Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Caracterización de dispositivos RTL-SDR para medida de señales aeronáuticas

Autor: Jonathan Majada Costas

Tutor: Joaquín Granado Romero

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Caracterización de dispositivos RTL-SDR para medida de señales aeronáuticas

Autor:
Jonathan Majada Costas

Tutor:
Joaquín Granado Romero
Profesor Titular de Universidad

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Caracterización de dispositivos RTL-SDR para medida de señales aeronáuticas

Autor: Jonathan Majada Costas
Tutor: Joaquín Granado Romero

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría agradecerle a mi tutor, Joaquín Granado Romero, por guiarme y aconsejarme a lo largo de este proyecto. Agradezco también a los profesores y trabajadores de la Escuela el gran trabajo que hacen para que los alumnos tengamos la oportunidad de aprender aquí.

También quiero agradecer a los compañeros que he tenido a lo largo de estos años por compartir conmigo esta experiencia, en especial a aquellos a los que ahora puedo llamar amigos: Pablo, Jose Ángel, Lorenzo, José, Manolo, Ricardo, Santi, Josema, Ignacio y Benito.

Muchas gracias a mis amigos de siempre por estar ahí, aunque yo haya estado tan lejos y haya tenido tan poco tiempo que daros, gracias kuadrilla.

Por último, y más importante, gracias a mis padres por darme la oportunidad de estudiar este grado y apoyarme a lo largo de estos años. Y gracias también mi abuela y mis primos por estar ahí y hacerme más feliz.

Jonathan Majada Costas

Sevilla, 2020

Resumen

El sistema de comunicaciones aeronáuticas ACARS está muy extendido en la navegación aérea actual, estando equipado en más de 10.000 aviones [1]. En sus inicios se utilizaba para automatizar ciertas comunicaciones de los pilotos con su compañía aérea. Con el tiempo se han incorporando nuevas aplicaciones, desde manejar información de mantenimiento hasta mantener conversaciones con el control de tráfico aéreo. El objetivo de este proyecto es la realización de una aplicación capaz de detectar la transmisión de mensajes del sistema de comunicaciones aeronáuticas ACARS.

Para ello, se ha utilizado un dispositivo receptor de radiofrecuencias programable mediante software para grabar señales sobre las que probar el detector. Es un RTL-SDR, y en este trabajo se analiza su funcionamiento, hardware y software. También se exploran las posibilidades que ofrecen este tipo de dispositivos y varios programas que permiten su uso. El dispositivo se conecta al ordenador a través de un cable USB y, para controlarlo, se han utilizado MATLAB y Python.

Dado que se quieren detectar transmisiones ACARS, en el documento se analiza este sistema de comunicaciones. Empezando por qué información se comunica, se explica la estructura de los mensajes transmitidos. Se continua examinando las redes actuales del sistema. Y por último, se trata la modulación utilizada.

Siendo Python un lenguaje de programación muy popular, a lo largo del trabajo se explican sus peculiaridades. Se analizan los intérpretes y librerías que facilitan su uso y aportan nuevas funciones. Además, se presentan varios códigos realizados por el autor del trabajo que simplifican el control del RTL-SDR.

La parte central del proyecto trata sobre la aplicación para detectar la presencia de mensajes del sistema ACARS. Se explican los distintos filtros y elementos que se han utilizado en el procesamiento de la señal capturada. Tras filtrarla se utiliza un generador de señales que se sincroniza con la portadora de los mensajes ACARS. Después se aplica un filtrado que permite detectar fácilmente cuándo se ha transmitido un mensaje a través del sistema ACARS. Se hace el modelo de la aplicación mediante MATLAB y se prueba sobre unas capturas reales en las que hay presencia de mensajes ACARS. Además, se implementa el detector en Python y se comprueba si funciona igual que en MATLAB. Por último, se analiza el rendimiento de la aplicación en los dos programas.

Finalmente se exponen las conclusiones y se presentan posibles líneas de trabajo futuras.

Abstract

The ACARS aeronautical communications system is widespread in today's air navigation, being equipped on more than 10,000 different aircrafts [1]. At first it was used to automate communications between pilots and their airlines. Over time, new apllications have been incorporated, from handling maintenance information to holding conversations with air traffic control. The objective of this project is to create an application capable of detecting the transmission of messages from the ACARS aeronautical communications system.

For this, a software programmable radio frequency receiver device has been used to record signals on which to test the detector. It is an RTL-SDR, and its hardware and software are analyzed in this work. The possibilities offered by these types of devices and various programs that allow their use are also explored. The device connects to the computer through a USB cable and MATLAB and Python have been used to control it.

Since ACARS transmissions are to be detected, this communication system is analyzed in the document. Beginning with what information is communicated, the structure of the transmitted messages is explained. Then, the current networks of the system are examined. And finally, the modulation used is described.

As Python is a very popular programming language, its peculiarities are explained throughout the work. Interpreters and libraries that facilitate its use and provide new functions are analyzed. In addition, several codes made by the author of this work, which simplify the control of the RTL-SDR, are introduced.

The central part of the project deals with the application to detect the presence of messages from the ACARS system. The different filters and elements that have been used in the processing of the captured signal are explained. After filtering it, a signal generator that synchronizes with the carrier of the ACARS messages is used. Filtering is then applied to easily detect when a message has been transmitted through the ACARS system. The application is modeled using MATLAB and tested on real captures in which there are ACARS messages. Also, the detector is implemented in Python and it is checked if it works the same as in MATLAB. Lastly, the performance of the application in the two programs is analyzed.

Finally, the conclusions and possible future lines of work are presented.

Índice

1 Introducción

La ingeniería aeroespacial es una carrera multidisciplinar, donde se aprende la importancia de todo tipo de campos de conocimiento. Como parte de la especialización de navegación aérea, se obtienen conocimientos relativos a las comunicaciones y su función en el sector aeroespacial. Además, se estudian varios dispositivos tecnológicos como las placas Arduino o las FPGA. También se tratan en profundidad el procesamiento de señales y la programación, dos campos indispensables en las comunicaciones.

Las comunicaciones son un elemento esencial en el sector de la aviación. Se utilizan para transmitir y recibir todo tipo de información relevante para poder realizar vuelos seguros. Uno tipos de comunicación muy utilizado es el sistema ACARS, equipado en más de 10.000 aviones [1] . Inventado en los años 70, ha ido evolucionando a lo largo de estos años hasta cobrar gran importancia en el día a día del tráfico aéreo. Como con todas las comunicaciones, el número de canales disponible para estas transmisiones está limitado. Esto puede crear situaciones de saturación de los canales.

Una forma sencilla de acercarse al campo de las comunicaciones es el uso de los dispositivos RTL-SDR. Son radios de pequeño tamaño y bajo precio controlables con cualquier ordenador e incluso desde teléfonos móviles. Siendo cada vez más conocidas, los usuarios de estas radios crean nuevas aplicaciones continuamente. Son dispositivos fáciles de utilizar y permiten recibir una gran variedad de señales, incluidas las del sistema ACARS.

Para controlar los RTL-SDR es necesario utilizar algún software en el ordenador al que se hayan conectado. Existen una gran variedad de opciones, dos de ellas muy destacables. Por un lado está MATLAB, un programa extremadamente conocido en la ingeniería y muy utilizado durante el grado. Por otro lado se encuentra Python, un lenguaje de programación que ha ganado en popularidad los últimos años. En la carrera se aprende programación, pero prácticamente solo centrada en MATLAB. Python es un lenguaje de programación muy versátil y muy utilizado en diferentes ramas de la ingeniería. Por desgracia, aún no se ha introducido en la docencia del grado.

Con todo esto en cuenta, se vio este trabajo como una oportunidad de ampliar conocimientos y combinar las distintas tecnologías mencionadas. Así, la intención del trabajo es utilizar un RTL-SDR para trabajar con señales ACARS a través de MATLAB y Python.

1.1 Alcance y objetivos

El objetivo principal de este trabajo es la creación de una aplicación o herramienta que combine el uso de un RTL-SDR con programación en MATLAB y Python. Para darle una utilidad en el sector de la aeronáutica, se pretende que esta aplicación sirva para detectar la presencia de mensajes transmitidos a través del sistema ACARS.

Este objetivo se ha elegido con la intención de profundizar en cada uno de los temas mencionados y cumplir así con varios objetivos secundarios:

- Investigar el posible uso de dispositivos RTL-SDR en aplicaciones aeronáuticas, estudiar los diversos programas que permiten su uso y explorar cómo usarlos desde MATLAB y Python.
- Estudiar el funcionamiento del sistema de comunicaciones ACARS.
- Realizar una aplicación básica consistente en utilizar las peculiaridades de la modulación de los mensajes ACARS para detectar su transmisión.

1.2 Estructura

Con estos objetivos en mente, el trabajo comienza analizando los distintos temas por separado y luego se juntan los conocimientos adquiridos para tratar la creación de la aplicación.

Se empieza, en el Capítulo 2, explicando qué son los RTL-SDR. Después se analiza la estructura interna del hardware de estos dispositivos. Tras comentar las posibles aplicaciones que tienen, se procede a considerar distintos software que permiten su uso. En esta parte se da una explicación sobre cómo utilizar un RTL-SDR mediante MATLAB y Python.

El Capítulo 3 trata sobre el sistema de comunicaciones ACARS, explicando desde cómo está estructurado hasta qué modulación se utiliza en la transmisión de los mensajes.

Python es una parte importante del trabajo y tiene dedicado el Capítulo 4. Se explica como utilizar los intérpretes para poder programar en Python y se analizan las librerías utilizadas en el trabajo. Después se introducen unos códigos desarrollados por el autor del trabajo para facilitar el uso del dispositivo RTL-SDR. Por último, se analizan los pros y contras de este lenguaje de programación.

En el Capítulo 5 se explica el procesamiento que se le va a realizar a las capturas de canales ACARS para la detección de mensajes. Se da una explicación del proceso y de las distintas herramientas y filtros utilizados. Se explica la implementación del código que aplica este procesamiento en MATLAB y Python. Se usa el detector con ejemplos reales de captura de señales ACARS y también se analiza el rendimiento de la aplicación en cada programa.

Por último, en el Capítulo 6 se revisan los resultados y se comentan posibles líneas futuras de trabajo.

2 RTL-SDR

Uno de los enfoques de este trabajo ha sido la utilización de un RTL-SDR, investigar cómo funciona, qué programas permiten su uso y ver sus posibles aplicaciones. En este apartado se empieza explicando qué es, cómo funciona y qué rango de frecuencias cubre. Después se habla sobre sus usos e importancia en el mundo tecnológico. Finalmente, se analizan varios programas que proporcionan control sobre el dispositivo.

2.1 Radio definida por software

Antes de centrarse en el RTL-SDR es necesario comenzar explicando el concepto de *Software Defined Radio* (SDR), una tecnología muy importante en el campo de las comunicaciones inalámbricas. Una radio definida por software es, según la definición de IEEE [2], "una radio cuyas funciones físicas (o parte de ellas) se definen mediante software". Es una agrupación de hardware y software que, mediante software programable, cumple las funciones de una radio. Una SDR puede sintonizar cualquier frecuencia dentro del rango que le permita su hardware y aplicar distintas demodulaciones y modulaciones gracias a su programabilidad.

La arquitectura de una SDR no es complicada. Ha ido evolucionando desde el trabajo de Mitola [3] [4] a finales de 1990. La idea tras esta tecnología es que la mayor parte de las tareas, tales como selección del canal o procesamiento de la señal, se realicen en el entorno digital, es decir, vía software. Por ello, se procura colocar los convertidores analógico a digital (ADC) y digital a analógico (DAC) lo más cercanos a la antena como sea posible [5]. Una de las posibles arquitecturas de una SDR se muestra en la Figura 2.1. Entre la antena y los convertidores solo se encuentran los elementos indispensables: amplificador, filtro y mezclador. Esa parte analógica se encarga por tanto de la transmisión y recepción, filtrado, amplificación y cambio a o desde frecuencia intermedia o baja. Aún siendo analógica también está controlada por software, como ejemplo, la selección de la ganancia del dispositivo. El ADC muestrea la señal y la transmite al bloque de procesamiento digital. El DAC hace lo inverso. La parte del procesamiento digital puede estar formada por una FPGA, una unidad de procesamiento digital de señal (DSP) o un ordenador. Existen tres configuraciones mayoritarias de SDR [5]: solo receptor, solo transmisor y transceptor. Los RTL-SDR se encuentran en la primera categoría.

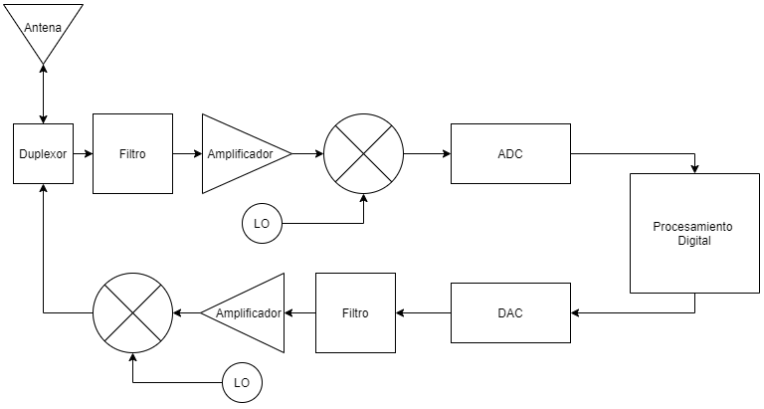


Figura 2.1 Diagrama de bloques de la arquitectura SDR.

2.2 Hardware del RTL-SDR

Un RTL-SDR es un dispositivo USB pequeño, compacto, sencillo de utilizar y, actualmente, muy barato. Es capaz de recibir señales de radiofrecuencia, convertirlas a digital y procesarlas para obtener los datos I/Q en 8 bits. Originalmente utilizados como receptores de televisión DVB-T [6], se descubrió que se podían utilizar como receptores SDR al cambiarlos a modo test. En este modo no se ejecuta la decodificación final, el RTL-SDR aporta directamente las muestras I/Q en banda base. Además, el dispositivo deja de estar limitado a recibir solo señales en la banda de frecuencias de televisión digital. Puede captar cualquier frecuencia en la que opere su sintonizador. El diagrama de las funciones del RTL-SDR se muestra en la Figura 2.2. La salida del dispositivo puede conectarse a un ordenador que permita su control, y los datos que transmite se pueden utilizar con una gran variedad de programas.

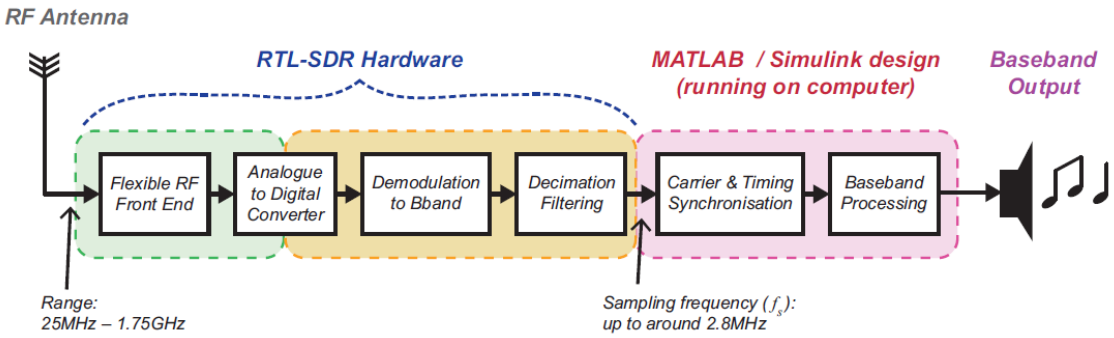


Figura 2.2 Diagrama de bloques del RTL-SDR [6].

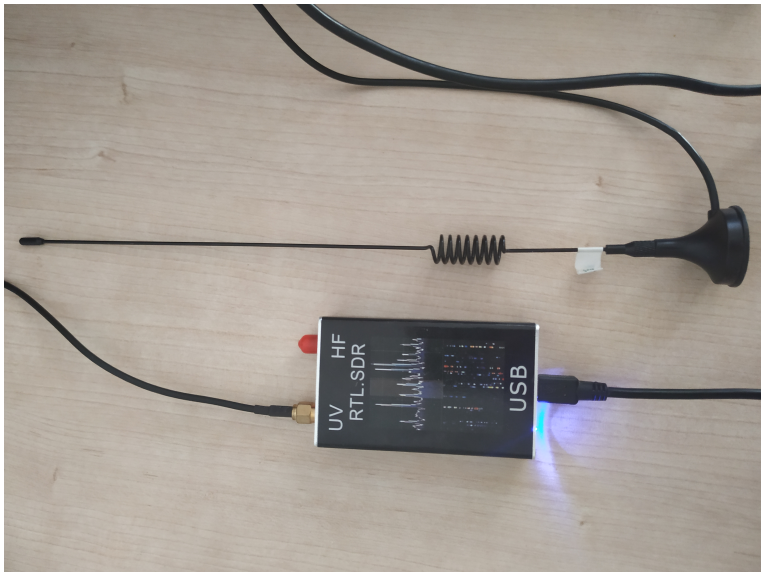


Figura 2.3 RTLS-SDR utilizado.

El RTL-SDR utilizado en este trabajo es el de la Figura 2.3, donde aparece junto a la antena utilizada. Esta formado por un chip RTL2832U [7] y un sintonizador R820T [8]. Este tipo de RTL-SDR es uno de los más comunes debido a su precio, cercano a unos 20 euros. El sintonizador cubre las frecuencias desde 25 MHz hasta 1.75 GHz. Se encarga de sintonizar la frecuencia indicada y convertir la señal a frecuencia intermedia (IF). El RTL2832U es un demodulador COFDM con interfaz USB 2.0. Muestrea la señal, la separa en datos I/Q y los transmite a través del USB. Conseguir los datos en este formato permite que la demodulación se haga en el software elegido por el usuario, posibilitando demodular señales que se encuentren en su rango de frecuencias, ya sean moduladas en AM, FM o incluso GPS.

El funcionamiento concreto se muestra en la Figura 2.4. Mediante software se controla la ganancia K del

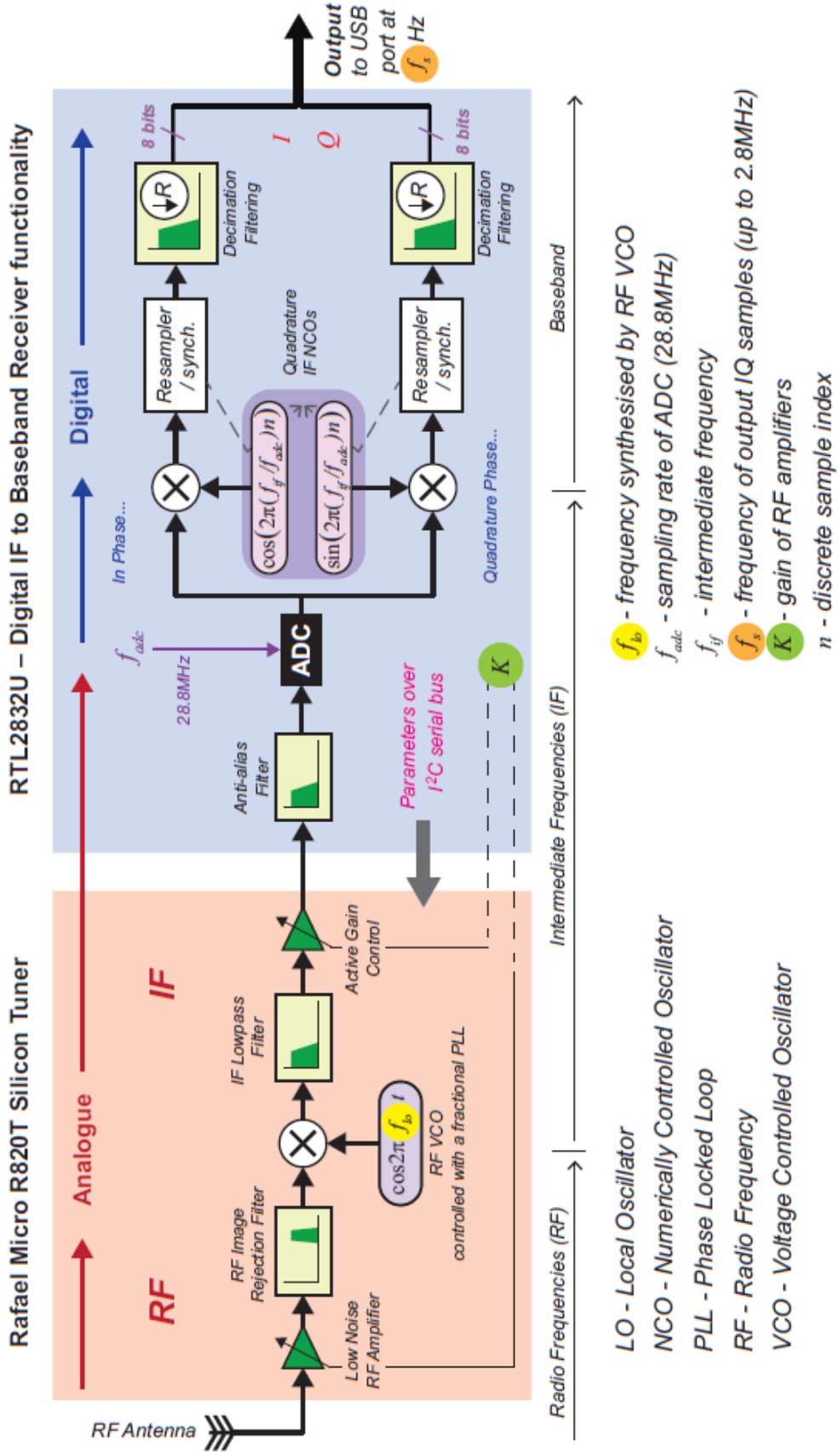


Figura 2.4 Estructura interna del RTL-SDR [6].

sintonizador y la frecuencia del canal a captar f_c. Una vez captada la señal RF, esta se convierte a IF mediante un oscilador controlado por voltaje (VCO). Se muestrea a una frecuencia de 28.8 MHz y después se pasa a banda base usando un oscilador en cuadratura controlado numéricamente (NCO). Este NCO convierte la señal a banda base y separa la señal en datos I/Q. La frecuencia f_s, que se selecciona mediante software, permite diezmar la señal transferida por USB. La tasa de muestreo máxima es de 3.2 MHz. Sin embargo, suele usarse con una tasa máxima de 2.56 MHz, ya que con una mayor el RTL-SDR se vuelve inestable y se pueden perder muestras [9]. Tras realizar varias pruebas, el RTL-SDR utilizado permite seleccionar una frecuencia de muestreo entre dos rangos, de 225 KHz a 300 KHz y de 900 KHz a 3.2 MHz.

2.3 Aplicaciones

Tal y como se ha mencionado antes, el RTL-SDR utilizado en este trabajo puede recibir cualquier frecuencia entre 25 MHz y 1.75 GHz. Esto significa que el dispositivo permite captar muchos tipos de señal. A continuación se enumeran algunas de las más importantes [10]:

- *87.5 - 108 MHz*: Radio FM.
- *108 - 117.975 MHz*: Comunicaciones aeronáuticas, GBAS, ILS, VOR.
- *117.975 - 138 MHz*: Comunicaciones aeronáuticas, incluido ACARS.
- *154 - 162.0375 MHz*: Comunicaciones marítimas.
- *876 - 960 MHz*: Comunicaciones móviles (GSM).
- *1164 - 1350 MHz*: Sistemas de navegación por satélite, Glonass, Galileo y GPS.
- *1350 - 1427 MHz*: Radio astronomía.

Con acceso a una banda tan amplia de frecuencias, el dispositivo puede usarse como escáner de radio con muchas posibles aplicaciones [11]:

- Escuchar comunicaciones de emergencias (ambulancias, bomberos, etc.).
- Escuchar conversaciones de control de tráfico aéreo.
- Seguimiento radar de posición de aeronaves decodificando el ADSB.
- Escucha y decodificación de mensajes ACARS.
- Seguimiento radar de posición de barcos decodificando AIS.
- Recepción de datos de globos meteorológicos.
- Escuchar la radio FM.
- Ver canales de televisión analógica y digital.
- Escuchar satélites y la Estación Espacial Internacional.
- Radio astronomía.

Que sea posible no implica que deba hacerse, lo primero es conocer qué es legal en el país en el que se esté realizando el uso del RTL-SDR.

Debido a la flexibilidad (pueden utilizarse en sistemas operativos muy variados, como Windows, MAC o LINUX), facilidad de uso y bajo precio de los RTL-SDR, se han ido encontrando cada vez más usos y aplicaciones para estos dispositivos. Su uso en la educación ha sido analizado en varios trabajos [12], [13] y [14], ya que facilita hacer demostraciones prácticas de lo estudiado. Se han estudiado todo tipo de aplicaciones prácticas, desde crear un analizador de espectro [15] o monitorizar el uso de parte del espectro en [16] y [17], hasta establecer una estación para monitorizar la integridad del GNSS [18]. Se pueden utilizar para detectar y decodificar señales con modulaciones concretas [19]. Como es una tecnología aún en pleno crecimiento, se sigue estudiando cómo mejorarlos [20].

Teniendo en cuenta todas las aplicaciones y posibles usos de estos dispositivos, es lógico que su utilización sea cada vez más común. Una de las razones por las que se ha decidido utilizar el RTL-SDR en este trabajo es comprobar cómo puede usarse mediante MATLAB y Python, dos programas muy extendidos en el sector de la ingeniería y las comunicaciones. La otra es comprobar si podría ser de utilidad en el sector de las comunicaciones aeronáuticas, razón por la que el trabajo se centra en las comunicaciones ACARS.

2.4 Instalación en Windows

El ordenador utilizado para realizar este trabajo tiene un sistema operativo Windows 8.1, en esta sección se va a explicar cómo instalar los drivers necesarios para que Windows reconozca el RTL-SDR. El procedimiento a seguir se obtuvo de [21]:

1. Tener instalado Microsoft.NET 4.6 [22] o una versión más reciente.
2. Tener instalado Microsoft Visual C++ [23], si no está instalado en los siguientes pasos se tiene la opción de instalarlo.
3. Descargar el *Community Package with Plugins* de [24].
4. Ejecutar el instalador que se descarga y seguir los pasos que se indican.
5. Una vez terminada la instalación conectar el RTL-SDR al ordenador y esperar hasta que el termine de intentar instalarlo, paso que fallará o instalará unos drivers que no sirven.
6. Ir la ubicación en la que se haya instalado el paso 4, por defecto en *C:\SDRSharp* y ejecutar *zadig.exe*.
7. Debe abrirse la ventana de la Figura 2.5. Entrar en opciones y activar *List All Devices*.

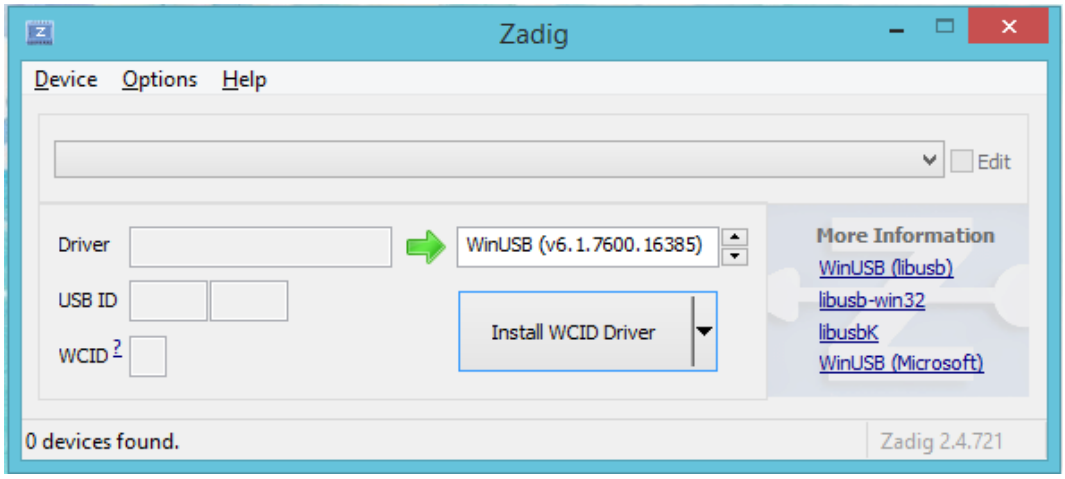


Figura 2.5 zadig.exe.

8. De la lista despegable elegir *RTL2838UHIDIR*, dependiendo del dispositivo los números pueden ser diferentes, e incluso aparecer *Bulk-In, Interface (Interface 0)*.

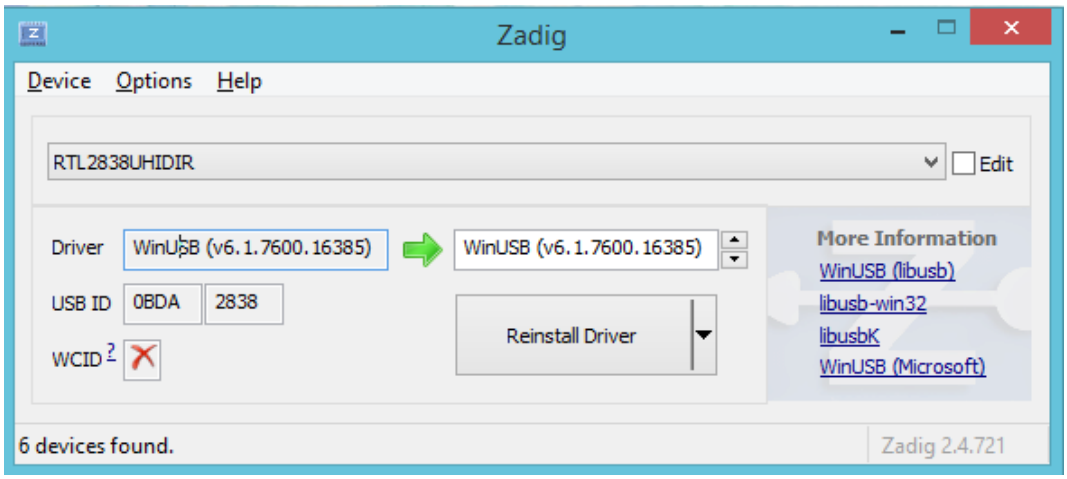


Figura 2.6 zadig.exe, instalación del driver.

9. En el bloque a la derecha de la flecha verde de la Figura 2.6 seleccionar *WinUSB*.
10. Hacer click en *Replace Driver* y ante posibles avisos de Windows dar permiso a la instalación.

Al seguir estos pasos se han instalado los drivers, por lo que ahora el ordenador reconoce el dispositivo. Además, durante los pasos 3 y 4 se ha aprovechado para descargar e instalar el programa SDRSharp que se usará y explicará más adelante.

2.5 Software del RTL-SDR

Existen muchos programas para interactuar con el RTL-SDR. La mayoría de estos programas son gratuitos y han sido recopilados en [25] y [26], donde pueden encontrarse muchos más.

- *PLSDR* [27]. Programa de código abierto utilizable en Windows y Linux, que requiere Python y GNU Radio. Capta las señales en directo y las muestra tanto en frecuencia como en cascada. Es un programa fácil desde el punto de vista del usuario, con una interfaz clara y sencilla. No tiene muchas opciones ni controles, pero cumple su función.
- *Gqrx* [28]. Programa de código abierto muy similar a *PLSDR*, con más opciones y posibilidad de usarlo también en Raspberry Pi. Soporta una mayor variedad de dispositivos SDR.
- *CubicSDR* [29]. Otro programa de código abierto que permite captar señales con el RTL-SDR y tiene demodulaciones AM y FM. Permite conectarse con otros programas para que esos demodulen otro tipo de señales. Funciona en Windows y Linux. La interfaz muestra una gran variedad de gráficos con todo tipo de información.
- *HSDR* [30]. Otro programa similar a los anteriores. solo utilizable en Windows. Como características especiales permite la aplicación de filtros manualmente. Puede centrar la frecuencia de forma automática.
- *SDR-Radio.com* [31]. Un programa solo para Windows más complicado de utilizar que los anteriores. A cambio, tiene varios decodificadores, seguimiento de satélites y posibilidad de escuchar 6 señales a la vez.
- *Linrad* [32]. Programa gratuito para Windows, Linux y MAC. Es difícil de aprender a utilizar, con una interfaz poco amigable. Pero es uno de los programas más completos y complejos que se pueden encontrar.
- *Studio1* [33]. Programa de pago para Windows. Con capacidades avanzadas, tiene una interfaz limpia aunque muestra mucha información. Parece ser uno de los software con menor gasto de CPU. Existe una versión gratuita llamada *SDRUno* [34].
- *ShinySDR* [35]. Se controla a través de una interfaz web. Como característica especial, permite mover, agrandar y resintonizar el espectrograma sin perder la información.
- *SDR Touch* [36]. Programa de pago para Android y Kindle. No tiene tantas opciones como los programas para ordenador, pero es una muestra de la versatilidad de los SDR, al poder utilizarse incluso en teléfonos móviles.
- *Zeus Radio* [37]. Programa de pago para Windows y Linux. Aparte de receptor, puede ser utilizado como transmisor con dispositivos adecuados.

Es debido hacer una mención especial a otro programa muy utilizado para trabajar con los RTL-SDR: GNU Radio [38]. Es una plataforma de código abierto para creación de aplicaciones SDR con código hecho en Python y C++. Su herramienta GNU Radio Companion es una interfaz similar a Simulink, que permite la conexión de bloques de procesamiento de señales para crear todo tipo de aplicaciones. No se ha intentado incorporar el uso de esta herramienta al proyecto, ya que se sale del alcance. Pero tiene un tutorial oficial [39], y existen muchos proyectos relacionados con los RTL-SDR que usan este programa. Por ejemplo, un receptor FM [40] y [41], su posible uso en educación [13] y en investigación [42].

Como se puede ver, el RTL-SDR es una tecnología muy utilizada y con muchos proyectos abiertos. Existen muchas posibilidades y en este trabajo no se van a estudiar todos programas existentes. Sin embargo, como se va a usar un RTL-SDR sí que es necesario trabajar con algún programa que posibilite su utilización. Este trabajo se va a centrar en utilizar tres opciones: MATLAB, Python y SDRSharp. En los próximos apartados se va a explicar cómo instalar todo lo necesario para usar el dispositivo en cada programa, cómo se utilizan y se analizará qué permiten hacer.

2.5.1 SDRSharp

SDR# o SDRSharp es un programa gratuito para Windows. Se descargó e instaló en la Sección 2.4. Se ha decidido probar a utilizar el RTL-SDR con este programa por ser uno de los más populares y por ser más sencillo empezar a usar el dispositivo con un receptor SDR ya diseñado que con MATLAB. Es un programa sencillo de entender, con una interfaz y controles amigables. Además existen muchos programas adicionales que se pueden instalar, añadiendo todo tipo de nuevas funciones [43]. Este programa permite utilizar el RTL-SDR como una radio, captando y decodificando en directo una gran variedad de señales. Presenta tres gráficos principales: Transformada rápida de Fourier (FFT), espectrograma en cascada y espectro IF.

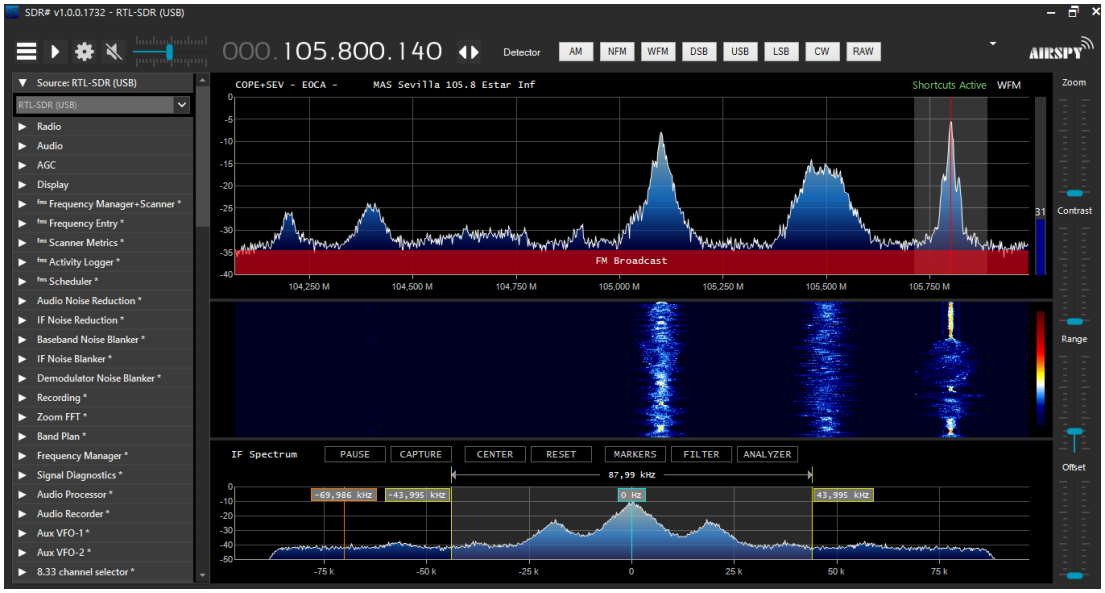


Figura 2.7 Interfaz de SDRSharp.

Al iniciarlo por primera vez es necesario seleccionar el dispositivo *RTL-SDR (USB)* en la pestaña *Source*. En la Figura 2.7 se muestra el programa. En el centro se encuentran las tres gráficas antes mencionadas. A la izquierda están todas las herramientas que ofrece el programa. A la derecha hay cuatro controles que permiten modificar las gráficas para ver bien las señales captadas. En la zona de arriba está el botón de activar el dispositivo, los ajustes (ganancia, tiempo de muestreo, modo de muestreo, etc.), volumen de audio de la señal captada, selección de frecuencia central y tipo de demodulación.

Los controles son intuitivos. La frecuencia puede seleccionarse escribiéndola arriba, seleccionando con el ratón sobre la gráfica e incluso moverla con la rueda del ratón. También puede controlarse el ancho de banda, ya sea con el ratón o dentro de la herramienta *Radio*. Contiene muchos tipos de demodulación, filtros, reductores de ruido y opciones para grabar la señal, ya sea el audio o la información I/Q pura.

No se va a explicar mucho más sobre este programa, ya que solo se ha usado para comprobar el funcionamiento del RTL-SDR. Para profundizar existe una guía [44] y una explicación de las herramientas adicionales [43]. Tras analizarlo, es un programa muy versátil y fácil de usar que ofrece la posibilidad de grabar y ver la señal al mismo tiempo. Es destacable la gran cantidad de tipos de demodulación que contiene. Teniendo todo esto en cuenta, es un buen primer programa para comprender el funcionamiento de estos dispositivos.

2.5.2 Procesamiento MATLAB en tiempo real

MATLAB [45] es un programa muy utilizado en la ingeniería. A lo largo del Grado en Ingeniería Aeroespacial hay varias asignaturas dedicadas a MATLAB, y otras en las que su uso es imprescindible. Por tanto, es un software conocido y, con la ayuda del trabajo de Stewart [6], se puede utilizar con el RTL-SDR. La versión de MATLAB utilizada es la R2020a para estudiantes [46]. Lo primero es descargar los paquetes, o *toolbox*, necesarios para poder trabajar con señales: *Communications Toolbox* [47], *DSP System Toolbox* [48] y *Signal Processing Toolbox* [49]. Además, para poder interactuar con el RTL-SDR es necesario el *Communications Toolbox Support Package for RTL-SDR Radio* [50]. Una vez estén todos instalados se puede empezar a trabajar.

Tras conectar el RTL-SDR se introduce el comando *my_rtlsdr = sdrinfo* para obtener la información de la Figura 2.8. Con esto se comprueba que MATLAB reconoce el dispositivo y además aporta datos concretos del mismo. A continuación se va a explicar cómo utilizar el RTL-SDR sin profundizar demasiado, ya que se sale del alcance de este trabajo. Para explicaciones y análisis en profundidad y ejemplos listos para usar lo mejor es acudir a [51], donde pueden obtenerse el libro y los ejemplos de forma gratuita. Si se descargan los archivos el siguiente paso es añadir a la ruta de MATLAB la carpeta de nombre *Intro* que contiene dos archivos. Gracias a ello se obtienen nuevas herramientas para usar con el RTL-SDR. Se puede trabajar tanto con Simulink como directamente con scripts y comandos.

```
my_rtlsdr =

                RadioName: 'Generic RTL2832U OEM'
             RadioAddress: '0'
              RadioIsOpen: 0
                TunerName: 'R820T'
             Manufacturer: 'Realtek'
                  Product: 'RTL2838UHIDIR'
               GainValues: [29×1 double]
    RTLCrystalFrequency: 28800000
  TunerCrystalFrequency: 28800000
           SamplingMode: 'Quadrature'
           OffsetTuning: 'Disabled'
```

Figura 2.8 Información del RTL-SDR.

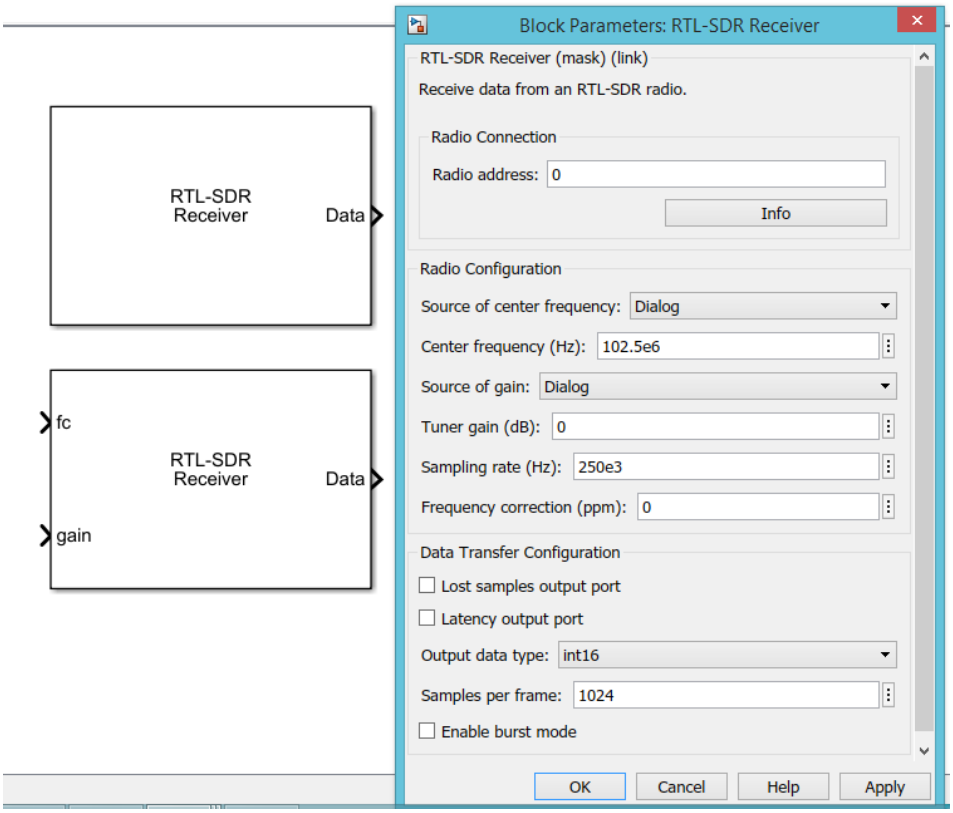


Figura 2.9 Bloque RTL-SDR Receiver y sus parámetros.

Primero se va a explicar cómo usar el dispositivo con Simulink por ser una herramienta de trabajo más gráfica. El toolbox de RTL-SDR proporciona un nuevo bloque para Simulink, el *RTL-SDR Receiver*, cuyos parámetros se ven en la Figura 2.9. En la línea *Radio address* hay que escribir la dirección que se indica en la Figura 2.8 para que el bloque actúe sobre el dispositivo conectado. Dentro del bloque de configuración de la

radio hay varias opciones. Se pueden elegir dos maneras de seleccionar la frecuencia central, escribiéndola directamente en este bloque o como entrada al mismo. Con la ganancia existen esas dos opciones y también la llamada *AGC*, que la selecciona de forma automática. Además, se establecen aquí la tasa de muestreo y la corrección de frecuencia. El último bloque configura la transferencia de los datos I/Q obtenidos del RTL-SDR. Se puede elegir el formato de los datos de salida (int16, single o double), el número de muestras por frame de salida y activar salidas de datos de latencia y muestras perdidas.

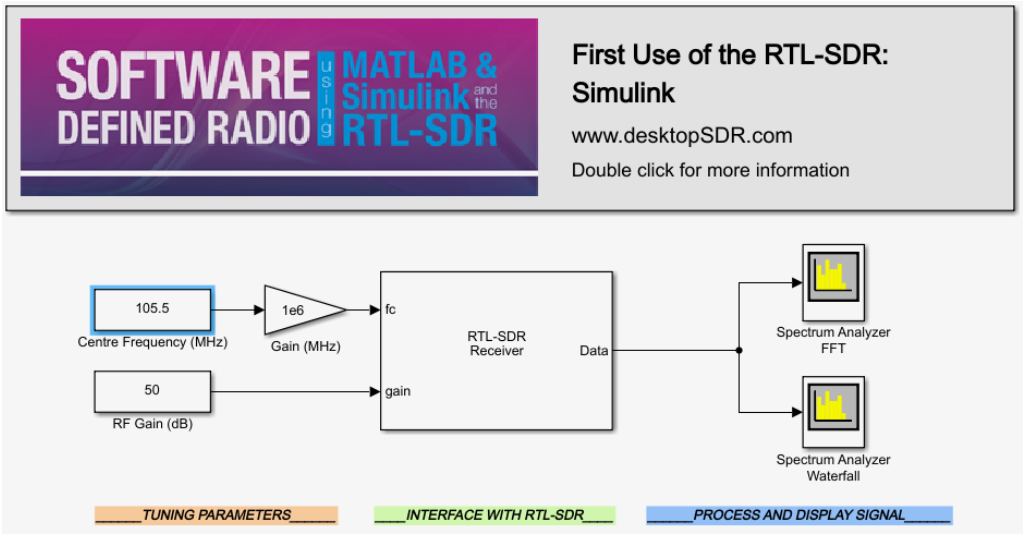


Figura 2.10 Modelo Simulink para utilizar el RTL-SDR.

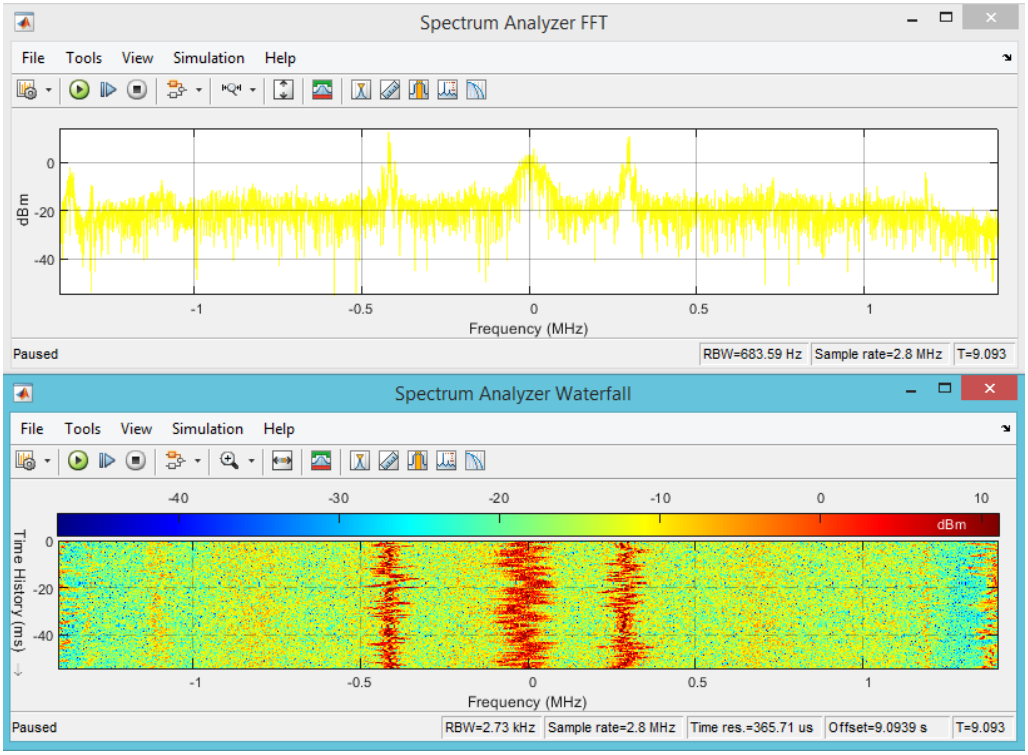


Figura 2.11 Captación de señal FM en Simulink.

A continuación se va a utilizar el ejemplo que viene en la carpeta *intro* llamado *rtlsdr_rx_startup_simulink.slx* que se ve en la Figura 2.10. En este se ha configurado la ganancia a 50 db y la frecuencia central a 105.5 MHz, donde se encuentra una emisora de radio FM. Con los dos bloques de analizador de espectros se consiguen las gráficas de la Figura 2.11.

Código 2.1 Configurar el RTL-SDR.

```
%% PARAMETROS
rtlsdr_id           = '0';          % RTL-SDR ID
rtlsdr_tunerfreq    = 105.5e6;        % RTL-SDR tuner frequency in Hz
rtlsdr_gain         = 50;           % RTL-SDR tuner gain in dB
rtlsdr_fs           = 2.4e6;        % RTL-SDR sampling rate in Hz
rtlsdr_frmlen       = 4096;         % RTL-SDR output data frame size
rtlsdr_datatype     = 'single';     % RTL-SDR output data type
rtlsdr_ppm          = 0;            % RTL-SDR tuner parts per million
    correction
sim_time            = 60;           % simulation time in seconds

%% rtl-sdr object
obj_rtlsdr = comm.SDRRTLReceiver(...
    rtlsdr_id,...
    'CenterFrequency', rtlsdr_tunerfreq,...
    'EnableTunerAGC', false,...
    'TunerGain', rtlsdr_gain,...
    'SampleRate', rtlsdr_fs, ...
    'SamplesPerFrame', rtlsdr_frmlen,...
    'OutputDataType', rtlsdr_datatype ,...
    'FrequencyCorrection', rtlsdr_ppm );
```

Código 2.2 Configurar los analizadores de espectro.

```
obj_specfft = dsp.SpectrumAnalyzer(...
    'Name', 'Spectrum Analyzer FFT',...
    'Title', 'Spectrum Analyzer FFT',...
    'SpectrumType', 'Power density',...
    'FrequencySpan', 'Full',...
    'SampleRate', rtlsdr_fs);
obj_specwaterfall = dsp.SpectrumAnalyzer(...
    'Name', 'Spectrum Analyzer Waterfall',...
    'Title', 'Spectrum Analyzer Waterfall',...
    'SpectrumType', 'Spectrogram',...
    'FrequencySpan', 'Full',...
    'SampleRate', rtlsdr_fs);
```

También es posible no utilizar Simulink, y controlar el RTL-SDR únicamente mediante comandos de MATLAB. Los códigos que se presentan a continuación se han obtenido de la carpeta *Intro*, de la función *rtlsdr_rx_startup_matlab.m*. En el Código 2.1 se configura el dispositivo. En los parámetros se controlan las mismas características que cuando se trabaja en Simulink. Con el comando *comm.SDRRTLReceiver* se introduce esa configuración en el hardware. Con el Código 2.2 se configuran los dos analizadores de espectros que ya se han visto en Simulink.

Código 2.3 Función para activar el RTL-SDR.

```
rtlsdr_frmtime = rtlsdr_frmlen/rtlsdr_fs;

%% SIMULATION

% check if RTL-SDR is active
if isempty(sdrinfo(obj_rtlsdr.RadioAddress))
    error(['RTL-SDR failure. Please check connection to ',...
        'MATLAB using the "sdrinfo" command.']);
```

```
end

% reset run_time to 0 (secs)
run_time = 0;

% run while run_time is less than sim_time
while run_time < sim_time

    % fetch a frame from the rtlsdr
    rtlsdr_data = step(obj_rtlsdr);

    % update spectrum analyzer windows with new data
    step(obj_specfft, rtlsdr_data);
    step(obj_specwaterfall, rtlsdr_data);

    % update run_time after processing another frame
    run_time = run_time + rtlsdr_frmtime;

end
```

Por último, el Código 2.3 pone en marcha la captación de señal. El resultado, Figura 2.12 y Figura 2.13, es el mismo que al utilizar Simulink. La decisión de cual de las dos formas utilizar dependerá de qué se quiera conseguir. En Simulink es más sencillo entender qué se está haciendo en cada momento por ser una interfaz clara. Al mismo tiempo, trabajar directamente con códigos es más rápido, aunque al principio sea más complicado.

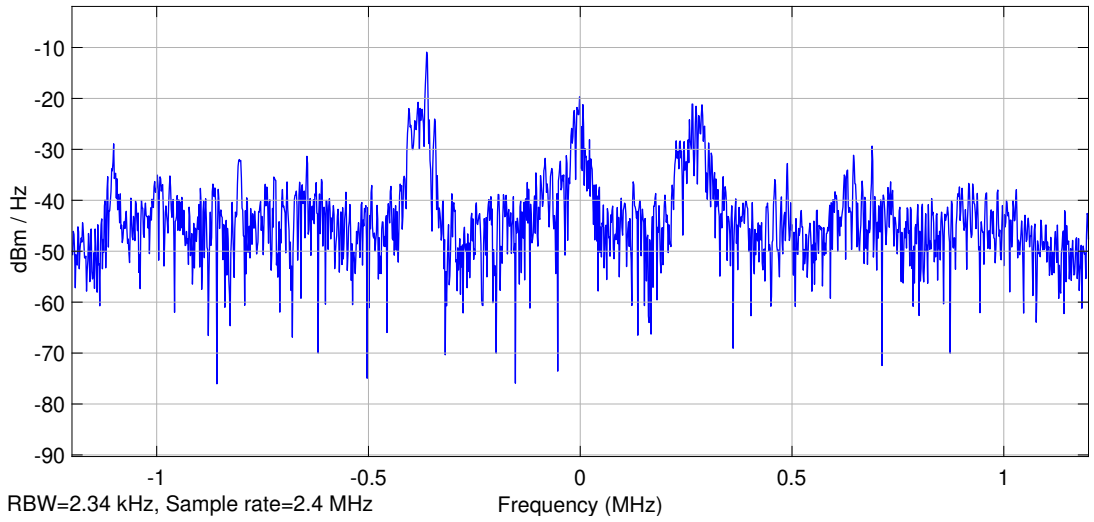


Figura 2.12 Captación de señal FM en MATLAB, analizador de espectro FFT .

Es importante destacar que, gracias a las toolbox instaladas se puede llegar a conseguir en MATLAB lo mismo que en los programas de receptores SDR. La librería que aporta [6] incluye varios demoduladores e incluso algunos codificadores que permitirían, con los dispositivos adecuados, no solo recibir sino también transmitir señales moduladas.

Uno de los problemas de utilizar MATLAB y Simulink es la carga que tiene sobre el ordenador. El utilizado para este trabajo no es especialmente potente, tiene 6 GB de memoria RAM y un procesador Intel Core i7 de 1.8 GHz. Por ello, el uso de MATLAB hace que la ocupación de la memoria RAM sea muy alta y ralentiza el funcionamiento. Si solo se quisiera ver el espectro o escuchar la radio es una mejor opción utilizar el SDRSharp. Por otro lado, en MATLAB se pueden guardar los datos I/Q y trabajar sobre ellos sin estar usando al mismo tiempo el RTL-SDR. Con MATLAB existe mayor libertad para investigar, demodular y trabajar con los datos I/Q, mientras que en otros programas solo se obtiene directamente el resultado final de

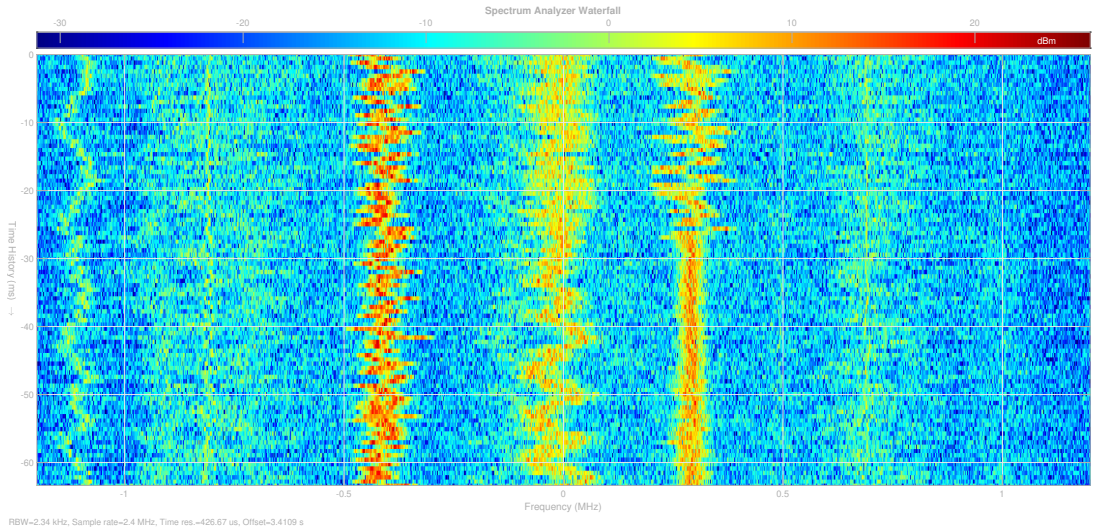


Figura 2.13 Captación de señal FM en MATLAB, analizador de espectro en cascada .

la demodulación. Mediante MATLAB y tecnología SDR se pueden realizar sistemas complejos como un decodificador de señales ADS-B para obtener las trayectorias de los aviones [52].

2.5.3 Procesamiento en Python

Python [53] es un lenguaje de programación de alto nivel orientado a objetos. Uno de los objetivos de este proyecto ha sido utilizar este lenguaje para controlar un dispositivo RTL-SDR. En el Capítulo 4 se explicará en profundidad el trabajo realizado con Python. En este sección solo se va a analizar qué es necesario para comunicarse con el dispositivo y qué nos permite hacer.

Como Python solo es un lenguaje de programación, para poder conectarse al RTL-SDR es necesario un paquete de códigos (llamados librerías) específico. *Pyrtlsdr* [54] es una librería creada por el usuario de GitHub [55] Roger [56]. Es una interfaz que permite utilizar el dispositivo mediante Python, solo es necesario conectar el RTL-SDR por USB y utilizar un intérprete de comandos de Python. El funcionamiento es muy parecido a cuando se trabaja con código en MATLAB. Con el Código 2.4 se configura el dispositivo, controlando los mismo parámetros que en MATLAB. La primera línea del código se explicará en su apartado correspondiente. Para captar una señal se usa la última linea del código, donde se indica el número de muestras que debe leer.

Código 2.4 Configuración RTL-SDR en Python.

```
from rtlsdr import *

sdr = RtlSdr()

sdr.sample_rate = 2.4e6
sdr.center_freq = 105.5e6
sdr.gain = 50
sdr.freq_correction = 93

samples = sdr.read_samples(256*1024)
```

En Python también se puede hacer que el dispositivo esté recogiendo datos de forma constante y así poder dibujar la señal en un analizador de espectros. El Código 2.5 se usa para obtener el analizador de espectros. También es posible obtenerlo en cascada, Código 2.6. Los dos se obtuvieron de [57]. En la Figura 2.14 y Figura 2.15 se muestra el resultado de estos códigos. Como se puede ver, mediante Python se pueden captar señales de la misma manera que en el resto del software estudiado. El espectro en cascada no tiene tan buena calidad como en el resto de opciones.

Código 2.5 Analizador de espectros en Python.

```
# Obtener una imagen estática
from matplotlib import pyplot as plt

plt.psd(samples, NFFT=1024, Fs=sdr.sample_rate/1e6, Fc=sdr.center_freq/1e6)
plt.xlabel('Frequency (MHz)')
plt.ylabel('Relative power (dB)')
plt.show()

#Obtener analizador de espectro FFT
import matplotlib.animation as animation

fig = plt.figure()
graph_out = fig.add_subplot(1, 1, 1)

def animate(i):
    graph_out.clear()
    samples = sdr.read_samples(128*1024)
    graph_out.psd(samples, NFFT=1024, Fs=sdr.sample_rate / 1e6, Fc=sdr.
    center_freq/1e6)

try:
    ani = animation.FuncAnimation(fig, animate, interval=10)
    plt.show()
except KeyboardInterrupt:
    pass
finally:
    sdr.close()
```

Código 2.6 Analizador de espectros en cascada..

```
from matplotlib import mlab as mlab
from rtlsdr import RtlSdr
import numpy as np
from PIL import Image
import time
import pygame

DISPLAY_WIDTH = 256
DISPLAY_HEIGHT = 200

sdr = RtlSdr()
# configure device
sdr.sample_rate = 2.4e6  # Hz
sdr.center_freq = 105.5e6  # Hz
#sdr.freq_correction = 60   # PPM
sdr.gain = 28


image = []

def get_data():
    samples = sdr.read_samples(16*1024)
    power, _ = mlab.psd(samples, NFFT=1024, Fs=sdr.sample_rate /
                        1e6)
```

```
25
26      max_pow = 0
27      min_pow = 10
28
29      # search whole data set for maximum and minimum value
30      for dat in power:
31          if dat > max_pow:
32              max_pow = dat
33          elif dat < min_pow:
34              min_pow = dat
35
36      # update image data
37      imagelist = []
38      for dat in power:
39          imagelist.append(mymap(dat, min_pow, max_pow, 0, 255))
40      image.append(imagelist[round(len(
41          imagelist)/2)-round(len(imagelist)/8): round(len(imagelist)/2)+round(
        len(imagelist)/8)])
42      if len(image) > 200:
43          image.pop(0)
44
45
46  def mymap(x, in_min, in_max, out_min, out_max):
47      return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
        )
48
49
50  pygame.init()
51  gameDisplay = pygame.display.set_mode((DISPLAY_WIDTH, DISPLAY_HEIGHT))
52  pygame.display.set_caption(f"DIY SDR")
53  clock = pygame.time.Clock()
54  background = pygame.Surface(gameDisplay.get_size())
55  background = background.convert()
56  background.fill((0, 0, 0))
57
58  game_quit = False
59
60  while not game_quit:
61
62      gameDisplay.blit(background, (0, 0))
63
64      for event in pygame.event.get():
65          if event.type == pygame.QUIT:
66              game_quit = True
67
68      get_data()
69      outimage = np.array(image, np.ubyte)
70      outimage = Image.fromarray(outimage, mode='L')
71      outimage = outimage.convert('RGBA')
72      strFormat = 'RGBA'
73      raw_str = outimage.tobytes("raw", strFormat)
74      surface = pygame.image.fromstring(raw_str, outimage.size, 'RGBA')
75      gameDisplay.blit(surface, (0, 0))
76      pygame.display.update()
77      clock.tick(60)
78
79  pygame.quit()
```

```
80
81 try:
82     pass
83 except KeyboardInterrupt:
84     pass
85 finally:
86     sdr.close()
```

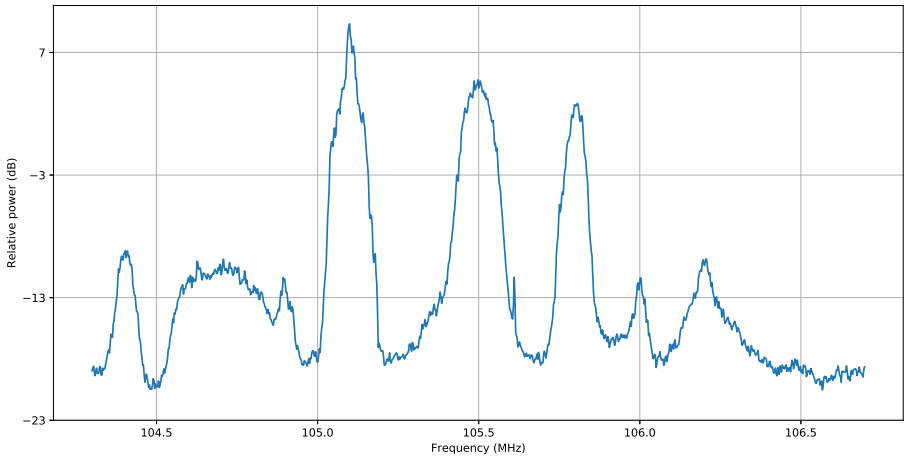


Figura 2.14 Captación de señal FM en Python, analizador de espectro FFT.

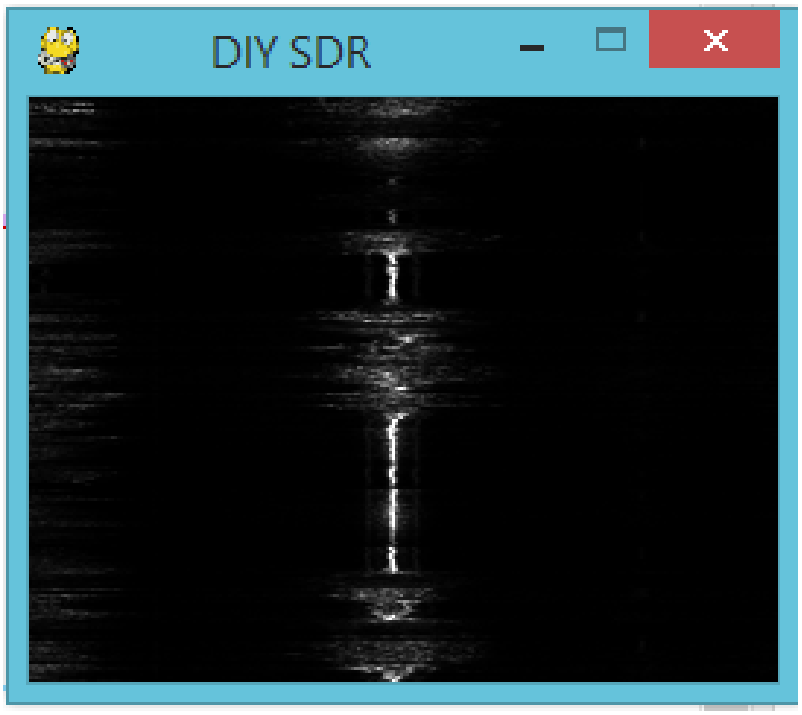


Figura 2.15 Captación de señal FM en Python, analizador de espectro en cascada .

2.6 Resumen

El RTL-SDR es un dispositivo barato que permite captar señales de una gran parte del espectro de radiofrecuencias. Es sencillo de utilizar y existen muchos programas que facilitan su uso. SDRSharp es una gran opción debido todas las herramientas que ofrece, además de tener una interfaz clara. MATLAB y Python ofrecen la posibilidad de trabajar directamente sobre los datos I/Q. MATLAB es un programa conocido, y gracias a Simulink se puede ver de forma más gráfica qué se está haciendo.

Una de las desventajas es que MATLAB sea de pago, aunque ahora se tiene acceso a la licencia del campus. Python, por otro lado, es un software libre con una gran base de usuarios que han creado una gran cantidad de librerías y códigos que se pueden utilizar en conjunto con el RTL-SDR. Es una herramienta bastante más versátil que MATLAB, aunque sea más complicada por tener que aprender a trabajar con cada librería. Otra diferencia destacable es la carga computacional de estos programas sobre el ordenador. Mientras que con MATLAB se han encontrado problemas que provocan un funcionamiento algo ralentizado, Python no exige tanta carga al ordenador. Al final no hay un software mejor que otro, cada uno tiene sus fortalezas y debilidades según qué se quiera hacer.

3 ACARS

Dado que la finalidad de este trabajo es crear una herramienta que detecte la señal ACARS, es debido comenzar explicando qué es el ACARS. El *Aircraft Communications Adressing and Reporting System* no es solo un tipo de señal, sino un sistema de comunicaciones muy utilizado en aviación para comunicarse con estaciones terrestres. ACARS establece una comunicación basada en caracteres ASCII por enlace de datos de muy alta frecuencia (VHF). El sistema se creó a finales de los años setenta con la intención de servir como enlace de datos que permitiera automatizar mensajes. La compañía estadounidense ARINC fue la encargada del desarrollo del sistema y aún hoy en día es la proveedora del servicio en los Estados Unidos. Originalmente se utilizó para sustituir las comunicaciones por voz VHF que podían automatizarse, ahorrando así tiempo a la tripulación. Con el tiempo se han incorporando comunicaciones HF y por satélite (SATCOM), estableciendo cobertura en todo el mundo.

El sistema ACARS puede dividirse en tres partes: el equipamiento del avión, el proveedor de servicios y las estaciones en tierra. La parte central del sistema a bordo es la *Management Unit* (MU), en sistemas más modernos la *Communications Management Unit* (CMU), y actúa como un router. Todos los mensajes que salgan o lleguen al avión pasan por esta unidad. Esta conectada a varios sistemas del avión, como el *Flight Management System* y el ordenador de manteminiento. Por otro lado, está conectada a transmisores VHF para permitir comunicaciones ACARS VHF, y a la Unidad de Datos Satélite (SDU) para acceder a SATCOM [58]. La MU solo se dedica a dirigir los mensajes a las antenas o sistema finales que se indiquen en el mensaje, no junta la información que venga en paquetes separados. La interfaz con la tripulación es la unidad multifunción de control y datos (MCDU). El proveedor de servicios de enlaces de datos es el encargado de que las transmisiones lleguen a sus destinos. Aportan los servicios VHF, HFDL y SATCOM. En Estados Unidos el proveedor principal es ARINC, en Europa, Asia y América del Sur lo provee SITA. En tierra se encuentra el receptor interesado, ya sea la compañía o los servicios ATC.

3.1 Mensajes

Los mensajes ACARS se transmiten en dos direcciones, de subida (desde tierra al avión) y de bajada. Esto permite que algunas aplicaciones puedan ser interactivas, desde respuestas de los receptores del mensaje (para confirmación de mensaje recibido) hasta conversaciones. Pueden llegar a enviarse más de 20 millones de mensajes al mes, con más de 10.000 aviones utilizando el sistema [1]. Los mensajes contienen todo tipo de información y varios destinatarios [59]:

- *Servicios de información.* ACARS permite obtener la información que divulga el Servicio de Información Aeronáutica (AIS), relacionados con condiciones temporales o restricciones del espacio aéreo.
- *Planes de vuelo.* También lo utilizan las compañías aéreas para enviar los planes de vuelo con información sobre las rutas a seguir.
- *Posición.* Aunque lo habitual es utilizar el sistema ADS-B, algunos aviones hacen uso del ACARS para transmitir su posición.
- *Diagnósticos de mantenimiento.* Una de las opciones más utilizadas, muchos sistemas del avión (como los motores) envían información sobre su estado automáticamente. Esto permite que los departamentos de mantenimiento de las compañías puedan preparar con antelación el trabajo que tendrán que realizar cuando el avión aterrice.

- *Información administrativa.* Comunicación con la compañía sobre horarios de la tripulación, retrasos, problemas, etc.
- *Coordinación con el aeropuerto.* Para informar sobre qué servicios se van a utilizar, como si es necesario repostar.
- *Operaciones de vuelo.* Se utiliza en comunicación con el control de tráfico aéreo (ATC) para peticiones no urgentes.
- *Eventos OOOI.* Desde el principio ACARS se ha utilizado para comunicar a la compañía el momento de realización de los llamados eventos OOI. Estos eventos son los momentos de aterrizaje, despegue, salida del puesto de estacionamiento y otros. La información va acompañada por marcas de tiempo. La compañía puede así tener conocimiento sobre dónde se encuentra cada avión.
- *Mensajes libres.* A veces es necesario transmitir información no directamente relacionada con ninguno de los temas explicados. ACARS permite el envió de todo tipo de texto.

Los mensajes tienen hasta 220 caracteres, cada uno formado por siete bits de información en formato ASCII y otro bit de paridad. Para transmitir mensajes más largos se pueden utilizar varios paquetes. Se va a explicar cómo es la estructura de un mensaje ACARS enviado desde un avión [58]:

Tabla 3.1 Estructura de un mensaje ACARS.

	Cabecera						
Nombre	SOH	Mode	Adress	TAK	Label	DBI	STX
Nº de caracteres	1	1	7	1	2	1	1

	Texto			Cola		
Nombre	MSN	Flight ID	Texto	Suffix	BCS	BCS suffix
Nº de caracteres	4	6	0-210	1	2	1

- *Start of Header (SOH).* Caracter que indica el inicio de la cabecera.
- *Mode Character.* Se elige en función del código de acceso del sistema en tierra preferido.
- *Adress.* Es la identificación del avión, ya sea la identificación de vuelo o matrícula registrada de la aeronave.
- *Positive Technical Acknowlegdment.* Para indicar recepción del mensaje.
- *Label.* Identifica el tipo de mensaje mediante 2 caracteres.
- *Downlink Block Identifier (DBI).* Identifica que el mensaje se envía desde el aire a tierra.
- *End of Preamble (STX).* Indica finalización de la cabecera.
- *Texto.* En este bloque se encuentra la información a transmitir, es de escritura libre hasta un máximo de 220 caracteres. Esto puede aumentar hasta 3.520 para comunicación por satélite.
- *Message Sequence Number (MSN) and Flight ID.* Caracteres incluidos en el texto informando sobre la línea aérea y el identificador del vuelo.
- *Suffix.* Caracter indicando final del texto.
- *Block check sequence (BCS).* 16 bits utilizados para corrección de errores.
- *BCS suffix.* Caracter que permite decodificar el BCS.

3.2 Redes

Al mismo tiempo que ACARS se hizo indispensable para las operaciones aéreas debido a su creciente utilidad, las limitaciones del enlace VHF también ganaron importancia. Estas limitaciones eran principalmente debidas al poco alcance y velocidad de transmisión. Se introdujeron dos nuevos enlaces de datos, por satélite (SATCOM) y por HF. Al principio se utilizaron satélites Inmarsat y ahora también la red de satélites IRIDIUM. El enlace de datos de alta frecuencia (HFDL) es también un enlace de largo alcance, esta vez con cobertura en todo el planeta. Hoy en día existen cuatro subredes ACARS en funcionamiento [1]:

- *VHF.* Es la red original de ACARS. Hace uso de canales de 25Khz en las frecuencias de 118 a 137 MHz. Usa la modulación *minimun shift keying* (MSK) de la que se hablará en profundidad más adelante. El protocolo de comunicaciones es *carrier-sensed multiple access* (CSMA). Antes de transmitir se escucha el canal y si este está ocupado se espera hasta que termine la transmisión antes de enviar

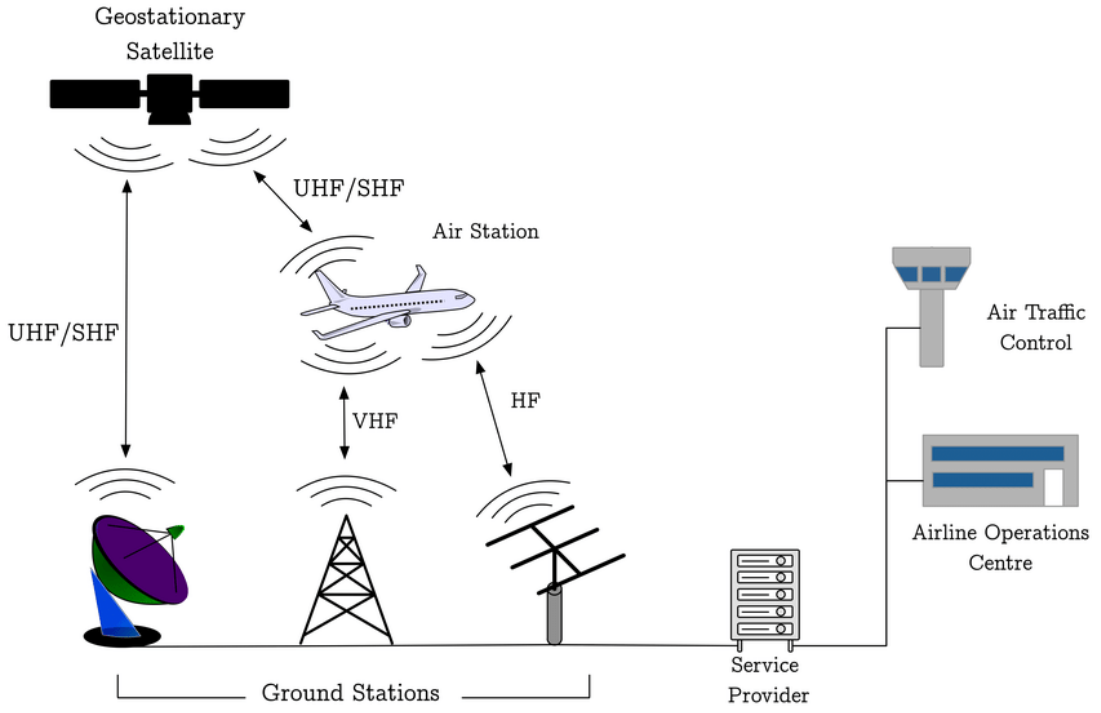


Figura 3.1 Representación de las redes ACARS [60].

la propia. De esta forma se evita que dos transmisiones sucedan a la vez. Tiene una velocidad de transmisión de 2.4 Kbps.

- *Satcom.* Lo componen las redes de satélites Inmarsat e Iridium. Inmarsat opera en la banda L, sobre frecuencias de 1 GHz. Los satélites reciben los paquetes transmitidos y los redirigen al receptor indicado. Inmarsat permite un enlace de datos de larga distancia, el único lugar del planeta donde no ofrece cobertura es el Polo Norte. Con este enlace se establecieron comunicaciones oceánicas directamente entre pilotos y controladores. Iridium ofrece cobertura de llamadas telefónicas y enlace de datos en cualquier parte del planeta.
- *HFDL.* Utiliza canales en la banda de frecuencias de voz HF. Hace uso de tres métodos de modulación *phase-shift* diferentes en función de la velocidad de datos a transmitir. El protocolo de comunicaciones es esta vez *time division multiple access* (TMDA). Estas diferencias respecto a la red VHF se deben a la necesidad de maximizar el alcance, mitigando las perdidas y el ruido. Aún así, son necesarias grandes antenas para poder utilizar esta red.
- *VDL Mode 2.* Opera en la misma frecuencia que la red VHF. Existen varios canales reservados. Utiliza modulación diferencial PSK de 8 niveles (D8PSK). Con una velocidad de transmisión de 31.5 kbps tiene unas 10 veces más capacidad que los canales VHF normales. Es, por tanto, una mejora de la red VHF original, más rápida y que permite reducir la congestión de los canales.

Cada red tiene sus ventajas y sus limitaciones. SATCOM y HFDL ofrecen mayor cobertura que las redes VHF, pero tienen mayor coste y complejidad. HF tiene máxima cobertura y puede hacer uso simultáneo de varios canales, pero la velocidad de transmisión es muy baja [61]. SATCOM ofrece mayor velocidad, pero es susceptible a interrupciones y no siempre está disponible a grandes altitudes. Con todas estas diferencias en cuenta, ninguna de las redes se ha impuesto sobre las demás, usándose todas ellas, según cúal sea más conveniente en cada situación.

3.3 Modulación

Se va a analizar en concreto la modulación que se realiza en la transmisión VHF, por ser las que se pueden captar con la configuración de RTL-SDR y antena disponibles en este trabajo. Los mensajes ACARS se transmiten en la banda VHF en canales de 25 KHz, donde los datos se transmiten a una velocidad de 2.4

kbps. Los estándares utilizados en la comunicación a través de ACARS vienen pautados por ARINC-618 [62]. La señal se modula mediante *minimun shift keying* (MSK).

MSK es un tipo de modulación CPFSK. La modulación FSK, *frequency shift keying*, consiste en que los símbolos "0" y "1" son transmitidos por dos señales sinusoidales cada una a una frecuencia concreta. La *continuous phase FSK* es una modulación más avanzada de FSK, donde se hace que, al cambiar de frecuencia (por cambiar el símbolo a transmitir), no haya variación de fase. Es decir, en una modulación CPFSK la fase de la señal modulada es constante, no existen cortes. Finalmente, la MSK es una versión más compleja. Consiste en que la desviación de frecuencia, la diferencia entre las dos frecuencias utilizadas, sea igual a la mitad de la tasa de bits transmitidos [63]. Es un tipo de modulación muy eficiente respecto al ancho de banda utilizado.

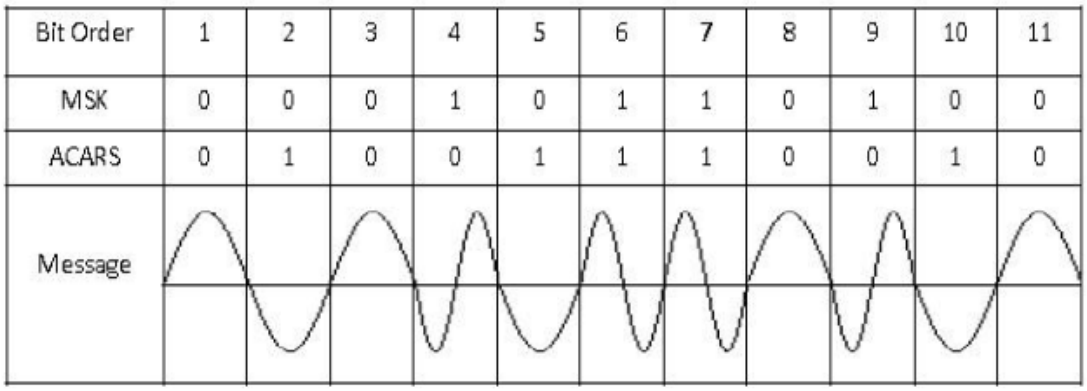


Figura 3.2 Comparación de modulación MSK y ACARS [64].

La señal ACARS consiste en dos subportadoras no simultáneas. Son dos tonos sinusoidales a 1.2 kHz y a 2.4 kHz. Se usa la primera cuando el bit a transmitir es el mismo que el enviado el instante anterior, y la segunda cuando el bit es diferente [61]. En la Figura 3.2 se puede ver la forma de utilizar esta modulación según qué bit se quiera transmitir. En la Figura 3.3 se muestra un ejemplo real de señal ACARS, donde se aprecia la modulación MSK. Al transmitir los datos se usa esta señal MSK para modular la amplitud de la portadora del canal VHF. Así, se obtiene una señal AM en doble banda lateral (DSB). La modulación utilizada se denomina DSB-AM-MSK.

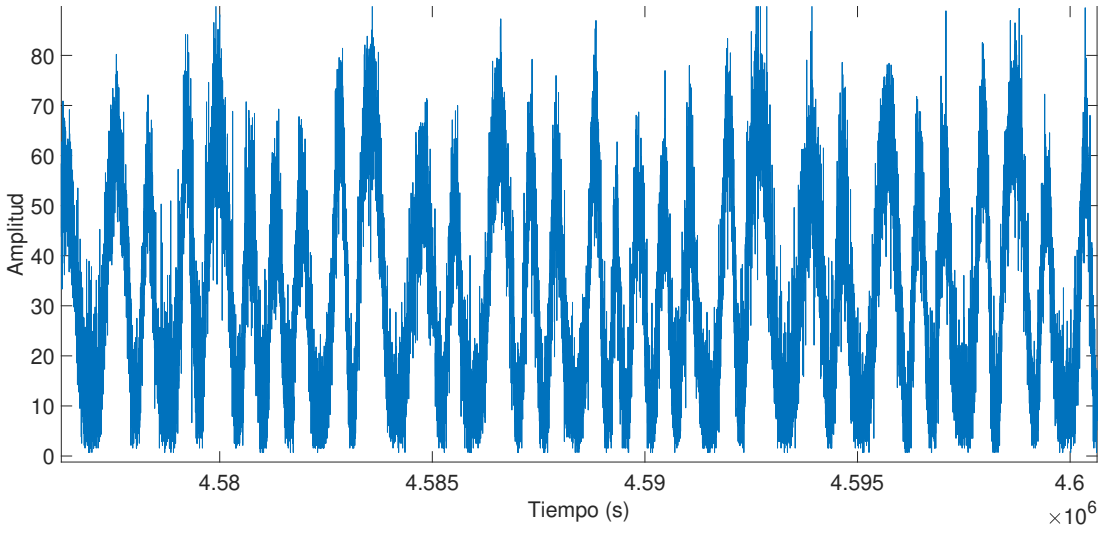


Figura 3.3 Ejemplo de señal ACARS en banda base.

3.4 Resumen

El sistema ACARS tiene muchas ventajas: comunicación directa entre controlador y piloto, automatización de informes de posición y peticiones de autorización, alcance global, aplicaciones en mantenimiento y muchas otras. Es un sistema muy utilizado, aunque también tiene algunos problemas. El más grave siendo el relacionado con la privacidad y seguridad de los mensajes. Varios estudios se han centrado en este punto, ya sea para crear un sistema más seguro [65], analizar los problemas por la facilidad de acceso a información sensible [59] o demodular los mensajes al no estar encriptados [64]. Al principio no era un gran problema debido al difícil acceso a herramientas que permitieran captar las señales. Con los rápidos avances en tecnología ahora es muy sencillo recibirlas. Con un ordenador normal, un SDR (ahora muy baratos) y una pequeña antena se pueden captar transmisiones ACARS de forma muy sencilla [66]. Es más, existen una gran variedad de programas que permiten también descodificar la señal y obtener la información. Varios ejemplos de esto pueden encontrarse en [67].

En este trabajo se ha decidido utilizar un RTL-SDR para capturar señales ACARS, siempre sin intentar decodificar la señal. El propósito es crear una aplicación que sea capaz de procesar una captura de señal, reconocer si se han producido comunicaciones a través de ACARS y obtener el tiempo de uso del canal. Gracias a esta aplicación se podría analizar cuánto se utilizan los canales, si están muy congestionados y, después de haber visto la importancia de la información que se transmite por ellos, reconocer el riesgo de utilizar el sistema sin añadir mayor seguridad. Esa era la intención inicial, pero, debido a la situación provocada por el coronavirus, no ha sido posible realizar capturas en canales ACARS al no haber casi ningún vuelo durante la realización de este trabajo. Por ello la herramienta creada no se ha podido utilizar para obtener estadísticas que analizar.

4 Python

Python es un lenguaje de programación de alto nivel. Está orientado a objetos y enfocado a que sus códigos sean fáciles de leer y comprender. Dado que la programación es importante y en la carrera solo se aprende a usar MATLAB y no lenguaje C, se decidió tomar como uno de los objetivos de este trabajo aprender a usar Python, que no es tan complicado como C y tiene algunas similitudes con MATLAB. Python es un lenguaje muy utilizado en programación en el mundo de la ingeniería y la ciencia. Hay numerosos libros que exploran las posibilidades de Python [68], [69] y [70]. En este capítulo se va a explicar cómo instalar y utilizar Python, explicar las librerías utilizadas durante el trabajo, exponer unos códigos creados para el uso del RTL-SDR y analizar las ventajas de este lenguaje de programación.

No se va a explicar el lenguaje en sí, es decir, las categorías, clases, sintaxis módulos, bucles y funciones. Es un tema muy extenso y no se tiene la suficiente experiencia como para explicarlo con claridad. Existen libros [70] y tutoriales [71] donde se trata esta cuestión.

4.1 Intérpretes

Al ser un lenguaje de programación gratuito, se puede obtener de [53] donde hay versiones para todos los sistemas operativos. Para este trabajo se ha utilizado un ordenador con Windows 8.1. La instalación se produce sola, no hay más que seguir los pasos del ejecutable. Para utilizar Python es necesario un intérprete. Con la instalación por defecto se obtiene el entorno de desarrollo IDLE [72], que permite un uso sencillo. Con IDLE se pueden crear archivos .py que son como los scripts .m de MATLAB. Se puede escribir el código y ejecutarlo en un Python Shell de forma separada a otros códigos.

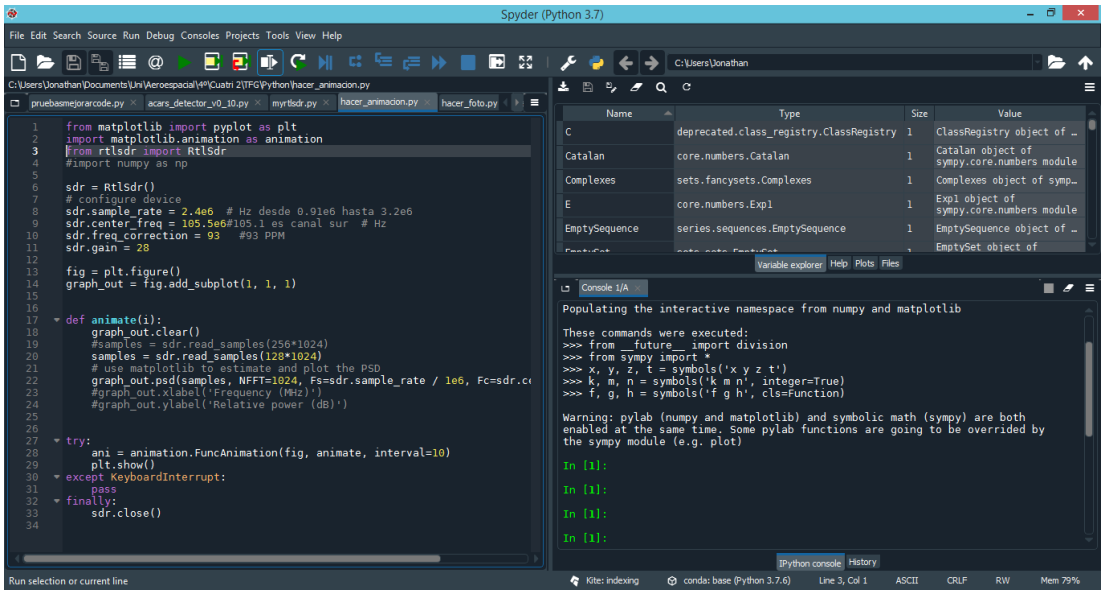


Figura 4.1 Interfaz de Spyder.

Este entorno de programación es sencillo de utilizar pero tiene muy pocas opciones que ayuden al usuario. Para este trabajo se ha decidido utilizar la plataforma Anaconda [73] que contiene una gran cantidad de paquetes para Python y ofrece mucha facilidad a la hora de instalar nuevas librerías. Como intérprete se ha utilizado Spyder [74]. Ofrece ayuda al editar, depurar código, trae instaladas librerías importantes, etc. Es gratuito, y enfocado a computación y análisis de datos. La interfaz de Spyder se muestra en la Figura 4.1 y es bastante similar a MATLAB. En la parte de arriba se encuentran todas las opciones, se usan especialmente los botones de nuevo archivo, abrir, ejecutar y depurar. A la izquierda se encuentra el editor, donde se trabaja sobre los archivos o scripts. Estos se ejecutan en la consola, a la derecha, donde se pueden introducir comandos directamente. La consola es como el *command window* de MATLAB. Encima de la consola se encuentra el explorador de variables, donde se muestran todas las variables de la sesión.

4.2 Librerías

Ahora que se tiene Spyder, en esta sección se van a explicar las librerías que se han utilizado durante el trabajo. Para instalar cualquier librería solo hay que seguir la guía [75]. Es rápido, si la librería está dentro de la base de datos de Anaconda basta con introducir en la consola de Spyder la línea *conda install scipy*, donde *scipy* es un ejemplo de librería. Si esto diera problemas hay que utilizar en instalador propio de Python llamado *pip*. Con la línea *conda install pip* se instala el instalador en Anaconda. Una vez esté incluido solo es necesario poner *pip install scipy* y la libería se instalará.

Para usar una librería en el código es necesario importarla con la línea *import nombre_libreria*. Esto se puede hacer de varias maneras, importándola entera, solo las partes que se van a utilizar, con un nombre concreto, etc. El cómo hacerlo suele depender de la librería en concreto y de la convención a la que hallan llegado los usuarios de la misma.

4.2.1 Pyrtlsdr

La librería *pyrtlsdr* [54] permite comunicarse con el RTL-SDR. Ha sido desarrollada por Roger, un usuario de GitHub. Mediante comandos Código 4.1 introducidos con esta librería se configura el dispositivo y se pueden obtener capturas del espectro de radiofrecuencias. La información entra directamente como datos I/Q. El cómo tratarlos depende enteramente de los comandos que se introduzcan después de la captura. Es por ello que son necesarias otras librerías relacionadas con el tratamiento de datos y las comunicaciones digitales.

Código 4.1 Ejemplo del uso de pyrtlsdr.

```
1  from rtlsdr import RtlSdr
2
3  sdr = RtlSdr()
4
5  sdr.sample_rate = 2.4e6
6  sdr.center_freq = 105.5e6
7  sdr.gain = 50
8  sdr.freq_correction = 93
9
10 samples = sdr.read_samples(256*1024)
```

Como se ve en la primera línea del Código 4.1, al importar esta librería es suficiente con coger el comando *RtlSdr* que permite controlar el dispositivo. Por convención se utiliza la abreviación *sdr* para referirse al RTL-SDR en el código.

Como los objetos que se introducen con cada librería tienen sus propias características, funciones y atributos, Python tiene una forma sencilla de obtenerlos. Se va a usar el ejemplo del Código 4.1. Basta con escribir *dir(sdr)* (una vez se haya declarado qué es sdr) en la consola para conseguir una lista de sus atributos. Otra forma de obtenerlos se consigue gracias a Spyder. Al escribir *sdr.* y darle al tabulador se obtiene un desplegable con los atributos de ese objeto. Para saber qué hace cada atributo se usa una línea de código como esta: *sdr.gain?*. Gracias a la interrogación aparece la información sobre ese atributo concreto, Figura 4.2.

```
In [9]: sdr.gain?
Type:        property
String form: <property object at 0x05DC56F0>
Docstring:
float or str: Get/Set gain of the tuner (in dB)

Notes:
    If set to 'auto', AGC mode is enabled; otherwise gain is in dB.
    The actual gain used is rounded to the nearest value supported by
    the device (see the values in :attr:`valid_gains_db`).
```

Figura 4.2 Ejemplo de cómo obtener información de los atributos.

4.2.2 Matplotlib

Matplotlib [76] es una de las librerías de Python más populares a la hora de representar datos en gráficos 2D y 3D. Es un paquete muy grande, con muchas utilidades diferentes, pudiendo crear gráficas estáticas, animadas e incluso interactivas. En este trabajo se ha utilizado dos de sus módulos: *pyplot* y *animation*. El primero para las gráficas normales y estáticas, con código similar al de MATLAB, y el segundo para las animadas, Código 4.2. Esta librería se ha elegido no solo por su popularidad, sino también por tener acceso a herramientas para representar señales de radiofrecuencia de distintas formas [77]. Y, en concreto, por la función *psd* [78] que permite representar la densidad espectral de potencia introduciendo como argumentos una muestra de datos I/Q. Matplotlib es una librería muy extensa y para este trabajo solo ha sido posible aprender lo necesario para obtener las gráficas relacionadas con el RTL-SDR. Para aprender a utilizarla se ha hecho uso principalmente de los tutoriales oficiales [79]. Si se quiere profundizar sobre cómo funciona y cómo utilizar Matplotlib lo mejor es dirigirse a los libros [80] y [81], donde se analiza esta librería en detalle.

Código 4.2 Ejemplo de código de matplotlib.

```
# Obtener una imagen estática
from matplotlib import pyplot as plt

plt.psd(samples, NFFT=1024, Fs=sdr.sample_rate/1e6, Fc=sdr.center_freq/1e6)
plt.xlabel('Frequency (MHz)')
plt.ylabel('Relative power (dB)')
plt.show()

#Obtener imagen animada
import matplotlib.animation as animation

fig = plt.figure()
graph_out = fig.add_subplot(1, 1, 1)

def animate(i):
    graph_out.clear()
    samples = sdr.read_samples(128*1024)
    graph_out.psd(samples, NFFT=1024, Fs=sdr.sample_rate / 1e6, Fc=sdr.
    center_freq/1e6)

try:
    ani = animation.FuncAnimation(fig, animate, interval=10)
    plt.show()
except KeyboardInterrupt:
    pass
finally:
    sdr.close()
```

4.2.3 Numpy

Numpy [82] es la librería estándar para programación científica en Python. Esta librería numérica provee a Python con la estructura de datos *ndarray*, es decir, de matrices y todas las operaciones que se puedan realizar con ellas. Esencialmente se podría decir que es como introducir MATLAB en Python. Es cierto que en Python también existen las matrices, pero Numpy está codificado de forma que todas las operaciones son mucho más rápidas que la versión base. Además, Numpy permite la vectorización, el poder operar directamente sobre los vectores completos sin necesidad de utilizar comandos especiales por tratarse de matrices. Por último, una ventaja de Numpy, y la mayor razón para utilizarla en este trabajo, es que tiene muchas funciones matemáticas esenciales para el tratamiento de señales como calcular la media y funciones trigonométricas. Como guía para aprender a utilizar Numpy se ha seguido la oficial [83]. Un buen libro para complementar ese conocimiento es [84].

Código 4.3 Ejemplo de código de Numpy.

```
import numpy as np

np.zeros(31,dtype=int)
Out[1]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0])

np.diag([1,7,-3])
Out[2]:
array([[ 1,  0,  0],
       [ 0,  7,  0],
       [ 0,  0, -3]])
```

4.2.4 Scipy

Scipy [85] es una librería de computación científica. Contiene módulos más específicos y optimizados que Numpy y Python. Trae funciones muy utilizadas en la ingeniería, relacionadas con optimización, integración e interpolación por ejemplo. Está preparada para ser usada en conjunto con Numpy, pudiendo utilizar sin problema sus estructuras de datos. Se ha utilizado durante el trabajo ya que ofrece funciones muy útiles en el procesamiento de señales: diezmado, creación y aplicación de filtros, convolución y análisis espectral entre otros. La guía oficial [86] ha sido útil a la hora de entender cómo usar esta librería.

Código 4.4 Ejemplo de código de Scipy.

```
import scipy.io
acars_signal = scipy.io.loadmat('se_acars_6_MENSA')

import scipy.signal as sig
sig.lfilter(CIC['Num'],CIC['Den'],señal_acars)
```

En concreto se van a utilizar dos módulos: *scipy.io* y *scipy.signal*. En el Código 4.4 se puede ver un ejemplo de su uso. El módulo *io* sirve para leer y obtener datos de ficheros guardados en formato .mat de MATLAB. Es muy útil a que en este proyecto se trabaja conjuntamente con Python y MATLAB. El módulo *signal* es el que contiene las funciones para trabajar con señales. Se ha utilizado principalmente para la aplicación de filtrados a la señal con la que se trabaja.

4.2.5 PyQtGraph

PyQtGraph [87] es una librería gráfica similar a Matplotlib. No tiene tanto tiempo ni base de usuarios como esta última. Sin embargo, puede llegar a ser más fácil de utilizar y, sobre todo, funciona más rápido. Esta es la razón por la que se ha utilizado en el trabajo, donde la rapidez al representar datos es importante. Para aprender a utilizarla se ha seguido la guía [88]. Permite crear gráficos interactivos que responden muy

rápidamente, seleccionar regiones en una gráfica y obtener su ampliación en otra e incluso crear gráficas a tiempo real más fácilmente que con Matplotlib. También permite exportar las gráficas en una gran cantidad de formatos sin más que hacer click derecho en la misma.

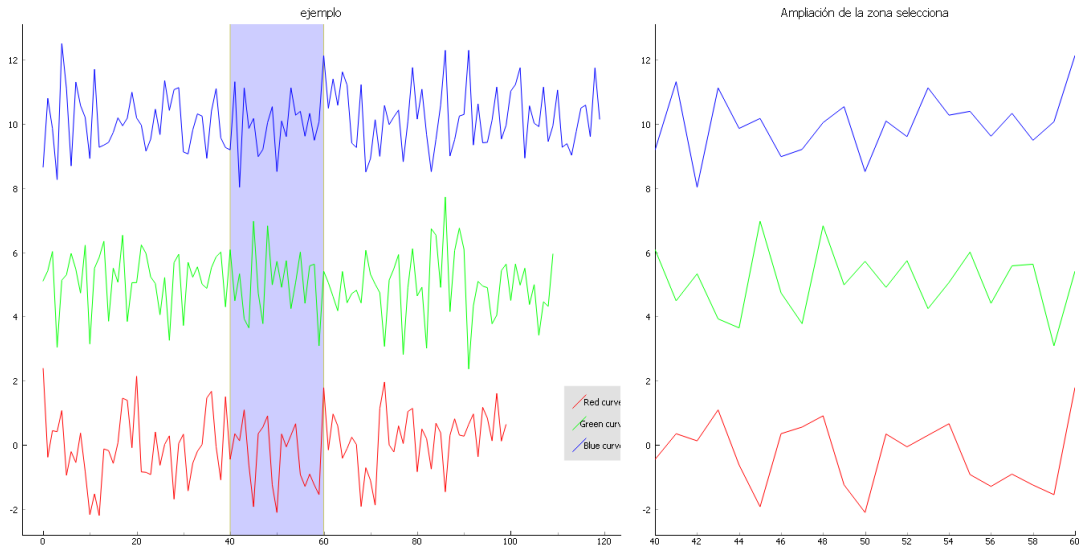


Figura 4.3 Ejemplo de gráfica con PyQtGraph.

Como es la librería de la que se van a obtener las gráficas de Python que aparezcan en este trabajo, se explica ahora un ejemplo. La Figura 4.3 se obtiene con el Código 4.5. La zona que aparece ampliada se puede mover con el ratón y la ampliación se actualiza automáticamente. Sin embargo, una de sus desventajas consiste en no poder cambiar el tamaño de títulos, nombres y ejes.

Código 4.5 Ejemplo de código de PyQtGraph.

```
from pyqtgraph.Qt import QtGui, QtCore
import numpy as np
import pyqtgraph as pg

app = QtGui.QApplication([])

# Configuración de colores de la figura
pg.setConfigOption('background', 'w')
pg.setConfigOption('foreground', 'k')

# Crea la ventana
win = pg.GraphicsWindow(title="Basic plotting examples")
win.resize(1000,600)

# Activa antialiasing
pg.setConfigOptions(antialias=True)

# Dibuja la primera gráfica
p2 = win.addPlot(title='ejemplo')
p2.addLegend()
data1 = np.random.normal(size=100)
data2 = np.random.normal(size=110)+5
data3 = np.random.normal(size=120)+10
p2.plot(data1, pen=(255,0,0), name="Red curve")
p2.plot(data2, pen=(0,255,0), name="Green curve")
```

```
p2.plot(data3, pen=(0,0,255), name="Blue curve")

# Configura la selección de región a ampliar
lr = pg.LinearRegionItem([40,60])
lr.setZValue(-10)
p2.addItem(lr)

# Dibuja la sección ampliada
p1 = win.addPlot(title="Ampliación de la zona selecciona")
p1.plot(data1,pen=(255,0,0))
p1.plot(data2,pen=(0,255,0))
p1.plot(data3,pen=(0,0,255))
def updatePlot():
    p1.setXRange(*lr.getRegion(), padding=0)
def updateRegion():
    lr.setRegion(p1.getViewBox().viewRange()[0])
lr.sigRegionChanged.connect(updatePlot)
p1.sigXRangeChanged.connect(updateRegion)
updatePlot()

## Ejecuta el código y mantiene los bucles activos
if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
```

4.3 Códigos propios

En esta sección se encuentran varios códigos para Python. Son propios, realizados por el autor del trabajo, y sirven para automatizar y simplificar el uso del RTL-SDR en Python. Se ha creado un módulo llamado *myrtlsdr* que, una vez importado, permite realizar varias acciones sobre el dispositivo con funciones simples de utilizar.

El módulo *myrtlsdr*, Código 4.6, introduce 6 funciones para el uso del RTL-SDR, aunque se han separado en varios códigos por claridad. Para ello es necesario importar bastantes librerías diferentes, que ya se acaban de comentar en este capítulo. La función *rtl_config()* sirve para configurar los parámetros del dispositivo, permitiendo hacerlo en una sola linea, y será utilizada en el resto de funciones.

Código 4.6 myrtlsdr.

```
"""Código propio para el uso del RTL-SDR.

Contiene varias funciones para usar el RTL-SDR:
    rtl_config
    matplot_spectrum
    plot_spectrum
    cont_spectrum
    freq_sweep
    save_IQ
"""

import time
import scipy.io
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt
```

```
from matplotlib.mlab import psd
import pyqtgraph as pg
from pyqtgraph.Qt import QtGui, QtCore
from rtlsdr import RtlSdr


def rtl_config(sdr, center_freq, sample_rate, gain, freq_correction):
    """ Configura los parámetros básicos del RTL-SDR.


    Parámetros de entrada
    ----------
    sdr: RtlSdrAio
        Nombre dado al dispositivo conectado.
    center_freq: float
        Frecuencia central de recepción en Hz.
    sample_rate: float
        Frecuencia de muestreo en Hz.
    gain: float or str
        Ganancia del dispositivo en dB.
    freq_correction: int
        Desviación de frecuencia en PPM.
    """
    sdr.sample_rate = sample_rate
    sdr.center_freq = center_freq
    sdr.gain = gain
    sdr.freq_correction = freq_correction
```

El Código 4.7 contiene dos funciones con la misma finalidad, representar la densidad espectral de potencia tras una lectura de la frecuencia especificada. La diferencia radica en que en la primera se utiliza la librería *matplotlib* para la representación gráfica, y en la segunda se usa la librería *PyQtGraph*. Se recomienda usar *plot_spectrum* ya que *PyQtGraph* es algo más rápida que *matplotlib*. De hecho, en el resto de funciones las gráficas se obtienen con *PyQtGraph*.

Código 4.7 matplot_spectrum() y plot_spectrum().

```
def matplot_spectrum(center_freq, sample_rate, gain=28,
                     samples_number=256*1024, freq_correction=93,
                     device_index=0):
    """ Obtiene una instantánea en la frecuencia seleccionada y representa
    su densidad espectral de potencia con matplotlib.


    Parámetros de entrada
    ----------
    center_freq: float
        Frecuencia central de recepción en Hz.
    sample_rate: float
        Frecuencia de muestreo en Hz.
    gain: float or str, opcional
        Ganancia del dispositivo en dB.
    samples_number: int, opcional
        Número de muestras a leer.
    freq_correction: int, opcional
        Desviación de frecuencia en PPM.
    device_index: int, opcional
        Número de puerto del dispositivo.
```

```
    """
    sdr = RtlSdr(device_index)
    rtl_config(sdr, center_freq, sample_rate, gain, freq_correction)
    samples = sdr.read_samples(256*1024)
    sdr.close()

    plt.psd(samples, NFFT=1024, Fs = sample_rate/1e6, Fc = center_freq/1e6)
    plt.title('Power espectral density')
    plt.xlabel('Frequency (MHz)')
    plt.ylabel('Relative power (db)')
    plt.show()

def plot_spectrum(center_freq, sample_rate, gain=28, samples_number=256*1024,
                  freq_correction=93, device_index=0):
    """ Obtiene una instantánea en la frecuencia seleccionada y representa
    su densidad espectral de potencia con PyQtGraph.


    Parámetros de entrada
    ----------
    center_freq: float
        Frecuencia central de recepción en Hz.
    sample_rate: float
        Frecuencia de muestreo en Hz.
    gain: float or str, opcional
        Ganancia del dispositivo en dB.
    samples_number: int, opcional
        Número de muestras a leer.
    freq_correction: int, opcional
        Desviación de frecuencia en PPM.
    device_index: int, opcional
        Número de puerto del dispositivo.
    """
    pg.setConfigOption('background', 'w')
    pg.setConfigOption('foreground', 'k')
    pg.setConfigOptions(antialias=True)

    sdr = RtlSdr(device_index)
    rtl_config(sdr, center_freq, sample_rate, gain, freq_correction)
    samples = sdr.read_samples(256*1024)
    sdr.close()

    power, psd_freq = psd(samples, NFFT=1024, Fs=sample_rate/1e6)
    psd_freq += center_freq/1e6

    p1 = pg.plot(psd_freq, 10*np.log10(power), pen='b',
                 title='Power espectral density')
    p1.setLabel('left', "Relative power (db)")
    p1.setLabel('bottom', "Frequency (MHz)")
```

Con el Código 4.8 se elige una frecuencia central y se obtiene una representación a tiempo real del espectro. Es un analizador de frecuencias fft simplificado, sin posibilidad de cambiar de frecuencia durante su ejecución u otras opciones avanzadas que se han visto en el Capítulo 2. Se ha encontrado una librería [89] que crea un analizador de espectros en Python. Por desgracia, no se puede instalar en el ordenador utilizado en el trabajo, por lo que no se ha podido probar su funcionamiento.

Código 4.8 cont_spectrum().

```
def cont_spectrum(center_freq, sample_rate, gain=28,  samples_number=128*1024,
                  freq_correction=93, device_index=0):
    """ Representa de forma continua la densidad espectral de potencia en la
    frecuencia elegida.


    Parámetros de entrada
    ----------
    center_freq: float
        Frecuencia central de recepción en Hz.
    sample_rate: float
        Frecuencia de muestreo en Hz.
    gain: float or str, opcional
        Ganancia del dispositivo en dB.
    samples_number: int, opcional
        Número de muestras a leer.
    freq_correction: int, opcional
        Desviación de frecuencia en PPM.
    device_index: int, opcional
        Número de puerto del dispositivo.
    """
    pg.setConfigOption('background', 'w')
    pg.setConfigOption('foreground', 'k')
    pg.setConfigOptions(antialias=True)

    sdr = RtlSdr(device_index)
    rtl_config(sdr, center_freq, sample_rate, gain, freq_correction)

    global curve, ptr, p1
    win = pg.GraphicsWindow()
    p1 = win.addPlot(title='Power espectral density')
    p1.setLabel('left', "Relative power (db)")
    p1.setLabel('bottom', "Frequency (MHz)")
    curve = p1.plot(pen='b')
    ptr = 0

    def update():
        global curve, ptr, p1
        samples = sdr.read_samples(samples_number)
        power, psd_freq = psd(samples, NFFT=1024, Fs=sample_rate/1e6)
        psd_freq += center_freq/1e6
        curve.setData(psd_freq, 10*np.log10(power))
        if ptr == 3:
            p1.enableAutoRange('xy', False)
        ptr += 1

    timer = QtCore.QTimer()
    timer.timeout.connect(update)
    timer.start(10)
    QtGui.QApplication.instance().exec_()
```

La función del Código 4.9 sirve para realizar un barrido entre las frecuencias especificadas y obtener dos matrices con los valores de la densidad espectral de potencia y las frecuencias centrales a las que corresponden. Además, se puede decidir si representar esos valores al terminar el barrido.

Código 4.9 freq_sweep().

```
def freq_sweep(freq_start, freq_end, do_plot=True, sample_rate=2.4e6, gain=28,
               samples_number=258*1024, freq_correction=93, device_index=0):
    """ Realiza un barrido en frecuencia entre las frecuencias especificadas.


    Parámetros de entrada
    ----------
    freq_start: float
        Frecuencia inicial del barrido en Hz.
    freq_end: float
        Frecuencia final de recepción en Hz.
    do_plot: bool
        Selecciona si representar la densidad espectral de potencia.
    sample_rate: float, opcional
        Frecuencia de muestreo en Hz.
    gain: float or str, opcional
        Ganancia del dispositivo en dB.
    samples_number: int, opcional
        Número de muestras a leer.
    freq_correction: int, opcional
        Desviación de frecuencia en PPM.
    device_index: int, opcional
        Número de puerto del dispositivo.

    Salidas
    ----------
    power: ndarray
        Matriz con la densidad espectral de potencia.
    freq: ndarray
        Matriz con las frecuencias barridas en MHz.
    """
    pg.setConfigOption('background', 'w')
    pg.setConfigOption('foreground', 'k')
    pg.setConfigOptions(antialias=True)

    power = np.ndarray(0)
    freq = np.ndarray(0)
    sdr = RtlSdr(device_index)
    rtl_config(sdr, freq_start, sample_rate, gain, freq_correction)

    for i in np.arange(freq_start, freq_end, sample_rate):
        sdr.center_freq = i
        samples = sdr.read_samples(samples_number) #128*1024
        power_aux, freq_aux = psd(samples, NFFT=1024, Fs=sample_rate/1e6)
        freq_aux += i/1e6
        power = np.concatenate((power, np.array(power_aux)))
        freq = np.concatenate((freq, np.array(freq_aux)))

    sdr.close()
    if do_plot is True:
        p1 = pg.plot(freq, 10*np.log10(power),pen='b',
                     title='Power espectral density')
        p1.setLabel('left', "Relative power (db)")
        p1.setLabel('bottom', "Frequency (MHz)")
    return power, freq
```

Por último, se ha querido realizar una función que automatice la captura y guardado de las muestras IQ. El Código 4.10 captura los datos IQ recibidos en la frecuencia especificada durante el tiempo que se indique y, además de devolver una matriz con esos datos, permite realizar un guardado automático en tres formatos de archivo diferentes.

Código 4.10 save_IQ().

```
def save_IQ(center_freq, sample_rate, time_stop, file_name, npy_file=True,
            wav_file=False, mat_file=False, gain=28, samples_number=128*1024,
            freq_correction=93, device_index=0):
    """ Obtiene datos IQ de la frecuencia central durante el tiempo estipulado
    y los guarda en el archivo elegido en formato .npy, .wav y/o .mat .


    Parámetros de entrada
    ----------
    center_freq: float
        Frecuencia central en Hz.
    sample_rate: float
        Frecuencia de muestreo en Hz.
    time_stop: int or float
        Tiempo de recepción en segundos.
    file_name: str
        Nombre del archivo donde guardar los datos.
    npy_file: bool, opcional
        Selecciona si guardar el archivo en formato .npy.
    wav_file: bool, opcional
        Selecciona si guardar el archivo en formato .wav.
    mat_file: bool, opcional
        Selecciona si guardar el archivo en formato .mat.
    gain: float or str, opcional
        Ganancia del dispositivo en dB.
    samples_number: int, opcional
        Número de muestras a leer.
    freq_correction: int, opcional
        Desviación de frecuencia en PPM.
    device_index: int, opcional
        Número de puerto del dispositivo.

    Salidas
    ----------
    IQ_data: ndarray
        Matriz con los datos IQ captados
    """
    sdr = RtlSdr(device_index)
    rtl_config(sdr, center_freq, sample_rate, gain, freq_correction)

    Fs = sample_rate
    IQ_data = np.ndarray(0)
    start = time.time()

    while True:
        samples = sdr.read_samples(samples_number) #128*1024
        IQ_data = np.concatenate((IQ_data, np.array(samples)))
        aux = time.time()-start
```

```
        if aux > time_stop:
            if npy_file is True:
                info=np.array([IQ_data, Fs])
                np.save(file_name,info)
            if wav_file is True:
                N=IQ_data.size
                wav_samples = np.zeros((N, 2), dtype=np.float32)
                wav_samples[...,0] = IQ_data.real
                wav_samples[...,1] = IQ_data.imag
                aux = file_name + '.wav'
                wavfile.write(aux, int(sdr.sample_rate), wav_samples)
            if mat_file is True:
                signal={'Fs':sample_rate, 'IQ':IQ_data}
                aux = file_name + '.mat'
                scipy.io.savemat(aux, signal)
            return IQ_data
    sdr.close()
```

4.4 Ventajas y desventajas

Una vez visto el funcionamiento de Python y alguna de sus librerías es importante mencionar qué ventajas ofrece trabajar con Python [68] y [90]:

- Es un software de código abierto, permite distribuir libremente los programas realizados sin requerir permisos.
- Completamente gratuito, al igual que todas las librerías.
- Sintaxis simple y sencilla de leer que facilita la escritura de programas y la localización de errores.
- Funciona en Window, Linux, Unix y MAC. El código que se escriba se puede ejecutar en cualquiera de estos sistemas operativos.
- Los intérpretes como Spyder permiten escribir código e ir experimentando con él al mismo tiempo.
- Tiene una gran comunidad de usuarios que se resuelven problemas mutuamente.
- Existen una gran cantidad de paquetes y librerías de todo tipo que expanden las funciones de Python.
- Python es flexible, se puede programar de forma procedimental, orientada a objetos o a funciones.
- Es bastante sencillo aprender a programar en Python, la sintaxis y términos utilizados son fáciles de comprender con el suficiente conocimiento de inglés.
- Los mensajes de errores suelen ser claros y ayudan a encontrar el problema.

Teniendo en cuenta todas estas ventajas y las librerías que se han visto, es natural que Python se utilice en muchos proyectos relacionados con los RTL-SDR. A destacar: un analizador de espectro [15], utilización en educación [14] y detección y decodificación de señales ASK [19].

Hay que destacar un problema que se ha encontrado al utilizar Python. Se debe al hecho de que el ordenador utilizado sea un Windows 8.1 no muy potente. El problema consiste en que Windows asocia a cada programa una cantidad de memoria que puede usar y no es posible aumentarla. Entonces, al utilizar Python no es posible representar algunas de las señales que se han utilizado, y que sí se han representado en MATLAB. Por ejemplo, no se pueden hacer las gráficas de las capturas originales, solo se pueden representar una vez realizado el diezmado. Por tanto, el problema es importante, ya que sucede con tamaños de unos cientos de Mb. Esto no significa que Python deje de ser útil, por ahora solo ha dado problemas al representar algunos datos, nunca al operar con ellos ni con el código detector de ACARS. Una probable solución del problema es utilizar un ordenador con otro sistema operativo.

5 Detector ACARS

En este capítulo se va a explicar el objetivo fundamental del trabajo, el programa utilizado para la detección de los mensajes ACARS. Primero se va a comentar la estructura de procesamiento de señal del programa. Después, se analizarán en detalle los diferentes filtros empleados. Se detallará el código en MATLAB y Python, mostrando su funcionamiento con capturas de señales en canales ACARS. Por último, se analizaran las diferencias de la implementación en MATLAB y Python.

5.1 Estructura de procesamiento

Los mensajes ACARS tienen una estructura muy concreta y han sido modulados en MSK, tal y como se ha explicado en el Capítulo 3. Esto permite que se pueda distinguir cuándo la señal captada en un canal asignado al uso de ACARS es una comunicación de este sistema o si solo se trata de interferencias, ruido u otro tipo de mensaje. En concreto, se va a hacer uso de que la señal capturada por el RTL-SDR, al haber sido reducida a la banda base, tenga dos frecuencias fundamentales: 1200 y 2400 Hz.

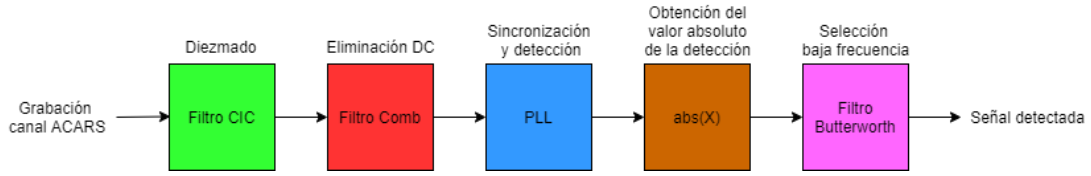


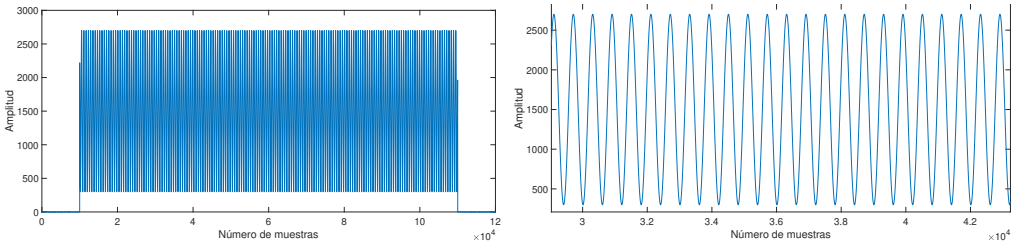
Figura 5.1 Estructura de la detección de ACARS.

En la Figura 5.1 se muestran las acciones que se realizan sobre la señal. Con un filtro *Cascaded integrator-comb filter* (CIC) se diezma la captura original. Esto sirve para reducir la tasa de muestreo y el volumen de datos a procesar. Después se eliminan las componentes de DC de la señal, para lo que se emplea, por simplicidad, un filtro Comb con nulo en DC. Así, se utiliza un *phase-locked loop* (PLL) que se sincroniza la portadora de 2400 del mensaje ACARS. Tras detectar la señal se obtiene su valor absoluto, lo que mueve la portadora a 0 y 4800 Hz. Se utiliza un filtro de Butterworth de paso bajo para eliminar el término de alta frecuencia. Tras pasar por todas estas fases se obtiene una señal limpia que permite saber en qué momento se ha transmitido un mensaje a través del sistema ACARS, además de cuánto tiempo ha durado esa transmisión.

En las siguientes secciones se va a analizar cada uno de los bloques de la Figura 5.1. Explicando los filtros utilizados y sus efectos sobre una señal. Para ver bien qué hace cada bloque se va a utilizar como entrada la señal ideal de la Figura 5.2. Está formada por dos señales muy pequeñas al principio y final de las muestras y un coseno de gran amplitud con frecuencia 2400 Hz. De esta forma se imita una señal ACARS ideal. Es importante destacar que la mayoría de los filtros utilizados se crean en función de la frecuencia de muestreo de la señal introducida en el detector, por lo que las respuestas frecuenciales y funciones de transferencia variarán según la señal a procesar.

5.1.1 Filtro CIC

Lo primero que se quiere hacer con la señal captada es diezmarla, ya que a las frecuencias de muestreo requeridas se obtiene una gran cantidad de muestras en poco tiempo de captura. Se va a diezmar con un



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.2 Señal ideal.

factor de 30, es decir, por cada 30 muestras solo se queda una. Por tratarse de un diezmado tan grande es muy posible que se produzcan problemas de aliasing. Para evitar esto, es habitual utilizar un filtro de paso bajo que reduzca las componentes de alta frecuencia.

En este caso se va a implementar un *Cascaded integrator-comb filter* (CIC). Un CIC es una clase de filtro *finite impulse response* (FIR) específicamente creado para diezmado e integración por Hogenauer [91]. Los CIC son filtros computacionalmente eficientes, ya que requieren muy poco tiempo de procesamiento. Son menos complejos de implementar que un solo FIR de paso bajo. Además, son muy utilizados como filtro anti-aliasing antes de ejecutar un diezmado.

$$H_{\mathbf{Z}}(\mathbf{z}) = \frac{1 - z^{-30}}{1 - z^{-1}} \tag{5.1}$$

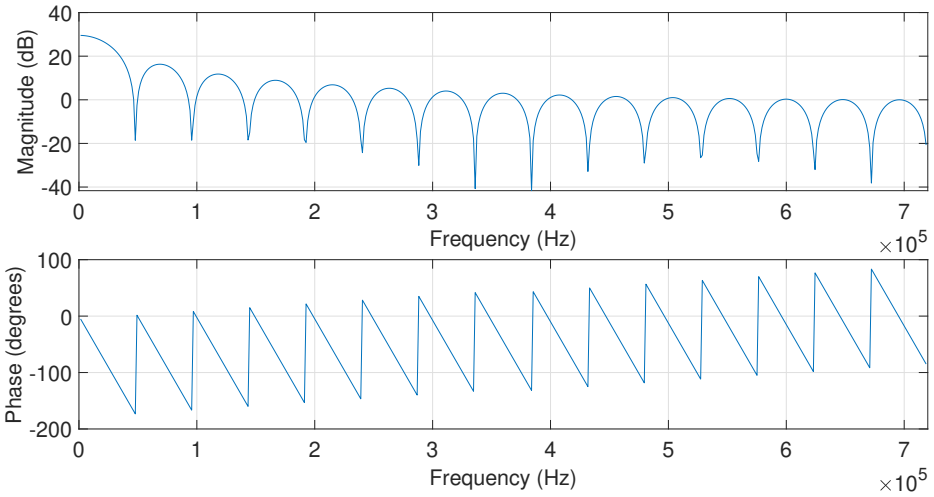


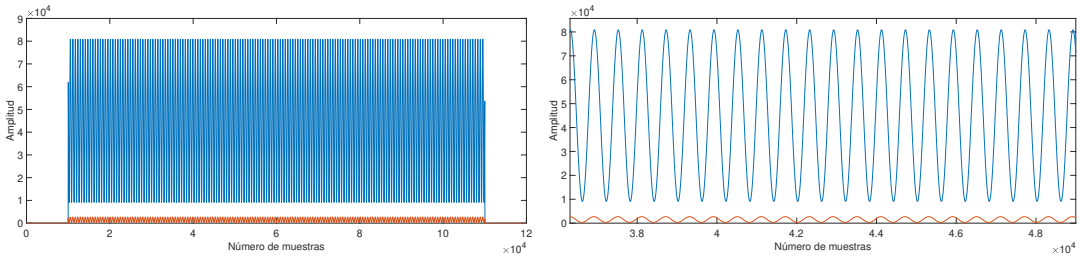
Figura 5.3 Respuesta frecuencial del filtro CIC.

La función de transferencia del CIC concreto que se va a usar es la de la ecuación (5.1). La respuesta frecuencial de ese filtro se observa en la Figura 5.3. Mezcla la respuesta de un filtro integrador con un filtro Comb, aumentando la magnitud de las frecuencias más pequeñas. Su efecto sobre la señal de la Figura 5.2 se puede ver en la Figura 5.4. La señal ha sido amplificada sin amplificar ruido.

Por último, se procede a diezmar la señal. En el resultado, Figura 5.5, se ve claramente que se han reducido el número de muestras. Esto hará más rápidos, en tiempo de procesamiento, todo el resto de pasos a aplicar. Gracias a haber aplicado el filtro CIC antes del diezmado la señal no ha sufrido aliasing.

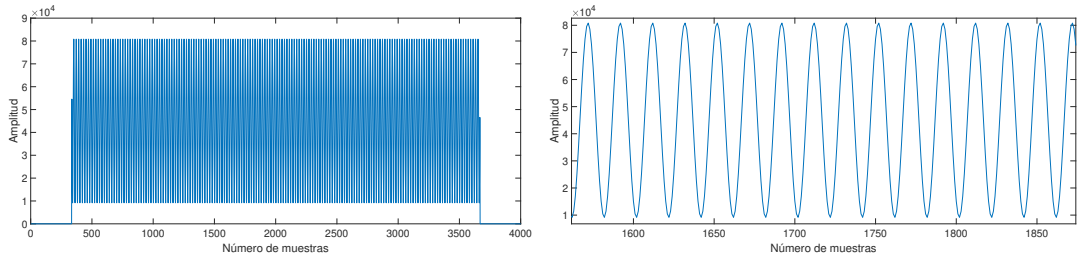
5.1.2 Filtro Comb

El siguiente paso es aplicar un filtro Comb. Este tipo de filtro se implementa mediante la suma a la señal de la misma señal retrasada o de la señal filtrada retrasada. Esto provoca interferencias destructivas y constructivas que la modifican. Se llaman filtros Comb, peine en ingles, debido a la forma de su respuesta en frecuencia. Tienen varias aplicaciones, como procesado de audio o ser parte de un filtro CIC. Son muy utilizados ya que,



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.4 Señal ideal antes (naranja) y después (azul) del filtrado CIC.



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.5 Señal ideal tras el diezmado.

debido a su simplicidad, casi no requieren tiempo de procesamiento. Existen dos versiones de filtros Comb: feedforward y feedback.

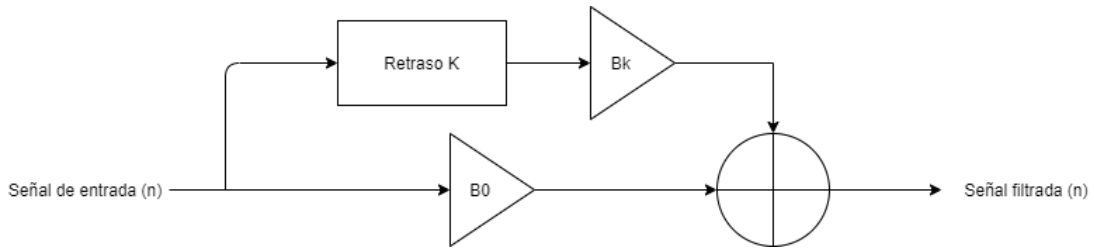


Figura 5.6 Estructura filtro Comb feedforward.

En este trabajo se utiliza un filtro Comb feedforward [92] para quitar la componente DC de la señal que se está procesando. Esto es necesario para que el PLL funcione correctamente. Su estructura, que se muestra en la Figura 5.6, indica cómo se suma la señal retrasada para realizar el filtrado. La función de transferencia de este filtro es la ecuación (5.2), pero sólo es un ejemplo, ya que depende de la frecuencia de muestreo de la señal introducida en el detector. En este caso la señal se retrasa en 13 muestras y se multiplica por -1. La señal de entrada está multiplicada por un factor unidad, por lo que no sufre cambios. Finalmente, debido a ese negativo, se resta la muestra retrasada a la señal para realizar el filtrado.

$$H_{\mathbf{Z}}(\mathbf{z}) = 1 - z^{-13} \tag{5.2}$$

Su respuesta frecuencial se muestra en la Figura 5.7. El filtro se ha creado de forma que elimine completamente la componente DC de la señal. Eso se ve en la figura, donde a frecuencias cercanas a cero la magnitud es muy pequeña. Por lo demás, es destacable que el filtro permite el paso de las dos frecuencias con las que se quiere trabajar (1200 y 2400 Hz). En esas dos frecuencias el filtro tiene una magnitud cercana a la máxima.

Solo queda analizar la señal una vez filtrada, Figura 5.8. Puede asegurarse que el filtro ha cumplido su función, ya que se ve la misma señal que antes, en forma y frecuencia, pero centrada en cero. Así, se ha eliminado la componente DC de forma satisfactoria y se puede proseguir procesando la señal.

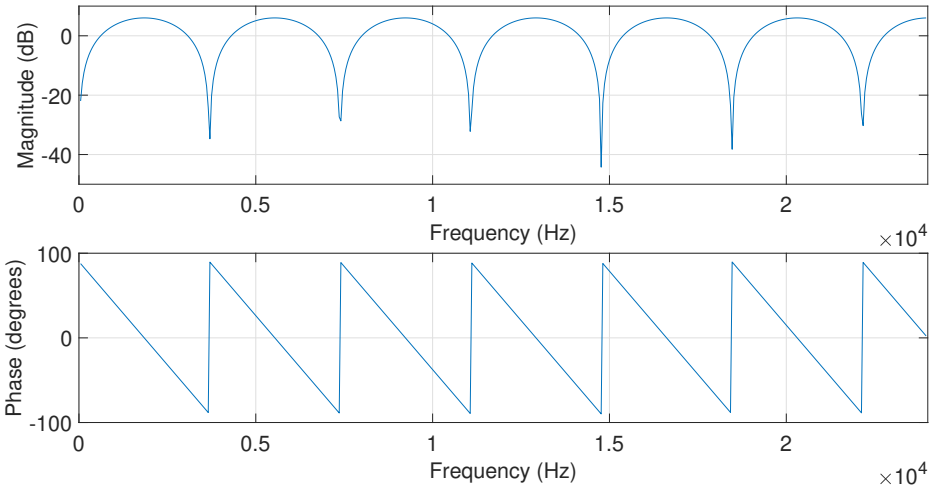
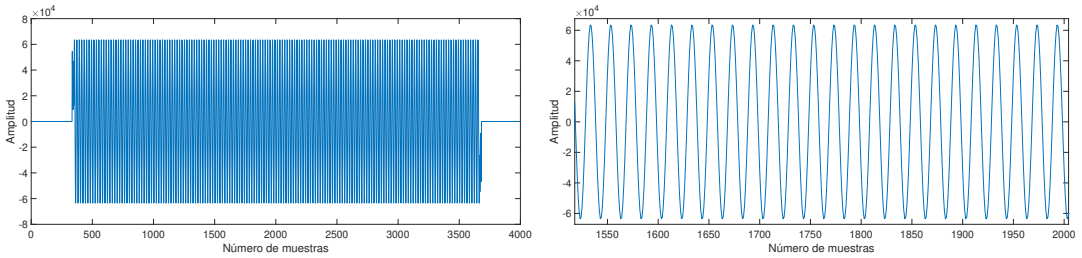


Figura 5.7 Respuesta frecuencial del filtro Comb.



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.8 Señal ideal tras el filtro Comb.

5.1.3 Phase-Locked Loop

Un PLL es un sistema de control que es capaz de sincronizarse con la señal de entrada. Son capaces de recrear la entrada, seguir las desviaciones en frecuencia y eliminar ruido. Se va a utilizar para crear una señal sinusoidal que se sincronice con la señal ACARS, a 2400 Hz de frecuencia. De esta forma, se podrá saber si hay un mensaje ACARS y cuándo. Un PLL está compuesto por tres componentes [6] organizados tal y como aparecen en la Figura 5.9.

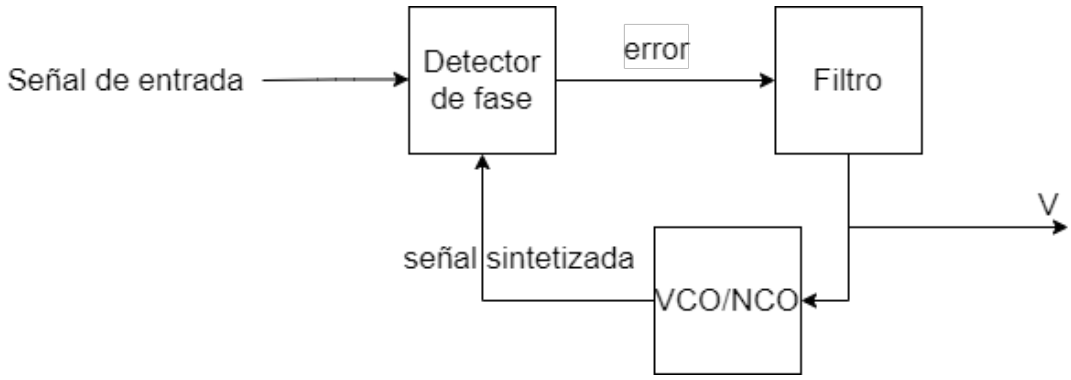


Figura 5.9 Estructura PLL.

- *Detector de fase*: Genera una señal proporcional a la diferencia de fase entre la señal de entrada y la señal retroalimentada del oscilador. Esa señal se conoce como el error de fase, y es igual a $K_p(\theta_i(n) - \theta_0(n))$. Los dos θ son las fases de la señal de entrada y la retroalimentada, respectivamente. Mientras que K_p

es la ganancia del detector de fase. Esta señal se puede generar de varias maneras. Aquí se va a utilizar el método de multiplicar las dos señales que entran en el detector.

- *Filtro*: Multiplicar las dos señales crea un término a bajas frecuencias que varía con la diferencia de fase y otro a altas frecuencias (al doble de la frecuencia de entrada). Por eso el siguiente paso es filtrar ese último término. En este caso se implementa un filtro paso bajo tipo Comb, cuya respuesta en frecuencia se enseña en la Figura 5.10 y su función de transferencia es la ecuación (5.3). Esta ecuación se trata de un ejemplo, ya que depende de la frecuencia de muestreo. Se ha elegido este filtro por ser uno de los más rápidos de implementar. Sirve para eliminar la componente de alta frecuencia. La señal que sale de este filtro es la que se utiliza para controlar el oscilador.

$$H_{\mathbf{Z}}(\mathbf{z}) = 1 + z^{-5} \tag{5.3}$$

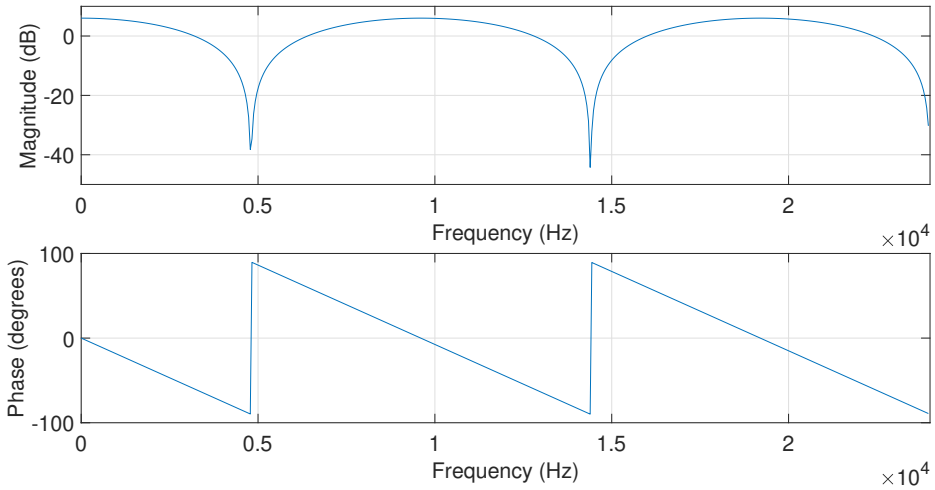
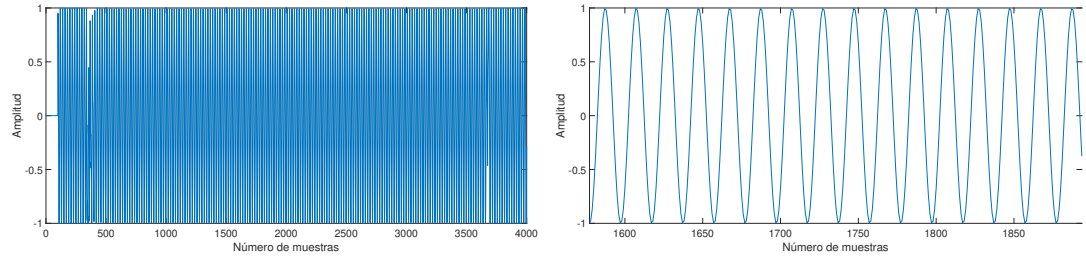


Figura 5.10 Respuesta frecuencial del filtro del PLL.

- *VCO/NCO*: Es un generador de señales sinusoidales cuya frecuencia está controlada por la señal obtenida del filtro. En sistemas analógicos se denomina oscilador controlado por voltaje (VCO). El que se va a utilizar en este trabajo, al estar realizado digitalmente, es un oscilador controlado numéricamente (NCO). Está ajustado a la frecuencia de 2400 Hz, de forma que se pueda sincronizar con un mensaje ACARS. Esto es posible ya que a esa frecuencia el PLL se sincroniza tanto con las partes del mensaje a 2400 Hz como con las partes a 1200 Hz. El oscilador que se va a implementar crea una señal coseno con una diferencia de fase de 90º respecto a la señal de entrada al PLL. Esta señal, Figura 5.11, es la que se utilizará para la detección del mensaje ACARS. Los cortes o discontinuidades que se ven en la figura son causados por los momentos en los que el PLL se sincroniza con la señal y en los que esa señal deja de enviarse.



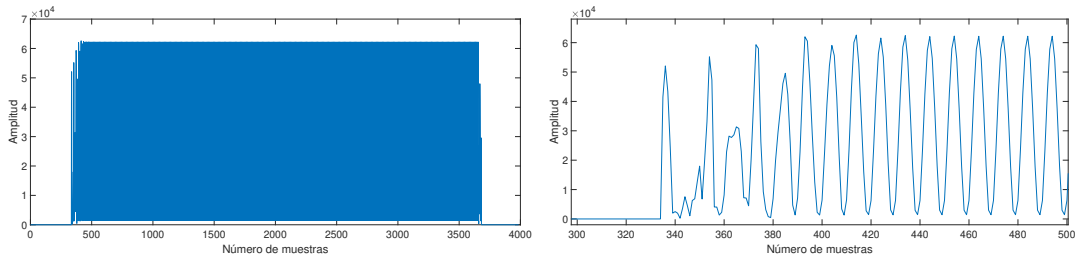
(a) Señal completa.

(b) Ampliación mostrando varios periodos.

Figura 5.11 Señal sintetizada en el oscilador.

5.1.4 Detección

Una vez realizados todos esos pasos es sencillo detectar los mensajes ACARS. Se multiplican la señal de entrada al PLL (la de salida del filtro Comb), Figura 5.8, y la creada en el oscilador del PLL, Figura 5.11. Esta última se multiplica adelantándola de forma que se corrija el desfase de 90º con el que se crea. Por último, como no se quiere demodular la señal y solo importa detectarla, para simplificarla se obtiene el valor absoluto de esa multiplicación.



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.12 Señal ideal tras la detección.

El resultado que se obtiene es el de la Figura 5.12. Se ve claramente que el uso del valor absoluto ha eliminado los componentes negativos de amplitud y doblado la frecuencia, que ha pasado a ser de 4800 Hz. Esto último ya no tiene importancia, al no buscar la información que trae el mensaje sino solo la existencia del mismo. La señal obtenida tiene valor no nulo solo cuando existe transmisión de un mensaje ACARS. En la ampliación se puede observar cómo al empezar a recibir el mensaje este no se detecta directamente, sino que tarda unas pocas oscilaciones. Esto es debido al proceso de sincronización del PLL.

5.1.5 Filtro Butterworth

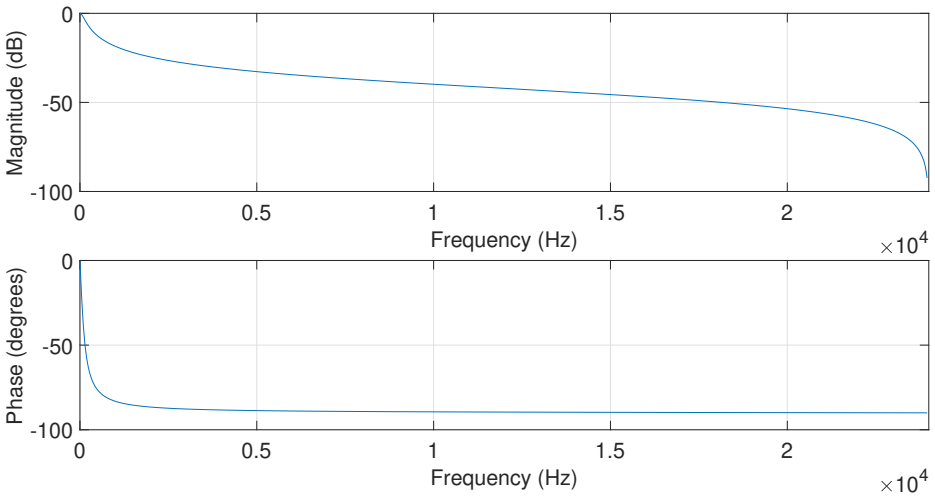
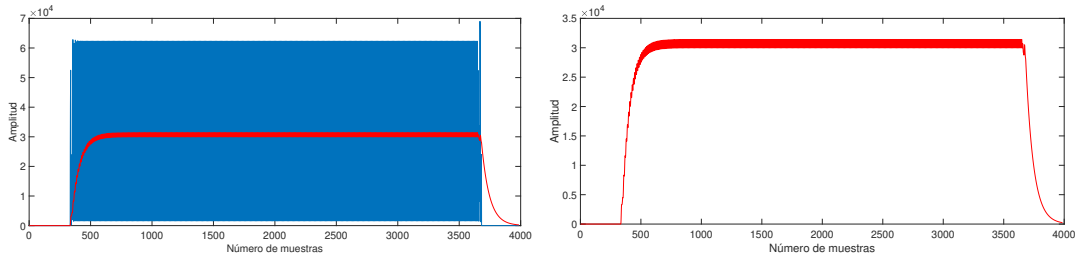


Figura 5.13 Respuesta frecuencial del filtro Butterworth.

Por último, para ver mejor la existencia del mensaje y facilitar la detección, se va a aplicar un filtro paso bajo. Se ha elegido un filtro Butterworth. Este tipo de filtro se diseña para tener una respuesta lo más plana posible en la banda de frecuencias cuyo paso se quiere permitir. Y después, una vez pasada la frecuencia de corte, su magnitud se reduce drásticamente. El filtro utilizado tiene la respuesta en frecuencia que se observa en la Figura 5.13. La intención al aplicar este filtro es eliminar todas componentes de alta frecuencia y dejar únicamente las componentes de frecuencias más cercanas a cero, es decir, la componente DC. Su efecto sobre la señal detectada se puede analizar en la Figura 5.14. Tras este filtrado es sencillo saber cuándo se ha producido una transmisión ACARS.



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.14 Señal ideal tras filtrado Butterworth (rojo) y la detectada (azul).

5.2 Modelo MATLAB

Una vez analizado el procesamiento que se la va a hacer a las capturas de señales en canales ACARS se puede proceder a explicar el código con el que se implementa. Se ha creado una función de MATLAB que recibe como entradas el valor absoluto de los datos I/Q capturados por el RTL-SDR y la frecuencia de muestreo que se utilizó en la captura.

Código 5.1 Acars_detector en MATLAB.

```
function acars_detector_final(Input,Fs)
% ACARS receiver (detecting signal)
% Input: abs(datos IQ)
% ---------------------------------------------------------------

F1 = 2400;
F2 = 1200;

% Filtro CIC
CIC.R            = 30;
CIC.Num(CIC.R+1) = -1;
CIC.Num(1)       = 1;
CIC.Den          = [1 -1];
% ---------------------------------

% Filtro Comb
Comb.K             = round(Fs/(CIC.R*(F1+F2)));
Comb.Num(Comb.K+1) = -1;
Comb.Num(1)        = 1;
Comb.Den           = 1;
% ---------------------------------

% PLL
PLL.Acc_max = 2^14;
PLL.Fs      = Fs/CIC.R;
PLL.Gain    = 10;
PLL.S       = 1e-2;

PLL.PPP                    = PLL.Fs/F1; % puntos por periodo en PLL
PLL.PPP_pi_2               = round(PLL.PPP/4); % desfase pi/2 en muestras
PLL.Comb.Num(1)            = 1;
PLL.Comb.K                 = round(Fs/CIC.R/(4*F1));
PLL.Comb.Num(PLL.Comb.K+1) = 1;
PLL.LUT                    = cos([0:PLL.Acc_max-1].*2*pi/PLL.Acc_max);
%-------------------------------
```

```
36
37 % Filtro Butterworth LPF
38 %------------------------------
39 [B,A] = butter(1,0.1*F1/PLL.Fs);
40 DET.Num = B;
41 DET.Den = A;
42 DET.THR = 100;
43 %------------------------------
44
45 % Reserva de memoria y valores iniciales
46 nTAM    = length(Input);
47 mTAM      = ceil(nTAM/CIC.R);
48 x_mixed   = zeros(mTAM,1);
49 x_lpf     = zeros(mTAM,1);
50 DVCO      = zeros(mTAM,1);
51 Acc       = 0;
52 DDSo      = 0;
53 INITSIM   = 100;
54
55 % ++++++++++++++++++++++++++
56 % +++++ Procesamiento ++++++
57 % ++++++++++++++++++++++++++
58 % Filtrado CIC
59 x_cic = filter(CIC.Num,CIC.Den,Input);
60
61 % Diezmado
62 x_cic_int = x_cic(1:CIC.R:end);
63
64 % Filtrado Comb
65 x_comb_dc = filter(Comb.Num,Comb.Den,x_cic_int);
66
67 % PLL
68 for m = INITSIM:length(x_comb_dc)
69
70     % Deteccion de fase
71     x_mixed(m) = x_comb_dc(m)*DDSo*PLL.Gain;
72
73     % LPF
74     x_lpf(m)    = x_mixed(m)+PLL.Comb.Num(end)*x_mixed(m-PLL.Comb.K);
75
76     % NCO
77     Acc         = mod(round(Acc+(F1+PLL.S*x_lpf(m))*PLL.Acc_max/PLL.Fs),PLL.
       Acc_max);
78     DDSo        = PLL.LUT(Acc+1);
79     DVCO(m)     = DDSo;
80 end
81
82 % *** Deteccion ***
83 % Mezclado
84 x_1   = abs(DVCO(3*PLL.PPP_pi_2-1:end-1).*x_comb_dc(1:end+1-3*PLL.PPP_pi_2));  %
       Portadora sincronizada
85 % Butterworth LPF
86 x_1f = filter(DET.Num, DET.Den,x_1);
87 % ------------------------
88
89 % Tiempo de ocupacion
90 c        = length(find(x_1f>DET.THR));
```

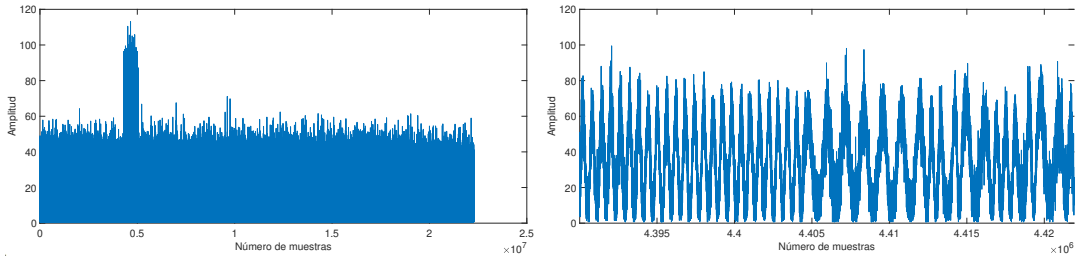
```
c_time  = c/PLL.Fs
c_ratio = c/mTAM

return
```

Es el Código 5.1. Lo primero que se hace es declarar las dos frecuencias del sistema ACARS. Luego se configuran los filtros y el PLL que se han visto en esta capítulo. Para la creación del PLL en un principio la línea 78 del código era $DDSo = cos(2 * pi * Acc/PLL.Acc\_max)$; pero, con la intención de hacer el procesamiento más rápido, se decidió utilizar una *lookup table* (LUT) [93]. Es una estructura de datos que, en este caso, guarda la información del coseno. De esta forma se elimina el proceso de computación de calcular el coseno cada vez, sustituyéndolo por una búsqueda de la componente de un vector en la memoria.

Después de declarar las distintas variables para realizar la reserva de memoria se procede con el procesamiento de la señal. Los filtros se aplican mediante el comando *filter*, que filtra la señal introducida mediante el filtro definido por el numerador y denominar dados. El diezmado se realiza "manualmente" seleccionando los datos. Para implementar el PLL es necesario un bucle, ya que, al ser un sistema complejo formado por varios componentes realimentados, no puede realizarse directamente sobre la señal como en el caso de los filtros. Del bucle se obtiene la señal DVCO.

La detección se realiza, tal y como se ha explicado, multiplicando la DVCO y la señal de salida del filtro Comb. A esa detección se le aplica el filtro Butterworth con el mismo comando que los filtros anteriores. Finalmente, se obtiene el tiempo de ocupación del canal, es decir, el tiempo durante el que se están trasmitiendo mensajes ACARS. Esto se consigue calculando el número de muestras de la señal detectada que presenta valores altos en amplitud. Se ha elegido un nivel de decisión de 250. La variable *c_time* indica cuantos segundos ha durado la ocupación. *c_ratio* es el porcentaje de tiempo que el canal ha sido utilizado respecto al tiempo de la captura.



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.15 Captura ACARS con un mensaje.

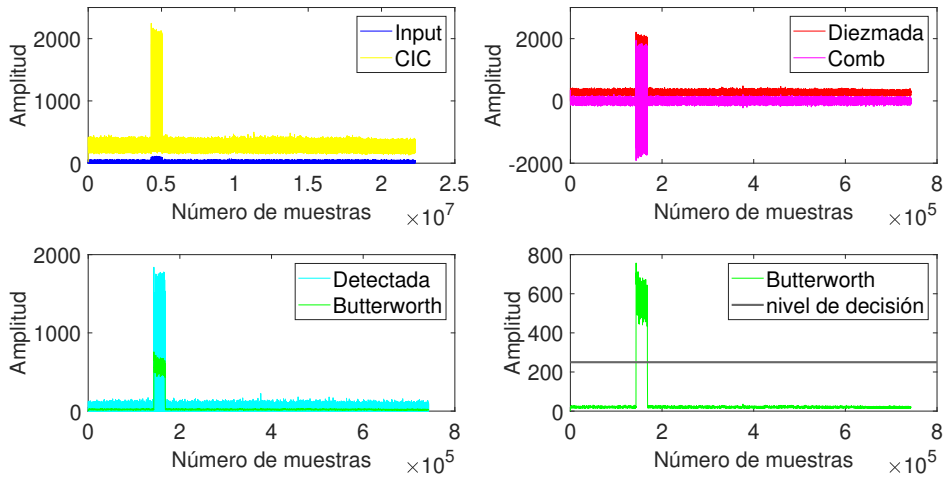


Figura 5.16 Ejemplo detección ACARS.

Se va a probar el código con una captura de un canal en la que aparece una transmisión ACARS, Figura 5.15. La captura no dura muchos segundos, pero se puede ver el ruido que tiene el canal y el mensaje. Se aprecia que el mensaje está modulado en MSK por las dos frecuencias que se distinguen al ampliarlo. Se pasa esta captura por el Código 5.1 y se obtiene que el mensaje dura 0.5160 segundos, un 3.33 % del tiempo que dura la captura.

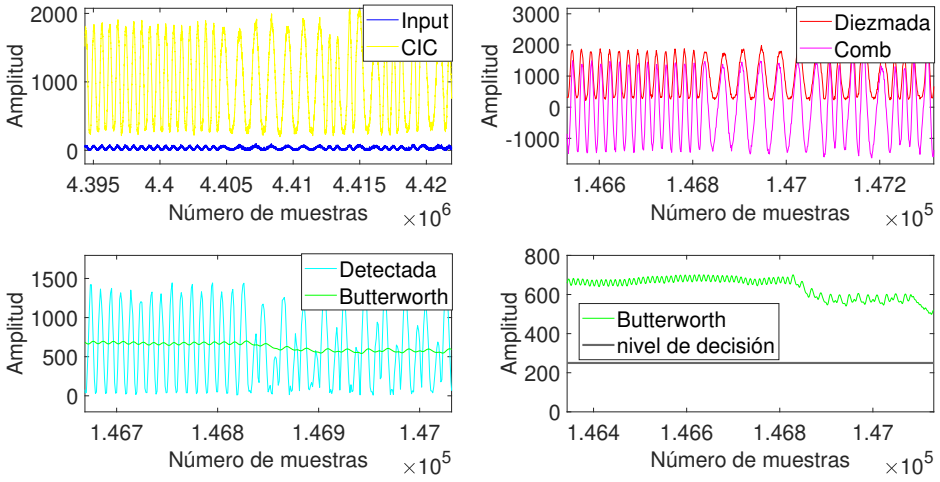


Figura 5.17 Ejemplo detección ACARS ampliado.

En la Figura 5.16 y la Figura 5.17 se muestran distintas fases del proceso. Se puede ver cómo el filtrado CIC aumenta la amplitud de la señal. El diezmado reduce el número de muestras sobre las que se trabaja y el filtro Comb elimina la componente en continua. Después se realiza la detección y el ultimo filtrado, tras lo que se puede calcular el tiempo de uso del canal. Con esto se ha comprobado que el detector funciona con una captura real de un canal RF.

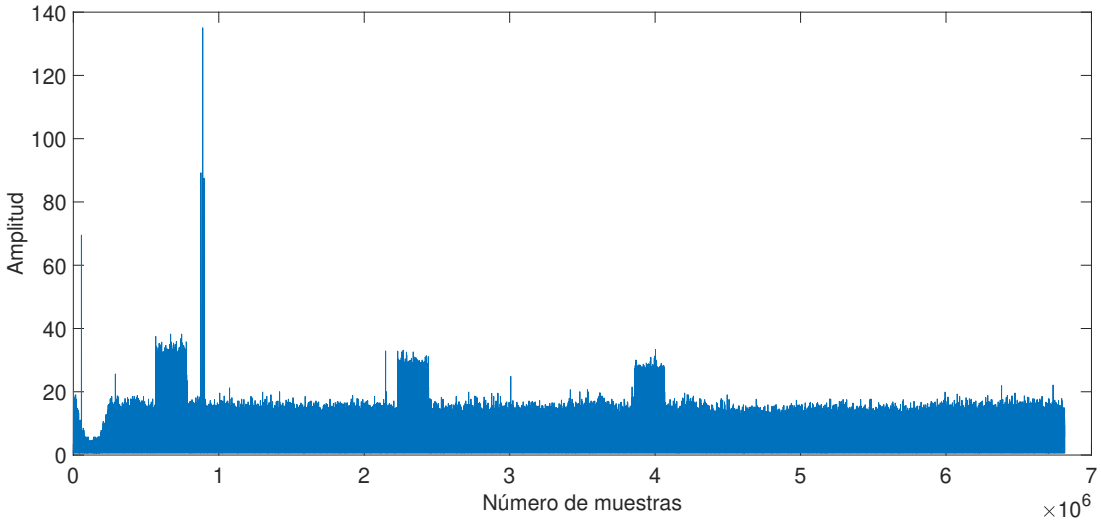


Figura 5.18 Captura ACARS con varios mensajes.

También se prueba el código con una captura en la que se han captado varios mensajes ACARS, Figura 5.18. El procesamiento se muestra en la Figura 5.19. Dado que en esta captura la amplitud de la señal es menor, es necesario reducir el valor del nivel de decisión para que detecte la señal correctamente. En este caso el canal está ocupado durante 2.6111 segundos, un 9.19 % de la duración del mensaje.

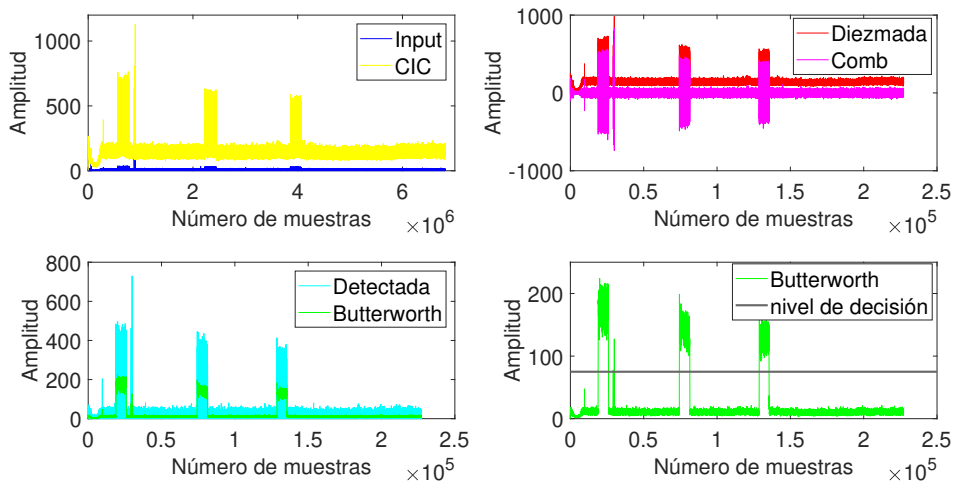


Figura 5.19 Ejemplo detección ACARS con varios mensajes.

5.3 Realización en Python

Tras aprender a utilizar Python y algunas de sus librerías, se procedió a convertir el Detector ACARS de MATLAB a código ejecutable en Python. La estructura de procesamiento de la señal es la misma, y la del código también. Al ser un lenguaje diferente, varían especialmente la forma de definir las variables y los comandos utilizados.

El Código 5.2 es la implementación en Python. Para la realización del código son necesarias dos librerías: *Numpy* para el uso de matrices y la mayoría de comandos utilizados; *Scipy* para el tratamiento de la señal y filtros. Se ha dividido la estructura en dos partes. Primero se define el detector y luego se muestra un ejemplo de cómo obtener las variables y ejecutarlo.

Lo primero es importar las librerías que se van a utilizar. Después, se configuran los filtros y el PLL. Tras hacer la reserva de memoria se comienza el procesamiento de señal, que tiene la misma estructura que en MATLAB, con solo el PLL dentro del bucle. El cálculo del tiempo de ocupación es similar al realizado en MATLAB. Por último, se muestran en pantalla los valores de tiempo y se declaran como variables de salida.

Código 5.2 Acars_detector en Python.

```
def acars_detector(Input, Fs):
    """ ACARS Detector


    Detecta la presencia de mensajes ACARS, calcula su duración y el tiempo de
    ocupación del canal.

    Parámetros de entrada
    ----------
    Input : ndarray
        Valor absoluto de los datos IQ de la captura.
    Fs : int
        Frecuencia de muestreo de la captura.

    Salidas
    ----------
    c_time : float
        Tiempo de transmisión de los mensajes ACARS.
    c_ratio : float
        Porcentaje de tiempo (sobre 1) que el canal está siendo utilizado.
```

```
    """
    import numpy as np
    import scipy.signal as sig

    F1 = 2400
    F2 = 1200

    # Filtro CIC
    CIC = {'R':30,
           'Num':np.zeros(31,dtype=int),
           'Den':np.array([1, -1])}
    CIC['Num'][-1] = -1
    CIC['Num'][0] = 1

    # Filtro Comb
    Comb = {'K':int(np.round(Fs/(CIC['R']*(F1+F2)))),'Den':1}
    Comb['Num'] = np.zeros(Comb['K']+1,dtype=int)
    Comb['Num'][-1] = -1
    Comb['Num'][0] = 1

    # PLL
    PLL = {'Acc_max':2**14, 'Fs':Fs/CIC['R'], 'Gain':10, 'S':1e-2,
           'Comb_K':int(np.round(Fs/CIC['R']/(4*F1)))}
    PLL['PPP'] = PLL['Fs']/F1
    PLL['PPP_pi_2'] = int(np.round(PLL['PPP']/4))
    PLL['Comb_Num'] = np.zeros(PLL['Comb_K']+1,dtype=int)
    PLL['Comb_Num'][-1] = 1
    PLL['Comb_Num'][0] = 1
    PLL['LUT'] = np.cos(np.arange(PLL['Acc_max']) * 2 * np.pi/PLL['Acc_max'])

    # Filtro LPF Butterworth
    B, A = sig.butter(1, 0.1*F1/PLL['Fs'])
    DET = {'Num':B, 'Den':A, 'THR':75}

    # Reserva de memoria y valores iniciales
    n = 99
    nTAM = Input.size
    mTAM = int(np.ceil(nTAM/CIC['R']))
    x_mixed = np.zeros([mTAM])
    x_lpf = np.zeros([mTAM])
    DVCO = np.zeros([mTAM])
    Acc = 0
    DDSo = 0

    # Filtrado CIC
    x_cic = sig.lfilter(CIC['Num'], CIC['Den'], Input)

    # Diezmado
    x_cic_int = x_cic[::CIC['R']]

    # Filtrado Comb
    x_comb_dc = sig.lfilter(Comb['Num'], Comb['Den'], x_cic_int)

    # PLL
    for m in range(n, mTAM):

        # Detección de fase
```

```
78          x_mixed[m] = x_comb_dc[m] * DDSo * PLL['Gain']
79
80          # LPF
81          x_lpf[m] = x_mixed[m] + PLL['Comb_Num'][-1] * x_mixed[m-PLL['Comb_K']]
82
83          # NCO
84          Acc = np.mod(int(np.round(Acc+ (F1 + PLL['S']*x_lpf[m])*PLL['Acc_max']/
        PLL['Fs'])),
85                       PLL['Acc_max'])
86          DDSo = PLL['LUT'][Acc]
87          DVCO[m] = DDSo
88
89      # Detección
90      x_1 = np.abs(DVCO[3*PLL['PPP_pi_2']-2:-1] * x_comb_dc[0:1-3*PLL['PPP_pi_2'
        ]])
91
92      # Butterworth LPF
93      x_1f = sig.lfilter(DET['Num'], DET['Den'], x_1)
94
95      # Tiempo de ocupación
96      c = np.where(x_1f > DET['THR'])[0]
97      c = c.size
98      c_time = c/PLL['Fs']
99      c_ratio = c/(mTAM)
100     print('c_time  = ', c_time)
101     print('c_ratio = ', c_ratio)
102     return (c_time, c_ratio)
103
104 import scipy.io
105 import numpy as np
106
107 #acars_signal = scipy.io.loadmat('se_acars_6_MENSA')
108 acars_signal = scipy.io.loadmat('ZE_17_131825')
109 Fs = int(acars_signal['x']['Fs'])
110 Input = np.abs(acars_signal['x']['IQ'][0,0])[:,0]
111 del(acars_signal)
112 (c_time, c_ratio) = acars_detector(Input, Fs)
```

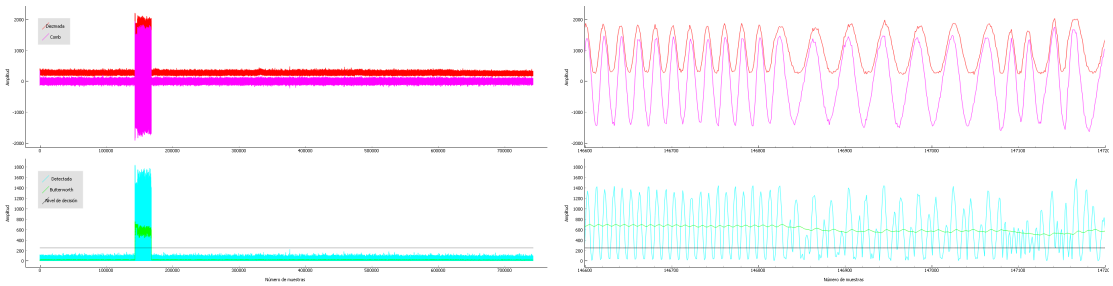
A partir de la línea 107 se encuentra el ejemplo. Como las capturas están guardadas en archivos de MATLAB se usa el comando *loadmat* para obtener sus datos. Tras obtener la frecuencia de muestreo y el valor absoluto de los datos IQ se procede a implementar el detector. La señal se borra de la memoria para ahorrar espacio. La llamada a la función es sencilla y no es necesario utilizar las variables de salida, es decir, la llamada puede ser simplemente el comando *acars_detector(Input, Fs)*.

5.4 Comparación de resultados

En esta sección se van a presentar los resultados al ejecutar el código en Python y analizar sus similitudes y diferencias respecto a MATLAB. Como ya se ha explicado, no es posible representar gráficamente las señales antes del diezmado, por lo que solo se pueden analizar las de después.

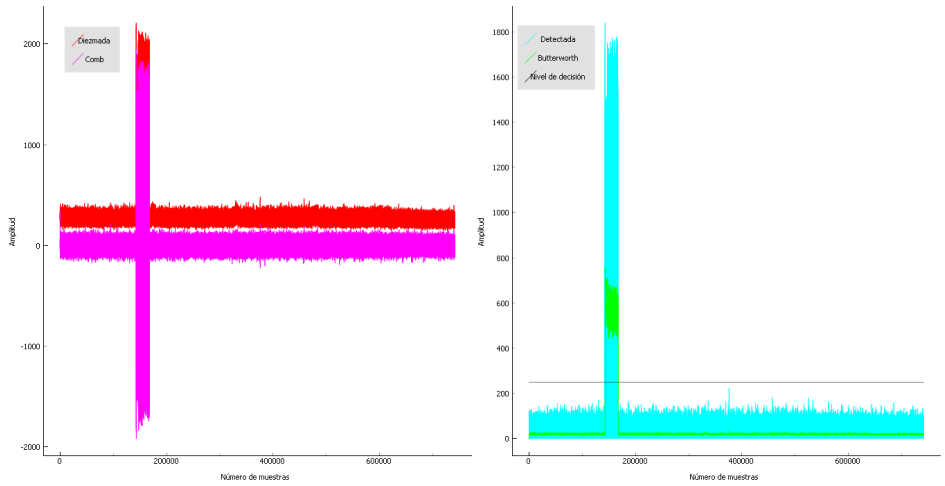
Empezando por la captura con un único mensaje ACARS, el proceso de detección se muestra en la Figura 5.20. El procesamiento ha funcionado igual que en MATLAB, obteniendo la detección sin problemas. En la Figura 5.21 se muestra junto a la obtenida en MATLAB. Se puede ver que son iguales. Pasando a la captura con varios mensajes, el resultado de la Figura 5.22 demuestra que tampoco hay diferencias respecto a usar el detector en MATLAB.

Los resultados de los tiempos de ocupación obtenidos con el detector se muestran en la Tabla 5.1. En los dos programas se obtiene el mismo resultado. Con lo que se puede concluir que el código en Python ha sido implementado correctamente.

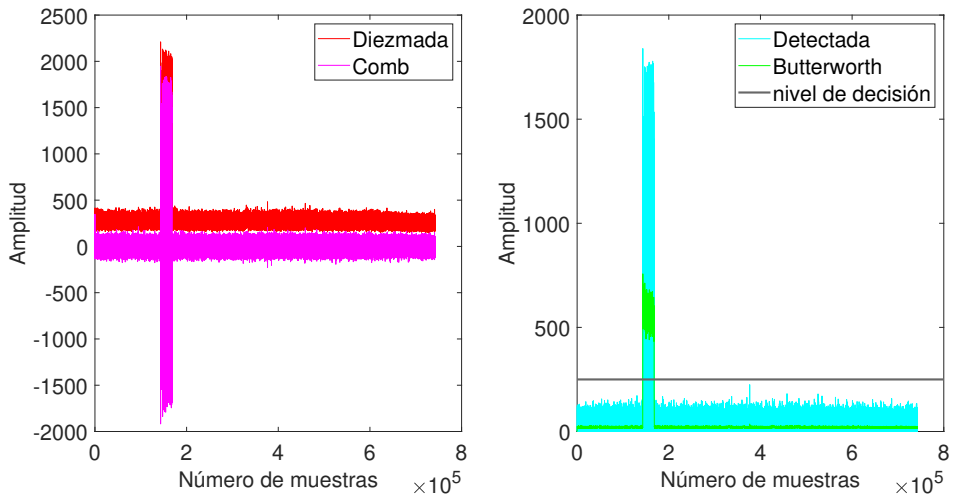


(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.20 Detección de ACARS en Python.



(a) En Python.

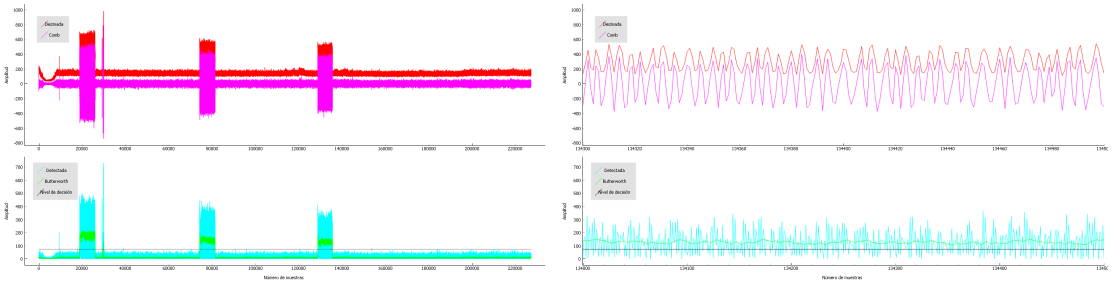


(b) En MATLAB.

Figura 5.21 Comparación detección de ACARS.

5.5 Análisis del rendimiento

Como se quiere poder utilizar este programa de forma seria, por su utilidad en investigación, es importante que no requiera mucho tiempo de procesamiento. De hecho, ya se han comentado ciertos cambios en el código y decisiones al elegir filtros guiados fundamentalmente por reducir el tiempo de procesamiento. Se va



(a) Señal completa. **(b)** Ampliación mostrando varios periodos.

Figura 5.22 Detección de varios ACARS en Python.

Tabla 5.1 Resultados de la detección.

Captura con	1 mensaje	Varios mensajes
Tiempo de ocupación (s)	0.5160	2.6111
Porcentaje de utilización (%)	3.33	9.19

obtener el tiempo que tarda la función en completarse. Este valor se ha obtenido con el comando *timeit* de MATLAB, que ejecuta el código varias veces y da una medida del tiempo requerido para su ejecución. Es necesario precisar que en este tiempo no entra lo que se tarda en cargar los datos de la captura de la señal.

Al usar el código con la captura en la que aparece una única transmisión ACARS, el programa tarda unos 4.15 segundos en ejecutarse. También se ha obtenido el tiempo en procesar la captura en la que aparecen varios mensajes ACARS, unos 1.27 segundos. Esta captura dura unos 28.4 segundos mientras que la primera dura 15.5 segundos. La explicación a porqué es más rápida la captura más larga es sencilla. Se debe a la frecuencia de muestreo utilizada. La captura larga tiene una frecuencia de muestreo de 240 KHz y la corta 1.44 MHz. Esto influye en el número de muestras de cada una. Teniendo la captura larga casi 7 millones de muestras mientras que la corta tiene algo más de 22 millones de muestras. Por tanto, en el tiempo de procesamiento influye mucho el número de muestras. Así, si se quieren obtener capturas de gran duración, es mejor utilizar una frecuencia de muestreo pequeña. Como no se quiere demodular la señal captada, no es especialmente importante usar una frecuencia de muestreo alta, ya que no es necesaria una precisión exacta en los datos recibidos.

Desafortunadamente las capturas utilizadas son de pocos segundos. Esto se debe a dos razones: que tienen un tamaño bastante grande (14 Kb y 58Kb) para su corta duración y a no haber podido realizar capturas propias con el RTL-SDR debido a la situación que se ha sufrido estos meses. Se quiere investigar si es posible capturar la señal de los canales en un formato de archivo diferente que requiera menor tamaño para la misma duración. Por ahora se ha probado con los formatos .npy, .mat y .wav, pero ninguno supone una disminución del tamaño del archivo.

Tabla 5.2 Comparación de tiempos de procesamiento.

Captura con	1 mensaje		Varios mensajes	
Programa utilizado	Python	MATLAB	Python	MATLAB
Tiempo en ejecutar el detector (s)	10	4.15	3.2	1.27

Por último, se va a comparar el rendimiento del programa. En la Tabla 5.2 aparecen los tiempos que tarda cada programa en procesar las señales capturadas. Hay que mencionar que en Python no existe una alternativa similar al *timeit* de MATLAB. Los tiempos se han obtenido ejecutando varias veces el programa con marcas de tiempo, colocadas al principio y al final del procesamiento, y haciendo la media de los tiempos obtenidos. El resultado es que Python es unas 2.5 veces más lento que MATLAB. Esto puede deberse a que al ser nuevo en el uso de Python, no se haya conseguido optimizar del todo el código utilizado. También puede ser debido a que MATLAB es un programa especializado en este tipo de procesamiento de datos, mientras que en Python hay que hacer uso de librerías específicas. Para saber algo más sería necesario probar el detector

en estos dos programas en distintos ordenadores modernos y con diferentes sistemas operativos. Como ya se ha mencionado, el ordenador utilizado ha dado algunos problemas de memoria con Python, y eso ha podido influir en su rapidez. Tras analizar un poco más se ha comprobado que el 90 % del tiempo de ejecución del programa en Python se debe al bucle for en el que se realiza el PLL. Por tanto, una posible solución es buscar una alternativa. Es probable que exista alguna librería o código creado por algún usuario que implemente un PLL de forma más eficaz. Por desgracia, no ha sido posible localizarlo.

Como conclusión, el detector se ha implementado satisfactoriamente tanto en MATLAB como en Python. En este último aún se puede llegar a optimizar para hacerlo más rápido, pero se sale del conocimiento del autor del trabajo. Se ha visto que el detector puede tener problemas si la señal captada no tiene mucha potencia. Esto puede resolverse de varias maneras:

- Analizar la potencia antes de usar el detector e introducir como variable de entrada el nivel de decisión deseado. Esto implica más trabajo para el usuario, por lo que no es deseable. Además, requeriría adivinar qué nivel es bueno para cada señal.
- Dentro del código del programa introducir unas líneas que midan el nivel del ruido del canal y establecer el nivel de decisión por encima de éste.
- Mejorar el filtro de Butterworth de forma que el ruido del canal se reduzca a un nivel imperceptible. De esta forma, se podría seleccionar un nivel de decisión pequeño sin que considere el ruido como parte de un mensaje.

6 Conclusiones

En este proyecto se ha creado una aplicación capaz de detectar la presencia de mensajes del sistema de comunicaciones aeronáuticas ACARS en una captura de un canal de radiofrecuencias. Se ha demostrado su funcionamiento con ejemplos de capturas reales y se ha comprobado que permite obtener los tiempos de ocupación de esos canales.

Se han analizado los dispositivos RTL-SDR y se han explorado algunas de sus posibles aplicaciones y programas. Es un dispositivo barato y muy versátil. Se han visto la gran cantidad de programas que existen y los que aún están en desarrollo, que permiten la utilización de estos dispositivos en ordenadores de todo tipo de sistemas operativos e incluso en teléfonos móviles. Se han probado tres programas concretos como receptores y analizadores de espectro: SDRSharp, MATLAB y Python. Se han comparado y se ha llegado a la conclusión de que para usar los dispositivos como simples receptores para ver el espectro o incluso para decodificar varios tipos de señales la mejor opción es SDRSharp y programas similares. En cuanto a MATLAB y Python, su uso es más oportuno en el campo de la investigación y enseñanza. Permiten, una vez capturados los datos I/Q de la señal, trabajar sobre estos para realizar operaciones de procesamiento de señal. Además, Python y MATLAB ofrecen la oportunidad de crear todo tipo de aplicaciones con los RTL-SDR. Cualquier aplicación de las examinadas debería poder realizarse con estos programas.

Se ha explicado el sistema de comunicaciones ACARS en profundidad. Se ha visto que la información se transmite en paquetes modulados de distinta manera según el canal de radiofrecuencia utilizado para la transmisión. El trabajo se ha centrado en la modulación MSK de los canales VHF. El conocimiento sobre esa modulación ha permitido realizar el detector. No se ha entrado a analizar la demodulación de los mensajes por dudas la legalidad de esta acción. Debido a la situación provocada por el coronavirus no ha sido posible realizar captaciones de transmisiones ACARS, por lo que no ha sido posible cumplir con una de las intenciones del proyecto: analizar el uso o tráfico de los canales reservados para ACARS.

Respecto a Python, se ha utilizado el intérprete Spyder junto con librerías específicas relacionadas con la representación de datos, el procesamiento de señales y el trabajo con matrices. Gracias a la librería *Pyrtlsdr* se ha podido controlar el RTL-SDR. Se han adquirido los suficientes conocimientos de este lenguaje y algunas de sus librerías como para crear códigos propios para el uso de estos dispositivos. Esto ha ayudado a la hora de pasar el código del detector de MATLAB al lenguaje Python.

En cuanto al procesamiento requerido para detectar la presencia de mensajes modulados ACARS, se han explicado los distintos elementos utilizados. Lo primero es realizar un diezmado, para tener muestras que procesar y así requerir menor tiempo de procesamiento. Para evitar el posible aliasing de este proceso se utiliza un filtro CIC. Después, se elimina la componente de DC de la señal mediante un filtro Comb. Esto se hace para asegurar el buen funcionamiento del PLL que se usa a continuación. Con el PLL se crea una señal sinusoidal que se sincroniza con las portadoras de 1200 y 2400 Hz de la modulación de ACARS. Multiplicando esa señal y la capturada se obtiene la detección, ya que se crea una señal de valor nulo excepto cuando existen mensajes ACARS. A esa señal se le aplica un filtro Butterworth para eliminar las altas frecuencias y dejar únicamente componentes de DC que indiquen si existen transmisiones ACARS en la captura procesada.

Este procesamiento se ha integrado en una función de MATLAB y en Python. Utiliza los distintos elementos para detectar la señal y dado un nivel de amplitud concreto, lo usa como umbral para decidir si existe detección. Gracias al umbral, se obtiene el tiempo de ocupación del canal, tanto en segundos como en porcentaje respecto al tiempo total de la captura introducida. Se ha probado el funcionamiento del detector con dos señales

reales. Se ha comprobado que el detector funciona igual en los dos programas y que se obtienen los mismos resultados.

Se han encontrado dos problemas que no se han podido resolver. Por un lado, las capturas son de solo unos segundos debido a que ocupan mucha memoria. Se han probado varios formatos de archivo en los que guardar la captura sin que ninguno reduzca significativamente el espacio requerido. Por otro lado, en cuanto al rendimiento del código, ha resultado considerablemente más lento, en tiempo de ejecución, en Python que en MATLAB. En MATLAB no requiere más que unos pocos segundos, pero en Python puede llegar a los 10 segundos. Esto no es aceptable si se quiere trabajar con capturas muy grandes, ya que el tiempo de procesamiento podría ser demasiado.

Finalmente, existen posibles vías de trabajo futuras. Se puede intentar utilizar el detector con capturas realizadas con el RTL-SDR y con los tiempos que calcula, realizar análisis estadísticos para examinar la ocupación de cada canal según distintas variables como la hora del día, la estación del año, los días festivos y parecidos. Esto serviría tanto para analizar la congestión de estos canales como para obtener una medida del tráfico aéreo, compararlo con los datos reales y ver así la popularidad del sistema de comunicaciones. También se puede intentar optimizar más el código, teniendo en cuenta que la parte que más tiempo requiere es el PLL. Otra propuesta es implementar el código en lenguaje C y comprobar si el rendimiento mejora.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] C. R. Spitzer, U. Ferrell, and T. Ferrell, *Digital avionics: Handbook*, 2017.

[2] IEEE Communications Society, *IEEE Standard Definitions and Concepts for Dynamic Spectrum Access: Terminology Relating to Emerging Wireless Networks, System Functionality, and Spectrum Management*, September 2008.

[3] J. Mitola, *Software Radio: Wireless Architecture for the 21st Century*, 2000.

[4] ——, "Software radio architecture: A mathematical perspective," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, pp. 514 – 538, April 1999.

[5] M. Mishra, A. Potnis, P. Dwivedy, and S. K. Meena, "Software defined radio based receivers using RTL - SDR: A review," in *International Conference on Recent Innovations in Signal Processing and Embedded Systems (RISE)*, October 2017.

[6] B. Stewart, K. Barlee, D. Atkinson, and L. Crockett, *Software Defined Radio Workflow Using MATLAB & Simulink and the RTL-SDR*, 2015. [Online]. Available: https://www.desktopsdr.com/download-files

[7] "RTL2832U DVB-T COFDM Demodulator + USB 2.0," 2019, accessed on 19-04-2020. [Online]. Available: https://www.realtek.com/en/products/communications-network-ics/item/rtl2832u

[8] W. Kadman, "R820T Rafael Micro Advanced Digital Silicon Tuner - Datasheet Application," March 2013, accessed on 19-04-2020. [Online]. Available: http://radioaficion.com/cms/r820t-rafael-micro/

[9] "About RTL-SDR," accessed on 19-04-2020. [Online]. Available: https://www.rtl-sdr.com/about-rtl-sdr/

[10] Electronic communication Committee, "The European table of frequency allocations and application in the frequency range 9 kHz to 3000 GHz," *Erc Report 25*, March 2019. [Online]. Available: https://www.ecodocdb.dk/download/2ca5fcbd-4090/ERCREP025.pdf

[11] "applications," accessed on 19-06-2020. [Online]. Available: https://www.rtl-sdr.com/tag/applications-2/

[12] R. W. Stewart, L. Crockett, D. Atkinson, K. Barlee, D. Crawford, I. Chalmers, M. McLernon, and E. Sozer, "A low-cost desktop software defined radio design environment using MATLAB, simulink, and the RTL-SDR," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 64–71, September 2015.

[13] H. Miyashiro, M. Medrano, J. Huarcaya, and J. Lezama, "Software defined radio for hands-on communication theory," in *Proceedings of the 2017 IEEE 24th International Congress on Electronics, Electrical Engineering and Computing (INTERCON)*, October 2017.

[14] B. Uengtrakul and D. Bunnjaweht, "A cost efficient software defined radio receiver for demonstrating concepts in communication and signal processing using Python and RTL-SDR," *2014 4th International Conference on Digital Information and Communication Technology and Its Applications (DICTAP)*, pp. 394–399, May 2014.

[15] E. Santos-Luna, A. Prieto-Guerrero, R. Aguilar-Gonzalez, V. Ramos, M. Lopez-Benitez, and M. Cardenas-Juarez, "A Spectrum Analyzer Based on a Low-Cost Hardware-Software Integration," *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 607–612, December 2019.

[16] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng, "Towards commoditized real-time spectrum monitoring," September 2014. [Online]. Available: https://sites.cs.ucsb.edu/~ravenben/publications/pdf/crowdsense-hotwireless14.pdf

[17] W. T. Chen, K. T. Chang, and C. P. Ko, "Spectrum monitoring for wireless TV and FM broadcast using software-defined radio," *Multimedia Tools and Applications*, vol. 75, no. 16, pp. 9819–9836, July 2015.

[18] G. Pirazzi, L. Cucchi, D. Marigi, and C. Dionisio, "A GNSS integrity monitoring station with Software Defined Radio and low cost receivers," *2012 IEEE 1st AESS European Conference on Satellite Telecommunications, (ESTEL)*, pp. 1–6, 2012.

[19] K. V. Ranga Rao, A. Ravikumar, R. Kumaraiah, and V. R. Ghanta, "Detection and decoding of ask signals using an SDR receiver," *Compusoft*, vol. 7, no. 9, pp. 2807–2813, 2018.

[20] N. Marriwala, "Error control coding for software defined radios using soft computing," in *Advances in Intelligent Systems and Computing*, 2020.

[21] "Quick Start Guide," accessed on 18-04-2020. [Online]. Available: https://www.rtl-sdr.com/rtl-sdr-quick-start-guide/

[22] "Download Microsoft .NET Framework 4.7 ," accessed on 20-06-2020. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=55167

[23] "Download Microsoft Visual C++ 2010 SP1 Redistributable," accessed on 20-06-2020. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=8328

[24] "SDR# and Airspy Downloads - airspy.com," accessed on 18-04-2020. [Online]. Available: https://airspy.com/download/

[25] "Software-Defined Radios," accessed on 19-04-2020. [Online]. Available: https://arachnoid.com/software_defined_radios/

[26] "The BIG List of RTL-SDR Supported Software," accessed on 20-06-2020. [Online]. Available: https://www.rtl-sdr.com/big-list-rtl-sdr-supported-software/

[27] "PLSDR," accessed on 20-06-2020. [Online]. Available: https://arachnoid.com/PLSDR/index.html

[28] "Gqrx SDR," accessed on 20-06-2020. [Online]. Available: https://gqrx.dk/

[29] "CubicSDR | Cross-Platform and Open-Source Software Defined Radio Application," accessed on 20-06-2020. [Online]. Available: https://cubicsdr.com/

[30] "HDSDR Homepage," accessed on 20-06-2020. [Online]. Available: http://www.hdsdr.de/

[31] "Software Defined Radio," accessed on 20-06-2020. [Online]. Available: https://www.sdr-radio.com/

[32] "Linrad home page." accessed on 20-06-2020. [Online]. Available: http://www.sm5bsz.com/linuxdsp/linrad.htm

[33] "SDR Apps," accessed on 20-06-2020. [Online]. Available: http://www.sdrapplications.it/

[34] "Downloads – SDRplay," accessed on 20-06-2020. [Online]. Available: http://www.sdrplay.com/downloads/#tab-tab-5037-0-0-2-5037-0

[35] "GitHub - kpreid/shinysdr," accessed on 20-06-2020. [Online]. Available: https://github.com/kpreid/shinysdr

[36] "SDR Touch - Live radio via USB," accessed on 20-06-2020. [Online]. Available: https://play.google.com/store/apps/details?id=marto.androsdr2&hl=en

[37] "Zeus Radio - software ham radio for SDR Receiver und Transceiver," accessed on 20-06-2020. [Online]. Available: https://www.hfrelectronics.com/en/zeus-radio/

[38] "GNU Radio - The Free & Open Source Radio Ecosystem · GNU Radio," accessed on 24-06-2020. [Online]. Available: https://www.gnuradio.org/

[39] "Tutorials - GNU Radio," accessed on 19-04-2020. [Online]. Available: https://wiki.gnuradio.org/index.php/Tutorials

[40] Jahnavi. S, Pooja. C. B, Santro Leeona FM, and Vijayageetha. R, "Implementation of Wide Band FM Receiver on RTL-SDR," *International Journal of Engineering Research and*, vol. V5, no. 05, pp. 511–516, May 2016.

[41] K. Vachhani and R. A. Mallari, "Experimental study on wide band FM receiver using GNURadio and RTL-SDR," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2015.

[42] M. Abirami, V. Hariharan, M. B. Sruthi, R. Gandhiraj, and K. P. Soman, "Exploiting GNU radio and USRP: An economical test bed for real time communication systems," in *2013 4th International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2013.

[43] "List of SDRSharp Plugins," accessed on 18-04-2020. [Online]. Available: https://www.rtl-sdr.com/sdrsharp-plugins/

[44] "SDRSharp Users Guide," accessed on 18-04-2020. [Online]. Available: https://www.rtl-sdr.com/sdrsharp-users-guide/

[45] "MATLAB - El lenguaje del cálculo técnico - MATLAB & Simulink," accessed on 21-06-2020. [Online]. Available: https://es.mathworks.com/products/matlab.html

[46] "R2020a - Actualizaciones de las familias de productos MATLAB y Simulink - MATLAB & Simulink," accessed on 21-06-2020. [Online]. Available: https://es.mathworks.com/products/new_products/latest_features.html

[47] "Communications Toolbox - MATLAB & Simulink," accessed on 21-06-2020. [Online]. Available: https://www.mathworks.com/products/communications.html

[48] "DSP System Toolbox - MATLAB & Simulink," accessed on 21-06-2020. [Online]. Available: https://www.mathworks.com/products/dsp-system.html

[49] "Signal Processing Toolbox - MATLAB," accessed on 21-06-2020. [Online]. Available: https://www.mathworks.com/products/signal.html

[50] "Communications Toolbox Support Package for RTL-SDR Radio," accessed on 21-06-2020. [Online]. Available: https://es.mathworks.com/matlabcentral/fileexchange/44991-communications-toolbox-support-package-for-rtl-sdr-radio

[51] "Download – desktopSDR.com," accessed on 21-06-2020. [Online]. Available: https://www.desktopsdr.com/download-files

[52] B. Laporte-Fauret, R. Tajan, and G. Ferré, "ADS-B Transmitter / Receiver Simulation to Decode Real Aircraft Trajectories with Software Defined Radios," *2019 29th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE)*, pp. 2–7, 2019.

[53] "Download Python," 2020, accessed on 18-04-2020. [Online]. Available: https://www.python.org/downloads/

[54] "pyrtlsdr · PyPI," accessed on 19-04-2020. [Online]. Available: https://pypi.org/project/pyrtlsdr/

[55] "The world's leading software development platform · GitHub," accessed on 01-07-2020. [Online]. Available: https://github.com/

[56] "roger- (Roger D) · GitHub," accessed on 01-07-2020. [Online]. Available: https://github.com/roger-

[57] "Project | DIY SDR for fun | Hackaday.io," accessed on 23-06-2020. [Online]. Available: https://hackaday.io/project/165403/logs?sort=oldest

[58] S. Sun, "ACARS data identification and application in aircraft maintenance," in *2009 1st International Workshop on Database Technology and Applications (DBTA)*, 2009, pp. 255–258.

[59] M. Smith, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "Undermining Privacy in the Aircraft Communications Addressing and Reporting System (ACARS)," *Proceedings on Privacy Enhancing Technologies*, 2018.

[60] M. Strohmeier, "Representation of the ACARS sub-systems." [Online]. Available: https://www.researchgate.net/figure/Representation-of-the-ACARS-sub-systems_fig1_303886077

[61] M. Tooley and D. Wyatt, "Aircraft communications and navigation systems principles, operation and maintenance," 2007.

[62] A. R. Inc., "Arinc 618-5, air/ground character-oriented protocol specification," August 2000.

[63] S. Haykin, *Digital Communications Systems*, 2009.

[64] P. Zhang and P. He, "The research of ACARS signal demodulation method based on spectrum estimation," *3rd International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, pp. 1007–1010, September 2013.

[65] A. Roy, "Secure Aircraft Communications Addressing and Reporting System (ACARS)," in *20th DASC. 20th Digital Avionics Systems Conference*, 2001.

[66] "ACARS (Aircraft Communication Addressing and Reporting System) – Blog SRSI," March 2017, accessed on 19-04-2020. [Online]. Available: https://blogsrsi.wordpress.com/2017/03/01/acars-aircraft-communication-addressing-and-reporting-system/

[67] "RTL-SDR Tutorial: Receiving Airplane Data with ACARS," accessed on 19-04-2020. [Online]. Available: https://www.rtl-sdr.com/rtl-sdr-radio-scanner-tutorial-receiving-airplane-data-with-acars/

[68] C. Hill, *Learning Scientific Programming with Python*, 2015.

[69] R. Johansson, *Numerical python: Scientific computing and data science applications with numpy, SciPy and matplotlib, Second edition*, 2018.

[70] M. Pilgrim, "Dive Into Python 3," 2009. [Online]. Available: https://diveintopython3.problemsolving.io/

[71] "The Python Tutorial — Python 3.8.2 documentation," accessed on 19-04-2020. [Online]. Available: https://docs.python.org/3/tutorial/index.html

[72] "IDLE — Python 3.8.3 documentation." [Online]. Available: https://docs.python.org/3/library/idle.html

[73] "Anaconda Python/R Distribution - Free Download," accessed on 18-04-2020. [Online]. Available: https://www.anaconda.com/distribution/#download-section

[74] "Spyder Website," accessed on 23-06-2020. [Online]. Available: https://www.spyder-ide.org/

[75] "Managing packages — conda," accessed on 23-06-2020. [Online]. Available: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html

[76] "Matplotlib: Python plotting — Matplotlib 3.2.2 documentation," accessed on 23-06-2020. [Online]. Available: https://matplotlib.org/

[77] "Spectrum Representations — Matplotlib 3.1.2 documentation," accessed on 23-06-2020. [Online]. Available: https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/spectrum_demo.html#sphx-glr-gallery-lines-bars-and-markers-spectrum-demo-py

[78] "Psd Demo — Matplotlib 3.1.2 documentation," accessed on 23-06-2020. [Online]. Available: https://matplotlib.org/3.1.1/gallery/lines_bars_and_markers/psd_demo.html#sphx-glr-gallery-lines-bars-and-markers-psd-demo-py

[79] "Tutorials — Matplotlib 3.1.2 documentation," accessed on 23-06-2020. [Online]. Available: https://matplotlib.org/3.1.1/tutorials/index.html

[80] A. Devert, *Matplotlib Plotting Cookbook*, 2014.

[81] S. Tosi, *Matplotlib for Python Developers*, 2009.

[82] "NumPy," accessed on 23-06-2020. [Online]. Available: https://numpy.org/

[83] "NumPy User Guide — NumPy v1.18 Manual," accessed on 23-06-2020. [Online]. Available: https://numpy.org/doc/stable/user/index.html

[84] O. Travis, *Numpybook*, 2007.

[85] "SciPy.org," accessed on 24-06-2020. [Online]. Available: https://www.scipy.org/

[86] "Scipy Lecture Notes — Scipy lecture notes," accessed on 24-06-2020. [Online]. Available: https://scipy-lectures.org/

[87] "PyQtGraph - Scientific Graphics and GUI Library for Python," accessed on 24-06-2020. [Online]. Available: http://www.pyqtgraph.org/

[88] "Welcome to the documentation for pyqtgraph — pyqtgraph 0.11.0 documentation," accessed on 24-06-2020. [Online]. Available: https://pyqtgraph.readthedocs.io/en/latest/

[89] "QSpectrumAnalyzer Pypi," accessed on 01-07-2020. [Online]. Available: https://pypi.org/project/QSpectrumAnalyzer/1.1/

[90] T. E. Oliphant, "Python for Scientific Computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[91] E. B. Hogenauer, "An Economical Class of Digital Filters for Decimation and Interpolation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 155–162, 1981.

[92] "Feedforward Comb Filters," accessed on 27-06-2020. [Online]. Available: https://web.archive.org/web/20110606210608/https://ccrma.stanford.edu/~jos/waveguide/Feedforward_Comb_Filters.html

[93] "The Basics of Direct Digital Synthesizers (DDSs) | DigiKey," accessed on 29-06-2020. [Online]. Available: https://www.digikey.com/en/articles/the-basics-of-direct-digital-synthesizers-ddss