Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Herramienta para la Generación de Superficies Recubiertas de Materiales Absorbentes al Radar (RAM) Mediante Parámetros Estadísticos

Autor: Mario Alan Solanillas Blanco

Tutor/es: Carlos Delgado Hita

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo de Fin de Grado

Herramienta para la Generación de Superficies Recubiertas de Materiales Absorbentes al Radar (RAM) Mediante Parámetros Estadísticos

Autor: D. Mario Alan Solanillas Blanco

Tutor: D. Carlos Delgado Hita

Tribunal:

Presidente: D.

Vocal 1: D.

Vocal 2 (Tutor): D.

Fecha: diciembre de 2020

Agradecimientos a mi familia y amigos,

que en un año tan complicado como este me han apoyado en todo momento.

Gracias a todos de corazón.

Índice de Apartados

Índice de figuras

Resumen

Este trabajo ha sido pensado con la idea de crear una aplicación, mediante el uso del lenguaje de *MATLAB*, para que el usuario que haga uso de ella cree una serie de ficheros que contengan unos comandos necesarios para la generación de modelados geométricos a partir de unos parámetros estadísticos, para que posteriormente sean cargados y procesados por el software *NewFASANT*.

El procedimiento a seguir por parte del usuario para la generación de dichas geometrías, será, en primera instancia, la descarga de dicha aplicación, después hacer uso de ella para dimensionar los parámetros necesarios para su elaboración, crear el fichero y por último cargarlo posteriormente en el software de simulación de *NewFASANT*, donde el usuario podrá realizar diferentes estudios y simulaciones, en nuestro caso con el uso de ondas planas, que tendrá posteriores aplicaciones en el ámbito de la ingeniería.

Palabras clave:

- *MONCROS*
- *Método de los Momentos*
- *MATLAB*
- *NewFASANT*
- *Ondas Electromagnéticas*
- *Interfaz de usuario*
- *Modelado geométrico*

Abstract

The work presented in this document is based on the development of an application using the *MATLAB* scripting language in order to allow the user to generate a series of files containing the commands required to create geometrical models using statistical parameters that, in turn, can be loaded and processed using the *NewFASANT* software suite.

The procedure followed to generate such geometries will entail downloading the application, after which it can be used to configure the required parameters, generate the script file and finally load it on the simulation software suite *NewFASANT*, where the user can perform different analyses and simulations, in this case using plane waves as excitations, with further applications in the engineering field.

Keywords:

- *MONCROS*
- *Method of Moments*
- *MATLAB*
- *NewFASANT*
- *Electromagnetic waves*
- *User interface*
- *Geometric modeling*

1. Introducción

1.1. Visión general de la aplicación

Este documento presentará de forma metódica y detallada el funcionamiento de la herramienta software para la creación de ficheros que generen superficies con material absorbente al radar, mediante parámetros estadísticos, para su posterior simulación mediante el programa *NewFASANT*.

Los ficheros generados por parte de la aplicación contendrán los comandos necesarios para que, una vez cargado en el software *NewFASANT*, este sea capaz de interpretarlos correctamente y generar la geometría deseada por el usuario. La finalidad de la generación de estos escenarios geométricos es su posterior implementación en simulaciones con ondas electromagnéticas en una banda y polarización determinada.

Esta aplicación está creada mediante el entorno de trabajo de *MATLAB* y podrá ser usada por cualquier usuario que disponga de este entorno de trabajo. El usuario en cuestión, una vez que abra la aplicación, podrá hacer uso de una interfaz de usuario intuitiva y sencilla donde se le permitirá dimensionar una serie de parámetros estadísticos que, después, creen el fichero mencionado para la generación del modelado geométrico.

1.2. Objetivos

El principal objetivo de este proyecto es el desarrollo de la aplicación para la generación de los ficheros con los comandos necesarios para que el software de simulación electromagnética *NewFASANT* sea capaz de interpretarlos debidamente, y así poder generar las superficies recubiertas con material absorbente al radar, y su posterior simulación.

Para poder generar este tipo de superficies, se utilizará una serie de elementos resistivos que se encontrarán distribuidos sobre el sustrato original mediante unos parámetros estadísticos, de forma que, dependiendo de su distribución y la frecuencia y la polarización de onda, se obtengan unos valores de onda u otros, como se podrá ver posteriormente.

Este objetivo principal que se pretende desarrollar a lo largo de este proyecto se realizará cumpliendo de igual forma el objetivo de que la aplicación sea lo más sencilla e intuitiva al mismo tiempo que la generación de estos escenarios sean los correctos y deseados. De este modo, conseguiremos que la funcionalidad de esta lo más accesible posible para el usuario y que su uso sea lo más satisfactorio que se pueda, de la misma forma que los escenarios generados cumplan con las especificaciones necesarias para su posterior estudio.

Estos escenarios, como se especifica en el título, se generarán mediante parámetros estadísticos, por lo que la aplicación deberá responder correctamente a esa

aleatoriedad sin que se produzcan potenciales errores que puedan aparecer cuando el usuario dimensiones los diferentes parámetros que ofrezca esta, estableciendo de manera correcta las proporciones del escenario geométrico para un buen estudio posterior.

2. Base teórica

En este apartado se expondrán los conceptos teóricos que son utilizados en este proyecto, es decir, que se explicará de manera más amplia el concepto de los materiales absorbentes al radar (RAM) y las ondas electromagnéticas.

2.1. Coordenadas esféricas

Para comenzar, se explicarán unas pequeñas nociones de las coordenadas esféricas, ya que en las simulaciones realizadas en *NewFASANT* que se mostrarán posteriormente, se ha trabajado en este tipo de coordenadas, por lo que haremos un breve resumen de estas.

Normalmente en los cálculos diarios que realizamos trabajamos con coordenadas rectangulares (o cartesianas), pero para algunos casos donde sea más interesante tener como referencias a los ángulos que a las distancias, como puede ser en el caso del tipo de simulaciones que vayamos a realizar ya que en lo que nos vayamos a fijar será en la dirección de onda, pues se utilizarán este tipo de coordenadas.

Las coordenadas esféricas son el caso de las polares, pero extendidas a tres dimensiones, siendo este un sistema similar al sistema utilizado para denotar altitud y latitud de un punto en la superficie de la Tierra, siendo el origen de coordenadas el centro de esta.

De manera gráfica, este sistema de coordenadas sería el siguiente:

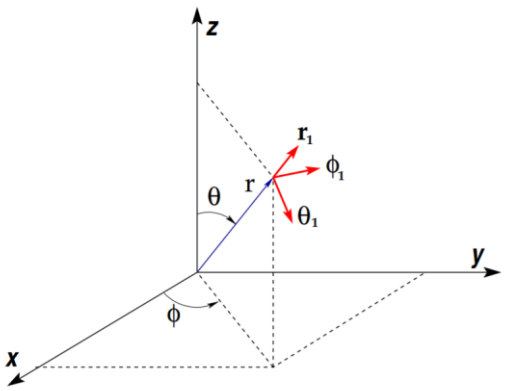


Ilustración 2.1.1 Coordenadas esféricas

En la ilustración 2.1.1, extraída de la referencia [1], se muestran las coordenadas esféricas de un punto cualquiera, siendo r la distancia de este al origen de coordenadas, θ el ángulo entre el eje Z y el radio vector, y, por último, ϕ el ángulo entre el eje X y la sombra del radio vector sobre el plano XY.

En este proyecto, no usaremos este sistema directamente para definir los diferentes parámetros de nuestra geometría, pero sí se usará el sistema polar para la simulación

en el software *NewFASANT*, por lo que es interesante tener unas nociones básicas de este.

2.2. Electromagnetismo

A continuación, seguiremos hablando del electromagnetismo para poder tener una mayor noción de los conceptos que se verán a lo largo del proyecto.

Según Ezequiel del Río, doctor de la Universidad Politécnica de Madrid [1], el electromagnetismo es la disciplina que estudia los campos eléctricos y magnéticos y su interacción con la materia. Esta ciencia nacería de la mano de James Clerk Maxwell, con la síntesis de estas ideas en las siguientes cuatro ecuaciones:

$$\nabla \cdot \vec{B} = 0 \quad (2.1)$$

$$\nabla \wedge \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad (2.2)$$

$$\nabla \cdot \vec{E} = \frac{\rho_e}{\varepsilon_0} \quad (2.3)$$

$$\nabla \wedge \vec{B} = \frac{1}{c^2} \cdot \frac{\partial \vec{E}}{\partial t} + \mu_0 \vec{J} \quad (2.4)$$

Estas cuatro ecuaciones de Maxwell describen el comportamiento de los campos eléctricos, $\vec{E}$, y magnéticos, $\vec{B}$, con los que después se pueden basar los motores eléctricos, por ejemplo. De estas ecuaciones también se deducirá el comportamiento de la propagación de ondas electromagnéticas, como se verá más adelante.

Estas ecuaciones cambiarían inevitablemente el trascurso de la historia de la humanidad, ya que el posterior uso de estas para la física moderna es imprescindible, llegando su uso práctico hasta nosotros a través de los dispositivos eléctricos y electrónicos que utilizamos de forma diaria.

Los campos eléctricos provienen de la fuerza electrostática de un sistema de partículas con cargas q_j en $\vec{r_j}$ sobre otra carga q en $\vec{r}$, la cual viene dada por la ley de Coulomb:

$$\overrightarrow{F_e} = \frac{1}{4\pi\varepsilon_0} \sum_j q_j \, q \, \frac{\vec{r} - \vec{r_j}}{\left|\vec{r} - \overrightarrow{r'}\right|} \quad (2.5)$$

siendo ε_0 la constante eléctrica. Lo que ocurre es que las cargas q_j crean un campo electrostático, el cual se rige de la siguiente forma:

$$\vec{E}(\vec{r}) = \frac{1}{4\pi\varepsilon_0} \sum_j q_j \frac{\vec{r} - \vec{r_j}}{\left|\vec{r} - \overrightarrow{r'}\right|^3} \quad (2.6)$$

Es decir, que una carga arbitraria q que se encuentra en un punto también arbitrario $\vec{r}$, sufre una fuerza $\vec{F_e} = q\vec{E}(\vec{r})$. Esto podemos ver que ocurre puntualmente en una sola carga, pero para una distribución de cargas con densidad $\rho_e(\vec{r})$, tendremos un valor de un campo electrostático que se regirá de la siguiente forma:

$$\vec{E}(\vec{r}) = \frac{1}{4\pi\varepsilon_0}\int_v \rho_e(\vec{r'})\frac{\vec{r}-\vec{r_j}}{\left|\vec{r}-\vec{r'}\right|^3}\,dv' \quad (2.7)$$

Siendo v cualquier volumen fuera del cual $\rho_e = 0$. Podemos representar los conceptos anteriores de la siguiente forma:

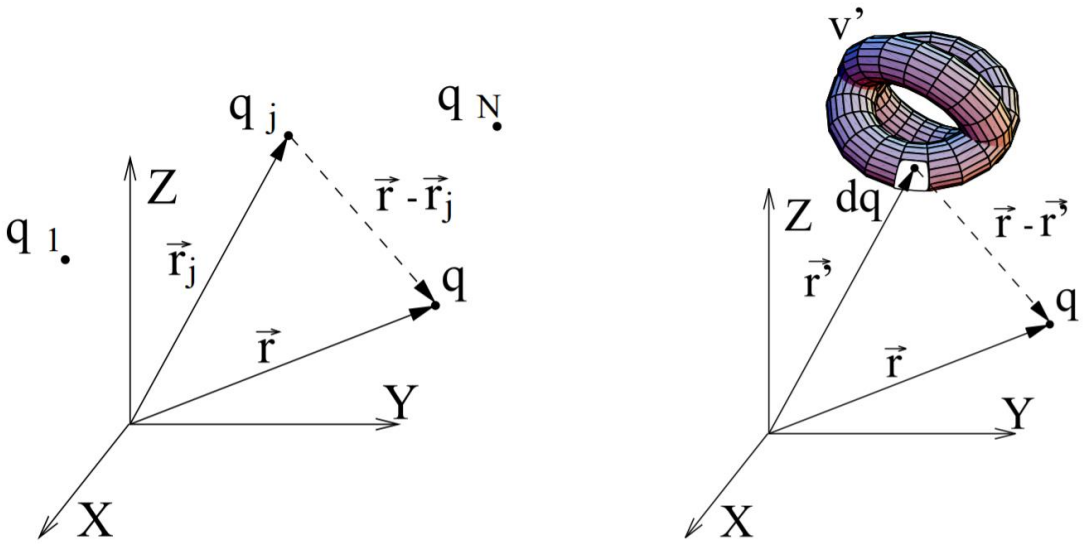


Ilustración 2.2.1 Campo electrostático en cargas puntuales y volúmenes con densidad de carga

En la imagen de la izquierda de la ilustración 2.2.1, referenciada en [1], tenemos sistema de cargas puntuales que crea un campo electrostático en las proximidades de q, siendo la representación gráfica de la ecuación (2.6), y en la derecha por otra parte, tenemos un volumen v' con densidad de carga ρ_e creando un campo electrostático alrededor de q, tal y como se describe en la ecuación (2.7).

Si volvemos a las ecuaciones de Maxwell, en específico a la ecuación (2.3), podemos ver que se puede llegar a deducir la siguiente expresión:

$$\nabla \wedge \vec{E} = 0 \quad (2.8)$$

Por lo que el campo electrostático se puede expresar en una función potencial:

$$\vec{E} = -\nabla \mathrm{V} \quad (2.9)$$

Donde V es el potencial electrostático. De esta ecuación, y aplicando el teorema fundamental del cálculo para integrales de línea de la ecuación, se obtiene:

$$\int_a^b \vec{E} \cdot \overrightarrow{dl} = -\int_a^b \nabla \mathrm{V} \cdot \overrightarrow{dl} = V(\overrightarrow{r_a}) - V(\overrightarrow{r_b}) \quad (2.10)$$

Esta expresión sirve para poder conseguir el potencial a partir del campo electrostático, por lo que según las expresiones (2.6) y (2.7) conseguimos que:

$$V = \frac{1}{4\pi\varepsilon_0} \sum_j \frac{q_j}{\left|\vec{r} - \vec{r_j}\right|^3} \quad (2.11)$$

O bien:

$$V = \frac{1}{4\pi\varepsilon_0} \int_v \frac{\rho(\overrightarrow{r'})}{\left|\vec{r} - \overrightarrow{r'}\right|} \, dv' \quad (2.12)$$

Explicando esta última y las fórmulas que contienen una distribución de carga suave $\rho_e(\overrightarrow{r'})$ en lugar de cargas puntuales, el comportamiento macroscópico del campo electrostático y el potencial eléctrico.

2.3. Ondas electromagnéticas

Ahora pasaremos a hablar de las ondas electromagnéticas, las cuales son las que no necesitan un medio material para propagarse, es decir, que hablamos de un tipo de ondas como la luz, ondas de radio, telefonía, televisión, etc.

Estas se propagan haciendo oscilar el campo eléctrico y magnético que tienen a su alrededor a una velocidad muy elevada, la conocida como velocidad de la luz (c=300000km/s), abarcando todas las frecuencias de onda posibles.

Las ondas electromagnéticas se encuentran formadas por fotones que viajan a través del espacio hasta que interactúan con la materia que se halla a su paso, pudiendo ser absorbidas por esta o reflejada.

Para comentar físicamente estas ondas, describiremos este tipo de ondas en el vacío, es decir, suponiendo el campo electromagnético que hay a su alrededor en el vacío, el cual se encuentra determinado por las ecuaciones de Maxwell (Ecuaciones (2.1), (2.2) (2.3) y (2.4)).

Para este caso en el que nos encontramos en el vacío, supondremos que las densidades de carga (ρ_t) y las de corriente ($\overrightarrow{J_t}$) son nulas, haciendo de las ecuaciones (2.3) y (2.4) las siguientes:

$$\nabla \cdot \vec{E} = 0 \quad (2.13)$$

$$\nabla \wedge \vec{B} = \frac{1}{c^2} \cdot \frac{\partial \vec{E}}{\partial t} \quad (2.14)$$

Lo que implica que un campo electromagnético puede existir aun no existiendo cargas. Los campos electromagnéticos que existen en el vacío en ausencia de cargas son llamados ondas electromagnéticas.

Podemos observar que con los campos de solución de la ecuación (2.14) ha de variar con el tiempo, sino tendríamos ecuaciones de electrostática además de una ausencia de cargas, lo que provoca que el valor del campo sea 0. Por lo que a través de una serie de cálculos numéricos y tomando en referencias las ecuaciones (2.13) y (2.14), se llega a la ecuación de onda:

$$\nabla^2\psi - \frac{1}{c^2}\cdot\frac{\partial^2\psi}{\partial t^2} = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} + \frac{\partial^2\psi}{\partial z^2} - \frac{1}{c^2}\cdot\frac{\partial^2\psi}{\partial t^2} \quad (2.15)$$

Donde ψ representa cualquiera de las cantidades $E_x, E_y, E_z, B_x, B_y, B_z$.

Así pues, si se introduce el concepto arbitrario de la función $\psi(\xi)$ de una variable real, donde $\xi = \vec{k}\cdot\vec{r} - \omega t + \phi$, obtenemos que se satisface lo siguiente:

$$c^2\cdot k^2 = \omega^2 \quad (2.16)$$

Las ondas electromagnéticas de esta forma ($\psi(\vec{k}\cdot\vec{r} - \omega t + \phi)$) se llaman ondas planas y son un tipo de ondas que nos son más familiares. Existen un tipo de ondas planas que se denominan ondas monocromáticas planas y responden a la siguiente ecuación:

$$\psi = \psi_0\cdot sen\left(\vec{k}\cdot\vec{r} - \omega t + \phi\right) \quad (2.17)$$

Estas ondas deben su nombre a que su frecuencia sí que corresponde a la parte visible por el ser humano. Las ondas electromagnéticas se pueden clasificar según su frecuencia, siendo el espectro electromagnético el siguiente:

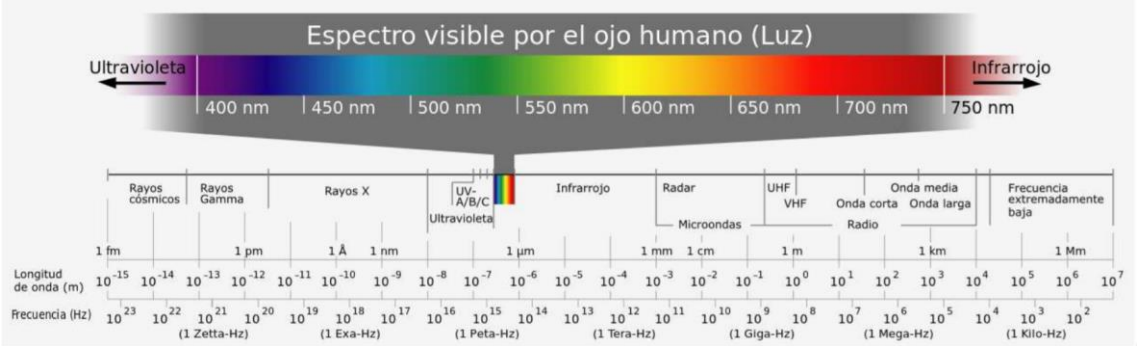


Ilustración 2.3.1 Espectro electromagnético

Como se puede ver en la ilustración 2.3.1, referenciada de [1], las categorizaciones de estas ondas se basan en su frecuencia o longitud de onda que tienen, siendo esta clasificación como sigue:

- **Ondas de radio:** son las ondas de frecuencia más baja (las frecuencias anteriores a estas son tan bajas que no llegan a poder clasificarse dentro de las ondas

electromagnéticas). Se encuentran dentro del rango desde unos pocos KHz hasta los GHZ, y son las que se suele usar en telefonía, radio y televisión. Aunque este es su uso común, la verdad es que la mayoría de los objetos de nuestro entorno emiten este tipo de ondas, ya que al emitir calor se emite a la vez radiación en todo el espectro, siendo las ondas de radio las de mayor cantidad.

- **Microondas:** se encuentran clasificadas por encima de las ondas de radio y son el segundo tipo de onda con el espectro más bajo de frecuencia, encontrándose en el rango de los GHz. A diferencia de las ondas de radio que pueden medir kilómetros de longitud, estas ondas miden pocos centímetros, pero pueden atravesar una cantidad mayor de objetos por su mayor frecuencia. Comúnmente utilizadas en hornos de las cocinas.
- **Ondas infrarrojas:** normalmente las emiten los cuerpos calientes y se encuentran en frecuencias inferiores a las de la luz visible (de 10^{11}Hz hasta 10^{14}Hz). Es por esto que un uso común de estas ondas pueden ser los equipos de visión nocturna, donde se detectan este tipo de ondas que son emitidas por los cuerpos que irradian calor.
- **Luz visible:** son las que nos permiten ver el mundo que nos rodea y suponen una franja muy pequeña, desde los $4x10^{14}$Hz hasta los $8x10^{14}$Hz. Los humanos tenemos la capacidad de detectar este tipo de ondas e interpretarlas debido a los sensores oculares (conos y bastones de nuestra retina) que poseemos.
- **Ultravioleta:** estas ondas comprenden un rango superior de frecuencias a las de la luz visible, desde los $8x10^{14}$Hz hasta los 10^{17}Hz. Tienen este nombre porque se encuentran justo después del color violeta, y son las ondas que provocan daños en nuestra piel y como consecuencia fabricamos melanina para protegernos.
- **Rayos X:** son las ondas que comprenden un rango desde los 10^{19}Hz hasta los 10^{19}Hz, siendo ya frecuencias bastante elevadas y, por lo tanto, potencialmente peligrosas en largos periodos de exposición a ellas. Son las que se utilizan para realizar radiografías en medicina.
- **Rayos gamma:** comprenden las frecuencias más altas del espectro (frecuencias superiores a 10^{19}Hz), por lo que solo los objetos más energéticamente cargados emiten este tipo de ondas, como las estrellas de neutrones, supernova o las desintegraciones radioactivas.

2.4. Polarización de onda

Una vez vista ya la definición de onda electromagnética y sus tipos, estudiaremos su polarización. La polarización es la característica que tiene toda onda transversal y podemos definirla como la figura geométrica que describe la punta de la onda a lo largo del tiempo. Para ver esto, tomaremos como consideración una onda plana monocromática.

Ya hemos visto antes la ecuación de las ondas planas monocromáticas (ecuación (2.17)), y si tomamos que la onda se propaga a través del eje X ($\vec{k} = k\vec{\imath}$), tenemos que la onda obra la siguiente forma:

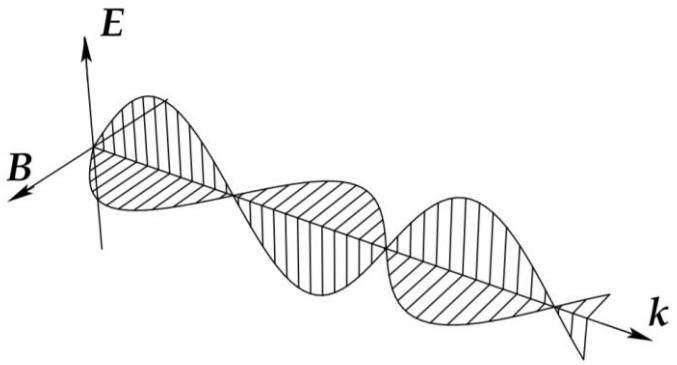


Ilustración 2.4.1 Onda plana monocromática

Vemos en la ilustración 2.4.1, referenciada de [1], que la onda que ahora los vectores $\vec{E}$ y $\vec{B}$ son paralelos a los ejes Y y Z respectivamente, es decir que $E_x = 0$ y $B_x = 0$. Esto hace que la onda plana de la imagen superior parta de la siguiente ecuación:

$$\vec{E} = E_x\vec{j} + E_z\vec{K} = E_{y_0} \cdot sen\big(kx - \omega t)\vec{j} + E_{z_0} \cdot sen(kx - \omega t + \phi\big)\vec{K} \quad (2.18)$$

Partiendo de las relaciones:

$$\frac{E_y}{E_{y_0}} = sen(kx - \omega t) \quad (2.19)$$

Y:

$$\frac{E_z}{E_{z_0}} = sen(kx - \omega t + \Delta) \quad (2.20)$$

Podemos llegar, tras una serie de largos cálculos, a la siguiente expresión:

$$\frac{E_y^2}{E_{y_0}^{\ 2}} + \frac{E_z^2}{E_{z_0}^{\ 2}} - 2\cos\Delta\frac{E_y}{E_{y_0}}\frac{E_z}{E_{z_0}} = sen^2\Delta \quad (2.21)$$

De esta última expresión podemos distinguir tres casos, que serán los tres tipos de polarización de onda que exista:

- **Polarización lineal:** Cuando tenemos $\Delta = 0$ o bien $\Delta = \pi$ obtenemos que la ecuación (2.21) tiene la siguiente forma:

$$\frac{E_y}{E_{y_0}} = \pm\frac{E_z}{E_{z_0}} \quad (2.22)$$

Por lo que si las componentes 'y' y 'z' de el vector $\vec{E}$ son proporcionales, la punta de la onda describirá una recta en el plano YZ, que es el perpendicular al de propagación. Gráficamente se puede ver del siguiente modo:

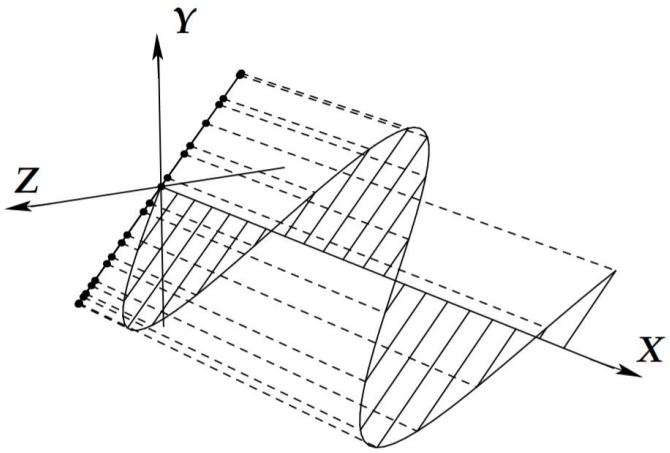


Ilustración 2.4.2 Polarización lineal

En la ilustración 2.4.2, referenciada en [1], se puede ver la pendiente de la recta trazada en el plano YZ, que variará dependiendo de las magnitudes que adopten las dos componentes del vector $\vec{E}$.

- **Polarización circular:** Ahora partimos de las premisas de que $\Delta = \pm\frac{\pi}{2}$ y $E_{y_0} = E_{z_0}$ obteniendo en la ecuación (2.21):

$$\frac{E_y^2}{E_{y_0}^{\ 2}} + \frac{E_z^2}{E_{z_0}^{\ 2}} = 1 \quad (2.23)$$

Este tipo de ecuación vemos rápidamente que corresponde a una circunferencia, dibujando la punta de la onda tal figura sobre el plano YZ a lo largo del tiempo:

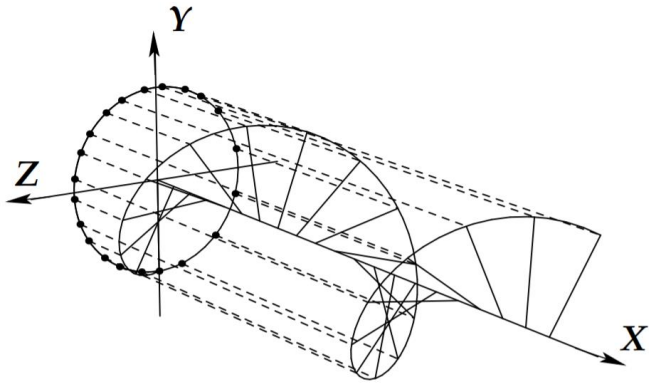


Ilustración 2.4.3 Polarización circular

Ilustración 2.4.3 referenciada en [1].

- **Polarización elíptica:** Por último, este caso corresponde a cuando el vector describe una elipse, viendo la onda polarizada gráficamente de la siguiente forma:

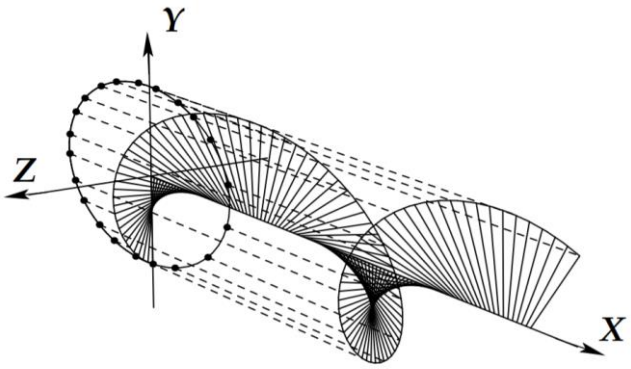


Ilustración 2.4.4 Polarización elíptica

Ilustración 2.4.4 referenciada en [1].

2.5. Materiales Absorbentes al Radar (RAM)

Como indica el título de este trabajo, el modelado geométrico que creemos serán una serie de superficies recubiertas de material absorbentes, donde utilizaremos para ello una serie de elementos resistivos distribuidos según unos parámetros estadísticos, por lo que será conveniente entender qué son este tipo de materiales.

Este tipo de materiales nace de la idea en los años 30 del siglo pasado de tratar de que los aviones o barcos pudiesen evitar a los radares enemigos. Es por eso que se trató de cubrir estos equipos con una serie de materiales que poseen unas propiedades en específico para poder absorber en la mayor medida posible este tipo de ondas. Estos materiales son conocidos como Materiales Absorbentes al Radar, o en inglés Radar Absorbing Materials (RAM).

Estos materiales lo que consiguen es atenuar la energía que contiene una onda electromagnética a partir de las propiedades del propio material y de la estructura en la que se encuentran dispuestos.

Las aplicaciones directas de estos materiales pueden ser muy extensas, las más evidentes pueden ser, como se ha comentado anteriormente, aplicaciones en el plano militar, de forma que los diferentes equipos no puedan ser detectados al radar, pero también este tipo de materiales puede utilizarse para eliminar la radiación exterior que pueda causar una interferencia en el equipo de una aplicación.

Así pues, los principales usos que tienen son los siguientes:

- Mejorar patrones de antenas.
- Reducir los reflejos no deseados de objetos y dispositivos.
- Cubrir el interior de salas de ensayo (cámaras anecoicas) con el fin de lograr condiciones de “espacio libre” para la medición de los componentes y sistemas.
- Reducir el RCS (sección transversal radar) de objetos.

- Mejorar la protección de recintos y recipientes a través de su uso como materiales de junta.

3. NewFASANT

En este capítulo se entrará en detalle sobre la herramienta de diseño asistido y de simulación *NewFASANT*, explicando una descripción de este y su funcionamiento, ya que este será la herramienta con la que simularemos en este trabajo.

3.1. Contexto

Desde el año 1995, el Grupo de Electromagnetismo Computacional (GEC) de la Universidad de Alcalá, ha estado trabajando en el desarrollo de un software dedicado a la simulación electromagnética, para que, de esta forma, se ahorrasen costes de fabricación o adquisición de componentes y en cámaras anecoicas.

Es por eso, que, en 2010, se crea la empresa *NewFASANT S.L.*, promovida por Felipe Cátedra (catedrático del departamento de Ciencias de la Computación) y participada en un 5% por la Universidad de Alcalá. Esta empresa con sede en el Parque Científico-Tecnológico de Castilla La Mancha (Guadalajara), se encarga en desarrollar y comercializar la herramienta software *NewFASANT*, la cual es una herramienta de simulación de campos electromagnéticos para el análisis de antenas, RCS (Radar Cross Section), diagramas de radiación, sistemas de telecomunicaciones avanzados y muchas más aplicaciones.

El equipo de *NewFASANT* es reconocido por ser pionero en la aplicación de técnicas de alta frecuencia, trazado de rayos y soluciones basadas en el Método de los Momentos (el cual será nuestro método de simulación) usando representaciones reales de geometrías mediante el uso de modelos creados por superficies NURBS (Non-Uniform Rational B-Spline).

Algunos de los módulos de simulación de *NewFASANT* son los siguientes:

- **GTD:** este módulo, cuyas siglas proceden del término Geometrical Theory of Diffraction (Teoría Geométrica de la difracción), se basa en la aplicación de la teoría que le da el nombre, y también en la Óptica Geométrica (GO) y Óptica Física (PO), teniendo como objetivo acelerar el trazado de rayos. GTD es una herramienta software muy eficiente en el plano de análisis de antenas embarcadas en un contexto de satélites, aviones u otros modelos más complejos.
- **MoM:** como dice su nombre, esta herramienta se basa en el método de simulación Método de los Momentos, y es un software que tiene como objetivo el análisis en tres dimensiones de antenas complejas, antenas embarcadas, compatibilidad electromagnética y sección radar (RCS), permitiendo así el diseño eficiente de antenas y otras estructuras.
- **PO:** proviene del acrónimo Physical Optics, que en español se traduciría a Óptica Física, y es un módulo de análisis de sección radar de objetos eléctricamente complejos. Es una herramienta que soporta computación paralelizada,

obteniendo una mayor rapidez a la hora de producir resultados, lo que será de gran utilidad a la hora de analizar grandes objetos eléctricamente complejos.

- **MONCROS:** este último módulo de simulación, el cual será el que vayamos a emplear en este proyecto, está basado en el Método de los Momentos junto al Método Rápido de los Multipolos Multinivel (FMLMP), y es una herramienta software paralelizada para la obtención de la sección radar.

Estos módulos de simulación son algunos de los que ofrece *NewFASANT* para la simulación electromagnética de distintos tipos de geometría, siendo el módulo MONCROS el que se emplee en este proyecto y se explique con mayor profundidad posteriormente.

3.2. Software de simulación NewFASANT

En este apartado se explicará más en detalle el funcionamiento de la herramienta *NewFASANT*, viendo de una forma más amplia la herramienta con la que se va a desarrollar parte de este proyecto y poder tener una mejor comprensión de esta para futuros apartados.

NewFASANT es una herramienta que, como se ha explicado previamente y explica perfectamente José Luis García Díaz en su Trabajo Fin de Grado "Diseño e implementación de un sistema de simulación electromagnética en remoto aplicado al análisis de antenas" [9], nace de la necesidad de emplear un software de simulación y análisis de sistemas de telecomunicaciones dada su complejidad, evitando de este modo pérdidas de tiempo y dinero en complicados prototipos. Es por eso por lo que su funcionamiento sea algo complejo pero muy efectivo a la hora de conseguir lo planteado.

Esta herramienta se compone de una interfaz de usuario, realizada en Java, y de unos núcleos de simulación y mallador desarrollados en FORTRAN que son los encargados de llevará a cabo la simulación y generar los resultados.

Esta interfaz de usuario será donde se vayan a realizar los pasos necesarios para la correcta ejecución de los núcleos de simulación, es decir, que será donde el usuario configure los parámetros de simulación y de observación dentro del producto y módulo que esté utilizando. Así pues, el proceso para hacer una simulación será el siguiente:

i. El usuario creará un nuevo proyecto donde deberá especificar el módulo de simulación.

ii. Se creará el escenario que se vaya a simular. Para ello se deberá realizar el modelado geométrico, que, en nuestro caso, será lo que vayamos a desempañar con ayuda de nuestra interfaz de usuario creada previamente.

iii. Se definen los parámetros de simulación y de observación. Los primeros, serán la frecuencia, tipo de simulación y otras opciones específicas de nuestro módulo de simulación (que se verán posteriormente), y los

segundos, serán los que vayan a determinar los puntos del espacio en los que se calculen los valores del campo.

iv. Una vez se haya especificado los parámetros del punto anterior, se procede a realizar el mallado, que es el proceso por el cual se toma la geometría generada y se procesa, dividiendo las superficies en pequeñas formas más sencillas como triángulos o cuadriláteros, para poder realizar los cálculos con los algoritmos necesarios.

v. Se ejecuta la simulación por parte del núcleo de simulación correspondiente.

vi. Se visualiza los resultados de la simulación, pudiendo hacerlo en el log de la simulación, es decir, en el código que ha creado el programa de los cálculos realizados, o de forma intuitiva a través de gráficas.

Una vez visto el modo de funcionamiento a la hora de realizar las simulaciones en *NewFASANT*, veremos cómo serán estos pasos en el módulo de simulación MONCROS, el cual es utilizado para el desarrollo de este proyecto.

3.3. Módulo de simulación MONCROS

Como ya se ha visto antes, MONCROS es un módulo de simulación basado en el Método de los Momentos junto al Método Rápido de los Multipolos Multinivel (FMLMP), y es una herramienta software paralelizada para la obtención de la sección radar.

Una de las razones por la que se ha utilizado esta herramienta y no otras, es que nuestras simulaciones requerían de una alimentación por onda plana, es decir, como son la onda proviniese de una distancia suficientemente lejana para que se deje de considerar esférica y pase a ser plana, por lo que podemos usar este módulo que su alimentación es por onda plana siempre.

Su método de simulación hemos visto que es el Método de los Momentos, y la razón de esto es que el tipo de método de simulación depende del tamaño eléctrico de la geometría.

El tamaño eléctrico es el tamaño referido a longitudes de onda. La longitud de onda depende inversamente de la frecuencia, según lambda = c/f. A partir de unas 10 longitudes de onda, y siempre que la superficie sea suave se pueden aplicar métodos asintóticos como GTD (Geometrical Theory of Diffraction), basado en trazado de rayos, o como PO (Physical Optics).

Si las superficies son eléctricamente pequeñas, o no son suaves, o se quiere tener mucha más precisión se debe recurrir a métodos rigurosos como el mencionado antes MoM, donde se establece un sistema de ecuaciones donde el número de incógnitas depende linealmente del número de subdominios resultante del mallado. Este sistema si es relativamente pequeño, digamos hasta unas 20000 incógnitas, se puede resolver de forma directa, pero no es posible hacerlo así para problemas más grandes, por ejemplo, de 1M de incógnitas. En esos casos hay que recurrir a solver iterativos como el GMRES

(Generalized Minimal Residual Method) o variantes del CGM (Conjugate Gradient Method).

Una vez explicado su método de simulación veremos cada uno de los parámetros de simulación que componen este módulo y sus características:

Simulation menu:

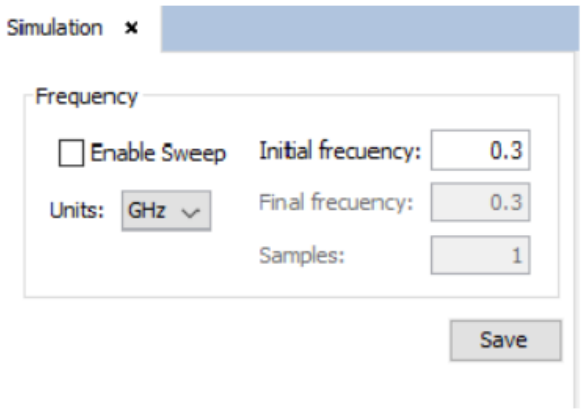


Ilustración 3.3.1 Menú de simulación

En este menú se configurará la frecuencia de simulación. Esto puede ser en una única frecuencia o también en un barrido de frecuencias, donde se nos permite introducir los valores del intervalo de frecuencias y las muestras a simular dentro del barrido.

Solver:

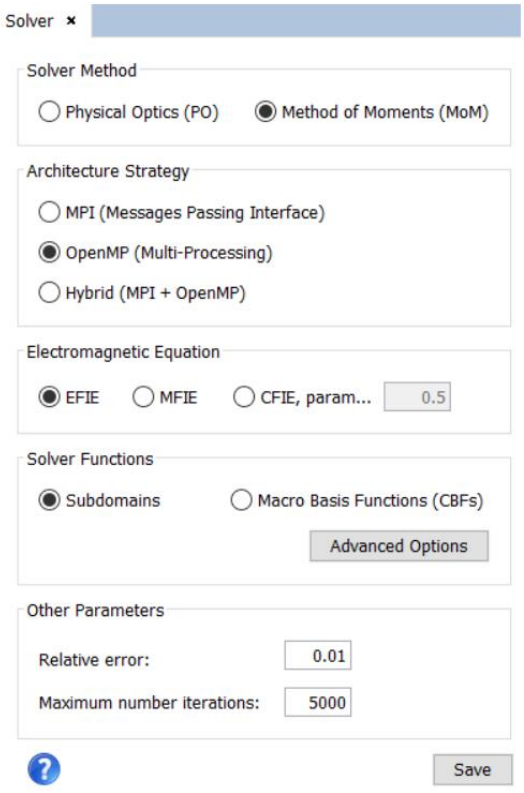


Ilustración 3.3.2 Solver

Este menú contiene las opciones de comportamiento del 'solver', es decir, aquí se definirán los parámetros para el cálculo de las incógnitas de los valores de campo de la simulación. En nuestras simulaciones los parámetros no cambiarán de valor salvo en caso de convergencia de los resultados de simulación, donde se aumentará el valor del error relativo para ser menos restrictivo a la hora de calcular los resultados.

RCS:

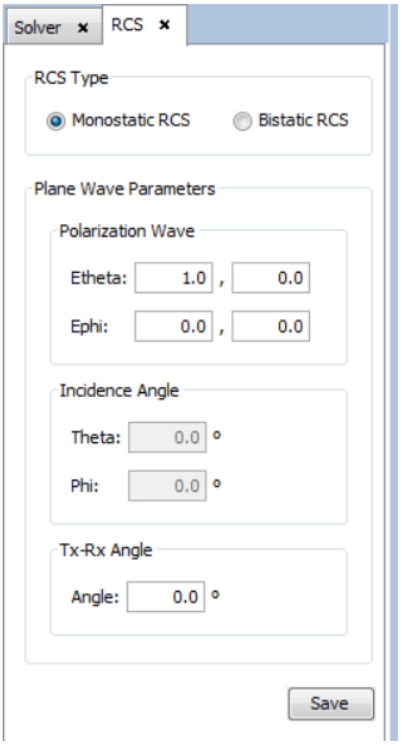


Ilustración 3.3.3 RCS

En el menú RCS, se definirá primero el tipo de seción radar (monostático o biestático), y después con Etheta y Ephi los valores de polarización de la onda plana. Cuando RCS es monostático se habilitará el valor del ángulo Tx-Rx, y cuando es biestático se habilitará el ángulo de incidencia.

Output menu:

Output menú, o en español menú de salida, nos da la posibilidad de modificar las direcciones de observación como aparece en la siguiente imagen:

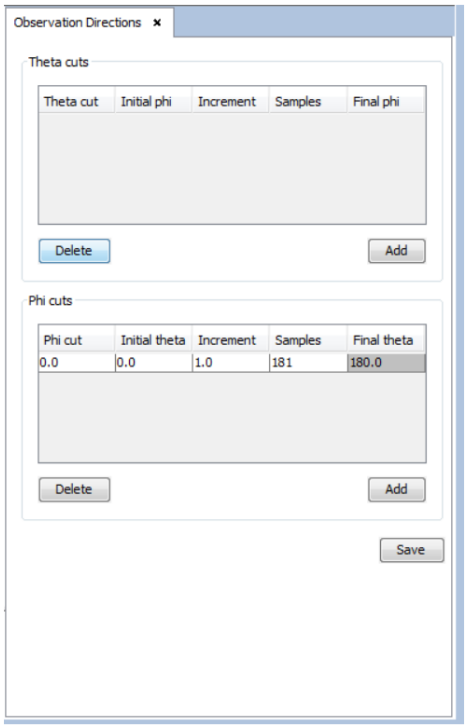


Ilustración 3.3.4 Direcciones de observación

Los parámetros que podremos definir de las direcciones de observación son las coordenadas de estas, que son coordenadas polares. Para ello fijaremos primero un corte el ángulo en 'phi', para después definir un barrido de observación desde un ángulo 'theta' inicial hasta otro ángulo 'theta' final, definiendo también los saltos de ángulo del barrido. Por ejemplo, por defecto nos encontramos un barrido en 'theta' de 0 a 180º, con saltos de 1º y corte en 'phi' en 0.

Meshing:

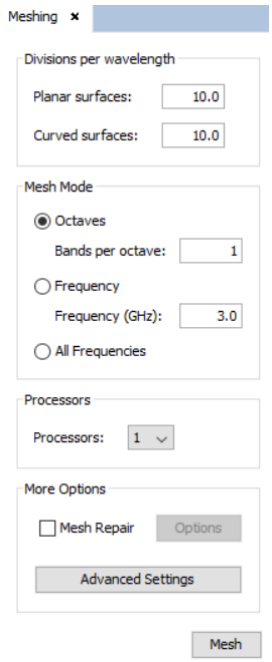


Ilustración 3.3.5 Mallado

Meshing, tal y como dice su nombre, es el menú donde se configurará el mallado para el cálculo de los valores de campo. En la imagen podemos ver que tenemos en la parte superior el parámetro de las divisiones por longitud de onda en superficies planas y curvas, donde mantendremos ese valor de 10, después el modo de mallado que en nuestro caso es el que marca por defecto (octavos) y por último el número de procesadores, que depende de la computadora donde nos encontramos, que en nuestro caso es de 16.

Calculate:

En este apartado, una vez se hayan configurado todos los parámetros de simulación deseados, tenemos la opción de ejecutar la simulación, con el número de procesadores que habíamos marcado previamente en el cálculo del mallado. Una vez ejecutado aparecerá el siguiente panel:

Ilustración 3.3.6 Log de simulación

En este panel, el programa muestra el registro de proceso de la simulación en ejecución. También ofrece la posibilidad de cancelar la simulación y guardar el contenido del registro del proceso en un archivo de texto.

Show results:

Ya ejecutada la simulación, se podrá entrar en este apartado para observar los resultados de forma gráfica. En nuestro caso, solo nos serán útiles las gráficas de corte por frecuencia y por ángulo.

4. Tecnologías utilizadas

En este apartado, se hará constancia de las tecnologías y las herramientas software utilizadas para el desarrollo de este proyecto, así como los motivos que han llevado a escoger estas tecnologías.

4.1. Lenguajes de script

De manera introductoria y para contextualizar los siguientes puntos, hablaremos de lo que son los lenguajes de script y su funcionalidad. Los lenguajes de script son pequeños lenguajes de programación, en los cuales se ejecuta el código situándolo dentro de un determinado contexto. La principal característica que tienen estos es que se ejecutan comando a comando siguiendo las instrucciones paso a paso.

Por lo tanto, podemos considerar a un script como un código fuente que se encuentre escrito en algún tipo de lenguaje interpretado. Estos lenguajes interpretados serán lenguajes de programación, siendo los más importantes y más conocidos Fortran, C, C++, BASIC, G, Visual Basic, Ada, ALGOL, D, COBOL, GO, Lisp o Pascal. Este código se podrá escribir dentro de cualquier tipo de editor de texto, o usando algún tipo de entorno de programación, como en nuestro caso, *MATLAB.*

Un paso importante para que la máquina pueda interpretar lo que se vaya a escribir dentro de un script, es que cuando se haya escrito el código fuente en un lenguaje de programación cualquiera, se compile posteriormente. De este modo, podremos pasar las órdenes que queremos que ejecute la máquina, que se encuentran en un lenguaje de alto nivel, a un lenguaje de más bajo nivel que ella sí que es capaz de entender, es decir, a un lenguaje máquina o binario.

Realmente, los scripts con los cuales estamos habituados a manejarnos, y que también veremos en este proyecto, suelen usar lenguajes interpretados como Python o JavaScript, siendo así el fichero donde se encuentre escrito el script, una serie de órdenes empaquetados por lotes. Al ser estos, como hemos dicho, lenguajes interpretados, no sería necesario el paso de compilar el código ya que se puede ejecutar de forma que lo entienda la máquina con ayuda del intérprete.

Con estos lenguajes de script, crearemos la herramienta para la generación del modelado geométrico que posteriormente se simulará en *NewFASANT*

4.2. MATLAB

Para el proyecto, el entorno donde se realizará la aplicación será *MATLAB. MATLAB* (abreviatura de “MAtriz LABoratory”) es un programa para realizar cálculos numéricos con vectores y matrices, y por tanto podremos trabajar en él con números escalares tanto reales como complejos, cadena de caracteres y otras estructuras de información más complejas.

La principal característica que tiene *MATLAB* como programa es que es al mismo tiempo un lenguaje y un entorno de programación, permitiendo la construcción de nuestras propias herramientas reutilizables. Esto nos permite crear nuestras propias funciones y programas en código *MATLAB*, el cual es un tipo de lenguaje de programación único para este programa que tiene cierta semejanza a los lenguajes que ya estamos habituados como puede ser Fortran o C. Esto nos permitirá como usuarios el escribir nuestros propios scripts.

Los scripts en *MATLAB* son una serie de comandos que se encuentran escritos en un fichero de texto, los cuales se podrán ejecutar con una orden, tal y como habíamos visto antes, por lo que, en nuestro caso, en *MATLAB* servirán para poder resolver problemas en concreto y también crear nuevas funciones.

En *MATLAB*, tendremos que diferenciar la zona de trabajo donde crearemos la función necesaria para la creación del modelado geométrico, y la interfaz de usuario.

4.2.1. Entorno de trabajo

El entorno de trabajo de *MATLAB* será donde creemos nuestra función de la aplicación, y lo vemos de la siguiente forma:

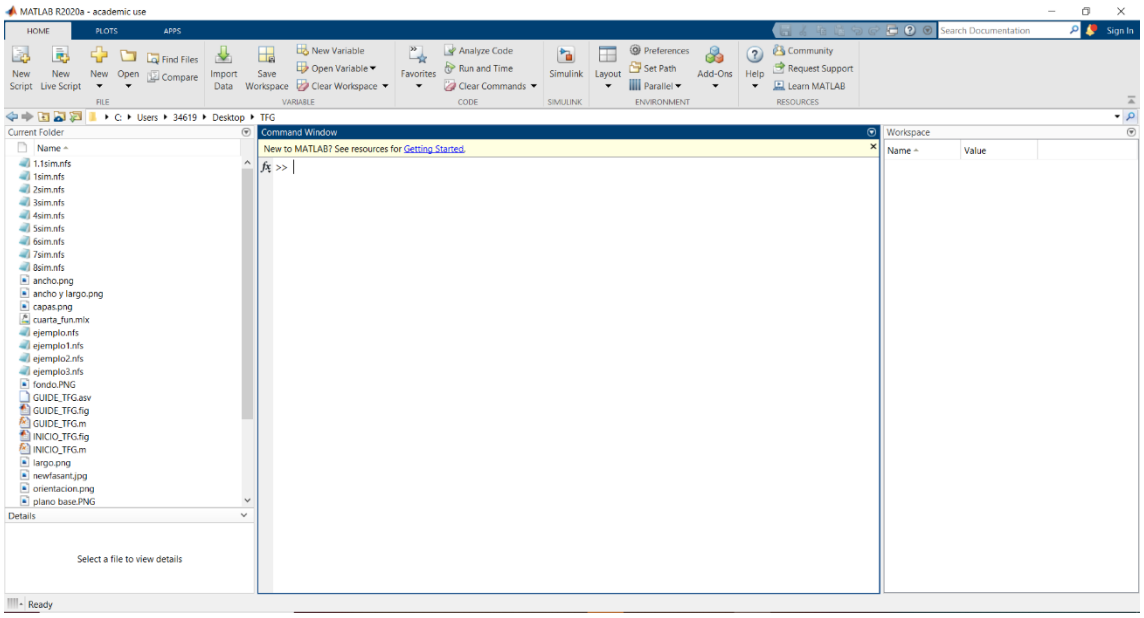


Ilustración 4.2.1 Entorno de trabajo MATLAB

Si vemos la imagen superior, la ventana de comandos es donde podremos introducir las distintas expresiones para que *MATLAB* las evalúe. Esta zona será para realizar cálculos sencillos, pero donde realizaremos nuestra función será en un script, donde podamos escribir todos los comandos de manera seguida para que el programa los ejecute de forma seguida.

La otra parte del programa que usaremos, que es la que verá el usuario cuando haga uso de la aplicación, será la interfaz de usuario.

4.2.2. Interfaz de usuario (GUI)

Las GUI (también conocidas como interfaces gráficas de usuario o interfaces de usuario) permiten un control sencillo (con uso de ratón) de las aplicaciones de software, lo cual elimina la necesidad de aprender un lenguaje y escribir comandos a fin de ejecutar una aplicación.

Las apps de *MATLAB* son programas autónomos de *MATLAB* con un frontal gráfico de usuario GUI que automatizan una tarea o un cálculo. Por lo general, la GUI incluye controles tales como menús, barras de herramientas, botones y controles deslizantes. También es posible crear apps personalizadas propias, incluidas las interfaces de usuario correspondientes, para que otras personas las utilicen.

Donde creemos la interfaz de usuario en *MATLAB* será en la siguiente ventana:

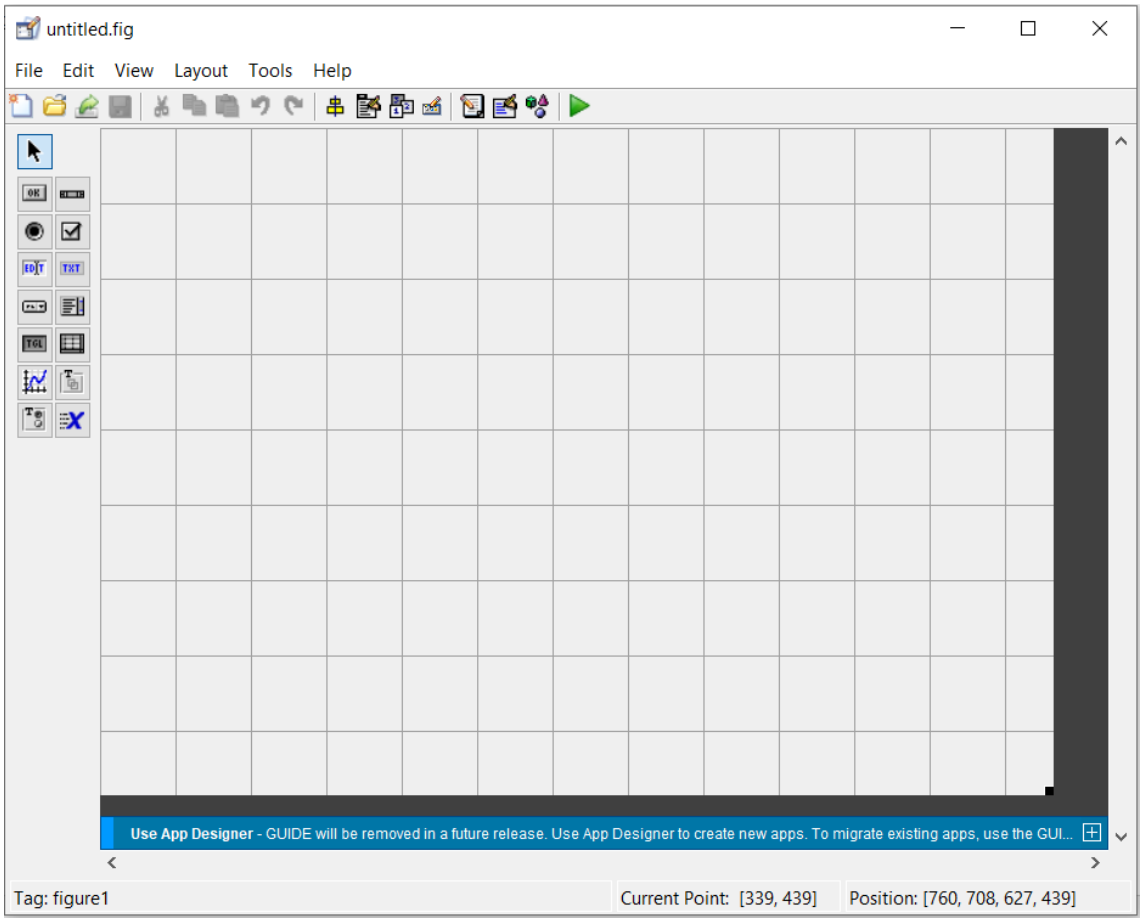


Ilustración 4.2.2 GUIDE MATLAB

En la parte izquierda de la ventana, podemos ver los elementos mencionados anteriormente para la creación de la interfaz, los menús, Toolbox, Check Box, etc. Los elementos utilizados en nuestra aplicación son los siguientes:

Push Button:

Ilustración 4.2.3 Push Button

Los Push Button son elementos que, como se indica, sirven como botón. A este elemento se le puede asignar una acción cuando se pulsa sobre él, en nuestra aplicación puede por ejemplo servir para realizar la acción de generar el fichero con los comandos de nuestra geometría o para ofrecer un panel de ayuda.

Edit Text:

Ilustración 4.2.4 Edit Text

Un Edit Text es un recuadro donde el usuario será capaz de insertar cualquier tipo de dato que quiera, numérico o una cadena de caracteres, lo que no será útil a la hora de introducir los datos necesarios para dimensionar nuestra geometría.

Static Text:

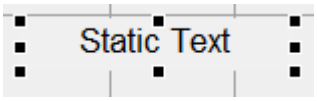


Ilustración 4.2.5 Static Text

Este componente es un recuadro que contiene un texto que no es modificable por parte del usuario, por lo que es un buen elemento para poner títulos.

Check Box:

Ilustración 4.2.6 Check Box

Un Check Box es básicamente una casilla donde el usuario podrá marcarla o no, teniendo únicamente esos dos estados, el de seleccionado o deseleccionado. Esto será útil para recolectar información sencilla, al estilo sí/no, por lo que en nuestra aplicación puede tener distintos usos.

Axes:

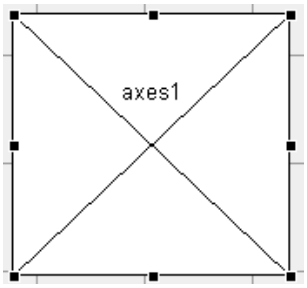


Ilustración 4.2.7 Axes

Por último, tenemos el elemento Axes. Este es un componente cuya principal utilidad es la de mostrar algún tipo de gráfica que sobre unos ejes que se defina en el código del programa, pero en nuestro caso, se ha utilizado este elemento en la aplicación para mostrar imágenes que sirvan de ayuda gráfica para el usuario a la hora de modelar los parámetros de la geometría que quiera crear.

Una vez que se haya creado la ventana gráfica de nuestra interfaz de usuario, se crea el siguiente script:

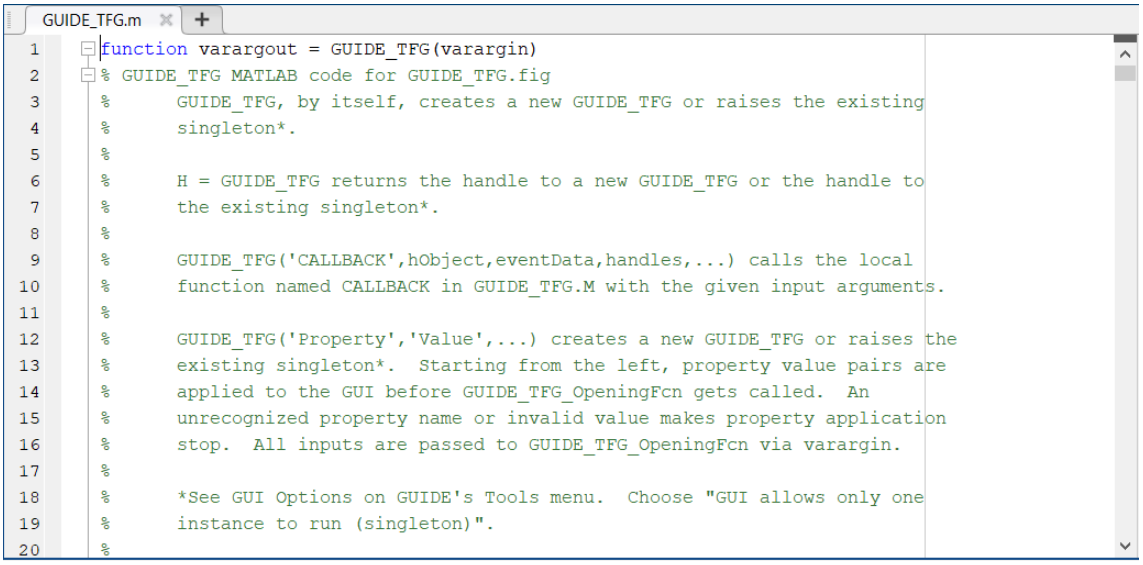


Ilustración 4.2.8 Script asociado a la interfaz de usuario

En este script será donde se encuentren los Callbacks de cada uno de los elementos mencionados previamente, los cuales son las funciones que al ejecutarlas permiten asociar nuestros objetos gráficos con la acción que desee el usuario, por lo que, una vez configurado correctamente este script, podremos utilizar nuestra aplicación de la manera deseada.

En este proyecto, la opción escogida ha sido *MATLAB*, ya que al ser un entorno que permitía generar una interfaz gráfica de usuario y es la herramienta de mayor uso en ingeniería industrial, pues suponía el elemento más cómodo con el que podía trabajar.

4.3. Diagramas UML (Diagramas de casos de uso)

Otra de las tecnologías utilizadas en este proyecto con el fin de describir los resultados que hayamos obtenido en el uso de la aplicación de manera gráfica serán los diagramas UML. UML proviene de Unified Modeling Language y, tal como dicen sus siglas, es un lenguaje de modelado de sistemas de software.

El UML se encuentra compuesto por diferentes elementos gráficos, consiguiendo diagramas combinándolos entre sí. Como es un tipo de lenguaje, va a contar con una serie de reglas a la hora de realizar dichos diagramas.

Estos diagramas tienen como finalidad mostrar distintos aspectos de un sistema, conociéndose estos como modelos. Los modelos son representaciones simples de una realidad, es decir, que un modelo UML lo que hará será describir lo que vaya a hacer un sistema en principio, pero sin mostrar como se implementa dicho sistema.

Existen varios tipos de diagramas UML (diagramas de clases, de objetos, de estados, etc.), pero de los que se harán uso en este proyecto serán los diagramas de caso de uso.

Los casos de uso son una descripción de cómo se ven las acciones de un sistema desde la perspectiva del usuario. Este tipo de diagramas consigue modelar un sistema completo utilizando los ya mencionados casos de uso y actores. Los casos de uso serán las funciones o servicios que suministre el sistema para el usuario.

Sus componentes principales son:

- **Sujeto:** sistema a modelar (en nuestro caso sería la aplicación).
- **Casos de uso:** unidades funcionales completas.
- **Actores:** entidades externas que interactúan con el sistema (en nuestro caso sería el usuario que utiliza la aplicación).

Podemos ver estos componentes en el siguiente ejemplo:

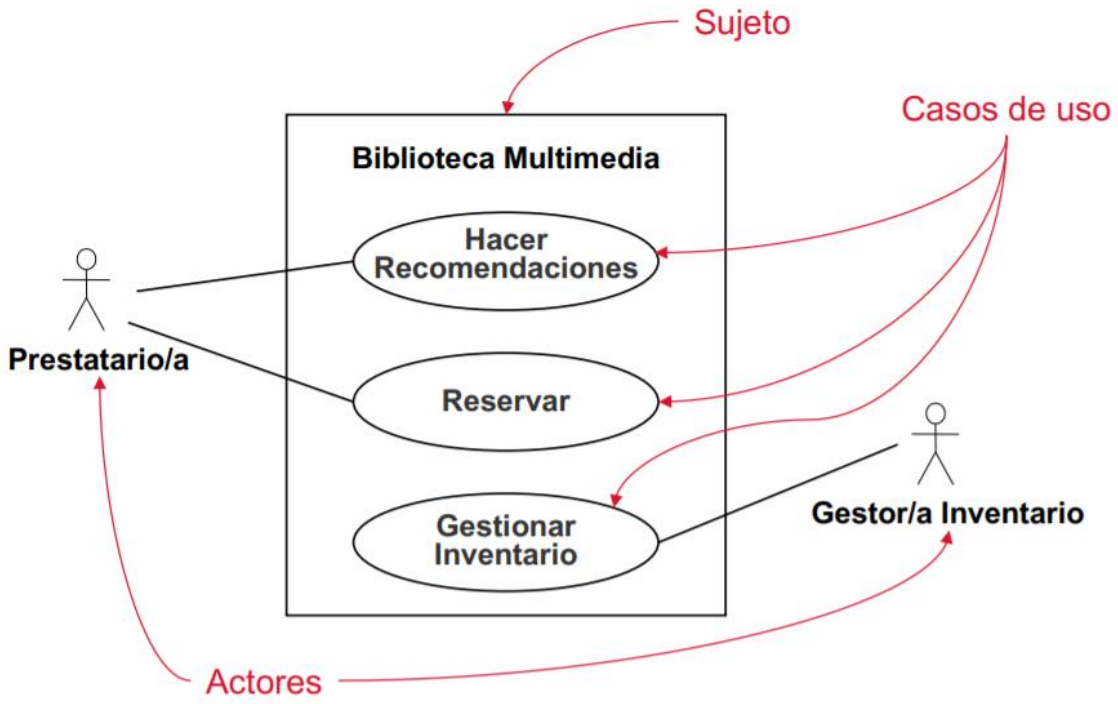


Ilustración 4.3.1 Ejemplo diagrama de casos de uso

En la ilustración 4.3.1, referenciada en [10], se puede ver un sencillo ejemplo, donde se ve el funcionamiento de un sistema de biblioteca multimedia, tenemos a dos actores que serían el prestatario y el gestor, un sujeto que sería la biblioteca multimedia, la cual se muestra como una caja negra que proporciona los casos de uso, y, por último, tres casos de uso que tienen asociaciones con los dos actores y ejemplifican un conjunto de acciones realizadas por el sistema, que dan lugar a un resultado observable.

Las características que tendrán los casos de uso son que siempre comenzarán por un actor, proporcionando valores a los actores, y poseyendo una funcionalidad completa. Se representarán con el nombre del caso de uso dentro o fuera de una elipse

(normalmente se encontrarán dentro como en el ejemplo), o con una pequeña elipse localizada en la esquina superior derecha que sirva como símbolo clasificador. Los casos de uso podrán tener asociaciones y dependencias con otro tipo de clasificadores:

- Entre actores y casos de uso.
    - Asociación: servirá como línea de comunicación entre un actor y un caso de uso donde participa.
- Entre casos de uso.
    - Generalización: esta relación existe entre un caso de uso general y un caso de uso más específico, adquiriendo y añadiendo propiedades al caso de uso base.
    - Inclusión: inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción.
    - Extensión: inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él.
- Entre caso de uso y colaboración.
    - Realización: establece una relación entre el caso de uso y los diagramas que describen la funcionalidad del caso de uso.

La notación que se usaría para estas relaciones es la siguiente:

Asociación	
Generalización	
Inclusión	
Extensión	
Realización	

Tabla 1 Relaciones de casos de uso

Uno de los otros componentes que hemos visto son los actores. Los actores son un tipo de clasificador que representa un rol cuya función es la de interactuar con el sujeto siendo totalmente externo a él. La forma de comunicarse que tendrán los actores con el sujeto será a partir del intercambio de mensajes, siendo en nuestro caso, la introducción de datos para dimensionar los parámetros necesarios de nuestro modelado geométrico.

La forma de representar a los actores será con el icono de “stick man” o “monigote” teniendo el nombre del actor cerca del símbolo, normalmente debajo, empezando este

con mayúscula. Se pueden usar otros símbolos para representar tipos de actores, por ejemplo, para representar actores no humanos.

En la siguiente tabla se muestra la notación que se utilizará para los actores:

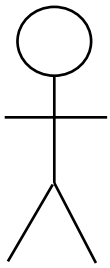
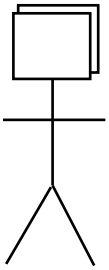
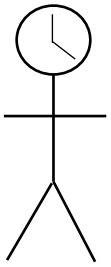
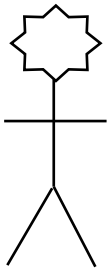
Actor humano	Sistema	Temporizador	Dispositivo
			

Tabla 2 Símbolos para representar tipos de actores

Una vez descritos los diagramas de casos de uso y sus componentes, veremos a lo largo del proyecto como hemos implementado estos de manera práctica.

5. Manual de usuario de la aplicación

5.1. Descarga del proyecto

La herramienta de trabajo para la creación de scripts de procesado geométrico, en mi caso ha sido *MATLAB*. Aquí será donde generaremos la interfaz gráfica para que el usuario pueda introducir los datos, y de esta forma generemos la geometría requerida para posteriormente cargarla dentro del software *NewFASANT*.

Para poder hacer uso de la interfaz gráfica tendremos que descargar el paquete .zip con el código y el resto de los elementos que la componen, descomprimir el paquete en el directorio deseado por el usuario y finalmente, abrir el proyecto en *MATLAB*.

5.2. Pantalla principal

Una vez abierto veremos la siguiente pantalla:

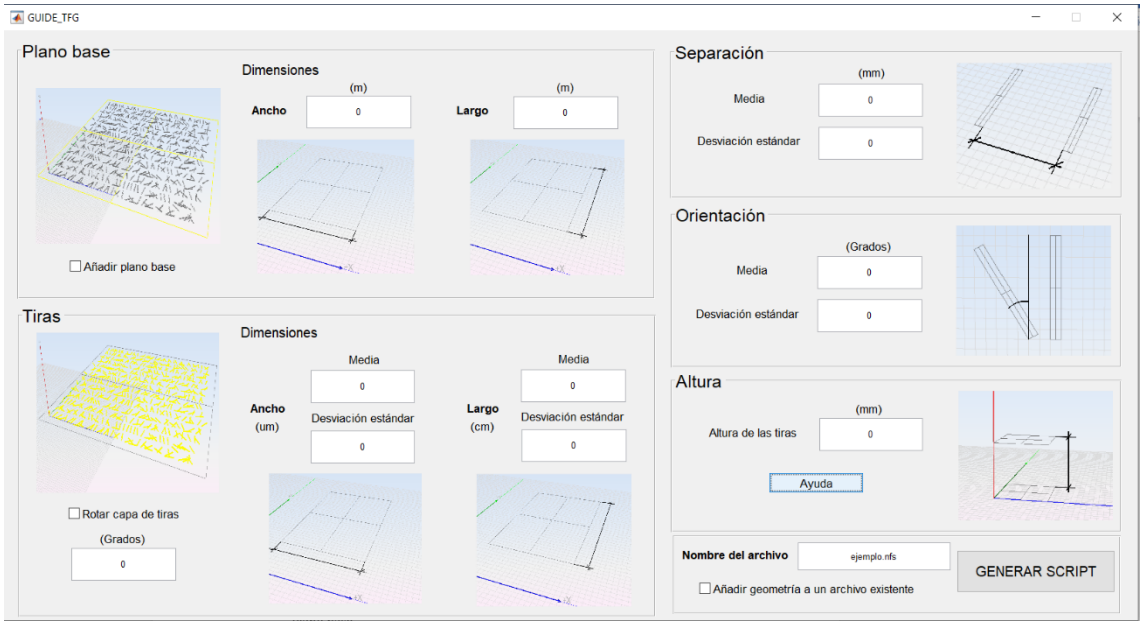


Ilustración 5.2.1 Pantalla principal de la aplicación

Como podemos ver en la imagen, se nos presentan distintas casillas en las que el usuario tiene la posibilidad de introducir distintos datos (donde por defecto se encuentran todos en 0), para la posterior creación de la geometría que se cargará dentro de *NewFASANT*. Estas casillas contienen, a parte de los recuadros donde se deben introducir los valores de cada dato correspondiente para generar la geometría deseada, una serie de imágenes donde se muestra gráficamente que es cada elemento de forma clara.

Las casillas presentes son las siguientes:

5.2.1. Plano base.

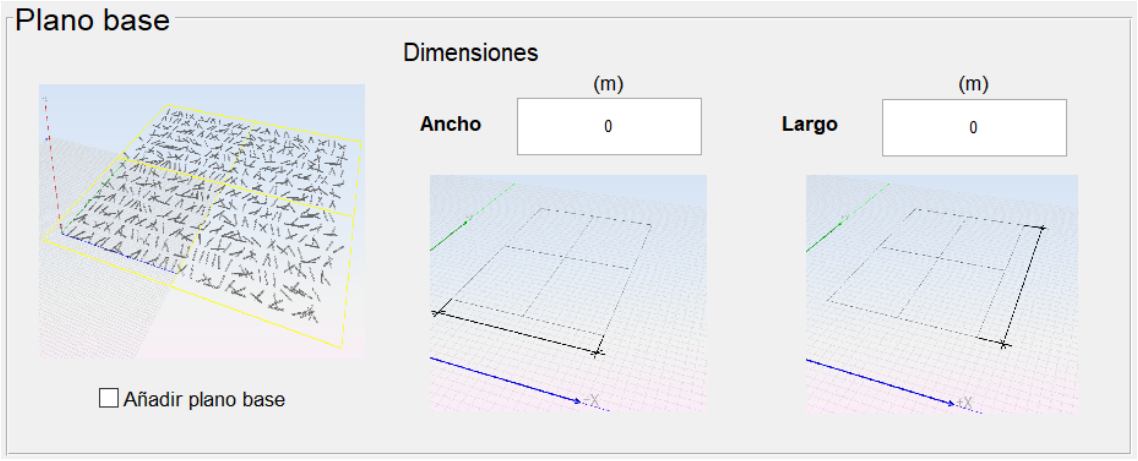


Ilustración 5.2.2 Plano base

En la parte superior izquierda, se nos presenta un recuadro donde el usuario establece las dimensiones de ancho y largo que tendrá el plano base, sobre el que se dispondrán posteriormente todas las tiras.

A parte de esto, en este recuadro también se dispone de un checkbox en la parte inferior izquierda, para que el usuario decida si generar este plano base o no, de forma que para poder generarlo se debe hacer click sobre este. En caso de no querer generar el plano base, el usuario también debe introducir unas dimensiones para el plano base, para indicar así la superficie donde se van a encontrar las tiras que se generarán a continuación.

En caso de querer generar la geometría del plano base, deberemos tener en cuenta que, para evitar que las tiras se salgan de este debido a que las rotemos un determinado ángulo sobre sí mismas, el programa generará el plano con unas dimensiones un poco mayores de las indicadas. Esta longitud será lo que mida el largo de las tiras de media, más tres veces la desviación estándar, de forma que, al generarse las tiras a partir de una distribución normal, si sumamos tres veces la desviación típica será suficiente para que, con toda seguridad, la tira no tenga una medida mayor a esta y no se pueda salir si la rotamos un ángulo muy brusco.

Por último, he de señalar que para entender qué datos vamos a introducir, podemos ver las imágenes que se nos presentan, empezando por la imagen que se encuentra a la izquierda, donde representa lo que va a ser el plano base gráficamente dentro del software *NewFASANT*. Después, a la derecha se nos muestra que va a ser el ancho y el largo del plano, siendo el ancho la longitud (en metros, como se especifica en la parte superior de la casilla) que tiene este a lo largo del eje x y el largo la longitud (también en metros) a lo largo del eje y.

5.2.2. Tiras.

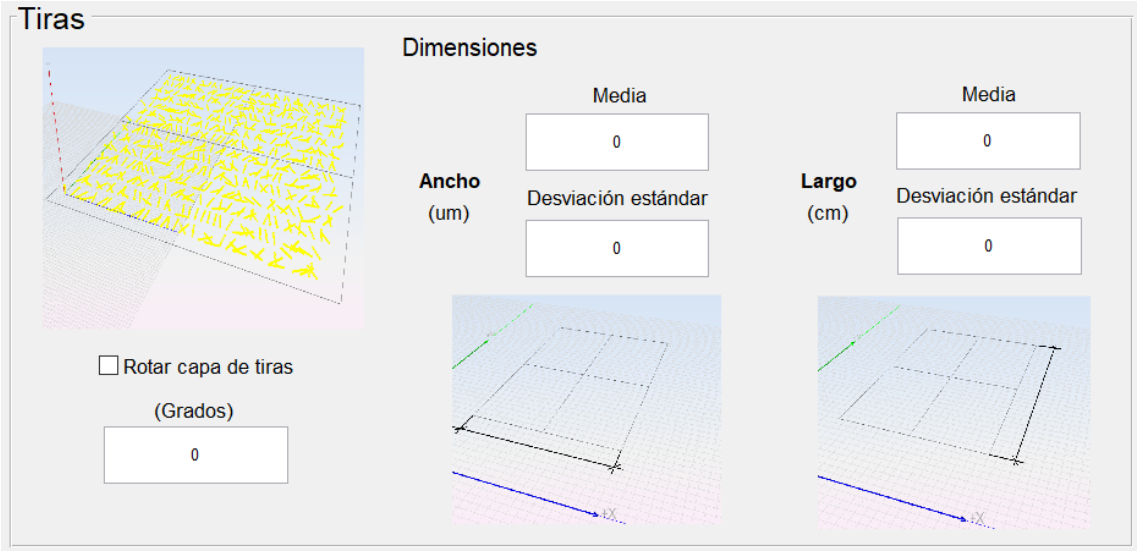


Ilustración 5.2.3 Tiras

En esta casilla, el usuario establece, como antes, las dimensiones que tendrán todas las tiras, pero esta vez lo hará mediante dos parámetros estadísticos (media y desviación típica), de forma que las tiras se generarán con unas dimensiones aleatorias que decidirá el programa a partir de la distribución normal con media y desviación típica indicada. En base a esto, las tiras se irán colocando sobre el plano base (o la superficie donde se encontraría el plano base en caso de que el usuario no quiera generarlo), hasta rellenar por completo toda el área indicada.

Las imágenes expuestas son, a la izquierda, un ejemplo de lo que serían las tiras puestas sobre un plano base, y la derecha, al igual que con el plano base, la representación gráfica de lo que es el ancho y largo del plano. Podemos ver también, que, en esta ocasión, por cómo serán las posteriores simulaciones que hagamos, las tiras tendrán una magnitud de micras en cuanto al ancho y en centímetros en cuanto al largo.

Por último, el usuario encontrará en la esquina inferior izquierda, un checkbox, donde si se pincha en él, se tendrá la posibilidad de rotar la capa de tiras que generemos al completo. Esto puede resultar útil si, por ejemplo, se quieren cruzar dos capas de tiras, de forma que tengamos unas en una orientación determinada, y la otra en una orientación distinta, lo cual hace que se crucen las tiras entre sí y que cada capa sea capaz de filtrar una polarización distinta.

5.2.3. Separación entre tiras.

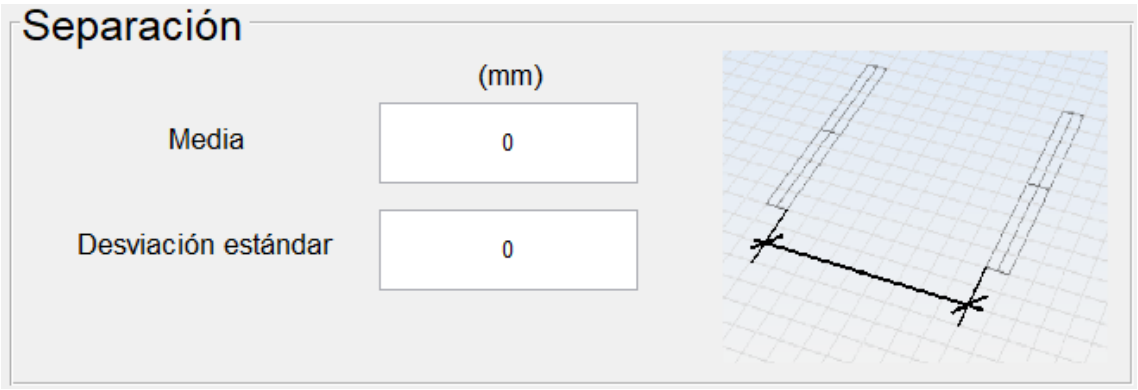


Ilustración 5.2.4 Separación entre tiras

El usuario también dispondrá de un recuadro en el que indique cuanta va a ser la separación que va a querer entre las tiras, de forma que al introducir de nuevo los parámetros estadísticos de la media y la desviación típica, las tiras quedarán separadas de forma aleatoria entre sí.

La separación se puede ver representada en la imagen como el espacio que existe entre una tira y otra, tanto en horizontal como en vertical. Las unidades indicadas en la parte superior de los recuadros para introducir los datos son los milímetros.

5.2.4. Orientación de las tiras.

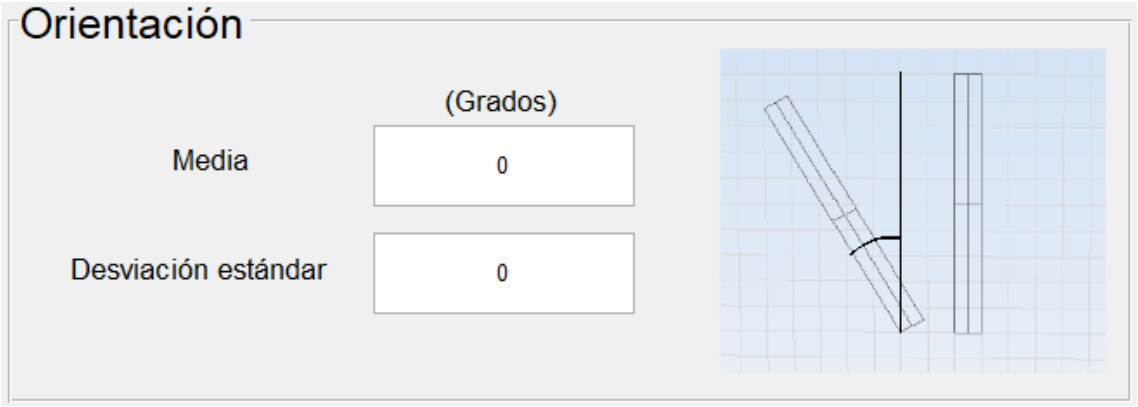


Ilustración 5.2.5 Orientación

Debajo de la separación, tenemos una casilla donde el usuario indicará la orientación que tendrán finalmente las tiras, es decir, los grados que deberán rotar las tiras en su posición final. De nuevo, esto se realizará de forma aleatoria a partir de una distribución normal en el que el usuario indicará la media y la desviación estándar de esta.

En la imagen de la parte derecha, se puede ver gráficamente como rota el plano, de forma que esta rotación será en sentido antihorario cuando el ángulo introducido es positivo y horario cuando es negativo.

5.2.5. Altura de las tiras.

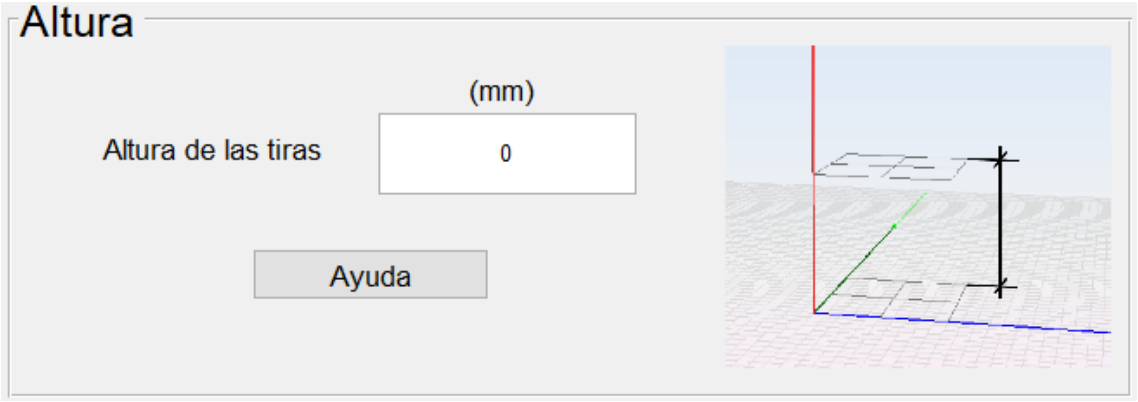


Ilustración 5.2.6 Altura de las tiras

Por último, se puede encontrar una casilla donde se podrá introducir la altura en mm en el eje z, para colocar las tiras. En caso de que el usuario no entienda a que quiere referirse la interfaz gráfica con lo que significa altura de las tiras, puede pulsar el botón de “Ayuda” para que se le muestre el siguiente mensaje:

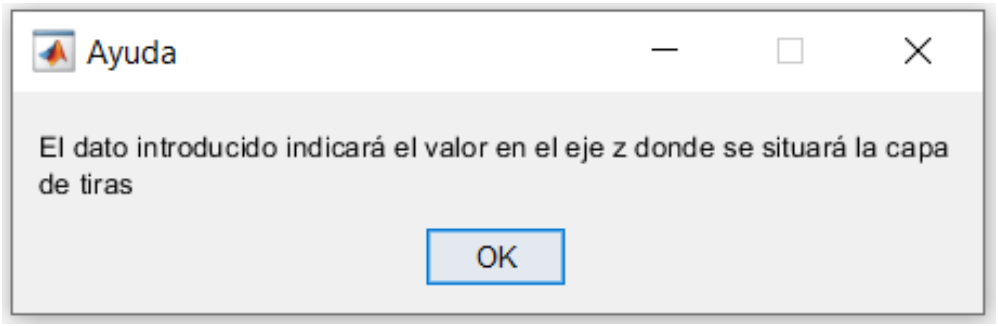


Ilustración 5.2.7 Mensaje de ayuda

Esta funcionalidad puede ser útil para el usuario en caso de querer poner varias capas de tiras, una encima de otra.

5.2.6. Generación de script

Una vez ya se han introducido los datos deseados, vamos al último recuadro que existe en la interfaz gráfica que es el siguiente:

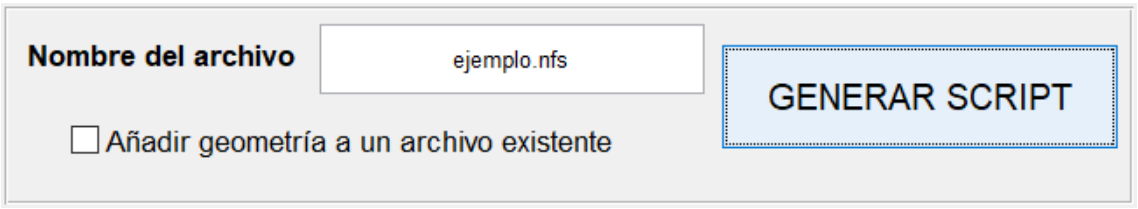


Ilustración 5.2.8 Generar script

Como se puede ver, pulsando el botón "GENERAR SCRIPT", el programa abrirá un archivo, con el nombre que indiquemos a la derecha, en la dirección donde se encuentre abierto nuestro programa. Por defecto, el nombre del archivo va a ser 'ejemplo.nfs', pero el usuario si lo desea, puede cambiarlo, de forma que pueda crear distintos archivos y no se sobrescriban unos encima de los otros.

En caso de que se desee añadir una nueva geometría a un script que ya hayamos creado, el usuario tendrá que hacer click en la checkbox "Añadir geometría a un archivo existente", que se encuentra debajo del recuadro para escribir el nombre del archivo. De esta forma, el programa no borrará los datos que existan en el archivo indicado, sino que se concatenarán los nuevos comandos debajo de los que ya están. Esto puede ser útil a la hora de, por ejemplo, querer introducir varias capas, de forma que primero pondríamos una capa a una cierta altura para después añadir otra con una altura distinta al archivo donde hemos escrito la primera.

Este archivo contendrá los comandos necesarios para que el programa *NewFASANT* los lea y cree la geometría deseada por el usuario. Es importante que, a la hora de nombrar el archivo, se indique la extensión de archivo que se quiere. Por defecto, el archivo tendrá una extensión '. nfs', pero se puede poner otra, siempre y cuando *NewFASANT* sea capaz de reconocerlo.

Si el usuario intentase generar el archivo sin los datos necesarios para poder crear la geometría, aparecería el siguiente mensaje:

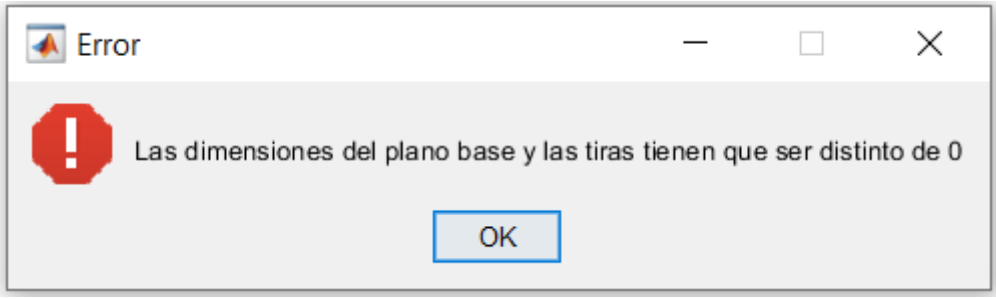


Ilustración 5.2.9 Mensaje de error

De esta forma, el programa informa de que alguna de las dimensiones de ancho o largo del plano base o las tiras es 0, y no se puede crear la geometría. Para las tiras sería 0 en caso de que tanto la media como la desviación típica fuesen 0.

5.3. Generación de script para el modelado geométrico

Una vez indicadas las dimensiones por parte del usuario y pulsando el botón de "GENERAR SCRIPT", se nos crea el archivo con el nombre indicado en la misma dirección en la que nos encontramos con los comandos necesarios. Los comandos que utilizamos para crear la geometría son únicamente dos:

- **Generar plano.**

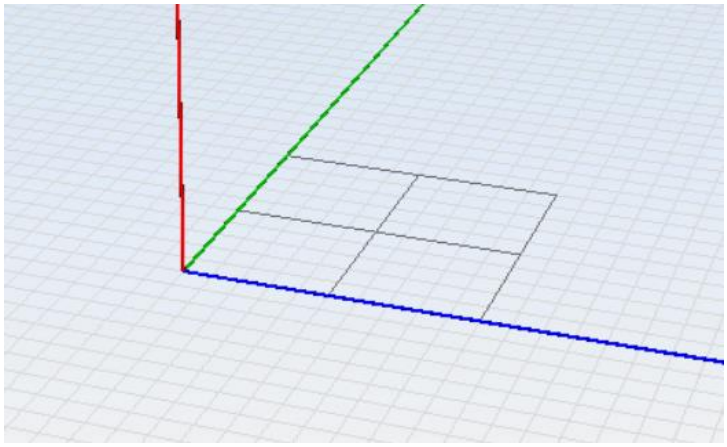


Ilustración 5.3.1 Ejemplo de generación de plano

Para poder crear un plano en *NewFASANT,* se utiliza el comando plane. Su modo de uso es el siguiente:

  - plane -h: El programa muestra el archivo que resume los parámetros para este comando.
  - plane -n <name> -p <point1_x> <point1_y > <point1_z> <width> <depth>: El programa crea un plano definiéndolo desde un punto de inicio en x, y, z, y con una anchura y longitud determinada.

- **Rotar plano.**

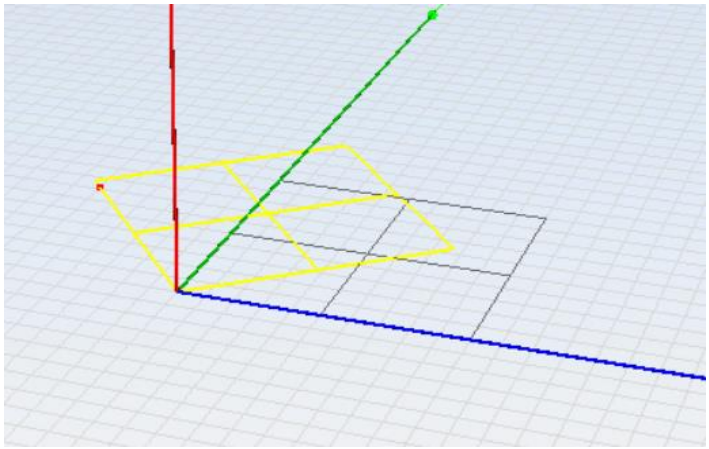


Ilustración 5.3.2 Ejemplo de rotación de plano

Para poder rotar el plano y que tenga una orientación determinada, el comando utilizado es el comando rotate, cuyo modo de uso es:

- rotate -h: El programa muestra el archivo que resume los parámetros para este comando.
- Rotate -s <name> -p <point1_x> <point_y > <point1_z> <point2_x> <point2_y > <point2_z> <angle>: El programa coge dos puntos definidos por el usuario, para poder crear un eje de giro y rotar el plano un cierto ángulo sobre este.

5.3.1. Ejemplo 1

Una vez ya vistos los comandos y los diferentes apartados que componen la pantalla principal de la interfaz de usuario, vamos a ver un ejemplo sencillo donde veamos cómo realiza nuestro programa la geometría deseada.

En este caso introduciremos los siguientes parámetros:

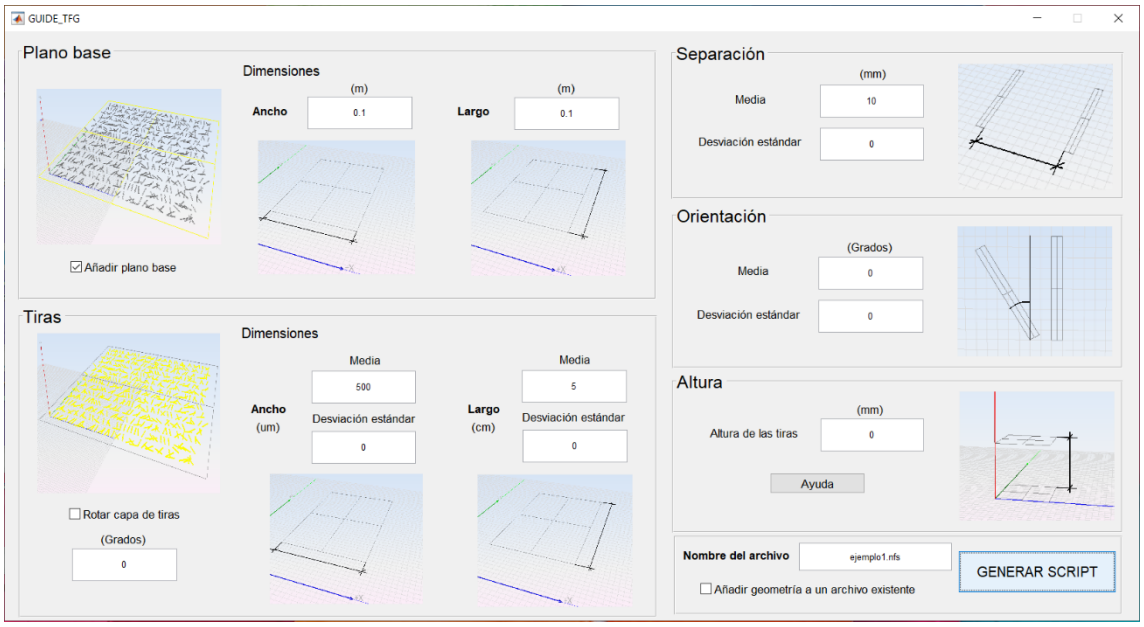


Ilustración 5.3.3 Ejemplo 1, parámetros

Los datos introducidos, como se pueden ver en la imagen superior, son los siguientes:

- **Plano base:**
  - Ancho = 2m
  - Largo = 2m
  - Generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 20cm de media y 0 de desviación estándar.
  - No rotamos capa de tiras.

- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0

- **Orientación:**
  - Media = 0
  - Desviación estándar = 0
- **Altura** = 0
- **Nombre del archivo** = ejemplo1.nfs
- Sin añadir a una geometría existente.

Una vez ya hemos introducido todos los datos, generamos el script, obteniendo el siguiente archivo en el mismo directorio en el que estemos ejecutando el programa:

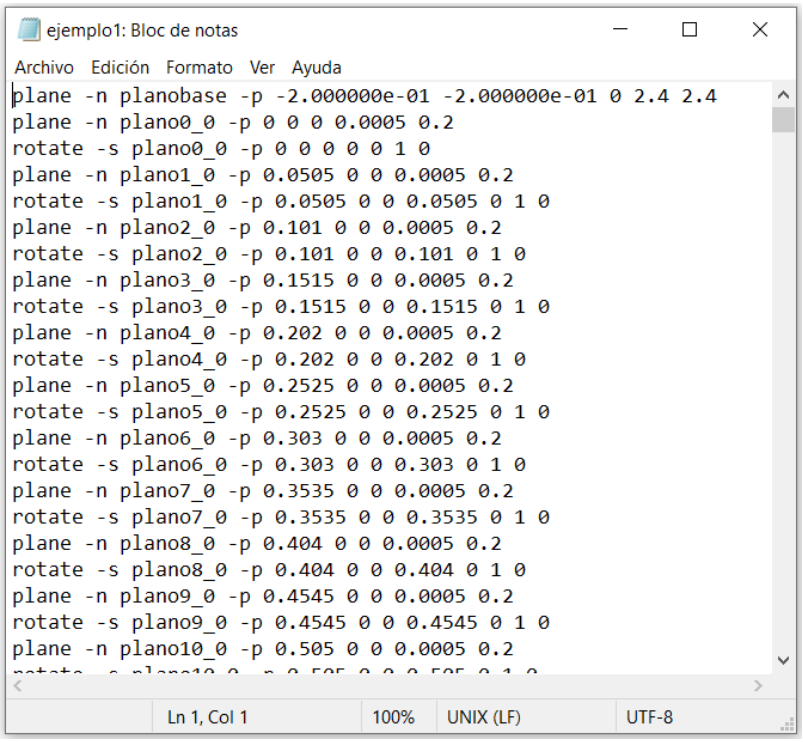


Ilustración 5.3.4 Ejemplo 1, fichero

Lo primero que podemos observar, es como el plano tiene esas medidas un poco superiores a las descritas previamente para asegurar que todas las tiras se encuentren dentro de este (aunque en este caso al tener una orientación de 0º no será necesario). Una vez generado el plano base, el programa empieza a escribir los comandos necesarios para generar cada una de las tiras con las dimensiones y orientación requeridas, hasta rellenar la superficie base, nombrando a cada una de las tiras como plano x_z, donde x es el número de tira generada de esa capa y z la altura en la que se encontrará la tira, diferenciando de esta forma las tiras de una capa con las de otra.

De forma que, una vez ya hayamos generado el archivo con los comandos mostrados, entramos en *NewFASANT*, y cargamos el script para que el software lo lea y genere la geometría:

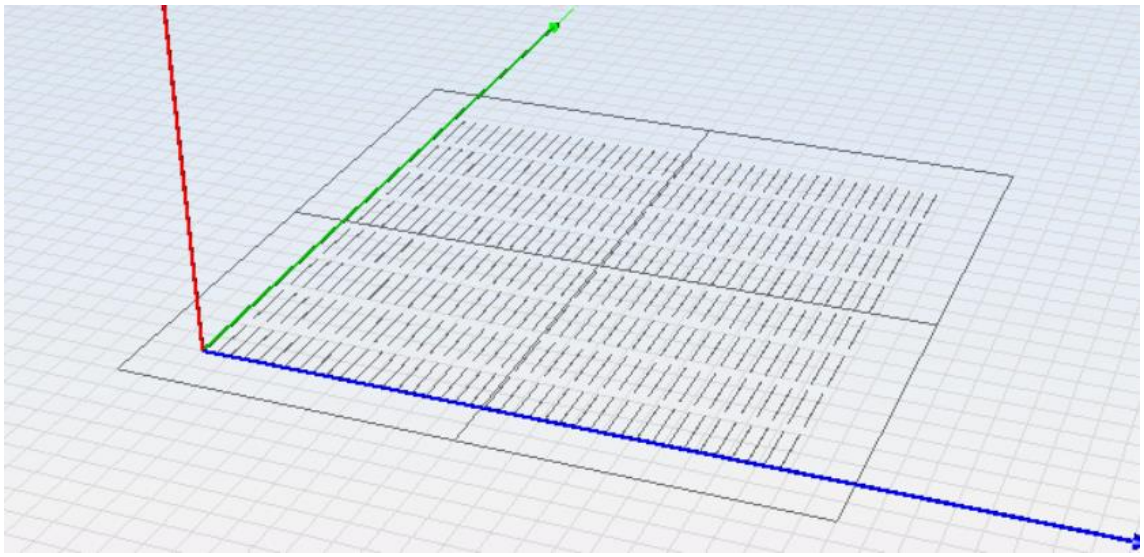


Ilustración 5.3.5 Ejemplo 1, geometría

Como podemos ver, *NewFASANT* genera exactamente la geometría deseada, dejando el espacio suficiente en el plano base para que, en caso de que las tiras tuviesen una orientación de 90º por ejemplo, no se encontrasen fuera de este.

Para estos ejemplos, también podemos ver reflejadas las distintas funcionalidades que emplea el usuario de forma esquematizada a través de los casos de uso, el cual podemos sintetizar del siguiente modo:

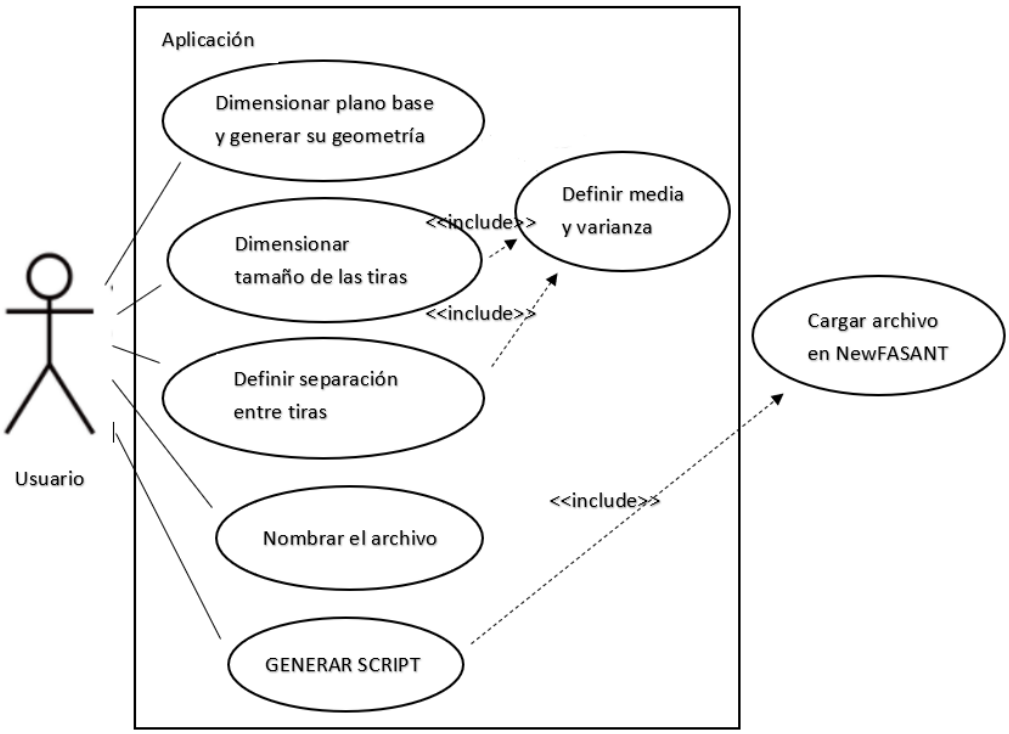


Ilustración 5.3.6 Ejemplo 1, diagrama de casos de uso

En él, se puede ver como el usuario dimensiona los parámetros necesarios para ejecutar el modelado geométrico deseado, donde observamos que, para el tamaño de las tiras y

la separación entre ellas, se incluye de forma implícita la definición de media y varianza de cada una de las medidas. De igual forma, el usuario después de esto y de nombrar el archivo, genera el script y se muestra que también de forma implícita, posteriormente se cargará el archivo y procesará en *NewFASANT*.

5.3.2. Ejemplo 2

Para ver mejor las otras funcionalidades, se muestra a continuación otro ejemplo, donde se vean el resto de las opciones que tiene el usuario para el modelado geométrico:

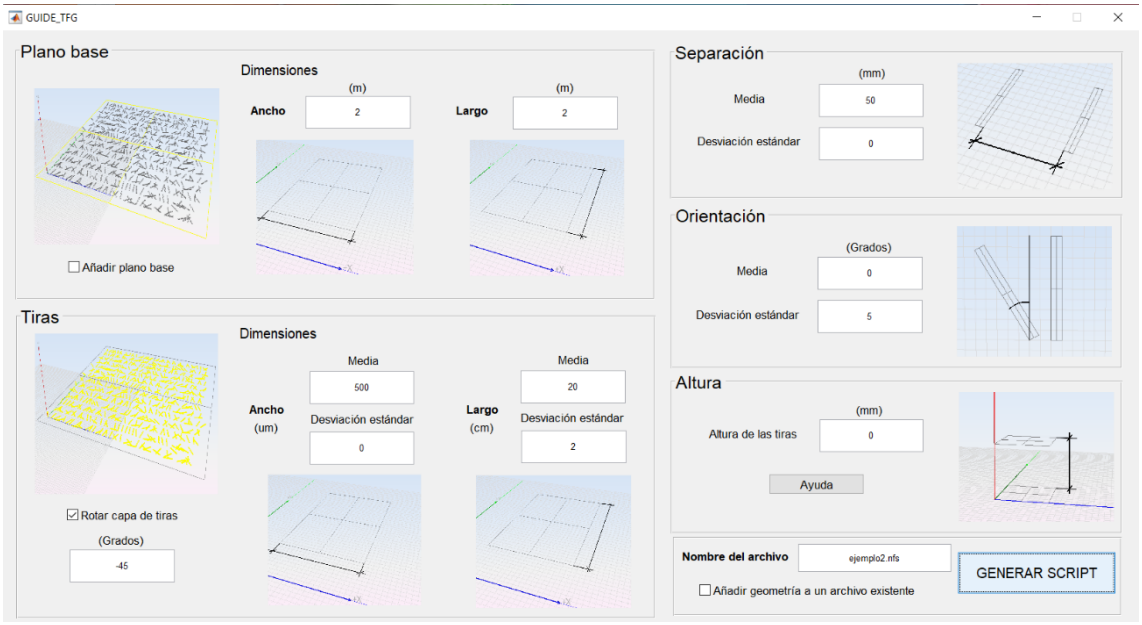


Ilustración 5.3.7 Ejemplo 2, parámetros

Las dimensiones introducidas han sido:

- **Plano base:**
  - Ancho = 2m
  - Largo = 2m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 20cm de media y 2cm de desviación estándar.
  - Rotamos capa de tiras -45º.
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 5º
- **Altura** = 0
- **Nombre del archivo** = ejemplo2.nfs
- Sin añadir a una geometría existente.

Generamos script y obtenemos el siguiente archivo:

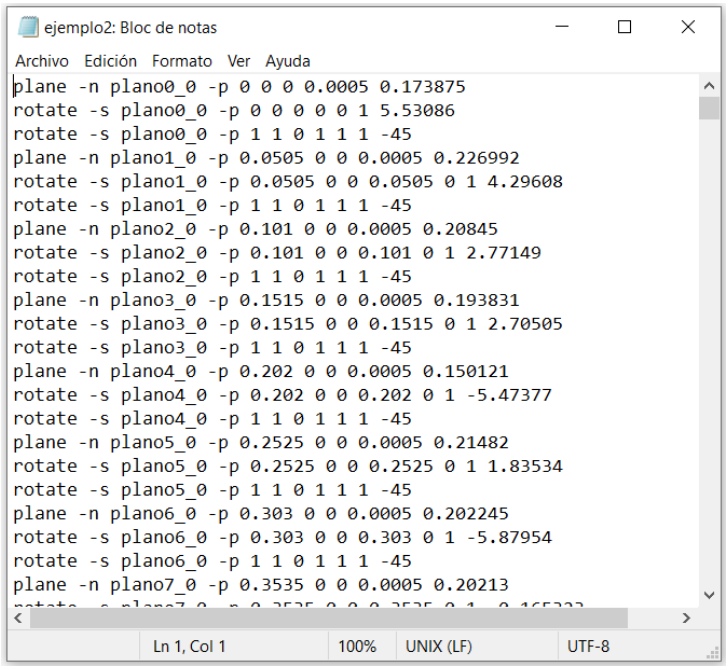


Ilustración 5.3.8 Ejemplo 2, fichero

Se puede observar cómo esta vez, el plano base no ha sido creado, por lo que el primer comando escrito por el programa es la primera tira que coloquemos en la superficie que hemos dimensionado como plano base. Cada tira creada, después, ha sido rotada sobre su punto de origen un ángulo aleatorio con media en 0 y desviación típica en 5º, para posteriormente rotar la tira entera -45º con el eje de giro en el centro del plano base, de forma que así conseguimos que toda la capa de tiras se encuentre rotada como había pedido el usuario.

Ahora, una vez generado el archivo mostrado, se procede a cargarlo en *NewFASANT* para que genere nuestra geometría, que es la siguiente:

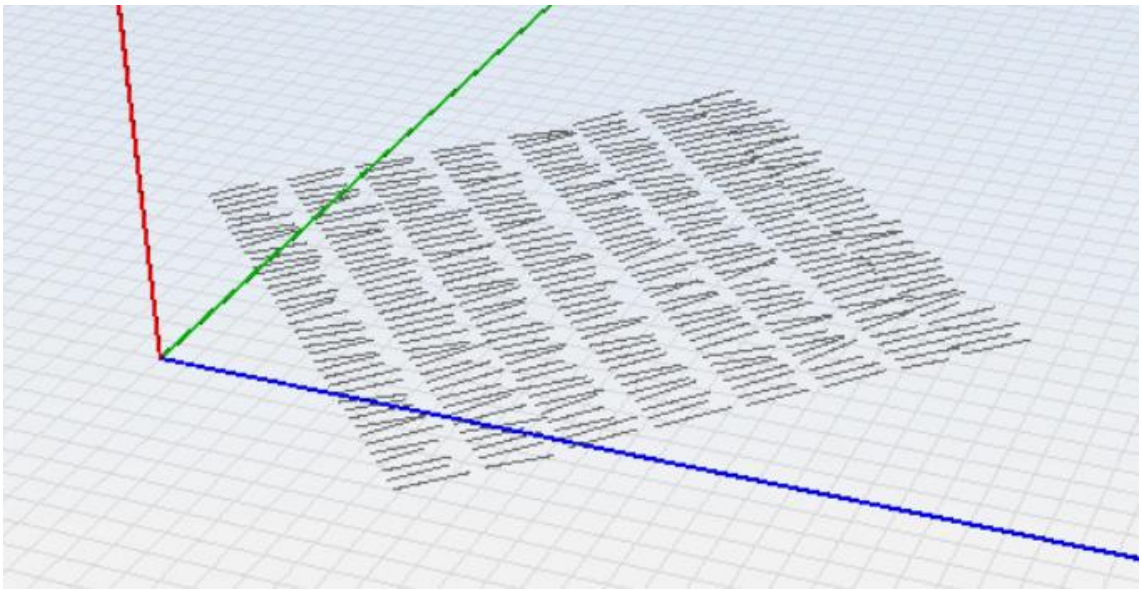


Ilustración 5.3.9 Ejemplo 2, geometría

Se puede ver como la geometría mostrada en la imagen superior, es la correspondiente a la especificada en las dimensiones previamente descritas, orientándose las tiras de forma aleatoria con la desviación típica de 5º, con un largo que también varía para cada una de ellas y finalmente rotando la capa completa de tiras 45º en sentido negativo al generado inicialmente.

El diagrama de caso de uso en este segundo ejemplo es el siguiente:

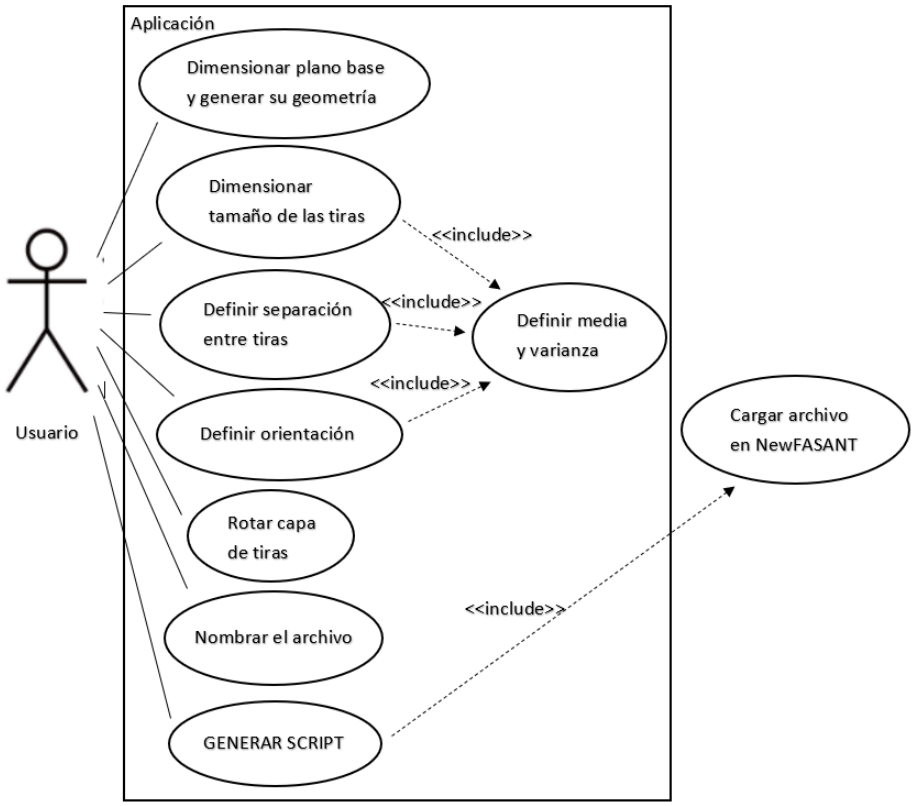


Ilustración 5.3.10 Ejemplo 2, diagrama de casos de uso

A diferencia del primer ejemplo, vemos que el usuario ha tenido que definir, además de los parámetros del ejemplo anterior, la orientación de las tiras (donde se ve que implícitamente también define su media y varianza) y la rotación de la capa de tiras. El resto no varía respecto al ejemplo anterior.

5.3.3. Ejemplo 3

A continuación, se mostrará un último ejemplo en el que se vea como realizar una geometría que contenga varias capas de tiras, puestas unas encima de otras a una determinada altura, y con las tiras colocadas con una orientación, separación y tamaño iguales para que se pueda mostrar de la forma más clara posible la diferencia entre cada una de las capas.

Capa 1:

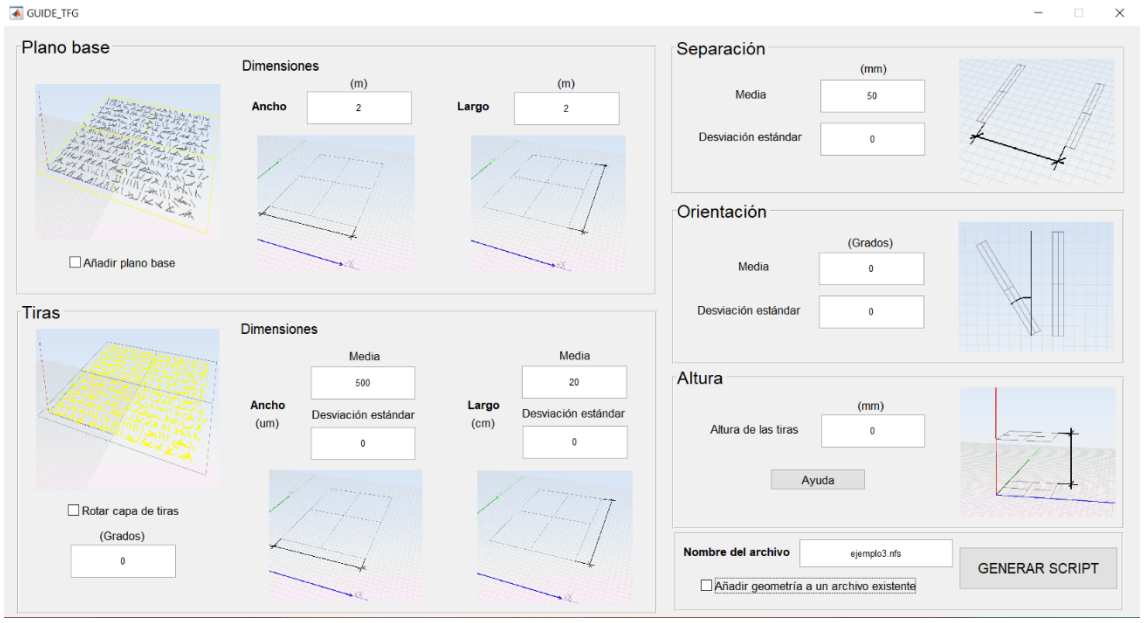


Ilustración 5.3.11 Ejemplo 3, parámetros de la capa 1

Capa 2:

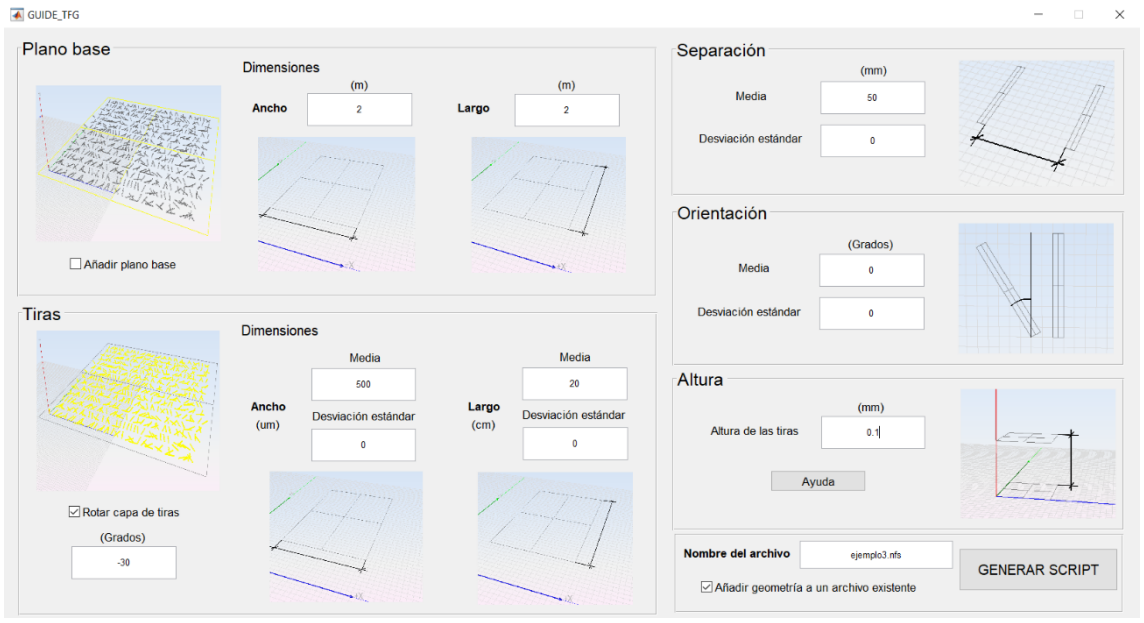


Ilustración 5.3.12 Ejemplo 3, parámetros de la capa 2

Capa 3:

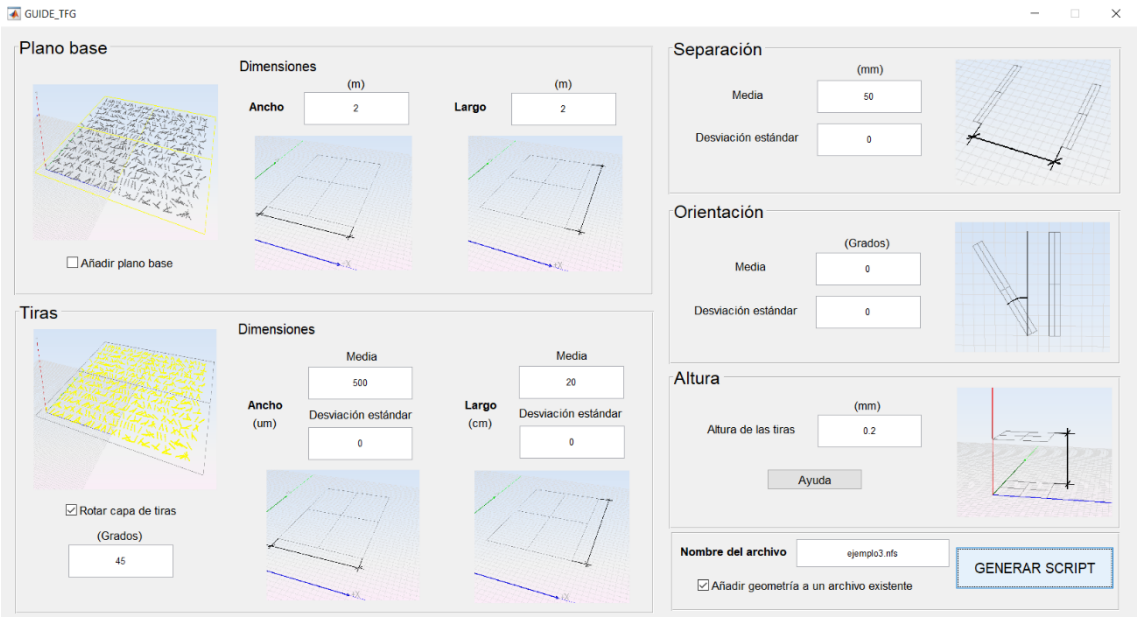


Ilustración 5.3.13 Ejemplo 3, parámetros de la capa 3

Las dimensiones, como se puede ver, has sido las siguientes:

- **Plano base:**
  - Ancho = 2m
  - Largo = 2m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 20cm de media y 0 de desviación estándar.
  - Rotación de la capa de tiras:
    - Capa 1 = 0
    - Capa 2 = -30º
    - Capa 3 = 45º
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 0
- **Altura:**
  - Capa 1 = 0
  - Capa 2 = 0,1mm
  - Capa 3 = 0,2mm
- **Nombre del archivo** = ejemplo3.nfs
- Primero creamos capa 1 dentro del archivo y después se añaden a este las otras dos capas.

Con estos datos, generamos el siguiente archivo de texto:

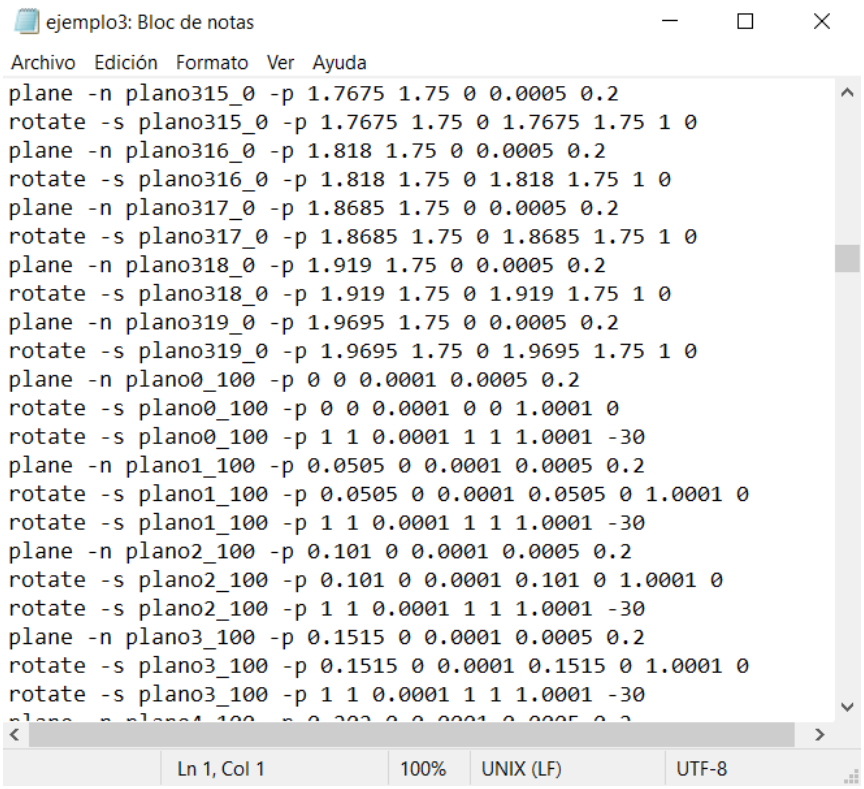


Ilustración 5.3.14 Ejemplo 3, fichero

Como se puede ver en la imagen superior, vemos que nuestro programa primero escribe los comandos correspondientes a las tiras de la primera capa, para después empezar a escribir debajo de estos comandos las tiras de la segunda capa, nombrando a cada una de las tiras como dijimos en el primer ejemplo (plano x_z, donde x es el número de tira generada de esa capa y z la altura en micras).

La geometría generada finalmente es la siguiente:

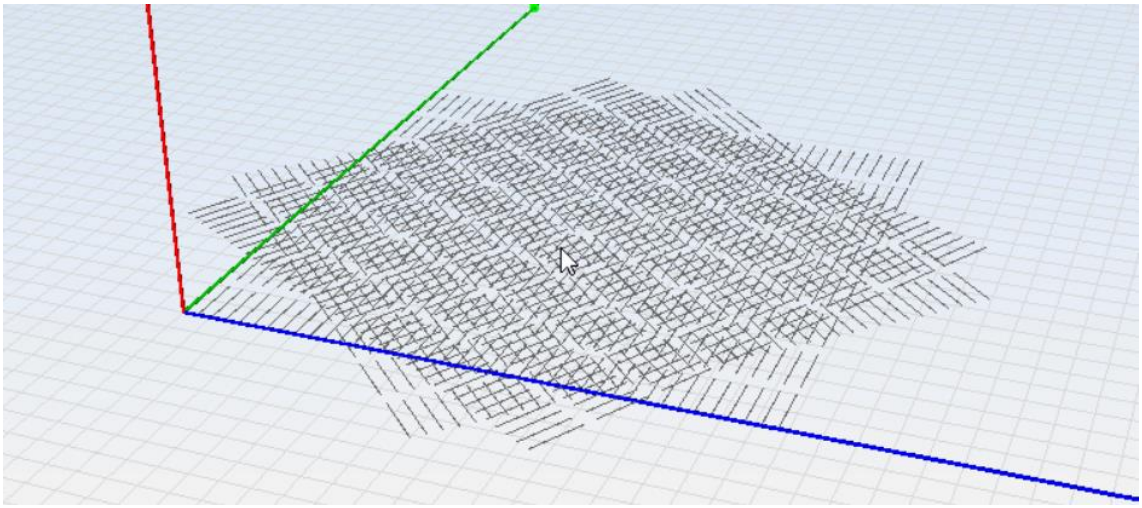


Ilustración 5.3.15 Ejemplo 3, geometría

Tal y como se esperaba, vemos la diferencia entre cada una de las capas perfectamente, viendo los ángulos de rotación de -30º y 45º como se había especificado.

Podemos ver de forma resumida este último ejemplo en el siguiente diagrama de caso de uso:

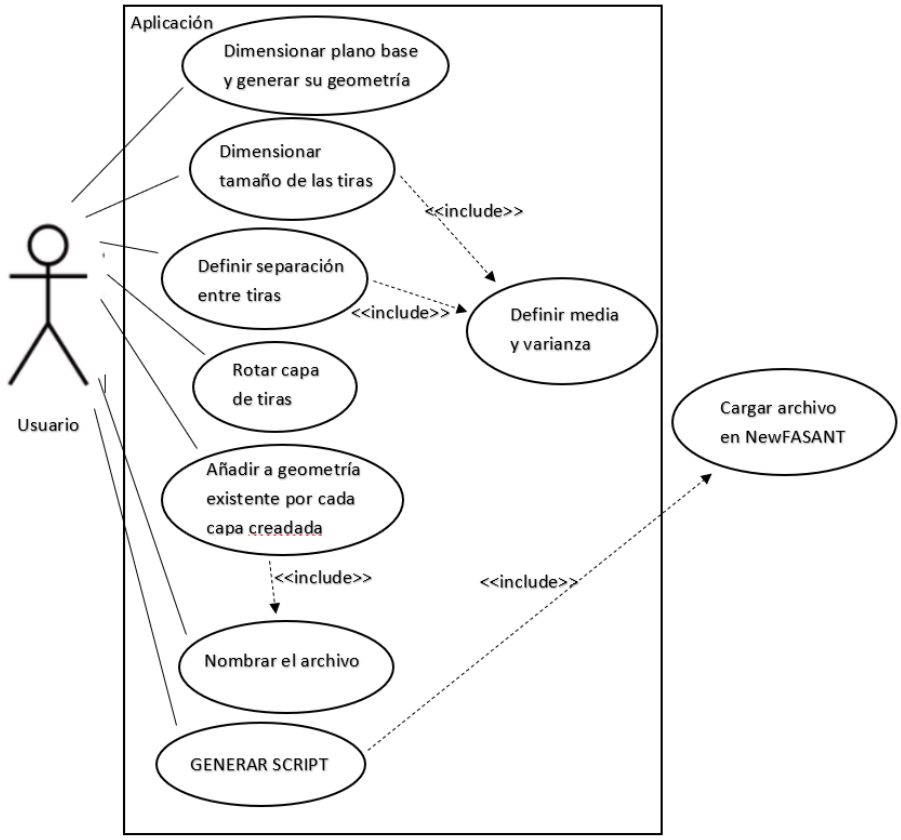


Ilustración 5.3.16 Ejemplo 3, diagrama de casos de uso

En el diagrama se puede ver esquematizado que, en este último ejemplo, el usuario debe añadir cada capa de tiras que desee crear a una geometría ya existente, donde se muestra también, que esto implica de manera implícita que el usuario deba definir un nombre de archivo donde ampliar dicha geometría. En el resto de las funcionalidades, el usuario actúa del mismo modo que en los ejemplos previos.

6. Simulaciones

En este apartado, veremos si el modelado geométrico generado por nuestra aplicación es procesado correctamente por *NewFASANT*, y se realizan correctamente las distintas simulaciones que se han explicado anteriormente en la introducción.

Para comprobar esto, se ejecutarán una serie de distintas simulaciones, de forma que podamos ver las distintas funcionalidades de nuestra aplicación y su puesta en práctica en la herramienta de *NewFASANT*.

El procedimiento que seguiremos será el de realizar una serie de modelados geométricos distintos con los que posteriormente serán cargados y procesados en *NewFASANT*, para así simular estos escenarios en l tipo de simulación MONCROS, donde se incidirá sobre las tiras con una onda plana, polarizando el campo eléctrico en una determinada dirección, y ver de esta forma como actúa este sobre las distintas superficies que vayamos a crear.

De este modo, se mostrará paso a paso todos los escenarios creados, explicando la geometría y los distintos parámetros de simulación que se han escogido a la hora de su ejecución.

6.1. Primeros conceptos de simulación

6.1.1. Ejemplo 0 de simulación

Dicho lo anterior, comenzamos creando un escenario sencillo, en el que podamos ver cómo vamos a proceder para las siguientes simulaciones que se vayan a hacer. Así pues, lo primero que crearemos será únicamente un plano cuyas dimensiones son de 1m de ancho y 4m de largo, con una altura z=0.

De esta forma, vamos a ver cómo actúa el campo eléctrico en un escenario donde no haya excesivos factores para posteriormente interpretar los resultados de la forma más próxima posible. El plano descrito anteriormente se ve tal que así:

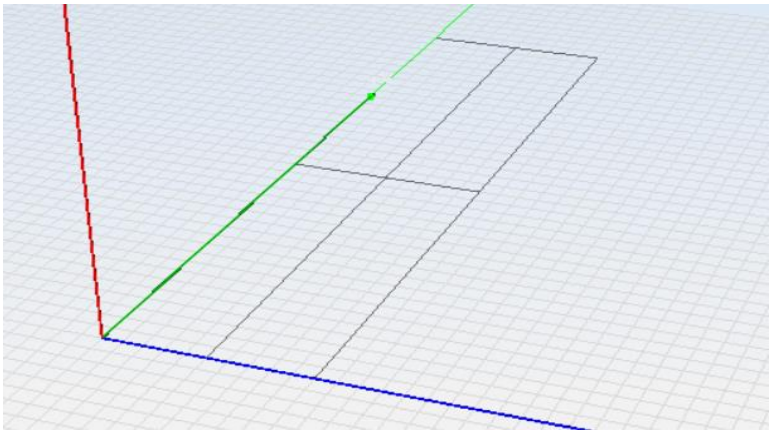


Ilustración 6.1.1 Ejemplo 0 de simulación, geometría

Una vez hemos creado el plano, se configuran los distintos parámetros de simulación (explicados previamente en el punto 3.3) de la siguiente forma:

- **Frecuencias:**
  - Sin Barrido.
  - 100MHz
- **Direcciones de observación:**
  - Corte en Phi = 0
  - Theta inicial = 0
  - Incremento = 1
  - Muestras = 181
  - Theta final = 180
- **RCS:**
  - Monostático
  - Polarización de onda: Etheta = 1; Ephi = 0;
- **Mesh:**
  - “Planar faces” = 10
  - “Curved faces” = 10
  - Bandas por octavo = 1
  - Procesadores = 16
- **Solver:**
  - “Method of Moments” (MoM)
  - “Architecture Strategy”: OpenMP
  - “Solver Functions”: Subdominios
  - Error relativo = 0.01
  - Máximo número de iteraciones = 5000

Con estos parámetros definidos, se procede primero, a realizar el mallado y posteriormente, a ejecutar la simulación, donde podemos ver lo siguiente:

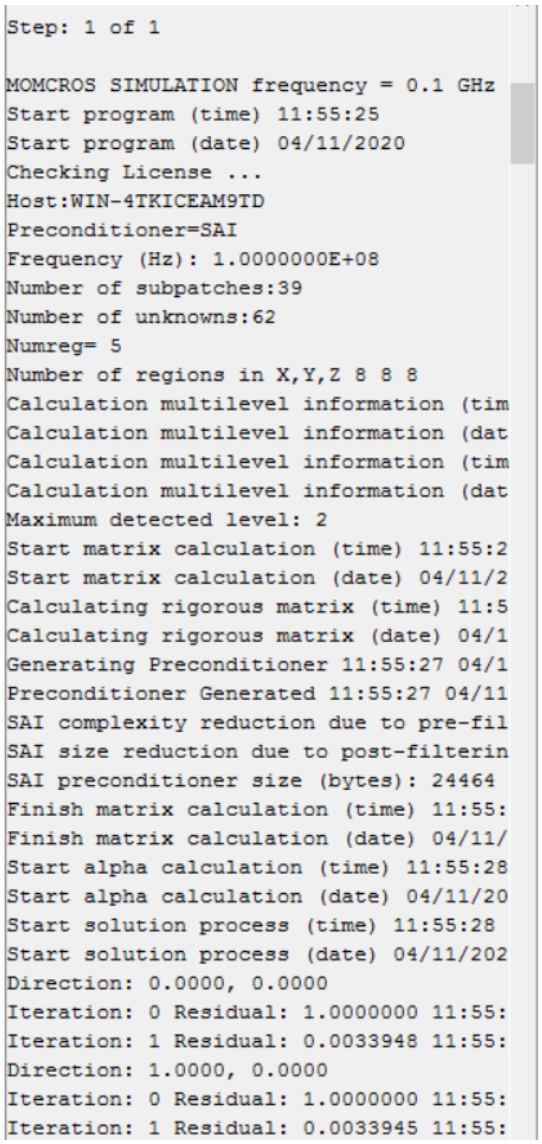


Ilustración 6.1.2 Ejemplo 0 de simulación, log

En primera instancia, si nos fijamos en el 'log' de la simulación vemos que la simulación se va a realizar a 100MHz (esta vez será la única frecuencia que ejecute el programa), y que el número de incógnitas a resolver serán un total de 62, un número bastante pequeño en comparación a las siguientes que veremos, debido a lo simple de nuestro escenario y el valor también pequeño de la frecuencia.

También se puede ver en la parte inferior de la imagen, cómo va a ir calculando los valores por cada dirección de observación, obteniendo al final una gráfica como la siguiente:

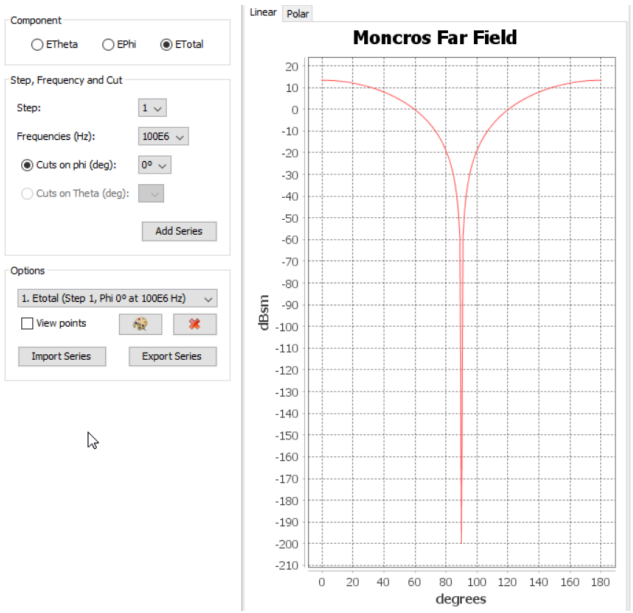


Ilustración 6.1.3 Ejemplo 0 de simulación, gráfica valor del campo respecto de la dirección de observación

En la gráfica tenemos, en el eje X, los grados de 'theta' en las coordenadas polares con los que se realiza la dirección de observación, donde se puede ver que en esta simulación tiene valores que van desde los 0º hasta los 180º, con corte en 'phi' = 0, y en el eje Y, los valores de campo calculados.

Así pues, se puede deducir, que a medida que nos acercamos a 'theta' = 90º el valor de campo tiende a ser muy pequeño, prácticamente 0. Esto se debe a que, al ser la incidencia rasante a la placa, casi no existe interacción, ya que el campo polarizado según 'theta' para esa dirección apunta a -z y no se inducen corrientes, es decir, es como si la onda no viese la placa, por lo que sería muy complicado que fuese detectado por radar.

Es por esto, que las siguientes simulaciones se realizarán con una dirección de observación perpendicular al plano de las tiras (de forma positiva al eje Z y de forma opuesta), haciendo más interesante el realizar un barrido de frecuencias y ver como interactúa nuestra superficie con el campo a medida que aumentamos o disminuimos la frecuencia de onda, como se verá a continuación.

6.1.2. Ejemplo 1 de simulación

Una vez ya hemos visto a un primer ejemplo de simulación en un escenario sencillo, para así hacernos una idea del procedimiento a seguir y los parámetros a tener en cuenta, procedemos ahora a crear escenarios más próximos a las simulaciones que van en la línea de trabajo de este proyecto.

Dichas simulaciones, tendrán como interés usar tiras de entre 3 y 30 cm de largo y entre 50 y 500 micras de ancho, en un intervalo de frecuencias de entre 0,5 GHz y 40 GHz. La alimentación, como ya hemos visto, será por onda plana, polarizando dicha onda de distinta forma según nos resulte conveniente, como ya iremos viendo.

En este segundo ejemplo de simulación, crearemos una geometría sin demasiadas tiras, por lo que 100 tiras será un buen número de momento, y tampoco generaremos un plano base para que no produzca interferencias innecesarias, ya que lo que intentaremos comprobar en esta serie de simulaciones, será ver como varía el campo en función a la variación de la disposición y densidad del número de tiras.

Es por esto por lo que, la primera geometría generada es la siguiente:

- **Plano base:**
  - Ancho = 2m
  - Largo = 2m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 30cm de media y 0 de desviación estándar.
  - No rotamos capa de tiras.
- **Separación:**
  - Media = 100mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 0
- **Altura** = 0
- **Nombre del archivo** = 1sim.nfs

Una vez el usuario haya especificado los parámetros mencionados en la aplicación, y haber cargado y procesado el modelado geométrico por *NewFASANT*, vemos lo siguiente:

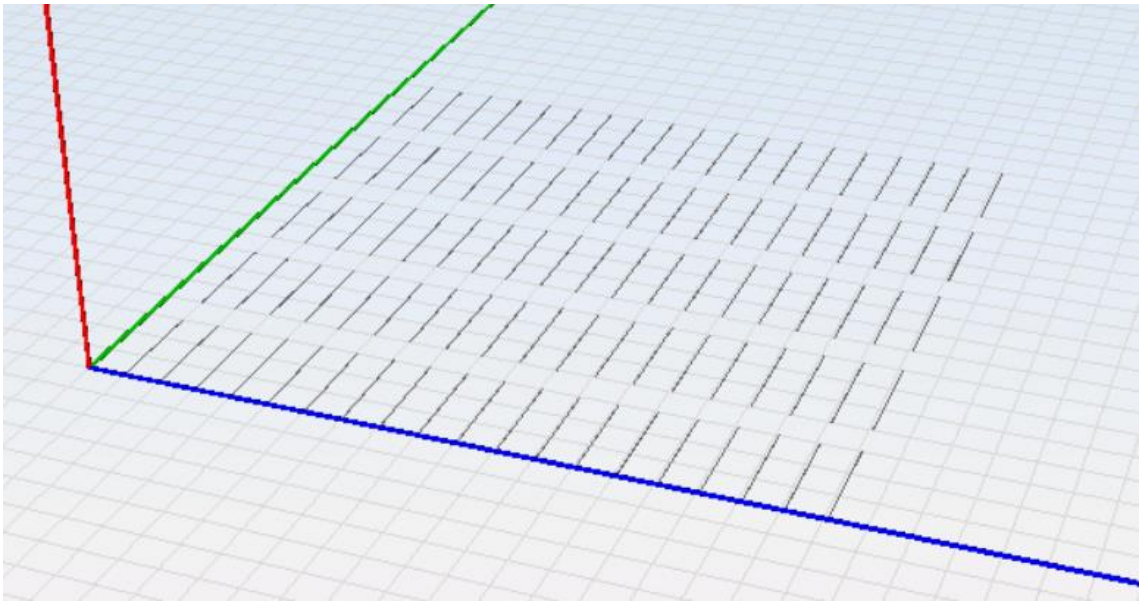


Ilustración 6.1.4 Ejemplo 1 de simulación, geometría (1)

De momento, como se muestra en la imagen superior, la densidad de tiras es bastante baja, y la orientación de estas es la misma, estando orientadas en el eje Y. Ahora que ya hemos creado la geometría deseada, procedemos a configurar los parámetros de simulación:

- **Frecuencias:**
  - Barrido de frecuencias.
  - 0,5GHz – 10GHz
- **Direcciones de observación:**
  - Corte en Phi = 0
  - Theta inicial = 0
  - Incremento = 180
  - Muestras = 2
  - Theta final = 180
- **RCS:**
  - Monostático
  - Polarización de onda: Etheta = 1; Ephi = 0;
- **Mesh:**
  - “Planar faces” = 10
  - “Curved faces” = 10
  - Bandas por octavo = 1
  - Procesadores = 16
- **Solver:**
  - “Method of Moments” (MoM)
  - “Architecture Strategy”: OpenMP
  - “Solver Functions”: Subdominios
  - Error relativo = 0.01
  - Máximo número de iteraciones = 5000

A diferencia del ejemplo expuesto anteriormente, esta vez realizaremos la simulación con un barrido de frecuencias desde 0,5GHz hasta 10GHz, teniendo como únicas direcciones de observación, las perpendiculares al plano de las tiras (de forma positiva al eje z y de forma opuesta), con corte en ‘phi ‘en 0, lo que implica que al estar polarizada la onda en ‘Etheta’ = 1 y ‘Ephi’ = 0, la dirección del campo será en el eje X y -X respectivamente.

Así pues, las gráficas que obtenemos una vez ejecutada la simulación son las siguientes:

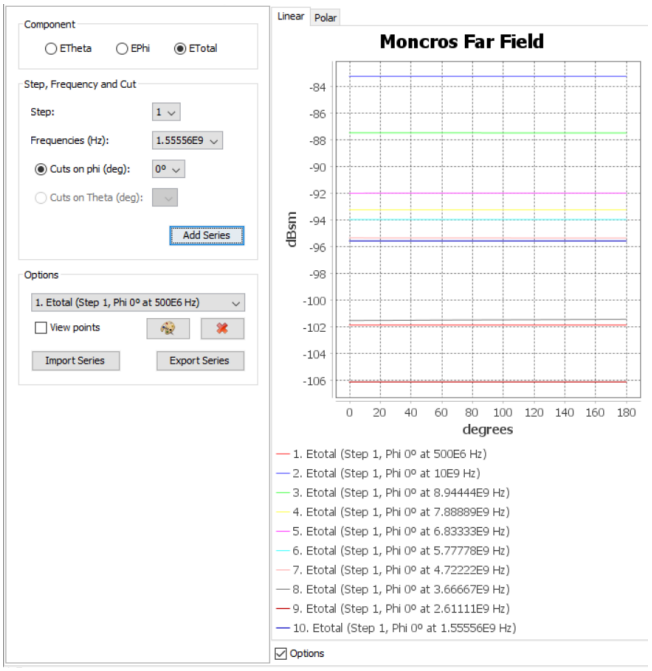


Ilustración 6.1.5 Ejemplo 1 de simulación, gráfica valor de campo respecto de la dirección de observación (1)

Primero observamos como varía el campo según la dirección de observación, y lo que podemos ver es que toma el mismo valor para 'theta' = 0 que para 180º, lo cual tiene sentido si la onda incide en la superficie en la misma dirección en ambos casos pero en sentido contrario una respecto de la otra, por lo que para los casos que tengamos las observaciones de dirección igual que en este caso, resultará más interesante ver cómo varía el campo en función de la frecuencia. Esa gráfica es la siguiente:

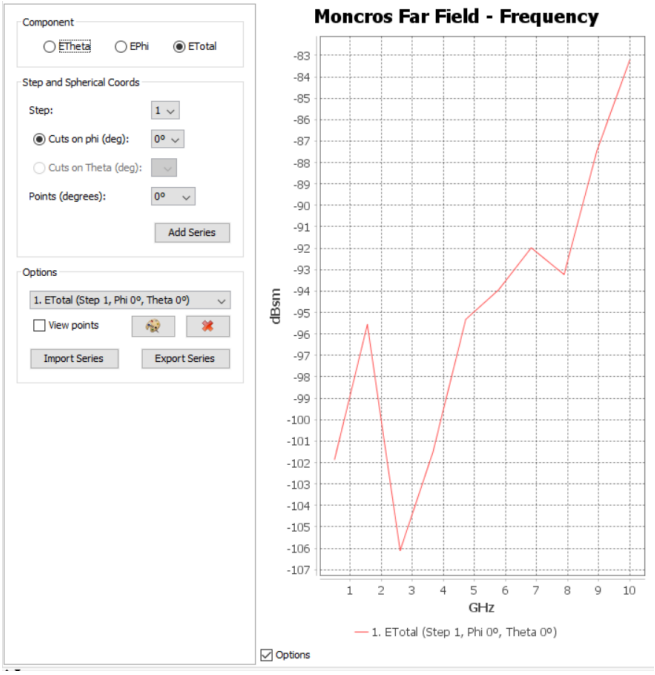


Ilustración 6.1.6 Ejemplo 1 de simulación, gráfica valor de campo respecto de la frecuencia (1)

Esta gráfica representa la unión de los distintos valores que teníamos antes, pero únicamente para cuando 'theta' = 0. Se puede ver como dichos valores de campo son

muy bajos, lo que se debe seguramente al poco número de tiras que hay y a su baja densidad, aparte de que el campo para ambas direcciones de observación se encuentra de forma opuesta a la de las tiras, lo que hace que los valores de campo sean aún menores.

Para comprobar si el campo sufre una gran variación en función del número de tiras que haya, cambiaremos la geometría existente, aumentando la superficie y reduciendo la separación entre ellas, aumentando así el número de tiras de 100 a 1400, un número significativamente mayor al anterior y más acorde a posibles escenarios reales que quiera ejecutar nuestro usuario. Así pues, la geometría generada es la siguiente:

- **Plano base:**
  - Ancho = 5m
  - Largo = 5m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 30cm de media y 0 de desviación estándar.
  - No rotamos capa de tiras.
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 0
- **Altura** = 0
- **Nombre del archivo** = 1.1sim.nfs

Visualmente, la geometría descrita sería la siguiente:

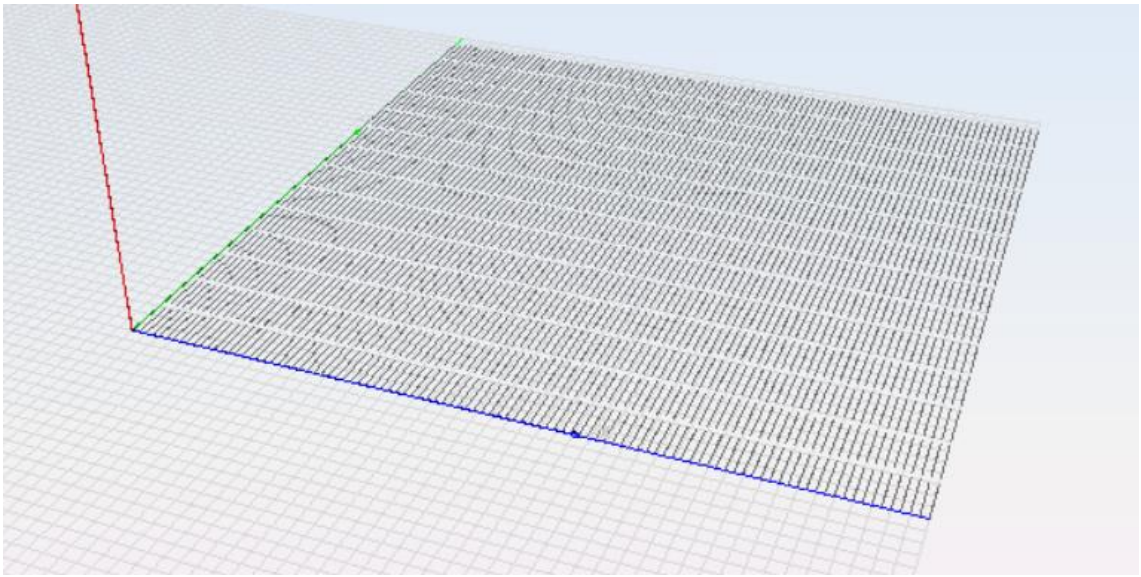


Ilustración 6.1.7 Ejemplo 1 de simulación, geometría (2)

Como se puede ver, ahora las tiras se encuentran con menos separación y en un número mucho mayor que antes, y, a parte, la superficie también es superior. Es por esto, por lo que, de esta forma, podemos ver si existe una gran variación del campo respecto a la geometría previa:

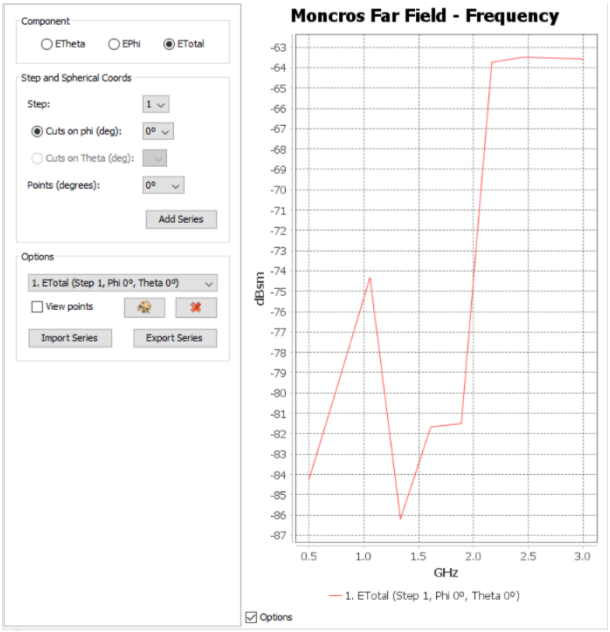


Ilustración 6.1.8 Ejemplo 1 de simulación, gráfica valor de campo respecto de la frecuencia (2)

Es claro que los valores de campo son todavía suficientemente pequeños como para considerar que es muy cercano a 0, ya que la polarización de la onda sigue siendo en sentido contrario a la dirección de las tiras, pero sí que se puede ver que estos han aumentado algo para deducir que el aumento del número de tiras implica el aumento del valor del campo.

Otro aspecto por destacar de las gráficas, son las subidas y bajadas de valores de campo que podemos ver que se forman. Esto se debe a que a ciertas frecuencias las corrientes se suman en fase y a otras en contrafase. La explicación de esto es que las corrientes se inducen en las tiras, y esas corrientes radian y dan lugar a campos, por lo que el total es una integral de todos los puntos que radian, es decir, es como si cada punto de las tiras fuera una antenita. A ciertas frecuencias la longitud de onda es tal que esos campos que se radian tienden a sumarse en fase, especialmente si la geometría es regular como en este caso. De ahí vienen los lóbulos.

6.1.3. Ejemplo 2 de simulación

Ahora que ya hemos visto una primera simulación con escenarios que van en la línea de trabajo de nuestro proyecto, pero con una dirección de campo contraria a la de las tiras, lo que haremos ahora será que ambas direcciones sean las mismas.

Para ello, la geometría no sufrirá ninguna variación respecto a la simulación anterior (refiriéndonos a la de las 1400 tiras) y lo que si cambiará serán los parámetros de simulación:

- **Frecuencias:**
  - Barrido de frecuencias.
  - 0,5GHz – 3GHz
- **Direcciones de observación:**
  - Corte en Phi = 90
  - Theta inicial = 0
  - Incremento = 180
  - Muestras = 2
  - Theta final = 180
- **RCS:**
  - Monostático
  - Polarización de onda: Etheta = 1; Ephi = 0;
- **Mesh:**
  - "Planar faces" = 10
  - "Curved faces" = 10
  - Bandas por octavo = 1
  - Procesadores = 16
- **Solver:**
  - "Method of Moments" (MoM)
  - "Architecture Strategy": OpenMP
  - "Solver Functions": Subdominios
  - Error relativo = 0.05
  - Máximo número de iteraciones = 5000

Para conseguir lo especificado, lo que se ha cambiado de los parámetros de simulación respecto a los anteriores es que, ahora, en las direcciones de observación, el corte en 'phi' es de 90º, consiguiendo así que si polarizamos la onda en 'Etheta' = 1 y 'Ephi = 0', la dirección del campo en esta ocasión será en paralelo al eje Y. Otros parámetros que también han variado son, la frecuencia de simulación, y el error relativo de "Solver".

En cuanto al error relativo, hemos tenido que aumentar el valor a 0,05 porque, sino, los valores comenzaban a converger y la simulación no terminaba de calcular nada, y en cuanto a las frecuencias de simulación, se han cambiado, ya que, en esta ocasión, tenemos un barrido de 0,5GHz a 3GHz, en lugar de 0,5GHz a 10GHz. Esto se debe a que lo único que queremos ver es cómo varía el campo en función de la frecuencia de onda en un rango de valores que estén acorde a los valores especificados en las simulaciones que vamos a hacer (de 0,5GHZ a 40GHz), pero sin que tampoco sea muy pesado el tiempo de simulación como puede ser con 40GHz, ya que, a mayor frecuencia, menor longitud de onda, y mayor número de incógnitas a resolver por el programa y, por tanto, mayor tiempo de simulación. Es por esto, que la simulación se realiza en con ese barrido.

Una vez ejecutado el programa, vemos la siguiente gráfica:

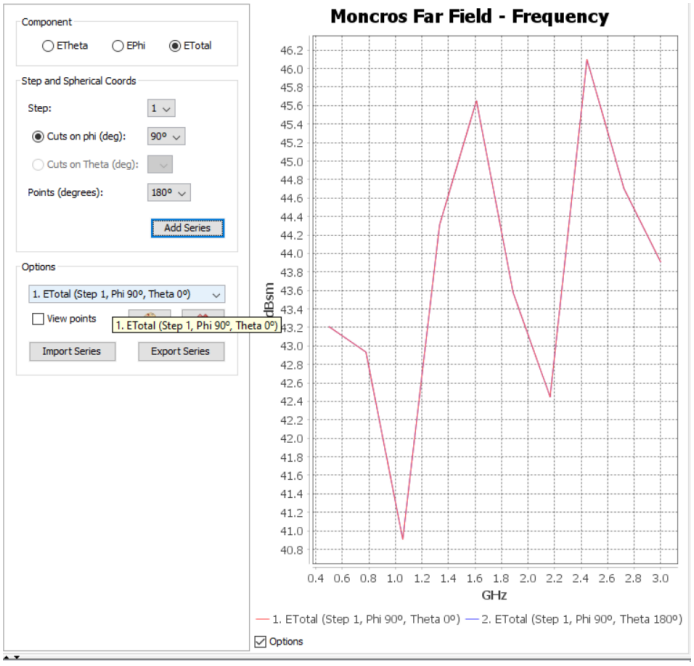


Ilustración 6.1.9 Ejemplo 2 de simulación, gráfica valor de campo respecto de la frecuencia

En la imagen se puede apreciar como ahora el valor de campo aumenta significativamente, alcanzando valores desde 41dBsm, hasta los 46 dBsm, lo que se traduce en unas mayores corrientes inducidas en las tiras en este caso. De esta forma, comprobamos que cuando el campo se orienta en la misma dirección que las tiras, los valores que obtenemos de él en las gráficas es bastante mayor en comparación a si fuese de forma opuesta, como en los casos anteriores.

6.2. Simulaciones introduciendo funcionalidades de la aplicación

En estas simulaciones, se tendrá como objetivo el ver cómo afectan la variación de algunas de las funcionalidades que se dispone en nuestra aplicación, de forma que iremos observando si nuestra aplicación crea una geometría más compleja, la cual *NewFASANT* sea capaz de procesarla correctamente y mostrarnos en su simulación resultados coherentes y cercanos a lo esperado.

6.2.1. Orientación

Primeramente, haremos pequeñas variaciones en la orientación de las tiras para ver qué ocurre y si existen grandes cambios respecto a lo anterior. El modelado geométrico será igual que en el ejemplo 2 del punto anterior, pero añadiremos una pequeña desviación típica en la orientación, de exactamente 5º, quedando nuestros valores geométricos tal que así:

- **Plano base:**
  - Ancho = 5m
  - Largo = 5m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 30cm de media y 0 de desviación estándar.
  - No rotamos capa de tiras.
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 5º
- **Altura** = 0
- **Nombre del archivo** = 3sim.nfs

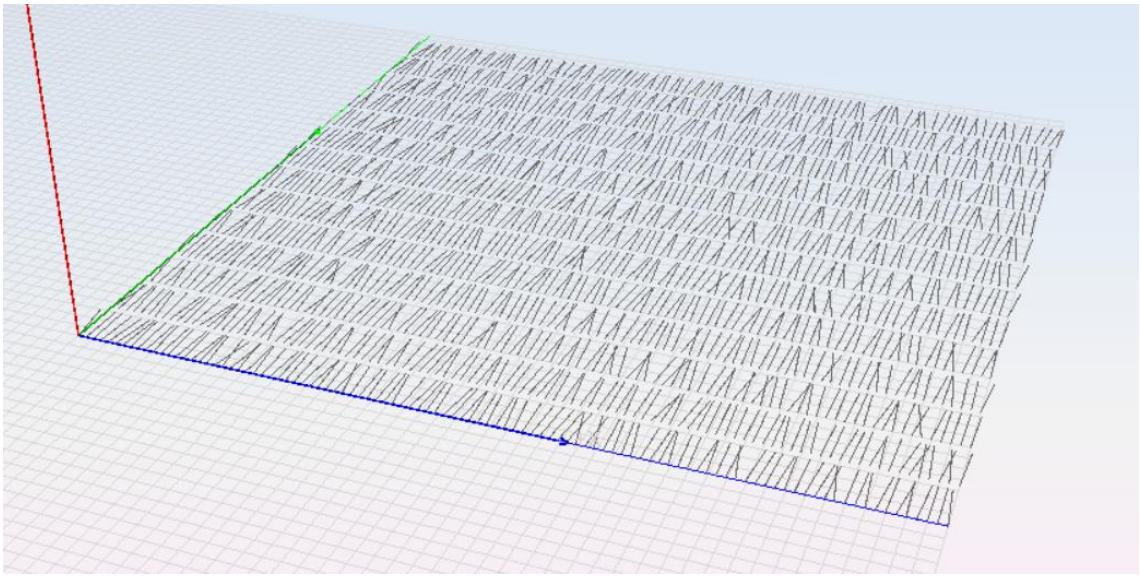


Ilustración 6.2.1 Simulación (orientación), geometría (1)

Podemos ver una geometría relativamente parecida a la anterior, pero esta vez las tiras sufrirán una leve rotación respecto de la dirección del eje Y, por lo que veremos en qué afecta en nuestra simulación.

Una vez simulado, con los parámetros del ejemplo 2 del punto anterior, vemos que la variación del campo respecto de la frecuencia es la siguiente:

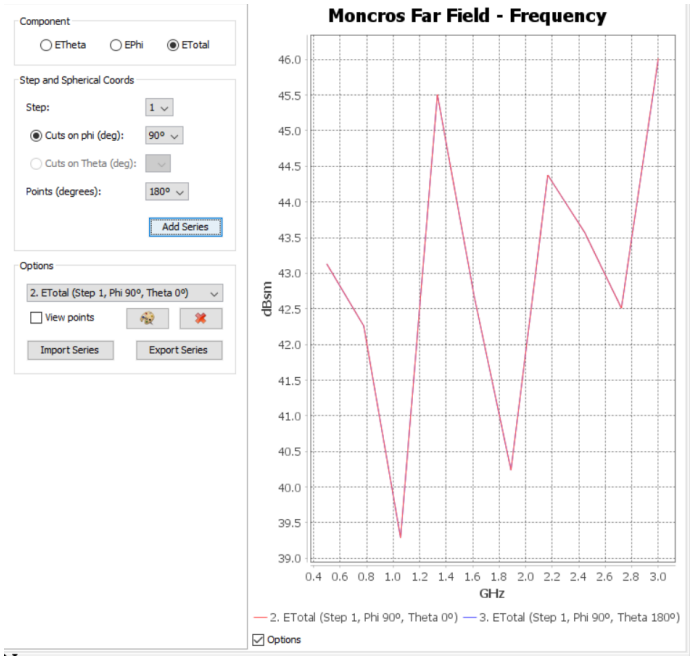


Ilustración 6.2.2 Simulación (orientación), gráfica valor de campo respecto de la frecuencia (1)

En cuanto a los valores de campo, se podría decir que se encuentran en un intervalo parecido al ejemplo previo, pero en esta ocasión la gráfica muestra lóbulos en distintas frecuencias que antes. Esto significa que, en este caso, al encontrarse las tiras orientadas de forma diferente, el campo también lo hace de manera distinta, por lo que las corrientes inducidas se sumarán de otra forma y crearán esas subidas y bajadas en otras frecuencias respecto a los ejemplos anteriores.

Si aumentamos la desviación estándar a 90º, de manera que ahora la orientación sea totalmente aleatoria, veremos que se vuelve a producir lo mismo que en este caso, pero de nuevo, los lóbulos los encontraremos en otras frecuencias. La geometría con luna orientación con desviación típica de 90º sería la siguiente:

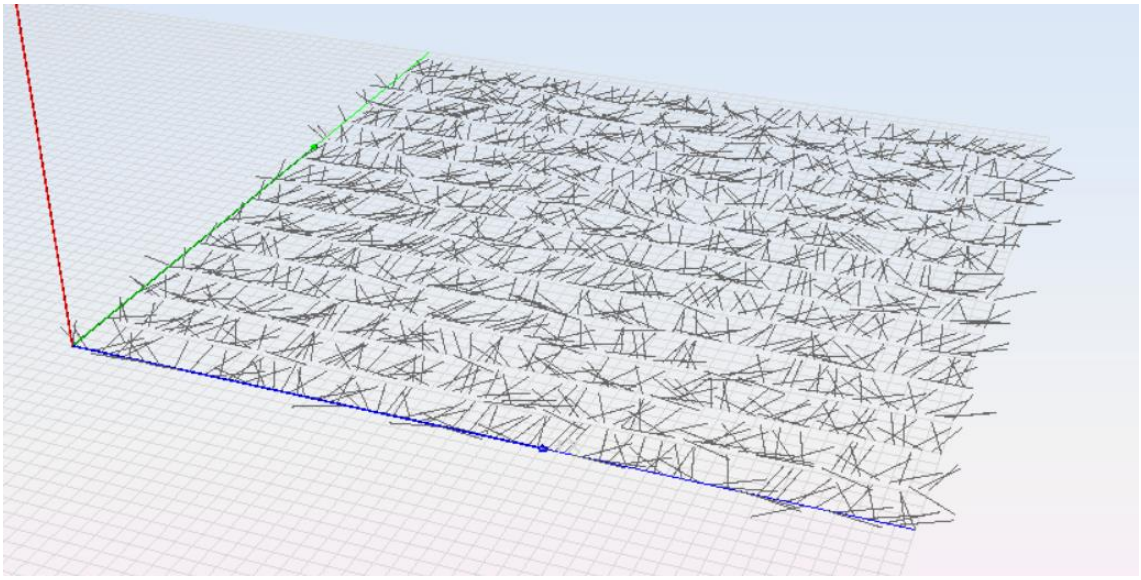


Ilustración 6.2.3 Simulación (orientación), geometría (2)

Y su gráfica de variación de campo respecto de la frecuencia:

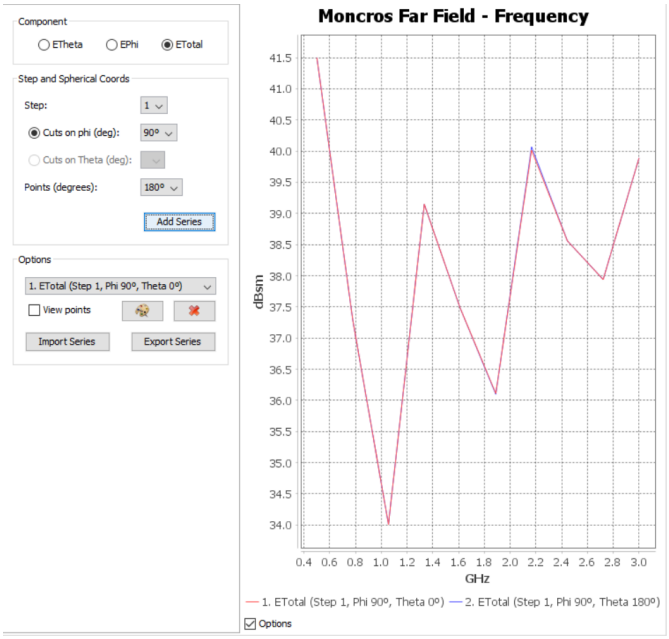


Ilustración 6.2.4 Simulación (orientación), gráfica valor de campo respecto de la frecuencia (2)

Efectivamente, se produce lo comentado anteriormente, salvo que los valores de campo han descendido levemente, lo que cual se puede deber a que, al no haber ahora una dirección parecida para todas las tiras, el campo no induce en ellas de manera tan efectiva como antes, lo que da lugar a esos valores inferiores respecto de lo anterior.

6.2.2. Adición de una o más capas

Ahora, haremos uso de la funcionalidad de añadir capas a una geometría existente. Para ello, crearemos una primera capa con la misma geometría que la anterior, para después, añadir a una altura diferente (de 1mm entre capa y capa) las siguientes, con los mismos parámetros que la primera:

- **Plano base:**
  - Ancho = 5m
  - Largo = 5m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 30cm de media y 0 de desviación estándar.
  - Sin rotación en las capas de tiras
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0

- Desviación estándar = 90º

- **Altura:**
  - Capa 1 = 0
  - Capa 2 = 1mm
  - Capa 3 = 2mm
- **Nombre del archivo** = 5sim.nfs

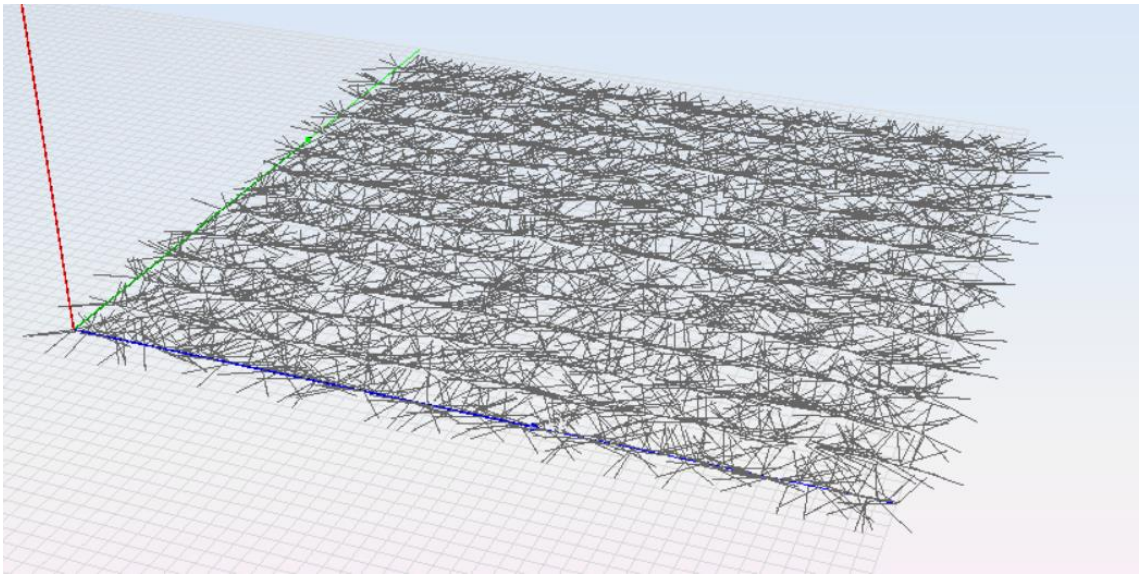


Ilustración 6.2.5 Simulación (adición de capas), geometría

En la imagen podemos ver que, en esta ocasión, la densidad de tiras que existe en la superficie del plano base es mucho mayor de lo que era anteriormente, ya que, aunque no se encuentren todas estas tiras en la misma capa, sí que se encuentran dentro del misma área, lo que hace que el número total de tiras total de nuestra geometría aumente hasta 4200.

Si vemos cómo es en esta ocasión la magnitud del campo, vemos lo siguiente:

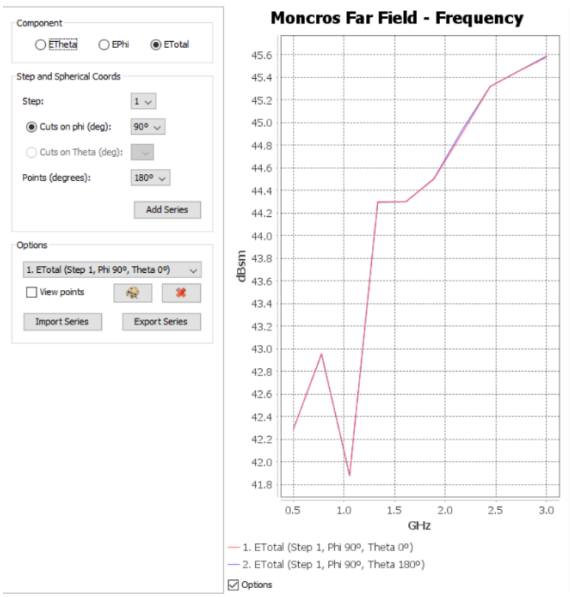


Ilustración 6.2.6 Simulación (adición de capas), gráfica valor de campo respecto de la frecuencia

El que haya aumentado el número de tiras, provoca que ahora los valores de campo vuelvan a valores de entre 40 y 45 dBsm, a pesar de que la orientación sea totalmente aleatoria.

6.2.3. Rotación de capas de tiras

Por último, veremos la funcionalidad de rotar la capa de tiras que vayamos a generar en las siguientes simulaciones. Para ello, estudiaremos la simulación de dos capas de tiras iguales pero cruzadas, para que, de esta forma, cada capa filtre una polarización.

Por lo tanto, la geometría que generemos será la siguiente:

- **Plano base:**
  - Ancho = 5m
  - Largo = 5m
  - No generamos geometría.
- **Tiras:**
  - Ancho = 500um de media y 0 de desviación estándar.
  - Largo = 30cm de media y 0 de desviación estándar.
  - Rotación de la capa de tiras:
    - Capa 1 = 0
    - Capa 2 = -90º
- **Separación:**
  - Media = 50mm
  - Desviación estándar = 0
- **Orientación:**
  - Media = 0
  - Desviación estándar = 0
- **Altura:**
  - Capa 1 = 0
  - Capa 2 = 0,1mm
- **Nombre del archivo** = 6sim.nfs

Al dimensionar los parámetros de nuestro modelado geométrico de esta forma, obtenemos una geometría como la siguiente:

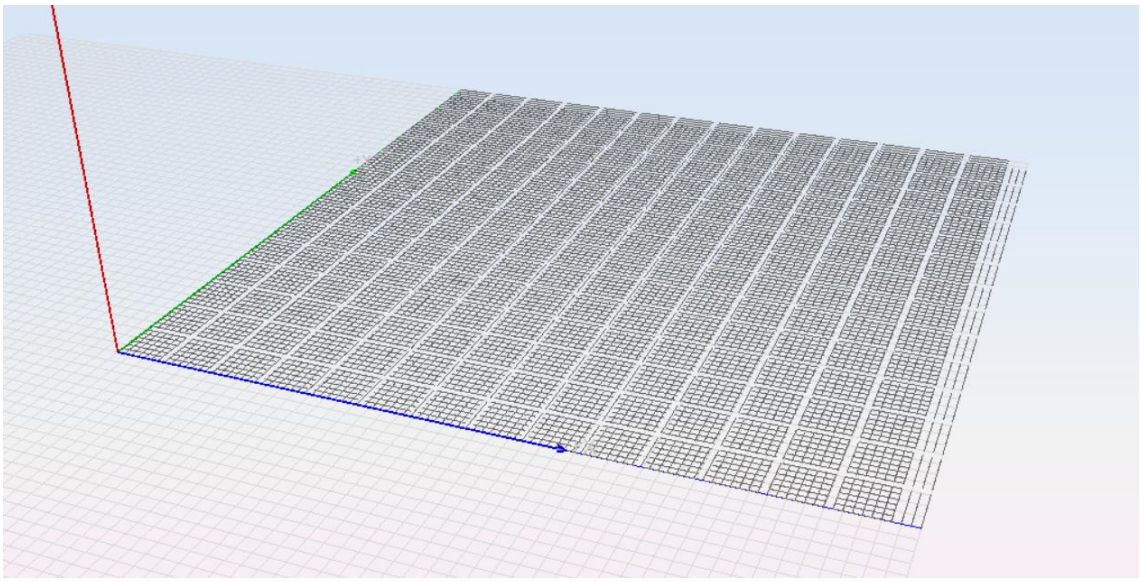


Ilustración 6.2.7 Simulación (rotación), geometría

Como se puede apreciar en la imagen, vemos que las tiras de la primera capa se encuentran direccionadas de forma paralela al eje Y, mientras que las de la segunda capa tienen una dirección perpendicular a las primeras (paralelas al eje X).

Así que, para ver como consigue nuestra geometría filtrar ambas polarizaciones de onda, lo que haremos será realizar las simulaciones con las direcciones de observación de la misma forma que antes (corte en 'phi' de 90º y 'theta' en 0 y 180º), polarizando el campo en un primer caso en la dirección del eje X, y después del eje Y, consiguiendo así ver si las tiras filtran de igual modo ambas polarizaciones.

De este modo, los parámetros de simulación serán los siguientes:

- **Frecuencias:**
  - Barrido de frecuencias.
  - 0,5GHz – 3GHz
- **Direcciones de observación:**
  - Corte en Phi = 90
  - Theta inicial = 0
  - Incremento = 180
  - Muestras = 2
  - Theta final = 180
- **RCS:**
  - Monostático
  - Polarización de onda:
    - Caso 1: Etheta = 1 y Ephi = 0
    - Caso 2: Etheta = 0 y Ephi = 1
- **Mesh:**
  - "Planar faces" = 10
  - "Curved faces" = 10
  - Bandas por octavo = 1

- Procesadores = 16

- **Solver:**
  - “Method of Moments” (MoM)
  - “Architecture Strategy”: OpenMP
  - “Solver Functions”: Subdominios
  - Error relativo = 0.01
  - Máximo número de iteraciones = 5000

Las gráficas obtenidas para ambos casos son las siguientes:

Caso 1:

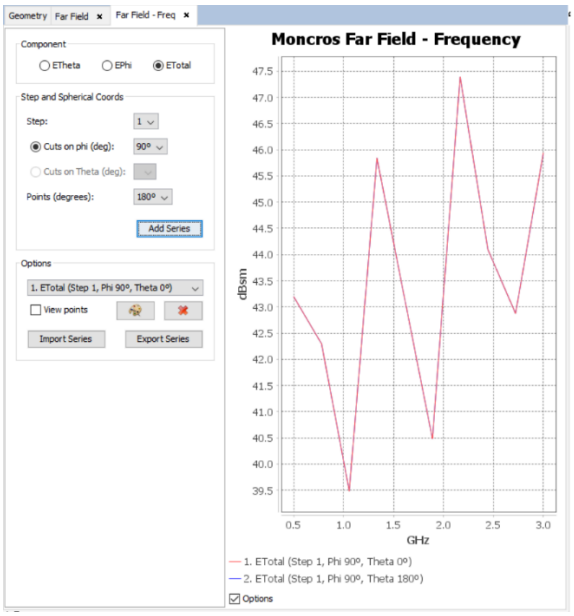


Ilustración 6.2.8 Simulación (rotación), gráfica valor de campo respecto de la frecuencia (caso 1)

Caso 2:

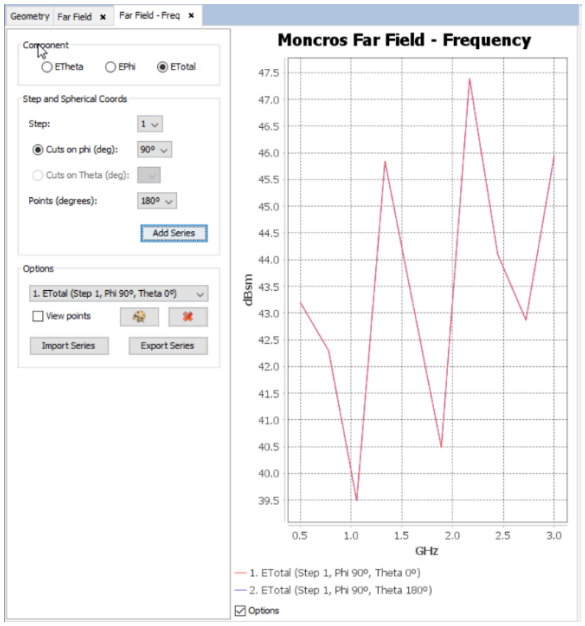


Ilustración 6.2.9 Simulación (rotación), gráfica valor de campo respecto de la frecuencia (caso 2)

Podemos observar que ambas gráficas son exactamente iguales, por lo que nuestra geometría filtra por igual las dos polarizaciones de onda.

Para hacer unas últimas comprobaciones, generaremos dos geometrías más, viendo a ver cómo varía el campo añadiendo una mayor densidad de tiras, y después, con disminuyéndola. La geometría será exactamente igual, lo único que variaremos es que primero, dimensionamos la separación entre tiras con 30mm y, en el segundo caso, con 90mm, consiguiendo de este modo, un total de 2460 tiras por capa para el primer caso y 728 tiras por capa para el segundo.

Las geometrías visualmente son tal que así:

Geometría 1:

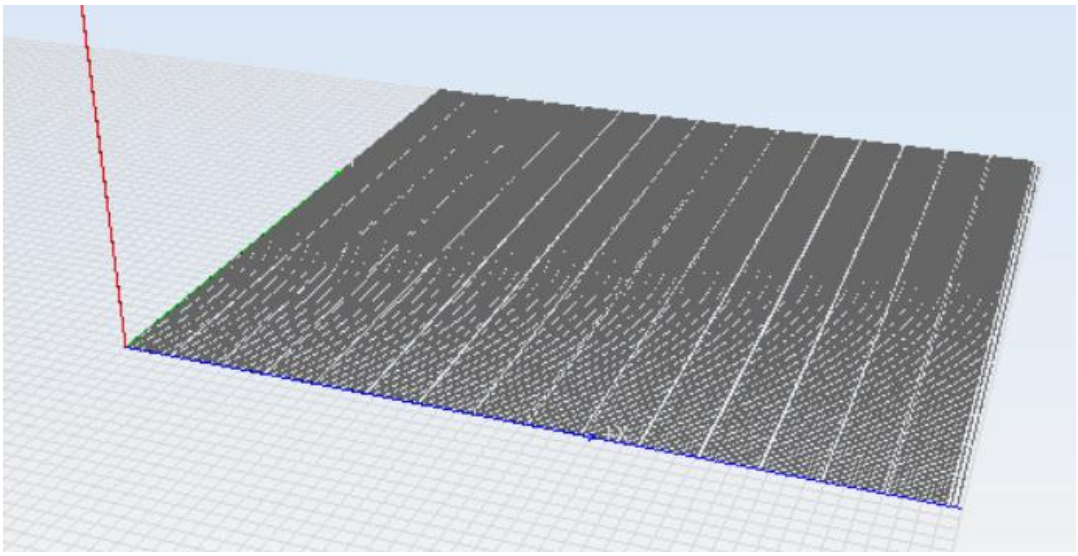


Ilustración 6.2.10 Simulación (rotación), geometría (1)

Geometría 2:

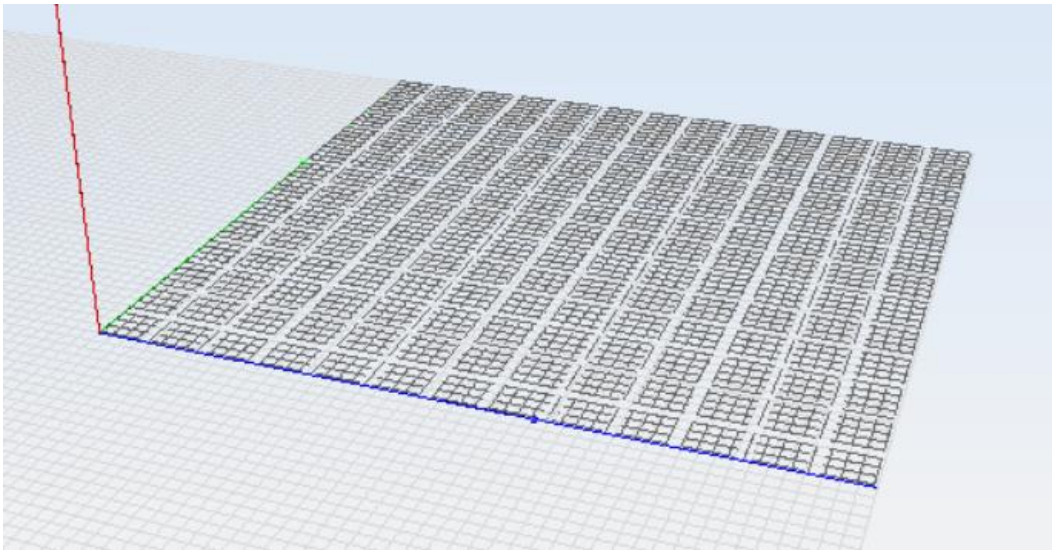


Ilustración 6.2.11 Simulación (rotación), geometría (2)

Y sus gráficas respectivamente son las siguientes:

Gráfica geometría 1:

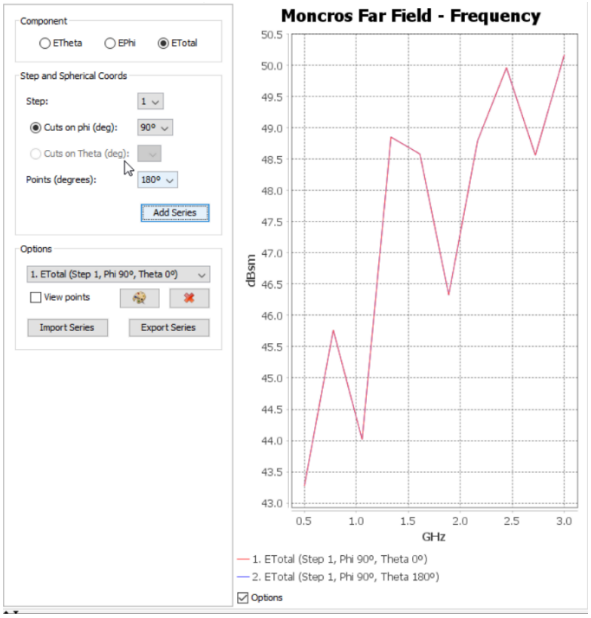


Ilustración 6.2.12 Simulación (rotación), gráfica valor de campo respecto de la frecuencia (geometría 1)

Gráfica geometría 2:

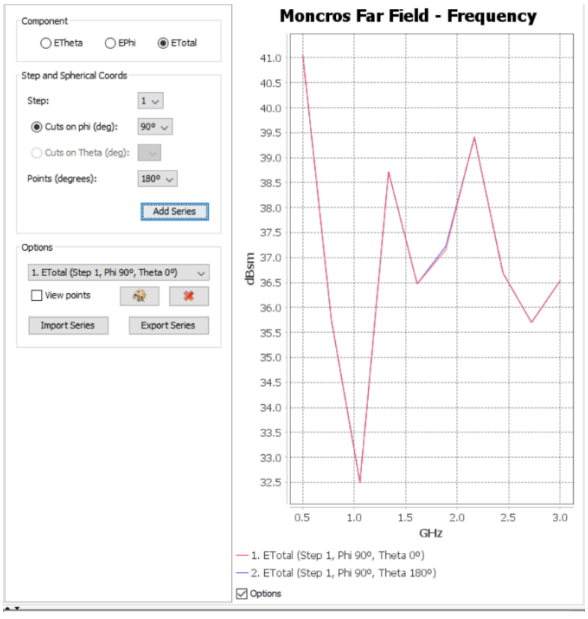


Ilustración 6.2.13 Simulación (rotación), gráfica valor de campo respecto de la frecuencia (geometría 2)

Tal y como esperábamos, al aumentar la densidad de tiras, aumenta el valor de campo, y al disminuir, este disminuye a su vez, por lo que las geometrías generadas son buenas para simular.

7. Conclusiones y futuras líneas de trabajo

En este último apartado, se mostrarán las conclusiones extraídas en la realización de este proyecto, al mismo tiempo que también se expondrán unas sugerencias de cómo este trabajo podría ser desarrollado en un futuro, ampliando así lo ya realizado hasta el momento y completando las carencias que se hayan podido tener.

7.1. Conclusiones

Las conclusiones que se han podido llegar en este este Trabajo de Fin de Grado son, en primera instancia, la realización con éxito de la aplicación deseada desde un primer momento para la generación de superficie recubiertas de material absorbente al radar mediante parámetros estadísticos, suponiendo de esta forma un avance a la hora de realizar simulaciones con softwares de simulación electromagnéticas como es *NewFASANT.*

Esto también llega a sacar la segunda conclusión, y es que el alumno, para conseguir este objetivo propuesto, ha obtenido conocimientos en el uso del software *NewFASANT*, que es donde se han realizado los modelos geométricos generados por la aplicación para su posterior simulación, del mismo modo que también el alumno ha conseguido nuevos conocimientos acerca del entorno de programación de *MATLAB*, conocimientos que ya poseía desde un principio, pero que se han visto ampliado en el desarrollo de este proyecto.

En el ámbito de los resultados obtenidos, cabe destacar que, como se ha comprobado en el punto anterior, se ha logrado de manera satisfactoria que el modelado geométrico creado por parte del usuario a partir de la aplicación corresponda con los datos que ha introducido previamente, haciendo que su posterior simulación sea la correcta respecto a lo esperado.

Estas simulaciones se han realizado bajo una serie de parámetros estadísticos, haciendo que la geometría no responda de una manera totalmente artificial y cuadriculada, sino de forma que los elementos resistivos que se superpongan encima del plano base lo hagan de un modo aleatorio, consiguiendo así que los escenarios generados sean lo más realistas posibles y que los resultados de la simulación que se obtengan, sean los más prácticos posibles.

De este modo, a la conclusión que se puede llegar en cuanto a la generación de la herramienta para la reproducción de estos escenarios geométricos, es que la creación de la aplicación aquí documentada proporciona al usuario una experiencia más eficiente a la hora de poder simular este tipo de modelados, siendo los resultados conseguidos, por tanto, realmente satisfactorios.

7.2. Futuras líneas de trabajo

Como último apartado, se ha querido mostrar una serie de ideas y sugerencias por parte del autor para el avance y ampliación del proyecto en cuestión. La aplicación creada para reproducción de escenarios en *NewFASANT* supone ya de por sí un avance en el campo de simulación de este software, ya que, como se ha mencionado en las conclusiones, hace que el usuario realice de un modo más eficiente una serie de simulaciones que pueden ser llevadas a la práctica a posteriori, pero esto no significa que sea una aplicación que ya se encuentre completa y que este proyecto, por tanto, este cerrado a nuevas ideas.

Así pues, algunas de las sugerencias planteadas por el autor para la continuación de este trabajo son, por ejemplo, la incorporación de nuevos tipos de geometría que puedan superponerse al plano base, como por ejemplo una serie de anillos.

Estos anillos proporcionarían nuevas propiedades a nuestro escenario, haciendo de él un entorno más complejo para la simulación, pero a su vez más interesante para el estudio. Esta geometría puede encontrarse de forma única sobre el plano base, sustituyendo así a las tiras que habíamos colocado antes como elementos resistivos, o se pueden encontrar de manera intercalada con estas, dejándolo así a elección del usuario. Este ejemplo de nueva geometría se encontraría definido del mismo modo que las anteriores, también mediante parámetros estadísticos.

Otra de las ideas que también puede resultar interesante es la opción de que el usuario pueda escoger la forma de colocar las tiras sobre el plano base, siendo la definición de los parámetros ya indicados hasta el momento (tamaño del plano base y tiras, separación entre tiras y orientación) una de entre varias opciones que tenga el usuario. Esto quiere decir que en lugar de colocar una tras otra las tiras como se ha hecho en este proyecto, el usuario podría definir un parámetro como puede ser la definición del número de tiras que quiere sobre el plano base y colocar ese número de tiras exactos sobre este, sin importar cómo se encuentren orientadas o separadas entre sí. Un ejemplo gráfico es que fuese como si el usuario tuviese un número de tiras en concreto en sus manos y las tirase sobre una superficie sin importar como se encuentren.

Esto lleva de manera implícita a una última idea, y es la ampliación de la aplicación en cuanto a más opciones y pantallas distintas para la configuración tanto de más parámetros geométricos como se han mencionado ya previamente, o también para la configuración por parte del usuario de la propia aplicación, para conseguir, de esta forma, una experiencia aún más sencilla y satisfactoria.

Anexos

Función de la interfaz gráfica de usuario:

```
function varargout = GUIDE_TFG(varargin)
% GUIDE_TFG MATLAB code for GUIDE_TFG.fig
%      GUIDE_TFG, by itself, creates a new GUIDE_TFG or
raises the existing
%      singleton*.
%
%      H = GUIDE_TFG returns the handle to a new GUIDE_TFG
or the handle to
%      the existing singleton*.
%
%      GUIDE_TFG('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in GUIDE_TFG.M with the
given input arguments.
%
%      GUIDE_TFG('Property','Value',...) creates a new
GUIDE_TFG or raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before GUIDE_TFG_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes
property application
%      stop.  All inputs are passed to GUIDE_TFG_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
GUIDE_TFG

% Last Modified by GUIDE v2.5 24-Oct-2020 13:20:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUIDE_TFG_OpeningFcn,
...
                   'gui_OutputFcn',  @GUIDE_TFG_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
```

```
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUIDE_TFG is made visible.
function GUIDE_TFG_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to GUIDE_TFG (see
VARARGIN)

axes(handles.axes1);
[x,map]=imread('plano base.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes2);
[x,map]=imread('ancho.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes3);
[x,map]=imread('largo.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes4);
[x,map]=imread('tiras.PNG');
image(x)
colormap(map);
axis off
```

```
axes(handles.axes5);
[x,map]=imread('ancho.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes6);
[x,map]=imread('largo.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes7);
[x,map]=imread('separacion.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes8);
[x,map]=imread('orientacion.PNG');
image(x)
colormap(map);
axis off
hold on

axes(handles.axes9);
[x,map]=imread('capas.PNG');
image(x)
colormap(map);
axis off

% Choose default command line output for GUIDE_TFG
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUIDE_TFG wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function varargout = GUIDE_TFG_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13
as text
%        str2double(get(hObject,'String')) returns contents
of edit13 as a double

nombre_archivo = get(hObject,'String');

% --- Executes during object creation, after setting all
properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
checkbox2
```

```
concatenar_script = get(hObject,'Value');

function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11
as text
%        str2double(get(hObject,'String')) returns contents
of edit11 as a double

altura = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12
as text
%        str2double(get(hObject,'String')) returns contents
of edit12 as a double
sep_capas = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit9 as
text
%        str2double(get(hObject,'String')) returns contents
of edit9 as a double

orientacion_media = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
```

```
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10
as text
%        str2double(get(hObject,'String')) returns contents
of edit10 as a double

orientacion_desv = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as
text
%        str2double(get(hObject,'String')) returns contents
of edit7 as a double

sep_media = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called
```

```
% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as
text
%        str2double(get(hObject,'String')) returns contents
of edit8 as a double

sep_desv = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as
text
%        str2double(get(hObject,'String')) returns contents
of edit3 as a double
```

```
ancho_tira_media = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as
text
%        str2double(get(hObject,'String')) returns contents
of edit4 as a double

ancho_tira_desv = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as
text
%        str2double(get(hObject,'String')) returns contents
of edit5 as a double

largo_tira_media = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as
text
%        str2double(get(hObject,'String')) returns contents
of edit6 as a double

largo_tira_desv = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as
text
%        str2double(get(hObject,'String')) returns contents
of edit1 as a double
ancho_base = str2double(get(hObject,'String'));


% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit2 as
text
%        str2double(get(hObject,'String')) returns contents
of edit2 as a double

largo_base = str2double(get(hObject,'String'));


% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
checkbox1

dibuj_pb = get(hObject,'Value');

% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
checkbox3

rotar_capa = get(hObject,'Value');
```

```
function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15
as text
%        str2double(get(hObject,'String')) returns contents
of edit15 as a double

grados_rot_capa = str2double(get(hObject,'String'));

% --- Executes during object creation, after setting all
properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

msgbox('El dato introducido indicará el valor en el eje z
donde se situará la capa de tiras','Ayuda');


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
ancho_base = str2double(get(handles.edit1,'String'));
largo_base = str2double(get(handles.edit2,'String'));
ancho_tira_media =
str2double(get(handles.edit3,'String'))*0.000001;
ancho_tira_desv =
str2double(get(handles.edit4,'String'))*0.000001;
largo_tira_media =
str2double(get(handles.edit5,'String'))*0.01;
largo_tira_desv =
str2double(get(handles.edit6,'String'))*0.01;
sep_media = str2double(get(handles.edit7,'String'))*0.001;
sep_desv = str2double(get(handles.edit8,'String'))*0.001;
orientacion_media =
str2double(get(handles.edit9,'String'));
orientacion_desv =
str2double(get(handles.edit10,'String'));
altura = str2double(get(handles.edit11,'String'))*0.001;
nombre_archivo = get(handles.edit13,'String');
grados_rot_capa = str2double(get(handles.edit15,'String'));
dibuj_pb = get(handles.checkbox1,'Value');
concatenar_script = get(handles.checkbox2,'Value');
rotar_capa = get(handles.checkbox3,'Value');

separacion_a = normrnd(sep_media,sep_desv);
separacion_l = normrnd(sep_media,sep_desv);
n_plano = 0;
ancho_tira =
abs(normrnd(ancho_tira_media,ancho_tira_desv));
largo_tira =
abs(normrnd(largo_tira_media,largo_tira_desv));
i=ancho_tira;
j=largo_tira;
x=0;
y=0;
angulo = normrnd(orientacion_media,orientacion_desv);
altura_rot = altura+1;
n_plano_altura = altura*1e6;
x_pbase = -(largo_tira_media+3*largo_tira_desv);
y_pbase = -(largo_tira_media+3*largo_tira_desv);
ancho_pbase =
ancho_base+2*(largo_tira_media+3*largo_tira_desv);
largo_pbase =
largo_base+2*(largo_tira_media+3*largo_tira_desv);

% Abrimos el fichero destruyendo lo que ya existe
if concatenar_script == 0
    fileID = fopen(nombre_archivo,'w+');
else
    fileID = fopen(nombre_archivo,'a+');
end
```

```
if dibuj_pb == 1
    str_base = sprintf("plane -n planobase -p %d %d 0 %g
%g",x_pbase,y_pbase,ancho_pbase,largo_pbase);
    str_BASE = strjoin([str_base])
    fprintf(fileID,"%s\n", str_BASE);
end


while i <= ancho_base & j <= largo_base

    if ancho_base == 0 || largo_base == 0 || ancho_tira ==
0 || largo_tira == 0
        errordlg('Las dimensiones del plano base y las
tiras tienen que ser distinto de 0','Error');
        i=1;
        j=1;
    else

        str_tira = sprintf("plane -n plano%d_%d -p %g %g %g
%g
%g",n_plano,n_plano_altura,x,y,altura,ancho_tira,largo_tira
);
        str_TIRA = strjoin([str_tira])
        fprintf(fileID,"%s\n", str_TIRA);

        angulo =
normrnd(orientacion_media,orientacion_desv);

        str_orientacion = sprintf("rotate -s plano%d_%d -p
%g %g %g %g %g %g
%g",n_plano,n_plano_altura,x,y,altura,x,y,altura_rot,angulo
);
        str_ORIENTACION = strjoin([str_orientacion])
        fprintf(fileID,"%s\n",str_ORIENTACION);

        if rotar_capa == 1
            str_rotacioncapa = sprintf("rotate -s
plano%d_%d -p %g %g %g %g %g %g
%g",n_plano,n_plano_altura,ancho_base/2,largo_base/2,altura
,ancho_base/2,largo_base/2,altura_rot,grados_rot_capa);
            str_ROTACIONCAPA = strjoin([str_rotacioncapa])
            fprintf(fileID,"%s\n",str_ROTACIONCAPA);
        end

        separacion_a = normrnd(sep_media,sep_desv);
        separacion_l = normrnd(sep_media,sep_desv);
        n_plano = n_plano+1;
        i = i+ancho_tira+separacion_a;
        x = x+ancho_tira+separacion_a;

        if i>ancho_base
```

```
            j = j+largo_tira+separacion_l;
            y = y+largo_tira+separacion_l;
            i = ancho_tira;
            x = 0;
        end

        ancho_tira =
abs(normrnd(ancho_tira_media,ancho_tira_desv));
        largo_tira =
abs(normrnd(largo_tira_media,largo_tira_desv));

    end
end

% Cerramos el fichero
fclose(fileID);
```

Bibliografía

[1] Dr. Ezequiel del Río, “Electrostática”, notas de la asignatura Física II, Universidad Politécnica de Madrid, 2017

[2] José Villasuso Gato, Ondas Electromagnéticas [Online] Consultado en: http://teleformacion.edu.aytolacoruna.es/FISICA/document/fisicaInteractiva/Ondasbachillerato/ondasEM/ondasEleMag_indice.htm

[3] Steven Rojas (12 de noviembre, 2019), Polarización de Ondas Electromagnéticas [Online] Consultado en: https://medium.com/@steven98.sr/polarizaci%C3%B3n-de-ondas-electromagn%C3%A9ticas-67bce51a203d

[4] Álvaro Botrán Díaz “Desarrollo de un Material Absorbente de Radiaciones Electromagnéticas en el Espectro” Trabajo Fin de Grado, Universidad Carlos III, Leganés, España, 2015

[5] Rosa Echevarría Líbano, “Breves Apuntes de MATLAB”, Departamento Ecuaciones Diferenciales y Análisis Numérico, Universidad de Sevilla

[6] M.ª Cristina Casado Fernández, “Manual Básico de MATLAB”, Servicios Informáticos, Universidad Complutense de Madrid

[7] (29 de enero, 2020), Portal de Comunicación, Universidad de Alcalá [Online] Consultado en: http://portalcomunicacion.uah.es/diario-digital/actualidad/la-compania-norteamericana-altair-adquiere-newfasant-ebt-de-la-universidad-de-alcala-2-2.html

[8] Página oficial de NewFASANT [Online] Consultado en: https://www.fasant.com/es/about-newfasant

[9] José Luis García “Diseño e implementación de un sistema de simulación electromagnética en remoto aplicado al análisis de antenas” Trabajo Fin de Grado, Universidad de Alcalá, Alcalá de Henares, España

[10] Francisco José García Peñalvo, Alicia García Holgado, “Fundamentos de la vista de casos de uso”, Departamento de Informática y Automática, Universidad de Salamanca

[11] “Diagramas del UML”, Cátedra de Proyecto [Online] Consultado en: https://www.teatroabadia.com/es/uploads/documentos/iagramas_del_uml.pdf

Lenguaje de script:

[12] Daniel Rodríguez, “Lenguajes de script” [Online] Consultado en: http://personal.cimat.mx:8181/~amor/Academic/Books/Tutorial_HTML/script.html

[13] “Significado de script” [Online] Consultado en: https://www.significados.com/script/

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá