Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Control remoto de instrumentación para radiocomunicación mediante Python

Autor: Iván Pulido Muñoz

Tutor: María José Madero Ayora

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

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Control remoto de instrumentación para radiocomunicación mediante Python

Autor:
Iván Pulido Muñoz

Tutor:
María José Madero Ayora
Profesor titular

Dep. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Proyecto Fin de Carrera: Control remoto de instrumentación para radiocomunicación mediante Python

Autor: Iván Pulido Muñoz

Tutor: María José Madero Ayora

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia y amigos

A mis maestros

Agradecimientos

En primer lugar, agradecer a mis padres todo lo que han hecho por mí, siempre supieron cómo motivarme para seguir adelante. A Laura y mi familia por su apoyo incondicional durante estos años.

En segundo lugar, me gustaría darle las gracias a mis amigos y compañeros que me han ayudado y apoyado en todo lo que han podido.

Para finalizar hacer una especial mención a mi tutora, María José Madero Ayora, que supo transmitirme la motivación y el interés necesario para afrontar el trabajo.

Iván Pulido Muñoz

Trabajo Fin de Grado en Ingeniería de las Tecnologías de Telecomunicación

Sevilla, 2017

Resumen

La finalidad de este proyecto es el diseño de un código en lenguaje Python que se encargue de automatizar medidas en instrumentos para radiocomunicación mediante control remoto. Posteriormente, para comprobar que esta tarea es eficiente se caracterizan de manera experimental una serie de dispositivos no lineales mediante la generación de un tono. Además, se hará una investigación acerca de las prestaciones que ofrece Python en este campo.

Abstract

The aim of this project is to design a code using the Python programming language that automate measurements on instruments for radiocommunication remotely. Later, to check the efficiency of this task it will be characterize non-linear devices by generating a tone. In addition, research will be done on the output offered by Python in this field.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En la actualidad, cada vez son más comunes los sistemas que intercambian información utilizando el espectro radioeléctrico como medio de transmisión, es decir, los sistemas de radiocomunicación. Debido a su expansión, sobre todo en comunicaciones móviles, resulta de un gran interés realizar medidas para observar y controlar el comportamiento de los dispositivos que usan estas tecnologías. Esto se lleva a cabo con la ayuda de la instrumentación de laboratorio, para ello es necesario realizar miles de medidas con tiempos de ejecución de horas e incluso días en algunos casos. El objetivo de este proyecto es hacer más eficientes estas tareas.

Mediante el lenguaje de programación Python es posible diseñar programas que automatizan una medida en un instrumento de forma remota. La ejecución de los mismos permite controlar y conseguir datos de los equipos en cuestión de minutos.

1.1. Objetivos y alcance

Como se acaba de comentar, la automatización de medidas es un trabajo muy tedioso de hacer manualmente ya que se requieren unos tiempos de ejecución elevados para cada prueba. Debido a ello, se pretenden facilitar estas tareas mediante el diseño de un programa que realice medidas experimentales usando los instrumentos del laboratorio mediante control remoto.

Específicamente, el objetivo de este proyecto es el diseño de un código en lenguaje Python que caracterice dispositvos no lineales a través de la generación de un tono y con ello comprobar la eficiencia con la que se obtienen los resultados, tanto en formato numérico como gráfico, para poder trabajar con ellos. Además, debido a que es la primera vez que se usa Python para realizar tareas de automatización, se quieren estudiar las posibilidades y prestaciones que ofrece.

1.2. Organización y metodología

La memoria de este Trabajo Fin de Grado se ha organizado en tres bloques de contenidos que se indican a continuación brevemente resumidos:

- En los primeros capítulos se realiza un desarrollo teórico de todo concepto y material necesario para la realización del proyecto (desde el 2 al 5, ambos incluidos). Se comienza con una explicación de la Instrumentación Virtual, particularizando en el estándar VISA, para seguidamente introducir el bus GPIB, que utiliza VISA y permite realizar una comunicación entre varios instrumentos. Luego se profundiza en el lenguaje de programación utilizado para controlar y realizar las funciones que marcan este trabajo, Python, haciendo una sinopsis del mismo, para a posteriori, explicar cómo se puede llevar a cabo el control remoto de equipos y los diferentes paquetes a utilizar para esta labor. Finalmente, se nombran los equipos con los que se trabaja en el laboratorio comentando brevemente su utilidad y se describen los tipos de problemas no lineales que se pueden encontrar en los elementos a caracterizar.
- En segundo lugar, se ha realizado un desarrollo del código diseñado destacando las partes más importantes que han permitido conseguir el objetivo del trabajo y obtener medidas de potencia de un dispositivo no lineal.
- Por último, se han analizado los escenarios de ejemplo que se han utilizado para comprobar el correcto funcionamiento del código, comentando los resultados obtenidos en cada una de las pruebas hechas.

Cabe destacar que en el desarrollo de los conceptos teóricos, Capítulos 2, 3, 4 y 5, donde se ilustra al lector con información de las herramientas y los fundamentos empleados, lógicamente, no se pretende publicar un manual de referencia de todos los recursos utilizados, sino una introducción y breve descripción de cada uno de ellos. De esta forma, se busca asentar las bases que permitirán comprender mejor el proceso de investigación, estudio, aprendizaje y puesta en funcionamiento realizado en este Trabajo Fin de Grado.

2 INSTRUMENTACIÓN VIRTUAL

2.1. Introducción

La instrumentación virtual es un concepto introducido por la compañía *Nacional Instruments*. En el año 1983, Truchard y Kodosky, decidieron enfrentar el problema de crear un software que permitiera utilizar el ordenador personal como instrumento para realizar mediciones. Tres años fueron necesarios para elaborar la primera versión del software que permitió, de una manera gráfica y sencilla, diseñar un instrumento en el ordenador. De esta manera surge el concepto de instrumento virtual, como lo definieron en *National Instruments*: "*un instrumento que no es real, se ejecuta en una computadora y tiene sus funciones definidas por software*".

A este software le dieron el nombre de *laboratory virtual instrument engineering workbench*, más comúnmente conocido por las siglas LabVIEW. A partir del concepto de instrumento virtual, se define la instrumentación virtual como un sistema de medición, análisis y control de señales físicas con un ordenador personal por medio de instrumentos virtuales. LabVIEW, por tanto, fue el primer software empleado para diseñar instrumentos en el ordenador personal y emplea una metodología de programación gráfica, a diferencia de los lenguajes de programación tradicionales. Su código no se realiza mediante secuencias de texto, sino en forma gráfica, similar a un diagrama de flujo.

Dicho concepto nace a partir del uso del ordenador como forma de reemplazar equipos físicos. El usuario opera un instrumento que no es real, realiza mediciones con él, que se ejecutan en un ordenador, pero realiza las mismas funciones que un instrumento real. El concepto de la instrumentación virtual es el de reemplazar elementos hardware por software obteniendo sus mejores ventajas e incluso mejorándolas. De esta manera el usuario final del sistema solo ve la representación gráfica de las variables manipuladas en el sistema y botones de control virtuales en la pantalla del ordenador. La instrumentación virtual implica la adquisición de señales, el procesamiento, análisis, almacenamiento, distribución y despliegue de los datos e información relacionados con la medición de una o varias señales, la interfaz hombre-máquina, la visualización, monitorización y supervisión remota del proceso, la comunicación con otros equipos, etc. Un sistema de instrumentación virtual está enfocado a los instrumentos encargados de medir señales, registrar datos y decidir las acciones de control. Evidentemente, se requiere de una etapa de actuación, que conforma la interfaz entre el ordenador y el sistema a controlar. Además, existen otras etapas auxiliares que no intervienen en el proceso de medida, como es el caso del subsistema de alimentación. En los últimos 20 años, el auge en el uso de los ordenadores ha generado un cambio en la instrumentación de ensayos, mediciones y automatización, por ello, la instrumentación virtual ofrece grandes ventajas a científicos e ingenieros que requieran mayor calidad, rendimiento y eficiencia para desempeñar su trabajo.

Resumiendo lo anteriormente explicado, un instrumento virtual consiste en el conjunto de un ordenador personal o una estación de trabajo, software de instrumentación, hardware (dependiendo de la tarea en particular) y los *drivers* que permitan al sistema operativo interactuar con los periféricos. Es decir, el PC comienza a ser utilizado para realizar mediciones de fenómenos físicos representados en señales de corriente y/o voltaje y con la instrumentación virtual se consigue involucrar la interfaz hombre-máquina, las funciones de análisis y procesamiento de señales, las rutinas de almacenamiento de datos y la comunicación con otros equipos.

2.2. Evolución de la tecnología

En sus orígenes, estos equipos, compuestos de una tarjeta de adquisición de datos con acondicionamiento de señales y el software apropiado estaban orientados a laboratorios, donde sus prestaciones eran muy requeridas por la gran precisión y capacidad de adecuar sus capacidades y cálculos de forma acorde al proceso que se estaba analizando. Con el tiempo, se fueron obteniendo cada vez soluciones y prestaciones más robustas en PC

industriales y con ello se encontraron aplicaciones en la industria, en sistemas de robótica y otras tantas aplicaciones donde la capacidad de cálculo hacía que una instrumentación tradicional no fuera capaz de responder al requerimiento del proceso.

Algunos de los beneficios que se empezaron a ofrecer con esta instrumentación virtual fueron:

- Flexibilidad.
- Bajo coste de mantenimiento.
- Reusabilidad.
- Personalización de cada instrumento.
- Rápida incorporación de nuevas tecnologías.
- Bajo coste por función y por canal.

2.2.1. Instrumentación Virtual frente a Instrumentación Tradicional

Los instrumentos virtuales son definidos por el usuario mientras que los tradicionales tienen funcionalidad fija.

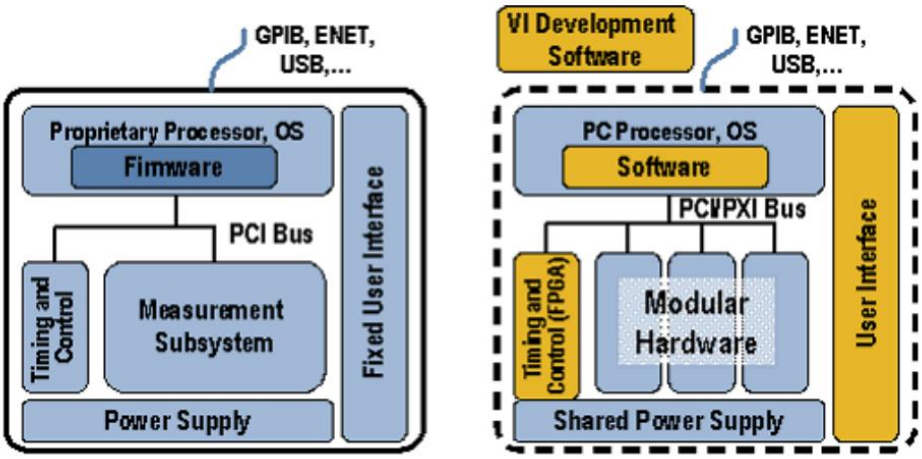


Figura 2-1. Instrumentos tradicionales frente a instrumentos virtuales.

Como se aprecia en la Figura 2-1 los instrumentos tradicionales (izquierda) e instrumentos virtuales basados en software (derecha) comparten a gran escala la misma arquitectura de componentes, pero con filosofías radicalmente diferentes. A continuación, se ven características de unos y otros en las que podemos apreciar notables diferencias:

- Instrumento tradicional:
  - Proporciona tanto software como circuitos de medición embebidos en un producto con lista finita o funcionalidad fija utilizando el instrumento del panel frontal.
- Instrumento virtual:
  - Desde una perspectiva funcional consta de dos partes, software y hardware. Al no utilizar software y hardware preestablecido se obtiene la máxima flexibilidad definida por el usuario.
  - Proporciona todo el software y hardware necesario para lograr la medición o tarea de control, es más, en un instrumento virtual se pueden integrar la adquisición, análisis, almacenamiento y funcionalidad de presentación.

- Son compatibles con los instrumentos tradicionales casi sin excepción, al revés no.
- El software de instrumentación virtual típicamente proporciona bibliotecas para crear interfaces con buses de instrumentos comunes como el GPIB o Ethernet.

2.3. Diseño de Instrumentos Virtuales

Para construir un instrumento virtual, sólo se requiere de un ordenador, una tarjeta de adquisición de datos con acondicionamiento de señales y el software apropiado. Debe realizar, como mínimo, las tres funciones básicas de un instrumento convencional: adquisición, análisis y presentación de datos.

Idealmente, una señal que es digitalizada y entregada por un instrumento virtual es la misma señal que entra al instrumento. Frecuentemente se pueden atribuir las diferencias entre las señales de entrada y salida del sistema al ruido del mismo, el cual proviene de un diverso número de fuentes, incluyendo el entorno y el mismo instrumento con el que se trabaja.

El diseño de un instrumento virtual requiere que se comprenda cómo el ruido puede afectar a la adquisición de datos, el diseño del hardware y el entorno en el que se trabaja.

Si no se entienden las preocupaciones relativas tales como la interferencia electromagnética, el manejo de la fuente de potencia, la puesta a tierra, la configuración electrónica, etc., entonces no se puede diseñar un instrumento que sea exacto dentro de un entorno eléctricamente ruidoso.

2.4. Calibración de Instrumentos Virtuales

La calibración cuantifica la incertidumbre en la medición comparando las medidas con una norma conocida, esto verifica que el instrumento se encuentra operando dentro de especificaciones establecidas.

En el pasado, los usuarios comprendieron la necesidad de calibrar instrumentos tradicionales. Los mismos principios se aplican a mediciones realizadas con ordenadores. Se deben seleccionar instrumentos virtuales que provean herramientas para realizar tanto calibraciones internas (conocidas como auto-calibraciones), como externas.

Las opciones de calibración externa e interna ofrecen dos beneficios diferentes:

- Con la calibración externa se puede asegurar que la exactitud de la medición está sujeta a una norma conocida.
- Con la calibración interna, se puede ajustar el instrumento para ser utilizado en entornos diferentes a los cuales se realizó la calibración externa.

La calibración externa requiere el uso de fuentes de alta precisión, también conocidas como patrones. Durante una calibración externa, las constantes de calibración internas y las referencias se ajustan con respecto a las constantes patrón externas. La calibración externa se reserva a laboratorios de metrología u otras instituciones que mantengan normas verificables. Una vez que se ha completado la calibración, las nuevas constantes se almacenan en un área protegida de la memoria de los instrumentos y no pueden ser modificadas por el usuario, esto protege la integridad de la calibración de la adulteración.

Debido a que la calibración interna no se basa en patrones externos, es un método mucho más simple. Con la calibración interna, las constantes de calibración del instrumento se ajustan con respecto a referencias precisas existentes en el mismo. Después de realizar la calibración externa del instrumento y colocarlo en un entorno donde las variables externas, tales como temperatura, difieren de las del entorno original, se puede utilizar este tipo de calibración.

2.5. Virtual Instrument Software Architecture (VISA)

2.5.1. Introducción

La programación de los instrumentos de medida puede ser tediosa, ya que existen diversos protocolos, que se envían y utilizan a través de diferentes interfaces y sistemas de bus (GPIB, RS232, USB). Por cada lenguaje de programación existen bibliotecas que apoyan tanto el dispositivo como su sistema de bus.

Con el fin de facilitar esta comunicación, a mediados de los años 90 se definió el estándar Virtual Instrument Software Architecture (VISA). Hoy, VISA se implementa en todos los sistemas operativos dominantes en el área comercial, tanto es así que varias empresas como Rohde & Schwarz, Keysight Technologies (antes Agilent), National Instruments, etc. se encargan de su comercialización poniendo a disposición del usuario incluso algunos paquetes de manera gratuita. Estas bibliotecas trabajan junto a dispositivos periféricos.

La especificación VISA tiene uniones explícitas a Virtual Basic, C y G (lenguaje gráfico de LabVIEW), aunque se puede utilizar con cualquier lenguaje capaz de llamar a funciones en un DLL, biblioteca de enlace dinámico (del inglés *Dinamic Link Library*), entre los que se incluye Python (con el que se trabaja en este proyecto).

2.5.2. Estándar VISA

Virtual Instrument Software Architecture es una implementación de la capa de entrada/salida de la API, interfaz de programación de aplicaciones (*Application Programming Interface*), usada en la industria de equipos de instrumentación y medida para la comunicación con instrumentos desde un PC. Dicho estándar incluye especificaciones para la comunicación con instrumentación de prueba y medida a través de buses específicos para interfaces de entrada/salida como GPIB, USB, etc. Aunque existen varias implementaciones y desarrolladores de la interfaz VISA, las aplicaciones escritas sobre esta especificación son intercambiables entre los distintos fabricantes gracias a su estandarización.

VISA se estandarizó inicialmente a través de la *XVIplug & play Alliance*, un cuerpo de estándares de test y medida actualmente extinto. El estándar actual "Especificación VISA 5.0" es mantenido por la Fundación IVI, ésta es un consorcio fundado para promover especificaciones para la programación de los instrumentos de medida, lo cual ayuda a simplificar la intercambiabilidad, proporcionar un mejor rendimiento y reducir el coste de desarrollo y mantenimiento de programas.

3 BUS GPIB

3.1. Introducción

A lo largo de la historia se ha trabajado con diferentes tipos de buses paralelos para interconectar los equipos que constituyen un entorno de instrumentación automatizado. Las principales características que se requieren en estos entornos son:

- Permitir la interconexión de un número reducido de instrumentos.
- Utilizar un estándar aceptado por la mayoría de los fabricantes.
- Ser apropiado para interconectar equipos próximos (ubicados en una habitación).
- Poseer una velocidad de intercambio de datos suficientemente alta para que la transferencia de los paquetes de datos sea inapreciable a un operador humano.

De entre los diferentes buses existentes en el mercado, uno de los más utilizados actualmente es el bus GPIB. Su mayor difusión se debió a que posteriormente a su propuesta como solución propietaria de *Hewlett Packard*, y debido a su rapidez y flexibilidad, fue adoptado por la organización IEEE. La funcionalidad del estándar GPIB ha evolucionado a lo largo del tiempo y se encuentra descrito en los siguientes apartados.

Figura 3-1. Bus GPIB.

3.2. Definición

GPIB es un bus[1] estándar desarrollado a finales de los años 60 y principio de los 70 por HP (*Hewlett Packard*), para la conexión de dispositivos de test y medida con equipos de control. El objetivo principal de este bus es gestionar la transferencia de información entre dos o más dispositivos.

Posteriormente al lanzamiento del primer dispositivo fabricado por HP conocido como HP-IB, bus de datos de *Hewlett Packard* (*Hewlett Packard Instrument Bus*), que implementaba el bus tratado, otros fabricantes lo imitaron implementándolo en sus equipos y denominándolo como *Gerenal Purpose Instrumentation Bus* (GPIB) que le dio el nombre común al bus.

Gerenal Purpose Instrumentation Bus es un protocolo paralelo de 8 bits, asíncrono, cuya arquitectura es maestro-esclavo, es decir, en la que únicamente existe un controlador del bus que es el encargado de supervisar todas las operaciones realizadas. Dicho controlador es el que gestiona cuál es el dispositivo que envía la información y en qué instante se produce el envío, para así evitar la simultaneidad de envío de varios

[1] Conjunto de líneas o cables de hardware utilizados para la transmisión de datos entre los componentes de un sistema informático.

equipos al mismo tiempo en una misma red. Además, cuenta con funciones de control de transferencia de datos o *data hardware handshake* para garantizar la recepción de los datos en los dispositivos esclavos.

3.2.1. Evolución Histórica

Cuando se quiere interconectar instrumentos de medida y controladores para formar un sistema de medición, éstos deben cumplir con alguna norma que permita su utilización independientemente de que provengan de distintos fabricantes. Las primeras propuestas de normalización surgieron de la Comisión Electrónica Internacional (del inglés *International Electrotechnical Commission*), IEC, en 1972. A partir de esta fecha, se va a hacer un pequeño análisis de los cambios fundamentales que ha ido sufriendo GPIB:

- 1972: IEC hace las primeras propuestas de normalización.
- 1975: IEEE publica la norma IEEE 488 para interconexión digital de instrumentos programables.
- 1976: ANSI, instituto de estandarización americano (del inglés *American National Standards Institute*), adopta la norma IEEE 488 como ANSI MC 1.1.
  Los miembros de la IEC sometieron a escrutinio esta norma como modelo para posteriores desarrollos.
- 1978: El IEEE publicó una versión revisada de la norma, con aclaraciones del texto que permitían una más fácil lectura y comprensión.
- 1987: IEEE modifica la norma a IEEE 488.2, que define las configuraciones mínimas, los comandos y formatos de datos básicos y comunes a todos los equipos, el manejo de errores y los protocolos que siguen en las comunicaciones.
- Hoy día: Existen alternativas de buses al GPIB que han ido reemplazandolo en aplicaciones cotidianas, sin embargo, este bus sigue siendo el estándar en aplicaciones industriales y de alto rendimiento por su versatilidad.

3.3. Características

Los datos a destacar de este bus son los siguientes:

- La baja latencia y el buen ancho de banda son la firma del bus GPIB, que cuenta con un ancho de banda de más de 1 MBytes/s, pudiéndose incrementar hasta la velocidad de 8 MBytes/s en la versión Hi-Speed (HS488). Sin embargo, el ancho de banda varía en función del número de equipos conectados a la red.
- Los dispositivos suelen conectarse mediante un cable apantallado de 24 polos. La gran robustez de los conectores IEEE 488 utilizados dota al sistema de una alta fiabilidad, con la particularidad de que dichos conectores de cada extremo son al mismo tiempo enchufe y receptáculo.
- Cada dispositivo de una red GPIB posee una dirección, codificada como un número entre 0 y 30. Es decir, existen 31 direcciones primarias permitidas para instrumentos transmisores de datos (del inglés *talkers*) conectados al bus interfaz GPIB. A cada instrumento se le asigna una dirección primaria codificada mediante 5 bits. Puede haber hasta 15 dispositivos conectados en un bus contiguo, siendo sólo uno de ellos el controlador. Es decir, a una tarjeta controladora pueden conectarse hasta 14 dispositivos, por ejemplo, encadenando cables IEEE 488 de un dispositivo al siguiente. Algunos dispositivos conectados al bus pueden direccionarse también mediante direcciones secundarias. Éstas hacen referencia a alguno de sus bloques funcionales. Por ejemplo, un osciloscopio con dirección primaria 4 puede tener un módulo de acondicionamiento de señal con dirección secundaria 1.
- La transferencia de datos es de 8 bits en paralelo. Se dedican pues 8 líneas a datos. Estas líneas son bidireccionales.
- Transferencia asíncrona de bytes en serie y bits en paralelo. Esta transferencia es controlada por las

líneas de *handshake*.

- La recepción de cada byte de datos es reconocida mediante el *handshake*. Consta de 3 líneas.
- Posee 5 funciones de comunicación primarias y 5 especializadas. La norma no exige a todos los instrumentos poseer todas las funciones.
- Existen cinco líneas de control para necesidades especiales que controlan el flujo de señal en el bus. En el posterior apartado 3.4.2. Líneas de Transmisión de señal, se detallan las funciones de las líneas del bus y su funcionamiento, con el fin de mostrar sus posibilidades de diseño fiable.
- La interconexión entre equipos se realiza utilizando cables de 24 hilos, finalizados en conectores de doble boca (macho por un lado y hembra por el otro), lo que permite la interconexión de los equipos en cualquier configuración (estrella, línea, o cualquier combinación de ellas).
- La normativa especifica que pueden usarse hasta 20 metros de cable encadenado en línea, apilando las conexiones; también recomienda no emplear tramos de cable de más de 4 metros de longitud. Una regla práctica consiste en multiplicar el número de dispositivos conectados por 2, comparar el resultado con 20 metros y tomar el menor número. Por ejemplo, si conectamos 8 dispositivos, la línea puede tener hasta 16 metros, y no se puede llegar a 20. Es decir, puede haber hasta 20 metros de cable entre el controlador y el último dispositivo, y entre cada dos dispositivos no puede haber más de 2 metros. La alternativa de conexión en estrella se emplea para aumentar el número de dispositivos interconectados, pero tiene como inconveniente la mayor capacidad parásita de la red, que introduce retardos y, en consecuencia, errores de transmisión. Al contrario que ocurre con el ancho de banda, la latencia de este protocolo es mejor incluso que la de USB 2.0, que es de 100 µs. Los cables que se comercializan son de 1, 2, 4 y 8 metros.

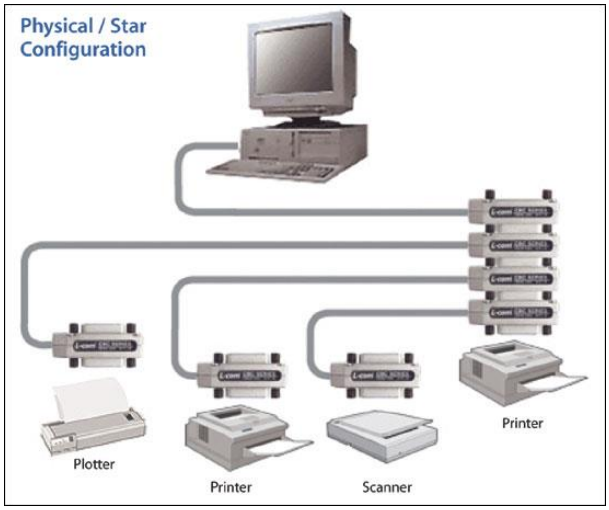


Figura 3-2. Ejemplo de configuración en estrella.

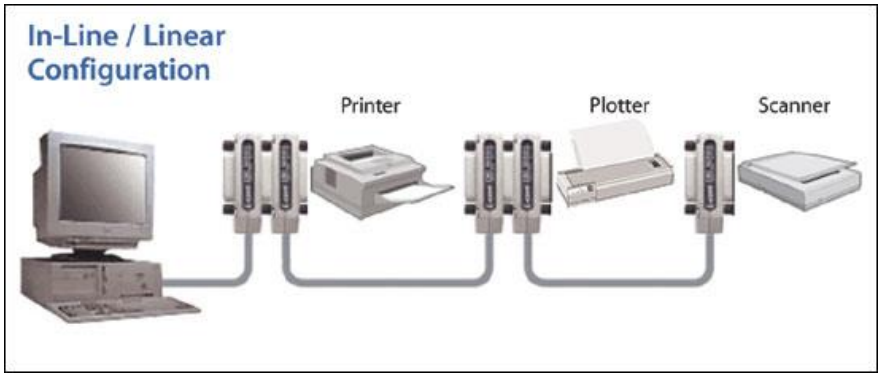


Figura 3-3. Ejemplo de configuración lineal.

3.3.1. Especificaciones

La funcionalidad del estándar GPIB ha ido evolucionando a lo largo de la historia y se encuentra descrita en las siguientes especificaciones:

- IEEE 488.1 (1975): Especificación que define las características de nivel físico (mecánico y eléctrico), así como sus características funcionales básicas.
- IEEE 488.2 (1987): Especificación que define las configuraciones mínimas, los comandos y formatos de datos básicos y comunes a todos los equipos, el manejo de errores y los protocolos que se siguen en las comunicaciones.
- SCPI, comandos estándar para instrumentos programables (*Standard Commands for Programmable Instrumentation*): Especificación construida sobre el estándar IEEE 488.2 que define una estructura de comandos estándar aceptados por múltiples instrumentos de muchos fabricantes.
- VISA: Librería que puede ser usada para desarrollar aplicaciones y *drivers* de entrada/salida, I/O, de forma que el software de diferentes empresas pueda trabajar conjuntamente sobre el mismo sistema y puedan ser instalados en conjunción con *drivers VXI plug&play* utilizando simultáneamente varios medios de comunicación (GPIB, VXI, RS232, LAN, etc.) y en aplicaciones desarrolladas con diferentes lenguajes (C, C++, VisualBasic, etc.).

3.3.2. Restricciones

Es de primordial importancia impedir que se transmitan 2 o más informaciones simultáneas a un dispositivo. Para cumplir con esas condiciones tenemos los siguientes elementos:

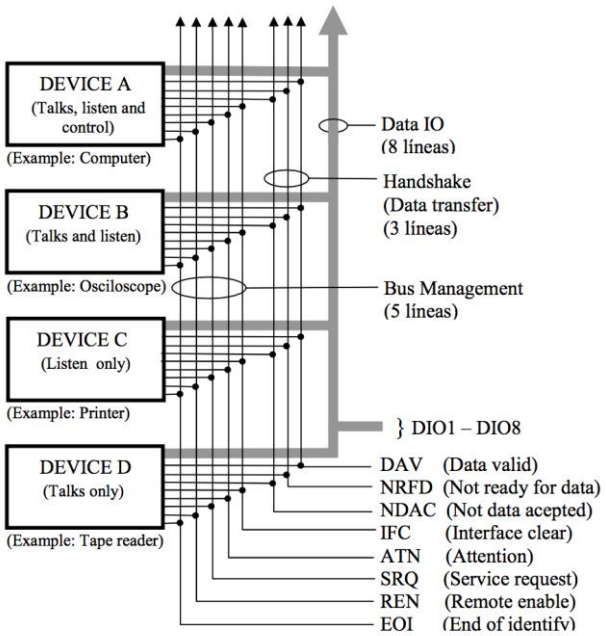


Figura 3-4. Diagrama de interconexión a través del bus GPIB.

- Transmisor (*TALKER*): Todo instrumento o dispositivo capaz de transmitir datos a través del bus, un ejemplo puede ser un voltímetro o un frecuencímetro. En cada bus pueden existir uno o varios equipos con capacidad de enviar datos a otros equipos por el bus, pero en cada instante sólo uno de ellos puede ser establecido por el *controller* para que opere como *talker*. Es el único equipo con capacidad de establecer el estado de las líneas DAV, dato válido (*Data Valid*). El equipo *talker*

sólo puede enviar un dato si todos los equipos que se encuentran en modo *listener* están en disposición de leerlo (Línea NRFD, dato no preparado (*Not Ready For Data*) a valor lógico FALSE).

- Receptor (*LISTENER*): Todo instrumento o dispositivo capaz de recibir datos digitales a través del bus, por ejemplo, una fuente programable. En cada bus pueden existir uno o varios equipos con capacidad de recibir datos desde el bus, y uno o varios de ellos se puede encontrar simultáneamente en modo *listen*. Todos los equipos que se encuentran en estado *listen* reciben simultáneamente todos los datos que son transferidos por el bus.
- *Idler*: Sin ninguna capacidad respecto del bus.
- Controlador (*CONTROLLER*): Todo instrumento o dispositivo capaz de administrar las comunicaciones a través del bus, designar los dispositivos que han de transmitir o recibir datos durante cada secuencia de medición, e interrumpir y ordenar acciones internas específicas en los dispositivos. Las tarjetas de interfaz GPIB son estos controladores. Logran gobernar el flujo de información mandando comandos a los instrumentos para que se comuniquen entre sí o respondan a pedidos de servicio de los dispositivos. Puede haber más de un controlador en un bus. Un controlador puede pasar el control del bus de sí mismo a otro controlador, pero solo uno puede tener el control del bus. Un sistema construido sobre el bus GPIB puede ser configurado en uno de los siguientes tres modos:
    - Sin uso de *controller*: En esta configuración uno de los equipos debe tener capacidad para actuar como *talker* únicamente, y los restantes solo como *listener*. La transferencia de datos posibles es desde el *talker* a todos los *listener* simultáneamente.
    - Con *controller* único: En esta configuración las transferencias de datos posibles son desde el *controller* a los equipos en modo comando y datos, de un equipo al *controller* solo en modo datos, y de un equipo a otro equipo solo en modo datos.
    - Con múltiples *controller*: En este caso tiene las mismas capacidades que la configuración anterior, solo que en ésta también es posible la transferencia entre equipos de la capacidad de operar como *controller* activo.

3.4. Transmisión de señal en el bus GPIB

3.4.1. Señales lógicas

Todas las líneas del GPIB operan con niveles de tensión TTL, lógica transistor a transistor (*Transistor-Transistor Logic*), y utilizando una lógica negativa. Quiere decir esto que un nivel de tensión que no supere el valor de 0.8 Voltios corresponde a un estado lógico TRUE, y un nivel de tensión superior a 2.5 Voltios corresponde a un estado lógico FALSE. Las puertas de salida en cada equipo sobre una línea del bus utilizan la tecnología *open-collector*, esto hace que una línea del bus estará en estado TRUE lógico (tensión baja) si ese es el valor de salida que corresponde para esa línea en algún equipo y en el estado FALSE lógico (tensión alta) si el valor corresponde en todos los equipos.

Algunas interfaces GPIB utilizan una tecnología *tri-state* para el control de las líneas del bus, pero siguiendo la misma funcionalidad que con la tecnología *open-collector*. Con esta tecnología se pueden conseguir mayores velocidades de transferencia de datos.

Dada la tecnología de control de líneas que se utiliza, aunque funcionalmente podrían conectarse a un bus más de 15 equipos (siempre que algunos de ellos sean sólo *Listener*), las prestaciones dinámicas del bus podrían deteriorarse con ello.

3.4.2. Líneas de transmisión de señal

La interfaz GPIB consta de 16 líneas. Las 8 restantes del bus (hasta 24) corresponden a líneas de retorno a tierra. De las 16 líneas, 8 son de datos (1 byte) y 8 para mensajes de control y estados de los dispositivos. De estas últimas 8 líneas, 3 son para el control de transferencia de datos (*handshake*) y 5 para el control general de

la interfaz. La Figura 3-5 muestra la estructura de las líneas del bus.

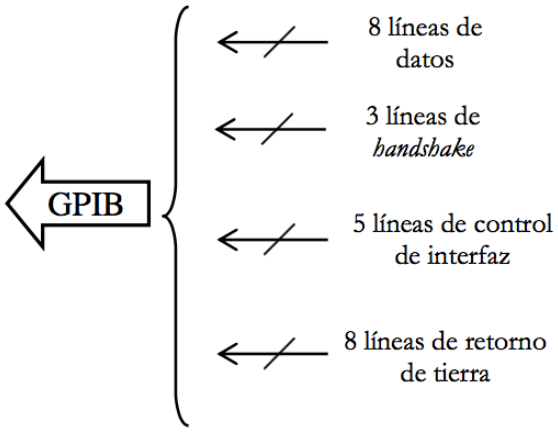


Figura 3-5. Líneas del bus GPIB.

A continuación, se describen con detalle las líneas que componen el bus GPIB:

- Líneas de datos: Las 8 líneas de datos DIO1-DIO8 pueden transportar tanto datos como órdenes. El estado de la línea de atención, ATN, (una de las 5 de control general) determina si son datos u órdenes los presentes en el bus, si está a nivel bajo son órdenes o direcciones, y si está a nivel alto son datos. Todas las órdenes y la mayoría de los datos emplean 7 bits codificados en ASCII, código estándar estadounidense para el intercambio de la información (*American Standard Code for Information Interchange*), o ISO, organización internacional de estandarización (*International Organization for Standardization*). En este caso el octavo bit se emplea para paridad o no se emplea. Permiten que el único equipo establecido como transmisor envíe un byte en paralelo hacia todos aquellos equipos que en ese instante estén definidos como receptor.
- Líneas de control de transferencia de datos (*handshake*): Estas 3 líneas realizan el control asíncrono de las transferencias de los mensajes en forma de byte entre los dispositivos. Este proceso garantiza que la transmisión y la recepción se han realizado sin errores, dotando a la transmisión de información de seguridad. Las líneas son:
    - NRFD: Indica cuándo un dispositivo está preparado para recibir un byte. La línea es conducida por todos los dispositivos cuando reciben órdenes y por el transmisor de datos cuando habilita el protocolo HS488. Gobernada por los equipos establecidos en escucha. Cuando esta línea está a nivel de tensión bajo, significa que algún equipo de entre los receptores no está aún dispuesto para aceptar nuevos datos. En esta situación la línea inhibe al equipo transmisor a que inicie el envío de un nuevo dato. El que esta línea esté a nivel de tensión alto, significa que todos los equipos receptores se encuentran a la espera de un nuevo dato, momento en el que el transmisor puede establecer el dato en el bus.
    - NDAC, dato no aceptado (*Not Data Accepted*): Indica cuándo un dispositivo ha aceptado un mensaje (en forma de byte). La línea es conducida por todos los dispositivos a recibir órdenes y/o datos cuando reciben la información y es gobernada por los equipos que están establecidos en escucha. Cuando se encuentra a nivel de tensión bajo, significa que alguno de los equipos establecidos como receptor aún está pendiente de leer un dato, en consecuencia, el transmisor debe esperar aún para retirar los datos. Cuando esta línea se encuentra a nivel de tensión alto significa que todos los equipos receptores han leído el dato transferido y por tanto el transmisor puede retirar el dato del bus.
    - DAV: Indica cuándo las señales en las líneas de datos se han estabilizado (se consideran válidas) y pueden ser aceptadas con seguridad por los dispositivos. El controlador conduce la línea de datos al enviar órdenes y los transmisores la conducen cuando envían mensajes de datos, además es gobernada por ellos. Un nivel bajo de tensión en esta línea significa que el equipo establecido como transmisor activo ha establecido unos datos

válidos sobre el bus de datos que deberán ser leídos por todos los equipos que están en escucha.

- Líneas de control general de la interfaz. A dicho grupo pertenecen:
    - SQR, petición de servicio (*Service Request*): Se emplea para solicitar turno al controlador. Cuando el controlador detecta una petición en esta línea, debe iniciar una encuesta (*polling*) para determinar qué equipo causó el requerimiento, y en el caso de que proceda, satisfacer su demanda.
    - IFC, reseteo de interfaz (*Interface Clear*): Realiza un *reset* (puesta a valor nulo) de los parámetros y direcciones del bus cuando en la línea se establece un nivel de tensión baja, además todos ellos deben pasar a su estado base. Bajo el exclusivo control del *system controller*.
    - REN, habilitación remota (*Remote Enable*): Establece el control remoto de un dispositivo deshabilitando su control a través del panel. Es decir, el usuario ya no podrá realizar un control manual.
    - EOI, fin de identidad (*End Or Identity*): Se emplea por el transmisor como indicador de fin de transferencia de datos. Además, esta línea es utilizada por el controlador para iniciar una encuesta paralela. Debe poner simultáneamente a un nivel de tensión bajo las señales ATN y EOI, y como respuesta a ello, los equipos que previamente hayan sido configurados para participar en la encuesta paralela transfieren su bit de *status* sobre el bus.
    - ATN, línea de atención (*Attention*): La emplea el controlador para distinguir entre los datos y los mensajes de control. Determina el tipo de información presente en las líneas de datos. Cuando la línea toma un nivel de tensión bajo equivale a que el dato que se envía por el bus de datos es un comando enviado por el controlador. Mientras que si toma un nivel alto indica que el byte del bus de datos debe ser considerado como un dato.

3.5. Normas internacionales del bus GPIB

En este apartado se va a mostrar mediante la Figura 3-6 las diferencias entre los diferentes tipos de buses GPIB que se usan en la actualidad.

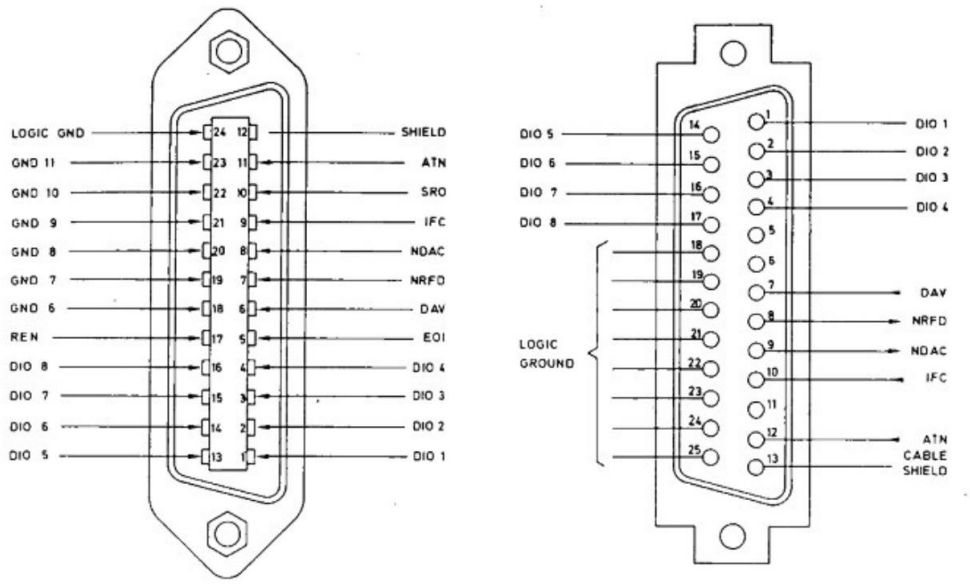


Figura 3-6. Conector GPIB norma americana (izquierda) y norma europea (derecha).

Los mensajes utilizados para el diálogo de comunicación entre los diferentes dispositivos a través del bus sí son universales y se muestran en la Tabla 3-1.

Más adelante, en el apartado 3.8. Comandos, se explica brevemente la utilidad de cada uno y su respectiva división en categorías dependiendo de la funcionalidad que pueden tener.

Tabla 3–1. Mensajes de comando en el bus GPIB.

MNEMONIC	***MESSAGE DESCRIPTION***	***COMMENTS***
ATN	*Attention*	*Received*
DAC	*Data accepted*	*Received or Sent*
DAV	*Data Valid*	*Receivedor Sent*
DCL	*Device Clear*	*Received*
GET	*Group Executive Trigger*	*Received*
IFC	*Interface Clear*	*Received*
MLA	*My Listen Address*	*Received*
MTA	*My Talk Address*	*Received*
OTA	*Other Talk Address*	*Received*
RFD	*Ready For Data*	*Received or Sent*
SDC	*Selected Device Clear*	*Received*
SPD	*Serial Poll Disable*	*Received*
SPE	*Serial Poll Enable*	*Received*
SRQ	*Service Request*	*Sent*
UNL	*Unlisten*	*Received*
UNT	*Untalk*	*Received*

3.6. Componentes que se utilizan en el control de la instrumentación por GPIB

- Equipo: Instrumento del entorno que se controla. Debe estar dotado con una tarjeta hardware de conexión al bus GPIB. Para su control dispone de software interno de control que interpreta los mensajes que recibe por el bus GPIB e interacciona con el *firmware* propio del equipo. El *parser* es el hilo de gestión del intercambio de mensajes por el bus GPIB.
- Nivel Físico (IEEE 488.1) : La comunicación entre los equipos se basa en un bus físico, compuesto por un conjunto de líneas con niveles lógicos bien definidos y con protocolos de comunicación basados en los estados lógicos de las líneas.
- Nivel Operativo (IEEE 488.2): El protocolo operativo básico dentro del que se encuadra el

intercambio de información, datos e instrucciones básicas de control.

- *Driver* GPIB (SCPI): El ordenador interacciona con el bus GPIB a través de una tarjeta de control hardware que resuelve y atiende los dos protocolos anteriores. El propio fabricante ofrece una interfaz software implementada por un conjunto de funciones que permiten el acceso de los programas a la funcionalidad del bus. Puede ofertar una interfaz constituida por una librería de funciones que corresponde al lenguaje SCPI.
- *Driver* VISA: Estándar de *driver* que ofrece un conjunto de librerías estandarizadas que permiten integrar equipos conectados por diferentes medios de comunicación.

3.7. Protocolos del GPIB

3.7.1. Introducción

En informática y telecomunicaciones, un protocolo es una convención, o acuerdo entre partes que regulan la conexión, la comunicación y la transferencia de datos entre dos sistemas. En su forma más simple, un protocolo se puede definir como las reglas que gobiernan la semántica, la sintáxis y la sincronización de la comunicación.

Los protocolos pueden estar implementados bien en hardware, software o una combinación de ambos.

3.7.2. Registros de Estado

La norma contempla tres registros de estado. Además, cada uno de ellos tiene un registro de habilitación/deshabilitación que permite enmascarar ciertos bits. Los tres registros son:

- *Standard Event Status Register* (8 bits), contiene las siguientes informaciones:
  - *Operation Complete* (operación completa).
  - *Request Complete* (petición completa).
  - *Query Error* (error de pregunta).
  - *Device Error* (error de dispositivo).
  - *Execution Error* (ejecución de error).
  - *Command Error* (comando de error).
  - *Power On* (alimentación encendida o apagada).
- *Questionable Data* (16 bits), contiene las siguientes informaciones:
  - *Voltage Overload* (caso de sobretensión).
  - *Current Overload* (caso de sobrecorriente).
  - *Ohms Overload* (sobreresistencia).
  - *Limit Test Fail LO* (fallo de la prueba de límite inferior).
  - *Limit Test Fail HI* (fallo de la prueba de límite superior).
- *Status Byte* (8 bits), registro de estado ya existente en la forma IEEE 488.1 que contiene los siguientes bits:
  - *Questionable Data*: Si alguno de los bits del registro *Questionable Data* está activo y habilitado.
  - *Message Avaliable*: Si hay un mensaje que el instrumento quiere enviar. Tiene una cola de salida (*output buffer*) donde están los posibles mensajes a enviar.

- *Standard Event*: Si alguno de los bits del registro *Standard Event Status Register* está activo y habilitado.
- *Request Service* (bit sin máscara): Si el instrumento ha perdido o no una petición al controlador.

3.7.3. Protocolo básico

Se indican a continuación algunas consideraciones del equipo y controlador de un sistema de bus GPIB:

- Los comandos son ejecutados en el orden en que han sido recibidos.
- El equipo y el controlador se comunican intercambiando mensajes de órdenes y de respuesta. Esto significa que el controlador siempre debe terminar un mensaje de órdenes, antes de intentar leer una respuesta.
- Equipo:
  - La regla básica del protocolo es: El equipo solo habla cuando está dispuesto a ello, y en ese caso, tiene que hablar antes de que se le ordene hacer una cosa nueva.
  - Solo habla como respuesta de una orden de requerimiento.
  - Cuando se enciende el equipo, o cuando recibe una orden de inicialización (*CLS), se inicializan los *buffers* de entrada y de salida, y el *parser* se inicializa a la raíz de su árbol de órdenes.
- Controlador:
  - Envía los mensajes de órdenes y pueden ser de dos tipos:
    - Órdenes de control: requieren un cambio de estado del equipo, pero no requieren ninguna respuesta.
    - Órdenes de requerimiento: solicitan información sobre el estado del equipo o sobre información que posee.
  - Puede enviar un mensaje conteniendo múltiples órdenes de requerimientos. A esto se le denomina un requerimiento compuesto. Los diferentes requerimientos dentro del mensaje deben estar separados por el delimitador ‘;’. Los mensajes de respuesta son encolados en la cola de salida, separados entre sí también por el delimitador ‘;’.
  - Solo admite un mensaje de salida (respuesta de una orden de requerimiento), y lo requiere antes de enviar un nuevo mensaje de órdenes. En caso contrario se genera una situación de bloqueo.
  - Si el equipo envía un mensaje de respuesta, el controlador debe leer siempre de forma completa el mensaje, antes de que envíe un nuevo mensaje de órdenes al equipo.

3.7.4. Protocolos de excepción

Cuando se produce un error en el proceso de intercambio de la información, éste no termina en la forma normal, sino que sigue un protocolo de excepción. En las siguientes líneas se indican los diferentes casos por los que puede dar problemas y la forma de acabar que tiene el sistema:

- Equipo direccionado para hablar sin nada en la cola:
  - Si es consecuencia de que el equipo no haya recibido una orden de requerimiento, el equipo indicará un error de encolamiento, y no enviará ningún byte por el bus.
  - Si es como consecuencia de que la orden de requerimiento previa no se ejecutó debido a un error, el equipo no indica ningún error de encolamiento, sino que espera a recibir el siguiente mensaje del controlador.

- Equipo direccionado para hablar sin que ninguno escuche: Esperará o bien a que algún equipo escuche, o a que el controlador tome el control.
- Error de orden: Se genera cuando se detecta un fallo de sintaxis o una orden no reconocible.
- Error de ejecución: Se genera si un parámetro está fuera de rango, o si el equipo se encuentra en un estado que no permita la ejecución del comando requerido.
- Error específico del equipo: Se produce cuando el equipo es incapaz de ejecutar una orden, como consecuencia de una razón estrictamente dependiente de él, y no del protocolo seguido por el bus.
- Error de encolamiento: Se genera si no se sigue el protocolo de lectura de los datos de la cola de salida.
- Condición inconclusa: Si el controlador intenta leer un mensaje de respuesta antes de que el programa haya concluido de ejecutar la orden que los genera. En este caso el *parser* se inicializa a sí mismo, la respuesta ya elaborada es limpiada de la cola de salida y ningún dato es transferido por el bus.
- Condición interrumpida: Si el controlador no lee completamente el mensaje generado por un mensaje de requerimiento, y envía otro mensaje de orden, el equipo genera un error de encolamiento, y el segmento de mensaje de salida no leído es eliminado. La orden que interrumpe es inafectada.
- Bloqueo de *buffer*: El equipo alcanza un estado de bloqueo si el *buffer* de entrada está lleno y también resulta llena la cola de salida. Esta situación ocurre si se envía un mensaje muy largo que contiene órdenes de requerimiento, y genera un mensaje de salida superior al que puede contener la cola. El controlador no puede terminar de enviar el mensaje de entrada porque no cabe y el *buffer* de entrada no se vacía porque espera que su cola de salida sea vaciada para concluir la orden en ejecución. En este caso el equipo rompe el bloqueo, limpiando la cola de salida y generando un error de encolamiento.

3.7.5. Estado de un equipo

El estándar IEEE 488.2 ofrece un mecanismo estandarizado de presentar y mostrar el estado interno del equipo. A través de este mecanismo, se puede tener información de si el equipo tiene un dato dispuesto para transferir, así como si ha ocurrido algún tipo de error. El mecanismo se basa en cuatro registros:

- SBR, registro de estado de bytes (*Status Byte Register*): Cada bit está asociado con un tipo de estado específico del instrumento. Cuando el estado cambia, el instrumento establece el correspondiente bit a 1. Se puede habilitar e inhibir el efecto de cada bit del SBR a efectos de requerir atención (bit SQR), con el correspondiente bit del registro SRER. Se puede determinar qué eventos han ocurrido leyendo los establecidos en el registro SBR.
- SRER, registro de solicitud de servicio (*Service Request Enable Register*): Máscara de los bits correspondientes del registro SBR que va a determinar si se establece el *Request Service* (bit SQR).
- ESR, registro de evento de estado (*Event Status Register*): Cada bit está asociado con un tipo específico de evento. Cuando un evento ocurre, el instrumento establece el correspondiente bit a 1. Se pueden habilitar o inhibir los eventos que van a ser proyectados sobre el bit ESR a través de los correspondientes bits de máscara de registro ESER. Se puede leer el evento que ha ocurrido leyendo el contenido del registro ESR.
- ESER, registro de habilitación de estados (*Event Status Enable Register*): Máscara de los bits del registro ESR que van a requerir el servicio a través del registro SBR.

3.8. Comandos

Los comandos de bus son siempre enviados desde el controlador a los otros equipos para sincronizar su estado de operación o para establecer su estado de operación.

En los comandos de bus se envían datos por el bus de datos, si es necesario, de igual modo que en la transferencia de datos, solo que en estos casos la señal ATN es establecida a nivel de tensión bajo por el controlador para indicar que es un comando, y todos los equipos con independencia de que sean transmisores o receptores reciben el comando.

El controlador puede enviar cinco tipos de comandos de bus a los otros equipos: *addressed*, *listen*, *talk*, *universal* y *secondary*. Solo los 7 bits menos significativos del bus son utilizados en los comandos de bus. Los tres bits b7, b6 y b5, definen la naturaleza de cada comando.

Tabla 3–2. Naturaleza de cada comando.

b7	b6	b5	Tipo Comando
0	0	0	*Addressed*
0	1	x	*Listen*
1	0	x	*Talk*
0	0	1	*Universal*
1	1	x	*Secondary*

Los equipos conectados al bus GPIB tienen asignado un código o dirección de bus comprendido entre 0 y 30, este código debe ser establecido en cada equipo. De tal forma que el código 31 queda reservado para hacer referencia a todos los equipos del bus simultáneamente. Normalmente, los cinco bits menos significativos de la línea de datos se utilizan en un comando para establecer a qué equipos hace referencia un comando.

3.8.1. Comandos *Addressed*

Van destinados y afectan únicamente a aquellos equipos que previamente han sido establecidos como receptores:

- GTL (*Go To Local*): Retorna el control de los paneles a todos los equipos en estado *Listen*.
- SDC (*Select Device Clear*): Inicializa las interfaces hardware/software en los equipos en estado *Listen*.
- PPC (*Parallel Poll Configure*): Configura la respuesta a una encuesta paralela de los equipos en estado *Listen*. Los equipos quedan a la espera de un comando MSA.
- MSA (*My Secondary Address*): Establece la línea y el estado con la que responden en una encuesta paralela de los equipos receptores.
- GET (*Group Trigger*): Dispara el *Trigger* de los equipos en estado *Listen*.
- TCT (*Take Control*): Establece como controlador activo al equipo que está establecido como receptor.

3.8.2. Comandos *Talk/Listen*

Este grupo está formado por los siguientes commandos:

- MTA (*My Talk Address*): Establece el modo *Talker* en el equipo indicado en la dirección binaria del comando.
- UNT (*UnTalk*): El equipo en modo *Talker* pasa a modo *Idler*.
- MLA (*My Listen Address*): Establece el modo *Listen* en el equipo indicado.
- UNL (*UNListen*): Todos los equipos en modo *Listen* pasan a modo *Idler*.

3.8.3. Comandos Universal

Mensajes enviados por el controlador a todos los equipos:

- LLO (*Local Lockout*): Se deshabilitan los paneles de control de todos los equipos conectados al bus.
- DCL (*Device Clear*): Inicializa las interfaces hardware/software de los equipos.
- PPU (*Parallel Poll Unconfigure*): Se cancela la programación previa de los equipos a fin de responder en la encuesta paralela.
- SPE (*Serial Poll Enable*): Habilita a todos los equipos a fin de que respondan a la encuesta serie.
- SPD (*Serial Poll Disable*): Deshabilita el modo de encuesta serie.

3.8.4. Comandos Comunes

Los comandos comunes son todos aquellos que son iguales para todos los dispositivos y los especiales son diferentes para cada uno.

- CLS (*Clear Status Command*): Despeja el registro de estado y los registros de incidencia.
- ESE (*Event Status Enable Command*): Habilita bits del registro de habilitación de incidencias.
- ESE? (*Event Status Enable Query*): Interroga el registro de habilitación de incidencias estándar.
- ESR? (*Event Status Register Query*): Interroga el registro de incidencias estándar.
- IDN? (*Identification Query*): Identifica el tipo de instrumento y versión software.
- LRN? (*Learn Device Setup Query*): Requiere el estado actual del equipo.
- OPC (*Operation Complete Command*): Fija el bit de "Operación Completa" del registro estándar.
- OPC? (*Operation Complete Query*): Responde con '1' si se han ejecutado órdenes previas.
- OPT? (*Option Identification Query*): Requiere la opción instalada en el equipo.
- RCL (*Recall Command*): Restaura el estado del equipo del registro *save/recall*.
- RST (*Reset Command*): Sitúa al equipo en el estado básico de referencia.
- SAV (*Save Command*): Almacena el estado actual en un registro *save/recall*.
- SRE (*Service Request Enable Command*): Habilita los bits del registro de habilitación de Byte de estado.
- SRE? (*Service Request Enable Query*): Requiere el contenido del registro SER de habilitación del Byte de estado.
- STB? (*Read Status Byte Query*): Requiere el estado del registro resumido del Byte de estado.
- TRG (*Trigger Command*): Arranca o dispara la operación del equipo de forma remota.
- TST? (*Selft-Test Query*): Requiere el resultado del *autoset* del equipo.
- WAI (*Wait-to-Continue Command*): Espera a que se realicen todas las operaciones pendientes.

3.8.5. Comandos SCPI

A pesar de los estándares IEEE 488.1 y IEEE 488.2, hace unos años existía libertad para que cada fabricante eligiera los comandos de sus instrumentos. En 1990, un grupo de empresas fabricantes de instrumentos acordaron crear un conjunto de órdenes con una sintaxis común, que fue llamada SCPI, comandos estándar para instrumentos programables (*Standard Commands for Programmable Instrumentation*). Lógicamente, SCPI se construyó respetando los principios del anterior estándar IEEE 488.2. A continuación se ven algunas de las características que destacaron de estos comandos:

- Si dos instrumentos de fabricantes distintos se adhieren al estándar SCPI es posible conectarlos con mínimas modificaciones en el programa de control.
- Tienen estructura jerárquica por niveles y están separados por dos puntos.
- Los caracteres en mayúsculas son necesarios para especificar una orden, mientras que los que están en minúsculas pueden suprimirse, sirviendo sólo para facilitar la lectura de programas por el usuario. Los comandos en sí pueden ser escritos inidistintamente en mayúsculas o en minúsculas.
- Los comandos se pueden concatenar utilizando punto y coma (;).
- La principal ventaja de estos comandos es la definición homogénea de comandos para todos los aparatos de una misma clase.

3.9. Funcionamiento. Transferencia de datos

3.9.1. Síntesis

El estándar IEEE 488.1 establece un proceso de negociación de 3 líneas de control de datos. La introducción de una lógica negativa permite la implementación de la función lógica OR mediante cableado. De esta forma, no se transmiten datos hasta que no esté listo el receptor (*listener*) más lento, y queda asegurado que la transmisión sea lo suficientemente lenta como para que al receptor más lento le dé tiempo a aceptar el dato.

La línea NRFD es controlada por cada receptor e indica si cada uno de ellos no está listo (nivel bajo) o lo está (nivel alto) para recibir datos. La línea DAV es controlada por el transmisor e indica si los datos en las líneas de datos (DIO) son correctos y, en consecuencia, pueden ser aceptados por los receptores. Finalmente, la línea NDAC es controlada por cada receptor para indicar que no ha recibido los datos (nivel bajo) o que los ha recibido (nivel alto).

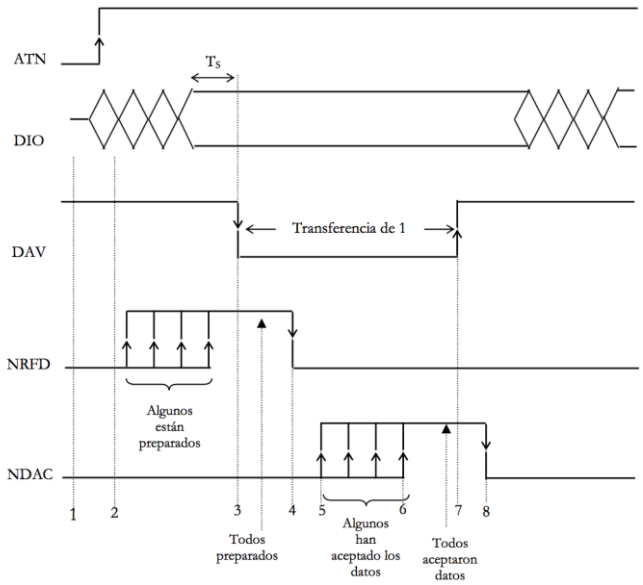


Figura 3-7. Diagrama de tiempos de operación. T_s es el tiempo de estabilización de los datos.

La Figura 3-7 muestra el diagrama de tiempos de operación. En principio, el transmisor comprueba que las líneas NRFD y NDAC están a nivel bajo. La primera indica que no todos los receptores están listos para recibir datos y la segunda indica que no han aceptado ningún nuevo byte. Obsérvese que la línea NRFD no pasa a nivel alto hasta que todos los receptores están listos. Una vez que el transmisor ha detectado que la línea NRFD está a nivel alto y transcurre cierto retardo, necesario para dar tiempo a estabilizar los niveles de los datos que envía a los receptores, pone la línea DAV a nivel bajo indicando que los datos que envía son válidos. Se transfiere así un byte de datos.

El receptor más rápido pone la línea NRFD a nivel bajo con el fin de indicar que no está listo para recibir otro byte, los demás harán lo mismo cada uno a su ritmo. Es decir, el receptor más rápido indica al equipo que no mande más información porque él ha tomado ya la que había y tiene que aceptarla o procesarla (es posible que se requiera de él una respuesta).

Finalmente, los receptores van aceptando el byte poniendo a nivel alto sus líneas NDAC. Cuando todos han aceptado los datos, la línea pasa a nivel alto, el transmisor lo detecta y pone la línea DAV a nivel alto para indicar que ya no valen los datos. El primer receptor que detecta que la línea DAV ha pasado a nivel alto pone la línea NDAC a nivel bajo. El transmisor pondrá otros datos nuevos en las líneas DIO y comienza otro nuevo ciclo.

3.9.2. Esquema de envío de información en GPIB

La función *send* se puede especificar en el terminador del mensaje mediante la entrada *mode* (0: sin terminador; 1: <NL> + EOI; 2: EOI). Si el instrumento que tenemos es compatible con IEEE 488.2 no habrá problema, pero si no lo es debemos averiguar qué tipo de terminador requiere y suministrárselo, de lo contrario no entenderá que hemos acabado la comunicación.

3.10. Otros métodos de control remoto

Además del método estudiado en el apartado anterior, que se ha desarrollado con detenimiento porque es con el que se trabaja en este proyecto, existen otros que se utilizan por igual en el mercado laboral. De ellos se van a nombrar y definir los más usuales.

3.10.1. LAN

Introducción:

Las redes de área local (*Local Area Network*) permiten compartir recursos físicos (impresoras, *router* de acceso a Internet, etc.) o lógicos (programas, etc.) a los usuarios de un área determinada. La utilización de LAN facilita además el mantenimiento, gestión y seguridad de los equipos informáticos englobados en dicha red.

Las primeras redes LAN operativas comenzaron a utilizarse en entornos ofimáticos sobre mediados de los 80. A mediados de los 90 se popularizó su utilización debido a la disminución del precio de la electrónica utilizada y actualmente se emplean también en entornos residenciales.

Una red de área local se puede representar como una nube a la que se conectan todas las estaciones de la misma, donde cada una de dichas estaciones puede enviar y recibir paquetes de cualquier otra. Hoy en día existen multitud de tipos de estaciones diferentes, como PC, servidores, impresoras, teléfonos IP, etc.

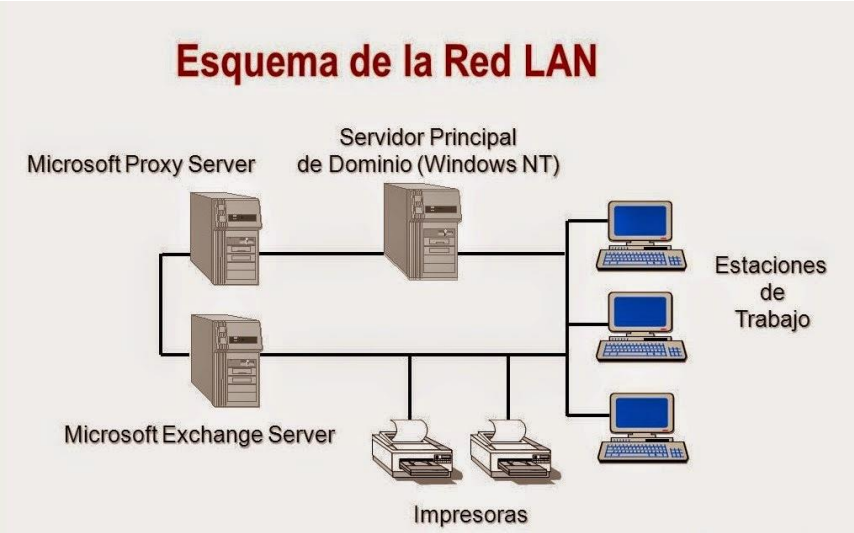


Figura 3-8. Esquema de una Red de Área Local (LAN).

La norma IEEE 802 se consolida como la más relevante en el campo de las LAN, donde se encuentran estandarizadas diferentes tecnologías pertenecientes a dicho grupo, como son Ethernet, Token Ring, WiFi, Bluetooth, etc.

El término LAN puede referirse a un gran número de tecnologías cuyas propiedades más destacadas son:

- Múltiples sistemas conectados a un medio compartido (en el caso inalámbrico es el aire). El medio compartido cableado disminuye el coste de la instalación, aunque la tendencia actual es la contraria por motivos de eficiencia y ancho de banda.
- Gran capacidad de transmisión: en el caso de medio compartido, el ancho de banda se reparte entre todas las estaciones.
- Bajo retardo y tasa de error de transmisión pequeña.
- Capacidad de difusión.
- Limitación en la extensión geográfica y en el número de estaciones.
- Relación de igualdad entre equipos conectados:
    - Todos deben tener la misma oportunidad de transmitir y el destino puede ser cualquier otro equipo dentro de la LAN.
    - Todos tienen normalmente el mismo nivel jerárquico, por lo que el concepto maestro-esclavo no se aplica para coordinar el acceso al medio compartido.
- Son de propiedad privada, por lo que no se encuentran reguladas por la administración.

Control remoto:

LAN es ideal para sistemas distribuidos y control remoto de equipos en gran parte debido a que tiene una amplia disponibilidad en los ordenadoraes de hoy día. La popularidad de esta tecnología ha crecido gracias a su uso como bus de comunicaciones para instrumentos independientes porque permite soportar largas distancias de cable utilizando conmutadores, *routers* y repetidores como ayuda. Los usuarios no tienen que mantener la instrumentación local y pueden distribuir instrumentos independientes a través de una red. Además, la especificación VXI-11 proporciona un conjunto estándar de protocolos para la comunicación con instrumentos a través de TCP/IP.

En aplicaciones de envío de datos intensivos, la comunicación LAN requiere un procesamiento significativo, ya que la pila de protocolos es implementada en software. Una regla útil es la regla *bit per Hertz*, una

estimación aproximada del procesamiento de CPU requerido para manejar una velocidad de enlace en Ethernet dada es, para cada bit por segundo de datos de red procesados, se requiere un hercio de procesamiento de CPU. Usando esta regla, por ejemplo, una comunicación mediante LAN de Gigabit ocupa aproximadamente un Gigahercio de procesamiento en un procesador de un ordenador actual cuando se transmite a velocidad completa. Por tanto, en sistemas de alta velocidad, la CPU puede tener que dedicar más procesamiento al enlace de comunicaciones que a la aplicación real. Esto puede ser un cuello de botella en sistemas que requieren un alto rendimiento de datos.

La sobrecarga de procesamiento en LAN puede aumentar el coste de un instrumento LAN de dos maneras. En primer lugar, en sistemas de alta velocidad, es probable que se requiera un procesador de clase servidor para procesar la pila TCP/IP. En segundo lugar, cuando las velocidades de datos en tiempo real no puedan lograrse a través de LAN, el diseñador debe insertar el procesamiento para la reducción de datos en el instrumento. Esto aumenta los costes y también reduce la flexibilidad del usuario.

Otra debilidad de LAN es la configuración necesaria para establecer una red. LAN requiere una dirección IP y una configuración de red, que puede estar sujeta a las políticas de Tecnología de la Información, IT (del inglés *Information Technology*), de la red en la que está instalado. De hecho, muchos de los beneficios de los diagnósticos remotos de un instrumento de LAN pueden ser negados por la política de IT particular de una compañía con respecto a los *firewalls* y otras redes de seguridad.

LXI:

En 2005, un grupo de proveedores publicó una especificación para el estándar llamado LXI, con el que se añadieron algunas características adicionales a los instrumentos LAN independientes, como una página de configuración HTML estándar y varias prácticas recomendadas para implementar instrumentos LAN. LXI también agregó funciones opcionales de sincronización incluyendo el protocolo de tiempo de precisión IEEE 1588 y un disparador de equipo en bus.

IEEE 1588 permite la sincronización a través de una red LAN. Utilizando equipos de LAN especializados, los dispositivos IEEE 1588 son capaces de lograr la sincronización en un tiempo de 100 ns. Esta capacidad hace que IEEE 1588 sea atractivo para aplicaciones con velocidades de adquisición relativamente bajas (por debajo de 1 MS/s) que requieren sincronización a grandes distancias.

La mayoría de los instrumentos LXI son muy similares a las implementaciones LAN existentes, de hecho, la mayoría de los actuales dispositivos LXI son versiones actualizadas de los productos LAN.

3.10.2. PXI

Se trata de una plataforma de ordenadores para sistemas de medidas y automatización. PXI combina características de bus eléctrico PCI[2] con el paquete *Eurocard modular*[3] y añade buses de sincronización especializada y características clave de software. PXI es una plataforma de alto rendimiento y bajo coste de implementación para aplicaciones de pruebas de manufactura, industriales, militares, aeroespaciales y monitorización de máquinas.

Desarrollado en 1997, es un estándar abierto en la industria gobernado por PXISA (*PXI Systems Alliance*), un grupo de más de 70 compañías cuyo objetivo es promocionar el estándar PXI, asegurar su interoperabilidad y mantener su especificación.

[2] *Peripheral Component Interconnect*, que en español significa Interconexión de Componentes Periféricos y que consiste en una conjunción de circuitos integrados o bien de tarjetas de expansión que se ajustan a los conectores de este tipo.
[3] Formato estándar europeo para placas de circuito impreso, que pueden conectarse a subbastidores estandarizados.

Figura 3-9. Sistema PXI con controlador embebido.

PXI está diseñado para aplicaciones de medición y automatización que requieren alto rendimiento y un robusto factor de forma industrial.

La mayoría de los módulos de los instrumentos PXI son productos basados en registros, que utilizan controladores de software alojados en un PC para configurarlos como instrumentos útiles, aprovechando el poder creciente de los ordenadores para mejorar el acceso al hardware y simplificar el software. La arquitectura abierta permite que el hardware se puede reconfigurar para proporcionar nuevas instalaciones y características que son difíciles de emular con los actuales.

Se pueden seleccionar los módulos a partir de un número de vendedores e integrarlos en un único sistema PXI. También hay empresas especializadas en la escritura de software para estos módulos, así como otras que prestan servicios de integración de hardware-software.

Es una tecnología con garantía de futuro y está diseñada para ser simple y rápida, gracias a que realiza una reprogramación de los requisitos de pruebas, medidas y automatización cuando éstos sufren algún cambio.

3.10.3. Comparativa de métodos de control remoto

Actualmente, hay sólo unos pocos cientos de instrumentos LAN en comparación con más de 10.000 instrumentos controlados por GPIB. LAN se utiliza principalmente en sistemas en los que es necesaria una larga distancia entre los instrumentos. Para aplicaciones de escritorio, GPIB es el más utilizado, y en validación y producción, GPIB y sistemas modulares como PXI son las opciones más populares.

4 PYTHON

4.1. Introducción

En este apartado se ilustará al lector acerca del lenguaje de programación empleado para la realización del presente proyecto. No se pretende publicar un manual de referencia, sino una introducción y una breve descripción del mismo. De esta forma, se busca una mejor comprensión de los programas diseñados para llevar a cabo la finalidad de este trabajo, controlar remotamente instrumentación para radiocomunicación.

4.2. Definición

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses "*Monty Python*". Posee una sintaxis muy limpia y que favorece un código legible.

Es un lenguaje poderoso y fácil de aprender, cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para escribir *scripts*[4] y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas. En el siguiente apartado se explican sus características más notables.

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas y puede distribuirse libremente. Se pueden encontrar también sin dificultad distribuciones y enlaces de muchos módulos, programas, herramientas y documentación adicional.

El intérprete de Python puede extenderse con nuevas funcionalidades y tipos de datos implementados en C u otros lenguajes accesibles desde el mismo, además puede usarse como lenguaje de extensiones para aplicaciones personalizables.

4.2.1. Evolución histórica

A continuación se analizan los cambios más simbólicos que ha ido experimentando Python y sus equipos de desarrolladores desde su creación:

- 1991: Lanzamiento inicial de Python por Van Rossum. En esta etapa del desarrollo ya estaban presentes clases con herencia, manejo de excepciones, funciones y algunos tipos modulares.
- 1994: Se forma el foro de discusión principal de Python, marcando un hito en el crecimiento del grupo de usuarios que hacen uso de este lenguaje. Python alcanza la versión 1.0 en enero el mismo año.
- 1995: Van Rossum continuó su trabajo en Python en la *Corporation for National Research Initiatives* (CNRI) en Virginia donde lanzó varias versiones del software.
- 2000: El equipo principal de desarrolladores de Python cambió su sede para formar un nuevo equipo de trabajo, el conocido como PythonLabs. En él, se publicó Python 2.0.
- 2007: Los desarrolladores de Python comienzan un proceso para hacer de él un lenguaje fácil de aprender y no muy arcano en su sintaxis y semántica, que esté al alcance de los "no-programadores".
- Hoy: Existen versiones de Python 3.x, en ellas todo lo nombrado en las entradas anteriores se ha conseguido (en el siguiente apartado se explica con detalle sus principales ventajas) y Python ha

[4] Archivos de texto en los que se sitúa el código.

llegado a ser uno de los idiomas de programación más usados. En la siguiente imagen se puede observar la evolución que ha experimentado en los últimos años.

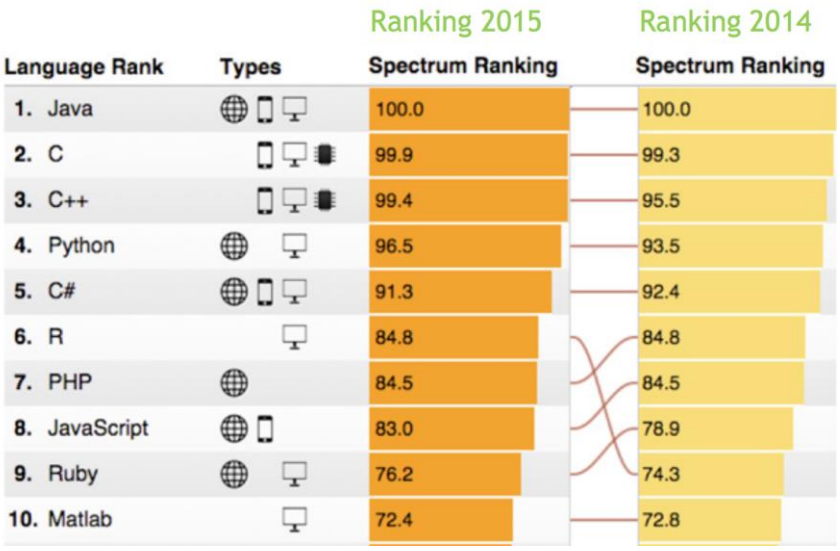


Figura 4-1. Popularidad de los lenguajes de programación (IEEE Spectrum 2015).

4.3. Características y ventajas para su uso

Se ha utilizado Python debido a que posee las siguientes características que lo hacen muy interesante para la realización de tareas de control remoto:

- Lenguaje de programación fácil de aprender con ciclos de desarrollo cortos. Sintaxis simple, clara y sencilla que permite realizar programas compactos y legibles.
- Lenguaje interpretado, ahorra mucho tiempo durante el desarrollo ya que no es necesario compilar ni enlazar. El intérprete puede usarse interactivamente, lo que facilita experimentar con características del lenguaje, escribir programas descartables o probar funciones cuando se hace desarrollo de programas de abajo hacia arriba.
- Representa un alto nivel de abstracción, que encaja perfectamente en el nivel de abstracción de los programas de medición.
- Tiene tipado dinámico, por lo que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne. A su vez, el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo posteriormente.
- Fuertemente tipado: No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente.
- Multiplataforma: El intérprete de Python está disponible en multitud de plataformas (UNIX, Linux, Windows, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.
- Posee un gestor de memoria.
- Python es extensible: quiere decir que se puede agregar una nueva función o módulo al intérprete, ya sea para realizar operaciones críticas a velocidad máxima, o para enlazar programas Python con bibliotecas que tal vez sólo estén disponibles en forma binaria.
- Tiene un muy amplio conjunto de bibliotecas nativas, incluyendo módulos numéricos y de trazado para el análisis de datos y visualización.

- Hay disponible un gran conjunto de libros y publicaciones.

4.4. Formas de trabajo en Python

En este apartado se indican las principales formas de trabajo por parte de los usuarios de este lenguaje de programación:

- Con el intérprete directamente.
- Con un procesador de texto, que se utiliza para ir creando *scripts* (.py). Las instrucciones se interpretan una a una por el terminal, en caso de producirse algún error salta.
- Lanzando el *script* como un programa. Para que esta técnica funcione hay que dar permisos de ejecución al *script* que queremos lanzar.

4.5. Tipos básicos

Las variables, como se ha indicado en una de las características anteriores, no tienen tipo, éste se le asigna cuando se declara y puede cambiar sobre la marcha. Las variables son objetos realmente. En las siguientes líneas se explican los tipos más comunes.

4.5.1. Números

En Python se pueden representar números enteros, reales y complejos:

- Enteros: Se representan mediante el tipo *int* o el tipo *long*. La única diferencia es que el tipo *long* permite almacenar números de mayor tamaño. Es aconsejable no utilizar este último a menos que sea necesario, para no malgastar memoria.
  El tipo *long* permite almacenar números de cualquier precisión, estando limitados solo por la memoria disponible en la máquina.
  Al asignar un número a una variable pasará a tener tipo *int* a menos que el número sea tan grande como para requerir el uso del tipo *long*.
- Reales: Son los que tienen decimales, se expresan mediante el tipo *float*. Python implementa este tipo a bajo nivel mediante una variable de tipo *double* de C, es decir, utilizando 64 bits, luego en Python siempre se utiliza doble precisión de la siguiente forma: 1 bit para el signo, 11 para el exponente, y 52 para la mantisa. Desde Python 2.4 también se tiene el tipo *decimal*, para el caso de que se necesite representar fracciones de forma más precisa.
  Para representar un número real, por tanto, se escribe primero la parte entera, seguido de un punto y por último la parte decimal. También se puede hacer mediante notación científica añadiendo el carácter "e" para indicar un exponente en base 10.
- Complejos: Se representan con el tipo llamado *complex*, también se almacenan usando coma flotante, debido a que estos números son una extensión de los números reales.

```
>>> a=complex(3,5)
>>> a.imag
5.0
>>> a.real
3.0
```

Figura 4-2. Ejemplo de numeración compleja.

- Operadores. En este apartado se ve qué se puede hacer con los tipos de números vistos

anteriormente usando los operadores por defecto. Para operaciones más complejas podemos recurrir al módulo *math*.

- Operadores aritméticos:

Tabla 4–1. Operadores aritméticos.

Operador	Descripción
+	Suma
-	Resta
-	Negación
*	Multiplicación
**	Exponente
/	División
//	División entera
%	Módulo

- Operadores a nivel de bit:

Tabla 4–2. Operadores a nivel de bit.

Operador	Descripción
&	And
|	Or
^	Xor
~	Not
<<	Desplazamiento izq.
>>	Desplazamiento der.

4.5.2. Cadenas

Las cadenas están formadas por texto encerrado entre comillas simples o dobles. Dentro de las comillas se pueden añadir caracteres especiales escapándolos con "\"; como, por ejemplo: "\n": nueva línea, "\t": tabulación, etc.

Una cadena puede estar precedida por el carácter 'u' o 'r', los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación *Unicode* o una cadena *raw*. Las cadenas *raw* se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas.

También es posible encerrar una cadena entre comillas triples, con ellas se puede escribir el texto en varias

líneas y al imprimir la cadena se respetan los saltos de línea que se introdujeron sin tener que recurrir al carácter "\n", así como las comillas sin tener que escaparlas.

```
>>> cadena = "Esta es una cadena larga que posee varias\n\
... líneas de texto.\n\
...      Hay que tener en cuenta que los espacios en blanco\n\
...    al principio de una línea son significantes."
>>> print (cadena)
Esta es una cadena larga que posee varias
líneas de texto.
     Hay que tener en cuenta que los espacios en blanco
   al principio de una línea son significantes.
```

Figura 4-3. Ejemplo de cadena.

Características de las cadenas:

Se pueden utilizar algunos de los operadores aritméticos para realizar diferentes tareas con las cadenas, sobre todo es especialmente útil para la concatenación de las mismas. En el siguiente ejemplo se aprecia este último caso:

```
>>> palabra = 'Ayuda' + 'A'
>>> palabra
'AyudaA'
>>> '<' + palabra*4 + '>'
'<AyudaAAyudaAAyudaAAyudaA>'
```

Figura 4-4. Concatenación de cadenas.

4.5.3. Booleanos

Una variable de tipo booleano sólo puede tener dos valores: True (cierto) y False (falso). Estos valores son muy significativos para las expresiones condicionales y los bucles.

En las siguientes tablas se ven los distintos tipos de operadores con los que se pueden trabajar con los valores booleanos, conocidos como operadores lógicos o condicionales:

Tabla 4–3. Operadores lógicos o condicionales.

Operador	Descripción
And	¿Se cumplen ambos?
Or	¿Se cumple alguno?
Not	Negación al término

Los valores booleanos son el resultado de expresiones de comparación entre valores, utilizan los operadores relacionales de la Tabla 4-4:

Tabla 4–4. Operadores relacionales.

Operador	Descripción
==	¿Son iguales?
!=	¿Son distintos?
<	¿Es menor que?
>	¿Es mayor que?
<=	¿Es menor o igual que?
>=	¿Es mayor o igual que?

4.6. Colecciones

En este apartado se ven algunos tipos de colecciones de datos como son: listas, tuplas y diccionarios.

4.6.1. Listas

Son un tipo de colección ordenada. Es equivalente a lo que, en otros lenguajes como C, se conoce por *arrays*, o vectores. Pueden contener cualquier tipo de dato: números, cadenas, booleanos e incluso otras listas.

Para crear una lista hay que indicar entre corchetes y separados por comas, los valores que queremos incluir en ella. Se puede acceder a cada uno de los elementos de la lista escribiendo el nombre de la lista e indicando el índice del elemento entre corchetes, sabiendo que el primer elemento siempre viende dado por el número 0. Se puede utilizar este operador para modificar un elemento de la lista si se coloca en la parte izquierda de una asignación.

El operador "[]" de Python con el que se definen las listas tiene varias curiosidades:

- Se pueden utilizar también como índice números negativos. Esto se traduce en que el índice empieza a contar desde el final hacia la izquierda, es decir, con [-1] accederíamos al último elemento de la lista, con [-2] al penúltimo, y así sucesivamente.
- Otra particularidad de este operador es lo que en Python se conoce como *slicing* o particionado, que consiste en ampliar este mecanismo para permitir seleccionar porciones de la lista. Si en lugar de un número escribimos dos, inicio y fin, separados por dos puntos (inicio: fin) Python interpretará que queremos una lista que vaya desde la posición inicio a la posición fin, sin incluir este último.

```
>>> lista = ['primero', 'segundo', 93, 1256]
>>> lista
['primero', 'segundo', 93, 1256]
```

Figura 4-5. Ejemplo de lista.

Operaciones a realizar con listas:

- *lista[i]*: Devuelve el elmento que está en la posición "i" de la lista.
- *lista.pop(i)*: Devuelve el elemento en la posición "i" de una lista y posteriormente lo elimina.

- *lista.append(elemento)*: Añade elemento al final de la lista.
- *lista.insert(i, elemento)*: Inserta elemento en la posición "i".
- *lista.extend(lista2)*: Fusiona lista con lista2.
- *lista.remove(elemento)*: Elimina la primera vez que aparece elemento.

4.6.2. Tuplas

Para las tuplas se puede aplicar todo lo dicho en el apartado anterior sobre las listas a excepción de la forma de definirlas, para lo que se utilizan paréntesis en lugar de corchetes. Además, hay que tener en cuenta que es necesario añadir una coma para las tuplas de un solo elemento, para diferenciarlo de un elemento cualquiera entre paréntesis.

Para referirnos al elemento de una tupla, como en una lista, se usa el operador "[]". Se puede utilizar dicho operador debido a que las tuplas, al igual que las listas, forman parte de un tipo de objetos llamados secuencias.

Las tuplas son inmutables, es decir, sus valores no se pueden modificar una vez creadas, además tienen un tamaño fijo. A cambio de estas limitaciones son más ligeras que las listas desde el punto de vista del procesador de la máquina, por lo que, si el uso que le vamos a dar a una colección es muy básico se pueden utilizar tuplas en lugar de listas y ahorrar memoria.

```
>>> tupla = 233, 9497, 'tercero'
>>> tupla
(233, 9497, 'tercero')
>>> tupla[2]
'tercero'
```

Figura 4-6. Ejemplo de tupla.

4.6.3. Diccionarios

Los diccionarios, también conocidos como matrices asociativas, deben su nombre a que son colecciones que relacionan una clave con un determinado valor. Como clave se puede utilizar cualquier valor inmutable: números, cadenas, booleanos, tuplas, ... pero no listas o diccionarios, dado que son mutables.

A los valores almacenados en un diccionario no se accede por su índice (como se hace en listas y tuplas), sino por su clave, utilizando el operador "[]". Esto es así porque los diccionarios no tienen orden.

Tampoco se puede utilizar *slicing*, porque los diccionarios no son secuencias, si no *mappings* (mapeados, asociaciones).

```
>>> diccionario = {'primero':"listas", 'segundo':'tuplas', 'tercero':'diccionarios'}
>>> print (diccionario)
{'primero': 'listas', 'segundo': 'tuplas', 'tercero': 'diccionarios'}
```

Figura 4-7. Ejemplo de diccionario.

Operaciones a realizar con diccionarios: Similares a las listas, con el matiz de que los diccionarios no tienen orden, por tanto, no tienen funciones en las que se tenga en cuenta la posición.

- *Diccionario.get('clave')*: Devuelve el valor que corresponde con la clave introducida.
- *Diccionario.pop('clave')*: Devuelve el valor que corresponde con la clave introducida, y después borra la clave y el valor.

- *"clave"* en *diccionario*: Devuelve verdadero (True) o falso (False) si la clave (no los valores) existe en el diccionario.
- *"definicion"* en *diccionario.values()*: Devuelve verdadero (True) o falso (False) si *definicion* existe en el diccionario (no como clave).

4.7. Control de flujo

En esta sección se van a explicar las estructuras condicionales y los bucles.

4.7.1. Sentencias condicionales

Los condicionales permiten comprobar condiciones y hacer que un programa se comporte de una forma o de otra, que ejecute un fragmento de código u otro, dependiendo de la condición anterior. En este apartado van a cobrar importancia el tipo booleano y los operadores lógicos y relacionales que se explicaron anteriormente.

if

La forma más simple de una sentencia condicional es el *if* seguido de la condición a evaluar, dos puntos y en la siguiente línea e indentado, el código a ejecutar en caso de que se cumpla dicha condición.

Es importante en Python indentar el código, es decir, pulsar tabulación antes de las órdenes de ejecución, dado que es la forma de hacerle saber al intérprete que se quieren ejecutar dichas órdenes solo en el caso de que se cumpla la condición.

if…else

Se usa para ejecutar ciertas órdenes en el caso de que la primera condición no se cumpliera. Sustituye el funcionamiento anterior del *if* en dos ocasiones. En vez de colocar dos estructuras iguales la segunda condición se sustituye por un *else* (con una traducción del estilo: si no, en caso contrario).

if…elif…elif…else

Se sigue el mismo procedimiento anterior en reiteradas ocasiones, *elif* es una contracción de *else if*. Una síntesis de su patrón de ejecución sería el siguiente:

Se evalúa la condición del *if*, si es cierta, se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple, se evalúa la condición del *elif*. Si se cumple la condición del *elif* se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple y hay más de un *elif* se continúa con el siguiente en orden de aparición. Si no se cumple la condición del *if* ni de ninguno de los *elif*, se ejecuta el código del *else*.

```
>>> x = 15
>>> if x < 0:
...     print ('numero negativo')
... elif x == 0:
...     print ('cero')
... else:
...     print ('mayor que cero')
...
mayor que cero
```

Figura 4-8. Ejemplo de sentencia condicional.

4.7.2. Bucles

Permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

while

Ejecuta un fragmento de código mientras se cumpla una condición. Posee dos palabras claves que tienen la siguiente funcionalidad:

- *Break*: Sale del bucle en el que estamos, muy útil para casos de bucles infinitos (los cuales se repiten indefinidamente hasta que se ejecute esta orden).
- *Continue*: Pasa directamente a la siguiente iteración del bucle.

```
>>> a, b = 0, 1
>>> while b < 10:
...     print (b)
...     a, b = b, a+b
...
1
1
2
3
5
8
```

Figura 4-9. Ejemplo de bucle *While*.

for…in

En Python se utiliza como una forma genérica de iterar sobre una secuencia, como, por ejemplo, una lista. Como si de lenguaje natural se tratara la cabecera de un bucle *for* sería: "*para cada elemento de la secuencia indicada*". Y esto es exactamente lo que hace el bucle: para cada elemento que tengamos en la secuencia, ejecuta las líneas de código introducidas posteriormente.

Python proporciona una función llamada *range* que permite generar una lista que vaya desde el primer número que le indiquemos al segundo. En el siguiente apartado de funciones se explican todas ellas con más detenimiento.

```
>>> for vuelta in range (1, 10):
...     print ("Vuelta" + str(vuelta))
...
Vuelta1
Vuelta2
Vuelta3
Vuelta4
Vuelta5
Vuelta6
Vuelta7
Vuelta8
Vuelta9
```

Figura 4-10. Ejemplo de bucle *for*.

4.8. Funciones

Son fragmentos de código con un nombre asociado que realizan una serie de tareas y devuelven un valor, conocidos por el nombre de procedimientos. En Python no existen los procedimientos, ya que cuando el programador no especifica un valor de retorno la función devuelve el valor *None*, equivalente al *Null* de otros lenguajes de programación.

Para la declaración de las funciones se sigue el siguiente procedimiento: La palabra clave *def* seguida del nombre de la función y entre paréntesis los argumentos separados por comas. A continuación, indentadas, se tendrían las líneas que conforman el código a ejecutar. Al declarar una función se está asociaciando un nombre al fragmento de código que la conforma, de esta manera es posible ejecutar dicho código más tarde referenciándolo por su nombre.

Se puede utilizar una cadena de texto como primera línea del cuerpo de la función, estas cadenas se conocen con el nombre de *docstring* (cadena de documentación) y sirven, como su nombre indica, a modo de documentación de la función. Todos los objetos pueden tener *docstrings*, no solo las funciones.

Para ejecutar una función se escribe el nombre de la función que se va a ejecutar seguido de los valores que se pasan como parámetros entre paréntesis. La asociación de los parámetros y los valores pasados a la función se hace normalmente de izquierda a derecha, aunque es posible modificar el orden de los parámetros indicando el nombre del parámetro al que asociar el valor a la hora de llamar a la función. El número de valores que se pasan como parámetros al llamar a una función tiene que coincidir con el número de parámetros que la función acepta según su declaración.

También es posible definir funciones con un número variable de argumentos y asignar valores por defecto a los parámetros para el caso de que no se indique ningún valor para los mismos al llamar a la función. Los valores por defecto se definen escribiendo un signo igual después del nombre del parámetro precedido del dato.

Para definir funciones con un número variable de argumentos se coloca un último parámetro en la función cuyo nombre se precede del carácter '*', esta sintaxis funciona creando una tupla en la que se almacenan los valores de todos los parámetros extra pasados como argumento. También se puede preceder el nombre del último parámetro con '**', en cuyo caso en lugar de una tupla se utiliza un diccionario.

En Python, en el paso por referencia lo que se pasa como argumento es un puntero a la variable, es decir, la dirección de memoria en la que se encuentra el contenido de la variable, y no el contenido en sí. En el paso por valor se pasa como argumento el valor que contiene la variable. La diferencia entre ambos está en que en el paso por valor los cambios que se hagan sobre el parámetro no se ven fuera de la función ya que en realidad lo que se le pasa a la función son copias de los valores y no las variables en sí.

No todos los cambios que se le hagan a los parámetros de una función en Python se reflejan fuera de ésta, ya que hay que tener en cuenta que existen objetos inmutables, como las tuplas, por lo que si intentamos modificar una tupla pasada como parámetro lo que ocurre en realidad es que se crea una nueva instancia, por lo que los cambios no se ven fuera de la función. En resumen, los valores mutables se comportan como paso por referencia, los inmutables como paso por valor.

Para finalizar con las funciones se explica cómo se devuelven valores en ellas, para lo que se utiliza la palabra clave *return*. Las funciones en Python no pueden devolver varios valores, lo que ocurre en realidad es que Python crea una tupla cuyos elementos son los valores a retornar y esta única variable es la que se devuelve.

```
>>> def fib(n):
...     """Función que escribe la serie de Fibonacci hasta n."""
...     a, b = 0, 1
...     while b < n:
...         print (b)
...         a, b = b, a+b
...
>>> fib(30)
1
1
2
3
5
8
13
21
```

Figura 4-11. Ejemplo de función.

4.9. Orientación a objetos

Python es un lenguaje multiparadigma en el se puede trabajar con programación estructurada (como se ha trabajado en los ejemplos de los apartados anteriores) o programación orientada a objetos, también llamada programación funcional.

Con la Programación Orientada a Objetos el problema a resolver se modela a través de clases y objetos, donde un programa consiste en una serie de interacciones entre estos objetos.

Hay tres conceptos que son básicos para cualquier lenguaje de programación orientado a objetos: herencia, polimorfismo y encapsulamiento, se explican en los siguientes apartados.

4.9.1. Clases y objetos

Una clase es una plantilla genérica a partir de la cuál se instancian los objetos, plantilla que define qué atributos y métodos tendrán los objetos de esa clase.

Un objeto es una entidad que agrupa un estado y una funcionalidad relacionadas. El estado del objeto se define a través de variables llamadas atributos, mientras que la funcionalidad se modela a través de funciones a las que se les conoce con el nombre de métodos del objeto.

Las clases se definen mediante la palabra reservada *class* seguida del nombre de la clase, dos puntos y el cuerpo de la clase indentado. Como en el caso de las funciones, si la primera línea del cuerpo se trata de una cadena de texto, esta será la cadena de documentación de la clase o *docstring*.

El método "*__init__*", con doble guion bajo al principio y al final, se ejecuta justo después de crear un nuevo objeto a partir de la clase, proceso que se conoce con el nombre de instanciación. El primer parámetro de "*__init__*" y del resto de métodos de la clase es siempre *self*. Este mecanismo es necesario para poder acceder a los atributos y métodos del objeto diferenciando, por ejemplo, una variable local de un atributo del objeto.

Para crear un objeto se escribe el nombre de la clase seguido de cualquier parámetro que sea necesario entre paréntesis. Estos parámetros se pasan al método "*__init__*", que como se ha explicado es el método que se llama al instanciar la clase.

Para acceder a los atributos y métodos de un objeto ya creado se utilizan las siguientes sintaxis: *objeto.atributo* y *objeto.metodo()*.

4.9.2. Herencia

En un lenguaje orientado a objetos cuando se hace que una clase (subclase) herede de otra clase (superclase) se provoca que la subclase contenga todos los atributos y métodos que tenía la superclase. Para indicar que una

clase hereda de otra se coloca el nombre de la clase de la que se hereda entre paréntesis después del nombre de la nueva clase.

Si se quiere especificar un nuevo parámetro dentro de un objeto que se va a crear en una subclase, se hace con el método "*__init__*", a esta acción se le conoce como sobreescribir métodos.

En el caso de que se quiera sobreescribir un método de la clase padre y en su interior ejecutar el mismo método sobreescrito se usa la sintaxis *SuperClase.metodo(self, args)* para llamar al método de igual nombre de la clase padre.

4.9.3. Polimorfismo

Se refiere a la habilidad de objetos de distintas clases de responder al mismo mensaje. Esto se puede conseguir a través de la herencia: un objeto de una clase derivada es al mismo tiempo un objeto de la clase padre, de forma que allí donde se requiere un objeto de la clase padre también se puede utilizar uno de la clase hija.

Python, al ser de tipado dinámico, no impone restricciones a los tipos que se le pueden pasar a una función, por esta razón el polimorfismo en Python no es de gran importancia.

En ocasiones también se utiliza este término para referirse a la sobrecarga de métodos, que se define como la capacidad del lenguaje en determinar qué método ejecutar de entre varios con igual nombre según el tipo o número de parámetros que se le pasa.

4.9.4. Encapsulación

Se refiere a impedir el acceso a determinados métodos y atributos de los objetos estableciendo así qué puede utilizarse desde fuera de la clase. Lo que se suele hacer en Python para esta tarea es, aprovechando que el acceso a una variable o función viene determinado por su nombre: si el nombre comienza con dos guiones bajos y no termina también con dos guiones bajos, se trata de una variable o función privada, en caso contrario es pública. Los métodos cuyo nombre comienza y termina con dos guiones bajos son métodos especiales que Python llama automáticamente bajo determinadas circunstancias.

Este mecanismo se basa en que los nombres que comienzan con un doble guion bajo se renombran para incluir el nombre de la clase, característica que se conoce como *name mangling*.

Puede ocurrir que se quiera permitir el acceso a algún atributo de nuestro objeto de forma controlada. Para ello se escriben métodos cuyo único cometido sea éste, los cuales normalmente tienen nombres como *getVariable* y *setVariable*; de ahí que se conozcan también con el nombre de *getters* y *setters*.

4.10. Programación funcional

La programación funcional es un paradigma en el que la creación de un programa se basa casi en su totalidad en funciones, entendiendo función según su definición matemática, y no como subprogramas.

Python, sin ser un lenguaje puramente funcional incluye varias características tomadas de los lenguajes funcionales como son las funciones de orden superior o las funciones lambda que se estudian en los siguientes apartados.

4.10.1. Funciones de orden superior

El concepto de funciones de orden superior está relacionado con el uso de funciones como si de un valor cualquiera se tratara, posibilitando el pasar funciones como parámetros de otras funciones o devolver funciones como valor de retorno. Esto es posible porque en Python todo son objetos (como se ha explicado anteriormente), incluidas las funciones.

Una de las características de dichas funciones es que no es necesario almacenar la función que se pasa como valor de retorno en una variable para poder llamarla.

4.10.2. Funciones Lambda

El operador lambda sirve para crear funciones anónimas, es decir, sin nombre, éstas no podrán ser referenciadas más tarde. Se construyen mediante el operador *lambda*, los parámetros de la función separados por comas, sin paréntesis, dos puntos y el código de la función.

Las funciones lambda están restringidas por la sintaxis a una sola expresión, estas contrucciones pueden ser de utilidad para reducir código en un programa extenso.

4.10.3. Generadores

La sintaxis de las expresiones generadoras es igual que la de creación de listas a excepción de que se utilizan paréntesis en lugar de corchetes, es por eso que su funcionamiento es muy similar. Difieren de las listas en que no se devuelve una lista, sino un generador.

Un generador es una clase especial de función que origina valores sobre los que iterar. Para devolver el siguiente valor sobre el que iterar se utiliza la palabra clave *yield*. El generador se puede utilizar en cualquier lugar donde se necesite un objeto iterable.

Como no se crea una lista completa en memoria, sino que se genera un solo valor cada vez que se necesita, en situaciones en las que no sea necesario tener la lista completa, el uso de generadores puede suponer una gran diferencia de memoria. En todo caso, es posible crear una lista a partir de un generador mediante la función *list*.

4.10.4. Decoradores

Es un tipo de función que recibe una función como parámetro y devuelve otra función como resultado. Una vez decorada una función, se llama precediendo la definición de la nueva que se quiere decorar con el carácter '@' seguido del nombre de la función decoradora.

Si se quiere aplicar más de un decorador basta con añadir una línea con el nuevo decorador, se ejecutan de abajo a arriba.

4.11. Módulos y paquetes

4.11.1. Módulos

Entidades que permiten una organización y división lógica de nuestro código, sirven para facilitar el mantenimiento y la lectura de programas demasiado largos. Los ficheros son su contrapartida física ya que cada archivo Python almacenado en disco equivale a un módulo. Cuando se crea un documento "*.py*" ha de guardarse en el mismo directorio en el que se encuentra el archivo del módulo debido a que es la única forma de que Python pueda encontrarlo para realizar su ejecución. En el caso de que no encontrara ningún módulo con el nombre especificado, se lanzaría una excepción de tipo *ImportError*.

Para utilizar la funcionalidad definida en un módulo hay que importarlo, para ello se utiliza la palabra clave *import* seguida del nombre del módulo, que consiste en el nombre del archivo menos la extensión. El importar no solo hace que se tenga disponible todo lo definido dentro del módulo, sino que también ejecuta el código del módulo. La cláusula *import* permite importar varios módulos en la misma línea.

Es necesario preceder el nombre de los objetos de un módulo que se importan con el nombre del módulo al que pertenecen, o lo que es lo mismo, el espacio de nombres en el que se encuentran. Esto permite que no se sobreescriba accidentalmente algún otro objeto que tuviera el mismo nombre al importar otro módulo. Aunque es posible no tener que indicar el nombre del módulo antes del objeto que interesa mediante la construcción *from-import*, de esta forma se importa el objeto o los objetos que indiquemos al espacio de nombres actual. También se pueden importar todos los nombres del módulo al espacio de nombres actual usando el caracter '*'.

Importación de los módulos *os*, *sys* o *time*:

Para estos módulos no se encuentran los archivos *os.py*, *sys.py* y *time.py* en el mismo directorio por tanto Python da un error como se comentó en el apartado anterior.

Cuando se importa un módulo, Python recorre todos los directorios indicados en la variable de entorno *PYTHONPATH* en busca de un archivo con el nombre adecuado. El valor de dicha variable se puede consultar desde Python mediante *sys.path*. De esta forma, para que estos módulos estuvieran disponibles para todos los programas del sistema bastaría con que se copien a uno de los directorios indicados en *PYTHONPATH*.

Los módulos también son objetos, de tipo *module*, lo que significa que pueden tener atributos y métodos. Algunos de sus atributos más usados son:

- "*__name__*": Se utiliza para incluir código ejecutable en un módulo, pero que éste sólo se ejecute si se llama al módulo como programa, y no al importarlo.
- "*__doc__*": Sirve a modo de documentación del objeto, *docstring*. Su valor es el de la primera línea del cuerpo del módulo en el caso de que ésta sea una cadena de texto, en caso contrario vale *None*.

4.11.2. Paquetes

Sirven para organizar los módulos, son tipos especiales de módulos (de tipo *module* también) que permiten agrupar módulos relacionados. Mientras los módulos se corresponden a nivel físico con los archivos, los paquetes se representan mediante directorios.

Para hacer que Python trate a un directorio como un paquete es necesario crear un archivo *__init__.py* en dicha carpeta. Para meter un módulo dentro de un paquete, hay que copiar el archivo que define el módulo al directorio del paquete.

Como en los modulos, para importar paquetes se utiliza *import* y *from-import*.

Para separar paquetes, subpaquetes y módulos se utiliza el caracter '.'.

4.12. Manejo de archivos

En este apartado y para finalizar con Python se va explicar una de las características más importantes que debe tener un lenguaje de programación, el hecho de interaccionar con el usuario. Además de describir detalladamente el uso de *print* para mostrar mensajes al usuario, se ven las funciones *input* y *raw_input* para pedir información, así como los argumentos de línea de comandos y, por último, la entrada/salida de ficheros.

4.12.1. Entrada estándar

La manera con menos complejidad de obtener información por parte del usuario es mediante la función *raw_input*. Esta función toma como parámetro una cadena a usar como *prompt* (texto a mostrar al usuario pidiendo la entrada) y devuelve una cadena con los caracteres introducidos por el usuario hasta el pulsado de la tecla *Enter*.

Si se requiere un entero como entrada en lugar de una cadena se usa la función *int* para convertir la cadena a entero. Hay que tener en cuenta que puede lanzarse una excepción si lo que introduce el usuario no es un número.

La función *input* utiliza *raw_input* para leer una cadena de la entrada estándar y después la evalúa como si fuera código Python, por tanto, *input* es la manera más compleja y sensible de realizar esta tarea.

4.12.2. Parámetros de línea de comando

Adicionalmente al uso de las funciones *input* y *raw_input*, existen otros métodos para obtener datos del usuario. Uno de ellos es el uso de parámetros a la hora de llamar al programa en la línea de comandos.

Existen módulos, como *optparse*, que facilitan el trabajo con los argumentos de la línea de comandos. Éste se encarga de buscar los *flags* y organizarlos de tal forma que se puede comprobar si el argumento tiene que ser cadena o entero para luego crear un diccionario con los argumentos y sus variables.

4.12.3. Salida estándar

La forma más sencilla de mostrar información en la salida estándar es mediante *print*. En su forma más básica a *print* le sigue una cadena, que se mostrará en la salida estándar al ejecutarse el comando. Después de imprimir la cadena pasada como parámetro el puntero se sitúa en la siguiente línea de la pantalla.

Posteriormente al estamento de *print* se le puede colocar una coma, lo que indica que se introduce automáticamente un espacio para separar cada una de las cadenas establecidas en dicha orden. Las cadenas que imprime la sentencia *print* permiten utilizar técnicas avanzadas de formateo.

Para dar formato a un valor se usan los especificadores. Los más sencillos están formados por el carácter "%" seguido de una letra que indica el tipo con el que formatear el valor. Los especificadores más comunes son los siguientes:

Tabla 4–5. Especificadores.

Especificador	**Formato**
%s	Cadena
%d	Entero
%o	Octal
%x	Hexadecimal
%f	Real

Se puede introducir un número entre "%" y el carácter que indica el tipo al que formatear, que indica el número mínimo de caracteres que queremos que ocupe la cadena. Si el tamaño de la cadena resultante es menor que este número, se añadirán espacios a la izquierda de la cadena. En el caso de que el número sea negativo, ocurrirá exactamente lo mismo, sólo que los espacios se añadirán a la derecha de la cadena.

En el caso de los reales es posible indicar la precisión a utilizar precediendo el carácter "f" de un punto seguido del número de decimales que se quieren mostrar. La misma sintaxis se puede utilizar para seleccionar un número de caracteres en una cadena.

4.12.4. Archivos

Los ficheros en Python son objetos de tipo *file* creados mediante la función *open*. Esta función toma como parámetros una cadena con la ruta del fichero a abrir, que puede ser relativa o absoluta, una cadena opcional indicando el modo de acceso (por defecto tiene el modo lectura) y, por último, un entero opcional para indicar un tamaño de *buffer* distinto del utilizado por defecto.

El modo de acceso puede ser cualquier combinación lógica de los siguientes:

- 'r': lectura (del inglés *read*). Abre el archivo en modo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción.
- 'w': escritura (del inglés *write*). Abre el archivo en modo escritura. Si el archivo no existe, se crea. Si existe, sobreescribe el contenido.
- 'a': añadir (del inglés *append*). Abre el archivo en modo escritura. Se diferencia del modo 'w' en

que en este caso no se sobreescribe el contenido del archivo en caso de que ya exista, sino que se comienza a escribir al final del archivo.

- 'b': binario (del inglés *binary*).
- '+': permite lectura y escritura simultáneas.
- 'U': saltos de línea universales (del inglés *universal newline*). Permite trabajar con archivos que tengan un formato para los saltos de línea que no coincide con el de la plataforma actual.

Tras crear el objeto se pueden realizar las operaciones de lectura/escritura pertinentes utilizando los métodos de objeto que se verán en los siguientes apartados. Una vez se acabe de trabajar con el archivo debemos cerrarlo mediante el método *close*.

Lectura de archivos:

Para la lectura de archivos se utilizan los métodos *read*, *readline* y *readlines*.

- *read*: Devuelve una cadena con el contenido del archivo o bien el contenido de los primeros n bytes, si se especifica el tamaño máximo a leer.
- *readline*: Sirve para leer las líneas de un fichero una a una. Es decir, cada vez que se llama a este método, se devuelve el contenido del archivo desde el puntero hasta que se encuentra un carácter de nueva línea, incluyendo este carácter.
- *readlines*: Funciona leyendo todas las líneas del archivo y devolviendo una lista con las líneas leídas.

Escritura de archivos:

Se usan los métodos *write* y *writelines*.

- *write*: Escribe en el archivo una cadena de texto que toma como parámetro.
- *writelines*: Toma como parámetro una lista de cadenas de texto indicando las líneas que se desean escribir en el fichero.

Mover el puntero de lectura/escritura:

En determinados casos se necesita mover el puntero de lectura/escritura a una posición determinada del archivo, uno de estos casos es, por ejemplo, si se pretende empezar a escribir en una posición determinada y no al final o al principio del archivo. Para ello se utilizan los métodos *seek* y *tell*.

- *seek*: Toma como parámetro un número positivo o negativo a utilizar como desplazamiento. También es posible utilizar un segundo parámetro para indicar desde dónde se quiere realizar el desplazamiento: 0 indicará que el desplazamiento se refiere al principio del fichero, 1 a la posición actual, y 2, al final del fichero.
- *tell*: Para determinar la posición en la que se encuentra actualmente el puntero. Devuelve un entero indicando la distancia en bytes desde el principio del fichero.

4.13. Control remoto con Python

4.13.1. Python vs. Matlab para el control remoto

Python en los últimos años se está convirtiendo en una alternativa atractiva a Matlab debido a que es gratuito, de código abierto y cada vez es capaz de realizar tareas más potentes. Por lo tanto, las tareas de control remoto cada vez están menos monopolizadas por Matlab. De hecho, este trabajo es una muestra del nivel que ha ido

adquiriendo Python con respecto a esta área del conocimiento.

En lo que sigue, se explican las ventajas más notables de los dos lenguajes con respecto a su homólogo, con ello, se desea demostrar la funcionalidad y alcance de Python.

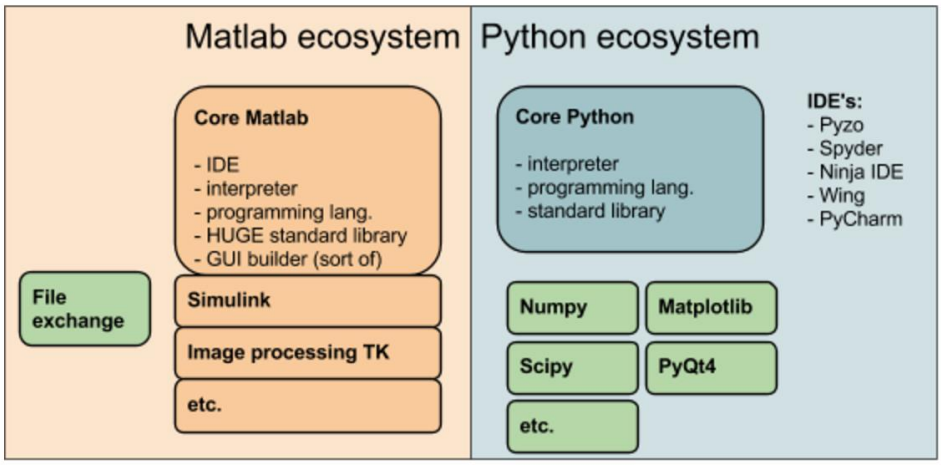


Figura 4-12. Diagrama que muestra las diferencias de ecosistema entre Matlab y Python.

Ventajas de Matlab:

- Sólida cantidad de funciones.
- Simulink no posee una buena alternativa aún.
- El paquete incluye todo lo que se necesita, mientras que en Python es necesario instalar paquetes adicionales y un IDE, entorno de desarrollo integrado (del inglés *Integrated Development Environment*), dependiendo del trabajo que se quiera desempeñar.
- Dispone de una gran comunidad científica, se utiliza en muchas universidades (aunque para pocas empresas es rentable comprar su licencia).

Ventajas de Python:

- Gratuito.
- Fue creado para ser un lenguaje genérico fácil de leer y entender, mientras que Matlab comenzó como un paquete de manipulación de matrices a la que se agregó un lenguaje de programación.
- Potente: Debido a su buen diseño, para el programador es más fácil que con otros lenguajes plasmar en código sus necesidades. Además, posee extensas librerías y tipos de datos que ayudan a organizar mejor los programas.
- Los espacios de nombres: Matlab es compatible con los espacios de nombres de sus funciones, pero el núcleo de Matlab no tiene ya que cada función se define en el espacio de nombres global. Python trabaja con módulos, que se deben importar si se desea utilizarlos. En este lenguaje todo es un objeto, por lo que cada objeto tiene un espacio de nombres en particular.
- Introspección: Las variables privadas sólo existen por convención, para que se pueda acceder a cualquier parte de la aplicación, incluyendo algunos de los componentes internos de Python.
- Manipulación de cadenas.
- Portabilidad: Debido a que Python es gratis, su código se puede ejecutar en todas partes y funciona en todos los sistemas operativos.

- Definiciones de clases y función: Funciones y clases se pueden definir en cualquier lugar. Además, en un archivo se pueden diseñar tantas funciones y clases como se desee.
- Grandes herramientas GUI, interfaz gráfica de usuario (*Graphical User Interface*): Se puede elegir cualquiera de las principales herramientas GUI.

4.13.2. PyVISA

Paquete de Python más usado en el control remoto de instrumentación, se apoya en el estándar VISA explicado en el apartado 2.5 de este trabajo. Su finalidad es el control de dispositivos de medición y equipos de prueba a través de GPIB, RS232, Ethernet o USB.

VISA y Python:

Python tiene varias características que lo hacen muy interesante para el control de medidas:

- Es un lenguaje de programación fácil de aprender con ciclos de desarrollo cortos.
- Representa un alto nivel de abstracción que encaja en el nivel de abstracción de los programas de medición.
- Tiene un amplio conjunto de bibliotecas nativas, incluyendo módulos numéricos y de trazado para el análisis de datos y visualización.
- Tiene disponible un gran conjunto de libros y publicaciones.

Uso:

En primer lugar, se importa el paquete VISA para posteriormente crear un objeto *ResourceManager*. Si se llama sin argumentos, PyVISA utilizará el servidor por defecto (NI), que trata de encontrar la biblioteca compartida VISA. En algunos casos, PyVISA no es capaz de encontrar la biblioteca que se requiere, lo que tiene como resultado un *OSError*. Para su solución hay que encontrar la ruta de la biblioteca manualmente para pasársela a *ResourceManager*, aunque también se puede especificar en un archivo de configuración.

Una vez que se obtiene el *ResourceManager* se listan los recursos utilizando el método *list_resources*, cuya salida es una tupla que enumera los nombres de los recursos de VISA.

Para finalizar se consulta el dispositivo con el mensaje "*IDN?", el cual significa: "¿qué eres?". A partir de este momento comienza la comunicación entre los dos dispositivos en cuestión.

4.14. Entorno de trabajo: Anaconda

Anaconda es la plataforma de trabajo utilizada para la realización de todos los programas en lenguaje Python que forman parte de este trabajo. En la Figura 4-13 se muestra gráficamente cómo es dicho entorno tras su instalación.

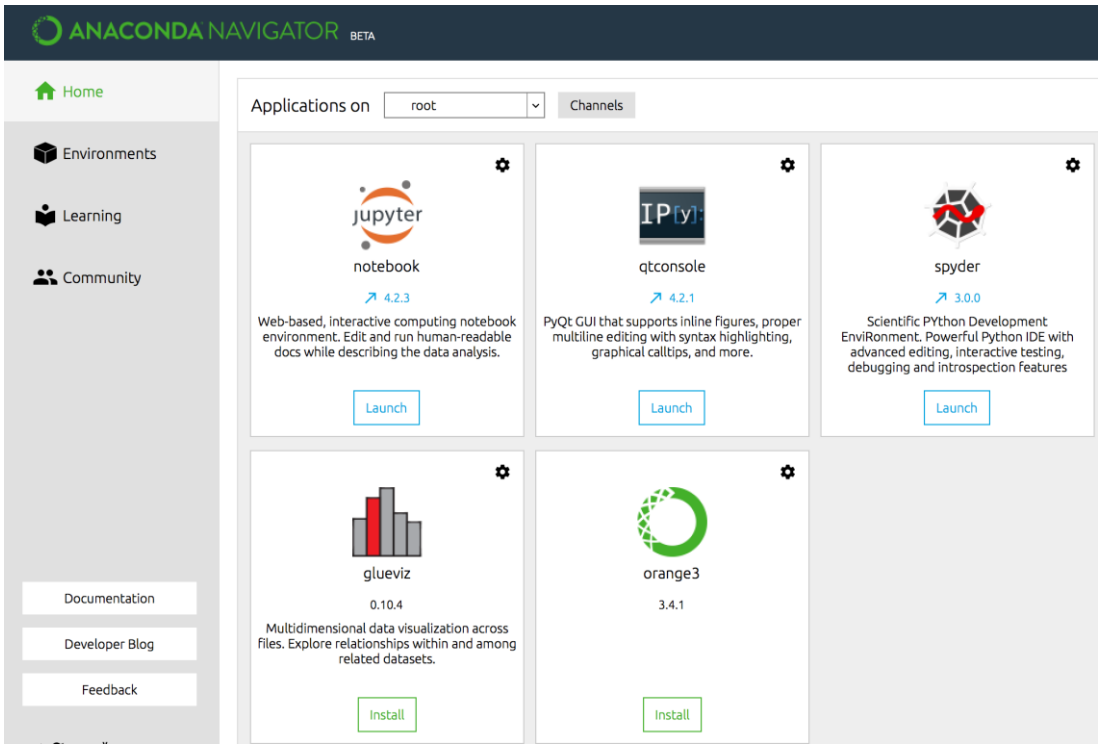


Figura 4-13. Navegador de Anaconda.

En particular, se ha trabajado con la herramienta llamada "spyder" que se aprecia en la parte superior derecha de la Figura 4-13. Spyder es el entorno perteneciente a Anaconda en el que se pueden realizar tanto los archivos de texto (.py) como su ejecución. En la Figura 4-14 se puede ver que la parte de la izquierda de esta herramienta es la que se dedica a la realización de los *scripts* y en la de la derecha se sitúa la consola.

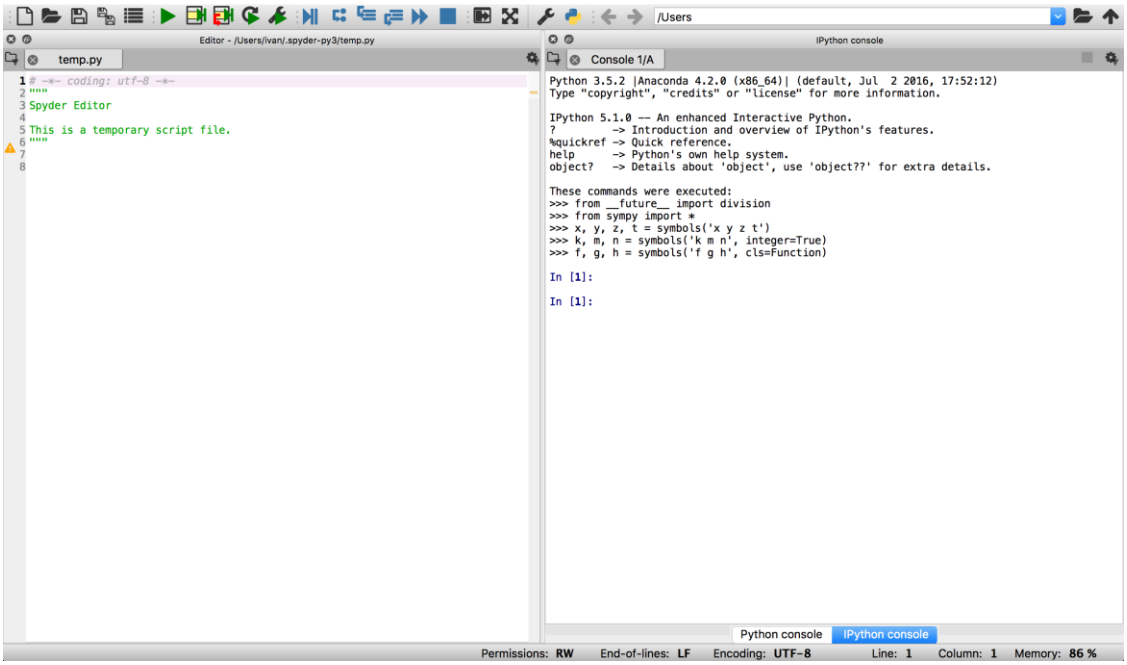


Figura 4-14. Herramienta Spyder.

5 LABORATORIO DE RADIOCOMUNICACIÓN

En este capítulo se presenta al lector la instrumentación usada en el laboratorio de radiocomunicación como ayuda para la realización de este trabajo. Posteriormente se definen los distintos tipos de errores no lineales que se pueden encontrar analizando dispositivos en el laboratorio.

5.1. Instrumentación

5.1.1. Analizador vectorial de señales

Los VSA, analizador vectorial de señales (del inglés *Vector Signal Analyzer*), combinan tecnología superheterodina (usada en los analizadores de barrido de espectro analógicos) con convertidores analógico-digitales (ADC) de alta velocidad y procesamiento digital ofreciendo medidas de alta resolución espectral, demodulación de señales y técnicas avanzadas de análisis en el dominio temporal. Este equipo puede ser controlado de forma remota desde el ordenador usando secuencias de instrucciones en Python. Éste se encuentra conectado al ordenador a través de una conexión mediante el bus GPIB.

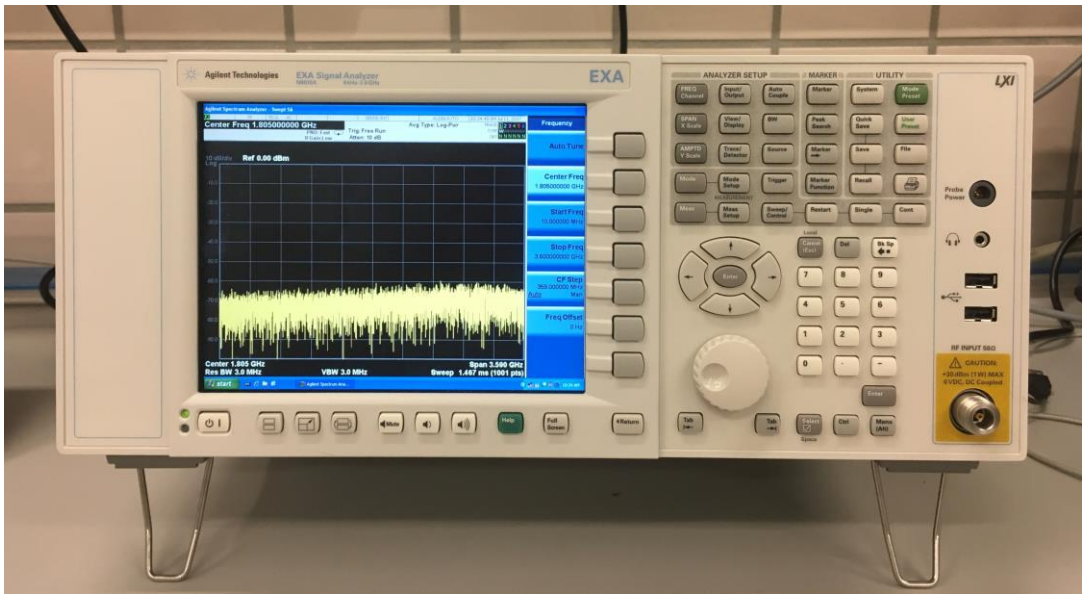


Figura 5-1. Analizador de señal.

La principal diferencia respecto a los analizadores analógicos es que su etapa de frecuencia intermedia está formada por una sección digital que implementa algoritmos FFT y de procesado digital. Los analizadores de señal son sistemas que usan señales digitales y algoritmos matemáticos para efectuar el análisis, sus funciones de mezclado, filtrado y demodulación son ejecutadas de forma digital. Una de sus características más notables es que están diseñados para medir y manipular señales complejas.

El proceso de medida de un analizador de señal simula un banco de filtros en paralelo y posteriormente realiza un procesado de todas las frecuencias simultáneamente. Este proceso se realiza en tiempo real, es decir, sin perder ninguna parte de la señal de entrada.

Los VSA poseen una pequeña fuente de almacenamiento que adquiere temporalmente señal para su posterior procesamiento o repetición. Esta aplicación es análoga a la de un osciloscopio de alta frecuencia, sin embargo, los VSA poseen una ventaja ante ellos, que es el mayor rango dinámico que son capaces de adquirir y la mejor relación señal a ruido.

El demodulador vectorial permite medir un amplio rango de estándares actuales como WCDMA, GSM, LTE, etc.

Medida del VSA:

El funcionamiento del VSA consiste en digitalizar la señal de entrada para computar las medidas posteriormente. Éstos implementan análisis FFT con técnicas que permiten eliminar los errores potenciales inherentes a las técnicas digitales de FFT. A continuación, se muestran las etapas que realiza el analizador de señal en el proceso de medidas:

- Acondicionamiento de señal y traslado de frecuencia.
- Conversión analógico-digital.
- Detección en cuadratura.
- Filtrado de la señal y remuestreo.
- Enventanado de la señal.
- Análisis FFT o vectorial.

5.1.2. Generador de señal

Los generadores de señal son capaces de crear señales periódicas o no periódicas en dominio tanto digital como analógico. Es uno de los equipos electrónicos más importantes de un laboratorio de radiocomunicaciones ya que el diseño electrónico, el test de circuitos y la reparación de equipos requieren señales controlables para simular su operación normal.

Al igual que los analizadores de señal, pueden ser controlados de forma remota. Para hacer posible esta tarea es necesario establecer una conexión GPIB entre el ordenador y el generador. Para la comunicación entre ambos se emplean secuencias de instrucciones en Python. A cada uno de los botones del panel frontal, le corresponde una secuencia de instrucciones.

Figura 5-2. Generador de señal.

Hay varios tipos de generadores de señal según el propósito y la aplicación:

- Generador de señales de microondas y radiofrecuencias: Se emplea en el test de componentes en un amplio rango de aplicaciones entre las que se incluyen: comunicaciones móviles, señales de audio y vídeo, comunicaciones por satélite, etc.

- Generador de funciones: Genera funciones de tipo sinusoidal, triangular, 'diente de sierra' y rectangular. Puede incluir algún tipo de modulación en amplitud, fase, o frecuencia FM, aunque está limitado a un reducido conjunto de salidas. Este dispositivo incorpora un oscilador electrónico, circuitos capaces de producir señales repetitivas en el tiempo y un segundo oscilador capaz de mezclar la señal generada.
- Generador de ondas arbitrarias: Permite producir cualquier forma de onda a la salida dentro de unos límites de potencia, frecuencia y precisión. La forma de onda se suele definir como una secuencia de puntos entre los que el generador va saltando de uno a otro, o implementa algún método de interpolación para suavizar las transiciones. Sintetiza las formas de onda usando técnicas de procesamiento digital de señales. Trabaja con atenuadoras variables para producir diferentes niveles de potencia a la salida y varios métodos de modulación.

5.2. Problema de la no linealidad

5.2.1. Introducción

Los sistemas no lineales se caracterizan por tener un comportamiento que no es expresable como la suma de los comportamientos de los subsistemas que lo forman. Más formalmente, un sistema es no lineal cuando las ecuaciones de movimiento o evolución que regulan su comportamiento son no lineales. En particular, el comportamiento de sistemas no lineales no está sujeto al principio de superposición, como lo es en un sistema lineal.

En este proyecto se trabaja con elementos no lineales como es el amplificador[5]. El comportamiento no lineal se hace notar a medida que se aumenta la potencia de trabajo de los equipos: a mayor potencia, mayor es el efecto de la no linealidad en el sistema.

Además, los sistemas reales son no lineales en su totalidad y, por tanto, el resultado a la salida no es proporcional a la entrada. Por otra parte, la señal obtenida a la salida contendrá un conjunto de componentes no deseadas en diferentes frecuencias que deben estudiarse para caracterizar el sistema. Si se introduce un tono, a la salida se añadirían otros tonos en frecuencias distintas. La relación entre la entrada x y las componentes que se añaden a la salida y tiene la siguiente expresión polinómica:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

Observando la ecuación, se puede ver cómo aparecen nuevos componentes de continua que se suman a unos términos de distorsión que son a su vez potencias consecutivas de la señal de entrada. A este efecto se le conoce como distorsión y se manifiesta con la aparición de distorsión armónica y productos de intermodulación.

5.2.2. Distorsión armónica

Si en un sistema no lineal se introduce un tono con frecuencia central f_0, a la salida se obtiene, además del tono amplificado, otros tonos de distinta frecuencia en $2f_0$, $3f_0$, $4f_0$… Estas componentes reciben el nombre de armónicos.

La aparición de armónicos modifica la señal deseada distorsionándola. La eliminación de estas componentes se realiza mediante el uso de filtros, permitiendo con ellos que solo se deje pasar el tono fundamental que se desee obtener a la salida.

[5] Su diseño basado en transistores BJT o FET, entre otros, hace que sea un dispositivo no lineal y presente los efectos característicos de los mismos.

5.2.3. Punto de compresión de 1 dB

En un sistema no lineal, a medida que la potencia de entrada aumenta, la ganancia se reduce produciéndose una "compresión de la ganancia". Siendo el punto de compresión de 1 dB el instante en el que la potencia de salida es 1 dB inferior a la que idealmente se tendría en un sistema lineal.

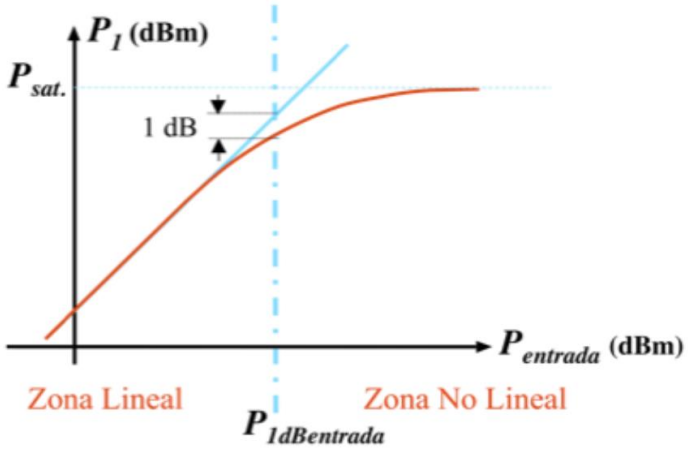


Figura 5-3. Punto de compresión de 1 dB.

5.2.4. Productos de intermodulación

Al introducir dos o más señales en un dispositivo no lineal aparecen, además de las componentes de entrada y los armónicos, unas componentes de mezcla. Por ejemplo, si se introducen dos tonos de frecuencia f_1 y f_2, a su salida se verán los productos de intermodulación en las frecuencias suma y diferencia de múltiplos de las frecuencias originales, es decir, como se observa en la siguiente ecuación:

$$f_{int} = af_1 \pm bf_2,$$

siendo a y b números enteros mayores que uno, ya que en los casos donde cualquiera de estos factores es igual a cero se produce la distorsión armónica explicada en el apartado anterior.

En una señal multicanal donde existan muchas más señales con diferentes frecuencias, los productos de intermodulación pueden caer encima de las frecuencias de interés resultando imposible su filtrado. Este tipo de problemas suele causar dificultades en la práctica a la hora de obtener un resultado deseado.

5.2.5. Punto de intercepto de tercer orden, IP_3

Punto de corte de la potencia de salida lineal (del tono fundamental) y la potencia del producto de intermodulación de tercer orden (TOI, del inglés *Third Order Input*).

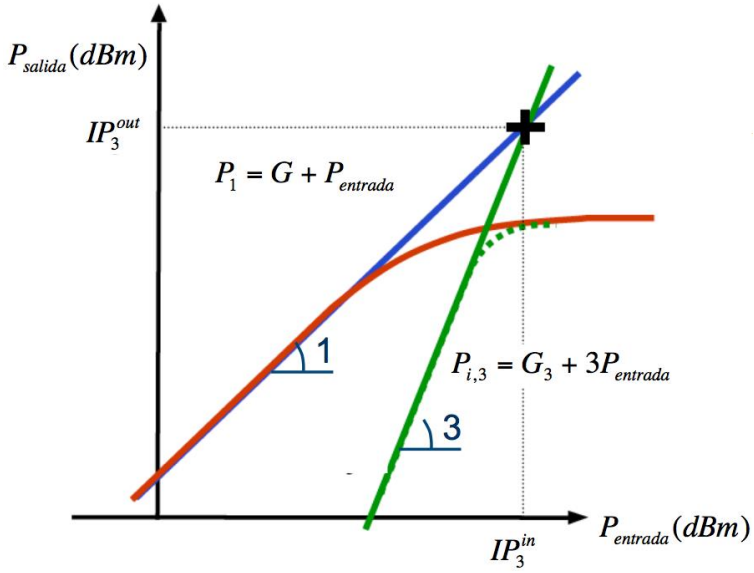


Figura 5-4. Punto de intercepto de tercer orden.

En este apartado cabe destacar la relación de protección (RP), que viene dada por la distancia mínima entre la potencia de salida del tono fundamental y la potencia del producto de intermodulación de tercer orden. Se representa mediante la siguiente ecuación:

$$IP_3^{out}(dBm) = \frac{3P_{1,salida}(dBm) - P_{i,3}(dBm)}{2}$$

$$P_{1,salida}(dBm) - P_{i,3}(dBm) = 2(IP_3^{out}(dBm) - P_{1,salida}(dBm)) \geq RP(dB)$$

5.2.6. ACPR

Una de las figuras de mérito que los investigadores utilizan para saber si existe una respuesta no lineal en un dispositivo es la distorsión fuera de banda. Los sistemas no lineales generan dicha distorsión, que es detectable en el recrecimiento espectral de los canales de comunicación adyacentes. El recrecimiento espectral puede ser cuantificado por medio de la relación de potencia de canal adyacente (ACPR) , usualmente definida como:

$$ACPR = \frac{P_{adj}}{P_{inband}},$$

donde P_{adj} es la potencia total en el canal adyacente y P_{inband} es la potencia en el canal deseado. La mayoría de los analizadores de espectro o analizadores de señales vectoriales incluyen módulos para el cálculo de esta figura de mérito, siendo el resultado similar al que se muestra en la Figura 5-5. En esta figura, se puede ver el canal central, que contiene la información deseada a transmitir y los canales adyacentes. Como sólo son preocupantes los canales adyacentes inferior o superior, es necesario especificar de qué manera se sabe qué canal está implicado en el cálculo de la ACPR. En las fórmulas que prosiguen, $ACPR_l$ está referenciado al canal adyacente inferior y $ACPR_u$ al canal adyacente superior.

$$ACPR_l = \frac{P_{adj,l}}{P_{inband}}$$

y,

$$ACPR_u = \frac{P_{adj,u}}{P_{inband}},$$

donde $P_{adj,l}$ y $P_{adj,u}$ son la potencia del canal adyacente inferior y superior, respectivamente. En el dominio de la frecuencia, cualquiera de los valores de potencia anteriores (representado por $\{\cdot\}$) se puede calcular como:

$$P_{\{\cdot\}} = \int_{f_1}^{f_2} S_{\{\cdot\}}(f)df,$$

donde f_1 y f_2 son los límites izquierdo y derecho del canal considerado, respectivamente, siendo $S(f)$ la densidad espectral de potencia.

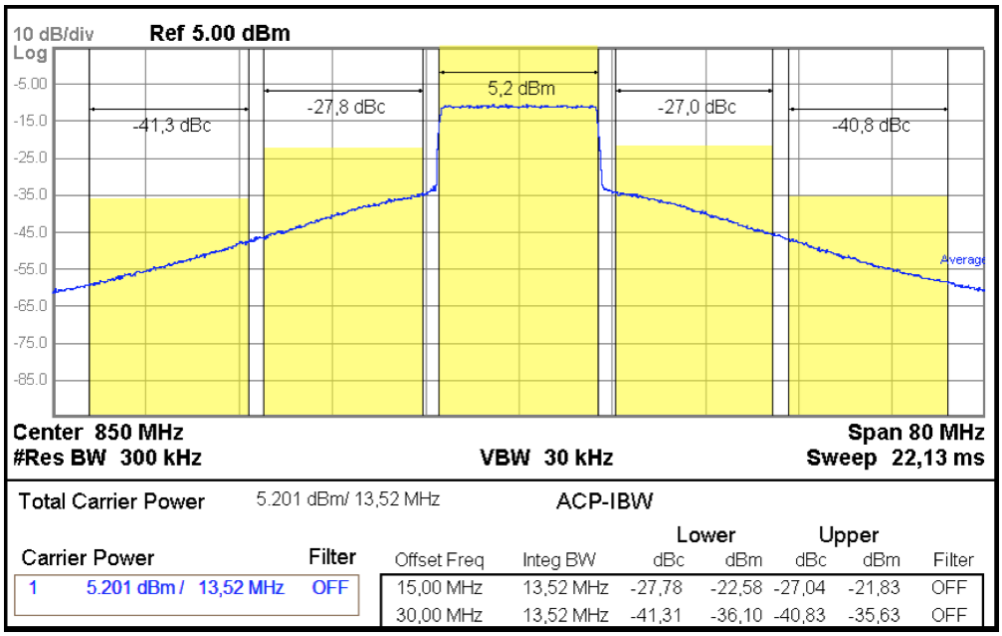


Figura 5-5. Figura tomada de un analizador de espectro en modo medición de potencia de canal adyacente.

6 DESARROLLO DEL CÓDIGO

En este capítulo se detalla la funcionalidad de cada parte del código realizado durante el trabajo (el código se incluye completo en el Anexo 3), cuya finalidad es la generación y posterior análisis de la salida, posiblemente no lineal, del dispositivo bajo prueba ante una excitación de un tono. Con dicho ejercicio, se han llevado a cabo dos pruebas, que se analizan en el Capítulo 7 de esta memoria.

6.1. Introducción

Se comienza el capítulo analizando cómo ha de generarse la conexión remota entre el ordenador y un instrumento de laboratorio mediante el método PyVISA, con el uso de objetos.

A continuación, se ven las funciones de lectura y escritura creadas para facilitar las comunicaciones que hay que llevar a cabo en el trabajo.

Posteriormente, se analizan los tipos de medidas que se pueden hacer describiendo cuáles son las configuraciones que se controlan tanto en el analizador de espectro como en el generador de señal, detallando los parámetros que podemos modificar y cómo jugar con las distintas opciones para estudiar las características de los dispositivos.

Por último, se definen las funciones comunes a todos los tipos de medidas, como son las de extraer los valores de potencia máxima en el analizador, guardar los resultados en un archivo externo e imprimir la gráfica de resultados.

6.2. Conexión remota

Para abrir una conexión entre un instrumento del laboratorio y un ordenador mediante Python se usan las órdenes de la Figura 6-1.

```
import visa
rm=visa.ResourceManager()
InstrObj=rm.open_resource('GPIB0::18::INSTR')
```

Figura 6-1. Apertura de conexión remota en Python.

En el caso del ejemplo de la Figura 6-1, la comunicación que se está iniciando es con el puerto 18 de un bus GPIB, que en este trabajo pertenece al analizador de señal porque así lo tiene asignado. En caso de tener varios instrumentos conectados en el mismo instante y no saber qué puerto de conexión posee cada uno, hay que listarlos todos, iniciar la conexión y hacerles preguntas para que se identifiquen y así saber con quién se quiere trabajar. Eso se realiza mediente el siguiente código.

```
import visa
rm=visa.ResourceManager()
a=rm.list_resources()
b=rm.open_resource(a[0])
print(b.query("*IDN?"))
```

Figura 6-2. Apertura de conexión con listado de instrumentos posibles.

El método *list_resources()* es el que se encarga de listar todos los instrumentos que se encuentran conectados al ordenador, mostrando además los puertos en los que se sitúan. Mediante la orden SCPI "*IDN?" se muestra la identificación de cada uno de ellos, pudiendo así relacionarlos con el puerto que tienen configurado. De esta manera es posible realizar la apertura de la conexión con el dispositivo que interese.

Además de los métodos vistos, existen otros para modificar parámetros de la conexión que hay que tener en cuenta en este trabajo. Los más significativos son los dos siguientes:

- *timeout*: Establece un tiempo de espera. Todas las operaciones que tardan más que el tiempo establecido se abortan y se genera una excepción. Los tiempos de espera son dados en milisegundos.
- *chunk_size*: PyVISA lee desde el dispositivo en trozos, cada uno con un tamaño de 20 kilobytes por defecto. Si cuando lee uno aun quedan datos por leer, repite el procedimiento y va concatenando los trozos. Sin embargo, esto puede dar problemas con algún dispositivo. Por lo tanto, antes de empezar un intercambio de mensajes con el instrumento es recomendable modificar el tamaño del *buffer* mediante este método.

En la siguiente figura se muestra un ejemplo en el cual se modifican dos parámetros del objeto creado en el código del comentario anterior: el tiempo de espera se fija a 25 segundos y el tamaño del *buffer* a 100 kilobytes.

```
b.timeout=25000
b.chunk_size=102400
```

Figura 6-3. Ejemplo de modificación del *timeout* y *chunk_size*.

6.3. Lectura y escritura en el instrumento

Para la escritura y lectura de comandos SCPI en los instrumentos se han creado dos *scripts* para trabajar de manera más cómoda a lo largo del proyecto (*send_command.py* y *send_query.py* respectivamente). En cuanto a las órdenes utilizadas para ello hay que diferenciarlas en dos tipos, para entender el sentido de los *scripts* nombrados anteriormente:

- Órdenes *No Query*: Solo necesitan un comando de escritura. Se le manda una acción al instrumento sin que responda con ningún valor. Ejemplo: *WAI.
- Órdenes *Only Query*: Necesitan un comando de escritura y posteriormente uno de lectura para extraer la información que se le requiere al instrumento. Son órdenes de acción-reacción ya que el dispositivo devuelve un valor tras procesarla. Todos los comandos SCPI de tipo *Only Query* finalizan con el símbolo de interrogación. Ejemplo: *IDN? .

```
InstrObj.write(strCommand)
```

Figura 6-4. Formato de órdenes *No Query*, utilizada en la función *send_command.py*.

```
InstrObj.write(strCommand)
Result=InstrObj.read_raw()
```

Figura 6-5. Formato de órdenes *Only Query*, utilizada en la función *send_query.py*.

Como se observa en las Figuras 6-4 y 6-5 se utilizan las variables "InstrObj" y "strCommand". Ambas son pasadas como parámetro en las funciones de escritura (*send_command.py*) y escritura/lectura (*send_query.py*). Se refieren, respectivamente, al objeto creado para la comunicación remota con los instrumentos (de la manera explicada en el apartado 6.2) y al comando SCPI que hay que mandar en la orden de escritura al dispositivo.

En el *script* de escritura/lectura, *query*, se utiliza el método *read_raw()*, que se encarga de leer el valor devuelto por el instrumento tras la petición realizada con la orden *write()*.

6.4. Tipos de barrido

En este trabajo, para las dos pruebas que se han llevado a cabo, se han realizado dos tipos de barrido en cada una de ellas, de potencia y de frecuencia. Todos los parámetros que se requieren para cada tipo se piden de forma ordenada en el programa principal (*medida_1tono_sinDC.py*) y son introducidos por el usuario que lo ejecuta. En el Capítulo 7 de esta memoria se indican los valores necesarios para cada prueba realizada.

```
print ("Introduzca el número asociado al tipo de barrido que desea realizar:")
print ("1.- Barrido de potencia.")
print ("2.- Barrido de frecuencia.")
tipo_barrido = int(raw_input("Tipo de barrido:    "))
```

Figura 6-6. Tipos de barrido.

Una vez el programa sabe el tipo de barrido que el usuario desea realizar, pide una serie de parámetros. Algunos de ellos son comunes a los dos tipos y otros son característicos. En la Figura 6-7, situada en la siguiente página en horizontal debido a su tamaño, se aprecian los parámetros comunes y en los apartados posteriores se indican los individuales con sus formas de procesamiento, para a posteriori, realizar la medida de potencia, finalidad del código desarrollado.

Los parámetros comunes que se piden para modificar los valores de cada tipo de barrido previamente a la medida del valor de la potencia son:

- Conversión de frecuencias: Si se va a realizar o no. Para trabajar con dispositivos, como por ejemplo mezcladores, es necesario considerar una frecuencia a su entrada que es diferente a la frecuencia a su salida porque el dispositivo realiza una conversión. En cambio, en el caso del amplificador de potencia, no tiene sentido realizarla.
- Ganancia del dispositivo: Es necesario indicar la ganancia esperada del dispositivo bajo prueba para configurar de forma adecuada el nivel de referencia del analizador de espectro (*Reference Level*), de forma que los valores a medir no se salgan del rango de medida ni queden demasiado alejados del nivel de referencia con la consiguiente pérdida de precisión en la medida.
- Número de armónicos con el que se desea trabajar.
- Valor de pérdida a la entrada, debida principalmente a los cables y conectores empleados para el montaje del dispositivo bajo prueba.
- Valores de pérdidas a la salida, también debidas a los cables y conectores empleados en el montaje, en este caso particularizadas para las diferentes frecuencias en las que se medirá.

En el siguiente capítulo, en el apartado en el que se analiza cada medida llevada a cabo en el laboratorio, se indican los valores introducidos para cada uno de los datos anteriores, cuya finalidad es conseguir una comprensión mejor del resultado obtenido.

```
mixer = float(raw_input("Introduzca un valor distindo de 0 si el dispositivo a medir realiza conversión de frecuencias:   "))
gain = float(raw_input("Introduzca la ganancia esperada del dispositivo (valor positivo en dB):   "))
while gain < 0:
    print("Valor no válido de ganancia. Debe ser positiva.")
    gain = float(raw_input("Introduzca la ganancia esperada del dispositivo (valor positivo en dB):   "))

num_armonicos = float(raw_input("Introduzca el numero de armonicos que desea medir (entre 1 y 5):   "))
num_armonicos=int(num_armonicos)
while (num_armonicos != 1) and (num_armonicos != 2) and (num_armonicos != 3) and (num_armonicos != 4) and (num_armonicos != 5):
    print("Numero de armónicos no válido. Debe estar entre 1 y 5.")
    num_armonicos = float(raw_input("Introduzca el numero de armonicos que desea medir (entre 1 y 5):   "))
    num_armonicos=int(num_armonicos)

correccion_in = float(raw_input("Introduzca el valor de las perdidas a la entrada (en dB):   "))
while correccion_in < 0:
    print("Las perdidas a la entrada deben ser positivas.")
    correccion_in = float(raw_input("Introduzca el valor de las perdidas a la entrada (en dB):   "))
Pin = P - correccion_in
Pin=np.matrix(Pin)

correccion_out_cad = raw_input("Introduzca un vector de 5 elementos con el valor de las perdidas a la salida (en dB) ([valor1, valor2, ...]):   ")
correccion_out = eval(correccion_out_cad)
while len(correccion_out) != 5 or all(i >= 0 for i in correccion_out) == False:
    print("Todas las pérdidas a la salida deben ser positivas y el vector debe contener 5 valores.")
    correccion_out_cad = raw_input("Introduzca un vector de 5 elementos con el valor de las perdidas a la salida (en dB) ([valor1, valor2, ...]):   ")
    correccion_out = eval(correccion_out_cad)
```

Figura 6-7. Parámetros comunes a los dos tipos de barrido.

6.4.1. Barrido de potencia

El objetivo de este tipo de barrido es obtener los valores de la potencia que presentan los tonos a diferentes frecuencias que constituyen los armónicos que se desean medir para un rango de potencias variables a la entrada del dispositivo conectado y con una frecuencia fundamental fija.

Para llevar a cabo esta tarea, se genera una señal en el generador de señales con frecuencia fija y con variaciones de su nivel de potencia en un barrido, abarcando un rango de potencias configurable por el usuario que se explica en los párrafos siguientes. Para cada tono y nivel de potencia, se busca el valor de los armónicos en las frecuencias correspondientes en el analizador de espectro, que dependiendo del número de armónicos que se quieran ver estarán situados en $f_{central}$, $2f_{central}$, $3f_{central}$, $4f_{central}$ y $5f_{central}$.

Los parámetros que se requieren en el caso de barrido de potencia vienen indicados en la siguiente imagen.

```
if tipo_barrido == 1:
    p_start = float(raw_input("Introduzca la potencia inicial (en dBm):   "))
    p_stop = float(raw_input("Introduzca la potencia final (en dBm):    "))
    p_inc = float(raw_input("Introduzca el incremento de potencia (en dB):    "))
    p_final = p_stop + (p_inc/2)
    p_tam = len(np.arange(p_start, p_final, p_inc))
    P = np.arange(p_start, p_final, p_inc).reshape((p_tam, 1))
    P=np.matrix(P)

    if mixer != 0:
        Fi = float(raw_input("Introduzca la frecuencia central de la entrada (en MHz):    "))
        Fo = float(raw_input("Introduzca la frecuencia central de la salida (en MHz):    "))
    else:
        Fi = float(raw_input("Introduzca la frecuencia central (en MHz):    "))
        Fo = Fi

    vector = P
```

Figura 6-8. Barrido de potencia.

La variable "P" es el vector de potencias comentado anteriormente, que tiene como valor de comienzo "p_start", valor de fin "p_stop" e incremento "p_inc", sendos tres datos pedidos por pantalla.

Una vez se tengan todos los valores, es necesario procesarlos para que sean válidos en el uso de la función *medida.py*, que se analiza en el apartado 6.5. Lectura de potencia.

```
if isinstance(P, float):
    Pv = P*ones(len(vector))
else:
    Pv = P
if tipo_barrido==1:
    Fiv = Fi*ones(len(vector))
    Fov = Fo*ones(len(vector))
```

Figura 6-9. Procesamiento de las variables en caso de barrido de potencia.

Se observa que en caso de ser la variable "P" de tipo *float* (únicamente en un barrido de frecuencia), se convierte en un vector del tamaño de la variable "vector" para que tengan todos los datos el mismo tamaño.

Para las variables “Fiv” y “Fov” se realiza el mismo procedimiento.

6.4.2. Barrido de frecuencia

Al igual que en el caso anterior, el objetivo de este tipo de barrido es obtener los valores de la potencia que presentan los tonos a diferentes frecuencias que constituyen los armónicos indicados por el usuario. Pero en este caso se realiza para un rango de frecuencias variables a la entrada y con un valor de potencia constante.

Para ello, se genera una señal en el generador de señales con potencia fija y abarcando un rango de frecuencias configurable por el usuario en cada barrido. Para cada tono y valor de frecuencia, se buscan los armónicos en las frecuencias correspondientes en el analizador de espectro, que dependiendo del número de armónicos que se quieran ver estarán situados en $f_{central}$, $2f_{central}$, $3f_{central}$, $4f_{central}$ y $5f_{central}$.

Los parámetros que se requieren en el caso de barrido de frecuencia vienen indicados en la Figura 6-11, situada en la siguiente página en horizontal debido a su tamaño. En este tipo de barrido existen dos casos bien diferenciados dependiendo de si hay conversión de frecuencias o no. Al ser un barrido en frecuencia este valor toma más protagonismo que en el caso anterior.

El vector a formar toma el nombre de “Fi” y viene dado por las variables introducidas por el usuario llamadas “f_start”, “f_stop” y “f_inc” (valores de inicio, fin e incremento). En el caso de que haya conversión de frecuencias hay que destacar la situación en la que se opera en *down-converter*[6], que modifica el valor de la frecuencia de salida “Fo” (por ejemplo, hay que tenerla en cuenta en la prueba que se realizará en el capítulo siguiente con el mezclador).

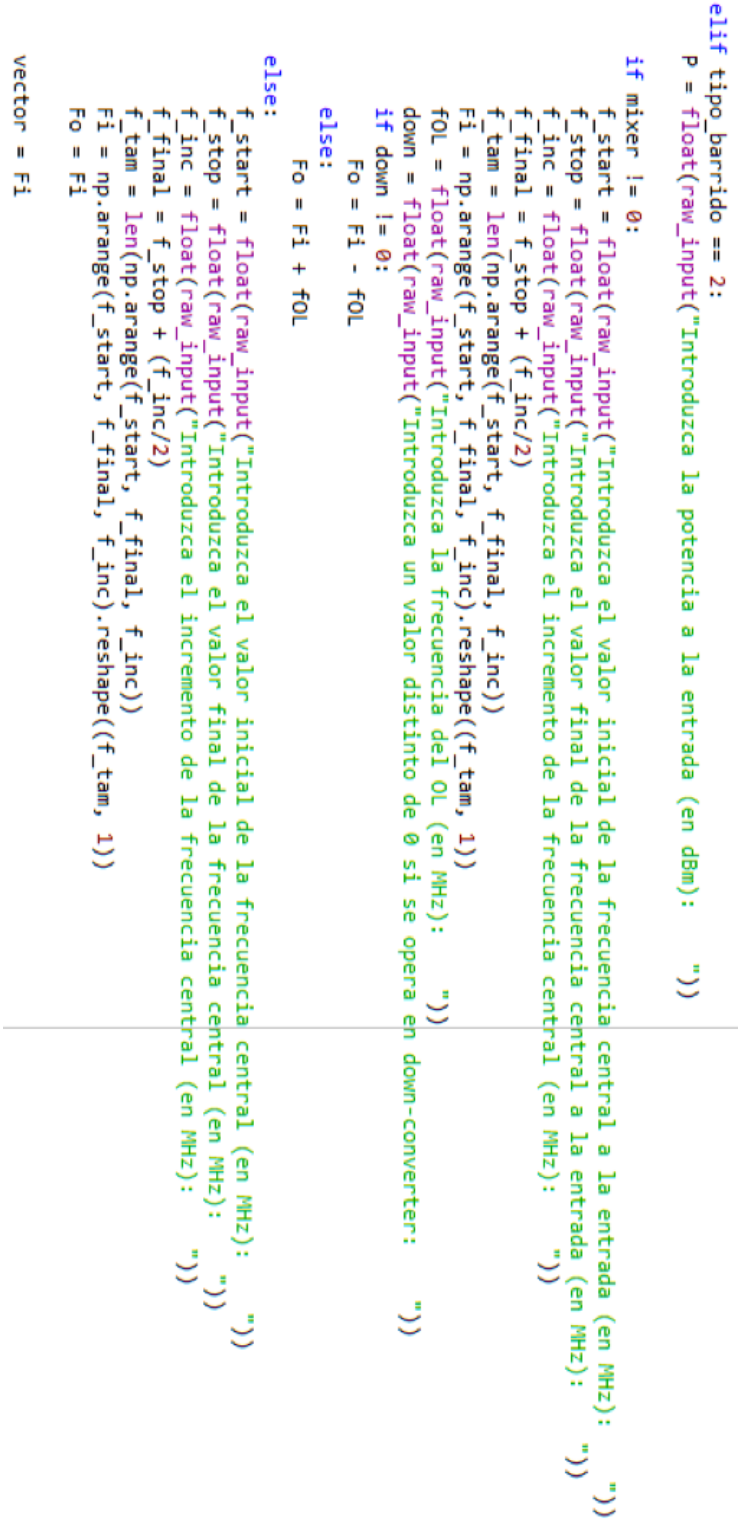
Una vez se tengan todos los valores es necesario procesarlos para que sean válidos en el uso de la función *medida.py*, que se analiza en el apartado 6.5. Lectura de potencia.

```
if isinstance(P, float):
    Pv = P*ones(len(vector))
else:
    Pv = P
if tipo_barrido==2:
    if len(Fi) == 1:
        Fiv = Fi*ones(len(vector))
        Fov = Fo*ones(len(vector))
    else:
        Fiv = Fi
        Fov = Fo
```

Figura 6-10. Procesamiento de las variables en caso de barrido de frecuencia.

Se observa que hay que obtener los mismos valores que en el barrido de potencia, solo que hay que tener en cuenta el vector “Fi”, ya que en caso de que el tamaño de este fuera la unidad daría error si se hace de la forma anterior.

[6] Convierte una señal de radiofrecuencia limitada en banda en una señal de frecuencia inferior llamada frecuencia intermedia para simplificar las etapas de radio siguientes en la estructura típica de un receptor superheterodino.



```
elif tipo_barrido == 2:
    P = float(raw_input("Introduzca la potencia a la entrada (en dBm):    "))
    if mixer != 0:
        f_start = float(raw_input("Introduzca el valor inicial de la frecuencia central a la entrada (en MHz):    "))
        f_stop = float(raw_input("Introduzca el valor final de la frecuencia central a la entrada (en MHz):    "))
        f_inc = float(raw_input("Introduzca el incremento de la frecuencia central (en MHz):    "))
        f_final = f_stop + (f_inc/2)
        f_tam = len(np.arange(f_start, f_final, f_inc))
        Fi = np.arange(f_start, f_final, f_inc).reshape((f_tam, 1))
        fOL = float(raw_input("Introduzca la frecuencia del OL (en MHz):    "))
        down = float(raw_input("Introduzca un valor distinto de 0 si se opera en down-converter:    "))
        if down != 0:
            Fo = Fi - fOL
        else:
            Fo = Fi + fOL
    else:
        f_start = float(raw_input("Introduzca el valor inicial de la frecuencia central (en MHz):    "))
        f_stop = float(raw_input("Introduzca el valor final de la frecuencia central (en MHz):    "))
        f_inc = float(raw_input("Introduzca el incremento de la frecuencia central (en MHz):    "))
        f_final = f_stop + (f_inc/2)
        f_tam = len(np.arange(f_start, f_final, f_inc))
        Fi = np.arange(f_start, f_final, f_inc).reshape((f_tam, 1))
        Fo = Fi

vector = Fi
```

Figura 6-11. Barrido de frecuencia.

6.5. Lectura de potencia

Para obtener y guardar todos los valores de potencia que se dan en una ejecución del programa de la mejor manera posible, se hace uso de un bucle *while* y de una matriz de resultados que se va rellenando en cada iteración con las medidas de potencia obtenidas en el analizador de espectro.

Cada una de las iteraciones del bucle son los valores del vector que se crea al principio del programa al indicar el tipo de barrido, "P" para un barrido de potencia y "Fi" para uno de frecuencia. Las variables anteriores corresponden con el rango de potencias y frecuencias configurado por el ususario una vez se sabe el tipo de análisis que se quiere llevar a cabo.

Para realizar la lectura de potencia en cada punto, al comienzo de cada iteración es necesario configurar el generador vectorial de señales para que active la señal de un tono adecuada, que se aplicará a la entrada del dispositivo bajo prueba. Para ello, se utiliza la función *un_tono_SMIQ.py*, cuyas partes más significativas se muestran es las Figuras 6-12 y 6-13.

```
import visa
rm=visa.ResourceManager()
smiq=rm.open_resource('GPIB0::28::INSTR')
```

Figura 6-12. Apertura de la conexión con el generador de señales.

```
freq=str(frecuencia)
send_command.send_command(smiq, ("FREQ %s MHz") %freq)
pot=str(potencia)
send_command.send_command(smiq, ("POW %s dBm") %pot)
send_command.send_command(smiq, ("OUTP:STAT ON"))
```

Figura 6-13. Configuración del generador de señales con los valores de frecuencia y potencia.

Para la configuración del generador de señales, como se aprecia en las imágenes anteriores, hay que seguir los siguientes pasos:

- En primer lugar, abrir una conexión remota con el instrumento (generador de señales, SMIQ) para poder mandarle los comandos SCPI. En el caso de este proyecto con el puerto 28 del bus GPIB, ya que es al que está relacionado el generador del laboratorio de radiocomunicación con el que se ha trabajado.
- Posteriormente, configurar en el mismo los valores de frecuencia y potencia a los que el usuario quiere realizar la medida mediante órdenes de escritura en el objeto donde se guarda comunicación. Cabe destacar que la última orden que se aprecia en la Figura 6-13 equivale a pulsar la tecla RF On en el equipo, es decir, con ella se activa la salida de RF del mismo.

Una vez configurado el generador, hay que hacer lo mismo con el analizador de espectro para poder leer la medida de potencia de manera correcta. Esto se realiza mediante la función *medida.py*, cuya llamada en el programa principal se realiza es usando la sintaxis del siguiente ejemplo:

```
medida_1t[k][0] = medida.medida(Fova, span, rlevel, BWres, ade)
```

Figura 6-14. Llamada a la función *medida.py* en el bucle del *script medida_1tono_sinDC.py*.

La variable "medida_1t" es la asociada a la matriz de resultados nombrada con anterioridad, de la forma:

$$\begin{pmatrix} P_{A_{11}} & P_{A_{21}} & \cdots & P_{A_{n1}} \\ P_{A_{12}} & P_{A_{22}} & \cdots & P_{A_{n2}} \\ \vdots & \vdots & \ddots & P_{A_{n3}} \\ P_{A_{1m}} & P_{A_{2m}} & \cdots & P_{A_{nm}} \end{pmatrix},$$

donde $P_{A_{nm}}$ es la potencia medida en el analizador en cada iteración en el armónico n y con las condiciones del barrido del tono en la posición m.

En la función *medida.py* se trabaja con el analizador, modificando en el mismo los 5 parámetros que se le pasa:

- "Fova": Valor de la frecuencia a la salida del dispositivo bajo prueba que se desea medir, que será la frecuencia fundamental o uno de sus armónicos. Este valor se configurará como frecuencia central en el analizador de espectro para poder visualizar correctamente el tono a medir.
- "span": El parámetro *span* indica la anchura del segmento de espectro mostrada en el analizador. El valor de este parámetro se selecciona experimentalmente, de forma que solo se visualice un tono en el analizador de espectro para medir correctamente. También se tiene en cuenta fijar un valor lo suficientemente pequeño de *span* para facilitar la selección del ancho de banda de resolución que produce la medida con la precisión adecuada.
- "rlevel": Nivel de referencia, que representa el mayor nivel de potencia que muestra el analizador de espectro de pantalla. El nivel de referencia debe escogerse de manera que el tono a medir se visualice en pantalla, para lo cual *rlevel* debe ser mayor que la potencia esperada del tono. Por otro lado, *rlevel* no debe distar demasiado del valor de potencia a medir porque se perdería precisión en la medida. Los valores a asignar a este parámetro se eligen de forma experimental atendiendo a los dos criterios anteriores. Por ejemplo, para medir el tono a la frecuencia fundamental, el valor adecuado de *rlevel* debe ser mayor que la potencia generada por el dispositivo a su salida. En cambio, para los armónicos *rlevel* tendrá una corrección negativa con respecto al valor que tomaría para la frecuencia fundamental porque la potencia a medir es menor.
- "BWres": Ancho de banda de resolución. Este parámetro está relacionado con la capacidad del analizador de espectro de medir potencias que pueden ser muy débiles, ya que permite reducir el fondo de ruido de la medida. Su valor se fija experimentalmente para cada uno de los armónicos a medir, teniendo en cuenta cuáles de ellos podrían ser señales muy débiles.
- "ade": Variable en la que se guarda el objeto de la comunicación con el analizador de señal.

El ejemplo de llamada a la función de la Figura 6-14 se corresponde con la medida de la potencia que hay a la salida del dispositivo para la frecuencia fundamental. En este caso, la estimación que se realiza para el nivel de referencia tiene en cuenta el valor de la potencia aplicada a la entrada del dispositivo, dado por el vector de potencias guardado en la variable "P", corregido con la ganancia y un pequeño margen:

$$rlevel = Pva + gain + 15 \text{ dB}$$

Tras un estudio de comportamiento de los armónicos en el laboratorio, se llegó a la conclusión de cómo deben ser los valores a configurar en el analizador para conseguir los resultados correctos. Para la configuración del *span* y ancho de banda de resolución se ha observado que el valor que hace que la medida sea fiable es el resultado de las siguientes operaciones:

$$span = \frac{f_{central}}{100}$$

$$BWres = \frac{span}{500}$$

Modificación de los cinco parámetros comentados anteriormente en el analizador de espectro:

```
frec2=str(frec1)
send_command.send_command(ade, (":SENS:FREQ:CENT %s MHZ; *WAI") %frec2)
span2=str(span)
send_command.send_command(ade, (":SENS:FREQ:SPAN %s MHZ; *WAI") %span2)
send_command.send_command(ade, ("UNIT:POW DBM"))
rlevel2=str(rlevel)
send_command.send_command(ade, (":DISP:WIND:TRAC:Y:RLEV %s DBM; *WAI") %rlevel2)
send_command.send_command(ade, (":DISP:WIND:TRAC:Y:LOG:RANG:AUTO OFF"))
BWres2=str(BWres1)
send_command.send_command(ade, ("BAND %s MHz; *WAI") %BWres2)
```

Figura 6-15. Configuración del analizador de señales con los parámetros pasados como argumento.

Por último, tras configurar el analizador, solo queda realizar la medida de la máxima potencia que se observa en la pantalla del equipo, es decir, el valor de potencia que se busca. Para ello se ejecuta la línea de código de la Figura 6-16.

```
(Status, potencia)=send_query.send_query(ade, (":CALC:MARK:Y?"))
```

Figura 6-16. Lectura del valor de potencia.

El segundo valor que devuelve la orden de la Figura 6-16, "potencia", es el que entrega la función *medida.py*. Esta cifra varía a medida que se ejecuta el bucle *while* y va siendo almacenada en la variable "medida_1t" hasta completar la matriz de resultados. La cantidad de datos no nulos en ella (valores en los que se ha leído potencia) depende del número de armónicos introducidos por pantalla.

Hay que tener en cuenta que la línea de código de la Figura 6-14 es para obtener la medida de potencia en caso de que el número de armónicos sea igual o superior a 1. Para una mayor cantidad de armónicos (en el código se tiene en cuenta hasta el quinto) la orden cambia levemente su llamada. Los dos parámetros que van cambiando a medida que sube el orden del armónico a medir son la frecuencia de salida, "Fova", y el valor de *reference level*, "rlevel".

Como se ha comentado anteriormente los armónicos van a estar situados en las frecuencias $f_{central}$, $2f_{central}$, $3f_{central}$, $4f_{central}$ y $5f_{central}$, por tanto, el valor de la variable "Fova" que se pasa a la función *medida.py* se va multiplicando de esta misma manera para situar el analizador de espectro en el lugar correcto para hacer la medida.

En cuanto a "rlevel" es algo más complejo ya que varía dependiendo del armónico. Como ya se ha mencionado, inicialmente se determina como el valor de la potencia aplicada a la entrada del dispositivo corrregido con la ganancia y un margen de 15 dB. Inicialmente se determina un valor resultado de sumar $25\ dB$ a la potencia del tono que se está evaluando. Partiendo de este nivel inicial, el valor de *reference level* para cada ármonico viene dado según se indica en la Tabla 6-1.

Tabla 6–1. Valores de "rlevel" para cada armónico.

Número de armónico	**Valor de "rlevel"**
Primero o tono fundamental	$rlevel_{inicial}$
Segundo	$rlevel_{inicial} - 10\ dB$
Tercero	$rlevel_{inicial} - 15\ dB$
Cuarto	$rlevel_{inicial} - 20\ dB$
Quinto	$rlevel_{inicial} - 25\ dB$

Finalmente, cuando se ha obtenido la matriz con todos los valores de potencia de la prueba realizada, es necesario hacer las correciones debidas a las pérdidas en la salida.

```
medida_1t=np.matrix(medida_1t)
correccion_out=np.tile(correccion_out, [len(vector), 1])
correccion_out=np.matrix(correccion_out)
medida_1t = medida_1t + correccion_out
```

Figura 6-17. Matriz de valores de potencia corregidos.

6.6. Impresión de resultados

Los resultados en el trabajo se imprimen de dos formas distintas, mediante una gráfica y en un archivo aparte con el nombre y en el directorio que indique el usuario. Cada uno de los dos tipos se puede diferenciar a su vez dependiendo del barrido que se realice. A continuación, se muestran imágenes del código con el que se han realizado estos formatos.

6.6.1. Archivo de valores finales

En el caso de un barrido de potencia, el que se aprecia en la Figura 6-18, se guarda en el documento de resultados el vector de potencias de entrada (con las correcciones por las pérdidas de los cables a la entrada) y la matriz de valores de potencia corregidos.

En el de un barrido de frecuencia se guardan los vectores de frecuencia de entrada y de salida y la matriz de valores de potencia corregidos.

```
import numpy as np
archivo=open(nombre, 'w')
archivo.write("Pin:")
archivo.write("\n\n")
Pin=np.matrix(Pin)
i=0
for i in Pin:
    i=float(i)
    archivo.write(str(round(i,3)))
    archivo.write("\n")
archivo.write("\n\n")
archivo.write("medida_1t:")
archivo.write("\n\n")
medida_1t=np.matrix(medida_1t)
i=0
for i in medida_1t:
    archivo.write(str(i))
    archivo.write("\n")
archivo.write("\n\n")
archivo.close()
del(archivo)
```

Figura 6-18. Generación del documento de resultados para un barrido de potencia.

6.6.2. Gráficas

```
plt.figure()
plt.plot(Fo,medida_1t[:,0], 'k')
plt.plot(Fo,medida_1t[:,1], 'b')
plt.plot(Fo,medida_1t[:,2], 'r')
plt.plot(Fo,medida_1t[:,3], 'm')
plt.plot(Fo,medida_1t[:,4], 'g')
plt.xlabel("Fout", fontsize=14, color='k')
plt.ylabel("Pout", fontsize=14, color='k')
plt.grid(True)
plt.grid(color='0.5', linestyle='--', linewidth=1)
plt.show()
```

Figura 6-19. Generación de la gráfica de resultados para un barrido de frecuencia.

En el caso de un barrido de frecuencia, el que se aprecia en la Figura 6-19, se representan los valores de potencia de salida (la matriz de resultados corregida) frente a los de frecuencia de salida.

En el de un barrido de potencia se representan los valores de potencia de salida, al igual que en el de frecuencia, pero frente a los valores de potencia de entrada (con la corrección de las pérdidas a la entrada).

7 RESULTADOS EXPERIMENTALES

7.1. Introducción

Una vez explicado el código realizado y su comportamiento es interesante comprobar el correcto funcionamiento del mismo. Para ello, se han realizado dos pruebas con distintos dispositivos a caracterizar (un amplificador de potencia y un mezclador). En los siguientes apartados se explican los resultados de un barrido de potencia y otro de frecuencia en cada uno de ellos. La hoja de características del amplificador y mezclador usados se facilita en los anexos 1 y 2 de esta memoria respectivamente.

Los pasos a seguir para la configuración de cualquier medida en el código son los siguientes:

- Ejecutar el programa principal que hace uso del resto de funciones, *medida_1tono_sinDC.py*.
- Indicar si se quiere realizar un barrido de potencia o de frecuencia por teclado.
- Introducir todos los parámetros que se piden por pantalla para cada tipo de medida teniendo en cuenta el elemento conectado entre el generador y el analizador de señal.
- Salvar los resultados en un documento si se desea, facilitando el directorio en el que guardarlos y el nombre que se le quiere poner al archivo.
- Finalmente aparecerá la gráfica de resultados en la consola de Python para que se pueda analizar el comportamiento de cada armónico a la salida del dispositivo con el que se esté simulando.

7.2. Consideraciones previas

En la realización de estas pruebas se tuvieron que tener en cuenta las limitaciones de los equipos antes de ejecutar el código:

- La frecuencia máxima con la que se configura el analizador no puede ser mayor que su frecuencia de trabajo, $26.5\ GHz$. Esta frecuencia máxima se corresponde con la equivalente al quinto armónico que se quiere estudiar. Debido a ello, se debe cumplir la siguiente ecuación:

$$9\ kHz\ <\ f_{central_{analizador}}\ <\ \frac{26.5\ GHz}{n^{\underline{o}}_{armónicos}}$$

- Como potencia final a la salida del dispositivo a medir no se puede escoger un valor superior al máximo que permite el analizador de espectro ($30\ dBm$).
- El generador de señal R&S SMIQ02B tiene un rango de frecuencias de trabajo que se encuentra entre $300\ kHz - 2.2\ GHz$.
- El generador de señal R&S SMIQ02B tiene una potencia máxima de aproximandamente $15\ dBm$ (el valor de la misma depende de la frecuencia de trabajo).
- El generador de señal R&S SMR20 tiene un rango de frecuencias de trabajo que oscila entre $1\ GHz - 20\ GHz$.

Para interpretar las gráficas de resultados hay que considerar que siempre se van a obtener las trazas de la componente fundamental o primer armónico (línea negra), segundo armónico (línea azul), tercer armónico (línea roja), cuarto armónico (línea violeta) y quinto armónico (línea verde).

7.3. Primera prueba: Comportamiento de un amplificador de potencia

7.3.1. Introducción

El amplificador que se va a usar para caracterizar su comportamiento no lineal es el modelo ZJL-6G+ del fabricante Mini-Circuits. Sus principales características son:

- Amplificador RF de baja potencia.
- Trabaja con frecuencias en el rango de $20\ MHz - 6000\ MHz$.
- Valor típico de ganancia de $13\ dB$ con un margen de $\pm 1.6\ dB$.
- Alimentación de $12\ V$.

Figura 7-1. Amplificador de potencia ZLJ-6G+.

7.3.2. Montaje

El proceso de esta aplicación consiste en introducir una señal a la entrada del amplificador mediante el generador SMIQ02B y observar cómo se comporta a su salida mediante el analizador de espectros EXA N9010A. Para llevar a cabo esta tarea es necesario alimentar el dispositivo con una fuente de tensión.

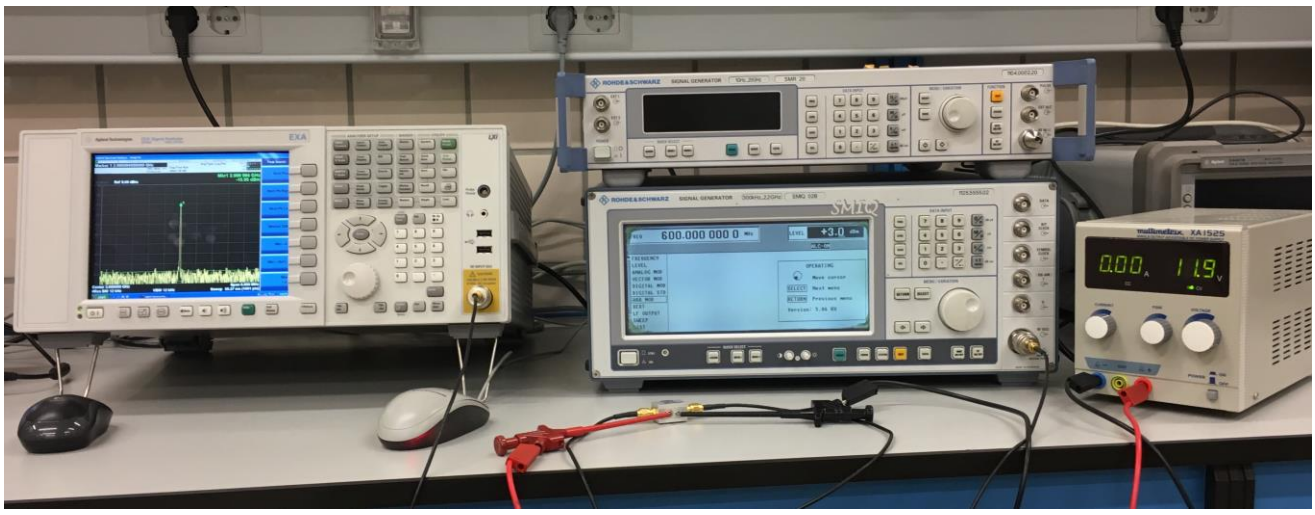


Figura 7-2. Escenario de la prueba 1.

Figura 7-3. Cable SMA hembra (izquierda) y macho (derecha).

Conexiones presentes en el escenario de la Figura 7-2:

- Ordenador con código en Python con analizador de señal Agilent EXA N9010A. Se realiza mediante una interfaz USB-GPIB Agilent 82357B. El ordenador se puede conectar indistintamente al analizador o al generador, se ha hecho de esta manera en particular por existir mayor espacio en el laboratorio.
- Analizador de señal Agilent EXA N9010A con generador de señal R&S SMIQ02B. Se conectan mediante bus GPIB estándar.
- Fuente de alimentación con amplificador ZLJ-6G+. La conexión se realiza mediante los cables banana-cocodrilo que se aprecian en la Figura 7-2 (cables rojo y negro conectados a la fuente de alimentación).
- Generador de señal R&S SMIQ02B con amplificador ZLJ-6G+ mediante un cable coaxial con conectores SMA[7] macho en los dos extremos. La salida RF-OUT del generador se conecta con la entrada IN del amplificador, ambos poseen conectores SMA hembra. En la Figura 7-3 se aprecia la diferencia entre los conectores SMA hembra y macho.
- Amplificador ZLJ-6G+ con analizador de señal Agilent EXA N9010A mediante un cable coaxial con conectores SMA macho en los dos extremos. La salida OUT del amplificador se conecta con la entrada RF-INPUT del analizador de espectro, ambos poseen conectores SMA hembra.

7.3.3. Medidas previas

Previamente a la ejecución de la aplicación, hay que analizar las pérdidas existentes en el circuito en cuanto a cables y conectores se refiere para obtener así un resultado más preciso y fiable en la caracterización del amplificador.

Para ello, hay que montar un escenario muy simple entre el analizador y el generador, conectando ambos (sin el amplificador) con uno de los cables SMA que se va a utilizar en el ejercicio. Es importante tener en cuenta para esta tarea las limitaciones de frecuencia de trabajo de los dos generadores de señal disponibles.

Siguiendo el siguiente proceso se obtienen las perdidas de los cables a la entrada y a la salida en las frecuencias de los cinco primeros armónicos. Los valores de frecuencia y potencia que se indican a continuación son los que se han usado para este ejercicio.

- Generar un tono en el generador de señal con una frecuencia de $600\ MHz$ y una potencia de $-25\ dBm$.

[7] Del inglés *SubMiniature version A*. Es un tipo de conector roscado para cable coaxial utilizado en microondas.

- Configurar el generador de señal a una frecuencia central de $600\ MHz$ y con un *span* de $10\ MHz$ para que se pueda apreciar el tono.
- Calcular el valor de las pérdidas de los cables con la ecuación que procede.

$$L_{cable} = P_{generador} - P_{analizador}$$

- Repetir los pasos anteriores para las frecuencias en las que se sitúan los armónicos, es decir, para $1200\ MHz$, $1800\ MHz$, $2400\ MHz$ y $3000\ MHz$.

Tras realizar este procedimiento se obtienen los siguientes valores de pérdidas con los que se van a trabajar:

Tabla 7–1. Pérdidas de los cables en el escenario de la primera prueba.

Componente	$\boldsymbol{L_{cable}(dB)}$
Componente Fundamental (600 MHz)	0.9
Segundo armónico (1200 MHz)	1.84
Tercer armónico (1800 MHz)	2.48
Cuarto armónico (2400 MHz)	2.28
Quinto armónico (3000 MHz)	2.5

7.3.4. Ejecución y resultados: Barrido de potencia

Los valores que se introdujeron durante la ejecución de esta aplicación vienen indicados en la Tabla 7-2.

Tabla 7–2. Valores introducidos en el barrido de potencia de la primera prueba.

Parámetro	**Valor introducido**
Tipo barrido	1
Conversión de frecuencias	0
Potencia inicial (dBm)	-25
Potencia final (dBm)	12
Incremento de potencia (dBm)	0.1
Frecuencia central (MHz)	600
Ganancia esperada (dB)	15
Número de armónicos	5
Pérdidas a la entrada (dB)	0.9
Pérdidas a la salida (dB)	[0.9, 1.84, 2.48, 2.28, 2.5]

Tras la ejecución del programa se obtiene la gráfica de la Figura 7-4:

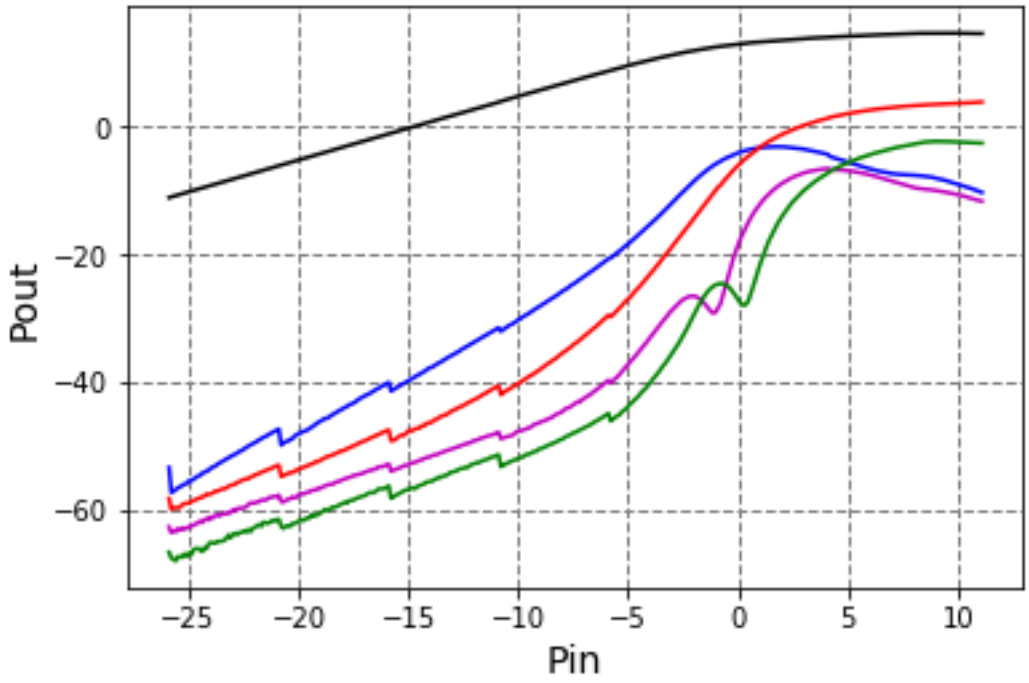


Figura 7-4. Resultado del barrido de potencia de la primera prueba.

Observando la Figura 7-4 se aprecia cómo la componente fundamental queda amplificada a la salida del amplificador, de la manera deseada, además tiene un comportamiento prácticamente lineal hasta un punto de potencia de entrada (aproximandamente $-5\ dBm$). En ese punto el amplificador comienza a saturar y la salida

acaba tomando valores cercanos a $13\ dBm$. Este último valor coincide con lo que se especifica en la hoja de características del amplificador ZJL-6G+ (disponible en el Anexo 1).

El resto de armónicos también aumentan a medida que crece la potencia de entrada, pero tienen potencias de salida mucho menores, como es lógico. Se observa que a partir del punto de saturación del amplificador los armónicos también saturan en potencia o incluso sufren caídas bruscas.

Antes de que la potencia de los armónicos sature a un valor máximo, para el caso del cuarto y quinto armónico hay un punto en el que la potencia cae bruscamente en torno a unos valores de entrada de $-1.5\ dBm$ y $0\ dBm$, respectivamente. Ese comportamiento se debe a que la suma de las componentes de distorsión de los diferentes órdenes incluye términos que se cancelan para este nivel de potencia. A este fenómeno se le conoce con la expresión inglesa: *sweet spot*[8].

Los pequeños escalones en los armónicos que se aprecian al comienzo de las gráficas no son propios del dispositivo bajo prueba, sino que se producen por los procesos de conmutación entre rangos de potencia internos en el equipo, dando lugar a una limitación en la precisión del mismo.

7.3.5. Ejecución y resultados: Barrido de frecuencia

Los valores introducidos en la ejecución de esta aplicación se aprecian en la siguiente tabla:

Tabla 7–3. Valores introducidos en el barrido de frecuencia de la primera prueba.

Parámetro	**Valor introducido**
Tipo barrido	2
Conversión de frecuencias	0
Potencia a la entrada (dBm)	3
Frecuencia inicial (MHz)	100
Frecuencia final (MHz)	600
Incremento de frecuencia (MHz)	10
Ganancia esperada (dB)	15
Número de armónicos	5
Pérdidas a la entrada (dB)	0.9
Pérdidas a la salida (dB)	[0.9, 1.84, 2.48, 2.28, 2.5]

Tras la ejecución del programa se obtiene la gráfica de la Figura 7-5.

[8] Punto justo en el que sucede un evento relevante.

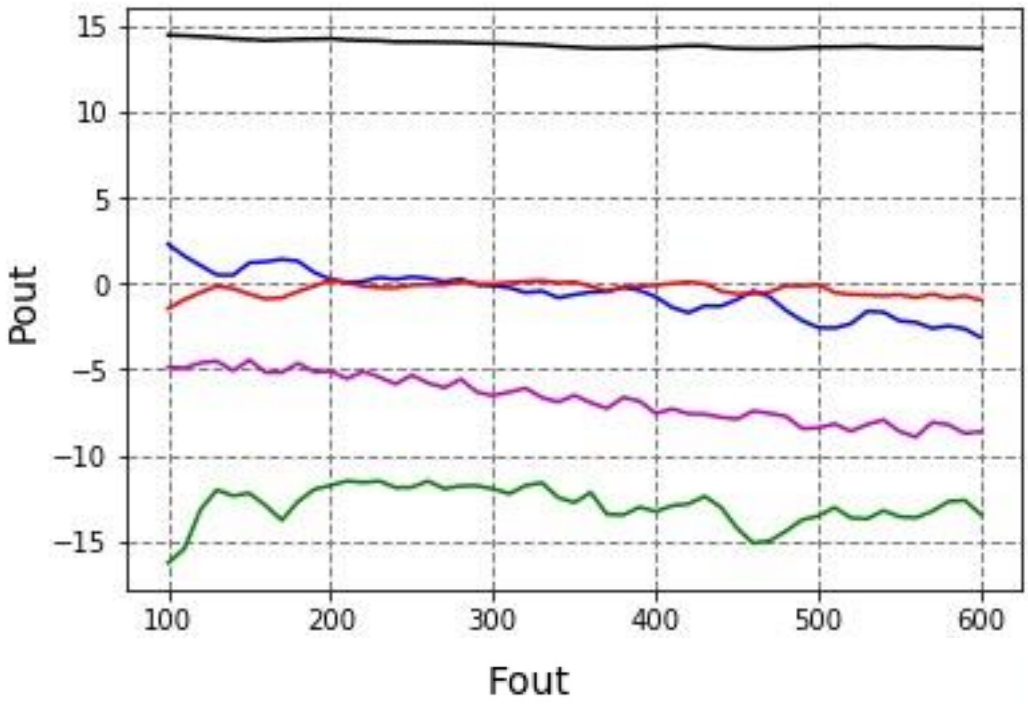


Figura 7-5. Resultado del barrido de frecuencia de la primera prueba.

En la gráfica de la Figura 7-5 se ve cómo la potencia de salida permanece prácticamente constante tanto en la componente fundamental como en los armónicos, mientras que la frecuencia es la que varía entre $100\ MHz - 600\ MHz$. En la componente fundamenal se observa el ligero descenso de la ganancia que presenta el amplificador al aumentar la frecuencia, según se especifica en su hoja de características. El valor que poseen los armónicos es alto debido a que en el ejercicio se trabaja en una zona no lineal.

El punto de operación en potencia a la entrada fue elegido con esas características para buscar un valor cercano a donde se producía el corte entre el segundo armónico (línea azul) y el tercero (línea roja).

7.4. Segunda prueba: Comportamiento de un mezclador

7.4.1. Introducción

El mezclador que se usa en esta aplicación para caracterizar las propiedades no lineales es el modelo MCA1-60+ de Mini-Circuits. Sus principales características son:

- Es un elemento pasivo. Tiene unas pérdidas máximas de $8.3\ dB$.
- Punto de intercepto de tercer orden centrado en $9\ dBm$.
- Frecuencia de trabajo en el rango $1.6\ GHz - 6\ GHz$.
- Aislamiento entre el oscilador local y el puerto de radiofrecuencia de $32\ dB$.
- Aislamiento entre el oscilador local y el puerto de frecuencia intermedia de $17\ dB$.

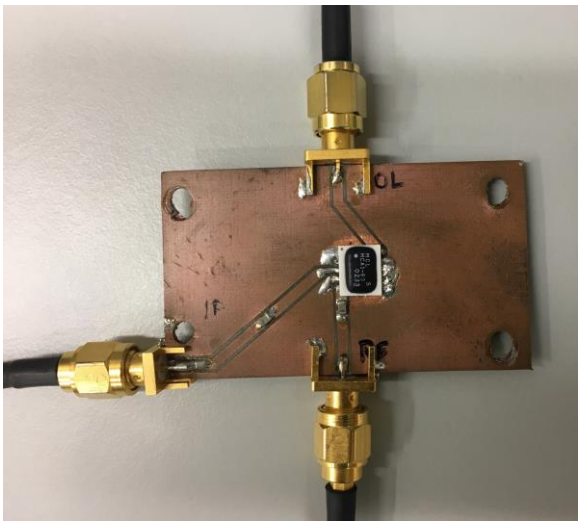


Figura 7-6. Mezclador MCA1-60+.

7.4.2. Montaje

El procedimiento de este ejercicio consiste en introducir una señal por la entrada OL del mezclador desde un generador de señal, otra por la entrada RF desde otro generador de señal y observar como se comporta a su salida mediante el analizador de espectros EXA N9010A.

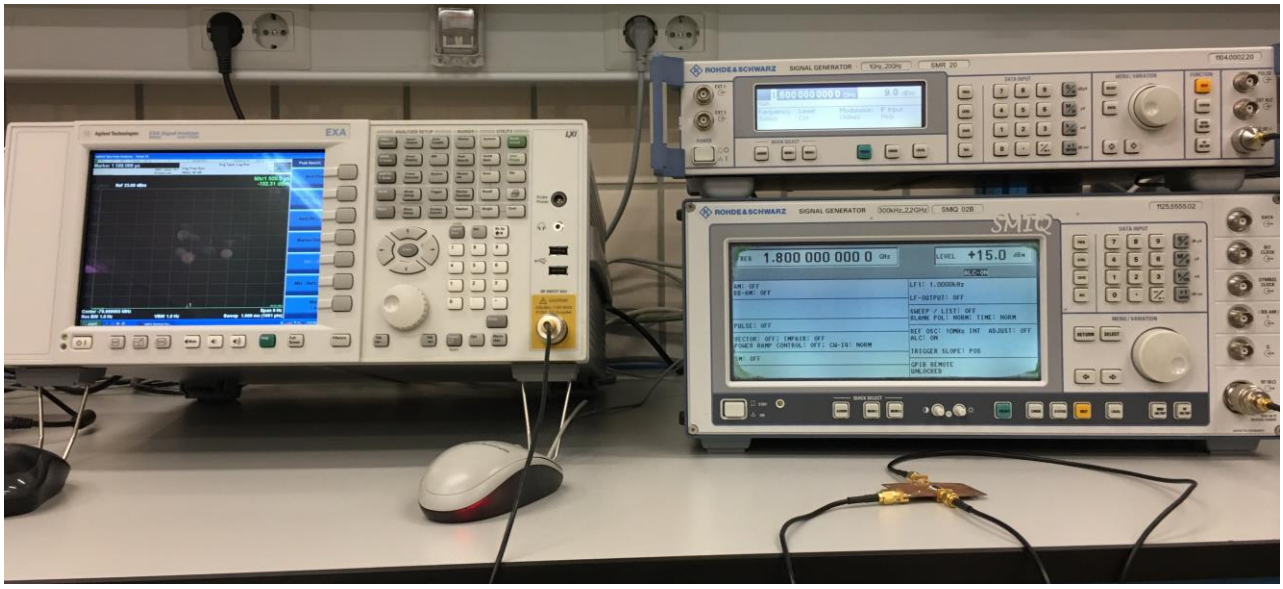


Figura 7-7. Escenario de la prueba 2.

Conexiones presentes en el escenario de la Figura 7-7:

- Ordenador con código en Python con analizador de señal Agilent EXA N9010A. Se realiza mediante una interfaz USB-GPIB Agilent 82357B. El ordenador se puede conectar indistintamente al analizador o al generador, se ha hecho de esta manera en particular por temas de espacio en el laboratorio.
- Analizador de señal Agilent EXA N9010A con generador de señal R&S SMIQ02B. Se conectan mediante bus GPIB estándar.

- Generador de señal R&S SMIQ02B con mezclador MCA1-60+ mediante un cable coaxial con conectores SMA macho en los dos extremos. La salida RF-OUT del generador se conecta con la entrada RF del mezclador, ambos poseen conectores SMA hembra.
- Generador de señal R&S SMR20 con mezclador MCA1-60+ mediante un cable coaxial con conectores SMA macho en los dos extremos. La salida RF-OUT del generador se conecta con la entrada OL del mezclador, ambos poseen conectores SMA hembra. En esta conexión se estableció un valor fijo de potencia en el generador de 9 dBm para que el nivel de entrada al dispositvo fuera de 7 dBm y una frecuencia de oscilador local de 1600 MHz.
- Mezclador MCA1-60+ con analizador de señal Agilent EXA N9010A mediante un cable coaxial con conectores SMA macho en los dos extremos. La salida IF del mezclador se conecta con la entrada RF-INPUT del analizador de espectro, ambos poseen conectores SMA hembra.

7.4.3. Medidas previas

De la misma manera que en el ejercicio anterior, antes de realizar una ejecución del código hay que analizar las pérdidas que se producen debido a los cables y conectores.

Se monta un escenario simple uniendo con un cable coaxial con conectores SMA el analizador y el generador, así se consigue obtener un valor de pérdidas en cada frecuencia a la que posteriormente se va a trabajar. Es importante tener en cuenta para esta tarea las limitaciones de frecuencia de trabajo de los dos generadores de señal disponibles.

Siguiendo el siguiente proceso se obtienen las pérdidas de los cables a la entrada y a la salida en las frecuencias de los cinco primeros armónicos. Los valores de frecuencia y potencia que se indican a continuación son los que se han usado para este ejercicio.

- Generar un tono en el generador de señal con una frecuencia de 2100 MHz y una potencia de 0 dBm.
- Configurar el generador de señal a una frecuencia central de 2100 MHz y con un *span* de 10 MHz para que se pueda apreciar el tono.
- Calcular el valor de las pérdidas de los cables a la entrada con la ecuación que procede.

$$L_{cable} = P_{generador} - P_{analizador}$$

- Generar un tono en el generador de señal con una frecuencia de 500 MHz y una potencia de 0 dBm.
- Configurar el generador de señal a una frecuencia central de 500 MHz y con un *span* de 10 MHz para que se pueda apreciar el tono.
- Calcular el valor de las pérdidas de los cables a la salida, L_{cable}, con la ecuación anterior.
- Repetir los pasos 4, 5 y 6 para las frecuencias en las que se sitúan los armónicos, es decir, para 1000 MHz, 1500 MHz, 2000 MHz y 2500 MHz.

Tras realizar este procedimiento se obtienen los siguientes valores de pérdidas con los que se van a trabajar:

Tabla 7–4. Pérdida de los cables a la entrada en el escenario de la segunda prueba.

Entrada	$\boldsymbol{L_{cable}(dB)}$
2100 MHz	2.28

Tabla 7–5. Pérdidas de los cables a la salida en el escenario de la segunda prueba.

Componente	$\boldsymbol{L_{cable}(dB)}$
Primer armónico (500 MHz)	0.73
Segundo armónico (1000 MHz)	1.11
Tercer armónico (1500 MHz)	1.94
Cuarto armónico (2000 MHz)	2.28
Quinto armónico (2500 MHz)	2.26

7.4.4. Ejecución y resultados: Barrido de potencia

Los valores que se introdujeron durante la ejecución de esta aplicación vienen indicados en la Tabla 7-6.

Tabla 7–6. Valores introducidos en el barrido de potencia de la segunda prueba.

Parámetro	**Valor introducido**
Tipo barrido	1
Conversión de frecuencias	!0
Potencia inicial (dBm)	-17
Potencia final (dBm)	13
Incremento de potencia (dBm)	0.1
Frecuencia central entrada (MHz)	2100
Frecuencia central salida (MHz)	500
Ganancia esperada (dB)	0
Número de armónicos	5
Pérdidas a la entrada (dB)	2.28
Pérdidas a la salida (dB)	[0.73, 1.11, 1.94, 2.28, 2.26]

Tras la ejecución del programa se obtiene la gráfica de la siguiente figura:

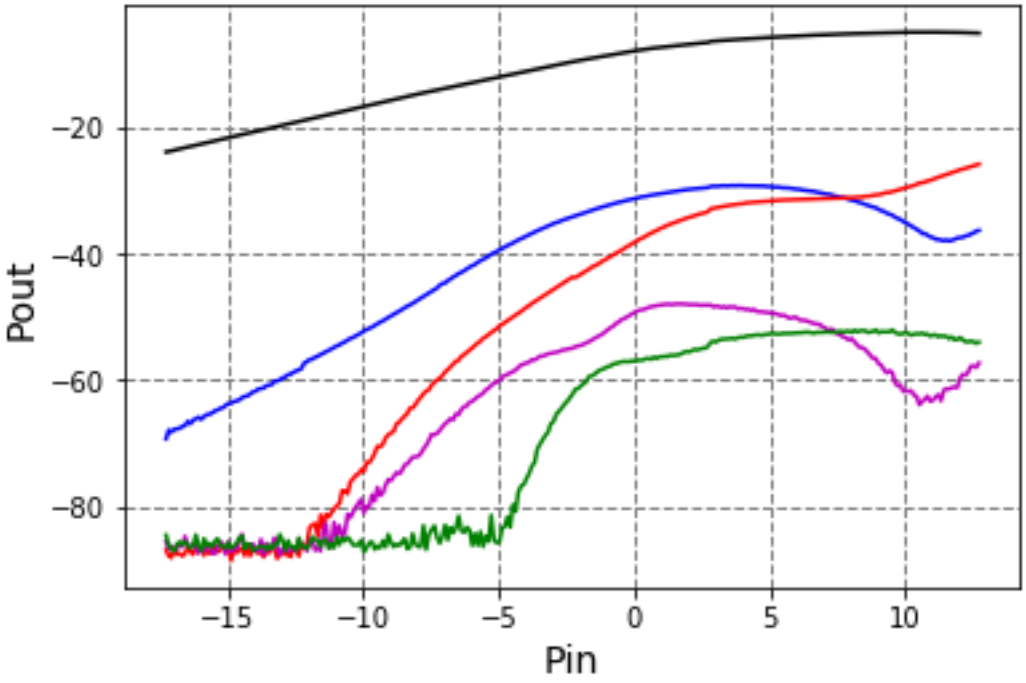


Figura 7-8. Resultado del barrido de potencia de la segunda prueba.

Se observa en la Figura 7-8 que la componente fundamental a la salida presenta las pérdidas de conversión esperadas de unos 6 dB y tiene un comportamiento prácticamente lineal hasta un punto de potencia de entrada (aproximandamente 0 dBm), a partir de la cual satura.

Al comienzo de la representación de los armónicos tercero, cuarto y quinto se aprecia un fondo de ruido, que ha sido generado por el equipo. Realmente, el valor de estos armónicos en esos puntos es inferior al del ruido, debido a ello es por lo que no se ve en la gráfica.

Como curiosidad se añade que la forma de la traza del segundo armónico es similar a la del cuarto, y a su vez, la del tercero lo es con el quinto ya que los armónicos pares se asemejan entre sí, al igual que los impares.

7.4.5. Ejecución y resultados: Barrido de frecuencia

Los valores introducidos en la ejecución de esta aplicación se aprecian en la siguiente tabla:

Tabla 7–7. Valores introducidos en el barrido de frecuencia de la segunda prueba.

Parámetro	**Valor introducido**
Tipo barrido	2
Conversión de frecuencias	!0
Potencia a la entrada (dBm)	15
Frecuencia inicial (MHz)	1800
Frecuencia final (MHz)	2200
Incremento de frecuencia (MHz)	20
Frecuencia del OL (MHz)	1600
Down-converter	!0
Ganancia esperada (dB)	0
Número de armónicos	5
Pérdidas a la entrada (dB)	2.28
Pérdidas a la salida (dB)	[0.73, 1.11, 1.94, 2.28, 2.26]

Tras la ejecución del programa se obtiene la gráfica de la Figura 7-9.

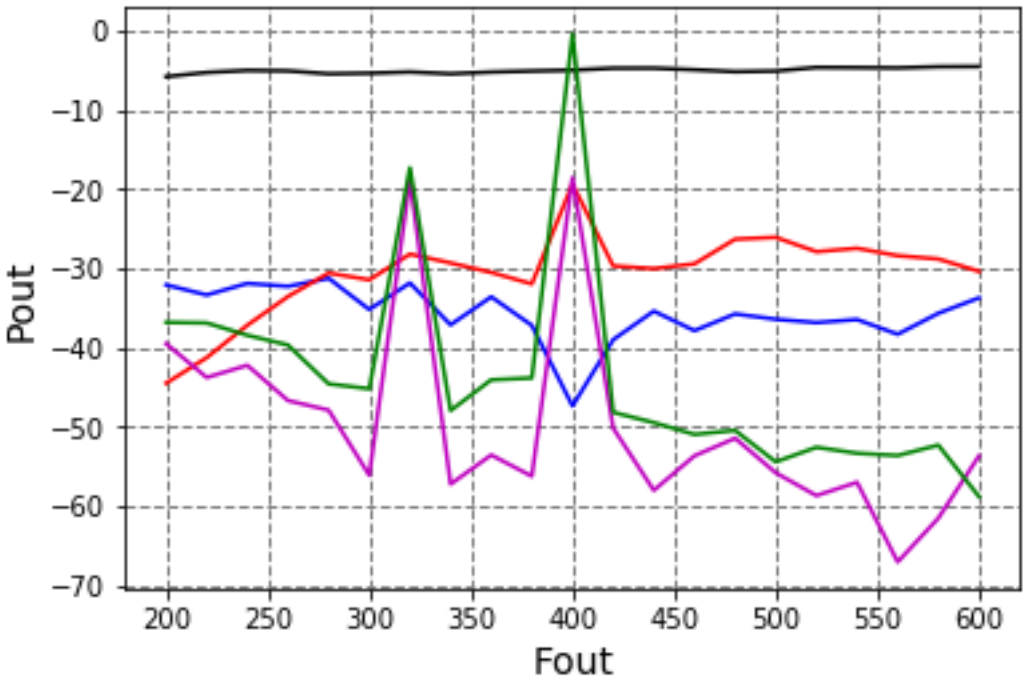


Figura 7-9. Resultado del barrido de frecuencia de la segunda prueba.

De la Figura 7-9 lo más significativo que se puede extraer es que, a $400\ MHz$, en el segundo, tercer, cuarto y quinto armónico de la frecuencia intermedia de salida hay alguna suma de componentes errónea por la cual se generan unos picos muy abruptos. En la siguiente tabla se muestra la fluctuación de potencia que se obtuvo en el laboratorio en dichos armónicos:

Tabla 7–8. Valores de fluctuación por los que se obtiene un pico abrupto.

Armónico	**Valor máximo (dBm)**	**Valor mínimo (dBm)**
Segundo	-30	-64
Tercero	-19	-25.61
Cuarto	-20.73	-25.66
Quinto	-2.68	-7.23

Tras el análisis de los valores de la Tabla 7-8 se halló que el principal problema se encontraba en las frecuencias del cuarto y quinto armónico. En la frecuencia del cuarto se representa la suma del residuo del oscilador local y el propio cuarto armónico. Mientras que en la frecuencia del quinto se dibuja la suma del residuo de radiofrecuencia (*RF leakage*[9]) con el propio quinto armónico. Por lo tanto, se producen unos errores que no dejan apreciar el resultado con claridad, provocando los picos abruptos mencionados en el párrafo anterior.

[9] Fuga de residuos en radiofrecuencia.

8 CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

A lo largo de este proyecto se demuestra que el lenguaje de programación Python ofrece una herramienta de trabajo muy eficiente para llevar a cabo tareas de control remoto con equipamiento de radiocomunicación. Como ya se advirtió en el capítulo de resumen, se ha diseñado un código que facilita la realización de medidas con el fin de caracterizar dispositivos no lineales de radiofrecuencia. Con su ayuda, el usuario final no tiene que interactuar directamente con el instrumento, pudiendo configurar los parámetros necesarios a través de la consola de Spyder (herramienta utilizada para programar en Python).

El aspecto más destacado de hacer tareas de control remoto con Python es el tiempo que ahorra en realizar medidas que necesitan una gran cantidad de configuraciones en los equipos. Como ejemplo de ello se toma la primera prueba de este proyecto, en la que se hace un barrido de potencia desde $-25\ dBm$ hasta $12\ dBm$ con saltos de $0.1\ dBm$. Para obtener su resultado hacen falta ejecutar 370 bloques de configuraciones, que de no usar el programa diseñado habría que realizar manualmente. Además, este método en comparación con otras formas de control remoto más habituales, como puede ser utilizando Matlab o LabVIEW, realiza las mediciones en los equipos en un tiempo sensiblemente más reducido.

Teniendo en cuenta los contenidos y la finalidad de este proyecto se pueden afrontar nuevos objetivos para explotar con más profundidad esta técnica de control remoto. Un ejemplo de ello puede ser realizar un programa que se encargue del estudio de una medida ACPR, brevemente definida en el apartado 5.2.6 de esta memoria. Otra línea de trabajo posible es la caracterización de la predistorsión u otras técnicas de linealización para el diseño de amplificadores, algo que sería de gran utilidad en los sistemas de radiocomunicación actuales.

REFERENCIAS

[1] Ingenieriaelectronica.org. Introducción de instrumentación virtual. [*online*] Disponible en: https://ingenieriaelectronica.org/instrumentacion-virtual-hardwares-y-beneficios/

[2] Emb.cl. Evolución de la tecnología. [*online*] Disponible en: http://www.emb.cl/electroindustria/articulo.mvc?xid=471

[3] Ocw.ehu.eus. Comparación entre instrumentación virtual y tradicional. [*online*] Disponible en: https://ocw.ehu.eus/file.php/54/GENERAL/Comparacion_IV_IT.pdf

[4] Monografias.com. Instrumentos virtuales. [*online*] Disponible en: http://www.monografias.com/trabajos38/instrumentacion-virtual-industrial/instrumentacion-virtual-industrial2.shtml#calibrac

[5] PyVISA. Estándar VISA. [*online*] Disponible en: https://pyvisa.readthedocs.io/en/stable/

[6] Wikipedia.org. Estándar VISA. [*online*] Disponible en: https://en.wikipedia.org/wiki/Virtual_Instrument_Software_Architecture

[7] Academia.edu. Bus GPIB. [*online*] Disponible en: http://www.academia.edu/6737142/BUS_GPIB

[8] Prezi.com. Bus GPIB. [*online*] Disponible en: https://prezi.com/pwdfft63mxp_/bus-de-instrumentacion-de-proposito-general-gpib-ieee-488/

[9] Magaly Sierra Vite, «Aplicaciones didácticas del GPIB a nivel licenciatura y postgrado para la UAEH», Tesis Doctoral, Marzo 2007.

[10] Instrumentación electrónica, Práctica 3, UPC, «Control de instrumentos a bajo nivel. Peticiones de servicio», Versión 1.0, Septiembre 1998.

[11] Juan José González de la Rosa, «Instrumentos electrónicos programables», UCA.

[12] National Instruments. Red de Área Local. [*online*] Disponible en: http://www.ni.com/white-paper/2922/en/

[13] National Instruments. Red de Área Local. [*online*] Disponible en: http://www.ni.com/white-paper/3518/en/#toc2

[14] María España Borrero Serrano, «Herramienta software para el control remoto de una fuente de alimentación mediante una interfaz gráfica.», Trabajo Fin de Grado, Capítulos 1 y 2, Julio 2011.

[15] National Instruments. PXI. [*online*] Disponible en: http://www.ni.com/pxi/whatis/esa/

[16] PyVISA. Python. [*online*] Disponible en: https://pypi.python.org/pypi/PyVISA

[17] Guido Van Rossum, Fred L. Drake, Jr., «Python», Septiembre 2009.

[18] Pythonmania.net. Python. [*online*] Disponible en: https://www.pythonmania.net/es/

[19] Gaël Varoquaux, Emmanuelle Gouillart, Olaf Vahtras, «*Scipy Lecture Notes*», 2015.

[20] Kang-Jing Huang, «*Introduction to Instrument Control Using Python*».

[21] Daniel Contreras Solórzano, «Plataforma experimental para la evaluación de modelos de comportamiento en amplificadores de potencia.», Proyecto fin de grado, Capítulo 2, Material y Método, Julio 2011.

[22] Agilent Technologies, «*Agilent X-Series signal analyzer*», Manual del analizador de señal.

[23] Rohde & Schwarz, «*Vector signal generator*», Manual del generador de señal, Volumen 2, Capítulo 3.

[24] Ignacio Pérez Lupiáñez, «Plataforma para la caracterización experimental de dispositivos de RF.», Trabajo Fin de Grado, Capítulos 5 y 6, 2016.

[25] Michel Allegue Martínez, «*Modeling and Compensation of Non-linear Effects in Wireless Communications Systems*», Tesis Doctoral, Capítulo 3, 2012.

[26] Minicircuits. Amplificador ZJL-6G+. [online] Disponible en: https://ww2.minicircuits.com/pdfs/ZJL-6G+.pdf

[27] Minicircuits. Mezclador MCA1-60+. [online] Disponible en: https://ww2.minicircuits.com/pdfs/MCA1-60+.pdf

GLOSARIO

ACPR: relación de potencia de canal adyacente (*Adjacent Channel Power Ratio*) 49, 77

ADC: convertidor analógico-digital (*Analog-to-Digital Converter*) 45

ANSI: instituto de estandarización americano (*American National Standards Institute*) 8

API: interfaz de programación de aplicaciones (*Application Programming Interface*) 6

ASCII: código estándar estadounidense para el intercambio de la información (*American Standard Code for Information Interchange*) 12

ATN: línea de atención (*Attention*) 12, 13, 18

BJT: transistor de unión bipolar (*Bipolar Junction Transistor*) 47

CLS: -comando común de GPIB - (*Clear Status Command*) 16, 19

CNRI: corporación para iniciativas nacionales de inversigación (*Corporation for National Research Initiatives*) 25

CPU: unidad central de procesamiento (*Central Processing Unit*) 23

DAV: dato válido (*Data Valid*) 10, 12, 20, 21

DCL: -comando universal de GPIB - (*Device Clear*) 19

DIO: línea de dato entrada/salida (*Data IO*) 12, 20, 21

DLL: biblioteca de enlace dinámico (*Dinamic Link Library*) 6

EOI: fin de identidad (*End Or Identity*) 13

ESE: -comando común de GPIB - (*Event Status Enable Command*) 19

ESE?: -comando común de GPIB - (*Event Status Enable Query*) 19

ESER: registro de habilitación de estados (*Event Status Enable Register*) 17

ESR: registro de evento de estado (*Event Status Register*) 17

ESR?: -comando común de GPIB - (*Event Status Register Query*) 19

FET: transistor de efecto de campo (*Field-Effect Transistor*) 47

FFT: transformada rápida de Fourier (*Fast Fourier Transform*) 45, 46

FM: frecuencia modulada (*Modulated Frecuency*) 47

GET: -comando *addressed* de GPIB - (*Group Trigger*) 18

GPIB: bus de datos de propósito general (*General-Purpose Instrumentation Bus*) 1, 5, 6, 7, 8, 10, 11, 13, 14, 16, 18, 21, 24, 42, 45, 46, 65, 70, 89, 90

GSM: sistema global para comunicaciones móviles (*Global System for Mobile communications*) 46

GTL: -comando *addressed* de GPIB- (*Go To Local*) 18

GUI: interfaz gráfica de usuario (*Graphical User Interface*) 42

HP: *Hewlett Packard* 7

HP-IB: bus de datos de *Hewlett Packard* (del inglés *Hewlett Packard Instrument Bus*) 7

HTML: lenguaje de marcados para hipertextos (*HyperText Markup Language*) 23

I/O: entrada/salida (*In/Out*) 10

IDE: entorno de desarrollo integrado (*Integrated Development Environment*) 41

IDN?: -comando común de GPIB - (*Identification Query*) 19, 42, 52

IEC: comisión electrónica internacional (*International Electrotechnical Commission*) 8

IEEE: instituto de ingenieros eléctricos y electrónicos (*Institute of Electrical and Electronics Engineers*) 7, 8, 10, 14, 15, 17, 20, 21, 23, 26

IFC: reseteo de interfaz (*Interface Clear*) 13

IP: protocolo de internet (*Internet Protocol*) 21, 22, 23

ISO: organización internacional de estandarización (*International Organization for Standardization*) 12

IT: tecnología de la información (*Information Technology*) 23

LabVIEW: software de ingeniería en sistemas (*Laboratory Virtual Instrument Engineering Workbench*) 3

LAN: red de área local (*Local Area Network*) 10, 21, 22, 23, 24

LLO: -comando universal de GPIB - (*Local Lockout*) 19

LRN?: -comando común de GPIB - (*Learn Device Setup Query*) 19

LTE: evolución a largo plazo (*Long Term Evolution*) 46

LXI: extensión de red de área local para instrumentación (*LAN eXtensions for Instrumentation*) 23

MLA: -comando *talk/listen* de GPIB - (*My Listen Address*) 19

MSA: -comando *addressed* de GPIB - (*My Secondary Address*) 18

MTA: -comando *talk/listen* de GPIB - (*My Talk Address*) 19

NDAC: dato no aceptado (*Not Data Accepted*) 12, 20, 21

NRFD: dato no preparado (*Not Ready For Data*) 11, 12, 20, 21

OPC: -comando común de GPIB - (*Operation Complete Command*) 19

OPC?: -comando común de GPIB - (*Operation Complete Query*) 19

OPT?: -comando común de GPIB - (*Option Identification Query*) 19

PC: ordenador personal (*Personal Computer*) 3, 6, 21, 24

PCI: interconexión de componentes periféricos (*Peripheral Component Interconnect*) 23

PPC: -comando *addressed* de GPIB - (*Parallel Poll Configure*) 18

PPU: -comando universal de GPIB - (*Parallel Poll Unconfigure*) 19

RCL: -comando común de GPIB - (*Recall Command*) 19

REN: habilitación remota (*Remote Enable*) 13

RS232: estándar recomendado 232 (*Recommended Standard 232*) 6, 42

RST: -comando común de GPIB - (*Reset Command*) 19

SAV: -comando común de GPIB - (*Save Command*) 19

SBR: registro de estado de bytes (*Status Byte Register*) 17

SCPI: comandos estándar para instrumentos programables (*Standard Commands for Programmable Instrumentation*) 10, 15, 20, 52, 53, 58

SDC: -comando *addressed* de GPIB- (*Select Device Clear*) 18

SMA: tipo de conector roscado para cable coaxial (*SubMiniature version A*) 65, 71

SPD: -comando universal de GPIB - (*Serial Poll Disable*) 19

SPE: -comando universal de GPIB - (*Serial Poll Enable*) 19

SQR: petición de servicio (*Service Request*) 13, 17

ANEXO 1

Se corresponde con el *data sheet*[10] del amplificador empleado en el estudio realizado en el apartado 7.3:

Coaxial

Amplifier

ZJL-6G+

50Ω Low Power 20 to 6000 MHz

CASE STYLE: BW459

Connectors	Model	Price	Qty.
SMA	ZJL-6G+	$114.95 ea.	(1-9)

+ RoHS compliant in accordance with EU Directive (2002/95/EC)

The +Suffix has been added in order to identify RoHS Compliance. See our web site for RoHS Compliance methodologies and qualifications.

Features

- ultra wideband, 20 to 6000 MHz
- compact rugged case, 1.07"x 0.61" (including mounting bracket)
- protected by US Patent, 6,943,629

Applications

- communications systems
- radar
- instrumentation
- laboratory use

Amplifier Electrical Specifications

MODEL NO.	FREQUENCY (MHz)		GAIN (dB)			MAXIMUM POWER (dBm)			DYNAMIC RANGE		VSWR (:1) Typ.		DC POWER	
						Output (1 dB Compr.)			NF (dB)	IP3 (dBm)			Volt (V)	Current (mA)
	f_L	f_U	Typ.	Min.	Flatness[1] Typ.	L	U	Input (no damage)	Typ.	Typ.	In	Out	Nom.	Max.
ZJL-6G+	20	6000	13	10	±1.6	+9	+10	+15	4.5	+24	1.5	1.4	12	50

1. Flatness specified to 0.75 fU, dynamic range at 2 GHz.

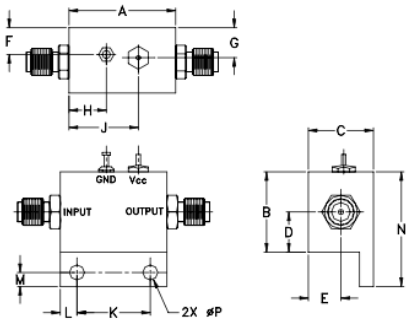
Open load is not recommended, potentially can cause damage.
With no load derate max input power by 20 dB

L= low range (f_L to $f_U/2$) U= upper range ($f_U/2$ to f_U)

Maximum Ratings

Operating Temperature	-40°C to 75°C
Storage Temperature	-55°C to 100°C
DC Voltage	+13V Max.

Outline Drawing



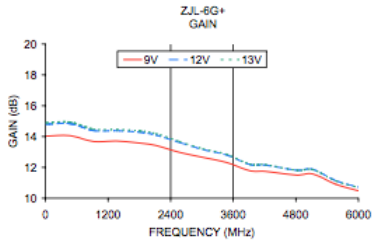
Outline Dimensions (inch / mm)

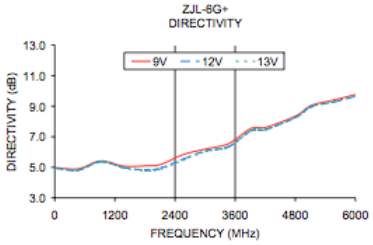
A	B	C	D	E	F	G	H	J	K	L	M	N	P	wt
1.00	.75	.61	.38	.29	.25	.26	.35	.65	.688	.156	.13	1.07	.140	grams
25.40	19.05	15.49	9.65	7.37	6.35	6.60	8.89	16.51	17.48	3.96	3.30	27.18	3.56	25

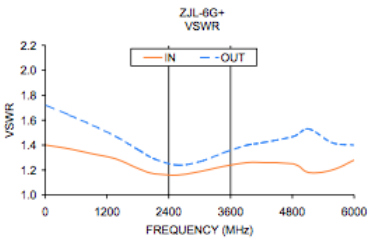
[10] Hoja de características.

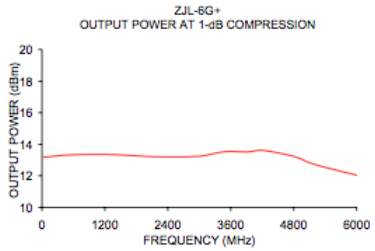
Typical Performance Data/Curves ZJL-6G+

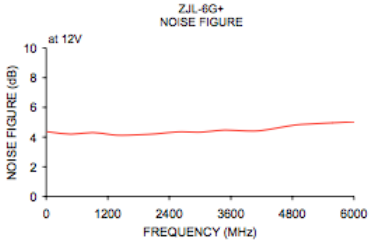
FREQUENCY (MHz)	GAIN (dB)			DIRECTIVITY (dB)			VSWR (:1)		NOISE FIGURE (dB)	POUT at 1 dB COMPR. (dBm)
	9V	12V	13V	9V	12V	13V	IN	OUT	12V	12V
20.00	14.03	14.80	14.90	4.98	4.94	4.94	1.40	1.72	4.36	13.17
470.00	14.06	14.84	14.94	4.90	4.82	4.83	1.37	1.64	4.20	13.31
920.00	13.68	14.39	14.48	5.42	5.37	5.39	1.33	1.56	4.30	13.35
1370.00	13.71	14.38	14.46	5.08	4.99	4.98	1.29	1.47	4.13	13.34
1820.00	13.57	14.28	14.37	5.10	4.83	4.78	1.21	1.36	4.16	13.26
2120.00	13.42	14.10	14.19	5.19	4.96	4.89	1.17	1.29	4.21	13.21
2570.00	12.98	13.62	13.69	5.84	5.55	5.48	1.16	1.24	4.35	13.20
3020.00	12.65	13.16	13.21	6.20	6.09	6.05	1.19	1.27	4.33	13.24
3470.00	12.32	12.81	12.86	6.55	6.36	6.33	1.23	1.34	4.47	13.52
3920.00	11.78	12.19	12.21	7.55	7.40	7.40	1.26	1.40	4.41	13.51
4220.00	11.74	12.15	12.18	7.62	7.49	7.46	1.26	1.42	4.45	13.61
4820.00	11.50	11.82	11.81	8.38	8.29	8.30	1.25	1.47	4.80	13.21
5120.00	11.58	11.88	11.86	9.05	8.98	8.95	1.18	1.53	4.88	12.80
5570.00	10.90	11.14	11.12	9.40	9.27	9.30	1.20	1.42	4.96	12.39
6000.00	10.48	10.71	10.69	9.76	9.64	9.67	1.28	1.40	5.03	12.04











ANEXO 2

Se corresponde con el *data sheet* del mezclador empleado en el estudio realizado en el apartado 7.4:

Ceramic

Frequency Mixer WIDE BAND

Level 7 (LO Power+7 dBm) 1600 to 6000 MHz

MCA1-60+

CASE STYLE: DZ885

+RoHS Compliant
The +Suffix identifies RoHS Compliance. See our web site for RoHS Compliance methodologies and qualifications

Available Tape and Reel at no extra cost

Reel Size	Devices/Reel
7"	10, 20, 50, 100, 200
13"	500,1000

Recommended Replacement:
MAC-60+
- *Footprint Compatible*
- *MIL Level Reliability*

Click here for data sheet

Maximum Ratings

Operating Temperature	-55°C to 100°C
Storage Temperature	-55°C to 100°C
RF Power	50 mW
IF Current	40 mA

Permanent damage may occur if any of these limits are exceeded.

Pin Connections

LO	10
RF	5
IF	3
GROUND	1,2,4,6,7,8,9

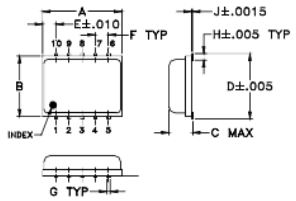
Features

- wide bandwidth, 1600 to 6000 MHz
- useable to 8000 MHz
- IF, DC to 2000 MHz
- LTCC double balanced mixer
- aqueous washable
- low cost
- low profile, 0.08"
- protected by US Patent 7,027,795

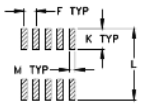
Applications

- PCN
- defense & weather radar
- WCDMA
- defense communications

Outline Drawing



PCB Land Pattern



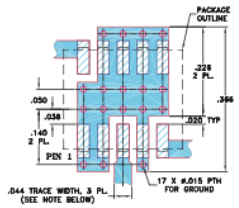
Suggested Layout,
Tolerance to be within ±.002

Outline Dimensions (inch / mm)

A	B	C	D	E	F	G
.30	.250	.085	.266	.050	.050	.012
7.62	6.35	2.16	6.76	1.27	1.27	0.30

H	J	K	L	M	wt
.029	.004	.085	.296	.030	grams
0.74	0.10	2.16	7.52	0.76	0.25

Demo Board MCL P/N: TB-144
Suggested PCB Layout (PL-045)



NOTES: 1. TRACE WIDTH IS SHOWN FOR ROGERS RO4350B WITH DIELECTRIC THICKNESS .020" ± .0015"; COPPER: 1/2 OZ. EACH SIDE. FOR OTHER MATERIALS TRACE WIDTH MAY NEED TO BE MODIFIED.
2. BOTTOM SIDE OF THE PCB IS CONTINUOUS GROUND PLANE.
DENOTES PCB COPPER LAYOUT WITH SMOBC (SOLDER MASK OVER BARE COPPER)
DENOTES COPPER LAND PATTERN FREE OF SOLDER MASK

Electrical Specifications (T_{AMB}=-55°C to 100°C)

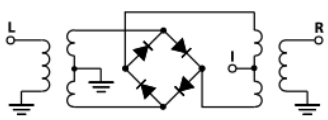
FREQUENCY (MHz)		CONVERSION LOSS (dB)			LO-RF ISOLATION (dB)		LO-IF ISOLATION (dB)		IP3 at center band (dBm)
LO/RF f_L-f_U	IF	$\overline{X}$	σ	Max.	Typ.	Min.	Typ.	Min.	Typ.
1600-4400	DC-2000	6.3	0.2	8.3*	32	20	17	—	9
4400-6000	DC-2000	6.2	0.3	8.5*	23	17	18	—	8

1 dB COMPR. +1 dBm typ.
*Conversion loss at 30 MHz IF, increases with IF frequency. See Graphs

Typical Performance Data

Frequency (MHz)		Conversion Loss (dB)	Isolation L-R (dB)	Isolation L-I (dB)	VSWR RF Port (:1)	VSWR LO Port (:1)
RF	LO	LO +7dBm	LO +7dBm	LO +7dBm	LO +7dBm	LO +7dBm
1600.00	1630.00	5.85	33.44	18.92	3.21	2.77
1800.00	1830.00	5.57	43.23	20.13	2.72	3.06
2000.00	2030.00	5.67	35.50	19.59	1.52	3.18
2300.00	2330.00	5.57	30.85	17.13	2.55	2.32
2500.00	2530.00	5.67	31.53	16.64	3.70	1.12
3000.00	3030.00	6.65	33.86	17.37	6.40	4.50
3300.00	3330.00	6.51	33.27	17.79	2.35	4.42
3500.00	3530.00	6.26	32.26	16.02	2.17	2.74
3800.00	3830.00	5.85	33.61	14.80	2.21	4.29
4000.00	4030.00	6.13	34.90	16.01	2.03	3.49
4300.00	4330.00	6.05	33.36	17.95	1.97	3.60
4500.00	4530.00	5.95	29.83	18.64	2.31	2.60
4700.00	4730.00	6.02	28.69	19.14	1.78	2.51
4900.00	4930.00	6.11	25.78	19.89	1.91	2.02
5000.00	5030.00	5.99	24.51	20.40	1.50	1.54
5200.00	5230.00	5.88	23.40	21.37	1.53	1.71
5300.00	5330.00	6.06	22.68	21.68	2.25	2.45
5500.00	5530.00	6.00	22.63	19.20	3.70	3.96
5700.00	5730.00	6.14	23.13	15.10	5.34	6.43
6000.00	6030.00	6.61	24.37	13.33	1.73	2.81

Electrical Schematic



Notes
A. Performance and quality attributes and conditions not expressly stated in this specification document are intended to be excluded and do not form a part of this specification document.
B. Electrical specifications and performance data contained in this specification document are based on Mini-Circuit's applicable established test performance criteria and measurement instructions.
C. The parts covered by this specification document are subject to Mini-Circuits standard limited warranty and terms and conditions (collectively, "Standard Terms"); Purchasers of this part are entitled to the rights and benefits contained therein. For a full statement of the Standard Terms and the exclusive rights and remedies thereunder, please visit Mini-Circuits' website at www.minicircuits.com/MCLStore/terms.jsp

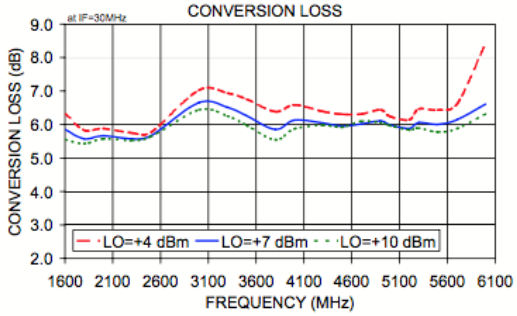
Mini-Circuits®
www.minicircuits.com P.O. Box 350166, Brooklyn, NY 11235-0003 (718) 934-4500 sales@minicircuits.com

REV. F
M151107
ED-10107/5
MCA1-60+
DJ/RS/CP/AM
160727

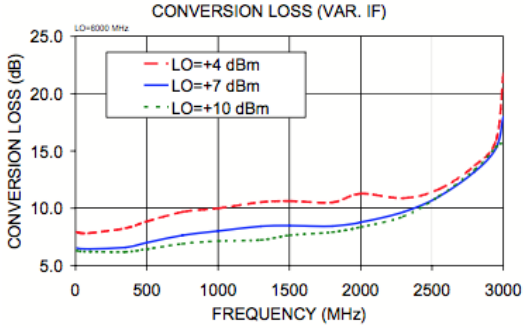
Performance Charts

MCA1-60+

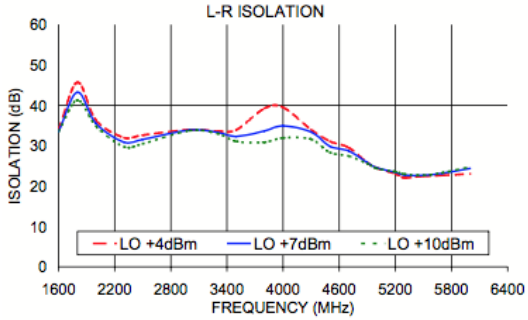
MCA1-60+
CONVERSION LOSS



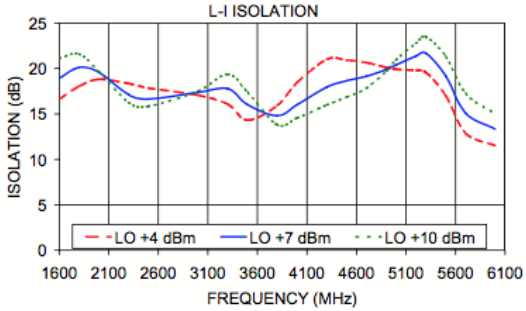
MCA1-60+
CONVERSION LOSS (VAR. IF)



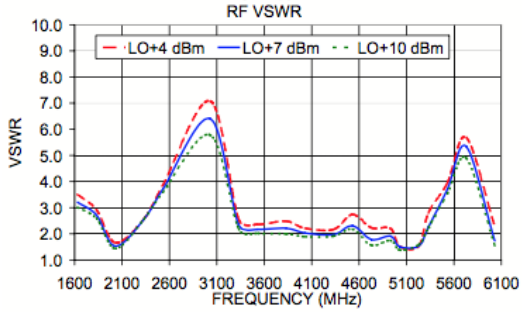
MCA1-60+
L-R ISOLATION



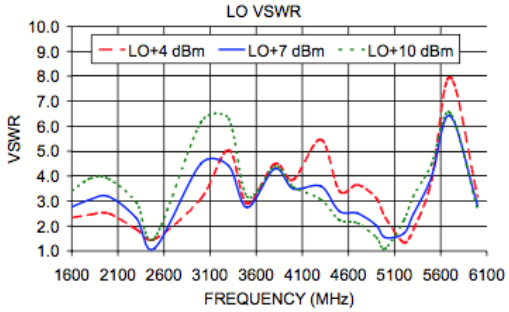
MCA1-60+
L-I ISOLATION



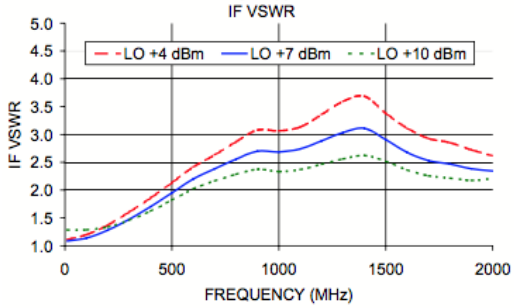
MCA1-60+
RF VSWR



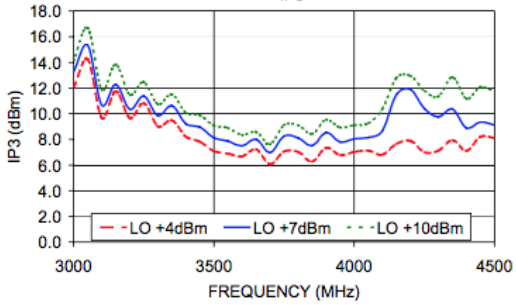
MCA1-60+
LO VSWR



MCA1-60+
IF VSWR



MCA1-60+
IP3



Notes
A. Performance and quality attributes and conditions not expressly stated in this specification document are intended to be excluded and do not form a part of this specification document.
B. Electrical specifications and performance data contained in this specification document are based on Mini-Circuit's applicable established test performance criteria and measurement instructions.
C. The parts covered by this specification document are subject to Mini-Circuits standard limited warranty and terms and conditions (collectively, "Standard Terms"); Purchasers of this part are entitled to the rights and benefits contained therein. For a full statement of the Standard Terms and the exclusive rights and remedies thereunder, please visit Mini-Circuits' website at www.minicircuits.com/MCLStore/terms.jsp

ANEXO 3

En este anexo se incluye el código completo realizado divido en las diferentes funciones:

***SMIQ_connect.py*:** En caso de que todos los parámetros sean correctos, llama a la función *build_GPIB_object.py* para crear el objeto a utilizar en la comunicación mediante el bus GPIB.

```
def SMIQ_connect (varargin):
    numargu=len(locals())
    import build_GPIB_object
    from six import string_types
    global Status
    Status = 0

    #Valores por defecto
    class primer():
        def __init__(self, protocol, **GPIB):
            self.protocol=protocol
            self.__dict__.update(GPIB)

    class segundo():
        def __init__(self, board, primaddr):
            self.board=board
            self.primaddr=primaddr

    ParamStruct=primer(protocol='GPIB', GPIB=segundo(0, 28))

    #Comprobar que el número de argumentos sea válido
    if numargu < 1:
        InstrObj = build_GPIB_object.build_GPIB_object(ParamStruct)

    #Comprobar que el primer argumento sea una cadena de caracteres
    if isinstance(varargin[0], string_types)==False:
        raise ("First Parameter must be a string defining the protocol type ''GPIB''")

    #Seleccionar el protocolo
    if varargin[0] == 'GPIB':
        ParamStruct.protocol = varargin[0]
    else:
        raise ("Protocol type must either be ''GPIB''")

    #Configurar la dirección del GPIB o construir el objeto TCPIP con la dirección y el nombre dado
    if numargu > 1:
        if ParamStruct.protocol == 'GPIB':
            ParamStruct.GPIB.board = int(varargin[1])

    if numargu > 2:
        ParamStruct.GPIB.primaddr = int(varargin[2])

    if numargu > 3:
        ParamStruct.GPIB.secaddr = int(varargin[3])

    #Crear el objeto con el que se va a mantener la comunicación mediante "build_GPIB_object.py"
    InstrObj = build_GPIB_object.build_GPIB_object(ParamStruct)

    return Status, InstrObj
```

build_GPIB_object.py: Crea el objeto para la comunicación mediante el bus GPIB.

```
def build_GPIB_object (ParamStruct):
    try:
        #Comprobar si existe el método secaddr en ParamStruct.GPIB
        if hasattr(ParamStruct.GPIB, 'secaddr'):
            import visa
            rm=visa.ResourceManager()
            InstrObj=rm.open_resource("%s%i::%i::%s") %(ParamStruct.protocol, ParamStruct.GPIB.board, ParamStruct.GPIB.primaddr, ParamStruct.GPIB.secaddr)
            InstrObj.timeout=10000
        #Se crea el objeto con el analizador de espectro, ya que para crear uno para el generador de señal se utiliza la función "un_tono_SMIQ.py"
        else:
            import visa
            rm=visa.ResourceManager()
            InstrObj=rm.open_resource('GPIB0::18::INSTR')
            InstrObj.timeout=10000
    except:
        print ("Object creation failed.")

    InstrObj.chunk_size=10000000

    global Status
    Status = 1

    return InstrObj
```

send_command.py: Escribe el comando pasado como argumento en el objeto que se le indica (generador vectorial de señales o analizador de señal).

```
def send_command (InstrObj, strCommand):
    numargu=len(locals())
    from six import string_types
    Status = 0

    #Comprobar que el número de argumentos sea válido
    if numargu != 2:
        raise ("Wrong number of input arguments.")

    #Comprobar que la cadena "strCommand" sea correcta
    if strCommand == "" or isinstance(strCommand, string_types)==False:
        raise ("Command string is empty or not a string.")

    #Escribir el comando SCPI en el objeto creado para la comunicación
    try:
        InstrObj.write(strCommand)
    except:
        del(InstrObj)
        print ("Could not open connection or write command. Recheck the connection settings.")
        print ("Maybe there is already an open connection to the device.")

    Status=1

    return Status
```

send_query.py: Escribe el comando pasado como argumento en el objeto que se le indica y recoge el resultado generado por el mismo (generador vectorial de señales o analizador de señal).

```
def send_query (InstrObj, strCommand):
    numargu=len(locals())
    from six import string_types
    Status = 0
    Result = 0

    #Comprobar que el número de argumentos sea válido
    if numargu != 2:
        raise ("Wrong number of input arguments.")

    #Comprobar que la cadena "strCommand" sea correcta
    if not strCommand != "" or isinstance(strCommand, string_types)==False:
        raise ("Command string is empty or not a string.")

    #Comprobar si la interrogación está presente en el mensaje
    if strCommand.find("?") == -1:
        raise ("Query strings must contain a question mark.")

    #Escribir el comando SCPI en el objeto creado para la comunicación
    #Posteriormente se lee la respuesta y se guarda en la variable "Result"
    InstrObj.write(strCommand)
    Result=InstrObj.read_raw()

    Status=1

    return Status, Result
```

un_tono_SMIQ.py: Crea el objeto con el generador de señales y le manda las órdenes requeridas por teclado relacionadas con el mismo.

```
def un_tono_SMIQ (potencia, frecuencia):
    import send_command
    import visa
    #Se crea el objeto con el generador de señal
    rm=visa.ResourceManager()
    smiq=rm.open_resource('GPIB0::28::INSTR')

    smiq.chunk_size=30000
    #Se mandan los valores introducidos por teclado mediante comandos SCPI
    send_command.send_command(smiq, (":ARB:STAT OFF"))
    send_command.send_command(smiq, (":SOUR:DM:STAT OFF"))
    freq=str(frecuencia)
    send_command.send_command(smiq, ("FREQ %s MHz") %freq)
    pot=str(potencia)
    send_command.send_command(smiq, ("POW %s dBm") %pot)
    send_command.send_command(smiq, ("OUTP:STAT ON"))

    del(smiq)
```

resultados.py: Imprime los resultados del ejercicio en un archivo independiente.

```
#Impresión de resultados en caso de barrido de potencia
def result1(nombre, Pin, medida_1t):
    import numpy as np

    archivo=open(nombre, 'w')
    archivo.write("Pin:")
    archivo.write("\n\n")
    Pin=np.matrix(Pin)
    i=0
    for i in Pin:
        i=float(i)
        archivo.write(str(round(i,3)))
        archivo.write("\n")
    archivo.write("\n\n")
    archivo.write("medida_1t:")
    archivo.write("\n\n")
    medida_1t=np.matrix(medida_1t)
    i=0
    for i in medida_1t:
        archivo.write(str(i))
        archivo.write("\n")
    archivo.write("\n\n")
    archivo.close()
    del(archivo)

#Impresión de resultados en caso de barrido de frecuencia
def result2(nombre, Fi, Fo, medida_1t):
    import numpy as np

    archivo=open(nombre, 'w')
    archivo.write("Fi:")
    archivo.write("\n\n")
    Fi=np.matrix(Fi)
    i=0
    for i in Fi:
        i=float(i)
        archivo.write(str(round(i,3)))
        archivo.write("\n")
    archivo.write("\n\n")
    archivo.write("Fo:")
    archivo.write("\n\n")
    Fo=np.matrix(Fo)
    i=0
    for i in Fo:
        i=float(i)
        archivo.write(str(round(i,3)))
        archivo.write("\n")
    archivo.write("\n\n")
    archivo.write("medida_1t:")
    archivo.write("\n\n")
    medida_1t=np.matrix(medida_1t)
    i=0
    for i in medida_1t:
        archivo.write(str(i))
        archivo.write("\n")
    archivo.close()
    del(archivo)
```

medida.py: Realiza la medición de la potencia del máximo en el analizador de señal.

```
def medida (frec1, span, rlevel, BWres1, ade):
    import send_command
    import send_query
    import time
    potencia=0
    #Configuración de parámetros introducidos por teclado para realizar la medida de potencia
    frec2=str(frec1)
    send_command.send_command(ade, (":SENS:FREQ:CENT %s MHZ; *WAI") %frec2)
    span2=str(span)
    send_command.send_command(ade, (":SENS:FREQ:SPAN %s MHZ; *WAI") %span2)
    send_command.send_command(ade, ("UNIT:POW DBM"))
    rlevel2=str(rlevel)
    send_command.send_command(ade, (":DISP:WIND:TRAC:Y:RLEV %s DBM; *WAI") %rlevel2)
    send_command.send_command(ade, (":DISP:WIND:TRAC:Y:LOG:RANG:AUTO OFF"))
    BWres2=str(BWres1)
    send_command.send_command(ade, ("BAND %s MHz; *WAI") %BWres2)

    send_command.send_command(ade, (":CALC:MARK:PEAK:EXC 10 dB"))
    send_command.send_command(ade, (":CALC:MARK:PEAK:THR -90 \n"))
    send_command.send_command(ade, ("*WAI"))
    (Status, tiempo)=send_query.send_query(ade, (":SENS:SWE:TIME?"))
    send_command.send_command(ade, ("INIT:IMM"))
    send_command.send_command(ade, ("INIT:CONT 0; *WAI"))
    send_command.send_command(ade, (":CALC:MARK:MAX"))

    if tiempo > 15:
        time.sleep(20)
    elif tiempo > 8:
        time.sleep(15)
    #Lectura de la potencia del máximo en pantalla del ADE
    (Status, potencia)=send_query.send_query(ade, (":CALC:MARK:Y?"))

    return potencia
```

medida_1tono_sinDC.py: Programa principal, utiliza las demás funciones para obtener el resultado de potencia corregido, objetivo de este trabajo.

```
#Programa principal
import SMIQ_connect
import send_command
import medida
import un_tono_SMIQ
import resultados
import matplotlib.pyplot as plt

import numpy as np
from numpy import ones
import os
workdir = os.getcwd()

#Se abre la conexión del ADE
(Status, ade) = SMIQ_connect.SMIQ_connect((['GPIB']))
ade.chunk_size=8000

send_command.send_command(ade, ("*CLS"))
#Impresión del menú con el que se va a trabajar y en el que se guardan los valores a procesar posteriormente
print ("Introduzca el número asociado al tipo de barrido que desea realizar:")
print ("1.- Barrido de potencia.")
print ("2.- Barrido de frecuencia.")
tipo_barrido = int(raw_input("Tipo de barrido:   "))

mixer = float(raw_input("Introduzca un valor distindo de 0 si el dispositivo a medir realiza conversión de frecuencias:    "))

if tipo_barrido == 1:
    p_start = float(raw_input("Introduzca la potencia inicial (en dBm):   "))
    p_stop = float(raw_input("Introduzca la potencia final (en dBm):    "))
    p_inc = float(raw_input("Introduzca el incremento de potencia (en dB):    "))
    p_final = p_stop + (p_inc/2)
    p_tam = len(np.arange(p_start, p_final, p_inc))
    P = np.arange(p_start, p_final, p_inc).reshape((p_tam, 1))
    P=np.matrix(P)

    if mixer != 0:
        Fi = float(raw_input("Introduzca la frecuencia central de la entrada (en MHz):    "))
        Fo = float(raw_input("Introduzca la frecuencia central de la salida (en MHz):    "))
    else:
        Fi = float(raw_input("Introduzca la frecuencia central (en MHz):    "))
        Fo = Fi

    vector = P

elif tipo_barrido == 2:
    P = float(raw_input("Introduzca la potencia a la entrada (en dBm):    "))

    if mixer != 0:
        f_start = float(raw_input("Introduzca el valor inicial de la frecuencia central a la entrada (en MHz):    "))
        f_stop = float(raw_input("Introduzca el valor final de la frecuencia central a la entrada (en MHz):    "))
        f_inc = float(raw_input("Introduzca el incremento de la frecuencia central (en MHz):    "))
        f_final = f_stop + (f_inc/2)
        f_tam = len(np.arange(f_start, f_final, f_inc))
        Fi = np.arange(f_start, f_final, f_inc).reshape((f_tam, 1))
        fOL = float(raw_input("Introduzca la frecuencia del OL (en MHz):    "))
        down = float(raw_input("Introduzca un valor distinto de 0 si se opera en down-converter:    "))
        if down != 0:
            Fo = Fi - fOL
        else:
            Fo = Fi + fOL
    else:
        f_start = float(raw_input("Introduzca el valor inicial de la frecuencia central (en MHz):    "))
        f_stop = float(raw_input("Introduzca el valor final de la frecuencia central (en MHz):    "))
        f_inc = float(raw_input("Introduzca el incremento de la frecuencia central (en MHz):    "))
        f_final = f_stop + (f_inc/2)
        f_tam = len(np.arange(f_start, f_final, f_inc))
        Fi = np.arange(f_start, f_final, f_inc).reshape((f_tam, 1))
        Fo = Fi

    vector = Fi

else:
    raise("La opcion introducida es incorrecta.")

#Parámetros a tener en cuenta sea cual sea el tipo de barrido requerido
gain = float(raw_input("Introduzca la ganancia esperada del dispositivo (valor positivo en dB):    "))
while gain < 0:
    print("Valor no válido de ganancia. Debe ser positiva.")
    gain = float(raw_input("Introduzca la ganancia esperada del dispositivo (valor positivo en dB):    "))

num_armonicos = float(raw_input("Introduzca el numero de armonicos que desea medir (entre 1 y 5):    "))
num_armonicos=int(num_armonicos)
while (num_armonicos != 1) and (num_armonicos != 2) and (num_armonicos != 3) and (num_armonicos != 4) and (num_armonicos != 5):
    print("Numero de armónicos no válido. Debe estar entre 1 y 5.")
    num_armonicos = float(raw_input("Introduzca el numero de armonicos que desea medir (entre 1 y 5):    "))
    num_armonicos=int(num_armonicos)

correccion_in = float(raw_input("Introduzca el valor de las perdidas a la entrada (en dB):    "))
while correccion_in < 0:
    print("Las perdidas a la entrada deben ser positivas.")
    correccion_in = float(raw_input("Introduzca el valor de las perdidas a la entrada (en dB):    "))
Pin = P - correccion_in
Pin=np.matrix(Pin)

correccion_out_cad = raw_input("Introduzca un vector de 5 elementos con el valor de las perdidas a la salida (en dB) ([valor1, valor2, ...]):    ")
correccion_out = eval(correccion_out_cad)
while len(correccion_out) != 5 or all(i >= 0 for i in correccion_out) == False:
    print("Todas las pérdidas a la salida deben ser positivas y el vector debe contener 5 valores.")
    correccion_out_cad = raw_input("Introduzca un vector de 5 elementos con el valor de las perdidas a la salida (en dB) ([valor1, valor2, ...]):    ")
    correccion_out = eval(correccion_out_cad)
```

```
#Preparación de los valores con los que se va a trabajar para obtener la matriz de resultados final
if isinstance(P, float):
    Pv = P*ones(len(vector))
else:
    Pv = P

if tipo_barrido==1:
    Fiv = Fi*ones(len(vector))
    Fov = Fo*ones(len(vector))
elif tipo_barrido==2:
    if len(Fi) == 1:
        Fiv = Fi*ones(len(vector))
        Fov = Fo*ones(len(vector))
    else:
        Fiv = Fi
        Fov = Fo

#En la variable medida_1t se guarda el valor final de la matriz de resultados. Se obtiene mediante llamadas a la función "medida"
k=0
medida_1t = [[0 for x in range(5)] for y in range(len(vector))]
while k <= (len(vector)-1):
   Pva=float(Pv[k])
   Fiva=float(Fiv[k])
   un_tono_SMIQ.un_tono_SMIQ(Pva, Fiva)

   Fova=float(Fov[k])
   rlevel = float(Pva) + gain + 15
   span = float(Fova)/float(100)
   BWres = float(span)/float(500)
   BWres=float(BWres)

   if num_armonicos >= 1:
       medida_1t[k][0] = medida.medida(Fova, span, rlevel, BWres, ade)
   else:
       medida_1t[k][0] = 0
   if num_armonicos >= 2:
       medida_1t[k][1] = medida.medida(2*Fova, span, rlevel-10, BWres, ade)
   else:
       medida_1t[k][1] = 0
   if num_armonicos >= 3:
       medida_1t[k][2] = medida.medida(3*Fova, span, rlevel-15, BWres, ade)
   else:
       medida_1t[k][2] = 0
   if num_armonicos >= 4:
       medida_1t[k][3] = medida.medida(4*Fova, span, rlevel-20, BWres, ade)
   else:
       medida_1t[k][3] = 0
   if num_armonicos >= 5:
       medida_1t[k][4] = medida.medida(5*Fova, span, rlevel-25, BWres, ade)
   else:
       medida_1t[k][4] = 0
   k=k+1

del(ade)

#Mejora de la solución añadiendo las correcciones
medida_1t=np.matrix(medida_1t)
correccion_out=np.tile(correccion_out, [len(vector), 1])
correccion_out=np.matrix(correccion_out)
medida_1t = medida_1t + correccion_out

#Representación de los resultados para los dos tipos de barrido con los que se ha trabajado
if tipo_barrido==1:
    plt.figure()
    plt.plot(Pin,medida_1t[:,0], 'k')
    plt.plot(Pin,medida_1t[:,1], 'b')
    plt.plot(Pin,medida_1t[:,2], 'r')
    plt.plot(Pin,medida_1t[:,3], 'm')
    plt.plot(Pin,medida_1t[:,4], 'g')
    plt.xlabel("Pin", fontsize=14, color='k')
    plt.ylabel("Pout", fontsize=14, color='k')
    plt.grid(True)
    plt.grid(color='0.5', linestyle='--', linewidth=1)
    plt.show()
elif tipo_barrido==2:
    plt.figure()
    plt.plot(Fo,medida_1t[:,0], 'k')
    plt.plot(Fo,medida_1t[:,1], 'b')
    plt.plot(Fo,medida_1t[:,2], 'r')
    plt.plot(Fo,medida_1t[:,3], 'm')
    plt.plot(Fo,medida_1t[:,4], 'g')
    plt.xlabel("Fout", fontsize=14, color='k')
    plt.ylabel("Pout", fontsize=14, color='k')
    plt.grid(True)
    plt.grid(color='0.5', linestyle='--', linewidth=1)
    plt.show()

#Salvado de los datos en el directorio y con el nombre deseado mediante el uso de la función "resultados"
salvar = int(raw_input("Introduzca un valor distinto de 0 si desea salvar los resultados:    "))
if salvar != 0:
    datadir = raw_input("Introduzca el nombre del directorio para salvar los datos con la ruta completa (ej: C:/Users/Usuario/Desktop  (sin comillas)):    ")
    nombre = raw_input("Introduzca un nombre identificativo para las medidas (ej: medida1.txt (sin comillas)):    ")

    os.chdir(datadir)
    if tipo_barrido == 1:
        resultados.result1(nombre, Pin, medida_1t)
    elif tipo_barrido == 2:
        resultados.result2(nombre, Fi, Fo, medida_1t)

    os.chdir(workdir)
```