

TRABAJO FIN DE GRADO

INGENIER´IA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

# Estudio de redes colaborativas/oportunistas para redes de comunicación inalámbrica

### Autor

Pablo Ráez Sánchez

### Directores

Pablo Padilla de la Torre



Escuela Te´cnica Superior de Ingenier´ıas Informa´tica y de Telecomunicacio´n

—

Granada, Septiembre de 2015

3



# Estudio de redes colaborativas/oportunistas para redes de comunicación inalámbrica

### Autor

Pablo Ráez Sánchez

### Directores

Pablo Padilla de la Torre

**Estudio de redes colaborativas/oportunistas para redes de comunicación inalámbrica**

Pablo Ráez Sánchez

**Palabras clave**: red colaborativa, inalámbrico, canal de propagación, nodo, transmisión, recepción, cobertura, salto, coste, ruta, encaminamiento, eficiencia energética, Matlab, simulador

### Resumen

En el presente proyecto se hace un estudio sobre el uso de redes colaborativas u oportunistas en las comunicaciones inalámbricas. La principal característica de la redes colaborativas es que se aprovecha la comunicación entre los distintos nodos que las forman con el objetivo de mejorar su eficiencia en algún aspecto.

Adía de hoy, las comunicaciones inalámbricas forman parte de nuestro día a día. Gran parte de los aparatos que nos rodean, como teléfonos móviles, tablets u ordenadores portátiles hacen uso de algunos estándares de comunicación inalámbrica, como pueden ser WiFi, 3G, 4G o Bluetooth. Otro lugar en el que se utilizan últimamente es en redes inalámbricas de sensores (en inglés WSN), y de cara al futuro son la base de las redes vehiculares.

Existen muchos retos que los investigadores y profesionales intentan superar en lo referente a mejorar este tipo de comunicaciones, como, por ejemplo aumentar las tasas de transferencia o mejorar tiempos de acceso. Otro reto es el de mejorar la eficiencia energética, que es algo muy importante ya que los equipos que se utilizan suelen tener una fuerte dependencia de la batería. Con lo cual, mejorar en este aspecto repercute en la autonomía de los dispositivos, además de ser más respetuoso con el medio ambiente.

Es por esto que este estudio está centrado en la eficiencia energética en las comunicaciones inalámbricas, y para ello se hace uso de las redes colaborativas. Se aprovecha el hecho de que este tipo de redes sean multi-salto, lo cual hace que los nodos transmitan a distancias más cortas, siendo así su consumo energético derivado en potencia de transmisión menor. Esto es así ya que la potencia de las señales que se transmiten decae con dependencia de la distancia recorrida al cuadrado, lo que implica que para hacer transmisiones a largas distancias suponga un gran consumo energético en transmisión.

Con el objetivo de poder analizar las redes, el primer paso que se establece es la implementación de un simulador, en el cual se puedan definir modelos de canal, nodos personalizables, y algunos algoritmos de encaminamiento. También es interesante añadir movilidad a los nodos, para enriquecer la variedad de escenarios que se pueden simular. Con esto hecho, ya se pueden hacer simulaciones de escenarios concretos, en las cuales se pueda analizar de forma precisa lo que sucede con las señales que se transmiten en el entorno. Además, también es interesante realizar varias simulaciones en las cuales varíe un parámetro concreto, de forma que se obtengan conclusiones de cómo afecta dicho parámetro a las comunicaciones.

Para la implementación de este simulador se utiliza la plataforma Matlab, la cual es de un uso muy extendido en el ámbito de la ciencia y la ingeniería. El hecho de usar esta plataforma es que está especialmente diseñada para hacer cálculos matemáticos. Además, la creación de interfaces gráficas de usuario es relativamente sencilla en comparación con otros leguajes, y existe una gran comunidad y proyectos realizados con anterioridad, de modo que se pueden aprovechar ciertos recursos, y es fácil encontrar ayuda para salir de algún problema que haya provocado que el desarrollo se quede estancado.

## RESEARCH OF MULTIHOP NETWORKS ON WIRELESS COMMUNICATIONS

Pablo Ráez Sánchez

**Keywords**: multihop network, Wireless, propagation channel, node, transmission,reception, coverage, hop, cost, path, routing, energetic efficiency, Matlab, simulator

### Abstract

In this project a study on the use of multihop networks in wireless communications is made. The main feature of the multihop networks is that the communicationis carried out between nodes,, looking for achive some improvements in comparison with classics end to end communications. This study is focused on the energy consumption of this kind of networks.

Nowadays, wireless communications are an essential part of our daily life. Much of the devices around us, such as mobile phones, tablets or laptops make use of some wireless communication standards, such as WiFi, 3G, 4G or Bluetooth. You can also see wireless communications in wireless sensor networks (WSN), and for the future are the basis of vehicular networks.

There are many challenges that researchers and practitioners try to overcome with regard to improving such communications, such as increasing bit-rates or improve access times. Another challenge is to improve energy efficiency, which is very important since the equipment used are often heavily dependent on the battery. Thus, improve upon that affects the autonomy of the devices, as well as being more environmentally friendly.

That is why this study focuses on energy efficiency in wireless communications, and for this use of multihop networks becomes. The fact that such networks are multi-hop exploits, which makes the nodes transmit at shorter distances, making their energy consumption resulting in lower transmission power. The main cuase for that result is that the power of a signal becomes lower depending on distance squared,implying that for transmission over long distances a high amount of energy is needed in transmission.

With the aim of analyzing that kin of networks, the first step established is the implementation of a simulator, in which you can be define channel models, customize nodes, and some use routing algorithms. It is also interesting to add mobility to nodes, to enrich the variety of scenarios that can be simulated. With this fact, you can do simulations of specific scenarios in which you can analyze precisely what happens with the signals transmitted in the environment. In addition, it is also interesting to make several simulations in which a particular parameter varies, trying to find how this parameter affects on the way the communications are taking place. The main goal is to prove that the use of multihop networks in urban environments improves the energy efficiency of the whole network, which means that node energy consumption is decreased.

In which the simulator implementation is concerned, Matlab is going to be the software development tool and environment, which is a widely used in the field of science and engineering. The fact is that using this platform is specifically designed to do mathematical calculations. Furthermore, the creation of graphical user interfaces is relatively easy compared to other programming languages, and there is a huge community and lots of projects builded previously, so you can take and advantage of a big amount of resources and people who can help you in case you get stuck in a certain moment of the development of the simulator.

Yo, **Pablo Ráez Sánchez** de la titulació n Ingeniería en Tecnologías de Telecomunicación de la **Escuela T´ecnica Superior de Ingenier´ıas Inform´atica y de Telecomunicaci´on de la Universidad de Granada**, con DNI 26502212W, autorizo la ubicaci´on de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Pablo Ráez Sánchez

Granada a 11 de Septiembre de 2015.

D. **Pablo Padilla de la Torre,** Profesor del área telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

### Informan:

Que el presente trabajo, titulado ***Estudio de redes colaborativas/oportunistas para redes de comunicación inalámbrica***, ha sido realizado bajo su supervisi´on por **Pablo Ráez Sánchez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de Septiembre de 2015.

### Los directores:

**Pablo Padilla de la Torre**

**Agradecimientos**

En primer lugar, quiero mostrar mi agradecimiento a mi tutor, Pablo, por todo el interés puesto para que este proyecto salga bien desde el primer momento. Agradecer su paciencia, y su buen hacer como profesor, tanto en el presente proyecto como en las distintas asignaturas en las que me ha impartido clase en la carrera. La verdad es que es una alegría ver como una persona se implica tanto en su profesión y además es capaz de crear buen ambiente entre los que le rodean. Desde luego, ha sido todo un placer compartir todo este tiempo

Siguiendo en el mundo de la docencia, dar mi agradecimiento a todos profesores que he tenido a lo largo de mi vida, ya que cada uno de ellos ha aportado una pequeña porción de sí para formar lo que soy hoy en día. Mi personalidad y forma de ver las cosas está claramente influenciada por estás personas. Por ello, gracias.

Recordar también, como no, a los amigos. Estos con los que pasas una gran cantidad de buenos momentos, y que además están dispuestos a seguir ahí. A mis compañeros de carrera, Adri, Borja, Crespo, Fran, Miguel Casares, Miguel Molina y Vicente, por la gran cantidad de momentos compartidos tanto en clase como fuera. A África, Danny (Vinny) y Miguel Rodríguez por todas las tardes y noches que hemos pasado por Granada. También recordar a los ubetenses Lasa y Pexu, con los cuales no he podido compartir mucho tiempo estos últimos años pero siempre están disponibles cuando paso por mi ciudad natal. Y por último, mi mejor amigo, al cual conozco desde los 3 años y nunca se ha separado de mí. Desde luego, mi vida no sería igual de no ser por todos. Gracias.

Finalmente, la familia. Mis padres, a los que quiero y les debo todo lo que soy. A mi tía, toda una segunda madre para mí, y mi tío, con los cuales comparto piso en Ganada. Y por supuesto a mis abuelos; mi abuela Evelia, y los que ya no están, Manolo, Paco y Loli, a los cuales nunca olvidaré.

A todos, de nuevo,

GRACIAS

Índice de contenido

[I. INTRODUCCIÓN 4](#_Toc429724575)

[II. OBJETIVOS DEL PROYECTO 6](#_Toc429724576)

[III. MARCO TEÓRICO 7](#_Toc429724577)

[A. EL CANAL DE PROPAGACIÓN 7](#_Toc429724578)

[1. PATHLOSS 9](#_Toc429724579)

[2. SHADOWING 12](#_Toc429724580)

[3. MULTIPATH 12](#_Toc429724581)

[B. EECTO DOPPLER 15](#_Toc429724582)

[C. ALGORITMOS DE ENCAMINAMIENTO GEOGRAFICO 16](#_Toc429724583)

[4. GREEDY MINIMUM ENERGY 17](#_Toc429724584)

[5. BOUNDED DISTACE FROM ABOVE ( 17](#_Toc429724585)

[6. BOUNDED DISTANCE FROM BELOW ( 18](#_Toc429724586)

[7. MODIFIED GEOGRAPHIC RANDOM FORWARDING ( 18](#_Toc429724587)

[8. MODIFIED ENERGY-EFFICIENT GEOGRAPHIC ROUTING ( 18](#_Toc429724588)

[9. DIJKSTRA 18](#_Toc429724589)

[D. EFICIENCIA ENERGETICA EN REDES COLABORATIVAS 19](#_Toc429724590)

[IV. DESARROLLO DEL SIMULADOR 21](#_Toc429724591)

[A. SELECCIÓN DE ENTORNO DE DESARROLLO. MATLAB 21](#_Toc429724592)

[B. PRIMERA APROXIMACION MEDIANTE SCRIPT 22](#_Toc429724593)

[C. IMPLEMENTACION DE LAS CLASES 23](#_Toc429724594)

[1. CANAL 25](#_Toc429724595)

[2. MAPA 28](#_Toc429724596)

[3. ENCAMINAMIENTO 30](#_Toc429724597)

[4. NODO 32](#_Toc429724598)

[5. MOVIMIENTO 34](#_Toc429724599)

[6. TRANSMISOR 36](#_Toc429724600)

[7. RECEPTOR 40](#_Toc429724601)

[8. ANTENA 42](#_Toc429724602)

[D. INTERFACES GRAFICAS 44](#_Toc429724603)

[1. CANALUI 44](#_Toc429724604)

[2. MAPAUI 48](#_Toc429724605)

[3. NODOUI 49](#_Toc429724606)

[4. FIGURARECEPCION 50](#_Toc429724607)

[5. MAINUI 55](#_Toc429724608)

[E. SCRIPTS DE SIMULACIÓN 58](#_Toc429724609)

[1. SIMULACIÓN ÚNICA 59](#_Toc429724610)

[2. VARIAS SIMULACIONES 61](#_Toc429724611)

[F. EL BUCLE DE EJECUCION 61](#_Toc429724612)

[1. SIMULACION SINGLE-THREAD 62](#_Toc429724613)

[2. SIMULACION PARALELA 63](#_Toc429724614)

[G. OTRAS FUNCIONES IMPLEMENTADAS 64](#_Toc429724615)

[V. DISEÑO DE LA EXPERIMENTACION 65](#_Toc429724616)

[A. EXPERIMENTO 1. SIMULACION SOBRE UN ENTORNO 65](#_Toc429724617)

[B. EXPERIMENTO 2. INFLUENCIA DEL NUMERO DE NODOS 65](#_Toc429724618)

[C. EXPERIMENTO 3: INFLUENCIA DEL CANAL 66](#_Toc429724619)

[D. EXPERIMENTO 4: INFLUENCIA DE LA POTENCIA ELÉCTRICA 67](#_Toc429724620)

[VI. ANALISIS DE RESULTADOS 69](#_Toc429724621)

[A. EXPERIMENTO 1. SIMULACION SOBRE UN ENTORNO 69](#_Toc429724622)

[B. EXPERIMENTO 2. INFLUENCIA DEL NUMERO DE NODOS 77](#_Toc429724623)

[C. EXPERIMENTO 3: INFLUENCIA DEL CANAL 79](#_Toc429724624)

[D. EXPERIMENTO 4: INFLUENCIA DE LA POTENCIA ELÉCTRICA 82](#_Toc429724625)

[VII. PLANIFICACION Y COSTES 84](#_Toc429724626)

[VIII. CONCLUSIONES Y LINEAS FUTURAS 86](#_Toc429724627)

[A. SIMULADOR 86](#_Toc429724628)

[B. EXPERIMENTACION 86](#_Toc429724629)

[C. LINEAS FUTURAS 87](#_Toc429724630)

[IX. REFERENCIAS 89](#_Toc429724631)

[APÉNCIDE A: TURORIAL DE USO DEL SIMULADOR 90](#_Toc429724632)

[APÉNDICE B: PROGRAMACIÓN PARALELA EN MATLAB 92](#_Toc429724633)

[APÉNDICE C: INTERFACES GRAFICAS EN MATLAB 94](#_Toc429724634)

Estudio de redes colaborativas/oportunistas para redes de comunicación inalámbrica

Pablo Ráez Sánchez - 26502212W

1. INTRODUCCIÓN

En la actualidad, las redes inalámbricas juegan un papel importante en el paradigma de las telecomunicaciones. Esto se debe, en gran parte, a la comodidad que ofrecen a los usuarios, ya que les permite moverse con libertad, sin tener que depender de un cable. Por eso, en los últimos años, las redes de este tipo se han desarrollado a un ritmo muy acelerado. Tanto es así, que un teléfono móvil de hoy día integra varios servicios basados en tecnologías inalámbricas, como pueden ser NFC, WiFi, Bluetooth y 2/3/4G.

Por otro lado, siguiendo con el ejemplo de la telefonía móvil, supone un gran reto para las operadoras soportar un gran volumen de tráfico de datos manteniendo una cierta calidad de servicio. Es por esto por lo que se han desarrollado grandes avances en las tecnologías de acceso radio, al igual que nuevas técnicas que mejoran la eficiencia espectral. A pesar de estos avances, las redes tradicionales siguen sin tener suficiente capacidad para la alta demanda de tráfico que se espera en un futuro próximo. Motivado por esto, en la comunidad investigadora se ha despertado un gran interés por las redes de celulares **multi-salto** (en adelante, MCN). Esto es debido a que estas redes aportan eficiencia energética y capacidad, además de permitir balanceo de carga. El alto número de terminales móviles activos viene como anillo al dedo para este tipo de redes, ya que esto permite que se mejoren las prestaciones de la red. Tanto es así, que el estándar **5G** hace uso de este tipo de redes, junto con otros nuevos avances tecnológicos, llegando a conseguir importantes mejorías con respecto al estándar 4G.

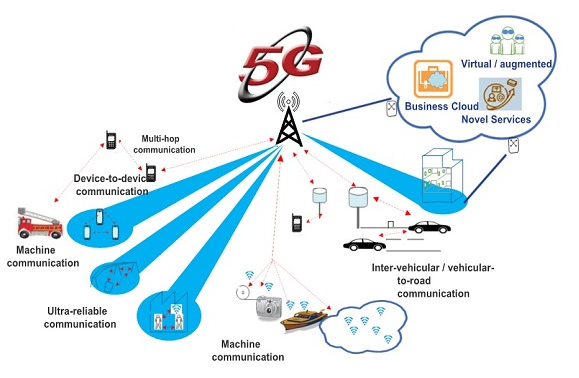


Ilustración 1: Tecnología 5G. Fuente: www.mobileeurope.co.uk

Otro ejemplo de la acogida que está teniendo la comunicación multi-salto son las redes inalámbricas de sensores (**WSN**, Wireless Sensor Networks). Estas redes están formadas por nodos autónomos distribuidos espacialmente, y cuya función es medir una serie de variables del entorno en el que se encuentran. Finalmente, se comunican entre ellos para llevar la información a un nodo final que se encarga de recopilar la información para, finalmente, distribuirla por Internet si es que así se desea. Las líneas de desarrollo de estos dispositivos se centran en mejorar su autonomía en términos energéticos, y, para ello, se aprovecha el uso de redes multi-salto que minimiza el consumo energético.

Las redes que utilizan comunicación multi-salto son conocidas como **redes colaborativas u oportunista**s, y son el principal objeto del presente estudio. Como primera aproximación al concepto de comunicación multi-salto y a modo de motivación, se propone el siguiente ejemplo:

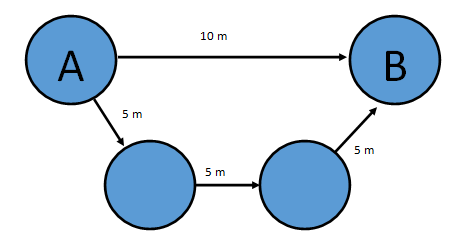


Ilustración 2: Ejemplo de comunicación multi-salto

Se sabe que el coste de una comunicación inalámbrica depende proporcionalmente de la distancia entre los nodos involucrados al cuadrado. Según la distribución que se puede observar en la ilustración 2, si se quiere realizar una comunicación entre A y B, se proponen dos caminos: uno directo, y otro utilizando los nodos que aparecen en la parte inferior. Si se toma el camino directo, se obtiene un coste de . Por otro lado, si se toma el camino inferior, de 3 saltos, se obtiene un coste de , que es menor que 100 y, por tanto, mejor. Si bien es cierto que aquí solo se ha tenido en cuenta la contribución de la distancia con el consumo de potencia para esta aproximación, pero existen muchos otros factores a tener en cuenta a la hora de estudiar este tipo de redes, como puede ser el consumo energético de cada nodo, el entorno en el que tiene lugar la comunicación, el retardo, etc.

En relación con lo anterior, una de las partes fundamentales implicadas de las tecnologías inalámbricas es el medio de propagación, que en este caso es el aire. Conocer el comportamiento del medio de transmisión es de vital importancia a la hora de diseñar los sistemas que tomarán parte en las comunicaciones, así como para elegir la topología que tendrá la red. A día de hoy, se conocen varios modelos de canal, que son aplicables dependiendo de las condiciones del entorno en el que se desarrolla la comunicación.

1. OBJETIVOS DEL PROYECTO

Teniendo en cuenta la temática presentada en la introducción, se marcan los siguientes objetivos para el presente proyecto:

1. Creación de un **simulador** que permita analizar diferentes escenarios, constituidos por un mapa en el cual aparecen nodos móviles con distintas características de transmisión. Adicionalmente se implementarán varios modelos de canal de propagación, así como distintos algoritmos de encaminamiento. Todo esto, manteniendo siempre el máximo nivel de personalización para generar escenarios de la mayor complejidad posible. La finalidad de este objetivo es aprovechar el simulador para analizar la influencia del canal en la transmisión de señales en el entorno de una red colaborativa. Para la implementación de este simulador se utilizará la plataforma Matlab, por razones que se comentarán posteriormente.
2. Estudio del funcionamiento de las **redes colaborativas** y posterior **comparación** con el **modelo clásico** ‘Punto de acceso - Terminal’. Concretamente, se hará uso del simulador implementado para obtener datos de interés cuanto a uso de energía, calidad de la señal, eficiencia de encaminamiento, etc.

Cabe mencionar que el presente proyecto se centra solamente en la **eficiencia energética** de las redes colaborativas. Existen muchos otros parámetros, como retardo de comunicación, bit-rate, balanceo de carga, etc, que son muy importantes a la hora de estudiar en profundidad las redes colaborativas. Estos parámetros no son objeto de estudio en el presente proyecto debido a la gran carga de trabajo que supondría incluirlos garantizando un resultado competitivo. Si bien, es cierto que el presente proyecto sirve como base para, más adelante, incluir estos parámetros.

1. MARCO TEÓRICO

En este apartado se comentan ciertos conceptos teóricos a tener en cuenta de cara a la realización del presente proyecto. Todo lo que aparece aquí más adelante tiene una repercusión directa en la implementación del simulador.

1. EL CANAL DE PROPAGACIÓN

Ver [1] [2] [3]

Un modelo de propagación es un conjunto de expresiones matemáticas, diagramas y algoritmos que se utilizan para representar las características de radio de un ambiente dado.

Hay que tener en cuenta que los modelos de propagación modelan una gran variedad de fenómenos físicos que se dan de forma simultánea. Por ejemplo, en una comunicación entre una estación base y un terminal móvil, se podrían producir los siguientes fenómenos fisicos:

* Reflexión: Se produce cuando las ondas se reflejan en los edificios que nos rodean.
* Refracción: Cuando las ondas atraviesan obstáculos, haciendo que su trayectoria cambie.
* Difracción: Se produce cuando las ondas esquivan los obstáculos que se encuentran. Por ejemplo, esquinas
* Efecto Doppler: En entornos de movilidad, la frecuencia con la que se perciben las ondas varía dependiendo de nuestro movimiento con respecto a estas.

La suma de todos estos fenómenos hace que la señal que se percibe en el receptor sea distinta a la que se origina en el transmisor, de tal forma que la comunicación se ve alterada. A todos estos efectos hay que añadirle la existencia de ruido e interferencias, que hacen que las comunicaciones se vean afectadas negativamente. Por otro lado, hay que tener en cuenta que el comportamiento del canal de propagación no es constante, es decir, varía con el tiempo, lo cual es una dificultad añadida a la hora de estudiarlo.

En la implementación de sistemas de comunicaciones inalámbricas, es muy importante conocer el comportamiento del canal de propagación, aunque sea de forma aproximada, ya que de este depende la cobertura entre nodos, o el retardo y el bit-rate que se pueden alcanzar en las comunicaciones. Por ello, es importante modelar este canal, de forma que se pueda hacer una estimación del efecto que este tiene en una comunicación punto a punto.

Centrándonos en el ámbito de la **potencia** de las señales que pasan por el canal, existen varios efectos a tener en cuenta, y se dividen según la escala en la que afectan con respecto a la distancia y la longitud de onda. Así, podemos encontrar efectos a gran escala, es decir, que se pueden apreciar cuando la distancia de entre el transmisor y el receptor es mayor que la que la longitud de onda. Por otro lado, se encuentran los efectos de pequeña escala, que se parecían en cambios de posición de unas proporciones aproximadas a la longitud de onda. A continuación, se presentan los efectos

* Efectos a gran escala
  + ***Pathloss*:** Perdidas de potencia debidas a la distancia entre emisor y receptor
  + ***Shadowing*.** Ensombrecimiento. Variaciones en la atenuación debidas a los obstáculos.
* Efectos de pequeña escala
  + ***Multipath*.** Multi-trayectoria. Variaciones en la atenuación debidas a la suma (constructiva o destructiva)de las distintas señales que se reciben, debido a que estas pueden tomar varios caminos.

En la siguiente imagen, se muestra un resumen de la contribución de los distintos efectos presentados en el modelo final del canal:

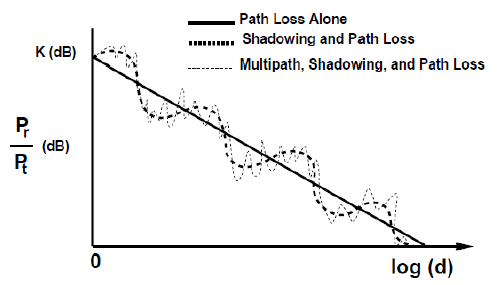


Ilustración 3: Modelo genérico de canal. Fuente: [3]

Una de las consecuencias apreciables de estos efectos es la forma del área de cobertura en un plano, cuya representación se puede observar en la siguiente imagen:

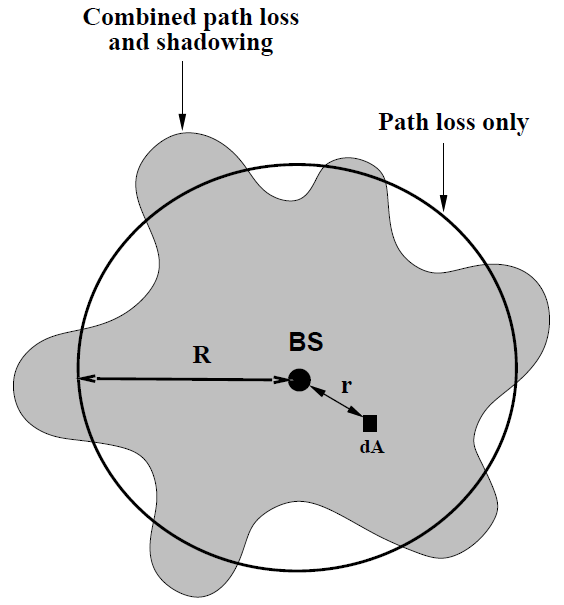


Ilustración 4: Vista de la cobertura sobre el plano: Fuente [3]

Una vez presentados estos efectos, se procede a estudiarlos individualmente en profundidad.

1. PATHLOSS

El *pathloss* son las pérdidas de potencia que se producen debido a la distancia que existe entre emisor y receptor. Además, este efecto es invariante en el tiempo. Dada la gran variedad de modelos existentes que parametrizan este efecto, es conveniente agruparlos. Según su origen, existen los siguientes tipos:

* **Empíricos**: También conocidos como estadísticos. Este tipo de modelos son construidos a partir de medidas en un entorno. A la hora de aplicar este tipo de modelos, es importante que las características del entorno que estamos analizando sean lo más parecidas a las del entorno original. Típicamente, en estos modelos se conocen unos márgenes para los parámetros que intervienen en sus expresiones. Dentro de estos márgenes, los resultados obtenidos tendrían validez. En cuanto a la carga computacional de estos modelos, no suele ser excesivamente alta. Entre estos modelos, se encuentran
* **Teóricos**: Estos modelos se basan en la aplicación de distintos principios físicos. Por lo tanto, pueden ser aplicados en escenarios distintos, siendo los resultados válidos. Por otro lado, el coste computacional de los cálculos requeridos al usar estos modelos es mayor en comparación con los modelos empíricos. Debido a esto, este tipo de modelos son más utilizados en áreas pequeñas o interiores.
* **Semi**-**empíricos**: Están basados en la combinación de los anteriores tipos de modelos.

Por otro lado, los modelos de propagación también se clasifican según el entorno en el que son aplicables. En este caso, se podría distinguir entre modelos de interior, de exterior, urbanos, suburbanos, etc.

En el presente proyecto, concretamente, se utilizan modelos teóricos. Si bien es cierto que cabe la posibilidad de implementar modelos de otra índole de cara al futuro. A continuación, se presentan algunos de estos modelos:

* **PROPAGACIÓN EN EL ESPACIO LIBRE**

Este conocido modelo teórico se basa en las ecuaciones de Maxwell. Es válido para una señal que se propaga en línea recta entre el emisor y el receptor sin que haya ningún obstáculo entre ambos. Este estado es conocido como LOS (*Line Of Sight*). Dadas estas circunstancias, no se produce ningún tipo de atenuación ni reflexión generada por obstáculos.

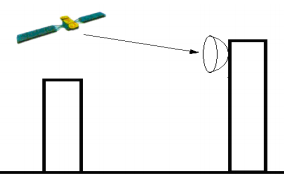


Ilustración 5: Visión directa. Fuente: [4]

Las pérdidas de propagación, o *Pathloss (PL)*, se calculan como la relación entre la potencia recibida y la potencia transmitida. En este modelo, concretamente, se calcula de la siguiente manera:

( 1 )

Siendo el producto de los patrones de radiación de las antenas transmisora y receptora.

En ingeniería, también es útil el cálculo de estos parámetros en decibelios. En ese caso, la expresión quedaría de la siguiente forma:

( 2 )

* **MODELO DE 2 RAYOS**

El modelo de 2 rayos, también teórico, es ideal en un escenario donde el único obstáculo es el suelo. Este modelo incluye el efecto de un rayo directo (**LOS**), y otro que se refleja en un obstáculo, por lo que llega al receptor con un retardo, además de ver cambiada su amplitud y fase. Este modelo es ideal para un escenario en el que transmisor y receptor se encuentran a la vista y además nos encontramos cerca de la tierra, donde se refleja otro rayo. A continuación, se puede ver una imagen que ilustra el escenario que representa este modelo:

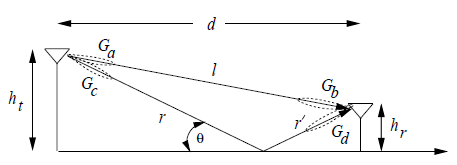


Ilustración 6: Modelo de 2 rayos. Fuente: [3]

Como se puede observar, algunos de los parámetros que tiene en cuenta este modelo son las alturas de las antenas, y las distintas ganancias, que dependen de la dirección por la que se recibe el rayo. En este caso, el *pathloss* se calcula según la siguiente ecuación:

( 3 )

Donde es la ganancia de la antena transmisora, la ganancia de la antena receptora, es la diferencia de fase entre las 2 señales y es el coeficiente de reflexión de la superficie reflectante.

Aplicando trigonometría, y algunas aproximaciones, se puede llegar a la siguiente expresión:

( 4 )

O, expresado en dB

( 5 )

Hay que tener en cuenta que, dadas las aproximaciones realizadas para llegar a esta solución, debe de cumplirse que

* **MODELO SIMPLIFICADO**

Debido a la alta complejidad de la propagación de señales, se torna difícil encontrar un modelo único que caracterice el *pathloss* de una forma precisa para diferentes entornos. Aun así, para un estudio más práctico, como el análisis de varios tipos de sistemas, no nos es necesario disponer de un modelo de gran precisión; basta con tener uno en el cual resida la esencia de la propagación de la señal. Esta es la razón por la cual se crea el modelo simplificado. En este modelo, el *pathloss* se calcula así:

( 6 )

Donde es una constante que depende de las características de las antenas y la atenuación, es una distancia de referencia para el campo lejano, y es el exponente del *pathloss.*

Este modelo es válido para para transmisiones en las que . En cuanto a , este toma un valor entre 1 y 10m para escenarios de interior, mientras que para escenarios de exterior toma valores en el rango 10-100m. suele tomar el *pathloss* evaluado a la distancia de referencia . Por último, , depende fuertemente del entorno. Suele tomar valores entre 2 y 4. Normalmente, este se obtiene realizando medidas empíricas y calculando que valor de produce el mínimo error cuadrático medio.

1. SHADOWING

El *shadowing* o **ensombrecimiento** es un efecto que se debe principalmente a los obstáculos que se encuentran entre transmisor y receptor. Las variaciones se aprecian en distancias del orden de los obstáculos, con lo que, para modelarlo, estas variaciones se toman con respecto al logaritmo en base 10 de la distancia como se puede apreciar en la ilustración 3. Debido a esto, se dice que el *shadowing* tiene una variación lenta.

* **MODELO LOG-NORMAL**

Este modelo es el que se utiliza comúnmente para modelar el shadowing. Es un modelo que sigue la distribución estadística log.normal, cuya función densidad de probabilidad (*PDF* en adelante) es la siguiente:

C:\Users\pablo\AppData\Local\Temp\x10sctmp0.png

( 7 )

Siendo el logaritmo de la media y el logaritmo de la desviación típica.

Si se quiere obtener los resultados directamente en decibelios, se puede sustituir la PDF log-normal por una gaussiana.

1. MULTIPATH

Este efecto es el que modela las variaciones en la potencia de recepción debidas a que en el receptor se observan varias señales procedentes del emisor, pero que llegan al receptor por **distintos** **caminos**. De esta forma, la suma de las distintas señales puede ser constructiva o destructiva, dándose el caso de que la señal recibida sea más fuerte o más débil. Además de contribuir en variaciones de potencia, esto también hace que las señales se deformen, produciendo, entre otras cosas, interferencia inter-simbólica en el caso de comunicaciones digitales.

* **MODELO DE RAILEIGH**

El modelo de Raileigh se utiliza comúnmente para modelar comunicaciones con dispositivos móviles. En este caso, el escenario que se estudia es aquel en el que la línea de visión directa entre emisor y receptor se ve obstaculizada, o bien, la potencia del rayo directo es inferior al nivel de ruido. La siguiente figura ilustra el escenario:

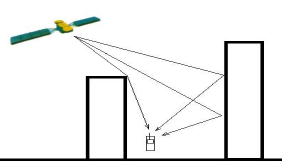


Ilustración 7: Sin visión directa. NLOS. Fuente: [4]

Suponiendo que la señal original mandada fuera de la forma

( 8 )

Debido a la propagación multi-trayecto en el receptor obtendríamos

( 9 )

Siendo las amplitudes y las fases valores aleatorios. Esta sumatoria puede reescribirse como

( 10 )

Donde y son variables aleatorias gaussianas. A partir de esta última expresión se puede obtener en forma de modulo y fase

( 11 )

con

Analizando estos resultados, se puede concluir que la fase sigue una distribución uniforme entre 0 y . En cuanto al módulo, este sigue la conocida distribución Raileigh, cuya *PDF* es:

( 12 )

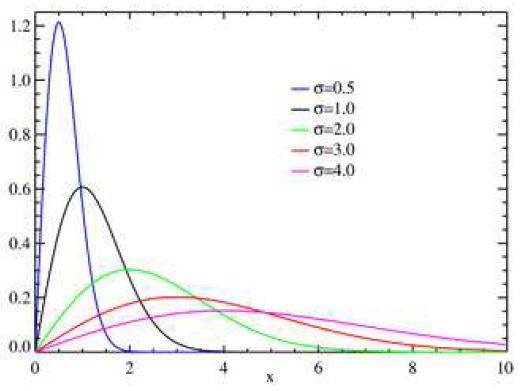


Ilustración 8: Distribución de Rayleigh

* **MODELO DE RICIAN**

El esquema que sigue el modelo de canal de Rician es similar al de Raileigh, pero, en este caso, además, hay línea de visión directa entre emisor y receptor.

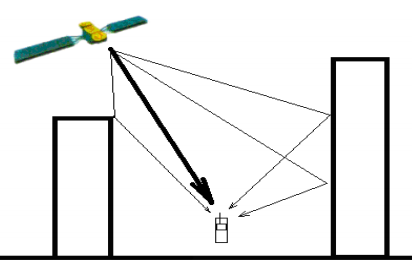
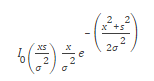


Ilustración 9: LOS con mult-itrayectoria: Fuente: [4]

En este caso, la *PDF* de un modelo de Rician viene descrita por la siguiente expresión



( 13 )

Siendo el parámetro de no centralidad y la desviación típica. Para se obtiene una distribución Rayleigh. La idea es utilizar el valor de para generar una *PDF* que devuelva valores menos dispersos de x con respecto a uno, de tal forma que lo que se obtiene en la práctica es un modelo de canal menos agresivo que el de Rayleigh

* **MODELO DE WEIBULL**

Este es otro modelo estadístico basado en Rayleigh, que, en este caso, utiliza la distribución de Weibull. Esta es su *PDF:*

C:\Users\pablo\AppData\Local\Temp\x10sctmp2.png

( 14 )

Siendo el parámetro de escala y el parámetro de forma.

Concretamente, cuando el parámetro de forma es igual a 2, se obtiene una distribución de Rayleigh. Jugando con estos dos parámetros, se pueden conseguir distintas PDFs que se ajusten mejor al canal que se está utilizando.

1. EECTO DOPPLER

Ver [5] [3]

El efecto Doppler es un fenómeno físico por el cual un receptor percibe una variación en la frecuencia de la señal que emite un transmisor. Esta variación puede provocar que la frecuencia de recepción sea mayor o menor, dependiendo de si emisor y receptor se alejan o se acercan. Dicho de otra forma, el la variación en frecuencia depende de la velocidad relativa entre emisor y receptor. Existen muchas formas de describir este efecto. A continuación, se muestra su fórmula descrita vectorialmente

( 15 )

Siendo la frecuencia captada por el receptor, la frecuencia emitida por el transmisor, la velocidad de propagación de la onda, la norma de la velocidad del observador, la norma de la velocidad del emisor y el vector unitario cuyo origen es el emisor y detino el receptor. A continuación se muestra una imagen a modo de aclaración:

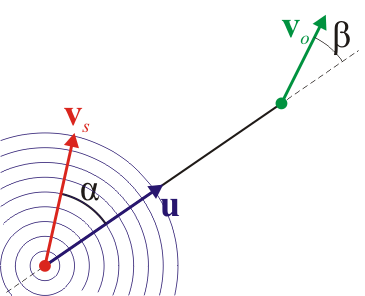


Ilustración 10: Efecto Doppler. Fuente: [5]

Este fenómeno es muy importante en las comunicaciones inalámbricas ya que estos cambios en frecuencia pueden desembocar en pérdidas de potencia o interferencia inter-simbólica en comunicaciones digitales.

En el caso de una comunicación directa, es decir, sin reflexiones, refracciones, dispersión, etc, el observador vería como el espectro de frecuencia de la señal se desplaza en linealmente, pero la señal mantiene su forma.

Por otro lado, en el caso de un entorno con multi-trayectoria, la situación se complica, ya que cada punto de *scattering* actúa como observador con respecto al emisor y como emisor con respecto al receptor. Esto hace que el receptor observe una señal deformada. Un caso bien estudiado de este suceso es el conocido como *Classical Doppler Spectrum* o *Jakes Spectrum*. A continuación, se muestra la representación de dicho espectro:

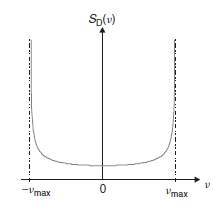


Ilustración 11:Espectro de Jakes. Fuente: [3]

Donde es la variación con respecto a la frecuencia inicial, y la variación máxima(es decir, cuando emisor y receptor se acercan/alejan a máxima velocidad).

Finalmente, se analiza la repercusión de este efecto sobre el presente proyecto. Para ello, hay que centrarse en el ámbito de la potencia. Suponiendo un entorno en el que los nodos se mueven a una velocidad de 100m/s y transmiten a una frecuencia de 800MHz, la máxima variación en frecuencia sería , lo cual es un valor muy pequeño como para tener repercusión en la potencia de recepción final, ya que (6 órdenes de magnitud).

Por esta razón **no se implementa** el efecto Doppler en el presente **simulador**. Si bien es cierto que cabe la posibilidad de simular escenarios con frecuencias de portadora bajas o velocidades muy altas, por lo que, en estos escenarios, la potencia de recepción sí que se ve afectada por el efecto Doppler. Por otro lado, estos escenarios quedan fuera del principal objetivo del proyecto, ya que no son comunes para la aplicación que se busca satisfacer con el simulador.

1. ALGORITMOS DE ENCAMINAMIENTO GEOGRAFICO

Ver [6] [7] [8]

En la introducción, se mencionaba un sencillo ejemplo del porque la comunicación multi-salto es interesante. Pues bien, situándonos en una perspectiva de un nodo (*Source*, fuente) perteneciente a una red colaborativa, y que quiere mandar información a otro nodo, (*Destination*, destino), este nodo S debe ser capaz de seleccionar a otro nodo vecino como siguiente salto a la hora de mandar información hacia . De forma intuitiva, este siguiente nodo debe de estar a una distancia de menor que la distancia entre y . Pero, en una red conformada por varios nodos… ¿Qué criterio debe tomarse a la hora de elegir a este siguiente nodo? ¿Existe alguna forma de establecer una ruta que sea lo más eficiente posible?

Pues bien, existen algoritmos que pueden ser utilizados para confeccionar una ruta entre y . Los algoritmos que se presentan a continuación han sido obtenidos de distintos artículos científicos. Concretamente, son algoritmos geográficos, ya que, para decidir cuál es el siguiente nodo, se tiene en cuenta la posición de los nodos en el plano. A continuación, se muestra una imagen en la que se describen distintos conceptos que van a ser utilizados en las subsiguientes secciones:



Ilustración 12: Escenario de red colaborativa. Fuente: [6]

Como se puede observar, aparecen y , acompañados por más nodos. Existe una zona sombreada llamada FTS (*Forward Transmision Sector*), que es la zona en la que se encuentran los nodos susceptibles de ser seleccionados como siguiente nodo, ya que se encuentran entre y . También aparece una circunferencia de radio , centrada en . Este radio es la distancia óptima que debería haber entre 2 nodos para que la potencia consumida en la comunicación sea mínima. Para saber más sobre este parámetro, hay que remitirse a la sección D más adelante. Esta circunferencia delimita 2 zonas: e .

A continuación, se detallan los algoritmos que se utilizan en presente proyecto.

1. GREEDY MINIMUM ENERGY

Se selecciona como *relay* (siguiente nodo) aquel que se encuentre en FTS y sea más cercano a .

*Nota: mencionar que, tras realizar un salto, el nodo seleccionado toma el papel de , y se calcula una nueva FTS,, etc, con las que encontrar un siguiente salto de forma recursiva hasta que se llega al nodo de destino . Esto se cumple en todos los algoritmos.*

1. BOUNDED DISTACE FROM ABOVE (

Si se encuentra en , se transmite directamente hacia el. De lo contrario, se toman los nodos del área FTS , y se selecciona a aquel más cercano a .

1. BOUNDED DISTANCE FROM BELOW (

Bastante similar al anterior. Si se encuentra en , se transmite directamente hacia el. De lo contrario, se toman los nodos del área FTS , y se selecciona a aquel más lejano a . En caso de no existir este nodo, se utiliza el algoritmo **.**

1. MODIFIED GEOGRAPHIC RANDOM FORWARDING (

Se toman los nodos que se encuentran en el anillo con centro en y que cubre el radio . De todos estos, se toma aquel que esté más cercano a . En caso de no existir ninguno, se utiliza el algoritmo .

1. MODIFIED ENERGY-EFFICIENT GEOGRAPHIC ROUTING (

Para comprender mejor este algoritmo, se presenta la siguiente figura:



Ilustración 13: Escenario EEGR. Fuente: [6]

En este algoritmo, se genera un círculo cuyo centro se encuentra en la intersección de la recta que une y y la circunferencia de radio . El radio de dicho círculo es . Si se encuentra en el círculo, se transmite directamente hacia el. De lo contrario, se transmite al nodo más cercano a presente en el círculo.

1. DIJKSTRA

Este algoritmo no es como tal de encaminamiento geográfico, si bien es un algoritmo de caminos mínimos. Este algoritmo requiere conocer la topología completa de la red y, a partir de esta, obtiene el camino más eficiente entre 2 nodos.

Por el hecho de que requiere el conocimiento de la topología completa de la red, no puede ser implementado en los nodos de una red colaborativa. Si bien es cierto que este algoritmo puede ser utilizado en el simulador para comparar el rendimiento entre los anteriores algoritmos mencionados. Aquel que se aproxime más al resultado obtenido por Dijkstra, será el más eficiente en el escenario estudiado.

1. EFICIENCIA ENERGETICA EN REDES COLABORATIVAS

Ver [6]

El consumo energético de un nodo en una red colaborativa viene dado por la potencia que consume para transmitir información más la potencia que consume por el mero hecho de encontrarse encendido.

Así pues se define el *advance ratio (s)*  de un salto como la relación entre la distancia y el consumo en un salto en el que se involucran dos nodos i y j:

( 16 )

Donde es la distancia entre los nodos, es la potencia con la que transmite un nodo en el salto y representa la potencia que consumen los nodos involucrados en el salto por estar encendidos. En este caso, se asume que ambos nodos consumen lo mismo.

Analizando esta función, se puede observar que en distancias cortas, s es muy pequeño, ya que es pequeño y es despreciable con respecto a . Conforme aumenta la distancia entre los nodos, s aumenta. Finalmente, se llega a un punto en el que s comienza a disminuir si la distancia es muy grande, debido a que el termino crece muy rápido.

Por tanto, existe una distancia para la cual se encuentra un máximo en el *advance ratio.* Para encontrar el valor de , basta con derivar y encontrar el punto en el que esta derivada se anula, o, lo que es lo mismo:

( 17 )

A continuación se desarrolla en *s* la potencia de transmisión. Para ello, se asumen unas pérdidas de propagación de modelo simplificado, sin tener en cuenta otros efectos como *shadowing* o *multipath*. Además, se supone que se transmite la potencia mínima como para satisfacer la sensibilidad del receptor. De esta forma, se obtiene la siguiente ecuación:

( 18 )

Una vez desarrollada *s*, se procede a derivaron respecto a . Cuando se iguala a cero, pasa a ser .

( 19 )

Este término,, es la distancia a la que deben encontrarse los nodos de una red colaborativa de modo que el ratio de avance sea máximo , o, lo que es lo mismo se cubre una mayor distancia sin sacrificar en consumo energético.

* EFECTO DE LA DISTANCIA ÓPTIMA EN LA CREACIÓN DE RUTAS

Algunos de los algoritmos de encaminamiento mencionados en el **apartado C** necesitan conocer la distancia óptima para establecer las rutas. Aquí se hace un pequeño resumen de cómo afecta este parámetro a las rutas finales.

En primer lugar, mencionar que es directamente proporcional a la potencia eléctrica de los nodos (), lo cual implica que, a mayor , mayor . Como consecuencia, si imaginamos un escenario estático, y variamos el consumo eléctrico de los nodos para hacerlo mayor, se obtiene como resultado que el número de saltos totales de la ruta disminuye. Tanto es así que se puede llegar a un punto en el que lo óptimo sea hacer una transmisión directa entre el nodo de origen y el de destino, ya que, de utilizar nodos de *relay* con alto consumo eléctrico hace que el coste de la ruta aumente demasiado.

A tenor de esto, mencionar que las anteriores formulas presentadas con teóricas. En ellas, se asume que la potencia de transmisión puede aumentar indefinidamente con la distancia, y, a partir de esa hipótesis, se obtienen los demás resultados. Pero, en la práctica, la potencia de transmisión de los nodos está limitada, y esto provoca que la distancia de cobertura sea limitada. De este modo, se puede llegar al siguiente caso:

( 20 )

De modo que, al ser la distancia entre los nodos menor que la distancia óptima, se intente hacer una transmisión directa. El problema, en este caso, es que esa transmisión nunca tendría existo ya que los nodos no están dentro de la zona de cobertura.

En este caso, utilizar este tipo de encaminamiento no tiene sentido, ya que no daría resultados válidos. En lugar de utilizar un encaminamiento en el que prime el consumo de óptimo, sería más conveniente usar algoritmos de encaminamiento en los que se llegue al destino. O bien, hacer ajustes sobre los algoritmos de los que disponemos para que encuentren una ruta que llegue al objetivo siendo lo más óptima posible, aun teniendo en cuenta que dar un salto directo sería lo más eficiente en caso de poder hacerse.

1. DESARROLLO DEL SIMULADOR

A continuación, se detallan las fases de desarrollo del primer objetivo: el simulador.

1. SELECCIÓN DE ENTORNO DE DESARROLLO. MATLAB

Para la realización del simulador, se elige Matlab como entorno de desarrollo de la aplicación frente a otras alternativas, como podrían ser Java, Python, C, etc. El principal motivo es que Matlab es un entorno que está diseñado para trabajar con datos numéricos, e implementa gran cantidad de métodos útiles a la hora de trabajar con estos.

Matlab es un software desarrollado por la empresa Mathworks que dispone de un lenguaje de alto nivel (m) y un entorno de desarrollo integrado, que, como se menciona anteriormente, está orientado totalmente al trabajo con números. Una de sus principales características, de la cual deriva su nombre, es el cálculo matricial (MATrix LABoratory). Es un software muy usado en el mundo de la ingeniería, como puede observarse por la inmensa cantidad de *toolboxes* (nombre de sus *plugins*) que existen para mejorar la experiencia del usuario y satisfacer las necesidades de distintos tipos de profesionales, ya que es usado en campos como telecomunicaciones, aeronáutica, finanzas, biología computacional, etc.

Uno de los puntos fuertes de Matlab, debido a lo mencionado anteriormente, es la gran comunidad que existe alrededor del mundo. Prácticamente cualquier duda que surja a día de hoy, ya ha sido publicada por Internet por otro usuario y resuelta. Además, Mathworks dispone de una plataforma llamada *FileExchanger*, en la cual los usuarios pueden compartir su código, con el fin de distribuir su trabajo y ayudar al resto de la comunidad. Todo esto se complementa con una documentación propia del software con un gran nivel de detalle y ejemplos, lo cual hace que, en conjunto con la gran comunidad, sea fácil encontrar una solución en un momento de bloqueo.

Como se ha mencionado anteriormente, Matlab está diseñado para cálculo matricial. Pero, además, también dispone de orientación a objetos, lo cual es una característica interesante para el presente proyecto, ya que así se pueden implementar los distintos elementos que conforman los escenarios, y sus correspondientes métodos.

En cuanto a la forma de ver los resultados, Matlab dispone de diversas funciones para trabajar con números y visualizarlos, tanto en 2D como en 3D. Además, se permite la creación de interfaces gráficas de usuario (en inglés *Graphical User Interfaces*, oGUI) de una manera relativamente intuitiva, lo cual es útil en este proyecto para hacer que los resultados se muestren ordenados. Con ello, también se permite la creación de un escenario concreto sin tener que escribir ninguna línea de código, lo cual da facilidades al usuario para trabajar con el *software*.

Pero no todo son ventajas a la hora de usar Matlab. La primera desventaja que nos encontramos con respecto a las otras posibles opciones es que Matlab es un software de pago, lo cual hace que cualquier persona que quiera utilizar el presente proyecto requiera de una licencia de uso del programa., siendo estas de un coste elevado.

Por otro lado, Matlab es un lenguaje de tipo ‘*single thread’*. Esto quiere decir que el código que se genera solo se ejecuta en un núcleo del procesador, y, a priori, no se pueden ejecutar más hebras de forma simultánea. Y se dice a priori, porque realmente existe un *toolbox* de procesamiento paralelo, que sí que permite hacer uso de más núcleos. Para ello, lo que hace este *toolbox* es ejecutar otras instancias independientes de Matlab, o *workers*, pasarle el código correspondiente y, una vez ejecutado este, pasar los resultados a la instancia principal de Matlab. Esta forma de emular la programación *multicore* no es la más óptima, con lo cual se consumen muchos recursos en comparación con otros lenguajes. Además, para hacer uso del *toolbox*, hay que programar de una forma específica para que Matlab pueda identificar de forma correcta los recursos que necesita y enviarlos a los *workers.* Más adelante, se explicará más en profundidad la forma en la que hay que programar para usar este *toolbox*.

La última de las desventajas reseñables es la forma en la que Matlab llama a las funciones, y es que todos los pasos de variables se hacen por valor. Esto significa que, si se dispone de una matriz, y se pasa como parámetro a una función, la matriz es copiada íntegramente al *scope* de la función. Esto hace que se consuma más memoria y se invierta un tiempo no desdeñable a la hora de llamar a una función. Con lo cual, es preferible que los argumentos que se pasan a las funciones sean del menor tamaño posible, e incluso, si es posible, ahorrarse el uso de funciones. Si se hace esto, se obtiene un rendimiento mayor, pero por el contrario se obtiene un código menos legible.

En resumen, Matlab es un lenguaje más que adecuado por su facilidad de uso y su gran acogida en la comunidad científica, si bien es cierto que hay que tener en cuenta sus posibles deficiencias a la hora de crear el código para minimizarlas y generar un software que tenga un rendimiento competente.

Ver [9] [10]

1. PRIMERA APROXIMACION MEDIANTE SCRIPT

El primer paso a dar en el desarrollo del simulador es la creación de un script en Matlab, en el que se simula un escenario mínimo. Concretamente, se simula un mapa de dimensiones variables, en el cual hay dos nodos; uno, receptor, situado en el centro del mapa y otro, emisor, que se mueve desde la esquina inferior izquierda hacia la superior derecha. El objetivo a cubrir en esta fase es poder observar la potencia de recepción en el nodo receptor, dada una potencia de transmisión fija en el transmisor. Por otro lado, se genera una aproximación al modelo de pérdidas de canal de propagación, que implementa *pathloss* en el espacio libre y efecto de *multipath* de tipo Rayleigh.

Al inicio de este script se definen los parámetros de la simulación, como:

* Tiempo total de simulación
* Tiempo entre cálculos (dt)
* Dimensiones del mapa
* Parámetros de transmisión y recepción de los nodos, como potencias, ganancias, etc.
* Parámetros de canal

Tras la definición de los parámetros de la simulación, se realiza una reserva de recursos (preallocation). Por ejemplo, si el vector de tiempos tiene una longitud n, se crea un vector donde se guardarán los datos de recepción de esa misma longitud.

A pesar de que Matlab permite el uso de vector de tamaño dinámico, su uso es ineficiente, ya que para añadir un elemento que no cabe en el vector, Matlab crea un vector nuevo con un tamaño suficiente como para que entre el dato, y copia el viejo vector dentro de este nuevo. Por tanto, es preferible realizar una reserva de recursos previa, para evitar mal rendimiento.

Tras esto, dentro de un bucle *for*, se van calculando la recepción en el instante determinado y el cambio de posición del nodo emisor debido a su movimiento. Tras terminar el bucle, se muestran los resultados en una gráfica, donde se puede verificar que todo está correctamente implementado; es decir, que conforme el nodo emisor se acerca al receptor, la potencia de recepción aumenta. Además, se observan variaciones debido al efecto multipath.

Adicionalmente, también se hace una prueba de dibujo 3D. Para ello, al inicio se genera una matriz que representa al mapa, y donde cada elemento representa la distancia con respecto al nodo receptor. A partir de esta matriz, se genera otra en la cual se almacena el valor de las pérdidas de canal. Finalmente, esta matriz se representa tras cada iteración, con lo cual, se puede observar la evolución de las pérdidas de propagación observadas desde el nodo central en 3 dimensiones y en tiempo de simulación.

Este primer script da una visión inicial del objetivo que se quiere cubrir, y una idea de las variables que hacen falta para realizar las simulaciones. Esta es la base del que será el bucle principal de ejecución de la aplicación. El siguiente paso es la creación de las diferentes clases que representan los elementos que forman parte del escenario de simulación. Tras la creación de dichas clases, se procede a la creación de diferentes interfaces gráficas; unas para facilitar la interacción con otras clases y otras para hacer la simulación en sí.

1. IMPLEMENTACION DE LAS CLASES

En este apartado, se da explicación a las distintas clases implementadas y cómo interactúan entre ellas.. Algunas de estas clases, tienen asociada una interfaz gráfica que ayuda a la creación de objetos que hace el simulador más amigable para el usuario. Estas interfaces gráficas se detallan en el apartado de interfaces gráficas. En primer lugar se muestra un diagrama UML de clases que ayuda interpretar de una manera más sencilla el trabajo de este apartado. Tras esto, se comienza a analizar cada clase individualmente. Mencionar que, adicionalmente, todas las funciones se encuentran documentadas, pudiendo accederse sus descripciones completas dentro de Matlab.

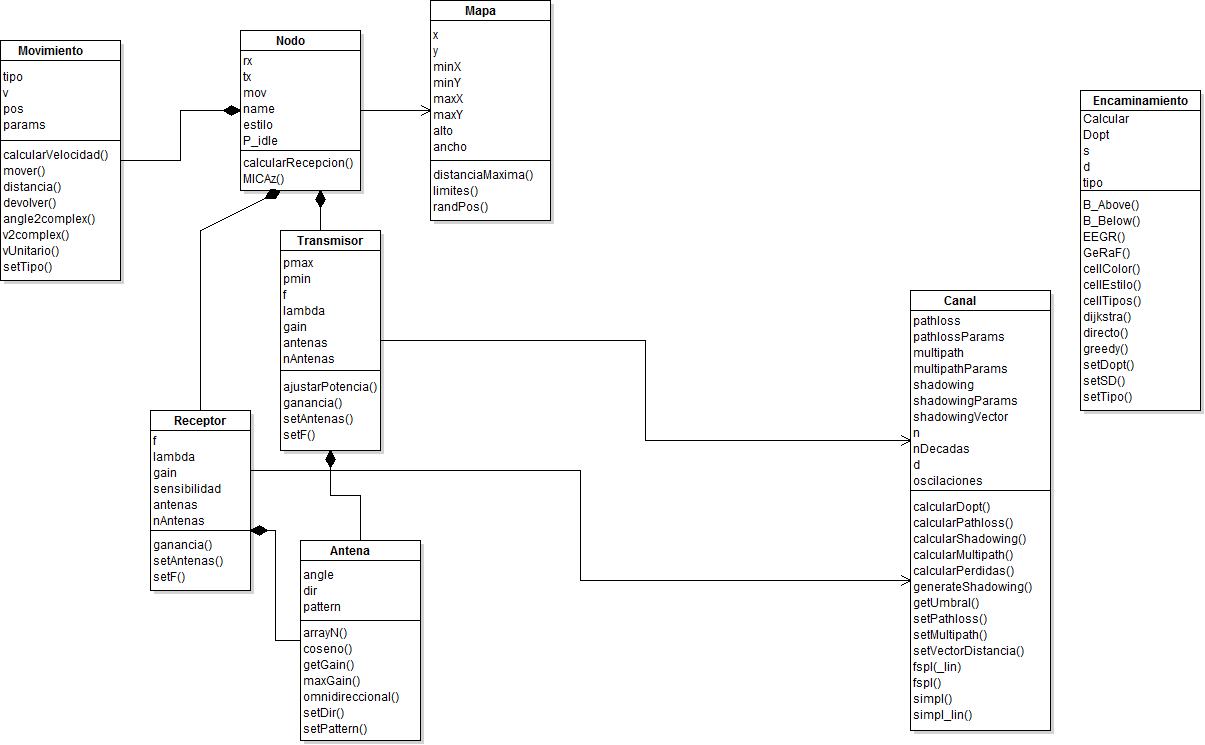
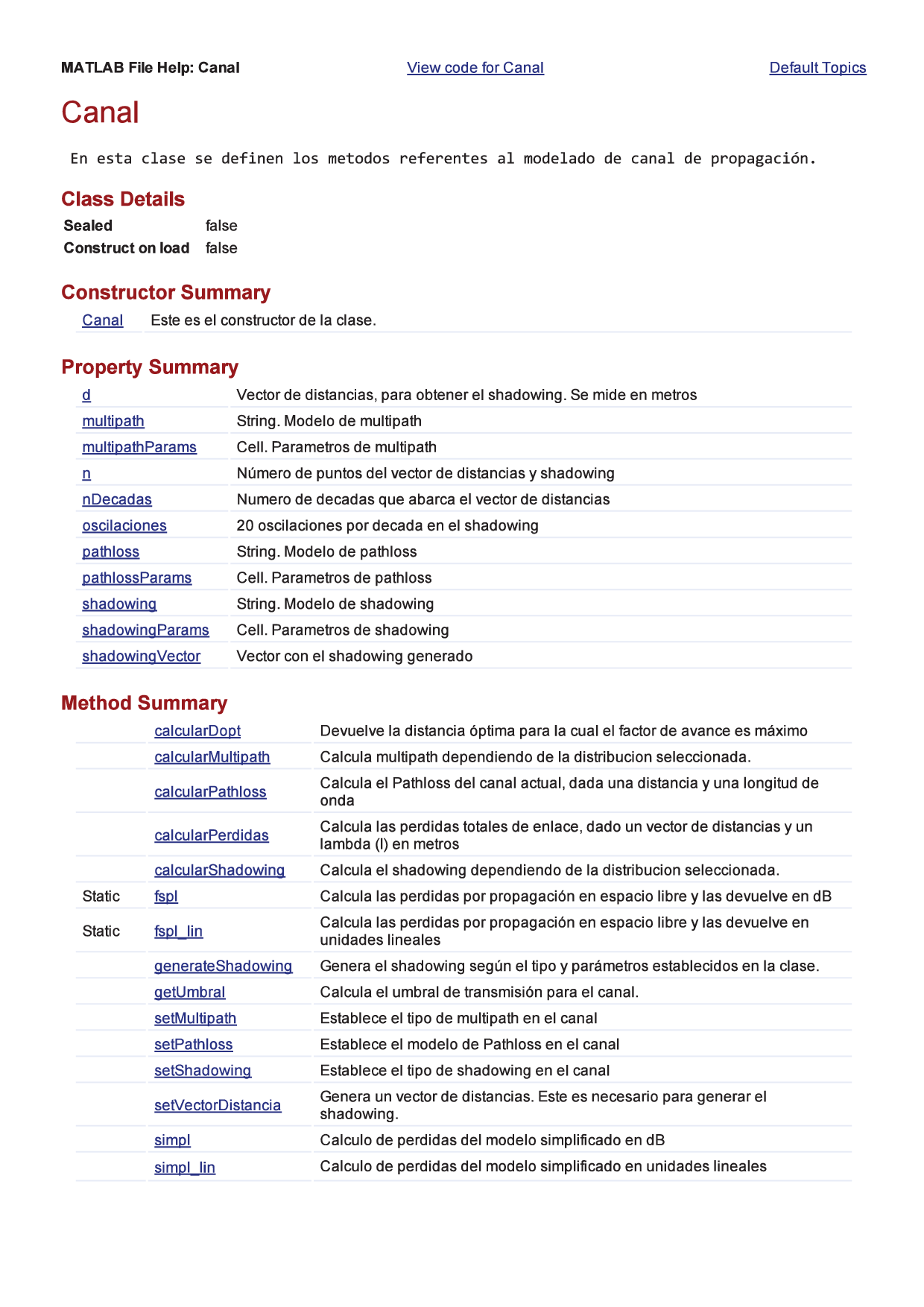


Ilustración 14: Diagrama UML de las clases que conforman el proyecto

1. CANAL

Ilustración 15: Resumen de la clase Canal



Esta clase es la encargada de representar al canal de propagación y sus correspondientes efectos. La principal finalidad de esta clase es que, una vez definida de forma correcta, devuelva el valor de las pérdidas de propagación de canal en un instante determinado, dados los parámetros de entrada necesarios. Se implementa lo siguiente

*Pathloss*:

* Perdidas por propagación en espacio libre
* Modelo simplificado

*Shadowing*

* Distribución lognormal

*Multipath*

* Rayleigh
* Rician
* Weibull

El único parámetro obligatorio es el *pathloss*. Los otros dos son opcionales, pudiendo incluso generar un modelo de canal en el que no existan *shadowing* y/o *multipath.* Por otro lado, cabe descatar que no existe una implementación del modelo Suzuki. La principal razón es que se puede modelar como la combinación de un canal con *shadowing lognormal* y *multipath Rayleigh.*

Para cada efecto, existe un atributo (cuyo nombre es el del efecto) en el que se guarda como string el tipo de efecto que se aplica. Por ejemplo, canal.pathloss=’fspl’, donde fspl son las siglas de *Free Space PathLoss*. Además, cada efecto dispone de otro campo en el que se guardan los correspondientes parámetros con forma de *cell*. Por ejemplo, en el atributo canal.multipathParams se guardaría un *cell* con los parámetros de la distribución seleccionada, como puede ser el parámetro b de Rayleigh.

Aparte de los atributos referentes al modelado del canal, también existe un vector de distancias en base logarítmica. Este vector es necesario ya que a la hora de calcular el *shadowing*, no vale con generar un número aleatorio. Esto es debido a que el *shadowing* varía lentamente con respecto a la distancia, por lo que hay que generar un vector completo para modelar correctamente este efecto.

Los métodos que componen esta clase son sencillos. Existen los métodos set para cada efecto, que se encargan de establecer los parámetros necesarios para dicho modelo, y comprobar que realmente son válidos. Por otro lado, están los métodos de cálculo. Estos se encargan de calcular la contribución del correspondiente efecto. En el caso del *pathloss*, basta con sustituir la distancia y la frecuencia en la correspondiente ecuación vista en el apartado de teoría. Para *multipath*, solo hay que generar números aleatorios. El más complicado de calcular es el *shadowing*, que se explica a continuación.

En primer lugar, se genera un vector de distancias, como se menciona anteriormente. Para delimitarlo, es necesario saber cuál es la distancia más grande que se puede alcanzar en el mapa, que no es más que la diagonal. La clase mapa dispone de una función que devuelve esta distancia. Por tanto, si el tamaño del mapa varía en algún momento, es importante notificar a la clase *Canal* de cambio para generar un nuevo vector de distancias. Si los parámetros de *shadowing* están establecidos, se puede generar el vector de *shadowing*. Para ello, se toman 20 valores por década del vector de distancias, y se generan números aleatorios de la correspondiente distribución (lognormal) para ellos. Tras esto, se hace una interpolación de los vectores pequeños, con lo que se genera un vector de *shadowing* suavizado de las dimensiones del vector de distancias.

Por otro lado, para calcular el *shadowing* dada una distancia, basta con buscar el índice de la distancia más parecida dentro del vector de distancias, y extraer del vector de shadowing el valor que ocupa esa posición. El algoritmo es el siguiente:

[~,index]=min(abs(this.d-d(i))); p(i)=this.shadowingVector(index);

Existe también una función que devuelve una matriz donde cada fila representa la contribución de cada efecto

Otra de las funciones de la clase Canal es devolver el valor de la distancia óptima (Véase Eficiencia energética en redes colaborativas). Para ello, se deben pasar como parámetro los dos nodos implicados en la comunicación, y el método se encarga de calcular la distancia óptima en función del modelo de *pathloss* establecido. Hay que mencionar que en la implementación de la formula se añade una variación, y es que a la sensibilidad se le suma un umbral de dB. Es decir, la potencia de transmisión debe satisfacer la sensibilidad del receptor más un cierto umbral, que garantiza que los efectos de *shadowing* y multipath no estropeen la comunicación. La fórmula quedaría de la siguiente forma:

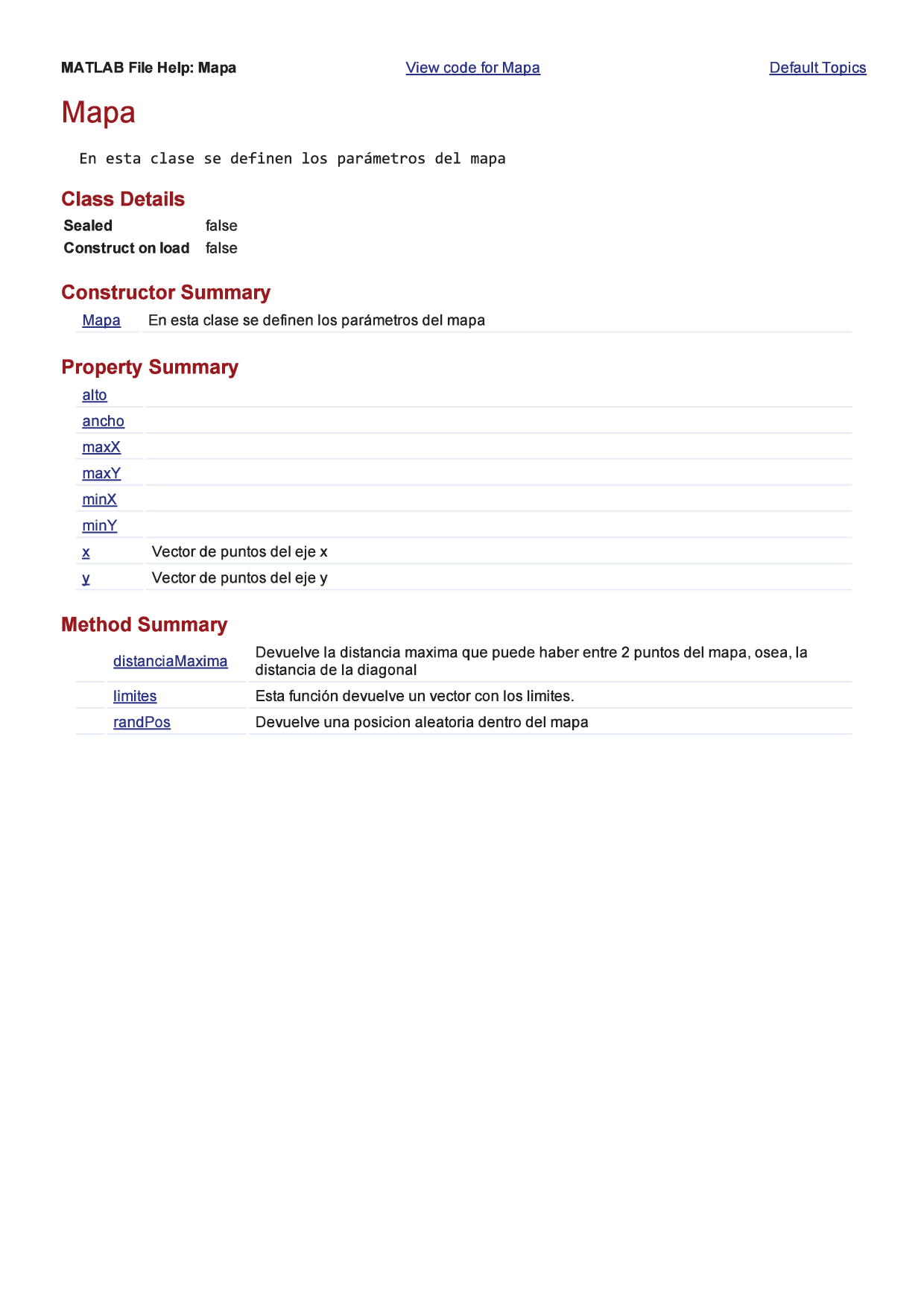
( 21 )

Este cambio también se puede observar en la forma en la que se calcula la potencia de transmisión en la clase **Transmisor**.

Por último, la clase Canal también es la encargada de calcular este umbral previamente mencionado. El umbral se define como la potencia que hay que añadir en transmisión para garantizar que el 98% de las veces se va a vencer al efecto del *shadowing*+*multipath.* Para calcularlo, se toma el vector de *shadowing* y se le añade el efecto multipath. Este nuevo vector alberga muestras de la variación del efecto total del canal con respecto al efecto del *pathloss*. Para calcular el umbral, se hace un histograma de este vector y se divide entre el número total de muestras, lo que nos da la PDF de las variaciones. Posteriormente, utilizando *cumsum* se genera CDF. En esta CDF, hay que buscar cual es el valor tiene un 2% de probabilidad, y este será nuestro umbral. Es decir, el umbral es la potencia que hay que añadir en transmisión para garantizar que el 98% de las veces se va a vencer al efecto del *shadowing*+*multipath*

1. MAPA

Ilustración 16: Resumen de la clase Mapa



Esta clase se encarga de definir el mapa, que es la zona espacial rectangular en la que se encuentran los nodos.

Para crear un mapa, basta con definir su alto, ancho y densidad de puntos. Densidad de puntos es un parámetro que sirve para generar posteriormente una matriz. El valor representa cuantos puntos por metro se obtienen en cada eje. Por ejemplo, un mapa 100x100 con densidad de puntos 2, tendría un vector x de longitud 200, la misma dimensión que y.

Además, esta clase dispone de la función *distanciaMaxima()* para obtener la mayor distancia dentro del mapa (que no es más que la diagonal). También dispone de la función *limites()* que devuelve los límites del mapa, muy útil para utilizarla cuando se usa la función *axis* en la representación gráfica del mapa.

Por último, la función *randPos()* devuelve un punto aleatorio dentro del mapa, lo cual es útil a la hora de generar nodos que se encuentren en posiciones aleatorias garantizando que estas se encuentran dentro del mapa.

1. ENCAMINAMIENTO

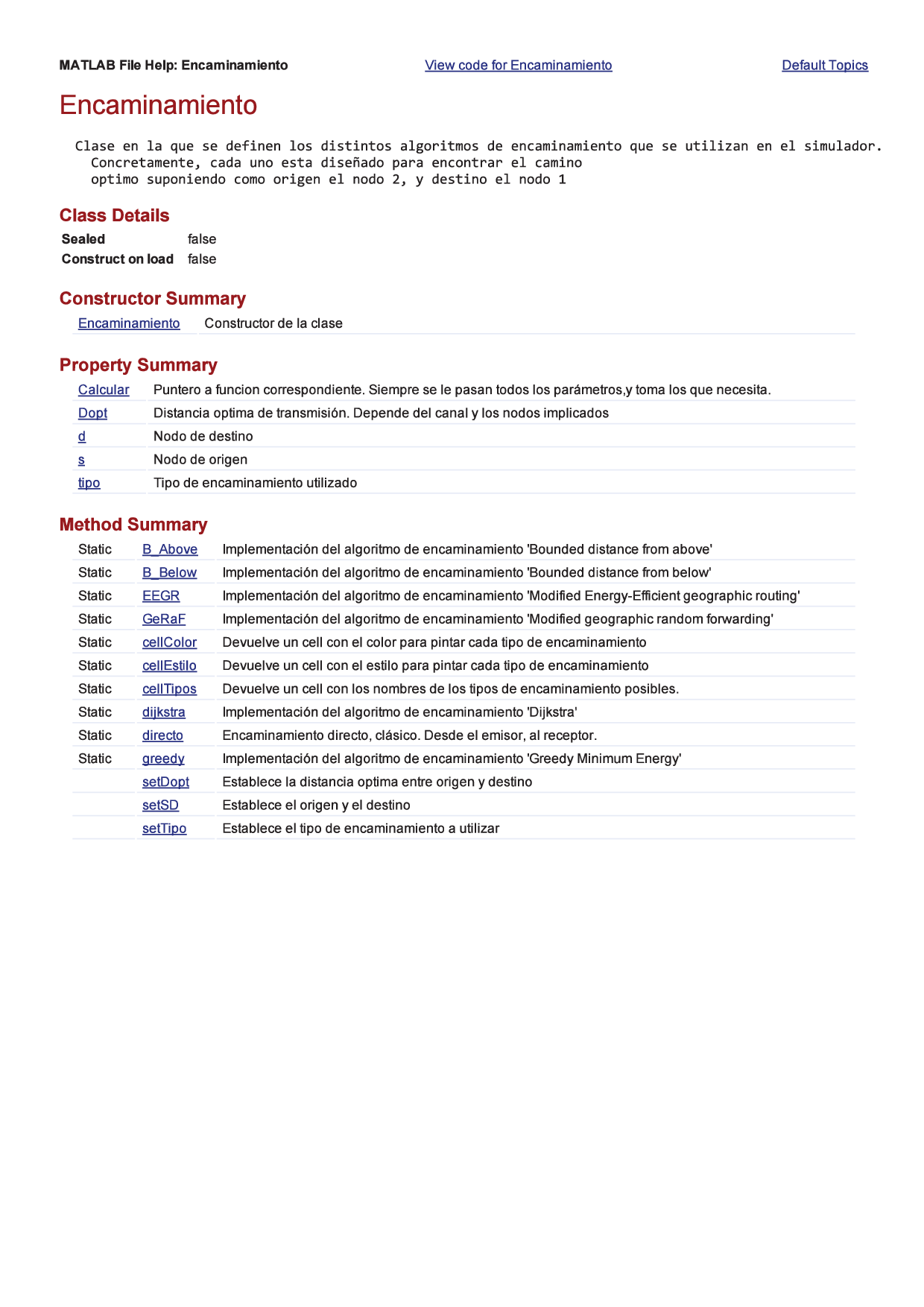


Ilustración 17: Resumen de la clase Encaminamiento

Esta clase se encarga de definir los algoritmos de encaminamiento. Además, si se crea un objeto de esta clase, este define el algoritmo que se está usando, y el puntero a la función correspondiente para calcularlo. De esta forma, basta con llamar al método calcular, y el objeto automáticamente llama a la función correspondiente según el tipo de encaminamiento seleccionado.

En esta clase se implementan los siguientes algoritmos:

* Directo
* Greedy
* GeRaF
* EEGR
* Dijkstra

A todos los algoritmos se le pasan los mismos parámetros para calcular la ruta correspondiente. Si bien es cierto que algunos obvian ciertos parámetros. Los parámetros son los siguientes

***This***: el propio objeto de la clase Encaminamiento. Necesario ya que dispone del número los nodos de origen y destino.

***Dist***: Matriz cuadra simétrica. Cada elemento representa la distancia entre el nodo i y j.

Consumo: Es una matriz a la que se accede de la forma (*rx*,*tx*). En la diagonal, se encuentra la potencia eléctrica del nodo en cuestión (ya que no tiene sentido que un nodo se transmita a si mismo). Fuera de la diagonal, se encuentra la potencia de transmisión del nodo *tx* cuando manda información a *rx* más la potencia por estar encendido. De esta matriz se obtienen los costes.

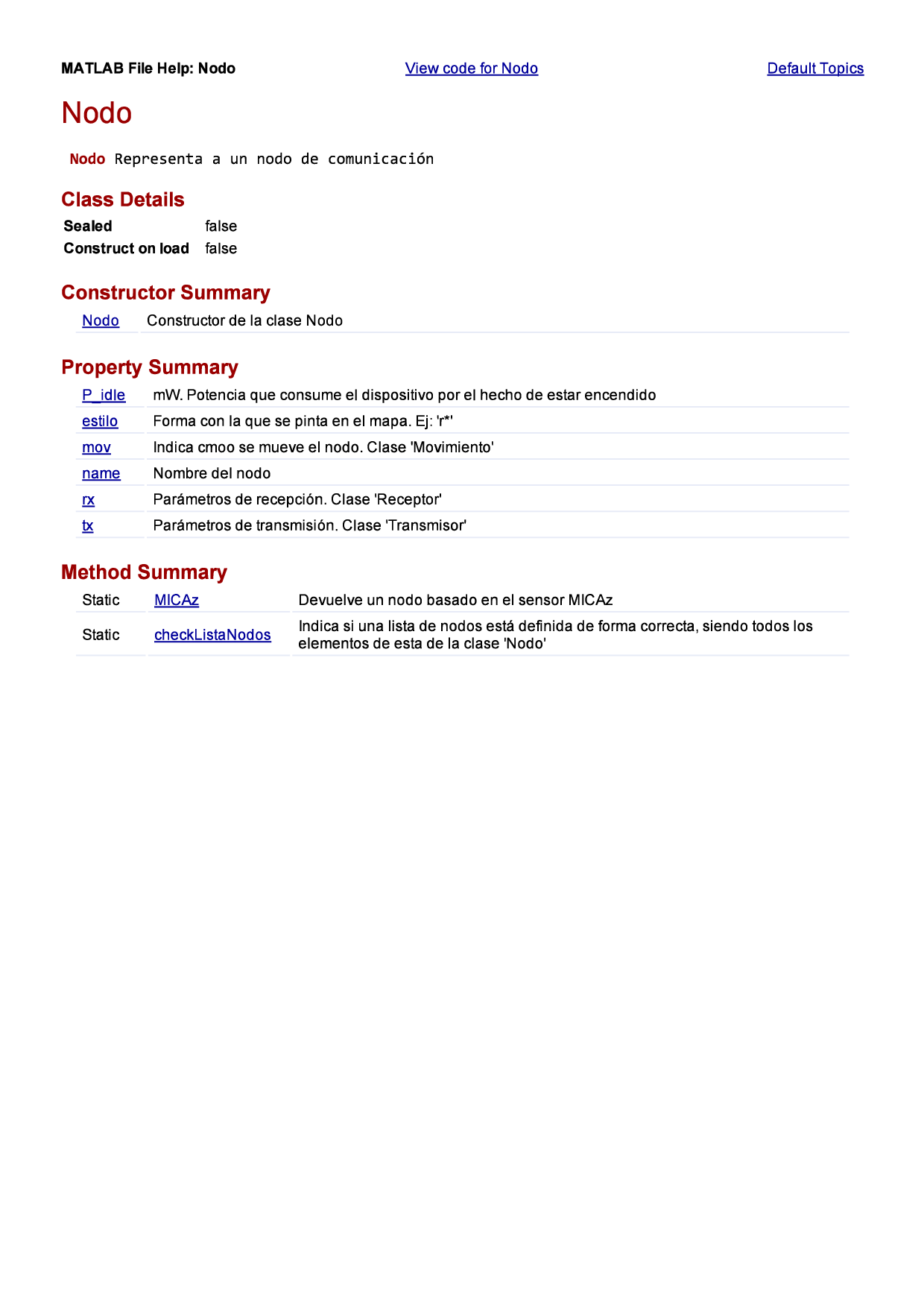
***Cobertura***: es una matriz de tipo booleano a la que se accede de la forma (*rx*,*tx*). Muestra si el nodo *rx* tiene cobertura cuando transmite *tx*. No tiene por qué ser simétrica, ya que es posible que en un momento dado, por efectos de multipath un nodo reciba suficiente potencia y el otro no.

Finalmente, se dispone de algunos métodos estáticos, que sirven de cierta utilidad a la hora de trabajar con interfaces gráficas. Básicamente, devuelven *cells*  con texto, que es útil a la hora de rellenar menús desplegables, o a la hora de representar gráficos con colores y formatos distintos.

Mencionar que, en caso de no ser posible el establecimiento de una ruta, se genera un error. De modo que es conveniente llamar a la función *calcular()* dentro de un bloque try-catch para controlar que resultados asignar al coste y la ruta.

1. NODO

Ilustración 18: Resumen de la clase Nodo



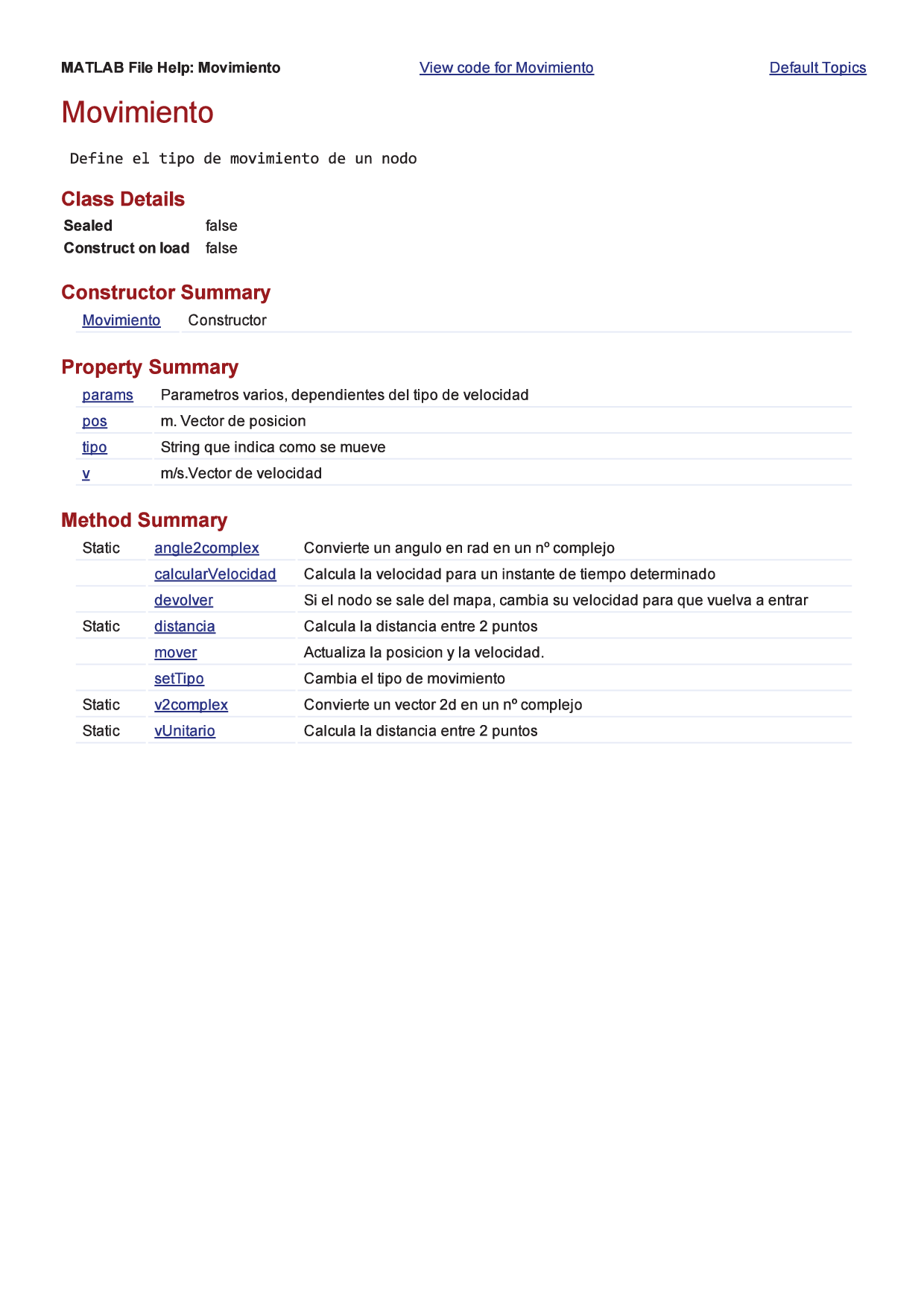
El uso de esta clase permite definir un conjunto de nodos pertenecientes a un escenario, y que cada uno tenga características distintas. Este conjunto de nodos debe definirse como un *cell*. Dentro de este debe haber tantos nodos como longitud tenga. Para comprobar que un cell de nodos es correcto, se dispone de una función estática llamada *checkListaNodos().*

Por otro lado, la función *MICAz()* es un constructor alternativo, que crea un nodo basado en el sensor del mismo nombre. Este tipo de sensor se utiliza en

Al igual que *MICAz(),* esta es la clase idónea para definir constructores de nodos personalizados, como funciones estáticas, de forma que se puedan crear redes sin tener que editar todos los parámetros necesarios de uno en uno para cada nodo predeterminado .

1. MOVIMIENTO

Ilustración 19: Resumen de la clase Movimiento



Esta clase es la encargada de modelar el movimiento de los nodos, definiendo su posición geográfica y la forma en la que se mueven. Cada objeto de la clase nodo debe disponer de un atributo *mov*, que es un objeto de esta clase, *Movimiento.* La clase dispone de los siguientes atributos.

Se implementan los siguientes tipos de movimiento:

* Fijo
* Aleatorio
* Rectilíneo
* Circular

Una de las características fundamentales de esta clase es que la definición de velocidad y posición se hace de forma vectorial. La razón por la que se hace así es porque Matlab está especialmente diseñado para hacer cálculos con matrices y vectores, lo cual implica una mayor facilidad de uso y rendimiento si se trabaja vectorialmente.

El constructor de la clase genera un movimiento predeterminado. Si se desea cambiar, existe una función *SetTipo*, que se encarga de establecer el tipo de movimiento deseado de entre los que aparecen anteriormente.

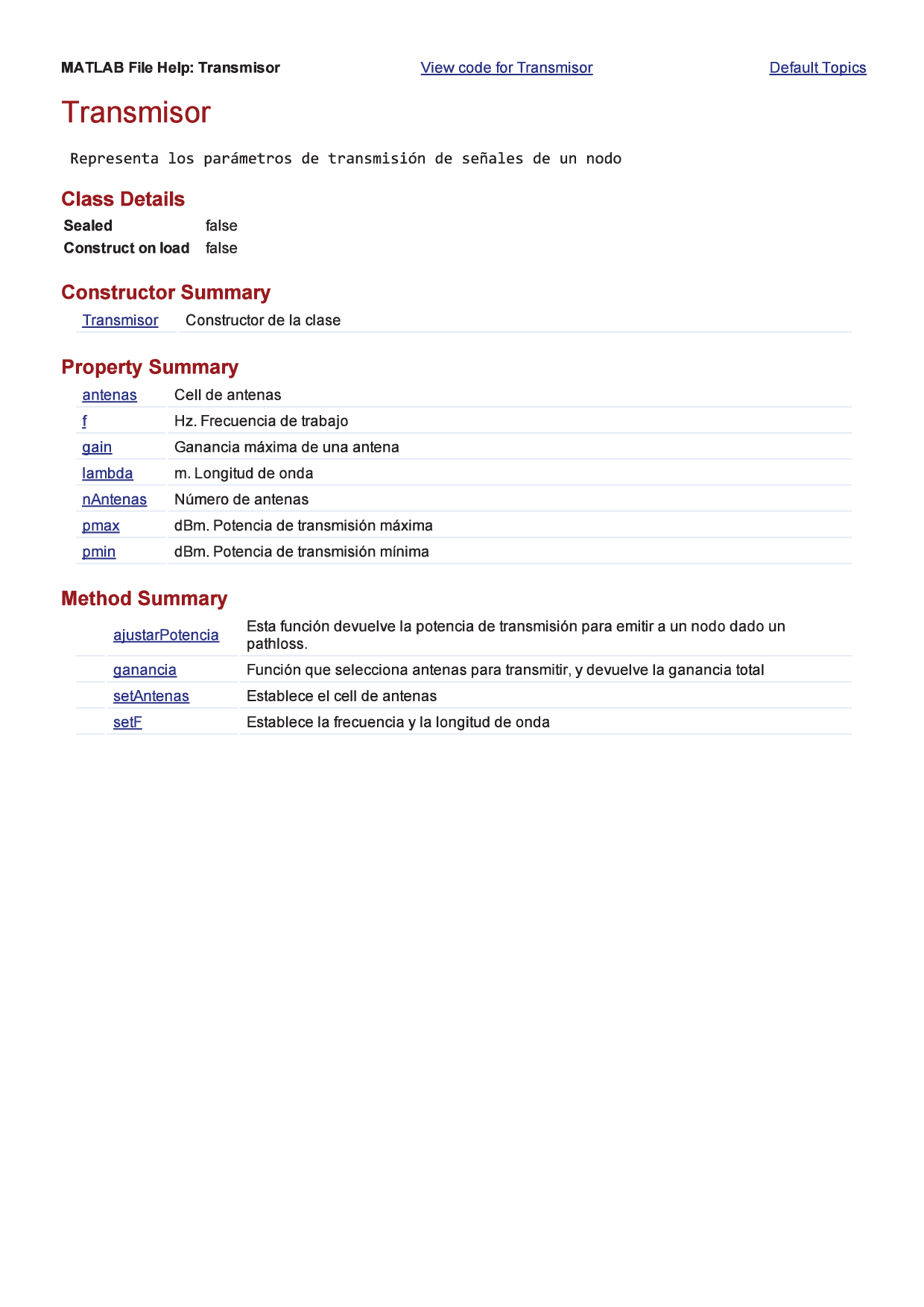
Una de los principales métodos de esta clase es *mover()*. Este se encarga, como su propio nombre indica, de hacer que la posición cambie. Para ello, en primer lugar, actualiza el vector de velocidad (haciendo uso de la función *calcularVelocidad().* A continuación, basta con sumar a la posición actual el vector de velocidad multiplicado por un valor *dt* que representa el incremento de tiempo.

El diseño de estos métodos está orientado para que mover los nodos sea lo más fácil posible desde el punto de vista del bucle de simulación principal. De esta forma, solo bastaría con llamar al método *mover()* de cada nodo dentro del bucle de simulación para generar el movimiento de los nodos.

Finalmente, se incluyen otros métodos estáticos adicionales que son útiles para realizar cálculos que tengan que ver con posiciones. Un ejemplo es el método *distancia()*, que se encarga de devolver la distancia entre 2 puntos (posiciones). Por otro lado, también se implementan métodos que convierten ángulos y vectores en números complejos. Esto será útil más adelante a la hora de calcular el ángulo de visión de las antenas de los nodos.

1. TRANSMISOR

Ilustración 20: Resumen de la clase Transmisor



Esta clase representa los parámetros referentes a transmisión de señales de un nodo.

En cuanto a los métodos más reseñables, se encuentra en primer lugar *ajustarPotencia()*. El objetivo de la función es calcular a que potencia debe de transmitir un nodo para establecer una comunicación con un nodo de destino, de tal forma que se transmita con suficiente potencia como para que el receptor capte la señal, y a la vez no desaprovechar energía.

Para realizar el cálculo, se impone que al receptor debe llegarle una potencia igual a su sensibilidad más un cierto umbral, que garantice que la señal no se desvanezca por efectos de canal. Para ello, el nodo transmisor debe conocer las ganancias de las antenas tanto receptora como transmisora, y la distancia a la que se encuentra el nodo de destino. A partir de esta distancia, se calcula un *pathloss*, De tal forma, que la ecuación final del cálculo quedaría de la siguiente manera:

( 22 )

Siendo el producto de las ganancias de recepción y transmisión. Todas las unidades de los miembros de la ecuación anterior se miden en decibelios.

El otro de los algoritmos destacables de esta clase es el del método *ganancia()*. En este método, se calcula la ganancia de la antena o antenas para un instante determinado. La importancia de este método se debe a que el transmisor puede disponer de varias antenas, que además pueden tener distintos diagramas de radiación y orientación. De esta forma., el método *ganancia()* se encarga de generar todos los posibles patrones de radiación del transmisor probando todas las posibles combinaciones entre estos, para posteriormente seleccionar el óptimo. A continuación y en primer lugar, se muestra el algoritmo con el que se obtiene la ganancia de cada antena.

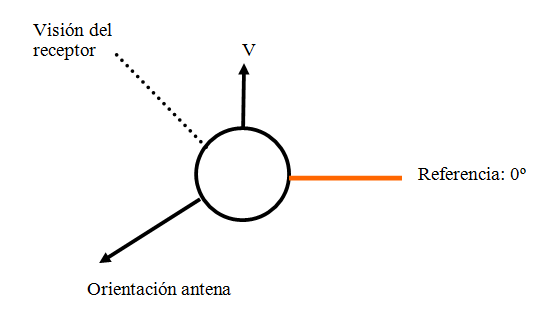


Ilustración 21: Diagrama de nodo con antenas direccionales

El objetivo es saber el ángulo de visión del receptor con respecto a la antena. Para ello, disponemos del ángulo de velocidad del nodo con respecto a la referencia, el ángulo de visión con respecto a la referencia y el ángulo de visión del receptor con respecto a V(Este ángulo se define en la clase **Antena**). En primer lugar, se calcula el ángulo de orientación de la antena con respecto a la referencia, como se muestra a continuación:

( 23 )

Siendo el ángulo de la antena con respecto a la velocidad. Una vez obtenido , se procede a obtener el ángulo con el que la antena ve al receptor. Para ello, hay que tener en cuenta que el diagrama de radiación de una antena se define en el intervalo , y que si el objetivo se ve por la derecha el ángulo debe ser positivo (es decir, intervalo ). De esta fórmula, el ángulo de visión se calcula:

( 24 )

Así, es el ángulo que hay que introducir a la función del patrón de radiación para obtener la ganancia de esa dirección.

Hay que tener en cuenta que el transmisor puede tener asociadas varias antenas, luego la función *ganancia()* debe calcular las ganancias para cada antena asociada de forma individual. El siguiente paso, en caso de que existan varias antenas, es establecer las antenas por las que hay que transmitir. En primer lugar, hay que tener en cuenta que si el transmisor dispone de n antenas, la potencia con la que transmite cada antena debe dividirse entre n. De esta forma, la potencia total transmitida al medio en una dirección quedaría de la siguiente forma:

( 25 )

Representando la ganancia de cada antena en la dirección determinada, la potencia de transmisión y el número de antenas. Ajustando, la potencia total se puede calcular como:

( 26 )

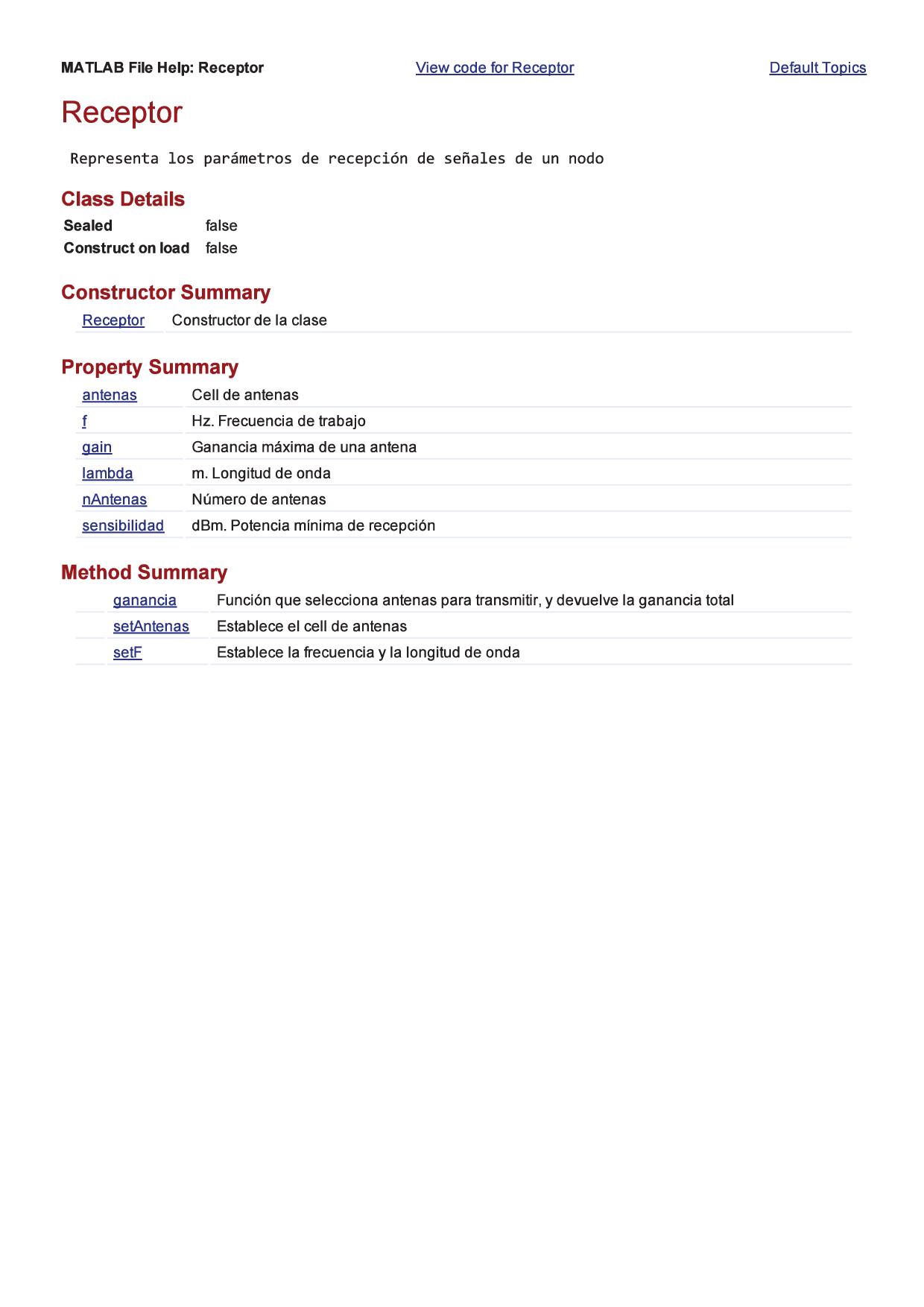
Este parámetro G es la ganancia del conjunto de antenas en una dirección.

Para calcular esta G en el simulador, se hacen combinaciones de las distintas ganancias obtenidas anteriormente. Estas combinaciones se hacen debido a que, en ciertos casos, si la contribución de una antena es 0, no es necesario transmitir por esta, ya que no mejora la potencia que obtendrá el receptor, y en un escenario real generaría interferencias. Por otro lado, en el simulador solo se desactivan las antenas cuya contribución es nula, pero gracias al sistema de generación de todas las posibles combinaciones, en un futuro se podría establecer un umbral, de modo que si el aporte de una antena no es nulo, pero es mínimo, también esta sea desactivada.

Cabe mencionar que se hace uso de números complejos para realizar los cálculos. De esta forma, los ángulos se almacenan como números complejos, y las sumas pasan a ser productos y las resta divisiones. Finalmente, se hace uso de la función *angle()*, que devuelve el ángulo de un número complejo en radiandes, en el intervalo .

1. RECEPTOR

Ilustración 22: Resumen de la clase Receptor



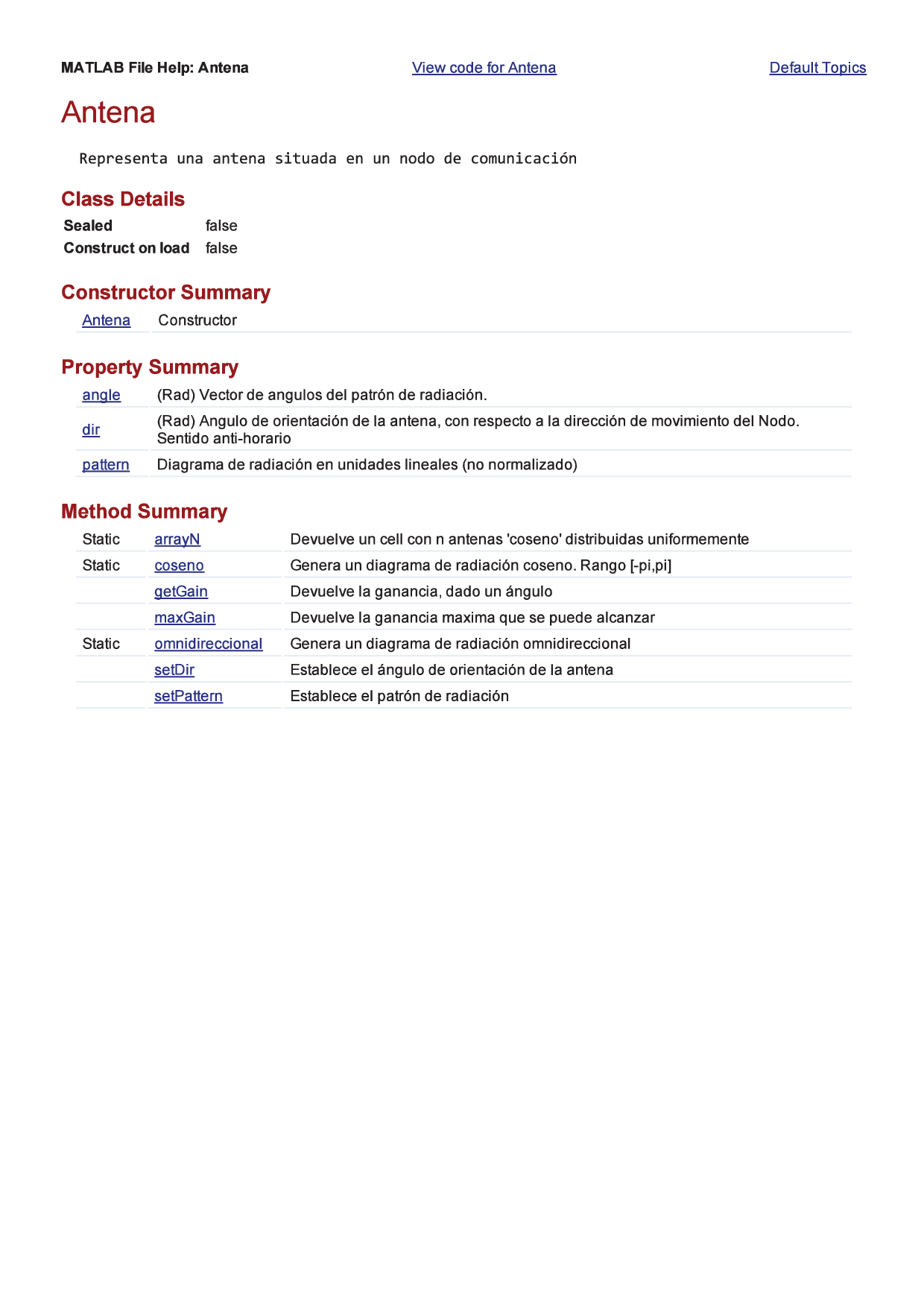
Esta clase es muy similar a la clase Transmisor. La principal diferencia es que, en este caso, el parámetro que se establece es la sensibilidad (en lugar de *Pmax* y *Pmin*), o lo que es lo mismo, la potencia mínima de recepción. En cuanto a los métodos implementados, una copia de los de la clase **Transmisor**, debido a que, por ejemplo, en el caso del método *ganancia()*, la obtención de la ganancia es igual en ambos casos.

Por otro lado, cabe destacar que en el simulador, tanto **Receptor** como **Transmisor** tienen un conjunto de antenas distinto. Se podría haber establecido de otra forma, asignando un conjunto de antenas a cada nodo, pero se implementa con así ya que de esta forma se da la libertad al usuario de poder definir nodos que utilizan antenas distintas para transmisión y recepción.

Un método que se puede echar en falta en esta clase es aquel dedicado al cálculo de potencia de recepción, análogo al del cálculo de potencia de transmisión en la correspondiente clase. La razón por la que no se implementa es que sería una función en la cual solo se tendría que hacer el producto (o suma si se trabaja en dB) de los distintos parámetros que se le pasen, a saber partencia del transmisor, perdidas de canal y ganancias de los nodos implicados. Por ello, este método no es necesario, ya que es preferible realizar el cálculo en el momento que sea preciso a crear una función tan simple, que lo único que haría sería empeorar el rendimiento del simulador.

1. ANTENA

Ilustración 23: Resumen de la clase Antena



Esta clase define a una antena. Para ello se dispone de su patrón de radiación, el vector de ángulos de dicho patrón y el ángulo de orientación, que es el ángulo hacia el cual está orientada la antena tomando como referencia la dirección de movimiento del nodo (o, lo que es lo mismo, la dirección de su vector de velocidad). Estos parámetros ya se mencionan en la descripción del algoritmo de obtención de ganancia, en la clase **Transmisor**.

En este caso, el algoritmo más destacable es el de obtención de la ganancia dado un ángulo . Para ello, se busca en el vector de ángulos el índice del ángulo más parecido a (ya que es posible que el ángulo no esté definido de forma exacta en el vector), y se devuelve el valor de ganancia que ocupa dicha posición en el vector *pattern*. Esta manera de obtener la ganancia, permite que se puedan simplificar los valores de los vectores de ángulos de los patrones de radiación. Por ejemplo, si se define un patrón de radiación en el intervalo , y se pide la ganancia para , el método devolverá la ganancia para que es el ángulo más cercano. De esta forma, se puede definir un patrón de radiación omnidireccional con un solo punto, ya que siempre va a devolver el valor de dicho punto.

Aparte de los métodos *get* y *set* que se pueden ver en el resumen, se dispone de métodos estáticos para crear patrones de radiación, tanto omnidireccional como coseno alzado. También se dispone del método *arrayN()*, que permite generar un cell de n antenas de tipo coseno alzado separadas uniformemente.

Finalmente, mencionar la existencia de un *toolbox* de Matlab llamado ‘*Antenna toolbox’* diseñado específicamente para el modelado de antenas. De esta forma, se puede utilizar dicho *toolbox* para diseñar una antena concreta, y obtener su correspondiente diagrama de radiación, en forma de vectores de ángulos y patrón, y estos vectores pueden asignarse de igual forma a la presente clase **Antena.**

1. INTERFACES GRAFICAS

En este apartado, se comentan las distintas interfaces gráficas implementadas para el simulador. Algunas de ellas tienen como objetivo servir de interfaz para la creación o edición de objetos de las clases anteriormente presentadas, y otras son la base del simulador. Es aconsejable ver el **Apéndice C**, en el que se explican los fundamentos de la programación de interfaces gráficas en Matlab**.** A continuación, se describe cada una de estas interfaces:

1. CANALUI

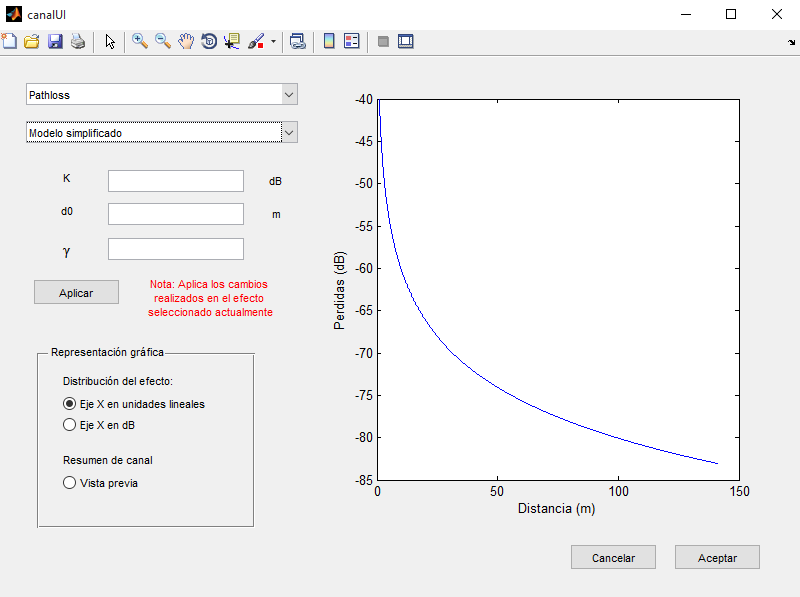


Ilustración 24: Interfaz principal CanalUI

Esta interfaz gráfica facilita la edición de un objeto de la clase **Canal.** Es de tipo *singleton* y bloqueante. Como se puede apreciar en la anterior imagen, la interfaz dispone de un panel lateral donde se selecciona el efecto deseado y su correspondiente. A continuación, aparecen unos cuadros de texto en los que se deben insertar los valores deseados para dicho para efecto-modelo. Acto seguido, aparece un botón para aplicar los valores que han sido introducidos en los cuadros de texto. Debajo, hay un panel que controla las opciones de visualización. Por último, a la derecha se puede observar un elemento gráfico, que es el encargado de representar las distintas gráficas correspondientes a la selección del momento y debajo de este elemento, se encuentran los botones de ‘*Aceptar’* y ‘*Cancelar’*, para guardar los cambios realizados o descartarlos respectivamente.

Para usar esta interfaz, es necesario llamarla pasándole como parámetro un objeto de la clase Canal. De esta forma, cuando se pulsa ‘*Aceptar’* se devuelve el objeto con los correspondientes cambios realizados, mientras que si se pulsa ‘*Cancelar’* se devuelve el mismo objeto sin alterar.

Para el caso del *pathloss*, en la gráfica se puede ver su valor en decibelios. Si se elige la opción *‘Eje X en dB’*, en este caso, el eje X pasaría a representarse en décadas. Este es un caso especial, ya que en el eje X se encuentra la distancia, que realmente tiene más sentido ser representada en décadas que en dB. A continuación, se muestra cómo quedaría la gráfica seleccionando esa opción:

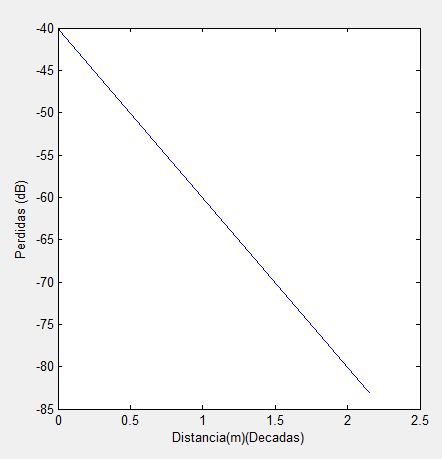


Ilustración 25: Pathloss con distancia en decadas

Se puede apreciar que al representarse la distancia en décadas, la curva pasa a ser una resta, como cabía esperar.

En cuanto a los efectos *shadowing* y *multipath*, en lugar de hacer una representación frente a la distancia, debido a la naturaleza estadística de los modelos que se implementan se representa la *PDF* de la correspondiente distribución dados sus correspondientes parámetros. En este caso, sí que se muestra el eje X en unidades lineales o en dB. A continuación, se muestran los gráficos de una distribución Rayleigh para :

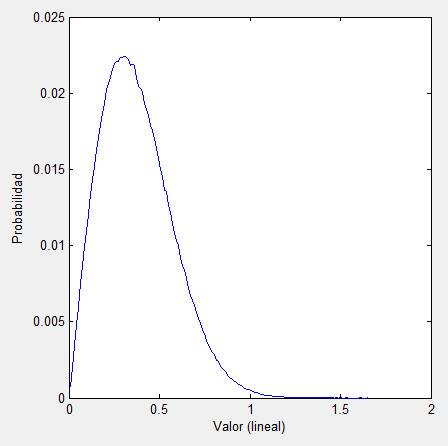


Ilustración 26: Distribución con eje x lineal

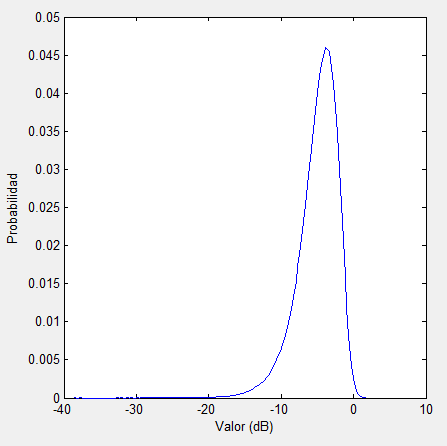


Ilustración 27: Distribución con eje x en dB

Por último en lo referente al apartado gráfico, se dispone de la opción de mostrar una vista preliminar del modelo final del canal, mostrando una gráfica similar a la que se puede encontrar en la ilustración 3, donde se puede observar la contribución de los distintos efectos a el modelo final de canal. A continuación, se muestra una captura:

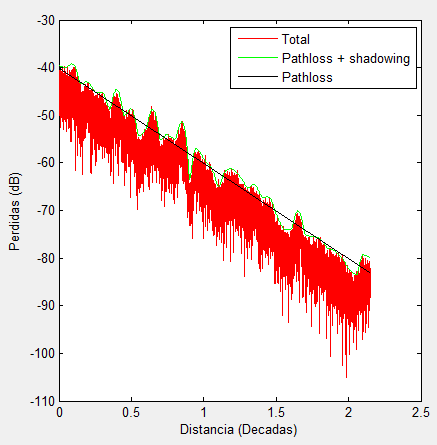


Ilustración 28: Resumen del modelo de canal final

En este caso, como se puede observar, se representa el valor de las perdidas en dB con respecto a la distancia en décadas. Se puede observar que las pérdidas totales tienen una forma un tanto rara. Esto es debido a que se toman muchas muestras y los valores varían mucho, pero esto puede ayudar a ver cuál es la variación máxima con respecto a los otros efectos (picos), además de dar una idea aproximada de que valores suele tomar (parte ancha).

1. MAPAUI

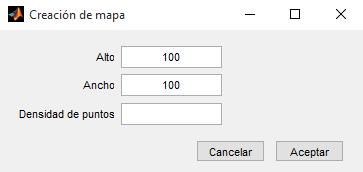


Ilustración 29: Interfaz principal de MapaUI

Esta es una interfaz simple cuyo objetivo es la edición de un objeto de la clase **Mapa**. Es de tipo *singleton* y bloqueante Para llamar a esta interfaz, es necesario pasarle como parámetro un objeto de dicha clase. Una vez abierta, basta con definir los tres parámetros (alto, ancho y densidad de puntos) que son necesarios para el constructor de la clase. De esta forma, si se pulsa ‘*Aceptar’* ,se devuelve un nuevo objeto creado a parir de dichos parámetros mientras que si se pulsa ‘*Cancelar’* se devuelve el mismo objeto que se pasó inicialmente.

1. NODOUI

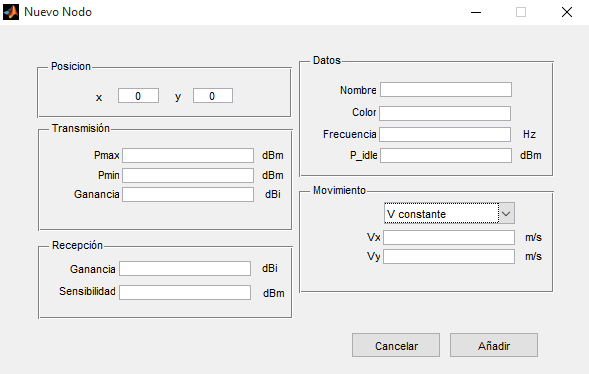


Ilustración 30: Interfaz principal de NodoUI

Esta es la interfaz asociada a la clase **Nodo**, y tiene como objetivo la edición de un objeto de dicha clase. Es de tipo *singleton* y bloqueante. Como se puede observar, dispone de un conjunto de paneles dentro de los cuales hay cuadros de texto que tienen como objetivo la introducción de los valores que definirán al nodo. Al igual que en las anteriores interfaces gráficas mencionadas, dispone de los botones de *‘Aceptar’* y *‘Cancelar’,* y el funcionamiento es el mismo que en esos casos.

Cabe destacar que esta clase no implementa todas las posibilidades que existen a la hora de crear un nodo a través de la API. Concretamente, solo se permite la creación de un nodo con una antena para transmisión y otra para recepción, ambas de tipo omnidireccional. En cuanto al movimiento, solo se permite establecer los de tipo ‘fijo’, ‘aleatorio’ y ‘velocidad constante’.

En cuanto a las funciones subyacentes, están las que se encargan de leer los datos de los cuadros de texto, validarlos, y almacenarlos de tal forma que puedan ser utilizados a la hora de interactuar con el objeto de tipo **Nodo** que se está editando.

1. FIGURARECEPCION

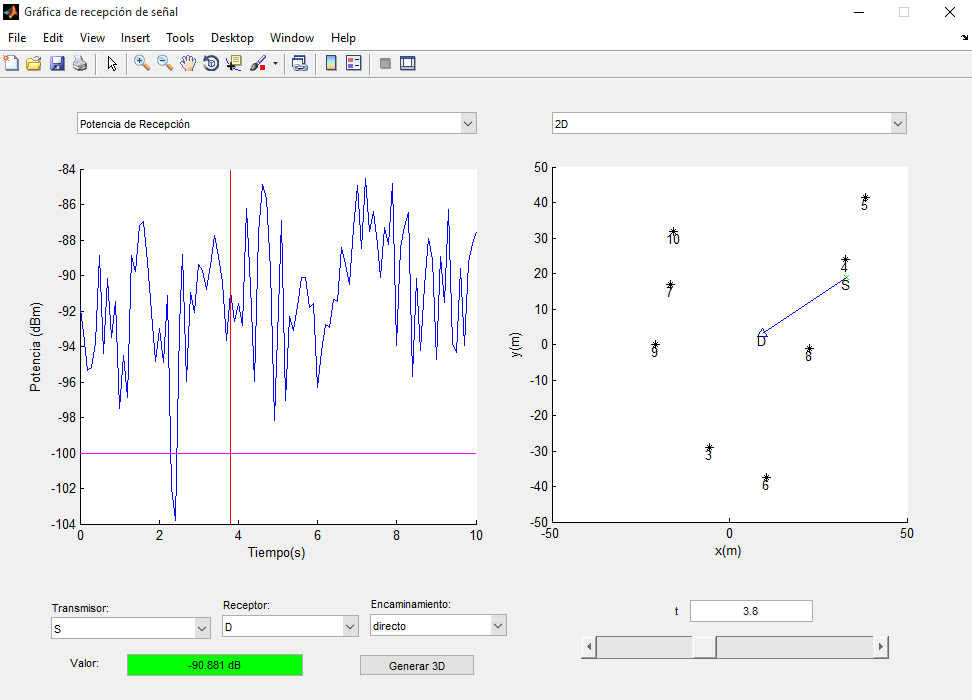


Ilustración 31: Interfaz principal de figuraRecepcion

En esta interfaz es donde aparecen los resultados de una simulación simple. No es *singleton* ni bloqueante. Dispone de 2 paneles gráficos. El de la izquierda es el encargado de mostrar los gráficos seleccionados por el menú que tiene justo encima, y el de la derecha es el encargado de mostrar el mapa, ya sea en 2 dimensiones o 3(siempre y cuando se haya realizado una simulación en 3D). En ese mapa se puede observar la distribución de los nodos y el camino que se establece según el algoritmo indicado. En La parte inferior izquierda de la interfaz se encuentran una serie de controles cuyo objetivo es seleccionar el nodo transmisor, el nodo receptor y el tipo de encaminamiento utilizado, respectivamente. Además, se cuenta con un cuadro de texto que va mostrando el valor de la gráfica para el momento seleccionado. También mencionar la existencia de un botón cuya función es generar un mapa con la potencia de recepción en cada punto (tridimensionalmente) y una copia del mapa en el momento actual marcando las zonas donde hay cobertura o no. Por último, a la derecha se encuentra una barra de deslizamiento cuyo objetivo es controlar el instante de ejecución. De este modo, se puede avanzar o retroceder en el tiempo hasta un momento concreto para analizarlo concretamente.

Esta interfaz tiene 2 modos de funcionamiento: dependiente o independiente. Al llamar a la función que la genera, esta devuelve un puntero a la figura. De forma predeterminada, la interfaz se inicia de en modo dependiente, de tal forma que sea la interfaz principal la encargada de dibujar los resultados en tiempo real. Además, en este caso, la interfaz principal solo puede pintar el mapa y la recepción dados un transmisor y un receptor. El resto de opciones quedan desactivadas. Una vez que termina la simulación (o se cancela), la interfaz principal debe llamar a la función *loadData()* pasando una serie de datos que permitan a **figuraRecepcion** funcionar de manera autónoma o independiente. A partir de este momento, la figura parsea estos datos y puede funcionar al completo. Adicionalmente, esta figura se puede guardar, incluyendo los datos de la simulación. Esto es útil de cara a almacenar distintas simulaciones o incluso compararlas.

Finalmente, se procede a presentar los distintos gráficos que puede representar la figura:

* **Potencia de recepción**

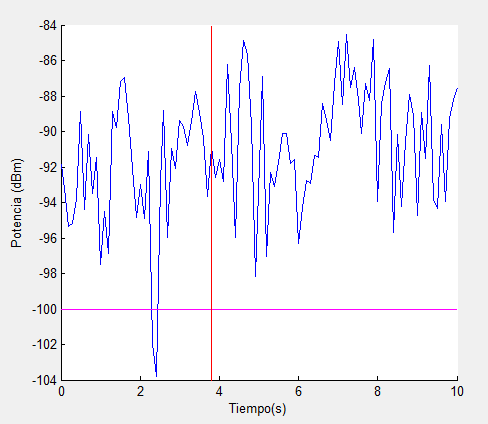


Ilustración 32: Potencia de recepción de un nodo

Muestra la potencia que recibe un nodo (Receptor) cuando transmite un nodo (Transmisor) Es independiente del encaminamiento utilizado. Además se muestra una línea horizontal en color magenta que representa la sensibilidad del receptor, de modo que mientras se está por encima de ese nivel hay cobertura, y por debajo no. Mencionar que en este caso, el cuadro de texto situado debajo del gráfico cambia de color dependiendo si se está en cobertura o no, poniéndose verde si hay o rojo en caso contrario.

*Nota: La línea vertical roja muestra la selección de tiempo, y está controlada por el slider. Esta línea es común en todas las gráficas.*

* **Potencia de transmisión**

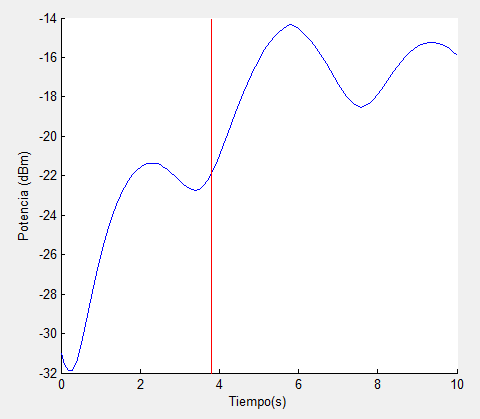


Ilustración 33: Potencia de transmisión de un nodo

Muestra la potencia que transmite un nodo (Transmisor) hacia otro nodo (Receptor) Es independiente del encaminamiento utilizado. Como se puede observar, las variaciones son más suaves que en el caso de la recepción. Esto se debe a que a la hora de calcular la potencia de transmisión no se utilizan valores concretos de *multipath*, sino un umbral. Para más información, ver la clase **Transmisor**.

* **Consumo energético**

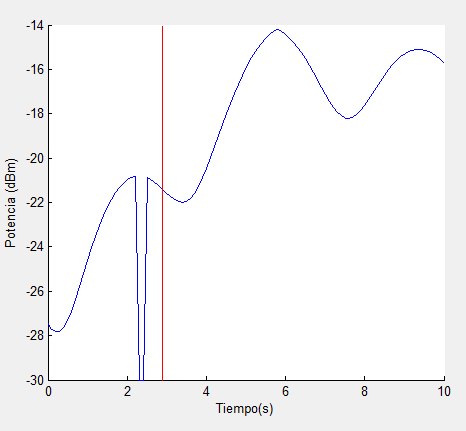


Ilustración 34: Consumo energético de un nodo

En este caso se muestra el consumo de un nodo (Transmisor) para una ruta concreta. El consumo de un nodo en una ruta es el valor de la potencia de transmisión más el del consumo eléctrico de dicho nodo. Cuando un nodo no forma parte de la ruta, su consumo es solamente el eléctrico. Se puede observar que la gráfica es muy similar a la anterior. Esto se debe a que se está utilizando un encaminamiento directo. Hay un punto en el que la señal deja de ser continua, y es que en ese momento se ha perdido la cobertura, cosa que puede pasar debido a los efectos de canal. Al no haber cobertura, no hay ruta, y el consumo pasa a ser el consumo eléctrico.

* **Número de saltos**

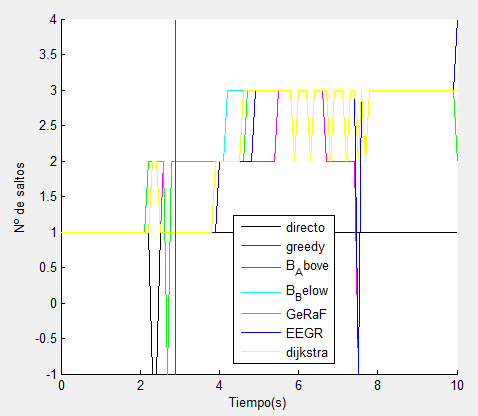


Ilustración 35: Número de saltos de las rutas

Aquí se representa el número de saltos para un tipo de encaminamiento, o para todos dependiendo de las opciones de simulación. De esta forma, se pueden comparar la reacción de los distintos algoritmos de encaminamiento para un mismo escenario. Cabe destacar que cuando el número de saltos es -1, esto significa que no es posible establecer una ruta en ese instante determinado.

* **Coste total de la ruta**

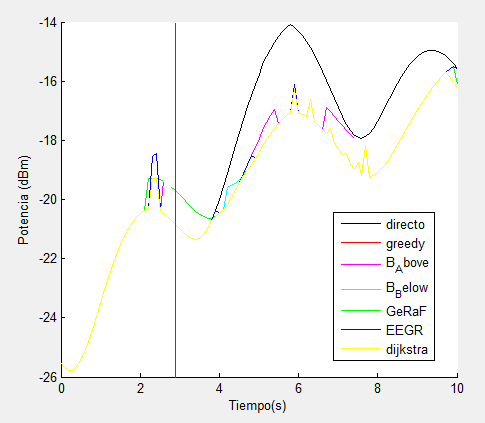


Ilustración 36: Coste de las rutas

En este caso, se representa el coste total de la ruta con respecto al tiempo. De este caso cabe destacar que, como se puede observar, *Dijkstra* siempre tiene el menor consumo, como se menciona en el apartado de teoría.

1. MAINUI



Ilustración 37: Interfaz principal deMainUI

Esta es la interfaz gráfica principal del simulador. Es una interfaz singleton, pero no bloqueante. Se encarga de controlar todos los parámetros de la simulación. Además, cuenta con una función, llamada *ejecutar(),* que es la función que se encarga de realizar las simulaciones, y, por tanto, es la pieza más importante de este simulador. Esta función se detalla en su propio apartado, debido a su gran importancia.

En cuanto a la interfaz, se puede observar que en la parte derecha dispone de un panel gráfico, que es el encargado de representar una vista previa del mapa. Si se pulsa con el botón derecho, permite añadir nodos. A continuación se muestra una captura del menú:

C:\Users\pablo\AppData\Local\Temp\x10sctmp1.png

Ilustración 38: Menu de pulsación de botón derecho sobre el mapa

Cuando se pulsa ‘Nuevo Nodo’, se abre una interfaz *NodoUI* para establecer los distintos parámetros del nodo. En caso de pulsar ‘Nuevo Nodo Rapido’ se crea un nodo con valores predeterminados en la posición actual del puntero.

En la parte superior izquierda se puede observar un menú desplegable titulado ‘Simulación’. En ese menú podemos encontrar las siguientes opciones:

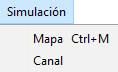


Ilustración 39: Menu de simulación

Concretamente, el botón ‘Mapa’ se encarga de abrir una instancia *MapaUI* para editar el mapa actual y ‘Canal’ abre una instancia *CanalUI* para editar el canal.

En la parte derecha de la ventana, podemos encontrar varios paneles. El primero que se puede ver es el de ‘Opciones de simulación’, que se muestra a continuación:

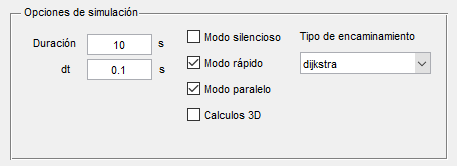


Ilustración 40: Panel de opciones de simulación

Aquí podemos seleccionar el tiempo total de simulación (Duración) y el intervalo de tiempo entre cálculos (dt). Además se pueden seleccionar lo siguientes modos de simulación:

* Modo silencioso: Muestra los resultados al final de la simulación. Si no se selecciona, se pueden ver resultados en tiempo de simulación.
* Modo rápido: Si está activado, solo se hacen cálculos con los nodos que son susceptibles de aparecer dentro de una ruta. Esto hace que se ahorre tiempo, pero por otro lado hay cálculos que no se realizan, y, por tanto, no pueden ser observados en la vista de resultados finales
* Modo paralelo: Con este modo activo, se realiza la simulación con todos los núcleos de los que disponga el procesador (o los que estén configurados para ser usados). Para hacer uso de este modo es necesario disponer del ‘*Parallel toolbox’* de Matlab.
* Cálculos 3D: Si se activa, se realizan cálculos de pérdidas de propagación para todo el mapa, teniendo en cuenta la densidad de puntos establecida en la creación del objeto **Mapa**. No es aconsejable su uso, ya que ralentiza mucho los cálculos y se dispone de otra herramienta para realizar estos cálculos una vez finalizada la simulación, como se comentaba en **figuraRecepcion**.

Por último, en este panel también se puede seleccionar el algoritmo de encaminamiento que va a ser utilizado en la simulación. También existe la posibilidad de ejecutar un barrido con los distintos algoritmos implementados.

El siguiente panel que aparece en la interfaz es el de ‘Scripts’. A continuación una captura de este:

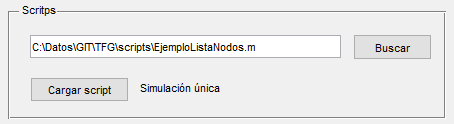


Ilustración 41: Panel de carga de scripts

Este panel es el encargado de cargar *scripts* de simulación. Los *scripts* son ficheros que al ser ejecutados preparan al simulador para hacer una simulación concreta. Los tipos de *scripts* y su funcionamiento se explican en su correspondiente apartado. Aquí, mencionar que existen 2 tipos: de única simulación y de varias simulaciones. Los de única simulación, se cargan, pueden ser editados en la interfaz, y al ejecutarse lanzan una instancia de **figuraRecepcion** con los resultados de la simulación, lo cual los hace idóneos para analizar un entorno concreto. Para los *scripts* de varias simulaciones, ejecutan una serie de *scripts* de única simulación tantas veces como se establezca oportuno y finalmente se muestra un resumen de los resultados obtenidos para el número de saltos y el coste de las distintas rutas según el encaminamiento. Además, se muestra un gráfico con la validez de los resultados obtenidos, que no es más que una forma de mostrar un porcentaje de las veces que se ha podido construir una ruta en las simulaciones. Esto sirve para saber si los resultados arrojados por las otras gráficas son útiles. Mencionar que al cargar un *script* de varias simulaciones, el panel de ‘Opciones de simulación’ cambia, quedando así:



Ilustración 42: Opciones de simulación en modo de varias simulaciones

Aquí, en este caso, se selecciona el número de repeticiones de cada simulación. Para ver más detalles, dirigirse al apartado ‘**SCRIPTS DE SIMULACIÓN’.**

Terminando con el panel de ‘Scripts’, mencionar que si se pulsa el botón buscar se abre un explorador de archivos para seleccionar el *script* deseado, y el botón ‘Cargar script’ es el encargado de validarlo y prepararlo para la posterior simulación. Teniendo en cuenta que bajo un script lo que realmente hay es una función de Matlab, el funcionamiento del botón de cargar es navegar hasta el directorio en el que se encuentra el script, crear un puntero hasta este y volver al directorio de ejecución. Finalmente, se carga la función y se actualiza la interfaz consecuentemente. La razón de hacerlo así es para evitar añadir directorios al *path* de Matlab, de forma que el simulador altere lo menos posible al entorno de ejecución.

El último panel del que dispone la interfaz es ‘Editar nodos’. A continuación una captura de este:

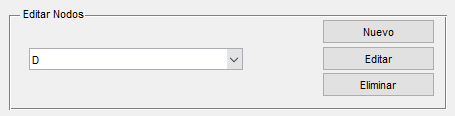


Ilustración 43: Panel de edición de nodos

Aquí, podemos ver una lista desplegable con los nodos que hay en la presente simulación. Así, se permite añadir nuevos nodos, editar el seleccionado o eliminarlo. Mencionar que este último panel solo se puede utilizar en el caso de que la simulación sea de tipo única, ya que para varias simulaciones no se tienen en cuenta estos cambios.

Finalmente, mencionar que en la parte inferior derecha de la interfaz se encuentran los botones para iniciar simulación y para cerrar la interfaz. En la esquina inferior izquierda, se muestra un cuadro de texto que indica el estado actual del simulador: *Ready* cuando está preparado para simular, y *Simulando* en caso de haber una simulación en proceso.

1. SCRIPTS DE SIMULACIÓN

Los *scripts*, como se menciona anteriormente, son los encargados de cargar una simulación. En esta sección se explica el funcionamiento de estos.

En primer lugar mencionar que existen 2 tipos de *scripts*: de simulación única y de varias simulaciones. En ambos casos, la forma de definirlos es crear un archivo con extensión ‘.m’ que contenga una función de Matlab. Dependiendo de los datos que devuelva la función, la interfaz **MainUI** determinará de qué tipo de *script* se trata.

1. SIMULACIÓN ÚNICA

En este caso, la función que se crea devuelve una serie de parámetros. A continuación, se presenta la captura de un ejemplo de este tipo de *script*:

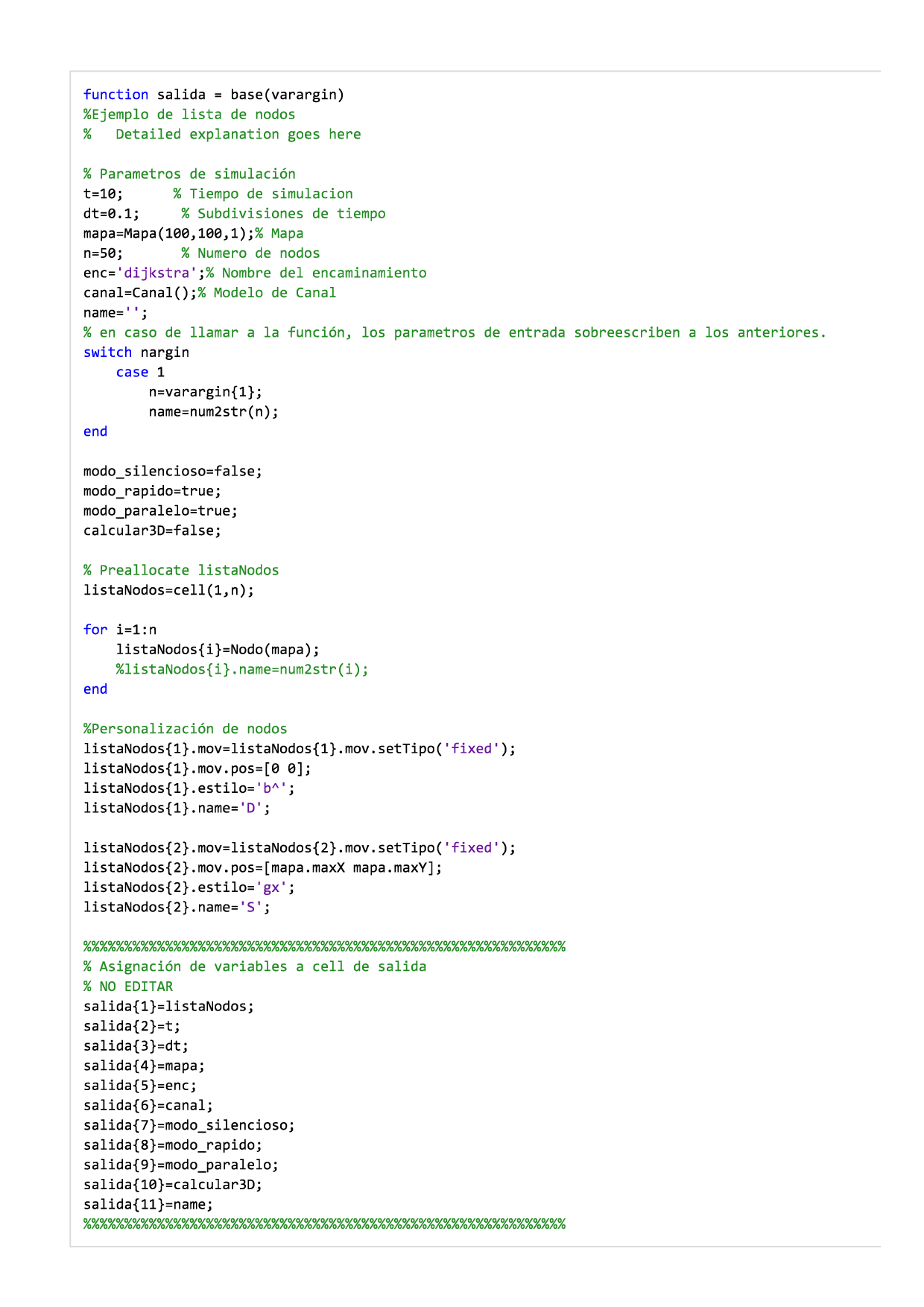


Ilustración 44: Script de simulación única

Como se puede apreciar en la imagen, en primer lugar se generan los parámetros globales de la simulación. Luego, se genera una lista de nodos, y finalmente se editan los nodos que se desee. Finalmente, todo esto se guarda en un *cell* llamado salida, formateado para que el programa principal pueda parsear los datos de forma correcta. Por ello, es aconsejable no editar la última parte del código.

Hay que mencionar que este script está preparado para funcionar de 2 maneras. La primera, es ser llamado desde **MainUI** sin pasarle ningún tipo de parámetro, con lo cual, se toma como una simulación única. Por otro lado, está preparado para ser llamado con parámetros, algo que es muy útil a la hora de hacer varias simulaciones como se verá en el siguiente apartado.

Cabe destacar que una vez cargado de forma individual, se puede utilizar la interfaz del simulador para editar algunos parámetros de simulación de la manera en que se desee.

1. VARIAS SIMULACIONES

En este caso, la función en lugar de devolver un cell con objetos, devuelve un cell con texto. En este caso, el simulador guarda este cell, y posteriormente carga cada elemento de tipo *String*. Cada *String* representa el nombre de la función que hay que llamar con sus correspondientes parámetros. A continuación, un ejemplo que además utiliza el *script* anterior:

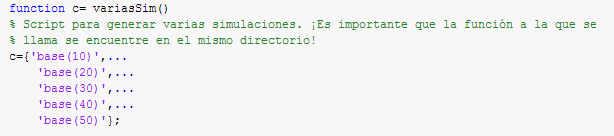


Ilustración 45: Script de varias simulaciones

Al cargar este script e iniciar la simulación, en primer lugar se cargará el script *base(10)*, luego *base(20)* y así sucesivamente.

1. EL BUCLE DE EJECUCION

El bucle de ejecución es el núcleo en sí del programa, ya que es aquí donde se hace uso de la mayoría de las clases implementadas, y se realizan los cálculos necesarios para obtener los resultados pertinentes a las simulaciones.

En primer lugar, se detecta el tipo de simulación, es decir, si es única o si son varias. El bucle de simulación está anidado dentro de 2 bucles de tipo *for*, uno que indica el índice de simulación y otro el índice de repetición. De esta forma, el primer bucle tiene tantas iteraciones como simulaciones vayan a hacerse, o una iteración en el caso de simulación única. El segundo bucle tiene tantas iteraciones como indique el número de repeticiones, o una para el caso de simulación única.

En cuanto al bucle principal de ejecución, será explicado detalladamente a continuación según el tipo de procesamiento que se haga. Tras la finalización de todos los cálculos, se procede a mostrar los resultados. En caso de ser una simulación única, se abre una instancia de *figuraRecepcion*. En caso de ser varias simulaciones, se muestran 2 gráficas, en las que se muestran los resultados medios de cada simulación representados en cuanto a número de saltos y coste de las distintas rutas dependiendo de los algoritmos utilizados. Para realizar estos cálculos, tras cada repetición se guarda la media de coste y número de saltos en un vector, y cuando finaliza el bucle de repeticiones, se hace la media a estos vectores y se almacena en un vector de resultados de simulación. Finalmente, solo es necesario representarlos.

A continuación se detalla el funcionamiento de los algoritmos que implementan el bucle de ejecución en función del número de *cores* que utilizan.

1. SIMULACION SINGLE-THREAD

Esta simulación se realiza con un solo núcleo del procesador, y sin utilizar características especiales de Matlab como el *Parallel toolbox.*

En primer lugar, antes de entrar en los bucles se adquieren los datos de la interfaz gráfica, ya que, si se ha cargado un *script*, estos datos han podido ser cambiados después mediante los distintos menús de los que dispone **MainUI**. En el caso de ser varias simulaciones, se llama a la función correspondiente para cargar los datos. Acto seguido, se inicializan una serie de matrices y cells que serán las encargadas de almacenar los distintos datos, como pueden ser posiciones, consumo, potencia de recepción, rutas, costes, etc. Tras esta inicialización, se entra dentro del bucle principal y comienzan los cálculos.

El bucle principal, de tipo *for*, pasa por los distintos valores de t. El número total de iteraciones es . Los cálculos que se realizan en cada iteración se detallan a continuación.

* Se comprueba si se ha pulsado el botón detener. De esta forma, se para la simulación y se pueden ver los resultados obtenidos hasta el momento.
* Se calcula el porcentaje de simulación.
* Cálculo de matriz de distancias
  + En modo rápido, se calcula la distancia que hay entra cada nodo y el nodo de destino. Si esta distancia es menor que la distancia entre origen y destino, el nodo cuenta como válido. Después, en un bucle *for*, se calculan las distancias entre los nodos válidos. Esto se hace así debido a que son los únicos susceptibles de entrar en una ruta. El resto de distancias se marcan como infinito.
  + Si no está activado el modo rápido, se calculan todas las distancias entre pares de nodos
  + Mencionar que, dado que la distancia A-B es igual que B-A, se hace lo siguiente: La matriz de distancias se inicializa con ceros. Se calcula la matriz triangular superior (sin contar la diagonal, ya que esta es cero). Finalmente, a esta matriz se le suma su traspuesta, de forma que queda una matriz simétrica. Con este algoritmo, se ahorran muchos cálculos pesados.
* Calculo de enlaces
  + En primer lugar, se guarda la posición actual de cada nodo en una matriz de posiciones.
  + Seguidamente, se calculan los enlaces: potencia de transmisión, pérdidas de canal y potencia de recepción. Además, se calcula la cobertura.
  + Finalmente, se mueven los nodos.
  + Los cálculos se hacen para *uplink* y *downlink*, es decir, bidireccionales. En el modo rápido, se evitan los cálculos de los nodos donde la distancia entre estos sea infinita.
* Si está activado el modo 3D, se generan los cálculos del canal para el mapa.
* Se calculan las rutas y sus correspondientes costes.
* Si el modo silencioso está desactivado, se dibujan los gráficos de potencia de recepción y mapa en la interfaz **figuraRecepcion.** En caso contrario, no se dibuja nada, ya que esta interfaz no estaría creada.

Mencionar que los datos que se calculan se guardan en su correspondiente matriz, dado el índice de la iteración.

1. SIMULACION PARALELA

En este tipo de simulación se hace uso del *Parallel toolbox* de Matlab, con lo cual es necesario disponer de este. Se aconseja ver el **apéndice B**, en el cual se explican las características de la computación paralela en Matlab. La forma en la que se realizan los cálculos es distinta al modo *single-thread,* de modo que a continuación se detallan los pasos que se siguen.

Se parte de la misma base que en la simulación single-thread, es decir, con los datos de simulación cargados y las matrices de cálculos con su correspondiente memoria reservada.

* En primer lugar, se calcula el movimiento (posición y velocidad) de cada nodo para toda la simulación. Esto se hace así ya que el movimiento depende de la posición en el instante anterior, por lo que no se puede calcular dentro de un bucle *parfor*.
* Acto seguido, se calcula la matriz de distancias para todo *t*. El algoritmo es similar al del modo *single-thread*, solo que en este caso el modo rápido es forzoso.
* Después, se calculan las pérdidas que va a sufrir cada transmisión. Para ello, se aprovecha que se dispone de la matriz de distancias completa, con lo que se pueden hacer llamadas a las funciones que calculan las pérdidas de canal de forma matricial para así ahorrar mucho tiempo computacional.
* Finalmente, se aprovechan todos los datos anteriormente calculados para calcular la potencia de recepción, consumo, cobertura, rutas y costes.

La forma en la que se hacen estos cálculos es mucho más eficiente ya que aprovecha la potencia de Matlab para realizar cálculos sobre matrices. Por supuesto se da que el hecho de utilizar más núcleos del procesador agiliza los cálculos, ya que se puede aprovechar hasta el 100% de la capacidad de procesamiento.

Por otro lado, hay que mencionar algunos inconvenientes.

* No se puede detener una simulación, ya que los cálculos hacen de manera desordenada. Si es posible utilizar las teclas **‘ctrl+c’** para cancelarla, pero no se obtendría ningún resultado de simulación.
* No hay manera de seguir el porcentaje de simulación. Esto se debe a que no es posible realizar comunicaciones entre los *workers* y el entorno principal, de manera que se pueda tener un *feedback* del estado actual. Hay que mencionar que existen algunas soluciones caseras que han publicado algunos usuarios, pero dada su escasa calidad no se utilizan en el simulador.
* No se pueden mostrar resultados en tiempo real, o, lo que es lo mismo, el **modo silencio** es obligatorio. Hay que esperar a que termine la simulación para ver los resultados.

1. OTRAS FUNCIONES IMPLEMENTADAS

Además de lo mencionado anteriormente, se implementan un par de funciones de manera adicional, a saber: *dB()* y *lin().dB()* se encarga de pasar a decibelios lo que se le dé como entrada y *lin()* hace lo contrario, que es pasar de decibelios a unidades lineales.

En el caso de *dB()*, Matlab dispone de una función similar que realiza el mismo cometido, si bien es cierto que su rendimiento es bastante menor debido a que permite indicar si las unidades de entrada son vatios, voltios, etc. En el caso del simulador, todas las conversiones que se realizan son del tipo:

Luego es preferible implementar la función de forma manual, ya que mejora el rendimiento final del simulador, y la creación de dicha función no supone un gran esfuerzo ya que es un cálculo sencillo.

1. DISEÑO DE LA EXPERIMENTACION

Una vez terminado el simulador, se realizan dos experimentos sobre este. El primero, una simulación para comprobar el correcto funcionamiento del simulador para analizar un entorno concreto. La segunda, un conjunto de simulaciones, de manera que se compruebe el correcto funcionamiento del modo de varias simulaciones. Adicionalmente, sobre estas simulaciones se extraerán distintas conclusiones para satisfacer el segundo objetivo del presente proyecto, que no es otro que demostrar la eficiencia energética de la redes colaborativas con respecto a las redes clásicas.

A continuación, se muestran los detalles de las simulaciones:

1. EXPERIMENTO 1. SIMULACION SOBRE UN ENTORNO

Las características de esta simulación son las siguientes:

* Nodos
  + Número de nodos: 50
  + Una antena omnidireccional de ganancia 2dBi
  + Potencia de transmisión en el rango: -200dBm – 0dBm
  + Sensibilidad: -100dBm
  + Consumo eléctrico: -9dBm
  + Frecuencia de trabajo: 2.4GHz
* Mapa: 100x100m
* Canal
  + *Pathloss*: Modelo simplificado, con y
  + *Shadowing*: Distribución log-normal, ,
  + *Multipath*: Rayleigh

La distribución de los nodos será aleatoria por todo el mapa, salvo el nodo de origen que se situará en la esquina superior izquierda del mapa y el nodo de destino que se situará en el centro. Los nodos se moverán aleatoriamente a 2m/s salvo origen y destino que tendrán posiciones estáticas.

En cuanto al tiempo de simulación, será de 10 segundos, realizando cálculos cada 0.1 segundos.

Esta es una simulación genérica, donde el principal objetivo es ver que los cálculos se realizan son coherentes, de manera que pueda concluirse que el simulador funciona de manera correcta. De paso, se aprovecha para analizar los resultados obtenidos.

1. EXPERIMENTO 2. INFLUENCIA DEL NUMERO DE NODOS

Las características de esta simulación son las siguientes:

* Nodos
  + Número de nodos: variable
  + Una antena omnidireccional de ganancia 2dBi
  + Potencia de transmisión en el rango: -200dBm – 0dBm
  + Sensibilidad: -100dBm
  + Consumo eléctrico: -30dBm
  + Frecuencia de trabajo: 2.4GHz
* Mapa: 100x100m
* Canal
  + *Pathloss*: Modelo de espacio libre.
  + *Shadowing*: Distribución log-normal, ,
  + *Multipath*: Rayleigh

La distribución de los nodos será aleatoria por todo el mapa, salvo el nodo de origen que se situará en la esquina superior izquierda del mapa y el nodo de destino que se situará en el centro. Los nodos se moverán aleatoriamente a 2m/s salvo origen y destino que tendrán posiciones estáticas.

En cuanto al tiempo de simulación, será de 20 segundos, realizando cálculos cada 0.1 segundos.

Se realizarán simulaciones para los siguientes números de nodos: 10, 20, 30, 40, 50, 75, 100, 150, 200. Cada una de las simulaciones se realizará 100 veces, haciendo así que los nodos distintos a origen y destino cambien su posición inicial.

Con esta simulación, se quiere observar cómo influye el número de nodos que forman la red en el rendimiento final de esta. A priori, se espera que el consumo de potencia de los principales algoritmos de encaminamiento decaiga, ya que, al haber más nodos, la posibilidad de que estos estén localizados en una posición óptima para retransmitir aumenta.

En la práctica, si nos centramos en una red de telefonía móvil, los escenarios con pocos móviles representarían las zonas poco pobladas o rurales, mientras que los escenarios con alto número de nodos representarían las zonas urbanas, donde la población es mayor, y por ende, es mayor la cantidad de dispositivos por metro cuadrado.

1. EXPERIMENTO 3: INFLUENCIA DEL CANAL

Las características de esta simulación son las siguientes:

* Nodos
  + Número de nodos: 50
  + Una antena omnidireccional de ganancia 2dBi
  + Potencia de transmisión en el rango: -200dBm – 0dBm
  + Sensibilidad: -100dBm
  + Consumo eléctrico: -9dBm
  + Frecuencia de trabajo: 2.4GHz
* Mapa: 100x100m
* Canal
  + *Pathloss*: Modelo simplificado, con y variable
  + *Shadowing*: Distribución log-normal, ,
  + *Multipath*: Rayleigh

La distribución de los nodos será aleatoria por todo el mapa, salvo el nodo de origen que se situará en la esquina superior izquierda del mapa y el nodo de destino que se situará en el centro. Los nodos se moverán aleatoriamente a 2m/s salvo origen y destino que tendrán posiciones estáticas.

En cuanto al tiempo de simulación, será de 10 segundos, realizando cálculos cada 0.1 segundos.

Se realizarán simulaciones para los siguientes valores de : 2, 2.4, 2.8, 3, 3.3, 3.6, 4. Cada una de las simulaciones se realizará 50 veces, haciendo así que los nodos distintos a origen y destino cambien su posición inicial.

El principal objetivo de esta simulación es ver la repercusión que tiene el canal en el rendimiento de la red. Para ello, se alteran los valores del exponente de pérdidas del modelo simplificado, ya que su valor es muy influyente en las pérdidas totales. Otras opciones podrían ser cambiar los modelos de *shadowing*, *multipath*, además de hacer cambios en sus parámetros. Se decide cambiar ya que su repercusión es más significativa.

1. EXPERIMENTO 4: INFLUENCIA DE LA POTENCIA ELÉCTRICA

Las características de esta simulación son las siguientes:

* Nodos
  + Número de nodos: 50
  + Una antena omnidireccional de ganancia 2dBi
  + Potencia de transmisión en el rango: -200dBm – 0dBm
  + Sensibilidad: -100dBm
  + Consumo eléctrico: variable
  + Frecuencia de trabajo: 2.4GHz
* Mapa: 100x100m
* Canal
  + *Pathloss*: Modelo simplificado, con y
  + *Shadowing*: Distribución log-normal, ,
  + *Multipath*: Rayleigh

La distribución de los nodos será aleatoria por todo el mapa, salvo el nodo de origen que se situará en la esquina superior izquierda del mapa y el nodo de destino que se situará en el centro. Los nodos se moverán aleatoriamente a 2m/s salvo origen y destino que tendrán posiciones estáticas.

En cuanto al tiempo de simulación, será de 10 segundos, realizando cálculos cada 0.1 segundos. El modelo de canal tomado garantiza que haya cobertura siempre entre S y D

En este caso, se varía el consumo eléctrico de los nodos, tomando este los siguientes valores en dBm: -30, -25, -20, -15, -10, -5, 0. Se realizarán 50 repeticiones de cada escenario, por las mismas razones que en los anteriores experimentos.

El interés en esta simulación es comprobar que realmente se cumple lo que se comenta en el **apartado D** del marco teórico. Con lo cual, se espera que conforme aumente el consumo eléctrico de los nodos, el número de saltos disminuya. Además, es interesante ver qué sucede con el consumo global de las rutas.

1. ANALISIS DE RESULTADOS

A continuación, se muestran los resultados obtenidos tras la simulación de los anteriores experimentos, con su consecuente análisis.

1. EXPERIMENTO 1. SIMULACION SOBRE UN ENTORNO

En primer lugar, se presenta el estado inicial del mapa con los correspondientes nodos:

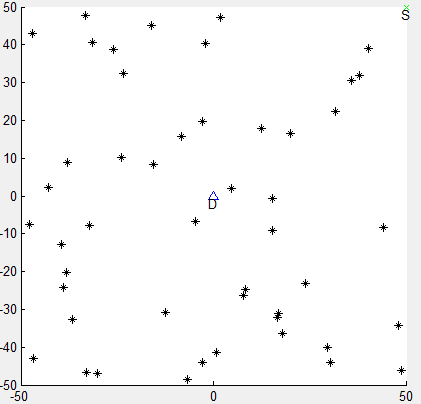


Ilustración 46: Mapa de la simulación

Como se puede apreciar, el nodo de origen se encuentra en la esquina superior derecha del mapa (S), mientras que el nodo de destino se encuentra en el centro (D).

En primer lugar, se analizará el enlace directo. A continuación, se puede ver una gráfica con la potencia que recibe D de S.

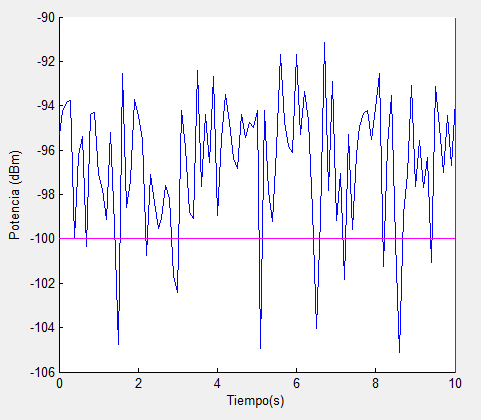


Ilustración 47: Potencia de recepción en D

Se puede apreciar que la mayor parte del tiempo, la potencia que recibe es mayor que su sensibilidad (100dBm), con lo cual, se encuentra en cobertura. A continuación, se selecciona un punto en el que no hay cobertura y se muestra una captura de la interfaz:

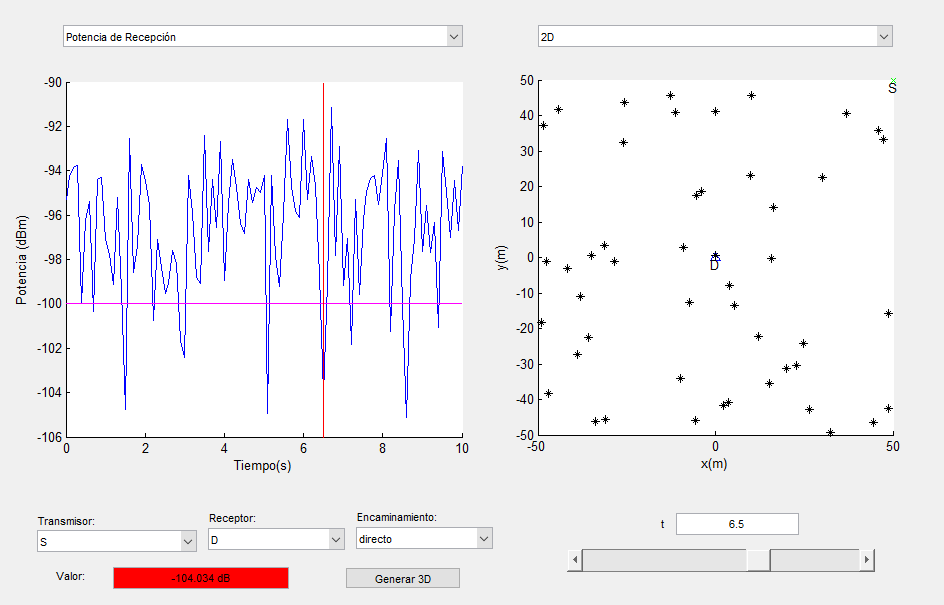


Ilustración 48: Captura de la interfaz en un momento en el que no hay cobertura

En ese instante, la lectura del valor aparece en color rojo ya que la potencia de recepción es menor a 100dBm, y además, en el mapa no aparece la ruta pintada. Lo siguiente a analizar es la potencia de transmisión:

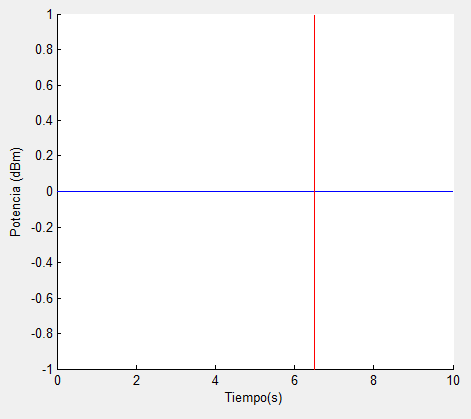


Ilustración 49: Potencia de transmisión de S a D

Como se puede apreciar, esta es constante, lo cual tiene sentido ya que la distancia entre los nodos no varía. Además, está fija en 0 dBm, que es la potencia máxima de transmisión. Visto esto, lo siguiente a observar es el mapa de cobertura que se genera, tanto en 3D como en 2D, para hacernos una idea del comportamiento del canal.

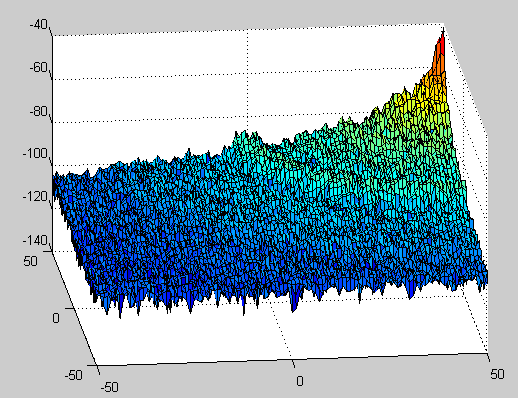


Ilustración 50: Mapa en 3D de la recepción de señal en cada punto

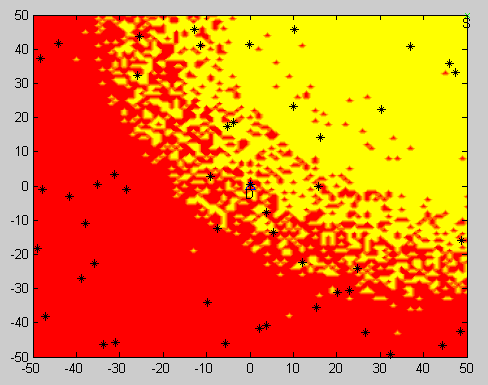


Ilustración 51: Mapa de cobertura de S

A la luz de lo que muestran estos mapas, se da una explicación de por qué la potencia de transmisión es siempre máxima, y es que el nodo D se encuentra en una zona donde la cobertura es mala. Esto explica también la existencia de instantes en los que la señal recibida fuera menor que la sensibilidad.

A continuación se muestra la potencia de transmisión con respecto al tiempo para un nodo cualquiera que se encuentra en una buena zona de cobertura:

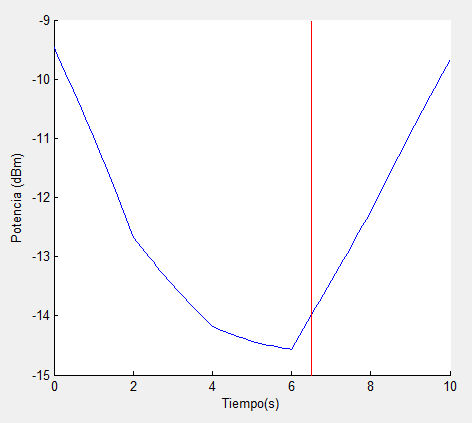


Ilustración 52: Potencia de transmisión de S hacia un nodo cualquiera en zona de cobertura

Como se puede ver, la potencia de transmisión varía. Esto se debe a que el nodo receptor en este caso se mueve aleatoriamente, de lo cual se puede deducir que en un momento se acerca a S y más adelante, a partir del segundo 6 de simulación, comienza a alejarse.

También es interesante ver el consumo S cuando la ruta es directa, y cuando se establece otra ruta. A continuación, se muestran los gráficos para encaminamiento directo y B\_Below

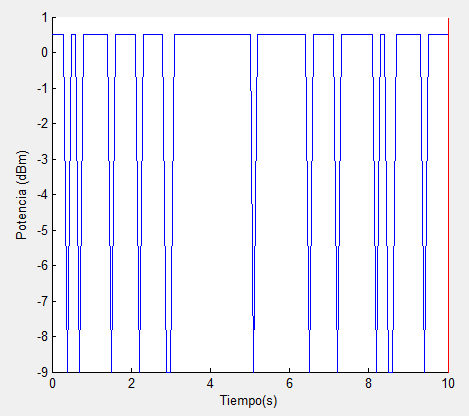


Ilustración 53: Consumo para encaminamiento directo

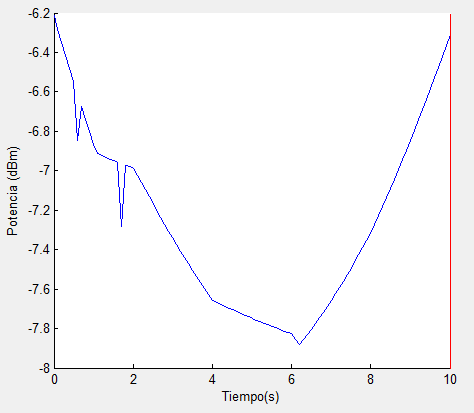


Ilustración 54: Consumo para encaminamiento B\_Below

A raíz de lo que se puede ver en los anteriores gráficos, se puede decir que el consumo medio de S es mejor cuando se utiliza encaminamiento B\_Below. De estos gráficos, hay algo que destacar los picos que se ven en el encaminamiento directo. Estos picos aparecen justo en los momentos en lo que el nodo de recepción. Se debe a la forma en la que el simulador hace los cálculos, y es que en caso de saber que la ruta no se puede alcanzar, se toma potencia de transmisión 0. Esto en la realidad no es así: en la realidad, el nodo consumiría potencia por transmitir, a pesar de que en ese momento no consiguiese comunicación. La razón de que esto se implemente así es que en caso de ser una red colaborativa, es imposible saber cuál sería el siguiente nodo, y, por ende, con que potencia transmitir. Por eso, el simulador asume que el nodo hace lo óptimo en cada momento. A la hora de extraer resultados, se hace la media, así que estos instantes puntuales no tienen tanto peso.

Una vez vistos los resultados de transmisión y recepción, se pasa a analizar las rutas que generan los distintos algoritmos y comparar los resultados. Para ello, se añaden las capturas que representan el número de saltos y el coste de las rutas.

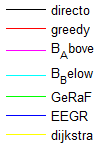


Ilustración 55: Leyenda

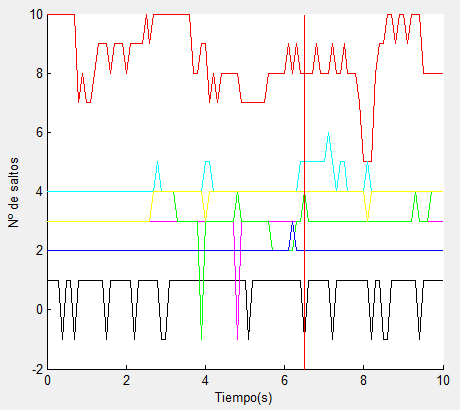


Ilustración 56: Número de saltos según el algoritmo de encaminamiento utilizado

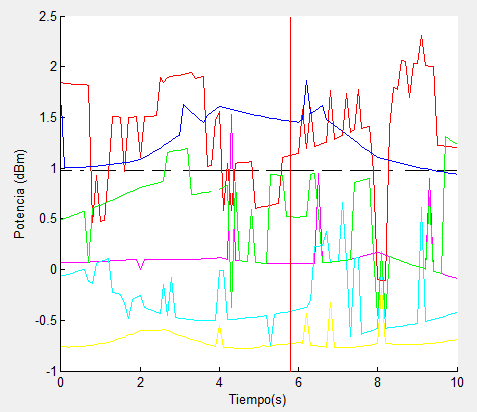


Ilustración 57: Coste según el algoritmo de encaminamiento utilizado

A raíz de los resultados obtenidos, se puede ver como para este entorno hay algoritmos de encaminamiento que actúan mejor o peor que el camino directo. Por ejemplo, B\_*Below* tiene el coste más bajo (el mejor por delante de *Dijkstra*, que, como ya se ha comentado, es el límite teórico) mientras que *EEGR* rinde peor que el camino directo. Por otro lado, *Greedy* también tiene un mal rendimiento. A continuación, se comparan estos algoritmos mencionados anteriormente y se explican los motivos. Para ello, se muestran capturas de las rutas para un instante dado:

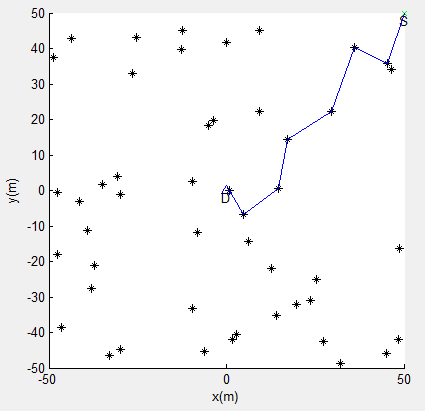


Ilustración 58: Ruta Greedy

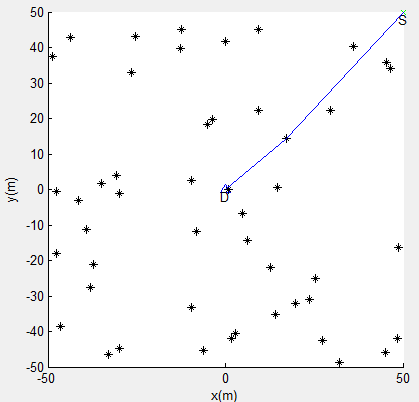


Ilustración 59: Ruta EEGR

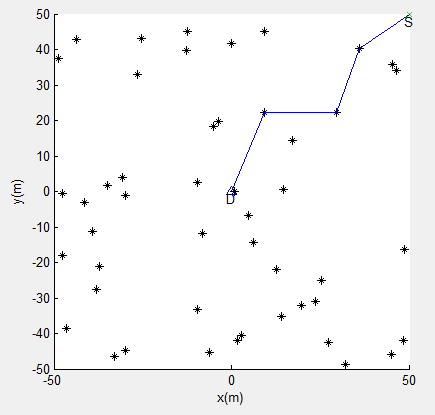


Ilustración 60: Ruta B\_Below

En primer lugar podemos ver *Greedy*. Como se puede ver, se hacen muchos saltos de corta distancia. Por ser distancias cortas, la potencia de transmisión es pequeña. El problema de este caso es que se suma el consumo eléctrico de un gran número de nodos, lo que hace el coste de la ruta alto.

En cuanto a *EEGR*, pasa lo contrario. Y es que, solo hay 2 saltos, pero estos saltos son muy grandes. Con lo cual, la suma del coste de los 2 saltos y la potencia eléctrica del nodo intermedio da un coste mayor que el encaminamiento directo.

Por último, está el caso de B\_*Below*. En este caso, los saltos son de media distancia, y no hay tantos saltos como en *Greedy.* Con lo cual, se obtiene un consumo global de potencia menor.

Para terminar, se añade una captura de la ruta de Dijkstra:

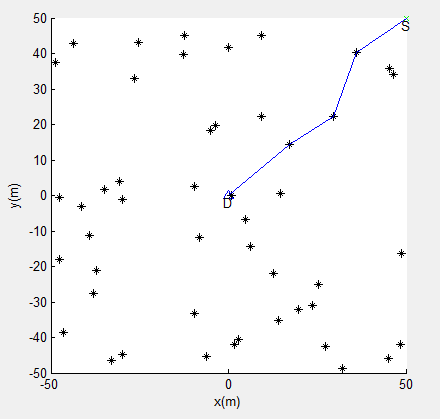


Ilustración 61: Ruta Dijkstra

Como se puede apreciar, la ruta es muy similar a la de *B\_Below*. Pero en este caso, se toman unos saltos finales más cortos, lo cual hace la ruta más eficiente. Esta captura sirve para demostrar que aquella ruta que se parezca más a la de *Dijkstra* tendrá un consumo menor, ya que esta ruta es la más eficiente.

1. EXPERIMENTO 2. INFLUENCIA DEL NUMERO DE NODOS

En esta simulación, hay que destacar varias diferencias con respecto al anterior escenario. En primer lugar, que el canal es menos agresivo, ya que el *pathloss* es de tipo espacio libre mientras que en el experimento 1 se utilizaba uno de modelo simplificado con exponente 3. Además, el consumo eléctrico de cada nodo es bastante inferior. Así pues, dado que el mapa es del mismo tamaño, se puede esperar que las áreas de cobertura sean más grandes y que el número de saltos que tomen los algoritmos sea mayor, debido al escaso consumo eléctrico de los nodos (o, lo que es lo mismo, en este escenario prima más el coste de transmisión que el consumo eléctrico). A continuación, se muestran los resultados obtenidos tras la simulación:

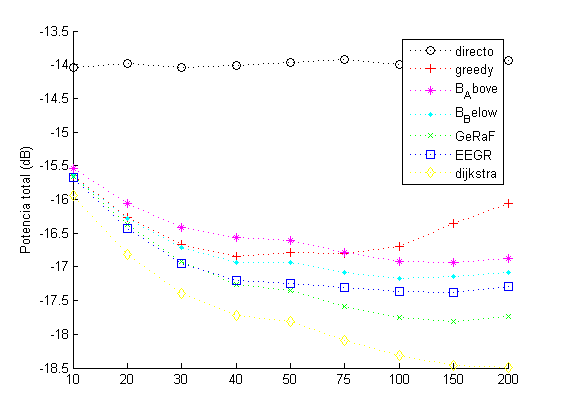


Ilustración 62: Evolución del consumo energético con respecto al número de nodos

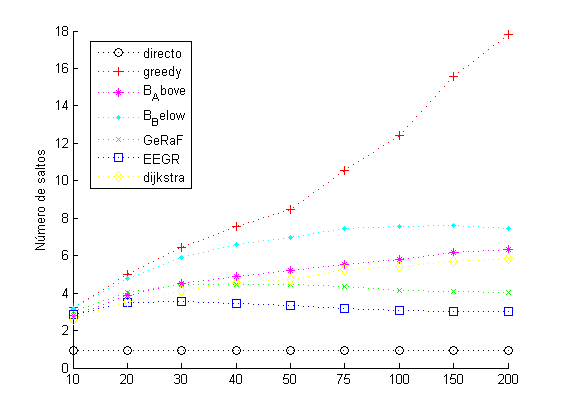


Ilustración 63: Evolución del número de saltos con respecto al número de nodos

Mencionar que la validez de los datos supera el 97% para todas las medidas, con lo cual no es necesario mostrar dicha gráfica y se dan por buenos los resultados obtenidos.

A la vista de los resultados, el primer dato reseñable es que el consumo del encaminamiento directo no varía, lo cual era de esperar ya que en este caso no influye el número de nodos que haya alrededor. Por otro lado, en términos de potencia, todos los algoritmos aportan mejores resultados que el encaminamiento directo. A continuación, se analizan los resultados del resto de algoritmos.

En primer lugar, *greedy*. Como se puede ver, conforme aumenta el número de nodos, este algoritmo aumenta el número de saltos, lo cual tiene sentido ya que en este algoritmos se toma como siguiente nodo al más cercano. Al haber más nodos, aparecen más nodos entre origen y destino, de ahí el crecimiento del tamaño de la ruta. Por otro lado, se puede ver como el coste comienza a ser mejor con el paso del tiempo, pero llega un momento en el que este comienza a empeorar. Esto es debido a que a partir de un número de saltos, la potencia eléctrica de los nodos implicados en la ruta empieza a contribuir negativamente en el consumo global de la misma.

Por otro lado, en cuanto al resto de algoritmos (sin contar *Dijkstra*), todos tienen una tendencia a mejorar su coste conforme aumenta el número de nodos. Esto se explica debido a que al haber más nodos, hay más posibilidades de que estos se encuentren cercanos a los puntos óptimos que establecen los algoritmos. Inicialmente, el rendimiento es malo ya que apenas hay nodos con los que establecer rutas. En lo que al coste se refiere, parece haber una tendencia a estabilizarse a partir de 100 nodos. Por otro lado, en cuanto al número de saltos, a partir de 50 saltos tiende a estabilizarse, salvo en el caso de *B\_Above* que parece tener una tendencia a crecer lentamente. Mencionar también que, si bien en la anterior experimentación EEGR tenía un resultado peor que el encaminamiento directo, en este caso obtiene el segundo mejor consumo. Con lo cual, se concluye que es necesario conocer las características del canal y de los nodos para saber que algoritmo de encaminamiento es el más adecuado.

Finalmente, mencionar los resultados de *Dijkstra*, que, como siempre, son los mejores en cuanto a consumo. La tendencia en este caso es a distanciarse más del resto de los algoritmos conforme se aumenta el número de nodos, lo cual tiene sentido ya que *Dijkstra* calcula todas las posibilidades y selecciona la mejor.

1. EXPERIMENTO 3: INFLUENCIA DEL CANAL

Como ya se ha mencionado anteriormente, en esta simulación el parámetro que varía es la agresividad del canal, que no viene a ser otra cosa que el factor de pérdidas de propagación. A continuación, se muestran los resultados arrojados por la simulación:

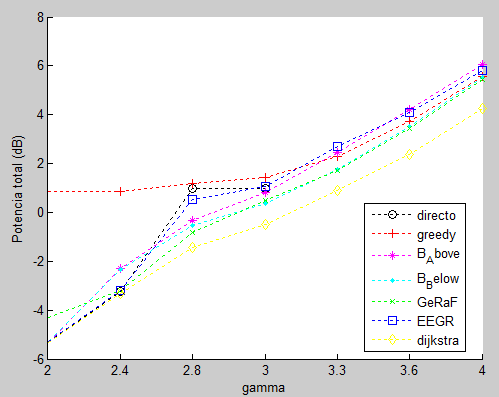


Ilustración 64: Evolución del consumo de las rutas con respecto al valor del exponente de pérdidas de propagación

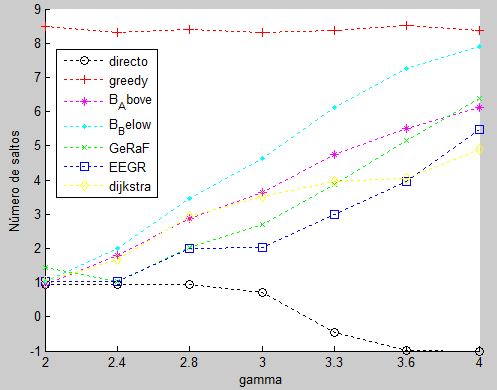


Ilustración 65: Evolución del número de saltos con respecto al valor del exponente de pérdidas de propagación

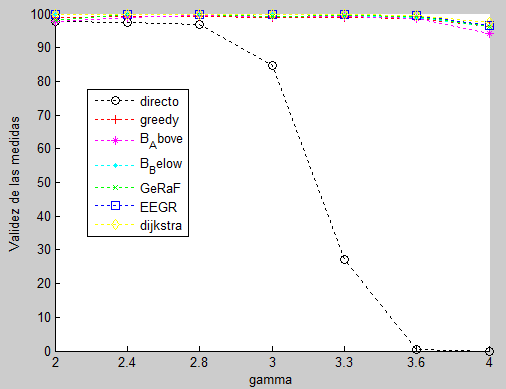


Ilustración 66: Validez de los resultados del experimento 3

En cuanto al consumo de potencia en las rutas de los principales algoritmos de encaminamiento, se observa una clara tendencia a aumentar. Esto era de esperar, ya que al haber más perdidas de propagación, los nodos se ven obligados a transmitir con más potencia.

Al fijarse en el número de saltos, también tiende a aumentar, salvo en dos casos (*greedy* y directo) que serán explicados más adelante. Echando un vistazo a la teoría, se recuerda que la distancia óptima de transmisión disminuía conforme aumentaba el exponente de pérdidas de propagación. Con lo cual, los nodos buscan realizar transmisiones a distancia más cortas, y se necesitan más saltos para cubrir la distancia que hay hasta el destino. Es por esto que cuando el valor de es 2, los algoritmos tienden a transmitir de forma directa al destino, y, conforme aumenta este valor, se utilizan más saltos.

Por otro lado, está el algoritmo *greedy*, que no depende de la distancia óptima. Como se puede observar, su número de saltos es constante (entre 8 y 9, ya que depende de la localización de los nodos en el escenario), ya que el algoritmo coge como siguiente nodo al más cercano. Es por esto que para valores de bajos, el consumo de potencia es mucho mayor con respecto al resto de rutas, ya que lo óptimo en este caso transmitir de forma directa al destino. Por el contrario, conforme aumenta , el consumo comienza a equipararse con el del resto de rutas, llegando al punto de incluso obtener mejores resultados que otras. Esto se debe a que en esos casos, la potencia de transmisión empieza a pesar mucho en transmisiones de largas distancias, con lo que prima más hacer más saltos de corta distancia.

Por último, mencionar el caso de la ruta directa. Para valores bajos de , el rendimiento es óptimo. Conforme aumenta su valor, el coste aumenta muy rápidamente, hasta que llega a un punto máximo, que es cuando se llega a la potencia máxima de transmisión. A partir de este momento, lo que sucede es que el radio de cobertura comienza a disminuir, llegando al punto de que los nodos no pueden comunicarse entre sí. Para ello, se puede ver la gráfica de validez de las simulaciones, que muestra el porcentaje de veces que se ha podido establecer la ruta en la simulación.

Como conclusión, mencionar que, conforme peor es el canal, peores resultados se obtienen. Si bien, es cierto que el uso de redes colaborativas permite mantener las comunicaciones que con el uso de una red clásica, se perderían.

1. EXPERIMENTO 4: INFLUENCIA DE LA POTENCIA ELÉCTRICA

En este caso, se realiza un barrido del parámetro *‘Potencia eléctrica’* de los nodos. A continuación, los resultados:

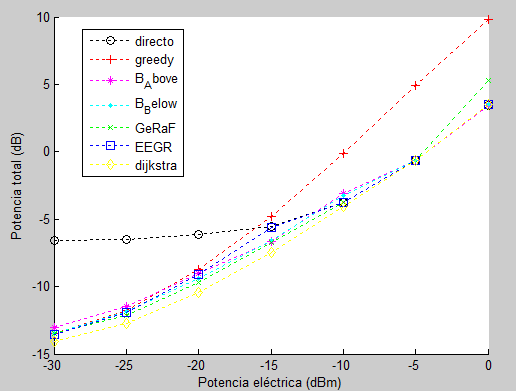


Ilustración 67: Evolución del coste de las rutas con respecto a la potencia eléctrica

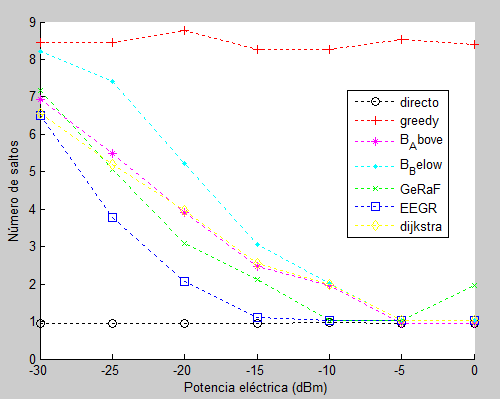


Ilustración 68: Evolución del número de saltos con respecto a la potencia eléctrica

En primer lugar, mencionar que la validez de los resultados supera en todos los casos el 96.5% con lo cual se dan por buenos y no es necesario mostrar la gráfica correspondiente.

En cuanto al consumo de potencia, se puede apreciar que tiende a aumentar conforme aumenta el consumo eléctrico de los nodos, lo cual tiene sentido. Si bien es cierto que hay que destacar 2 casos.

El primero, el caso de la transmisión directa, y es que, con bajo consumo eléctrico, se obtiene un rendimiento peor. Esto es debido a que pesa más la potencia de transmisión en este caso. Es por ello que, conforme aumenta este consumo eléctrico, el consumo de la ruta directa tiende a igualarse con el resto de rutas. De hecho, este consumo es constante hasta que se llega a -15dBm, donde se iguala con el resto de consumos y comienza a aumentar. Se puede deducir que es en este punto en el que la potencia de transmisión se iguala a la potencia eléctrica.

Por otro lado, está *greedy*. Y es que en ese mismo punto, el consumo global de este tipo de ruta aumenta de una forma más pronunciada que el resto. Esto se debe a que *greedy* establece siempre rutas con un alto número de nodos, con lo que si el consumo de estos aumenta, el coste de las rutas se ve perjudicado.

En cuanto al número de saltos, se obtiene el resultado esperado, y es que disminuya hasta ser uno. Esto se debe a que, la distancia óptima de los saltos aumenta conforme aumenta la potencia eléctrica de los nodos, hasta el punto de que es mejor hacer comunicaciones directas entre origen y destino.

A raíz de lo observado en este experimento, se puede concluir que las redes colaborativas dan un rendimiento mayor cuando el consumo eléctrico de los nodos que las forman es menor, y más concretamente cuando la potencia de transmisión es mayor que la potencia eléctrica de los nodos.

1. PLANIFICACION Y COSTES

Para ilustrar la planificación llevada a cabo en el presente proyecto, se adjunta un diagrama de Gantt:

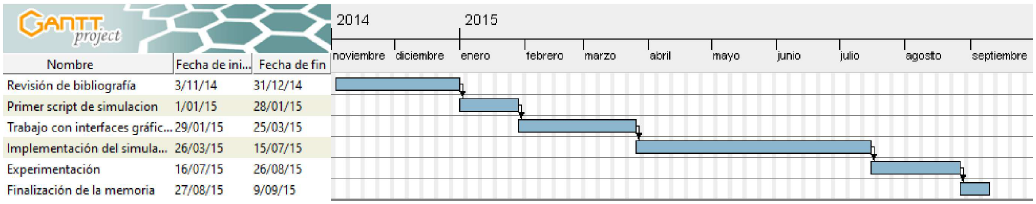


Ilustración 69: Diagrama de Gantt del presente proyecto

A continuación, se detallan las fases de realización del proyecto presentadas en el anterior diagrama:

* **Revisión de bibliografía**: En esta fase, el trabajo se centra en extraer información de las fuentes bibliográficas. Concretamente, en lo concerniente a redes colaborativas, transmisiones inalámbricas y modelado de canal. La fase concluye con la realización de la introducción y el marco teórico del presente documento.
* **Primer *script* de simulación**: Se marca como objetivo la realización de un pequeño *script* como primera aproximación al simulador final.
* **Trabajo con interfaces gráficas en Matlab**: Estudio del funcionamiento de las interfaces gráficas en Matlab. Es un paso previo necesario para poder implementar más tarde la interfaz del simulador.
* **Implementación del simulador**: Aquí el trabajo se centra en cumplir el primer objetivo del proyecto, que es la implementación de un simulador en el cual se puedan realizar experimentos de redes colaborativas. Esta parte implica la escritura del código y su correspondiente documentación
* **Experimentación**: El objetivo que se fija es realizar distintos experimentos, para comprobar que el simulador funciona correctamente y finalmente alcanzar el segundo objetivo del proyecto, que no es otro que extraer conclusiones sobre las redes colaborativas en términos de potencia.
* **Finalización de la memoria**: Por último, se procede a la finalización del presente documento, dando formato a las distintas partes realizadas con anterioridad.

Finalmente, a continuación se dispone una tabla en la cual se resumen los supuestos costes que habría tenido la realización de este proyecto en ámbito profesional:

|  |  |  |
| --- | --- | --- |
| ELEMENTO | COSTE MENSUAL | TOTAL |
| Matlab |  | 2420 |
| Statistics toolbox |  | 1210 |
| Parallel toolbox |  | 1210 |
| Salario | 750 | 8250 |
| Total |  | 13000 |

Tabla 1: Resumen de costes

Hay que asumir que el tiempo total de duración del proyecto es de 11 meses. Por otro lado, las licencias de Matlab y sus correspondientes *toolbox* son de uso comercial, y por tiempo indefinido. Los precios son los existentes a fecha de septiembre de 2015. Además, se ha añadido el Impuesto de Valor Añadido de España (21%) a los precios que proporciona Matlab, ya que estos vienen exentos de impuestos.

1. CONCLUSIONES Y LINEAS FUTURAS

A continuación se muestran las conclusiones que se obtienen del presente proyecto, por un lado del simulador y por otro de los experimentos realizados sobre este. Por último, se añaden una serie de líneas futuras de desarrollo, tomando como base el presente estudio.

1. SIMULADOR

El simulador de redes colaborativas desarrollado en este proyecto es una aplicación desarrollada sobre Matlab. Para hacer uso de esta aplicación es necesario disponer del software Matlab, además del *statistics toolbox* y, opcionalmente, el *parallel toolbox*.

A continuación, se muestra un resumen de las características del simulador:

* Principalmente, permite definir un escenario de red colaborativa para hacer un exhaustivo análisis de la eficiencia energética de esta.
* Personalización de modelo de canal.
* Personalización de características de los nodos, como su consumo eléctrico y sus parámetros de transmisión y recepción de señales. También se pueden definir patrones de radiación, y uso de varias antenas para transmitir.
* Nodos con movilidad.
* Varios algoritmos de encaminamiento geográfico implementados.
* Ejecución de análisis del funcionamiento de una red con respecto al tiempo, viendo en detalle potencias de recepción, transmisión, consumo, y características de las rutas según el algoritmo utilizado.
* Ejecución de conjuntos de simulaciones de escenarios distintos para poder comparar los resultados obtenidos en estas.
* Establece la ruta entre el segundo nodo de la lista y el primero. Si bien es cierto que las funciones que están implementadas permiten cambiar el índice de origen y destino, pero el simulador no permite cambiar este dato.

Por otro lado, el simulador también tiene algunas limitaciones

* Orientado solo al estudio de potencias. No se estudian otros parámetros de la red como retardos, velocidad de transmisión de datos, etc.
* Diseñado para hacer simulaciones en las que los nodos trabajen en altas frecuencias y baja movilidad.

El código del simulador se puede encontrar en el siguiente repositorio:

<https://github.com/kharenzze/Simulador-de-redes-inalambricas>

Adicionalmente, en el **Apéndice A** se encuentra un tutorial de uso del simulador.

1. EXPERIMENTACION

De las simulaciones realizadas, se puede concluir que, por normal general, el uso de redes colaborativas mejora los resultados en cuanto a consumo energético de los nodos en comparación con las redes clásicas, en las que las comunicaciones se hacen directamente entre origen y destino. La principal razón es que el consumo de potencia se reparte entre los nodos implicados, de modo que, al sumar los consumos de cada nodo por separado, se obtiene un consumo menor que en el de una transmisión directa. Por otro lado, si nos fijamos en el consumo energético del nodo de origen, este es el que sale mejor parado, ya que en el caso de las redes clásicas, este nodo es el que carga con el coste total.

Como se ha visto existen varios algoritmos de encaminamiento. Realmente, no se puede decir que uno sea el mejor, ya que generalmente hay una fuerte dependencia del escenario concreto. Con lo cual, hay que estudiar primeramente el entorno para así poder seleccionar aquel algoritmo de encaminamiento que de mejores resultados.

Otra de las conclusiones que se obtiene a raíz de los experimentos realizados es que la eficiencia de las redes colaborativas tiende a mejorar conforme aumenta el número de nodos. La explicación de este fenómeno es que, al haber más nodos, es más probable que estos se encuentren situados en las posiciones óptimas que permiten obtener los mejores resultados.

En cuanto a la influencia del canal, mencionar que cuanto peores son las características de transmisión, peor es el rendimiento obtenido. Si bien es cierto, que en las redes clásicas se llega a un punto en el que la cobertura se pierde, mientras que las redes colaborativas permiten mantener la comunicación en estos casos.

En lo referente a la influencia del consumo eléctrico de los nodos, mencionar que las redes colaborativas dan resultados mejores que las clásicas cuando el consumo eléctrico de los nodos es menor. Si este grande, las rutas que se generan convergen en una comunicación directa, o, lo que es lo mismo, en una red clásica.

Esta última conclusión es la que hace que las redes colaborativas sean muy interesantes para la telefonía móvil, ya que en entornos urbanos, donde la densidad de nodos es muy alta y el canal de propagación es malo, este tipo de redes obtendrían una gran mejora energética con respecto al estado actual. De ahí que su uso esté previsto en los estándares de futuro como 4G o 5G.

1. LINEAS FUTURAS

El presente proyecto sirve como base para seguir desarrollándolo, ya que, como ya se ha dicho anteriormente, solo está centrado en el estudio de la eficiencia energética. Si bien es cierto que existen muchos parámetros que pueden ser estudiados para conocer mejor el funcionamiento de las redes colaborativas.

En lo referente al simulador, se puede seguir desarrollando para obtener los datos anteriormente mencionados. También es conveniente eliminar sus limitaciones. Por otro lado, también se pueden seguir añadiendo otros modelos de canal para enriquecer la diversidad de experimentos posibles, o los algoritmos de encaminamiento. Además de esto, se pueden añadir modelos de nodos predeterminados (en forma de constructores de la clase **Nodo**). Por otro lado, los nodos están preparados para soportar la tecnología MIMO (*Multiple Input, Multiple Output*), con lo que solo falta implementar su funcionamiento.

De hecho, sería interesante añadir una capa de enlace y capa de red a los nodos, añadiendo en esta última distintos algoritmos de encaminamiento como AODV, DSDV, etc. Por último, otra cosa que se puede añadir y que es interesante es un *plugin* de seguridad, para, por ejemplo, establecer nodos maliciosos que hagan malos encaminamientos a propósito, para comprobar la repercusión que tendría en la red global este comportamiento.

Por supuesto, una vez añadidas estas nuevas características al simulador, se abre un mundo de posibilidades a la hora de hacer simulaciones y obtener resultados, ya que se pueden establecer más combinaciones de parámetros, aumentando así la cantidad de posibles escenarios.

Aun sin añadir novedades al simulador, se pueden hacer muchos estudios para extraer conclusiones. Por ejemplo

* Comprobar la influencia del *multipath*
* Comprobar la influencia de la sensibilidad
* Simulaciones con nodos de distintas características, siendo más realistas los resultados obtenidos a la hora de ser aplicados a la práctica. Por ejemplo, en las redes de telefonía móvil, los nodos tienen distintas características.
* Establecer nodos que dispongan de varias antenas, siendo estas de tipo direccional. Con este tipo de nodos, repetir los anteriores experimentos.

Estos son solo unos pocos ejemplos, ya que son muchos los parámetros que pueden cambiarse, y de los cuales se pueden obtener distintos resultados prácticos.

1. REFERENCIAS

|  |  |
| --- | --- |
| [1] | A. S. Molisch, Wireless communications, 2nd edition, Wiley, 2005 (1st edition). |
| [2] | T. S. Rappaport, Wireless communications: Principles and practice, Prentice Hall. |
| [3] | A. Goldsmith, Wireless communications, 2004. |
| [4] | R. M. Rius, «Dieseño cooperativo de un sistema de comunicaciones por satélite basado en diversidad espacial,» Barcelona, 2008. |
| [5] | «Efecto Doppler,» [En línea]. Available: http://laplace.us.es/wiki/index.php/Efecto\_Doppler. [Último acceso: 5 9 2015]. |
| [6] | P. Padilla · J. Camacho · G. Maciá-Fernández ·J. E. Díaz-Verdejo · P. García-Teodoro · C.Gómez-Calero, «On the Influence of the Propagation Channel in the Performance of Energy-Efficient Geographic Routing Algorithms forWireless Sensor Networks (WSN),» Springer, 2012. |
| [7] | Karim Seada, Marco Zuniga, Ahmed Helmy y Bhaskar Krishnamachari, «EnergyEfficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks». |
| [8] | M. Zorzi y Ramesh R. Rao, «Geographic Random Forwarding (GeRaF) for ad hoc and sensor networks: multihop performance». |
| [9] | Mathworks, «Ayuda de Matlab,» Mathworks, [En línea]. Available: http://es.mathworks.com/help/matlab/. |
| [10] | «Improving the Speed of MATLAB Calculations,» [En línea]. Available: http://web.cecs.pdx.edu/~gerry/MATLAB/programming/performance.html. [Último acceso: 5 5 2015]. |
| [11] | F. Pérez Fontán, y P. Mariño Espineira, Modeling the wireless propagation channel: A simulation approach with Matlab, 2008. |
| [12] | Javier Gozalvez y Baldomero Coll-Perales, «Experimental Evaluation of Multihop Cellular Networks Using Mobile Relays,» 2013. |
| [13] | Baldomero Coll Perales, Javier Gozalvez y Joaquin Sanchez-Soriano, «Empirical Performance Models for P2P and Two Hops Multi-hop Cellular Networks with Mobile Relays». |
| [14] | B. Coll-Perales, Javier Gozalvez y Vasilis Friderikos, «Comunicaciones Oportunistas y Contextuales para Redes Multi-hop Celular con Retransmisores Móviles». |

APÉNCIDE A: TURORIAL DE USO DEL SIMULADOR

En este apéndice se muestran los pasos a seguir para realizar la primera simulación:

**Paso 1:** Descarga del código. Para ello, hay que descargar el contenido del siguiente repositorio:

<https://github.com/kharenzze/Simulador-de-redes-inalambricas>

**Paso 2:** Dentro de Matlab, navegar hasta la carpeta ‘codigo’, dentro de la ruta en la que se hayan guardado los archivos descargados anteriormente. Opcionalmente, puede añadirse esta carpeta al *path* de Matlab. Se debe ver lo siguiente:

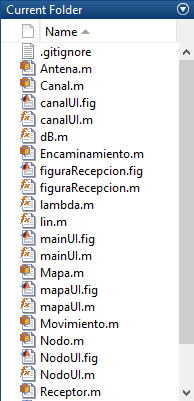


Ilustración 70: Carpeta 'codigo'

**Paso 3:** Ejecutar el programa. Para ello, basta con escribir mainUI en la línea de comandos de Matlab.

**Paso 4:** La interfaz gráfica del simulador debería estar visualizándose. En este passo, se va a preparar la simulación:

* Poner al menos 2 nodos en el mapa. Para ello, hacer click derecho en alguna posición, y pulsar ‘Añadir Nodo Rápido’
* En las opciones de simulación, establecer como algoritmo de encaminamiento ‘Barrido’.

Debería quedar algo parecido a lo siguiente:

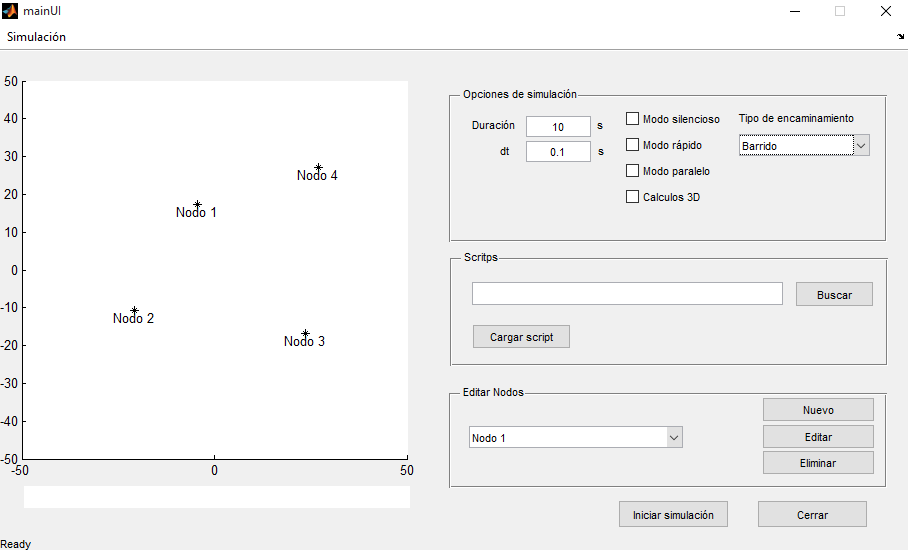


Ilustración 71: Ejemplo de ejecución

Finalmente, pulsar el botón ‘**Iniciar simulación**’

**Paso 5:** En este momento, debería de aparecer la interfaz de resultados, rellenándose en tiempo real. Esperar a que termine para poder acceder a todos los datos de la simulación.

A través de los distintos controles, se pueden ver los todos resultados de la simulación.

Con esto, ya se ha terminado la primera simulación. A continuación, se puede probar a volver a la interfaz principal. Una vez allí, se puede probar a cambiar los distintos parámetros de simulación: editar los nodos actuales, añadir/eliminar nodos, editar el modelo de canal, cambiar el tamaño del mapa, etc. También se puede probar a hacer simulaciones rápidas, o con procesamiento paralelo.

Recordar que la primera vez que se ejecuta una simulación con procesamiento paralelo, tarda en iniciarse debido a que Matlab debe de activar los *workers* (Siempre y cuando no haya sido activado previamente por el usuario)

APÉNDICE B: PROGRAMACIÓN PARALELA EN MATLAB

En este apéndice se habla sobre la programación utilizando varios procesadores en Matlab. Recordar que, para hacer uso de esta característica, es necesario disponer del *Parallel toolbox* de Matlab.

Matlab es un lenguaje de los que se conoce como *single-thread*, o, lo cual quiere decir que se ejecuta sentencia a sentencia de código, y solo hay una hebra de forma simultánea. Esto, a día de hoy, es una gran limitación si se quiere construir un código que obtenga un buen rendimiento, ya que los típicos procesadores de los que se dispone son de varios núcleos.

Es por ello que Matlab crea el *Parallel toolbox.* Cuando se activa esta herramienta, que previamente debe ser configurada o de lo contrario tomara valores por defecto) se abren unas instancias de Matlab llamadas *workers*. A efectos prácticos, son como otras ejecuciones del entorno principal, con *workspace* propio. Por tanto, también son *single-thread*, de manera que no se pierda la esencia del lenguaje.

Una vez creados los *workers*, la interfaz principal de Matlab puede delegarles ciertos cálculos, para lo cual se necesita primeramente copiar las variables al *workspace* del *worker* correspondiente.

Llegado a este punto, se pasa a explicar cómo se puede utilizar todo esto a la hora de desarrollar código. Para ello, se procede a explicar cómo funcionan los bucles paralelos, llamados *parfor:*

Un bucle *parfor* es similar a un clásico bucle *for*, pero, en este caso el bucle se subdivide y se reparte entre los *workers* disponibles. Por ejemplo, para i=1:20 con 2 *workers*, un *worker* haría las iteraciones i=1:10 y el otro las i=11:20. Con esto, se aprovecha al máximo el rendimiento del procesador, pero también se tienen unas ciertas limitaciones.

La primera limitación de los bucles *parfor* es que las iteraciones deben ser independiente entre sí. Para ello, se muestra el siguiente bucle.

x=1;

for i=1:100

if <condicion>

x=x+1;

end

end

En este caso, al terminar el bucle, x tendrá un valor cualquiera dependiendo de las veces que se haya entrado en el bloque condicional. Por otro lado, si este bucle fuera de tipo *parfor* no funcionaría, debido a que el valor de x depende del valor que tomara en las anteriores iteraciones. Al dividir el bucle, esta información se perdería.

La otra de las limitaciones es que se debe detectar previamente el tipo de las variables que actúan dentro del bucle. Los tipos se comentan a continuación:

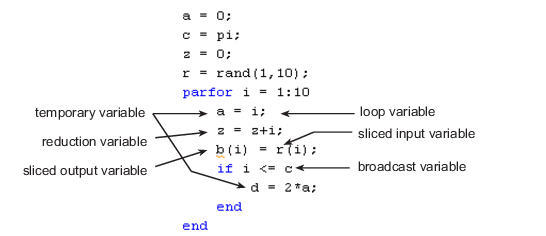


Ilustración 72: Tipos de variables en bucles parfor

* **Temporary variable:** Variable creada dentro del bucle, pero a la que no se accede fuera de este.
* **Loop variable:** Variable de iteraciones del bucle
* **Broadcast variable:** Variables definidas fuera del bucle, y a las cuales se accede pero no se les asigna nunca ningún valor
* **Reduction variable:** Su valor cambia en cada iteración, pero el cambio es independiente del orden en el que se hagan las iteraciones.
* **Sliced variable:** Son variables de tipo vectorial, a las que se accede a índices concretos dependiendo del índice de iteración actual. Pueden ser de entrada o de salida.

De todas estas variables, las más difíciles de utilizar son las *sliced,* ya que dentro del bucle solo se puede acceder a ellas de una forma concreta. Esto es debido a que estas variables también deben de partirse al igual que se parte la variable de iteración. Existen algunos trucos para utilizarlas de forma legal sin producir errores.

Por otro lado, se recomienda utilizar el mínimo número de variables de *broadcast*, ya que estas tienen que copiarse a cada *worker*, lo cual puede producir sobrecargas de memoria.

Por último, mencionar que este tipo de bucles son interesantes cuando el número de iteraciones es muy grande y hay que hacer muchos cálculos. Para bucles pequeños, es más eficiente utilizar bucles *for* clásicos, ya que los *parfor* requieren un tiempo para intercambiar la información con la interfaz principal.

APÉNDICE C: INTERFACES GRAFICAS EN MATLAB

En este apéndice se comentan algunos de los conceptos básicos a tener en cuenta en la creación de interfaces gráficas de usuario (GUIs) en Matlab.

En Matlab, a cada instancia de una interfaz gráfica se la conoce como figura. Una figura es el equivalente a lo que se denomina como formulario en otros lenguajes de programación. A priori, una figura no es más que una ventana vacía. A través de la herramienta de creación de interfaces gráficas de Matlab, llamada GUIDE, se pueden editar las figuras para añadirle una serie de componentes, como por ejemplo:

* Botones
* Etiquetas
* Cuadros de entrada de texto
* Menús desplegables
* Paneles de agrupación, para introducir otros componentes
* Paneles gráficos

A continuación, se muestra una captura de la herramienta de creación de GUIs, en la que se muestra una de las figuras que se utiliza en el presente proyecto:

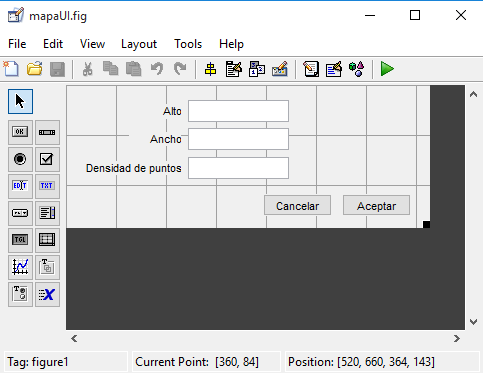


Ilustración 73: Captura de GUIDE

Cada elemento tiene una serie de características. Algunas son comunes, y otras especificas según el tipo. Por ejemplo, *String* contiene el texto del elemento. En el caso del botón de aceptar, el contenido es ‘Aceptar’. Otro parámetro importante es el tag, que es el nombre que se le asigna al elemento de forma interna. Por último, también cabe destacar la posición, que representa la posición que ocupa el elemento en la figura. A continuación se muestra una captura del editor de parámetros de un botón:

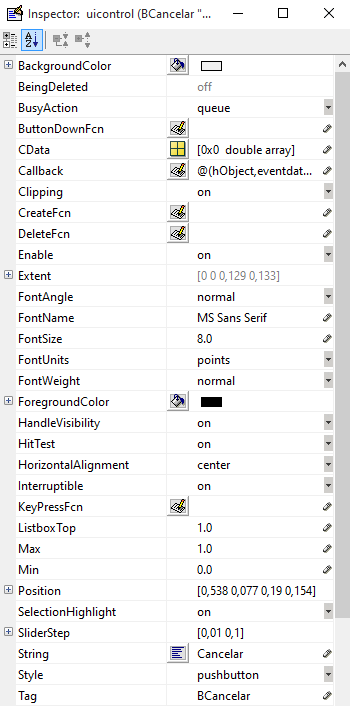


Ilustración 74: Editor de parámetros

Otros de los parámetros más importantes son los *Callbacks*. Un *Callback* es la función que se ejecuta después de que un evento tenga lugar, como por ejemplo, un click en el objeto. Una figura se guarda en un archivo de extensión .fig, y, además de este, se guarda otro archivo con el mismo nombre y extensión .m. En este archivo .m es donde se deben encontrar los distintos *callbacks* de los elementos que conforman la figura.

Para entrar en la programación de GUIs, el elemento más importante a tener en cuenta son los *handles*. Un *handle* de Matlab es el equivalente a lo que en otros lenguajes se conoce como puntero. Cada elemento gráfico tiene un handle asociado. De modo, que si se quiere cambiar o obtener un parámetro de un elemento gráfico, se haría de la siguiente manera:

x=get(handle,nombreParametro)

set(handle,nombreParametro,valor)

Una función muy importante a la hora de programar interfaces gráficas en Matlab es *guidata()* Esta función devuelve un *struct* que contiene como atributos todos los tags asociados a una figura. Cada atributo tiene como valor el handle del elemento al que representa. Con esto, se puede tener acceso desde el código a cada elemento gráfico. Además otro de los usos que tiene este *struct* es almacenar información personalizada, de forma que los distintos *callbacks* puedan compartirla con simplemente llamar a *guidata*().

Otro de los elementos importantes a tener en cuenta son los paneles gráficos o *axes*. En ellos, se pueden representar los distintos gráficos en 2D y 3D que sean necesarios. Su uso es similar al que se está acostumbrado en el uso general de Matlab. Basta recordar que para pintar en un *axes* concreto, hay que pasar como primer parámetro su handle a las funciones que se encargan de dicha tarea, como *plot(), surf(), contour(), hold(),* etc.

No hay que olvidar que Matlab es un lenguaje *single-thread*, con lo cual si durante la ejecución de un *callback* se activa otro, estos no van a ejecutarse de manera simultánea. En este caso, hay que definir para cada elemento gráfico la política a seguir en caso de que ocurra esto. Hay dos opciones: interrumpir el *callback* actual, o poner el *callback* nuevo en una cola.

Finalmente, mencionar que las interfaces gráficas en Matlab pueden ser de tipo *singleton* o no. Si la GUI es *singleton*, solo puede haber una instancia de esta de forma simultánea. Haciendo referencia al simulador desarrollado en el proyecto, la interfaz **MainUI** es *singleton*, y por ejemplo, **figuraRecepcion** no lo es.