A continuación, se explicarán los scripts creados que hacen uso de la toolbox estudiada en el apartado anterior para poder ejecutar todas las simulaciones que se llevan a cabo en este proyecto.

*lanzador.m*

Este es el script principal. Es una función a la que se le pueden pasar los siguientes parámetros: *ejecucion*, *caso* y *clusters\_fijados*. El uso de esta función es:

***lanzador***(*ejecucion, caso, clusters\_fijados*) (3.3.1)

La explicación de cada uno de los valores que puede recibir la función se puede encontrar en la siguiente tabla.

|  |  |
| --- | --- |
| **Variables de la función *lanzador*** | |
| **Variable** | **Explicación** |
| *ejecucion* | Equivale a la realización del caso y su único fin es el de poder identificar dicha realización cuando se guarden las variables en un fichero, pues el salvado de estas variables se lleva a cabo en cada realización |
| *caso* | Caso a simular |
| *clusters\_fijados* | Parámetro que establece la cantidad de clusters que se desea establecer en el escenario |

Tabla 3.3.1. Variables de la función *lanzador*.

Cabe mencionar que en todo momento, la máxima cantidad de clusters es igual a la cantidad de sensores en el escenario; sin embargo, *clusters\_fijados* permitirá elegir cuántos clusters se desea que sean dispuestos. Si *clusters\_fijados* es igual a 0, se le estará indicando a la simulación que optimice la cantidad de clusters, en lugar de que venga fijada.

Dado el *caso* introducido, el script contiene gran cantidad de condiciones para poder identificar cada uno de estos casos. Las diferencias entre ellos serán las distintas características que diferencian los escenarios que se desean simular: máxima carga de batería para todos los sensores o carga aleatoria para todos ellos, sensores fijos a lo largo del tiempo o posiciones aleatorias, la dirección en la que se desea radiar (*ϕ* y *θ*), los pasos de integración *deltheta*  y *delphi*, la cantidad de generaciones y población, el tiempo de simulación, etc. El tiempo de simulación es un entero que representa la cantidad de tiempo durante el cual se simulará el escenario en caso de que los nodos sean móviles.

Cabe mencionar que en una misma simulación, se pueden llevar a cabo distintos experimentos. Mediante varios bucles *for* situados en el script *bucle\_principal.m*, se recorrerán los vectores *antenas* (que contiene el tipo de antena a usar), *phi* (que contiene el ángulo *ϕ* de la dirección en la que se desea radiar) y *theta* (que contiene el ángulo *θ* de dicha dirección). En caso de radiar a estaciones base o nodos enemigos, el script recorrerá los vectores *estaciones\_base* o  *nodos\_enemigos* en lugar de *theta* y *phi*, pues *theta* y *phi* corresponden a direcciones de radiación genéricas. De este modo, mediante las combinaciones de las componentes de los vectores mencionados se dará lugar a distintos experimentos, correspondientes a cada *caso*.

Este script se compilará mediante la herramienta *Deploytool* de Matlab, que lo convertirá en un archivo ejecutable, con extensión .*exe*. Gracias a tener todo el código compilado de esta forma, se conseguirá que, mediante la introducción de parámetros, se puedan ejecutar cientos o incluso miles de simulaciones en paralelo en múltiples instancias en distintos ordenadores. El principal motivo para llevar a cabo las simulaciones de esta manera es conseguir gran cantidad de realizaciones de un mismo experimento, en el mismo tiempo que tardaría en ejecutarse una sola realización.

Una vez identificado mediante el script el caso que se desea simular, se le pasarán las variables específicas de ese caso al script *casos.m*.

*casos.m*

Este script se encarga de “traducir” algunas de las variables del caso en variables que puedan utilizar las siguientes funciones. Por ejemplo,  si la dimensión del caso a tratar es 2, *casos* establecerá el tamaño del escenario, dando los valores máximos permitidos en el eje X e Y, estableciendo el valor máximo del eje Z a cero, pues se requiere que los nodos del escenario se sitúen en un plano (2D).

Tras esto, casos llamará al script *bucle\_principal.m*, que se encargará del resto. Cuando *bucle\_principal* termine, casos guardará las variables como:

[*caso\_X\_ejecucion\_Y\_clusters\_fijados\_Z.mat*](http://caso_X_ejecucion_Y_clusters_fijados_Z.mat) (3.3.2)

*bucle\_principal.m*

Este script será el encargado de disponer el escenario, llamar al algoritmo de optimización, calcular el tiempo de vida de los sensores del escenario y el tiempo de vida de los sensores que no usen beamforming. Con estos dos últimos datos, calculará la eficiencia dividiéndolos.

En primer lugar, se definen algunas variables necesarias para la correcta ejecución, como *range\_config* y la frecuencia de trabajo de las antenas. XXX

En segundo lugar, se asignan las posiciones que tomarán los sensores en el escenario, dependiendo del tamaño del mismo, que vendrá dado por la dimensión del caso a simular y de los parámetros establecidos en el script *casos.m*. De este modo, para calcular la posición de los sensores, se tendrá en cuenta la dimensión del escenario, pero también se tendrá en cuenta si los sensores adquirirán posiciones fijas a lo largo del tiempo de simulación o posiciones móviles. En este segundo caso, se llamará al script *movilidad.m*, que devolverá las posiciones de los nodos en movimiento en función del tiempo. En dicho script, se puede editar la velocidad de los sensores, la probabilidad de que estén parados, etc.

En tercer lugar, se asignará el porcentaje de batería a cada uno de los sensores. A todos se les asignará el 100% de la batería en caso de que la variable *potencia\_fija* tenga asignado el valor 1 en el script *casos*.*m*; en cambio, la potencia de los sensores será aleatoria en caso de que esa variable tenga asignado el valor 0.

Tras estas asignaciones, comenzará el bucle principal de la simulación. Este bucle principal consiste en cuatro bucles *for* distintos: el primero, recorre el tipo de antena del vector antenas; el segundo, recorre la componente *θ* del vector *theta*; el tercero, la componente *ϕ* del vector *phi*; el cuarto, recorrerá el tiempo de simulación, en caso de que sea superior a 1. En caso de estaciones base o nodos enemigos, no se recorrerán los vectores *theta* ni *phi*, sino *estaciones\_base* o ­*nodos\_enemigos*.

En el bucle principal, se inicializa la variable *array\_config* con las posiciones de los sensores en el escenario y su orientación. Tras esto, el script *optimiza.m* utilizará las distintas variables ya establecidas, devolviendo la variable *array\_config* rellena con los valores de amplitud y fase optimizados para cada uno de los nodos. En esa función *optimiza.m*, se llama al algoritmo genético para poder aumentar la directividad en la dirección deseada, es decir, realiza la optimización que se pretende conseguir en este proyecto.

De este modo, una vez devuelta la variable *array\_config* con los valores optimizados de fase y amplitud, se calculará el tiempo de vida de los sensores, calculando en primer lugar la potencia radiada y la potencia que se recibiría en la dirección deseada. Además de esos valores, el script *optimiza.m* devolverá la cantidad de clusters resultado de la optimización (en caso de que *clusters\_fijados* sea igual a 0), qué sensores pertenecen a cada cluster y el centro del cluster de cada uno. De este modo, el bucle principal hará los cálculos de los tiempos de vida de cada sensor dependiendo de si pertenecen al cluster en cuestión.

Tras calcular el tiempo de vida de cada uno de los sensores, solo se tendrá en cuenta el mínimo para poder obtener el valor de la eficiencia al usar beamforming. Tras esto, se calculará el tiempo de vida de los sensores que no usan beamforming para poder realizar la comparación. Estos cálculos se llevarán a cabo sensor por sensor, y no en conjunto, como los cálculos del beamforming. Finalmente, se tomará para hacer la comparación el menor tiempo de vida.

Tras esto, se guardarán las variables. Ha de tenerse en cuenta que en caso de guardarse todas las variables (incluidos el campo E y P de cada cluster y para cada estación, además del campo de cada sensor de la comparación), el peso de cada archivo *.mat* puede llegar a pesar cientos de MB, dependiendo de la cantidad de sensores. Debido a que las variables del campo resultarán de menor importancia a la hora de analizar los resultados, podemos prescindir de ellas y no guardarlas, pues son las que hacen que el peso de los archivos se eleve tanto. Con este borrado, las variables ocuparán menos de 50 KB. En caso de necesitar estas variables para poder representar el campo en forma de diagramas de radiación, puede obtenerse fácilmente a partir de la variable *array\_config* guardada.

*optimiza.m*

El script *optimiza.m* se encargará de tomar todas las variables necesarias para establecer adecuadamente los parámetros de los algoritmos genéticos. Como resultado, rellenará la variable *array\_config* con las amplitudes y fases óptimas para cada antena.

Las variables que utiliza son la variable *array\_config* inicializada con las posiciones de los sensores, las posiciones (X, Y, Z) de todos los sensores (para el algoritmo de clusters), la cantidad de generaciones y la población, la variable *clusters\_fijados*, la carga de batería de los sensores, la frecuencia de funcionamiento y la dirección de radiación deseada.

En primer lugar, el algoritmo decidirá cuántas variables se han de introducir en los algoritmos genéticos: “2*X*” (*X* es la cantidad de sensores) si *clusters\_fijados* es distinto de 0 o “2*X*+1” si *clusters\_fijados* es igual 0, pues será necesaria una variable más para optimizar la cantidad de clusters. El vector que contiene todas las variables será el vector *x*, que en un principio se inicializa con valores aleatorios normalizados. Los *X* primeros valores serán las amplitudes; los siguientes *X* serán las fases y, en caso de existir, la última variable representará la cantidad de clusters. Tras esto, se establecen los valores máximos y mínimos que ha de alcanzar cada variable. Se consideran normalizadas, de modo que estos límites serán 0 y 1 para todas ellas.

Una vez establecidas las variables de entrada, se han de seleccionar los parámetros de los algoritmos genéticos, que son la cantidad de generaciones, la población, mutación uniforme con probabilidad 0.1, selección por restos y cruzamiento heurístico con probabilidad de 0.8, como se estudió en el apartado “2.3.3. Adaptación de los algoritmos genéticos al proyecto”. Entonces, se le pasará el vector *x* a la función llamada *funcion\_de\_busqueda*, que usará las mismas variables que utiliza *optimiza.m*. Esta función devolverá el vector *x* con las variables ya optimizadas.

Finalmente, solo resta obtener los valores de las variables y transformarlos a unidades conocidas (amplitudes en dB y fases en radianes) e introducirlas en la variable *array\_config*, que será devolverá el script *optimiza.m*.

*funcion\_de\_busqueda.m*

Esta función recibirá una nueva variable *x* distinta en casa nueva generación del algoritmo genético y devolverá una variable llamada *scores*, que se explicará más adelante. Su uso es el siguiente:

*scores* = ***funcion\_de\_busqueda*** (*x, array\_config, carga\_baterias, theta, phi*) (3.3.3)

En primer lugar, la función obtiene de la variable x los distintos valores: amplitudes, fases y cantidad de clusters, si es que se ha seleccionado la variable *clusters\_fijados* igual a 0. Tras esto, creará una variable *array\_config* con estos valores y con las posiciones de los sensores. Se calcularán los clusters mediante la función *kmeans* de MATLAB, que devolverá el centro de cada cluster y un vector que establce qué sensor pertenece a cada cluster.

Después, se itera en función de la cantidad de clusters y en función de la cantidad de estaciones (si es que se están considerando estaciones base en lugar de direcciones de radiación genéricas) o en función de la cantidad de direcciones genéricas. En cada iteración se calcula la distancia y la dirección a la estación base, se llama a la función *calcula\_campo* para poder obtener la directividad en la dirección deseada, la potencia recibida y el tiempo de vida de cada sensor. Entonces, se calcula el menor tiempo de vida que se ha obtenido (de todos los clusters). Esta variable, tal y como se estudió en el “2.3.3. Adaptación de los algoritmos genéticos al proyecto”, es la variable que se ha de optimizar.

Finalmente, se realiza:

*scores* = 1/*min*(*min*(tiempo de vida por cluster)) (3.3.4)

de forma que esta función devuelva este valor a los algoritmos genéticos. Así, *scores* es un valor de referencia para el algoritmo genético que le indicará la mejora o el empeoramiento de la optimización en cada generación.

Cabe mencionar que en caso de considerar nodos enemigos, tan solo es necesario calcular la directividad para obtener el tiempo de vida de los sensores. En ese caso, la variable *scores* tendrá el valor:

*scores* = *max*(*min*(tiempo de vida por cluster)) (3.3.5)

En caso de combinar estaciones base y nodos enemigos, el valor de *scores* será:

*scores* = 1/*min*(*min*(tiempo de vida por cluster) +

Σ(*min*(tiempo de vida por cluster)\*0.01 (3.3.6)

Cabe mencionar que el factor 0.01 se ha obtenido de forma experimental.