

UNIVERSIDAD MIGUEL HERNÁNDEZ

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y  
AUTOMÁTICA INDUSTRIAL



UNIVERSITAS  
*Miguel Hernández*



Biblioteca

“DISEÑO E IMPLEMENTACIÓN DE  
SISTEMA DE RECEPCIÓN Y  
PROCESAMIENTO DE SEÑALES RF  
PARA INCORPORACIÓN A APP DE  
CONTROL DE PESCA”

TRABAJO DE FIN DE GRADO

Febrero – 2024

Autor: Alejandro Bermejo Jimeno  
Director: Alejandro Pomares Padilla



## RESUMEN

El presente proyecto tiene como objetivo la obtención de nuevos datos para el entrenamiento de la APP de Control de Pesca FAMIS mediante el diseño y prueba de un sistema independiente y autónomo de recepción y procesamiento de señales de Radio-Frecuencia. Todo ello mediante el uso de hardware de bajo coste económico y reducido consumo energético basado en receptor Rtl-Sdr y microcontroladores programados en lenguaje de alto nivel Python.

Tras estudiar y probar diferentes soluciones, además de solventar ciertos problemas, se obtiene un sistema que cumple con los requisitos mencionados.

Finalmente, se proponen nuevos desarrollos del sistema así como futuras investigaciones.





## **ABSTRACT**

The present project aims to obtain new data used for the training of the FAMIS Fishing Control APP through the design and test of an independent and autonomous system for receiving and processing Radio-Frequency signals with the use of low-cost and reduced energy consumption hardware based on a Rtl-Sdr receiver and several microcontrollers programmed in high-level Python language.

After studying and testing different solutions, as well as addressing certain issues, a system that meets the mentioned requirements is obtained.

Finally, new developments of the system are proposed as well as future research.





## **AGRADECIMIENTOS**

Me gustaría agradecer especialmente a mi tutor de este Trabajo, el Dr. Alejandro Pomares Padilla, por su atención, ayuda y paciencia. Así como los conocimientos transmitidos y el trato recibido.

También, agradezco a mis compañeros del Grado su afecto y amistad en todo momento.

Por último, deseo dar las gracias al profesorado de la Escuela Politécnica por formarme en el noble mundo de la ingeniería, la industria y la empresa. Esto me ha permitido crecer profesional y personalmente.







# ÍNDICE DE CONTENIDOS

RESUMEN .....	2
ABSTRACT .....	4
AGRADECIMIENTOS .....	6
Capítulo 1. Introducción .....	13
1.1.    Motivación.....	14
1.2.    Objetivos .....	14
1.3.    Estructura de la memoria .....	15
Capítulo 2. Descripción de propuesta .....	17
2.1.    Contexto .....	18
2.2.    Fundamentos.....	25
2.3.    Antecedentes .....	27
2.4.    Propuesta .....	30
Capítulo 3. Materiales y Métodos .....	34
3.1.    Implementación .....	34
3.1.1.    Hardware .....	40
3.1.2.    Software .....	45
3.2.    Configuración .....	47
3.3.    Programación .....	53
3.3.1.    Adquisición de los datos .....	53
3.3.2.    Representación de los datos.....	58
Capítulo 4. Resultados y Discusión .....	65
4.1.    El Sistema .....	66
4.2.    Los Datos .....	70
Capítulo 5. Conclusiones .....	80
Capítulo 6. Futuras investigaciones.....	82
BIBLIOGRAFÍA .....	84
ANEXO I: Ejemplo de comunicación TCP/IP con SDR .....	87

# ÍNDICE DE FIGURAS

Figura 1. Red de Deriva.....	21
Figura 2. Red FAD no enmallante (izquierda) y red FAD tradicional (derecha).....	21
Figura 3. Buque Atunero 'Txori Argi' largando red de cerco. Fuente: INPESCA.....	22
Figura 4. Ejemplos de radiobalizas: EPIRB, AIS-SART, PLB, MOB.....	24
Figura 5. Espectro radioeléctrico. Usos frecuentes.....	25
Figura 6. B/O Miguel Oliver. Litoral Catalán.....	28
Figura 7. Interfaz de usuario App FAMIS. Flujo de información.....	28
Figura 8. Modelo de Red Neuronal.....	29
Figura 9. Placa "dongle" de SDR. Chip RTL2832U.....	31
Figura 10. Arduino UNO, ejemplo de Microcontrolador.....	32
Figura 11. Raspberry Pi, ejemplo de Microcomputador.....	32
Figura 12. ESP32-S3-USB-OTG. ESP32-S3-DevKitM-1.....	35
Figura 13. Repositorio MicroPython para descarga firmware ESP32-S3.....	36
Figura 14. Firmware CircuitPython para ESP32-S3-USB-OTG.....	37
Figura 15. Firmware CircuitPython para ESP32-S3-DevKitM-1.....	37
Figura 16. Ventana selección "flash_download_tool".....	38
Figura 17. Kit NooElec NESDR smartXTR.....	41
Figura 18. Raspberry Pi Zero 2W.....	42
Figura 19. Ejemplo de Control Remoto de automóvil (MHz).....	43
Figura 20. Ejemplo de Control Remoto de coche RC (GHz).....	43
Figura 21. Prototipo de laboratorio. Modelo Experimental.....	44
Figura 22. Herramienta Raspberry Imager v1.8.1.....	47
Figura 23. Selección de dispositivo Raspberry.....	48
Figura 24. Selección de Sistema Operativo Raspberry.....	48
Figura 25. Configuración PuTTY.....	49
Figura 26. Autenticación RealVNC Viewer.....	49
Figura 27. Escritorio Linux (Debian) en RPi.....	50
Figura 28. IDE Thonny en RPi. Manage packages.....	50
Figura 29. Archivo rc.local.....	52
Figura 30. Estructura fichero 'data.txt'.....	53
Figura 31. Medidas de Voltaje y Temperatura RPi.....	67
Figura 32. Monitor recursos del sistema (htop).....	68
Figura 33. Lista de procesos en ejecución del sistema (top).....	68
Figura 34. Uso de memoria y disco duro.....	69
Figura 35. Captura fichero de texto 'data.txt'. Detección de 'Baliza'.....	71

Figura 36. Captura software SDRSharp-AirSpy. Detección de 'Baliza' .....	72
Figura 37. Captura fichero de texto 'data.txt'. Detección de 'Radar'. .....	73
Figura 38. Captura software SDRSharp-AirSpy. Detección de 'Radar' .....	74
Figura 39. Captura AirSpy. Banda emisoras FM (100 MHz).....	75
Figura 40. Señal Original.....	76
Figura 41. Valor Absoluto de la Señal Original. ....	76
Figura 42. FFT de la Señal Original.....	77
Figura 43. Valor Absoluto de la FFT.....	77
Figura 44. PSD de la Señal Original.....	78
Figura 45. Valor Absoluto de la PSD. ....	78



## ÍNDICE DE TABLAS

Tabla 1. La pesca en cifras. Fuente: CEPESCA (2019). .....	18
Tabla 2. Clasificación de las Artes de Pesca. Tipo de uso. ....	20
Tabla 3. Comandos Linux construcción librería 'librtlsdr' .....	51
Tabla 4. Comandos Linux carga automática de scripts.....	51
Tabla 5. Consumos energéticos.....	67





## Capítulo 1. Introducción

La pesca es una actividad fundamental del sector primario, fuente principal de alimento y generadora de empleo y crecimiento económico-social.

Desde los inicios de esta práctica, los artesanos del mar han sabido desempeñar su oficio manteniendo un equilibrio con los recursos naturales. Pero en la actualidad, con el ritmo acelerado de las sociedades y los mercados, debido a la globalización así como a la industrialización, han aparecido peligros para la sostenibilidad de dichos recursos.

Es por ello por lo que son tan importantes las campañas de concienciación, regulación, control y persecución de las actividades delictivas en este ámbito.

Hoy en día, los Estados así como las Organizaciones Internacionales emplean soluciones como la defensa de las aguas por medio de fuerzas armadas, el uso de sistemas informatizados con comunicaciones satelitales o el control de la venta de capturas en mercados y lonjas.

Sin embargo, estas soluciones más efectivas por el momento son no obstante muy costosas en términos humanos, económicos y tecnológicos y poco precisas en ciertos ámbitos.

Por este motivo nace el sistema FAMIS, para ofrecer una solución precisa, rápida, sencilla y al alcance de todo usuario. Pudiendo extrapolarse el control de pesca a cualquier actividad por pequeña y específica que sea, en este ámbito. Todo ello gracias al uso de nuevas tecnologías como la obtención de información generada por el uso de sensores en smartphome Android, la implementación de técnicas de aprendizaje profundo mediante modelo de red neuronal.

## **1.1. Motivación**

La motivación de este proyecto es el uso de nuevas tecnologías en el sector marítimo dedicado al control de pesca y la seguridad en el mar, mediante la aplicación de instrumental electrónico multipropósito y la programación en lenguajes de alto nivel.

Con el fin de preservar los ecosistemas y recursos marinos así como investigar y desarrollar nuevas soluciones en este ámbito.

## **1.2. Objetivos**

El objetivo principal de este trabajo de investigación es el diseño y puesta en funcionamiento, en fase experimental, de un sistema independiente y de gran autonomía de recepción y procesamiento de señales RF, sirviendo de información adicional para incorporarla en el sistema ya existente FAMIS.

Como objetivos secundarios se proponen: el uso de hardware multipropósito de bajo coste económico y la no injerencia de la tecnología usada en el proyecto con la del propio buque o las posibles usadas en artes de pesca, para evitar conflicto entre el interés investigador y el profesional a bordo.

### 1.3. Estructura de la memoria

La memoria del proyecto queda estructurada como se indica:

- **Capítulo 1. Introducción:** Se da un contexto inicial, la motivación que lleva a realizar el proyecto así como los diferentes objetivos que han de cumplirse.
- **Capítulo 2. Descripción de propuesta:** Se describe la propuesta, justificándola con un contexto de aplicación y una serie de fundamentos teóricos.
- **Capítulo 3. Materiales y métodos:** Se describe el equipamiento usado así como los procedimientos seguidos.
- **Capítulo 4. Resultados y discusión:** Comprende los resultados obtenidos y la discusión correspondiente de los mismos.
- **Capítulo 5. Conclusiones:** Se concluye la viabilidad del sistema diseñado y el cumplimiento del trabajo con los objetivos descritos.
- **Capítulo 6. Futuras investigaciones:** Se proponen nuevas vías de investigación además de avances en el propio sistema desarrollado.



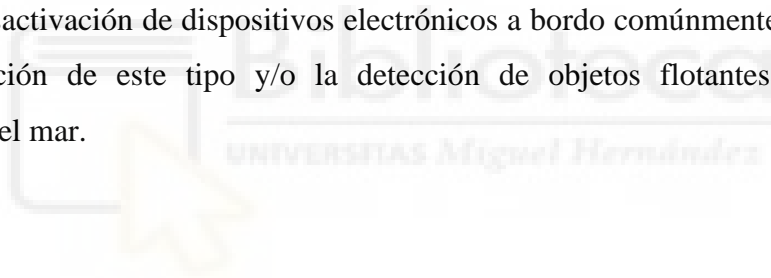


## Capítulo 2. Descripción de propuesta

En este capítulo se proporcionan conceptos básicos y fundamentos de las áreas de conocimiento tratadas en el proyecto. Asimismo se describe la propuesta a desarrollar.

Este proyecto comprende el diseño, desarrollo y prueba de un sistema independiente de obtención de datos sobre la actividad de un buque de pesca que posteriormente servirán como información adicional para el entrenamiento de modelo de red neuronal, ya desarrollado, para uso en el control de pesca mediante, la también desarrollada, correspondiente APP Android.

La propuesta trata el diseño e implementación de un sistema basado en la recepción de señales de Radio-Frecuencia mediante un receptor de Gran Banda Ancha y una serie de dispositivos electrónicos para su procesamiento con el fin de detectar variaciones en el espectro radioeléctrico estudiado que ayuden a determinar si estas corresponden a la activación/desactivación de dispositivos electrónicos a bordo comúnmente utilizados en una embarcación de este tipo y/o la detección de objetos flotantes emisores de frecuencia en el mar.



## 2.1. Contexto

En la actualidad, la pesca resulta de gran relevancia en términos económicos y sociales. En España, la actividad pesquera constituye entre el 1% y el 10% del PIB y genera alrededor de 32000 puestos de empleo directos, representando un 20,66% del empleo pesquero de la UE. <sup>[1]</sup>

La siguiente tabla contiene algunos de los datos más relevantes de la pesca en España y Europa:

Tabla 1. La pesca en cifras.

LA PESCA EN ESPAÑA-EUROPA			
	España	Europa	
Flota	8972 buques	82780 buques	11%
Empleo directo	31473	354000	9%
Capturas marinas	922564 Tm	5322194 Tm	17%
Valor de la producción	2147 M		
Importaciones de pescado	1773048 Tm		
Valor importaciones	7332,6 M	51172 M	14%
Exportaciones de pescado	1216734 Tm		
Valor exportaciones	4344,7 M	32359 M	13%
Consumo en hogares (kg/año)	23,1	24,3	
VAB pesca	1142 M		

Sin embargo, la pesca ilegal no declarada y no reglamentada (IUU, en inglés) hoy en día es una amenaza creciente para los ecosistemas y recursos del mar así como para el sector económico al que pertenece.

En España una de las instituciones que desempeña un papel importante en el control de la pesca es la Armada, con operaciones directas en las aguas territoriales, tanto oceánicas, marítimas como fluviales. <sup>[2]</sup>

Para ello es necesario el empleo de navíos así como sus respectivas dotaciones y otros recursos de distinta índole, pero en todo caso de alta inversión económica, logística y humanitaria.

Otra de las soluciones con más uso en el control de pesca es el sistema VMS (Vessel Monitoring System).

Su objetivo principal es el seguimiento de embarcaciones pesqueras por parte de autoridades marítimas y administraciones de la actividad pesquera. Para ello obtienen información en tiempo real de la posición GPS, la velocidad, el rumbo o el estado de la actividad pesquera que se esté efectuando.

Consta de transmisores instalados en las embarcaciones de pesca y estaciones receptoras en tierra.

El fin es perseguir y prevenir la pesca ilegal, hacer cumplir las regulaciones de pesca pertinentes y preservar la sostenibilidad de los recursos pesqueros. Además sirve para la mejora de la seguridad y la eficiencia en la navegación.

Otro de los sistemas satelitales usados en el Monitoreo y Seguimiento de buques es el Sistema de Identificación Automática (AIS, en inglés).

El sistema AIS se usa para el seguimiento y localización de embarcaciones. El propósito principal es la seguridad y la eficiencia del tránsito en el mar, ya que los buques conocen información de otros barcos en áreas cercanas como son la posición, la velocidad y el rumbo.

El sistema se basa en dos tipos de equipos. Por un lado se encuentran los transpondedores instalados en los barcos y por otro las estaciones ubicadas en tierra. Para ello trabaja mediante señales de frecuencia VHF y obtiene los datos de los dispositivos de localización a bordo: plotter o GPS. Cada embarcación está equipada con receptor y emisor y se identifica por un identificador único MMSI (Mobile Maritime Ship Station Identity) de 9 cifras. Las tres primeras cifras corresponden al país del pabellón de la embarcación.

En lo que se refiere a la actividad pesquera es necesario introducir las Artes de Pesca. Se entiende por Arte de Pesca a toda técnica empleada con el fin de capturar cualquier especie acuática. Estas pueden ser artesanales, también denominadas menores, o industriales.

Mientras que las artes menores son empleadas en zonas de interior, tales como ríos o lagos así como en zonas marítimas cercanas a la costa, las artes industriales por su parte operan en el resto de zonas exteriores, con intereses económicos y de suministro alimentario.

Asimismo las artes de pesca se pueden clasificar en activas y pasivas. [3]

*Tabla 2. Clasificación de las Artes de Pesca. Tipo de uso.*

<b>ARTES PASIVAS</b>	<b>ARTES ACTIVAS</b>
Redes Agulleras	Lanzas/Arpones
Tasmallos	Arrastres/Dragas
Curricanes	Chinchorros
Palangres	Redes de cerco
Nasas/Trampas	Redes de tiro

Además de estas artes de pesca cabe destacar especialmente otro tipo: Las redes de deriva, ya que son el caso de estudio de este proyecto. Como su nombre indica son redes de enmalle que flotan a la deriva y que cuentan con un lastre en su parte inferior para mantenerse vertical. Estas redes se encuentran señalizadas por boyas y/o banderas y una radiobaliza que emite una frecuencia determinada para poder ser detectadas.

Su uso, no obstante, implica un importante impacto medioambiental como la captura incidental de especies no deseadas. Por ello se han desarrollado otras formas menos perjudiciales para el ambiente marino y que no suponen un descenso significativo en las capturas. Destacan las redes FAD no enmallantes (en inglés) o Dispositivos de Agregación de Peces. Se diferencian de las redes de deriva o las redes FAD tradicionales ya que no disponen de malla vertical. Crean un hábitat artificial para atraer a los peces que posteriormente serán capturados con otros medios como las redes de cerco. [4]

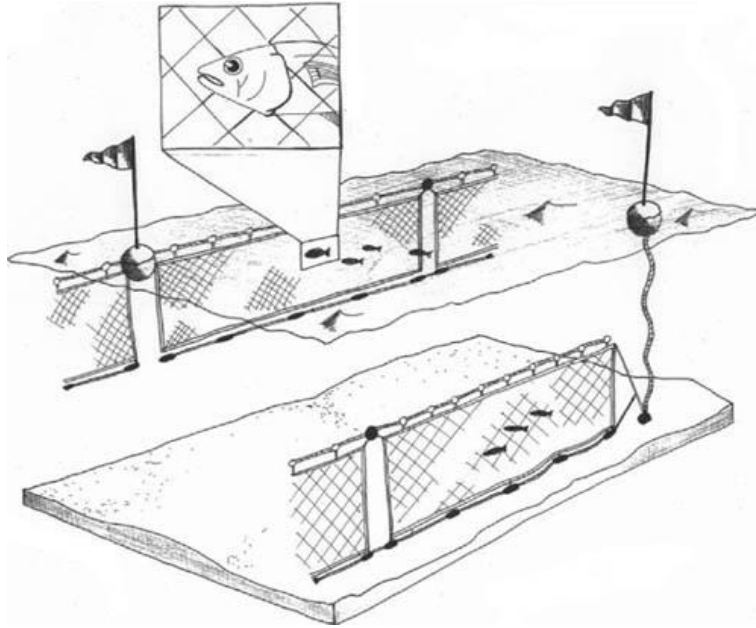


Figura 1. Red de Deriva.



Figura 2. Red FAD no enmallante (izquierda) y red FAD tradicional (derecha).

Los buques que emplean este tipo de arte de pesca suelen ser navíos de grandes dimensiones. Un ejemplo es el buque atunero Txori Argi, el mayor en la flota de la compañía INPESCA S.A. Este barco cuenta con una eslora de 106.5 m, 16 m de manga y una capacidad de carga de 2000 Toneladas. Para ello es propulsado por un motor diesel de 8000 Caballos de Potencia. [5]



*Figura 3. Buque Atunero 'Txori Argi' largando red de cerco. Fuente: INPESCA.*

Existe una multitud de dispositivos electrónicos que se pueden encontrar a bordo de un buque, concretamente en uno de pesca.

Pero de todos estos, los específicos para su uso directo en la actividad pesquera y que están presentes en toda embarcación de este tipo son:

- Sondas: Son dispositivos de medición utilizados para determinar la profundidad del agua. En el contexto de la pesca son usadas para conocer la cota del lecho marino y poder identificar las zonas más propicias para el ejercicio. También se usan para detectar bancos de peces.
- Radares: Son dispositivos esenciales, sirven de protección contra colisiones con objetos u otros navíos tanto en el fondo marino como en la superficie, por ejemplo en escenarios de baja visibilidad, y también para la detección de bancos de peces. También se pueden usar para obtener información meteorológica.

Asimismo hay otros tipos de dispositivos que si bien se emplean para el mismo propósito, la pesca, se encuentran fuera de la embarcación, es decir en el mar. De entre los distintos dispositivos destacan claramente las radiobalizas.

Las radiobalizas son dispositivos electrónicos, flotantes, que emiten una señal de radio a una frecuencia determinada y que pueden ser detectadas desde grandes distancias. Sirven para indicar la ubicación de objetos o personas en la mar.

Concretamente en la pesca son utilizadas para seguimiento de embarcaciones, marcado de artes de pesca y para seguridad.



Existen varios tipos de radiobalizas dependiendo del uso al que estén destinadas: [6]

- **EPIRB**: son radiobalizas de emergencia utilizadas en situaciones críticas. Se activan manualmente o automáticamente en contacto con el agua y transmiten señales de socorro que incluyen la posición GPS del dispositivo. La mayoría trabajan en los 406 MHz y son monitoreadas por satélite. También hay
- **AIS-SART**: son radiobalizas diseñadas para ser utilizadas en situaciones de búsqueda y rescate. Transmiten señales AIS (Automatic Identification System) que incluyen información sobre la ubicación y la identificación del barco o la persona en peligro. Su frecuencia de trabajo es 121.5 MHz y son monitoreadas por aviones y/o embarcaciones.
- **PLB**: son balizas personales (Personal Locator Beacon) de localización utilizadas por individuos en situaciones de emergencia en el mar. Se activan manualmente y transmiten señales de socorro que incluyen la posición GPS.
- **MOB**: Estos dispositivos son diseñados para ser utilizados en situaciones en las que una persona cae al agua (Man Overboard Beacon). Emiten señales para ayudar en la rápida localización y recuperación de la persona.



Figura 4. Ejemplos de radiobalizas: EPIRB, AIS-SART, PLB, MOB.

## 2.2. Fundamentos

El área principal tratada en este proyecto es la relacionada con el estudio de las ondas electromagnéticas, el espectro radioeléctrico y en definitiva las telecomunicaciones RF. El espectro radioeléctrico comprende desde los 30 Hz a los 300 GHz del espectro electromagnético total. Son las llamadas *ondas de radio* y son ampliamente utilizadas en el sector de las telecomunicaciones y de la ingeniería de sistemas. Se puede clasificar el espectro radioeléctrico según los siguientes rangos de frecuencia: [7][8]

- **Frecuencias bajas:** Las Bajas Frecuencias están comprendidas entre los 10KHz para las *VLF (Very Low Frecuencias)* y los 300KHz para las *LF (Low Frecuencias)*.
- **Frecuencias Medias:** Las Medias Frecuencias o *MF (Medium Frecuencias)* comprenden desde los 300KHz a los 3MHz. Su uso principal es el de Radiodifusión en AM (Amplitud Modulada).
- **Frecuencias Altas:** Las Altas Frecuencias están comprendidas entre las *HF (High Frecuencias)*, 3 a 30MHz y las *UHF (Ultra High Frecuencias)*, 300MHz a 3GHz, pasando por las *VHF (Very High Frecuencias)*, de los 30MHz a los 300MHz.

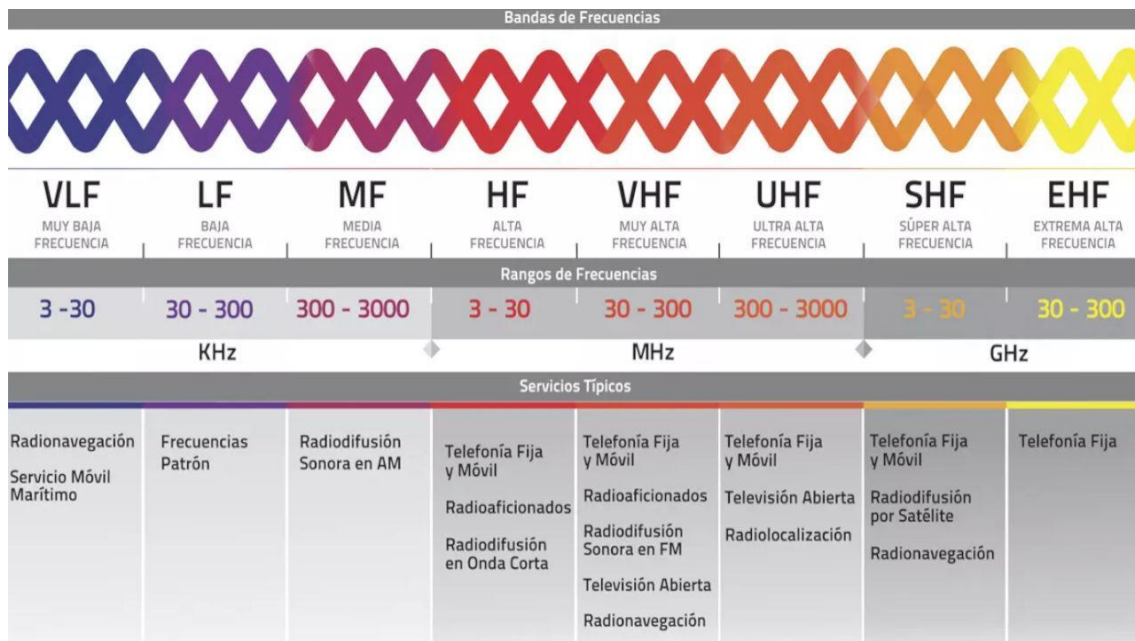


Figura 5. Espectro radioeléctrico. Usos frecuentes.

Para la representación de los datos obtenidos, se calculará la Transformada de Fourier y la Densidad Espectral de Potencia.

### **Transformada de Fourier**

Es una herramienta matemática utilizada para representar señales en el dominio del tiempo convirtiéndolas al dominio de la frecuencia.

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t} dt$$

$\omega$ : Frecuencia Angular (rad/s)

$t$ : Tiempo (s)

### **Densidad Espectral de Potencia**

Una vez calculada la Transformada de Fourier es útil obtener la Densidad Espectral de Potencia (PSD en inglés). La PSD describe cómo se distribuye la potencia de una señal en el dominio de la frecuencia. Proporciona información detallada sobre las características espectrales de una señal.

$$PSD(\omega) = \frac{|F(\omega)|^2}{\Delta f}$$

$F(\omega)$ : Transformada de Fourier

$\omega$ : Frecuencia Angular (rad/s)

$\Delta f$ : Ancho de Banda en la frecuencia  $f$

### 2.3. Antecedentes

Este proyecto comprende el avance del sistema FAMIS, desarrollado por el Dr. Alejandro Pomares Padilla en colaboración con otros investigadores e instituciones.

FAMIS (*Fishing Activity Monitoring Integral System*) fue creado con el fin de poder ofrecer un sistema de monitorización de la actividad pesquera mediante una app Android.

Otros sistemas ya existentes como el popular AIS o VMS, si bien se trata de sistemas robustos, resultan poco eficientes en algunos escenarios como la pesca recreativa en bajura o directamente desde la costa, dificultando el control de estos pequeños, aunque numerosos usuarios.

#### *Using mobile device's sensors to identify fishing activity* <sup>[9]</sup>

FAMIS aprovecha la información generada por los sensores que se encuentran en cualquier dispositivo móvil, tales como smartphones o tablets. Por tanto el pilar fundamental es la portabilidad y el acceso fácil, directo y económico por medio de una App. Si bien hay pequeñas variaciones en cuanto a los sensores que disponen algunos dispositivos móviles, todos tienen en común:

- GPS: Da la posición en los tres ejes cardinales (Ox, Oy, Oz).
- Acelerómetro: Mide el movimiento, en 3D, para saber en qué orientación, horizontal o vertical, está el dispositivo.
- Giroscopio: Amplia los grados de libertad del acelerómetro, añadiendo una cuarta dimensión, la rotación.
- Magnetómetro: Basado en el efecto Hall, normalmente es utilizado a modo de brújula electrónica detectando el polo norte magnético terrestre.

Con este sistema se estudian principalmente las fases de navegación (*shipping*), lance (*towing*), (*setting*) y recogida (*hauling*).

La prueba experimental se llevó a cabo a bordo del buque de 70 metros de eslora y 12 metros de manga *Miguel Oliver*, especializado en la investigación pesquera y oceanográfica.

Para ello se realizaron 22 lances a lo largo de la costa del litoral español (Castellón, Tarragona, Barcelona y Gerona).



Figura 6. B/O Miguel Oliver. Litoral Catalán.

Una vez realizado el estudio de los datos recopilados, se confirmó la viabilidad de este sistema frente a otros en cuestión de certeza, precisión y tiempos de cómputo.



Figura 7. Interfaz de usuario App FAMIS. Flujo de información.

*Is the vessel fishing? Discrimination of fishing activity with low-cost intelligent mobile devices through traditional and heuristic approaches* [\[10\]](#)

En la siguiente etapa del proyecto se implementa una red neuronal para el tratamiento de los datos obtenidos por el sistema FAMIS.

Para ello se utilizan varias técnicas de aproximación: Análisis lineal (*LDA*), Red Neuronal Probabilística (*PNN*), Perceptrón Multicapa (*MLP*) y Sistema de Vector Soporte (*SVM*).

Siendo las métricas a estudiar: Exactitud (*Accuracy*), Precisión (Precision) y Ratio de verdaderos positivos (*Recall*).

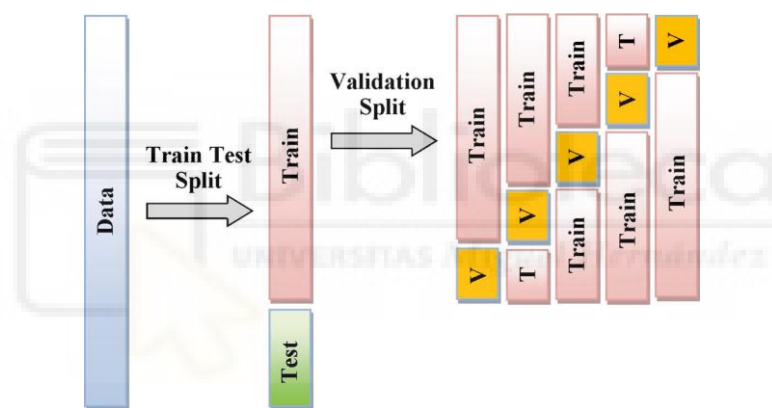


Figura 8. Modelo de Red Neuronal.

Tras realizar el experimento se llega a la conclusión de que el sistema FAMIS es una solución viable para el control de la actividad pesquera ya que arroja resultados significativos en términos de certeza y precisión, así como en tiempos de respuesta.

## 2.4. Propuesta

El núcleo del actual proyecto es el procesamiento de señales en el dominio de la frecuencia, artificialmente generadas, equiparables a las de los dispositivos reales a detectar como sondas, balizas, radares... ya que son típicamente empleados para la detección de bancos de peces (sondas), del fondo marino (radares) o piscifactorías de deriva (balizas) entre otros muchos.

De entre todos, se eligen estos dispositivos ya que se pueden agrupar en rangos de frecuencia bien diferenciados. Mientras que las sondas suelen trabajar en frecuencias del orden de kHz a pocos MHz, las balizas lo hacen en el orden de varios cientos de MHz y los radares en GHz.

Una vez captadas y recopiladas las frecuencias deseadas, usarán como información adicional a la ya existente en el modelo neuronal.

Para la captación de las diferentes frecuencias se hará uso de un receptor de gran banda ancha o *RTL-SDR (Software Defined Radio)* y para el procesamiento de estas, un microcomputador con microprocesador quad-core (4 núcleos), memoria RAM y conectividades inalámbricas WiFi y Bluetooth.

Para la programación de estos dispositivos se usará el lenguaje de Alto Nivel Python debido a que se trata de uno de los lenguajes más usados actualmente para múltiples aplicaciones y que por tanto existen herramientas, información y feedback tanto de usuarios como fabricantes y desarrolladores profesionales.

Se puede decir que Python es hoy en día el lenguaje de programación para aplicaciones IoT y de microcontroladores por antonomasia.

Se trata de un lenguaje puramente interpretado basado en la Programación Orientada a Objetos con una sintaxis sencilla, práctica e intuitiva que si bien es aplicable a cualquier ámbito sirve especialmente de gran ayuda en la programación de dispositivos electrónicos como microcontroladores, sensores, receptores así como el procesamiento de señales con funciones matemáticas equivalentes a las disponibles en otros lenguajes como MATLAB y el tratamiento de datos con técnicas avanzadas de aprendizaje profundo.

Los SDR o (Radio Definida por Software) son dispositivos basados en el chip RTL2832u del fabricante Realtek<sup>1</sup>, y son usados principalmente por radioaficionados.

El RTL2832u inicialmente fue diseñado para actuar como demodulador de señales de Televisión digital, pero gracias a su bajo coste y su capacidad para muestrear una amplia gama de frecuencias, se ha convertido en el núcleo por antonomasia de todo SDR.

Es capaz de captar frecuencias que van desde los 25 MHz hasta los 1.5 GHz, pudiendo ampliarse este rango por medio de un Amplificador de banda.

Para poder hacer uso de él es necesario disponer de controladores y software adecuados y específicos para tal propósito, si bien cada vez más existen librerías multipropósito creadas por desarrolladores particulares para usarse en lenguajes de programación extendidos como Java/JavaScript, MATLAB/Simulink, C/C++/C# o Python. [\[11\]](#)



Figura 9. Placa "dongle" de SDR. Chip RTL2832U.



Para el acceso y uso del receptor RTL-SDR es necesario un dispositivo que procese las muestras de las señales capturadas aplicable a la situación y el contexto tratados en este documento: bajo coste económico, reducido consumo energético, múltiples conectividades, potencia de cómputo y tamaño compacto. Los dispositivos que cumplen con estos requisitos y que se encuentran actualmente en el mercado son: un microcontrolador o un microcomputador. [12]

Los microcontroladores son dispositivos electrónicos que integran un procesador, una memoria y diversos periféricos en un mismo chip. Son muy versátiles y están diseñados para realizar tareas específicas, dependiendo la placa de desarrollo en la que estén implementados. Unas de esas aplicaciones son las llamadas IoT (Internet Of Things).



Figura 10. Arduino UNO, ejemplo de Microcontrolador.

Los microcomputadores por su parte, son dispositivos más complejos pero ofrecen mayor potencia y compatibilidad con otros sistemas. Son computadores a todos los efectos, pero se diferencian en que están configurados en una placa de desarrollo no configurable.



Figura 11. Raspberry Pi, ejemplo de Microcomputador.

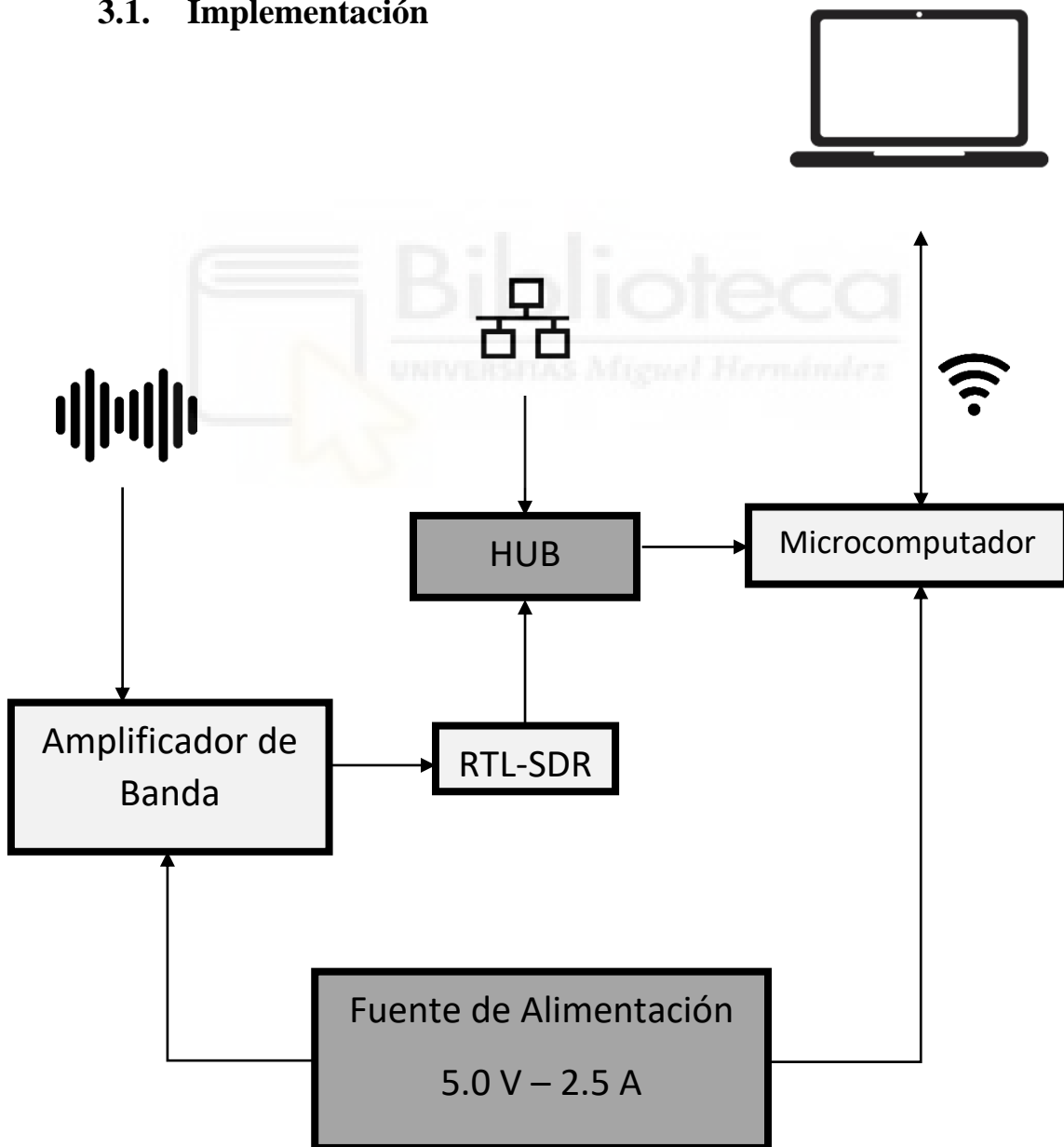


## Capítulo 3. Materiales y Métodos

En este capítulo se describe el material empleado así como los métodos seguidos. Se enumerarán cada uno de los dispositivos y componentes *Hardware* y seguidamente las herramientas *software*.

Se proporcionará información de las configuraciones realizadas y del código desarrollado.

### 3.1. Implementación



El diseño inicial comprendía el uso de microcontroladores para el procesamiento de las frecuencias ya que son la mejor opción para tal propósito debido a su bajo consumo energético, sus prestaciones y sus múltiples conectividades.

En concreto se escogieron los microcontroladores del fabricante ESP: el modelo ESP32-S3-USB-OTG a modo de “maestro” y otros tres microcontroladores del modelo ESP32-S3-DevKitM-1 como “esclavos”. [\[13\]](#)[\[14\]](#)



*Figura 12. ESP32-S3-USB-OTG, ESP32-S3-DevKitM-1.*

Ambos microcontroladores están basados en el mismo SoC (System On Chip), el ESP32-S3.

El primero de ellos, el maestro, fue escogido de la enorme oferta actualmente disponible en el mercado por estar diseñado específicamente para su uso en aplicaciones IoT. Incorpora puertos USB-Dev y Host para su uso como dispositivo USB-OTG (On The Go), asimismo dispone de pantalla LCD para poder visualizar información. Todo esto añadido a las prestaciones que ofrece el propio SoC ESP32-S3 como es su CPU, su memoria flash RAM, conectividades Bluetooth y WiFi y diversas entradas y salidas analógicas y/o digitales.

Durante la configuración de estos dispositivos surgieron diversos problemas.

El primero de ellos fue la carga del *firmware* respectivo de cada uno con la distribución de Python para microcontroladores *micropython*. En el repositorio de *micropython* existen firmwares compatibles con numerosas placas de desarrollo de diferentes modelos y fabricantes, de entre ellas la placa de desarrollo ESP32-S3-DevKitM-1, pero no ocurre lo mismo para el ESP32-S3-USB-OTG. Esto es debido a que se trata de una implementación específica, con pocas referencias en el mercado y ayuda así como feedback de usuarios en internet. El archivo con el firmware MicroPython sigue el formato: [\[15\]](#)

ESP32\_GENERIC\_S3-SPIRAM\_OCT-20231005-v1.21.0.bin

Dónde ESP32\_GENERIC\_S3 es el modelo de microcontrolador, SPIRAM\_OCT el tipo de memoria SPI flash, que en este caso es OCTAL y .bin la extensión de archivo binario.

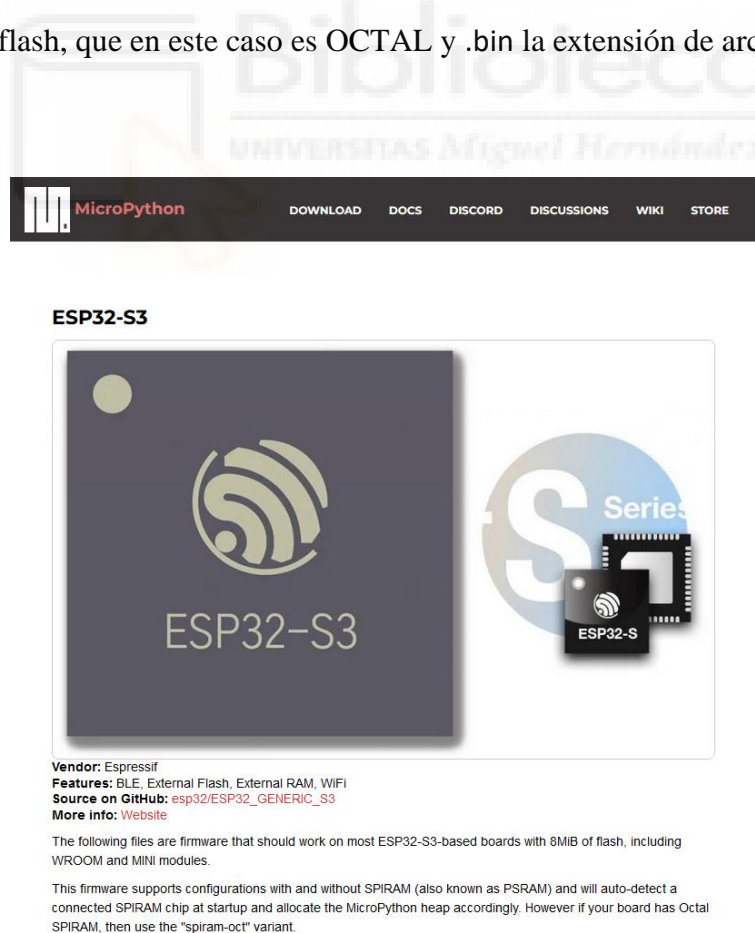


Figura 13. Repositorio MicroPython para descarga firmware ESP32-S3.

Finalmente, tras sucesivas pruebas y carga de diferentes paquetes de firmware proporcionado por el repositorio *Micropython* para ESP32-S3, se decide cargar otra distribución de Python compatible con estos microcontroladores: *CircuitPython*, desarrollada por Adafruit. La instalación resulta sencilla y guiada ya que Circuitpython usa al ESP32-S-USB-OTG como si de una memoria USB externa convencional se tratase. Por lo tanto solo basta copiar/pegar los ficheros necesarios en la carpeta. Asimismo se prueban varios *scripts* sencillos para comprobar el funcionamiento y activación de las diferentes entradas y salidas analógicas/digitales así como otras funcionalidades como es la pantalla LCD del maestro, por ejemplo.

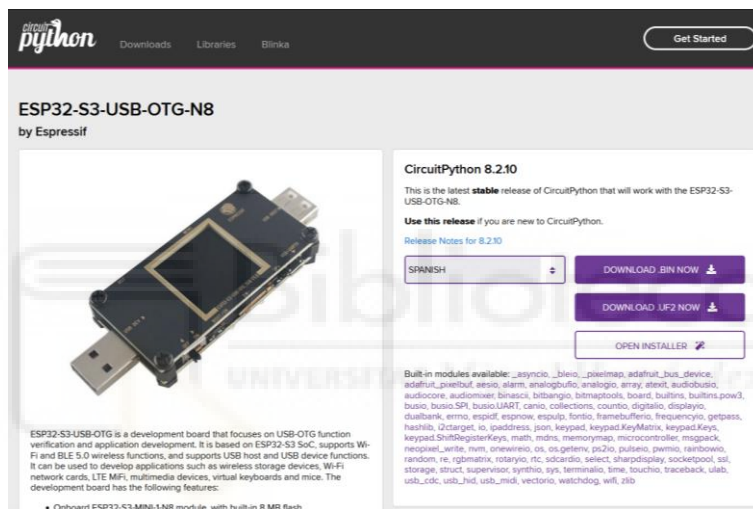


Figura 14. Firmware CircuitPython para ESP32-S3-USB-OTG.

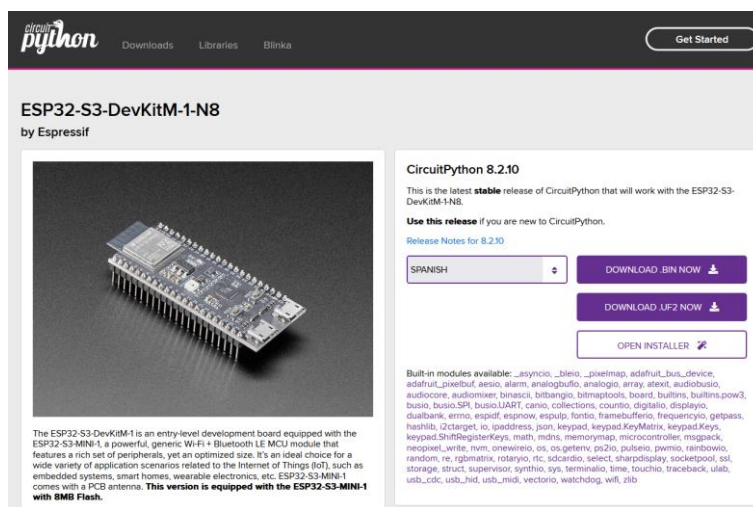


Figura 15. Firmware CircuitPython para ESP32-S3-DevKitM-1.

El primer paso para poder disponer de los microcontroladores con todo el software requerido y poder programar el código deseado, es la instalación del firmware correspondiente y de la librería para el manejo de microcontroladores, *CircuitPython*.

En primer lugar, se procede a la descarga del firmware, disponible en el sitio web oficial de Adafruit en forma de archivo binario *firmware.uf2* (nombre genérico), con la denominación específica: [16][17]

adafruit-circuitpython-espessif\_esp32s3\_usb\_otg\_n8-es-8.2.10.uf2

adafruit-circuitpython-espessif\_esp32s3\_devkitm\_1\_n8-es-8.2.7.uf2

Dónde “espessif” es el fabricante, “esp32s3\_usb\_otg\_n8” y “esp32s3\_devkitm\_1\_n8” los modelos de las placas de desarrollo, “es” el idioma (Español), “8.2.7” la versión del firmware y “.uf2” la extensión binaria.

Seguidamente, se instala este firmware en el microcontrolador. Para ello se decide hacer uso de la herramienta “*flash\_download\_tool*”, proporcionada por el fabricante (*Espressif*).

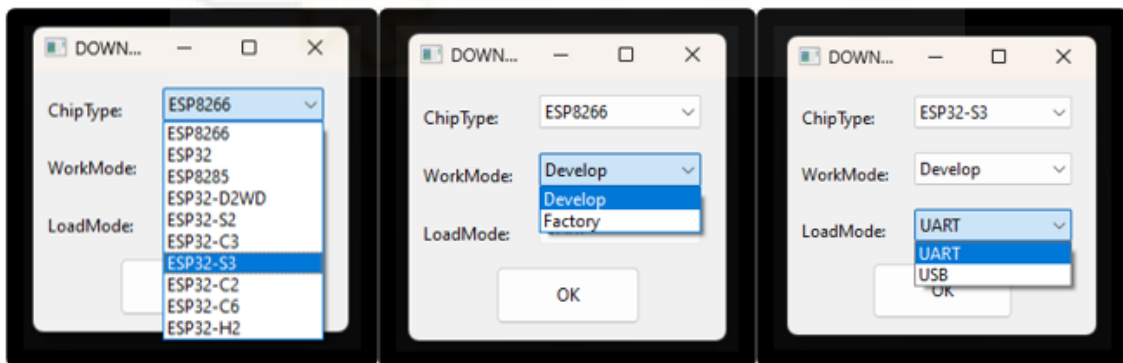


Figura 16. Ventana selección “*flash\_download\_tool*”.

Primero se selecciona el Tipo de Chip (ChipType), que en este caso es *ESP32-S3*. A continuación se selecciona el Modo de Trabajo (WorkMode), se elige el modo *Develop* para poder cargar otros firmwares adicionales si fuera necesario. Por último se escoge el Modo de Carga (LoadMode), se decide usar el puerto UART para ambos modelos de microcontroladores.

El segundo problema, sin solución final, es la instalación de la librería de Python específica para el uso de dispositivos RTL-SDR, *pyrtlsdr*. Si bien esta librería funciona correctamente en entornos de python3, en computadores convencionales con SO Windows/MacOS/Linux, no ocurre lo mismo en entornos de microcontroladores micropython/Circuitpython, imposibilitando por tanto la conexión de estos con el receptor sdr.

La solución a este problema es la creación de una librería propia equivalente a *pyrtlsdr* pero compatible con estos dispositivos, ya que no hay ninguna implementada hasta el momento. Esta solución acaba siendo inviable en cuanto a tiempo y recursos para este Trabajo de Fin de Grado, por lo que se propone como futura mejora e investigación.

Por tanto se encuentra otra solución, temporal: el uso de otro dispositivo de procesamiento, equivalente a un computador convencional en cuanto a compatibilidad de librerías y paquetes y que disponga de características similares a las de un microcontrolador como son el reducido tamaño y consumo energético, las múltiples conectividades y suficiente potencia. Este dispositivo en cuestión es un microcomputador, que por excelencia se trata del modelo Zero 2W del fabricante Raspberry, ya que fue diseñado específicamente para aplicaciones IoT, como es la deseada.

En cuanto al RTL-SDR, se decide elegir el modelo SmartXTR del fabricante norteamericano NOOELEC junto con el kit de ampliación de ancho de banda Ham It Up (HIU). Siendo la mejor opción encontrada en el mercado en cuanto a fiabilidad y robustez, ya que es uno de los principales fabricantes especializados en este tipo de dispositivos. Asimismo este modelo es el que abarca un mayor ancho de banda, gracias al HIU, y siendo el de menor coste económico. Además queda destacar su arquitectura: carcasa de aluminio a modo de cámara de Faraday para evitar interferencias electromagnéticas y ruidos parásitos de fuentes externas cercanas, su forma dongle USB para una conexión sencilla y rápida y finalmente los diversos accesorios como son el conjunto de antenas y conectores coaxiales.

Como sistema de alimentación se escoge una fuente estabilizada a 5.0V/2.5A y potencia de 60 W, con múltiples salidas USB así como una salida USB-C y equipada con pantalla LCD con Voltímetro/Amperímetro/Watímetro.



### 3.1.1. Hardware

A continuación se enumera cada uno de los componentes que constituyen el sistema.

Lista de componentes *Hardware* junto a sus características:

#### **Receptor**

- Nooelec smartXTR sdr-rtl: Receptor RF del fabricante estadounidense Nooelec.
- Ham it Up: Módulo de ampliación de banda de captura de frecuencias.

#### **Microcomputador**

- Raspberry Pi Zero 2W: Microcomputador para aplicaciones IoT del fabricante Raspberry Pi.

#### **Alimentación**

- Fuente de alimentación: Fuente de alimentación de 60W con salida estabilizada USB 5V/2.5A, incluye pequeño LCD con valores de voltímetro y amperímetro.

#### **Almacenamiento**

- MicroSD-card: Tarjeta microSD 256GB de capacidad.

#### **Concentrador de puertos**

- Hub USB-C: Ethernet, USB 2.0, USB 3.0, USB-C, HDMI, Thunderbolt, MiniSDcard, SDcard.

#### **Conexiones**

- Cableado y antenas: Cables MicroUSB-B a USB-A, cable USB-A a USB-B.
- Adaptadores: USB-C a USB-A, USB-A a USB-C, USB-C a MicroUSB-B, MicroUSB-B a USB-C.

La etapa de recepción comienza con el amplificador de banda (*Ham It Up*) conectado al receptor de banda ancha (RTL-SDR), en una configuración estándar de *stick* USB o *dongle* para conexión directa en computadores y dispositivos similares.

El conjunto adquirido cuenta además con una base con varios tipos de antenas y diferentes conectores. [18]



Figura 17. Kit NooElec NESDR smartXTR.

Una vez conectada toda la etapa de recepción, se encuentra la “Raspberry Pi Zero 2W”. Se trata de un microcomputador implementado en placa de desarrollo, el modelo “zero 2W” en concreto está destinado principalmente a aplicaciones IoT.

Se decide emplear este modelo por su bajo consumo, por ser compatible con dispositivos USB-otg como el RTL-SDR así como con lenguajes de programación de alto nivel con las respectivas librerías, por su capacidad de cómputo y por sus reducidas dimensiones.

Este modelo está basado en el microprocesador común de Raspberry Pi. Dispone de un circuito integrado para las conectividades inalámbricas y diversos puertos.

En primer lugar se encuentra el puerto o *slot* para introducir la tarjeta microSD a modo de disco duro que contendrá, entre otros, el Sistema Operativo.

Seguidamente se encuentra un puerto miniHDMI para la conexión de periféricos de salida de imagen (pantallas), un puerto microUSB para alimentar la propia placa (pwr usb) y un segundo puerto microUSB del tipo OTG *On The Go* para la conexión de múltiples periféricos o dispositivos de distinto tipo y funcionamiento, actuando como puerto de entrada/salida (E/S). <sup>[19]</sup>

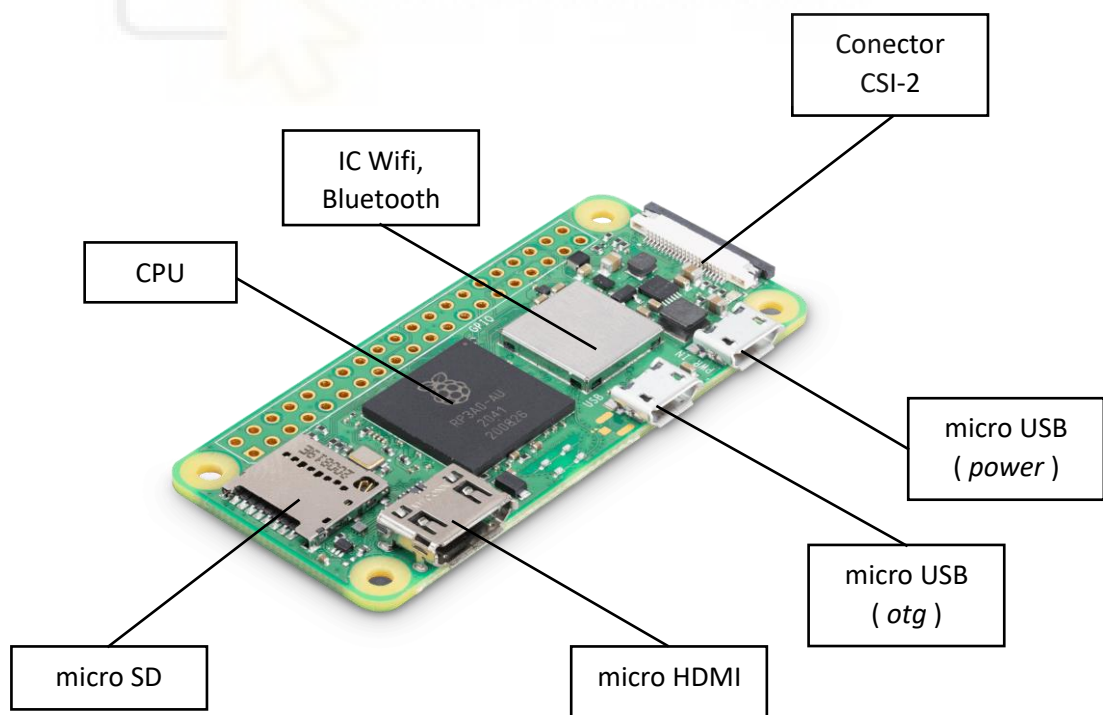


Figura 18. Raspberry Pi Zero 2W.

Para simular las frecuencias que posteriormente serán captadas por el SDR se decide usar dispositivos comerciales de fácil acceso como son: control remoto de apertura/cierre de automóvil para la emisión en MHz y control remoto de coche Radio Control para la emisión en GHz.



*Figura 19. Ejemplo de Control Remoto de automóvil (MHz).*



*Figura 20. Ejemplo de Control Remoto de coche RC (GHz).*

A continuación se presenta el prototipo de laboratorio usado del sistema con todos los componentes principales, sin cablear.

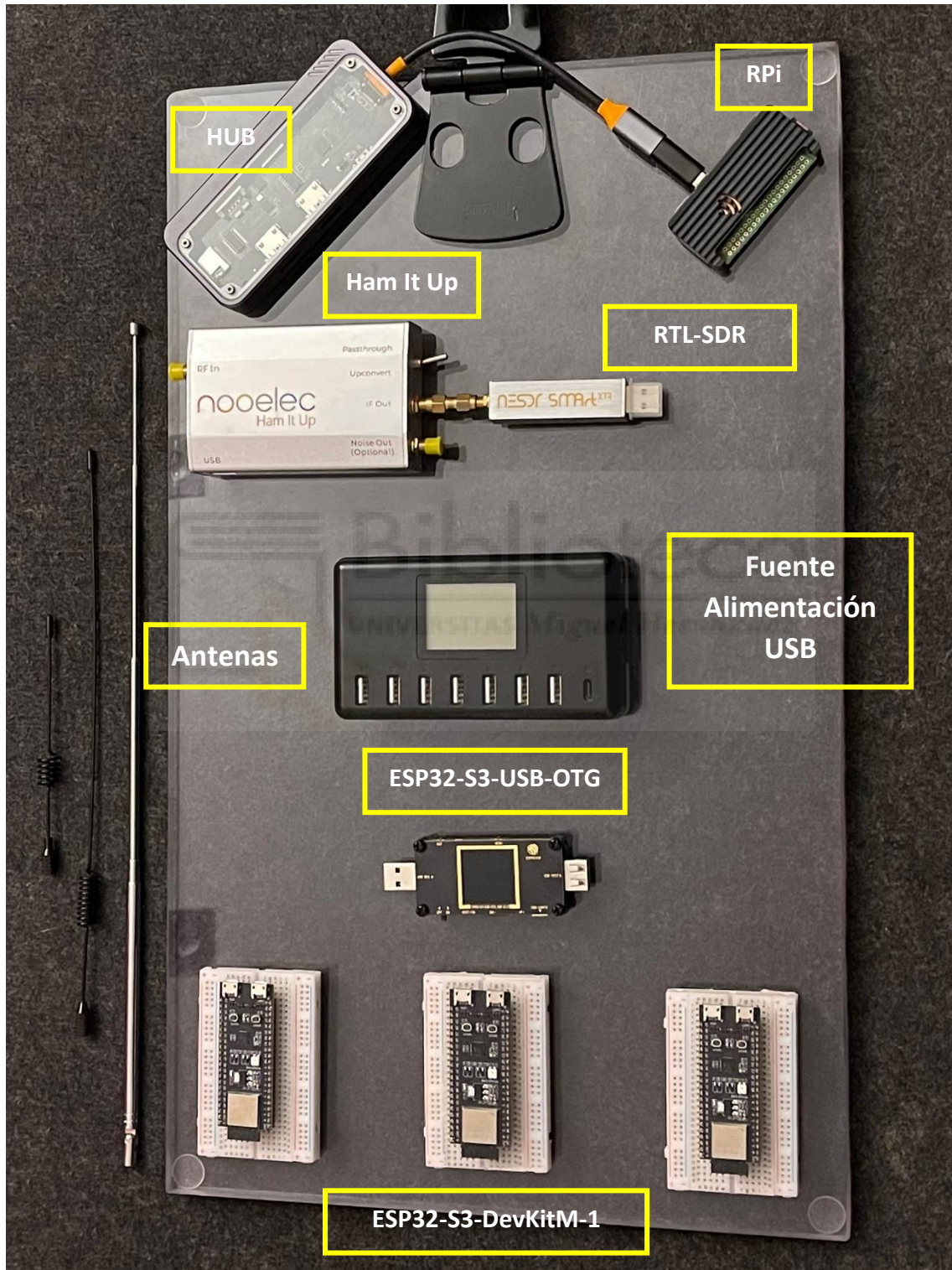


Figura 21. Prototipo de laboratorio.

### 3.1.2. Software

Por practicidad y eficiencia de código se ha decidido usar el lenguaje de programación de alto nivel *Python* con sus respectivas librerías específicas para tal propósito, *pyrtlsdr* para la programación del SDR, *numpy* para el tratamiento de los datos y *matplotlib* para la representación de los resultados.

Lista de librerías y paquetes:

- *Pyrtlsdr*: Librería destinada al uso de receptores RTL-SDR
- *Numpy*: Se usa principalmente para el tratamiento de los datos, como por ejemplo aplicar diferentes métricas estadísticas.
- *Matplotlib*: Sirve para graficar (“plotear”) las variables o funciones deseadas.
- *Os*: Permite usar recursos del S.O.
- *Time*: Obtiene el Tiempo del Sistema.

#### **PyRtlsdr** [\[20\]](#)

La librería de uso libre, *opensource*, *Pyrtlsdr* es una biblioteca de Python que actúa a modo de interfaz entre dispositivos rtl-sdr anteriormente mencionados y sistemas operativos permitiendo la comunicación y control de estos mediante funciones específicas, tales como:

- Recepción de señales de radio
- Configuración de dispositivos receptores
- Procesamiento de señales
- Interfaz de usuario

El grueso del proyecto, en cuanto a programación, se centrará en el uso de las múltiples funciones que ofrece la librería *pyrtlsdr*.

## **Numpy** [\[21\]](#)

Ambas librerías se usan para realizar operaciones matemáticas, pero mientras que *Numpy* es usada para manejo de matrices y funciones básicas, *scipy* por su parte añade más funciones y algoritmos más complejos para trabajar con métodos diferenciales, integrales, transformada de Fourier... entre otros.

## **Matplotlib** [\[22\]](#)

Matplotlib es una librería de Python altamente usada y conocida en campos como la ingeniería de sistemas y estadística y que ofrece funciones equivalentes a Matlab en cuanto a gestión, manejo y visualización de datos numéricos.

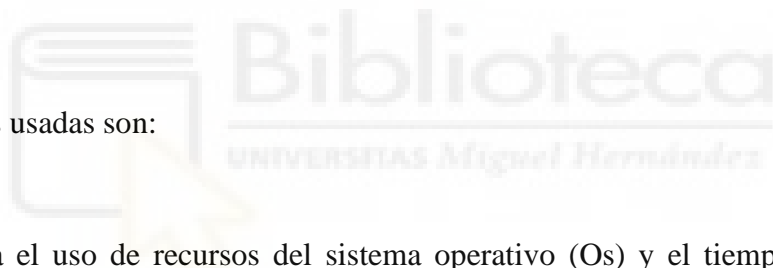
Su uso principal es el de ploteado de diferentes tipos de funciones matemáticas.

Normalmente se usa en paralelo con la librería NumPy que trabaja con datos almacenados en forma matricial.

Otras librerías usadas son:

### **Os / Time**

Librerías para el uso de recursos del sistema operativo (Os) y el tiempo del sistema (Time)



## 3.2. Configuración

Debido a que la Raspberry Pi es un entorno más complejo (Linux) que el firmware de los microcontroladores, es necesario realizar una serie de configuraciones.

El primer paso de la configuración de la microcomputadora es la instalación del sistema operativo *Raspberry Pi OS*.

Para ello el fabricante proporciona una herramienta sencilla y cómoda (*Raspberry Imager v1.8.1*).

Solamente quedará seleccionar el modelo de dispositivo (*Raspberry Pi Zero 2W*), el tipo de sistema operativo a instalar (*Raspberry Pi OS Full 64-bit*) y el almacenamiento (miniSDcard).



Figura 22. Herramienta Raspberry Imager v1.8.1.



El primer paso es seleccionar el dispositivo Raspberry Pi, en este caso se trata del modelo *Raspberry Pi Zero 2W*.

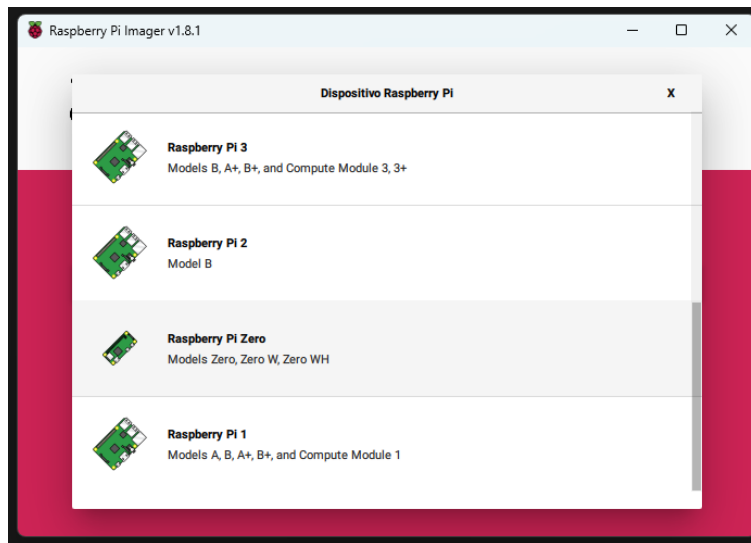


Figura 23. Selección de dispositivo Raspberry.

Seguidamente, se selecciona el SO que se desea instalar, en este caso se decide instalar la versión completa de 64-bits.

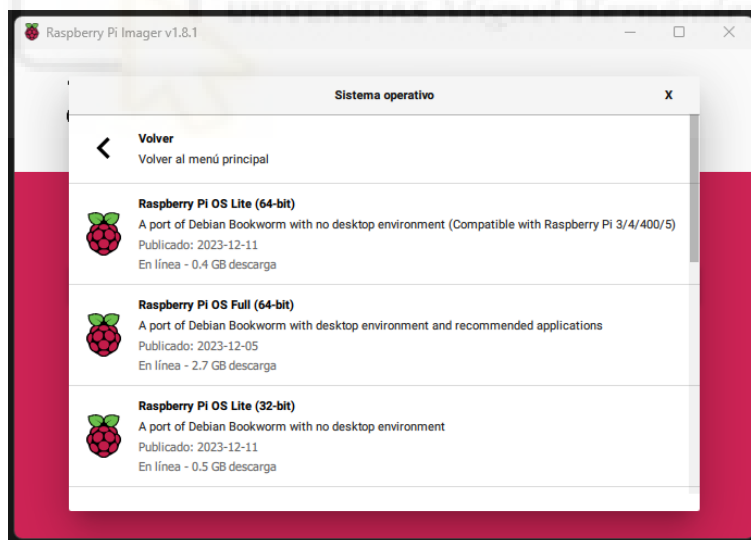


Figura 24. Selección de Sistema Operativo Raspberry.

Asimismo, en ajustes se habilitará la conexión SSH para trabajar con mayor comodidad desde otro PC mediante protocolo SSH.

Tras instalar el S.O. se procede a la conexión remota vía SSH por medio de *PuTTY* para la ventana de comandos y vía *RealVNC Viewer* para el entorno gráfico. Para la configuración de PuTTY se indica la dirección IP, el puerto y el tipo de conexión (SSH).

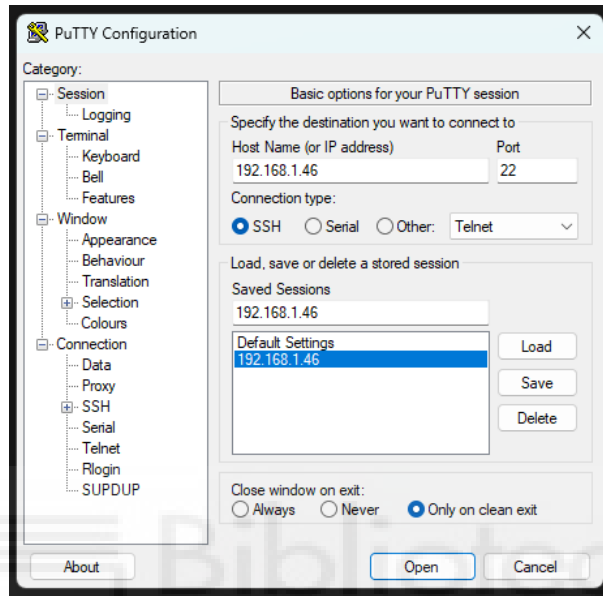


Figura 25. Configuración PuTTY.

Asimismo, se establece la conexión VNC, para ello es necesaria la autenticación mediante el usuario y la contraseña de la R.PI.

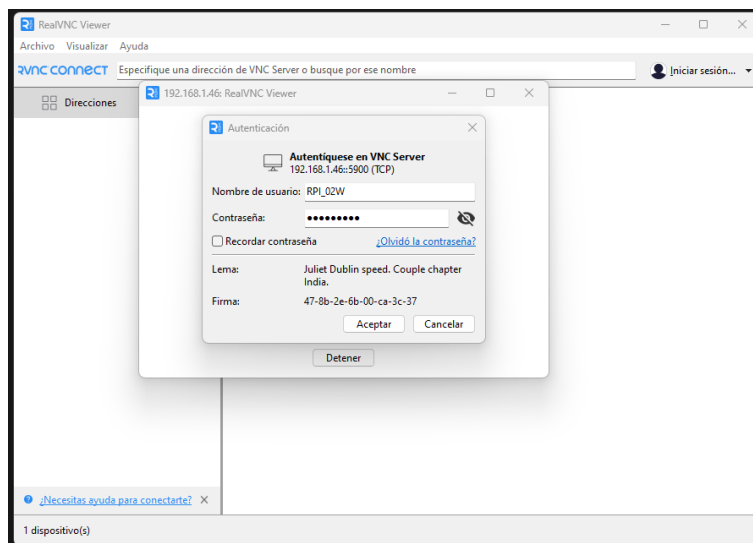


Figura 26. Autenticación RealVNC Viewer.

Una vez configurado *PuTTY* y *RealVNC Viewer* ya se puede acceder a la RPi, tanto por ventana de comandos (PuTTY) como por interfaz gráfica (RealVNC Viewer). Lo primero que aparece es el escritorio de la R.Pi.

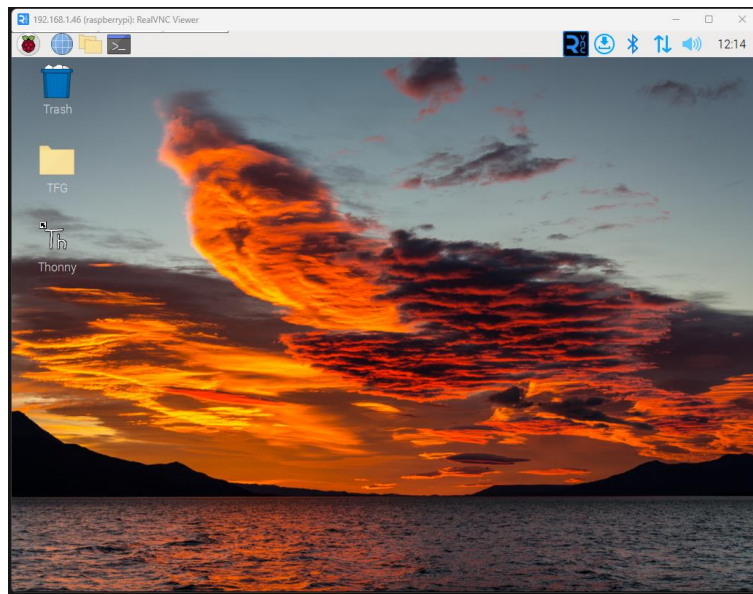


Figura 27. Escritorio Linux (Debian) en RPi.

Una vez dentro de la Raspberry Pi, accederemos al IDE de programación *Thonny*. En la pestaña “Manage packets/Manejar paquetes” buscamos e instalamos las librerías necesarias.

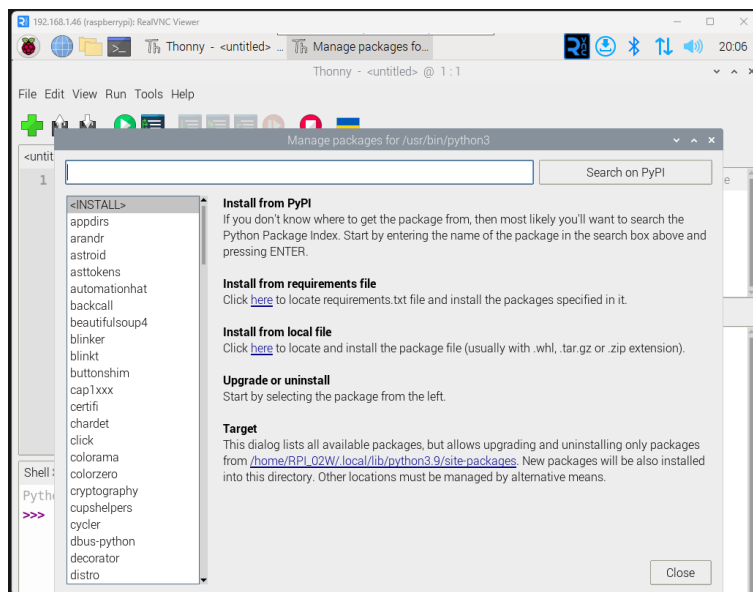


Figura 28. IDE Thonny en RPi. Manage packages.

La librería para el acceso al sdr y sus funcionalidades *pyrtlsdr* no es posible instalarla usando la herramienta *Manage packages*, esto es debido a que está desarrollada para distribuciones de SO Linux comunes para PC (ej. Ubuntu) pero no para la distribución propietaria de R.Pi. (Raspbian), basada en Desbian. Por lo tanto es necesario “construirla” manualmente mediante ventana de comandos. Los comandos para construir la librería son:

Tabla 3. Comandos Linux construcción librería 'librtlsdr'.

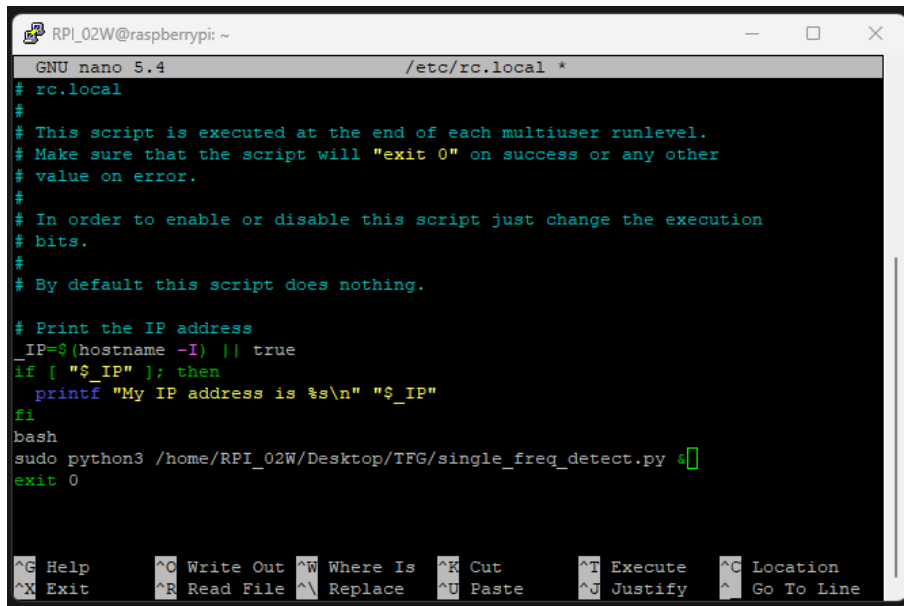
<p><b>1. Instalar dependencias</b>  sudo apt update  sudo apt install git cmake libusb-1.0-0-dev</p> <p><b>2. Clonar el repositorio del controlador RTL-SDR</b>  git clone <a href="https://github.com/osmocom/rtl-sdr">https://github.com/osmocom/rtl-sdr</a></p>	<p><b>3. Compilar e instalar el controlador RTL-SDR</b>  cd rtl-sdr/  mkdir build  cd build  cmake ../ -DINSTALL_UDEV_RULES=ON  make  sudo make install</p> <p><b>4. Cargar la librería:</b>  sudo ldconfig</p>
--	---

Otra de las configuraciones de la R.Pi. es la carga automática del *script* encargado de tomar muestras del receptor de frecuencias. Para ello existe una forma fácil en Linux, siguiendo algunos pasos:

Tabla 4. Comandos Linux carga automática de scripts.

<p><b>1. Copiar ruta script Python:</b>  /home/RPI_02W/Desktop/TFG/single_freq_detect.py</p> <p><b>2. Editar archivo rc.local:</b>  bash  sudo nano /etc/rc.local</p> <p><b>3. Agregar ruta script Python:</b>  bash  sudo python3 /home/RPI_02W/Desktop/TFG/single_freq_detect.py &amp; *</p> <p><b>4. Guardar y salir:</b>  Ctrl+X (Exit) para salir y Y (Yes) para Guardar.</p> <p><b>5. Reiniciar Raspberry Pi:</b>  bash  sudo reboot</p>
--

\*usar *sudo* para obtener permisos especiales en caso de ser necesarios. Se añade & al final para que el programa se ejecute en segundo plano.



```
RPI_02W@raspberrypi: ~
GNU nano 5.4 /etc/rc.local *
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
IP=$(hostname -I) || true
if [ "$IP" ]; then
  printf "My IP address is %s\n" "$IP"
fi
bash
sudo python3 /home/RPI_02W/Desktop/TFG/single_freq_detect.py &
exit 0

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Figura 29. Archivo rc.local.

Otra configuración trata el acceso a visualización del uso de recursos de SO. Si bien esta configuración no es crítica para el funcionamiento del conjunto RPi-Sdr, es de gran utilidad ya que la RPi es un sistema de recursos y potencia limitados. Se utilizan los comandos: *htop* (Monitor recursos sistema) y *top* (Lista procesos ejecución).

Finalmente, también se usan distintos comandos para obtener información del voltaje así como la temperatura y por tanto tener una idea del estado del microcomputador. Con *vcgencmd measure\_volts* se obtiene el voltaje y con *vcgencmd measure\_temp* la temperatura.

### 3.3. Programación

#### 3.3.1. Adquisición de los datos

Para la lectura de muestras capturadas por el Sdr se desarrollan dos *scripts* sencillos. El primero (*Single\_freq\_det*) comprende la lectura de frecuencias dada una frecuencia central. El segundo (*Multi\_freq\_det*) es equivalente al anterior pero dado un rango de frecuencias. Ambos programas sirven como comprobación del funcionamiento del sdr así como la correcta lectura de la frecuencia (o rango de frecuencias) asignada y los pulsos simulados artificialmente.

##### *Single\_freq\_detect*

Se simulará la detección de una BALIZA y de un RADAR que emiten a una frecuencia del orden 400MHz. Para ello se generarán pulsos de manera artificial usando un control remoto de automóvil (434MHz). Asimismo se simulará la detección de un Radar (2.5GHz), emitiendo pulsos con un control remoto de coche RC (2.4GHz).

##### *Multi\_freq\_detect*

Este segundo programa es equivalente al primero, se añade un rango de frecuencias referenciado a una frecuencia central dada. Se estudia la detección de diferentes tipos de radiobalizas (EPIRB, PLB, MOB). Para ambos programas se registran en un fichero de texto 'data.txt' los valores leídos (Amplitud y Fase) siguiendo la estructura:

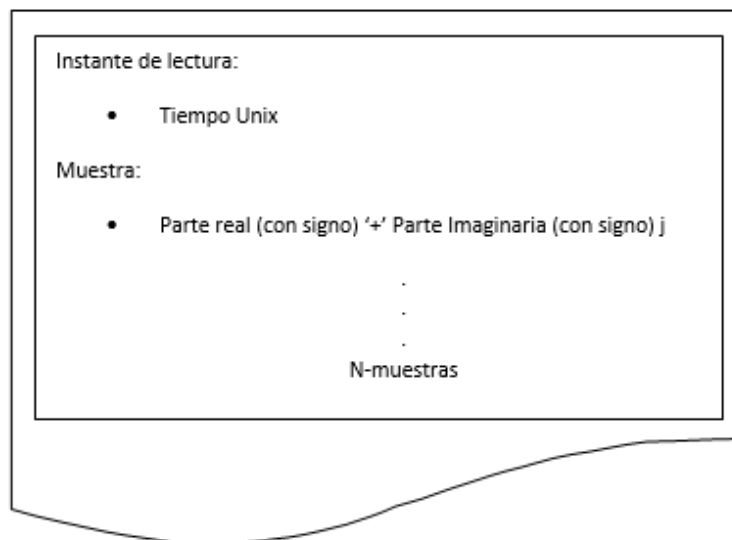


Figura 30. Estructura fichero 'data.txt'.

## *Single\_freq\_detect*

A continuación, se detalla el uso de las diferentes funciones y métodos.

En primer lugar, se declaran las librerías junto a sus respectivos paquetes: *RtlSdr* para la configuración y uso del RTL-SDR, *numpy* (np) para el tratamiento de los datos y *os* y *time* para la gestión de recursos del Sistema Operativo.

```
from rtlsdr import RtlSdr
import numpy as np
import time
```

Seguidamente, se crean las constantes a usar para la fijación de las diferentes frecuencias. Como bien queda explicado en el capítulo anterior, se simularán frecuencias equivalentes a las de trabajo de diferentes dispositivos comunes en embarcaciones de pesca como BALIZA (MHz) o RADAR (GHz):

```
''' Declaracion de constantes '''
BALIZA = 434e6 # MHz
RADAR = 2.1e9 # GHz
```

A continuación, se configura el SDR: se dan los valores deseados a las variables.

```
'''Configuracion del sdr'''
sdr = RtlSdr()
sdr.sample_rate = 2.4e6
sdr.center_freq = 434e6
sdr.gain = 'auto'
sdr.bandwidth = 1e6
```

Se da una tasa de muestreo (*sample\_rate*) de 2.4MHz y se ajusta la frecuencia central a 434 MHz. La ganancia (*gain*) se deja por defecto en 'auto' y se especifica un ancho de banda (*bandwidth*) de 1MHZ.

Para la lectura de las muestras se crea un bucle *for*

```
'''Bucle de lectura y registro de muestras'''
with open('data.txt', 'w') as fich:
    for _ in range(10): # Toma de 10 muestras

        muestras = sdr.read_samples(1) # muestras por segundo

        # Se Convierten las muestras a cadena
        real = ' '.join(map(str, np.real(muestras)))
        imag = ' '.join(map(str, np.imag(muestras)))

        # Añade el carácter '+' solo si el valor imaginario es
positivo
        if np.imag(muestras) >= 0:
            fich.write(str(time.time()) + '\n' + real + '+' + imag +
'j' + '\n')
        else:
            fich.write(str(time.time()) + '\n' + real + imag + 'j' +
'\n')

        print(real, imag)

        time.sleep(1) # Dormir durante un segundo
```

Finalmente se cierra la conexión con el dispositivo, para ello se llama al método *sdr.close()*.

```
'''Cierre dispositivo sdr'''
sdr.close()
```



## *Multi\_freq\_detect*

Este segundo script se programa para la detección 'Baliza', se crean varias constantes dependiendo de la frecuencia a la que emite cada tipo de baliza, si bien se prueba solo las del tipo EPIRB ya que es la frecuencia a la que se pueden simular los pulsos artificialmente (406 MHz).

```
''' Declaracion de constantes '''  
EPIRB=406e6  
PLB=162e6
```

Se configura el dispositivo sdr:

```
'''Configuracion del sdr'''  
sdr = RtlSdr()  
sdr.sample_rate = 2.4e6  
sdr.gain = 'auto'
```

Para recorrer el rango se usa la función *np.linspace()* de la librería numpy. Esta función divide un rango especificado en tantas divisiones como se desee. Para ello se dan como argumentos la frecuencia inicial, la frecuencia final y el número de frecuencias intermedias (divisiones) deseado. Se ajusta el número de divisiones (11) de tal forma que se incrementa la frecuencia cada 1 MHz.

```
rango_frecs= np.linspace(400e6, 410e6, num=11)
```

Se crean dos bucles anidados, el primero recorre el rango de frecuencias y el segundo captura las muestras a razón de 10 muestras por cada frecuencia y a una muestra por segundo:

```
'''Bucle de lectura y registro de muestras'''
with open('data.txt', 'w') as fich:

    # Bucle para recorrer el rango de frecuencias
    for frec in rango_frecs:
        # Configurar el SDR a la frecuencia actual
        sdr.center_freq= frec

        # Bucle de lectura de muestras
        for _ in range(10): # muestras por frecuencia
            muestras = sdr.read_samples(1) # muestras por segundo

            # Se Convierten las muestras a cadena
            real = ' '.join(map(str, np.real(muestras)))
            imag = ' '.join(map(str, np.imag(muestras)))

            # Añade el carácter '+' solo si el valor imaginario es
positivo
            if np.imag(muestras) >= 0:
                fich.write(str(time.time()) + '\n' + real + '+' + imag
+ 'j' + '\n')
            else:
                fich.write(str(time.time()) + '\n' + real + imag + 'j'
+ '\n')

            print(real, imag)

            time.sleep(1) # Dormir durante un segundo
```

### 3.3.2. Representación de los datos

#### *Sdr\_data\_graphics*

Una vez recopilada la información 'data.txt', se crea un programa para el análisis de los datos. Con él se calcula, registra y representa la Señal Original captada, la Transformada de Fourier y la Densidad Espectral de Potencia, de dicha señal.

A continuación se hace una breve descripción de las funciones creadas:

#### **def SDR\_CONFIG()**

La primera función *def SDR\_CONFIG()* se encarga de configurar el dispositivo SDR.

#### **def SDR\_READ\_DATA()**

Seguidamente se encuentra la función *def SDR\_READ\_DATA()*. Es el núcleo del programa, ya que es la encargada de la lectura de frecuencias del rtl-sdr.

#### **def SDR\_CLOSE()**

Como su nombre indica, cierra la conexión con el RTL-SDR.

#### **def SDR\_DATA\_FFT()**

Se encarga de calcular la Transformada de Fourier (FFT).

#### **def SDR\_DATA\_PSD()**

Calcula la Densidad Espectral de Potencia (PSD).

#### **def SDR\_PLOT\_DATA()**

Representa en diferentes gráficas las señales obtenidas: Señal Original, FFT(señal original) y PSD(señal original). Asimismo representa el valor absoluto de dichas señales para tener una mejor comprensión.

El primer paso comprende la declaración de librerías. Las librerías que se usan son las mismas que en los otros *scripts*: *pyrtlsdr*, *numpy* a excepción de *matplotlib*, que es la que se usará para representar las gráficas de las funciones obtenidas.

```
from rtlsdr import RtlSdr
import numpy as np
import time
import matplotlib.pyplot as plt
```

La primera función trata la configuración del sdr, dando los valores a las variables requeridas: objeto sdr (*RtlSdr*), tasa de muestreo (*sample\_rate*), frecuencia central (*center\_freq*), ganancia (*gain*) y ancho de banda (*bandwidth*)

```
'''Configuracion del dispositivo SDR'''
def SDR_CONFIG(tasa_muestreo, freq_cent, gain, ancho_banda):
    sdr = RtlSdr() #Declaracion del objeto RtlSdr

    #Variables de configuracion
    sdr.sample_rate = tasa_muestreo
    sdr.center_freq = freq_cent
    sdr.gain = gain
    sdr.bandwidth = ancho_banda

    return sdr
```

La siguiente función es la que realiza la lectura de las muestras y el registro de las mismas en el correspondiente fichero de texto.

```
'''Lectura de frecuencias'''
def SDR_READ_DATA(sdr, num_muestras, nombrefichero):
    muestras = sdr.read_samples(num_muestras) #Captura de muestras

    with open(nombrefichero, 'w') as fich: #Registro de las muestras
en un fichero de texto
        for muestra in muestras:
fich.write(f"{str(time.process_time())}\n{np.abs(muestra)}, {np.angle(m
uestra)}\n") # Abs: amplitud, Angle: Fase

    return muestras
```

Se cierra la conexión con el SDR:

```
'''Cierre del dispositivo Sdr'''  
  
def SDR_CLOSE(sdr):  
    sdr.close()
```

Una vez capturadas las muestras se procede al cálculo de las funciones para pasar del dominio del tiempo al dominio de la frecuencia.

La primera de estas funciones calcula la Transformada Rápida de Fourier (Fast Fourier Transform, en inglés). Para este cálculo se usan las funciones de la librería numpy: *np.fft.fft* para el cálculo de la magnitud de la transformada y *np.fft.fftfreq* para el cálculo de las frecuencias de la transformada. Finalmente se hace un bucle para guardar estos valores de Magnitud y Frecuencia en un fichero de texto.

```
'''Calculo de la Transformada de Fourier'''  
  
def SDR_DATA_FFT(muestras, num_muestras, nombrefichero):  
    fourier = np.fft.fft(muestras) #Calculo de la Transformada de  
    Fourier  
    frecs_fourier = np.fft.fftfreq(num_muestras, 1/2.4e6)  
    with open(nombrefichero, 'w') as file:  
        for magnitud, frecuencia in zip(np.abs(fourier),  
    frecs_fourier):  
file.write(f"{str(time.process_time())}\n{magnitud},{frecuencia}\n")  
  
    return fourier, frecs_fourier
```

El siguiente cálculo es el de la Densidad Espectral de Potencia, para ello se sigue la fórmula teórica basada en la FFT. Asimismo se usa la función *np.linspace* para filtrar aquellas frecuencias dentro del rango deseado (rango).

```
'''Calculo de la Densidad Espectral de Potencia'''  
  
def SDR_DATA_PSD(muestras, num_muestras, rango, nombrefichero):  
    # Obtener frecuencias reales  
  
    frecs_reales = np.linspace(95e6, 95e6 + 2.4e6, num_muestras,  
endpoint=False)  
  
    # Filtrar valores dentro del rango de frecuencias  
    filtro = (frecs_reales >= rango[0]) & (frecs_reales <= rango[1])  
    muestras_filtradas = muestras[filtro]  
  
    # Calcular PSD utilizando NumPy  
    psd = np.fft.fft(muestras_filtradas) ** 2 / num_muestras  
  
    # Calcular las frecuencias correspondientes  
    frecs_psd = frecs_reales[filtro]  
  
    # Guardar potencia relativa y frecuencia en un archivo de texto  
    with open(nombrefichero, 'w') as fich:  
        for potencia, frecuencia in zip(psd, frecs_psd):  
            fich.write(f"{potencia},{frecuencia}\n")  
  
    return psd, frecs_psd
```

La función devuelve los objetos *psd* y *freces\_psd*, el primero contiene los valores de potencia en Decibelios y el segundo las frecuencias correspondientes a esos valores en Hercios.

Finalmente se hace un plotado de los resultados:

```
'''Representacion de las señales'''  
  
def SDR_PLOT_DATA(muestras, fourier, frecs_fourier, psd, frecs_psd):  
    # Grafica de la Señal Original  
    plt.figure()  
    plt.plot(muestras, color='grey')  
    plt.title('Señal Original')  
    plt.xlabel('Fase (°)')  
    plt.ylabel('Amplitud')  
    plt.grid(True) # Activar la rejilla  
    plt.savefig('señal_original.png') # Guardar la gráfica como  
imagen  
    # Grafica del Valor Absoluto de la Señal Original  
    plt.figure()  
    plt.plot(np.abs(muestras), color='gold')  
    plt.title('Valor Absoluto Señal Original')  
    plt.xlabel('Fase (°)')  
    plt.ylabel('Amplitud')  
    plt.grid(True)  
    plt.savefig('abs_señal_original.png')  
  
    # Grafica de la Transformada de Fourier (FFT)  
    plt.figure()  
    plt.plot(frecs_fourier, fourier, color='grey')  
    plt.title('Transformada de Fourier (FFT)')  
    plt.xlabel('Magnitud (dB)')  
    plt.ylabel('Frecuencia (MHz)')  
    plt.grid(True)  
    plt.savefig('fourier.png')  
    # Grafica del Valor Absoluto de la FFT  
    plt.figure()  
    plt.plot(frecs_fourier, np.abs(fourier), color='gold')  
    plt.title('Valor Absoluto FFT')  
    plt.xlabel('Magnitud (dB)')  
    plt.ylabel('Frecuencia (MHz)')  
    plt.grid(True)  
    plt.savefig('abs_fourier.png')  
  
    # Grafica de la Densidad Espectral de Potencia (PSD)  
    plt.figure()  
    plt.plot(frecs_psd, psd, color='grey')  
    plt.title('Densidad Espectral de Potencia (PSD)')  
    plt.xlabel('Frecuencia (Hz)')  
    plt.ylabel('PSD (dB/Hz)')  
    plt.grid(True)  
    plt.savefig('potencia.png')  
  
    # Grafica del Valor Absoluto de la PSD  
    plt.figure()  
    plt.plot(frecs_psd, np.abs(psd), color='gold')  
    plt.title('Valor Absoluto PSD')  
    plt.xlabel('Frecuencia (Hz)')  
    plt.ylabel('PSD (dB/Hz)')  
    plt.grid(True)  
    plt.savefig('abs_potencia.png')  
  
    plt.show()
```

Para ello se usan las funciones del conjunto *matplotlib.pyplot* de la librería *matplotlib*. Los métodos principales son los de *plt.figure* para crear la ventana, *plt.plot* para representar la función y *plt.show* para mostrar la gráfica. El resto de métodos se usan para dar forma y ajustar los valores de los ejes de la gráfica. Para las etiquetas se usan: *plt.title* para el título de la gráfica y *plt.xlabel* y *plt.ylabel* para indicar la variable mostrada en el eje X y el eje Y, respectivamente. Por último, el método *plt.grid* muestra la rejilla de la gráfica cuando se le da el argumento 'True'.

Para guardar las gráficas se usa la función *plt.savefig*.

Por cada función a representar (Señal Original, Transformada de Fourier, Densidad Espectral de Potencia) se crean dos gráficas, en una se muestran los valores generados y en la segunda los resultantes tras aplicar el valor absoluto.

También se crea un sencillo *main* para probar el programa, llamando a cada una de las funciones y dándole los valores necesarios a las variables: tasa de muestreo (valor por defecto), frecuencia central (100 MHz), ancho de banda (1MHz).

```
def main():
    tasa_muestreo = 2.4e6
    frec_centro = 100e6
    gain = 'auto'
    ancho_banda = 1e6

    num_muestras = int(tasa_muestreo)
    rango = (95e6, 105e6)

    # Configurar SDR
    mySDR = SDR_CONFIG(tasa_muestreo, frec_centro, gain, ancho_banda)

    # Capturar muestras y guardar en archivo
    myDATA = SDR_READ_DATA(mySDR, num_muestras, "muestras.txt")

    # Calcular FFT y guardar en archivo
    myFFT, myFRECS_FFT = SDR_DATA_FFT(myDATA, num_muestras, "fft.txt")

    # Calcular PSD y guardar en archivo
    myPSD, myFRECS_PSD = SDR_DATA_PSD(myDATA, num_muestras, rango,
    "psd.txt")

    # Graficar resultados
    SDR_PLOT_DATA(myDATA, myFFT, myFRECS_FFT, myPSD, myFRECS_PSD)

if __name__ == "__main__":
    main()
```





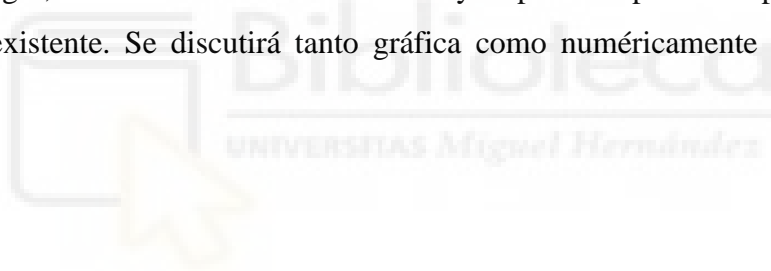
## Capítulo 4. Resultados y Discusión

En este capítulo se exponen y discuten los resultados obtenidos tras aplicar los métodos descritos en el capítulo 3.

Los resultados se dividirán en 2 grandes bloques:

En primer lugar, se evaluará la aplicación de los dispositivos elegidos para la recepción así como del procesamiento de las frecuencias siguiendo los criterios de compatibilidad, potencia, eficiencia, consumo, robustez y otros parámetros. Se discutirá la aplicación de los dispositivos elegidos frente a otros también testeados.

En segundo lugar, se tratarán los datos obtenidos y su posible aportación para el modelo neuronal ya existente. Se discutirá tanto gráfica como numéricamente los resultados obtenidos.



## 4.1. El Sistema

### Compatibilidad

En lo que respecta a compatibilidad del receptor SDR con microcontroladores programados en Python, los resultados son claros. La librería ya desarrollada pyrtlsdr no es compatible con microcontroladores, es exclusiva para sistemas con Python estándar (python3). Por lo tanto es necesario emplear un computador convencional o un equivalente, en el caso de este proyecto el microcomputador Raspberry Pi, que si bien dispone de una potencia más limitada que un PC al uso, soluciona este problema.

### Potencia

En cuanto a potencia, es cierto que el receptor RTL-SDR puede demandar una potencia significativa, si bien en este caso concreto, al muestrear con tiempos poco sensibles se reduce notablemente esta demanda, pudiendo usar dispositivos más económicos y con tecnologías más limitadas.

### Consumo energético

Si bien los microcontroladores son los dispositivos por excelencia para aplicaciones IoT como la tratada, debido a su notoriamente reducido consumo. La Raspberry Pi, en concreto el modelo Zero 2W, siendo un dispositivo más complejo en términos tecnológicos y de potencia, no supone un consumo demasiado elevado ya que fue diseñada para este tipo de aplicaciones.

### Uso de recursos S.O.

Asimismo la Raspberry Pi resulta más ventajosa en materia de consumo de recursos del S.O. pudiendo realizar tareas más demandantes en comparación con los microcontroladores.

Para evaluar el consumo de los diferentes dispositivos involucrados, se estiman 3 usos:

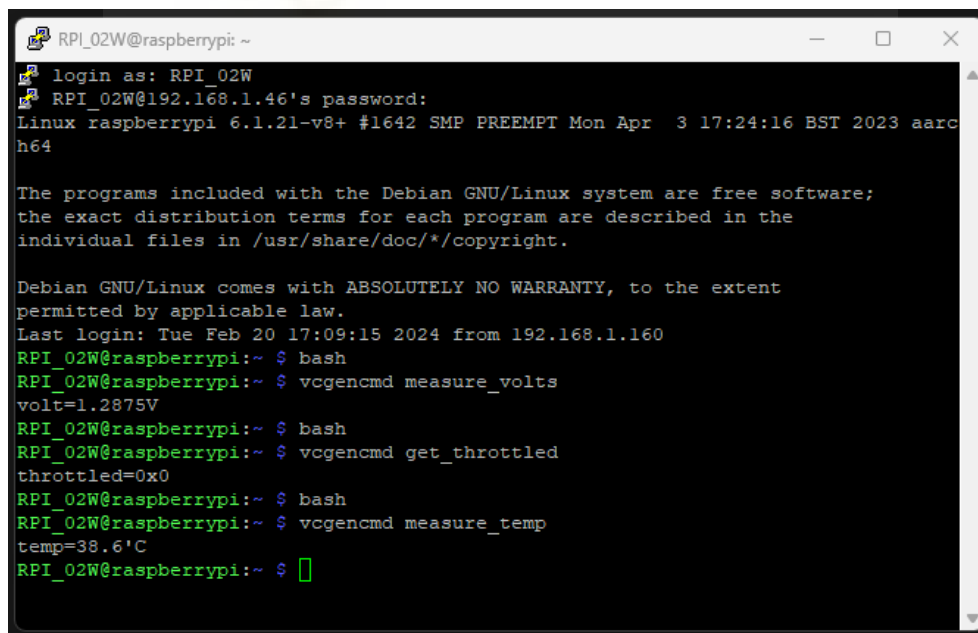
1. Alimentación
2. Detección de frecuencias simples (*Single\_freq\_detect*)
3. Detección de frecuencias múltiples (*Multi\_freq\_detect*)

Para los tres usos el consumo se mantiene constante en los siguientes valores:

Tabla 5. Consumos energéticos.

DISPOSITIVO	TENSIÓN (V)	CORRIENTE (A)	POTENCIA (W)
Raspberry Pi	5.0	0.5	2.5
Ham It Up	5.0	0.2	1.0
SDR	5.0	0.1	0.5

Además se obtienen medidas de voltaje. La temperatura no alcanza los 40°C, lejos de los 85°C de temperatura máxima de funcionamiento recomendada por el fabricante.



```
RPI_02W@raspberrypi: ~  
login as: RPI_02W  
RPI_02W@192.168.1.46's password:  
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Feb 20 17:09:15 2024 from 192.168.1.160  
RPI_02W@raspberrypi:~ $ bash  
RPI_02W@raspberrypi:~ $ vcgencmd measure_volts  
volt=1.2875V  
RPI_02W@raspberrypi:~ $ bash  
RPI_02W@raspberrypi:~ $ vcgencmd get_throttled  
throttled=0x0  
RPI_02W@raspberrypi:~ $ bash  
RPI_02W@raspberrypi:~ $ vcgencmd measure_temp  
temp=38.6'C  
RPI_02W@raspberrypi:~ $
```

Figura 31. Medidas de Voltaje y Temperatura RPi.

En cuanto al uso de recursos del S.O. si bien la RPi es limitada en CPU y memoria RAM, en comparación con un computador convencional, es capaz de ejecutar el script y otras tareas de forma satisfactoria.

```

RPI_02W@raspberrypi: ~
0[          0.0%]  Tasks: 72, 81 thr; 1 running
1[          1.9%]  Load average: 0.09 0.37 0.22
2[|         3.9%]  Uptime: 00:05:20
3[|         1.3%]
Mem[|||||||||||||||||||||173M/419M]
Swp[|||||||||||||||||||||87.1M/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
  ---  ---      ---  --  ---    ---    ---  -  ---  ---   ---+   ---
1680 RPI_02W    20   0 10392  3652  2832  R   2.6  0.9  0:00.62 htop
   1 root       20   0   164M  7372  5240  S   0.0  1.7  0:07.06 /sbin/init spla
  141 root       20   0 49108 11584 11092  S   0.0  2.7  0:01.49 /lib/systemd/sy
  173 root       20   0 22124  2180  2132  S   0.0  0.5  0:01.78 /lib/systemd/sy
  276 systemd-t  20   0 88100  3596  3380  S   0.0  0.8  0:00.61 /lib/systemd/sy
  335 systemd-t  20   0 88100  3596  3380  S   0.0  0.8  0:00.02 /lib/systemd/sy
  336 avahi      20   0  7068  2648  2452  S   0.0  0.6  0:00.37 avahi-daemon: r
  338 root       20   0  9224  2136  2132  S   0.0  0.5  0:00.02 /usr/sbin/cron
  339 messagebu 20   0  8568  3684  2816  S   0.0  0.9  0:01.86 /usr/bin/dbus-d
  349 avahi      20   0  6888   52    0  S   0.0  0.0  0:00.00 avahi-daemon: c
  350 root       20   0 233M  8084  7400  S   0.0  1.9  0:00.50 /usr/libexec/po
  360 root       20   0 215M  2220  2068  S   0.0  0.5  0:00.24 /usr/sbin/rsysl

F1 Help  F2 Setup  F3 Search  F4 Filter  F5 Tree  F6 SortBy  F7 Nice -  F8 Nice +  F9 Kill  F10 Quit

```

Figura 32. Monitor recursos del sistema (htop).

```

RPI_02W@raspberrypi: ~
top - 17:08:38 up 6 min, 4 users, load average: 0,16, 0,32, 0,22
Tasks: 180 total, 1 running, 179 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,5 us, 1,1 sy, 0,0 ni, 98,4 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 419,3 total, 104,0 free, 173,5 used, 141,9 buff/cache
MiB Swap: 100,0 total, 12,9 free, 87,1 used, 179,3 avail Mem

  PID USER      PR  NI  VIRT   RES   SHR  S  %CPU  %MEM   TIME+  COMMAND
  ---  ---      --  --  ---    ---    ---  -  ---  ---   ---+   ---
1680 RPI_02W    20   0 10392  3652  2832  S   3,6  0,9  0:03.57 htop
1724 RPI_02W    20   0 12504  3064  2588  R   0,7  0,7  0:00.64 top
   59 root       20   0    0    0    0  I   0,3  0,0  0:00.34 kworker+
  569 root       20   0 1158592 27844 15672  S   0,3  6,5  0:04.82 Xorg
  620 root       20   0  16560  5468  5080  S   0,3  1,3  0:01.83 vncagent
  928 RPI_02W    20   0 1426344 35824 24260  S   0,3  8,3  0:04.31 lxpanel
1709 RPI_02W    20   0  16056  4752  3540  S   0,3  1,1  0:00.03 sshd
   1 root       20   0 167988  7372  5240  S   0,0  1,7  0:07.08 systemd
   2 root       20   0    0    0    0  S   0,0  0,0  0:00.03 kthreadd
   3 root       0 -20    0    0    0  I   0,0  0,0  0:00.00 rcu_gp
   4 root       0 -20    0    0    0  I   0,0  0,0  0:00.00 rcu_par+
   5 root       0 -20    0    0    0  I   0,0  0,0  0:00.00 slub_fl+
   6 root       0 -20    0    0    0  I   0,0  0,0  0:00.00 netns
   9 root       20   0    0    0    0  I   0,0  0,0  0:02.16 kworker+
  10 root       0 -20    0    0    0  I   0,0  0,0  0:00.00 mm_perc+
  11 root       20   0    0    0    0  I   0,0  0,0  0:00.00 rcu_tas+
  12 root       20   0    0    0    0  I   0,0  0,0  0:00.00 rcu_tas+

```

Figura 33. Lista de procesos en ejecución del sistema (top).

Aunque no resulta crítico, se obtiene el uso de memoria y disco duro.

```
RPI_02W@raspberrypi: ~  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Feb 20 17:07:42 2024 from 192.168.1.160  
RPI_02W@raspberrypi:~ $ bash  
RPI_02W@raspberrypi:~ $ free -m  
              total          used         free      shared  buff/cache   available  
Mem:           419           179           98          6          141          173  
Swap:          99            87            12  
RPI_02W@raspberrypi:~ $ bash  
RPI_02W@raspberrypi:~ $ df -h  
Filesystem      Size  Used Avail Use% Mounted on  
/dev/root       229G  11G  209G   5% /  
devtmpfs        80M   0    80M   0% /dev  
tmpfs           210M   0   210M   0% /dev/shm  
tmpfs           84M   1,1M  83M   2% /run  
tmpfs           5,0M   4,0K  5,0M   1% /run/lock  
/dev/mmcblk0p1 255M   31M  225M  13% /boot  
tmpfs           42M   24K   42M   1% /run/user/1000  
RPI_02W@raspberrypi:~ $
```

Figura 34. Uso de memoria y disco duro.



## 4.2. Los Datos

Para evaluar los datos obtenidos, puesto que se trata de una etapa inicial se tendrá en cuenta únicamente la correcta recepción de las frecuencias deseadas mediante el estudio a nivel cuantitativo de los datos y a nivel cualitativo de los gráficos obtenidos, comparándolos con los generados por otras herramientas software.

Los valores generados por las funciones de la librería *pyrtlsdr* al captar frecuencias con el SDR dan como resultado números complejos donde la parte real corresponde a la Amplitud y la parte imaginaria a la Fase.

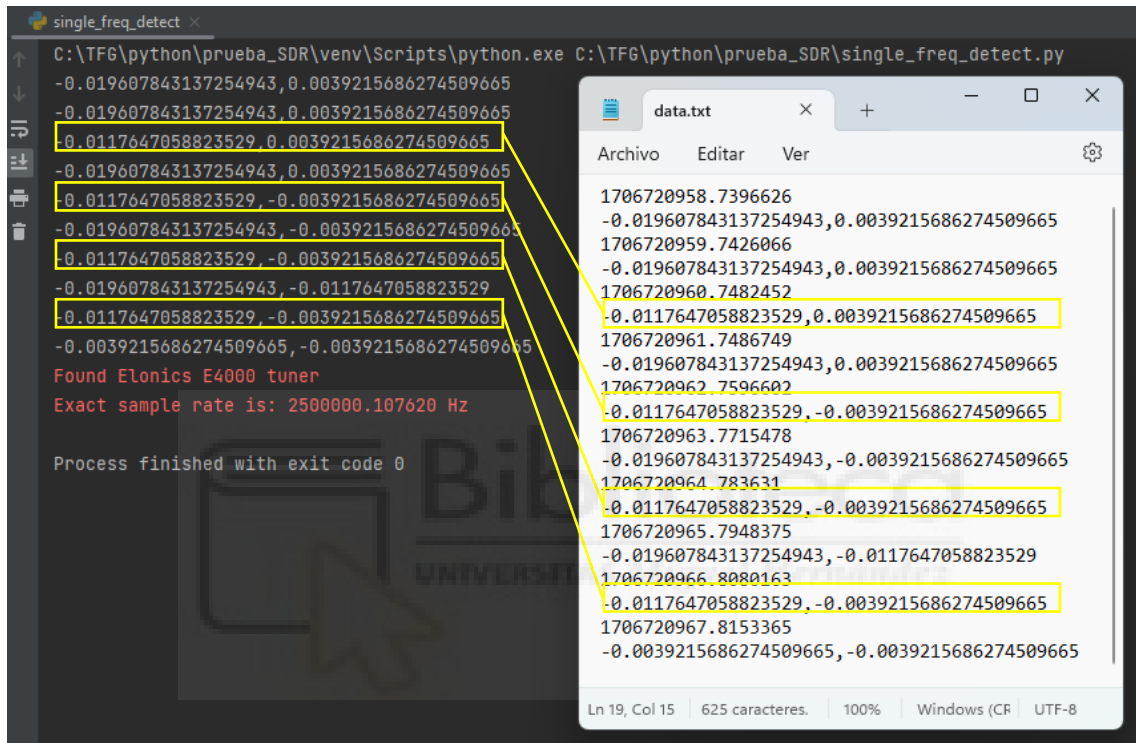
Si bien estos valores son registrados en el fichero 'data.txt' en bruto, sin alteración alguna. Posteriormente se aplica la Transformada de Fourier Rápida a la señal obtenida para poder estudiar los datos en el dominio de la frecuencia, siendo la parte real la frecuencia en Hercios (Hz) y la parte imaginaria la energía en Decibelios (dB).

Asimismo se calcula la función de Densidad Espectral de Potencia (PSD en inglés) para obtener de forma clara y visual aquellas regiones del subespectro muestreado que indican mayor distribución de potencia (dB/Hz), posibles picos significativos en determinadas frecuencias (Hz).

En primer lugar se ejecuta en la Raspberry Pi el script *single\_freq\_detect* para obtener un número determinado de muestras y poder analizar los valores generados.

Se configura el sistema, mediante código, para tomar 10 muestras. Asimismo, se emiten de forma manual 3 pulsos con los controles remotos de automóvil y coche RC.

El primer paso comprende la correcta recepción de la frecuencia BALIZA (434 MHz). Se obtienen los siguientes resultados, para una captura de 10 muestras:



```
single_freq_detect x
C:\TFG\python\prueba_SDR\venv\Scripts\python.exe C:\TFG\python\prueba_SDR\single_freq_detect.py
-0.019607843137254943,0.0039215686274509665
-0.019607843137254943,0.0039215686274509665
-0.0117647058823529,0.0039215686274509665
-0.019607843137254943,0.0039215686274509665
-0.0117647058823529,-0.0039215686274509665
-0.019607843137254943,-0.0039215686274509665
-0.0117647058823529,-0.0039215686274509665
-0.019607843137254943,-0.0117647058823529
-0.0117647058823529,-0.0039215686274509665
-0.0039215686274509665,-0.0039215686274509665
Found Elonics E4000 tuner
Exact sample rate is: 2500000.107620 Hz
Process finished with exit code 0
```

```
data.txt
Archivo  Editar  Ver
1706720958.7396626
-0.019607843137254943,0.0039215686274509665
1706720959.7426066
-0.019607843137254943,0.0039215686274509665
1706720960.7482452
-0.0117647058823529,0.0039215686274509665
1706720961.7486749
-0.019607843137254943,0.0039215686274509665
1706720962.7596602
-0.0117647058823529,-0.0039215686274509665
1706720963.7715478
-0.019607843137254943,-0.0039215686274509665
1706720964.783631
-0.0117647058823529,-0.0039215686274509665
1706720965.7948375
-0.019607843137254943,-0.0117647058823529
1706720966.8080163
-0.0117647058823529,-0.0039215686274509665
1706720967.8153365
-0.0039215686274509665,-0.0039215686274509665
Ln 19, Col 15  625 caracteres.  100%  Windows (CF)  UTF-8
```

Figura 35. Captura fichero de texto 'data.txt'. Detección de 'Baliza'.

Los valores representan la Amplitud y la Fase de la señal recibida. Asimismo se registra el instante de captura de cada muestra, para ello se usa el Tiempo Unix dado por el sistema.

Aunque, como se ha indicado antes, se emiten 3 pulsos, el receptor capta un 4º pulso.

Dentro de las marcas amarillas se puede comprobar el cambio de los valores, siendo estos los correspondientes a los pulsos emitidos.



Asimismo, se prueba el dispositivo usando el software SDRSharp AirSpy para ver los pulsos emitidos de forma gráfica en el cambio del espectrograma.

Las regiones contenidas dentro de las marcas de color amarillo indican los pulsos generados por el control remoto de automóvil (434.43 MHz), simulando los emitidos por una radio-baliza.

Por tanto, queda demostrado que el dispositivo SDR es capaz de captar esta frecuencia y detectar los pequeños pulsos.

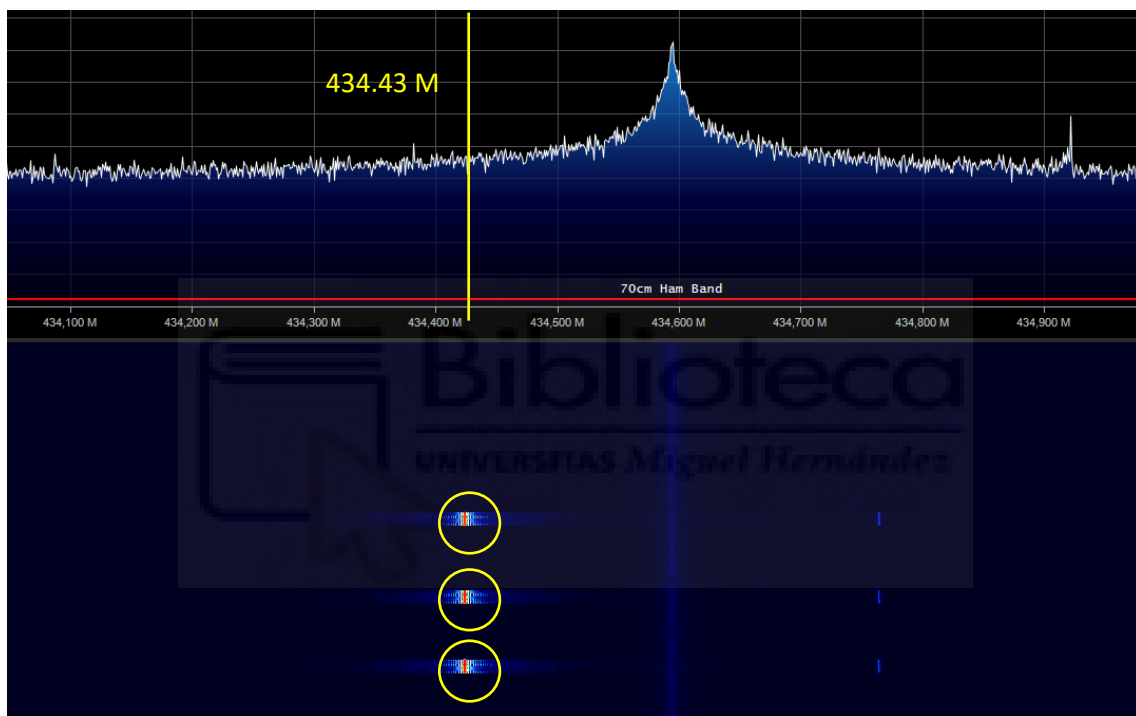
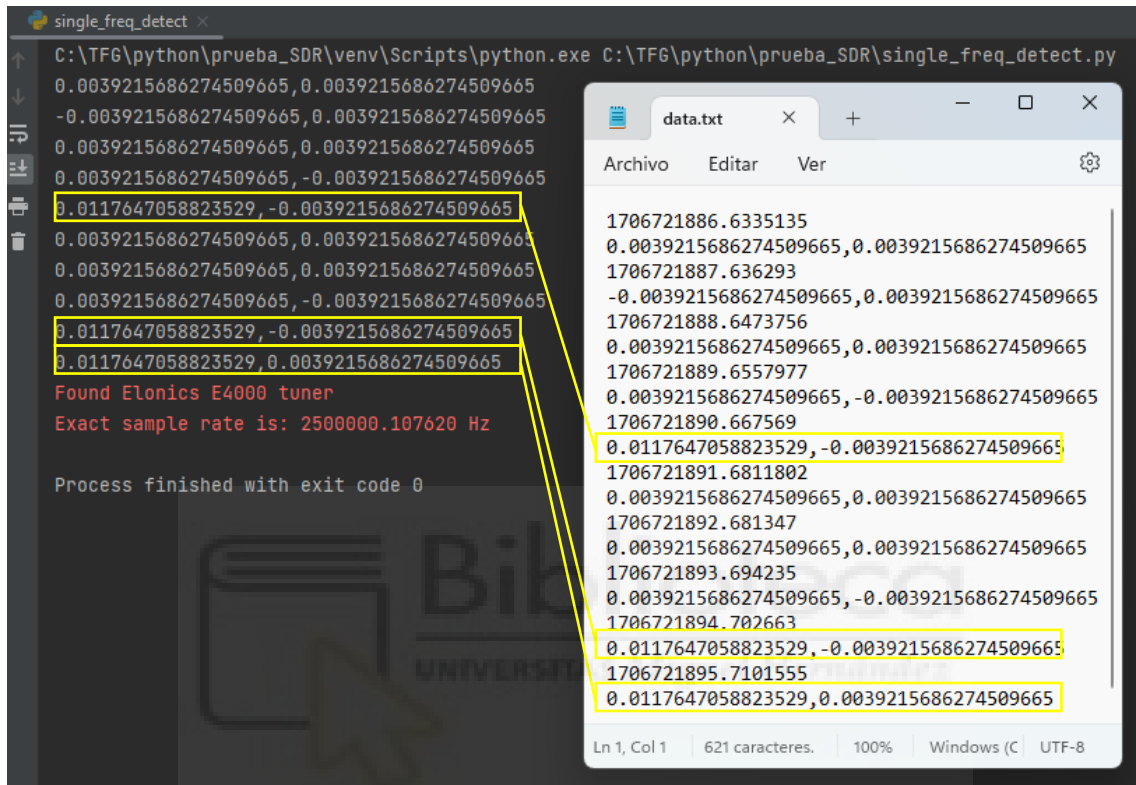


Figura 36. Captura software SDRSharp-AirSpy. Detección de 'Baliza'.

Se repite el experimento para la detección de las frecuencias en el rango de pocos GHz, equivalentes a RADAR. Los resultados obtenidos son equivalentes al experimento en MHz, se perciben claramente los 3 pulsos emitidos.

Por tanto, efectivamente como indica el fabricante el dispositivo sdr es capaz de trabajar con frecuencias del orden de GHz (máx. 2.3GHz) y registrar pequeñas variaciones.



The image shows a terminal window titled 'single\_freq\_detect' and a text editor window titled 'data.txt'. The terminal window displays the output of a Python script, including the exact sample rate: 'Exact sample rate is: 2500000.107620 Hz'. The text editor window shows a list of numerical data points, with several lines highlighted in yellow. These highlighted lines correspond to the data points in the terminal window that are also highlighted in yellow.

```
0.0039215686274509665,0.0039215686274509665
-0.0039215686274509665,0.0039215686274509665
0.0039215686274509665,0.0039215686274509665
0.0039215686274509665,-0.0039215686274509665
0.0117647058823529,-0.0039215686274509665
0.0039215686274509665,0.0039215686274509665
0.0039215686274509665,0.0039215686274509665
0.0039215686274509665,-0.0039215686274509665
0.0117647058823529,-0.0039215686274509665
0.0117647058823529,0.0039215686274509665
Found Elonics E4000 tuner
Exact sample rate is: 2500000.107620 Hz
Process finished with exit code 0
```

```
1706721886.6335135
0.0039215686274509665,0.0039215686274509665
1706721887.636293
-0.0039215686274509665,0.0039215686274509665
1706721888.6473756
0.0039215686274509665,0.0039215686274509665
1706721889.6557977
0.0039215686274509665,-0.0039215686274509665
1706721890.667569
0.0117647058823529,-0.0039215686274509665
1706721891.6811802
0.0039215686274509665,0.0039215686274509665
1706721892.681347
0.0039215686274509665,0.0039215686274509665
1706721893.694235
0.0039215686274509665,-0.0039215686274509665
1706721894.702663
0.0117647058823529,-0.0039215686274509665
1706721895.7101555
0.0117647058823529,0.0039215686274509665
```

Figura 37. Captura fichero de texto 'data.txt'. Detección de 'Radar'.

Cabe observar en la imagen que si bien se ajusta la frecuencia a 2 GHz, puesto que en un principio no es posible ajustarla a 2.4 GHz, finalmente la frecuencia exacta ajustada automáticamente por el SDR es de 2500000.107620 Hz, es decir aproximadamente 2.5 GHz.

Del mismo modo que en la detección de la Baliza, se muestra una captura del software SDRSharp en este caso para la detección del RADAR.

Mientras que en el caso de la Baliza, los pulsos emitidos artificialmente (434.43 MHz) no correspondían exactamente a los 434 MHz, en el caso del Radar sí que coinciden exactamente con los 2.4 GHz donde aparece una señal fuerte y concreta, de otra fuente emisora.

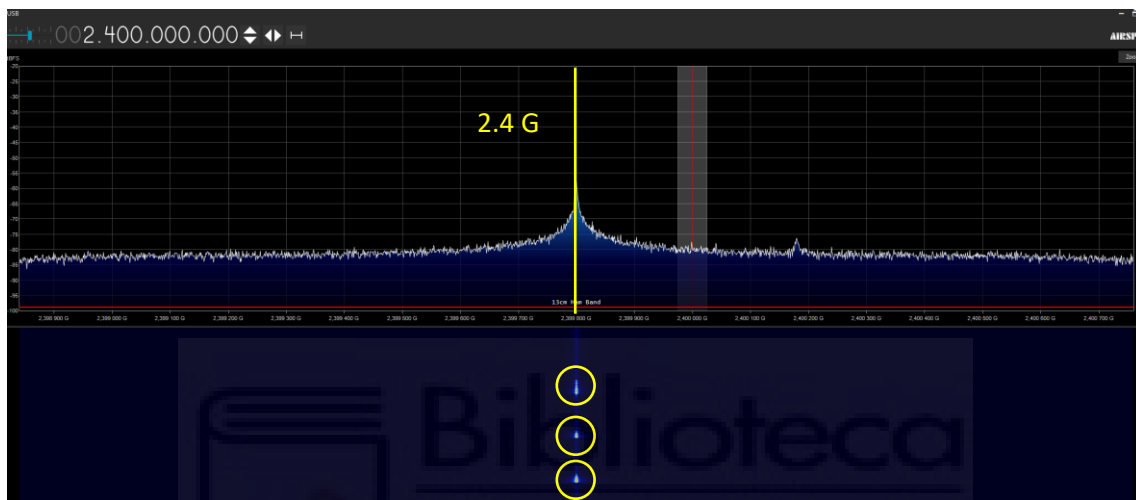


Figura 38. Captura software SDRSharp-AirSpy. Detección de 'Radar'.

En la segunda etapa del proyecto se estudian, gráficamente, los datos. Para ello se usa la Transformada Rápida de Fourier (FFT) para poder visualizar y comprender mejor los valores obtenidos, pasando al dominio de la Frecuencia. Correspondiendo el eje de ordenadas de la gráfica (OX) a la Frecuencia en Hz y el eje de abscisas (OY) la Magnitud en dBs.

También se calcula la Densidad Espectral de Potencia para conocer y observar aquellas regiones del espectro registrado donde existe una mayor o menor distribución de la potencia y por tanto poder deducir los posibles picos significativos. El valor de la Densidad (dB/Hz) queda representado en el eje OX y la Frecuencia (Hz) a la que corresponde en el eje OY.

Para la obtención de las gráficas se estudia un caso adicional que es la banda de los 100MHz, la cual comprende las emisoras de radio FM, puesto que ofrece más información visual de la señal (picos, anchos de banda...). De esta banda se registran los datos de las muestras de la señal capturada así como de su valor absoluto: la Amplitud y la Fase, la Magnitud y Frecuencia tras aplicar la FFT y la Potencia Relativa y la Frecuencia tras calcular la PSD.

Para verificar la correcta representación de la FFT/PSD, y por lo tanto la correcta recepción de datos, de nuevo se usa el software Airspy para poder comparar con los resultados obtenidos con *Sdr\_data\_graphics*.

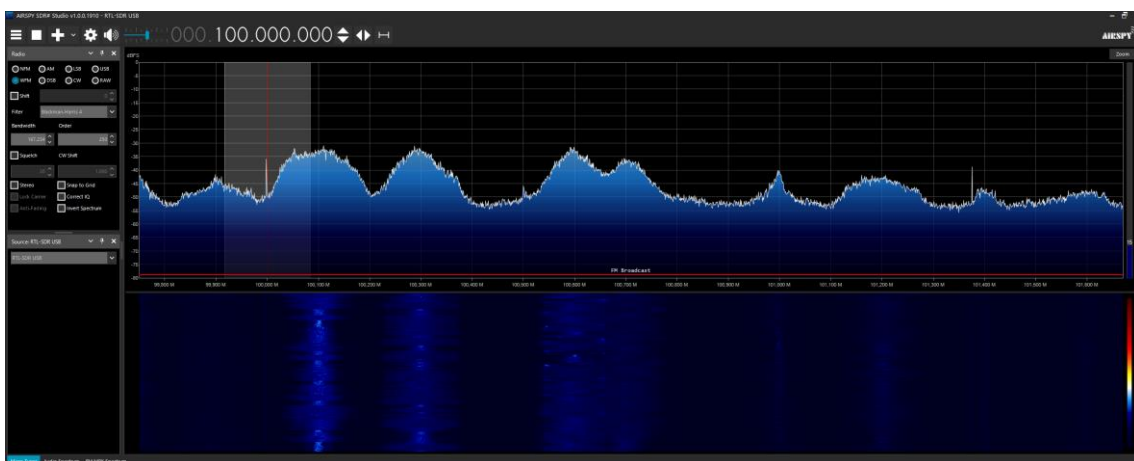


Figura 39. Captura AirSpy. Banda emisoras FM (100 MHz).

Se observa que la forma de ambas señales, tanto en Python como en AirSpy, es igual y se distinguen claramente las diferentes regiones que equivalen a las diferentes emisoras de radio FM.

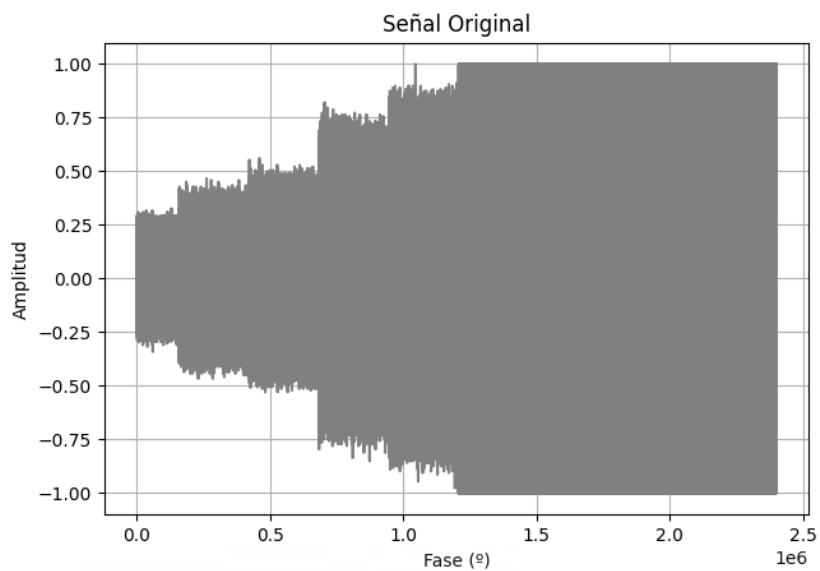


Figura 40. Señal Original.

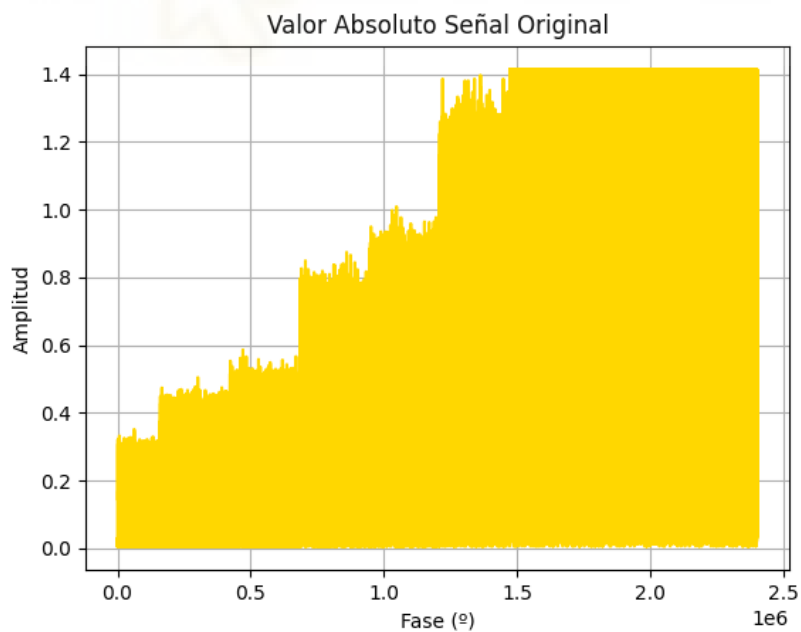


Figura 41. Valor Absoluto de la Señal Original.

Los resultados de la FFT se representan normalizados y centrados en el origen (0,0). En el eje OX la Frecuencia se encuentra normalizada entre los valores de -1 y 1 en escala de MHz (1e6 Hz). En cuanto a los valores de Magnitud, también se mantiene la simetría respecto al eje horizontal desde el origen (0,0) pero las magnitudes no se normalizan.

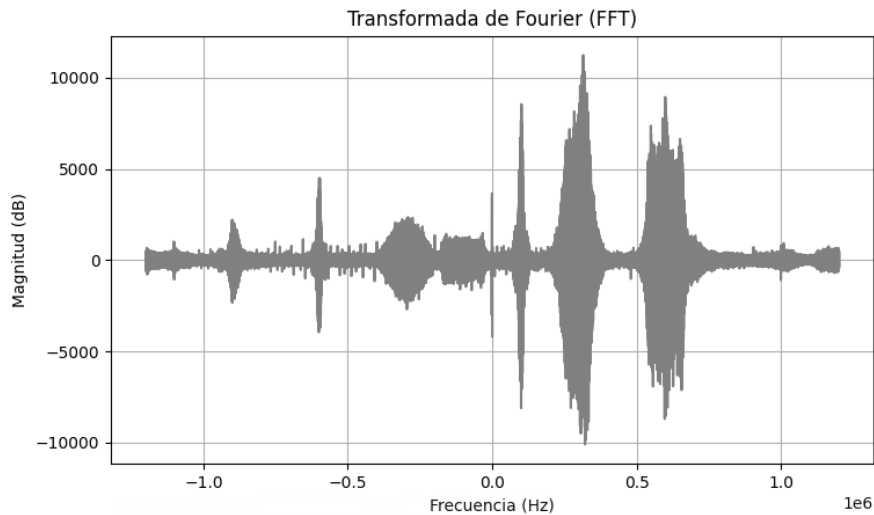


Figura 42. FFT de la Señal Original.

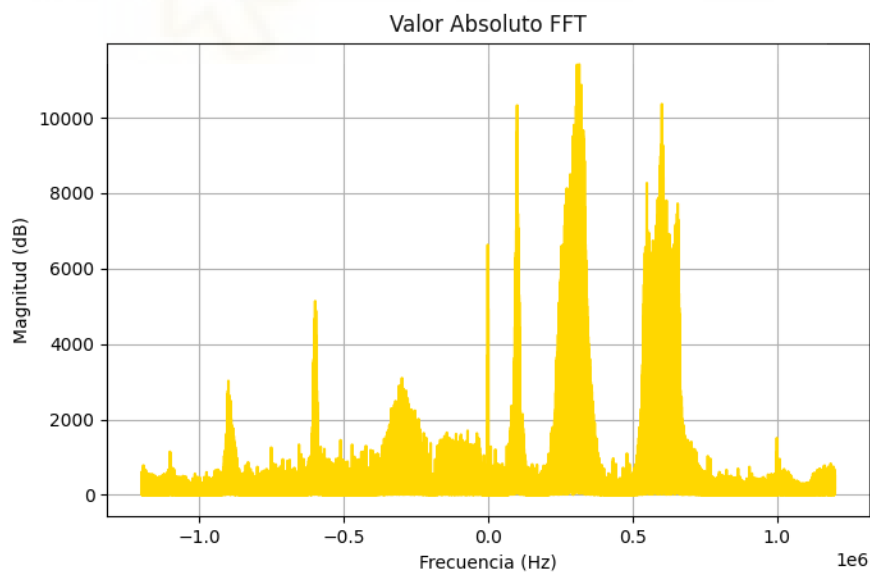


Figura 43. Valor Absoluto de la FFT.

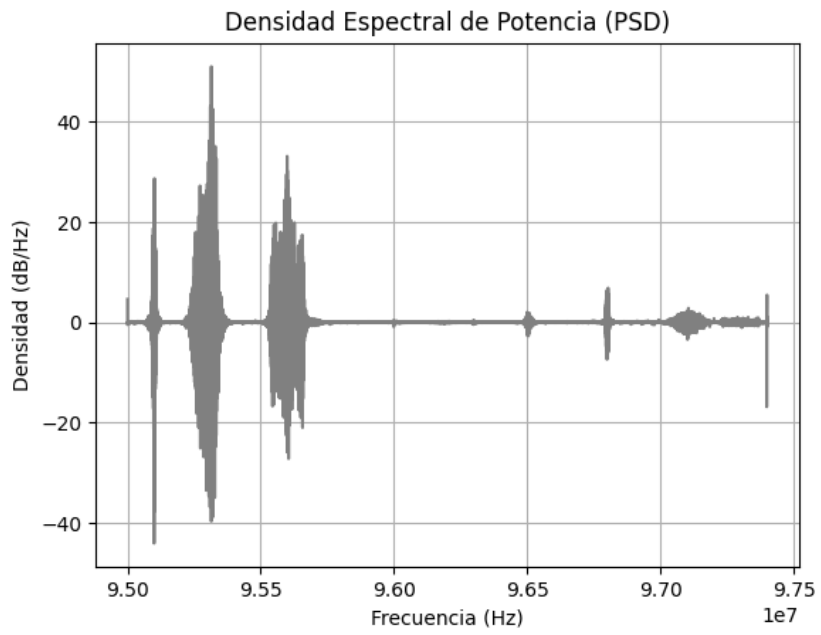


Figura 44. PSD de la Señal Original.

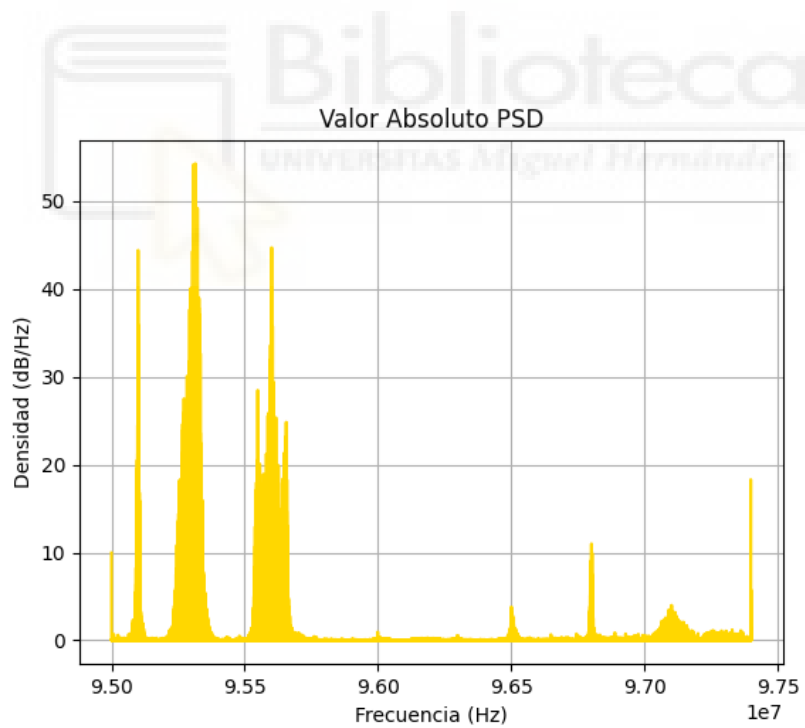


Figura 45. Valor Absoluto de la PSD.



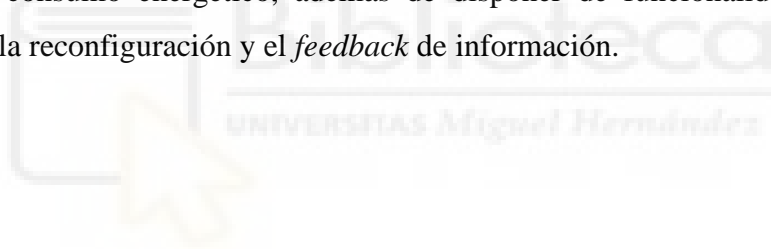


## Capítulo 5. Conclusiones

A modo de finalización, se hará una breve evaluación del sistema empleado así como de los resultados obtenidos.

Tras estudiar, diseñar, implementar y testear el sistema descrito en el presente documento se confirma la viabilidad de este para la aplicación final como prototipo de obtención de datos en el ámbito de la actividad pesquera, si bien presenta algunas limitaciones técnicas. Asimismo, cabe mencionar la ingente cantidad de información que este sistema es capaz de obtener una vez estudiado más detenidamente.

Por lo tanto, en lo que refiere a los objetivos propuestos en el inicio de este documento, se concluye que se han alcanzado totalmente: estudio de diversas soluciones existentes, iniciativa y diseño de un nuevo sistema basado en dispositivos disponibles en el mercado actualmente, atendiendo a las restricciones de independencia tecnológica, coste económico y consumo energético, además de disponer de funcionalidades como la conectividad, la reconfiguración y el *feedback* de información.





## Capítulo 6. Futuras investigaciones

Aunque el sistema desarrollado no arroja muchos resultados a nivel cuantitativo ya que comprende únicamente el diseño y la puesta en marcha del mismo, sienta las bases de futuros proyectos de investigación y aplicaciones relacionadas con el procesamiento de señales RF.

Para futuros avances de este sistema se propone el uso exclusivo de microcontroladores para reducir significativamente el consumo de potencia. La depuración de errores y el diseño de un prototipo físico final usando tecnologías de impresión 3D así como el diseño de un nuevo sistema electrónico que agrupe las tecnologías de recepción de señales RF y procesamiento en un mismo circuito impreso destinado específicamente para incorporarlo a la app Android y el sistema FAMIS en el ámbito del control de pesca.





## BIBLIOGRAFÍA

- [1] La pesca en la economía española.  
[https://www.mapa.gob.es/es/ministerio/servicios/analisis-y-prospectiva/aypseriepescan4vabspa2018v4\\_tcm30-551141.pdf](https://www.mapa.gob.es/es/ministerio/servicios/analisis-y-prospectiva/aypseriepescan4vabspa2018v4_tcm30-551141.pdf)
- [2] Misiones de la Armada de vigilancia pesquera y marítima.  
[https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspañola/conocenosnoticias/prefLang-es/00noticias--2023--06--NT-123-PATRULLEROS-ALTURA-es?\\_selectedNodeID=5785087&\\_pageAction=selectItem](https://armada.defensa.gob.es/ArmadaPortal/page/Portal/ArmadaEspañola/conocenosnoticias/prefLang-es/00noticias--2023--06--NT-123-PATRULLEROS-ALTURA-es?_selectedNodeID=5785087&_pageAction=selectItem)
- [3] FAO: Artes de Pesca.  
<https://www.fao.org/3/y3427s/y3427s04.htm>
- [4] OPAGAC: Las redes FAD no enmallantes.  
<https://opagac.org/tag/fad-no-enmallantes/>
- [5] Listado de Buques de la flota de INPESCA S.A.  
<https://www.inpesca.com/inpesca/dm/buques.asp?nombre=2033&hoja=0&sesion=1>
- [6] SASS: Información sobre radiobalizas.  
<http://www.sass.gov.ar/txt/406.html>
- [7] Comisión Europea: Espectro Radioeléctrico.  
<https://digital-strategy.ec.europa.eu/es/policies/radio-spectrum>
- [8] Cuadro Nacional de Atribución de Frecuencias.  
<https://www.boe.es/boe/dias/2023/06/16/pdfs/BOE-A-2023-14422.pdf>
- [9] Sistema FAMIS: *Using mobile device's sensors to identify fishing activity.*  
<https://link.springer.com/article/10.1007/s00773-019-00694-5>
- [10] Sistema FAMIS: *Is the vessel fishing? Discrimination of fishing activity with low-cost intelligent mobile devices through traditional and heuristic approaches.*  
<https://www.sciencedirect.com/science/article/pii/S0957417422004973>
- [11] Rtl-Sdr.com: sitio web especializado en el uso de dispositivos SDR.  
<https://www.rtl-sdr.com/>
- [12] ESPloradores: sitio web sobre microcontroladores.  
<https://www.esploradores.com/>

- [13] Hojas de datos del microcontrolador ESP32-S3-USB-OTG.  
[https://espressif-docs.readthedocs-hosted.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-usb-otg/user\\_guide.html](https://espressif-docs.readthedocs-hosted.com/projects/esp-dev-kits/en/latest/esp32s3/esp32-s3-usb-otg/user_guide.html)
- [14] Hojas de datos del microcontrolador ESP32-S3-DEVKITM-1.  
<https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32s3/hw-reference/esp32s3/user-guide-devkitm-1.html>
- [15] Repositorio de firmwares MicroPython para ESP32-S3 SoC.  
[https://micropython.org/download/ESP32\\_GENERIC\\_S3/](https://micropython.org/download/ESP32_GENERIC_S3/)
- [16] Repositorio firmware CircuitPython para ESP32-S3-USB-OTG.  
[https://circuitpython.org/board/espressif\\_esp32s3\\_usb\\_otg\\_n8/](https://circuitpython.org/board/espressif_esp32s3_usb_otg_n8/)
- [17] Repositorio firmware CircuitPython para ESP32-S3-DevKitM-1.  
[https://circuitpython.org/board/espressif\\_esp32s3\\_devkitm\\_1\\_n8/](https://circuitpython.org/board/espressif_esp32s3_devkitm_1_n8/)
- [18] NooElec: Kit HF NESDR smartXTR.  
<https://www.nooelec.com/store/nesdr-smart-xtr-hf.html>
- [19] Hojas de datos de Raspberry Pi Zero 2W  
<https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf>
- [20] Librería de Python Pyrtlsdr  
<https://pypi.org/project/pyrtlsdr/>
- [21] Librería de Python Numpy  
<https://pypi.org/project/numpy/>
- [22] Librería de Python Matplotlib  
<https://pypi.org/project/matplotlib/>



## ANEXO I: Ejemplo de comunicación TCP/IP con SDR

Como funcionalidad adicional se presenta un ejemplo de comunicación remota por protocolo TCP/IP con el SDR usando las clases y métodos que ofrece la librería pyrtlsdr para ello:

- RtlSdrTcpServer: Clase para crear el objeto “Servidor”.
- RtlSdrTcpClient: Clase para crear el objeto “Cliente”.
- Sdr.connect: Método para establecer la conexión TCP/IP.

### def SDR\_TCP\_SERVER()

Configura el Servidor. Declara el objeto “Servidor” y arranca la conexión (run\_forever).

```
from rtlsdr import RtlSdrTcpServer

def SDR_TCP_SERVER():
    servidor = RtlSdrTcpServer()
    servidor.run_forever()

    print("Servidor RTL-SDR esperando conexion...")

if __name__ == "__main__":
    SDR_TCP_SERVER()
```

### def SDR\_TCP\_CLIENT()

Configura el Cliente. Declara el objeto “Cliente” y establece la conexión (cliente.connect).

```
from rtlsdr import RtlSdrTcpClient

def SDR_TCP_CLIENT(direccion_IP, puerto_IP, num_muestras):
    cliente = RtlSdrTcpClient(direccion_IP, puerto_IP) # Cambiar por
la dirección IP del servidor y el puerto utilizado
    sdr = cliente.connect()
    muestras = sdr.read_samples(num_muestras) # Ejemplo de lectura de
muestras
    cliente.disconnect()
    cliente.close()

    return muestras

if __name__ == "__main__":
    num_muestras = 1024
    direccion_IP = '192.168.1.46' # Dirección IP RPI_02W
    puerto_IP = '22' # Puerto IP RPI_02W
    muestras_recibidas = SDR_TCP_CLIENT()
    print("Muestras recibidas:", muestras_recibidas)
```





