UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

# IMPLEMENTACIÓN DE UN SIMULADOR VECTORIAL DE SEÑALES RADAR CON AGILIDAD INTRAPULSO E INTERPULSO

**Santiago Carrilero Robredo**

**Tutor: Javier Martí Sendra**

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2016-17

Valencia, 24 de julio de 2017

## Resumen

En pleno era de la tecnología, los radares van ganando importancia en una gran cantidad de campos, desde su ya consolidado terreno que es la aviación y el sector naval, hasta los modernos coches autónomos e inteligentes que dependen de los radares para su funcionamiento. La palabra o término "RADAR" proviene de "RAdio Detection And Ranging", un método utilizado para conocer la localización y velocidad de objetos. En este trabajo nos centraremos en la generación de este tipo de señales. En él crearemos un software capaz de generar las formas de onda más modernas dentro de este campo. Su desarrollo se realizará en la plataforma Python, con la intención de generar una interfaz para ayudar al usuario. Paralelamente, se creará otro generador de señales de Radar en la plataforma GNU Radio. Se generarán señales radar con agilidad intrapulso e interpulso en alta frecuencia de hasta 40 GHz. De esta forma se conseguirá una plataforma ideal para testear equipos y favorecer su desarrollo.



## Resum

En plena era de la tecnologia, els radars van guanyant importància en una gran quantitat de camps, des del seu terreny ja consolidat que és l'aviació i el sector naval, fins als moderns cotxes autònoms i intel·ligents que depenen dels radars per al seu funcionament. La paraula o terme "RADAR" prové de "Ràdio Detection And Ranging", un mètode utilitzat per conèixer la localització i velocitat d'objectes. En aquest treball ens centrarem en la generació d'aquest tipus de senyals. En ell crearem un programa capaç de generar les formes d'ona més modernes dins d'aquest camp. El seu desenvolupament es realitzarà a la plataforma Python, amb la intenció de generar una interficie per ajudar l'usuari. Paral·lelament, es crearà un altre generador de senyals de Radar a la plataforma GNU Ràdio. Es generaran senyals radar amb agilitat intrapols i interpols en alta freqüència de fins a 40 GHz. D'aquesta manera s'aconseguirà una plataforma ideal per testejar equips i afavorir el seu desenvolupament.



## Abstract

Nowadays, the importance of radars is raising in a large number of fields, from its already consolidated terrain that is aviation and naval sector, to the modern autonomous and intelligent cars that depend on the radars for their operation. The word or term "RADAR" comes from "RAdio Detection And Ranging", a method used to know the location and speed of objects. In this thesis, we will focus on the generation of this type of signals. In it we will create a software capable of generating the most modern waveforms within this field. Its development will be carried out on the Python platform, with the intention of generating an interface to help the user. In parallel, another Radar signal generator will be created on the GNU Radio platform. Radar signals will be generated with intrapulse and interpulse agility at high frequencies up to 40 GHz. This will provide an ideal platform to test equipment and promote its development.

# Agradecimientos

Como punto y final de esta gran experiencia que ha sido la carrera de Teleco donde he tenido una gran cantidad de buenos momentos, quería agradecer a una serie de personas que me han ayudado a superarla:

En primer lugar, a mi familia, ya que ellos siempre han sido los primeros en ayudarme con todo y han depositado la confianza en mí desde el primer día que comencé el grado.

A todos mis compañeros y, sobre todo, amigos con los que he compartido los buenos y malos momentos que nos ha dado esta carrera.

También quiero agradecer a esa persona toda la ayuda que me ha ofrecido hasta el final, académica y personalmente. Gran parte de lo que soy te lo debo a ti.

A cada profesor que ha puesto el ímpetu por transmitirme parte de sus conocimientos y conseguir salir orgulloso de una carrera que debo admitir que entre con dudas.

Por último, pero no menos importante, quiero agradecer a DAS Photonics, la empresa donde he hecho las prácticas durante año y medio, por haber apostado y puesto toda su confianza en mí. Por supuesto, a todos los compañeros que lo han hecho posible y me han aportado tanto.

¡Gracias!

# Índice

# Capítulo 1. Introducción y motivación

Nos encontramos en un mundo en pleno desarrollo y con incesantes cambios e innovaciones. Cada mañana nos levantamos con la llegada nuevos Smartphones, no solo más bonitos, sino que, con procesadores más rápidos, cámaras con mayor cantidad de megapixels y mayor luminosidad, mejores pantallas y una gran cantidad de mejoras.

La tecnología y en especial la electrónica, se ha metido en el día a día de las personas, tanto que la inmensa mayoría de las personas ya no sabría cómo vivir sin ella. Cada día somos más dependientes de ella, nuestros coches dependen de electrónica para funcionar, incluso nuestros trabajos dependemos de ella, ya que más de la mitad de la población la necesita en su trabajo. Puede parecer un avance negativo, ya que nunca suele ser bueno depender de algo, pero sin embargo la tecnología ha evolucionado de una forma que nos facilita la vida y el día a día.

Paralelamente, el sector militar se trata de otro aspecto donde a pesar de no ser conscientes de su enorme evolución, ya sea porque tampoco es un tema de gran interés entre la población, o por su estado de secretismo, es sabido que la mayor parte de las nuevas armas no son de interés hacerlas públicas para que el enemigo no tenga conocimiento de ellas y pueda combatirlas. Pero, sin embargo, se encuentra en el sector más puntero de la tecnología, donde los países invierten enormes cantidades de dinero para ir en cabeza ante el resto de países.

Y es que a pesar de encontrarnos en pleno siglo XXI, las guerras siguen existiendo, los conflictos armados siguen a la orden del día, y la rivalidad entre países se hace patente en las noticias que vemos por los medios de comunicación. Por ello, la paz mundial se sigue considerando una utopía lamentablemente.

Así, ningún país quiere problemas dentro de su territorio, pues no solo se desarrolla material militar para el ataque, sino también para su propia defensa. En consonancia con nuestro Grado en Telecomunicaciones, existe un sector llamado Guerra Electrónica, o más comúnmente conocido como Electronic Warfare. Resumiendo, se basa en la guerra del espectro radioeléctrico, que nació con la creación del primer Radar. A día de hoy se intenta combatir, bien evitándolo como los aviones más avanzados o conociendo quien intenta hallar nuestra posición. Esto es posible gracias a los equipos ELINT (Electronic Intelligence), aparatos que rastrean el espectro en busca de señales radar.

Nuestra motivación sobre este trabajo, comienza con la necesidad de conocer y confirmar el buen funcionamiento de los equipos ELINT que son creados en la empresa que he tenido la oportunidad de realizar las prácticas. La necesidad de crear un programa de testing como tal sale de la ausencia de señales radar modernas en el territorio que nos rodea, donde solo podríamos hallar señales básicas como las de los barcos que tenemos en nuestra costa o de aviones que nos sobrevuelan. Por ello, nos centraremos en recrear el posible espectro que nos podríamos encontrar en un entorno hostil con la generación de las más modernas formas de onda.

# Capítulo 2. Estado del arte

## 2.1 Tecnologías Radar

La invención del RADAR data del siglo pasado motivado por un gran número de aplicaciones desde su descubrimiento del fenómeno físico clave, hasta la infinidad de utilidades prácticas.

El principio de la historia del Radar comienza en 1886 cuando Heinrich Rudolf Hertz consiguió demostrar que las ondas electromagnéticas se reflejan en superficies metálicas.

El primer Radar en aparecer no tardó más de 30 años y es que en 1904, el alemán Christian Huelsmeyer, se creó el "Telemobiloscopio". Este se trata de un sistema transmisor-receptor utilizado para detectar objetos metálicos por medio de ondas electromagnéticas. Fue diseñado como un dispositivo anticolisión para buques con unos buenos resultados. El rango de detección conseguido alcanzaba los 3000 metros sin ningún tipo de amplificación con unas longitudes de onda entre los 40-50 cm. A pesar de los buenos resultados y su gran utilidad, el invento pudo prosperar debido a la falta de interés por las autoridades navales y la industria. Sin embargo, Christian patentó su telemobiloscopio en EE.UU. con fecha de 16 de enero de 1906 y abrió su propia empresa con la que planeó un nuevo telemobiloscopio que funcionaría hasta los 10.000 metros.

*Figura 1. Antena, medio receptor y transmisor del Telemobiloscopio.*

En 1917, Nikola Tesla estableció los principios teóricos del Radar moderno, teniendo en cuenta las frecuencias y niveles de potencia. Siguiendo los principios que dictó, en 1934 se realizan ensayos sobre sistemas de detección de onda corta, gracias a un estudio sistemático del magnetrón.

Durante los años previos a la Segunda Guerra Mundial, muchos inventores, científicos e ingenieros contribuyeron en el desarrollo del radar. Las grandes potencias mundiales como fueron Alemania, Reino Unido y Estados Unidos fueron desarrollando de forma paralela distintos sistemas radar, aportando grandes avances cada uno de ellos para llegar a lo que hoy conocemos sobre sistemas Radar.

Cabe destacar el gran trabajo realizado por los físicos Robert Watson-Watt y Arnold Wilkins quienes con la intención de crear un "rayo de la muerte", concluyeron en la creación del radar más avanzado de la época que permitió a Reino Unido avanzarse tecnológicamente a sus competidores.

En la actualidad, son muy utilizados en varios ámbitos como es el sector naval, siendo obligatoria su utilización para medir el rumbo y distancia de los barcos para evitar la colisión entre ellos. También facilita la navegación y ayuda al posicionamiento en el mar cuando se está dentro del alcance de la costa u otras referencias como naves o islas. En el puerto, se utilizan sistemas de

radar para vigilar y regular el movimiento de cada barco en aguas cercanas. Para mejorar la RCS (Radar Cross Section) se obliga a cada barco a instalar un reflector radar.

En la aviación, cada avión está equipado de dispositivos radar que advierten de otros aviones u obstáculos en aproximación a su camino, mostrar información meteorológica y dar lecturas más precisas de la altitud. De esta manera se consigue mejorar la seguridad, permitiendo incluso aterrizajes en condiciones adversas, como niebla, siendo asistido por radares.

Para realizar previsiones meteorológicas es utilizado ya que consigue vigilar la precipitación, convirtiéndose en la principal herramienta para la predicción a corto plazo de tormentas eléctricas, tornados o tormentas de nieve. Los geólogos utilizan radares especializados en la tierra para obtener la composición de la corteza terrestre. La policía utiliza radares para monitorizar la velocidad de los vehículos en las carreteras.

## 2.2 Dispositivos Contra Radares

Ante la llegada al escenario bélico de los sistemas radar, supuso una gran ventaja ante el enemigo de forma ofensiva, donde permite conocer con mayor precisión la posición del enemigo, y defensiva, donde en caso de ataque de pueda realizar una defensa más rápida, efectiva y organizada. Por tanto, se comenzaron a idear distintos métodos contra esta medida.

Naciendo así el término de Guerra Electrónica, que consiste en una actividad tecnológica y electrónica con el fin de determinar, explotar, reducir o impedir el uso hostil de todos los espectros de energía, por parte del adversario y a la vez conservar la utilización de dicho espectro en beneficio propio.

### *2.2.1 Jamming*

El Jamming, o interferencia intencionada, es la transmisión de señales de radiocomunicación de forma deliberada para perturbar la transmisión de otra señal de radio, en nuestro caso, señales radar. A diferencia de la no intencionada, en la que se trata de transmitir la señal de forma negligente o accidental en un espectro de frecuencias ya que está siendo usado, la interferencia intencionada tiene por objetivo impedir una comunicación por radiofrecuencias alterándola o anulándola lo suficiente como para que el receptor no pueda interpretarla.

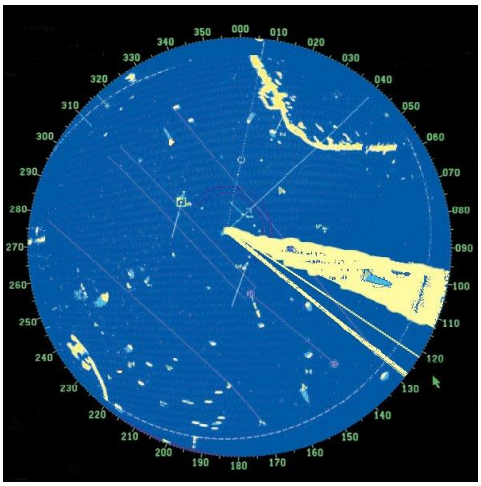



*Figura 2. Ejemplo de Jamming en 105º*

La complejidad del Jammer consta en conocer la frecuencia de funcionamiento del Radar, para ello, se realiza un barrido de ruido en banda ancha para cubrir la mayor cantidad de radares con un solo Jammer compensando así cualquier incertidumbre en la frecuencia del radar.

### *2.2.2 Decepción electrónica*

La decepción electrónica o engaño electrónico, significa la radiación deliberada, la reradiación, la alteración, la supresión, la absorción, la negación, la mejora o la reflexión de la energía electromagnética de una manera destinada a transmitir información engañosa y negar la información válida a un enemigo.

El primer tipo de engaño electrónico es el manipulador o de simulación, donde se realizan acciones para alterar o simular radiaciones electromagnéticas amigas, con el fin de lograr la decepción.

El segundo tipo es la imitativa, donde nuestra intención será emitir las propias radiaciones enemigas. Un ejemplo sería la retransmisión de cada pulso radar funcionando a una frecuencia ligeramente diferente, donde se imitaría una velocidad, y con un retardo diferente, imitando estar en una posición alternativa.

### *2.2.3 Sistemas ELINT*

En este caso no nos encontramos ante una contra medida, ya que un ELINT (Electronic Inteligence) se dirige a la recopilación (observación y registro) y proceso para posteriores fines de inteligencia para detectar señales de radar. Este método permite interceptar cada señal e identificarla conociendo mediante una base de datos el tipo de objetivo que radía la señal. El funcionamiento también puede ser a la inversa ya que con el uso de varios sistemas ELINT y mediante sistemas de triangulación, podemos conocer la posición del objeto que radía la señal. La diferencia se basa en que nosotros podemos conocer su ubicación si que el objetivo sea consciente de ello ya que nosotros no emitimos ningún tipo de señal.

# Capítulo 3. Metodología de trabajo

## 3.1 Gestión del proyecto

Para la creación del simulador vectorial de señales Radar, debemos tener en cuenta las necesidades básicas del proyecto. Se tratará del desarrollo de un generador de entornos complejos de Guerra Electrónica (Señales Radar) y el control de instrumentación de Microondas de hasta 40 GHz. Se deberá desarrollar un simulador radar de laboratorio, haciendo uso de hardware SDR (Software-Defined Radio) y tratamiento digital de la señal.

Para el simulador, se deseará que se cumplan los siguientes objetivos y campos:

- **Aprendizaje sencillo**: se orientará para un primer uso lo más sencillo posible para una persona que utilice el software por primera vez.
- **Facilidad de uso**: el software deberá ser sencillo a todos los niveles y requerir un bajo esfuerzo para un menor tiempo de configuración.
- **Fácil instalación**: se realizará el simulador de modo que necesitemos utilizar la menor cantidad de plug-ins externos y software.
- **Modular**: el objetivo será una implementación para poder reutilizar partes de código. Además, con una estructura de fácil
- **Lenguaje de programación sencillo**: debido a la importancia de la continuación de este proyecto, es una gran elección un lenguaje de programación como Python ya que conseguimos un lenguaje sencillo, de fácil aprendizaje y ampliamente conocido por la comunidad de programadores.

En la segunda parte del proyecto, nos centraremos en desarrollar un sistema Radar en una SDR, que en nuestro caso será la USRP B100 de la empresa Ettus Research montada con una Daughterboard modelo WBX, de quienes hablaré más adelante sobre sus especificaciones.

Para este proyecto, se piensa en una herramienta diferente, una que nos facilite el trabajo y nos permita un uso rápido y fiable. El software además debe ser compatible con los drivers[1] necesarios para controlar nuestro USRP. En suma a ello, debemos tener control al momento de la transmisión y, en caso de querer procesar la señal rebotada, de la recepción de datos. Por ello, necesitaremos poder procesar todo dato de forma instantánea. Así que fue elegido el GNU Radio Companion[2] debido a que cumple los requisitos esenciales.

[1] Controlador, rutina o programa que enlaza un dispositivo periférico al sistema operativo.
[2] https://www.gnuradio.org/

# Capítulo 4. Teoría

Los sistemas radar emiten señales con formas de onda características, generalmente, más sencillas que las empleadas para modulación de señales de comunicaciones. La información que puede suministrarnos un radar depende en gran medida del tipo de forma de onda que se utiliza, y por ello, comenzaremos a analizar las posibles características de una onda.

## 4.1 Características de la forma de onda

Las principales características que puede presentar una forma de onda de un radar son:

- La energía de la misma
- Resolución en alcance y en velocidad radial que proporciona
- Capacidad de rechazar respuestas indeseadas de blancos presentes en la escena

La **energía de la forma de onda del radar** viene dada por la integral

$$E_w = \int P(t)dt \qquad (1)$$

Donde *P* es la potencia instantánea transmitida por el radar. Se entenderá como forma de onda la gráfica completa, donde también se incluyen los momentos de silencio, teniendo la equivalencia siguiente:

$$E_W = P_{P,e}\,\tau \qquad (2)$$

Donde $P_{P,e}$ es la potencia de pico eficaz trasmitida y $\tau$ es la duración del pulso en cada ciclo.

La energía será relacionada con la capacidad de transmisión de potencia del sistema, su ciclo de trabajo máximo de operatividad (debido a las consideraciones de su hardware y el calentamiento térmico del sistema, como por la necesidad de estar en silencio para escuchar los ecos), y su resolución en alcance.

La **resolución del radar en alcance** será:

$$\delta R = \frac{c\tau}{2} \qquad (3)$$

El ancho del pulso es el factor primario en resolución en distancia. Por tanto, un sistema bien diseñado deberá tener la habilidad de distinguir objetivos que se encuentren separados a la mitad del tiempo del ancho del pulso $\tau$.

La capacidad de evitar blancos no deseados será tratada en cada forma de onda para mayor profundidad.

## 4.2 Ondas continuas (CW)

Los Radares de onda continua, se caracterizan por la utilización de dos antenas, una para transmisión y otra para recibir. En cambio, un radar pulsado, al existir periodos de silencio, durante los cuales la antena no transmite, podrán ser utilizados pues para recibir.

En los radares de onda continua la señal podrá ser monofrecuencia o estar modulada en frecuencia de una cierta manera, generalmente mediante una variación lineal (LFM, Linear Frequency Modulation). Su ecuación matemática será:

$$\boldsymbol{s(t) = A\cos\left[2\pi\left(f_0 t + \frac{k}{2}t^2\right)\right] \quad 0 \le t < T} \tag{4}$$

$$\boldsymbol{s(t) = s(t - nT) \quad nT \le t < (n+1)T} \tag{5}$$

Siendo $k$ la rampa de frecuencia.

Deberemos saber que en el caso de utilizar un radar CW monofrecuencia nos será imposible detectar el rango de un blanco, pero si su velocidad. En cambio, con un radar LFM-CW, donde T hace el papel del intervalo interpulso (PPI), sí nos será posible hacerlo, aunque existirá un rango máximo no ambiguo, del mismo modo que en los radares pulsados, dado por

$$\boldsymbol{R_u = c\frac{T}{2}} \tag{6}$$

Aunque utilizar un LFM-CW no será el único modo de obtener medidas de alcance con un CW. De hecho, también podremos utilizar esquemas multifrecuencia. Por ejemplo, se pueden transmitir dos señales monofrecuencia y mezclar sus ecos heterodinamente en la recepción. La diferencia de fase entre las dos señales que son del tipo

$$\begin{aligned} \boldsymbol{s_1(t)} &= \boldsymbol{A\cos(2\pi f_1 t)} \\ \boldsymbol{s_2(t)} &= \boldsymbol{A\cos(2\pi f_2 t)} \end{aligned} \tag{7}$$

Que nos llegan de la forma

$$\begin{aligned} \boldsymbol{s_1(t) = A\cos(2\pi f_1 t + \emptyset_1);} \quad & \boldsymbol{\emptyset_1 + 2\pi n_1) = \frac{4\pi f_1 R}{c}} \\ \boldsymbol{s_2(t) = A\cos(2\pi f_2 t + \emptyset_2);} \quad & \boldsymbol{\emptyset_2 + 2\pi n_2) = \frac{4\pi f_2 R}{c}} \end{aligned} \tag{8}$$

Siendo $n_1$ y $n_2$ son los responsables de que no podamos averiguar R a partir de $\emptyset_1$ o $\emptyset_2$ por separado, obteniendo

$$\Delta\emptyset + \boldsymbol{2\pi(n_2 - n_1) = \frac{4\pi(f_2 - f_1)R}{c}} \tag{9}$$

$$\Delta\emptyset = \boldsymbol{\emptyset_2 - \emptyset_1} \tag{10}$$

Continuamos de nuevo, teniendo a $n_1$ o $n_2$ que nos impiden recuperar R. Sin embargo, bajo la condición de que $n_1 = n_2$ tendremos que

$$\Delta\emptyset = \boldsymbol{\frac{4\pi(f_2 - f_1)R}{c} \rightarrow R = \frac{c\Delta\emptyset}{4\pi\Delta f}} \tag{11}$$

La condición $n_1 = n_2$ será cierta dentro de una cierta distancia. En efecto, si estamos en el punto justo delante del radar donde R sea cero, tendremos que $\emptyset_2 = \emptyset_1$, en concreto será cero, y por tanto, serán $n_1$ y $n_2$ también cero. Lo que ocurre es que, si R es demasiado grande, entonces $\Delta\emptyset$ será mayor que $2\pi$, pero nosotros medimos la fase entre 0 y $2\pi$ solo, de modo que la implicación en la anterior formula no es cierta, aunque si lo sea la ecuación a su izquierda. De modo que podemos concluir que la ecuación de alcance máximo no ambiguo vista, y que estaba referida a formas de ondas pulsadas no es válida y ha de ser sustituida por

$$R_u = \frac{c}{2\Delta f} \qquad (12)$$

Para radares CW bifrecuencia. Pudiéndose generalizar a radares multifrecuencia con más de dos frecuencias.

## 4.3 Radares pulsados

Los radares pulsados transmiten y reciben un tren de pulsos modulados. El alcance de un blanco se calcula como hemos explicado hasta ahora: como el resultado de calcular el recorrido de la señal en el camino de ida y vuelta de los pulsos al blanco. Un desplazamiento Doppler se debe igualmente en una velocidad radial relativa entre el blanco y el radar. Los cuatro parámetros fundamentales que definen la señal de un radar pulsado son las siguientes

- Frecuencia de la portadora
- Longitud del pulso
- Periodo de repetición
- Modulación del pulso

### *4.3.1 Pulsos monofrecuencia rectangulares*

Forma de onda más simple y empleada por los primeros sistemas radar. Son los más sencillos de generar en el transmisor y sencillos de procesar.

La detección de un eco en el caso de los radares CW se realiza mezclando la señal recibida por el radar con una réplica de la señal enviada que circula internamente en la cadena de recepción del radar, es decir, una copia de la señal que, en lugar de ir a la antena se mezcla con las señales que lleguen al radar en recepción. En el caso de los radares pulsados el mecanismo es diferente: la señal se filtra. Este filtrado se realiza digitalmente, es decir, una vez que la señal se ha muestreado y convertido en digital. Lo importante es que se puede demostrar que la respuesta ideal del filtro implementado en el radar para que el impacto del ruido presente en la señal afecte lo menos posible a la detección del eco se llama filtro adaptado y corresponde a una respuesta en frecuencia que corresponde exactamente a la transformada de Fourier del perfil del pulso que se envía invertido en tiempo. Entendiendo que:

- La aplicación de un filtro adaptado es equivalente a correlar el eco con una réplica de la señal.
- La mejor manera de reconocer un pulso recibido como eco será el buscar su similitud o correlación con la forma que tiene.

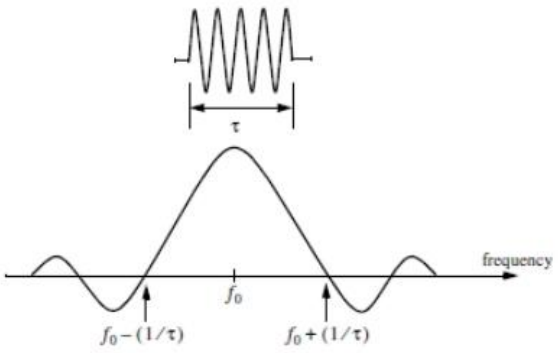



*Figura 3. Pulso monofrecuencia y su espectro*

Por tanto, la respuesta en frecuencia que corresponde exactamente a la transformada de Fourier del perfil del pulso que se envía invertido en tiempo corresponderá a dicho filtro.

### *4.3.2 Pulsos LFM o Chirp*

Un pulso lineal FM o también llamado Chirp, viene dado por la siguiente formula

$$s(t) = rect\left(\frac{t}{T}\right) Acos\left(2\pi f_0 + \frac{k^2}{2t^2}\right) \quad (13)$$

Donde

$$rect(x) = \begin{cases} 1\ si\ 0\ \leq x \geq 1 \\ 0\ si\ |x| > 1 \end{cases} \quad (14)$$

Para la detección de ecos se utiliza el mismo método de convolución que para los pulsos CW. Sin embargo, ahora el pulso se decorrela consigo mismo más rápido de lo que lo hacía antes, como se puede entender visualizando la operación de correlación como la concordancia en el solape de la señal con ella misma cuando desplazamos la réplica o copia sobre el original. La resolución espacial que me proporciona la convolución en la recepción del eco de una chirp es:

$$\delta R = \frac{c}{2B} \quad (15)$$

A diferencia de la resolución espacial de un pulso CW, que es:

$$\delta R = \frac{c\tau}{2} \quad (16)$$

Por tanto, podremos definir el producto entre ambos y llamarlo tasa de compresión de pulso:

$$PC = \tau B \quad (17)$$

Normalmente, la tasa de compresión es del orden de $10^6$ o mayor aún, de manera que la resolución espacial aportada por una chirp será mucho mejor que la aportada por un pulso CW.

También definimos como resolución en velocidad la anchura de la zona de incertidumbre:

$$\delta v = \frac{\lambda}{2\tau} \quad (18)$$

Que será la misma que tenemos para los pulsos CW

### *4.3.3 Pulsos con fase codificada*

Mediante una modulación en fase empleamos códigos que generalmente suelen ser binarios, lo que significa que existen dos fases diferente, 0º y 180º. Los códigos binarios son secuencias de 0 y 1 ó 1 y -1 donde vienen a significar lo mismo. En especial, los códigos Barker son un tipo de secuencia binaria óptima muy empleada en la compresión de pulsos radar, dando unos lóbulos pequeños en el proceso de la autocorrelación. Aunque tampoco podemos hablar del mejor método para compresión de pulsos ya que el máximo de longitud es 13 por lo que se limita la compresión.

Los códigos Barker se rigen según la siguiente tabla:

| Code Length | Barker Code |
|---|---|
| **2** | [+1, -1] |
| **3** | [+1, +1, -1] |
| **4** | [+1, +1, -1, +1] |
| **5** | [+1, +1, +1, -1, +1] |
| **7** | [+1, +1, +1, -1, -1, +1, -1] |
| **11** | [+1, +1, +1, -1, -1, -1, +1, -1, -1, +1, -1] |
| **13** | [+1, +1, +1, +1, +1, -1, -1, +1, +1, -1, +1, -1, +1] |

*Tabla 1. Codificación Barker*

En el siguiente ejemplo podemos observar un ejemplo de la codificación en fase utilizando el código Barker de 13 elementos:

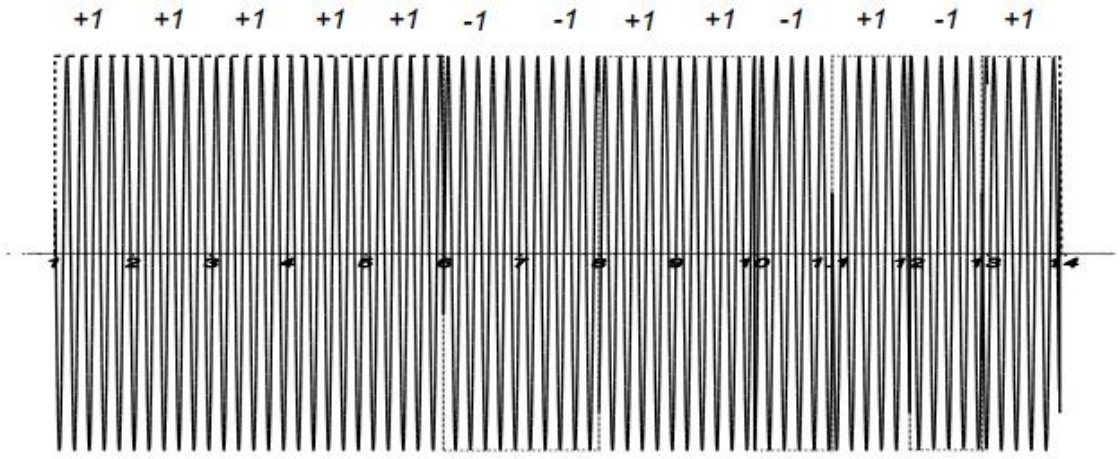



*Figura 4 Ejemplo codificación en fase.*

## 4.4 Software

Los lenguajes interpretados son aquellos donde sus instrucciones o código fuente, está escrito en un lenguaje de alto nivel.

Será entonces traducido por un intérprete a un lenguaje entendible para la maquina paso a paso. Siendo una ejecución generalmente más lenta y costosa. El proceso es repetido cada vez que el programa ejecuta el código.

Los lenguajes interpretados permiten un tipado dinámico de datos, en otras palabras, no será necesario inicializar una variable con un determinado tipo de dato, sino que esta puede cambiar su condición al dato que almacene entre otras características.

Como principal desventaja, nos encontramos con que el tiempo de ejecución de estos lenguajes será mayor. Al tener que ser traducido al lenguaje maquina con cada ejecución, este proceso será más lento que en los lenguajes compilados. Sin embargo, también nos encontramos con lenguajes que poseen una máquina virtual que hace una traducción al lenguaje intermedio, de forma que al traducirlo a lenguaje de bajo nivel le tomará menos tiempo.

### *4.4.1 Python*

El lenguaje de programación Python[3] creado a finales de los 80 por Guido van Rossum, ha llegado a la actualidad como uno de los más conocidos y utilizados debido a sus características. Python nació como Open Source siendo modificado para poder funcionar en diferentes plataformas como son Windows, Linux, Macintosh, Solaris y más.

Como otros lenguajes líderes, es orientado a objetos, combinando así datos y funcionalidades. Es un lenguaje de alto nivel, por lo que no necesitamos preocuparnos por detalles como el manejo de la memoria. Como veremos a continuación en el ejemplo en GNU Radio, es un lenguaje incrustable, pudiéndose insertar el código en Python dentro de otros programas como C/C++, abriéndonos grandes facilidades dentro del scripting.

Es un lenguaje con extensas librerías, tipos de datos, funciones incorporadas en el propio lenguaje que ayudan a realizar tareas comunes sin necesidad de tener que programarlas nosotros. Estas librerías nos permiten realizar tareas como expresiones regulares, pruebas, procesos, generar documentos, interacción con bases de datos, servicios web, ftp, correo electrónico, etcétera.

Su sintaxis es muy clara y visual, ya que su identación se realiza mediante tabulados, estos al ser de carácter obligatorio, hacen que el código se encuentre muy organizado y se pueda entender por otro programador de forma rápida y sencilla, debido a la similaridad entre programas.

*Figura 5. Logo oficial Python*

[3] https://www.python.org/

### *4.4.2 GNU Radio*

GNU Radio[4] es una herramienta software de desarrollo libre y open-source [5]que proporciona al usuario una serie de bloques de procesamiento de señal implementables en SDR[6]. Puede utilizarse con hardware externo de bajo coste para la creación de Radios Definidas por Software (SDR), o sin hardware, debido a la inclusión de diferentes entornos de simulación. Es ampliamente usado en investigación, industria, temas académicos, ambientes gubernamentales o como hobbie.

Los diferentes proyectos de GNU Radio son construidos mediante el entorno gráfico GNU Radio Companion o mediante lenguaje de programación Python, donde en caso de buscar un rendimiento mayor se deberá implementaren lenguaje C++, como también pueden ser sus librerías. De este modo se permite desarrollar sistemas de radio en tiempo real y de un rendimiento alto mediante el uso de un entorno de desarrollo de aplicaciones simple y rápido.

Uno de los grandes avances de la plataforma es la posibilidad de desarrollar de forma libre nuevos algoritmos de procesamiento de señal sin la necesidad de utilizar hardware real.

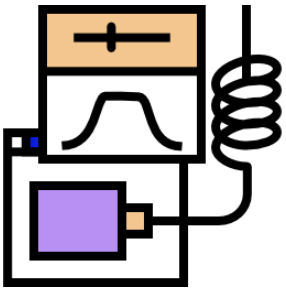


*Figura 6. Logo oficial GNU Radio Companion*

En primer lugar, comenzaremos hablando de la plataforma GNU Radio Companion[7], este se trata de un lenguaje de programación visual de código libre para procesamiento de señales utilizando las librerías de GNU Radio conocidas por las siglas GRC.

El entorno gráfico permite el desarrollo de aplicaciones sin el conocimiento previo de ningún lenguaje de programación como pueden ser Python o C++. Esto es posible debido a sus esquemas de bloques.

Una de las funcionalidades principales de GNU Radio es su facilidad de uso. Esto es conseguido con la utilización de grafos en forma de bloque (procesamiento de datos) y las uniones, que representan el flujo de información entre cada bloque.

La implementación de cada bloque viene en lenguaje C++, y la llamada a cada uno ha sido realizada en Python. De este modo, al utilizar Python, conseguimos una serie de ventajas, como es la agilidad que proporciona ser un lenguaje interpretado que no necesita compilación. Así en su desarrollo no será necesaria la implementación de los bloques ya que simplemente constará de enlazar un bloque con otro encaminando el flujo de datos hasta la salida, siendo transparente el funcionamiento de cada bloque.

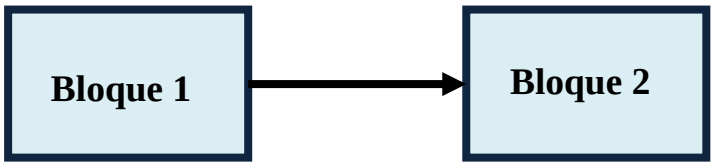



*Figura 7. Diagrama de bloques*

[4] https://www.gnuradio.org/
[5] https://es.wikipedia.org/wiki/Código_abierto
[6] https://en.wikipedia.org/wiki/Software-defined_radio
[7] https://wiki.gnuradio.org/index.php/Main_Page

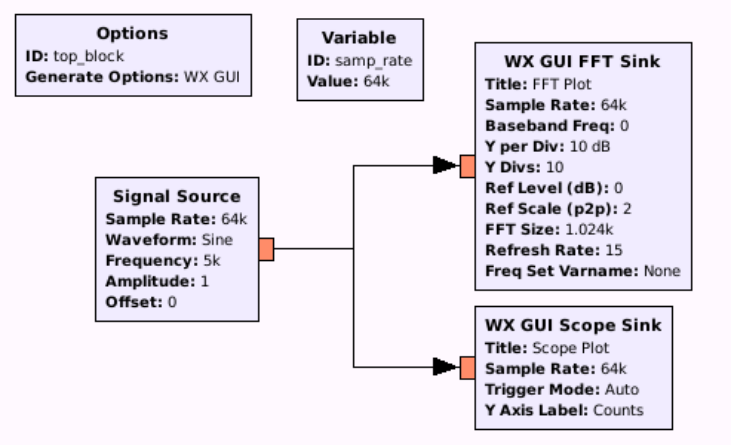
Cada bloque se caracteriza por el número de puertos de entrada/salida y el tipo de dato, que en cierto tipo de bloque podrá ser configurado y se distinguen mediante los siguientes colores:

- Tipo Int / Short → Color Verde
- Tipo Complex → Color Azul
- Tipo Float → Color Naranja
- Tipo Byte → Color Rosa

GNU Radio, incluye de forma predefinida en sus librerías más de 150 bloques destinados para diferentes finalidades como puede ser Audio, operadores Booleanos, generadores de señal, instrumentación de medida, etc.

Un ejemplo básico de funcionamiento que nos permitirá entender mejor la plataforma es el siguiente:

*Figura 8. Ejemplo diagrama.*




En la figura podemos observar un generador de señales básico que nos permite configurarlo con diferentes formas de onda como pueden ser una constante, seno, cose, cuadrada, triangular o diente de sierra. En este caso, se ha configurado para generar un seno a una frecuencia de 5 kHz y a una tasa de 64000 muestras/segundo, muy por encima de la frecuencia de Nyquist.

Conectado al generador tenemos dos GUI (Interfaces de usuario) que nos permitirán visualizar la señal generada en tiempo, gracias al WX GUI Scope Sink, y en frecuencia, gracias al WX GUI FFT Sink:

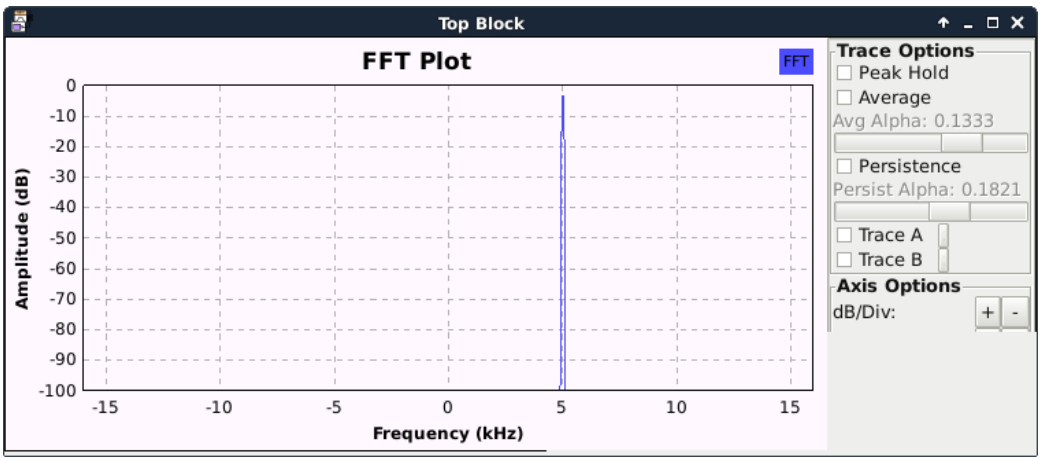



*Figura 9. Representación de un seno en frecuencia.*

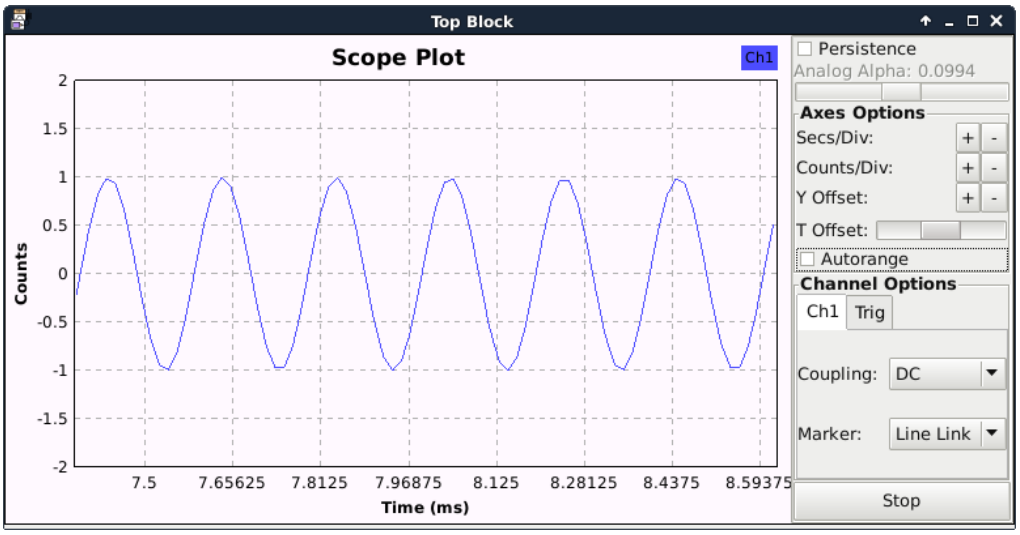



*Figura 10. Representación de un seno en tiempo.*

Como podemos observar, se trata de una interfaz user-friendly, donde sin necesidad de conocer ningún lenguaje de programación, hemos conseguido generar y visualizar un seno en un tiempo muy inferior que en otras plataformas. Una de las ventajas que este método aporta es la visualización, ya que podemos ver la señal como si la tuviésemos conectada a un osciloscopio y ver como varía en tiempo real.

### Creación de bloques

El entorno GNU Radio al ser open-source y de uso libre, no solo permite la utilización de los bloques que ya vienen en la librería, sino que existe una gran cantidad de aportaciones de la comunidad incluyendo bloques más especializados en diversos campos. Este tipo de bloques son llamados Out-Of-Tree Modules[8], con ellos conseguimos añadir funcionalidades adicionales al código principal. Para facilitar su creación existen librerías como son *gr-modtool* (Disponible en https://github.com/mbr0wn/gr-modtool) que nos ayuda en la creación de los bloques base para a continuación poder realizar modificaciones sobre ellos para que realicen la función deseada. En este caso será necesario tener conocimientos básicos sobre programación en “C” y “Python”.

[8] https://wiki.gnuradio.org/index.php/OutOfTreeModules

# Capítulo 5. Desarrollo del proyecto

## 5.1 Interfaz

Los primeros pasos del proyecto se centrarán en la interfaz. Como se ha comentado anteriormente, el programa será utilizado como programa de test de equipos que recibirán las señales que nosotros generemos para poder ser procesadas. Los programas de test se caracterizan por pasar por una gran cantidad de manos, debido a que se trata de programas muy polivalentes. Por tanto, nos debemos centrar en realizar una interfaz lo más sencilla posible donde cualquier persona que vaya a realizar un uso del programa sea capaz de utilizar cada una de sus funciones sin ayuda del creador.

En caso de Python no disponemos de ninguna gran herramienta para la implementación de una interfaz, de modo que nos encomendaremos a módulos externos que nos permitan un fácil desarrollo.

El primer módulo escogido es Tkinter[9]. Esta versión viene incluida en Python, es fácil de usar y se caracteriza por ser multiplataforma. Es el estándar en la programación GUI en Python. Ofrece una buena integración visual muy completa, un gran número de widgets y una personalización de la estética del programa.

El siguiente módulo escogido para el desarrollo del programa es PyQt4[10]. Se trata de un binding (adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita) de la biblioteca gráfica Qt para el lenguaje de programación Python. También incluye una gran cantidad de herramientas extras a las proporcionadas por Tkinter.


*Figura 11. Logo Oficial PyQt.*

[9] https://wiki.python.org/moin/TkInter
[10] http://pyqt.sourceforge.net/Docs/PyQt4/index.html

Por último, utilizaremos un módulo destinado a representar cada una de las señales. En este caso nos referenciamos a Matplotlib[11], una librería destinada para plotting en 2D en Python. Nos permitirá generar fácilmente gráficas, histogramas o diagramas espectrales entre otros. Nos ofrece la posibilidad de modificar el estilo de dibujado de las líneas, el estilo de las fuentes, propiedades de los ejes, etc, todo ello orientado a una modificación similar a la de MATLAB ayudando así a los usuarios. Gracias a su uso podremos visualizar cada señal generada para realizar una rápida comprobación de si se ha generado correctamente.


*Figura 12. Logo oficial Matplotlib*

De este modo comenzaremos a realizar la base de la interfaz. Se le llamará "canvas", lienzo en español. Además, añadiremos 3 opciones para comenzar a utilizar el programa. Ellas serán "Radar Generator" donde se introducirán los valores para crear cada señal, "Signal viewer" donde podremos ver las señales generadas y una última pestaña útil para cuando se quiera implementar algunos ajustes llamada "Settings".

El resultado obtenido es el siguiente:

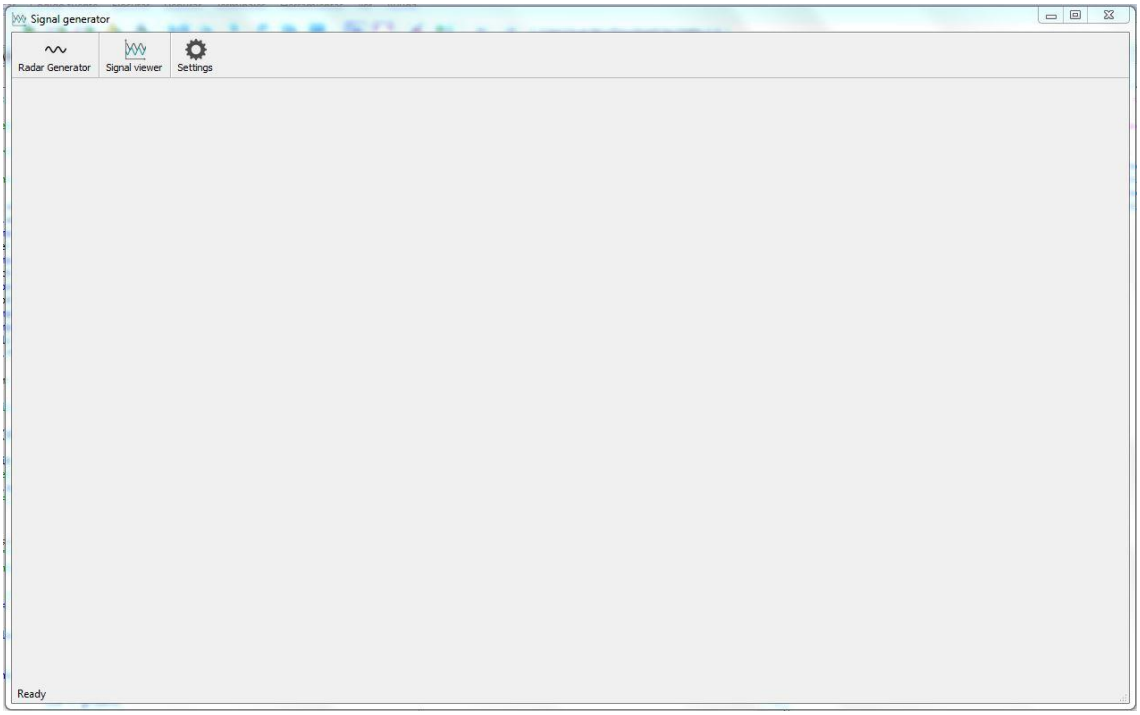



*Figura 13. Interfaz principal*

En este punto deberemos escoger que pestaña escoger, donde generalmente se comenzará a utilizar el Radar Generator.

[11] https://matplotlib.org/

El siguiente paso será pues desarrollar la interfaz para la pestaña “Radar Generator”

El diseño constará en una tabla donde poder introducir los datos deseados de cada señal a generar, ya que nuestra finalidad es poder generar más de una señal, tal y como nos encontraríamos en un entorno real. La tabla será generada de modo que se puedan introducir 10 señales. De modo que la tabla tendrá una altura de 10 filas. Ahora falta determinar el número de columnas. Las columnas nos permitirán introducir los datos de entrada de cada pulso. Los campos de la tabla decididos son:

- Modo: permite escoger el tipo de modulación.
- F (MHz): para introducir a la frecuencia que se generará el pulso
- PRI (us): Periodo de repetición de la señal
- PW(us): Pulse Width. Se indicará el ancho del pulso.
- Power (dBm): potencia de salida de la señal
- Offset (%): Desplazamiento respecto al punto inicial.
- BW (MHz): BandWidth. Implementado para ciertos tipos de señales como Chirp
- Barker: para introducir modulación Barker en caso deseado
- Duty Cycle (%): indica automáticamente el ciclo de trabajo de cada señal
- Modo de radar: Permite modelar la forma de la señal en consonancia con radares modernos que tienen funciones de tracking.

En las opciones de Modo y Modo de Radar introduciremos dos widgets desplegables facilitando así la programación posterior y ayudando al usuario a escoger cada opción. Así que en el primer widget introduciremos las formas de onda o tipos de modulación de cada pulso. Siendo estos los siguientes:

- None: Para cuando no queremos generar ninguna señal en esa fila.
- Continuous wave (CW): generar onda continua
- Pulse radar: generación de pulso simple
- Frecuency hopping
- PW Agility
- PW + FW: Fusión de PW Agility y Frequency Hopping
- Chirp
- Fast Hopper

En el caso de Modo de Radar tenemos 3 modos:

- Modo Fijo: Se transmite continuamente la misma señal
- Modo Search
- Modo Track

Se han introducido los modos más comunes en los entornos radar dentro de la Guerra Electrónica. Más adelante se explicará cada modo, por qué se genera cada uno con cada forma y su utilidad.

Para finalizar, introduciremos un apartado en el que tendremos la opción de guardar la tabla en archivos CSV[12] que se permiten leer y modificar fácilmente mediante el programa Microsoft Excel[13]. De este modo podremos guardar una serie de señales predefinidas para tenerlas listas en un tiempo menor.

Todo ello ha sido crear en 3 pasos.

Un primer paso para guardar la tabla en el archivo CSV que consta de las siguientes instrucciones:

1) Primero realizamos una comprobación fila a fila de cuantas filas han sido rellenadas de modo que obtengamos el número de señales a generar facilitándonos así los pasos posteriores.
2) El siguiente paso constará de recorrer la tabla recuadro a recuadro de modo que se vaya guardando cada valor en una matriz del mismo tamaño que la tabla con señales.
3) Por último, se realiza el guardado de la matriz en la carpeta especificada y en un fichero en el que tendremos la opción de decidir su nombre.

```
savePath = "./SignalReports/" + self.textboxwrite.text()

f = file(savePath, 'w')
savetxt(f,matrixtoSave,delimiter=",")
f.close()
```

*Código 1. Guardado de la matriz*

El otro proceso constará en leer el archivo CSV que se explicará a continuación:

1) El primer paso se trata de encontrar el archivo que se intenta abrir y en caso contrario, de avisar al usuario de que ese archivo no ha sido encontrado.

```
try:
    matrixRead = genfromtxt("./SignalReports/" + self.textboxread.text(),delimiter=',')
except:
    self.statusBar().showMessage('Unable to find the file ' + self.textboxread.text())
    return 0
```

*Código 2. Comprobación de la existencia del archivo a leer*

2) El segundo paso constará de leer el archivo y guardar cada columna en una variable tal como se muestra a continuación. Además, hallaremos y guardaremos el número de señales, cuyo número nos servirá para el siguiente paso.

```
Modo = matrixRead[:,0]
F = matrixRead[:,1]
PRI = matrixRead[:,2]
PW = matrixRead[:,3]
Power = matrixRead[:,4]
Offset = matrixRead[:,5]
BW = matrixRead[:,6]
barker =matrixRead[:,7]

nTotal = Modo.size
self.table.setRowCount(10)
```

*Código 3. Lectura del archivo CSV*

[12] https://es.wikipedia.org/wiki/Valores_separados_por_comas
[13] https://es.wikipedia.org/wiki/Microsoft_Excel

3) Por último, rellenamos la tabla con los valores que recientemente hemos guardado en variables. El valor obtenido del número de señales nos indicará el tamaño del bucle destinado a recorrer la tabla. La dificultad se centrará en colocar cada widget en su posición correcta, debido a que no se basa en insertar simplemente un número, sino que debemos identificar el tipo de señal y modificar el widget. Exactamente se deberá hacer para la pestaña "Modo de Radar"

Con la intención de ayudar al usuario con un conocimiento menor sobre este tipo de señales, se decidió pasar a programar un sistema de ayuda. Esto es debido a que no todas las columnas son útiles para todo tipo de señal. Por tanto, para no caer en el error de intentar modificar columnas que no tienen ninguna función, se ha decidido programarlas de modo que sean bloqueadas e inmodificables.

```
for i in range(10): #Comprobamos que valores de la tabla no se modifiquen
    if Modo[i]==1: #CW
        self.table.setItem(i,3,QtGui.QTableWidgetItem('                    0'))
        self.table.setItem(i,5,QtGui.QTableWidgetItem('                    0'))
        self.table.setItem(i,6,QtGui.QTableWidgetItem('                    0'))
    if Modo[i]==4: #Pulse Radar o PW Agility
        self.table.setItem(i,6,QtGui.QTableWidgetItem('                    0'))
    if Modo[i]==1 or Modo[i]==3 or Modo[i]==4 or Modo[i]==5: #Bloqueo Barker
        self.table.setItem(i,7,QtGui.QTableWidgetItem('                    0'))
```

*Código 4. Bloqueo de columnas*

Otro problema es el mal uso de la columna Barker. Se trata de una columna de la que ya indicaremos y explicaremos más tarde su uso. El problema se halla en que no es posible introducir cualquier número, ya que solo existen ciertos tipos de modulación Barker. De modo que ha sido programado tal que:

```
if self.table.item(i,7):
    if float(self.table.item(i,7).text()) ==0:
        self.table.setItem(i,7,QtGui.QTableWidgetItem(str(1)))
    if float(self.table.item(i,7).text()) > 5: #Comprobamos que se introduzca bien el barker
        if float(self.table.item(i,7).text()) < 7:
            self.table.setItem(i,7,QtGui.QTableWidgetItem(str(7)))
        elif float(self.table.item(i,7).text()) < 11 and float(self.table.item(i,7).text()) !
            self.table.setItem(i,7,QtGui.QTableWidgetItem(str(11)))
        elif float(self.table.item(i,7).text()) > 11 and float(self.table.item(i,7).text())
            != 7 and float(self.table.item(i,7).text()) != 11:
            self.table.setItem(i,7,QtGui.QTableWidgetItem(str(13)))
```

*Código 5. Comprobación de Barker*

El resultado es el siguiente:

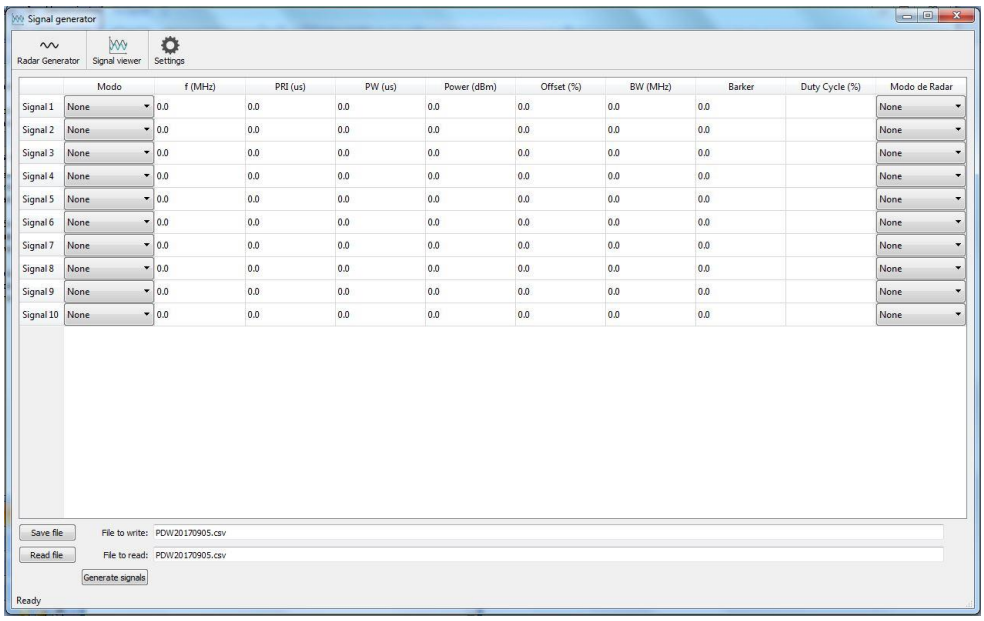



*Figura 14. Interfaz Radar Generator*

Por último, nos queda por desarrollar la pestaña del Signal Viewer. En esta pestaña nuestro objetivo será poder visualizar cada señal generada. Por tanto, introduciremos 3 gráficas en la pestaña. La primera será la señal generada en función del tiempo. Así se podrá ver la forma en que haya sido generada. La segunda gráfica se realizará una FFT. Nos permitirá ver si la señal ha sido generada con la frecuencia correcta y en caso de señales complejas como Chirp, ver la evolución en frecuencia. En la última gráfica se representará el Modo de Radar. Así podremos observar la forma que se producirá.

El lienzo sin ninguna señal se verá como:

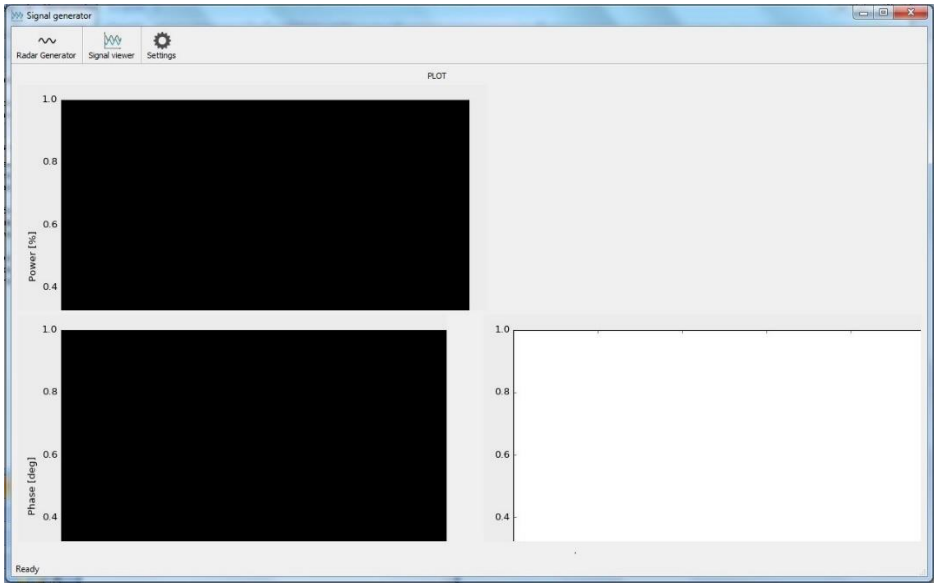



*Figura 15. Interfaz Signal Viewer*

## 5.2 Generación de señales

En este apartado nos centraremos en cómo ha sido generado cada tipo de señal y el resultado obtenido.

En este apartado también toca hablar de módulos externos que nos has facilitado mucho el trabajo. El primer caso, se trata de uno de los módulos más conocidos y utilizados. Hablamos de Numpy[14], un paquete que incluye computación científica básica como puede ser operaciones con arrays de N-dimensiones, funciones de algebra lineal, transformaciones de Fourier entre otras competencias.


*Figura 16. Logo Oficial Numpy*

En nuestro caso las instrucciones más utilizadas han sido las siguientes:

***numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)***

La instrucción *numpy.linspace* devuelve números uniformemente espaciados en un intervalo especificado. Devuelve un número de muestras uniformemente espaciadas, calculadas sobre el intervalo [start, stop].

***numpy.random.rand(d0, d1, ..., dn)***

En este caso, la instrucción *numpy.random.rand* crea un array de muestras aleatorias de una distribución uniforme sobre [0, 1]. Los valores incluidos en el paréntesis serán generalmente el número de filas y el número de columnas del array.

***numpy.concatenate((a1, a2, ...), axis=0)***

La anterior instrucción llamada *numpy.concatenate* une secuencias de arrays a lo largo de un eje. La forma de unir los arrays será indicada en *axis*.

***numpy.arange([start, ]stop, [step, ]dtype=None)***

La instrucción *numpy.arange* es una de las más utilizadas debido a su gran utilidad. Se encarga de generar valores entre el intervalo semi-abierto [start, stop[ de manera que el intervalo incluirá start pero excluirá stop. Tambien permite la opción de incluir un paso, saltándose valores entre start y stop.

[14] http://www.numpy.org/

Otro módulo externo no tan utilizado, pero si de mucha importancia es SciPy[15]. Se trata de una biblioteca open source de herramientas y algoritmos matemáticos para Python. Incluye módulos para optimización, algebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen. También está dirigida al mismo tipo de usuarios que utilitarios de aplicaciones como Matlab, GNU Octave o Scilab.

En nuestro caso ha sido utilizada la función *numpy.fft*, la transformada de Fourier más simple de las que nos puede ofrecer Scilab, siendo esta de solo una dimensión. Con ello conseguiremos visualizar la señal en función de la frecuencia.


*Figura 17. Logo oficial SciPy*

El último módulo utilizado ha sido *math*[16]. Las funciones que nos han facilitado el trabajo han sido:

- *math.ceil(x)*: Devuelve el valor entero superior de *x* como tipo float.
- *math.floor(x)*: Devuelve el valor entero inferior de *x* como tipo float.

Incluso con la inclusión de varios módulos, nos ha sido necesario incluir funciones propias como es el cálculo del mínimo común múltiplo que nos ha sido necesario para el desarrollo del programa:

```
def lcm(self,x, y):
        """This function takes two
        integers and returns the L.C.M."""
        # choose the greater number
        if x > y:
            greater = x
        else:
            greater = y

        while(True):
            if((greater % x == 0) and (greater % y == 0))
                lcm = greater
                break
            greater += 1
        return lcm
```

*Código 6. Función para hallar el mínimo común múltiplo*

[15] https://www.scipy.org/
[16] https://docs.python.org/2/library/math.html

Así pues, podemos pasar a hablar sobre la generación de señales. Comenzaremos hablando sobre el proceso general que se realiza cuando cualquier señal es generada. Se ha optado por generalizar lo más posible cada paso para primero, compactar lo más posible el número de líneas de código haciéndolo así más ligero y sencillo de leer, y segundo, poder tener la facilidad de poder incluir nuevos tipos de señales sin tener que realizar grandes modificaciones en el código que afecten a otros tipos de señales.

El primer paso constará guardar la tabla en variables globales. Esto se debe a que, en Python, las variables creadas dentro de una función, no serán reconocidas fuera de esa función. En caso de querer utilizar la misma variable en otra función o querer transportar su contenido a otra función, las deberemos declarar tal que:

Global + [Nombre variable]

Una vez recopilada toda la tabla dentro de variables globales, pasamos a dibujar la pestaña de Signal Viewer. Donde a partir de este punto solo necesitaremos generar las señales. Para ello, inicializaremos las siguientes variables:

- Frecuencia de muestreo: fs = 1.25e9
- Frecuencia mínima: fmin = 1000.0e6
- Número de señales: nTotal = 0
- Modmax = -100

Necesitaremos saber en primera instancia el número de señales a generar, siendo necesario para el siguiente paso, que consistirá en calcular el número de muestras.

```
for i in range(10):
 if Modo[i] !=0:
     nTotal = nTotal +1
```

*Código 7. Cálculo del número de señales*

Una vez sabemos el número de señales, pasamos a calcular el número de muestras que va a ocupar la señal o el conjunto de ellas. El cálculo dependerá del tipo de señal como se verá a continuación:

```
def calc_Ns(self,Modo,PRI,fs,nTotal,track):
    N_AWG = 8
    maxAWG = 8e6
    #Ns = fs  * 1e-6
    lcm_PRI = 1
    flagHopping = 0
    print track
    for i in numpy.arange(0,nTotal):
        if Modo[i]==2 or Modo[i]==3 or Modo[i]==4 or Modo[i]==6:
            lcm_PRI = self.lcm(lcm_PRI,PRI[i]) #minimo común múltiplo PRI
        if Modo[i]==3 or Modo[i]==4 or Modo[i]==5 :
            flagHopping = 1

    if flagHopping==1:
        Ns = fs * lcm_PRI * 1e-6
        Njumps = floor(maxAWG/Ns)
        if maxAWG/Ns < 1:
            print('Incompatibilidad de PRI: no habran frequency hopping\n')
        Ns = round(Njumps*N_AWG*floor(Ns/N_AWG)/2)
    else:
        Ns = fs*lcm_PRI * 1e-6
        Ns = N_AWG*ceil(Ns/N_AWG)
    return Ns
```

*Código 8. Cálculo número muestras*

Se pondrá para empezar dos condiciones que el programa Radar Generator pueda funcionar con el Generador de señales, cuyo aparato tiene una tasa de muestras de entradas de 8 Megamuestras. Para calcular el número de muestras se obtendrá primero el mínimo común múltiplo para el caso en que tengamos más de una señal, sepamos el número de señales necesarias hasta completar el ciclo. Por último, teniendo en cuenta la frecuencia de muestro, el mínimo común múltiplo entre PRI y el valor aportado, se realiza el cálculo de las muestras.

Así continuamos recopilando más datos necesarios como son:

```
PRI_int = PRI[i]/1e6
PW_int = PW[i]/1e6
track_int = track[i]
Freq = F[i]*1e6 - fmin*1e6 + offsetPSGcal
Pot = Power[i] - modmax
print Pot
N = ceil(Ns/fs/PRI_int)
print('N= '+ str(N))
fase_ini=Offset[i]/100.0
BW_int=BW[i]*1e6
barker_int=barker[i]
```

*Código 9. Obtención de datos para el cálculo*

### *5.2.1 Pulsos simples*

```
signali = self.genera_pulsos_radar(fs,Freq,1/PRI_int,PW_int,Pot,N,fase_ini,BW_int,barker_int,0)
signal = signal + signali[0:Ns]
```

*Código 10. Cálculo pulsos*

Para poder realizar el cálculo, se utilizarán dos variables, una principal donde se almacenará la señal principal y una segunda variable temporal que se irá guardando cada señal calculada. Como podemos observar, accederemos a la función genera_pulsos_radar para realizar el cálculo. En esta función realizaremos los cálculos previos como son la variable t, la frecuencia inicial, aunque hablaremos más tarde de ella debido a ser necesaria para el CHIRP, y la potencia de salida.

```
t = linspace(1, PW*fs, PW*fs)/fs
f_ini = f_if - Dop_BW/2
A = pow(10.0,(A/10.0))
secBarker =[]

#reserva de espacio para la señal
signal=zeros(N*(len(t) + round(fs*(1/PRF-t[-1]))),dtype=numpy.complex)

    f=f_ini
    phase0=zeros(len(t))
for i in range(int(N)):
    Nini = i*round(fs*(1/PRF))
    pulse = A*exp(1j*(2*pi*t*f + secBarkerN[0:len(t)]*pi+phase0))


[Nini:(Nini+len(pulse))]=pulse   # Pulso + silencio
len_pulse = len(pulse)
toRoll = int(floor(fs*(1/PRF-t[-1])*fase_ini))
signal = roll(signal,toRoll)
Nceros = len(signal)-len_pulse
#plot(real(signal))
#show()
signaltoSave = concatenate((signal.real,signal.imag))
```

*Código 11. Generación de un pulso*

La señal se genera primero creando un vector de ceros del tamaño del número de señales a generar multiplicado por el PRI (Pulse Repetition Interval). A continuación, centraremos la frecuencia del pulso en la frecuencia indicada. Para generar cada señal, hallaremos la variable Nini que se trata del punto donde insertar la señal en el vector signal. Tan solo nos quedará crear el pulso con la formula mostrada anteriormente y añadirla en el vector anteriormente creado de ceros. Para devolverla a la señal principal, la concatenaremos primero la parte real seguida de la parte imaginaria.

Para comprobar su funcionamiento introduciremos los siguientes valores en la tabla:

| **Modo** | **F(MHz)** | **PRI(us)** | **PW(us)** | **Power** | **Offset** | **BW** | **Barker** | **Modo Radar** |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 1 | None |

*Tabla 2. Datos para pulso simple*

Y obtenemos el siguiente resultado:

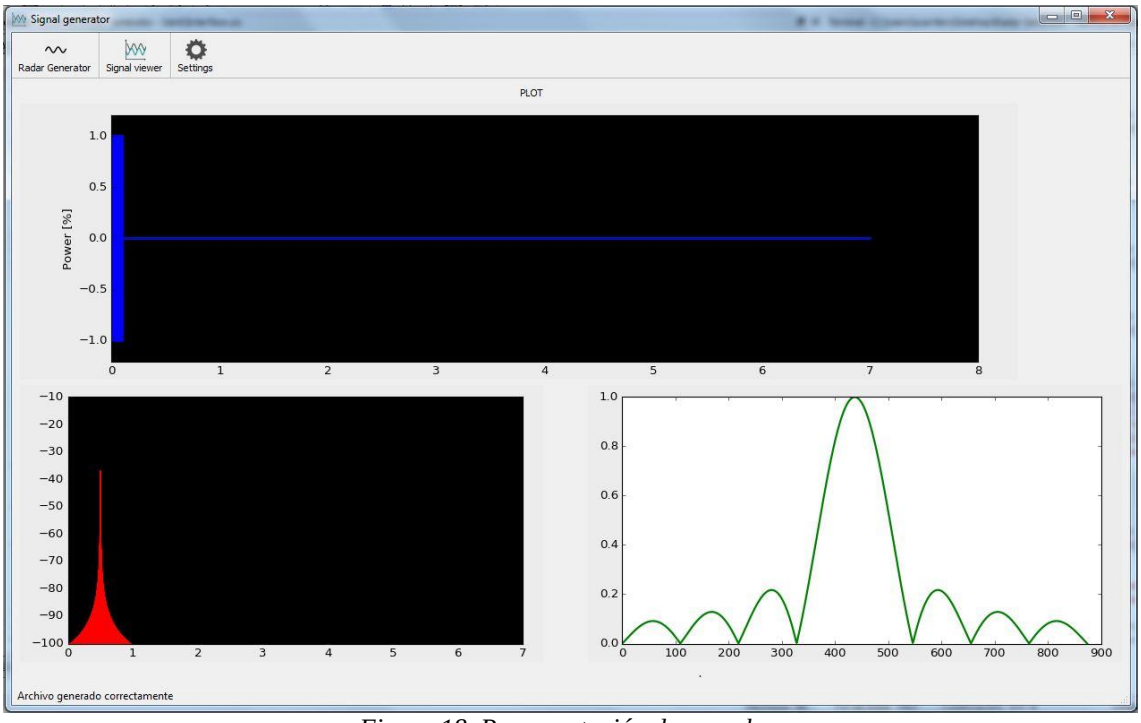



*Figura 18. Representación de un pulso.*

Podemos observar la señal en el tiempo, donde confirmamos que se ha generado con el PRI indicado, el PW que nosotros hemos marcado y la potencia deseada. Además, mediante la representación en frecuencia, se confirma que se ha generado el pulso a 500 MHz.

La siguiente prueba se centrará en generar más de una señal. Para ellos introduciremos los siguientes datos en la tabla:

| Modo | F(MHz) | PRI(us) | PW(us) | Power | Offset | BW | Barker | Modo Radar |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 1 | None |
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 20.0 | 0.0 | 1 | None |

*Tabla 4. Datos para dos pulsos.*

Para ser capaces de diferenciar cada pulso hemos añadido a la segunda señal un desfase del 20%, así comprobando también el buen funcionamiento del Offset. Obtenemos las siguientes gráficas:

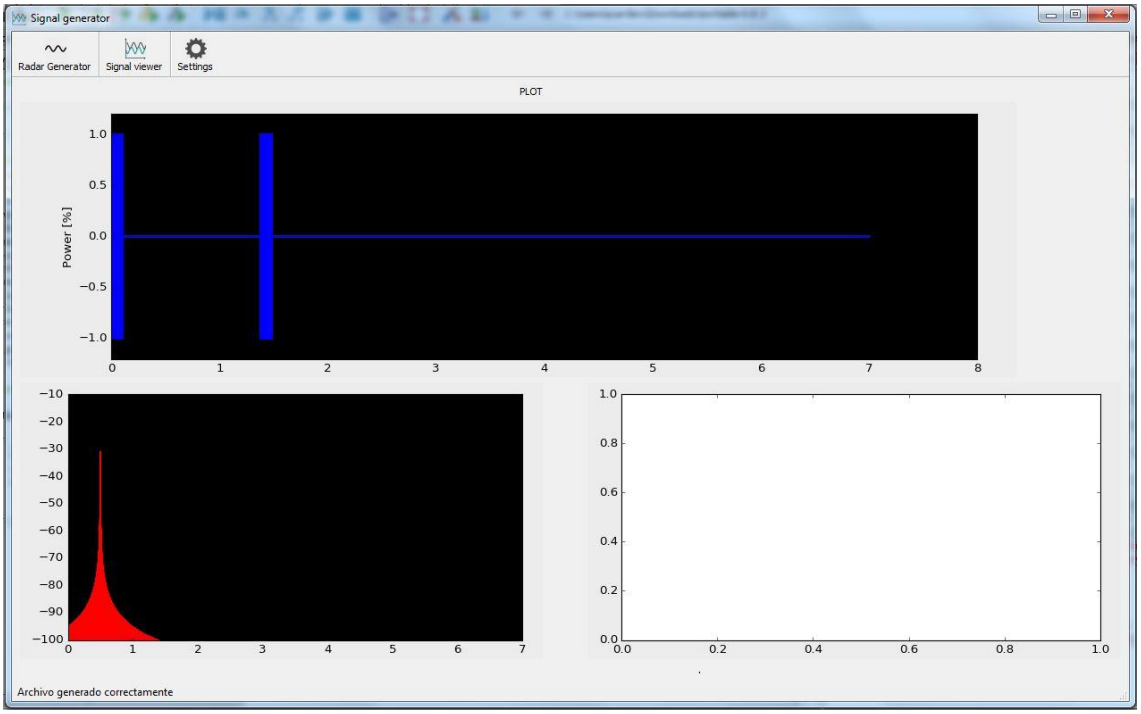



*Figura 19. Representación de dos pulsos.*

Vemos que ambos pulsos se han generado en la misma frecuencia, como nosotros hemos indicado. Además, comprobamos así el uso correcto de la columna Offset, debido al correcto desplazamiento del segundo pulso. Esto, como indica la tabla se puede realizar hasta para diez pulsos. Para el testing se trata de una gran funcionalidad debido a la capacidad de probar el sistema en la capacidad de poder distinguir entre pulsos muy cercanos en tiempo o poder distinguir pulsos muy cercanos en frecuencia, funcionalidad que se probará a continuación.

Como ya se ha dicho, pasaremos a generar dos pulsos en distintas frecuencias. Para ello se introducirán los siguientes datos:

| Modo | F(MHz) | PRI(us) | PW(us) | Power | Offset | BW | Barker | Modo Radar |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 1 | None |
| **Pulse Radar** | 2500.0 | 700.0 | 10.0 | 10.0 | 20.0 | 0.0 | 1 | None |

*Tabla 4. Datos para doble pulso en distinta frecuencia.*

Que nos ofrece como resultado:

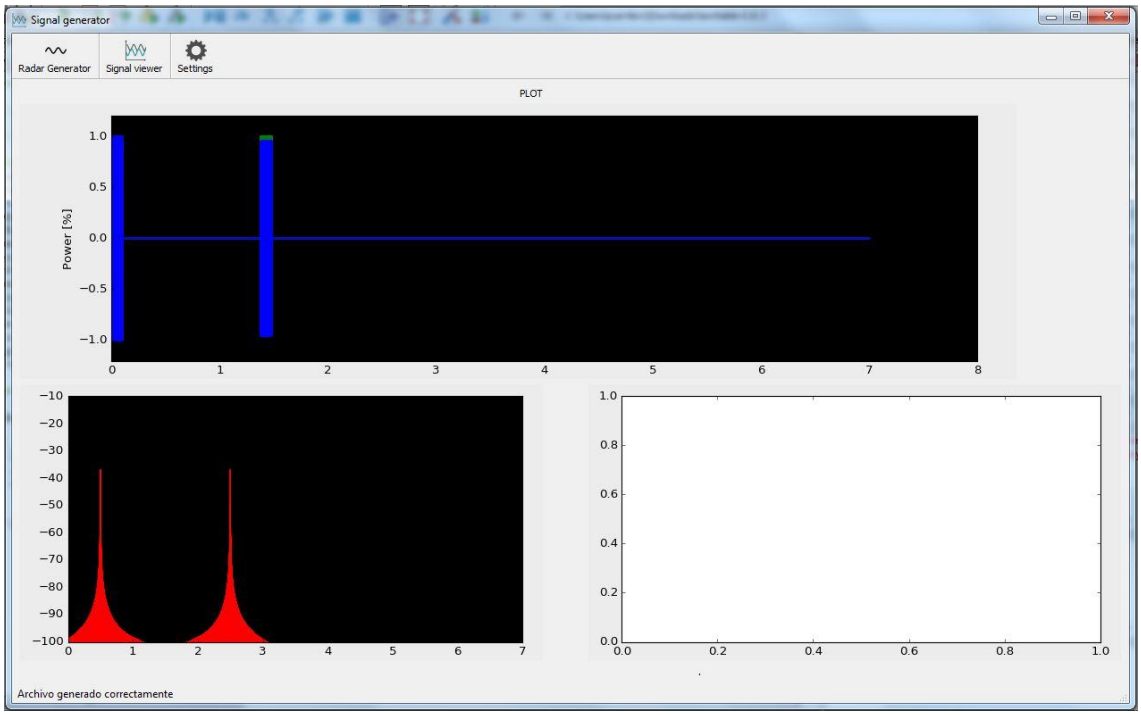



*Figura 20. Representación doble pulso a doble frecuencia.*

Podemos ver pues que se produce una buena representación de cada una de las señales.

Una vez comprobado la posible representación de cada forma de los pulsos simples, pasamos a representar pulsos modulados uno a uno para comprobar su buen funcionamiento además de explicar cómo han sido generadas.

### *5.2.2 Pulso CHIRP*

Se trata de un pulso con una modulación donde la frecuencia va en aumento. Para poder utilizarla en nuestro programa, tendremos la opción para cuando escojamos el Modo: Pulse Radar donde deberemos modificar la columna BW. Así pues, indicaremos en la primera columna la frecuencia en la que se centrará el pulso, seguido de ello en la columna con el nombre BW introduciremos el ancho de banda que va a variar el pulso sobre la frecuencia central.

Para poder generar este tipo de señal se deben pasar por los siguientes pasos:

```
f_ini = f_if - Dop_BW/2

elif abs(Dop_BW) > 0: #Doppler - CHIRP
    f=f_ini + t*Dop_BW/t[-1]/2
    phase0=zeros(len(t))
```

*Código 12. Cálculo vector frecuencia.*

Como vemos, el programa pasará por dos procesos esenciales. Primero fijar la frecuencia inicial donde posteriormente se generará el pulso. Y a continuación, cuando se cumpla el "if" con la condición de que la columna BW sea mayor que cero, crearemos el vector f, este será resultado de la frecuencia inicial y la suma del vector tiempo mostrado antes. Como resultado tendremos un vector del mismo tamaño que el vector de tiempo. Es nuestro objetivo debido a que cuando vayamos a realizar el cálculo del pulso, se multipliquen los vectores de tiempo y frecuencia para que muestra a muestra la frecuencia vaya en aumento. Por tanto, nos quedará probarlo, donde introduciremos los siguientes parámetros:

| **Modo** | **F(MHz)** | **PRI(us)** | **PW(us)** | **Power** | **Offset** | **BW** | **Barker** | **Modo Radar** |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 10.0 | 200.0 | 1 | None |

*Tabla 5. Datos pulso Chirp.*

Ofreciendo el siguiente resultado:

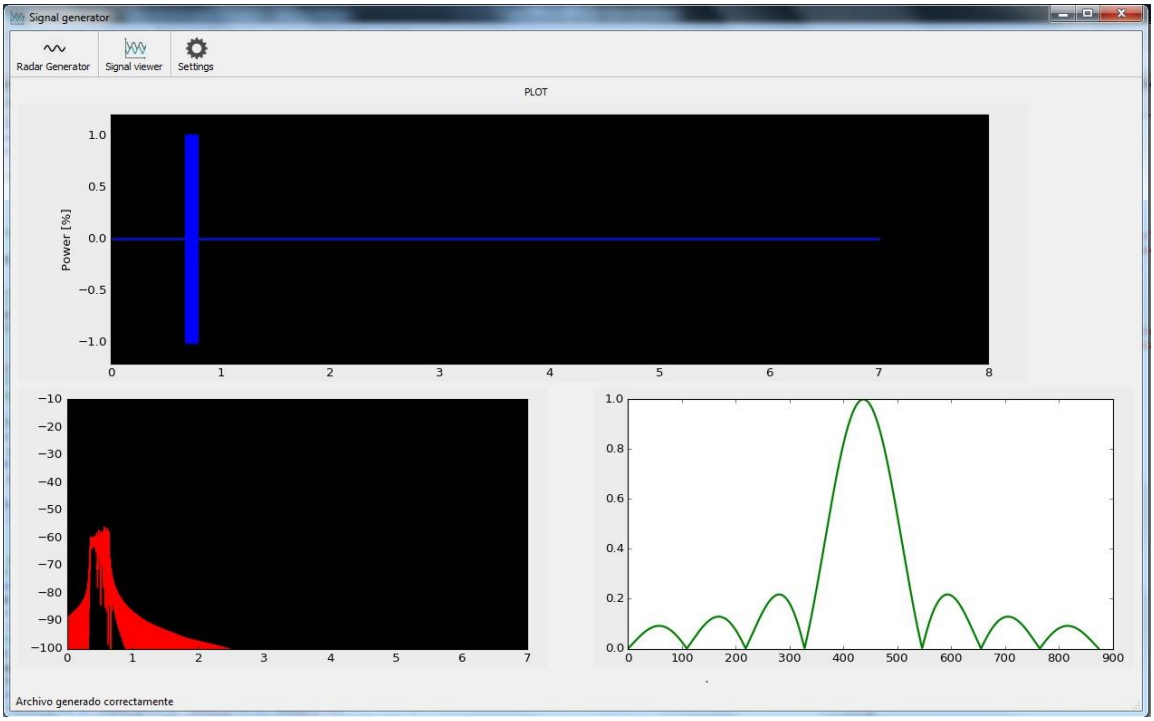



*Figura 21. Representación pulso Chirp*

En la gráfica que nos representa la frecuencia, observamos el desplazamiento que sufre, permitiéndonos confirmar el correcto funcionamiento.

### *5.2.3 Modulación Barker*

Pasamos a hablar a una de las modulaciones más utilizadas dentro del mundo de la Guerra Electrónica. Se trata de unas secuencias de tipo binario compuestas por la sucesión de -1's y +1's de una longitud finita. Utilizando las secuencias aplicamos una modulación en fase. Con este tipo de modulación favorecemos la correlación entre pulsos, mejorando así la relación señal a ruido S/N. Pasaremos pues a explicar la implementación en el programa.

Existen 8 tipos de modulación Barker, los cuales vienen predefinidos por los ingenieros Turyn y Storer. Para poder indicar cual debemos utilizar, tendremos la columna Barker para ello. En caso de introducir de manera no que secuencia queremos utilizar, se programó, como ya se explicó antes en el apartado sobre la interfaz, un sistema de asistencia para introducir correctamente el tipo de modulación. Se programará la secuencia de la siguiente forma:

```
secBarker =[]

if barker <2:
    secBarker = [0]
    barker=1
elif barker ==2:
    secBarker = [0, 1]
elif barker ==3:
    secBarker = [0, 0, 1]
elif barker ==4:
    secBarker = [0, 0, 1, 0]
elif barker ==5:
    secBarker = [0, 0, 0, 1, 0]
elif barker ==7:
    secBarker = [0, 0, 0, 1, 1, 0, 1]
elif barker ==11:
    secBarker = [0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1]
elif barker ==13:
    secBarker = [0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0]

secBarkerN = reshape(np.matlib.repmat(np.transpose(secBarker),ceil(len(t)/barker),1),(1,-1), order='F').flatten()
```

*Código 13. Programación Barker*

El proceso comienza creando el vector. Después, con una serie de "if´s" introducimos la serie de datos que correspondan al tipo de Barker. Por último, acomodamos los datos facilitándonos el trabajo a posteriori. Por último, se generará el pulso según ya hemos mostrado antes. Nos quedará pues, generar diferentes pulsos con diferentes modulaciones Barker.

| **Modo** | **F(MHz)** | **PRI(us)** | **PW(us)** | **Power** | **Offset** | **BW** | **Barker** | **Modo Radar** |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 5 | None |

*Tabla 6. Modulación Barker5*

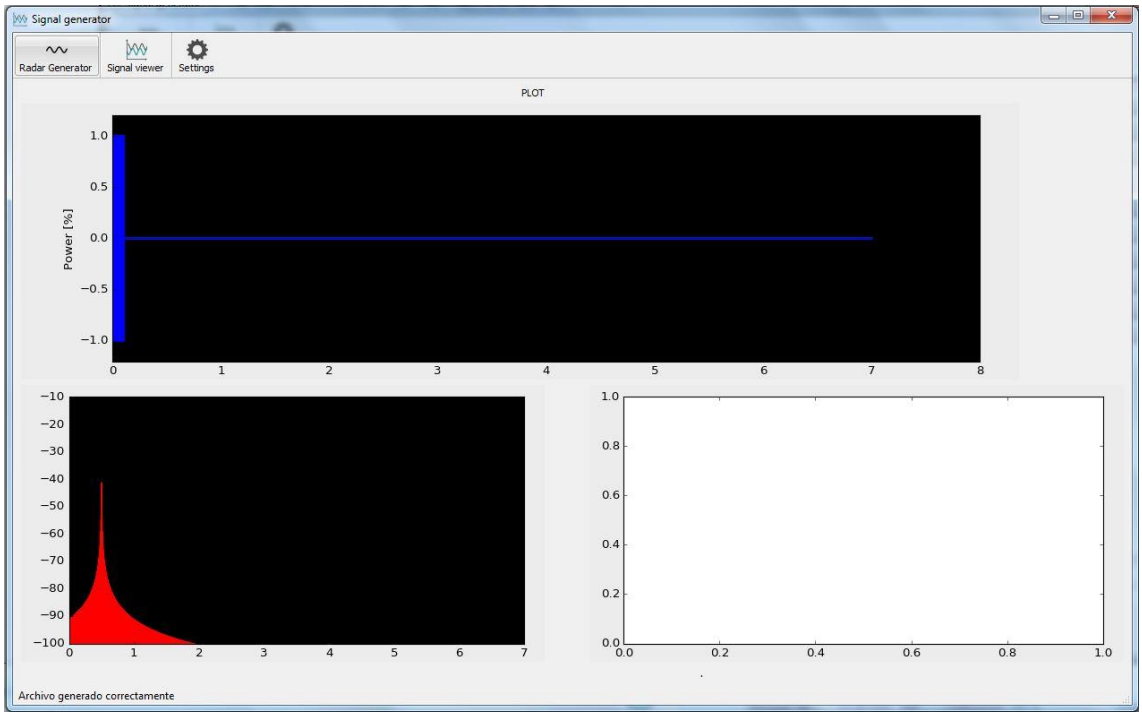



*Figura 22. Representación Barker 5*

Seguimos probando con más valores para ver las diferencias:

| **Modo** | **F(MHz)** | **PRI(us)** | **PW(us)** | **Power** | **Offset** | **BW** | **Barker** | **Modo Radar** |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 11 | None |

*Tabla 7. Modulación Barker11*

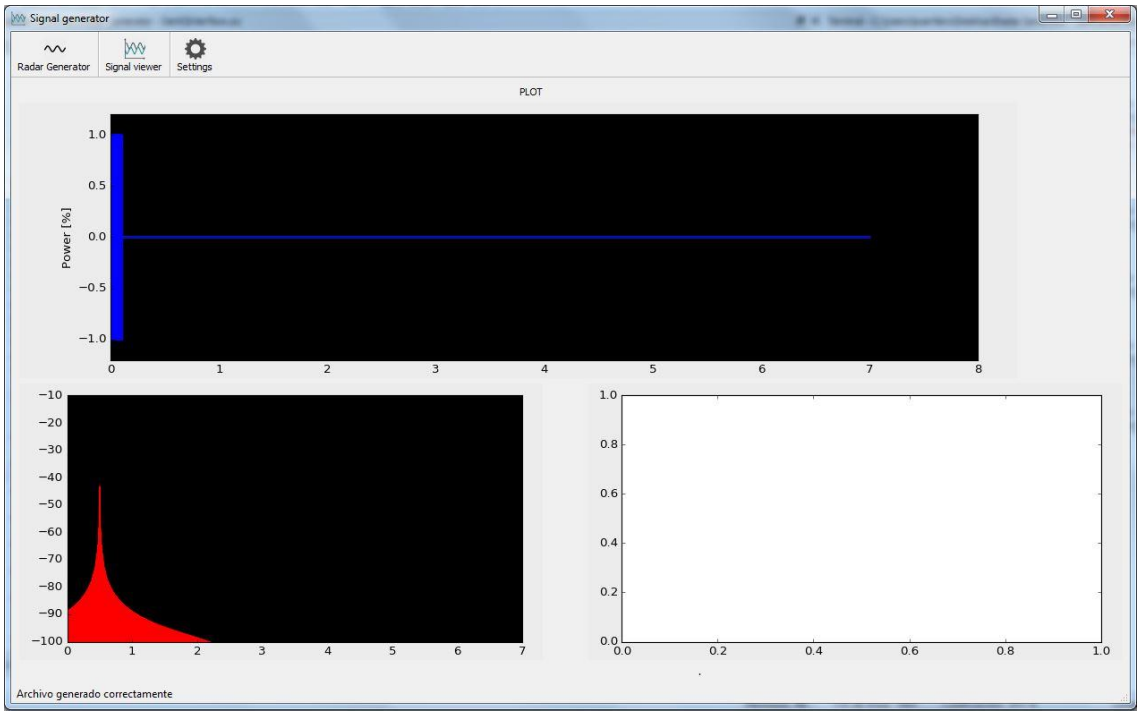



*Figura 23. Representación Barker 11*

Probamos un último valor:

| Modo | F(MHz) | PRI(us) | PW(us) | Power | Offset | BW | Barker | Modo Radar |
|---|---|---|---|---|---|---|---|---|
| **Pulse Radar** | 500.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 13 | None |

*Tabla 8. Modulación Barker 13*

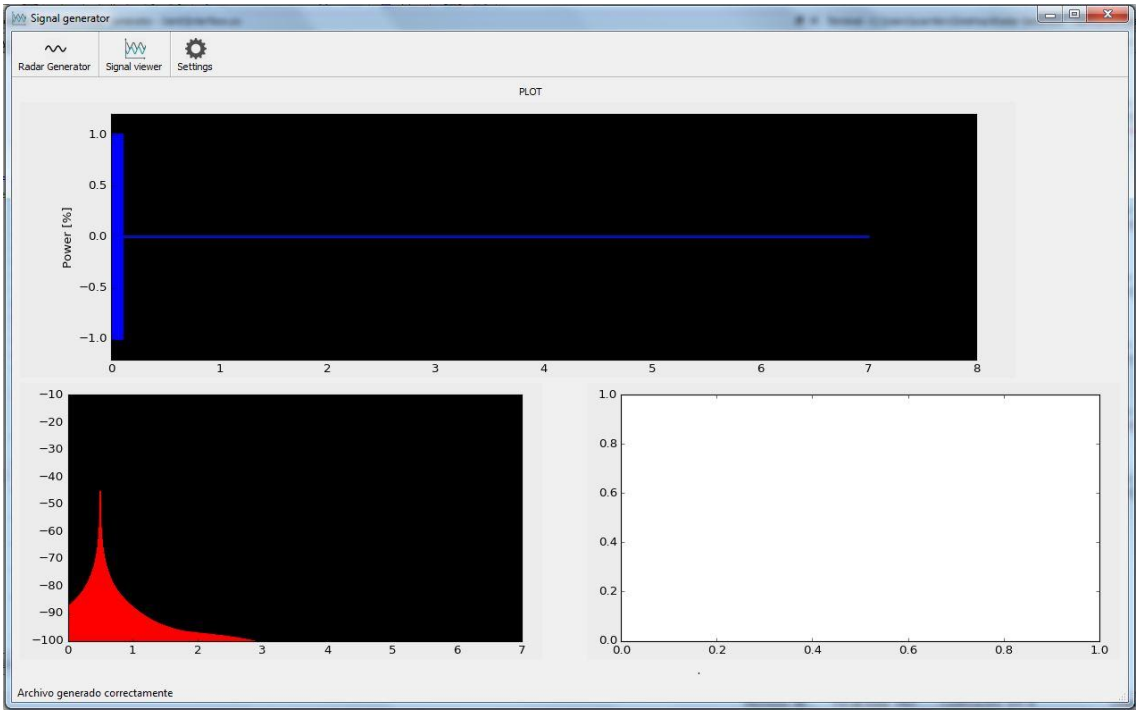



*Figura 24. Representación Barker 13*

Si realizamos una comparativa entre todas las gráficas, comprobamos que al aumentar la longitud del vector Barker, la Función de Transferencia de Fourier se ensancha. Se trata de un efecto deseado ya que es general al generar un pulso con modulación Barker. Por ello confirmamos su buen funcionamiento.

En estos apartados hemos visto las posibles modulaciones interpulso, es decir, modulaciones que afectan solo a un pulso. A partir de ahora comenzaremos a hablar de modulaciones interpulso.

### *5.2.4 Modulación PW Agility*

Como hemos explicado anteriormente en el apartado de teoría, la resolución espacial depende del ancho de pulso. Para prevenir un error provocado por el PW, existen radares que lo varían pulso a pulso. Así consiguiendo también, reducir la capacidad de intercepción por sistemas ELINT. Por el gran interés, se ha añadido al programa de la siguiente manera:

```
f_aux = Freq + np.linspace(-BW_int/2.0,BW_int/2.0,N)
f = [f_aux[0], f_aux[-1]]
for k in numpy.arange(1,N):
    f_next = 0
    while f_next==0 or abs(f_next - f[-1]) < BW_int/4:
        vector=np.random.rand(1)
        f_next = [f_aux[int(ceil(N*vector))-1]]
    f = np.concatenate((f, f_next),axis=0)
if Modo[i]==4 or Modo[i]==5:
    #PW_mod = PW_int*(np.around(rand(N)*2,decimals=5) + 0.2)    ##OLD
    PW_range = map(lambda x: x * PW_int, [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8])
    PW_mod = [PW_range[0], PW_range[-1]]
    for k in numpy.arange(1,N):
        PW_next = 0
        while PW_next==0:
            rand = random.randint(1, 8)
            PW_next = [PW_range[rand]]
        PW_mod = np.concatenate((PW_mod, PW_next),axis=0)
    print PW_mod
```

*Código 14. Implementación PW Agility*

Primero definimos la frecuencia de modo que podemos incluir el BW para este caso. El siguiente bucle "for" está destinado a otro tipo de modulación que hablaremos más tarde. Por tanto, entraremos al "if" comenzando a crear un vector de PW del tamaño que el programa indica, es decir, tendremos 9 pulsos con diferente ancho. Para crear mayor confusión en una posible intercepción de los pulsos, se desordenan como se puede observar en las siguientes instrucciones. En ellas crearemos el vector PW_mod, es decir, donde modificaremos el orden. A continuación, iremos recorriendo punto a punto del vector para no dejarnos ningún ancho sin incluir.

Dentro de la anterior programación caben muchas modificaciones, en nuestro caso este ha sido la mejor debido a nuestros intereses, ya que también es interesante conocer que se manda para comparar con lo que nuestro equipo es capaz de hallar. En caso de anchos de pulso aleatorios, nos sería más complicado realizar comparaciones que nos diesen confianza. Pero se podrían incluir por ejemplo infinitos anchos de pulso. Solo nos quedará generar pulso a pulso y unirlos, donde obtendríamos:

| **Modo** | **F(MHz)** | **PRI(us)** | **PW(us)** | **Power** | **Offset** | **BW** | **Barker** | **Modo Radar** |
|---|---|---|---|---|---|---|---|---|
| **PW Agility** | 200.0 | 200.0 | 10.0 | 10.0 | 10.0 | 0.0 | 1 | None |

*Tabla 9. Datos entrada PW Agility*

Obteniendo la siguiente gráfica:

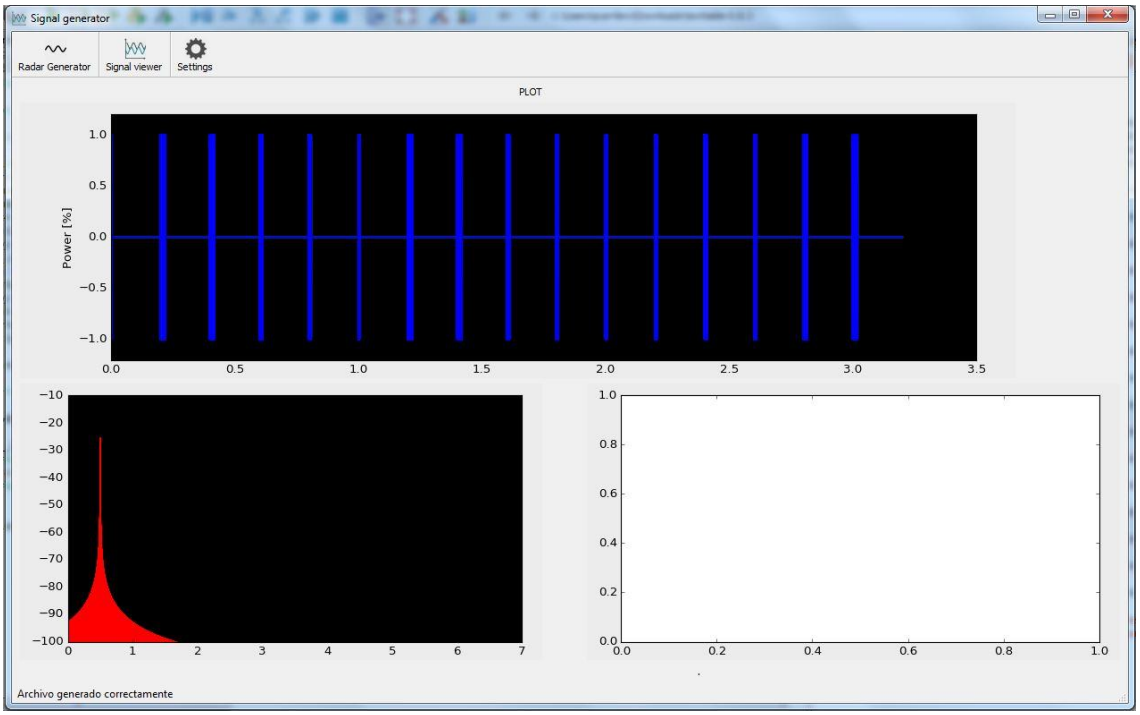



*Figura 24. Representación PW Agility*

Como podemos observar a simple vista, tenemos una gran cantidad de pulsos todos ellos de un ancho diferente, creando así un tren de pulsos PW Agility.

### *5.2.5 Frequency hopping*

Se trata de otro de los principales métodos utilizados en guerra electrónica para reducir la probabilidad de intercepción. Se basa en variar la frecuencia entre pulso y pulso. De esta forma un sistema que intente interceptar radares enemigos puede obviar dicho radar puesto que puede tomarlo como varios objetivos, o simplemente solo captando los pulsos a cierta frecuencia y obviando el resto. La implementación ha sido la siguiente:

```
f_aux = Freq + np.linspace(-BW_int/2.0,BW_int/2.0,N)
f = [f_aux[0], f_aux[-1]]
for k in numpy.arange(1,N):
    f_next = 0
    while f_next==0 or abs(f_next - f[-1]) < BW_int/4:
        vector=np.random.rand(1)
        f_next = [f_aux[int(ceil(N*vector))-1]]
    f = np.concatenate((f, f_next),axis=0)
```

*Código 15. Implementación Frequency Hopping*

Se trata del mismo proceso enseñado antes ya que comparte función con la modulación PW Agility. En este caso, comenzamos creando dos vectores, uno donde iremos almacenando valores y otro para el vector de frecuencias finales. El tamaño del vector será indicado por el máximo impuesto por el Generador y calculado por el mínimo común múltiplo. Mediante el valor BW añadido en la tabla hallaremos las frecuencias a generar y el vector que mandaremos para generar cada señal. Podemos pues pasar a realizar una prueba.

Los valores incluidos en la tabla son los siguientes:

| Modo | F(MHz) | PRI(us) | PW(us) | Power | Offset | BW | Barker | Modo Radar |
|---|---|---|---|---|---|---|---|---|
| **Frequency hopping** | 200.0 | 700.0 | 10.0 | 10.0 | 0.0 | 0.0 | 1 | None |

*Tabla 10. Datos Frequency hopping*

De modo que obtenemos las siguientes gráficas:

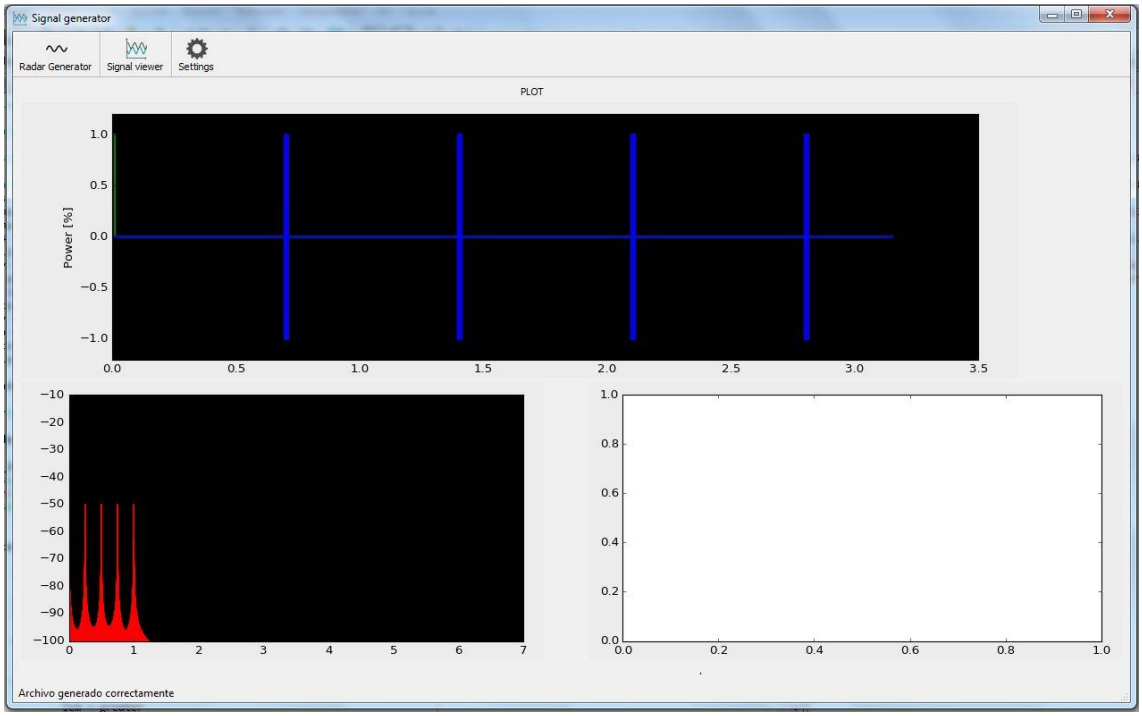



*Figura 25. Representación Frequency hopping*

Observamos que se han dibujado cinco pulsos en cinco frecuencias distintas. Como podemos ver, también podemos obtener frecuencia cero.

Tenemos la opción de combinar ambas modulaciones, obteniendo así una modulación llamada "PW + FH". De este modo producimos una señal con una tasa de intercepción aún más baja. Por tanto, un reto mayor para nuestro testing. Introduciremos los datos:

| Modo | F(MHz) | PRI(us) | PW(us) | Power | Offset | BW | Barker | Modo Radar |
|---|---|---|---|---|---|---|---|---|
| **PW +FH** | 500.0 | 100.0 | 5.0 | 10.0 | 0.0 | 10.0 | 1 | None |

*Tabla 11. Introducción datos PW + FH*

Obteniendo la siguiente gráfica:

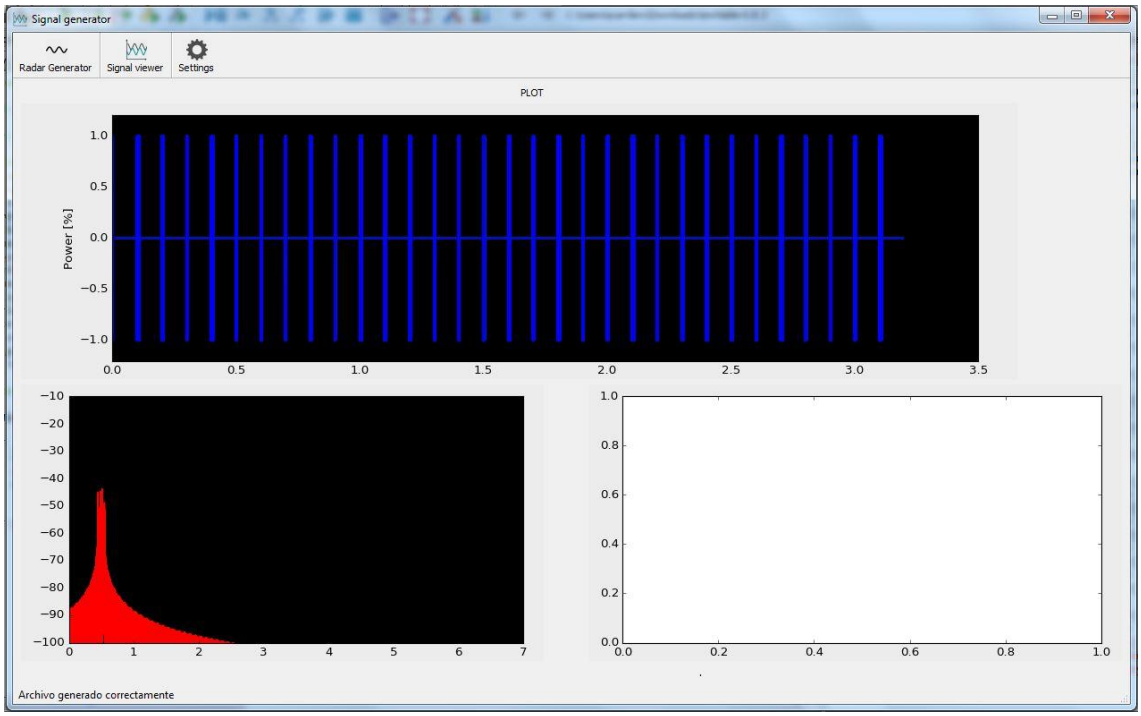



*Figura 26. Representación FH + PW*

Primero que todo, comprobamos a simple vista que se han producido ambas modulaciones, en la gráfica superior distinguimos que las señales tienen diferentes anchos. En la gráfica inferior, observamos el leve cambio en frecuencia que nosotros hemos indicado. Así garantizamos el buen funcionamiento. Además, podemos obtener más información, y se debe a la cantidad de pulsos generados, en este caso se ha indicado un PRI mucho menor que en ejemplos anteriores, provocando así una necesidad menor de muestras para generar cada pulso y su silencio.

## 5.3 Modo de Radar

Pasamos pues, a un tema diferente donde ya no hablaremos de las modulaciones u operaciones que puedan afectar al pulso, sino operaciones que afectan a toda la señal. Y es que como sabemos, un radar no es un sistema fijo que siempre esté apuntando hacia nosotros y cada pulso nos vaya a llegar con la misma intensidad. Por tanto, será necesario recrear el movimiento del Radar como punto importante del testing del sistema.

Los radares modernos incluyen una parte de inteligencia electrónica, mejorando su efectividad en gran medida. Su funcionamiento se caracteriza en una base de datos en la que se tiene conocimiento del tipo de objetivo que se busca. En caso de, por ejemplo, la búsqueda de un avión, podemos conocer que clase de características que nuestro radar debe reconocer. Se debe a que existe una serie de factores como pueden ser la RCS (Radar Cross Section o Sección de Radar Equivalente) que es común en objetos de similar tamaño. Aunque también se trata de un campo que avanza en gran medida con la llegada de aviones tipo Stealth (Sigilo), que consiguen ser prácticamente invisibles a los ojos de un Radar.

Como veremos a continuación en una imagen ejemplo, existe una gran cantidad de formas de Scan. Cada una tiene una forma y está destinada a un uso:

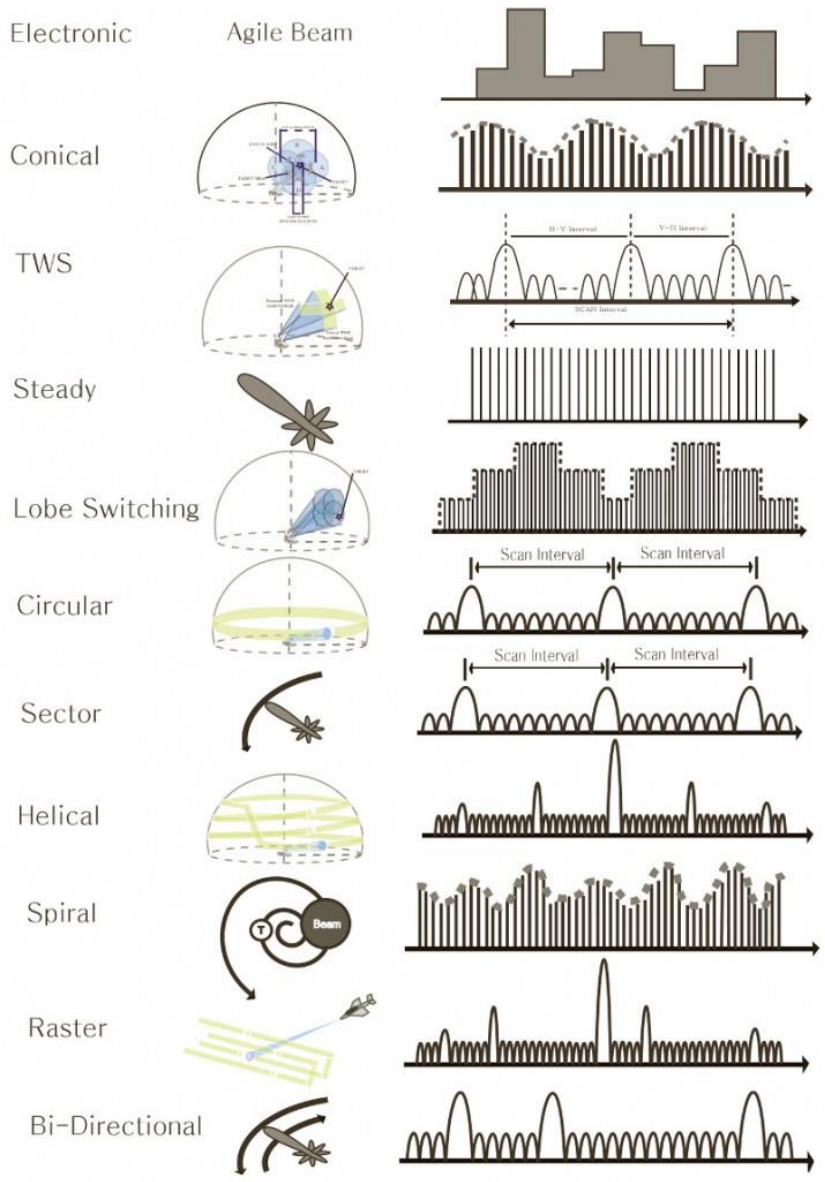



*Figura 27. Tipos de Scan*

En nuestro caso se ha producido la programación de dos tipos de Scan. El primero ha sido llamado track, donde se desea simular el movimiento clásico de un radar. Se ha programado de la siguiente forma:

```
x = np.linspace(-4, 4, Npulsos)
diag = abs(np.sinc(x))
```

*Código 16. Programación Scan*

El proceso consta en crear primero un vector de puntos por el que se generará la siguiente función. El diagrama se calculará a partir de una función sinc. Para mejorar la forma de onda, convertiremos la parte negativa en positiva. Si lo generamos obtenemos la siguiente gráfica:

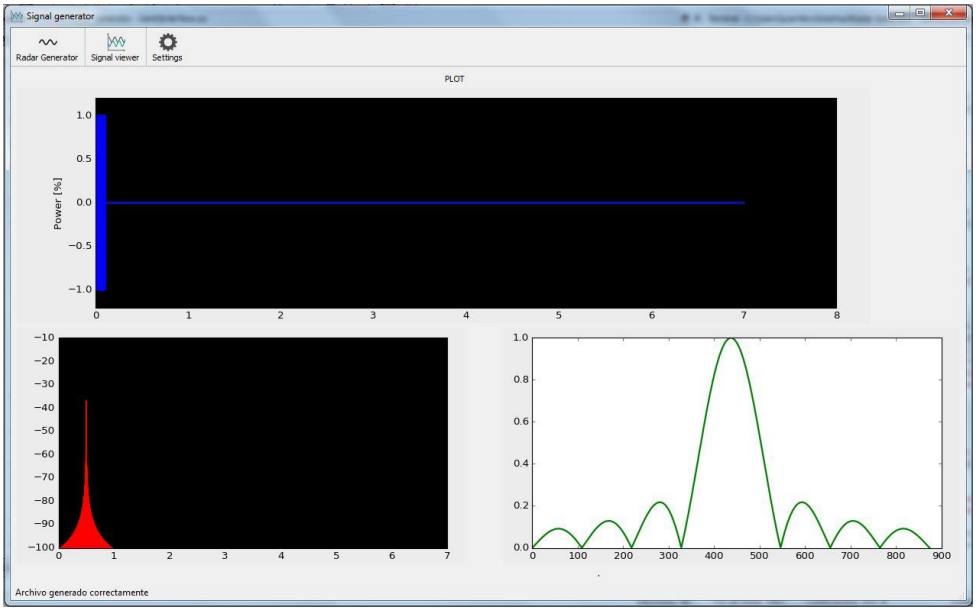



*Figura 28. Representación modo track*

Y seguiremos con la siguiente forma de onda, en este caso nos hemos centrado en una forma más sencilla, como se verá a continuación:

```
x = np.linspace(0, 25.129999, Npulsos)
diag = abs(sin(x))
```

*Código 17. Programación Scan 2*

Se ha programado de una similar al otro tipo de Scan. En la creación del vector se ha modificado la generación para cuadrar el inicio con el final, haciéndolo circular y que sea más realista. Para la segunda parte, hemos modelado la forma de onda con un seno, donde otra vez hemos vuelto a convertir la parte negativa en positiva.

Donde obtenemos:

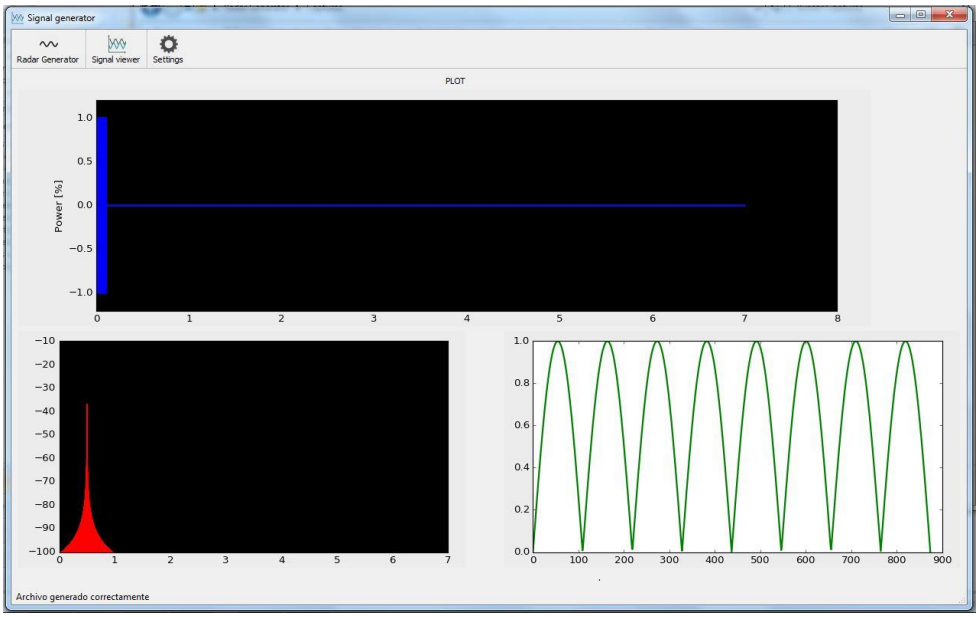



*Figura 29. Representación modo Scan.*

Por último, veremos las operaciones en las que se efectúa la unión con los pulsos:

```
for i in range(0,Npulsos):
    for j in range(0,len_pulse):
        Matrix[i][j] = pulse[j]*diag[i]
```

*Código 18. Obtención forma de onda.*

Mediante un bucle “for” recorremos los pulsos creados y el diagrama multiplicando muestra a muestra de cada vector. El resultado final será guardado en Matrix.

## 5.4 Desarrollo en GNU Radio

Paralelamente al desarrollo en Python sobre un generador de señales, el cual hemos estudiado en los últimos capítulos, se realizó otro generador de señales en la plataforma GNU Radio. Se piensa en esta plataforma por la facilidad aportada al tener en posesión una radio SDR de la compañía Ettus Research17 modelo B100. Su gran capacidad en la generación de señales, nos podía aportar mayor versatilidad que un generador profesional, aun teniendo en cuenta la perdida en calidad de señal o no poder llegar a frecuencias como 40 GHz. Su pequeño tamaño es ideal para un fácil transporte y en cualquiera circunstancia. El pequeño precio suponía un punto a favor en el uso como equipo de calibración. Debido a todas estas ventajas, se decidió crear un generador lo más potente posible en esta plataforma. Y los resultados han sido los siguientes:

**Pulse_Generator_Burst.grc**

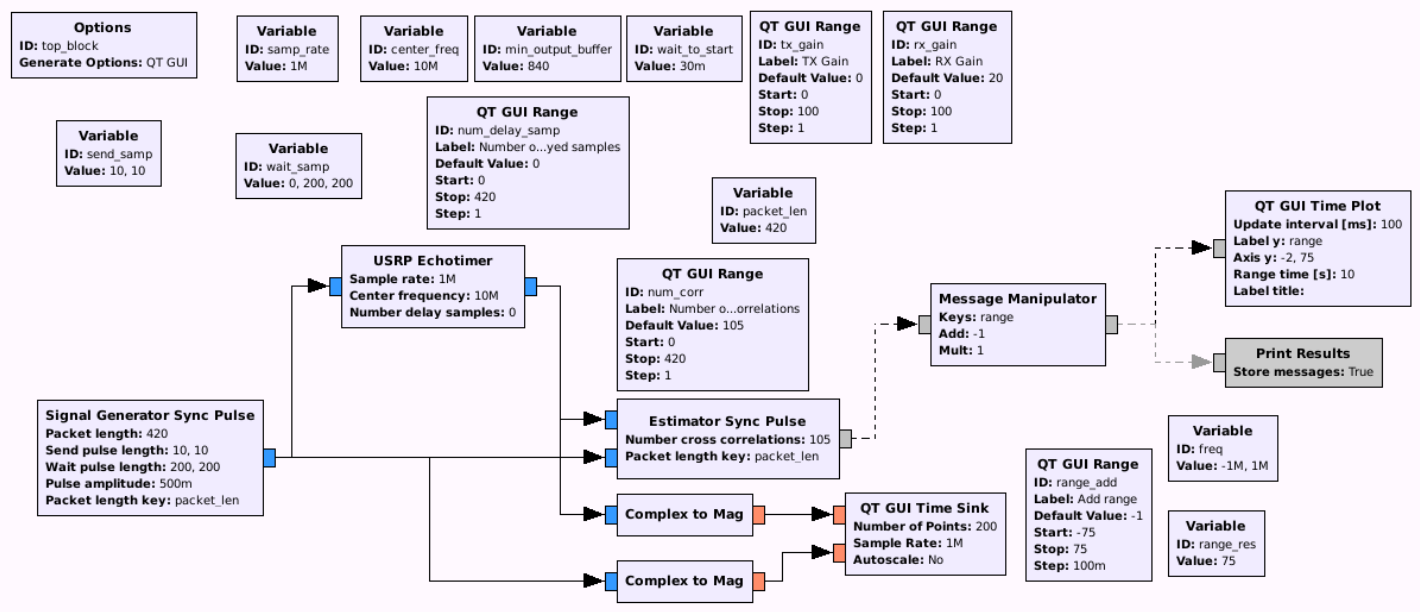



*Figura 30. Diagrama Pulse_Generator_Burst.grc*

**Signal Generator Sync Pulse:**

La señal que este bloque genera es un pulso con una amplitud constante con diferentes longitudes de pulso y silencios entre ellos. Está estructurado para alternar las partes de silencio con las partes de pulso, empezando por la parte de silencio. Los pulsos son completamente señales reales.

Los valores de entrada son:

- Packet lenght: el valor deberá ser mayor o igual a la suma de send_samp y wait_samp.
- Send pulse lenght: la longitud de cada pulso está definida en un vector que viene dada en número de muestras. Los pulsos son generados alternativamente con los wait samples definidos en wait_samp
- Wait pulse lenght: Las muestras de silencio son definidas en un vector. El primer segmento de toda la señal es el primer elemento del vector wait_samp. Este elemento puede ser cero.
- Pulse amplitude: amplitud de la señal generada
- El tamaño del PW y PRI variará según las variables send_samp y wait_samp. Dichas variables se tratan de vectores que el Signal Generator Sync Pulse leerá de forma intercalada.

[17] https://www.ettus.com/

$$Ancho\ silencio = \frac{wait\_samp}{samp\_rate}\ [s] \quad (19)$$

$$Ancho\ pulso = \frac{send\_samp}{samp\_rate}\ [s] \quad (20)$$

$$\text{Send\_samp} = (\text{PulseWidth1, PW2, ..., PWn}) \quad (21)$$

$$\text{Wait\_samp} = (\text{Silencio inicial, SilenceWidth1, SW2, ..., SWn}) \quad (22)$$

$$\text{PRI} = \text{PW} + \text{SW} \quad (23)$$

**USRP Echotimer:**

Se encarga de conectarse con el USRP y de enviarle la señal introducida. Para poder realizar una sincronización de la señal recibida, el Echotimer le asigna a cada "Ráfaga" un timestamp de modo que, al recibirla, la señal se podrá visualizar correctamente. El periodo de cada ráfaga es de 50 ms, donde durante ese tiempo, enviará la señal y recibirá cada eco. Por tanto, podemos detectar un objetivo 20 veces por segundo.

Debemos indicarle la frecuencia de muestreo mediante samp_rate y la frecuencia en la que transmitirá y recibirá mediante center_freq

*Para conocer la dirección de la SDR, podemos utilizar el comando "uhd_find_devices"

El Echotimer nos permite utilizar 2 SDR, una para transmitir y otra para recibir, cuyo uso es más recomendable debido a que las USRP B100 son "Half-Duplex" y no permiten transmisión y recepción simultánea.

Para indicar la antena que deseamos utilizar lo deberemos indicar mediante:

TX/RX → Conexión a la izquierda, permite transmisión y recepción.

RX2 → Conexión a la derecha, solo permite recepción.

La ganancia de cada antena, transmisión y recepción, está incluida en 2 sliders que permiten modificar su valor una vez esté funcionando, pudiendo así realizar un ajuste más preciso al momento. También nos permite configurar diferentes valores como el reloj de funcionamiento o los tiempos de espera para cada transmisión.

**Estimator Sync Pulse:**

Permite realizar una correlación de la señal recibida y la señal enviada, de modo que obtenemos el tiempo de llegada, pudiendo así obtener el rango del objeto reflejado.

Sus valores de entrada el número de correlaciones cruzadas y el "packet lenght" que coincide con la suma de send_samp y wait_samp.

A su salida se debe obtener en forma de "mensaje", el reception time. De modo, que es pasado al Plot para poder ser representado en una gráfica en tiempo real.

Sin embargo, el valor de salida es un mensaje de error. Por tanto, funcionamiento no ha sido posible debido a errores de ejecución en el código del bloque. Esto ha sido imposible de subsanar debido a la inexistente documentación sobre su funcionamiento.

## Pulse_Generator.grc

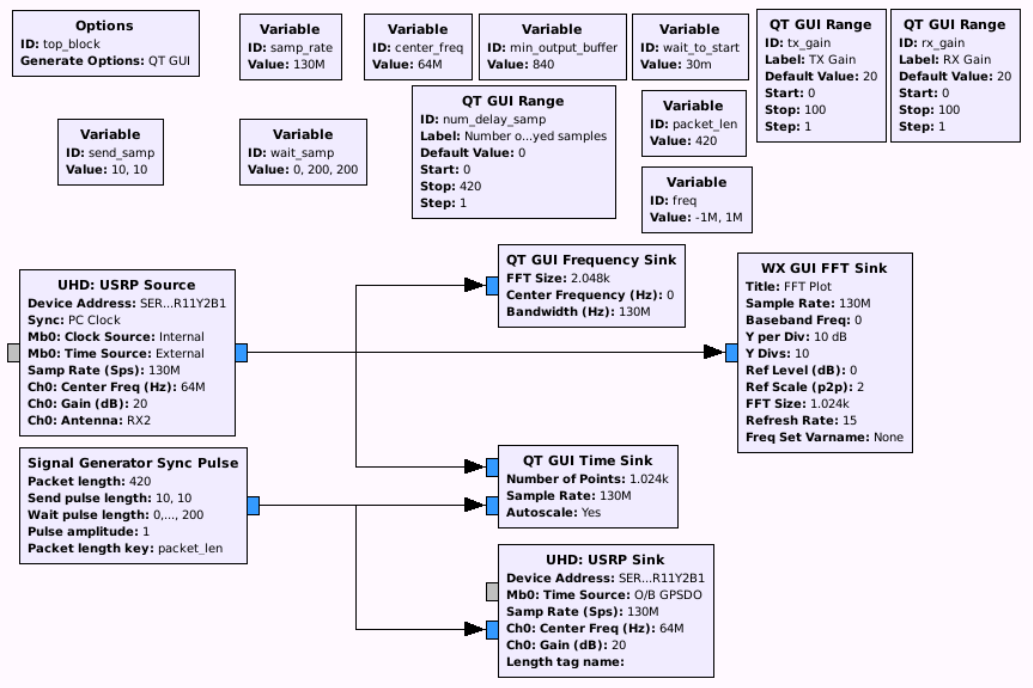



*Figura 31. Diagrama Pulse_Generator.grc*

Nos encontramos ante otra versión de un generador de pulsos. En este caso, a diferencia del anterior, nos permite generar y enviar pulsos de forma continua. Esta configuración será más útil en caso de realizar testing.

El funcionamiento es similar al anterior, donde la única diferencia se halla en el bloque encargado de enviar la señal al USRP.

**UHD: USRP Sink:**

El bloque se encargará de enviar de manera ininterrumpida el flujo de datos a su entrada.

Debemos indicar los siguientes campos:

- Device Address: obtenido mediante el comando *und_find_devices.*
- Samp Rate (Sps): Tasa de muestras por segundo introducidas al bloque.
- Center Freq (HZ): Frecuencia de emisión.
- Gain (dB): ganacia que le quieras aportar a la transmisión.
- Antenna: se indicará TX/RX ya que la finalidad es transmitir.
- Bandwidth: por defecto se configurará a cero. En caso de ciertos dispositivos, tendremos la posibilidad de configurarlo.

Como opciones extra de configuración tenemos la tasa de cada reloj y la sincronización. La sincronización se trata de un tema esencial para el uso del sistema como un radar.

**UHD: USRP Source:**

Se trata de un bloque idéntico al *UHD: USRP Sink* pero en este caso indicado para recibir una señal. Su funcionamiento también es ininterrumpido, por lo que se hace ideal para la aplicación de radar.

Su configuración se realizará de la misma forma que con el transmisor excepto en ciertos parámetros:

- Device Address: igual al transmisor en caso de compartir SDR o se obtendrá mediante el comando *und_find_devices.*
- Samp Rate (Sps): Tasa de muestreo. Mínimo: 2*Center_Freq
- Center Freq (HZ): Frecuencia de recepción.
- Gain (dB): ganancia que será aportada a la señal recibida.
- Antenna: se indicará RX2 ya que la finalidad es recibir
- Bandwidth: por defecto se configurará a cero. En caso de ciertos dispositivos, tendremos la posibilidad de configurarlo.

Como desventaja ante el Pulse_Generator_Burst, cabe decir que nos será imposible una sincronización con los métodos actuales. Las limitaciones son debidas en primera instancia al Hardware utilizado, que en este caso ha sido el USRP B100 que no permite sincronización debido a su limitación Half-Duplex.

## Pulse_CHIRP.grc

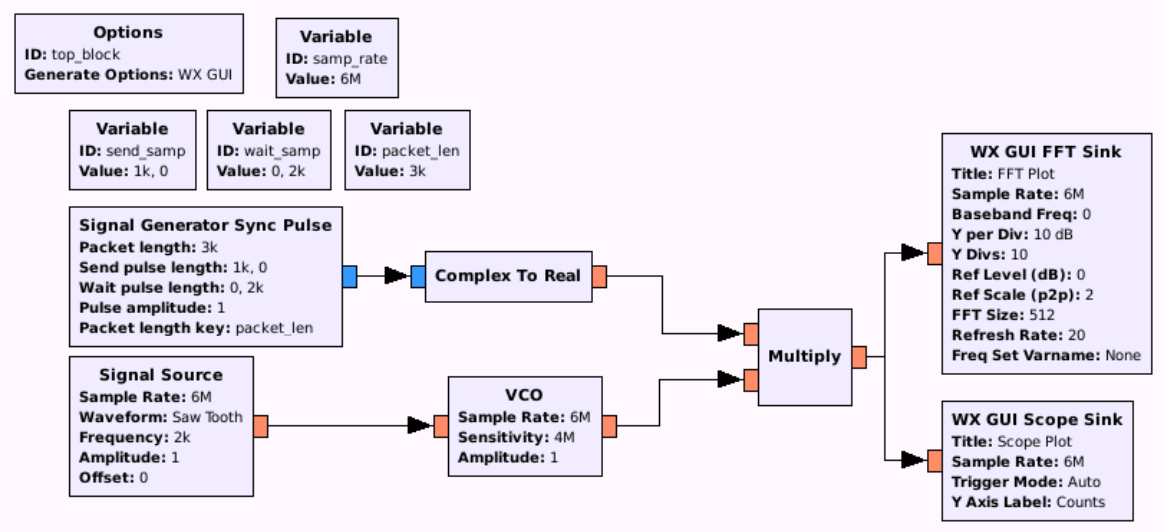
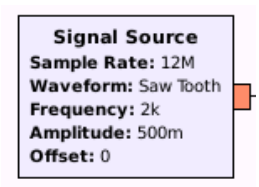



*Figura 32. Diagrama Pulse_CHIRP.grc*

Para conseguir una mayor resolución es cada pulso, es conveniente transmitir el pulso con formas de onda más complejas, como es en este caso una chrip.

La modulación chirp consiste en una modulación en frecuencia lineal. Esta se caracteriza por su variación en frecuencia que puede aumentar, llamándose up-chirp, o disminuir, down-chirp.

El primer bloque a analizar será ***Signal Source:***



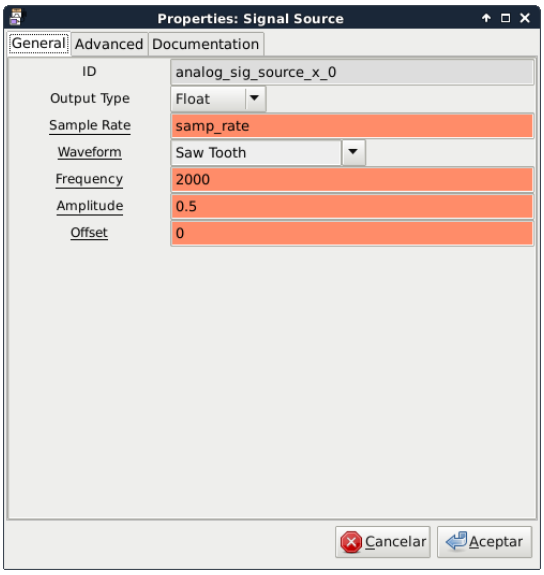
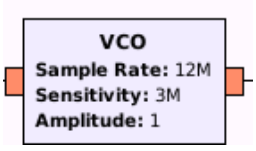




*Figura 33. Bloque Signal Source*

Se trata de un generador de señales básicas, los parámetros de entrada serán:

- Output Type: permite elegir el tipo de dato entre Complex, Float, Int, Short
- Sample_rate: debemos escoger una tasa de muestreo, siempre mayor al doble de la frecuencia.
- Waveform: Permite escoger el tipo de onda a generar. Las opciones serán Constant, Sine, Cosine, Square, Triangle y Saw Tooth.
- Frecuencia: Los hercios en que se generará la señal
- Amplitud: Indica el voltaje de salida de la señal
- Offset: indica el desplazamiento temporal de la señal.

El siguiente bloque a analizar es el **VCO**:



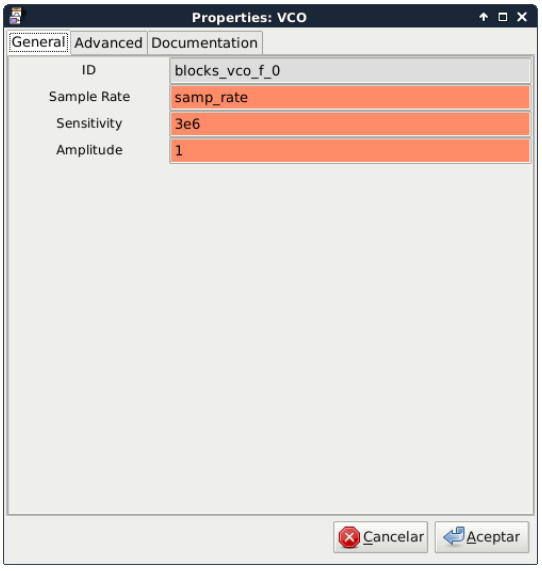




*Figura 34. Bloque BCO*

Este bloque implementa un oscilador controlado por el voltaje de entrada. La frecuencia de salida es linealmente proporcional al voltaje de entrada multiplicado por la sensibilidad.

Por tanto, deberemos indicarle el mismo sample rate que al signal source.

La Amplitude nos permitirá multiplicar por ese valor la señal de salida.

Por último, la sensibilidad funcionará del siguiente modo:

$$Frec_{salida} = V_{entrada} * (Sensitivity * \frac{1}{2\pi}) \quad (24)$$

Esto es debido a que el parámetro Sensitivity está configurado en rad/sec, por tanto, para conocer la frecuencia que tendremos a la salida del VCO, deberemos realizar la conversión con la formula anterior.

Por tanto, tendremos que la frecuencia menor será el menor voltaje que tenga a la entrada. Debido a que el signal source genera una señal (Saw Tooth) que su nivel mínimo es cero y su nivel máximo "Amplitude", para evitar que la señal llegue a frecuencia igual a cero, se le suma una constante para que el valor mínimo sea siempre esa contante.

De este modo, la $f_0$ estará determinada por el valor que se indique en **Constant Source** y **Sensitivity.** Y la frecuencia mayor que puede alcanzar será determinada por la suma del valor de Constant Source y la Amplitude de salida de Signal Source.

Por tanto, la clave para la correcta generación de una CHIRP será primero escoger la señal generada:

- SAWTOOTH: Se generará una UP-CHIRP con una pendiente lineal
- TRIANGLE: Correctamente sincronizado, se generará una UP-CHIRP//DOWN-CHIRP
- SINE/COSINE: Se generará una UP-CHIRP//DOWN-CHIRP pero con una pendiente correspondiente a un seno.

A continuación, se deberá escoger el ancho de banda de la CHIRP indicándola en Signal Source y en Constant. Pero también teniendo en cuenta la sensibilidad en el VCO. Donde deberemos realizar la conversión de Hercios a Rad/sec.

Por último, deberemos sincronizar el pulso con la señal generada. Por una parte, tener en cuenta el ancho del pulso más el silencio en tiempo y luego asignarle la frecuencia correcta al Signal Source. El resultado se puede observar en los ejemplos siguientes:

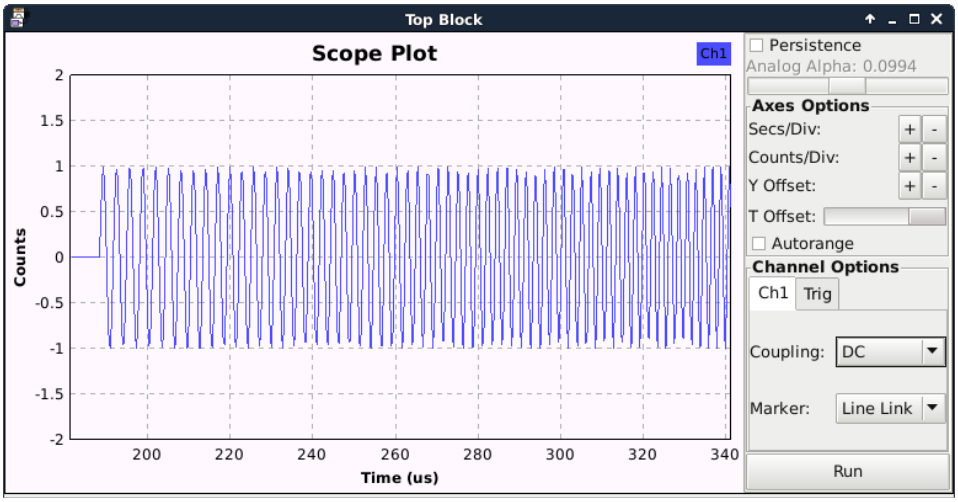



Figura 35. Plot de la señal

Donde en función del tiempo, podemos ver a simple vista como aumenta la frecuencia. Pero podremos ver el desplazamiento en frecuencia mediante la FFT.

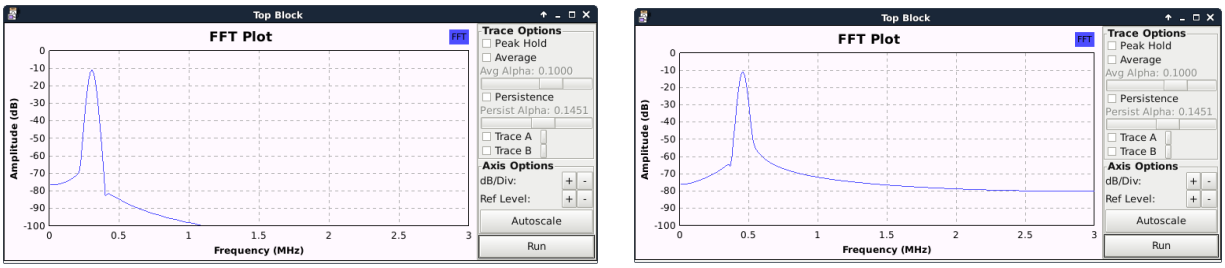



*Figura 36. Desplazamiento en frecuencia*

## Generador_interpulso.grc

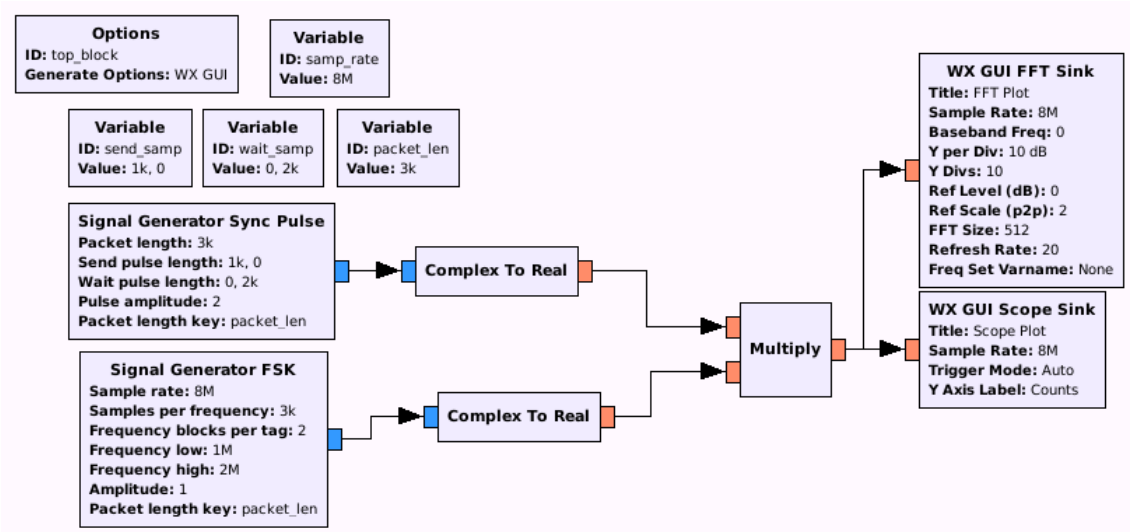



*Figura 37. Diagrama Generador_interpulso.grc*

En este caso nos encontramos ante un Generador de pulsos con salto en frecuencia entre pulsos. Esta clase de Radar se clasifica como Square-cave modulation (simple frequency-shift keying, FSK). Esta modulación es usada para unas medidas de distancia muy precisas por comparación del eco de dos señales en distinta frecuencia. Este tipo de radar sufre la desventaja de que el eco de varios objetivos no puede ser separado uno de otros. Por tanto, tan solo podremos utilizarlo para mejorar el Rango máximo no ambiguo.

El bloque principal será el **Signal Generator FSK:**

Su funcionamiento es muy simple, se basa en un generador de señales que se encarga de alternar las dos frecuencias que le sean indicadas. De este modo, los valores de entrada, deberán ser la *Frecuency Low* y la *Frecuency High*. Además, para indicar la duración de cada periodo, se deberá indicar en número de *Samples per frecuency*.

Para conseguir una FSK interpulso, deberemos hacer que el largo de muestras de cada frecuencia coincida con el largo de la suma de send_samp junto con wait_samp. Así, cada pulso generado coincidirá con cada cambio en frecuencia.

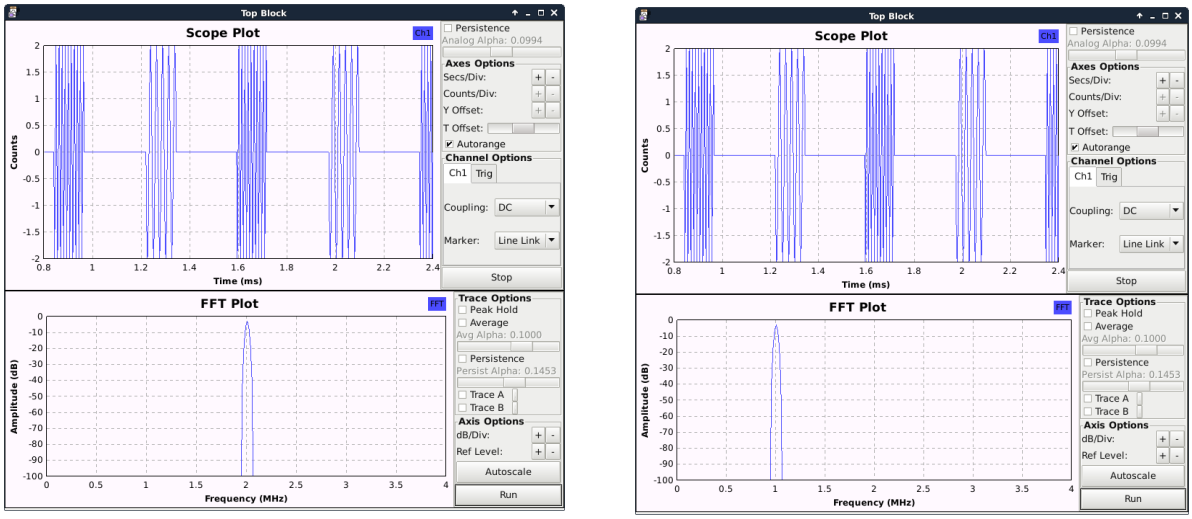



*Figura 38. Desplazamiento en frecuencia*

# Generador_intrapulso.grc

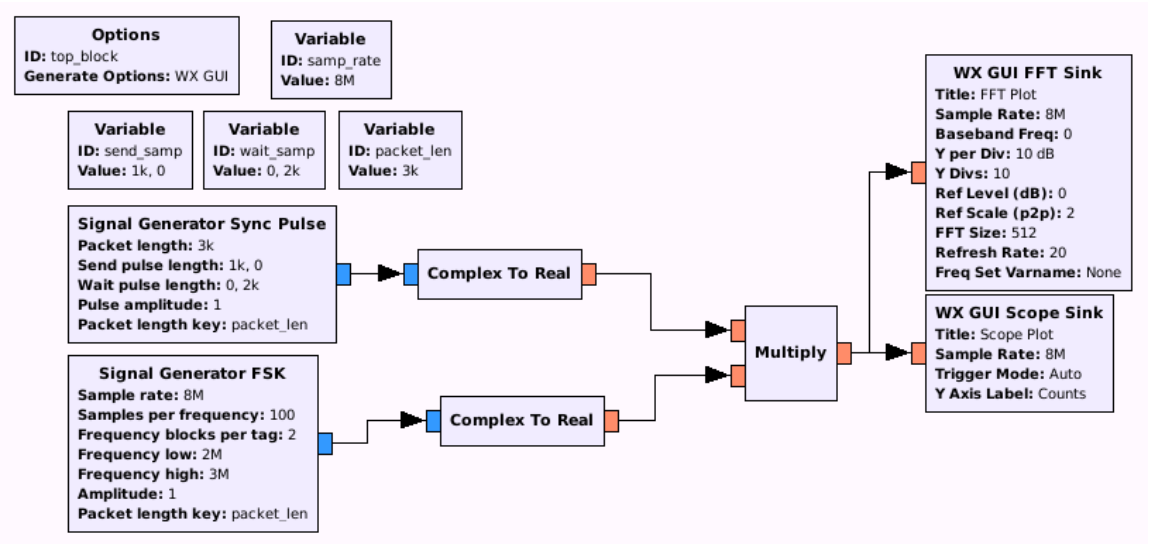



*Figura 39. Diagrama Generador_intrapulso.grc*

En este caso nos encontramos, igual que antes, ante un generador FSK, pero donde el salto en frecuencia se produce en el mismo pulso. De este modo añadimos una funcionalidad más al generador anterior.

Los bloques utilizados son los mismos al programa anterior. Por tanto, para que el salto en frecuencia se produzca en mitad del pulso, el tamaño de deberá ser de un tamaño igual o menor al ancho del pulso.

El resultado lo podemos observar en las capturas siguientes:

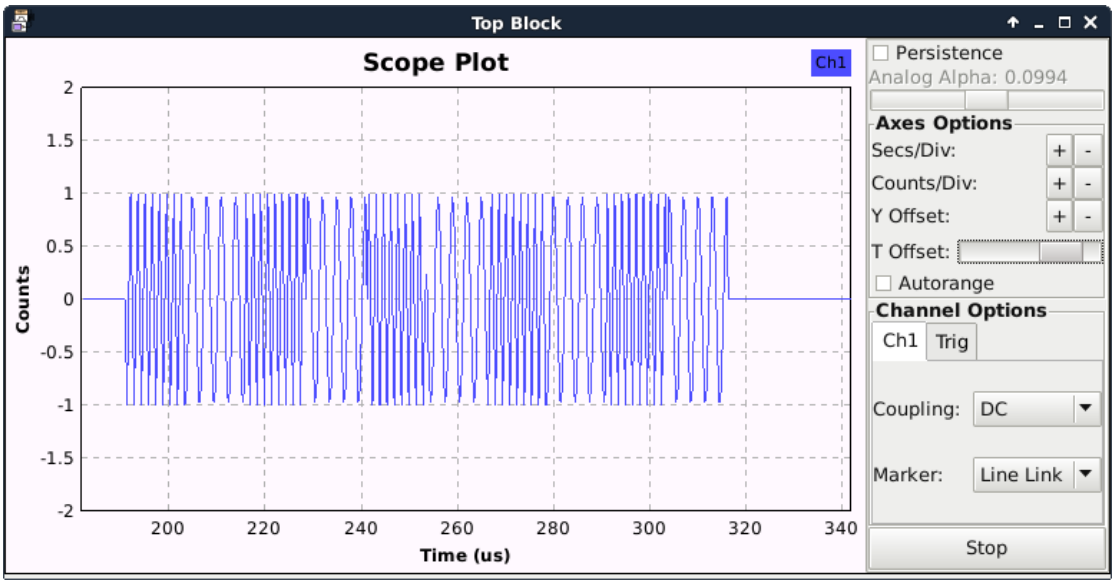



*Figura 40. Representación en tiempo*

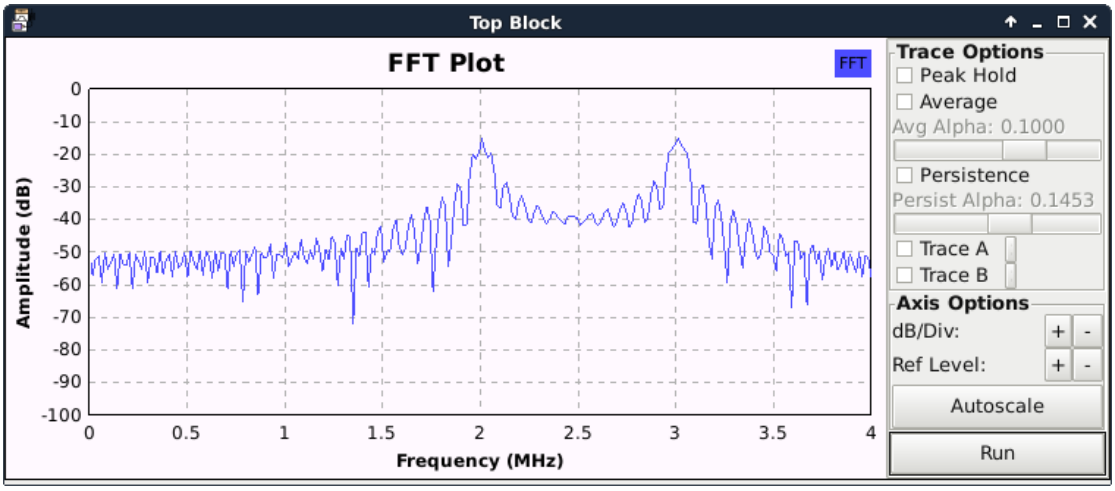



*Figura 41. Representación en frecuencia*

# Capítulo 6. Conclusiones y propuestas de trabajo futuro

Una vez realizado el presente Trabajo Final de Grado (o TFG), donde han sido desarrollados dos simuladores de radar, se concluirá con las diferencias entre ellos, teniendo en cuentas las ventajas e inconvenientes que supone el uso de ellos:

- Ambos suponen un problema a la hora de comenzar a utilizarlos, en caso de Python, se requieren una gran cantidad de módulos que no vienen preinstalados en Python, lo que conlleva querer utilizarlo por primera vez, tiempo en su instalación y ciertos conocimientos en el entorno Python, ya que no son tan sencillos como la instalación de un programa cualquiera. En el caso de GNU Radio sufrimos en mismo problema, y es que para poder utilizar el programa deberemos utilizar el sistema operativo Linux, plataforma en que está programado. A continuación, se requerirá la inclusión de los módulos externos, donde también se requieren ciertos conocimientos en el tema. Por todo ello, no son generadores de un uno rápido y sencillo.
- En ambos simuladores, para generar la señal en analógico, requeriremos de un generador externo. En el caso del generador programado en GNU Radio nos será más barato ya que podemos optar por la compra de SDR por valor de menos de 1000 euros. Sin embargo, en el caso del generador en Python, necesitaremos generadores de señal programables con precios superiores a los 50000 euros, ya que nuestros requerimientos llegan hasta la generación de señales de hasta 40 GHz. Sin embargo, cabe destacar que ni la calidad de la señal será la misma, ni que las SDR pueden generar señales por encima de los 5 GHZ.

En caso de las futuras propuestas y avances dentro del trabajo realizado serían:

- En el simulador en Python existe mucho trabajo posible, como puede ser la creación de más formas de onda, mayor cantidad de opciones en la generación de las señales, como es, por ejemplo, poder controlar la pendiente o la inclusión de más formas de Scan. Pero en especial, el siguiente paso a desarrollar sería la creación de los drivers para poder conectar el Software al Generador de señales.
- En caso del simulador en GNU Radio, sería interesante continuar por la creación de un radar útil capaz de detectar blancos. Así aprovechando las capacidades que nos aportan el software, ya que tiene muchas herramientas de procesado, y la SDR, ya que permite el envío y la recepción de señales de forma simultánea.

# Capítulo 7. Bibliografía

[1] Meikle, H. (2008). *Modern radar systems*. Boston: Artech House.

[2] Poisel, R. (2011). Modern communications jamming principles and techniques. Boston: Artech House.

[3] Anon, (2017). Formas de Onda de Radar. [online] Available at: http://agamenon.tsc.uah.es/Asignaturas/ittst/rdet/apuntes/handout2.pdf [Accessed 10 Sep. 2017].

[4] Adamy, D. (2004). EW 102. Boston: Artech House.