ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE COMUNICACIÓN INALÁMBRICA EN TIEMPO REAL BASADA EN OFDM, QUE CONTENGA DIFERENTES TIPOS DE MAPEO, REFORMADO DE PULSO Y CORRECCIÓN DE ERRORES, EMPLEANDO EQUIPOS SDR.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

CEVALLOS CRUZ BRYAN ALEXANDER

DIRECTOR: PhD. ROBIN GERARDO ÁLVAREZ RUEDA

Quito, mayo 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Bryan Alexander Cevallos Cruz, bajo mi supervisión.

_____

PHD. ROBIN ÁLVAREZ RUEDA

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Bryan Alexander Cevallos Cruz, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas Vigentes.

_____

BRYAN ALEXANDER CEVALLOS CRUZ

DEDICATORIA

Este trabajo lo dedico a mis padres Hugo y Guadalupe, a mi hermano Andrés, a mis primos Eddy, Adriana y Eduardo, quienes han sido la parte esencial a lo largo de mi carrera universitaria.

Bryan Cevallos

AGRADECIMIENTO

A mi madre Guadalupe, que me ha entregado su amor incondicional, siempre creyendo en mis capacidades, apoyándome en los objetivos que me he propuesto y ha sabido guiarme para ser mejor persona cada día. Gracias por nunca rendirte a pesar de las circunstancias.

A mi padre Hugo, que ha sido un ejemplo de carácter, me ha apoyado cada día de mi vida para poder cumplir mis sueños y me ha demostrado el amor de padre a su manera.

A mi hermano Andrés, que ha estado conmigo a lo largo de mi vida, siendo un gran apoyo en todo momento y siempre llenándome de orgullo de ser su hermano

A mis primos Eddy, Adriana y Eduardo que han conformado mi segundo hogar durante mi carrera universitaria. Gracias por guiarme y motivarme a crecer como persona cada día.

A mi abuelita Magdalena que ha sido un ejemplo de fortaleza, sabiduría y absoluta bondad.

A mis padrinos Oswaldo y Aída que me han cuidado, apoyado, aconsejado y protegido desde niño.

A mi tutor que ha sabido guiarme con paciencia y vocación a lo largo de este trabajo.

Bryan Cevallos

ÍNDICE DE CONTENIDO

RESUMEN

En este trabajo se desarrolla un prototipo de comunicación inalámbrica en tiempo real, utilizando Matlab y equipos de Radio Definida por Software (SDR). El prototipo implementa de manera práctica la técnica de transmisión OFDM (Multiplexación por División de Frecuencia Ortogonal), la técnica de reformado de pulso y corrección de errores mediante codificación convolucional. Cada etapa se aborda iniciando por la parte teórica, seguido de la implementación en Matlab y mostrando los resultados parciales correspondientes. Luego de abordar todas las etapas, se describe el prototipo implementado en las interfaces gráficas.

El prototipo permite variar las técnicas de mapeo o modulación digital (BPSK, QPSK, 16-QAM y 64-QAM) y evaluar el efecto de éstas en la tasa de bits errados (BER). También brinda la posibilidad de activar o desactivar la técnica de reformado de pulso y de corrección de errores, mostrando en tiempo real su efecto en el diagrama de constelaciones, en la información recibida y en el BER obtenido, lo cual permite que el prototipo tenga un carácter didáctico. Con un mismo esquema de mapeo, el reformado de pulso y/o la corrección de errores permiten obtener una mejora en el BER, siendo que utilizar las dos técnicas en conjunto resulta en la menor tasa de errores. Finalmente, debido a la poca información existente a nivel mundial, este trabajo constituye un punto de desembocadura de algunos trabajos anteriores y por ello el nivel de detalle genera gran cantidad de páginas, pero asegura el entendimiento de cada una de las etapas evitando posibles errores que serían fatales. Para futuros trabajos, este será la referencia fundamental y ya no habrá necesidad de volver a describir está parte básica.

PALABRAS CLAVE: SDR en tiempo real, corrección de errores, reformado de pulso, OFDM.

ABSTRACT

In this work, a real-time wireless communication prototype is developed, by using Matlab and Software Defined Radio (SDR) devices. The practical prototype implements OFDM (Orthogonal Frequency Division Multiplexing) transmission method, the pulse shaping technique and error correction through convolutional coding. Each stage is approached starting with the theoretical part, followed by the implementation in Matlab and showing the corresponding partial results. After approaching all the stages, the prototype implemented in the graphic interfaces is described.

The prototype allows to vary the mapping scheme (BPSK, QPSK, 16-QAM and 64-QAM) and to evaluate their effect on the bit error rate (BER). It also offers the possibility of activating or deactivating the pulse shaping and error correction technique, showing in real time its effect on the constellation diagram, on the information received and on the obtained BER, which allows the prototype to have a didactic character. With the same mapping scheme, pulse shaping and/or error correction allow to obtain an improvement in the BER, and by using the two techniques together results in the lowest error rate. Finally, due to the little information that exists worldwide, this work constitutes a point of lead for some previous works and therefore the level of detail generates a large number of pages, but ensures the understanding of each of the stages avoiding possible errors that they would be fatal.

KEY WORDS: real-time SDR, error correction, pulse shaping, OFDM.

1. INTRODUCCIÓN

La comunicación inalámbrica consiste en el envío y recepción de información de un punto a otro, utilizando un medio no guiado y ondas electromagnéticas. A lo largo del tiempo, las comunicaciones inalámbricas han tenido un largo desarrollo, pasando por varios métodos y sistemas inalámbricos que han florecido y otros tantos que han desaparecido [1]. Actualmente, este tipo de comunicación es ampliamente utilizada en diversos ámbitos del día a día de las personas, por ejemplo; la telefonía móvil, el sistema de posicionamiento global (GPS), Bluetooth y WI-FI. La principal ventaja de la comunicación inalámbrica es la movilidad.

Al medio por donde se propagan las ondas electromagnéticas que contienen información se lo denomina canal inalámbrico. El canal inalámbrico es por naturaleza dinámico e impredecible [2]. Además, el canal está sujeto a varios efectos como la dispersión, ruido, distorsión e interferencia. Estos efectos generan errores en la información recibida. Existen varios métodos y técnicas que permiten reducir la cantidad de errores en la comunicación inalámbrica.

La Radio Definido por Software (SDR) es una tecnología de comunicación inalámbrica, que surgió con el objetivo de trasladar los problemas que se pueden presentar en el hardware hacia la parte del software [3]. Esto permite que los usuarios puedan desarrollar prototipos de comunicaciones inalámbricas para diferentes sectores, sin tener que invertir en diferente hardware. Los dispositivos SDR se pueden encontrar en diferentes costos, dependiendo para qué estén destinados. Los sectores en donde se utiliza la tecnología SDR son: el sector militar, telefonía celular, el sector de entusiastas radioaficionados y sector educativo, entre otros. Actualmente, para el sector educativo se pueden encontrar equipos SDR de bajo costo, que resultan útiles para demostrar de manera real el comportamiento de varios sistemas de comunicación inalámbrica.

En este documento se desarrolla un prototipo de comunicación inalámbrica en tiempo real, que utiliza equipos SDR orientados al sector académico. Este prototipo se desarrolla para implementar de manera práctica diferentes técnicas y algoritmos empleados para transmitir y recibir a través del canal inalámbrico. El prototipo de comunicación inalámbrica incluye la técnica de modulación OFDM (Multiplexación por División de Frecuencia Ortogonal) que evita varias alteraciones debidas al canal inalámbrico. Para reducir la cantidad de errores se utiliza un algoritmo para corrección de errores a nivel de bit y un algoritmo de reformado de pulso. Además, para detectar la trama OFDM en recepción se utiliza el preámbulo empleado en el estándar IEEE 802.11a.

Se espera que la parte teórica abordada en este documento y el prototipo implementado puedan servir tanto para estudiantes como a profesores, para que puedan abordar las comunicaciones inalámbricas de manera práctica y también contribuir al estudio del SDR.

1.1. OBJETIVOS

El objetivo general de este proyecto técnico es desarrollar un prototipo de comunicación inalámbrica en tiempo real basado en OFDM, que contenga diferentes tipos de mapeo, reformado de pulso y corrección de errores, empleando equipos SDR.

Los objetivos específicos son:

- Estudiar las características relevantes de los dispositivos SDR Adalm Pluto.
- Estudiar los algoritmos asociados a las etapas de transmisión y recepción.
- Implementar el sistema de transmisión para que funcione en tiempo real utilizando una interfaz gráfica en Matlab y dispositivos SDR Adalm Pluto.
- Implementar el sistema de recepción para que funcione en tiempo real utilizando una interfaz gráfica en Matlab y dispositivos SDR Adalm Pluto.
- Realizar pruebas para analizar el funcionamiento del prototipo de comunicación inalámbrica.

1.2. ALCANCE

En este trabajo de titulación se desarrolla un prototipo de comunicación inalámbrica en tiempo real, haciendo uso de dos dispositivos SDR Adalm Pluto: uno como transmisor y otro como receptor. El prototipo final se implementa en interfaces gráficas de Matlab y se transmiten únicamente archivos de texto de extensión .txt. El usuario del prototipo puede realizar la configuración de equipos SDR, del procesamiento de datos, de la transmisión de datos, de la recepción de datos y de la recuperación de datos dentro de dichas interfaces gráficas.

Dentro de los bloques constitutivos del sistema se puede utilizar corrección de errores a nivel de bit, escoger un esquema de mapeo, utilizar la técnica de reformado de pulso y utilizar la técnica de transmisión OFDM. También se emplea el preámbulo IEEE 802.11a para detectar la trama OFDM, corregir el desplazamiento en frecuencia que sufren las portadoras, identificar el inicio exacto de la trama OFDM y corregir la alteración en fase. Puesto que el prototipo de comunicación transmite, recibe y procesa los datos del canal inalámbrico en tiempo real, no se pueden mostrar los resultados de cada etapa. Por esta

razón en este documento también se implementan las etapas en scripts, que permiten observar los resultados parciales de cada etapa.

El prototipo de comunicación se divide en tres interfaces gráficas; una principal, una para transmisión y otra para recepción. La interfaz gráfica principal permite realizar las siguientes acciones:

- Configurar los equipos SDR Adalm Pluto, mediante el paquete de Matlab llamado *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"*.
- Acceder a la interfaz gráfica de transmisión.
- Acceder a la interfaz gráfica de recepción.

La interfaz gráfica de transmisión permite:

- Seleccionar y abrir un archivo de texto.
- Mostrar el archivo de texto a transmitir.
- Activar o desactivar la corrección de errores.
- Seleccionar una técnica de mapeo y mostrar su diagrama de constelación.
- Activar o desactivar el reformado de pulso.
- Procesar los datos a transmitir con un único botón.
- Seleccionar un intervalo de tiempo para transmitir recursivamente el archivo de texto procesado.
- Transmitir los datos de forma repetitiva cada cierto tiempo seleccionado.
- Detener el proceso de transmisión.

La interfaz gráfica de recepción permite:

- Activar o desactivar el reformado de pulso.
- Seleccionar una técnica de demapeo.
- Activar o desactivar la corrección de errores.
- Seleccionar y abrir un archivo de texto para compararlo con los datos recuperados y obtener un valor de BER.
- Observar el texto del archivo original.
- Seleccionar un umbral para la detección de la información recibida.
- Iniciar el proceso de recepción en tiempo real, tal que se reciban y procesen datos del canal inalámbrico de manera repetitiva.
- Graficar el diagrama de constelación de los datos recibidos.
- Mostrar el texto recuperado del canal inalámbrico.

- Mostrar el valor del BER.
- Detener el proceso de recepción.

1.3. MARCO TEÓRICO

1.3.1. COMUNICACIÓN INALÁMBRICA

La comunicación inalámbrica consiste en transmitir información entre dos terminales de radio, utilizando ondas electromagnéticas en un medio no guiado. Este medio se lo suele llamar canal inalámbrico o interfaz aire, ya que por lo general la comunicación inalámbrica se realiza a través del aire. El canal inalámbrico es dinámico e impredecible, cosa que dificulta el análisis de un sistema de comunicación inalámbrica [2]. Actualmente, este tipo de comunicación es ampliamente utilizada en diversos ámbitos del día a día de las personas, principalmente por la movilidad que brinda a los usuarios.

En la comunicación inalámbrica se puede realizar la transmisión en modo simplex, half duplex y full duplex [1]. En el modo simplex se transmiten los datos en una sola dirección. En el modo half duplex se transmite en forma bidireccional, pero solamente en un sentido a la vez. En el modo full duplex se transmite en forma bidireccional en los dos sentidos a la vez.

1.3.2. RADIO DEFINIDA POR SOFTWARE (SDR)

En la tecnología SDR (Software Defined Radio), varias o todas las funciones de la capa física de una radio son definidas en software [4]. Con la tecnología SDR es posible trasladar los problemas de hardware hacia el software [3]. Esto permite reducir los costos de desarrollo y la posibilidad de reutilizar equipos SDR en sistemas que tienen arquitecturas diferentes. Los dispositivos SDR se pueden encontrar en el mercado en varios costos, dependiendo para el sector que estén destinados. Por lo general los dispositivos SDR para el sector educativo son de bajo costo, tal es el caso del módulo de aprendizaje activo **Adalm Pluto**.

Los dispositivos SDR orientados al aprendizaje permiten comprender el comportamiento real de las comunicaciones inalámbricas de una manera sencilla, ya que mediante el software se puede lograr una gran variedad de prototipos con diferentes características.

1.3.2.1. Dispositivo SDR Adalm Pluto

El módulo de aprendizaje ADALM PLUTO de Analog Devices Inc. es una herramienta que permite a los usuarios aplicar los conceptos de radiofrecuencia y comunicaciones inalámbricas. El dispositivo se puede utilizar para interactuar con señales de radiofrecuencia en conjunto con: Matlab, Simulink, Python, GNU Radio, C, C++, Windows, Linux, Mac o Raspberry Pi [5]. El equipo SDR se conecta a un computador a través de una interfaz USB 2.0.

Figura 1.1. Dispositivo SDR Adalm Pluto, con el cable USB 2.0 y dos antenas conectadas.

El equipo SDR Adalm Pluto tiene un conector para transmisión y otro de recepción, lo que le permite trabajar en modo full dúplex. También puede trabajar en modo half dúplex y simplex. El dispositivo cuenta con el transceptor AD9363 que permite trabajar con señales de radiofrecuencia en el rango de los 325 MHz a los 3.8 GHz [5]. Sin embargo, hay que considerar el rango de frecuencia de operación de las antenas que se conectan al equipo, para que ambas partes estén de acuerdo y pueda funcionar el sistema. En la Tabla 2.1 se muestran las características del equipo SDR Adalm Pluto.

Para utilizar el dispositivo Adalm Pluto con Matlab y Simulink es necesario el paquete de soporte *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"*. En Matlab se puede implementar interfaces gráficas o scripts para desarrollar prototipos que permitan utilizar el o los dispositivos Adalm Pluto. El paquete de soporte permite variar algunos parámetros de configuración del SDR Adalm Pluto cuando éste se encuentre en funcionamiento, a esta utilidad se los suele llamar *Over-The-Air*.

1.3.3. CORRECCIÓN DE ERRORES A NIVEL DE BIT

En este trabajo se implementa solamente la corrección de errores a nivel de bit utilizando códigos convolucionales del tipo FEC (Forward Error Correction). Las técnicas FEC

generan datos adicionales para ser enviados junto con la información, estas técnicas no solicitan retransmisiones al emisor. La codificación y decodificación convolucional, en conjunto, permiten corregir errores a nivel de bit. El proceso de codificación convolucional se implementa en transmisión, mientras que el decodificador se implementa en recepción, para poder recuperar los bits de información.

En el transmisor se codifica convolucionalmente los bits de entrada utilizando el diagrama de trellis. En recepción se decodifica convolucionalmente los bits utilizando el algoritmo de Viterbi, que utiliza el diagrama de trellis para buscar el camino más probable dentro de una secuencia de bits. El diagrama de trellis tiene forma de una red y permite observar la evolución de cada estado de codificación. La codificación convolucional es un sistema con memoria, ya que los bits a la salida del sistema se determinan mediante operaciones lógicas en el bit actual y en un grupo de bits anteriores [6].

La parte teórica se complementa en el capítulo 2 en las etapas correspondientes al codificador convolucional y al decodificador convolucional.

1.3.4. REFORMADO DE PULSO

El reformado de pulso consiste en aplicar un filtro a cada pulso de una señal o de una secuencia de datos. Esto con el objetivo de contrarrestar la interferencia entre símbolos (ISI) al hacer que la señal se vuelva más suave, sin cambios bruscos. El filtrado que se utiliza para implementar esta técnica se reparte entre transmisión y recepción: en ambos se utiliza un filtro raíz cuadrada de coseno elevado (SRRC). El resultado de utilizar dos filtros SRRC da como resultado en un filtro coseno elevado (RC).

1.3.5. OFDM

La Multiplexación por División de Frecuencia Ortogonal (OFDM) consiste en dividir un flujo de datos de entrada en múltiples subportadoras ortogonales, con el objetivo de evitar varias alteraciones introducidas por el canal inalámbrico. Las alteraciones que se evitan al utilizar OFDM son: la interferencia entre símbolos (ISI), la interferencia entre portadoras (ICI), el desvanecimiento selectivo en frecuencia y el multitrayecto.

En OFDM cada símbolo se compone de: subportadoras de datos, subportadoras pilotos, subportadoras de guarda y una subportadora DC [7]. Las **portadoras de guarda** permiten evitar la interferencia de canal adyacente (ACI). Las **portadoras piloto** se utilizan para corregir en recepción; desviaciones en fase, desviaciones en frecuencia y los efectos del canal inalámbrico [4]. Es decir, en cada símbolo OFDM recibido se realiza la ecualización de las portadoras de datos, utilizando las portadoras piloto.

Las portadoras de datos contienen la información previamente modulada digitalmente. El estándar WLAN IEEE 802.11A establece que los tipos de modulación a utilizar para las portadoras de datos de OFDM son BPSK, QPSK, 16-QAM y 64-QAM [8]. Puede darse el caso en que las portadoras de datos sean previamente moduladas digitalmente y aplicadas la técnica de **reformado de pulso**.

Considerando que ya se han mencionado las técnicas de modulación digital a utilizar, en la transmisión OFDM se tiene las siguientes etapas:

- **Creación del símbolo OFDM:** ensambla cada símbolo de acuerdo a la estructura de subportadoras de datos, piloto, guarda y DC.
- **IFFT:** corresponde al modulador OFDM y consiste en convertir los símbolos OFDM del dominio de la frecuencia al dominio del tiempo. Esta etapa aplica la IDFT (Transformada Discreta de Fourier Inversa) a través de la IFFT (Transformada Rápida de Fourier Inversa).
- **Prefijo cíclico:** consiste en copiar un número de muestras *N* del final del símbolo OFDM resultante de la IFFT al inicio del mismo símbolo. Esto con el objetivo de evitar el ISI.

En la parte de recepción OFDM se tienen las siguientes etapas:

- **Eliminación el prefijo cíclico:** elimina las primeras muestras *N* muestras de cada símbolo OFDM.
- **FFT:** corresponde al demodulador OFDM y consiste en convertir los símbolos OFDM del dominio del tiempo al dominio de la frecuencia. Esta etapa aplica la DFT (Transformada Discreta de Fourier) a través de la FFT (Transformada Rápida de Fourier).
- **Estimación del canal y ecualización:** utiliza las portadoras piloto de cada símbolo para estimar los efectos del canal inalámbrico y ecualizar cada símbolo por separado.
- **Recuperación datos del símbolo OFDM:** extrae las portadoras de datos de cada símbolo OFDM. Estas portadoras recuperadas están moduladas con esquemas BPSK, QPSK, 16-QAM y 64-QAM.

1.3.6. PREÁMBULO IEEE 802.11A

Este preámbulo se antepone a los símbolos OFDM serializados, consiste en una sección corta o LSTF (Legacy Short Training Field) y en una sección larga o LLTF (Legacy Long Training Field). La **sección corta** se compone de 10 copias repetidas de una secuencia

corta y cada secuencia corta se compone de 16 muestras, es decir la longitud de la sección LSTF es igual a 160. La **sección larga** se compone de 2 secuencias largas y de un prefijo cíclico. Una secuencia larga se compone de 64 muestras y el prefijo cíclico tiene 32 muestras, es decir que la longitud de la sección LLTF es igual a 160. Entonces, la longitud total de preámbulo de es 320 muestras.

En recepción el preámbulo IEEE 802.11a permite realizar las siguientes funciones:

- **Detectar la trama OFDM:** permite detectar si los datos recibidos corresponden o no a un bloque de datos OFDM.
- **Corregir el desplazamiento de frecuencia (CFO):** permite estimar y corregir el desplazamiento de frecuencia del bloque de datos recibido.
- **Sincronizar cada símbolo:** permite determinar el inicio exacto de los datos OFDM.
- **Corregir el desplazamiento de fase:** Estima y corrige el desplazamiento de fase del bloque de datos sincronizado.

Estas etapas se implementan antes de las etapas de recepción de OFDM vistas en la sección anterior.

2. METODOLOGÍA

2.1. INSTALACIÓN DE SOFTWARE

El prototipo de comunicación inalámbrica utiliza dos computadores y dos equipos SDR Adalm Pluto, por lo que es necesario realizar la instalación de software en ambos equipos. El programa de software necesario para la implementación del prototipo es Matlab.

Para el desarrollo de este trabajo se instaló Matlab R2021a, por lo que se recomienda utilizar esta versión si se desea replicar el prototipo. La instalación de Matlab R2021a se ha llevado a cabo utilizando los servicios que provee la DGIP de la Escuela Politécnica Nacional. El paquete de Matlab que se utiliza para conectar los dispositivos SDR con este programa se llama *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"*: Este paquete es funcional desde la versión R2017a de Matlab hasta versiones posteriores, por lo que bajo ningún motivo se recomienda utilizar versiones de Matlab anteriores a R2017a.

2.1.1. INSTALACIÓN DE TOOLBOX PARA LA RADIO ADALM PLUTO

El paquete *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"* permite utilizar el dispositivo SDR Adalm Pluto para diseñar y verificar sistemas prácticos, en condiciones reales, con Matlab y Simulink [9]. Este paquete se puede instalar directamente desde la aplicación de Matlab.

El procedimiento para la instalación es el siguiente:

1) Abrir Matlab y ubicarse en la pestaña *HOME*.
2) En el apartado *ENVIROMENT* dar clic en *Add-Ons* y dar clic en *Get Hardware Support Packages*, como se muestra en la Figura 2.1.

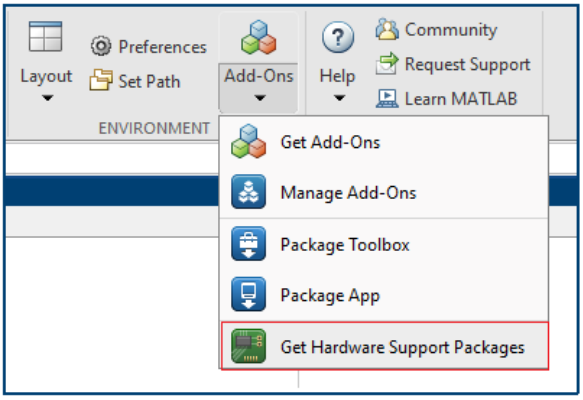


Figura 2.1. Obtener hardware para soporte de paquetes en Matlab.

3) Cuando se despliegue la ventana de *Add-On Explorer*, buscar el paquete *"Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"*. En la Figura 2.2 se muestra la búsqueda de este paquete.

Figura 2.2. Búsqueda del paquete de soporte en la ventana *Add-On Explorer*.

4) Dar clic en el paquete e instalar el mismo, luego aceptar los términos de la licencia y esperar a que la instalación concluya. Al finalizar la Instalación se obtiene el mensaje de la Figura 2.3.

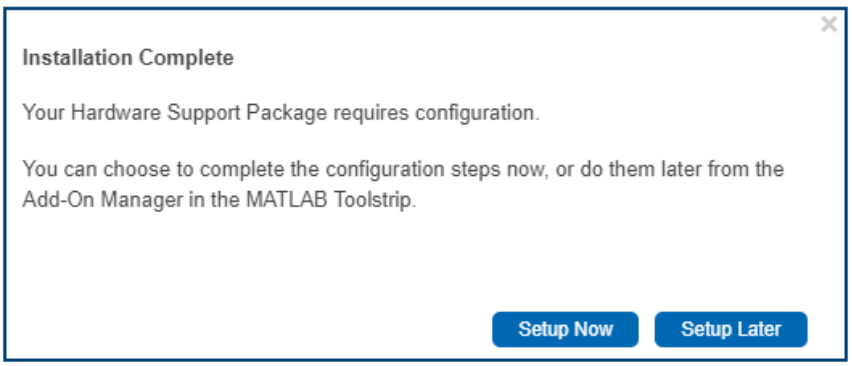


Figura 2.3. Mensaje al finalizar la instalación.

Cuando se completa la instalación del paquete, si se elige la opción *Setup Later* se dará por terminado todo el proceso de instalación. Si se selecciona la opción *Setup Now* se pedirá al usuario configurar el hardware del equipo SDR Adalm Pluto.

El procedimiento para configurar el hardware es el siguiente:

1) En la ventana que se despliega luego de la instalación del paquete (Figura 2.3) se presiona el botón *Setup Now*.

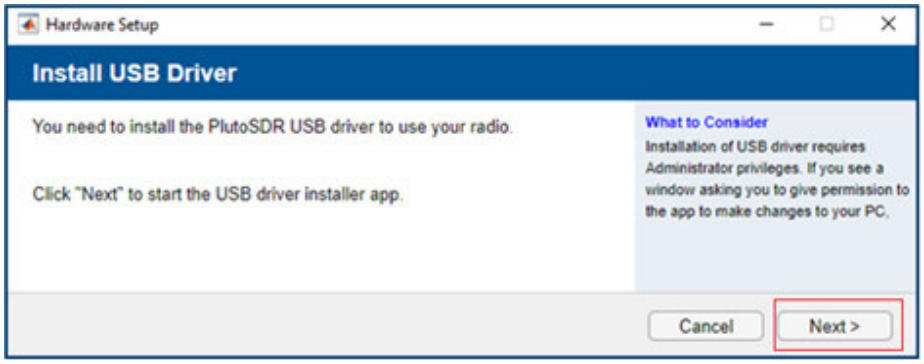


Figura 2.4. Primera ventana de la configuración de hardware.

2) Luego de que se despliegue la ventana de la Figura 2.4, se presiona el botón *Next* (se encierra con rojo). Después se abre una ventana para solicitar al usuario que se conecte el equipo Adalm Pluto.

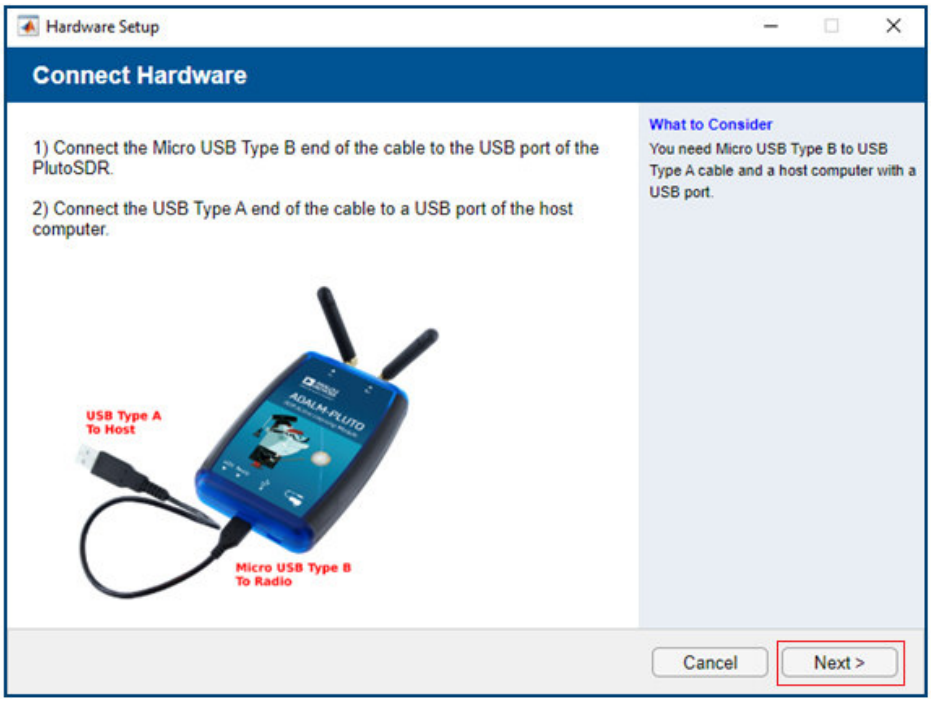


Figura 2.5. Solicitud de conexión de hardware.

3) El paso siguiente es conectar mediante el cable USB el equipo SDR al computador. Las dos antenas que vienen en el equipo deben estar conectadas. Luego se da clic en *Next*. En la Figura 2.6 se muestra un requerimiento de actualización del firmware para la radio Adalm Pluto que está conectada. El firmware es un programa o porción de código que controla los circuitos electrónicos de un dispositivo.

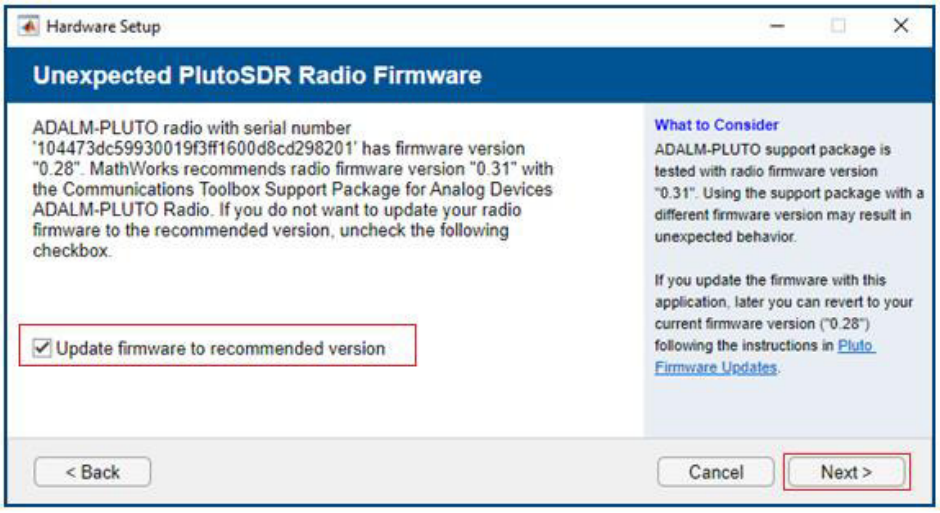


Figura 2.6. Ventana de requerimiento de actualización de firmware.

4) A continuación, señalar la opción para actualizar el firmware a la versión recomendada y dar clic en *Next*. Luego se pide al usuario que extraiga el dispositivo SDR, este mensaje se muestra en la Figura 2.7.

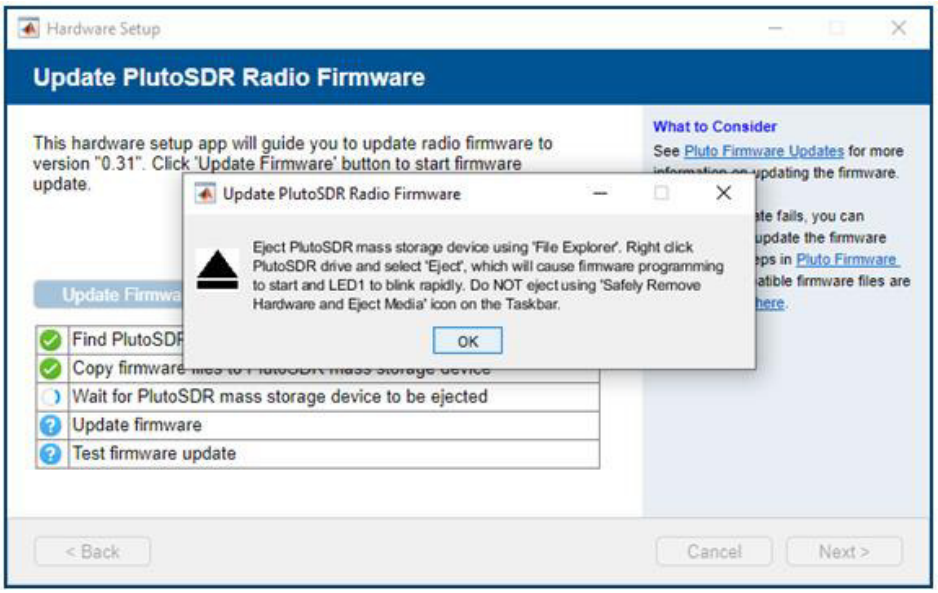


Figura 2.7. Mensaje para expulsar el equipo SDR

5) Antes de presionar *Ok* hay que expulsar el dispositivo SDR, tal como se muestra en la Figura 2.8. En donde se señala con azul la opción que se escoge.

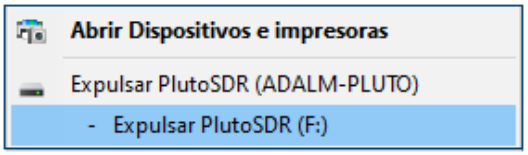


Figura 2.8. Expulsar equipo SDR.

6) Luego de expulsar se presiona en *OK* y se obtiene la gráfica de la Figura 2.9.

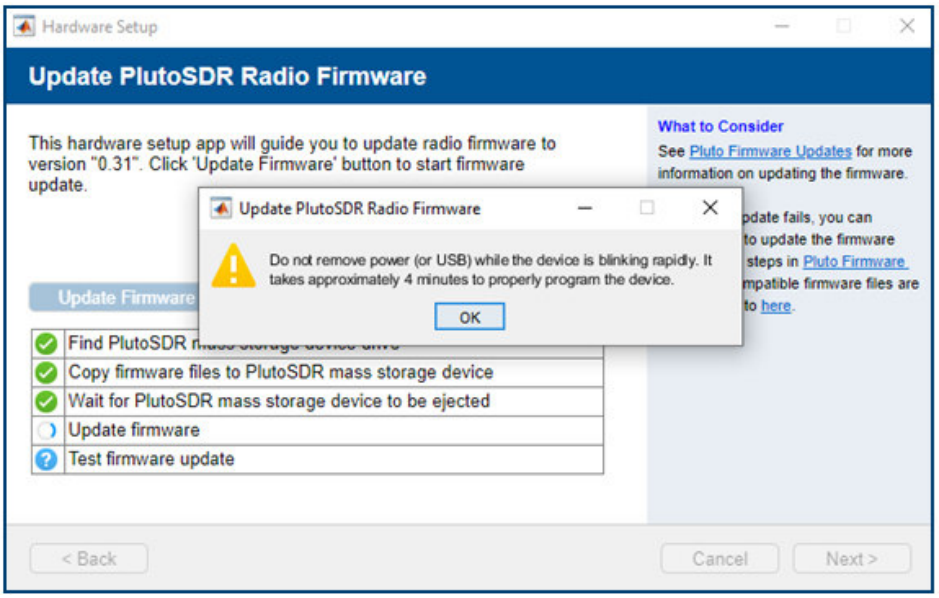


Figura 2.9. Mensaje de información para que el usuario espere.

7) Presionar *OK* y esperar a que el proceso de actualización y el test de actualización de firmware termine.

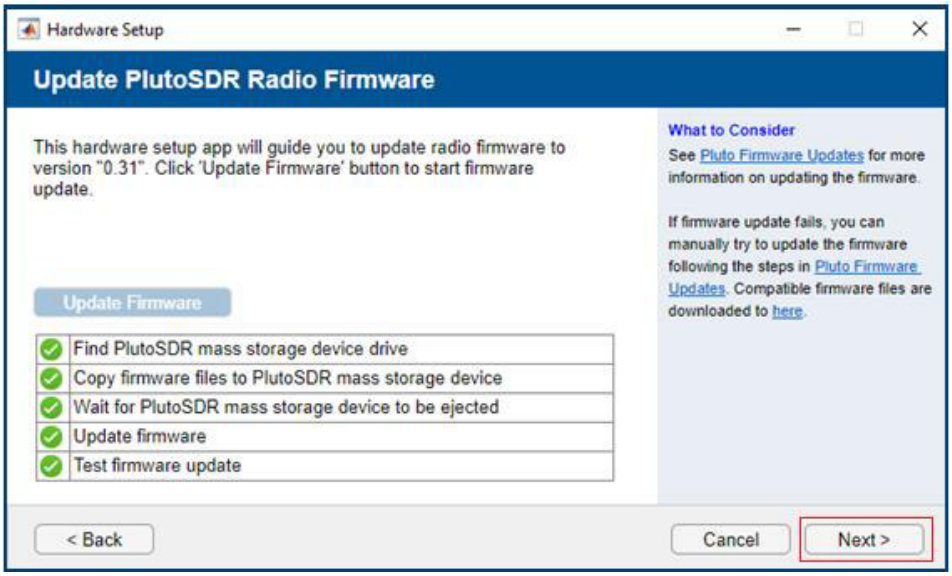


Figura 2.10. Firmware actualizado.

8) Luego se presiona en *Next* y en la ventana siguiente se presiona en *Test Connection*. En la Figura 2.11 se muestran los resultados cuando se pasan todas las pruebas de conexión.

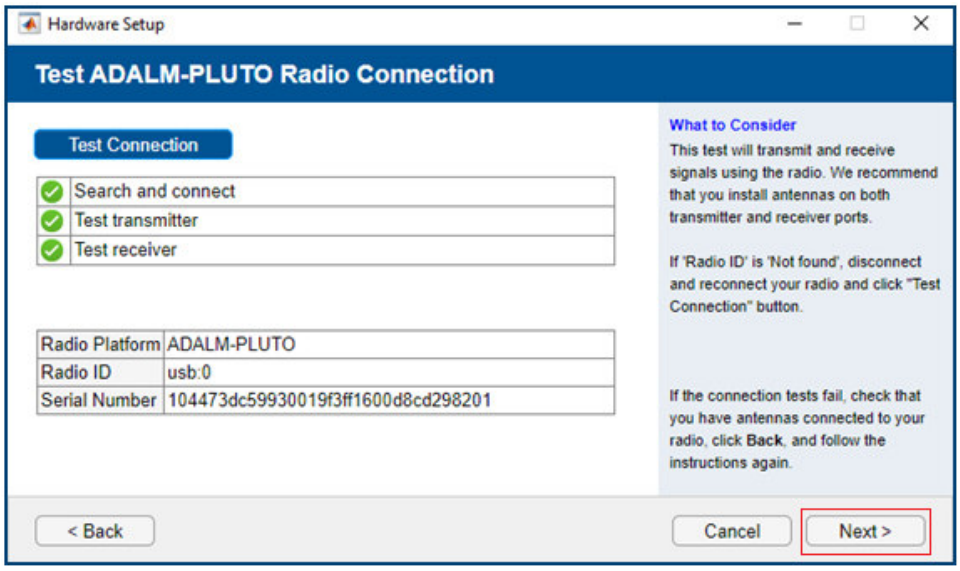


Figura 2.11. Test de conexión de la radio Adalm Pluto.

9) Después se presiona en *Next* y se abrirá la ventana final, esta ventana se muestra en la Figura 2.12. Luego se puede seleccionar una opción para mostrar ejemplos del paquete instalado y con el botón *Finish* se puede finalizar el proceso de configuración del Hardware.

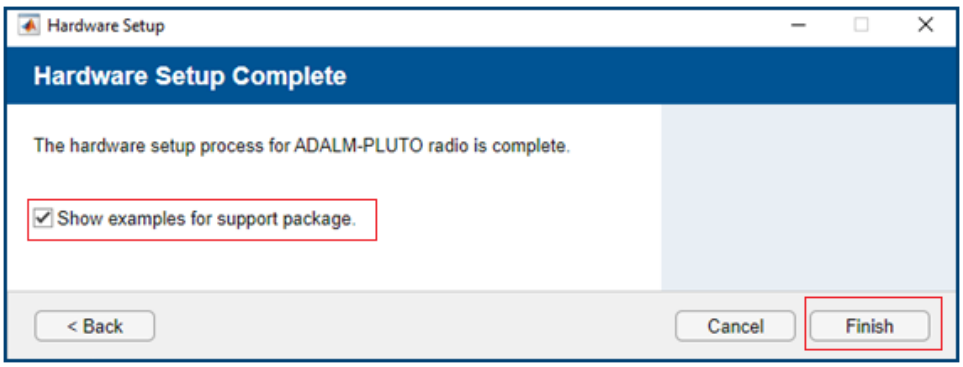


Figura 2.12. Ventana final de la configuración del hardware.

2.2. CONFIGURACIÓN DEL SDR ADALM PLUTO

La configuración de los dispositivos Adalm Pluto se realiza a través de objetos globales, un objeto para el transmisor y otro para el receptor. Las especificaciones del dispositivo SDR se muestran en la Tabla 2.1.

Tabla 2.1. Especificaciones principales del dispositivo SDR Adalm Pluto [5].

Especificaciones	Valores típicos
Entrada DC (USB)	4.5V a 5.5V
Frecuencia de muestreo ADC y DAC	65.2 ksps a 61.44 Msps
Resolución ADC y DAC	12 bits
Precisión de frecuencia	±25 ppm
Rango de Sintonización	325 MHz a 3800 MHz
Ancho de banda instantáneo	Hasta 20 MHz
Conectores	SMA 50 Ω
Modo de transmisión	Half duplex o Full duplex
Potencia de salida en Tx	7 dBm
Figura de ruido Rx	< 3.5 dB
Precisión de modulación en Tx y Rx	-34 dB (2%)
Blindaje RF	No
USB	2.0 On-the-Go
Temperatura	10°C a 40°C

El dispositivo Adalm Pluto incluye dos antenas una utilizada para transmisión y otra para recepción, el rango de frecuencia de las mismas es de 824 MHz a 894 MHz y de 1710 MHz a 2170 MHz [5]. El prototipo de comunicación inalámbrica utiliza estas antenas que se incluyen con los equipos Adalm Pluto.

La frecuencia de operación del Adalm Pluto se ve limitada por las antenas que incluyen los dispositivos.

2.2.1. IDEA A IMPLEMENTAR

Se busca que el prototipo de comunicación pueda ser utilizado de manera rápida, por lo que el dispositivo SDR se configura con valores predeterminados al abrir la **interfaz gráfica principal** del prototipo. El usuario no tiene la necesidad de establecer los parámetros de configuración, si no que puede acceder a las interfaces de transmisión o recepción desde la interfaz gráfica principal. Por otro lado, si el usuario desea variar los parámetros por defecto puede acceder a una interfaz gráfica de configuración. Además, la interfaz gráfica principal permite acceder a la interfaz de transmisión y recepción.

Para realizar las pruebas de funcionamiento del prototipo se utiliza dos computadores y dos equipos SDR Adalm Pluto de modo que se pueda variar la distancia sin inconvenientes. La interfaz gráfica principal se ejecuta independientemente de si se encuentre en el transmisor o en el receptor. La Figura 2.13 muestra el esquema de hardware del prototipo de comunicación inalámbrica.

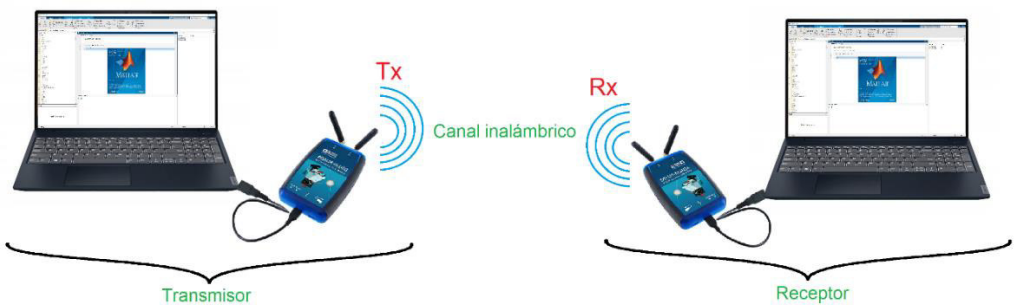


Figura 2.13. Esquema de hardware del prototipo de comunicación inalámbrica.

2.2.2. CONFIGURACIÓN DEL SDR ADALM PLUTO EN MATLAB

Para configurar los equipos SDR se utiliza un objeto del sistema para transmisión y otro para recepción. El objeto del sistema *comm.SDRTxPluto* es una fuente de señal que **transmite** datos a una radio Adalm Pluto [10]. El objeto del sistema *comm.SDRRxPluto* es una fuente se señal que **recibe** datos de una radio Adalm Pluto [11]. La Figura 2.14 muestra la interacción entre Matlab, el objeto para transmisión, el objeto para recepción y el hardware del equipo Adalm Pluto.

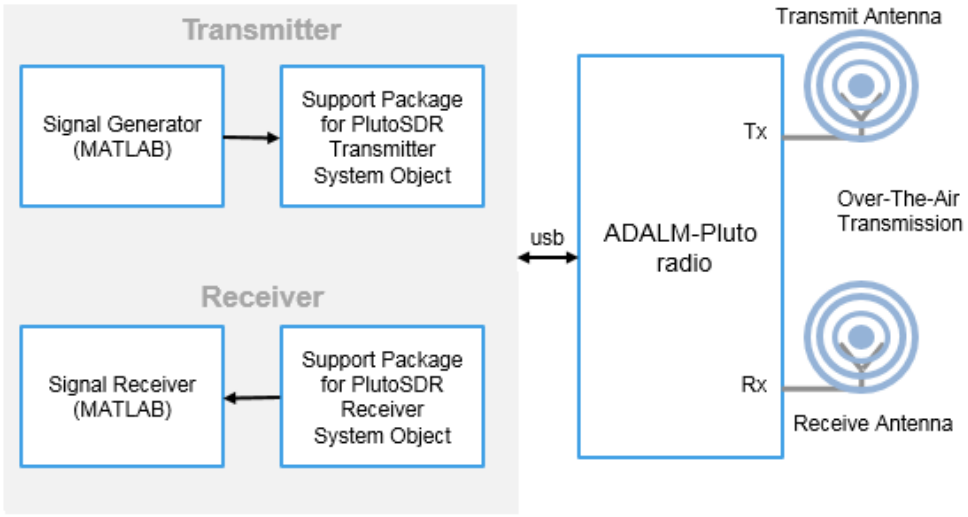


Figura 2.14. Interacción entre Matlab, los objetos del sistema, y el hardware de la radio [10], [11].

Para crear el objeto de transmisión *comm.SDRTxPluto* se utiliza otro objeto del sistema llamado *sdrtx()*. El cual tiene la siguiente sintaxis [10]:

```
txAdalm = sdrtx('Pluto',Name,Value)
```

En esta sintaxis la variable *txAdalm* almacena un objeto de sistema con la configuración especificada por los parámetros de entrada. En los parámetros de entrada la propiedad *Name* se especifica con el valor que puede tomar *Value*. Más adelante se detallan las propiedades de entrada.

- **DeviceName:** Es el nombre o modelo del dispositivo SDR, aunque actualmente la única entrada posible es 'Pluto', que hace referencia a los dispositivos SDR Adalm Pluto de Analog Devices.
- **RadioID:** Se refiere a un número de identificación de la radio, el cual está relacionado al puerto USB del computador en donde se ha conectado. Este parámetro puede tener como valor 'usb:0', 'usb:1', 'usb:2', etc. Además, el mismo es útil cuando se trabaja con varios dispositivos SDR en un solo computador. El identificador suele ser asignado según el orden de conexión a los puertos USB del computador, por ejemplo, al primer radio en conectarse le corresponde 'usb:0', al segundo 'usb:1' y así sucesivamente.
- **CenterFrecuency:** Ajuste de la frecuencia central de RF. Se especifica como un número escalar que va desde 70.0e6 hasta 6.0e9 (de 70 MHz a 6 GHz). El valor por defecto de este parámetro es de 2.4e9 (2.4 GHz). Si se deja con este valor, no estaría de acuerdo con los rangos de frecuencia de las antenas empleadas.
- **Gain:** Es la ganancia en dB, la cual se especifica con un número escalar que va de -89.75 hasta 0 [dB]. Además, la resolución es de 0.25 y el valor predeterminado de este parámetro es -10.
- **ChannelMaping:** Es una propiedad de solo lectura, siempre está seteada en 1 y su valor por defecto es precisamente 1.
- **BasebandSampleRate:** Es la frecuencia de muestreo en banda base en Hz, se especifica como con un escalar que va desde 65105 a 61.44e6 muestras por segundo (sps). El valor por defecto es 1.0e6.
- **ShowAdvanceProperties:** Es la opción para mostrar las opciones avanzadas se especifica como true o false (1 o 0). El valor por defecto es false.

Para crear el objeto de recepción *comm.SDRRxPluto* se utiliza otro objeto del sistema llamado *sdrrx()*. Este objeto se utiliza a través de la siguiente sintaxis [11]:

rxAdalm = sdrrx('Pluto',Name,Value)

En esta sintaxis la variable *rxAdalm* almacena un objeto de sistema con la configuración especificada por los parámetros de entrada. En los parámetros de entrada la propiedad *Name* se especifica con el valor almacenado en *Value*. Más adelante se detallan las propiedades de entrada.

- **DeviceName:** Es el nombre o modelo del dispositivo SDR, aunque actualmente la única entrada posible es 'Pluto'.
- **RadioID:** Se refiere a un número de identificación de la radio, el cual está relacionado al puerto USB del computador en donde se ha conectado. Este parámetro puede tener como valor 'usb:0', 'usb:1', 'usb:2', etc.
- **CenterFrecuency:** Ajuste de la frecuencia central de RF. Se especifica como un número escalar que va desde 70.0e6 hasta 6.0e9 (de 70 MHz a 6 GHz). Aquí también se debe configurar de acuerdo a la emplead a en el transmisor.
- **ChannelMapping:** Es una propiedad de solo lectura, siempre está seteada en 1.
- **GainSource:** Es la fuente de ganancia y se puede especificar como:
  - 'AGC Slow Attack' que se utiliza para señales que los niveles de potencia cambian lentamente, además es el valor por defecto.
  - 'AGC Fast Attack' que se utiliza para señales que los niveles de potencia cambian rápidamente.
  - 'Manual' que se utiliza para configurar de manera manual la ganancia con la propiedad Gain.
- **Gain:** Es la ganancia en dB del receptor de radio, se especifica como un número escalar de -4 a 71 y el valor predeterminado es 10. Esta propiedad puede ser usada solo si GainSource fue seteada en 'Manual'.
- **BasebandSample:** Es la frecuencia de muestreo en banda base en Hz, se especifica como con un escalar que va desde 65105 a 61.44e6 muestras por segundo (sps). El valor por defecto es 1.0e6.
- **OutputDataType:** Es el tipo de datos de la señal de salida y pueden ser:
  - 'int16' (predeterminado): entero de 16 bits con signo, rango de valores de -32768 a 32767 y emplea 16 bits por cada valor.
  - 'single': punto flotante de doble precisión, con un rango de valores de -1 a 1 y emplea 32 bits por cada valor.
  - 'double': punto flotante de precisión simple, con un rango de valores de -1 a 1 y emplea 64 bits por cada valor.
- **SamplesPerFrame:** Es el número de muestras por trama, se especifica como un entero positivo par que va desde 2 a 16,777,216 y el valor por defecto es de 20000. Los valores inferiores a 3660 pueden producir un rendimiento deficiente.
- **EnableBurstMode:** Opción para el modo de ráfaga, especificado como false o true, el valor predeterminado es false. Cuando se establece en true, se produce un conjunto de muestras contiguas sin desbordamiento, es decir que se almacena en un búfer un conjunto de muestras contiguas. Esta configuración puede ayudar a

simular modelos que no se pueden ejecutar en tiempo real. Si esta opción se establece en true se debe configurar la opción NumFramesInBurst.

- **ShowAdvancedProperties:** Es la opción para mostrar las opciones avanzadas se especifica como true o false (1 o 0) y el valor por defecto es false.
- **FrecuencyCorrection:** Es el valor de corrección de frecuencia en ppm (parts per million), se especifica como un escalar de -200 a 200 y su valor por defecto es 0. Esta propiedad corrige los cambios de frecuencia en los datos recibidos debido al desplazamiento de la frecuencia del oscilador local o la inexactitud de la frecuencia del reloj.

  La propiedad *FrecuencyCorrection* cambia la configuración de radio real para las propiedades de *BasebandSampleRate* y *CenterFrequency*. Además, la misma depende de que la propiedad *ShowAdvanceProperties* esté especificada como *true*.

2.2.2.1. Determinación de la frecuencia central

Como se mencionó antes, si bien el dispositivo Adalm Pluto tiene el rango de sintonización que va desde los 325 MHz hasta los 3.8 GHz, la frecuencia de operación se ve limitada por las antenas que incluyen los dispositivos. El rango de frecuencia de las antenas es de 824 MHz a 894 MHz y de 1710 MHz a 2170 MHz, por lo que la frecuencia central a utilizar debe estar entre estos dos rangos de frecuencia. Recordando que a menor frecuencia se tiene un mayor alcance, resulta una buena opción ubicar la propiedad *CenterFrecuency* (frecuencia central) entre el rango de 824 MHz a 894 MHz.

En el prototipo de comunicación inalámbrica se optó por utilizar una frecuencia central igual a 860 MHz.

2.2.2.2. Determinación de la frecuencia de muestreo

La propiedad *BasebandSample* (frecuencia de muestreo) en Matlab se puede configurar con valores de 65105 a 61.44e6 muestras por segundo (sps). Sin embargo, se debe determinar el valor máximo y mínimo de la frecuencia de muestreo, que puede utilizar el dispositivo Adalm Pluto en conjunto con Matlab. Para esto hay que tener en cuenta el tipo de datos que se va a transmitir, la tasa de transferencia de la interfaz USB del equipo, el ancho de banda instantáneo que proporciona el equipo Adalm Pluto y la técnica de transmisión OFDM.

Para determinar el valor máximo de la frecuencia de muestreo hay que considerar que los equipos SDR Adalm Pluto trabajan con la interfaz USB 2.0. Esta interfaz alcanza tasas de transferencia de hasta 480 Mbps, pero en la práctica este valor disminuye quedándose

alrededor de los 280 Mbps [12]. También se debe considerar el tipo de dato con el que se va a trabajar, en el presente trabajo se utilizan del tipo *single* que son datos de punto flotante de precisión simple, con un rango de valores de -1 a 1. Los datos tipo *single* emplea 32 bits por cada muestra o valor, como ya se describió en el parámetro *Outputdatatype* del objeto de recepción *sdrrx*. La siguiente ecuación sirve para determinar la frecuencia de muestreo máxima.

$$f_s * Nbits < C_{int} \tag{2.1}$$

Donde f_s es la frecuencia de muestreo, $Nbits$ es el número de bits del tipo de datos por cada muestra y C_{int} es la capacidad de la interfaz.

Los valores conocidos son:

$$Nbits = 32 \left[\frac{bits}{muestra}\right]$$

$$C_{int} = 280\ [Mbps] = 280 \left[\frac{Mbits}{segundo}\right]$$

Desarrollando la ecuación 2.1 se tiene:

$$f_s < \frac{C_{int}}{Nbits}$$

Reemplazando valores:

$$f_s < \frac{280 \left[\frac{M.bits}{segundo}\right]}{32 \left[\frac{bits}{muestra}\right]}$$

$$f_s < 8.75 \left[\frac{M.muestra}{segundo}\right]$$

$$f_s < 8.75\ Msps$$

Y finalmente se tiene que el valor de la frecuencia de muestreo tiene que ser menor a 8.75 Msps. Si se quisiera utilizar valores del tipo double, el valor de la frecuencia de muestreo debería ser menor a 4.375 Msps. Es decir que el límite superior de la frecuencia de muestreo disminuye, por lo que resulta mejor trabajar con valores del tipo single.

Para determinar el valor mínimo de la frecuencia de muestreo se utiliza el teorema de muestreo de Nyquist. El teorema dicta que, para recuperar una señal, la frecuencia de muestreo f_s debe ser mayor a dos veces el máximo componente de frecuencia de dicha señal. La siguiente ecuación representa el teorema de muestreo de Nyquist.

$$f_s > 2f_{max} \tag{2.2}$$

Como se indicó en la Tabla 2.1 el dispositivo SDR Adalm Pluto alcanza un ancho de banda instantáneo de 20 MHz. Considerando que se va a trabajar con la técnica de transmisión OFDM, en donde su estructura se compone de 64 portadoras en total. Los 20 MHz de ancho de banda se dividen para las 64 portadoras dando como resultado un ancho de banda de 312.5 kHz, para cada portadora. La frecuencia máxima es la mitad del ancho de banda de 312.5 kHz. Entonces de acuerdo a la ecuación 1.2 se tiene:

$$f_s > 2 * \left(\frac{312.5\ kHz}{2}\right)$$

$$f_s > 312.5\ ksps$$

El valor de la frecuencia de muestreo tiene que ser mayor a 312.5 ksps. El rango calculado de la frecuencia de muestreo va de 312.5 ksps a 8.75 Msps.

2.2.2.3. Determinación del tamaño del bloque de transmisión y recepción

El número de muestras por trama se especifica en el objeto del sistema de recepción con el parámetro *SamplesPerFrame*. Si bien este parámetro no se configura para el objeto del sistema de transmisión, el bloque de datos a enviar debe ser igual al mismo. Para determinar el número adecuado del parámetro *SamplesPerFrame*, hay que considerar el número de muestras que se va a enviar. Para calcular esto hay que tener en cuenta lo siguiente:

- Número de caracteres del archivo de texto a enviar (Nc).
- Bits utilizados para representar cada carácter (Bc).
- Tasa del codificador convolucional utilizado para la corrección de errores a nivel de bit (R_c).
- Técnica de Mapeo a utilizar. Específicamente el número de bits utilizados por cada símbolo mapeado (Nb).
- Muestras por símbolo en el reformado de pulso (Nsps).
- Tamaño del símbolo OFDM con prefijo cíclico (Lo).
- Datos por cada símbolo OFDM (Ld).
- Tamaño del preámbulo 802.11a (Lp)

Los parámetros listados anteriormente se explican a detalle en el capítulo de transmisión. Con la ecuación 2.3 se puede calcular el número de muestras a enviar $N_{muestras}$.

$$N_{muestras} = Nc * Bc * \frac{1}{R_c} * \frac{1}{Nb} * Nsps * \frac{Lo}{Ld} + Lp \quad (2.3)$$

En las interfaces gráficas del prototipo que se está desarrollando se busca que el usuario pueda observar el texto transmitido y recibido. Por lo tanto, un tamaño adecuado del archivo de texto es de hasta 1700 caracteres, es decir Nc=1700. Para representar cada carácter se utiliza 8 bits, es decir que Bc=8. La tasa del codificador convolucional R_c es igual a 1/2.

El valor Nb se encuentra en el denominador de la ecuación 2.3. Por lo tanto, para calcular el mayor número de muestras posibles que se va a obtener, hay que considerar la técnica de mapeo que utilice la menor cantidad de bits por cada símbolo. Esta técnica es el mapeo BPSK que utiliza 1 bit por cada símbolo, es decir que Nb=1. En el reformado de pulso Nsps es igual a 4. El valor de Lo es igual a 80 y Ld es igual a 40. El preámbulo 802.11a tiene una longitud de 320.

Evaluando la ecuación 2.3 se obtiene:

$$N_{muestras} = 1700 * 8 * \frac{1}{\frac{1}{2}} * \frac{1}{1} * 4 * \frac{80}{40} + 320 = 217920\ muestras$$

Entonces considerando un archivo de texto de un tamaño no muy grande (1700 caracteres) y el escenario en donde se tiene la mayor cantidad de muestras de datos luego del procesamiento. Se tiene que sin considerar los posibles rellenos que se lleven a cabo en las etapas de transmisión, el número máximo de muestras de datos en un bloque es de 217 920 muestras.

En recepción, el equipo SDR muestrea el canal tal que en cada iteración recupera un bloque de datos de tamaño igual al definido en el parámetro *SamplesPerFrame* del objeto de recepción. El valor máximo que puede tomar el parámetro es de 16 777 216, pero hay que considerar que, a mayor tamaño del bloque, mayor es el tiempo de procesamiento de los datos en recepción, por lo que no es eficiente utilizar un tamaño de bloque muy grande. El escenario ideal es que los datos lleguen contenidos dentro de un solo bloque. Pero mientras más se acerque el tamaño del bloque, al número de muestras de datos menos probable es que la información llegue en un solo bloque.

Con el objetivo de buscar que la información llegue **contenida dentro de un solo bloque** se establece que un tamaño adecuado para el mismo es de 800 000.

Considerar que a medida que aumenten los puntos en la constelación de la modulación digital, la longitud de los datos de información va a disminuir. Lo mismo ocurre si no se utiliza reformado de pulso y/o corrección de errores.

Nota: Si se requiere enviar bloques de información de un tamaño superior a 16 777 216 sería una mejor opción que el transmisor envíe todo el bloque de información y que en **recepción** se muestree en bloques de 800 000, pero no se debe limitar el número máximo de bloques de información que se almacenan para el procesamiento.

2.2.3. IMPLEMENTACIÓN EN MATLAB

En la sección anterior se explicó los fundamentos teóricos respecto a la configuración de equipos SDR Adalm Pluto con Matlab, a través de objetos del sistema. En esta sección se configuran un dispositivo SDR mediante un script de Matlab. En la Tabla 2.2 se muestran los parámetros de configuración del objeto del sistema para transmisión, una pequeña descripción del parámetro, las opciones o rangos que pueden tomar y el valor de configuración a implementar.

Tabla 2.2. Parámetros de configuración para el objeto sdrtx().

Parámetro	Descripción	Opciones	Valor
DeviceName	Modelo del dispositivo SDR	'Pluto'	'Pluto'
RadioID	Identificador de Interfaz USB (puerto) donde está conectado el SDR	'usb:0' (default) 'usb:1' 'usb:2', …	'usb:0'
CenterFrecuency	Frecuencia central de RF en Hz	2.4e9 (default) 70.0e6 a 6.0e9 (rango)	860e6
Gain	Ganancia en dB	-10 (default) -89.75 a 0, con una resolución de 0.25	-2
Baseband-SampleRate	Frecuencia de muestreo en banda base	1.0e6 (default) 65105 a 61.44e6 sps	2000e3
ShowAdvanced-Properties	Mostrar las opciones avanzadas	false (default) false (0) o true (1)	0

En la Tabla 2.3 se muestran los parámetros de configuración del objeto del sistema para recepción, una pequeña descripción del parámetro, las opciones o rangos que pueden tomar y el valor de configuración a implementar.

Tabla 2.3. Parámetros de configuración para el objeto sdrrx().

Parámetro	Descripción	Opciones	Valor
DeviceName	Modelo del dispositivo SDR	'Pluto'	'Pluto'
RadioID	Identificador de Interfaz USB donde está conectado el SDR	'usb:0' (default) 'usb:1' 'usb:2', …	'usb:0'
CenterFrecuency	Frecuencia central de RF en Hz	2.4e9 (default) 70.0e6 a 6.0e9 (rango)	860e6
GainSource	Fuente de ganancia:	'AGC Slow Attack': Señales con niveles de potencia que cambian lento. (default) 'AGC Fast Attack': Señales con niveles de potencia que cambian rápido. 'Manual': Para configurar la ganancia manualmente con el parámetro **Gain**.	'Manual'
Gain	Ganancia en dB	10 (default) -4 a 71 (rango)	20
Baseband-SampleRate	Frecuencia de muestreo en banda base	1.0e6 (default) 65105 a 61.44e6 sps	2000e3
OutputDataType	Tipo de datos de la señal de salida	'int16': Enteros de 16 bits con signo. 'double': Punto flotante de doble precisión, 64 bits por muestra 'single': Punto flotante de precisión simple, 32 bits por muestra	'single'
SamplesPer-Frame	Número de muestras por trama (tamaño de trama), entero positivo	20000 (default) 2 a 16777216 (rango)	800000
ShowAdvanced-Properties	Mostrar las opciones avanzadas	false (0) o true (1)	1
Frequency-Correction	Valor de corrección de frecuencia en ppm.	0 (default) -200 a 200 (rango)	0

Antes de realizar las configuraciones a través de los objetos del sistema de transmisión y recepción es necesario conocer si algún dispositivo está conectado al computador por una interfaz USB 2.0. Para determinar si un dispositivo SDR se encuentra conectado, se utiliza la función ***findPlutoRadio***, que devuelve una estructura con el ID de la radio y el número de serie de todos los dispositivos SDR Adalm Pluto conectados mediante una interfaz USB al host. Esto se puede ver en Figura 2.15 literal a y b. El tamaño de la estructura es proporcional al número de dispositivos Adalm Pluto conectados. Si la función no encuentra

ningún dispositivo SDR conectado, la matriz está vacía (ver Figura 2.15 c) y por ende si se obtiene la longitud de la estructura, será cero.

Figura 2.15. Función ***findPlutoRadio*** evaluada en diferentes escenarios. a) Dos radios conectadas. b) Una radio conectada. c) Ninguna radio conectada.

Si se implementa la sección de código directamente en la interfaz gráfica, esta no permite visualizar las variables de salida de cada etapa. Para poder seguir paso a paso la evolución del algoritmo, cada una de las etapas será implementada en scripts de forma independiente. Una vez que se verifique el correcto funcionamiento de los algoritmos, éstos serán incorporados en la programación de la interfaz gráfica correspondiente. Esta política será utilizada en cada una de las etapas que constituye el sistema de comunicación.

Para mostrar un ejemplo de configuración se crea un script en Matlab llamado *config_SDR.m*. Este script utiliza la función *findPlutoRadio* para determinar si algún dispositivo SDR Adalm Pluto está conectado o no. La longitud de la estructura que devuelve *findPlutoRadio* es cero cuando hay ningún dispositivo conectado y mayor a cero cuando encuentra al menos un dispositivo conectado. Si no se encuentra ningún dispositivo se mostrará un texto que informe lo propio. Cuando se encuentre un dispositivo conectado se configura el objeto transmisión y el objeto de recepción. También se muestra las características configuradas en el objeto de transmisión y recepción. Esto último se logra colocando el nombre de los objetos en una línea de código.

Además, se utiliza la función *info* para obtener la información del dispositivo SDR Adalm Pluto conectado. Los valores reales que se obtienen con la función *info()* pueden variar ligeramente de los valores especificados en los objetos de transmisión o recepción. El código del script *config_SDR.m* se presenta a continuación debidamente comentado.

```
% Script para mostrar un ejemplo de configuración del dispositivo SDR
% adalm Pluto
encontrarSDR=findPlutoRadio; %Busca la identificación de la radio
% y el número de serie
numeroEncontrado=length(encontrarSDR); % toma el valor de 0 si no
% se encuentra ningun dispositivo conectado
if(numeroEncontrado==0) %No está conectado ningún dispositivo
    disp('No se encontró ningun dispositivo SDR conectado.')
else
    % Se ha detectado almenos un dispositivo SDR conectado
    frecuenciaCentral=860e6; % Frecuencia central
    frecuenciaMuestreo=2.0e6; % Frecuencia de muestreo en banda base
    % Configuración del objeto para la tx:
    txAdalm=sdrtx('Pluto',...
        'RadioID','usb:0',...
        'CenterFrequency',frecuenciaCentral,...
        'Gain',-2,...
        'BasebandSampleRate',frecuenciaMuestreo);
    disp('Objeto para la transmisión configurado:')
    txAdalm % Para comprobar las propiedades configuradas
    % Configuración del objeto para la rx:
    rxAdalm = sdrrx('Pluto',...
        'RadioID','usb:0',...
        'CenterFrequency',frecuenciaCentral,...
        'GainSource','Manual',...
        'Gain',20,...
        'BasebandSampleRate',frecuenciaMuestreo,...
        'OutputDataType','single',...
        'SamplesPerFrame',800000,...
        'ShowAdvancedProperties',1,...
        'FrequencyCorrection',0);
    disp('Objeto para la recepción configurado:')
    rxAdalm % Para comprobar las propiedades configuradas
    disp('Información de los objetos:')
    disp('Objeto para transmisión')
    info(txAdalm) %Muestra algunas propiedades de txAdalm
    disp('Objeto para recepción')
    info(rxAdalm) %Muestra algunas propiedades de rxAdalm
end
```

Cuando el código se ejecuta y no está ningún equipo SDR Adalm Pluto conectado se obtiene como resultado el mensaje informativo siguiente:

```
>> config_SDR
No se encontró ningun dispositivo SDR conectado.
```

Cuando se ha conectado al menos una radio Adalm Pluto, primero se configura el objeto de transmisión llamado *txAdalm* y se muestra en la ventana de comandos los valores de las propiedades que se han configurado dicho objeto. Luego se configura el objeto de recepción llamado *rxAdalm* y se muestra en la ventana de comandos los valores de las propiedades configuradas en dicho objeto. En la Figura 2.16 se muestra en la ventana de comandos los valores configurados del objeto de transmisión y recepción.

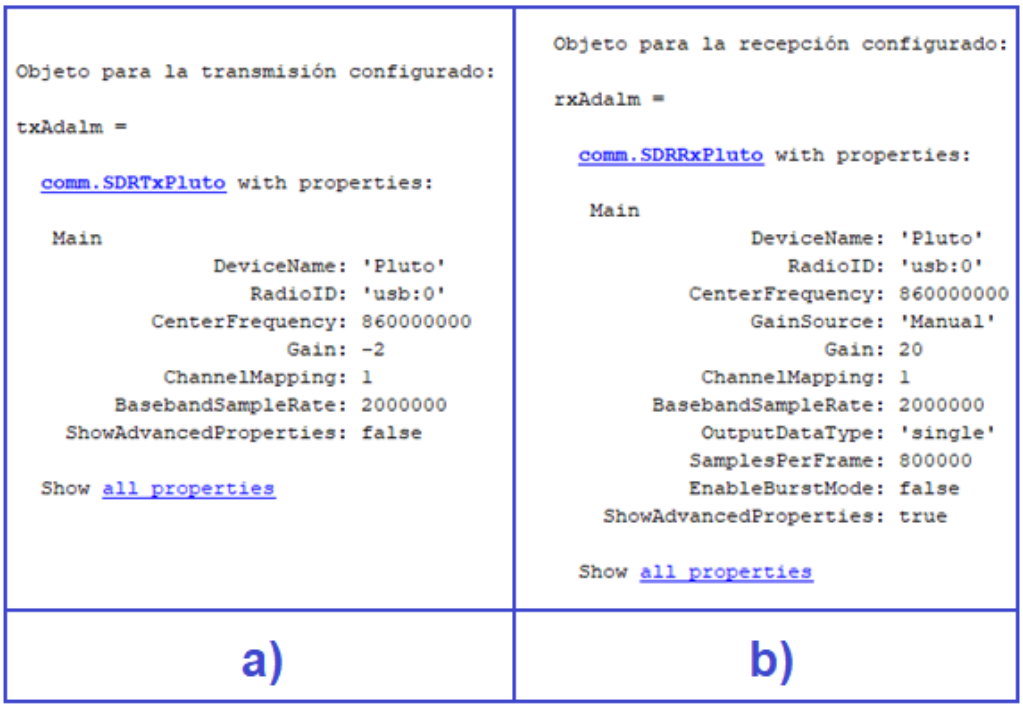


Figura 2.16. a) Valores configurados en el objeto de transmisión. b) Valores configurados en el objeto de recepción.

Mediante la función *info()*, Matlab puede conectarse al equipo Adalm Pluto y consultar los valores reales de frecuencia central, ganancia y frecuencia de muestreo, configurados en el objeto de transmisión o en el de recepción. Además, se consulta el número de serie del equipo, esto es útil cuando se tiene dos equipos conectados al computador. El resultado de utilizar *info(txAdalm)* se observa en la Figura 2.17 y el resultado de *info(rxAdalm)* se muestra en la Figura 2.18.

```
Información de los objetos:
Objeto para transmisión
## Establishing connection to hardware. This process can take several seconds.

ans =

  struct with fields:

                  Status: 'Full information'
        CenterFrequency: 859999998
      BasebandSampleRate: 2000000
               SerialNum: '104473dc59930019f3ff1600d8cd298201'
                    Gain: -2
    RadioFirmwareVersion: "0.31"
  ExpectedFirmwareVersion: "0.31"
        HardwareVersion: "B0"
```

Figura 2.17. Resultado de utilizar *info(txAdalm)*.

```
Objeto para recepción
## Establishing connection to hardware. This process can take several seconds.

ans =

  struct with fields:

                    Status: 'Full information'
            CenterFrequency: 859999998
          BasebandSampleRate: 2000000
                   SerialNum: '104473dc59930019f3ff1600d8cd298201'
                        Gain: 20
       RadioFirmwareVersion: "0.31"
    ExpectedFirmwareVersion: "0.31"
             HardwareVersion: "B0"
    EnableQuadratureTracking: 1
          EnableRFDCTracking: 1
    EnableBasebandDCTracking: 1
         FrequencyCorrection: 0
```

Figura 2.18. Resultado de utilizar *info(rxAdalm)*.

2.3. ETAPAS DE TRANSMISIÓN

En esta sección se aborda en detalle cada una de las etapas necesarias para procesar y transmitir un archivo de texto a través de un dispositivo SDR Adalm Pluto. Las etapas de transmisión se muestran en un diagrama de bloques en la Figura 2.19.

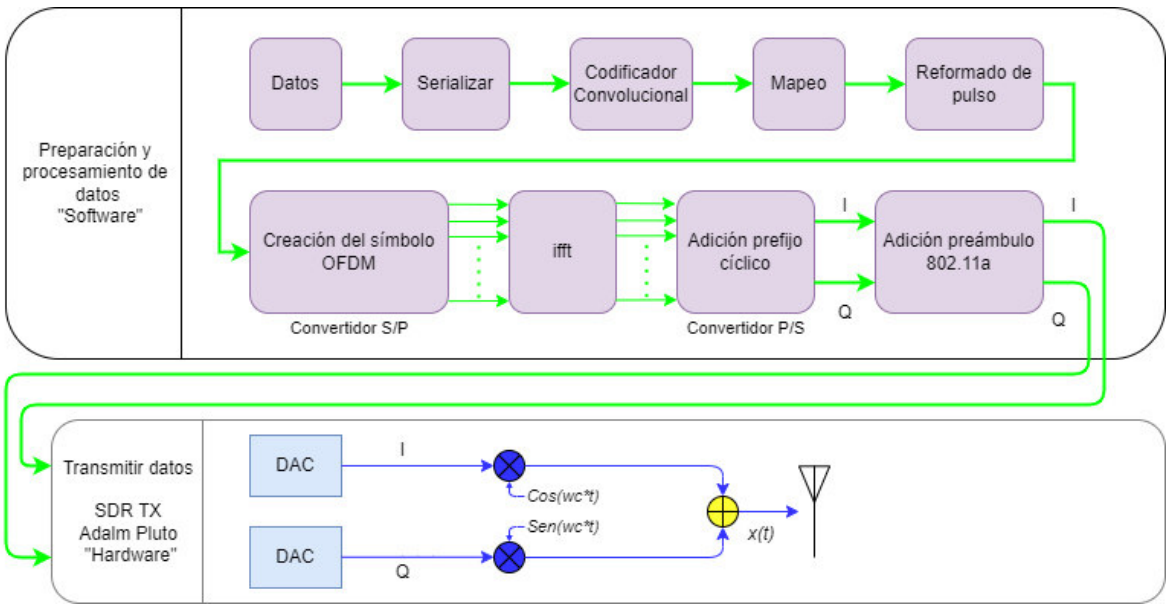


Figura 2.19. Etapas en el transmisor.

A continuación, se enumeran y describen brevemente las etapas de transmisión:

1) Datos. – Selecciona y abre el archivo de texto a transmitir.
2) Serializar. – Se encarga de representar cada carácter en binario y luego serializar los datos.

3) Codificador Convolucional: Etapa que se utiliza para la corrección de errores a nivel de bit. En la interfaz gráfica de transmisión esta etapa es opcional, es decir que se puede activar o desactivar según el usuario lo requiera.
4) Mapeo. – Etapa representa los bits serializados a un esquema de modulación digital que puede ser BPSK, QPSK, 16-QAM y 64-QAM.
5) Reformado de pulso. – Los datos mapeados pasan por un filtro raíz cuadrada de coseno elevado (SRRC). Esta etapa es opcional en la interfaz gráfica de transmisión.
6) Creación del símbolo OFDM. – Se construye cada símbolo OFDM con portadoras de datos, piloto, DC y de guarda.
7) IFFT (Transformada Inversa de Fourier). – Se encarga de pasar los símbolos OFDM del dominio de la frecuencia al dominio del tiempo.
8) Adición del prefijo cíclico. – En esta etapa se agrega las últimas muestras de cada símbolo OFDM al inicio del mismo.
9) Adición preámbulo 802.11a. – Se añade el preámbulo dado por el estándar IEEE 802.11a. Este preámbulo se compone de secuencias cortas y de secuencias largas.
10) Transmitir los datos. – Se transmiten los datos al canal inalámbrico utilizando el equipo SDR. Para transmitir los datos se utiliza el objeto global de transmisión configurado en la sección anterior. En la interfaz gráfica de transmisión los datos se envían recursivamente cada cierto intervalo de tiempo.

En las subsecciones posteriores cada etapa se detalla paso a paso, abordando la idea principal de la misma, el por qué se implementa, la teoría detrás, la implementación en Matlab y resultados parciales obtenidos.

2.3.1. ETAPA DE LECTURA DE DATOS

La etapa Datos se identifica en la Figura 2.20 con color anaranjado dentro de la parte de preparación y procesamiento del archivo de texto en transmisión. En esta etapa se busca que el usuario escoja un archivo de texto de extensión .txt para posteriormente procesarlo y transmitirlo. La manera más simple de que un usuario escoja un archivo, es mediante un cuadro de diálogo que enumere solamente los archivos de texto que contiene la carpeta actual.

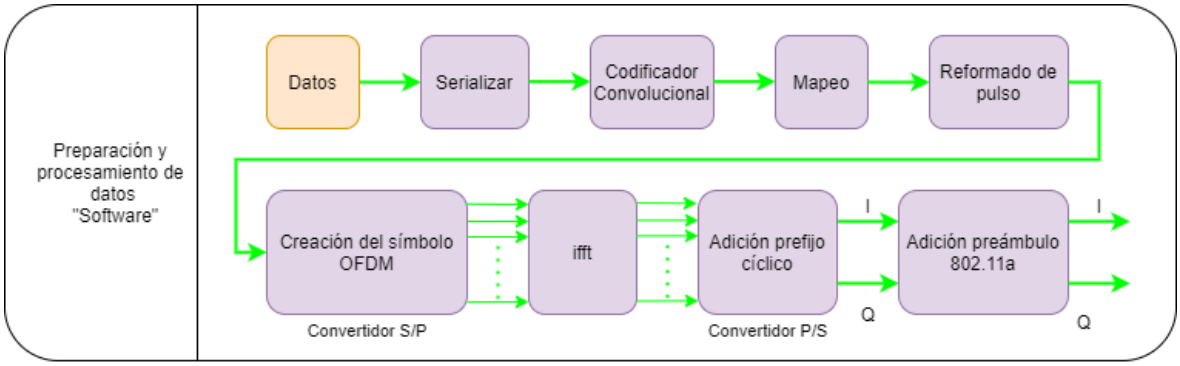


Figura 2.20. Etapas de transmisión con la etapa Datos identificada.

2.3.1.1. Fundamentos teóricos

En el presente proyecto se trabaja solo con archivos de texto de extensión .txt para procesarlos y transmitirlos. Esto debido a que el usuario puede visualizar y leer el archivo con el que se probará el prototipo de comunicación inalámbrica antes de enviarlo. En la recepción, además del BER, el usuario puede observar el archivo original y el archivo recuperado del canal inalámbrico, para así tener un referente visual del efecto del canal y de los diferentes algoritmos utilizados.

Para que el usuario interactúe adecuadamente con el prototipo, se selecciona el archivo de texto a través de un cuadro de diálogo. Los archivos a mostrar en el cuadro de diálogo deben ser únicamente de extensión .txt, por lo que se filtran estos archivos para evitar que el usuario seleccione un tipo de archivo diferente al archivo de texto.

2.3.1.2. Implementación en Matlab

En Matlab se puede seleccionar un archivo a través de un cuadro de diálogo utilizando la función *uigetfile*. Además, para abrir el archivo seleccionado se utiliza la función *fopen*, para leer el archivo se utiliza la función *fread* y para cerrar el archivo se utiliza la función *fclose*.

La función *uigetfile* tiene la siguiente sintaxis:

```
nombreArchivo = uigetfile(filter,title)
```

La variable *nombreArchivo* guarda un vector de caracteres con el nombre del archivo seleccionado. Los argumentos de entrada de la función *uigetfile* se explican a continuación.

- **filter**: Filtro de archivo, se especifica como un vector de caracteres o una matriz de vectores de caracteres. Este argumento especifica la o las extensiones de archivos que se muestran en el cuadro de diálogo. Por ejemplo *'*.txt'* para mostrar solamente los archivos de texto.

- **title**: Título del cuadro de diálogo que se especifica como un vector de caracteres.

Para abrir un archivo por medio de su nombre se utiliza la función *fopen*, con la sintaxis:

fID = fopen(nombreArchivo).

El argumento de entrada *nombreArchivo* de la sintaxis anterior, es un vector de caracteres con el nombre del archivo que se pretende abrir y *fID es* la variable donde se almacena el identificador del resultado de la lectura. Cuando el archivo se lee correctamente *fopen* devuelve un identificador que es un número entero mayor o igual que 3. Matlab reserva el identificador 0 para entrada estándar, 1 para salida estándar (la pantalla) y 2 para el error estándar. Si *fopen* no puede abrir el archivo, entonces *fID* es -1.

Una vez abierto el archivo se puede leer los datos del mismo, con la función *fread*. Esta función utiliza la sintaxix:

x=fread(fID,precision)

La función *fread* permite leer los datos de un archivo binario abierto y guardarlo en el vector columna *x*.

- **fID:** Es el número entero de identificador de archivo binario abierto. Este argumento se obtiene con la función *fopen*.
- **precision:** Es la clase y tamaño en bits de los valores a leer, especificados como un vector de caracteres o un escalar. Como se va a leer caracteres se debe usar '*char'.

Finalmente, para cerrar un archivo abierto se utiliza la función *fclose*, la sintaxis a utilizar es:

fclose(fID)

El parámetro *fID* es el número entero de identificador de archivo. Este parámetro se obtiene de la función *fopen*.

2.3.1.3. Lectura de un archivo de texto

El código utilizado para seleccionar y abrir un archivo de texto se implementa en un script llamado *abrir_archivo.m*. El script utiliza la función *uigetfile* para seleccionar y filtrar los archivos de texto. Cuando no se escoja ningún archivo se muestra un mensaje en la ventana de comandos y cuando se seleccione un archivo se procede a abrir, leer y guardar

el mismo. El código del script *abrir_archivo.m* se presenta a continuación debidamente comentado.

```
% Script para mostrar un ejemplo de abrir un archivo .txt
% Cuadro de diálogo para abrir un archivo:
nombreArchivo=uigetfile('*.txt','Seleccionar archivo a transmitir');
% La variable nombreArchivo toma el valor de 0 cuando no se ha
% seleccionado ningun archivo
if (nombreArchivo == 0)
    % Archivo no encontrado
    disp('Archivo no encontrado')
else
    % Archivo seleccionado, se procede a abrir el mismo
    fID = fopen(nombreArchivo);
    x=fread(fID,'*char'); % Leer el archivo
    fclose(fID); % Cerrar el archivo
end
```

Cuando se ejecuta el script *abrir_archivo.m* primero se abre un cuadro de diálogo, en donde se muestran todos los archivos de texto de la carpeta actual. En la Figura 2.21 se muestra un cuadro de diálogo con todos los archivos de texto que se tiene en la carpeta actual. Se selecciona el archivo que se desea abrir y se da clic en *Abrir*.

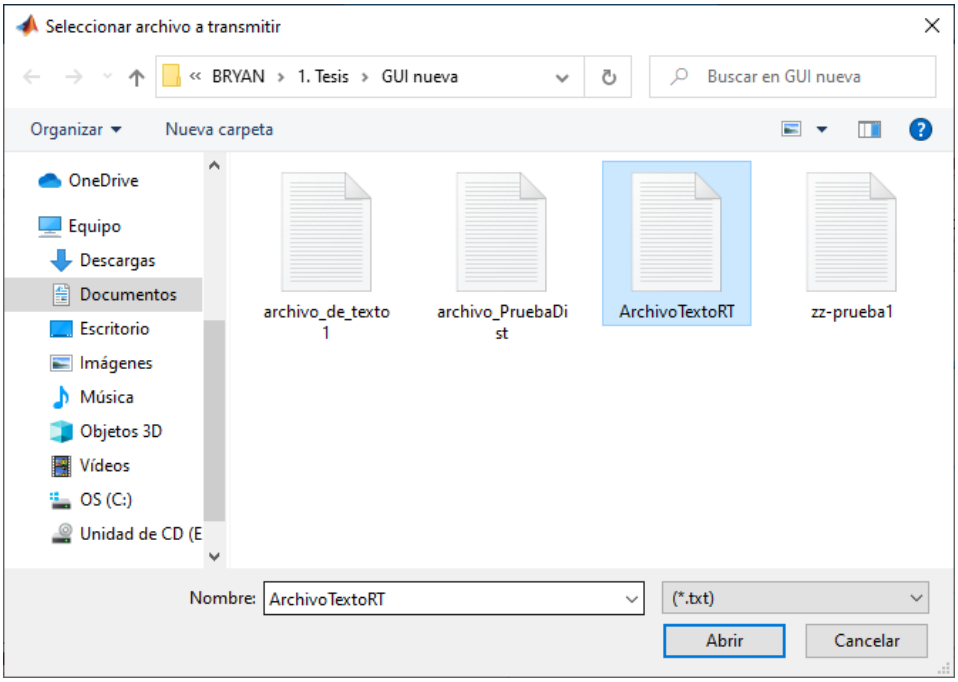


Figura 2.21. Cuadro de diálogo con un archivo de texto seleccionado.

El archivo de texto con el que se va a trabajar en etapas posteriores se llama *ArchivoTextoRT.txt* y contiene 1454 caracteres. Estos caracteres se muestran en el siguiente cuadro e incluyen caracteres mayúsculas, minúsculas, vocales tildadas, números, caracteres especiales, esto se realiza con el objetivo de que se puedan copiar y además para verificar si en la recepción no hay problema con algunos de éstos.

Este texto se va a transmitir y recebir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
Fragmento de texto de la novela Crónica de una muerte anunciada de Gabrial García Márquez.

"Prometió ocuparse de eso al instante, pero entró en el Club Social a confirmar una cita de dominó para esa noche, y cuando volvió a salir ya estaba consumado el crimen.Cristo Bedoya cometió entonces su único error mortal: pensó que Santiago Nasar había resuelto a última hora desayunar en nuestra casa antes de cambiarse de ropa, y allá se fue a buscarlo. Se apresuró por la orilla del río, preguntándole a todo el que encontraba si lo habían visto pasar, pero nadie le dio razón. No se alarmó, porque había otros caminos para nuestra casa. Próspera Arango, la cachaca, le suplicó que hiciera algo por su padre que estaba agonizando en el sardinel de su casa, inmune a la bendición fugaz del obispo. «Yo lo había visto al pasar -me dijo mi hermana Margot-, y ya tenía cara de muerto.» Cristo Bedoya demoró cuatro minutos en establecer el estado del enfermo, y prometió volver más tarde para un recurso de urgencia, pero perdió tres minutos más ayudando a Próspera Arango a llevarlo hasta el dormitorio. Cuando volvió a salir sintió gritos remotos y le pareció que estaban reventando cohetes por el rumbo de la plaza..."

Una vez seleccionado el archivo se lo lee y se almacena en una matriz tipo *char*, la cual se llama *x*. Esta matriz es una secuencia de caracteres, es decir que cada carácter es un elemento de la matriz. Además, *x* es de tamaño igual al número de caracteres, esta matriz puede verse como un vector columna de caracteres. La Figura 2.22 muestra las variables del espacio de trabajo, cuando se ha seleccionado un archivo y los primeros elementos de la variable x.

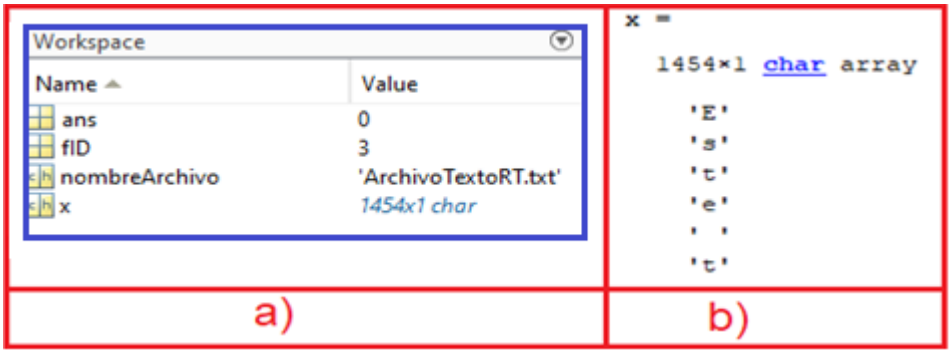


Figura 2.22. a) Variables en el espacio de trabajo. b) Variable *x*, que contiene los datos del archivo abierto.

En caso de que no se seleccione ningún archivo y se cierre el cuadro de diálogo se tendrá como resultado en el espacio de trabajo y en la ventana de comandos el mensaje de la Figura 2.23.

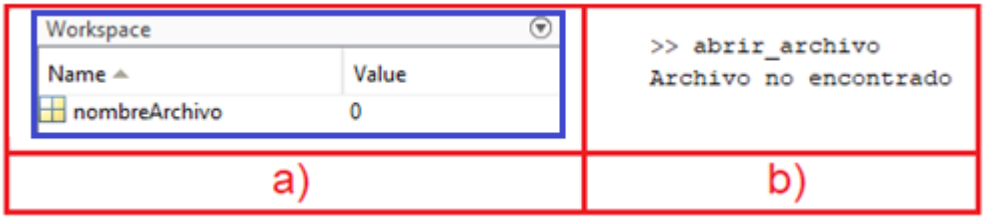


Figura 2.23. a) Variable en el espacio de trabajo. b) Mensaje en la ventana de comandos cuando no se selecciona ningún archivo.

2.3.2. ETAPA DE SERIALIZACIÓN

La etapa Serializar se identifica en la Figura 2.24 con color anaranjado dentro de la parte de preparación y procesamiento del archivo de texto en transmisión. En esta etapa se busca pasar los datos del vector de caracteres, obtenido en la etapa anterior, a una secuencia de bits serializados. Entonces cada carácter del archivo de texto se representa en binario para luego serializar los mismos.

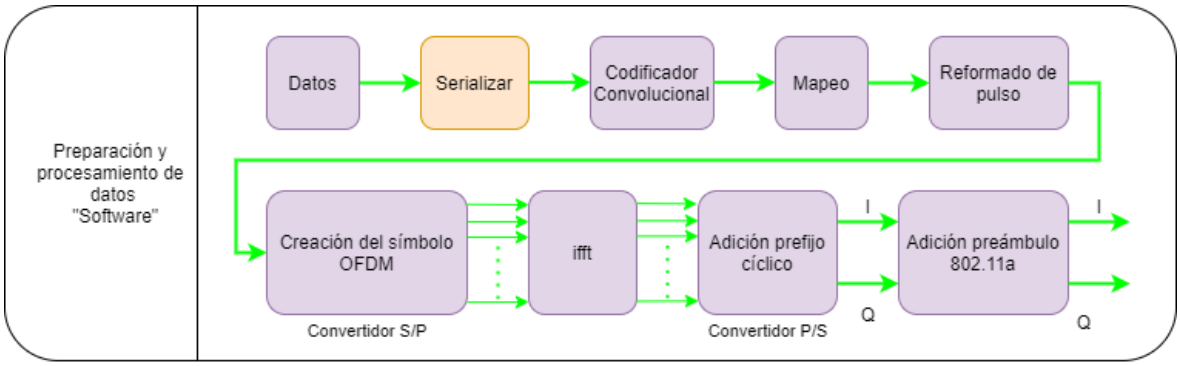


Figura 2.24. Etapas de transmisión con la etapa Serializar identificada

2.3.2.1. Fundamentos teóricos

En la etapa anterior se obtuvo un vector columna tipo *char*, en esta etapa primero se representa cada uno de los caracteres del vector en binario. El resultado de este proceso es una matriz tipo *char* en donde en cada fila se encuentra el número binario que corresponde a un carácter. Como siguiente paso se serializa la matriz obteniendo un vector columna del tipo *char*. Luego el vector obtenido se representa numéricamente para trabajar en etapas posteriores.

Matlab almacena cada uno de los caracteres como caracteres Unicode, en donde los primeros 128 símbolos corresponden a caracteres ASCII, es decir que ASCII y Unicode tienen los mismos códigos numéricos [13]. Esto aplica para matrices de caracteres y para matrices de cadenas (string). Cada carácter se representa con un número decimal, el cual a su vez puede representarse en binario. La Tabla 2.4 muestra los caracteres ASCII imprimibles, con su representación en decimal.

Tabla 2.4. Caracteres imprimibles del código ASCII [14].

Caracter	Código ASCII	Caracter	Código ASCII	Caracter	Código ASCII	Caracter	Código ASCII
espacio	32	8	56	P	80	h	104
!	33	9	57	Q	81	i	105
"	34	:	58	R	82	j	106
#	35	;	59	S	83	k	107
$	36	<	60	T	84	l	108
%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
'	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	_	95	w	119
0	48	H	72	`	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103		

2.3.2.2. Implementación en Matlab

Matlab dispone de la función *dec2bin* que convierte un entero decimal en su representación binaria. Debido a que cada caracter tiene asociado un número entero decimal, se puede utilizar la función *dec2bin* para convertir directamente un carácter en bits. La sintaxis utilizada es la siguiente:

```
textoBinario = dec2bin(Datos,nD)
```

La función *dec2bin* devuelve una representación binaria con al menos *nD* dígitos. Los argumentos de entrada se describen a continuación.

- **Datos:** Corresponde al vector columna de caracteres a representar en binario.

- **nD:** Es el número mínimo de bits con el que se representa cada carácter y se especifica como un número entero.

En *textoBinario* se almacena una matriz tipo *char* que contiene un número de filas igual al número de caracteres y un número de columnas igual a *nD.* Luego se serializan los datos, utilizando lazos *for*. Después los bits serializados dentro del vector tipo *char*, se convierte al tipo *double* (punto flotante de doble precisión). Esto debido a que Matlab almacena los valores numéricos, de forma predeterminada en tipo *double* [15]. En consecuencia, resulta una mejor alternativa trabajar con datos del tipo *double*.

Matlab dispone de la función *str2double*, la cual se utiliza para convertir ya sea un vector de caracteres, una matriz de celdas de caracteres o una matriz de cadenas de caracteres en valores tipo *double*. La sintaxis que se utiliza es:

Xdouble = str2double(str)

El argumento de entrada es *str* y en éste caso es un vector de caracteres.

2.3.2.3. Serialización del archivo de texto

En la etapa anterior, llamada *Datos*, se seleccionó, abrió y guardó los caracteres de un archivo se texto mediante un script llamado *abrir_archivo.m*. La variable de salida en dicho script se llama *x*, la cual se utiliza como datos de entrada para la etapa actual (*Serializar*). El código utilizado para implementar un ejemplo de la etapa de serialización se implementa en un script llamado *serializar_datos.m*.

Cabe mencionar que las variables obtenidas en la etapa *Datos* a excepción de *x*, deben ser borradas con el objetivo de no confundir al lector y que se mantengan solo las variables de la etapa actual. Las variables pueden ser borradas del espacio de trabajo mediante el comando:

clearvars -except nombreDeVariable

El argumento *nombreDeVariable* es el nombre de la variable que no se quiere borrar. Puede colocarse varios nombres de variables separadas por un espacio. El código que se utiliza para borrar todas las variables, excepto la variable de entrada del script *serializar_datos.m* se muestra a continuación.

```
% Borrar variables en serializar:
clearvars -except x % Borrar todas las variables excepto x
```

El código del script *serializar_datos.m* inicia por representar cada carácter del vector *x* con 8 bits, con el fin de obtener un valor fijo del número de bits, la variable en donde se guarda cada carácter en binario se llama *textoBinario*. Luego para serializar los bits almacenados en *textoBinario* se utilizan dos lazos *for* anidados: el primer lazo *for* recorre cada fila y el segundo recorre los elementos (8 columnas) de cada fila almacenando cada uno de ellos en un nuevo vector. Mediante una variable auxiliar, inicializada antes de ingresar a los lazos, se almacenan los datos en la posición que indique esta variable. La variable que guarda los datos serializados se llama *textoBitsSerial*, esta variable sigue siendo del tipo *char*.

Para convertir los datos tipo *char* a datos tipo *double*, se utiliza un lazo *for* para recorrer cada bit del vector *textoBitsSerial*. Dentro del lazo se utiliza la función *str2double* en cada uno de los bits, obteniendo como resultado los bits serializados dentro de un vector tipo *double*. Finalmente, se almacenan los datos serializados en una variable llamada *datosSerializados*. El código descrito en estos dos párrafos se presenta debidamente comentado, a continuación.

```
% Script para mostrar un ejemplo de la etapa Serializar
% Como arcumento de entrada se debe tener un vector de caracteres
% tipo columna, en este caso llamado x
textoBinario = dec2bin(x,8); % representacion de cada caracter
% con 8 bits (codificación)
% Para serializar los datos se utiliza dos lazos for(). El primer
% lazo for() recorre cada fila, length(textoBinario) representa el
% número de filas. El segundo lazo for() en cada fila recorre todos
% los elementos (8 columnas). Y con una variable auxiliar (aux) se
% almacena los datos en la posición que dicte la variable aux.
aux=1; % Inicializar variable auxiliar
for i = 1:length(textoBinario)
    for j=1:8
        textoBitsSerial(aux)=textoBinario(i,j); % tipo char
        aux=aux+1; % Sumar 1 en la variable auxiliar
    end
end
% Para convertir los datos del tipo char al tipo str se utiliza un
% lazo for para recorrer todos los elementos del vector serializado
for aux=1:length(textoBitsSerial)
    % Se utiliza una variable auxiliar para almacenar los datos
    datosS_aux(aux)=str2double(textoBitsSerial(aux));
end
% Se copia los datos en una variable
datosSerializados=datosS_aux; %datos serializados
```

Una vez ejecutado el script de la etapa *Datos*, borradas las variables que no se utilizan de dicha etapa y ejecutado el script de esta etapa que contiene el código anterior se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.25.

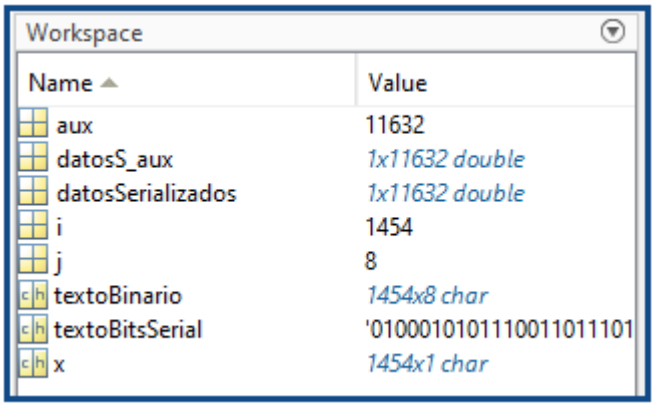


Figura 2.25. Variables en el espacio de trabajo luego de ejecutar *serializar_datos.m*

En la Figura 2.26 se compara los primeros datos del vector *x* (entrada de la etapa) con los primeros datos de la matriz codificada de nombre *textoBinario*. Los valores se ubican de tal forma que a la izquierda se tiene el carácter y a la izquierda se tiene su representación binaria. Por ejemplo, para el primer carácter 'E', según la Tabla 2.4 el código ASCII es 69 y este número en binario de 8 bits es igual a 01000101.

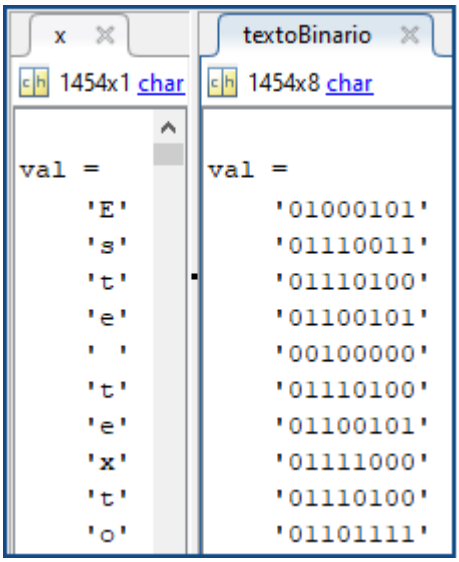


Figura 2.26. Caracteres de entrada y caracteres codificados.

Luego de codificar los caracteres, se serializa la matriz *textoBinario* mediante el uso se lazos *for* y se almacena en la variable de nombre *textoBitsSerializado*. En la Figura 2.27 se comparan los primeros elementos de estas dos variables y se identifica con colores cada carácter codificado dentro de las mismas.

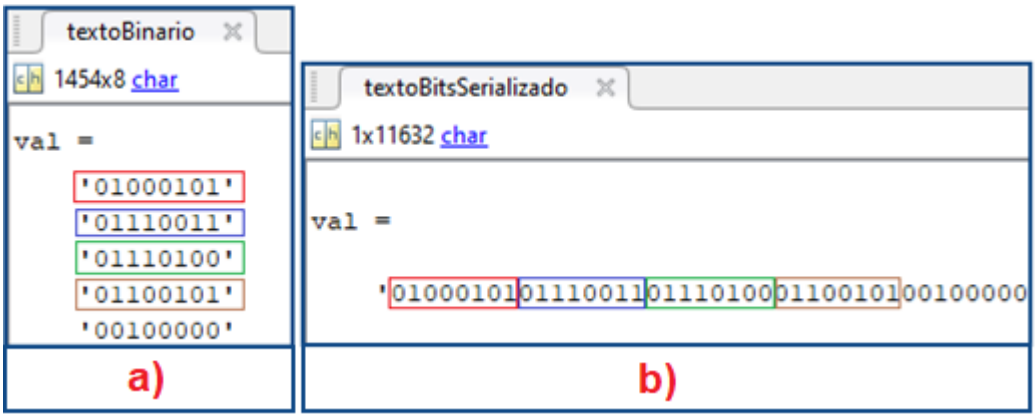


Figura 2.27. a) Datos binarios en paralelo. b) Datos binarios serializados.

Luego de utilizar un lazo *for* y la función *str2double* se obtiene los bits serializados que se almacenan en la variable *datosS_aux* cuyos datos son copiados en la variable *datosSerializados*. En la Figura 2.28 se muestra los primeros 12 elementos del vector *datosSerializados* y los elementos se señalan con los mismos colores correspondientes a los bits que se mostraron en la Figura 2.27.

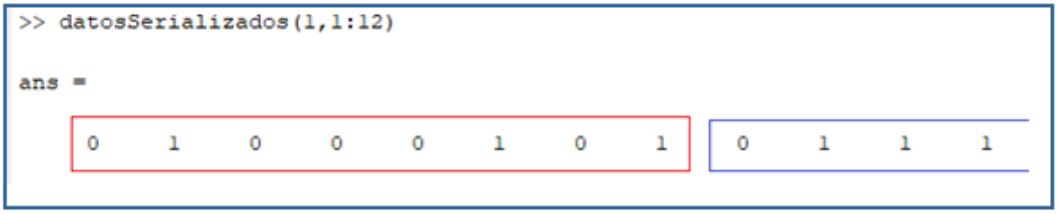


Figura 2.28. Primeros 12 elementos de *datosSerializados*.

2.3.3. ETAPA DE CODIFICACIÓN CONVOLUCIONAL

La etapa Codificador Convolucional se identifica en la Figura 2.29 con color naranja dentro de la parte de preparación y procesamiento del archivo de texto en transmisión. En esta etapa se busca implementar un código corrector de errores a nivel bit mediante un codificador convolucional. La naturaleza aleatoria del canal inalámbrico introduce errores en el proceso de transmisión, por lo que, al utilizar corrección de errores, se puede reducir el BER significativamente.

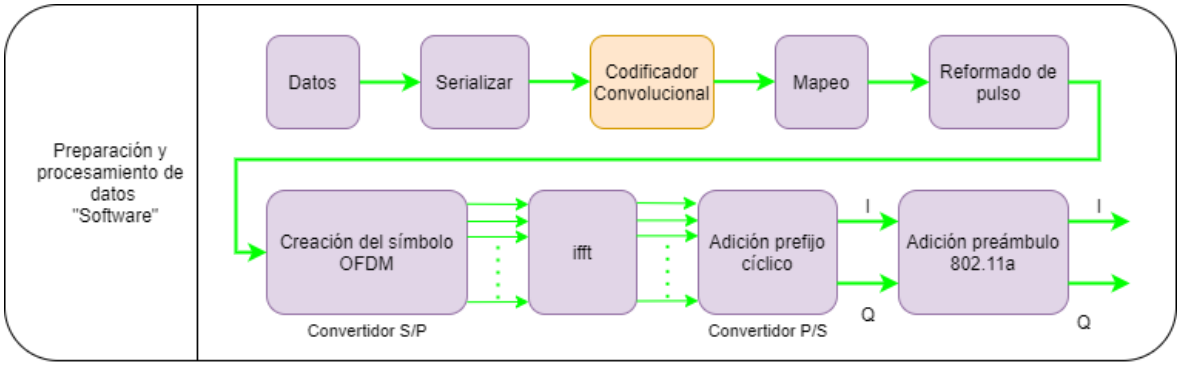


Figura 2.29. Etapas de transmisión con la etapa Codificador Convolucional identificada.

2.3.3.1. Fundamentos teóricos

La cantidad de errores en un sistema de comunicación inalámbrico está relacionada con la relación señal al ruido (SNR – Signal to Noise Ratio), si se eleva esta relación la cantidad de errores disminuirá [6]. Cuando no se quiere utilizar técnicas de corrección de errores, para reducir la cantidad de errores del sistema se puede elevar la SNR. Para elevar la SNR se aumenta la potencia del transmisor o se disminuye la velocidad de transmisión. El utilizar técnicas de codificación para la corrección de errores permite, emplear una potencia de transmisión más baja o bien aumentar la velocidad de transmisión, manteniendo la tasa de error.

Si se mantiene la misma potencia y velocidad de transmisión, la cantidad de errores disminuye significativamente cuando se utilizan técnicas de corrección de errores.

En el prototipo de comunicación inalámbrica que se desarrolla en este trabajo, se evalúa el efecto que tiene la corrección de errores mediante la tasa de bits errados (BER). Para este fin, se ha optado por utilizar códigos convolucionales del tipo FEC (Forward Error Correction). Las técnicas FEC generan datos adicionales para ser enviados junto con la información, estas técnicas no solicitan retransmisiones al emisor. Los códigos convolucionales son un método FEC que trabaja a nivel de bit y no a nivel de bloques.

La codificación convolucional es un método en donde los bits que se tendrán a la salida del sistema se determinan mediante operaciones lógicas en el bit actual y en un grupo de bits anteriores [6]. Es decir que el codificador convolucional es un sistema con memoria, debido a que la salida depende de los bits previos. Los códigos convolucionales son códigos no sistemáticos, es decir que la información codificada no se puede ver de manera explícita en la palabra codificada [16].

El funcionamiento de este tipo de codificación puede representarse a través de registros, diagrama de estados, diagrama de árbol, diagrama de trellis y convolución siendo esta última implementada intrínsecamente en los métodos anteriores [17]. El diagrama de trellis

es el método más ampliamente utilizado para la descripción de códigos convolucionales y con el cual se va a implementar el codificador convolucional en Matlab. Sin embargo, es necesario abordar algunos otros métodos para entender a la codificación convolucional. Los métodos a abordar antes de llegar al diagrama de trellis son los registros de desplazamiento y el diagrama de estados.

2.3.3.1.1. Registro de desplazamiento

La implementación de un codificador convolucional mediante registros de desplazamiento es una forma simple y detallada de explicar el funcionamiento del codificador mencionado. Los parámetros principales del codificador convolucional son:

- ***k*:** los bits de entrada
- ***n*:** la longitud de la palabra código por cada ingreso de k bits
- ***m*:** el tamaño de registro de la memoria

Otro parámetro es la tasa de codificación del codificador convolucional $R_c = (k/n)$, es decir la relación entre los bits de entrada y los de salida. Cuanto más grande es esta tasa, mayor es la protección contra errores. Sin embargo, si la tasa del codificador aumenta los retardos por procesamiento y el número de bits a transmitir también crecen. Lo ideal es encontrar un equilibrio entre la protección contra errores y los retardos por procesamiento.

En el codificador convolucional los bits de datos se introducen en un registro de desplazamiento de longitud ***m*** y que contiene ***k*** sumadores binarios. Los codificadores convolucionales suelen representarse con la sintaxis (***n***, ***k***, ***m***), para indicar el valor de sus parámetros. En la Figura 2.30 se presenta un codificador convolucional con tres registros de desplazamiento, dos sumadores binarios y un bit de entrada de datos, es decir un codificador convolucional (2, 1, 3). Los registros x[n], x[n-1] y x[n-2] representa el estado actual, el primer estado anterior y el segundo estado anterior, respectivamente. Es decir que los tiempos de muestreo discretos se etiquetan como n, n-1 y n-2.

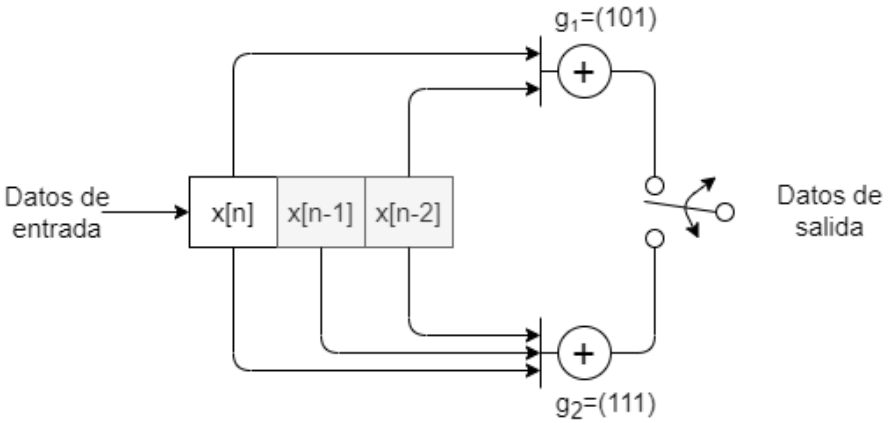


Figura 2.30. Codificador convolucional (2, 1, 3)

Los polinomios generadores son otro concepto importante sobre la codificación convolucional, por cada sumador binario, debe existir un polinomio generador. Estos polinomios se refieren a las conexiones que se tienen entre los registros de desplazamiento y las salidas del codificador. Por lo general se suele utilizar un 1 para indicar que existe una conexión y un 0 para indicar que no existe una conexión. En la Figura 2.30 las conexiones entre cada registro y el sumador binario se representa mediante flechas y los polinomios generadores son los vectores binarios tipo fila $g_1 = (101)$ y $g_2 = (111)$. Estos polinomios también pueden representarse de forma octal que corresponderían a $g_1 = 5_{oct}$ y $g_2 = 7_{oct}$.

En este trabajo se utiliza polinomios generadores $g_1 = 5_{oct}$ y $g_2 = 7_{oct}$, pero hay que considerar que existen otras opciones, por ejemplo, en el estándar 802.11 se definen polinomios generadores $g_1 = 1338_{oct}$ y $g_2 = 1778_{oct}$ [18].

Antes de ingresar los bits a codificar al registro de desplazamiento para iniciar con el proceso de codificación es necesario establecer la manera en la que se va a inicializar los registros de memoria del codificador convolucional. También es importante establecer la forma de terminar el proceso de codificación. A continuación, se describen tres formas en las que se pueden iniciar y terminar los registros de codificación [17].

- **Método por relleno de ceros:** Este método asegura que el estado inicial y el final de la codificación sea cero. En el estado inicial se configura todos los registros de desplazamiento en 0. Para asegurar que los últimos estados sean 0, se rellena los bits a codificar con ***m*** bits en 0.
- **Método por truncado:** En el estado inicial se configura todos los registros de desplazamiento en 0, igual que el estado anterior. Para finalizar la codificación no se realiza ningún relleno y por ende el estado final es desconocido y esto puede incrementar los errores en los últimos bits.

- **Método de inicio por bits de cola:** En este método se copian los últimos ***m*** bits del total de bits a codificar, asegurando así que el estado final y el inicial sean los mismos.

Para describir el funcionamiento del codificador convolucional utilizando registros de desplazamiento en los párrafos siguientes se desarrolla un **ejemplo** de codificación.

En el **ejemplo** se tiene como datos de entrada los bits 1101, un codificador convolucional (2, 1, 3), polinomios generadores son $g_1 = (101)$ y $g_2 = (111)$. Además, el método de inicio y terminación utilizado para desarrollar el ejemplo es el método por truncado. En la Figura 2.31 se muestra el codificador inicializado con ceros en todos los registros de desplazamiento.

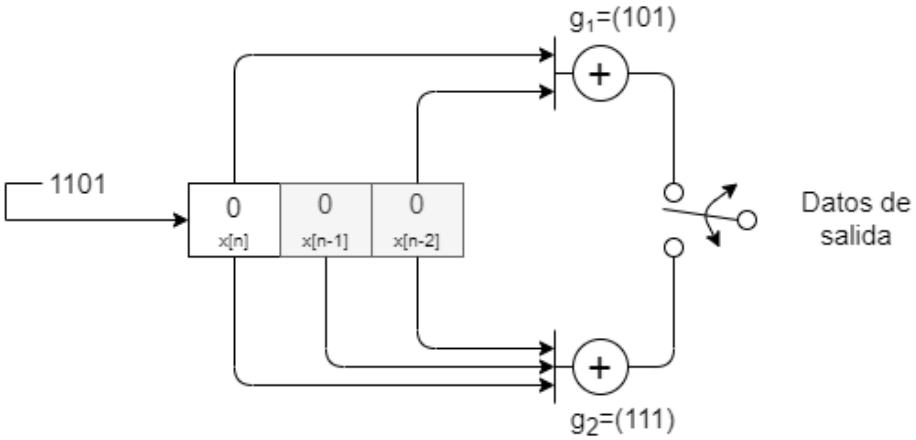


Figura 2.31. Codificador convolucional (2, 1, 3), inicializado con ceros en los registros de desplazamiento.

A medida que cada bit ingresa a la izquierda del registro de desplazamiento, los bits anteriores se desplazan hacia la derecha y el bit más antiguo se elimina del registro. En la Figura 2.32 se muestra el codificador con el primer bit ingresado en el primer registro y los dígitos anteriores desplazados. También se muestra el resultado de cada sumador binario y además los datos de salida codificados. En la salida de los sumadores binarios dan como resultado un uno lógico, si es que las conexiones a unos lógicos es impar, caso contrario el resultado es un cero lógico [6].

Debido a que en los dos sumadores de la Figura 2.32 se ingresa un número impar de unos lógicos, el resultado a la salida de cada sumador es un uno lógico. Entonces, los datos codificados a la salida del codificador, cuando se ingresa el primer bit, corresponden a la secuencia 11.

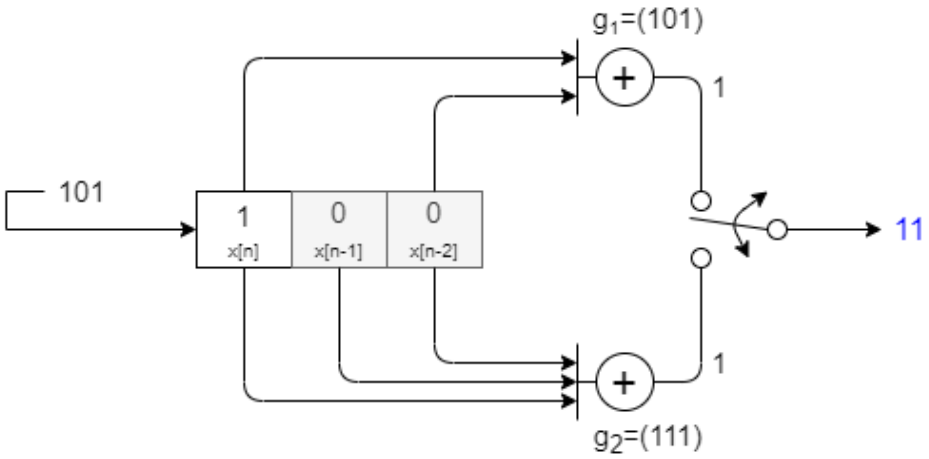


Figura 2.32. Codificador convolucional con el primer bit ingresado.

En la Figura 2.33 se ingresa el segundo bit, el resultado del sumatorio superior es uno lógico y el resultado del sumatorio inferior es cero lógico, debido a que se tiene un número par de unos lógicos. Entonces los datos codificados a la salida corresponden a la secuencia 10.

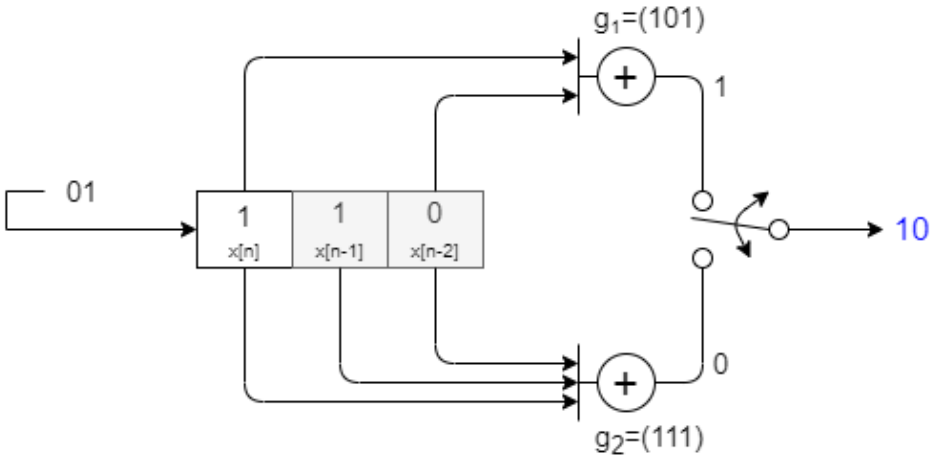


Figura 2.33. Codificador convolucional con el segundo bit ingresado.

En la Figura 2.34 se ingresa el tercer bit, el resultado del sumatorio superior es uno lógico y el resultado del sumatorio inferior es cero lógico, debido a que se tiene un número par de unos lógicos. Entonces los datos codificados a la salida corresponden a la secuencia 10.

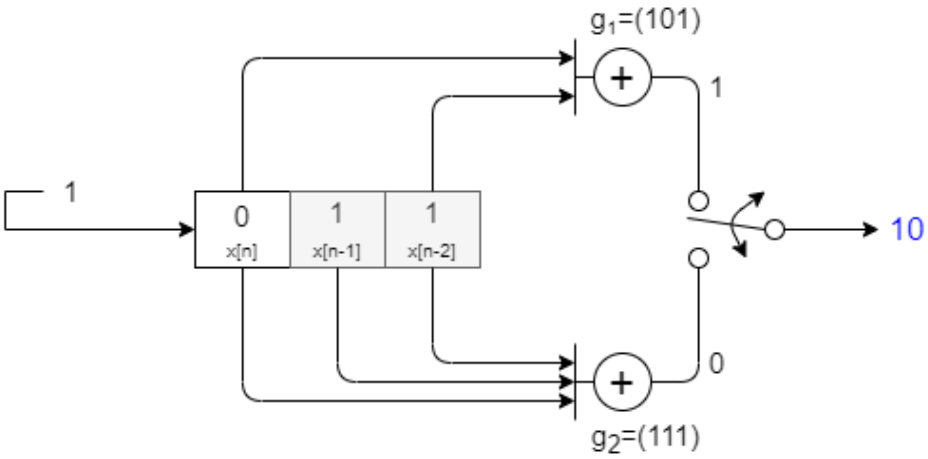


Figura 2.34. Codificador convolucional con el tercer bit ingresado.

En la Figura 2.35 se ingresa el cuarto bit, el resultado del sumatorio superior es cero lógico y el resultado del sumatorio inferior es cero lógico, debido a que se tiene un número par de unos lógicos en ambos casos. Entonces los datos codificados a la salida corresponden a la secuencia 00.

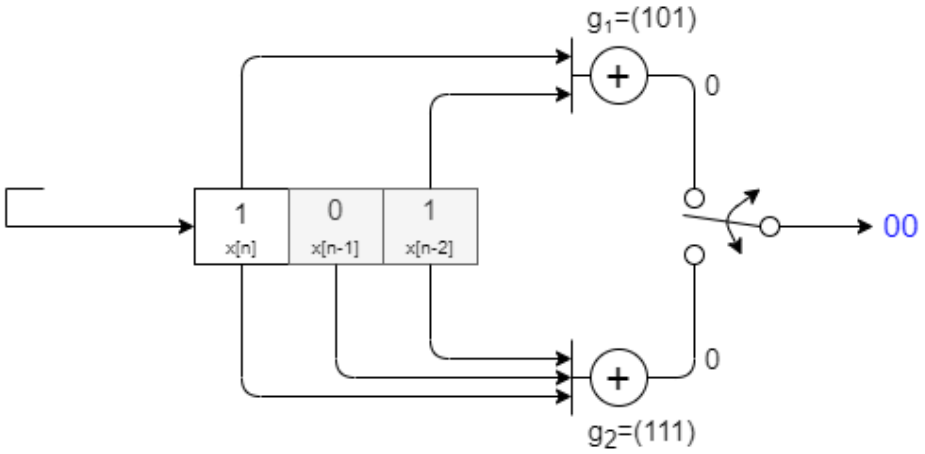


Figura 2.35. Codificador convolucional con el cuarto bit ingresado.

En consecuencia, cuando se utiliza un codificador convolucional (2, 1, 3), polinomios generadores $g_1 = (101)$ y $g_2 = (111)$, inicio de la codificación por ceros lógicos (método de truncado) y bits de entrada igual a la secuencia 1101, se obtiene como salida la secuencia codificada 11101000.

2.3.3.1.2. Diagrama de estados

El diagrama de estados permite tener una representación de todos los estados posibles para el codificador. Este diagrama se construye iniciando por ubicar todos los estados en orden, tal que al avanzar por cualquiera, los bits de los estados recorren hacia la derecha. Por ejemplo, si el estado inicial es 00 e ingresa un bit de entrada igual a 1 el estado siguiente es 10, si en este estado ingresa un 1 el estado siguiente será 11.

Para determinar la salida de bits codificados en el diagrama de estados hay que basarse en el codificador por registros de desplazamiento visto previamente. En la Tabla 2.5 se muestra un resumen de todos los estados posibles de los registros de desplazamiento del codificador convolucional. Además, se muestran los valores que toman a la salida de cada sumatorio que utiliza un polinomio generador y finalmente la salida codificada por cada bit de entrada (x[n]).

Tabla 2.5. Entradas y salidas de un codificador convolucional (2, 1, 3) por registros de desplazamiento y con polinomios generadores $g_1 = (101)$ y $g_2 = (111)$.

x[n]	x[n-1]	x[n-2]	Salida g1=(101)	Salida g2=(111)	Salida del codificador
0	0	0	0	0	**00**
0	0	1	1	1	**11**
0	1	0	0	1	**01**
0	1	1	1	0	**10**
1	0	0	1	1	**11**
1	0	1	0	0	**00**
1	1	0	1	0	**10**
1	1	1	1	1	**11**

En la Figura 2.36 se ubican todos los estados y a la salida de cada estado se escribe el bit de entrada y los bits codificados separados por un ‘ / ’ (bit_entrada / bits_codificados). Además, cabe mencionar que el diagrama de estados corresponde a un codificador (2, 1, 3) con polinomios generadores $g_1 = (101)$ y $g_2 = (111)$. Si los polinomios generadores cambian, puede ocasionar que el diagrama de estados también cambie.

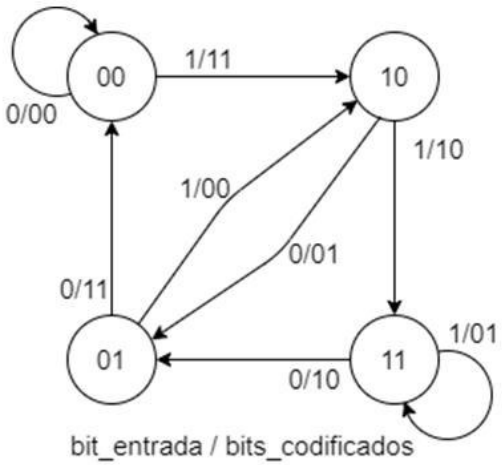


Figura 2.36. Codificador convolucional (2, 1, 3), por diagrama de estados.

Al igual que con el codificador por registros de desplazamiento, en este codificador convolucional por diagrama de estados se va a codificar la secuencia de bits 1101. Los pasos correspondientes se describen a continuación.

- El estado inicial es 00, debido a que se utiliza el método de truncado, el bit de ingreso es 1 y por tanto **los bits codificados son 11**. El estado siguiente es 10.
- Luego el estado actual resulta ser 10, se ingresa el segundo bit que corresponde a 1 y **los bits codificados son 10**. El estado siguiente es 11.
- Después se ingresa el tercer bit que es 0 y **los bits codificados son 10**. El estado siguiente es 01.
- Finalmente, se ingresa el cuarto bit correspondiente a 1 y **los bits codificados son 00.**

Entonces el resultado de codificar la secuencia 1101 es 11101000, que es el mismo valor que se obtuvo utilizando los registros de desplazamiento.

2.3.3.1.3. Diagrama de Trellis

El diagrama de trellis tiene forma de una red y permite observar la evolución de cada estado en la codificación. Los estados se ubican en columnas y se replican en forma horizontal hasta tener un número de estados igual al número de bits a codificar más uno. Como estado inicial se suele representar el tiempo n=0, donde n corresponde al valor del tiempo discreto. Cada columna representa el avance en el transcurso del tiempo discreto. Es decir que, si se tiene 4 bits a codificar, se deberá tener en total 5 columnas, iniciando en el tiempo discreto n=0 y terminando en n=1.

Para la construcción del diagrama de trellis de un codificador convolucional (2, 1, 3) y con polinomios generadores $g_1 = (101)$ y $g_2 = (111)$ se puede tomar como base el diagrama de estados de la sección anterior (Figura 2.36).

Entonces al igual que el diagrama de estados, el diagrama de trellis tiene 4 estados, los cuales se colocan en una columna y se los replica empezando en el tiempo discreto n=0 (estado inicial) hasta n igual al número de bits de entrada. Para representar el cambio de estado cuando ingresa un bit cero (0) se utiliza una línea continua de color azul y una línea punteada de color verde cuando ingresa un uno (1). El diagrama de trellis inicia en el estado 00 y si ingresa un 0 el estado siguiente es 00, por el contrario, si ingresa un 1 el estado siguiente es 10 y así siguiendo las secuencias descritas en el diagrama de estados. Los bits codificados se colocan sobre la línea que representa el ingreso del bit 0 o 1.

En la Figura 2.37 se muestra el diagrama de trellis, con valores de tiempo discreto, correspondiente a un codificador convolucional (2, 1, 3) y con polinomios generadores $g_1 = (101)$ y $g_2 = (111)$.

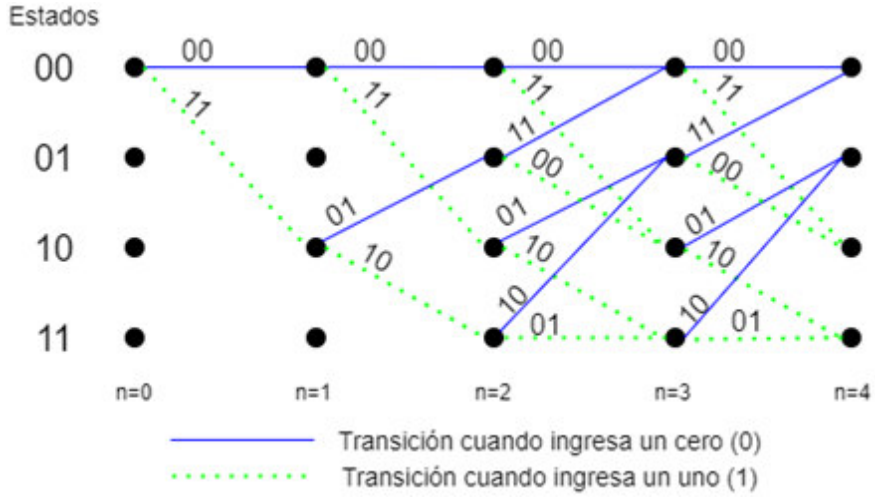


Figura 2.37. Diagrama de Trellis de un codificador convolucional (2, 1, 3).

Para describir y comprobar el funcionamiento del diagrama de trellis, se utiliza la secuencia de bits 1101, que es la misma con la que se ha venido trabajando en los anteriores métodos de codificación convolucional. En la Figura 2.38 se señala con rojo los bits codificados y a continuación se describe el avance y codificación para entender de mejor manera el proceso.

- El estado inicial es 00, el primer bit que ingresa es 1, la secuencia codificada es 11 y el estado siguiente es 10.
- Luego el estado actual pasa a ser 10, el bit que ingresa es 1, la secuencia codificada es 10 y el estado siguiente es 11.
- Después el estado actual pasa a ser 11, el bit que ingresa es 0, la secuencia codificada es 10 y el estado siguiente es 01.
- Finalmente, el estado actual pasa a ser 01, el bit que ingresa es 1 y la secuencia codificada es 00.

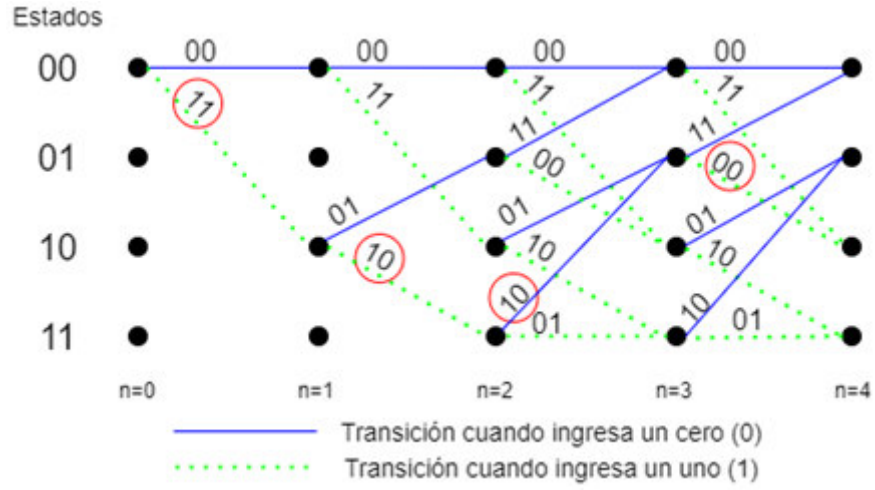


Figura 2.38. Diagrama de trellis de un codificador convolucional (2, 1, 3), con la secuencia codificada para los bits de entrada 1101.

Entonces los bits codificados para la entrada 1101 es 11101000, que es la misma secuencia obtenida con el registro de desplazamiento y con los diagramas de estados.

2.3.3.2. Implementación en Matlab

Como se mencionó anteriormente, el método con el que se va a implementar el codificador convolucional es mediante el diagrama de trellis. Matlab dispone de la función *poly2trellis* que convierte los polinomios de código convolucional en una descripción de trellis de tasa k/n [19]. Hay que recordar que en la tasa de codificación $R_c = (k/n)$, ***k*** es la cantidad de flujos de bits a la entrada del codificador y que ***n*** es la longitud de la palabra código por cada ingreso de ***k*** bits. La sintaxis que se utiliza es la siguiente:

P_Trellis = poly2trellis(ConstraintLength,CodeGenerator)

Los parámetros de entrada de la función *poly2trellis* se describen a continuación:

- **ConstraintLength:** es un vector fila de longitud ***k*** y especifica el retardo de los flujos de bits de entrada al codificador o la longitud de restricción. Es decir, especifica la memoria ***m*** del codificador.
- **CodeGenerator**: es una matriz de números octales de longitud ***k*** por ***n*** que especifica las conexiones de salida para los bits de entrada al codificador, es decir los polinomios generadores. Entonces se puede decir que ***n*** es igual a la cantidad de polinomios generadores que se tienen.

La salida *P_Trellis* devuelve una estructura de trellis que se puede utilizar como entrada para las funciones *convenc* y *vitdec*. La función *convenc* se utiliza para implementar el codificador convolucional y *vitdec* se utiliza para implementar el decodificador convolucional. En esta sección solo se describe la función *convenc* debido a que es la que se utiliza en el procesamiento de los datos en transmisión.

La función *convenc* de Matlab codifica convolucionalmente una secuencia de bits [20], es decir, esta función implementa un codificador convolucional con una estructura de trellis previamente creada en base a polinomios generadores y a una longitud de restricción. La sintaxis que se utiliza para implementar el codificador convolucional es:

CodConv = convenc(msg, P_Trellis)

Los parámetros de entrada se describen a continuación:

- **msg:** es el mensaje binario, se especifica como un vector de bits.

- **P_Trellis:** es la estructura de trellis que se obtuvo previamente mediante la función *poly2trellis*.

2.3.3.2.1. Codificador convolucional (2,1,3) en Matlab

Para implementar en Matlab el codificador convolucional con el que se ha venido trabajando en los ejemplos teóricos anteriores. Primero hay que recordar que el codificador convolucional se suele definir con los parámetros y la sintaxis (***n***, ***k***, ***m***). El codificador convolucional a implementar tiene la sintaxis (2, 1, 3). Para implementarlo se crea un script llamado *EjemploCodConv.m*.

El código inicia por crear una estructura de trellis con polinomios generadores $g_1 = (101) = 5_{oct}$ y $g_2 = (111) = 7_{oct}$ y con un valor de memoria del codificador igual a 3. Esta parte del código se presenta a continuación.

```
m=3; % m es el tamaño de registro de la memoria
g1=5; % Polinomio generador
g2=7; % Polinomio generador
% Creción de la estructura de Trellis
P_Trellis = poly2trellis(m, [g1 g2]);
```

En la sección de código anterior el vector fila ***m*** tiene una longitud igual a uno, por lo que el flujo de bits al ingreso del codificador (***k***) es igual a 1. Debido a que existen dos polinomios generadores, el parámetro ***n*** es igual a 2. Entonces resulta que la tasa de codificación $R_c = (1/2)$. La estructura de trellis se almacena en la variable *P_Trellis*.

La estructura se almacenada en la variable *P_Trellis* se puede ver en la Figura 2.39. Dentro de la estructura de trellis se aprecian cinco campos, que se detallan a continuación.

- **numInputSymbols:** Es el número de símbolos de entrada, representado como un valor escalar 2^k, en donde ***k*** representa el número de flujos de bits de entrada al codificador. Como ***k***=1, el número de símbolos de entrada es igual a 2, que corresponden a los posibles bits de entrada 0 o 1.
- **numOutputSymbols:** Es el número de símbolos de salida, toma el valor de un valor escalar 2^n, en donde ***n*** es el número de bits de salida por cada ***k*** bits de entrada. Como ***n***=2, este parámetro de la estructura toma el valor de 4, es decir que se tendrán 4 posibles combinaciones de bits codificados que corresponden a 00, 01, 10 y 11.
- **numStates:** es el número de estados en el codificador y es devuelto como escalar. Este parámetro toma el valor de 4, ya que los posibles estados son 00, 01, 10 y 11.

- **nextStates:** es la matriz de estados siguientes para todas las combinaciones de estados actuales y entradas actuales, devueltos como una matriz de tamaño numStates por 2^k. Es decir que el tamaño de esta matriz es de 4 x 2.
- **Outputs:** es una matriz de números octales que contiene las salidas para todas las combinaciones de estados actuales y entradas actuales, es decir las salidas codificadas. La matriz devuelta es de tamaño numStates por 2^k, en este caso el tamaño de esta matriz es de 4 x 2.

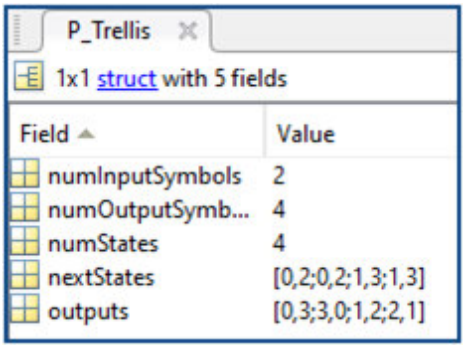


Figura 2.39. Estructura de trellis creada.

En la Figura 2.40 se presentan los campos *nextStates* y *outputs* de la estructura de trellis creada. También se muestra el diagrama de trellis creado previamente de manera manual, con el objetivo de explicar los campos mencionados. Entonces si el estado siguiente resulta ser 2 (10), los estados actuales posibles son 00 y 01 y las salidas codificadas asociadas a estos estados son 3 (11) y 0 (00) respectivamente. El estado siguiente 2 se señala con color amarillo y las salidas codificadas asociadas al estado actual se señalan con color rojo o verde, dependiendo de cuál sea el estado actual.

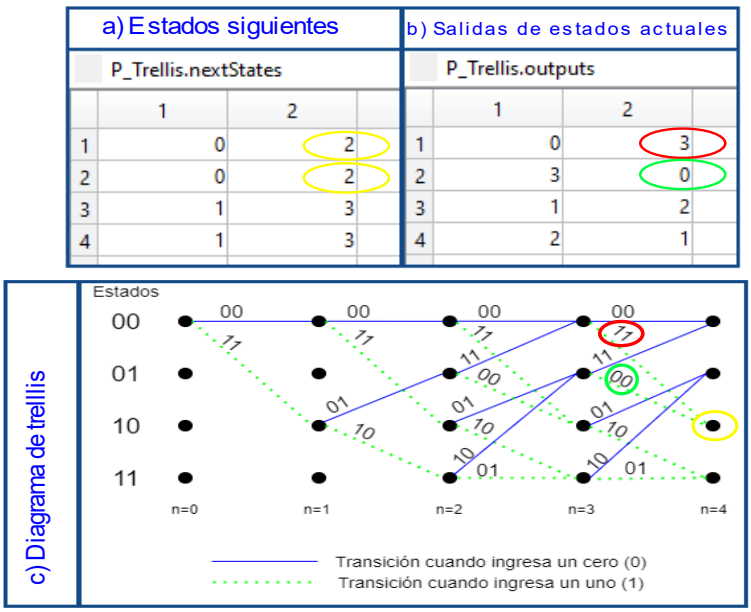


Figura 2.40. a) Estados siguientes de la estructura de trellis creada. b) Salidas de los estados actuales de la estructura de trellis creada. c) Diagrama de trellis.

La siguiente parte de código del script *EjemploCodConv.m*, utiliza la función *convenc* para codificar convolucionalmente la secuencia de bits 1101, empleando la estructura de trellis creada previamente. El código necesario para esta implementación se presenta debidamente comentado a continuación.

```
msg=[1 1 0 1]; % bits a codificar
% Codificación convolucional:
CodConv = convenc(msg, P_Trellis);%CodConv es la salida del codificador
```

Como resultado al ingreso de los bits 1101 se tiene una secuencia codificada 11101000. Esta secuencia concuerda con los resultados obtenidos anteriormente en la parte teórica de la codificación convolucional. En la Figura 2.41 se muestra las variables en el espacio de trabajo, dentro de las cuales se señalan con rojo el vector con los bits de entrada y con verde la secuencia de bits codificados.

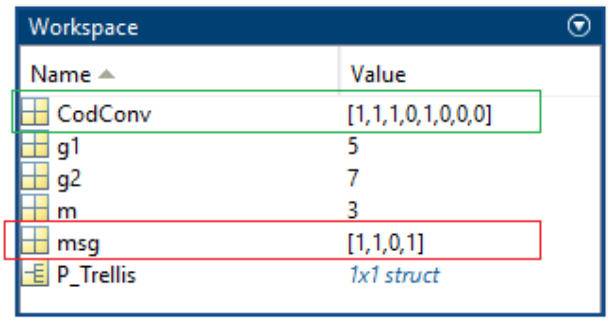


Figura 2.41. Variables del codificador con bits de entrada señalados con rojo y los bits codificados identificados en verde.

2.3.3.3. Codificador Convolucional aplicado al archivo de texto

En la etapa actual llamada *Codificador Convolucional* se implementa un codificador (2, 1, 3), con polinomios generadores $g_1 = 5_{oct}$ y $g_2 = 7_{oct}$ y con una tasa de codificación $R_c = (1/2)$, es decir que se utiliza el mismo codificador con el cual se ha venido trabajando en los ejemplos de codificación convolucional anteriores. Los bits serializados se van a codificar tal que, la longitud de la secuencia de bits codificados sea igual al número de portadoras de datos del símbolo OFDM. En este proyecto se utiliza un símbolo OFDM que se conforma de una portadora DC, 40 portadoras de datos, 19 portadoras de guarda y 4 portadoras pilotos. Los fundamentos teóricos relacionados a OFDM se abordan en la etapa posterior llamada *Creación del símbolo OFDM*.

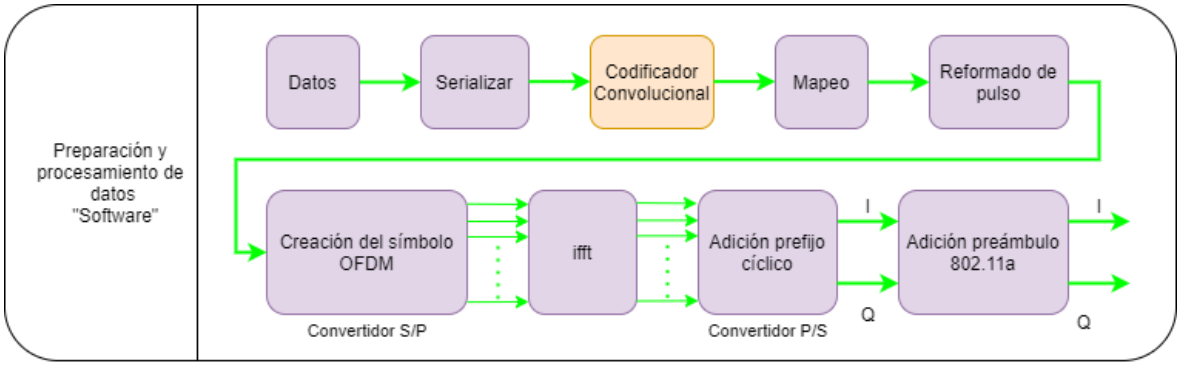


Figura 2.42. Etapas de transmisión con la etapa Codificador Convolucional identificada en color naranja.

Para el cálculo del número de bits serializados a codificar, además del número de portadoras de datos por cada símbolo OFDM, hay que tener en cuenta la etapa Mapeo y la etapa Reformado de pulso.

En la etapa Mapeo, dependiendo de la técnica de modulación con la que se trabaje, el número de bits representado por cada símbolo modulado (*Nbps*) va a cambiar. Este número se puede calcular en base al número de los puntos de constelación (*M*) de la técnica de mapeo, mediante la ecuación 2.4.

$$Nbps = log_2(M) \tag{2.4}$$

En la Tabla 2.6 se muestran los valores de *M* y *Nbps* para cada técnica de modulación. Los fundamentos teóricos relacionados a la etapa Mapeo se abordarán posteriormente.

Tabla 2.6. Número de puntos en la constelación (*M*) y número de bits por símbolo modulado (*Nbps*) para cada técnica de Mapeo.

Tipo de mapeo	Número de puntos en la constelación (*M*)	Número de bits por símbolo modulado (*Nbps*)
BPSK	2	1
QPSK	4	2
16-QAM	16	4
64-QAM	64	6

En la etapa Reformado de pulso cada símbolo de entrada es representado por un determinado número de muestras. En este proyecto se utiliza 4 muestras para representar cada símbolo a la entrada de dicha etapa. En otras palabras, la cantidad de símbolos que se tendrán a la salida de esta etapa es cuatro veces mayor a la cantidad de símbolos a la entrada.

El número de bits serializados a codificar, por cada símbolo OFDM, se puede calcular mediante la siguiente fórmula [16].

$$Nbits = NPdatos * Nbps * \frac{1}{Nsps} * R_c \quad (2.5)$$

En donde,

- *Nbits* es el número de bits serializados a codificar.
- *NPdatos* es el número de portadora de datos del símbolo OFDM.
- *Nbps* es el número de bits por cada símbolo modulado.
- *Nsps* es el número de muestras por cada símbolo, de la etapa *Reformado de pulso*.
- R_c es la tasa de codificación del codificador convolucional.

Por ejemplo, como *NPdatos* es igual a 40, *Nsps* es igual a 4, la tasa de codificación $R_c = \frac{1}{2}$ y suponiendo que se utilice una técnica de mapeo QPSK (*Nbps=2*), se tiene que *Nbits* es igual a:

$$Nbits = 40 * 2 * \frac{1}{4} * \frac{1}{2} = 10\ bits$$

En el prototipo, siempre que se utiliza la corrección de errores, el valor de *NPdatos* y de R_c se mantienen constantes. El parámetro *Nbps* cambia dependiendo de la técnica de mapeo que se utilice, estos valores se presentaron en la Tabla 2.6. El parámetro *Nsps* es igual a 4 cuando la técnica de reformado de pulso está activada y Nsps es igual a 1 cuando el reformado de pulso está desactivado. En la Tabla 2.7 se presenta un resumen con los posibles valores que puede tomar *Nbits* dependiendo del tipo de mapeo y del reformado de pulso.

Tabla 2.7. Tabla resumen del valor que toma *Nbits* en función del tipo de mapeo y reformado de pulso, con *NPdatos=40*, *Nsps=4* y $R_c = \frac{1}{2}$.

Tipo de mapeo	Reformado de pulso	Número de bits serializados a codificar por símbolo OFDM (*Nbits*)
BPSK	On	5
	Off	20
QPSK	On	10
	Off	40
16-QAM	On	20
	Off	80
64-QAM	On	30
	Off	120

Nota: Se podría utilizar un número de bits de entrada al codificador de manera fija, para que la etapa de codificación sea independiente de las siguientes etapas. El tamaño que se

recomienda utilizar es de 240, ya que todos los valores de la Tabla 3.7 son múltiplos del mismo.

En las etapas anteriores se implementó los scripts *abrir_archivo.m* y *serializar_datos.m*, correspondientes a las etapas de Datos y Serializar respectivamente. A la salida de la etapa *Serializar* se obtuvo un vector llamado *datosSerializados* que contiene los bits serializados correspondientes a un archivo de texto. Para facilidad de visualización de las nuevas variables en el espacio de trabajo, se eliminan las variables generadas en la etapa Serializar a excepción del vector *datosSerializados*, mediante el siguiente comando.

```
% Borrar variables en la etapa codificador convolucional:
clearvars -except datosSerializados % Borrar todas las variables
% excepto la variable datosSerializados
```

Para implementar la etapa actual llamada *Codificador Convolucional* se crea un script en Matlab llamado *cod_conv.m*. En dicho script se inicia por copiar los datos de la variable *datosSerializados* en una nueva variable llamada *datosBina*. Luego se establece el valor del número de portadoras de datos por símbolo OFDM, que toma un valor de 40. Después se selecciona el tipo de mapeo a utilizar. Para esto se utiliza un condicional if en donde en base al valor de la variable *seleccionMapeo* se escoge el número de puntos en la constelación. Para el mapeo BPSK la variable *seleccionMapeo* es igual a 10, para QPSK *seleccionMapeo* es igual a 20, para 16-QAM *seleccionMapeo* es igual a 30 y para 64-QAM *seleccionMapeo* es igual a 40.

Para este script y para el desarrollo del ejemplo de procesamiento y transmisión del archivo de texto, se ha optado por utilizar una modulación QPSK. El producto entregable final, es decir, el prototipo de comunicación inalámbrica, sí cuenta con las modulaciones digitales; BPSK, QPSK, 16-QAM y 64-QAM.

Entonces se configura la variable *seleccionMapeo* en 20, para seleccionar el mapeo QPSK, con este mapeo los puntos de constelación (*M*) es igual a 4. Luego se calcula los bits que se utilizan por cada símbolo mapeado con la ecuación 2.4 y se almacena en *Nbps_M*. A continuación, se establece el número de muestras por símbolo de la etapa Reformado de pulso, este valor corresponde a 4. Después recordando que la sintaxis del codificador convolucional es (***n***, ***k***, ***m***), se establece los parámetros en (2, 1, 3), los polinomios generadores son $g_1 = 5_{oct}$ y $g_2 = 7_{oct}$ y la tasa de codificación $R_c = (k/n)$. Luego se calcula el número de bits serializados que se codificarán por cada símbolo OFDM y se almacena en la variable *Nbits*.

Paso seguido se calcula el número total de símbolos OFDM que se van a tener, dividiendo la longitud del vector de bits serializados para la variable *Nbits*. El valor obtenido sirve para calcular, en caso de ser necesario, un relleno de ceros del vector *datosBina*. En la variable *datosConPadd* se almacena los datos serializados seguidos de un relleno de ceros.

Luego se inicializa una variable llamada *datosCodConv*, la cual se utiliza posteriormente para almacenar los bits codificados de forma serial. Después se crea una estructura de trellis con un tamaño de registro de memoria ***m*** y con los polinomios generadores g_1 y g_2. Con la estructura creada se puede codificar los bits serializados, tal que con un lazo *for()* se recorra los bits del vector *datosConPadd* en múltiplos *Nbits*. Las secuencias codificadas y serializadas se almacenan en el vector *datosCodConv*. Finalmente, se copia este vector en una variable llamada *datosBinario*, la cual consideraremos como la variable de salida de esta etapa.

El código descrito en párrafos anteriores y correspondiente al script *cod_conv.m* se presenta a continuación debidamente comentado.

```
% Script para mostrar un ejemplo para el codificador convolucional
% Como argumento de entrada se tiene un vector fila tipo double,
% este vector se obtiene en el scritp serializar_datos.m
% datosSerializados es el vector de entrada
datosBina=datosSerializados; % copiar el vector de entrada
% Primero se define los valores a considerar de las etapas posteriores
NPdatos=40; % Número de portadoras de datos por símbolo OFDM
% Para elegir el tipo de mapeo que se pretende utilizar se utiliza la
% variable seleccionMapeo, en donde si se quiere seleccionar BPSK es
% igual a 10, para QPSK es igual a 20, para 16-QAM es igual a 30 y
% para 64-QAM es igual a 40:
seleccionMapeo=20; % Se ha seleccionado mapeo QPSK
% Se selecciona el tipo de mapeo con un if
if (seleccionMapeo==10)
    % BPSK
    M = 2; % Puntos en la constelación (símbolos); 2 para BPSK
elseif (seleccionMapeo==20)
    % QPSK
    M = 4; % Puntos en la constelación (símbolos); 4 para QPSK
elseif (seleccionMapeo==30)
    % 16-QAM
    M = 16; % Puntos en la constelación (símbolos); 16 para 16-QAM
elseif (seleccionMapeo==40)
    % 64-QAM
    M = 64; % Puntos en la constelación (símbolos); 64 para 64-QAM
end
Nbps_M=log2(M); % Se calcula los bits por símbolo en según la
% opción de mapeo que se ha escogido
Nsps=4; % Muestras por símbolo en la etapa reformado
% Parámetros del codificador convolucional:
nc=2; % n es la longitud por cada ingreso de k bits
kc=1; % k es los bits de entrada
mc=3; % m es el tamaño de registro de la memoria
g1=5; % Polinomio generador
```

```
g2=7; % Polinomio generador
Rcc=kc/nc; % tasa del codificador convolucional
% Cálculo del número de bits serializados que se tiene por cada símbolo
% OFDM:
Nbits=Nbps_M*NPdatos*Rcc*(1/Nsps);
% Cálculo para rellenar el vector de entrada, y así obtener un vector
% que sea múltiplo de Nbits:
NumSimb=ceil(length(datosBina)/Nbits); % Número de simbolos OFDM
longitudPadd=Nbits*NumSimb-length(datosBina); % Longitud de relleno
vector_padd=zeros(1,longitudPadd); % vector de relleno de ceros
datosConPadd=[datosBina vector_padd]; % Datos serializados con relleno
% Creción de la estructura de Trellis:
P_Trellis = poly2trellis(mc, [g1 g2]);
% Codificador:
datosCodConv=[]; % se inicializa la variable que va a almacenar los
% datos codificados y serializados
% Se recorre los todos los símbolos OFDM con un lazo for
for i=1:NumSimb
    auxDatosCc = datosConPadd(1,(Nbits*(i-1)+1):Nbits*i); % variable
    % auxiliar para almacenar los datos a codificar
    % Codificación convolucional:
    CodConv = convenc(auxDatosCc, P_Trellis);
    datosCodConv=[datosCodConv CodConv]; % Datos de salida
end
datosBinario=datosCodConv; % Copiar los datos de salida en una variable
```

Considerando que las variables de los scripts de etapas anteriores se eliminaron, las variables del espacio de trabajo obtenidas en esta etapa se muestran en la Figura 2.43.

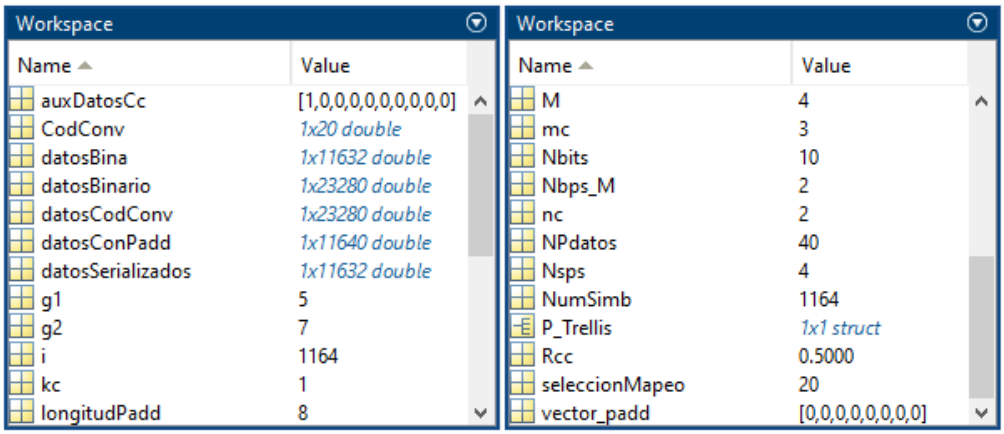


Figura 2.43. Variables del espacio de trabajo luego de ejecutar *cod_conv.m*

La variable *Nbits* toma un valor de 10, lo que significa que se codificaron en grupos de 10 bits. Como la tasa de codificación del codificador convolucional Rcc=0.5, se tendrá 20 bits codificados por cada 10 bits serializados. En la Figura 2.44 se muestran los primeros 12 bits del vector de entrada *datosSerializados* y los primeros 24 bits del vector de salida *datosBinario*. Además, se señalan con rojo los primeros 10 bits de entrada y los primeros 20 bits codificados.

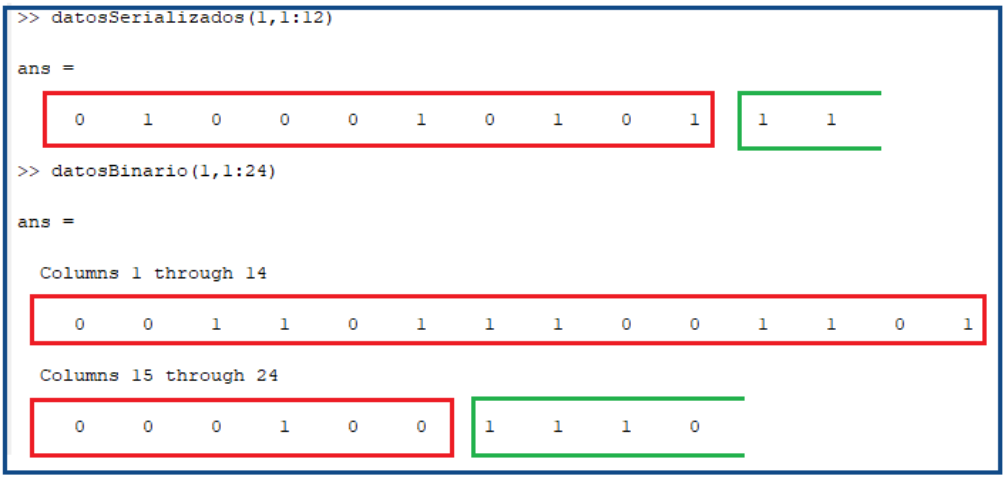


Figura 2.44. Primeros datos del vector de entrada de la etapa y del vector de bits codificados.

2.3.4. ETAPA DE MAPEO

La etapa Mapeo se identifica en la Figura 2.45 con color naranja dentro de la parte de preparación y procesamiento del archivo de texto en transmisión. En esta etapa se busca implementar diferentes técnicas de mapeo para luego, en recepción, comparar el desempeño de las mismas al aumentar la cantidad de símbolos en el diagrama de constelación. En el prototipo de comunicación inalámbrica se utiliza cuatro esquemas de modulación que son BPSK, QPSK, 16-QAM y 64-QAM.

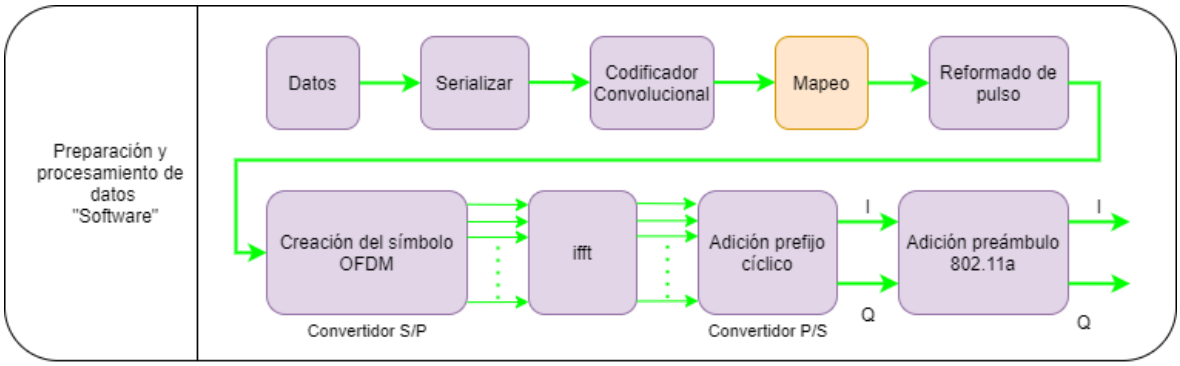


Figura 2.45. Etapas de transmisión con la etapa Mapeo identificada.

2.3.4.1. Fundamentos teóricos

La modulación digital consiste en tener un mensaje digital que modula una señal de forma de onda continua, manipulando la información de amplitud y fase de la señal durante cada periodo de tiempo [4]. Algunas de las técnicas de modulación digital son; la modulación por desplazamiento de amplitud del inglés Amplitude Shift Keying (ASK), modulación por desplazamiento de frecuencia del inglés Frequency Shift Keying (FSK), modulación por

desplazamiento de fase del inglés Phase Shift Keying (PSK) y modulación de amplitud en cuadratura del inglés Quadrature Amplitude Modulation (QAM) [21]. Sin embargo, no todas estas técnicas de modulación digital pueden ser utilizadas con la técnica de transmisión OFDM (Orthogonal Frequency Division Multiplexing).

OFDM consiste en la multiplexación de un conjunto de ondas portadoras ortogonales. Estas portadoras deben estar previamente moduladas con alguna técnica de modulación digital, sin que afecte la ortogonalidad de las portadoras [16]. El estándar WLAN IEEE 802.11A establece que los tipos de modulación a utilizar para las portadoras de datos de OFDM son BPSK, QPSK, 16-QAM y 64-QAM [8]. Por lo que, se analizarán brevemente estos cuatro tipos de modulación digital.

Cabe mencionar que las técnicas de modulación digital son diseñadas mapeando un grupo de bits de información en un diagrama de constelación de dos dimensiones, es decir en un plano. Los ejes de este plano son el componente en fase (I), correspondiente al eje horizontal, y el componente en cuadratura (Q), correspondiente al eje vertical. Los componentes en fase y cuadratura corresponden a un número complejo, que se obtiene a partir de uno o de un grupo de bits. En la ecuación 2.3 d_k representa al grupo de bits que se iguala a varias representaciones de un número complejo.

$$d_k = A_k * (e^{j\emptyset_k}) = A_k * [\cos(\emptyset_k) + sen(\emptyset_k) * i] = I_k + Q_k i \quad (2.6)$$

Para asignar los puntos de las costelaciones a cada grupo de bits de entrada se utiliza el código GRAY, con el objetivo de reducir el BER [16]. Este código consiste en que los valores binarios sucesivos difieran solamente de un dígito. El ordenar los puntos de la constelación mediante el código GRAY permite reducir en un bit el error que pueda producirse, por cada símbolo. Es decir, que si al momento de recuperar los bits, este cae dentro de los límites de otro símbolo cercano, solo existe un error de un bit, reduciendo así el BER. En la Tabla 2.8 se muestra un ejemplo del código GRAY comparando la representación, decimal, binaria y gray.

Tabla 2.8. Ejemplo del código GRAY.

Representación Decimal	Representación Binaria	Código GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

- **Mapeo BPSK**

Modulación por desplazamiento de fase binaria (BPSK), en donde por cada bit de entrada se asigna una posición en el diagrama de constelación, es decir que se utiliza un bit por símbolo modulado. La posición correspondiente al ingreso de un 0L está separada 180° de la posición correspondiente a un 1L. El diagrama de constelación puede ser el que se presenta en la Figura 2.46, en donde los símbolos están separados 180° el uno del otro.

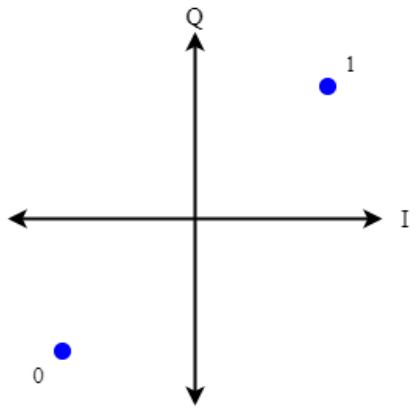


Figura 2.46. Diagrama de constelación para mapeo BPSK.

En la Tabla 2.9 se muestran los bits de entrada y su correspondiente símbolo mapeado, de la modulación BPSK.

Tabla 2.9. Modulación BPSK

Bits	Símbolos
0	- 0.707 - 0.707i
1	+ 0.707 + 0.707i

- **Mapeo QPSK**

La modulación por desplazamiento de fase cuaternaria o cuadrafásica (QPSK) utiliza dos bits de entrada para crear cada símbolo del diagrama de constelación. Los símbolos del diagrama de constelación se separan 90° tal como se puede ver en la Figura 2.47. Además de estar distribuidos de tal forma que se utiliza codificación gray

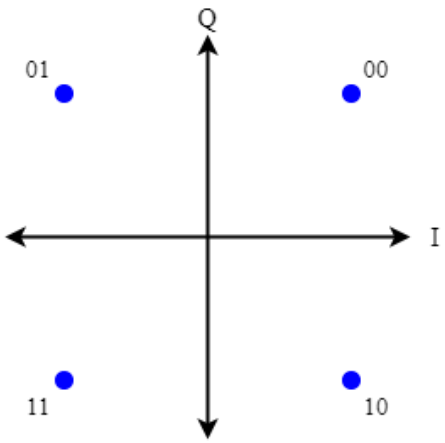


Figura 2.47. Diagrama de constelación para mapeo QPSK, con codificación GRAY.

En la Tabla 2.10 se muestran las combinaciones de bits de entrada y su correspondiente símbolo mapeado, de la modulación QPSK.

Tabla 2.10. Modulación QPSK

Bits	Símbolos
00	+0.707 + 0.707i
01	-0.707 + 0.707i
10	+0.707 - 0.707i
11	-0.707 - 0.707i

- **Mapeo 16-QAM**

La modulación digital QAM es una técnica en la cual el mensaje está contenido en la variación de la fase como en PSK y en la variación de amplitud como en ASK. La modulación QAM se basa en transmitir dos mensajes diferentes a través de un solo camino y además de desfasa en 90° una portadora de la otra [21]. En 16-QAM se tiene en total 16 símbolos y por cada símbolo se emplean 4 bits. En la Figura 2.48 se muestran todos los símbolos con los bits de entrada para el mapeo 16-QAM. Además, se muestran los valores

de fase (I) y cuadratura (Q) para cada símbolo mapeado, por lo que no es necesario detallar en una tabla separada. Recordar que la fase (I) representa el componente real y la componente en cuadratura la parte imaginaria.

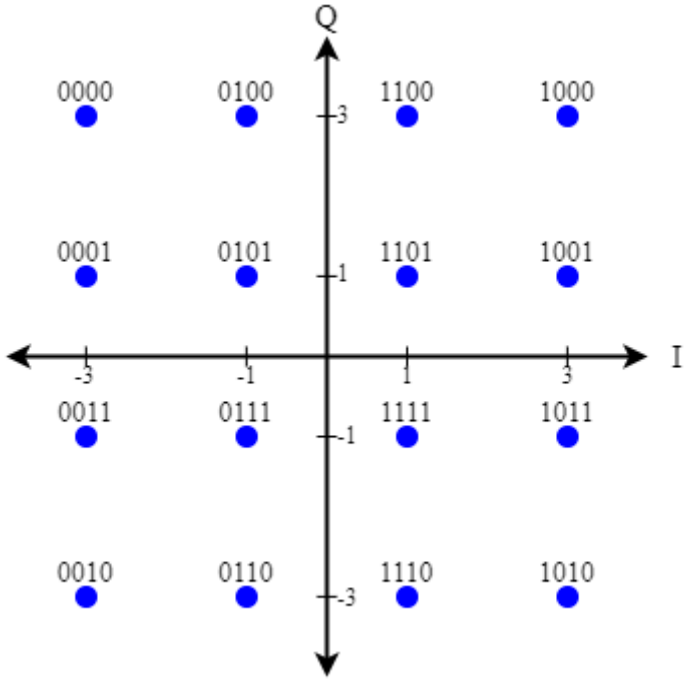


Figura 2.48. Diagrama de constelación para mapeo 16-QAM, con codificación GRAY.

- **Mapeo 64-QAM**

La modulación 64-QAM tiene en total 64 símbolos en el diagrama de constelación y en cada uno de ellos se emplean 6 bits. En la Figura 2.49 se muestran todos los símbolos para el mapeo 64-QAM, en este caso no se coloca la secuencia de bits correspondiente a cada símbolo, sino su representación en decimal, esto debido a que la cantidad de símbolos es bastante grande. Recordar que la componente en fase (I) representa la parte real y la componente en cuadratura la parte imaginaria, del número complejo correspondiente a cada símbolo.

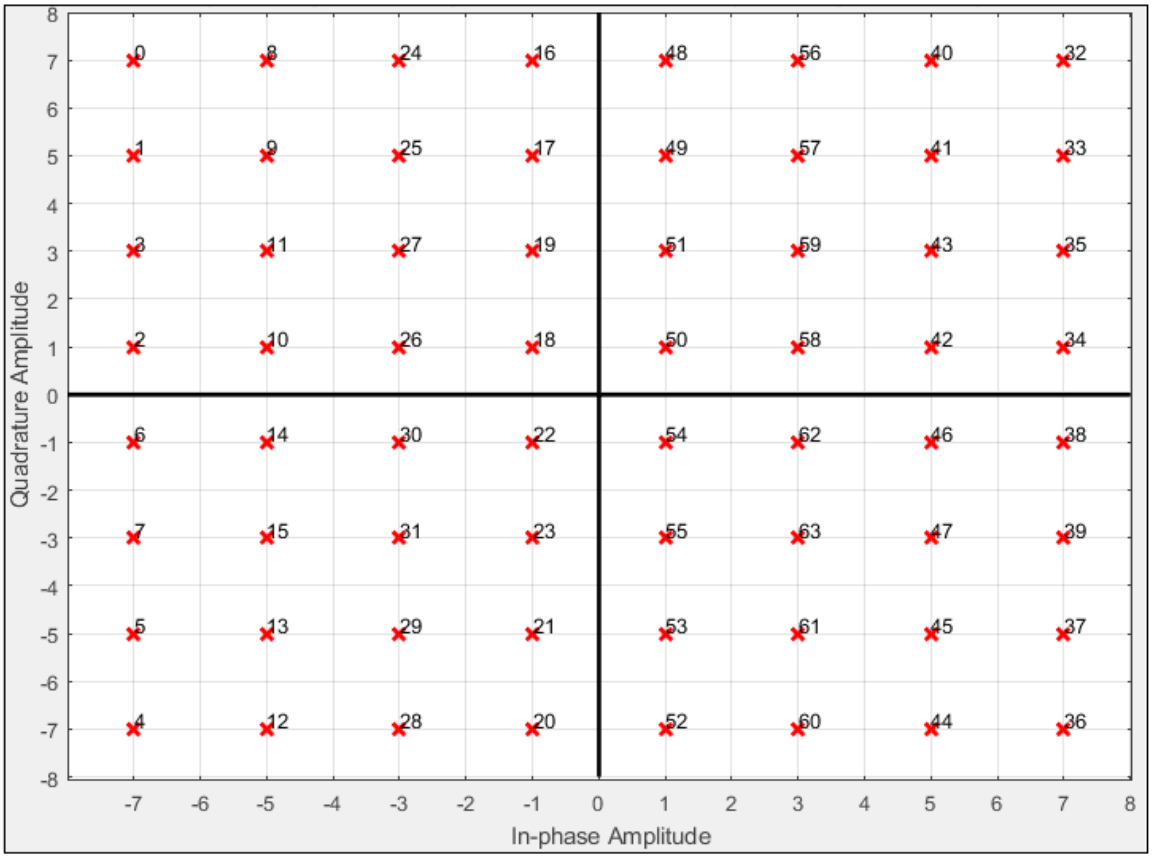


Figura 2.49. Diagrama de constelación para mapeo 64-QAM, con codificación GRAY.

En general el número de bits para representar cada símbolo del diagrama de constelación se puede calcular mediante la ecuación 2.4, en donde *M* representa el número de símbolos en el diagrama de constelación.

$$Número\ de\ bits = \log_2(M) \tag{2.7}$$

Un aspecto importante a considerar es que mientras mayor sea el número de estados en el diagrama de constelación mayor es la velocidad de transmisión, pero la probabilidad de error se incrementa [21]. Tomando en cuenta la modulación digital BPSK, QPSK, 16-QAM y 64-QAM, se tiene que para una misma potencia de transmisión BPSK presentará la menor cantidad de errores y 64-QAM la mayor cantidad de errores.

Otro aspecto importante a considerar es el factor de normalización de energía para lograr que todos los símbolos de la constelación tengan una potencia media igual a 1 [16]. Los fatores de normalización, para las técnicas de mapeo descritas anteriormente, se presentan en la Tabla 2.11.

Tabla 2.11. Factores de normalización de varias técnicas de mapeo [16].

Mapeo	Factor de normalización
BPSK	
QPSK	$1/\sqrt{2}$
16-QAM	$1/\sqrt{10}$
64-QAM	$1/\sqrt{42}$

Cuando no se realiza la normalización de los datos mapeados con QPSK, 16-QAM o 64-QAM, la amplitud de la parte de los datos va a ser mayor a la amplitud del preámbulo IEEE 802.11a. Esto genera que en recepción el preámbulo se confunda con el ruido del canal inalámbrico.

En la Figura 2.50 se muestra un ejemplo de mapeo 16-QAM sin utilizar el factor de normalización, los datos tienen una amplitud mayor a la del preámbulo. En el prototipo de comunicación inalámbrica, antes de transmitir el bloque de datos (incluido el preámbulo) al canal hay que normalizar este bloque para poder transmitir los datos utilizando el equipo SDR Adalm Pluto. Esto genera que la amplitud del preámbulo sea muy baja respecto a la amplitud de los datos.

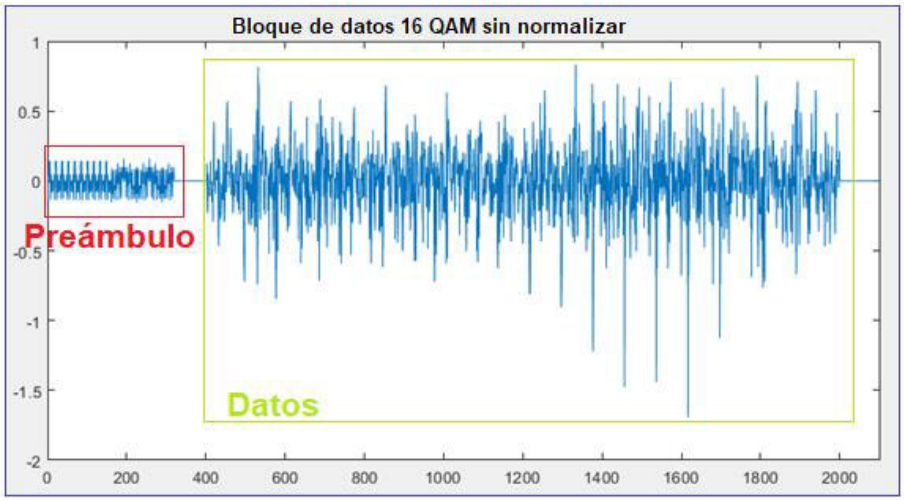


Figura 2.50. Ejemplo de mapeo 16-QAM sin normalizar y con el preámbulo a la izquierda.

2.3.4.2. Implementación en Matlab

En esta sección se va a implementar en Matlab las técnicas de mapeo tratadas anteriormente, es decir el mapeo BPSK, QPSK, 16-QAM y 64-QAM. Con el objetivo de no crear muchos scripts innecesariamente, se ha optado por implementar todas las técnicas de mapeo en un solo script llamado *mapeo.m*. En dicho script se separa por secciones las diferentes técnicas de modulación digital. Luego de implementar ejemplos para todas las

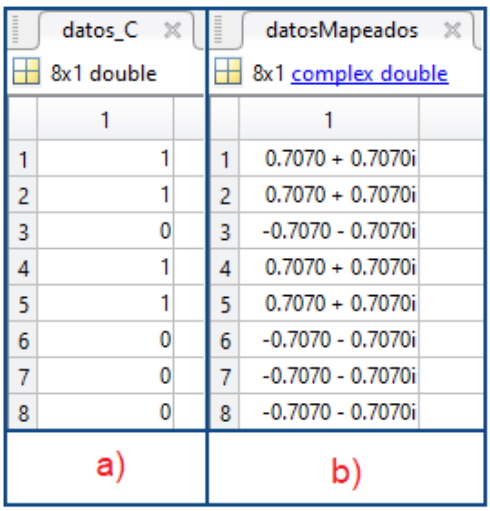
técnicas de mapeo, en una sección aparte se va a seguir con el mapeo de los datos codificados convolucionalmente, obtenidos en la etapa Codificador Convolucional.

2.3.4.2.1. Mapeo BPSK en Matlab

Para implementar el mapeo BPSK, la sección de código dentro del script *mapeo.m* es bastante sencilla, ya que solo se requiere discriminar dos valores 1 o 0 y asignar a cada uno de ellos un símbolo de la constelación. Esto se realiza recorriendo cada bit del vector de entrada mediante un lazo *for()* y con un condicional if*()* para verificar cual es el valor del bit actual. El código descrito para mapeo BPSK se presenta a continuación debidamente comentado. Cabe mencionar que se utiliza una secuencia de bits de entrada igual a 11011000.

```
%% Mapeo BPSK
datos_C=[1;1;0;1;1;0;0;0]; % Bits de entrada a mapear
% Con el lazo for se recorre los bits del vector de entrada
for i=1:length(datos_C)
    % Con el lazo if se discrimina si es igual a 1 y se asigna el
    % valor 0.707+0.707i, caso contrario es 0 y se asigna el valor de
    %  -0.707-0.707i
    if(datos_C(i)==1)
        datosMapeados(i,1)=complex(0.707,0.707);
    else
        datosMapeados(i,1)=complex(-0.707,-0.707);
    end
end
% El vector de salida es datosMapeados y tiene la misma longitud
% del vector de entrada.
% Factor de normalización = 1
```

En la Figura 2.51 se muestra una comparación de los bits de entrada con los símbolos mapeados con BPSK. El resultado corresponde a los valores descritos en la Tabla 2.9.



datos_C — 8x1 double

	1
1	1
2	1
3	0
4	1
5	1
6	0
7	0
8	0

a)

datosMapeados — 8x1 complex double

	1
1	0.7070 + 0.7070i
2	0.7070 + 0.7070i
3	-0.7070 - 0.7070i
4	0.7070 + 0.7070i
5	0.7070 + 0.7070i
6	-0.7070 - 0.7070i
7	-0.7070 - 0.7070i
8	-0.7070 - 0.7070i

b)

Figura 2.51. a) Bits de entrada. b) Símbolos mapeados con BPSK.

2.3.4.2.2. Mapeo QPSK en Matlab

Implementar el mapeo QPSK en Matlab, escribiendo código propio, resulta más complejo que en el caso del mapeo BPSK. Esto se debe a que se tiene un mayor número de puntos en el diagrama de constelación de QPSK.

El código para mapear QPSK se implementa en la segunda sección del script *mapeo.m*. El cual inicia por definir la secuencia de bits a mapear, la secuencia que se utiliza es 11011000. Teniendo en cuenta que el vector de salida no va a tener la misma longitud que el vector de entrada, se inicializa una variable auxiliar. La cual se utiliza para establecer la posición de los datos mapeados en el vector de salida.

Luego mediante un lazo *for()* se recorre en pares los bits del vector de entrada, debido a que QPSK utiliza 2 bits por cada símbolo. Dentro del lazo *for()* se discrimina si el primer bit de entrada corresponde a un 0 o a un 1, mediante un primer condicional i*f()*. Dentro de las dos opciones del primer *if()* se implementa otro *if()* que discrimina si el segundo bit es 0 o 1. Entonces cuando los bits de entrada corresponden a 00 se asigna el símbolo 0.707+0.707i, en caso de ser 01 se asigna el símbolo -0.707+0.707i, en caso de ser 10 se asigna el símbolo 0.707-0.707i y en caso de ser 01 se asigna el símbolo -0.707-0.707i. En la Tabla 2.10 se puede ver esta asignación para el mapeo QPSK.

Luego de mapear todos los bits de entrada se multiplica por el factor de normalización de la Tabla 2.11 para QPSK. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Mapeo QPSK
datos_C=[1;1;0;1;1;0;0;0]; % Bits de entrada a mapear
aux1=1; % Variable auxiliar para guardar los datos mapeados
% Con el lazo for se recorre los bits del vector de entrada tal que
% en cada iteracion se tome 2 bits
for i=1:2:length(datos_C)
    % Con el lazo if se discrimina si el primer bit es igual a 0
    % o es igual a 1
    if(datos_C(i)==0)
        % Si ingresa a este lazo el primer bit es igual a 0 y con
        % otro lazo se discrimina el valor del segundo bit
        if(datos_C(i+1)==0)
            % bits de entrada 00, datos mapeados igual a 0.707+0.707i
            datosM(aux1,1)=complex(0.707,0.707);
        else
            % bits de entrada 01, datos mapeados igual a -0.707+0.707i
            datosM(aux1,1)=complex(-0.707,0.707);
        end
    else
        % Si ingresa a este lazo el primer bit es igual a 1 y con
        % otro lazo se discrimina el valor del segundo bit
        if(datos_C(i+1)==0)
            % bits de entrada 10, datos mapeados igual a -0.707-0.707i
```

```
            datosM(aux1,1)=complex(0.707,-0.707);
        else
            % bits de entrada 11,datos mapeados igual a 0.707-0.707i
            datosM(aux1,1)=complex(-0.707,-0.707);
        end
    end
    aux1=aux1+1; % se suma 1 a la variable auxiliar
end
% Factor de normalización = 1/sqrt(2)
datosMapeados=(1/sqrt(2))*datosM; %El vector de salida es datosMapeados
```

En la Figura 2.52 se muestra una comparación de los bits de entrada con los símbolos mapeados con QPSK y con los símbolos normalizados. El resultado obtenido corresponde a lo planteado en la Tabla 2.8.

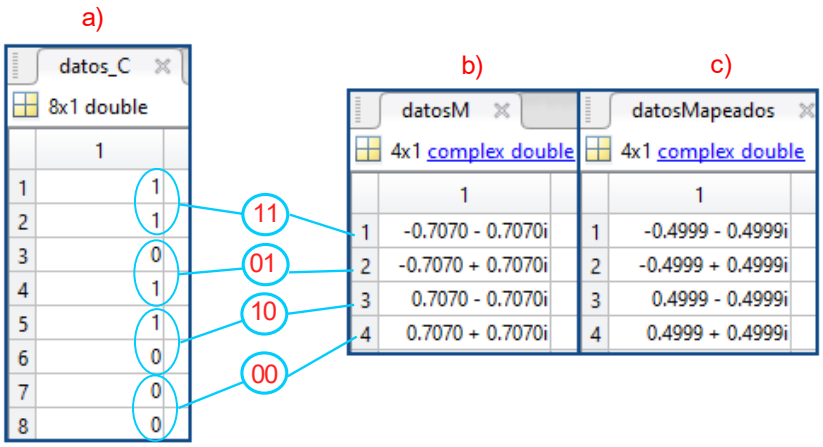


Figura 2.52. a) Bits de entrada. b) Símbolos mapeados con QPSK. c) Símbolos mapeados con QPSK normalizados.

Comparando el código implementado para el mapeo BPSK con el código para QPSK, se puede observar el aumento de complejidad al aumentar la cantidad de símbolos en la constelación. Es decir que para el mapeo 16-QAM o 64-QAM implementar la modulación de manera manual puede resultar bastante tedioso y ocupar varias líneas de código, complicándose la programación.

Afortunadamente Matlab cuenta con funciones directas para la implementación de modulación y demodulación digital.

Si se quiere implementar la modulación QPSK a través de funciones directas, Matlab dispone de la función *pskmod*. Esta función se utiliza para implementar la modulación digital PSK. La sintaxis que se utiliza es:

datosModulados = pskmod(datosEnt,M,ini_phase,symorder)

Los parámetros de entrada se describen a continuación [22]:

- **datosEnt:** Es el vector que contiene los símbolos de entrada a ser modulados, estos símbolos son representados por números decimales. Para QPSK los símbolos son 0, 1, 2 y 3 que corresponden a 00, 01, 10 y 11 respectivamente
- **M:** Es el orden de modulación, se especifica como una potencia de 2. Corresponde a la cantidad de símbolos en la constelación. Para QPSK es igual a 4.
- **ini_phase:** Es la fase inicial de la modulación PSK y el valor se especifica en radianes. El valor por defecto es 0, pero para el mapeo QPSK con el que se está trabajando se utiliza pi/4.
- **symorder:** Especifica el orden de los símbolos en el diagrama de constelación, también llamada la representación o codificación que se utiliza en los símbolos cercanos. Puede utilizarse codificación binaria o codificación gray, los cuales se especifican como 'bin' o 'gray', siendo el primero el valor por defecto. Para el mapeo QPSK con el que se está trabajando se asigna como 'gray'.

La función directa para mapeo PSK recibe como datos de entrada símbolos decimales. Por otro lado, a la salida de la etapa Codificador Convolucional se tiene un vector de bits. Entonces es necesario pasar de una representación binaria a una representación decimal. Para este objetivo se utiliza la función *reshape* y la función *bi2de* de Matlab.

La función *reshape* cambia la forma (dimensiones) de una matriz y la sintaxis que se utiliza es:

B = reshape(A,s1,s2)

En la sintaxis anterior, *B* resulta en una matriz de tamaño *s1* filas por *s2* columnas que contiene los elementos de la matriz *A*. Cuando se requiere calcular de manera automática el valor de filas o de columnas, para que *B* coincida con el número de elementos de *A*, *s1* o *s2* (solo uno de los dos) pueden tomar el valor de [].

La función *reshape* trabaja tal que, se recorre los elementos de cada columna de *A* y los coloca en cada columna de *B* según las dimensiones escogidas. Entonces para agrupar los bits que contiene el vector *A*, *s1* corresponde al número de bits que se utilizan por cada símbolo modulado y *s2* toma el valor de [], dando como resultado que cada columna represente a una combinación de bits

La función *bi2de* transforma cada fila de una matriz de números binarios a números decimales. La sintaxis que se utiliza es:

D = bi2de(b,flg)

En la sintaxis anterior *b* tiene que ser una matriz de números binarios, en la cual cada fila corresponde a la secuencia de bits a transformar a una representación decimal. El parámetro *flg* puede tomar los valores de 'right-msb' (valor por defecto) o 'left-msb'. El primer valor representa que el bit a la derecha es el más significativo y 'left-msb' representa el bit a la izquierda como el más significativo. Siendo este último valor con el cual se va a trabajar.

Debido que a la salida de la función *reshape* cada columna representa la combinación de bits a transformar en números decimales y que la función *bi2de* transforma cada fila en números decimales, es necesario obtener la matriz transpuesta de la salida de la función *reshape*. Esto con el objetivo de que, en dicha matriz, cada fila corresponda los dígitos binarios a representar en decimal. La matriz transpuesta de *B* se puede obtener mediante la expresión ***B=B .'***.

En una sección consecutiva dentro del script *mapeo.m* se implementa el mapeo QPSK con funciones directas. Se inicia por definir los bits a codificar, que es la misma secuencia utilizada anteriormente, correspondiente a 11011000. Luego se define la cantidad de puntos en la constelación y se calcula la cantidad de bits por símbolos a utilizar.

Después es necesario pasar los bits a símbolos decimales con las funciones *reshape* y *bi2de*. Luego se modula los números decimales con la función *pskmod*, con los parámetros que se mencionaron previamente. Finalmente, se normaliza los datos mapeados multiplicándolos por $1/\sqrt{2}$. El código debidamente comentado se presenta a continuación.

```
%% Mapeo QPSK
datos_C=[1;1;0;1;1;0;0;0]; % Bits de entrada a mapear
M = 4; % Puntos en la constelación (símbolos)
Nb=log2(M); % Bits por símbolo
%Pasar de binario a símbolos:
dataMatriz=reshape(datos_C,Nb,[]).'; %Nb columanas, cada fila
% representaun un símbolo
dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
% Modulador:
datosM1 = pskmod(dataSimb,M,pi/4,'gray'); % Datos modulados QPSK
% Factor de normalización = 1/sqrt(2)
datosMapeados1=(1/sqrt(2))*datosM1; % Vector de salida
```

Una vez ejecutada la sección de código se tiene como resultado en el espacio de trabajo las variables que se muestran en la Figura 2.53.

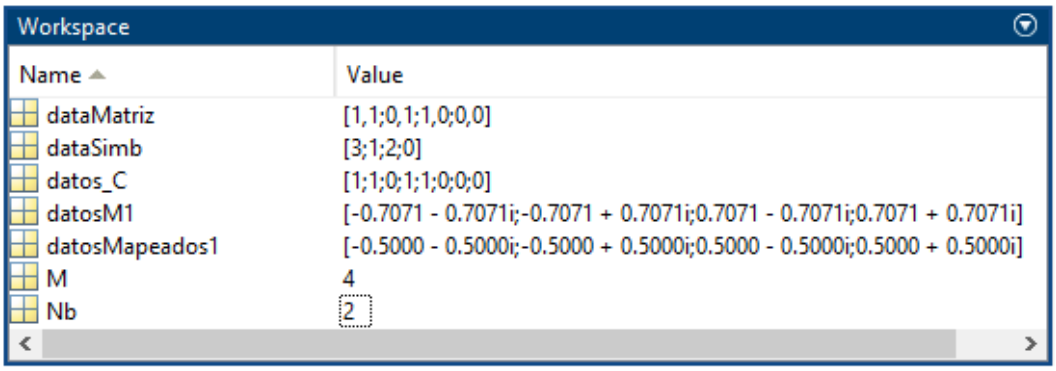


Figura 2.53. Variables del espacio de trabajo luego de ejecutar el mapeo QPSK con una función directa de Matlab.

En la Figura 2.54 se compara el resultado de mapear con QPSK la secuencia de bits 11011000, con código escrito de **manera manual** y con la **función directa** de Matlab. En dicha figura se presentan los datos mapeados sin normalizar y normalizados. Existen unas pequeñas diferencias en los valores obtenidos, pero se debe a que Matlab utiliza un mayor número de decimales. En general los valores obtenidos son iguales.

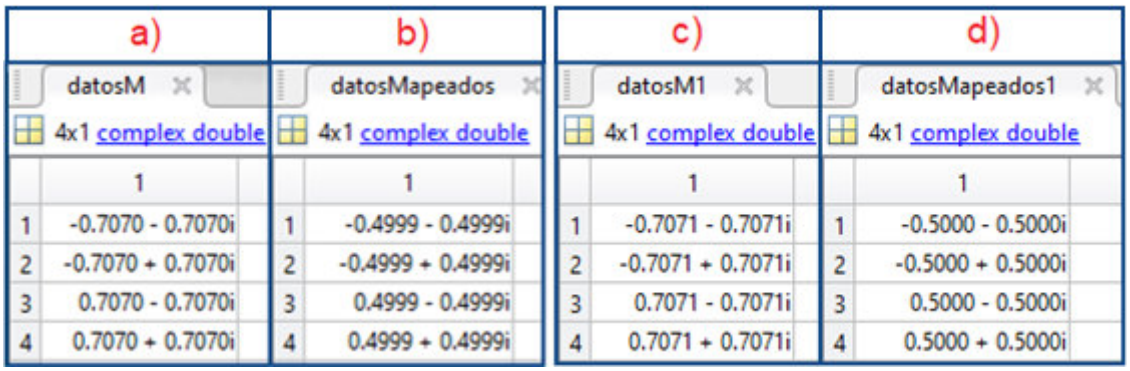


Figura 2.54. a) Símbolos mapeados manualmente con QPSK, sin normalizar. b) Símbolos normalizados mapeados manualmente con QPSK. c) Símbolos sin normalizar mapeados en QPSK con una función directa. d) Símbolos normalizados mapeados en QPSK con una función directa.

Comparando la cantidad de líneas de código de la implementación manual, con la implementación por función directa, el código por función directa resulta ser más corto que en la implementación manual. Debido a que en el prototipo de comunicación inalámbrica se implementan varias etapas de manera consecutiva, resulta una mejor opción reducir la cantidad de líneas de código.

Para el presente caso es mejor utilizar una función directa para implementar el mapeo QPSK. Lo mismo ocurre para la modulación digital 16-QAM y 64- QAM. Sin embargo, hay que tener en cuenta que las funciones directas pueden reemplazarse o eliminarse en versiones futuras de Matlab. A manera de recordatorio, cabe mencionar que esté trabajo se desarrolló en Matlab R2021a.

2.3.4.2.3. Mapeo 16-QAM en Matlab

Al igual que para el mapeo QPSK, para implementar el mapeo 16-QAM resulta una mejor opción utilizar una función directa. En general para implementar la modulación digital QAM se utiliza la función *qammod*. La sintaxis que se utiliza es:

y = qammod(datosEnt,M,symOrder)

Los argumentos de entrada se describen a continuación [23]:

- **datosEnt:** Es el vector que contiene los símbolos de entrada a ser modulados y estos símbolos son representados por números decimales. El rango de valores debe estar entre 0 a *M-1*.
- **M:** Es el orden de modulación, se especifica como una potencia de 2. Corresponde a la cantidad de símbolos en la constelación.
- **symorder:** Especifica el orden de los símbolos en el diagrama de constelación, también llamada la representación o codificación que se utiliza en los símbolos cercanos. Puede utilizarse codificación gray, binaria o un orden personalizado. El valor por defecto es codificación gray especificado como 'gray', para especificar la codificación binaria se utiliza 'bin' o bien se puede utilizar un vector con valores entre 0 y *M-1*.

El ejemplo de mapeo 16-QAM se implementa en una sección del script *mapeo.m*. Debido a la cantidad de símbolos en la constelación de 16-QAM, se ha optado por generar un vector de números decimales para utilizarlo como datos de entrada y no utilizar un vector de bits. El código inicia por definir el parámetro *M*, en base al cual se genera el vector con los números decimales de entrada. Luego se modulan los datos de entrada, especificando el orden de modulación *M*. Después se normaliza los datos modulados multiplicándolos por $1/\sqrt{10}$. Finalmente se grafica el diagrama de constelación de los datos modulados con y sin normalizar. El código descrito se presenta a continuación, debidamente comentado.

```
%% Mapeo 16-QAM
M = 16; % Puntos en la constelación (símbolos)
datos_C=(0:M-1).'; % vector de entrada
% Modulador:
% Factor de normalización = 1/sqrt(10)
datosM = qammod(datos_C,M); % Modulación 16-QAM (datos sin normalizar)
datosModulados = (1/sqrt(10))*datosM;% Símbolos modulados normalizados
%Grafíco los puntos del diagrama de constelación:
subplot(1,2,1)
plot(datosM,'r.','MarkerSize',20);
title('Mapeo 16-QAM sin Normalizar')
xlabel('I (fase)');
ylabel('Q (cuadratura)');
```

```
grid on
axis([-3.5 3.5 -3.5 3.5])
subplot(1,2,2)
plot(datosModulados,'b.','MarkerSize',20);
title('Mapeo 16-QAM Normalizado')
xlabel('I (fase)');
ylabel('Q (cuadratura)');
axis([-1.1 1.1 -1.1 1.1])
grid on
```

En la Figura 2.55 se muestra los números decimales de entrada al modulador 16-QAM, los datos modulados sin normalizar y los datos modulados normalizados.

a)

datos_C — 16x1 double

	1
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14
16	15

b)

datosM — 16x1 complex double

	1
1	-3.0000 + 3.0000i
2	-3.0000 + 1.0000i
3	-3.0000 - 3.0000i
4	-3.0000 - 1.0000i
5	-1.0000 + 3.0000i
6	-1.0000 + 1.0000i
7	-1.0000 - 3.0000i
8	-1.0000 - 1.0000i
9	3.0000 + 3.0000i
10	3.0000 + 1.0000i
11	3.0000 - 3.0000i
12	3.0000 - 1.0000i
13	1.0000 + 3.0000i
14	1.0000 + 1.0000i
15	1.0000 - 3.0000i
16	1.0000 - 1.0000i

c)

datosModulados — 16x1 complex double

	1
1	-0.9487 + 0.9487i
2	-0.9487 + 0.3162i
3	-0.9487 - 0.9487i
4	-0.9487 - 0.3162i
5	-0.3162 + 0.9487i
6	-0.3162 + 0.3162i
7	-0.3162 - 0.9487i
8	-0.3162 - 0.3162i
9	0.9487 + 0.9487i
10	0.9487 + 0.3162i
11	0.9487 - 0.9487i
12	0.9487 - 0.3162i
13	0.3162 + 0.9487i
14	0.3162 + 0.3162i
15	0.3162 - 0.9487i
16	0.3162 - 0.3162i

Figura 2.55. a) Símbolos de entrada b) Símbolos modulados con 16-QAM, sin normalizar. c) Símbolos normalizados modulados con 16-QAM.

En la Figura 2.56 se muestran las constelaciones obtenidas de la ejecución del código anterior, es decir, el mapeo 16-QAM con y sin normalización.

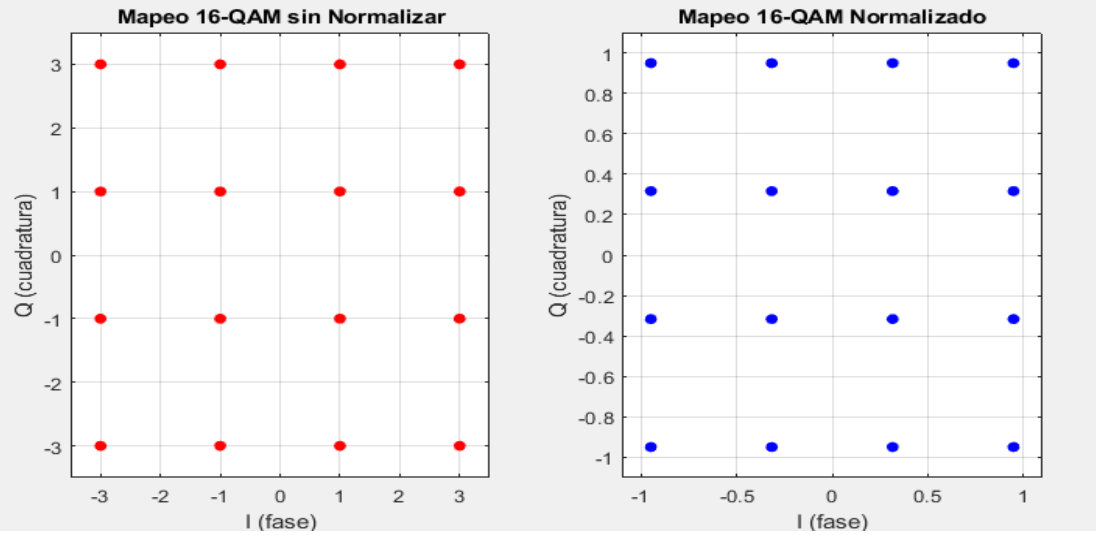


Figura 2.56. Diagrama de constelación de mapeo 16-QAM sin normalización y con normalización.

2.3.4.2.4. Mapeo 64-QAM en Matlab

Al igual que en el mapeo 16-QAM el mapeo 64-QAM se implementa con la función *qammod*. El ejemplo de modulación digital 64-QAM se implementa en una sección del script *mapeo.m*. El código inicia por definir el parámetro *M=64*, en base al cual se genera el vector con los números decimales de entrada. Luego se modulan los datos de entrada, especificando el orden de modulación *M*. Después se normaliza los datos modulados multiplicándolos por $1/\sqrt{42}$. Finalmente se grafica el diagrama de constelación de los datos modulados con y sin normalizar. El código descrito se presenta a continuación, debidamente comentado.

```
%% Mapeo 64-QAM
M = 64; % Puntos en la constelación (símbolos)
datos_C=(0:M-1).'; % vector de entrada
% Modulador:
% Factor de normalización = (1/sqrt(42))
datosM = qammod(datos_C,M); % Modulación 64-QAM (datos sin normalizar)
datosModulados = (1/sqrt(42))*datosM;% Símbolos modulados normalizados
%Graficar los puntos del diagrama de constelación:
subplot(1,2,1)
plot(datosM,'r.','MarkerSize',20);
title('Mapeo 64-QAM sin Normalizar')
xlabel('I (fase)');
ylabel('Q (cuadratura)');
grid on
axis([-7.6 7.6 -7.6 7.6])
subplot(1,2,2)
plot(datosModulados,'b.','MarkerSize',20);
title('Mapeo 64-QAM Normalizado')
xlabel('I (fase)');
ylabel('Q (cuadratura)');
axis([-1.15 1.15 -1.15 1.15])
grid on
```

Una vez ejecutada la sección de código anterior se obtienen las variables en el espacio de trabajo, mostradas en la Figura 2.57. En donde la primera variable contiene los símbolos decimales de 0 al 63, correspondientes a los datos de entrada del modulador. La segunda variable contiene los símbolos mapeados, la tercera variable corresponden a los símbolos mapeados normalizados.

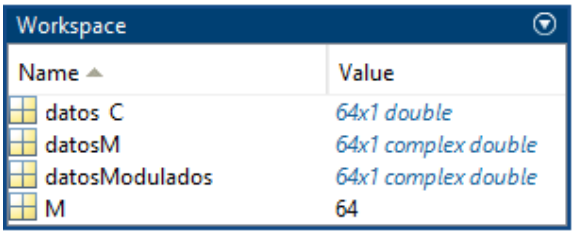


Figura 2.57. Variables luego de ejecutar el ejemplo de mapeo 64-QAM.

En la Figura 2.58 se muestran las constelaciones obtenidas de la ejecución del código anterior, es decir, el mapeo 64-QAM con y sin normalización.

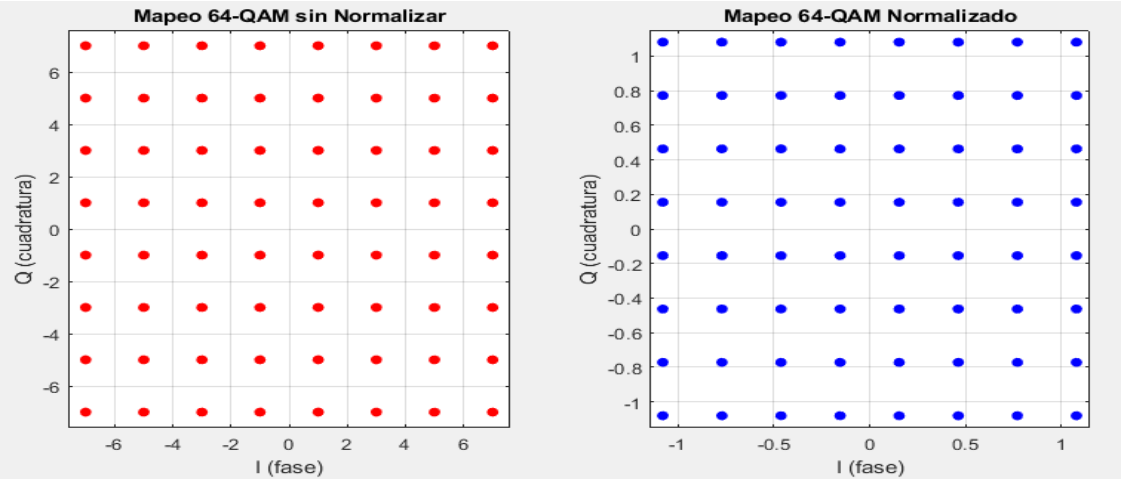


Figura 2.58. Diagrama de constelación de mapeo 64-QAM sin normalización y con normalización.

2.3.4.3. Mapeo aplicado al archivo de texto codificado

En la etapa Codificador Convolucional se estableció que para el ejemplo del procesamiento de archivo de texto a transmitir se va a utilizar mapeo QPSK. Entonces para desarrollar el ejemplo de mapeo de los bits codificados obtenidos en la etapa Codificador Convolucional, se utiliza únicamente la modulación digital QPSK, mediante funciones directas.

Sin embargo, cabe mencionar que en la interfaz gráfica de transmisión se implementan las técnicas de mapeo BPSK, QPSK, 16-QAM y 64-QAM.

Siguiendo el ejemplo de procesamiento de los datos del archivo de texto, en las etapas anteriores se implementó los scripts *abrir_archivo.m*, *serializar_datos.m* y *cod_conv.m*. Estos scripts corresponden a las etapas de Datos, Serializar y Codificador Convolucional respectivamente. A la salida de la etapa Codificador Convolucional se obtuvo un vector con

los bits codificados correspondientes al archivo de texto seleccionado en la etapa Datos. Los bits a la salida de la etapa Codificador Convolucional se convierten en los datos de entrada de la etapa Mapeo, estos bits están almacenados en el vector *datosBinario*.

Para borrar todas las variables generadas en scripts previos, a excepción del vector *datosBinario*, se utiliza el siguiente comando.

```
clearvars -except datosBinario % Borrar todas las variables excepto
% la variable datosBinario
```

La etapa actual se va a implementar en un script de nombre *mapeo_datos.m*, en donde se brinda la opción de seleccionar entre los mapeos BPSK, QPSK, 16-QAM y 64-QAM, mediante la variable *seleccionMapeo* y un condicional if. Si se desea seleccionar BPSK se asigna el valor de 10 a la variable *seleccionMapeo*, para QPSK se asigana el valor de 20, para 16-QAM se asigna el valor de 30 y para 64-QAM se asigna el valor de 40. El mapeo BPSK se lo implementa de manera manual, es decir sin utilizar funciones directas y los datos modulados se almacenan en el vector *datosMapeados*.

Para las técnicas de mapeo QPSK, 16-QAM y 64-QAM se inicia por definir el valor del número de símbolos en la constelación (*M*) y calcular el número de bits que se utilizan por símbolo (*Nb*). Luego se calcula la longitud de relleno, para asegurar que el vector de entrada sea múltiplo de *Nb*. Cuando ya se obtiene un vector con los bits de entrada relleno con ceros, se pasan los bits a símbolos decimales. Luego se modulan con funciones directas los símbolos decimales con codificación gray, además se aplica el factor de normalización (Tabla 2.11) a los datos modulados y se guardan los mismos en la variable de nombre *datosMapeados*.

Debido a que en el ejemplo de transmisión se utiliza **mapeo QPSK**, se asigna el valor de 20 a la variable *seleccionMapeo*. Entonces, los resultados que se obtienen aplican solo para QPSK. Si se desea cambiar el tipo de modulación será necesario cambiar el valor de *seleccionMapeo*. Esto aplica también para el script *cod_conv.m* de la etapa anterior (codificación convolucional).

El código de script *mapeo_datos.m* se presenta debidamente comentado a continuación.

```
% Script para mostrar un ejemplo para la etapa de mapeo
% Como argumento de entrada se tiene un vector fila tipo double,
% este vector se obtiene en el scritp cod_conv.m
% datosBinario es el vector de entrada
% El valor en la variable seleccionMapeo se define el tipo de
% mapeo a utilizar BPSK (10), QPSK (20), 16-QAM(30) o 64-QAM(40)
seleccionMapeo=20; % Se selecciona mapeo QPSK, ya que se trabaja con
% este para los ejemplos
```

```
% Con un lazo if se discrimina el mapeo a utilizar
if(seleccionMapeo==10)
    % Mapeo BPSK
    datosColumna=datosBinario.'; % Se transpone la matriz que contiene
    % los datos de entrada para obtener un vector columna
    % Se recorre cada bit de entrada y se asigna un símbolo
    for i=1:length(datosColumna)
        % Con if se discrimina si es igual a 1 y se asigna el
        % valor 0.707+0.707i, caso contrario es 0 y se asigna el
        % valor de -0.707-0.707i
        if(datosColumna(i)==1)
            datosMapeados(i,1)=complex(0.707,0.707);
        else
            datosMapeados(i,1)=complex(-0.707,-0.707);
        end
    end
    % Factor de normalización = 1
    disp('Mapeado con BPSK')
elseif(seleccionMapeo==20)
    % Mapeo QPSK
    M = 4; % Puntos en la constelación (símbolos)
    Nb=log2(M); % Bits por símbolo, se utiliza 2 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % Modulador:
    % Factor de normalización = 1/sqrt(2)
    datosMapeados = (1/sqrt(2))*pskmod(dataSimb,M,pi/4,'gray'); % Datos
    % mapeados con QPSK
    disp('Mapeado con QPSK')
elseif (seleccionMapeo==30)
    % Mapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    Nb=log2(M); % Bits por símbolo, se usan 4 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % Modulador:
    % Factor de normalización = 1/sqrt(10)
    datosMapeados = (1/sqrt(10))*qammod(dataSimb,M); %Modulación 16-QAM
    disp('Mapeado con 16-QAM')
elseif (seleccionMapeo==40)
    % Mapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos)
```

```
    Nb=log2(M); % Bits por símbolo, se usan 6 bits por símbolo
    % Hay que asegurarse que la longitud de los datos de entrada sea
    % múltiplo de Nb, por lo que se rellena de ceros cuando no es
    % múltiplo de Nb.
    % Se calcula la longitud de relleno y se almacena en long_padd0
    long_padd0=ceil(length(datosBinario)/Nb)*Nb-length(datosBinario);
    paddQ=zeros(1,long_padd0); % Vector de relleno con ceros
    datosbin=[datosBinario paddQ]; % Vector con relleno
    %Pasar de binario a símbolos decimales:
    dataMatriz=reshape(datosbin,Nb,[]).'; % Matriz de Nb columnas y
    % cada fila corresponde a un símbolo
    dataSimb=bi2de(dataMatriz,'left-msb'); % Símbolos a modular
    % Modulador:
    % Factor de normalización = 1/sqrt(42)
    datosMapeados = (1/sqrt(42))*(qammod(dataSimb,M)); % Datos
    % mapeados con 64-QAM
    disp('Mapeado con 64-QAM')
end
```

Una vez ejecutada la sección de código anterior, se obtiene en el espacio de trabajo las variables mostradas en la Figura 2.59. El vector fila *datosBinario* contienen a los bits de entrada a ser modulados, mientras que el vector columna *datosMapeados* contiene los datos mapeados con QPSK. En el caso del ejemplo que se está manejando no se requirió ningún relleno, por lo que *long_padd0* es igual a cero y además *paddQ* está vacío.

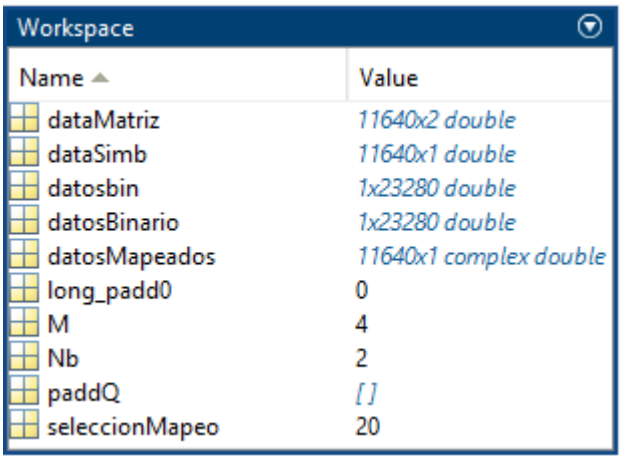


Figura 2.59. Variables en el espacio de trabajo luego de ejecutar *mapeo_datos.m*.

En la Figura 2.60 se comparan los primeros datos de; el vector de bits de entrada, la matriz con los bits agrupados en filas a ser convertidos en números decimales, el vector de símbolos decimales y el vector de datos mapeados.

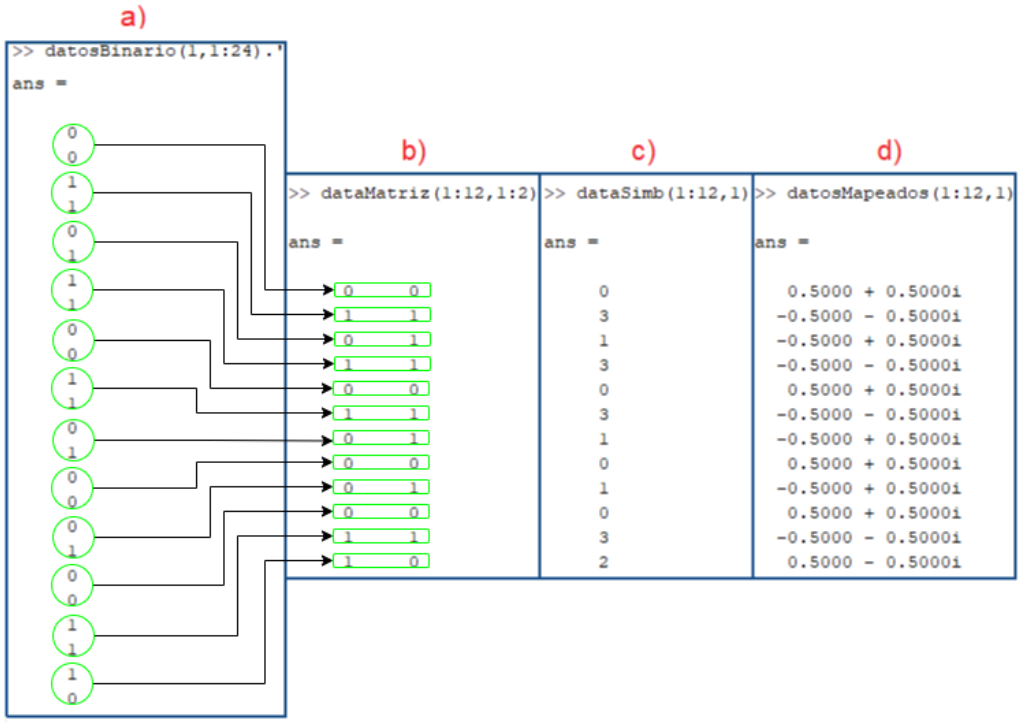


Figura 2.60. Primeros elementos de: a) Vector de bits de entrada. b) Matriz de bits agrupados. c) Vector de símbolos decimales. d) Vector de datos mapeados normalizados

2.3.5. ETAPA DE REFORMADO DE PULSO

Esta etapa se identifica en la Figura 2.61 con color anaranjado. El reformado de pulso consiste en aplicar un filtro a cada pulso de una señal o de una secuencia de datos, con el objetivo de contrarrestar la interferencia entre símbolos o también llamada ISI (Inter-Symbol Interference). El prototipo de comunicación inalámbrica que se desarrolló brinda al usuario la posibilidad de activar o desactivar la etapa de reformado de pulso.

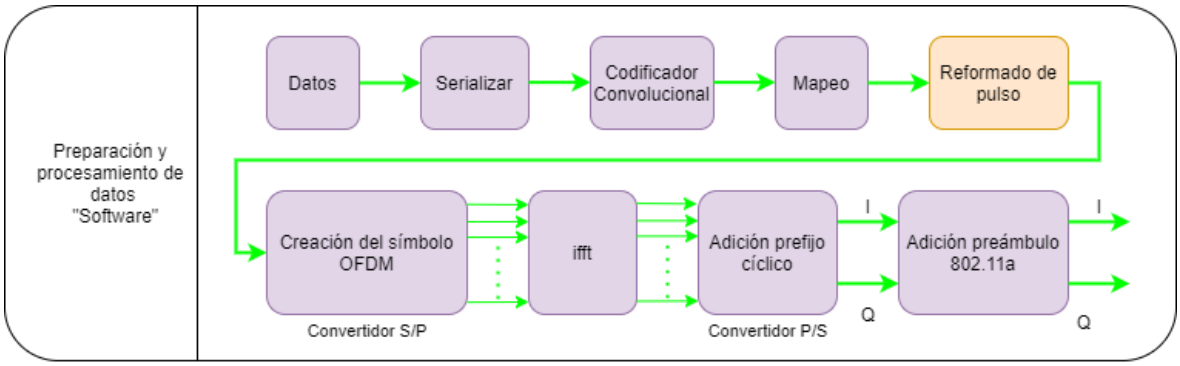


Figura 2.61. Etapas de transmisión con la etapa Reformado de pulso identificada.

En el prototipo, a la entrada de la etapa Reformado de pulso se puede tener los datos mapeados ya sea con BPSK, QPSK, 16-QAM o 64-QAM. A la salida de esta etapa se tienen los datos filtrados con un filtro raíz cuadrada de coseno elevado (SRRC).

Con el filtrado se obtiene una señal más suave, sin cambios bruscos, esto permite reducir la ISI.

El filtrado que se utiliza para implementar la técnica de reformado de pulso, se reparte entre transmisión y recepción. Tanto en transmisión como en recepción se utiliza un filtro SRRC. El resultado de utilizar dos filtros SRRC da como resultado en un filtro coseno elevado (RC). Antes de explicar la implementación del filtrado en Matlab, primero es necesario abordar los fundamentos teóricos relacionados al reformado de pulso.

2.3.5.1. Fundamentos teóricos

El reformado de pulso busca mitigar la interferencia entre símbolos (ISI), reduciendo el ancho de banda que ocupa una secuencia de datos, esta secuencia bien puede estar conformada por bits o por símbolos. La ISI aparece cuando se sobrepasa el ancho de banda del canal de comunicaciones, ya que se eliminan componentes de frecuencias de la señal, ensanchando cada símbolo en el dominio del tiempo [24]. Esto provoca que en el receptor se dificulte la detección de cada símbolo y por ende se producen errores. Cuando se utiliza la técnica de reformado de pulso se reduce la ISI y con esto también disminuye la tasa de errores en recepción [25].

Con el objetivo de entender a mayor detalle los conceptos relacionados con el reformado de pulso, en las siguientes secciones se analizan las principales características de los filtros sinc: coseno elevado del inglés raised-cosine (RC) y raíz cuadrada de coseno elevado del inglés square-root raised-cosine (SRRC).

2.3.5.1.1. Filtro sinc

El filtro sinc permite evitar la ISI ocupando la menor cantidad de ancho de banda posible [26]. Un **pulso sinc** en el dominio del tiempo se define en la ecuación 2.8, en donde t es el tiempo y T es el periodo de la señal.

$$p(t) = sinc\left(\frac{t}{T}\right) = \begin{cases} \dfrac{\sin\left(\frac{\pi t}{T}\right)}{\frac{\pi t}{T}}, t \neq 0 \\ 1 \quad , t = 0 \end{cases} \tag{2.8}$$

La respuesta en frecuencia del pulso sinc matemáticamente es un pulso rectangular (ecuación 2.9) donde f representa los valores de frecuencia.

$$P(f) = \begin{cases} T, & \frac{-1}{2T} \leq f \leq \frac{1}{2T} \\ 0, & otro\ caso \end{cases} \tag{2.9}$$

El pulso sinc tiene la característica de decaer lentamente, a una velocidad de $(1/|t|)$, lo que implica que las muestras muy lejanas pueden causar ISI cuando existen errores pequeños de sincronismo [26]. Otra característica del pulso sinc es que las transiciones por el eje del tiempo ocurren cada kT, en donde $k = 1,2,3,\ldots$, es decir k es un número entero diferente de cero.

En el dominio del tiempo un pulso sinc ideal tiene una duración infinita, por lo que para realizar una implementación práctica se debe realizar un truncamiento a una longitud finita de muestras. El término truncar se refiere a limitar el número de componentes, en este caso se trunca la señal en tiempo. El truncamiento del pulso produce lóbulos laterales en el dominio de la frecuencia, los cuales se relacionan con el fenómeno de Gibbs.

El pulso sinc es muy sensible a las fluctuaciones de tiempo en el receptor. Los lóbulos laterales y la sensibilidad a las fluctuaciones del tiempo pueden resolverse cuando la transición en el dominio de la frecuencia es más suave [26]. Para lograr transiciones más suaves en el dominio de la frecuencia se utiliza el filtro coseno elevado (RC).

En la Figura 2.62 se muestra un pulso sinc de periodo $T = 1\ segundo$ y su respuesta en el dominio de la frecuencia calculada computacionalmente. Para dicho pulso sinc teóricamente su respuesta en frecuencia sería igual a 1 (desde $-\frac{1}{2}$ hasta $\frac{1}{2}$) y para el resto de los valores igual a 0; sin embargo, esto no ocurre en la implementación práctica.

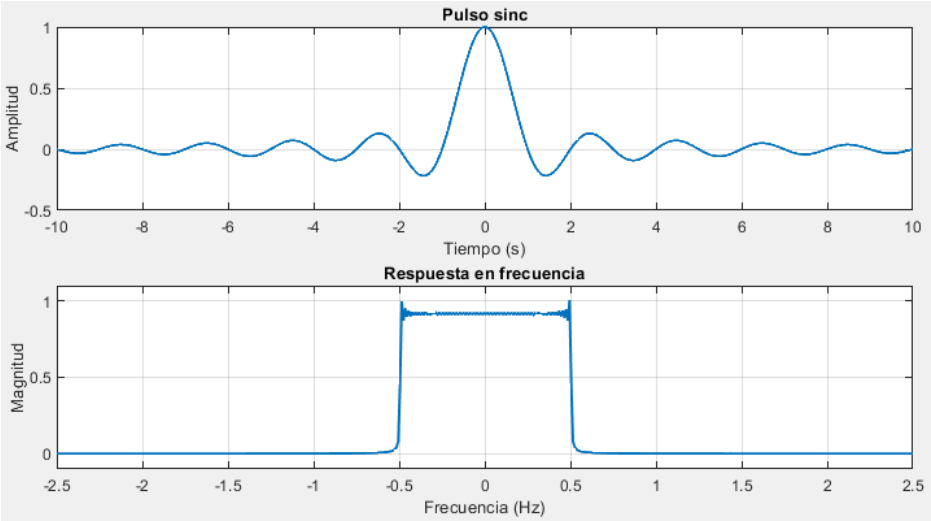


Figura 2.62. Pulso sinc de periodo $T = 1\ [s]$ y su respuesta en frecuencia.

La Figura 2.62 se obtiene mediante una sección de código en Matlab que grafica un pulso sinc en el dominio del tiempo y su respuesta en frecuencia. El pulso tiene un periodo $T =$

$1\ segundo$ y la señal se trunca en tiempo desde -50 hasta $50-(1/fs)$. La frecuencia de muestreo (fs), establece cuántas muestras tiene la señal en cada segundo y esta se configura en 200. El pulso sinc se calcula mediante una función de Matlab que contiene a la ecuación 2.8.

La respuesta en frecuencia se calcula con la función *fft* de Matlab, la cual calcula la DFT de la señal en tiempo para pasar la misma al dominio de la frecuencia. Luego se desplaza el componente de frecuencia cero al centro del espectro con la función *fftshift*. Después se obtienen los valores de magnitud normalizada de la respuesta en frecuencia del pulso sinc. Finalmente se grafica el pulso sinc y la respuesta en frecuencia, limitando los ejes para que se pueda observar adecuadamente las gráficas. El código comentado, descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Pulso sinc en tiempo y su respuesta en frecuencia
fs=200; % frecuencia de muestreo
ltr=50; % límite del truncado de la función sinc
t = -1*ltr:1/fs:ltr-1/fs; % tiempo
T=1; % Periodo
p_t=sinc(t./T); % señal en el dominio del tiempo
PF=fft(p_t); % Pasar al dominio de la frecuencia
PFshift=fftshift(PF); % Desplazar el componente de frecuencia cero
% al centro del espectro
PFgraf=(abs(PFshift)./max(abs(PFshift))).*T; % señal a graficar. Es la
% magnitud normalizada de la respuesta en frecuencia
f=fs/2*linspace(-1,1,2*ltr*fs); % Calcular los valores de frecuencia
% en Hz, para graficar la respuesta en frecuencia
% Graficar
figure()
subplot(2,1,1)
plot(t,p_t,'LineWidth',1.5) % señal en tiempo
grid on
title('Pulso sinc')
xlabel('Tiempo (s)')
ylabel('Amplitud')
xlim([-10 10]) % limitar el gráfico
subplot(2,1,2)
plot(f,PFgraf,'LineWidth',1.15) % Respuesta en frecuencia
grid on
title('Respuesta en frecuencia')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
xlim([-2.5 2.5]) % limitar el gráfico
ylim([-0.1 1.1])
```

2.3.5.1.2. Fenómeno de Gibbs

Cuando una señal tiene cambios repentinos en el dominio del tiempo, en el dominio de la frecuencia esa misma señal tiene componentes de frecuencias infinitos [27]. Una señal de onda cuadrada y una señal pulso son ejemplos de señales con cambios repentinos o

abruptos. Computacionalmente no se puede lograr señales con ancho de banda infinito, por lo que es necesario truncar la cantidad de componentes de frecuencia. El truncar una señal en frecuencia también implica realizar el truncamiento en tiempo. Realizar el truncado produce el llamado fenómeno de Gibbs. En la Figura 2.63 se identifica el fenómeno de Gibbs, también llamado ringing artifact.

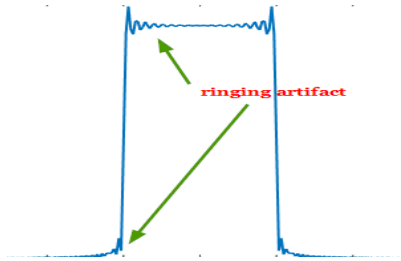


Figura 2.63. Fenómeno de Gibbs en una onda cuadrada cuando se trunca los componentes de frecuencia.

Los ringing artifact que resultan de una operación de filtrado están relacionados con las transiciones bruscas presentes en la forma de la respuesta al impulso del filtro [27].

2.3.5.1.3. Filtro coseno elevado o raised-cosine (RC)

El fenómeno de Gibbs, que se presenta en el filtro sinc, puede reducirse haciendo que las transiciones en el dominio de la frecuencia sean más suaves, mismas que pueden lograrse utilizando un filtro coseno elevado (RC). En el filtro RC existe un parámetro que permite ajustar la suavidad de la transición, esté parámetro se lo suele llamar factor de roll-off y se representa con el símbolo α. El parámetro α puede tomar valores de entre 0 a 1, siendo el valor de 0 cuando se tiene la transición más brusca y 1 cuando se tiene la transición más suave.

La ecuación 2.10 define la respuesta en frecuencia del filtro RC, los valores están parametrizados [28].

$$P(f) = \begin{cases} T, & |f| \leq \dfrac{1-\alpha}{2T} \\ \dfrac{T}{2}\left[1+\cos\left(\dfrac{\pi T}{\alpha}\left(|f| - \dfrac{1-\alpha}{2T}\right)\right)\right], & \dfrac{1-\alpha}{2T} \leq f \leq \dfrac{1+\alpha}{2T} \\ 0, & |f| \geq \dfrac{1+\alpha}{2T} \end{cases} \tag{2.10}$$

La ecuación 2.11 define la representación de un pulso RC en el dominio del tiempo [28]. Esta ecuación se puede obtener calculando la transformada inversa de Fourier de la ecuación 2.10 y luego evaluando las singularidades de la ecuación en el dominio del tiempo mediante la regla de L'Hospital. Además, T representa el periodo de la señal.

$$p(t) = \begin{cases} 1, & t = 0 \\ \frac{\alpha}{2}.\sin\left(\frac{\pi}{2\alpha}\right), & t = \pm\frac{T}{2\alpha} \\ \frac{\sin\left(\frac{\pi t}{T}\right)}{\frac{\pi t}{T}} . \frac{\cos\left(\frac{\pi\alpha t}{T}\right)}{1-\left(\frac{2\alpha t}{T}\right)^2}, & otro\ caso \end{cases} \quad (2.11)$$

Otra ventaja de un pulso RC sobre un pulso sinc es la mejora en la velocidad de decaimiento [28]. El pulso RC decae a una velocidad de $(1/|t|^3)$ y el segundo decae a una velocidad de $(1/|t|)$. Con esto se disminuye el riesgo de que las muestras muy lejanas causen ISI.

Para mostrar varios pulsos RC en el dominio del tiempo y en el dominio de la frecuencia se implementa una sección de código en Matlab, que inicia por definir una frecuencia de muestreo (fs) igual a 200. La frecuencia de muestreo establece cuantas muestras tiene la señal en cada segundo. El valor de 200 se utiliza para que las señales resultantes sean bastantes suaves. Luego se define el vector tiempo que va desde -50 hasta $50 - (1/fs)$ segundos, es decir que la señal está trucada en dicho valor. Después se define el perioto *T* de la señal en 1 segundo y se define el vector *alpha* que almacena los factores de roll-off 0, 0.3, 0.5, 0.7 y 1.

Mediante un lazo *for* se recorre desde 1 hasta la longitud del vector a*lpha*. En cada iteración del lazo, se calcula el pulso RC en el dominio del tiempo en base a la ecuación 2.11 y con el factor de roll-off en la posición correspondiente del vector *alpha*. En cada columna de la matriz *p_t* se almacena los pulsos en tiempo para cada valor de roll-off. Luego se calcula la respuesta en frecuencia para el pulso en tiempo, esto mediante la función *fft*, la cual calcula la DFT de la señal en tiempo para pasar la misma al dominio de la frecuencia. Se desplaza el componente de frecuencia cero al centro del espectro con la función *fftshift*. Después se obtienen los valores de magnitud normalizada de la respuesta en frecuencia del pulso RC y se almacena en una columna de la matriz *PFgraf*.

Al salir del lazo *for* se calcula los valores de frecuencia en Hz. Finalmente, se grafica los pulsos RC en tiempo y las respuestas en frecuencia, limitando los ejes de coordenadas para que se pueda observar adecuadamente las gráficas. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Pulsos RC y su respuesta en frecuencia
fs=200; % frecuencia de muestreo
ltr=50; % límite del truncado de la función sinc
t = -1*ltr:1/fs:ltr-1/fs; % tiempo
T=1; % Periodo
alpha=[0 0.3 0.5 0.7 1]; % Diferentes valores de roll off
```

```
p_t=[]; % Se inicializa la variable que contiene los pulsos RC en
% tiempo
PFgraf=[]; % Variable que contiene la magnitud normalizada de la
% respuesta en frecuencia
% Se recorre cada valor de roll-off
for j=1:length(alpha)
    % Se calcula cada señal RC en el dominio del tiempo, mediante
    % ecuaciones
    a=alpha(j);
    for i=1:length(t)
        if t(i)==0
            p(i)=1;
        elseif t(i)==T/(2*a) || t(i)==-T/(2*a)
            p(i)=(a/2)*sin(pi/(2*a));
        else
            p(i)=((sin(pi*t(i)/T)/(pi*t(i)/T))...
                *(cos(pi*a*t(i)/T)/(1-(2*a*t(i)/T)^2)));
        end
    end
    p_t(:,j)=p.'; % Se guarda un pulso RC con un valor de roll-off
    % diferente en cada columna
    PF=fft(p); % Pasar de tiempo a frecuencia
    PFshift = fftshift(PF); % Desplazar el componente de frecuencia
    % cero al centro del espectro
    PFgraf(:,j)=((abs(PFshift)./max(abs(PFshift))).*T).'; % Se guarda
    % en cada columna la magnitud normalizada de la respuesta en
    % frecuencia señal graficar. A cada columna le corresponde un
    % factor de roll off diferente
end
f = fs/2*linspace(-1,1,2*ltr*fs); % Calcular los valores de frecuencia
% en Hz
% Graficar los pulsos RC y la respuesta en frecuencia, para cada
% factor de roll off:
figure
subplot(2,1,1)
plot(t,p_t(:,1),t,p_t(:,2),t,p_t(:,3),...
    t,p_t(:,4),t,p_t(:,5),'LineWidth',1.15) % Pulsos RC
grid on
title('Pulsos coseno elevado RC')
xlabel('Tiempo (s)')
ylabel('Amplitud')
legend('alpha=0','alpha=0.3','alpha=0.5','alpha=0.7','alpha=1')
xlim([-5 5]) % Limitar los ejes
subplot(2,1,2)
plot(f,PFgraf(:,1),f,PFgraf(:,2),f,PFgraf(:,3),...
    f,PFgraf(:,4),f,PFgraf(:,5),'LineWidth',1.15) % Respuesta en freq.
grid on
title('Respuesta en frecuencia')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
legend('alpha=0','alpha=0.3','alpha=0.5','alpha=0.7','alpha=1')
xlim([-2.5 2.5]) % Limitar los ejes
ylim([-0.1 1.1]) % Limitar los ejes
```

El resultado de ejecutar a sección de código anterior es la Figura 2.64, en donde se muestran varios pulsos RC en el dominio del tiempo y las correspondientes respuestas en el dominio de la frecuencia. El periodo de todos los pulsos RC es igual a 1 segundo y se

utiliza factores de roll-off α de 0, 0.3, 0.5, 0.7 y 1. Cuando α es igual a 0, el filtro RC es igual al filtro sinc y por tanto la ventana del filtro va desde $-\frac{1}{2T}$ hasta $\frac{1}{2T}$. En el otro extremo si α es igual a 1 la ventana del filtro RC va desde $-\frac{1}{T}$ hasta $\frac{1}{T}$, el doble que cuando α es igual a 0. Entonces se puede decir que a medida que el factor de roll-off aumenta, el tamaño de la ventana también aumenta.

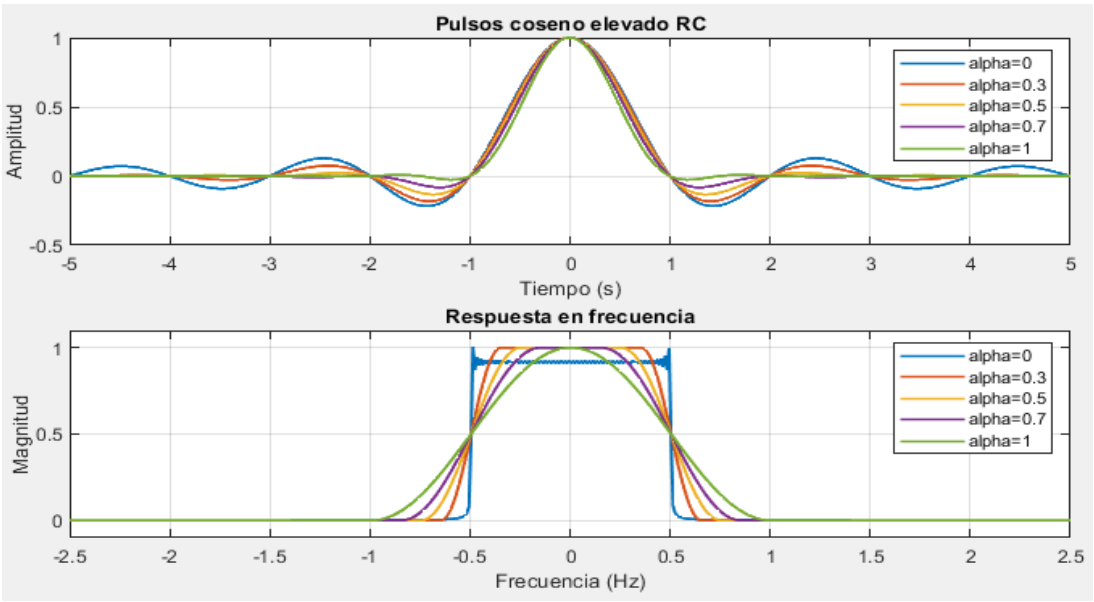


Figura 2.64. Pulsos coseno elevado (RC) con $T = 1\ [s]$, para valores de α de 0, 0.3, 0.5, 0.7 y 1, y su respuesta en frecuencia.

La técnica de reformado de pulso utiliza el filtrado RC, pero dividido entre el transmisor y el receptor.

En la Figura 2.65 se muestra una representación básica de un sistema de comunicaciones, en donde se ha dividido el filtrado RC en dos filtros, uno se ubica en el lado del transmisor y otro en el lado del receptor.

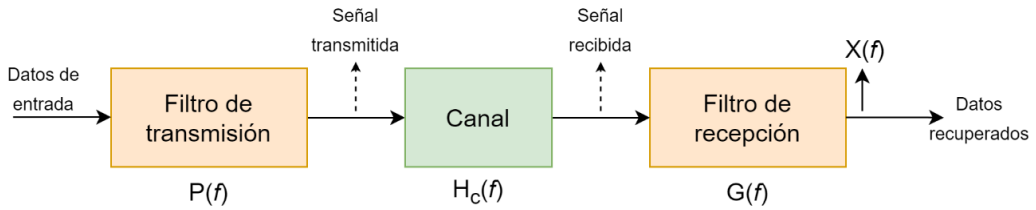


Figura 2.65. Representación básica de un filtro en transmisión, uno en recepción y el canal.

En la Figura 2.65, $P(f)$ es la respuesta en frecuencia del filtro de transmisión, $H_c(f)$ es la respuesta del canal, $G(f)$ es la respuesta en frecuencia del filtro de recepción y $X(f)$ es la respuesta en frecuencia del sistema. La relación entre las respuestas en frecuencia anteriores se muestra en la ecuación 2.12.

$$X(f) = P(f) * H_c(f) * G(f) \tag{2.12}$$

Si se asume que $H_c(f) = 1$, en el lado derecho de la ecuación quedan solamente las respuestas de los filtros de transmisión y recepción [29]. Como se mencionó anteriormente, el reformado de pulso utiliza el filtrado RC y por tanto $X(f)$ pasa a ser igual a la respuesta en frecuencia de un filtro RC. Además, $X(f)$ pasa a llamarse $X_{rc}(f)$. La ecuación 2.13 representa la relación entre un filtro RC y los filtros de transmisión y recepción.

$$X_{rc}(f) = P(f) * G(f) \tag{2.13}$$

Por lo general, en sistemas de comunicaciones se puede asumir que los filtros de transmisión y recepción son idénticos [4]. Por lo que, $P(f)$ es igual a $G(f)$ y la expresión anterior se convierte en:

$$X_{rc}(f) = P(f) * G(f) = |P(f)|^2 \tag{2.14}$$

Despejando, el filtro de transmisión y de recepción se puede representar con la siguiente expresión:

$$P(f) = G(f) = \sqrt{|X_{rc}(f)|} \tag{2.15}$$

Como $X_{rc}(f)$ es la respuesta en frecuencia de un filtro RC, la expresión de la ecuación 2.15 es la raíz cuadrada de la respuesta en frecuencia del filtro RC. Al dividir el filtrado RC entre transmisor y receptor, se obtiene un filtrado raíz cuadrada de coseno elevado (SRRC) en transmisor y otro en el receptor. En la siguiente sección se analiza en mayor detalle el filtro SRRC.

2.3.5.1.4. Filtro raíz cuadrada de coseno elevado (SRRC)

El reformado de pulso utiliza el filtrado coseno elevado (RC), el cual se obtiene utilizando dos filtros raíz cuadrada de coseno elevado (SRRC), uno en el transmisor y otro en el receptor, obteniendo una ISI mínima [30]. En el filtro SRRC el factor de roll-off se representa con el símbolo β y puede tomar valores entre 0 y 1. El impulso de SRRC se puede describir en tiempo, como se presenta en la ecuación 2.16 [29].

$$p(t) = \begin{cases} \frac{1}{\sqrt{T}}.\left[(1-\beta) + \frac{4\beta}{\pi}\right], & t = 0 \\ \frac{\beta}{\sqrt{2T}}.\left[\left(1+\frac{2}{\pi}\right) sin\left(\frac{\pi}{4\beta}\right) + \left(1-\frac{2}{\pi}\right) cos\left(\frac{\pi}{4\beta}\right)\right], & t = \pm\frac{T}{4\beta} \\ \frac{1}{\sqrt{T}}.\dfrac{\sin\left(\frac{\pi t(1-\beta)}{T}\right) + \frac{4\beta t}{T}\cos\left(\frac{\pi t(1+\beta)}{T}\right)}{\frac{\pi t}{T}\left(1-\left(\frac{4\beta t}{T}\right)^2\right)}, & otro\ caso \end{cases} \tag{2.16}$$

Para mostrar pulsos SRRC con diferentes valores de roll-off, en el dominio del tiempo y en el dominio de la frecuencia se implementa una sección de código en Matlab. La que inicia por definir una frecuencia de muestreo (fs) igual a 200. Luego se define el vector tiempo que va desde -50 hasta $50-(1/fs)$ segundos, es decir que la señal está trucada en dicho valor. Después se define el perioto *T* de la señal en 1 segundo y se define el vector *beta* que almacena los factores de roll-off 0, 0.3, 0.5, 0.7 y 1.

Mediante un lazo *for* se recorre desde 1 hasta la longitud del vector *beta*. En cada iteración del lazo, se calcula el pulso SRRC en el dominio del tiempo en base a la ecuación 2.16 y con el factor de roll-off en la posición correspondiente del vector *beta*. En cada columna de la matriz *p_t* se almacena, de acuerdo al valor de roll-off, el correspondiente pulso SRRC en tiempo. Luego se calcula la respuesta en frecuencia para el pulso en tiempo, esto mediante la función *fft* y la función *fftshift*. Después se obtienen los valores de magnitud normalizada de la respuesta en frecuencia del pulso SRRC y se almacena en una columna de la matriz *PFgraf*.

Al salir del lazo *for* se calcula los valores de frecuencia en Hz. Finalmente, se grafica los pulsos SRRC en tiempo y las respuestas en frecuencia, limitando los ejes de coordenadas para que se pueda observar adecuadamente las gráficas. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Pulsos SRRC y sus respuestas en frecuencia
fs=200; % frecuencia de muestro
ltr=50; % límite del truncado de la función sinc
t = -1*ltr:1/fs:ltr-1/fs; % tiempo
T=1; % Periodo
beta=[0 0.3 0.5 0.7 1]; % Diferentes valores de roll off
p_t=[]; % Se inicializa la variable que contiene los pulsos SRRC en
% tiempo
PFgraf=[];% Inicializar la variable que contiene la magnitud
% normalizada de la respuesta en frecuencia
% Se recorre cada valor de roll-off
for j=1:length(beta)
    % Se calcula cada pulso SRRC en el dominio del tiempo, mediante
    % ecuaciones
    B=beta(j);
    for i=1:length(t)
        if t(i)==0
            p(i)=(1/sqrt(T))*((1-B)+(4*B/pi));
        elseif t(i)==T/(4*B) || t(i)==-T/(4*B)
            p(i)=(B/sqrt(2*T))*((1+2/pi)*sin(pi/(4*B))+...
                (1-2/pi)*cos(pi/(4*B)));
        else
            p(i)=(1/sqrt(T))*((sin(pi*t(i)*(1-B)/T)+...
                (4*B*t(i)/T)*cos(pi*t(i)*(1+B)/T))/...
                ((pi*t(i)/T)*(1-(4*B*t(i)/T)^2)));
        end
    end
    p_t(:,j)=p; % Almacenar los valores de los pulsos SRRC
```

```
    % correspondientes a cada beta.
    PF=fft(p); % Pasar de tiempo a frecuencia
    PFshift = fftshift(PF); % Desplazar el componente de frecuencia
    % cero al centro del espectro
    PFgraf(:,j)=((abs(PFshift)./max(abs(PFshift))).*T).'; % Se guarda
    % en cada columna la magnitud normalizada de la respuesta en
    % frecuencia señal graficar. A cada columna le corresponde un
    % factor de roll off diferente
end

f = fs/2*linspace(-1,1,2*ltr*fs); % Calcular valores de frecuencia(Hz)
% Graficar los pulsos SRRC y la respuesta en frecuencia, para cada
% factor de roll off:
figure
subplot(2,1,1)
plot(t,p_t(:,1),t,p_t(:,2),t,p_t(:,3),...
    t,p_t(:,4),t,p_t(:,5),'LineWidth',1.15) % Pulsos SRRC
grid on
xlim([-5 5]) % Limitar los ejes
title('Pulsos raiz de coseno elevado SRRC')
xlabel('Tiempo (s)')
ylabel('Amplitud')
legend('beta=0','beta=0.3','beta=0.5','beta=0.7','beta=1')

subplot(2,1,2)
plot(f,PFgraf(:,1),f,PFgraf(:,2),f,PFgraf(:,3),...
    f,PFgraf(:,4),f,PFgraf(:,5),'LineWidth',1.15) % Resp. freq.
grid on
title('Respuesta en frecuencia')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
legend('beta=0','beta=0.3','beta=0.5','beta=0.7','beta=1')
xlim([-2.5 2.5]) % Limitar los ejes
ylim([-0.1 1.1])
```

El resultado de la sección de código previa es la Figura 2.66, en donde se muestra pulsos SRRC en tiempo y las correspondientes respuestas en frecuencia. Para diferentes factores de roll-off, pero en todos los casos el periodo $T = 1\ [s]$. En el filtro SRRC cuando β es igual a 0 la ventana del filtro va desde $-\frac{1}{2T}$ hasta $\frac{1}{2T}$. En el otro extremo si β es igual a 1 la ventana del filtro SRRC va desde $-\frac{1}{T}$ hasta $\frac{1}{T}$.

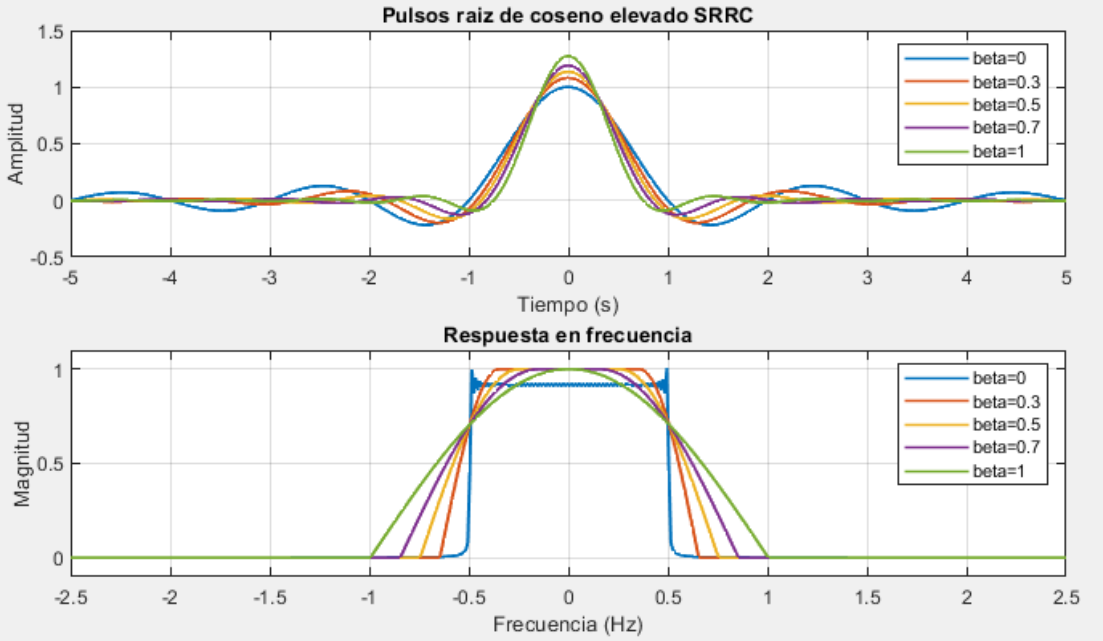


Figura 2.66. Pulsos raíz cuadrada de coseno elevado (SRRC) con $T = 1\ [s]$, para valores de β de 0, 0.3, 0.5, 0.7 y 1, y su respuesta en frecuencia.

Para demostrar que el producto de dos respuestas en frecuencia de pulsos SRRC es igual a la respuesta en frecuencia de un pulso RC, se implementa una sección de código en Matlab. En esta sección inicia por definir fs el tiempo y el periodo para los pulsos. Para implementar los pulsos SRRC se define los factores de β en el vector *beta*. Utilizando un lazo *for* se calculan los pulsos SRRC en tiempo para cada valor de *beta* mediante la ecuación 2.16 y su respuesta en frecuencia se calcula con las funciones *fft* y *fftshift*. Seguido se calcula el producto de dos respuestas en frecuencia del correspondiente pulso SRRC.

Luego para calcular los pulsos RC se define los valores de α en un vector llamado *Alpha*, tal que sean igual a los del vector *beta*. En un lazo *for* se calculan los pulsos RC mediante la ecuación 2.11, además de sus correspondientes respuestas en frecuencia. Finalmente, se grafica: las respuestas en frecuencia de los pulsos SRRC, el producto de la multiplicación en frecuencia de dos pulsos SRRC y las respuestas en frecuencia de los pulsos RC. El código descrito en estos párrafos se presenta a continuación, debidamente comentado.

```
%% Comparación de la respuesta en frecuencia de SRRC, la
% respuesta combinada de dos SRRC y la respuesta de RC.
fs=200; % frecuencia de muestro
ltr=50; % límite del truncado de la función sinc
t = -1*ltr:1/fs:ltr-1/fs; % tiempo
T=1; % Periodo
% SRRC
beta=[0 0.3 0.5 0.7 1];
p_t=[]; % Se inicializa la variable que contiene los pulsos SRRC en
```

```
% tiempo
PFgraf=[];% Inicializar la variable que contiene la magnitud
% Se recorre cada valor de roll-off
for j=1:length(beta)
    % Se calcula cada pulso SRRC en el dominio del tiempo, mediante
    % ecuaciones
    B=beta(j);
    for i=1:length(t)
        if t(i)==0
            p(i)=(1/sqrt(T))*((1-B)+(4*B/pi));
        elseif t(i)==T/(4*B) || t(i)==-T/(4*B)
            p(i)=(B/sqrt(2*T))*((1+2/pi)*sin(pi/(4*B))+...
                (1-2/pi)*cos(pi/(4*B)));
        else
            p(i)=(1/sqrt(T))*((sin(pi*t(i)*(1-B)/T)+...
                (4*B*t(i)/T)*cos(pi*t(i)*(1+B)/T))/...
                ((pi*t(i)/T)*(1-(4*B*t(i)/T)^2)));
        end
    end
    p_t(:,j)=p; % Almacenar los valores de tiempo para cada beta
    PF=fft(p); % pasar al dominio de la freq.
    PFshift = fftshift(PF); % Desplazar el componente de freq cero
    % al centro del espectro
    PFshiftComb = fftshift(PF).*fftshift(PF); %Esta expresión combina
    % la respuesta de dos filtros SRRC
    PFgraf(:,j)=((abs(PFshift)./max(abs(PFshift))).*T).'; % señal a
    % graficar, magnitud normalizada de la respuesta en frecuencia,
    % respuesta de 1 SRRC
    PFgrafComb(:,j)=((abs(PFshiftComb)./max(abs(PFshiftComb))).*T).';
    % señal a graficar, magnitud normalizada de la respuesta en
    % frecuencia, respuesta de 2 SRRC
end
% RC
alpha=[0 0.3 0.5 0.7 1];
p_trc=[]; % inicializar variable que contiene los pulsos SRRC en tiempo
PFgrafrc=[]; % inicializar variable que contiene las respuestas en freq
for j=1:length(alpha)
    % Se calcula cada pulso RC en el dominio del tiempo, mediante
    % ecuaciones
    a=alpha(j);
    for i=1:length(t)
        if t(i)==0
            prc(i)=1;
        elseif t(i)==T/(2*a) || t(i)==-T/(2*a)
            prc(i)=(a/2)*sin(pi/(2*a));
        else
            prc(i)=((sin(pi*t(i)/T)/(pi*t(i)/T))...
                *(cos(pi*a*t(i)/T)/(1-(2*a*t(i)/T)^2)));
        end
    end
    p_trc(:,j)=prc.'; % Pulsos RC para cada valor de alpha
    PFrc=fft(prc); % Pasar al dominio de la fecuencia
    PFshiftrc = fftshift(PFrc); % Desplazar el componente de frecuencia
    % cero al centro del espectro
    PFgrafrc(:,j)=((abs(PFshiftrc)./max(abs(PFshiftrc))).*T).'; % señal
    % a graficar (magnitud normalizada d ela resp en freq)
end
f = fs/2*linspace(-1,1,2*ltr*fs); % Calcular los valores de frecuencia
% en Hz
% Graficar las respuestas en frecuencia de un SRRC, 2 SRRC combinados
```

```
% y un RC
figure()
subplot(3,1,1) % Graficar SRRC
plot(f,PFgraf(:,1),f,PFgraf(:,2),f,PFgraf(:,3),...
    f,PFgraf(:,4),f,PFgraf(:,5),'LineWidth',1.15)
grid on
title('Respuesta en frecuencia pulso SRRC')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
legend('beta=0','beta=0.3','beta=0.5','beta=0.7','beta=1')
xlim([-2.5 2.5])
ylim([-0.1 1.1])
subplot(3,1,2) % Graficar 2 respuestas combinadas SRRC
plot(f,PFgrafComb(:,1),f,PFgrafComb(:,2),f,PFgrafComb(:,3),...
    f,PFgrafComb(:,4),f,PFgrafComb(:,5),'LineWidth',1.15)
grid on
title('Producto de dos respuestas en frecuencia de SRRC')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
legend('beta=0','beta=0.3','beta=0.5','beta=0.7','beta=1')
xlim([-2.5 2.5])
ylim([-0.1 1.1])
subplot(3,1,3) % Graficar RC
plot(f,PFgrafrc(:,1),f,PFgrafrc(:,2),f,PFgrafrc(:,3),...
    f,PFgrafrc(:,4),f,PFgrafrc(:,5),'LineWidth',1.15)
grid on
xlim([-2.5 2.5])
ylim([-0.1 1.1])
title('Respuesta en frecuencia de un pulso RC')
xlabel('Frecuencia (Hz)')
ylabel('Magnitud')
legend('alpha=0','alpha=0.3','alpha=0.5','alpha=0.7','alpha=1')
```

El resultado de la sección de código anterior se muestra en la Figura 2.67, en donde se utilizan varios factores de roll-off para mostrar gráficas de las respuestas en frecuencia de pulsos SRRC, del producto de dos respuestas en frecuencia de pulsos SRRC y de las respuestas en frecuencia de pulsos RC. Estas dos últimas gráficas son iguales, corroborando que utilizar dos filtros SRRC es igual a utilizar un solo filtro RC.

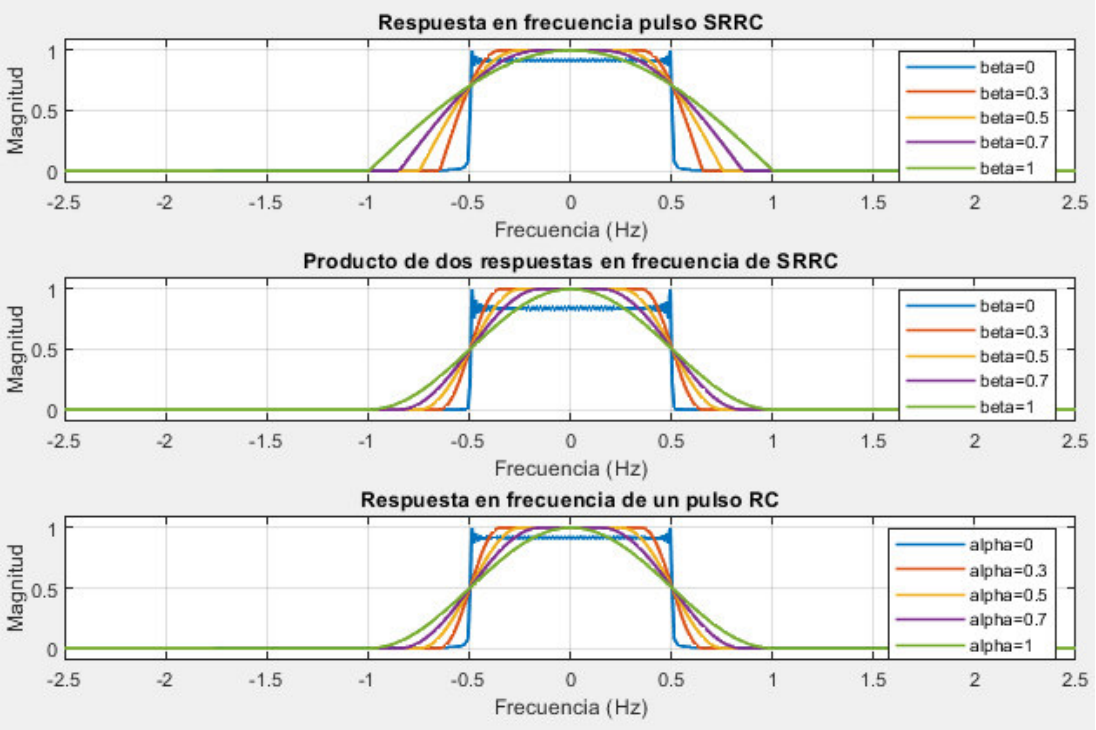


Figura 2.67. Gráficas de respuestas en frecuencia de pulsos SRRC, del producto de dos respuestas en frecuencia de pulsos SRRC y de las respuestas en frecuencia de pulsos RC y con $T = 1[s]$, para varios valores de β.

2.3.5.2. Implementación en Matlab

La implementación del reformado de pulso en Matlab se realiza dividiendo el filtrado entre el transmisor y el receptor. Tanto en el transmisor como en el receptor se deben usar filtros raíz cuadrada de coseno elevado (SRRC), para que, con la combinación de ambos, se obtenga un filtrado de coseno elevado (RC).

Los parámetros principales de un filtro RC o SRRC son: el número de muestras de salida por cada símbolo de entrada (*sps*), el tamaño en símbolos de la ventana *(span*) y el factor de roll-off, representado con α para el filtro RC y con β para SRRC. En la Figura 2.68 se muestra un pulso RC con un factor de roll-off de 0.5, un tamaño de la ventana igual a 6 símbolos y 8 muestras por cada símbolo. Cuando se aplique el filtro, el símbolo de entrada debe estar en el tiempo 0.

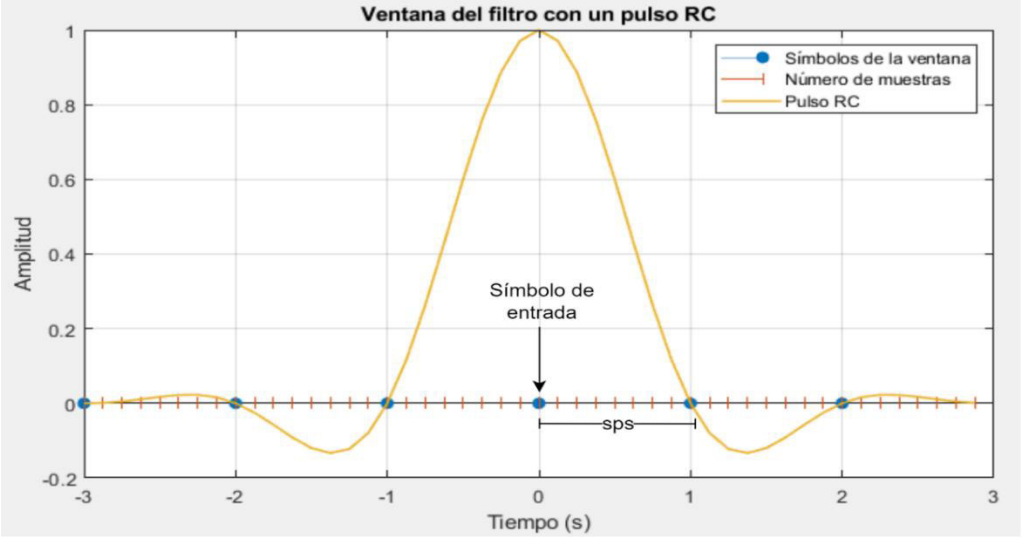


Figura 2.68. Pulso RC en tiempo, con *roll-off=0.5*, *sps=8* y *span=6*.

En un script llamado *reformado.m* se va a implementar varios ejemplos que tienen que ver con la técnica de reformado de pulso.

En la primera sección del script *reformado.m* se implementa un ejemplo de filtrado RC y SRRC para una secuencia de bits; 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1 y 1. El código inicia por definir los parámetros de $sps = 8$, $span = 6$, $\alpha = 0.5$ y $\beta = 0.5$. Luego se calcula el número de muestras totales en la ventana, multiplicando *sps* por *span*. Después se calcula el vector de tiempo para cada pulso RC y SRRC. Además, el periodo de las señales se establece en uno.

Luego, recorriendo cada muestra de tiempo y mediante la ecuación 2.11 se crea y almacena un pulso RC y con la ecuación 2.16 se implementa un pulso SRRC. Después se define el vector que contiene a los bits de entrada y se calcula el vector de tiempo para este vector. Antes de filtrar los datos con los pulsos RC y SRRC se inicializa con ceros los vectores que almacenan la señal filtrada, tal que todos los datos queden centrados como en la Figura 2.68. Para obtener la longitud de dichos vectores se suma la longitud del vector de entrada con el valor de *span* y a este resultado se multiplica por *sps*.

Para aplicar los filtros, hay que tener en cuenta que el dato que ingrese al filtro debe estar centrado en el pulso correspondiente, tal que el primer dato esté centrado en cero, el segundo en uno, el tercero en dos y así sucesivamente. Entonces con un lazo *for* se recorre las posiciones del vector de entrada. Tanto para el filtrado RC y SRRC, cada dato de entrada se multiplica por el pulso previamente creado. Este resultado se suma, en las posiciones adecuadas, a los valores que contiene el vector previamente inicializado con ceros. Una vez filtrado los datos se crea el vector de tiempo para dichos datos.

Finalmente se grafican los pulsos RC y SRRC, junto con los datos de *span* y de *sps*. También se grafica la señal de entrada junto con las señales filtradas con los filtros RC y SRRC. El código descrito en estos últimos párrafos se presenta a continuación, el mismo se encuentra debidamente comentado.

```
%% Comparar entre RC y SRRC con las fórmulas
sps=8; % Número de muestras de salida por cada símbolo de entrada
span=6; % Tamaño de la ventana (número de símbolos de la ventana)
B=0.5; % roll-off para SRRC
a=0.5; % roll-of para RC
ltr=span/2; % límite del truncado del pulso RC o SRRC
fs=sps*span; % Número de muestras en cada ventana
t = -1*ltr:1/sps:ltr-1/fs; % Vector de tiempo, para el pulso RC y SRRC
T=1; % Periodo, es preferible no cambiar este valor, si se requiere
% un periodo diferente es mejor dividir los resultados finales
% Pulso RC en el dominio del tiempo:
prc=[]; % iniciar vector donde se almacena el pulso RC
% Se recorre el vector tiempo y se aplica la ecuación, evaluando
también
% las indeterminaciones
for i=1:length(t)
    if t(i)==0
        prc(i)=1;
    elseif t(i)==T/(2*a) || t(i)==-T/(2*a)
        prc(i)=(a/2)*sin(pi/(2*a));
    else
        prc(i)=((sin(pi*t(i)/T)/(pi*t(i)/T))...
            *(cos(pi*a*t(i)/T)/(1-(2*a*t(i)/T)^2)));
    end
end
% Pulso SRRC en el dominio del tiempo:
p=[]; % iniciar vector donde se almacena el pulso SRRC
% Se recorre el vector tiempo y se aplica la ecuación, evaluando
% las indeterminaciones
for i=1:length(t)
    if t(i)==0
        p(i)=(1/sqrt(T))*((1-B)+(4*B/pi));
    elseif t(i)==T/(4*B) || t(i)==-T/(4*B)
        p(i)=(B/sqrt(2*T))*((1+2/pi)*sin(pi/(4*B))+...
            (1-2/pi)*cos(pi/(4*B)));
    else
        p(i)=(1/sqrt(T))*((sin(pi*t(i)*(1-B)/T)+...
            (4*B*t(i)/T)*cos(pi*t(i)*(1+B)/T))/...
            ((pi*t(i)/T)*(1-(4*B*t(i)/T)^2)));
    end
end
% Definir los datos de entrada y el vector tiempo d elos mismos
x=[1 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1]; % Vector de datos
tx=(0:length(x)-1); % Vector de tiempo para los datos x
yrc=zeros(1,(length(x)+span)*sps); % Iniciar var. que almacena el RC
y=zeros(1,(length(x)+span)*sps); % Iniciar var. que almacena el SRRC
% Aplicar el filtro RC y SRRC:
for i=1:length(x)
    % filtro RC:
    Auxrc=x(i)*prc; % Variable auxiliar, almacena la multiplicación del
    % dato de entrada actual por el pulso
    yrc(1,1+sps*(i-1):sps*(span+i-1))=...
        yrc(1,1+sps*(i-1):sps*(span+i-1))+Auxrc; % los datos se
```

```
    % almacenan y se suman a los anteriores
    % filtro SRRC:
    respAux=x(i)*p; % Variable auxiliar, almacena la multiplicación del
    % dato de entrada actual por el pulso
    y(1,1+sps*(i-1):sps*(span+i-1))=...
        y(1,1+sps*(i-1):sps*(span+i-1))+respAux; % los datos se
    % almacenan y se suman a los anteriores
end
ty=linspace(-ltr,length(x)+ltr-1/sps,length(y)); % tiempo para 'y'
% Graficar el pulso RC y SRRC, con las divisiones de span y sps
t_span=-ltr:ltr-1;
v_span=zeros(1,length(t_span));
v_SPS=zeros(1,length(t));
figure
stem(t_span,v_span,'filled')
hold on
stem(t,v_SPS,'-|')
plot(t,prc,'LineWidth',1.15) % Pulso RC
plot(t,p,'LineWidth',1.15) % Pulso SRRC
grid on
title('Ventana del filtro')
xlabel('Tiempo (s)')
ylabel('Amplitud')
legend('Símbolos de la ventana',...
    'Número de muestras','Pulso RC','Pulso SRRC')
% Graficar la señal de entrada y los datos filtrados con RC y SRRC:
figure()
stem(tx, x, 'filled') % señal de entrada
hold on
plot(ty, yrc, '-r','LineWidth',1.15) % RC
plot(ty, y, '-g','LineWidth',1.15) % SRRC
hold off
title('Filtrado con RC y SRRC')
xlabel('Tiempo')
ylabel('Amplitud')
legend('Datos de entrada','RC','SRRC')
```

Luego de ejecutar la sección de código anterior se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.69. Además, se obtienen las gráficas que se muestran en la Figura 2.70 y en la Figura 2.71.

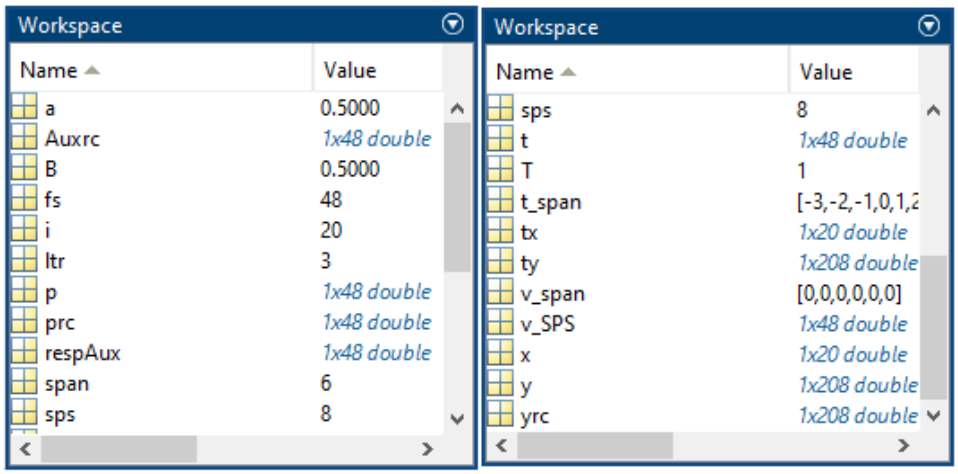


Figura 2.69. Variables en el espacio de trabajo de Matlab luego de ejecutar la primera sección del script *reformado.m*.

En la Figura 2.70 se muestran los pulsos RC y SRRC, utilizados para crear los filtros que llevan su mismo nombre. En el eje de las abscisas se ubica el número de símbolos (con color azul) y el número de muestras totales (con color rojo) que conforman los pulsos mencionados.

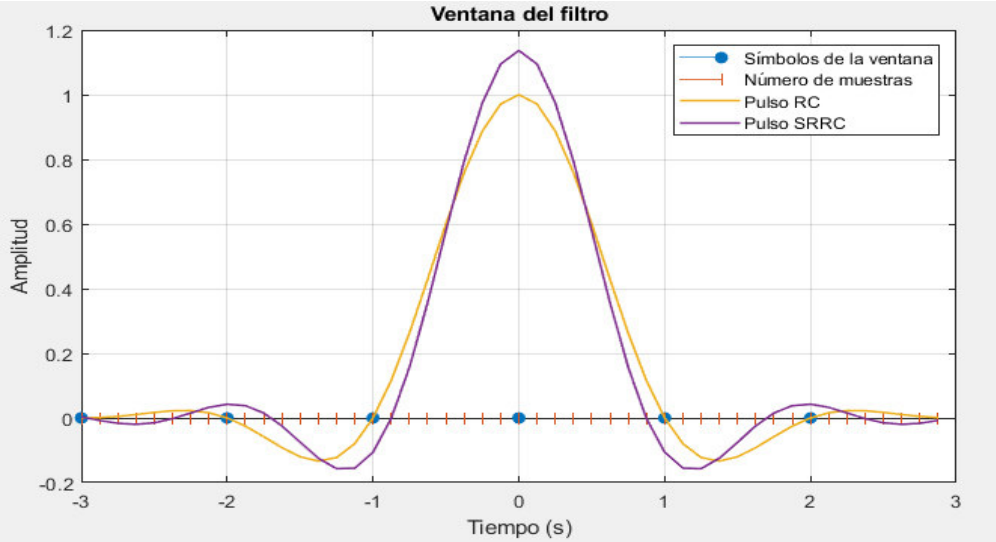


Figura 2.70. Ventana en tiempo de los pulsos RC y SRRC, con factor de *roll-off=0.5*, *sps=8* y *span=6*.

La Figura 2.71 muestra a la secuencia de bits de entrada (con color azul), la señal resultante de pasar los bits de entrada por con un filtro RC (con color rojo) y la señal filtrada con un filtro SRRC (con color verde).

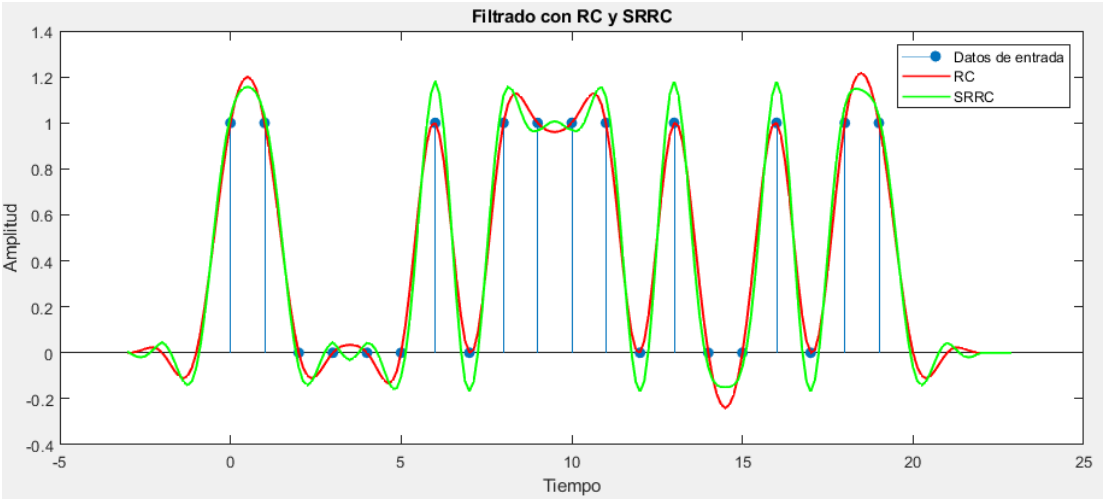


Figura 2.71. Secuencia de bits filtradas con filtros RC y SRRC, con factor de *roll-off=0.5*, *sps=8* y *span=6*.

Implementar un solo filtro, ya sea RC o SRRC, se consigue mediante las fórmulas planteadas en la parte teórica de esta etapa (Reformado de pulso). Sin embargo, el código de dichos filtros, resultan un poco largos y tediosos de implementar manualmente. En la parte de recepción, implementar el filtro SRRC se complicaría aún más, debido a que aparte de filtrar nuevamente la señal que se recibe, se debe recuperar los datos de entrada al filtro de transmisión.

Matlab cuenta con objetos de sistemas que permiten implementar el filtrado SRRC en transmisión y en recepción, resolviendo los problemas planteados anteriormente.

2.3.5.2.1. Funciones directas en Matlab

El filtro SRRC de transmisión se implementa con el objeto del sistema llamado *comm.RaisedCosine**Transmit**Filter* mientras que el filtro SRRC de recepción se implementa con el objeto *comm.RaisedCosine**Receive**Filter*. En esta sección se describe las características de los dos objetos del sistema, ya que ambos filtros son necesarios para cumplir con la parte teórica que plantea que dos filtros SRRC resultan en uno RC.

La sintaxis del objeto del sistema utilizado para implementar el filtro de transmisión es la siguiente:

txfilter = comm.RaisedCosineTransmitFilter(Name,Value)

En la sintaxis anterior se devuelve un objeto que contiene un filtro FIR (Finite Impulse Response) que puede tener la forma RC o SRRC y además tiene energía unitaria [31]. En la sintaxis del cuadro anterior, *txfilter* almacena el filtro creado y permite aplicarlo a una secuencia de datos. Para cada propiedad *Name* se debe especificarse un valor *Value* entre

comillas. *Name* representa el nombre de una propiedad de entrada y *Value* el valor que se configura de dicha propiedad.

Las propiedades de entrada del filtro se describen a continuación [31]:

- **Shape:** Es la forma del filtro, se puede especificar como 'Normal' o como 'Square root', que corresponden a un filtro RC y a uno SRRC respectivamente. El valor por defecto es 'Square root'.
- **RolloffFactor:** Es el factor roll-off o factor de caída, identificado anteriormente con α para un filtro RC o con β para un filtro SRRC. El valor que puede tomar esta propiedad es un número escalar que va desde 0 hasta 1, el valor por defecto es 0.2. Además, esta propiedad especifica indirectamente el ancho de banda del filtro.
- **FilterSpanInSymbols:** indica el intervalo del filtro en símbolos, es decir el truncado del tamaño de la ventana especificado por un número de símbolos. En las imlementaciones anteriores a este parámetro se lo llama *span*. La propiedad debe tomar el valor de un número entero positivo y su valor por defecto es 10.
- **OutputSamplesPerSymbol:** indica las muestras de salida que se va a tener por cada símbolo de entrada. Se especifica como un número entero positivo y el valor por defecto es de 8.
- **Gain:** es la ganancia del filtro y se especifica como un número escalar positivo. El valor por defecto de esta propiedad es 1.

Una vez creado el filtro *txfilter*, se puede aplicar un filtro FIR RC o SRRC utilizando la siguiente sintaxis:

datosFiltrados = txfilter (datosEnt)

El argumento *datosEnt* corresponde a un vector columna que contiene los datos de entrada a ser filtrados. El argumento de salida *datosFiltrados* devuelve un vector columna de longitud igual a la longitud del vector de entrada multiplicada por *OutputSamplesPerSymbol*.

La sintaxis del objeto del sistema utilizado para implementar el filtro de recepción es la siguiente:

rxfilter = comm.RaisedCosineReceiveFilter(Name,Value)

El objeto del sistema *comm.RaisedCosineReceiveFilter* diezma la señal de entrada, utilizando un filtro FIR que puede tener la forma RC o SRRC [32]. En la sintaxis del cuadro anterior, *Name* representa el nombre de la propiedad que se define con *Value*, *rxfilter*

almacena el filtro de recepción creado en base a lo que se define con las propiedades de entrada *Name* y *Value*.

Las propiedades de entrada del filtro se describen a continuación[32]:

- **Shape:** Es la forma del filtro, se puede especificar como 'Square root' o como 'Normal', que corresponden a un filtro SRRC y a uno RC respectivamente. El valor por defecto es 'Square root'.
- **RolloffFactor:** Es el factor roll-off o factor de caída, identificado anteriormente con α para un filtro RC y con β para un filtro SRRC. El valor que puede tomar esta propiedad es un número escalar que va desde 0 hasta 1 y el valor por defecto es 0.2.
- **FilterSpanInSymbols:** Es el intervalo del filtro en símbolos, es decir el truncado del tamaño de la ventana especificado por un número de símbolos. La propiedad debe tomar el valor de un número entero positivo y su valor por defecto es 10.
- **InputSamplesPerSymbol:** Es el número de muestras de entrada por cada símbolo. Se especifica como un número entero positivo y el valor por defecto es de 8.
- **DecimationFactor:** Es el factor de diezmado, que se especifica con un número escalar positivo, que puede tomar valores entre 1 e *InputSamplesPerSymbol*. El valor que se configure en esta propiedad divide al valor de *InputSamplesPerSymbol*. El valor por defecto es de 8.
- **DecimationOffset:** Desplazamiento de diezmado, se especifica como un número entero en el rango de 0 hasta (*DecimationFactor*- 1) y el valor por defecto es 0. Esta propiedad especifica el número de muestras filtradas que el objeto descarta antes de reducir la resolución.
- **Gain:** Es la ganancia del filtro y se especifica como un número escalar positivo. El valor por defecto de esta propiedad es 1.

Una vez creado el filtro *rxfilter*, se puede aplicar el mismo, tal que se diezme la señal de entrada utilizando la siguiente sintaxis:

datosSal = rxfilter (datoFiltrados)

El argumento *datosFiltrados* corresponde a un vector columna que contiene los datos de entrada que previamente se han filtrado con el objeto del sistema *comm.RaisedCosineTransmitFilter*. El argumento de salida *datosSal* devuelve un vector columna que contiene la señal diezmada mediante la propiedad *DecimationFactor*.

Como se mencionó anteriormente, el script llamado *reformado.m* se utiliza para implementar varios **ejemplos** que tienen que ver con la técnica de reformado de pulso. En la primera sección se implementó un ejemplo de filtrado RC y SRRC, mediante las ecuaciones descritas en la parte teórica.

En la sección 2 del script *reformado.m* se crea un ejemplo en donde se compara el filtrado RC y SRRC mediante fórmulas, con el filtrado RC y SRRC implementado mediante el objeto de transmisión *comm.RaisedCosineTransmitFilter*.

La sección 2 del script *reformado.m* inicia por limpiar las variables del espacio de trabajo de Matlab y cerrar las figuras abiertas. Luego se define la secuencia de bits de entrada a ser filtrados, esta secuencia es; 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1 y 1. Después se calcula el vector tiempo para el vector de entrada. También se definen los parámetros generales a utilizarse en los filtros, como son; el número de muestras por símbolo (*sps*), el tamaño de la ventana (*span*) y el factor de roll-off. Seguido se implementan los filtros mediante las fórmulas de la parte teórica, anteriormente ya se describió esta parte del código, por lo que no es necesario describirlo nuevamente.

Luego se implementa el filtrado RC y SRRC a través de funciones directas de Matlab. Los filtros FIR se crean mediante el objeto del sistema llamado *comm.RaisedCosineTransmitFilter*, la propiedad *Shape* toma el valor de 'Normal' para el filtro RC y el valor de 'Square root' para el filtro SRRC. Tanto para el filtro RC como para el filtro SRRC la propiedad *RolloffFactor* toma el valor definido en la variable *rolloff*, la propiedad *FilterSpanInSymbols* toma el valor seteado en *span* y la propiedad *OutputSamplesPerSymbol* toma el valor guardado dentro de la variable *sps*.

Cuando se implementa el reformado de pulso mediante los objetos del sistema de transmisión y recepción, es necesario realizar un relleno de ceros, para que en recepción se puedan recuperar todos los bits de información que ingresaron al filtro.

Si se utiliza los objetos del sistema para implementar el filtrado, los datos de entrada se pueden recuperar en recepción cuando la longitud del vector de relleno es igual a *span*. Entonces en el código, el vector de entrada se lo transforma en un vector tipo columna y se agrega el relleno de ceros de longitud igual a *span*. Después se calcula el vector de tiempo para los datos con relleno.

Luego se filtra los datos con los filtros RC y SRRC creados previamente. También se calcula el vector del tiempo para los datos filtrados. Se debe realizar un ajuste del tiempo, es decir retrasar la señal filtrada un tiempo igual a *span/2*. Después se grafican los datos

de entrada junto con la señal filtrada con el filtro RC implementado con funciones directas. Esto se hace con el objetivo de mostrar la señal desfasada y los datos de relleno.

Para comprobar que el resultado es el mismo se grafican las señales obtenidas mediante el filtrado por medio de ecuaciones y el filtrado por funciones directas. También se normalizan y se grafican señales filtradas por funciones directas para que coincidan con las señales obtenidas por ecuaciones, esto con el objetivo de demostrar que las funciones directas funcionan adecuadamente y sirven para implementar el reformado de pulso.

El código de la sección 2 del script *reformado.m* y el cual se ha descrito en estos últimos párrafos se presenta a continuación. Dicha sección de código se encuentra comentada adecuadamente para una mejor comprensión.

```
%% Sección 2: Comparar filtros RC y SRRC con las fórmulas y con
% funciones directas
clear ; close all;
x=[1 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1]; % Datos de entrada
tx=(0:length(x)-1); % Vector de tiempo para los datos x
% Parámetros generales de lso filtros:
sps=8; % Número de muestras de salida por cada símbolo de entrada
span=6; % Tamaño de la ventana (número de símbolos de la ventana)
rolloff= 0.5; % Factor de caída del filtro (roll off)
%************** Filtrado RC y SRRC mediante las fórmulas **************
B=rolloff; % roll-off para SRRC
a=rolloff; % roll-of para RC
ltr=span/2; % límite del truncado del pulso RC o SRRC
fs=sps*span; % Número de muestras en cada ventana
t = -1*ltr:1/sps:ltr-1/fs; % Vector de tiempo, para el pulso RC y SRRC
T=1; % Periodo, es preferible no cambiar este valor
% Pulso RC en el dominio del tiempo (con ecuaciones):
prc=[]; % iniciar vector donde se almacena el pulso RC
for i=1:length(t)
    if t(i)==0
        prc(i)=1;
    elseif t(i)==T/(2*a) || t(i)==-T/(2*a)
        prc(i)=(a/2)*sin(pi/(2*a));
    else
        prc(i)=((sin(pi*t(i)/T)/(pi*t(i)/T))...
            *(cos(pi*a*t(i)/T)/(1-(2*a*t(i)/T)^2)));
    end
end
% Pulso SRRC en el dominio del tiempo (con ecuaciones):
p=[]; % iniciar vector donde se almacena el pulso SRRC
for i=1:length(t)
    if t(i)==0
        p(i)=(1/sqrt(T))*((1-B)+(4*B/pi));
    elseif t(i)==T/(4*B) || t(i)==-T/(4*B)
        p(i)=(B/sqrt(2*T))*((1+2/pi)*sin(pi/(4*B))+...
            (1-2/pi)*cos(pi/(4*B)));
    else
        p(i)=(1/sqrt(T))*((sin(pi*t(i)*(1-B)/T)+...
            (4*B*t(i)/T)*cos(pi*t(i)*(1+B)/T))/...
            ((pi*t(i)/T)*(1-(4*B*t(i)/T)^2)));
    end
```

```
end
yrc=zeros(1,(length(x)+span)*sps); % Iniciar var. que almacena el RC
y=zeros(1,(length(x)+span)*sps); % Iniciar var. que almacena el SRRC
% Aplicar el filtro RC y SRRC:
for i=1:length(x)
    % filtro RC:
    Auxrc=x(i)*prc; % Multiplica el dato de entrada por el pulso RC
    yrc(1,1+sps*(i-1):sps*(span+i-1))=...
        yrc(1,1+sps*(i-1):sps*(span+i-1))+Auxrc; % señal filtrada
    % filtro SRRC:
    respAux=x(i)*p; % Multiplica el dato de entrada por el pulso SRRC
    y(1,1+sps*(i-1):sps*(span+i-1))=...
        y(1,1+sps*(i-1):sps*(span+i-1))+respAux; % señal filtrada
end
ty=linspace(-ltr,length(x)+ltr-1/sps,length(y)); % tiempo para 'y'
%*********** Filtrado RC y SRRC por funciones directas ***************
% Creación del filtro RC:
txfilterRC = comm.RaisedCosineTransmitFilter(...
    'Shape','Normal','RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span,'OutputSamplesPerSymbol',sps);
% Creación del filtro SRRC:
txfilter = comm.RaisedCosineTransmitFilter(...
    'Shape','Square root','RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span,'OutputSamplesPerSymbol',sps);
xr=[x.';zeros(span,1)]; % Crear un vector con relleno de ceros,
% para que obtener el tamaño de la ventana completo y recuperar
% todos los datos de entrada en recepción.
txr =(0:length(xr)-1); % Vector de tiempo para los datos xr
% Filtrado:
yf0RC = txfilterRC(xr); % filtrar los datos con el filtro RC
yf0 = txfilter(xr); % filtrar los datos con el filtro SRRC
tf0 = (0:(length(x)+span)*sps-1)/sps; % vector de tiempo de datos filt
% Corregir delay:
t = tf0-(span/2); % se retrasa el tiempo para centrar la señal filtra-
% da con los datos de entrada.
%****** Graficar los datos de entrada y las señales filtradas *******
% Graficar los datos de entrada y la señal sin corregir en fase:
figure()
stem(txr(1:20),x,'filled') % datos de entrada
hold on
stem(txr(21:26),xr(21:26),'filled')
plot(tf0, yf0RC,'LineWidth',1.2) % datos filtrados con RC sin
% corregir en fase
hold off
axis([-1 30 -0.3 1.3])
title('Señal sin corregir la fase')
xlabel('Tiempo')
ylabel('Amplitud')
legend('Datos de entrada','Relleno de ceros','Señal filtrada')
% Graficar los datos de entrada y las señales filtradas:
figure()
stem(tx, x, 'filled') % señal de entrada
hold on
plot(ty, yrc,'LineWidth',1.15) % RC
plot(ty, y,'LineWidth',1.15) % SRRC
plot(t, yf0RC,'LineWidth',1.15) % RC
plot(t, yf0,'LineWidth',1.15) % SRRC
hold off
title('Filtrado con RC y SRRC')
xlabel('Tiempo')
```

```
ylabel('Amplitud')
legend('Datos de entrada','RC fórmula','SRRC fórmula',...
    'RC fnc directa','SRRC fnc directa')
figure()
stem(tx, x, 'filled') % señal de entrada
hold on
plot(ty, yrc,'LineWidth',1.15) % RC
plot(ty, y,'LineWidth',1.15) % SRRC
plot(t, yf0RC*(1/max(coeffs(txfilterRC).Numerator)),'LineWidth',1.15)
plot(t, yf0*(sqrt(sps)),'LineWidth',1.15) % SRRC
hold off
title('Filtrado con RC y SRRC (señales normalizadas)')
xlabel('Tiempo')
ylabel('Amplitud')
legend('Datos de entrada','RC fórmula','SRRC fórmula',...
    'RC fnc directa','SRRC fnc directa')
```

Una vez ejecutada la sección de código anterior se obtienen las variables en el espacio de trabajo de Matlab que se muestran en la Figura 2.72. También se obtienen las Gráficas de la Figura 2.73, Figura 2.74 y Figura 2.75.

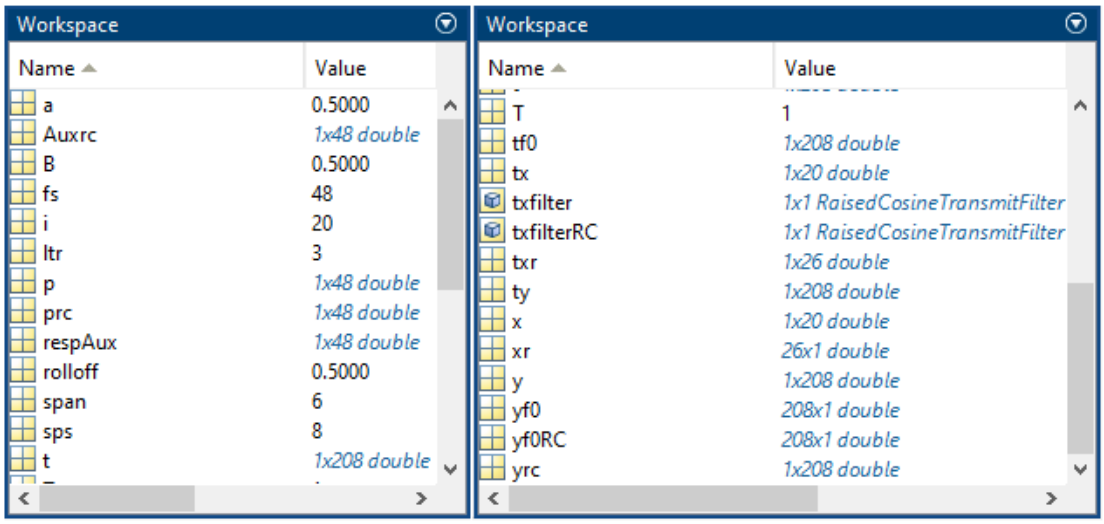


Figura 2.72. Variables en el espacio de trabajo luego de ejecutar la sección 2 del script *reformado.m*.

En la Figura 2.73 se muestra los datos originales de entrada con color azul y el relleno de ceros se identifica con color rojo. En color amarillo se sobrepone la señal filtrada utilizando el filtro RC creado con el objeto del sistema de transmisión. Dicha señal no está en fase con los datos de entrada, por lo cual se debe realizar una corrección de fase en el eje del tiempo. La corrección de fase se consigue restando *span/2* al vector tiempo.

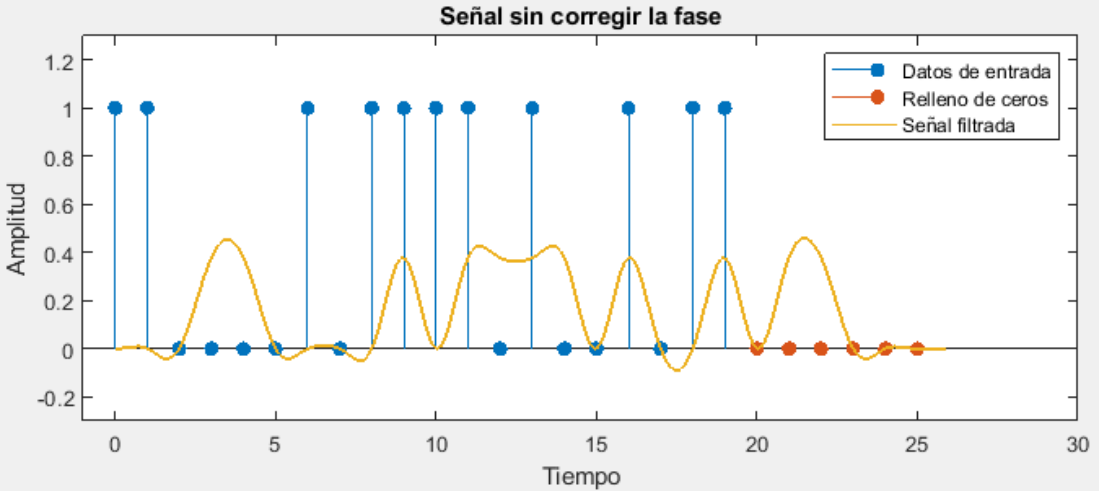


Figura 2.73. Datos de entrada con relleno de ceros y señal filtrada con RC mediante función directa.

En la Figura 2.74 se grafica en el dominio del tiempo lo siguiente:

- Con color azul los datos de entrada.
- Con color rojo la señal del filtrado RC mediante ecuaciones.
- Con color amarillo la señal del filtrado SRRC mediante ecuaciones.
- Con color morado la señal del filtrado RC mediante el objeto del sistema *comm.RaisedCosineTransmitFilter*.
- Con color verde la señal del filtrado SRRC mediante el objeto del sistema *comm.RaisedCosineTransmitFilter*.

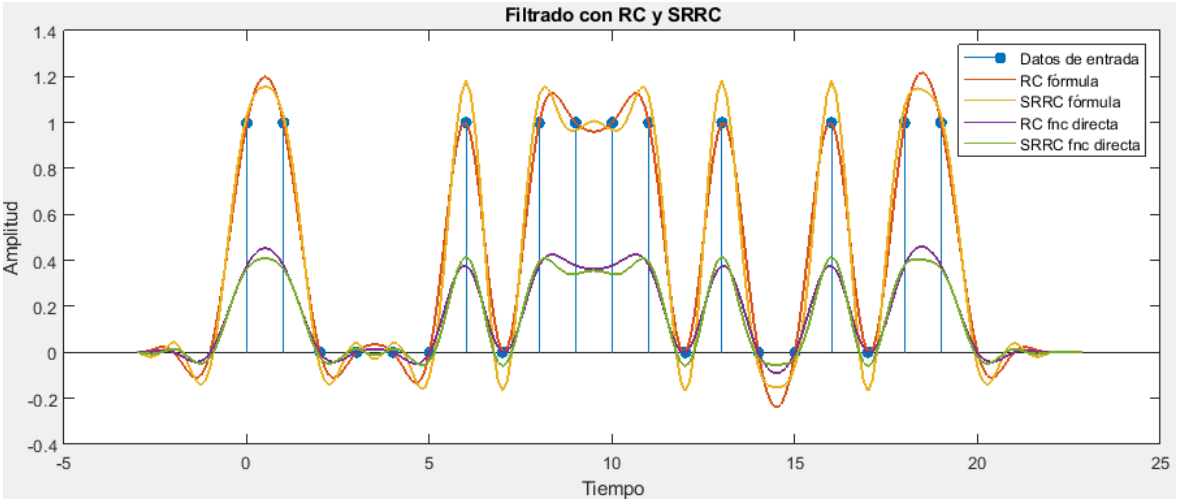


Figura 2.74. Comparación del filtrado RC y SRRC mediante ecuaciones y función directa de Matlab.

Las señales obtenidas mediante ecuaciones son similares a las señales obtenidas mediante la función directa. Estas últimas requieren de un factor de normalización para que coincidan exactamente con las señales obtenidas con ecuaciones. La señal filtrada con RC se normaliza multiplicando por un número escalar el cual se obtiene con la expresión;

(*1/max(coeffs(txfilterRC).Numerator)*, esta expresión significa que se divide uno para el máximo valor del campo *Numerator* de una estructura que calcula los coeficientes del polinomio del filtro RC. Para normalizar la señal filtrada con SRRC se multiplica la misma por $\sqrt{sps}$.

En la Figura 2.76 se grafican las mismas señales que en la Figura 2.75, con la diferencia que las señales obtenidas mediante el objeto del sistema *comm.RaisedCosineTransmitFilter* estaán normalizadas. En la gráfica se observa que las señales normalizadas se sobreponen perfectamente a las señales obtenidas por medio de ecuaciones.

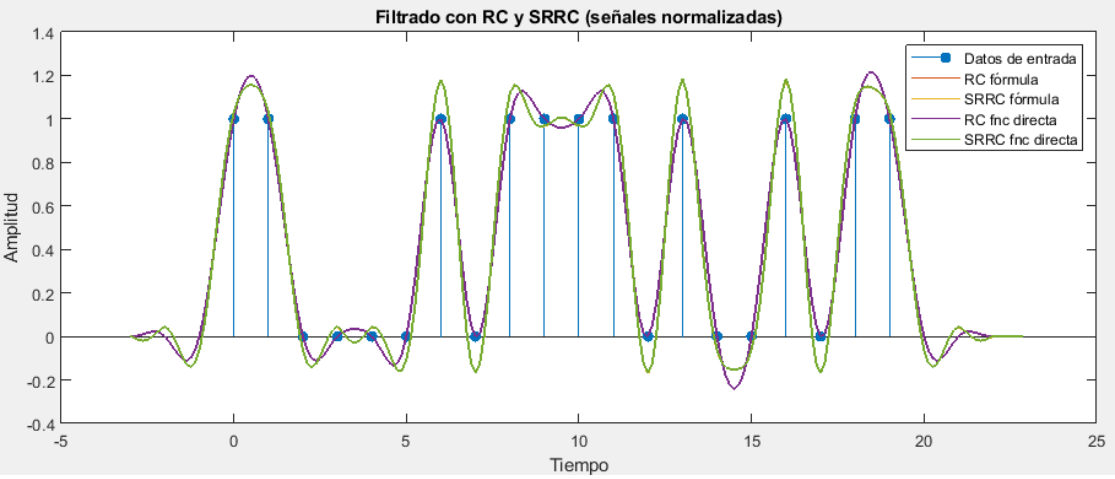


Figura 2.76. Comparación del filtrado RC y SRRC mediante ecuaciones y función directa. Las señales obtenidas con la función directa están normalizadas.

Para demostrar que en la práctica la señal filtrada con dos filtros SRRC resulta igual a utilizar un filtro RC, se implementa una tercera sección de código en el script *reformado.m*.

En la sección 3 del script *reformado.m* se implementa un filtro SRRC de transmisión con el objeto del sistema *comm.RaisedCosineTransmitFilter* y un filtro SRRC de recepción con el objeto del sistema *comm.RaisedCosineReceiveFilter*. Además, se implementa un filtro RC normalizado para comparar la señal que devuelve dicho filtro, con la señal que resulta luego de utilizar dos filtros SRRC. La sección de código inicia por limpiar las variables del espacio de trabajo de Matlab y cerrar las figuras abiertas. Luego se definen los parámetros generales a utilizarse en los filtros, como son; el número de muestras por símbolo (*sps*), el tamaño de la ventana (*span*) y el factor de roll-off.

Después se crean los filtros de transmisión RC y SRRC, de la misma manera en la que se describió para la sección 2 del script *reformado.m*. Además, se normaliza el filtro RC ajustando la ganancia del mismo mediante el parámetro *Gain*. El filtro de recepción SRRC se implementa mediante el objeto del sistema llamado *comm.RaisedCosineReceiveFilter*,

la propiedad *Shape* toma el valor de 'Square root', la propiedad *RolloffFactor* toma el valor definido en la variable *rolloff*, la propiedad *FilterSpanInSymbols* toma el valor seteado en *span*, la propiedad *InputSamplesPerSymbol* toma el valor guardo dentro de la variable *sps* y la propiedad *DecimationFactor* toma el valor de 1. Cuando en esta última propiedad se asigna el valor de 1, no se diezma la señal de salida cuando se aplique el filtrado.

Luego de se define la secuencia de bits de entrada a ser filtrados, esta secuencia es; 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1 y 1. Después se calcula el vector tiempo para el vector de entrada. Luego el vector de entrada se transforma en un vector columna y se rellena con *span* ceros. A continuación, se aplican los filtros de transmisión RC y SRRC al vector rellenado. Después se calcula el vector de tiempo para las señales filtradas y se corrige la fase de dicho vector tiempo.

A la señal de salida del filtro SRRC de transmisión se debe aplicar el filtro SRRC de recepción. Pero antes de hacerlo es necesario agregar un relleno de ceros igual a *span*sps* para poder obtener la señal completa que se desea graficar. Una vez filtrada la señal se recortan los datos que han sido pasados por dos filtros SRRC, tal que se ajuste al tamaño de la ventana con el que se ha venido trabajando. Finalmente se grafican los datos de entrada con color azul, la señal filtrada con RC con color rojo y la señal filtrada por dos filtros SRRC con color verde.

La sección de código descrita en estos últimos párrafos y correspondiente a una tercera parte del script *reformado.m*, se presenta debidamente comentada a continuación.

```
%% Sección 3: Dos Filtros SRRC y comparación con uno RC
clear; close all;
% Parámetros del filtro de transmisión:
rolloff = 0.5;  % Factor de caída del filtro (roll off) (beta)
span = 6;       % Define el tamaño de la ventana en símbolos
sps = 8;        % # de muestras por cada símbolo
% Creación del filtro RC:
txfilterRC = comm.RaisedCosineTransmitFilter(...
    'Shape','Normal','RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span,'OutputSamplesPerSymbol',sps);
b = coeffs(txfilterRC); % Obtener los ceficientes del polinomio
txfilterRC.Gain = 1/max(b.Numerator); % Normalizar el filtro RC a
% través de la propiedad Gain del filtro RC
% Creación del filtro SRRC de transmisión:
txfilter = comm.RaisedCosineTransmitFilter(...
    'Shape','Square root',...
    'RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span, ...
    'OutputSamplesPerSymbol',sps);
% Creación del filtro SRRC de recepción:
rxfilter = comm.RaisedCosineReceiveFilter(...
    'Shape','Square root','RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span,'InputSamplesPerSymbol',sps, ...
    'DecimationFactor',1);
```

```
% Se utiliza un 'DecimationFactor' igual a 1 para graficar la señal
% filtrada con dos filtros SRRC y es igual a sps cuando se quiere
% recuperar los bits de entrada
x=[1 1 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 1]; % Vector de datos
tx=(0:length(x)-1); % Vector de tiempo para los datos de entrada
xr=[x.';zeros(span,1)]; % Crear un vector con relleno de ceros,
% para que obtener el tamaño de la ventana completo y recuperar
% todos los datos de entrada en recepción.
% Filtrado:
yf0RC = txfilterRC(xr); % Aplicar el filtro RC a los datos de entrada
yf0 = txfilter(xr); % Aplicar el filtro SRRC a los datos de entrada
tf0 = (0:(length(x)+span)*sps-1)/sps; % vector de tiempo para yf0
% Corregir delay del vecot tiempo de yf0:
t = tf0-(span/2); % se retrasa el tiempo para centrar la señal
% Filtro SRRC de recepción:
yr= rxfilter([yf0; zeros(span*sps, 1)]); % Filtrar los datos agregando
% un relleno de ceros de longuitud span*sps
yr0= yr(span*sps/2+1:end-span*sps/2); % Recortar la señal filtrada
% Graficar los filtros implementados:
figure()
stem(tx,x,'filled') % señal de entrada
hold on
plot(t, yf0RC, '-r','LineWidth',2) % Datos en fase RC
plot(t, yr0, '-g','LineWidth',1.2) % Datos en fase de dos SRRC
hold off
ylim([-0.3 1.3]) % Limitar
title('Filtrado SRRC de transmisión y recepción')
xlabel('Muestras de tiempo')
ylabel('Amplitud')
legend('Datos de entrada','RC','2 SRRC')
```

Luego de ejecutar la sección de código anterior se obtiene las variables en el espacio de trabajo mostradas en la Figura 2.77. Además, se obtiene la gráfica que se muestra en la Figura 2.78.

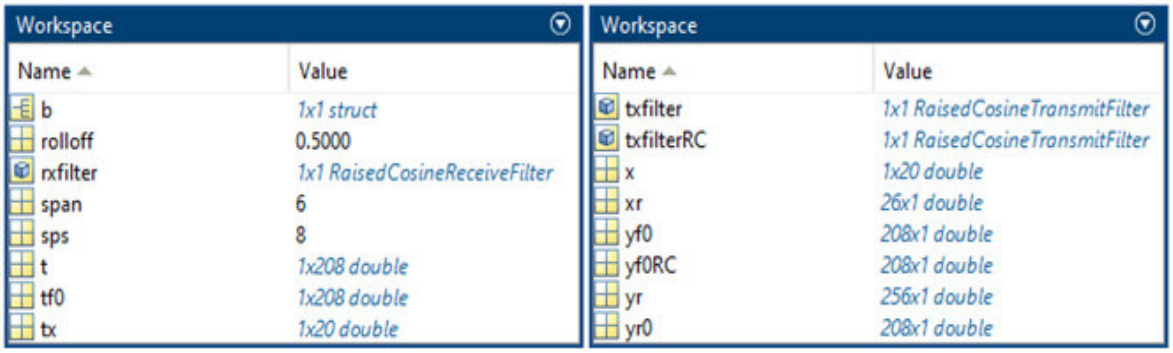


Figura 2.77. Variables en el espacio de trabajo luego de ejecutar la sección 3 del script *reformado.m*.

En la Figura 2.78 se muestra los datos de entrada con color azul, la señal filtrada con un filtro RC con color rojo y la señal filtrada por dos filtros SRRC con color verde. La línea roja es más gruesa que la verde, esto con el objetivo de que se evidencie de mejor manera, como se sobrepone la señal en verde. Las señales filtradas son las mismas, esto comprueba lo visto en la parte teórica, de que dos filtros SRRC resultan en un filtro RC.

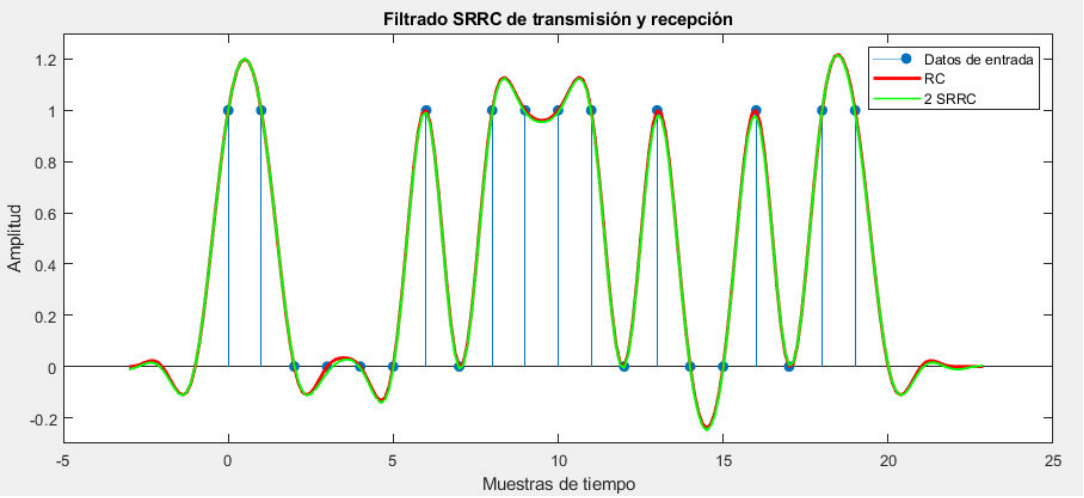


Figura 2.78. Comparación del filtrado utilizando dos filtros SRRC con utilizar un filtro RC.

Los objetos del sistema de transmisión y recepción permiten implementar la técnica de reformado de pulso de una manera eficiente. Por esto el prototipo de comunicación inalámbrica que se está desarrollando utilizará los objetos del sistema *comm.RaisedCosineTransmitFilter* y *comm.RaisedCosineReceiveFilter*.

2.3.5.3. Filtro SRRC aplicado al archivo de texto mapeado

En la etapa Mapeo se modularon con QPSK los bits codificados del archivo de texto previamente seleccionado. Para implementar en un script la etapa Reformado de pulso, se utilizan los datos mapeados con QPSK como entrada del filtro SRRC de transmisión. Sin embargo, este ejemplo puede servir para cualquier esquema de mapeo, es decir es un script general.

En las etapas anteriores se implementaron los scripts *abrir_archivo.m*, *serializar_datos.m*, *cod_conv.m* y *mapeo_datos.m* correspondientes a las etapas de Datos, Serializar, Codificador Convolucional y Mapeo respectivamente. Entonces, antes de continuar con la etapa actual es necesario que se ejecuten los scripts mencionados. Es preferible que el espacio de trabajo de Matlab no se sature de variables de todos los scripts ejecutados previamente, razón por la cual se borran todas las variables generadas en scripts previos, a excepción del vector *datosMapeados*. Para dicho efecto se ejecuta el siguiente código.

```
clearvars -except datosMapeados % Borrar todas las variables excepto
% la variable datosMapeados
```

La variable *datosMapeados* contiene los símbolos modulados con QPSK y normalizados, obtenidos en la etapa Mapeo.

El filtro de transmisión SRRC correspondiente a la etapa de Reformado de pulso se implementa en un script llamado *reformado_tx.m*.

El código del script *reformado_tx.m* inicia por establecer los parámetros del filtro de transmisión. Se utiliza una variable llamada *rolloff* en donde se guarda el valor del factor de caída del filtro que para un filtro SRRC se lo suele identificar con el símbolo β. Se utiliza un factor de caída igual a un valor de 0.5, para no caer en ninguno de los extremos, es decir 0 o 1. El tamaño de la ventana se almacena en la variable *span* y se guarda el valor de 6. El número de muestras que se tendrán por cada símbolo de entrada se almacena en la variable *sps* y toma el valor de 4.

El valor de *sps* tiene que ser necesariamente 4 ya que en la etapa Codificador Convolucional se definió este valor para calcular el número de bits de datos que se van a tomar por cada símbolo OFDM para aplicar la corrección de errores. Si la corrección de errores está activada se podría utilizar un valor de *sps* de 2 o de 4, si se utiliza 2 la señal resultante seguiría siendo brusca y por tanto no mejoraría en gran medida el BER. Cuando no se utiliza la técnica de corrección de errores es posible subir el valor de *sps* a 8, mejorando así el BER, un valor mayor a 8 conlleva a que la señal a transmitir crezca de longitud lo cual no es deseable.

Luego se crea el filtro SRRC con el objeto del sistema llamado *comm.RaisedCosineTransmitFilter*. En el parámetro *Shape* se asigna el valor 'Square root', en el parámetro *RolloffFactor* se asigna el valor guardado en la variable *rolloff*, en el parámetro *FilterSpanInSymbols* se asigna el valor dentro de *span* y a la propiedad *OutputSamplesPerSymbol* se asigna *sps*. El filtro creado se almacena en la variable *txfilter*.

Después es necesario rellenar el vector columna *datosMapeados* con un vector de ceros de longitud *span*, el resultado se guarda en la variable *datosM_relleno*. El realizar el relleno permite recuperar todos los datos filtrados en recepción, en caso de no hacerlo se pierden los últimos *span* datos. Luego se aplica el filtro *txfilter* a los datos que contiene la variable *datosM_relleno* y el resultado se almacena en la variable *datosReformado*.

Luego se grafica la parte real de los primeros 50 datos de entrada que contiene el vector *datosMapeados* y se sobrepone la señal filtrada real. También se grafica el diagrama de constelación del vector de entrada y en la misma gráfica se sobrepone el diagrama de constelación con los datos filtrados. El código debidamente comentado, descrito en párrafos anteriores, se presenta a continuación.

```
% Script para mostrar un ejemplo del filtro de transmisión de la
% etapa Reformado de Pulso
```

```
% Previamente a ejecutar el script actual es necesario ejecutar en
% orden los scripts:
% abrir_archivo.m, serializar_datos.m, cod_conv.m y mapeo_datos.m
% Como argumento de entrada del script actual se tiene una vector
% columa con símbolos mapeados y normalizados, el cual se obtiene
% en el script mapeo_datos.m y que tiene el nombre de datosMapeados
% Parámetros del filtro de transmisión:
rolloff = 0.5;  % Factor de caída del filtro
span = 6;       % Define el tamaño de la ventana en símbolos
sps = 4;        % # de muestras por cada símbolo
% En el filtro de transmisión se define el parámetro 'Shape' como
% 'Square root' para obtener un filtro SRRC
% Creación del filtro SRRC:
txfilter = comm.RaisedCosineTransmitFilter(...
    'Shape','Square root','RolloffFactor',rolloff, ...
    'FilterSpanInSymbols',span,'OutputSamplesPerSymbol',sps);
% A los datos de entrada datosMapeados de aplica el filtro txfilter
% y se almacena en la variable datosReformado
datosM_relleno=[datosMapeados; zeros(span,1)];
datosReformado=txfilter(datosM_relleno); % datos filtrados
% Graficar:
% Primero se grafica la parte real de los primeros 50 datos de
% datosMapeados y luego la señal filtrada para esos datos
t = ((0:(50+span/2-1)*sps)/sps)-span/2; % vector tiempo para
% graficar la señal filtrada
figure()
stem((0:50-1),real(datosMapeados(1:50)), 'filled')
hold on
plot(t,real(datosReformado(1:length(t))),'LineWidth',1.2)
hold off
title('Primeros datos de la señal antes y después del filtrado SRRC')
legend('Antes','Después')
xlabel('Muestras de tiempo');
ylabel('Amplitud');
axis([-4 60 -0.6 0.6]) % Limitar ejes
% Graficar digramas de constelación de los datos filtrados y de
% los datos modulados normalizados (datos de entrada):
figure()
% Graficar los datos modulados normalizados (QPSK solo tiene 4 simb.)
plot(datosMapeados(1:100,1),'r.','MarkerSize',20);
hold on
% Graficar los puntos de los datos filtrados, con azul
plot(datosReformado,'b.');
title('Señal antes y después del filtrado SRRC')
legend('Antes','Después')
xlabel('I (fase)');
ylabel('Q (cuadratura)');
axis([-0.6 0.9 -0.6 0.6]) % Limitar ejes
hold off
```

Una vez ejecutado el script *reformado_tx.m*, se obtiene en el espacio de trabajo las variables que se muestran en la Figura 2.79. También se obtiene una gráfica de la parte real de los primeros 50 datos de entrada y de la señal real filtrada obtenida para dichos datos, esta gráfica se muestra en la Figura 2.80. Además, se obtiene el diagrama de

constelación de los datos de entrada y de los datos filtrados, el diagrama se puede ver en la Figura 2.81.

La variable *datosReformado* contiene a los datos filtrados y su longitud es *sps* veces la longitud del vector *datosM_relleno*. Este vector contiene los datos de entrada y el relleno de *span* ceros.

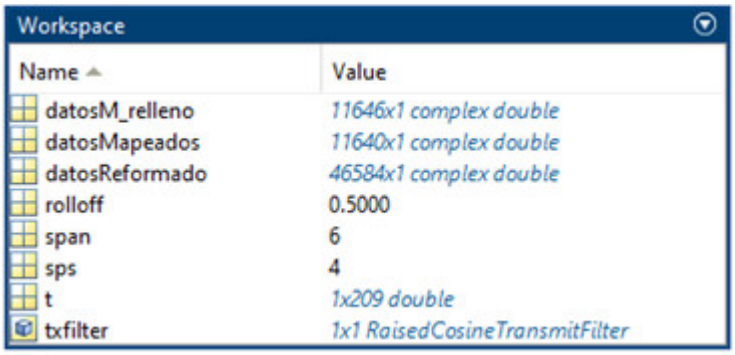


Figura 2.79. Variables en el espacio de trabajo luego de ejecutar *reformado_tx.m*.

En la Figura 2.80 se muestra con color azul la parte real de los primeros 50 datos de entrada (datos mapeados con QPSK y normalizados). Mientras que con color rojo se muestra la parte real de la señal filtrada con el filtro SRRC correspondiente a los 50 datos de entrada.

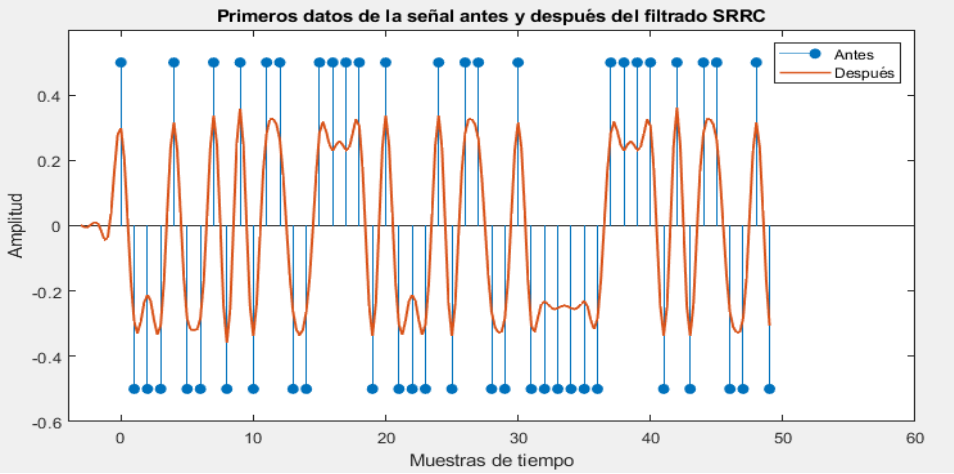


Figura 2.80. Parte real de los primeros 50 datos antes y después del filtrado SRRC.

En el diagrama de constelación de la Figura 2.81, los datos mapeados con QPSK y normalizados se identifican con puntos rojos. Por otro lado, los puntos azules corresponden a los datos filtrados contenidos en el vector columna *datosReformado*. Estos datos corresponden al vector de salida de la etapa Reformado de pulso y consecuentemente al vector de entrada de la etapa Creación del símbolo OFDM.

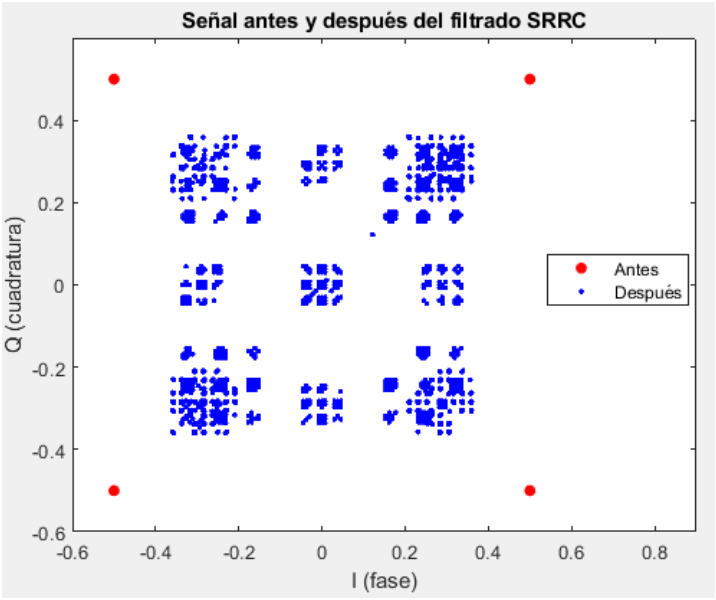


Figura 2.81. Datos de entrada mapeados con QPSK y luego de pasar por el filtro SRRC de transmisión.

2.3.6. ETAPA DE OFDM

Las siglas de la técnica OFDM viene del inglés Orthogonal Frequency Division Multiplexing, que en español se traduciría como Multiplexación por División de Frecuencia Ortogonal. OFDM consiste en dividir un flujo de datos de entrada en múltiples subportadoras, con el objetivo de evitar varias alteraciones del canal inalámbrico. El flujo de datos de entrada puede ser datos mapeados con BPSK, QPSK, 16-QAM o 64-QAM, también pueden ser filtrados con un filtro SRRC y previamente mapeados con cualquiera de las técnicas mencionadas.

Entre las alteraciones que se pueden evitar con OFDM está la interferencia entre símbolos (ISI), siglas que vienen del inglés Inter-Symbol Interference. Otra alteración que se evita con OFDM es la interferencia entre portadoras (ICI), siglas que vienen del inglés Inter-Carrier Interference. Además, OFDM es resistente al multitrayecto y al desvanecimiento selectivo en frecuencia.

Las etapas de transmisión que intervienen para la implementación de la técnica de transmisión OFDM se identifican con color anaranjado en la Figura 2.82. La etapa Creación del símbolo OFDM se encarga de crear símbolos OFDM con una estructura que contiene varios tipos de subportadoras. La etapa IFFT se encarga de modular los símbolos OFM creados. La etapa Adición prefijo cíclico se encarga de copiar las últimas muestras de cada símbolo al inicio de los mismos.

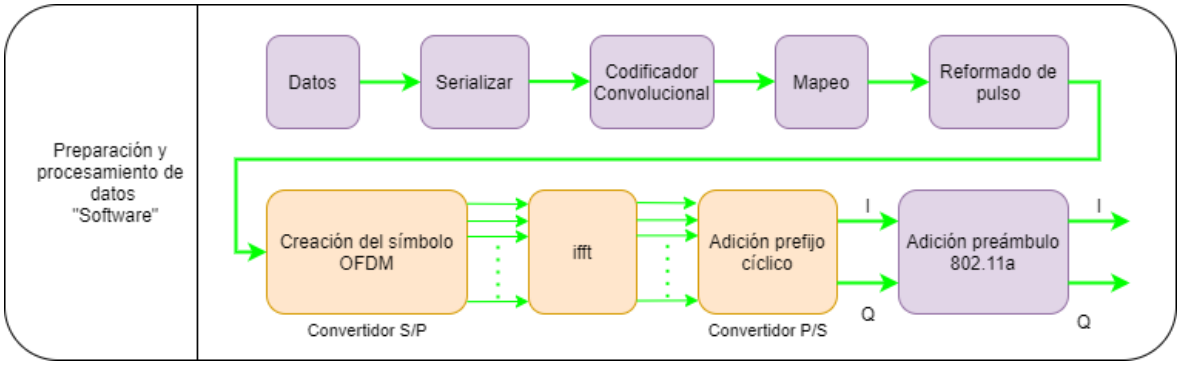


Figura 2.82. Etapas de transmisión con las etapas que intervienen en el esquema OFDM identificadas.

2.3.6.1. Fundamentos teóricos de OFDM

OFDM es la forma más popular de la modulación de múltiples portadoras (MCM), iniciales que vienen del inglés Multi-carrier Modulation [4]. Esta técnica de modulación tiene como entrada símbolos mapeados con PSK o QAM.

Las principales características de la técnica de transmisión OFDM se listan a continuación:

- Contrarresta los efectos de la ISI y la ICI.
- Es resistente al desvanecimiento selectivo de frecuencia.
- Es resistente las distorsiones del multitrayecto
- Permite la estimación y la ecualización del canal en el receptor.

En OFDM los datos se transmiten como una combinación de señales ortogonales de banda estrecha a las cuales se suele llamar subportadoras o subcanales [33]. OFDM utiliza un conjunto portadoras ortogonales sobrelapadas, lo que permite obtener un uso óptimo del espectro de frecuencia y evitar la interferencia entre portadoras ICI. Un ejemplo de portadoras ortogonales sobrelapadas se muestra en la Figura 2.83.

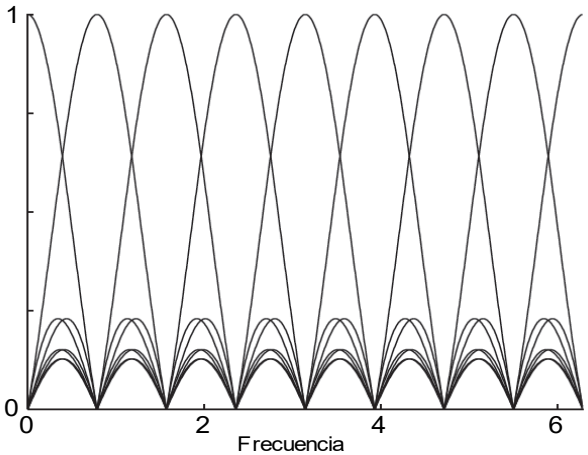


Figura 2.83. Ejemplo de portadoras de frecuencia ortogonales.

OFDM permite combatir los efectos del desvanecimiento selectivo en frecuencia al dividir los datos en varias subportadoras. Cuando aparece una alteración en frecuencia del canal inalámbrico, esta no va a afectar a todo el rango de frecuencia, sino más bien a unas pocas portadoras. Esto permite que no se pierdan todos los datos, sino solamente los datos dentro de las subportadoras afectadas.

2.3.6.1.1. Estructura de un símbolo OFDM

Cada símbolo OFDM está compuesto de subportadoras de datos, subportadoras pilotos, subportadoras de guarda y una subportadora DC [7]. La estructura en banda base del símbolo OFDM que se utiliza en el presente proyecto se muestra en la Figura 2.84. Dicha estructura contiene una subportadora DC, 40 subportadoras de datos, 4 subportadoras piloto y 19 subportadoras de guarda, danto un total de 64 subportadoras por cada símbolo OFDM.

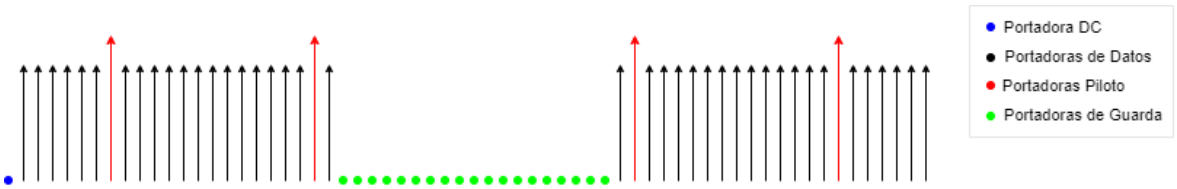


Figura 2.84. Estructura de un símbolo OFDM en banda base.

Las **portadoras de guarda** permiten evitar la interferencia de canal adyacente (ACI). El espectro de cualquier señal OFDM es el resultado de la suma de varias funciones sinc, desplazadas en frecuencia. Esto conlleva a tener una gran potencia fuera de banda y consecuentemente producir el fenómeno ACI con canales vecinos [2]. Para evitar el efecto de la ACI en sistemas OFDM se utilizan bandas de guarda, con un valor de cero.

Las **portadoras piloto** se utilizan para corregir desviaciones en fase, desviaciones en frecuencia y los efectos del canal inalámbrico [4]. Estas subportadoras permiten realizar la estimación y la ecualización del canal de manera independiente para cada símbolo OFDM. Este proceso se realiza con la finalidad de compensar las alteraciones que se puedan presentar en las portadoras de datos. Dicho proceso se realiza en la parte de recepción, en transmisión solo se crean los símbolos OFDM.

En este proyecto se utiliza el arreglo de portadoras Comb Type, dado que se maneja una estructura de símbolo OFDM fija. Con Comb Type en cada símbolo OFDM las portadoras piloto se ubican periódicamente en el dominio de frecuencia y para estimar el canal se usa interpolación en este eje. En la Figura 2.85 se muestra un ejemplo del arreglo de portadoras Comb Type, con los ejes del tiempo y de frecuencia identificados.

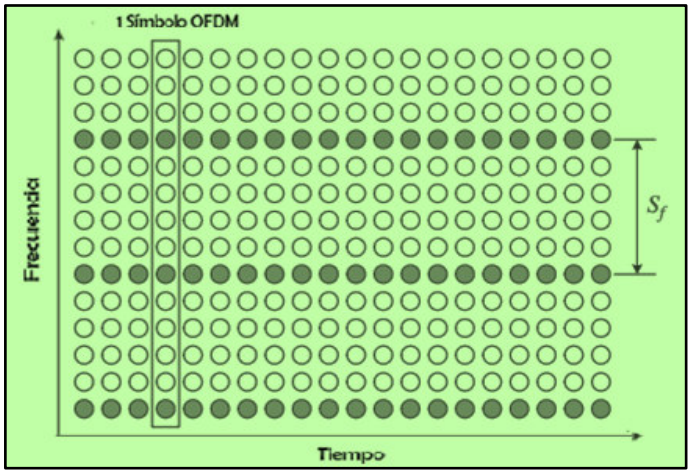


Figura 2.85. Arreglo de portadoras Comb Type [34].

2.3.6.1.2. Modulador y demodulador OFDM

OFDM emplea la DFT (Transformada de Fourier Discreta) y la DFT inversa (IDFT) para demodular y modular los flujos de datos paralelos [4]. El significado físico de la IDFT es convertir señales del dominio de la frecuencia al dominio del tiempo. Mientras que con la DFT se puede convertir señales del dominio del tiempo al dominio de la frecuencia.

En OFDM los datos mapeados con PSK o QAM son modulados con la IDFT. La ecuación 2.17 es la IDFT para N subportadoras, representadas por $X_l[k]$.

$$x_l[n] = \sum_{k=0}^{N-1} X_l[k] e^{j2\pi \frac{kn}{N}}, \qquad para\ n = 0,1,2,\dots,N-1 \tag{2.17}$$

- l es el número de símbolo OFDM y puede tomar valores de 0 a ∞.
- k es el número de subportadora dentro de un símbolo OFDM y toma valores de 0 a N-1.
- N es el número total de subportadoras del símbolo OFDM.

Para demodular los símbolos OFDM luego de pasar por un canal con ruido se utiliza la DFT. La ecuación 2.18 es la DFT para N subportadoras moduladas pasadas por un canal con ruido y representadas por $y_l[k]$. Los parámetros l, k y N representan lo mismo que en la ecuación 2.17.

$$Y_l[k] = \sum_{k=0}^{N-1} y_l[k] e^{-j2\pi \frac{kn}{N}}, \qquad para\ n = 0,1,2,\dots,N-1 \tag{2.18}$$

La Figura 2.86 muestra un diagrama de bloques que representa la modulación y demodulación de un símbolo OFDM con $N = 6$. De izquierda a derecha, se inicia por la entrada de un flujo de datos, continua con mapeo PSK o QAM, se convierte de serie a

paralelo, se modula los datos, el símbolo OFDM se transmite, se demodulan los datos, se convierte de paralelo a serie, se demapean los datos en serie y se recupera el flujo de datos.

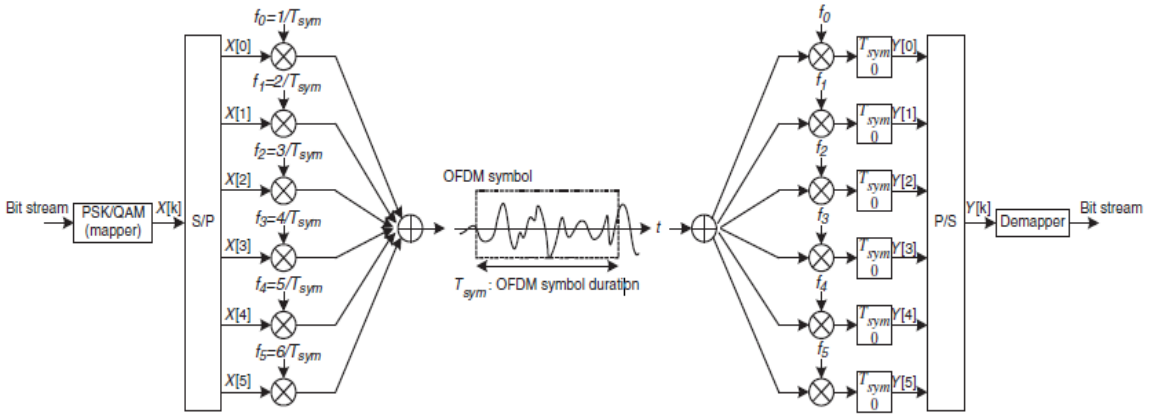


Figura 2.86. Diagrama de bloques de la modulación y demodulación OFDM [2].

La implementación eficiente de la DFT y de la IDFT se realiza mediante la FFT (Transformada Rápida de Fourier) y la FFT invertida (IFFT), respectivamente. Entonces, cada símbolo OFDM se debe modular aplicando la IFFT y demodular mediante la FFT.

2.3.6.1.3. Prefijo cíclico (PC)

El multitrayecto provoca que símbolos OFDM consecutivos se mezclen provocando ISI y al perder la ortogonalidad de las subportadoras también incurra en ICI [2]. Estos efectos se contrarrestan insertando un intervalo de guarda al inicio de cada símbolo OFDM, el cual es conocido como Prefijo Cíclico (PC).

El PC consiste en copiar un número de muestras del final del símbolo OFDM resultante de la IFFT al inicio del mismo símbolo. En la Figura 2.87 se muestra dos símbolos OFDM con prefijo cíclico, en donde T_{sub} es la duración del símbolo sin PC, T_G es la duración del PC y T_{sym} es la duración del símbolo OFDM con PC. Al agregar este prefijo el símbolo OFDM crece de tamaño y por ende la eficiencia espectral disminuye.

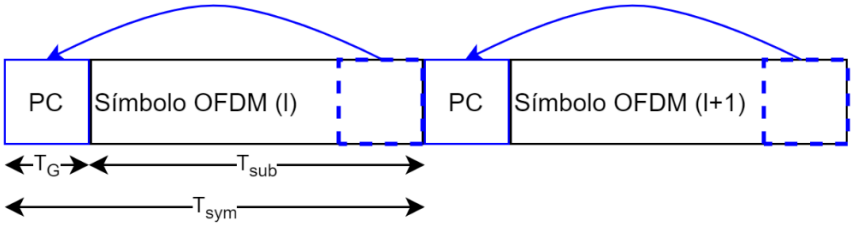


Figura 2.87. Símbolos OFDM con PC.

Copiar las muestras finales al inicio del símbolo permite mantener la ortogonalidad de las subportadoras y por ende evitar la ICI. En la Figura 2.88 se muestra el desvanecimiento a

la entrada y a la salida de un símbolo OFDM recibido, en cualquier caso, es menor al intervalo de tiempo que dura el prefijo cíclico (T_G).

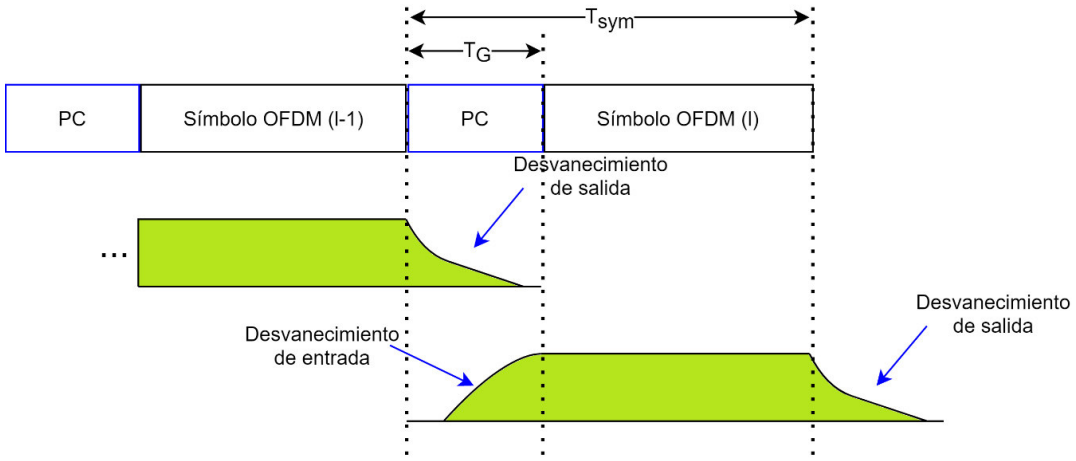


Figura 2.88. Efecto del prefijo cíclico (PC) en los símbolos OFDM.

Cuando la longitud de PC es mayor o igual al retardo máximo de un canal multitrayecto, el efecto del ISI de un símbolo OFDM sobre el siguiente símbolo, se mantiene confinado dentro del intervalo del PC [2].

En las pruebas de funcionamiento del prototipo de comunicación inalámbrica no se pretende superar los 10 metros de separación directa entre el SDR de transmisión y el SDR de recepción. Asumiendo que la onda reflejada más larga recorra en total 20 metros, la diferencia de distancia es de 10 metros. En la Figura 2.89 se presenta un esquema básico de una onda directa y de una onda reflejada.

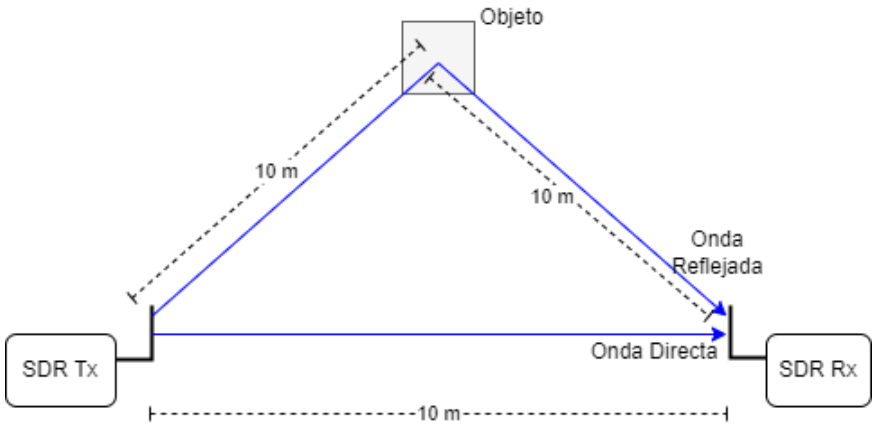


Figura 2.89. Esquema básico de una onda reflejada en un canal multitrayecto.

El tiempo de propagación t_p de una onda electromagnética se puede calcular mediante la ecuación 2.19. En dicha ecuación d representa la distancia en metros y v_p es la velocidad de propagación de una onda electromagnética, esta velocidad es igual a $300.000.000\,\frac{m}{s}$.

$$t_p = \frac{d}{v_p} \tag{2.19}$$

Para calcular el retardo máximo que se va a tener en las pruebas de funcionamiento futuras, se utiliza la ecuación 2.19. En donde d toma el valor de 10 metros, este valor corresponde a la diferencia de la distancia entre la onda directa y la onda reflejada de la Figura 2.90. Evaluando la ecuación se tiene que:

$$t_p = \frac{d}{v_p} = \frac{10\ m}{3 * \frac{10^8 m}{s}} = 33{,}33 * 10^{-9}\ s = 33{,}33\ ns$$

Entonces el retardo máximo del canal multitrayecto es aproximadamente de $33{,}33\ ns$. El prefijo cíclico debe durar más que dicho valor, por lo que primero se debe calcular la duración de un símbolo OFDM. Hay que considerar que los dispositivos SDR Adalm Pluto se configuraron con una frecuencia de muestreo de $2 * 10^6$ muestras por segundo (sps) y que cada símbolo OFDM sin PC está conformado por 64 muestras. Con estos valores se puede aplicar una regla de tres para calcular la duración en tiempo de un símbolo OFDM (t_{simb}):

$muestras$	$tiempo\ (s)$
$2 * 10^6$	1
64	t_{simb}

Desarrollando se tiene que:

$$t_{simb} = \frac{64 * 1}{2 * 10^6}\ [s] = 32 * 10^{-6}\ s = 32\ \mu s$$

Un símbolo OFDM sin PC tiene una duración de $32\ \mu s$, si se considera que el prefijo cíclico sea la cuarta parte del símbolo OFDM la duración del PC sería de $8\ \mu s$. Este valor resulta ser mayor que el retardo máximo de $33{,}33\ ns$ y por tanto se cumple que la longitud de PC es mayor al retardo máximo de un canal multitrayecto. Esto implica que el efecto del ISI de un símbolo OFDM sobre el siguiente símbolo, se mantiene confinado dentro del intervalo del prefijo cíclico [2]. Como el símbolo OFDM tiene una longitud de 64 muestras, un PC de tamaño igual a la cuarta parte del símbolo tendría una longitud de 16 muestras. En el presente proyecto se utiliza 16 muestras para crear el PC.

2.3.6.2. Implementación de OFDM en Matlab

Como se mencionó anteriormente, la técnica de transmisión OFDM se implementa en las etapas de: Creación del símbolo OFDM, IFFT y Adición prefijo cíclico. En la Figura 2.90 se

identifican estas tres etapas, dentro de la preparación y procesamiento de los datos en transmisión.

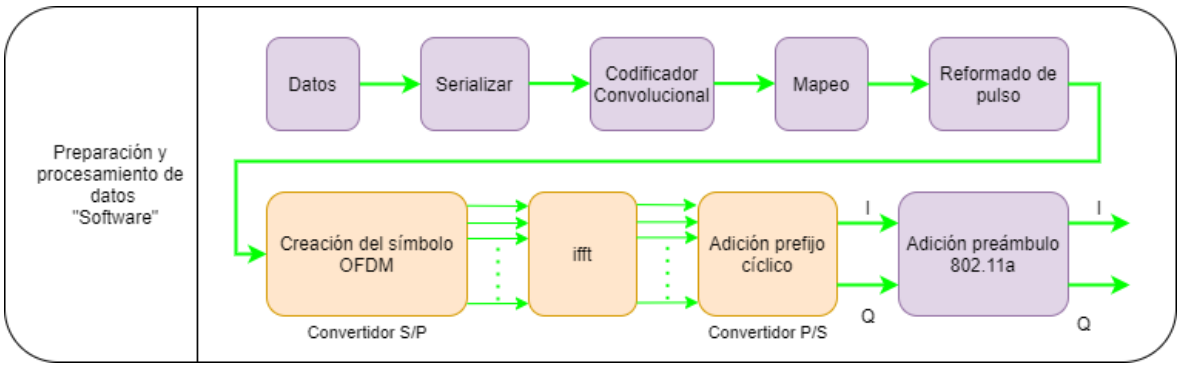


Figura 2.90. Etapas de transmisión con las subetapas que intervienen en el esquema OFDM identificadas.

La implementación en Matlab de la **subetapa Creación del símbolo OFDM** se realiza de manera manual, por lo que no es necesario utilizar una función directa para la creación del símbolo OFDM. En primer lugar, se realiza un relleno del vector de entrada para que sea múltiplo del número de subportadoras de datos. Luego mediante un lazo *for* se crea los símbolos OFDM, tal que cada uno contenga una portadora DC, 4 pilotos, 40 de datos y 19 de guarda, como se muestra en la Figura 2.91.

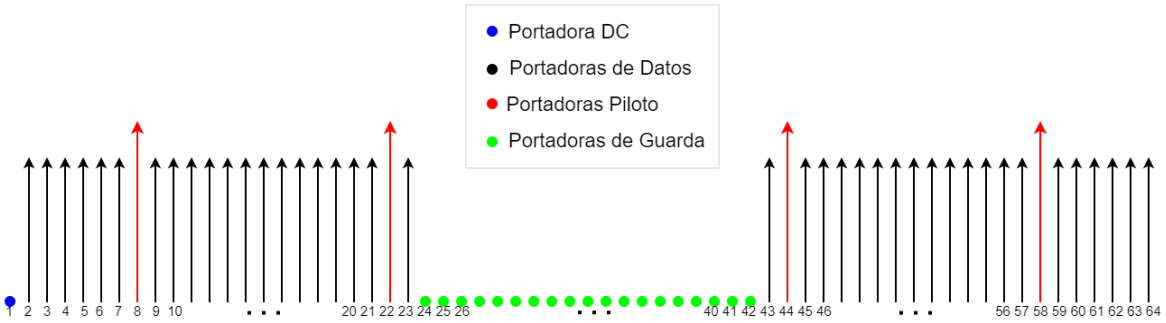


Figura 2.91. Estructura del símbolo OFDM.

En la Tabla 2.12 se muestra la asignación de los 40 datos dentro de las portadoras de datos del símbolo OFDM. Además, cada símbolo OFDM creado se almacena en una columna de la matriz de salida de la etapa.

Tabla 2.12. Posición de los 40 datos en el símbolo OFDM.

Posición en el símbolo	Posición datos	Posición en el símbolo	Posición datos	Posición en el símbolo	Posición datos	Posición en el símbolo	Posición datos
2	21	13	31	43	1	54	11
3	22	14	32	45	2	55	12
4	23	15	33	46	3	56	13
5	24	16	34	47	4	57	14
6	25	17	35	48	5	59	15
7	26	18	36	49	6	60	16
9	27	19	37	50	7	61	17
10	28	20	38	51	8	62	18
11	29	21	39	52	9	63	19
12	30	23	40	53	10	64	20

La **etapa IFFT** se implementa en Matlab mediante la función directa llamada *ifft*. La sintaxis que utiliza dicha función es la siguiente:

```
datS=ifft(datosEnt)
```

Mediante la sintaxis anterior se calcula la transformada inversa discreta de Fourier (IDFT) de la variable de entrada *datosEnt*, utilizando un algoritmo de transformada rápida de Fourier. El argumento de entrada *datosEnt* puede ser un vector columna, en cuyo caso se devuelve la transformada inversa de dicho vector, o también puede ser una matriz, en cuyo caso se devuelve la transformada inversa de cada columna de la matriz [35]. Los datos de entrada de la etapa IFFT es una matriz que contiene en cada columna un símbolo OFDM, por lo que se puede aplicar la *ifft* directamente a la matriz de entrada.

La **etapa Adición prefijo cíclico** se implementa mediante un lazo *for*, recorriendo las columnas de la matriz que contiene los símbolos modulados con la *ifft*. El símbolo OFDM tiene una longitud de 64 y como se va a utilizar un prefijo cíclico (PC) de un cuarto de la longitud del símbolo, la longitud del prefijo cíclico es de 16. En cada iteración del lazo *for* se copia las muestras finales del símbolo al inicio del mismo y además se serializa los símbolos con PC en un vector.

2.3.6.3. OFDM aplicado al archivo de texto filtrado con SRRC

En etapas anteriores se procesaron los datos de un archivo de texto hasta pasarlos por un filtro SRRC en la etapa Reformado de pulso. Por lo que, es necesario ejecutar los scripts correspondientes a cada una de las etapas anteriores. Las etapas Datos, Serializar, Codificador Convolucional, Mapeo y Reformado de pulso se implementan en los scripts

abrir_archivo.m, *serializar_datos.m*, *cod_conv.m*, *mapeo_datos.m* y *reformado_tx.m* respectivamente.

Una vez ejecutados los scripts con el archivo de texto *ArchivoTextoRT.txt* se debe borrar todas las variables del espacio de trabajo a excepción de *datosReformado*. Esta variable contiene los datos filtrados y que servirán como entrada a la etapa Creación del símbolo OFDM. La siguiente sección de código permite eliminar todas las variables del espacio de trabajo a excepción de *datosReformado*.

```
clearvars -except datosReformado  % Borrar todas las variables excepto
% la variable datosReformado
```

Siguiendo el ejemplo con el que se ha venido trabajando en el documento actual, la variable *datosReformado* contiene datos modulados con QPSK y filtrados con un filtro SRRC. En el workspace aparece la variable **datosReformado** con una dimensión de 46584x1 complex.

Las etapas de Creación del símbolo OFDM, IFFT y Adición prefijo cíclico se implementan dentro de un script llamado *ofdm_tx.m* y separadas en secciones.

La **primera sección del script *ofdm_tx.m*** corresponde a la e**tapa Creación del símbolo OFDM** y el código inicia por establecer el número de portadoras de datos. Luego se realiza un relleno del vector de entrada para que el mismo sea múltiplo de las portadoras de datos, es decir múltiplo de 40. El relleno se realiza al final del vector con los primeros datos del mismo vector. En caso de no ser necesario un relleno, los datos de entrada se mantienen sin cambios.

Después se inicializa en uno una variable que sirve para almacenar los símbolos OFDM en la matriz de salida de la etapa Creación de símbolo OFDM, la variable toma el nombre de *numeroSimbolos*. Luego con un lazo *for* se construye los símbolos OFDM. Dicho lazo recorre desde un valor inicial de uno, con una separación de 40, hasta llegar a la longitud del vector que contiene los datos de entrada con relleno. Dentro del lazo *for* en una variable auxiliar se copian los 40 datos, que se colocan en las subportadoras de datos del símbolo OFDM correspondiente.

Luego se establece el valor de la subportadora DC en cero y el de las subportadoras piloto en uno. Seguido se inicializa un vector columna de longitud 64 que contendrá el símbolo OFDM actual, el valor inicial es todas las posiciones en cero. Después se asigna los valores a las subportadoras DC, a las piloto y a las de datos, las subportadoras de guarda se mantienen en cero. Las posiciones se establecen igual al de la Figura 2.91.

Una vez creado el símbolo OFDM, en una variable llamada *simboloOFDMCreados* se almacena en cada columna el símbolo OFDM completo. La posición del símbolo la dicta la variable *numeroSimbolos* y luego dicha variable aumenta en uno. Cuando se terminen de crear todos los símbolos, la variable *numeroSimbolos* disminuye en uno para que coincida con la cantidad total de símbolos creados.

La sección correspondiente a la etapa IFFT realiza la modulación de los símbolos OFDM creados a través de la función directa de Matlab llamada *ifft*. Cuando dicha función recibe como argumento de entrada una matriz, se aplica a cada columna la IFFT y por tanto devuelve una matriz del mismo tamaño. La matriz de salida de la etapa IFFT se llama *simbolosOFDM*.

La sección de código correspondiente a la etapa Adición prefijo cíclico inicia con establecer un vector vacío que almacenará los símbolos serializados. Se utiliza un lazo *for* para recorrer todos los símbolos modulados y copiar las últimas 16 muestras al inicio del mismo símbolo. En la **matriz *simbolosOFDMpc*** se almacenan los símbolos OFDM modulados y con prefijo cíclico en cada columna. En el vector *datosEnviar* se almacenan los símbolos modulados y con prefijo cíclico de forma serial.

Finalmente se grafica el primer símbolo para mostrar un ejemplo visual de lo que se está haciendo en el prefijo cíclico. El código descrito en estos párrafos, correspondiente al script *ofdm_tx.m*, se muestra a continuación y el mismo se encuentra debidamente comentado.

```
% Script para implementar la técnica de transmisión OFDM en la parte
% del transmisor, consta de las etapas de Creación del símbolo OFDM,
% IFFT y Adición prefijo cíclico.
% Previamente a ejecutar el script actual es necesario ejecutar en
% orden los scripts: abrir_archivo.m, serializar_datos.m, cod_conv.m,
% mapeo_datos.m y reformado_tx.m
% Como argumento de entrada del script actual se tiene un vector
% columa con símbolos mapeados, normalizados y filtrados, o bien solo
% mapeados y normalizados, el cual se obtiene en el script
% reformado_tx.m y que tiene el nombre de datosReformado
%% %%%%%%%%%%%%%%%%%%% CREAR LOS SIMBOLOS OFDM %%%%%%%%%%%%%%%%%%%%%%%%
% El vector de salida de la etapa es simbolosOFDMCreados
% La estructura del símbolo OFDM consta de una subportadora DC,
% 4 pilotos, 19 de guarda y 40 de datos.
nPdatos=40; % Número de portadoras de datos
% El vector de entrada debe ser múltiplo de nPdatos, cuando no se
% se cumple dicha condición se realiza un relleno con los datos del
% inicio al final de vector. Esto con el objetivo de hacer que el
% vector sea múltiplo de nPdatos:
longDatos=length(datosReformado); % Longitud del vector de entrada
nSimbOFDM=ceil(longDatos/nPdatos); % Calcular el número de símbolos
longPadd=nSimbOFDM*nPdatos-longDatos; % Calcular la longitud del
% relleno
datosEntradaPadd=datosReformado; % Copio los datos de entrada en una
% nueva variable auxiliar
```

```
% Si la longitud de relleno es cero no se realiza ninguna operación
% adicional, si es mayor que cero se realiza el relleno
if(longPadd>0)
    % Se hace el relleno con los datos del inicio del vector de entrada
    datosEntradaPadd(end+1:end+longPadd)=datosReformado(1:longPadd);
end
% Se procede a crear los simbolos OFDM recorriendo en múltiplos de
% nPdatos, mediante un lazo for:
numeroSimbolos=1; % Inicializo la variable que sirve para almacenar los
% símbolos OFDM creados, en una matriz.
longDatosPadd=length(datosEntradaPadd); % Longitud de los datos de
% entrada con relleno
for i=1:nPdatos:longDatosPadd
    datosAux=datosEntradaPadd(i:i+nPdatos-1); %Copio los datos nPdatos,
    % es decir los 40 datos a introducirse dentro del símbolo OFDM
    DC=0;% Valor para la subportadora DC
    % Las variables p1, p2, p3, p4 son los valores que toman las
    % subportadoras piloto, usadas para poder hacer una estimación del
    % canal en recepción
    p1=1;
    p2=1;
    p3=1;
    p4=1;
    simbOFDM=zeros(64,1); % Inicializo símbolo OFDM, con todos los
    % valores en cero
    simbOFDM(1)=DC; % Portadora DC
    % Asignación del valor a cada subportadora de datos, y a las
    % subportadoras piloto:
    simbOFDM(2:7)=datosAux(21:26);
    simbOFDM(8)=p1; % Piloto
    simbOFDM(9:21)=datosAux(27:39);
    simbOFDM(22)=p2; % Piloto
    simbOFDM(23)=datosAux(40);
    % simbOFDM de 24 a 42 son de guarda
    simbOFDM(43)=datosAux(1);
    simbOFDM(44)=p3;% Piloto
    simbOFDM(45:57)=datosAux(2:14);
    simbOFDM(58)=p4; % Piloto
    simbOFDM(59:64)=datosAux(15:20);
    simbolosOFDMCreados(1:64,numeroSimbolos)=simbOFDM; % Matriz de
    % salida que almacena el símbolo OFDM completo, según la columna
    % que dicte la variable numeroSimbolo
    numeroSimbolos=numeroSimbolos+1; % Aumento en uno la variable
end
numeroSimbolos=numeroSimbolos-1; % Disminuyo en uno el valor de la
% variable para que concuerde con la cantidad total de símbolos OFDM
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% IFFT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% En esta sección se realiza la modulación de los símbolos OFDM
% mediante la ifft. La matriz de entrada es simbolosOFDMCreados y la de
% salida es simbolosOFDM.
% La función ifft admite como entrada un vector columna, pero tambien
% una matriz. En la matriz aplica la ifft a cada columna. Como la
% matriz de entrada contiene un spimbolo en cada columna se puede
% aplicar directamente la función ifft
simbolosOFDM=ifft(simbolosOFDMCreados); % matriz de salida
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% AÑADIR PC %%%%%%%%%%%%%%%%%%%%%%%%%%%%
% El prefijo cíclico (PC) se añade copiando la parte final de cada
% símbolo, al inicio del mismo.
% La matriz de entrada es simbolosOFDM y a la salida se tiene una
% matriz llamada simbolosOFDMpc con los símbolos en cada columna y
```

```
% un vector contodos los símbolos serializados llamado datosEnviar
% Con un lazo for se recorre cada símbolo modulado, se añade el PC
% y se serializa los datos
datosEnviar=[]; % Variable que almacenara los datos serializados
for auxPC=1:numeroSimbolos
    % Añadir el prefijo cíclico, de tamaño igual a la cuarta parte del
    % símbolo:

simbolosOFDMpc(1:80,auxPC)=[simbolosOFDM(49:64,auxPC);simbolosOFDM(:,au
xPC)];
    datosEnviar=[datosEnviar;simbolosOFDMpc(:,auxPC)]; % Datos
    % serializados
end
%% Graficar el primer símbolo con PC en el dominio del tiempo
t=1:80; % Variable que representa el tiempo
figure()
subplot(2,1,1)
% Graficar el PC y el símbolo sin PC
plot(t,real(simbolosOFDMpc(:,1)),'r')
hold on
plot(t(16:80),real(simbolosOFDMpc(16:80,1)),'b')
title('Símbolo OFDM con Prefijo Cíclico')
xlabel('Muestras de tiempo')
legend('Prefijo cíclico','Símbolo OFDM')
xlim([0 105])
hold off
subplot(2,1,2)
% Graficar el PC y las muestras finales que se usan como PC
plot(t,real(simbolosOFDMpc(:,1)),'r')
hold on
plot(t(65:80),real(simbolosOFDMpc(65:80,1)),'g')
plot(t(16:65),real(simbolosOFDMpc(16:65,1)),'b')
title('Símbolo OFDM con Prefijo Cíclico')
xlabel('Muestras de tiempo')
legend('Prefijo cíclico','Muestra finales')
xlim([0 105])
hold off
```

Una vez ejecutado todas las secciones del código anterior, en el espacio de trabajo de Matlab se obtiene las variables que se muestran en la Figura 2.92. Del total obtenidas, las variables que contienen la información más relevante se describen a continuación:

- El vector *datosReformado* contiene a los datos de entrada.
- El vector *datosEntradaPadd* contiene los datos de entrada con relleno, y este vector se utiliza para la creación de símbolos OFDM.
- La matriz *simbolosOFDMCreados* contiene los símbolos OFDM en el dominio de la frecuencia. Cada columna contiene un símbolo y cada símbolo tiene una longitud de 64 muestras.
- La matriz *simbolosOFDM* contiene los símbolos en el dominio del tiempo. Igual que en caso anterior, cada columna contiene un símbolo OFDM de una longitud de 64 muestras.

- La matriz *simbolosOFDMpc* contiene en cada columna a los símbolos OFDM con prefijo cíclico en el dominio del tiempo. Cada símbolo tiene una longitud de 80muestras.
- El vector columna *datosEnviar* contiene a todos los símbolos OFDM de la matriz *simbolosOFDMpc* serializados.

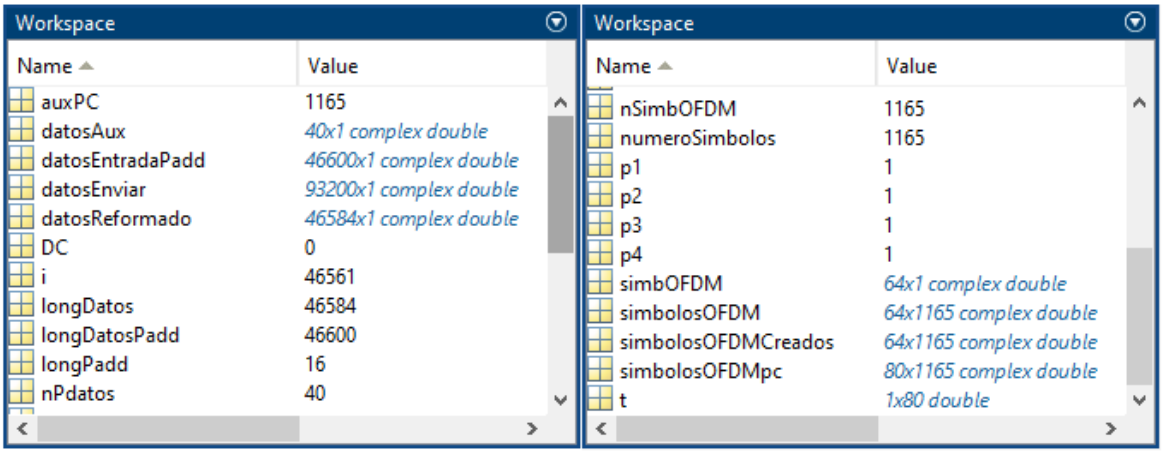


Figura 2.92. Variables en el espacio de trabajo luego de ejecutar el script *ofdm_tx.m*.

Además de las variables del espacio de trabajo descritas previamente, el código anterior grafica el primer símbolo de la variable *simbolosOFDMpc*. Dicho símbolo se muestra en la Figura 2.93, en donde se separa al prefijo cíclico del símbolo OFDM de longitud 64. También se identifica la parte final del símbolo, de la cual se copió los datos al inicio del mismo símbolo.

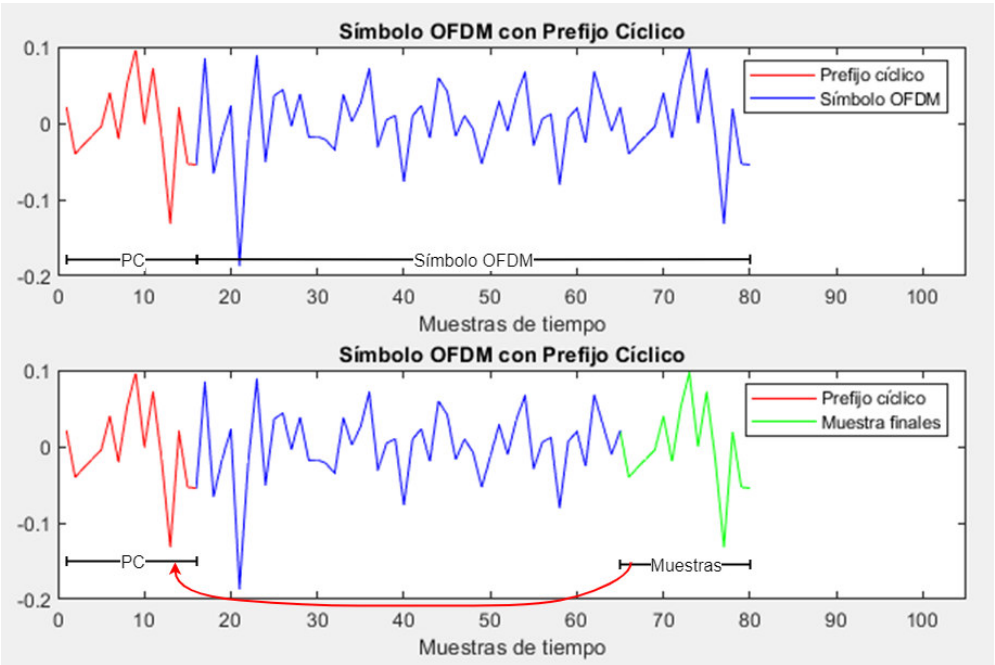


Figura 2.93. Gráfica del primer símbolo OFDM en tiempo.

2.3.7. ETAPA DE ADICIÓN DEL PREÁMBULO 802.11A

Dentro de las etapas de procesamiento de datos en transmisión, esta etapa se identifica con color naranja (Figura 2.94).

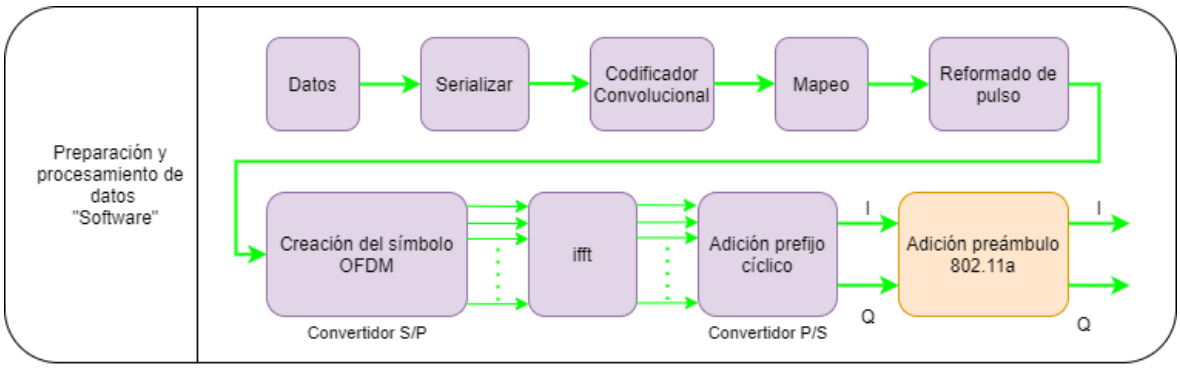


Figura 2.94. Etapas de transmisión con la etapa Adición preámbulo 802.11a identificada.

La etapa actual recibe como entrada a símbolos OFDM serializados y añade el preámbulo OFDM al inicio de los símbolos serializados. El preámbulo OFDM del estándar 802.11a permite al receptor realizar lo siguiente: a) detectar la trama OFDM transmitida, b) realizar la estimación del desplazamiento de frecuencia de la portadora, conocida por sus siglas CFO (Carrier Frecuency Offset), c) identificar el inicio exacto del paquete OFDM y d) corregir el paquete OFDM en fase.

2.3.7.1. Fundamentos teóricos

El preámbulo del estándar IEEE 802.11a está compuesto por dos partes: una sección corta (conocida como LSTF - Legacy Short Training Field) y una sección larga (conocida como LSTF - Legacy Short Training Field). En la Figura 2.95 se muestra una representación del preámbulo 802.11a seguido de símbolos OFDM, que se denomina como carga útil.

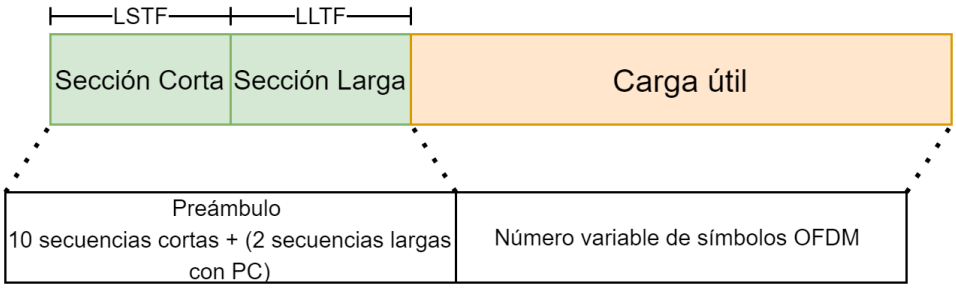


Figura 2.95. Preámbulo del estándar 802.11a y carga útil que contiene símbolos OFDM.

La **sección corta o LSTF** se compone de 10 copias repetidas de una **secuencia corta.**

Para obtener estas secuencias en el **dominio del tiempo**, primero se define 64 valores en el dominio de la frecuencia. En la Tabla 2.13 se puede ver los componentes reales e imaginarios para estos 64 valores. Para pasar los valores del dominio de la frecuencia al

dominio del tiempo se utiliza la IDFT siendo el tamaño de esta igual a 64, la IDFT se implementa en Matlab con la función *ifft*. Una vez aplicada la IDFT se obtiene **4 secuencias cortas** (64 muestras) en el dominio del tiempo. Como el objetivo es conseguir 10 secuencias cortas, primero se copia la última de las 4 secuencias al inicio (como si fuera un prefijo cíclico), dando así un total de 5 secuencias y luego se duplica estas 5 secuencias dando en total las **10 secuencias cortas** en el dominio del tiempo.

Tabla 2.13. Valores reales e imaginarios de 64 muestras (4 secuencias cortas) en el dominio de la frecuencia [34].

N°.	Valor		N°.	Valor		N°.	Valor	
	Real	Imag		Real	Imag		Real	Imag
1	0	0	23	0	0	45	-1.4720	-1.4720
2	0	0	24	0	0	46	0	0
3	0	0	25	1.4720	1.4720	47	0	0
4	0	0	26	0	0	48	0	0
5	-1.4720	-1.4720	27	0	0	49	1.4720	1.4720
6	0	0	28	0	0	50	0	0
7	0	0	29	0	0	51	0	0
8	0	0	30	0	0	52	0	0
9	-1.4720	-1.4720	31	0	0	53	-1.4720	-1.4720
10	0	0	32	0	0	54	0	0
11	0	0	33	0	0	55	0	0
12	0	0	34	0	0	56	0	0
13	1.4720	1.4720	35	0	0	57	-1.4720	-1.4720
14	0	0	36	0	0	58	0	0
15	0	0	37	0	0	59	0	0
16	0	0	38	0	0	60	0	0
17	1.4720	1.4720	39	0	0	61	1.4720	1.4720
18	0	0	40	0	0	62	0	0
19	0	0	41	1.4720	1.4720	63	0	0
20	0	0	42	0	0	64	0	0
21	1.4720	1.4720	43	0	0			
22	0	0	44	0	0			

El propósito de la sección LSTF o sección corta, es que en recepción se pueda detectar si la trama recibida pertenece a una trama OFDM. La sección corta también permite al receptor la estimación del desplazamiento de frecuencia de la portadora (CFO) [4]. En la Figura 2.96 se muestra la parte real y parte imaginaria, en el dominio del tiempo, de la sección corta compuesta por 10 secuencias. Las secuencias están numeradas del 1 al 10 y acotadas, tanto en la parte real como imaginaria. Además, la primera secuencia corta se identifica con color rojo para facilidad de visualización.

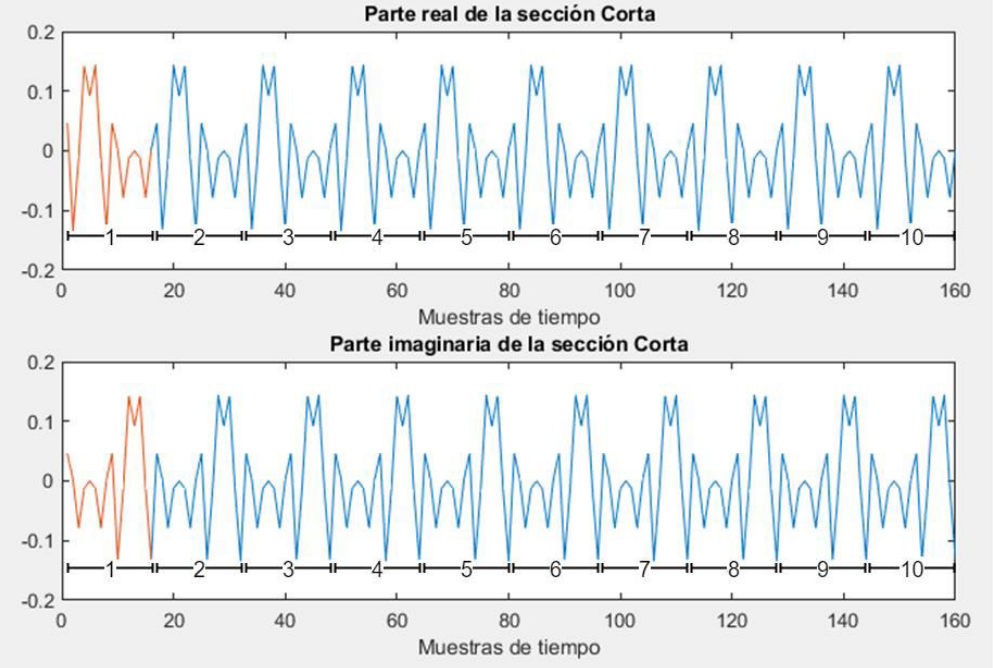


Figura 2.96. Sección corta (LSTF) en el dominio del tiempo.

La **sección larga o LLTF** se compone de **2 secuencias largas de 64 muestras** cada una y de un prefijo cíclico de longitud 32.

Para obtener una secuencia larga en el **dominio del tiempo**, primero se la define en el dominio de la frecuencia. En la Tabla 2.14 se puede ver los componentes reales e imaginarios para los 64 valores que conforman una secuencia larga, en el dominio de la frecuencia. Para pasar los valores del dominio de la frecuencia al dominio del tiempo se utiliza la IDFT que en Matlab se implementa con la función *ifft*. Una vez aplicada la IDFT se obtiene **1 secuenica larga** (64 muestras) en el dominio del tiempo. A partir de esta secuencia en tiempo se crea la sección LLTF en tiempo, primero con un prefijo cíclico (de la secuencia larga) de longitud 32, seguido de dos secuneicas largas.

Tabla 2.14. Valores reales e imaginarios de 1 secuencia corta (64 muestras) en el dominio de la frecuencia [34].

N°.	Valor		N°.	Valor		N°.	Valor	
	Real	Imag		Real	Imag		Real	Imag
1	0	0	23	-1	0	45	-1	0
2	1	0	24	1	0	46	1	0
3	-1	0	25	1	0	47	-1	0
4	-1	0	26	1	0	48	1	0
5	1	0	27	1	0	49	1	0
6	1	0	28	0	0	50	1	0
7	-1	0	29	0	0	51	1	0
8	1	0	30	0	0	52	1	0
9	-1	0	31	0	0	53	1	0
10	1	0	32	0	0	54	-1	0
11	-1	0	33	0	0	55	-1	0
12	-1	0	34	0	0	56	1	0
13	-1	0	35	0	0	57	1	0
14	-1	0	36	0	0	58	-1	0
15	-1	0	37	0	0	59	1	0
16	1	0	38	0	0	60	-1	0
17	1	0	39	1	0	61	1	0
18	-1	0	40	1	0	62	1	0
19	-1	0	41	-1	0	63	1	0
20	1	0	42	-1	0	64	1	0
21	-1	0	43	1	0			
22	1	0	44	1	0			

El propósito de la sección LLTF es identificar el inicio exacto del paquete OFDM y eliminar las distorsiones de fase residual [4]. La Figura 2.97 muestra en el dominio del tiempo la parte real y parte imaginaria de la sección larga del preámbulo 802.11a. La primera secuencia larga de longitud igual a 64 se identifica con color rojo y la segunda secuencia larga se ubica seguido de la primera. El prefijo cíclico se ubica al inicio y tiene una longitud de 32 muestras de tiempo, las cuales se copian del final del segundo símbolo.

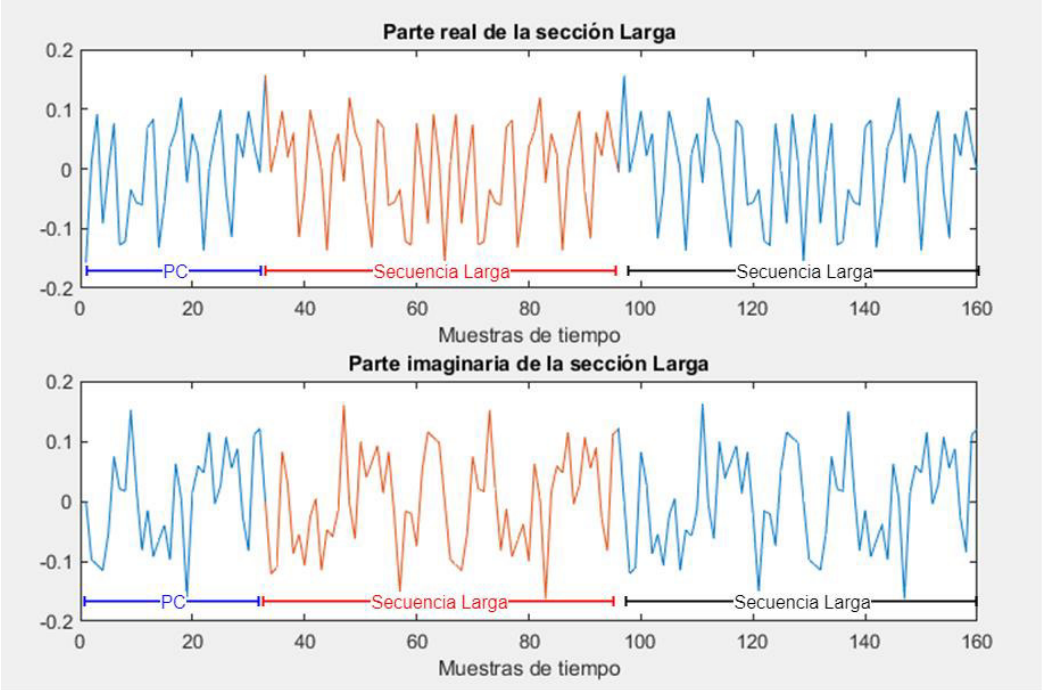


Figura 2.97. Sección larga (LLTF) en el dominio del tiempo.

2.3.7.2. Implementación en Matlab

La implementación en Matlab de la **etapa Adición preámbulo 802.11a** se realiza de forma manual sin utilizar funciones directas. Un ejemplo de creación del preámbulo 802.11a se implementa en un script llamado *preambuloOFDM.m*. Este script inicia por definir en el dominio de la frecuencia 4 secuencias cortas, que se almacenan en el vector *parteCorta64*, este vector tiene una longitud de 64. Luego se define en el dominio de la frecuencia una secuencia larga que se almacena en el vector *parteLarga*, este vector tiene una longitud igual a 64.

Inicialmente se utilizan valores en el dominio de la frecuencia, para luego obtener los valores en el dominio del tiempo, mediante la función de Matlab *ifft*.

El vector *parteCorta64* se transforma al dominio del tiempo y se almacena en el vector *parteCorta64Time*. Luego se toman las últimas 16 muestras del vector *parteCorta64Time* y se almacena al inicio del vector *parteCorta80Time*, también al final de este último vector se almacena todo el vector *parteCorta64Time*. El vector *parteCorta80Time* tiene una longitud de 80 y contiene 5 secuencias cortas en el dominio de la frecuencia. Después en un vector llamado *preambulo* se almacena 2 veces el vector *parteCorta80Time*, tal que hasta este punto el vector *preambulo* tiene una longitud de 160 y almacena 10 secuencias cortas, es decir la parte LSTF del preámbulo.

Luego se transforma el vector *parteLarga* al dominio del tiempo y se almacena en un nuevo vector llamado *parteLargaTime*. En el vector *preambulo* a partir de la posición 161 se copian las últimas 32 muestras del vector *parteLargaTime*, seguido de dos veces el vector *parteLargaTime*. Entonces, el vector *preambulo* resulta tener una longitud total de 320, en donde desde la posición 161 a la 320, se encuentra la parte LLTF del preámbulo 802.11a.

Finalmente, se grafica la parte real y parte imaginaria del vector *preambulo*, en la gráfica se colorea con rojo la parte LSTF y con azul la parte LLTF. El código descrito en estos últimos párrafos y correspondiente al script *preambuloOFDM.m*, se presenta a continuación debidamente comentado.

```
% Creación del preámbulo 802.11a
% Se inicia por definir parte de la sección corta y larga en el dominio
% de la frecuencia para luego obtener el preambulo en el dominio del
% tiempo:
% 4 secuencias cortas (parte de LSTF) en el dominio de la frecuencia
parteCorta64=[0;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0];
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y creación de la
% sección corta completa (LSTF):
parteCorta64Time=ifft(parteCorta64); % pasar al dominio del tiempo
parteCorta80Time=parteCorta64Time(49:64);% copiar las ultimas
% 16 muestras al inicio de la variable
parteCorta80Time(17:80)=parteCorta64Time; % 5 secuencias cortas
preambulo=parteCorta80Time; % copiar las 5 secuencias cortas
preambulo(81:160)=parteCorta80Time; % añadir las otras 5 secuencias
% cortas para obtener la sección corta (LSTF)
% Conversión del dominio de la frecuencia al tiempo y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo la
% secuencia larga
preambulo(161:192)=parteLargaTime(33:64); % PC en el preámbulo
preambulo(193:256)=parteLargaTime; % primera secuencia larga
preambulo(257:320)=parteLargaTime; % segunda secuecia larga y se
obtiene
% el preámbulo completo.
% Graficar el preámmbulo 802.11a en el dominio del tiempo:
figure()
subplot(2,1,1)
plot(real(preambulo))
hold on
plot(real(preambulo(1:160)))
title('Parte real del preámbulo 802.11a')
```

```
xlabel('Muestras de tiempo')
legend('LLTF','LSTF')
xlim([0 380])
hold off
subplot(2,1,2)
plot(imag(preambulo))
hold on
plot(imag(preambulo(1:160)))
title('Parte imaginaria del preámbulo 802.11a')
xlabel('Muestras de tiempo')
legend('LLTF','LSTF')
xlim([0 380])
hold off
grid off
```

Como resultado de ejecutar la sección de código anterior se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.98 y la gráfica de la Figura 2.99. El vector *preambulo* contiene a los datos que conforman el preámbulo 802.11a en el dominio del tiempo.

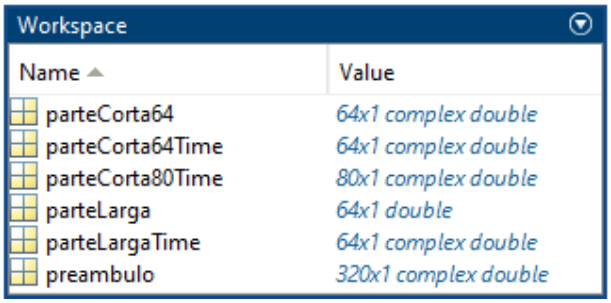


Figura 2.98. Variables en el espacio de trabajo luego de ejecutar *preambuloOFDM.m*.

En la Figura 2.99 se muestra la sección corta (LSTF) en color rojo y la sección larga (LLTF) en color azul.

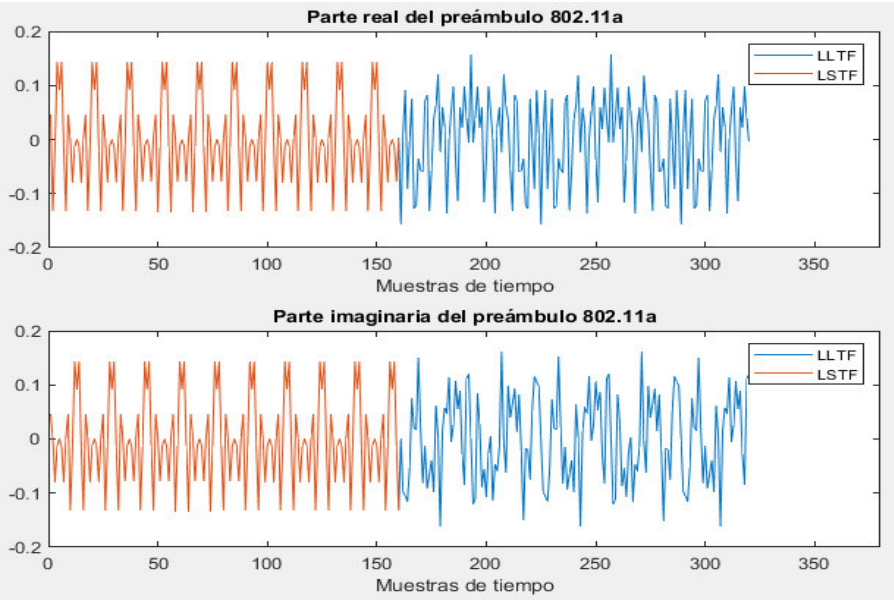


Figura 2.99 Gráfica del preámbulo 802.11a.

Una vez creado el preámbulo 802.11a se puede anteponer el mismo a los símbolos OFDM a transmitir, para lo cual se puede utilizar la siguiente sintaxis:

```
Datos=[preambulo;SimbolosOFDM];
```

La variable *preambulo* es un vector columna que contienen al preámbulo 802.11a en el dominio del tiempo, la variable *SimbolosOFDM* es un vector columna que contiene a los símbolos OFDM serializados. Por tanto, *Datos* es un vector columna que contiene el preámbulo junto con los símbolos OFDM.

El dispositivo SDR Adalm Pluto se configuró para que en recepción, recupere un tamaño fijo de bloque. Por esto también en transmisión, se opta por enviar un bloque de datos de tamaño fijo.

Seguido de los valores del vector *Datos* se debe realizar un relleno con ceros, hasta obtener un vector de tamaño igual al bloque que se va a enviar por el dispositivo Adalm Pluto. En este caso la manera más rápida y simple de obtener un relleno de ceros es; inicializar un vector columna con todos los valores en cero y de tamaño igual al bloque que utiliza el SDR y luego copiar el vector *Datos* al inicio del vector de ceros. La siguiente sintaxis sirve para implementar el relleno:

```
BloqueEnv=zeros(tamanioBloque,1);
[longDat,~]=size(Datos);
BloqueEnv(1:longDat,1)=Datos;
```

La variable *tamanioBloque* es un número escalar que especifica el tamaño del bloque a ser enviado por el equipo SDR Adalm Pluto. La variable *longDat* almacena la longitud del vector *Datos*. Como resultado se obtendrá el vector *BloqueEnv* que contiene la información del vector *Datos*, seguido de un relleno de ceros.

2.3.7.3. Adición del preámbulo 802.11a al archivo de texto procesado

En las etapas de transmisión previas se procesaron los datos de un archivo de texto hasta crear símbolos OFDM y serializarlos. Para implementar la etapa actual es necesario ejecutar los scripts correspondientes a cada una de las etapas anteriores. Los scripts que se deben ejecutar previamente son: *abrir_archivo.m*, *serializar_datos.m*, *cod_conv.m*, *mapeo_datos.m*, *reformado_tx.m* y *ofdm_tx.m*. En la Figura 2.100 se identifican con color gris las etapas previas y con color anaranjado la etapa actual.

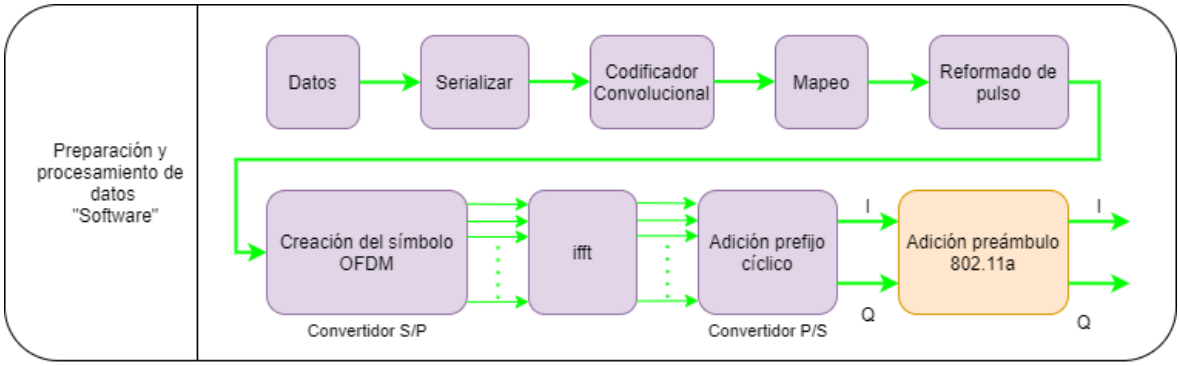


Figura 2.100. Etapas de transmisión con la etapa Adición preámbulo 802.11a identificada.

Una vez ejecutados los scripts mencionados y seleccionado el archivo de texto *ArchivoTextoRT.txt* se debe borrar todas las variables del espacio de trabajo a excepción del vector *datosEnviar*. Dicho vector contiene a los símbolos OFDM serializados. La siguiente sección de código permite borrar todas las variables del espacio de trabajo a excepción de *datosEnviar*.

```
clearvars -except datosEnviar  % Borrar todas las variables excepto
% la variable datosEnviar
```

La etapa Adición preámbulo 802.11a se implementa en un script llamado *preambulo_tx.m*. El código del script inicia por definir las 4 secuencias cortas y una secuencia larga en el dominio de la frecuencia. Luego se crea la sección corta y sección larga en el dominio del tiempo y se almacena en el vector *preámbulo*. Este proceso se describió a mayor detalle previamente.

Después se almacena el preámbulo 802.11a y los símbolos OFDM del vector *datosEnviar*, dentro de un nuevo vector llamado *datosConPre*. Luego se define el tamaño del bloque (tamaño de trama) a ser enviado por el equipo SDR Adalm Pluto y se almacena en la variable *tamanioBloque*. El valor que se asigna es de 800000 muestras por trama. Esto debido a que en el **ejemplo de configuración de equipos** del script *config_SDR.m*, se configuró el parámetro *SamplesPerFrame* del objeto de recepción en 800000 y el tamaño de trama recibido debe corresponder al enviado.

Luego se inicializa un vector tipo columna con ceros, que tiene una longitud igual al tamaño del bloque a ser enviado y toma el nombre de *auxDat*. Después se copia el vector *datosConPre* en las posiciones iniciales del vector *auxDat*. De tal forma que el vector *auxDat* contiene al preámbulo 802.11a, a los símbolos OFDM serializados, un relleno de ceros y mantiene una longitud de 800000. Luego se copia los datos de *auxDat* en un vector llamado *datosEnviarBloque*.

Finalmente, se grafica el vector *datosEnviarBloque* completo y también los primeros 500 datos del mismo vector para identificar el preámbulo y el inicio de la carga útil. El código descrito en estos párrafos se presenta debidamente comentado a continuación.

```
% Script para añadir el preámbulo 802.11a a los símbolos OFDM creados
% El script corresponde a la etapa Adición preámbulo 802.11a
% Previamente a ejecutar el script actual es necesario ejecutar en
% orden los scripts: abrir_archivo.m, serializar_datos.m, cod_conv.m,
% mapeo_datos.m, reformado_tx.m y ofdm_tx.m
% Como argumento de entrada del script actual se tiene un vector
columna
% que contiene símbolos OFDM serializados, el vector tiene el nombre
% de datosEnviar y se obtiene del script ofdm_tx.m
% Se inicia por definir parte de la sección corta y larga en el dominio
% de la frecuencia para luego obtener el preambulo en el dominio del
% tiempo:
% 4 secuenias cortas(parte de LSTF) en el dominio de la frecuencia
parteCorta64=[0;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;1.4720+1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    1.4720+1.4720i;0;0;0;-1.4720-1.4720i;0;0;0
    -1.4720-1.4720i;0;0;0;1.4720+1.4720i;0;0;0];
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y creación de la
% sección corta completa (LSTF):
parteCorta64Time=ifft(parteCorta64); % pasar al dominio del tiempo
parteCorta80Time=parteCorta64Time(49:64);% copiar las ultimas
% 16 muestras al inicio de la variable
parteCorta80Time(17:80)=parteCorta64Time; % 5 secuencias cortas
preambulo=parteCorta80Time; % copiar las 5 secuencias cortas
preambulo(81:160)=parteCorta80Time; % añadir las otras 5 secuencias
% cortas para obtener la sección corta (LSTF), es decir las 10
% secuencias cortas
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo la
% secuencia larga
preambulo(161:192)=parteLargaTime(33:64); % PC en el preámbulo
preambulo(193:256)=parteLargaTime; % primera secuencia larga
preambulo(257:320)=parteLargaTime; % segunda secuecia larga y se
obtiene
% el preámbulo completo.
% Unir el preambulo a los símbolos OFDM serializados:
datosConPre=[preambulo;datosEnviar]; % Preambulo con símbolos OFDM
% creados previamente
% Inicializar con ceros una variable auxiliar, que tenga el tamaño del
% bloque a enviar:
tamanioBloque=800000; % tamaño del bloque a transmitir
auxDat=zeros(tamanioBloque,1); % inicializar variable
[longDat,~]=size(datosConPre); % longitud de los datos a enviar
auxDat(1:longDat,1)=datosConPre; % copiar el preambulo con los símbolos
```

```
% OFDM y los demás datos del bloque se mantienen en cero
datosEnviarBloque=auxDat;% Datos de salida a enviar
% Graficar la parte real del bloque a transmitir y la parte real de los
% primeros 500 datos del bloque a transmitir, tal que el preambulo
% queda en color rojo para poder identificarlo:
figure()
subplot(2,1,1)
plot(real(datosEnviarBloque))
title('Bloque a enviar (parte real)')
subplot(2,1,2)
plot(real(datosEnviarBloque(1:500)))
hold on
plot(real(datosEnviarBloque(1:320)))
title(' Primeras muestras del bloque a enviar (parte real)')
legend('Simbolos OFDM','Preámbulo 802.11a')
xlim([0 660])
hold off
```

Una vez ejecutadas las secciones de código anteriores se obtiene en el espacio de trabajo las variables que se muestran en la Figura 2.101.

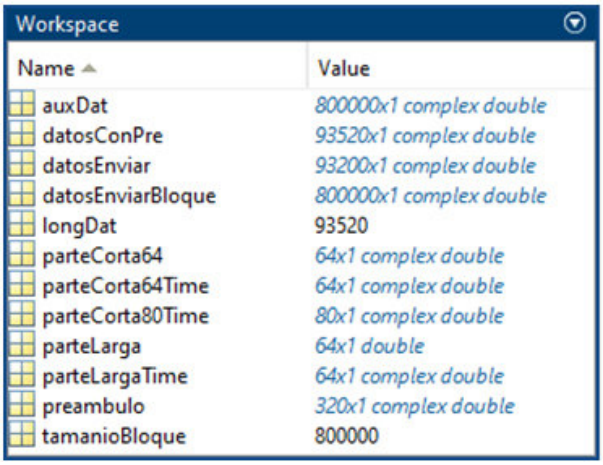


Figura 2.101. Variables en el espacio de trabajo luego de ejecutar el script *preambulo_tx.m*.

En la Figura 2.102 se muestra la parte real de los datos de la variable *datosEnviarBloque*. También se muestran los primeros datos de la variable mencionada, en donde se identifica con rojo el preámbulo 802.11a y con azul el inicio de la carga útil.

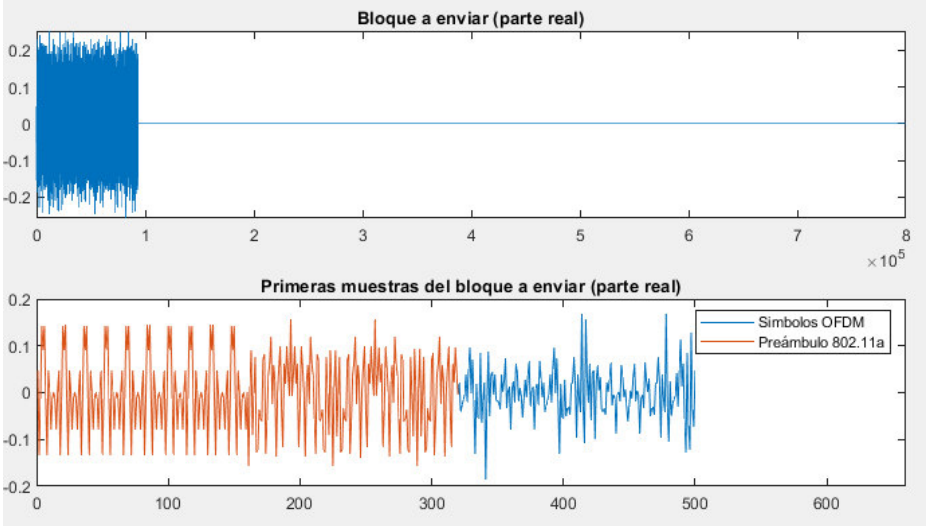


Figura 2.102. Parte real del bloque a enviar.

2.3.8. ETAPA DE TRANSMISIÓN DE DATOS

La transmisión de datos consiste en enviar los datos procesados al canal inalámbrico utilizando un dispositivo SDR Adalm Pluto. La base teórica sobre los equipos Adalm Pluto y su configuración se abordaron en la sección 2 de este capítulo. En esta sección también se implementó el script llamado *config_SDR.m*, en donde se configuran los objetos de transmisión y recepción.

Para transmitir una señal de datos desde un dispositivo SDR Adalm Pluto, el cual se representa en Matlab por un objeto de sistema *comm.SDRTxPluto*, se utiliza la sintaxis:

```
txAdalm(data)
```

En el cuadro anterior *txAdalm()* corresponde al nombre del objeto del sistema configurado previamente con la función *sdrtx()*. El argumento de entrada *data* es un vector columna que contiene los datos que se quieren transmitir. Los datos válidos a ser transmitidos por el equipo SDR Adalm Pluto pueden ser [10]:

- **Tipo double** con valores escalares entre -1 y 1. Este tipo de datos utiliza 64 bits para representar cada valor.
- **Tipo single** con valores escalares entre -1 y 1. Este tipo de datos utiliza 32 bits para representar cada valor.
- **Tipo int16** con valores entre -32768 y 32767, pero los datos transmitidos por la radio ADALM-PLUTO pierden los 4 bits menos significativos. Este tipo de datos utiliza 16 bits para representar cada muestra.

Para mostrar un ejemplo del envío de los datos al canal inalámbrico se va a seguir con el ejemplo del archivo de texto que se ha venido procesando en las etapas anteriores.

2.3.8.1. Transmisión del archivo de texto procesado

La transmisión de datos del archivo de texto procesado se implementa dentro de un script llamado *transmitir.m*. Antes de ejecutar este script es necesario ejecutar los scripts creados en etapas anteriores, estos son; *abrir_archivo.m*, *serializar_datos.m*, *cod_conv.m*, *mapeo_datos.m*, *reformado_tx.m*, *ofdm_tx.m* y *preambulo_tx.m*. Luego se debe ejecutar la siguiente línea de código para eliminar las variables del espacio de trabajo, a excepción de la variable *datosEnviarBloque*.

```
clearvars -except datosEnviarBloque % Borrar todas las variables
% excepto la variable datosEnviarBloque
```

La variable *datosEnviarBloque* se obtiene del script *preambulo_tx.m* y contiene los datos procesados junto con el preámbulo 802.11a y un relleno para ajustar el tamaño de bloque fijo de 800 000. Esta variable se utiliza como entrada del script *transmitir.m*

Luego, con un dispositivo SDR Adalm Pluto conectado al ordenador, se debe ejecutar el script *config_SDR.m*, creado y descrito en la sección 2.2.3 de este documento. Una vez ejecutado dicho script, en el espacio de trabajo quedan las variables que se muestran en la Figura 2.103.

Figura 2.103. Variables en el espacio de trabajo luego de ejecutar *confid_SDR.m*.

El objeto *txAdalm* contiene el objeto del sistema *comm.SDRTxPluto* que comunica el equipo SDR con Matlab y se utiliza para la transmisión de datos. Mientras que el objeto *rxAdalm* contiene el objeto del sistema *comm.SDRRxPluto* que comunica el equipo SDR con Matlab y se utiliza para la recepción de datos. En la Figura 2.104 se muestra características de *txAdalm* y *rxAdalm*, que es parte del resultado de la ventana de comandos luego de ejecutar *config_SDR.m*.

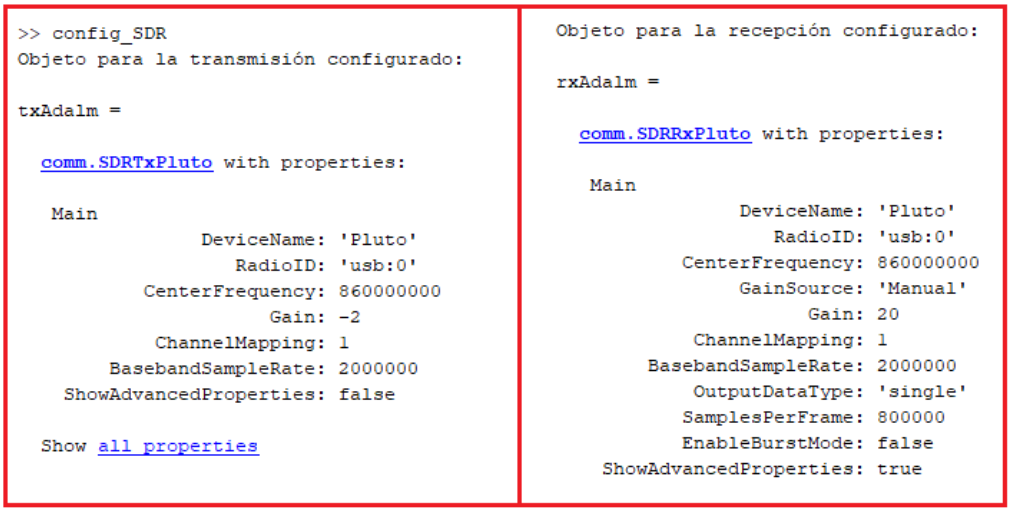


Figura 2.104. Características de los objetos *txAdalm* y *rxAdalm*.

En la parte de configuración se estableció que los datos a enviar deben ser del **tipo single**, que utilizan 32 bits para representar cada muestra. El rango de valores que acepta este tipo de datos va de -1 a 1.

El script *transmitir.m* **inicia por normalizar los datos del bloque** que se va a transmitir, contenidos dentro de la variable *datosEnviarBloque*. Para esto se encuentra el máximo valor absoluto de la parte real y de la parte imaginaria (por separado). Luego con un condicional *if* se discrimina si el máximo valor está en la parte real o la parte imaginaria. Cuando ya se encuentre el máximo valor, se normaliza los datos dividiendo cada elemento del vector de entrada *datosEnviarBloque* para el valor máximo encontrado y la respuesta se almacena en una variable auxiliar llamada *aux_TxNorm*.

Luego se transforma la variable *aux_TxNorm* del tipo **double** al tipo **single** y se almacena en una variable llamada *auxTx*. En la parte de configuración se estableció que los datos a enviar deben ser del tipo **single**, que utilizan **32 bits** para representar cada muestra. Después utilizando el objeto *txAdalm* se envían los datos con el equipo SDR al canal inalámbrico. Luego se muestra un mensaje para indicar que se han transmitido lo datos. Finalmente se grafica el bloque de datos sin normalizar y normalizados.

El código descrito en estos dos párrafos, correspondiente al script *transmitir.m*, se presente a continuación debidamente comentado.

```
% Script para mostrar un ejemplo del envío de datos de la etapa
% Transmitir Datos
% Previamente a ejecutar el script actual es necesario ejecutar en
% orden los scripts; abrir_archivo.m, serializar_datos.m,
% cod_conv.m, mapeo_datos.m, reformado_tx.m, ofdm_tx.m y
% preambulo_tx.m. Además se debe ejecutar config_SDR.m.
% Como argumento de entrada del script actual se tiene un vector
% columna obtenido del script preambulo_tx.m.
```

```
% Normalizar los datos a transmitir:
% Se evalua la parte real y la parte imaginaria de los vectores, para
% encontrar el módulo máximo de la parte real e imaginaria.
v_maxR=max(abs(real(datosEnviarBloque))); % Valor máximo, parte real
v_maxI=max(abs(imag(datosEnviarBloque))); % Valor máximo, parte imag
% Con un condicional se evalúa si la parte imaginaria o la parte real
% tiene el valor máximo para normalizar los datos
if(v_maxR<v_maxI)
    valorMax=v_maxI; % La parte imaginaria tiene el máximo valor abs
else
    valorMax=v_maxR; % La parte real tiene el máximo valor absoluto
end
% Normalizo y almaceno los datos a transmitir en una variable auxiliar
aux_TxNorm=datosEnviarBloque/valorMax;
% Convertimos los datos a un tipo single que emplea 32 bits, mientras
% que double emplea 64 bits para la representacion de cada valor:
auxTx=single(aux_TxNorm);
% Se transmiten los datos Normalizados:
txAdalm(auxTx); % Transmisión de datos
disp('Datos transmitidos') % Mostramos un mensaje al usuario
% Graficar datos de entrada de la etapa y los datos Normalizados
figure()
subplot(2,2,1)
plot(real(datosEnviarBloque))
title('Bloque de datos sin normalizar (parte real)')
subplot(2,2,3)
plot(imag(datosEnviarBloque))
title('Bloque de datos sin normalizar (parte imaginaria)')
subplot(2,2,2)
plot(real(aux_TxNorm),'g')
title('Bloque de datos normalizados (parte real)')
subplot(2,2,4)
plot(imag(aux_TxNorm),'g')
title('Bloque de datos normalizados (parte imaginaria)')
```

Luego de ejecutar el script *transmitir.m* se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.105 y la gráfica que se muestra en la Figura 2.106. Además, los datos normalizados de la variable *auxTx* se transmiten al canal inalámbrico a través del dispositivo SDR Adalm Pluto.

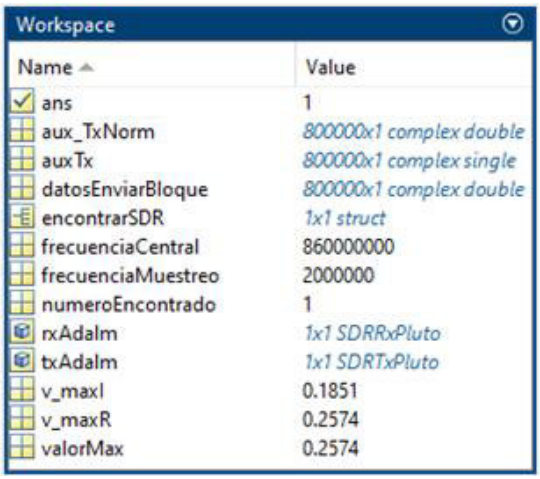


Figura 2.105. Variables en el espacio de trabajo luego de ejecutar el script *transmitir.m*.

En la Figura 2.106 se muestra al lado izquierdo, la parte real y parte imaginaria de los datos que contiene el vector *datosEnviarBloque*, que son los datos sin normalizar. Al lado derecho se muestra la parte real y parte imaginaria de los datos normalizados, es decir que los datos que se envían por el canal inalámbrico.

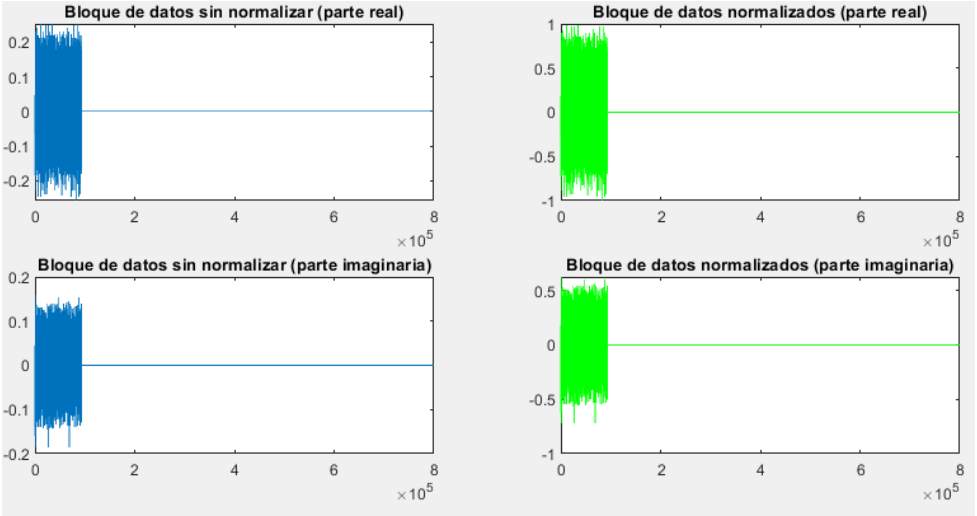


Figura 2.106. Bloques de datos a enviar sin normalizar y normalizados (parte real y parte imaginaria).

En la sección correspondiente a la recepción se analizará el cómo recibir los datos mediante el dispositivo SDR Adalm Pluto.

2.4. CANAL INALÁMBRICO

El canal inalámbrico es un medio no guiado por el cual se propagan ondas electromagnéticas. En una comunicación inalámbrica el transmisor envía al receptor información a través de este canal, utilizando ondas electromagnéticas. El canal inalámbrico es dinámico e impredecible lo que dificulta el análisis de un sistema de comunicación inalámbrica [2]. En el prototipo de comunicaciones que se está desarrollando, el canal inalámbrico es el medio de transmisión por el cual se propaga la señal transmitida por el dispositivo SDR Adalm Pluto.

Debido a la naturaleza del canal inalámbrico resulta imposible mantener las mismas condiciones de prueba durante un intervalo de tiempo prolongado. Por esto, actualmente existen modelos que permiten estimar el comportamiento del canal inalámbrico. Una característica típica del canal inalámbrico es el fenómeno del desvanecimiento, es decir que la amplitud de la señal fluctúa en el tiempo y en la frecuencia [36]. Existen dos tipos de modelado que son; el modelo de desvanecimiento a gran escala y en el modelo del desvanecimiento a pequeña escala.

El desvanecimiento a gran escala ocurre cuando los equipos de transmisión y de recepción están a una gran distancia [36]. Este desvanecimiento se divide en un modelo de pérdida general y el modelo Okummuera/Hata. A su vez el modelo de pérdida general se divide en; modelo de espacio libre (free space path loss model), modelo log distance (log distance path loss model) y modelo lognormal (lognormal shadow path loss model). En el presente trabajo no se considera variación de distancia entre Tx y Rx en tiempo real y por ellos estos fenómenos no son considerados.

El desvanecimiento a pequeña escala ocurre cuando los equipos de transmisor y el receptor se encuentran a distancias cortas. Este desvanecimiento se atribuye a la propagación de múltiples rutas (multitrayecto), la velocidad de los objetos circundantes, la velocidad de estaciones móviles y el ancho de banda de transmisión de la señal [2]. El desvanecimiento a pequeña escala se puede describir mediante un canal selectivo en frecuencia [36]. El multitrayecto incurre en la interferencia constructiva o destructiva de la señal y en fluctuaciones rápidas del nivel de la señal. Entre los modelos estadísticos que mejor se ajustan a los desvanecimientos a pequeña escala se encuentran el desvanecimiento Rayleigh y desvanecimiento Rice.

El canal inalámbrico introduce efectos no deseados en las ondas de radio que pasan a través de este. Entre los efectos que presenta el canal inalámbrico están: el desplazamiento en frecuencia de las portadoras (CFO), el desplazamiento en fase y la introducción de ruido aditivo gaussiano. Estos efectos generan que al recuperar los datos del canal inalámbrico exista una tasa de error. En recepción se corrige el desplazamiento en frecuencia y el desplazamiento en fase para reducir la tasa de error en los datos recibidos.

2.4.1. RUIDO BLANCO GAUSSIANO

El ruido blanco gaussiano está presente en el entorno y se compone de contribuciones aleatorias de diferentes fuentes, lo que desemboca en que este ruido tenga una distribución de probabilidad gaussiana o normal [16]. En los diagramas de constelación de los datos mapeados, el ruido blanco provoca que los símbolos se dispersen alrededor de la posisión original. En un script llamado *EjemploCanal.m* en la primera sección se implementa un ejemplo del efecto del ruido blanco gaussiano sobre datos mapeados con QPSK.

El código inicia por por limpiar las variables del espacio de trabajo e Matlab y cerrar todas las figuras. Luego se define los símbolos de entrada al modulador, se mapean con QPSK con la función *pskmod* y se normalizan los datos mapeados. Después se añade ruido blanco gaussiano a los datos QPSK normalizados mediante la función ***awgn***. Esta función

permite añadir ruido blanco gaussiano a una señal de entrada en base a un valor de relación señal al ruido (SNR) en dB. En este caso se utiliza una SNR de 20 dB. El vector que contiene los datos mapeados con QPSK, normalizados y con ruido se llama *datosMapeo*. Finalmente, se grafica el diagrama de constelación de los datos mapeados con QPSK con y sin ruido. El código descrito en este párrafo y correspondiente a la primera sección del script *EjemploCanal.m* se presenta a continuación.

```
% Ejemplo de las alteraciones del canal inalámbrico
clear; close all;
% RUIDO ADITIVO GAUSSIANO
% Datos de entrada:
datos=randi([0 3],256,1); % Simbolos de entrada
% Mapeo QPSK:
M = 4; % Puntos en la constelación (símbolos)
datosM = pskmod(datos,M,pi/4,'gray'); % Datos modulados QPSK
% Factor de normalización = 1/sqrt(2)
datosMapeados=(1/sqrt(2))*datosM; % Vector con datos modulados y norm
% Añadir ruido:
datosMapeo= awgn(datosMapeados,20,'measured'); % Señal con ruido
% Graficar:
figure()
plot(datosMapeo,'r.','MarkerSize',10); hold on;
plot(datosMapeados,'b.','MarkerSize',20); grid on;
title('Señal con ruido aditivo');
legend('Con ruido', 'Sin ruido');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-1 1 -1 1]);
```

El resultado de la sección de código anterior es la Figura 2.107, en donde se muestra el efecto que tiene el ruido blanco gaussiano sobre datos modulados y normalizados con QPSK. En el diagrama de constelación los datos con ruido (color rojo) se dispersan alrededor de los datos originales sin ruido (color azul).

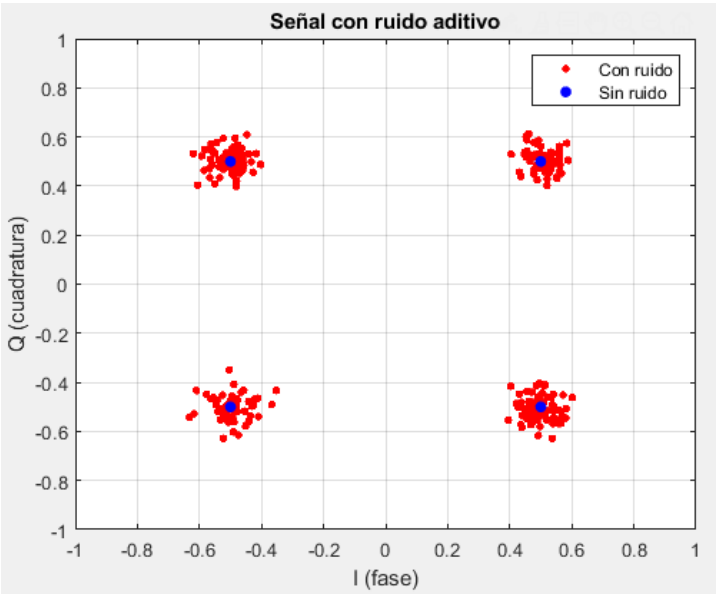


Figura 2.107. Diagrama de constelación de datos mapeados con QPSK con y sin ruido gaussiano.

2.4.2. DESPLAZAMIENTO EN FRECUENCIA

El desplazamiento de frecuencia de las portadoras (CFO) provoca que la ortogonalidad de las portadoras que pasan por el canal inalámbrico se pierdad. En OFDM cuando las portadoras dejan de ser otogonales, se produce interferencia entre portadoras (ICI). En Matlab el objeto del sistema *comm.PhaseFrequencyOffset* permite aplicar compensaciones de fase y frecuencia a una señal de entrada [37]. En esta sección se utiliza el objeto para insertar un desplazamiento en frecuencia.

Gracias al objeto *comm.PhaseFrequencyOffset* se puede simular un desplazamiento de fase y frecuencia a una señal de datos. En base a algoritmos se puede estimar estas alteriones, esto se lleva a cabo en recepción en la parte del procesamiento del preambulo.

La sintaxis del objeto es la siguiente:

corrPhFr = comm.PhaseFrequencyOffset(Name,Value)

En esta sintaxis, *Name* representa el nombre de la propiedad a configurar con el valor que toma el parámetro *Value*. El parámetro *corrPhFr* almacena el objeto que permite aplicar las compensaciones configuradas.

Las propiedades de entrada del objeto del sistema *comm.PhaseFrequencyOffset* se presentan a continuación [37]:

- **PhaseOffset:** Es el desplazamiento de fase especificado en grados y el valor por defecto es 0.
- **FrequencyOffset:** Es el desplazamiento de frecuencia especificado en Hz y el valor por defecto es de 0.
- **SampleRate:** Es la frecuencia de muestreo de la señal de entrada en Hz, se especifica como un escalar positivo real y el valor por defecto es 1. Este valor debe ser el mismo que se configuró en los dispositivos SDR Adalm Pluto, en el prototipo se utiliza 2Msps.

Para aplicar las compensaciones configuradas con las respectivas propiedades de utiliza la siguiente sintaxis:

seCorr=corrPhFr(seEnt)

La variable *seEnt* contiene a los datos de la señal de entrada a ser corregida y *seCorr* almacena la señal ya corregida.

En el script llamado *EjemploCanal.m* en la segunda sección se implementa un ejemplo del desplazamiento de frecuencia. El código inicia por definir la frecuencia de muestreo, esta es igual a la configurada en el equipo de radio Adalm Pluto. Luego con el objeto *comm.PhaseFrequencyOffset* se crea otro objeto llamado *senialDesf* que permite aplicar un desplazamiento de frecuencia de 750 Hz a la señal de entrada. En este caso el vector de entrada se lo obtiene de la sección 1 del script *EjemploCanal.m*, este vector se llama *datosMapeo* y contiene datos QPSK normalizados y con ruido blanco gaussiano. La señal desplazada en frecuencia se almacena en *datosSR*. Luego se grafica los datos de *datosMapeo* y *datosSR*, para visualizar el efecto del desplazamiento de frecuencia en el diagrama de constelación.

El código descrito en el párrafo anterior, correspondiente a la segunda sección del script *EjemploCanal.m* se presenta a continuación.

```
%% DESPLAZAR LOS DATOS EN FRECUENCIA
Fs=2.0e6; % frecuencia de muestreo
senialDesf = comm.PhaseFrequencyOffset('FrequencyOffset',750,...
    'SampleRate',Fs);
datosSR=senialDesf(datosMapeo); % Señal desplazada
% Graficar
figure()
plot(datosSR,'r.','MarkerSize',10); hold on;
plot(datosMapeo,'b.','MarkerSize',10); grid on;
title('Señal desplazada en frecuencia');
legend('Desplazada','Original');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-1 1 -1 1]);
```

En la Figura 2.108 se muestra el efecto que tiene el desplazamiento de frecuencia (color rojo) sobre los datos con QPSK con ruido aditivo gaussiano (color azul), esto en el diagrama de constelación. Se observa que un desplazamiento de frecuencia positivo la dispersión de datos se da en sentido antihorario. Por el contrario, si se tuviera un valor negativo de desplazamiento de frecuencia, los datos se dispersarían en sentido antihorario.

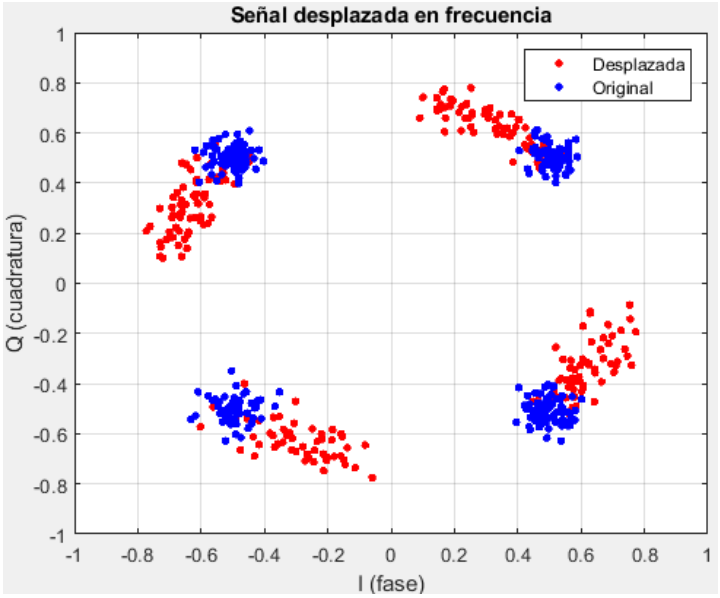


Figura 2.108. Diagrama de constelación de datos mapeados con QPSK y con ruido (color azul) vs. Los mismos datos desplazados en frecuencia (color rojo).

2.4.3. DESPLAZAMIENTO EN FASE

El desplazamiento en fase, desde el punto de vista de los diagramas de constelación, provoca que los datos se desplacen un cierto ángulo. Si el desplazamiento en fase es positivo, los datos se desplazan en sentido antihorario y si es negativo, los datos se dispersan en sentido horario. En esta sección se utiliza el objeto *comm.PhaseFrequencyOffset* para insertar un desplazamiento en fase. En una tercera sección del script *EjemploCanal.m* se implementa un ejemplo del desplazamiento en fase.

El códido inicia por definir la frecuencia de muestreo que se toma un valor igual a la configurada en el equipo de radio Adalm Pluto. Luego con el objeto *comm.PhaseFrequencyOffset* se crea otro objeto llamado *DesfSenial* que permite aplicar un desplazamiento de frecuencia de 30° a la señal de entrada. El vector de entrada se lo obtiene de la sección 1 del script *EjemploCanal.m*, este vector se llama *datosMapeo* y contiene datos QPSK, normalizados y con ruido blanco gaussiano. La señal desplazada en frecuencia se almacena en *datosDesfase*. Luego se grafica los datos de *datosMapeo* y *datosDesfase*, para visualizar el efecto del desplazamiento de fase en el diagrama de constelación.

El código descrito en el párrafo anterior, correspondiente a la tercera sección del script *EjemploCanal.m* se presenta a continuación.

```
%% DESPLAZAR EN FASE
Fs=2.0e6; % frecuencia de muestreo
```

```
DesfSenial = comm.PhaseFrequencyOffset('PhaseOffset',30,...
    'SampleRate',Fs);
datosDesfase=DesfSenial(datosMapeo); % Aplicar el desfase a datosMapeo
% Graficar
figure()
plot(datosMapeo,'r.','MarkerSize',10); hold on;
plot(datosDesfase,'b.','MarkerSize',10); grid on;
title('Señal desplazada en fase');
legend('Original', 'Desplazada');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-1 1 -1 1]);
```

Cuando se ejecuta la sección anterior de código, se obtiene los resultados que se muestran en la Figura 2.109, en donde se observa el efecto que tiene el desplazamiento de fase sobre los datos con QPSK con ruido aditivo gaussiano. Se ha introducido un desplazamiento en fase igual a 30°, en la figura se señala el desplazamiento para un dato en específico, pero esto aplica para cada uno de los datos de la constelación.

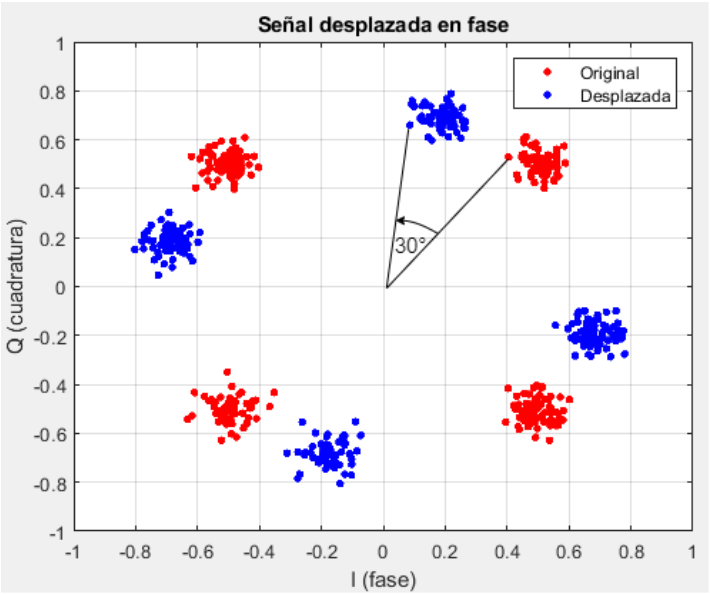


Figura 2.109. Diagrama de constelación de datos mapeados con QPSK y desplazados en fase de 30 grados.

2.5. ETAPAS DE RECEPCIÓN

En esta sección se aborda en detalle cada una de las etapas necesarias para recibir y procesar los datos recuperados del canal inalámbrico. Los datos enviados por el dispositivo SDR de transmisión se reciben con otro equipo SDR Adalm Pluto. Luego de procesar los datos del canal inalámbrico se espera recuperar los caracteres del archivo de texto enviado. En la Figura 2.110 se muestran mediante un diagrama de bloques las etapas de recepción.

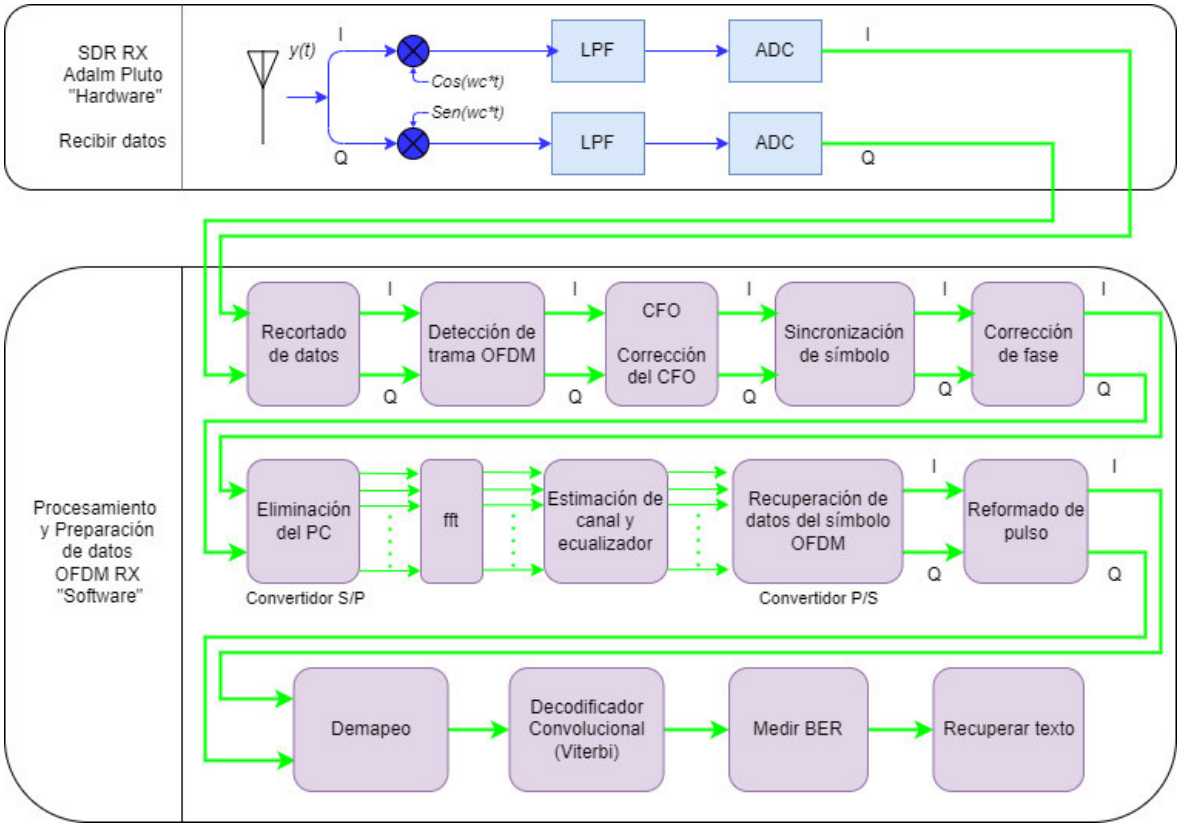


Figura 2.110. Etapas en el receptor.

Las etapas de recepción se describen brevemente a continuación:

1) Recibir datos. – Esta etapa recibe los datos del canal inalámbrico, utilizando un equipo SDR Adalm Pluto. Además, se utiliza el objeto global de recepción *rxAdalm* obtenido en la sección 1 de este capítulo. El canal se muestrea de manera repetitiva hasta que se detecte los datos transmitidos.
2) Recortado de datos. – En base a un **umbral** se recorta los datos recibidos para reducir la carga de procesamiento en las etapas posteriores.
3) Detección de trama OFDM. – Etapa que permite detectar si el bloque de datos pertenece a una secuencia OFDM.
4) Corrección de CFO. – Esta etapa permite estimar y corregir el desplazamiento de frecuencia de los datos recibidos.
5) Sincronización de símbolo. – Etapa en donde se determina el comienzo exacto de la trama OFDM, es decir que se realiza la sincronización en tiempo.
6) Corrección de fase. – En esta etapa se estima y se corrige el desplazamiento de fase de todo el bloque de datos que obtuvo en la etapa anterior.
7) Eliminación del prefijo cíclico (PC). – Se elimina las muestras que se agregaron en transmisión al inicio de cada símbolo OFDM.

8) FFT (Transformada Rápida de Fourier). – Etapa que se encarga de pasar los símbolos OFDM del dominio del tiempo al dominio de la frecuencia.
9) Estimación de canal y ecualizador. – En cada símbolo OFDM se realiza la estimación del canal a través de las portadoras piloto.
10) Recuperación de datos del símbolo OFDM. – Etapa en donde se realiza el desamblaje de los símbolos OFDM, es decir se eliminan las portadoras de guarda, las piloto y la DC, manteniendo solo las portadoras que contienen los datos.
11) Reformado de pulso. – En esta etapa se pasa los datos a través de un filtro de recepción SRRC. Esta etapa debe activarse solamente si en transmisión se activó la misma.
12) Demapeo. – Etapa que pasa los esquemas de modulación BPSK, QPSK, 16-QAM o 64-QAM a bits serializados.
13) Decodificador Convolucional (Viterbi). – Etapa que se utiliza para la corrección de errores a nivel de bit. En esta etapa se decodifica convolucionalmente los bits utilizando el método Viterbi. Esta etapa se debe activar o desactivar según se haya seleccionado en el transmisor.
14) Medición del BER. – Etapa que compara los bits serializados recibidos con los bits serializados del archivo original.
15) Decodificador. – Esta etapa se encarga de pasar los bits serializados a caracteres de texto, es decir que la información se habrá recuperado.

En las subsecciones posteriores se detalla paso a paso cada etapa de recepción. Se inicia por describir la idea principal de la etapa, en algunas etapas será necesario abordar una parte teórica, pero no en todos los casos debido a que en transmisión ya se abordó en detalle esta parte. También se realiza la implementación de ejemplos en Matlab, junto con los resultados parciales.

2.5.1. RECEPCIÓN Y RECORTADO DE DATOS

En esta sección se desarrolla la etapa Recepción de datos y la etapa Recortado de datos, estas etapas se identifican en la Figura 2.111.

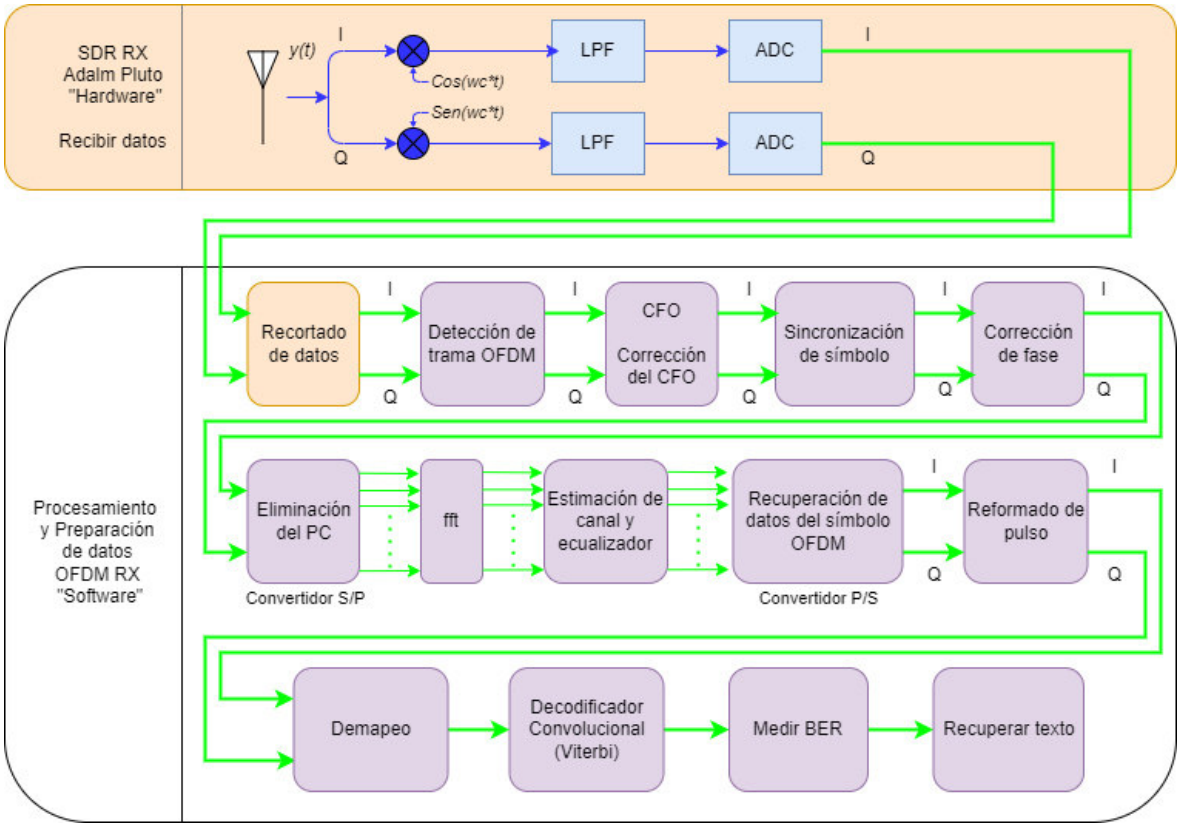


Figura 2.111. Etapas de recepción con la etapa Recepción de datos y la etapa Recortado de datos identificadas.

La etapa de **Recepción de datos** consiste en recuperar los datos del canal inalámbrico, utilizando un dispositivo SDR Adalm Pluto. Mientras el equipo SDR recepción sondea el canal el equipo de transmisión debe enviar los datos al canal.

La base teórica sobre los equipos Adalm Pluto se abarcó en la sección 2.2 de este documento. Dentro de dicha sección se implementó el script llamado *config_SDR.m*, en donde se configuran los objetos de transmisión y recepción. Para recibir una señal de datos mediante un dispositivo SDR Adalm Pluto, el cual se representa en Matlab por un objeto de sistema *comm.SDRRxPluto*, se utiliza la sintaxis:

datosRx=rxAdalm();

En el cuadro anterior *rxAdalm()* corresponde al nombre del objeto del sistema configurado previamente la sección 2.2.3 con la función *sdrrx()*. El argumento *datosRx* es un vector columna de valores complejos que contiene los datos recibidos. Los valores y rangos de los datos recibidos dependen del tipo de datos con el que se haya configurado el objeto del sistema *rxAdalm*, los pueden ser [11]:

- Tipo int16 con valores entre –2048 y 2047. Este tipo de datos utiliza 16 bits para representar cada muestra.

- Tipo double con valores escalares entre -1 y 1. Este tipo de datos utiliza 64 bits para representar cada valor.
- Tipo single con valores escalares entre -1 y 1. Este tipo de datos utiliza 32 bits para representar cada valor.

En el prototipo de comunicación inalámbrica que se está desarrollando en este trabajo se utilizan valores del tipo **single**. Matlab trabaja por defecto con datos del tipo double, por lo que el vector de datos *datosRx* del tipo single debe transformarse a un vector de datos del tipo double. Para esto se utiliza el comando del siguiente cuadro.

```
datosRecibidos=double(datosRx);
```

La función de Matlab *double()* transforma el vector *datosRx*, que en nuestro caso es del tipo single, en tipo double y se almacena en la variable *datosRecibidos*.

La etapa de **Recortado de datos** consiste en eliminar parte de las muestras iniciales y finales que contengan solamente ruido, reduciendo el tamaño del bloque a procesar.

El recortado se realiza en base a un **umbral**, tal que se discrimina el rango de valores que está debajo de dicho umbral. En la Figura 2.112 se muestra encerrado en rojo los valores que se desea mantener, mientras que lo demás se debe eliminar. Esta extracción se señal permitirá reducir la carga computacional en las etapas posteriores.

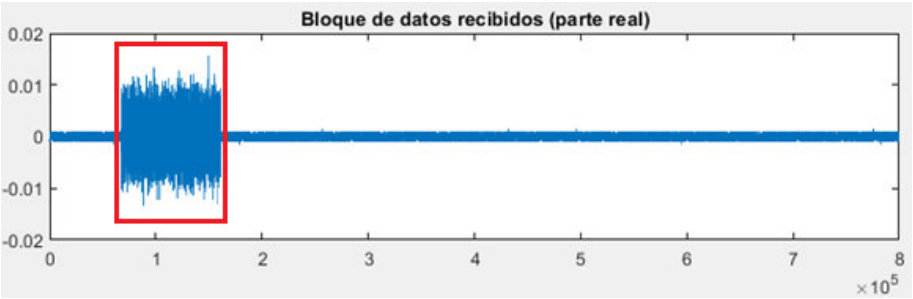


Figura 2.112. Bloque de datos recuperado del canal inalámbrico e identificado con un cuadro rojo las muestras que se buscan conservar.

2.5.1.1. Establecimiento del umbral de recepción

El umbral de recepción se puede establecer en base al nivel del ruido presente en el canal inalámbrico. Para medir el nivel de ruido del canal se crea un script llamado *medir_ruido.m*, el cual se encarga de obtener un valor promedio del ruido presente en el canal inalámbrico. Este script se utiliza únicamente para obtener el valor de ruido captado por el equipo SDR de recepción, más no para recibir datos. Hay que considerar que el script se debe ejecutar

cuando no se esté enviando información desde el equipo SDR de transmisión, tal que en el canal inalámbrico se tenga solamente ruido.

El código del script *medir_ruido.m* inicia por limpiar todas las variables del espacio de trabajo. Luego ejecuta el script *config_SDR.m*, obtenido en el capítulo de configuración, dicho script configura los objetos el sistema de transmisión y recepción, el objeto de recepción se llama *rxAdalm*. Después se inicializa una variable que va a almacenar los valores máximos de cada bloque de datos de ruido que se obtengan del canal inalámbrico.

Utilizando un lazo *for* que va del 1 al 30 (se tomará unas 30 porciones de ruido, suficientes para estimar el máximo valor del ruido), se obtiene el máximo valor del ruido de cada bloque y se almacena en la variable inicializada previamente. Dentro del lazo se reciben los datos utilizando el objeto del sistema de recepción y se almacenan en una variable. Los datos recibidos se transforman del tipo single al tipo double. Luego se obtiene el valor máximo de la magnitud de los datos recibidos y se muestra este valor. Después se almacena dicho valor en una posición del vector que se inicializó anteriormente. Para que los siguientes valores de ruido del canal inalámbrico no sean en un tiempo cercano, se realiza una pausa de 2 segundos.

Luego con la función de Matlab *mean()*, se calcula el valor promedio del vector que contiene los valores máximos de ruido. Finalmente, se muestra el valor promedio del ruido presente en el canal inalámbrico. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
% Script para obtener un promedio del nivel del ruido del canal
% inalámbrico y poder establecer un umbral para la recepción de
% datos
clear
config_SDR() % Ejecutar el script de configuración de los equipos SDR
% Medir el nivel de ruido:
valorMuestras=[]; % Iniciar el vector que almacena los valores maxim.
% Con un lazo for se muestrea se toma 30 bloques de datos
for i=1:30
    datosRxComplejos = rxAdalm(); % Recibir los datos
    datosRecibidos = double(datosRxComplejos); % De single a double
    v_max=max(abs(datosRecibidos)); % Máximo valor del bloque de
    % datos obtenido con el SDR Adalm Pluto
    fprintf('Valor máximo de las muestras: %7.4f\n',v_max) % Mostrar
    % el valor máximo
    valorMuestras(i)=v_max; % Guardar los valores máximos
    pause(2)
end % Final del bucle while
promedioRuido=mean(valorMuestras); % encontrar el promedio
disp('**********************************************')
% Mostrar el promedio:
fprintf('Valor promedio del ruido es: %7.4f\n',promedioRuido)
disp('**********************************************')
```

Antes de ejecutar el script *medir_ruido.m*, se debe tener conectado al computador un dispositivo SDR Adalm Pluto que esté con la antena de recepción conectada. Entonces cuando se ejecuta el script para medir el ruido del canal inalámbrico se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.113. Cabe mencionar que el script *medir_ruido.m* tardará alrededor de un minuto en terminar su ejecución.

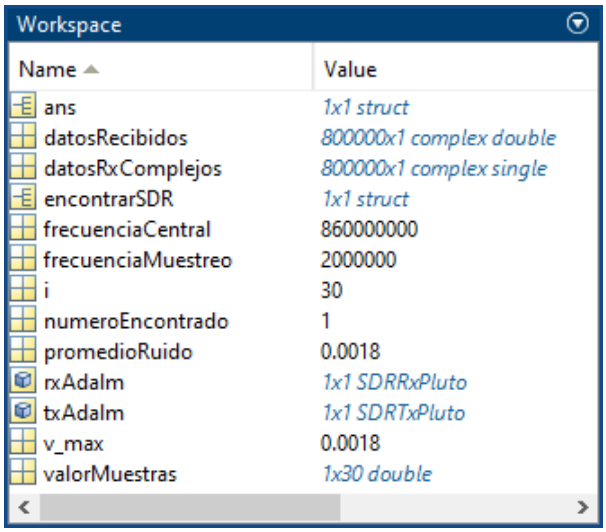


Figura 2.113. Variables en el espacio de trabajo luego de ejecutar el script *medir_ruido.m*.

La variable *promedioRuido* almacena el valor del ruido promedio presente en el canal inalámbrico. En la Figura 2.114 se muestran las últimas líneas del resultado que se obtiene en la ventana de comandos, luego de ejecutar en 3 diferentes ocasiones el script *medir_ruido.m*. El equipo SDR que se utilizó solo tenía la antena de recepción conectada. El valor promedio del ruido presente en el canal inalámbrico se encierra con color verde y en todos los casos es de 0.0018.

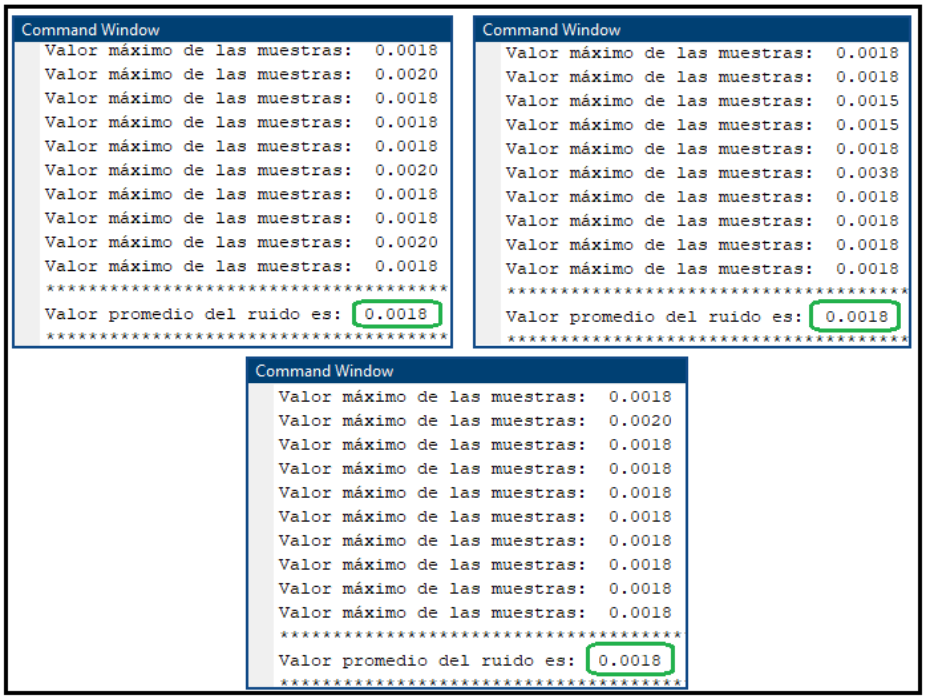


Figura 2.114. Valor promedio del ruido obtenido luego de ejecutar en 3 ocasiones el script *medir_ruido.m*, con solo la antena de recepción conectada.

Cuando se tiene conectada la antena de transmisión y de recepción en el equipo SDR Adalm Pluto, el nivel de ruido que se detecta del canal inalámbrico se incrementa. En la Figura 2.115 se muestra las últimas líneas del resultado que se obtiene en la ventana de comandos, luego de ejecutar en 3 diferentes ocasiones el script *medir_ruido.m*. El valor promedio del ruido presente en el canal inalámbrico presenta valores alrededor de 0.0030.

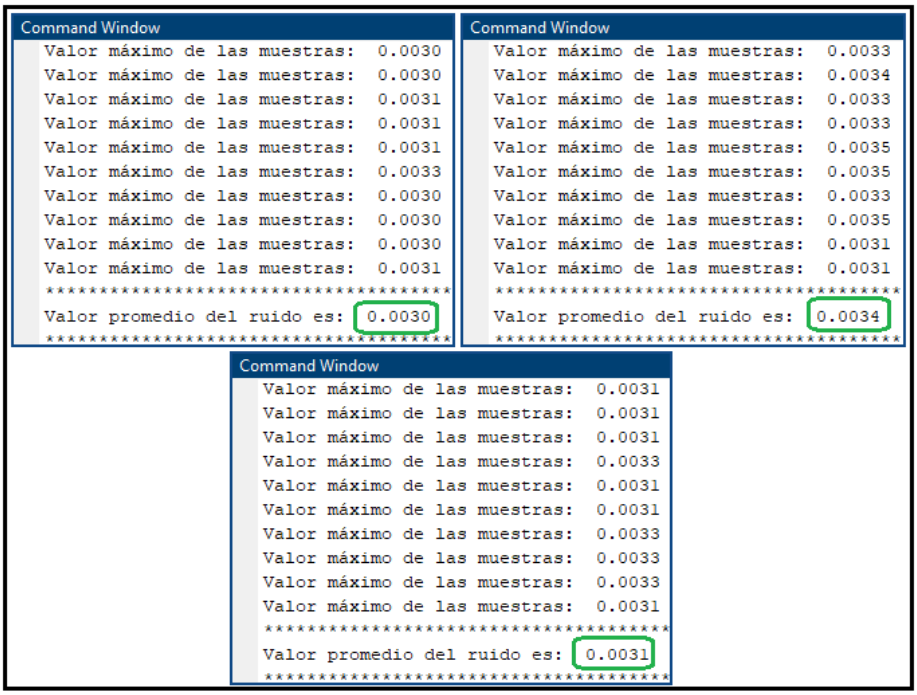


Figura 2.115. Valor promedio del ruido obtenido luego de ejecutar en 3 ocasiones el script *medir_ruido.m*, con las dos antenas conectadas al equipo Adalm Pluto

En base a las pruebas realizadas, resulta que en el prototipo que se está desarrollando es una mejor opción utilizar los equipos SDR con una sola antena, ya sea de transmisión o de recepción. El prototipo no utiliza ninguna clase de respuesta por parte del receptor, es decir se puede utilizar comunicación simplex. Cuando se conectan las dos antenas, el receptor puede convertirse en transmisor y viceversa.

Aunque se pretende trabajar solo con una antena en cada dispositivo SDR Adalm Pluto, si se desea trabajar en modo duplex se conectan las dos antenas. Por tal razón, un umbral de recepción adecuado para desarrollar ejemplos demostrativos de recepción de datos es de 0.0040.

2.5.1.2. Recepción del archivo de texto enviado

En la sección de transmisión se explicó cómo procesar y enviar los datos de un archivo de texto al canal inalámbrico. Para recibir los datos de información del canal inalámbrico, es necesario que en otro computador se tenga el archivo de texto procesado listo para ser enviado. En el computador de transmisión, cuando se tenga un archivo de texto procesado y un dispositivo SDR Adalm Pluto configurado para transmitir, se puede ejecutar el script *transmitir.m*. Este script se encarga de enviar los datos procesados al canal inalámbrico utilizando el equipo Adalm Pluto.

En el computador que se va a utilizar para la **recepción** se debe ejecutar el script ***config_SDR.m*** con el equipo SDR Adalm Pluto conectado. Después se puede ejecutar un script que se encarga de la recepción de los datos del canal inalámbrico.

Como resultado de ejecutar el script *config_SDR.m*, obtenido en el capítulo de configuración, se obtiene las variables en el espacio de trabajo que se muestra en la Figura 2.116.

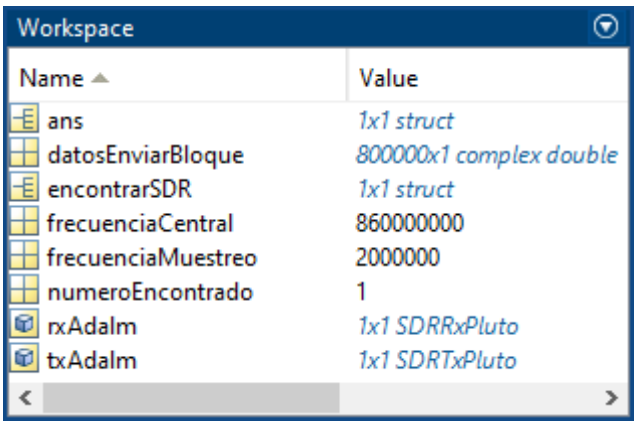


Figura 2.116. Variables en el espacio de trabajo luego de ejecutar *config_SDR.m* con un equipo SDR conectado al computador.

La variable ***rxAdalm*** contiene el objeto creado por sdrrx mientras que la variable ***txAdalm*** contiene el objeto creado por sdrtx. En la Figura 2.117 se muestra el contenido de estas dos variables luego de ejecutar *config_SDR.m*.

```
>> config_SDR
Objeto para la transmisión configurado:

txAdalm =

  comm.SDRTxPluto with properties:

   Main
                DeviceName: 'Pluto'
                   RadioID: 'usb:0'
           CenterFrequency: 860000000
                      Gain: -2
            ChannelMapping: 1
         BasebandSampleRate: 2000000
     ShowAdvancedProperties: false

  Show all properties
```

```
Objeto para la recepción configurado:

rxAdalm =

  comm.SDRRxPluto with properties:

   Main
                DeviceName: 'Pluto'
                   RadioID: 'usb:0'
           CenterFrequency: 860000000
                GainSource: 'Manual'
                      Gain: 20
            ChannelMapping: 1
         BasebandSampleRate: 2000000
             OutputDataType: 'single'
           SamplesPerFrame: 800000
           EnableBurstMode: false
     ShowAdvancedProperties: true
```

Figura 2.117. Características de los objetos *txAdalm* y *rxAdalm*.

El script *recibir.m* se encarga de recuperar los datos enviados por el transmisor al canal inalámbrico. El código del script inicia por sondear el canal varias veces con un lazo *for*, este proceso se realiza con el objetivo de eliminar muestras que pudieron haberse quedado en el buffer del SDR. Luego se define el umbral de recepción igual a 0.0040. La recepción de datos se implementa mediante un lazo ***while*** que sondea el canal inalámbrico hasta que los datos recibidos superen el umbral de recepción. Antes del lazo se define una variable llamada *condAux* y se le asigna un valor de 0. La variable *condAux* se utiliza para permanecer dentro del lazo *while*.

Dentro del *while* se recuperan datos del canal inalámbrico con el objeto del sistema *rxAdalm* y se almacenan en un vector. Este vector es del tipo **single** y se transforma a tipo **double** con la función de Matlab llamada *double()*. Luego se obtiene el valor máximo de la magnitud de cada muestra del vector de datos recuperados y se muestra dicho valor. Después con un condicional *if* se verifica si el valor máximo obtenido supera al umbral de recepción. En caso de hacerlo se muestrea el canal nuevamente, ya que puede darse el caso que la señal de datos esté dividida en dos bloques, como se muestra en la Figura 2.124.

Si se supera el umbral nuevamente, se guardan los dos bloques dentro del vector que almacena los datos recuperados. Luego a la variable *condAux* se asigna el valor de 1 para salir del lazo *while*. Finalmente, ya fuera del lazo *while*, se grafica la parte real y parte imaginaria del bloque de datos que superó el umbral establecido.

El código que se describió en estos últimos párrafos y correspondiente al script *recibir.m* se presenta a continuación, debidamente comentado.

```
% Script que permite recibir el bloque de datos enviado por el
% transmisor, antes de correr este script es necesario ejecutar
% el script config_SDR.
% Vaciar posibles muestras almacenadas en el buffer
for i=1:6
    vaciar = rxAdalm(); % Este vector no se utiliza, solo se muestrea
    % el canal varias veces para que en caso de existir allgún buffer
    % este se elimine
end
umbralRx=0.0040; % Establecer el umbral mayor a 0.0018;
%Recepcion de los datos:
condAux=0; % Condición auxiliar, para estar dentro del while
while(condAux==0)
    datosRxComplejos = rxAdalm(); % Guarda la señal recuperada del
    % canal inalámbrico
    datosRecibidos=double(datosRxComplejos); % De single a double
    v_max=max(abs(datosRecibidos)); % Máximo valor de la magnitud
    % de cada dato recibido
    fprintf('Valor máximo de las muestras: %8.5f\n',v_max) % Mostrar
    % la magnitud máxima
    % Verificar si se supera el umbral establecido:
    if(v_max>umbralRx)
        datosRxAux=rxAdalm(); % Guarda la señal recuperada
        datosRec2=double(datosRxAux); % De single a double
        % Verificar si se supera el umbral:
        if (max(abs(datosRec2))>umbralRx)
            % Guardar en el mismo vector
            datosRecibidos=[datosRecibidos;datosRec2];
        end
        condAux=1; % Cambiar el valor para salir del lazo while
    end
end
% Graficar la parte real y parte imaginaria d elos datos recibidos:
figure()
subplot(2,1,1)
plot(real(datosRecibidos))
title('Bloque de datos recibidos (parte real)')
subplot(2,1,2)
plot(imag(datosRecibidos))
title('Bloque de datos recibidos (parte imaginaria)')
```

En la Figura 2.118 se observa una representación del escenario en el cual se va a realizar la transmisión del archivo de texto ya procesado. Se utiliza solamente una antena en el equipo receptor y otra antena en el equipo transmisor, no es necesario que se conecten las dos antenas en el equipo SDR Adalm Pluto. De hecho, se obtiene mejores resultados si se utiliza solo una antena por equipo.

En el computador del transmisor, se tiene el equipo SDR configurado y el archivo de texto procesado. También es necesario que el script de transmisión de datos llamado *transmitir.m*, se haya ejecutado al menos una vez. Esto se debe a que la primera vez que

el script se ejecuta, Matlab establece conexión con el SDR y este emite una señal que supera el umbral de recepción.

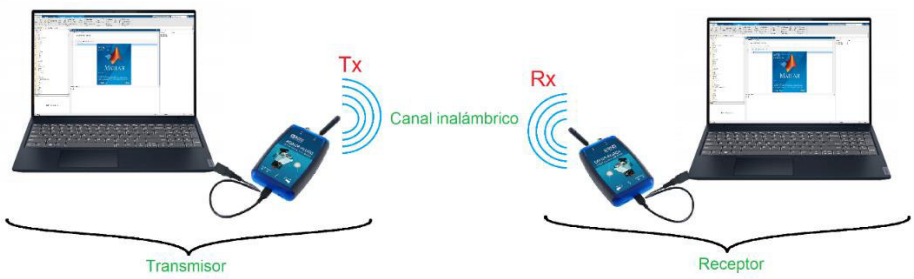


Figura 2.118. Esquema de hardware del prototipo de comunicación.

Cuando se ejecuta el script *recibir.m* antes de que se haya ejecutado al menos una vez el *transmitir.m*, se obtiene como resultado la señal que se muestra en la Figura 2.119. Esta señal no corresponde a la señal de datos enviada por el transmisor, si no a la señal que emite el SDR cuando establece la conexión con Matlab.

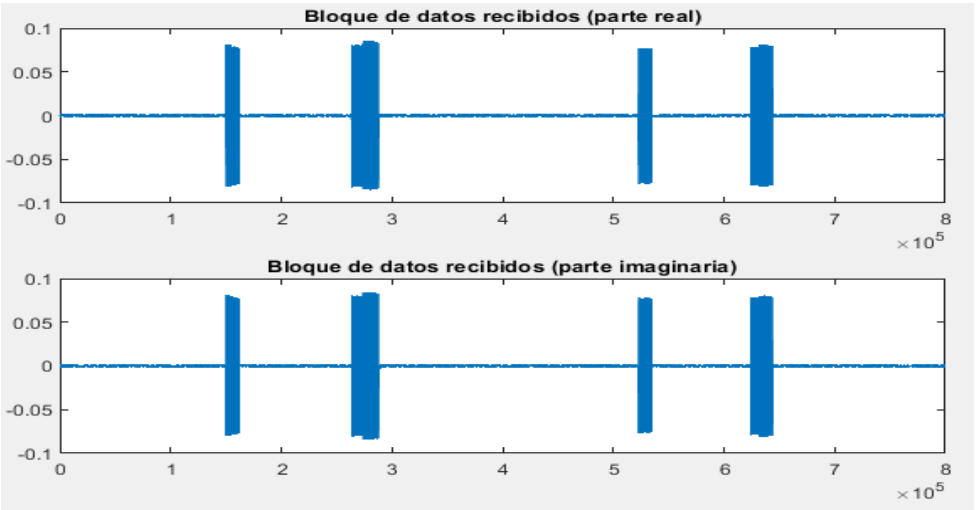


Figura 2.119. Señal que emite el SDR de transmisión cuando este establece una conexión con Matlab.

La distancia entre los SDR de transmisión y recepción son 150 centímetros, con línea de vista directa.

En el computador de recepción, con el script *cofig_SDR.m* ejecutado, cuando se ejecuta el script *recibir.m*, se establece la conexión con el dispositivo SDR Adalm Pluto, esto se puede ver en la segunda línea de la Figura 2.120. Esta conexión solo ocurrirá en la primera ocasión que se ejecute el script de recepción, luego de configurar los equipos SDR. Si se ejecuta por una segunda ocasión el script *recibir.m* el dispositivo SDR ya estará conectado, por lo que no aparecerá el mensaje mencionado.

Cuando se está ejecutando el script de recepción, en la ventana de comando de Matlab se observa que se muestra el máximo valor de las muestras y con esto se da a entender que

se está muestreando el canal inalámbrico. Solamente el último valor máximo corresponde a las muestras del bloque de información y por ende se sale del lazo *while*, terminando así el bucle.

En la Figura 2.120 se puede ver los valores máximos de las muestras obtenidas del canal inalámbrico en cada iteración de lazo, hasta que aparece un valor que supera el umbral de recepción. Esto significa que en el transmisor se ha enviado datos mediante el script *transmitir.m* y que el receptor ha detectado estos datos.

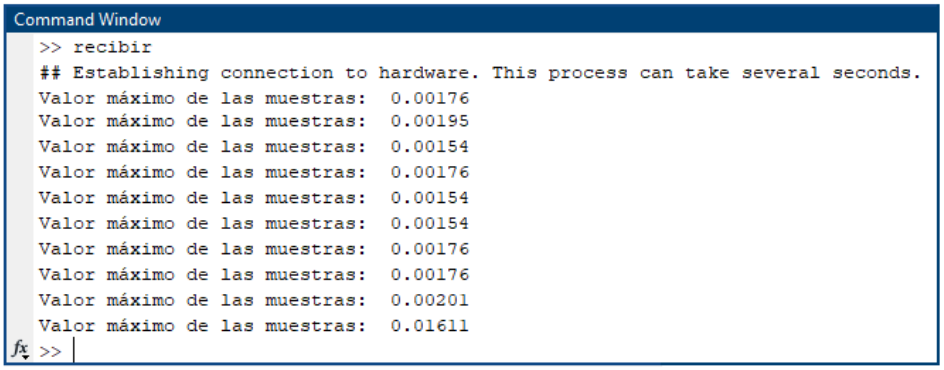


Figura 2.120. Resultado en la ventana de comandos luego de ejecutar *recibir.m* y que el transmisor haya enviado datos de información.

En la Figura 2.121 se muestra las variables que se obtienen en el espacio de trabajo, luego de ejecutar *config_SDR.m* y *recibir.m*. Cuando los datos hayan sido recuperados en dos bloques y no en solamente uno, el vector *datosRecibidos* tendrá una longitud de 1600000 (Figura 2.124). En el caso de la Figura 2.122 los datos han sido recuperados solo con un bloque.

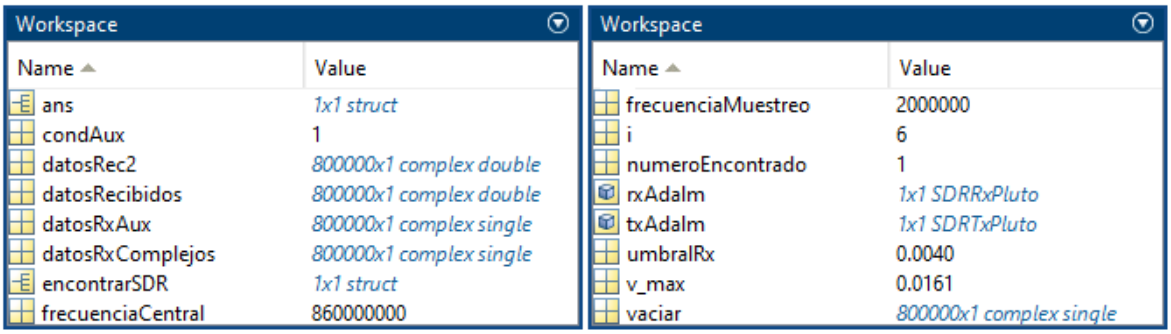


Figura 2.122. Variables en el espacio de trabajo luego de ejecutar *config_SDR.m* y *recibir.m*.

En la Figura 2.123 se muestra la parte real y parte imaginaria del bloque de datos recuperado del canal inalámbrico. Los datos de información del canal inalámbrico se pudieron recuperar solamente con un bloque.

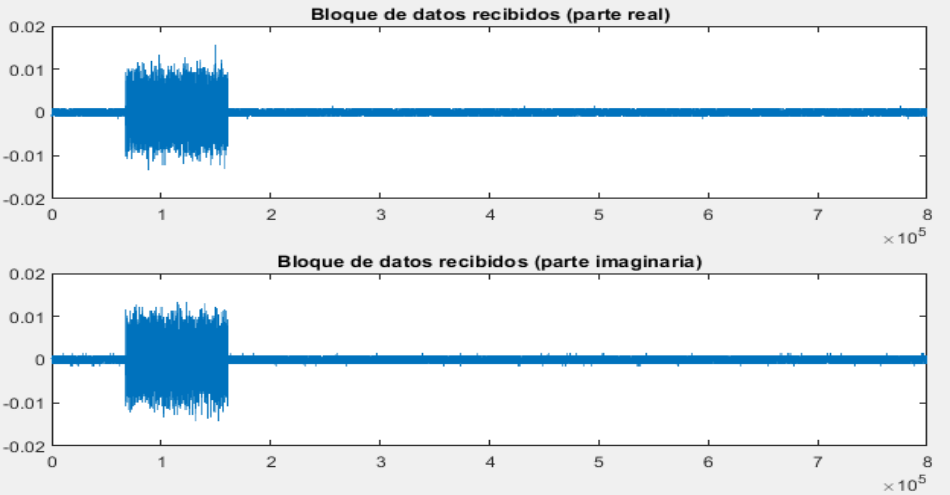


Figura 2.123. Bloque de datos recuperado del canal inalámbrico.

Para trabajar en los ejemplos de recuperación del archivo de texto enviado y que estos tengan continuidad, se debe guardar los datos de información recuperados del canal inalámbrico contenidos en el vector *datosRecuperados*.

Los datos se pueden guardar en un archivo *.mat* utilizando la función de Matlab *save()*. En el siguiente cuadro se coloca la línea de código que se puede utilizar para guardar el vector *datosRecibidos*, en el archivo *datosRecibidos150.mat*

```
save('datosRecibidos150.mat','datosRecibidos')
```

Para cargar los datos almacenados en el archivo *datosRecibidos150.mat* se utiliza el comando del siguiente cuadro.

```
load('datosRecibidos150.mat')
```

Luego de ejecutar el comando anterior en el espacio de trabajo se obtiene el vector *datosRecibidos*, el que contiene los datos que se grafican en la Figura 2.123.

En la Figura 2.124 se muestra que ocurre cuando la información se recupera en dos bloques. El tamaño de muestras respecto a la Figura 2.123 se duplica, pero los datos se siguen manteniendo dentro del vector *datosRecibidos*.

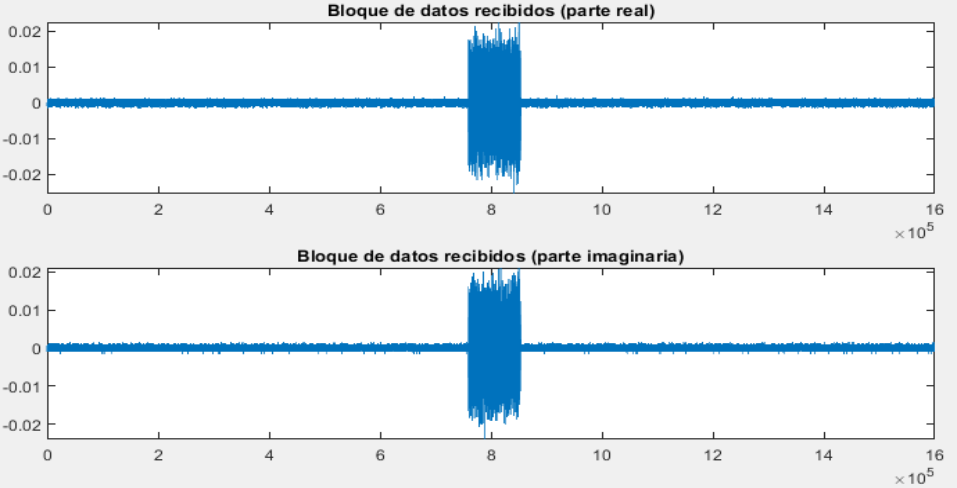


Figura 2.124. Datos de información recuperados del canal inalámbrico, se utilizan dos bloques.

2.5.1.3. Recortado de datos en el archivo de texto recibido

Esta etapa consiste en disminuir el tamaño del bloque recibido, con el objetivo de reducir la carga computacional de procesamiento en etapas posteriores. El recortado se realiza en base al umbral utilizado para la recepción de datos en la etapa anterior. Los datos de entrada de esta etapa se encuentran en un vector columna llamado *datosRecibidos*. Este vector se puede obtener directamente del script *recibir.m* o bien del bloque de datos almacenados en el archivo *datosRecibidos150.mat*.

Para tener continuidad con el procesamiento de los datos mostrado en la etapa anterior se ha optado por trabajar con el archivo *datosRecibidos150.mat*. Este archivo se puede cargar mediante el comando que se presenta a continuación. Y dicho archivo contiene

```
load('datosRecibidos150.mat') % Cargar los datos guardados
```

Luego de ejecutar el comando se obtiene en el espacio de trabajo de Matlab el vector *datosRecibidos*, es del tipo complex double y tiene una longitud de 800 000.

La etapa Recortado de datos se implementa en un script llamado *recortar_rx.m*. El código inicia por definir el umbral de recepción que es de 0.0040. Luego se inicializa en 0 una variable que almacenará la primera posición en donde el vector *datosRecibidos* supera el umbral. Con un lazo *for* se recorre el vector *datosRecibidos* y con un condicional *if*, se compara si el valor máximo de la magnitud del dato actual supera al umbral. Cuando se encuentre el primer valor que supera el umbral se almacena la posición y sale del lazo *for* con un *break*.

Luego se guarda 400 datos antes de encontrar el umbral, en caso de que el umbral se haya superado antes de los primeros 400 datos, solo se copia el vector de entrada. Después se

invierte el vector recortado, tal que los datos finales pasen al inicio y viceversa. Con esto se puede repetir el proceso de encontrar el primer valor que supere el umbral y en este caso se guardan desde 200 datos antes de superar el umbral. Cuando se haya recortado los datos, se invierte el vector por segunda ocasión, tal que la disposición quede como al inicio.

Se guardan 400 muestras antes del primer valor que supera el umbral, debido a que podría darse el caso que el preámbulo tenga una menor amplitud que los símbolos OFDM. Si bien la diferencia no será significativa, hay que considerar ese posible escenario. Se recortan los datos finales para reducir la carga de procesamiento, ya que se busca que el prototipo procese los datos lo más rápido posible.

Finalmente, se grafica los datos totales recibidos y los datos recortados, tanto la parte real como imaginaria. El código del script *recortar_tx.m*, debidamente comentado, se presenta a continuación

```
% Script que permite recortar los datos de información del bloque
% de datos recibido en el script recibir.m
% Antes de correr este script es necesario ejecutar el script
% config_SDR.m y recibir.m o bien se necesita cargar los datos
% guardados con load('datosRecibidos150.mat')
% En vector de entrada de esta etapa es datosRecibidos
% RECORTAR DATOS
umbralRx=0.0040;
% Se recorre el vector de entrada y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_inic=0; % Inicializar variable
for i=1:length(datosRecibidos)
    if(max(abs(datosRecibidos(i)))>umbralRx)
        pos_inic=i; % Primera posición arriba del umbral
        break;
    end
end
% Guardar desde 400 datos antes de superar el umbral, hasta el final:
if(pos_inic>400)
    datosRec=datosRecibidos(pos_inic-400:end); % Datos recortados
else
    % El umbral se superó antes de las primeras 400 posiciones, solo
    % se copia los datos
    datosRec=datosRecibidos;
end
datosInv=flip(datosRec); % Invierte el vector, tal que los datos
% finales queden al inicio, con esto se puede repetir el proceso
% anterior.
% Se recorre el vector de invertido y  se obtiene la posición del
% primer valor por arriba del umbral:
pos_fin=0; % Inicializar variable
for i=1:length(datosRec)
    if (max(abs(datosInv(i)))>umbralRx)
        pos_fin=i; % Primera posición arriba del umbral
        break
    end
```

```
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_fin>200)
    datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
else
    % El umbral se superó antes de las primeras 200 posiciones, solo
    % se copia los datos
    datosInvRec=datosInv;
end
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
% para obtener los datos recortados
% Graficar datos de entrada de la etapa y los datos recortados
figure()
subplot(2,2,1)
plot(real(datosRecibidos))
title('Datos recibidos (parte real)')
subplot(2,2,3)
plot(imag(datosRecibidos))
title('Datos recibidos (parte imaginaria)')
subplot(2,2,2)
plot(real(datosRecortados),'g')
title('Datos recortados (parte real)')
subplot(2,2,4)
plot(imag(datosRecortados),'g')
title('Datos recortados (parte imaginaria)')
```

Luego de aplicar el recortado de datos al vector *datosRecibidos* se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.125. La variable que contiene los datos de salida de la etapa es *datosRecibidos*. Además, se obtiene las gráficas de la Figura 2.126.

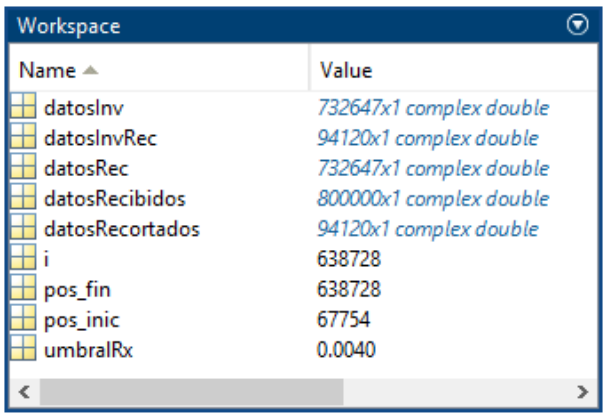


Figura 2.125. Variables en el espacio de trabajo luego de ejecutar el script *recortar_rx.m*.

En la Figura 2.126 se muestra con azul la parte real e imaginaria de los datos recibidos y con verde la parte real e imaginaria de los datos recortados. Además, en las gráficas en azul se encierra la trama de datos que se va a obtener después de recortar, es decir los datos recortados que se muestran con color verde.

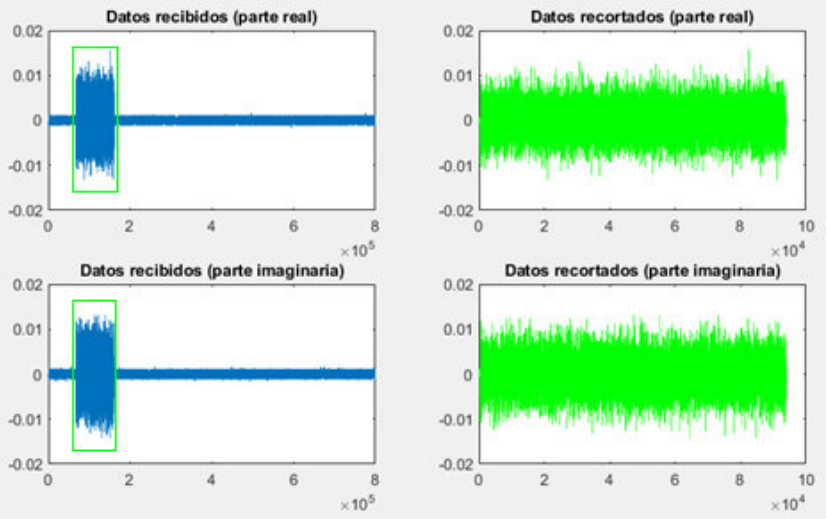


Figura 2.126. Gráficas de los datos recibidos y de los datos recortados.

2.5.2. PREÁMBULO 802.11A

En transmisión se agregó el preámbulo IEEE 802.11a compuesto por la sección corta o LSTF (Legacy Short Training Field) y por la sección larga o LLTF (Legacy Long Training Field). **En recepción** el preámbulo permite: detectar la trama OFDM, corregir el CFO, estimar el inicio de la trama y corregir la fase de la señal recibida.

En la Figura 2.127 se muestran con color anaranjado todas las etapas que se abarca en la sección actual.

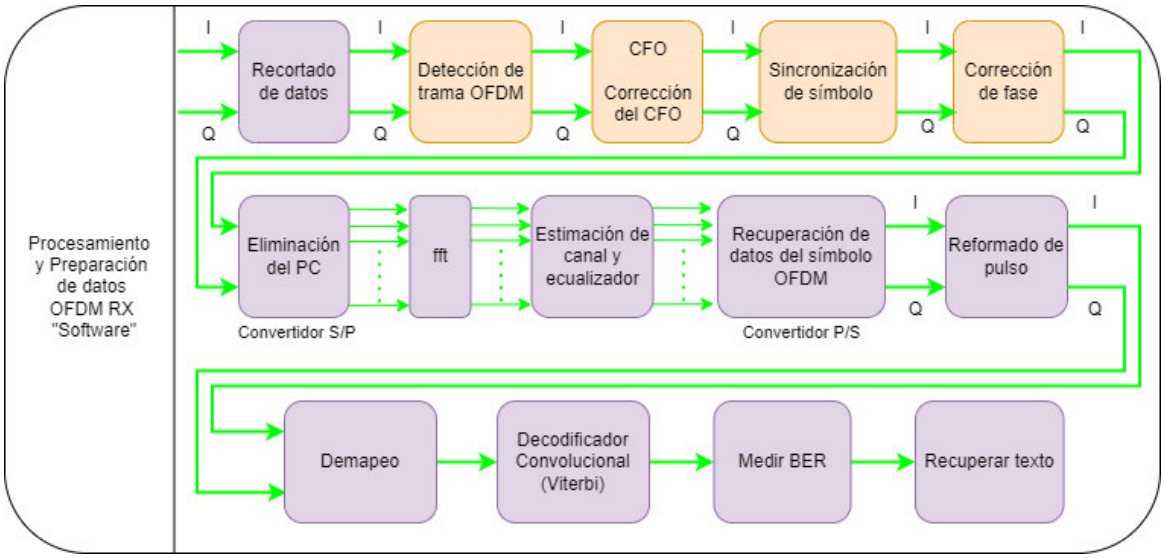


Figura 2.127. Etapas de recepción con todas las etapas que se implementan con el preámbulo 802.11a.

La sección corta del preámbulo se utiliza para implementar las etapas de Detección de trama OFDM y Corrección del CFO. **Con la sección larga** se implementa las etapas de

Sincronización de símbolo y la Corrección de fase. Cada una de estas etapas se implementa en una subsección, dentro de las mismas se abarca los aspectos teóricos y el procesamiento del archivo de texto recibido.

2.5.2.1. Detección de la trama OFDM

La detección del paquete OFDM permite evaluar si los datos recuperados del canal inalámbrico pertenecen a una trama OFDM. Cuando no pertenece a una trama OFDM se descarta el paquete para evitar seguir con las siguientes etapas innecesariamente. En caso de que se detecte una señal OFDM se realiza un ajuste en tiempo, que no es nada más que recortar nuevamente los datos recibidos. Esta decisión se la toma en base a una métrica llamada $M(k)$ y que se la verá más adelante.

En la **etapa Recortado de datos** se reduce el tamaño del bloque recibido, pero no se discrimina si pertenece o no a una trama OFDM. El preámbulo 802.11a permite detectar la trama OFDM mediante la sección corta o **LSTF**. Las secuencias cortas permiten obtener un pico, cuando se calcula la métrica de tiempo del detector de paquetes mediante la ecuación 2.20 [4].

$$M(k) = \frac{\sum_{m=0}^{L-1} r^*(k+m)r(k+m+L)}{\sum_{m=0}^{L-1} |r(k+m+L)|^2} \qquad (2.20)$$

En donde:

- r es la señal recibida
- $*$ es la conjugación compleja
- L es un múltiplo de la longitud de una secuencia corta, es decir $L = 16 * n$, en donde $n = 1,2, ...$
- M es la métrica de tiempo
- k es el índice de M

La métrica de una señal OFDM presenta valores de alrededor de uno, cuando se detectan las secuencias cortas, ya que estas secuencias presentan un **alto valor de autocorrelación**. La autocorrelación es el resultado de comparar una señal consigo misma, cuanto más se parezca mayor es el valor de la métrica. Por otro lado, cuando se analiza los datos de una señal cualquiera los valores de la métrica $M(k)$ son bajos y están por debajo de un umbral. Una forma de atenuar los valores bajos de la métrica $M(k)$ es obteniendo el cuadrado de estos valores. Además, al obtener el cuadrado de los valores que están alrededor de uno, estos se mantienen casi iguales. De esta manera se puede obtener un rango más amplio para determinar un umbral para la métrica $M(k)$.

Para comparar los valores de la métrica $M(k)$ con el cuadrado de la misma se implementa un ejemplo en la primera sección de un script llamado *aux_grafPreambuloRx.m*. Como entrada se debe tener los datos recibidos del canal inalámbrico, que se han guardado en el archivo *datosRecibidos150.mat*. El código inicia por reccortar los datos, es decir eliminar la mayoría de las muestras de ruido del bloque. Luego se toman las primeras 1000 muestras de los datos recortados y se almacena en el vector ***r*** para que coincida con los parámetros de la ecuación 2.20. El tomar solo las primeras 1000 muestras permite aumentar la velocidad de procesamiento, ya que se espera que el preámbulo se encuentre en estas primerar muestras.

Después se define el parámetro L en 48, se calcula el máximo valor que puede tomar k y se almacena en el vector *inMax*, se inicializa la variable *M* que almacena los valores de la métrica y se define el valor máximo que puede tomar m en la variable del mismo nombre. Con un lazo *for* se recorre el índice k de la métrica $M(k)$. Dentro del lazo se implementa la ecuación 2.20 para cada valor de k, el sumatorio del numerador se implementa directamente con una multiplicación de matrices. El sumatorio del denominador se implementa mediante la función *sum*. Luego se calcula las magnitudes de la métrica y se almacena en *M1* y el cuadrado de las magnitudes de aalmacena en *M_cua*. Finalmente, se grafican las primeras 700 muestras de *M1*, *M_cua* y de la parte real de los datos recortados.

El código correspondiente a la primera sección del script *aux_grafPreambuloRx.m*, el cual se describió en los dos últimos párrafos, se presenata a continuación.

```
%% Graficar la métrica junto con los primeros datos
% RECORTAR DATOS:
umbralRx=0.0040; % Umbral para recortar
% Se recorre el vector de entrada:
pos_inic=0; % Inicializar variable
for i=1:length(datosRecibidos)
    if(max(abs(datosRecibidos(i)))>umbralRx)
        pos_inic=i; % Primera posición arriba del umbral
        break;
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_inic>200)
    datosRec=datosRecibidos(pos_inic-200:end); % Datos recortados
else
    datosRec=datosRecibidos;
end
datosInv=flip(datosRec); % Invierte el vector
% Se recorre el vector de invertido:
pos_fin=0; % Inicializar variable
for i=1:length(datosRec)
    if (max(abs(datosInv(i)))>umbralRx)
        pos_fin=i; % Primera posición arriba del umbral
```

```
        break
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_fin>200)
    datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
else
    datosInvRec=datosInv;
end
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
% Detección de inicio del paquete OFDM:
datosRX=datosRecortados; % Copiar en una variable
Mmues=1000; % Muestras que se toman para calcular la métrica
if(length(datosRX)>Mmues)
    r=datosRX(1:Mmues);
else
    r=datosRX;
end
L = 48; % Múltiplo de la longitud de una secuencia corta (16*x)
inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
% el indice maximo a evaluar es la longitud de r menos 2L.
M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
m=L-1; % Máximo valor de m
Numerador=0; % Inicializar
Denom=0; % Inicializar
% Determinación de la métrica M(k) para saber si es un paquete OFDM:
for k=1:inMax
    Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
    Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
    M(k)=Numerador/Denom; % Métrica
end
% Graficar:
M_cua=(abs(M)).^2; % Magnitud cuadrada de cada valor de la métrica
M1=abs(M); % Magnitud de cada valor de la métrica
figure()
subplot(3,1,1);plot(M_cua(1:Mmues-300));grid on
title('Cuadrado de la magnitud de la métrica M(k)')
subplot(3,1,2);plot(M1(1:Mmues-300));grid on
title('Magnitud de la métrica M(k)')
subplot(3,1,3);plot(real(datosRecortados(1:Mmues-300)))
title('Primeros datos recortados');grid on
```

El resultado del código anterior se muestra en la Figura 2.128, en donde se compara la magnitud de la métrica elevada al cuadrado, de la magnitud de la métrica (sin elevarla al cuadrado) y de la señal OFDM recuperada del canal inalámbrico en base a la cual se calculó la métrica $M(k)$. Además, se señala con cotas el valor correspondiente en donde se encuentra la parte LSTF del preámbulo 802.11a. Al observar las gráficas resulta que es una mejor opción evaluar la métrica obteniendo el cuadrado de la magnitud, ya que los picos alrededor de uno se mantienen alrededor de uno y los valores bajos se atenúan de manera considerable.

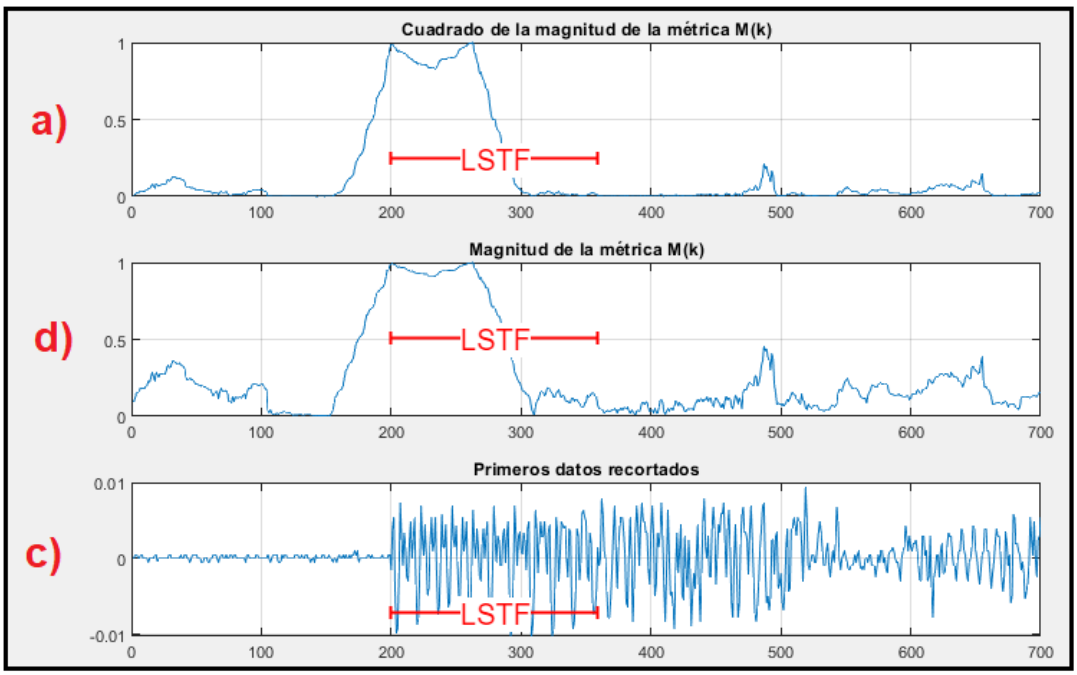


Figura 2.128. Gráfica de la magnitud al cuadrado de la métrica de una señal recupera del canal inalámbrico vs la magnitud (sin elevar al cuadrado) de la misma métrica.

En teoría L puede ser un múltiplo de la longitud de una secuencia corta, que se compone de 16 muestras, es decir que L puede ser igual a 16, 32, 48, etc. Para determinar cuál es el valor de L que mejor resultados presenta para medir la métrica, se debe **evaluar la métrica** con diferentes valores de L. Esto se implementa en una segunda sección de código que se almacena en el script *aux_grafPreambuloRx.m*.

Para esta sección se necesita el vetor ***r*** de la sección 1 del script *aux_grafPreambuloRx.m*. El código inicia por definir los valores que puede tomar L en el vector *Lt*, los valores son: 16, 32, 48, 80, 96. Luego con un lazo *for* se recorre los elemento de *Lt* y se calcula la correspondiente magnitud cuadrada de los elementos de la métrica $M(k)$ y se estos se almacenan en una columna de la matriz *M*. Luego se grafican los valores claculados de la magnitud cuadrada de métrica y se coloca el título respectivo. El código descrito en este párrafo se presenta a continuación.

```
%% Graficar la métrica para L=16,32,48,64,80,96;
%Nota: Correr la sección anterior
Lt = [16 32 48 64 80 96] ;
inMax = length(r)-2.*Lt+1;
M = zeros(Mmues,length(Lt)); % Inicializar M para mayor velocidad
mt=Lt-1; % Máximo valor de mm=L-1; % Máximo valor de m
% Determinar la métrica M(k) para cada L:
for i=1:length(Lt)
    L=Lt(i);
    m=mt(i);
    Numerador=0; % Inicializar
    Denom=0; % Inicializar
    for k=1:inMax(i)
```

```
        Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sum del numerador
        Denom = sum((abs(r(k+L:k+m+L))).^2); % Sum del denominador
        M(k,i)=(abs((Numerador/Denom)))^2; % Magnitud cuadrada
        % de cada valor de la métrica
    end
end
% Graficar:
figure()
subplot(3,2,1);plot(M(:,1)); title('Métrica M(k), L=16')
subplot(3,2,2);plot(M(:,2)); title('Métrica M(k), L=32')
subplot(3,2,3);plot(M(:,3)); title('Métrica M(k), L=48')
subplot(3,2,4);plot(M(:,4)); title('Métrica M(k), L=64')
subplot(3,2,5);plot(M(:,5)); title('Métrica M(k), L=80')
subplot(3,2,6);plot(M(:,6)); title('Métrica M(k), L=96')
```

El resultado de la segunda sección de código del script *aux_grafPreambuloRx.m* se muestra en la Figura 2.129, todos los valores corresponden al cuadrado de la métrica de una misma señal OFDM, recuperada del canal inalámbrico, para varios valores de L.

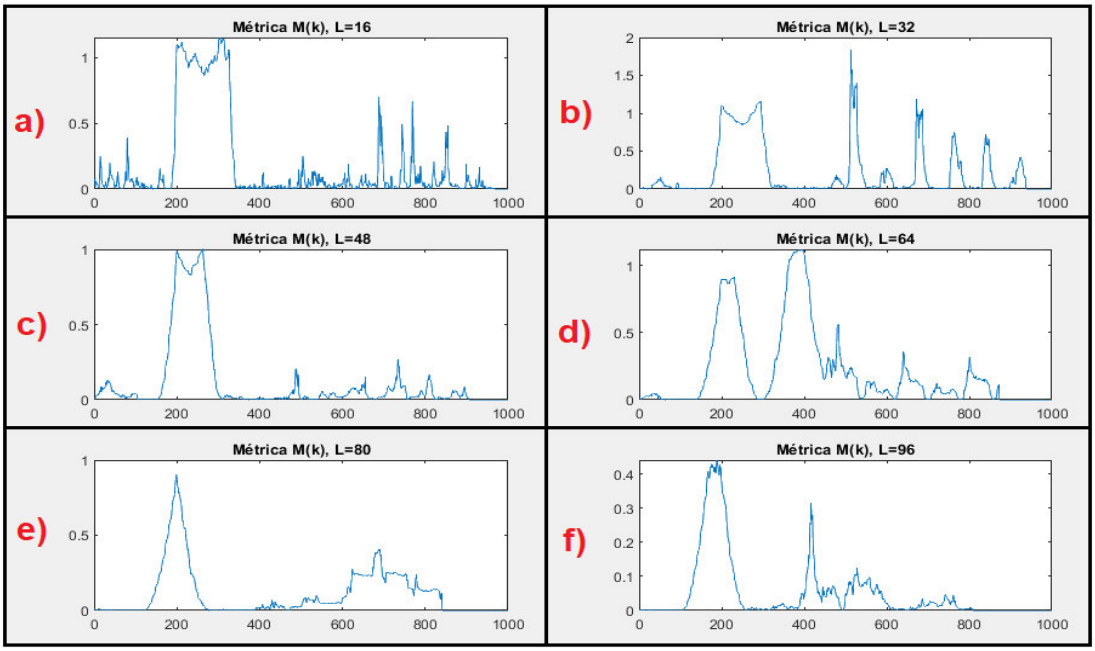


Figura 2.129. Métrica para varios valores de L. a) $L = 16$. b) $L = 32$. c) $L = 48$. d) $L = 64$. e) $L = 80$. f) $L = 96$.

En base a los resultados obtenidos en la Figura 2.129, el valor $L = 48$ es el que mejor resultados presenta para medir la métrica $M(k)$. Con $L = 48$ se obtienen valores bajos de autocorrelación cuando no se analiza las secuencias cortas de LSTF, mientras que cuando se evalúan las secuencias cortas, los valores de autocorrelación están alrededor de uno.

Para poder establecer un **umbral adecuado para evaluar la métrica $M(k)$** y poder evaluar si se trata de una trama OFDM se han recibido muestras de datos en diferentes escenarios. En la Figura 2.130 se muestra varios bloques de datos recibidos en diferentes escenarios, para cada bloque se grafica la métrica. En todos los casos la autocorrelación alcanza en

algún punto valores cercanos a uno, cuando se evalúa la sección LSTF mediante la métrica. Mientras que los demás valores de la métrica no superan los 0.4, en las gráficas de la métrica se trazó una línea roja, justamente en 0.4. Entonces el umbral para identificar la sección LSTF del preámbulo OFDM 802.11a tendrá que ser superior a 0.4.

Para implementar la etapa de Detección de trama OFDM en el archivo de texto recibido, se va a utilizar un umbral de 0.6 para evaluar si los datos recibidos pertenecen a una trama OFDM y para recortar los datos a partir de dicho umbral.

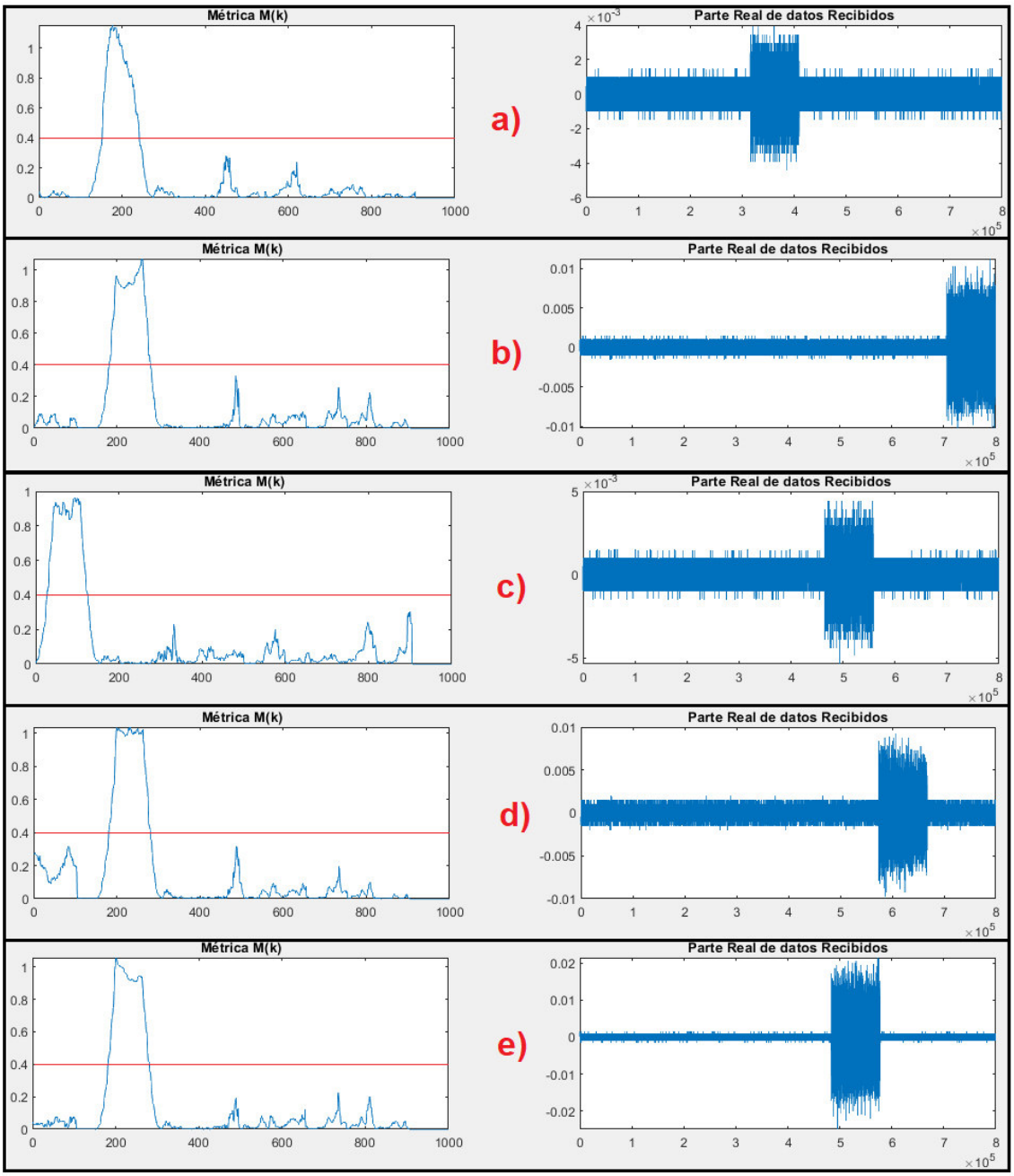


Figura 2.130. Métrica junto con el bloque de datos recibido en distintos escenarios; a) 5 metros sin línea de vista. b) 4.5 metros con línea de vista c) 4 metros sin línea de vista. d) 4 metros sin línea de vista. e) 1.2 metros con línea de vista.

Para obtener cada gráfica de la Figura 2.130 se debe ubicar los equipos en diferente posición (escenarios), con el archivo listo para enviar en el transmisor se corre en el receptor el script *recibir.m* hasta que se envíen los datos y el receptor los recupere. Luego se debe ejecutar la siguiente sección de código, que recorta los datos, calcula la métrica y gráfica el bloque recibido junto con el cuadrado de la magnitud de cada valor de la métrica. Esto se repite las veces que se desee para poder establecer un umbral adecuado en base a la gráfica de la métrica.

```
%% Graficar la métrica junto con el bloque de datos recibido
% RECORTAR DATOS:
umbralRx=0.0040; % Umbral para recortar
% Se recorre el vector de entrada:
pos_inic=0; % Inicializar variable
for i=1:length(datosRecibidos)
    if(max(abs(datosRecibidos(i)))>umbralRx)
        pos_inic=i; % Primera posición arriba del umbral
        break;
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_inic>200)
    datosRec=datosRecibidos(pos_inic-200:end); % Datos recortados
else
    datosRec=datosRecibidos;
end
datosInv=flip(datosRec); % Invierte el vector
% Se recorre el vector de invertido:
pos_fin=0; % Inicializar variable
for i=1:length(datosRec)
    if (max(abs(datosInv(i)))>umbralRx)
        pos_fin=i; % Primera posición arriba del umbral
        break
    end
end
% Guardar desde 200 datos antes de superar el umbral, hasta el final:
if(pos_fin>200)
    datosInvRec=datosInv(pos_fin-200:end); % Recortar datos
else
    datosInvRec=datosInv;
end
datosRecortados=flip(datosInvRec); % Invertir nuevamente el vector
% Detección de inicio del paquete OFDM:
datosRX=datosRecortados; % Copiar en una variable
Mmues=1000; % Muestras que se toman para calcular la métrica
if(length(datosRX)>Mmues)
    r=datosRX(1:Mmues);
else
    r=datosRX;
end
L = 48; % Múltiplo dela longitud de una secuencia corta (16*x)
inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
% el indice maximo a evaluar es la longitud de r menos 2L.
M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
m=L-1; % Máximo valor de m
Numerador=0; % Inicializar
Denom=0; % Inicializar
```

```
% Determinar la métrica M(k) para comprobar si es un paquete OFDM:
for k=1:inMax
    Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
    Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
    M(k)=Numerador/Denom; % Métrica
end
% Graficar:
M_cua=(abs(M)).^2; % Magnitud cuadrada de cada valor de la métrica
figure()
subplot(1,2,1);plot(M_cua);title('Métrica M(k)')
subplot(1,2,2);plot(real(datosRecibidos))
title('Parte Real de datos Recibidos')
```

2.5.2.1.1. Detección de trama OFDM en el archivo de texto recibido

El vector de datos de esta etapa es *datosRecortados* y se lo obtiene del script *recortar_rx.m*. Previo a dicho script se debe obtener tener los datos del canal inalámbrico, esto se puede realizar con el script *recibir.m*. Pero para tener continuidad en los ejemplos presentados respecto al procesamiento del archivo de texto recibido, se carga los datos que contiene el archivo *datosRecibidos150.mat* con el comando *load*. Para eliminar del espacio de trabajo cualquier variable que no sea *datosRecortados* se utiliza el siguiente comando.

```
clearvars -except datosRecortados % Borrar todas las variables
% excepto la variable datosRecortados
```

El código de *encontrarTrama_rx.m* inicia por identificar si el tamaño de los datos recortados es mayor al tamaño del bloque que se utiliza para enviar y recibir datos en los equipos SDR, en este caso 800000. En ningún caso los datos pueden superar el tamaño del bloque a enviar, también hay que considerar los 600 datos extras que se pudieron haber aumentado en la etapa de recortado de datos. Si los datos superan el valor del tamaño del bloque más 600, entonces las muestras son inválidas y se publica un mensaje. Esta medida se toma para previnir un posible caso, en donde el equipo SDR de recepción capte señales que provengan de algún otro equipo fuera del prototipo o de otras fuentes. Entonces, si en dado caso el equipo capta otras señales de mayor tamaño que el bloque elegido, esto implica que no son datos enviados por el SDR de transmisión.

Luego se toman las primeras muestras del bloque de datos, esto se realiza debido a que, si se trabajan con todas las muestras, el tiempo de procesamiento se eleva considerablemente. Se supone que el preámbulo está al inicio del bloque por lo que no afecta el tomar solo los primeros datos para el cálculo de la métrica. Se opta por tomar las primeras 1400 muestras, considerando que en la etapa recortado de datos se mantuvieron

los primeros 400 datos antes de superar el umbral por lo que cuando se tiene un buen nivel de la señal el preámbulo estaría a partir de los 400, por lo que se mantienen 1000 muestras mas para asegurarnos que el preámbulo esté en las primeras 1400 muestras. Con el prototipo de comunicación inalámbrica se busca muestrear constantemente el canal y si el procesamiento tarda demasiado no se puede recibir las nuevas muestras tan rápido como se desearía.

Después se define el parámetro L en 48, se calcula el máximo valor que puede tomar k y m, se inicializa la variable que almacena los valores de la métrica. Con un lazo *for* se recorre el índice k de la métrica $M(k)$. Dentro del lazo se implementa la ecuación 2.20 para cada valor de k, el sumatorio del numerador se implementa directamente con una multiplicación de matrices. El sumatorio del denominador se implementa mediante la función *sum*.

Después se obtiene el cuadrado de la magnitud de cada valor del vector de la métrica y se almacena en *M2*. Luego se define un umbral que sirve para detectar la autocorrelación y con esto inferir que el bloque contiene una trama OFDM. El umbral se define en 0.60. Luego se define la variable *tramaEncontrada* en 0, esto indica que aún no se ha encontrado la trama. A continuación, se recorre cada valor del vector *M2* y cuando se encuentre un valor que supere el umbral, se almacena la posición en *auxET*. También se asigna el valor de 1 a la variable *tramaEncontrada*, para indicar que se ha encontrado una trama.

Luego se inicializa el vector *datosOFDM* y si se ha detectado una trama OFDM se almacena los valores de los datos de entrada desde la posición *auxET* hasta el final del vector, este proceso es el primer ajuste en tiempo. Cuando no se ha encontrado ningún valor que supere el umbral y por ende la trama no se ha detectado, se muestra un mensaje que informa al usuario lo propio. Finalmente, si los datos no han superado la longitud máxima del tamaño del bloque y si se ha detectado la trama, se grafica el vector *M2*, la parte real de los primeros datos del vector de entrada y la parte real de la trama detectada.

El código del script *encontrarTrama_rx.m* que se ha descrito en estos párrafos se presenta a continuación, el mismo está debidamente comentado.

```
% Script que permite encontrar la trama OFDM y realizar un ajuste en
% tiempo
% Antes de correr este script es necesario ejecutar el script
% config_SDR.m, recibir.m, recortar_rx.m o bien cargar los datos
% guardados en datosRecibidos150.mat y ejecutar recortar_rx.m
% En vector de entrada de esta etapa es datosRecortados
% DETECCIÓN TRAMA OFDM:
datosRX=datosRecortados; % Copiar los datos en una nueva variable
SamplesPerFrame=800000; % tamaño del bloque recibido
% Los datos enviados nunca pueden ser mayor a SamplesPerFrame+600,
% el 600 se pudo aumentar en la etapa recortado de datos
```

```
c1=1; % Variable de control
if length(datosRX)>(SamplesPerFrame+600)
    disp('***** Muestras recibidas inválidas *****');
    c1=0;
end
% El preambulo se encuentra al inicio de cada paquete y por tanto
% se trabaja con las primeras muestras, con esto se evita
% procesamiento innecesario
Mmues=1400; % Muestras para calcular la métrica
if(length(datosRX)>Mmues)
    r=datosRX(1:Mmues);
else
    % Si es menor a Mmues se trabaja con todas las muestras
    r=datosRX;
end
L = 48; % Múltiplo dela longitud de una secuencia corta (16*x)
inMax = length(r)-2*L+1; % k+m puede tomar valores de hasta 2L-1
% el indice maximo a evaluar es la longitud de r menos 2L.
m=L-1; % Máximo valor de m
M = zeros(Mmues,1); % Inicializar la variable para mayor velocidad
% Determinar la métrica M(k) para comprobar si es un paquete OFDM:
for k=1:inMax
    Numerador = (r(k:k+m)'*r(k+L:k+m+L)); % Sumatorio del numerador
    Denom = sum((abs(r(k+L:k+m+L))).^2); % Sumatorio del denominador
    M(k)=Numerador/Denom; % Métrica
end
M2=(abs(M)).^2; % Cuadrado de la magnitud de cada valor
umbral=0.60; % Umbral para estimar si existio un paquete OFDM, debe
% ser mayor a 0.40
tramaEncontrada=0; % 0 trama no encontrada
auxET=0; % Inicializar variable que almacena la posición en donde
% se superera el umbral de la métrica
for i=1:length(M2)
    %Se obtiene la posición del primer valor por arriba del umbral:
    if(M2(i)> umbral)
        auxET=i;
        tramaEncontrada=1; % Trama encontrada
        break;
    end
end
datosOFDM=[]; % Iniciar variable que contiene la trama OFDM
if(tramaEncontrada==1)
    % Para el caso de que se haya encontrado una paquete OFDM
    datosOFDM(:,1)=datosRX(auxET:end);
else
    % Si no existe un paquete OFDM se empieza de nuevo el bucle while
    disp('***** Trama OFDM NO encontrada *****');
end
% datosOFDM son los datos de salida
% Graficar datos de entrada de la etapa, la métrica y la trama
% encontrada
if (tramaEncontrada==1 && c1==1)
    disp('Trama OFDM detectada');
    figure()
    subplot(3,1,1); plot(M2); title('Métrica M(k)'); hold on
    stem(auxET,M2(auxET),'filled'); hold off
    subplot(3,1,2)
    plot(real(datosRecortados(1:Mmues))); hold on;
    stem(auxET,real(datosRecortados(auxET)),'filled'); hold off
    title('Primeras muestras (parte real)')
```

```
    subplot(3,1,3)
    plot(real(datosOFDM));
    title('Trama detectada y recortada (parte real)')
end
```

Luego de aplicar la etapa Detección de trama OFDM a los datos que contiene la variable *datosRecortados*, se tiene en el espacio de trabajo de Matlab las variables que se muestran en la Figura 2.131 y las gráficas de la Figura 2.132.

El vector de salida de esta etapa es *datosOFDM* y sirve como entrada a la etapa Corrección del CFO.

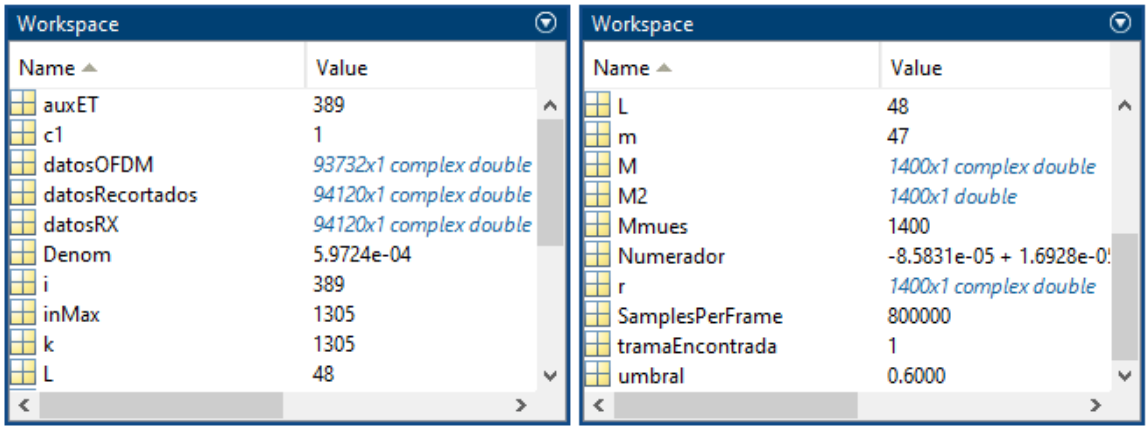


Figura 2.131. Variables en el espacio de trabajo luego de ejecutar *encontrarTrama_rx.m*.

En este caso ya que se detectó la trama OFDM, en la ventana de comandos se obtiene el mensaje *"Trama OFDM detectada"*. En el caso que no se encuentre la trama OFDM se obtiene el mensaje "***** *Trama OFDM NO encontrada* *****" en la ventana de comandos.

En la Figura 2.132, en la primera gráfica se muestra el cuadrado de la magnitud de la métrica $M(k)$ y se identifica con un punto rojo el primer valor en superar el umbral de 0.60. En la segunda gráfica se muestra las primeras 1400 muestras de la parte real del bloque recibido, es decir en base a las que se calculó la métrica, y se identifica con un punto rojo la posición desde donde se recorta el vector de datos. Además, se grafica la parte real de los datos de salida de la etapa Detección trama OFDM.

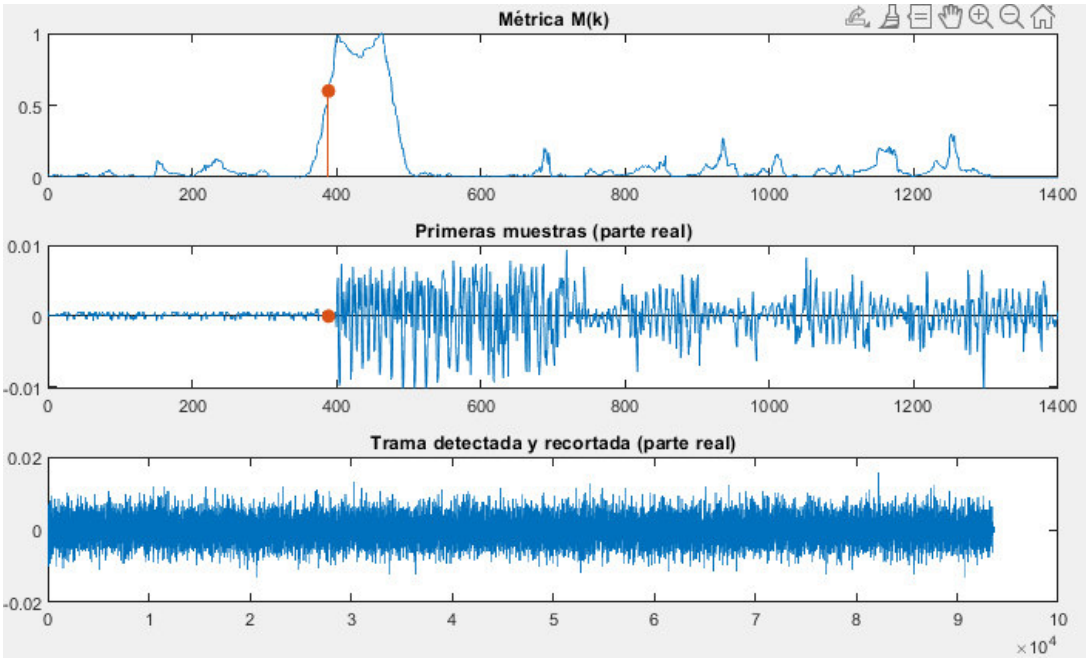


Figura 2.132. Gráfica obtenida del script *encontrarTrama_rx.m*. La primera gráfica es la magnitud de la métrica al cuadrado, la segunda la señal de datos en base a la cual se calculó la métrica y el punto rojo señala la posición desde donde se toma los datos de salida de esta etapa. La tercera gráfica corresponde a la parte real de los datos de salida de la etapa.

2.5.2.2. Corrección del CFO

La CFO (Carrier Frequency Offset) o desplazamiento de frecuencia de las portadoras, se produce en los símbolos OFDM cuando pasan a través del canal inalámbrico. La señal ODFM deja de ser ortogonal cuando los símbolos experimentan un desplazamiento de frecuencia superior a la mitad del ancho de banda de la subportadora. Si se pierde la ortogonalidad se produce ICI en la señal.

La señal recibida $r(t)$ se puede representar como:

$$r(t) = s(t)e^{j2\pi\Delta_f/f_s} \tag{2.21}$$

En donde,

- $s(t)$ es la señal transmitida
- Δ_f es el desplazamiento de frecuencia
- f_s es la frecuencia de muestreo

Teniendo en cuenta que en tiempo las dos mitades de una secuencia corta son idénticas, se puede calcular la diferencia de fase entre estas. El desplazamiento de frecuencia es el resultado directo de la diferencia de fase entre las mitades de las secuencias cortas del preámbulo [4]. Para un desplazamiento de frecuencia Δ_f la diferencia de fase $\emptyset$, entre las mitades de los símbolos del preámbulo, se pueden relacionar directamente por:

$$\emptyset = \frac{2\pi L \Delta_f}{f_s} \tag{2.22}$$

En donde L es la longitud de una secuencia corta. Despejando el desplazamiento de frecuencia se tiene:

$$\Delta_f = \frac{f_s \emptyset}{2\pi L} \tag{2.23}$$

La diferencia de fase $\emptyset$ se puede representar como el ángulo de fase del valor que toma $p(k)$, en donde $p(k)$ se puede calcular mediante [4]:

$$p(k) = \sum_{m=0}^{L-1} r^*(k+m) r(k+m+L) \tag{2.24}$$

- r es la señal recibida.
- $*$ es la conjugación compleja.
- L es la longitud de una secuencia corta, es decir 16.
- k es el índice de p y puede tomar valores máximos a múltiplos de 16. El valor máximo de k se va a representar con la letra ***N***.

$p(k)$ devuelve un número complejo en donde el ángulo de fase de dicho número representa a la diferencia de fase $\emptyset$. Esta diferencia de fase toma valores entre -π y π. Entonces para calcular el CFO se debe utilizar la ecuación 2.23 y la ecuación 2.24.

Cuando ya se obtiene el valor del desplazamiento de frecuencia, el mismo se puede corregir mediante un objeto del sistema de Matlab llamado *comm.PhaseFrequencyOffset*. Este objeto se detalló en la sección 2.4.2.

La corrección del CFO se realiza en base a la parte corta del preámbulo IEEE 802.11a que se presentó en la **Figura 2.96.** La sección corta se compone de 10 secuencias cortas.

2.5.2.2.1. Simulación de estimación del CFO

Antes de implementar la corrección del CFO en los datos OFDM recibidos del canal inalámbrico, es necesario simular un desplazamiento en frecuencia y luego calcular el mismo mediante la ecuación 2.23 y ecuación 2.24. Esto se realiza para comprobar que el código de estimación del CFO trabaje correctamente. Para luego poder aplicar la corrección del CFO mediante el objeto del sistema *comm.PhaseFrequencyOffset*.

En la ecuación 2.24 el valor máximo de *k* debe ser un múltiplo de 16, pero no se sabe con cuál valor se obtiene un mejor resultado. Para determinar esto, se desarrolla el ejemplo en

donde se compara valores máximos de *k* desde 16 hasta 160 en múltiplos de 16. Es decir, en donde ***N*** sea igual a 16, 32, 48, 64, 80, 96, 112, 128, 144 y 160.

El código de simulación y estimación del desplazamiento de frecuencia se implementa en un script de nombre *EjemploCFO.m*. Este script se ha dividido por secciones, con el objetivo de explicar los pasos previos antes de implementar los ejemplos de estimación del CFO. En la **sección uno** se obtiene un vector llamado *RxEj* que inicia con un relleno de 400 ceros, seguido de 4600 muestras que incluye el preámbulo 802.11a y algunos símbolos OFDM. El vector *RxEj* tiene una longitud de 5000 y se utilizará para simular un desplazamiento de tiempo. Para ejecutar esta primera sección se debieron crear los scripts de la parte de transmisión que procesan el archivo de texto *ArchivoTextoRT.txt*. El código se presenta a continuación debidamente comentado.

```
%% Sección 1: Obtener los datos de entrada
abrir_archivo()      % Abir archivo, mediante un cuadro de diálogo
serializar_datos()   % Serializar datos
cod_conv()           % Codificador COnvolucional
mapeo_datos()        % Mapeo QPSK
reformado_tx()       % Filtro SRRC de transmisión
ofdm_tx()            % Creación del los símbolos OFDM
preambulo_tx()       % Agregar el preambulo 802.11a
close all % Cerrar las gráficas que se abren en los scripts anteriores
RxEj=[zeros(400,1);datosConPre(1:4600)]; % Agregar un relleno de ceros
% al inicio para simular el desfase de tiempo
clearvars -except RxEj % Borrar todas las variables excepto RxEj
% Graficar los datos obtenidos:
figure()
plot(real(RxEj))
title('Preámbulo 802.11a y símbolos OFDM')
```

Luego de ejecutar la sección 1 del script *EjemploCFO.m*, se obtiene como resultado la variable *RxEj* en el espacio de trabajo. Esta variable es un vector columna de longitud 5000 y contiene datos complejos. Además, se obtiene la gráfica de la Figura 2.133, en donde se muestra la parte real del vector que contiene un relleno inicial de ceros, el preámbulo 802.11a y algunos símbolos OFDM. Se toma solamente los primeros símbolos OFDM ya que el objetivo de este ejemplo es estimar el desplazamiento de frecuencia.

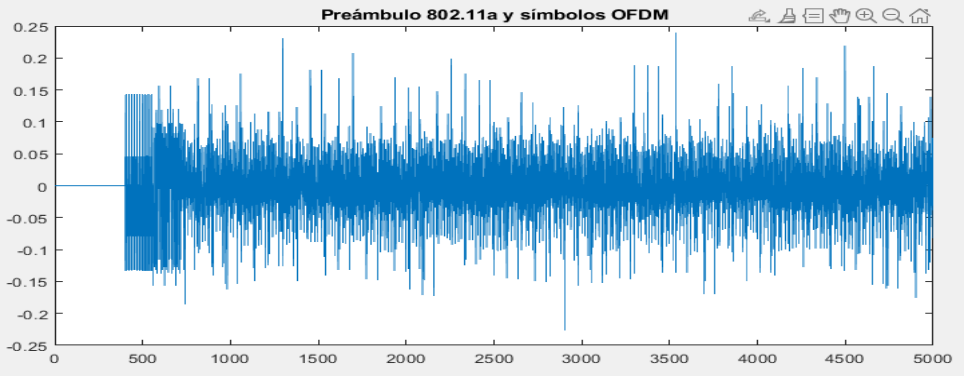


Figura 2.133. Parte real de *RxEj*.

En la sección dos del script *EjemploCFO.m* se implementa un desplazamiento en frecuencia a los datos contenidos en la variable *RxEj*. Este desplazamiento es de 667 Hz y se inserta con el objeto del sistema *comm.PhaseFrequencyOffset*. Además, es necesario especificar la frecuencia de muestreo de la señal, esta frecuencia **se estableció en el capítulo configuración y es igual a 2Msps**. La señal que ya tiene el desplazamiento de frecuencia se almacena en el vector *datosRecortados*.

Luego se ejecuta el script *encontrarTrama_rx.m* para que se detecte la trama OFDM y se realice una corrección en tiempo. El vector de entrada de dicho script es *datosRecortados* y el vector de salida es *datosOFDM*. En este punto aun no se conoce el inicio exacto de la señal, pero se espera que esté entre los 10 primeros valores. Después se cierra las figuras que pudieran estar abiertas, también se borra las variables del espacio de trabajo excepto las más importantes.

Para implementar la ecuación 2.23 y la ecuación 2.24 se utiliza la misma frecuencia de muestreo ya antes definida. También se copia los datos de *datosOFDM* en una variable *r*, se define *L* igual a 16 y en *m* se almacena el máximo valor que puede tomar este parámetro, es decir *L-1*. Con un lazo *for* se recorre los valores de ***N*** de 16 a 160, en múltiplos de 16, recordando que *N* representa el valor máximo de *k*.

Ya dentro del lazo *for* se inicializa con ceros la variable que almacena la estimación de frecuencia. Con otro lazo *for* se recorre los valores de *k* de 1 hasta *N*, para calcular la p y con esto la estimación en frecuencia. Entonces, dentro de este segundo lazo *for* se puede calcular la ecuación 2.24 de manera directa, ya que se puede obtener la transposición conjugada compleja del vector y luego realizar una multiplicación, tal que se multiplica y se suma cada valor de los vectores, este resultado se almacena en *p*. Luego se calcula el desplazamiento de frecuencia (ecuación 2.23) para cada valor de *k* en base a la diferencia de fase ∅ del valor que contiene *p* y se almacena en *fEst(k)*. La diferencia de fase de *p* se calcula con la función *angle()*.

Cuando se sale del segundo lazo *for*, se calcula el promedio de *fEst* y se invierte de signo ya que luego se deberá corregir el desplazamiento de frecuencia introducido. También se muestra el valor de *N* y el valor de la estimación de frecuencia con el signo contrario. Esto hasta completar todos los valores de *N*.

El código de la sección dos del script *EjemploCFO.m* se presenta a continuación debidamente comentado.

```
%% Sección 2: Ejemplo de estimación del CFO sin ruido
% El vector de datos de entrada se llama RxEj, se obtiene en la
sección1
Fs=2.0e6; % frecuencia de muestreo
% Asignar a la señal de entrada un desplazamiento de frecuencia
senialDesf = comm.PhaseFrequencyOffset('FrequencyOffset',677,...
    'SampleRate',Fs);
datosRecortados=senialDesf(RxEj); % Señal desplazada y de entrada al
% script encontrarTrama_rx.m
encontrarTrama_rx() % Los datos de salida son datosOFDM
close all % Cerrar las figuras abiertas
clearvars -except RxEj datosOFDM Fs datosRecortados senialDesf % Borrar
% todas las variables excepto ... las especificadas
% La variable r se la usa para reducir el tamaño del codigo
r = datosOFDM; % Señal recibibida para estimar el CFO
L=16; % Longitud de una secuencia corta
m=L-1; % Máximo valor de m
disp('****************************************')
% Se recorre diferentes valores de N, este valor es múltiplo de 16 y
% especifica el máximo valor que va a tomar k. Esto se hace con el
% objetivo de encontrar el mejor valor de N
for N=16:16:160
    fEst = zeros(N,1); % Iniciar vector que almacena el desplazamiento
    % Implementar el sumatorio:
    for k=1:N
        % Implementar el sumatorio para p(k):
        p = r(k:k+m)'*r(k+L:k+m+L);
        % Calcular el desplazamiento en frecuencia, con angle() se
        % calcula la diferencia de fase
        fEst(k) = (Fs/(2*pi*L))*(angle(p));
    end
    % Obtenemos el promedio del CFO estimado
    estFreq=-mean(fEst);
    fprintf('N=%3.0f, Estimación= %f\n',N,estFreq) % Mostrar
end
disp('****************************************')
```

Una vez ejecutada la sección de código anterior se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.134. Mientras que en la ventana de comandos se obtiene el resultado que se presenta en la Figura 2.135.

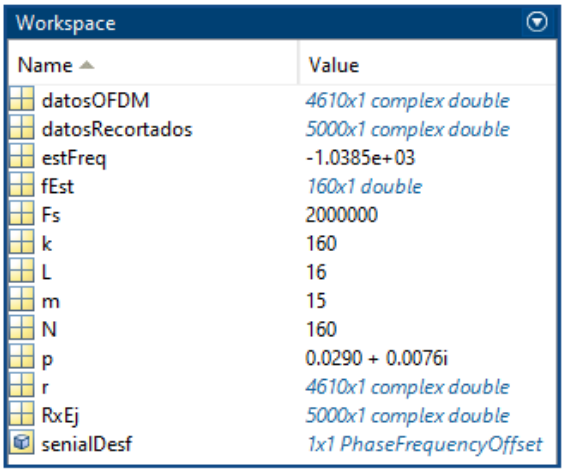


Figura 2.134. Variables en el espacio de trabajo luego de ejecutar la segunda sección de *EjemploCFO.m*.

Cuando no se tiene ruido se observa que de *N* igual a 16 hasta *N* igual a 128, la estimación del desplazamiento de frecuencia es correcta, ya que se introdujo un desplazamiento de 677 Hz, mientras que los valores de *N* de 144 y 160 fallan en la estimación.

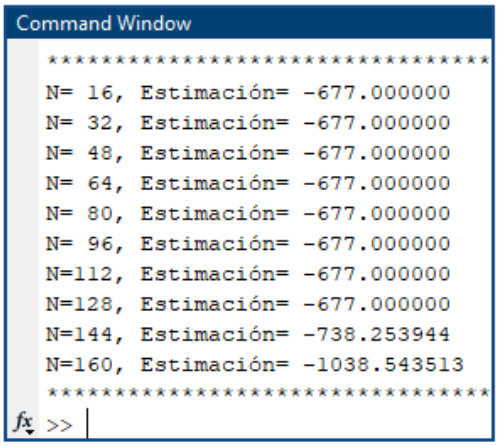


Figura 2.135. Resultado en la ventana de comandos después de ejecutar la segunda sección de *EjemploCFO.m*. Cualquier valor de N hasta 128 entrega el valor correcto de CFO (677 Hz)

El canal inalámbrico presenta ruido por lo que en la sección tres del script *EjemploCFO.m* se implementa la estimación del desplazamiento de frecuencia, pero en una señal con ruido. El código de la sección tres es similar al código de la sección dos. La diferencia es que se introduce un desplazamiento de frecuencia de 7000 Hz. Después de desplazar en frecuencia la señal de entrada, se introduce ruido blanco gaussiano mediante la función de Matlab *awgn*. Además, el valor de *N* va desde 16 a 128 en saltos de 16, descartando los valores de 144 y 160 que se utilizaron en la sección dos del script *EjemploCFO.m*. A continuación, se presenta el código de la sección tres comentado.

```
%% Sección 3: Ejemplo de estimación del CFO con ruido
% El vector de datos de entrada se llama RxEj, se obtiene en la
sección1
Fs=2.0e6; % frecuencia de muestreo
```

```
% Asignar a la señal de entrada un desplazamiento de frecuencia
senialDesf = comm.PhaseFrequencyOffset('FrequencyOffset',7000,...
    'SampleRate',Fs);
datosSR=senialDesf(RxEj); % Señal desplazada
datosRecortados= awgn(datosSR,20,'measured'); % Señal con ruido,
entrada
% al script encontrarTrama_rx.m
encontrarTrama_rx() % Los datos de salida son datosOFDM
close all % Cerrar las figuras abiertas
% Borrar todas las variables excepto las especificadas:
clearvars -except RxEj datosOFDM Fs datosRecortados senialDesf datosSR
% La variable r se la usa para reducir el tamaño del codigo
r = datosOFDM; % Señal recibibida para estimar el CFO
L=16; % Longitud de una secuencia corta
m=L-1; % Máximo valor de m
disp('****************************************')
% Se recorre diferentes valores de N, este valor es múltiplo de L y
% especifica el máximo valor que va a tomar k
for N=16:16:128
    fEst = zeros(N,1); % Iniciar vector que almacena el desplazamiento
    for k=1:N
        p = r(k:k+m)'*r(k+L:k+m+L);
        fEst(k) = (Fs/(2*pi*L))*(angle(p));
    end
    % Obtenemos el promedio del CFO estimado
    estFreq=-mean(fEst);
    fprintf('N=%3.0f, Estimación= %f\n',N,estFreq) % Mostrar
end
disp('****************************************')
```

Luego de ejecutar la sección de código anterior, se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.136.

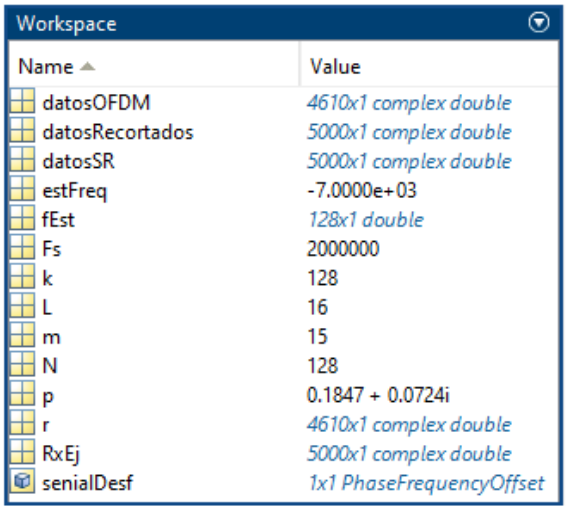


Figura 2.136. Variables en el espacio de trabajo luego de ejecutar la tercera sección de *EjemploCFO.m*.

La estimación del desplazamiento de frecuencia cuando se agrega ruido aditivo puede cambiar en cada ejecución del código. En la Figura 2.137 se muestra el resultado en la ventana de comandos en dos ejecuciones distintas. Los resultados son diferentes, pero se

mantienen alrededor de -7000, y ya que se introdujo un desplazamiento de 7000 Hz, el resultado es correcto.

Para desarrollar el ejemplo de la corrección CFO en los datos recuperados del canal inalámbrico se utilizará *N*=128.

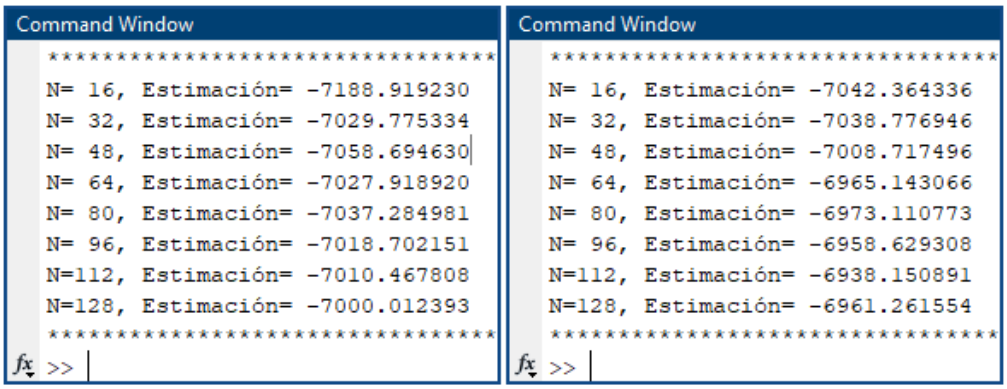


Figura 2.137. Resultados en la ventana de comandos después de ejecutar la tercera sección de *EjemploCFO.m*. Como se observa, N = 128 entrega el resultado más exacto (CFO = 7000 Hz).

2.5.2.2.2. Corrección del CFO en el archivo de texto recibido

El vector de entrada de esta etapa es *datosOFDM* y se obtiene del script *encontrarTrama_rx.m*, luego de haber cargado los datos recibidos del canal inalámbrico y recortado los datos con el script *recortar_rx.m*. Para limpiar las variables del espacio de trabajo y dejar únicamente el vector *datosOFDM*, se utiliza el comando que se muestra a continuación.

```
clearvars -except datosOFDM % Borrar todas las variables
% excepto la variable datosOFDM (entrada corr CFO)
```

En un script llamado *corrCFO_rx.m* se implementa el código para estimar y corregir el desplazamiento de frecuencia que introduce el canal inalámbrico. El código inicia por estimar el desplazamiento de frecuencia, este proceso ya se explicó anteriormente. Al desplazamiento obtenido se lo multiplica por -1 y se almacena en la variable ***estFreq***. Luego con la frecuencia de muestreo de la señal (configurada en los SDR) y con *estFreq*, se crea el objeto para corregir el desplazamiento de frecuencia y se almacena en *despFreqObj*. Finalmente, a los datos de entrada, contenido en el vector *datosOFDM,* se aplica la corrección del CFO con el objeto *despFreqObj* y los datos corregidos se almacena en el vector *datosCorrCFO*. El código descrito en este párrafo se presenta a continuación debidamente comentado.

```
% Script que permite estimar y corregir la CFO
% Antes de correr este script es necesario cargar los datos
% guardados en datosRecibidos150.mat, ejecutar recortar_rx.m,
```

```
% encontrarTrama_rx.m
% En vector de entrada de esta etapa es datosOFDM
% CORRECCIÓN DEL CFO:
r = datosOFDM; % Copiar los datos en r para reducir el nombre
Fs=2.0e6; % Frecuencia de muestreo con la que se recibió los datos
N = 128; % máximo valor del indice k
L=16; % longitud d euna secuencia corta
m=L-1; % Máxima longitud que puede tomar m
fEst = zeros(N,1); % Iniciar vector que almacena la estimación de freq
% Lazo que recorre de k igual a 1 hasta N
for k=1:N
    P = r(k:k+m)'*r(k+L:k+m+L); % Sumarotio p(k)
    fEst(k) = (Fs/(2*pi*L))*(angle(P)); % Calcular la estimación
    % de frecuencia en la posición k
end
estFreq= -mean(fEst); % Negativo del promedio del CFO estimado
%Corrección de frecuencia con la estimación realizada:
despFreqObj = comm.PhaseFrequencyOffset('FrequencyOffset',...
    estFreq,'SampleRate',Fs);
% Aplicar la estimacion de CFO a las muestras recibidas para
% corregir el CFO:
datosCorrCFO=despFreqObj(datosOFDM); % Datos corregidos el CFO
```

Luego de ejecutar el script *corrCFO_rx.m* se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.138. Se encierra con verde el negativo del desplazamiento de frecuencia estimado de los datos recibidos. Además, el vector de salida de esta etapa es *datosCorrCFO*.

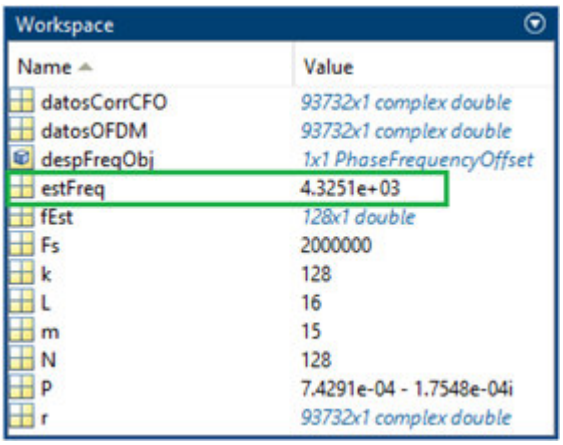


Figura 2.138. Variables en el espacio de trabajo luego de ejecutar *corrCFO_rx.m*.

2.5.2.3. Sincronización de símbolo

La etapa Sincronización de símbolo permite detectar el inicio exacto de la trama OFDM, es decir se corrige el tiempo de los símbolos recibidos. Esta etapa utiliza la sección larga (LLTF) del preámbulo OFDM 802.11a. Para determinar el inicio de la trama OFDM se utiliza la técnica de correlación cruzada entre la sección larga y la señal recibida. En la ecuación 2.25 se define la métrica c que calcula el valor absoluto al cuadrado de la correlación cruzada entre la señal d_{LLTF} y la señal r [4].

$$c(k) = \left| \sum_{m=0}^{L-1} d_{LLTF}^{*}(m) r(m+k) \right|^{2} \qquad (2.25)$$

En donde:

- d_{LLTF} son las primeras 80 muestras de la sección larga del preámbulo.
- $*$ es la conjugación compleja
- r es la señal recibida
- L es la longitud de d_{LLTF}, es decir 80.
- k es el índice de la métrica c

La **sección larga** del preámbulo 802.11a está conformada por un prefijo cíclico de las últimas 32 muestras de una secuencia larga y de dos secuencias largas (**Figura 2.139**). Debido a que d_{LLTF} son las primeras 80 muestras de la sección larga del preámbulo, cuando se calcula la correlación cruzada se espera que exista un pico en la primera posición de la sección larga. El segundo pico se espera que aparezca 64 posiciones más adelante, es decir en la posición 65. La diferencia entre las posiciones del primer y segundo pico es de 64. En la Figura 2.139 se identifica el primer pico con color negro en la parte real e imaginaria de la sección LLTF, el segundo pico se identifica con color verde. Además, se acota con color negro las primeras 80 muestras de la sección LLTF que genera el primer pico y con color verde las 80 muestras que generan el segundo pico.

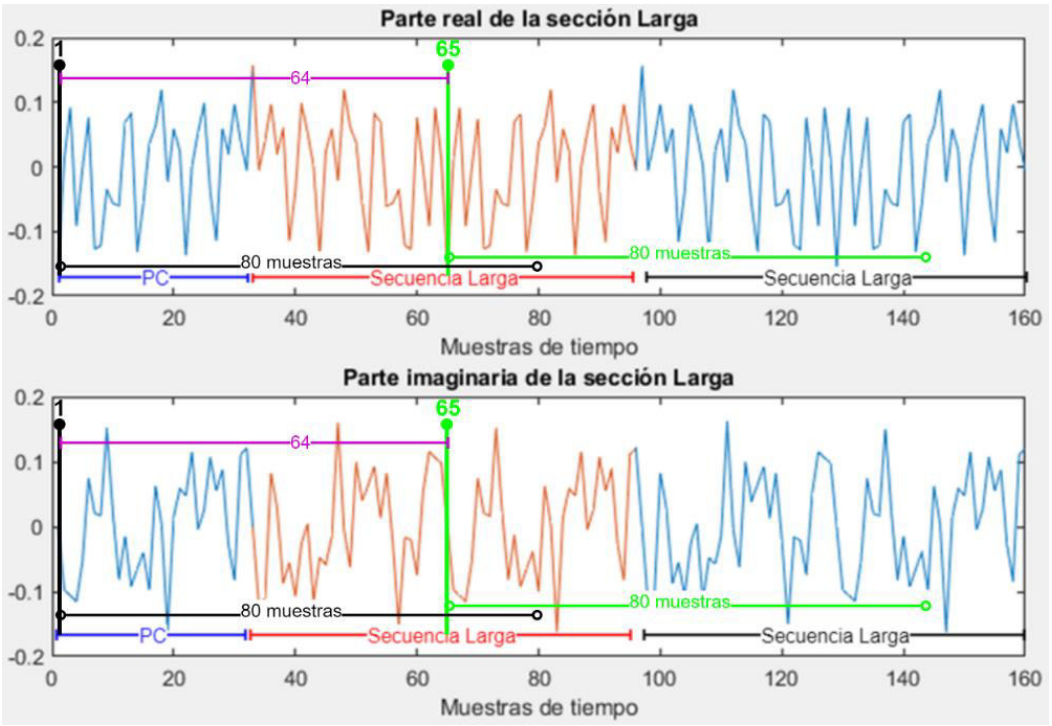


Figura 2.139. Parte LLTF del preámbulo 802.11a, con los picos que se deben generar ubicados en las posiciones correspondientes.

Matlab cuenta con una función directa que permite calcular la correlación cruzada de dos señales. Esta función se llama *xcorr* y la sintaxis que utiliza es la siguiente:

datCorr = xcorr(r,d)

La función *xcorr* devuelve la correlación cruzada de dos secuencias de tiempo discreto [38]. Cuando los vectores de entrada no tienen la misma longitud, la función agrega ceros al final del vector de menor longitud, para obtener dos vectores de igual longitud. Para el prototipo que se está desarrollando el vector de entrada *r* es la señal recibida y el vector *d* son las primeras 80 muestras de la parte LLTF del preámbulo. En *datCorr* se almacena la correlación cruzada y tiene una longitud de 2N-1, en donde N en la longitud del vector *r*. Además, los valores de correlación válidos se encuentran desde N hasta el final del vector. Para terminar de implementar la ecuación 2.25 se debe calcular la magnitud y elevar al cuadrado cada elemento del vector *datCorr*.

Para implementar un ejemplo de la sincronización de símbolo mediante la correlación cruzada se implementa el script *EjemploSincro.m*. El cual inicia por obtener los datos a enviar con el preámbulo, estos datos están almacenados en el vector *datosConPre*. Luego en el vector ***datSinCFO*** se almacenan los primeros 3000 datos del vector *datosConPre* y se agrega un relleno inicial de 40 ceros, esto ya que solo se quiere demostrar cómo funciona la sincronización de símbolo y no son necesarios mas datos porque el preámbulo está contenido dentro de los primeros 3000 datos. En este punto el primer dato del preámbulo está en la posición 41 y el primer dato de la secuencia larga se encuentra en la posición 201 del vector *datSinCFO*. Para que el espacio de trabajo de Matlab no se sature con variables que no influyen en esta etapa, se eliminan todas las variables excepto *datSinCFO*.

Luego se crea la sección larga (LLTF), como se vio en la parte de adición del preámbulo 802.11a en el capítulo de transmisión. Se almacenan los primeros 80 datos del LLTF en un vector de nombre *secLLTF80*. Con la función ***xcorr*** se aplica la correlación cruzada de la señal de entrada con el vector *secLLTF80* y además se calcula el cuadrado de cada magnitud. Después se eliminan las primeras N-1 muestras del vector obtenido, ya que esos datos corresponden al relleno que introduce *xcorr* y los datos se almacenan en un vector llamado *datosCorr*. Finalmente, se grafica la parte real de los datos de *datSinCFO* normalizados y se sobrepone los datos de *datosCorr*. El código descrito en estos párrafos se presenta a continuación.

```
% Ejemplo de sincronización de símbolo
% Obtener los datos de entrada:
abrir_archivo()     % Abir archivo, mediante un cuadro de diálogo
```

```
serializar_datos()  % Serializar datos
cod_conv()          % Codificador COnvolucional
mapeo_datos()       % Mapeo QPSK
reformado_tx()      % Filtro SRRC de transmisión
ofdm_tx()           % Creación del los símbolos OFDM
preambulo_tx()      % Agregar el preambulo 802.11a
close all % Cerrar las gráficas que se abren en los scripts anteriores
datSinCFO=[zeros(40,1);datosConPre(1:3000)]; % Recortar los datos y
% agregar un relleno de ceros
% al inicio para simular el desfase de tiempo
clearvars -except datSinCFO % Borrar todas las variables excepto RxEj
% Crear la parte larga del preámbulo 802.11a:
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
LLTF=[parteLargaTime(33:64);parteLargaTime;parteLargaTime]; % LLTF
secLLTF80 = LLTF(1:80); % Primeras 80 muestras de la LLTF
% Cuadrado del valor absoluto de la orrelación cruzada entre
% la señal recibida y secLLTF80:
datosCorrRell = (abs(xcorr(datSinCFO,secLLTF80))).^2;
% Se corrige en tiempo los datos correlacionados, para que
% coincidan con las posiciones de los datos de entrada:
datosCorr = datosCorrRell(length(datSinCFO):end);
% Graficar los datos y la correlación:
figure()
plot(real(datSinCFO)/max(real(datSinCFO)))
hold on
plot(datosCorr)
hold off
xlim([0 500])
title('Sincronización de símbolo')
legend('Señal recibida','Correlación cruzada')
```

Luego de ejecutar el script *EjemploSincro.m* se obtiene las variables en el espacio de trabajo que se muestra en la Figura 2.140 y la gráfica de la Figura 2.141.

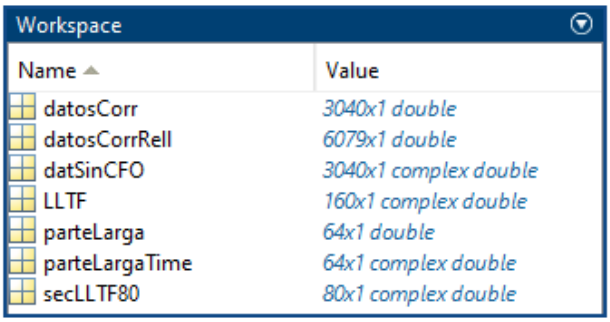


Figura 2.140. Variables en el espacio de trabajo luego de ejecutar *EjemploSincro.m*.

En la Figura 2.141 se muestra con azul la parte real de la señal almacenada en *datSinCFO* y con rojo la magnitud al cuadrado de la correlación cruzada. Tal como se esperaba el

primer pico se encuentra en la **posición 201** y el segundo pico 64 muestras después, es decir en la posición 265. El primer valor de la **sección corta (LSTF)** se ubicaría en la posición 41, esto se consigue de la resta de la posición de del primer pico (201) menos la longitud de la sección corta (160). Recordando que se agregaron 40 ceros de relleno al inicio de los datos, el valor inicial de la sección corta sí se encuentra en la posición 41.

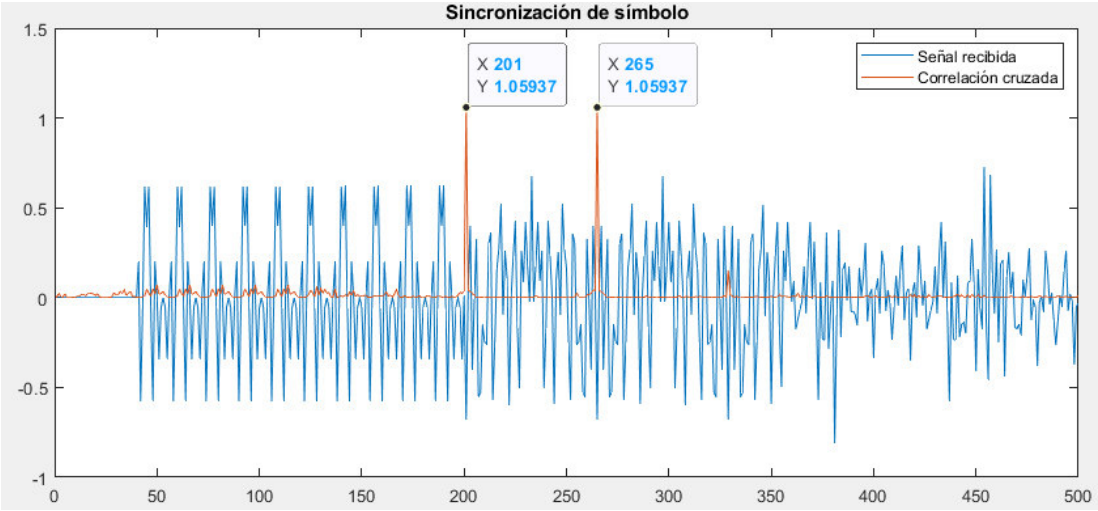


Figura 2.141. Gráfica de la simulación de una señal recibida y del cuadrado de la magnitud de correlación cruzada.

2.5.2.3.1. Sincronización de símbolo en el archivo de texto recibido

Los datos de entrada para implementar la etapa Sincronización de símbolo, se encuentran en el vector *datosCorrCFO* y se obtienen del script *corrCFO_rx.m* que corresponde a la etapa Corrección CFO. Para obtener el vector *datosCorrCFO* es necesario cargar los datos recibidos del canal inalámbrico, ejecutar el script *recortar_rx.m*, el script *econtrarTrama_rx.m* y el script *corrCFO_rx.m*. Para eliminar todas las variables del espacio de trabajo se utiliza el siguiente comando.

```
clearvars -except datosCorrCFO % Borrar todas las variables
% excepto la variable datosCorrCFO (entrada SincSimbolo_rx)
```

Para implementar la etapa Sincronización de símbolo a los datos corregidos en fase se crea el script *SincSimbolo_rx.m*. El código inicia por almacenar los primeros 5000 datos del vector *datosCorrCFO* en un vector llamado *datSinCFO*, esto con el objetivo de reducir los datos a procesar ya que el preámbulo está en las primeras muestras, con un tamaño de 5000 se obtiene un rendimiento adecuado del procesamiento. Luego se crea una secuencia larga en el dominio de la frecuencia y en base a esta se crea la sección LLTF en el dominio del tiempo. Después de toman los primeros 80 datos del LLTF y se almacena en la variable *secLLTF80*.

Luego se calcula el cuadrado de cada magnitud de la correlación cruzada de *secLLTF80 y datSinCFO*. La correlación cruzada se calcula con la función *xcorr* y devuelve un vector de tamaño *2N-1*, en donde *N* es la longitud del vector *datSinCFO*. Después se elimina el relleno que introduce la función *xcorr*, se eliminan los primeros *N-1* elementos y se almacena en una variable llamada *datosCorr*.

A continuación, se obtienen las posiciones de los dos picos máximos del vector *datosCorr* y con un condicional *if* se evalúa si la diferencia entre las posiciones es 64. Cuando la diferencia no sea 64 se descarta la trama recibida ya que no se pudo determinar el inicio exacto de la misma. En caso de que sea 64 se calcula el inicio exacto del preámbulo, el cual se encuentra 160 muestras antes del primer pico. Paso seguido se evalúa si el inicio es mayor a 0, cuando no lo es significa que tampoco se ha podido determinar el inicio exacto de la trama. En caso de que el inicio sea mayor a cero, se guardan los datos con el primer valor del preámbulo en la posición uno de un vector llamado ***datosSincronizados***.

Cuando no se haya podido encontrar el inicio exacto de la trama OFDM se publica un mensaje informando lo propio al usuario. Si se ha podido encontrar el inicio exacto, además de sincronizar los datos, se grafica los primeros 500 datos de la parte real del vector *datosSincronizados*. El código de *SincSimbolo_rx.m* se presenta a continuación, el mismo está debidamente comentado.

```
% Script que permite realizar la sincronización de símbolo
% Antes de correr este script es necesario cargar los datos
% guardados en datosRecibidos150.mat, ejecutar recortar_rx.m,
% encontrarTrama_rx.m, corrCFO_rx.m
% En vector de entrada de esta etapa es datosCorrCFO
% SINCRONIZACIÓN DE SÍMBOLO:
datSinCFO=datosCorrCFO(1:5000); % Se toma los primeros 5000 datos
% para reducir el tiempo de procesamiento
% Crear la parte larga del preámbulo 802.11a:
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
LLTF=[parteLargaTime(33:64);parteLargaTime;parteLargaTime]; % LLTF
secLLTF80=LLTF(1:80); % Primeras 80 muestras de la LLTF
% Cuadrado del valor absoluto de la orrelación cruzada entre
% la señal recibida y secLLTF80:
datosCorrRell=(abs(xcorr(datSinCFO,secLLTF80))).^2;
% Se corrige en tiempo los datos correlacionados, para que
% coincidan con las posiciones de los datos de entrada:
datosCorr=datosCorrRell(length(datSinCFO):end);
% Almacenar la posición de los dos picos:
[~,posPico1]=max(datosCorr); % Posición del primer pico
```

```
datosCorr(posPico1)=0; % Eliminar el primer pico para poder
% encontrar el segundo
[~,posPico2]=max(datosCorr); % Posición del segundo pico
% Se comprueba que la distancia entre los dos picos sea 64
if abs(posPico2-posPico1)==64
    % Se encuentra el inicio real del LTTF y de LSTF
    inicioLLTF=min([posPico1 posPico2]); % Se toma la posición
    % menor y esa es la ubicación del primer dato del LLTF
    inicioLSTF=inicioLLTF-160; % Determinar el inicio del preambulo
    % Comprobar que la posición inicial sea mayor a 0
    if (inicioLSTF>0)
        % Recortar los datos
        datosSincronizados=datosCorrCFO(inicioLSTF:end); % Datos
        % de salida
        %Graficar:
        figure()
        plot(real(datosSincronizados(1:500)))
        title('Parte real de los datos sincronizados')
    else
        disp('Falla en la sincronizacion de simbolo (2)')
    end
else
    disp('Falla de sincronización de símbolo')
end
% datosSincronizados es los datos de salida
```

Luego de ejecutar el script *SincSimbolo_rx.m* se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.142 y la gráfica de la Figura 2.143. El vector de salida de la etapa Sincronización de símbolo es *datosSincronizados*.

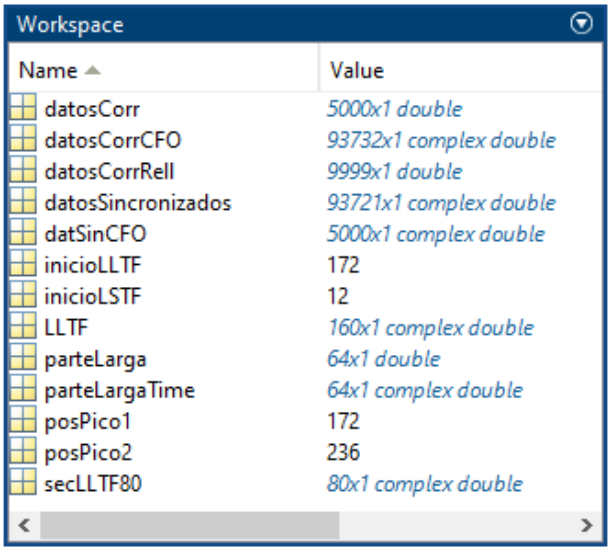


Figura 2.142. Variables en el espacio de trabajo luego de ejecutar *SincSimbolo_rx.m*.

En la Figura 2.143 se muestra la parte real de los primeros 500 datos del vector con la señal recibida sincronizada en tiempo, los primeros 320 corresponden al preámbulo IEE 802.11a.

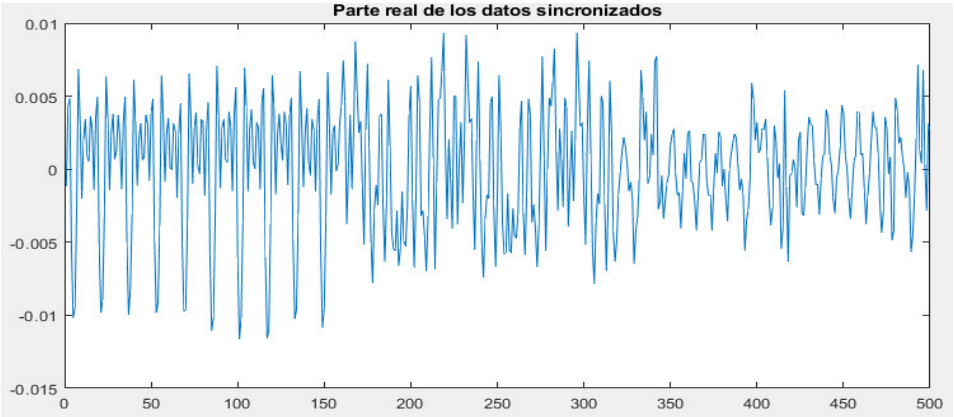


Figura 4.143. Parte real de los primeros datos sincronizados en tiempo.

2.5.2.4. Corrección de fase

En esta etapa se utiliza la **sección LLTF** del preámbulo 802.11a para estimar y corregir el desplazamiento en fase que se ha introducido en la señal de datos al pasar por el canal inalámbrico. En el prototipo que se está desarrollando en este trabajo, en una etapa posterior también se implementa la ecualización. La ecualización busca eliminar cualquier efecto residual de fase o de frecuencia que se presente en la señal recibida, esta sección se implementa luego de la demodulación de los símbolos OFDM [4]. En la ecualización se corrige cada símbolo mediante las portadoras piloto, mientras que en la etapa actual se utiliza la sección larga del preámbulo para corregir la fase de toda la señal recibida.

En la Figura 2.144 se muestra la parte real e imaginaria, en el dominio de la frecuencia, de una **secuencia larga** del preámbulo 802.11a. La parte imaginaria tiene valores prácticamente de 0 y los valores de la parte real toma valores de 0, 1 y -1. Entonces, cuando la parte real es igual a 1 y se obtiene el ángulo entre la componente real e imaginaria, el resultado será un ángulo de 0°.

Si se introduce un desplazamiento de fase, el ángulo que antes era 0° pasa a ser igual al desplazamiento introducido.

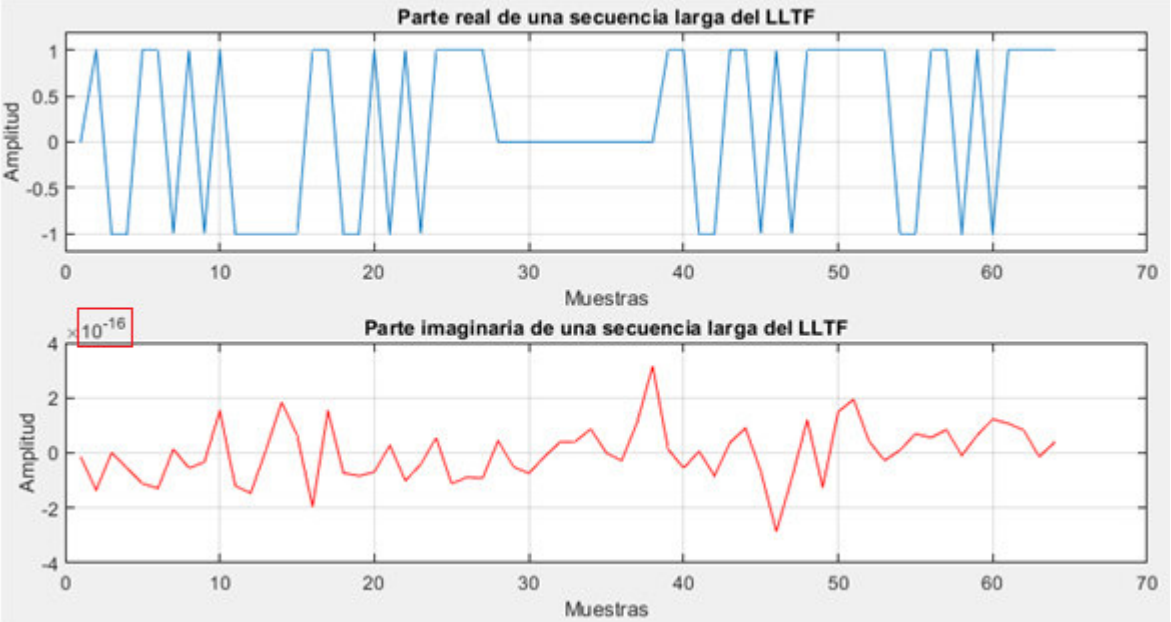
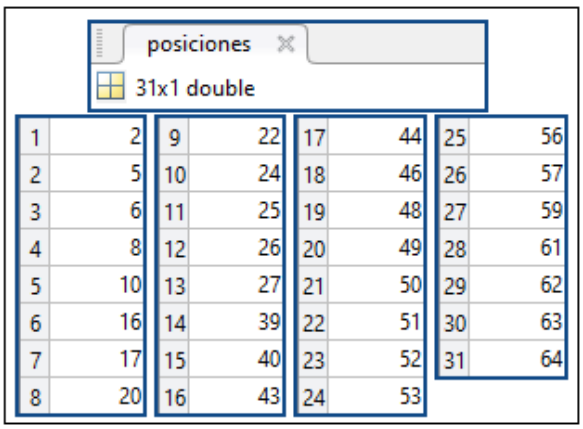


Figura 2.144. Parte real e imaginaria de una secuencia larga en el dominio de la frecuencia. La parte imaginaria tiene valores prácticamente de 0 y es la clave para poder calcular el desplazamiento de fase.

En la Figura 2.145 se muestran las posiciones de todos los valores reales, de una secuencia larga, que son igual a 1. En total son 31 valores, en donde el ángulo entre la parte real e imaginaria es 0°. Sin embargo, solo se utilizan **los primeros y los últimos 8 valores de cada secuencia larga** en el dominio de la frecuencia, para realizar la estimación de fase [34].



1	2	9	22	17	44	25	56
2	5	10	24	18	46	26	57
3	6	11	25	19	48	27	59
4	8	12	26	20	49	28	61
5	10	13	27	21	50	29	62
6	16	14	39	22	51	30	63
7	17	15	40	23	52	31	64
8	20	16	43	24	53		

Figura 2.145. Posiciones de los valores reales igual a 1 de una secuencia larga.

Entonces, las posiciones de cada **secuencia larga** que se utilizan para realizar la estimación de fase son: 2, 5, 6, 8, 10, 16, 17,20, 53, 56, 57, 59, 61, 62,63 y 64 de cada secuencia larga.

La Figura 2.144 y los datos del vector *posiciones* de la Figura 2.145 se obtuvieron mediante la siguiente una sección de código en Matlab. En esta sección se obtiene una secuencia larga en el dominio de la frecuencia. Luego se guarda las posiciones de la secuencia larga

que son igual a uno, para considerar un margen de error se guardan las posiciones que superan el valor de 0.9. El código descrito en este párrafo se adjunta a continuación debidamente comentado.

```
% Graficar una secuencia larga en el dominio de la frecuencia y
% almacenar las posiciones cuando la secuencia larga es 1.
% Crear la parte larga del preámbulo 802.11a:
% Secuencia larga (parte de LLTF) en el dominio de la frecuencia
parteLarga=[...
    0;1;-1;-1;1;1;-1;1;-1;1;-1;-1;-1;-1;-1;1
    1;-1;-1;1;-1;1;-1;1;1;1;1;0;0;0;0;0
    0;0;0;0;0;0;1;1;-1;-1;1;1;-1;1;-1;1
    1;1;1;1;1;-1;-1;1;1;-1;1;-1;1;1;1;1];
% Conversión del dominio de la frecuencia al tiempo, y adición de PC
% de la sección larga:
parteLargaTime=ifft(parteLarga); % conversión al dominio del tiempo
secLargaF=fft(parteLargaTime); % Conversión del tiempo a frecuencia
j=1; % Inicializar variable
posiciones=[]; % Inicializar vector que almacena las posiciones de los
% valores que son igual a 1 en secLargaF
% Recorrer cada posición de secLargaF y almacenar las que sean mayor
% a 0.9, se considera este valor por un posible margen de error:
for i=1:length(secLargaF)
    if  (real(secLargaF(i)) > 0.9)
        posiciones(j,1)=i;
        j=j+1;
    end
end
% Graficar la parte real e imaginaria de la secuencia larga
figure()
subplot(2,1,1)
plot(real(secLargaF))
xlabel('Muestras'); ylabel('Amplitud')
title('Parte real de una secuencia larga del LLTF')
ylim([-1.2 1.2]);grid on
subplot(2,1,2)
plot(imag(secLargaF),'-r')
xlabel('Muestras'); ylabel('Amplitud')
title('Parte imaginaria de una secuencia larga del LLTF'); grid on
```

2.5.2.4.1. Simulación de la estimación del desplazamiento en fase

Antes de implementar la corrección de fase en los datos OFDM recibidos, es necesario simular un desplazamiento de fase y estimar el mismo para verificar el correcto funcionamiento de la sección de código creada. El desplazamiento de fase se realiza mediante el objeto del sistema *comm.PhaseFrequencyOffset* visto en la sección de corrección del CFO. En un script llamado *EjemploFase.m* se implementan dos ejemplos de estimación del desplazamiento de fase, cada ejemplo se coloca en secciones diferentes dentro del script. El primer ejemplo se realiza en datos sin ruido y el segundo en datos con ruido blanco gaussiano.

La sección uno del script *EjemploFase.m* abre y procesa un archivo de texto con los scripts obtenidos en transmisión. También se cierran las gráficas que se pudieran abrir. El vector que contiene el preámbulo 802.11a seguido de símbolos OFDM es *datosConPre*. Con el objetivo de reducir el procesamiento innecesario, se toman las primeras 3000 muestras de dicho vector y se almacenan en una variable llamada *datosSinc*. Además, se borran todas las variables excepto *datosSinc*. El código de la sección uno se presenta a continuación.

```
%% Sección 1: Obtener los datos de entrada
abrir_archivo()      % Abir archivo, mediante un cuadro de diálogo
serializar_datos()   % Serializar datos
cod_conv()           % Codificador COnvolucional
mapeo_datos()        % Mapeo QPSK
reformado_tx()       % Filtro SRRC de transmisión
ofdm_tx()            % Creación del los símbolos OFDM
preambulo_tx()       % Agregar el preambulo 802.11a
close all % Cerrar las gráficas que se abren en los scripts anteriores
datosSinc=datosConPre(1:3000); % Recortar datos, queda el preámbulo
% seguido de algunos símbolos OFDM
clearvars -except datosSinc % Borrar todas las variables excepto
% datosSinc
```

Luego de ejecutar la sección 1 se obtiene en el espacio de trabajo el vector *datosSinc*, que servirán como datos de entrada a la sección 2 y sección 3 del script *EjemploFase.m*. En la Figura 2.146 se muestra el vector *datosSinc* en el espacio de trabajo de Matlab.

Figura 2.146. Variables en el espacio de trabajo luego de ejecutar la sección 1 de *EjemploFase.m*.

En la sección 2 se inicia por definir el valor de desfase y la frecuencia de muestreo. Luego mediante el objeto del sistema *comm.PhaseFrequencyOffset*, se crea un objeto y se aplica el desplazamiento de fase al vector *datosSinc*. Después se pasa las dos secuencias largas del LLTF del preámbulo al dominio de la frecuencia. En un vector llamado *angulosAux* se almacenan los valores correspondientes a las posiciones 2, 5, 6, 8, 10, 16, 17,20, 53, 56, 57, 59, 61, 62,63 y 64, de cada secuencia larga, ya explicado anteriormente.

Después se calcula el valor de cada ángulo del vector *angulosAux*, mediante la función de Matlab *angle* y se pasa de radianes a grados. Finalmente, se calcula el promedio de cada ángulo obtenido y se multiplica por -1, para que sea el opuesto al introducido inicialmente. El ángulo calculado se almacena en la variable ***anguloProm***. A continuación, se presenta el código de la sección 2 del script *EjemploFase.m*.

```
%% Sección 2: Ejemplo de estimación de fase sin ruido
desfase=156; % Valor de desfase en grados
Fs=2.0e6; % Frecuencia de muestreo (configurado en los SDR)
% Asignar a la señal de entrada un desplazamiento de fase:
DesfSenial = comm.PhaseFrequencyOffset('PhaseOffset',desfase,...
    'SampleRate',Fs);
datosDesfase=DesfSenial(datosSinc); % Aplicar el desfase a datosSinc
% Pasar las dos secuencias largas del LLTF del dominio del tiempo al
% dominio de la frecuencia:
a=fft(datosDesfase(257:320)); % segunda sección larga de LLTF
b=fft(datosDesfase(193:256)); % Primera sección larga de LLTF
%  Valores de las secuencias largas a calcular el ángulo:
angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
    a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
    b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
    b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
% radianes a grados
anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
```

En la Figura 2.147 se muestra las variables en el espacio de trabajo luego de ejecutar la sección 2. La variable señalada con verde llamada *desfase* contiene el valor del desplazamiento de fase introducido con el objeto *comm.PhaseFrequencyOffset*. Mientras que la variable señalada con rojo llamada *anguloProm* es el negativo del desplazamiento estimado. Al comparar el valor de *anguloProm* con *desfase* se puede decir estimación de fase es correcta.

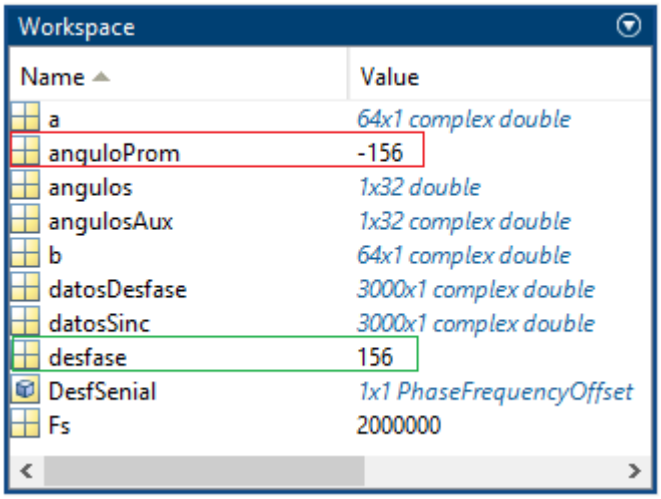


Figura 2.147. Variables en el espacio de trabajo luego de ejecutar la sección 2 de *EjemploFase.m*.

En la sección 3 es exactamente igual a la sección 2, la única diferencia es que se cambia el valor de desplazamiento en fase y que se añade ruido blanco gaussiano antes de aplicar el desplazamiento. El ruido blanco se añade con la función *awgn* de Matlab. El código de la sección 3 se presenta a continuación debidamente comentado.

```
%% Sección 3: Ejemplo de estimación de fase con ruido
desfase=111; % Valor de desfase en grados
```

```
Fs=2.0e6; % Frecuencia de muestreo (configurado en los SDR)
% Asignar a la señal de entrada un desplazamiento de fase:
DesfSenial = comm.PhaseFrequencyOffset('PhaseOffset',desfase,...
    'SampleRate',Fs);
datosSinc1 = awgn(datosSinc,20,'measured'); % Señal con ruido, entrada
datosDesfase=DesfSenial(datosSinc1); % Aplicar el desfase a datosSinc
% Pasar las dos secuencias largas del LLTF del dominio del tiempo al
% dominio de la frecuencia:
a=fft(datosDesfase(257:320)); % segunda sección larga de LLTF
b=fft(datosDesfase(193:256)); % Primera sección larga de LLTF
%  Valores de las secuencias largas a calcular el ángulo:
angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
    a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
    b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
    b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
% radianes a grados
anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
```

En la Figura 2.148 y la Figura 2.149 se muestran las variables en el espacio de trabajo luego de ejecutar la sección 3 de *ejemploFase.m*. Debido a que el ruido introducido es aleatorio, se obtienen valores distintos de estimación del desplazamiento de fase en cada ejecución.

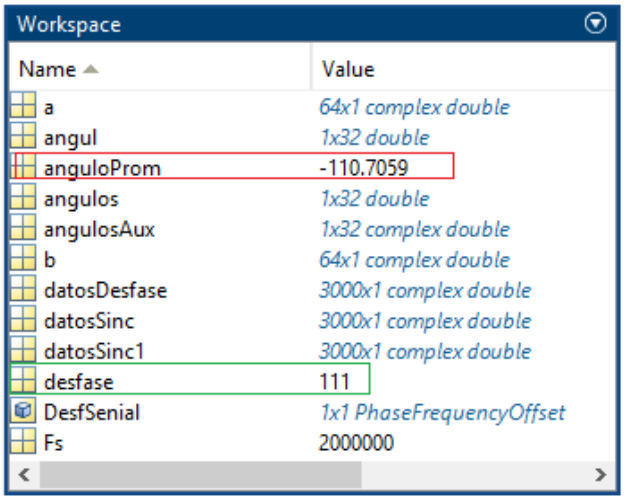


Figura 2.148. Variables en el espacio de trabajo luego de ejecutar la sección 3 de *EjemploFase.m*.

Las variables señaladas con verde llamada *desfase* contiene el valor del desplazamiento de fase introducido con *comm.PhaseFrequencyOffset*. Las variables señaladas con rojo llamadas *anguloProm* es el negativo del desplazamiento estimado. Al comparar el valor de *anguloProm* con el valor que contiene la variable *desfase*, en los dos casos, la estimación de fase es muy similar al desplazamiento introducido.

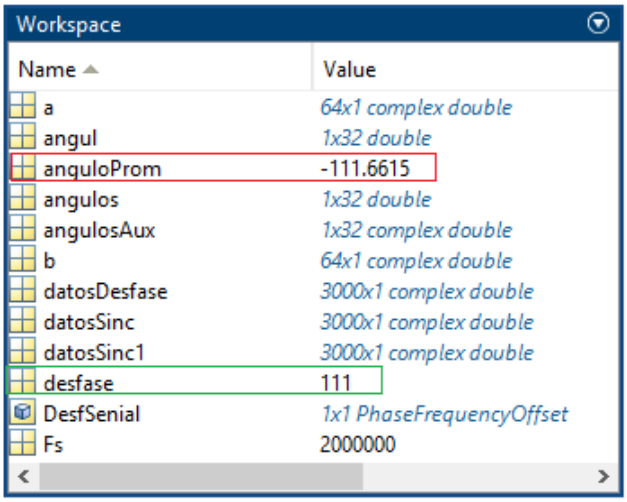


Figura 2.149. Variables en el espacio de trabajo luego de ejecutar la sección 3 de *EjemploFase.m*.

Tanto en la simulación con ruido como en la simulación sin ruido se pudo obtener una estimación del desplazamiento de fase bastante aceptable, por lo que el código funciona adecuadamente y se puede aplicar a los datos recibidos del canal inalámbrico.

2.5.2.4.2. Corrección de fase en el archivo de texto recibido

Previo a implementar la etapa Corrección de fase a los datos recibidos del canal inalámbrico, se debe cargar los datos recibidos, luego ejecutar el script *recortar_rx.m*, *encontrarTrama_rx.m*, *corrCFO_rx.m* y *SincSimbolo_rx.m*. El vector de entrada de la etapa actual es *datosSincronizados* y se obtiene del script *SincSimbolo_rx.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando.

```
clearvars -except datosSincronizados % Borrar todas las variables
% excepto la variable datosSincronizados (entrada de corrFase_rx)
```

Para implementar la corrección de fase a la trama OFDM contenida en *datosSincronizados*, se crea un script de nombre *corrFase_rx.m*. El código de dicho script inicia por realizar la estimación del desplazamiento de fase introducido por el canal inalámbrico. Este proceso ya se describió en el ejemplo de estimación del desplazamiento de fase. El negativo del desplazamiento de fase estimado se almacena en la variable *anguloProm*.

Luego se utiliza el objeto del sistema *comm.PhaseFrequencyOffset* junto con *anguloProm* y con la frecuencia de muestreo con la que se configuró los SDR (2Msps), para crear el objeto *desFaseObj*. Con dicho objeto se corrige el desplazamiento de fase del vector *datosSincronizados* y se almacenan en el vector *datFase*. En esta etapa también se

introduce un relleno, para que el vector de datos sea múltiplo de 80, que es el tamaño del símbolo OFDM. Los datos rellenados se almacenan en un vector llamado *datosCorrFase*, este vector sirve como entrada a la etapa de eliminación del prefijo cíclico. En dicha etapa se recorre el vector *datosCorrFase* en múltiplos de 80, por lo que el relleno es necesario.

A continuación, se presenta el código del script *corrFase_rx.m* debidamente comentado.

```
% Script que permite realizar la corrección de fase
% Antes de correr este script es necesario cargar los datos
% guardados en datosRecibidos150.mat, ejecutar recortar_rx.m,
% encontrarTrama_rx.m, corrCFO_rx.m y SincSimbolo_rx.m
% En vector de entrada de esta etapa es datosSincronizados
% CORRECCIÓN FASE:
% Pasar las dos secuencias largas del LLTF del dominio del tiempo al
% dominio de la frecuencia:
a=fft(datosSincronizados(257:320)); % segunda sección larga de LLTF
b=fft(datosSincronizados(193:256)); % Primera sección larga de LLTF
%  Valores de las secuencias largas a calcular el ángulo:
angulosAux=[a(2) a(5) a(6) a(8) a(10) a(16) a(17) a(20) ...
    a(53) a(56) a(57) a(59) a(61) a(62) a(63) a(64) ...
    b(2) b(5) b(6) b(8) b(10) b(16) b(17) b(20) ...
    b(53) b(56) b(57) b(59) b(61) b(62) b(63) b(64)];
angulos=(angle(angulosAux)*180)/pi; % Calcular el ángulo y pasar de
% radianes a grados
anguloProm=-mean(angulos); % Negativo del promedio de los ángulos
% Corregir el desfase
Fs=2.0e6; % Frecuencia de muestreo (configurado en los SDR)
desFaseObj = comm.PhaseFrequencyOffset('PhaseOffset',anguloProm,...
    'SampleRate',Fs);
datFase=desFaseObj(datosSincronizados);
% Se verifica que los datos corregidos sean multiplo de 80,
% caso contrario se realiza un padding de las ultimas muestras
% Esto ya que la siguiente etapa se recorre el vector de entrada en
% multiplos de 80
longDatF=length(datFase);
padd80=ceil(longDatF/80)*80-longDatF;
if(padd80>=1)
    datosCorrFase=[datFase;datFase(end-padd80+1:end)];
else
    datosCorrFase=datFase;
end
% El vector de salida es datosCorrFase
```

Luego de ejecutar esta sección de código se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.150 Se encierra en un recuadro verde el negativo del desplazamiento de fase estimado. El vector *datosCorrFase* es el vector de salida de la etapa Corrección de Fase.

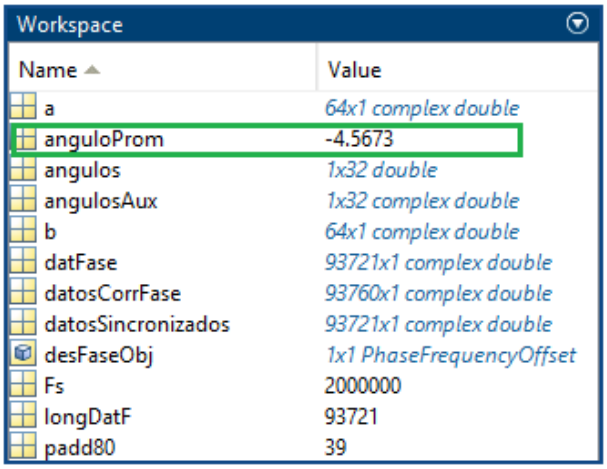


Figura 2.150. Variables en el espacio de trabajo luego de ejecutar el script *corrFase_rx.m*.

2.5.3. PROCESAMIENTO OFDM

En el capítulo de transmisión se abordó los conceptos teóricos relacionados con las características de OFDM. En este capítulo se aborda el procesamiento de los símbolos OFDM para recuperar los datos que contiene cada símbolo. En la Figura 2.151 se señalan con color naranja las etapas que se abordan en este capítulo. A grandes rasgos, la etapa Eliminación del PC elimina las primeras 16 muestras de cada símbolo OFDM, es decir que se elimina el prefijo cíclico. La etapa FFT demodula los símbolos OFDM recibidos, es decir se pasan del dominio del tiempo al dominio de la frecuencia. La etapa Estimación de canal y ecualizador utiliza las portadoras piloto para ecualizar cada símbolo. Por último, la etapa Recuperación de datos del símbolo OFDM extrae y serializa los datos de cada símbolo OFDM.

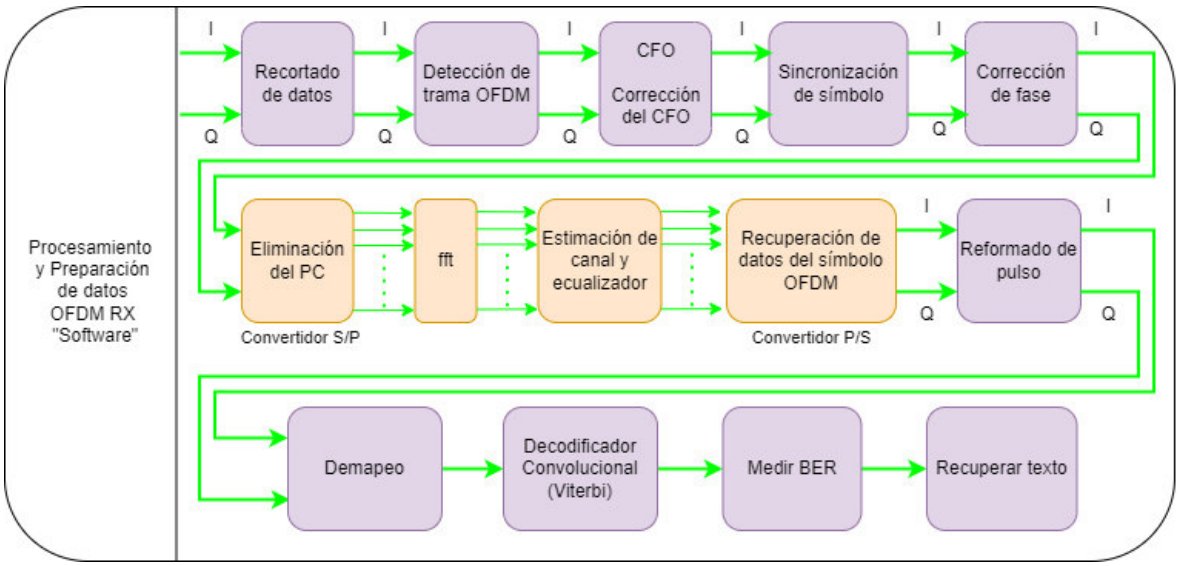


Figura 2.151. Etapas de recepción con todas las etapas que intervienen en el procesamiento del esquema OFDM.

La **etapa Eliminación del PC** se implementa en Matlab mediante un lazo *for*. Este lazo recorre cada símbolo recuperado y elimina las primeras 16 muestras, que corresponde al prefijo cíclico. Sin el prefijo cíclico cada símbolo OFDM tiene 64 muestras. Luego se debe implementar la **etapa FFT** para **demodular** los símbolos OFDM. Mediante la función *fft* de Matlab se aplica la DFT a cada símbolo OFDM, con esto se pasan los símbolos del dominio del tiempo al dominio de la frecuencia. La sintaxis de la función *fft* se muestra a continuación.

datFreq = fft(datT)

La función *fft* calcula la transformada discreta de Fourier (DFT) de *datT*, utilizando un algoritmo de transformada rápida de Fourier (FFT) [39]. El parámetro de entrada *datT* puede ser un vector columna o una matriz, en cuyo caso devuelve la DFT de cada columna.

La **etapa Estimación de canal y ecualizador** utiliza las portadoras piloto para estimar los efectos del canal inalámbrico sobre cada símbolo OFDM recibido. En el dominio de la frecuencia, las portadoras piloto deben tener un valor de 1. Pero al pasar por el canal inalámbrico estas portadoras son afectadas y por tanto tendrán un valor diferente de 1.

Para estimar el canal hay que hacer que cada portadora piloto sea igual a 1, esto se logra dividiendo el valor recibido para sí mismo. Luego se realiza una interpolación lineal para obtener el valor por el cual dividir las portadoras de datos.

En la Figura 2.152 se muestra una representación de la estructura de un símbolo OFDM. Se identifica con rojo el valor que debe tener la portadora piloto, es decir uno. Por otro lado, se identifica con morado el valor de las portadoras piloto luego de pasar por el canal inalámbrico. También se encuentra con color morado la interpolación lineal que se debe realizar, para conseguir el valor del divisor de las portadoras de datos. Por esto se señala en los extremos de las líneas moradas los límites de las portadoras de datos.

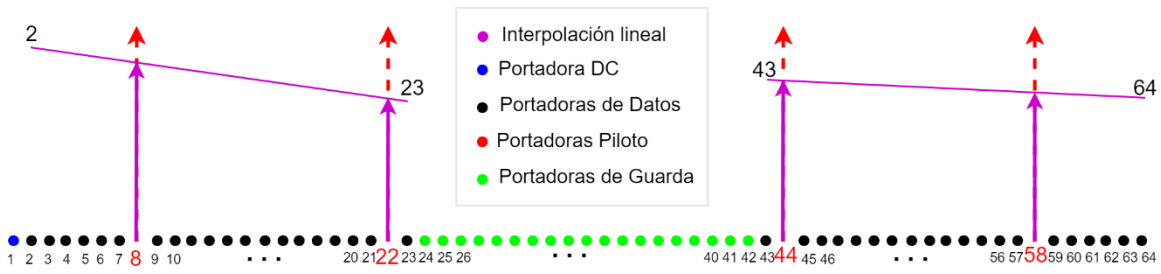


Figura 2.152. Estructura de un símbolo OFDM e interpolación lineal.

La interpolación lineal se debe realizar en cada lado del símbolo OFDM. En el lado izquierdo se utiliza las portadoras piloto en las posiciones 8 y 22, para realizar la interpolación y extrapolación lineal de la posición 2 a la 23. En el lado derecho se utiliza las

portadoras piloto en las posiciones 44 y 58, para realizar la interpolación y extrapolación lineal de la posición 43 a la 64.

Para realizar la interpolación y extrapolación lineal se utiliza la función de Matlab ***interp1***, cuya sintaxis a utilizar se presenta a continuación.

val_in = interp1(x,v,xq,method,extrapolation)

La función *interp1* devuelve valores interpolados de una función de una dimensión en puntos de consulta específicos mediante interpolación lineal [40]. El parámetro de salida *val_in* almacena un vector con los datos interpolados. Los parámetros de entrada se listan y se describen a continuación.

- ***v*** es un vector que contiene los valores de los puntos a interpolar. En nuestro caso es los valores de las portadoras piloto recibidas.
- ***x*** contiene las posiciones de los puntos del vector ***v***, tal que se tiene ***v(x)***. En nuestro caso es la posición de las portadoras piloto, ya sea del lado izquierdo o del lado derecho.
- ***xq*** contiene las coordenadas de los puntos en donde se va a realizar la interpolación. En nuestro caso debe ser de 2 a 23 o bien de 43 a 64.
- ***method*** especifica el método por el cual se van a interpolar los datos, el valor por defecto es 'linear', el cual especifica una interpolación lineal, es decir con una línea recta entre los valores de ***v(x)***. Otros métodos que se pueden especificar son; 'nearest', 'next', 'previous', 'pchip', 'cubic', 'v5cubic', 'makima', or 'spline' [40].
- ***extrapolation*** es la estrategia de evaluación de los puntos fuera de ***x***. Este parámetro debe establecerse en 'extrap' para aplicar la extrapolación a los datos en posiciones fuera de lo especificado por ***x***.

La etapa **Recuperación de datos del símbolo OFDM** se encarga de extraer las portadoras de datos de cada símbolo OFDM. Se realiza el proceso contrario que en recepción, para que las portadoras de datos queden ordenadas. Luego se serializan todos los datos recuperados. En la Tabla 2.15 se muestra las posiciones de los datos dentro del símbolo OFDM, para poder recuperar la información ordenada.

Tabla 2.15. Posición de los 40 datos de información dentro del símbolo OFDM.

Posición datos	Posición en el símbolo	Posición datos	Posición en el símbolo	Posición datos	Posición en el símbolo	Posición datos	Posición en el símbolo
1	43	11	54	21	2	31	13
2	45	12	55	22	3	32	14
3	46	13	56	23	4	33	15
4	47	14	57	24	5	34	16
5	48	15	59	25	6	35	17
6	49	16	60	26	7	36	18
7	50	17	61	27	9	37	19
8	51	18	62	28	10	38	20
9	52	19	63	29	11	39	21
10	53	20	64	30	12	40	23

2.5.3.1. Procesamiento OFDM en el archivo de texto recibido

Previo a implementar el procesamiento OFDM a los datos recibidos del canal inalámbrico, se debe cargar los datos recuperados del canal. Los datos con los que se está trabajando en los ejemplos que se han presentado, se encuentran en el archivo *datosRecibidos150.mat*. Luego se debe ejecutar el script *recortar_rx.m*, correspondiente a la etapa Recortado de datos. Después se ejecutan los scripts correspondientes al procesamiento del preámbulo IEE 802.11a que son; *recortar_rx.m*, *encontrarTrama_rx.m*, *corrCFO_rx.m, SincSimbolo_rx.m* y *corrFase_rx.m*.

El vector de entrada de la etapa actual es *datosCorrFase* y se obtiene del script *corrFase_rx.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando

```
clearvars -except datosCorrFase % Borrar todas las variables
% excepto la variable datosCorrFase (entrada ofdm_rx)
```

Todas las etapas del procesamiento OFDM del archivo de texto recibido se implementa en un script llamado *ofdm_rx.m*. La **etapa Eliminación del PC** empieza por recortar los datos a procesar, eliminando el preámbulo OFDM 802.11a. También se inicializa la variable *nSimOFDM* en cero, esta variable es un contador de los símbolos OFDM. Luego se recorre cada símbolo que tiene una longitud de 80 y se elimina las primeras 16 muestras correspondientes al prefijo cíclico. La matriz que almacena los símbolos de 64 muestras, en cada columna, es *simbolosSinPC*.

Luego se implementa la **etapa FFT** aplicando la función *fft* a la matriz *simbolosSinPC* y el resultado se almacena en una matriz de nombre *simbolosfft*.

La **etapa Estimación de canal y ecualizador** empieza por inicializar el vector *simbolosEcualizados*, el cual almacena los símbolos ecualizados. Con un lazo *for* se recorre los símbolos OFDM mediante *nSimOFDM*. En cada iteración del lazo *for* se realiza la interpolación y extrapolación lineal utilizando las portadoras piloto y la función *interp1*. La interpolación del lado izquierdo, de la posición 2 a la 23 utilizando las pilotos 8 y 22, se almacena en la variable *val_interp1*. Mientras que la interpolación del lado derecho, de la posición 43 a la 64 utilizando las pilotos 44 y 58, se almacena en la variable *val_interp2*.

Luego se inicializa con ceros un vector columna llamado *DatosEc* que almacena los datos ecualizados. Paso seguido, se ecualiza el símbolo recibido dividiendo los valores de las posiciones 2 a la 23 y de 43 a 64, para *val_interp1* y *val_interp2* respectivamente. El resultado se almacena en *DatosEc* en las mismas posiciones evaluadas del símbolo recibido. Luego se guarda el símbolo ecualizado en *simbolosEcualizados* y el bucle *for* continúa.

La **etapa Recuperación de datos del símbolo OFDM** inicia por definir el número de portadora de datos que es 40. Se inicia con ceros la variable que contendrá a los datos serializados, la cual se llama *datosReformado*. Con un lazo *for* se recorre los símbolos OFDM hasta en valor que contiene *nSimOFDM*. Luego se extrae las portadoras de datos de cada símbolo, como se describió en la Tabla 2.15. Finalmente, en cada iteración del lazo *for* se serializa los datos y se almacena en el vector *datosReformado*.

El código del script *ofdm_rx.m* que contiene las cuatro etapas del procesamiento OFDM, se presenta a continuación debidamente comentado.

```
% Script que permite eliminar el PC,  demodular los símbolos OFDM
% ecualizar los símbolos y recuperar los datos de cada símbolo
% Antes de correr este script es necesario cargar los datos
% guardados en datosRecibidos150.mat, ejecutar recortar_rx.m,
% encontrarTrama_rx.m, corrCFO_rx.m, SincSimbolo_rx.m y corrFase_rx.m
% En vector de entrada de esta etapa es datosCorrFase
%% %%%%%%%%%%%%%%%%%%%%%%%%  ELIMINAR PC  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nSimOFDM=0; % Iniciar contador de símbolos OFDM
tramaDatos=datosCorrFase(321:end); % Quitar el preambulo
% Recorrer cada símbolo y eliminar el PC de cada uno
for i=1:80:length(tramaDatos)
    nSimOFDM=nSimOFDM+1; % Sumar 1 en la cantidad de símbolos OFDM
    simbolosSinPC(:,nSimOFDM)=tramaDatos(i+16:i+79); % Guardar el
    % símbolo sin PC
end
%% %%%%%%%%%%%%%%%%%%%%%%% DEMODULACIÓN OFDM (FFT) %%%%%%%%%%%%%%%%%%%
% La función fft aplica la DFT a cada columna de la matriz
% simbolosSinPC
simbolosfft=fft(simbolosSinPC); % Demodular los símbolos sin PC
%% %%%%%%%%%%%%    ESTIMACIÓN DE CANAL Y ECUALIZACIÓN    %%%%%%%%%%%%%
simbolosEcualizados=[]; % Iniciar vector que almacena los simb ecualiz
% Con un lazo for se recorre todos los símbolos OFDM en frecuencia
```

```
for auxnSimb=1:nSimOFDM
    simbF=simbolosfft(:,auxnSimb); % Símbolo en frecuencia
    % Se realiza la interpolacion lineal utilizando las portadoras
    % piloto, que estan en las posiciones 8 22 44 y 58:
    x1=[8 22]; % Puntos de la muestra
    v1=[simbF(8) simbF(22)]; % Valores de pilotos recibidos
    xq1=(2:23); % Puntos a estimar
    x2=[44 58]; % Puntos de la muestra
    v2=[simbF(44) simbF(58)]; % Valores de pilotos recibidos (2)
    xq2=(43:64); % Puntos a estimar
    val_interp1 = (interp1(x1,v1,xq1,'linear','extrap')).';
    val_interp2 = (interp1(x2,v2,xq2,'linear','extrap')).';
    % Se aplica la interpolación a las portadoras de datos:
    DatosEc=zeros(64,1); % Iniciar variable q almacena datos ecualiz
    DatosEc(2:23)=simbF(2:23)./val_interp1; % Ecualizar datos
    DatosEc(43:64)=simbF(43:64)./val_interp2; % Ecualizar datos
    simbolosEcualizados(:,auxnSimb)=DatosEc; % Datos de salida
end
nSimOFDM=auxnSimb; % Copiar el número de símbolos OFDM
%% %%%%%%%%%%%%%%%%%    RECUPERACIÓN DE DATOS    %%%%%%%%%%%%%%%%%%
nPdatos=40; % Número de portadoras de datos
datosReformado=zeros(nPdatos*nSimOFDM,1); % Iniciar variable que
% contiene los datos seriaizados
% Con un lazo for se recorre todos los símbolos OFDM:
for auxnSimb=1:nSimOFDM
    % obtener los datos OFDM
    simAux=simbolosEcualizados(:,auxnSimb); % Símbolo ecualizado
    % Desamblaje del símbolo OFDM:
    datosAux=zeros(40,1); % Inicializar en 0 un vector auxiliar
    %               simAux(1) es Portadora DC
    datosAux(21:26)=simAux(2:7);
    %               simAux(8) es la primera piloto
    datosAux(27:39)=simAux(9:21);
    %               simAux(22) es la primera piloto
    datosAux(40)=simAux(23);
    %            simAux(24:42) son las portadoras de guarda
    datosAux(1)=simAux(43);
    %           simAux(44) es la primera piloto
    datosAux(2:14)=simAux(45:57);
    %              simAux(58) es la primera piloto
    datosAux(15:20)=simAux(59:64);
    % Serializar los datos recuperados:
    datosReformado((auxnSimb-1)*nPdatos+1:auxnSimb*nPdatos)=datosAux;
end
% Los datos de salida son datosReformado
```

Luego de ejecutar el código anterior se tienen las variables en el espacio de trabajo que se muestran en la Figura 2.153. Se encierra en un recuadro rojo el vector de entrada y en verde el vector de salida *datosReformado*. Este último vector sirve como datos de entrada de la etapa Reformado de pulso.

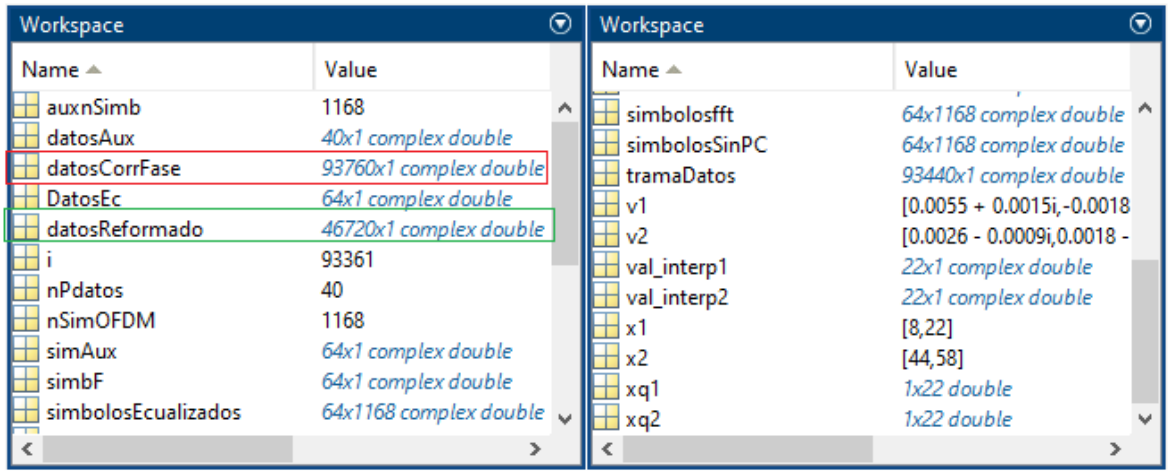


Figura 2.153. Variables en el espacio de trabajo luego de ejecutar el script *ofdm_rx.m*.

2.5.4. ETAPA DE REFORMADO DE PULSO

Los fundamentos teóricos de la técnica de reformado de pulso se expusieron en el capítulo de transmisión. Además, se describió la implementación en Matlab mediante funciones directas. La implementación práctica de la etapa Reformado de pulso se realiza mediante dos filtros raíz cuadrada de coseno elevado (SRRC), uno en transmisión y otro en recepción. En la Figura 2.154 se identifica la etapa actual con color naranja, dentro de la etapa de recepción de procesamiento de datos. En la interfaz gráfica de recepción esta etapa se puede activar o desactivar, igual que en la interfaz gráfica de transmisión.

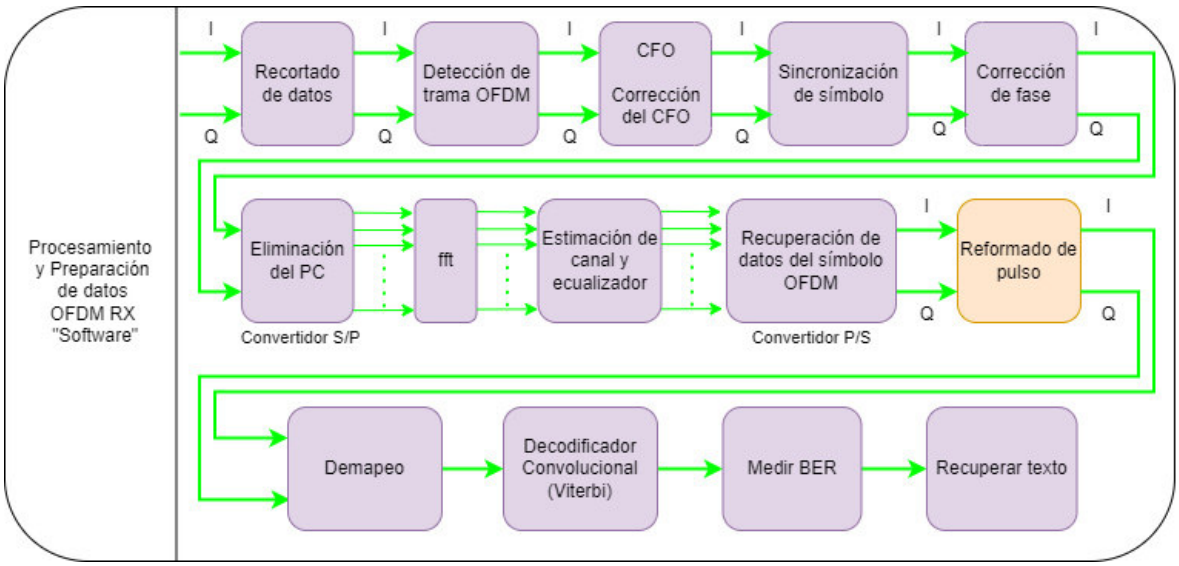


Figura 2.154. Etapas de recepción con la etapa Reformado de pulso señalada con naranja.

El filtrado SRRC de recepción se implementa mediante un objeto del sistema de Matlab llamado *comm.RaisedCosineReceiveFilter*. Este objeto tiene la siguiente sintaxis:

rxfilter = comm.RaisedCosineReceiveFilter(Name,Value)

El objeto del sistema *comm.RaisedCosineReceiveFilter* diezma la señal de entrada, utilizando un filtro FIR que puede tener la forma RC o SRRC [32]. En la sintaxis anterior, *Name* representa el nombre de la propiedad que se define con *Value*. La variable *rxfilter* almacena el filtro de recepción creado en base a lo que se define con las propiedades de entrada.

Las propiedades de entrada del filtro se describen a continuación:

- **Shape:** Es la forma del filtro, especificada como 'Square root' para implementar un filtro SRRC de recepción.
- **RolloffFactor:** Es el factor roll-off o factor de caída del filtro, que puede tomar valores entre 0 y 1, el valor por defecto es 0.2.
- **FilterSpanInSymbols:** Es el tamaño de la ventana del filtro intervalo, especificado en símbolos, es decir el truncado del tamaño de la ventana. Esta propiedad debe tomar un valor entero positivo y el valor por defecto es 10.
- **InputSamplesPerSymbol:** Es el número de muestras de entrada por cada símbolo. Se especifica como un número entero positivo y el valor por defecto es 8.
- **DecimationFactor:** Es el factor de diezmado, que se especifica con un número escalar positivo, que puede tomar valores entre 1 e *InputSamplesPerSymbol*. El tamaño del vector de salida es igual al tamaño del vector de entrada para *DecimationFactor*. El valor por defecto es de 8.
- **DecimationOffset:** Desplazamiento de diezmado, se especifica como un número entero en el rango de 0 hasta (*DecimationFactor*- 1) y el valor por defecto es 0. Esta propiedad especifica el número de muestras filtradas que el objeto descarta antes de reducir la resolución.
- **Gain:** Es la ganancia del filtro, se especifica como un número escalar positivo y el valor por defecto es 1.

Una vez creado el filtro *rxfilter*, se puede aplicar el mismo, tal que se diezme la señal de entrada utilizando la siguiente sintaxis:

datosSal = rxfilter (datoFiltrados)

El argumento *datosFiltrados* corresponde a un vector columna que contiene los datos de entrada que previamente se han filtrado con el objeto del sistema *comm.RaisedCosineTransmitFilter*. El argumento de salida *datosSal* devuelve un vector columna que contiene la señal diezmada mediante la propiedad *DecimationFactor*.

Luego de filtrar los datos recibidos es necesario eliminar los primeros *FilterSpanInSymbols*, ya que al pasar por los filtros de transmisión y recepción se introducen esa cantidad de datos de relleno al inicio de los datos recuperados.

2.5.4.1. Reformado de pulso en el archivo de texto recibido

Previo a aplicar el filtro SRRC de recepción a los datos recibidos del canal inalámbrico, para implementar la técnica de reformado de pulso, se debe cargar los datos recuperados del canal. Los datos con los que se han estado trabajando en los ejemplos previos, se encuentran en el archivo *datosRecibidos150.mat*. Luego se debe ejecutar el script *recortar_rx.m*, correspondiente a la etapa Recortado de datos. Después se ejecutan los scripts correspondientes al procesamiento del preámbulo IEE 802.11a que son; *recortar_rx.m*, *encontrarTrama_rx.m*, *corrCFO_rx.m, SincSimbolo_rx.m* y *corrFase_rx.m*. A continuación, se ejecuta el script *ofdm_rx.m* que corresponde al procesamiento OFDM

El vector de entrada de la etapa actual es *datosReformado* y se obtiene del script *ofdm_rx.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando

```
clearvars -except datosReformado % Borrar todas las variables
% excepto la variable datosReformado (entrada oreformado_rx.m)
```

La etapa de Reformado de pulso del archivo de texto recibido se implementa en un script llamado *reformado_rx.m*. El código inicia por establecer los parámetros del filtro de recepción, estos deben ser los mismos que se utilizaron en el filtro de transmisión.

Se utiliza la variable *rolloff* en donde se guarda el valor del factor de caída del filtro SRRC, y se asigna un valor de 0.5. El tamaño de la ventana se almacena en la variable *span* y se guarda el valor de 6. El número de muestras de entrada por cada símbolo se almacena en la variable *sps* y toma el valor de 4, el valor de *sps* también se utiliza en el factor de diezmado del filtro para recuperar los símbolos mapeados.

Luego se crea el filtro SRRC de recepción, con el objeto del sistema llamado *comm.RaisedCosineReceiveFilter*. En el parámetro *Shape* se asigna el valor 'Square root'. En el parámetro *RolloffFactor* se asigna el valor guardado en la variable *rolloff*. En el parámetro *FilterSpanInSymbols* se asigna el valor dentro de *span*. En la propiedad *InputSamplesPerSymbol* se asigna *sps*, al igual que en la propiedad *DecimationFactor*. El filtro creado se almacena en la variable *rxfilter*.

Después se realiza un relleno de ceros en el vector *datosReformado* para que los datos a filtrar sean múltiplo de *sps*, los datos con relleno se almacenan en la variable *datosRellAux*. Luego se filtran los datos con el filtro *rxfilter* y el resultado se almacena en *datosMapAux*. A continuación, se eliminan los primeros *span* datos filtrados, ya que luego de pasar por el filtro de transmisión y recepción se introducen *span* datos de relleno al inicio de los datos recuperados. Los datos recortados se almacenan en la variable *datosMapeo*.

Finalmente, se grafica el diagrama de constelación de los datos antes del filtrado SRRC de recepción y el diagrama de constelación luego del filtrado. El código del script *reformado_rx.m* se presenta a continuación debidamente comentado.

```
% Script que permite aplicar el filtro SRRC de recepción y
% recuperar los datos mapeados
% Antes de correr este script es necesario cargar los datos guardados
%en datosRecibidos150.mat, ejecutar recortar_rx.m, encontrarTrama_rx.m,
% corrCFO_rx.m, SincSimbolo_rx.m, corrFase_rx.m y ofdm_rx,m
% En vector de entrada de esta etapa es datosReformado
% Parámetros del filtro de recepción:
rolloff = 0.5;  % Factor de caída del filtro
span = 6;       % Define el tamaño de la ventana en símbolos
sps = 4;        % # de muestras por cada símbolo
% En el filtro de recepción se define el parámetro 'Shape' como
% 'Square root' para obtener un filtro SRRC
% Creación del filtro SRRC:
rxfilter = comm.RaisedCosineReceiveFilter('Shape','Square root',...
    'RolloffFactor',rolloff,'FilterSpanInSymbols',span, ...
    'InputSamplesPerSymbol',sps,'DecimationFactor',sps);
% Agregar relleno para asegurar que los datos a filtrar sean multipo
% de sps:
longDatR=length(datosReformado); % Longitud de datos
mltSPS=ceil(longDatR/sps); % Calcular un multiplo de sps
nceros=mltSPS*sps-longDatR; % Calcular el relleno de ceros
datosRellAux=[datosReformado; zeros(nceros,1)]; % Rellenar el vector
% A los datos de entrada datosRellAux se aplica el filtro rxfilter
% y se almacena en la variable datosMapAux
datosMapAux=rxfilter(datosRellAux);
datosMapeo=datosMapAux(span+1:end,1);  % Datos de salida
% Graficar el diagrama de constelación de los datos de entrada y
% de salida:
figure()
plot(datosReformado,'r.'); hold on
plot(datosMapeo,'b.');
title('Señal QPSK recuperada')
legend('Señal recibida','Datos QPSK')
xlabel('I (fase)'); ylabel('Q (cuadratura)'); hold off
```

Luego de ejecutar el script *reformado_rx.m* se obtiene las variables del espacio de trabajo que se muestra en la Figura 2.155 y también la gráfica de la Figura 1.156. En la Figura 2.155 se encierra en un recuadro verde el vector de salida, que sirve como datos de entrada

para la etapa Demapeo. También se observa que no fue necesario realizar un relleno de ceros, ya que el vector de entrada es múltiplo de *sps*.

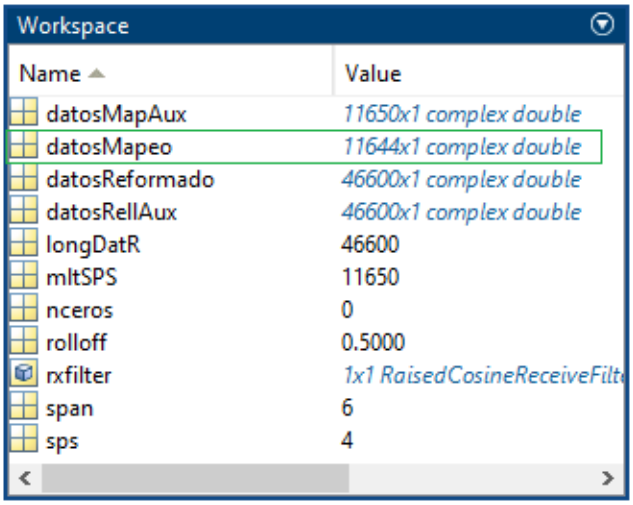


Figura 2.155. Variables en el espacio de trabajo luego de ejecutar el script *reformado_rx.m*.

En la Figura 2.156 se muestra con rojo los puntos de la constelación recibida antes del filtrado SRRC de recepción, se puede ver que los puntos se encuentran dispersos. La dispersión de datos es normal, hasta cierto punto, debido a que el filtro de transmisión introduce esta dispersión para hacer la señal más suave y el canal inalámbrico también hace que los puntos se dispersen. Mientras que con azul se identifican los datos luego del filtrado, es decir los símbolos mapeados recuperados. En transmisión se mapeo con QPSK y se observa que los puntos azules se asemejan a dicha técnica de mapeo, solo que los datos están dispersos debido al canal inalámbrico.

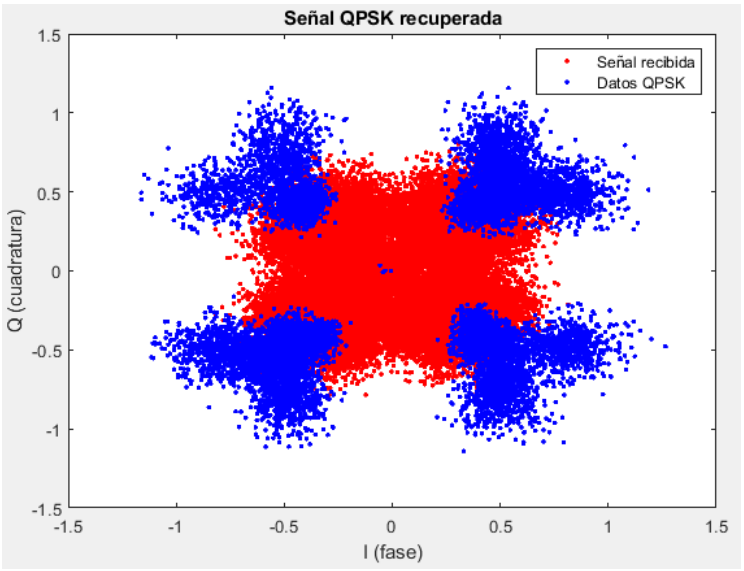


Figura 2.156. Diagrama de constelación antes y después del filtrado SRRC de recepción.

2.5.5. ETAPA DE DEMAPEO

En el prototipo de comunicación inalámbrica se utiliza cuatro esquemas de modulación digital o mapeo, estos son; BPSK, QPSK, 16-QAM y 64-QAM. En el capítulo de transmisión se abordaron los fundamentos teóricos y se explicó la razón por la cual se implementa el mapeo. En la sección actual se pretende explicar e implementar la demodulación digital o demapeo de los esquemas BPSK, QPSK, 16-QAM y 64-QAM. En la Figura 2.157 se muestran las etapas de recepción y se identifica con color naranja la etapa Demapeo.

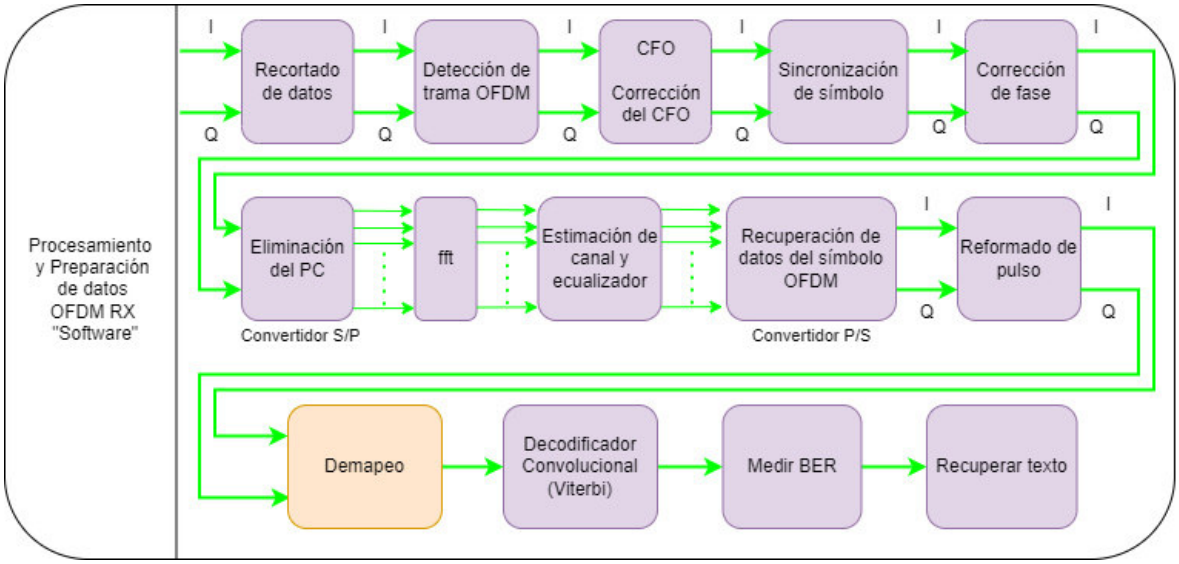


Figura 2.157. Etapas de recepción con la etapa Demapeo señalada con naranja.

El demapeo consiste en recuperar los bits de información (o bien los bits codificados) a partir de los símbolos mapeados con BPSK, QPSK, 16-QAM o 64-QAM.

En transmisión dependiendo la técnica de mapeo utilizada se multiplica por un factor de normalización. Este factor se utiliza para lograr que los símbolos de la constelación tengan una potencia media igual a 1 [16]. En recepción, antes de demodular los símbolos mapeados, es necesario quitar la normalización multiplicando los símbolos por el inverso de los factores de normalización. En la Tabla 2.16 se muestran estos factores de multiplicación, para cada técnica de mapeo utilizada.

Tabla 2.16. Factores de multiplicación de varías técnicas de mapeo.

Mapeo	Factor de multiplicación
BPSK	
QPSK	$\sqrt{2}$
16-QAM	$\sqrt{10}$
64-QAM	$\sqrt{42}$

2.5.5.1. Implementación del demapeo en Matlab

En esta subsección se implementa ejemplos de las técnicas el demapeo BPSK, QPSK, 16-QAM y 64-QAM en Matlab. Estos ejemplos se implementan en un script llamado *EjemploDemapeo.m* y el mismo se divide en secciones para poder ejecutar cada una por separado y replicar los resultados que se muestren.

2.5.5.1.1. Demapeo BPSK en Matlab

El mapeo BPSK tiene solamente dos símbolos en su diagrama de constelación y utiliza un bit de entrada por cada símbolo. En la Figura 2.158 se muestra con azul los símbolos BPSK antes de ser transmitidos y con rojo los mismos puntos luego de pasar por el canal inalámbrico. Al pasar los símbolos mapeados por el canal inalámbrico estos se dispersan.

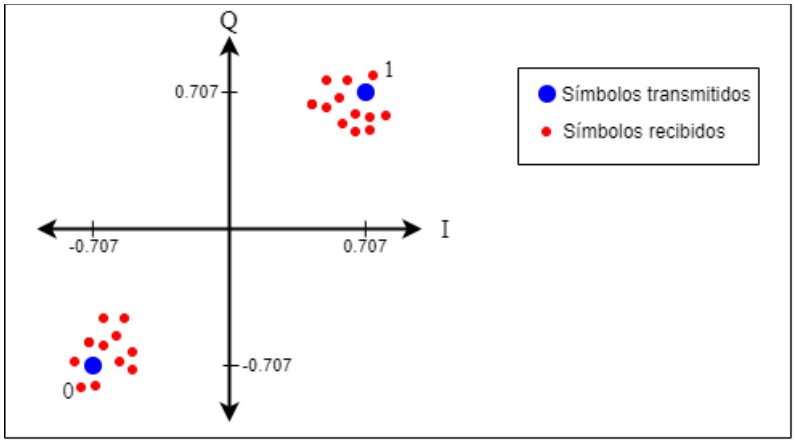


Figura 2.158. Ejemplo de símbolos BPSK al pasar por el canal inalámbrico.

El demapeo BPSK se implementa en Matlab de manera manual, mediante un condicional if se discrimina en que parte del cuadrante se encuentra en símbolo recibido.

Cuando el símbolo recibido es mayor o igual a cero en la parte real e imaginaria, el valor recuperado es 1. Si el símbolo es menor que cero en la parte real e imaginaria el valor recuperado es 0. Para mostrar un ejemplo de mapeo y demapeo BPSK se implementa una sección de código en el script *EjemploDemapeo.m*. La sección de código inicia por establecer el vector de bits de estrada que es; 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1 y 0.

Luego se mapean los bits de entrada y el resultado se almacena en el vector *datosMapeados*. Para mayor detalle del mapeo se recomienda revisar la sección Mapeo del capítulo de transmisión. Luego se añade ruido blanco gaussiano con la función *awgn* y el resultado con ruido se guarda en el vector *datosMapeo*. Después se inicia con ceros el vector *datosDemod*, el cual almacena los bits demapeados. A continuación, se utiliza un

lazo *for* para recorrer todos los elementos del vector *datosMapeo* y mediante un *if* se discrimina en qué cuadrante está el símbolo que se evalúa.

Finalmente, se grafican los bits de entrada y los bits recuperados demapeados. También se grafica el diagrama de constelación sin ruido y con ruido. El código descrito en estos párrafos se presenta a continuación, debidamente comentado.

```
%% Sección 1: Ejemplo de Demapeo BPSK
datos_C=[1;0;1;1;0;0;0;1;0;1;1;1;0;0;1;1;0;1;1;0]; % Bits de entrada
% Mapeo BPSK:
for i=1:length(datos_C) % Se recorre los bits del vector de entrada
    % Con el lazo if se discrimina si es igual a 1 y se asigna el
    % valor 0.707+0.707i, caso contrario es 0 y se asigna el valor de
    %  -0.707-0.707i
    if(datos_C(i)==1)
        datosMapeados(i,1)=complex(0.707,0.707);
    else
        datosMapeados(i,1)=complex(-0.707,-0.707);
    end
end
% El vector con los datos mapeados es datosMapeados
% Añadir ruido:
datosMapeo= awgn(datosMapeados,20,'measured'); % Señal con ruido,
% Demapeo BPSK:
datosDemod=zeros(length(datosMapeo),1); % Iniciar vector para
% almacenar los datos demapeados
for i=1:length(datosMapeo) % Recorrer los datos con ruido
    % Con un if se evalua en que cuadrante está
    if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
        datosDemod(i)=1;
    elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
        datosDemod(i)=0;
    end
end
% Graficar los datos de entrada y salida, tambien las costelaciones:
figure()
subplot(1,2,1)
stem(datos_C,'b','filled');hold on
stem(datosDemod,'r','filled','MarkerSize',4)
title('Bits de entrada y bits recuperados'); xlabel('Muestras');
legend('Entrada', 'Recuperados'); ylim([-0.1 1.3])
subplot(1,2,2)
plot(datosMapeados,'b.','MarkerSize',20); hold on;
plot(datosMapeo,'r.','MarkerSize',10); grid on;
title('Mapeo BPSK'); legend('Sin ruido', 'Con ruido');
xlabel('I (fase)'); ylabel('Q (cuadratura)'); axis([-1 1 -1 1]);
```

Luego de ejecutar la primera sección del script *EjemploDemapeo.m* se obtiene las variables del espacio de trabajo que se muestran en la Figura 2.159 y también las gráficas que se muestran en la Figura 2.160.

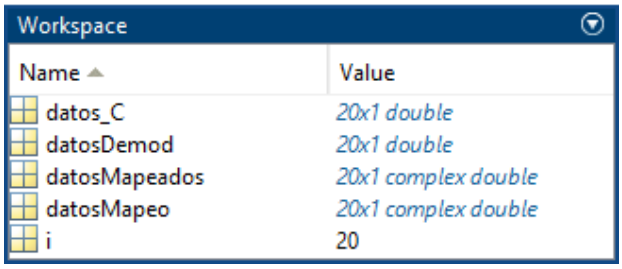


Figura 2.159. Variables en el espacio de trabajo, luego de ejecutar la modulación y demodulación BPSK.

En la Figura 2.160a se muestra con color azul los bits de entrada y con color rojo los bits de salida, es decir los bits recuperados luego de ser mapeados, añadido ruido y demapeados. Mientras que en la Figura 2.160b se muestra con azul los símbolos mapeados sin ruido y con rojo los símbolos mapeados con ruido. Se observa que cuando se añade ruido el diagrama de constelación se dispersa.

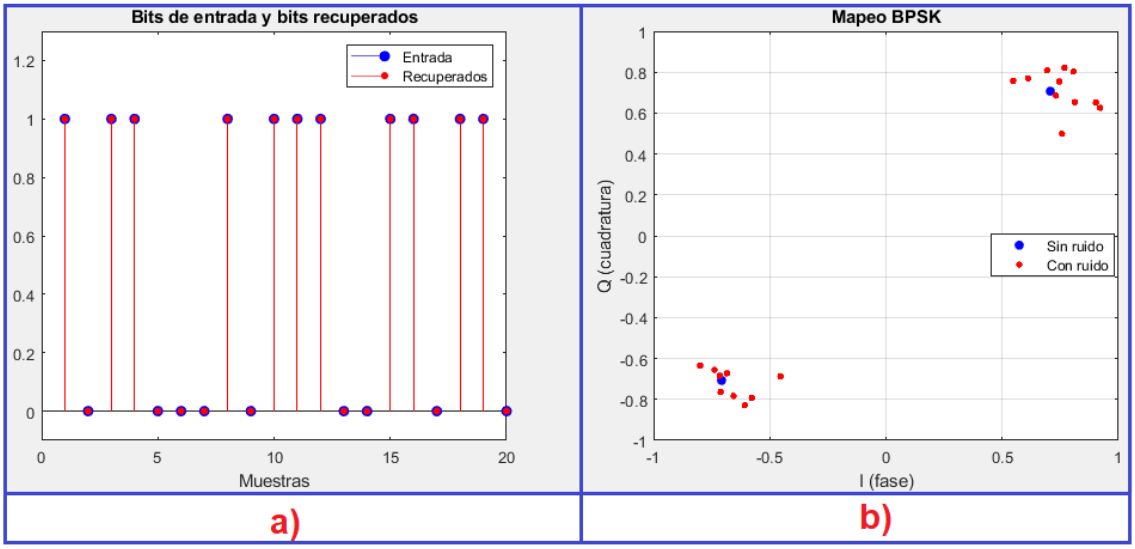


Figura 2.160b. Ejemplo de BPSK. a) Bits de entrada y bits de salida. b) Diagrama de constelación con y sin ruido.

2.5.5.1.2. Demapeo QPSK en Matlab

El mapeo QPSK tiene cuatro símbolos en su diagrama de constelación y utiliza dos bits de entrada por cada símbolo. En la Figura 2.161 se muestra el diagrama de constelación del mapeo QPSK con codificación GRAY y sin normalizar. Cabe recordar que antes de transmitir los símbolos es necesario normalizar los estos símbolos multiplicándolos por $1/\sqrt{2}$. Por lo cual, en recepción se multiplica por $\sqrt{2}$ para poder demapear los símbolos y recuperar los bits de información.

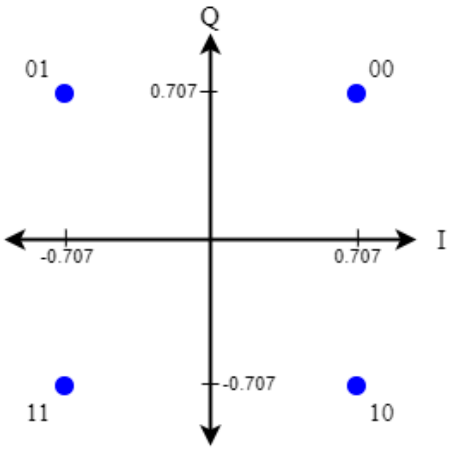


Figura 2.161. Diagrama de constelación QPSK con codificación GRAY y sin normalizar.

El demapeo QPSK se implementa en Matlab mediante funciones directas, lo mismo ocurre con el demapeo 16-QAM y 64-QAM.

La función directa que permite implementar el demapeo QPSK es *pskdemod* y tiene la siguiente sintaxis:

datosDemod = pskdemod(datosM,M,ini_phase,symorder)

La función *pskdemod* permite implementar demodulación digital PSK [41]. En el parámetro de salida *datosDemod* se almacena los símbolos decimales que se han demodulado con PSK. Para una demodulación QPSK se obtendrán como salida símbolos del 0 al 3. Los parámetros de entrada se describen a continuación:

- **datosM:** Es un vector columna que contiene los símbolos modulados con PSK y sin ninguna normalización.
- **M:** Es el orden de modulación, se especifica como una potencia de 2. Corresponde a la cantidad de símbolos en la constelación. Para QPSK es igual a 4.
- **ini_phase:** Es la fase inicial de la modulación PSK, especificada en radianes como un escalar real. El valor por defecto es 0. Para demodular en QPSK se utiliza el valor de pi/4.
- **symorder:** Es la codificación que se utiliza para recuperar los símbolos del diagrama de constelación. Puede utilizarse codificación binaria o codificación gray, especificados como 'bin' o 'gray' respectivamente. El valor por defecto es 'bin'. Para el demapeo QPSK con el que se está trabajando se asigna como 'gray'.

En la sección 2 del script *EjemploDemapeo.m* se implementa un ejemplo de mapeo y demapeo QPSK. El código inicia borrando las variables del espacio de trabajo. Luego se define los símbolos decimales que sirven como entrada para el mapeo QPSK. Después se

define el número de puntos en el diagrama de constelación, se mapea con QPSK utilizando la función de Matlab *pskmod* y se normalizan estos datos. La función *pskmod* se explicó en el capítulo de transmisión.

A los datos mapeados y normalizados se añade ruido blanco gaussiano mediante la función *awgn*. Luego se quita la normalización y se aplica la función *pskdemod* para recuperar los datos de entrada. Después se grafican los símbolos decimales de entrada y los recuperados, también se grafica el diagrama de constelación normalizado con y sin ruido. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Sección 2: Ejemplo de Demapeo QPSK
clear % Limpiar el espacio de trabajo
datos_C=[1;0;3;2;2;1;3;0;0;1;2;3;0;1;3;3;3;1;0;2]; % Simb de entrada
% Mapeo QPSK:
M = 4; % Puntos en la constelación (símbolos)
datosM = pskmod(datos_C,M,pi/4,'gray'); % Datos modulados QPSK
% Factor de normalización = 1/sqrt(2)
datosMapeados=(1/sqrt(2))*datosM; % Vector con datos modulados y norm
% Añadir ruido:
datosMapeo= awgn(datosMapeados,20,'measured'); % Señal con ruido
% Demapeo QPSK:
datosMapeo1=sqrt(2)*datosMapeo; % Multiplicar por el inverso del
% factor de normalización
datosDemod=pskdemod(datosMapeo1,M,pi/4,'gray'); % Demodulación QPSK
% Graficar los datos de entrada y salida, tambien las costelaciones:
figure()
subplot(1,2,1)
stem(datos_C,'b','filled');hold on
stem(datosDemod,'r','filled','MarkerSize',4)
title('Símbolos de entrada y símbolos recuperados');xlabel('Muestras');
legend('Entrada', 'Recuperados'); ylim([-0.1 3.7])
subplot(1,2,2)
plot(datosMapeados,'b.','MarkerSize',20); hold on;
plot(datosMapeo,'r.','MarkerSize',10); grid on;
title('Diagrama de constelación normalizado');
legend('Sin ruido', 'Con ruido');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-0.8 0.8 -0.8 0.8]);
```

Luego de ejecutar la sección 2 del script *EjemploDemapeo.m* se obtiene las variables del espacio de trabajo que se muestran en la Figura 2.162 y también las gráficas que se muestran en la Figura 2.163.

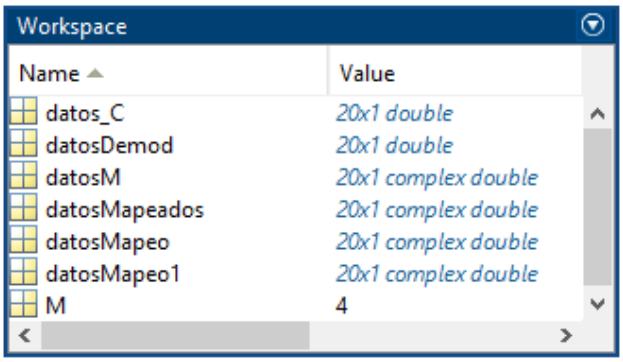


Figura 2.162. Variables en el espacio de trabajo, luego de ejecutar la modulación y demodulación QPSK.

En la Figura 2.163a se muestra con color azul los símbolos decimales de entrada y con color rojo los símbolos de salida, es decir los símbolos recuperados luego de ser mapeados, añadido ruido y demapeados. Se puede ver que se recupera satisfactoriamente los símbolos decimales de entrada. En la Figura 2.163b se muestra con azul los símbolos mapeados normalizados sin ruido y con rojo los símbolos mapeados normalizados con ruido. Al añadir ruido el diagrama de constelación se dispersa.

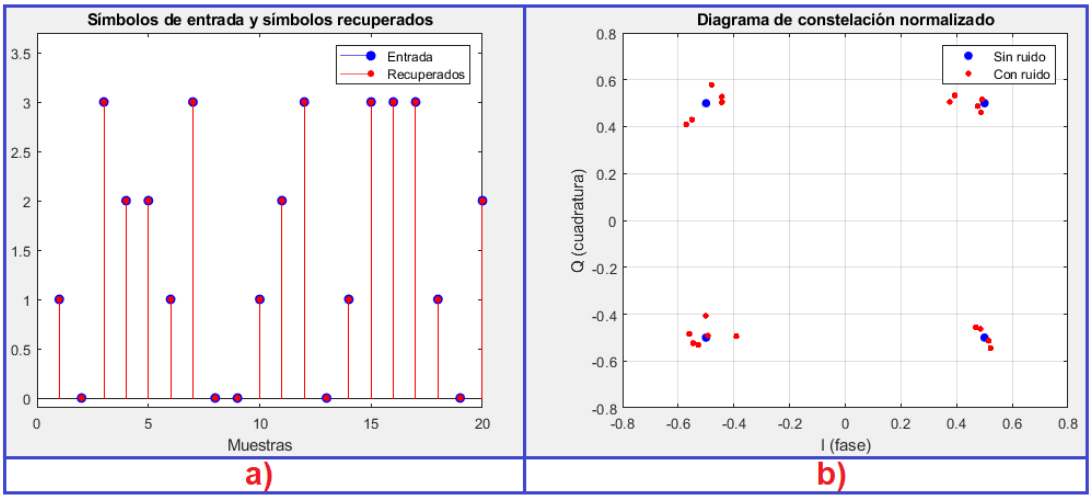


Figura 2.163. Ejemplo de QPSK. a) Símbolos decimales de entrada y de salida. b) Diagrama de constelación con y sin ruido.

2.5.5.1.3. Demapeo 16-QAM en Matlab

El mapeo 16-QAM utiliza 4 bits por cada símbolo en el diagrama de constelación. En la Figura 2.164 se muestra el diagrama de constelación del mapeo QPSK, con la respectiva codificación en cada símbolo. El factor de normalización que se utiliza en 16-QAM es $1/\sqrt{10}$ y por lo tanto en recepción se debe multiplicar los datos por $\sqrt{10}$ antes de aplicar el demapeo.

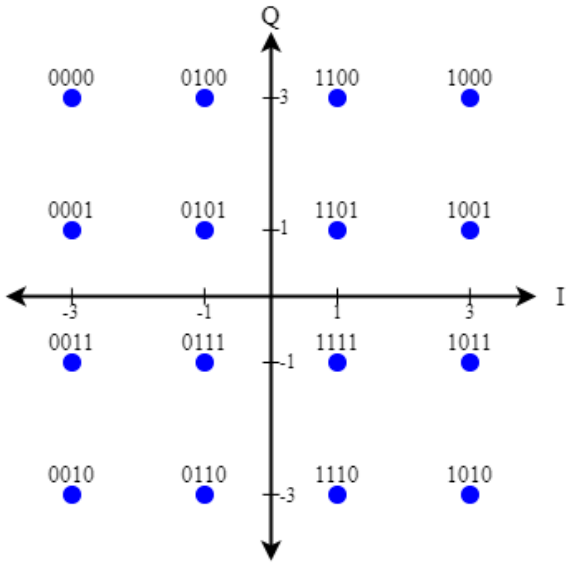


Figura 2.164. Diagrama de constelación del mapeo 16-QAM, con codificación GRAY.

Para implementar el demapeo 16-QAM y 64-QAM se utiliza la función directa de Matlab llamada *qamdemod*.

La sintaxis de la función *qamdemod* es la siguiente:

datosDemod = qamdemod(datosM,M,symorder)

La función *qamdemod* permite implementar demodulación digital QAM [42]. En el parámetro de salida *datosDemod* se almacena los símbolos decimales que se han demodulado utilizado la técnica M-QAM, en donde *M* es un parámetro de entrada. Los parámetros de entrada se describen a continuación:

- **datosM:** Es un vector columna que contiene los símbolos modulados con M-QAM y sin ninguna clase de normalización.
- **M:** Es el orden de modulación, se especifica como una potencia de 2. Corresponde a la cantidad de símbolos en la constelación. Para 16-QAM este parámetro es igual a 16 y para 64-QAM este parámetro es igual a 64.
- **symorder:** Es la codificación que se utiliza para recuperar los símbolos del diagrama de constelación. Puede utilizarse codificación binaria o codificación gray, especificados como 'bin' o 'gray' respectivamente. El valor por **defecto es 'gray'**. Para el demapeo 16-QAM y 64-QAM se debe utilizar la codificación Gray.

En la sección 3 del script *EjemploDemapeo.m* se implementa un ejemplo de mapeo y demapeo 16-QAM. El código inicia borrando las variables del espacio de trabajo. Luego se crea un vector aleatorio que contiene símbolos decimales del 0 al 15 y que tiene una longitud igual a 64, este vector contiene los símbolos de entrada dele ejemplo. Después se

define el número de puntos en el diagrama de constelación, que para 16-QAM es igual a 16. Luego se mapea los símbolos de entrada utilizando la función *qammod*, para mayor detalle de la modulación se recomienda consultar el capítulo de transmisión. Además, se normalizan los datos mapeados.

A los datos mapeados y normalizados se añade ruido blanco gaussiano mediante la función *awgn*. Luego se quita la normalización y a estos datos se aplica la función *qamdemod* para recuperar los datos de entrada. Después se grafican los símbolos decimales de entrada y los recuperados, también se grafica el diagrama de constelación normalizado con y sin ruido. El código descrito en estos párrafos se presenta a continuación debidamente comentado.

```
%% Sección 3: Ejemplo de Demapeo 16-QAM
clear % Limpiar el espacio de trabajo
datos_C=randi([0 15],64,1); % Simb de entrada
% Mapeo 16-QAM:
M = 16; % Puntos en la constelación (símbolos)
datosM = qammod(datos_C,M); % Datos modulados 16-QAM
% Factor de normalización = 1/sqrt(10)
datosMapeados=(1/sqrt(10))*datosM; % Vector con datos modulados y norm
% Añadir ruido:
datosMapeo= awgn(datosMapeados,20,'measured'); % Señal con ruido
% Demapeo 16-QAM:
datosMapeo1=sqrt(10)*datosMapeo; % Multiplicar por el inverso del
% factor de normalización
datosDemod=qamdemod(datosMapeo1,M); % Demodulación 16_QAM
% Graficar los datos de entrada y salida, tambien las costelaciones:
figure()
subplot(1,2,1)
stem(datos_C,'b','filled');hold on
stem(datosDemod,'r','filled','MarkerSize',4)
title('Símbolos de entrada y símbolos recuperados');xlabel('Muestras');
legend('Entrada', 'Recuperados'); ylim([-0.1 18])
subplot(1,2,2)
plot(datosMapeados,'b.','MarkerSize',20); hold on;
plot(datosMapeo,'r.','MarkerSize',10); grid on;
title('Diagrama de constelación normalizado');
legend('Sin ruido', 'Con ruido');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-1.3 1.3 -1.3 1.3]);
```

Luego de ejecutar la sección 3 del script *EjemploDemapeo.m* se obtiene las variables del espacio de trabajo que se muestran en la Figura 2.165 y también las gráficas que se muestran en la Figura 2.166.

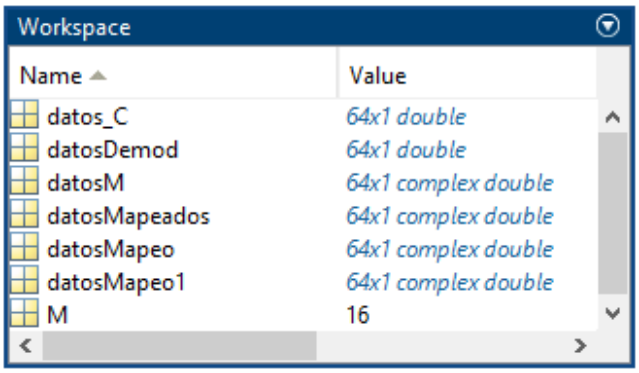


Figura 2.165 Variables en el espacio de trabajo, luego de ejecutar la modulación y demodulación 16-QAM.

En la Figura 2.166a se muestra con color azul los símbolos de entrada y con rojo los símbolos de recuperados luego de mapear, añadir ruido y demapear los símbolos de entrada. También se observa que se recuperan satisfactoriamente los símbolos decimales de entrada. En la Figura 2.166b se muestra con azul los símbolos mapeados normalizados sin ruido y con rojo los símbolos mapeados normalizados con ruido. Al igual que en los casos anteriores se observa que al añadir ruido, los símbolos se dispersan.

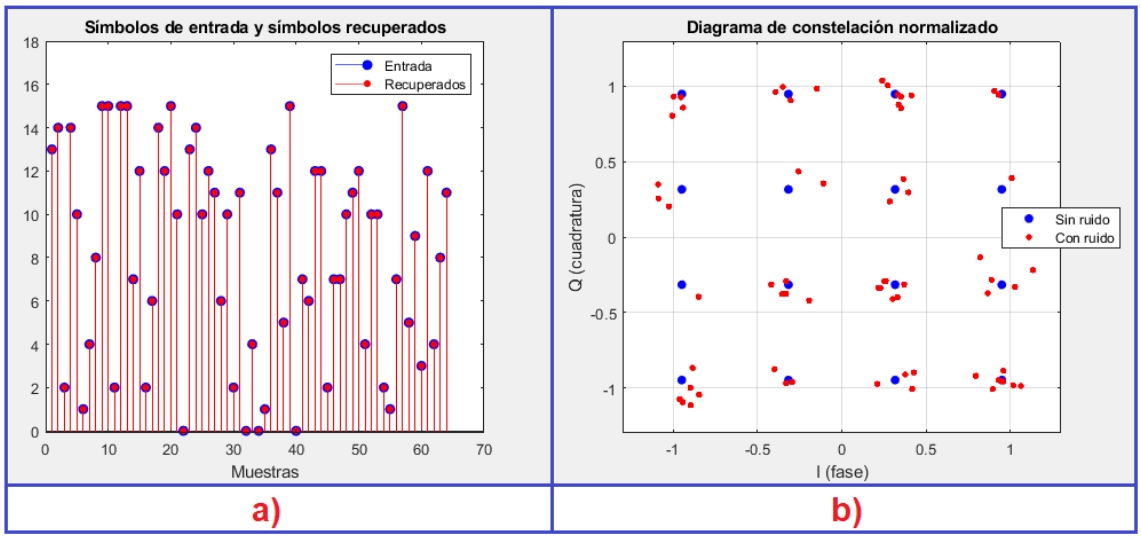


Figura 2.166. Ejemplo de 16-QAM. a) Símbolos decimales de entrada y de salida. b) Diagrama de constelación con y sin ruido.

2.5.5.1.4. Demapeo 64-QAM en Matlab

El mapeo 64-QAM utiliza 6 bits por cada símbolo presente en su diagrama de constelación. En la Figura 2.167 se muestra el diagrama de constelación de la modulación 64-QAM, los símbolos se muestran en formato decimal y además están dispuestos en codificación Gray. El factor de normalización de esta técnica de mapeo es $1/\sqrt{42}$, por ende el factor de multiplicación en la demodulación es $\sqrt{42}$.

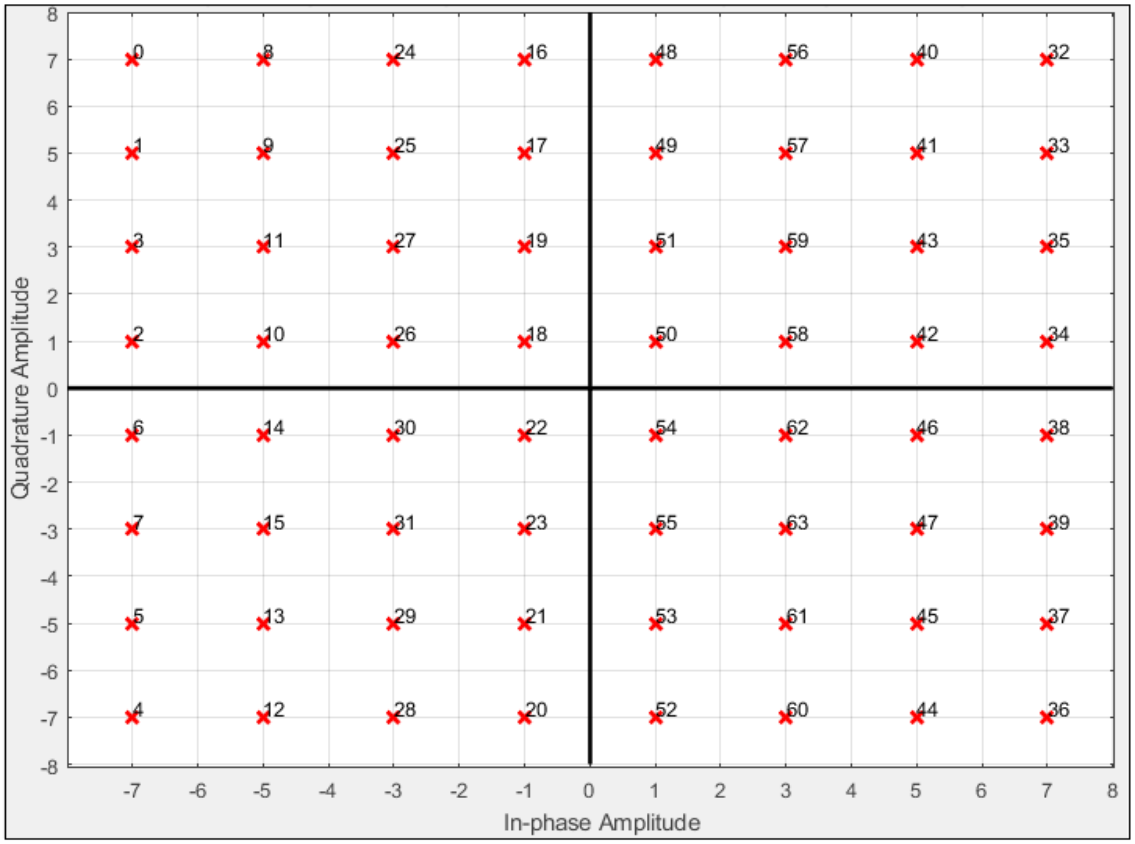


Figura 2.167. Diagrama de constelación para mapeo 64-QAM, con codificación GRAY.

El demapeo 64-QAM se implementa mediante la función directa *qamdemod*, la cual ya se explicó previamente.

En la sección 4 del script *EjemploDemapeo.m* se implementa un ejemplo de mapeo y demapeo 64-QAM. El mismo inicia por limpiar las variables del espacio de trabajo. A continuación, se crea un vector columna de longitud 256 y que contiene números enteros aleatorios entre 0 y 63. Luego se define el número de símbolos en la constelación, es decir 64. Seguido se modulan los símbolos decimales de entrada con *qammod* y además se normalizan los símbolos modulados. La función *qammod* se abordó en el capítulo de transmisión en la etapa mapeo.

Luego de normalizar los símbolos mapeados se añade ruido a los mismos. Paso seguido se quita la normalización de los símbolos con ruido. Después se aplica la demodulación 64-QAM con la función *qamdemod*. Finalmente, se grafican los símbolos de entrada versus los de salida, también se grafican los diagramas constelación normalizados con y sin ruido. A continuación, se presenta la sección de código descrita en estos dos últimos párrafos.

```
%% Sección 4: Ejemplo de Demapeo 64-QAM
clear % Limpiar el espacio de trabajo
datos_C=randi([0 63],256,1); % Simb de entrada
% Mapeo 64-QAM:
```

```
M = 64; % Puntos en la constelación (símbolos)
datosM = qammod(datos_C,M); % Datos modulados 64-QAM
% Factor de normalización = 1/sqrt(42)
datosMapeados=(1/sqrt(42))*datosM; % Vector con datos modulados y norm
% Añadir ruido:
datosMapeo= awgn(datosMapeados,20,'measured'); % Señal con ruido
% Demapeo 64-QAM:
datosMapeo1=sqrt(42)*datosMapeo; % Multiplicar por el inverso del
% factor de normalización
datosDemod=qamdemod(datosMapeo1,M); % Demodulación 64-QAM
% Graficar los datos de entrada y salida, tambien las costelaciones:
figure()
stem(datos_C,'b','filled');hold on
stem(datosDemod,'r','filled','MarkerSize',4)
title('Símbolos de entrada y símbolos recuperados');xlabel('Muestras');
legend('Entrada', 'Recuperados'); ylim([-1 68]); hold off
figure()
plot(datosMapeados,'b.','MarkerSize',20); hold on;
plot(datosMapeo,'r.','MarkerSize',10); grid on;
title('Diagrama de constelación normalizado');
legend('Sin ruido', 'Con ruido');
xlabel('I (fase)');ylabel('Q (cuadratura)');axis([-1.3 2 -1.3 1.3]);
```

Luego de ejecutar la sección 4 del script *EjemploDemapeo.m* se obtiene las variables del espacio de trabajo que se muestran en la Figura 2.168, también las gráficas de la Figura 2.169 y de la Figura 2.170.

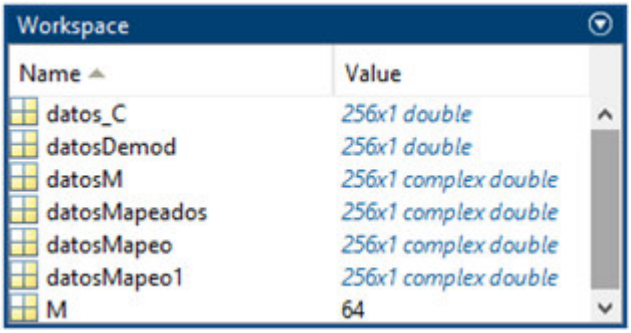


Figura 2.168. Variables en el espacio de trabajo, luego de ejecutar la modulación y demodulación 16-QAM.

En la Figura 2.169 se muestra con color azul los símbolos de entrada y con rojo los símbolos de recuperados luego de mapear, añadir ruido y demapear los símbolos de entrada. En este caso, la mayoría de los símbolos decimales pueden recuperarse satisfactoriamente, pero ya aparecen algunos errores en la recuperación. Esto se debe a la dispersión de datos en el diagrama de constelación.

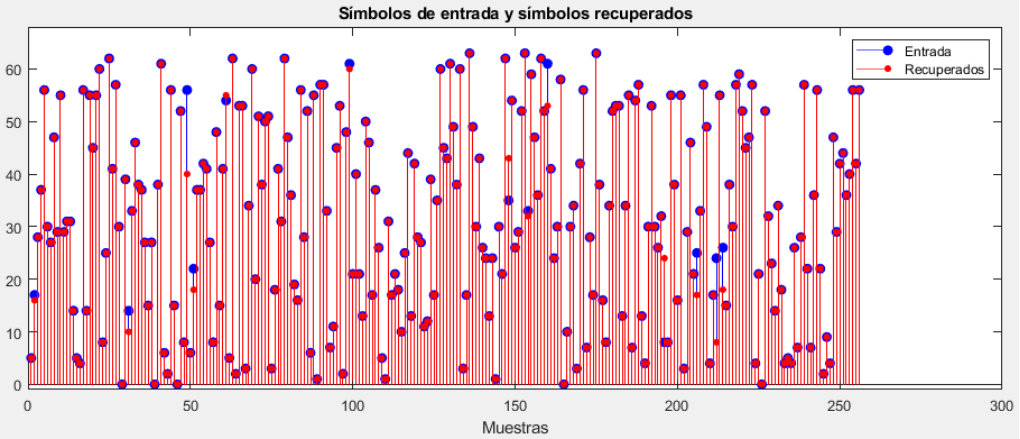


Figura 2.169. Símbolos decimales de entrada y de salida en el ejemplo de mapeo y demapeo 64-QAM.

En la Figura 2.170 se muestra con azul los símbolos mapeados normalizados sin ruido y con rojo los símbolos mapeados normalizados con ruido. Al igual que en los casos anteriores se observa que al añadir ruido los símbolos se dispersan. En este caso resulta que los símbolos están más cerca entre ellos, esto conlleva a que la cantidad de errores aumente al momento de la demodulación.

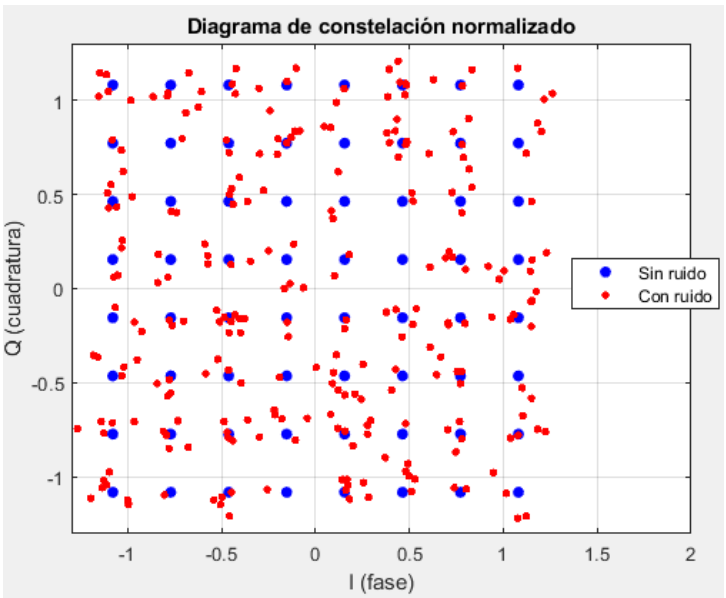


Figura 2.170. Diagrama de constelación del mapeo 64-QAM con y sin ruido.

2.5.5.2. Demapeo QPSK en el archivo de texto recibido

En transmisión se enviaron los datos mapeados con QPSK, por lo que es necesario demapear los datos con QPSK para poder recuperarlos. Previo a implementar el demapeo de datos en el archivo de texto recibido se debe cargar los datos recuperados del canal contenidos en *datosRecibidos150.mat*. Luego se debe ejecutar el script *recortar_rx.m*, correspondiente a la etapa Recortado de datos. Después se ejecutan los scripts correspondientes al procesamiento del preámbulo IEE 802.11a que son; *recortar_rx.m*,

encontrarTrama_rx.m, *corrCFO_rx.m*, *SincSimbolo_rx.m* y *corrFase_rx.m*. Luego se ejecuta el script *ofdm_rx.m* que corresponde al procesamiento OFDM. También se debe ejecutar el script *reformado_rx.m* correspondiente a la etapa de reformado de pulso.

El vector de entrada para el demapeo es *datosMapeo* y se obtiene del script *reformado_rx.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando

```
clearvars -except datosMapeo % Borrar todas las variables
% excepto la variable datosDemod (entrada demapeo_datos.m)
```

El demapeo, aplicable al archivo de texto recibido y procesado hasta la etapa de reformado de pulso, se implementa en un script llamado *demapeo_datos.m*. Este script permite elegir entre el demapeo BPSK, QPSK, 16-QAM y 64-QAM mediante la variable *seleccionMapeo* y un condicional *if*. Dicha variable debe tomar el valor de 10 cuando se quiera demapear con BPSK, 20 con QPSK, 30 con 16-QAM y 40 con 64-QAM. El vector de entrada del script es *datosMapeo* y el de salida es *datosDemod*. El demapeo BPSK se implementa manualmente sin funciones directas, una vez demapeados los datos se almacenan en *datosDemod* que corresponde.

Para implementar el demapeo QPSK, 16-QAM y 64-QAM los pasos son similares, se inicia por definir los puntos del diagrama de constelación (*M*), se calcula el número de bits por cada símbolo (*Nb*). Luego, en caso de ser necesario, se realiza un relleno de ceros para que el vector de entrada sea múltiplo de los bits empleados por símbolo. El relleno se almacena en el vector *paddM*. Luego se quita la normalización de los símbolos mapeados con los factores de la Tabla 2.16 y se demapea los datos mediante funciones directas (explicadas anteriormente). Los símbolos decimales recuperados en el vector *datosDemS*, se transforman a bits, se serializan y se almacenan en el vector *datosDemod*. Este vector es el vector de salida de la etapa.

Como los datos que se están procesando se enviaron mapeados con QPSK, el demapeo debe hacerse tambien con QPSK, por esto a la variable *seleccionMapeo* se le asigna el valor de 20. El código del script *demapeo_datos.m* se presenta a continuación, debidamente comentado.

```
% Script que permite aplicar el demapeo y recuperar los datos
% codificados
% Antes de correr este script es necesario cargar los datos guardados
%en datosRecibidos150.mat, ejecutar recortar_rx.m, encontrarTrama_rx.m,
% corrCFO_rx.m, SincSimbolo_rx.m, corrFase_rx.m, ofdm_rx,m y
% reformado_rx.m
% En vector de entrada de esta etapa es datosMapeo
```

```
% El valor en la variable seleccionMapeo define el tipo de demapeo a
% utilizar: BPSK (10), QPSK (20), 16-QAM(30) o 64-QAM(40)
seleccionMapeo=20; % Se selecciona demapeo QPSK, ya que se trabaja con
% este para los ejemplos
% Con un lazo if se discrimina el demapeo a utilizar
if(seleccionMapeo==10)
    % Demapeo BPSK
    % Los datos filtrados no se normalizan
    longitud=length(datosMapeo); % Longitud datos de entrada
    datosDemod=zeros(longitud,1); % Inicializar vector
    % Implementar el Demapeo BPSK con el lazo for
    for i=1:longitud
        if(real(datosMapeo(i))>=0 && imag(datosMapeo(i))>=0)
            datosDemod(i)=1;
        elseif(real(datosMapeo(i))<0 && imag(datosMapeo(i))<0)
            datosDemod(i)=0;
        end
    end
elseif(seleccionMapeo==20)
    % Demapeo QPSK
    M = 4; % Puntos en la constelación (símbolos)
    Nb=log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(2)
    ModDat=(sqrt(2))*[datosMapeo;paddM]; %Datos con relleno
    % Demodulación QPSK:
    datosDemodS=pskdemod(ModDat,M,pi/4,'gray');
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    datosDemod=reshape(demodMatriz.',1,[]).'; % Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==30)
    % Demapeo 16-QAM
    M = 16; % Puntos en la constelación (símbolos)
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(10)
    ModDat=(sqrt(10))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 16-QAM:
    datosDemodS=qamdemod(ModDat,M);
    % Pasar de símbolos a bits:
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
elseif(seleccionMapeo==40)
    % Demapeo 64-QAM
    M = 64; % Puntos en la constelación (símbolos)
    Nb = log2(M); % Bits por símbolo
    % Relleno de ceros:
    long_pM=ceil(length(datosMapeo)/Nb)*Nb-length(datosMapeo);
    paddM=zeros(long_pM,1); % relleno con ceros
    % Los datos filtrados solo se multiplican por sqrt(42)
    ModDat=(sqrt(42))*[datosMapeo;paddM]; % Datos con relleno
    % Demodulación 64-QAM:
    datosDemodS=qamdemod(ModDat,M);
    % Pasar de símbolos a bits:
```

```
    demodMatriz=de2bi(datosDemodS,Nb,'left-msb');
    datosDemod=reshape(demodMatriz.',1,[]).'; %Datos
    % recuperados (bits Serializados)
end
```

Luego de realizar el demapeo QPSK con el script *demapeo_datos.m* se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.171. En la misma se señala con rojo el vector de entrada y con verde el vector de salida de la etapa, el cual contiene los bits codificados serializados. El vector de salida tiene el doble del tamaño del vector de entrada ya que se utiliza dos bits por cada símbolo en la constelación. Además, ya que en este caso no se requirió ningún relleno el vector correspondiente al mismo *paddM* está vacío.

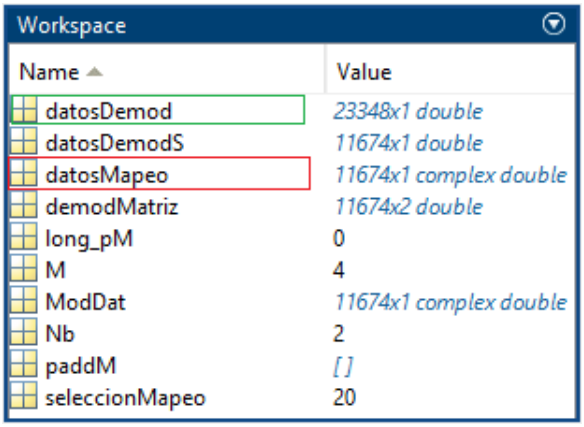


Figura 2.171. Variables en el espacio de trabajo luego de ejecutar el script *demapeo_datos.m*.

A continuación, se muestran los primeros 24 bits recuperados luego de ser demapeados con QPSK.

```
>> datosDemod(1:24).'
ans =
  Columns 1 through 14
     0     0     1     1     0     1     1     1     0     0     1     1     0     1
  Columns 15 through 24
     0     0     0     1     0     0     1     1     1     0
```

2.5.6. ETAPA DE DECODIFICACIÓN CONVOLUCIONAL

En la etapa actual se implementa la decodificación convolucional a través del algoritmo de Viterbi. La codificación y decodificación convolucional en conjunto permiten corregir errores a nivel de bit. El proceso de codificación convolucional se implementó en transmisión, mientras que el decodificador se implementa en recepción, para poder recuperar los **bits de información**. En la Figura 2.172 se identifica con color anaranjado la etapa Codificador Convolucional dentro de las etapas del procesamiento en recepción.

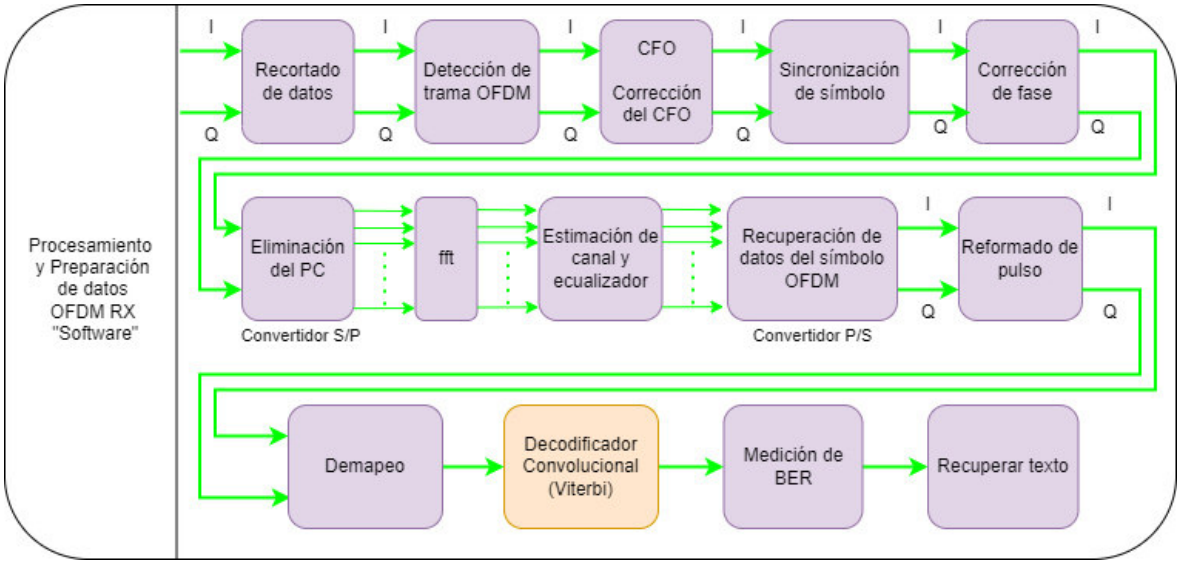


Figura 2.172. Etapas de recepción con la etapa Decodificador Convolucional señalada con naranja.

En el prototipo que se está desarrollando la decodificación convolucional se implementa mediante el algoritmo de Viterbi.

Este algoritmo utiliza el diagrama de Trellis para buscar el camino más probable dentro de una secuencia de bits de entrada, los cuales han sido codificados previamente. El algoritmo de Viterbi no almacena todas las posibles secuencias codificadas, si no que almacena el camino más probable para cada posible estado de salida. El camino más probable o también llamado mejor camino, se determina en base a la distancia de una palabra código hacia otra palabra código [16].

La distancia de una palabra código asociada con una transición entre etapas, se denomina como la **métrica de rama**, mientras que la métrica acumulada de los estados anteriores y el actual se conoce como **métrica de camino** [17]. Las **métricas de rama** se calculan para todas las transiciones existentes pero solo se mantiene la menor métrica en cada estado y en base al valor obtenido se calcula la métrica obtenida.

Los pasos para decodificar son:

1. Generar el diagrama de Trellis con los mismos polinomios generadores utilizados en la parte de transmisión.
2. Para el nivel actual se calcula la **métrica de rama** en base a los datos que se tenga como entrada.
3. Luego se calcula la **métrica de camino**, tal que en cada nodo se mantenga el valor de la menor métrica.
4. Se repite el paso 2 y 3 hasta que se terminen los datos de entrada.

5. Se escoge la ruta de la menor **métrica de camino** que se obtuvo en los nodos finales. La ruta elegida se denomina **ruta superviviente** [16]. Esta ruta se refiere al camino que recorrió a lo largo de los diferentes nodos hasta llegar al nodo final.

Ejemplo de decodificación:

Para explicar a mayor detalle la decodificación convolucional mediante el algoritmo de Viterbi, se desarrolla un ejemplo. En el capítulo de Transmisión, en la sección del Codificador Convolucional, se codificó la secuencia de bits **1101** y se obtuvo la secuencia **11101000**. Si a esa secuencia se introduce un bit errado se puede obtener la secuencia **11101100**, esta secuencia se utiliza como datos de entrada de este ejemplo.

En diagrama de Trellis que se creó en transmisión es el mismo que se utiliza para decodificar, este diagrama se muestra en la Figura 2.173. Este diagrama utiliza polinomios generadores $g_1 = 5$ y $g_2 = 7$, contiene 4 estados en cada valor de tiempo discreto n.

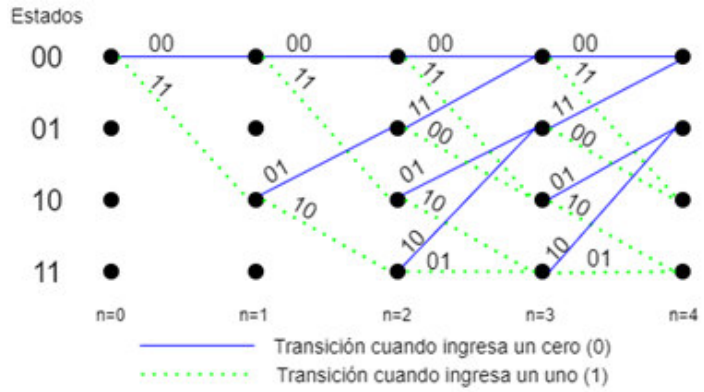


Figura 2.173. Diagrama de Trellis de un codificador convolucional (2, 1, 3), con $g_1 = 5$ y $g_2 = 7$.

En la Figura 2.174, en la parte superior del diagrama de Trellis se ubica la secuencia de entrada **11101100**, tal que entre cada tiempo discreto n se coloca un número de bits de la misma longitud de la palabra código. Esto se realiza con el objetivo de visualizar los bits de entrada correspondiente a los intervalos de las palabras código. Además, en la Figura 2.174 se señala con color naranja la **métrica de camino**, que en n=0 está en el estado inicial y por tanto es igual a cero. La **métrica de rama** se coloca a un lado de las palabras código tal que; $'palabra\ código'/'métrica\ de\ rama'$. Hay que recordar que la palabra código son los dos dígitos binarios ubicados sobre la línea azul (cuando la entrada es cero) y cobre la línea verde punteada (cuando la entrada es uno).

La **métrica de rama** se calcula en base a los dos dígitos correspondientes de la secuencia de entrada, es decir **11** entre n=0 y n=1. La palabra código 11 tiene una métrica 0 y la palabra código 00 tiene una métrica igual a 2, esto debido a que se encuentra a una distancia 2 del nodo de destino que ingresa la palabra código 11 (con métrica 0).

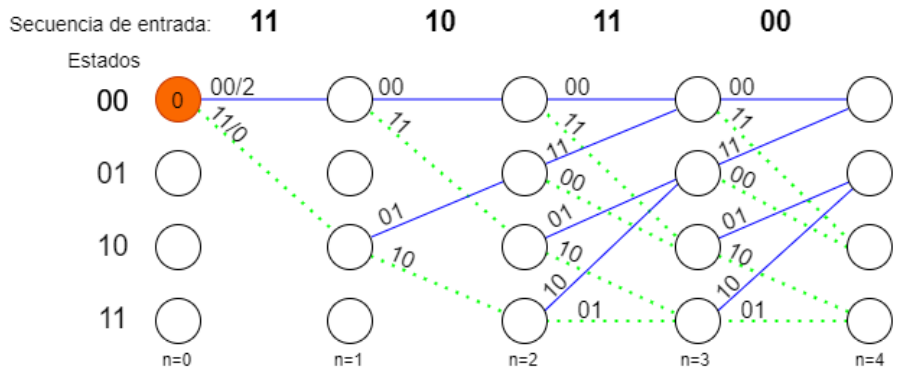


Figura 2.174. Inicio de la decodificación con el algoritmo de Viterbi utilizando el diagrama de Trellis. Ubicación del inicio de la métrica de camino (con naranja) y métrica de rama (a la derecha de la palabra código).

Luego se ubica la métrica de camino en los nodos correspondientes de n=1. En este punto del ejemplo solo existen dos nodos con un valor de métrica de camino, en la Figura 2.175 los dos nodos se colorean con naranja. Además, la secuencia de entrada que sigue es **10** y en base a la misma se calcula la métrica de rama para cada palabra código. Las métricas de rama se colocan a la derecha de todas las palabras código existentes a la salida de los nodos de n=1.

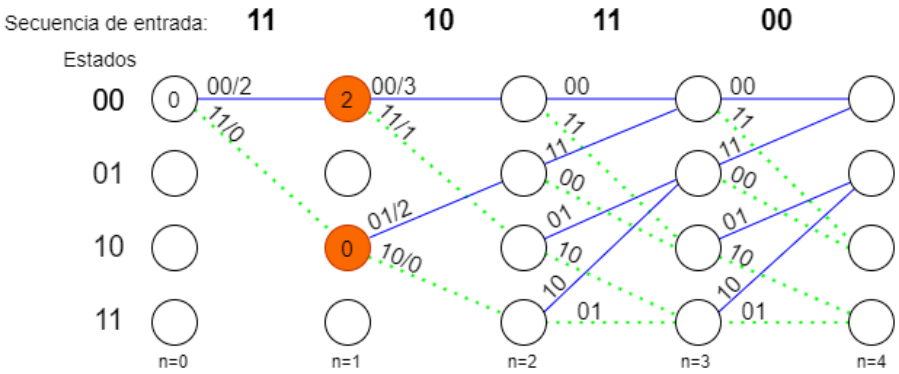


Figura 2.175. Decodificación con el algoritmo de Viterbi utilizando el diagrama de Trellis. La métrica de camino se señala con naranja y métrica de rama se coloca a la derecha de la palabra código.

Después en los nodos de n=2 se colocan las métricas de camino correspondientes, en este punto ya existe una métrica de camino en cada nodo, en la Figura 2.176, las métricas de camino se señalan con naranja. La secuencia de entrada que sigue es **11** y en base a esta se calcula las métricas de rama para cada nodo, se tiene dos por nodo. Antes del siguiente nodo se coloca con números en color rojo los valores posibles de las métricas de camino. Para cada nodo se tiene dos posibles valores, de los cuales se debe escoger el menor valor.

Cuando las posibles métricas de camino tienen igual valor, se puede escoger cualquier ruta ya que tiene la misma probabilidad.

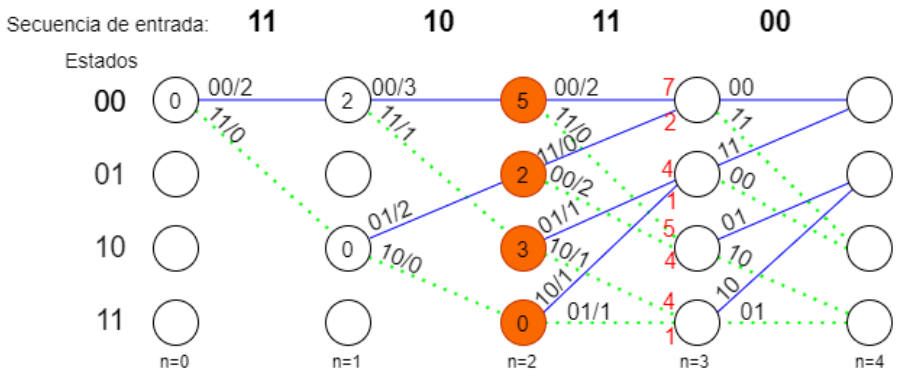


Figura 2.176. Decodificación con el algoritmo de Viterbi utilizando el diagrama de Trellis. La métrica de camino se señala con naranja, métrica de rama se coloca a la derecha de la palabra código y las posibles métricas de camino siguientes se colocan con rojo en el siguiente nodo.

En el diagrama de Trellis cuando n=3, en el nodo correspondiente al estado 00 se tiene como posibles opciones métricas de camino igual a 7 y 2, se debe escoger la menor, es decir 2. El valor de 2 se coloca dentro del nodo del estado 00, el proceso anterior se realiza para las métricas de camino de los estados restantes. En la Figura 2.177 se muestran las métricas de camino coloreadas con naranja en n=3. La secuencia de entrada que sigue es **00**, en base a este valor se calculan las métricas de rama y se colocan a la derecha de las palabras codificadas. Para los nodos de n=4 se tiene dos posibles entradas de métrica de camino y se debe escoger la menor y colocar en el nodo correspondiente.

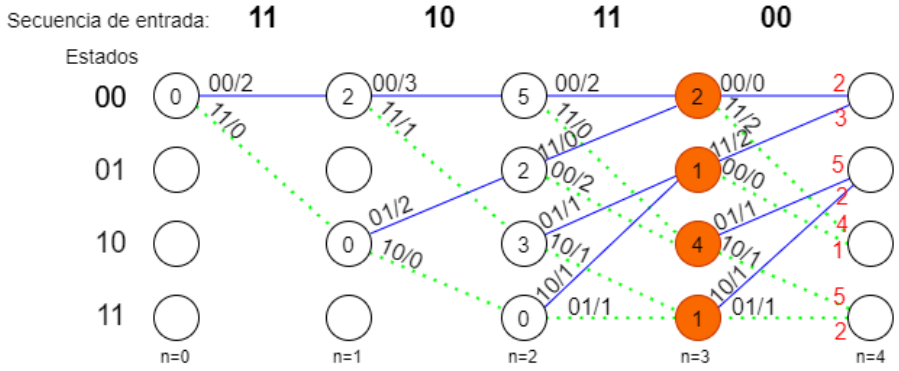


Figura 2.177. Decodificación con el algoritmo de Viterbi utilizando el diagrama de Trellis. La métrica de camino se señala con naranja, métrica de rama se coloca a la derecha de la palabra código y las posibles métricas de camino siguientes se colocan con rojo los nodos siguientes.

Una vez colocado en cada nodo de n=4, el valor final de la métrica de camino, se escoge la menor de todas estas para determinar la **ruta superviviente**. En la Figura 2.178 se muestra los valores finales de las métricas de ruta en cada nodo de n=4. Además, se señala con color amarillo los nodos de la ruta superviviente.

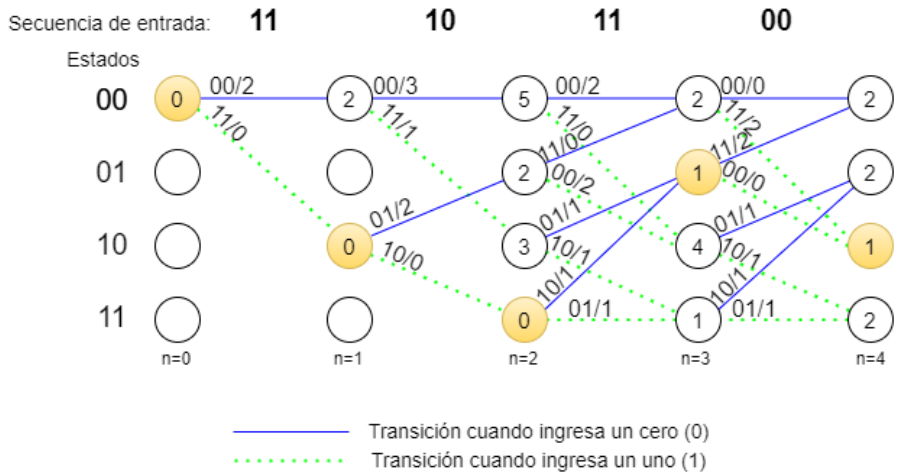


Figura 2.178. Decodificación con el algoritmo de Viterbi utilizando el diagrama de Trellis. La métrica de camino se ubica en cada nodo, métrica de rama se coloca a la derecha de la palabra código y los nodos de la ruta superviviente se señala con color amarillo.

Mediante la ruta superviviente se puede recuperar los bits sin codificar. Considerando que en el diagrama de Trellis, de la Figura 2.178, las líneas azules representan un cero (0) y las líneas verdes entrecortadas representan un uno (1), los bits decodificados son **1101**. Este valor concuerda con los bits de entrada al decodificador convolucional en transmisión.

2.5.6.1. Decodificador Convolucional en Matlab

Antes de decodificar una secuencia de bits se debe crear el diagrama de Trellis mediante la función de Matlab *polly2trellis*, esta función ya se explicó en el capítulo de transmisión. Para implementar un decodificador convolucional que emplee el algoritmo de Viterbi, se utiliza la función de Matlab *vitdec* con tiene la siguiente sintaxis.

decod_out = vitdec(codedin,P_trellis,tbdepth,opmode,dectype)

La función *vitdec* permite decodificar convolucionalmente datos binarios del vector *codedin* utilizando el algoritmo de Viterbi [43]. Los parámetros de entrada se describen a continuación.

- **codedin:** Es un vector que contiene los datos codificados convolucionalmente. El vector puede contener valores binarios o bien valores numéricos.
- **P_trellis:** Es la estructura de Trellis que se utiliza para la codificación y para la decodificación, la cual se crea con la función directa *polly2trellis*.
- **tbdepth:** Es la profundidad de rastreo y se especifica como un número entero positivo. En el proyecto se utiliza una profundidad de rastreo igual al número de bits de entrada al codificador, que será la misma cantidad de bits de salida del decodificador.

- **opmode:** Es el modo de funcionamiento y se puede especificar como; 'cont', 'term' o 'trunc'. Se establece en base a los parámetros del codificador. Con 'cont' se especifica un modo de funcionamiento continuo. Con 'term' se especifica que el codificador inició y terminó con los estados en cero. Con 'trunc' se especifica que el modo de funcionamiento es por truncado, es decir que el codificador inicio en ceros.
- **dectype:** Especifica el tipo de datos a decodificar que contiene *codedin*, puede ser 'unquant', 'hard' o 'soft'. En 'unquant' se espera valores de entrada numéricos consigo. En 'hard' se espera valores binarios. En 'soft' se espera valores de entrada enteros.

Para implementar un ejemplo de la decodificación convolucional se crea un script en Matlab llamado *EjemploDecConv.m*.

Este script inicia por definir la secuencia de bits a decodificar, la secuencia es la misma que se utilizó para el ejemplo anterior desarrollado de forma manual y detallada. Es decir que en *codein* se almacena la secuencia **11101100**, la cual contiene un bit errado respecto a la secuencia **11101000** que es la secuencia codificada sin errores obtenida de **1101**. Luego se crea una estructura de trellis con polinomios generadores $g_1 = 5$ y $g_2 = 7$ y con una memoria igual a 3. La estructura se almacena en *P_Trellis*.

Después se establece el parámetro de la profundidad de rastreo en 4, esto debido a que al codificador ingresaron en un principio 4 bits. Luego se decodifican los bits del vector columna *codein* y el resultado se almacena en el vector *DecoVit*. El parámetro *opmode* se establece en 'trunc', ya que el codificador inicia su estado de codificación en ceros. El parámetro *dectype* se establece en 'hard', ya que se está trabajando con valores binarios. A continuación, se presenta el código del script *EjemploDecConv.m*.

```
% Ejemplo de decodificación convolucional utilizando el algoritmo de
% Viterbi:
codedin=[1;1;1;0;1;1;0;0]; % Datos codificados con un bit errado,
% la secuencia original es 11101000 y se obtiene de 1101
% Crear la estructura de trellis:
m=3; % m es el tamaño de registro de la memoria
g1=5; % Polinomio generador
g2=7; % Polinomio generador
P_Trellis = poly2trellis(m, [g1 g2]); % Estructura de trellis
% Decodificador convolucional:
tbdepth=4; % Produndiad de rastreo, igual a la longitud de los bits
% originales
DecodVit = vitdec(codedin,P_Trellis,tbdepth,'trunc','hard'); % Bits
% decodificados
```

Luego de ejecutar la sección de código anterior se obtiene las variables en el espacio de trabajo que se muestran en la Figura 2.179. Con color rojo se encierran los bits codificados y con un bit errado. Con color verde se encierra los bits recuperados luego de ser decodificados con el diagrama de trellis, mediante el algoritmo de Viterbi. Los bits recuperados coinciden con los bits originales de información.

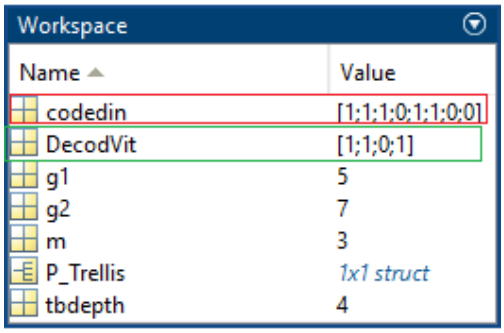


Figura 2.179. Variables en el espacio de trabajo luego de ejecutar el script *EjemploDecConv.m*.

2.5.6.2. Decodificación Convolucional en el archivo de texto recibido

Antes de implementar el Decodificador Convolucional mediante el algoritmo de Viterbi, en los datos recibidos del canal inalámbrico, es necesario cargar y procesar estos datos con los scripts de las etapas previas. Los datos recuperados del canal inalámbrico se almacenaron en el archivo *datosRecibidos150.mat* y se cargan con el comando *load*.

Luego se debe ejecutar el script *recortar_rx.m*, correspondiente a la etapa Recortado de datos. También se deben ejecutar los scripts correspondientes al procesamiento del preámbulo IEE 802.11a que son; *recortar_rx.m*, *encontrarTrama_rx.m*, *corrCFO_rx.m*, *SincSimbolo_rx.m* y *corrFase_rx.m*. Después se ejecuta el script *ofdm_rx.m* que corresponde al procesamiento OFDM. También se ejecuta el script *reformado_rx.m* correspondiente a la etapa de reformado de pulso. Finalmente, se ejecuta el script *demapeo_datos.m* que corresponde al demapeo QPSK.

El vector de entrada para el Decodificador Convolucional es *datosDemod* y se obtiene del script *demapeo_datos.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando.

```
clearvars -except datosDemod % Borrar todas las variables
% excepto la variable datosMapeo (entrada decod_conv.m)
```

La decodificación convolucional del archivo de texto recibido se implementa en un script llamado *decod_conv.m*. El código inicia por calcular el número de bits codificados que se recuperó por cada símbolo OFDM. Para esto, se almacena el número de portadoras de datos del símbolo OFDM (40) en *NPdatos*. En base al tipo de mapeo que se está utilizando

se elige el valor del número de estados en la constelación (*M*), esto se realiza con un condicional *if* y con la variable *seleccionMapeo*, que cuando se utiliza BPSK es 10, QPSK es 20, 16-QAM es 30 y 64-QAM es 40. Luego con el valor de *M* se calcula el número de bits por cada símbolo mapeado y se almacena en *Nb*. Como se está utilizando la técnica de reformado de pulso se almacena el valor de 1/sps en la variable *Nsps*, sps es las muestras que se utiliza el filtro SRRC para representar cada símbolo mapeado y es igual a 4. Entonces el número de bits codificados que se recuperó por cada símbolo OFDM se almacena en *nBc* y se lo obtiene de la multiplicación de *NPdatos* por *Nb* y por *Nsps*.

Cabe aclarar que como se envió los datos con QPSK el valor de *seleccionMapeo* es 20, pero *seleccionMapeo* debe ajustarse según la técnica de modulación digital seleccionada.

Luego se realiza un relleno de ceros al vector de entrada *datosDemod* para que la longitud de estos datos sean múltiplo de *nBc*. En este proceso se calcula la cantidad de grupos de bits de longitud *nBc* que se obtienen de los datos de entrada, en el variable *nGrupos* se almacena el número de grupos que se tiene. En el vector *datosDemodRell* se almacenan los datos de entrada con el relleno de ceros correspondiente.

Después se definen los parámetros que se utilizaron para el codificador convolucional, ya que estos se utilizan para crear la estructura de trellis y para calcular la profundidad de rastreo. La estructura de trellis se almacena en *P_trellis* y se crea con polinomios generadores $g_1 = 5$ y $g_2 = 7$ y con una memoria igual a 3. La profundidad de rastreo tiene que ser igual al número de bits que ingresaron al codificador, este valor se almacena en *tb* y se calcula multiplicando *nBc* por la tasa de codificación del codificador convolucional almacenada en la variable *Rcc*.

Luego se inicializa el vector que va a almacenar los datos decodificados, este vector se llama *datosDecodConv*. Con un lazo *for* se implemente la decodificación convolucional a los grupos de bits codificados, el lazo recorre de 1 hasta el valor que toma *nGrupos*. Dentro del lazo, se copia en una variable auxiliar llamada *auxDecod* el rango de bits correspondiente del vector *datosDemodRell*. Luego se decodifica los datos de *auxDecod* utilizando la función *vitdec* y se almacena en *DecodVit*. Los datos decodificados se almacenan en la variable de salida *datosDecodConv*, tal que a medida que avanza el lazo los bits decodificados se colocan serializados.

Los detalles respecto a la decodificación convolucional en Matlab, utilizando la función *vitdec*, se abordaron en la sección anterior.

El código comentado del script *decod_conv.m* se presenta a continuación, debidamente comentado.

```
% Script que permite aplicar la decodificación convolucional mediante
% el algoritmo de Viterbi para corregir errores a nivel de bit
% Antes de correr este script es necesario cargar los datos guardados
%en datosRecibidos150.mat, ejecutar recortar_rx.m, encontrarTrama_rx.m,
% corrCFO_rx.m, SincSimbolo_rx.m, corrFase_rx.m, ofdm_rx,m,
% reformado_rx.m y demapeo_datos.m
% El vector de entrada es datosDemod y se obtiene de demapeo_datos.m
% Calcular el número de bits codificados por cada símbolo OFDM:
NPdatos=40; % número de portadoras de datos por símbolo OFDM
% El valor en la variable seleccionMapeo define el tipo de demapeo a
% utilizar: BPSK (10), QPSK (20), 16-QAM(30) o 64-QAM(40)
seleccionMapeo=20; % Se selecciona demapeo QPSK, ya que se trabaja con
% este para los ejemplos
% Se selecciona el tipo de mapeo/demapeo con un if
if (seleccionMapeo==10)
    % BPSK
    M = 2; % Puntos en la constelación (símbolos); 2 para BPSK
elseif (seleccionMapeo==20)
    % QPSK
    M = 4; % Puntos en la constelación (símbolos); 4 para QPSK
elseif (seleccionMapeo==30)
    % 16-QAM
    M = 16; % Puntos en la constelación (símbolos); 16 para 16-QAM
elseif (seleccionMapeo==40)
    % 64-QAM
    M = 64; % Puntos en la constelación (símbolos); 64 para 64-QAM
end
Nb=log2(M); % Número de bits utilizado por cada símbolo
Nsps=1/4; % Este valor tiene que ser igual 1/sps, sps se define en
% la etapa reformado de pulso y es igual a 4 cuando está activado
nBc=Nb*NPdatos*Nsps; % Número de bits codificados por simb OFDM
% Relleno de bits para que el vector de entrada sea multiplo de nBc:
nGrupos=ceil(length(datosDemod)/nBc); % Calcular el número de grupos
% de bits a decodificar
longiPadd=nBc*nGrupos-length(datosDemod); % Longitud de relleno
v_relleno=zeros(longiPadd,1); % relleno de ceros
datosDemodRell=[datosDemod;v_relleno]; % Datos serializados con
% relleno de ceros
% Parámetros del codificador convolucional:
nc=2; % n es la longitud por cada ingreso de k bits
kc=1; % k es los bits de entrada
mc=3; % m es el tamaño de registro de la memoria
g1=5; % Polinomio generador
g2=7; % Polinomio generador
Rcc=kc/nc; % tasa del codificador convolucional
% Estructura de Trellis:
P_Trellis = poly2trellis(mc, [g1 g2]); % Igual que en tx
% Decodificación convolucional:
tb=nBc*Rcc; % La profundidad de rastreo debe ser igual al número de
% bits que se tuvo a la entrada del codificador
datosDecodConv=[]; % Inicilizar vector que almacena los datos
% decodificados
% Con un lazo for se recorre los grupos de bits a ser decodificados
for i=1:nGrupos
    auxDecod = datosDemodRell((nBc*(i-1)+1):nBc*i,1); % Se toma el
    % grupo correspondiente
```

```
    DecodVit= vitdec(auxDecod,P_Trellis,tb,'trunc','hard'); % Se
    % decodifica los bits
    datosDecodConv=[datosDecodConv;DecodVit]; % Almacenar los bits
    % decodificados
end
```

Luego de ejecutar la sección de código anterior se obtienen las variables en el espacio de trabajo que se muestran en la Figura 2.180. En la figura se señala con color rojo el vector de entrada, con color amarillor el vector de entrada con relleno y con color verde el vector de salida de la etapa actual. El vector de salida es la mitad del vector de entrada con relleno, esto tiene sentido ya que la tasa de codificación (*Rcc*) es igual a 1/2.

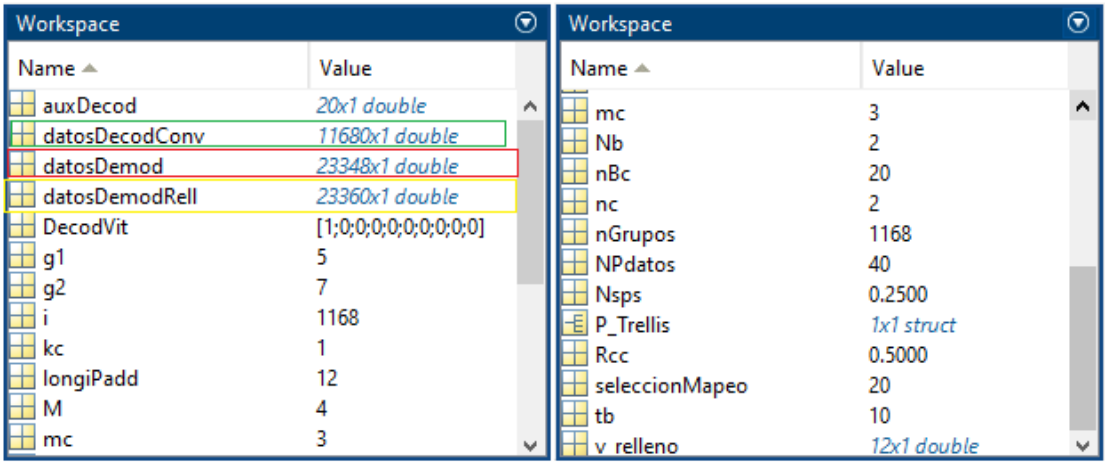


Figura 2.180. Variables en el espacio de trabajo luego de ejecutar el script *decod_conv.m*.

En la Figura 2.181 se muestran los primeros 20 bits del vector de entrada, es decir los datos codificados y alado se coloca los primeros 10 bits decodificados. Estos datos corresponden a los bits recuperados del primer grupo de bits.

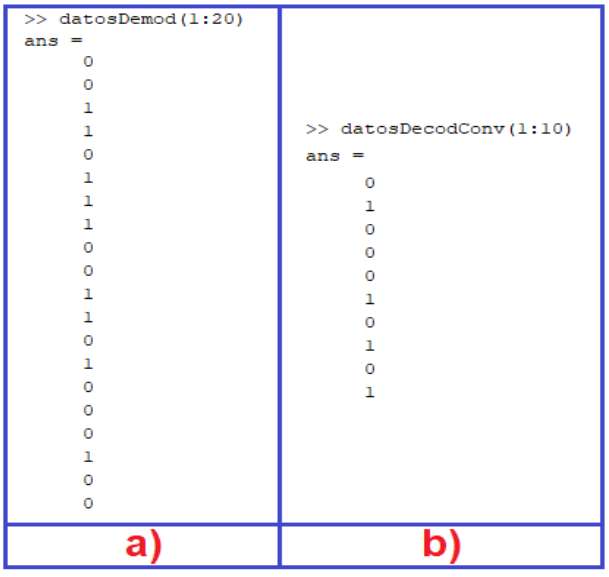


Figura 2.181. a) Primeros bits del vector *datosDemod*. b) Primeros bits del vector *datosDecodConv*.

2.5.7. MEDICIÓN DEL BER Y RECUPERACIÓN DEL TEXTO

En esta sección se abordan las etapas Medir BER y Recuperar texto, estas etapas se identifican con color naranja en la Figura 2.182. La etapa Medir BER consiste en comparar los bits de los datos recibidos con los bits del archivo original y obtener una tasa de bits errados (**BER**). Para seleccionar el archivo de extensión .txt el usuario lo hace mediante un cuadro de diálogo. La etapa Recuperar texto consiste en transformar los bits recuperados del canal inalámbrico en texto y mostrar el texto para que el usuario lo pueda ver.

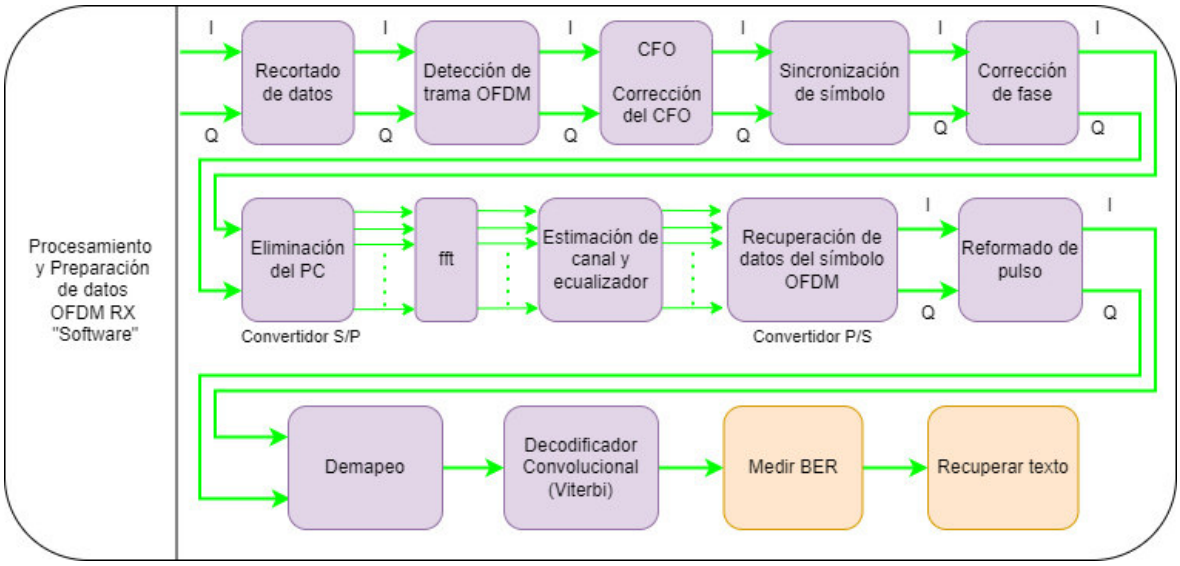


Figura 2.182. Etapas de recepción con las etapas Medir BER y Recuperar texto señalada con naranja.

En esta sección no es necesario abordar fundamentos teóricos debido a que las funciones que se utilizan ya se han descrito previamente.

Para implementar un ejemplo de la obtención de una tasa de bits errados se crea el script *EjemploBER.m*. Este script inicia por definir dos vectores los cuales difiere en 5 bits, el uno del otro, estos bits se señalan con color rojo en la Figura 2.183.

```
datos1=[0;1;0;1;1;0;1;1;1;0;0;1;0;1;0;1;1;0;0;0];
datos2=[1;1;0;1;0;0;1;0;1;0;0;1;0;1;0;1;1;0;1;1];
```

Figura 2.183. Vectores de bits a comparar en el script *EjemploBER.m*.

Luego se inicializa con cero la variable *bitsErrado*, esta almacena la cantidad de bits errados totales luego de comparar cada elemento de los vectores de entrada. Después se recorre mediante un lazo *for* los elementos de los vectores de entrada, tal que dicho lazo determine la posición de los bits a evaluar. Dentro del lazo, se compara con un condicional

if si es que el bit de *datos1* no es igual al bit correspondiente de *datos2*, en caso de no ser igual el valor de *bitsErrado* se incrementa en uno.

Después se muestra los bits errados almacenados en *bitsErrado* sobre la cantidad de bits totales. También se muestra el resultado de la división anterior, es decir el BER. El código de este ejemplo se describe a continuación debidamente comentado.

```
% Ejemplo que permite medir el BER de dos vectores
% Vectores a comparar:
datos1=[0;1;0;1;1;0;1;1;1;0;0;1;0;1;0;1;1;0;0;0];
datos2=[1;1;0;1;0;0;1;0;1;0;0;1;0;1;0;1;1;0;1;1];
bitsErrado=0; % Iniciar contador de bits errados
% Recorrer todo el vector de bits recuperados
for aux=1:length(datos1)
    % Comparar bit recuperado con el bit original
    if(datos1(aux)~=datos2(aux))
        % Se no son iguales se aumenta en uno los bits errados
        bitsErrado=bitsErrado+1;
    end
end
% Mostrar los bits errados sobre los totales y el BER:
fprintf('Bits errados/Bits Totales = %g/%g \n',...
    bitsErrado,length(datos1));
fprintf('BER = %g \n',bitsErrado/length(datos1));
```

Luego de ejecutar el script *EjemploBER.m* se obtiene las variables en el espacio de trabajo que se muestra en la Figura 2.184 y en la ventana de comandos se obtiene los resultados que se muestran en la Figura 2.185. En la variable *bitsErrado* se encuentra almacenado el valor de 5, este valor concuerda con la cantidad de errores introducidos inicialmente en el código.

Figura 2.184. Variables en el espacio de trabajo luego de ejecutar *EjemploBER.m*.

En la Figura 2.185 se muestra a modo de fracción la cantidad de bits errados sobre los bits totales. También se muestra la tasa de bits errados como número decimal. En base a los resultados obtenidos, se puede afirmar que el código para medir la cantidad de bits errados funciona correctamente.

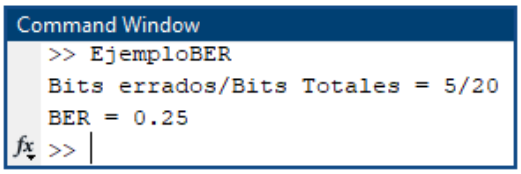


Figura 2.185. Resultado en la ventana de comandos luego de ejecutar *EjemploBER.m*.

En las siguientes subsecciones se describe el código para implementar las etapas Medir BER y Recuperar texto en los datos recuperados del canal inalámbrico.

2.5.7.1. Medición del BER en el archivo de texto recibido

Previo a implementar la etapa Medir BER en los datos recuperados del canal inalámbrico, es necesario cargar y procesar los datos con los scripts de las etapas previas. Los datos recuperados del canal inalámbrico se almacenaron en el archivo *datosRecibidos150.mat* y se cargan con el comando *load*. Luego se debe ejecutar el script *recortar_rx.m*, correspondiente a la etapa Recortado de datos.

Después se deben ejecutar los scripts correspondientes al procesamiento del preámbulo IEE 802.11a que son; *recortar_rx.m*, *encontrarTrama_rx.m*, *corrCFO_rx.m*, *SincSimbolo_rx.m* y *corrFase_rx.m*. Luego se ejecuta el script *ofdm_rx.m* que corresponde al procesamiento OFDM. También se ejecuta el script *reformado_rx.m* correspondiente a la etapa de reformado de pulso. Después se ejecuta el script *demapeo_datos.m* que corresponde al demapeo QPSK. Finalmente, se ejecuta el script *decod_conv.m* correspondiente a la etapa Decodificador Convolucional.

El vector de entrada para la etapa Medir BER es *datosDemod* y se obtiene del script *decod_conv.m*. Para limpiar todas las variables del espacio de trabajo, excepto la variable de entrada se utiliza el siguiente comando.

```
clearvars -except datosDecodConv % Borrar todas las variables
% excepto la variable datosDecodConv (entrada medirBER_rx.m)
```

La etapa Medir BER se implementa en un script llamado *medirBER_rx.m*. Este código inicia por seleccionar y abrir un archivo de extensión .txt mediante un cuadro de diálogo, esto para que el usuario pueda escoger el archivo original con el cual se compara los bits del archivo recibido. En el capítulo de transmisión se abordó a detalle la manera en que se puede seleccionar y abrir un archivo de texto mediante un cuadro de diálogo. Los bits serializados del archivo original se almacenan en el vector *datosMedirBER* y la longitud del mismo se almacena en el vector *longDatosBER*.

Luego para obtener la cantidad de bits errados, se inicia guardando la longitud del vector *datosDecodConv* en *LongDatDecod*, El vector *datosDecodConv* contiene los bits recuperados y serializados del canal inalámbrico. Con un condicional *if* se evalúa si la longitud de los bits del archivo de texto original es menor o igual a la longitud de los bits recuperados. Cuando no se cumple la condición, se muestra un mensaje informando lo propio y los datos de entrada se copian en la variable de salida *datosSerializadosRx*. Cuando la condición del *if* se cumple se recortan los datos de entrada, tal que estos queden de la misma longitud de *longDatosBER*.

Este recorte se lleva a cabo de esta forma ya que no se agregó ningún protocolo para recuperar la longitud de los bits originales. Si se realizara el recorte en base a algún protocolo puede darse el caso que en el prototipo en tiempo real exista errores y no se pueda recortar correctamente, En un prototipo en tiempo real lo que se busca es recuperar los datos lo más rápido posible, el uso de un protocolo puede aumentar el tiempo de procesamiento.

Los datos recortados se almacenan en la variable de salida *datosSerializadosRx*. Después se inicializa en cero la variable *bitsErrado* que almacena la cantidad de bits errados. Seguido con un lazo *for* se recorre todas las posiciones de los vectores a comparar para medir los bits errados. Dentro del *for* con un condicional *if* se evalúa si el bit correspondiente del vector *datosSerializadosRx* es igual al bit del vector *datosMedirBER*. Cuando no son iguales significa que hay un bit errado y la variable *bitsErrado* aumenta en uno.

Cuando termina el lazo *for*, se calcula la tasa de bits errados (BER) y se almacena en la variable *v_BER*. Luego se muestra en la ventana de comandos la cantidad de bits errados sobre los bits totales y también la tasa de bits errados. El código del script *medirBER_rx.m* se presenta a continuación debidamente comentado.

```
% Script que permite medir el BER del archivo de texto recuperado
% Antes de correr este script es necesario cargar los datos guardados
%en datosRecibidos150.mat, ejecutar recortar_rx.m, encontrarTrama_rx.m,
% corrCFO_rx.m, SincSimbolo_rx.m, corrFase_rx.m, ofdm_rx,m,
% reformado_rx.m, demapeo_datos.m y decod_conv.m
% El vector de entrada es datosDecodConv y se obtiene de decod_conv.m
% %%%%%%%%%%%%%%%%%%%%%% ABRIR ARCHIVO: %%%%%%%%%%%%%%%%%%%%%%
clearvars textoBitsSerial datosMedirBER
nombreArchivo=uigetfile({'*.txt'},'Seleccionar archivo'); %Seleccionar
% un archivo mediante un cuadra de diálogo
fid =fopen(nombreArchivo); % Abrir el archivo seleccionado
x=fread(fid,'*char'); % Leer y guardar el archivo
fclose(fid); % Cerrar el archivo
textoBinario = dec2bin(x,8); % Representar cada caracter con 8 bits
% Serializar datos del vector x:
aux=1; % Iniciar variable auxiliar
for i = 1:length(textoBinario)
```

```
    for j=1:8
        textoBitsSerial(aux)=textoBinario(i,j);
        aux=aux+1;
    end
end
% Pasar del tipo char al tipo double;
for aux=1:length(textoBitsSerial)
    datosMedirBER(aux,1)=str2double(textoBitsSerial(aux)); % Datos
    % originales serializados
end
longDatosBER=length(datosMedirBER); % Longitud de bits originales
% %%%%%%%%%%%%%%%%%%%%%% MEDIR BER: %%%%%%%%%%%%%%%%%%%%%%%%%
LongDatDecod=length(datosDecodConv); % longitud de bits recuperados
if (longDatosBER<=LongDatDecod)
    % La longitud del archivo recuperado es mayor o igual que la
    % del original
    datosSerializadosRx=datosDecodConv(1:longDatosBER); % Recortar
    bitsErrado=0; % Iniciar contador de bits errados
    % Recorrer todo el vector de bits recuperados
    for aux=1:longDatosBER
        % Comparar bit recuperado con el bit original
        if(datosSerializadosRx(aux)~=datosMedirBER(aux))
            % Se no son iguales se aumenta en uno los bits errados
            bitsErrado=bitsErrado+1;
        end
    end
    % Mostrar la cantidad de bits errados sobre la longitud total:
    v_BER=bitsErrado/longDatosBER; % Valor de BER
    fprintf('Bits errados/Bits totales = %g/%g \n',...
        bitsErrado,longDatosBER);
    fprintf('BER = %g \n',v_BER);
else
    % La longitud del archivo recuperado es menor que la del original
    disp('Archivo no recuperado completamente, BER no medido')
    disp('Longitud del archivo recuperdado menor al original');
    datosSerializadosRx=datosDecodConv; % Copiar los datos
end
```

Cuando se ejecuta el script *medirBER_rx.m* se abre un cuadro de diálogo para seleccionar el archivo de texto original. En la Figura 2.186 se muestra el cuadro de diálogo con el archivo *ArchivoTextoRT.txt* seleccionado. Este archivo es el mismo que se escogió en transmisión para procesarlo y enviarlo a través del canal inalámbrico.

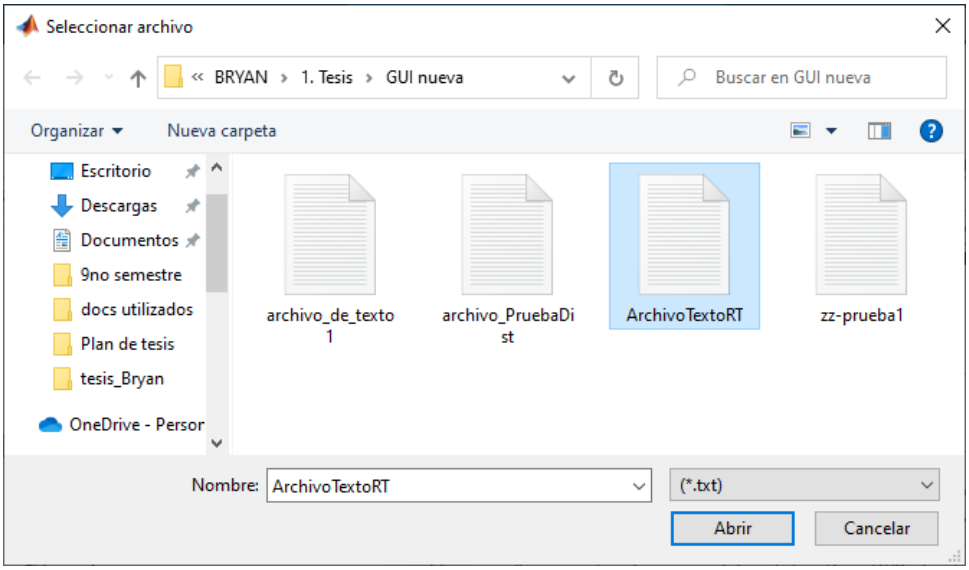


Figura 2.186. Cuadro de diálogo para seleccionar un archivo de texto.

Luego de seleccionar el archivo se crean las variables en el espacio de trabajo que se muestra en la Figura 2.187 y el resultado en la ventana de comandos que se muestra en la Figura 2.188. Dentro de las variables en el espacio de trabajo se identifica con color rojo el vector de entrada y con color verde el vector de salida. El vector de salida tiene una longitud menor que el vector de entrada.

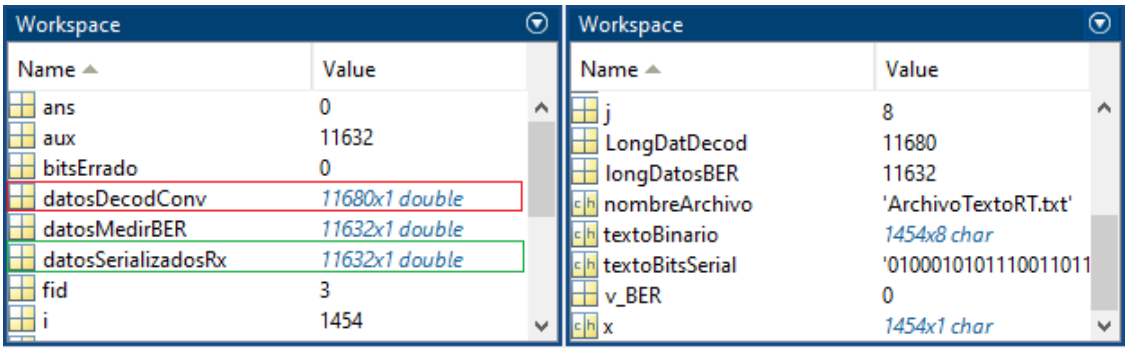


Figura 2.187. Variables en el espacio de trabajo luego de ejecutar el script *medirBER_rx.m*.

En la Figura 2.188 se observa que, para los datos recuperados del canal inalámbrico, no existe ningún bit errado. Esto se debe a que la distancia entre los equipos SDR fue corta y también a que se utilizó una técnica de mapeo que no tiene muchos puntos en las costelaciones. También se debe a que se utilizó la técnica de reformado de pulso y de corrección de errores a nivel de bit.

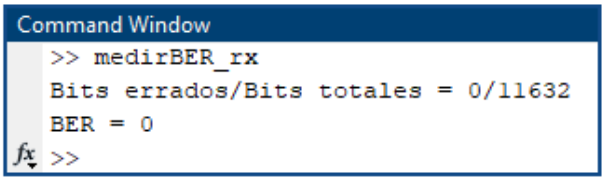


Figura 2.188. Resultado en la ventaja de comandos al ejecutar *medirBER_rx.m*.

En el capítulo de pruebas de funcionamiento se va a correr el prototipo con distintas configuraciones lo que permitirá observar que sucede con el valor del BER.

2.5.7.2. Recuperación del texto del archivo recibido

Luego de ejecutar el script *medirBER_rx.m* correspondiente a la etapa Medir BER se puede seguir con la etapa Recuperar texto. Esta etapa se implementa en un script llamado *recuperar_texto.m*. Previo a ejecutar el script se limpia las variables del espacio de trabajo a excepción de la variable *datosSerializadosRx*, que contiene los bits de información recuperados del canal inalámbrico y que sirve como entrada a la etapa Recuperar texto. El comando que se utiliza para limpiar las variables es el que se presenta a continuación.

```
clearvars -except datosSerializadosRx % Borrar todas las variables
% excepto la variable datosSerializadosRx (entrada recuperar_texto.m)
```

El código del script *recuperar_texto.m* inicia por realizar un relleno de ceros al vector de entrada para que este sea múltiplo de 8, ya que se utiliza 8 bits para representar cada caracter de texto. Este relleno se realiza para asegurar que independientemente que se haya o no medido el BER se pueda mostrar los datos recibidos. Los datos con relleno se almacenan en la variable *datosSerializados*. Cuando no se realiza un relleno de ceros el vector de entrada *datosSerializadosRx* es el mismo que el vector *datosSerializados*.

Luego con la función *reshape* se obtiene una matriz llamada *datosMtz*, la cual contiene la representación binaria de cada caracter en cada fila y se obtiene a partir del vector de bits serializados llamado *datosSerializados*. Después se transforma cada fila de datos binarios a su representación decimal, esto se realiza con un lazo *for* y dentro del lazo se utiliza la función *bi2de*. Finalmente, se transforma los símbolos decimales a caracteres mediante la función *char* y se almacena en el vector *textoRecuperado*. Al realizar esta última operación se obvia el punto y coma (;) al final de la línea de código, para que se imprima los caracteres en la ventana de comandos.

Las funciones *rechape*, *bi2de* y *char* se abordaron en el capítulo de transmisión, por lo que en este apartado solo nos limitamos a mencionarlas.

El código debidamente comando de script *recuperar_texto.m* se presenta a continuación debidamente comentado.

```
% Script que permite Recuperar los caracteres de Texto
% Antes de correr este script es necesario cargar los datos guardados
%en datosRecibidos150.mat, ejecutar recortar_rx.m, encontrarTrama_rx.m,
% corrCFO_rx.m, SincSimbolo_rx.m, corrFase_rx.m, ofdm_rx,m,
% reformado_rx.m, demapeo_datos.m, decod_conv.m y medirBER_rx.m
```

```
% El vector de entrada es datosSerializadosRx
% Relleno de ceros:
bPc=8; % Bits por cada caracter
numC=ceil(length(datosSerializadosRx)/8); % Número caracteres
LongRellnumC=bPc*numC-length(datosSerializadosRx); % Longitud de
% relleno para hacer múltiplo de 8
rellenoC=zeros(LongRellnumC,1); % Vector de relleno ceros
datosSerializados=[datosSerializadosRx ; rellenoC]; % Rellenar en
% caso de ser necesario
% Pasar de los bits serializados a una matriz, en donde las filas
% contiene a los caracteres en binario:
datosMtz=reshape(datosSerializados,[8 numC]).'; % numC filas x 8
columnas
% Pasar de binario a decimal, los datos de cada fila:
datosTexto=[]; % Inicializar vector
for i=1:numC
    % Se pasa cada fila a decimal y se almacena en datosTexto
    datosTexto(1,i)=double(bi2de(datosMtz(i,:),'left-msb'));
end
% Pasar de decimal a caracteres:
textoRecuperado=char(datosTexto) % Vector de salida, contiene los
% caracteres de texto recuperados
```

Luego de ejecutar la sección de código correspondiente al script *recuperar_texto.m* en el espacio de trabajo se obtienen las variables que se muestran en la Figura 2.189. También se obtiene en la ventana de comandos lo que se muestra en la Figura 2.190.

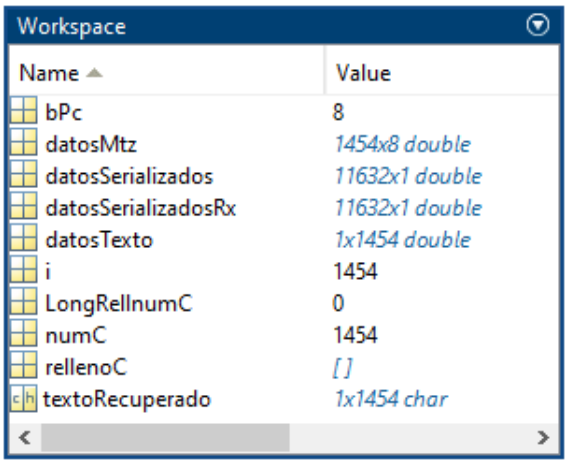


Figura 2.189. Variables en el espacio de trabajo luego de ejecutar el script *recuperar_texto.m*.

En la Figura 2.190 se muestra el texto recuperado e impreso en la ventana de comandos de Matlab. Este texto es el mismo que se encuentra en el archivo *ArchivoTextoRT.txt*, debido a que no existen bits errados en la información recuperada.

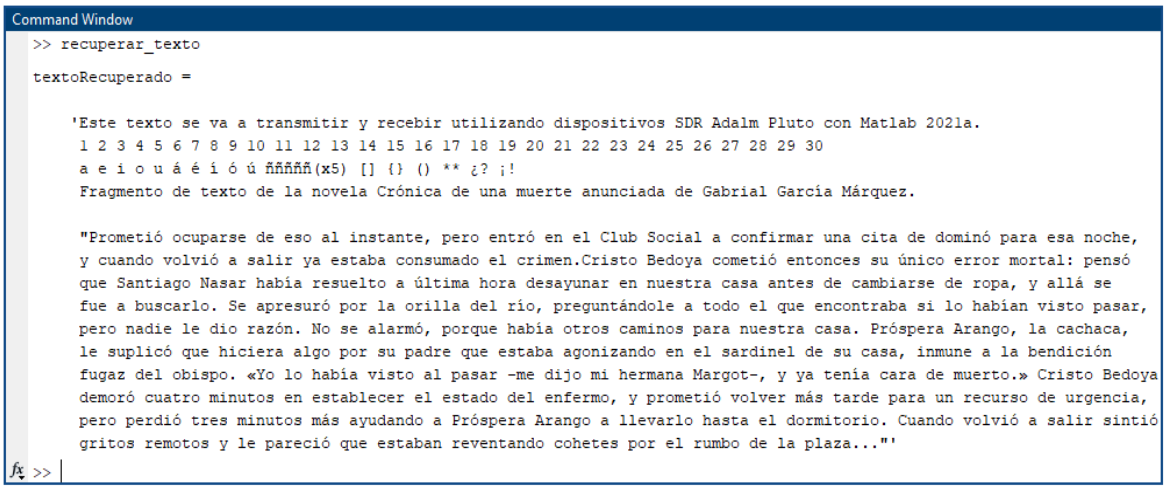


Figura 2.190. Resultado en la ventana de comandos luego de ejecutar el script *recuperar_texto.m*.

Para tener un punto de comparación en el siguiente cuadro se muestra el texto que contiene el archivo *ArchivoTextoRT.m*.

Este texto se va a transmitir y recebir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
Fragmento de texto de la novela Crónica de una muerte anunciada de Gabrial García Márquez.

"Prometió ocuparse de eso al instante, pero entró en el Club Social a confirmar una cita de dominó para esa noche,
y cuando volvió a salir ya estaba consumado el crimen.Cristo Bedoya cometió entonces su único error mortal: pensó
que Santiago Nasar había resuelto a última hora desayunar en nuestra casa antes de cambiarse de ropa, y allá se
fue a buscarlo. Se apresuró por la orilla del río, preguntándole a todo el que encontraba si lo habían visto pasar,
pero nadie le dio razón. No se alarmó, porque había otros caminos para nuestra casa. Próspera Arango, la cachaca,
le suplicó que hiciera algo por su padre que estaba agonizando en el sardinel de su casa, inmune a la bendición
fugaz del obispo. «Yo lo había visto al pasar -me dijo mi hermana Margot-, y ya tenía cara de muerto.» Cristo Bedoya
demoró cuatro minutos en establecer el estado del enfermo, y prometió volver más tarde para un recurso de urgencia,
pero perdió tres minutos más ayudando a Próspera Arango a llevarlo hasta el dormitorio. Cuando volvió a salir sintió
gritos remotos y le pareció que estaban reventando cohetes por el rumbo de la plaza..."

2.6. INTERFACES GRÁFICAS DEL PROTOTIPO

El prototipo de comunicación inalámbrica utiliza dos computadores con Matlab R2021a y dos equipos SDR Adalm Pluto, uno se utiliza para la transmisión y otro para la recepción. El prototipo consta en total de cuatro interfaces gráficas, una interfaz gráfica principal, una de configuración, una de transmisión y una de recepción. La interfaz gráfica principal permite configurar automáticamente los objetos de transmisión y de recepción que permiten conectar Matlab con el dispositivo SDR.

La interfaz gráfica principal permite acceder a las interfaces gráficas de; configuración, transmisión y recepción. Entonces, independientemente de que se trate del transmisor o receptor se tiene que ejecutar primero esta interfaz gráfica principal.

La interfaz gráfica de configuración permite variar algunos parámetros de los objetos de transmisión y de recepción. Cuando en un computador se varían los parámetros de

configuración, en el otro computador también se deben cambiar los parámetros para que coincidan. Si la configuración en los computadores difiere, no se podrá realizar la transmisión y recepción de datos.

La interfaz gráfica de **transmisión** permite seleccionar, procesar y transmitir recursivamente los datos al canal inalámbrico. Los datos se transmiten utilizando un equipo SDR Adalm Pluto. El transmitir recursivamente los datos al canal inalámbrico, permite que, al presionar el botón de transmisión, el usuario pueda ubicar el transmisor en un lugar fijo y concentrarse en los resultados que se obtiene en recepción.

La interfaz gráfica de **recepción** permite recuperar los datos del canal inalámbrico utilizando un equipo SDR Adalm Pluto. Cuando se recuperan los datos del canal, inmediatamente se realiza el procesamiento de los datos recibidos. Durante el procesamiento se grafican los valores de estimación del desplazamiento de frecuencia, el diagrama de constelación antes de demapear los datos, el valor de la tasa de bits errados (BER) y los caracteres de texto recuperados. La tasa de bits errados se obtiene en base al archivo de texto original y al recuperado. El proceso de recepción y procesamiento se realiza de manera recursiva, para que el usuario pueda mover el receptor o modificar el escenario de pruebas y obtener los resultados en tiempo real.

Para referirnos a una interfaz gráfica *interfaz.fig* asociada a *interfaz.fig*, utilizaremos la identificación de *interfaz()*, es decir se reemplaza las extensiones de los archivos por paréntesis.

2.6.1. INTERFAZ GRÁFICA PRINCIPAL

En esta sección se va a describir la interfaz gráfica principal del prototipo de comunicación inalámbrica. Dicha interfaz detecta si se encuentra algún dispositivo SDR Adalm Pluto conectado por una interfaz USB 2.0 y de ser el caso configura el mismo cuando se abre la interfaz gráfica. En caso de que no se haya conectado algún equipo SDR previo a abrir la interfaz gráfica principal, se brinda la opción de configurar el dispositivo con un botón, cuando ya se ha conectado un equipo Adalm Pluto. Una vez se ha configurado el dispositivo SDR Adalm Pluto, el usuario puede acceder a las etapas de transmisión y recepción en tiempo real.

La interfaz gráfica principal se llama *main.fig* y está asociada a *main.m*. En la Figura 2.191 se muestra la interfaz gráfica *main.fig*, la cual se construyó con “GUIDE” de Matlab. En dicha figura se utilizan **indicadores** para cada uno de los componentes de la interfaz

gráfica. En la Tabla 2.17 se presentan los componentes principales de la interfaz gráfica *main.fig* con su respectivo indicador, tag y string. El tag corresponde a la asignación de nombre dentro de *main.m* y el string corresponde al texto a mostrar en el componente de *main.fig*.

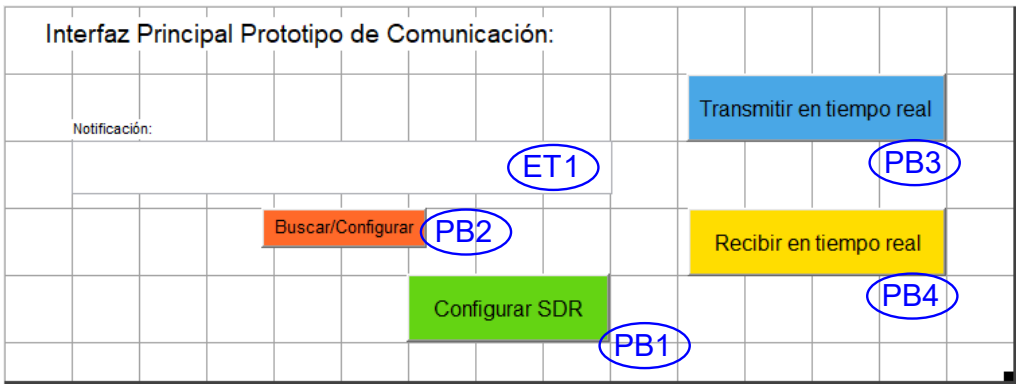


Figura 2.191. Interfaz gráfica principal *main.fig*.

Explicación breve sobre los componentes de la interfaz *main.fig*:

- El botón **Configurar SDR** abre una interfaz gráfica que permite variar los parámetros de configuración del dispositivo Adalm Pluto.
- El botón **Buscar/Configurar** permite configurar los objetos de transmisión y recepción.
- El botón **Transmitir en tiempo real** abre la interfaz gráfica de transmisión.
- El botón **Recibir en tiempo real** abre una interfaz gráfica recepción.
- El *Edit Text* **ET1** sirve para mostrar mensajes de información sobre la configuración.

Tabla 2.17. Nombres de los componentes de *main.fig*

Indicador	Componente	Tag	String
ET1	Edit Text	txtNotificacion	
PB1	Push Button	ConfigurarSDR	Configurar SDR
PB2	Push Button	btnBuscarConfig	Buscar/Configurar
PB3	Push Button	btnTx	Transmitir en tiempo real
PB4	Push Button	btnRx	Recibir en tiempo real

2.6.1.1. Funcionamiento de la interfaz gráfica principal

En esta sección se explica lo que sucede cuando se ejecuta la interfaz *main()* y lo que ocurre cuando se presiona cada botón de la misma. Además, se muestran los resultados que se obtienen al utilizar la interfaz gráfica principal.

Cuando se desea que una sección de código se ejecute apenas corre la aplicación, hay que programarlo en opening function o función de apertura de la interfaz gráfica. Es decir

que cuando la interfaz gráfica *main.fig* asociada a *main.m* se ejecuta, opening function también lo hará.

2.6.1.1.1. Función de apertura de la interfaz gráfica principal

Cuando se ejecuta *main.m* se discrimina si existe al menos un dispositivo equipo SDR conectado al computador mediante una interfaz USB. En caso de que no se encuentre ningún dispositivo, se muestra un mensaje informativo en **ET1**, el mensaje es; *"Dispositivo no encontrado y no configurado. Presione Buscar/Configurar"*. En Figura 2.192 se muestra la interfaz gráfica principal ejecutada y sin ningún equipo SDR Adalm Pluto conectado al computador. Además, en la ventana de comandos de Matlab se muestra el mensaje; "*Dispositivo no encontrado*".

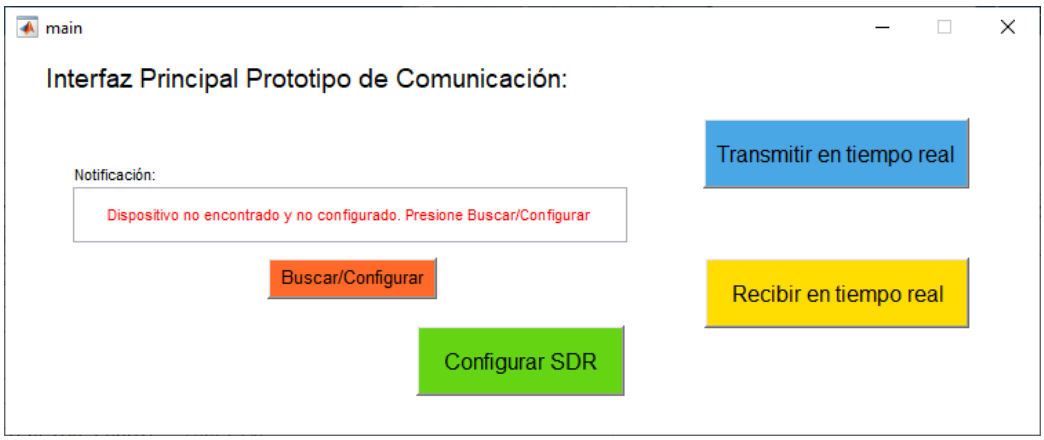


Figura 2.192. Interfaz gráfica principal ejecutada y sin un equipo SDR conectado.

En caso de que se detecte al menos un dispositivo SDR Adalm Pluto, se configuran los objetos de transmisión con los parámetros que se establecieron en la Tabla 2.2 y 2.3. El objeto de transmisión se llama *txAdalm* y el objeto de recepción se llama *rxAdalm*. En la ventana de comandos de Matlab se muestra algunas propiedades de estos objetos (Figura 2.193).

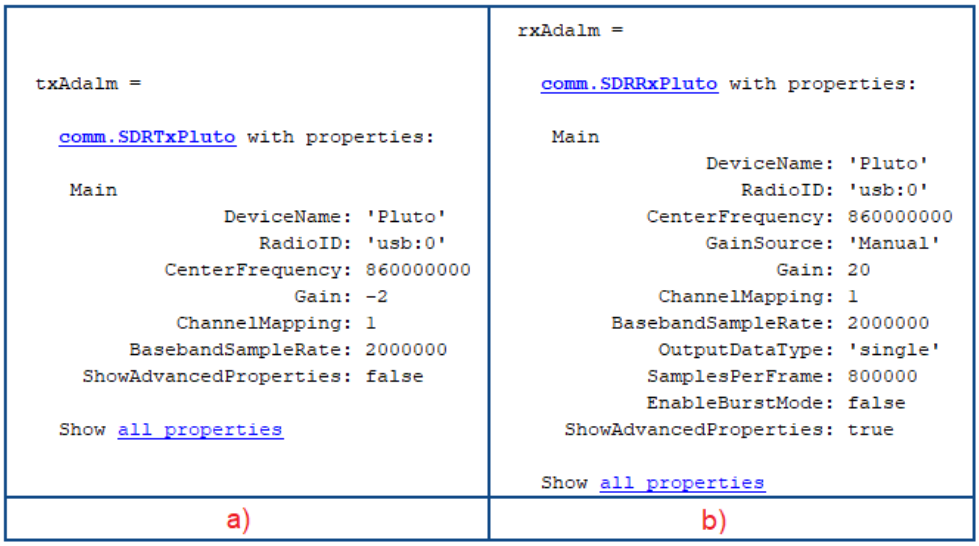


Figura 2.193. a) Información del objeto de transmisión. b) Información del objeto de recepción.

Además, en el elemento *ET1* se muestra el mensaje; *"SDR conectado y configurado inicialmente."*. También se desactiva el botón **Buscar/Configurar**, con el fin de evitar que el usuario vuelva a configurar el dispositivo sin que sea necesario. En la ventana de comandos de Matlab se muestra el mensaje; "*Dispositivo configurado*". En la Figura 2.194 se muestra la interfaz gráfica principal cuando se ha detectado un equipo SDR y se han configurado los objetos de transmisión y recepción.

Figura 2.194. Interfaz gráfica principal ejecutada, con un equipo SDR conectado.

2.6.1.1.2. Botón Buscar/Configurar

Este botón se utiliza cuando se conecta un equipo SDR Adalm Pluto luego de ejecutar la interfaz gráfica *main.fig*, es decir cuando en un inicio no se detectó un equipo SDR. El botón actual muestra el mensaje *"Buscando dispositivo..."* en la ventana de comandos, para informar al usuario que se está buscando dispositivos conectados al computador. Luego discrimina si existe al menos un dispositivo equipo SDR conectado. Cuando no se encuentra ningún dispositivo, se muestra un mensaje informativo en **ET1**, el mensaje es; *"Dispositivo no encontrado y no configurado. Presione Buscar/Configurar"*.

En caso de que se encuentre al menos un dispositivo Adalm Pluto conectado, se configuran los objetos de transmisión con los parámetros que se establecieron en la Tabla 2.2 y 2.3. En la ventana de comandos se muestra algunas propiedades de estos objetos y en **ET1** se muestra el mensaje; *"SDR conectado y configurado inicialmente."*. También se desactiva el botón actual, con el fin de evitar que el usuario vuelva a configurar el dispositivo sin que sea necesario. En la ventana de comandos de Matlab se muestra el mensaje; "*Dispositivo configurado*".

2.6.1.1.3. Botón Configurar SDR

Este botón permite abrir una interfaz gráfica para variar los parámetros de configuración de los objetos de transmisión y recepción. Si se ha detectado previamente que está conectado un equipo SDR Adalm Pluto, al presionar este botón se ejecuta la interfaz gráfica *configurarSDR.fig*, asociada a *configurarSDR.m*. En caso de que no se haya encontrado un dispositivo SDR previamente, se muestra un mensaje de error, informando que no se ha conectado ningún dispositivo (Figura 2.195).

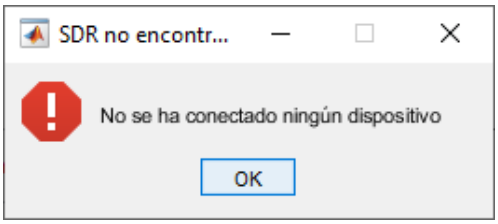


Figura 2.195. Mensaje de error.

2.6.1.1.4. Botón Transmitir en tiempo real

Este botón permite ejecutar la interfaz gráfica de transmisión llamada *txRealTime*, cuando previamente se ha configurado el dispositivo SDR Adalm Pluto. En caso de que no se haya configurado un equipo SDR se muestra un mensaje de error igual al que se mostró en la Figura 2.195.

2.6.1.1.5. Botón Recibir en tiempo real

Al presionar este botón se ejecuta la interfaz gráfica de recepción llamada *rxRealtime*, cuando previamente se ha configurado el equipo Adalm Pluto. En caso de que no se haya configurado el SDR se muestra un mensaje de error igual al que se mostró en la Figura 2.195.

2.6.2. INTERFAZ GRÁFICA DE CONFIGURACIÓN

Esta interfaz gráfica permite variar algunos parámetros de configuración de los objetos de transmisión y recepción. Los parámetros que se pueden variar en la interfaz gráfica son; la interfaz USB a la que se está configurando, el tamaño de bloque que se envía y que se recibe, la ganancia de transmisión y la ganancia de recepción. El nombre de la interfaz gráfica de configuración es *configurarSDR.fig* y esta interfaz se muestra en la Figura 2.196, junto con los elementos más importantes identificados.

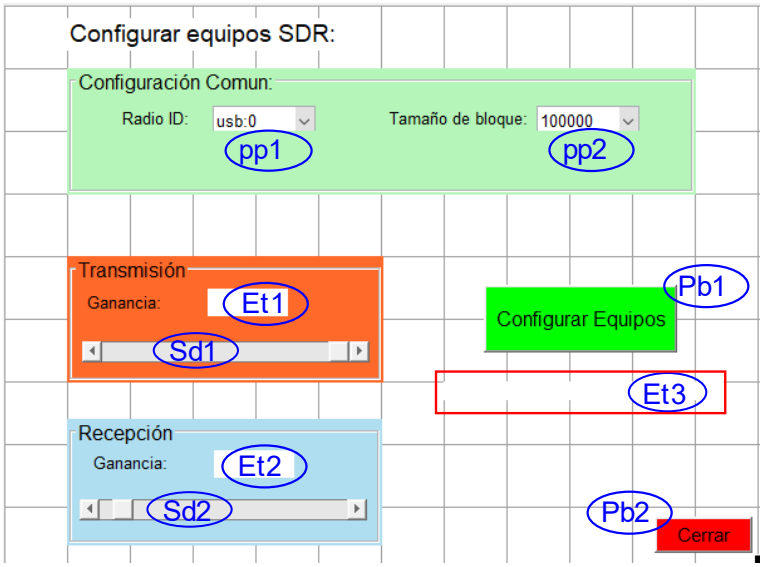


Figura 2.196. Interfaz gráfica de configuración *configurarSDR.fig*.

En la Tabla 2.18 se muestran en detalle los elementos de la interfaz gráfica de configuración. El tag corresponde a la asignación de nombre dentro de *configurarSDR.m* y el string es el texto a mostrar en el componente de *configurarSDR.fig*.

Tabla 2.18. Nombres de los componentes de *main.fig*

Indicador	Componente	Tag	String
Et1	Statit Text	txtGainTx	
Et2	Statit Text	txtGainRx	
Et3	Statit Text	txtNotificacion	
pp1	Pop-up Menu	popupRadio	usb:0 usb:1
pp2	Pop-up Menu	popupBloque	100000 200000 400000 800000 1200000
Pb1	Push Button	btnConfigurar	Configurar Equipos
Pb2	Push Button	btnCerrar	Cerrar
Sd1	Slider	sliderGTx	
Sd2	Slider	sliderGRx	

A continuación, se explica brevemente los componentes de la interfaz gráfica de configuración:

- El menú desplegable **pp1** permite seleccionar la identificación de la interfaz USB de la radio Adalm Pluto.
- El menú desplegable **pp2** permite seleccionar el tamaño de bloque de transmisión y recepción.
- El slider **Sd1** sirve para variar la ganancia de transmisión y mostrarlo en **Et1**.
- El slider **Sd2** sirve para variar la ganancia de recepción y mostrarlo en **Et2**.
- El botón **Configurar Equipos** permite aplicar la configuración a los objetos de transmisión y recepción, en base a los parámetros seleccionados en la interfaz. Cuando se configura **Et3** se muestra un mensaje informativo.
- El botón **Cerrar** permite cerrar la interfaz gráfica.

2.6.2.1. Funcionamiento de la interfaz gráfica de configuración

En esta sección se explica lo que sucede cuando se ejecuta *configurarSDR()*, recordando que esta interfaz se abre o se ejecuta a través de la interfaz gráfica principal. Se describe lo que ocurre cuando se utilizan los sliders, botones y menús desplegables de *configurarSDR()*. Además, se muestran los resultados que se obtienen al utilizar la interfaz gráfica actual.

2.6.2.1.1. Función de apertura de la interfaz gráfica de configuración

Cuando se ejecuta *configurarSDR()*, se establecen los parámetros de configuración que se han establecido previamente. En **pp1** se coloca la identificación de la radio, en **pp2** se coloca el tamaño del bloque, en **Et1** el valor de la ganancia de transmisión y en **Et2** el valor de la ganancia de recepción. En la Figura 2.197 se muestra la interfaz gráfica de configuración ejecutada a través de la interfaz gráfica principal. Los valores de los parámetros de configuración se mantienen en los configurados previamente.

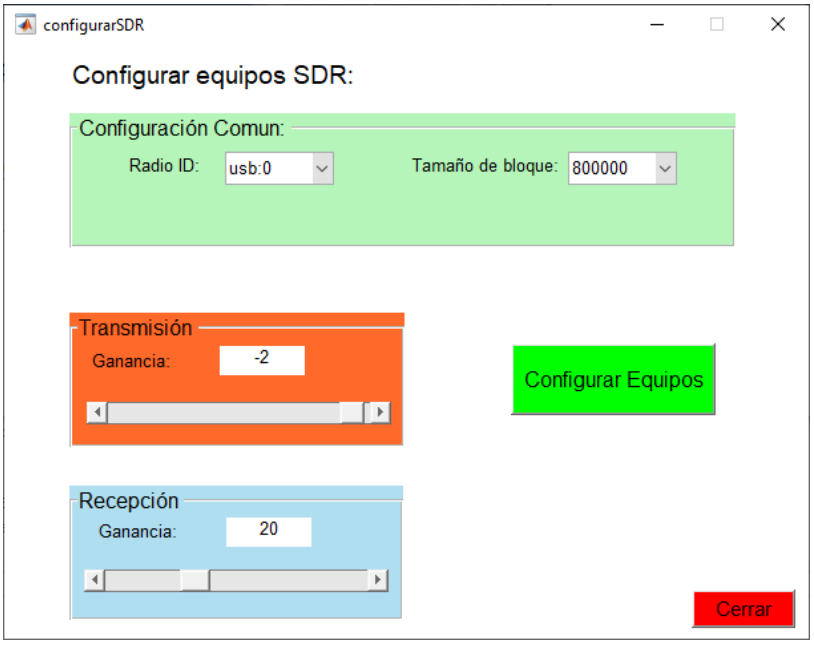


Figura 2.197. Interfaz gráfica de configuración ejecutada.

2.6.2.1.2. Configuración común para los objetos de transmisión y recepción

Los parámetros que se pueden variar en la configuración común es la identificación de la radio y el tamaño del bloque. Esto se selecciona mediante menús desplegables. En la Figura 2.198 se muestran los menús **pp1** y **pp2** desplegados y encerrados con color rojo y azul respectivamente. La identificación de la radio puede cambiarse a *usb:1* cuando se tenga dos equipos SDR conectados al computador.

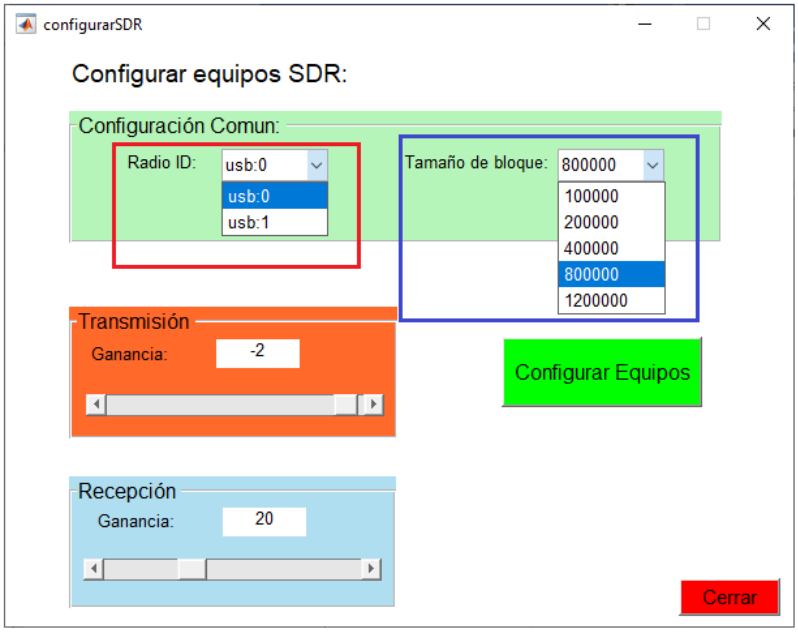


Figura 2.198. Interfaz gráfica de configuración con los *Pop-up Menu* desplegados.

2.6.2.1.3. Ganancia de transmisión y de recepción

La ganancia de transmisión se selecciona en el slider **Sd1** y se muestra su valor en **Et1**. La ganancia de recepción se selecciona en el slider **Sd2** y su valor se muestra en **Et2**. Mediante las flechas de los sliders se puede variar los valores en pasos de uno. En la Figura 2.199 se muestran las ganancias de transmisión y recepción, con diferentes valores a los por defecto.

2.6.2.1.4. Botón Configurar Equipos

Este botón permite configurar los objetos de transmisión y recepción con los valores seleccionados en la interfaz gráfica. Los parámetros que no se muestran en la interfaz se mantienen igual a los que se especificó antes en la interfaz gráfica principal. Al presionar este botón en la ventana de comandos de Matlab se muestra las características de los objetos configurados. Además, en **Et3** se muestra un mensaje informando que se ha configurado el equipo SDR. En la Figura 2.199 se muestra la interfaz gráfica *configurarSDR()*, junto con los parámetros de configuración seleccionados.

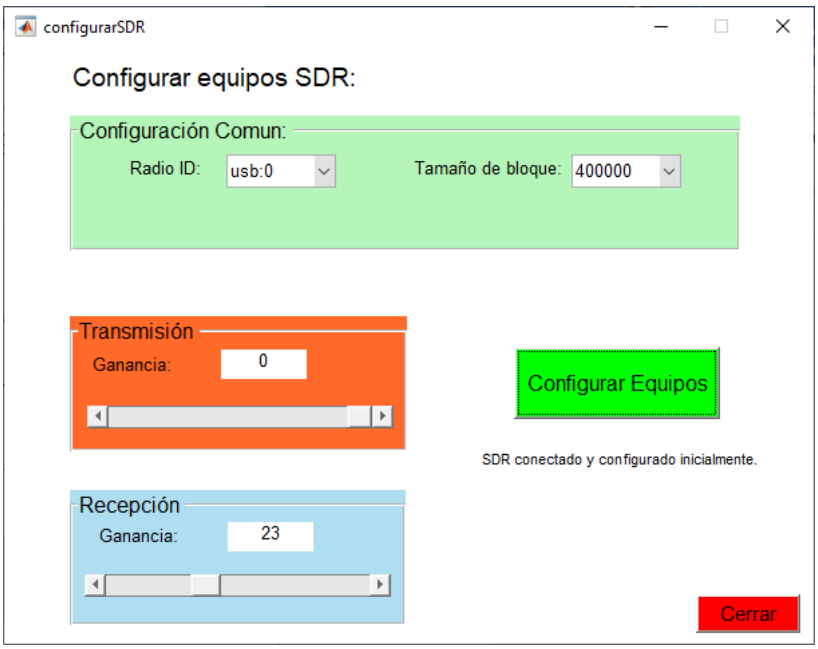


Figura 2.199. Interfaz gráfica de configuración *configurarSDR()* con los valores por defecto cambiados.

En la Figura 2.200 se muestra lo que se obtiene en la ventana de comandos luego de presionar el botón **Configurar Equipos**. El objeto del sistema *txAdalm* es el objeto de transmisión y el objeto *rxAdalm* es el objeto de recepción.

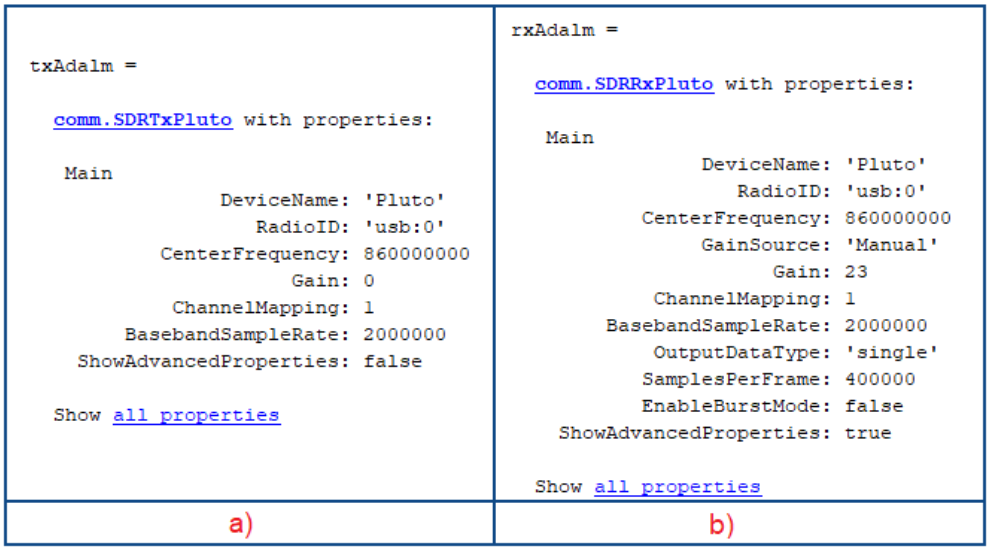


Figura 2.200. a) Información del objeto de transmisión. b) Información del objeto de recepción.

2.6.3. INTERFAZ GRÁFICA DE TRANSMISIÓN

En esta sección se va a describir la interfaz gráfica de transmisión en tiempo real, que se llama *txRealTime()*. A esta interfaz se accede a través de la interfaz gráfica principal *main()*. La interfaz gráfica *txRealTime()* permite realizar las siguientes tareas:

- Abrir un archivo de texto.
- Mostrar el archivo de texto a transmitir.
- Seleccionar la técnica de mapeo.
- Activar o desactivar el reformado de pulso.
- Activar o desactivar la corrección de errores.
- Procesar los datos a transmitir.
- Seleccionar un intervalo de tiempo para transmitir en un lazo recursivo.
- Transmitir los datos según el intervalo de tiempo seleccionado.
- Detener el proceso de transmisión.

En la Figura 2.201 se muestra la interfaz gráfica *txRealTime.fig* y se identifican 3 componentes, esto ya que la interfaz gráfica es bastante grande y no se pueden identificar claramente todos los elementos.

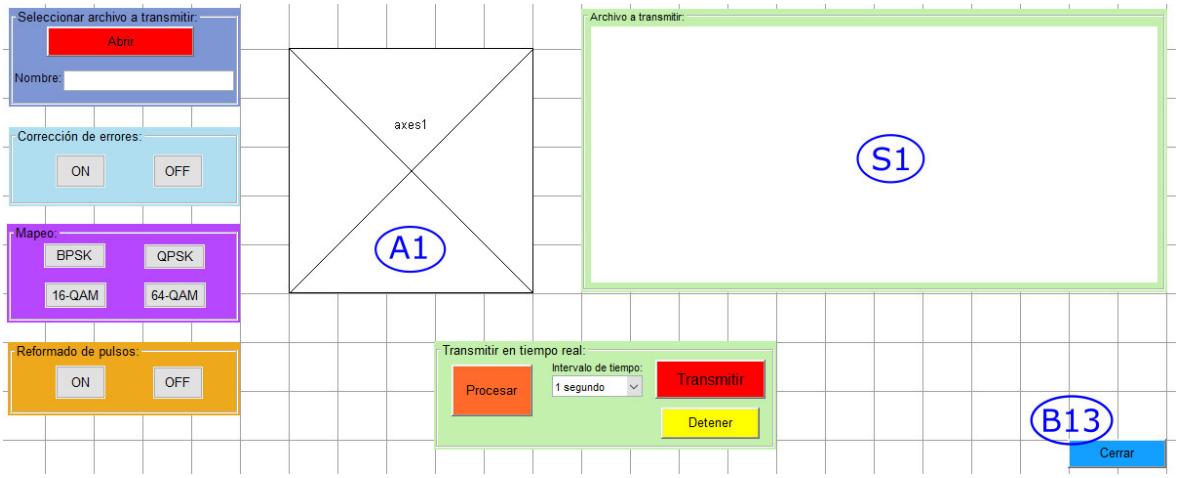


Figura 2.201. Interfaz gráfica *txRealTime.fig*

En la Figura 2.202 se muestra un acercamiento en la interfaz *txRealTime.fig*, esto con el objetivo de añadir los identificadores al resto de componentes. Cabe mencionar que los botones y/o elementos se dividen en secciones utilizando un elemento *Panel* para cada una de ellas. Las secciones que se tienen son; Seleccionar archivo a transmitir, Corrección de errores, Mapeo, Reformado de pulso, Transmitir en tiempo real, Archivo a transmitir.

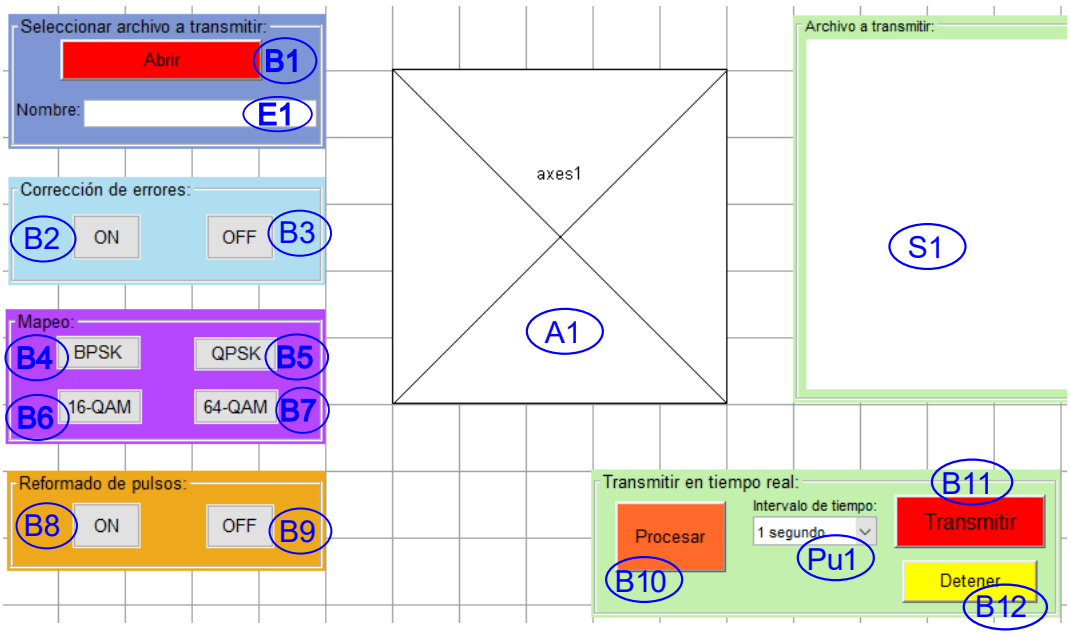


Figura 2.202. Parte de la interfaz gráfica *txRealTime.fig.*

En la Tabla 2.19 se presentan los componentes principales de la interfaz gráfica *txRealTime.fig* con su respectivo indicador, tag y string. El indicador se utiliza para identificar el elemento dentro de una figura, el tag corresponde a la asignación de nombre dentro de *txRealTime.m* y el string corresponde al texto a mostrar en el componente de *txRealTime.fig*.

Tabla 2.19. Nombres de los componentes de *txRealTime.fig.*

Indicador	Componente	Tag	String
A1	Axes	axes1	
B1	Push Button	btnAbrir	Abrir
B2	Push Button	btnOnErrores	ON
B3	Push Button	btnOffErrores	OFF
B4	Push Button	btnBPSK	BPSK
B5	Push Button	btnQPSK	QPSK
B6	Push Button	btn16QAM	16-QAM
B7	Push Button	btn64QAM	64-QAM
B8	Push Button	btnON	ON
B9	Push Button	btnOFF	OFF
B10	Push Button	btnProcesar	Procesar
B11	Push Button	btnTransmitir	Transmitir
B12	Push Button	btnDetener	Detener
B13	Push Button	btnCerrar	Cerrar
ET1	Edit Text	editNombre	
Pu1	Pop-up Menu	intervalo	1 segundo 2 segundos 3 segundos 4 segundos 5 segundos 6 segundos 7 segundos 8 segundos 9 segundos 10 segundos
S1	Static Text	txtOriginal	

Explicación breve sobre todos los componentes de la interfaz *txRealTime.fig*:

- El botón **Abrir (B1)** permite seleccionar y abrir un archivo de texto. En **E1** se muestra el nombre del archivo seleccionado y en **S1** muestra los caracteres del archivo seleccionado.
- El botón **ON (B2)** activa la corrección de errores a nivel de bit.
- El botón **OFF (B3)** desactiva la corrección de errores a nivel de bit.
- El botón **BPSK (B4)** permite seleccionar la técnica de mapeo BPSK.
- El botón **QPSK (B5)** permite seleccionar la técnica de mapeo QPSK.
- El botón **16-QAM (B6)** permite seleccionar la técnica de mapeo 16-QAM.
- El botón **64-QAM (B7)** permite seleccionar la técnica de mapeo 64-QAM.
- En el componente **A1** se muestra el diagrama de constelación de la técnica de mapeo seleccionada.
- El botón **ON (B8)** activa el reformado se pulso.
- El botón **OFF (B9)** desactiva el reformado se pulso.

- El botón **Procesar (B10)** permite procesar los datos serializados con las técnicas seleccionadas.
- El botón **Transmitir (B11)** permite transmitir los datos recursivamente de acuerdo al intervalo de tiempo escogido en **Pu1**.
- El botón **Detener (B12)** permite detener el proceso de transmisión recursiva.
- El botón **Cerrar (B13)** cierra la interfaz gráfica.

2.6.3.1. Funcionamiento de la interfaz gráfica de transmisión

En esta sección se explica el funcionamiento de la interfaz gráfica *txRealTime()*, recordando que esta interfaz se abre o se ejecuta a través de la interfaz gráfica principal. Además, se describe la utilidad de los componentes de la interfaz gráfica de transmisión.

2.6.3.1.1. Función de apertura de la interfaz gráfica de transmisión

Cuando se ejecuta *txRealTime()* se establece como opciones predeterminadas; la corrección de errores desactivada, el mapeo BPSK, el reformado de pulso desactivado y el intervalo de tiempo entre transmisiones en 3 segundos. En los botones se colorea con verde las opciones que están activadas. Además, se inhabilitan los botones **Procesar**, **Transmitir** y **Detener**. En la Figura 2.203 se muestra la interfaz gráfica de transmisión ejecutada a través de la interfaz gráfica principal y con los parámetros por defecto.

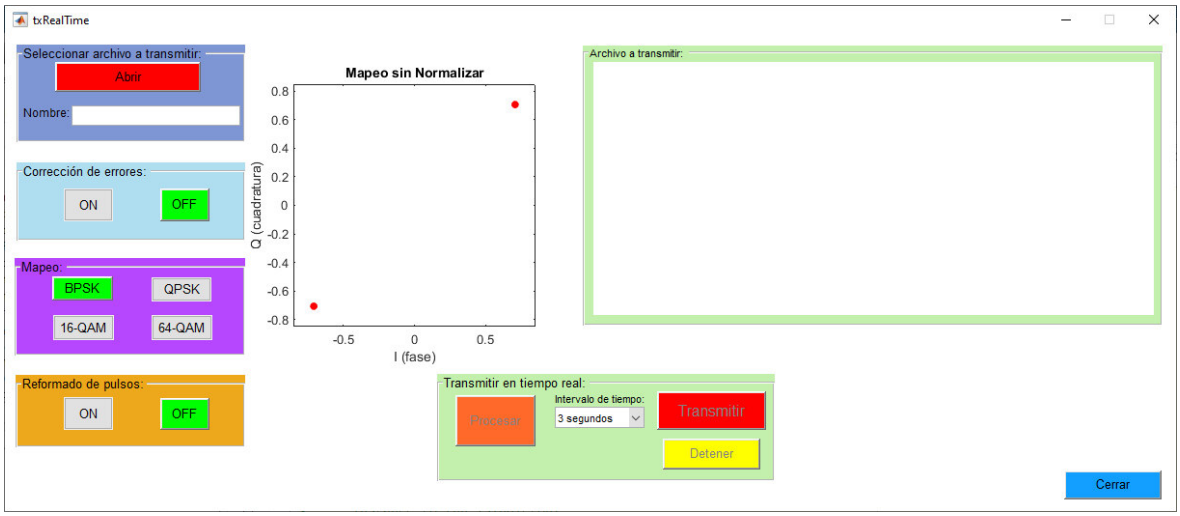


Figura 2.203. Interfaz gráfica *txRealTime()*.

2.6.3.1.2. Seleccionar archivo de texto a transmitir

Para seleccionar un archivo de texto mediante un cuadro de diálogo se presiona el botón **Abrir**. Si en el cuadro de diálogo no se selecciona ningún archivo en **E1** se muestra el mensaje "*Archivo no seleccionado*" para informar al usuario. Cuando se selecciona un archivo, en **E1** se muestra el nombre del archivo, en **S1** el contenido del mismo y se habilita

el botón **Procesar**. Además, en la ventana de comandos se informa al usuario que se ha abierto un archivo.

En la Figura 2.204 se muestra la interfaz gráfica de transmisión con el archivo de texto *ArchivoTextoRT.txt* seleccionado. El contenido del archivo de texto *ArchivoTextoRT.txt* se definió en la sección 2.3.1.3 de este documento.

El botón **Abrir** implementa la etapa Datos (ver Figura 2.206), correspondiente a una de las etapas de preparación y procesamiento de datos en transmisión.

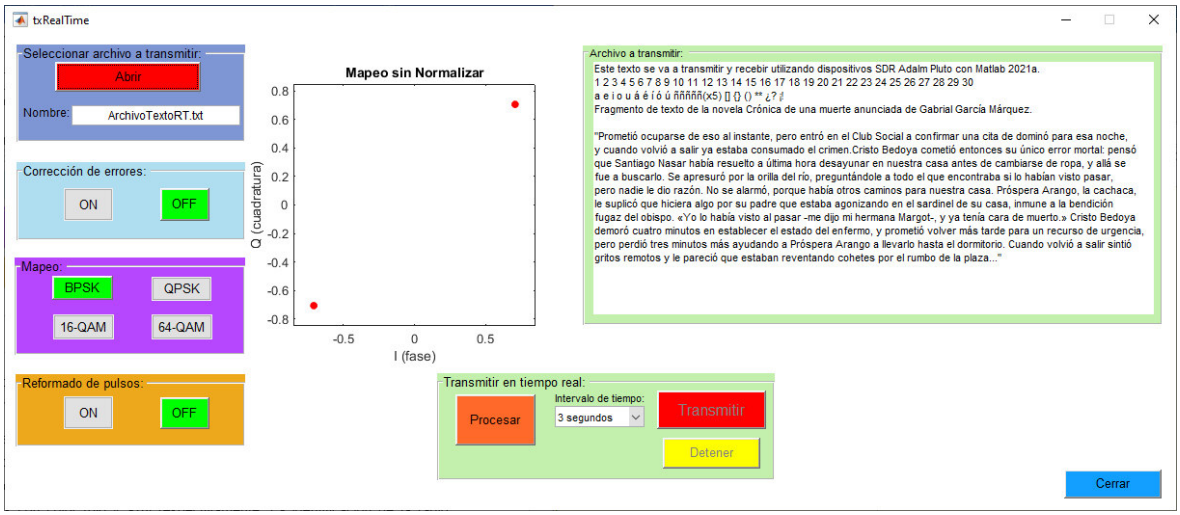


Figura 2.204. Interfaz gráfica *txRealTime()* con un archivo seleccionado.

2.6.3.1.3. Selección de técnicas

La corrección de errores a nivel de bit se puede activar o desactivar mediante los botones **ON (B2)** y **OFF (B3)** respectivamente. Con los botones **BPSK**, **QPSK**, **16-QAM** y **64-QAM** se puede seleccionar entre las diferentes técnicas de mapeo y el diagrama de constelación correspondiente se muestra en **A1**. El reformado de pulso se puede activar o desactivar mediante los botones **ON (B8)** y **OFF (B9)**.

2.6.3.1.4. Procesar datos

Cuando se presiona el botón **Procesar**, se aplica el procesamiento de todas las etapas que se señalan con color naranja en la Figura 2.205, a los datos del archivo de texto abierto previamente. También se colorea de verde y se habilita el botón **Transmitir**, el botón **Procesar (B10)** se desactiva, se colorea de verde claro y se cambia de nombre a "*Procesado*". Además, en la ventana de comandos de Matlab se muestran todas las etapas que se aplican al archivo de texto abierto.

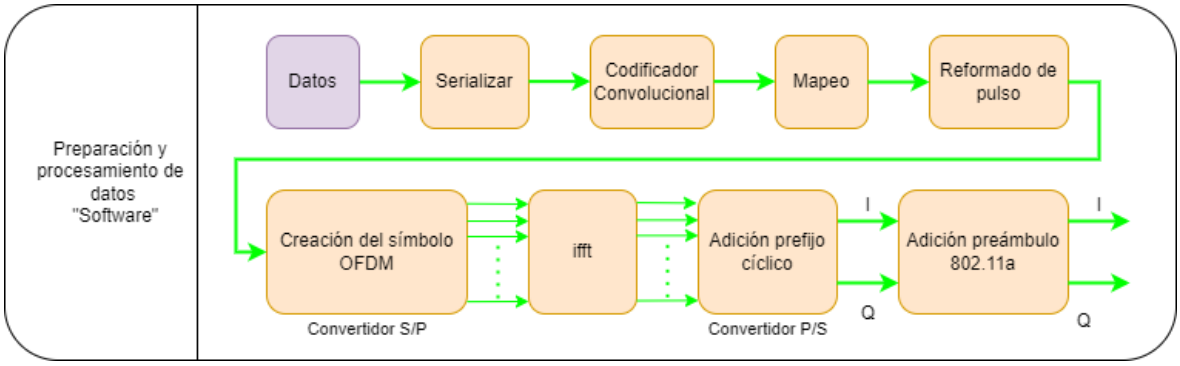


Figura 2.205. Etapas de preparación y procesamiento en el transmisor.

En la Figura 2.206 se muestra la interfaz gráfica de transmisión con la corrección de errores activada, con el mapeo 16-QAM seleccionado y con el reformado de pulso activado. Además, el botón **Procesar (B10)** ya se presionó y por ende se activó el botón **Transmitir** y se coloreo de verde, también se desactivó, se cambió de color y de nombre el botón **B10**.

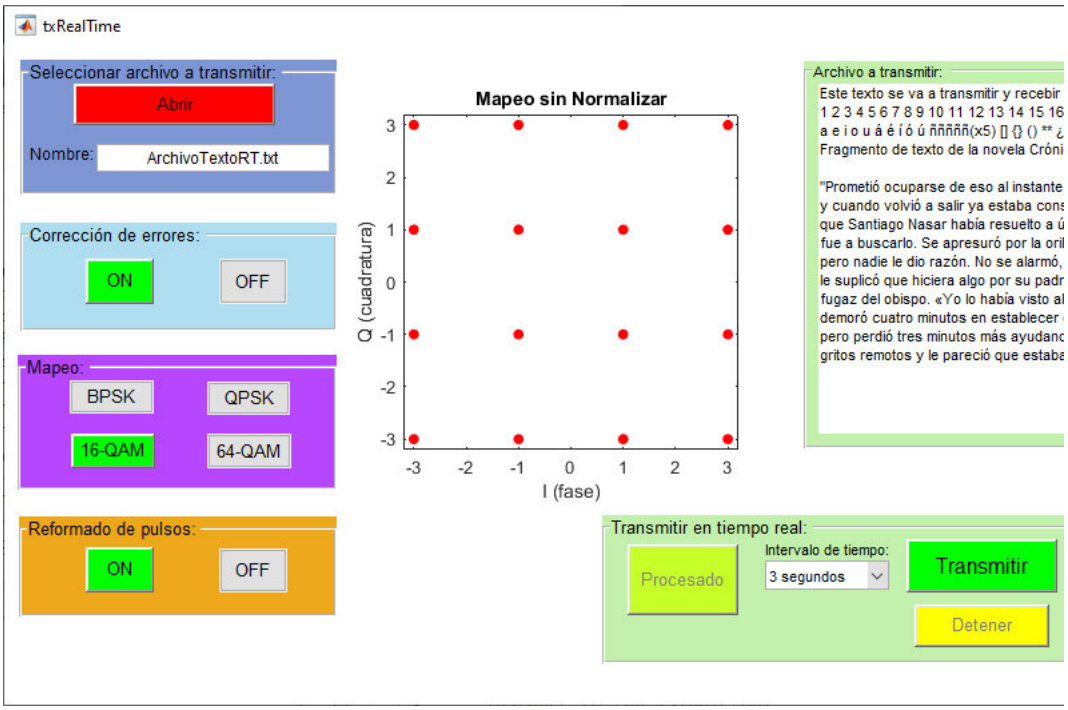


Figura 2.206. Interfaz gráfica *txRealTime()* con los datos procesados.

En la Figura 2.207 se muestra el resultado en la ventana de comandos luego de abrir y procesar el archivo de texto. La parte señalada con color verde corresponde al resultado cuando se selecciona y se abre un archivo de texto. Mientras que la parte con rojo corresponde al resultado cuando se presiona el botón **Procesar (B10)**, teniendo en cuenta las opciones seleccionadas en la interfaz gráfica de la Figura 2.206.

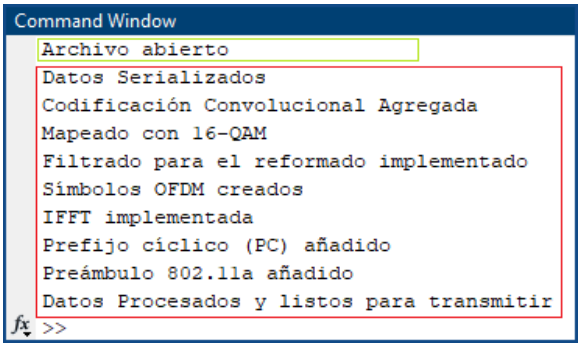


Figura 2.207. Resultado en la ventana de comandos cuando se procesan los datos.

2.6.3.1.5. Transmitir datos

Una vez procesados los datos del archivo de texto se transmiten los datos al canal inalámbrico. El menú desplegable **Pu1** permite seleccionar el intervalo de tiempo que se espera entre transmisiones, los valores que se pueden seleccionar son de 1 segundo hasta 10 segundos. Cuando se presiona el botón **Transmitir (B11)**, se transmiten los datos al canal inalámbrico de manera recursiva esperando el tiempo que especifica **Pu1**. También se cambia el nombre del botón **B11** a "*Transmitiendo…"* y se colorea en celeste. Además, el menú desplegable y todos los botones (excepto **B11** y **Detener (B12)**) se desactivan. En la ventana de comandos se muestra un mensaje para informar que se ha transmitido los datos.

En la Figura 2.208 se muestra parte de la interfaz gráfica de transmisión, cuando se está transmitiendo los datos al canal inalámbrico cada 3 segundos. Para parar la transmisión se presiona el botón **Detener** y la interfaz vuelve al estado de la Figura 2.206.

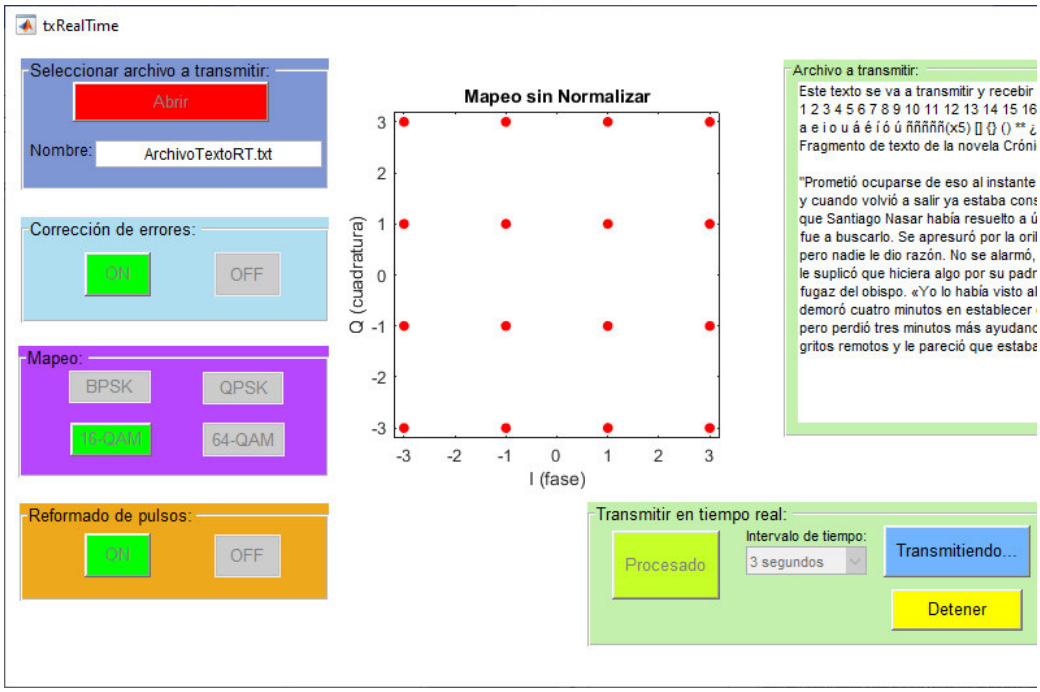


Figura 2.208. Interfaz gráfica *txRealTime()* transmitiendo los datos al canal.

En la Figura 2.209 se muestra encerrado en color verde el resultado en la ventana de comandos cuando se presiona el botón **Transmitir**. Antes de enviar los datos al canal inalámbrico utilizando el SDR Adalm Pluto, se establece la conexión con el mismo. Luego cuando se transmite los datos se muestra el mensaje "*Datos transmitidos…*". En este caso se enviaron 5 veces los datos antes de detener la transmisión. Cuando se detiene la transmisión mediante el botón **Detener** se muestra el mensaje encerrado color rojo.

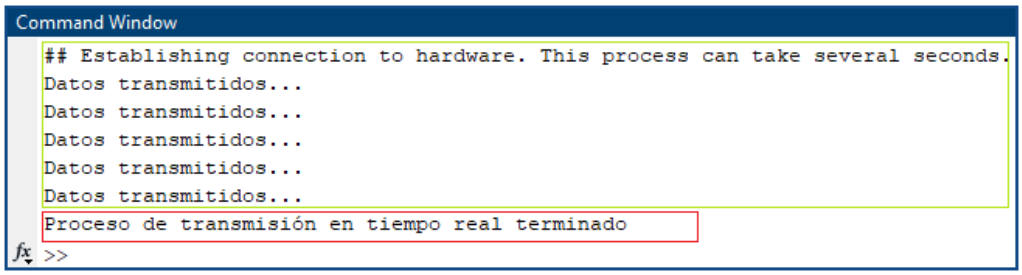


Figura 2.209. Resultado en la ventana de comandos cuando se transmiten los datos.

2.6.4. INTERFAZ GRÁFICA DE RECEPCIÓN

En esta sección se describe la interfaz gráfica de recepción en tiempo real, que se llama *rxRealtime()*. A esta interfaz se accede a través de la interfaz gráfica principal *main()*, luego de haber configurado los objetos de transmisión y recepción. La interfaz gráfica *rxRealtime()* permite realizar las siguientes tareas:

- Activar o desactivar el reformado de pulso.
- Seleccionar la técnica de demapeo.
- Activar o desactivar la corrección de errores.
- Activar o desactivar la medición del BER.
- Seleccionar el archivo de texto original con el que se compara los datos recuperados.
- Mostrar los caracteres del archivo de texto seleccionado.
- Seleccionar el umbral de detección del equipo SDR Adalm Pluto de recepción.
- Iniciar la recepción y procesamiento de datos del canal inalámbrico, mediante un botón, es decir, iniciar la recepción en tiempo real.
- Graficar los valores de la estimación del desplazamiento de frecuencia.
- Graficar las constelaciones de los datos recibidos antes de ser demapeados.
- Mostrar el valor del BER en decimal y en fracción.
- Mostrar los caracteres del texto recuperado del canal inalámbrico.
- Detener la recepción en tiempo real.

En la Figura 2.210 se muestra la interfaz gráfica *rxRealtime.fig* completa y se identifican algunos de los componentes. Esto debido a que, la interfaz gráfica es bastante grande y no se pueden identificar claramente todos los elementos, cuando se presenta la imagen de la misma.

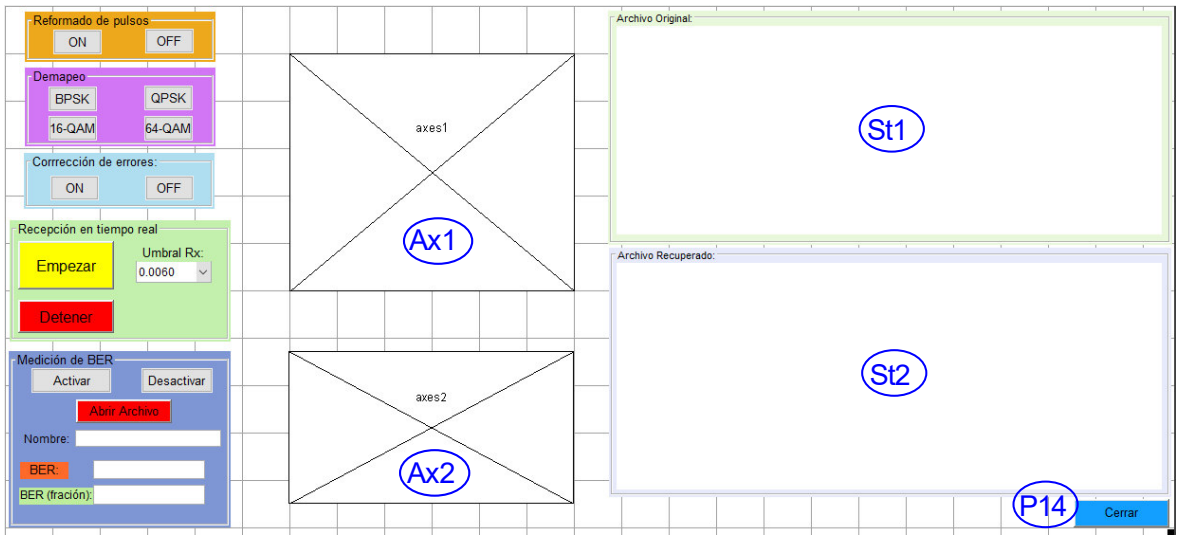


Figura 2.210. Interfaz gráfica *rxRealtime.fig*

En la Figura 2.211 se muestra la interfaz gráfica *rxRealtime.fig* con un acercamiento y con la mayoría de los componentes con identificadores. Los botones y/o elementos se dividen en secciones utilizando un elemento *Panel*. Las secciones que se tienen son; Reformado de pulso, Demapeo, Corrección de errores, Recepción en tiempo real, Medición del BER, Archivo Original y Archivo Recuperado.

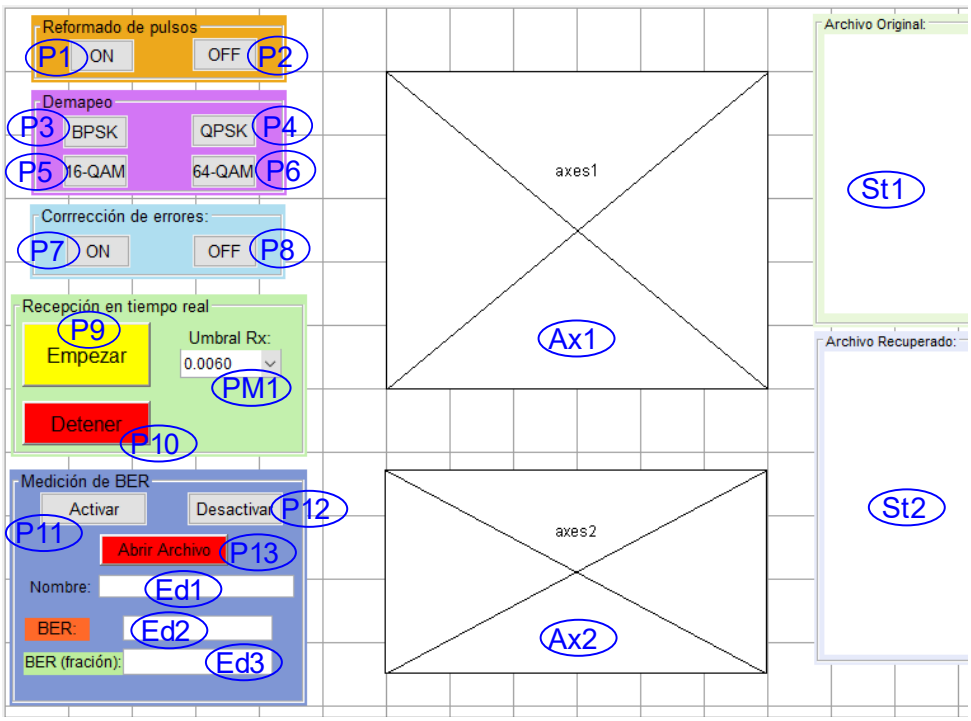


Figura 2.211. Parte de la interfaz gráfica *rxRealtime.fig.*

En la Tabla 2.20 se presenta los componentes principales de la interfaz gráfica *rxRealtime.fig* con su respectivo indicador, tag y string. El indicador identifica al elemento dentro de una figura, el tag corresponde a la asignación de nombre dentro de *rxRealtime.m* y el string corresponde al texto a mostrar en el componente de *rxRealtime.fig*.

Tabla 2.20. Nombres de los componentes de *rxRealtime.fig*.

Indicador	Componente	Tag	String
Ax1	Axes	axes1	
Ax2	Axes	axes2	
Ed1	Edit Text	editNombre	
Ed2	Edit Text	editBER	
Ed3	Edit Text	editBER2	
P1	Push Button	btnON	ON
P2	Push Button	btnOFF	OFF
P3	Push Button	btnBPSK	BPSK
P4	Push Button	btnQPSK	QPSK
P5	Push Button	btn16QAM	16-QAM
P6	Push Button	btn64QAM	64-QAM
P7	Push Button	btnOnErrores	ON
P8	Push Button	btnOffErrores	OFF
P9	Push Button	btnEmpezar	Empezar
P10	Push Button	btnDetener	Detener
P11	Push Button	btnActivar	Activar
P12	Push Button	btnDesactivar	Desactivar
P13	Push Button	btnAbrir	Abrir Archivo
P14	Push Button	btnCerrar	Cerrar
PM1	Pop-up Menu	RTumbral	0.0060 0.0050 0.0040 0.0030
St1	Static Text	txtAOriginales	
St2	Static Text	txtARecuperado	

Explicación breve sobre todos los componentes de la interfaz gráfica de recepción:

- El botón **ON (B1)** activa el reformado se pulso.
- El botón **OFF (B2)** desactiva el reformado se pulso.
- El botón **BPSK (B3)** permite seleccionar la técnica de demapeo BPSK.
- El botón **QPSK (B4)** permite seleccionar la técnica de demapeo QPSK.
- El botón **16-QAM (B5)** permite seleccionar la técnica de demapeo 16-QAM.
- El botón **64-QAM (B6)** permite seleccionar la técnica de demapeo 64-QAM.
- El botón **ON (P7)** activa la corrección de errores a nivel de bit.
- El botón **OFF (P8)** desactiva la corrección de errores a nivel de bit.
- El botón **Activar (P11)** activa la medición del BER.

- El botón **Desactivar (P12)** desactiva la medición del BER.
- El botón **Abrir Archivo (P13)** permite seleccionar y abrir un archivo de texto cuando se ha activado la medición del BER. En **Ed1** se muestra el nombre del archivo seleccionado y en **St1** muestra los caracteres del archivo seleccionado.
- El menú desplegable **PM1** permite seleccionar el umbral de recepción.
- El botón **Empezar (P9)** permite recibir, procesar y recuperar los datos del canal inalámbrico, en base a las técnicas seleccionadas. Este proceso se realiza en un bucle recursivo.
- En **Ax1** se grafica el diagrama de constelación antes de ser demapeados.
- En **Ax2** se grafica los valores de la estimación del desplazamiento de frecuencia.
- En **St2** se muestra los caracteres recuperados del canal inalámbrico.
- En **Ed2** se muestra el valor del BER en decimales y en **Ed3** se muestra el valor del BER en fracción.
- El botón **Detener (B10)** permite detener el proceso de recepción recursiva.
- El botón **Cerrar (B14)** cierra la interfaz gráfica.

2.6.4.1. Funcionamiento de la interfaz gráfica de recepción

En esta sección se explica el funcionamiento de la interfaz gráfica de recepción en tiempo real *rxRealtime()*. Además, se describe la utilidad de los componentes de esta interfaz gráfica. Hay que recordar que esta interfaz se abre o se ejecuta a través de la interfaz gráfica principal *main()*.

2.6.4.1.1. Función de apertura de la interfaz gráfica de recepción

Cuando se ejecuta *rxRealtime()* se establece como opciones predeterminadas; la corrección de errores desactivada, el mapeo BPSK y el reformado de pulso desactivado, igual que en la interfaz gráfica de transmisión. La medición del BER se activa por defecto y el usuario puede seleccionar el archivo de texto con el botón **Abrir Archivo**. Para identificar los botones seleccionados, por defecto, estos se colorean de verde. En la interfaz gráfica se inhabilitan los botones **Empezar** y **Detener**. Además, se establece el umbral de recepción en 0.040. En la Figura 2.212 se muestra la interfaz gráfica de transmisión ejecutada a través de la interfaz gráfica principal y con los parámetros por defecto establecidos.

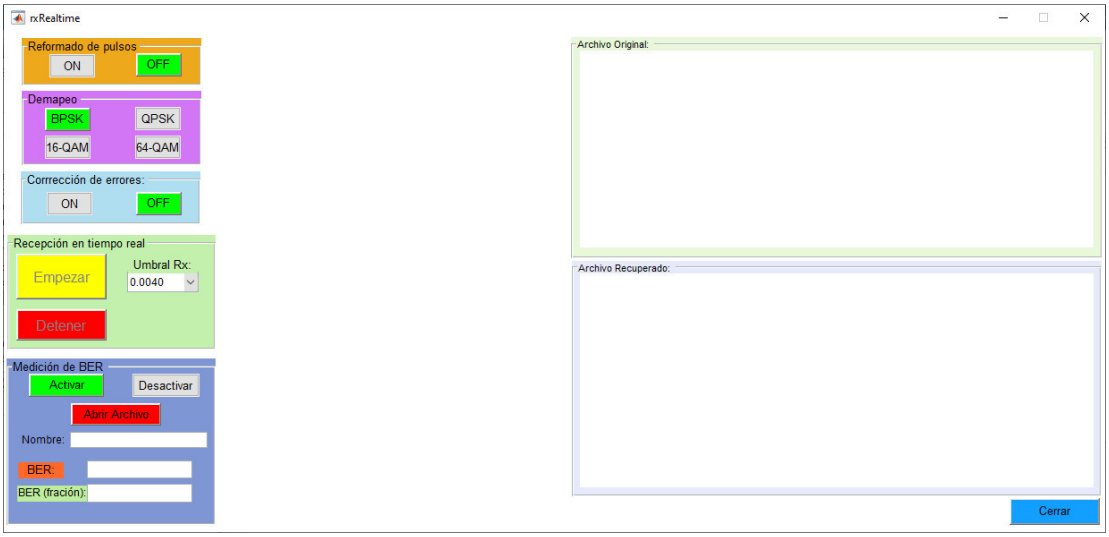


Figura 2.212. Interfaz gráfica de recepción *rxRealtime()*.

2.6.4.1.2. Selección de técnicas

La técnica de demodulación digital, la activación o desactivación de la corrección de errores y del reformado de pulso deben coincidir con las opciones seleccionadas en la interfaz gráfica de transmisión. Si las opciones seleccionadas difieren en recepción que en transmisión, no se podrá recuperar los datos satisfactoriamente.

El reformado de pulso se puede activar o desactivar mediante los botones **ON (B1)** y **OFF (B2)**. Con los botones **BPSK**, **QPSK**, **16-QAM** y **64-QAM** se puede seleccionar entre las diferentes técnicas de demapeo. La corrección de errores a nivel de bit se puede activar o desactivar mediante los botones **ON (B7)** y **OFF (B8)** respectivamente. La medición del BER se puede activar con los botones **Activar** y **Desactivar**, respectivamente.

Cuando la medición del BER está activada se selecciona el archivo de texto original con el botón **Abrir Archivo**. Cuando se selecciona el archivo, los caracteres de este se muestran en **St1** y se habilita el botón **Empezar** para iniciar la recepción de datos en tiempo real. Si se desactiva la medición del BER, se habilita el botón **Empezar** y se vacía el contenido que pudiera estar almacenado en **St1**, **Ed2** y **Ed3**.

En la Figura 2.213 se muestra la interfaz gráfica de recepción con el archivo de texto *ArchivoTextoRT.txt* seleccionado para comparar con los datos recuperados. Este archivo se definió en la sección 2.3.1.3 de este documento. Al igual que en el ejemplo de la interfaz gráfica de transmisión se selecciona el demapeo 16-QAM, el reformado de pulso y la corrección de errores a nivel de bit activada. El umbral se recepción que se puede seleccionar con el menú desplegable **PM1** se mantiene en el valor por defecto, es decir 0.0040. Además, el botón **Empezar** ya está habilitado, para que el usuario lo pueda presionar.

Figura 2.213. Interfaz gráfica *rxRealtime()* con un archivo seleccionado para la medición del BER.

2.6.4.1.3. Recibir y procesar datos en tiempo real

Cuando se presiona el botón **Empezar (P9)**, primero se habilita el botón **Detener** y se inhabilita todos los botones de la interfaz gráfica a excepción del botón **Detener** y **Cerrar**. También se cambia el nombre del botón **P9** a *"Recibiendo…"* y en la ventana de comandos de Matlab se muestra el mensaje *"Recepción en tiempo real iniciado"*, esto para informar al usuario que se ha iniciado el proceso de recepción. Además, el botón **Empezar** permite implementar todas las etapas de recepción (Figura 2.214).

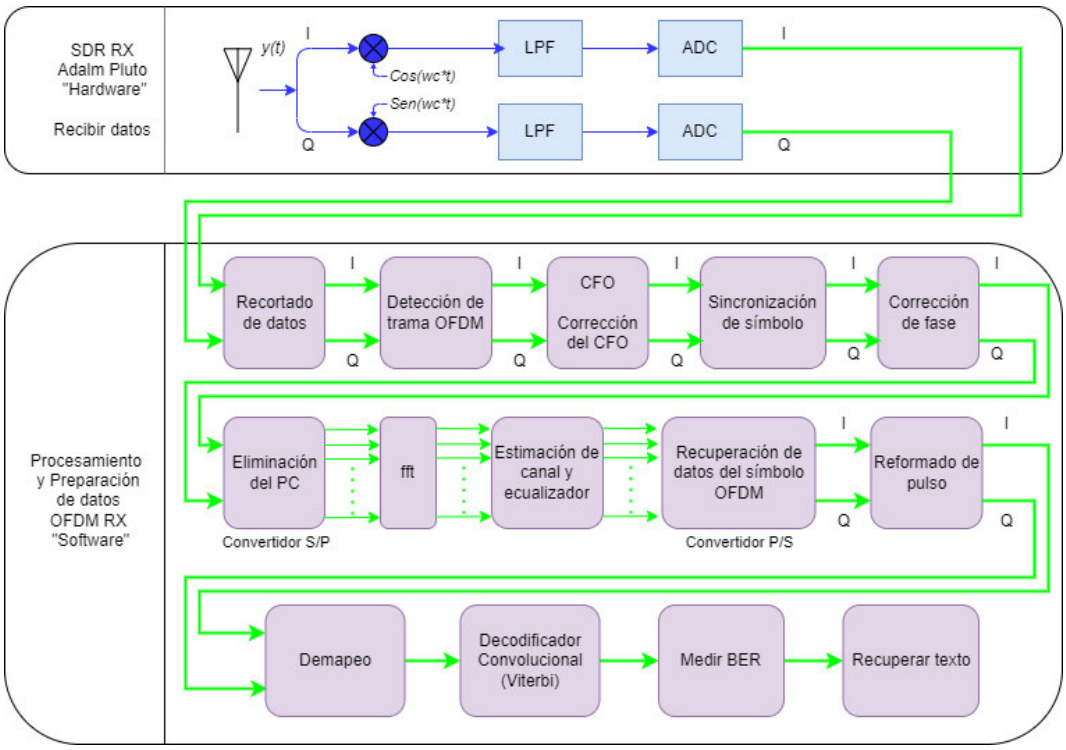


Figura 2.214. Etapas en el receptor.

Cuando se desactivan las etapas Reformado de Pulso, Decodificador Convolucional o Medir BER, estas no se implementan y simplemente se copian los datos de las etapas previas en la variable de entrada de la etapa siguiente.

En la Figura 2.215 se muestra la interfaz gráfica de recepción cuando ya se presionó el botón **Empezar** y se han recibido y procesado en 5 ocasiones los datos enviados por la interfaz gráfica de transmisión. Se sabe que ya se han recibido 5 veces los datos debido a la gráfica de los valores de la estimación de frecuencia en **Ax2**. La gráfica de constelaciones de **Ax1**, el valor del BER obtenido y el texto en **St2**, corresponden a los últimos datos recuperados.

El escenario en donde se realizó las transmisiones es a **1.5 metros** y con **línea de vista.**

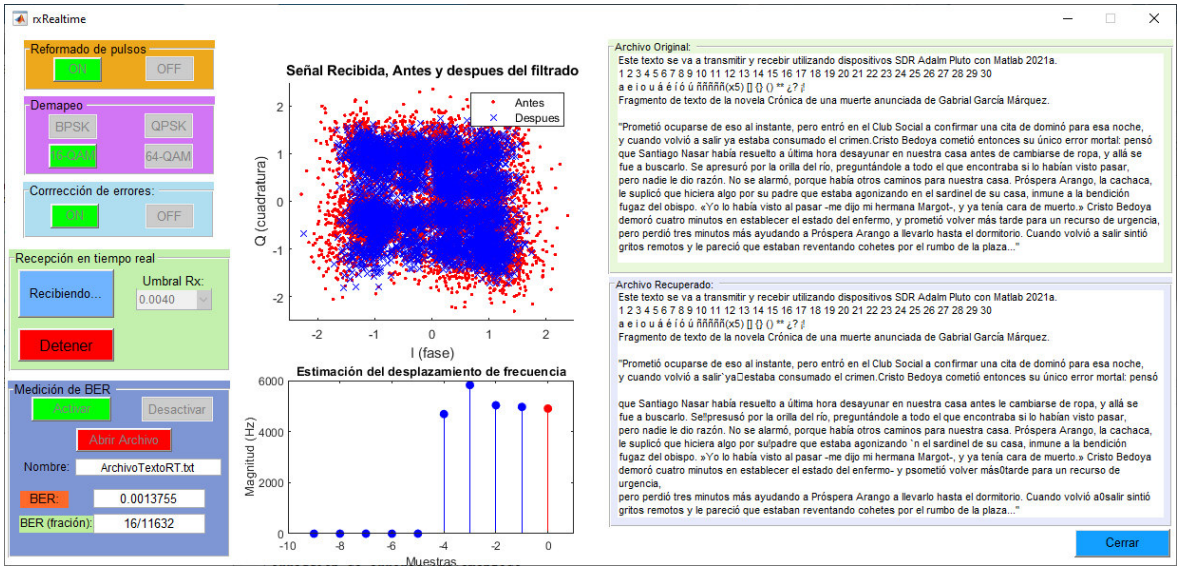


Figura 2.215. Interfaz gráfica *rxRealtime()* recibiendo y procesando los datos en tiempo real.

En la Figura 2.216 se muestra un zoom a la interfaz gráfica de recepción, para poder observar de mejor manera los valores del BER medido, el diagrama de constelación y la gráfica de los valores de estimación de frecuencia.

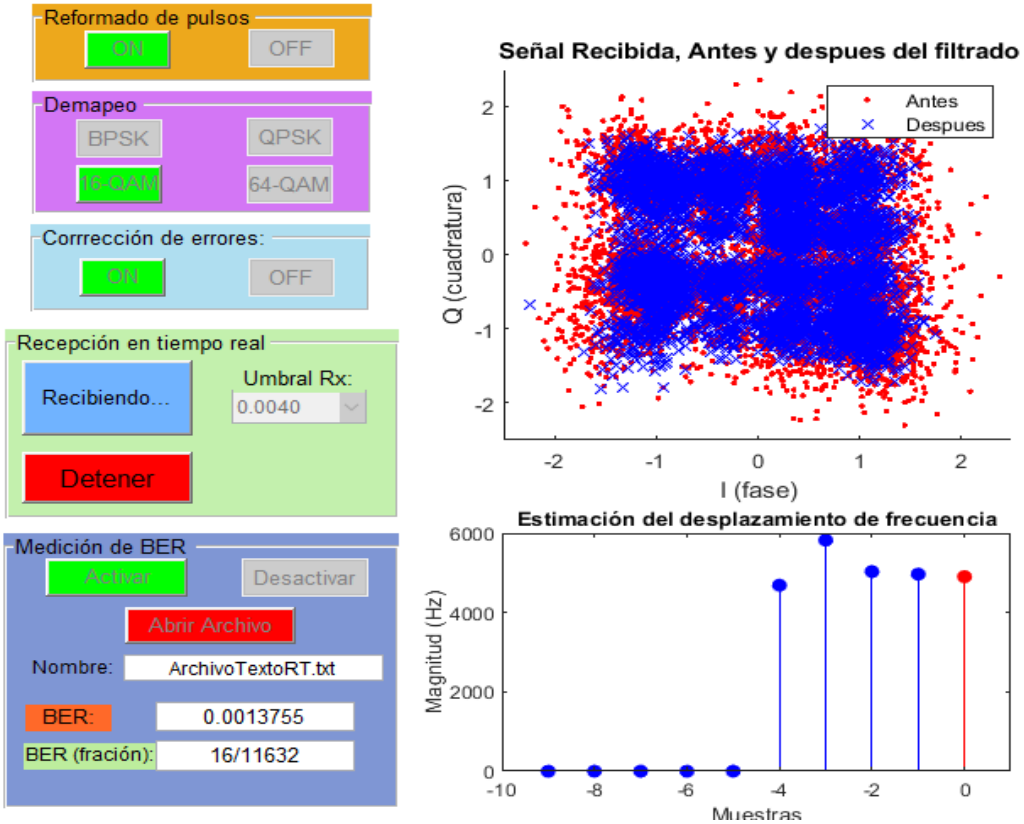


Figura 2.216. Acercamiento a la interfaz gráfica *rxRealtime()*.

En la Figura 2.217 se muestra el texto del archivo original en la parte superior y el texto de los datos recibidos en la parte de abajo.

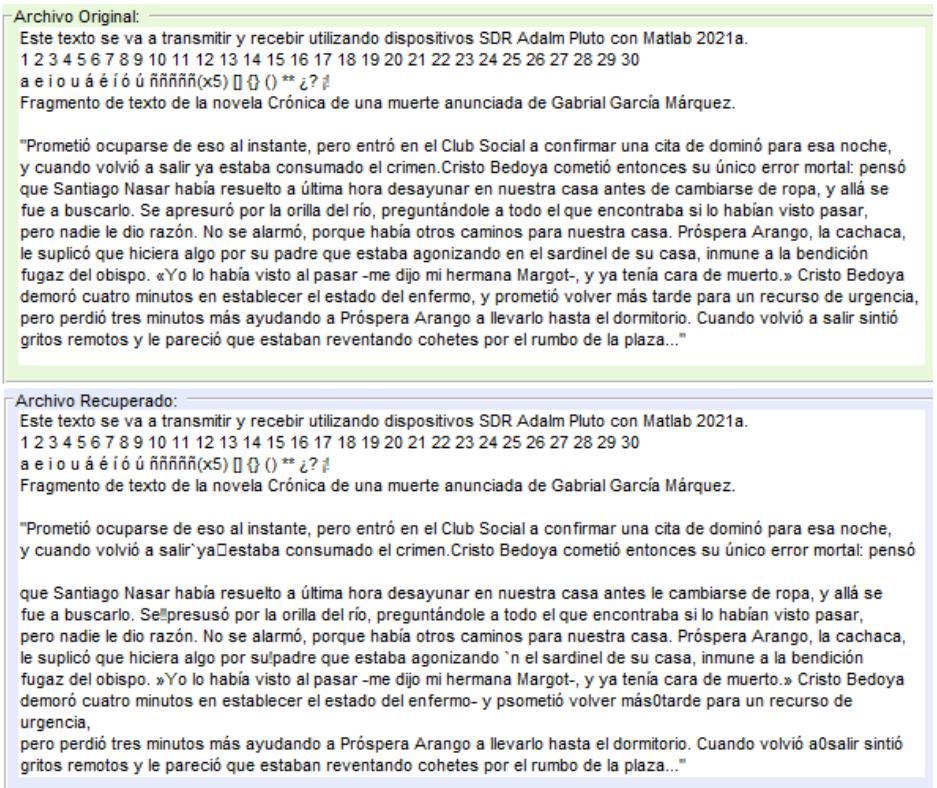


Figura 2.217. Archivo de texto original vs recibido.

En la Figura 2.218 se muestra un ejemplo de los resultados en la ventana de comandos cuando se presiona por primera vez en botón **Empezar**. Se inicia con un mensaje informativo, luego se conecta Matlab con la radio Adalm Pluto a través del objeto de recepción. Después se muestrea el canal inalámbrico hasta superar el valor del umbral de 0.0040. Luego se muestra los mensajes informativos de cada etapa de recepción, esto sirve para que el usuario pueda ir siguiendo el proceso. Finalmente, se informa al usuario que se ha recuperado el archivo de texto.

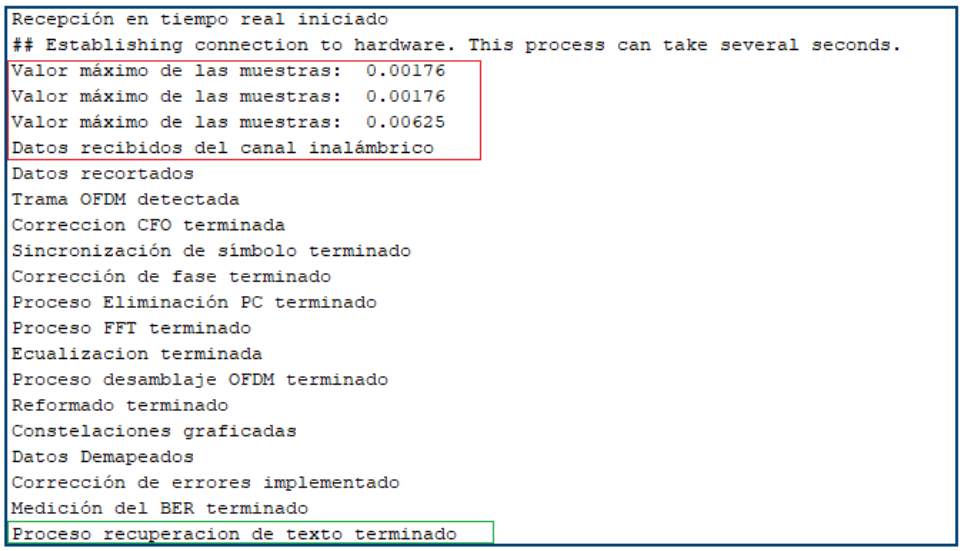


Figura 2.218. Resultado en la ventana de comandos cuando se presiona por primera vez el botón **Empezar**.

3. RESULTADOS Y DISCUSIÓN

3.1. PRUEBAS DE TRANSMISIÓN CON LÍNEA DE VISTA (INTERIORES)

El parámetro de evaluación para comprobar la utilidad de los algoritmos de modulación digital, de la corrección de errores a nivel de bit y del reformado de pulso, es la tasa de bits errados (BER). En esta sección se lleva a cabo las pruebas de funcionamiento con línea de vista y a diferentes distancias. Para cada valor de distancia se tiene una tabla resumen, en donde se presentan todas las posibles combinaciones de las técnicas de prototipo.

Los parámetros de configuración de los objetos del sistema de transmisión y recepción se mantienen en los parámetros por defecto configurados en la interfaz gráfica principal. Estos parámetros se detallaron en la sección de configuración del capítulo anterior, específicamente en la Tabla 2.2 y Tabla 2.3. Entre los parámetros más importantes están:

- Ganancia de transmisión igual a -2 dB.
- Ganancia de recepción igual a 20 dB.
- Tamaño del bloque de datos igual a 800 000.
- Frecuencia central igual a 860 MHz.
- Frecuencia de muestreo en 2 MHz.

El archivo de texto que se utiliza para realizar las pruebas de funcionamiento es "*ArchivoTextoRT.txt*", este contiene 1454 caracteres y el contenido del mismo se presenta a continuación:

Este texto se va a transmitir y recebir utilizando dispositivos SDR Adalm Pluto con Matlab 2021a.
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
a e i o u á é í ó ú ñññññ(x5) [] {} () ** ¿? ¡!
Fragmento de texto de la novela Crónica de una muerte anunciada de Gabrial García Márquez.

"Prometió ocuparse de eso al instante, pero entró en el Club Social a confirmar una cita de dominó para esa noche, y cuando volvió a salir ya estaba consumado el crimen.Cristo Bedoya cometió entonces su único error mortal: pensó que Santiago Nasar había resuelto a última hora desayunar en nuestra casa antes de cambiarse de ropa, y allá se fue a buscarlo. Se apresuró por la orilla del río, preguntándole a todo el que encontraba si lo habían visto pasar, pero nadie le dio razón. No se alarmó, porque había otros caminos para nuestra casa. Próspera Arango, la cachaca, le suplicó que hiciera algo por su padre que estaba agonizando en el sardinel de su casa, inmune a la bendición fugaz del obispo. «Yo lo había visto al pasar -me dijo mi hermana Margot-, y ya tenía cara de muerto.» Cristo Bedoya demoró cuatro minutos en establecer el estado del enfermo, y prometió volver más tarde para un recurso de urgencia, pero perdió tres minutos más ayudando a Próspera Arango a llevarlo hasta el dormitorio. Cuando volvió a salir sintió gritos remotos y le pareció que estaban reventando cohetes por el rumbo de la plaza..."

En la Figura 3.1 se muestra un diagrama con el ambiente en donde se realizan las pruebas de funcionamiento. Este ambiente es un **ambiente interior y con línea de vista**. En el diagrama todas las medidas están en centímetros, la cota con azul corresponde a la línea de vista entre los equipos SDR y las cotas con verde son las medidas del ambiente interior. Para las pruebas, el equipo SDR de transmisión se mantiene fijo, mientras que el equipo SDR de recepción se coloca a diferentes distancias a lo largo de la línea azul. Cabe mencionar que los equipos SDR se encuentran a una **altura de 50 centímetros.**

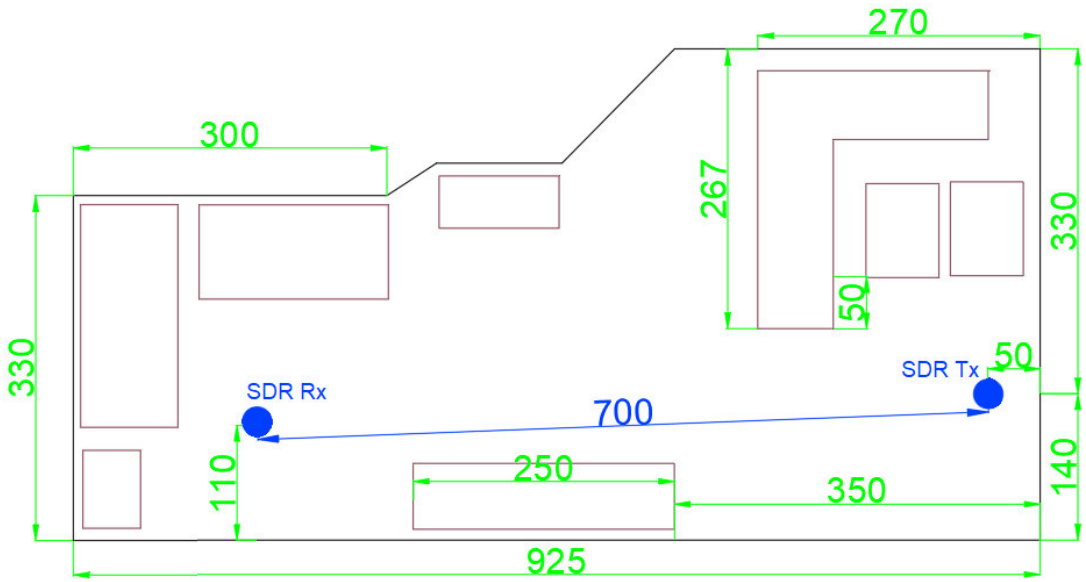


Figura 3.1. Representación del ambiente interior con línea de vista en donde se realizan las pruebas. Las distancias están en centímetros (cm).

En la Tabla 3.1 se muestran los resultados de las pruebas realizadas, en donde los equipos se colocan a una distancia de **un metro** de separación entre ellos. Se utiliza línea de vista directa y el ambiente interior de la Figura 3.1.

Cada valor de BER mostrado en las tablas de este capítulo y correspondiente a cada una de las combinaciones o configuraciones obtenidas con el prototipo de comunicación inalámbrica, se obtiene a partir del promedio de al menos 5 valores. Es decir que, en cada combinación, se espera a recuperar al menos 5 valores de BER de la interfaz gráfica de recepción, para obtener el promedio y seguir con la siguiente combinación.

Tabla 3.1. Pruebas de funcionamiento a 1 metro, con línea de vista y ambiente interior.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0	0/11632
		X	X		0	0/11632
	X			X	0	0/11632
	X		X		0	0/11632
QPSK		X		X	0	0/11632
		X	X		0	0/11632
	X			X	0	0/11632
	X		X		0	0/11632
16-QAM		X		X	0.022524	262/11632
		X	X		0.011520	134/11632
	X			X	0.002321	27/11632
	X		X		0.001376	16/11632
64-QAM		X		X	0.072215	840/11632
		X	X		0.062586	728/11632
	X			X	0.042383	493/11632
	X		X		0.024759	288/11632

De la Tabla 3.1 los resultados del mapeo BPSK y QPSK podemos decir que en estos dos esquemas se recupera la información con un valor de BER igual a 0, en cualquier combinación posible. Esto se debe al bajo número de símbolos en la constelación, ya que la dispersión de los mismos no afecta en gran medida a la recuperación de datos. Cuando se utiliza el esquema de mapeo 16-QAM sin ninguna técnica adicional, ya se tiene una medida de BER y resulta que en 64-QAM esta medida aumenta significativamente. Esto se debe a que a mayor número de estados en la constelación mayor es la dificultad en realizar su clasificación y por tanto mayor será la cantidad de bits errados.

En el caso del mapeo 16-QAM y 64-QAM se puede ver la utilidad del reformado de pulso y de la corrección de errores sobre el BER. Cuando solamente se activa el reformado de pulso se obtiene una mejora en el valor del BER. Cuando solo se activa la corrección de errores también se obtiene una mejora y resulta ser más efectivo que el reformado de pulso. Si se utiliza el reformado y la corrección en conjunto, se obtiene el mejor valor posible del BER.

En la Tabla 3.2 se muestran los resultados de las pruebas realizadas, en donde los equipos se colocan a una distancia de **3 metros** entre ellos. Se utiliza línea de vista directa y el ambiente interior de la Figura 3.1.

Tabla 3.2. Pruebas de funcionamiento a 3 metros y con línea de vista.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0.000258	3/11632
		X	X		0.000086	1/11632
	X			X	0.000086	1/11632
	X		X		0	0/11632
QPSK		X		X	0.000344	4/11632
		X	X		0.000086	1/11632
	X			X	0.000086	1/11632
	X		X		0	0/11632
16-QAM		X		X	0.075911	883/11632
		X	X		0.024587	286/11632
	X			X	0.008769	102/11632
	X		X		0.001633	19/11632
64-QAM		X		X	0.105227	1224/11632
		X	X		0.096802	1126/11632
	X			X	0.085368	993/11632
	X		X		0.061382	714/11632

En la Tabla 3.2 se observa que tanto en BPSK como en QPSK ya se tiene una medida de error, siendo el error que se presenta en QPSK ligeramente mayor a BPSK, cuando no se utiliza ni reformado ni corrección de errores. Tanto en QPSK como en BPSK, cuando se utiliza reformado o corrección de errores el valor es el mismo, es decir resulta igual de efectivo utilizar cualquiera de las dos técnicas por separado. Además, el resultado de utilizar reformado y corrección de errores combinados en BPSK y QPSK, es que se tenga un BER de 0.

En la Tabla 3.2 en los esquemas de mapeo 16-QAM y 64-QAM existe un aumento de BER en comparación con los resultados de las pruebas realizadas a un metro (Tabla 3.1). Pero se mantienen el patrón de que cuando se utiliza reformado de pulso y/o corrección de errores mejora el valor del BER. Utilizando la corrección de errores se obtienen mejores resultados que utilizando el reformado de pulso y utilizando las dos técnicas combinadas se obtienen los mejores resultados del BER.

En la Tabla 3.3 se muestra los resultados de las pruebas realizadas, en donde los equipos se colocan a una distancia de **5 metros** entre ellos. Se utiliza línea de vista directa y el ambiente interior de la Figura 3.1. Además, cabe mencionar que en unas pocas ocasiones no se pudo recuperar los datos de información, lo que causa que no se obtenga un valor de BER. Como no hubo necesidad de cambiar el umbral de recepción, se puede decir que con las ganancias configuradas por defecto el prototipo está cerca del límite de distancia.

Tabla 3.3. Pruebas de funcionamiento a 5 metros, con línea de vista y ambiente interior.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0.000946	11/11632
		X	X		0.000172	2/11632
	X			X	0.000172	2/11632
	X		X		0.000172	2/11632
QPSK		X		X	0.000688	8/11632
		X	X		0.000430	5/11632
	X			X	0.000516	6/11632
	X		X		0.000086	1/11632
16-QAM		X		X	0.110729	1288/11632
		X	X		0.045306	527/11632
	X			X	0.049003	570/11632
	X		X		0.009027	105/11632
64-QAM		X		X	0.139271	1620/11632
		X	X		0.127235	1480/11632
	X			X	0.135660	1578/11632
	X		X		0.092675	1078/11632

En la Tabla 3.3 se mantiene el patrón que cuando se utiliza solamente la modulación digital (BPSK, QPSK, 16-QAM o 64-QAM) se obtiene el mayor valor de BER en comparación a utilizar reformado de pulso y/o corrección de errores. También, cuando se utiliza en conjunto el reformado de pulso y la corrección de errores se obtiene el menor valor del BER, de todas las combinaciones posibles para un mismo tipo de mapeo. Este resultado concuerda con los resultados obtenidos a la distancia de 1 metro y de 3 metros.

Además, en la Tabla 3.3 se observa un resultado particular, si se utiliza solo el mapeo BPSK se obtiene más errores que utilizar solamente el mapeo QPSK, cuando el resultado que se esperaba es que en QPSK se obtenga un mayor BER que en BPSK. Ya que la diferencia de BER no es significativa, el resultado obtenido puede deberse a la naturaleza impredecible del canal inalámbrico. Lo que se comprueba en los resultados de pruebas posteriores, ya que no se vuelve a repetir que en QPSK se obtenga mejores valores de BER en comparación a BPSK.

Otro resultado que cambia en la Tabla 3.3 en comparación a las pruebas anteriores, es que en QPSK, 16-QAM y 64-QAM, al utilizar el reformado de pulso se obtiene un menor valor del BER, en comparación a utilizar corrección de errores. Como las pruebas se realizaron a una mayor distancia (5 metros) que en los casos anteriores (1 y 3 metros), se puede decir lo siguiente:

El reformado de pulso resulta de mayor utilidad que la corrección de errores, cuando la relación señal al ruido (SNR) disminuye.

Entendiendo que el prototipo no permite obtener una medida de SNR, pero tomando en cuenta que el nivel de ruido y la amplitud aproximada de la señal de información se puede calcular cual es el valor aproximado de la SNR, cuando con el reformado de pulso se obtiene un menor valor de BER que con la corrección de errores. El ruido del canal está alrededor de 0.0018 y el umbral de recepción está en 0.0040. En dicho umbral ya se generan problemas en la recuperación de datos (a 5 metros), lo que implica que la amplitud de la señal es un poco superior a 0.0040. Entonces, para los resultados de la Tabla 3.3, si la amplitud de la señal es 0.0045 la SNR es igual a 2.5.

Con las **ganancias** de transmisión y recepción configuradas por defecto en la interfaz gráfica principal, no se pudo recuperar los datos a una distancia de **7 metros**, por esto se modifican las ganancias de transmisión y recepción. Mediante la interfaz gráfica principal se accede a la interfaz gráfica de configuración, en donde se modifica únicamente la ganancia de transmisión y la ganancia de recepción.

- La ganancia de transmisión pasa de -2 dB o 0 dB.
- La ganancia de recepción pasa de 20 dB a 30 dB.

Para analizar lo que ocurre cuando se aumenta la ganancia de transmisión y recepción, se realiza las pruebas de funcionamiento para una distancia de **5 metros** con las ganancias modificadas. Estos resultados se muestran en la Tabla 3.4, el ambiente de pruebas se mantiene igual al de la Figura 3.1, es decir un ambiente interior con línea de vista.

Tabla 3.4. Pruebas de funcionamiento a 5 metros, con línea de vista, ambiente interior y ganancias elevadas.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0	0/11632
		X	X		0	0/11632
	X			X	0	0/11632
	X		X		0	0/11632
QPSK		X		X	0	0/11632
		X	X		0	0/11632
	X			X	0	0/11632
	X		X		0	0/11632
16-QAM		X		X	0.034732	404/11632
		X	X		0.012895	150/11632
	X			X	0.001118	13/11632
	X		X		0.000430	5/11632
64-QAM		X		X	0.070925	825/11632
		X	X		0.060351	702/11632
	X			X	0.054677	636/11632
	X		X		0.028542	332/11632

En la Tabla 3.4 se observa que al elevar las ganancias de transmisión y recepción el valor del BER mejora en todas las combinaciones. Además, en 16-QAM y 64-QAM utilizar la corrección de errores resulta en un menor valor del BER que al utilizar reformado de pulso. Es decir que estos resultados cambian en comparación a los vistos en la Tabla 3.3, la razón es que, al elevar las ganancias de transmisión y recepción, el valor del SNR aumenta.

Cuando aumenta la SNR, la corrección de errores es más efectiva que el reformado de pulsos para reducir la tasa de bits errados.

En la Tabla 3.5 se muestran los resultados de las pruebas realizadas, con la ganancia de transmisión en 0 dB y la de recepción en 30 dB. Los equipos se colocan a una distancia de **7 metros** entre ellos. Se utiliza línea de vista directa y el ambiente interior de la Figura 3.1.

Tabla 3.5. Pruebas de funcionamiento a 7 metros, con línea de vista, ambiente interior y ganancias elevadas.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0.000258	3/11632
		X	X		0	0/11632
	X			X	0	0/11632
	X		X		0	0/11632
QPSK		X		X	0.001376	16/11632
		X	X		0.000086	1/11632
	X			X	0.000086	1/11632
	X		X		0	0/11632
16-QAM		X		X	0.032411	377/11632
		X	X		0.015303	178/11632
	X			X	0.001633	19/11632
	X		X		0.000774	9/11632
64-QAM		X		X	0.071871	836/11632
		X	X		0.063188	735/11632
	X			X	0.051668	601/11632
	X		X		0.033528	390/11632

En los resultados que se presentan en la Tabla 3.5 se observa que a medida que aumenta el esquema de mapeo, el valor del BER aumenta. En cada esquema de mapeo se obtiene el menor valor del BER, cuando se utiliza la corrección de errores y el reformado de pulso en conjunto. Mientras que el mayor valor de BER se obtiene cuando se utiliza el esquema de mapeo solo. En el mapeo 16-QAM y 64-QAM se obtiene mejores resultados cuando se utiliza la corrección de errores, que cuando se utiliza el reformado de pulso.

3.2. PRUEBAS DE TRANSMISIÓN CON LÍNEA DE VISTA (EXTERIORES)

En la Figura 3.2 se muestra un diagrama que representa el ambiente exterior en donde se realizan las pruebas de funcionamiento, este ambiente es un patio. En el diagrama todas las medidas están en centímetros, las cotas con verde son las medidas del ambiente exterior. La línea de vista y las posiciones de los equipos SDR Adalm Pluto se señalan con color azul. El SDR de transmisión (SDR Tx) se mantiene fijo y el SDR de recepción (SDR de Rx) se ubica primero en la posición P1 y luego en la posición P2.

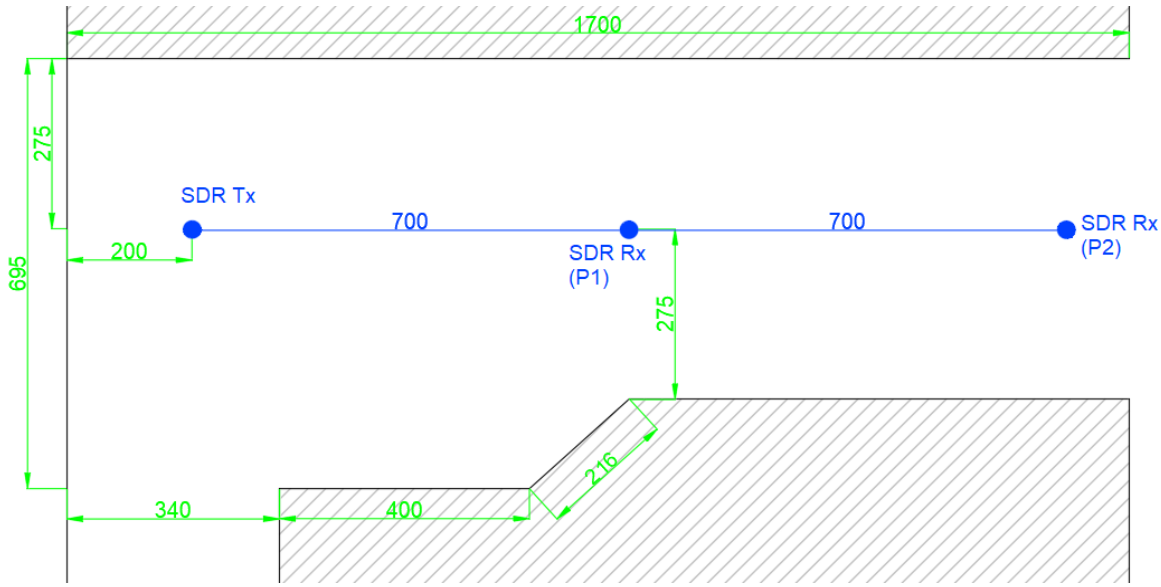


Figura 3.2. Representación del ambiente interior con línea de vista en donde se realizan las pruebas. Las distancias están en cm.

Para realizar las pruebas de funcionamiento los dispositivos SDR se colocan a una altura de 50 centímetros sobre el piso. Los principales parámetros de funcionamiento que se utilizan para la configuración de equipos son los siguientes.

- Ganancia de transmisión igual a **0 dB**.
- Ganancia de recepción igual a **30 dB**.
- Tamaño del bloque de datos igual a 800 000.
- Frecuencia central igual a 860 MHz.
- Frecuencia de muestreo en 2 MHz.

El archivo de texto que se utiliza para realizar las pruebas de funcionamiento es "*ArchivoTextoRT.txt*", este archivo contiene 1454 caracteres y el contenido del mismo se presentó en la sección anterior.

En la Tabla 3.6 se comparan los resultados obtenidos en un ambiente interior y exterior con los equipos ubicados a **7 metros**. Los resultados del ambiente interior se copian de la Tabla 3.5. Estos resultados se ubican en la Tabla 3.6 en la sección de BER en la columna **Interior**. Mientras que los resultados del ambiente exterior descrito en la Figura 3.2 se colocan en la columna **Exterior**. Los valores de BER se presentan como fracción, ya que así es sencillo observar y comparar los mismos.

Tabla 3.6. Pruebas de funcionamiento a 7 metros en ambiente interior y exterior.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER (en fracción)	
	ON	OFF	ON	OFF	Interior	Exterior
BPSK		X		X	3/11632	0/11632
		X	X		0/11632	0/11632
	X			X	0/11632	0/11632
	X		X		0/11632	0/11632
QPSK		X		X	16/11632	0/11632
		X	X		1/11632	0/11632
	X			X	1/11632	0/11632
	X		X		0/11632	0/11632
16-QAM		X		X	377/11632	384/11632
		X	X		178/11632	173/11632
	X			X	19/11632	11/11632
	X		X		9/11632	1/11632
64-QAM		X		X	836/11632	850/11632
		X	X		735/11632	697/11632
	X			X	601/11632	372/11632
	X		X		390/11632	167/11632

De los resultados obtenidos en la Tabla 3.6, en general existe una ligera mejora en el valor del BER cuando se trabaja en un ambiente exterior. Solamente con el mapeo 64-QAM se observa una diferencia más significativa cuando se utiliza corrección de errores y por ende cuando se utiliza corrección de errores y reformado de pulso en conjunto. Cuando se utiliza solamente el mapeo en 16-QAM y 64-QAM, sin una técnica adicional, el valor del BER en un ambiente exterior empeora. Debido a que, la diferencia no es significativa, este incremento puede deberse a la naturaleza aleatoria del canal inalámbrico, a las posiciones de los equipos SDR o incluso a la temperatura de los equipos de radio. Ya que al estar sometidos a un tiempo de funcionamiento prolongado y a la influencia del sol, los equipos elevan su temperatura.

Entonces, al trabajar en un ambiente exterior la BER mejora en comparación a un ambiente interior. Obteniendo una mayor mejora utilizando la corrección de errores y/o el reformado de pulso.

La siguiente prueba de transmisión se realiza en la posición P2, que se describe en la Figura 3.2. Los equipos SDR Adalm Pluto se colocan a 14 metros entre sí y el umbral de recepción se cambia de 0.0040 a **0.0030**, ya que con un umbral de 0.0040 los datos no se recuperan satisfactoriamente, lo que causa que no se pueda medir el BER. Además, esto quiere decir que la SNR ha disminuido respecto a las pruebas realizadas a 7 metros. En la

Tabla 3.7 se muestran los resultados obtenidos para los esquemas de mapeo BPSK y QPSK.

Tabla 3.7. Pruebas de funcionamiento a 14 metros, con línea de vista y ambiente exterior.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER	
	ON	OFF	ON	OFF	Valor	Fracción
BPSK		X		X	0.016506	192/11632
		X	X		0.007135	83/11632
	X			X	0.011606	135/11632
	X		X		0.000344	4/11632
QPSK		X		X	0.051582	600/11632
		X	X		0.017194	200/11632
	X			X	0.027768	323/11632
	X		X		0.012724	148/11632

A la distancia de 14 metros entre transmisor y receptor, con ganancia de transmisión 0 dB, ganancia de recepción de 30 dB y con el umbral de recepción en 0.0030, no siempre se pueden recuperar los datos del canal inalámbrico. Es decir que para esta configuración de los equipos SDR, el límite de distancia entre equipos es de 14 metros. Para superar ese límite se puede aumentar la ganancia de recepción, ya que la de transmisión ya está en el límite.

En los resultados de la Tabla 3.7 se comprueba nuevamente que el menor valor de BER se obtiene cuando se utiliza en conjunto el reformado de pulso y la corrección de errores. Mientras que el peor mayor valor del BER se obtiene cuando se utiliza solo el esquema de modulación digital respectivo. El resultado que cambia respecto a la Tabla 3.6, es que con el reformado de pulso se obtiene un menor valor de BER que con la corrección de errores.

Esto confirma que cuando se tiene un nivel bajo de la SNR, el reformado de pulso resulta ser más efectivo para reducir el BER que la corrección de errores.

Con el umbral de recepción 0.0040 no se pudieron recuperar los datos satisfactoriamente. Pero si se supera el umbral de recepción, esto implica que la amplitud de la señal de datos puede estar alrededor de los 0.0045. Tomando en cuenta el nivel de ruido que está alrededor de 0.018, la SNR sería nuevamente 2.5.

3.3. PRUEBAS DE TRANSMISIÓN SIN LÍNEA DE VISTA

En la Figura 3.3 se muestra un diagrama, que representa el ambiente en donde se realizan las pruebas de funcionamiento, cabe mencionar que es un ambiente interior. En el diagrama todas las medidas están en centímetros, las cotas con verde son las medidas del ambiente interior. Se señala con color rojo una puerta que tiene un grosor de 4 centímetros, un ancho de 75 centímetros y que sirve para evaluar el prototipo con y sin línea de vista. La línea azul representa la línea de vista cuando se abre la puerta.

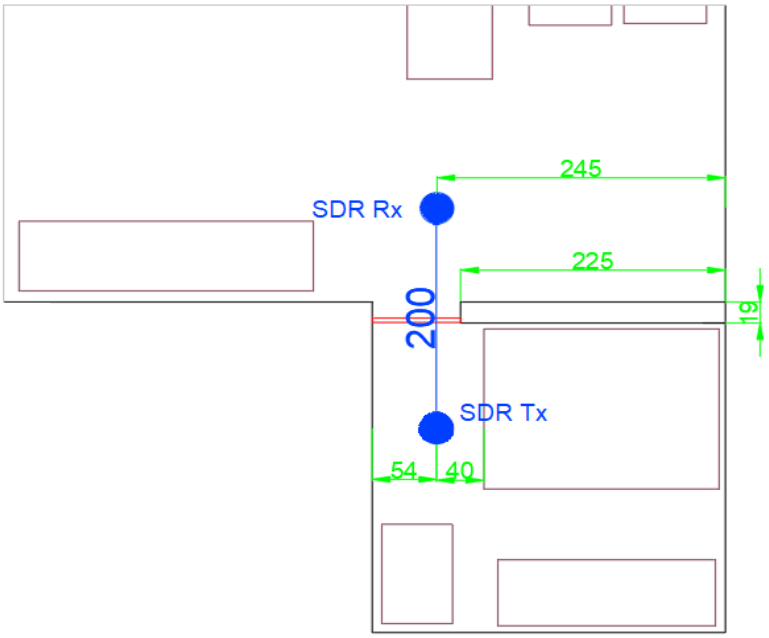


Figura 3.3. Representación del ambiente interior sin línea de vista en donde se realizan las pruebas.

Para realizar las pruebas de funcionamiento los dispositivos SDR se colocan a una altura de 50 centímetros y ambos equipos se mantienen en una posición fija a lo largo de las pruebas. Los principales parámetros de funcionamiento que se utilizan para la configuración de equipos son los siguientes.

- Ganancia de transmisión igual a -2 dB.
- Ganancia de recepción igual a **25 dB**.
- Tamaño del bloque de datos igual a 800 000.
- Frecuencia central igual a 860 MHz.
- Frecuencia de muestreo en 2 MHz.

El archivo de texto que se utiliza para realizar las pruebas de funcionamiento es "*ArchivoTextoRT.txt*", este archivo contiene 1454 caracteres y el contenido del mismo se presentó en la primera sección de este capítulo. Además, el umbral de recepción de la interfaz gráfica de recepción se cambia de 0.0040 a **0.0030**.

En la Tabla 3.8 se muestran los resultados de las pruebas realizadas, en donde los equipos se colocan a una distancia de **2 metros**. Para las diferentes combinaciones que se pueden lograr se presentan los valores de BER que se obtienen con y sin línea de vista. Para obtener los valores con línea de vista se abrió la puerta que se señala con rojo en la Figura 3.3 y para obtener los valores de BER son línea de vista se cerró la puerta. Los valores de BER se presentan como fracción, ya que así es sencillo observar y comparar los valores obtenidos.

Tabla 3.8. Pruebas de funcionamiento a 2 metros con y sin línea de vista.

Esquema de Mapeo	Corrección de errores		Reformado de pulso		BER (en fracción)	
	ON	OFF	ON	OFF	Con línea de vista	Sin línea de vista
BPSK		X		X	0/11632	8/11632
		X	X		0/11632	2/11632
	X			X	0/11632	2/11632
	X		X		0/11632	0/11632
QPSK		X		X	1/11632	11/11632
		X	X		0/11632	2/11632
	X			X	0/11632	2/11632
	X		X		0/11632	1/11632
16-QAM		X		X	625/11632	1522/11632
		X	X		181/11632	568/11632
	X			X	38/11632	682/11632
	X		X		13/11632	157/11632
64-QAM		X		X	1130/11632	2178/11632
		X	X		962/11632	1610/11632
	X			X	851/11632	1742/11632
	X		X		728/11632	1498/11632

De los resultados obtenidos en la Tabla 3.8 en general la cantidad de errores aumenta cuando no se tiene línea de vista, esto se evidencia mayormente en los esquemas de mapeo 16-QAM y 64-QAM. Con o sin línea de vista en cada esquema de mapeo se obtiene la menor BER cuando se utiliza reformado de pulso y corrección de errores en conjunto. Mientras que el mayor valor de BER se obtiene cuando no se utiliza ni reformado de pulso, ni corrección de errores.

En los esquemas de mapeo BPSK y QPSK, utilizar solo el reformado de pulso o solo la corrección de errores da igual resultado en el valor del BER. Esto se debe principalmente a que estos esquemas de mapeo de por sí, presentan una baja tasa de bits errados y por lo tanto no se puede evaluar el efecto del reformado o de la corrección de errores por separado. Cuando se tiene línea de vista y se utiliza el mapeo 16-QAM y 64-QAM, la tasa

de bits errados mejora en mayor medida cuando se utiliza corrección de errores, que cuando se utiliza el reformado de pulso. Mientras que sin línea de vista resulta en un mejor valor de BER cuando se utiliza reformado de pulso, que cuando se utiliza corrección de errores. Cabe mencionar que sin línea de vista la diferencia entre la corrección de errores y el reformado de pulso no es muy significativa, en cuanto al valor del BER.

Entonces, cuando no se tiene línea de vista, la técnica de reformado de pulso es más efectiva para reducir la tasa de bits errados, que la corrección de errores a nivel de bit.

3.4. EFECTO DE LA CORRECCIÓN DEL CFO Y CORRECCIÓN EN FASE SOBRE EL BER

En esta sección se evalúa la influencia real, sobre la tasa de bits errados, de la corrección del CFO y la corrección en fase, realizada mediante el preámbulo 802.11a. Debido a que, el preámbulo no depende de las técnicas de mapeo, basta con probar solamente con una opción de mapeo, en este caso se elige 16-QAM. Está técnica permite obtener un BER considerable a corta distancia.

Para realizar las pruebas se trabaja en un ambiente interior, con línea de vista directa y a una distancia de un metro entre el equipo de transmisión y el equipo de recepción. Los equipos se encuentran sobre un escritorio a una altura de 78 centímetros. Los principales parámetros de funcionamiento que se utilizan para la configuración de equipos son los siguientes.

- Ganancia de transmisión igual a -2 dB.
- Ganancia de recepción igual a 20 dB.
- Tamaño del bloque de datos igual a 800 000.
- Frecuencia central igual a 860 MHz.
- Frecuencia de muestreo en 2 MHz.

El archivo de texto que se utiliza para realizar las pruebas de funcionamiento es "*ArchivoTextoRT.txt*", este archivo contiene 1454 caracteres y el contenido del mismo se presentó en la primera sección de este capítulo. El umbral de recepción de la interfaz gráfica del receptor se mantiene en 0.0040.

En la Tabla 3.9 se muestran los resultados del BER de las pruebas realizadas, utilizando mapeo 16-QAM, sin reformado de pulsos, corrección de errores (activada/desactivada), corrección de fase (activada/desactivada) y corrección del CFO (activada/desactivada). En

el prototipo de comunicación inalámbrica tanto la corrección del CFO como la corrección de fase se implementan en la parte de recepción. El prototipo no cuenta con un botón para activar o desactivar estas correcciones, por lo que se debió hacer cambios directamente en el código.

Tabla 3.9. Pruebas de funcionamiento a 1 metro, con 16-QAM y sin reformado de pulso.

Mapeo 16-QAM						
Corrección de errores		Corrección de fase		Corrección del CFO		BER (en fracción)
ON	OFF	ON	OFF	ON	OFF	
	X	X		X		466/11632
	X	X			X	1572/11632
	X		X	X		477/11632
X		X		X		9/11632
X		X			X	651/11632
X			X	X		13/11632

De los resultados obtenidos en la Tabla 3.9 se puede decir que, la cantidade de errores aumenta cuando no se utiliza la corrección en frecuencia (CFO). Esto independientemente que se utilice corrección de errores o no. El BER aumenta significativamen, por lo que la correción del CFO es de suma importancia en el prototipo. En el caso de la corrección en fase, cuando se desactiva esta no se ve un incremento notable en el BER, lo que significa que en el prototipo no influye mucho la etapa de corrección en fase. Esto puede deberse a que en OFDM las portadoras piloto corrigen los desplazamientos en fase en cada símbolo.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

En las pruebas realizadas se obtuvo que los esquemas de modulación digital, más robustos frente a errores son BPSK y QPSK. Los valores de BER de estos esquemas, en casi todas las pruebas se mantienen en cero o en valores cercanos a cero, a excepción de los resultados presentados en la Tabla 3.7. En esta tabla se almacenaron los valores de BER, para BPSK y QPSK, cuando se realizó las pruebas de funcionamiento a una distancia de 14 metros entre transmisor y receptor. Entonces, en los valores de la Tabla 3.7 se puede observar claramente que utilizar el esquema de mapeo BPSK resulta en una tasa de bits errados más baja que utilizar QPSK. Siendo que, sin utilizar alguna técnica adicional, con la modulación digital BPSK se obtiene una tasa de bits errados igual a 192/11632 y para QPSK se obtiene un valor de BER igual a 600/11632. Se utilizan los valores de BER en fracción, ya que resulta sencilla la comparación de resultados.

Cuando se utiliza el mapeo 16-QAM la cantidad de errores se eleva significativamente respecto al mapeo QPSK, esto se observa en todas las tablas de resultados, a excepción de la Tabla 3.8, que no considera el mapeo 16-QAM ni el 64-QAM. Por ejemplo, en la Tabla 3.2 utilizando solo el mapeo QPSK se obtiene una BER de 4/11632, mientras que si se utiliza 16-QAM el BER se eleva a 883/11632. En el caso de utilizar la modulación digital 64-QAM el BER se vuelve a elevar significativamente respecto al mapeo 16-QAM, esto ocurre en todos los casos. Siguiendo los resultados de la Tabla 3.2 el BER en el mapeo 64-QAM se eleva a 1224/11632. Entonces, se comprueba la parte teórica vista en la sección 2.3.4.1, de que a medida que aumenta la cantidad de puntos en las constelaciones de los esquemas de mapeo, el BER también se eleva, esto se debe principalmente a la dispersión de los datos en la constelación.

En cualquier escenario el utilizar la técnica de reformado de pulso, permite reducir la cantidad de bits errados, respecto a utilizar un tipo de mapeo solo. Esto se comprueba en todas las tablas de resultados que se obtuvieron en el capítulo 3, las cuales corroboran la parte teórica aborda en la sección 2.3.5.1. Por ejemplo, en la Tabla 3.1, utilizando solamente el esquema de mapeo 16-QAM se obtuvo una BER igual a 262/11632 y si se utiliza en conjunto 16-QAM con la técnica de reformado de pulso se obtiene una BER igual a 134/11632.

Independientemente del escenario de pruebas, la corrección de errores a nivel de bit permite mejorar la tasa de bits errados. En todas las tablas de resultados del capítulo 3, si

se utiliza la corrección de errores, el valor del BER mejora respecto a solamente utilizar la modulación digital. Por ejemplo, en la Tabla 3.1, en el esquema de mapeo 64-QAM se obtuvo una BER igual a 840/11632 y utilizando la corrección de errores el BER mejoró a un valor de 493/11632. Es decir que, se comprueba la parte teórica abordada en la sección 2.3.3.1.

La corrección de errores a nivel de bit resulta mejor para reducir la cantidad de bits errados en comparación al reformado de pulso, siempre y cuando se tenga una relación señal al ruido (SNR) elevada. Entendiendo como SNR elevada, cuando en el prototipo de comunicación inalámbrica la amplitud de la señal de información, supera ampliamente el umbral de recepción de 0.0040 y por ende también el promedio de amplitud de ruido del canal (0.0018). En las pruebas realizadas con línea de vista, se obtiene un mejor resultado en el BER utilizando corrección de errores que reformado de pulso, cuando la ganancia de transmisión es -2 dB y la de recepción es 20 dB, la distancia entre equipos es de 1 metro (Tabla 3.1) y a 3 metros (Tabla 3.2). El resultado se repite cuando la ganancia de transmisión se eleva a 0 dB, la de recepción se eleva a 30 dB y se ubica los equipos a 5 metros (Tabla 3.4) y 7 metros (Tabla 3.5 y 3.6).

La técnica de reformado de pulso resulta más efectiva para reducir el BER que la corrección de errores, cuando la SNR disminuye. El SNR aproximado al que se obtuvo estos resultados es 2.5, considerando que el nivel de ruido es 0.0018 y que el nivel de la señal de información está alrededor de 0.0045, ya que supera ligeramente el umbral de recepción de 0.0040. En las pruebas realizadas con línea de vista, se obtiene un mejor valor de BER con el reformado de pulsos que con la corrección de errores, con la ganancia transmisión igual a -2 dB y de recepción igual 20 dB, cuando la distancia entre equipos es de 5 metros (Tabla 3.5). El resultado se repite con una ganancia de transmisión de 0 dB y de recepción de 30 dB, cuando la distancia entre equipos es de 14 metros (Tabla 3.7).

En un ambiente interior sin línea de vista, el reformado de pulso permite obtener una mejor BER que la corrección de errores. Por ejemplo, en la Tabla 3.8 al utilizar 16-QAM con reformado de pulso y sin línea de vista se obtiene una BER de 682/11632 y con corrección de errores se obtiene una BER igual a 568/11632. Para el mismo ambiente interior, pero con línea de vista, se obtiene un mejor valor de BER con la corrección de errores que con el reformado de pulsos. Siguiendo el ejemplo de la Tabla 3.8 en 16-QAM con línea de vista, se obtiene con reformado de pulso una BER de 181/11632 y con corrección de errores una BER de 38/11632.

Para un mismo esquema de mapeo, el menor valor de BER se obtiene cuando se utiliza en conjunto la técnica de reformado de pulso y la corrección de errores. Esto se puede ver en todas las tablas de resultados del capítulo 3. Por ejemplo, en la Tabla 3.5 utilizando el mapeo 16-QAM se obtiene una BER igual a 377/11632 y utilizando 16-QAM en conjunto con el reformado de pulso y la corrección de errores se tiene una BER de 9/11632.

El utilizar la corrección del CFO con el preámbulo 802.11a permite reducir significativamente el valor del BER y con esto aumentar el rendimiento de un sistema de comunicaciones. En la Tabla 3.9 se obtuvo que para el mapeo 16-QAM, sin utilizar reformado de pulso ni corrección de errores, cuando se utiliza la corrección del CFO el valor del BER es de 466/11632 y cuando no se la utiliza el BER se eleva a 1572/11632.

4.2. RECOMENDACIONES

Si se cambia el tamaño de bloque de datos a transmitir y recibir, hay que considerar el tamaño del archivo de texto a transmitir. En la interfaz de configuración se puede elegir el tamaño de bloque, si se elige 100000 y se para la transmisión se utiliza el archivo de texto *ArchivoTextoRT.txt* con 1454 caracteres, no se puede utilizar BPSK en conjunto con el reformado de pulso y la corrección de errores. Esto se debe a que el tamaño de los datos procesados es de 186432, el resultado se obtiene de aplicar la ecuación 2.3 con los parámetros que se utilizan en los algoritmos y técnicas del prototipo de comunicaciones. Si se requiere transmitir un archivo de texto más largo hay que tener en cuenta el tamaño de bloque elegido y la ecuación 2.3.

Para utilizar el prototipo con un solo computador es necesario abrir dos veces Matlab R2021a, uno se destina a transmisión y otro a recepción. El primer SDR en conectarse al computador debe ser el de transmisión y el segundo el de recepción. Luego en el primer Matlab se ejecuta la interfaz gráfica principal, se accede a la etapa de transmisión y se empieza a transmitir los datos al canal. Luego en el segundo Matlab se ejecuta la interfaz gráfica principal, si se puede configurar el equipo SDR se abre la interfaz de recepción, caso contrario se abre la interfaz de configuración y se cambia en ID de la radio. Una vez configurado el equipo ya se pueden recibir los datos.

Se recomienda utilizar el prototipo con Matlab R2021a. En caso de que se requiera utilizar el prototipo en otra versión de Matlab (no inferior a Matlab R2017a), antes de correr las interfaces gráficas, es recomendable primero probar los scripts creados en las diferentes etapas. Con esto el usuario podrá comprobar que todo esté correcto o en su defecto,

corregir y actualizar los scripts, para luego hacer lo propio con interfaces gráficas del prototipo de comunicación inalámbrica.

En el caso de que el usuario desee utilizar antenas distintas a las que se incluyen con los equipos SDR Adalm Pluto, se debe tener en cuenta sus rangos de funcionamiento y en base a los mismos configurar los objetos de transmisión y recepción, de ser necesario.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] R. Teja, “Wireless Communication: Introduction, Types and Applications”. 3 de abril de 2021. Accedido: 15 de diciembre de 2021. [En línea]. Disponible en: https://www.electronicshub.org/wireless-communication-introduction-types-applications/

[2] Yong Soo Choo, J. Kim, Won Young Yang, y C.-G. Kang, *MIMO-OFDM wireless communications with MATLAB*. Singapore: John Wiley & Sons, 2010.

[3] A. Vega, “Radio Definido por Software”. 16 de mayo de 2020. Accedido: 14 de enero de 2022. [En línea]. Disponible en: https://telecomunicaciones.edu.ec/repositorio/articulos-blog/redes/radio-definido-por-software

[4] T. Collins, R. Getz, D. Pu, y A. Wyglinski, *Software-Defined Radio for Engineers*. U.S.A.: Analog Devices, 2018. Accedido: 30 de septiembre de 2020. [En línea]. Disponible en: https://www.analog.com/media/en/training-seminars/design-handbooks/Software-Defined-Radio-for-Engineers-2018/SDR4Engineers.pdf

[5] Analog Devices, “ADALM-PLUTO SDR Active Learning Module”. 2021. Accedido: 13 de agosto de 2021. [En línea]. Disponible en: https://www.analog.com/media/en/news-marketing-collateral/product-highlight/ADALM-PLUTO-Product-Highlight.pdf

[6] A. Bensky, *Short-range Wireless Communication*, 3era ed. Reino Unido: Elsevier Inc., 2019.

[7] C. Vargas, W. Lopez, y C. Rocha, “Sistemas de Comunicación Inalámbrica MIMO - OFDM”. junio de 2007. Accedido: 18 de octubre de 2021. [En línea]. Disponible en: http://www.scielo.org.bo/pdf/ran/v3n4/v3n4a09.pdf

[8] L. Cheuk, “Method of Synchronization using IEEE 802.1 la OFDM Training Structure for Indoor Wireless Applications”, SIMON FRASER, California, 2004.

[9] MathWorks, “Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio”. Accedido: 25 de agosto de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/supportpkg/plutoradio/index.html?s_cid=doc_ftr

[10] MathWorks, “comm.SDRTxPluto”. 2021. Accedido: 25 de agosto de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/supportpkg/plutoradio/ref/comm.sdrtxpluto-system-object.html

[11] MathWorks, “comm.SDRRxPluto”. 2021. Accedido: 25 de agosto de 2021. [En línea]. Disponible en:

https://www.mathworks.com/help/supportpkg/plutoradio/ref/comm.sdrrxpluto-system-object.html

[12] Y. Fernandez, “Tipos de USB: estándares, conectores y características de cada uno”. 21 de abril de 2021. Accedido: 31 de agosto de 2021. [En línea]. Disponible en: https://www.xataka.com/basics/tipos-usb-estandares-conectores-caracteristicas-cada-uno

[13] MathWorks, “Unicode and ASCII Values”. 2021. Accedido: 14 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/matlab_prog/unicode-and-ascii-values.html

[14] “ASCII table , ascii codes”. Accedido: 14 de septiembre de 2021. [En línea]. Disponible en: https://theasciicode.com.ar/

[15] MathWorks, “Numeric Types”. 2021. Accedido: 16 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/numeric-types.html

[16] O. Quilumbango, “MATERIAL DIDÁCTICO PARA EL ANÁLISIS Y SIMULACIÓN DEL DESEMPEÑO DE LA CAPA FÍSICA DE WIMAX/IEEE 802.16, USANDO LA INTERFAZ GRÁFICA DE MATLAB (GUI)”, Escuela Politécnica Nacional, Quito-Ecuador, 2011.

[17] C. Espín, “ANÁLISIS, SIMULACIÓN E IMPLEMENTACIÓN CON CARÁCTER DIDÁCTICO DEL MÉTODO DE DETECCIÓN Y CORRECCIÓN DE ERRORES CONVOLUCIONAL Y SU DECODIFICACIÓN USANDO EL ALGORITMO DE VITERVI”, Escuela Politécnica Nacional, Quito-Ecuador, 2017.

[18] UNAM, “Estándar IEEE 802.11 WiFi”. Accedido: 16 de enero de 2022. [En línea]. Disponible en: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/164/A6.pdf?sequence=6

[19] MathWorks, “poly2trellis”. 2021. Accedido: 22 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/poly2trellis.html

[20] MathWorks, “convenc”. 2021. Accedido: 23 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/convenc.html#d123e28769

[21] S. Jiménez y D. Panchi, “DISEÑO E IMPLEMENTACIÓN DE UN MODULADOR Y UN DEMODULADOR N-QAM EMPLEANDO XILINX ISE, SYSTEM GENERATOR Y SIMULINK SOBRE UNA TARJETA DE ENTRENAMIENTO BASADA EN UN FPGA DE XILINX”, Escuela Politécnica Nacional, Quito-Ecuador, 2011.

[22] MathWorks, “pskmod”. 2021. Accedido: 25 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/pskmod.html

[23] MathWorks, “qammod”. 2021. Accedido: 25 de septiembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/qammod.html

[24] J. Rueda, “ESTUDIO COMPARATIVO DE LAS SECUENCIAS DE SINCRONIZACIÓN (BARKER, GOLAY Y ZADOFF-CHU) EMPLEANDO EQUIPOS DE RADIO DEFINIDA POR SOFTWARE”, Escuela Politécnica Nacional, Quito-Ecuador, 2021.

[25] Mathuranathan, “Inter-symbol interference & pulse shaping”. 26 de septiembre de 2021. Accedido: 7 de octubre de 2021. [En línea]. Disponible en: https://www.gaussianwaves.com/2018/09/introduction-to-intersymbol-interference-isi/

[26] Mathuranathan, “Sinc pulse shaping”. 5 de octubre de 2018. Accedido: 7 de octubre de 2021. [En línea]. Disponible en: https://www.gaussianwaves.com/2018/10/sinc-pulse-shaping/

[27] Mathuranathan, “Understanding Gibbs Phenomenon in signal processing”. 6 de abril de 2010. Accedido: 7 de octubre de 2021. [En línea]. Disponible en: https://www.gaussianwaves.com/2010/04/gibbs-phenomena-a-demonstration/

[28] Mathuranathan, “Raised cosine pulse shaping”. 9 de octubre de 2018. Accedido: 7 de octubre de 2021. [En línea]. Disponible en: https://www.gaussianwaves.com/2018/10/raised-cosine-pulse-shaping/

[29] Mathuranathan, “Square-root raised-cosine pulse shaping”. 15 de octubre de 2018. Accedido: 11 de octubre de 2021. [En línea]. Disponible en: https://www.gaussianwaves.com/2018/10/square-root-raised-cosine-pulse-shaping/

[30] MathWorks, “Raised Cosine Filtering”. 2021. Accedido: 11 de octubre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ug/raised-cosine-filtering.html#responsive_offcanvas

[31] MathWorks, “comm.RaisedCosineTransmitFilter”. 2021. Accedido: 13 de octubre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/comm.raisedcosinetransmitfilter-system-object.html#responsive_offcanvas

[32] MathWorks, “comm.RaisedCosineReceiveFilter”. 2021. Accedido: 23 de noviembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/comm.raisedcosinereceivefilter-system-object.html

[33] MathWorks, “Design and evaluate OFDM waveforms in a wireless system”. 2021. Accedido: 15 de octubre de 2021. [En línea]. Disponible en: https://www.mathworks.com/discovery/ofdm.html

[34] M. Gordón y F. Hidalgo, “DESARROLLO DE UN PROTOTIPO DE COMUNICACIÓN INALÁMBRICA DE USO DIDÁCTICO UTILIZANDO EQUIPOS SDR, INCLUYENDO LA TÉCNICA OFDM Y EL SERVICIO DE CONFIDENCIALIDAD”, Escuela Politécnica Nacional, Quito-Ecuador, 2020.

[35] MathWorks, “ifft (Inverse fast Fourier transform)”. 2021. Accedido: 22 de octubre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/ifft.html

[36] H. Chen, L. Yuan, J. Gong, y Y. Huang, “Various Channel Models in Wireless Communication”. International Conference on Computer Systems, Electronics and Control (ICCSEC), 2017.

[37] MathWorks, “comm.PhaseFrequencyOffset”. 2021. Accedido: 7 de diciembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/comm.phasefrequencyoffset-system-object.html

[38] MathWorks, “xcorr”. 2021. Accedido: 12 de diciembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/xcorr.html

[39] MathWorks, “fft (Fast Fourier transform)”. 2021. Accedido: 13 de diciembre de 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/fft.html

[40] MathWorks, “Interpolación de datos 1-D interp1”. 2021. [En línea]. Disponible en: https://www.mathworks.com/help/matlab/ref/interp1.html

[41] MathWorks, “pskdemod”. 2021. Accedido: 3 de enero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/pskdemod.html

[42] MathWorks, “qamdemod”. 2021. Accedido: 4 de enero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/qamdemod.html

[43] MathWorks, “vitdec”. 2021. Accedido: 8 de enero de 2022. [En línea]. Disponible en: https://www.mathworks.com/help/comm/ref/vitdec.html

ANEXOS

ANEXO A. Manual de usuario del prototipo desarrollado

ANEXO A

Manual de usuario del prototipo desarrollado

MANUAL DE USUARIO

1. INTRODUCCIÓN

El prototipo de comunicación inalámbrica cuenta con cuatro interfaces gráficas: una principal, una de configuración, una de transmisión y otra de recepción. En un ANEXO DIGITAL se incluyen los archivos que conforman y que permiten utilizar el prototipo de comunicación inalámbrica. Los archivos *main.m* y *main.fig* corresponden a la interfaz gráfica principal. Los archivos *configurarSDR.m* y *configurarSDR.fig* corresponden a la interfaz gráfica de configuración. Los archivos *txRealTime.m* y *txRealTime.fig* corresponden a la interfaz gráfica de transmisión en tiempo real. Los archivos *rxRealtime.m* y *rxRealtime.fig* corresponde a la interfaz gráfica de recepción en tiempo real. Además, *ArchivoTextoRT.txt* es el archivo de texto con el que se ha venido trabajando como ejemplo a lo largo del documento escrito del trabajo de titulación.

Para utilizar el prototipo de comunicaciones se debe tener los siguientes elementos:

- 2 computadores con Matlab R2021a y el paquete de comunicaciones "*Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio"* instalados. La instalación del paquete de comunicaciones se detalló en la sección 2.1.1.1 de este documento.
- 2 equipos SDR Adalm Pluto con al menos una antena, que incluye el kit del equipo, conectada. En el equipo de transmisión se debe conectar la antena en el conector señalado con *Tx*. En el equipo de recepción se debe conectar la antena en el conector señalado con *Rx*. Si se conectan las dos antenas se puede usar como transmisor o como receptor cualquiera de los dos equipos.
- El ANEXO DIGITAL con los archivos mencionados previamente.

2. INICIALIZACIÓN Y CONFIGURACIÓN DEL PROTOTIPO

Para utilizar el prototipo de comunicación inalámbrica, los primeros pasos son **comunes** para el transmisor y el receptor, estos pasos se detallan a continuación:

1) Abrir Matlab R2021a y ubicarse en el directorio del ANEXO DIGITAL.
2) Conectar el equipo SDR al computador, mediante la interfaz USB.
3) Abrir el script *main.m* y ejecutar el mismo con el botón *Run*, para abrir la interfaz gráfica principal. También se puede ejecutar mediante el comando *main()* en la ventana de comandos.

Cuando se ejecuta la interfaz principal el dispositivo se configura con los parámetros descritos en la Tabla 2.2 y en la Tabla 2.3 de este trabajo de titulación. En la Figura 6.1 se muestra la interfaz gráfica principal con los objetos de transmisión y recepción configurados.

Figura 6.1. Interfaz gráfica principal con un SDR conectado y configurado.

Además, en la ventana de comandos se obtienen las propiedades configuradas en el objeto de transmisión llamado *txAdalm* y en el objeto de recepción *rxAdalm* (ver Figura 6.2).

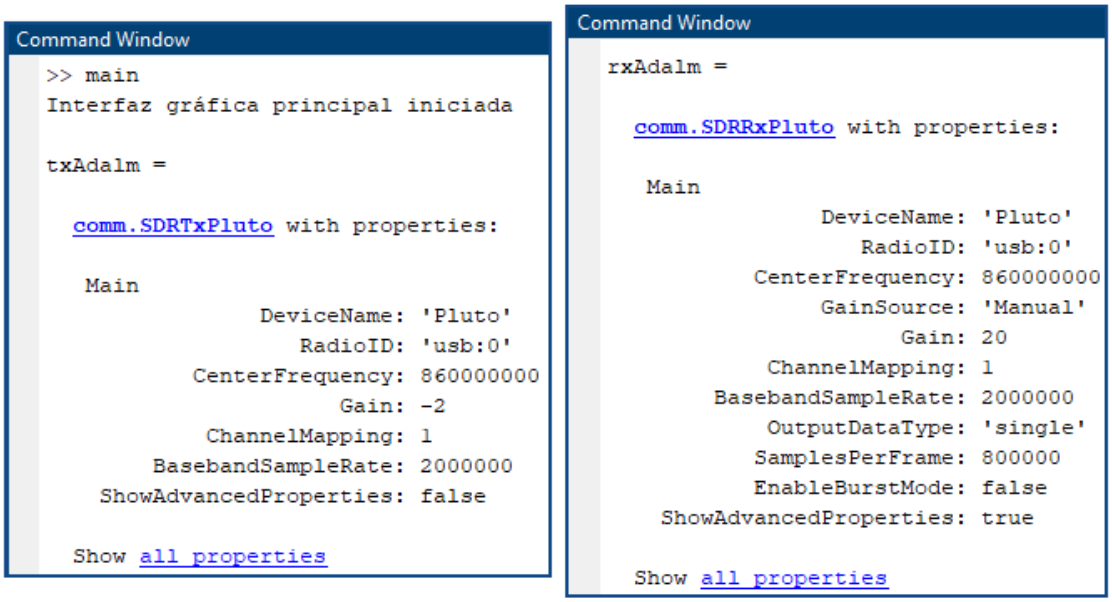


Figura 6.2. Valores configurados en los objetos de transmisión y recepción al ejecutar la interfaz gráfica principal.

En este punto ya se puede acceder a la interfaz gráfica de transmisión o a la interfaz gráfica de recepción. Además, en caso de ser necesario se puede cambiar algunos parámetros de configuración con la interfaz gráfica de **configuración**, para esto se siguen los siguientes pasos:

1) Presionar el botón **Configurar SDR** en la interfaz gráfica principal (ver Figura 6.1). Con esto se abre la interfaz gráfica que se muestra en la Figura 6.3.

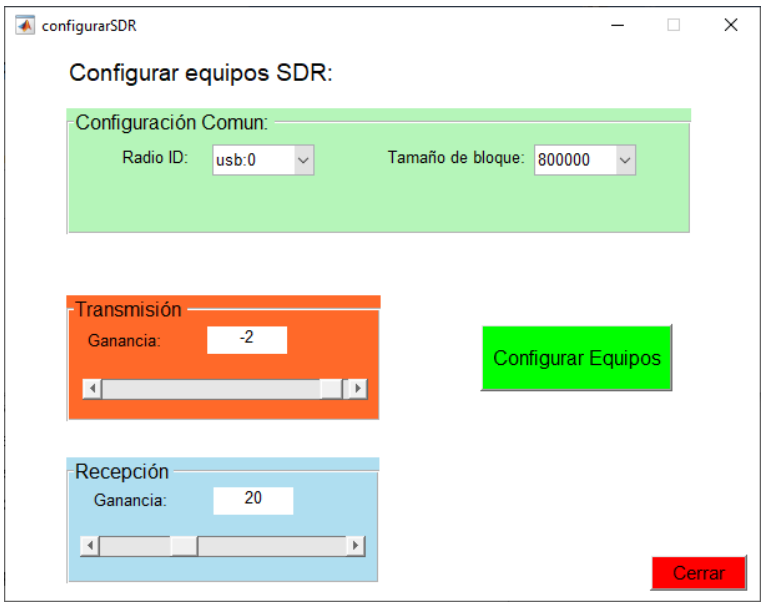


Figura 6.3. Interfaz gráfica de configuración con los parámetros iniciales.

2) Seleccionar el tamaño de bloque, la ganancia de transmisión y la ganancia de recepción según corresponda. La identificación de la radio (Radio ID) se cambia cuando se tiene dos radios conectadas al computador y se requiere utilizar la segunda radio que se conectó.
3) Presionar el botón **Configurar Equipos** para aplicar la configuración que se ha seleccionado. En la Figura 6.4 se muestra la interfaz gráfica de configuración con los valores iniciales modificados.

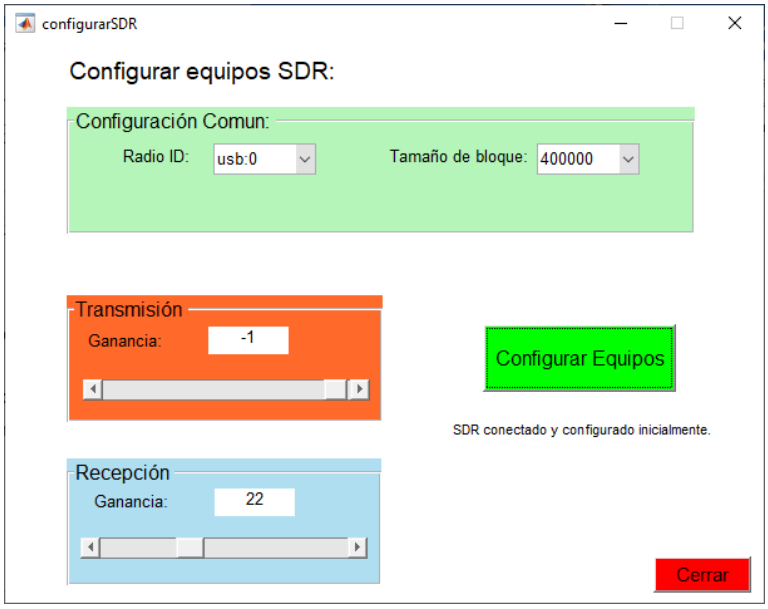


Figura 6.4. Interfaz gráfica de configuración con los parámetros de configuración cambiados.

4) Presionar el botón **Cerrar** para cerrar la interfaz gráfica de configuración.

3. TRANSMISIÓN DE DATOS

Una vez que se ha configurado el equipo SDR Adalm Pluto se puede acceder a la interfaz gráfica de transmisión para procesar y transmitir los datos de un archivo de texto al canal inalámbrico. Los pasos a seguir se detallan a continuación.

1) Presionar el botón **Transmitir en tiempo real** de la interfaz gráfica principal (ver Figura 6.1), para acceder a la interfaz gráfica de transmisión en tiempo real. En la Figura 6.5 se muestra la interfaz de transmisión con los valores por defecto que se configuran al abrir la misma.

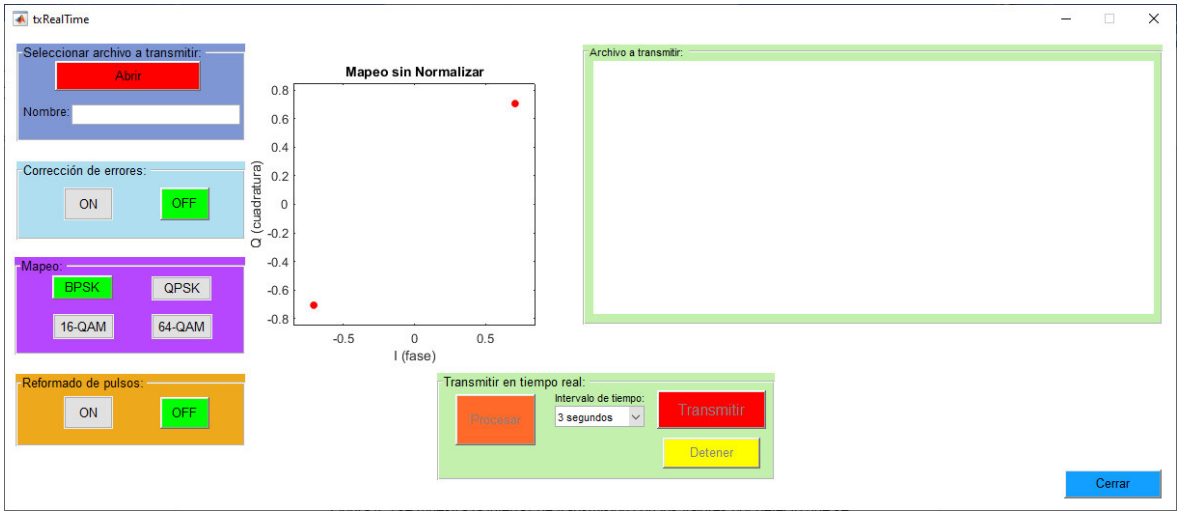


Figura 6.5. Interfaz gráfica de transmisión en tiempo real, con los parámetros por defecto.

2) En la interfaz gráfica de transmisión, presionar el botón **Abrir** para seleccionar un archivo de texto *ArchivoTextoRT.txt* en la carpeta actual, mediante un cuadro de diálogo (ver Figura 6.6).

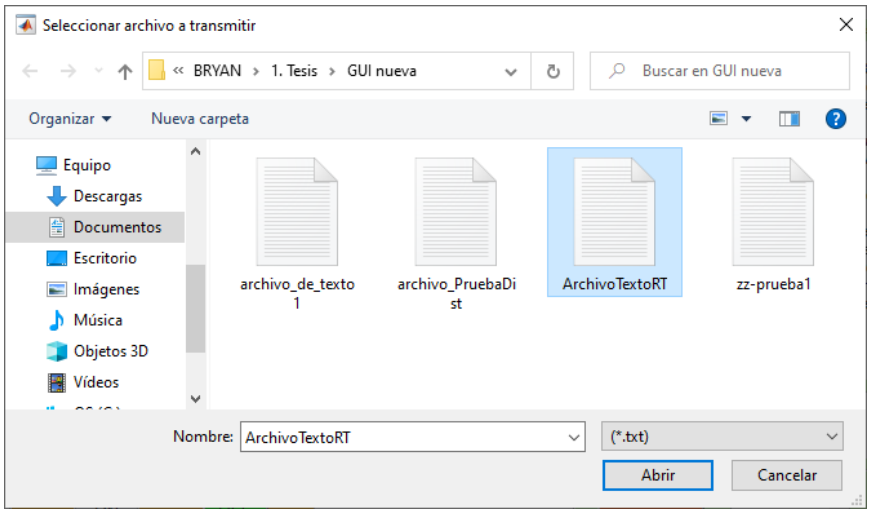


Figura 6.6. Cuadro de diálogo con los archivos de texto de la carpeta actual y con el archivo *ArchivoTextoRT.txt* seleccionado.

3) En el cuadro de diálogo de la Figura 6.6 presionar el botón **Abrir**.

4) En la interfaz gráfica de transmisión seleccionar y/o activar los algoritmos y técnicas que se quieren utilizar. En la Figura 6.7 se muestra la interfaz gráfica de transmisión con el archivo de texto *ArchivoTextoRT.txt* abierto, con la corrección de errores activada, con mapeo 64-QAM seleccionado, con el reformado de pulso desactivado y con un intervalo de tiempo entre transmisiones de 4 segundos.

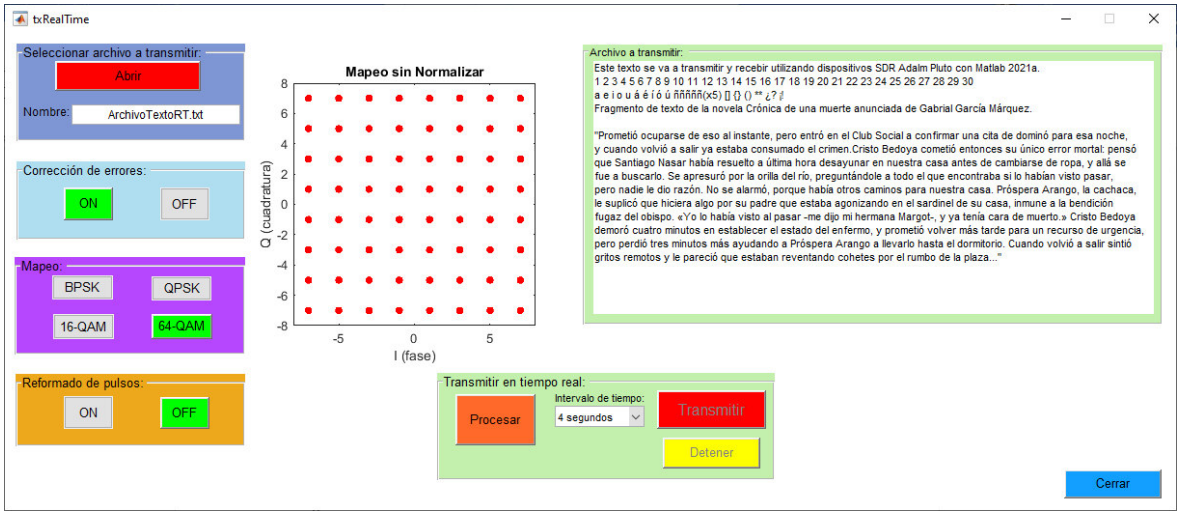


Figura 6.7. Interfaz gráfica de transmisión en tiempo real, con un archivo de texto abierto.

5) Presionar el botón **Procesar** para preparar los datos con las técnicas seleccionadas, para ser transmitidos. En la Figura 6.8 se muestra la interfaz de transmisión con los datos procesados y el botón **Transmitir** habilitado. En la ventana de comandos de Matlab se muestran mensajes de información (ver Figura 6.9), para que el usuario pueda ver las etapas se han utilizado.

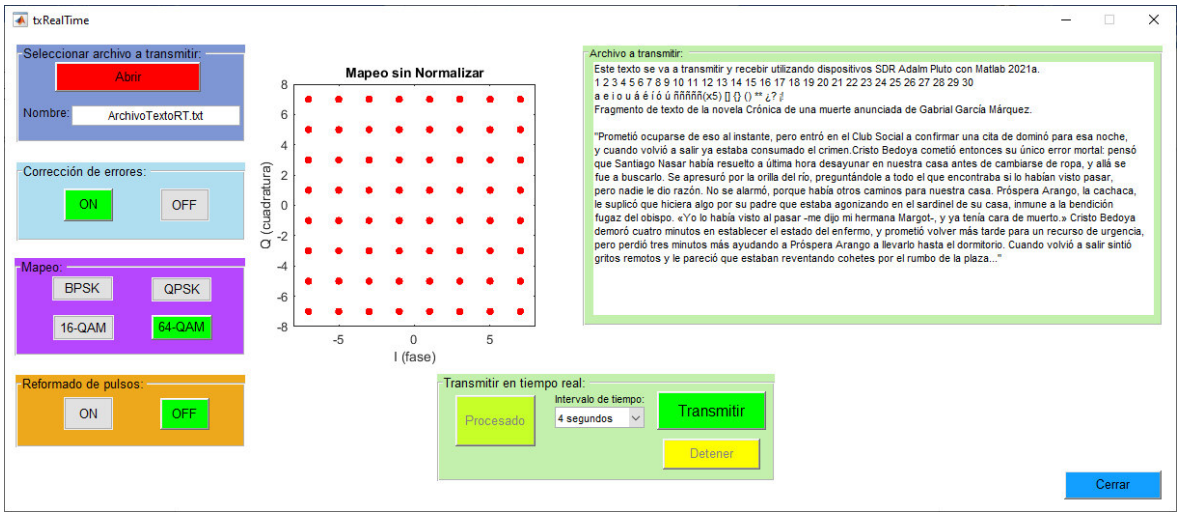


Figura 6.8. Interfaz gráfica de transmisión en tiempo real, con los datos procesados.

Si se abre otro archivo de texto, se activa/desactiva la corrección de errores, se elige otro esquema de mapeo o se activa/desactiva el reformado de pulso, en la interfaz gráfica de inhabilita el botón **Transmitir** y se habilita el botón **Procesar**.

En la Figura 6.9 se señala con color rojo el mensaje en la ventana de comandos cuando se abre un archivo de texto y con color azul los mensajes que se generan cuando se presiona el botón **Procesar**.

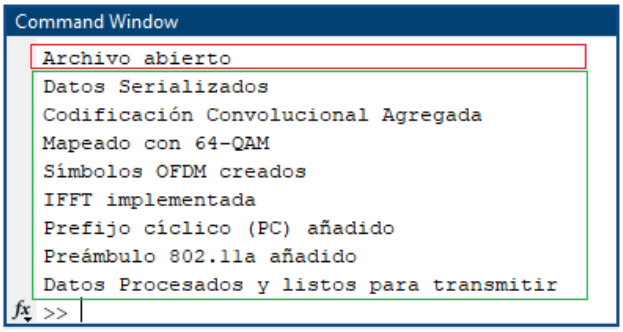


Figura 6.9. Resultado en la ventana de comandos luego de abrir y procesar un archivo de texto.

6) Presionar el botón **Transmitir** para iniciar la transmisión recursiva al canal inalámbrico, utilizando el equipo SDR Adalm Pluto. En la Figura 6.10 se muestra la interfaz gráfica de transmisión cuando está transmitiendo los datos al canal inalámbrico, los botones y el menú desplegable de inhabilitan, a excepción del botón **Detener** y el botón **Cerrar**. En la ventana de comandos de Matlab se obtiene el mensaje *"Datos transmitidos…"* cada vez que se transmiten los datos al canal inalámbrico (ver Figura 6.11).

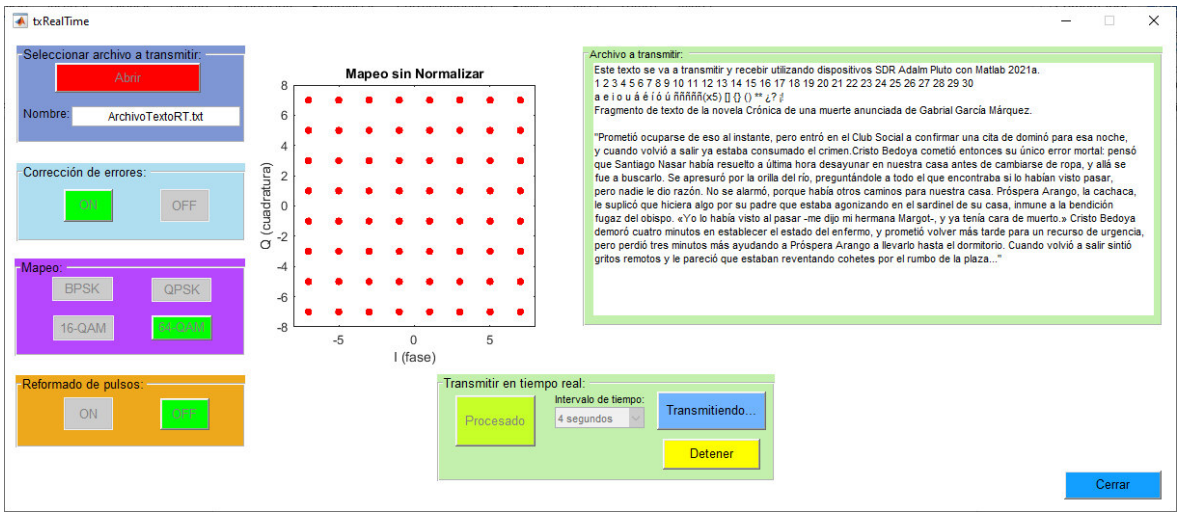


Figura 6.10. Interfaz gráfica de transmisión enviando los datos al canal.

En la Figura 6.11 se muestra el resultado en la ventana de comandos cuando se presiona por primera vez el botón **Transmitir** y se han transmitido los datos al canal ya por 6 ocasiones.

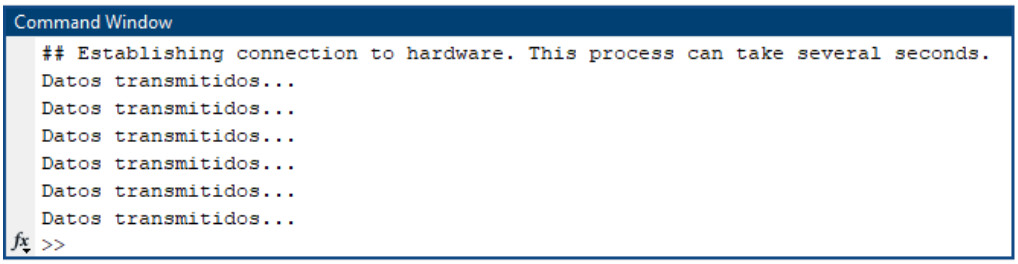


Figura 6.11. Resultado en la ventana de comandos cuando se está transmitiendo los datos al canal inalámbrico

7) Para parar la transmisión se presiona el botón **Detener** y la interfaz regresa al estado de la Figura 6.8. Además, en la ventana de comandos se muestra el mensaje *"Proceso de transmisión en tiempo real terminado"* para informar al usuario.
8) La interfaz gráfica de transmisión se cierra con el botón **Cerrar**.

4. RECEPCIÓN DE DATOS

Una vez que se ha configurado el equipo SDR Adalm Pluto, se puede acceder a la interfaz gráfica de recepción para recibir, procesar y recuperar los caracteres de texto. Además, de obtener la tasa de bits errados (BER) en base al archivo de texto original y a los bits recuperados del canal inalámbrico. Los pasos a seguir se detallan a continuación.

1) Presionar el botón **Recibir en tiempo real** de la interfaz gráfica principal (ver Figura 6.1), para acceder a la interfaz gráfica de recepción en tiempo real. En la Figura 6.12 se muestra la interfaz de recepción con los valores por defecto que se configuran al abrir la misma.

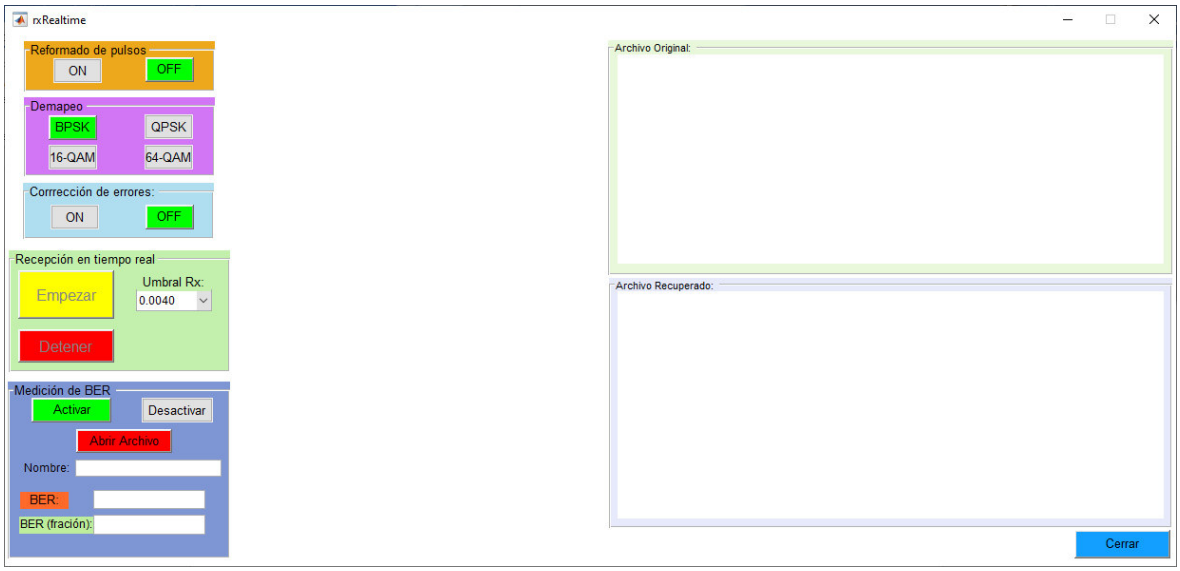


Figura 6.12. Interfaz gráfica de recepción en tiempo real, con los parámetros por defecto.

2) Seleccionar y/o activar los algoritmos y técnicas que se quieren utilizar. Estas deben coincidir con las opciones seleccionadas en el transmisor.
3) Seleccionar el umbral de recepción con el menú desplegable. Mientras menor sea el umbral, mayor será la sensibilidad del receptor.
4) Presionar el botón **Abrir Archivo** para seleccionar y abrir el archivo de texto original, mediante un cuadro de diálogo. El archivo sirve para medir el BER de los datos recibidos del canal inalámbrico. Si no se tiene el archivo original o si no se desea medir el BER se presiona el botón **Desactivar**.

   En la Figura 6.13 se muestra la interfaz gráfica de recepción, con el archivo de texto ArchivoTextoRT.txt abierto, con la corrección de errores activada, con el demapeo 64-QAM seleccionado, con el reformado de pulso desactivado y con el umbral de recepción en 0.0040.

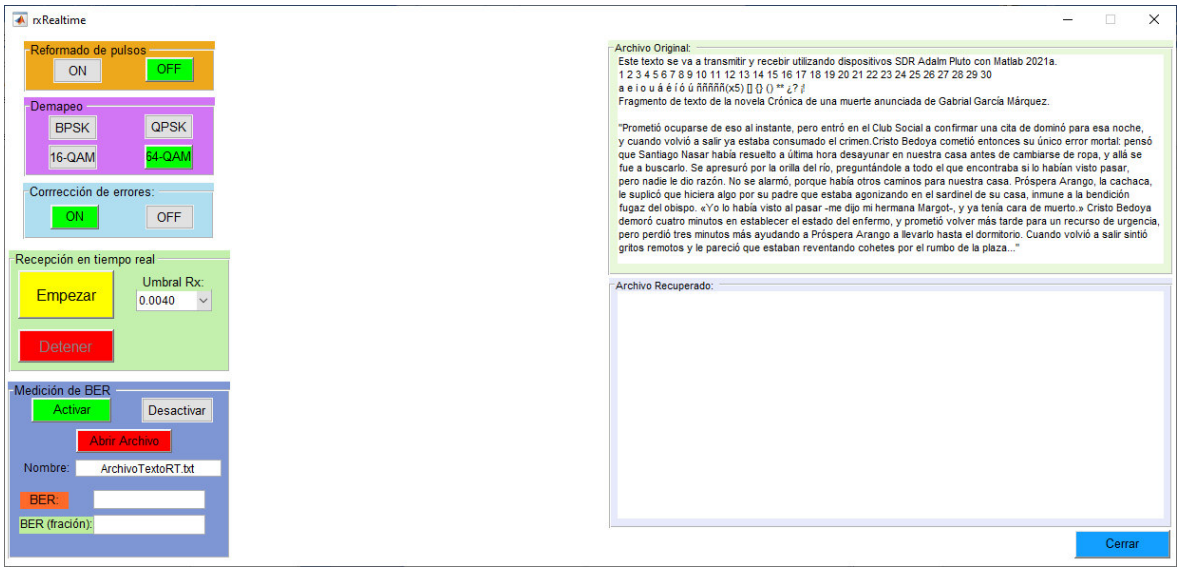


Figura 6.13. Interfaz gráfica de recepción en tiempo real, con un archivo abierto.

5) Presionar el botón **Empezar** para iniciar el proceso de recepción y procesamiento de datos en tiempo real. Cabe mencionar que el transmisor debe estar enviando los datos al canal inalámbrico.

   Cuando se presiona dicho botón por primera vez, en la ventana de comandos de Matlab se muestra un mensaje informativo para informar que se está estableciendo la conexión con el hardware (SDR Adalm Pluto). Luego empieza el proceso de muestreo del canal inalámbrico y cuando se supera el umbral inicia el procesamiento. A medida que se va avanzando en las etapas de procesamiento, se muestran mensajes informativos en la ventana de comandos.

Cuando termina el procesamiento se muestra el tiempo de procesamiento de los datos. Luego se espera se espera 4 segundos para que el usuario tenga un lapso de tiempo para poder observar los datos del BER. Y empieza nuevamente el muestreo de datos. En la Figura 6.14 se muestran los primeros resultados en la ventana de comandos. En caso de existir errores también se informan en la ventana de comandos.

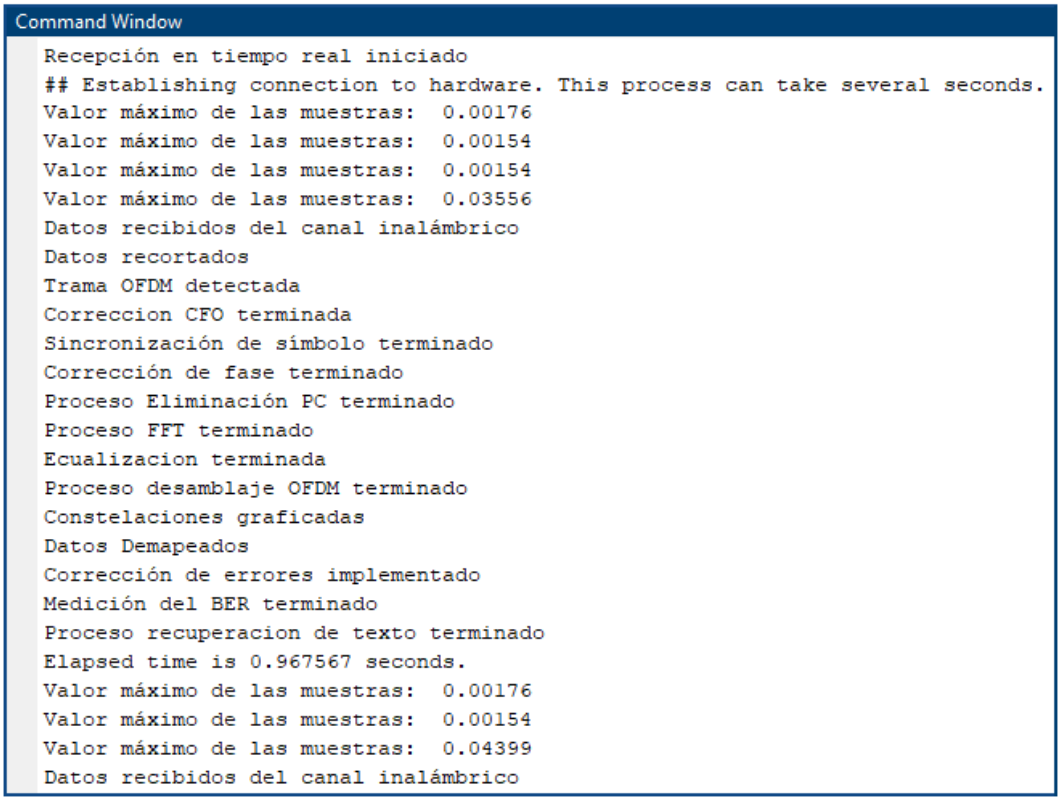


Figura 6.14. Resultado en la ventana de comandos cuando se inicia la recepción en tiempo real y se recuperan los primeros datos satisfactoriamente.

En la Figura 6.15 se muestra la interfaz gráfica de transmisión cuando se ha recibido y procesado adecuadamente los datos del canal inalámbrico, en una ocasión desde que se presionó el botón **Empezar**. Todos los botones se desactivan a excepción del botón **Detener**.

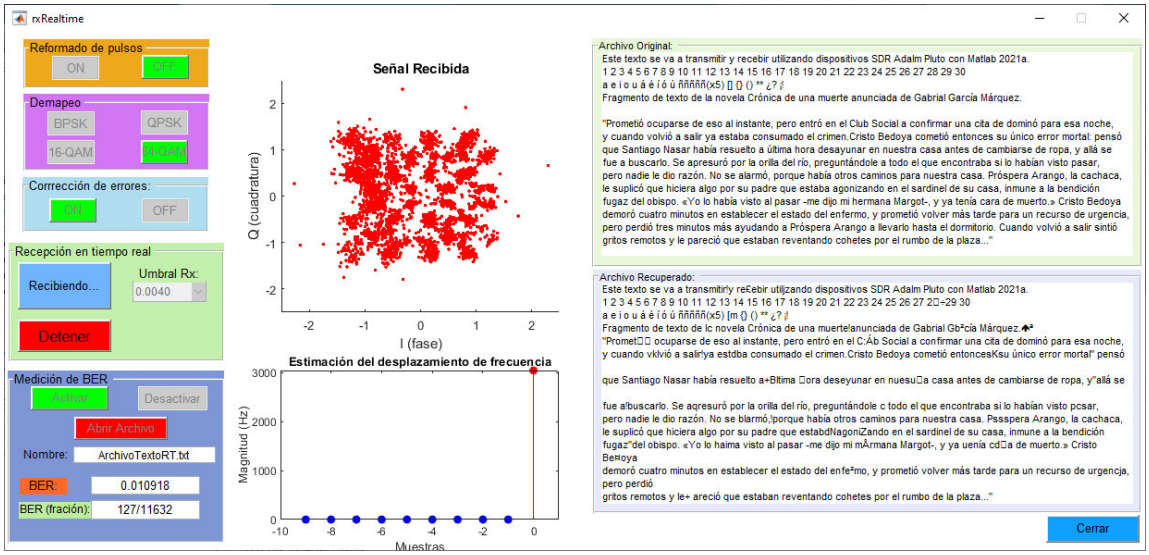


Figura 6.15. Interfaz gráfica de recepción con datos recuperados una vez.

A medida que avanza el tiempo, se reciben y procesan tramas datos que el transmisor envía recursivamente. En la gráfica inferior de la interfaz gráfica se almacenan y grafican los valores pasados (azul) y actual (rojo) de la estimación del CFO.

6) Presionar el botón **Detener** para parar la recepción y procesamiento recursivo de los datos del canal inalámbrico. En la interfaz se conservan los últimos valores del diagrama de constelación, BER y caracteres recuperados del canal. Además, se mantiene la gráfica de los valores acumulados de la estimación del CFO. En la **Figura 6.16** se muestra la interfaz gráfica de recepción detenida.

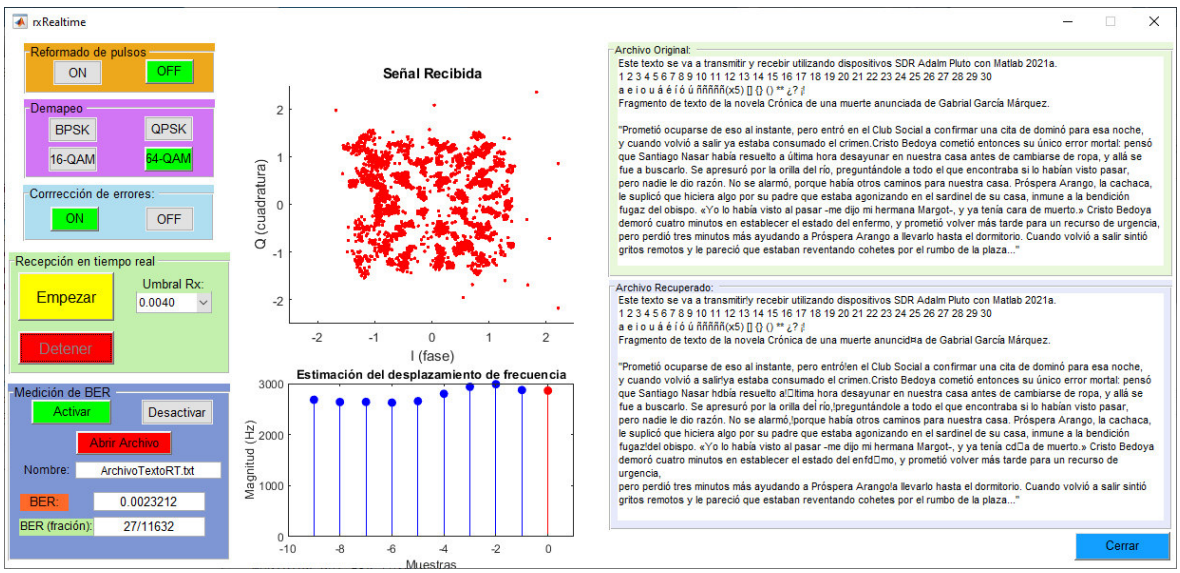


Figura 6.16. Interfaz gráfica de recepción con datos recuperados, al menos en 10 ocasiones.

7) Se puede repetir los pasos 2, 3 y/o 4 para luego seguir nuevamente con el paso 5.
8) Cerrar la interfaz gráfica de transmisión con el botón **Cerrar**.

ORDEN DE EMPASTADO