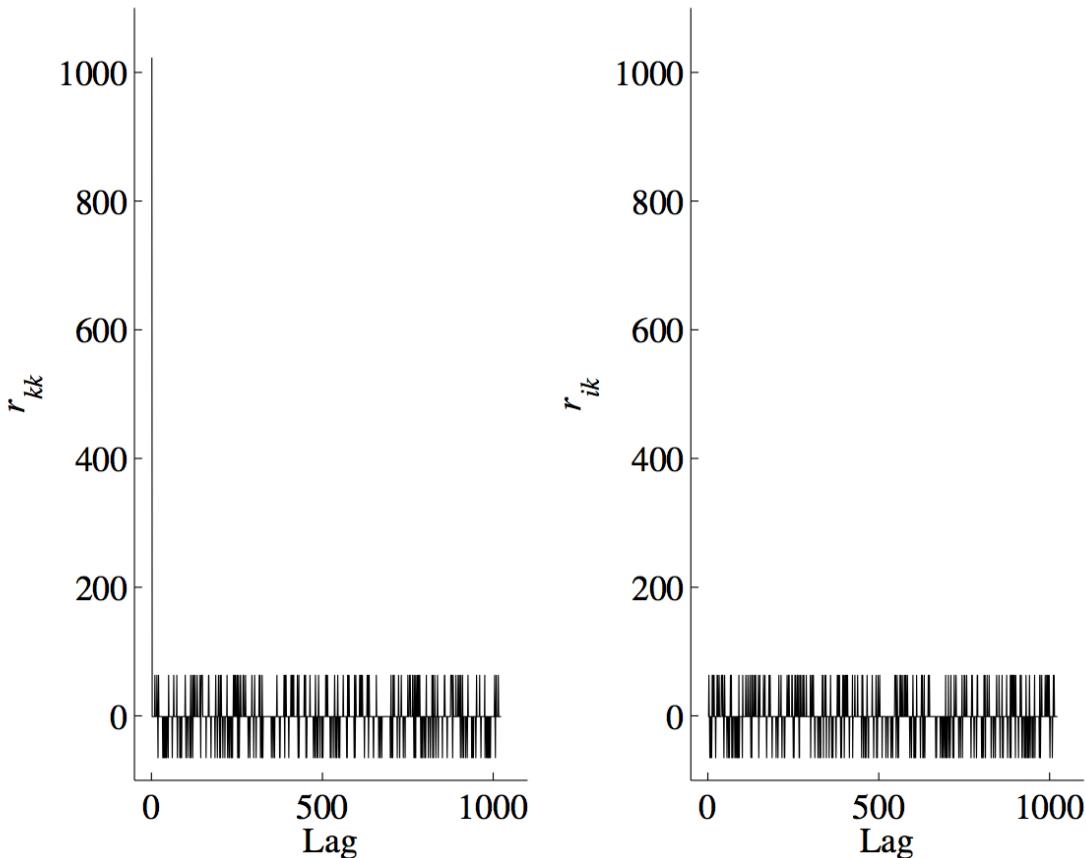


Figura 4-1 : Propiedades de las correlaciones de los códigos PRN. Izquierda: autocorrelación del código PRN 1. Derecha: correlación cruzadas de los códigos PRN 1 y 2. Figura extraída de [4-1].

4.2.3 Planteamiento general del algoritmo de adquisición

En base al modelo de señal explicado en el apartado anterior podemos establecer los fundamentos para cualquier algoritmo de adquisición: para la correcta eliminación del código PRN se hace necesaria la multiplicación de la señal de entrada por una réplica generada localmente del código del satélite perfectamente alineada con el código de la señal de entrada para obtener un valor aproximado al máximo de la correlación de la Figura 4-1, mientras que para eliminar la frecuencia portadora es necesario desmodular la señal multiplicando por un fasor de la misma frecuencia f_{d_i} y signo inverso.

De esta forma se puede ver el proceso de adquisición como una búsqueda sobre un espacio o cuadrícula (*grid*) bidimensional cuyos ejes estarían formados por las variables a estimar (el retardo de código τ_i y el desplazamiento frecuencial f_{d_i}).

Dado que ambos valores son magnitudes continuas debemos establecer un error máximo en la estimación inicial que se realiza en el bloque de adquisición y que delimitará la resolución del espacio de búsqueda. A los incrementos frecuenciales y de código se les denomina bins, mientras que la combinación de un bin de código y un bin frecuencial se le conoce como celda (*cell*) del espacio de búsqueda (*grid search*). Una representación gráfica de este concepto se puede ver en la Figura 4-2.

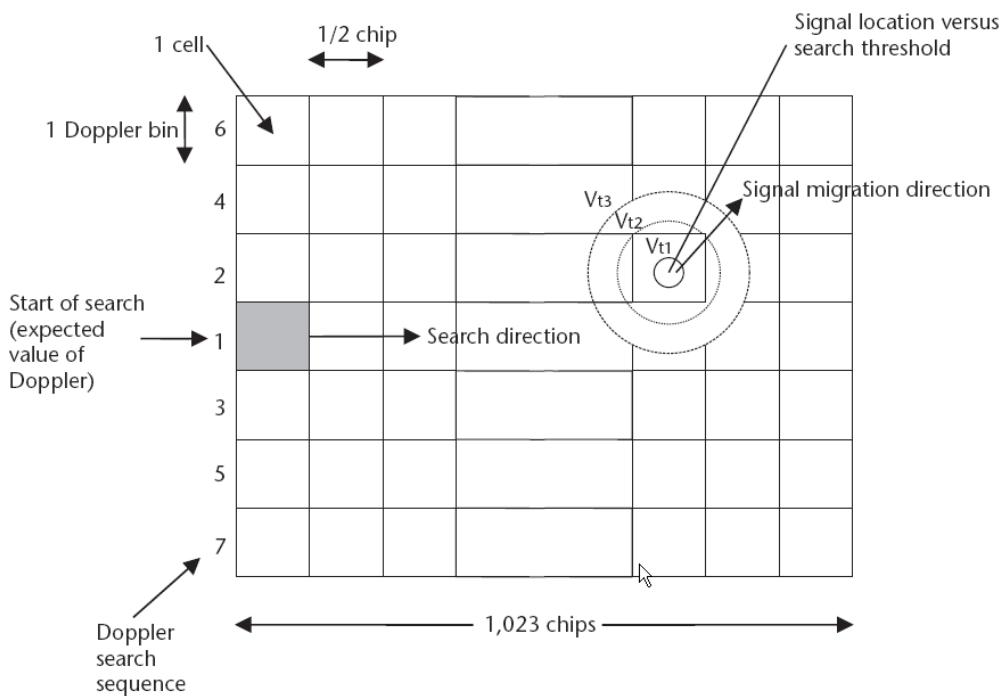


Figura 4-2 : Grid bidimensional de la búsqueda de un código CA. Figura extraída de [4-5].

Para realizar esta búsqueda, en cada celda se debe evaluar una función de ambigüedad, también llamada estadístico de prueba, y dependiendo de su valor numérico se tomará la decisión de si el satélite buscado se puede considerar presente y si los valores de esa celda se corresponden con los parámetros de sincronización a

estimar. El estadístico de prueba suele ser una función que depende de la correlación entre 2 señales, la de entrada y una generada localmente. De esta forma cuando los valores de una celda se aproximen a los valores de los parámetros de sincronización a estimar el valor del estadístico debería ser máximo en base a lo establecido en la ecuación (4-2).

En los apartados siguientes nos centraremos en varios aspectos clave de la búsqueda que se lleva a cabo en el algoritmo de la adquisición, como son:

- Dimensiones y resolución del espacio de búsqueda.
- Formas de tomar la decisión sobre la presencia o ausencia de un satélite.
- Formas de recorrer el espacio de búsqueda.
- Diseño y caracterización del estadístico de prueba.
- Medidas de la calidad de la adquisición: probabilidades de detección, falsa alarma y detección perdida.
- Cálculo del umbral de decisión.

4.3 Dimensiones y resolución del espacio de búsqueda (*grid*)

Un aspecto crucial en cualquier algoritmo de adquisición es definir las dimensiones del *grid* (en qué márgenes frecuenciales y de código se va a realizar la búsqueda) así como el incremento que se produce de ambos parámetros al cambiar de bin.

Si se empieza analizando el eje de código del espacio de búsqueda cabe comentar que el margen de búsqueda (amplitud del eje) viene delimitado por la longitud de los códigos de ensanchamiento (1023 chips en el caso de las señales GPS L1 C/A). En cuanto al incremento que hay que utilizar entre dos bins de código consecutivos en [4-1] se explica que como la resolución (máximo error permitido) del código debe de ser como máximo de medio chip, se deben implementar unos bins de código de un chip, lo que daría lugar a un total de 1023 bins de código. En otros textos de referencia como [4-5] se plantea realizar una búsqueda de mayor resolución utilizando incrementos de medio chip por bin de código, dividiendo por tanto el *grid* en 2046 bins de código.

En cuanto al eje frecuencial, por lo que respecta a su amplitud, tanto en [4-2] como en [4-6] se establecen unos márgenes de búsqueda de $\pm 5\text{ kHz}$ para el caso de un observador estático y de $\pm 10\text{ kHz}$ en el caso de un vehículo de alta velocidad. Para la resolución frecuencial, en [4-6] se explica que arbitrariamente se suele utilizar una resolución de $\frac{1}{2T}$ donde T es el periodo de integración coherente para una celda (la longitud en segundos de la secuencia de señal analizada en cada adquisición conocida también como ‘dwell time’), lo que da lugar a unos bins frecuenciales de $\frac{1}{T}$.

Hz. Este hecho provoca que para períodos de integración mayores los bins frecuenciales sean más estrechos. Dicho periodo de integración debe ser como mínimo de 1 milisegundo en el caso de señales GPS (para abarcar un código completo), en este caso los bins frecuenciales serían de 1 kHz, mientras que en el caso de señales débiles se suele utilizar 10 milisegundos de señal, dando lugar a unos bins frecuenciales de 100 Hz. En otros textos como [4-2] y [4-5] se aboga por utilizar incrementos frecuenciales de $\frac{2}{3T}$, lo que da lugar a unos bins frecuenciales de 667 Hz en el caso de señales con una alta CN_0 , mientras que para señales débiles los bins serían de 67 Hz.

4.4 Decisión sobre la presencia o ausencia de un satélite.

Una vez delimitado el margen de búsqueda de los parámetros a estimar se procede a explicar diversas técnicas de adquisición de un solo dwell (*single-dwell acquisition*), es decir, aquellos algoritmos en que la decisión de si los parámetros de sincronización buscados están presentes o no en una determinada celda se realiza una sola vez (en contra de los algoritmos denominados de *multiple-dwell*, que repiten el análisis sobre una misma celda a intervalos de tiempo regulares y realizan algún tipo de promediado antes de tomar la decisión).

Como hemos explicado con anterioridad el proceso de adquisición consiste en una búsqueda sobre el *grid* con el objetivo de determinar si el satélite buscado está presente y cuál de las celdas del espacio de búsqueda se corresponde con los parámetros de sincronización a estimar. Con este objetivo se evalúa una función denominada estadístico de prueba, que se supone debe ser máxima en aquella celda cuya posición se aproxime a los valores de dichos parámetros. Para determinar si el valor que toma el estadístico de prueba es lo suficientemente alto como para considerar que el resultado de la adquisición es positivo se compara este valor con un umbral de referencia. Si el estadístico supera dicho umbral se considera el satélite presente y se pasan los denominados parámetros de sincronización al algoritmo de *tracking*.

Existen varias formas de realizar la búsqueda sobre el *grid* y de tomar una decisión en consecuencia. En [4-13] y [4-14] se explican 3 de ellas:

- Búsqueda del máximo: El estadístico de prueba se evalúa en todas las celdas del espacio de búsqueda, para cada posible valor del desplazamiento Doppler y del retardo de código. Entonces se toma la decisión en función del valor del máximo del estadístico de prueba de todo el espacio de búsqueda. Si el valor máximo de dicho estadístico es mayor que el umbral prestablecido, el satélite se considera presente y la estimación del desplazamiento Doppler y del retardo de código se corresponden con la posición del máximo del estadístico de prueba.
- Búsqueda en serie: Esta estrategia consiste en evaluar el estadístico de prueba consecutivamente celda a celda. De esta forma, en cada iteración se evalúa una celda, se obtiene un valor que se compara inmediatamente con el umbral y el proceso de adquisición se detiene en la primera celda que supere el umbral.

La estimación del desplazamiento Doppler y del retardo de código se corresponden con la posición de la celda cuyo valor ha superado el umbral. De este modo sólo se evalúan, en promedio, la mitad de las celdas del espacio de búsqueda. Esto reduce el coste computacional del algoritmo de adquisición, pero por el contrario, pueden producirse la aparición de falsos resultados positivos ya que si el valor de algunas de las celdas no evaluadas fuera superior al valor del estadístico de la primera celda que ha superado el umbral, seguramente esto implicaría que los parámetros de sincronización se aproximarían al valor de la posición de esas celdas.

- Búsqueda híbrida: Este método es una mezcla de los 2 anteriores. En él, se evalúa el estadístico de prueba fila a fila (o columna a columna), aprovechando, por ejemplo, el paralelismo que se puede realizar gracias a los algoritmos basados en la FFT que se explicarán en el próximo apartado, y la decisión se toma a partir del máximo de cada fila (o columna). El proceso de adquisición termina tan pronto como el máximo de la fila (o columna) actual supera el umbral. Los parámetros de sincronización se corresponden con la posición de ese máximo.

Los resultados mostrados en [4-13] y [4-14] demuestran que el primero de los métodos (Búsqueda del máximo) es el que mejores prestaciones tiene en cuanto a la verosimilitud de la decisión que se toma en la ejecución del algoritmo. En base a estas conclusiones se ha tomado la decisión de implementar este método en el primer bloque de adquisición de GNSS-SDR.

4.5 Formas de recorrer el espacio de búsqueda.

Como se ha comentado en el apartado anterior, uno de los métodos más efectivos para realizar la adquisición (en cuanto a la fiabilidad de los resultados se refiere) consiste en recorrer todo el espacio de búsqueda con el fin de encontrar el valor máximo del estadístico de prueba y compararlo con un umbral de referencia. Este hecho implica inicialmente evaluar dicho estadístico en todas las celdas del *grid*, con el coste computacional que esto acarrea. Una revisión histórica sobre las diversas técnicas basadas en el dominio temporal se puede encontrar en [4-7]. En [4-1], [4-2], [4-6] y [4-8] se describen técnicas modernas de adquisición rápida basadas en el dominio frecuencial. En este apartado se presentan algunas técnicas que permiten reducir el coste computacional aprovechando algunas propiedades de la transformada de Fourier y paralelizar la búsqueda en uno de los dos ejes del espacio de búsqueda, aportando una considerable reducción de los cálculos a realizar.

Los subapartados siguientes describen tres métodos estándar de adquisición con la finalidad de discernir un método eficaz para su implementación en un receptor software. Estos tres métodos son:

- *Serial Search Acquisition*
- *Parallel Frequency Space Search Acquisition*
- *Parallel Code Phase Search Acquisition*

Se hace hincapié especialmente en el último método mencionado ya que es el que se ha implementado hasta el momento en el software GNSS-SDR.

4.5.1 Serial Search Acquisition

El método *Serial Search Acquisition* (adquisición mediante búsqueda en serie) es un método de adquisición basado en el dominio temporal frecuentemente utilizado en la adquisición de los sistemas GNSS, como GPS, implementados por hardware. Como se observa en el diagrama de bloques de la Figura 4-3, el algoritmo se basa en la multiplicación de la señal de entrada (4-1) por el código PRN de un satélite generado localmente con un cierto retardo de código $\hat{\tau}_i$, que se va variando con el fin de recorrer todos los bins de código del espacio de búsqueda. La señal se multiplica posteriormente por un fasor a frecuencia \hat{f}_{d_i} , esta frecuencia se corresponde inicialmente con el valor de la frecuencia intermedia IF (que es el valor del bin central del eje frecuencial) si ésta no ha sido eliminada con anterioridad, o el valor nulo en el caso de que haya sido eliminada, y se va actualizando cada vez que se han recorrido todos los bins de código. La forma de actualizar los valores frecuenciales se efectúa de forma simétrica (incrementando y decrementando a partir del valor inicial) hasta recorrer todo el *grid*. Para cada celda, se suman los valores de la señal a lo largo de un periodo de integración, dando lugar a la siguiente señal:

$$\begin{aligned}
 r_{xs_i} &= \sum_{m=0}^{L_c-1} x(t_m) s_i(t_m - \hat{\tau}_i) e^{-j2\pi\hat{f}_{d_i}t_m} = \\
 &= \sum_{m=0}^{L_c-1} \sum_{k=1}^M a_k \sum_{l=-\infty}^{\infty} D_k(l) s_k(t_m - \tau_k - lT_b) s_i(t_m - \hat{\tau}_i) e^{j2\pi(f_{dk} - \hat{f}_{d_i})t_m} + \\
 &\quad + \sum_{m=0}^{L_c-1} n(t_m) s_i(t_m - \hat{\tau}_i) e^{-j2\pi\hat{f}_{d_i}t_m}
 \end{aligned} \tag{4-3}$$

La salida es un valor escalado de la correlación entre la señal entrante y la señal generada localmente. Analizando el término de señal de la ecuación (4-3) se puede concluir que cuando el identificador de satélite de la señal generada localmente coincide con alguno de los satélites presentes en la señal de entrada (subíndices ‘i’ y ‘k’ de dicha expresión) y los valores de los parámetros de sincronización ($\hat{\tau}_i$ y \hat{f}_{d_i}) sean iguales a los reales (τ_i y f_{d_i}) la correlación r_{xs_i} tomará su valor máximo. Como se puede ver en la Figura 4-3 el cuarto bloque del diagrama obtiene el módulo al cuadrado del valor de dicha correlación. Esto es debido a que el valor de la correlación calculada puede tomar valores complejos, y es preferible obtener un valor real como estadístico de prueba para poder compararlo con el umbral de decisión.

Con el fin de encontrar el valor del máximo del estadístico de prueba (en este caso el módulo al cuadrado de la correlación) de todo el espacio de búsqueda, se suele guardar inicialmente el valor correspondiente al de la primera celda analizada, así como sus parámetros de sincronización asociados ($\hat{\tau}_i$ y \hat{f}_{d_i}) y se procede a incrementar dichos valores con el fin de analizar la celda siguiente. Si el valor del estadístico de la nueva celda analizada excede el del valor guardado, se guarda el nuevo valor y si no se pasa a la celda siguiente. De esta forma, después de recorrer todo el espacio de búsqueda se habrá encontrado el valor del máximo que es el que se pasa al último bloque: el comparador. Este bloque, compara el valor del módulo al cuadrado de la correlación con un umbral predefinido, si excede el umbral, los

parámetros de frecuencia y fase de código se consideran correctos, el satélite se considera presente y los parámetros se pueden transmitir al algoritmo de *tracking*. En caso de que el valor no exceda el umbral se considera el satélite no presente y se vuelve a iniciar el proceso con el PRN del satélite siguiente en la lista de satélites a adquirir.

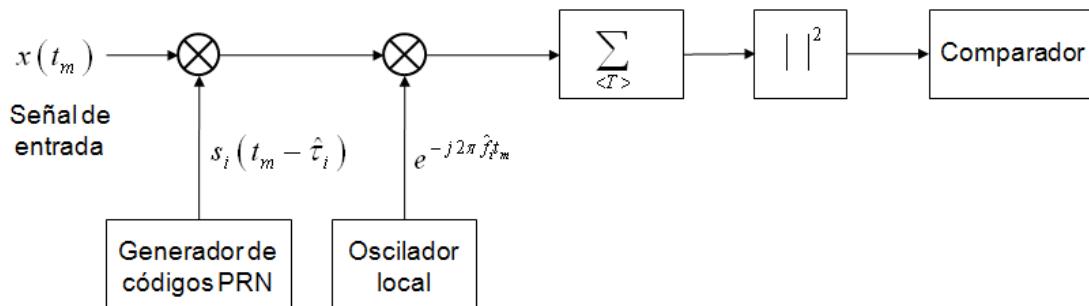


Figura 4-3: Diagrama de bloques de la *Serial Search Acquisition*.

El gran problema del que adolece este método es el gran número de operaciones que requiere. Por ejemplo, en el caso de realizar períodos de integración de un milisegundo y utilizar bins frecuenciales de 667 Hz para cubrir un rango de $IF \pm 10$ kHz e incrementos de código de medio chip se obtienen:

$$\underbrace{2 \cdot 1023}_{\text{bins de código}} \cdot \underbrace{\left(\frac{2 \cdot 10000}{667} + 1 \right)}_{\text{bins frecuenciales}} = 2046 \cdot 31 = 63426 \text{ combinaciones} \quad (4-4)$$

Además, en cada una de las 63426 combinaciones se debe realizar la correlación formulada en (4-3), con el coste computacional que esto acarrea (se considera que el coste de la convolución es de $O(N^2)$, donde $N = T_{\text{int}} \cdot f_s$, siendo T_{int} el periodo de integración utilizado, el mínimo es de 1 ms, y f_s la frecuencia de muestreo).

Los dos métodos siguientes intentan parallelizar una de las 2 dimensiones de búsqueda con el fin de reducir el número de combinaciones.

4.5.2 Parallel Frequency Space Search Acquisition

El método *Serial Search Acquisition* explicado en el apartado anterior consume muchos recursos debido a que realiza una búsqueda secuencial de los parámetros en las dos dimensiones del *grid* (tanto en la dimensión del código como en la de la frecuencia). El método *Parallel Frequency Space Search Acquisition* (adquisición mediante la paralelización frecuencial del espacio de búsqueda), paralleliza la búsqueda para uno de los dos parámetros. Este método se basa en la transformada de Fourier para realizar esta parallelización mediante una transformación del dominio temporal al dominio frecuencial. Para ello aprovecha las propiedades de los códigos de ensanchamiento de la señal DS-CDMA ya que si se multiplica la señal CDMA (cuyo espectro ensanchado se encuentra por debajo de nivel de ruido) por un código bien alineado, teóricamente la multiplicación de todos los chips daría uno, con lo que el

código de ensanchamiento sería eliminado, y por tanto el espectro se amplificaría y se observaría un pico en el índice de la frecuencia donde está centrada la señal, tal y como puede observarse en la Figura 4-4. Además, si la señal entrante contiene componentes de señal de otros satélites, estas componentes se reducirán al mínimo como resultado de las propiedades de correlación cruzada de las secuencias PRN.

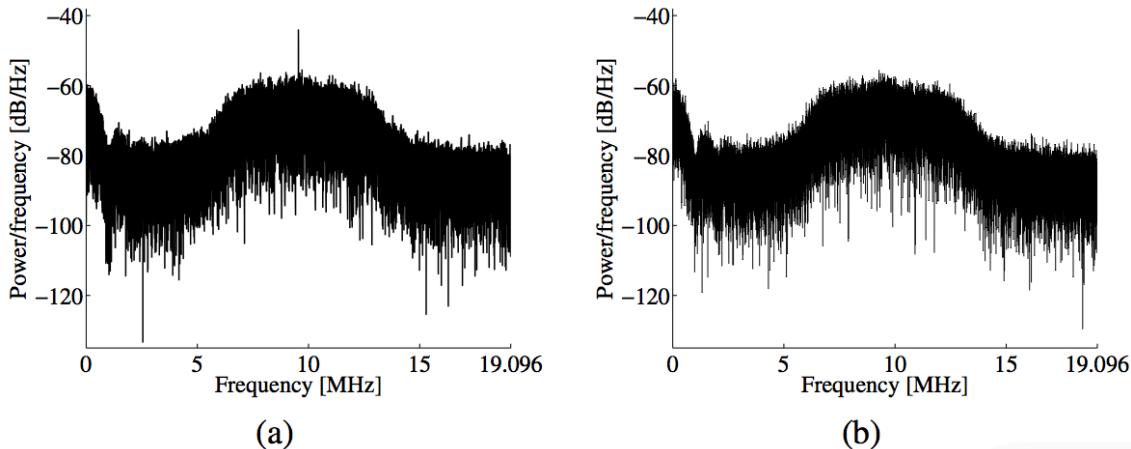


Figura 4-4: Densidad espectral de una señal GPS a una IF de 9.548 MHz multiplicada por una secuencia PRN generada localmente. (a) Cuando se multiplica por un código PRN perfectamente alineado, la salida presenta un pico en la frecuencia portadora. (b) Cuando se multiplica por un código no alineado, la salida no presenta picos. Figura extraída de [4-1].

El diagrama de bloques de una posible implementación de este algoritmo para GNSS-SDR se representa en la Figura 4-5. La primera parte de este esquema es idéntica a la primera parte del método *Serial Search Acquisition* explicado en el apartado anterior ya que el método consiste en multiplicar la señal de entrada (4-1) por el código PRN de un satélite generado localmente con un cierto retardo de código $\hat{\tau}_i$ que se va variando con el fin de recorrer todos los bins de código del espacio de búsqueda. La señal resultante se transforma en el dominio de la frecuencia mediante la transformada de Fourier. Dicha transformada de Fourier puede ser implementada como una transformada discreta de Fourier (DFT: *Discrete Fourier Transform*) o una transformada rápida de Fourier (FFT: *Fast Fourier Transform*). La FFT es la más rápida de las dos implementaciones, pero requiere una secuencia de entrada con una longitud que sea potencia de 2, es decir, 2^n donde n toma un valor entero positivo para que esto sea cierto.

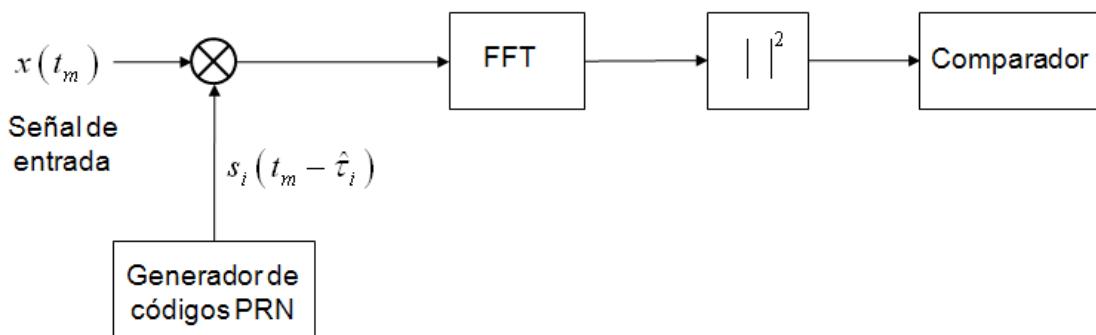


Figura 4-5: Diagrama de bloques del algoritmo *Parallel Frequency Space Search Acquisition*.

Si el código generado localmente está bien alineado con el código de un satélite de la señal entrante, la salida de la FFT tendrá un pico espectral en la frecuencia donde la señal esté centrada, que se corresponde con el desplazamiento frecuencial que se está buscando.

Como la señal resultante de realizar la FFT es compleja, se obtiene su módulo al cuadrado que es lo que se utiliza como estadístico de prueba y se busca su máximo. Dicho máximo se guarda (así como los parámetros de sincronización asociados) y se procede a repetir el procedimiento para el siguiente bin de código, reemplazando el valor del máximo sólo si el nuevo valor supera al guardado. De esta forma, después de repetir el procedimiento para todos los bins de código se habrá obtenido el máximo del estadístico de prueba para todo el espacio de búsqueda, y se procede a utilizar su valor en el bloque comparador.

Dicho bloque comparador compara la amplitud de este máximo con un umbral preestablecido y si el valor del máximo supera el valor del umbral se declara el satélite presente, y se pasan los parámetros de sincronización al bloque de *tracking*, el retardo de código \hat{t}_i que es el que se ha utilizado en el primer bloque de la Figura 4-5, mientras que la frecuencia Doppler \hat{f}_{d_i} se corresponde con la posición del máximo encontrado. Por otro lado, si el máximo de la señal no supera el umbral se considera el satélite no presente y se pasa a buscar el siguiente satélite en la lista.

En cuanto al número de veces que hay que aplicar el algoritmo para cubrir todo el *grid* para un determinado satélite, este número se ve drásticamente reducido respecto al del método *Serial Search Acquisition* explicado en el apartado anterior, ya que en lugar de recorrer todas las celdas del *grid* tan sólo se recorre uno de los ejes (concretamente el eje de código). De esta forma, si se utilizan períodos de integración de un milisegundo e incrementos de código de medio chip para la señal GPS L1 C/A (al igual que en el ejemplo del apartado anterior), se obtienen:

$$\underbrace{2 \cdot 1023}_{\text{bins de código}} = 2046 \text{ combinaciones} \quad (4-5)$$

frente a las 63426 calculadas para el método *Serial Search Acquisition* en (4-4). Por contra hay que tener en cuenta el coste de la transformación frecuencial que debe realizarse para cada bin de código. Dependiendo de la implementación de la transformada, debería ser posible hacer una aplicación más rápida con este método que con el método *Serial Search Acquisition*.

Por último cabe comentar que con este método si bien la resolución en el eje de código puede escogerse arbitrariamente, la resolución frecuencial viene determinada por la longitud de la DFT (y por tanto por el periodo de integración). Si definimos el tamaño de un bin frecuencial como el cociente entre la frecuencia de muestreo (f_s) y el número de muestras de la DFT (N), como dicho número de muestras (en cada iteración del algoritmo) se corresponde al producto entre el periodo de integración (T) y la frecuencia de muestreo, se obtiene que el tamaño de un bin frecuencial es el inverso de dicho periodo de integración:

$$\Delta f = \frac{f_s}{N} = \frac{f_s}{T \cdot f_s} = \frac{1}{T} \quad (4-6)$$

Así, para períodos de integración de 1 ms se obtienen unos bins frecuenciales de 1 kHz, que dan lugar a una resolución máxima de 500 Hz. Para obtener una mejor resolución, es decir una resolución menor, debería utilizarse un periodo de integración mayor con el fin de incrementar el número de muestras de la DFT de la ecuación (4-6).

4.5.3 Parallel Code Phase Search Acquisition

Como puede verse en la ecuación (4-4), la cantidad de pasos de búsqueda en la dimensión del código es significativamente mayor que en la dimensión frecuencial (2046 frente a aproximadamente 31). El método de paralelización explicado en el apartado anterior (*Parallel Frequency Space Search Acquisition*) elimina la necesidad de buscar a través de las 31 posibles frecuencias realizando una paralelización de la búsqueda en esta dimensión. Si la adquisición pudiera realizarse a partir de una paralelización de la dimensión del código en lugar de la dimensión frecuencial tan sólo se deberían realizar 31 iteraciones para cubrir todo el *grid* en lugar de 2046.

Existe un método de adquisición utilizado en receptores software GPS que utiliza esta paralelización de la búsqueda del retardo del código. Este método es conocido como *Parallel Code Phase Search Acquisition* y utiliza la correlación circular basada en la DFT que se explica a continuación.

4.5.3.1 Correlación circular y DFT (o FFT)

Como se ha visto en los apartados anteriores una de las funciones de la adquisición es realizar la correlación entre la señal entrante y un código PRN generado localmente. En lugar de multiplicar la señal de entrada por un código PRN con 2046 desplazamientos de código diferentes (en el caso de querer trabajar con una resolución de medio chip) como se realiza en el método *Serial Search Acquisition*, es más conveniente realizar una correlación circular entre la entrada y el código PRN generado localmente y aprovechar las propiedades de la DFT (y de su implementación mediante la FFT) para realizar dicha correlación.

Tal y como se explica en [4-1] y [4-6], dadas 2 señales discretas $x[n]$ e $y[n]$ de N muestras cada una, se define la correlación circular entre ambas como:

$$r_{xy}[n] = z[n] = \sum_{l=0}^{N-1} x[(n+l)]_N \cdot y^*[l] \quad (4-7)$$

donde $[\bullet]_N$ es el operador módulo N . Cabe remarcar que el resultado de dicha operación es una secuencia de N muestras (mientras que en una correlación lineal se obtendrían $2N - 1$ muestras). La realización de la correlación requiere de un número elevado de operaciones ya que para cada valor del índice n se deben realizar N multiplicaciones y sumar sus resultados. Dicho número de operaciones puede verse disminuido si utilizamos la DFT y se aprovechan sus propiedades.

De esta forma, si se define la DFT para una secuencia $x[n]$ como:

$$X[k] = DFT(x[n]) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j \frac{2\pi k}{N} n} \quad (4-8)$$

y, análogamente, la DFT inversa de la misma secuencia:

$$x[n] = DFT^{-1}(X[k]) = \sum_{k=0}^{N-1} X[k] \cdot e^{j \frac{2\pi k}{N} n} \quad (4-9)$$

Si se realiza la DFT de la correlación circular definida en (4-7), se obtiene:

$$\begin{aligned} Z[k] &= DFT(z[n]) = \sum_{n=0}^{N-1} z[n] \cdot e^{-j \frac{2\pi k}{N} n} = \sum_{n=0}^{N-1} \sum_{l=0}^{N-1} x[(n+l)]_N \cdot y^*[l] \cdot e^{-j \frac{2\pi k}{N} n} = \\ &= \sum_{l=0}^{N-1} \left(\sum_{n=0}^{N-1} x[(n+l)]_N \cdot e^{-j \frac{2\pi k}{N} (n+l)} \right) \cdot y^*[l] \cdot e^{j \frac{2\pi k}{N} l} = \sum_{l=0}^{N-1} X[k] \cdot y^*[l] \cdot e^{j \frac{2\pi k}{N} l} = \\ &= X[k] \cdot \left(\sum_{l=0}^{N-1} y[l] \cdot e^{-j \frac{2\pi k}{N} l} \right)^* = X[k] \cdot Y^*[k] \end{aligned} \quad (4-10)$$

Del resultado final de (4-10) se observa que la DFT de la correlación circular equivale a la multiplicación de la DFT de las 2 secuencias originales, aplicando el complejo conjugado a una de ellas. De esta forma, una posible forma de realizar la correlación circular entre 2 secuencias consiste en realizar la DFT de cada secuencia, conjugar una de ellas, multiplicar ambas transformadas y realizar la DFT inversa de la secuencia resultante de realizar el producto, es decir:

$$r_{xy}[n] = z[n] = DFT^{-1}(Z[k]) = DFT^{-1}(X[k] \cdot Y^*[k]) \quad (4-11)$$

Si para la realización de DFT, así como de la DFT inversa, se utiliza la FFT, el cálculo resulta en un número mucho menor de operaciones que el requerido en la forma directa de la ecuación (4-7), $O(N \log N)$ frente a $O(N^2)$ respectivamente.

4.5.3.2 Algoritmo Parallel Code Phase Search Acquisition

Basándose en lo expuesto en el apartado anterior, una posible implementación del algoritmo *Parallel Code Phase Search Acquisition* se muestra en la Figura 4-6. En ella se muestra como en primer lugar se multiplica la señal de entrada $x_{IN}(t_m)$ por un fasor a frecuencia \hat{f}_{d_i} , que se corresponde con uno de los valores centrales de los bins frecuenciales del espacio de búsqueda. A la señal resultante $x(t_m)$ se le realiza la FFT, y el resultado de dicha transformación se multiplica por la FFT compleja conjugada de un código PRN generado localmente. Al producto de ambas transformadas se le aplica la FFT inversa obteniendo, tal y como se ha explicado en el apartado anterior (concretamente en la ecuación (4-11)), la correlación circular entre la señal $x(t_m)$ y el código PRN $s_i(t_m)$. Es de esperar que, si el satélite i-ésimo (del cual

se ha generado el código PRN localmente) está presente en la señal de entrada y el valor de la frecuencia \hat{f}_{d_i} del oscilador local se acerca al de la frecuencia Doppler, la correlación circular presentará un máximo muy acentuado. Como el valor de la correlación puede ser complejo, se calcula su módulo al cuadrado que es el que se utiliza como estadístico de prueba.

De nuevo, al igual que en el método anterior, se procede a buscar, comparar y almacenar el valor del máximo del estadístico para cada iteración, que esta vez consiste en ir variando el desplazamiento frecuencial, así como guardar el valor de los parámetros de sincronización asociados. De esta forma, cuando se han recorrido todos los bins frecuenciales se dispone del máximo del estadístico de prueba para todo el espacio de búsqueda.

Dicho máximo es el que se entrega al bloque comparador que, al igual que el que en los otros dos algoritmos explicados en los apartados anteriores, compara el valor del máximo con un umbral preestablecido y si el valor de la correlación supera al del umbral se considera el satélite presente y se pasan los parámetros de sincronización al módulo de *tracking*, mientras que si no supera el umbral del comparador se declara el satélite no presente y se procede a iniciar de nuevo el algoritmo con el siguiente satélite de la lista de satélites del receptor.

Los parámetros de sincronización son en este caso la \hat{f}_{d_i} utilizada en el primer bloque de la Figura 4-6, y el índice de la correlación para el que se ha obtenido el valor máximo, ya que se corresponderá con el desplazamiento del código $\hat{\tau}_i$ en muestras.

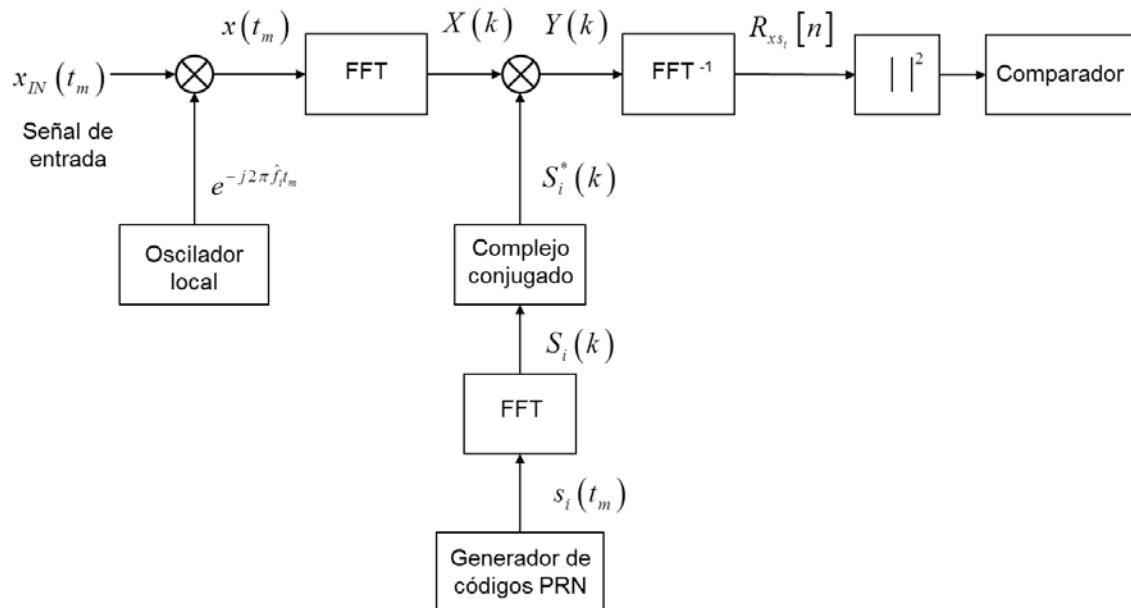


Figura 4-6: Diagrama de bloques de la Parallel Code Phase Search Acquisition.

Con este algoritmo, para recorrer todo el *grid* tan sólo es necesario recorrer todos los bins frecuenciales, por lo que si se utilizan bins frecuenciales de 667 Hz para cubrir el rango de $IF \pm 10$ kHz, al igual que en el algoritmo *Serial Search Acquisition* explicado en el apartado 4.5.1 se obtienen:

$$\frac{2 \cdot 10000}{\underbrace{667}_{\text{bins frecuenciales}}} + 1 \approx 31 \text{ combinaciones} \quad (4-12)$$

Este número es muy inferior a las 63426 calculadas para el método *Serial Search Acquisition* en (4-4) y también es considerablemente inferior a las 2046 calculadas para el método *Parallel Frequency Search Acquisition* en (4-5).

Además, cabe comentar que la transformada de Fourier del código PRN generado localmente sólo debe realizarse una vez para cada adquisición de un satélite en concreto, mientras que para cada una de las 31 frecuencias se debe llevar a cabo una transformada de Fourier y una transformada inversa de Fourier, por lo que la eficiencia computacional del método depende de la implementación de estas funciones que, como se ha comentado, se lleva a cabo mediante la FFT.

Por último comentar que en cuanto a la precisión de los parámetros estimados por este método la resolución frecuencial es similar a la del método *Serial Search Acquisition*, ya que se deja a la elección del diseñador la elección de la misma (recordemos que se recomiendan unos bins frecuenciales de 667 Hz en el caso de un periodo de integración de un milisegundo). En cuanto a la resolución del desplazamiento del código PRN, ésta depende directamente de la frecuencia de muestreo, ya que se obtiene un valor de la correlación para cada instante de muestreo. Es decir, si se utiliza una frecuencia de muestreo de 4 Msps (4 millones de muestras por segundo), un código PRN equivaldría a 4000 muestras, por lo que el eje del retardo de código se dividiría en 4000 bins diferentes en lugar de 2046.

En resumen, los valores de sincronismo estimados por la adquisición tienen un margen de error de $\pm \frac{1}{2}$ bin (en el caso de la frecuencia) o de $\pm \frac{1}{2} T_s$ (en el caso del tiempo).

4.6 Diseño del estadístico de prueba

En los tres algoritmos de adquisición explicados en los apartados 4.5.1, 4.5.2 y 4.5.3, el último bloque de cada implementación es un comparador que recibe como señal de entrada un valor que se corresponde con una versión modificada (normalmente el módulo al cuadrado) de la correlación cruzada entre la señal proporcionada por el bloque *Signal Conditioner* y una señal de referencia generada localmente. El valor del máximo de esta señal se compara con un umbral predeterminado con el fin de validar la presencia de un determinado satélite.

En este apartado se van a presentar los cálculos matemáticos para la obtención del umbral del comparador. El trabajo presentado en este apartado es una revisión y extensión de [4-9].

Para la obtención del umbral de decisión se hace necesario reescribir el modelo de señal de (4-1), con el fin de aplicar notación matricial que es la que se utiliza para realizar los cálculos teóricos.

Considerando que en la antena se reciben M señales correspondientes a otros tantos satélites GNSS, el modelo de la señal discreta banda base recibida es:

$$\underline{x} = \sum_{i=1}^M a_i \underline{d}_i(f_{d_i}, \tau_i) + \underline{n} \quad (4-13)$$

Donde:

- $\underline{x} = [x(t_0) \ x(t_1) \ \dots \ x(t_{N-1})]^T \in \mathbb{C}^{N \times 1}$ es el vector de señal recibida y N es el número de muestras utilizadas en una adquisición. El tiempo de adquisición coherente se puede definir como $T_{acq} = NT_s$ donde $f_s = 1/T_s$ es la frecuencia de muestreo.
- a_i es la amplitud compleja del satélite i -ésimo, y se asume constante durante el tiempo de adquisición. De esta forma, la potencia del satélite i -ésimo se puede definir como $P_i = |a_i|^2$, donde $|\bullet|$ es el operador módulo.
- $\underline{d}_i = [s_i(t_0 - \tau_i)e^{j2\pi f_{d_i} t_0} \ s_i(t_1 - \tau_i)e^{j2\pi f_{d_i} t_1} \ \dots \ s_i(t_{N-1} - \tau_i)e^{j2\pi f_{d_i} t_{N-1}}]^T \in \mathbb{C}^{N \times 1}$ es la señal GNSS banda base compleja DS-CDMA del satélite i -ésimo recibido. En este apartado consideramos la modulación BPSK para la señal DS-CDMA en que $s_i(t) = \sum_{k=-\infty}^{\infty} c_k p_k(t - kT_c)$, donde c_k es el ‘spreading code’, $p_k(t)$ es un pulso rectangular de duración T_c , y T_c es el periodo de chip utilizado en la señal GPS L1 C/A. También se pueden utilizar otras estructuras de señal, como la modulación CBOC (*Composite Binary Offset Carrier*) utilizada por la señal Galileo E1 [4-10]. Hay que tener en cuenta que la diferencia entre satélites dentro del mismo GNSS reside tan sólo en la secuencia de chips del ‘spreading code’ c_k .
- f_{d_i} y τ_i son el desplazamiento Doppler y el retardo de código para el satélite i -ésimo respectivamente, denominados parámetros de sincronización.
- $\underline{n} = [n(t_0) \ n(t_1) \ \dots \ n(t_{N-1})]^T \in \mathbb{C}^{N \times 1}$ es el vector de muestras de ruido. Dicho ruido es un proceso complejo, estacionario, circularmente simétrico y AWGN (*Additive White Gaussian Noise*) de media nula y varianza σ^2 , el cual modela al ruido térmico.

En pro de la simplicidad del modelo no se incluyen el efecto multicamino ni señales interferentes.

En una primera aproximación se analizará el caso en que tan sólo llegue a la antena un único satélite. En este caso, el proceso de detección se puede modelar como un contraste de hipótesis binario dado que tenemos 2 posibles hipótesis, por un lado se denominará H_0 o hipótesis nula cuando la señal del satélite buscado no esté presente, y H_1 o hipótesis alternativa cuando la señal deseada esté presente. El modelo de señal para ambas hipótesis se puede definir de la forma siguiente:

$$\underline{x} = \begin{cases} H_1 : a_{id} \underline{d}_{id} (f_{d_{id}}, \tau_{id}) + \underline{n} \\ H_0 : a_i \underline{d}_i (f_{d_i}, \tau_i) + \underline{n}_{i \neq id} \end{cases} \quad (4-14)$$

donde el satélite buscado se indica mediante el subíndice id , tal y como se observa en el modelo de la hipótesis alternativa. En la hipótesis nula, dado que la señal del satélite que llega a la antena no es la buscada (no coincide en alguno de los parámetros que se buscan: identificador de satélite, desplazamiento Doppler o retardo de código), se considera como una interferente, pero dada que su potencia es inferior a la del ruido se considera despreciable.

Un criterio comúnmente utilizado para abordar el problema de la detección es la maximización de la probabilidad de detección (P_d), definida como la probabilidad de detectar un determinado satélite cuando éste está presente ($P_d = P(H_1 / H_1)$) sujeto a una determinada probabilidad de falsa alarma (P_{fa}), definida como la probabilidad de detectar un determinado satélite cuando éste no está presente ($P_{fa} = P(H_1 / H_0)$). Una posible solución al problema se puede encontrar aplicando el teorema de Neyman-Pearson) [4-11].

TEOREMA DE NEYMAN-PEARSON: Para maximizar P_d fijada una cierta $P_{fa} = \alpha$, decidir H_1 si

$$L(\underline{x}) = \frac{p(\underline{x}; H_1)}{p(\underline{x}; H_0)} > \gamma \quad (4-15)$$

donde el umbral γ se puede encontrar a partir de

$$P_{fa} = \int_{\{\underline{x}: L(\underline{x}) > \gamma\}} p(\underline{x}; H_0) d\underline{x} = \alpha \quad (4-16)$$

En las expresiones (4-15) y (4-16) $p(\cdot)$ es la función de densidad de probabilidad conjunta (PDF- *Probability Density Function*) de las muestras del vector \underline{x} en las hipótesis H_1 y H_0 respectivamente. Dichas funciones dependen de ciertos parámetros ($a_{id}, \tau_{id}, f_{d_{id}}$ y σ^2) que son desconocidos por el receptor y por tanto deben ser estimados, dando lugar al denominado *Generalized Likelihood Ratio Test* (GLRT) [4-11]. La expresión del GLRT es:

$$L(\underline{x}) = \frac{p(\underline{x}; \hat{\sigma}_{H_1}^2, \hat{\theta}, H_1)}{p(\underline{x}; \hat{\sigma}_{H_0}^2, H_0)} > \gamma \quad (4-17)$$

donde $\hat{\underline{\theta}}$ es el estimador de máxima verosimilitud (MLE-Maximum Likelihood Estimator) de los parámetros de sincronización, en nuestro caso $\hat{\underline{\theta}} = \begin{bmatrix} \hat{a}_{id} & \hat{f}_{d_{id}} & \hat{\tau}_{id} \end{bmatrix}^T$, $\hat{\sigma}_{H_1}^2$ y $\hat{\sigma}_{H_0}^2$ son los estimadores ML de la varianza del ruido en cada hipótesis, y γ es el umbral de detección a diseñar.

En los siguientes sub-apartados se describen la derivación de los estimadores ML y del test de la adquisición respectivamente.

4.6.1 Estimadores de Máxima Verosimilitud

Dado que en ambas hipótesis el vector de señal \underline{x} es un vector Gaussiano complejo, $\underline{x} : \mathbb{C}^{N \times 1}(\underline{m}_x, \underline{\underline{C}}_x)$, su PDF se modela según:

$$p(\underline{x}) = \frac{1}{\pi^N |\underline{\underline{C}}_x|} e^{-[\underline{x} - \underline{m}_x]^H \underline{\underline{C}}_x^{-1} [\underline{x} - \underline{m}_x]} \quad (4-18)$$

En la expresión anterior $\underline{m}_x = E(\underline{x}) \in \mathbb{C}^{N \times 1}$ es el vector de esperanzas del vector de muestras de la señal, $\underline{\underline{C}}_x = E([\underline{x} - \underline{m}_x][\underline{x} - \underline{m}_x]^H) \in \mathbb{C}^{N \times N}$ es su matriz de covarianzas, y $(\cdot)^H$ indica el hermítico (transpuesto y conjugado).

En el caso de la hipótesis nula, el cálculo de la esperanza da:

$$\underline{m}_{x;H_0} = E(\underline{x}; H_0) = E(\underline{n}) = \underline{0} \quad (4-19)$$

y el de la matriz de covarianzas:

$$\underline{\underline{C}}_{x;H_0} = E([\underline{x} - \underline{m}_x][\underline{x} - \underline{m}_x]^H; H_0) = E(\underline{n}\underline{n}^H) = \underline{\underline{R}}_n = \sigma^2 \underline{\underline{I}} \quad (4-20)$$

donde $\underline{\underline{R}}_n$ es la matriz de correlación del vector de muestras del ruido e $\underline{\underline{I}}$ es la matriz identidad de dimensión $N \times N$.

Substituyendo (4-19) y (4-20) en la ecuación (4-18) se obtiene la PDF para la hipótesis nula:

$$p(\underline{x}; \sigma^2, H_0) = (\pi\sigma^2)^{-N} e^{-\frac{1}{\sigma^2} \underline{x}^H \underline{x}} \quad (4-21)$$

Aplicando el operador logaritmo natural ($L(\cdot)$) se obtiene la función de verosimilitud logarítmica:

$$\Lambda(\sigma^2; H_0) = L(p(\underline{x}; \sigma^2, H_0)) = -NL(\pi\sigma^2) - \frac{1}{\sigma^2} \underline{x}^H \underline{x} \quad (4-22)$$

Maximizando la ecuación anterior respecto del parámetro a estimar σ^2 se obtiene el MLE de la potencia del ruido en la hipótesis nula:

$$\hat{\sigma}_{H_0}^2 = \frac{1}{N} \underline{x}^H \underline{x} \quad (4-23)$$

En el caso de la hipótesis alternativa, el cálculo del vector de esperanzas da:

$$\underline{m}_{x;H_1} = E(\underline{x}; H_1) = E(a_{id} \underline{d}_{id} (f_{d_{id}}, \tau_{id}) + \underline{n}) = a_{id} \underline{d}_{id} (f_{d_{id}}, \tau_{id}) \quad (4-24)$$

En el resto de este apartado, para facilitar la notación, se prescinde del identificador id , y de la dependencia del vector $\underline{d}(f_{d_{id}}, \tau_{id})$ con el desplazamiento Doppler $f_{d_{id}}$ y el retardo de código τ_{id} , con lo que la ecuación (4-24) se puede reescribir como:

$$\underline{m}_{x;H_1} = a_{id} \underline{d}_{id} (f_{d_{id}}, \tau_{id}) = a \underline{d} \quad (4-25)$$

Si se calcula la matriz de covarianzas para la hipótesis alternativa se obtiene:

$$\underline{\underline{C}}_{x;H_1} = E([\underline{x} - \underline{m}_x][\underline{x} - \underline{m}_x]^H; H_1) = E([\underline{x} - a \underline{d}][\underline{x} - a \underline{d}]^H) = E(\underline{n} \underline{n}^H) = \underline{\underline{R}}_n = \sigma^2 \underline{\underline{I}} \quad (4-26)$$

Al substituir las expresiones (4-25) y (4-26) en la ecuación (4-18) se obtiene la PDF de la hipótesis alternativa:

$$p(\underline{x}; \underline{\theta}, \sigma^2, H_1) = (\pi\sigma^2)^{-N} e^{-\frac{1}{\sigma^2} [\underline{x} - a \underline{d}]^H [\underline{x} - a \underline{d}]} = (\pi\sigma^2)^{-N} e^{-\frac{NR}{\sigma^2}} \quad (4-27)$$

Donde el escalar R se define como:

$$R = \frac{1}{N} [\underline{x} - a \underline{d}]^H [\underline{x} - a \underline{d}] = \frac{1}{N} \underline{x}^H \underline{x} + |a|^2 \frac{1}{N} \underline{d}^H \underline{d} - a^* \frac{1}{N} \underline{d}^H \underline{x} - a \frac{1}{N} \underline{x}^H \underline{d} \quad (4-28)$$

En esta expresión aparecen ciertos términos que pueden interpretarse como estimadores de las correlaciones de las señales que forman el modelo de señal:

- $\hat{R}_{xx} = \frac{1}{N} \underline{x}^H \underline{x}$, es el estimador de la potencia de la señal de entrada.

- $\hat{R}_{xd} = \frac{1}{N} \underline{d}^H \underline{x}$, es el estimador de la correlación cruzada entre la señal de entrada y la señal GNSS de referencia.
- $\hat{R}_{dd} = \frac{1}{N} \underline{d}^H \underline{d}$, es el estimador de la potencia de la señal GNSS de referencia. Considerando que se trabaja con potencia normalizada, se puede aproximar por la unidad, $\hat{R}_{dd} \approx 1$. Reseñar también que \hat{R}_{dd} no depende de los parámetros de sincronización a estimar.

Aplicando las definiciones anteriores se obtiene finalmente que el escalar R puede escribirse como:

$$R = \hat{R}_{xx} + |a|^2 \hat{R}_{dd} - a^* \hat{R}_{xd} - a \hat{R}_{xd}^* \quad (4-29)$$

Aplicando el operador logaritmo natural ($L(\cdot)$) a la ecuación (4-27) se obtiene la nueva función de verosimilitud logarítmica:

$$\Lambda(\underline{\theta}, \sigma^2; H_1) = L(p(\underline{x}; \underline{\theta}, \sigma^2, H_1)) = -NL(\pi\sigma^2) - \frac{NR}{\sigma^2} \quad (4-30)$$

Maximizar la función de verosimilitud anterior equivale a minimizar la siguiente función de coste:

$$F(\underline{\theta}, \sigma^2; H_1) = L(\sigma^2) + \frac{R}{\sigma^2} \quad (4-31)$$

Minimizando la función anterior respecto a σ^2 (y considerando el resto de parámetros a estimar conocidos) se obtiene el MLE de la potencia del ruido en la hipótesis alternativa:

$$\hat{\sigma}_{H_1}^2 = R|_{a=\hat{a}, f=\hat{f}, \tau=\hat{\tau}} \quad (4-32)$$

Substituyendo el resultado anterior en la ecuación (4-31) se obtiene la nueva función de coste:

$$F(\underline{\theta}; H_1) = F(\underline{\theta}, \hat{\sigma}_{H_1}^2; H_1) = L(R) + 1 \quad (4-33)$$

Minimizar la nueva función de coste respecto al parámetro a equivale a minimizar el escalar $R = \hat{R}_{xx} + |a|^2 \hat{R}_{dd} - a^* \hat{R}_{xd} - a \hat{R}_{xd}^*$ respecto al mismo parámetro. Al realizar dicha minimización se obtiene el MLE de la amplitud de la señal del satélite:

$$\hat{a} = \left. \frac{\hat{R}_{xd}}{\hat{R}_{dd}} \right|_{\sigma^2 = \hat{\sigma}_{H_1}^2, f = \hat{f}, \tau = \hat{\tau}} \quad (4-34)$$

Al substituir el nuevo estimador en la función (4-33) y prescindir de las constantes superfluas se obtiene la nueva función a minimizar:

$$\begin{aligned} G(f, \tau; H_1) &= F(\hat{a}, f, \tau; H_1) - 1 = L(\hat{R}) = L\left(\hat{R}_{xx} + |\hat{a}|^2 \hat{R}_{dd} - \hat{a}^* \hat{R}_{xd} - \hat{a} \hat{R}_{xd}^*\right) = \\ &= L\left(\hat{R}_{xx} + \hat{a}^* \left(\cancel{\hat{a} \hat{R}_{dd}} - \cancel{\hat{R}_{xd}} \right) - \hat{a} \hat{R}_{xd}^*\right) = L\left(\hat{R}_{xx} - \hat{a} \hat{R}_{xd}^*\right) = L\left(\hat{R}_{xx} - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{dd}}\right) \end{aligned} \quad (4-35)$$

Dado que el término \hat{R}_{dd} es una estimación del término R_{dd} cuyo valor, en caso de trabajar con potencia normalizada, es a la unidad, minimizar la anterior función de coste equivale a maximizar el término $|\hat{R}_{xd}|^2$, que es el módulo al cuadrado de la correlación cruzada entre la señal recibida y la señal GNSS que se está buscando. Esta expresión es conocida como '*non-coherent Matched Filter (MF) estimator*' [4-12].

$$\left(\hat{f}_d, \hat{\tau}\right) = \min_{f_d, \tau} G(f_d, \tau; H_1) = \min_{f_d, \tau} L\left(\hat{R}_{xx} - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{dd}}\right) = \max_{f_d, \tau} |\hat{R}_{xd}|^2 \quad (4-36)$$

Dado que no es posible obtener una expresión cerrada para \hat{f} ni para $\hat{\tau}$, se puede realizar una búsqueda basada en un *grid* (*grid based search*) para encontrar el máximo de la función tal y como se explicó en los algoritmos de los apartados 4.5.1, 4.5.2 y 4.5.3.

4.6.2 Función Test de la Adquisición

Con los cálculos realizados en el apartado anterior ya se está en disposición de encontrar la función del GLRT $L(\underline{x})$ definida en (4-17). Substituyendo las PDFs (4-21) y (4-27) en la expresión (4-17) se obtiene:

$$L(\underline{x}) = \frac{p(\underline{x}; \hat{\sigma}_{H_1}^2, \hat{\theta}, H_1)}{p(\underline{x}; \hat{\sigma}_{H_0}^2, H_0)} = \frac{\left(\pi \hat{\sigma}_{H_1}^2\right)^{-N} e^{-\frac{N\hat{R}}{\hat{\sigma}_{H_1}^2}}}{\left(\pi \hat{\sigma}_{H_0}^2\right)^{-N} e^{-\frac{1}{\hat{\sigma}_{H_0}^2} \underline{x}^H \underline{x}}} = \left(\frac{\hat{\sigma}_{H_0}^2}{\hat{\sigma}_{H_1}^2}\right)^N e^{-\frac{N\hat{R}}{\hat{\sigma}_{H_1}^2} + \frac{N\hat{R}_{xx}}{\hat{\sigma}_{H_0}^2}} > \gamma \quad (4-37)$$

Substituyendo las expresiones de las estimaciones de la varianza del ruido encontradas en (4-23) y (4-32) en la expresión anterior se obtiene la función de test:

$$L(\underline{x}) = \left(\frac{\hat{R}_{xx}}{\hat{R}} \right)^N > \gamma \quad (4-38)$$

Tomando la raíz N -ésima de la expresión anterior se obtiene:

$$\sqrt[N]{L(\underline{x})} = \frac{\hat{R}_{xx}}{\hat{R}} = \frac{\hat{R}_{xx}}{\hat{R}_{xx} - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{dd}}} = \frac{1}{1 - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}\hat{R}_{dd}}} \approx \frac{1}{1 - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}}} > \sqrt[N]{\gamma} \quad (4-39)$$

En la ecuación anterior se ha cambiado el término \hat{R}_{dd} por su valor real $R_{dd} = 1$ dado que es conocido por definición y no hace falta estimarlo. Dado que ambos términos, tanto numerador como denominador son positivos, finalmente podemos aislar:

$$T(\underline{x}) = \frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}} > \gamma_{th} \quad \text{con} \quad \gamma_{th} = 1 - \gamma^{\frac{N}{2}} \quad (4-40)$$

De esta forma, se puede definir el estadístico de prueba para todo el *Grid Search* como:

$$T_{GS}(\underline{x}) = \max T_i(\underline{x}) = \max \left(\frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}} \right) \quad \forall i = 1, \dots, N_{cells} \quad (4-41)$$

Donde N_{cells} es el número total de celdas del espacio de búsqueda.

La función de test resultante es conocida como “generalized non-coherent MF detector” [4-11]. El detector resultante funciona de la siguiente forma:

$$\begin{cases} T_{GS}(\underline{x}) \geq \gamma_{th} \Rightarrow \text{satélite presente} & y \quad (\hat{f}_d, \hat{\tau}) = \arg \max_{(f_d, \tau)} T_i(\underline{x}) \\ T_{GS}(\underline{x}) < \gamma_{th} \Rightarrow \text{satélite no presente} & \end{cases} \quad (4-42)$$

4.7 Cálculo del umbral de decisión

El cálculo del umbral de decisión es un aspecto crítico en cualquier algoritmo de adquisición ya que si el valor de éste es demasiado bajo se producirán falsos resultados positivos con más frecuencia de lo deseable, mientras que si es demasiado elevado se producirá el hecho de que no se detectarán satélites que sí están presentes. Para ello debemos formular ciertas medidas probabilísticas que nos definan la calidad del algoritmo midiendo la fiabilidad de sus resultados. Algunas de ellas (como la probabilidad de detección y falsa alarma) ya han sido introducidas brevemente en los apartados anteriores, pero se hace necesario profundizar en su definición y posterior cálculo, incluyendo la caracterización del estadístico de prueba, con el fin de obtener una forma de calcular el valor del umbral de decisión que optimice las prestaciones del algoritmo de adquisición. En las siguientes secciones se tratan todas estas cuestiones.

4.7.1 Medidas de la fiabilidad de la adquisición: probabilidades de detección, falsa alarma y detección fallida.

Existen fundamentalmente tres medidas que caracterizan la fiabilidad de un algoritmo de adquisición:

- Probabilidad de detección (P_D), que es la probabilidad de detectar un satélite cuando éste está presente.
- Probabilidad de falsa alarma (P_{FA}), que es la probabilidad de un falso positivo en el resultado de una adquisición. Según un análisis exhaustivo desarrollado en [4-14] existen dos casos en los que esto puede suceder: cuando se declara presente un satélite que en realidad está ausente (P_{FA}^a), o cuando se detecta un satélite que sí está presente, pero los valores de los parámetros de sincronización estimados no son correctos (P_{FA}^p), es decir, aparece un máximo del estadístico de prueba que supera el umbral de decisión en una celda que no es la correcta.
- Probabilidad de detección fallida (*Missed Detection*) (P_{MD}), que se define como la probabilidad de no detectar un satélite cuando éste está presente, es decir, el máximo del estadístico de prueba no supera el umbral.

El objetivo de cualquier algoritmo de adquisición es que la probabilidad de detección sea lo más elevada, mientras que es deseable que las otras dos probabilidades (de falsa alarma y detección fallida) sean lo menores posibles.

Cabe comentar que aunque las probabilidades anteriores se calculan en base a la función de test definida en (4-42), que utiliza el estadístico de prueba definido en (4-41) y que se corresponde con el máximo de todo el espacio de búsqueda, es conveniente analizar primero estas probabilidades para el caso del análisis de una sola celda, debido a su simplicidad, y extender los resultados a la búsqueda en todo el espacio.

4.7.2 Cálculo de las probabilidades de detección, falsa alarma y detección fallida para una celda.

Centrado el análisis en una sola celda, el estadístico de prueba a utilizar es el formulado en (4-40) y se define la probabilidad de detección para una celda como la probabilidad de que el estadístico de prueba cuando el satélite buscado está presente ($T(\underline{x}; H_1)$) supere el umbral de decisión γ , es decir:

$$P_d(\gamma) = P(T(\underline{x}; H_1) > \gamma) = \int_{\gamma}^{\infty} p_{T;H_1}(t) dt = 1 - F_{T;H_1}(\gamma) \quad (4-43)$$

donde $p_{T;H_1}(t)$ y $F_{T;H_1}(t)$ son las funciones de densidad de probabilidad (PDF) y de distribución acumulada (CDF), respectivamente, del estadístico de prueba $T(\underline{x}; H_1)$.

De la misma forma, se define la probabilidad de falsa alarma como la probabilidad de que el estadístico de prueba en ausencia de satélite ($T(\underline{x}; H_0)$) supere el umbral:

$$P_{fa}(\gamma) = P(T(\underline{x}; H_0) > \gamma) = \int_{\gamma}^{\infty} p_{T;H_0}(t) dt = 1 - F_{T;H_0}(\gamma) \quad (4-44)$$

Cabe remarcar que en el caso de analizar una única celda no tiene sentido distinguir dos casos para la probabilidad de falsa alarma ya que el caso en que el satélite está presente hace referencia a un fenómeno que se produce al analizar todo el espacio de búsqueda y no sólo una celda.

Por último, se define la probabilidad de detección fallida como la probabilidad de que el estadístico de prueba en presencia de satélite ($T(\underline{x}; H_1)$) no supere el umbral de decisión, aunque se puede calcular también a partir de la probabilidad de detección:

$$P_{md}(\gamma) = P(T(\underline{x}; H_1) < \gamma) = \int_{-\infty}^{\gamma} p_{T;H_1}(t) dt = F_{T;H_1}(\gamma) = 1 - P_d(\gamma) \quad (4-45)$$

Notar el uso de subíndices en minúscula cuando se formulen probabilidades para una sola celda (por ejemplo P_d para la probabilidad de detección) mientras que se utilizan subíndices en mayúscula cuando se formulen las mismas probabilidades para todo el espacio de búsqueda (en este caso la probabilidad de detección se formula como P_D).

Antes de extender el análisis a todo el espacio de búsqueda se hace necesario caracterizar aleatoriamente el estadístico de prueba en las dos hipótesis definidas en (4-14), H_1 y H_0 , ya que como se acaba de explicar, sus distribuciones serán de utilidad para el cálculo de las probabilidades definidas en (4-43), (4-44) y (4-45).

4.7.3 Caracterización del estadístico de prueba para una celda.

Si se analiza el estadístico de prueba de la expresión (4-40), al ser éste el cociente entre $|\hat{R}_{xd}|^2$ y \hat{R}_{xx} es necesario analizar inicialmente la estadística de \hat{R}_{xd} en cada una de las 2 hipótesis.

Dado que (tal y como se vio en el apartado 4.6.1) en ambas hipótesis el vector de señal \underline{x} se modela como un vector Gaussiano complejo, $\underline{x} : \mathbb{C}N^{N \times 1}(\underline{m}_x, \underline{\underline{C}}_x)$, el estimador de la correlación cruzada \hat{R}_{xd} es una variable aleatoria Gaussiana compleja:

$$\hat{R}_{xd} = \frac{1}{N} \underline{d}^H \underline{x} : \mathbb{C}N\left(\mu_{\hat{R}_{xd}}, \sigma_{\hat{R}_{xd}}^2\right) \quad (4-46)$$

Si calculamos la esperanza de dicha variable, esta depende del valor de \underline{m}_x :

$$\mu_{\hat{R}_{xd}} = E\left(\frac{1}{K} \underline{d}^H \underline{x}\right) = \frac{1}{N} \underline{d}^H \underline{m}_x \quad (4-47)$$

Al aplicar los valores de \underline{m}_x calculados para cada hipótesis en (4-19) y (4-24) se obtiene la media de \hat{R}_{xd} para la hipótesis nula:

$$\mu_{\hat{R}_{xd};H_0} = \frac{1}{K} \underline{d}^H \underline{m}_{x;H_0} = \frac{1}{K} \underline{d}^H \underline{0} = 0 \quad (4-48)$$

y para la hipótesis alternativa:

$$\mu_{\hat{R}_{xd};H_1} = \frac{1}{K} \underline{d}^H \underline{m}_{x;H_1} = a \frac{1}{K} \underline{d}^H \underline{d} = a \hat{R}_{dd} \simeq a \quad (4-49)$$

Si se desarrolla el cálculo de la varianza del estimador:

$$\begin{aligned} \sigma_{\hat{R}_{xd}}^2 &= E\left(\left|\hat{R}_{xd} - \mu_{\hat{R}_{xd}}\right|^2\right) = E\left(\left|\frac{1}{N} \underline{d}^H \underline{x} - \frac{1}{N} \underline{d}^H \underline{m}_x\right|^2\right) = \frac{1}{N^2} E\left(\left|\underline{d}^H (\underline{x} - \underline{m}_x)\right|^2\right) = \\ &= \frac{1}{N^2} E\left(\underline{d}^H (\underline{x} - \underline{m}_x)(\underline{x} - \underline{m}_x)^H \underline{d}\right) = \frac{1}{N^2} \underline{d}^H \underline{\underline{C}}_x \underline{d} \end{aligned} \quad (4-50)$$

Dado que, tal y como se mostró en (4-20) y (4-26), la matriz de covarianzas del vector \underline{x} en ambas hipótesis da el mismo resultado ($\underline{\underline{C}}_x = \sigma^2 \underline{\underline{I}}$), al substituirlo en la ecuación anterior se obtiene:

$$\sigma_{\hat{R}_{xd;H_0}}^2 = \sigma_{\hat{R}_{xd;H_1}}^2 = \sigma^2 \frac{1}{N^2} \underline{d}^H \underline{d} = \sigma^2 \frac{1}{N} \hat{R}_{dd} \simeq \frac{\sigma^2}{N} \quad (4-51)$$

De esta forma se obtiene que el estimador de la correlación cruzada en cada una de las hipótesis se caracteriza como una variable aleatoria Gaussiana compleja con la misma varianza y que sólo difiere en su media:

$$\hat{R}_{xd;H_0} : \mathbb{C}N\left(0, \frac{\sigma^2}{N}\right) \quad y \quad \hat{R}_{xd;H_1} : \mathbb{C}N\left(a, \frac{\sigma^2}{N}\right) \quad (4-52)$$

Como la amplitud de la señal del satélite a y la potencia del ruido σ^2 son desconocidos por el receptor se hace necesario utilizar alguna estimación en función de los datos (las muestras de los vectores \underline{x} y \underline{d}). En el caso de la amplitud de la señal del satélite se utiliza el estimador MLE obtenido en (4-34):

$$a = \frac{\hat{R}_{xd}}{\hat{R}_{dd}} = \hat{R}_{xd} \quad (4-53)$$

Para la potencia del ruido se tienen las 2 estimaciones diferentes calculadas en (4-23) y (4-32), respectivamente, según la hipótesis en la que se esté trabajando. Dado que la potencia del ruido realmente es la misma en ambas, se plantea la duda de cuál utilizar. Asumiendo las aproximaciones necesarias se obtiene:

$$\hat{\sigma}_{H_1}^2 = \hat{R} = \hat{R}_{xx} - \frac{|\hat{R}_{xd}|^2}{\hat{R}_{dd}} \simeq \hat{R}_{xx} - |\hat{R}_{xd}|^2 \simeq \hat{R}_{xx} = \hat{\sigma}_{H_0}^2 \quad (4-54)$$

donde en la primera aproximación se ha cambiado el término \hat{R}_{dd} por su valor real $R_{dd} = 1$ dado que es conocido y no hace falta estimarlo y en la última aproximación se ha aplicado que $|\hat{R}_{xd}|^2 \ll \hat{R}_{xx}$ gracias a las propiedades de las correlaciones de los códigos de ensanchamiento que se reflejan en la Figura 4-1.

De esta forma, para ambas hipótesis la varianza se aproximará por:

$$\sigma_{\hat{R}_{xd}}^2 \simeq \frac{\hat{R}_{xx}}{N} \quad (4-55)$$

Una vez se ha caracterizado estadísticamente el estimador de la correlación cruzada \hat{R}_{xd} , se procede a caracterizar del estadístico $T(\underline{x})$ definido en (4-40). Para ello se asume, tal y como se explica en [4-9], que al realizar la estimación de la potencia de la señal de entrada con un número elevado número de muestras, esta se puede formular como $\hat{R}_{xx} = \lim_{N \rightarrow \infty} \frac{1}{N} \underline{x}^H \underline{x}$ y que por tanto, si se asume la ergodicidad

del proceso, se cumple que $\hat{R}_{xx} \simeq R_{xx}$. De esta forma, el estadístico se puede expresar como:

$$T(\underline{x}) = \frac{|\hat{R}_{xd}|^2}{R_{xx}} \simeq \hat{R}_{xd} R_{xx}^{-1} \hat{R}_{xd}^* = VV^* = |V|^2 \quad (4-56)$$

Donde se descompone R_{xx} como:

$$R_{xx} = QQ^* \quad (4-57)$$

De esta forma se define la variable aleatoria V como:

$$V = Q^{-1} \hat{R}_{xd} : \mathbb{C}N\left(\mu_V, \sigma_V^2\right) \quad (4-58)$$

Si se calcula la esperanza de la variable V se obtiene:

$$\mu_V = E(V) = Q^{-1} E(\hat{R}_{xd}) = Q^{-1} \mu_{\hat{R}_{xd}} \quad (4-59)$$

Y al calcular la varianza:

$$\sigma_V^2 = \text{var}(Q^{-1} \hat{R}_{xd}) = |Q^{-1}|^2 \sigma_{\hat{R}_{xd}}^2 \simeq |Q^{-1}|^2 \frac{\hat{R}_{xx}}{N} \simeq |Q^{-1}|^2 \frac{R_{xx}}{N} = |Q^{-1}|^2 \frac{|Q|^2}{N} = \frac{1}{N} \quad (4-60)$$

Así, finalmente, la variable V queda caracterizada como:

$$V : \mathbb{C}N\left(Q^{-1} \mu_{\hat{R}_{xd}}, \frac{1}{N}\right) \quad (4-61)$$

En el caso de la hipótesis nula la variable V tiene una estadística:

$$V_{H_0} : \mathbb{C}N\left(0, \frac{1}{N}\right) \quad (4-62)$$

En el caso de la hipótesis alternativa:

$$V_{H_1} : \mathbb{C}N\left(Q^{-1} a, \frac{1}{N}\right) \quad (4-63)$$

Dado que la variable $V \in \mathbb{C}$ se puede expresar como $V = V_{\mathbb{R}} + jV_{\text{Im}}$ donde las variables que representan a la parte real y la parte imaginaria de V son también

Gaussianas y cumplen que $\sigma_{V_{\mathbb{R}}}^2 = \sigma_{V_{\text{Im}}}^2 = \frac{\sigma_V^2}{2} = \frac{1}{2N}$ y que su covarianza cruzada es nula. Para normalizar a la unidad el valor de la varianza de $V_{\mathbb{R}}$ y V_{Im} , se debe definir una nueva variable:

$$Z = \sqrt{2N} \cdot V = \sqrt{2N} \cdot V_{\mathbb{R}} + j\sqrt{2N} \cdot V_{\text{Im}} = Z_{\mathbb{R}} + jZ_{\text{Im}} \quad (4-64)$$

De esta forma las variables $Z_{\mathbb{R}}$ y Z_{Im} serán variables aleatorias Gaussianas reales normalizadas en el caso de la hipótesis nula:

$$Z_{\mathbb{R};H_0} : N(0,1) \quad y \quad Z_{\text{Im};H_0} : N(0,1) \quad (4-65)$$

Y en el caso de la hipótesis alternativa:

$$Z_{\mathbb{R};H_1} : N(\sqrt{2N} \Re(Q^{-1}a), 1) \quad y \quad Z_{\text{Im};H_1} : N(\sqrt{2N} \Im(Q^{-1}a), 1) \quad (4-66)$$

Finalmente, si se redefine el estadístico de prueba (4-56), multiplicándolo por el factor $2N$ para normalizarlo, se obtiene:

$$\Gamma(\underline{x}) = 2N \cdot T(\underline{x}) = 2N \frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}} \simeq 2N |V|^2 = |Z|^2 = Z_{\mathbb{R}}^2 + Z_{\text{Im}}^2 \quad (4-67)$$

Para la hipótesis nula, como las variables $Z_{\mathbb{R}}$ y Z_{Im} son $N(0,1)$, el estadístico de la ecuación (4-67) se caracteriza como una variable chi-cuadrado con 2 grados de libertad (dado que la variable chi-cuadrado con M grados de libertad se define como la suma de M variables gaussianas $N(0,1)$ elevadas al cuadrado) o, equivalentemente, como una variable exponencial de media 2:

$$\Gamma(\underline{x}; H_0) = Z_{\mathbb{R};H_0}^2 + Z_{\text{Im};H_0}^2 : \chi_2^2 \quad (4-68)$$

En el caso de la hipótesis alternativa, el estadístico de la ecuación (4-67) se caracteriza como una variable chi-cuadrado descentralizada con 2 grados de libertad:

$$\Gamma(\underline{x}; H_1) = Z_{\mathbb{R};H_1}^2 + Z_{\text{Im};H_1}^2 : \chi_2^2(\delta) \quad (4-69)$$

En este último caso el parámetro de descentralización se calcula como:

$$\begin{aligned}\delta &= \mu_{Z_{\mathbb{R}, H_1}}^2 + \mu_{Z_{\text{Im}, H_1}}^2 = \left(\sqrt{2N} \mathbb{R}(Q^{-1}a) \right)^2 + \left(\sqrt{2N} \text{Im}(Q^{-1}a) \right)^2 = \\ &= 2N |Q^{-1}a|^2 = 2N \frac{|a|^2}{|Q|^2} = 2N \frac{|a|^2}{R_{xx}}\end{aligned}\quad (4-70)$$

Dado que en el caso de la hipótesis H_1 la señal $\underline{x} = \underline{ad} + \underline{n}$, si se calcula la potencia de la señal \underline{x} a partir de su estimación:

$$\begin{aligned}R_{xx} &= E(\hat{R}_{xx}) = \frac{1}{N} E(\underline{x}^H \underline{x}) = \frac{1}{N} E((\underline{ad} + \underline{n})^H (\underline{ad} + \underline{n})) = \\ &= \frac{1}{N} \left(|a|^2 \underline{d}^H \underline{d} + E(\underline{n}^H \underline{n}) + a^* \underline{d}^H E(\underline{n}) + a E(\underline{n}^H) \underline{d} \right) = |a|^2 + \sigma^2\end{aligned}\quad (4-71)$$

la relación señal a ruido se puede calcular como:

$$SNR = \frac{\text{Potencia de señal útil}}{\text{Potencia de ruido}} = \frac{|a|^2}{\sigma^2} \quad (4-72)$$

Por tanto el parámetro de descentralización también puede calcularse como:

$$\delta = 2N \frac{|a|^2}{R_{xx}} = 2N \frac{|a|^2}{|a|^2 + \sigma^2} = 2N \frac{SNR}{SNR + 1} \quad (4-73)$$

Resumiendo, para una sola celda del espacio de búsqueda, el estadístico de prueba normalizado:

$$\Gamma(\underline{x}) = 2N \cdot T(\underline{x}) = 2N \frac{|\hat{R}_{xd}|^2}{\hat{R}_{xx}} \quad (4-74)$$

Se caracteriza en cada hipótesis según:

$$\left\{ \begin{array}{l} \Gamma(\underline{x}; H_0) : \chi_2^2 \\ \Gamma(\underline{x}; H_1) : \chi_2^2(\delta) \quad \text{con} \quad \delta = 2N \frac{|a|^2}{R_{xx}} = 2N \frac{SNR}{SNR + 1} \end{array} \right. \quad (4-75)$$

Por último, si se desea deshacer la normalización realizada en (4-74), el estadístico inicial expresado en (4-40) puede reescribirse como:

$$T(\underline{x}) = \frac{1}{2N} \Gamma(\underline{x}) \quad (4-76)$$

y se puede relacionar la función de densidad (PDF) de ambos estadísticos de la siguiente forma:

$$p_T(t) = 2N \cdot p_{\Gamma}(2Nt) \quad (4-77)$$

así como relacionar sus funciones de distribución acumuladas (CDFs):

$$F_T(t) = F_{\Gamma}(2Nt) \quad (4-78)$$

lo que en el caso de la hipótesis H_0 da lugar a una variable exponencial de parámetro $\lambda = N$.

A partir de la distribución de probabilidad del estadístico de prueba, ya se está en disposición de obtener valores numéricos para las probabilidades de detección (4-43), falsa alarma (4-44) y detección fallida (4-45) para una celda definidas en la sección anterior.

Una vez se ha hallado como calcular estas probabilidades se pasa a extender los resultados a todo el espacio de búsqueda y a partir de ellos obtener el umbral de decisión.

4.7.4 Extensión de los resultados al espacio de búsqueda

4.7.4.1 Condiciones iniciales

Para analizar todo el espacio de búsqueda, en [4-14] se establecen ciertas presunciones:

1. La hipótesis alternativa H_1 se verifica tan sólo en una celda de todo el espacio de búsqueda, y por tanto se asume que el lóbulo principal de la correlación es lo suficientemente estrecho para afectar tan sólo a una celda.
2. Cuando el satélite buscado está presente, tan sólo hay una variable aleatoria de tipo $T(\underline{x}; H_1)$ en todo el espacio de búsqueda, el resto se corresponden a variables de tipo $T(\underline{x}; H_0)$.
3. La variable $T(\underline{x}; H_1)$ puede estar en cualquier celda con una probabilidad uniforme de $\frac{1}{N_{cells}}$, donde N_{cells} se corresponde con el número de celdas del espacio de búsqueda.

4. Todas las variables aleatorias que caracterizan a las celdas del espacio de búsqueda son independientes entre sí.

Además, comentar que en caso de ausencia del satélite buscado en la ejecución de la adquisición, todas las celdas del espacio de búsqueda quedan caracterizadas por variables $T(\underline{x}; H_0)$.

4.7.4.2 Planteamiento de las probabilidades de detección, falsa alarma y detección fallida para el espacio de búsqueda

En base a las condiciones planteadas en la sección anterior se está en disposición de formular las probabilidades que caracterizan a todo el espacio de búsqueda. Recordando la expresión del estadístico de prueba para todo el *grid*, definida en (4-41) como el máximo valor de los estadísticos de todas las celdas del espacio de búsqueda $(T_{GS}(\underline{x}) = \max T_i(\underline{x}) \quad \forall i = 1, \dots, N_{cells})$, la probabilidad de detección se corresponde con la probabilidad de que el máximo se produzca en la celda donde se encuentran los parámetros de sincronización, y el valor de este máximo supere el umbral de decisión:

$$P_D = P\left(\{T_{GS;H_1}(\underline{x}) = T(\underline{x}; H_1)\} \cap \{T_{GS;H_1}(\underline{x}) > \gamma_{th}\}\right) \quad (4-79)$$

La probabilidad de falsa alarma en presencia del satélite buscado se corresponde con la probabilidad de que el máximo del espacio de búsqueda supere el umbral de decisión pero no se produzca en la celda donde se encuentran los parámetros de sincronización:

$$P_{FA}^p = P\left(\{T_{GS;H_1}(\underline{x}) \neq T(\underline{x}; H_1)\} \cap \{T_{GS;H_1}(\underline{x}) > \gamma_{th}\}\right) \quad (4-80)$$

En este caso se produce una falsa alarma dado que, aunque la adquisición da un resultado positivo para el código PRN del satélite que se está buscando, los parámetros de sincronización estimados no son correctos.

La probabilidad de falsa alarma en ausencia de satélite se corresponde con la probabilidad de que el máximo del espacio de búsqueda supere el umbral de decisión cuando el satélite buscado no está presente:

$$P_{FA}^a = P(T_{GS;H_0}(\underline{x}) > \gamma_{th}) \quad (4-81)$$

Por último, la probabilidad de detección fallida se corresponde con la probabilidad del máximo del espacio de búsqueda no supere el umbral de decisión cuando el satélite de decisión sí está presente:

$$P_{MD} = P(T_{GS;H_1}(\underline{x}) \leq \gamma_{th}) \quad (4-82)$$

En las siguientes secciones se describe la forma de diseñar el umbral a partir de las probabilidades anteriores.

4.7.4.3 Detector CFAR

El algoritmo utilizado para el cálculo del umbral de decisión consiste en fijar un valor de la probabilidad de falsa alarma y aislar el umbral de la expresión de dicha probabilidad. Posteriormente, con éste valor del umbral se calculan las probabilidades de detección y detección fallida.

El valor de la probabilidad de detección obtenido con este método será máximo para el valor de probabilidad de falsa alarma fijado ya que el estadístico de prueba utilizado se ha diseñado a partir del *Generalized Likelihood Ratio Test* (GLRT) definido en (4-17).

El algoritmo implementado es conocido con el nombre de detector CFAR (*Constant False Alarm Rate*) dado que la probabilidad de falsa alarma se fija de antemano y se mantiene constante durante la ejecución de todo el algoritmo.

Para poder aplicar el algoritmo se debe decidir cuál de las dos probabilidades de falsa alarma se va a utilizar para la obtención del umbral, la probabilidad de falsa alarma en ausencia de satélite (P_{FA}^a) definida en (4-81) o la probabilidad de falsa alarma en presencia de satélite (P_{FA}^p) definida en (4-80). El cálculo de esta última entraña una mayor dificultad dado que depende de parámetros que el receptor no conoce, aunque dicha dificultad puede verse reducida si se calcula como:

$$P_{FA}^p = 1 - P_D - P_{MD} \quad (4-83)$$

De todas formas, en [4-14] se demuestra experimentalmente que esta probabilidad siempre es menor o igual que la que se obtiene en ausencia del satélite, para cualquier valor del umbral, es decir:

$$P_{FA}^a \geq P_{FA}^p \quad \forall \gamma_{th} \quad (4-84)$$

Por este motivo se decide utilizar la probabilidad de falsa alarma en ausencia de satélite (P_{FA}^a) como el parámetro para fijar el umbral de decisión dado que resulta en un mejor indicador del rendimiento del sistema y, además, acota el valor de la probabilidad de falsa alarma en presencia de satélite (P_{FA}^p).

4.7.4.3.1 Probabilidad de falsa alarma y cálculo del umbral.

Para la búsqueda del umbral de decisión, el primer paso consiste en desarrollar la probabilidad de falsa alarma en ausencia de satélite (P_{FA}^a) definida en (4-81) y aislar el valor del umbral de decisión. Para ello se puede aplicar el concepto de que para que el máximo de un conjunto de elementos esté por debajo de un valor, todos los elementos de dicho conjunto deben estar por debajo de ese valor. Es decir:

$$\begin{aligned}
 P_{FA}^a &= P(T_{GS;H_0}(\underline{x}) > \gamma_{th}) = 1 - P(T_{GS;H_0}(\underline{x}) \leq \gamma_{th}) = 1 - P(\max T_i(\underline{x}) \leq \gamma_{th}) = \\
 &= 1 - P(\{T_1(\underline{x}) \leq \gamma_{th}\} \cap \dots \cap \{T_{N_{cells}}(\underline{x}) \leq \gamma_{th}\}) = 1 - \prod_{i=1}^{N_{cells}} P(T_i(\underline{x}) \leq \gamma_{th})
 \end{aligned} \tag{4-85}$$

Dado que en el caso de ausencia de satélite, los estadísticos de todas las celdas del espacio de búsqueda se corresponden con variables que siguen una distribución $T(\underline{x}; H_0)$ se puede expresar la probabilidad de falsa alarma calculada en (4-85) en función de la función de densidad (PDF) o de la función de distribución acumulada (CDF) del estadístico de prueba:

$$P_{FA}^a = 1 - \left(P(T_{H_0}(\underline{x}) \leq \gamma_{th}) \right)^{N_{cells}} = 1 - \left(\int_{-\infty}^{\gamma_{th}} p_{T;H_0}(t) dt \right)^{N_{cells}} = 1 - \left(F_{T;H_0}(\gamma_{th}) \right)^{N_{cells}} \tag{4-86}$$

Fijando el valor de la probabilidad de falsa alarma se puede despejar el valor de la CDF (o de la PDF) del estadístico de prueba para una celda en la hipótesis H_0 :

$$F_{T;H_0}(\gamma_{th}) = \sqrt[N_{cells}]{1 - P_{FA}^a} \tag{4-87}$$

Dado que, como se ha visto en el apartado anterior (concretamente en las expresiones (4-76) y (4-75)), dicho estadístico se puede relacionar fácilmente con una variable aleatoria chi-cuadrado con 2 grados de libertad (o una variable exponencial de parámetro $\lambda = N$), se puede aprovechar la existencia, tanto en Matlab como en algunas librerías de C++, de funciones que proporcionan la inversa de su CDF y poder así encontrar el valor del umbral a partir de:

$$\gamma_{th} = F_{T;H_0}^{-1} \left(\sqrt[N_{cells}]{1 - P_{FA}^a} \right) \tag{4-88}$$

Este es el método implementado en el algoritmo de adquisición de GNSS-SDR para el cálculo del umbral.

Se va a formular a continuación una nueva forma de obtener el umbral a partir de las definiciones de la función de densidad (PDF) y distribución (CDF) para el estadístico de prueba de todo el espacio de búsqueda en la hipótesis H_0 .

Partiendo de la función de distribución acumulada (CDF) del estadístico de prueba para todo el *grid search* y de la relación entre ambos estadísticos $(T_{GS}(\underline{x}) = \max T_i(\underline{x}) \quad \forall i = 1, \dots, N_{cells})$ de la ecuación (4-41), se obtiene:

$$F_{T_{GS}}(\gamma) = P(T_{GS}(\underline{x}) \leq \gamma) = P(\max(T_i(\underline{x})) \leq \gamma) \quad \forall i = 1, \dots, N_{cells} \tag{4-89}$$

Asumiendo que las celdas son independientes, y volviendo a aplicar el concepto de que para que el máximo de un conjunto de elementos sea inferior a un

determinado valor todos los elementos del conjunto deben ser inferiores a dicho valor, se desarrolla la expresión anterior:

$$F_{T_{GS}}(\gamma) = P\left(\bigcap_{i=1}^{N_{cells}} (T_i(\underline{x}) \leq \gamma)\right) = \prod_{i=1}^{N_{cells}} P(T_i(\underline{x}) \leq \gamma) = \prod_{i=1}^{N_{cells}} F_{T_i}(\gamma) \quad (4-90)$$

Si se particulariza la expresión anterior para la hipótesis H_0 , se puede afirmar que en este caso todas las variables aleatorias tendrán la misma distribución que $T(\underline{x}; H_0)$ y por tanto la CDF de la expresión (4-90) puede reescribirse como:

$$F_{T_{GS};H_0}(\gamma) = (F_{T;H_0}(\gamma))^{N_{cells}} \quad (4-91)$$

A partir de la relación entre la función de distribución (CDF) y la función de densidad (PDF), derivando la expresión (4-91) se obtiene la relación entre las PDFs:

$$\begin{aligned} p_{T_{GS};H_0}(\gamma) &= \frac{\partial}{\partial \gamma} (F_{T_{GS};H_0}(\gamma)) = \frac{\partial}{\partial \gamma} \left((F_{T;H_0}(\gamma))^{N_{cells}} \right) = \\ &= N_{cells} (F_{T;H_0}(\gamma))^{N_{cells}-1} p_{T;H_0}(\gamma) \end{aligned} \quad (4-92)$$

De esta forma, una vez obtenida la PDF del estadístico de prueba para todo el espacio de búsqueda en la hipótesis H_0 se puede reformular la probabilidad de falsa alarma en ausencia de satélite de la expresión (4-81), substituyendo en ella la expresión (4-92) :

$$P_{FA}^a = \int_{\gamma_{th}}^{\infty} p_{T_{GS};H_0}(t) dt = \int_{\gamma_{th}}^{\infty} N_{cells} (F_{T;H_0}(\gamma))^{N_{cells}-1} p_{T;H_0}(\gamma) dt \quad (4-93)$$

Aislar el valor del umbral de esta expresión no es evidente, pero si se relaciona la función de distribución acumulada de $T(\underline{x}; H_0)$ con la probabilidad de falsa alarma para una celda calculada en (4-44), se puede recalcular la expresión anterior y aproximarla (en el caso de valores pequeños de $P_{fa}(\gamma)$) de la siguiente forma:

$$P_{FA}^a = \int_{\gamma_{th}}^{\infty} N_{cells} (1 - P_{fa}(\gamma))^{N_{cells}-1} p_{T;H_0}(\gamma) dt \approx N_{cells} \int_{\gamma_{th}}^{\infty} p_{T;H_0}(\gamma) dt \quad (4-94)$$

En el fondo, dicha aproximación equivale a:

$$p_{T_{GS};H_0}(\gamma) = N_{cells} (F_{T;H_0}(\gamma))^{N_{cells}-1} p_{T;H_0}(\gamma) \underset{\substack{N_{cells} \gg 1 \\ \gamma > \gamma_{th}}}{\simeq} N_{cells} \cdot p_{T;H_0}(\gamma) \quad (4-95)$$

Dicha afirmación puede comprobarse en la Figura 4-7, donde se muestran las PDFs exacta y aproximadas de la expresión (4-95) para el caso en que el estadístico

de prueba $T(\underline{x}; H_0)$ siga una distribución exponencial con $\lambda = N$ tal y como se dedujo en el apartado 4.7.3.

La gráfica se ha realizado utilizando $N_{cells} = 164000$. Este número resulta de utilizar 41 bins frecuenciales, que se corresponden con el análisis de un espacio de búsqueda de ± 10 kHz y 500 Hz para cada bin frecuencial y 4000 bins de código, que son los utilizados por el algoritmo *Parallel Code Phase Search Acquisition* comentado en el apartado 4.5.3.2 en el caso de utilizar una señal muestreada a 4 Msps. En la gráfica podemos apreciar que para valores de γ_{th} superiores a 0.004 la aproximación es completamente válida.

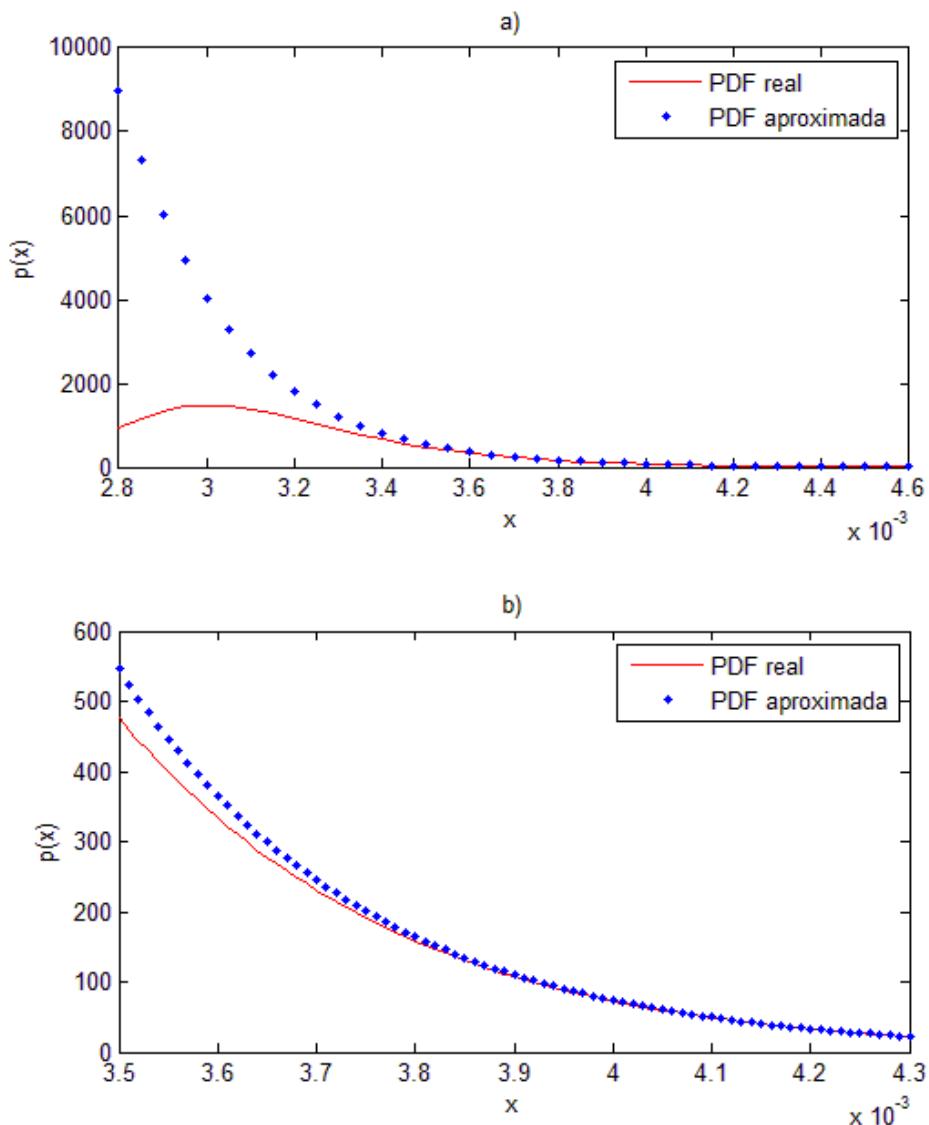


Figura 4-7: 2 representaciones de las PDFs real y aproximadas del estadístico de prueba (sin normalizar por el factor $2N$) formuladas en la expresión (4-95) para todo el espacio de búsqueda en la hipótesis H_0 cuando el estadístico de prueba normalizado de una celda sigue una distribución chi-cuadrado con 2 grados de libertad. En a) puede observarse la forma general de ambas PDFs, mientras que b) muestra un zoom sobre la zona de convergencia de ambas gráficas, que se da para valores del umbral entorno a $3-4 \times 10^{-3}$.

Analizando la expresión (4-93) puede observarse que la dificultad que entraña aislar el umbral si se utiliza la expresión exacta de la PDF de $T_{GS}(\underline{x}; H_0)$ desaparece al realizar la aproximación (4-94).

Expresando la probabilidad de falsa alarma en función de la función de distribución (CDF) para el estadístico de una celda:

$$P_{FA}^a \approx N_{cells} \int_{\gamma_{th}}^{\infty} p_{T;H_0}(\gamma) dt = N_{cells} \left(1 - F_{T;H_0}(\gamma_{th})\right) \quad (4-96)$$

Si se aísla la CDF:

$$F_{T;H_0}(\gamma_{th}) \approx 1 - \frac{P_{FA}^a}{N_{cells}} \quad (4-97)$$

Y el umbral se podría obtener como:

$$\gamma_{th} \approx F_{T;H_0}^{-1} \left(1 - \frac{P_{FA}^a}{N_{cells}}\right) \quad (4-98)$$

De la misma forma que para el primer umbral calculado (4-88) (que se obtiene a partir de lo explicado en [4-14]) existen funciones en Matlab y C++ que nos proporcionan la inversa de la CDF del estadístico $T(\underline{x}; H_0)$ para poder encontrar el valor del umbral. La ventaja radica en este caso en que no es necesario realizar una raíz N_{cells} -ésima, tan sólo una división y una resta, y por tanto conlleva un menor coste computacional.

4.7.4.3.2 Probabilidades de detección y detección fallida.

Una vez calculado el valor del umbral de decisión γ_{th} a partir de la expresión (4-88) o de la expresión aproximada en (4-98), se pasa a utilizar este valor para calcular las probabilidades de detección y detección fallida planteadas en (4-79) y (4-82) respectivamente.

En cuanto a la probabilidad de detección para todo el espacio de búsqueda, partiendo de la expresión (4-79), ésta puede reescribirse como:

$$P_D = P \left(\left\{ T(\underline{x}; H_1) = \max(T_i(\underline{x})) \right\} \cap \left\{ T(\underline{x}; H_1) > \gamma_{th} \right\} \right) \quad (4-99)$$

Si $T(\underline{x}; H_1)$ es igual al máximo de todos los estadísticos de las celdas del espacio de búsqueda, su valor debe ser igual o superior al de todos estos estadísticos, es decir:

$$P_D = P\left(\{T(\underline{x}; H_1) \geq T_i(\underline{x})\} \cap \dots \cap \{T(\underline{x}; H_1) \geq T_{N_{cells}}(\underline{x})\} \cap \{T(\underline{x}; H_1) > \gamma_{th}\}\right) \quad (4-100)$$

Utilizando el teorema de la probabilidad total para variables aleatorias continuas, la probabilidad puede expresarse como:

$$\begin{aligned} P_D &= \int_{\gamma_{th}}^{\infty} P\left(\bigcap_{i=1}^{N_{cells}} T(\underline{x}; H_1) \geq T_i(\underline{x}) / T(\underline{x}; H_1) = \gamma\right) f_{T;H_1}(\gamma) d\gamma = \\ &= \int_{\gamma_{th}}^{\infty} \prod_{i=1}^{N_{cells}} P(T(\underline{x}; H_1) \geq T_i(\underline{x}) / T(\underline{x}; H_1) = \gamma) f_{T;H_1}(\gamma) d\gamma \end{aligned} \quad (4-101)$$

En el productorio de la expresión anterior hay que tener en cuenta que todos los estadísticos $T_i(\underline{x})$ siguen una distribución $T(\underline{x}; H_0)$ excepto uno que sigue la distribución $T(\underline{x}; H_1)$. De esta forma, de los N_{cells} términos del productorio, $N_{cells} - 1$ se pueden calcular como:

$$\begin{aligned} P(T(\underline{x}; H_1) \geq T_i(\underline{x}) / T(\underline{x}; H_1) = \gamma) &= P(T(\underline{x}; H_1) \geq T(\underline{x}; H_0) / T(\underline{x}; H_1) = \gamma) = \\ &= P(\gamma \geq T(\underline{x}; H_0) / T(\underline{x}; H_1) = \gamma) = P(T(\underline{x}; H_0) \leq \gamma) = F_{T;H_0}(\gamma) \end{aligned} \quad (4-102)$$

mientras que el término restante daría como resultado una probabilidad igual a la unidad:

$$\begin{aligned} P(T(\underline{x}; H_1) \geq T_i(\underline{x}) / T(\underline{x}; H_1) = \gamma) &= P(T(\underline{x}; H_1) \geq T(\underline{x}; H_1) / T(\underline{x}; H_1) = \gamma) = \\ P(\gamma \geq \gamma / T(\underline{x}; H_1) = \gamma) &= P(\gamma \geq \gamma) = 1 \end{aligned} \quad (4-103)$$

Substituyendo las expresiones (4-102) y (4-103) en la probabilidad de detección calculada en (4-101), encontramos la siguiente expresión:

$$P_D = \int_{\gamma_{th}}^{\infty} \left(F_{T;H_0}(\gamma)\right)^{N_{cells}-1} f_{T;H_1}(\gamma) d\gamma \quad (4-104)$$

Dada la relación que existe entre la función de distribución de $T(\underline{x}; H_0)$ y la probabilidad de falsa alarma para una celda expresada en (4-44) podemos de nuevo aproximar la expresión (para valores de $P_{fa}(\gamma)$ pequeños) como:

$$P_D = \int_{\gamma_{th}}^{\infty} \left(1 - P_{fa}(\gamma)\right)^{N_{cells}-1} f_{T;H_1}(\gamma) d\gamma \approx \int_{\gamma_{th}}^{\infty} f_{T;H_1}(\gamma) d\gamma = 1 - F_{T;H_1}(\gamma) \quad (4-105)$$

Por último se procede a calcular la probabilidad de detección fallida. Partiendo de la expresión (4-82):

$$\begin{aligned}
 P_{MD} &= P\left(\max\left(T_i(\underline{x})\right) \leq \gamma_{th} / H_1\right) = P\left(\bigcap_{i=1}^{N_{cells}} T_i(\underline{x}) \leq \gamma_{th} / H_1\right) = \\
 &= \prod_{i=1}^{N_{cells}} P\left(T_i(\underline{x}) \leq \gamma_{th}\right) = \prod_{i=1}^{N_{cells}} F_{T_i}(\gamma_{th}) = \left(F_{T;H_0}(\gamma_{th})\right)^{N_{cells}-1} F_{T;H_1}(\gamma_{th})
 \end{aligned} \tag{4-106}$$

donde de nuevo se ha vuelto a aplicar que en la hipótesis H_1 los estadísticos de todas las celdas del espacio de búsqueda quedan caracterizados por una variable de tipo $T(\underline{x}; H_0)$, excepto el de una celda que queda caracterizado por $T(\underline{x}; H_1)$.

En el próximo apartado se presentan los bloques diseñados que implementan el algoritmo *Parallel Code Phase Search Acquisition* para el software GNSS-SDR y se verifican los cálculos teóricos presentados en este apartado y los apartados previos de este capítulo.

4.8 Bloques diseñados

Para su inclusión en el software GNSS-SDR se han desarrollado los siguientes bloques:

- *AcquisitionInterface*: interfaz común a todos los bloques de adquisición en el software GNSS-SDR.
- *GpsL1CaPcpsAcquisition*: bloque que se encarga de adaptar el bloque *pcps_acquisition_cc* a la interfaz *AcquisitionInterface* para señales GPS L1 C/A.
- *pcps_acquisition_cc*: bloque GNU Radio [4-15] que implementa el algoritmo *Parallel Code Phase Search Algorithm* explicado en el apartado 4.5.3 independientemente del sistema utilizado (GPS o Galileo).

En las siguientes secciones se explica con detalle las características más relevantes de cada uno de estos bloques.

4.8.1 *AcquisitionInterface*

El bloque *AcquisitionInterface* ha sido modificado a partir de una implementación inicial diseñada por el ingeniero informático Sr. Carlos Avilés, quien realizó el diseño inicial del esqueleto de GNSS-SDR, y con quien el autor de este proyecto ha colaborado en [4-16].

Como se ha comentado inicialmente, *AcquisitionInterface* es una interfaz de una clase abstracta para los algoritmos de adquisición de GNSS-SDR. Dado que todos los métodos son virtuales, esta clase no puede ser instanciada directamente, y sólo se pueden crear instancias directamente de una subclase de ésta si todos los métodos virtuales puros heredados se implementan por esa subclase o una clase padre de la misma.

Los métodos virtuales que deben ser implementados en los diferentes bloques de adquisición son:

- *virtual void set_gnss_synchro (Gnss_Synchro* gnss_synchro)*: el objetivo de este método es que el canal proporcione al bloque de adquisición un puntero a un objeto *Gnss_Synchro*, que es el objeto donde se guardan (entre otros datos) los parámetros de sincronización. De esta forma, cuando el bloque de adquisición obtenga un resultado positivo, puede escribir los parámetros de sincronización obtenidos en dicho objeto, y cuando se active el bloque de *tracking* (que también tiene un puntero a dicho objeto), éste tan sólo tiene que leer los resultados allí guardados.
- *virtual void set_channel (unsigned int channel)*: el objetivo de este método es que el *flowgraph* asigne a cada bloque de adquisición el identificador de canal (dentro del receptor) que le corresponde. Esta función es de relevante importancia para la correcta utilización del *logging* del sistema GNSS-SDR explicado en el apartado 2.4.4, ya que como se paralleliza la ejecución de los canales en diferentes hilos concurrentes, se hace necesaria una identificación de los mismos para realizar un seguimiento de su ejecución.
- *virtual void set_threshold (float threshold)*: este método permite asignar el valor del umbral de decisión a utilizar en el algoritmo de adquisición. La finalidad de esta función es poder asignar el valor numérico del umbral directamente desde el fichero de configuración de GNSS-SDR en aquellas implementaciones de la adquisición donde los cálculos presentados en el apartado 4.7 no puedan aplicarse (algoritmos de adquisición con información ‘a priori’, con múltiples dwells, etc ...) o a partir de una función que calcule el valor del umbral a partir de la probabilidad de falsa alarma tal y como se ha explicado en dicho apartado.
- *virtual void set_doppler_max (unsigned int doppler_max)*: permite al canal asignar el valor máximo del eje frecuencial del espacio de búsqueda para las implementaciones del algoritmo de adquisición que lo necesiten, como el método *Parallel Code Phase Search Algorithm*.
- *virtual void set_doppler_step (unsigned int doppler_step)*: este método permite asignar el tamaño de los bins frecuenciales del espacio de búsqueda.
- *virtual void set_channel_queue (concurrent_queue<int> * channel_internal_queue)*: este método asigna a cada bloque de adquisición un puntero a la cola concurrente que posee el canal y que permite comunicarse con los bloques de adquisición y *tracking* para activarlos o desactivarlos en función de sus resultados. La interacción entre estos bloques se explica detalladamente en el Capítulo 5 de este proyecto.
- *virtual void init()*: este método permite inicializar el algoritmo de adquisición (asignar valores iniciales, calcular los códigos PRN que se van a utilizar para correlar, ...).
- *virtual signed int mag()*: este método permite acceder al valor de una magnitud determinada del algoritmo de adquisición (por ejemplo el máximo del estadístico de prueba).
- *virtual void reset()*: este método permite reactivar el algoritmo de adquisición cuando el bloque se encuentra en estado de espera y debe despertarse y

empezar a adquirir porque el bloque de *tracking* no puede seguir estimando los parámetros de sincronización.

Los métodos citados anteriormente deben ser implementados por todas los bloques de adquisición a utilizar en el sistema GNSS-SDR. A continuación se procede a explicar los bloques implementados para el algoritmo *Parallel Phase Search Acquisition* explicado en el apartado 4.5.3.

4.8.2 Bloques que realizan la implementación del algoritmo *Parallel Phase Search Acquisition* para la señal GPS L1 C/A.

En este apartado se detallan los métodos principales de los bloques *GpsL1CaPcpsAcquisition* y *pcps_acquisition_cc*, que son los bloques que se encargan de la implementación del algoritmo *Parallel Code Phase Search Acquisition* para la señal GPS L1 C/A. El primero de ellos es un adaptador que, como ya se explicó en el apartado 2.5.2, es una clase que se encarga de encapsular a los bloques de GNU Radio para poder utilizarlos en el *flowgraph* de GNSS-SDR, mientras que el segundo es un bloque GNU Radio que se encarga del procesado de señal.

Ambas implementaciones se han realizado a partir de un diseño inicial del Dr. Javier Arribas Lázaro, coautor, entre otros trabajos, de [4-9] y con quien el autor de este proyecto ha colaborado en [4-16], [4-17] y [4-18].

El trabajo realizado ha consistido en rediseñar ambos bloques con el fin de independizar el algoritmo *Parallel Code Phase Search Acquisition* de la parte que depende propiamente del tipo de señal al que se aplique (GPS L1 C/A, Galileo E1, ...), así como añadir funcionalidades a la implementación de dichos bloques, como el cálculo automático del umbral de decisión a partir de la probabilidad de falsa alarma cuyos cálculos teóricos se han presentado en los apartados 4.6 y 4.7 y se comprueban experimentalmente en el apartado 4.9.

4.8.2.1 *GpsL1CaPcpsAcquisition*

La clase *GpsL1CaPcpsAcquisition* hereda directamente de *AcquisitionInterface* y es básicamente un adaptador que encapsula y conecta la clase de GNU Radio *gr::blocks::stream_to_vector* [4-19] (que se encarga de convertir el flujo de datos continuo en vectores de muestras) con la clase *pcps_acquisition_cc* (que es quien realmente implementa el algoritmo *Parallel Code Phase Search Acquisition*).

Los métodos públicos propios (no heredados de *AcquisitionInterface*) más relevantes de *GpsL1CaPcpsAcquisition* son:

- *GpsL1CaPcpsAcquisition(ConfigurationInterface* configuration, std::string role, unsigned int in_streams, unsigned int out_streams, gr_msg_queue_sptr queue)*: es el constructor público de la clase. Entre otras funciones, se encarga de leer del fichero de configuración los parámetros relevantes para la ejecución del algoritmo (frecuencia de muestreo, frecuencia intermedia, probabilidad de falsa alarma, ...). El acceso al fichero de configuración se realiza a través de un puntero al objeto de la clase *ConfigurationInterface* que es la que se encarga de gestionar la lectura de estos datos. Otra de las funciones relevantes del constructor es instanciar los dos objetos de las clases *gr::blocks::stream_to_vector* y *pcps_acquisition_cc*.

- `void connect(gr_top_block_sptr top_block)`: método que se encarga de conectar las instancias de los bloques de las clases `gr::blocks::stream_to_vector` y `pcps_acquisition_cc`.
- `void disconnect(gr_top_block_sptr top_block)`: desconecta las instancias de los bloques de las clases `gr::blocks::stream_to_vector` y `pcps_acquisition_cc`.

Los métodos públicos heredados de `AcquisitionInterface` son:

- `virtual void set_gnss_synchro (Gnss_Synchro* gnss_synchro);`
- `virtual void set_channel (unsigned int channel);`
- `virtual void set_doppler_max (unsigned int doppler_max);`
- `virtual void set_doppler_step (unsigned int doppler_step);`
- `virtual void set_channel_queue (concurrent_queue<int> * channel_internal_queue);`

Estos métodos encapsulan la asignación de los valores de diversas variables (y de punteros a objetos) que realmente se llevan a cabo a través de los métodos del mismo nombre de la clase `pcps_acquisition_cc`, que es la clase donde se realizan las implementaciones de dichas asignaciones. Los valores de las variables y punteros a asignar se le pasan a `GpsL1CaPcpsAcquisition` mediante el fichero de configuración de GNSS-SDR o por el constructor de clase.

Otros métodos públicos heredados de `AcquisitionInterface` que se implementan directamente en el adaptador y que realizan acciones relevantes son:

- `virtual void init()`: la implementación de este método consiste fundamentalmente en la llamada a otros 3 métodos:
 1. `gps_l1_ca_code_gen_complex_sampled(code_, gnss_synchro_->PRN, fs_in_, 0)`: es un método de la librería de GNSS-SDR `gps_sdr_signal_processing.h` creada para el tratamiento de señales GPS y que se encarga de generar el código local del satélite cuyo identificador está guardado en `gnss_synchro_->PRN`, a una determinada frecuencia `fs_in_`. De esta forma, en lugar de guardar los códigos de todos los posibles satélites, se genera el código PRN local en tiempo de ejecución cuando éste se necesita.
 2. `acquisition_cc_->set_local_code(code_)`: es un método público de la clase `pcps_acquisition_cc` que asigna el código PRN calculado por el método anterior para ser utilizado localmente en el algoritmo de adquisición.
 3. `acquisition_cc_->init()`: método público de la clase `pcps_acquisition_cc` que realiza las tareas de inicialización propias del algoritmo *Parallel Phase Search Acquisition*.
- `virtual void reset()`: como se comentó en el apartado anterior este método permite reactivar el algoritmo de adquisición cuando el bloque se encuentra en estado de espera y debe despertarse y empezar a adquirir porque el bloque de *tracking* no puede seguir estimando los parámetros de sincronización. Para ello

ejecuta `acquisition_cc->set_active(true)` donde `set_active` es un método de la clase `pcps_acquisition_cc` que asigna el valor `true` a una variable que es la encargada de controlar si el bloque `pcps_acquisition_cc` se encuentra en estado de espera o adquiriendo.

- `virtual void set_threshold (float threshold)`: este método debe ser llamado por el canal después de instanciar el bloque de adquisición. En la llamada a la función se le pasa el valor del umbral que previamente el canal ha leído del fichero de configuración. En la ejecución de este método se comprueba si en el fichero de configuración el usuario ha definido un valor para la probabilidad de falsa alarma y, si ésta se encuentra definida, se llama al método privado `float calculate_threshold(float pfa)` que devuelve el valor del umbral para dicha probabilidad de falsa alarma. Por otro lado, si la probabilidad de falsa no ha sido definida por el usuario en el fichero de configuración, se utiliza directamente el valor del umbral que le pasa el canal. Finalmente se asigna el valor del umbral al bloque de GNU Radio `pcps_acquisition_cc` (que es el que realmente realiza la adquisición) mediante el método `acquisition_cc->set_threshold(threshold_)`.

Por último, se ha añadido a la clase el método privado:

- `float calculate_threshold(float pfa)`: este método se encarga de calcular el umbral de decisión a partir de la probabilidad de falsa alarma. El cálculo se lleva a cabo a partir de la expresión teórica formulada en (4-88). Para la realización de los cálculos se ha incluido una librería de Boost, que es un conjunto de librerías avanzadas de C++ compatibles con la librería estándar del lenguaje. Concretamente la `boost/math/distributions/exponential.hpp` [4-20] que incluye los métodos necesarios para los cálculos estadísticos que conlleva la obtención del umbral tales como calcular la inversa de la CDF del estadístico $T(\underline{x}; H_0)$ caracterizado en el apartado 4.7.3.

En la Figura 2-7 del Capítulo 2 se puede observar la relación (mediante diagramas UML) entre las diferentes clases que forman la adquisición.

4.8.2.2 `pcps_acquisition_cc`

La clase `pcps_acquisition_cc` es la encargada de llevar a cabo el procesado de datos del algoritmo *Parallel Phase Search Acquisition* explicado en el apartado 4.5.3. Se trata de un bloque de GNU Radio (ya que hereda públicamente de la clase `gr_block` [4-21]) creado específicamente para GNSS-SDR.

A continuación se detallan los métodos públicos más relevantes de esta clase. Como la mayoría son implementaciones de los métodos virtuales puros de `AcquisitionInterface`, y ya han sido explicados detalladamente con anterioridad en el apartado 4.8.1, se realiza una breve descripción de los mismos y se procede a explicar de forma más minuciosa aquellos métodos no heredados de dicho interfaz, o aquellos cuya implementación resulte relevante para el lector:

- `void set_gnss_synchro(Gnss_Synchro* p_gnss_synchro)`: asigna el puntero de un objeto `Gnss_Synchro` común para el intercambio de datos de sincronización entre los bloques de adquisición y de *tracking*.
- `unsigned int mag()`: devuelve el valor del máximo del estadístico de prueba del espacio de búsqueda.

- `void init()`: inicializa el algoritmo de adquisición. Esto consiste en inicializar al valor nulo los parámetros de sincronización del objeto Gnss_Synchro y realizar la FFT del código PRN local así como su complejo conjugado. Cabe remarcar que el cálculo de la transformada rápida de Fourier (FFT) se realiza a través de los métodos de GNU Radio que encapsulan funciones de la librería FFTW3 [4-22], donde se implementa un método eficiente para calcular la Transformada Discreta de Fourier.
- `void set_local_code(std::complex<float> * code)`: asigna el valor del código local.
- `void set_active(bool active)`: Inicia algoritmo de adquisición, pasando del modo de espera al modo activo.
- `void set_channel(unsigned int channel)`: asigna a la adquisición un identificador de canal único.
- `void set_threshold(float threshold)`: asigna el valor del umbral para el algoritmo de adquisición
- `void set_doppler_max(unsigned int doppler_max)`: asigna el valor máximo de la frecuencia Doppler del espacio de búsqueda.
- `void set_doppler_step(unsigned int doppler_step)`: asigna el tamaño de los bins frecuenciales del espacio de búsqueda.
- `void set_channel_queue(concurrent_queue<int> *channel_internal_queue)`: asigna un puntero a la cola de mensajes interna del canal.
- `int general_work(int noutput_items, gr_vector_int &ninput_items, gr_vector_const_void_star &input_items, gr_vector_void_star &output_items)`: método que realiza el procesado de señal del algoritmo *Parallel Code Phase Search Acquisition*. Mientras la variable booleana `d_active` tiene un valor `false` el bloque está en espera y deja circular las muestras sin hacer nada, mientras que cuando cambia su estado a `true` se procede a ejecutar el algoritmo de adquisición. La ejecución de este algoritmo se divide principalmente en 5 pasos:
 1. Calcular la estimación de la potencia de señal de entrada.
 2. Recorrer el espacio de búsqueda mediante un bucle que va variando la frecuencia Doppler. En cada iteración de dicho bucle:
 - a. Realizar la convolución circular basada en la FFT entre la señal de entrada y la generada localmente. Para aumentar la eficiencia computacional en las operaciones que requieren multiplicar vectores de señales complejas se utilizan las funciones de la librería VOLK (*Vector-Optimized Library of Kernels*) [4-23], una librería diseñada para facilitar el uso por parte del programador del con conjunto de instrucciones SIMD del procesador, ya que encapsula el acceso a estas instrucciones en funciones de más alto nivel.
 - b. Registrar el valor máximo de la correlación y los parámetros de sincronización asociados a dicho máximo

- c. Calcular el estadístico de prueba y realizar la comparación con el umbral de decisión.
- d. Declarar la adquisición positiva o negativa en la cola de mensajes concurrente compartida con el canal.

Una vez explicados los cálculos teóricos y los bloques implementados para el software GNSS-SDR, en el siguiente apartado se procede a explicar las pruebas realizadas con el fin de verificar los cálculos teóricos y los resultados obtenidos en estas pruebas.

4.9 Pruebas realizadas y resultados obtenidos

Con el fin de validar los resultados teóricos obtenidos en los apartados anteriores de este capítulo se han desarrollado numerosas pruebas de todo tipo, en este apartado se presentan las más relevantes.

Las pruebas realizadas pueden agruparse en dos tipos bien diferenciados:

- Por un lado se han realizado varios test unitarios (*Unit Test*) que, como ya se explicó en el apartado 2.4.3, son trozos de código desarrollados con el único objetivo de verificar que una rutina o función del software está funcionando según se espera, así como varios test de integración, que comprueban la interacción de los bloques diseñados con otros bloques del software GNSS-SDR. Dichos test se han incluido en el repositorio del software GNSS-SDR con la finalidad de que sirvan de modelo para los desarrolladores de futuras implementaciones de algoritmos de adquisición.
- Por otro lado, se encuentran otro tipo de medidas, en este caso de rendimiento de los algoritmos diseñados, que se han llevado a cabo modificando las clases implementadas para medir cierto comportamiento del algoritmo y validar los resultados obtenidos mediante otras herramientas de análisis de datos externas al software GNSS-SDR, como Matlab.

Se procede a describir en los siguientes apartados estos dos tipos de pruebas y los resultados obtenidos.

4.9.1 Test unitarios y de integración para la adquisición

Como se explicó en el apartado 2.4.3, para la realización de los test unitarios y de integración de GNSS-SDR se utiliza el marco de test para C++ de Google (*Google C++ Testing Framework*) [4-24], una biblioteca que se ocupa de toda la infraestructura de los test, dejando que el desarrollador se centre en el contenido de los mismos.

Con el fin de verificar el comportamiento del código a testear, Google Test permite escribir aserciones (*assertions*), que son declaraciones que comprueban si una condición es verdadera. El resultado de una aserción puede ser éxito (*success*), error no fatal (*nonfatal error*) o error fatal (*fatal error*). Si se produce un error fatal, se aborta la ejecución de la función en curso, de lo contrario el programa continúa ejecutándose normalmente. Si un test falla o tiene una aserción fallida, entonces el resultado global del test es error, de lo contrario el resultado es éxito.

Como en el caso de la adquisición se quiere probar más de una funcionalidad, los diferentes test se agrupan en casos de prueba (*test cases*) que contienen uno o varios test que reflejan la estructura del código del test. Además, como hay que realizar múltiples test en un mismo caso de prueba que comparten objetos y tienen subrutinas comunes, se han agrupado en varios equipos de prueba (*test fixtures*). Estos fixtures se encuentran en el archivo

```
/src/tests/gnss_block/gps_l1_ca_pcps_acquisition_test.cc
```

En este archivo se definen tres fixtures así como las clases comunes a todos ellos. Los *fixtures* implementados son:

- *TEST_F(GpsL1CaPcpsAcquisitionTest, Instantiate)*

Se trata de un test que comprueba la instanciación de un objeto de la clase *GpsL1CaPcpsAcquisition* mediante la llamada a su constructor.

- *TEST_F(GpsL1CaPcpsAcquisitionTest, ConnectAndRun)*

Se trata de un test de integración formado por un conjunto de test unitarios que consisten en crear un *gr_top_block* [4-25] que, como se explicó en 2.5.2, es el bloque jerárquico de mayor alto nivel que representa un *flowgraph*, y ejecutarlo mediante la función *run ()*.

Este *top_block* está constituido por la interconexión de tres bloques en serie: un bloque de GNU Radio que implementa una fuente de señal (*gr_sig_source* [4-26]), seguido de un bloque de GNSS-SDR que sirve para detener el flujo de datos después de que se procesen un determinado número de muestras (*gnss_sdr_valve*) y del bloque de adquisición a testear.

- *TEST_F(GpsL1CaPcpsAcquisitionTest, ValidationOfResults)*

Se trata de un conjunto de pruebas que se encargan de la validación de los resultados de la adquisición. En él se realiza la adquisición de un milisegundo de una señal generada sintéticamente formada por un solo código PRN sin ruido y con parámetros de sincronización conocidos, esta señal ha sido generada mediante un programa realizado con Matlab y se encuentra en el repositorio de GNSS-SDR, concretamente en */src/tests/signal_samples/GPS_L1_CA_ID_1_Fs_4Mps_2ms.dat*. En primer lugar se comprueba que el resultado de la adquisición es positiva (el valor del estadístico de prueba supera el umbral de decisión) y posteriormente se compara el valor de los parámetros de sincronización asociados al máximo del estadístico de prueba del *grid* obtenidos a través del bloque de adquisición testeado con los de la señal original y si coinciden se da el test como correcto, si no, se da como fallido.

El resultado de los test implementados puede ser observado por cualquier usuario de GNSS-SDR al ejecutar el archivo *./run_test* que es uno de los dos ejecutables que se generan en la carpeta */install* cuando se compila el software del repositorio.

4.9.2 Pruebas para verificar los algoritmos implementados y su calidad

Con el fin de ir verificando paso a paso el algoritmo de adquisición implementado se han ido realizando diversas pruebas que se basan la modificación

del archivo `src/algorithms/acquisition/gnuradio_blocks/pcps_acquisition_cc.cc` para extraer ciertos valores intermedios en la ejecución de la adquisición y proceder a su posterior validación en Matlab, tanto numérica como gráficamente.

Para la realización de las pruebas de verificación se han generado, sintéticamente con Matlab (gracias al software proporcionado por el Dr. Javier Arribas Lázaro y modificado por el autor de este proyecto), seis señales de 20 segundos de duración consistentes en una señal GPS L1/CA que contiene la señal de un satélite (concretamente el PRN 11) inmersa en ruido AWGN. Las seis señales difieren entre sí en el nivel de ruido, de esta forma abarcan niveles de CN_0 que van desde 35 dBHz hasta 50 dBHz a intervalos de 3 dBHz. Todas las señales se han generado a una frecuencia de muestreo de 4 Msps (con un ancho de banda de 2 MHz), un desplazamiento Doppler f_{d_id} de 4906 Hz, un retardo de código τ_{id} de 3037 muestras y una frecuencia intermedia IF de 0 Hz (es decir, una señal compleja en banda base).

Para el análisis de dichas señales se ha fijado el número de canales del receptor a uno (mediante la opción `Channels.count=1` del fichero de configuración de GNSS-SDR), un espacio de búsqueda de ± 10 kHz para el eje frecuencial (mediante la opción `Acquisition0.doppler_max=10000`) y 500 Hz para cada bin (opción `Acquisition0.doppler_step=500`), lo que da lugar a un total de 41 bins frecuenciales, mientras que en el eje de código se utilizan 4000 bins (determinados por la frecuencia de muestreo de las señales analizadas). De esta forma, el espacio de búsqueda analizado tiene un total de $N_{cells} = 41 \times 4000 = 164000$ celdas.

4.9.2.1 Verificación de la funcionalidad del estadístico de prueba

Con el fin de verificar que el algoritmo de adquisición implementado produce un estadístico de prueba acorde a lo explicado en los apartados teóricos y que presenta un máximo muy marcado en la celda correspondiente a los parámetros de sincronización de la señal analizada se ha realizado inicialmente una prueba meramente descriptiva.

Para realizar la prueba, se hace uso de la opción `dump` implementada en `pcps_acquisition_cc.cc`. Esta opción se puede activar desde el fichero de configuración de GNSS-SDR. Si la opción está activada el algoritmo escribe a fichero el valor del módulo al cuadrado de la correlación $|\hat{R}_{xd}|^2$ que se utiliza en el estadístico de prueba descrito en (4-41) y que se obtiene de la realización de la correlación circular mediante la FFT para un bin frecuencial. De esta forma, se genera un fichero para cada bin frecuencial del espacio de búsqueda (41 ficheros en nuestro caso). Posteriormente desde Matlab se pueden leer los ficheros generados por GNSS-SDR y disponiendo su contenido en filas o columnas se puede generar una matriz que corresponde al espacio de búsqueda para el estadístico $|\hat{R}_{xd}|^2$ y realizar diversas operaciones tales como analizar la posición y el valor de su máximo o realizar una representación gráfica de todo el espacio de búsqueda.

Con el fin de verificar las 2 hipótesis descritas inicialmente en (4-14), H_0 (satélite buscado ausente) y H_1 (satélite buscado presente), se ha procesado dos veces con GNSS-SDR una de las señales sintéticas generadas con Matlab (concretamente la generada con una CN_0 de 50 dBHz). La primera vez se ha realizado la adquisición del satélite PRN 1 (no presente en la señal y por tanto correspondiente a

la hipótesis H_0) y la segunda vez se ha realizado la adquisición del satélite con PRN 11 (presente en la señal y por tanto correspondiente a la hipótesis H_1).

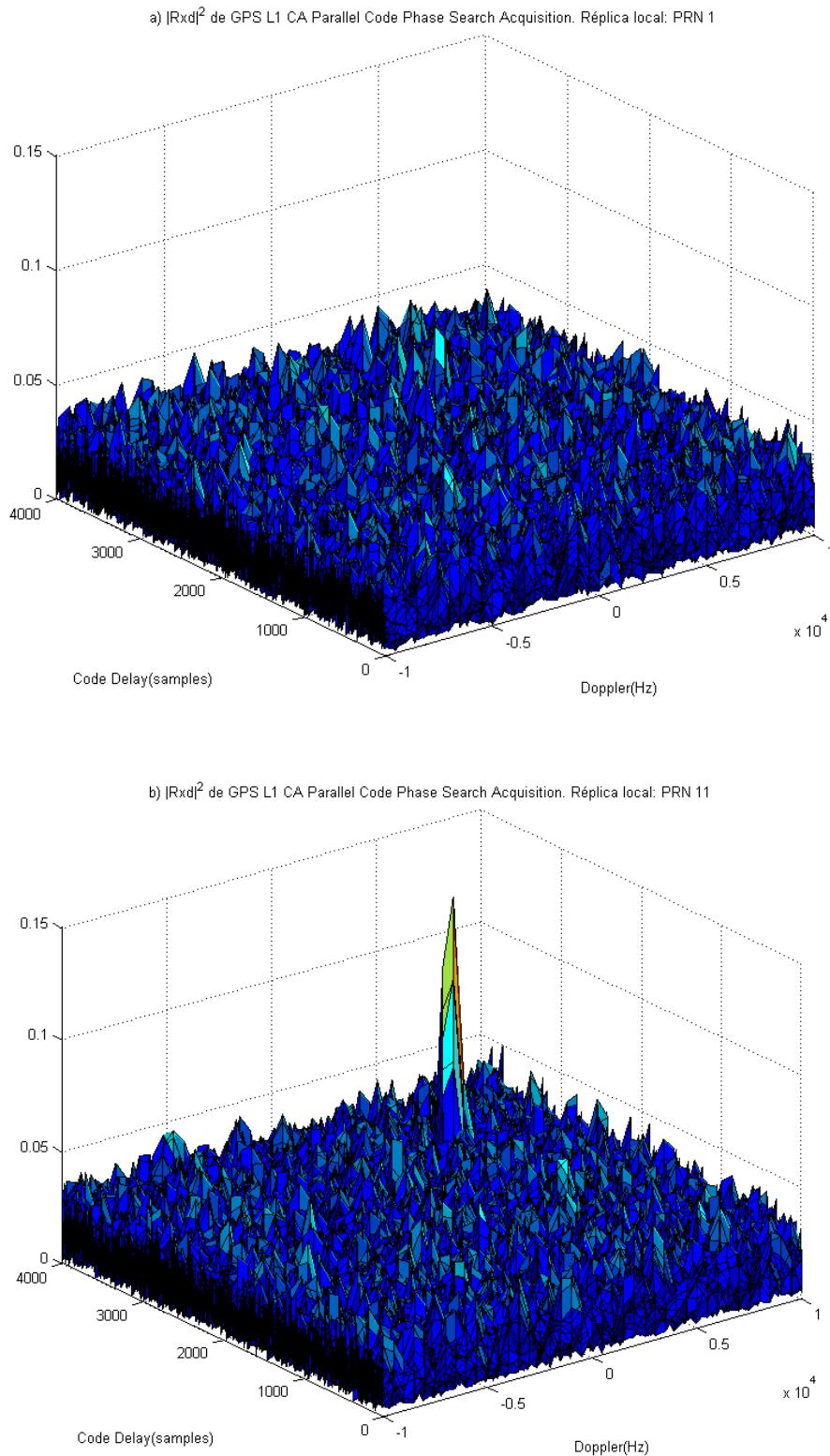


Figura 4-8: Representación del espacio de búsqueda del estadístico $|R_{xd}|^2$ a) para el satélite con PRN 1 (no presente) b) para el satélite con PRN 11 (presente).

Los ficheros resultantes de ambas ejecuciones se analizan con Matlab y se pasa a mostrar los resultados obtenidos mediante una representación gráfica del espacio de búsqueda para cada hipótesis (ver la Figura 4-8).

Concretamente, en la Figura 4-8.a se presenta el espacio de búsqueda para el código de satélite PRN 1 (ausente en la señal procesada) y como puede observarse no se aprecia ningún pico destacado en el espacio de búsqueda.

En la Figura 4-8.b se presenta el espacio de búsqueda para el código de satélite PRN 11 (presente en la señal procesada) y en él se puede apreciar la aparición de un pico destacado en las coordenadas correspondientes a los parámetros de sincronización de la señal generada sintéticamente (concretamente el máximo se produce en la celda cuyo bin frecuencial corresponde a un valor de 5 kHz y cuyo bin de código tiene un valor de 3037).

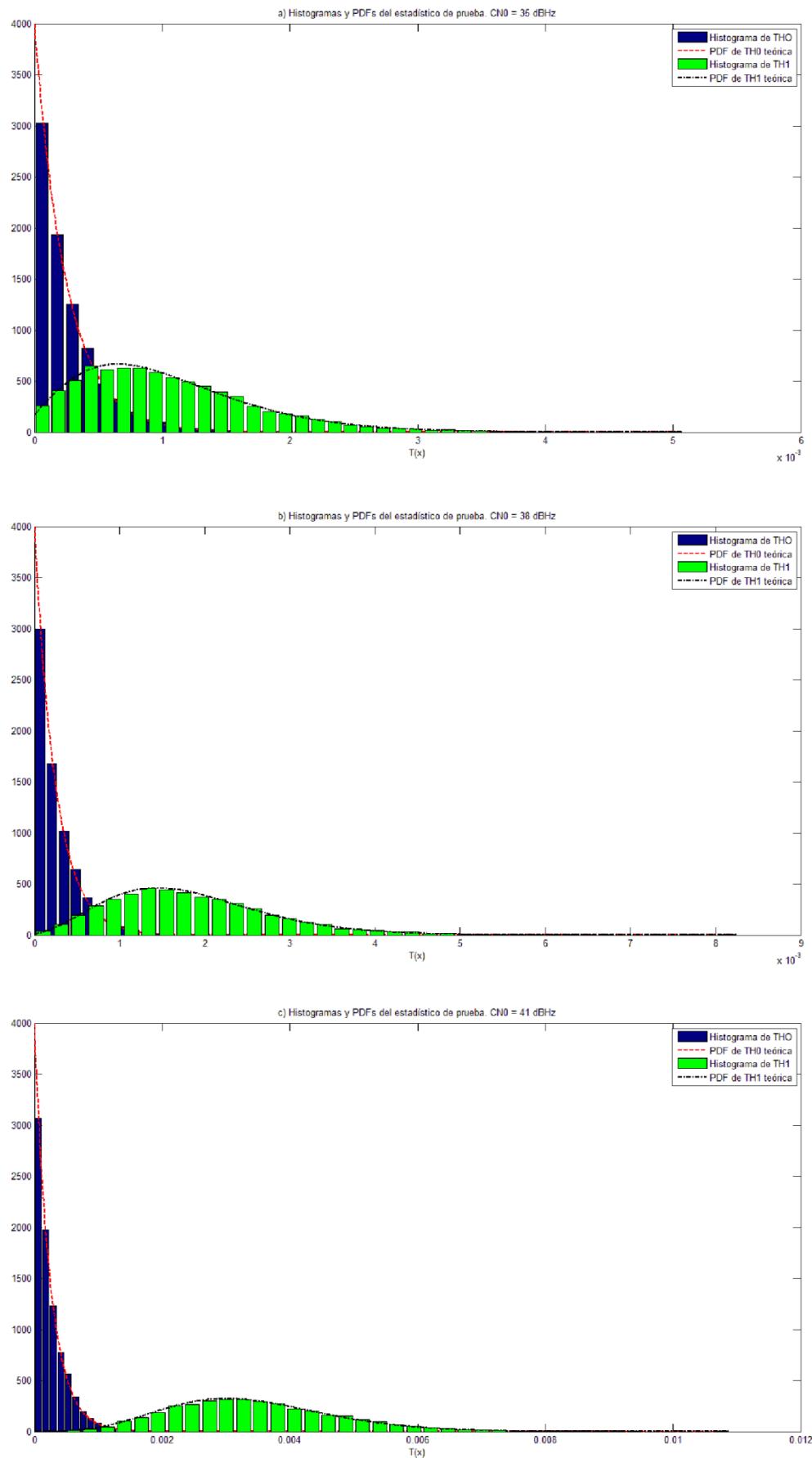
Después de esta primera comprobación de la funcionalidad del estadístico de prueba utilizado se pasa a realizar otras pruebas más detalladas con el fin de validar numéricamente los cálculos realizados para su caracterización estadística.

4.9.2.2 Resultados de la caracterización del estadístico de prueba para una celda

El siguiente paso consiste en validar los modelos probabilísticos desarrollados en el apartado 4.7.3 que permiten caracterizar el estadístico de prueba para una celda del espacio de búsqueda $T(\underline{x})$ desarrollado en (4-40) y cuya estadística está definida por las ecuaciones (4-75), (4-76), (4-77) y (4-78).

Con este objetivo se ha procesado dos veces, con el software GNSS-SDR, cada uno de los seis archivos de señal generados sintéticamente con Matlab.

En la primera ejecución se ha utilizado como señal generada localmente la correspondiente al código del satélite PRN 11 (presente en la señal procesada) y se ha activado la opción `Acquisition0.repeat_satellite=true` del fichero de configuración, que permite repetir la adquisición del mismo código PRN ininterrumpidamente. Además se ha modificado el archivo `pcps_acquisition_cc.cc` de la siguiente manera: en lugar de recorrer todo el espacio de búsqueda se ha desactivado el bucle que recorre todos los bins frecuenciales y se fuerza al algoritmo a realizar la correlación circular solamente en aquel bin donde se espera obtener el máximo del estadístico de prueba (5 kHz, ya que la señal generada sintéticamente tiene una frecuencia Doppler $f_{d_{id}}$ de 4906 Hz y los bins frecuenciales se generan cada 500 Hz) y se guarda en un fichero el valor del estadístico de prueba correspondiente a la posición 3037, correspondiente al retardo de código τ_{id} de la señal analizada, y que es donde se espera obtener el máximo del estadístico de prueba. Mediante la inclusión de un contador, este proceso se ha repetido 10000 veces (procesando 10000 milisegundos consecutivos de señal) y se han guardado los resultados en ficheros separados (uno para cada señal procesada). Se dispone así de 6 ficheros con 10000 realizaciones del estadístico de prueba $T(\underline{x}; H_1)$ para los distintos valores de CN_0 .



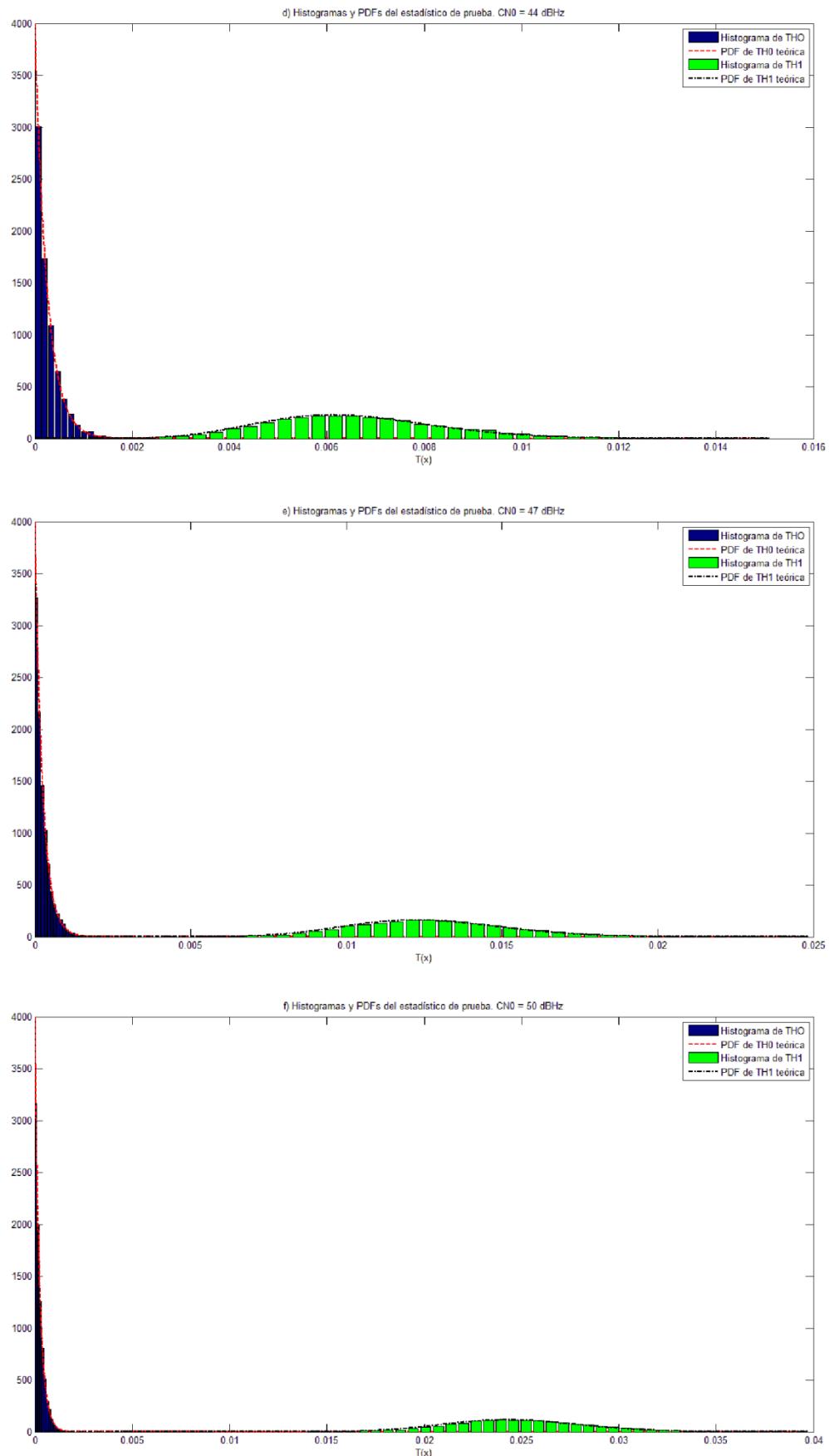


Figura 4-9: Histograma y PDF teórica para los estadísticos de prueba TH_0 y TH_1 para diversos valores de CN_0 : a) 35 dBHz, b) 38 dBHz, c) 41 dBHz, d) 44 dBHz, e) 47 dBHz y f) 50 dBHz.

En la segunda ejecución se ha repetido el mismo experimento para cada archivo de señal pero con la diferencia de que el PRN del satélite de la señal generada localmente en GNSS-SDR es el PRN 1 (satélite ausente en las señales de los ficheros procesados). De esta forma se dispone también de 6 ficheros con 10000 realizaciones del estadístico de prueba $T(\underline{x}; H_0)$ para los mismos valores de CN_0 .

Posteriormente estos ficheros se han procesado en Matlab por parejas (según el valor de CN_0) con el fin de calcular los histogramas de $T(\underline{x}; H_1)$ y $T(\underline{x}; H_0)$ y compararlos con las expresiones teóricas de las ecuaciones (4-75), (4-76), (4-77) y (4-78).

Los resultados pueden observarse en las gráficas de la Figura 4-9, donde se puede apreciar que los valores de las PDFs teóricas coinciden con los histogramas obtenidos en las simulaciones.

En dichas gráficas puede observarse que, mientras que la PDF del estadístico $T(\underline{x}; H_1)$ va cambiando según el valor de la CN_0 (debido a que el parámetro de descentralización δ depende de la SNR, tal y como se vio en (4-73)), la PDF del estadístico $T(\underline{x}; H_0)$ se mantiene prácticamente inalterada para los diferentes valores de CN_0 .

Este hecho parece refrendar la aproximación que se realiza en (4-14), según la cual en la hipótesis H_0 el término de señal es despreciable frente al término de ruido:

$$a_i d_i(f_{d_i}, \tau_i) + \frac{n}{i \neq id} \approx n \quad (4-107)$$

Para acabar de validar este último punto, y dar por buena la aproximación se ha realizado un experimento adicional que consiste en generar (de nuevo desde Matlab) una señal que sólo contenga el término de ruido (sin ningún satélite) y procesarla con GNSS-SDR de la misma manera que las seis señales procesadas anteriormente. Posteriormente se ha vuelto a generar en Matlab el histograma para $T(\underline{x}; H_0)$ con los resultados obtenidos con esta señal y se compara con los seis anteriores (para diferentes valores de CN_0) y con el modelo teórico.

Los resultados se pueden ver en la Figura 4-10 y validan plenamente la aproximación hecha en (4-107) y (4-14), ya que no se aprecian apenas diferencias entre los 7 histogramas, ni de éstos con la gráfica de la PDF teórica del estadístico $T(\underline{x}; H_0)$.

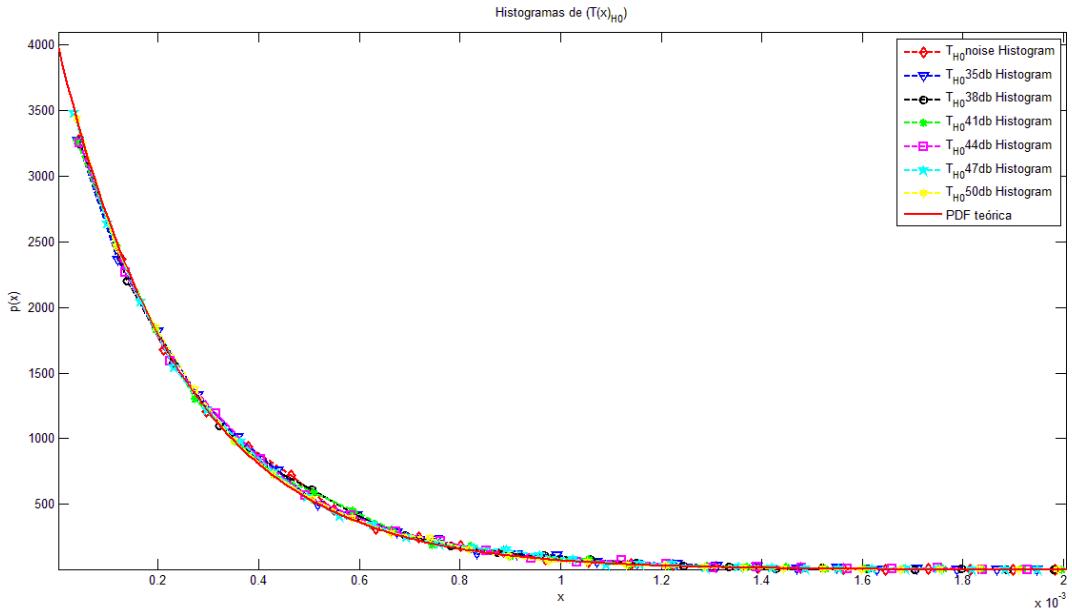


Figura 4-10: Representación de los histogramas reales y de la PDF teórica del estadístico de prueba T_{H0} para diferentes valores de CN_0 .

A continuación se pasa a verificar los resultados de las probabilidades que caracterizan el análisis de todo el espacio de búsqueda.

4.9.2.3 Verificación de las probabilidades de Detección (P_D), Falsa Alarma (P_{FA}) y Detección Fallida (P_{MD}) para todo el espacio de búsqueda

En este último apartado se procede a verificar los cálculos sobre las probabilidades de detección (P_D), de falsa alarma, tanto en ausencia (P_{FA}^a) como en presencia (P_{FA}^p) de la señal buscada, así como la probabilidad de detección fallida (P_{MD}) que se definieron en el apartado 4.7.1 y se formularon matemáticamente en el apartado 4.7.4.

Para verificar dichas probabilidades se han realizado varios experimentos consistentes en volver a procesar dos veces los mismos seis ficheros de señal que en el apartado anterior (que incluyen la señal GPS 1 C/A del satélite PRN 11 más una señal de ruido AWGN con diferentes valores de CN_0) introduciendo ciertos cambios en el fichero *pcps_acquisition.cc*.

La primera vez que se ha procesado cada fichero se han realizado 10.000 adquisiciones consecutivas utilizando el algoritmo *Parallel Code Phase Search Acquisition* explicado en 4.5.3 sobre todo el espacio de búsqueda utilizando como señal local la correspondiente al satélite de PRN 1 (satélite ausente y por tanto correspondiente a la hipótesis H_0), guardando en un fichero el valor del estadístico de prueba $T_{GS}(\underline{x}) = \max_i T_i(\underline{x}) \quad \forall i = 1, \dots, N_{cells}$, definido en (4-41).

El archivo generado se ha analizado posteriormente con Matlab de la siguiente manera: se ha ido variando el valor del umbral γ_{th} a intervalos de 0.001 desde el valor 0 hasta el valor máximo del estadístico de prueba y para cada valor del umbral γ_{th} se ha contabilizado el número de casos en los que $T_{GS}(\underline{x}) > \gamma_{th}$. Posteriormente se ha dividido este valor por el número de realizaciones computando así la probabilidad de falsa alarma en ausencia de satélite (P_{FA}^a) como:

$$P_{FA}^a = \frac{\text{Casos en que } (T_{GS:H_0}(\underline{x}) > \gamma_{th})}{10000} \quad (4-108)$$

La segunda vez se ha vuelto a repetir el mismo procedimiento para cada archivo pero utilizando como señal generada localmente la correspondiente al satélite de PRN 11 (satélite presente y por tanto correspondiente a la hipótesis H_1). Esta vez se ha modificado el archivo *pcps_acquisition.cc* para guardar en fichero el valor del estadístico de prueba $T_{GS}(\underline{x}) = \max T_i(\underline{x}) \quad \forall i = 1, \dots, N_{cells}$, además se han marcado (para poder diferenciarlos posteriormente) aquellos valores del estadístico que se han producido en una celda diferente a la que contiene los parámetros de sincronización correctos (la celda cuyo bin frecuencial corresponde a un valor de 5 kHz y cuyo bin de código se encuentra en la posición 3037).

De esta forma, al analizar el fichero generado desde GNSS-SDR con Matlab, se vuelve a realizar la misma prueba que se utilizó para computar el valor de la probabilidad de falsa alarma en ausencia de satélite (P_{FA}^a): se vuelve a variar el valor del umbral γ_{th} a intervalos de 0.001 desde el valor 0 hasta el valor máximo del estadístico de prueba y para cada valor del umbral se comprueba cuántos estadísticos de prueba están por debajo del umbral y se contabilizan como casos de detección fallida (P_{MD}), luego se contabiliza cuantos estadísticos están por encima del umbral, distinguiendo entre aquellos en que el estadístico se da en la celda correcta, que se contabilizan como casos favorables de la probabilidad de detección (P_D), de aquellos en los que el máximo que supera el umbral no se encuentra en las coordenadas correctas, que se cuentan como casos de la probabilidad de falsa alarma en presencia de satélite (P_{FA}^p). Es decir:

$$\begin{cases} T_{GS:H_1}(\underline{x}) \leq \gamma_{th} & \Rightarrow P_{MD} \\ (T_{GS:H_1}(\underline{x}) > \gamma_{th}) \cap (\hat{f}_d = 5000 \text{ Hz}, \hat{\tau}_i = 3037) & \Rightarrow P_D \\ (T_{GS:H_1}(\underline{x}) > \gamma_{th}) \cap (\hat{f}_d \neq 5000 \text{ Hz}, \hat{\tau}_i \neq 3037) & \Rightarrow P_{FA}^p \end{cases} \quad (4-109)$$

Los resultados de estas pruebas se representan gráficamente en las Figuras 4-11 a 4-27. En la primera serie de figuras (Figura 4-11 a Figura 4-16) se representan los valores de todas las probabilidades calculadas (P_D, P_{FA}^a, P_{FA}^p y P_{MD}) en función del umbral de decisión (γ_{th}) para los 6 valores de CN_0 analizados.

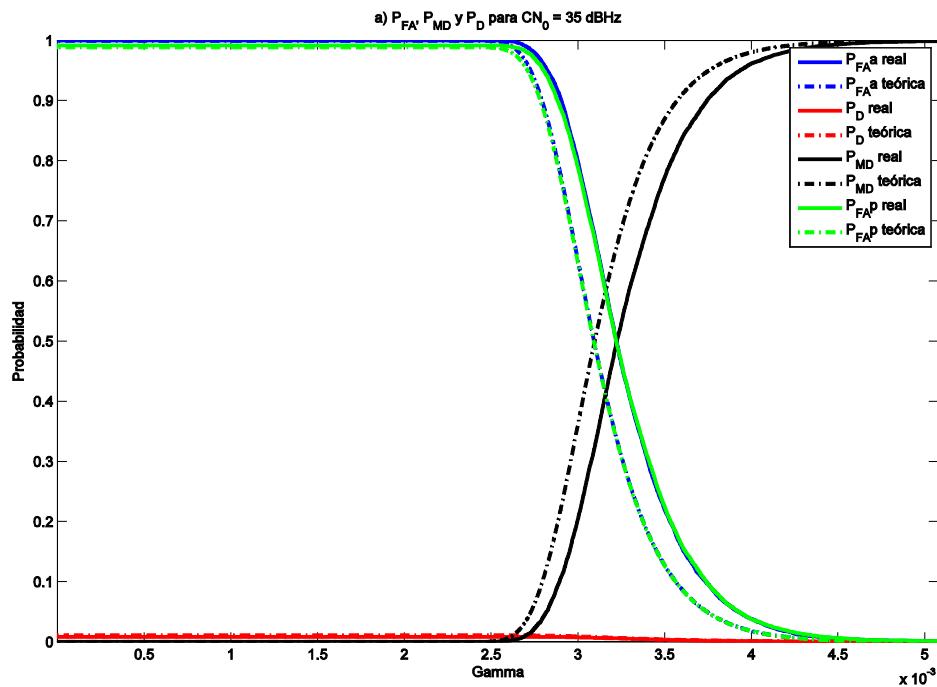


Figura 4-11: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 35 dBHz .

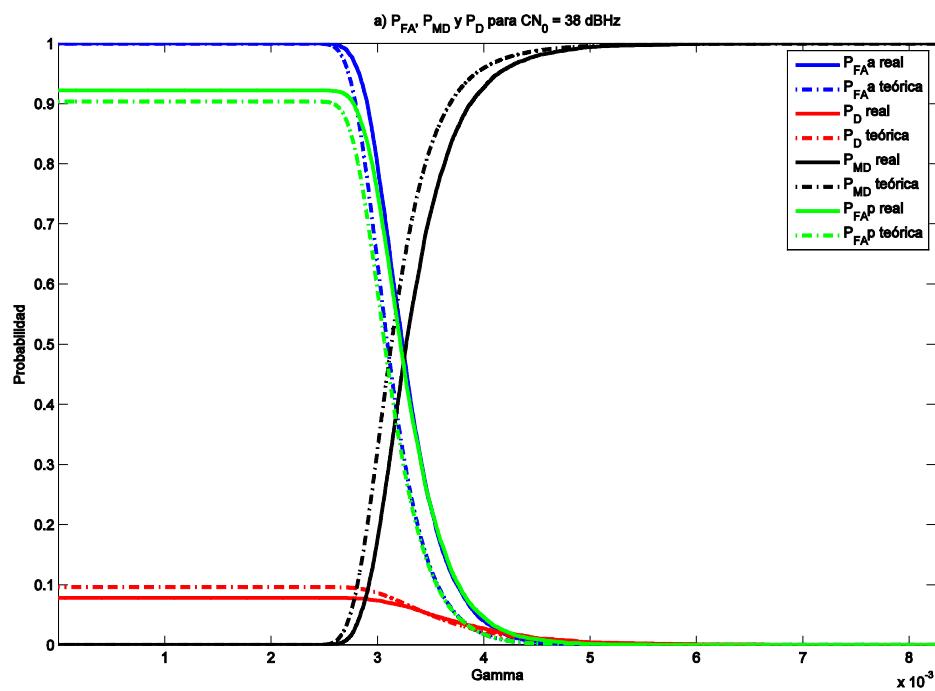


Figura 4-12: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 38 dBHz .

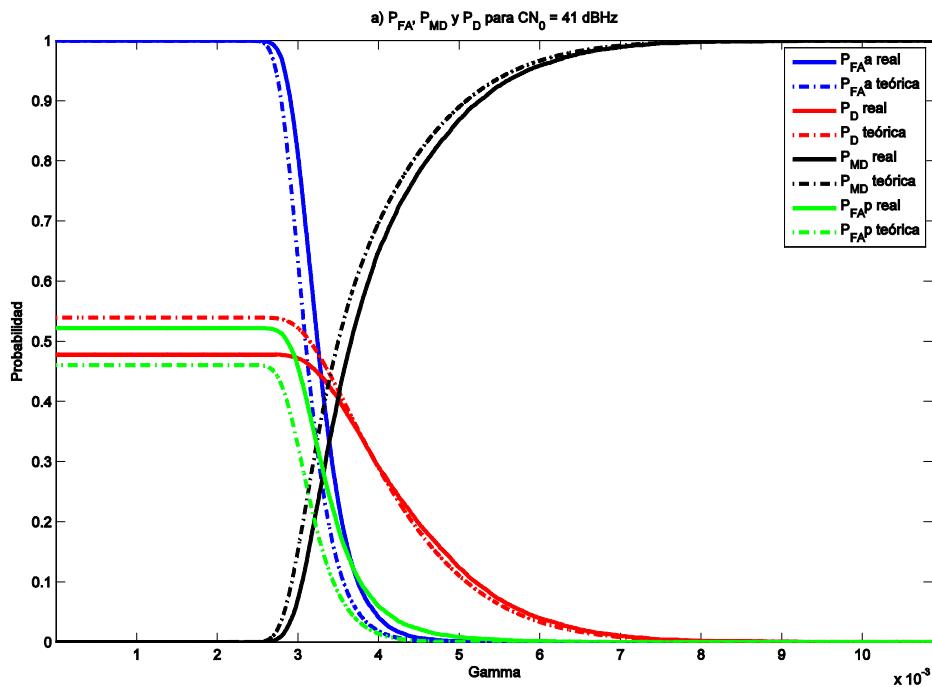


Figura 4-13: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 41 dBHz .

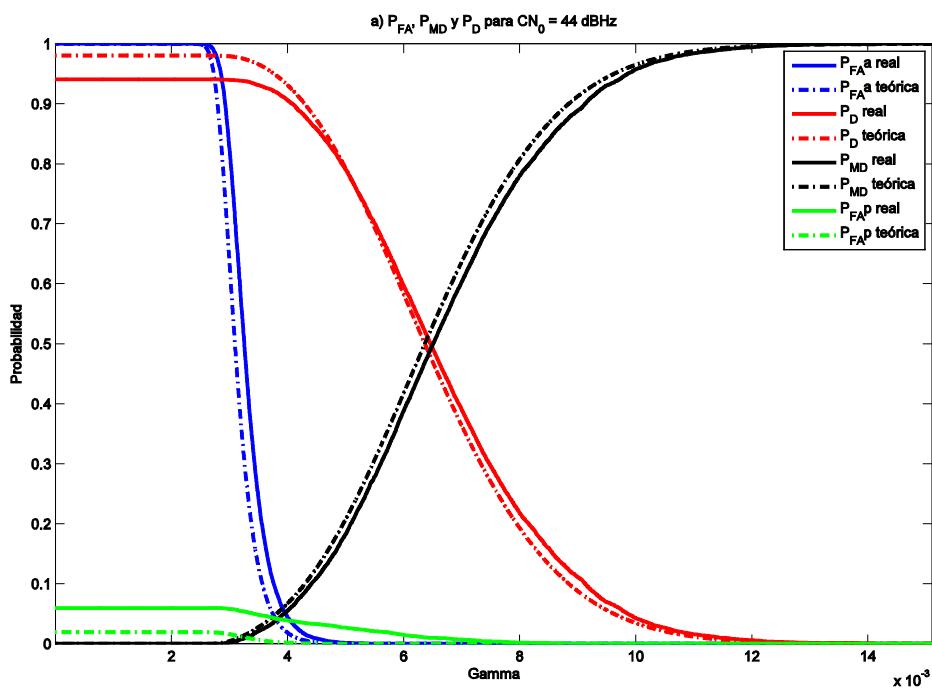


Figura 4-14: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 44 dBHz .

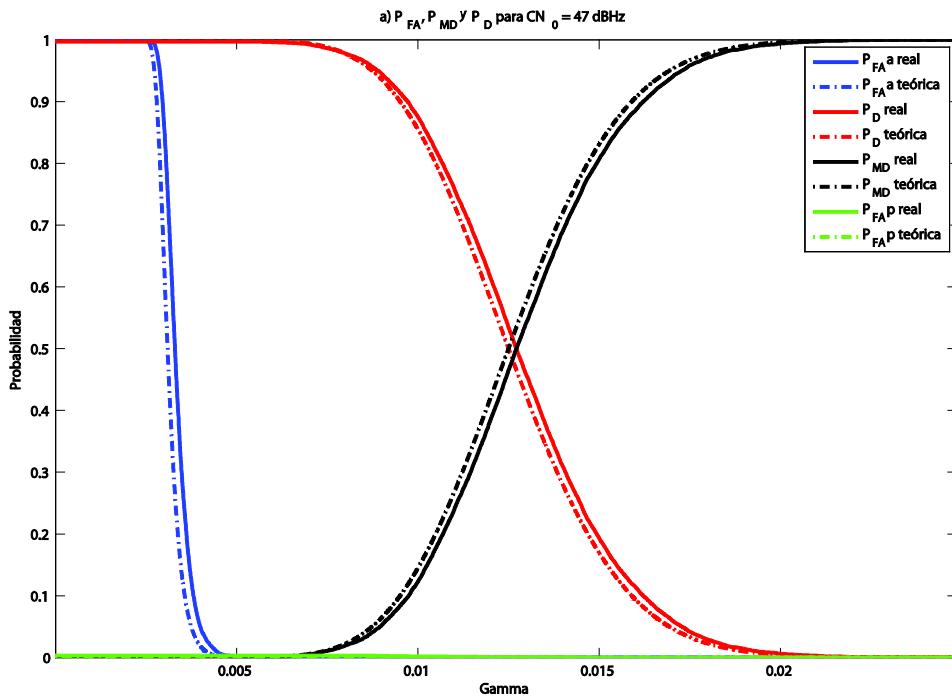


Figura 4-15: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 47 dBHz .

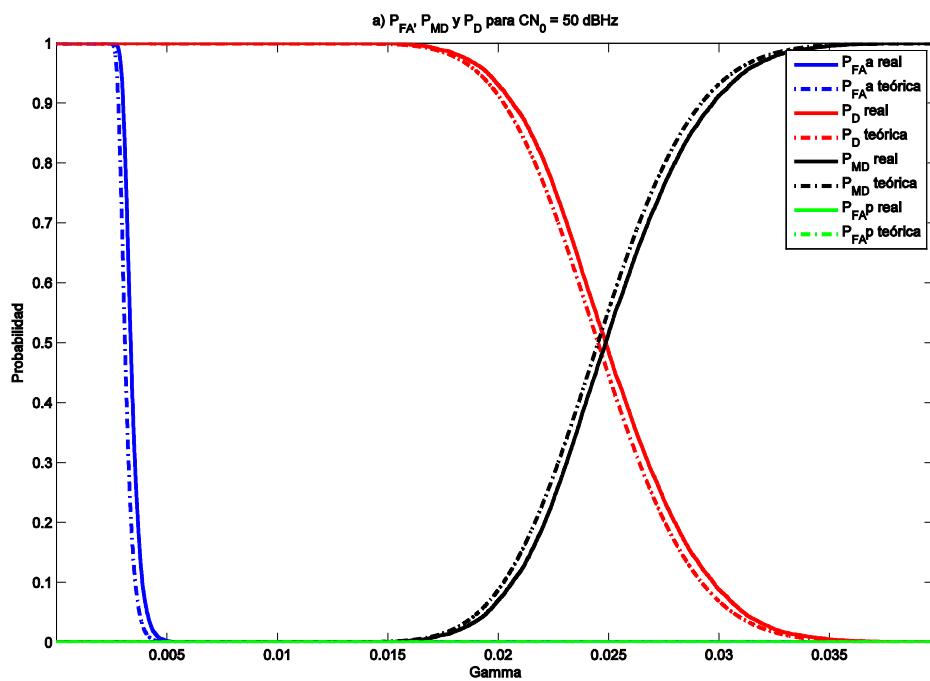


Figura 4-16: Representación de las probabilidades de detección, de falsa alarma (en ausencia y en presencia de la señal buscada) y de la probabilidad de detección fallida para una CN_0 de 50 dBHz .

En las gráficas obtenidas puede verse que los valores obtenidos por simulación se aproximan a los valores teóricos obtenidos en el apartado 4.7.4.

Realizando un análisis más exhaustivo puede observarse que los resultados para la probabilidad de falsa alarma en ausencia de la señal buscada (P_{FA}^a) y la probabilidad de detección fallida (P_{MD}) se ajustan bastante bien a las expresiones teóricas para cualquier valor de CN_0 , con una leve mejora en el caso de valores altos de la CN_0 pero que no puede considerarse muy significativa. En el caso de la probabilidad de falsa alarma en presencia de la señal buscada (P_{FA}^p) y de la probabilidad de detección (P_D) los resultados son igualmente buenos para el caso de valores muy bajos de la CN_0 (35 dBHz, ver Figura 4-11) y para valores altos de la misma (47-50 dBHz, ver Figura 4-15 y Figura 4-16), en el caso de valores no tan extremos puede observarse una desviación entre los valores teóricos y los simulados para valores bajos del umbral de decisión (γ_{th}). Concretamente puede observarse que para valores pequeños del umbral la probabilidad de falsa alarma en presencia de satélite (P_{FA}^p) obtenida mediante simulación es superior a la esperada, mientras que la probabilidad de detección (P_D) es inferior al valor teórico.

Una posible explicación de este fenómeno podría ser que para valores relativamente bajos de CN_0 (38-41 dBHz) y con la resolución de código (4 muestras por chip) utilizada podría haber mas de una muestra que tenga valores cercanos al máximo (debido a que al ser la correlación un pulso triangular no habría decaído demasiado al cabo de 1/4 de chip) que sumados al valor del ruido pueden provocar la aparición de falsos positivos en celdas vecinas a la que realmente contiene los parámetros de sincronización, cosa que justificaría la disminución de la probabilidad de detección (P_D) y el aumento de la probabilidad de falsa alarma en presencia de señal (P_{FA}^p).

En la segunda serie de figuras (Figura 4-17 a Figura 4-21) se representa la probabilidad de falsa alarma en presencia de la señal buscada (P_{FA}^p), en el eje de ordenadas, frente a la probabilidad de falsa alarma en ausencia de señal (P_{FA}^a), en el eje de abscisas. El objetivo de esta representación es verificar la suposición extraída de [4-14], y según la cual $P_{FA}^a \geq P_{FA}^p \quad \forall \gamma_{th}$, ya que si la función representada se encuentra por debajo de la diagonal ($P_{FA}^p = P_{FA}^a$), se verificará esta suposición.

En las gráficas se ha representado tanto los valores teóricos como los valores simulados para los diferentes valores de CN_0 . Se omite la gráfica para una CN_0 de 47 dBHz por su similitud con la de 50 dBHz (como se puede observar comparando los resultados de la Figura 4-15 y la Figura 4-16).

En ellas puede observarse que los cálculos teóricos realizados verifican la suposición (ya que todas las gráficas se encuentran por debajo de la diagonal) y además puede observarse que la probabilidad de falsa alarma en presencia de la señal buscada (P_{FA}^p) va disminuyendo con respecto a la probabilidad de falsa alarma en ausencia de señal (P_{FA}^a) conforme aumenta la el valor de la CN_0 .

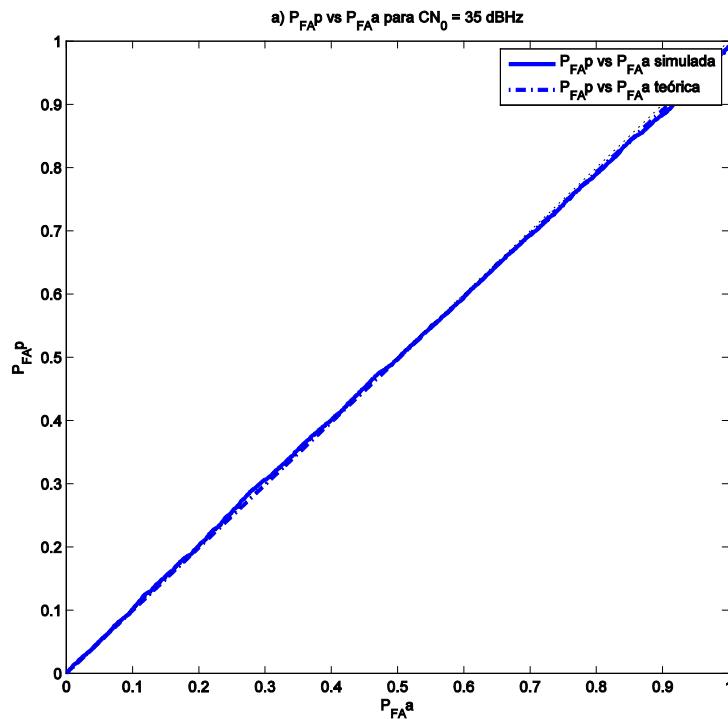


Figura 4-17: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para $CN_0 = 35$ dBHz.

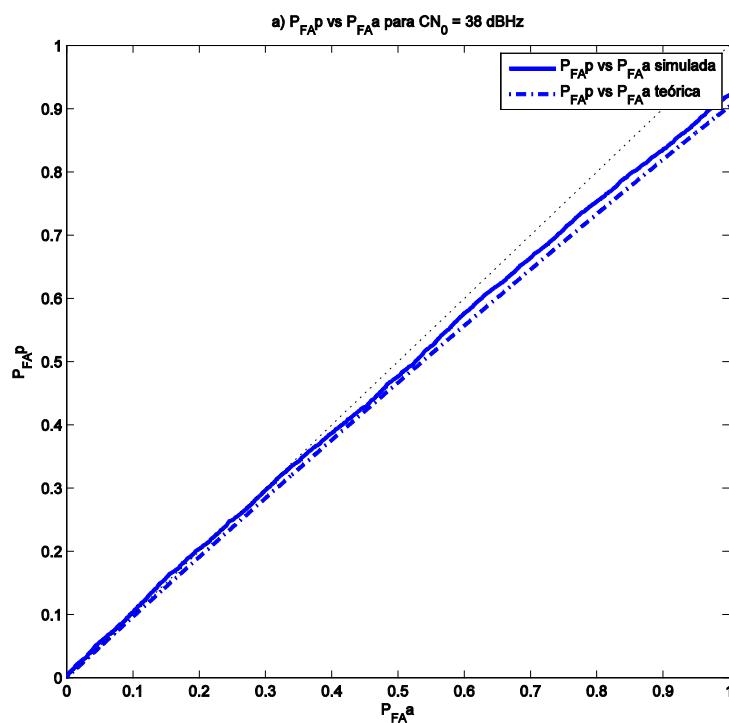


Figura 4-18: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para $CN_0 = 38$ dBHz.

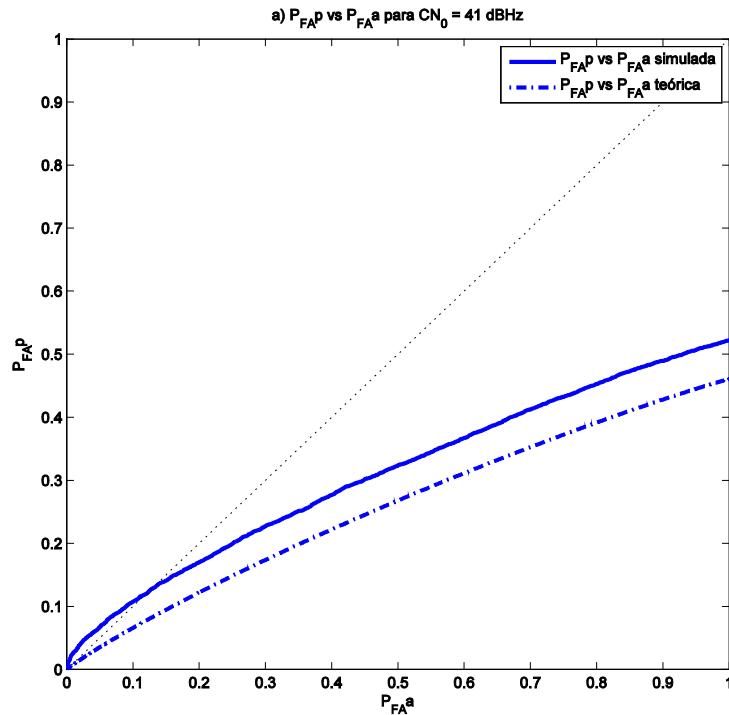


Figura 4-19: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para $CN_0 = 41$ dBHz.

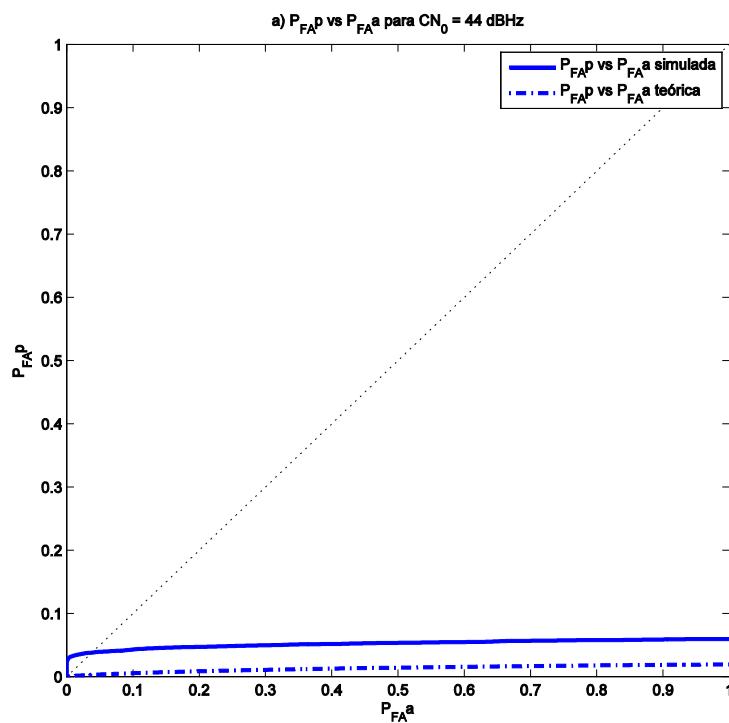


Figura 4-20: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para $CN_0 = 44$ dBHz.

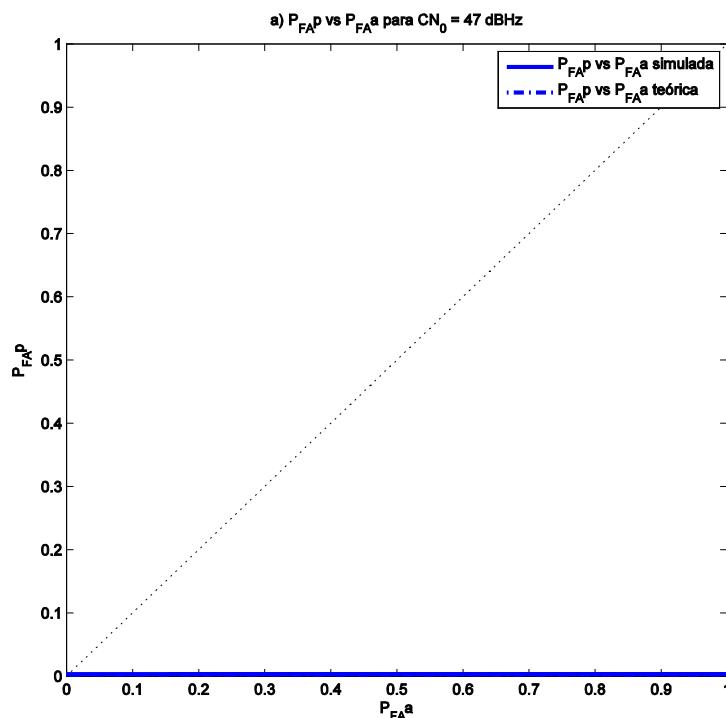


Figura 4-21: Representación de la probabilidad de de falsa alarma en presencia de la señal buscada frente a la probabilidad de de falsa alarma en ausencia de la misma para $CN_0 = 47$ dBHz.

Esto se debe a que al aumentar la CN_0 es más probable que el máximo del estadístico de prueba se dé en la celda que realmente contiene los parámetros de sincronización, con la consecuente disminución de la probabilidad de falsa alarma en presencia de la señal buscada (P_{FA}^p). De esta forma queda plenamente justificado el uso de la probabilidad de falsa alarma en ausencia de señal (P_{FA}^a) como medida para la obtención del umbral de decisión para valores de CN_0 medios y altos.

En cuanto a los valores obtenidos por simulación puede observarse que para valores muy bajos y relativamente altos de la CN_0 los resultados se ajustan a los valores teóricos mientras que para valores de CN_0 medio bajos se observa de nuevo una desviación entre los valores simulados y los teóricos. Se cree que este fenómeno es el mismo que el explicado en las gráficas de la primer serie de figuras (Figura 4-11 a Figura 4-16) y que es debido a que, para los valores de resolución utilizados, a baja CN_0 se producen falsos positivos que provocan el aumento de la probabilidad de falsa alarma en presencia de la señal buscada (P_{FA}^p).

La siguiente serie de figuras (de la Figura 4-22 a la Figura 4-26) presentan las gráficas (para cada valor de CN_0) de la curva ROC (Receiver Operating Characteristic) que es una medida de la calidad del algoritmo de adquisición. Esta curva se genera representando en el eje de ordenadas los valores de la probabilidad de detección (P_D) frente a los valores de la probabilidad de falsa alarma en ausencia de señal (P_{FA}^a) en el eje de abscisas. Como el objetivo de cualquier algoritmo de adquisición es que la probabilidad de detección sea lo más alta posible, mientras que la de falsa alarma sea lo más pequeña posible, es deseable que la curva ROC se sitúe por encima de la diagonal, y que la gráfica tienda a su valor máximo (a poder ser 1) en el eje de ordenadas lo antes posible. Podemos observar que conforme aumenta el valor de la CN_0 las gráficas de la curva ROC se ajustan cada vez más al comportamiento deseado. También puede observarse que la diferencia entre los valores teóricos y los simulados es más significativa para valores de CN_0 medio bajos ya que en la gráfica vuelve a representarse uno de los valores afectados por el fenómeno explicado anteriormente, en este caso la probabilidad de detección (P_D) que, como ya se explicó, para esos valores de CN_0 resulta en unos valores simulados inferiores a los esperados.

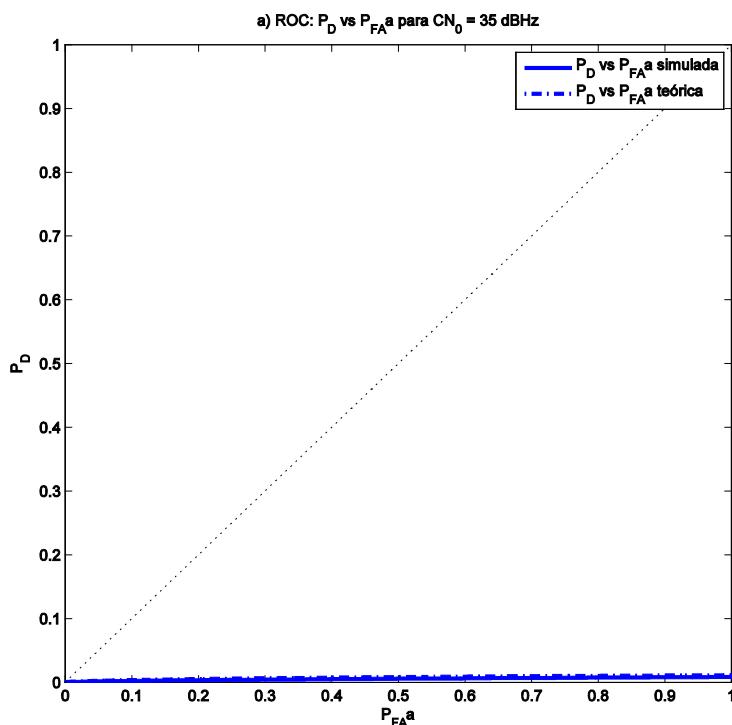


Figura 4-22: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 35 \text{ dBHz}$.

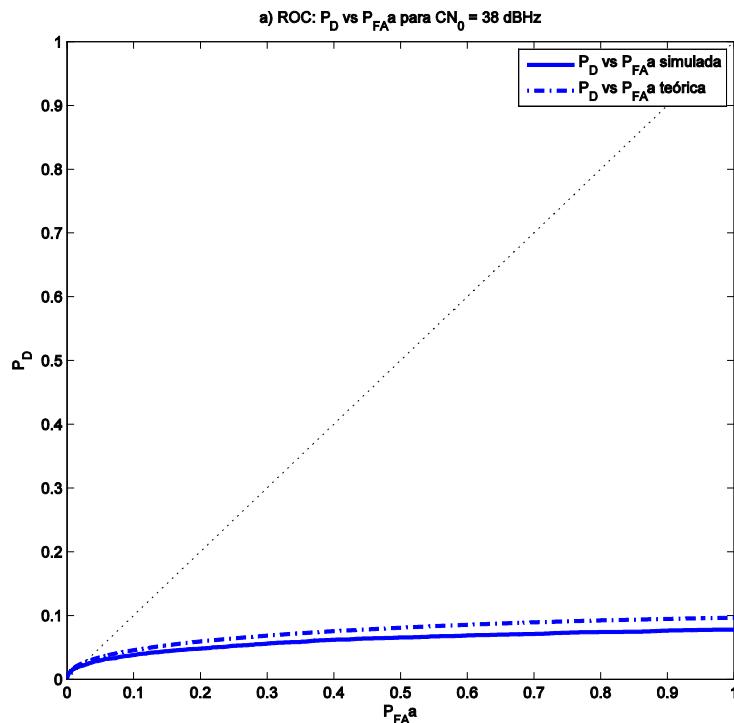


Figura 4-23: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 38 \text{ dBHz}$.

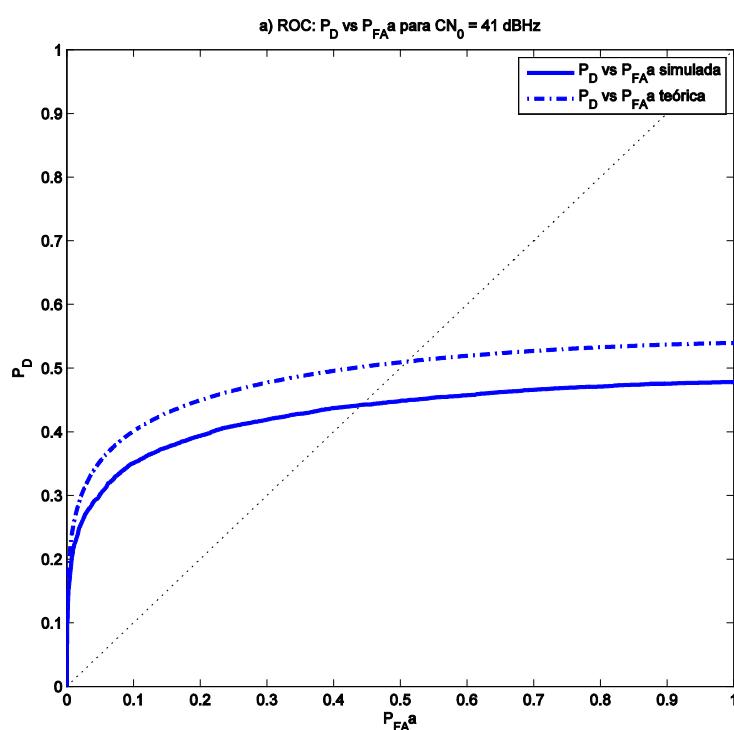


Figura 4-24: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 41 \text{ dBHz}$.

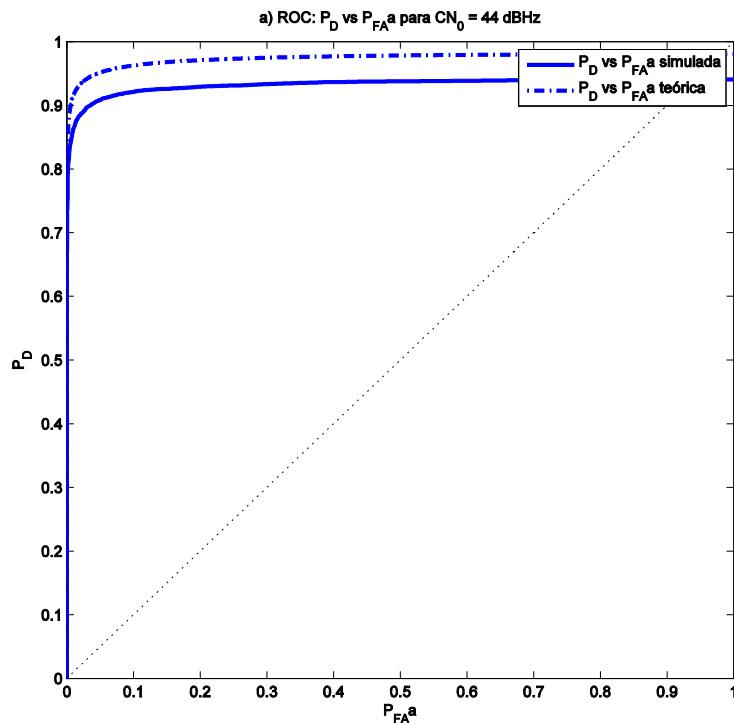


Figura 4-25: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 44 \text{ dBHz}$.

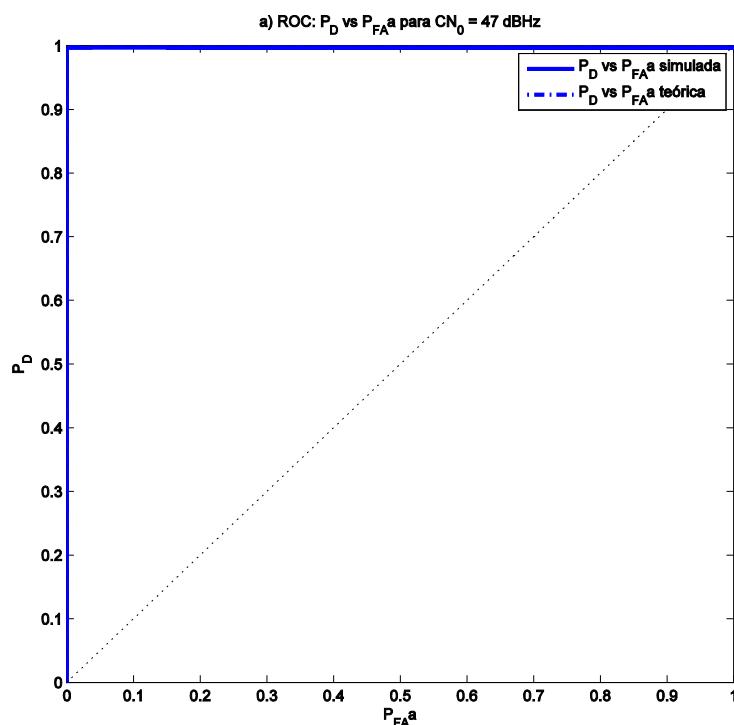


Figura 4-26: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 47 \text{ dBHz}$.

Por último, se va a mostrar el impacto de la frecuencia de muestreo (y por tanto del ancho de banda de la señal) en las prestaciones de la adquisición. Para ello se han generado 3 señales de la misma forma en que se explicó al principio del apartado 4.9.2, es decir, tres señales de 20 segundos de duración consistentes en una señal GPS L1/CA que contiene la señal de un satélite (concretamente el PRN 11) inmersa en ruido AWGN. Todas las señales se han generado con una CN_0 de 44 dBHz, un desplazamiento Doppler $f_{d_{id}}$ de 4906 Hz, un retardo de código τ_{id} de 1052 muestras y una frecuencia intermedia IF de 0 Hz (es decir, una señal compleja en banda base). En este caso las tres señales difieren entre sí en la frecuencia de muestreo, se han generado señales con un ancho de banda de 1 MHz (equivalente a una frecuencia de muestreo de 2 Msps), 2 MHz (4 Msps) y 4 MHz (8 Msps) y se han vuelto a procesar con GNSS-SDR para obtener las probabilidades de detección y falsa alarma.

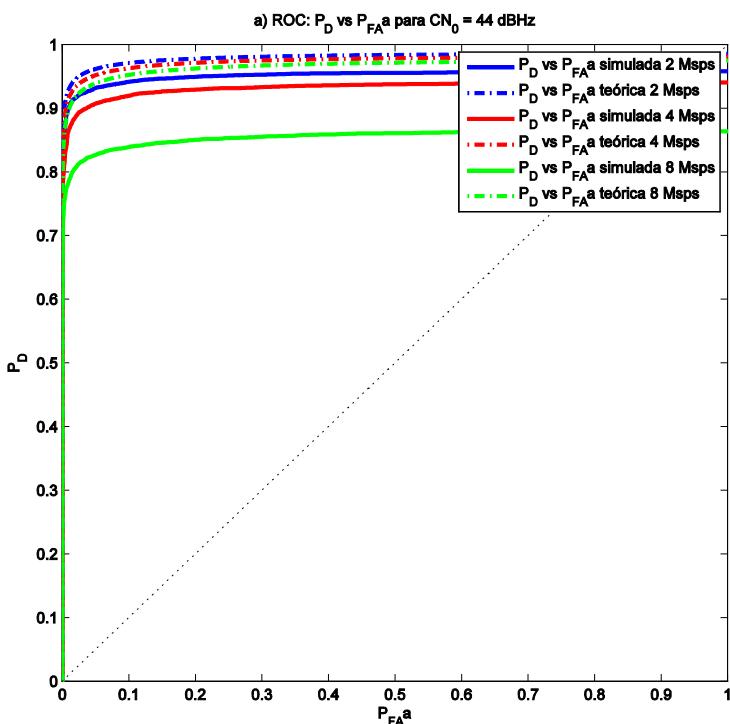


Figura 4-27: Representación de la ROC (probabilidad de detección frente a la probabilidad de falsa alarma en ausencia de la señal buscada) para una $CN_0 = 44$ dBHz y diversas frecuencias de muestreo (2, 4 y 8 Msps).

En la Figura 4-27 se presentan los resultados obtenidos para la ROC de las 3 señales. Se puede observar que tanto los resultados teóricos como los prácticos verifican la suposición de que a mayor número de muestras por chip se obtienen peores resultados para la adquisición, esto es debido a que al aumentar el ancho de banda aumenta en mayor proporción la potencia de ruido que entra al sistema que el nivel de señal. También se puede observar que este empeoramiento es más exagerado en los resultados simulados que en los teóricos. Una posible justificación a este distanciamiento de los resultados prácticos frente a los teóricos conforme aumenta el número de muestras por chip puede deberse a que el modelo de señal elegido funciona correctamente a frecuencias de muestreo bajas, pero a frecuencias de muestreo más altas debería optimizarse ya que, en este caso, para valores de CN_0 bajos es posible que se den falsos positivos en celdas cercanas al máximo debido a la forma de la correlación del código y el modelo no lo contempla. Una posible solución que plantea el autor es estudiar como influye la forma de la correlación del código (un

pulso triangular) conforme se aumenta la frecuencia de muestreo y tenerla en cuenta a la hora de plantear las hipótesis y los estadísticos de prueba que caracterizan a las celdas del espacio de búsqueda.

4.10 Conclusiones y futuros trabajos

En este capítulo se ha presentado uno de los procesos iniciales más importantes en cualquier software GNSS, la adquisición.

Inicialmente se ha hecho una descripción de los diferentes algoritmos de adquisición existentes, haciendo hincapié en el implementado en el software GNSS-SDR, el *Parallel Code Phase Search Acquisition*.

A continuación se han desarrollado los cálculos matemáticos para la obtención del umbral de decisión del algoritmo, poniendo especial atención en la caracterización del estadístico de prueba, inicialmente para una celda del espacio de búsqueda y posteriormente extendiendo los resultados a todo el espacio.

Posteriormente se han presentado los bloques implementados para realizar la adquisición en el software GNSS-SDR, y que incluyen una interfaz común a todos los posibles algoritmos de adquisición (*AcquisitionInterface*), un bloque de GNU Radio que se encarga de realizar todo el procesado de señal del algoritmo de *Parallel Code Phase Search Acquisition* (*pcps_acquisition_cc*) y una clase adaptadora (*GpsL1CaPcpsAcquisition*) que se encarga de conectar las dos clases previas así como de realizar los cálculos propios de la señal utilizada (GPS L1 C/A en este caso).

Finalmente se han presentado los diferentes tipos de pruebas para verificar el comportamiento de los bloques implementados así como contrastar todos los cálculos teóricos realizados en los apartados previos de este capítulo y los resultados obtenidos con estas pruebas. Dichas pruebas se agrupan en dos bloques bien diferenciados, por un lado los test unitarios y de integración, y por otro lado las medidas de calidad del algoritmo que necesitan de herramientas externas para ser analizadas (fundamentalmente Matlab).

Como futuros trabajos se propone desarrollar un modelo de señal más detallado que incluya la forma de la correlación del código con el objetivo de que en situaciones de CN_0 bajas y resoluciones altas el modelo sea más fidedigno y obtener unos valores teóricos más ajustados a los valores que se obtienen por simulación. Otra opción es desarrollar algoritmos de adquisición con períodos de integración mayores que permitan mejorar los resultados obtenidos.

Por otro lado comentar que una posible mejora, consistente en el desarrollo de un algoritmo de adquisición que incluya asistencia por internet está siendo implementada en el momento de la escritura de este proyecto por el Dr. Javier Arribas Lázaro.

Otras mejoras como aumentar la eficiencia del algoritmo de adquisición en cuanto a coste computacional (aprovechando la capacidad de procesado que ofrecen las GPUs de las tarjetas gráficas presentes en la mayoría de ordenadores) se llevarán a cabo en el futuro programa Google Summer of Code 2013 (GSOC 2013 [4-27]), un programa de becas para estudiantes patrocinado por Google en el que GNSS-SDR ha sido incluida como organización mentora.

4.11 Bibliografía

- [4-1] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S. H. Jensen, "A Software-Defined GPS and Galileo Receiver. A Single-Frequency Approach", 1^a edición, Boston: Birkhäuser, noviembre de 2006. Capítulo 6: Acquisiton. pp. 75-86.
- [4-2] C. Fernández-Prades "Advanced Signal Processing Techniques For Global Navigation Satellite Systems Receivers" Tesis doctoral, UPC, Departament de Teoria del Senyal, 2006.
- [4-3] Global Positioning Systems Wing (GPSW), "Navstar GPS space segment. Navigation user interfaces (IS-GPS-200E). Revision IS-GPS- 200E," Tech. Rep., GPS Navstar JPO, junio de 2010.
- [4-4] R. Gold, "Optimal binary sequences for spread spectrum multiplexing." IEEE Transactions on Information Theory, Vol. IT-13(5), pp. 619–621, 1967.
- [4-5] E. D. Kaplan, C. J. Hegarty, "Understanding GPS. Principles and Applications", 2^a edición, Boston/Londres: Artech House, noviembre de 2005. Capítulo 5: Satellite Signal Acquisition, Tracking, and Data Demodulation Acquisiton. Sección 5.8: Signal Acquisition pp. 219-231.
- [4-6] J. B. Tsui, "Fundamentals of Global Positioning System Receivers. A Software Approach.", 2^a edición, New Jersey: John Wiley & Sons, Inc., 2005. Capítulo 7: Acquisition of GPS C/A Code Signals. pp. 129-159.
- [4-7] M. K. Simon, J. K. Omura, R. A. Scholtz, B. K. Levitt, "Spread Spectrum Communications Handbook", rev. ed., New York: McGraw-Hill, 1994, pp. 751–900
- [4-8] L. Scott, A. Jovancevic, S. Ganguly, "Rapid Signal Acquisition Techniques for Civilian and Military User Equipments Using DSP Based FFT Processing," Proc. of 14th International Technical Meeting of The Satellite Division of The Institute of Navigation, Salt Lake City, UT, September 2001, pp. 2418–2427.
- [4-9] J. Arribas, P. Closas , C. Fernández-Prades "Joint acquisition strategy of GNSS satellites for computational cost reduction" , Proceedings of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010), 8-10 December 2010, Noordwijk (The Netherlands).
- [4-10] European Union, "European GNSS (Galileo) Open Service. Signal In Space Interface Control Document. Ref: OS SIS ICD, Issue 1.1", septiembre de 2010.
- [4-11] S. M. Kay, "Fundamentals of statistical signal processing: Detection theory", Prentice Hall, Upper Saddle River, New Jersey, 1998.
- [4-12] S. M. Kay, "Fundamentals of statistical signal processing: Estimation theory", Prentice Hall, Upper Saddle River, New Jersey, 1993.
- [4-13] D.Borio, L.Camoriano, and L.Lo Presti, "Impact of the acquisition searching strategy on the detection and false alarm probabilities in a CDMA receiver," in Position, Location, And Navigation Symposium, PLANS'06, April 2006, pp. 1100–1107.

[4-14] D.Borio, L.Camoriano, and L.Lo Presti, "Impact of GPS Acquisition Strategy on Decision Probabilities", in IEEE Transactions on Aerospace and Electronic Systems, Vol. 44, No. 3, July 2008, pp. 996–1011.

[4-15] "GNU Radio", <http://gnuradio.org>, Comprobado: 18 de mayo de 2013.

[4-16] C. Fernández-Prades, C. Avilés, L. Esteve, J. Arribas, and P. Closas, "Design patterns for GNSS software receivers", in Proc. of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010), ESTEC, Noordwijk, The Netherlands, Dec. 2010.

[4-17] C. Fernández-Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: an open source tool for researchers and developers", in Proc. of the ION GNSS 2011 Conference, Portland, Oregon, Sept. 19-23, 2011.

[4-18] C. Fernández-Prades, J. Arribas, L. Esteve, D. Pubill, P. Closas, "An Open Source Galileo E1 Software Receiver", in Proc. of the 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012), ESTEC, Noordwijk, The Netherlands, Dec. 2012.

[4-19] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Streams Operators. Classes. gr::blocks::stream_to_vector." http://gnuradio.org/doc/doxygen/classgr_1_1blocks_1_1stream_to_vector.html. Comprobado: 3 de junio de 2013.

[4-20] "Boost. Math Toolkit. Statistical Distributions. Exponential Distribution" http://www.boost.org/doc/libs/1_53_0/libs/math/doc/sf_and_dist/html/math_toolkit/dist/dist_ref/dists/exp_dist.html. Comprobado: 18 de mayo de 2013.

[4-21] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Base classes for GR Blocks. gr_block." http://gnuradio.org/doc/doxygen/classgr_block.html. Comprobado: 18 de mayo de 2013.

[4-22] M. Frigo y S. G. Johnson, "The design and implementation of FFTW3," Proceedings of the IEEE, vol. 93, no. 2, pp. 216–231,2005, Special issue on "Program Generation, Optimization, and Platform Adaptation".

[4-23] "VOLK," <http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk>, Comprobado: 22 de mayo de 2013.

[4-24] "Google C++ Testing Framework", <http://code.google.com/p/googletest/>, Comprobado: 22 de mayo de 2013.

[4-25] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Top Block and Hierarchical Block Base Classes. Classes. gr_top_block" http://gnuradio.org/doc/doxygen/classgr_1_1top_block.html. Comprobado: 3 de junio de 2013.

[4-26] "GNU Radio 3.6.5 C++ API. Modules. GNU Radio C++ Signal Processing Blocks. Waveform Generators. Classes. gr_sig_source". http://gnuradio.org/doc/doxygen/classgr_1_1analog_1_1sig_source_c.html. Comprobado: 3 de junio de 2013.

[4-27] "Google Summer of Code 2013. GSOC 2013." <http://www.google-melange.com/gsoc/homepage/google/gsoc2013>. Comprobado: 3 de junio de 2013.

5 GNSS-SDR: CHANNEL

5.1 Resumen.

En este capítulo se explica la arquitectura y funcionamiento del bloque Channel del receptor GNSS-SDR. Inicialmente se explican las funcionalidades principales de un canal para cualquier receptor GNSS, a continuación se pasa a explicar los detalles de la implementación realizada para el software GNSS-SDR, haciendo especial hincapié en la inclusión de una máquina de estados finitos (*Finite State Machine*, FSM) para el control de la interacción de los bloques que forman el canal y finalmente se extraen ciertas conclusiones.

5.2 Arquitectura y funcionalidades del canal de un receptor GNSS

Un canal (*Channel*) encapsula todo el procesado de señal dedicado a un solo satélite. Por lo tanto es un gran objeto compuesto que encapsula los bloques de adquisición, *tracking* y decodificación (que se corresponden respectivamente con los bloques *Acquisition*, *Tracking* y *Telemetry Decoder* de la Figura 5-1).

Al ser un objeto compuesto, puede ser tratado como una entidad única, cosa que propicia que pueda ser replicado con facilidad. Además, dado que el número de canales es seleccionable por el usuario, este enfoque ayuda a mejorar la escalabilidad así como el mantenimiento del receptor.

Tal y como se comentó en el apartado 2.4.2.3, la solución sigue el patrón de diseño Arquitectura de Canal (*Channel Arquitecture pattern*) [5-1], que agrupa todos las funciones de procesado de señal relacionadas con un solo satélite en un subsistema canal. Un canal puede ser considerado como un tubo que transforma los datos de entrada de forma secuencial en datos de salida, cambiando en algunos casos la velocidad de dichos datos. El patrón de Arquitectura de Canal se adapta bien a la transformación secuencial de datos de un estado o forma a otro diferente. Esto simplifica los algoritmos, que se pueden descomponer fácilmente en una serie de pasos que operan sobre elementos aislados de la secuencia de datos. Además, se pueden añadir varias instancias del subsistema canal y así ampliar el número de satélites procesados. De esta forma la arquitectura se puede adaptar fácilmente para manejar múltiples elementos del flujo de datos en paralelo, incluso aunque estén en diferentes etapas de procesado.

Este módulo también está a cargo de la gestión de la interacción entre los bloques de adquisición y *tracking*.

El bloque canal es quien debe controlar el tipo de arranque del bloque de adquisición que, como se comentó en el Capítulo 4 de este proyecto, se puede inicializar de varias maneras dependiendo de la información previa disponible: arranque en frío (*cold start*), cuando el receptor no tiene información sobre su posición ni el almanaque de los satélites, arranque templado (*warm start*) cuando están disponibles una ubicación aproximada y el tiempo aproximado del día, así como una versión del almanaque grabado recientemente en el receptor, o arranque en caliente (*hot start*) cuando el receptor se encontraba en la fase de *tracking* de un satélite y éste se ha perdido de vista durante un corto período de tiempo, pero los datos de las efemérides y el almanaque siguen siendo válidos, o se le proporciona esta información por otros medios.

El proceso de adquisición puede acabar decidiendo que el satélite no está presente o que el satélite sí está presente. En este último caso, el canal debe parar el proceso de adquisición y activar el módulo de *tracking* con las estimaciones aproximadas de los parámetros de sincronización.

Dependiendo del tipo de adquisición y de *tracking* utilizados pueden utilizarse varias estrategias para la interacción entre ambos bloques, desde estrategias sencillas, como la comentada en el párrafo anterior, a estrategias más elaboradas en las que puede decidirse que es necesario un periodo de integración más largo para confirmar la presencia del satélite, utilizando técnicas de adquisición con periodos de integración coherentes y/o no coherentes.

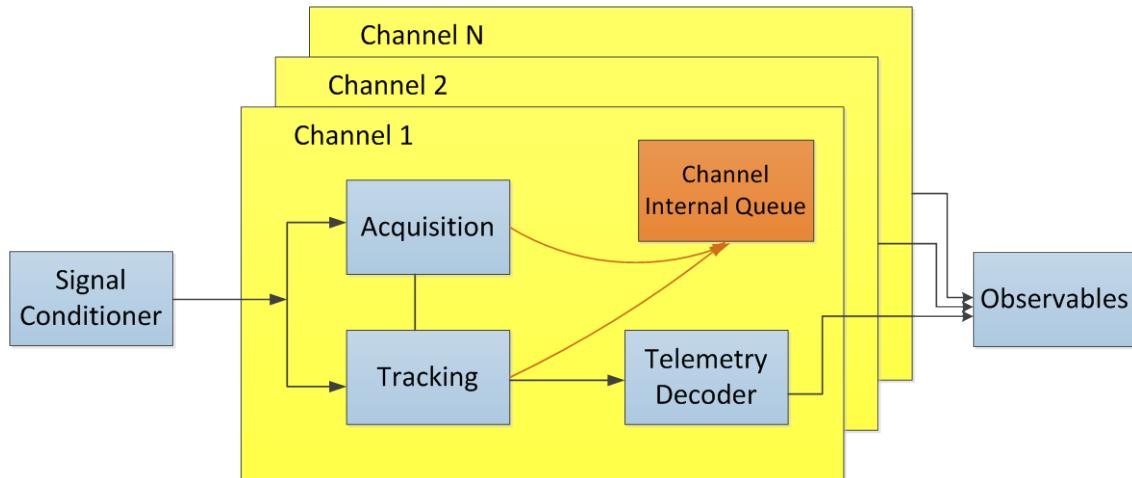


Figura 5-1: Arquitectura del bloque Channel de GNSS-SDR que encapsula a los bloques Acquisition, Tracking y Telemetry Decoder y puede replicarse fácilmente. Además incluye una cola concurrente de mensajes para comunicarse con los bloques de adquisición y tracking

La abstracción matemática utilizada en GNSS-SDR para el diseño de la lógica que debe implementar el canal se conoce como máquina de estados finitos (*Finite State Machine*, FSM), que es un modelo de comportamiento compuesto de un número finito de estados, transiciones entre esos estados y acciones. Este tipo de abstracción debe ser utilizado en la mayoría de receptores GNSS, tal y como se recomienda en [5-2]. Existen en la literatura varios ejemplos exitosos de su utilización como el receptor JUZZLE SW, un receptor multiplataforma escrito en Ansi-C que incluye además la simulación del canal de comunicaciones [5-3] y el receptor N-Gene, un

receptor de software totalmente desarrollado por el Istituto Superiore Mario Boella (ISMB) y el Politecnico di Torino que es capaz de procesar en tiempo real señales de radiodifusión de GPS y Galileo en las bandas L1/E1, así como demodular las correcciones diferenciales transmitidas por el sistema EGNOS. Este receptor es capaz de procesar en tiempo real más de 12 canales, con una frecuencia de muestreo de aproximadamente 17.05 MHz con 8 bits por muestra [5-4].

Se proceda a continuación a explicar la implementación del bloque Channel realizado para GNSS-SDR.

5.3 Implementación de GNSS-SDR Channel.

Como se ha comentado en el apartado anterior, el bloque canal es el encargado de encapsular los bloques que realizan las tareas de adquisición, *tracking* y decodificación, pero realmente no realiza ninguna tarea de procesado de señal por sí mismo. Éste es el motivo de que a la hora de implementarlo tan sólo sea necesario una interfaz (*ChannelInterface*) y un bloque adaptador que encapsule al resto de bloques que realizan el procesado de señal (*Channel*), pero no es necesario implementar ningún bloque GNU Radio. En el diseño realizado se ha implementado también una clase que encapsula la librería utilizada para generar la FSM del canal (*GpsL1CaChannelFsm*).

Las 2 primeras clases han sido modificadas por el autor de este proyecto a partir de unas implementaciones iniciales realizadas por el ingeniero informático Sr. Carlos Avilés, mientras que la última (*GpsL1CaChannelFsm*), ha sido diseñada e implementada en su totalidad por el autor de este proyecto.

En los siguientes sub-apartados se procede a explicar estas tres clases.

5.3.1 ChannelInterface

Como en todos los bloques de GNSS-SDR, el acceso a las funcionalidades del canal debe realizarse mediante una interfaz: *ChannelInterface*, que es una interfaz de una clase abstracta para cualquier bloque canal que se implemente en GNSS-SDR. Dado que todos los métodos son virtuales, esta clase no puede ser instanciada directamente, y sólo se pueden crear instancias directamente de una subclase de ésta si todos los métodos virtuales puros heredados se implementan por esa subclase o una clase padre de la misma.

Los métodos virtuales (todos ellos públicos) que deben ser definidos en las diferentes implementaciones que se realicen del bloque canal son:

- *virtual void start()*: método que permite al *flowgraph* de GNSS-SDR inicializar un canal en concreto después de haberlo conectado al resto de bloques del receptor.
- *virtual void standby()*: método que permite al *flowgraph* de GNSS-SDR activar el modo de espera del canal, en el cual, todos los bloques internos del canal dejan de realizar ningún tipo de operación sobre las muestras que provienen del bloque acondicionador de canal.

- *virtual void stop()*: método que permite al *flowgraph* de GNSS-SDR detener un canal en concreto.
- *virtual void start_acquisition()*: método que permite al *flowgraph* de GNSS-SDR activar la adquisición de un canal en concreto.
- *virtual void set_signal(Gnss_Signal)*: método que permite al *flowgraph* de GNSS-SDR asignar al canal una señal de la lista de señales disponibles (en el momento de la escritura de este proyecto 32 señales GPS L1/CA y 4 señales Galileo E1) con el objetivo de que se realice su adquisición, seguimiento y posterior decodificación en caso de que se determine que el satélite esté presente.
- *virtual Gnss_Signal get_signal() const*: método que permite al *flowgraph* de GNSS-SDR acceder al objeto *Gnss_Signal* de un canal en concreto. Este objeto contiene la información del tipo de señal asignada al canal (GPS L1/Ca, Galileo E1, ...) y del PRN del satélite que se está analizando.

5.3.2 Channel

La clase *Channel* es un adaptador que hereda directamente de *ChannelInterface*. Las tareas principales de esta clase consisten en interconectar los bloques de adquisición, *tracking* y decodificación así como gestionar la interacción entre ellos.

Como se comentó al principio de este capítulo, la lógica que controla la interacción entre estos bloques se realiza a través de una máquina de estados finitos (FSM). Para la implementación de la FSM se ha utilizado la librería Boost.Statechart [5-5], que proporciona las características necesarias tales como soporte para máquinas de estado asíncronas, *multi-threading*, concurrencia, manejo de errores y validación en tiempo de compilación. El diseño de esta máquina de estados se discute con más detalle en el apartado siguiente.

Para la gestión de la comunicación entre los bloques de adquisición y *tracking* que forman el canal se ha optado (tal y como se puede ver en la Figura 5-1) por una cola concurrente de mensajes, disponible en las librerías del receptor GNSS-SDR (*src/core/receiver/concurrent_queue.h*), que fundamentalmente es una clase que implementa una cola de objetos thread-safe de la librería std de C++ (*std::queue* [5-6]) y que, basándose en el código disponible en [5-7], implementa la exclusión mutua (MUTEX) mediante la librería Boost.Thread [5-8].

El funcionamiento de esta cola es el siguiente: cada canal posee un objeto privado de esta clase (denominado *channel_internal_queue_*), y cuando se instancia el canal, en el constructor se asigna los bloques de adquisición y de *tracking* un puntero a esta cola mediante los métodos públicos que proporcionan las interfaces de la adquisición (*acq_->set_channel_queue(&channel_internal_queue_)*) y del *tracking* (*trk_->set_channel_queue(&channel_internal_queue_)*). De esta forma los bloques de adquisición y *tracking* pueden introducir mensajes en la cola, mientras que el canal queda en espera de recibir dichos mensajes para leerlos y actuar en consecuencia.

Existen 3 tipos de mensajes: stop, adquisición positiva y adquisición negativa, que están codificados mediante variables enteras (de valores 0,1 y 2 respectivamente). Se ha elegido este método debido a su escalabilidad, ya que se

puede ir aumentando el número de mensajes con tan sólo ir añadiendo nuevos valores enteros.

Se ha optado por una cola de mensajes es asíncrona debido a que los productores de mensajes (la adquisición y el *tracking*) son bloques de GNU Radio y se ejecutan en hilos concurrentes independientes. A causa de esta elección el canal debe quedarse escuchando la cola de mensajes hasta que se reciba un mensaje. Como este hecho provocaría un bloqueo en la ejecución del receptor, y dado que deben poder ejecutarse varios canales en paralelo, cada uno con su cola interna de mensajes, se ha optado por implementar la ejecución de este proceso dentro de un hilo concurrente aprovechando las clases diseñadas para tal fin de la librería Boost.Thread [5-8].

Además, como la mayoría de bloques del receptor, la clase *Channel* dispone también de un puntero a la cola general de mensajes del receptor (*gr_msg_queue_sptr queue_*) para poder comunicarse con el *flowgraph* de GNSS-SDR.

Se procede a continuación a detallar los objetos y métodos privados y públicos más relevantes de la clase.

Objetos privados (accesibles solamente por la propia clase) más relevantes:

- *AcquisitionInterface *acq_, TrackingInterface *trk_, TelemetryDecoderInterface *nav_*: punteros a las interfaces de la adquisición, del *tracking* y del decodificador, respectivamente. Sus valores son asignados por el *flowgraph* de GNSS-SDR mediante el constructor público de clase.
- *unsigned int channel_*: identificador de canal (dentro del receptor). Como se comentó en el capítulo anterior este identificador es de relevante importancia para la correcta utilización del *logging* del sistema GNSS-SDR explicado en el apartado 2.4.4, ya que como se paraleliza la ejecución de los canales en diferentes hilos concurrentes, se hace necesaria una identificación de los mismos para realizar un seguimiento de su ejecución. Su valor es asignado por el *flowgraph* de GNSS-SDR mediante el constructor público de clase.
- *Gnss_Synchro gnss_synchro_*: objeto de la clase *Gnss_Synchro*, que guarda la información compartida entre los bloques de procesado del canal, tales como los parámetros de sincronización explicados en el capítulo anterior dedicado a la adquisición. Su valor es asignado por el *flowgraph* de GNSS-SDR mediante el método público *void set_signal(Gnss_Signal gnss_signal_)*.
- *Gnss_Signal gnss_signal_*: objeto de la clase *Gnss_Signal*. Su valor es asignado por el *flowgraph* de GNSS-SDR mediante el método público *void set_signal(Gnss_Signal gnss_signal_)*.
- *bool stop_*: variable que controla si el canal está en ejecución (FALSE) o debe detenerse (TRUE). Esta variable se inicializa al valor FALSE en el constructor de clase y puede ser cambiada a TRUE por el *flowgraph* de GNSS-SDR a través del método público *void stop()*.
- *int message_*: variable que contiene el contenido del último mensaje extraído de la cola concurrente de mensajes interna del canal. Su valor se cambia a través del método privado *channel_internal_queue_.wait_and_pop(message_)* de la cola concurrente de mensajes del canal que se ejecuta en el método privado *void run()*.

- *bool repeat_*: variable que controla si la señal que debe procesarse en el bloque de adquisición pertenece siempre al mismo satélite (valor TRUE) o por el contrario debe asignarse el PRN de otro satélite cuando se ha recibida una notificación de adquisición negativa (valor FALSE). Este valor se asigna en el constructor de clase y puede ser fijado por el usuario en el fichero de configuración a través de la opción *AcquisitionID.repeat=true* o *false*, donde ID se corresponde con el identificador de canal.
- *GpsL1CaChannelFsm channel_fsm_*: máquina de estados finitos (FSM) que se encarga de la interacción de los bloques del canal. Se estudia con más detalle en el apartado siguiente.
- *gr_msg_queue_sptr queue_*: puntero a la cola de mensajes general del receptor GNSS-SDR. Su valor es asignado por el *flowgraph* de GNSS-SDR mediante el constructor público de clase.
- *concurrent_queue<int> channel_internal_queue_*: cola concurrente de mensajes interna del canal que le permite comunicarse con los bloques de adquisición y *tracking*.
- *boost::thread ch_thread_*: proceso concurrente de la librería Boost.Thread [5-8], que permite que cada canal se mantenga escuchando la cola interna de mensajes bloqueando su ejecución a la espera de recibir un mensaje de la adquisición o del *tracking* independientemente de lo que hagan el resto de canales.

Los métodos privados de la clase son:

- *void run()*: este método privado es ejecutado dentro del hilo concurrente del canal y consiste en la ejecución de un bucle condicional (que se ejecuta mientras la variable *stop_* tiene el valor FALSE) y que consiste en la ejecución de otros dos métodos:
 1. *channel_internal_queue_.wait_and_pop(message_)*: este método deja al canal a la espera de recibir un mensaje y cuando lo recibe lo extrae de la cola y lo guarda en la variable *message_*.
 2. *process_channel_messages()*: método privado del canal que se explica a continuación.
- *void process_channel_messages()*: método privado que es ejecutado dentro del método *run()* cuando se recibe un mensaje de la cola de mensajes interna del canal y que procesa dicho mensaje que está guardado en la variable *message_*. El método consta de la instrucción *Switch* que en función del valor numérico del mensaje realiza las siguientes tareas:
 - si el mensaje tiene un valor 0 (stop), simplemente realiza una función de *logging* indicando que se procede a detener la ejecución del canal.
 - Si el mensaje tiene un valor de 1 (adquisición positiva) procede a activar el evento correspondiente en la máquina de estados del canal (*channel_fsm_.Event_gps_valid_acquisition()*).
 - Si el mensaje tiene un valor de 2 (adquisición negativa) puede realizar las siguientes tareas:

- Si la variable `repeat_` tiene el valor TRUE procede a activar el evento `channel_fsm_.Event_gps_failed_acquisition_repeat()` de la máquina de estados del canal.
- Si la variable `repeat_` tiene el valor FALSE procede a activar el evento `channel_fsm_.Event_gps_failed_acquisition_no_repeat()` de la máquina de estados del canal.

Entre los métodos públicos propios (no heredados de `ChannelInterface`) cabe destacar el constructor de clase:

- `Channel (ConfigurationInterface *configuration, unsigned int channel, GNSSBlockInterface *pass_through, AcquisitionInterface *acq, TrackingInterface *trk, TelemetryDecoderInterface *nav, std::string role, std::string implementation, gr_msg_queue_sptr queue) : pass_through_(pass_through), acq_(acq), trk_(trk), nav_(nav), role_(role), implementation_(implementation), channel_(channel), queue_(queue)` : constructor público de la clase. En él, además de inicializar la mayoría de variables privadas del canal, se ejecutan una serie de métodos y asignaciones. Entre las más relevantes se encuentran: la asignación del identificador de canal a los bloques de adquisición, *tracking* y decodificación; la asignación de un puntero al objeto de la clase `GnssSynchro` a los bloques de adquisición y *tracking* para que puedan acceder a su contenido y escribir en él los parámetros que obtienen como resultado; la asignación de los parámetros de configuración iniciales del bloque de adquisición; la asignación a los bloques de adquisición y *tracking* de un puntero a la cola concurrente de mensajes para que puedan introducir en ella los mensajes comentados los párrafos anteriores; y la asignación, a la máquina de estados del canal, de punteros a las interfaces de la adquisición, del *tracking* y del decodificador, para que la máquina de estados pueda acceder a los métodos de estas clases, así como un puntero a la cola general de mensajes del receptor GNSS-SDR, para que la máquina de estados pueda comunicarse con el *flowgraph*. Hay que tener en cuenta que cada canal conoce en todo momento (gracias a la cola interna de mensajes) en qué estado se encuentran los bloques de adquisición y *tracking*, pero no tiene información de lo que están haciendo el resto de canales, esa información la tiene el *flowgraph* gracias a la cola general de mensajes del receptor que comparte con todos los canales.

En cuanto a los métodos públicos heredados cabe comentar la implementación de:

- *virtual void start()*: este método se ejecuta desde el *flowgraph* de GNSS-SDR después de que el bloque canal haya sido instanciado y se haya realizado su conexión (tanto la de sus bloques internamente, como la del canal al resto de los bloques del *flowgraph*). El método tan sólo consta de una instrucción, `ch_thread_ = boost::thread(&Channel::run, this)`, que consiste en la ejecución del método privado `run()` explicado anteriormente dentro de un hilo concurrente (`ch_thread_`).
- *virtual void standby()*: método que permite al *flowgraph* de GNSS-SDR activar el modo de espera del canal. La implementación realizada para tal fin consiste en activar el evento `channel_fsm_.Event_gps_failed_tracking_standby()` en la máquina de estados del canal .

- *virtual void stop()*: método que permite al *flowgraph* de GNSS-SDR detener un canal en concreto ya sea porque se ha recibido la instrucción desde la línea de comandos o porque se ha llegado al final del fichero de datos que se está analizando. El método realiza las siguientes funciones:
 - Coloca el mensaje 0 (stop) en la cola interna de mensajes.
 - Asigna el valor TRUE a la variable *stop_* para detener el bucle que mantiene al canal en escucha de la cola de mensajes.
 - *ch_thread_join()*: La ejecución de este método es de vital importancia para la correcta finalización de la ejecución del receptor GNSS-SDR ya que cada canal consta de un hilo concurrente que debe finalizarse correctamente cuando se finaliza la ejecución del software receptor. Cuando un objeto de la clase *boost::thread* que representa un hilo de ejecución se destruye, el hilo de ejecución permanece en ejecución de forma separada hasta que termina la función que está ejecutando. Con el fin de esperar a que finalice el hilo de ejecución deben utilizarse los métodos *join()* o *timed_join()* que son unos métodos propios del objeto de la clase *boost::thread*. Concretamente, el método *join()* utilizado bloquea la ejecución del receptor hasta que el hilo de ejecución representado por el objeto de tipo *boost::thread* haya terminado.
- *virtual void start_acquisition()*: método que permite al *flowgraph* de GNSS-SDR activar la adquisición de un canal en concreto. La implementación se efectúa mediante la activación del evento *channel_fsm_.Event_gps_start_acquisition()* de la máquina de estados del canal.

Se procede por último a explicar la máquina de estados finito (FSM) que controla la lógica que debe implementarse en el bloque canal.

5.3.3 GpsL1CaChannelFsm

Como se ha comentado en los apartados anteriores de este capítulo, toda la lógica que debe llevarse a cabo en la ejecución del bloque canal se implementa a través de una máquina de estados finitos (*Finite State Machine*, FSM). Para la implementación de la FSM se ha utilizado la librería Boost.Statechart [5-5], que proporciona las características necesarias tales como soporte para máquinas de estado asíncronas, *multi-threading*, concurrencia, manejo de errores y validación en tiempo de compilación.

El funcionamiento de la lógica implementada en la FSM es el siguiente:

- se define una serie finita de estados para el canal (cuatro en nuestro caso),
- la transición entre estados sólo puede darse a través de una serie de eventos que pueden lanzarse desde el propio canal o desde el *flowgraph* de GNSS-SDR,
- en la entrada o salida de cada estado se ejecutan una serie de métodos que desarrollan la funcionalidad del canal.

La clase diseñada e implementada para tal efecto recibe el nombre de *GpsL1CaChannelFsm*, y se encuentra en una librería de del software GNSS-SDR, concretamente en los archivos:

src/algorithms/channel/libs/gps_l1_ca_channel_fsm.h y

src/algorithms/channel/libs/gps_l1_ca_channel_fsm.cc

La clase *GpsL1CaChannelFsm* es una clase que hereda públicamente de la clase *boost::statechart::statemachine*, de la librería Boost.Statechart [5-5].

Esta clase posee como objetos privados punteros a las interfaces de los bloques de adquisición, *tracking* y decodificación del canal, así como un puntero a la cola de mensajes general del receptor GNSS-SDR. El valor de estos punteros es asignado en el constructor público de la clase *Channel* explicada en el apartado anterior mediante métodos públicos de la clase *GpsL1CaChannelFsm*: *void set_acquisition(AcquisitionInterface *acquisition)*, *void set_tracking(TrackingInterface *tracking)* y *void set_queue(gr_msg_queue_sptr queue)*.

La clase además posee otros métodos públicos como el constructor y los eventos (que pueden ser activados desde el *flowgraph* o el propio canal).

Por último, esta clase posee 3 métodos privados que se encargan de la ejecución de las funcionalidades del canal:

- *void start_acquisition()*: este método ejecuta el método público *reset()* del bloque de adquisición a través del puntero a la interfaz *AcquisitionInterface* y su finalidad es activar el proceso de adquisición.
- *void start_tracking()*: este método consiste en la ejecución del método público del mismo nombre del bloque de *tracking* a través del puntero a la interfaz *TrackingInterface*. Además, genera un mensaje en la cola de mensajes general de GNSS-SDR indicando que se ha producido una adquisición positiva. Dado que el proceso de adquisición es el que más recursos consume, para evitar que se cree un cuello de botella en la ejecución de GNSS-SDR, se da la opción al usuario de seleccionar cuantos canales pueden estar realizando la función de adquisición en paralelo (mediante la asignación de un valor numérico a la opción *Channels.in_acquisition=* del fichero de configuración). De esta forma, el *flowgraph* debe llevar la cuenta de qué canales se encuentran en adquisición al mismo tiempo y controlar cuando un canal ha finalizado dicho proceso para poder activar el proceso de adquisición de otro canal que se encuentre a la espera. El objetivo de este mensaje es, por tanto, avisar al *flowgraph* para que libere una plaza de la lista de canales en adquisición y actúe en consecuencia, asignando esa plaza a otro canal.
- *void request_satellite()*: este método consiste en generar un mensaje en la cola de mensajes general de GNSS-SDR indicando que se ha producido una adquisición negativa. De esta forma el *flowgraph* sabe que el canal ya ha acabado la adquisición y puede asignarle otro código PRN para volver a iniciar el proceso de adquisición.

En el bloque *Channel* explicado en el apartado anterior de este capítulo se instancia un objeto privado de esta clase denominado *channel_fsm_* que es el que permite controlar la máquina de estados del canal.

Se pasa a continuación a explicar en detalle los cuatro estados diseñados: cómo puede llegarse a ellos, qué métodos se ejecutan al entrar o salir del estado, cuáles son los eventos que provocan las transiciones de estado y quién los activa, así como a qué estados se va a parar en cada transición. Toda esta información queda refleja en la Figura 5-2, en la que se representa un diagrama de estados de la FSM diseñada para GNSS-SDR.

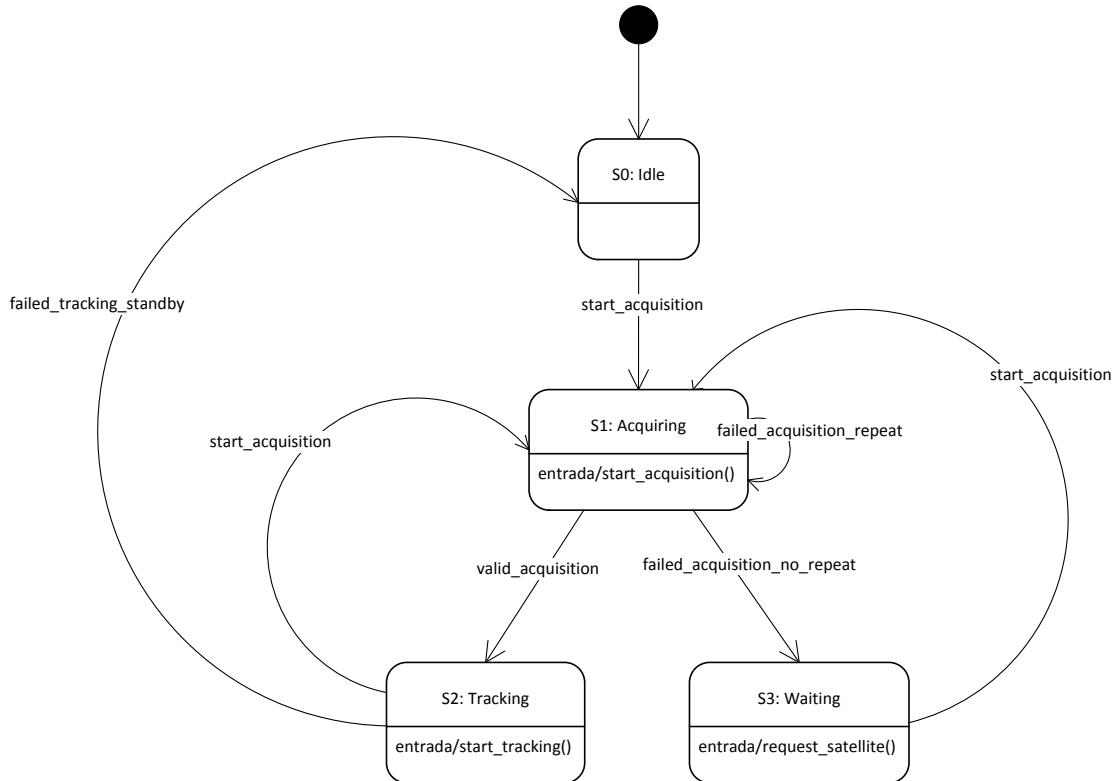


Figura 5-2: Diagrama de estados de la máquina de estados finitos implementada en el canal.

La descripción de estados que posee la FSM es la siguiente:

- **S0 (channel_idle)**: estado inicial de reposo en el que se encuentra la máquina de estados del objeto `channel_fsm_` cuando se instancia el bloque de la clase `Channel`. También puede accederse a este estado desde el estado **S2 (cannel_tracking)** cuando se activa el evento `Ev_gps_channel_failed_tracking_standby`, este evento se activa desde el `flowgraph` de GNSS-SDR a través del método público `Channel::standby()` en el momento en que el `tracking` de un canal ha perdido el seguimiento de los parámetros de sincronización pero no puede volver al estado de adquisición debido a que el número de canales simultáneos en adquisición ha llegado a su límite.

No tiene ningún método de ejecución ni en la entrada ni en la salida del estado.

Si se recibe un evento `Ev_gps_channel_start_acquisition`, se pasa al estado **S1 (channel_acquiring)**. Este evento se activa desde el `flowgraph` a través de la

función pública *Channel::start_acquisition()* después de que el canal haya sido conectado y arrancado mediante el método público *Channel::start()* o bien cuando se libera una plaza en el número de canales que pueden realizar la adquisición simultáneamente.

- *S1 (channel_acquiring)*: estado de adquisición del canal. Se puede acceder a él cuando se activa el evento *Ev_gps_channel_start_acquisition* desde el *flowgraph* a través de la función pública *Channel::start_acquisition()* en tres situaciones diferentes:

1. Desde el estado de reposo *S0 (channel_idle)* cuando el *flowgraph* recibe la notificación de que una plaza para realizar la adquisición ha quedado libre.
2. Desde el estado de *tracking S2 (channel_tracking)* cuando el *flowgraph* recibe la notificación de que el *tracking* se ha perdido y hay una plaza libre en lista de canales para realizar una nueva adquisición.
3. Desde el estado de espera *S3 (channel_waiting)* cuando el *flowgraph* recibe un mensaje de adquisición negativa por parte del canal y le asigna un nuevo PRN a adquirir.

Además, existe una cuarta situación de entrada a este estado que se produce cuando estando en el mismo estado de adquisición se activa el evento *Ev_gps_channel_failed_acquisition_repeat*. Este evento se activa desde el propio canal , concretamente a través del método privado del objeto de la clase *Channel::process_channel_messages()* cuando se recibe un mensaje de adquisición negativa en la cola de mensajes concurrente del canal y la variable privada *repeat_* del objeto de la clase *Channel* tiene asignado el valor TRUE. En este caso no es necesario que el canal se comunique con el *flowgraph* ya que va a volver a realizar la adquisición de la señal del mismo satélite que acaba de analizar y no necesita que el *flowgraph* le asigne un nuevo PRN. Además, la plaza para realizar la adquisición que necesita ya la estaba ocupando antes de la activación del evento y nunca la ha llegado a liberar por lo que no debe esperar a que el *flowgraph* le asigne una plaza nueva.

Este estado tiene un método que se ejecuta cuando se entra en él, concretamente el método *GpsL1CaChannelFsm::start_acquisition()* que, como se ha explicado anteriormente, consiste en la ejecución del método público *reset()* de la interfaz de la adquisición del canal, cuya función es activar el bloque de adquisición.

Como eventos de salida del estado se pueden dar:

1. *Ev_gps_channel_failed_acquisition_no_repeat*, que es un evento de transición hacia el estado de espera *S3 (channel_waiting)* y que se activa desde el propio canal , a través del método privado *Channel::process_channel_messages()* cuando se recibe un mensaje de adquisición negativa en la cola de mensajes concurrente del canal y la variable privada *repeat_* contiene el valor FALSE.
2. *Ev_gps_channel_failed_acquisition_repeat*. que lleva de nuevo al mismo estado de adquisición como se ha explicado en esta misma página.

3. *Ev_gps_channel_valid_acquisition*, que es un evento de transición hacia el estado de *tracking S2 (channel_tracking)* y que se activa desde el método privado del canal *Channel::process_channel_messages()* cuando se recibe un mensaje de adquisición positiva en la cola de mensajes concurrente del canal.
- *S2 (channel_tracking)*: estado de *tracking* del canal. Como se comenta en el párrafo anterior, se accede a él desde el estado de adquisición cuando se activa desde el propio canal el evento *Ev_gps_channel_valid_acquisition* al recibir un mensaje de adquisición positiva.

Este estado ejecuta a su entrada el método *GpsL1CaChannelFsm::start_tracking()*, que activa el bloque de *tracking* e informa al *flowgraph* de su estado.

Como eventos de salida del estado se pueden dar:

1. *Ev_gps_channel_failed_tracking_standby*, que es un evento de transición hacia el estado de reposo *S0 (channel_idle)* y que se activa desde el *flowgraph* a través del método público del canal *Channel::standby()* cuando recibe un mensaje de fallo en el proceso de *tracking* en la cola de mensajes general del receptor GNSS-SDR y no hay plaza disponible en la lista de canales para iniciar una adquisición, por lo que el canal debe desactivarse hasta que quede una plaza libre y pueda volver a iniciar el proceso de adquisición.
 2. *Ev_gps_channel_start_acquisition*. que es un evento de transición hacia el estado de adquisición *S1 (channel_acquiring)* y que se activa desde el *flowgraph*, a través del método público del canal *Channel::start_acquisition()*, cuando recibe un mensaje fallo en el proceso de *tracking* en la cola de mensajes general del receptor GNSS-SDR y hay plaza disponible en la lista de canales para iniciar una adquisición. En este caso no se asigna un nuevo PRN, ya que es probable que el satélite que estaba siguiendo el *tracking* se haya perdido de vista momentáneamente, por lo que se vuelve a ejecutar la adquisición sobre mismo satélite que acaba de perderse, es lo que se denomina proceso de readquisición.
- *S3 (channel_waiting)*: estado de espera del canal. Se accede a él desde el estado de adquisición cuando se activa desde el propio canal el evento *Ev_gps_channel_failed_acquisition_no_repeat* a través del método privado *Channel::process_channel_messages()*, esto se produce cuando se recibe un mensaje de adquisición negativa en la cola de mensajes concurrente del canal y la variable privada *repeat_* contiene el valor FALSE.

Al entrar en este estado se informa al *flowgraph* (mediante un mensaje en la cola general del receptor) de que el canal está a la espera de que se le asigne un nuevo satélite para empezar a adquirir ya que tiene plaza asignada para poder hacerlo (dado que no la ha llegado a liberar). Ésta es la principal diferencia con el estado de reposo *S0 (channel_idle)*.

Para salir de este estado se debe producir el evento *Ev_gps_channel_start_acquisition* que es activado por el *flowgraph* en cuanto extrae un nuevo PRN de la lista de señales para procesar y se lo asigna al canal, que pasa de nuevo al estado de adquisición *S1 (channel_acquiring)*.

Después de haber detallado el diseño de la FSM cabe remarcar que su funcionamiento puede verse en la ejecución del software receptor, ya que se escriben por pantalla la evolución de los estados de todos los canales instanciados.

En el capítulo siguiente se presentan experimentos con señal real donde se corrobora el buen funcionamiento del bloque diseñado y su integración total dentro del sistema GNSS-SDR.

5.4 Conclusiones y futuros trabajos

Se ha explicado en este capítulo el funcionamiento y la arquitectura del bloque canal implementado en el receptor GNSS-SDR, en particular se han explicado por separado su interfaz *ChannelInterface*, su adaptador *Channel*, y especialmente la máquina de estados finitos (FSM) *GpsL1CaChannelFsm*, que implementa toda la lógica para controlar la funcionalidad del canal y su interacción con los bloques internos de adquisición y *tracking*, así como con el *flowgraph* general de GNSS-SDR.

Como futuros trabajos se proponen implementaciones de canal más complejas que podrían desarrollar estrategias más sofisticadas, como las de los *vector tracking loops*, que implican a señales de más de un satélite, así como optimizaciones realizadas directamente en el dominio de la posición [5-9].

5.5 Bibliografía

[5-1] B. P. Douglass, "Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems", Addison Wesley, Upper Saddle River, NJ, 2002.

[5-2] F. Principe, G. Bacci, F. Giannetti, M. Luise: "Software-defined radio technologies for GNSS receivers: A tutorial approach to a simple design and implementation", Int. J. Navigation and Observation, vol. 2011, Article ID 979815, 27 pages, 2011.

[5-3] G. Artaud, G. Menard, L. Ries, J. Dantepal, J. L. Issler, "JUZZLE SOFTWARE RECEIVER", NAVITEC '2006, 11 - 14 December 2006 at ESTEC, Noordwijk, the Netherlands.

[5-4] M. Fantino, A. Molino, and M. Nicola, "N-Gene GNSS receiver: Benefits of software radio in navigation," in Proc. of the European Navigation Conference - Global Navigation Satellite Systems (ENCGNSS), Napoles, Italy, May 2009.

[5-5] "The Boost Statechart Library," www.boost.org/doc/libs/release/libs/statechart/doc/index.html, Comprobado: 19 de Junio de 2013.

[5-6] "std::queue", <http://www.cplusplus.com/reference/queue/queue/>. Comprobado: 19 de junio de 2013.

[5-7] "Implementing a Thread-Safe Queue using Condition Variables", <http://www.justsoftwaresolutions.co.uk/threading/implementing-a-thread-safe-queue-using-condition-variables.html>. Comprobado : 19 de Junio de 2013.

[5-8] “The Boost Thread Library,” www.boost.org/doc/libs/release/libs/thread/doc/index.html, Comprobado: 19 de Junio de 2013.

[5-9] P. Closas and C. Fernández-Prades, “Bayesian nonlinear filters for Direct Position Estimation,” in Proc. of IEEE Aerospace Conference, Big Sky, MT, March 2010, pp. 1–12, DOI: 10.1109/AERO.2010.5446676.

6 EXPERIMENTOS CON SEÑAL REAL

6.1 Resumen

El objetivo de este capítulo es realizar experimentos con señales GPS L1/CA reales para probar la integración de todos los bloques diseñados por el autor y que han sido presentados en los capítulos anteriores de este proyecto y comprobar su funcionalidad dentro del receptor GNSS-SDR para conseguir posicionamiento en tiempo real. Para más información sobre los bloques de *tracking*, decodificación del mensaje de navegación y cálculo de la posición necesarios para mostrar los resultados de este capítulo, el lector es referido a la página del proyecto, <http://gnss.sdr.org>, así como a las publicaciones referidas en las conclusiones de esta memoria en el Capítulo 8.

Con este fin en los dos apartados siguientes se presentan sendos experimentos con diferentes configuraciones en cuanto a hardware y software se refiere. Por último se presentan las conclusiones de ambos experimentos.

6.2 Experimento 1: Posicionamiento en tiempo real con la USRP y un ordenador de gama baja.

El primer experimento consiste en probar la funcionalidad del bloque *direct_resampler_conditioner* realizado por el autor y explicado en el Capítulo 3 de este proyecto.

El objetivo de este bloque es reducir la frecuencia de muestreo de la señal de entrada al software GNSS-SDR de forma arbitraria para poder ser ejecutado en ordenadores personales antiguos o de gama baja y que no disponen de la misma capacidad de procesado que los ordenadores más actuales.

6.2.1 Hardware utilizado

El hardware que forma el cabezal de radiofrecuencia utilizado en este experimento puede verse en la Figura 6-1 y se constituye de una antena activa (Novatel GPS-600), que tiene una banda de paso a 3 dB en la banda L1 ($1575 \pm 8 \text{ MHz}$) e incorpora un amplificador de bajo nivel de ruido que proporciona 28 ± 3 dB de ganancia. La antena se conecta a la placa DBSRX [6-1], que es la que se encarga de la amplificación, bajada en frecuencia (*downconversion*) y filtrado de la señal. La placa DBSRX está conectada a la placa madre de la USRP v1 rev 4.5 [6-2],

que realiza la conversión analógico digital, así como otra bajada en frecuencia mucho más fina y un diezmado de la señal. Además incorpora la interfaz USB 2.0 para transmitir el flujo de datos al ordenador anfitrión.

La placa DBSRX es un sistema receptor completo para señales en la banda de frecuencias que va de 800 MHz a 2.3 GHz con una figura de ruido de 3-5 dB. La DBSRX incluye un filtro de canal controlable por software cuya banda de paso puede ir desde 1 MHz hasta 60 MHz. Después de la entrada de la antena SMA hay un amplificador de 17 dB, seguido por un amplificador programable (0-56 dB), cuya salida se conecta a dos sintonizadores (I&Q) de conversión directa (Maxim MAX2118, que abarcan desde 925 hasta 2175 MHz), seguidos por otro amplificador programable (0-24 dB), un filtro paso bajo programable (de 2 a 33 MHz) y un amplificador comutable de 5 dB. Las señales de salida están conectadas a la USRP que, mediante un oscilador de cristal controlado por tensión con un reloj de referencia de 64 MHz (± 20 ppm) y el conversor A/D AD9862 de Analog Devices (14 bits por muestra, $f_s = 64$ Msps) muestrea las componentes de la señal I&Q. A continuación, una FPGA Altera Cyclone EP1C12Q240C8 realiza la bajada en frecuencia digital a banda base con valores de diezmado programables, y finalmente se envía la señal digital a través de un puerto USB 2.0 (que ofrece un ancho de banda 32 MBps) hacia el equipo que ejecuta el software receptor.



Figura 6-1: Antena Novatel GPS-600 (imagen izquierda) y Cabezal USRP equipada con la placa DBSRX (imagen derecha) utilizados en el experimento 1.

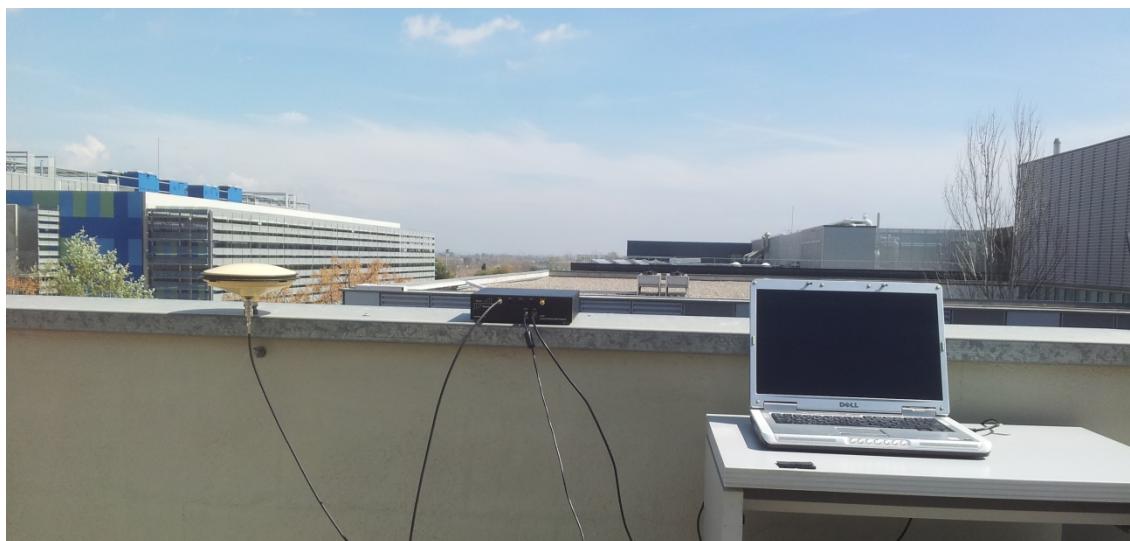


Figura 6-2: Equipo completo utilizado en el experimento 1.

El equipo anfitrión es un PC portátil DELL Inspiron 6400 equipado con un procesador Intel Centrino duo Genuine Intel ® CPU T2400 con 2 núcleos a 1.83 GHz y con 2 GB de RAM, sobre Linux Ubuntu 10.10 32 bits y GNU Radio 3.6.2. Una imagen del equipo completo puede verse en la Figura 6-2.

6.2.2 Configuración del Software GNSS-SDR

Con el fin de usar el dispositivo USRP v1 rev 4.5 es necesario seleccionar *UHDSignalSource* como el bloque *SignalSource* en el archivo de configuración de GNSS-SDR (gnss-sdr.conf). Para el uso de este cabezal de radiofrecuencia se deben fijar 2 opciones más: por un lado la ganancia del cabezal en dB (que se ha configurado a 60 dB) y por otro lado en qué ranura se ha conectado la tarjeta DBSRX (ranura A:0 o B:0), ya que la USRP dispone de dos ranuras para conectarla (en este experimento está conectada en la ranura B:0). A continuación se puede encontrar una configuración de trabajo para la recepción de una señal GPS L1 C/A:

```
[GNSS-SDR]
; # OPTIONS ##### GLOBAL #####
GNSS-SDR.internal_fs_hz = 2048000
; ##### CONTROL_THREAD CONFIG #####
ControlThread.wait_for_flowgraph = false
; ##### SIGNAL_SOURCE CONFIG #####
SignalSource.implementation = UHD_Signal_Source
SignalSource.item_type = gr_complex
SignalSource.sampling_frequency = 4000000
SignalSource.enable_throttle_control = false
SignalSource.freq=1575420000
SignalSource.gain=60
SignalSource.subdevice=B:0
```

La frecuencia de muestreo del front-end está fijada originalmente a 64 Msps, pero debe reducirse por hardware aplicando un diezmado entero (8x para trabajar a 8 Msps, 16x para trabajar a 4 Msps, ...) ya que el interfaz USB tiene una limitación de velocidad de 32 MBps (bytes por segundo). En el experimento al fijar la frecuencia a 4000000 se está aplicando un diezmado de 16x. Sin embargo, esta frecuencia de muestreo es demasiado alta para ejecutar el receptor en tiempo real con múltiples canales en un ordenador antiguo como el que se pretende usar en este experimento, el objetivo es trabajar a una frecuencia de muestreo de 2048000 Hz ya que de esta forma el número de muestras por milisegundo es de 2048, que al ser un múltiplo de 2 aprovecha la capacidad de la FFT en el bloque de adquisición sin tener que aplicar la técnica de *zero-padding*. Para conseguir la frecuencia de muestreo deseada debemos activar el *resampler* en el bloque *SignalConditioner*.

El receptor software puede resolver el problema al habilitar el bloque *DirectResampler* explicado en el apartado 3.4.1.23.3.4, que realiza un remuestreo arbitrario gastando muy pocos recursos. La configuración del bloque acondicionador de señal es la siguiente:

```
;##### SIGNAL_CONDITIONER CONFIG #####
;## It holds blocks to change data type, filter and resample input data.
;#implementation: Use [Pass_Through] or [Signal_Conditioner]
;#[Pass_Through] disables this block and the [DataTypeAdapter], [InputFilter]
;#and [Resampler] blocks
;#[Signal_Conditioner] enables this block. Then you have to configure
;#[DataTypeAdapter], [InputFilter] and [Resampler] blocks

SignalConditioner.implementation=Signal_Conditioner

;##### DATA_TYPE_ADAPTER CONFIG #####
;## Changes the type of input data. Please disable it in this version.
;#implementation: [Pass_Through] disables this block

DataTypeAdapter.implementation=Pass_Through

;##### INPUT_FILTER CONFIG #####
;## Filter the input data. Can be combined with frequency translation for IF
;#signals
;#implementation: Use [Pass_Through] or [Fir_Filter] or [Freq_Xlating_Fir_Filter]
;#[Freq_Xlating_Fir_Filter] enables FIR filter and a composite frequency
;#translation that shifts IF down to zero Hz.

InputFilter.implementation= Pass_Through

;##### RESAMPLER CONFIG #####
;## Resamples the input data.
;#implementation: Use [Pass_Through] or [Direct_Resampler]
;#[Pass_Through] disables this block

Resampler.implementation=Direct_Resampler

;#sample_freq_in: the sample frequency of the input signal

Resampler.sample_freq_in=4000000

;#sample_freq_out: the desired sample frequency of the output signal

Resampler.sample_freq_out=2048000
```

El experimento se ha realizado con cuatro configurados para la señal GPS L1 C/A, tan sólo un canal en adquisición simultáneamente, y todos los canales de adquisición con la implementación *GpsL1CaPcpsAcquisition* explicada en el apartado 4.8.2.1, con un espacio de búsqueda de ± 10 kHz, 500 Hz para cada bin frecuencial y una probabilidad de falsa alarma de 0.00001.

6.2.3 Resultados obtenidos

El experimento ha dado como resultado un posicionamiento positivo en tiempo real, generando los correspondientes ficheros .kml y .nmea explicados en el apartado 2.4.1.2 de este proyecto. Estos ficheros han sido procesados con software externo para poder validar y representar gráficamente los resultados.

Por un lado se ha utilizado el software VisualGPS [6-7], un software que en su versión gratuita permite leer y ejecutar los ficheros .nmea y representar gráficamente varios medidas del experimento en diversas ventanas.

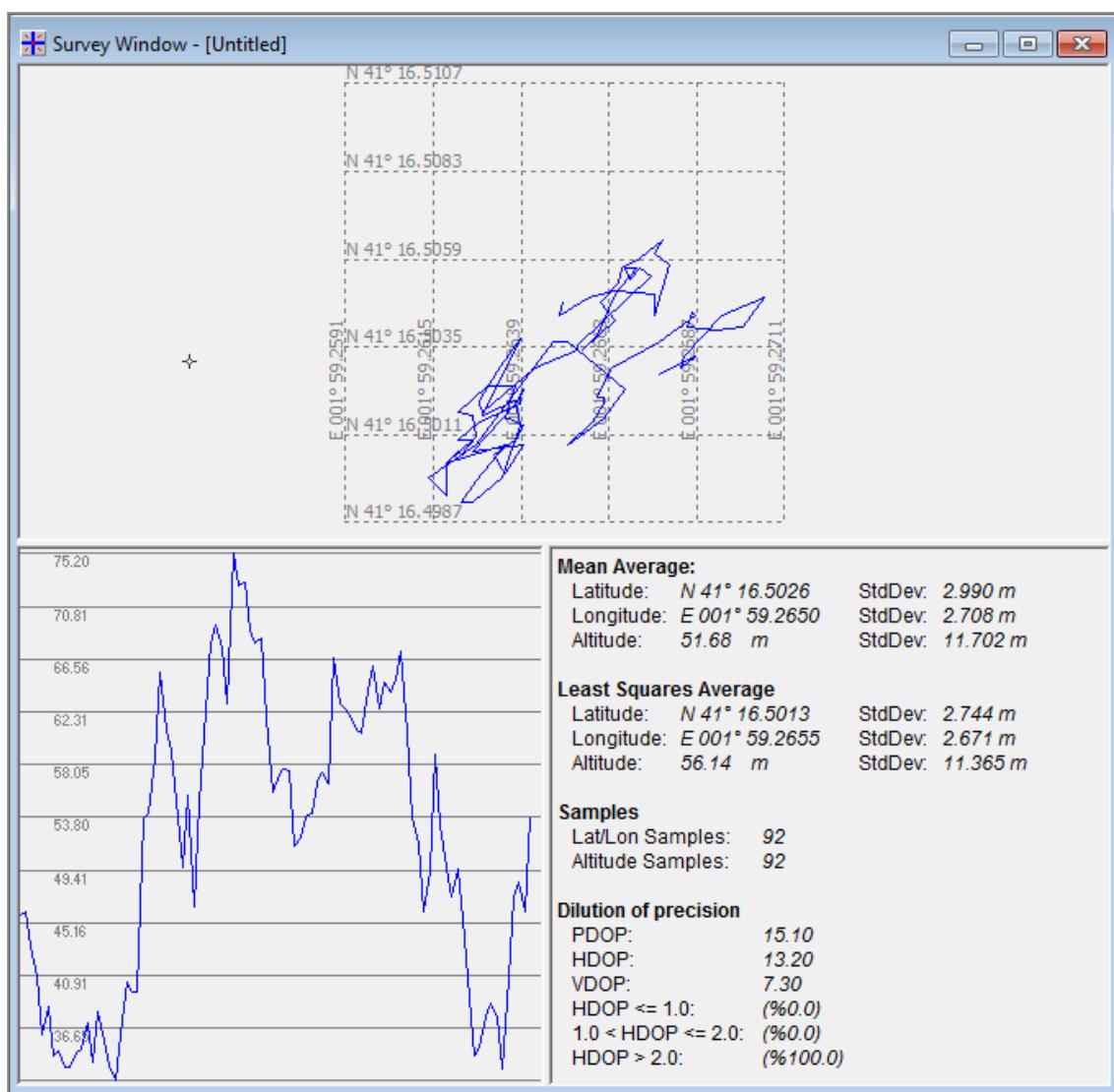


Figura 6-3: Survey Window del software VisualGPS que representa gráficamente las medidas de posición, así como los promedios de estas medidas, para el experimento 1.

En la Figura 6-3 puede observarse la ventana Survey Window que muestra una representación de la posición, tanto de la latitud y longitud (parte superior de la ventana) como de la altitud (parte inferior izquierda de la misma). Además nos da los promedios de las tres magnitudes para todas las realizaciones del experimento. Para ello realiza dos tipos de promedios: *Mean Average* (la media aritmética que consiste en la suma de todos los valores dividida por el número de realizaciones) y *Least Squares Average* (el mejor método de ajuste por mínimos cuadrados). De estos resultados comentar que como es de esperar, las medidas de altitud dan peores resultados (con una desviación estándar de 11-12 metros) que las medidas de latitud y longitud (con una desviación estándar de 2-3 metros). Esto es debido a la disposición de los satélites con respecto al receptor.

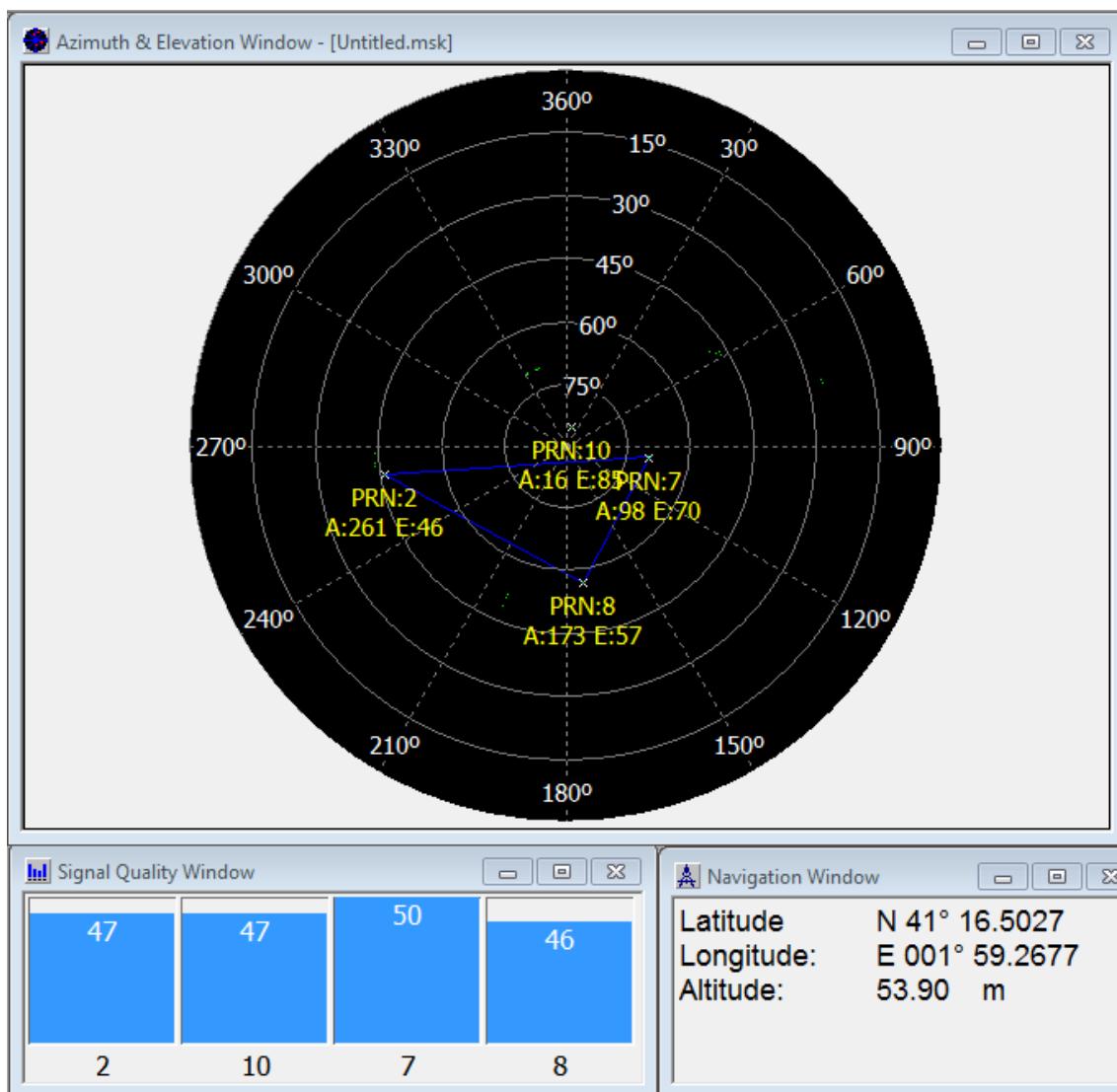


Figura 6-4: Azimuth&Elevation Window, Signal Quality Window y Navigation Window del software VisualGPS que representan la información de elevación y azimuth de los satélites, su SNR y las medidas de la última posición respectivamente para las señales del experimento 1.

Otras medidas que proporciona el software VisualGPS pueden verse en la Figura 6-4, en ella se muestran 3 ventanas más del software que proporcionan:

- La información del PRN de cada satélite que ha sido seguido por un canal, así como su azimuth y su elevación (*Azimuth&Elevation Window*), y que nos permite observar la disposición geométrica de los satélites respecto al receptor.

- La información de la SNR con que llega la señal de cada satélite (*Signal Quality Window*).
- La información numérica de la última posición positiva (*Navigation Window*).

Concretamente, en este primer experimento se puede observar en la ventana *Azimuth&Elevation Window* que 2 satélites (PRN 10 y PRN 7) se encuentran muy cercanos, lo que da lugar a una geometría muy cerrada, cosa que empeora la estimación de las distancias; por el contrario, se puede observar que todos los satélites tienen una elevación suficientemente alta para que llegue un buen nivel de potencia de señal, aspecto que queda reflejado en los valores de las SNRs que muestra la ventana *Signal Quality Window*.

Por último, aprovechando la capacidad para generar ficheros .kml del receptor GNSS-SDR se ha verificado en Google Earth [6-8] (soft gratuito de Google que representa las medidas de posición sobre un mapa del globo terráqueo) la veracidad de la posición obtenida. El resultado puede verse en la Figura 6-5, donde se observa la posición sobre la azotea del edificio del CTTC (Centre Tecnològic de Telecomunicacions de Catalunya) donde se llevó a cabo este experimento.



Figura 6-5: Representación en Google Earth de los datos de la posición guardada en el fichero .kml para el experimento 1.

6.3 Experimento 2: Posicionamiento en tiempo real con SiGE GN3S y un ordenador de gama media.

El objetivo de este apartado es probar el funcionamiento del software GNSS-SDR con el *front-end* USB SiGe GN3S Sampler v2 [6-3]. Uno de los problemas que se derivan de su uso es que la señal que entrega el cabezal se encuentra a una frecuencia intermedia (IF) de 38,4 kHz que debe ser eliminada por el bloque *Input Filter* (explicado en el apartado 3.4) del *Signal Conditioner* (Capítulo 3) de GNSS-SDR. Concretamente se probará el funcionamiento del bloque *Frequency Translating FIR Filter* explicado en el apartado 3.4.1.2.

6.3.1 Hardware utilizado

El hardware utilizado en este experimento se constituye de la antena ANT-555 de On Shine Enterprise Co. Ltd [6-4], que es una antena de cerámica de tipo patch equipada con un amplificador interno de bajo ruido (LNA) para reducir la figura de ruido total.



Figura 6-6: Equipo completo utilizado en el experimento 2.

La antena se conecta al front-end de RF SiGe GN3S v2 USB, desarrollado por SiGe Semiconductor (adquirida en junio de 2011 por Skyworks) en colaboración con el GNSS Lab de la Universidad de Colorado [6-5]. Este cabezal de radiofrecuencia se compone básicamente de dos circuitos integrados distintos. Por un lado, las

operaciones relacionadas con las funciones GNSS se realizan en el circuito integrado SiGe 4120 , un ASIC (Application Specific Integrated Circuit) para GPS que realiza la amplificación de RF, el filtrado, la conversión a baja frecuencia, y el muestreo en banda base. Por otro lado, el microcontrolador USB 2.0 EZ-USB FX2LP de Cypress Semiconductors [6-6] es el encargado de leer las muestras digitales procedentes del ASIC SiGe 4120 y enviarlos en tiempo real al PC a través del bus serie universal (USB).

El conversor AD del ASIC SiGe 4120 está configurado de serie para proporcionar un flujo de datos con una frecuencia de muestreo de 8.1838 Msps a una frecuencia intermedia (IF) de 38400 kHz. El microcontrolador FX2LP envía las muestras de la señal agrupadas por parejas, en fase y en cuadratura (I/Q) en formato short.

Por último, el cabezal se conecta a un ordenador portátil ASUS U44S que dispone de una CPU Intel Core i5-2450M de 4 núcleos a 2.5 GHz, con 8 GBytes de memoria RAM, sobre Linux Ubuntu 12.04 y GNU Radio 3.6.4.

En la Figura 6-6 puede observarse el montaje del hardware utilizado.

6.3.2 Configuración del Software GNSS-SDR

Con el fin de usar el dispositivo SiGe GN3S Sampler v2 es necesario seleccionar *Gn3sSignalSource* como el bloque *SignalSource* en el archivo de configuración de GNSS-SDR (*gnss-sdr.conf*). Dado que éste es un front-end específico para la recepción de la señal GNSS, no hay necesidad de configurar ningún otro parámetro. A continuación se puede encontrar una configuración de trabajo para la recepción de una señal GPS L1 C/A:

```
[GNSS-SDR]
; # OPTIONS ##### GLOBAL #####
GNSS-SDR.internal_fs_hz = 2045950
; ##### CONTROL_THREAD CONFIG #####
ControlThread.wait_for_flowgraph = false
; ##### SIGNAL_SOURCE CONFIG #####
SignalSource.implementation = GN3S_Signal_Source
SignalSource.item_type = gr_complex
SignalSource.sampling_frequency = 8183800
SignalSource.enable_throttle_control = false
```

La frecuencia de muestreo del *front-end* está fijada por el ASIC del SiGe a 8.1838 Msps. Sin embargo, esta frecuencia de muestreo es demasiado alta para ejecutar el receptor en tiempo real con múltiples canales, al menos en un ordenador personal de gama media como el utilizado en el experimento. Además de este

problema, las muestras de señal que entrega el front-end están a una frecuencia intermedia de 38.4 kHz que debe ser compensada por el bloque *SignalConditioner*.

El receptor software puede resolver ambos problemas al habilitar el bloque *Frequency Translating Finite Impulse Response* (FIR) explicado en el apartado 3.4.1.2. Este bloque de GNU Radio realiza las siguientes operaciones:

- Elimina la frecuencia IF.
- Realiza un filtrado paso bajo para evitar aliasing.
- Realiza una operación de diezmado para reducir la frecuencia de muestreo.

En este experimento se ha utilizado un factor de diezmado de 4. La frecuencia de muestreo resultante, que es la frecuencia de muestreo interna de GNSS-SDR, es $8183800/4 = 2045950$ Hz. La configuración del bloque acondicionador de señal es la siguiente:

```
;##### SIGNAL_CONDITIONER CONFIG #####
;## It holds blocks to change data type, filter and resample input data.
;#implementation: Use [Pass_Through] or [Signal_Conditioner]
;#[Pass_Through] disables this block and the [DataTypeAdapter], [InputFilter]
;#and [Resampler] blocks
;#[Signal_Conditioner] enables this block. Then you have to configure
;#[DataTypeAdapter], [InputFilter] and [Resampler] blocks

SignalConditioner.implementation=Signal_Conditioner

;##### DATA_TYPE_ADAPTER CONFIG #####
;## Changes the type of input data. Please disable it in this version.
;#implementation: [Pass_Through] disables this block

DataTypeAdapter.implementation=Pass_Through

;##### INPUT_FILTER CONFIG #####
;## Filter the input data. Can be combined with frequency translation for IF
;#signals
;#implementation: Use [Pass_Through] or [Fir_Filter] or [Freq_Xlating_Fir_Filter]
;#[Freq_Xlating_Fir_Filter] enables FIR filter and a composite frequency
;#translation that shifts IF down to zero Hz.

InputFilter.implementation=Freq_Xlating_Fir_Filter

;#The following options are used in the filter design of Fir_Filter and
;#Freq_Xlating_Fir_Filter implementation.
;#These options are based on parameters of gnuradio's function: gr_remez.
```

```
;#These function calculates the optimal (in the Chebyshev/minimax sense) FIR
;#filter impulse response given a set of band edges, the desired response on those
;#bands, and the weight given to the error in those bands.
```

```
; -- Los parámetros del filtro y sus coeficientes se omiten en este documento --
```

```
InputFilter.sampling_frequency=8183800
```

```
InputFilter.IF=38400
```

```
InputFilter.decimation_factor=4
```

```
,##### RESAMPLER CONFIG #####
```

```
,## Resamples the input data.
```

```
,#implementation: Use [Pass_Through] or [Direct_Resampler]
```

```
,#[Pass_Through] disables this block
```

```
Resampler.implementation=Pass_Through
```

En el experimento se ha realizado con seis canales con la misma configuración que el experimento del apartado 6.2. Esto es, seis canales de GPS, tan sólo un canal en adquisición simultáneamente, y todos los canales de adquisición con la implementación *GpsL1CaPcpsAcquisition* explicada en el apartado 4.8.2.1, con un espacio de búsqueda de ± 10 kHz, 500 Hz para cada bin frecuencial y una probabilidad de falsa alarma de 0.00001.

6.3.3 Resultados obtenidos.

El experimento ha dado como resultado un posicionamiento positivo en tiempo real. De nuevo se ha procedido a utilizar el software VisualGPS y los resultados se muestran en la Figura 6-7 y la Figura 6-8.

En la Figura 6-7 puede observarse la ventana *Survey Window* con la representación gráfica de la posición y el promedio de sus medidas. Volver a comentar que las medidas de altitud dan peores resultados (con una desviación estándar de 7-8 metros) que las medidas de latitud y longitud (con una desviación estándar de 2-3 metros). Pero comparando con las medidas del experimento 1 (explicado en el apartado 6.2) podemos observar que las medidas han mejorado (6-7 metros de desviación de las medidas de altitud en este experimento frente a los 11-12 metros del experimento 1). Este fenómeno puede haberse producido gracias al incremento del número de satélites así como su disposición geométrica frente al receptor, disposición que puede verse de nuevo en la ventana *Azimuth&Elevation Window* de la Figura 6-8.

Por otro lado comentar que en este caso sí puede observarse la diferencia de SNR entre satélites de elevación más baja (satélites PRN 2 y PRN 44 con 27° y 44° de elevación respectivamente y una SNR de 42 dB) frente a los de elevación más alta (satélite PRN 25, con elevación 70° y SNR de 45 dB).

Por último, aprovechando de nuevo la capacidad para generar ficheros .kml del receptor GNSS-SDR se ha verificado en Google Earth (soft gratuito de Google que representa las medidas de posición sobre un mapa del globo terráqueo) la veracidad

de la posición obtenida. El resultado puede verse en la Figura 6-9, donde se observa la posición sobre el césped del Campus Nord de la UPC (Universitat Politècnica de Catalunya) donde se llevó a cabo este experimento.

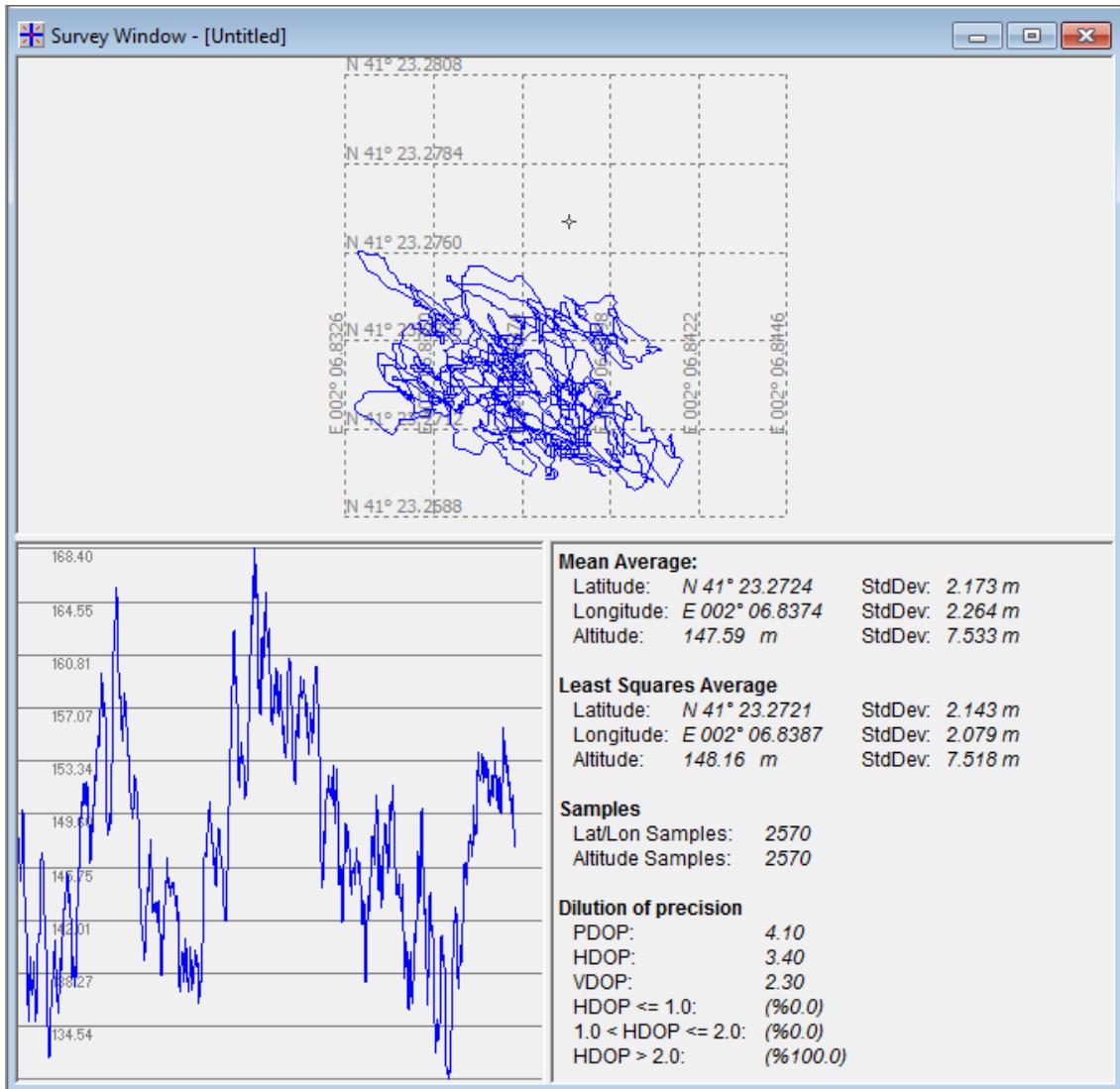


Figura 6-7: Survey Window del software VisualGPS que representa gráficamente las medidas de posición, así como los promedios de estas medidas, para el experimento 2.

Comentar además que en los dos experimentos llevados a cabo se han utilizado con éxito los bloques de adquisición *GpsL1CaPcpsAcquisition* y *pcps_acquisition_cc* (explicados en el apartado 4.8) y de canal *Channel* (explicado en el Capítulo 5) en el que el correcto funcionamiento de la máquina de estados finitos (FSM) ha sido verificado al ver la evolución de los estados que son escritos por pantalla durante la ejecución del receptor GNSS-SDR.

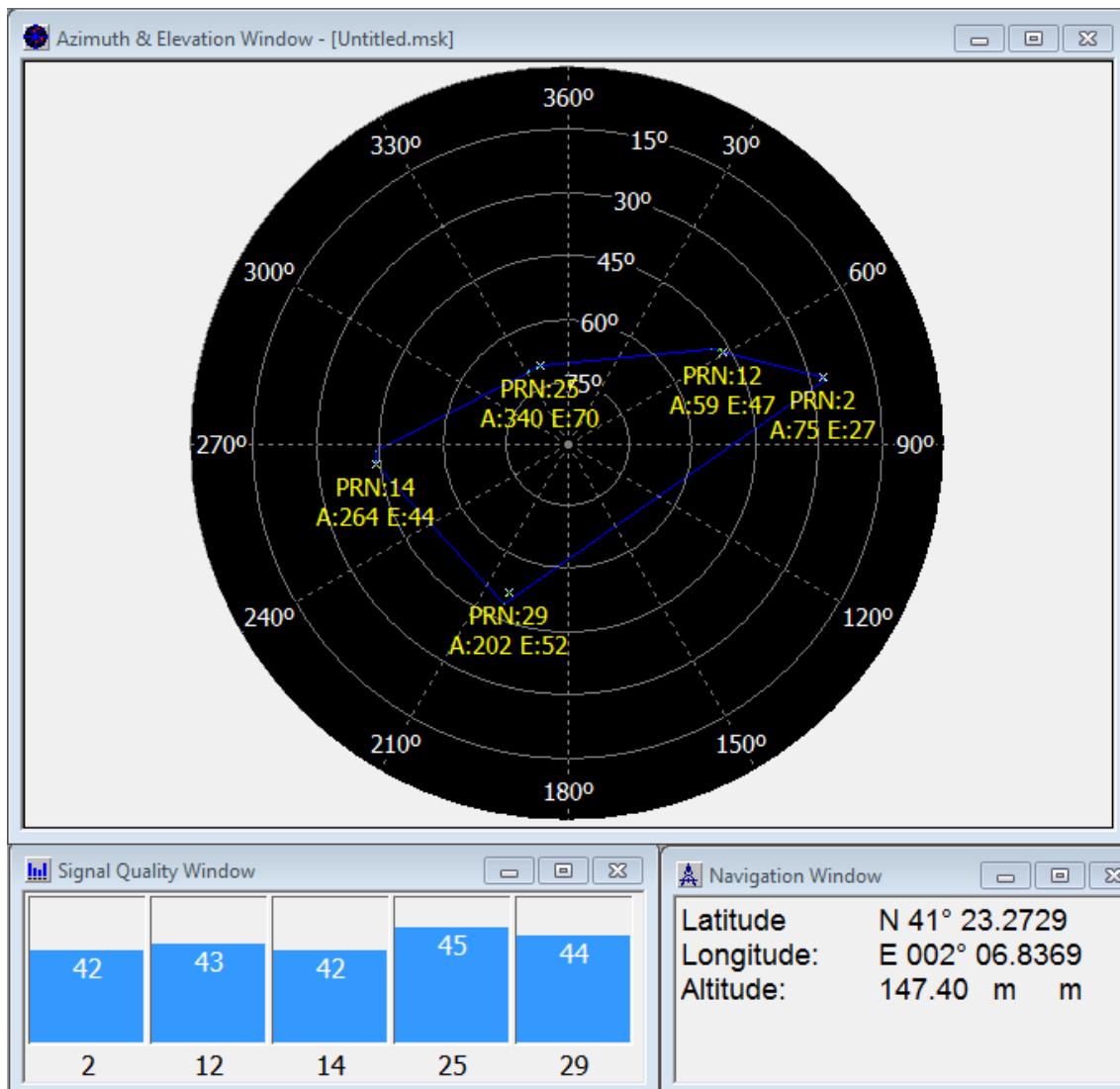


Figura 6-8: Azimuth&Elevation Window, Signal Quality Window y Navigation Window del software VisualGPS que representan la información de elevación y azimuth de los satélites, su SNR y las medidas de la última posición respectivamente para las señales del experimento 2.

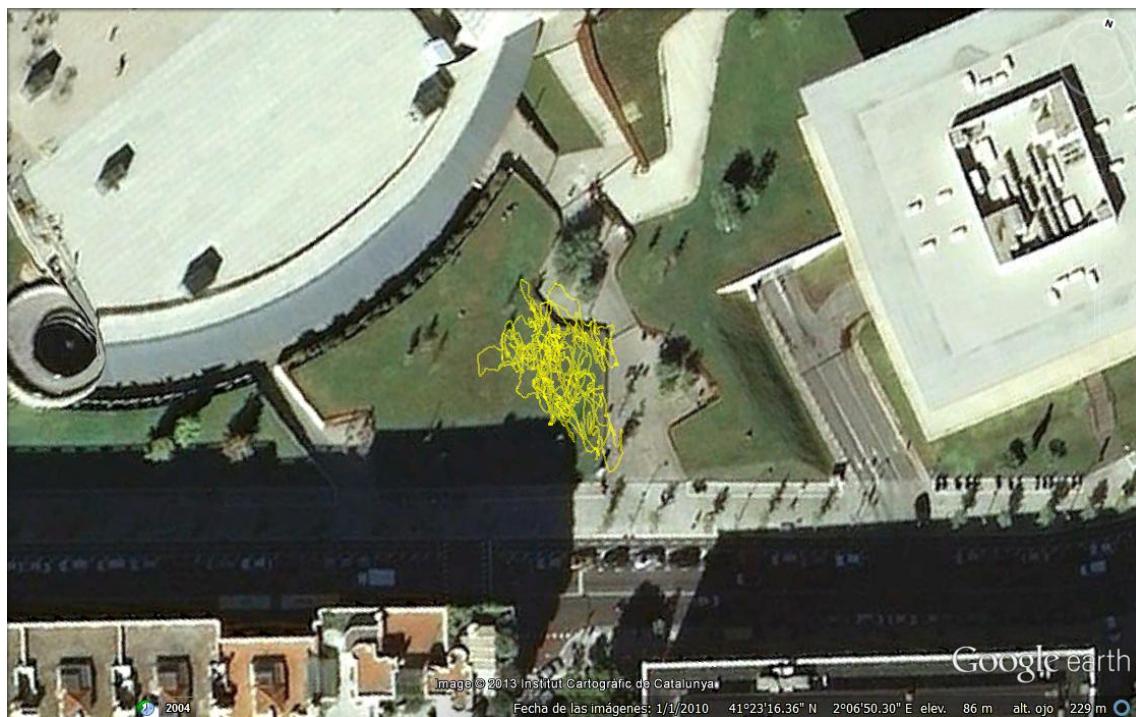


Figura 6-9: Representación en Google Earth de los datos de la posición guardada en el fichero .kml para el experimento 2.

6.4 Conclusiones.

Se han presentado en este capítulo dos experimentos con el fin de validar los bloques desarrollados por el autor en los capítulos anteriores del proyecto y ver su funcionalidad a la hora de formar parte de un receptor que funcione en tiempo real.

Para ello se ha realizado un primer experimento en el que se ha conseguido probar la utilidad del bloque *DirectResampler* explicado en el apartado 3.3.4 para conseguir posicionamiento en tiempo real en un ordenador de bajas prestaciones.

El segundo experimento ha consistido en probar el bloque *Frequency Translating FIR Filter* explicado en el apartado 3.4.1.2 y que ha posibilitado utilizar el cabezal de radiofrecuencia USB SiGe GN3S Sampler v2 eliminando la frecuencia intermedia (FI) que tiene el cabezal y consiguiendo un nuevo posicionamiento en tiempo real.

Para futuros trabajos comentar que el cuello de botella del receptor sigue estando en el bloque de adquisición y se propone mejorar su rendimiento utilizando intensivamente la librería Volk [6-9] así como la posibilidad de desviar cierto volumen de carga computacional del proceso de adquisición a la GPU de la tarjeta gráfica del ordenador anfitrión cuando sea posible.

6.5 Bibliografía

[6-1] Ettus Research, "Product Detail: DBSRX2 800-2300 MHz Rx", <http://www.ettus.com/product/details/DBSRX2>, Comprobado: 9 de junio de 2013.

[6-2] Ettus Research, "USRP1", <https://www.ettus.com/product/details/USRPPKG>, Comprobado: 9 de junio de 2013.

[6-3] Spark Fun Electronics Inc., "SiGe GN3S Sampler v2", <https://www.sparkfun.com/products/8238>, Comprobado: 9 de junio de 2013.

[6-4] On Shine Enterprise Co. Ltd, "ANT-555 series", <http://php2.twinner.com.tw/files/onshine/ANT555-2011.pdf>, Comprobado: 9 de junio de 2013.

[6-5] Colorado Center For Astrodynamics Research, <http://ccar.colorado.edu/gnss/>, Comprobado: 9 de junio de 2013.

[6-6] Cypress Semiconductors, "EZ-USB FX2LP" <http://www.cypress.com/?id=193>, Comprobado: 9 de junio de 2013.

[6-7] VisualGPS LLC, <http://www.visualgps.net/VisualGPS/>. Comprobado: 29 de junio de 2013.

[6-8] Google Earth, <http://www.google.com/earth/index.html>. Comprobado: 29 de junio de 2013.

[6-9] "VOLK," <http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk>, Comprobado: 22 de mayo de 2013.

7 EXTENSIÓN A LA SEÑAL GALILEO E1

7.1 Resumen.

La carrera para completar la constelación de satélites Galileo por fin ha comenzado. Dos satélites In-Orbit Validation (IOV) están transmitiendo señales de navegación desde el 12 de diciembre de 2011, y otros dos satélites IOV iniciaron su actividad el 12 de octubre de 2012 dando sus primeros resultados el 12 de marzo de 2013 con el primer posicionamiento positivo de una localización terrestre a partir de los cuatro satélites Galileo actualmente en órbita y de sus estaciones terrestres [7-1]. Para finales de 2014 se espera tener en órbita una constelación de servicio inicial de 18 satélites, y a finales de la década estará disponible un servicio totalmente operativo, con un mínimo de 30 satélites, según lo estimado por la Autoridad Europea de Supervisión de GNSS (GSA).

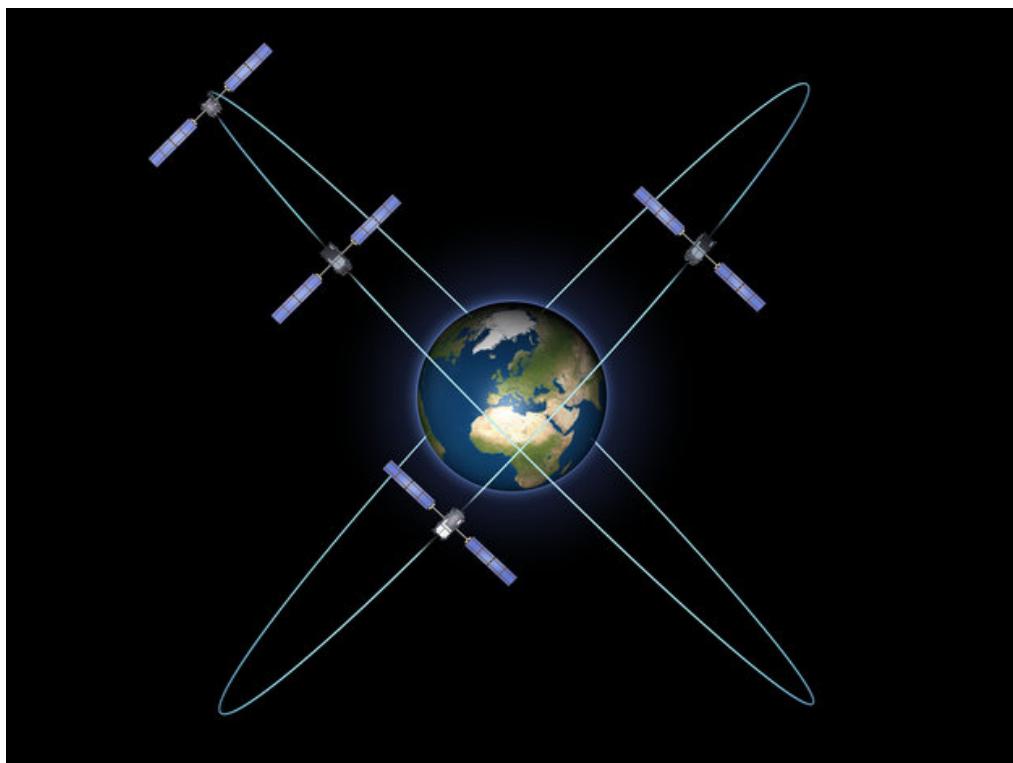


Figura 7-1: Los 4 satélites Galileo en órbita han dado su primer posicionamiento positivo el 12 de marzo de 2013. Figura extraída de [7-1]

En este capítulo se presenta la ampliación de la funcionalidad del receptor GNSS-SDR para adquirir, realizar el *tracking* y desmodular el mensaje de navegación de la señal libre Galileo E1 (*Galileo E1 open signal*), que incluye tanto la componente de datos E1B como la señal piloto E1C. El autor de este proyecto ha desarrollado e implementado el bloque de adquisición y ha participado en la implementación del bloque de *tracking* a partir de los bloques ya existentes para el *tracking* de la señal GPS. En el siguiente apartado se describe el modelo matemático de las señales a seguir y, a continuación, en el apartado 7.3 se presenta la arquitectura de los bloques diseñados. En el apartado 7.4 se presentan algunos resultados experimentales y, finalmente, el apartado 7.5 concluye el capítulo.

El trabajo realizado en este capítulo ha sido financiado por el programa Google Summer of Code 2012 (GSOC 2012) [7-2] y ha sido presentado en la Conferencia GNU Radio 2012 [7-3]. El vídeo presentado en dicha conferencia puede verse en [7-4].

7.2 Modelo de señal

La representación analítica de una señal recibida desde un satélite GNSS genérico se puede expresar como:

$$r(t) = a(t) s_T(t - \tau(t)) e^{-j2\pi f_d(t)} e^{j2\pi f_c t} + n(t) \quad (7-1)$$

donde $a(t)$ es la amplitud compleja, $s_T(t)$ es la señal compleja banda base transmitida, $\tau(t)$ es el retardo variante con el tiempo, $f_d(t) = f_c \cdot \tau(t)$ es el desplazamiento Doppler, f_c es la frecuencia portadora, y $n(t)$ es el ruido térmico. La forma de onda de $s_T(t)$ y sus parámetros dependen del sistema GNSS en particular y de la banda de frecuencia utilizadas por el receptor. La banda Galileo E1, centrada en $f_c^{(GalEI)} = 1575.420$ MHz y con un ancho de banda de referencia de 24.5520 MHz, utiliza la modulación denominada *Composite Binary Offset Carrier CBOC* $\left(6,1,\frac{1}{11}\right)$, definida en banda base como:

$$\begin{aligned} s_T^{(GalEI)}(t) = & \frac{1}{\sqrt{2}} \left(e_{EIB}(t) (\alpha \cdot sc_{EIBa}(t) + \beta \cdot sc_{EIBb}(t)) - \right. \\ & \left. - e_{EIC}(t) (\alpha \cdot sc_{EICa}(t) - \beta \cdot sc_{EICb}(t)) \right) \end{aligned} \quad (7-2)$$

donde las sub-portadoras $sc_{EIBa}(t)$, $sc_{EIBb}(t)$, $sc_{EICa}(t)$ y $sc_{EICb}(t)$ se definen como:

$$sc_{EIBa}(t) = sc_{EICa}(t) = sign(\sin(2\pi f_{s,EIa} t)) \quad (7-3)$$

$$sc_{E1Bb}(t) = sc_{E1Cb}(t) = \text{sign}(\sin(2\pi f_{s,E1b} t)) \quad (7-4)$$

y $f_{s,E1a} = 1.023 \text{ MHz}$ y $f_{s,E1b} = 6.138 \text{ MHz}$ son las frecuencias de las sub-portadoras. Los parámetros α y β se eligen para que la potencia combinada de las sub-portadoras $sc_A(t)$ y $sc_B(t)$ sea igual a $\frac{1}{11}$ de la potencia total de $e_{E1B}(t)$ más $e_{E1C}(t)$, antes de la aplicación de cualquier limitación de ancho de banda. Esto da como resultado unos valores de $\alpha = \sqrt{\frac{10}{11}}$ y $\beta = \sqrt{\frac{1}{11}}$. Una representación gráfica de la combinación de dichas sub-portadoras puede observarse en la Figura 7-2.

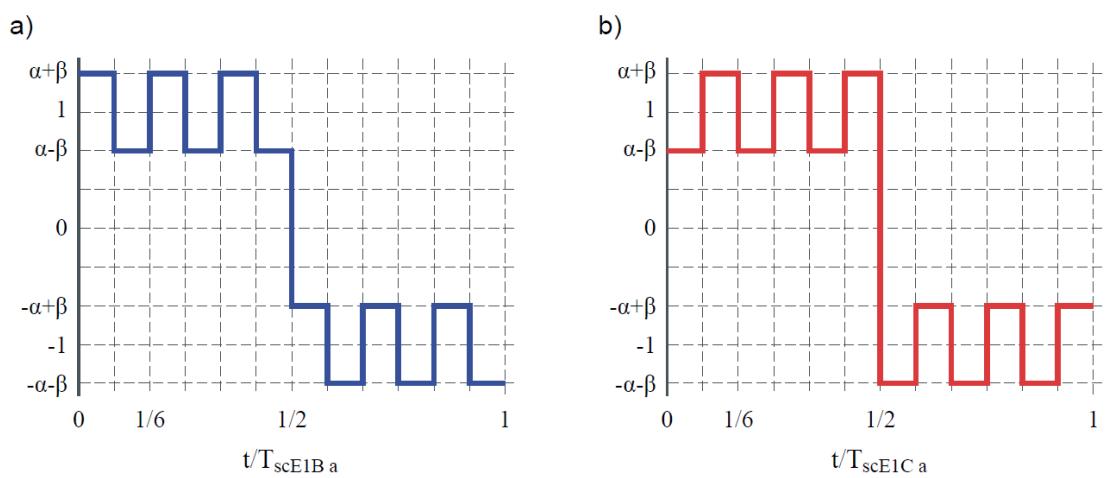


Figura 7-2: Un período de la sub-portadora CBOC para a) la componente de señal E1-B, y b) la componente de la señal E1-C. Figura extraída de [7-5].

La componente E1B contiene la secuencia de bits del Mensaje de Integridad/Navegación, $D_{I/NAV}$, transmitido a 250 sps (muestras/segundo) y destinado a los servicios de navegación, integridad y seguridad vital (Safety-of-Life, SoL):

$$e_{E1B}(t) = \sum_{l=-\infty}^{\infty} D_{I/NAV} [[l]_{4092}] \oplus C_{E1B} [|l|_{4092}] p(t - lT_{c,E1B}) \quad (7-5)$$

donde \oplus es el operador or-exclusivo (suma en módulo 2), C_{E1B} es un código de ruido pseudoaleatorio (PRN) único para cada satélite, $[l]_L$ significa la parte entera de $\frac{l}{L}$, $|l|_L$ significa l en módulo L , $T_{c,E1B} = \frac{1}{1.023} \mu\text{s}$, y $p(t)$ es un pulso rectangular de duración igual al periodo de chip, centrado en $t=0$ y filtrado en transmisión.

La componente E1C es una señal piloto (sin datos) con un código secundario, formando una codificación por niveles:

$$e_{EIC}(t) = \sum_{m=-\infty}^{\infty} C_{EICs} [m]_{25} \oplus \sum_{l=1}^{4092} C_{EICp}[l] p(t - mT_{c,EICs} - lT_{c,EICp}) \quad (7-6)$$

con $T_{c,EICp} = \frac{1}{1.023} \mu s$ y $T_{c,EICs} = 4 ms$. Los códigos primarios C_{EIB} y C_{EICp} son secuencias de códigos de memoria pseudoaleatorios definidos en [7-5, Anexo C.7 y C.8]. La secuencia binaria del código secundario C_{EICs} es 001110000001010110110010. Esta banda también contiene otra componente, la señal Galileo E1A, destinada al Servicio Público Regulado (*Public Regulated Service*, PRS), que utiliza la denominada modulación BOC(15,2.5) con subportadora cuadrada con fase de coseno y frecuencia $f_{s,EIA} = 15.345 MHz$ y periodo de chip $T_{c,EIA} = \frac{1}{2.5575} \mu s$. Los códigos de ensanchamiento de la señal PRS y la estructura del mensaje de navegación no se han hecho públicos, y por lo tanto la señal E1A no es un objetivo de estudio del presente proyecto.

Como en el caso de otras señales GNSS, el nivel de potencia recibida en la superficie de la Tierra de $r(t)$ es extremadamente débil, muy por debajo del umbral de ruido térmico. La potencia mínima recibida en Tierra, definida a la salida de la antena del receptor del usuario adaptada a 0 dBi para una polarización circular a derechas cuando la elevación del satélite es superior a 10 grados, es de -157dBW, considerando que la potencia se comparte al 50/50% entre E1B/E1C.

7.3 Implementación

Para el presente capítulo, se han desarrollado implementaciones específicas para la señal Galileo E1 de los bloques de Adquisición y *Tracking* del receptor GNSS-SDR así como la adición correspondiente de las señales PRN de satélites Galileo en el Plano de Control (PRN 11 y PRN 12, los dos satélites en órbita en el momento de la realización de este capítulo del proyecto), y una serie de test unitarios que garanticen el correcto funcionamiento de tales implementaciones.

7.3.1 Galileo E1 Acquisition

Como se comentó en el Capítulo 4, el papel de un bloque de adquisición es la detección de la presencia/ausencia de señales procedentes de un satélite GNSS en concreto. En caso de una detección positiva, debería proporcionar una primera estimación del retardo de código $\hat{\tau}$ y de la frecuencia Doppler \hat{f}_d , estas estimaciones deben ser lo suficientemente precisas para inicializar los lazos de seguimiento del retardo (DLL) y de la fase (PLL) de la señal. Como ya se explicó en el apartado 4.6, mediante la explotación de los conceptos y la metodología de la teoría de la estimación, es posible demostrar que la estimación de máxima verosimilitud (ML) de τ y f_d se puede obtener mediante la maximización de la función

$$\hat{f}_{d_{ML}}, \hat{\tau}_{ML} = \arg \max_{f_d, \tau} \left\{ \left| \hat{R}_{xd} (f_d, \tau) \right|^2 \right\} \quad (7-7)$$

donde

$$\hat{R}_{xd} (f_d, \tau) = \frac{1}{N} \sum_{n=0}^{N-1} x_{IN} [n] d[nT_s - \tau] e^{-j2\pi f_d n T_s} \quad (7-8)$$

En la expresión (7-8) $x_{IN} [n]$ es la señal compleja que contiene las muestras I&Q de la señal recibida, T_s es el periodo de muestreo, τ es el retardo de código de la señal recibida respecto a una referencia local, f_d es el desplazamiento Doppler, N es el número de muestras en un código de ensanchamiento (correspondiente a 4 ms de señal para el código primario de la señal Galileo E1) y $d[n]$ es la señal de referencia generada localmente. El usuario también puede configurar la forma de $d[n]$ lo que permite simplificaciones que reducen la carga computacional. Como se muestra en la Figura 7-3, en receptores de banda estrecha la modulación CBOC puede ser sustituida por una modulación sinBOC con una penalización de rendimiento muy pequeña [7-6]. Para la componente de señal E1B, las señales de referencia disponibles en nuestra implementación son

$$d_{E1B}^{(CBOC)} [n] = \sum_{l=-\infty}^{\infty} C_{E1B} \left[l \Big|_{4092} \right] p(n - l T_{c,E1B}) (\alpha \cdot sc_{E1Ba} [n] + \beta \cdot sc_{E1Bb} [n]) \quad (7-9)$$

y

$$d_{E1B}^{(\sin BO C)} [n] = \sum_{l=-\infty}^{\infty} C_{E1B} \left[l \Big|_{4092} \right] p(n - l T_{c,E1B}) sc_{E1Ba} [n] \quad (7-10)$$

Mientras que para la señal E1C los usuarios pueden escoger entre

$$d_{E1C}^{(CBOC)} [n] = \sum_{m=-\infty}^{\infty} \sum_{l=1}^{4092} C_{E1Cp} [l] p(n - m T_{c,E1Cs} - l T_{c,E1Cp}) (\alpha \cdot sc_{E1Ca} [n] - \beta \cdot sc_{E1Cb} [n]) \quad (7-11)$$

y

$$d_{E1C}^{(\sin BO C)} [n] = \sum_{m=-\infty}^{\infty} \sum_{l=1}^{4092} C_{E1Cp} [l] p(n - m T_{c,E1Cs} - l T_{c,E1Cp}) sc_{E1Ca} [n] \quad (7-12)$$

La maximización en (7-7) requiere de la búsqueda en dos dimensiones de los resultados de una función. Estos resultados de salida provienen de la multiplicación y la suma de N muestras complejas, convirtiéndose en el cuello de botella computacional de todo el proceso. Como se explicó en el apartado 4.5.3.1, un método

habitual para aliviar este problema es recurrir a la convolución circular basada en la FFT, que intercambia el costoso proceso de multiplicación y suma por una transformada discreta de Fourier, un producto vectorial y una transformada inversa, aprovechándose de las implementaciones eficientes disponibles para tales operaciones [7-7].

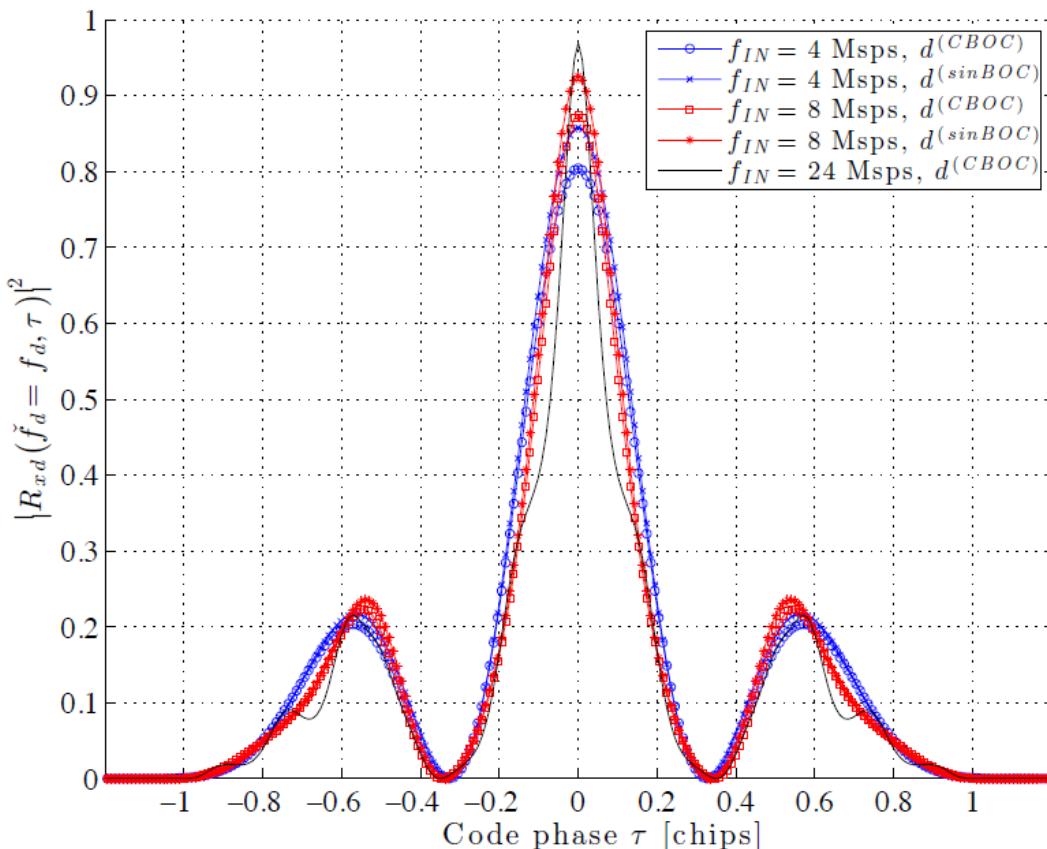


Figura 7-3: $|R_{xd}(f=f_d, \tau)|^2$ normalizada para diversas formas de onda de la señal local $d[n]$ y diferentes frecuencias de muestreo.

La magnitud de (7-8), también conocida como función de ambigüedad cruzada, se utiliza para decidir si el satélite correspondiente a la réplica local $d[n]$ está a la vista o no.

Recurriendo a la teoría de detección de señales, es posible definir estadísticos de prueba con las propiedades deseadas explicadas en el apartado 4.7.4.1. Como ya se vio en el apartado 4.7 un criterio comúnmente utilizado en el problema de la detección es la maximización de la probabilidad de detección (P_d) sujeto a una determinada probabilidad de falsa alarma (P_{fa}). Una posible solución al problema se puede encontrar aplicando el criterio de Neyman-Pearson, que requiere un perfecto conocimiento de los parámetros de la señal [7-8]. Suponiendo ruido blanco gaussiano aditivo y sustituyendo los parámetros de sincronización por sus estimadores ML en el detector NP, se obtiene la función del denominado *Generalized Likelihood Ratio Test* (GLRT), que puede ser escrita como:

$$T_{GLRT}(x_{IN}) = \max_{f_d, \tau} \left(\frac{|\hat{R}_{xd}(f_d, \tau)|^2}{\hat{R}_{xx}} \right) \quad (7-13)$$

donde \hat{R}_{xx} es una estimación de la potencia de la señal de entrada. Como se explicó en 4.7.4.3 se puede demostrar [7-8] que el algoritmo implementado es conocido con el nombre de detector CFAR (*Constant False Alarm Rate*) porque la probabilidad de falsa alarma no depende de la potencia de ruido y se mantiene constante durante la ejecución de todo el algoritmo.

Algorithm 1 Parallel Code Phase Search (PCPS) acquisition algorithm.

Require: Input signal buffer x_{IN} of N complex samples, provided by the Signal Conditioner; on-memory FFT of the local replica, $D[n] = FFT_N\{d[n]\}$; acquisition threshold γ ; freq. span $[f_{min} f_{max}]$; freq. step f_{step} .

Ensure: Decision positive or negative signal acquisition. In case of positive detection, it provides coarse estimations of code phase $\hat{\tau}_{acq}$ and Doppler shift $\hat{f}_{d_{acq}}$ to the Tracking block.

- 1: Compute input signal power estimation:

$$\hat{P}_{in} = \frac{1}{N} \sum_{n=0}^{N-1} |x_{IN}[n]|^2.$$
 - 2: **for** $\check{f}_d = f_{min} : f_{step} : \check{f}_d = f_{max}$ **do**
 - 3: Carrier wipe-off:

$$x[n] = x_{IN}[n] \cdot e^{-(j2\pi\check{f}_d n T_s)}, \text{ for } n = 0, \dots, N - 1.$$
 - 4: $X[n] = FFT_N\{x[n]\}$
 - 5: $Y[n] = X[n] \cdot D[n], \text{ for } n = 0, \dots, N - 1.$
 - 6: $R_{xd}(\check{f}_d, \tau) = \frac{1}{N^2} IFFT_N\{Y[n]\}$
 - 7: **end for**
 - 8: Search maximum and its indices in the search grid:

$$\{S_{max}, f_i, \tau_j\} \Leftarrow \max_{f, \tau} |R_{xd}(f, \tau)|^2$$
 - 9: Compute the GLRT function with normalized variance:

$$\Gamma_{GLRT} = \frac{2 \cdot N \cdot S_{max}}{\hat{P}_{in}}$$
 - 10: **if** $\Gamma_{GLRT} > \gamma$ **then**
 - 11: Declare positive acquisition and provide $\hat{f}_{d_{acq}} = f_i$ and $\hat{\tau}_{acq} = \tau_j$.
 - 12: **else**
 - 13: Declare negative acquisition.
 - 14: **end if**
-

Figura 7-4: Algoritmo 1, algoritmo que implementa la adquisición mediante el método *Parallel Code Phase Search* (PCPS) para el software GNSS-SDR.

La aplicación implementada se describe en el Algoritmo 1 de la Figura 7-4. El cálculo de la transformada rápida de Fourier y su inversa (pasos 4 y 6 del algoritmo) se realiza a través de los métodos de GNU Radio que encapsulan funciones de la librería FFTW3 [7-9], donde se implementa un método eficiente para calcular la Transformada Discreta de Fourier. El producto entre vectores de señales complejas que debe llevarse a cabo en el paso 5 se realiza mediante las funciones de la librería VOLK (Vector-Optimized Library of Kernels) [7-10], una librería diseñada para facilitar el uso por parte del programador del con conjunto de instrucciones SIMD del procesador, ya que encapsula el acceso a estas instrucciones en funciones de más alto nivel que se aprovechan de las técnicas de computación paralela y permiten escribir un código eficiente y portable. La librería VOLK dispone de implementaciones de funciones matemáticas básicas en diferentes conjuntos de instrucciones SIMD (SSE, SSE2, SEE3, MMX, AVX, etc.) y, en tiempo de compilación, utiliza la implementación más eficiente en el procesador en el que se ejecutará el programa..

La implementación del bloque que realiza la adquisición de las señales Galileo E1B y E1C explicado en este apartado consta tan sólo de un adaptador *GalileoE1PcpsAmbiguousAcquisition*, que hereda directamente de la clase *AcquisitionInterface* explicada en el apartado 4.8.1, y se encarga de adaptarla al bloque de GNU Radio *pcps_acquisition_cc* explicado en el apartado 4.8.2.2. Cabe comentar que en este caso no es necesario crear un nuevo bloque de GNU Radio dado que el algoritmo utilizado para la adquisición (el algoritmo *Parallel Code Phase Search Acquisition*) es el mismo independientemente del sistema utilizado (GPS o Galileo). Esta clase se encarga también de generar las señales locales definidas en las ecuaciones (7-9), (7-10), (7-11) y (7-12) a partir de la librería *galileo_e1_signal_processing.h* (también diseñada por el autor de este proyecto) en la que se definen las funciones necesarias para tal fin. Por último, también se ha implementado un fichero de cabeceras (*Galileo_E1.h*) que es un fichero donde se definen ciertos parámetros propios de las señales Galileo E1B y E1C tales como la frecuencia portadora, la frecuencia de los diferentes códigos (primario y secundario), los códigos PRN de los satélites, y otros parámetros del sistema.

7.3.2 Galileo E1 Tracking

El bloque de *tracking* también está recibiendo el flujo de datos $x_{IN}[n]$ del acondicionador de señal en paralelo al bloque de adquisición, pero no inicia su actividad hasta que recibe un mensaje del Plano de Control de GNSS-SDR conforme se ha producido una "adquisición positiva". En este momento, el canal activa el bloque de *tracking*, que puede leer las estimaciones gruesas de los observables (retardo de código τ y frecuencia Doppler f_d) que la adquisición ha guardado en el objeto *GnssSynchro* que comparten ambos bloques con el canal. Una vez activado, el papel del bloque de *tracking* es el de refinar estas estimaciones y rastrear sus cambios a lo largo del tiempo así como desmodular los bits de navegación.

Tres parámetros son pertinentes para el seguimiento de las señales: la evolución del retardo o fase de código τ , el desplazamiento Doppler f_d , y la fase de la portadora ϕ . Según el principio de Máxima Verosimilitud (ML) expresado en (7-7), la obtención de los estimadores óptimos implica la maximización de la correlación de la señal entrante con la señal de referencia generada localmente. Esto se consigue normalmente con estructuras de bucle cerrado diseñadas para minimizar la diferencia entre el retardo de código, la fase de la portadora y la frecuencia Doppler de la señal entrante con respecto a la réplica local.

En el caso del seguimiento del retardo de código, la función de coste se maximiza usando circuitos realimentados que realizan la derivada $\frac{dR_{xd}(\tau)}{d\tau}$ utilizando los cruces por cero como un detector del error temporal. Este es el caso de la arquitectura utilizada en los Lazos de Seguimiento del Retardo (Delay Lock Loop, DLL) y su amplia gama de variantes que habitualmente se usan en el *tracking* de las señales GPS L1, donde el receptor calcula tres muestras de R_{xd} , que se conocen normalmente con el nombre de *Early* $E = R_{xd}(\hat{\tau} - \varepsilon)$, *Prompt* $P = R_{xd}(\hat{\tau})$ y *Late* $L = R_{xd}(\hat{\tau} + \varepsilon)$, con un valor de ε que va, normalmente, desde $0.1T_c$ hasta $0.5T_c$, y después calcula el error temporal con alguna combinación de estas tres muestras, dicha combinación se conoce como discriminador. En el caso de Galileo E1, la función de correlación de la modulación CBOC $\left(6,1,\frac{1}{11}\right)$, puede generar ambigüedades, tal y como se muestra en la Figura 7-3. La posibilidad de realizar el seguimiento de un máximo local en lugar del máximo global puede evitarse mediante el uso de discriminadores que consideren dos muestras adicionales de la función de coste, denominadas *Very Early* $VE = R_{xd}(\hat{\tau} - \varepsilon')$ y *Very Late* $VL = R_{xd}(\hat{\tau} + \varepsilon')$, con $\varepsilon' > \varepsilon$.

En el caso de los lazos de seguimiento de portadora, puede utilizarse el canal piloto E1C para la estimación del error de fase, ya que no contiene transiciones de bits de datos, y, teóricamente, se puede realizar una integración coherente de varios períodos de código durante el tiempo que sea necesario. Como consecuencia, se puede utilizar un discriminador que sea insensible a saltos de fase. Realizar el seguimiento de la señal piloto mediante un PLL puro, así como utilizar integraciones coherentes más largas, mejora la sensibilidad del *tracking* de la portadora, es decir, la potencia de señal mínima a la que el receptor puede mantener el proceso de *tracking* enganchado.

Además de realizar un seguimiento de los parámetros de sincronización, el bloque de *tracking* también debe implementar detectores de enganche (*lock detectors*) tanto del retardo de código como de la portadora, proporcionando indicadores de rendimiento del *tracking*.

La implementación realizada en el receptor GNSS-SDR se describe en el Algoritmo 2 de la Figura 7-5. El cálculo de los valores complejos VE, E, P, L y VL en el paso 4 se ha llevado a cabo utilizando la biblioteca VOLK. El discriminador del PLL implementado en el paso 5 es el conocido *arctangent (four-quadrant) discriminator*, y para el DLL se ha utilizado el denominado *Nornalized Very Early Minus Late Power discriminator* propuesto en [7-11] y cuya formulación puede verse en el paso 9 del algoritmo. Para el detector de *lock* del retardo de código (paso 12 del algoritmo), se ha utilizado el estimador denominado *Squared Signal-to-Noise Variance* (SNV) propuesto en [7-12]. En el caso del detector de *lock* de la portadora (paso 13), se ha utilizado la estimación normalizada del coseno de dos veces la fase de la portadora [7-13]. Los valores del indicador de *lock* van desde -1, cuando la portadora generada localmente está completamente desenganchada, hasta 1, que indica una coincidencia perfecta. Cuando los detectores de *lock* del retardo de código o de la portadora están por debajo de un umbral determinado durante un número consecutivo de períodos de código ϑ' (paso 19 del algoritmo), el bloque de *tracking* informa de este hecho al Plano de Control de GNSS-SDR a través de un mensaje en la cola de mensajes general del receptor.

Algorithm 2 Implemented tracking algorithm.

Require: Complex sample stream, \mathbf{x}_{IN} ; estimations of code phase $\hat{\tau}_{acq}$ and Doppler shift $\hat{f}_{d_{acq}}$; buffer size for power estimation, \mathcal{U} ; carrier lock detector threshold, \mathcal{T} ; $CN0_{min}$; maximum value for the lock fail counter, ϑ ; correlators spacing ϵ and ϵ' ; loop filters bandwidth BW_{DLL} and BW_{PLL} ; integration time T_{int} .

Ensure: Track signal's synchronization parameters within a given lock margin. Inform about a loss of lock.

Initialization: Using $\hat{\tau}_{acq}$ and a sample counter \mathcal{N} , skip samples until \mathbf{x}_{IN} is aligned with local PRN replica. Set $v = 0$, $k = 0$, $\hat{f}_{d_0} = \hat{f}_{d_{acq}}$, $\hat{\phi}_0 = 0$, $\psi_1 = 0$, $N_1 = \text{round}(T_{int}f_{IN})$.

- 1: Increase the integration period counter: $k = k + 1$.
- 2: Generate local code references: for $n = 1 \dots N_k$,
 $s[n] = d_{E1B/E1C_p} [\text{round}(\delta_k \cdot n + \psi_k)]$, where
 $\delta_k = \frac{1}{T_{c,E1B} \cdot f_{IN}} \left(1 + \frac{\hat{f}_{d_{k-1}}}{f_c^{(Gal E1)}} \right)$, and the Very Early, Early, Late, and Very Late versions with ϵ and ϵ' .
- 3: Generate local carrier: for $n = 1 \dots N_k$,
 $c[n] = e^{-j(2\pi \hat{f}_{d_{k-1}} \frac{n}{f_{IN}} + \text{mod}(\hat{\phi}_{k-1}, 2\pi))}$.
- 4: Perform carrier wipe-off and compute the complex samples VE_k , E_k , P_k , L_k and VL_k .
Example: $P_k = \frac{1}{N_k} \sum_{n=0}^{N_k-1} x_{IN}[n] s[n] c[n]$.
- 5: Compute PLL discriminator: $\Delta\hat{\phi}_k = \text{atan2} \left(\frac{P_{Q_k}}{P_{I_k}} \right)$.
- 6: Filter $\Delta\hat{\phi}_k$ with a bandwidth BW_{PLL} : $h_{PLL}(\Delta\hat{\phi}_k)$.
- 7: Update carrier frequency estimation (in Hz):
 $\hat{f}_{d_k} = \hat{f}_{d_{acq}} + \frac{1}{2\pi T_{int}} h_{PLL}(\Delta\hat{\phi}_k)$.
- 8: Update carrier phase estimation (in rad):
 $\hat{\phi}_k = \hat{\phi}_{k-1} + 2\pi \hat{f}_{d_k} T_{int} + h_{PLL}(\Delta\hat{\phi}_k)$.
- 9: Compute DLL discriminator: $\Delta\hat{\tau}_k = \frac{\mathcal{E}_k - \mathcal{L}_k}{\mathcal{E}_k + \mathcal{L}_k}$, where:
 $\mathcal{E}_k = \sqrt{VE_{I_k}^2 + VE_{Q_k}^2 + E_{I_k}^2 + E_{Q_k}^2}$, and
 $\mathcal{L}_k = \sqrt{VL_{I_k}^2 + VL_{Q_k}^2 + L_{I_k}^2 + L_{Q_k}^2}$.
- 10: Filter $\Delta\hat{\tau}_k$ with a bandwidth BW_{DLL} : $h_{DLL}(\Delta\hat{\tau}_k)$.
- 11: Update code phase estimation (in samples):
 $N_{k+1} = \text{round}(S)$ and $\psi_{k+1} = S - N_{k+1}$, where
 $S = \frac{T_{int}f_{IN}}{\left(1 + \frac{\hat{f}_{d_k}}{f_c^{(Gal E1)}} \right)} + \psi_k + h_{DLL}(\Delta\hat{\tau}_k)f_{IN}$.
- 12: Code lock indicator:
 $CN0 = 10 \cdot \log_{10}(\hat{\rho}) + 10 \cdot \log_{10}\left(\frac{f_{IN}}{2}\right) - 10 \cdot \log_{10}(L_{PRN})$,
where: $\hat{\rho} = \frac{\hat{P}_s}{\hat{P}_n} = \frac{\hat{P}_s}{\hat{P}_{tot} - \hat{P}_s}$,
 $\hat{P}_s = \left(\frac{1}{\mathcal{U}} \sum_{i=0}^{\mathcal{U}-1} |\mathbf{P}_{I_{k-i}}| \right)^2$, and $\hat{P}_{tot} = \frac{1}{\mathcal{U}} \sum_{i=0}^{\mathcal{U}-1} |\mathbf{P}_{k-i}|^2$.
- 13: Phase lock indicator:
 $T_{carrier} = \frac{\left(\sum_{i=0}^{\mathcal{U}-1} \mathbf{P}_{I_{k-i}} \right)^2 - \left(\sum_{i=0}^{\mathcal{U}-1} \mathbf{P}_{Q_{k-i}} \right)^2}{\left(\sum_{i=0}^{\mathcal{U}-1} \mathbf{P}_{I_{k-i}} \right)^2 + \left(\sum_{i=0}^{\mathcal{U}-1} \mathbf{P}_{Q_{k-i}} \right)^2}$.
- 14: **if** $T_{carrier} < \mathcal{T}$ or $CN0 < CN0_{min}$ **then**
- 15: Increase lock fail counter $v \leftarrow v + 1$.
- 16: **else**
- 17: Decrease lock fail counter $v \leftarrow \max(v - 1, 0)$.
- 18: **end if**
- 19: **if** $v > \vartheta$ **then**
- 20: Notify the loss of lock to the control plane through the message queue.
- 21: **end if**
- 22: Output: P_k , accumulated carrier phase error $\hat{\phi}_k$, code phase $\mathcal{N} \leftarrow \mathcal{N} + N_k + \psi_k$, $CN0$.

Figura 7-5: Algoritmo 2, algoritmo de *Tracking* para señales Galileo E1 utilizado en GNSS-SDR.

Para la realización de los bloques de *tracking* se ha partido del diseño de los bloques existentes para la señal GPS L1/CA creados por el Dr. Javier Arribas Lázaro realizando ciertas modificaciones para poder adaptarlos a las señales Galileo E1.

Así, se ha creado un adaptador *GalileoE1DllVemlTracking* que hereda de la clase *TrackingInterface* ya presente en GNSS-SDR y que se encarga de adaptarla al bloque de GNU Radio *galileo_e1_dll_pll_veml_tracking_cc* diseñado por el autor a partir del bloque *gps_l1_ca_dll_pll_tracking_cc* y cuyas modificaciones consisten fundamentalmente en cambiar los discriminadores utilizados. Además, se han añadido funciones para calcular dichos discriminadores (como la función *float dll_nc_vemlp_normalized(gr_complex very_early_s1, gr_complex early_s1, gr_complex late_s1, gr_complex very_late_s1)*, que calcula el denominado *Nornalized Very Early Minus Late Power discriminator* comentado con anterioridad en este apartado) en la librería *tracking_discriminators* del receptor GNSS-SDR.

7.4 Resultados experimentales con señales reales

Los bloques del receptor descritos en este capítulo del proyecto se han probado utilizando señales reales transmitidas por los satélites en órbita de Galileo.

El hardware del cabezal de radiofrecuencia utilizado en este experimento es el mismo que el del apartado 6.2.1 y se constituye de una antena activa (Novatel GPS-600), que tiene una banda de paso a 3 dB en la banda E1 ($1575 \pm 8 \text{ MHz}$) e incorpora un amplificador de bajo nivel de ruido que proporciona 28 ± 3 dB de ganancia. La antena se conecta a la placa DBSRX [7-14], que es la que se encarga de la amplificación, bajada en frecuencia (downconversion) y filtrado de la señal. La placa DBSRX está conectada a la placa madre de la USRP v1 rev 4.5 [7-15], que realiza la conversión analógico digital, así como de otra bajada en frecuencia mucho más fina y un diezmado de la señal. Además incorpora la interfaz USB 2.0.



Figura 7-6: Equipo utilizado para realizar las pruebas de adquisición y *tracking* de las señales Galileo E1B y E1C.

El equipo anfitrión utilizado es un ordenador personal de sobremesa (PC) con un procesador Intel Core 2 Quad Q9400 a 2,66 GHz con 4 GB de RAM, utilizando Linux Ubuntu 12.04 de 32 bits como sistema operativo y GNU Radio 3.6.2.

En la Figura 7-6, puede verse una imagen del equipo completo utilizado.

El bloque *Signal Source* se ha configurado con el controlador UHD, estableciendo $f_c = 1575.420$ MHz y una amplificación de 60 dB (el controlador UHD ajusta los amplificadores programables internamente con el fin de obtener una baja figura de ruido y evitar la saturación del ADC), utilizando unos valores de diezmado de 8 y 16 (correspondientes a $f_{IN} = 8$ y $f_{IN} = 4$ Msps respectivamente), y se ha procedido a realizar una captura de muestras que se han guardado en el disco duro del ordenador anfitrión gracias la utilidad de *capture-to-file* del controlador UHD.

Posteriormente, se ha configurado el bloque *Signal Source* para leer desde el archivo correspondiente, evitando así restricciones de tiempo real y entregando un flujo de muestras complejas, en banda base, con partes real e imaginaria en tipo de datos *float* (IEEE-754 coma flotante de 32 bits). El bloque acondicionador de señal (*Signal Conditioner*) se ha desactivado (*pass-through*) y los canales del receptor se han configurado para utilizar el Algoritmo 1 de la Figura 7-4 para el bloque de Adquisición (utilizando la implementación *Galileo_E1_PCPS_Ambiguous_Acquisition*) y el Algoritmo 2 de la Figura 7-5 para el bloque de *Tracking* (utilizando la implementación *Galileo_E1_DLL_PLL_VEML_Tracking*).

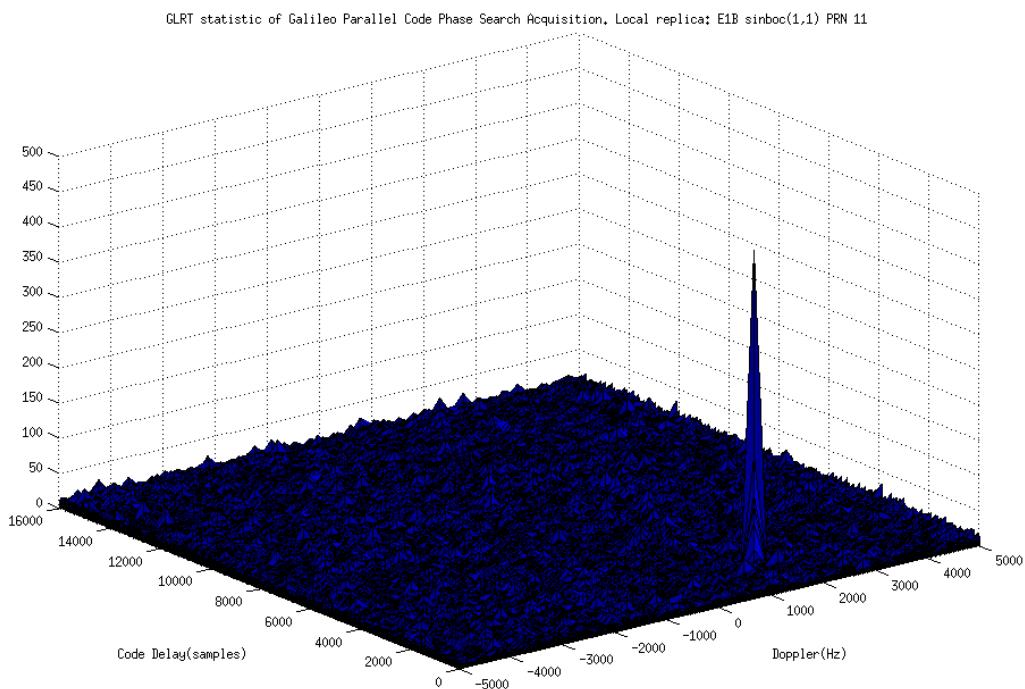


Figura 7-7: Estadístico GLRT para el algoritmo de adquisición *Parallel Code Phase Search* y una configuración con $f_{IN}=4$ Msps, un margen frecuencial que va de -5 kHz a 5 kHz con unos pasos de 250 Hz, utilizando como réplica local la señal sinBOC E1B correspondiente al satélite de PRN 11.

Los parámetros del Algoritmo 1 que se han utilizado para la adquisición son $T_{\text{int}} = 4 \text{ ms}$, $N = T_{\text{int}} f_{IN}$ (por lo tanto $N = 32000$ y $N = 16000$ muestras por período de código, que se corresponden a un ancho de banda del receptor de 4 MHz y 2 MHz, respectivamente), $\gamma = 50$, $f_{d_{\min}} = -5 \text{ kHz}$, $f_{d_{\max}} = 5 \text{ kHz}$ y $f_{step} = 250 \text{ Hz}$. La forma de onda de la señal local de referencia utilizada es $d_{EIB}^{(\sin BO C)}[n]$, definida en (7-10), para el satélite Galileo identificado con el PRN 11. La Figura 7-7 muestra el espacio de búsqueda calculado en los pasos 2-7 del Algoritmo 1. Se han obtenido resultados similares cuando se utiliza $d_{EIB}^{(CBOC)}[n]$, $d_{EIC}^{(\sin BO C)}[n]$ o $d_{EIC}^{(CBOC)}[n]$, para configuraciones con los dos anchos de banda comentados.

El algoritmo de *tracking* se ha configurado con $T_{\text{int}} = 4 \text{ ms}$, $\Gamma = 0.85$, $\varepsilon = 0.15 \cdot T_{c,E1B}$, $\varepsilon' = 0.6 \cdot T_{c,E1B}$, $v = 10$, $CN0_{\min} = 25 \text{ dB-Hz}$, $BW_{DLL} = 15 \text{ Hz}$, y $BW_{PLL} = 2 \text{ Hz}$ (ver descripción de los parámetros en el Algoritmo 2 de la Figura 7-5). La Figura 7-8 muestra la evolución en el tiempo de las correlaciones VE, E, P, L y VL obtenidas con dicha configuración. En ella puede observarse confirmar el correcto seguimiento de la señal, ya que las muestras del *Prompt* (P) están claramente por encima de las muestras del resto de correladores (Ve, E, L y VL).

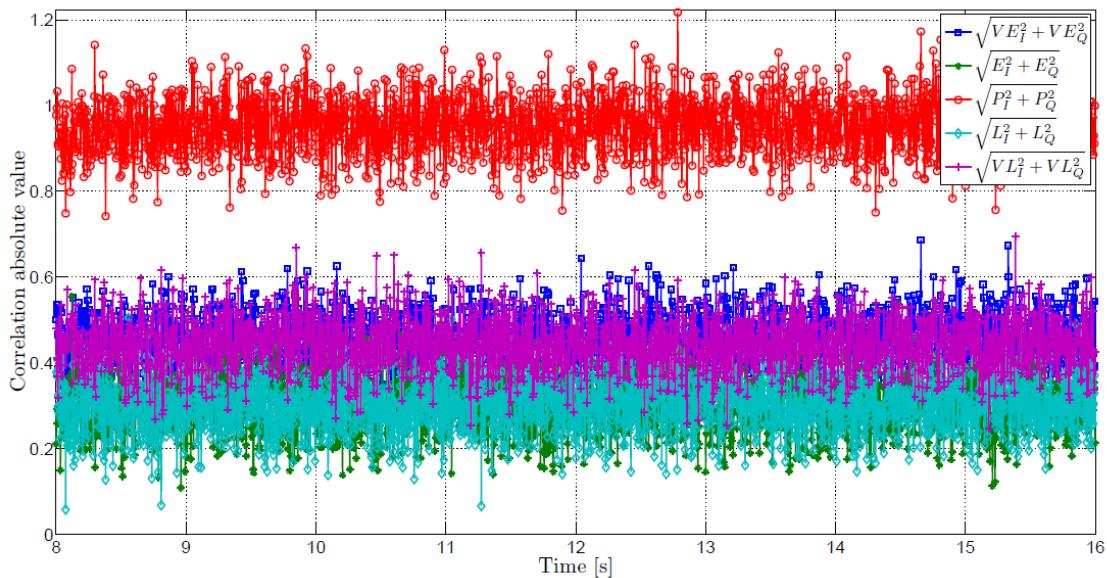


Figura 7-8: Evolución de las muestras de la correlación VE, E, P, L y VL en una ejecución del bloque de *tracking* GalileoE1DIIPII/VemITracking.

7.5 Conclusiones y futuros trabajos

En este capítulo se han presentado los bloques realizados para la adquisición y el *tracking* de señales de Galileo E1B y E1C. Se ha analizado el modelo de la señal Galileo E1, los aspectos principales de los bloques diseñados, y se han presentados los resultados experimentales obtenidos utilizando señales procedentes de los satélites en órbita de Galileo que validan el funcionamiento de dichos bloques.

Como futuros trabajos, en un nivel superior al Algoritmo 1 (o cualquier otro que proporcione las mismas salidas), se podrían integrar los resultados de más de un periodo de código consecutivo con el fin de aumentar la sensibilidad de la adquisición, entendida como la potencia de señal mínima a la que un receptor puede identificar correctamente la presencia de una señal de satélite en particular en la señal RF entrante dentro de un intervalo de tiempo de espera determinado.

7.6 Bibliografía

- [7-1] European Space Agency. Our Activities. Navigation. “Galileo fixes Europe’s position in history”, http://www.esa.int/Our_Activities/Navigation/Galileo_fixes_Europe_s_position_in_history, Comprobado: 25 de marzo de 2013.
- [7-2] “Google Summer of Code 2012”, <http://www.google-melange.com/gsoc/homepage/google/gsoc2012>. Comprobado: 25 de junio de 2013.
- [7-3] “GNU Radio Conference 2012”, <http://www.trondeau.com/gnu-radio-conference-2012/>. Comprobado: 25 de junio de 2013.
- [7-4] “Processing Galileo signals with GNSS-SDR”, http://www.youtube.com/watch?v=ajh2_xBCZSM. Comprobado: 25 de junio de 2013.
- [7-5] European Union, “European GNSS (Galileo) Open Service. Signal In Space Interface Control Document. Ref: OS SIS ICD, Issue 1.1”, septiembre de 2010.
- [7-6] E. Simona Lohan, “Limited bandwidths and correlation ambiguities: Do they co-exist in Galileo receivers,” Positioning, vol. 2, no. 1, pp. 14–21, Feb. 2011, DOI: 10.4236/pos.2011.21002.
- [7-7] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A Software-Defined GPS and Galileo Receiver. A Single-Frequency Approach, Applied and Numerical Harmonic Analysis*. Birkhäuser, Boston, MA, 2007.
- [7-8] S. M. Kay, *Fundamentals of statistical signal processing, Volume II: Detection theory*, Prentice Hall, Upper Saddle River, NJ, 1998.
- [7-9] M. Frigo y S. G. Johnson, “The design and implementation of FFTW3,” Proceedings of the IEEE, vol. 93, no. 2, pp. 216–231, 2005, Special issue on “Program Generation, Optimization, and Platform Adaptation”.
- [7-10] “VOLK,” <http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk>, Comprobado: 22 de mayo de 2013.
- [7-11] A. Jovanovic, C. Mongrédition, Y. Tawk, C. Botteron, and P. A. Farine, “Two-Step Galileo E1 CBOC Tracking Algorithm: When Reliability and Robustness Are Keys!,” International Journal of Navigation and Observation, 2012, Article ID 135401. Special issue on “Advances in Signal Tracking for GNSS Receivers: Theory and Implementation”. DOI:10.1155/2012/135401.
- [7-12] D. R. Pauluzzi and N. C. Beaulieu, “A comparison of SNR estimation techniques for the AWGN channel,” IEEE Transactions on Communications, vol. 48, no. 10, pp. 1681–1691, Oct. 2000.

[7-13] A. J. Van Dierendonck, "GPS Receivers," in Global Positioning System: Theory and Applications, B. W . Parkinson and J. J. Spilker, Jr., Eds., vol. I, chapter 8, pp. 329–407. American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 1995.

[7-14] Ettus Research, "Product Detail: DBSRX2 800-2300 MHz Rx", <http://www.ettus.com/product/details/DBSRX2>, Comprobado: 9 de junio de 2013.

[7-15] Ettus Research, "USRP1", <https://www.ettus.com/product/details/USRPPKG>, Comprobado: 9 de junio de 2013.

8 CONCLUSIONES Y FUTUROS TRABAJOS

Se ha presentado en este proyecto la contribución que ha hecho el autor al diseño de GNSS-SDR, un receptor GNSS de código abierto. Dicha contribución se centra en la creación de los bloques de acondicionamiento de señal (Signal Conditioner), de adquisición (fundamentalmente en el análisis estadístico del bloque), el diseño del bloque canal y la extensión de la funcionalidad del receptor para realizar la adquisición y *tracking* de señales Galileo E1. Además, el autor ha participado en las decisiones a más alto nivel del diseño del Plano de Procesado de Señal del receptor y en la elaboración de artículos científicos que han sido presentados en conferencias internacionales.

De esta forma, en el Capítulo 2 se ha realizado una descripción general del software GNSS-SDR.

En el capítulo 3, dedicado al acondicionador de señal, se ha podido ver que el *resampler* implementado (*direct_resampler_conditioner*) cumple con los requerimientos para los que fue diseñado. Así se ha visto que si bien es la opción de peor calidad ésta no dista mucho de la de algunos *resamplers* polifásicos, pero por el contrario, su tiempo de procesado es muy inferior al del resto de soluciones analizadas por lo que su viabilidad para el sistema GNSS-SDR con el fin de poder analizar señales GNSS en tiempo real es total.

En el capítulo 4 se ha presentado uno de los procesos iniciales más importantes en cualquier software GNSS, la adquisición.

Inicialmente se ha hecho una descripción de los diferentes algoritmos de adquisición existentes, haciendo hincapié en el implementado en el software GNSS-SDR, el *Parallel Code Phase Search Acquisition*.

A continuación se han desarrollado los cálculos matemáticos para la obtención del umbral de decisión del algoritmo, poniendo especial atención en la caracterización del estadístico de prueba, inicialmente para una celda del espacio de búsqueda y posteriormente extendiendo los resultados a todo el espacio.

Posteriormente se han presentado los bloques implementados para realizar la adquisición en el software GNSS-SDR, y que incluyen una interfaz común a todos los posibles algoritmos de adquisición (*AcquisitionInterface*), un bloque de GNU Radio que se encarga de realizar todo el procesado de señal del algoritmo de *Parallel Code Phase Search Acquisition* (*pcps_acquisition_cc*) y una clase adaptadora (*GpsL1CaPcpsAcquisition*) que se encarga de conectar las dos clases previas así como de realizar los cálculos propios de la señal utilizada (GPS L1 C/A en este caso).

Finalmente se han presentado los diferentes tipos de pruebas para verificar el comportamiento de los bloques implementados así como contrastar todos los cálculos teóricos realizados en los apartados previos de este capítulo y los resultados

obtenidos con estas pruebas. Dichas pruebas se agrupan en dos bloques bien diferenciados, por un lado los test unitarios y de integración, y por otro lado las medidas de calidad del algoritmo que necesitan de herramientas externas para ser analizadas (fundamentalmente Matlab).

Como futuros trabajos se propone desarrollar un modelo de señal más detallado que incluya la forma de la correlación del código con el objetivo de que en situaciones de CN_0 bajas y resoluciones altas el modelo sea más fidedigno y obtener unos valores teóricos más ajustados a los valores que se obtienen por simulación. Otra opción es desarrollar algoritmos de adquisición con periodos de integración mayores que permitan mejorar los resultados obtenidos.

Por otro lado comentar que una posible mejora, consistente en el desarrollo de un algoritmo de adquisición que incluya asistencia por internet está siendo implementada en el momento de la escritura de este proyecto por el Dr. Javier Arribas Lázaro.

En el capítulo 5 ha explicado el funcionamiento y la arquitectura del bloque canal implementado en el receptor GNSS-SDR. En particular se han explicado por separado su interfaz *ChannelInterface*, su adaptador *Channel*, y especialmente la máquina de estados finitos (FSM) *GpsL1CaChannelFsm*, que implementa toda la lógica para controlar la funcionalidad del canal y su interacción con los bloques internos de adquisición y *tracking*, así como con el *flowgraph* general de GNSS-SDR.

Como futuros trabajos se proponen implementaciones de canal más complejas que podrían desarrollar estrategias más sofisticadas, como las de los *vector tracking loops*, que implican a señales de más de un satélite, así como optimizaciones realizadas directamente en el dominio de la posición.

En el Capítulo 6 se han presentado dos experimentos con el fin de validar los bloques desarrollados por el autor en los capítulos anteriores del proyecto y ver su funcionalidad a la hora de formar parte de un receptor que funcione en tiempo real.

Para ello se ha realizado un primer experimento en el que se ha conseguido probar la utilidad del bloque *DirectResampler* explicado en el apartado 3.3.4 para conseguir posicionamiento en tiempo real en un ordenador de bajas prestaciones.

El segundo experimento ha consistido en probar el bloque *Frequency Translating FIR Filter* explicado en el apartado 3.4.1.2 y que ha posibilitado utilizar el cabezal de radiofrecuencia USB SiGe GN3S Sampler v2 eliminando la frecuencia intermedia (FI) que tiene el cabezal y consiguiendo un nuevo posicionamiento en tiempo real.

Para futuros trabajos comentar que el cuello de botella del receptor sigue estando en el bloque de adquisición y se propone mejorar su rendimiento utilizando intensivamente la librería Volk así como la posibilidad de desviar cierto volumen de carga computacional del proceso de adquisición a la GPU de la tarjeta gráfica del ordenador anfitrión cuando sea posible. Dichas mejoras se llevarán a cabo en el futuro programa *Google Summer of Code 2013* (GSOC 2013), un programa de becas para estudiantes patrocinado por Google en el que GNSS-SDR ha sido incluida como organización mentora.

Por último, en el Capítulo 7, se han presentado los bloques realizados para la adquisición y el *tracking* de señales de Galileo E1B y E1C. Se ha analizado el modelo de la señal Galileo E1, los aspectos principales de los bloques diseñados, y se han

presentados los resultados experimentales obtenidos utilizando señales procedentes de los satélites en órbita de Galileo que validan el funcionamiento de dichos bloques.

Como futuros trabajos, en un nivel superior al Algoritmo 1 de la adquisición presentado en la Figura 7-4 (o cualquier otro que proporcione las mismas salidas), se podrían integrar los resultados de más de un periodo de código consecutivo con el fin de aumentar la sensibilidad de la adquisición, entendida como la potencia de señal mínima a la que un receptor puede identificar correctamente la presencia de una señal de satélite en particular en la señal RF entrante dentro de un intervalo de tiempo de espera determinado.

Se detalla a continuación una lista de los artículos publicados en los que el autor ha participado en su elaboración gracias al trabajo desarrollado en este proyecto:

C. Fernández-Prades, J. Arribas, L. Esteve-Elfau, D. Pubill, P. Closas, “An Open Source Galileo E1 Software Receiver”, in Proceedings of the 6th ESA Workshop on Satellite Navigation Technologies (NAVITEC 2012), 5-7 December 2012, ESTEC, Noordwijk (The Netherlands).

L. Esteve, C. Fernández-Prades, J. Arribas, P. Closas, “Processing Galileo signals with GNSS-SDR”, Presentation at GNU Radio Conference 2012, 24-27 September, Atlanta, GA (USA).

C. Fernández-Prades, J. Arribas, P. Closas, C. Avilés, L. Esteve, “GNSS-SDR: an open source tool for researchers and developers”, in Proceedings of the ION GNSS Conference 2011, September 19-23, 2011, Portland, Oregon (USA).

C. Fernández-Prades, C. Avilés, L. Esteve, J. Arribas, P. Closas, “Design patterns for GNSS software receivers”, in Proceedings of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010), 8-10 December 2010, Noordwijk (The Netherlands).