Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Desarrollo de un algoritmo inteligente para optimización de cobertura de un sistema de posicionamiento para entornos complejos

Autor: Javier Beltrán Vázquez

Tutor: Joaquín Bernal Méndez

Dpto. Física Aplicada
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Desarrollo de un algoritmo inteligente para optimización de cobertura de un sistema de posicionamiento para entornos complejos

Autor:
Javier Beltrán Vázquez

Tutor:
Joaquín Bernal Méndez
Profesor Titular

Dpto. Física Aplicada
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Desarrollo de un algoritmo inteligente para optimización de cobertura de un sistema de posicionamiento para entornos complejos

Autor: Javier Beltrán Vázquez
Tutor: Joaquín Bernal Méndez

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

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

El siguiente proyecto se entiende como el punto y final a un ciclo en mi vida, el final de mis estudios en el grado de ingeniería aeroespacial. Muchas son las personas que durante estos largos años de carrera me han ayudado a alcanzar mis metas y crecer, no sólo en el ámbito educativo sino también en el personal; y es por ello que un trocito de este trabajo corresponde a cada uno de ellos.

En primer lugar esto no habría sido posible sin mi familia, que ha estado presente en todo momento para apoyarme en las buenas y en las malas. Gracias a mis padres y a mi hermana por motivarme cuando era necesario y alegrarse de mis éxitos como si de ellos mismos se tratara; y gracias en especial a mi tío Jose y mi tía Margari que han tenido una implicación constante en mis años de estudiante. Todos ellos estuvieron siempre atentos y siempre lo estarán.

En segundo lugar, debo mencionar a aquel grupo de personas que primero fueron compañeros para finalmente convertirse en grandes amigos. Muchos han sido los momentos compartidos y todos ellos excepcionales.

Por último, mencionar a las personas directamente implicadas en este proyecto y en especial a mi tutor Joaquín por su implicación y grata atención.

Javier Beltrán Vázquez
Sevilla, 2021

Resumen

En el presente trabajo se exponen las características de la herramienta software desarrollada en Python con ánimo de encontrar las posiciones de balizas electromagnéticas que permitan obtener un campo magnético óptimo en un recinto dado.

Estas balizas servirán a posteriori en otros proyectos para idear un sistema de localización según diversas técnicas como el posicionamiento por triangulación o el posicionamiento por huellas. De esta forma, se considera necesario desarrollar una herramienta que permita obtener qué posiciones deben tener las balizas electromagnéticas para que creen el campo magnético más beneficioso para utilizar dichas técnicas.

Por tanto, en primer lugar se desglosarán en detalle las restricciones que modelan el problema, como pueden ser las características de los recintos en los que optimizar el campo magnético, las condiciones que debe cumplir un punto del espacio para ser óptimo o la caracterización física de las balizas a utilizar.

En segundo lugar se expondrán las distintas funciones que permitirán encontrar soluciones al problema según las diversas restricciones que hayan sido definidas. En concreto, no sólo se ha desarrollado y se explicará en profundidad la función que permitirá obtener las posiciones óptimas de las balizas, sino que también se expondrán diversas funciones creadas con el fin de estudiar numérica y visualmente las distintas soluciones obtenidas.

Por último, se comentará el funcionamiento y características de la interfaz gráfica para usuarios desarrollada con el fin de poder configurar de forma rápida y sencilla cualquier tipología de problema a resolver y; finalmente se realizará un ejemplo de aplicación real que nos permita comprender la potencia del software desarrollado.

En dicho ejemplo de aplicación podrá observarse que un mismo problema puede ser definido de distintas formas por el usuario y consecuentemente podrán obtenerse distintas soluciones; lo que hace comprender la complejidad del problema y la versatilidad de la herramienta desarrollada.

Abstract

In this work, the characteristics of the software tool developed in Python in order to find the positions of electromagnetic beacons that allow obtaining an optimal magnetic field in a given enclosure are exposed.

These beacons will be used in other projects to develop a location system according to various techniques such as positioning by triangulation or positioning by fingerprinting. In this way, it is considered necessary to develop a tool that allows obtaining which positions the electromagnetic beacons should have so that they create the most beneficial magnetic field to use these techniques.

Therefore, in the first place, the restrictions that model the problem will be broken down in detail, such as the characteristics of the enclosure where the magnetic field must be optimized, the conditions that a point in this enclosure must meet to be optimal or the physical characterization of the beacons used.

Secondly, the different functions that will allow finding solutions to the problem according to the various restrictions that have been defined will be exposed. In particular, not only the function that obtains the optimal positions of the beacons will be explained in depth, but also various functions created in order to study numerically and visually the different solutions obtained.

Last but ot least, the motion and characteristics of the graphical user interface developed to quickly and easily configure any type of problem will be commented and; finally an example of a real application will be shown in order to allow us understanding the power of the software developed.

In that application example, it can be observed that the same problem can be defined in different ways by the user and consequently different solutions can be obtained; what allows us to understand the complexity of the problem and the versatility of the developed tool.

Índice

1 Introducción

Como es bien sabido, los sistemas de posicionamiento son ampliamente utilizados en multitud de ámbitos, ya sea de la vida cotidiana o profesional. Dichos sistemas permiten a los usuarios conocer su localización; siendo de gran ayuda en caso de encontrarse en entornos desconocidos o en los que es difícil orientarse de forma correcta.

Con este propósito se han desarrollado distintas herramientas, como pueden ser los sistemas GPS (*"Global Positioning System"*) o IPS (*"Indoor Positioning System"*) [1]. En concreto, los sistemas IPS se sustentan en una amplia variedad de tecnologías, como pueden ser la tecnología Wi-Fi, la tecnología Bluetooth o la tecnología RFID, entre otras.

Estos sistemas basan su funcionamiento en el envío de señales electromagnéticas a partir de distintos emisores (Routers, dispositivos bluetooth, etc) y su recepción por parte de un dispositivo receptor. El posicionamiento se realiza mediante dos métodos principales:

- **Posicionamiento por Triangulación [2]:** Conocida la posición de 3 o más emisores de referencia, es posible calcular la posición del receptor. Éste mide la intensidad de la señal generada por cada emisor, o el tiempo que tarda la señal en llegar al receptor; de forma que es posible conocer la distancia emisor-receptor. Dado que conocemos la posición del emisor, el repector se encontrará en una superficie esférica con centro la posición del emisor y radio la distancia emisor-receptor. Conocidas 3 o más de estas esferas, el receptor se situará en el punto de intersección de dichas esferas.

- **Posicionamiento por huellas ("Fingerprinting"):** Conocida la posición de los emisores y el campo electromagnético que producen en el entorno, se genera un mapa de referencia. Este mapa consiste en un mallado de puntos en los que se conocen los datos anteriores, es decir, se conoce la intensidad de campo generada por cada emisor en cada uno de los puntos del mapa. De esta forma, una vez el receptor mide la distribución del campo en una posición cualquiera, bastará con buscar qué punto del mapa coincide con lo que se está midiendo.

Es importante destacar que todas estas tecnologías se basan en el envío de ondas electromagnéticas a altas frecuencias. Esto tiene un inconveniente principal: el apantallamiento producido por materiales metálicos debido a dos efectos principales, reflexión y absorción. El apantallamiento crece fuertemente con la frecuencia de la señal transmitida, de forma que las tecnologías anteriores podrán verse muy restringidas en entornos metálicos en los que las señales puedan verse atenuadas, impidiendo el correcto funcionamiento del sistema.

1.1 Proyecto SILEME

El proyecto *SILEME* (Sistema Interior de Localización en Entornos Metálicos) elaborado por la empresa *Skylife Engineering* propone como solución al problema anterior el uso de balizas generadoras de campo magnético a bajas frecuencias. De esta forma, se reduce de forma eficiente el problema de apantallamiento, dado que como se mencionó anteriormente, crece fuertemente con la frecuencia.

La herramienta elaborada se basará en los métodos de localización mencionados anteriormente, de forma que midiendo el campo magnético producido por las balizas en cierto punto, podremos conocer la posición del sensor (receptor) comparando con un mapeo previo almacenado en una base de datos o mediante triangulación.

Dada esta forma de operar, podemos concluir que la distribución de campo electromagnético generada por las balizas en el entorno es de vital importancia. Esto se debe a:

- **Sensibilidad del Sensor:** los sensores estarán formados por una serie de bobinas, de forma que el campo electromagnético inducirá en ellas una diferencia de potencial. Midiendo esta diferencia de potencial, seremos capaces de medir el campo magnético en el punto en el que se encuentra el sensor. Dicho esto, para el correcto funcionamiento de los sensores, se necesitará detectar un valor de campo mínimo (sensibilidad electromagnética) que nos permita detectar el campo de forma precisa.

- **Número mínimo de balizas para triangulación:** Si se pretende realizar localización por triangulación, será necesario que el campo producido por al menos 3 balizas sea superior a la sensibilidad del sensor.

1.2 Objetivo del Trabajo

Atendiendo a lo expuesto anteriormente, parece necesario desarrollar una herramienta que nos permita encontrar las posiciones de las balizas que optimizan el campo magnético en un recinto dado; teniendo en cuenta restricciones como la sensibilidad del sensor, el número de balizas o el recinto en el que optimizar el campo. Estas restricciones serán analizadas en profundidad más adelante.

Una primera versión de un software de optimización de la ubicación de las balizas en un problema de localización de este tipo fue realizada por el alumno Juan de Dios Muñoz Guzmán en su proyecto fin de carrera [3]. Su desarrollo se realizó en MATLAB, atendiendo a restricciones mucho más laxas que las que se presentarán en el siguiente proyecto. Por tanto, el primer objetivo será mejorar el trabajo previamente hecho; mejorando tiempos de ejecución, configurabilidad del sistema y capacidad de definición de restricciones.

Realizar comparativas objetivas de eficiencia entre ambos proyectos no será posible dado que se introducen diferencias notables de funcionamiento ya desde el primer momento. Las mejoras más significativas que se han introducido en el software respecto la versión preliminar son las siguientes:

- En primer lugar, mientras que la versión anterior sólo tiene en cuenta problemas bidimensionales, el siguiente trabajo se extenderá a problemas de cualquier tipo, pudiendo definir cualquier región bidimensional o tridimensional deseada.

- Se añade la posibilidad de definir las regiones en las que pueden moverse las balizas a la hora de realizar la optimización; e incluso añadir regiones en las que no estará permitido situar ninguna de estas balizas.

- Por otro lado, el algoritmo que permite determinar si un punto es óptimo también será profundamente modificado. Mientras que la primera versión sólo tenía en cuenta problemas con 3 balizas, aquí se extenderá a un número indefinido de éstas y además se cambiará la filosofía de cálculo[1] para mejorar en tiempos de ejecución y en eficiencia de uso de memoria.

- Esta nueva versión se desarrollará por completo en lenguaje Python, dados su versatilidad, su alto grado de aceptación y uso en el ámbito profesional y el hecho de que se trata de una herramienta de acceso abierto. El uso de Python se plantea también en este trabajo por su valor pedagógico, ya que permitirá al autor profundizar en el aprendizaje de un nuevo lenguaje de programación muy útil en el ámbito de ingeniería. El interés de esto estriba en que son pocos los lenguajes de programación impartidos con cierto grado de profundidad a lo largo del grado de Ingeniería Aeroespacial; habiendo sido MATLAB el más usado a lo largo del grado.

- Será necesario dotar a la herramienta de la posibilidad de estudiar el campo magnético resultante tras la optimización, pues será de gran utilidad poder obtener qué zonas del recinto cumplen con las restricciones, observar visualmente la posición de las balizas en el recinto, la distribución de campo resultante o las zonas en las que el campo magnético supera los límites permitidos de radiación.

- Finalmente, se agruparán todas las funcionalidades en una interfaz gráfica para usuarios (GUI) que permita configurar todas las opciones de forma sencilla e intuitiva, guiando al usuario por todos los pasos a seguir para definir el problema, dando valores de referencia de los distintos parámetros y mostrando los resultados de la forma más visual posible.

[1] El cálculo de puntos óptimos será desarrollado en profundidad en el capítulo 4: Función de coste

2 Restricciones y requisitos del Proyecto

Como se ha mencionado previamente, el proyecto parte de los requisitos que debe cumplir un punto del recinto para ser óptimo; estando éstos expuestos en profundidad en la sección 2.2 del presente capítulo. A partir de estos requisitos, se pretende encontrar las posiciones de las balizas que maximicen el número de puntos óptimos; es decir, aumentar el volumen en el que se cumplen los requerimientos. Además, deberemos ser capaces de definir una serie de restricciones del problema, como el recinto a estudiar, volúmenes en los que no podremos colocar las balizas o el número de balizas disponibles. Nos disponemos en esta sección a analizar los requerimientos y restricciones en profundidad.

2.1 Restricciones al problema

- **Región de optimización:** Debe poder definirse la región donde se tratará de optimizar el campo magnético. Por ejemplo, si queremos optimizar el campo en un pasillo con forma de L, deberemos ser capaces de definir sus dimensiones para optimizar el campo únicamente en dicho pasillo.

  Además, puede ser interesante optimizar no sólo regiones volumétricas, sino también superficies planas. De esta forma, si queremos orientarnos en un pasillo, generalmente una persona de altura media llevará el sensor a una altura de unos 1.5m; de forma que lo que nos interesará será optimizar el campo en un plano a 1.5m de altura.

- **Posición de las balizas:** Será necesario poder restringir el movimiento de las balizas. Volviendo al ejemplo del pasillo, es lógico pensar que no se podrán colocar balizas en medio de éste, sino que será necesario colocarlas en el techo o en los laterales del pasillo.

  Por tanto, deberemos poder definir los espacios volumétricos o planares en los que se podrán mover cada una de las balizas. Las balizas deben poder colocarse dentro o fuera del recinto a optimizar. La orientación también debe ser configurable dado que en ocasiones podremos girar la baliza como más nos convenga, y en otras deberán tener una orientación fija; por ejemplo si las colocamos en el techo de una habitación.

  Por último, habrá situaciones en las que será más fácil indicar dónde no pueden situarse las balizas en vez de dónde sí pueden situarse. Como ejemplo, volvemos a recurrir al pasillo. Si

queremos colocar las balizas en los laterales del pasillo, será más fácil indicar un volumen prohibido (dentro del pasillo) que indicar todos y cada uno de los laterales.

- **Parámetros de optimización:** Debe ser posible configurar los parámetros en los que se basa la optimización del problema, como es el mallado del recinto en el que optimizar el campo; definiendo la distancia entre puntos de la malla, el número máximo de iteraciones al realizar la optimización o el número de balizas de las que se dispone.

- **Parámetros del sensor:** El sensor utilizado para medir el campo tendrá una serie de parámetros que deben poder ser configurables. En primer lugar, la sensibilidad del sensor, que es el campo mínimo que podrá detectar. En segundo lugar, los factores de calibración (C) y de frecuencia (F). Esto se debe a que el voltaje medido por el sensor dependerá, por un lado de su calibración inicial (parámetros de fabricación), y por otro lado, para sensores basados en Ley de Faraday, este voltaje será típicamente proporcional a la frecuencia (f) en la que emiten las balizas. De esta forma, el voltaje que mide un sensor se expresará en general como: $V_{medido} = F(f)CB$. Si se tiene un sensor cuyo V_{medido} es independiente de f, tendríamos $F(f) = 1$.

2.2 Requisitos

- **Cálculo de campo magnético:** Dado que las balizas serán en general espiras de corriente, y que típicamente interesará el campo magnético a distancias de las balizas mayores que sus propias dimensiones, el campo magnético generado por éstas se calculará a partir del campo magnético producido por un dipolo magnético. Debe ser posible definir las dimensiones de las balizas y sus características principales, como la intesidad que las recorre (I) o el número de espiras que forma a cada una.

- **Requisito Nº1 de Punto Óptimo. Número mínimo de balizas cuyo campo debe ser superior a la sensibilidad del sensor:** En primer lugar, para que sea posible el posicionamiento por triangulación, será necesario que, en cada uno de los puntos de la región que se desee cubrir, haya un número mínimo de balizas cuyo campo magnético sea superior a la sensibilidad del sensor. En general, siendo capaces de detectar el campo de 3 balizas será suficiente. Aún así, se deberá implementar la posibilidad de definir el número mínimo de balizas que deben generar un campo superior a la sensibilidad del sensor en un punto para que este sea óptimo.

- **Requisito Nº2 de Punto Óptimo. Diferencia de magnitud entre balizas:** Como requisito adicional, se pide que la diferencia de campo magnético entre n pares de balizas no sea mayor a 2 órdenes de magnitud[1]. Es decir, si tenemos 5 balizas [A1,A2,...,A5] cuyos módulos de campo magnético son $[B1,B2,...,B5]$; definiendo $n = 3$, deberá ocurrir que tengamos 3 pares de balizas que cumplan (2.1).

$$0.01 < B_i/B_j < 100 \quad \forall i,j \in \tag{2.1}$$

[1] Aunque generalmente serán 2 órdenes de magnitud, este parámetro también deberá ser configurable.

- **Representación de la solución:** El resultado tras optimización debe ser representado visualmente. Deben poder representarse gráficamente los puntos de la malla que, dentro del recinto de interés, cumplen con los requisitos y por tanto son óptimos, junto con la representación gráfica de la posición de las balizas y su orientación con respecto al recinto estudiado. Por otro lado, también debe poder obtenerse una representación del campo producido por las balizas en un plano X, Y ó Z; que permita estudiar y evaluar el campo resultante.

  Por último, se requiere que se muestren las zonas dentro del recinto o su entorno que superen los umbrales de campo magnético permitidos por la normativa que limita la exposición humana a campos electromagnéticos. En concreto, se distinguen 3 normativas de referencia, siendo [4] la normativa vigente en el ámbito público español, [5] la normativa vigente europea en el ámbito laboral y [6] la trasposición de esta última a la legislación española.

3 Campo generado por un dipolo magnético

En primer lugar se procede a obtener el campo magnético producido por una baliza en un punto cualquiera del espacio $[x,y,z]$ (ejes globales); estando dicha baliza situada en el punto $[x_0,y_0,z_0]$ y girada unos ángulos $[\theta,\phi]$, siendo estos los giros alrededor de los ejes Z y X respectivamente. La configuración del problema se muestra en la Fig.3.1.

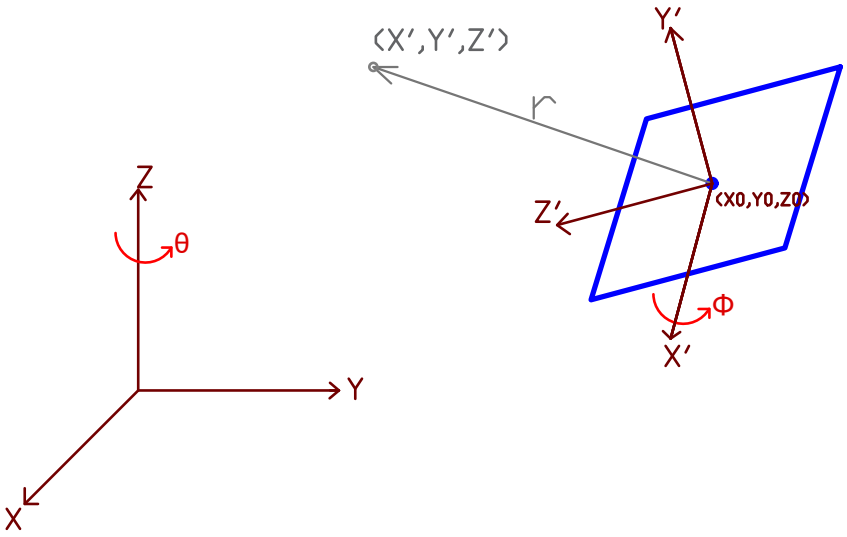


Figura 3.1 Configuración del problema: cálculo campo generado por un dipolo.

Se parte del campo producido por un dipolo magnético. La distribución de campo magnético que genera un dipolo (en ejes cuerpo) viene dada por (3.1); siendo $\vec{r} = [x',y',z']^T$ la posición del punto en el que obtener el campo y $\vec{m} = j(I*A*N_{espiras})$ el momento dipolar magnético (espiras orientadas en el eje Y). Como parámetros de la baliza encontramos la intensidad que la recorre (I), el área de la baliza (A) y el número de espiras que tiene ($N_{espiras}$). Esta es una ecuación vectorial, de forma que proporciona los módulos del campo en los ejes de referencia; es decir, obtenemos $[Bx,By,Bz]$.

$$\vec{B} = \frac{\mu_0}{4\pi}\frac{3(\vec{m}\vec{r})\vec{r} - r^2\vec{m}}{r^5} \tag{3.1}$$

Dado que la ecuación (3.1) viene dada en ejes cuerpo, necesitaremos calcular la matriz de rotación (*Rot*) según los ángulos de giro θ y ϕ que nos permita obtener el punto $[x',y',z']$ en ejes cuerpo a partir del punto $[x,y,z]$ en ejes globales. Además, esta misma matriz nos permitirá pasar del campo $\vec{B}$ obtenido en ejes cuerpo al campo magnético producido en ejes globales. El cálculo de dicha matriz se realiza con la función **MatRotacion** y se muestra en el **Código A.1**. Una vez tenemos la matriz de rotación estamos en disposición de poder calcular el campo magnético producido por una baliza en cualquier punto del espacio. Este cálculo se realiza con la función **Dipolo** que se muestra en el **Código A.2**. Sus variables de entrada y salida se muestran en la Tabla 3.1.

Tabla 3.1 Variables de entrada y salida de la función **Dipolo.**

Variables de entrada	
Pos_antena	Posición [x0,y0,z0] donde se encuentra la baliza
Pos_sensor	posición [x,y,z] donde se mide el campo
Semilado	longitud del semilado de la baliza
I	Intensidad que recorre la baliza
N_espiras	Número de espiras que tiene la baliza
Rot	matriz de rotación según los ángulos de giro de la baliza θ y ϕ

Variables de salida	
B	campo [Bx,By,Bz] en ejes globales
modB	módulo del campo magnético

En resumen, se transforma el punto $[x,y,z]$ al punto $[x',y',z']$ y se calcula el campo magnético en ejes cuerpo de la baliza, para finalmente obtener el campo $\vec{B} = [Bx,By,Bz]$ en ejes globales mediante la matriz de rotación.

3.1 Representación líneas de campo generadas por balizas

El cálculo del campo magnético será el pilar de este trabajo. Por tanto, es necesario asegurarse de hacerlo de manera correcta. Por ello, se realiza una función que dadas N balizas, represente las líneas de campo de dichas balizas en un plano 2D y permita observar una distribución coherente. El plano se malla con un número de puntos concreto establecido, por ejemplo un mallado de 64x64 puntos, en los que se calculará el campo magnético vectorial producido por cada baliza y se sumarán sus contribuciones. A partir del vector campo magnético resultante en cada punto se representan las líneas de campo[1]. Esto se realiza en el **Código A.3**.

Se realizan dos pruebas distintas con distinto número de balizas y distinta colocación de estas mismas en el plano Z=0. La distribución de balizas para cada prueba se muestra en la Tabla 3.2 y los resultados obtenidos se muestran en la Fig.3.2. Fíjándonos en ellos, puede observarse una distribución muy coherente, en la que cada baliza produce un campo magnético saliente por uno de sus lados y entrante por el otro. Las líneas de campo de las balizas comienzan a mezclarse entre ellas pero nunca se cortan. Por último, el sistema de colores nos muestra cómo en el centro de las balizas se producen las mayores magnitudes de campo (más brillante); mientras que al alejarnos éste comienza a disminuir (más oscuro).

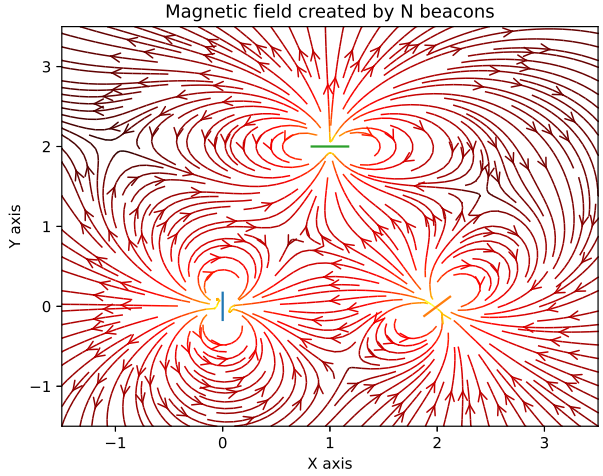
[1] Código basado en la representación de campos vectoriales con Matplotlib [7]

Como puede observarse, la representación mediante líneas de campo es de gran utilidad a la hora de observar si el campo producido por el software es el correcto. Sin embargo, puede no ser tan útil a la hora de mostrar el valor del campo producido en todos los puntos del mapa. Es por ello que más adelante, en la sección 6.3 del capítulo 6, se desarrollará una función que representará el valor del módulo del campo magnético producido mediante un mapa de color (Fig.3.3).

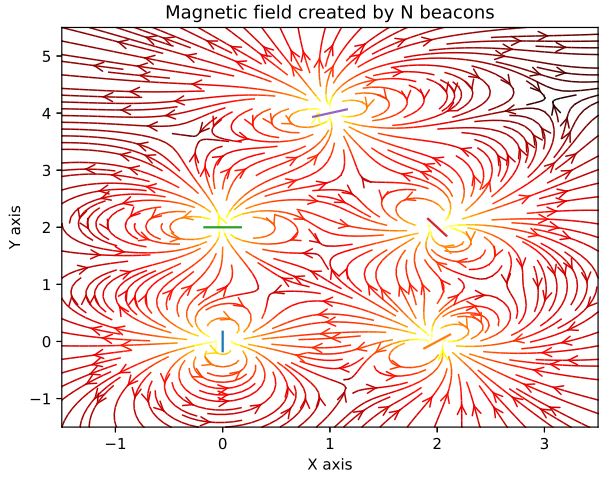
Tabla 3.2 Posicionamiento de balizas en Pruebas 1 y 2.

Prueba 1		
Baliza	**Posición** $[x,y,z]$	**Ángulo de giro** θ
1	0,0,0	$\pi/2$
2	2,0,0	$\pi/4$
3	1,2,0	0

Prueba 2		
Baliza	**Posición** $[x,y,z]$	**Ángulo de giro** θ
1	0,0,0	$\pi/2$
2	2,0,0	$\pi/4$
3	0,2,0	0
4	2,2,0	$-\pi/3$
5	1,4,0	$\pi/8$



(a) Prueba 1.



(b) Prueba 2.

Figura 3.2 Resultados representación campo magnético creado por N balizas.

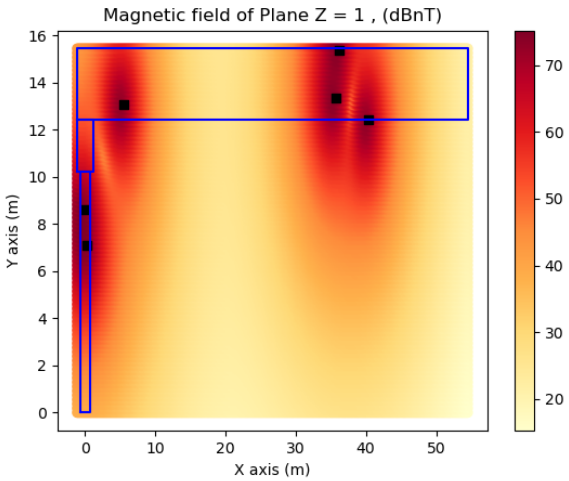


Figura 3.3 Ejemplo mapeo de módulo de campo magnético (sección 6.3).

4 Función de coste

Una vez se dispone de la función **Dipolo** que permite obtener el campo producido por una baliza en cualquier punto del espacio, se está en condiciones de implementar el algoritmo que permite decidir si un punto del espacio es óptimo o no es óptimo. Posteriormente, haciendo uso de dicho algoritmo se implementará una función de coste que dará el número de puntos óptimos en un recinto dado para una distribución de balizas determinada. A la hora de optimizar (capítulo 5), éste será el parámetro que se tratará de maximizar mediante el movimiento de las distintas balizas.

Recordemos que un punto será óptimo si cumple con dos requisitos fundamentales: el campo generado en dicho punto debe ser superior a la sensibilidad del sensor para un número mínimo de balizas y debe haber un número mínimo de parejas de balizas cuyos campos no difieren más de n órdenes de magnitud, siendo n un parámetro configurable.

En el **Código A.4** se realiza un script en el que proporcionando las posiciones de las balizas en formato $[x_1,y_1,z_1,x_2,y_2,z_2,...]$ y sus ángulos de giro dados como $[\theta_1,\theta_2,\theta_3,...]$ y $[\phi_1,\phi_2,\phi_3,...]$ permite obtener y representar los puntos óptimos en un recinto dado. En esta primera versión, el recinto en el que estudiar el número de puntos óptimos se define como un único prisma rectangular de dimensiones $[(x_{max},x_{min}),(y_{max},y_{min}),(z_{max},z_{min})]$. Todas estas coordenadas serán referenciadas respecto un origen de referencia arbitrario. Es decir, el origen podrá colocarse por ejemplo en una esquina del volumen, en una posición cualquiera de una de las balizas o en cualquier punto arbitrario. Lo realmente importante será la colocación de las balizas respecto del volumen a estudiar. En caso de querer estudiar recintos más complejos, estos se estudiarán como unión de prismas como se verá más adelante. También será posible definir el mallado dentro de este volumen, definiendo el número de puntos en cada eje (n_x,n_y,n_z) o la distancia entre puntos (d) .

4.1 Algoritmo de búsqueda de puntos óptimos

La parte más importante en esta sección es el algoritmo que permite decidir si un punto es óptimo o no es óptimo según los requisitos expuestos en secciones anteriores. Éste debe ser lo más rápido y eficiente posible dado que será la base de la optimización. A la hora de optimizar, la idea es buscar las posiciones de las balizas que maximizan el número de puntos óptimos; de forma que se calcularán los puntos óptimos para distintas distribuciones de balizas y se compararán los resultados. Por lo tanto, se tendrán que estudiar todos los puntos del recinto, viendo si estos son óptimos o no óptimos para cada distribución. Por ejemplo, si tenemos un recinto con 50.000 puntos y queremos estudiar 100 distribuciones distintas; el algoritmo se ejecutará 50.000x100 veces. Este es un número muy elevado, de forma que debemos reducir los tiempos lo máximo posible.

En el **Código 4.1** se expone el extracto del **Código A.4** encargado de determinar si un punto es o no óptimo. Éste se explica a continuación.

Código 4.1 Algoritmo búsqueda de puntos óptimos.

```
#########################################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
#########################################################################
n_antenas_sens=3 #nº minimo de balizas que deben cumplir requisitos de
    sensibilidad en un punto
sensibilidad=2.679319e-09 #(T)

n_antenas_dif=3 #nº de pares de balizas cuya diferencia entre módulos
    debe cumplir con los requisitos
diferencia_magnitudes=0.01 #Dado en la forma: B1>0.01B2

#Creación matriz con matrices de rotación: Creamos una única matriz con
    todas las matrices de rotación de todas las balizas. Será una matriz
     de dimensiones [3,3*N] que podremos usar sin tener que calcular
    constantemente dichas matrices:

Rotaciones=MatRotacion(theta[0],phi[0])
for n in range(1,N):
    Rot=MatRotacion(theta[n],phi[n])
    Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)

Puntos_Optimos=0 #variable donde guardamos el nº de puntos optimos

for i in range(nx):
    for j in range(ny):
        for k in range(nz):

            #PRIMERA BALIZA:
            Pos_sensor=[x[i],y[j],z[k]]
            Pos_antena_n=[Pos_antena[0],Pos_antena[1],Pos_antena[2]]
            [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado[0], I[0],
                N_espiras[0], Rotaciones[0:3,0:3])
            modB=f[0]*c*modB

            CAMPOS=[modB] #Lista donde iremos guardando los modulos de
                las balizas [B1,B2,B3...]

            dif_aux=0 #pares de balizas que cumplen requisitos de
                diferencia de modulos

            sens_aux=0 #numero de balizas que cumplen requisitos de
                sensibilidad

            if modB>sensibilidad:
                #Si cumple con sensibilidad añadimos 1 baliza que cumple
```

```
                sens_aux+=1

            l=3 #Repretimos este procedimiento para resto de balizas:
            opt=1 #variable auxiliar
            for n in range(1,N):
                if opt == 1: #si ya es óptimo, pasamos a otro punto
                    continue
                Pos_sensor=[x[i],y[j],z[k]]
                Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],Pos_antena[l
                    +2]]
                [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado[n],I[n],
                    N_espiras[n],Rotaciones[0:3,l:l+3])
                modB=f[n]*c*modB

                CAMPOS.append(modB) #añadimos campos [B1,B2,B3...]

                for r in range(n):
                #comparamos el modulo de esta baliza 'n' con las balizas
                    anteriores 'r=1,2,3...'

                    if ((modB>CAMPOS[r] and CAMPOS[r]>
                        diferencia_magnitudes*modB)
                        or (CAMPOS[r]>modB and modB>diferencia_magnitudes*
                            CAMPOS[r])):

                        #EJ: si estamos en la 3a baliza (n=2--->r=[0,1]),
                            comparamos B3/B1 y B3/B2

                        dif_aux+=1 #si la diferencia entre los módulos
                            cumple los requisitos, añadimos un par


                if modB>sensibilidad:
                    sens_aux+=1 #Si cumple con sensibilidad añadimos 1
                        baliza

                #si tenemos el nº mínimo de balizas con sensibilidad
                    suficiente y el nº mínimo de pares de balizas que
                    cumple con la diferencia de modulos, añadimos
                    punto_opt y pasamos al siguiente punto:

                if dif_aux>=n_antenas_dif and sens_aux>=n_antenas_sens:
                    Puntos_Optimos+=1
                    opt=1
                    continue

                l+=3

Puntos_Totales=nx*ny*nz
Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos
```

Partimos de una distribución de balizas dada y un volumen mallado a estudiar (Código A.4). En primer lugar deben definirse los requisitos mediante los siguientes parámetros: *n_antenas_sens*, *sensibilidad*, *n_antenas_dif* y *diferencia_magnitudes*. A través de ellos, definimos el mínimo número de balizas cuyo campo debe ser mayor a la sensibilidad del sensor junto con el mínimo número de pares de balizas cuya diferencia entre campos no puede superar la diferencia $Bmin < 0.01Bmax$ (si *diferencia_magnitudes* = 0.01).

Una vez definidos estos parámetros, lo primero será crear la matriz *Rotaciones* en la que almacenaremos todas las matrices de rotación correspondientes a cada una de las balizas. Con esto evitamos tener que calcular la matriz de rotación (*Rot*) de cada baliza más de una vez. dependiendo de la baliza que estemos estudiando, sólo tendremos que acceder a su matriz correspondiente dentro de la matriz *Rotaciones* = $[Rot_1, Rot_2, ..., Rot_N]$.

A continuación se procede a analizar todos los puntos de la malla uno por uno. La forma de determinar si un punto es óptimo será la siguiente:

- 1) Calcular campo generado por la primera baliza ($B1$) y guardarlo en la variable auxiliar *CAMPOS* en la que se almacena el campo producido por cada baliza en el punto a estudiar. Si cumple requisito de sensibilidad, se añade una baliza que cumple con dicho requisito en la variable *sens_aux*. El segundo requisito no puede estudiarse dado que se necesitan al menos 2 balizas.

- 2) Calcular campo de la segunda baliza ($B2$) y almacenar en *CAMPOS* = $[B1, B2]$. Comprobar requisitos; si su sensibilidad es mayor que la requerida se añade una baliza que cumple con sensibilidad. Comprobar diferencia entre magnitudes: debe compararse B2 con B1. Si la diferencia entre magnitudes cumple con los requisitos se añade un par de balizas que cumplen con este requisito en la variable *dif_aux*. Dado que *CAMPOS* se reinicia al cambiar de punto, y es la única matriz usada en todo el algoritmo, la matriz de mayores dimensiones que podrá encontrarse será como máximo un vector de dimensión el número de balizas; haciendo por tanto un uso eficiente en memoria.

- 3) Realizar este bucle para las sucesivas balizas. La forma de comparar las magnitudes de las balizas será la siguiente: dado el campo de una nueva baliza (Bn), debe compararse ésta nueva baliza con el campo de las balizas anteriores; es decir, deben hacerse las comparaciones (Bn,$B1$), (Bn,$B2$), (Bn,$B3$)... De esta forma no se harán más comparaciones de las necesarias.

- 4) Una vez alcanzado el número mínimo de balizas que deben cumplir con el requisito de sensibilidad y el número mínimo de pares de balizas que cumplen con el requisito de diferencia de magnitudes, se asigna punto como óptimo y se deja de calcular en dicho punto. De esta forma, si se tienen 100 balizas y se cumple con los requisitos con las 4 primeras, nos ahorramos hacer el bucle y calcular los campos para 96 balizas.

- 5) Si se ha pasado por todas las balizas y no se ha cumplido con los requisitos, el punto será no óptimo y se pasa al siguiente punto.

La diferencia fundamental con la primera versión que se realizó del proyecto reside en este algoritmo. En aquella primera versión, se calculaba y almacenaba el campo producido por todas las balizas en todos los puntos. De esta forma, el uso de memoria era excesivo; en caso de tener 100 balizas y 10,000 puntos de mallado, se requería almacenar 100x10,000 datos. Además, el tiempo de ejecución tampoco era óptimo, dado que se calculaba el campo producido por todas las balizas independientemente de haber podido determinar que el punto era óptimo con solo 3 o

4 balizas. Dado que sólo se tenían 3 balizas podría no ser un gran problema, pero si se requiere hacer cálculos para un alto número de balizas y un mallado muy denso, puede ser muy problemático.

Como se ha podido ver, esto ha sido ampliamente mejorado, dado que si determinamos que un punto es óptimo, dejamos de hacer cálculos para el resto de las balizas; y el mayor espacio en memoria utilizado será igual al número de balizas de las que se dispone (junto con la definición de las constantes).

El algoritmo también cambia en la forma de determinar si un punto es óptimo. La primera versión que se implementó sólo tenía en cuenta 3 balizas, de forma que el requisito de diferencia de magnitud entre balizas era distinto. En concreto, se pedía que el mayor campo producido por una de las balizas no superara en más de dos órdenes de magnitud al campo más pequeño producido por una de estas balizas.

Esto no es implementable si tenemos un recinto de grandes dimensiones con un gran número de balizas. Como ejemplo, imaginemos una aeronave. Si estudiamos un punto situado en la cola del avión, las balizas situadas en esta zona superarán muy ampliamente a las balizas que se encuentren en la cabeza de dicho avión. De esta forma, si se compara el mayor campo producido por cualquier baliza en dicho punto con el menor campo producido también por cualquier baliza, se obtendrá que $Bmin < 0.01 Bmax$ y por tanto siempre se determinaría que el punto es no óptimo.

Es por ello que se cambió este requisito a la forma explicada en el capítulo de requisitos, en el que debe ocurrir que la diferencia entre módulos de varios pares de balizas cumpla con el requisito de diferencia de magnitudes.

4.2 Tiempos de ejecución y representación de puntos óptimos

A continuación se realizan dos pruebas para dos volúmenes de estudio distintos consistentes en prismas rectangulares (uno más grande que otro) y una distribución de 3 balizas fija[1]. Para cada volumen se probarán distintos mallados. La idea será estudiar el tiempo de ejecución del algoritmo de cálculo dependiendo del número de puntos a estudiar y dependiendo de la cantidad de puntos óptimos; dado que a priori, no sería lo mismo un volumen con un 100 % de puntos óptimos frente a otro con un 50 % de puntos óptimos.

En la Tabla 4.1 se muestran los valores del volumen estudiado en cada prueba y en la Tabla 4.2 se muestran los resultados de tiempos de ejecución y el número de puntos óptimos. Como variables principales del problema encontramos las variables de las que depende que un punto sea o no óptimo. En concreto, se han tomado los siguientes valores: *n_antenas_sens* = 3, *sensibilidad* = $2.679319 \cdot 10^{-9}$, *n_antenas_dif* = 3 y *diferencia_magnitudes* = 0.01 [2].

Se observan valores de tiempos de ejecución bastante aceptables, no superando los 3 minutos con un mallado de puntos del orden de 10^6 puntos. El recinto con mayor porcentaje de puntos óptimos será el recinto más pequeño (prueba 1) dado que las balizas tendrán un menor volumen que cubrir. Cuanto más grande sea el volumen en el que optimizar el campo, menor será el porcentaje de puntos óptimos. Además, la diferencia de tiempos entre el recinto con mayor porcentaje de puntos óptimos (Prueba 1) y el recinto con un menor porcentaje de puntos óptimos (Prueba 2) con un mismo número

[1] La distribución de balizas ha sido elegida de forma aleatoria y se ha mantenido en ambas pruebas. No es de gran interés, dado que la importancia de estas pruebas reside en el número de puntos óptimos y el tiempo de ejecución.

[2] Los parámetros n_antenas_sens y diferencia_magnitudes se toman como 3 pues se necesitan 3 balizas para posicionamiento por triangulación. El valor de sensibilidad es el propuesto por Skylife.

Tabla 4.1 Dimensiones volúmes estudiados (m).

	xmax	xmin	ymax	ymin	zmax	zmin
Prueba 1	4	-1	4	-1	4	-1
Prueba 2	7	-4	7	-4	7	-4

Tabla 4.2 Tiempos de ejecución función de coste.

Prueba 1			
Número de puntos (mallado)	**Tiempo de ejecución (s)**	**Número de puntos óptimos**	**% Puntos óptimos**
12,167	2.3874	10,938	89.8999
175,616	28.5276	157,114	89.4645
1,367,631	202.8674	1,221,731	89.3319

Prueba 2			
Número de puntos (mallado)	**Tiempo de ejecución (s)**	**Número de puntos óptimos**	**% Puntos óptimos**
12,167	2.6026	4,974	40.8811
175,616	24.9079	76,907	43.7927
1,367,631	194.5770	613,632	44,8682

de puntos a estudiar no es destacable. De esta forma, no será crítico que se tenga un mayor o menor número de puntos óptimos en el recinto a estudiar. La representación gráfica para la Prueba 2 con un mallado de 12,167 puntos se muestra en la Fig.4.1. La forma de realizar esta representación se comentará en el capítulo 6: *Representación de la solución*.

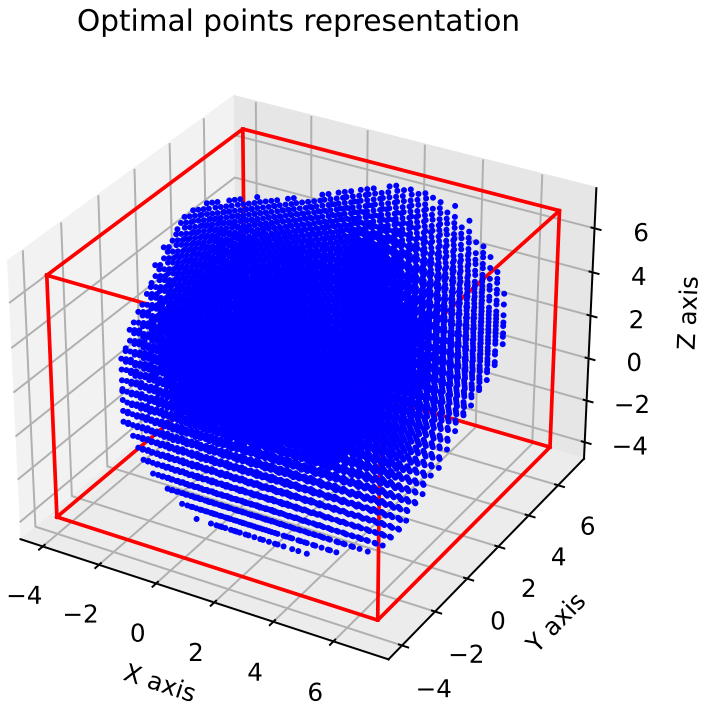


Figura 4.1 Puntos óptimos Prueba 2: mallado de 12,167 puntos.

5 Optimización: Cálculo de posiciones óptimas de balizas

Obtenida la función de coste, que proporciona el número de puntos óptimos (y por tanto también el número de puntos no óptimos) en un prisma rectangular, se procede a realizar la función que permitirá optimizar el número de dichos puntos óptimos en un recinto mediante el correcto posicionamiento de las balizas. Para ello se hace uso del algoritmo de optimización por enjambre de partículas [8] mediante la función **pso** proporcionada por el paquete *pyswarm* de Python [9].

5.1 Parámetros configurables del problema

La herramienta de optimización debe ser flexible y debe permitir configurar los parámetros comentados en el Capítulo 2. El conjunto de estos parámetros se muestran en la Tabla 5.1.

El **Código A.6** consiste en un script en el que se configuran todos los parámetros expuestos en la tabla anterior junto con otros parámetros de representación de la solución y se llama a las distintas funciones de optimización y de representación de la solución que se comentarán más adelante. A continuación, en el **Código 5.1** mostramos el extracto de dicho código que se encarga de configurar las variables expuestas en la Tabla 5.1.

Código 5.1 Definición de parámetros del problema de optimización.

```
#DATOS BALIZAS
semilado=[0.17,0.17,0.17]    #(m)
N_espiras=[84,84,84]         #Numero de espiras
I=[0.707,0.707,0.707]        #Intensidad que circula por las balizas

#DATOS CALIBRACION
c=1                          #Calibracion
f=[1,1,1]                    #factor de conversion Vmedido=f*C*B

########################################################################
"""PARAMETROS OPTIMIZACION"""
########################################################################

N=3              #numero de balizas
```

Tabla 5.1 Parámetros configurables de optimización.

	Parámetros balizas
semilado	Lista: Longitud del semilado de las balizas (m)
N_espiras	Lista: Número de espiras de las balizas
I	Lista: Intensidades que recorren las balizas (A)
C	Factor de calibración de las balizas
F	Lista: Factor de conversión F(f) según frecuencia de balizas

	Parámetros optimización
N	Número de balizas de las que se dispone
d	Distancia entre puntos al realizar el mallado
maxiter	Número de iteraciones al realizar la optimización mediante el algoritmo de enjambre
swarmsize	Número de semillas para búsqueda de puntos óptimos mediante el algoritmo de enjambre
n_antenas_sens	Número de antenas cuyo campo debe superar la sensibilidad del sensor para que un punto sea óptimo
sensibilidad	Sensibilidad del sensor (T)
n_antenas_dif	Número de pares de antenas cuya diferencia entre modulos debe ser inferior a la máxima permitida
diferencia_magnitudes	Máxima diferencia de magnitud entre dos antenas permitida
VOL	Recinto en el que optimizar el número de puntos óptimos. Volúmenes complejos se formarán mediante conjunto de prismas rectangulares. Además, debe permitirse que estos recintos sean bidimensionales e incluso mono dimensionales.
VOLP	Volúmenes prohibidos. Volúmenes definidos como prismas rectangulares en los que no pueden encontrarse las balizas.
PlanoZ, PlanoX PlanoY	En caso de querer optimizar un solo plano dentro del recinto a estudiar, se da la posibilidad de indicar el plano X,Y,Z que pretenda optimizarse dentro del recinto. Esta forma es más intuitiva que definir el recinto como recinto 2D
Restricciones: ub, lb	Restricciones al movimiento de las balizas. A parte de los volúmenes prohibidos en los que no pueden encontrarse las balizas, debemos definir los volúmenes en los que sí pueden ubicarse y el rango de giro posible de cada baliza

```
d=3               #distancia entre puntos del mallado
maxiter=100       #iteraciones de optimizacion
swarmsize=200     #semillas

#REQUISITOS:
#nº minimo de balizas que deben cumplir con sensibilidad en un punto:
n_antenas_sens=3
sensibilidad=2.679319e-09 #Sensibilidad mínima

#nº de pares de balizas cuya diferencia entre modulos debe cumplir con
    los requisitos:
n_antenas_dif=3
diferencia_magnitudes=0.01 #diferencia magitudes de balizas de forma que
     debe cumplirse: B1min>0.01B2max



#_________________ DIFINICION VOLUMEN DE CALCULO: _____________________
#volumenes en los que se calculara el nº de puntos no optimos: a elegir
    segun el problema
VOL=[]            #lista en la que guardamos todos los volumenes
```

```
#VOLUMEN 1:
xmax1, xmin1=2,0  #Puntos maximos y minimos del volumen
ymax1, ymin1=6,0
zmax1, zmin1=6,0
Vol=[xmax1,xmin1,ymax1,ymin1,zmax1,zmin1]
VOL.append(Vol)

#VOLUMEN 2:
xmax2, xmin2=6,0
ymax2, ymin2=8,6
zmax2, zmin2=6,0
Vol=[xmax2,xmin2,ymax2,ymin2,zmax2,zmin2]
"""activar la siguiente linea si tenemos 2 o mas volumenes"""
VOL.append(Vol)

#VOLUMEN 3:
xmax3, xmin3=4,2
ymax3, ymin3=6,4
zmax3, zmin3=6,0
Vol=[xmax3,xmin3,ymax3,ymin3,zmax3,zmin3]
"""activar la siguiente linea si tenemos 3 o mas volumenes"""
#VOL.append(Vol)

#COMPROBAR SI SE INTERSECTAN LOS VOLUMENES DE CALCULO:
Interseccion=InterseccionVolumenes(VOL)
if Interseccion==True:
    print('Warning: Los volúmenes se intersectan')

#PLANOS QUE PODEMOS OPTIMIZAR-->solo podemos elegir 1
PlanoZ=[3]                    #Plano Z en caso de optimizacion 2D
PlanoX=[]                     #Plano X en caso de optimizacion 2D
PlanoY=[]                     #Plano Y en caso de optimizacion 2D
"""Si PlanoZ,X,Y=[] (listas vacías)----->PROBLEMA 3D"""


#_________________ DIFINICION VOLUMEN PROHIBIDO: _____________________
#Volúmenes en los que no se pueden situar las balizas (columnas,etc)
VOLP=[]
#VOLUMEN PROHIBIDO 1:
xmaxp, xminp=1.5,1
ymaxp, yminp=4,2
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
VOLP.append(Volp)

#VOLUMEN PROHIBIDO 2:
xmaxp, xminp=3.5,3
ymaxp, yminp=5,4.5
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
```

```
"""activar la siguiente linea si tenemos 2 o mas volumenes"""
VOLP.append(Volp)

#VOLUMEN PROHIBIDO 3:
xmaxp, xminp=4,2
ymaxp, yminp=7,6.5
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
"""activar la siguiente linea si tenemos 3 o mas volumenes"""
#VOLP.append(Volp)

#_________________ DEFINICION RESTRICCIONES BALIZAS _____________________
#DEFINIR LOWER Y UPPER BOUNDS: define las zonas en las que pueden estar
    las balizas y sus angulos: son las restricciones del problema (
    posiciones y angulos entre los que se pueden mover las balizas)

#[x1min,y1min,z1min] // [x1max,y1max,z1max] BALIZA 1:
A1min, A1max=[0,0,0], [2,6,6]

#[x2min,y2min,z2min] // [x2max,y2max,z2max] BALIZA 2:
A2min, A2max=[0,6,0], [6,8,6]

#[x3min,y3min,z3min] // [x3max,y3max,z3max] BALIZA 3:
A3min, A3max=[2,4,0], [4,6,6]
"""Al observar la solución debe observarse que las balizas están en
    estos recintos"""

#[theta1min, theta2min...]//[theta1max,theta2max...]:
thetamin, thetamax=[-pi,-pi,-pi], [pi,pi,pi]

#[phi1min,phi2min...]//[phi1max,phi2max...]:
phimin, phimax=[-pi/2,-pi/2,-pi/2], [pi/2,pi/2,pi/2]

lb=A1min+A2min+A3min+thetamin+phimin #[x1min,y1min,z1min,x2min,y2min...
    theta1min,theta2min...,phi1min...] VALORES MINIMOS
ub=A1max+A2max+A3max+thetamax+phimax # VALORES MAXIMOS
```

Las variables *VOL* y *VOLP* se formarán como una lista de listas en las que se guardan las dimensiones de todos los volúmenes del rencito a estudiar y los volúmenes prohibidos. Podrán añadirse todos los volúmenes que sean necesarios. Además, se desarrolla la función auxiliar **InterseccionVolumenes** (Código A.5) que permitirá determinar si se produce alguna intersección entre los volúmenes definidos para formar el recinto *VOL*. Si este fallo se produce se enviará un aviso para que se corrija el error, dado que si se produce una intersección entre volúmenes se estarían optimizando los mismos puntos más de una vez.

Por otra parte, las restricciones de los rangos de movimiento de las balizas (posición y ángulos) se definen como listas únicas *lb* y *ub* para poder introducirlas en la función **pso** de forma correcta como se verá más adelante. Además, es importante hacer notar que el rango de movimiento de las balizas es completamente independiente al recinto de optimización *VOL*, dado que se definen con variables distintas.

A continuación se muestran distintos ejemplos de la configurabilidad del problema. Se representan distintos resultados tras realizar la optimización de diversos problemas, variando los recintos a estudiar, los volúmenes prohibidos, los planos a estudiar y las restricciones de posicionamiento de las balizas. La forma de realizar la optimización se mostrará en la siguiente sección. No se especifican las restricciones impuestas para cada figura dado que éstas sólo sirven a nivel ilustrativo para entender las posibles configuraciones que puede adoptar el problema.

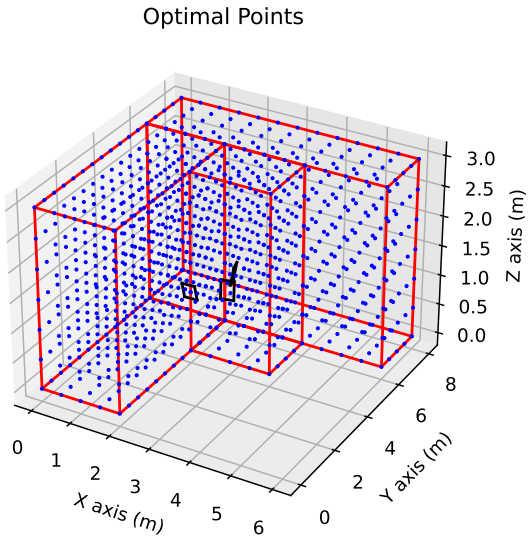


Figura 5.1 Recinto complejo de estudio (rojo).

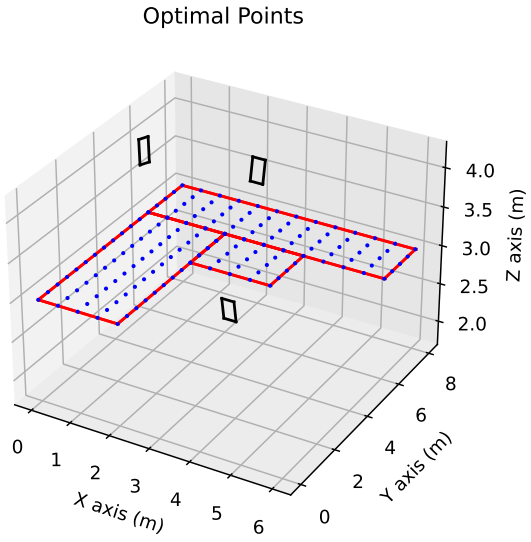


Figura 5.2 Recinto 2D (zmax=zmin para todos los volúmenes).

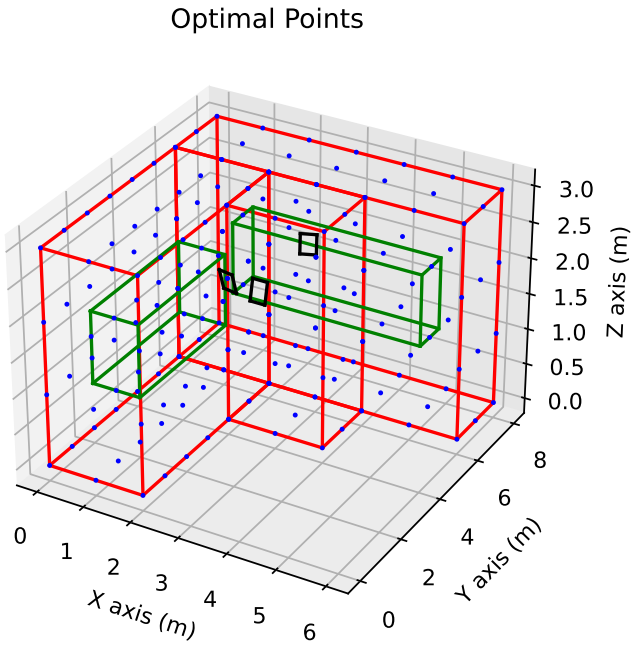


Figura 5.3 Volúmenes prohibidos (verde) y mallado con menor número de puntos.

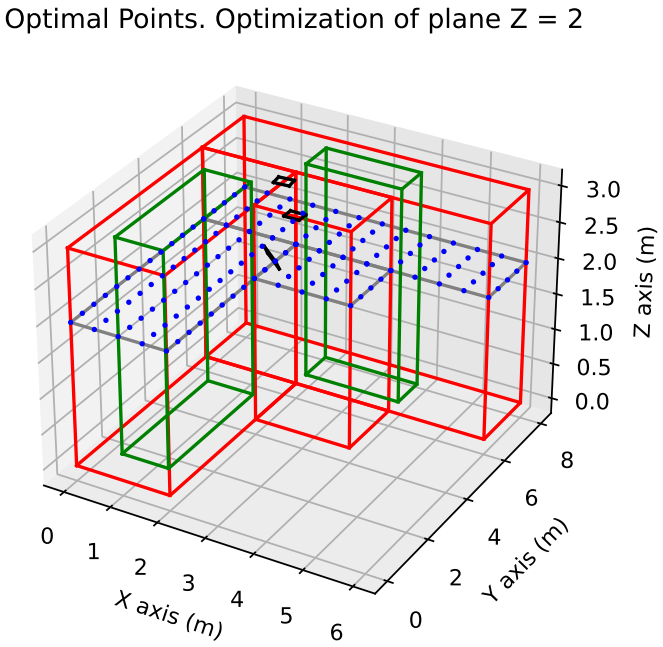


Figura 5.4 Optimización de un plano de un recinto 3D: PlanoZ=[2].

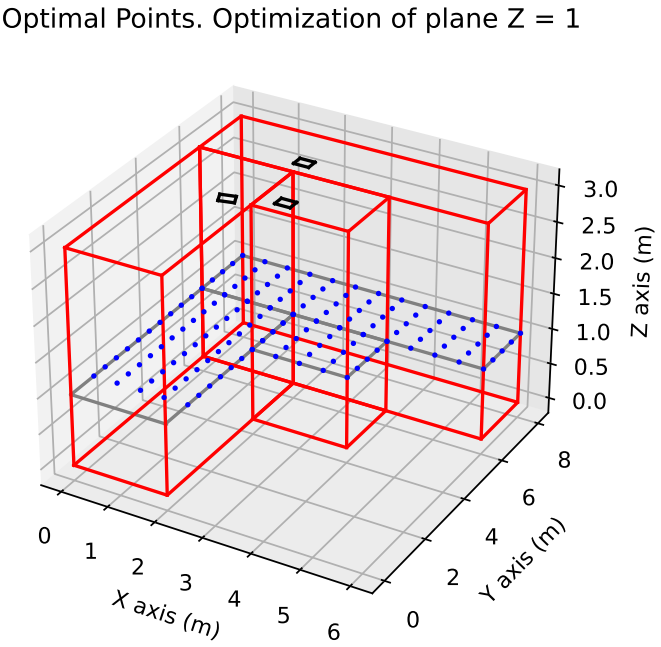


Figura 5.5 Restricción posiciones de balizas: En el techo y en horizontal (zmax=zmin y phimax=phimin=$\pi/2$ para todas las balizas).

5.2 Función de optimización

La función **Optimizacion** que permite realizar la optimización del problema según los datos introducidos en el apartado anterior se muestra en el **Código A.7**. Dentro de esta función, se introducen 4 funciones de coste distintas dependiendo del problema. Estas funciones serán adaptaciones de la función de coste desarrollada en el **Código A.4** previamente explicada en el capítulo 4.

De esta forma, si se tiene un recinto 3D o 2D a optimizar se llamará a la función de coste **Puntos_No_Opt**; mientras que si se especifica un plano a optimizar en concreto dentro de un recinto 3D, llamaremos a las funciones **Puntos_No_Opt2DX**, **Puntos_No_Opt2DY** ó **Puntos_No_Opt2DZ**.

Teniendo dichas funciones de coste, la función **pso** optimizará el número de puntos óptimos en el recinto mediante el algoritmo de optimización por enjambre de partículas. Este algoritmo toma un serie de muestras (distribuciones de balizas) como posible solución del problema y calcula el resultado de la función de coste para dichas muestras. A partir de las soluciones obtenidas, mueve las muestras según su mejor posición local hallada hasta el momento y también según las mejores posiciones globales alcanzadas por el resto de partículas para intentar que la nube de partículas converja hacia la mejor solución. De esta forma, tendremos un proceso iterativo de prueba y búsqueda de nuevas soluciones [8]. En nuestro caso concreto, la función **pso** se configura para minimizar el número de puntos no óptimos en el recinto. Finalmente, los argumentos de salida serán las listas $Pos\_antena = [x1,y1,z1,x2,y2,z2,...]$, $theta = [\theta1,\theta2,\theta3,...]$ y $phi = [\phi1,\phi2,\phi3...]$ solución del problema

El **Código 5.2** muestra el extracto perteneciente al **Código A.7** encargado de realizar la llamada correcta de **pso** dependiendo de la configuración del problema. Como puede observarse, la función

necesitará como parámetros de entrada la función de coste a la que debe recurrir, las restricciones de movimiento de las balizas (*lb*,*ub*), los argumentos de entrada para las funciones de coste (recinto a estudiar, volúmenes prohibidos, distancia entre puntos del mallado...) y los parámetros de optimización *maxiter* y *swarmsize* que indican el número máximo de iteraciones y las semillas al realizar la optimización.

Código 5.2 Llamada a la función **pso.**

```
#########################################################################
"""ALGORITMO GENETICO // NO EDITAR"""
#########################################################################
#Minimiza el Nº de puntos no optimos

n_bounds=len(ub)
for n in range(n_bounds):
    if lb[n]==ub[n]:
        ub[n]=lb[n]+0.001
    #si los limites inferiores y superiores son iguales, debemos añ
        adir un pequeño incremento para el correcto funcionamiento de
         la funcion pso

#En caso de realizar optimizacion 3D:
if PlanoZ==[] and PlanoX==[] and PlanoY==[]:
    args=(N,VOL,VOLP,d)
    SOLopt, fopt = pso(Puntos_No_Opt, lb, ub, args=args, maxiter=
        maxiter,swarmsize=swarmsize)
    #SOLopt=SOLUCION OPTIMA ENCONTRADA POR EL ALGORITMO

#En caso de que solo queramos optimizar un solo plano Z
if PlanoZ!=[]:
    args=(N,VOL,VOLP,d,PlanoZ)
    SOLopt, fopt = pso(Puntos_No_Opt2DZ, lb, ub, args=args, maxiter=
        maxiter,swarmsize=swarmsize)

#En caso de que solo queramos optimizar un solo plano X
if PlanoX!=[]:
    args=(N,VOL,VOLP,d,PlanoX)
    SOLopt, fopt = pso(Puntos_No_Opt2DX, lb, ub, args=args, maxiter=
        maxiter,swarmsize=swarmsize)

#En caso de que solo queramos optimizar un solo plano Y
if PlanoY!=[]:
    args=(N,VOL,VOLP,d,PlanoY)
    SOLopt, fopt = pso(Puntos_No_Opt2DY, lb, ub, args=args, maxiter=
        maxiter,swarmsize=swarmsize)

#Obtenemos los valores de posiciones y angulos optimos obtenidos:
Pos_antena=SOLopt[:(N*3)] #posicion optima de las balizas [x1,y1,z1,
    x2,y2,z2...]
theta=SOLopt[(N*3):(N*3 + N)] #angulos theta optimos de las balizas
```

```
    phi=SOLopt[(N*3 + N):]

    #TIEMPO DE EJECUCION
    fin=time.time()
    print('Tiempo de Optimización:', fin-inicio)

    ########################################################################
    #Finalmente devolemos la solución obtenida
    return(Pos_antena,theta,phi)
```

5.3 Funciones de coste adaptadas

Con respecto las funciones de coste que pueden encontrarse en el **Código A.7**, se han adaptado para que éstas reciban un sólo parámetro (s) que contenga las posiciones de las balizas y sus ángulos; es decir: s = [Pos_antena,$theta$,phi]. Siendo *Pos_antena, theta, phi* listas en las que se guardan las posiciones y ángulos de todas las balizas. Este parámetro será el parámetro que la función **pso** podrá variar atendiendo a las restricciones *lb* y *ub*. Es decir, irá probando distintas distribuciones de las balizas para buscar la mejor opción dentro del rango permitido por *lb* y *ub*.

Por otra parte las funciones recibirán el parámetro *args* en el que se han guardado todas las restricciones del problema como pueden ser el recinto a estudiar, los volúmenes prohibidos o la distancia entre puntos del mallado. Finalmente las funciones devolverán el número de puntos no óptimos según la distribución de balizas recibida.

Dentro del funcionamiento de estas funciones, cabe destacar que en primer lugar se comprueba si alguna de las balizas se encuentra en uno de los volúmenes prohibidos (*VOLP*) en los que no podemos situarlas. En caso de encontrarse en una zona prohibida, se le asigna directamente a esta distribución de las balizas un número de puntos no óptimos muy alto (1e20) y no realizamos el cálculo del número de puntos no óptimos. De esta forma, la función **pso** no cogerá dicha distribución de las balizas. Si las balizas no se encuentran en una posición prohibida se continúa realizando el cálculo del número de puntos no óptimos.

Por otro lado, si se tienen recintos complejos formados por más de un prisma rectangular, simplemente se calculará el número de puntos no óptimos en cada prisma, siendo el total la suma de todos ellos. Además, si se tiene un recinto complejo formado por varios prismas y se indica que quiere realizarse una optimización de un plano del recinto, la función sólo tendrá en cuenta aquellos prismas que contienen al plano en cuestión. Es decir, si por ejemplo tenemos un volumen V1 con [$zmax$ = 3, $zmin$ = 0] y un volumen V2 con [$zmax$ = 8, $zmin$ = 0] y se indica que se quiere optimizar el plano Z = 6, sólo se tendrá en cuenta el volumen V2, dado que V1 no contiene al plano Z = 6. Un ejemplo de ello se muestra en la Fig.5.6.

5.4 Tiempos de ejecución

A la vista de los numerosos parámetros que rigen el problema, puede concluirse que la optimización no será un proceso sencillo. En primer lugar las variables sujetas a variaciones (s) para mejorar la optimización son numerosas y aumentan fuertemente conforme el número de balizas aumenta.

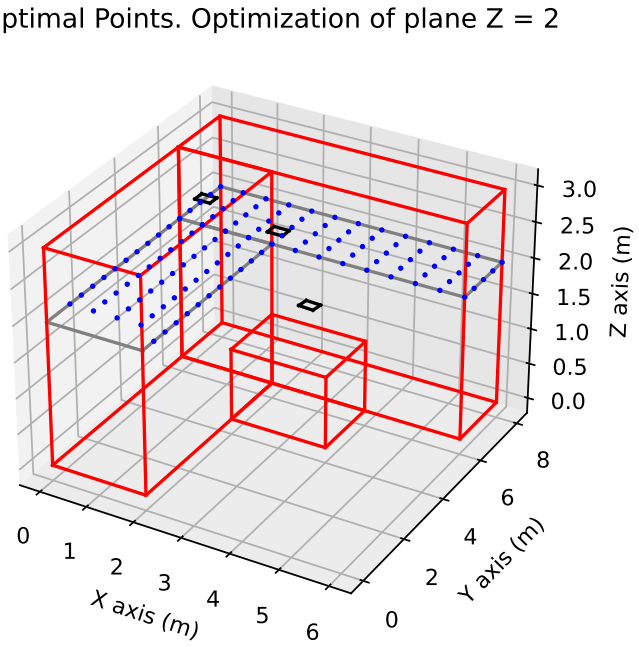


Figura 5.6 Estudio de un plano que no contiene a uno de los prismas.

En segundo lugar, el carácter de las funciones de coste usadas para la resolución del problema es no lineal debido a las funciones trigonométricas que rigen el movimiento angular de las balizas. Por último, la resolución del problema estará fuertemente ligada al mallado del recinto. Mallados muy densos provocarán grandes tiempos de ejecución.

Es por ello que se realizan distintas pruebas de optimización con distinto número de puntos de mallado y se comprueba que los tiempos de ejecución sean aceptables. También se variará el número de iteraciones y el número de semillas usadas en la optimización. Las pruebas se realizarán para un recinto cúbico y 3 balizas que podrán moverse por todo el recinto excepto un cubo interior (Fig.5.7). Los parámetros que determinarán un punto óptimo han sido configurados como: *n_antenas_sens* = 3, *sensibilidad* = $2.679319 \cdot 10^{-9}$, *n_antenas_dif* = 3 y *diferencia_magnitudes* = 0.01[1]. Los resultados se muestran en la Tabla 5.2.

Tabla 5.2 Tiempos de ejecución y resultados tras optimización.

Puntos (mallado)	iteraciones	swarmsize	Tiempo de ejecución (s)	% Puntos óptimos
27	50	100	13.45	99.2262
	100	100	35.3618	97.4961
	100	200	44.8440	99.0441
343	50	100	157.0478	98.9076
	100	100	191.5022	99.8634
	100	200	670.5596	99.8179
729	50	100	200.5807	99.9089
	100	100	547.1022	99.7724
	100	200	858.0807	99.4993

[1] Mismos valores que en el capítulo 4, sección 4.2.

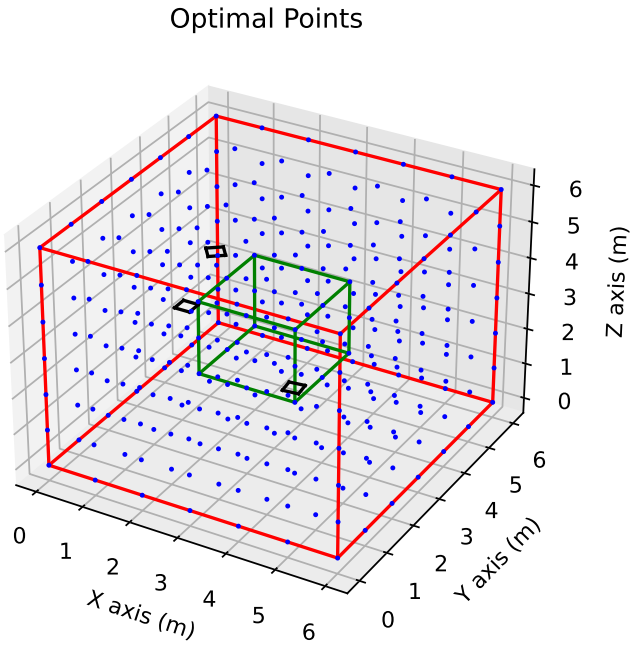


Figura 5.7 Recinto estudiado en prueba de tiempos de ejecución.

Es interesante observar que la diferencia de resultados entre la optimización con 27 puntos de mallado, 50 iteraciones y 100 semillas es ínfima respecto la optimización con 729 puntos, 100 iteraciones y 200 semillas. Sin embargo, la diferencia de tiempos sí que es muy significativa. Esto hace pensar que una buena estrategia de trabajo será realizar una primera optimización menos precisa (menos puntos de mallado, menos iteraciones y menos semillas); y en caso de no obtener resultados satisfactorios aumentar estos parámetros.

6 Cálculo y representación de la solución

Las figuras mostradas en el capítulo anterior son el resultado gráfico de los puntos óptimos y posiciones óptimas de las balizas tras optimización de distintas pruebas que se han realizado. En esta sección pasamos a explicar la función que realiza dicha representación junto con otras dos funciones que permiten representar los puntos críticos de máximos umbrales magnéticos y representar la distribución de campo magnético que se produce en distintos planos.

6.1 Representación de Puntos Óptimos

El proceso de representación de puntos óptimos es simple. Siempre que se realiza el algoritmo de búsqueda de puntos óptimos se genera un mallado $[X,Y,Z]$ en el que se generan todos los puntos del recinto a estudiar. De esta forma, bastará con realizar este algoritmo y si, resulta que el punto no es óptimo, guardaremos el valor Z del punto $[i,j,k]$ como NaN. Es decir, la representación consistirá en realizar la función de coste adaptada para representar el resultado obtenido.

Una vez realizado esto, haciendo uso del paquete *matplotlib.pyplot* [10] y representando los puntos de la malla $[X,Y,Z]$, los puntos que contengan $Z[i,j,k] = np.nan$ no serán representados. De esta manera sólo se representan los puntos óptimos. Este proceso se muestra en el **Código 6.1**; que es un extracto de la representación de puntos óptimos para la función de coste del **Código A.4**; que calculaba y representaba los puntos óptimos en un volumen simple con forma de prisma rectangular.

Para la representación de las balizas se toman los vértices en ejes cuerpo, se rotan en ejes globales a través de la matriz de rotación y se transportan al punto del espacio adecuado según la posición de la baliza correspondiente. Por otra parte, la representación del volumen consiste en la representación de un prisma rectangular según los vértices del volumen[1]. Esto mismo también se muestra en el **Código 6.1**.

Código 6.1 Representación de la solución.

```
########################################################################
"""REPRESENTACION PUNTOS OPTIMOS EN EL VOLUMEN ESTUDIADO"""
########################################################################
figura = plt.figure()
```

[1] Código basado en la representación de cubos 3D con librería matplotlib presentada en [11]

```
grafica = figura.add_subplot(111,projection = '3d')
grafica.scatter3D(X,Y,Z, c='blue',marker='o',s=2)
    #Representamos los puntos óptimos

#REPRESENTACION BALIZAS: calculamos los 4 vértices de cada baliza y
    representamos las líneas que unen a dichos vértices
k=0
for n in range(N):
    x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
        #posicion de la baliza 'n'
    Rot=Rotaciones[0:3,k:k+3]
        #matriz de rotación para la baliza 'n'

    #Vértices de las balizas en ejes cuerpo
    p1, p2=[2*semilado[n],0,2*semilado[n]], [-2*semilado[n],0,2*semilado
        [n]]
    p3, p4=[-2*semilado[n],0,-2*semilado[n]],[2*semilado[n],0,-2*
        semilado[n]]
    #agrandamos el tamaño de la baliza para mejor visualizacion
    #vértices en ejes globales:
    p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
    p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

    P=np.concatenate((p1,p2,p3,p4,p1))
    for r in [0,3,6,9]:
    #Líneas que unen vértices:
        X=np.linspace(x+P[r],x+P[r+3],10)
        Y=np.linspace(y+P[r+1],y+P[r+4],10)
        Z=np.linspace(z+P[r+2],z+P[r+5],10)
        grafica.plot(X,Y,Z,color='black')
    k+=3

#representamos el volumen en donde estamos calculando los puntos óptimos
xx = [xmin, xmin, xmax, xmax, xmin]
yy = [ymin, ymax, ymax, ymin, ymin]
color='red'
kwargs = {'alpha': 1, 'color': color}
grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **kwargs)
grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **kwargs)
grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **kwargs)
grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **kwargs)

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

La función **Solucion** mostrada en el **Código A.8** extiende esta idea sea cual sea el recinto estudiado, dando la posibilidad de estar formado por varios volúmenes distintos o de estudiar un solo plano dentro del recinto (en definitiva mismas posibilidades de configuración que para la optimización). Los argumentos de entrada son los mostrados en la Tabla 6.1.

Tabla 6.1 Parámetros función de representación de puntos óptimos **Solucion.**

	Parámetros balizas
semilado	Lista: Longitud del semilado de las balizas (m)
N_espiras	Lista: Número de espiras de las balizas
I	Lista: Intensidades que recorren las balizas (A)
C	Factor de calibración de las balizas
F	Lista: Factor de conversión F(f) según frecuencia de balizas
N	Número de balizas de las que se dispone
Pos_antena	Posiciones [$x1$,$y1$,$z1$,$x2$,$y2$,$z2$...] de las balizas
theta	Ángulos [$\theta 1$,$\theta 2$,...] de las balizas
phi	Ángulos [$\phi 1$,$\phi 2$,...] de las balizas

	Parámetros cálculo de puntos óptimos y representación
ds	Distancia entre puntos al realizar el mallado
n_antenas_sens	Número de antenas cuyo campo debe superar la sensibilidad del sensor para que un punto sea óptimo
sensibilidad	Sensibilidad del sensor (T)
n_antenas_dif	Número de pares de antenas cuya diferencia entre modulos debe ser inferior a la máxima permitida
diferencia_magnitudes	Máxima diferencia de magnitud entre dos antenas permitida
VOL	Recinto que se ha querido optimizar. Puede estar formado por varios volúmenes.
VOLP	Volúmenes prohibidos
PlanoZ, PlanoX PlanoY	Plano que se ha querido optimizar (en caso de haber querido optimizar un plano de un recinto 3D)
Tant	Parámetro para aumentar o disminuir el tamaño de las antenas en la representación

En primer lugar se diferencia dependiendo de si el recinto es 3D o 2D, o se pide optimizar un plano concreto de un recinto 3D (al igual que se hacía al optimizar). A continuación se obtienen los puntos óptimos para todos los volúmenes o áreas y se representan junto con las balizas colocadas en sus respectivas posiciones, los volúmenes que forman el recinto a estudiar, el área a estudiar (en caso de pedirse) y los volúmenes prohibidos.

Por tanto, introduciendo los datos de las posiciones de balizas y sus ángulos obtenidos tras la optimización; el recinto estudiado y los volúmenes prohibidos, se representará la solución obtenida. Aún así, esta función no tendrá como único uso introducir la solución obtenida tras optimización. Dado que **Solucion** es en definitiva una función de coste que permite obtener la representación de los puntos óptimos y ésta es externa a la función **Optimizacion**; pueden introducirse los datos de la forma que más convenga, posibilitando poder comparar distintas distribuciones y configuraciones del problema que el usuario pueda requerir estudiar de forma directa sin pasar por la optimización.

Además, esta función no solo representa la solución; sino que proporciona como argumentos de salida el número de puntos estudiados, el número de puntos óptimos y el porcentaje de puntos óptimos. De esta forma, podemos realizar una optimización con un mallado con un menor número de puntos para reducir tiempos de optimización (la función **Optimizacion** recibe el parámetro d para realizar el mallado), y luego aumentar el número de puntos del mallado al usar la función **Solucion** (la función **Solucion** recibe el parámetro ds para realizar el mallado) de forma que obtenemos resultados más precisos. Es decir, podemos realizar la optimización con $d = 1$ y luego estudiar y representar la solución con $dc = 0.1$ (mayor densidad de puntos), de forma que obtenemos un resultado más preciso de la solución obtenida (o de la distribución que se quiera estudiar).

6.2 Representación de Puntos Críticos

Como se comentó previamente, uno de los requisitos impuestos al software es ser capaz de representar los puntos en los que se producen valores de campo magnético demasiado altos. Esto se debe a que a la hora de disponer en un recinto una serie de balizas emisoras de campos magnéticos hay que tener en cuenta que, si ese recinto es accesible a las personas, éstas no pueden estar sometidas a campos electromagnéticos que sobrepasen cierto umbral. Este umbral depende de la frecuencia de trabajo y viene determinado por normativa específica para público en general [4] y también para ámbito laboral [5]. Un ejemplo de los valores de referencia en el ámbito público se muestra en la Tabla 6.2.

Tabla 6.2 Niveles de referencia para campos eléctricos, magnéticos y electromagnéticos [4].

Frecuencia	Intensidad de campo E (V/m)	Intensidad de campo H (A/m)	Campo B (μT)
0-1 Hz	-	$3.2{\cdot}10^4$	$4{\cdot}10^4$
1-8 Hz	10,000	$3.2{\cdot}10^4$ / f^2	$4{\cdot}10^4$ / f^2
8-25 Hz	10,000	4,000 / f	5,000 / f
0.025-0.8 kHz	250 / f	4 / f	5 / f
...	...	...	...

Por esta razón, una funcionalidad interesante del software de optimización debe ser proporcionar de manera automática un mapa de puntos donde ese umbral se supere, pudiendo definir el umbral necesario según las circunstancias de trabajo. La idea es asegurar que el campo magnético esté por debajo del umbral de seguridad en todas las regiones donde puedan circular o permanecer personas.

La función **Criticos** es la encargada de representar los puntos del espacio que superan los límites de campo magnético admisibles para el ser humano. Esta función se muestra en el **Código A.9**.

Su funcionamiento es muy similar a la representación de puntos óptimos. Se hará uso de un algoritmo que permitirá establecer si un punto del espacio supera o no supera los umbrales de campo magnético. En caso de no superarlos, el valor $Z[i,j,k]$ del mallado se establecerá como NaN y no se representará. De esta forma, los puntos sólo se representarán en caso de incumplir los límites. La parte del **Código A.9** que se encarga de establecer si un punto es crítico se muestra en el **Código 6.2**. En él, se suman los módulos de los campos producidos por las distintas balizas y se compara con el parámetro *Exposicion*, que será el valor máximo al que un ser humano puede estar expuesto.

Código 6.2 Representación de puntos críticos.

```
        ################################################################
        """ALGORITMO BUSQUEDA DE PUNTOS CRITICOS"""
        ################################################################
        for i in range(nx):
            for j in range(ny):
                for k in range(nz):
                    MODB=0 #Suma del módulo del campo de todas las
                        balizas

                    l=0
                    for n in range(N):
                        Pos_sensor=[x[i],y[j],z[k]]
```

```
                    Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                        Pos_antena[l+2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado[n
                        ], I[n], N_espiras[n], Rotaciones[0:3,l:l+3])
                    MODB+=modB
                    l+=3
                #Si el campo es menor al campo máximo admisible, no
                    representamos:
                if MODB<Exposicion:
                    Z[j,i,k]=np.nan

    #Representamos puntos CRITICOS del volumen 'v':
    grafica2.scatter3D(X,Y,Z, c='orange',marker='o',s=2)
```

Se ha comprobado que los puntos donde no se cumple con los umbrales de campo magnético se producen en zonas cercanas al centro de las balizas, como era de esperar. De esta manera, en vez de estudiar todos los puntos del recinto que se haya estudiado (sería una pérdida de tiempo dado que la mayoría serán válidos), se estudiarán recintos pequeños alrededor de las distintas balizas. En concreto se estudiarán recintos cúbicos centrados en la posición de cada baliza y longitud de lado *dc*; siendo la distancia entre puntos del mallado en estos recintos también configurable (*ds_exp*). Si se observa que el valor *dc* es insuficiente para mostrar todos los puntos críticos, bastará con aumentar su valor.

Por último, como viene siendo costumbre habrá que introducir los parámetros de las balizas, el recinto estudiado, etc...Un ejemplo de la representación obtenida con $dc = 1$ y $ds\_exp = 0.1$ se muestra en la Fig.6.1.

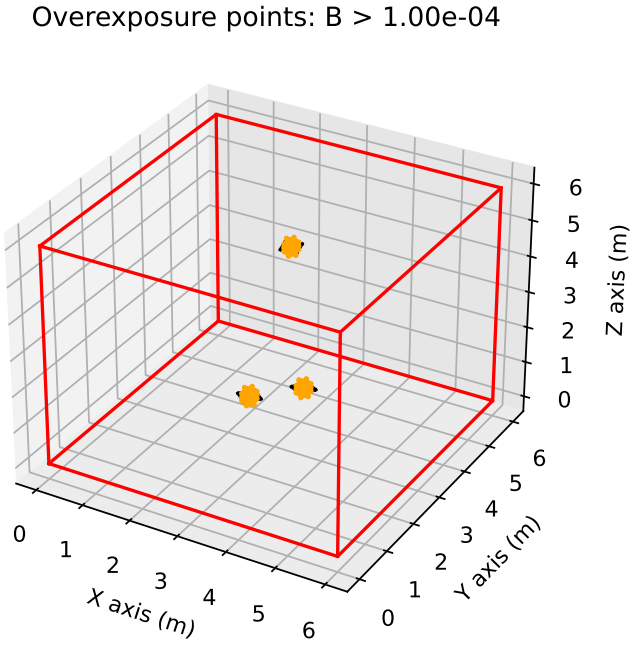


Figura 6.1 Puntos críticos: umbrales de campo magnético.

6.3 Representación de Campo Magnético

Finalmente, como última opción de representación que permita estudiar el campo generado por cierta distribución de balizas en un recinto dado, se desarrolla la función **RepresentacionPlano** presentada en el **Código A.10**. Esta función proporcionará la distribución de campo magnético (en módulo) en distintos planos del recinto estudiado. Esta es una opción muy interesante, dado que permitirá observar el mapeo resultante para realizar posicionamiento por huellas o permitirá observar las zonas en las que el campo será más pequeño (peor cobertura).

Como parámetros de entrada se dará la opción de introducir los planos X, Y o Z que se deseen estudiar (mediante las variables *RepresentacionZ*, *RepresentacionX*, *RepresentacionY* en el Código A.6); el volumen (*VOL*) a estudiar, las posiciones de las balizas junto con sus parámetros característicos y como última opción, la variable *dB*, que permitirá representar el campo en dBnT ($20log_{10}(\frac{B}{10^{-9}})$) si su valor es *True*, o en escala logarítmica si su valor es *False*.

El algoritmo para el cálculo del campo magnético y su representación se muestra en el **Código 6.3** (caso de un plano Z). En él, se crea una rejilla 2D que formará los puntos en los que estudiar los campos. Se calculará el campo magnético de cada baliza en forma vectorial $[Bx,By,Bz]$ y se sumarán las contribuciones de cada baliza. Por último se calcula el módulo resultante. Dado que se producen diferencias demasiado altas entre campos (incluso estando en escala logarítmica) debidas a la singularidad de la función **Dipolo** en el centro de las balizas, se establece un máximo de campo de 10^{-5} (T) y un mínimo igual a la sensibilidad del sensor. Puntos en los que se produzcan campos mayores o menores a estos valores, se igualarán al máximo o mínimo permitido. La representación del campo consistirá finalmente en la representación de puntos mediante mapa de color basado en [12] y [13].

Código 6.3 Cálculo y representación de secciones de campo magnético.

```
    #Si nos piden un plano Z=cte:
    if RepresentacionZ!=[]:
        ###################################################################
        """DEFINICION AREA DE CALCULO"""
        ###################################################################
        #Creacion rejilla con puntos en los que evaluar el campo en ese
            volumen :
        nx, ny = int(((xmax-xmin)/((xmax-xmin)/300))+1), int(((ymax-ymin)
            /((xmax-xmin)/300))+1)
        #300 puntos en cada eje: (más puntos en la representación para
            mejor visualizacion)

        x = np.linspace(xmin, xmax, num=nx)
        y = np.linspace(ymin, ymax, num=ny)
        Xm,Ym=np.meshgrid(x,y)

        ###################################################################
        """ALGORITMO CALCULO CAMPO MAGNETICO"""
        ###################################################################
        Bx,By,Bz=np.zeros((ny,nx)),np.zeros((ny,nx)), np.zeros((ny,nx))

        for i in range(nx):
```

```
        for j in range(ny):

            l=0
            for n in range(N):
                Pos_sensor=[x[i],y[j],RepresentacionZ[0]]
                Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                    Pos_antena[l+2]]

                [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I
                    , N_espiras, Rotaciones[0:3,l:l+3])

                Bx[j,i]+=f[n]*B[0,0]
                By[j,i]+=f[n]*B[0,1]
                Bz[j,i]+=f[n]*B[0,2]

                l+=3

r=np.add(np.add(Bx**2,By**2),Bz**2)
MODB=np.sqrt(r) #Modulo del campo magnetico en cada punto de la
    malla

#evitar indeterminacion de 'Dipolo' cuando Pos_sensor==
    Pos_antena_n:
for i in range(nx):
    for j in range(ny):
        if MODB[j,i]>1e-5:
            MODB[j,i]=1e-5

        #Si el campo es menor que la sensibilidad no lo
            representamos:
        if MODB[j,i]<sensibilidad:
            MODB[j,i]=sensibilidad

#################################################################
"""Representacion"""
#################################################################
fig, ax=plt.subplots(1)
cm=plt.cm.get_cmap('YlOrRd')

if dB==True:  #Si se pide representar en dB:
    MODBdB=20*np.log10(MODB/1e-9)
    pc=ax.scatter(Xm,Ym,c=MODBdB,norm=colors.Normalize(vmin=
        MODBdB.min(), vmax=MODBdB.max()), cmap=cm)
    plt.title('Campo magnético en Plano Z = %i , (dBnT)' %
        RepresentacionZ[0])
else:
    pc=ax.scatter(Xm,Ym,c=MODB,norm=colors.LogNorm(vmin=MODB.min
        (), vmax=MODB.max()),cmap=cm)
    plt.title('Campo magnético en Plano Z = %i , (T)' %
        RepresentacionZ[0])
```

```
        fig.colorbar(pc,ax=ax)
        plt.show()
```

Como ejemplo, la Fig.6.3 muestra el campo resultante en el plano Z = 4m que se obtiene para la distribución de balizas y el recinto mostrados en la Fig.6.2. Los volúmenes que contienen a dicho plano se representan en línea continua, mientras que los volúmenes que no contienen a dicho plano se represetan en línea discontinua.

Puede observarse que los mayores campos se producen en zonas cercanas a las balizas y, además, cuanto más cerca se encuentre una baliza del plano, mayor será el campo producido por ésta en el plano estudiado. Para demostrar este efecto se muestra la posición en Z de las distintas balizas en la Tabla 6.3. La baliza 2, más cercana a Z = 4m, produce mayor campo que las otras dos balizas en este plano.

Tabla 6.3 Posición en Z de las distintas balizas.

Baliza	1	2	3
Posición en Z	2.757	3.663	1.857

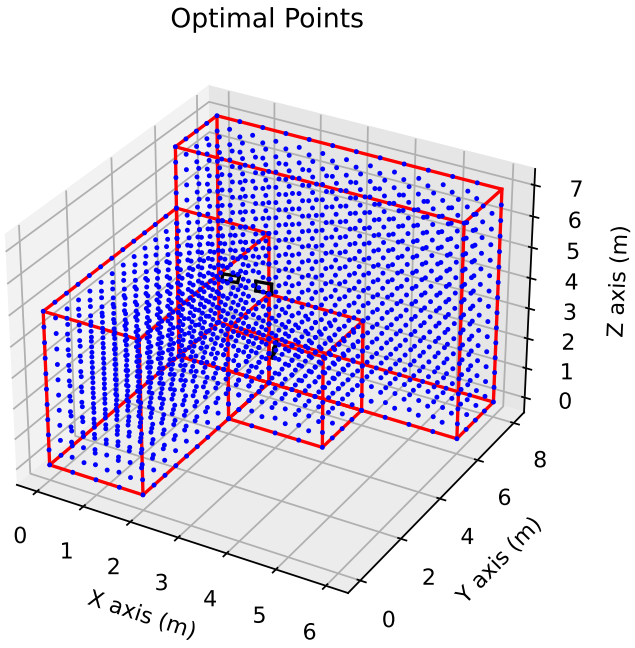


Figura 6.2 Recinto y balizas prueba de representación de planos.

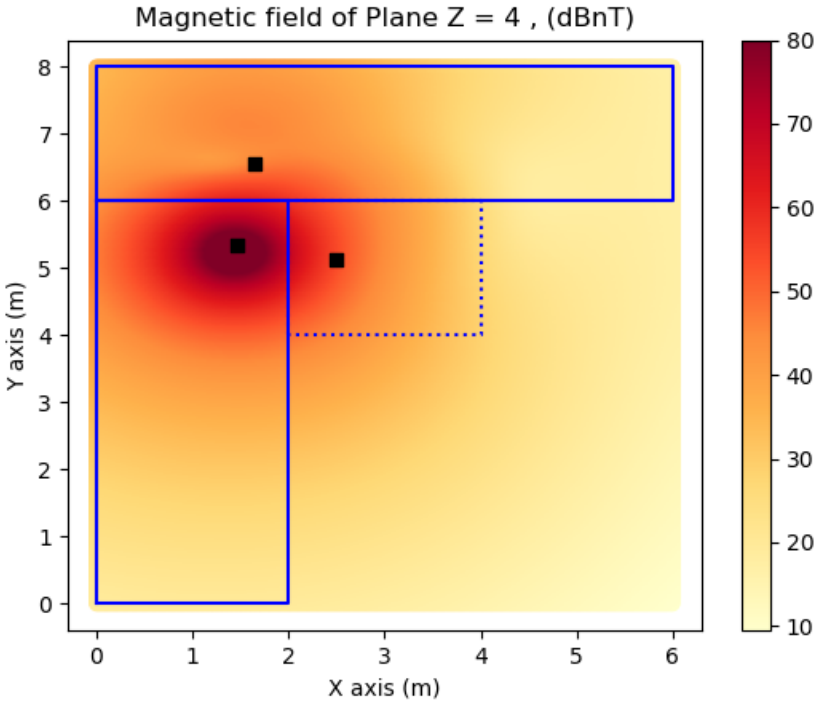


Figura 6.3 Resultado campo magnético producido en plano Z.

7 Interfaz para usuarios

Por último se ha desarrollado una interfaz gráfica que permitirá una configuración más sencilla de los distintos parámetros del problema por parte del usuario. Su programación se muestra en el **Código A.11**. La Tabla 7.1 muestra un resumen de las funciones desarrolladas en capítulos anteriores a las que la interfaz recurrirá a la hora de realizar los cálculos.

Tabla 7.1 Funciones y subfunciones para resolución del problema a través de la interfaz.

Funciones	**Subfunciones**	
Optimizacion (pso)	Búsqueda de posiciones óptimas de balizas	
	Puntos_No_opt	función de coste recinto 3D, 2D o 1D
	Puntos_No_Opt2DX	función de coste en plano X de un recinto 3D
	Puntos_No_Opt2DY	función de coste en plano Y de un recinto 3D
	Puntos_No_Opt2DZ	función de coste en plano Z de un recinto 3D
Solucion	Función de cálculo y representación de puntos óptimos	
Criticos	Función representación de puntos críticos	
RepresentacionPlano	Representación campo magnético en planos X,Y,Z	

Como aspecto a destacar, cabe indicar que se ha hecho uso de la librería **tkinter** [14]. Esta herramienta permitirá crear el conjunto de *frames* y *widgets* que formarán la aplicación. Además, pueden definirse distintas funciones que controlarán el comportamiento de la navegación a través de las diferentes pestañas. A continuación, se comentan las distintas pestañas que han sido desarrolladas y su modo de funcionamiento.

7.1 Definición del recinto a estudiar

En la Fig.7.1 se muestra la primera pestaña de la interfaz. En ella se definirá el recinto en el que querrán realizarse los cálculos. Podrán añadirse todos los volúmenes que sean necesarios o eliminarlos haciendo uso de los botones (+) o (-). Estos botones son una llamada a las funciones *addVopt* y *delVopt*, que añaden o eliminan filas para definir nuevos volúmenes. Además, no podrá avanzarse en caso de dejar uno de los espacios vacíos, saltando un aviso en dicho caso. También se añade el botón *Update* que realiza la llamada a la función *drawVopt* encargada de realizar la representación gráfica del recinto, y que nos permitirá asegurarnos de que las coordenadas introducidas dan lugar al recinto deseado.

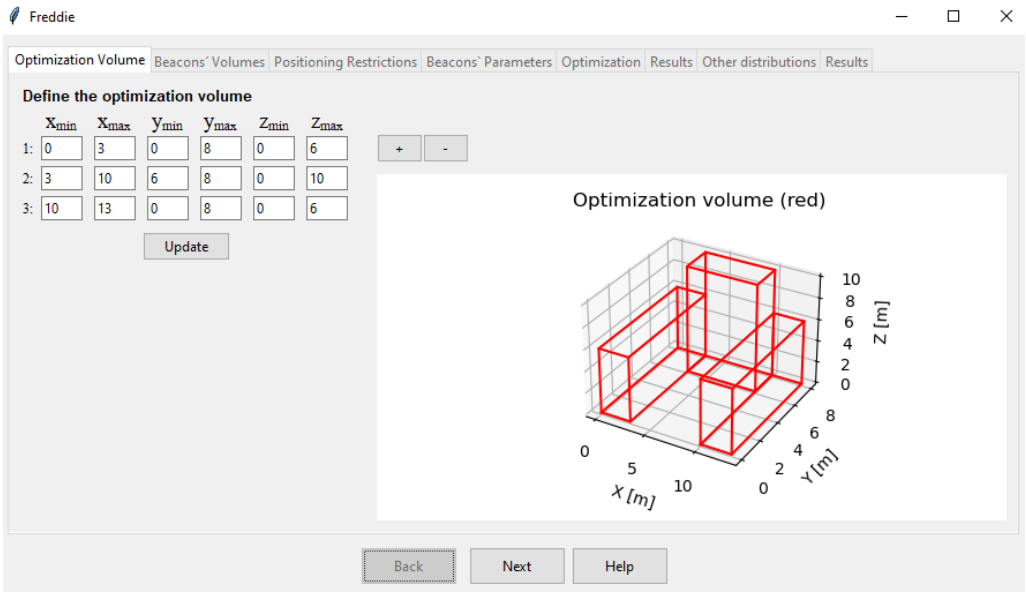


Figura 7.1 Pestaña 1: creación de recinto a estudiar.

Por último encontramos el botón *Next*. Éste nos permitirá navegar por las diferentes pestañas de configuración, haciendo uso de la función *nextFcn*. En caso de producirse intersección entre alguno de los volúmenes definidos, también se producirá un aviso, aunque se dejará continuar si el usuario lo desea. La función *nextFcn* es un punto clave en el funcionamiento de la interfaz. Dependiendo de la pestaña en la que nos encontremos, será la encargada de guardar los valores de las entradas que el usuario ha configurado, configurar el formato de nuevas pestañas, etc. En concreto, al pulsar *Next* en esta pestaña, se guardarán los valores de los volúmenes que forman el recinto en la variable *self.VOL*

7.2 Definición de los recintos de movimiento de las balizas

La Fig.7.2 muestra la pestaña en la que se definirán los volúmenes por los que se podrán mover cada una de las balizas a la hora de realizar la optimización. Estos volúmenes se muestran en azul. El número de balizas a utilizar será por tanto igual al número de volúmenes que hemos definido.

El funcionamiento de esta pestaña es muy similar a la anterior, se hace uso de las funciones *addVant*, *addSameVant* y *delVant* para añadir o eliminar balizas. La función *addSameVant* (botón +same volume) añade balizas cuyo volumen de movimiento es igual al último volumen añadido; facilitando la definición de balizas si todas pueden moverse por el mismo volumen. Además se define la función *drawVant* para representar los volúmenes que se han definido, en la que también se muestra el recinto en el que optimizar el campo (rojo) para tener una imagen más completa del problema.

Como en el caso anterior, no se dejará avanzar en caso de dejar una entrada vacía. Sin embargo, sí podrán producirse intersecciones dado que puede querer definirse un mismo volumen de movimiento para varias o todas las balizas. Al pulsar *Next* guardaremos los valores en las variables *self.lb* y *self.ub*; que como vimos en anteriores capítulos, serán las restricciones al movimiento en la función **pso**.

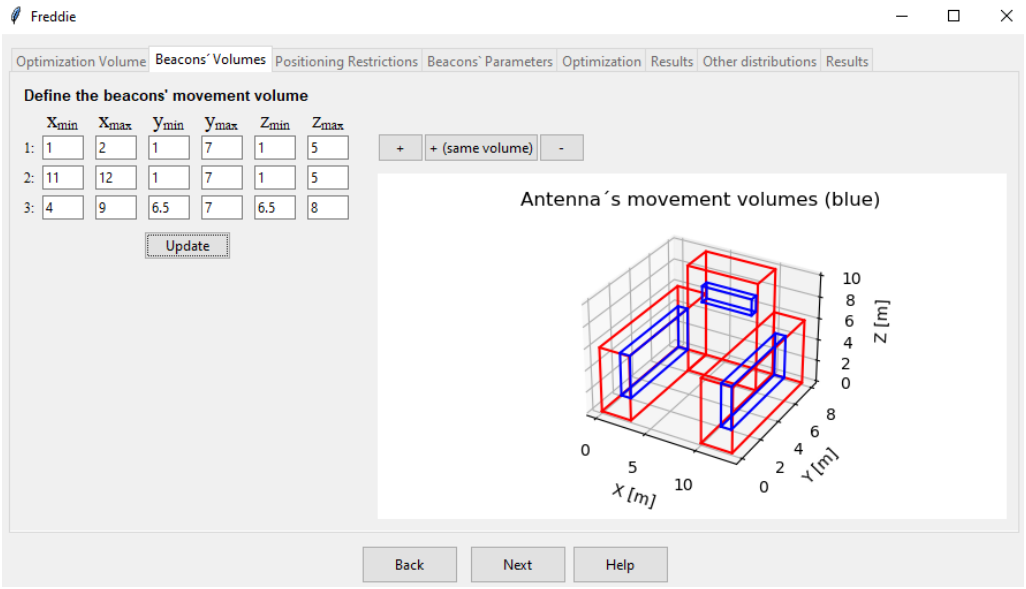


Figura 7.2 Pestaña 2: creación de recintos de movimiento de antenas.

Por último se añade el botón *Help* (aparecerá en todas las pestañas de ahora en adelante), que mostrará información de utilidad con explicaciones de cada pestaña.

7.3 Definición de restricciones al movimiento de las balizas

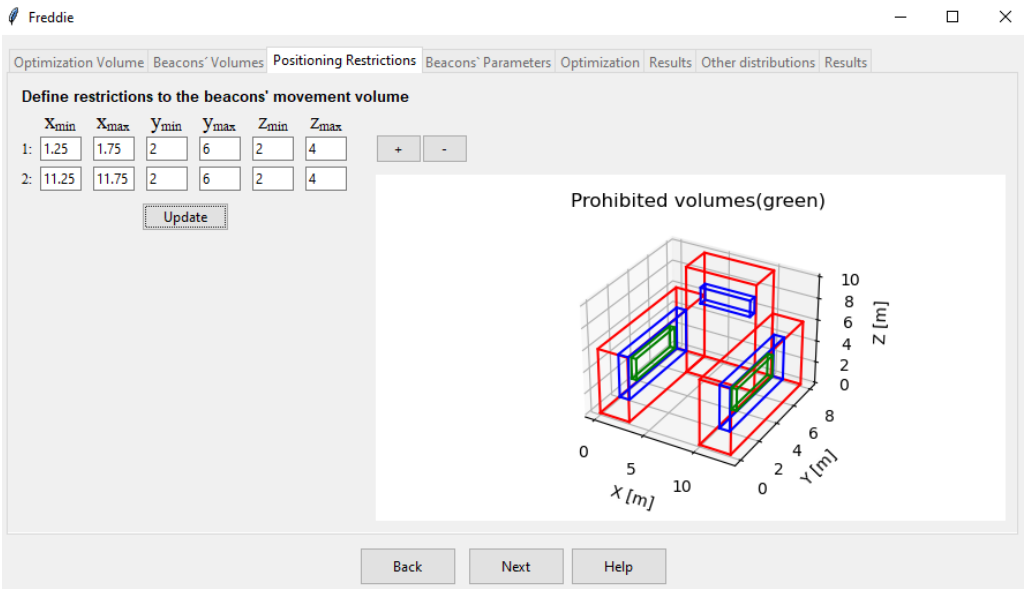


Figura 7.3 Pestaña 3: creación de recintos prohibidos (verde) para posicionamiento de balizas.

A continuación se definen los volúmenes en los que no podrá ubicarse ninguna baliza. Puede ser interesante definir este tipo de restricciones si el recinto de movimiento de una baliza no es similar a un prisma o si quieren definirse zonas concretas del espacio como columnas o vigas. Un ejemplo de ello se muestra en la Fig.7.3.

Como diferencia fundamental respecto de las dos anteriores pestañas, mientras que en éstas era obligatorio definir al menos un volumen (necesitamos al menos un volumen que optimizar y una baliza), en esta última no será necesario definir ningún volumen prohibido. En caso de añadirse, se guardarán en la variable *self.VOLP*.

7.4 Definición de restricciones y parámetros generales de balizas

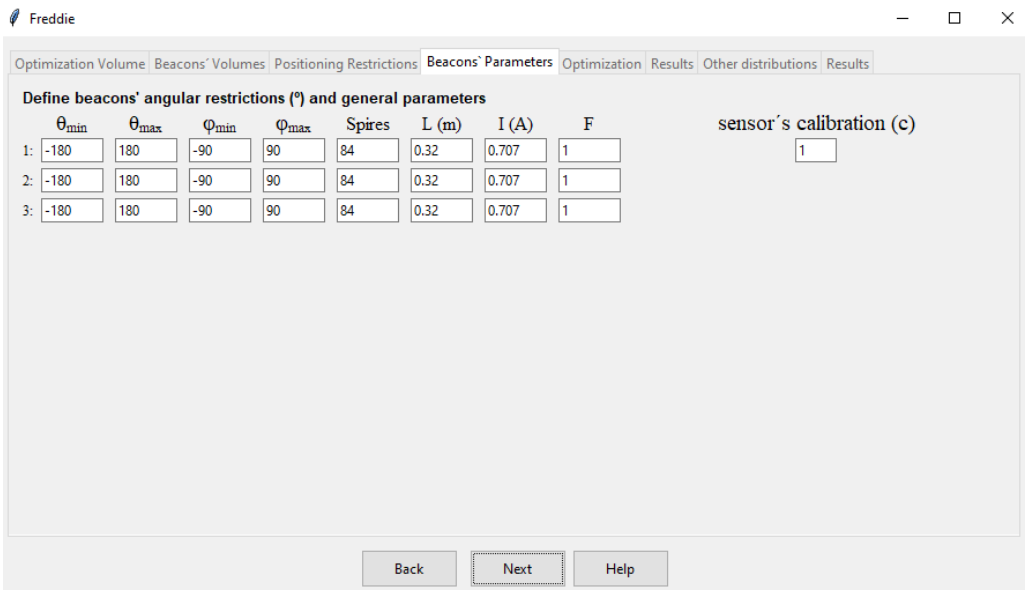


Figura 7.4 Pestaña 4: Restricciones angulares de las balizas y parámetros generales de éstas.

En la cuarta pestaña (Fig.7.4) se definen las restricciones angulares de todas las balizas (rango de ángulos en los que se pueden girar a la hora de optimizar) y los valores característicos de éstas, como son su número de espiras, la intensidad que recorre a cada una de ellas, su longitud o su factor $F(f)$. Además podremos configurar el factor C de calibración del sensor.

En este caso no encontramos botones para añadir o eliminar filas dado que se formarán automáticamente según el número de balizas que hayan sido definidas con anterioridad. Como ayuda al usuario, las entradas aparecen configuradas desde el inicio con los valores iniciales mostrados en la figura. Tampoco se permitirá avanzar si se deja en blanco alguno de ellos.

7.5 Parámetros de optimización y de cálculo y representación de la solución

El último paso para realizar la optimización del problema será definir los datos de mallado, los requisitos que debe cumplir un punto para ser óptimo y las iteraciones que se desean realizar. Estas variables se muestran en la Fig.7.5, junto con la posibilidad de definir un plano X, Y o Z concreto que optimizar.

Por otro lado, podrá indicarse si se quiere o no obtener los resultados gráficos y numéricos concretos de la solución mediante la función **Solucion**. En caso afirmativo, deberá definirse el mallado *ds* con el que se calculará la solución. También podrá indicarse si se requiere obtener la representación de puntos críticos y se muestran los parámetros necesarios para dicha representación,

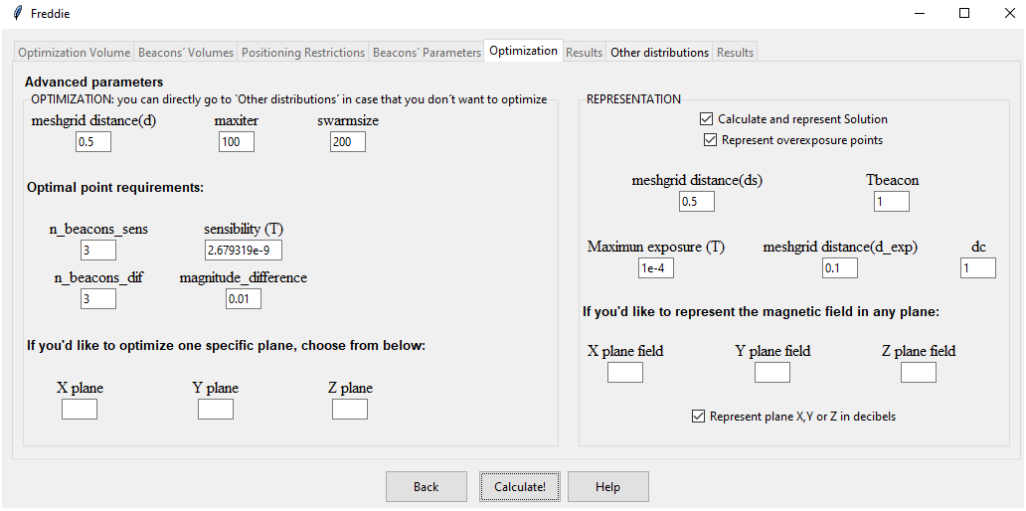


Figura 7.5 Pestaña 5: Parámetros de optimización y de cálculo y representación de la solución.

como el valor máximo de exposición o el mallado *d_exp*. Por último, podrán indicarse planos X, Y o Z para obtener la representación del campo magnético resultante en dichos planos, y la opción de representarlos en dB.

Todos los parámetros se mostrarán con una configuración inicial que servirá de referencia para el usuario. Los parámetros de optimización no podrán dejarse vacíos, mientras que los parámetros de representación sí podrán estarlo si no se pide calcular y representar la solución.

Una vez pulsado *Calculate!* se procederá a realizar la optimización. Cabe destacar que, en caso de definir un recinto complejo con volúmenes prohibidos muy amplios y un gran número de balizas; es posible que todas las semillas iniciales de la optimización (distintas configuraciones de posiciones y ángulos de las balizas) tengan balizas colocadas en alguno de estos volúmenes prohibidos. Si esto sucede, el valor de la función de coste será el mismo para todas las semillas (1e20) y la función **pso** no sabrá continuar, dando como solución una solución no válida. En este caso, se desplegará un warning (Fig.7.6) advirtiendo de lo sucedido y se recomendará aumentar el número de semillas (*swarmsize*) o cambiar la definición de las restricciones de forma que se reduzcan los volúmenes prohibidos.

Por otro lado, al llegar a la pestaña de optimización, una vez hemos definido los datos de las balizas en la pestaña anterior; también se activa directamente la pestaña *Other distributions* que se comentará más adelante. Esto es para poder realizar pruebas directamente con las distribuciones de balizas que se quiera sin tener que pasar por la optimización. De esta forma pueden realizarse cálculos para otras muchas distribuciones sin tener que esperar el tiempo de cálculo que implica la optimización.

7.6 Resultados de optimización

Tras realizar la optimización se mostrarán los resultados numéricos (Fig.7.7). En primer lugar se mostrarán las posiciones y ángulos óptimos obtenidos para cada baliza. En segundo lugar, se mostrará el tiempo empleando en la optimización y, en tercer lugar, si se ha pedido calcular y representar la solución, se desplegarán las figuras correspondientes a la solución y el resultado numérico de puntos óptimos obtenido.

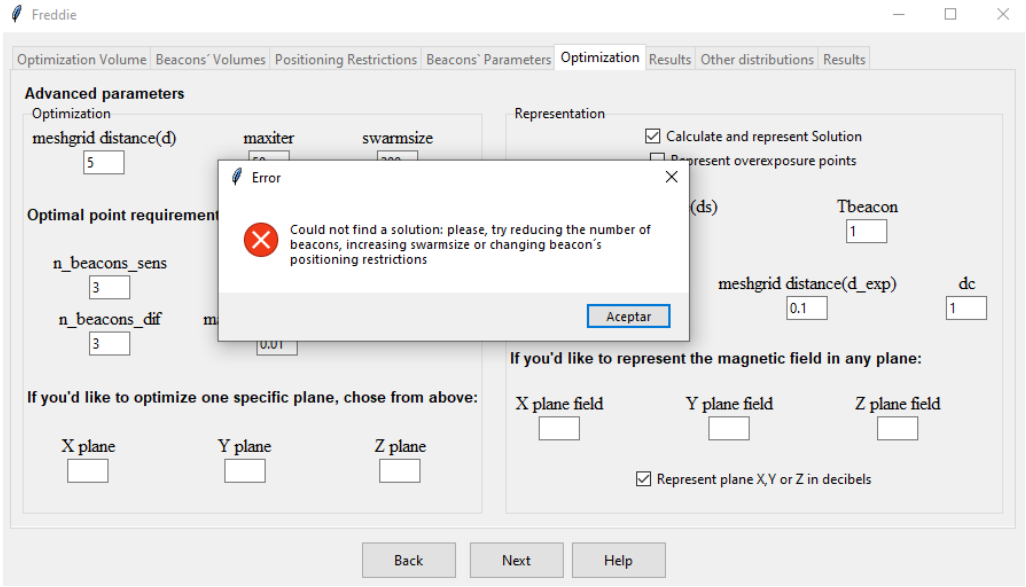


Figura 7.6 Warning: fallo al encontrar solución.

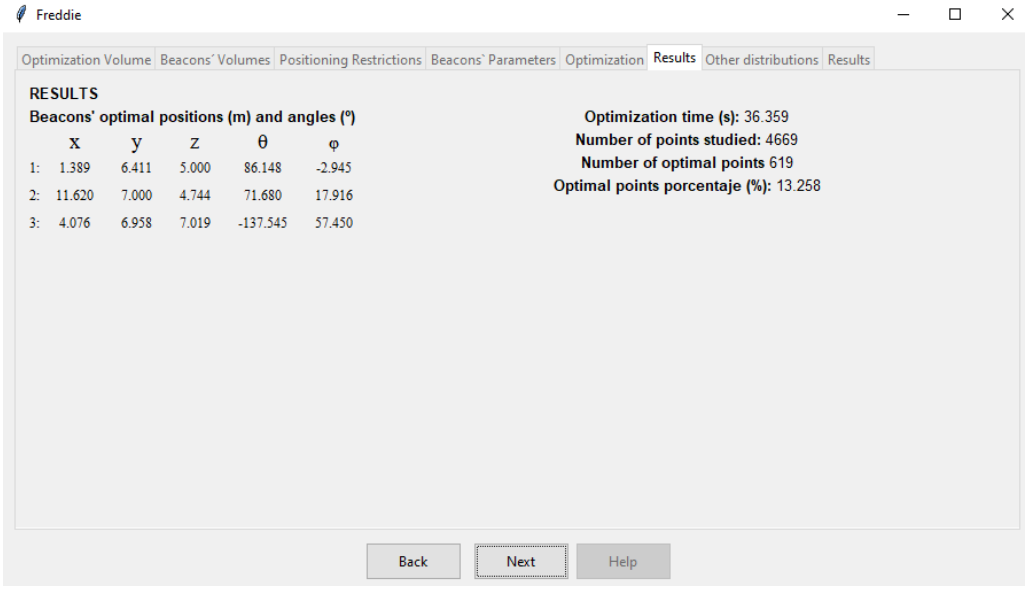


Figura 7.7 Pestaña 6: Resultados de optimización.

7.7 Otras distribuciones de balizas

Obtenidos los resultados tras optimización o directamente sin pasar por la optimización, se da la posibilidad de comprobar cualquier distribución de las balizas (posición y ángulos) que se desee y obtener sus resultados numéricos y gráficos; de la misma forma que se mostraban los resultados al realizar la optimización. También podrán representarse planos concretos del recinto y obtener el campo magnético que se produce en ellos. Esta opción puede ser interesante si se quiere comparar la solución obtenida mediante optimización frente a soluciones específicas.

Como ejemplo, si resulta que la posición de una de las balizas es cercana al techo, puede interesar colocarla directamente en éste (más fácil físicamente). De esta forma, se podrá comprobar si los resultados colocando la baliza en el techo no difieren demasiado de los óptimos.

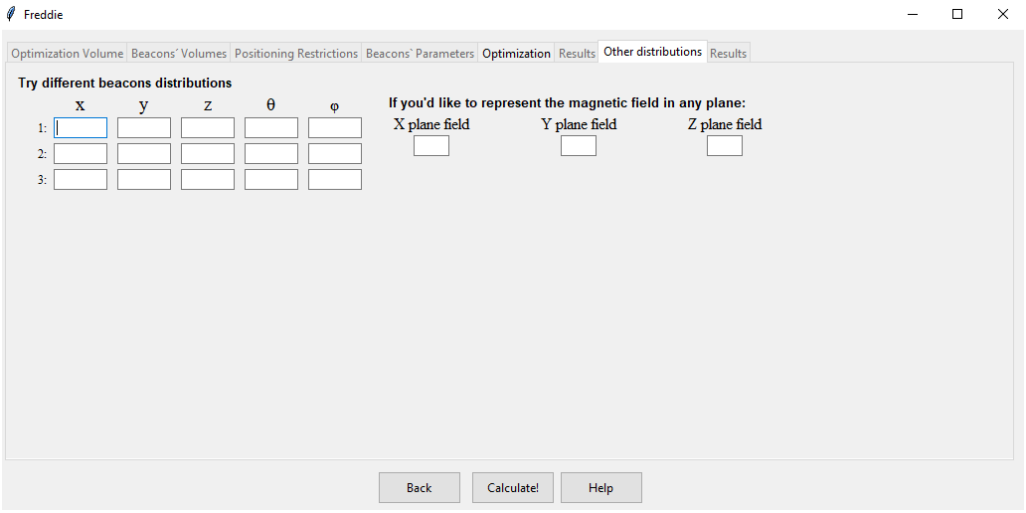


Figura 7.8 Pestaña 7: Otras distribuciones de balizas.

Será necesario que los datos del apartado de optimización esté rellenos, pues son los que indican el mallado de la representación, las condiciones de puntos óptimos o las condiciones de puntos críticos. En caso de darle al botón *Calculate!* y que éstos no estén rellenos, saltará un warning avisando de que debemos rellenar dichos datos.

8 Ejemplo de aplicación

Finalmente, en este capítulo se pondrá a prueba la herramienta desarrollada con un caso real. En concreto, se tratará de optimizar el campo generado por las balizas en un pasillo de la planta baja de la sección de consultas externas del hospital Virgen del Rocío de Sevilla, cuyos planos generales se muestran en la Fig.8.1.

La situación del pasillo concreto a optimizar, junto con sus dimensiones, se muestran en la Fig.8.2. Se tratará de buscar el número mínimo de balizas que produzcan un campo óptimo cercano al 100 % en el pasillo. Las balizas deberán estar colocadas en el techo, a 3.5m de altura y en horizontal. Para ello, se adoptarán diversas estrategias de definición del problema, definiendo de distintas formas los volúmenes de movimiento de las balizas; y se compararán los resultados.

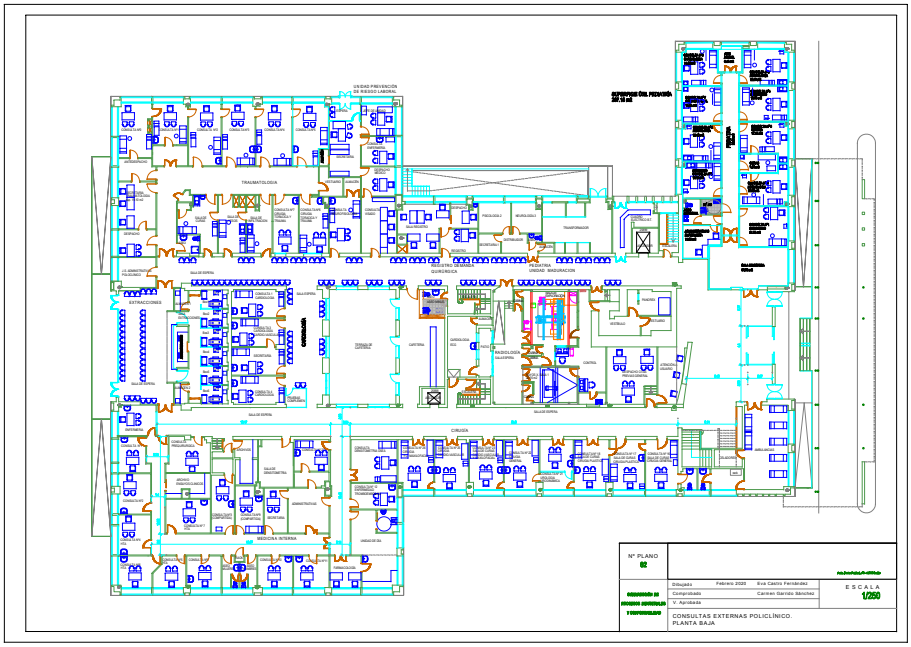
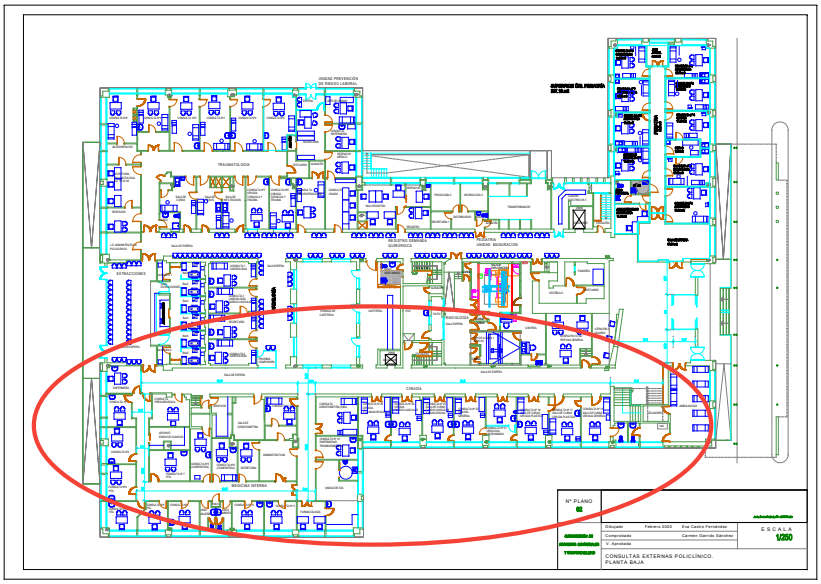
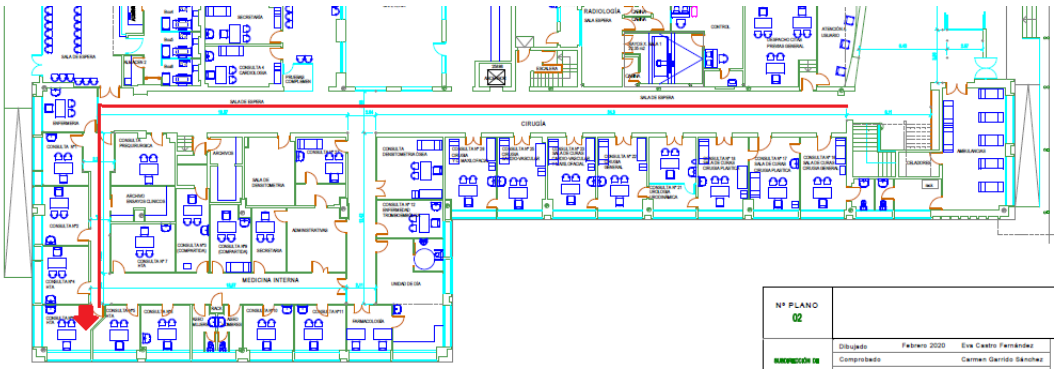


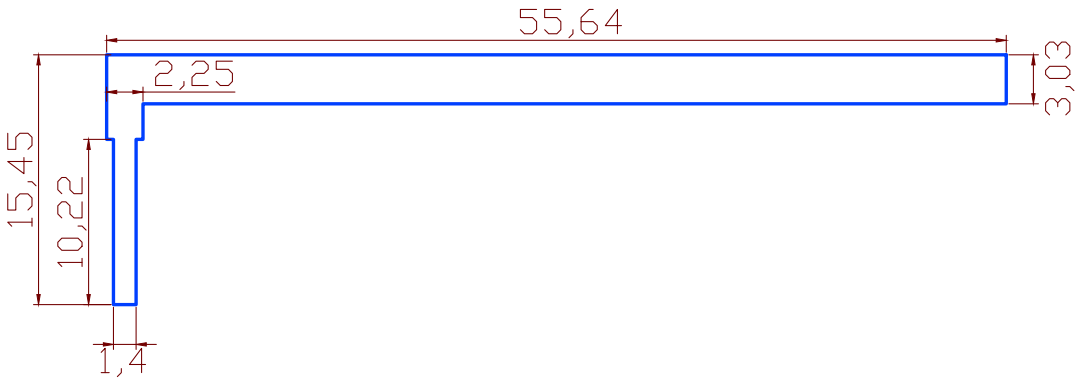
Figura 8.1 Sección de consultas externas, planta baja, hospital Virgen del Rocío.



(a) Situación dentro del plano general.



(b) Pasillo.



(c) Dimensiones Pasillo (m).

Figura 8.2 Pasillo concreto en el que optimizar el campo magnético creado por las balizas.

.

Todas las balizas serán iguales físicamente, de forma que todas tendrán el mismo número de espiras, serán recorridas por la misma intensidad y tendrán la misma longitud. Los valores concretos adoptados en este problema son los mostrados en la Tabla 8.1. Además, como viene siendo costumbre en los apartados anteriores, se congifuran las variables que definen los requisitos de puntos óptimos como *n_antenas_sens* = 3, *sensibilidad* = $2.679319 \cdot 10^{-9}$, *n_antenas_dif* = 3 y *diferencia_magnitudes* = 0.01 dado que son necesarias al menos 3 balizas para posicionamiento por triangulación y $2.679319 \cdot 10^{-9}$ es el valor proporcionado por *Skylife* para la sensibilidad del sensor.

Tabla 8.1 Parámetros físicos de balizas a utilizar.

Número de espiras	Intensidad (A)	Longitud (m)
500	1	0.6

8.1 Prueba 1: Balizas colocadas en el centro del pasillo

En primer lugar se probará a colocar las balizas en el centro del pasillo; de forma que sólo podrán moverse por las rectas azules representadas en la Fig.8.3. Además se configurarán para que todas tengan la misma orientación con $\theta = 0$ y $\phi = 90$; es decir, lados de las balizas paralelos a los lados del pasillo y en horizontal.

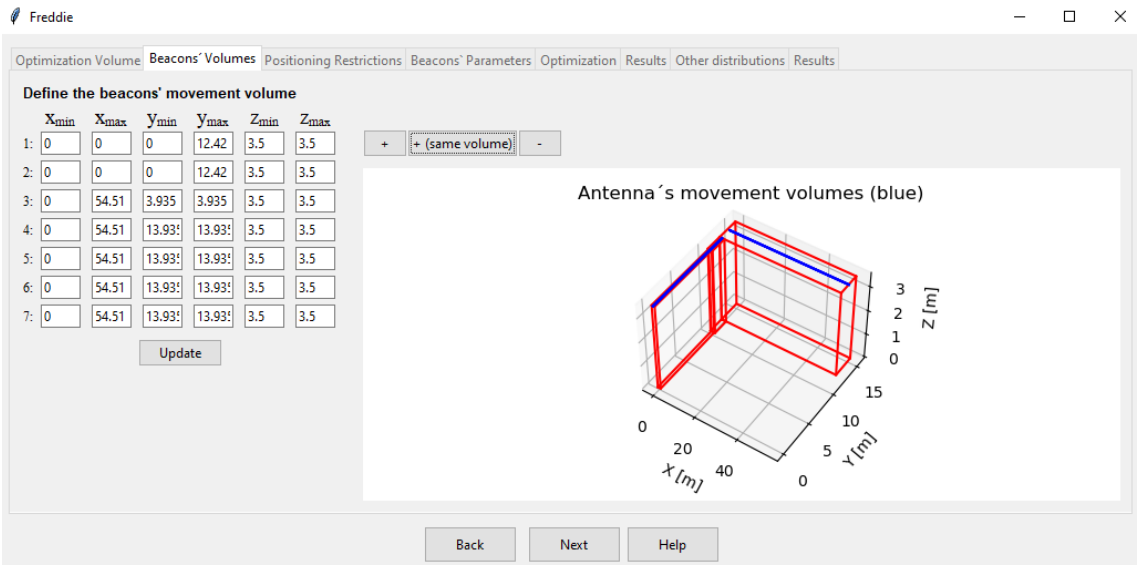


Figura 8.3 Zona de movimiento de balizas (azul) en Prueba 1.

Tras varias pruebas, variando el número de balizas, se encuentra que con 7 balizas se obtiene un porcentaje del 99.885 %[1], colocadas en las posiciones que se muestran en la Tabla 8.2. Para 6 balizas los resultados ya decaen a un rango no válido de entorno el 70 %. En la Fig.8.4 se puede observar el resultado gráfico del problema para 7 balizas.

[1] Para un mismo número de balizas, cada vez que se realice la optimización obtendremos un resultado ligeramente distinto dada la aletoriedad de la función **pso**. Sin embargo, estas diferencias son ínfimas, siendo los resultados prácticamente iguales.

Tabla 8.2 Posiciones de balizas resultantes de realizar optimización: Prueba 1.

Baliza	x (m)	y (m)	z (m)	θ (º)	ϕ (º)
1	0	6.940	3.5	0	90
2	0	9.316	3.5	0	90
3	37.015	13.935	3.5	0	90
4	23.049	13.935	3.5	0	90
5	10.374	13.935	3.5	0	90
6	41.458	13.935	3.5	0	90
7	53.458	13.935	3.5	0	90
Resultado	99.885 %				

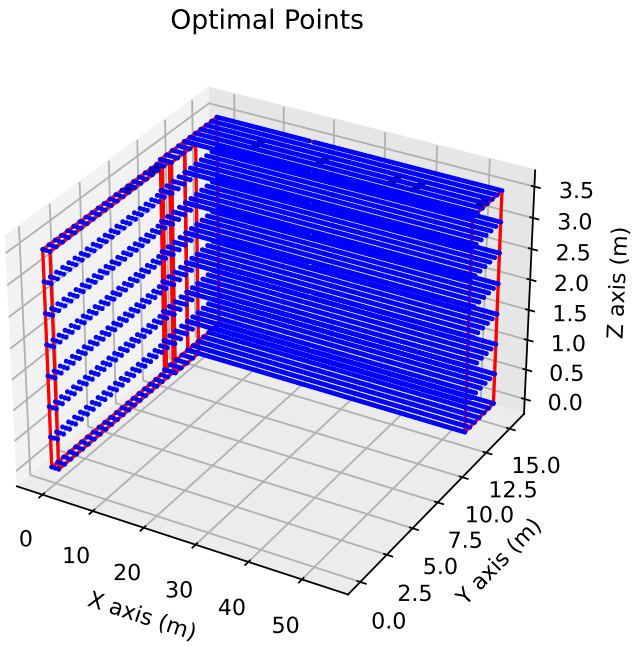


Figura 8.4 Resultado gráfico Prueba 1: 7 balizas.

8.2 Prueba 2: Balizas colocadas por todo el techo del pasillo

En esta ocasión se da la posibilidad de que las balizas se muevan por todo el techo y no solo por la zona central, tal como se muestra en la Fig.8.5. En primer lugar se prueba a optimizar con 7 balizas, siendo los resultados similares a los encontrados en la sección anterior, y en segundo lugar se prueba a realizar una optimización con 6 balizas. Dado que el rango de movimiento de las balizas es mayor que en la Prueba 1, es posible que 6 balizas puedan encontrar una posición capaz de dar una solución adecuada cercana al 100 %. Los ángulos seguirán estando restringidos a $\theta = 0$ y $\phi = 90$.

Finalmente se consigue obtener un resultado del 99.756 % de puntos óptimos con una baliza menos que en la sección anterior. Es decir, sólo serán necesarias 6 balizas. Sus posiciones y ángulos se muestran en la Tabla 8.3 y su resultado gráfico en la Fig.8.6.

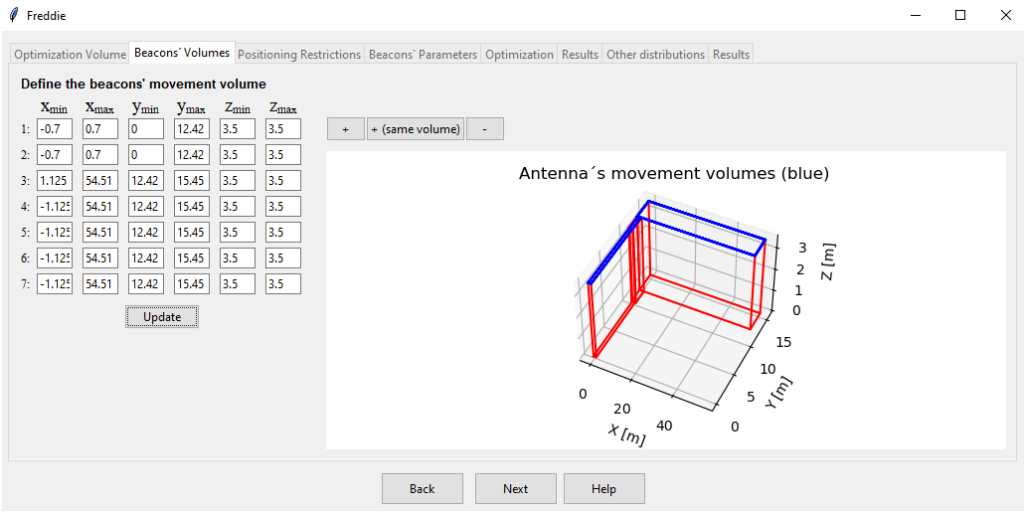


Figura 8.5 Zona de movimiento de balizas (azul) en Prueba 2.

Tabla 8.3 Posiciones de balizas resultantes de realizar optimización: Prueba 2.

Baliza	**x (m)**	**y (m)**	**z (m)**	θ (º)	ϕ (º)
1	0.327	7.090	3.5	0	90
2	-0.025	8.640	3.5	0	90
3	5.456	13.089	3.5	0	90
4	36.147	15.398	3.5	0	90
5	40.302	12.420	3.5	0	90
6	35.688	13.339	3.5	0	90
Resultado	99.756 %				

Cabe destacar que la función de optimización **pso** tiene carácter aleatorio dado que ésta toma semillas aleatorias a partir de las cuales comienza a buscar las posiciones óptimas. De esta forma, cada vez que se realiza una optimización se obtienen resultados ligeramente distintos. Es por ello que siempre que se ha llevado a cabo una prueba de optimización para una configuración inicial concreta (cierto número de balizas con cierto rango de movimiento concreto), ésta se ha realizado varias veces.

Además, el correcto desempeño del método está fuertemente ligado a los parámetros *swarmsize* y *maxiter* pues marcan el número de semillas e iteraciones a realizar. Cuantas más semillas y más iteraciones, mayores serán las posibilidades de encontrar soluciones satisfactorias, en detrimento del tiempo de ejecución. Por tanto, para cada configuración incial, no sólo se ha repetido la optimización en más de una ocasión, sino que también se han variado los parámetros *swarmsize* y *maxiter* hasta converger en la solución. Como resultado a destacar, los resultados obtenidos para una misma configuración de partida al alcanzar cierta convergencia en su solución nunca han variado en más de un $\pm 1\,\%$.

Esto es importante dado que nos lleva a confirmar que el haber sido capaces de reducir en 1 baliza la solución del problema en la Prueba 2 no ha sido fruto del carácter aleatorio de la función **pso**, sino que la definición de la configuración de partida del problema (en la que se daba más capacidad de movimiento a las balizas respecto a la Prueba 1) ha permitido obtener este nuevo resultado.

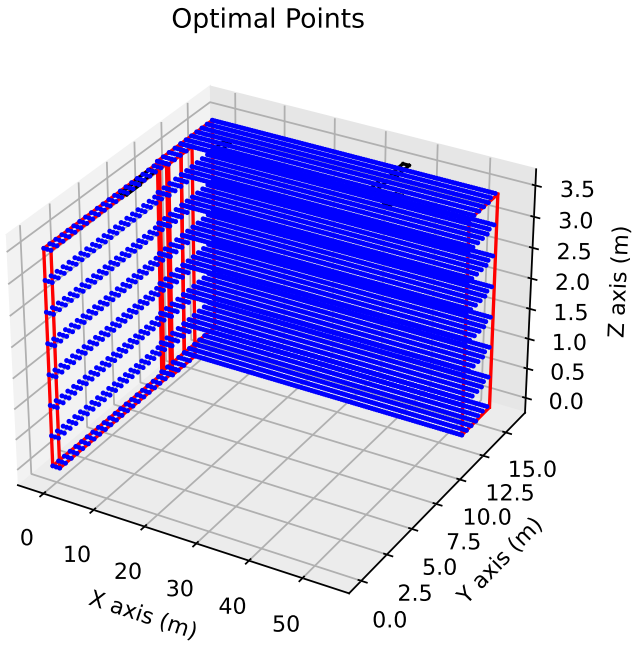


Figura 8.6 Resultado gráfico Prueba 2: 6 balizas.

8.3 Prueba 3: Definición de volumen de movimiento de balizas con volúmenes prohibidos

En los casos anteriores, se definieron las áreas en las que podían moverse las balizas de forma directa. Al no ser un área rectangular (pasillo con forma de L), se optó por definir dos áreas distintas, de forma que las 2 primeras balizas podían moverse por la zona del pasillo más estrecha y corta (zona vertical en la Fig.8.2(c)) y las restantes podían moverse por la zona más ancha y larga (zona horizontal).

Otra forma de definir el problema es la mostrada en las Fig.8.7 y Fig.8.8. En ella, se define un mismo área rectangular de movimiento para todas las balizas (azul) y luego se define otro área rectangular más pequeño (verde) en el que no podrán estar colocadas las balizas. De esta forma no se fuerza a ninguna baliza a estar colocada en ninguna zona concreta.

Dado que este será el último intento por reducir el número de balizas necesarias, se deja que los ángulos θ varíen entre -180º y 180º y se configura el problema para 5 balizas. Sin embargo, los resultados con 5 balizas no son del todo satisfactorios, obteniendo como mejor resultado un 60.665 % de puntos óptimos. El resultado gráfico de la solución encontrada se muestra en la Fig.8.9.

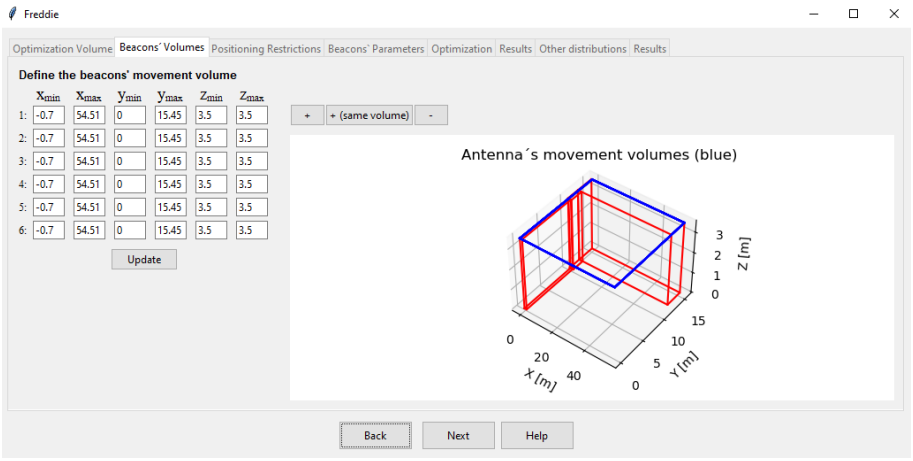


Figura 8.7 Área de movimiento de las balizas (azul).

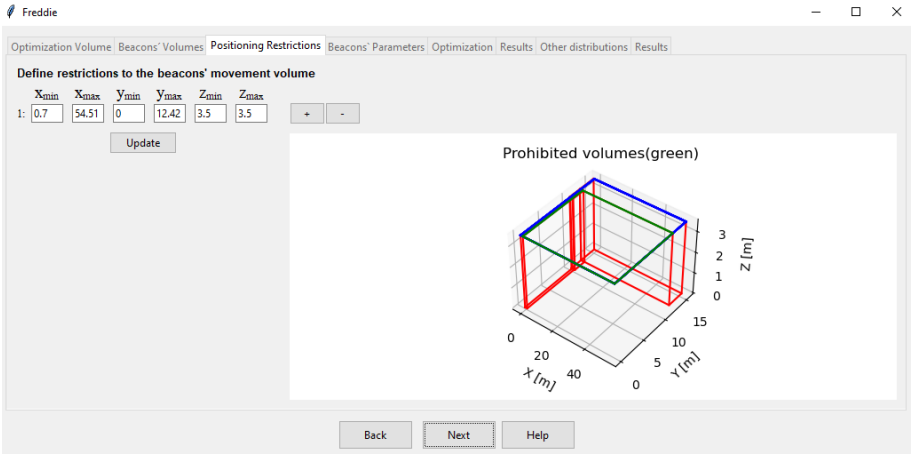


Figura 8.8 Volumen Prohibido (verde).

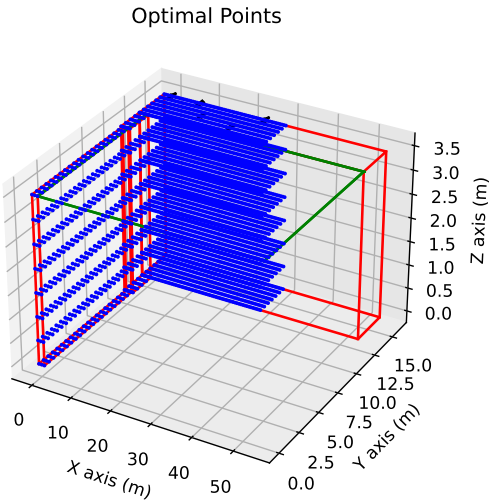


Figura 8.9 Resultado gráfico Prueba 3: 5 balizas.

8.4 Estudio de la solución

En la Tabla 8.4 se muestra un resumen de los resultados para las distintas pruebas realizadas y se concluye que la mejor solución será la obtenida en la Prueba 2. Pese a que la solución obtenida en la Prueba 1 tiene el mayor porcentaje de puntos óptimos (99.885 %), la Prueba 2 proporciona un ratio muy similar (99.756 %) con una baliza menos. La opción 3 es directamente descartable.

Tabla 8.4 Resultados Prueba 1, Prueba 2 y Prueba 3.

Prueba	Balizas utilizadas	% Puntos óptimos
1	7	99.885
2	6	99.756
3	5	60.665

Obtenida la solución, se realiza el estudio del campo magnético resultante con la distribución de balizas elegida. Como ejemplo, se muestra el campo resultante en el plano Z = 1m y en los planos X = 0 e Y = 14m (centro del pasillo). Los resultados se muestran en la Figuras 8.10, 8.11 y 8.12.

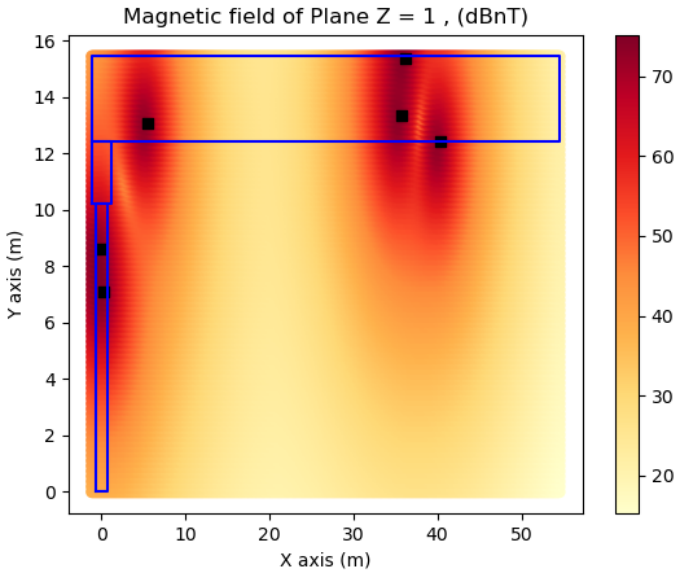


Figura 8.10 Campo magnético resultante en Z = 1m.

Como cabe esperar, se observa que en las zonas cercanas a las balizas se producen campos muy altos. Además, el campo que producen las balizas bajo ellas también es elevado dado que están a una altura de sólo 3.5m. Por otro lado, como es lógico, en el plano X=0 el campo predominante es el producido por las dos balizas que se colocan en el pasillo vertical; mientras que en el plano Y=14m el campo predominante es el producido por las cuatro balizas que se sitúan en el pasillo horizontal.

Como posible punto en contra de esta distribución se encuentra que en el centro del pasillo horizontal (Fig.8.12) puede observarse una zona bastante amplia (de los 10 a los 30m) con un campo relativamente bajo. El campo producido es superior a la sensibilidad del sensor dado que

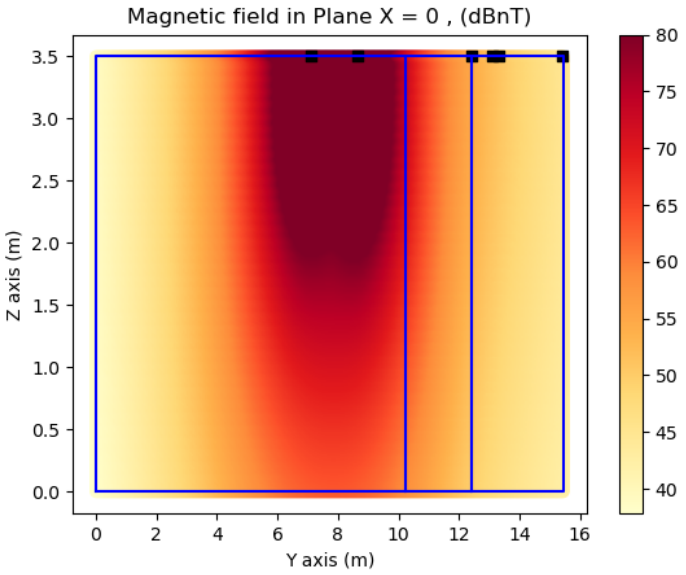


Figura 8.11 Campo magnético resultante en X = 0m.

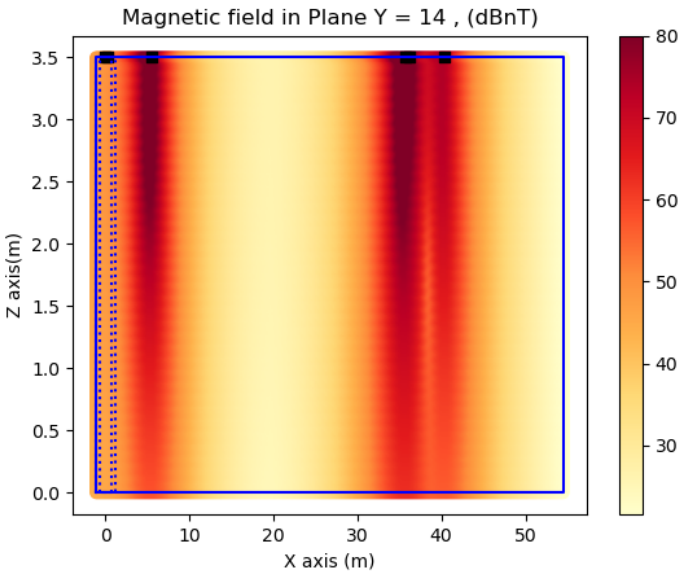


Figura 8.12 Campo magnético resultante en Y = 14m.

la optimización nos marcaba un 99.756 % de puntos óptimos, pero quizás podría encontrarse una distribución algo más equiespaciada de las 4 balizas que se sitúan en la zona horizontal del pasillo.

Se prueba esta idea con la distribución mostrada en la Tabla 8.5; en la que colocamos las 4 balizas situadas en el pasillo horizontal algo más equiespaciadas. Como puede observarse en la Fig.8.13 el campo resultante está mucho mejor distribuido a lo largo de todo el pasillo, lo que nos hace pensar que puede ser una distribución válida. Como contrapartida, el porcentaje de puntos óptimos disminuye significativamente hasta el 70.298 %. Los puntos óptimos se muestran en la Fig.8.14.

Tabla 8.5 Posiciones balizas: balizas equiespaciadas en pasillo horizontal.

Baliza	x (m)	y (m)	z (m)	θ (º)	ϕ (º)
1	0.327	7.090	3.5	0	90
2	-0.025	8.640	3.5	0	90
3	5	13.935	3.5	0	90
4	20	13.935	3.5	0	90
5	30	13.935	3.5	0	90
6	45	13.935	3.5	0	90
Resultado	70.298 %				

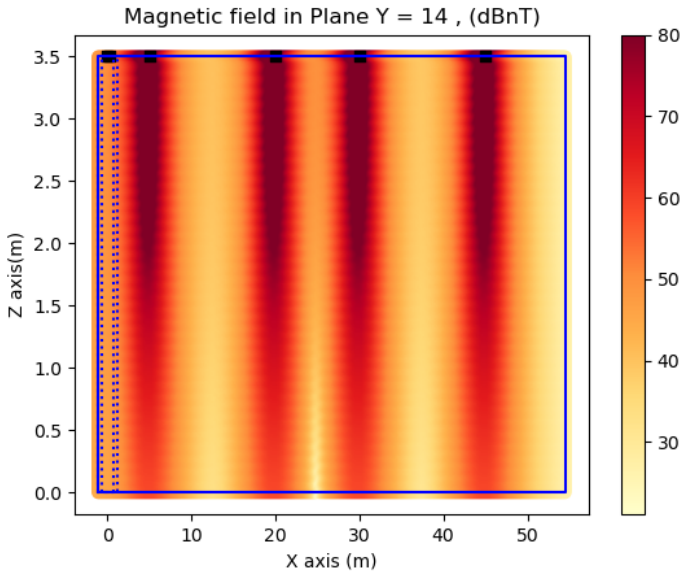


Figura 8.13 Campo magnético resultante en Y=14m con balizas equiespaciadas en pasillo horizontal.

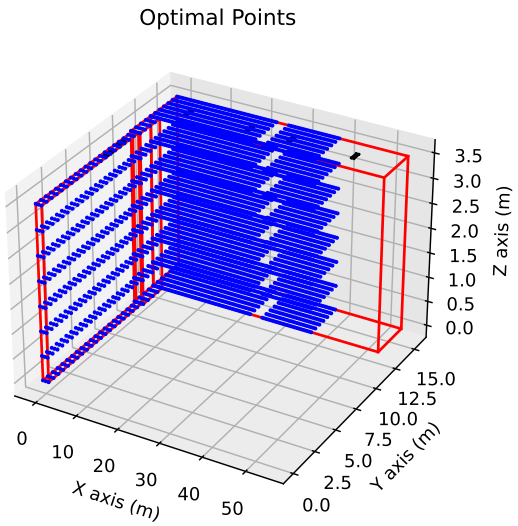


Figura 8.14 Puntos óptimos con balizas equiespaciadas en pasillo horizontal.

9 Conclusiones y mejoras futuras del software

Con la herramienta desarrollada en el presente trabajo se ha intentado facilitar una ayuda eficaz que permita establecer las mejores posiciones en las que situar ciertas balizas electromagnéticas, de forma que se optimice el campo creado por ellas en un recinto concreto según una serie de requisitos previamente impuestos. Estas balizas tendrán como finalidad última crear un sistema de posicionamiento basado en campos magnéticos de baja frecuencia, eficaz en entornos metálicos, mediante el cuál podrá realizarse posicionamiento por triangulación o mediante *fingerprinting*.

9.1 Funcionalidades finales incluidas en el software

Dado que estos métodos de posicionamiento basarán su funcionamiento en la medición del campo magnético creado por las distintas balizas a través de un sensor, será de vital importancia que éstas creen un campo óptimo que permita a los sensores situarse en cualquier punto dentro del recinto de estudio. En este sentido, el parámetro de mayor importancia será la sensibilidad del sensor, pues un sensor necesitará medir el campo de al menos 3 balizas para poder realizar de forma correcta posicionamiento por triangulación [2].

De esta forma, el software ha sido desarrollado para dar completa capacidad de definición de diversos requisitos que debe cumplir un punto óptimo; de forma que podrán configurarse dependiendo de las circunstancias del problema y la aplicación que se quiera dar a las balizas. Según estos requisitos y las características de las balizas y el sensor (también configurables) el software buscará las posiciones que optimizan el campo creado dentro del recinto de estudio. La optimización se realizará mediante optimización por enjambre de partículas [8] a través de la función **pso** [9] en Python.

Además, la herramienta ha sido pensada para ser lo más versátil posible a la hora de definir el recinto en el que optimizar el campo mediante el correcto posicionamiento de las balizas. De esta forma, el software nos brinda la posibilidad de crear recintos de gran complejidad de la forma más simple posible; pudiendo definirse recintos tridimensionales, bidimensionales o incluso monodimensionales. Por otro lado, también podrá definirse el rango de movimiento de todas y cada una de las balizas; pues es posible que éstas tengan zonas limitadas en las que poder ser posicionadas. Así pues, también se tendrá la posibilidad de definir en qué zonas el software debe buscar las posiciones de las balizas y en qué zonas estará prohibido colocarlas.

Una vez realizada la optimización, es necesario poder analizar los resultados. Es por ello que se han implementado funciones con capacidad de mostrar los distintos resultados, numérica y visualmente. Podrán calcularse y observarse los puntos óptimos dentro del recinto junto con las posiciones de las balizas para poder entender con mayor facilidad por qué las posiciones obtenidas son óptimas; y podrá estudiarse el campo magnético generado en distintos planos del recinto. Esto último será importante para analizar zonas conflictivas en las que el campo pueda ser demasiado bajo y también para analizar los mapeos necesarios a la hora de realizar posicionamiento mediante *fingerprinting*. Como ayuda extra, también se dará la posibilidad de poder estudiar cualquier configuración de balizas deseada sin tener que realizar la optimización. Esto también será de gran ayuda, pues es posible que aunque la optimización pueda dar los resultados óptimos, haya otras posiciones que también puedan ser de utilidad y más fáciles de implementar físicamente.

Por último, dado que el ser humano no puede estar expuesto a ciertos niveles de campo magnético regulados en el ámbito público [4] y en el ámbito laboral [5], también se ha incorporado la posibilidad de obtener gráficamente aquellos puntos en los que el campo creado por las balizas excede dichos niveles. Esto será muy importante dado el carácter de obligado cumplimiento de la normativa y dado que por lo general, las balizas no necesariamente estarán colocadas en zonas lejanas a las personas. Será necesario por tanto analizar siempre estos resultados y asegurar que las zonas críticas están fuera del alcance de los seres humanos.

Dada la complejidad del problema, que como ha podido verse contiene gran cantidad de variables y parámetros configurables, se ha aunado todo en una interfaz gráfica que permita al usuario definir el problema de la forma más visual y simple posible. Éste será guiado a través de las distintas pestañas que permitirán definir paso a paso el problema junto con ayudas explicativas y parámetros de referencia, de forma que será mucho más simple para cualquier persona poder realizar una correcta configuración del problema a resolver.

9.2 Complejidad del problema

Los problemas de optimización son diversos y pueden tener amplios rangos de complejidad. En este sentido, es importante comprender cuánto de complejo es el problema a resolver para poder elegir la mejor herramienta de resolución posible.

Como ha podido detectarse de la sección anterior, este no es un problema simple. En primer lugar, las restricciones al problema son diversas y pueden variar fácilmente en complejidad. Ejemplo de ello es el recinto de estudio, que puede ser tanto más complejo como pueda quererse. En segundo lugar, las variables modificables en la optimización (posiciones y ángulos de las balizas) no son fijas, pues su número aumentará conforme aumente el número de balizas. De esta forma, un amplio número de balizas creará un amplio número de variables aumentando por tanto la complejidad del problema. Un añadido más de complejidad encontramos también en la función de coste que permite obtener el número de puntos óptimos según la posición de las balizas, dado que esta función no será lineal debido a las funciones trigonométricas que aparecen al modelar los giros de las balizas.

Por todo ello se escoge como método de resolución la optimización por enjambre de partículas [8], que permite realizar optimizaciones en grandes espacios de soluciones candidatas. Concretamente, usando como ya se ha mencionado la función **pso** [9]. Sin embargo, con esta función habrá también que tener cuidado pues, como se ha visto en capítulos anteriores, la elección de los parámetros *swarmsize* (semillas) y *maxiter* será determinante en el correcto desempeño de la optimización.

9.3 Resultados en aplicación real de la herramienta

En el capítulo 8: *Ejemplo de aplicación*, se ha mostrado la resolución de un problema real. Tras analizar los resultados obtenidos, podemos extraer que se ha conseguido alcanzar un resultado ciertamente satisfactorio pero que sin embargo, la herramienta no puede ser utilizada sin un pensamiento previo del problema a resolver. Se ha podido observar que un mismo problema ha podido ser definido de diversas formas y que unas llevaban a soluciones más óptimas que otras; junto a que los parámetros *swarmsize* (semillas) y *maxiter* no pueden configurarse a la ligera.

Dicho esto, el software ha sido capaz de modelar un problema de cierta complejidad, pudiendo definir el recinto en el que optimizar el campo y las zonas de movimiento de balizas con cierta sencillez; y finalmente ha conseguido optimizar el campo creado en un pasillo de dimensiones considerables (orden de los $500 m^3$) con sólo 6 balizas, alcanzando casi el 100 % de puntos óptimos.

9.4 Futuras mejoras del software

Como todo software, éste está sujeto a futuras mejoras que puedan implementarse para mejorar la funcionalidad de la herramienta. A continuación se muestra una lista de posibles líneas de mejora que se han detectado durante su uso:

- **Mejora en tiempos de ejecución:** Como en todo software, los tiempos de ejecución son parte fundamental de la herramienta. En especial, la investigación y el estudio del algoritmo que permite obtener si un punto es o no óptimo debe estar siempre en mente.

  En este sentido, una posible mejora a la hora de realizar el cálculo de puntos óptimos sería ordenar las balizas por cercanía al punto de estudio, dado que a la hora de comprobar los requisitos que debe cumplir un punto óptimo, es más probable que estos se cumplan entre las balizas más cercanas. Sin embargo, esta reordenación de las balizas puede no ser eficiente; de forma que sería un nuevo frente a estudiar.

  Otra posibilidad se encuentra en cambiar el uso de matrices de cosenos directores por cuaterniones a la hora de modelar los giros de las balizas. Puede que este cambio produzca un ahorro en tiempos de computación y una simplificación en la optimización al eliminar los cálculos trigonométricos que deben realizarse con la matriz de cosenos directores.

- **Estudio de planos distintos a planos X,Y,Z:** a la hora de optimizar u obtener los campos magnéticos producidos en planos concretos, será una idea interesante extender la funcionalidad de la herramienta para poder especificar cualquier plano deseado según su ecuación y no sólo planos X, Y o Z.

- **BUG zonas prohibidas:** Como se ha mencionado en capítulos anteriores, en caso de tener un amplio número de balizas y una zona prohibida muy extensa, es posible que las semillas tomadas por el algoritmo de **pso** no den una solución correcta. Debe investigarse cómo solucionar este error sin que el usuario deba aumentar el número de semillas en exceso. Una posible idea es investigar cómo incluir una semilla inicial que sea correcta, para que el algoritmo itere a partir de ella.

- **Posibilidad de cancelación:** incluir la posibilidad de cancelación del proceso de optimización. En caso de introducir un dato erróneo, como por ejemplo un número de iteraciones excesivo, es importante poder cancelar el proceso para no tener que cerrar el programa y

repetir todo el proceso.

- **Mínimo número de balizas:** en vez de ser el usuario quien defina el número de balizas a utilizar, puede incluirse en el software la posibilidad de calcular el número mínimo de balizas necesarias para cumplir con unas especificaciones concretas de puntos óptimos.

- **Cargar setups:** incluir la posibilidad de incluir configuraciones predeterminadas del problema para no tener que definir todas las restricciones cada vez que se haga uso del programa.

- **Posibilidad de guardar y cargar solución:** una vez obtenida la solución tras optimización, una mejora interesante será poder cargar directamente los resultados en el apartado de *Other Distributions* y poder descargarlos en algún tipo de fichero.

- **Mejoras generales de interfaz:** mejorar la interfaz gráfica de forma que se haga más visual y amigable. Por ejemplo, incluir ayudas contextuales dinámicas que expliquen los diferentes parámetros de forma más directa que el botón *Help*, o incluir una explicación visual del significado de los ángulos de movimiento θ y ϕ de las distintas balizas.

Apéndice A
Códigos

A.1 Matriz de rotación

Código A.1 Matriz de rotación según ángulos de giro.

```
###########################################################################
"""FUNCION CALCULO MATRIZ DE ROTACION SEGUN ANGULOS THETA Y PHI"""
###########################################################################
import numpy as np
from math import pi, cos, sin, sqrt

def MatRotacion(theta,phi):
    #Definicion matrices de cambio de posicion a ejes cuerpo de antena
    Rotz=np.array([(cos(theta),sin(theta),0), #rotación Z
                   (-sin(theta),cos(theta),0),
                   (0,0,1)])
    Rotx=np.array([(1,0,0),                   #rotación X
                   (0,cos(phi),sin(phi)),
                   (0,-sin(phi),cos(phi))])
    Rot=Rotx.dot(Rotz)
    return(Rot)
```

A.2 Campo generado por un Dipolo

Código A.2 Cálculo campo magnético generado por una baliza.

```
###########################################################################
"""CAMPO MAGNETICO PRODUCIDO POR UNA BALIZA"""
###########################################################################

import numpy as np
from math import pi, cos, sin, sqrt
```

```
def Dipolo(Pos_antena,Pos_sensor,semilado, I, N_espiras,Rot):

    #ENTRADAS:
    #Pos_sensor: Lista [x,y,z]; punto donde se mide el campo (en
        coordenadas globales)
    #Pos_antena: Lista [x0,y0,z0]: posición de la antena en coordenadas
        globales
    #semilado: longitud del semilado de la espira
    #Rot: matriz de rotación según theta y phi

    #CONSTANTES:
    eps = np.finfo(float).eps
    A=(2*semilado)**2 #Area de la espira
    mu0=4*pi*1e-7  #permitividad del vacío

    x0, y0, z0=Pos_antena[0], Pos_antena[1], Pos_antena[2] #Pos antena
    x, y, z=Pos_sensor[0], Pos_sensor[1], Pos_sensor[2] #Posición sensor

    m=np.array([(0,I*A*N_espiras,0)]) #momento dipolar en ejes cuerpo
    Pos=np.array([x-x0,y-y0,z-z0]) #posicion relativa del sensor

    #Cambio posicion ejes globales a ejes cuerpo:
    Pos2=Rot.dot(Pos) #Posicion relativa en ejes cuerpo de la antena
    x2,y2,z2=Pos2[0],Pos2[1],Pos2[2]

    #CÁLCULO CAMPO MAGNETICO
    r=sqrt(x2**2 + y2**2 + z2**2) + eps
    a=(3*m.dot(Pos2))*Pos2 - ((r**2)*m)
    B=(mu0/(4*pi))*a/(r**5)
    modB=sqrt(B[0,0]**2 + B[0,1]**2 + B[0,2]**2)

    #campo magnético en sistema de referencia global
    B=(Rot.T).dot((B.T))
    B=(B.T) #campo magnetico en ejes globales

    return(B,modB)
```

A.3 Representación Campo generado por N balizas

Código A.3 Representación Campo generado por N balizas.

```
##########################################################################
"""REPRESENTACION CAMPO MAGNETICO DE VARIAS ANTENAS"""
##########################################################################
#Representacion líneas de campo 2D del campo magnetico generado por
    varias antenas situadas en distintas posiciones y con distintos
    angulos theta y phi
```

```
import numpy as np
import matplotlib.pyplot as plt
from math import pi, cos, sin, tan, sqrt
#importar 'MatRotaciones' y 'Dipolo'

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO"""

Pos_antena=[-1,0,0,1,0,...] #posicion antenas [x1,y1,z1,x2,y2,z2...]
N=len(Pos_antena)

xmin, xmax = min(Pos_antena[0:N:3]), max(Pos_antena[0:N:3])
    #Posicion minima y maxima en X de las antenas
ymin, ymax = min(Pos_antena[1:N:3]), max(Pos_antena[1:N:3])
    #Posicion minima y maxima en Y de las antenas
zmin, zmax = min(Pos_antena[2:N:3]), max(Pos_antena[2:N:3])
    #Posicion minima y maxima en Z de las antenas


semilado=0.17 #(m)
I=0.707 #(A)
N_espiras=84

theta=[pi/2,pi/4,0] #angulos theta de las antenas [theta1,theta2,...]
phi=[0,0,0] # " " phi " " [phi1,phi2,phi3...]
N=len(theta) #numero de antenas

#Creacion rejilla con puntos en los que evaluar el campo
xmax, xmin= xmax+1.5, xmin-1.5
ymax, ymin= ymax+1.5, ymin-1.5

nx, ny = 64,64 #numero de puntos en cada eje
x = np.linspace(xmin, xmax, num=nx) #puntos de x e y
y = np.linspace(ymin, ymax, num=ny)

##########################################################################
"""CÁLCULO COMPONENTES CAMPO MAGNETICO EN TODOS LOS PUNTOS DE LA MALLA
    """
##########################################################################
Bx, By= np.zeros((ny, nx)), np.zeros((ny, nx))
#Bx, By: matrices de dimensiones [ny X nx] donde guardamos las
    componentes del campo

l=0  #variable auxiliar para obtener [x0n,y0n,z0n] de cada antena en el
    bucle

for n in range(N):    #calculamos campo de cada antena
    Rot= MatRotacion(theta[n],phi[n]) #matriz Rot de la antena 'n'

    #calculamos campo en todos los puntos de la malla:
    for i in range(nx):
```

```
        for j in range(ny):

            Pos_sensor=[x[i],y[j],0] #Punto donde medimos el campo
            Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],Pos_antena[l+2]]
                #posicion de la antena 'n' [x0n,y0n,z0n]
            [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I,N_espiras
                ,Rot) #llamada a la funcion que proporciona el campo

            #guardamos valores de cada componente del campo debida a cada
                 antena y las sumamos (superposicion de campos):
            Bx[j,i] += B[0,0]
            By[j,i] += B[0,1]

    l+=3

#########################################################################
"""REPRESENTACION GRAFICA 2D"""
#########################################################################
fig = plt.figure(1)
ax = fig.add_subplot(111)

#Plot de streamlines y espiras
color = 2 * np.log(np.hypot(Bx, By))
ax.streamplot(x, y, Bx, By, color=color, linewidth=1, cmap=plt.cm.hot,
    density=2, arrowstyle='->', arrowsize=1.5)

k=0
for i in range(N):    #representacion antenas
    if theta[i]!=pi/2 or theta[i]!=-pi/2 :
        x = np.linspace(Pos_antena[k]-semilado*cos(theta[i]),Pos_antena[k
            ]+ semilado*cos(theta[i]), 100)
        y= np.linspace(Pos_antena[k+1]-semilado*sin(theta[i]),Pos_antena
            [k+1]+ semilado*sin(theta[i]),100)
        ax.plot(x, y)
    else:
        ax.axvline(Pos_antena[k],Pos_antena[k+1]-semilado, Pos_antena[k
            +1]+semilado)
    k=k+3

plt.title("Campo Magnético de N Antenas")
plt.xlabel("Coordenada x")
plt.ylabel("Coordenada y")
```

A.4 Función de coste

Código A.4 Función de Coste: Cálculo de puntos óptimos.

```
##########################################################################
"""CALCULO Y REPRESENTACION DE PUNTOS OPTIMOS"""
##########################################################################
#función que devuelve el numero de puntos optimos que cumplen las
   restricciones impuestas para que el punto sea óptimo y una
   representacion grafica de dichos puntos en 3D.

import numpy as np
from math import pi, cos, sin
import time
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
#importar 'MatRotaciones' y 'Dipolo'

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO"""

inicio=time.time() #cálculo de tiempo de ejecución

# CTES DE LAS BALIZAS:
semilado=[0.17,0.17,0.17] #(m)
I=[0.707,0.707,0.707] #(A)
N_espiras=[84,84,84]

"""INSERTAR POSICIONES DE ANTENAS"""
Pos_antena=[1,1,1,2,3,4,1,3,2]
   #posicion de las antenas [x1,y1,z1,x2,y2,z2...]

L=len(Pos_antena)
xmin, xmax = min(Pos_antena[0:L:3]), max(Pos_antena[0:L:3])
   #Posicion minima y maxima en X de las antenas
ymin, ymax = min(Pos_antena[1:L:3]), max(Pos_antena[1:L:3])
   #Posicion minima y maxima en Y de las antenas
zmin, zmax = min(Pos_antena[2:L:3]), max(Pos_antena[2:L:3])
   #Posicion minima y maxima en Z de las antenas


"""INSERTAR ANGULOS Y CONSTANTES"""
#ángulos theta de las antenas [theta1,theta2,theta3...]
theta=[3.14159265, 1.69845429, -3.14159265]
# phi = [phi1,phi2,phi3...]
phi=[-0.58019902, -1.30838587, -0.82235577]
c=1             #Constante de calibracion
f=[1,1,1]       #factor de conversion Vmedido=f*C*B
N=len(theta)    #número de antenas
```

```
#############################################################################
"""DEFINICION VOLUMEN DE CALCULO"""
#############################################################################
#VOLUMEN EN EL QUE SE VAN A CALCULAR LOS PUNTOS OPTIMOS

#Puntos máximos y mínimos del volumen a calcular: podemos especificarlos
    como mas nos convengan
xmax, xmin= xmax+4, xmin-4 #Ej: Posicion maxima de las antenas +-4
ymax, ymin= ymax+4, ymin-4
zmax, zmin= zmax+4, zmin-4

#número de puntos en cada eje: malla de 0.5m entre cada punto:
nx, ny = int(((xmax-xmin)/0.5)+1), int(((ymax-ymin)/0.5)+1)
nz=int(((zmax-zmin)/0.5)+1)

x = np.linspace(xmin, xmax, num=nx) #puntos de mallado de x,y,z
y = np.linspace(ymin, ymax, num=ny)
z = np.linspace(zmin, zmax, num=nz)
X,Y,Z=np.meshgrid(x,y,z)

#############################################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
#############################################################################
n_antenas_sens=3 #nº minimo de antenas que deben cumplir requisitos de
   sensibilidad en un punto
sensibilidad=2.679319e-09 #(T)

n_antenas_dif=3 #nº de pares de antenas cuya diferencia entre módulos
   debe cumplir con los requisitos
diferencia_magnitudes=0.01 #Dado en la forma: B1>0.01B2

#Creación matriz con matrices de rotación: Creamos una única matriz con
   todas las matrices de rotación de todas las antenas. Será una matriz
    de dimensiones [3,3*N] que podremos usar sin tener que calcular
   constantemente dichas matrices:

Rotaciones=MatRotacion(theta[0],phi[0])
for n in range(1,N):
    Rot=MatRotacion(theta[n],phi[n])
    Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)

Puntos_Optimos=0 #variable donde guardamos el nº de puntos optimos

for i in range(nx):
    for j in range(ny):
        for k in range(nz):

            #PRIMERA BALIZA:
            Pos_sensor=[x[i],y[j],z[k]]
            Pos_antena_n=[Pos_antena[0],Pos_antena[1],Pos_antena[2]]
```

```
[B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado[0], I[0],
    N_espiras[0], Rotaciones[0:3,0:3])
modB=f[0]*c*modB

CAMPOS=[modB] #Lista donde iremos guardando los modulos de
    las antenas [B1,B2,B3...]

dif_aux=0 #pares de antenas que cumplen requisitos de
    diferencia de modulos

sens_aux=0 #numero de antenas que cumplen requisitos de
    sensibilidad

if modB>sensibilidad:
    #Si cumple con sensibilidad añadimos 1 antena que cumple
    sens_aux+=1

l=3 #Repretimos este procedimiento para resto de antenas:
opt=1 #variable auxiliar
for n in range(1,N):
    if opt == 1: #si hemos detectado punto optimo, seguimos
        al siguiente
         continue
    Pos_sensor=[x[i],y[j],z[k]]
    Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],Pos_antena[l
        +2]]
    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado[n],I[n],
        N_espiras[n],Rotaciones[0:3,l:l+3])
    modB=f[n]*c*modB

    CAMPOS.append(modB) #añadimos campos [B1,B2,B3...]

    for r in range(n):
    #comparamos el modulo de esta antena 'n' con las antenas
        anteriores 'r=1,2,3...'

        if ((modB>CAMPOS[r] and CAMPOS[r]>
            diferencia_magnitudes*modB)
            or (CAMPOS[r]>modB and modB>diferencia_magnitudes*
                CAMPOS[r])):

            #EJ: si estamos en la 3a antena (n=2--->r=[0,1]),
                comparamos B3/B1 y B3/B2

            dif_aux+=1 #si la diferencia entre los módulos
                cumple los requisitos, añadimos un par


    if modB>sensibilidad:
```

```
                    sens_aux+=1 #Si cumple con sensibilidad añadimos 1
                        antena

                #si tenemos el nº mínimo de antenas con sensibilidad
                    suficiente y el nº mínimo de pares de antenas que
                    cumple con la diferencia de modulos, añadimos
                    punto_opt y pasamos al siguiente punto:

                if dif_aux>=n_antenas_dif and sens_aux>=n_antenas_sens:
                    Puntos_Optimos+=1
                    opt=1
                    continue

                #si al pasar por todas las antenas no es un punto óptimo:
                if n==N-1 and (dif_aux<=n_antenas_dif or sens_aux<=
                    n_antenas_sens):
                    Z[j,i,k]=np.nan #borramos para no representarlo

                l+=3

Puntos_Totales=nx*ny*nz
Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos

fin=time.time() #fin cálculo de puntos óptimos

###############################################################################
"""REPRESENTACION PUNTOS OPTIMOS EN EL VOLUMEN ESTUDIADO"""
###############################################################################
#REPRESENTACION GRAFICA 3D
figura = plt.figure()
grafica = figura.add_subplot(111,projection = '3d')
grafica.scatter3D(X,Y,Z, c='blue',marker='o',s=2)
    #Representamos los puntos óptimos

#REPRESENTACION BALIZAS: calculamos los 4 vértices de cada baliza y
    representamos las líneas que unen a dichos vértices
k=0
for n in range(N):

    x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
        #posicion de la antena 'n'
    Rot=Rotaciones[0:3,k:k+3]
        #matriz de rotación para la antena 'n'

    #Vértices de las antenas en ejes cuerpo
    p1, p2=[2*semilado,0,2*semilado], [-2*semilado,0,2*semilad
    p3, p4=[-2*semilado,0,-2*semilado],[2*semilado,0,-2*semilado]
    #agrandamos el tamaño de la antena para mejor visualizacion
    #vértices en ejes globales:
    p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
```

```
    p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

    P=np.concatenate((p1,p2,p3,p4,p1))
    for r in [0,3,6,9]:
    #Líneas que unen vértices:
        X=np.linspace(x+P[r],x+P[r+3],10)
        Y=np.linspace(y+P[r+1],y+P[r+4],10)
        Z=np.linspace(z+P[r+2],z+P[r+5],10)
        grafica.plot(X,Y,Z,color='black')

    k+=3

#representamos el volumen en donde estamos calculando los puntos óptimos
xx = [xmin, xmin, xmax, xmax, xmin]
yy = [ymin, ymax, ymax, ymin, ymin]
color='red'
kwargs = {'alpha': 1, 'color': color}
grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **kwargs)
grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **kwargs)
grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **kwargs)
grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **kwargs)

grafica.set_title('Puntos Óptimos')
grafica.set_xlabel('eje x')
grafica.set_ylabel('eje y')
grafica.set_zlabel('eje z')
plt.show()
#########################################################################

print('PUNTOS TOTALES:', Puntos_Totales)
print('PUNTOS OPTIMOS:', Puntos_Optimos)
print('Tiempo de ejecución:', fin-inicio)
```

A.5 Intersección entre volúmenes

Código A.5 Intersección entre volúmenes.

```
#########################################################################
"""FUNCION INTERSECCION DE VOLUMENES"""
#########################################################################
#Obtiene si se produce una interseccion en un conjunto de volumenes VOL
    =[[vol1],[vol2],[vol3]...]
def InterseccionVolumenes(VOL):

    T=False
    V=len(VOL) #numero de volumenes
```

```
    for i in range(V):
        xmax, xmin= VOL[i][0], VOL[i][1]
        ymax, ymin= VOL[i][2], VOL[i][3]
        zmax, zmin= VOL[i][4], VOL[i][5]
        for r in range(i+1,V):
            xmax2, xmin2= VOL[r][0], VOL[r][1]
            ymax2, ymin2= VOL[r][2], VOL[r][3]
            zmax2, zmin2= VOL[r][4], VOL[r][5]
            if ((xmin<xmin2<xmax or xmin<xmax2<xmax or (xmin2<=xmin and
                xmax2>=xmax)) and
                (ymin<ymin2<ymax or ymin<ymax2<ymax or (ymin2<=ymin and
                    ymax2>=ymax)) and
                (zmin<zmin2<zmax or zmin<zmax2<zmax or (zmin2<=zmin and
                    zmax2>=zmax))):

                 T=True  #True si se produce una interseccion

    return(T)
#########################################################################
```

A.6 Definición de parámetros del problema

Código A.6 Definición de parámetros del problema y llamada a las funciones de optimizacion y representación de la solución de forma manual.

```
import numpy as np
from math import pi, cos, sin
#importar OPTIMIZACION, SOLUCION, CRITICOS, RepresentacionPlano,
    MatRotacion, Dipolo, InterseccionVolumenes

#SE MUESTRA A CONTINUACIÓN LA FORMA DE DEFINIR TODOS LOS PARÁMETROS QUE
    RIGEN EL PROBLEMA Y LA FORMA DE LLAMAR A LAS DISTINTAS FUNCIONES QUE
     CALCULAN LAS SOLUCIONES A ESTE PROBLEMA Y SU REPRESENTACION:

#DATOS ANTENAS
semilado=[0.17,0.17,0.17]    #(m)
N_espiras=[84,84,84]         #Numero de espiras
I=[0.707,0.707,0.707 ]       #Intensidad que circula por la antena

#DATOS CALIBRACION
c=1                          #Calibracion
f=[1,1,1]                    #factor de conversion Vmedido=f*C*B

#################################################################
"""OPTIMIZACION"""
#################################################################

N=3              #numero de antenas
```

```
d=3               #distancia entre puntos del mallado
maxiter=100       #iteraciones de optimizacion
swarmsize=200     #semillas

#REQUISITOS:
#nº minimo de antenas que deben cumplir con sensibilidad en un punto:
n_antenas_sens=3
sensibilidad=2.679319e-09 #Sensibilidad mínima

#nº de pares de antenas cuya diferencia entre modulos debe cumplir con
    los requisitos:
n_antenas_dif=3
diferencia_magnitudes=0.01 #diferencia magitudes de antenas de forma que
     debe cumplirse: B1min>0.01B2max


#_________________ DIFINICION VOLUMEN DE CALCULO: _____________________
#volumenes en los que se calculara el nº de puntos no optimos: a elegir
    segun el problema
VOL=[]            #lista en la que guardamos todos los volumenes
#VOLUMEN 1:
xmax1, xmin1=2,0  #Puntos maximos y minimos del volumen
ymax1, ymin1=6,0
zmax1, zmin1=6,0
Vol=[xmax1,xmin1,ymax1,ymin1,zmax1,zmin1]
VOL.append(Vol)

#VOLUMEN 2:
xmax2, xmin2=6,0
ymax2, ymin2=8,6
zmax2, zmin2=6,0
Vol=[xmax2,xmin2,ymax2,ymin2,zmax2,zmin2]
"""activar la siguiente linea si tenemos 2 o mas volumenes"""
VOL.append(Vol)

#VOLUMEN 3:
xmax3, xmin3=4,2
ymax3, ymin3=6,4
zmax3, zmin3=6,0
Vol=[xmax3,xmin3,ymax3,ymin3,zmax3,zmin3]
"""activar la siguiente linea si tenemos 3 o mas volumenes"""
#VOL.append(Vol)

#COMPROBAR SI SE INTERSECTAN LOS VOLUMENES DE CALCULO:
Interseccion=InterseccionVolumenes(VOL)
if Interseccion==True:
    print('Warning: Los volúmenes se intersectan')

#PLANOS QUE PODEMOS OPTIMIZAR-->solo podemos elegir 1
PlanoZ=[3]                    #Plano Z en caso de optimizacion 2D
```

```
PlanoX=[]                    #Plano X en caso de optimizacion 2D
PlanoY=[]                    #Plano Y en caso de optimizacion 2D
"""Si PlanoZ,X,Y=[] (listas vacías)----->PROBLEMA 3D"""


#_________________ DIFINICION VOLUMEN PROHIBIDO: _________________________
#Volúmenes en los que no se pueden situar al antenas (columnas,etc)
VOLP=[]
#VOLUMEN PROHIBIDO 1:
xmaxp, xminp=1.5,1
ymaxp, yminp=4,2
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
VOLP.append(Volp)

#VOLUMEN PROHIBIDO 2:
xmaxp, xminp=3.5,3
ymaxp, yminp=5,4.5
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
"""activar la siguiente linea si tenemos 2 o mas volumenes"""
VOLP.append(Volp)

#VOLUMEN PROHIBIDO 3:
xmaxp, xminp=4,2
ymaxp, yminp=7,6.5
zmaxp, zminp=6,0
Volp=[xmaxp,xminp,ymaxp,yminp,zmaxp,zminp]
"""activar la siguiente linea si tenemos 3 o mas volumenes"""
#VOLP.append(Volp)

#_________________ DEFINICION RESTRICCIONES ANTENAS _____________________
#DEFINIR LOWER Y UPPER BOUNDS: define las zonas en las que pueden estar
   las antenas y sus angulos: son las restricciones del problema (
   posiciones y angulos entre los que se pueden mover las antenas)

#[x1min,y1min,z1min] // [x1max,y1max,z1max] ANTENA 1:
A1min, A1max=[0,0,0], [2,6,6]

#[x2min,y2min,z2min] // [x2max,y2max,z2max] ANTENA 2:
A2min, A2max=[0,6,0], [6,8,6]

#[x3min,y3min,z3min] // [x3max,y3max,z3max] ANTENA 3:
A3min, A3max=[2,4,0], [4,6,6]
"""Al observar la solución debe observarse que las antenas están en
   estos recintos"""

#[theta1min, theta2min...]//[theta1max,theta2max...]:
thetamin, thetamax=[-pi,-pi,-pi], [pi,pi,pi]
```

```
#[phi1min,phi2min...]//[phi1max,phi2max...]:
phimin, phimax=[-pi/2,-pi/2,-pi/2], [pi/2,pi/2,pi/2]

lb=A1min+A2min+A3min+thetamin+phimin #[x1min,y1min,z1min,x2min,y2min...
    theta1min,theta2min...,phi1min...] VALORES MINIMOS
ub=A1max+A2max+A3max+thetamax+phimax # VALORES MAXIMOS


######################################################################
"""REPRESENTACION DE LA SOLUCION Y PUNTOS CRITICOS"""
######################################################################
#distancia entre puntos al calcular y representar la solucion:
ds=0.5

#Maximo valor de campo magnetico al que el ser humano puede exponerse:
Exposicion=1e-4
#Distancia entre puntos al representar los puntos de maxima exposicion:
ds_exp=0.1
#Distancia alrededor de las antenas que queremos evaluar:
dc=1

Tant=2  #factor representacion de antenas


#En caso de querer representar el campo en algun plano concreto:
CampoZ=[]
CampoX=[4]
CampoY=[2]
dB=True #True si queremos representar como 20*log10(MODB/1e-9)==dBnT


######################################################################
"""LLAMADA A LA FUNCION OPTIMIZACION Y REPRESENTACION DE LA SOLUCION"""
######################################################################
if __name__ == '__main__':
    [Pos_antena,theta,phi]=Optimizacion(semilado,N_espiras,I,c,f,
        sensibilidad,d,N,Tant,maxiter, swarmsize, PlanoZ, PlanoX, PlanoY,
         VOL, VOLP, lb, ub, Exposicion,n_antenas_sens,n_antenas_dif,
        diferencia_magnitudes)

    #representacion de la solucion
    Solucion(Pos_antena,theta,phi,semilado,N_espiras,I,c,f,sensibilidad,
        N,ds,Tant,PlanoZ, PlanoX, PlanoY, VOL, VOLP,n_antenas_sens,
        n_antenas_dif,diferencia_magnitudes)

    #repreentacion de puntos criticos
    Criticos(Pos_antena,theta,phi,semilado,N_espiras,I,N,ds_exp,dc,Tant,
        VOL, Exposicion)
```

```
    #Si hemos hecho optimizacion 2D, representamos el plano optimizado:
    RepresentacionPlano(PlanoZ,PlanoX,PlanoY,Pos_antena,theta, phi,
        semilado, I, N_espiras,VOL, N, c,f, sensibilidad,dB)

    #En caso de querer ver el campo en un plano concreto:
    RepresentacionPlano(CampoZ,CampoX,CampoY,Pos_antena,theta, phi,
        semilado, I, N_espiras,VOL, N, c,f, sensibilidad,dB)
```

A.7 Optimización

Código A.7 Optimización: cálculo de posiciones óptimas de balizas.

```
import numpy as np
from math import pi, cos, sin, sqrt
import time
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D
from pyswarm import pso
#importar 'MatRotaciones' y 'Dipolo' y 'InterseccionVolumenes'

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO //
    INTERSECCION VOLUMENES"""

#########################################################################
"""FUNCION DE OPTIMIZACION-->MAXIMIZAR PUNTOS OPTIMOS"""
#Funcion a la que llamaremos para optimizar un volumen VOL
#########################################################################
def Optimizacion(semilado,N_espiras,I,c,f,sensibilidad,d,N,Tant,maxiter,
     swarmsize, PlanoZ, PlanoX, PlanoY, VOL, VOLP, lb, ub, Exposicion,
    n_antenas_sens,n_antenas_dif,diferencia_magnitudes):

    inicio=time.time()

    """(2) FUNCIONES DE COSTE"""
    #A continuación se definen las distintas funciones de coste que
        calculan el número de puntos óptimos en un problema 3D o en los
        problemas 2D
    #########################################################################
    """CALCULO PUNTOS NO OPTIMOS 3D// FUNCION DE COSTE"""
    #########################################################################
    def Puntos_No_Opt(s, *args):
    #Funcion para el calculo del nº de puntos no optimos

        """"Argumentos de entrada"""
        #s=[Pos_antena,theta,phi] VARIABLES DEL PROBLEMA: se da todo
            como 1 solo argumento de entrada y luego lo separamos por
            simplicidad para la llamada al algoritmo genetico
```

```
        #Pos_antena=[x1,y1,z1,x2,y2,z2...]
        #theta=[theta1,theta2,theta3...]
        #phi=[phi1,phi2,phi3...]

        #*args=(N,VOL=[Vol1;Vol2;Vol3....],VOLP=[Volp1,Volp2...],d):
            ARGUMENTOS OPCIONALES: definen el Nº de antenas(N), los
            volumenes (VOL) en los que calcular el nº de puntos optimos,
            los volúmens prohibidos (VOLP) y la distancia entre puntos (d
            )

        N,VOL,VOLP,d = args #guardamos los argumentos de entrada no
            variables
        V=len(VOL) #numero de volumenes a estudiar
        VP=len(VOLP) #numero de volumenes prohibidos

        #Obtenemos las listas Pos_antena, theta y phi a partir de (s) :
        Pos_antena=s[:(N*3)]
        theta=s[(N*3):(N*3 + N)]
        phi=s[(N*3 + N):]

        #Comprobamos que ninguna antena esté en uno de los volúmenes
            prohibidos:
        Posicion_No_Valida=False
        for v in range(VP):
            xmaxp, xminp= VOLP[v][0], VOLP[v][1]
            ymaxp, yminp= VOLP[v][2], VOLP[v][3]
            zmaxp, zminp= VOLP[v][4], VOLP[v][5]

            l=0
            for n in range(N):
                if (xminp<=Pos_antena[l]<=xmaxp and yminp<=Pos_antena[l
                    +1]<=ymaxp and zminp<=Pos_antena[l+2]<=zmaxp):
                    Puntos_No_Optimos=1e20
                    Posicion_No_Valida=True
                    #Si la antena está en una posición prohibida, haremos
                        que Puntos_No_Optimos sea muy grande para no
                        escoger dicho punto
                l+=3

        #Si ninguna antena está en una posición prohibida, continuamos:
        if Posicion_No_Valida==False:
            Puntos_Optimos=0
            Puntos_No_Optimos=0
            Puntos_Totales=0

            #Creacion matriz con matrices de rotacion
            Rotaciones=MatRotacion(theta[0],phi[0])
            for n in range(1,N):
                Rot=MatRotacion(theta[n],phi[n])
                Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
```

```
for v in range(V): #Hacemos calculos para cada volumen


    ############################################################
    """DEFINICION VOLUMEN DE CALCULO"""
    ############################################################
    #obtenemos puntos maximos y minimos definidos para el
        volume 'v'

    xmax, xmin= VOL[v][0], VOL[v][1]
    ymax, ymin= VOL[v][2], VOL[v][3]
    zmax, zmin= VOL[v][4], VOL[v][5]

    #numero de puntos en cada eje: malla de 'd' metros entre
        cada punto
    nx, ny = int(((xmax-xmin)/d)+1), int(((ymax-ymin)/d)+1)
    nz=int(((zmax-zmin)/d)+1)

    x = np.linspace(xmin, xmax, num=nx)
    y = np.linspace(ymin, ymax, num=ny)
    z = np.linspace(zmin, zmax, num=nz)


    ############################################################
    """ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
    ############################################################

    for i in range(nx):
        for j in range(ny):
            for k in range(nz):
                Pos_sensor=[x[i],y[j],z[k]]
                Pos_antena_n=[Pos_antena[0],Pos_antena[1],
                    Pos_antena[2]]
                [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                    semilado, I, N_espiras, Rotaciones
                    [0:3,0:3])
                modB=f[0]*c*modB

                CAMPOS=[modB]
                dif_aux=0
                sens_aux=0

                if modB>sensibilidad:
                    sens_aux+=1

                l=3
                opt=0
                for n in range(1,N):
                   if opt == 1:
```

```
                            continue
                        Pos_sensor=[x[i],y[j],z[k]]
                        Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                            Pos_antena[l+2]]
                         [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                            semilado, I, N_espiras, Rotaciones[0:3,
                            l:l+3])
                        modB=f[n]*c*modB

                        CAMPOS.append(modB)

                        for r in range(n):
                            if ((modB>CAMPOS[r] and CAMPOS[r]>
                                diferencia_magnitudes*modB)
                                or (CAMPOS[r]>modB and modB>
                                    diferencia_magnitudes*CAMPOS[r])
                                ):
                                dif_aux+=1

                         if modB>sensibilidad:
                            sens_aux+=1

                         if dif_aux>=n_antenas_dif and sens_aux>=
                            n_antenas_sens:
                            Puntos_Optimos+=1
                            opt=1
                            continue

                         l+=3

                Puntos_Totales+=nx*ny*nz #sumamos el nº de puntos del
                    volumen 'v'

            Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos
        return(Puntos_No_Optimos)
    ######################################################################
    ######################################################################
    """CALCULO PUNTOS NO OPTIMOS 2D PLANO Z// NO EDITAR"""
    ######################################################################
    def Puntos_No_Opt2DZ(s, *args):
    #Funcion para el calculo del nº de puntos no optimos en el plano '
        PlanoZ': muy similar al problema 3D, salvo que sólo calculamos
        los puntos óptimos en un plano del recinto que estamos estudiando

        N,VOL,VOLP,d,PlanoZ = args
        V=len(VOL)
        VP=len(VOLP)

        Pos_antena=s[:(N*3)]
        theta=s[(N*3):(N*3 + N)]
```

```
        phi=s[(N*3 + N):]

        #Comprobamos que ninguna antena esté en uno de los volúmenes
            prohibidos
        Posicion_No_Valida=False
        for v in range(VP):
            xmaxp, xminp= VOLP[v][0], VOLP[v][1]
            ymaxp, yminp= VOLP[v][2], VOLP[v][3]
            zmaxp, zminp= VOLP[v][4], VOLP[v][5]

            l=0

            for n in range(N):
                if (xminp<=Pos_antena[l]<=xmaxp and yminp<=Pos_antena[l
                    +1]<=ymaxp
                    and zminp<=Pos_antena[l+2]<=zmaxp):
                    Puntos_No_Optimos=1e20
                    Posicion_No_Valida=True
                l+=3
        #si las antenas no están en un volumen prohibido continuamos
        if Posicion_No_Valida==False:
            Puntos_Optimos=0
            Puntos_No_Optimos=0
            Puntos_Totales=0

            Rotaciones=MatRotacion(theta[0],phi[0])
            for n in range(1,N):
                Rot=MatRotacion(theta[n],phi[n])
                Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)

            for v in range(V): #calculamos puntos optimos en todos los
                volúmenes que forman el recinto
                ############################################################
                """DEFINICION AREA DE CALCULO"""
                ############################################################
                #obtenemos puntos maximos y minimos definidos para el
                    volumen 'v'

                xmax, xmin= VOL[v][0], VOL[v][1]
                ymax, ymin= VOL[v][2], VOL[v][3]
                zmax, zmin= VOL[v][4], VOL[v][5]

                #si el volumen no contiene ese plano Z, pasamos al
                    siguiente:
                if PlanoZ[0]<zmin or PlanoZ[0]>zmax:
                    continue

                #mallado
                nx, ny = int(((xmax-xmin)/d)+1), int(((ymax-ymin)/d)+1)
                x = np.linspace(xmin, xmax, num=nx)
```

```
y = np.linspace(ymin, ymax, num=ny)

############################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS EN EL AREA"""
############################################################
for i in range(nx):
    for j in range(ny):
        Pos_sensor=[x[i],y[j],PlanoZ[0]]
        Pos_antena_n=[Pos_antena[0],Pos_antena[1],
            Pos_antena[2]]
        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
            I, N_espiras, Rotaciones[0:3,0:3])
        modB=f[0]*c*modB

        CAMPOS=[modB]
        dif_aux=0
        sens_aux=0

        if modB>sensibilidad:
            sens_aux+=1

        l=3
        opt=1
        for n in range(1,N):
            if opt == 1:
               continue
            Pos_sensor=[x[i],y[j],PlanoZ[0]]
            Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                Pos_antena[l+2]]
            [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                semilado, I, N_espiras, Rotaciones[0:3,l:l
                +3])
            modB=f[n]*c*modB

            CAMPOS.append(modB)

            for r in range(n):
                if ((modB>CAMPOS[r] and CAMPOS[r]>
                    diferencia_magnitudes*modB) or (CAMPOS[
                    r]>modB and modB>diferencia_magnitudes*
                    CAMPOS[r])):
                    dif_aux+=1

            if modB>sensibilidad:
                sens_aux+=1

            if dif_aux>=n_antenas_dif and sens_aux>=
                n_antenas_sens:
                Puntos_Optimos+=1
                opt=1
```

```
                            continue

                        l+=3

            Puntos_Totales+=nx*ny
        Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos

    return(Puntos_No_Optimos)
#########################################################################
#########################################################################
"""CALCULO PUNTOS NO OPTIMOS 2D PLANO X// NO EDITAR"""
#########################################################################
def Puntos_No_Opt2DX(s, *args):
#igual que con 'PlanoZ' pero para 'PlanoX'

    N,VOL,VOLP,d,PlanoX = args
    V=len(VOL)
    VP=len(VOLP)

    Pos_antena=s[:(N*3)]
    theta=s[(N*3):(N*3 + N)]
    phi=s[(N*3 + N):]

    #Comprobamos que ninguna antena esté en uno de los volúmenes
       prohibidos
    Posicion_No_Valida=False
    for v in range(VP):
        xmaxp, xminp= VOLP[v][0], VOLP[v][1]
        ymaxp, yminp= VOLP[v][2], VOLP[v][3]
        zmaxp, zminp= VOLP[v][4], VOLP[v][5]

        l=0

        for n in range(N):
            if (xminp<=Pos_antena[l]<=xmaxp and yminp<=Pos_antena[l
                +1]<=ymaxp
                and zminp<=Pos_antena[l+2]<=zmaxp):
                Puntos_No_Optimos=1e20
                Posicion_No_Valida=True
            l+=3

    #Si ninguna antena está en una posición prohibida, continuamos:
    if Posicion_No_Valida==False:
        Puntos_Optimos=0
        Puntos_No_Optimos=0
        Puntos_Totales=0

        Rotaciones=MatRotacion(theta[0],phi[0])
        for n in range(1,N):
            Rot=MatRotacion(theta[n],phi[n])
```

```
            Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
        for v in range(V):
            ############################################################
            """DEFINICION AREA DE CALCULO"""
            ############################################################
            xmax, xmin= VOL[v][0], VOL[v][1]
            ymax, ymin= VOL[v][2], VOL[v][3]
            zmax, zmin= VOL[v][4], VOL[v][5]

            #si el volumen no contiene ese plano X, pasamos al
                siguiente:
            if PlanoX[0]<xmin or PlanoX[0]>xmax:
                continue

            ny, nz = int(((ymax-ymin)/d)+1), int(((zmax-zmin)/d)+1)
            y = np.linspace(ymin, ymax, num=ny)
            z = np.linspace(zmin, zmax, num=nz)

            ############################################################
            """ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
            ############################################################
            for j in range(ny):
                for k in range(nz):
                    Pos_sensor=[PlanoX[0],y[j],z[k]]
                    Pos_antena_n=[Pos_antena[0],Pos_antena[1],
                        Pos_antena[2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                         I, N_espiras, Rotaciones[0:3,0:3])
                    modB=f[0]*c*modB

                    CAMPOS=[modB]
                    dif_aux=0
                    sens_aux=0

                    if modB>sensibilidad:
                        sens_aux+=1

                    l=3
                    opt=0
                    for n in range(1,N):
                        if opt == 1:
                           continue
                        Pos_sensor=[PlanoX[0],y[j],z[k]]
                        Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                            Pos_antena[l+2]]
                        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                            semilado, I, N_espiras, Rotaciones[0:3,l:l
                            +3])
                        modB=f[n]*c*modB
```

```
                        CAMPOS.append(modB)

                        for r in range(n):
                            if ((modB>CAMPOS[r] and CAMPOS[r]>
                                diferencia_magnitudes*modB) or (CAMPOS[
                                r]>modB and modB>diferencia_magnitudes*
                                CAMPOS[r])):
                                dif_aux+=1

                        if modB>sensibilidad:
                            sens_aux+=1

                        if dif_aux>=n_antenas_dif and sens_aux>=
                            n_antenas_sens:
                            Puntos_Optimos+=1
                            opt=1
                            continue

                        l+=3

                Puntos_Totales+=ny*nz
            Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos
        return(Puntos_No_Optimos)
    ##########################################################################
    ##########################################################################
    """CALCULO PUNTOS NO OPTIMOS 2D PLANO Y// NO EDITAR"""
    ##########################################################################
    def Puntos_No_Opt2DY(s, *args):
    #Igual que con 'PlanoZ' y 'PlanoX'

        N,VOL,VOLP,d,PlanoY = args
        V=len(VOL)
        VP=len(VOLP)

        Pos_antena=s[:(N*3)]
        theta=s[(N*3):(N*3 + N)]
        phi=s[(N*3 + N):]

        #Comprobamos que ninguna antena esté en uno de los volúmenes
            prohibidos
        Posicion_No_Valida=False
        for v in range(VP):
            xmaxp, xminp= VOLP[v][0], VOLP[v][1]
            ymaxp, yminp= VOLP[v][2], VOLP[v][3]
            zmaxp, zminp= VOLP[v][4], VOLP[v][5]

            l=0
            for n in range(N):
                if (xminp<=Pos_antena[l]<=xmaxp and yminp<=Pos_antena[l
                    +1]<=ymaxp
```

```
                and zminp<=Pos_antena[l+2]<=zmaxp):
                Puntos_No_Optimos=1e20
                Posicion_No_Valida=True
            l+=3

    #Si ninguna antena está en una posición prohibida, continuamos:
    if Posicion_No_Valida==False:
        Puntos_Optimos=0
        Puntos_No_Optimos=0
        Puntos_Totales=0

        Rotaciones=MatRotacion(theta[0],phi[0])
        for n in range(1,N):
            Rot=MatRotacion(theta[n],phi[n])
            Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
        for v in range(V):
            ############################################################
            """DEFINICION AREA DE CALCULO"""
            ############################################################
            xmax, xmin= VOL[v][0], VOL[v][1]
            ymax, ymin= VOL[v][2], VOL[v][3]
            zmax, zmin= VOL[v][4], VOL[v][5]

            if PlanoY[0]<ymin or PlanoY[0]>ymax:
                continue

            nx, nz = int(((xmax-xmin)/d)+1), int(((zmax-zmin)/d)+1)
            x = np.linspace(xmin, xmax, num=nx)
            z = np.linspace(zmin, zmax, num=nz)

            ############################################################
            """ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
            ############################################################
            for i in range(nx):
                for k in range(nz):
                    Pos_sensor=[x[i],PlanoY[0],z[k]]
                    Pos_antena_n=[Pos_antena[0],Pos_antena[1],
                        Pos_antena[2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                         I, N_espiras, Rotaciones[0:3,0:3])
                    modB=f[0]*c*modB

                    CAMPOS=[modB]
                    dif_aux=0
                    sens_aux=0

                    if modB>sensibilidad:
                        sens_aux+=1

                    l=3
```

```
                    opt=0
                    for n in range(1,N):
                        if opt == 1:
                           continue
                        Pos_sensor=[x[i],PlanoY[0],z[k]]
                        Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                            Pos_antena[l+2]]
                        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                            semilado, I, N_espiras, Rotaciones[0:3,l:l
                            +3])
                        modB=f[n]*c*modB

                        CAMPOS.append(modB)

                        for r in range(n):
                            if ((modB>CAMPOS[r] and CAMPOS[r]>
                                diferencia_magnitudes*modB) or (CAMPOS
                                [r]>modB and modB>
                                diferencia_magnitudes*CAMPOS[r])):
                                dif_aux+=1
                        if modB>sensibilidad:
                            sens_aux+=1

                        if dif_aux>=n_antenas_dif and sens_aux>=
                            n_antenas_sens:
                            Puntos_Optimos+=1
                            opt=1
                            continue

                        l+=3

            Puntos_Totales+=nx*nz
        Puntos_No_Optimos=Puntos_Totales-Puntos_Optimos
    return(Puntos_No_Optimos)
#########################################################################




""" LLAMADA AL ALGORITMO GENETICO"""
#########################################################################
"""ALGORITMO GENETICO // NO EDITAR"""
#########################################################################
#Minimiza el Nº de puntos no optimos

n_bounds=len(ub)
for n in range(n_bounds):
    if lb[n]==ub[n]:
        ub[n]=lb[n]+0.01
```

```
        #si los limites inferiores y superiores son iguales, debemos añ
            adir un pequeño incremento para el correcto funcionamiento de
             la funcion pso

    #En caso de realizar optimizacion 3D:
    if PlanoZ==[] and PlanoX==[] and PlanoY==[]:
        args=(N,VOL,VOLP,d)
        SOLopt, fopt = pso(Puntos_No_Opt, lb, ub, args=args, maxiter=
            maxiter,swarmsize=swarmsize)
        #SOLopt=SOLUCION OPTIMA ENCONTRADA POR EL ALGORITMO

    #En caso de que solo queramos optimizar un solo plano Z
    if PlanoZ!=[]:
        args=(N,VOL,VOLP,d,PlanoZ)
        SOLopt, fopt = pso(Puntos_No_Opt2DZ, lb, ub, args=args, maxiter=
            maxiter,swarmsize=swarmsize)

    #En caso de que solo queramos optimizar un solo plano X
    if PlanoX!=[]:
        args=(N,VOL,VOLP,d,PlanoX)
        SOLopt, fopt = pso(Puntos_No_Opt2DX, lb, ub, args=args, maxiter=
            maxiter,swarmsize=swarmsize)

    #En caso de que solo queramos optimizar un solo plano Y
    if PlanoY!=[]:
        args=(N,VOL,VOLP,d,PlanoY)
        SOLopt, fopt = pso(Puntos_No_Opt2DY, lb, ub, args=args, maxiter=
            maxiter,swarmsize=swarmsize)

    #Obtenemos los valores de posiciones y angulos optimos obtenidos:
    Pos_antena=SOLopt[:(N*3)] #posicion optima de las antenas [x1,y1,z1,
        x2,y2,z2...]
    theta=SOLopt[(N*3):(N*3 + N)] #angulos theta optimos de las antenas
    phi=SOLopt[(N*3 + N):]

    #TIEMPO DE EJECUCION
    fin=time.time()
    tiempo=fin-inicio
    print('Tiempo de Optimización:', tiempo)

    ########################################################################
    #Finalmente Devolemos la solución obtenida
    return(Pos_antena,theta,phi,tiempo)
```

A.8 Solución tras optimización

Código A.8 Representación puntos óptimos de la solución obtenida tras optimización.

```
########################################################################
"""(3) CALCULO DE LA SOLUCION Y REPRESENTACION"""
########################################################################
import numpy as np
from math import pi, cos, sin, sqrt
import time
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D
from pyswarm import pso
#importar 'MatRotaciones' y 'Dipolo'

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO"""

def Solucion(Pos_antena,theta,phi,semilado,N_espiras,I,c,f,sensibilidad,
    N,ds,Tant,PlanoZ, PlanoX, PlanoY, VOL, VOLP,n_antenas_sens,
    n_antenas_dif,diferencia_magnitudes):

    ########################################################################
    """REPRESENTACION DE LA SOLUCION 3D O 2D"""
    ########################################################################
    #Representamos la solucion obtenida SOLopt para su visualizacion
        grafica. Se introducen las posiciones de las antenas obtenidas '
        Pos_antena' y sus angulos. Se vuelve a realizar el algoritmo de b
        úsqueda de putos optimos y se representa la solucion

    V=len(VOL)
    VP=len(VOLP)

    Puntos_Optimos=0
    Puntos_Totales=0

    Rotaciones=MatRotacion(theta[0],phi[0])
    for n in range(1,N):
        Rot=MatRotacion(theta[n],phi[n])
        Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
    figura = plt.figure()
    grafica = figura.add_subplot(111,projection = '3d')

    ########################################################################
    if PlanoZ==[] and PlanoX==[] and PlanoY==[]:
        """CALCULO Y REPRESENTACION GRAFICA DE LA SOLUCION 3D // NO
            EDITAR"""

        for v in range(V): #Hacemos calculos para cada volumen
```

```
############################################################
"""DEFINICION VOLUMEN DE CALCULO"""
############################################################
xmax, xmin= VOL[v][0], VOL[v][1]
ymax, ymin= VOL[v][2], VOL[v][3]
zmax, zmin= VOL[v][4], VOL[v][5]

nx, ny = int(((xmax-xmin)/ds)+1), int(((ymax-ymin)/ds)+1)
nz=int(((zmax-zmin)/ds)+1)

x = np.linspace(xmin, xmax, num=nx)
y = np.linspace(ymin, ymax, num=ny)
z = np.linspace(zmin, zmax, num=nz)
X,Y,Z=np.meshgrid(x,y,z)

############################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
############################################################

for i in range(nx):
    for j in range(ny):
        for k in range(nz):
            Pos_sensor=[x[i],y[j],z[k]]
            Pos_antena_n=[Pos_antena[0],Pos_antena[1],
                Pos_antena[2]]
            [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I
                , N_espiras, Rotaciones[0:3,0:3])
            modB=f[0]*c*modB

            CAMPOS=[modB]
            dif_aux=0
            sens_aux=0

            if modB>sensibilidad:
                sens_aux+=1

            l=3
            opt=1
            for n in range(1,N):
                if opt == 1:
                    continue
                Pos_sensor=[x[i],y[j],z[k]]
                Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                    Pos_antena[l+2]]
                [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,
                    semilado, I, N_espiras, Rotaciones[0:3,l:l
                    +3])
                modB=f[n]*c*modB
```

```
                        CAMPOS.append(modB)

                        for r in range(n):
                            if ((modB>CAMPOS[r] and CAMPOS[r]>
                                diferencia_magnitudes*modB) or (CAMPOS[r
                                ]>modB and modB>diferencia_magnitudes*
                                CAMPOS[r])):
                                dif_aux+=1

                        if modB>sensibilidad:
                            sens_aux+=1

                        if dif_aux>=n_antenas_dif and sens_aux>=
                            n_antenas_sens:
                            Puntos_Optimos+=1
                            opt=1
                            continue

                        #si al pasar por todas las antenas no es un
                            punto óptimo:
                        if n==N-1 and (dif_aux<n_antenas_dif or
                            sens_aux<n_antenas_sens):
                            Z[j,i,k]=np.nan #hará que no representemos
                                el punto

                        l+=3

    Puntos_Totales+=nx*ny*nz

    ############################################################
    """Representacion puntos optimos en cada volumen"""
    ############################################################
    #representamos los puntos óptimos de cada volumen:
    grafica.scatter3D(X,Y,Z, c='blue',marker='o',s=1.5)

    #representamos cada volumen
    xx = [xmin, xmin, xmax, xmax, xmin]
    yy = [ymin, ymax, ymax, ymin, ymin]
    color='red'
    kwargs = {'alpha': 1, 'color': color}
    grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
    grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
    grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
        kwargs)
    grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
        kwargs)
    grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
        kwargs)
    grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
        kwargs)
```

```
#Representamos los volumenes prohibidos:
for v in range(VP):
   xmaxp, xminp= VOLP[v][0], VOLP[v][1]
   ymaxp, yminp= VOLP[v][2], VOLP[v][3]
   zmaxp, zminp= VOLP[v][4], VOLP[v][5]

   xx = [xminp, xminp, xmaxp, xmaxp, xminp]
   yy = [yminp, ymaxp, ymaxp, yminp, yminp]
   color='green'
   kwargs = {'alpha': 1, 'color': color}
   grafica.plot3D(xx, yy, [zminp]*5, **kwargs)
   grafica.plot3D(xx, yy, [zmaxp]*5, **kwargs)
   grafica.plot3D([xminp, xminp], [yminp, yminp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xminp, xminp], [ymaxp, ymaxp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xmaxp, xmaxp], [ymaxp, ymaxp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xmaxp, xmaxp], [yminp, yminp], [zminp, zmaxp],
        **kwargs)

#Antenas:
k=0
for n in range(N):

    x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
    Rot=Rotaciones[0:3,k:k+3]

    #Vertices de las antenas en ejes cuerpo
    p1, p2=[Tant*semilado,0,Tant*semilado], [-Tant*semilado,0,
        Tant*semilado]
    p3, p4=[-Tant*semilado,0,-Tant*semilado], [Tant*semilado,0,-
        Tant*semilado]

    #vertices en ejes globales:
    p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
    p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

    for r in [0,3,6,9]:
        X=np.linspace(x+P[r],x+P[r+3],10)
        Y=np.linspace(y+P[r+1],y+P[r+4],10)
        Z=np.linspace(z+P[r+2],z+P[r+5],10)
        grafica.plot(X,Y,Z,color='black')
    k+=3

grafica.set_title('Puntos Óptimos en el Recinto')
grafica.set_xlabel('eje x (m)')
grafica.set_ylabel('eje y (m)')
```

```
        grafica.set_zlabel('eje z (m)')
        plt.show()

        Porcentaje_Puntos_Optimos=100*Puntos_Optimos/Puntos_Totales
        print('Número de puntos en el recinto:', Puntos_Totales)
        print('El número de puntos óptimos en el recinto es de',
            Puntos_Optimos)
        print('El porcentaje de Puntos Óptimos es del:',
            Porcentaje_Puntos_Optimos, '%')


    ######################################################################
    if PlanoZ!=[]:
        """CALCULO Y REPRESENTACION GRAFICA DE LA SOLUCION PLANO Z=CTE//
             NO EDITAR"""

        for v in range(V): #Hacemos calculos para cada volumen
            ############################################################
            """DEFINICION VOLUMEN(AREA) DE CALCULO"""
            ############################################################
            xmax, xmin= VOL[v][0], VOL[v][1]
            ymax, ymin= VOL[v][2], VOL[v][3]
            zmax, zmin= VOL[v][4], VOL[v][5]

            #Si el volumen no contiene el plano Z que estamos estudiando
                pasamos al siguiente:
            if PlanoZ[0]<zmin or PlanoZ[0]>zmax:
                continue

            nx, ny = int(((xmax-xmin)/ds)+1), int(((ymax-ymin)/ds)+1)
            nz=1
            x = np.linspace(xmin, xmax, num=nx)
            y = np.linspace(ymin, ymax, num=ny)
            z = np.linspace(PlanoZ[0],PlanoZ[0],num=nz)
            Xm,Ym,Zm=np.meshgrid(x,y,z)

            ############################################################
            """ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
            ############################################################
            for i in range(nx):
                for j in range(ny):

                    Pos_sensor=[x[i],y[j],PlanoZ[0]]
                    Pos_antena_n=[Pos_antena[0],Pos_antena[1],Pos_antena
                        [2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I,
                        N_espiras, Rotaciones[0:3,0:3])
                    modB=f[0]*c*modB

                    CAMPOS=[modB]
```

```
                dif_aux=0
                sens_aux=0

                if modB>sensibilidad:
                    sens_aux+=1

                l=3
                opt=0
                for n in range(1,N):
                    if opt == 1:
                           continue
                    Pos_sensor=[x[i],y[j],PlanoZ[0]]
                    Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                        Pos_antena[l+2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                        I, N_espiras, Rotaciones[0:3,l:l+3])
                    modB=f[n]*c*modB

                    CAMPOS.append(modB)

                    for r in range(n):
                        if ((modB>CAMPOS[r] and CAMPOS[r]>
                            diferencia_magnitudes*modB) or (CAMPOS[r]>
                            modB and modB>diferencia_magnitudes*CAMPOS[
                            r])):
                            dif_aux+=1

                    if modB>sensibilidad:
                        sens_aux+=1

                    if dif_aux>=n_antenas_dif and sens_aux>=
                        n_antenas_sens:
                        Puntos_Optimos+=1
                        opt=1
                        continue

                    #si al pasar por todas las antenas no es un punto
                        optimo:
                    if n==N-1 and (dif_aux<n_antenas_dif or sens_aux<
                        n_antenas_sens):
                        Zm[j,i]=np.nan

                    l+=3

        Puntos_Totales+=nx*ny

        ############################################################
        """Representacion puntos optimos en cada area"""
        ############################################################
        #representamos los puntos optimos de cada area:
```

```
            grafica.scatter3D(Xm,Ym,Zm,c='blue',marker='o',s=2)

            #representamos cada AREA de estudio
            xx = [xmin, xmin, xmax, xmax, xmin]
            yy = [ymin, ymax, ymax, ymin, ymin]
            color='grey'
            kwargs = {'alpha': 1, 'color': color}
            grafica.plot3D(xx, yy, [PlanoZ[0]]*5, **kwargs)
            grafica.plot3D([xmin, xmin], [ymin, ymin], [PlanoZ[0]], **
                kwargs)
            grafica.plot3D([xmin, xmin], [ymax, ymax], [PlanoZ[0]], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymax, ymax], [PlanoZ[0]], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymin, ymin], [PlanoZ[0]], **
                kwargs)

        #Representamos los volumenes:
        for v in range(V):
            xmax, xmin= VOL[v][0], VOL[v][1]
            ymax, ymin= VOL[v][2], VOL[v][3]
            zmax, zmin= VOL[v][4], VOL[v][5]

            xx = [xmin, xmin, xmax, xmax, xmin]
            yy = [ymin, ymax, ymax, ymin, ymin]
            color='red'
            kwargs = {'alpha': 1, 'color': color}
            grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
            grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
            grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
                kwargs)

        #Representamos los volumenes prohibidos:
        for v in range(VP):
            xmaxp, xminp= VOLP[v][0], VOLP[v][1]
            ymaxp, yminp= VOLP[v][2], VOLP[v][3]
            zmaxp, zminp= VOLP[v][4], VOLP[v][5]

            xx = [xminp, xminp, xmaxp, xmaxp, xminp]
            yy = [yminp, ymaxp, ymaxp, yminp, yminp]
            color='green'
            kwargs = {'alpha': 1, 'color': color}
            grafica.plot3D(xx, yy, [zminp]*5, **kwargs)
            grafica.plot3D(xx, yy, [zmaxp]*5, **kwargs)
```

```
        grafica.plot3D([xminp, xminp], [yminp, yminp], [zminp, zmaxp],
            **kwargs)
        grafica.plot3D([xminp, xminp], [ymaxp, ymaxp], [zminp, zmaxp],
            **kwargs)
        grafica.plot3D([xmaxp, xmaxp], [ymaxp, ymaxp], [zminp, zmaxp],
            **kwargs)
        grafica.plot3D([xmaxp, xmaxp], [yminp, yminp], [zminp, zmaxp],
            **kwargs)

    #Antenas:
    k=0
    for n in range(N):

        x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
        Rot=Rotaciones[0:3,k:k+3]
        #Vertices de las antenas en ejes cuerpo
        p1, p2=[Tant*semilado,0,Tant*semilado], [-Tant*semilado,0,
            Tant*semilado]
        p3, p4=[-Tant*semilado,0,-Tant*semilado], [Tant*semilado,0,-
            Tant*semilado]

        p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
        p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

        P=np.concatenate((p1,p2,p3,p4,p1))
        for r in [0,3,6,9]:
            X=np.linspace(x+P[r],x+P[r+3],10)
            Y=np.linspace(y+P[r+1],y+P[r+4],10)
            Z=np.linspace(z+P[r+2],z+P[r+5],10)
            grafica.plot(X,Y,Z,color='black')
        k+=3

    grafica.set_title('Puntos Óptimos en el Recinto. Optimización
        del plano Z = %i' %PlanoZ[0])
    grafica.set_xlabel('eje x (m)')
    grafica.set_ylabel('eje y (m)')
    grafica.set_zlabel('eje z (m)')
    plt.show()

    Porcentaje_Puntos_Optimos=100*Puntos_Optimos/Puntos_Totales
    print('Número de puntos en el recinto:', Puntos_Totales)
    print('El número de puntos óptimos en el recinto es de',
        Puntos_Optimos)
    print('El porcentaje de Puntos Óptimos es del:',
        Porcentaje_Puntos_Optimos, '%')


########################################################################
if PlanoX!=[]:
```

```
        """CALCULO Y REPRESENTACION GRAFICA DE LA SOLUCION PLANO X=CTE //
            NO EDITAR"""

        for v in range(V): #Hacemos calculos para cada volumen
           ##############################################################
           """DEFINICION VOLUMEN (AREA) DE CALCULO"""
           ##############################################################
           xmax, xmin= VOL[v][0], VOL[v][1]
           ymax, ymin= VOL[v][2], VOL[v][3]
           zmax, zmin= VOL[v][4], VOL[v][5]

           if PlanoX[0]<xmin or PlanoX[0]>xmax:
               continue

           ny, nz = int(((ymax-ymin)/ds)+1), int(((zmax-zmin)/ds)+1)
           nx=1
           x = np.linspace(PlanoX[0], PlanoX[0], num=nx)
           y = np.linspace(ymin, ymax, num=ny)
           z = np.linspace(zmin, zmax,num=nz)
           Xm,Ym,Zm=np.meshgrid(x,y,z)

           ##############################################################
           """ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
           ##############################################################
           for j in range(ny):
               for k in range(nz):

                    Pos_sensor=[PlanoX[0],y[j],z[k]]
                    Pos_antena_n=[Pos_antena[0],Pos_antena[1],Pos_antena
                        [2]]
                    [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I,
                        N_espiras, Rotaciones[0:3,0:3])
                    modB=f[0]*c*modB

                    CAMPOS=[modB]
                    dif_aux=0
                    sens_aux=0

                    if modB>sensibilidad:
                    l=3
                    opt=0
                    for n in range(1,N):
                        if opt == 1:
                               continue

                        Pos_sensor=[PlanoX[0],y[j],z[k]]
                        Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                            Pos_antena[l+2]]
                        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                            I, N_espiras, Rotaciones[0:3,l:l+3])
```

```
                modB=f[n]*c*modB

                CAMPOS.append(modB)

                for r in range(n):
                    if ((modB>CAMPOS[r] and CAMPOS[r]>
                        diferencia_magnitudes*modB) or (CAMPOS[r]>
                        modB and modB>diferencia_magnitudes*CAMPOS[
                        r])):
                        dif_aux+=1

                if modB>sensibilidad:
                    sens_aux+=1

                if dif_aux>=n_antenas_dif and sens_aux>=
                    n_antenas_sens:
                    Puntos_Optimos+=1
                    opt=1
                    continue

                if n==N-1 and (dif_aux<n_antenas_dif or sens_aux<
                    n_antenas_sens):
                    Xm[j,0,k]=np.nan

                l+=3

        Puntos_Totales+=ny*nz

        ############################################################
        """Representacion puntos optimos en cada volumen"""
        ############################################################
        #representamos los puntos optimos de cada area:
        grafica.scatter3D(Xm,Ym,Zm, c='blue',marker='o',s=2)
        #representamos cada AREA de estudio
        xmin, xmax= PlanoX[0], PlanoX[0]

        xx = [xmin, xmin, xmax, xmax, xmin]
        yy = [ymin, ymax, ymax, ymin, ymin]
        color='grey'
        kwargs = {'alpha': 1, 'color': color}
        grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
        grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
        grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
            kwargs)
```

```
#Representamos los volumenes:
for v in range(V):
   xmax, xmin= VOL[v][0], VOL[v][1]
   ymax, ymin= VOL[v][2], VOL[v][3]
   zmax, zmin= VOL[v][4], VOL[v][5]

   xx = [xmin, xmin, xmax, xmax, xmin]
   yy = [ymin, ymax, ymax, ymin, ymin]
   color='red'
   kwargs = {'alpha': 1, 'color': color}
   grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
   grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
   grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
       kwargs)
   grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
       kwargs)
   grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
       kwargs)
   grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
       kwargs)

#Representamos los volumenes prohibidos:
for v in range(VP):
   xmaxp, xminp= VOLP[v][0], VOLP[v][1]
   ymaxp, yminp= VOLP[v][2], VOLP[v][3]
   zmaxp, zminp= VOLP[v][4], VOLP[v][5]

   xx = [xminp, xminp, xmaxp, xmaxp, xminp]
   yy = [yminp, ymaxp, ymaxp, yminp, yminp]
   color='green'
   kwargs = {'alpha': 1, 'color': color}
   grafica.plot3D(xx, yy, [zminp]*5, **kwargs)
   grafica.plot3D(xx, yy, [zmaxp]*5, **kwargs)
   grafica.plot3D([xminp, xminp], [yminp, yminp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xminp, xminp], [ymaxp, ymaxp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xmaxp, xmaxp], [ymaxp, ymaxp], [zminp, zmaxp],
        **kwargs)
   grafica.plot3D([xmaxp, xmaxp], [yminp, yminp], [zminp, zmaxp],
        **kwargs)

#Antenas:
k=0
for n in range(N):

    x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
    Rot=Rotaciones[0:3,k:k+3]
```

```
        p1, p2=[Tant*semilado,0,Tant*semilado], [-Tant*semilado,0,
            Tant*semilado]
        p3, p4=[-Tant*semilado,0,-Tant*semilado], [Tant*semilado,0,-
            Tant*semilado]

        p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
        p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

        P=np.concatenate((p1,p2,p3,p4,p1))
        for r in [0,3,6,9]:
            X=np.linspace(x+P[r],x+P[r+3],10)
            Y=np.linspace(y+P[r+1],y+P[r+4],10)
            Z=np.linspace(z+P[r+2],z+P[r+5],10)
            grafica.plot(X,Y,Z,color='black')
        k+=3

    grafica.set_title('Puntos Óptimos en el Recinto. Optimización
        del plano X = %i' %PlanoX[0])
    grafica.set_xlabel('eje x (m)')
    grafica.set_ylabel('eje y (m)')
    grafica.set_zlabel('eje z (m)')
    plt.show()

    Porcentaje_Puntos_Optimos=100*Puntos_Optimos/Puntos_Totales
    print('Número de puntos en el recinto:', Puntos_Totales)
    print('El número de puntos óptimos en el recinto es de',
        Puntos_Optimos)
    print('El porcentaje de Puntos Óptimos es del:',
        Porcentaje_Puntos_Optimos, '%')



######################################################################
if PlanoY!=[]:
    """CALCULO Y REPRESENTACION GRAFICA DE LA SOLUCION PLANO Y=CTE//
        NO EDITAR"""

    for v in range(V): #Hacemos calculos para cada volumen

        ############################################################
        """DEFINICION VOLUMEN (AREA) DE CALCULO"""
        ############################################################
        xmax, xmin= VOL[v][0], VOL[v][1]
        ymax, ymin= VOL[v][2], VOL[v][3]
        zmax, zmin= VOL[v][4], VOL[v][5]

        if PlanoY[0]<ymin or PlanoY[0]>ymax:
            continue

        nx, nz = int(((xmax-xmin)/ds)+1), int(((zmax-zmin)/ds)+1)
```

```
ny=1

x = np.linspace(xmin, xmax, num=nx)
y = np.linspace(PlanoY[0], PlanoY[0], num=ny)
z = np.linspace(zmin, zmax,num=nz)
Xm,Ym,Zm=np.meshgrid(x,y,z)

############################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
############################################################
for i in range(nx):
    for k in range(nz):

        Pos_sensor=[x[i],PlanoY[0],z[k]]
        Pos_antena_n=[Pos_antena[0],Pos_antena[1],Pos_antena
            [2]]
        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I,
            N_espiras, Rotaciones[0:3,0:3])
        modB=f[0]*c*modB

        CAMPOS=[modB]
        dif_aux=0
        sens_aux=0

        if modB>sensibilidad:                          sens_aux+=1


        l=3
        opt=0
        for n in range(1,N):
            if opt == 1:
                  continue

            Pos_sensor=[x[i],PlanoY[0],z[k]]
            Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                Pos_antena[l+2]]
            [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                I, N_espiras, Rotaciones[0:3,l:l+3])
            modB=f[n]*c*modB

            CAMPOS.append(modB)

            for r in range(n):

                if ((modB>CAMPOS[r] and CAMPOS[r]>
                    diferencia_magnitudes*modB) or (CAMPOS[r]>
                    modB and modB>diferencia_magnitudes*CAMPOS[
                    r])):
                    dif_aux+=1
```

```
                    if modB>sensibilidad:
                        sens_aux+=1

                    if dif_aux>=n_antenas_dif and sens_aux>=
                        n_antenas_sens:
                        Puntos_Optimos+=1
                        opt=1
                        continue

                    if n==N-1 and (dif_aux<n_antenas_dif or sens_aux<
                        n_antenas_sens):
                        Ym[0,i,k]=np.nan

                    l+=3

        Puntos_Totales+=nx*nz

        ############################################################
        """Representacion puntos optimos en cada area"""
        ############################################################
        #representamos los puntos optimos de cada area:
        grafica.scatter3D(Xm,Ym,Zm, c='blue',marker='o',s=2)

        #representamos cada AREA de estudio
        ymin, ymax= PlanoY[0], PlanoY[0]

        xx = [xmin, xmin, xmax, xmax, xmin]
        yy = [ymin, ymax, ymax, ymin, ymin]
        color='grey'
        kwargs = {'alpha': 1, 'color': color}
        grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
        grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
        grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
            kwargs)

    #Representamos los volumenes:
    for v in range(V):
        xmax, xmin= VOL[v][0], VOL[v][1]
        ymax, ymin= VOL[v][2], VOL[v][3]
        zmax, zmin= VOL[v][4], VOL[v][5]

        xx = [xmin, xmin, xmax, xmax, xmin]
        yy = [ymin, ymax, ymax, ymin, ymin]
        color='red'
```

```
            kwargs = {'alpha': 1, 'color': color}
            grafica.plot3D(xx, yy, [zmin]*5, **kwargs)
            grafica.plot3D(xx, yy, [zmax]*5, **kwargs)
            grafica.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
                kwargs)
            grafica.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
                kwargs)

        #Representamos los volumenes prohibidos:
        for v in range(VP):
            xmaxp, xminp= VOLP[v][0], VOLP[v][1]
            ymaxp, yminp= VOLP[v][2], VOLP[v][3]
            zmaxp, zminp= VOLP[v][4], VOLP[v][5]

            xx = [xminp, xminp, xmaxp, xmaxp, xminp]
            yy = [yminp, ymaxp, ymaxp, yminp, yminp]
            color='green'
            kwargs = {'alpha': 1, 'color': color}
            grafica.plot3D(xx, yy, [zminp]*5, **kwargs)
            grafica.plot3D(xx, yy, [zmaxp]*5, **kwargs)
            grafica.plot3D([xminp, xminp], [yminp, yminp], [zminp, zmaxp],
                 **kwargs)
            grafica.plot3D([xminp, xminp], [ymaxp, ymaxp], [zminp, zmaxp],
                 **kwargs)
            grafica.plot3D([xmaxp, xmaxp], [ymaxp, ymaxp], [zminp, zmaxp],
                 **kwargs)
            grafica.plot3D([xmaxp, xmaxp], [yminp, yminp], [zminp, zmaxp],
                 **kwargs)

        #Antenas:
        k=0
        for n in range(N):

            x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
            Rot=Rotaciones[0:3,k:k+3]
            p1, p2=[Tant*semilado,0,Tant*semilado], [-Tant*semilado,0,
                Tant*semilado]
            p3, p4=[-Tant*semilado,0,-Tant*semilado], [Tant*semilado,0,-
                Tant*semilado]

            p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
            p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

            P=np.concatenate((p1,p2,p3,p4,p1))
            for r in [0,3,6,9]:
                X=np.linspace(x+P[r],x+P[r+3],10)
```

```
                Y=np.linspace(y+P[r+1],y+P[r+4],10)
                Z=np.linspace(z+P[r+2],z+P[r+5],10)
                grafica.plot(X,Y,Z,color='black')
            k+=3

        grafica.set_title('Puntos Óptimos en el Recinto. Optimización
            del plano Y = %i' %PlanoY[0])
        grafica.set_xlabel('eje x (m)')
        grafica.set_ylabel('eje y (m)')
        grafica.set_zlabel('eje z (m)')
        plt.show()

        Porcentaje_Puntos_Optimos=100*Puntos_Optimos/Puntos_Totales
        print('Número de puntos en el recinto:', Puntos_Totales)
        print('El número de puntos óptimos en el recinto es de',
            Puntos_Optimos)
        print('El porcentaje de Puntos Óptimos es del:',
            Porcentaje_Puntos_Optimos, '%')
```

A.9 Representación de puntos críticos: Umbrales magnéticos

Código A.9 Representación de puntos críticos: Umbrales magnéticos.

```
import numpy as np
from math import pi, cos, sin, sqrt
import time
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D
from pyswarm import pso
#importar 'MatRotaciones' y 'Dipolo''

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO"""

"""(4) ESTUDIO DE LOS UMBRALES DE CAMPO MAGNETICO ADMITIDOS: MAXIMA
    EXPOSICION CAMPO MAGNETICO ADMISIBLE""" #El ser humano no puede
    estar expuesto a cierto valor del campo magnetico

def Criticos(Pos_antena,theta,phi,semilado,N_espiras,I,N,ds_exp,dc,Tant,
    VOL, Exposicion):

    Rotaciones=MatRotacion(theta[0],phi[0])
    for n in range(1,N):
        Rot=MatRotacion(theta[n],phi[n])
        Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
    r=0

    figura2 = plt.figure()
```

```
    grafica2 = figura2.add_subplot(111,projection = '3d')

    for n in range(N):
    #calculamos puntos críticos de exposición en volumenes alrededor de
        las antenas: Posicion minima y maxima en X de las antenas +-dc
        xmin, xmax = Pos_antena[r]-dc/2,Pos_antena[r]+dc/2
        ymin, ymax = Pos_antena[r+1]-dc/2,Pos_antena[r+1]+dc/2
        zmin, zmax = Pos_antena[r+2]-dc/2,Pos_antena[r+2]+dc/2
        r+=3

        nx, ny = int(((xmax-xmin)/ds_exp)+1), int(((ymax-ymin)/ds_exp)+1)

        nz=int(((zmax-zmin)/ds_exp)+1)
        x = np.linspace(xmin, xmax, num=nx)
        y = np.linspace(ymin, ymax, num=ny)
        z = np.linspace(zmin, zmax, num=nz)
        X,Y,Z=np.meshgrid(x,y,z)

        ##################################################################
        """ALGORITMO BUSQUEDA DE PUNTOS CRITICOS"""
        ##################################################################
        for i in range(nx):
            for j in range(ny):
                for k in range(nz):
                    MODB=0 #Suma del módulo del campo de todas las
                        antenas

                    l=0
                    for n in range(N):
                        Pos_sensor=[x[i],y[j],z[k]]
                        Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                            Pos_antena[l+2]]
                        [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado,
                            I, N_espiras, Rotaciones[0:3,l:l+3])
                        MODB+=modB
                        l+=3

                    #Si el campo es menor al campo máximo admisible, no
                        representamos:
                    if MODB<Exposicion:
                        Z[j,i,k]=np.nan

        #Representamos puntos CRITICOS del volumen 'v':
        grafica2.scatter3D(X,Y,Z, c='orange',marker='o',s=2)

    #Representamos los volumenes de estudio:
    V=len(VOL)
    for v in range(V):
        xmax, xmin= VOL[v][0], VOL[v][1]
        ymax, ymin= VOL[v][2], VOL[v][3]
```

```
        zmax, zmin= VOL[v][4], VOL[v][5]

        xx = [xmin, xmin, xmax, xmax, xmin]
        yy = [ymin, ymax, ymax, ymin, ymin]
        color='red'
        kwargs = {'alpha': 1, 'color': color}
        grafica2.plot3D(xx, yy, [zmin]*5, **kwargs)
        grafica2.plot3D(xx, yy, [zmax]*5, **kwargs)
        grafica2.plot3D([xmin, xmin], [ymin, ymin], [zmin, zmax], **
            kwargs)
        grafica2.plot3D([xmin, xmin], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica2.plot3D([xmax, xmax], [ymax, ymax], [zmin, zmax], **
            kwargs)
        grafica2.plot3D([xmax, xmax], [ymin, ymin], [zmin, zmax], **
            kwargs)

    #Antenas:
    k=0
    for n in range(N):

         x, y, z = Pos_antena[k], Pos_antena[k+1], Pos_antena[k+2]
         Rot=Rotaciones[0:3,k:k+3]
         p1, p2=[Tant*semilado,0,Tant*semilado], [-Tant*semilado,0,Tant*
             semilado]
         p3, p4=[-Tant*semilado,0,-Tant*semilado], [Tant*semilado,0,-Tant
             *semilado]

         p1, p2 =(Rot.T).dot(p1), (Rot.T).dot(p2)
         p3, p4 =(Rot.T).dot(p3), (Rot.T).dot(p4)

         P=np.concatenate((p1,p2,p3,p4,p1))
         for r in [0,3,6,9]:
             X=np.linspace(x+P[r],x+P[r+3],10)
             Y=np.linspace(y+P[r+1],y+P[r+4],10)
             Z=np.linspace(z+P[r+2],z+P[r+5],10)
             grafica2.plot(X,Y,Z,color='black')
         k+=3

    grafica2.set_title('Puntos Críticos de Exposición Electromagnética:
        B > ' + str(format(Exposicion,'0.2e')) )
    grafica2.set_xlabel('eje x (m)')
    grafica2.set_ylabel('eje y (m)')
    grafica2.set_zlabel('eje z (m)')
```

A.10 Representación secciones de campo magnético

Código A.10 Representación secciones de campo magnético.

```
import numpy as np
from math import pi, cos, sin, sqrt
import time
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.mplot3d import Axes3D
from pyswarm import pso
#importar 'MatRotaciones' y 'Dipolo'

"""FUNCIONES AUXILIARES: MATRIZ ROTACION // CALCULO CAMPO MAGNETICO"""

"""(5) REPRESENTACION DEL CAMPO MAGNETICO EN UN PLANO X, Y o Z: Funcion
    para la representacion del valor del campo magnetico en un plano
    cualquiera X,Y o Z = cte. El plano que quiere representarse se da
    como una lista: RepresentacionZ=[*] o RepresentacionX=[*] o
    RepresentacionY=[*]."""

def RepresentacionPlano(RepresentacionZ,RepresentacionX,RepresentacionY,
    Pos_antena,theta,phi,semilado, I, N_espiras,VOL, N, c,f,
    sensibilidad,dB):

    Rotaciones=MatRotacion(theta[0],phi[0])
    for n in range(1,N):
        Rot=MatRotacion(theta[n],phi[n])
        Rotaciones=np.concatenate((Rotaciones,Rot),axis=1)
    V=len(VOL)

    #Puntos maximos y minimos de entre todos los volumenes de VOL:
    xmax,xmin= (max(VOL, key=lambda x: x[0]))[0], (min(VOL, key=lambda x:
         x[1]))[1]
    ymax,ymin= (max(VOL, key=lambda x: x[2]))[2], (min(VOL, key=lambda x:
         x[3]))[3]
    zmax,zmin= (max(VOL, key=lambda x: x[4]))[4], (min(VOL, key=lambda x:
         x[5]))[5]

    #Si nos piden un plano Z=cte:
    if RepresentacionZ!=[]:
        ###################################################################
        """DEFINICION AREA DE CALCULO"""
        ###################################################################
        #Creacion rejilla con puntos en los que evaluar el campo en ese
            volumen :
        nx, ny = int(((xmax-xmin)/((xmax-xmin)/300))+1), int(((ymax-ymin)
            /((xmax-xmin)/300))+1)
```

```
#300 puntos en cada eje: (más puntos en la representación para
    mejor visualizacion)

x = np.linspace(xmin, xmax, num=nx)
y = np.linspace(ymin, ymax, num=ny)
Xm,Ym=np.meshgrid(x,y)

#################################################################
"""ALGORITMO CALCULO CAMPO MAGNETICO"""
#################################################################
Bx,By,Bz=np.zeros((ny,nx)),np.zeros((ny,nx)), np.zeros((ny,nx))

for i in range(nx):
     for j in range(ny):

          l=0
          for n in range(N):
               Pos_sensor=[x[i],y[j],RepresentacionZ[0]]
               Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                   Pos_antena[l+2]]


               [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I
                   , N_espiras, Rotaciones[0:3,l:l+3])

               Bx[j,i]+=f[n]*B[0,0]
               By[j,i]+=f[n]*B[0,1]
               Bz[j,i]+=f[n]*B[0,2]

               l+=3

r=np.add(np.add(Bx**2,By**2),Bz**2)
MODB=np.sqrt(r) #Modulo del campo magnetico en cada punto de la
    malla

#evitar indeterminacion de 'Dipolo' cuando Pos_sensor==
    Pos_antena_n:
for i in range(nx):
    for j in range(ny):
        if MODB[j,i]>1e-5:
            MODB[j,i]=1e-5

        #Si el campo es menor que la sensibilidad no lo
            representamos:
        if MODB[j,i]<sensibilidad:
            MODB[j,i]=sensibilidad

#################################################################
"""Representacion"""
#################################################################
```

```
        fig, ax=plt.subplots(1)
        cm=plt.cm.get_cmap('YlOrRd')

        if dB==True:  #Si se pide representar en dB:
            MODBdB=20*np.log10(MODB/1e-9)
            pc=ax.scatter(Xm,Ym,c=MODBdB,norm=colors.Normalize(vmin=
                MODBdB.min(), vmax=MODBdB.max()), cmap=cm)
            plt.title('Campo magnético en Plano Z = %i , (dBnT)' %
                RepresentacionZ[0])
        else:
            pc=ax.scatter(Xm,Ym,c=MODB,norm=colors.LogNorm(vmin=MODB.min
                (), vmax=MODB.max()),cmap=cm)
            plt.title('Campo magnético en Plano Z = %i , (T)' %
                RepresentacionZ[0])

        fig.colorbar(pc,ax=ax)
        plt.show()

        #representacion areas de calculo
        for v in range (V):
            xmaxr, xminr= VOL[v][0], VOL[v][1]
            ymaxr, yminr= VOL[v][2], VOL[v][3]
            zmaxr, zminr= VOL[v][4], VOL[v][5]

            #representamos cada AREA de estudio
            xx = [xminr, xminr, xmaxr, xmaxr, xminr]
            yy = [yminr, ymaxr, ymaxr, yminr, yminr]
            color='blue'
            kwargs = {'alpha': 1, 'color': color}

            if zmaxr<RepresentacionZ[0] or RepresentacionZ[0]<zminr:
                kwargs = {'alpha': 1, 'color': color, 'linestyle':':'}
            plt.plot(xx, yy, **kwargs)

        #representacion antenas
        l=0
        for n in range(N):
            plt.scatter(Pos_antena[l],Pos_antena[l+1],color='black',
                marker='s')
            l+=3

        plt.xlabel('x(m)')
        plt.ylabel('y(m)')



    #Si nos piden un plano X=cte
    if RepresentacionX!=[]:
        ###################################################################
        """DEFINICION AREA DE CALCULO"""
```

```
#################################################################
#Creacion rejilla con puntos en los que evaluar el campo en ese
    volumen
ny, nz = int(((ymax-ymin)/((ymax-ymin)/300))+1), int(((zmax-zmin)
    /((zmax-zmin)/300))+1)

y = np.linspace(ymin, ymax, num=ny)
z = np.linspace(zmin, zmax, num=nz)

Ym,Zm=np.meshgrid(y,z)


#################################################################
"""ALGORITMO CALCULO DE CAMPO MAGNETICO"""
#################################################################
Bx,By,Bz=np.zeros((nz,ny)),np.zeros((nz,ny)), np.zeros((nz,ny))

for j in range(ny):
     for k in range(nz):

          l=0
          for n in range(N):
               Pos_sensor=[RepresentacionX[0],y[j],z[k]]
               Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                   Pos_antena[l+2]]
               [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I
                   , N_espiras, Rotaciones[0:3,l:l+3])

               Bx[k,j]+=B[0,0]
               By[k,j]+=B[0,1]
               Bz[k,j]+=B[0,2]

               l+=3

r=np.add(np.add(Bx**2,By**2),Bz**2)
MODB=np.sqrt(r)

#evitar indeterminacion de 'Dipolo' cuando Pos_sensor==
    Pos_antena_n:
for j in range(ny):
    for k in range(nz):
        if MODB[k,j]>1e-5:
            MODB[k,j]=1e-5

        if MODB[k,j]<sensibilidad:
            MODB[k,j]=sensibilidad

#################################################################
"""Representacion"""
#################################################################
```

```
        fig, ax=plt.subplots(1)
        cm=plt.cm.get_cmap('YlOrRd')

        if dB==True:
            MODBdB=20*np.log10(MODB/1e-9)
            pc=ax.scatter(Ym,Zm,c=MODBdB,norm=colors.Normalize(vmin=
                MODBdB.min(), vmax=MODBdB.max()), cmap=cm)
            plt.title('Campo magnético en Plano X = %i , (dBnT)' %
                RepresentacionX[0])
        else:
            pc=ax.scatter(Ym,Zm,c=MODB,norm=colors.LogNorm(vmin=MODB.min
                (), vmax=MODB.max()),cmap=cm)
            plt.title('Campo magnético en Plano X = %i , (T)' %
                RepresentacionX[0])

        fig.colorbar(pc,ax=ax)
        plt.show()

        #representacion areas de calculo
        for v in range (V):
            xmaxr, xminr= VOL[v][0], VOL[v][1]
            ymaxr, yminr= VOL[v][2], VOL[v][3]
            zmaxr, zminr= VOL[v][4], VOL[v][5]
            #representamos cada AREA de estudio
            yy = [yminr, yminr, ymaxr, ymaxr, yminr]
            zz = [zminr, zmaxr, zmaxr, zminr, zminr]
            color='blue'
            kwargs = {'alpha': 1, 'color': color}
            if xmaxr<RepresentacionX[0] or RepresentacionX[0]<xminr:
                kwargs = {'alpha': 1, 'color': color, 'linestyle':':'}
            plt.plot(yy, zz, **kwargs)

        #representacion antenas
        l=0
        for n in range(N):
            plt.scatter(Pos_antena[l+1],Pos_antena[l+2],color='black',
                marker='s')
            l+=3

        plt.xlabel('y(m)')
        plt.ylabel('z(m)')



    #Si nos piden representar un plano Y=cte
    if RepresentacionY!=[]:
        ###################################################################
        """DEFINICION AREA DE CALCULO"""
        ###################################################################
```

```
#Creacion rejilla con puntos en los que evaluar el campo en ese
    volumen
nx, nz = int(((xmax-xmin)/((xmax-xmin)/300))+1), int(((zmax-zmin)
    /((zmax-zmin)/300))+1)

x = np.linspace(xmin, xmax, num=nx)
z = np.linspace(zmin, zmax, num=nz)
Xm,Zm=np.meshgrid(x,z)

#################################################################
"""ALGORITMO BUSQUEDA DE PUNTOS OPTIMOS"""
#################################################################
Bx,By,Bz=np.zeros((nz,nx)),np.zeros((nz,nx)), np.zeros((nz,nx))

for i in range(nx):
     for k in range(nz):

          l=0
          for n in range(N):
               Pos_sensor=[x[i],RepresentacionY[0],z[k]]
               Pos_antena_n=[Pos_antena[l],Pos_antena[l+1],
                   Pos_antena[l+2]]
               [B,modB]=Dipolo(Pos_antena_n,Pos_sensor,semilado, I
                   , N_espiras, Rotaciones[0:3,l:l+3])

               Bx[k,i]+=B[0,0]
               By[k,i]+=B[0,1]
               Bz[k,i]+=B[0,2]

               l+=3

r=np.add(np.add(Bx**2,By**2),Bz**2)
MODB=np.sqrt(r)
#evitar indeterminacion de 'Dipolo' cuando Pos_sensor==
    Pos_antena_n:
for i in range(nx):
    for k in range(nz):
        if MODB[k,i]>1e-5:
            MODB[k,i]=1e-5

        if MODB[k,i]<sensibilidad:
            MODB[k,i]=sensibilidad
#################################################################
"""Representacion"""
#################################################################
fig, ax=plt.subplots(1)
cm=plt.cm.get_cmap('YlOrRd')

if dB==True:
    MODBdB=20*np.log10(MODB/1e-9)
```

```
            pc=ax.scatter(Xm,Zm,c=MODBdB,norm=colors.Normalize(vmin=
                MODBdB.min(), vmax=MODBdB.max()), cmap=cm)
            plt.title('Campo magnético en Plano Y = %i , (dBnT)' %
                RepresentacionY[0])
        else:
            pc=ax.scatter(Xm,Zm,c=MODB,norm=colors.LogNorm(vmin=MODB.min
                (), vmax=MODB.max()),cmap=cm)
            plt.title('Campo magnético en Plano Y = %i , (T)' %
                RepresentacionY[0])

        fig.colorbar(pc,ax=ax)
        plt.show()

        #representacion areas de calculo
        for v in range (V):
            xmaxr, xminr= VOL[v][0], VOL[v][1]
            ymaxr, yminr= VOL[v][2], VOL[v][3]
            zmaxr, zminr= VOL[v][4], VOL[v][5]
            #representamos cada AREA de estudio
            xx = [xminr, xminr, xmaxr, xmaxr, xminr]
            zz = [zminr, zmaxr, zmaxr, zminr, zminr]
            color='blue'
            kwargs = {'alpha': 1, 'color': color}
            if ymaxr<RepresentacionY[0] or RepresentacionY[0]<yminr:
                kwargs = {'alpha': 1, 'color': color, 'linestyle':':'}
            plt.plot(xx, zz, **kwargs)

        #representacion antenas
        l=0
        for n in range(N):
            plt.scatter(Pos_antena[l],Pos_antena[l+2],color='black',
                marker='s')
            l+=3

        plt.xlabel('x(m)')
        plt.ylabel('z(m)')
```

A.11 Interfaz

Código A.11 Interfaz gráfica.

```
import tkinter as tk
from tkinter import Tk, ttk, messagebox
# import matplotlib.pyplot as plt
# from mpl_toolkits.mplot3d import Axes3D
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from math import pi
```

```
from GA_Definitivo_NuevosReq import (OPTIMIZACION, SOLUCION, CRITICOS,
    RepresentacionPlano,MatRotacion, Dipolo, InterseccionVolumenes)


class myApp():
    def __init__(self,master):

        ###################################################################
        """DEFINICION FRAMES PRINCIPALES Y SUS CONFIGURACIONES INICIALES
            """
        ###################################################################
        # CONTENEDOR
        self.nb = ttk.Notebook(master)
        self.nb.pack(fill='both',expand=True,padx=10,pady=10)

        # STEP 1: Definir volumen de optimización
        self.f1 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f1,text="Define the optimization volume",font='
            Helvetica 10 bold').pack(anchor='nw') # Descripción
        self.f11 = ttk.Frame(self.f1) # Subcontenedor
        self.f11.pack(fill='both',expand=True)
        self.f111 = ttk.Frame(self.f11) # Left side
        self.f111.pack(side='left',anchor='nw')
        self.f1111 = ttk.Frame(self.f111) # Top left side
        self.f1111.pack()
        ttk.Label(self.f1111,text=u'x\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=1,padx=5) # Unicode para subscripts
        ttk.Label(self.f1111,text=u'x\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=2,padx=5)
        ttk.Label(self.f1111,text=u'y\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=3,padx=5) # Unicode para subscripts
        ttk.Label(self.f1111,text=u'y\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=4,padx=5)
        ttk.Label(self.f1111,text=u'z\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=5,padx=5) # Unicode para subscripts
        ttk.Label(self.f1111,text=u'z\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=6,padx=5)
        self.VoptLabel = [] # Inicializar vector de identificadores de
            filas
        self.VoptLabel.append(ttk.Label(self.f1111,text='1:',font='times
             9'))
        self.VoptLabel[0].grid(row=1,column=0,pady=(0,5))
        self.Vopt = [[]] # Inicializar matriz de 'entries'
        for i in range(6):
            self.Vopt[0].append(ttk.Entry(self.f1111,width=5))
            self.Vopt[0][i].grid(row=1,column=i+1,padx=5,pady=(0,5))
        self.f1112 = ttk.Frame(self.f111) # Bottom left side
        self.f1112.pack(pady=(5,0))
        ttk.Button(self.f1112,text='Update',command=self.drawVopt).pack()
             # Botón para actualizar el plot
```

```
        self.f112 = ttk.Frame(self.f11) # Right side
        self.f112.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        self.f1121 = ttk.Frame(self.f112) # Botones
        self.f1121.pack(anchor='nw',pady=(23,10))
        ttk.Button(self.f1121,text="+",width=5,command=self.addVopt).pack
            (side='left') # Añadir fila
        ttk.Button(self.f1121,text="-",width=5,command=self.delVopt).pack
            (side='left') # Eliminar fila
        self.f1122 = ttk.Frame(self.f112) # Plot
        self.f1122.pack(fill='both', expand=True)
        self.figVopt = Figure(figsize=(4, 3), dpi=100) # Figura
        self.canvasVopt = FigureCanvasTkAgg(self.figVopt, self.f1122) #
            Región de dibujo
        self.canvasVopt.draw()
        self.axVopt = self.figVopt.add_subplot(111,projection = '3d') #
            Ejes
        self.axVopt.set_title('Optimization volume')
        self.axVopt.set_xlabel('X [m]')
        self.axVopt.set_ylabel('Y [m]')
        self.axVopt.set_zlabel('Z [m]')
        self.canvasVopt.get_tk_widget().pack(fill='both', expand=True)
        self.resultsok=0

        #_____________________________________________________________
        # STEP 2: Definir volumen de posicionamiento de antenas
        self.f2 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f2,text="Define the beacons' movement volume",
            font='Helvetica 10 bold').pack(anchor='nw') # Descripción
        self.f21 = ttk.Frame(self.f2) # Subcontenedor
        self.f21.pack(fill='both',expand=True)
        self.f211 = ttk.Frame(self.f21) # Left side
        self.f211.pack(side='left',anchor='nw')
        self.f2111 = ttk.Frame(self.f211) # Top left side
        self.f2111.pack()
        ttk.Label(self.f2111,text=u'x\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=1,padx=5) # Unicode para subscripts
        ttk.Label(self.f2111,text=u'x\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=2,padx=5)
        ttk.Label(self.f2111,text=u'y\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=3,padx=5) # Unicode para subscripts
        ttk.Label(self.f2111,text=u'y\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=4,padx=5)
        ttk.Label(self.f2111,text=u'z\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=5,padx=5) # Unicode para subscripts
        ttk.Label(self.f2111,text=u'z\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=6,padx=5)
        self.VantLabel = [] # Inicializar vector de identificadores de
            filas
```

```
        self.VantLabel.append(ttk.Label(self.f2111,text='1:',font='times
            9'))
        self.VantLabel[0].grid(row=1,column=0,pady=(0,5))
        self.Vant = [[]] # Inicializar matriz de 'entries'
        for i in range(6):
            self.Vant[0].append(ttk.Entry(self.f2111,width=5))
            self.Vant[0][i].grid(row=1,column=i+1,padx=5,pady=(0,5))
        self.f2112 = ttk.Frame(self.f211) # Bottom left side
        self.f2112.pack(pady=(5,0))
        ttk.Button(self.f2112,text='Update',command=self.drawVant).pack()
            # Botón para actualizar el plot
        self.f212 = ttk.Frame(self.f21) # Right side
        self.f212.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        self.f2121 = ttk.Frame(self.f212) # Botones
        self.f2121.pack(anchor='nw',pady=(23,10))
        ttk.Button(self.f2121,text="+",width=5,command=self.addVant).pack
            (side='left') # Añadir fila
        ttk.Button(self.f2121,text="+ (same volume)",width=15,command=
            self.addSameVant).pack(side='left') # Añadir fila
        ttk.Button(self.f2121,text="-",width=5,command=self.delVant).pack
            (side='left') # Eliminar fila
        self.f2122 = ttk.Frame(self.f212) # Plot
        self.f2122.pack(fill='both', expand=True)
        self.figVant = Figure(figsize=(4, 3), dpi=100) # Figura
        self.canvasVant = FigureCanvasTkAgg(self.figVant, self.f2122) #
            Región de dibujo
        self.canvasVant.draw()
        self.axVant = self.figVant.add_subplot(111,projection = '3d') #
            Ejes
        self.axVant.set_title('Search volume')
        self.axVant.set_xlabel('X [m]')
        self.axVant.set_ylabel('Y [m]')
        self.axVant.set_zlabel('Z [m]')
        self.canvasVant.get_tk_widget().pack(fill='both', expand=True)

        #-----------------------------------------------------------------
        # STEP 3: Definir restricciones al volumen de búsqueda
        self.f3 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f3,text="Define restrictions to the beacons'
            movement volume",font='Helvetica 10 bold').pack(anchor='nw')
            # Descripción
        self.f31 = ttk.Frame(self.f3) # Subcontenedor
        self.f31.pack(fill='both',expand=True)
        self.f311 = ttk.Frame(self.f31) # Left side
        self.f311.pack(side='left',anchor='nw')
        self.f3111 = ttk.Frame(self.f311) # Top left side
        self.f3111.pack()
        ttk.Label(self.f3111,text=u'x\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=1,padx=5) # Unicode para subscripts
```

```
        ttk.Label(self.f3111,text=u'x\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=2,padx=5)
        ttk.Label(self.f3111,text=u'y\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=3,padx=5) # Unicode para subscripts
        ttk.Label(self.f3111,text=u'y\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=4,padx=5)
        ttk.Label(self.f3111,text=u'z\u2098\u1D62\u2099',font='times 14')
            .grid(row=0,column=5,padx=5) # Unicode para subscripts
        ttk.Label(self.f3111,text=u'z\u2098\u2090\u2093',font='times 14')
            .grid(row=0,column=6,padx=5)
        self.Vpro = [[]] # Inicializar matriz de 'entries'
        self.f3112 = ttk.Frame(self.f311) # Bottom left side
        self.f3112.pack(pady=(5,0))
        ttk.Button(self.f3112,text='Update',command=self.drawVpro).pack()
             # Botón para actualizar el plot
        self.f312 = ttk.Frame(self.f31) # Right side
        self.f312.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        self.f3121 = ttk.Frame(self.f312) # Botones
        self.f3121.pack(anchor='nw',pady=(23,10))
        ttk.Button(self.f3121,text="+",width=5,command=self.addVpro).pack
            (side='left') # Añadir fila
        ttk.Button(self.f3121,text="-",width=5,command=self.delVpro).pack
            (side='left') # Eliminar fila
        self.f3122 = ttk.Frame(self.f312) # Plot
        self.f3122.pack(fill='both', expand=True)
        self.figVpro = Figure(figsize=(4, 3), dpi=100) # Figura
        self.canvasVpro = FigureCanvasTkAgg(self.figVpro, self.f3122) #
            Región de dibujo
        self.canvasVpro.draw()
        self.axVpro = self.figVpro.add_subplot(111,projection = '3d') #
            Ejes
        self.axVpro.set_title('Search volume')
        self.axVpro.set_xlabel('X [m]')
        self.axVpro.set_ylabel('Y [m]')
        self.axVpro.set_zlabel('Z [m]')
        self.canvasVpro.get_tk_widget().pack(fill='both', expand=True)

        #_____________________________________________________________
        # STEP 4: Definir parámetros de las antenas
        self.f4 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f4,text="Define beacons' angular restrictions (º)
            and general parameters",font='Helvetica 10 bold').pack(anchor
            ='nw') # Descripción
        self.f41 = ttk.Frame(self.f4) # Subcontenedor
        self.f41.pack(fill='both',expand=True)
        self.f411 = ttk.Frame(self.f41) # Left side
        self.f411.pack(side='left',anchor='nw')
        self.f412 = ttk.Frame(self.f41) # Left side
        self.f412.pack()
```

```
        self.f4111 = ttk.Frame(self.f411) # Top left side
        self.f4111.pack()
        ttk.Label(self.f4111,text=u'\u2098\u1D62\u2099',font='times 14').
            grid(row=0,column=1,padx=5) # Unicode para subscripts
        ttk.Label(self.f4111,text=u'\u2098\u2090\u2093',font='times 14').
            grid(row=0,column=2,padx=5)
        ttk.Label(self.f4111,text=u'\u2098\u1D62\u2099',font='times 14').
            grid(row=0,column=3,padx=5) # Unicode para subscripts
        ttk.Label(self.f4111,text=u'\u2098\u2090\u2093',font='times 14').
            grid(row=0,column=4,padx=5)
        ttk.Label(self.f4111,text='Spires',font='times 12').grid(row=0,
            column=5,padx=5)
        ttk.Label(self.f4111,text='L (m)',font='times 12').grid(row=0,
            column=6,padx=5)
        ttk.Label(self.f4111,text='I (A)',font='times 12').grid(row=0,
            column=7,padx=5)
        ttk.Label(self.f4111,text='F',font='times 12').grid(row=0,column
            =8,padx=5)
        self.paramLabel = [] # Inicializar vector de identificadores de
            filas
        self.paramLabel.append(ttk.Label(self.f4111,text='1:',font='
            times 9'))
        self.paramLabel[0].grid(row=1,column=0,pady=(0,5))
        self.param = [[]] # Inicializar matriz de 'entries'
        for i in range(8):
                self.param[0].append(ttk.Entry(self.f4111,width=8))
                self.param[0][i].grid(row=1,column=i+1,padx=5,pady=(0,5))
                #Definimos valores iniciales
                if i==0:
                    self.param[0][i].insert(-1, '-180')
                elif i==1:
                    self.param[0][i].insert(-1, '180')
                elif i==2:
                    self.param[0][i].insert(-1, '-90')
                elif i==3:
                    self.param[0][i].insert(-1, '90')
                elif i==4:
                    self.param[0][i].insert(-1, '500')
                elif i==5:
                    self.param[0][i].insert(-1, '0.6')
                elif i==6:
                    self.param[0][i].insert(-1, '1')
                elif i==7:
                    self.param[0][i].insert(-1, '1')

        ttk.Label(self.f412,text='sensors calibration (c)',font='times 14
            ').grid(row=0,column=1,padx=50)
        self.c = ttk.Entry(self.f412,width=5)
        self.c.grid(row=1,column=1,padx=50,pady=(0,5))
        self.c.insert(-1, '1')
```

```
#_____________________________________________________________
# STEP 5: Parámetros avanzados
self.f5 = ttk.Frame(self.nb,padding=10)
ttk.Label(self.f5,text="Advanced parameters",font='Helvetica 10
    bold').pack(anchor='nw') # Descripción
self.f51 = ttk.Frame(self.f5) # Subcontenedor
self.f51.pack(fill='both',expand=True)
self.f511 = ttk.LabelFrame(self.f51,text='OPTIMIZATION: you can
    directly go to 'Other distributions in case that you dont
    want to optimize') # Left side
self.f511.pack(side='left',anchor='nw',expand=True,fill='both')
self.f5111 = ttk.Frame(self.f511) # Top left side
self.f5111.pack(fill='both',expand=True)
self.f5115 = ttk.Frame(self.f511) # bottom left side
self.f5115.pack(fill='both',expand=True)
self.f5112 = ttk.Frame(self.f511) # middle left side
self.f5112.pack(fill='both',expand=True)
self.f5114 = ttk.Frame(self.f511) # bottom left side
self.f5114.pack(fill='both',expand=True)
self.f5113 = ttk.Frame(self.f511) # bottom left side
self.f5113.pack(fill='both',expand=True)
ttk.Label(self.f5111,text='meshgrid distance(d)',font='times 12')
    .grid(row=0,column=1,padx=5)
ttk.Label(self.f5111,text='maxiter',font='times 12').grid(row=0,
    column=3,padx=50)
ttk.Label(self.f5111,text='swarmsize',font='times 12').grid(row
    =0,column=5,padx=5)
self.d = ttk.Entry(self.f5111,width=5)
self.d.grid(row=1,column=1,padx=5,pady=(0,5))
self.d.insert(-1, '0.5')
self.maxiter = ttk.Entry(self.f5111,width=5)
self.maxiter.grid(row=1,column=3,padx=50,pady=(0,5))
self.maxiter.insert(-1, '100')
self.swarmsize = ttk.Entry(self.f5111,width=5)
self.swarmsize.grid(row=1,column=5,padx=5,pady=(0,5))
self.swarmsize.insert(-1, '200')
ttk.Label(self.f5115,text="Optimal point requirements:",font='
    Helvetica 10 bold').pack(anchor='nw')
ttk.Label(self.f5112,text='n_beacons_sens',font='times 12').grid
    (row=0,column=0,padx=23)
ttk.Label(self.f5112,text='sensibility (T)',font='times 12').
    grid(row=0,column=1,padx=23)
ttk.Label(self.f5112,text='n_beacons_dif',font='times 12').grid(
    row=2,column=0,padx=23)
ttk.Label(self.f5112,text='magnitude_difference',font='times 12')
    .grid(row=2,column=1,padx=5)
self.n_antenas_sens = ttk.Entry(self.f5112,width=5)
self.n_antenas_sens.grid(row=1,column=0,padx=5,pady=(0,5))
self.n_antenas_sens.insert(-1, '3')
```

```
        self.sensibilidad = ttk.Entry(self.f5112,width=12)
        self.sensibilidad.grid(row=1,column=1,padx=10,pady=(0,5))
        self.sensibilidad.insert(-1, '2.679319e-9')
        self.n_antenas_dif = ttk.Entry(self.f5112,width=5)
        self.n_antenas_dif.grid(row=3,column=0,padx=5,pady=(0,5))
        self.n_antenas_dif.insert(-1, '3')
        self.diferencia_magnitudes = ttk.Entry(self.f5112,width=5)
        self.diferencia_magnitudes.grid(row=3,column=1,padx=5,pady=(0,5))

        self.diferencia_magnitudes.insert(-1, '0.01')
        ttk.Label(self.f5114,text="If you'd like to optimize one
            specific plane, choose from below:",font='Helvetica 10 bold').
            pack(anchor='nw')
        ttk.Label(self.f5113,text='X plane',font='times 12').grid(row=0,
            column=0,padx=30)
        ttk.Label(self.f5113,text='Y plane',font='times 12').grid(row=0,
            column=1,padx=30)
        ttk.Label(self.f5113,text='Z plane',font='times 12').grid(row=0,
            column=2,padx=30)
        self.PlanoX = ttk.Entry(self.f5113,width=5)
        self.PlanoX.grid(row=1,column=0,padx=5,pady=(0,5))
        self.PlanoY = ttk.Entry(self.f5113,width=5)
        self.PlanoY.grid(row=1,column=1,padx=63,pady=(0,5))
        self.PlanoZ = ttk.Entry(self.f5113,width=5)
        self.PlanoZ.grid(row=1,column=2,padx=5,pady=(0,5))

        #lado derecho:
        self.f512 = ttk.LabelFrame(self.f51,text='REPRESENTATION') #
            Right side
        self.f512.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        self.f5121 = ttk.Frame(self.f512) # Top rigth side
        self.f5121.pack(fill='both',expand=True)
        self.f5122 = ttk.Frame(self.f512) # middle right side
        self.f5122.pack(fill='both',expand=True)
        self.f5125 = ttk.Frame(self.f512) # bottom right side
        self.f5125.pack(fill='both',expand=True)
        self.f5126 = ttk.Frame(self.f512) # bottom right side
        self.f5126.pack(fill='both',expand=True)
        self.f5123 = ttk.Frame(self.f512) # bottom right side
        self.f5123.pack(fill='both',expand=True)
        self.f5124 = ttk.Frame(self.f512) # bottom right side
        self.f5124.pack(fill='both',expand=True)
        self.Represent_value = tk.IntVar() #boton para elegir si
            representar solucion
        self.Represent_value.set(1)
        self.Represent=ttk.Checkbutton(self.f5121,text='Calculate and
            represent Solution',variable=self.Represent_value,onvalue=1,
            offvalue=0).pack()
```

```
        self.Represent_Critics_value = tk.IntVar() #boton para elegir si
            representar puntos criticos
        self.Represent_Critics_value.set(1)
        self.Represent_Critics=ttk.Checkbutton(self.f5121,text='
            Represent overexposure points',variable=self.
            Represent_Critics_value,onvalue=1,offvalue=0).pack()
        ttk.Label(self.f5122,text='meshgrid distance(ds)',font='times 12'
            ).grid(row=0,column=0,padx=50)
        ttk.Label(self.f5122,text='Tbeacon',font='times 12').grid(row=0,
            column=1,padx=50)
        self.ds = ttk.Entry(self.f5122,width=5)
        self.ds.grid(row=1,column=0,padx=5,pady=(0,5))
        self.ds.insert(-1, '0.5')
        self.Tant = ttk.Entry(self.f5122,width=5)
        self.Tant.grid(row=1,column=1,padx=5,pady=(0,5))
        self.Tant.insert(-1, '1')
        ttk.Label(self.f5126,text="If you'd like to represent the
            magnetic field in any plane:",font='Helvetica 10 bold').pack(
            anchor='nw')
        ttk.Label(self.f5123,text='X plane field',font='times 12').grid(
            row=0,column=1,padx=5)
        ttk.Label(self.f5123,text='Y plane field',font='times 12').grid(
            row=0,column=3,padx=63)
        ttk.Label(self.f5123,text='Z plane field',font='times 12').grid(
            row=0,column=5,padx=5)
        self.CampoX = ttk.Entry(self.f5123,width=5)
        self.CampoX.grid(row=1,column=1,padx=5,pady=(0,5))
        self.CampoY = ttk.Entry(self.f5123,width=5)
        self.CampoY.grid(row=1,column=3,padx=63,pady=(0,5))
        self.CampoZ = ttk.Entry(self.f5123,width=5)
        self.CampoZ.grid(row=1,column=5,padx=5,pady=(0,5))
        ttk.Label(self.f5125,text='Maximun exposure (T)',font='times 12')
            .grid(row=0,column=1,padx=5)
        ttk.Label(self.f5125,text='meshgrid distance(d_exp)',font='times
             12').grid(row=0,column=3,padx=30)
        ttk.Label(self.f5125,text='dc',font='times 12').grid(row=0,
            column=5,padx=20)
        self.Exposicion = ttk.Entry(self.f5125,width=5)
        self.Exposicion.grid(row=1,column=1,padx=5,pady=(0,5))
        self.Exposicion.insert(-1, '1e-4')
        self.d_exp = ttk.Entry(self.f5125,width=5)
        self.d_exp.grid(row=1,column=3,padx=30,pady=(0,5))
        self.d_exp.insert(-1, '0.1')
        self.dc = ttk.Entry(self.f5125,width=5)
        self.dc.grid(row=1,column=5,padx=5,pady=(0,5))
        self.dc.insert(-1, '1')
        self.dB_value = tk.BooleanVar()
        self.dB_value.set(True)
        self.dB=ttk.Checkbutton(self.f5124,text='Represent plane X,Y or
            Z in decibels',variable=self.dB_value,onvalue=True).pack()
```

```
#_____________________________________________________________
# STEP 6: Resultados
self.f6 = ttk.Frame(self.nb,padding=10)
ttk.Label(self.f6,text="RESULTS",font='Helvetica 10 bold').pack(
    anchor='nw') # Descripción
self.f61 = ttk.Frame(self.f6) # Subcontenedor
self.f61.pack(fill='both',expand=True)
self.f611 = ttk.Frame(self.f61) # Left side
self.f611.pack(side='left',anchor='nw')
ttk.Label(self.f611,text="Beacons' optimal positions (m) and
    angles (º)",font='Helvetica 10 bold').pack(anchor='nw')
self.f6111 = ttk.Frame(self.f611) # Top left side
self.f6111.pack()
ttk.Label(self.f6111,text=u'x',font='times 14').grid(row=0,
    column=1,padx=5) # Unicode para subscripts
ttk.Label(self.f6111,text=u'y',font='times 14').grid(row=0,
    column=2,padx=5)
ttk.Label(self.f6111,text=u'z',font='times 14').grid(row=0,
    column=3,padx=5) # Unicode para subscripts
ttk.Label(self.f6111,text=u'',font='times 14').grid(row=0,column
    =4,padx=5)
ttk.Label(self.f6111,text=u'',font='times 12').grid(row=0,column
    =5,padx=5)
self.resultsLabel = [] # Inicializar vector de identificadores
    de filas
self.resultsLabel.append(ttk.Label(self.f6111,text='1:',font='
    times 9'))
self.resultsLabel[0].grid(row=1,column=0,pady=(0,5))
self.results = [[]] # Inicializar matriz de 'entries'
self.f612 = ttk.Frame(self.f61) # right side
self.f612.pack()
self.f6121 = ttk.Frame(self.f612)
self.f6121.pack()
self.f6122 = ttk.Frame(self.f612)
self.f6122.pack()
self.f6123 = ttk.Frame(self.f612)
self.f6123.pack()
self.f6124 = ttk.Frame(self.f612)
self.f6124.pack()
ttk.Label(self.f6121,text="Optimization time (s):",font='
    Helvetica 10 bold').pack(side='left',anchor='nw',expand=True,
    fill='both')
self.timeText = tk.StringVar()
self.optime=ttk.Label(self.f6121,textvariable=self.timeText,font
    ='Helvetica 10').pack(side='right',anchor='nw',expand=True,
    fill='both')
self.PuntosTotalesText1 = tk.StringVar()
self.PuntosTotalesText1.set('')
self.PuntosTotalesText2 = tk.StringVar()
```

```
        self.PuntosTotalesText2.set('')
        ttk.Label(self.f6122,textvariable=self.PuntosTotalesText1,font='
            Helvetica 10 bold').pack(side='left',anchor='nw',expand=True,
            fill='both')
        ttk.Label(self.f6122,textvariable=self.PuntosTotalesText2,font='
            Helvetica 10').pack(side='right',anchor='nw',expand=True,fill
            ='both')
        self.PuntosOptimosText1 = tk.StringVar()
        self.PuntosOptimosText1.set('')
        self.PuntosOptimosText2 = tk.StringVar()
        self.PuntosOptimosText2.set('')
        ttk.Label(self.f6123,textvariable=self.PuntosOptimosText1,font='
            Helvetica 10 bold').pack(side='left',anchor='nw',expand=True,
            fill='both')
        ttk.Label(self.f6123,textvariable=self.PuntosOptimosText2,font='
            Helvetica 10').pack(side='right',anchor='nw',expand=True,fill
            ='both')
        self.PuntosOptText1 = tk.StringVar()
        self.PuntosOptText1.set('')
        self.PuntosOptText2 = tk.StringVar()
        self.PuntosOptText2.set('')
        ttk.Label(self.f6124,textvariable=self.PuntosOptText1,font='
            Helvetica 10 bold').pack(side='left',anchor='nw',expand=True,
            fill='both')
        ttk.Label(self.f6124,textvariable=self.PuntosOptText2,font='
            Helvetica 10').pack(side='right',anchor='nw',expand=True,fill
            ='both')

        #____________________________________________________________
        # STEP 7: Definir parámetros de distintas distribuciones de
            antenas:
        self.f7 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f7,text="Try different beacons distributions",
            font='Helvetica 10 bold').pack(anchor='nw') # Descripción
        self.f71 = ttk.Frame(self.f7) # Subcontenedor
        self.f71.pack(fill='both',expand=True)
        self.f711 = ttk.Frame(self.f71) # Left side
        self.f711.pack(side='left',anchor='nw')
        self.f7111 = ttk.Frame(self.f711) # Top left side
        self.f7111.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        ttk.Label(self.f7111,text=u'x',font='times 14').grid(row=0,
            column=1,padx=5) # Unicode para subscripts
        ttk.Label(self.f7111,text=u'y',font='times 14').grid(row=0,
            column=2,padx=5)
        ttk.Label(self.f7111,text=u'z',font='times 14').grid(row=0,
            column=3,padx=5) # Unicode para subscripts
        ttk.Label(self.f7111,text=u'',font='times 14').grid(row=0,column
            =4,padx=5)
```

```
        ttk.Label(self.f7111,text=u'',font='times 12').grid(row=0,column
            =5,padx=5)
        self.TryparamLabel = [] # Inicializar vector de identificadores
            de filas
        self.TryparamLabel.append(ttk.Label(self.f7111,text='1:',font='
            times 9'))
        self.TryparamLabel[0].grid(row=1,column=0,pady=(0,5))
        self.Tryparam = [[]] # Inicializar matriz de 'entries'
        for i in range(5):
                self.Tryparam[0].append(ttk.Entry(self.f7111,width=8))
                self.Tryparam[0][i].grid(row=1,column=i+1,padx=5,pady
                    =(0,5))
        self.f7112 = ttk.Frame(self.f711) # Top right side
        self.f7112.pack(side='left',anchor='nw',expand=True,fill='both',
            padx=(20,0))
        ttk.Label(self.f7112,text="If you'd like to represent the
            magnetic field in any plane:",font='Helvetica 10 bold').pack(
            anchor='nw')
        self.f71121 = ttk.Frame(self.f7112) # Top right side
        self.f71121.pack()
        ttk.Label(self.f71121,text='X plane field',font='times 12').grid
            (row=0,column=1,padx=5)
        ttk.Label(self.f71121,text='Y plane field',font='times 12').grid
            (row=0,column=3,padx=63)
        ttk.Label(self.f71121,text='Z plane field',font='times 12').grid
            (row=0,column=5,padx=5)
        self.TryCampoX = ttk.Entry(self.f71121,width=5)
        self.TryCampoX.grid(row=1,column=1,padx=5,pady=(0,5))
        self.TryCampoY = ttk.Entry(self.f71121,width=5)
        self.TryCampoY.grid(row=1,column=3,padx=63,pady=(0,5))
        self.TryCampoZ = ttk.Entry(self.f71121,width=5)
        self.TryCampoZ.grid(row=1,column=5,padx=5,pady=(0,5))

        #-----------------------------------------------------------------
        # STEP 8: solucion de prueba:
        self.f8 = ttk.Frame(self.nb,padding=10)
        ttk.Label(self.f8,text="RESULTS",font='Helvetica 10 bold').pack(
            anchor='nw') # Descripción
        self.f81 = ttk.Frame(self.f8) # Subcontenedor
        self.f81.pack(fill='both',expand=True)
        self.f811 = ttk.Frame(self.f81) # Subcontenedor
        self.f811.pack(fill='both',expand=True)
        self.f812 = ttk.Frame(self.f81) # Subcontenedor
        self.f812.pack(fill='both',expand=True)
        self.f813 = ttk.Frame(self.f81) # Subcontenedor
        self.f813.pack(fill='both',expand=True)
        self.TryPuntosTotalesText1 = tk.StringVar()
        self.TryPuntosTotalesText1.set('Number of points studied:')
        self.TryPuntosTotalesText2 = tk.StringVar()
        self.TryPuntosTotalesText2.set('')
```

```
        ttk.Label(self.f811,textvariable=self.TryPuntosTotalesText1,font
            ='Helvetica 10 bold').pack(side='left',anchor='nw',expand=
            True,fill='both')
        ttk.Label(self.f811,textvariable=self.TryPuntosTotalesText2,font
            ='Helvetica 10').pack(side='right',anchor='nw',expand=True,
            fill='both')
        self.TryPuntosOptimosText1 = tk.StringVar()
        self.TryPuntosOptimosText1.set('Number of optimal points:')
        self.TryPuntosOptimosText2 = tk.StringVar()
        self.TryPuntosOptimosText2.set('')
        ttk.Label(self.f812,textvariable=self.TryPuntosOptimosText1,font
            ='Helvetica 10 bold').pack(side='left',anchor='nw',expand=
            True,fill='both')
        ttk.Label(self.f812,textvariable=self.TryPuntosOptimosText2,font
            ='Helvetica 10').pack(side='right',anchor='nw',expand=True,
            fill='both')
        self.TryPuntosOptText1 = tk.StringVar()
        self.TryPuntosOptText1.set('Optimal points porcentaje (%):')
        self.TryPuntosOptText2 = tk.StringVar()
        self.TryPuntosOptText2.set('')
        ttk.Label(self.f813,textvariable=self.TryPuntosOptText1,font='
            Helvetica 10 bold').pack(side='left',anchor='nw',expand=True,
            fill='both')
        ttk.Label(self.f813,textvariable=self.TryPuntosOptText2,font='
            Helvetica 10').pack(side='right',anchor='nw',expand=True,fill
            ='both')


        #_____________________________________________________________
        # ANADIR FRAMES AL CONTENEDOR Y OCULTAR:
        self.nb.add(self.f1,text='Optimization Volume')
        self.nb.add(self.f2,text='Beacons Volumes',state='disabled')
        self.nb.add(self.f3,text='Positioning Restrictions',state='
            disabled')
        self.nb.add(self.f4,text='Beacons‘ Parameters',state='disabled')
        self.nb.add(self.f5,text='Optimization',state='disabled')
        self.nb.add(self.f6,text='Results',state='disabled')
        self.nb.add(self.f7,text='Other distributions',state='disabled')
        self.nb.add(self.f8,text='Results',state='disabled')

        #_____________________________________________________________
        # CAJA DE NAVEGACION: NEXT, BACK, HELP
        self.navBox = ttk.Frame(master,relief='flat')
        self.backBtn = ttk.Button(self.navBox,text="Back",padding=5,
            state=['disabled'],command=self.backFcn)
        self.backBtn.pack(side='left',padx=(0,5))
        self.nextBtn = ttk.Button(self.navBox,text="Next",padding=5,
            command=self.nextFcn)
        self.nextBtn.pack(side='left',padx=(5,0))
```

```
        self.infoBtn = ttk.Button(self.navBox,text="Help",padding=5,
            command=self.infoFcn)
        self.infoBtn.pack(side='left',padx=(5,0))
        self.navBox.pack(pady=(0,10))

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


    ########################################################################
    """FUNCIONES"""
    ########################################################################

    #FUNCION NEXT: indica los pasos a seguir al pasar de un frame a otro
         según la pestaña (index) en la que estemos: carga de datos,
        configuración de pestañas...
    def nextFcn(self):
        index = self.nb.index(self.nb.select())
        if index == 0:
            if not self.drawVopt(): # Actualizar plot y realizar
                comprobaciones
                return False # Abortar

            #crear VOL: variable en la que guardamos los volumenes a
                estudiar
            n = len(self.Vopt) # Número de volúmenes
            self.VOL = [] # Inicializar VOL (valores numéricos, no
                entries)
            for i in range(n):
                xmin = float(self.Vopt[i][0].get())
                xmax = float(self.Vopt[i][1].get())
                ymin = float(self.Vopt[i][2].get())
                ymax = float(self.Vopt[i][3].get())
                zmin = float(self.Vopt[i][4].get())
                zmax = float(self.Vopt[i][5].get())
                self.VOL.append([xmax, xmin, ymax, ymin, zmax, zmin])
            if InterseccionVolumenes(self.VOL): # COMPROBAR OVERLAPPING Y
                 LANZAR UN WARNING
                go = messagebox.askyesno('Overlapping detected','Some
                    regions overlap. This could lead to inaccurate results
                    . Do you want to continue?')
                if not go:
                    return False # Abortar
            self.backBtn.state(['!disabled'])

        if index == 1:
            if not self.drawVant(): # Actualizar plot y realizar
                comprobaciones
                return False # Abortar
```

```
            #crear ub,lb: valores maximos y minimos donde se puede
                colocar cada antena:
            n = len(self.Vant) # Número de volúmenes
            self.lb1 = [] # Inicializar lb (valores numéricos, no entries
                )
            self.ub1 = [] # Inicializar ub (valores numéricos, no entries
                )
            for i in range(n):
                xmin = float(self.Vant[i][0].get())
                xmax = float(self.Vant[i][1].get())
                ymin = float(self.Vant[i][2].get())
                ymax = float(self.Vant[i][3].get())
                zmin = float(self.Vant[i][4].get())
                zmax = float(self.Vant[i][5].get())

                #guardar datos (lb,ub)
                self.lb1=self.lb1+[xmin,ymin,zmin]
                self.ub1=self.ub1+[xmax,ymax,zmax]

        if index == 2:
            #crear VOLP: volumenes prohibidos
            if not self.drawVpro(): # Actualizar plot y realizar
                comprobaciones
                return False # Abortar
            self.VOLP = [] # Inicializar VOLP (valores numéricos, no
                entries)
            if self.Vpro!=[[]]:
                n = len(self.Vpro) # Número de volúmenes

                for i in range(n):
                    xmin = float(self.Vpro[i][0].get())
                    xmax = float(self.Vpro[i][1].get())
                    ymin = float(self.Vpro[i][2].get())
                    ymax = float(self.Vpro[i][3].get())
                    zmin = float(self.Vpro[i][4].get())
                    zmax = float(self.Vpro[i][5].get())
                    #guardar VOLP
                    self.VOLP.append([xmax,xmin,ymax,ymin,zmax,zmin])

                if InterseccionVolumenes(self.VOLP):
                    go = messagebox.askyesno('Overlapping detected','Some
                        regions overlap. This could lead to inaccurate
                        results. Do you want to continue?')
                    if not go:
                        return False # Abortar

        if index == 3:
            n = len(self.Vant) # Número de antenas
            #comprobar que no haya datos vacíos
            for i in range(n):
```

```
            try:
                thetamin = float(self.param[i][0].get())
                thetamax = float(self.param[i][1].get())
                phimin = float(self.param[i][2].get())
                phimax = float(self.param[i][3].get())
                self.N_espiras = float(self.param[i][4].get())
                self.semilado = float(self.param[i][5].get())/2
                self.I = float(self.param[i][6].get())
                self.F = float(self.param[i][7].get())
            except:
                messagebox.showerror('Impossible to update',f'Row {i
                    +1} contains no numerical inputs')
                return False
            if thetamax < thetamin or phimax < phimin :
                messagebox.showerror('Impossible to update',f'In row {
                    i+1} an upper limit (angle) is inferior to its
                    lower limit')
                return False
        try:
            c=float(self.c.get())
        except:
            messagebox.showerror('Impossible to update','Calibration
                (c) contains no numerical input')
            return False

        #guardar datos:
        self.nextBtn['text'] = 'Calculate!'
        self.N_espiras=[]
        self.I=[]
        self.semilado=[]
        self.F=[]
        self.lb=[]
        self.ub=[]
        thetamin, thetamax,phimin,phimax=[],[],[],[]
        for i in range(n):
            thetamin = thetamin+[float(self.param[i][0].get())*pi
                /180]
            thetamax = thetamax+[float(self.param[i][1].get())*pi
                /180]
            phimin = phimin+[float(self.param[i][2].get())*pi/180]
            phimax = phimax+[float(self.param[i][3].get())*pi/180]
            self.N_espiras = self.N_espiras+[float(self.param[i][4].
                get())]
            self.semilado = self.semilado+[float(self.param[i][5].get
                ())/2]
            self.I = self.I+[float(self.param[i][6].get())]
            self.F = self.F+[float(self.param[i][7].get())]

        self.lb=self.lb1+thetamin+phimin
        self.ub=self.ub1+thetamax+phimax
```

```
            self.C=float(self.c.get())

            self.nb.tab(6,state='normal')

        if index == 4:
            self.nextBtn['text'] = 'Next'

            #comprobar y guardar datos de entries:
            try:
                self.d1=float(self.d.get())
                self.maxiter1=int(self.maxiter.get())
                self.swarmsize1=int(self.swarmsize.get())
                self.n_antenas_sens1=float(self.n_antenas_sens.get())
                self.sensibilidad1=float(self.sensibilidad.get())
                self.n_antenas_dif1=float(self.n_antenas_dif.get())
                self.diferencia_magnitudes1=float(self.
                    diferencia_magnitudes.get())
            except:
                messagebox.showerror('Impossible to update','Optimization
                     parameter contains no numerical input')
                return False

            self.d1=float(self.d.get())
            self.maxiter1=int(self.maxiter.get())
            self.swarmsize1=int(self.swarmsize.get())
            self.n_antenas_sens1=float(self.n_antenas_sens.get())
            self.sensibilidad1=float(self.sensibilidad.get())
            self.n_antenas_dif1=float(self.n_antenas_dif.get())
            self.diferencia_magnitudes1=float(self.diferencia_magnitudes.
                get())

            #guardar valores de checkbuttoms:
            self.dB1=self.dB_value.get()
            self.Represent1=self.Represent_value.get()
            self.Represent_critics1=self.Represent_Critics_value.get()

            if self.Represent1==1: #si piden representar, comprobamos que
                 los valores no estén vacíos
                try:
                    self.ds1=float(self.ds.get())
                    self.Tant1=float(self.Tant.get())
                except:
                    messagebox.showerror('Impossible to update','
                        Representation parameter contains no numerical
                        input')
                    return False

            if self.Represent_critics1==1: #si piden representar puntos
                críticos, comprobamos que los valores no estén vacíos
                try:
```

```
            self.dc1=float(self.dc.get())
            self.d_exp1=float(self.d_exp.get())
            self.Exposicion1=float(self.Exposicion.get())
        except:
            messagebox.showerror('Impossible to update','
                Overexposure representation parameters contains no
                 numerical input')
            return False


    self.ds1=float(self.ds.get())
    self.Tant1=float(self.Tant.get())
    self.Exposicion1=float(self.Exposicion.get())
    self.d_exp1=float(self.d_exp.get())
    self.dc1=float(self.dc.get())

    if self.PlanoX.get()!='':
        self.PlanoX1=[float(self.PlanoX.get())]
    else:
        self.PlanoX1=[]
    if self.PlanoY.get()!='':
        self.PlanoY1=[float(self.PlanoY.get())]
    else:
        self.PlanoY1=[]
    if self.PlanoZ.get()!='':
        self.PlanoZ1=[float(self.PlanoZ.get())]
    else:
        self.PlanoZ1=[]

    if self.CampoX.get()!='':
        self.CampoX1=[float(self.CampoX.get())]
    else:
        self.CampoX1=[]
    if self.CampoY.get()!='':
        self.CampoY1=[float(self.CampoY.get())]
    else:
        self.CampoY1=[]
    if self.CampoZ.get()!='':
        self.CampoZ1=[float(self.CampoZ.get())]
    else:
        self.CampoZ1=[]


    #CALCULO DE RESULTADOS: OPTIMIZACION Y REPRESENTACION:
    [self.Pos_antena,self.theta,self.phi,self.time]=Optimizacion(
        self.semilado,self.N_espiras,self.I,self.C,self.F,self.
        sensibilidad1,self.d1,len(self.Vant),self.Tant1,self.
        maxiter1,self.swarmsize1, self.PlanoZ1, self.PlanoX1,
        self.PlanoY1, self.VOL, self.VOLP, self.lb, self.ub, self.
```

```
            Exposicion1, self.n_antenas_sens1,self.n_antenas_dif1,
            self.diferencia_magnitudes1)

        self.timeText.set(f'{"{:.3f}".format(self.time)}')

        VP=len(self.VOLP)
        for v in range(VP):
            xmaxp, xminp= self.VOLP[v][0], self.VOLP[v][1]
            ymaxp, yminp= self.VOLP[v][2], self.VOLP[v][3]
            zmaxp, zminp= self.VOLP[v][4], self.VOLP[v][5]

            l=0
            N = len(self.Vant)
            for n in range(N):
                if (xminp-0.1<=self.Pos_antena[l]<=xmaxp+0.1 and
                    yminp-0.1<=self.Pos_antena[l+1]<=ymaxp+0.1
                    and zminp-0.1<=self.Pos_antena[l+2]<=zmaxp+0.1)
                        :
                    messagebox.showerror('Error','Could not find a
                         solution: please, try reducing the number
                        of beacons, increasing swarmsize or
                        changing beacons positioning restrictions')
                    return False
                l+=3


        #representacion de la solucion
        if self.Represent1 == 1:
            [self.Puntos_Totales,self.Puntos_Optimos,self.
                Porcentaje_Puntos_Optimos]=Solucion(self.Pos_antena,
                self.theta,self.phi,self.semilado,self.N_espiras,self.
                I,self.C,self.F,self.sensibilidad1,len(self.Vant),self
                .ds1,self.Tant1,self.PlanoZ1, self.PlanoX1, self.
                PlanoY1, self.VOL, self.VOLP,self.n_antenas_sens1,self
                .n_antenas_dif1,self.diferencia_magnitudes1)

            #Si hemos hecho optimizacion 2D, representamos el plano
                optimizado:
            RepresentacionPlano(self.PlanoZ1,self.PlanoX1,self.
                PlanoY1,self.Pos_antena,self.theta, self.phi,self.
                semilado, self.I,self.N_espiras,self.VOL, len(self.
                Vant), self.C,self.F, self.sensibilidad1,self.dB1)

            #En caso de querer ver el campo en un plano concreto
            RepresentacionPlano(self.CampoZ1,self.CampoX1,self.
                CampoY1,self.Pos_antena,self.theta, self.phi,self.
                semilado, self.I, self.N_espiras,self.VOL, len(self.
                Vant), self.C,self.F, self.sensibilidad1,self.dB1)

            self.PuntosTotalesText1.set('Number of points studied:')
```

```
            self.PuntosTotalesText2.set(f'{self.Puntos_Totales}')

            self.PuntosOptimosText1.set('Number of optimal points')
            self.PuntosOptimosText2.set(f'{self.Puntos_Optimos}')

            self.PuntosOptText1.set('Optimal points porcentaje (%):')
            self.PuntosOptText2.set(f'{"{:.3f}".format(self.
                Porcentaje_Puntos_Optimos)}')

        else:

            self.PuntosTotalesText1.set('')
            self.PuntosTotalesText2.set('')

            self.PuntosOptimosText1.set('')
            self.PuntosOptimosText2.set('')

            self.PuntosOptText1.set('')
            self.PuntosOptText2.set('')

        #representacion de puntos criticos
        if self.Represent_critics1 == 1:
            Criticos(self.Pos_antena,self.theta,self.phi,self.
                semilado,self.N_espiras,self.I,len(self.Vant),self.
                d_exp1,self.dc1,self.Tant1,self.VOL, self.Exposicion1)

        #configuramos pestaña para resultados de antenas: se añaden
            directamente las filas que hagan falta según el número de
             antenas que se han definido (nº de antenas==nº volumenes
            de antenas)
        n = len(self.Vant) # Número de antenas
        m = len(self.results) # Número de filas de resultados de
            antenas

        #primera antena:
        for i in range(3):
            self.results[0].append(ttk.Label(self.f6111,text=f'{"{:.3
                f}".format(self.Pos_antena[i])}',font='times 9'))
            self.results[0][i].grid(row=1,column=i+1,padx=10,pady
                =(0,5))

        self.results[0].append(ttk.Label(self.f6111,text=f'{"{:.3f}".
            format(self.theta[0]*180/pi)}',font='times 9'))
        self.results[0][3].grid(row=1,column=4,padx=10,pady=(0,5))
        self.results[0].append(ttk.Label(self.f6111,text=f'{"{:.3f}".
            format(self.phi[0]*180/pi)}',font='times 9'))
        self.results[0][4].grid(row=1,column=5,padx=10,pady=(0,5))

        #resto de antenas:
        l=3
```

```
            while n > m:
                self.resultsLabel.append(ttk.Label(self.f6111,text=f'{m
                    +1}:',font='times 9'))
                self.resultsLabel[m].grid(row=m+1,column=0,pady=(0,5))
                self.results.append([])

                for i in range(3):
                    self.results[m].append(ttk.Label(self.f6111,text=f'
                        {"{:.3f}".format(self.Pos_antena[l+i])}',font='
                        times 9'))
                    self.results[m][i].grid(row=m+1,column=i+1,padx=10,
                        pady=(0,5))

                self.results[m].append(ttk.Label(self.f6111,text=f'{"{:.3
                    f}".format(self.theta[m]*180/pi)}',font='times 9'))
                self.results[m][3].grid(row=m+1,column=4,padx=10,pady
                    =(0,5))
                self.results[m].append(ttk.Label(self.f6111,text=f'{"{:.3
                    f}".format(self.phi[m]*180/pi)}',font='times 9'))
                self.results[m][4].grid(row=m+1,column=5,padx=10,pady
                    =(0,5))
                m = len(self.results) # Número de filas de parámetros de
                    antenas
                l+=3
            while n < m:
                self.resultsLabel[m-1].destroy()
                self.resultsLabel.pop(m-1)
                for i in range(5):
                    self.results[m-1][i].destroy()
                self.results.pop(m-1)
                m = len(self.results) # Número de filas de parámetros de
                    antenas

            self.resultsok=1 #variable para saber si hemos optimizado
            self.infoBtn.state(['disabled'])


        if index == 5:
            self.infoBtn.state(['!disabled'])
            self.nextBtn['text'] = 'Calculate!'

        if index == 6:
            n = len(self.Vant) # Número de antenas
            #comprobar que no haya datos vacíos
            for i in range(n):
                try:
                    float(self.Tryparam[i][0].get())
                    float(self.Tryparam[i][1].get())
                    float(self.Tryparam[i][2].get())
                    float(self.Tryparam[i][3].get())
```

```
                float(self.Tryparam[i][4].get())
            except:
                messagebox.showerror('Impossible to calculate',f'Row {
                    i+1} contains no numerical inputs')
                return False

        if self.resultsok != 1: #si hemos optimizado
           try:
                self.n_antenas_sens1=float(self.n_antenas_sens.get())
                self.sensibilidad1=float(self.sensibilidad.get())
                self.diferencia_magnitudes1=float(self.
                    diferencia_magnitudes.get())
                self.n_antenas_dif1=float(self.n_antenas_dif.get())
                self.ds1=float(self.ds.get())
                self.Tant1=float(self.Tant.get())
                self.dc1=float(self.dc.get())
                self.d_exp1=float(self.d_exp.get())
                self.Exposicion1=float(self.Exposicion.get())
           except:
                messagebox.showerror('Impossible to caculate','Please,
                     fill out 'Representation and 'Optimal point
                    requirements from 'Optimization frame')
                return False

        self.n_antenas_sens1=float(self.n_antenas_sens.get())
        self.sensibilidad1=float(self.sensibilidad.get())
        self.diferencia_magnitudes1=float(self.diferencia_magnitudes.
            get())
        self.n_antenas_dif1=float(self.n_antenas_dif.get())
        self.ds1=float(self.ds.get())
        self.Tant1=float(self.Tant.get())
        self.dc1=float(self.dc.get())
        self.d_exp1=float(self.d_exp.get())
        self.Exposicion1=float(self.Exposicion.get())
        if self.PlanoX.get()!='':
            self.PlanoX1=[float(self.PlanoX.get())]
        else:
            self.PlanoX1=[]
        if self.PlanoY.get()!='':
            self.PlanoY1=[float(self.PlanoY.get())]
        else:
            self.PlanoY1=[]
        if self.PlanoZ.get()!='':
            self.PlanoZ1=[float(self.PlanoZ.get())]
        else:
            self.PlanoZ1=[]

        #crear self.Pos_antena, self.theta y self.phi:
        n = len(self.Vant) # Número de antenas
        self.Pos_antena = []
```

```
            self.theta = []
            self.phi = []
            for i in range(n):
                x = float(self.Tryparam[i][0].get())
                y = float(self.Tryparam[i][1].get())
                z = float(self.Tryparam[i][2].get())
                theta = float(self.Tryparam[i][3].get())*pi/180
                phi = float(self.Tryparam[i][4].get())*pi/180

                self.Pos_antena = self.Pos_antena + [x,y,z]
                self.theta=self.theta+[theta]
                self.phi=self.phi+[phi]

            [self.Puntos_Totales,self.Puntos_Optimos,self.
                Porcentaje_Puntos_Optimos]=Solucion(self.Pos_antena,self.
                theta,self.phi,self.semilado,self.N_espiras,self.I,self.C,
                self.F,self.sensibilidad1,len(self.Vant),self.ds1,self.
                Tant1,self.PlanoZ1, self.PlanoX1, self.PlanoY1, self.VOL,
                 self.VOLP,self.n_antenas_sens1,self.n_antenas_dif1,self.
                diferencia_magnitudes1)

            self.TryPuntosTotalesText2.set(f'{self.Puntos_Totales}')
            self.TryPuntosOptimosText2.set(f'{self.Puntos_Optimos}')
            self.TryPuntosOptText2.set(f'{"{:.3f}".format(self.
                Porcentaje_Puntos_Optimos)}')

            self.nextBtn.state(['disabled'])
            self.infoBtn.state(['disabled'])

            if self.TryCampoX.get()!='':
                self.TryCampoX1=[float(self.TryCampoX.get())]
            else:
                self.TryCampoX1=[]
            if self.TryCampoY.get()!='':
                self.TryCampoY1=[float(self.TryCampoY.get())]
            else:
                self.TryCampoY1=[]
            if self.TryCampoZ.get()!='':
                self.TryCampoZ1=[float(self.TryCampoZ.get())]
            else:
                self.TryCampoZ1=[]

            self.dB1=self.dB_value.get()
            RepresentacionPlano(self.TryCampoZ1,self.TryCampoX1,self.
                TryCampoY1,self.Pos_antena,self.theta, self.phi,self.
                semilado, self.I, self.N_espiras,self.VOL, len(self.Vant),
                 self.C,self.F, self.sensibilidad1,self.dB1)
            self.nb.tab(4,state='disabled')

        self.nb.tab(index+1,state='normal')
```

```
        self.nb.select(index+1)
        self.nb.tab(index,state='disabled')


    #_____________________________________________________________________
    #FUNCION BACK: indica los pasos a seguir al retroceder una pestaña:
        eliminar parámetros, reconfigurar pestañas...
    def backFcn(self):

        index = self.nb.index(self.nb.select())
        if index == 1:
            self.backBtn.state(['disabled'])

        if index == 2:
            self.nb.tab(4,state='disabled')
        if index == 4:
            self.nextBtn['text'] = 'Next'
            self.nb.tab(6,state='disabled')

        if index == 5:
            self.infoBtn.state(['!disabled'])
            self.nextBtn['text'] = 'Calculate!'
            self.resultsLabel = [] # Inicializar vector de
                identificadores de filas
            self.resultsLabel.append(ttk.Label(self.f6111,text='1:',font=
                'times 9'))
            self.resultsLabel[0].grid(row=1,column=0,pady=(0,5))
            self.results = [[]] # Inicializar matriz de 'entries'

        if index == 6:
            if self.resultsok != 1:
                messagebox.showerror('Cant go to optimization results','
                    You have not optimized yet, but you can go directly to
                     'Optimization ')
                return False
            self.nextBtn['text'] = 'Next'
            self.infoBtn.state(['disabled'])

        if index == 7:
            self.nextBtn.state(['!disabled'])
            self.infoBtn.state(['!disabled'])
            self.nb.tab(4,state='normal')

        self.nb.tab(index-1,state='normal')
        self.nb.select(index-1)
        self.nb.tab(index,state='disabled')

        if index == 6:
            self.nb.tab(6,state='normal')
```

```
    #_____________________________________________________________________
    #FUNCION HELO: desplegable con informacion de cada pestaña
    def infoFcn(self):
        index = self.nb.index(self.nb.select())
        if index == 0:
            messagebox.showinfo('Help','Here you can define the volumes
                where magnetic field must be optimized')
        if index == 1:
            messagebox.showinfo('Help','Here you can define the volumes
                in which each beacon can be positioned. You must define
                one volumen for each beacon')
        if index == 2:
            messagebox.showinfo('Help','Here you can define the volumes
                in which beacons can not be positioned')
        if index == 3:
            messagebox.showinfo('Help','Define angular movement
                restrictions for each beacon and its general parameters:
                number of spires, length (L), intensity (I) and frecuency
                 factor (F)')
        if index == 4:
            messagebox.showinfo('Help',message=['n_beacons_sens: minimun
                number of beacons with bigger magnetic module than
                sensibility needed in an optimal point','n_beacons_dif:
                minimun number of pairs of beacons whose modules cant
                superate the difference: Bmin<0.01Bmax','d,ds,d_exp:
                distance between points (meshgrid) in optimization and
                solution and overexosure representatios respectively','dc:
                 side length of cubes centered in every beacon where
                study overexposure', 'maximun_exposure: maximun magnetic
                field permited (human security)','Tbeacon: representation
                 factor for beacons (increase or reduce their length)'])
        if index == 6:
            messagebox.showinfo('Help','Here you can try any distribution
                 of beacons that you would like to consider')

    #_____________________________________________________________________
    #ANADIR FILA PARA DEFINIR VOLUMEN DEL RECINTO
    def addVopt(self):
        n = len(self.Vopt) # Número de filas
        self.VoptLabel.append(ttk.Label(self.f1111,text=f'{n+1}:',font='
            times 9'))
        self.VoptLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.Vopt.append([])
        for i in range(6):
            self.Vopt[n].append(ttk.Entry(self.f1111,width=5))
            self.Vopt[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5))

    #_____________________________________________________________________
    #ELIMINAR FILA DE DEFINICION DE VOLUMEN DEL RECINTO
    def delVopt(self):
```

```
        n = len(self.Vopt) # Número de filas
        if n > 1:
            self.VoptLabel[n-1].destroy()
            self.VoptLabel.pop(n-1)
            for i in range(6):
                self.Vopt[n-1][i].destroy()
            self.Vopt.pop(n-1)
        else:
            for i in range(6):
                self.Vopt[n-1][i].delete(0,100)

    #_____________________________________________________________________
    #REPRESENTAR RECINTO A OPTIMIZAR
    def drawVopt(self):
        n = len(self.Vopt) # Número de filas
        self.axVopt.cla() # Clear axes
        self.axVopt.set_title('Optimization volume (red)')
        self.axVopt.set_xlabel('X [m]')
        self.axVopt.set_ylabel('Y [m]')
        self.axVopt.set_zlabel('Z [m]')
        for i in range(n):
            try:
                xmin = float(self.Vopt[i][0].get())
                xmax = float(self.Vopt[i][1].get())
                ymin = float(self.Vopt[i][2].get())
                ymax = float(self.Vopt[i][3].get())
                zmin = float(self.Vopt[i][4].get())
                zmax = float(self.Vopt[i][5].get())
            except:
                messagebox.showerror('Impossible to update',f'Row {i+1}
                    contains no numerical inputs')
                return False
            if xmax < xmin or ymax < ymin or zmax < zmin:
                messagebox.showerror('Impossible to update',f'In row {i+1}
                     an upper limit is inferior to its lower limit')
                return False
            kwargs = {'alpha': 1, 'color': 'red'}
            self.axVopt.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                ymax,ymax,ymin], [zmin]*5, **kwargs)
            self.axVopt.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [zmin,
                zmax,zmax], **kwargs)
            self.axVopt.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVopt.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVopt.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [zmin,
                zmax,zmax], **kwargs)
        self.canvasVopt.draw() # Actualizar canvas
        return True
```

```
    #_____________________________________________________________________
    #ANADIR VOLMEN DE MOVIMIENTO DE BALIZA
    def addVant(self):
        #al añadir 1 antena añadimos filas en la definicion del volumen
            de la antena, también añadimos 1 fila en la pestaña de pará
            metros de antenas y también añadimos una fila en la pestaña
            de prueba de otras distribuciones
        n = len(self.Vant) # Número de filas
        self.VantLabel.append(ttk.Label(self.f2111,text=f'{n+1}:',font='
            times 9'))
        self.VantLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.Vant.append([])
        self.paramLabel.append(ttk.Label(self.f4111,text=f'{n+1}:',font='
            times 9'))
        self.paramLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.param.append([])
        self.TryparamLabel.append(ttk.Label(self.f7111,text=f'{n+1}:',
            font='times 9'))
        self.TryparamLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.Tryparam.append([])
        for i in range(6):
            self.Vant[n].append(ttk.Entry(self.f2111,width=5))
            self.Vant[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5))
        for i in range(8):
            self.param[n].append(ttk.Entry(self.f4111,width=8))
            self.param[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5))
            if i==0:
                self.param[n][i].insert(-1, '-180')
            elif i==1:
                self.param[n][i].insert(-1, '180')
            elif i==2:
                self.param[n][i].insert(-1, '-90')
            elif i==3:
                self.param[n][i].insert(-1, '90')
            elif i==4:
                self.param[n][i].insert(-1, '500')
            elif i==5:
                self.param[n][i].insert(-1, '0.6')
            elif i==6:
                self.param[n][i].insert(-1, '1')
            elif i==7:
                self.param[n][i].insert(-1, '1')
        for i in range(5):
            self.Tryparam[n].append(ttk.Entry(self.f7111,width=8))
            self.Tryparam[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5)
                )

    #_____________________________________________________________________
    def addSameVant(self): #añadir mismo volumen de antena que el
        anterior
```

```
        n = len(self.Vant) # Número de filas
        self.VantLabel.append(ttk.Label(self.f2111,text=f'{n+1}:',font='
            times 9'))
        self.VantLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.Vant.append([])
        self.paramLabel.append(ttk.Label(self.f4111,text=f'{n+1}:',font='
            times 9'))
        self.paramLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.param.append([])
        self.TryparamLabel.append(ttk.Label(self.f7111,text=f'{n+1}:',
            font='times 9'))
        self.TryparamLabel[n].grid(row=n+1,column=0,pady=(0,5))
        self.Tryparam.append([])
        for i in range(6):
            self.Vant[n].append(ttk.Entry(self.f2111,width=5))
            self.Vant[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5))
            self.Vant[n][i].insert(-1, f'{str(self.Vant[n-1][i].get())}')
        for i in range(8):
           self.param[n].append(ttk.Entry(self.f4111,width=8))
           self.param[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5))
           if i==0:
                self.param[n][i].insert(-1, '-180')
           elif i==1:
                self.param[n][i].insert(-1, '180')
           elif i==2:
                self.param[n][i].insert(-1, '-90')
           elif i==3:
                self.param[n][i].insert(-1, '90')
           elif i==4:
                self.param[n][i].insert(-1, '500')
           elif i==5:
                self.param[n][i].insert(-1, '0.6')
           elif i==6:
                self.param[n][i].insert(-1, '1')
           elif i==7:
                self.param[n][i].insert(-1, '1')
        for i in range(5):
            self.Tryparam[n].append(ttk.Entry(self.f7111,width=8))
            self.Tryparam[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5)
                )

    #_____________________________________________________________________
    #ELIMINAR BALIZA
    def delVant(self):
        n = len(self.Vant) # Número de filas
        if n > 1:
            self.VantLabel[n-1].destroy()
            self.VantLabel.pop(n-1)
            for i in range(6):
                self.Vant[n-1][i].destroy()
```

```
            self.Vant.pop(n-1)

            self.paramLabel[n-1].destroy()
            self.paramLabel.pop(n-1)
            for i in range(8):
                self.param[n-1][i].destroy()
            self.param.pop(n-1)

            self.TryparamLabel[n-1].destroy()
            self.TryparamLabel.pop(n-1)
            for i in range(5):
                self.Tryparam[n-1][i].destroy()
            self.Tryparam.pop(n-1)
        else:
            for i in range(6):
                self.Vant[n-1][i].delete(0,100)
            for i in range(5):
                self.Tryparam[n-1][i].delete(0,100)
            for i in range(8):
                self.param[n-1][i].delete(0,100)
                self.param[0].append(ttk.Entry(self.f4111,width=8))
                self.param[0][i].grid(row=1,column=i+1,padx=5,pady=(0,5))
                #Definimos valores iniciales
                if i==0:
                    self.param[0][i].insert(-1, '-180')
                elif i==1:
                    self.param[0][i].insert(-1, '180')
                elif i==2:
                    self.param[0][i].insert(-1, '-90')
                elif i==3:
                    self.param[0][i].insert(-1, '90')
                elif i==4:
                    self.param[0][i].insert(-1, '500')
                elif i==5:
                    self.param[0][i].insert(-1, '0.6')
                elif i==6:
                    self.param[0][i].insert(-1, '1')
                elif i==7:
                    self.param[0][i].insert(-1, '1')

    #_____________________________________________________________________
    #REPRESENTAR VOLUMENES DE MOVIMIENTO DE BALIZAS
    def drawVant(self):
        #representar Vopt
        n = len(self.Vopt) # Número de filas Vopt
        self.axVant.cla() # Clear axes
        self.axVant.set_title('Antennas movement volumes (blue)')
        self.axVant.set_xlabel('X [m]')
        self.axVant.set_ylabel('Y [m]')
        self.axVant.set_zlabel('Z [m]')
```

```
        for i in range(n):
            xmin = float(self.Vopt[i][0].get())
            xmax = float(self.Vopt[i][1].get())
            ymin = float(self.Vopt[i][2].get())
            ymax = float(self.Vopt[i][3].get())
            zmin = float(self.Vopt[i][4].get())
            zmax = float(self.Vopt[i][5].get())

            kwargs = {'alpha': 1, 'color': 'red'}
            self.axVant.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                ymax,ymax,ymin], [zmin]*5, **kwargs)
            self.axVant.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [zmin,
                zmax,zmax], **kwargs)

        #representar Vant
        n = len(self.Vant) # Número de filas Vant
        for i in range(n):
            try:
                xmin = float(self.Vant[i][0].get())
                xmax = float(self.Vant[i][1].get())
                ymin = float(self.Vant[i][2].get())
                ymax = float(self.Vant[i][3].get())
                zmin = float(self.Vant[i][4].get())
                zmax = float(self.Vant[i][5].get())
            except:
                messagebox.showerror('Impossible to update',f'Row {i+1}
                    contains no numerical inputs')
                return False
            if xmax < xmin or ymax < ymin or zmax < zmin:
                messagebox.showerror('Impossible to update',f'In row {i+1}
                     an upper limit is inferior to its lower limit')
                return False
            kwargs = {'alpha': 1, 'color': 'blue'}
            self.axVant.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                ymax,ymax,ymin], [zmin]*5, **kwargs)
            self.axVant.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVant.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [zmin,
                zmax,zmax], **kwargs)
        self.canvasVant.draw() # Actualizar canvas
```

```
        return True

    #_____________________________________________________________________
    #ANADIR VOLUMEN PROHIBIDO
    def addVpro(self):
        n = len(self.Vpro) # Número de filas
        if self.Vpro==[[]]:
             self.VproLabel = [] # Inicializar vector de identificadores
                 de filas
             self.VproLabel.append(ttk.Label(self.f3111,text='1:',font='
                 times 9'))
             self.VproLabel[0].grid(row=1,column=0,pady=(0,5))
             for i in range(6):
                self.Vpro[0].append(ttk.Entry(self.f3111,width=5))
                self.Vpro[0][i].grid(row=1,column=i+1,padx=5,pady=(0,5))
        else:
            self.VproLabel.append(ttk.Label(self.f3111,text=f'{n+1}:',
                font='times 9'))
            self.VproLabel[n].grid(row=n+1,column=0,pady=(0,5))
            self.Vpro.append([])
            for i in range(6):
                self.Vpro[n].append(ttk.Entry(self.f3111,width=5))
                self.Vpro[n][i].grid(row=n+1,column=i+1,padx=5,pady=(0,5)
                    )

    #_____________________________________________________________________
    #ELIMINAR VOLUMEN PROHIBIDO
    def delVpro(self):
        n = len(self.Vpro) # Número de filas
        if self.Vpro!=[[]]:
            self.VproLabel[n-1].destroy()
            self.VproLabel.pop(n-1)
            for i in range(6):
                self.Vpro[n-1][i].destroy()
            self.Vpro.pop(n-1)
        if n==1:
            self.Vpro=[[]]

    #_____________________________________________________________________
    #REPRESENTAR VOLUMEN PROHIBIDO
    def drawVpro(self):
        #representar Vopt
        n = len(self.Vopt) # Número de filas Vopt
        self.axVpro.cla() # Clear axes
        self.axVpro.set_title('Prohibited volumes(green)')
        self.axVpro.set_xlabel('X [m]')
        self.axVpro.set_ylabel('Y [m]')
        self.axVpro.set_zlabel('Z [m]')
        for i in range(n):
            xmin = float(self.Vopt[i][0].get())
```

```
            xmax = float(self.Vopt[i][1].get())
            ymin = float(self.Vopt[i][2].get())
            ymax = float(self.Vopt[i][3].get())
            zmin = float(self.Vopt[i][4].get())
            zmax = float(self.Vopt[i][5].get())

            kwargs = {'alpha': 1, 'color': 'red'}
            self.axVpro.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                ymax,ymax,ymin], [zmin]*5, **kwargs)
            self.axVpro.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [zmin,
                zmax,zmax], **kwargs)

        #representar Vant
        n = len(self.Vant) # Número de filas Vant
        for i in range(n):

            xmin = float(self.Vant[i][0].get())
            xmax = float(self.Vant[i][1].get())
            ymin = float(self.Vant[i][2].get())
            ymax = float(self.Vant[i][3].get())
            zmin = float(self.Vant[i][4].get())
            zmax = float(self.Vant[i][5].get())

            kwargs = {'alpha': 1, 'color': 'blue'}
            self.axVpro.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                ymax,ymax,ymin], [zmin]*5, **kwargs)
            self.axVpro.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [zmin,
                zmax,zmax], **kwargs)
            self.axVpro.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [zmin,
                zmax,zmax], **kwargs)
        self.canvasVpro.draw() # Actualizar canvas

        #representar Vpro
        n = len(self.Vpro) # Número de filas Vpro
        if self.Vpro==[[]]:
            return True
        else:
            for i in range(n):
                try:
                    xmin = float(self.Vpro[i][0].get())
```

```
                    xmax = float(self.Vpro[i][1].get())
                    ymin = float(self.Vpro[i][2].get())
                    ymax = float(self.Vpro[i][3].get())
                    zmin = float(self.Vpro[i][4].get())
                    zmax = float(self.Vpro[i][5].get())
                except:
                    messagebox.showerror('Impossible to update',f'Row {i
                        +1} contains no numerical inputs')
                    return False
                if xmax < xmin or ymax < ymin or zmax < zmin:
                    messagebox.showerror('Impossible to update',f'In row {
                        i+1} an upper limit is inferior to its lower limit'
                        )
                    return False
                kwargs = {'alpha': 1, 'color': 'green'}
                self.axVpro.plot3D([xmin,xmax,xmax,xmin,xmin], [ymin,ymin,
                    ymax,ymax,ymin], [zmin]*5, **kwargs)
                self.axVpro.plot3D([xmin,xmin,xmax], [ymin,ymin,ymin], [
                    zmin,zmax,zmax], **kwargs)
                self.axVpro.plot3D([xmax,xmax,xmax], [ymin,ymin,ymax], [
                    zmin,zmax,zmax], **kwargs)
                self.axVpro.plot3D([xmax,xmax,xmin], [ymax,ymax,ymax], [
                    zmin,zmax,zmax], **kwargs)
                self.axVpro.plot3D([xmin,xmin,xmin], [ymax,ymax,ymin], [
                    zmin,zmax,zmax], **kwargs)
        self.canvasVpro.draw() # Actualizar canvas
        return True

def main():
    root = Tk() # Main window
    myApp(root) # Constructor on root
    root.title("Freddie") # Nombre de la app
    root.update_idletasks()
    root.mainloop() # Execute

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

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] M. Compte. (2018) Posicionamiento indoor. [Online]. Available: https://www.unigis.es/posicionamiento-indoor/

[2] Wikipedia. (2021) Trilateración. [Online]. Available: https://es.wikipedia.org/wiki/Trilateración

[3] J. de Dios Muñoz Guzmán, *TFG: estudio de implementación sobre el sistema de posicionamiento indoor SILEME*. Universidad de Sevilla, 2019.

[4] «BOE». núm. 234, de 29/09/2001. real decreto 1066/2001, de 28 de septiembre, por el que se aprueba el reglamento que establece condiciones de protección del dominio público radioeléctrico, restricciones a las emisiones radioeléctricas y medidas de protección sanitaria frente a emisiones radioeléctricas. [Online]. Available: https://www.boe.es/eli/es/rd/2001/09/28/1066/con

[5] «DOUE». núm. 179, de 29 de junio de 2013. directiva 2013/35/ue del parlamento europeo y del consejo, de 26 de junio de 2013, sobre las disposiciones mínimas de salud y seguridad relativas a la exposición de los trabajadores a los riesgos derivados de agentes físicos (campos electromagnéticos) (vigésima directiva específica con arreglo al artículo 16, apartado 1, de la directiva 89/391/cee), y por la que se deroga la directiva 2004/40/ce.

[6] «BOE». núm. 182, de 29 de julio de 2016. real decreto 299/2016, de 22 de julio, sobre la protección de la salud y la seguridad de los trabajadores contra los riesgos relacionados con la exposición a campos electromagnéticos. [Online]. Available: https://www.boe.es/eli/es/rd/2016/07/22/299

[7] Matplotlib. (2012) Streamplot. [Online]. Available: https://matplotlib.org/stable/gallery/images_contours_and_fields/plot_streamplot.html

[8] Wikipedia. (2020) Optimización por enjambre de partículas. [Online]. Available: https://es.wikipedia.org/wiki/Optimización_por_enjambre_de_partículas

[9] Particle swarm optimization. [Online]. Available: https://pypi.org/project/pyswarm/

[10] Matplotlib. (2012) mplot3d. [Online]. Available: https://matplotlib.org/2.0.2/mpl_toolkits/mplot3d/tutorial.html#surface-plots

[11] xiaodong. (2018) Use matplotlib to draw 3d cube plots. [Online]. Available: https://www.programmersought.com/article/10884559421/

[12] Matplotlib. (2012) Colormap normalization. [Online]. Available: https://matplotlib.org/stable/tutorials/colors/colormapnorms.html

[13] A. Jana. (2018) How to visualize gradient descent using contour plot in python. [Online]. Available: http://www.adeveloperdiary.com/data-science/how-to-visualize-gradient-descent-using-contour-plot-in-python/

[14] Python. (2021) Graphical user interfaces with tk. [Online]. Available: https://docs.python.org/3/library/tkinter.html