Proyecto Fin de Grado
Ingeniería de Telecomunicación

Diseño de sistema de comunicaciones usando software defined radio

Autor: Antonio Padilla Esquivel
Tutor: Juan José Murillo Fuentes

Dep. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Grado
Ingeniería de Telecomunicación

Diseño de sistema de comunicaciones usando software defined radio

Autor:
Antonio Padilla Esquivel

Tutor:
Juan José Murillo Fuentes
Profesor titular

Dep. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Grado: Diseño de sistema de comunicaciones usando software defined radio

Autor: Antonio Padilla Esquivel

Tutor: Juan José Murillo Fuentes

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2015

Agradecimientos

Primero me gustaría comenzar dando mis infinitas gracias a Juan José Murillo, mi tutor del proyecto, por su paciencia, su dedicación y su apoyo tanto en el ámbito académico como en el personal. Espero que sus lecciones me ayuden a convertirme en un buen ingeniero en el futuro.

Gracias a mis padres, por sus noches en vela, su cariño, sus frases de consuelo y para qué negarlo, también gracias por sus broncas. Cada palabra que dicen es una lección de vida que todo el mundo debería escuchar. Faltan papel y palabras en el mundo para daros las gracias como realmente merecéis.

Gracias a mi hermano, una de las personas más fuertes y con mayor espíritu de superación que conozco, capaz de sacarme de mi mundo y devolverme a la realidad cada vez que lo necesito.

Gracias al resto de mi familia de la que me enorgullezco de formar partes y que ha estado apoyándome en todos los momentos importantes de mi vida.

Gracias a mis amigos por estar cuando se les necesita y por saber hacerme sonreír en cualquier circunstancia. Sin ellos la vida sería mucho más aburrida.

Gracias a mis profesores, que siempre tienen algo que enseñar y están dispuestos a ello.

En definitiva, gracias a todo aquel que se haya cruzado alguna vez en mi camino, pues en mayor o menor medida han hecho que sea la persona que está hoy escribiendo estas palabras.

Antonio Padilla Esquivel

Sevilla, 2015

Resumen

Este trabajo se centra en el estudio de la sincronización de símbolo a nivel de bit en una comunicación basada en Software Defined Radio utilizando como equipo transmisor y receptor el USRP1 de Ettus Research. Tras detectar a través de mediciones experimentales un error en el sincronismo de símbolo en el sistema DQPSK propuesto en las referencias se procede a caracterizar dicho error. Una vez determinado el foco del problema se estudian diferentes métodos para solucionarlo y se implementa uno basado en la apertura del diagrama de ojo.

Este trabajo incluye las mediciones realizadas, tanto por cable como por antena, así como los códigos para realizar la detección de sincronismo y la caracterización del error.

Abstract

This work focuses on the study of bit -level symbol synchronization in a communication based on Software Defined Radio using the USRP1 of Ettus Research as transmitter and receiver. After detecting through experimental measurements an error in the symbol synchronization in the DQPSK system proposed in references we proceed to characterize said error. Once the focus of the problem is found different methods to solve it are studied and it is implemented one based on the opening of the eye diagram.
This work includes measurements made by antenna and cable as well as codes for synchronization detection and characterization of error.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En el departamento de Teoría de la Señal y Comunicaciones se lleva un tiempo trabajando en el diseño de sistemas de comunicaciones basados en Software Defined Radio o SDR. Estos dispositivos permiten programar el transmisor y el receptor mediante software hasta su etapa de frecuencia intermedia, trabajando con altas tasas de muestreo gracias a los convertidores digitales de subida y bajada que contienen. En el transmisor la señal se genera en banda base, se sobremuestrea y se pasa a IF donde se convertirá a analógica, para posteriormente subirla hasta frecuencia RF. En el receptor se realiza el proceso inverso, donde se baja la señal recibida de RF a IF y se digitaliza para posteriormente detectar los bits. Esta programación de transmisor y receptor es completa, para pasar desde/hacia los bits de información hacia/desde la señal muestreada a frecuencia intermedia. En el receptor, será necesario programar tanto las labores de demodulación como de detección, lo que obliga a incluir un proceso de sincronismo tanto de frecuencia y fase (recuperación de portadora) como de tiempo.

1.1 Objetivo

El objetivo inicial de este proyecto era incluir un módulo de corrección de frecuencia en un sistema de comunicaciones basado en el trabajo anterior de otros alumnos del departamento. Este sistema implementaba una codificación diferencial DQPSK para evitar tener que corregir constantemente el error de frecuencia y se pretendía convertirlo en un sistema de modulación coherente QPSK.

Sin embargo, al intentar implementar este bloque, se observó que el sincronismo de símbolo no era el adecuado por lo que se pasó a analizar y caracterizar el problema e intentar aportar una solución con un nuevo módulo de sincronismo temporal.

1.2 Secciones

En el Capítulo 2 profundizaremos en los conceptos básicos de la tecnología SDR, así como en el hardware y software empleados en el desarrollo de este trabajo, el periférico USRP1 de Ettus Research y el toolbox de Matlab SDR4All.

Será en el Capítulo 3 donde se resumirá la funcionalidad de los scripts en Matlab propuestos en la fuente principal de este trabajo que componen el sistema DQPSK.

En el Capítulo 4 se presentan los valores experimentales obtenidos al ejecutar el sistema DQPSK, así como una caracterización de los errores de muestreo surgidos durante las pruebas.

En el Capítulo 5 se estudian dos posibles métodos de sincronismo de símbolo y se aplica un tercero, comprobándose si el error ha sido subsanado.

Finalmente en los Anexos se procede a facilitar un tutorial de instalación del software empleado, los códigos implementados y una breve descripción de los sistemas DQPSK.

2 CONCEPTOS BÁSICOS

"A software radio is a radio whose channel modulation waveforms are defined in software."

- Joseph Mitola -

En este capítulo se tratarán los conceptos esenciales para entender el trabajo posterior, por lo que se comenzará haciendo una introducción sobre el concepto de Software Defined Radio, su definición, una contextualización histórica y una breve descripción de su funcionamiento.

Lo siguiente será describir el dispositivo hardware que usaremos durante todo el proyecto, el USRP1, sus elementos internos, y otros componentes empleados en las simulaciones.

Finalmente se hablará de la herramienta empleada para la parte de procesado digital de señal que todo sistema SDR[1] necesita. En nuestro caso se ha elegido el toolbox de Matlab SDR4All, del que se describirán las funciones que incorpora.

2.1 SDR

A pesar de no existir un estándar respecto a los dispositivos SDR, el Wireless Innovation Forum en colaboración con el grupo P19001 del IEEE[2], establecen una definición y unos esquemas comunes para estas tecnologías:

"Radio en la cual alguna o todas las funciones de la capa física son definidas mediante software." [1]

Es decir, características como la frecuencia de portadora o la modulación empleada son establecidas de manera flexible a través de software.

El concepto de SDR tiene como objetivo aumentar la flexibilidad en los sistemas de comunicaciones, en contraste con los sistemas tradicionales basados en hardware.

La variación en el comportamiento de un elemento hardware a través de software, siguiendo una arquitectura común, permite una reducción drástica de costes de producción y mantenimiento, así como una inversión mínima a la hora de añadir nuevas prestación a los sistemas, lo que se traduce en un menor coste que beneficia tanto a proveedores de servicio como al usuario final que pretende tener acceso a comunicaciones inalámbricas de manera eficiente y a un precio económico [1].

[1] Software Defined Radio
[2] Institute of Electrical and Electronics Engineers

Además de por todo lo descrito, el auge en la tecnología SDR se debe a la creciente implantación de técnicas de gestión eficiente del espectro, radio adaptativa y cognitiva, donde se dota a los equipos de cierta inteligencia para fijar de manera dinámica los parámetros de operación según su situación, lo que resulta más sencillo a través de software.

2.1.1 Contexto histórico

SPEAKeasy es el primer proyecto de envergadura que sienta las bases de lo que hoy día consideramos como radio definida por software. Surge en 1991 en el seno del departamento de defensa de los Estados Unidos, concretamente en la agencia DARPA[3], con el propósito de aumentar la interoperabilidad entre equipos posibilitando la compatibilidad de un solo terminal con 10 protocolos de comunicación diferentes, gracias a su arquitectura modular enfocado al software [2] [3].

A partir de esta tecnología, y con la intención de impulsar el desarrollo de aplicaciones de propósito tanto civil como militar de los sistemas SDR se funda en 1996 el Wireless Innovation Forum (primero conocido como SDR Forum), como una "sociedad de beneficio mutuo" sin ánimo de lucro, un foro de colaboración entre empresas dedicada a abogar por el uso innovador de espectro y avance de las tecnologías de radio que soportan comunicaciones esenciales o críticas en todo el mundo, según como ellos mismos se definen [1].

Otro hito destacable en la historia del SDR es la creación en el año 2001 del Proyecto de GNU GNU Radio, software de desarrollo de herramientas de código libre y abierto que proporciona bloques de procesamiento de señales para implementar software radio. Este puede ser usado con hardware de RF[4] externo de bajo costo para crear radios definidos por software o sin hardware utilizándolo como entorno de simulación [4].

Ya en 2009 se distribuye una interfaz gráfica de usuario para desarrollar aplicaciones GNU Radio en código Python, conocida como GNU Radio Companion.

En 2005 la empresa Ettus Research produce por primera vez un periférico de la familia USRP, el USRP1 con el objetivo de crear una plataforma de software defined radio de bajo coste, potenciando el uso de esta tecnología [5].

Gracias a su capacidad para flexibilizar las comunicaciones, Alcatel investiga el diseño de estaciones base de telefonía móvil con tecnología SDR que permitan implementar más de una tecnología de acceso radio. Con ello pretende conseguir hacerlas compatibles con diferentes frecuencias y standars, reduciendo drásticamente los costes [6].

Hacia el año 2010 surgen los RTL-SDR, una familia de dispositivos receptores que emplean un demodulador DVB-T, el RTL2832U, al que se le detectaron capacidades para su uso como receptor SDR de banda ancha, Su importancia respecto a otros productos de prestaciones similares se basa en su reducido precio (20$) lo que permite acercar la tecnología SDR al usuario [7] [8].

Se continua la tendencia iniciada con el USRP1 de tener transceptores basados SDR a bajo coste que permitan trabajar con multitud de tecnologías radio, siendo en parte o completamente de código abierto para aumentar la compatibilidad y facilitar que el usuario desarrolle las aplicaciones que crea oportunas. Algunos ejemplos de estos transceptores son el HackRF [9], BladeRF [10] o los nuevos modelos de USRP de Ettus Research [11].

Actualmente las pruebas y diseño de prototipos de las redes móviles 5G se están realizando mediante tecnología SDR (como el USRP) gracias a su capacidad para trabajar en múltiples bandas y protocolos, lo que aumenta la velocidad a la que se pueden efectuar dichas pruebas. Un ejemplo lo tenemos en Samsung, que trabaja en un sistema MIMO tridimensional (FD-MIMO) y las pruebas se han realizado con USRPs RIO de la empresa National Instrument como receptores, que facilita operar en las bandas necesarias para realizar las simulaciones. [12] [13].

[3] Defence Advanced Research Projects Agency
[4] Radio Frecuency

2.1.2 Funcionamiento

El esquema común de los dispositivos SDR incluye Field Programmable Gate Array (FPGA), procesadores de señal digital (DSP) , procesadores de propósito general (GPP), System on a Chip(SoC) u otros procesadores programables específicos de la aplicación [14] y se encuentra dividido en tres secciones principales:

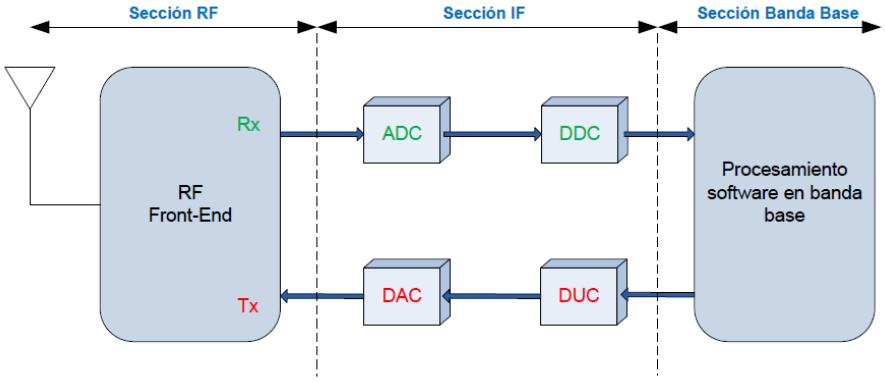

Figura 2-1. Diagrama de bloques de SDR [2]

La llamada cabecera de RF donde se realiza en paso de RF a frecuencia intermedia o viceversa, según se trate de transmisión o recepción. En el caso de transmisión este bloque también se encarga de amplificar la señal a transmitir. Puede darse el caso en que esta frecuencia intermedia sea cero, conocido esto como zero-IF[5].

La sección IF tiene como propósito el paso de frecuencia intermedia a banda base. Los ADC[6] y DAC[7] realizan la conversión analógico-digital o digital-analógica y los DDC[8] y DUC[9] son los encargados de cambiar la tasa de muestras entre la sección IF y la de banda base.

La sección en banda base se implementa mediante software y es donde se hará todo el procesamiento digital de señal [2].

2.2 USRP1

El USRP1 o Universal Software Radio Peripheral es una hardware diseñado para aplicaciones software radio con la función de llevar a banda base, mediante la sección de IF, los datos recibidos en la cabecera de RF.

Está elaborado por la compañía Ettus Research, con el objetivo de proporcionar una plataforma SDR a coste reducido, donde su mercado principal es el de laboratorios de investigación, universidades y aficionados.

El funcionamiento interno de este dispositivo ha sido ampliamente descrito en el Capítulo 2 de [2].

[5] Intermediate Frecuency
[6] Analog-to-Digital Conversion
[7] Digital-to-Analog Conversion
[8] Digital Down Converter
[9] Digital Up Converter

Figura 2-2. USRP1 de Ettus Research

2.2.1 Partes

Su arquitectura presenta dos elementos principales y está concebida de manera modular.

- Una placa base donde está implementado el FPGA y los ADC/DDC y DAC/DUC.

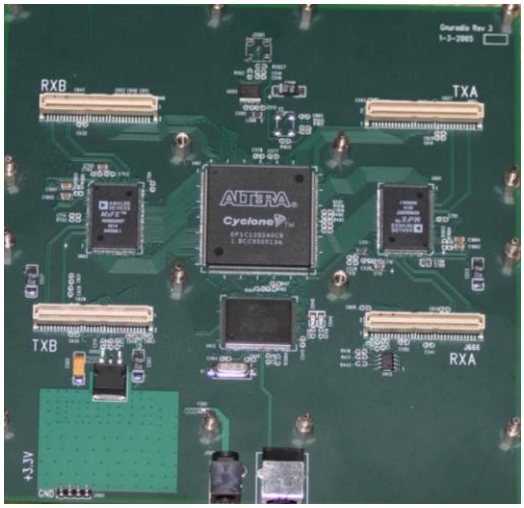

Figura 2-3. Placa base o motherboard

- Tarjetas secundarias: El USRP presenta dos ranuras para las tarjetas secundarias, A y B, para cambiar la funcionalidad de su cabecera de RF.

Figura 2-4. Tarjeta secundaria RFX2400 [15]

En nuestro caso emplearemos las tarjetas de la familia RFX2400 daughterboards: tarjetas transceptoras usadas en la banda ISM[10] de 2,4 GHz.

Estas presentan dos conectores, uno para transmisión y recepción Tx/Rx y otro solo de recepción Rx2.

Su ganancia, configurable solo en recepción, oscila entre los 0 y los 70 dB.

- Como dispositivos de transmisión empleamos antenas VERT2400 de doble banda 2400-2480 MHz y 4,9-5,9 GHz; y cables de conectores SMA.

Figura 2-5. Antenas y cables [16] [17]

- El USRP interactúa a su vez con la sección de BB, el ordenador, a través de un USB de conectores tipo A/B.

Figura 2-6. Cable USB [18]

2.3 SDR4All

Dentro de la amplia familia de herramientas disponibles para la programación software de los USRP se ha elegido el toolbox de Matlab SDR4All.

Desde que se elaboró el libro [2] hasta ahora, este toolbox ha dejado de tener soporte y su página web de consulta y descarga ha desaparecido.

Este toolbox está diseñado específicamente para el USRP1, y más concretamente para sus tarjetas secundarias RFX2400, además de para una versión de Windows de 32 bits.

Para conocer en mayor profundidad como instalar este toolbox consultar el Anexo B.

[10] Industrial, Scientific and Medical

2.3.1 Partes

SDR4All está compuesto de dos partes principales para poder interactuar con los USRP.

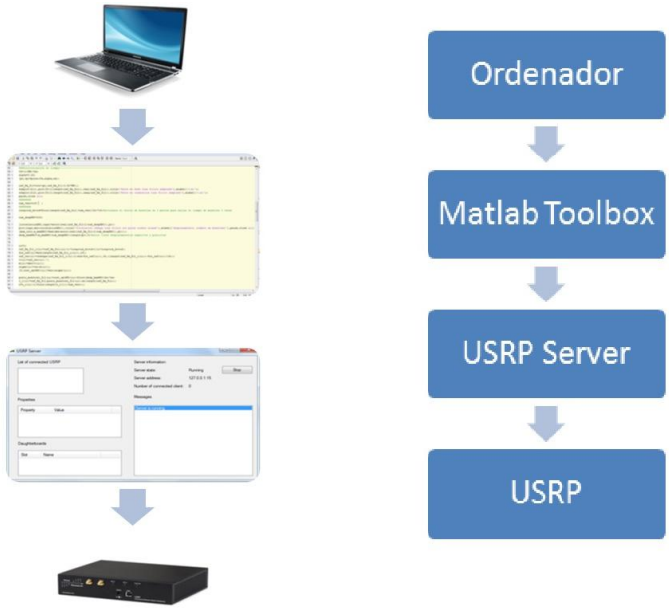

Figura 2-7. Diagrama funcionamiento SDR4All

- Una serie de funciones en Matlab para fijar los parámetros de transmisión y recepción:
  - Puerto=SDR4All_Connect(num,slot,tipo): genera un puerto "Puerto" para el USRP número 'num' en el slot 'slot' (en nuestro caso A o B) y seleccionando su función "tipo", es decir, si es un transmisor ("Tx") o un receptor ("Rx").

    Ej: `sock=SDR4All_Connect(0,'SlotA','TX')`

  - SDR4All_SetGain(Puerto,Gain): establece la ganancia 'Gain' [dB] que empleará el USRP en su transmisión o recepción.

    Ej: `SDR4All_SetGain(sock,45)`

  - SDR4All_SetFreq(Puerto,freq): fija la frecuencia del oscilador de RF en 'freq' [Hz]

    Ej: `SDR4All_SetFreq(sock,2.5e9)`

  - SDR4All_SetInterpRate(Puerto,interp): selecciona el factor de interpolación.

    Ej: `SDR4All_SetInterpRate(sock,256)`

  - SDR4All_SetDecimRate(Puerto,decim): selecciona el factor de diezmado.

    Ej: `SDR4All_SetDecimRate(sock,128)`

- SDR4All_SendData(Puerto,dato): transmite los datos contenidos en la variable 'dato'.

  Ej: `SDR4All_SendData(sock,sig_Tx)`

- [dato]=SDR4All_GetData(Puerto,time): recibe los datos y los almacena en la variable 'dato', durante un número de muestras 'time'.

  Ej: `[sig_Rx]=SDR4All_GetData(sock,2000)`

- El ejecutable USRP Server cuya interfaz se aprecia en la Figura 2-8, que permite la interconexión del USRP y el ordenador a través de los drivers que incorpora, a diferencia de otros toolbox existentes, que necesitan de software externo para esta función. Realmente es el elemento que hace de puente entre el script de Matlab y el USRP y por tanto deberá estar iniciado cuando se pretendan usar las funciones anteriormente descritas.

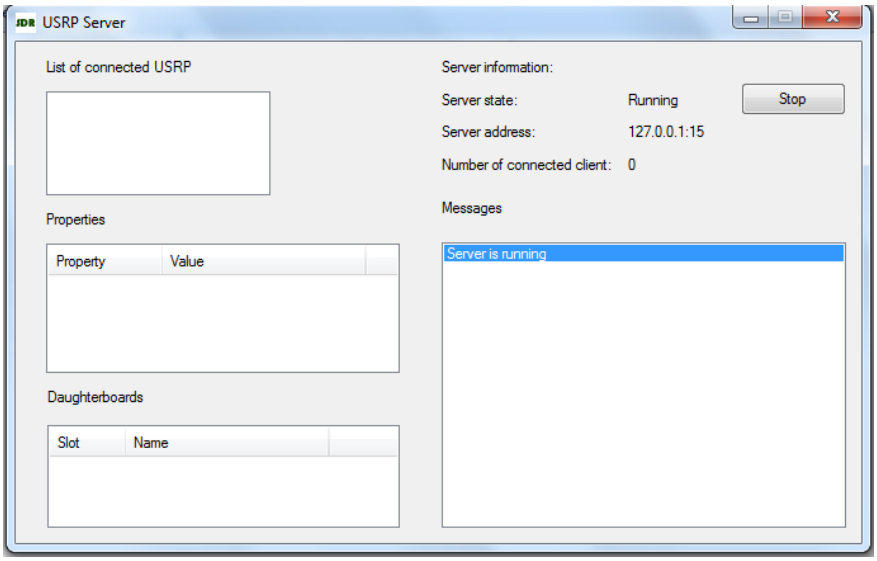

Figura 2-8. Interfaz USRP Server

3 SISTEMA DQPSK PROPUESTO

"El USRP1, o un SDR similar, es una excelente y, en nuestra opinión, indispensable herramienta para enseñar e investigar en ingeniería radio."

- Iván Pinar -

Este capítulo pretende aclarar el funcionamiento de los scripts de Matlab propuestos en [2] sobre un sistema de comunicación DQPSK, que se ha usado como punto de partida en este trabajo.

Estos scripts son:

Transmisor_DQPSK.m que simulará la parte transmisora del sistema, realizando las labores de codificador de canal, modulación y filtrado, generación de la trama a enviar y transmisión de datos al USRP transmisor.

Receptor_DQPSK.m que contiene un bloque para la recepción de datos, el filtro adaptado, un bloque de sincronismo en frecuencia y tiempo, la demodulación y decodificación de canal y finalmente estima la BER.

Pulso.m: que genera un pulso raíz de coseno alzado usado para filtrar tanto el transmisor como en el receptor.

Los USRP empleados para obtener los datos ilustrados en las imágenes se comunicaban entre sí por vía aire a través de una antena VERT2400.

3.1 Scripts propuestos

3.1.1 Transmisor_DQPSK

Primero se cargan los datos almacenados en el fichero datos.b a la variable bits_info.

Estos bits se pasan por un código convolucional de tasa ½ que añadirá redundancia, con lo que se dotará a la información de mayor robustez frente a errores.

Una vez hecho esto, se procede a codificar los bits de dos en dos siguiendo un esquema de modulación DQPSK, con estado inicial $\pi/4$. La modulación DQPSK queda aclarada en el Anexo C, junto con las ecuaciones del pulso RRC.

Lo siguiente es pasar la señal por el filtro conformador del transmisor. En nuestro caso se trata de un pulso raíz de coseno alzado implementado en la función pulso.m cuyos valores están en la Tabla 3–1.

Tabla 3–1. Valores Pulso Raíz Coseno Alzado

Dato	Valor
Tasa de muestreo f_m	500 kS/s
Muestras por símbolo	16
Tiempo de símbolo T_s	32 µs
Tasa de símbolo R_s	31.25 kbaudios
Tasa de bit R_b	62.5 kbps
Factor de roll-off α	0.22
Ancho banda BW_{Tx}	38.125 kHz

Para comprobar las prestaciones de los bloques desarrollados se emplea una estructura de trama sencilla e ineficiente, compuesta por un tramo de continua (0.2 s), que se empleará en el receptor para el sincronismo en frecuencia, un tramo de ceros (0.1 s), la señal de información (0.8 s) y finalmente otro tramo de ceros (0.1 s) con una duración total de 1.2 s. Esta trama se repite 5 veces para facilitar la recepción, pues con recibir una de las cinco tramas será suficiente. Ver Figura 3-1

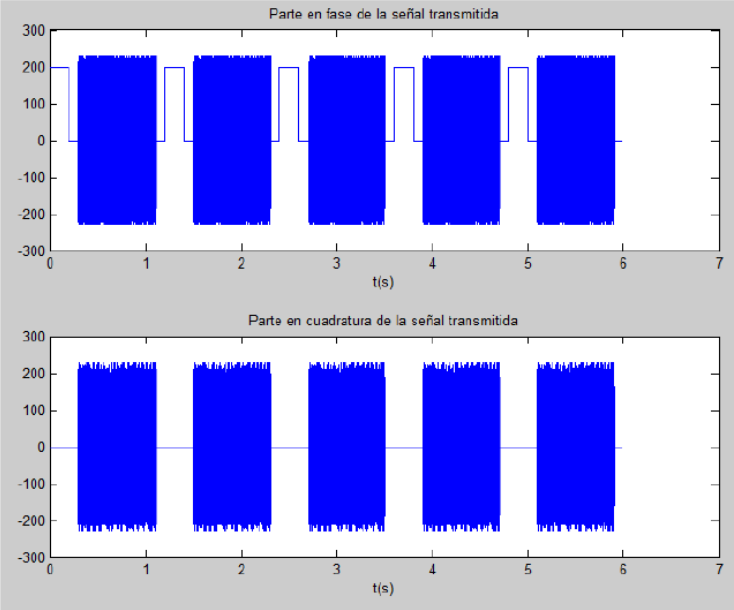

Figura 3-1. Secuencia transmitida [2]

Tras todo este proceso se emplean las funciones facilitadas por SDR4All, descritas en la Sección 2.3 para establecer los parámetros de transmisión que se presentan en la Tabla 3–2, debiendo iniciarse USRP Server antes de ejecutarlas.

Tabla 3–2. Parámetros de transmisión

Parámetro	Valor
Ganancia	20 dB
Frecuencia	2500 MHz
Interpolado	256

En el caso del interpolado es necesario aclarar que como tenemos una tasa de muestreo de 500 kS/s en la interfaz USB, para conseguir los 128MS/s necesarios en el DAC habrá que emplear un factor de interpolado de 256.

Se recomienda que tras cada envío o recepción de datos se reinicie el ejecutable USRP Server, pues puede incurrir en errores en la siguiente comunicación si no se hace.

3.1.2 Receptor_DQPSK

Del lado receptor usamos el script Receptor_DQPSK.m con los parámetros en recepción del USRP Server descritos en la Tabla 3–3, almacenando los datos recibidos en la variable sig_Rx.

Tabla 3–3. Parámetros de recepción

Parámetro	Valor
Ganancia	Variable
Frecuencia	2500MHz
Diezmado	128

En el caso del diezmado, como el ADC tiene una tasa de 64 MS/s, y se necesitan 500kS/s en la interfaz USB será necesario un factor de diezmado de 128.

En este caso la recepción se realiza durante 10 segundos para contribuir a la llegada de la información en su totalidad como se muestra en la Figura 3-1.

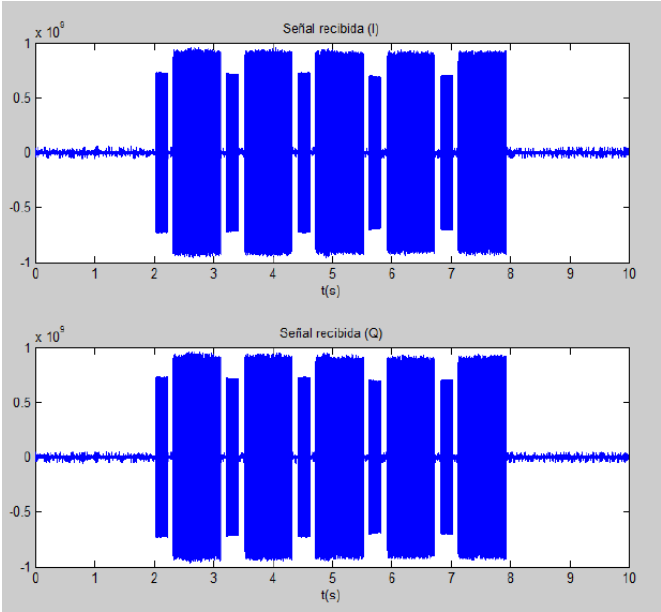

Figura 3-2. Secuencia recibida [2]

Para saber dónde comienza la trama y separarla del ruido inicial, se calcula el valor RMS de la señal recibida. Allí donde se supere por primera vez este valor será el inicio de nuestra ráfaga, que sabemos que tiene una duración de 1.2 s y podremos separarla del resto de la trama.

Una vez que tenemos nuestra ráfaga por separado será necesario hacer un ajuste en frecuencia, debido a que los osciladores locales de los USRP no son perfectos lo que provoca que la secuencia esté desplazada en frecuencia con respecto a banda base.

Este offset en frecuencia se encuentra también en el tramo de continua en forma de una exponencial compleja $e^{2 \cdot j \cdot \pi \cdot \Delta f}$, donde Δf es exactamente esta desviación.

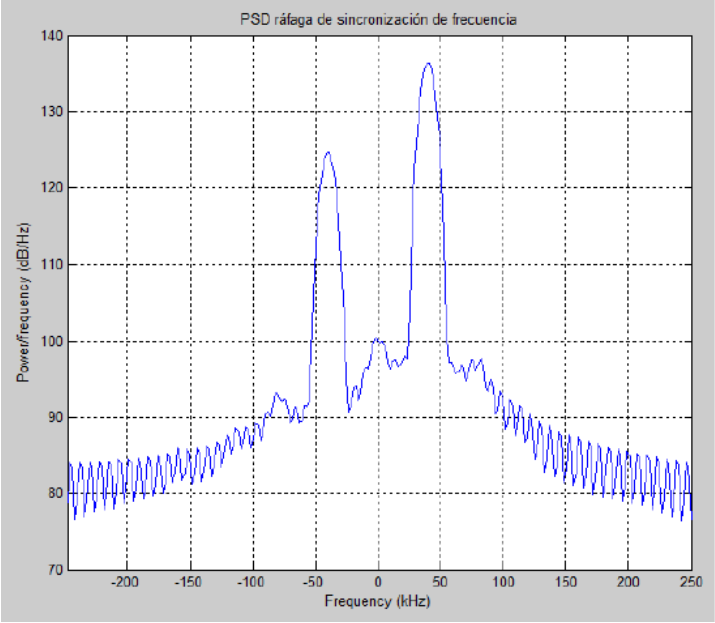

Figura 3-3. Espectro secuencia sincronismo [2]

La desviación Δf que se aprecia en la Figura 3-3 se calcula buscando los puntos máximos en el espectro de la ráfaga de sincronización y se usa para multiplicar la ráfaga de datos por $e^{-2 \cdot j \cdot \pi \cdot \Delta f}$ desplazándola en frecuencia y haciendo un ajuste de ésta.

La pequeña desviación en frecuencia restante, fruto de la variabilidad con el tiempo, así como la desviación en fase, serán soportadas por la modulación DQPSK gracias a su carácter diferencial.

Después comienza la sincronización temporal, en la cual muestreamos la señal en sus 16 instantes de tiempo posibles, los almacenamos en 16 vectores diferentes y calculamos la varianza de cada uno de ellos. Aquel con menor varianza será el óptimo.

Aunque en el caso del código propuesto este cálculo se realiza dos veces, una antes de pasar la señal por el filtro RRC del receptor y otra después, se observa que solo es necesario este paso en el segundo de los casos.

Se correla la señal con el pulso RRC para determinar el instante de inicio de la información, obviando el retraso inicial. Éste será el punto donde la señal de información más se asemeje al pulso raíz de coseno alzado, y por tanto donde termina el retraso.

Ya solo resta demodular y calcular la BER antes (Ver Figura 3-4) y después del decodificador de canal.

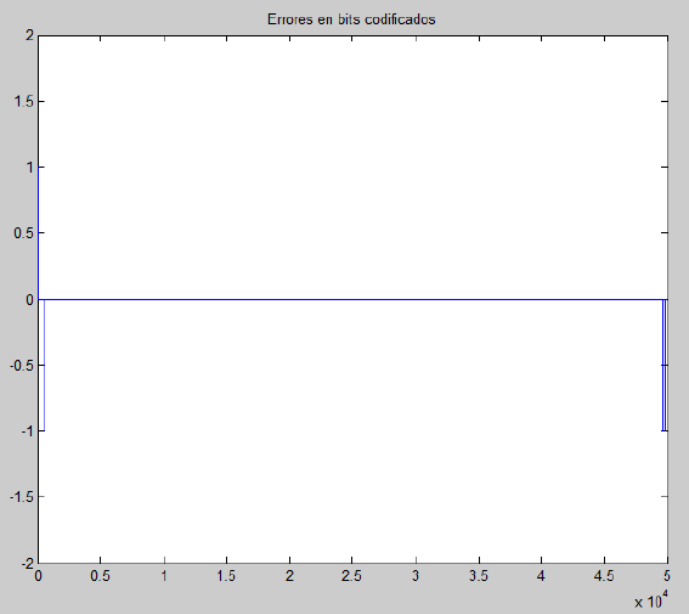

Figura 3-4. Errores en bits codificados [2]

4 APLICACIÓN REAL DEL SISTEMA DQPSK

"La gente ha utilizado USRPs para construir estaciones base de telefonía móvil para proporcionar la cobertura de telefonía celular en los países subdesarrollados, en las islas en el Pacífico y en África, donde no hay otra infraestructura."

- Matt Ettus -

En este capítulo se pretende comprobar la funcionalidad real del sistema DQPSK desarrollado mediante los scripts proporcionados en [2] descritos en el Capítulo 3.

Para ello enviaremos el fichero de datos datos_b desde el USRP transmisor hasta el receptor, tanto por antena como por cable, encontrándose durante dicha prueba diferencias significativas con respecto a lo esperado.

Esto nos lleva a determinar la existencia de un error de carácter desconocido a la hora de recibir nuestro mensaje y tras varias simulaciones nos llevará a concluir que existía un error en el sincronismo de símbolo.

Para confirmar nuestras hipótesis se envía además una secuencia conocida donde podremos observar de manera óptima este error.

4.1 Materiales necesarios y esquema de montaje

Para la realización de las pruebas se han usado los siguientes materiales:

- Dos ordenadores con Windows 32 bits, Matlab con el toolbox SDR4All y USRP Server.
- Dos USRP1 con tarjetas secundarias RFX2400.
- Cable con conectores SMA-SMA y Antenas VERT2400.
- Script Transmisor_DQPSK.m, Receptor_DQPSK.m y pulso.m [2].
- Scripts almacenaDatos.m.
- Script Receptor_SecuenciaConocida.m: para la Sección 4.3.
- Ficheros de datos
    - sigRX250070dB [2]: fichero propuesto con ganancia de recepción 45 dB.

- datos_b [2]: se usará tanto para la transmisión como para compararlo con los datos recibidos y calcular la BER.
- datos_c [2]: se usará para compararlo con lo recibido y calcular la BER antes del decodificador de canal.
- sigRXCable45dB: fichero obtenido experimentalmente con conexión entre USRP por cable y ganancia de recepción 45 dB.
- sigRXAire50dB: fichero obtenido experimentalmente con conexión entre USRP por antena y ganancia de recepción 50 dB.

El ordenador que ejecutará el script Transmisor_DQPSK.m se conectará al USRP con tarjeta RFX2400 en el slot A, debido a la configuración de las funciones de SDR4All. El ordenador que almacenará los datos recibidos con almacenaDatos.m se conectará al USRP con tarjeta RFX2400 en el slot B y entre sí los USRP se comunican conectando los elementos de transmisión (antena o cable) en los conectores Tx/Rx de su slot correspondiente.

Figura 4-1. Esquema de montaje

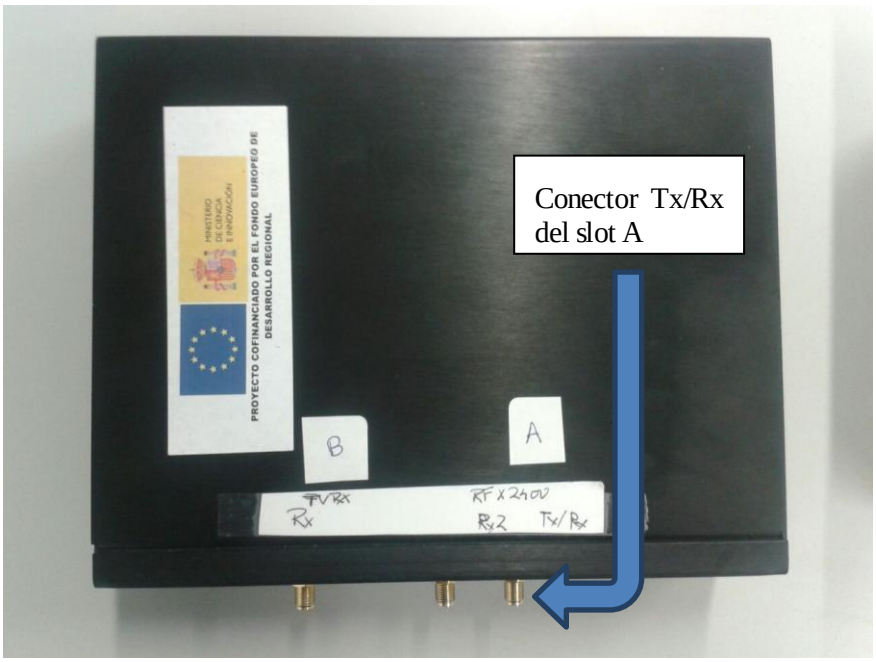

Figura 4-2. USRP transmisor

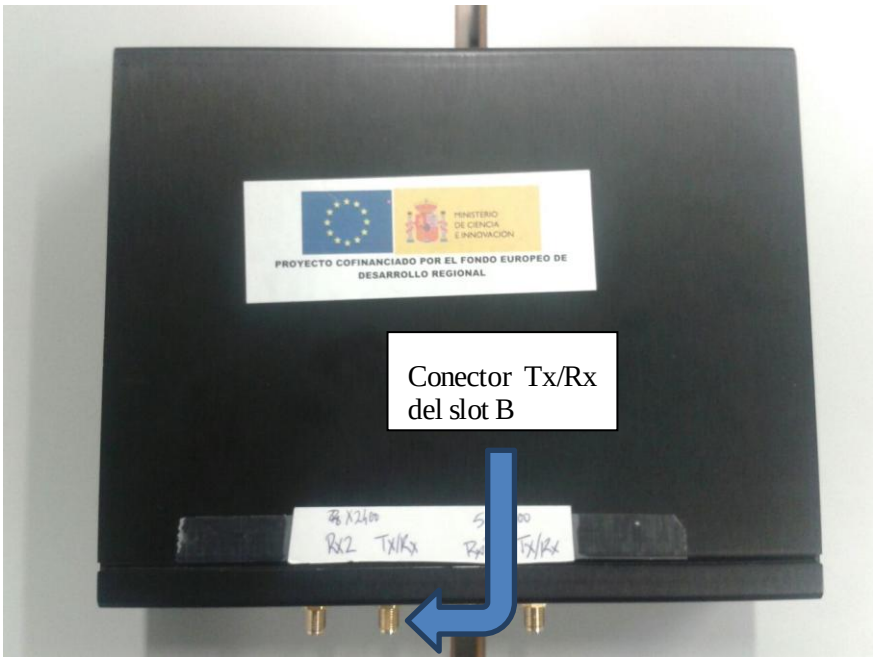

Figura 4-3. USRP receptor

Se realizaron dos envíos de datos, uno mediante antena, obteniéndose el fichero sigRXAire50dB con los parámetros de transmisión y recepción de la Tabla 4–1 y el otro por cable, con fichero de datos sigRXCable45dB y parámetros mostrados en la Tabla 4–2.

Tabla 4–1. Parámetros de transmisión y recepción de sigRXAire50dB

Parámetro	Valor
Tipo de conexión	Antena
Ganancia Tx	20 dB
Frecuencia Tx	2500MHz
Interpolado	256
Ganancia Rx	50 dB
Frecuencia	2500MHz
Diezmado	128

Tabla 4–2. Parámetros de transmisión y recepción de sigRXCable45dB

Parámetro	Valor
Tipo de conexión	Cable
Ganancia Tx	20 dB
Frecuencia Tx	2500MHz
Interpolado	256
Ganancia Rx	45 dB
Frecuencia	2500MHz
Diezmado	128

4.2 Resultados obtenidos y diferencias

Una vez simulado el sistema descrito en el Capítulo 3, al ejecutar el script Receptor_DQPSK.m con el fichero de datos sigRXAire50dB al detectar el offset de frecuencia se observó que mientras que en el caso propuesto dicho offset era de 39896 Hz en nuestro caso pasó a ser de 85768 Hz.

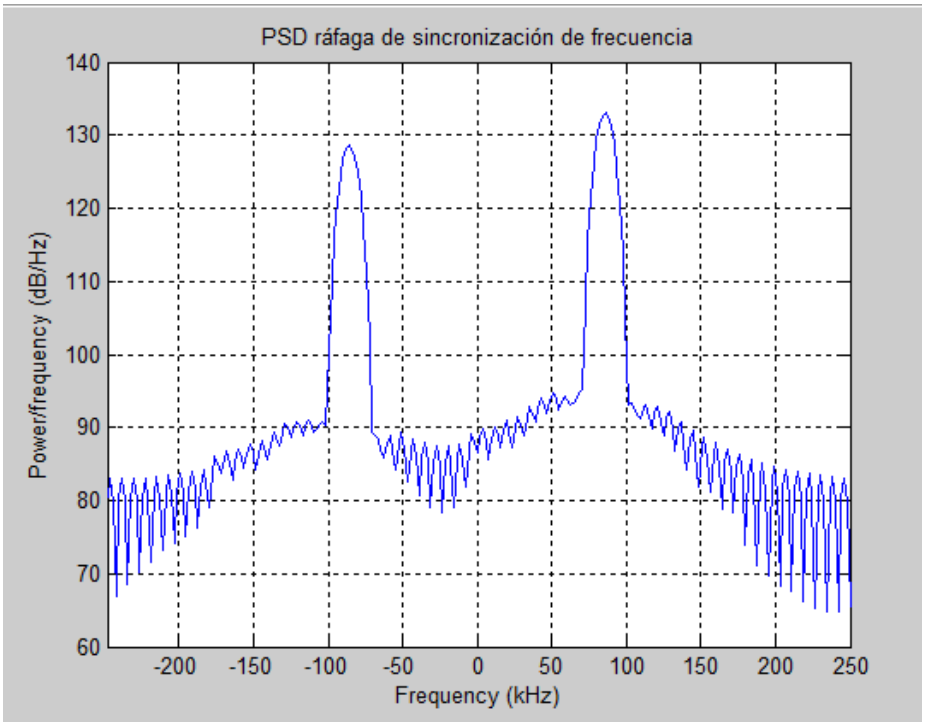

Figura 4-4. Espectro secuencia sincronismo

Para descartar un posible efecto adverso del canal como origen de este aumento en la desviación de frecuencia se emplea el fichero sigRXCable45dB, pues al tratarse de una transmisión por cable, la variabilidad del medio radioeléctrico desaparece. Como el resultado obtenido es similar, desechamos esta hipótesis en favor de un posible envejecimiento de los equipos USRP o una variación en las condiciones de trabajo como la temperatura o la humedad, aunque aún se desconoce el foco exacto de esta desviación.

Tras hacer el ajuste en frecuencia en ambos ficheros, se continua con el proceso de recepción, donde se sincroniza en tiempo, se demodulan los datos y se calcula la BER antes de decodificar. Es aquí donde se observa un aumento significativo de ésta, por causas en ese momento desconocidas.

Exactamente se obtiene una BER=0.0317 significativamente mayor que en el caso propuesto, con una BER=$7.9 \cdot 10^{-5}$.

También se descarta que la causa se deba a una mayor desviación en frecuencia residual, pues esto solo provoca un giro en la constelación y al emplear una modulación diferencial esto no afecta a la BER.

Se procede entonces a analizar el error a lo largo del tiempo, con un instante de muestreo fijo, en este caso el 14, pues es el óptimo según estima el código del receptor, y se obtiene la Figura 4-5.

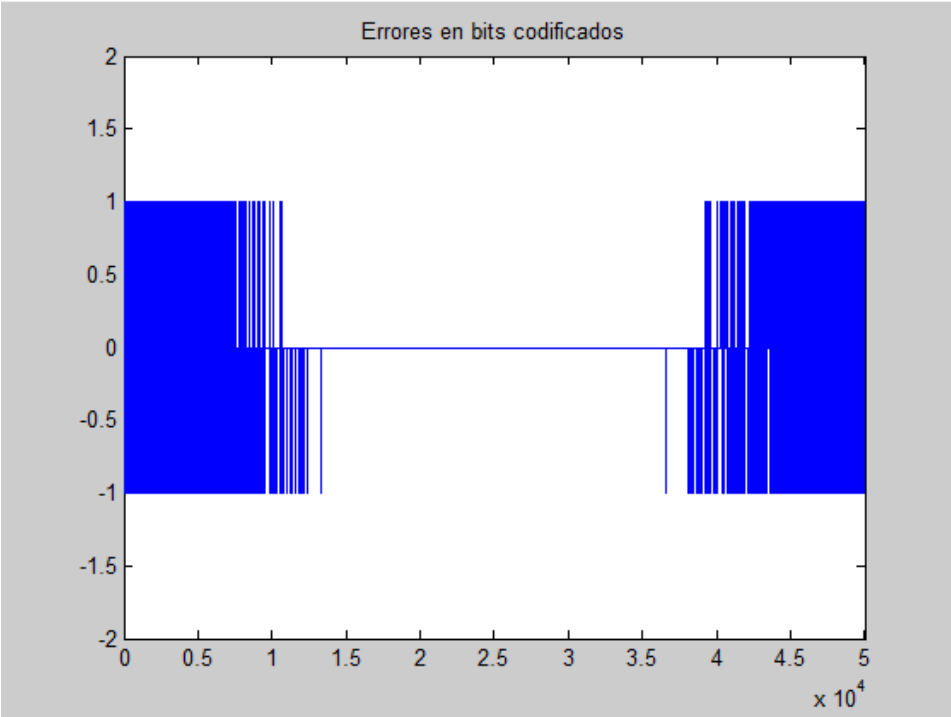

Figura 4-5. Errores en bits codificados para muestra 14

La gran cantidad de errores al principio y al final de nuestra secuencia hacen suponer que el instante óptimo de muestreo varía conforme avanza la secuencia, por lo que se realiza el muestreo en instantes anteriores, por ejemplo en la muestra 10 representado en la Figura 4-6 y en instantes posteriores como el 18, obteniéndose la Figura 4-7. Es significativo observar que los errores se desplazan en el tiempo, desapareciendo al inicio de la secuencia si se muestrea antes y al final si se muestrea después. De hecho en la Figura 4-7, se ha elegido el instante 18 para demostrar que la variación en el muestreo es tan grande que llega incluso a retrasarse un símbolo completo, pues muestrear en el instante 18 es realmente muestrear en el instante 2 un símbolo completo después. Para determinar exactamente cada cuántas muestras se retrasa un símbolo completo se realizará un estudio posterior en la Sección 4.3.

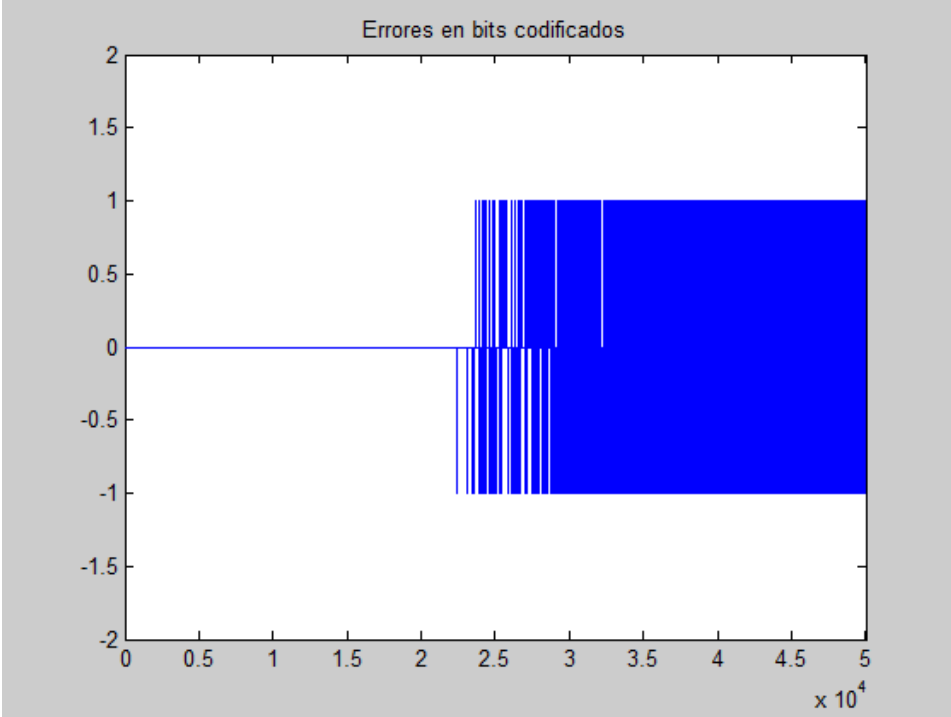

Figura 4-6. Errores en bits codificados para muestra 10

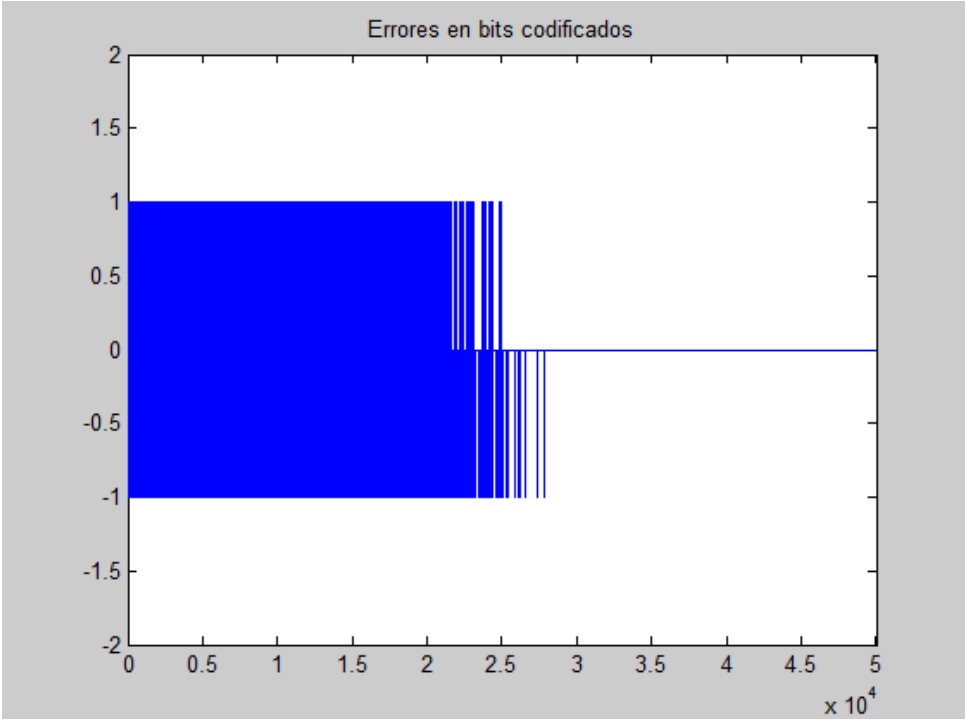

Figura 4-7. Errores en bits codificados para muestra 18

Todo lo anterior nos lleva a concluir que el error en el sincronismo de símbolo era sensiblemente mayor que en las pruebas realizadas con los mismos equipo y configuración similar en [2] y aunque la causa exacta es desconocida debe estar relacionada sin duda con la baja calidad de los relojes internos de los USRP.

Estos relojes presentan una precisión de 25 ppm o partes por millón en el mejor de los casos, es decir, que se retrasan 25 microsegundos cada segundo, lo que está en el orden de nuestro tiempo de símbolo de 32 microsegundos, lo que provoca que después de aproximadamente un segundo de transmisión las muestras se hayan retrasado un tiempo de símbolo aproximadamente, aunque en nuestro caso probablemente el reloj presente un error incluso mayor.

4.3 Análisis del instante de muestreo óptimo

Para poder determinar las características del error del instante óptimo de muestreo se envió una secuencia conocida, en la que observar de manera óptima esta variación a la hora de muestrear.

Se optó por enviar una secuencia de 24000 símbolos alternando el símbolo 1+j con el -1-j precedidos de un intervalo de continua para el sincronismo de frecuencia y de 40 símbolos 1+j y -1-j a los que se les pretendía dar otro uso que posteriormente quedó descartado. Esta secuencia recibida se encuentra almacenada en el fichero sigSecuenciaConocida.

```
secuencia_prueba=kron(kron(ones(1,6000),[1+1j -1-1j 1+1j -1-1j]),[1
zeros(1,ss-1)]);
```

Con esta secuencia se puede observar claramente el instante óptimo de muestreo, puesto que siempre coincide con la muestra para la que su valor absoluto a la salida del filtro adaptado es máximo. Para un caso ideal en el que la secuencia simplemente se pasa por los filtros del transmisor y del receptor si muestreamos de manera constante cada 16 muestras el valor obtenido siempre es máximo, como se observa destacado en rojo en la Figura 4-8 superpuesto en azul con el valor absoluto de la salida del filtro RRC.

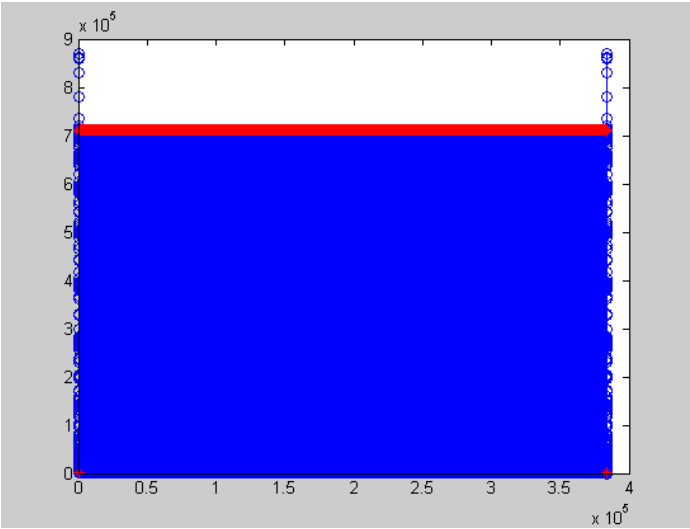

Figura 4-8. Valor absoluto secuencia conocida en caso ideal y muestreo (rojo)

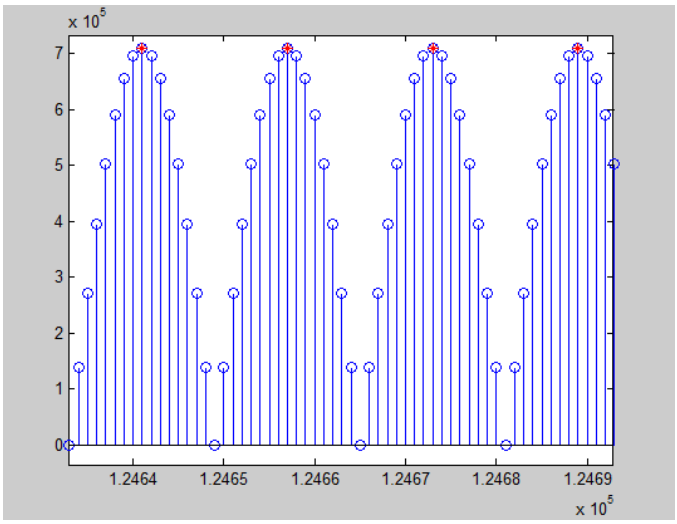

Figura 4-9. Detalle valor absoluto secuencia conocida en caso ideal y muestreo (rojo)

En cambio en el caso real si hacemos el mismo muestreo constante, los valores (en rojo) solo se maximizan en ciertos momentos de la secuencia, como se observa en la Figura 4-10.

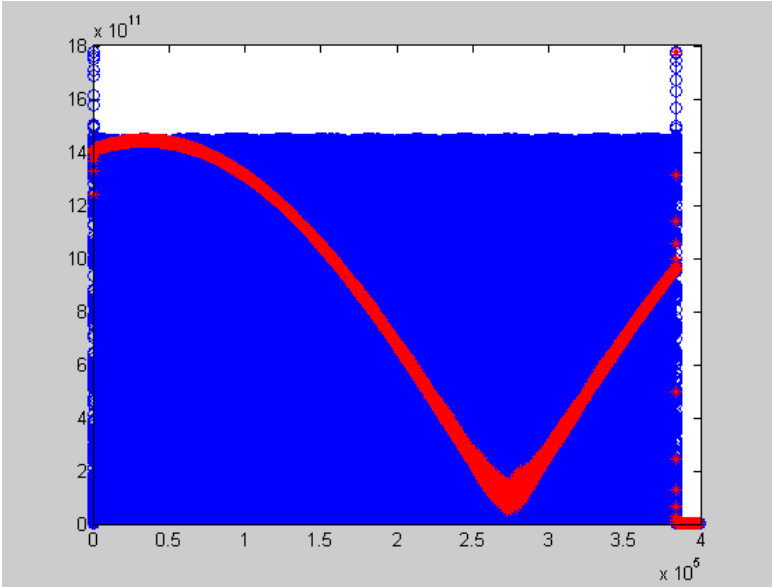

Figura 4-10. Valor absoluto secuencia conocida en caso real y muestreo en instante 7 (rojo)

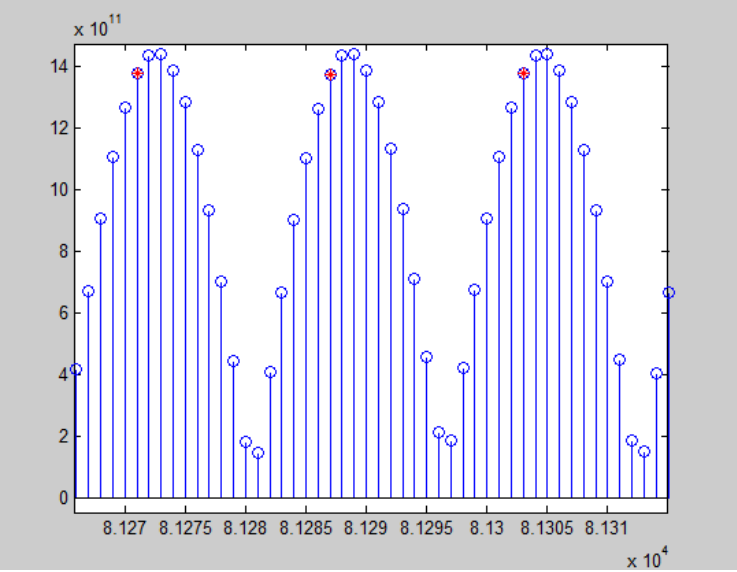

Figura 4-11. Detalle valor absoluto secuencia conocida en caso real y muestreo (rojo)

Si muestreamos en instantes anteriores este máximo se desplaza hacia el inicio de la secuencia y si lo hacemos en instantes posteriores se desplaza hacia el final de la secuencia ilustrado en la Figura 4-12, tal y como ocurría en Figura 4-5, Figura 4-6 y Figura 4-7, corroborándose así la hipótesis de que existe un error en el sincronismo de símbolo.

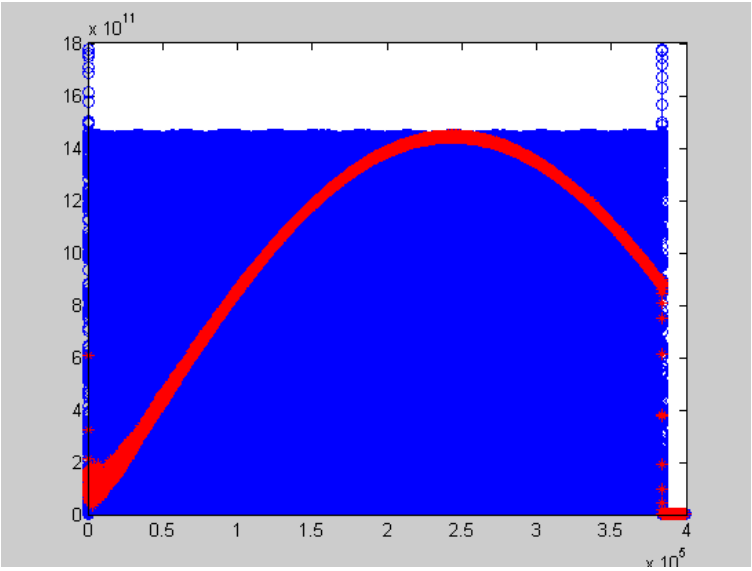

Figura 4-12. Valor absoluto secuencia conocida en caso real y muestreo en instante 14 (rojo)

Se procede entonces a caracterizar esta variación con el tiempo, calculando cuales son los instantes óptimos a lo largo de la secuencia, para lo cual se comienza muestreando la secuencia para el menor instante posible, el 1, y en cada símbolo se compara el valor absoluto de esta muestra con la inmediatamente anterior y posterior. Si la muestra es menor que alguna de ellas, no es el instante óptimo y por tanto será necesario adelantar o atrasar el muestreo según qué instante sea el mayor, si el posterior, donde adelantaremos, o el anterior, donde retrasaremos. Durante el cambio entre instantes existe una pequeña oscilación pero se estabiliza tras pocos símbolos.

La Figura 4-13 contiene en rojo los valores de la secuencia muestreando en los instantes óptimos y se observa que ha quedado corregido nuestro error de sincronismo.

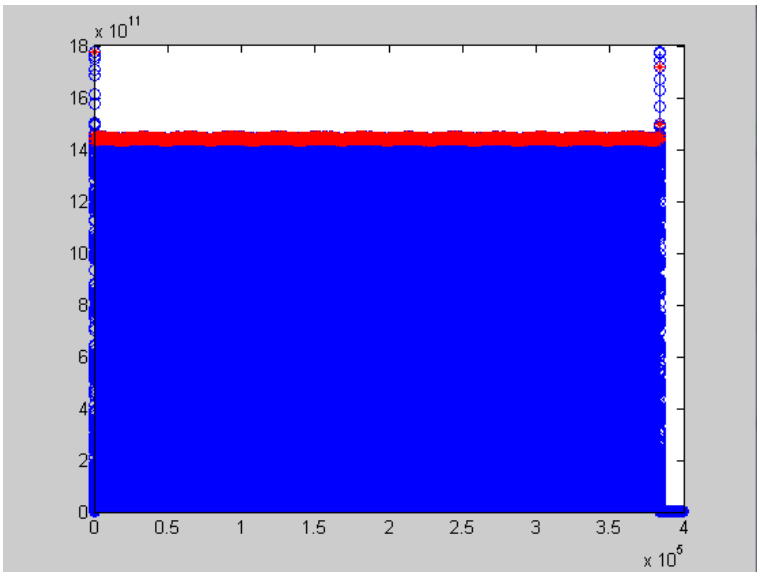

Figura 4-13. Valor absoluto secuencia conocida en caso real y muestreo corregido (rojo)

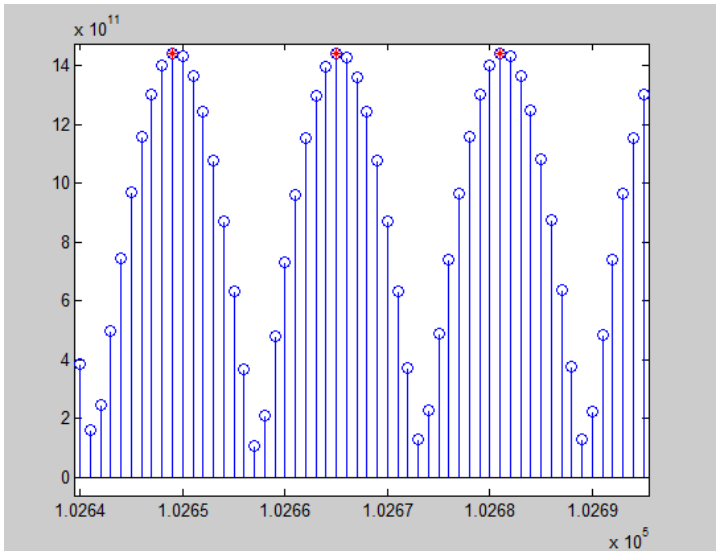

Figura 4-14. Valor absoluto secuencia conocida en caso real y muestreo corregido (rojo)

Si dibujamos los diferentes instantes de muestreo óptimos a lo largo de toda la secuencia y le calculamos a dicha gráfica su recta de regresión lineal, podremos caracterizar la variación del instante de muestreo como se muestra en la Figura 4-15, donde observamos que después de 20000 símbolos (0.64 s) el muestreo se ha saltado un símbolo completo, por ser instantes mayores a 16.

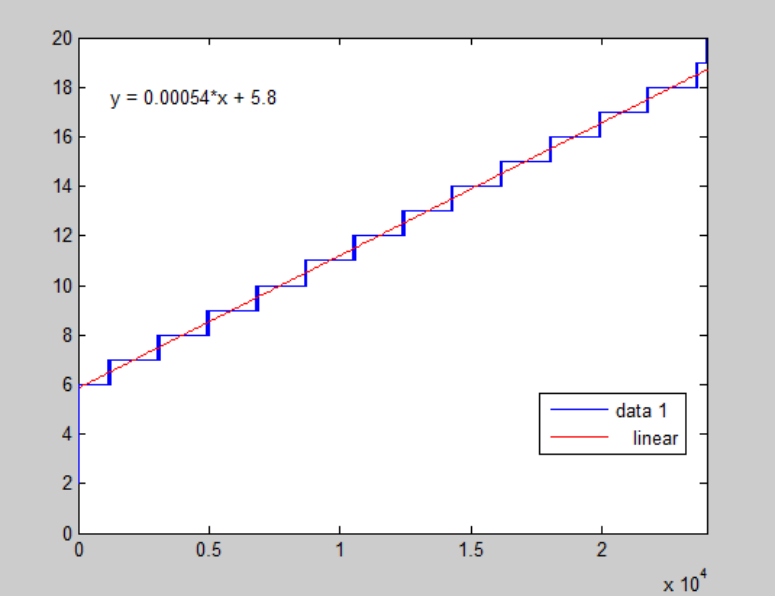

Figura 4-15. Instantes óptimos de muestreo en función del símbolo y recta de regresión

5 SINCRONISMO DE SÍMBOLO

"Synchronization is a critical function in digital communications; its failures may have catastrophic effects on the transmission system performance."

- Umberto Mengali -

Tras las observaciones al intentar reproducir los resultados del sistema DQPSK propuesto, se determinó que existía un error en el sincronismo temporal, que pretendemos dar solución en este capítulo. Por ello se propone la inclusión de un bloque de sincronismo diferente al propuesto en [2], aunque una solución posible a este problema podría ser reemplazar la fuente del error, es decir, introducir una señal de reloj externa en el USRP con una mayor precisión. Otro enfoque posible ante casos como este podría ser el emplear tramas de duración menor, en las que se pueda considerar que el instante óptimo de muestreo es aproximadamente constante para toda la trama.

En el caso propuesto, si se estima una única vez el instante de muestreo en una secuencia en la que éste varía, se determina un "instante óptimo medio", uno que solo será óptimo para una parte de la secuencia. Realmente no se está muestreando correctamente y solo se recupera gran parte de la señal debido a la elevada SNR, que nos permite muestrear en instantes no óptimos y aun así recuperar la información. El error en sincronismo ya existía en el caso propuesto, pero al tener una magnitud menor que en nuestro caso actual no se manifestaba con tanta claridad.

5.1 Bloques de sincronismo

Dentro de los bloques de sincronismo temporal existen tres enfoques: analógico, hibrido y digital. En los dos primeros se corrige la frecuencia del oscilador local después de estimar los errores de sincronismos, en el primero de manera analógica y en el híbrido de manera digital extrayendo la información sobre el sincronismo de la propia señal.

En nuestro caso no tenemos control sobre este oscilador y por tanto recurriremos al tercer enfoque, que se basa en muestrear a una tasa fija, y con un procesamiento digital posterior se corrige el error. Una vez que se tienen las muestras a la salida del filtro adaptado se pasan por un interpolador, que genera muestras intermedias entre las que ya teníamos, para así poder variar el muestreo. Este interpolador será realimentado por una señal de error que corregirá la frecuencia y fase del muestreo, previamente filtrada por un filtro paso de baja que suaviza la señal de error.

El bloque que genera la señal de error se conoce como Timing Error Detector o TED, que presenta una funcionalidad diferente según qué método digital elijamos.

Los dos métodos estudiados son el Gardner y el Early-Late:

- El algoritmo Gardner [19] se basa en buscar el paso por cero entre dos símbolos de signos alternos. Si esto se produce exactamente en la muestra intermedia entre ambos se está muestreando correctamente, para lo que se genera la siguiente señal de error:

$$Error = \Big(y(nT) - y\big((n-1)T\big)\Big) * y(nT - T/2)$$

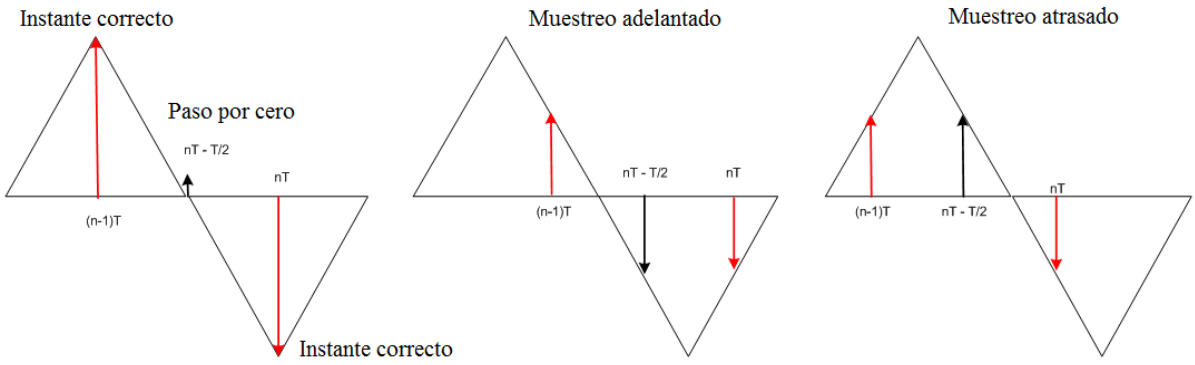

Figura 5-1. Método Gardner [20]

- El algoritmo Early-Late [21] toma una muestra retrasada y otra adelantada con respecto al punto de estimación ideal. Si estas dos muestras son iguales significa que se está muestreando correctamente puesto que la salida del filtro adaptado del receptor es simétrica con respecto a este punto de muestreo. Si la muestra retrasada es mayor será necesario retrasar el muestreo y si es mayor la adelantada se adelanta.

$$Error = \big(y(nT + T_s) - y(nT - T_s)\big) * y(nT)$$

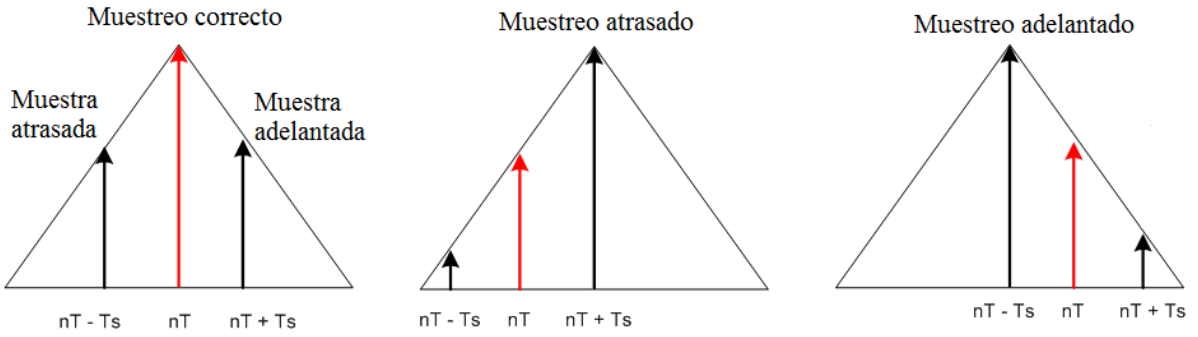

Figura 5-2. Método Early-Late [22]

Estos métodos vienen descritos en la bibliografía consultada para secuencias que alternan los signos de los símbolos (+1 -1 +1 -1) y para pulsos cuadrados, y a la hora de implementarlos en nuestro caso no se ha conseguido que funcionen, por lo que se descarta su uso en favor de otro método diferente.

El método finalmente implementado se basa en buscar la máxima apertura del diagrama de ojo en diferentes partes de la secuencia, que presentó un funcionamiento satisfactorio para subsanar nuestro error.

5.1.1 Método de sincronismo basado en la apertura del diagrama de ojo

El diagrama de ojo es la superposición de las distintas formas de ondas correspondientes a todas las posibles secuencias de bits en un rango de tiempo determinado [23]. El centro del diagrama de ojo, aquel con mayor apertura vertical, presenta la propiedad de ser el instante óptimo de muestreo.

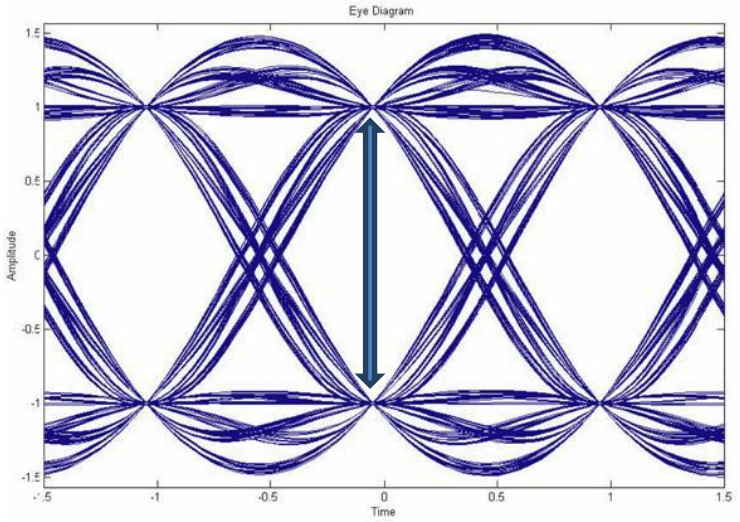

Figura 5-3. Diagrama de ojo

En nuestro caso, para una señal con dos posibles valores (1 y -1) en cada una de las ramas I y Q, el instante óptimo será el que presente una mayor varianza entre sus muestras (apertura de ojo máxima).

Si tomamos el valor absoluto del diagrama, se cumple que el instante óptimo se produce donde la varianza es menor, puesto que en este punto, la media será aproximadamente 1, y la desviación respecto a ésta será mínima, a diferencia de los puntos de muestreo no óptimos, que presentarán una varianza mayor.

El primer paso en el método de sincronismo elegido será tras obtener la señal después del filtro adaptado (*raf_Rx_fil*) calcularle su longitud y dividirla entre el número de veces que se calcularán los instantes de muestreo (*num_vent*), obteniéndose la longitud de cada subsecuencia o ventana donde estimar (*longitud_divid*).

```
longitud_divid=floor(length(raf_Rx_fil)/num_vent/16)*16;
```

Dividimos también entre 16 para así tener símbolos completos y no muestras sueltas y tras redondear volvemos a multiplicar para que *longitud_divid* contenga número de muestras.

Separamos entonces la secuencia en *num_vent* partes de longitud *longitud_divid* y se almacena cada una de las partes en un cell array de tamaño *num_vent* donde cada elemento del cell array será un vector con las muestras de cada ventana.

Por ejemplo la primera ventana tendrá las muestras desde el índice 1 hasta *longitud_divid* y la segunda ventana desde el índice *longitud_divid*+1 hasta 2·*longitud_divid*.

Para cada una de estas ventanas será necesario calcular el instante de muestreo, para lo que aprovecharemos las propiedades del diagrama de ojo. Para ello se muestrea cada subsecuencia en sus 16 instantes de tiempo

posibles y se almacena lo obtenido en 16 vectores diferentes, cada uno conteniendo las muestras en un instante de tiempo diferente.

A cada vector se le calcula la varianza de su valor absoluto y, por las características del diagrama de ojo anteriormente citadas, aquel vector con menor varianza determinará el instante óptimo de muestreo, es decir, si el vector con menor varianza es el muestreado en el instante 7, el instante óptimo de esa ventana es el 7. Estos instantes óptimos se van almacenando en un vector de instantes *inst_optRRC* con longitud el número de ventanas.

De la Figura 4-15 que caracteriza el error de muestreo podemos estudiar la variación de este instante con el tiempo, y observamos que tras un cierto tiempo, ninguno de los 16 instantes de muestreo es el óptimo, ya que es necesario saltarse un símbolo completo, es decir 16 muestras, para poder muestrear correctamente. Como el crecimiento del error es aproximadamente lineal, cuando estimemos que el instante óptimo para el símbolo n es menor que para el símbolo n-1 significará que es necesario incrementar en 16 este instante, es decir saltarse un símbolo. Como ejemplo, si estimo como instante óptimo de muestreo el 1 después de haber obtenido el 16, en realidad debo muestrear en 1+16=17.

Para poder hacer esta suposición debemos fijar un tamaño mínimo de ventana que nos asegure que deberemos saltar un símbolo completo solo cuando hayamos completado un ciclo, esto es pasar por la muestra 16 antes de volver a muestrear en el instante 1. Para nuestro caso este tamaño mínimo es de aproximadamente 1832 símbolos según la Figura 4-15, ya que el error de muestreo se incrementa en 1 tras 1832 símbolos, que coincide con dividir nuestra secuencia en 14 partes (1831 símbolos por ventana) y por tanto el número de ventanas elegido es 14.

Una vez obtenidos los *num_vent* (14) instantes de tiempo óptimos, uno para cada ventana, se procede a muestrear la secuencia completa (*raf_Rx_fil*) *num_vent* veces, cada vez en un instante de tiempo diferente del vector *inst_optRRC* y se almacena lo obtenido en *num_vent* vectores, cada uno muestreado en un instante distinto.

Como existe un retraso inicial en la secuencia se debe calcular el instante inicial de muestreo. Para ello se correla la secuencia (*raf_Rx_fil*) con el pulso RRC y aquel punto donde la correlación sea mayor indicará el inicio real de la secuencia. Como primer punto de muestreo tomaremos este inicio de la secuencia incrementado el valor del instante óptimo de muestreo.

De cada uno de estos *num_vent* (en nuestro caso 14) vectores solo una *num_vent*-ava parte de sus muestras será óptima, la que corresponde con las muestras de la ventana donde este instante era óptimo. Por tanto iremos extrayendo en cada vector una *num_vent*-ava parte, es decir, del primer vector de muestras se toma la primera *num_vent*-ava parte, del segundo vector la segunda *num_vent*-ava parte y así sucesivamente, uniendo las muestras óptimas en un solo vector. Este paso se ilustra en la Figura 5-4.

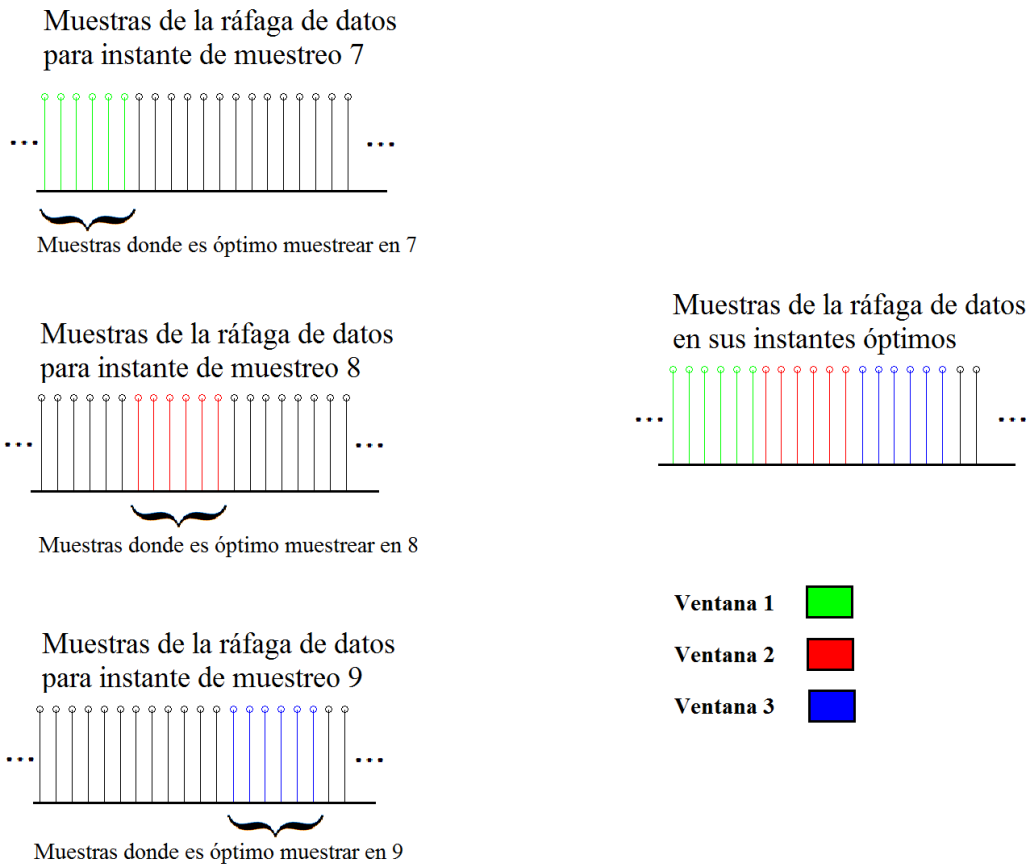

Figura 5-4. Selección y unión de muestras en un solo vector óptimo

Si dibujamos los instantes de muestreo para cada muestra obtenemos la Figura 5-5 para la transmisión por cable y la Figura 5-6 para el caso de transmisión vía aire, que presentan un comportamiento similar al previsto en la Sección 4.3, con pendientes en la recta de regresión similares y una variación en el instante de muestreo de la ventana final entre el caso cableado y el caso aéreo.

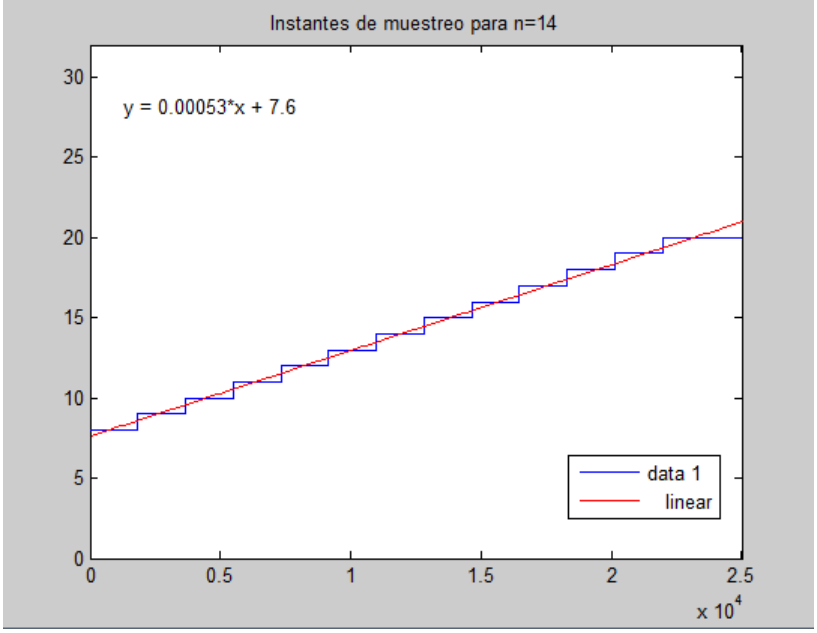

Figura 5-5. Instantes de muestreo para cada símbolo y recta de regresión para cable

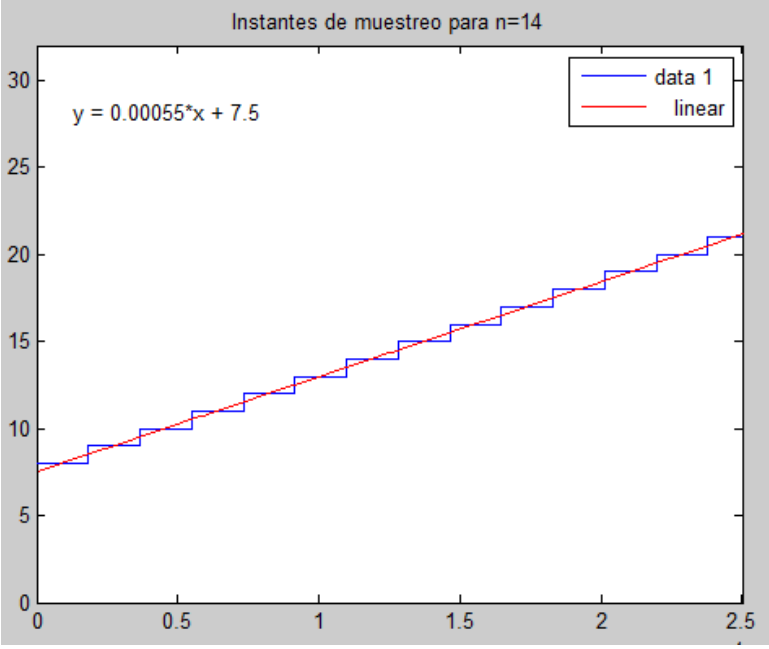

Figura 5-6. Instantes de muestreo para cada símbolo y recta de regresión para canal aire

Finalmente demodulamos el vector de muestras óptimo y calculamos la BER comparando los bits obtenidos con los originales del fichero datos_c. Observamos en la Figura 5-7Figura 5-5 que el error de sincronismo ha sido subsanado.

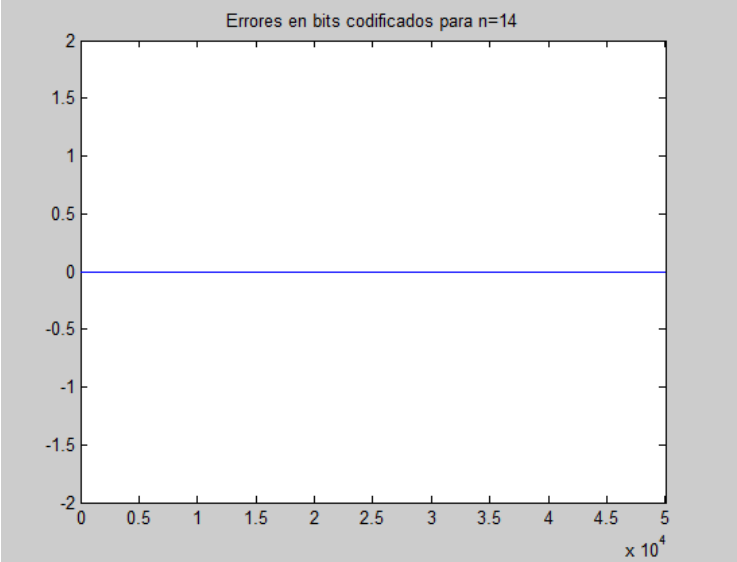

Figura 5-7. BER antes de decodificar tras sincronismo temporal

Ahora procedemos a resumir la funcionalidad del algoritmo de estimación temporal cuyo código puede encontrarse en el Anexo A:

Paso 1: Se calcula el instante de inicio de la secuencia obviando el transitorio inicial.

- Correlamos la parte real de las primeras *num_despRRC* muestras de la señal tras el filtro adaptado (*raf_Rx_fil*) con el pulso RRC con la orden *xcorr*.
- Calculamos el máximo de su valor absoluto y tomamos el índice de este máximo (*x_maxRRC*) en el vector que devuelve *xcorr*.
- Como *xcorr* tiene desplazamientos negativos y positivos y los índices son solo positivos deberemos desplazar este índice *x_maxRRC* para tener el retraso que corresponde, sumándole la mitad de la longitud del pulso RRC y restándole *num_despRRC*.

Paso 2: Se fija el número de ventanas donde estimar (*num_vent*) y calculo la longitud de cada ventana (*longitud_divid*).

```
longitud_divid=floor(length(raf_Rx_fil)/num_vent/16)*16;
```

Paso 3: Se realizan todos los cálculos del instante de muestreo para la ventana 1.

- La primera ventana se almacena en *raf_Rx_fil_i{1}* tomando las muestras de *raf_Rx_fil* desde la 1 hasta *longitud_divid*.
- Con la orden *reshape* reordeno las muestras de *raf_Rx_fil_i{1}* en una matriz *raf_rec{1}* donde cada fila contiene muestras tomadas en uno de los 16 instantes posibles.
- La matriz se traspone y se calcula el máximo de su valor absoluto columna a columna.
- El índice de la columna con menor varianza es el instante óptimo y se almacena en el primer elemento del vector *inst_optRRC(1)*.
- Se calcula el punto inicial de muestreo como:

  ```
  punto_muestreo_fil(1)=inst_optRRC(1)+floor(desp_maxRRC/16)*16;
  ```

- Dividimos entre 16 y redondeamos para así comenzar en el inicio del símbolo más cercano a *desp_maxRRC* y multiplicamos por 16 para cambiar de símbolos a muestras.

- Muestreo *raf_Rx_fil* tomando una muestra cada 16 desde *punto_muestreo_fil(1)* hasta el final del vector y se almacena en *r_i{1}*.

- De este vector tomamos solo la primera ventana y la almacenamos en un vector de muestras óptimo (r).

```
r=r_i{1}(1:floor(length(r_i{1})/num_vent));
```

Paso 4: Bucle for desde *ii=2* hasta *ii=num_vent* para calcular el resto de ventanas.

- Almacenamos en *raf_Rx_fil_i{ii}* el resto de ventanas comenzando desde la muestras final de la ventana anterior+1.

```
raf_Rx_fil_i{ii}=raf_Rx_fil((ii-1)*longitud_divid+1:ii*longitud_divid);
```

- Se reordena *raf_Rx_fil_i{ii}* con la orden *reshape* obteniendo la matriz *raf_rec{ii}*, se traspone esta matriz y se calcula mínimo de la varianza del valor absoluto de cada columna, cuyo índice determina el instante óptimo de muestreo *inst_optRRC(ii)*.
- Si *inst_optRRC(ii-1)* es mayor que inst*_optRRC(ii)* se incrementa en 16.
- Se calcula el punto inicial de muestreo como:

```
punto_muestreo_fil(ii)=inst_optRRC(ii)+floor(desp_maxRRC/16)*16;
```

- Se muestrea *raf_Rx_fil* tomando una muestra cada 16 desde *punto_muestreo_fil(ii)* hasta el final, almacenando lo obtenido en *r_i{ii}*.
- Se toma solo la ventana *ii* de *r_i{ii}*:

```
r_i{ii}(floor((ii-1)*length(r_i{2})/num_vent)+1...
:floor((ii)*length(r_i{2})/num_vent))];
```

- Lo resultante se añade al final del vector de muestras óptimas *r*.

6 CONCLUSIONES

La tecnología SDR está en su mejor momento. En pocos años ha pasado de mero concepto teórico a consolidarse como uno de los sistemas con mayor proyección dentro de las telecomunicaciones. Su gran versatilidad y su relativamente bajo coste hacen que cada vez más gente se interese por esta tecnología. Por tanto será importante para los alumnos que le sigan la pista a todo lo concerniente al mundo de los software radios.

La cada vez mayor necesidad de hacer uso eficiente de un cada vez más limitado espectro y la tendencia a los sistemas radio con inteligencia y adaptabilidad, como la radio cognitiva, nos lleva a pensar que en el futuro será necesario para los ingenieros conocer en mayor o menor grado las tecnologías de radio definida por software.

Uno de los problemas de la cantidad de protocolos, standars y frecuencias que permite desarrollar SDR es dar un mal uso de esta versatilidad. Actualmente se están produciendo ataques a sistemas aparentemente seguros de manera sencilla a través de equipos SDR, que además como el RTL-SDR son de coste reducido. Por este motivo las comunicaciones inalámbricas deben aumentar su seguridad frente a esta nueva fuente de peligros lo que debe ser campo de estudio para los futuros ingenieros.

En el caso de la sincronización temporal en los sistemas de telecomunicación, durante todo el grado se ha tratado este problema como si estuviese resuelto de antemano, y por tanto, al menos yo, pensaba que se trataba de un paso puramente trivial y sencillo de implementar. Esto se aleja completamente de la realidad, pues la sincronización (temporal, fase y frecuencia), es un proceso laborioso, con multitud de variantes según qué canal o qué modulación, en incluso según qué enfoque (analógico o digital) y por tanto creo que sería interesante que durante la carrera se hiciese mayor hincapié a dicho proceso, del cual depende en gran medida una correcta recepción de la señal, sin incurrir en errores.

En este trabajo se ha abordado un posible método para solucionar nuestro problema de sincronismo, una solución basada en la apertura del diagrama de ojo, aunque existen multitud de variantes y enfoques, de los cuales se han programado además del descrito un algoritmo de Gardner y otro de Early-Late pero sin conseguir una sincronización temporal satisfactoria dejándose como posible visión futura su reimplementación.

Como posible línea de investigación futura se deja propuesto el caso de un mayor ruido en el canal, que provoque que al estimar el instante en cada ventana, este ya no siga un incremento perfectamente lineal, debido al ruido y donde no podremos suponer que se completará un ciclo completo (pasar por el instante 16) antes de volver a muestrear en el instante 1 pero una muestra adelantada.

La raíz del problema de sincronismo que surgió al trabajar con el USRP no ha podido ser determinada con exactitud, aunque consideramos que se debe sin duda a la baja calidad de los relojes internos de los USRP, que afecta tanto a la desviación en frecuencia como al sincronismo temporal, aunque esto es solo una mera hipótesis. Al tratarse de un hardware con tanta variedad de tecnologías implicadas (FPGA, DSP, DAC, etc) resulta difícil, al menos con el tiempo y los conocimientos disponibles, determinar el foco exacto del error estudiado.

ANEXO A. CÓDIGOS MATLAB

Receptor Apertura Diagrama de Ojo

```
%--------Receptor basado en apertura diagrama de ojo-----------------------
clear all;

%%
%load('sigRX250070dB.mat');%Después de hacer la transmisión Archivo de Iván
%load('sigRXAire50dB.mat');%Después de hacer la transmisión Archivo de
Antonio Aire
load('sigRXCable45dB.mat');%Después de hacer la transmisión Archivo de
Antonio Cable

sig_Rx=sig_Rx(1000:end);%Elimina posible transitorio inicial

BW = 500e3;

ss=16;%muestras por símbolo
Te = (1:length(sig_Rx))/BW;
subplot(211),plot(Te,real(sig_Rx)),title('Señal recibida
(I)'),xlabel('t(s)');
subplot(212),plot(Te,imag(sig_Rx)),title('Señal recibida
(Q)'),xlabel('t(s)');
pause
close all;

%-------------------------------------------------------------------------
%%SELECCIÓN DE RÁFAGA
%-------------------------------------------------------------------------
v_rms=norm(sig_Rx)/sqrt(length(sig_Rx));
int_seg=v_rms*0.0;%intervalo para soportar un pico de interferencia
ind_mod=find(abs(sig_Rx)>(v_rms+int_seg));
ind_raf=ind_mod(1);
raf_Rx=sig_Rx(ind_raf:ind_raf+1.2*BW); %%se queda con 1.2 s (la ráfaga de
datos está en 0.8s finales)
Tr=(1:length(raf_Rx))/BW;
figure,subplot(211),plot(Tr,real(raf_Rx)),title('Parte en fase de la ráfaga
seleccionada'),xlabel('t(s)');
subplot(212),plot(Tr,imag(raf_Rx)),title('Parte en cuadratura de la ráfaga
seleccionada'),xlabel('t(s)');
pause,close all;

raf_Rx_datos=raf_Rx(0.3*BW:end);
Trd=(1:length(raf_Rx_datos))/BW;
figure,subplot(211),plot(Trd,real(raf_Rx_datos)),title('Parte en fase de la
ráfaga de datos'),xlabel('t(s)');
subplot(212),plot(Trd,imag(raf_Rx_datos)),title('Parte en cuadratura de la
ráfaga de datos'),xlabel('t(s)');
pause,close all;
h=spectrum.welch;
```

```
psd(h,raf_Rx_datos(1:(end-0.2*BW)),'CenterDC',true,'Fs',BW),title('PSD ráfaga
Rx');
pause,close all;


%%%Corrección de frecuencia-------------------------------------------------
N=5*BW;
f=0:BW/N:(BW-BW/N);
psd(h,raf_Rx(1:(0.19*BW)),'CenterDC',true,'Fs',BW),title('PSD ráfaga de
sincronización de frecuencia');
pause,close all;
[amp,f_off]=max(abs(fft(raf_Rx(1:0.19*BW),N)));%f_off será el valor del eje x
en número de muestra
f_off=f_off*BW/N;%se calcula f_off a partir del número de muestra del máximo
teniendo en cuenta el sobremuestreo
raf_Rx_f=raf_Rx_datos.*exp(-j*2*pi*f_off*Trd.');
psd(h,raf_Rx_f(1:(end-0.2*BW)),'CenterDC',true,'Fs',BW),title('PSD ráfaga sin
offset de frecuencia');
pause,close all;
%%

%%%Sincronización en tiempo-------------------------------------------
Ts=(1/BW)*ss;
alpha=0.22;
[pt,tp]=pulso(Ts,alpha,ss); %Definimos el filtro adaptado

raf_Rx_fil=conv(pt,raf_Rx_f(1:0.82*BW)); %Convolucionamos la señal corregida
en frecuencia con el filtro adaptado
subplot(211),plot(Tr(1:length(raf_Rx_fil)),real(raf_Rx_fil)),title('Parte en
fase tras filtro adaptado'),xlabel('t(s)');
subplot(212),plot(Tr(1:length(raf_Rx_fil)),imag(raf_Rx_fil)),title('Parte en
cuadratura tras filtro adaptado'),xlabel('t(s)');
pause,close all;

%Buscamos el instante inicial calculando la correlacion de la rafaga con el
%filtro RRC
num_despRRC=200;%Posibles puntos iniciales

[correlacionRRC,lags]=xcorr(real(raf_Rx_fil(1:num_despRRC)),pt);
plot(lags,abs(correlacionRRC)),title('Correlación ráfaga tras filtro con
pulso coseno alzado'),xlabel('desplazamiento (número de
muestras)'),pause,close all;
[max_corr,x_maxRRC]=max(abs(xcorr(real(raf_Rx_fil(1:num_despRRC)),pt)));
desp_maxRRC=(x_maxRRC-num_despRRC)+length(pt)/2;%xcorr tiene desplazamientos
negativos y positivos


%Número de subsecuencias en las que dividir la ráfaga recibida-------------
num_vent=14;
%--------------------------------------------------------------------------
longitud_divid=floor(length(raf_Rx_fil)/num_vent/16)*16;%dividimos el vector
de muestras en num_vent partes para hallar el tiempo de muestreo en cada una
de ellas
%redondeamos con floor puesto que el número de muestras no es divisible
%exactamente por num_vent, dividimos por 16 para así tomar símbolos
%completos y multiplicamos por 16 para así tener de nuevo muestras y no
%símbolos

% Realizamos el cálculo del instante de muestreo primero para la ventana
```

```
% inicial ii=1-----------------------------------------------------------------
---
ii=1;
raf_Rx_fil_i{ii}=raf_Rx_fil((ii-1)*longitud_divid+1:ii*longitud_divid);
%raf_Rx_fil_i es un cell array que contiene en cada uno de sus elementos el
vector con las muestras de cada ventana
fin_raf{ii}=mod(length(raf_Rx_fil_i{ii}),16); %Para eliminar las muestras
finales y reshape opere adecuadamente
raf_rec{ii}=reshape(raf_Rx_fil_i{ii}(1:end-
fin_raf{ii}),16,((length(raf_Rx_fil_i{ii})-fin_raf{ii})/16)); %Para tomar las
muestras en los 16 instantes posibles y tenerlas separadas en 16 vectores
diferentes
v{ii}=raf_rec{ii}.';%reshape va ordenando por columnas y se pretende tener en
cada fila 16 muestras
m{ii}=abs(v{ii});
sigma{ii}=var(m{ii}); %calculamos la varianza de cada una de las 16 columnas
[Y,inst_optRRC(ii)]=min(sigma{ii});%la menor de las varianzas será para el
instante de muestreo óptimo de los 16 posibles
%inst_optRRC contiene todos los instantes de muestreo optimos, uno para cada
ventana

punto_muestreo_fil(ii)=inst_optRRC(ii)+floor(desp_maxRRC/ss)*ss; %el punto de
muestreo inicial será el instante de muestreo óptimo más cercano al maximo de
la correlación
r_i{ii}=raf_Rx_fil(punto_muestreo_fil(ii):ss:length(raf_Rx_fil));
%muestreamos la señal en el instante óptimo de cada ventana
%r_i contiene en cada elemento un vector de muestras, cada uno muestreado
%en un instante de los óptimos para cada ventana. Ej: r_i{1} contiene el
%vector muestreado en el instante óptimo de la primera ventana
%contenido en inst_optRRC(1)

r=r_i{ii}(1:floor(length(r_i{1})/num_vent));%tomamos solo la parte de r_i{ii}
donde el muestreo era óptimo, es decir 1/num_vent del total
%r contendrá todas las muestras óptimas
%-------------------------------------------------------------------------

% Realizamos el cálculo del instante de muestreo para el resto de ventanas
for ii=2:num_vent
raf_Rx_fil_i{ii}=raf_Rx_fil((ii-1)*longitud_divid+1:ii*longitud_divid);
fin_raf{ii}=mod(length(raf_Rx_fil_i{ii}),16);
raf_rec{ii}=reshape(raf_Rx_fil_i{ii}(1:end-
fin_raf{ii}),16,((length(raf_Rx_fil_i{ii})-fin_raf{ii})/16));
v{ii}=raf_rec{ii}.';
m{ii}=abs(v{ii});
sigma{ii}=var(m{ii});
[Y,inst_optRRC(ii)]=min(sigma{ii});

%Si se da el caso de que el instante calculado es menor que el anterior,
%se incrementa en 16 su valor, hay que saltarse un simbolo
if inst_optRRC(ii)<inst_optRRC(ii-1)
    inst_optRRC(ii)=16+inst_optRRC(ii);
end
punto_muestreo_fil(ii)=inst_optRRC(ii)+floor(desp_maxRRC/ss)*ss;%cambiamos el
instante óptimo

r_i{ii}=raf_Rx_fil(punto_muestreo_fil(ii):ss:length(raf_Rx_fil));%muestreamos
la señal en el instante óptimo de cada ventana

if ii~=num_vent
r=[r;r_i{ii}(floor((ii-
1)*length(r_i{2})/num_vent)+1:floor((ii)*length(r_i{2})/num_vent))];%tomamos
```

```
solo la parte de r_i{ii} donde el muestreo era óptimo, es decir 1/num_vent
del total
%r contendrá todas las muestras óptimas

else %la última ventana tiene un tamaño ligeramente diferente
r=[r;r_i{ii}(floor((ii-1)*length(r_i{2})/num_vent)+1:end)];
end


end
%--------------------------------------------------------------------------


%--------------------------------------------------------------------------
%%DEMODULADOR DQPSK
%--------------------------------------------------------------------------

est_anterior=abs(real(r(1)))+j*abs(imag(r(1)));
n=1;
for h=1:length(r)
    dif1=abs(r(h)-est_anterior);
    dif2=abs(r(h)-est_anterior*exp(j*pi/2));
    dif3=abs(r(h)-est_anterior*exp(j*pi));
    dif4=abs(r(h)-est_anterior*exp(-j*pi/2));

    vec_dif=[dif1 dif2 dif3 dif4];
    [min_dif,pos_min_dif]=min(vec_dif);

    if pos_min_dif==1
        datos_dem(n:n+1)=[0 0];
    else
        if pos_min_dif==2
            datos_dem(n:n+1)=[0 1];
        else
            if pos_min_dif==3
                datos_dem(n:n+1)=[1 1];
            else
                datos_dem(n:n+1)=[1 0];
            end
        end
    end
    n=n+2;
    est_anterior=r(h);
end

%%
%--------------------------------------------------------------------------
%%Tasa de error bits codificados
%--------------------------------------------------------------------------
load datos_c;%carga la variable datos_c

Pc=sum(abs(datos_dem(1:length(datos_c))-datos_c))/length(datos_c)%Tasa de
error de bits codificados
figure,plot(datos_dem(1:length(datos_c))-datos_c),title(['Errores en bits
codificados para n=',int2str(num_vent)]),axis([0 50006 -2 2]),pause,close
all;

%--------------------------------------------------------------------------
%%DECODIFICADOR DE CANAL
```

```
%-------------------------------------------------------------------------
t = poly2trellis(3,[3 5]);
k = log2(t.numInputSymbols);
tblen = 3;
%[d m p in] = vitdec(datos_c,t,tblen,'cont','hard');
[d m p in] = vitdec(datos_dem(1:length(datos_c)),t,tblen,'cont','hard');
datos_dec=d(tblen*k+1:end);%Se elimina el transitorio inicial

%-------------------------------------------------------------------------
%%BER
%-------------------------------------------------------------------------
load datos_b;
BER=sum(abs(datos_dec(1:length(datos_b))-datos_b))/length(datos_b)%Tasa de
error de bits codificados
figure,plot(datos_dec(1:length(datos_b))-datos_b),title(['Errores en bits
para n=',int2str(num_vent)]),axis([0 25000 -2 2]),pause,close all;

%%%%%% plot de los instantes de muestreo

instantes=kron(inst_optRRC,ones(1,floor(length(r_i{2})/num_vent)));%para
tener cada instante optimo repetido durante el tamaño de cada ventana
ejex=1:length(datos_c)/2;
figure,plot(ejex,instantes(1:length(ejex))),title(['Instantes de muestreo
para n=',int2str(num_vent)]),axis([0 length(datos_c)/2 0 32])
pause,close all
```

Receptor secuencia conocida

```
%----------------------Receptor secuencia conocida-----------------------

%%RECEPCIÓN
%-------------------------------------------------------------------------
clear all;
load('sigSecuenciaConocida.mat');%Después de hacer la transmisión
sig_Rx=sig_Rx(1000:end);%Elimina posible transitorio inicial


BW = 500e3;

ss=16;%muestras por símbolo
Te = (1:length(sig_Rx))/BW;
subplot(211),plot(Te,real(sig_Rx)),title('Señal recibida
(I)'),xlabel('t(s)');
subplot(212),plot(Te,imag(sig_Rx)),title('Señal recibida
(Q)'),xlabel('t(s)');
pause
close all;
%-------------------------------------------------------------------------
%%SELECCIÓN DE RÁFAGA
%-------------------------------------------------------------------------
v_rms=norm(sig_Rx)/sqrt(length(sig_Rx));
int_seg=v_rms*0.0;%intervalo para soportar un pico de interferencia
ind_mod=find(abs(sig_Rx)>(v_rms+int_seg));
ind_raf=ind_mod(1);
raf_Rx=sig_Rx(ind_raf:ind_raf+1.1*BW); %%se queda con 1.2 s (la ráfaga de
datos está en 0.8s finales)
Tr=(1:length(raf_Rx))/BW;
figure,subplot(211),plot(Tr,real(raf_Rx)),title('Parte en fase de la ráfaga
seleccionada'),xlabel('t(s)');
```

```
subplot(212),plot(Tr,imag(raf_Rx)),title('Parte en cuadratura de la ráfaga
seleccionada'),xlabel('t(s)');
pause,close all;

raf_Rx_prueba=raf_Rx(0.3*BW:end);
Trp=(1:length(raf_Rx_prueba))/BW;
figure,subplot(211),plot(Trp,real(raf_Rx_prueba)),title('Parte en fase de la
ráfaga de prueba'),xlabel('t(s)');
subplot(212),plot(Trp,imag(raf_Rx_prueba)),title('Parte en cuadratura de la
ráfaga de prueba'),xlabel('t(s)');
pause,close all;
h=spectrum.welch;
% psd(h,raf_Rx_datos(1:(end-0.2*BW)),'CenterDC',true,'Fs',BW),title('PSD
ráfaga Rx');
% pause,close all;

%%% Separamos rafaga de sincro con rafaga de "datos", esta rafaga de sincro
%%% al final no tiene uso en el codigo
raf_Rx_sincro=raf_Rx_prueba(1:800);
Trs=(1:length(raf_Rx_sincro))/BW;
figure,subplot(211),plot(Trs,real(raf_Rx_sincro)),title('Parte en fase de la
ráfaga de sincro'),xlabel('t(s)');
subplot(212),plot(Trs,imag(raf_Rx_sincro)),title('Parte en cuadratura de la
ráfaga de sincro'),xlabel('t(s)');
pause,close all;
psd(h,raf_Rx_sincro,'CenterDC',true,'Fs',BW),title('PSD ráfaga Rx sincro');
pause,close all;

raf_Rx_datos=raf_Rx_prueba(801:end);
Trd=(1:length(raf_Rx_datos))/BW;
figure,subplot(211),plot(Trd,real(raf_Rx_datos)),title('Parte en fase de la
ráfaga de datos'),xlabel('t(s)');
subplot(212),plot(Trd,imag(raf_Rx_datos)),title('Parte en cuadratura de la
ráfaga de datos'),xlabel('t(s)');
pause,close all;
psd(h,raf_Rx_datos,'CenterDC',true,'Fs',BW),title('PSD ráfaga Rx datos');
pause,close all;

%%%

%-------------------------------------------------------------------------
%%SINCRONIZACIÓN
%-------------------------------------------------------------------------

%%%Corrección de frecuencia-----------------------------------------------
N=5*BW;
f=0:BW/N:(BW-BW/N);
psd(h,raf_Rx(1:(0.19*BW)),'CenterDC',true,'Fs',BW),title('PSD ráfaga de
sincronización de frecuencia');
pause,close all;
[amp,f_off]=max(abs(fft(raf_Rx(1:0.19*BW),N)));%f_off será el valor del eje x
en número de muestra
f_off=f_off*BW/N;%se calcula f_off a partir del número de muestra del máximo
teniendo en cuenta el sobremuestreo

% Correccion de frecuencia rafaga sincro
raf_Rx_f_sincro=raf_Rx_sincro.*exp(-j*2*pi*f_off*Trs.');
psd(h,raf_Rx_f_sincro,'CenterDC',true,'Fs',BW),title('PSD ráfaga sincro sin
offset de frecuencia');
pause,close all;
```

```
Trfs=(1:length(raf_Rx_f_sincro))/BW;
figure,subplot(211),plot(Trfs,real(raf_Rx_f_sincro)),title('Parte en fase de
la ráfaga de sincro'),xlabel('t(s)');
subplot(212),plot(Trfs,imag(raf_Rx_f_sincro)),title('Parte en cuadratura de
la ráfaga de sincro'),xlabel('t(s)');
pause,close all;

% Correccion de frecuencia rafaga datos
raf_Rx_f_datos=raf_Rx_datos.*exp(-j*2*pi*f_off*Trd.');
psd(h,raf_Rx_f_datos,'CenterDC',true,'Fs',BW),title('PSD ráfaga datos sin
offset de frecuencia');
pause,close all;
Trfd=(1:length(raf_Rx_f_datos))/BW;
figure,subplot(211),plot(Trfd,real(raf_Rx_f_datos)),title('Parte en fase de
la ráfaga de datos'),xlabel('t(s)');
subplot(212),plot(Trfd,imag(raf_Rx_f_datos)),title('Parte en cuadratura de la
ráfaga de datos'),xlabel('t(s)');
pause,close all;


%Se define ya el filtro RRC para calcular la correlación y el instante
%óptimo de muestreo
Ts=(1/BW)*ss;
alpha=0.22;
[pt,tp]=pulso(Ts,alpha,ss);
num_desp=100;

%-------------------------------------------------------------------------
%%FILTRO ADAPTADO RRC
%-------------------------------------------------------------------------
raf_Rx_sincro_fil=conv(pt,raf_Rx_f_sincro);
Tsinc=(1:length(raf_Rx_sincro_fil))/BW;
subplot(211),plot(Tsinc,real(raf_Rx_sincro_fil)),title('Parte en fase sincro
tras filtro adaptado'),xlabel('t(s)');
subplot(212),plot(Tsinc,imag(raf_Rx_sincro_fil)),title('Parte en cuadratura
sincro tras filtro adaptado'),xlabel('t(s)');
pause,close all;

% filtramos la rafaga de datos
raf_Rx_datos_fil=conv(pt,raf_Rx_f_datos);
Tdat=(1:length(raf_Rx_datos_fil))/BW;
subplot(211),plot(Tdat,real(raf_Rx_datos_fil)),title('Parte en fase sincro
tras filtro adaptado'),xlabel('t(s)');
subplot(212),plot(Tdat,imag(raf_Rx_datos_fil)),title('Parte en cuadratura
sincro tras filtro adaptado'),xlabel('t(s)');
pause,close all;


%%%%%%%%%%%%%%% RESINCRONIZACIÓN EN TIEMPO

% Seleccionamos el instante inicial calculando la correlacion con el pulso
% RRC
num_despRRC=200;
[correlacionRRC,lags]=xcorr(real(raf_Rx_datos_fil(1:num_despRRC)),pt);
plot(lags,abs(correlacionRRC)),title('Correlación ráfaga tras filtro con
pulso coseno alzado'),xlabel('desplazamiento (número de
muestras)'),pause,close all;
[max_corr,x_maxRRC]=max(abs(xcorr(real(raf_Rx_datos_fil(1:num_despRRC)),pt)))
;
desp_maxRRC=(x_maxRRC-num_despRRC)+length(pt)/2;%xcorr tiene desplazamientos
negativos y positivos
```

```
% Para calcular el instante optimo comenzamos muestreando en el instante 1
% y modificando su valor segun convenga para cada símbolo ( se enviaron
% 24000 simbolos)
inst_optRRC=1;
for ii=1:24000
punto_muestreo_fil=inst_optRRC+floor(desp_maxRRC/ss)*ss; %el punto de
muestreo inicial será el el instante de muestreo óptimo más cercano al máximo
de la correlación

% Si el valor absoluto de la señal es menor que en muestras anteriores y
% posteriores se cambia el instante de muestreo
if abs(raf_Rx_datos_fil(punto_muestreo_fil+1+16*(ii-
1)))>abs(raf_Rx_datos_fil(punto_muestreo_fil+16*(ii-1)))
 inst_optRRC=inst_optRRC+1; % adelantamos
elseif abs(raf_Rx_datos_fil(punto_muestreo_fil-1+16*(ii-
1)))>abs(raf_Rx_datos_fil(punto_muestreo_fil+16*(ii-1)))
 inst_optRRC=inst_optRRC-1; % retrasamos
end

vector(ii)=inst_optRRC; % vector contiene los instantes de muestreo optimos
para cada símbolo
punto_muestreo_fil=inst_optRRC+floor(desp_maxRRC/ss)*ss;
r(ii)=raf_Rx_datos_fil(punto_muestreo_fil+16*(ii-1)); % vector con la señal
ya muestreada
ejex(ii)=punto_muestreo_fil+16*(ii-1); % contiene que muestras de cada
simbolo se han tomado
end

%%%% Grafica con el valor absoluto de la señal + valor absoluto de las
%%%% muestras en un instante fijo, en este caso el 7
inst_optRRCfijo=7;
punto_muestreo_fil=inst_optRRCfijo+floor(desp_maxRRC/ss)*ss;

y2=abs(raf_Rx_datos_fil); % valor absoluto de los valores de la señal tras el
filtro adaptado
stem(abs(raf_Rx_datos_fil));

hold on, plot([punto_muestreo_fil:16:length(y2)],
y2(punto_muestreo_fil:16:end),'r*'),hold off
pause, close all
%%%% ------------------------------------------------------------------------

%%%% Grafica con el valor absoluto de la señal + valor absoluto de las
%%%% muestras en sus instantes optimos
stem(abs(raf_Rx_datos_fil));
hold on, plot(ejex, abs(r),'r*'),hold off
pause, close all
%%%% ------------------------------------------------------------------------

%%%% Caso ideal donde se genera la misma señal que en el caso real
secuencia_prueba=kron(kron(ones(1,6000),[1+1j -1-1j 1+1j -1-1j]),[1
zeros(1,ss-1)]);
sig_prueba=conv(secuencia_prueba,pt); % se filtra por el filtro conformador
del Tx

sig=conv(pt,sig_prueba); % se filtra por el filtro adaptado del Rx

%%%% Grafica con el valor absoluto de la señal + valor absoluto de las
%%%% muestras para el caso ideal
```

```
stem(abs(sig));
hold on, plot([1:16:length(sig)], abs(sig(1:16:end)),'r*'),hold off

%%%% Instantes de muestreo optimos para casa simbolo
plot(1:24000,vector),axis([0 24000 0 20])
```

almacenaDatos

```
% Script de Matlab para la recepcion de datos y almacenamiento en un
% fichero


input('Pulsar enter para configurar parámetros de recepción');
sock=SDR4All_Connect(0,'SlotB','RX'); %  USRP #0, slot B en recepcion
G=input('Seleccionar ganancia (dB) [0-90]: ');
SDR4All_SetGain(sock,G); % Ganancia de transmisión máxima
F=input('Seleccionar frecuencia (MHz) [2400-2700]: ');
SDR4All_SetFreq(sock,F*1e6); % Frecuencia de recepcion


BW=500e3; % Ancho de banda
input('Ancho de banda'),BW
SDR4All_SetDecimRate(sock,64e6/(BW)); % Tasa de muestras y ancho de banda de
500 KHz (128MHz/256)
input('Pulsar enter para comenzar la recepción');
[sig_Rx] = SDR4All_GetData(sock,10*BW); % 10 segundos de señal son
almacenados en la variable sig_Rx

% Almacenamos en el fichero los datos recibidos en la variable
save sigRXCable45dB.mat sig_Rx;
```

ANEXO B. MANUAL INSTALACIÓN SDR4ALL

La instalación de SDR4All presenta una serie de peculiaridades que este manual pretende solucionar.

Tras seguir el proceso de instalación normal ejecutando el fichero Install.msi de los dos posibles en la carpeta Tools4SDR_v121 el toolbox no queda almacenado correctamente en Matlab. Por ello será necesario ejecutar manualmente los siguientes pasos:

1. Entramos en la carpeta que haya generado el instalador.
2. Dentro de ella habrá otra carpeta llamada Matlab, donde deberemos entrar y copiar todos los ficheros .mexw32
3. Vamos al directorio raíz de Matlab y una vez en el entramos en la carpeta llamada toolbox
4. Creamos una carpeta que se llame SDR4All y pegamos los ficheros .mexw32
5. Cada vez que iniciemos Matlab y queramos usar las funciones anteriores deberemos teclear los comandos

```
addpath('ruta a la carpeta anteriormente creada');
rehash toolbox;
```

ANEXO C. MODELO MATEMÁTICO SISTEMA DQPSK

En este apartado se procede a realizar el modelado matemático de un sistema DQPSK [10] a través de un canal AWGN que presenta una cierta desviación en fase y frecuencia.

Una modulación DQPSK paso de banda presenta el siguiente modelo matemático:

$$S_{IF}(t) = Re\{S_{CE}(t)e^{j2\pi f_c t}\}$$

Donde f_c representa la frecuencia de la portadora y S_{CE} la envolvente compleja de la señal.

Esta envolvente se puede definir a su vez como

$$S_{CE}(t) = \sum_i c_i g(t - iT)$$

c_i representa los símbolos de información y $g(t)$ es el pulso conformador.

En nuestro caso, este pulso es una raíz de coseno alzado (RRC)

$$g(t) = \frac{2\alpha}{\pi\sqrt{T}} \frac{cos\left[\frac{(1+\alpha)\pi t}{T}\right] + \frac{sen\left[\frac{(1-\alpha)\pi t}{T}\right]}{\frac{4\alpha t}{T}}}{1 - \left(\frac{4\alpha t}{T}\right)^2}$$

Para una modulación DQPSK

$$c_i = e^{j\alpha_i}$$

Al tratarse de una modulación diferencial, existe recurrencia entre los símbolos.

$$\alpha_k = \alpha_{k-1} + \delta_k$$

$$c_k = c_{k-1}e^{j\delta_k}$$

Con posibles estados iniciales $\alpha_i = \left\{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right\}$

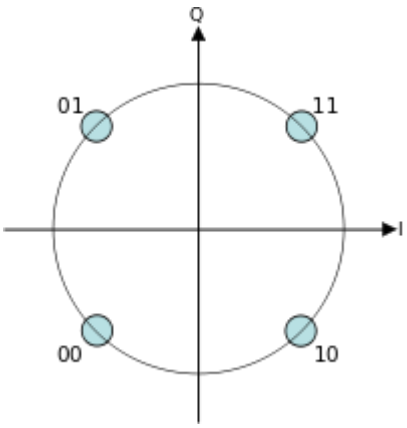

Figura 0-1. Constelación DQPSK

La señal pasa a continuación por un canal AWGN que introduce una componente de ruido $w_{IF}(t)$ y un retraso τ

$$r_{IF}(t) = s_{IF}(t-\tau) + w_{IF}(t)$$

Ya en el lado del receptor, al tratarse de una señal con componente en fase y cuadratura, se multiplica por $2\cos(2\pi f_{cL}t + \phi_L)$ y $-2\text{sen}(2\pi f_{cL}t + \phi_L)$ para bajar la frecuencia.

Se introduce entonces en un filtro paso de bajo para eliminar las componentes espectrales indeseadas.

La frecuencia del oscilador receptor no es exactamente igual que la transmitida, por lo que existirá una desviación de la frecuencia portadora $\nu = f_c - f_{cL}$

La salida del filtro paso de baja queda de la forma

$$r(t) = s(t) + w(t)$$

$$s(t) \triangleq e^{j(2\pi\nu t+\theta)} s_{CE}(t-\tau)$$

$$w(t) = w_{IF} e^{-j(2\pi f_{cL} t+\theta)}$$

$s(t)$ puede expresarse también como

$$s(t) = e^{j(2\pi\nu t+\theta)} \sum_i c_i g(t - iT - \tau)$$

En este punto se intenta corregir la desviación en frecuencia surgida en los apartados anteriores

$$r'(t) = r(t)e\text{^}(-j2\pi\hat{\nu}t)$$
$$s'(t) = e^{j(2\pi\nu t+\theta)} e^{-j2\pi\hat{\nu}t} s_{CE}(t-\tau)$$
$$s'(t) = e^{j(2\pi\Delta\nu t+\theta)} s_{CE}(t-\tau)$$

Donde $\Delta\nu = \nu - \hat{\nu}$

Esta señal se filtra entonces por otro pulso raíz de coseno alzado y se muestrea:

$$x(k) = c_k e^{j[2\pi\Delta\nu(kT+\tau)+\theta]} + n(k)$$

Finalmente a la entrada del detector $z(k) = c_k c_{k-1}^* e^{j2\pi\Delta\nu T} + N(k)$

$$z(k) = e^{j\delta_k} e^{j2\pi\Delta\nu T} + N(k)$$

Que no presenta dependencia con el error en fase, solo con la desviación en frecuencia.

REFERENCIAS

[1] The Wireless Innovation Forum, "The Wireless Innovation Forum," 2014. [Online]. Available: http://www.wirelessinnovation.org/. [Accessed Julio 2015].

[2] I. Pinar Domínguez and J. J. Murillo Fuentes, Laboratorio de Comunicaciones Digitales Radio Definida por Software, Sevilla: Universidad de Sevilla, 2011.

[3] J. H. Reed, Software Radio: A Modern Approach to Radio Engineering, New Jersey: Prentice-Hall, 2002.

[4] GNU Radio Companion, "GNU Radio," 2013. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki. [Accessed Julio 2015].

[5] Ettus Research, "Ettus Research," 2015. [Online]. Available: http://www.ettus.com/. [Accessed Julio 2015].

[6] Alcatel-Lucent, "Alcatel-Lucent. New Architectures and Technologies for Software-Defined Radio Base Stations," [Online]. Available: http://www3.alcatel-lucent.com/wps/DocumentStreamerServlet?LMSG_CABINET=Docs_and_Resource_Ctr&LMSG_CONTENT_FILE=White_Papers/Software-Defined_Radio_Base_Stations.pdf&lu_lang_code=en_WW. [Accessed Agosto 2015].

[7] M. Puerto González, Estudio piloto de los demoduladores de la serie RTL de Realtek para la Radio Definida por Software, Sevilla: Universidad de Sevilla, 2014.

[8] "RTL-SDR blog," 2015. [Online]. Available: http://www.rtl-sdr.com/. [Accessed Julio 2015].

[9] GreatScottGadgets, "HackRF," 2015. [Online]. Available: https://greatscottgadgets.com/hackrf/. [Accessed Julio 2015].

[10] nuand, "BladeRF," 2015. [Online]. Available: http://nuand.com/. [Accessed Julio 2015].

[11] Ettus Research, "USRP Networked Series," 2015. [Online]. Available: http://www.ettus.com/product/category/USRP-Networked-Series. [Accessed Julio 2015].

[12] National Instruments, "USRP RIO," 2015. [Online]. Available: http://sine.ni.com/nips/cds/view/p/lang/es/nid/212991. [Accessed Julio 2015].

[13] IEEE, "IEEE Spectrum. Samsung, Nokia Show 5G Tech at NI Week," 2015. [Online]. Available: http://spectrum.ieee.org/tech-talk/at-work/test-and-measurement/samsung-nokia-show-5g-tech-at-ni-week. [Accessed Agosto 2015].

[14] The Wireless Innovation Forum, "Software Defined Radio," 2014. [Online]. Available:

http://www.wirelessinnovation.org/assets/documents/SoftwareDefinedRadio.pdf. [Accessed Julio 2015].

[15] Ettus Research, "Ettus Research RFX2400 Daughterboard," 2015. [Online]. Available: http://www.ettus.com/product/details/RFX2400. [Accessed Julio 2015].

[16] Ettus Research, "Ettus Research Antenna VERT2450," 2015. [Online]. Available: http://www.ettus.com/product/details/VERT2450. [Accessed Julio 2015].

[17] Ettus Research, "Ettus Research SMA Cable," 2015. [Online]. Available: http://www.ettus.com/product/details/SMA-SMA. [Accessed Julio 2015].

[18] Ettus Research, "Ettus Research USB," 2015. [Online]. Available: http://www.ettus.com/product/details/USB-CBL. [Accessed Julio 2015].

[19] U. Mengali and A. N. D'Andrea, Synchronization Techniques for Digital Receivers, New York: Springer Science Business Media , 1997.

[20] Nutaq, "Implementation of Gardner symbol timing recovery in System Generator," 2014. [Online]. Available: http://nutaq.com/en/blog/implementation-gardner-symbol-timing-recovery-system-generator. [Accessed Julio 2015].

[21] J. G. Proakis and M. Salehi, Digital Communications, New York: McGraw-Hill International Edition, 2008.

[22] Nutaq, "Symbol timing recovery methods for digital I/Q demodulator," 2014. [Online]. Available: http://nutaq.com/en/blog/symbol-timing-recovery-methods-digital-iq-demodulator. [Accessed Julio 2015].

[23] Tektronix, "Tektronix Eye Diagram," 2010. [Online]. Available: www.tek.com/dl/65W_26042_0_Letter.pdf. [Accessed Julio 2015].

[24] W. Tuttlebee, Software Defined Radio: Origins, driver and international perspectives, New York: Wiley, 2002.

[25] M. Rice, Digital Communications. A Discrete-Time Approach, New Jersey: Pearson, 2009.

[26] Ettus Research, "Ettus Research Products," 2015. [Online]. Available: http://www.ettus.com/product/details/USRPPKG. [Accessed Julio 2015].

[27] Wikipedia, "USRP," 2015. [Online]. Available: https://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral. [Accessed Julio 2015].

[28] Youtube, "Matt Ettus USRP/RFNoC Update," 2014. [Online]. Available: https://www.youtube.com/watch?v=ZH4ix-yr7f0. [Accessed Julio 2015].

[29] Ettus Research, "Matt Ettus Interview," Enero 2015. [Online]. Available: http://www.ettus.com/blog/2015/01/an-interview-with-the-inventor-of-usrp-on-the-10th-birthday. [Accessed Julio 2015].