ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

Desarrollo de un software de medida automatizada "bead pull" de cavidades RFQ para aceleradores de partículas

(Development of a software for automated measurement "bead pull" of RFQ cavities for particle accelerators)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autor: Alejandro Gutiérrez Camacho

Septiembre, 2021

Agradecimientos

Me gustaría agradecer este trabajo a ESS Bilbao y más en concreto a Pedro, tanto por su paciencia como por su gran ayuda y colaboración en este proyecto. También agradecer a Nagore y a Ibon el trato recibido y la ayuda mostrada cuando ha sido necesario acudir a Bilbao para realizar medidas.

También agradecerle a Tomás toda la ayuda mostrada durante todo este trabajo, estando siempre que se le ha necesitado y siendo la pieza fundamental para poder trabajar con Pedro y ESS Bilbao.

A Edurne, por ser un apoyo fundamental en esta etapa, con la que he compartido tantas mañanas en el laboratorio y nos hemos ayudado en todo lo posible.

Por último, pero no menos importante, agradecer a toda mi familia y amigos, sin los cuales este proyecto no hubiese sido llevado a cabo.

Resumen

Este Trabajo Fin de Máster, ha sido realizado en colaboración con el Consorcio ESS Bilbao. Consta de dos trabajos distintos y se ha realizado en conjunto con mi compañera Edurne Monteoliva de la Pedraja

En esta memoria en concreto, se recoge el desarrollo de un software de medida automatizada “bead pull” de cavidades RFQ para aceleradores de partículas. Este software ha sido diseñado con el lenguaje de programación Python. Este trabajo se lleva a cabo debido a la importancia que tiene el obtener el perfil del campo acelerante en los RFQ de los aceleradores de partículas, de tal manera que se optimice la aceleración, el guiado y la focalización de las partículas. Para ello, se ha elaborado esta interfaz gráfica de usuario, la cual permite obtener la diferencia de fase que existe entre la cavidad sin perturbar en comparación con la fase que se obtiene al hacer pasar una perla (dieléctrica o metálica) por la cavidad. Una vez que se han realizado las medidas por cada uno de los cuadrantes del RFQ, se procesan los resultados con un algoritmo de alineamiento para que todas las medidas partan del mismo punto de referencia. Una vez que se tiene este resultado y debido a que la diferencia de fase es proporcional al cuadrado del campo, se obtiene una medida relativa del campo del modo quadrupolar, que es el objetivo final del trabajo. A partir de este resultado, será otro algoritmo desarrollado por ESS Bilbao el encargado de llevar a cabo la optimización del campo acelerante en el RFQ.

Abstract

This Master's Thesis has been carried out in collaboration with the ESS Bilbao Consortium. It consists of two different works and has been done in conjunction with my partner Edurne Monteoliva de la Pedraja.

In this specific report, the development of a software for automated bead pull measurement of RFQ cavities for particle accelerators is collected. This software has been designed with the Python programming language. This work is carried out due to the importance of obtaining the profile of the accelerating field in the RFQs of particle accelerators, in such a way as to optimize both the acceleration, guidance and focusing of the particles. To do this, a graphical user interface has been developed, which makes it possible to obtain the phase difference that exists between the cavity without perturbation compared to the phase obtained by passing a dielectric or metallic bead through the cavity. Once the measurements have been performed for each of the quadrants of the RFQ, the results are processed with an alignment algorithm so that all the measurements start from the same reference point. Once this result is obtained and because the phase difference is proportional to the square of the field, an estimate of the quadrupole mode field profile is obtained, which is the final objective of this work. Thanks to this quadrupole mode profile, another algorithm developed by ESS Bilbao will guide the user to carry out the optimization of the accelerating field in the RFQ.

Indice

Indice de figuras

Capítulo 1: Introducción

Para situar el contexto de este TFM, se empezará introduciendo El Consorcio ESS Bilbao [1]. ESS Bilbao es un consorcio público de los gobiernos central y vasco. Su principal fin es aportar conocimiento y valor añadido en los campos de los aceleradores de partículas y las ciencias y tecnologías neutrónicas, a través de la contribución en especie a la Fuente Europea de Neutrones por Espalación, ESS (European Spallation Source). La fuente de neutrones por espalación está basada en un acelerador lineal de partículas. Este proyecto está ubicado en Lund, Suecia y tiene un total de 15 países que participan en la fase de construcción de esta instalación científica, la cual está previsto que finalice en 2015.

La Fuente Europea de Neutrones (ESS) será la más potente del mundo, siendo así un centro líder en muchos tipos de investigación, desde alimentación, farmacia, pasando por temas medioambientales hasta la nanociencia.

ESS Bilbao data de 2010 y además de las tareas que deben presentar para el gran proyecto europeo, ofrece la posibilidad de trabajar con proyectos en todo el campo de las tecnologías relacionadas con los aceleradores de partículas, de tal manera que todo el país pueda verse beneficiado de las investigaciones, ya que es uno de los centros de investigación más importantes en este ámbito en España.

Como parte de la contribución a ESS Lund, en ESS Bilbao existen diferentes paquetes de trabajo de los que forma parte. Estos proyectos son: MEBT, Sistemas RF, Blanco (Target) y el instrumento Miracles. Todos ellos son sistemas o subsistemas completos necesarios para el proyecto de ESS Lund, que incluyen como tareas, diseñar, construir, validar en las propias instalaciones de ESS Bilbao y por último instalar y poner en marcha en Suecia.

Para entender un poco más sobre los proyectos en los que trabaja ESS Bilbao se va a explicar en qué consisten y que objetivos tienen dichos proyectos:

El primero de los proyectos es el diseño y la fabricación del MEBT [2] (Medium Energy Beam Transport), que es la sección del acelerador de partículas que está entre el RFQ (Radio Frequency Quadrupole) y el DTL (Drift Tube Linac). En el MEBT (Medium Energy Beam Transport), se diagnostican y optimizan las características del haz que proporciona el RFQ para poder conseguir una eficiente aceleración en el DTL.

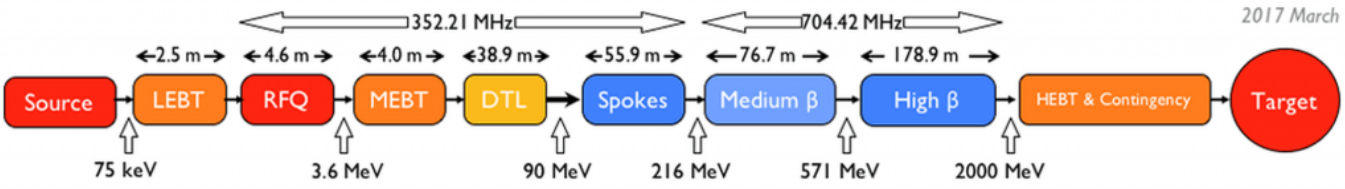

Figura 1. Diagrama del acelerador lineal de ESS con las diferentes secciones

Como puede verse en la imagen, el MEBT opera con un haz de protones a 3.6 MeV con corrientes de pico de 62.5 mA. Además, tiene una longitud total de unos 4 metros en las que se incluyen todos los componentes para cumplir sus propósitos.

El MEBT incluye un total de 11 cuadrupolos que se encargan de las características transversales del haz, también 3 cavidades buncher de RF para así cumplir con los parámetros longitudinales del haz. Para conformar el pulso de haz se utiliza un "chopper" rápido en tecnología stripline para poder conseguir tiempos de subida inferiores a 10 ns.

Este proyecto tiene los siguientes 3 objetivos:

1. Hacer que las características del haz de salida del RFQ coincidan con la entrada del DTL, utilizando cuadrupolos para las características transversales y cavidades buncher para las longitudinales, modelando los pulsos con el Fast Chopper.
2. Caracterizar el haz (corriente de haz, forma de pulso, tamaño…) mediante un conjunto completo de diagnósticos.
3. Desarrollo del control e integración en el entorno ESS EPICS (Experimental Physics and Industrial Control System).

El siguiente proyecto es el sistema de RF [3]. Como hemos dicho anteriormente, la fuente de neutrones pulsados de ESS será la más potente del mundo y está basada en un acelerador lineal (LINAC) el cual acelera un haz de protones de 62,5 mA hasta un pico de 2000 MeV.

El LINAC de ESS tiene una sección normalmente conductora que está compuesta por cavidades resonantes de RF basadas en cobre: un RFQ (Radio Frequency Quadrupole), el MEBT con 3 cavidades buncher y por último 5 tanques DTL. Seguidamente, viene la sección superconductora, la cual empieza con las cavidades "Spoke" para continuar con cavidades elípticas de media y alta beta.

Para alimentar la primera sección se utilizan 9 fuentes de potencia de RF: una de ellas para el RFQ, tres para las cavidades buncher en el MEBT y cinco para los DTL. Todas ellas operan a una frecuencia de 352.21 MHz y tienen que ser capaces de proporcionar la energía necesaria para el haz de protones de 62.5 mA, con una longitud de pulso de 2.86 mseg y una tasa de repetición de pulso de 14 Hz.

Así que este sistema de RF se puede definir como todos los componentes y subsistemas necesarios para generar y entregar la energía a todas las cavidades, toda esta cadena incluye moduladores (fuentes de alta tensión pulsada), amplificadores de RF de alta potencia, sistemas de distribución de RF, sistemas de control…

Por lo que las metas a lograr con este proyecto son las siguientes:

1. Que los sistemas de RF generen la potencia necesaria para acelerar las partículas correctamente.
2. Los sistemas de RF deben mantener tanto la amplitud como la fase del campo de aceleración en torno a una tolerancia dada.
3. Los sistemas de RF tienen que proteger tanto las cavidades como sus propios componentes de cualquier fallo.

El tercer proyecto que se va a describir es el del blanco o target [4]. En este proyecto, ESS Bilbao está encargado de la principal contribución. Este proyecto es muy complejo y multifacético ya que requiere el desarrollo en cualquier nivel de ingeniería. En el fondo de todo el sistema está el propio blanco, en el cual se generan neutrones para utilizarlos en experimentos científicos.

El proceso de espalación tiene lugar cuando el haz de protones previamente acelerado golpea el bloque de tungsteno de 11 toneladas. Es aquí donde se arrancan los neutrones, los cuales tienen que ser dirigidos a través de moderadores y guías de neutrones a los instrumentos que lo necesiten.

Para apreciar la magnitud de este proyecto y todos los campos que abarca, este proyecto incluye todos los niveles de la ingeniería nuclear, desde la física básica, hasta la construcción e instalación. El resultado final de este proyecto será una construcción de más de 3000 toneladas de acero, capaces de producir neutrones en el blanco y extraerlos para los instrumentos.

Los objetivos de este proyecto son los siguientes:

1. Diseño conceptual, diseño de detalle, fabricación y ensamblaje de componentes para la ESS Target Station.
2. Coordinar el proceso de diseño, incluyendo todas las interfaces relevantes del resto de partes interesadas.
3. Desarrollar conocimientos y capacidades técnicas en el diseño de componentes críticos en entornos de alta radiación para instalaciones específicas.
4. Producir haces de neutrones para experimentos científicos en múltiples disciplinas.

El último proyecto que se va a tratar es el instrumento Miracles[5], el cual será el espectrómetro de retrodispersión de tiempo de vuelo de neutrones de la fuente de espalación. ESS Bilbao será el principal socio en el diseño, construcción, instalación y puesta en marcha del instrumento.

Este proyecto tiene las siguientes metas:

1. Proporcionar un rango de transferencia de energía de h = 1 meV.
2. Rango dinámico de longitud de onda de aproximadamente 1,5 Å centrado en la línea elástica.
3. Obtener la mejor resolución energética alcanzable, de 2 μeV.

En este trabajo fin de máster se han realizado realmente dos paquetes de tareas diferentes junto a mi compañera Edurne Monteoliva de la Pedraja. Cada una de las memorias recogerá más en profundidad los diferentes trabajos. Igualmente, en esta memoria, se dedicará algún capitulo a hablar sobre lo más importante del trabajo escrito por Edurne.

La estructura de este trabajo será de la siguiente manera. En el **primer capítulo**, el que ahora mismo se lee, en el que se introduce y contextualiza el marco en el que se realiza el trabajo, describiendo a ESS Bilbao y trabajos en los que participa, además se hace una introducción a los aceleradores de partículas y se dan ejemplos de los mismos dentro de la industria.

En el **segundo capítulo** se hablará más en profundidad sobre las cavidades resonantes y concretamente sobre las cavidades buncher y los RFQ ya que son las 2 cavidades en las que se han centrado ambos trabajos.

En el **tercer capítulo** se describe la caracterización de los principales parámetros de las cavidades, sus modos resonantes, el factor de calidad, acoplos, R/Q.

En el **cuarto capítulo** se recoge una explicación dela técnica que se utiliza para este trabajo, bead-pull, donde quedará clara la utilidad y cómo se aplica.

En el **quinto capítulo** se describe la interfaz gráfica de usuario de Python con la que funciona el proyecto, para medidas de bead-pull en cavidades RFQ, explicando desde el desarrollo de la interfaz hasta la recogida y procesado de resultados.

Por último, en el **sexto capítulo**, se hará una valoración final del proyecto, así como unas conclusiones y líneas futuras de estudios en torno al trabajo realizado.

1.1 Introducción a los aceleradores de partículas

Para poder encontrar el primer acelerador de partículas de la historia, hay que remontarse al año 1896 en el que ocurrió uno de los mayores descubrimientos científicos para la humanidad [6]. Se trata del descubrimiento de la radioactividad que básicamente es la capacidad que tienen algunos átomos de producir energía espontáneamente, a partir de lo cual aparecieron las partículas conocidas como alfa, beta y gamma.

Una vez que se descubrió esto, en 1911, Rutherford realizó un experimento en cual consistía en emitir partículas alfa con una fuente radiactiva, colimar el haz de la misma y dispararlas hacía una fina lámina de oro. La idea principal era que las partículas alfa debían desviarse ligeramente en relación al ángulo de incidencia, pero resultó que 1 de cada diez mil rebotaba en ángulos mayores. Esto solo pudieron explicarlo diciendo que la mayor parte de los átomos del oro estaban concentrados en un mismo punto y fue lo que hoy se conoce como núcleo atómico. Este tipo de experimentos fue lo que originó la física nuclear.

El problema de todos estos experimentos era la necesidad de disponer instrumentos que generasen proyectiles que la persona que estuviese realizando el experimento fuese capaz de controlar tanto el tipo de partícula, como su energía y su flujo, así que esto fue lo que dio lugar a lo que hoy conocemos como **aceleradores de partículas y colisionadores de partículas**.

A partir de aquí fue cuando comenzó toda la carrera por ver quien conseguía crear el primer acelerador de partículas, esto duró dos décadas desde el experimento de Rutherford a principios del siglo XX y no fue hasta 1932 cuando Cockcroft y Walton construyeron el primer acelerador de partículas de iones positivos, se generaba un haz de protones a bajas energías y disparaban a isótopos de litio, gracias a este invento se consiguió la primera transmutación nuclear hecha por el hombre y ganaron el premio Nobel de Física.

Cuando los aceleradores de partículas empezaron a construirse estaban muy vinculados a la física nuclear, pero en la actualidad están extendidos a múltiples áreas de investigación, como por ejemplo en la medicina, campo en el que actualmente se estudia mucho sobre esto debido a las propiedades que tiene para tratar temas como el cáncer o diagnóstico de enfermedades.

Los aceleradores de partículas como su propio nombre indica su principal tarea es acelerar las partículas, que al fin y al cabo es aumentar la energía de las mismas. Además, las partículas deben ser guiadas y focalizadas a lo largo de su trayectoria. Para ello, los aceleradores de partículas constan de:

- Elementos por los que circulan las partículas, como son las cámaras de vacío.
- Elementos que aceleran las partículas, como por ejemplo las cavidades resonantes de RF.
- Elementos que guían estas partículas, como dipolos o cuadrupolos.
- Elementos que miden las características del haz de partículas.

Existen diferentes tipos de aceleradores de partículas, los más conocidos son los aceleradores lineales y los circulares.

Los aceleradores circulares utilizan tanto campos magnéticos como eléctricos y permiten mayores aceleraciones en espacios más reducidos. El problema que tienen es que a medida que se aumenta la aceleración se pierde una fracción de la energía y presentan un límite en la energía que puede alcanzarse. Un ejemplo muy importante es el acelerador de partículas del CERN, el LHC (Large Hadron Collider) que es un colisionador de hadrones. El LHC consiste en un anillo de 27 kilómetros de imanes superconductores en el que viajan las partículas a una velocidad muy cercana a la de la luz antes de que choquen.

Por otro lado, están los aceleradores de partículas lineales, del cual se presenta un diagrama con cada una de las partes para poder explicar un poco mejor su funcionamiento. Este trabajo se centra en explicar el acelerador lineal porque es el tipo de acelerador que utiliza ESS Bilbao, por lo que para este proyecto tiene más importancia.

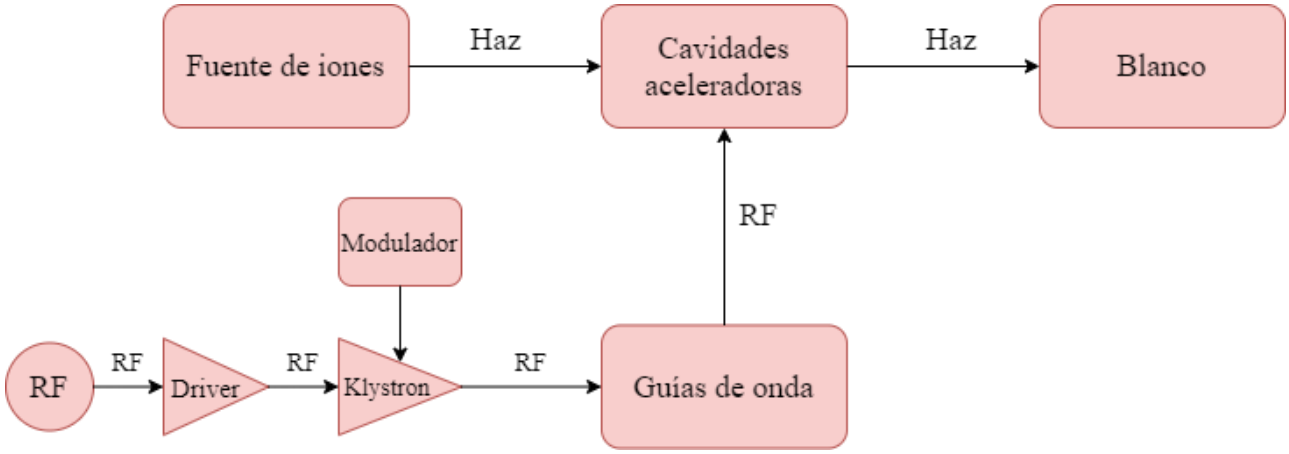

Figura 2. Diagrama simplificado de un acelerador de partículas lineal

- Fuente de iones: Es la encargada de generar el haz de partículas. Particularmente genera plasma en una cavidad que está en alto vacío y donde se inyecta hidrógeno. En determinadas condiciones de campo magnético a lo largo de la cavidad y la potencia y frecuencia de RF inyectada, se crea una situación de resonancia ciclotrón del electrón ECR (Electron Cyclotron Resonance), gracias a la cual es posible generar un plasma de alta densidad de iones. Los iones se extraen de la fuente por diferencia de potencial electrostático.

- Cavidades aceleradoras [7]: Son cavidades resonantes de RF, cámaras metálicas las cuales contienen un campo electromagnético que oscila a la frecuencia de resonancia. Estas cavidades operan en alto vacío. La aceleración se produce por la oscilación del campo eléctrico paralelo al eje longitudinal.

- Blanco o target: En los aceleradores lineales es el último elemento de la cadena y es hacía donde se dirigen las partículas. Como hemos comentado en el proyecto de ESS, se producen neutrones gracias a la colisión de los protones a una gran velocidad contra el núcleo de tungsteno.

- Generador de RF: Simplemente genera una señal a una frecuencia determinada que será la encargada de alimentar muchos de los componentes del acelerador.

- Driver: Es una etapa de preamplificación (habitualmente amplificadores de estado sólido) para que llegue con la amplitud de radiofrecuencia óptima a la entrada del amplificador de alta potencia (habitualmente un Klystron, un tetrodo o un tubo de salida inductiva). En el caso concreto de ESS Bilbao el valor a la salida de estos preamplificadores es de 300 W de potencia de pico.

- Klystron: Es el amplificador de potencia de la cadena, el cual es un tubo de vacío que funciona modulando la velocidad del haz de electrones de su interior.

- Moduladores [8]: Estos dispositivos convierten la alimentación de AC a pulsos de alta tensión los cuales alimentarán a los Klystron, por lo que son una parte muy importante de los aceleradores de partículas lineales.

- Guías de onda: Se utilizan para poder guiar la señal de radiofrecuencia a la salida del Klystron hasta las cavidades de radiofrecuencia con muy bajas pérdidas de inserción.

1.2 Acelerador de partículas de ESS

El acelerador lineal de ESS es un acelerador de protones el cual tendrá una longitud total de 602.5 metros el cual estará compuesto de las secciones que se ven la Figura 3.

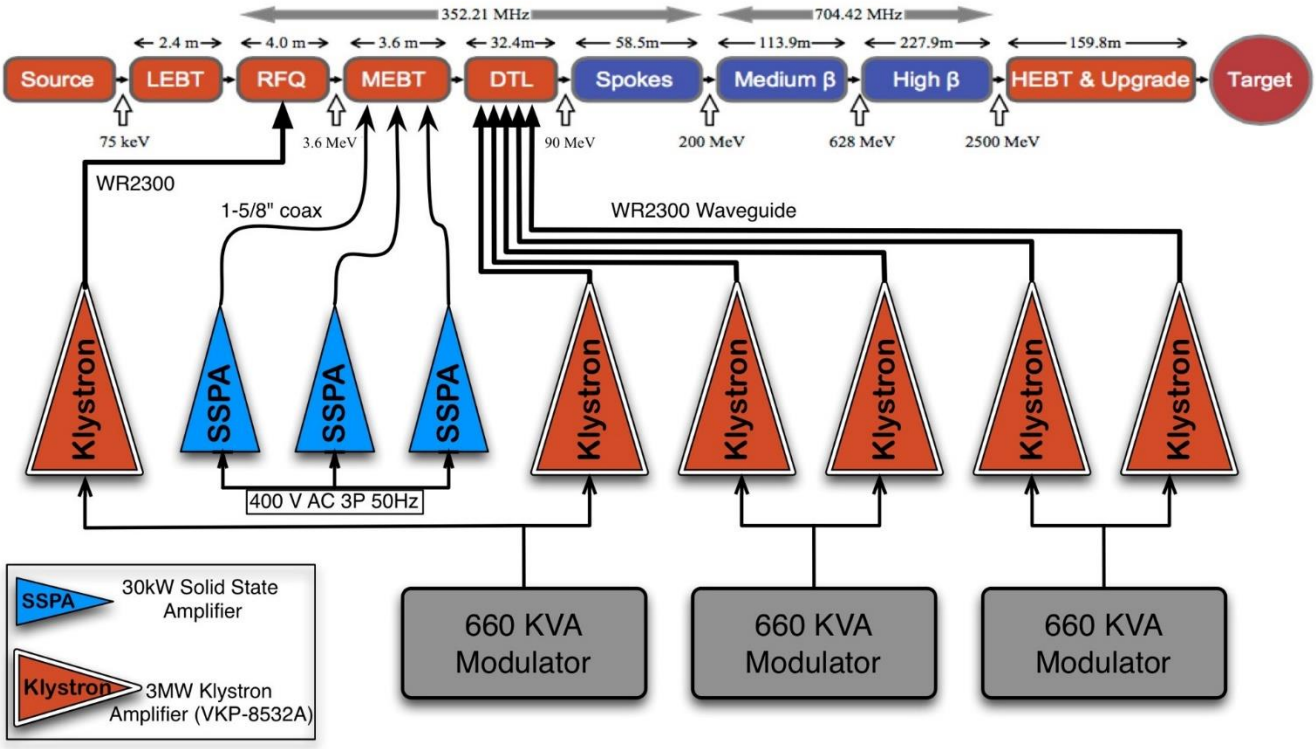

Figura 3. Acelerador de protones de ESS y estaciones de RF

En la Figura 3, los colores que se utilizan se hacen para diferencias las diferentes secciones del acelerador, en primer lugar, las secciones de color naranja (Source, LEBT, RFQ, MEBT y DTL) son las secciones conductoras que operan a temperatura ambiente, en cambio, las secciones de color azul son las secciones superconductoras. A continuación, se describen todas estas secciones del acelerador de protones de ESS:

- Source: La fuente de iones es una cámara de descarga, en su interior existe un campo electromagnético el cual oscila a una frecuencia de microondas dada. Este campo, en el momento que detecta moléculas de H_2 genera plasma de protones.

- LEBT: Estas siglas significan transporte de haz de baja energía (Low Energy Beam Transport), esta sección es para diagnóstico de haz y para conducirlo a la primera sección de aceleración, el RFQ (Radio Frequency Quadrupole).

- RFQ: Es un cuadrupolo de radiofrecuencia, el cual tiene tres propósitos fundamentales, el primero de ellos es acelerar el haz desde 70 keV hasta 3.62 MeV, en segundo lugar, debe focalizar el haz sin necesidad de imanes y por último agrupa las partículas en forma de 'bunches' (paquetes).

- MEBT; Su significado es transporte de haz de media energía (Medium Energy Beam Transport) y cumple cuatro funciones importantes, la primera es adaptar y dirigir el haz desde la salida del RFQ hasta el interior del DTL, gracias a dispositivos de instrumentación de haz los cuales permiten la colimación de la distribución transversal de las partículas. Esta sección posee un chopper (mecanismo de corte o interrupción del haz que permite conformar los flancos de subida y de bajada del haz) más rápido que el del LEBT.

- DTL: Esta sección es la encargada de subir la energía del haz desde los 3.62 MeV que salen del RFQ hasta los 90 MeV.

- Spokes: Es la primera sección de superconducción y son cavidades de tipo doble spoke, con un valor de beta^2 geométrico de 0.50.

- Mediumβ y Highβ: Estas secciones superconductoras se componen de cavidades elípticas las cuales se encargan de elevar la energía del haz hasta 1.3 GeV.

- HEBT: Estas siglas vienen de High Energy Beam Transport y es la última etapa para conducir el haz hasta el blanco rotatorio.

- Blanco o target: Esta es el último elemento del acelerador el cual se compone de una estructura giratoria de tungsteno contra la que choca el haz de partículas.

1.3 Ejemplos de aplicaciones de los aceleradores de partículas

Como se explicó en la introducción a los aceleradores de partículas, estos fueron ideados en primera instancia para investigar la estructura de la materia, pero hoy en día se ha extendido a muchos otros campos, pero donde quizá tenga más importancia actualmente sea en la medicina y en la industria. Uno de los ejemplos más claros es la radioterapia y para el diagnóstico de enfermedades a través de imagen.

En cuanto a la radioterapia [9], se utiliza dependiendo del estado del cáncer, ya que se puede utilizar tanto como tratamiento curativo como paliativo para hacer que los pacientes sufran menos con la enfermedad. El objetivo de esta técnica es administrar una dosis de radiación ionizante hacía el tumor, minimizando toda la radiación que pueda absorber el tejido sano. Existen dos tipos de radiación, la ionizante y la no ionizante, en el caso de los aceleradores de partículas se utiliza la radiación ionizante.

Figura 4. Acelerador lineal en el ámbito médico

En el caso del diagnóstico de enfermedades, la imagen PET (Tomografía por Emisión de Positrones) ha tenido una evolución exponencial. Su funcionamiento se basa en el estudio de los rayos gamma que emiten al colisionar un positrón con un electrón, de tal manera que se pueden obtener imágenes del funcionamiento de distintos órganos. Se está utilizando mucho en los últimos tiempos para realizar mamografías [10].

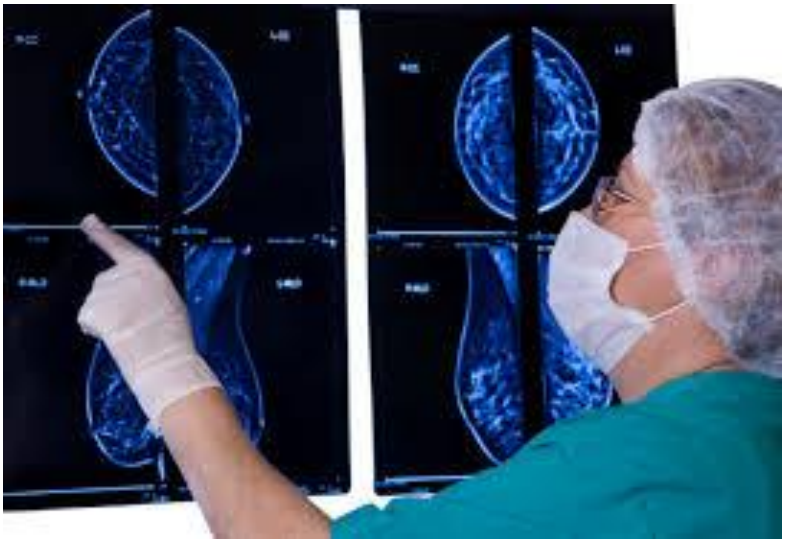

Figura 5. Mamografía con técnica PET

Fuente: https://www.i-cpan.es/es/content/medicina

En cuanto a la industria, el mayor uso que tienen es para el procesado de materiales, este uso que se le da a los aceleradores de partículas es para modificar los polímeros de los materiales, básicamente se trata de unir cadenas de polímeros para formar enlaces químicos tridimensionales. Esto hace que los materiales mejoren. Algunos de los ejemplos que podemos observar con estas técnicas es la curación del caucho o algo que nos coge más a mano es el entrecruzamiento de películas termo retractiles que se utilizan en el envasado de alimentos, lo que hace que la vida útil tanto de productos agrícolas, como de la carne y productos lácteos. También se utilizan para mejorar la resistencia y el aislamiento de los cables eléctricos.

Capítulo 2: Cavidades resonantes para aceleradores

En este capítulo se explicarán más en profundidad las cavidades resonantes en los aceleradores de partículas y en concreto se estudiarán más a fondo las cavidades buncher y RFQ ya que son los dos tipos de cavidades objetivo de estos trabajos.

El objetivo de estas cavidades es tanto acelerar como guiar y focalizar el haz de partículas mediante campos electromagnéticos alternos.

Para explicar la manera en que estas cavidades aceleran podemos poner un ejemplo con los surfistas que van a coger olas. Estos surfistas para coger la máxima velocidad que pueden lo que hacen es subirse cerca de la cresta de la ola, donde les permite acelerar al máximo y ganar más velocidad. En el caso de las partículas el objetivo es el mismo, se hacen pequeñas agrupaciones de las mismas en paquetes ("bunches") y deben llegar en el momento en el que el campo tiene el sentido correcto para acelerar, como se ve en la siguiente imagen [11]:

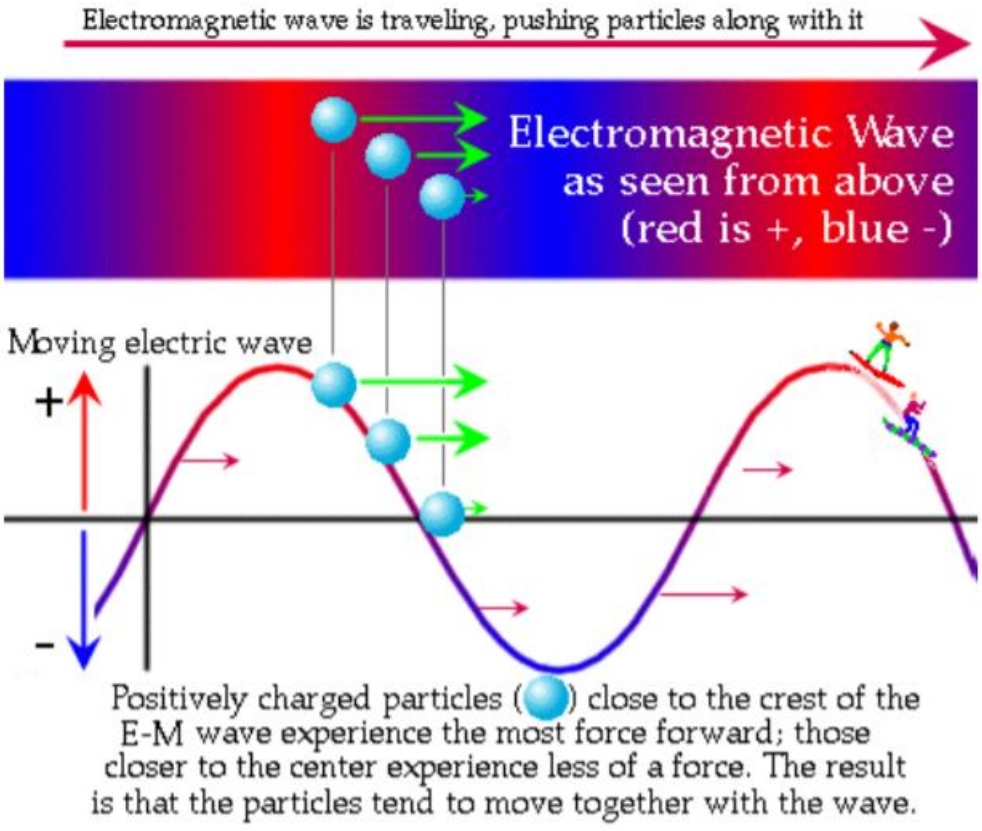

Figura 6. Funcionamiento de las cavidades

Como se puede ver en la Figura 5, el campo eléctrico se invierte en cada semiperiodo, de tal manera que las partículas deben llegar en el momento en el que el campo es máximo.

En estas cavidades, la energía queda almacenada resonando en las mismas, como dijimos en el apartado anterior. La potencia se produce mediante los generadores de RF, que luego viene amplificada por los Klystron. La frecuencia de resonancia puede variar con el tiempo, en función de la potencia disipada, la temperatura ambiente, etc. Este es uno de los mayores problemas en las cavidades, ya que la frecuencia tiene que ser muy precisa y no debe variar mucho para poder seguir acelerando las partículas como es debido.

Además de la aceleración de las partículas en cavidades, es necesario realizar un guiado y enfoque de las mismas. Para ello se utilizan imanes (bien permanentes o bien electroimanes), los cuales tienen diferente función en base al número de polos magnéticos que tengan, este número viene dado por 2^n, los tipos que existen son los siguientes:

- Con n = 1 → Dipolo, su función es curvar la trayectoria hacía uno de los planos.
- Con n = 2 → Cuadrupolo, su función es focalizar el haz y conseguir que siga la trayectoria que se necesita.
- Con n = 3 → Sextupolo, corrige las partículas que tienen energía distinta a la nominal, es decir, corrige la cromaticidad.
- Con n > 3 → Multipolos, correcciones de la óptica no lineales.

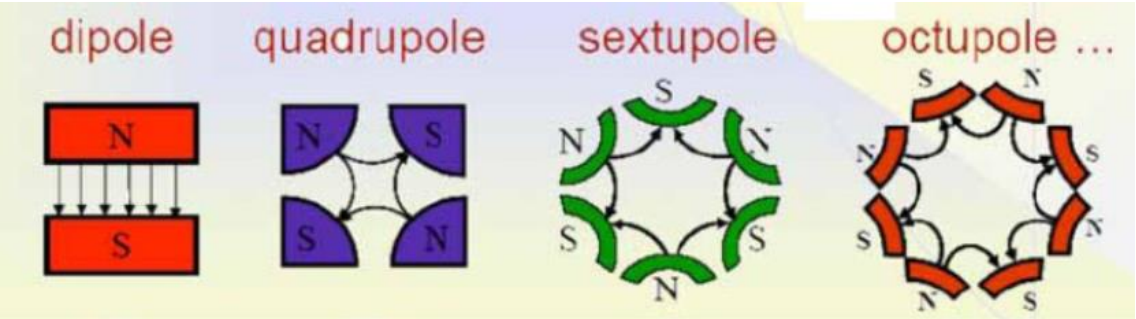

Figura 7. Modos en función de los polos

Las cavidades que forman la sección normalmente conductora de la cadena de RF, suelen estar fabricadas en cobre y operan a temperatura ambiente. Uno de los problemas de esto es que en las paredes de las cavidades se disipa una parte sensible de la potencia de RF que se suministra, por lo que es necesario potencias de RF altísimas, de incluso megavatios.

Como se dijo en el primer capítulo, ESS Bilbao es el encargado de la parte de las cadenas de RF para alimentar toda la sección normalmente conductora, la cual comprende el RFQ, MEBT y el DTL. Todas ellas operan a una frecuencia de 352 MHz, es por esto que se ven obligados a utilizar los Klystron de potencia, para poder suministrar la potencia necesaria a estas cavidades.

2.1 Cavidad Buncher

Estas cavidades se utilizan principalmente para conseguir paquetes (“bunches”, de ahí su nombre) de partículas en la dirección de la propagación, de tal manera que se optimiza su aceleración. Se formar paquetes para que sean acelerados por las crestas de las olas. Si el haz es continuo, una buena parte de las partículas serán frenadas, por lo que se perderían para la aceleración.

Estas cavidades están fabricadas en cobre en todo su interior, con un sellado que permite lograr altos niveles de vacío en las cavidades para su correcto funcionamiento. Tiene también unos elementos auxiliares como son los sintonizadores, fijos y móviles, los cuales permiten la corrección de la desviación en frecuencia. Un acoplador de potencia (de lazo magnético o de antena eléctrica), gracias al cual se introduce la potencia de RF necesaria a la cavidad para crear el campo eléctrico acelerante con unas pérdidas mínimas, y que además sirven de barrera para separar el vacío del interior de la presión ambiental del exterior y tiene unas dimensiones considerables como puede verse en la Figura7, por lo que es muy importante la calidad del mismo. También tiene una sonda de medida o “pickup” para poder extraer una muestra y comprobar el funcionamiento.

Figura 8. Acoplador cavidad Buncher

Estas cavidades se sitúan en el MEBT, entre el RFQ y el DTL. En el caso de ESS Bilbao hay 3 cavidades de este tipo, cada una con su amplificador RF de media potencia y sus correspondientes líneas de distribución coaxiales.

Cabe destacar, que las 3 cavidades buncher de la cadena de ESS han sido desarrolladas de principio a fin por ESS Bilbao, empezando por, el diseño óptico de la línea de haz para poder hacer luego el diseño electromagnético además de hacer un análisis termomecánico acoplado. De esta manera, se optimizaron las características del resonador, tales como el factor de calidad, muy importante en este tipo de cavidades ya que hablamos de un factor de calidad en torno a los 20.000, la frecuencia de resonancia, el campo acelerante y la impedancia transversal. Al hacer los estudios termomecánicos, hay que tener en cuenta que como se va a utilizar una potencia muy elevada, la cavidad se puede dilatar, así que hay que tener tanto en cuenta estos factores como los de la refrigeración líquida que tienen estas cavidades. El proceso de fabricación se ve en la Figura 9.

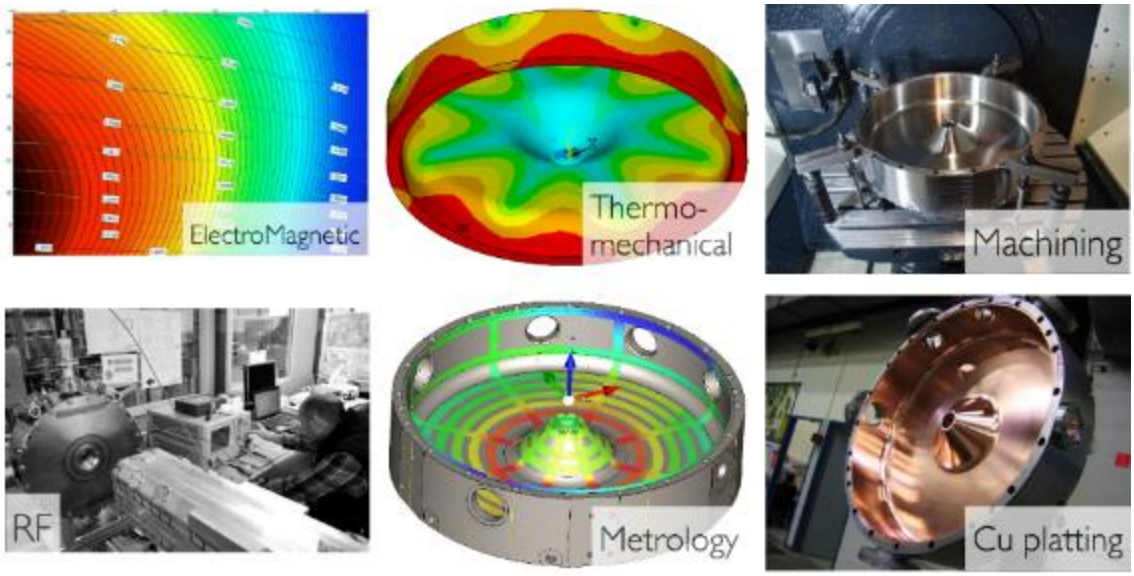

Figura 9. Proceso de fabricación de la cavidad buncher en ESS Bilbao

En 2015 se realizó el prototipo de la cavidad buncher. Este prototipo le sirvió a ESS Bilbao para validar tanto el proceso de diseño como todos los procesos de fabricación y finalmente los, sistemas de caracterización de la cavidad. A partir del último trimestre de 2019 se desarrolló todo el proceso de acondicionamiento de la primera cavidad buncher con su acoplador. Durante este proceso se trata de ir inyectando potencia de RF a la cavidad progresivamente e ir aumentando el ancho del pulso y su frecuencia de repetición hasta llegar a los niveles nominales de operación, mientras que se monitoriza y garantiza un alto nivel de vacío, sin rebasar los umbrales de aviso y de alarma. Una vez que la frecuencia de resonancia está controlada y los niveles de vacío monitorizados se puede garantizar una operación segura. A finales del año 2020 todos estos procesos fueron completados por ESS Bilbao, por lo que estas 3 cavidades han sido finalizadas con éxito. Las cavidades se muestran en la Figura10 y Figura11.

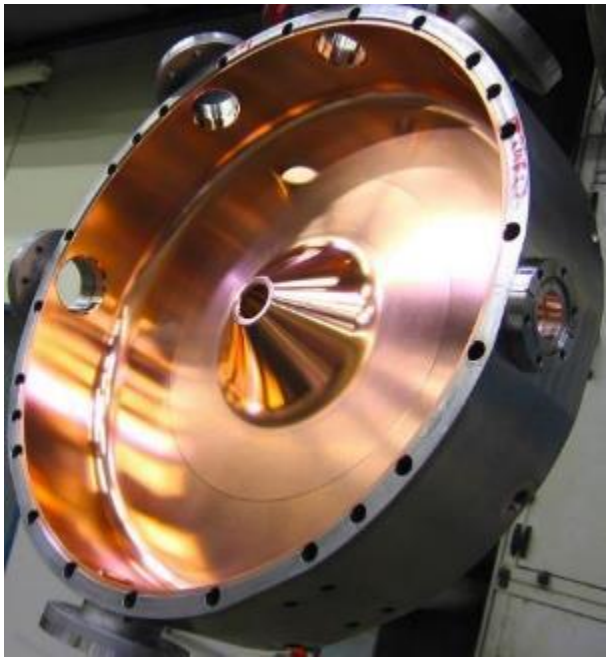

Figura 10. Cavidad buncher, medio cuerpo

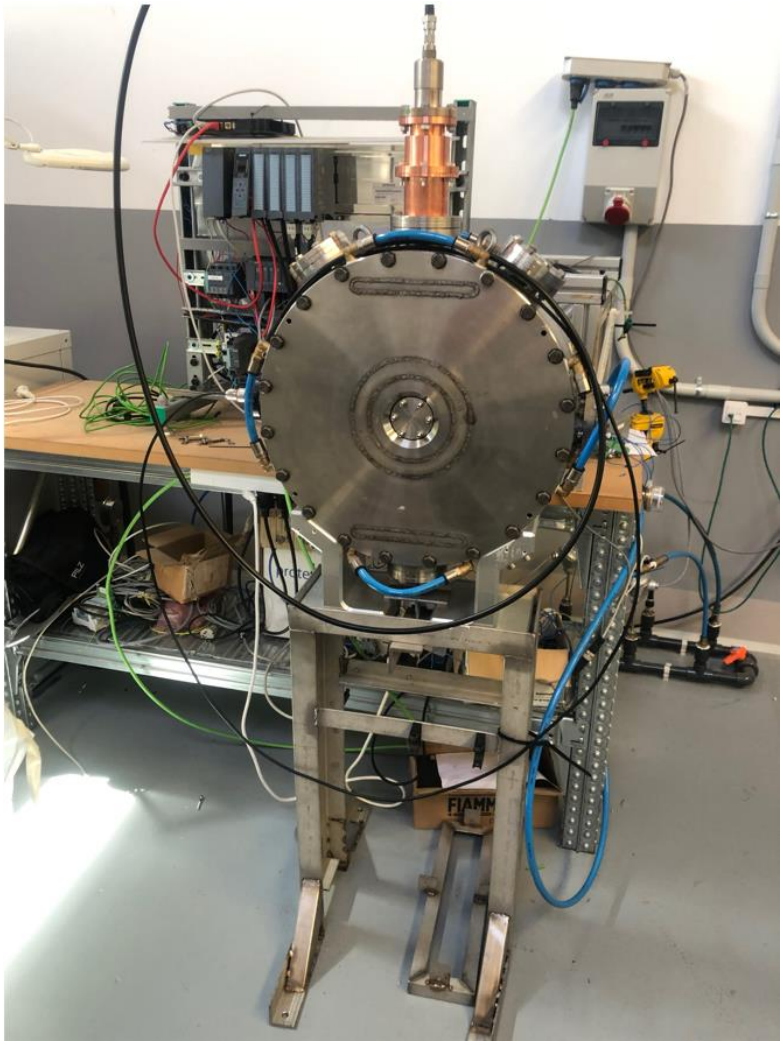

Figura 11. Cavidad buncher completa

2.2 RFQ

Este tipo de cavidad fue inventada en los 70 y fue lo que permitió una preparación más eficiente de un haz de hadrones de alta intensidad y baja potencia para que fuese acelerado en el DTL, de tal manera que la eficiencia pasó del 50% al 90%. Es conocido como RFQ ya que viene de las siglas de 'Radio Frequency Quadrupole'. Se consiguió una rápida difusión de este tipo de cavidades, ya que en la década de los 80 se logró acelerar un haz de protones de 100 keV a 640 keV con una eficiencia del 90% y a partir de aquí todos los laboratorios empezaron a utilizarlo.

El RFQ [12] es básicamente un acelerador lineal el cual acelera, agrupa y enfoca las partículas con una eficiencia muy alta. Para conseguir esto, la cavidad se excita con un campo electromagnético de radiofrecuencia del modo resonante adecuado.

Para el caso concreto de ESS Bilbao, el RFQ que se ha desarrollado tiene una longitud de 3.2 metros y está dividido en 4 secciones de 80 cm cada uno. Además, tiene las mencionadas tres funciones: acelerar el haz extraído del inyector (entra con una potencia de 45 keV y debe salir a 3 MeV), enfocar el haz y agrupar longitudinalmente las parículas.

La fabricación del primero de los segmentos llegó a ESS Bilbao en 2019 y se han realizado pruebas encaminadas a validar la fabricación, ensamblaje y desde el punto de vista de RF y medidas de frecuencia de resonancia, factor de calidad, perfil del campo acelerante y comportamiento de la temperatura del agua de refrigeración para el diseño y fabricación de los siguientes segmentos. El primer segmento puede verse en la Figura 12 y Figura13.

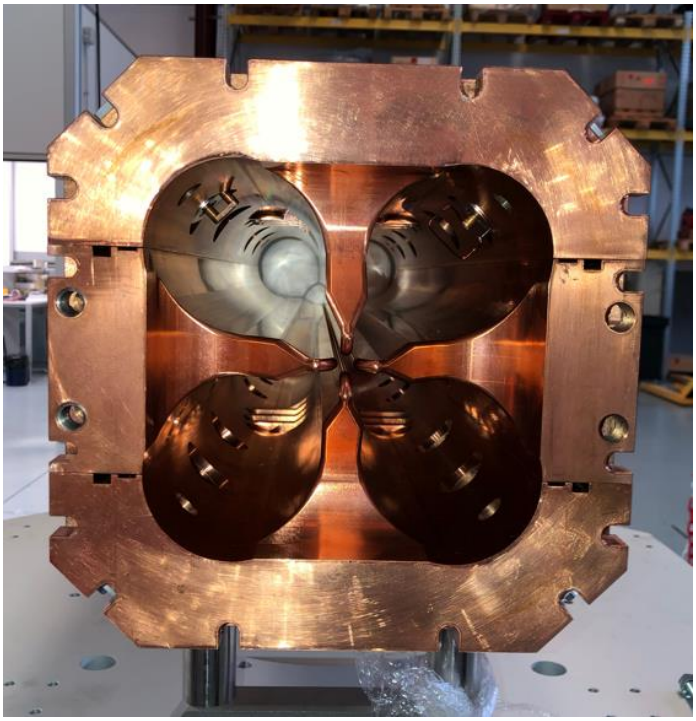

Figura 12. Entrada del RFQ

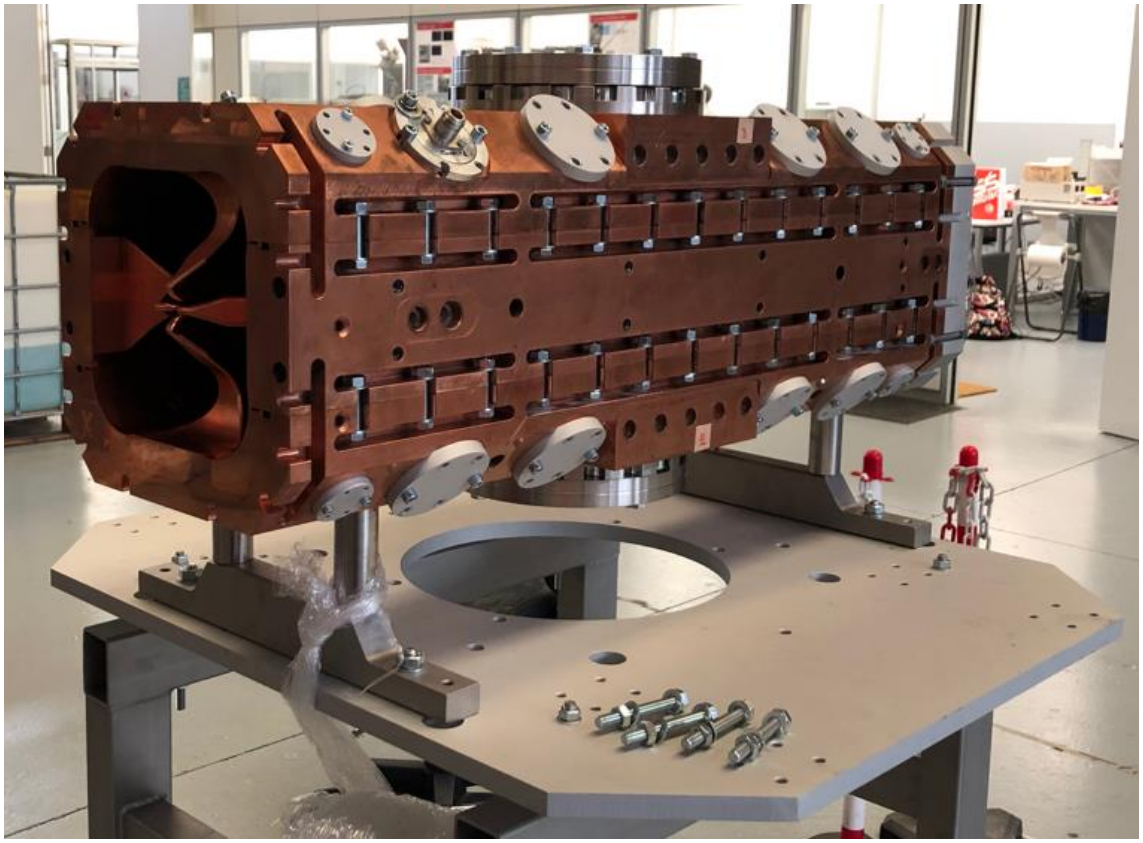

Figura 13. Segmento completo del RFQ

Capítulo 3: Medidas de caracterización de cavidades

Una vez vista la utilidad e importancia que tienen las cavidades resonantes de RF en los aceleradores de partículas, es importante también conocer cuáles son los parámetros más importantes a caracterizar en las mismas. En este caso se verán los modos resonantes, el factor de calidad, acoplos, mapas de campo y R/Q.

3.1 Modos resonantes

Los modos resonantes dentro de las cavidades resonantes satisfacen las ecuaciones de Maxwell [13].

$$\left(\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right)\begin{Bmatrix}\vec{E}\\ \vec{H}\end{Bmatrix} = 0$$

En un medio con las condiciones de contorno adecuadas, como son las cavidades resonantes en este caso, se cumplen las dos siguientes condiciones:

- Sin campo eléctrico tangencial:

  $\vec{n} \times \vec{E} = 0$

- Sin campo magnético perpendicular:

  $\vec{n} \circ \vec{H} = 0$

Una vez que se diseña una cavidad, para su geometría las ecuaciones de Maxwell tienen un número de soluciones infinito con una dependencia sinusoidal con el tiempo, entonces para obtener una aceleración más eficiente, se fabrican las cavidades resonantes con una determinada geometría y un modo con las siguientes características:

- Que el campo eléctrico esté a lo largo de la trayectoria de la partícula, como se explicó en el capítulo de las cavidades con el ejemplo de los surfistas.

- Que el campo magnético sea nulo a lo largo de la trayectoria de la partícula.

- Que la velocidad del campo electromagnético coincida con la velocidad de la partícula.

Las cavidades tienen un gran número de modos resonantes, cada uno definido por una frecuencia, pero lo importante es buscar que la resonancia sea en el modo fundamental. Normalmente los modos de orden superior no interesan y hay que intentar amortiguarlos [14].

Los modos importantes son los modos TE y TM:

- Modo TE: Transversal eléctrico, el campo eléctrico es perpendicular a la dirección de propagación en una cavidad cilíndrica.
- Modo TM: Transversal magnético, el campo magnético es perpendicular a la dirección de propagación en una cavidad cilíndrica.

Los dos modos TE que se utilizan para cavidades aceleradoras son los modos TE_{11} el cual es el modo dipolar y el TE_{21} el cual es el modo cuadrupolar.

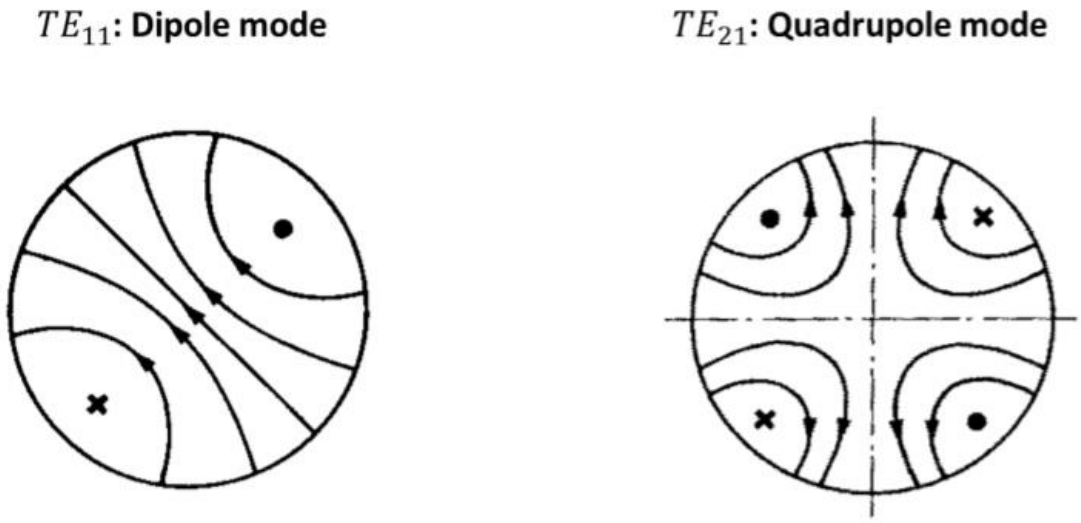

Figura 14. Modos dipolar y cuadrupolar

- Modo monopolar: Estos modos son los que no tienen variación azimutal, los modos TM de este tipo tienen un campo eléctrico longitudinal en el eje, lo cual les permite interactuar con el haz, esto es lo que sucede en las cavidades buncher, hay una parte muy pequeña en el eje que es lo que permite que las partículas se aceleren.

  La distribución radial del campo E_z sigue a J_0, hasta el lugar donde se satisfacen las condiciones de contorno, por lo que en la pared conductora de radio a $E_z = 0$ y máximo en el centro, igualmente H_f y E_r tienen un valor infinito en la pared y cero en el centro.

  La distribución de los campos para los modos monopolares pueden verse en la Figura 15.

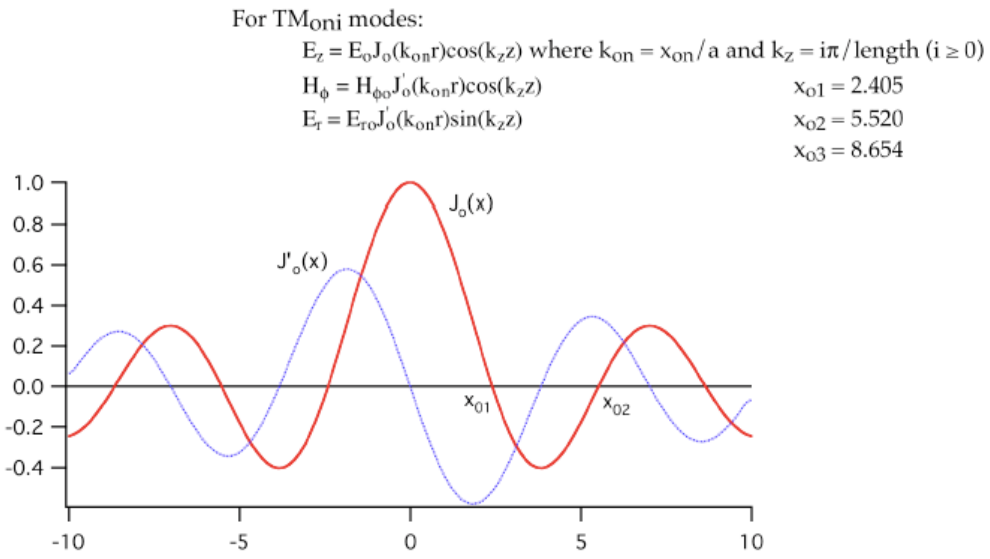

Figura 15. Distribución de campos para modos monopolares

- Modo dipolar: En este caso, la variación en relación al azimut es de un periodo completo (m = 1). En los modos TM no existe un campo longitudinal a lo largo del eje y la intensidad del campo crece linealmente con el radio de la cavidad, pero siendo de signo opuesto en cada lado del eje.

  Lo que se consigue con este gradiente transversal al campo longitudinal es que el voltaje transversal crezca y sea proporcional a la corriente y al desplazamiento del haz, de tal manera que si no existe este gradiente no existirá la crecida del voltaje transversal.

  La distribución de campos será la que se ve en la Figura 16, obviamente será al contrario que en los modos monopolo, J_1 será 0 en el eje, mientras que J_1' será máximo.

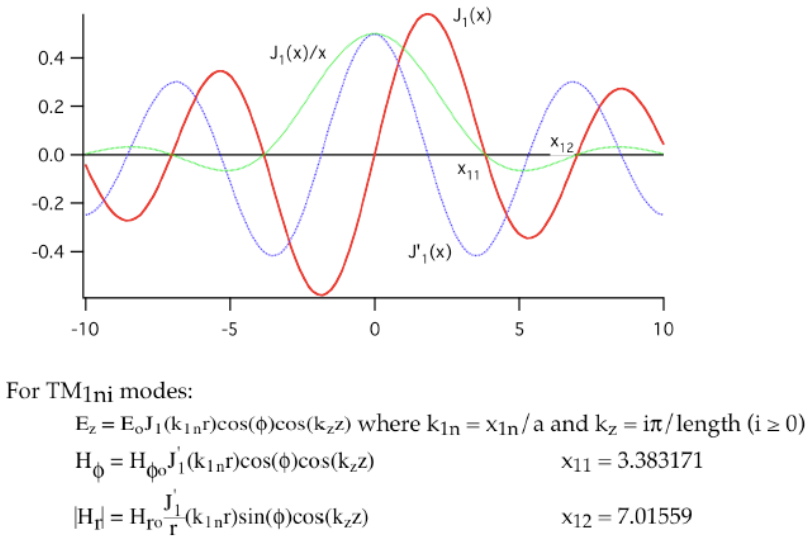

Figura 16. Distribución de campos para el modo dipolar

- Modos de orden superior (m > 1): Estos son los modos que tienen un orden azimutal más alto, por ejemplo, para m = 2 es un cuadripolo (RFQ), para m = 3 un sextupolo etc. En este caso, los campos cercanos al eje del haz van siendo progresivamente más débiles mientras la energía almacenada se concentra en el exterior de la cavidad.

  Los modos con m par no tienen el cambio de signo a cada lado del eje como se comentó para el caso del dipolo, además tienen un factor de tiempo de tránsito pequeño.

  Los modos con m impar pueden acoplarse débilmente al movimiento transversal del haz, aunque generalmente no suele ser muy problemático.

3.2 Factor de calidad

El factor de calidad es uno de los parámetros más importantes a caracterizar en las cavidades aceleradoras, ya que habitualmente se requieren valores altísimos para reducir las pérdidas en las paredes de estas, incluso de órdenes de magnitud mayores que los de los resonadores empleados habitualmente en filtros y en osciladores (coaxiales, en guía de onda, dieléctricos…). No sólo se necesita maximizar el factor de calidad, sino que se debe conocer con la mayor exactitud posible su valor en términos absolutos, a fin de estimar la amplitud del campo acelerante a partir de la potencia de RF inyectada en la cavidad. Este es uno de los objetivos en los que centra el primero de los trabajos fin de máster, el cual se explicará ampliamente en dicho trabajo [15]. Igualmente, en el presente trabajo se expondrá de manera clara a que hace referencia este parámetro y se explicará de manera breve los métodos de medida del factor de.

El factor Q [16] es básicamente la medida de la ‘calidad’ de la resonancia, en el sentido de que una cavidad con factor Q alto es aquella en la que la tasa de disipación de energía es baja, o de igual manera, presenta almacenamiento de energía durante largos periodos de tiempo.

Se puede comprender el concepto de factor de calidad mediante teoría de circuitos. Si tenemos un circuito resonante paralelo (RLC), tenemos que la frecuencia de resonancia (pulsación) se calcula de la siguiente manera:

$$\omega_0 = \frac{1}{\sqrt{LC}}\,, (1)$$

Para dicha frecuencia, el valor absoluto de la impedancia de entrada alcanza un máximo, le ocurre todo lo contrario a la admitancia, la cual está en su mínimo. El factor Q de este circuito se define básicamente como la relación que existe entre la susceptancia a la frecuencia de resonancia con la conductancia del circuito ($G_0 = 1/R_0$).

$$Q_0 = \frac{\omega_0 C}{G_0}\ , (2)$$

Si pasamos al rango de RF y microondas, los resonadores no se fabrican con capacidades, inductancias y conductancias discretas o concentradas, sino que suelen ser circuitos distribuidos como pueden ser las líneas de transmisión. Esto implica que los factores de calidad que se consiguen son mayores y ya en el momento que se necesitan utilizar potencias muy elevadas, como es el caso de los aceleradores de partículas, es cuando hay que dar paso a las cavidades cilíndricas o rectangulares. Las propiedades de todos estos resonadores cerca de la frecuencia de resonancia son similares a las del circuito RLC, su impedancia de entrada sigue mostrando una curva resonante en forma de campana, la anchura de esta curva es inversamente proporcional al factor de calidad, es por esto por lo que interesa que el factor de calidad sea muy elevado.

Para tener una definición más general del factor Q hay que darla en términos de energía y potencia, como se dijo al principio de este punto, la energía máxima almacenada (W_{max}) y la potencia promedio disipada (P_d), como puede verse en la siguiente ecuación:

$$Q = \left[\frac{\omega W_{max}}{P_d}\right]_{\omega = \omega_0}\ , (3)$$

Una vez llegado a este punto, es importante saber que el factor de calidad tiene dos variantes, el factor Q descargado y el factor Q cargado. Es posible integrar la energía que se almacena sobre el volumen del resonados para así determinar la potencia que se disipa en el mismo, bien sea debido a pérdidas en el conductor y/o a pérdidas en el dieléctrico. Si se sustituyen estos valores en la fórmula anterior tenemos el valor de la Q intrínseca del resonador descargado (Q_u). Por otro lado, cuando el resonador se encuentra cargado con redes de acoplo, se observa una frecuencia de resonancia la cual es ligeramente diferente, la cual es conocida como la frecuencia de resonancia cargada, ω_L. De esta manera se obtiene un valor de Q ligeramente menor que es la Q cargada, Q_L, para calcularlo hay que aplicar el inverso de la ecuación 3:

$$\left[\frac{P_d}{\omega W_{max}}\right]_{\omega = \omega_L} = \frac{P_0}{\omega_L W_{max}} + \frac{P_{ex}}{\omega_L W_{max}}\ , (4)$$

En este caso, la potencia disipada tiene dos partes, la potencia que se disipa en el resonador descargado, P_0, y por otro lado la potencia disipada en el circuito externo, P_{ex}, este circuito es el que se forma al hacer el equivalente Thevenin para realizar todos los cálculos. Cada uno de estos términos se puede identificar como cada uno de los parámetros Q que existen: Q_L es el factor cargado y por tanto el Q general del conjunto, Q_u que es el factor Q descargado y Q_{ex} que es el factor Q externo:

$$\frac{1}{Q_L} = \frac{1}{Q_u} + \frac{1}{Q_{ex}}\ , (5)$$

La relación que existe entre la potencia que se disipa en el resonador con la que se disipa en el circuito externo es lo que se llama factor de acoplo, denominado con k, el cual es también muy importante en las cavidades aceleradoras. Si se elimina la Q_{ex} de la ecuación 4 y se utiliza la expresión de la ecuación 5, se puede obtener una relación entre la Q descargada, Q_u y la Q cargada, Q_L:

$$Q_u = Q_L(1+k), (6)$$

En este TFM, se utilizan 3 tipos de métodos para medir el factor de calidad, el primero de ellos y más simple es el método de los 3 puntos. Esta medida tiene 2 variantes, en modo de reflexión y en modo de transmisión, para la primera de ellas lo primero que se debe hacer es localizar la frecuencia de resonancia con un analizador de redes, f_0, conectando únicamente el puerto 1 del analizador de redes al acoplador seguido de la cavidad, en dicho punto el valor de S_{11} será mínimo, después habrá que localizar dos frecuencias, f_2 y f_3, las cuales serán donde el Im(S_{11}) sea mínimo y máximo respectivamente, una vez que se tienen estas 3 frecuencias, el factor de calidad se calcula de la siguiente manera:

$$Q_L = \frac{f_0}{f_3 - f_2}, (7)$$

Una vez que se ha obtenido el valor de la Q cargada (Q_L), si sustituimos el valor en la expresión (6), es posible obtener también el valor de la Q descargada. Este valor también es posible obtenerlo si se buscan las frecuencias donde Im(Z) = Re(Z) y donde Im(Z) = - Re(Z), tendremos f_4 y f_5 respectivamente y aplicando la siguiente fórmula se obtiene el valor de la Q descargada:

$$Q_u = \frac{f_0}{f_5 - f_4}, (7)$$

Para la segunda de las opciones, medida en transmisión, se debe conectar el puerto 1 del analizador de redes al acoplador y el 2 a un captador, de tal manera que se pueda medir el S_{11}, S_{22} y S_{21}. Como en el caso anterior, se coloca un marcador a la frecuencia de resonancia, donde el S_{21} tendrá un valor máximo, se deben colocar otros 2 marcadores a las frecuencias donde el S_{21} cae 3dB, lo cual es su ancho de banda a 3 dB, así obtendremos f_2 y f_3. Aplicando la expresión (7) se obtiene el valor de la Q cargada. En esta medida es más complicado obtener la Q descargada ya que los acoplos no se obtienen de manera inmediata. Se pueden obtener los factores de acoplo configurando el analizador de redes en formato VSWR para el marcador de la frecuencia de resonancia, si esta sobreacoplado entonces $K_{m1,2}$ = 1/SWR y si tenemos acoplo crítico o infraacoplado $K_{m1,2}$ = SWR.

Cuando existe más de un acoplo, los acoplos medidos son diferentes a los reales. Para obtener los reales se debe aplicar las siguientes fórmulas:

$$k_1 = k_{m1}\left(\frac{1 + k_{m2}}{1 - k_{m1}k_{m2}}\right), (8)$$

$$k_2 = k_{m2}\left(\frac{1 + k_{m1}}{1 - k_{m1}k_{m2}}\right), (9)$$

Una vez que se han obtenido los acoplos reales, se puede proceder al cálculo de la Q descargada, la cual se calcula con la siguiente expresión:

$$Q_u = Q_L(1 + k_1 + k_2), (10)$$

Si se da el caso en el que los acoplos son idénticos, se puede obtener la Q descargada utilizando la Q cargada y las pérdidas por inserción a la frecuencia de resonancia:

$$Q_u = \frac{Q_L}{1 - S_{11}(f_0)}, (11)$$

Como bien se ha dicho, estos métodos son conocidos como métodos de los 3 puntos y proporcionan una medida aproximada y rápida tanto del factor de calidad como de los acoplos, el problema es que son métodos muy sensibles a errores de medida. Es por esto que para conseguir medidas más precisas y con menor error se debe recurrir a métodos de ajustes de curvas (curve fitting). En particular en el TFM se recurren a métodos de los autores Kajfez y K. Leong. Estos métodos requieren un análisis más complejo y están explicados en profundidad en [15].

3.3 Factor de acoplo

Como bien se ha dicho, este es otro de los parámetros a tener en cuenta en cuanto a las cavidades, ya que es la relación existente entre la potencia que se disipa en el circuito exterior con la potencia que se disipa en el propio resonador. Para poder medir el factor de acoplo, si nos fijamos en la Figura 17, se puede observar que el comportamiento del coeficiente de reflexión con la frecuencia representa un círculo, donde se destacan dos puntos.

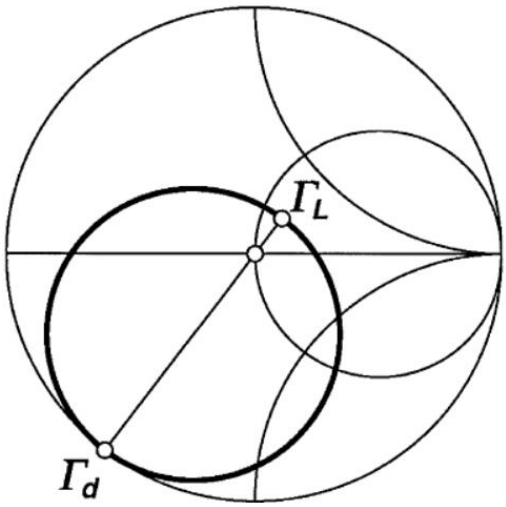

Figura 17. Coeficiente de reflexión de un resonador

Cuando la frecuencia de operación está alejada de la frecuencia de resonancia, el coeficiente de reflexión que se obtiene es Γ_d. En cambio, cuando la frecuencia f = f_L (mínima reflexión) tenemos el siguiente coeficiente de reflexión, Γ_L. El diámetro del círculo Q es la distancia que existe entre Γ_d y Γ_L y con ese diámetro, d, se puede obtener el factor de acoplo:

$$k = \frac{1}{\frac{2}{d} - 1}, (7)$$

Se distinguen tres situaciones en función del valor del factor de acoplo:

- Acoplo crítico: Esto ocurre cuando k = 1, se disipa la misma cantidad de potencia en el circuito externo como en el resonador. El círculo Q toca el centro de la gráfica de Smith. Cuando el acoplo es crítico, la Q descargada es el doble que la Q cargada.

- Acoplo subcrítico: Esto ocurre cuando k < 1, la potencia que se disipa dentro del resonador es superior a la que se disipa en el circuito externo. El círculo Q no rodea al centro de la gráfica de Smith. Cuando ocurre esto, los valores de Q cargada y descargada tienen menos diferencia.

- Acoplo sobrecrítico: Este caso es que k > 1, en este último caso la potencia que se disipa en el resonador es inferior a la potencia disipada en el circuito exterior. El centro de la gráfica de Smith está rodeado por el círculo Q. La diferencia entre los valores de Q cargada y descargada es mayor.

3.4 R/Q

Este parámetro es una medida física de las cavidades ya que depende únicamente de la geometría de la cavidad y nos da una medida de cuanto se puede llegar a acelerar en la cavidad para una determinada potencia disipada. Este parámetro se utiliza para poder optimizar las cavidades aceleradoras, de tal manera que se consiga una aceleración mejor.

Un parámetro importante cuando se está hablando de la R/Q [17] es la impedancia transversal (Shunt impedance, R_{sh}). Este parámetro es la relación que existe entre la tensión acelerante y la corriente del haz. Como es una impedancia su unidad son los ohmios y el valor que tiene es el siguiente:

$$R_{sh} = \frac{V_c^2}{P_d}, (8)$$

Para maximizar este parámetro y por ende maximizar la aceleración hay que tratar de conseguir que la impedancia transversal alcance el valor más alto y así conseguir que una optimización de las cavidades. Ciertos autores que escriben sobre esta impedancia transversal añaden un factor 2 en el divisor (“definición circuital”).

Finalmente, la expresión para calcular la R/Q es la siguiente:

$$\frac{R}{Q} = \frac{\frac{V_c^2}{P_d}}{\frac{\omega U}{P_d}} = \frac{V_c^2}{\omega U} \ (\Omega), (9)$$

Siendo V_c el voltaje de aceleración de la cavidad normalizado a la energía almacenada en la misma U, a la frecuencia de resonancia ω.

Para poder realizar el cálculo de todo este proceso se debe realizar un mapeado de la distribución del campo y así poder integrar la impedancia transversal de la cavidad. Para ello, se utiliza una técnica llamada bead-pull, que básicamente consiste en hacer pasar una perla con unas características eléctricas y magnéticas conocidas por la cavidad. Esta técnica se explica con más detalle en el siguiente capítulo. Al hacer esto, se puede expresar la impedancia transversal de la siguiente manera:

$$RT^2 = \frac{(V_c T)^2}{P} = \frac{\left[\int E_Z(Z) e^{jw\frac{Z}{v}} dZ\right]^2}{P}, (10)$$

$$siendo\ E^2 = -\frac{\Delta\omega P_d Q(\varepsilon_r + 2)}{\omega^2 \pi r^3 \varepsilon_0 (\varepsilon_r - 1)}, (11)$$

En la expresión (11), los parámetros ε_r y r son ambos referidos a la perla que se hace pasar por la cavidad. Finalmente, siendo una cavidad simétrica en torno al eje Z, se cumple la siguiente expresión:

$$\frac{RT^2}{Q} = -\frac{Q(\varepsilon_r + 2)}{\omega \pi r^3 \varepsilon_0 (\varepsilon_r - 1)} \left[\int \sqrt{\frac{\Delta\omega}{\omega}}(z) \left(cos\frac{\omega z}{c} + jsin\frac{\omega}{c}\right) dz\right]^2, (12)$$

Capítulo 4: Bead-pull

En este capítulo, se va a describir la técnica del bead-pull, en qué consiste y qué finalidades tiene. En este TFM se ha elaborado una interfaz de usuario con el lenguaje de programación Python, la cual muestra los resultados que se obtienen al realizar una serie de medidas con la técnica del bead-pull.

Esta técnica consiste en utilizar un cable no conductor, normalmente se utilizan cables de pesca de nylon, trilene o similares para tirar de una perla (bead), la cual puede ser dieléctrica, metálica o ferromagnética (en función de si se desea perturbar el campo eléctrico o el campo magnético), a través de una cavidad de RF y así poder medir la distribución de los campos electromagnéticos en el interior de la misma. La técnica de bead pull se basa en la teoría de pequeñas perturbaciones de J.C Slater [18]. En este trabajo se utiliza esta técnica para cavidades RFQ concretamente.

Al introducir la perla en la cavidad, tanto la frecuencia de resonancia como el factor de calidad pueden sufrir cambios debido a las perturbaciones de los campos electromagnéticos, las medidas de bead-pull implican 2 tipos de perturbaciones:

1. El primero de ellos (Figura 18) es una pequeña perturbación del material al introducir una perla dieléctrica pequeña en un gran volumen como es la cavidad. El cambio en la permitividad del aire en comparación con el de las perlas puede ser grande, pero el volumen de las perlas es pequeño.

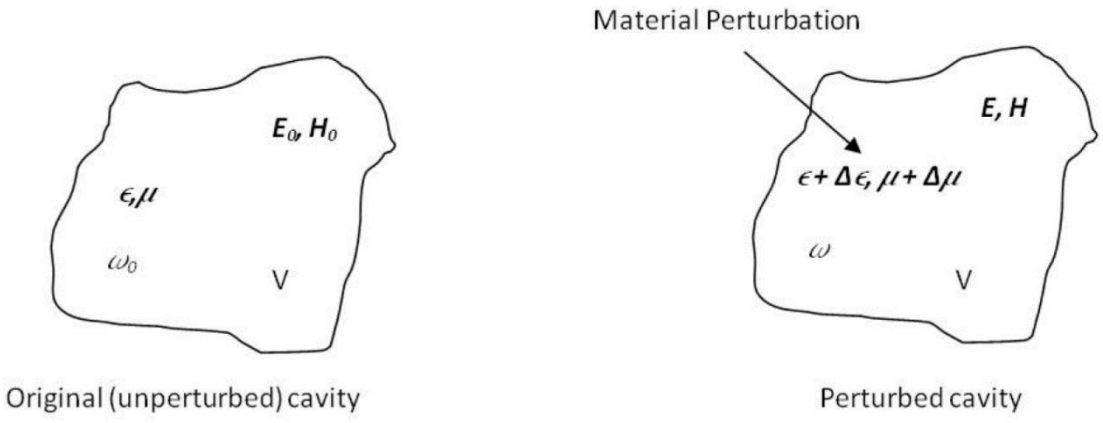

Figura 18. Cavidad perturbada por una perla dieléctrica

2. El segundo es un pequeño cambio en el volumen de la cavidad al introducir una pequeña perla metálica en la cavidad (Figura 19). Aunque este cambio puede estar o no estar en la pared, el volumen de la perla es pequeño en comparación con la cavidad.

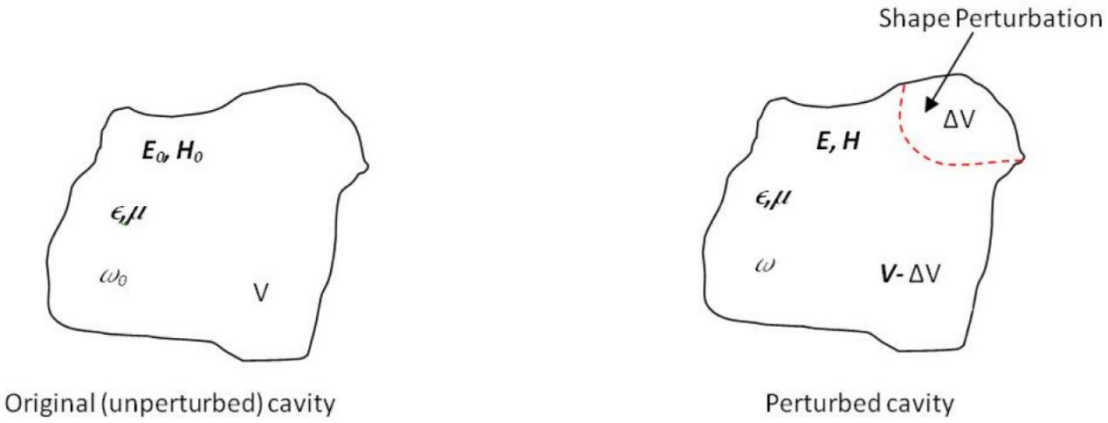

Figura 19. Cavidad perturbada por una perla metálica

En definitiva, las perlas que se pueden utilizar pueden ser dieléctricas $\varepsilon_r > 1$, metálicas $\sigma \rightarrow \infty$ o de bajo σ pero ferromagnéticas $\mu_r > 1$. También se pueden utilizar materiales con pérdidas siempre que la Q cargada en la cavidad tenga en cuenta el cambio de frecuencias.

Antes de la perturbación, el campo electromagnético dentro de la cavidad se puede describir de la siguiente manera [17]:

$$E = E_0 e^{j\omega t} \ , (13)$$

$$H = H_0 e^{j\omega t} \ , (14)$$

Los valores de E_0 y H_0 son en función de la posición antes de la perturbación. Una vez que se introduce la perla y se genera la perturbación, se modifican ligeramente la magnitud de los campos eléctrico y magnéticos, así como la frecuencia de resonancia:

$$E' = (E_0 + E_1) e^{j(\omega + \Delta\omega)t} \ , (15)$$

$$H' = (H_0 + H_1) e^{j(\omega + \Delta\omega)t} \ , (16)$$

En este punto $E_1 << E_0$ y $H_1 << H_0$ porque son resultados de una pequeña perturbación dentro de un gran volumen, si se tienen en cuenta las ecuaciones de Maxwell:

$$\nabla \times E = -\frac{\partial B}{\partial t} \ , (17)$$

$$\nabla \times H = \frac{\partial D}{\partial t} \ , (18)$$

Aplicando las ecuaciones (13) y (15) a (17), se obtiene el siguiente resultado:

$$\nabla \times E_0 = -j\omega B_0 \ , (19)$$

$$\nabla \times (E_0 + E_1) = -j(\omega + \Delta\omega)(B_0 + B_1) \ , (20)$$

Operando también sobre estos dos últimos resultados, ecuaciones (19) y (20), se obtiene que la perturbación es:

$$\nabla \times E_1 = -j\omega B_1 - j\Delta\omega(B_0 + B_1) \ ,(21)$$

De igual manera que se ha hecho con el campo eléctrico, se hace con el campo magnético, operando primero sobre (14), (16) a (18):

$$\nabla \times H_0 = j\omega D_0 \ ,(22)$$

$$\nabla \times (H_0 + H_1) = j(\omega + \Delta\omega)(D_0 + D_1) \ ,(23)$$

De igual manera, se opera sobre los resultados (22) y (23) para continuar con los mismos pasos que se realizó sobre la perturbación del campo eléctrico:

$$\nabla \times H_1 = j\big(\omega D_1 + \Delta\omega(D_0 + D_1)\big) \ ,(24)$$

Utilizando H_0* que es el conjugado de H_0 y multiplicándolo en la ecuación (21), de igual manera que con E_0* conjugado de E_0 multiplicando en la ecuación (24):

$$H_0^* \cdot \nabla \times E_1 = -j[\omega H_0^* \cdot B_1 + \Delta\omega H_0^* \cdot (B_0 + B_1)] \ ,(25)$$

$$E_0^* \cdot \nabla \times H_1 = j[\omega E_0^* \cdot D_1 + \Delta\omega E_0^* \cdot (D_0 + D_1)] \ ,(26)$$

Operando sobre (25) – (26):

$$E_0^* \cdot \nabla \times H_1 - H_0^* \cdot \nabla \times E_1 = j\omega(E_0^* \cdot D_1 + H_0^* \cdot B_1)$$

$$+j\Delta\omega[(E_0^* \cdot D_0 + H_0^* \cdot B_0)(E_0^* \cdot D_1 + H_0^* \cdot B_1)] \ ,(27)$$

Utilizando la operación diferencial vectorial y las relaciones (19) y (22) tenemos los siguiente:

$$\nabla \cdot (E_0^* \times H_1 - H_0^* \times E_1)$$

$$\equiv H_1 \cdot \nabla \times E_0^* - E_0^* \cdot \nabla \times H_1 + E_1 \cdot \nabla \times H_0^* + H_0^* \cdot \nabla \times E_1$$

$$= j\omega H_1 \cdot B_0^* + j\omega E_1 \cdot D_0^* - (E_0^* \cdot \nabla \times H_1 - H_0^* \cdot \nabla \times E_1)$$

Esto lleva a:

$$(E_0^* \cdot \nabla \times H_1 - H_0^* \cdot \nabla \times E_1)$$

$$= j\omega(H_1 \cdot B_0^* + E_1 \cdot D_0^*) - \nabla \cdot (E_0^* \times H_1 - H_0^* \times E_1) \ ,(28)$$

Aplicando (28) a (27):

$$j\omega(H_1 \cdot B_0^* + E_1 \cdot D_0^*) - \nabla \cdot (E_0^* \times H_1 - H_0^* \times E_1)$$

$$= j\omega(E_0^* \cdot D_1 + H_0^* \cdot B_1) + j\Delta\omega[(E_0^* \cdot D_0 + H_0^* \cdot B_0) + (E_0^* \cdot D_1 + H_0^* \cdot B_1)] \ ,(29)$$

Integrando en todo el volumen de la cavidad V_0 en ambos lados de la ecuación (29):

$$j\Delta\omega \iiint_{V_0} [(E_0^* \cdot D_0 + H_0^* \cdot B_0) + (E_0^* \cdot D_1 + H_0^* \cdot B_1)]\, dv$$

$$= j\omega \iiint_{V_0} [(E_1 \cdot D_0^* - E_0^* \cdot D_1) + (H_1 \cdot B_0^* - H_0^* \cdot B_1)]\, dv$$

$$- \iiint_{V_0} \nabla(E_0^* \times H_1 - H_0^* \times E_1)\, dv \quad , (30)$$

De acuerdo con el teorema de la divergencia, el segundo término de la expresión (30) puede convertirse directamente en la integración de la superficie de la pared interior de la cavidad S_0 donde E_0 y H_0 desaparecen.

$$\iiint_{V_0} \nabla(E_0^* \times H_1 - H_0^* \times E_1)\, dv = \iint_{S_0} (E_0^* \times H_1 - H_0^* \times E_1) \cdot ds = 0 \, , (31)$$

De esta manera, la expresión (30) se convierte directamente en la siguiente ecuación:

$$\frac{\Delta\omega}{\omega} = \frac{\iiint_{V_0} [(E_1 \cdot D_0^* - E_0^* \cdot D_1) + (H_1 \cdot B_0^* - H_0^* \cdot B_1)]\, dv}{\iiint_{V_0} [(E_0^* \cdot D_0 + H_0^* \cdot B_0) + (E_0^* \cdot D_1 + H_0^* \cdot B_1)]\, dv} \quad , (31)$$

La ecuación (31) es una expresión exacta del cambio de frecuencia relativa de la cavidad debido a las perturbaciones o bien en la forma o el medio de la cavidad. Para su aplicación práctica con la técnica bead-pull se necesita alguna aproximación más. Como $|D_1| << |D_0|$ y $|B_1| << |B_0|$ o su contribución en V_1 a la integración del volumen de la cavidad V_0 es muy pequeña, el segundo término de dicha expresión es aproximadamente 0. Además, E_1, D_1 y H_1 y B_1 en el volumen $V_0 - V_1$ son prácticamente iguales a los valores que toman en el volumen V_0. Por lo tanto, en la expresión (31), el volumen de integración V_0 puede ser aproximado al volumen V_1. Por lo tanto, la expresión quedaría de la siguiente manera:

$$\frac{\Delta f}{f} = \frac{\Delta\omega}{\omega} = \frac{\iiint_{V_1} [(E_1 \cdot D_0^* - E_0^* \cdot D_1) + (H_1 \cdot B_0^* - H_0^* \cdot B_1)]\, dv}{\iiint_{V_0} [(E_0^* \cdot D_0 + H_0^* \cdot B_0)]\, dv} \quad , (32)$$

Si V_1 es un volumen metálico, dentro de V_1 se dan las siguientes características con las que se pueden seguir haciendo aproximaciones:

$$E' = 0 \qquad D' = 0 \qquad B' = 0 \qquad H' = H_0 \quad , (33)$$

Entonces, los campos dentro de V_1 también cambian, por lo que se obtendría lo siguiente:

$$E_1 = E' - E_0 = -E_0 \qquad D_1 = D' - D_0 = 0 \quad ,(34)$$

$$B_1 = B' - B_0 = -B_0 \qquad H_1 = H' - H_0 = 0 \quad ,(35)$$

Si se aplica lo obtenido en las expresiones (34) y (35) a la expresión (32), el cambio de frecuencia de la cavidad debido a una perturbación del límite metálico es:

$$\frac{\Delta f}{f} \cong \frac{\oiint_{V_1}(H_0^* \cdot B_0 - E_0 \cdot D_0^*)\, dv}{\oiint_{V_0}(E_0^* \cdot D_0 + H_0^* \cdot B_0)\, dv} \quad ,(36)$$

En el caso de que V_1 se deforme de tal manera que pueda considerarse con una pequeña perturbación, la nueva superficie S_1 será paralela a la superficie original S_0 entonces el perímetro del volumen metálico de δV:

$$B_0 = \mu H_0 \qquad D_0 = \varepsilon E_0 \quad ,(37)$$

$$\frac{\Delta f}{f} \cong \frac{\oiint_{\delta V}(\mu|H_0|^2 - \varepsilon|E_0|^2)\, dv}{\oiint_{V_0}(\mu|H_0|^2 + \varepsilon|E_0|^2)\, dv} \quad ,(38)$$

En relación a la expresión (38) se pueden sacar las siguientes tres conclusiones:

- Si el cambio de la frecuencia es debido a una perturbación del campo eléctrico, la frecuencia disminuye, en cambio sí se perturba el magnético, esta aumenta.

- Si se utiliza una perla dieléctrica, únicamente se perturba el campo eléctrico. Pero si se utiliza una perla metálica se perturban ambos campos, tanto el eléctrico como el magnético.

- Para obtener un campo eléctrico y magnético independiente, se debe realizar el bead-pull tanto con una perla dieléctrica como con una metálica, aplicando la técnica por separado.

En el caso de utilizar una perla dieléctrica, si se lleva a cabo el mismo proceso realizado anteriormente, se llega a la siguiente expresión de la variación de frecuencia:

$$\frac{\Delta f}{f} = \frac{\oiint_{\delta V}[(\mu_r - 1)\mu_0 H_1 \cdot H_0^* + (\varepsilon_r - 1)\varepsilon_0 E_1 \cdot E_0^*]\, dv}{\oiint_{V_0}(\mu|H_0|^2 + \varepsilon|E_0|^2)\, dv} \quad ,(39)$$

De igual manera, con la expresión (39) se pueden existen otras cuatro conclusiones que se exponen a continuación:

- Si se utiliza el bead-pull con una perla dieléctrica con $\varepsilon_r > 1$ y $\mu_r = 1$, se mediría únicamente el campo eléctrico.

- Materiales con un alto μ_r se pueden utilizar para medir el campo magnético, pero es mejor hacerlo con perlas metálicas.

- Cuando ε_r >> 1 y μ_r >> 1, E_1 y H_1 no pueden ser aproximados a 0.

En el caso de tener una pequeña esfera no conductora, de radio r, el campo no perturbado puede considerarse uniforme en una región más grande que la perla. Esto se puede demostrando con la siguiente ecuación:

$$\frac{\Delta\omega}{\omega} = \frac{\Delta U}{U} = -\frac{\omega\pi r^3}{PQ}\left[\varepsilon_0\frac{\varepsilon_r - 1}{\varepsilon_r + 2}{E_0}^2 + \mu_0\frac{\mu_r - 1}{\mu_r + 2}{H_0}^2\right], \quad (40)$$

En el caso de tener una perla dieléctrica (μ_r = 1), la expresión 40 se puede reducir a la siguiente expresión:

$$\frac{\Delta\omega}{\omega} = -\frac{\pi r^3}{U}\left[\varepsilon_0\frac{\varepsilon_r - 1}{\varepsilon_r + 2}{E_0}^2\right], \quad (41)$$

Por otro lado, en el caso de tener una perla metálica (μ_r → 0 y ε_r → ∞) la expresión 40 se puede reducir a:

$$\frac{\Delta\omega}{\omega} = -\frac{\pi r^3}{U}\left[\varepsilon_0{E_0}^2 + \mu_0\frac{\mu_0}{2}{H_0}^2\right], \quad (42)$$

A la hora de realizar las medidas de bead-pull en un RFQ, como estas cavidades tienen 4 lóbulos, la perla se debe hacer pasar secuencialmente por cada uno de ellos. En el caso de este trabajo, en el que se hicieron medidas con un prototipo de RFQ, cada vez que se mide un lóbulo hay que cambiar todo el sistema de poleas para hacerlo pasar por el siguiente lóbulo. Para ello, se debe ajustar el cable de nylon ajustándolo con pequeñas poleas y así conseguir que la perla pase de manera adecuada por el lóbulo como puede verse en la siguiente imagen, a una determinada distancia del eje longitudinal del RFQ.

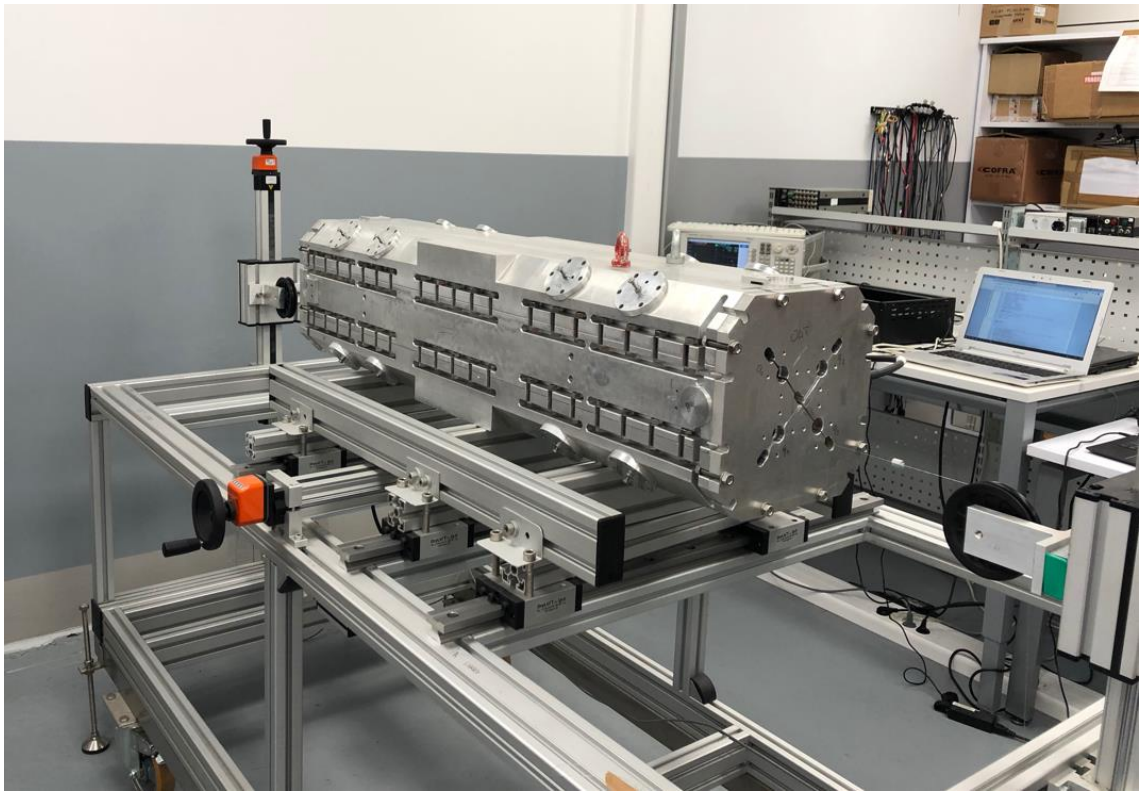

Figura 20. Sistema de medidas bead-pull

Para poder apreciar el sistema de medidas que se ha utilizado en este trabajo, se van a ver una serie de fotos en las que queda más claro el funcionamiento del bead pull. En la Figura 20, se puede ver todo el banco de medidas completo, con la perla entrando en uno de los cuadrantes, para ello, el cable de nylon se desplaza gracias al motor de la Figura 21.

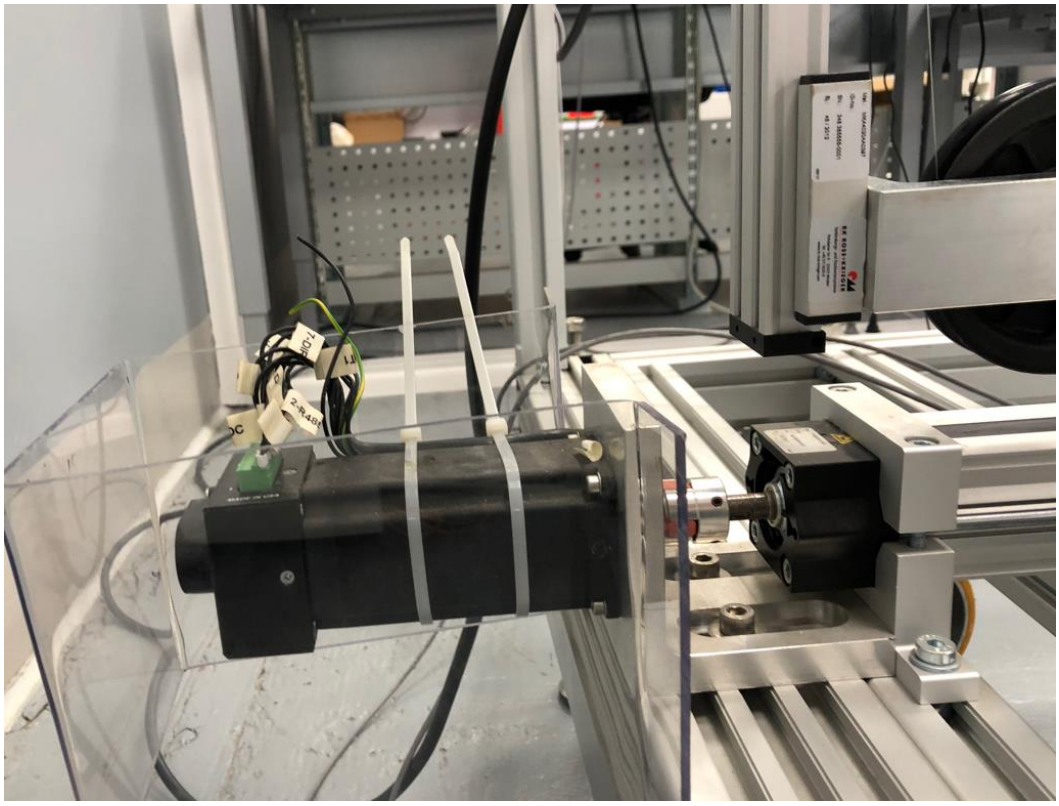

Figura 21. Motor que desplaza la perla

Este motor se encarga de desplazar el cable de nylon el cual está enganchado a dos garfios que permiten el movimiento de la perla gracias al soporte que se muestra en la Figura 22.

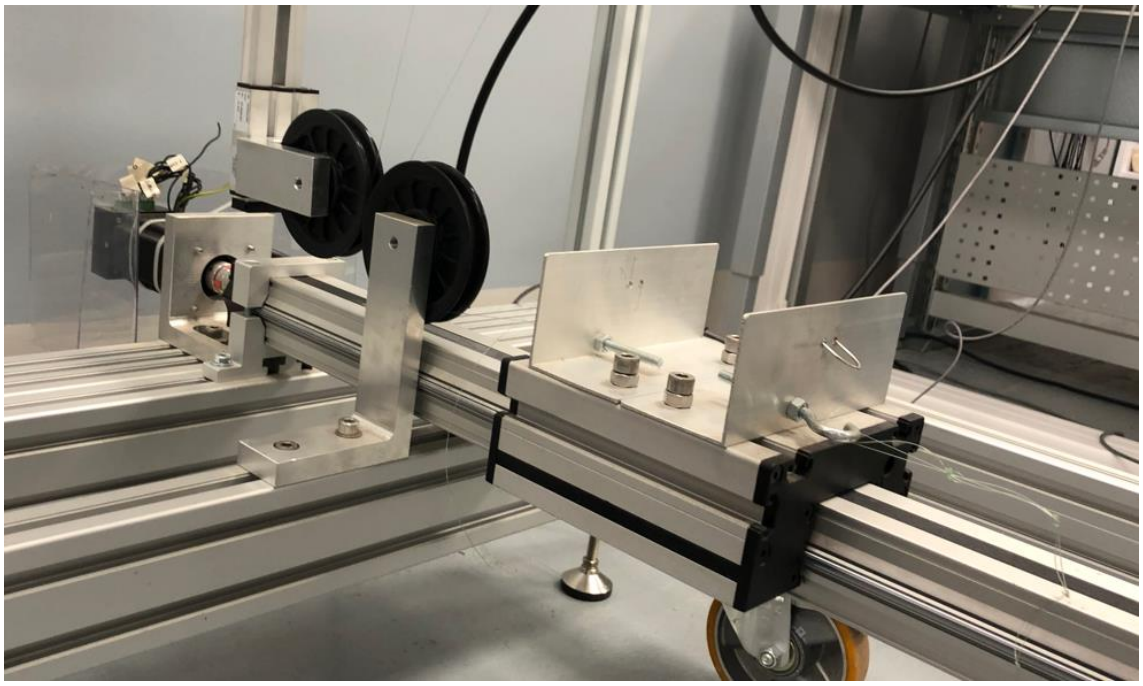

Figura 22. Sistema para desplazar la perla

Cuando la perla entra en la cavidad el campo se va eliminando con el movimiento de la perla, ya que esta ocupa un volumen, ya que la energía está almacenada en el campo, al entrar la perla en la cavidad hace que la energía de la misma disminuya. Tomando de referencia la teoría de Slater, donde dice que, para pequeñas perturbaciones, el cambio relativo de energía es igual al cambio de frecuencia o de fase.

$$\frac{\Delta W}{W} = \frac{\Delta \omega}{\omega} = \frac{\Delta \emptyset}{\emptyset} \quad , (43)$$

En la Figura 23, puede apreciarse como es el cambio de frecuencia y de fase causado por una perla.

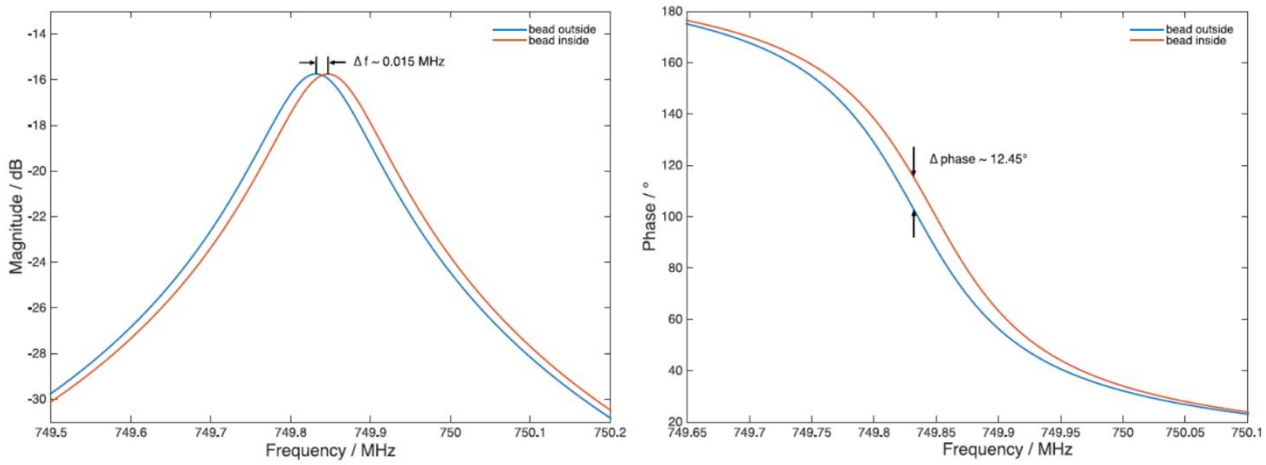

Figura 23. Cambio de frecuencia y de fase causado por una perla

La medida o bien del desplazamiento de frecuencia o del cambio de fase con bead-pull es proporcional al cuadrado de los campos y depende también de la posición de la perla. Mientras que la perla recorre la cavidad con una velocidad constante, es muy sencillo medir el cambio de fase con un analizador de redes (VNA), que será el instrumento de medida que se utilice en este trabajo. La posición instantánea de la perla viene dada por la velocidad de la misma y la medida del tiempo. El eje temporal puede verse como los puntos de muestreo del VNA y es proporcional al eje de la posición longitudinal. En la Figura 24, puede verse una medida del cambio de fase por uno de los lóbulos de un RFQ en función de la posición de la perla.

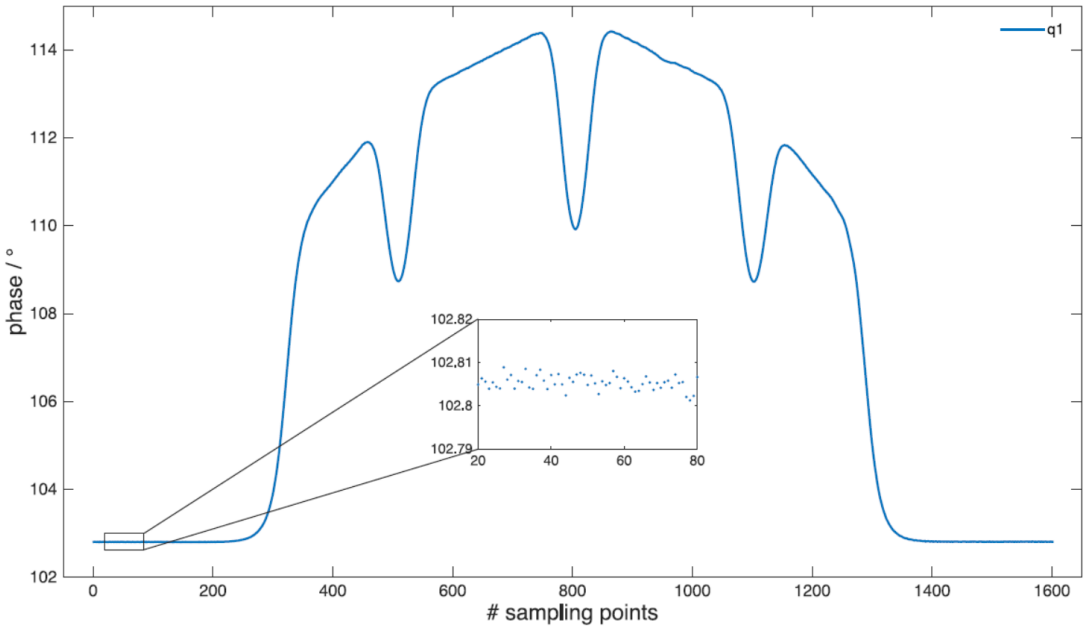

Figura 24. Ejemplo de medida bead-pull de uno de los lóbulos de un RFQ

Capítulo 5: Desarrollo del software de medida automatizada bead-pull.

En este quinto capítulo, se va a describir el proceso seguido para la elaboración del software de medida automática mediante la técnica de bead pull y de la interfaz gráfica de usuario, explicando el código y las funciones que tiene. Todo el código fuente desarrollado en este proyecto se presenta en el Anexo. Una vez explicada la interfaz de usuario, se explica el proceso de medida y el procesado de los resultados, exponiendo los resultados de una medida realizada en el laboratorio de ESS Bilbao.

Este proyecto se ha realizado con el lenguaje de programación Python y concretamente con el entorno de desarrollo integrado Jupiter Notebook, que forma parte del paquete Anaconda.

Como explicación general, este programa configura un analizador vectorial de redes y así mismo comanda un motor paso a paso el cual es el encargado de tirar de la perla para poder realizar las medidas a lo largo de la cavidad. Cuando el usuario necesita empezar la medida, hace click en el botón y el programa sincroniza la medida del VNA con la velocidad del motor para obtener la medida de la diferencia de fase en función de la posición longitudinal instantánea de la perla.

5.1 Desarrollo de la interfaz de usuario

En este primer apartado se describe la interfaz de usuario, lo que se busca a la hora de desarrollar una interfaz de este tipo es que sea clara y sencilla para que luego cualquier usuario que necesite hacer uso de ella pueda entenderlo con facilidad. El resultado final de la interfaz de usuario se ve en la siguiente figura:

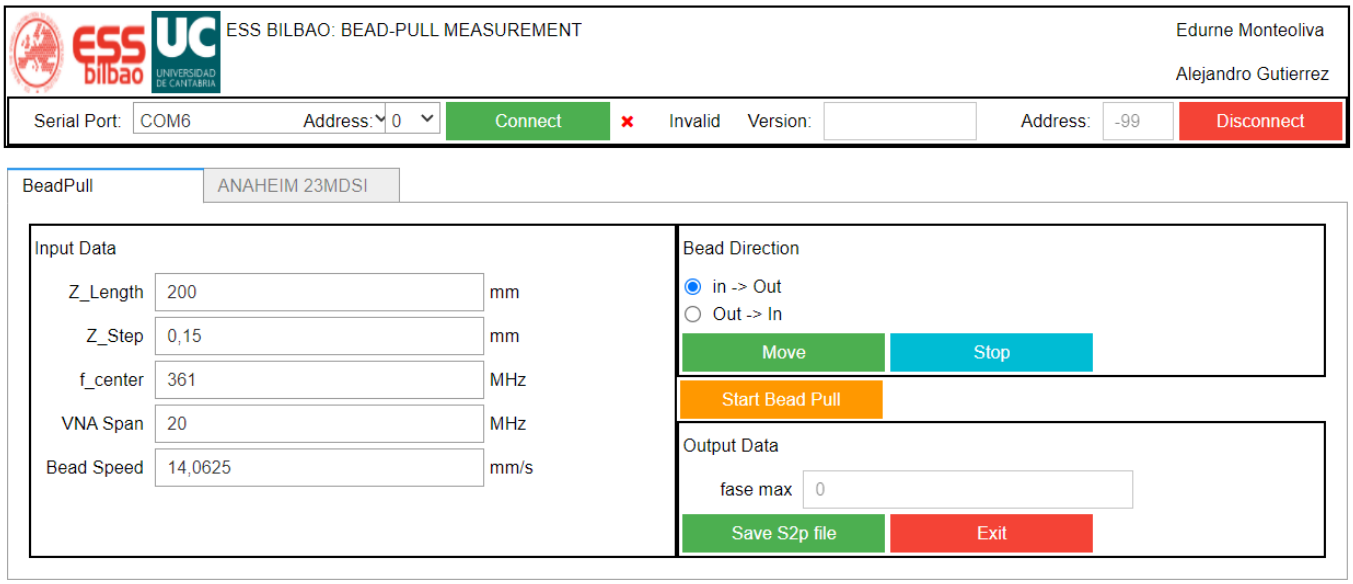

Figura 25. Interfaz gráfica de usuario para medidas bead-pull

Como puede apreciarse en la Figura 25, existen tres partes muy diferenciadas en la propia interfaz. La primera de ellas es en la que simplemente viene tanto el título, como los nombres de los autores y los logos de la Universidad de Cantabria y de ESS Bilbao. Además, esa parte de la interfaz de usuario, proporciona información acerca de la conexión con el controlador del motor a través de un puerto serie RS-485. El puerto serie, debe ser seleccionado por el usuario. El propio código proporciona una utilidad para explorar los puertos serie e identificar aquél en el que está conectado el motor. Una vez que esté todo en orden, se pulsa el botón de conectar, que indicará si el motor está correctamente conectado o no. También existe otro botón para finalizar la sesión con el motor.

La segunda parte de la interfaz es la de los datos de entrada, que es donde el usuario introducirá los principales parámetros de configuración de la medida. Estos datos son los siguientes:

- Z_Length: Longitud de la cavidad que se quiera medir, como indica en la interfaz puesta en milímetros.
- Z_Step: Paso de muestreo longitudinal, en milímetros. El analizador de redes (VNA) tomará medidas de desplazamiento de fase con un período de muestreo (en mseg) sincronizado con la velocidad del motor (en mm/seg), para satisfacer el Z_Step requerido.
- f_center: Esta será la frecuencia central preliminar que seleccionará el analizador de redes para la medida. Se indica en MHz.
- VNA Span: Una vez que se ha seleccionado una frecuencia central, hay que elegir también el rango de frecuencias de medida del VNA.
- Bead Speed: Este parámetro no es configurable, ya que viene dado por la velocidad de giro del motor y el paso de rosca de la varilla roscada que tira de la perla; en esta caja se presenta el resultado de la velocidad de la perla calculado.

Por último, en la parte derecha, seleciona tanto la dirección del movimiento de la perla, como los botones para poner en funcionamiento y parar el motor. En este caso hay 3 botones diferentes. El primero de ellos, botón de “Move” es para arrancar el motor y posicionar la perla en la posición deseada, para lo cual hay que pararlo utilizando el botón de “Stop”. El tercer botón “Start Bead Pull” es el botón que hace uso de los valores de entrada e indica al motor el tiempo que tiene que estar en funcionamiento para recorrer toda la cavidad y tomar las medias de forma sincronizada. Finalmente, en la parte inferior se presentan los resultados de la diferencia de fase máxima obtenida, la cual da una idea de que la medida ha salido correctamente. Además, existe un botón para guardar los resultados en un fichero de texto con formato S2P y un botón para salir de la interfaz de usuario.

5.2 Funcionamiento de la interfaz de usuario

Una vez que se conoce cómo es la interfaz de usuario, el siguiente paso es saber cómo funciona. Para ello se debe conocer también el motor con el que se ha trabajado en este proyecto, el cual es el 23MDSI Series Stepper Motor de Anaheim Automation. Es un motor paso a paso con controlador integrado que permite control mediante puerto serie y la configuración de diferentes velocidades de funcionamiento, aceleración, etc.

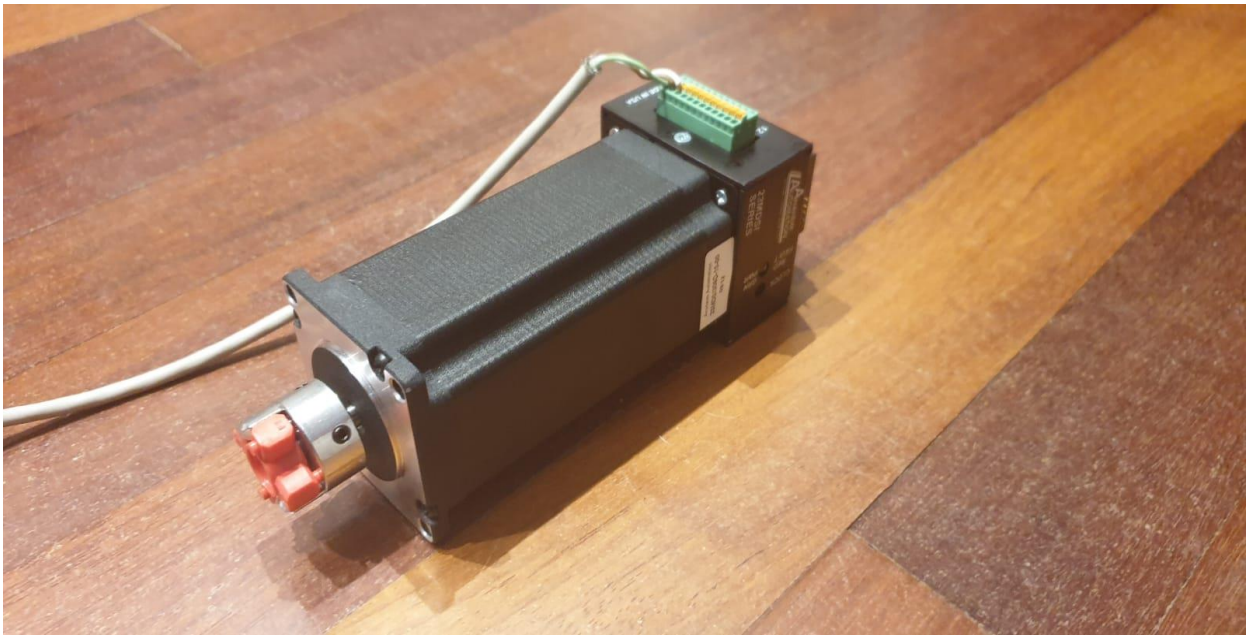

Figura 26. Motor 23MDSI paso a paso

Este motor se conecta por el puerto serie a través del protocolo RS485 y puede alimentarse o bien con 12 o 24 voltios. Una vez que el motor está alimentado y sabemos que el ordenador lo ha reconocido, debemos pasar a configurar el motor. Para este motor se creó una interfaz de usuario propio, la cual permite que se configure el motor como se necesite para la aplicación requerida. La interfaz de usuario es la que se ve en la Figura 27.

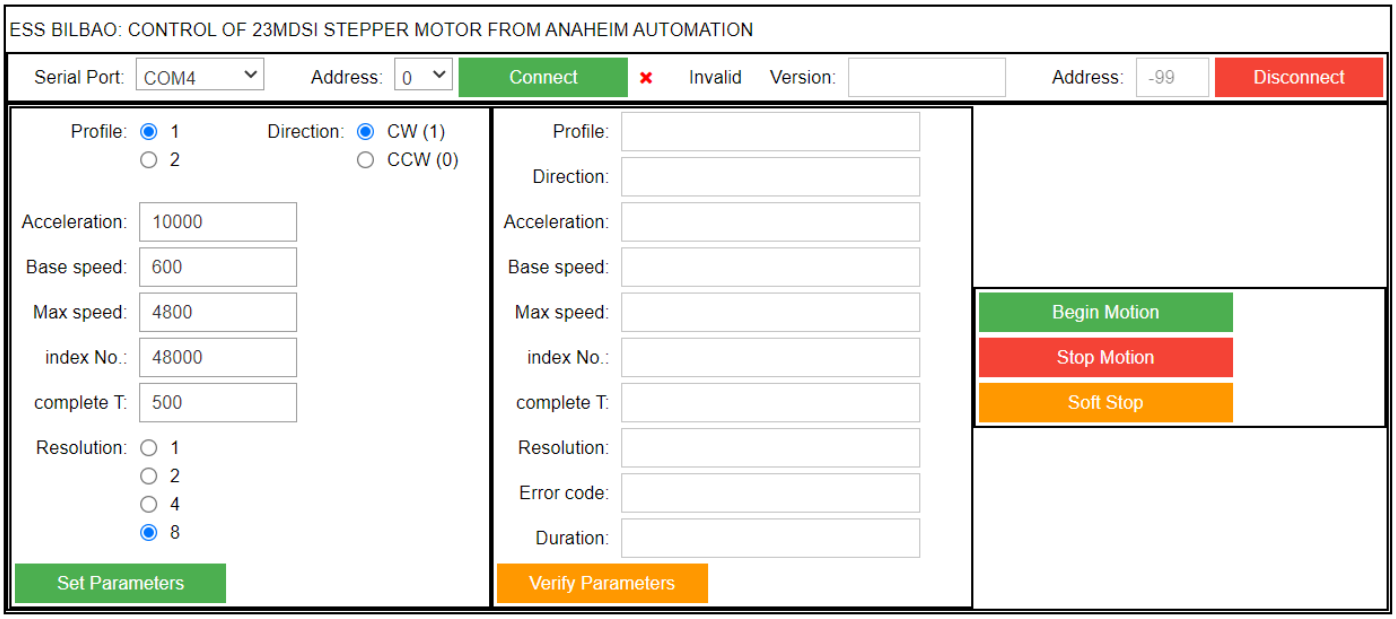

Figura 27. Interfaz de usuario para configurar el motor

Como puede verse en la Figura 27, la primera parte es la misma, ya que es la de conexión con el motor, después lo que se tiene son las características del motor. Para las medidas que se han realizado del bead-pull, haciendo numerosas pruebas se determinó una velocidad óptima para realizar las medidas, teniendo en cuenta la velocidad de la perla, la operación del analizador de redes y la varilla roscada. Como se veía en la Figura 25, la velocidad de la perla era de 14.0625 mm/s. Este valor se obtiene a partir de los datos de esta interfaz de usuario. Una vez que el motor se fija con unas características, lo guarda en su memoria no volátil y hasta que no se vuelva a cambiar sigue teniendo estas características. Este motor ofrece la posibilidad de guardar dos perfiles, gracias al radio_button de Profile, y así poder tener almacenadas dos configuraciones diferentes para realizar las medidas.

Todas estas características tienen unos valores máximos y mínimos que vienen perfectamente detallados en el código del programa para evitar configurar el motor con valores no válidos. Se pueden conseguir muchas variaciones de velocidad con la cantidad de posibilidades que ofrece este motor.

Para establecer todas las características, se crea una clase del motor en Python y así se recoge todo organizado. Esta estrategia es la que se ha seguido en este trabajo para poder definir las características del motor.

Una vez que se ha fijado la velocidad a la que trabaja el motor hay que pasar a definir los parámetros de la cavidad y por lo tanto el tiempo que tiene que estar el motor en funcionamiento para que la perla pase por toda la cavidad.

Lo primero que se debe hacer es comprobar que el motor se conecta correctamente, cuando esto ocurre, aparece un tick verde al lado del botón de color verde “Connect”, como se ve en la Figura 28.

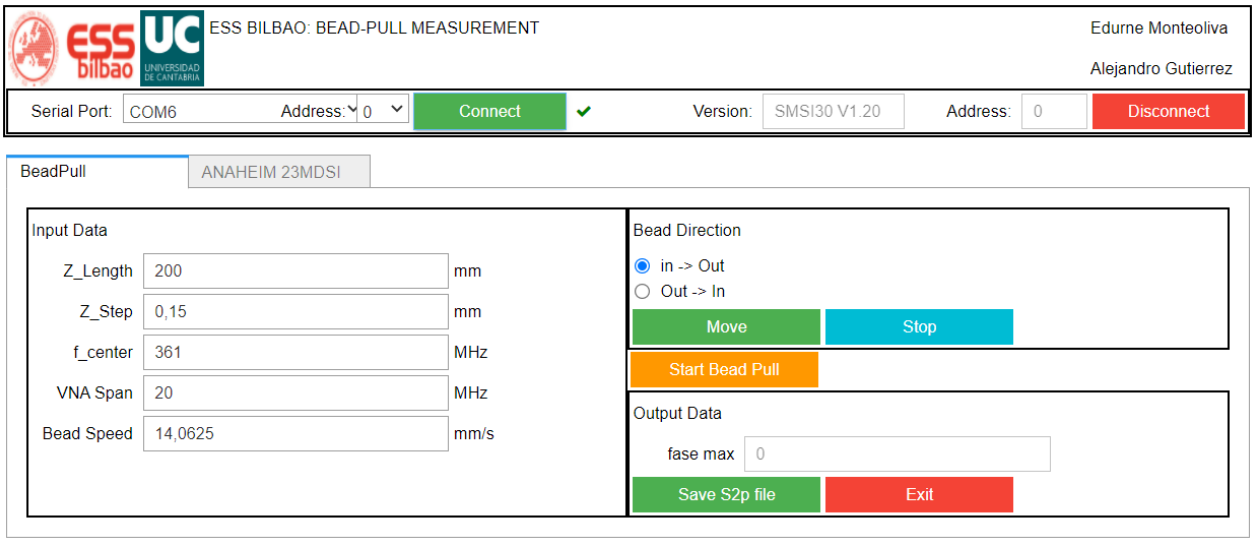

Figura 28. Interfaz gráfica con el motor conectado

Al pulsar dicho botón, el programa llama a la siguiente función:

```
#Abrir una conexion con el motor
def cb_do_connect(objt):
    global ser, motor1
    ser = serial.Serial(port=slct_serialport.value)
    motor1 = M23MDSI(ser, address=slct_address.value)
    if verbose: print(motor1)
    inttxt_address.value = motor1.get_address()
    txt_version.value = motor1.get_version()
    D = 1 if rdbtn_in_out.value == 'in -> Out' else 0
    motor1.set_direction(D)
    #cb_get_parameters(obj)
    if txt_version.value != '':
        chckbx_connection.value = True
```

Figura 109. Función para la conexión del motor

Esta función lo que hace es crear un objeto de la clase del motor, en este caso ‘motor1’ que tiene como variables tanto la dirección del motor como el puerto serie al que está conectado. Para hacer la llamada a una función desde cualquier botón se utiliza el comando que se ve en la Figura 30.

```
btn_connect.on_click(cb_do_connect)
```

Figura 110. Comando para función de botones

En términos generales, cuando se define un botón, además de su aspecto visual, se debe declarar su funcionalidad, es decir, la función o “callback” a la que se llama al hacer click. Esto se implementa con el nombre del botón, acompañado del atributo ‘on_click’ y entre paréntesis la función a la que hay que hacer la llamada.

Como bien se dijo antes, tenemos dos opciones para poner en funcionamiento el motor. La primera de ellas es el botón “Move”, que puede usarse por ejemplo en el caso en que la perla se queda dentro de la cavidad y es necesario extraerla. De esta manera resulta mucho más cómodo mover la perla hacia la posición deseada. Para para el movimiento simplemente se debe pulsar el botón de “Stop”.

Es importante establecer la dirección del motor (sentido en el que va a girar y, por tanto, el sentido de movimiento de la perla). Para ello, está el radio_button de la parte superior derecha, en la caja de Bead direction, como se ve en la Figura 22. Al fijar una de las dos direcciones el programa indica al motor el sentido del giro.

Como siguiente paso importante está el botón “Start Bead Pull”, que es donde realmente hay que pulsar si queremos llevar a cabo una medida de un cuadrante del RFQ. Al pulsar este botón, el programa lee los parámetros de entrada que definen el movimiento de la perla. En concreto, la dirección la longitud de la cavidad (Cavity_Length), el paso del motor (Z_Step), la frecuencia central (f_Center_Aprox) y el rango frecuencial (Span). El programa comprueba que ninguno de estos valores sea incorrecto, en cuyo caso produce un mensaje de error y no empieza la medida. En el caso de que todo está correcto, el programa mide la verdadera frecuencia de resonancia actual del RFQ y configura el analizador de redes vectorial para realizar la medida de barrido temporal sincronizado con el arranque y la velocidad del motor (y de la perla). Esto se realiza mediante la función, “configureVNA”, cuyo código es el siguiente:

```
def Start_BeadPull(verargin):
    if 'in -> Out':
        Dir = 1
    if 'Out -> in':
        Dir = 0

    #Get params
    global Cavity_Length, Z_Step, f_Center_Aprox, Span
    Cavity_Length = txt_z_length.value
    Z_Step = txt_Z_step.value
    f_Center_Aprox = txt_f_center.value*1e6
    Span = txt_VNA_span.value*1e6

    if (Cavity_Length == 0 or Z_Step == 0 or f_Center_Aprox == 0 or Span == 0):
        print('ERROR: Input data is empty')
    else:
        v_motor = txt_Bead_Speed.value
        Span_t = Cavity_Length / v_motor
        print('Span_t = ', Span_t)

        print('Antes de entrar a configurar el VNA')
        #prueba(Span_t)
        configureVNA(Span_t)

        #Programa de matlab pasan el valor de fase max en un structure
        Delta_Phase = beadPullMeasurements(Span_t)
        txt_fase.value = round(Delta_Phase, 2)
```

Figura 31. Función Start bead-pull

A modo de comparación rápida, tras cada medida de la función beadpullMeasurement se presenta en un cuadro la diferencia de fase máxima obtenida. Este valor debe ser típicamente del orden de 10 a 25º para que los resultados sean correctos, con suficiente relación señal a ruido y dentro de los límites de "pequeña perturbación".

En la función "configurreVNA", como su propio nombre indica, lo que se realiza es la configuración necesaria para que el analizador de redes mida lo que se necesita, la función comienza haciendo la conexión remota al VNA, que en este caso se hace con la librería pyvisa, que implementa el protocolo VISA (Virtual Instrument Software Architecture). Primero, hay que saber en qué dirección se abre el canal de comunicación del VNA con el ordenador, normalmente suele ser el 16, la secuencia de comandos es la siguiente:

```
print(' CONFIGURATION OF VNA FOR BEAD PULL MEASUREMENT ')
pyvisa.ResourceManager('@py')

rm = pyvisa.ResourceManager()
print(rm)
print(rm.list_resources())

pna = rm.open_resource('GPIB0::16::INSTR')
print(pna)
```

Figura 32. Conexión con el VNA

El siguiente paso es la configuración de lo que queremos que mida en cada canal y a su vez en cada traza de cada canal. En primer lugar, se fijan los valores de la frecuencia central y del span, los cuales han sido enviados desde la propia interfaz de usuario y se fijan valores de ancho de banda de resolución, tipo de barrido que en este caso será continuo y número de puntos. Seguidamente, se definen la primera y segunda traza del canal 1, serán para medir el S11 y el S21 respectivamente como se ve en la Figura 33.

```
#channel 1 trace 2
pna.write('DISP:WIND:STAte OFF')
pna.write('DISP:WIND1:STAte ON')
pna.write('CALC:PAR:DEF "DBS11_f" ,S11')
pna.write('CALC:PAR:DEF "DBS21_f" ,S21')
pna.write('DISP:WIND1:TRAC1:FEED "DBS11_f"')
pna.write('DISP:WIND1:TRAC2:FEED "DBS21_f"')
```

Figura 33. Definición de las trazas

Como el barrido que se eligió anteriormente era un barrido continuo, al ejecutar estas instrucciones ya aparecerán tanto el S11 como el S21 en la pantalla del VNA. Una vez que ya están en la pantalla seleccionamos un marcador que se sitúe en el máximo del S_{21}, para ello, hay que seleccionar primero la traza y ya operar sobre la misma. Se guardará el valor del marcador y se presentará por pantalla como f_0, ya que será la frecuencia de resonancia de la cavidad.

```
pna.write('CALC:PAR:SEL "DBS21_f"')
pna.write('CALC:CORR:STATe OFF')
pna.write('CALC:FORM MLOG')
sleep(2)
pna.write('CALC:MARK:STAT ON')
# SLEEP PARA QUE LE DE TIEMPO A REALIZAR LA MEDIDA
sleep(1)
pna.write('CALC:MARK1:FUNC:EXEC MAX')
#Se acopla el primer market
pna.write('CALC:MARK1:COUP ON')
#f0 = eval(pna.query('CALC1:MARK1:X?'))
f0 = pna.query('CALC1:MARK1:X?')
print('f0 = ', f0)
```

Figura 34. Obtener frecuencia de resonancia de la cavidad

El siguiente paso es configurar la traza 3 del primer canal, en la cual se va a seleccionar la fase del S21 y se evaluará la fase del marcador para luego ser comparada con otra.

Para finalizar con la configuración, hay que seleccionar la traza número 4 del canal 2. Esta traza será con la que se realice la medida de fase con la perla en movimiento. Para ello, de igual manera que se hizo con la traza anterior, seleccionamos la fase del S_{21} ya que es el parámetro que interesa medir. En este caso el tipo de barrido que se seleccionará es que haga un solo barrido y que sea en el tiempo, ya que simplemente queremos que haga la medida mientras la perla recorre la cavidad y que se mantenga esa medida en pantalla. Por último, se pondrá un marcador del que se evaluará la fase sin perturbación en el dominio del tiempo y se comparará con la obtenida en el dominio de la frecuencia.

Esto es una simple comprobación ya que si el error de fase es mayor que 1º es debido a algún error o cambio de condiciones ambientales, y debe presentarse un aviso.

Por último, se deben fijar el número de puntos y el rango temporal (Span_t) que se necesita. El número de puntos se calcula dividiendo la longitud de la cavidad entre el paso elegido.

```
Npoints_t = Cavity_Length/Z_Step
```

Figura 35. Cálculo del número de puntos

Este número de puntos junto con el Span_t que se calculó previamente es lo que se debe de establecer como parámetros para el segundo barrido, ya que el Span_t será el tiempo que tiene que estar el motor tirando de la perla para poder hacer el barrido completo y realizar las medidas del VNA. Las instrucciones para ellos son las siguientes:

```
pna.write('SENS2:SWE:POIN ', str(Npoints_t))
pna.write('SENS2:SWE:TIME ', str(Span_t))
```

Figura 36. Definición del número de puntos y del tiempo de barrido

Una vez que se ha completado la configuración del VNA hay que dar paso a la llamada de las medidas, en este caso la función es "beadpullMeasurement", en la que lo que se hace es el barrido que se configuró previamente, teniendo el número de puntos igual a Npoints_t y durante un tiempo igual a Span_t, para completar todo el barrido. Se añade una pequeña pausa (sleep), para garantizar la sincronización entre el analizador de redes y el motor, y finalmente se hace la llamada a la función de movimiento "begin_motion", durante el tiempo Span_t, para, por último, parar el motor.

```
#Para el movimineto del motor ->simple llamada a funcion
print('Dentro de las medidas antes begin_motion')
motor1.begin_motion()
print('despues de begin_motion')
sleep(Span_t)
motor1.stop_motion()
```

Figura 37. Llamadas para mover y parar el motor

Como se dijo anteriormente, el resultado que se muestra por pantalla en la interfaz de usuario es el desfase máximo que se presenta a lo largo de la cavidad, por lo que una vez completada la medida, hay que colocar un marcador en el máximo y en el mínimo de la misma y extraer tanto el valor del eje x (tiempo en que se produce) como el del eje y, valor de fase. Por último, se deben restar estos dos valores y así obtener el máximo desfase que existe en la cavidad.

```
pna.write('CALC2:PAR:SEL "ARGS21_t"')
pna.write('CALC2:MARK1:FUNC:EXEC MIN')
t_phio_min = pna.query('CALC2:MARK1:X?')
phiostr_min = pna.query('CALC2:MARK1:Y?')
ind=phiostr_min.split(',')
phio_min = float(ind[0])

pna.write('CALC2:PAR:SEL "ARGS21_t"')
pna.write('CALC2:MARK2:STAT ON')
pna.write('CALC2:MARK2:FUNC:EXEC MAX')
t_phio_max = pna.query('CALC2:MARK2:X?')
phiostr_max = pna.query('CALC2:MARK2:Y?')
ind=phiostr_max.split(',')
phio_max = float(ind[0])
print('phio_min = ', phio_min)
print('t_phio_min= ', t_phio_min)
print('phio_max = ', phio_max)
print('t_phio_max= ', t_phio_max)
inc_phase = abs(phio_max - phio_min)
print('inc_phase = ', inc_phase)
```

Figura 38. Cálculo del máximo desfase producido en la cavidad

Una vez realizada la medida, el usuario tiene la posibilidad de guardar los resultados de los parámetros S en un fichero de texto con formato .S2P, de tal manera que después puedan ser procesados fuera de la línea. El fichero se guarda con el nombre de resultados bead pull incluyendo la fecha en que las medidas han sido realizadas. En el fichero se recoge también la fecha, la frecuencia de resonancia, así como todos los parámetros S con su fase medida a la frecuencia central en el barrido temporal.

```
resultados = datetime.datetime.now().strftime("%Y_%m_%d__%H_%M_%S_ResultsBeadPull.s2p")
date = datetime.datetime.now().strftime("%Y/%m/%d--%H:%M:%S")

#fichero de escritura de resultados
fichero_resultados = open(resultados,"w")
fichero_resultados.write("!RF SYSTEMS: ESS BILBAO\n")
fichero_resultados.write("!Date: %s\n" % date)
fichero_resultados.write("!CW TIME SWEEP\n")
fichero_resultados.write("!CW FREQ: %s\n" % f0)
fichero_resultados.write("# S S DB R 50\n")
fichero_resultados.write("!\n")
fichero_resultados.write("!            S11            S21            S12            S22\n")
fichero_resultados.write("!TIME       DB       ANG       DB       ANG       DB       ANG       DB       ANG\n")

print('len(t): ',str(len(t)))
for i in range(0,len(t)):
    fichero_resultados.write("{:9.6f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}\n"
                             .format(t[i], dBS11[i], AS11[i], dBS21[i], AS21[i], dBS12[i], AS12[i], dBS22[i], AS22[i]))

fichero_resultados.close()
```

Figura 39. Comandos para guardar fichero

Para ver un ejemplo de cómo se recogen las medidas en este programa, en la Figura 40 se ve un ejemplo de uno de los cuadrantes del prototipo de RFQ de ESS Bilbao, La primera columna es el tiempo en el que va recogiendo cada punto de medida, esto habrá que tenerlo en cuenta a la hora del procesado.

```
!RF SYSTEMS: ESS BILBAO
!Date: 2021/06/09--15:28:18
!CW TIME SWEEP
!CW FREQ: +3.56765000000E+008
# S S DB R 50
!
!            S11            S21            S12            S22
!TIME       DB       ANG       DB       ANG       DB       ANG       DB       ANG
 0.000000  -200.000   45.000  -15.138  -177.561  -200.000   45.000  -200.000   45.000
 0.007112  -200.000   45.000  -15.136  -177.550  -200.000   45.000  -200.000   45.000
 0.014223  -200.000   45.000  -15.137  -177.556  -200.000   45.000  -200.000   45.000
 0.021335  -200.000   45.000  -15.137  -177.562  -200.000   45.000  -200.000   45.000
 0.028447  -200.000   45.000  -15.137  -177.553  -200.000   45.000  -200.000   45.000
 0.035558  -200.000   45.000  -15.138  -177.564  -200.000   45.000  -200.000   45.000
 0.042670  -200.000   45.000  -15.136  -177.548  -200.000   45.000  -200.000   45.000
 0.049782  -200.000   45.000  -15.137  -177.553  -200.000   45.000  -200.000   45.000
 0.056893  -200.000   45.000  -15.136  -177.553  -200.000   45.000  -200.000   45.000
 0.064005  -200.000   45.000  -15.136  -177.562  -200.000   45.000  -200.000   45.000
 0.071117  -200.000   45.000  -15.135  -177.563  -200.000   45.000  -200.000   45.000
 0.078228  -200.000   45.000  -15.136  -177.556  -200.000   45.000  -200.000   45.000
 0.085340  -200.000   45.000  -15.137  -177.551  -200.000   45.000  -200.000   45.000
 0.092452  -200.000   45.000  -15.138  -177.557  -200.000   45.000  -200.000   45.000
 0.099563  -200.000   45.000  -15.137  -177.560  -200.000   45.000  -200.000   45.000
 0.106675  -200.000   45.000  -15.137  -177.557  -200.000   45.000  -200.000   45.000
 0.113787  -200.000   45.000  -15.137  -177.557  -200.000   45.000  -200.000   45.000
 0.120898  -200.000   45.000  -15.136  -177.560  -200.000   45.000  -200.000   45.000
 0.128010  -200.000   45.000  -15.137  -177.565  -200.000   45.000  -200.000   45.000
 0.135122  -200.000   45.000  -15.137  -177.556  -200.000   45.000  -200.000   45.000
```

Figura 40. Ejemplo de fichero de resultados de medidas bead-pull

Como puede verse en la Figura 40 en cada fichero de resultados habrá tantas filas como número de puntos se hubiesen obtenido en el programa, yendo desde los 0 segundos hasta el tiempo final de la medida en función de la longitud de la cavidad resonante.

Nótese que los parámetros S11, S12 y S22 indican -200 dB y 45º. Estos valores no son significativos, sino que son los valores por defecto con que se rellena un fichero S2P cuando no existe una medida del parámetro. Los únicos valores significativos son los del parámetro S21, con su magnitud en dB y su fase en grados.

Finalmente, una vez recogidas las medidas, se deben procesar los resultados. Para tener una medida completa del bead pull, hay que realizar cuatro veces la medida, una por cada lóbulo del RFQ. Cada uno de los cuatro segmentos del RFQ de ESS Bilbao, de unos 80 centímetros cada uno, incorpora 4 sintonizadores por cada cuadrante, lo que hace un total de 16 sintonizadores por segmento (64 en total para el RFQ de 3.2 m). Estos sintonizadores se introducen una determinada profundidad, variando el espacio del interior, y afectado a la frecuencia de resonancia y al perfil de campo acelerante a lo largo de la cavidad. Lo que se busca en estas cavidades RFQ es que, sin necesidad de imanes,

sea posible focalizar el haz además de guiarlo y acelerarlo. Para eso es necesario cumplir con unas estrictas tolerancias de mecanizado de los cuatro cuadrantes y de ensamblado y alineación. Para corregir las inevitables desviaciones mecánicas se utilizan los sintonizadores mecánicos. El problema es determinar la posición óptima de todos los sintonizadores para conseguir que el campor eléctrico acelerante en el eje del RFQ sea el deseado. Para ello se debe caracterizar el efecto individual de cada uno de los sintonizadores por separado y llevar a cabo un proceso iterativo de medida con la técnica de bead-pull, procesado de los resultados y reajuste de la posición de los sintonizadores hasta lograr el objetivo. El complejo proceso de optimización del campo acelerante del RFQ no entra dentro del alcance de este TFM, pero sí el procesado de los resultados de medida de bead pull para alimentar al optimizador.

En primer lugar, hay que leer los ficheros de resultados .S2P que se generaron de las medidas con una función que recoge los valores de todos los parámetros S (S11, S21, S12 y S22) así como el eje temporal en que se hizo cada medida. Una vez que se tienen todos los valores, el valor que realmente interesa es el S21, por lo que en primer lugar se dibujan las medidas de los cuatro cuadrantes. Se representa en particular la fase del S21 en grados frente a la posición de la perla en el tiempo, equivalentemente, la distancia, considerando el periodo muestro del VNA y la velocidad de la perla. Hay que tener en cuenta que no todos los cuadrantes realizan la medida en el mismo sentido, ya que, al hacer primero la medida del primer cuadrante, esta se hace en la dirección del haz, entonces luego en el segundo cuadrante viaja en contra del haz y ocurre lo mismo con los cuadrantes tercero y cuarto. Estas medidas se deben alinear para eliminar varias fuentes de error, como se va a describir a continuación.

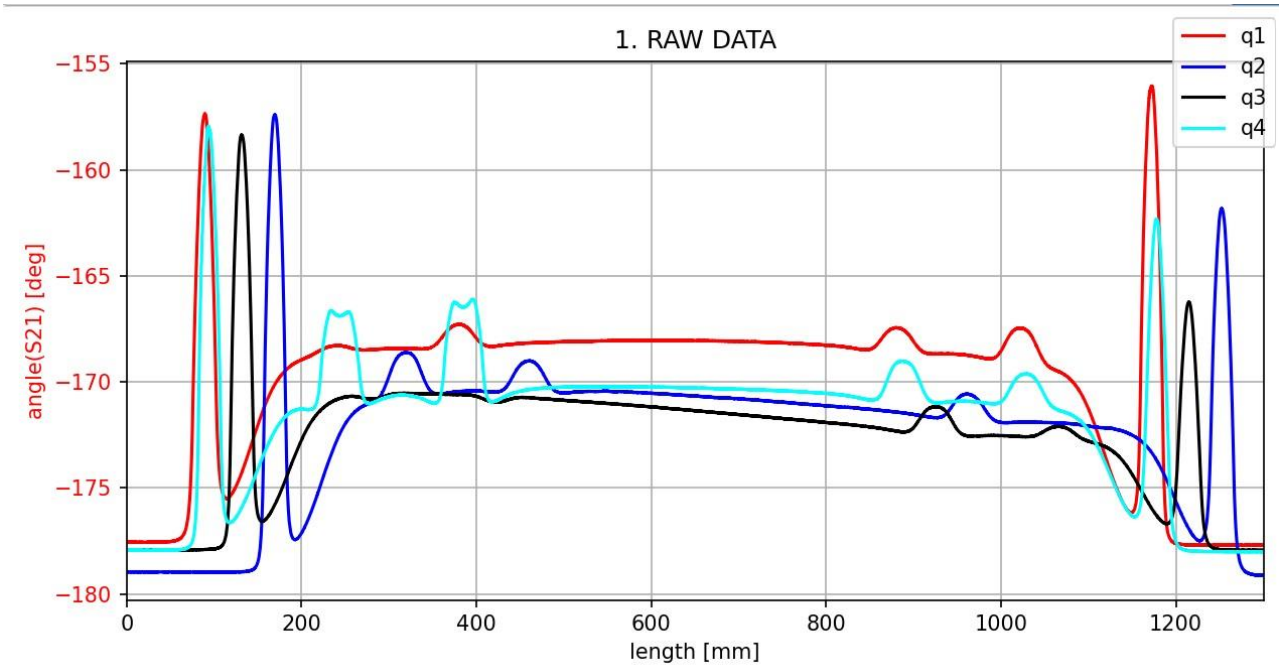

Figura 121. Medidas recogidas de los 4 cuadrantes

En la figura 41 se muestra la fase del parámetro S21 correspondiente a los 4 cuadrantes, tal cual se leen de los ficheros de resultados. Como puede apreciarse, existe un pico en cada medida de los cuadrantes tanto en la entrada como en la salida de la cavidad. Esto se debe a que, al ser un prototipo del RFQ se hicieron diferentes pruebas y una de ellas fue hacer un corte con forma de x que unía todos los cuadrantes, a fin de facilitar la operación del cambio de cuadrante de la perla. Esto fue lo que generó estos picos, que en un RFQ normal, no existirían. Como ventaja inesperada de estos picos, ocurre que se puede encontrar fácilmente el lugar en el que comienza y termina la medida ayudando al alineamiento.

Por otro lado, las pequeñas subidas que aparecen en la medida, dos al principio y dos al final, son debidas a los sintonizadores que se comentó con anterioridad. Dependiendo de la profundidad de los sintonizadores estos picos serán más o menos pronunciados.

En primer lugar, este programa hace una medida del ruido de la medida de fase. Para ello se seleccionan las primeras muestras de la medida del primer cuadrante y se le realiza un zoom.

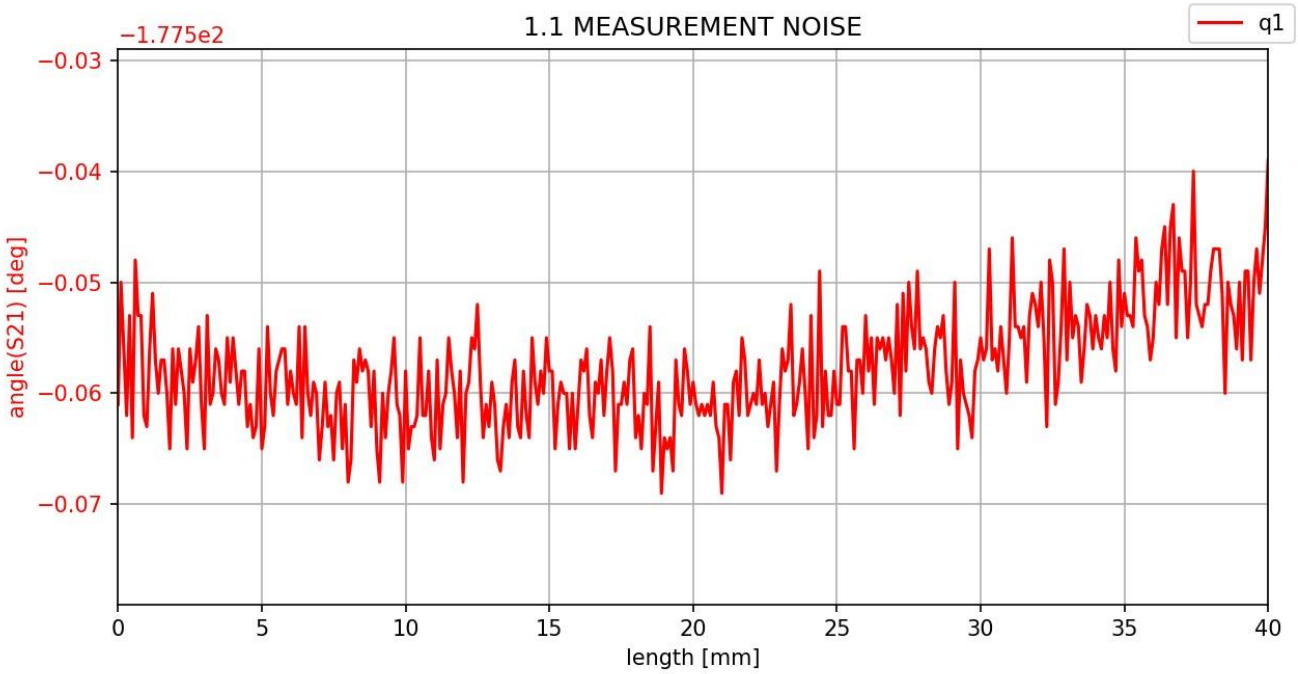

Figura 42. Ruido de fase de la medida

Como bien se ha dicho, uno de los problemas es que las medidas del primero y tercer cuadrante están realizadas en sentido contrario a los cuadrantes segundo y tercero, por lo que en primer lugar se debe invertir o 'dar la vuelta' a estas medidas, para orientar

todas de la misma forma. Para ello simplemente es necesario ordenar el vector de medidas de ambos cuadrantes a la inversa con la siguiente instrucción.

```
AS21_q1_o = AS21_q1[::-1]
AS21_q2_o = AS21_q2[::]
AS21_q3_o = AS21_q3[::-1]
AS21_q4_o = AS21_q4[::]
```

Figura 43. Comandos para invertir la orientación de la medida

El resultado de estas medidas es que se ve en la Figura 44.

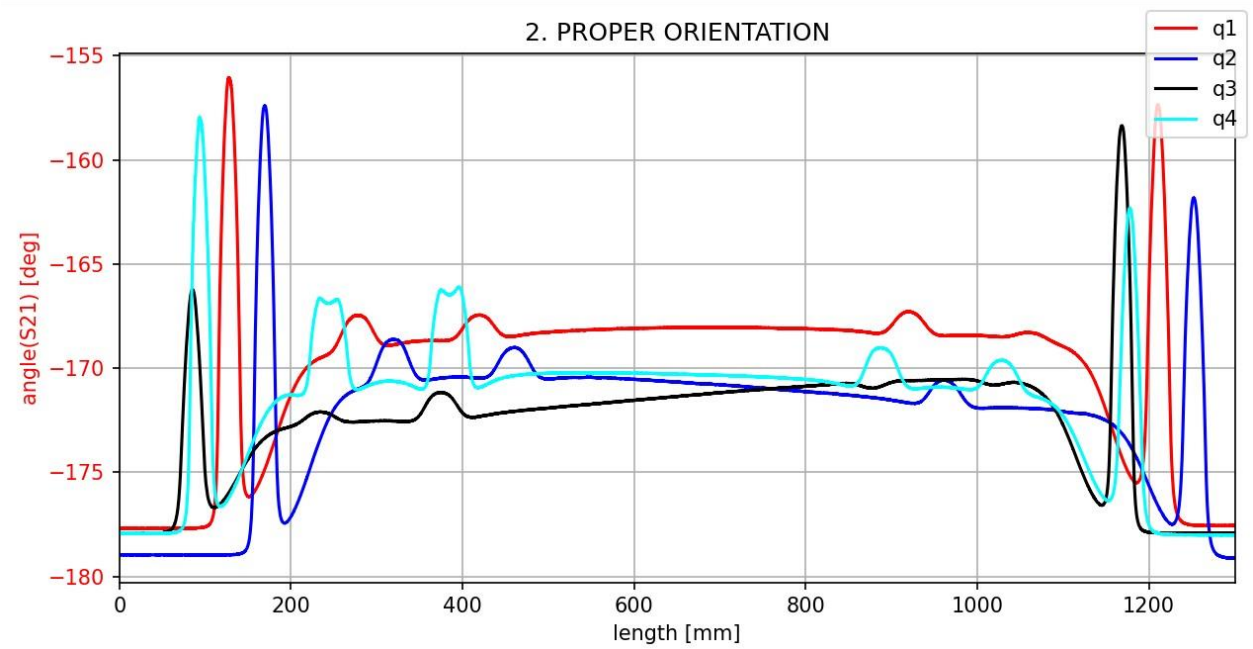

Figura 44. Orientación adecuada

Una vez que la orientación de las cuatro medidas es la correcta, el siguiente paso es alinearlas verticalmente, ya que como puede apreciarse a la izquierda de la Figura 44, no empiezan todas las medidas desde el mismo punto, sino entre 177.5 y 179º. Para ello, se calcula el promedio de 10 puntos de cada medida entre las muestras 100 y 200, y luego se resta con la original, para que todas empiecen desde el mismo punto, el código utilizado para este proceso se ve en la Figura 45 y el resultado en la Figura 46.

```python
def vertical_alignment(AS21_q1_o, x_mm):
    start1 = 100
    end1 = 200
    print(x_mm[start1:end1:10])
    print(AS21_q1_o[start1:end1:10])

    phaseRef_q1 = np.mean(AS21_q1_o[start1:end1])
    print('phaseRef (deg):  '+str(phaseRef_q1))

    AS21_q1_va = AS21_q1_o - phaseRef_q1

    return AS21_q1_va
```

Figura 45. Código para la orientación vertical

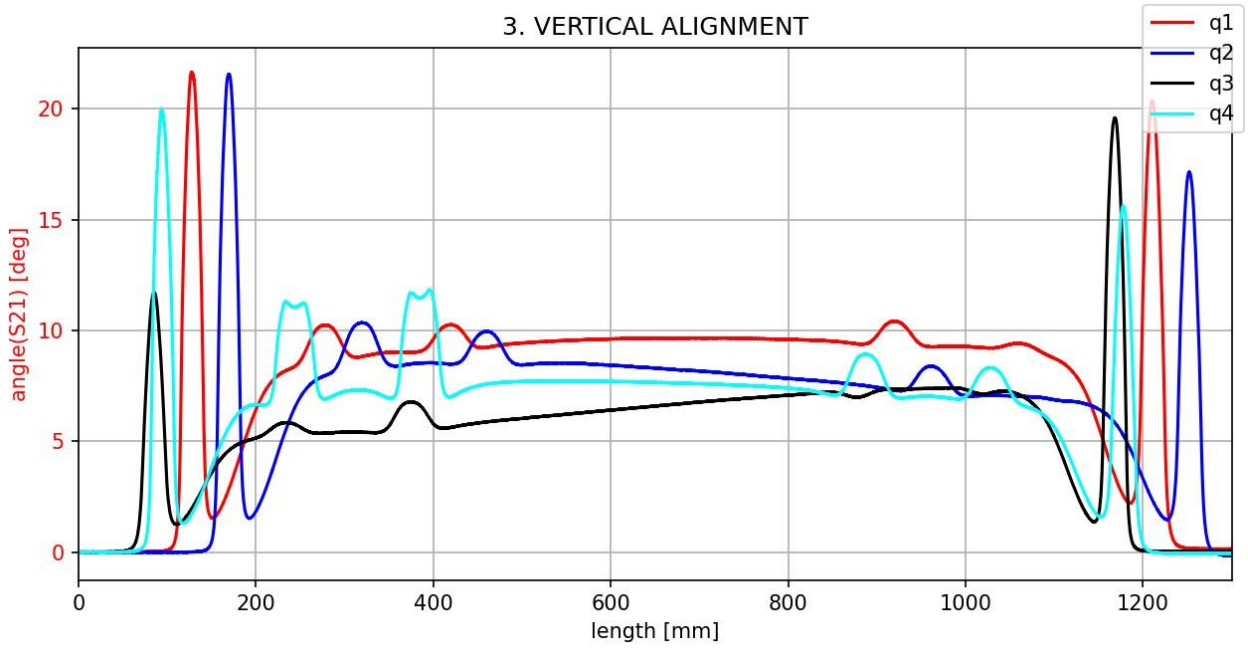

Figura 46. Orientación vertical

El siguiente paso es para compensar la desviación o deriva de la fase a lo largo de la medida. El código que se utiliza es el que se ve en la Figura 47.

```python
def phase_drift(AS21_q1_va, x_mm):
    N = len(x_mm)
    start2 = -100 #end1
    end2 = -200 #start1
    print('len x: '+str(N))
    print(x_mm[start2:end2:-10])
    print(AS21_q1_va[start2:end2:-10])

    phase2_q1 = np.mean(AS21_q1_va[start2:end2:-1])
    print('phase2 (deg):  '+str(phase2_q1))

    phaseSlope_q1 = (phase2_q1 - 0) / ( x_mm[start2-50]-x_mm[-start2+50] )
    print('phaseSlope (deg/mm):  '+str(phaseSlope_q1))

    linearDrift_q1 = phaseSlope_q1 * x_mm
    AS21_q1_pd = AS21_q1_va - linearDrift_q1

    return AS21_q1_pd
```

Figura 47. Código para la compensación de fase

En este caso se cogen 10 puntos de cada medida, pero del final de la misma, se hace la media de estos puntos y se compensa la diferencia de fase entre el comienzo y el final linealmente sobre la longitud del cuadrante de RFQ. El resultado se muestra en la Figura 48. Es necesario prestar cuidado a que la medida está bien centrada, es decir, que el punto medio del RFQ sea aproximadamente el punto medio de la medida. Para ello, la perla debe empezar su medida aproximadamente a unos 10 cm de la cavidad y acabar a la misma distancia.

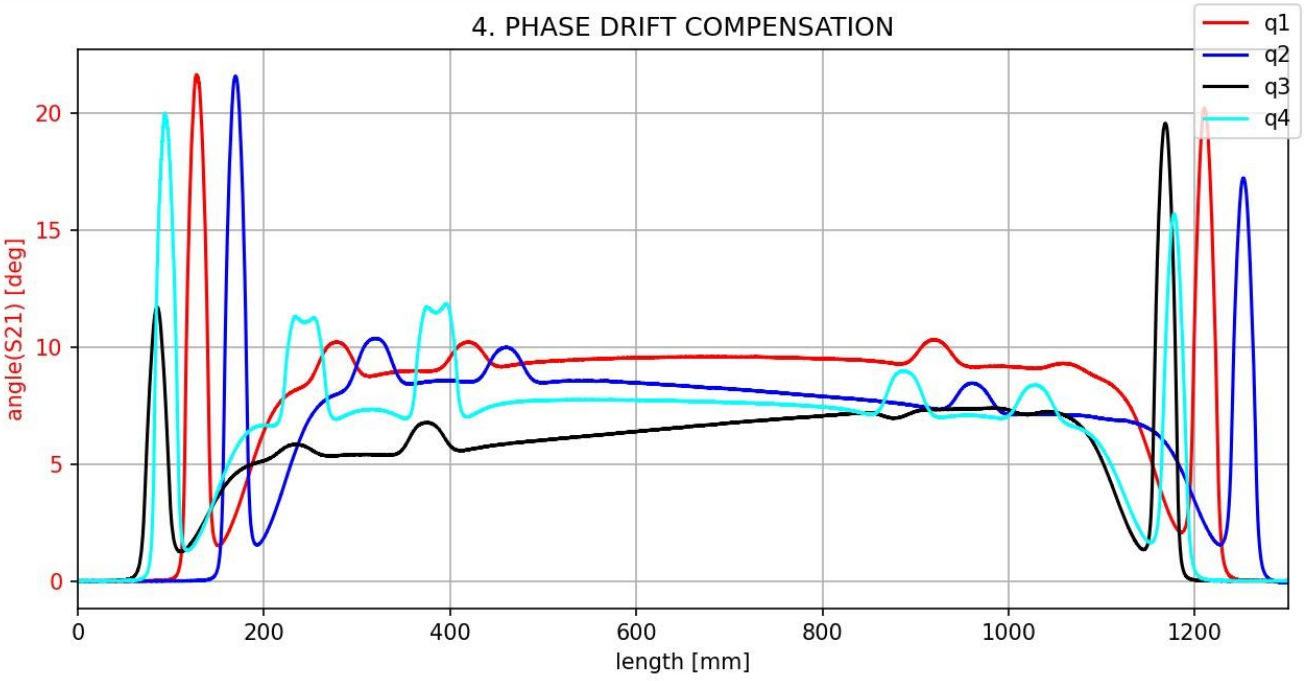

Figura 48. Compensación de fase

El siguiente paso sirve para reducir el rizado de la medida, suavizando o filtrando los resultados de medida con una media móvil. El código utilizado es el que se ve en la Figura 49 y el resultado es la Figura 50.

```
def moving_average1(a, n=3):
    # filtered array has the same length as the original
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    ma = ret[n - 1:] / float(n)
    ma = np.append(np.zeros(int((n-1)/2)), ma)
    ma = np.append(ma, np.zeros(int((n-1)/2)))
    return ma
```

Figura 49. Código para el suavizado

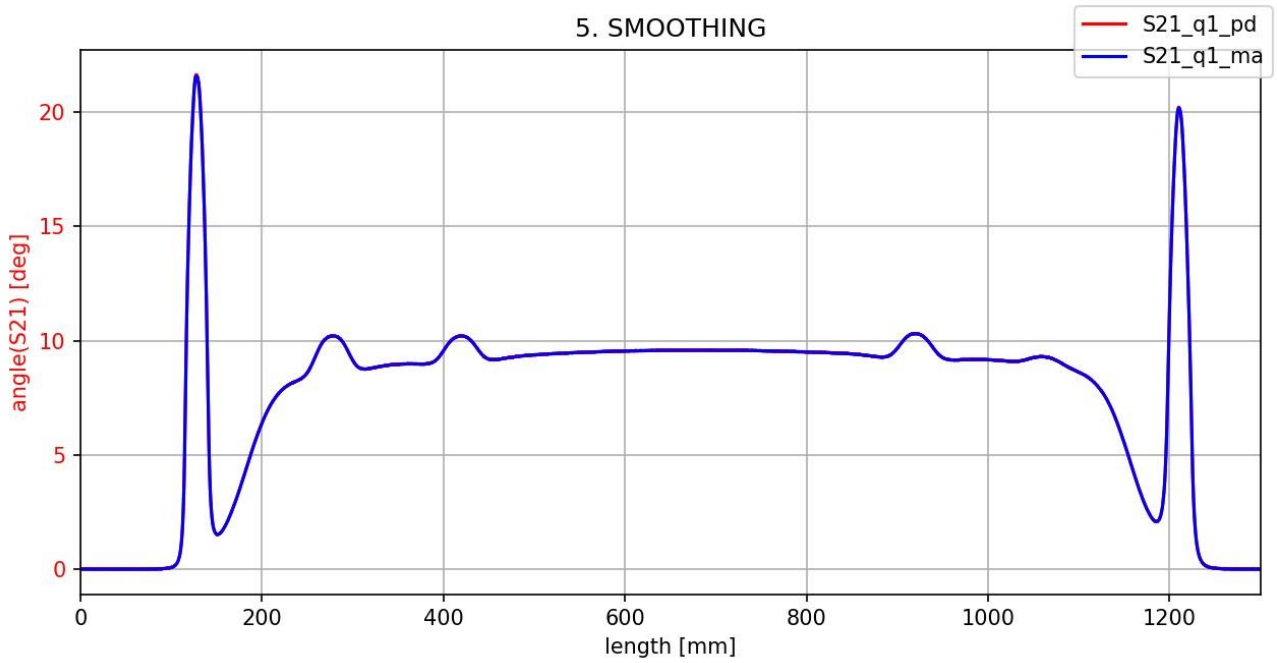

Figura 50. Suavizado de la medida

Para apreciar mejor el efecto de este suavizado, se realiza un zoom sobre esta última medida y se aprecia cómo claramente la aplicación del promediado con la media móvil resulta en unas curvas mucho más suave y no tiene tanta pequeña variación.

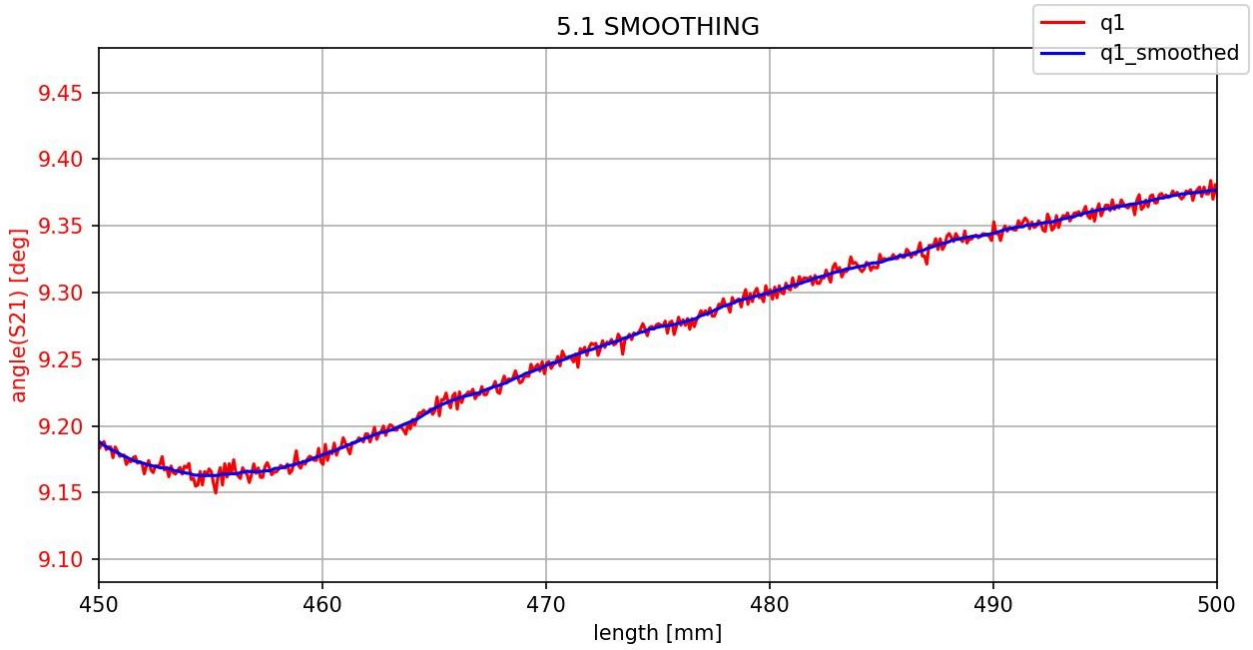

Figura 51. Zoom del suavizado

El siguiente paso es el alineado horizontal o longitudinal. Este paso sirve para "sincronizar" las medidas, esto es, hacer que las medidas de los cuatro cuadrantes empiecen desde el mismo punto de referencia. Como apreciación gráfica para el resultado que se muestra en la Figura 53, significa que todos los picos iniciales y finales coincidan, así como las pequeñas elevaciones que surgen por culpa de los sintonizadores. Para conseguir esto, el código utilizado es el que se ve en la Figura 52.

```
def horizontal_alignment(AS21_q2_ma, AS21_q1_ha, x_mm_q1, phase_threshold):

    AS21_q1_mad = np.roll(AS21_q1_ma,-1) # desplazamos los elementos del array 1 posición hacia la izquierda
    ind1 = np.where( (AS21_q1_ma <= phase_threshold) & (AS21_q1_mad > phase_threshold) )
    ind1 = int(ind1[0][0])
    print('ind1: ', str(ind1))
    AS21_q1_ma[ind1];  AS21_q1_mad[ind1]
    print(phase_threshold, AS21_q1_ma[ind1], AS21_q1_mad[ind1])
    print(len(x_mm_q1))
    print(ind1,x_mm_q1[ind1])

    AS21_q2_mad = np.roll(AS21_q2_ma,-1) # desplazamos los elementos del array 1 posición hacia la izqueierda
    ind2 = np.where( (AS21_q2_ma <= phase_threshold) & (AS21_q2_mad > phase_threshold) )
    ind2 = int(ind2[0][0])
    print('ind2: ', str(ind2))

    AS21_q2_ma[ind2];  AS21_q2_mad[ind2]
    print(AS21_q2_ma[ind2], AS21_q2_mad[ind2])
    print(ind2, x_mm[ind2])

    AS21_q2_ha = np.roll(AS21_q2_ma,-(ind2-ind1))

    return AS21_q2_ha
```

Figura 132. Código para la alineación horizontal

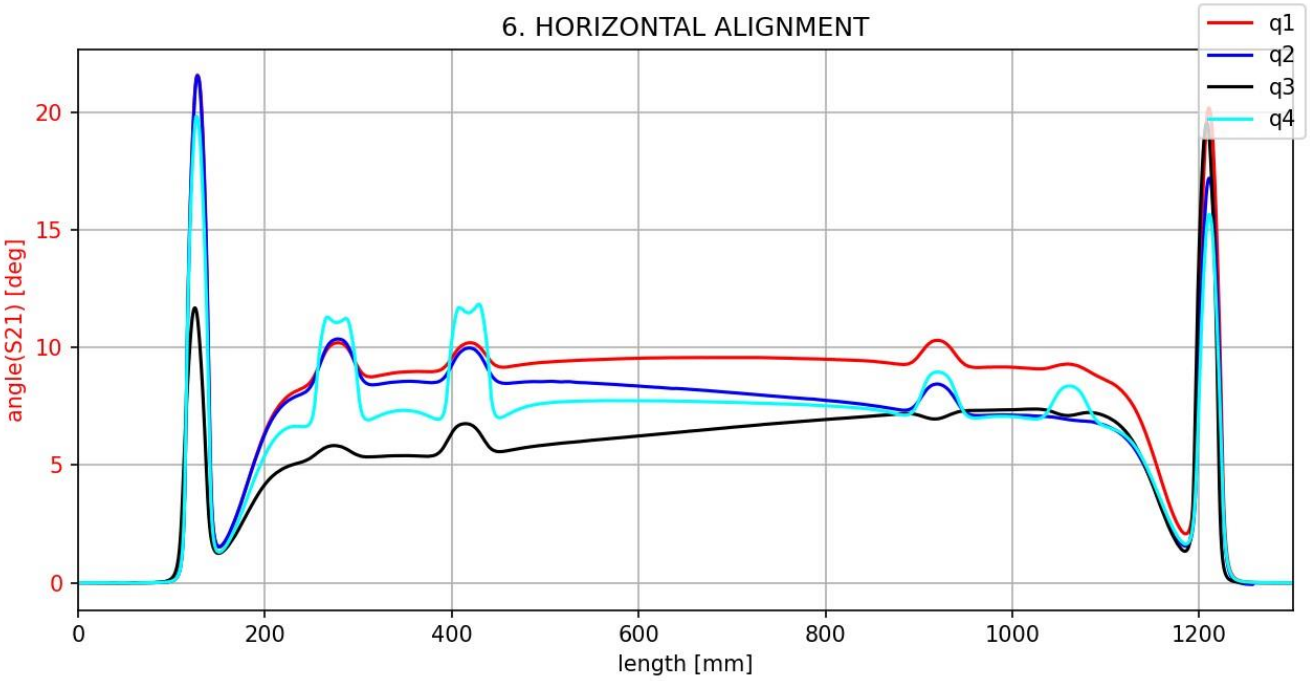

Figura 53. Alineación horizontal

De esta manera ya se ha conseguido realizar todas las alineaciones necesarias con las medidas de los cuatro cuadrantes. Una vez realizados todos estos pasos, lo siguiente que se debe hacer es la raíz cuadrada de la fase, ya que esta es proporcional al campo magnético dentro de la cavidad. Como la dirección del campo en los cuadrantes adyacentes cambia de signo para los modos cuadrupolar y dipolares, hay que aplicar el signo correcto en los cuadrupolos. Los signos que hay que aplicar son, por convenio:

- Primer cuadrante: Signo +
- Segundo cuadrante: Signo –
- Tercer cuadrante: Signo +
- Cuarto cuadrante: Signo –

Una vez aplicados estos signos y la raíz cuadrada, el resultado es el que se ve en la Figura 54.

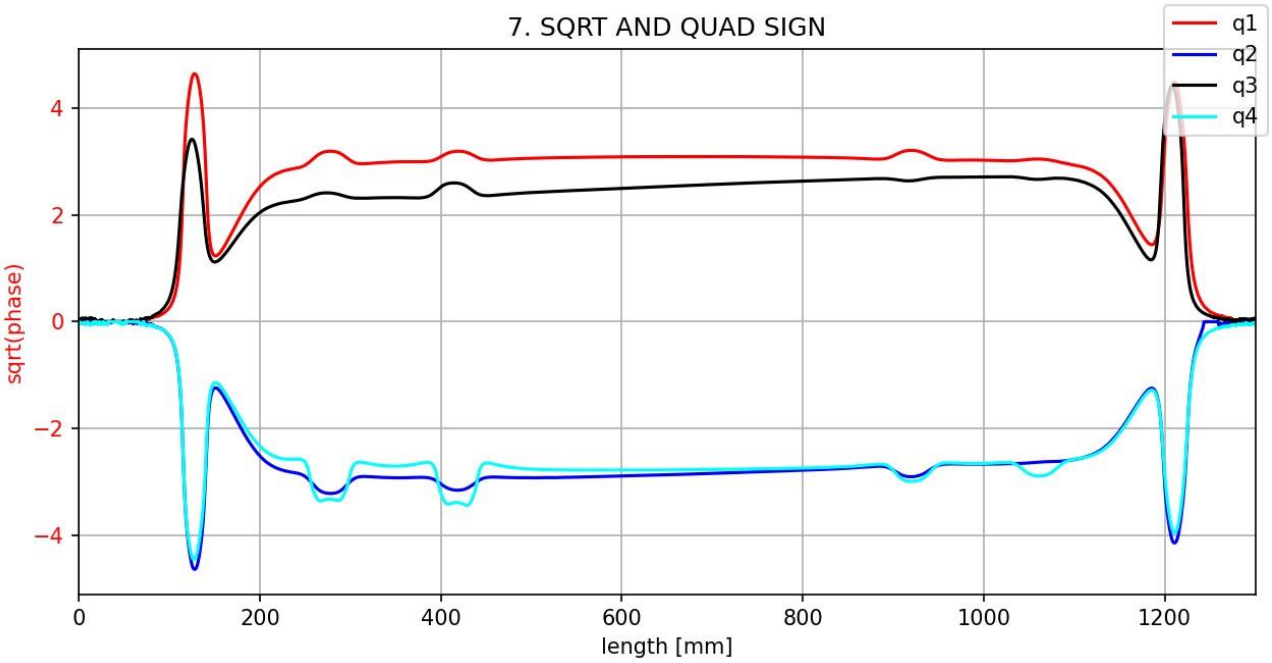

Figura 54. Raíz cuadrada y signo cuadrupolar

Por último, se pueden calcular los componentes cuadrupolar y dipolares, los cuales vienen definidos por las siguientes ecuaciones:

$$Q = \frac{(q_1 - q_2 + q_3 - q_4)}{4} \quad ,(44)$$

$$D_s = \frac{(q_1 - q_3)}{2} \quad ,(45)$$

$$D_t = \frac{(q_2 - q_4)}{2} \quad ,(46)$$

Al aplicar las ecuaciones 44, 45 y 46, los modos dipolares deberían anularse en las cavidades RFQ y únicamente tener el cuadrupolar, lo cual ser´´ia indicativo de una buena simetría de los cuadrantes. El resultado final es el que se ve en la Figura 55.

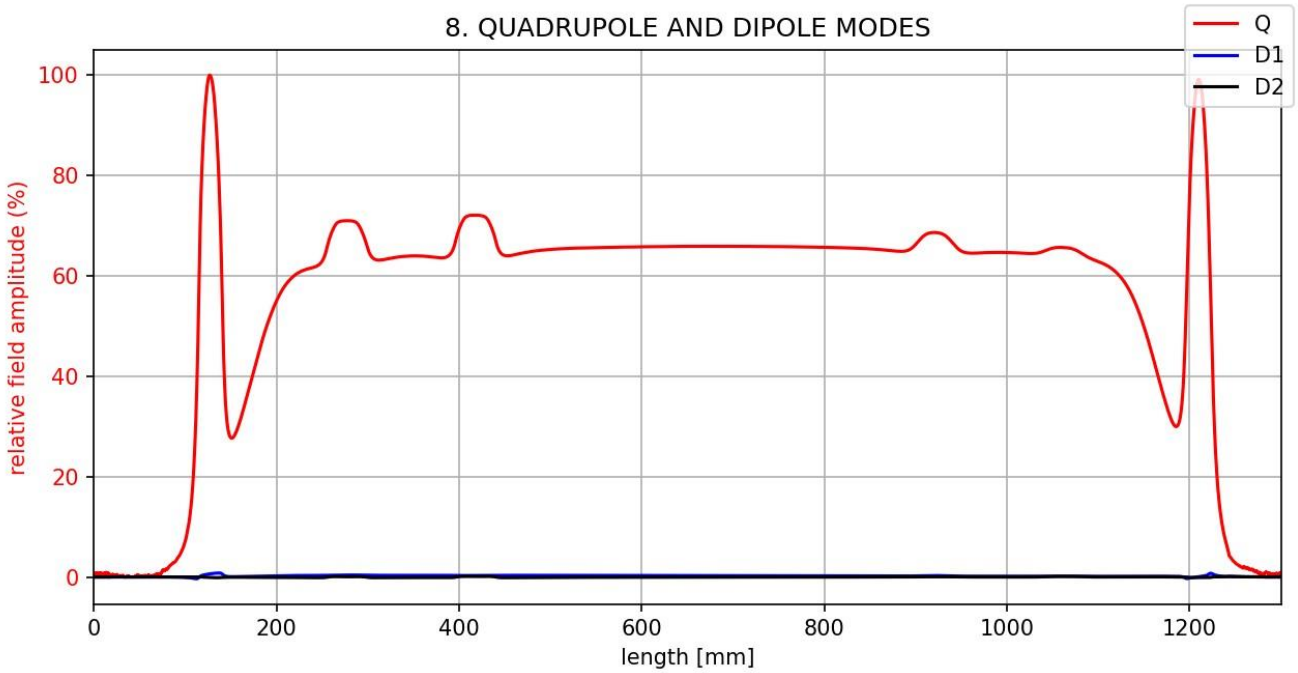

Figura 55. Modos cuadrupolar y dipolar

Una vez que se ha completado todos los alineamientos necesarios y se tiene el modo cuadrupolar, este resultado hay que introducirlo en un nuevo algoritmo de optimización iterativo, el cual será capaz de, viendo este modo, dar la posición en la que se deben colocar los sintonizadores para conseguir que el campo eléctrico acelerante que pasa por el eje de la cavidad sea el deseado, de tal manera que se optimice la aceleración de las partículas, así como su focalización y guiado. Este algoritmo ha sido desarrollado internamente por ESS Bilbao lo cual facilitará mucho las tareas de ajuste y puesta en marcha del RFQ.

Capítulo 6: Conclusiones y líneas futuras

En este Trabajo Fin de Máster se ha mostrado el proyecto realizado en colaboración con el Consorcio ESS Bilbao que trata sobre el desarrollo de un software de medida automatizada “bead pull” de cavidades RFQ para aceleradores de partículas. A su vez, como el trabajo era un proyecto dual, se desarrolló también un software de medida avanzada de factor de calidad de cavidades resonantes para aceleradores de partículas.

Se han desarrollado dos interfaces gráficas de usuario en las cuales el principal objetivo fue que se tratara de interfaces de usuario muy intuitivas y amigables, para que a la hora de que cualquier usuario tenga que hacer uso de ellas, sepa manejarlas desde el principio, sin necesidad de tratar mucho tiempo con ellas.

En cuanto al software desarrollado en este trabajo Fin de Máster, permite realizar medidas “bead pull” de tal manera que el analizador de redes se sincroniza con el motor para obtener una medida de la diferencia de fase de toda la cavidad. Gracias a la relación que existe entre la diferencia de fase con la perturbación de los campos electromagnéticos en el interior de la cavidad resonante, es posible determinar un perfil del campo acelerante. Esta información permite caracterizar las propiedades de la operación de la cavidad aceleradora, y sirve asimismo para alimentar otro algoritmo de ajuste y optimización desarrollado por ESS Bilbao, que calcula la posición en la que deben estar los sintonizadores del RFQ para maximizar tanto la aceleración como la focalización y el guiado de las partículas.

Para comprobar que funcionaba correctamente se hicieron medidas sobre el prototipo de RFQ que hay disponible en las instalaciones de ESS Bilbao. Se obtuvieron las medidas de los cuatro cuadrantes y se procesaron los resultados para identificar los perfiles de los modos cuadrupolar y dipolares, para certificar que el sistema completo de medida bead pull funcionaba de manera correcta y sincronizada, y que los resultados de las medidas tomadas tenían sentido.

En cuanto a las líneas futuras, se propone añadir en la propia interfaz de usuario la configuración del motor, es decir, que el propio usuario pueda seleccionar las características de velocidad y aceleración del motor, ya que es posible que estas medidas puedan optimizarse en términos de velocidad o de errores. Esto además ofrece una facilidad mayor al usuario, sin tener que depender de abrir otro programa para configurar el motor.

Por otro lado, el sistema de poleas actual sólo permite hacer la medida de uno de los cuadrantes. Para medir el siguiente cuadrante, se debe cambiar manualmente la posición de las poleas e introducir el cable por el siguiente cuadrante. Como línea futura, se propone diseñar un sistema que sea capaz de realizar las cuatro medidas secuencialmente, ya que no supone un problema para el motor hacer un recorrido más largo y para el usuario significaría un ahorro de tiempo muy importante y una reducción de los errores de posicionamiento mecánico. Todo ello contribuiría, con el uso del software desarrollado, a un proceso de ajuste y puesta en marcha del RFQ más sistemático y menos teioso.

En relación con el sistema de poleas que se acaba de mencionar, si esto se consiguiese, sería muy interesante añadir directamente el procesado de los resultados a la propia interfaz, de tal manera que el usuario pueda representar gráficamente el modo cuadrupolar y alimentar directamente al algoritmo de optimización para una nueva iteración de ajuste de las posiciones de los sintonizadores.

Por último, destacar que una de las mayores dificultades al desarrollar este software, fue la sincronización entre el motor y la medida del analizador vectorial de redes. Para ello, como futura tarea, se propone la implementación de un sensor de movimiento, el cual detecte el paso de la perla por la entrada y la salida de cada uno de los cuadrantes y dé la orden al analizador de redes de comenzar y finalizar la medida. De esta manera no sería necesario el cálculo del tiempo que tarda en realizar el barrido completo por la cavidad, ya que con la señal del detector empezaría y pararía la medida.

Bibliografía

[1] *ESS Bilbao – Strategic hub for neutron science and technologies*. (2021, 17 septiembre). ESS Bilbao. https://www.essbilbao.org/es/

[2] *MEBT – ESS Bilbao*. (2021, 17 septiembre). MEBT ESS Bilbao. https://www.essbilbao.org/in-kind-contribution/mebt/

[3] *RF systems – ESS Bilbao*. (2021, 17 septiembre). Cadena RF ESS Bilbao. https://www.essbilbao.org/in-kind-contribution/rf-systems/

[4] *Target – ESS Bilbao*. (2021, 17 septiembre). Blanco ESS Bilbao. https://www.essbilbao.org/in-kind-contribution/target/

[5] *Miracles Instrument – ESS Bilbao*. (2021, 17 septiembre). Instrumento Miracles ESS Bilbao. https://www.essbilbao.org/in-kind-contribution/instrument/

[6] *Historia Aceleradores Partículas*. (2001, 5 julio). Historia aceleradores de partículas. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.978.1689&rep=rep1&type=pdf#page=339

[7] *Radiofrequency cavities*. (2012, 17 septiembre). CERN Document Server. https://cds.cern.ch/record/1997424?ln=es

[8] *Modulator – ESS Bilbao*. (2016, 8 abril). Moduladores ESS Bilbao. https://www.essbilbao.org/es/in-kind-contribution/rf-systems/modulator/

[9] S. Hanna - RF linear accelerators for medical and industrial applications, 2012

[10] *Medicina*. (2019, 23 mayo). CPAN - Centro Nacional de Física de Partículas, Astropartículas y Nuclear. https://www.i-cpan.es/es/content/medicina

[11] Stefano Mattei – Principios de aceleradores de partículas. https://indico.cern.ch/event/346149/contributions/1749754/attachments/682406/937460/Principios_de_aceleradores3.pdf

[12] A.M. Lombardi – Radio Frequency Quadrupole. CERN, Geneva, Swizterland https://cds.cern.ch/record/1005049/files/p201.pdf

[13] Jhon Byrd – Resonant Cavities, Lawrence Berkeley National Laboratory, USPAS, June 23, 2009 https://uspas.fnal.gov/materials/09UNM/ResonantCavities.pdf

[14] *Resonant Modes*. (2015). Modos resonantes. https://indico.cern.ch/event/356897/contributions/1769073/attachments/709980/974611/JUAS-part2.pdf

[15] Edurne Monteoliva de la Pedraja (2021). Desarrollo de un software de meddia avanzada de factor de calidad de cavidades resonantes para aceleradores de partículas. Trabajo Fin de Máster

[16] Kajfez, D. Q-factor. Encyclopdia RF Microwave Qfactor.

[17] *Bead-Pull measurements techniques and Multipoles components of DQW crab-cavity*. (2018). R/Q. https://indico.cern.ch/event/807860/sessions/309055/attachments/1827766/2991933/PaulFrancoisGapais_SummerStudent2018.pdf

[18] J.C. Slater - Field Strength Measurements in Resonant Cavities 1952

Anexo: Código fuente

Interfaz de usuario para el bead pull

```
###### calculo bead-pull #########

"""
Created on Mo Mayo 10 10:34:00 2021

@author: Edurne Monteoliva y Alejandro Gutierrez
"""
```

```
'\nCreated on Mo Mayo 10 10:34:00 2021\n\n@author: Edurne Monteo
liva y Alejandro Gutierrez\n'
```

PROGRAMA

```
################################################################################

import serial
import numpy as np
from time import sleep

verbose = False

class M23MDSI(serial.Serial):
    """
    This class represents the 23MDSI stepper motor from Anaheim
Automation.
    A serial RS485 connection is established.
    """
    def __init__(self, ser, address=0):
        #self.ser = serial.Serial(port=port, baudrate=38400,
       #bytesize=8, stopbits=1, parity='N',timeout=0,xonxoff=1)
        #ser.open() # already opened
        # Serial port connection
        self.ser = ser
        self.ser.baudrate=38400;  self.ser.bytesize=8;  self.ser
.stopbits=1;
        self.ser.parity='N';  self.ser.timeout=0;  self.ser.xonx
off=1
        if verbose: print('\n23MDSI Motor connected to Serial Po
rt? '+str(self.ser.is_open))
        self.address = self.set_address(address)
        self.profile = 1
        self.errorcode = 0
```

```
    def __str__(self):
        return '\n23MDSI STEPPER MOTOR FROM ANAHEIM AUTOMATION\n
'

    def set_address(self, address=0):
        # Set the motor address
        self.address = self.check_value('address', address)
        data = '@~'+str(self.address)+'\r' # Address
        if verbose: print(data)
        self.ser.write(data.encode())
        sleep(0.05)
        if verbose: print('Motor Address set to : '+str(self.add
ress))

    def get_address(self):
        # Query the motor address
        data = '@%\r'
        if verbose: print(data)
        self.ser.write(data.encode())
        sleep(0.1)
        data = self.ser.read(4)
        self.address = int(data.decode())
        if verbose: print('Motor Address is: '+str(self.address)
)
        return self.address

    def get_version(self):
        # Ask motor controller version number
        data = '@'+str(self.address)+'$\r'
        if verbose: print(data)
        self.ser.write(data.encode())  #self.ser.write(b'@0$\r')
        sleep(0.1)
        data = self.ser.read(20)
        datad = data.decode()
        self.version = datad.replace('\r', ' ')
        if verbose: print('Version is: '+self.version)
        return self.version

    def get_errorcode(self):
        # Ask motor error code
        data = '@'+str(self.address)+'!\r'
        if verbose: print(data)
        self.ser.write(data.encode())  #self.ser.write(b'@0~\r')
        sleep(0.1)
        data = self.ser.read(20)
        self.errorcode = int(data.decode())
        ec_dict = {0: 'No error',
                   1: 'Receive overflow error (serial communicati
on error caused by the computer)',
```

```
                  2: 'Range error (invalid number of characters)
',
                  4: 'Command error (command not valid or sent w
hen the motor is running/not running)',
                  6: 'Unknown error (e.g.: incorrect index numbe
r verification?)',
                  8: 'Transmit error (too many paramenters sent
back to computer from motor)',
                  16: 'Motor error (speed profile set incorrectl
y: e.g.: base speed > max speed)',
                  32: 'Zero parameters error (command sent witho
ut required parameters)'}
        if verbose: print('Error code '+str(self.errorcode)+': '
+ec_dict[self.errorcode])
        if verbose: print('Error reseted') if self.errorcode !=
0 else ''
        self.errorcode = 0
        return self.errorcode

    def reset(self):
        self.get_errorcode()

    def verify(self, profile=1):
        if verbose: print('\nVERIFICATING PARAMETERS OF: 23MDSI
stepper motor')
        if verbose: print('Address:  '+str(self.address))  # Add
ress
        self.profile = self.check_value('profile', profile)  # P
rofile
        if verbose: print('Profile:  '+str(self.profile))

        self.A = self.get_acceleration(self.profile)
        self.B = self.get_basespeed(self.profile)
        self.M = self.get_maxspeed(self.profile)
        self.N = self.get_indexnumber(self.profile)
        self.T = self.get_completetime(self.profile)
        self.D = self.get_direction()
        self.R = self.get_resolution()

        #self.duration = self.get_duration(self, selfprofile)
        return self.A, self.B, self.M, self.N, self.T, self.D, s
elf.R

    def verify_parameters(self, profile=1):
        self.verify(profile)

    def get_parameters(self, profile=1):
        self.verify(profile)

    def get_acceleration(self, profile=1):
```

```
        self.profile = self.check_value('profile', profile)  # P
rofile
        data = '@'+str(self.address)+'VA'+str(self.profile)+'\r'
        if verbose: print(data)


        self.ser.write(data.encode())  # verify Acceleration
        sleep(0.1)
        data = self.ser.read(20)
        self.A = int(data.decode())
        if verbose: print('Acceleration:  '+str(self.A))
        return self.A

    def get_basespeed(self, profile=1):
        self.profile = self.check_value('profile', profile)  # P
rofile
        data = '@'+str(self.address)+'VB'+str(self.profile)+'\r'


        if verbose: print(data)
        self.ser.write(data.encode())  # verify Base speed
        sleep(0.1)
        data = self.ser.read(20)
        self.B = int(data.decode())
        if verbose: print('Base speed:    '+str(self.B))
        return self.B

    def get_maxspeed(self, profile=1):
        self.profile = self.check_value('profile', profile)  # P
rofile
        data = '@'+str(self.address)+'VM'+str(self.profile)+'\r'


        if verbose: print(data)
        self.ser.write(data.encode())  # verify Max speed
        sleep(0.1)
        data = self.ser.read(20)
        self.M = int(data.decode())
        if verbose: print('Max speed:     '+str(self.M))
        return self.M

    def get_indexnumber(self, profile=1):
        self.profile = self.check_value('profile', profile)  # P
rofile
        data = '@'+str(self.address)+'VN'+str(self.profile)+'\r'


        if verbose: print(data)
        self.ser.write(data.encode())  # verify index Number
        sleep(0.1)
        data = self.ser.read(20)
```

```
        self.N =  int(data.decode())
        if verbose: print('Index Number:  '+str(self.N))
        return self.N

    def get_completetime(self, profile=1):
        self.profile = self.check_value('profile', profile)  # P
rofile
        data = '@'+str(self.address)+'VT'+str(self.profile)+'\r'
# verify complete Time
        if verbose: print(data)
        self.ser.write(data.encode())  # verify complete Time
        sleep(0.1)
        data = self.ser.read(20)
        self.T = int(data.decode())
        if verbose: print('Complete Time: '+str(self.T))
        return self.T

    def get_direction(self):
        data = '@'+str(self.address)+'V+\r' # verify Direction
        if verbose: print(data)
        self.ser.write(data.encode())   # verify Direction
        sleep(0.1)
        data = self.ser.read(20)
        self.D = int(data.decode())
        if verbose: print('Direction:     '+str(self.D))
        return self.D

    def get_resolution(self):
        data = '@'+str(self.address)+'VR\r' # verify Resolution
        if verbose: print(data)
        self.ser.write(data.encode())  # verify Resolution
        sleep(0.1)
        data = self.ser.read(20)
        self.R = int(data.decode())
        if verbose: print('Resolution:    '+str(self.R))
        return self.R

    def set_parameters(self, profile=1, A=10000, B=600, M=4800,
N=48000, T=500, D=0, R=8):
        if verbose: print('\nCONFIGURING PARAMETERS OF: 23MDSI s
tepper motor')
        if verbose: print('Address:  '+str(self.address))
        self.profile = self.check_value('profile', profile)  # P
rofile
        if verbose: print('Profile:  '+str(self.profile))
        self.set_acceleration(self.profile, A)
        self.set_basespeed(self.profile, B)
        self.set_maxspeed(self.profile, M)
        self.set_indexnumber(self.profile, N)
        self.set_completetime(self.profile, T)
```

```
        self.set_direction(D)
        self.set_resolution(R)
        #self.duration = self.get_duration(self, self.profile)

    def set_acceleration(self, profile=1, A=10000):
        self.profile = self.check_value('profile', profile)  # P
rofile
        self.A = self.check_value('acceleration', A)  # Accelera
tion
        data = '@'+str(self.address)+'A'+str(self.profile)+'_'+s
tr(self.A)+'\r' # set Acceleration
        if verbose: print(data)
        self.ser.write(data.encode())  # set Acceleration
        sleep(0.1)
        if verbose: print('Acceleration set to  '+str(self.A))

    def set_basespeed(self, profile=1, B=600):
        self.profile = self.check_value('profile', profile)  # P
rofile
        self.B = self.check_value('basespeed', B)  # Base speed
        data = '@'+str(self.address)+'B'+str(self.profile)+'_'+s
tr(self.B)+'\r' # set Base speed
        if verbose: print(data)
        self.ser.write(data.encode())  # set Base speed
        sleep(0.1)
        if verbose: print('Base speed set to    '+str(self.B))

    def set_maxspeed(self, profile=1, M=4800):
        self.profile = self.check_value('profile', profile)  # P
rofile
        self.M = self.check_value('maxspeed', M)  # Max speed
        data = '@'+str(self.address)+'M'+str(self.profile)+'_'+s
tr(self.M)+'\r' # set Max speed
        if verbose: print(data)
        self.ser.write(data.encode())  # set Max speed
        sleep(0.1)
        if verbose: print('Max speed set to     '+str(self.M))

    def set_indexnumber(self, profile=1, N=48000):
        self.profile = self.check_value('profile', profile)  # P
rofile
        self.N = self.check_value('indexnumber', N)  # index Num
ber
        data = '@'+str(self.address)+'N'+str(self.profile)+'_'+s
tr(self.N)+'\r'  # set index Number
        if verbose: print(data)
        self.ser.write(data.encode())  # set index Number
        sleep(0.1)
        if verbose: print('Index Number set to  '+str(self.N))
```

```
    def set_completetime(self, profile=1, T=500):
        self.profile = self.check_value('profile', profile)  # P
rofile
        self.T = self.check_value('completetime', T)  # complete
Time
        data = '@'+str(self.address)+'T'+str(self.profile)+'_'+s
tr(self.T)+'\r' # set complete Time
        if verbose: print(data)
        self.ser.write(data.encode())  # set complete Time
        sleep(0.1)
        if verbose: print('Complete time set to '+str(self.T))

    def set_direction(self, D=0):
        self.D = self.check_value('direction', D)  # direction
        dir = '+' if self.D == 1 else '-'
        data = '@'+str(self.address)+str(dir)+'\r' # set Directi
on
        if verbose: print(data)
        self.ser.write(data.encode())  # set Direction
        sleep(0.1)
        if verbose: print('Direction set to     '+str(self.D))

    def change_direction(self):
        if verbose: print('Direction is:     '+str(self.D))
        D = 1 if self.D == 0 else 0   #change direction
        self.set_direction(D)

    def set_resolution(self, R=8):
        self.R = self.check_value('resolution', R)  # direction
        data = '@'+str(self.address)+'R'+str(self.R)+'\r' # set
Resolution
        if verbose: print(data)
        self.ser.write(data.encode())  # set Resolution
        sleep(0.1)
        if verbose: print('Resolution set to    '+str(self.R))

    def begin_motion(self, profile=1):
        #print("Estamos moviendonos")
        # Move using current settings
        self.profile = self.check_value('profile', profile)  # P
rofile
        if verbose: print('Profile:  '+str(self.profile))
        if verbose: print('Direction: '+str(self.D))
        data = '@'+str(self.address)+'G'+str(self.profile)+'\r'
# Go (Run) using profile
        if verbose: print(data)
        self.ser.write(data.encode()) # Go (Run) using profile 1
        try:
            if verbose: print('Motor is running... It is expecte
d to stop after '+str(self.duration)+' sec')
```

```
            except:
                pass


        def run(self, profile=1):
            self.begin_motion(profile)

        def go(self, profile=1):
            self.begin_motion(profile)

        def stop_motion(self):
            #print("Nos paramos")
            data = '@'+str(self.address)+'H\r' # Message
            if verbose: print(data)
            self.ser.write(data.encode()) # self.ser.write(b'@0H\r')
            if verbose: print('Motor stopped!')
            sleep(0.1)

        def stop(self):
            self.stop_motion()

        def hard_limit(self):
            self.stop_motion()

        def soft_stop(self):
            data = '@'+str(self.address)+'S\r' # Soft Stop
            if verbose: print(data)
            self.ser.write(data.encode()) # self.ser.write(b'@0S\r')
            if verbose: print('Motor stopping!')
            sleep(0.1)

        def soft_limit(self):
            self.soft_stop()

        def close(self):
            self.ser.close()
            if verbose: print( 'Connected to Serial Port? '+str(self
.ser.is_open()) )

        def get_duration(self, profile=1):
            # Calculation of motion duration
            self.profile = self.check_value('profile', profile)  # P
rofile
            if verbose: print('Profile is      '+str(profile))

            motor_max_speed = self.M
            if verbose: print('Max speed is    '+str(self.M))
            index_number = self.N
            if verbose: print('Index Number is '+str(self.N))
```

```
        steps_per_sec = motor_max_speed
        duration_sec = float(index_number / steps_per_sec)
        self.duration = duration_sec
        if verbose: print('Expected motion duration is '+str(sel
f.duration))
        return self.duration

    def check_value(self, parameter='profile', value=1):
        if parameter == 'address':
            self.address = int(value)
            if (self.address not in range(0,100,1)):
                print('Warning: Address should be between 0 and
99; Set to 0');
                self.address = 0
            return self.address
        if parameter == 'profile':
            self.profile = int(value)
            if (self.profile not in [1,2]):
                print('Warning: Profile should be 1 or 2; Set to
1');
                self.profile = 1
            return self.profile
        if parameter == 'acceleration' or parameter =='A':
            self.A = int(value)
            if (self.A < 100 or self.A > 9999999):   # Accelerat
ion
                print('Warning: Acceleration should be between 1
00 and 9999999; Set to 10000');
                self.A = 10000
            return self.A
        if parameter == 'basespeed' or parameter =='B':
            self.B = int(value)
            if (self.B < 1 or self.B > 5000): # Base speed
                print('Warning: Base speed should be between 1 a
nd 5000; Set to 600');
                self.B = 800
            return self.B
        if parameter == 'maxspeed' or parameter =='M':
            self.M = int(value)
            if (self.M < 1 or self.M > 50000):  # Max speed
                print('Warning: Max speed should be between 1 an
d 50000; Set to 4800');
                self.M = 4800
            return self.M
        if parameter == 'indexnumber' or parameter =='N':
            self.N = int(value)
            if (self.N < 1 or self.N > 8388607):  # index Number
                print('Warning: index Number should be between 1
and 8388607; Set to 48000');
```

```
                self.N = 48000
            return self.N
        if parameter == 'completetime' or parameter =='T':
            self.T = int(value)
            if (self.T < 1 or self.T > 1000):  # complete Time
                print('Warning: complete Time should be between
1 and 1000; Set to 500');
                self.T = 500
            return self.T
        if parameter == 'direction' or parameter =='D':
            self.D = int(value)
            if (self.D not in [0,1]):  # Direction
                print('Warning: Direction should be 0 (-: CCW) o
r 1 (+: CW); Set to 0');
                self.D = 0
            return self.D
        if parameter == 'resolution' or parameter =='R':
            self.R = int(value)
            if (self.R not in [1,2,4,8]):  # Resolution
                print('Warning: Resolution should be 1, 2, 4 or
8; Set to 8');
                self.R = 8
            return self.R
######################################################################


######################################################################
```

VNA MEDIDAS

```
#Funcion para realizar la medida de la fase en la cavidad resona
nte
def configureVNA(Span_t):
    global f0

    print(' CONFIGURATION OF VNA FOR BEAD PULL MEASUREMENT ')
    pyvisa.ResourceManager('@py')

    rm = pyvisa.ResourceManager()
    print(rm)
    print(rm.list_resources())

    pna = rm.open_resource('GPIB0::16::INSTR')
    print(pna)
    pna.timeout = 5000
    #pna.read_termination = '\n'
    #pna.write_termination = '\n'

    print(pna.query('*IDN?'))
```

```
Npoints_t = Cavity_Length/Z_Step #Z_Leng /Zstep

pna.write('FORM:BORD SWAP')
pna.write('FORM REAL, 64')

#Channel 1
pna.write('SENS1:SWE:TYPE LIN')
pna.write('SENS1:SWE:MODE CONT')
pna.write('SENS1:SWE:POIN 801')
pna.write('SENS1:SWE:TIME 1S')
pna.write('SENS1:BAND:RES 10e3')
sleep(1)
# Frecuencia central y span utilizado en el PNA
pna.write('SENS1:FREQ:CENTER ',str(f_Center_Aprox))
pna.write('SENS1:FREQ:SPAN ', str(Span))

#channel 1 trace 2
pna.write('DISP:WIND:STAte OFF')
pna.write('DISP:WIND1:STAte ON')
pna.write('CALC:PAR:DEF "DBS11_f" ,S11')
pna.write('CALC:PAR:DEF "DBS21_f" ,S21')
pna.write('DISP:WIND1:TRAC1:FEED "DBS11_f"')
pna.write('DISP:WIND1:TRAC2:FEED "DBS21_f"')
#Delete Trace
#pna.write('DISP:WIND1:TRAC1:DEL')

pna.write('CALC:PAR:SEL "DBS21_f"')
pna.write('CALC:CORR:STATe OFF')
pna.write('CALC:FORM MLOG')
sleep(2)
pna.write('CALC:MARK:STAT ON')
# SLEEP PARA QUE LE DE TIEMPO A REALIZAR LA MEDIDA
sleep(1)
pna.write('CALC:MARK1:FUNC:EXEC MAX')
#Se acopla el primer market
pna.write('CALC:MARK1:COUP ON')
#f0 = eval(pna.query('CALC1:MARK1:X?'))
f0 = pna.query('CALC1:MARK1:X?')
print('f0 = ', f0)

#Channel 1 trace 3
pna.write('CALC:PAR:DEF "ARGS21_f", S21')
pna.write('DISP:WIND1:TRAC3:FEED "ARGS21_f"')
pna.write('CALC:PAR:SEL "ARGS21_f"')
pna.write('CALC:CORR:STATe OFF')
pna.write('CALC:FORM PHAS')
#sleep(1)
pna.write('CALC:MARK1:STAT ON')

f0 = pna.query('CALC:MARK1:X?')
```

```
    phiostr = pna.query('CALC:MARK1:Y?')
    print('phiostr = ', phiostr)
    ind = phiostr.split(',')
    #phio = eval(phiostr(0:ind-1))
    phio = float(ind[0])
    print('phio = ', phio)

    #Channel 2 ->trace 4
    #Primero se crea el canal 2 y la traza 4
    pna.write('CALC2:PAR:DEF "ARGS21_t", S21')
    pna.write('DISP:WIND1:TRAC4:FEED "ARGS21_t"')
    pna.write('CALC2:PAR:SEL "ARGS21_t"')
    pna.write('CALC2:CORR:STATe OFF')
    pna.write('CALC2:FORM PHAS')

    #Channel 2-> Trace 4(CW TIME Sweep)
    pna.write('SENS2:SWEEP:TYPE CW')
    ####COMPROBAR EN EL LAB TENGO DUDAS DE QUE SEA NECESARIO ASI
    pna.write('SENS2:FREQ:CW ', str(f0))
    pna.write('SENS2:SWE:POIN 801')

    CW_sweep_time = 2
    pna.write('SENS2:SWE:TIME ', str(CW_sweep_time)) #send 1 sec
ond
    pna.write('SENS2:BAND:RES 1e3')
    #pna.write('SENS2:SWE:MODE SING')
    pna.write('SENS2:SWE:GROups:COUN 1')
    pna.write('SENS2:SWE:MODE GRO')
    sleep(CW_sweep_time + 4)# tiempo para calcular el maximo
    pna.write('CALC2:MARK1:STATe ON')
    #pna.write('CALC2:MARK1:X ', str(CW_sweep_time/2))
    phiotstr = pna.query('CALC1:MARK1:Y?')
    print('phiotstr = ', phiotstr)
    indt = phiotstr.split(',')
    #phio = eval(phiostr(0:ind-1))
    phiot = float(indt[0])
    print('phiot = ', phiot)

    if abs(phio-phiot) > 1 :
        print('CUIDADO: No coincide la fase del S21(fo) en frecu
encia y en tiempo')

    pna.write('SENS2:SWE:POIN ', str(Npoints_t))
    pna.write('SENS2:SWE:TIME ', str(Span_t))
    fr = f0


def beadPullMeasurements(Span_t):#, structure):
```

```
    #Trigger single
    #Equivalente al getVISA
    pyvisa.ResourceManager('@py')

    rm = pyvisa.ResourceManager()
    print(rm)
    print(rm.list_resources())

    pna = rm.open_resource('GPIB0::16::INSTR')
    print(pna)
    pna.timeout = 5000

    #pna.write('SENS2:SWE:GROups:COUN 1')
    #pna.write('TRIG:SOUR MAN')
    #pna.write('INIT2:IMM')
    pna.write('SENS2:SWE:MODE GRO')#Start Single Sweep
    #pna.write('SENS":SWE:MODE SING ')#Start Sweep
    sleep(1.5)#first channel one. It´s going to finish after 2 s
econs

    ########DAQ#######
    #Movements starts after falling slope detection in profile2

    #Hay que comenzar el movimiento del motor para realizar las
     #medidas usando profile 2

    #Para el movimineto del motor ->simple llamada a funcion
    print('Dentro de las medidas antes begin_motion')
    motor1.begin_motion()
    print('despues de begin_motion')
    sleep(Span_t)
    motor1.stop_motion()

    #trabajando con el PNA
    pna.write('CALC2:PAR:SEL "ARGS21_t"')
    pna.write('CALC2:MARK1:FUNC:EXEC MIN')
    t_phio_min = pna.query('CALC2:MARK1:X?')
    phiostr_min = pna.query('CALC2:MARK1:Y?')
    ind=phiostr_min.split(',')
    phio_min = float(ind[0])

    pna.write('CALC2:PAR:SEL "ARGS21_t"')
    pna.write('CALC2:MARK2:STAT ON')
    pna.write('CALC2:MARK2:FUNC:EXEC MAX')
    t_phio_max = pna.query('CALC2:MARK2:X?')
    phiostr_max = pna.query('CALC2:MARK2:Y?')
    ind=phiostr_max.split(',')
    phio_max = float(ind[0])
    print('phio_min = ', phio_min)
    print('t_phio_min= ', t_phio_min)
```

```
    print('phio_max = ', phio_max)
    print('t_phio_max= ', t_phio_max)
    inc_phase = abs(phio_max - phio_min)
    print('inc_phase = ', inc_phase)

    #####NECESARIO######
    #set(structure, 'str', num2str(inc_phase)) #cambia el valor
de fase max por el calculado

    pna.write('DISP:WIND:TRAC4:Y:AUTO')

    return inc_phase

######################################################################


import datetime
import time

def saveS2Pfile():

    # Al ejecutar la función de guardar fichero, desaparece la t
raza 4 que en este caso es la phase del S21
    pyvisa.ResourceManager('@py')
    rm = pyvisa.ResourceManager()
    pna = rm.open_resource('GPIB0::16::INSTR')
    pna.timeout = 5000
    #pna.write('SENS2:SWE:MODE HOLD')
    pna.write('CALC2:PAR:SEL "ARGS21_t"')

    # Se leen los valores del VNA, SNP returns 9*N data points
    pna.write('MMEM:STOR:TRAC:FORM:SNP DB') # Fija que se envíen
los datos en dB
    #data = pna.query_binary_values('CALC2:DATA:SNP?', datatype=
'd', is_big_endian=False) # N5241A superseded
    data = pna.query_binary_values('CALC2:DATA:SNP:PORTs? "1, 2"
', datatype='d', is_big_endian=False)
    #pna.write('SENS2:SWE:MODE HOLD')

    # WITH ARRAYS
    data = np.array(data)
    print("data = ", data[15000])

    # Reshape data so it is split into 9 columns (freq, S11dB, S
11deg, S21...
    N = int(len(data)/9)
    print(N)
    t = data[0:N]
    dBS11 = data[N:2*N];        AS11 = data[2*N:3*N]
```

```
    dBS21 = data[3*N:4*N];      AS21 = data[4*N:5*N]
    dBS12 = data[5*N:6*N];      AS12 = data[6*N:7*N]
    dBS22 = data[7*N:8*N];      AS22 = data[8*N:9*N]

    phit = AS21

    resultados = datetime.datetime.now().strftime("%Y_%m_%d__%H_
%M_%S_ResultsBeadPull.s2p")
    date = datetime.datetime.now().strftime("%Y/%m/%d--%H:%M:%S"
)

    #fichero de escritura de resultados
    fichero_resultados = open(resultados,"w")
    fichero_resultados.write("!RF SYSTEMS: ESS BILBAO\n")
    fichero_resultados.write("!Date: %s\n" % date)
    fichero_resultados.write("!CW TIME SWEEP\n")
    fichero_resultados.write("!CW FREQ: %s\n" % f0)
    fichero_resultados.write("# S S DB R 50\n")
    fichero_resultados.write("!\n")
    fichero_resultados.write("!            S11            S21
S12            S22\n")
    fichero_resultados.write("!TIME       DB       ANG       DB
ANG       DB       ANG       DB       ANG\n")

    print('len(t): ',str(len(t)))
    for i in range(0,len(t)):
        fichero_resultados.write("{:9.6f}  {:7.3f}  {:7.3f}  {:7
.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}  {:7.3f}\n"
                                 .format(t[i], dBS11[i], AS11[i]
, dBS21[i], AS21[i], dBS12[i], AS12[i], dBS22[i], AS22[i]))

    fichero_resultados.close()

    print("Fichero .S2P guardado")


######################################################################
```

DEFINING A GUI TO VIEW RESULTS

```
# Generate GUI

import pyvisa
import ipywidgets as widgets
from ipywidgets import Layout
from ipywidgets import Image
from IPython.display import display
from IPython.display import clear_output, display
```

```
class Bead_Pull_VNA_Gui():

    def __init__(self):

        global ser, motor1, obj

        obj = 0

        # TITLE
        titulo = widgets.Label(value = 'ESS BILBAO: BEAD-PULL ME
ASUREMENT')
        nombreE = widgets.Label(value = 'Edurne Monteoliva')
        nombreA = widgets.Label(value = 'Alejandro Gutierrez')

        with open('logo_ess_grande.jpg','rb') as f:
            logo = f.read()
        logo1 = Image(value=logo, layout = Layout(width = '100px
                '))

        with open('logoUC.png','rb') as f:
            logo = f.read()
        logo2 = Image(value=logo, layout = Layout(width = '50px'
                ))

        box1 = widgets.HBox(children = [logo1, logo2, titulo],
                layout = Layout(width = '87%'))
        box2 = widgets.VBox(children = [nombreE, nombreA])

        box_titulo = widgets.HBox(children = [box1, box2],
                layout = Layout(width = '100%'))

        #Puerto serie
        slct_serialport = widgets.Select(options=['COM1','COM2',
'COM3','COM4','COM5','COM6','COM7','COM8','COM9',
                                                  'COM10','/dev/
tty.usbserial-FTTCE3UT'],
                                         value='COM6', description='S
erial Port:', rows=1, disabled=False)
        #Adress y boton conect
        slct_address = widgets.Select(options=[0,1,2,3,4,5,6,7,8
,9], value=0, description='Address:', rows=1, disabled=False)
        btn_connect = widgets.Button(description='Connect', disa
bled=False, button_style='success',
                     tooltip='Click to connect to motor through t
he serial port', icon='')

        chckbx_connection = widgets.Valid(value=False,descriptio
n='', disabled=True)

        txt_version = widgets.Text(value='',description='Version
```

```
                    :',disabled=True)
        inttxt_address = widgets.IntText(value=-99,description='
                    Address:',disabled=True)
        btn_disconnect = widgets.Button(description='Disconnect'
                    , disabled=False, button_style='danger',
                   tooltip='Click to disconnect', icon='')

        slct_serialport.layout.width = '180px';
                slct_address.layout.width = '130px'
        btn_connect.layout.width = '120px'    ;
                chckbx_connection.layout.width = '60px'
        txt_version.layout.width = '200px'    ;
                inttxt_address.layout.width = '140px'
        btn_disconnect.layout.width = '120px'

        box_Serial_Address = widgets.HBox(children = [slct_seria
lport, slct_address, btn_connect, chckbx_connection, txt_version
, inttxt_address, btn_disconnect ], layout = Layout(border = 'so
lid 2px'))

        #Tab BeadPull
        #GUI tres zonas: -> Input data, Bead direction
                           y output Data
        #INPUT DATA
        Input_data_title = widgets.Label(value = 'Input Data')
            #Valores
        txt_z_length = widgets.FloatText(value = '200',
        description = 'Z_Length',layout = Layout(width = '70%'))
        txt_Z_step = widgets.FloatText(value = '0.15',
        description = 'Z_Step', layout = Layout(width = '70%'))
        txt_f_center = widgets.FloatText(value = '361.0',
        description = 'f_center',layout = Layout(width = '70%'))
        txt_VNA_span = widgets.FloatText(value = '20.0',
        description = 'VNA Span',layout = Layout(width = '70%'))
        txt_Bead_Speed = widgets.FloatText(value = '14.0625',
        description = 'Bead Speed',layout = Layout(width='70%'))
            #unidades
        txt_z_length_mm = widgets.Label(value = 'mm')
        txt_z_step_mm = widgets.Label(value = 'mm')
        txt_f_center_MHz = widgets.Label(value = 'MHz')
        txt_VNA_Span_MHz = widgets.Label(value = 'MHz')
        txt_Bead_Speed_mm_s = widgets.Label(value = 'mm/s')

        box_input_data_1 = widgets.HBox(children =
                           [txt_z_length, txt_z_length_mm])
        box_input_data_2 = widgets.HBox(children =
                           [txt_Z_step, txt_z_step_mm])
        box_input_data_3 = widgets.HBox(children =
                           [txt_f_center, txt_f_center_MHz])
        box_input_data_4 = widgets.HBox(children =
```

```
                            [txt_VNA_span, txt_VNA_Span_MHz])
        box_input_data_5 = widgets.HBox(children =
                            [txt_Bead_Speed,txt_Bead_Speed_mm_s])

        #Bead Direction
        Bead_direction_title = widgets.Label(value = 'Bead Direc
tion', style = {'description_width': 'initial'})
            #in->out/out>in --> Radiobutton
        rdbtn_in_out = widgets.RadioButtons(options = ['in -> Ou
t','Out -> In'], value = 'in -> Out', disabled = False)
            #in->out/out>in --> Button


            #Orden de la GUI
        box_bead_rdbtn =  widgets.VBox(children =
                            [Bead_direction_title, rdbtn_in_out],
                                 layout = Layout(width = '100%'))
            #move/Stop -> Button
        Move_botton = widgets.Button(description = 'Move',
                     disabled = False, button_style ='success')
        Stop_botton = widgets.Button(description = 'Stop',
                     disabled = False, button_style = 'info')

        #Start bead- pull test
        Start_Bead_botton = widgets.Button(description = 'Start
Bead Pull', disabled = False,
                                           button_style = 'warni
ng', tooltip = 'click if you want star the measure')

        #Output data
        Out_Data_title = widgets.Label(value = 'Output Data', st
yle = {'description_width': 'initial'})
        txt_fase = widgets.FloatText(description = 'fase max', d
isabled = True, layout = Layout(width = '70%'))
            #guardar fichero y salir
        save_file_botton = widgets.Button(description = 'Save S2
p file ', disabled = False,
                                           button_style = 'succe
ss', tooltip = 'click if you want save the file')
        Exit_botton = widgets.Button(description = 'Exit', disab
led = False,
                                           button_style = 'dange
r', tooltip = 'click if you want to exit')

        box_save_exit = widgets.HBox(children = [save_file_botto
n, Exit_botton])


        #Tab del motor
        #Set Parameters
```

```
        rdbtn_profile = widgets.RadioButtons(options=['1', '2'],
value='1', description='Profile:',disabled=False)
        flttxt_A = widgets.BoundedFloatText(value=10000,descript
ion='Acceleration:',min=100,max=9999999,step=1,disabled=False)
        flttxt_B = widgets.BoundedFloatText(value=600,descriptio
n='Base speed:',min=1,max=5000,step=1,disabled=False)
        flttxt_M = widgets.BoundedFloatText(value=4800,descripti
on='Max speed:',min=1,max=50000,step=1,disabled=False)
        flttxt_N = widgets.BoundedFloatText(value=480000,descrip
tion='index No.:',min=1,max=8388607,step=1,disabled=False)
        flttxt_T = widgets.BoundedFloatText(value=500,descriptio
n='complete T:',min=1,max=1000,step=1,disabled=False)
        rdbtn_R = widgets.RadioButtons(options=['1','2','4','8']
,value='8', description='Resolution:',disabled=False)
        btn_setparameters = widgets.Button(description='Set Para
meters', disabled=False, button_style='success',
                    tooltip='Click to send parameters to motor',
icon='')

        rdbtn_profile.layout.width = '150px';
        flttxt_A.layout.width = '200px';
         flttxt_B.layout.width = '200px'
        flttxt_M.layout.width = '200px';
         flttxt_N.layout.width = '200px'
        flttxt_T.layout.width = '200px';
         rdbtn_R.layout.width = '150px'
        btn_setparameters.layout.width = '150px'

        #Box BeadPuLL
        box_Input = widgets.VBox(children = [Input_data_title, b
ox_input_data_1, box_input_data_2, box_input_data_3, box_input_d
ata_4, box_input_data_5],
                                 layout = Layout(border = 'solid
2px', width = '50%'))

        box_move_stop = widgets.HBox(children = [Move_botton, St
op_botton])

        box_Bead_Direction = widgets.VBox(children = [box_bead_r
dbtn, box_move_stop],
                                          layout = Layout(border
= 'solid 2px'))

        Box_bead_star = widgets.VBox(children = [box_Bead_Direct
ion, Start_Bead_botton],
                                     layout = Layout(border = 'so
lid 2px'))

        box_Output = widgets.VBox(children = [Out_Data_title, tx
t_fase, box_save_exit],
```

```
                                    layout = Layout(border = 'solid
2px'))

        Box_bead_start_Output = widgets.VBox(children = [box_Bea
d_Direction, Start_Bead_botton, box_Output],
                                             layout = Layout(widt
h = '50%'))

        box_BeadPull = widgets.HBox(children = [box_Input, Box_b
ead_start_Output])

        #Box Motor ANAHEIM 23MDSI
        titulo_motor = widgets.Label(value = 'Motor ANAHEIM 23MD
SI')
        box_motor = widgets.VBox(children = [titulo_motor, rdbtn
_profile, flttxt_A, flttxt_B, flttxt_M, flttxt_N,
                                             flttxt_T, rdbtn_R,
btn_setparameters],
                                             layout = Layout(bor
der = 'solid 2px', width = '40%'))

        tab = widgets.Tab(children = [box_BeadPull, box_motor])
        tab.set_title(0,'BeadPull'); tab.set_title(1,'ANAHEIM 23
MDSI')


        box_complete= widgets.VBox(children = [box_titulo, box_S
erial_Address],
                                   layout = Layout(border = 'so
lid 2px')) #Box serial Port

        #Abrir una conexion con el motor
        def cb_do_connect(objt):
            global ser, motor1
            ser = serial.Serial(port=slct_serialport.value)
            motor1 = M23MDSI(ser, address=slct_address.value)
            if verbose: print(motor1)
            inttxt_address.value = motor1.get_address()
            txt_version.value = motor1.get_version()
            D = 1 if rdbtn_in_out.value == 'in -> Out' else 0
            motor1.set_direction(D)
            #cb_get_parameters(obj)
            if txt_version.value != '':
                chckbx_connection.value = True

        def cb_do_disconnect(obj):
            ser.close()
            ser.is_open
            if verbose: print('Serial Port is open? '+str(ser.is
_open))
```

```
            chckbx_connection.value = ser.is_open
            txt_version.value = ''
            inttxt_address.value = -99

        btn_connect.on_click(cb_do_connect)
        btn_disconnect.on_click(cb_do_disconnect)


        def cb_set_direction (change):
            option = change.new #option va valer 0 si In->out co
n D=1
            if option == 0:
                D = 1
            else:
                D = 0
            motor1.set_direction(D)

        rdbtn_in_out.observe(cb_set_direction, names = 'index')

        #Funcion de paso de parametros que de momento solo hace
falta calcular la fase -> más adelante

        #Motion -> Se usa con el bead pull y con el Move
        def  cb_begin_motion(obj):
            Start_Bead_botton.description = 'Running the Bead Pu
ll...'
            Move_botton.description = 'Running...'
            motor1.begin_motion(profile = motor1.profile)
            sleep(1)
            Move_botton.description = 'Move'
            Start_Bead_botton.description = 'Start Bead Pull'

            return

        def Start_BeadPull(verargin):
            if 'in -> Out':
                Dir = 1
            if 'Out -> in':
                Dir = 0

            #Get params
            global Cavity_Length, Z_Step, f_Center_Aprox, Span
            Cavity_Length = txt_z_length.value
            Z_Step = txt_Z_step.value
            f_Center_Aprox = txt_f_center.value*1e6
            Span = txt_VNA_span.value*1e6

            if (Cavity_Length == 0 or Z_Step == 0 or f_Center_Ap
rox == 0 or Span == 0):
                print('ERROR: Input data is empty')
```

```
            else:
                v_motor = txt_Bead_Speed.value
                Span_t = Cavity_Length / v_motor
                print('Span_t = ', Span_t)


                configureVNA(Span_t)


                Delta_Phase = beadPullMeasurements(Span_t)
                txt_fase.value = round(Delta_Phase, 2)


        Start_Bead_botton.on_click(Start_BeadPull)
        Move_botton.on_click(cb_begin_motion)

        #Para del motor -> Pulsando Stop
        def cb_stop_motion(obj):
            motor1.stop_motion()
            sleep(1)
            motor1.reset()
            return

        Stop_botton.on_click(cb_stop_motion)

        def cb_set_parameters(obj):
            D = 1 if rdbtn_in_out.value == 'in -> Out' else 0
            motor1.set_parameters(profile=rdbtn_profile.value, A
=flttxt_A.value, B=flttxt_B.value,
                                  M=flttxt_M.value, N=flttxt_N.v
alue, T=flttxt_T.value, D=D, R=rdbtn_R.value)
            cb_get_parameters(0)

        btn_setparameters.on_click(cb_set_parameters)

        def Save(change):
            saveS2Pfile()

        save_file_botton.on_click(Save)

        def Exit(change):
            clear_output()

        Exit_botton.on_click(Exit)

        #Display
        display(box_complete)
        display(tab)
        out = widgets.Output()

# Run GUI
```

```
QoF = Bead_Pull_VNA_Gui()

######################################################################


######################################################################
import sys
import glob
import serial

#Programa para saber en qué puerto serie está conectado el motor

def serial_ports():
    """ Lists serial port names

        :raises EnvironmentError:
            On unsupported or unknown platforms
        :returns:
            A list of the serial ports available on the system
    """
    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.starts
with('cygwin'):
        # this excludes your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result


if __name__ == '__main__':
    print(serial_ports())

['COM4', 'COM6']
```

Procesado de resultados

S2P FILE READ FROM BEADPULL MEASUREMENTS TO PREPARE DATA FOR RFQ TUNING ALGORITHM

""" Created on March 19, 2021 (preliminary version) Version 1: 20210619 (June 19, 2021), importing 4 quadrant bead pull measurements

@author: PJG """# Main reference: RF Measurements and Tuning of the 750 MHz HF-RFQ, B. Koubek, A. Grudiev and M. Timmins, # CERN-AA-NOTE-2017-0006

```
import numpy as np

#####################################################################


def lees2p(filename):
    # Function that reads a .s2p file, i.e. an ascii file accord
ing to the so called
    # "Touchstone" format, listing the scattering parameters (S
parameters) of a
    # 2-port network as a function of frequency.
    # frequency and 4 complex valued S parameters (freq, S11, S2
1, S12 and S22) are
    # stored in 9 columns

    if filename.lower().split('.')[-1] != 's2p':
        print('ERROR: illegal file extension',  filename [-3:-1]
)

    with open(filename, "r") as files2p:
        f,S11_1,S11_2,S21_1,S21_2,S12_1,S12_2,S22_1,S22_2 = [],[
],[],[],[],[],[],[],[]

        while 1:
            line = files2p.readline()
            #print(line)

            if not type(line) == str:
                line = line.decode("ascii")  # for python3 zipfi
le compatibility
            if not line: # end function
                break
            if len(line) == 0: # continue
                continue
            if line[0] == "!": # comments
                continue
            if line[0] == '#': # this means options
                options = line[1:].strip().split()
```

```
                funits = options[0].lower()
                tipoparam = options[1].lower()
                formato = options[2].lower()
                Z0 = options[4]

                if funits not in ['hz','khz', 'mhz', 'ghz','s']:
                    print('ERROR: illegal frequency_unit %s',  f
                          units)
                fmult = {'hz': 1, 'khz': 1e3, 'mhz': 1e6, 'ghz':
                1e9, 's': 1}.get(funits)
                if funits == 's': print('Warning: x-axis is time
                                        in seconds')

                if tipoparam not in 's': # originally 'syzgh'
                    print('ERROR: illegal parameter value %s',
                      tipoparam)

                if formato not in ['ma', 'db', 'ri']:
                    print('ERROR: illegal format value %s]',
                      formato)

            else: # read the parameters
                f.append(fmult*float(line.split()[0]))
                S11_1.append(float(line.split()[1]));
                S11_2.append(float(line.split()[2]))
                S21_1.append(float(line.split()[3]));
                S21_2.append(float(line.split()[4]))
                S12_1.append(float(line.split()[5]));
                S12_2.append(float(line.split()[6]))
                S22_1.append(float(line.split()[7]));
                S22_2.append(float(line.split()[8]))

    f = np.array(f)
    S11_1 = np.array(S11_1); S11_2 = np.array(S11_2);
     S21_1 = np.array(S21_1); S21_2 = np.array(S21_2)
    S12_1 = np.array(S12_1); S12_2 = np.array(S12_2);
     S22_1 = np.array(S22_1); S22_2 = np.array(S22_2)


    # S parameters (complex values) calculated depending on the
format specified in s2p file
    if formato == 'ri':
        S11 = S11_1 + 1j*S11_2;    S21 = S21_1 + 1j*S21_2
        S12 = S12_1 + 1j*S12_2;    S22 = S22_1 + 1j*S22_2
    if formato == 'ma':
        S11 = S11_1 * np.exp(1j*S11_2*np.pi/180);
        S21 = S21_1 * np.exp(1j*S21_2*np.pi/180)
        S12 = S12_1 * np.exp(1j*S12_2*np.pi/180);
        S22 = S22_1 * np.exp(1j*S22_2*np.pi/180)
    if formato == 'db':
```

```
        S11 = 10**(S11_1/20) * np.exp(1j*S11_2*np.pi/180);
        S21 = 10**(S21_1/20) * np.exp(1j*S21_2*np.pi/180)
        S12 = 10**(S12_1/20) * np.exp(1j*S12_2*np.pi/180);
        S22 = 10**(S22_1/20) * np.exp(1j*S22_2*np.pi/180)

    return f, S11, S21, S12, S22

#####################################################################
##################

#############################################

filename_q1 = '2021_06_09__15_28_18_ResultsBeadPull_Q1_OutIn.s2p
'
filename_q2 = '2021_06_09__15_23_38_ResultsBeadPull_Q2_InOut.s2p
'
filename_q3 = '2021_06_09__15_02_17_ResultsBeadPull_Q3.s2p'
filename_q3 = '2021_06_09__15_04_21_ResultsBeadPull_Q3.s2p'
filename_q4 = '2021_06_09__15_34_08_ResultsBeadPull_Q4_InOut.s2p
'

t_sec, S11, S21_q1, S12, S22 = lees2p(filename_q1)
t_sec, S11, S21_q2, S12, S22 = lees2p(filename_q2)
t_sec, S11, S21_q3, S12, S22 = lees2p(filename_q3)
t_sec, S11, S21_q4, S12, S22 = lees2p(filename_q4)


AS21_q1 = np.unwrap(np.angle(S21_q1))*180/np.pi
AS21_q2 = np.unwrap(np.angle(S21_q2))*180/np.pi
AS21_q3 = np.unwrap(np.angle(S21_q3))*180/np.pi
AS21_q4 = np.unwrap(np.angle(S21_q4))*180/np.pi

x_mm = t_sec * 14.0625 # longitudinal axis (considering speed of
14.0625 mm/sec)
deltat = t_sec[1]-t_sec[0]
deltax = x_mm[1]-x_mm[0]
print('deltat (sec): '+str(deltat))
print('deltax (mm):  '+str(deltax))

#%matplotlib notebook
#%matplotlib widget
#%matplotlib inline
%matplotlib ipympl
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('1. RAW DATA') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
```

```
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q2, 'blue', label='q2')
line13, = ax1.plot(x_mm, AS21_q3, 'black', label='q3')
line14, = ax1.plot(x_mm, AS21_q4, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
fig.show()

# PHASE MEASUREMENT NOISE
zoomx = 40

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('1.1 MEASUREMENT NOISE') # figure title
ax1.set_xlabel('t [sec]') # x axis label
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(0, zoomx)  # initial x limits
ax1.set_ylim(np.min(AS21_q1[:10*zoomx])-0.01, np.min(AS21_q1[:10
*zoomx])+0.04) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1, 'red', label='q1')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# ORIENTATION: REVERSING OF OUT-TO-IN MEASUREMENTS

AS21_q1_o = AS21_q1[::-1]
AS21_q2_o = AS21_q2[::]
AS21_q3_o = AS21_q3[::-1]
AS21_q4_o = AS21_q4[::]


fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('2. PROPER ORIENTATION') # figure title
ax1.set_xlabel('length [mm]') # x axis label
```

```
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1_o, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q2_o, 'blue', label='q2')
line13, = ax1.plot(x_mm, AS21_q3_o, 'black', label='q3')
line14, = ax1.plot(x_mm, AS21_q4_o, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# VERTICAL ALIGNMENT (SET REFERENCE FOR 0º)

# WARNING: VERTICAL ALIGNMENT REQUIRES THAT THE RFQ IS WELL CENT
ERED IN THE MEASUREMENT SWEEP TIME,
# I.E.: THE DISTANCES TRAVELED BY THE BEAD PULL AT THE INPUT AND
THE OUTPUT OF THE RFQ ARE SIMILAR,
# E.G.: AROUND 10 CM AT BOTH SIDES

def vertical_alignment(AS21_q1_o, x_mm):
    start1 = 100
    end1 = 200
    print(x_mm[start1:end1:10])
    print(AS21_q1_o[start1:end1:10])

    phaseRef_q1 = np.mean(AS21_q1_o[start1:end1])
    print('phaseRef (deg):  '+str(phaseRef_q1))

    AS21_q1_va = AS21_q1_o - phaseRef_q1

    return AS21_q1_va

AS21_q1_va = vertical_alignment(AS21_q1_o,x_mm)
AS21_q2_va = vertical_alignment(AS21_q2_o,x_mm)
AS21_q3_va = vertical_alignment(AS21_q3_o,x_mm)
AS21_q4_va = vertical_alignment(AS21_q4_o,x_mm)


fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('3. VERTICAL ALIGNMENT') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
```

```
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1_va, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q2_va, 'blue', label='q2')
line13, = ax1.plot(x_mm, AS21_q3_va, 'black', label='q3')
line14, = ax1.plot(x_mm, AS21_q4_va, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# PHASE DRIFT (LINEAR COMPENSATION))

# WARNING: PHASE DRIFT COMPENSATION ONLY WORKS IF THE RFQ IS WEL
L CENTERED IN THE MEASUREMENT SWEEP TIME,
# I.E.: THE DISTANCES TRAVELED BY THE BEAD PULL AT THE INPUT AND
THE OUTPUT OF THE RFQ ARE SIMILAR,
# E.G.: AROUND 10 CM AT BOTH SIDES

def phase_drift(AS21_q1_va, x_mm):
    N = len(x_mm)
    start2 = -100 #end1
    end2 = -200 #start1
    print('len x: '+str(N))
    print(x_mm[start2:end2:-10])
    print(AS21_q1_va[start2:end2:-10])

    phase2_q1 = np.mean(AS21_q1_va[start2:end2:-1])
    print('phase2 (deg):  '+str(phase2_q1))

    phaseSlope_q1 = (phase2_q1 - 0) / ( x_mm[start2-50]-x_mm[-st
art2+50] )
    print('phaseSlope (deg/mm):  '+str(phaseSlope_q1))

    linearDrift_q1 = phaseSlope_q1 * x_mm
    AS21_q1_pd = AS21_q1_va - linearDrift_q1

    return AS21_q1_pd


print('len x_mm: ',len(x_mm))
AS21_q1_pd = phase_drift(AS21_q1_va,x_mm)
AS21_q2_pd = phase_drift(AS21_q2_va,x_mm)
AS21_q3_pd = phase_drift(AS21_q3_va,x_mm)
AS21_q4_pd = phase_drift(AS21_q4_va,x_mm)

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
```

```
ax1.set_title('4. PHASE DRIFT COMPENSATION') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1_pd, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q2_pd, 'blue', label='q2')
line13, = ax1.plot(x_mm, AS21_q3_pd, 'black', label='q3')
line14, = ax1.plot(x_mm, AS21_q4_pd, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# SMOOTHING

#define moving average function
def moving_average1(a, n=3):
    # filtered array has the same length as the original
    ret = np.cumsum(a, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    ma = ret[n - 1:] / float(n)
    ma = np.append(np.zeros(int((n-1)/2)), ma)
    ma = np.append(ma, np.zeros(int((n-1)/2)))
    return ma

ma_n = 21 # n = 15 looks good
AS21_q1_ma = moving_average1(AS21_q1_pd, n=ma_n)
AS21_q2_ma = moving_average1(AS21_q2_pd, n=ma_n)
AS21_q3_ma = moving_average1(AS21_q3_pd, n=ma_n)
AS21_q4_ma = moving_average1(AS21_q4_pd, n=ma_n)

print('Moving average order: ' +str(ma_n))
print(len(AS21_q1),len(AS21_q1_ma))

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('5. SMOOTHING') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid
```

```
line11, = ax1.plot(x_mm, AS21_q1_pd, 'red', label='S21_q1_pd')
line12, = ax1.plot(x_mm, AS21_q1_ma, 'blue', label='S21_q1_ma')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# SMOOTHING
zoomx = 450

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('5.1 SMOOTHING') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(zoomx, zoomx+50) # initial x limits
ax1.set_ylim(AS21_q1_pd[zoomx*10]-0.1, AS21_q1_pd[zoomx*10]+0.3)
# y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1_pd, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q1_ma, 'blue', label='q1_smoothed'
)

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# HORIZONTAL ALIGNMENT

def horizontal_alignment(AS21_q2_ma, AS21_q1_ha, x_mm_q1, phase_
threshold):

    AS21_q1_mad = np.roll(AS21_q1_ma,-1) # desplazamos los eleme
ntos del array 1 posición hacia la izquierda
    ind1 = np.where( (AS21_q1_ma <= phase_threshold) & (AS21_q1_
mad > phase_threshold) )
    ind1 = int(ind1[0][0])
    print('ind1: ', str(ind1))
    AS21_q1_ma[ind1];  AS21_q1_mad[ind1]
    print(phase_threshold, AS21_q1_ma[ind1], AS21_q1_mad[ind1])
    print(len(x_mm_q1))
    print(ind1,x_mm_q1[ind1])

    AS21_q2_mad = np.roll(AS21_q2_ma,-1) # desplazamos los eleme
ntos del array 1 posición hacia la izqueierda
```

```
    ind2 = np.where( (AS21_q2_ma <= phase_threshold) & (AS21_q2_
mad > phase_threshold) )
    ind2 = int(ind2[0][0])
    print('ind2: ', str(ind2))

    AS21_q2_ma[ind2];  AS21_q2_mad[ind2]
    print(AS21_q2_ma[ind2], AS21_q2_mad[ind2])
    print(ind2, x_mm[ind2])

    AS21_q2_ha = np.roll(AS21_q2_ma,-(ind2-ind1))

    return AS21_q2_ha



AS21_q1_ha = AS21_q1_ma
x_mm_q1 = x_mm
phase_threshold = np.max(AS21_q1_ha)/3

print('len AS21_q1_ma: ',len(AS21_q1_ma))
print('len AS21_q2_ma: ',len(AS21_q2_ma))
print('len x_mm_q1: ',len(x_mm_q1))

print('phase_threshold: ',str(phase_threshold))

AS21_q2_ha = horizontal_alignment(AS21_q2_ma,AS21_q1_ha,x_mm_q1,
phase_threshold)
AS21_q3_ha = horizontal_alignment(AS21_q3_ma,AS21_q1_ha,x_mm_q1,
phase_threshold)
AS21_q4_ha = horizontal_alignment(AS21_q4_ma,AS21_q1_ha,x_mm_q1,
phase_threshold)

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('6. HORIZONTAL ALIGNMENT') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('angle(S21) [deg]', color='red') # left y axis la
bel
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, AS21_q1_ha, 'red', label='q1')
line12, = ax1.plot(x_mm, AS21_q2_ha, 'blue', label='q2')
line13, = ax1.plot(x_mm, AS21_q3_ha, 'black', label='q3')
line14, = ax1.plot(x_mm, AS21_q4_ha, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
```

```
ipped
#fig.show()

# SQUARE ROOT OF PHASE AND QUADRUPOLE/DIPOLE SIGNS
AS21_q1_ha[AS21_q1_ha<0] = 0
AS21_q2_ha[AS21_q2_ha<0] = 0
AS21_q3_ha[AS21_q3_ha<0] = 0
AS21_q4_ha[AS21_q4_ha<0] = 0

q1 = np.sqrt(AS21_q1_ha)
q2 = -np.sqrt(AS21_q2_ha)
q3 = np.sqrt(AS21_q3_ha)
q4 = -np.sqrt(AS21_q4_ha)

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('7. SQRT AND QUAD SIGN') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('sqrt(phase)', color='red') # left y axis label
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, q1, 'red', label='q1')
line12, = ax1.plot(x_mm, q2, 'blue', label='q2')
line13, = ax1.plot(x_mm, q3, 'black', label='q3')
line14, = ax1.plot(x_mm, q4, 'cyan', label='q4')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()

# QUADRUPOLE AND DIPOLE MODES

Q = (q1 - q2 + q3 - q4) / 4
D1 = (q1 - q3) / 2
D2 = (q2 - q4) / 2

# normalization
Q = Q / np.max(Q)*100
D1 = D1 / np.max(Q)*100
D2 = D2 / np.max(Q)*100

fig = plt.figure(figsize=(8.5,4.5))
ax1 = fig.add_subplot(111)
ax1.set_title('8. QUADRUPOLE AND DIPOLE MODES') # figure title
ax1.set_xlabel('length [mm]') # x axis label
ax1.set_ylabel('relative field amplitude (%)', color='red') # le
ft y axis label
```

```
ax1.set_xlim(min(x_mm), max(x_mm)) # initial x limits
#ax1.set_ylim(-200, 200) # y limits
ax1.tick_params(axis='y', labelcolor='red') # ticks
ax1.axes.grid('both') #grid

line11, = ax1.plot(x_mm, Q, 'red', label='Q')
line12, = ax1.plot(x_mm, D1, 'blue', label='D1')
line13, = ax1.plot(x_mm, D2, 'black', label='D2')

fig.legend() #(loc='center', bbox_to_anchor=(0.07, 0.7))
fig.tight_layout()  # otherwise the right y-label is slightly cl
ipped
#fig.show()
```