



UNIVERSIDAD  
DE MÁLAGA



## ESCUELA DE INGENIERÍAS INDUSTRIALES

**Ingeniería de Comunicaciones**

**Teoría de la señal y comunicaciones**

## TRABAJO FIN DE GRADO

**Técnicas de localización en interiores para sistemas inalámbricos**

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Itinerario en Sistemas Mecatrónicos en Vehículos

Autor: Alberto Gabriel Buftea

Tutor: Francisco Javier López Martínez

MÁLAGA, 10 de Junio de 2019



# Resumen

El incremento de los dispositivos conectados a la red denominada *IoT Internet Of Things* expande las aplicaciones que tiene localizar los dispositivos, incluso cuando estos se encuentran dentro de los edificios.

Como la señal *GPS* no se puede usar en estos casos, se han de implementar nuevos sistemas denominados *ILS, Indoor Positioning Sistem*, que se encargan explícitamente de determinar la ubicación de los objetos en interiores.

Los sistemas *ILS* consisten en varios dispositivos emisores/receptores junto con al menos un dispositivo coordinador que tiene un poder de procesamiento superior. A todos estos dispositivos se les denominan nodos y pueden implementarse siguiendo cualquier topología de red existente.

De todas las tecnologías existentes en el mercado para la instalación de un sistema de localización en interiores, se prefiere el uso de las que trabajen con ondas de radio pues son las que habitualmente usan los dispositivos inalámbricos, como *Bluetoooh* y *WiFi* por ejemplo.

Para implementar los algoritmos de localización se ha de medir una característica de dicha señal. En este caso decidimos que es preferible medir la potencia de señal recibida *RSSI, Received Signal Strength*, ya que presenta varias ventajas respecto a otras mediciones.

Realizando un estudio más profundo de la *RSSI* vemos que es poco probable determinar la ubicación exacta del objeto, puesto que las técnicas de localización trabajan con distancias y se necesita conocer con exactitud la atenuación que sufre la señal mientras recorre el canal para determinar la distancia exacta. La atenuación depende de varios factores que pueden variar con el tiempo, es por ello muy complicado determinar.

Así, estos sistemas han de trabajar con cierta incertidumbre en cuanto a la atenuación. Por lo tanto, los algoritmos empleados usan aproximaciones cometiendo un error en el cálculo de la posición del objeto.

Para implementar cualquiera de los algoritmos de localización, será necesario conocer la posición de al menos tres nodos del sistema. Veremos a lo largo de este trabajo que los factores que más influyen en los errores cometidos son, la distancia máxima que hay entre el objeto y dichos nodos y la posición relativa entre el objeto y los tres nodos. Factores que influyen en mayor medida que la incertidumbre en la atenuación.

Así, se determina que es importante tener una determinada densidad de nodos para acotar el error dentro de un rango necesario.



# Dedicación

Soy partidario de una educación libre, internacional y de calidad. Éste trabajo va dedicado a todos aquellos alumnos, investigadores, profesores y miembros de la comunidad científica que emplean su tiempo personal en hacer de este mundo un sitio mejor.



# Agradecimientos

Deseo comenzar agradeciendo a mis padres, por el soporte y apoyo incondicional que me han proporcionado en los momentos de gran estrés. Han vivido y sufrido mis errores, aún así, siempre han estado a mi lado animándome a seguir luchando. Gracias por ocupar un lugar especial en mi corazón.

Después quiero agradecer en especial a mí tutor, Javier. Él, que ha presenciado todas las fases por las que ha pasado este *TFG*, ha sabido guiar mis pensamientos e inquietudes de una forma que me motive a investigar y afrontar con ambición los retos y problemas surgidos. Gracias por confiar en mí.

Por último, pero no menos importante, deseo agradecer a mi grupo de amigos, en especial a aquellos que forman o han formado parte de la comunidad universitaria. Llevamos muchos años viviendo experiencias en común, apoyándonos mutuamente y haciendo de esto una gran familia. Gracias por aceptarme tal y como soy.



# Palabras Clave

- *Localización en interiores*
- *Sistemas inalámbricos de localización*
- *Tecnologías para sistemas ILS*
- *Topologías para una red de sensores inalámbricos*
- *Nodos en un sistema inalámbrico de localización*
- *Mediciones usadas para la localización*
- *Nivel de potencia de la señal de radio RSSI*
- *Modelo matemático de las pérdidas del canal y coeficiente de atenuación*
- *Incertidumbre en la atenuación de la señal*
- *Técnicas y algoritmos de localización RSSI*
- *Densidad de nodos en un sistema ILS*
- *Posición de los nodos en un sistema de localización inalámbrico*
- *Precisión de un sistema ILS*
- *Simulación de un sistema ILS*
- *Distancia en la precisión de un sistema ILS*
- *Comparación técnicas de localización en interiores*



# Índice general

|  |           |
|--|-----------|
| <b>1. Introducción</b>   | <b>1</b>  |
| 1.1. Antecedentes . . . . .  | 2         |
| 1.2. Objetivo . . . . .  | 3         |
| 1.3. Estructura . . . . .  | 3         |
| <b>2. Fundamentos teóricos</b>   | <b>5</b>  |
| 2.1. Localización inalámbrica en entornos cerrados . . . . .   | 5         |
| 2.2. Tecnologías para la implementación de sistemas <i>ILS</i> en interiores . . . . .                         | 13        |
| 2.3. Pérdida de potencia de la señal de radio <i>RSSI</i> ( <i>Radio Signal Strength Indicator</i> ) . . . . . | 17        |
| 2.4. Técnicas de localización por <i>RSSI</i> . . . . .  | 20        |
| <b>3. Técnicas de localización propuestas</b>  | <b>24</b> |
| 3.1. Parámetros considerados . . . . .   | 25        |
| 3.2. Descripción de los algoritmos usados . . . . .  | 28        |
| 3.3. Presentación de resultados . . . . .  | 43        |
| <b>4. Aplicación práctica</b>  | <b>58</b> |
| 4.1. Contexto, preparación del experimento y algoritmo . . . . .   | 59        |
| 4.2. Presentación de los resultados . . . . .  | 62        |
| 4.3. Estimación de costes . . . . .  | 67        |
| <b>5. Conclusiones</b>   | <b>71</b> |
| 5.1. Principales Aprendizajes . . . . .  | 71        |
| 5.2. Líneas futuras . . . . .  | 73        |
| <b>Bibliografía</b>  | <b>73</b> |
| <b>A. Códigos de <i>MatLab</i></b>   | <b>76</b> |
| A.1. <i>Script ErrorsVsAlpha</i> . . . . .   | 76        |
| A.2. <i>Script ErrorsPerDensity</i> . . . . .  | 80        |
| A.3. <i>Script ParkingTrajectory</i> . . . . .   | 84        |
| A.4. Funciones de generación de nodos . . . . .  | 86        |
| A.4.1. <i>Function GenerateEquidistantPoints</i> . . . . .   | 86        |
| A.4.2. <i>Function GenerateTrackingPoints</i> . . . . .  | 87        |
| A.5. Funciones para el cálculo de distancias . . . . .   | 88        |
| A.5.1. <i>Function GetDistancesToFixedPoints</i> . . . . .   | 88        |
| A.5.2. <i>Function GetClosestPoints</i> . . . . .  | 89        |
| A.6. Funciones para la estimación de la posición . . . . .   | 90        |
| A.6.1. <i>Function CalculateLocalizationTheory</i> . . . . .   | 90        |
| A.6.2. <i>Function MakeCirclesCross</i> . . . . .  | 92        |

|  |            |
|--|------------|
| A.6.3. <i>Function</i> CalculateLocalizationMaxMin . . . . .   | 95         |
| A.6.4. <i>Function</i> CalculateLocalizationCentroid . . . . . | 96         |
| A.6.5. <i>Function</i> CalculateLocalization . . . . .         | 97         |
| A.6.6. <i>Function</i> GetRectanglesArea . . . . .             | 98         |
| A.6.7. <i>Function</i> ApplyCenterOfArea . . . . .             | 99         |
| A.6.8. <i>Function</i> ApplyTriangleMethod . . . . .           | 99         |
| A.7. Funciones para el cálculo de errores . . . . .            | 100        |
| A.7.1. <i>Function</i> CalculatePositionErrors . . . . .       | 100        |
| A.7.2. <i>Function</i> CalculateErrorsPerDensity . . . . .     | 101        |
| A.7.3. <i>Function</i> CalculateRelativeErrors . . . . .       | 101        |
| <b>B. Fichas de datos de los dispositivos</b>                  | <b>102</b> |
| B.1. Ficha de datos del Arduino . . . . .                      | 102        |
| B.2. Ficha de datos de XBee . . . . .                          | 107        |

# Índice de figuras

|  |    |
|--|----|
| 1.1. incremento de los dispositivos conectados al IoT, 2012-2020, en miles de millones . . . . .   | 1  |
| 2.1. Topologías existentes para redes inalámbricas . . . . .   | 6  |
| 2.2. Mediciones para implementar las técnicas de localización . . . . .  | 8  |
| 2.3. Ángulo de llegada . . . . .   | 9  |
| 2.4. Localización empleando la medida del TDoA . . . . .   | 11 |
| 2.5. Localización por trilateración . . . . .  | 12 |
| 2.6. Triangularización . . . . .   | 21 |
| 2.7. Punto de cruce entre tres círculos . . . . .  | 22 |
| 2.8. Técnica Min-Max . . . . .   | 23 |
| 2.9. Técnina de la centroide . . . . .   | 23 |
| 3.1. Coeficiente de atenuación según la distancia entre emisor y receptor, para distintas frecuencias . . . . .                            | 26 |
| 3.2. Determinar error en la estimación de la distancia . . . . .   | 29 |
| 3.3. Determinar error en estimación de la posición . . . . .   | 30 |
| 3.4. Determinar errores en función de la densidad de nodos . . . . .   | 32 |
| 3.5. Delimitar errores para una densidad de nodos específica . . . . .   | 33 |
| 3.6. Estructura lógica de <i>GetClosestPoints</i> . . . . .  | 37 |
| 3.7. Error cometido al calcular la distancia entre emisor y receptor . . . . .   | 44 |
| 3.8. Error cometido al calcular la distancia considerando $\epsilon$ del 20 % . . . . .  | 45 |
| 3.9. Error cometido al calcular la distancia entre emisor y receptor . . . . .   | 45 |
| 3.10. Posiciones consideradas para los nodos . . . . .   | 47 |
| 3.11. Error en la posición frente error en $\alpha$ . . . . .  | 48 |
| 3.12. Circunferencias y estimación de la posición para distintos valores de $\epsilon$ en la posición central . . . . .                    | 50 |
| 3.13. Circunferencias y posiciones estimadas para valores de $\epsilon$ elevados . . . . .   | 51 |
| 3.14. Comparación entre <i>CalculateLocalization</i> , <i>CalculateLocalizationMaxMin</i> y <i>CalculateLocalizationCentroid</i> . . . . . | 52 |
| 3.15. Errores cometidos en la posición al considerar valores de $\alpha_s$ inferiores a $\alpha$ . . . . .                                 | 53 |
| 3.16. Errores máximos, mínimos y medios en función de la densidad de nodos existente . . . . .   | 55 |
| 3.17. Errores medios en función de la densidad de nodos y $\epsilon$ . . . . .   | 55 |
| 3.18. Errores cometidos al estimar las posiciones en el mapa de $400m^2$ . . . . .   | 56 |
| 4.1. Complejo residencial diseñado en Bordeaux . . . . .   | 59 |
| 4.2. Esquema del aparcamiento . . . . .  | 60 |
| 4.3. Estructura lógica de <i>ParkingTrajectory</i> . . . . .   | 61 |
| 4.4. Trayectorías definidas líneas continuas . . . . .   | 61 |
| 4.5. Trayectorias y estimación con un incertidumbre del 15 % con 108 nodos fijos en el garaje . . . . .                                    | 63 |

|  |    |
|--|----|
| 4.6. Errores cometidos durante la estimación con una incertidumbre del 15 % y 108 nodos fijos en el garaje . . . . . | 64 |
| 4.7. Trayectorias y estimación con una incertidumbre del 15 % con 162 nodos fijos en el garaje . . . . .             | 65 |
| 4.8. Errores cometidos durante la estimación con una incertidumbre del 15 % y 164 nodos fijos en el garaje . . . . . | 66 |
| 4.9. Errores durante la estimación de las trayectorias a distintas incertidumbres y densidades de nodos . . . . .    | 67 |

# Índice de tablas

|   |    |
|---|----|
| 2.1. Comparativa de las tecnologías disponibles para implementar localización . . . . . | 16 |
| 4.1. Precios de los materiales . . . . .  | 68 |
| 4.2. Coste de la mano de obra . . . . .   | 69 |

# Capítulo 1

## Introducción

Actualmente se está expandiendo la cantidad de dispositivos inteligentes conectados a la red IoT (Internet of Things). Desde teléfonos inteligentes hasta lavadoras, frigoríficos y microondas. Con el desarrollo de las aplicaciones móviles se ha conseguido sacarle una utilidad extra a todo dispositivo electrónico fabricado actualmente si este se conecta a Internet. Esto se confirma por los datos de dispositivos conectados a Internet cada año figura 1.1 donde pueden observar como el número de dispositivos sigue incrementando, con la previsión para el año que viene de tener casi diez mil millones de dispositivos conectados más.

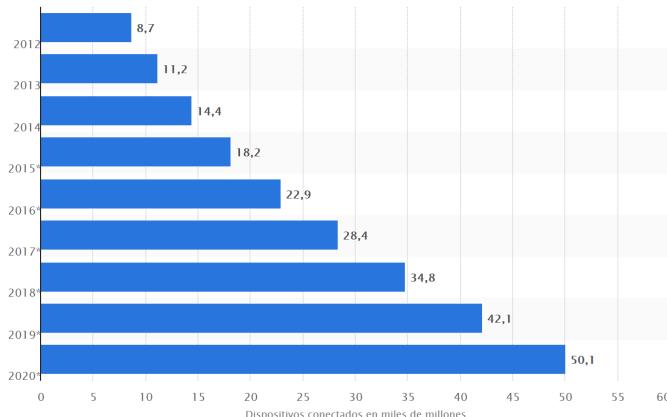


Figura 1.1: incremento de los dispositivos conectados al IoT, 2012-2020, en miles de millones

Fuente: [20]

Puede intuirse que cada vez será de mayor utilidad tener localizados algunos de estos dispositivos electrónicos cuando se encuentren dentro de edificios. El rango de aplicaciones que esto tendría sería colosal y es vital para el desarrollo de las nuevas arquitecturas inteligentes que están emergiendo en la industria tecnológica (ciudades inteligentes, edificios inteligentes, redes inteligentes).

Por poner un simple ejemplo, piensen en el wifi gratuito que ofrecen algunas cadenas de supermercados. Para poder conectarse se ha de aceptar la política de privacidad, habiendo leído las condiciones de uso. Buscando productos en la aplicación de dicho supermercado, de repente salta un *pop-up* con un descuento de 20 % en zumos, qué casualidad que te encuentres justo en el pasillo de los refrescos.

## 1.1. Antecedentes

La geo-localización juega un papel importante en los distintos ámbitos de la sociedad actual. Raro es no tener instalada una aplicación en el teléfono inteligente que precise recolectar la ubicación del mismo para hacer recomendaciones al usuario (restaurantes, cines, centros comerciales cercanos) o usar dicha información por motivos estadísticos.

Este tipo de localización se realiza mediante el uso del sistema de localización global (GPS), el cuál hace uso de los satélites para determinar la localización de un determinado objeto sobre la tierra. Esto se consigue gracias a las señales intercambiados entre el satélite y el objeto a localizar.

Sin embargo, este modo de funcionamiento resulta imposible de implementar para la localización de objetos en entornos cerrados puesto que se necesita la correcta transmisión y recepción de las señales entre el satélite y el objeto. Estas señales se ven atenuadas por los materiales de la construcción, y por tanto, la cobertura de la señal *GPS* en interiores se ve reducida.

En los últimos años se han llevado a cabo muchas propuestas e investigaciones sobre los sistemas *ILS*, aún así ninguno de ellos ha conseguido alcanzar la misma popularidad que los sistemas de localización en el exterior como los *GPS*. Esto se debe en gran parte al factor económico. La implementación de estos sistemas es cara puesto que cada edificio necesita una infraestructura propia y un sistema de gestión para determinar la posición de los objetos. El tiempo necesario para recuperar la inversión en comparación a los beneficios económicos quizás no haya sido atractivo, hasta ahora. Con la incremento de la popularidad de la *IoT*, esto está cambiando.

Volviendo al ejemplo del supermercado, la localización se lleva a cabo debido a la existencia de unos dispositivos emisores/receptores que se comunican entre ellos, el teléfono móvil y el repetidor de señal *WiFi*, de la misma forma que se comunica un satélite con un dispositivo *GPS* para determinar su posición en latitud y longitud.

En el caso de los sistemas *ILS*, a estos dispositivos que se comunican entre sí se les llaman nodos y se distinguen dos tipos. Los nodos fijos, que son aquellos de posición conocida y constante y los nodos móviles, que habitualmente serán los dispositivos cuya posición es desconocida y se ha de determinar.

Como vemos, las aplicaciones comerciales que presentan estos sistemas están incrementando, lo que se traduce en la posibilidad de obtener mayores beneficios por su implementación. Es más, al ser posible realizar este tipo de localización usando la tecnología *WiFi*, se puede aprovechar la infraestructura ya existente en edificios para implementar el sistema de localización en entornos cerrados, abaratando los costes.

Ahora bien, se necesitará realizar un estudio para ver la viabilidad de usar dicha infraestructura. Como es habitual en cualquier trabajo, surgen cuestiones, ¿será suficiente con la cantidad de repetidores instalados? ¿cuál es el margen de error para conseguir la precisión deseada?.

## 1.2. Objetivo

El objetivo de este trabajo de fin de grado es determinar cómo se puede realizar la localización de objetos en un mapa de dos dimensiones (simulando la localización en entornos cerrados), determinar el margen de error máximo admisible para asegurar la precisión necesaria y una densidad de nodos fijos mínima para evitar puntos muertos.

Los puntos principales a desarrollar son:

- Determinar las técnicas y tecnologías usadas para realizar localización en entornos cerrados
- Diseñar un algoritmo que determine la localización de un objeto en función de la señal recibida
- Estudiar qué sucede por la presencia de una incertidumbre al considerar la attenuación del nivel de señal recibida
- Analizar la consecuencia de incrementar o disminuir la densidad de nodos por unidad de superficie en las prestaciones del sistema.
- Realizar una simulación de un caso práctico con el número de nodos y el error máximo determinado
- Efectuar una estimación de los costes que conllevaría implementar el sistema *ILS* para el caso práctico analizado.

## 1.3. Estructura

Se divide esta memoria en cinco capítulos siguiendo una distribución lógica que resuelve las dudas del lector en el orden que surgen naturalmente al interesarse por la materia de estudio. Todos los algoritmos de programación empleados se implementarán haciendo uso del software *MatLab* gracias a que se dispone de licencia universitaria.

Se procede a detallar cada capítulo y su contenido.

- Capítulo 1 (**Introducción**)

Siendo éste el actual, se pretende dar una visión general del ámbito y el problema que abarca este trabajo de fin de grado. Además, se persigue transmitir al lector la motivación que ha empujado a la realización de esta investigación.

- Capítulo 2 (**Fundamentos teóricos**)

En este capítulo se realizará una breve explicación del funcionamiento de un sistema de localización en interiores, veremos las tecnologías disponibles en el mercado y las distintas técnicas posibles para implementarlo. Luego se indagará más en las técnicas de localización por la lectura de la *RSSI*.

- Capítulo 3 (**Técnicas de localización propuestas**)

En este apartado se construye un modelo de simulación del sistema *ILS*. Describiremos el funcionamiento de los algoritmos desarrollados, se explicarán las principales

variables tenidas en cuenta y se mostrarán los resultados de las simulaciones sacando las conclusiones pertinentes.

- Capítulo 4 (Aplicación práctica)

A partir del trabajo realizado en el apartado anterior, se muestra un ejemplo de aplicación práctica para un caso real. Se implementan nodos fijos en un aparcamiento y se procede a estimar la posición de vehículos. También se realiza una estimación del coste de implementación del sistema de localización.

- Capítulo 5 (Conclusiones)

Por último, se analizarán los resultados y se presentarán las principales conclusiones obtenidas. Se verán los principales factores no tenidos en cuenta al tomar idealizaciones y se propondrán futuros trabajos a realizar para extender y continuar el estudio realizado en éste.

- Apéndice A (Códigos de programación)

A lo largo de esta memoria no se incluirá el código de ninguna función o *script* de *MatLab*. Explicaremos su funcionamiento de manera gráfica y escrita para facilitar su comprensión. Todos los códigos usados se introducirán en este apéndice, para posibilitar su consulta en caso de interés.

- Apéndice B (Fichas de datos de los dispositivos)

Con la estimación de los costes de implementación se tendrán en cuenta las características de varios dispositivos. Para facilitar la consulta de los datos indicados, se incluyen las ficha técnicas en este apéndice.

# Capítulo 2

## Fundamentos teóricos

En este capítulo se hace una introducción a la localización de objetos en interiores y se habla sobre las técnicas que se llevan a cabo para su implementación. Luego se profundizará más en la localización basada en el nivel de señal *RSSI* puesto que será la escogida para realizar las simulaciones y los algoritmos del siguiente apartado.

### 2.1. Localización inalámbrica en entornos cerrados

La finalidad de un sistema de localización *ILS*, por sus siglas en Inglés *Indoor Localization System*, es determinar la ubicación de un objeto con una precisión mínima determinada por el problema que se desea resolver. Así, se puede cometer un error de varios metros si lo que deseamos es conocer la tienda que más frecuentemente visita una persona en un centro comercial, sin embargo, si querremos determinar en qué pasillo del supermercado se encuentra dicha persona, el error cometido deberá ser inferior al metro. Como se puede leer en [13], existen diferentes tecnologías en el mercado que permiten realizar la localización inalámbrica de objetos. Los dos principales factores que se usan para clasificarlas son el algoritmo que se emplea para determinar la ubicación del objeto y la capa física del sistema *ILS*. Estos sistemas realizan una serie de medidas sobre las señales recibidas y estiman la posición realizando una serie de cálculos en función de ellas, es por ello que el sistema deberá estar compuesto de al menos dos componentes, un módulo transmisor/receptor de señal y un módulo de procesamiento encargado de realizar las medidas y los cálculos necesarios.

En [3] se menciona que el sistema de computación que calcula la posición de un objeto móvil dentro de un edificio basa sus cálculos en una serie de nodos cuya posición es conocida, denominados nodos fijos, instalados en el interior de dicho edificio. Entendemos por nodo a cualquier dispositivo emisores/receptor de señal que forme parte del sistema mientras que la herramienta de computación es cualquier dispositivo que tenga la capacidad de recibir toda la información transmitida por los nodos y realizar operaciones lógicas y algorítmicas en función de la misma.

En [2] se ve que existen diferentes formas de colocar los sensores en una red inalámbrica, lo que se denomina topología de la red, se presenta en la *figura 2.1* un esquema de cada una de ellas. Describiremos dichas topologías pensando en un sistema *ILS*, así, mencionamos que el nodo rojo *coordinador* se considera el sistema de control central, los nodos verdes *de enrutamiento* se equiparan con dispositivos emisores-receptores de señal cuya función es recibir toda la información de una serie de dispositivos emisores y entregarla al sistema central. Los nodos azules *finales* son dichos dispositivos emisores que no tienen

comunicación con otro dispositivo de la red, tan sólo emite la información de la forma que ha sido programado.

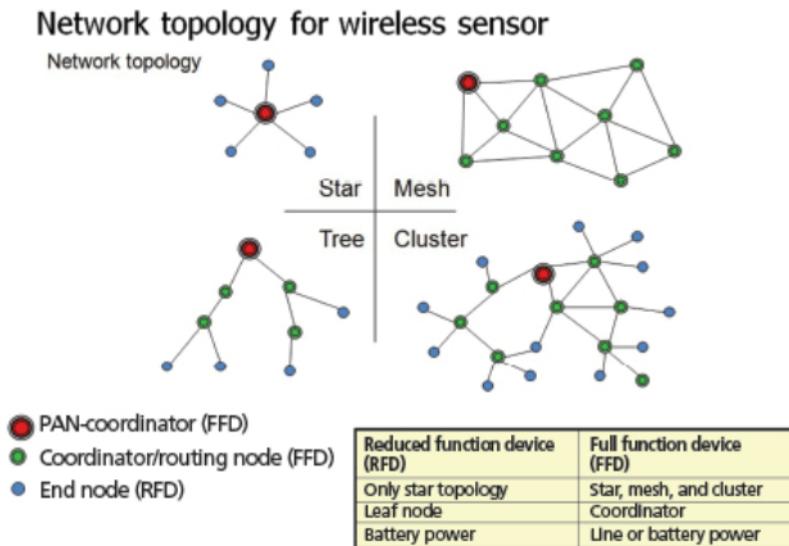


Figura 2.1: Topologías existentes para redes inalámbricas

Fuente: [2]

Es importante destacar que los nodos de enrutamiento pueden tener la capacidad de realizar pequeñas operaciones para entregar al sistema de control central la información parcialmente procesada. Sería el caso de una placa Arduino con un módulo inalámbrico instalado. Es más, si las dimensiones del sistema y la capacidad de procesamiento lo permite, un nodo puede ejercer la función de coordinador y sistema de control a la vez. Procedemos a explicar cada una de las topologías de la figura enfocando la explicación hacia los sistemas *ILS* objetivo de este TFG. Se mencionará un ejemplo de aplicación práctica para facilitar su entendimiento.

## 1. Topología en estrella

Es la más básica y consiste de un nodo central que recibe y procesa la información de varios nodos finales. Este tipo de topología podría usarse cuando es el propio sistema central el que trata de ubicarse a si mismo, para lo cual necesitará conocer la ubicación de los nodos finales. Aunque la función de los nodos finales es tan sólo la de emitir información, esta topología tiene la gran ventaja de que estos nodos consumen muy poca potencia ya que se podrían poner en suspensión cuando no están transmitiendo. Otra gran ventaja de esta topología es que evita un fallo del sistema completo si un nodo final da error. Un ejemplo práctico sería la necesidad de un ordenador de ubicarse a si mismo en función de la información que le llega de unos emisores instalados en la pared. Si el ordenador conoce las posiciones de los emisores, podría calcular su propia posición usando las técnicas que se verán más adelante.

## 2. Topología en malla

Consiste en la interconexión de todos los nodos de la red entre sí. Cada uno de los nodos tiene la capacidad de enrutar y procesar la información recibida de otro. Tiene la gran ventaja de que la información puede llegar al nodo central por tantas

vías como combinaciones de nodos haya, por lo que asegura el funcionamiento del sistema aún cuando uno o varios nodos fallan. Siguiendo con el mismo ejemplo de antes, si en vez de esos dispositivos emisores que usaba el ordenador para auto-ubicarse se instalan placas Arduino con módulos inalámbricos y se interconectan entre ellas, obtenemos una topología en malla. Puede verse que la implementación de esta topología para un sistema *ILS* quizás no sea económicamente eficiente.

### 3. Topología en árbol

Consiste en un sistema de control central al cual se conectan varios sistemas que tienen una topología en estrella. Imaginen que se desea gestionar la cantidad de ordenadores en cada habitáculo dentro de un edificio. Si en cada habitación está instalado el sistema de localización que pusimos como ejemplo de la topología en estrella, y todos los ordenadores del edificio informan de su posición a un servidor central, se genera una topología en árbol.

### 4. Topología en grupo

Consiste en una combinación de la topología en estrella y malla. Siguiendo con la línea anterior, si todos los ordenadores del edificio tuviesen la capacidad de comunicarse entre sí, aparte de con el servidor central, se formaría una topología en grupo. Sería de utilidad por si alguno de los ordenadores pierde la conexión directa con el servidor central, en este caso podría mandar su ubicación a través de otro ordenador dentro del edificio.

El siguiente factor muy importante a tener en cuenta para determinar el funcionamiento de un sistema *ILS* es la técnica empleada para realizar la ubicación de los objetos, que es función de la característica que se mide en la señal recibida. En [12] se mencionan tres grandes grupos para clasificar las técnicas de localización en función de las mediciones que se realicen, ángulo de llegada *AOA* por sus siglas en inglés *angle of arrival*, mediciones relacionadas con la distancia *Distance Related* y el perfilado de la intensidad de la señal de radio *RSS* por sus siglas en inglés *Radio Signal Strength*). Vean la clasificación en la figura 2.2.

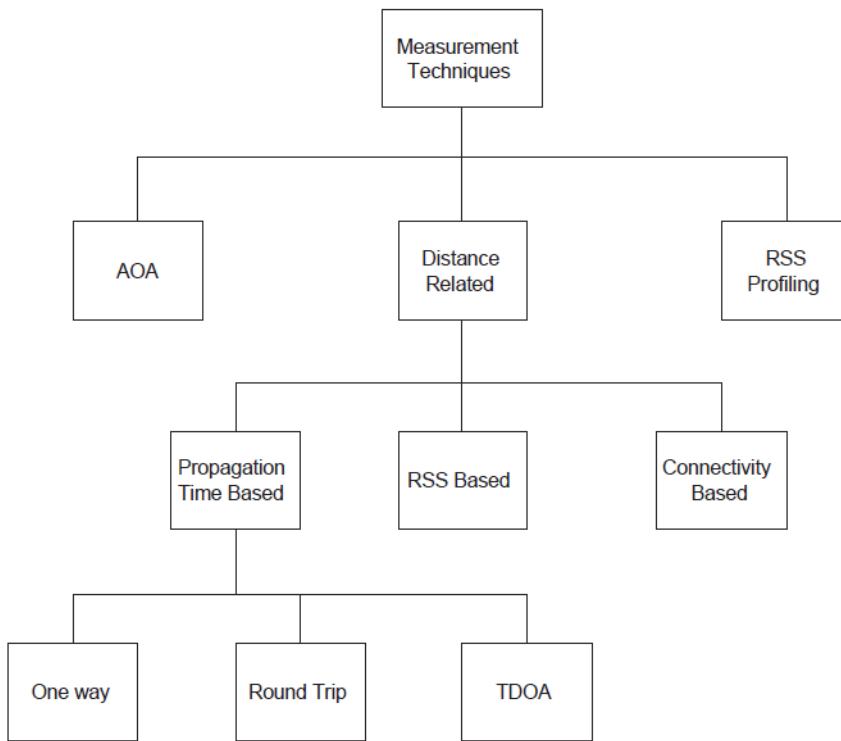


Figura 2.2: Mediciones para implementar las técnicas de localización

Fuente: [12]

Veamos una descripción de cada una de estas mediciones.

### 1. Ángulo de llegada (AOA)

Esta medición se basa en estimar el ángulo con el cual llega la señal transmitida al receptor. Se menciona en [23] que se necesita emplear un array de antenas en el receptor y calculando la diferencia de tiempo que tarda en llegar la señal a cada una de ellas se puede obtener el ángulo.

Esta técnica que se muestra en la *figura 2.3* es efectiva cuando la señal no puede llegar al receptor por varios trayectos, no pueden haber reflexiones, por ejemplo, puesto que en caso contrario se realizaría una mala interpretación del ángulo de llegada. Es por ello que se precisa de visión directa entre el receptor y el emisor para realizar la medida del AOA.

La mayor ventaja de esta medida es que tan sólo se necesitan dos dispositivos receptores para realizar la localización en dos dimensiones, ya que es tan simple como trazar dos líneas rectas, cada una con el ángulo calculado, y la ubicación desconocida del objeto será el punto de cruce de ambas.

La gran desventaja es que conforme aumenta la distancia entre el emisor y el receptor, un pequeño error al calcular el ángulo de llegada se traduce en un error cada vez más grande en la estimación de la posición. Otra gran desventaja es su gran sensibilidad a las reflexiones, lo que dificulta su implementación en interiores.

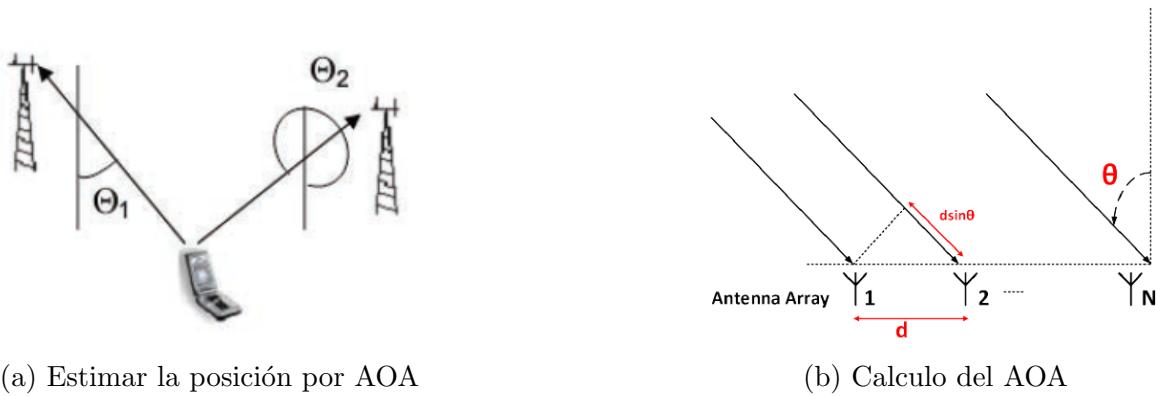


Figura 2.3: Ángulo de llegada

Fuente: [23]

## 2. Mediciones de Distancia

Este tipo de mediciones se basan en estimar la distancia desde el receptor al emisor y realizar los cálculos pertinentes con las distancias obtenidas. A continuación describiremos las diferentes formas en las que se puede obtener la distancia entre el emisor y el receptor.

### a) Tiempo de propagación

Se basa en determinar el tiempo que tarda la señal en llegar desde el emisor hasta el receptor. Vemos que existen varias forma de realizar este proceso así que explicamos cada uno.

- Propagación en un sentido *ToF Time of Flight*

Esta técnica consiste en determinar la distancia entre el emisor y el receptor conociendo el tiempo que ha tardado la señal en viajar por el canal. Esto se consigue codificando el tiempo de envío en la señal emitida, así el receptor puede determinar cuanto ha tardado la señal en llegar computando la diferencia entre el momento en el que fue recibida y el momento en el que fue enviada. Como la señal viaja con la velocidad de la luz " $c = 3 * 10^8 m/s$ " y conocemos el tiempo " $t$ " que ha tardado en llegar al receptor, podremos despejar la distancia recorrida como  $D = c * t$

Vemos en [23] y [12] que el factor más importante a tener en cuenta para la realización de esta medida es la sincronización entre el reloj del emisor y el receptor ha de ser muy exacta, siendo el caso que un error en el tiempo de tan sólo  $1ns$  equivale a  $0,3m$  de error en la distancia estimada. A esto le sumamos el hecho de que se producen errores cuando no hay visión directa entre el emisor y el receptor debido a las difracciones que sufre la señal al rebotar contra obstáculos

- Propagación ida y vuelta *RToF Return Time of Flight*

En este caso se intercambian dos mensajes y ambos nodos involucrados hacen tanto al función de emisor como de receptor. El método consiste en determinar la distancia entre los nodos midiendo el tiempo que ha tardado una señal en llegar de vuelta al nodo que la envió en primera instancia.

Es similar al método *ToF* solo que se toma el tiempo de propagación ida-vuelta, por lo que se ha de ser consciente de que la distancia calculada será el doble de la real en un caso ideal.

Esto incurre el gran beneficio de que no se necesita una sincronización entre los relojes del emisor y el receptor pues se computa el tiempo usando el reloj de tan sólo uno de los dispositivos. Los problemas de esta técnica son la introducción de un retraso adicional, que es el tiempo que tarda el receptor del primer mensaje en procesar la llegada y volver a realizar el envío, y la sobrecarga del canal puesto que cada señal se envía dos veces.

- Diferencia de tiempos de llegada (*TDoA Time Difference of Arrival*)

La idea es determinar la posición de un dispositivo emisor usando como medida la diferencia de tiempo que hay en la llegada de la señal a varios receptores. En [9] se tiene una descripción detallada de la técnica usada para estimar la posición del emisor suponiendo conocidas las posiciones de los receptores.

La idea es comparar las señales recibidas por los pares de receptores  $i, j$  con  $1 \leq i \leq j \leq n$  donde  $n$  representa el número total de receptores. Se obtienen los puntos  $d_{i,j} = c * (\tau_i - \tau_j)$  donde  $(\tau_i - \tau_j)$  es la diferencia en el tiempo de propagación y  $c$  es la velocidad de propagación de la onda y resulta que cada punto  $d_{i,j}$  se encontrará en una hipérbola de focos los receptores y de rango la constante la diferencia  $\Delta d = d_2 - d_1$ . Así se puede resolver la posición del emisor puesto que se encontrará en la hipérbola de ecuación:

$$\Delta d = \sqrt{(x_i - x)^2 + (y_i - y)^2} - \sqrt{(x_j - x)^2 + (y_j - y)^2}$$

La gran ventaja de esta técnica es que no se necesita codificar el tiempo en el cual fue enviado el mensaje, por lo que elimina la necesidad de sincronizar los relojes del emisor y el receptor, sin embargo, requiere sincronizar los relojes de todos los receptores para poder implementar la técnica de localización. En este caso no se emplea trilateración para determinar la posición desconocida del emisor puesto que no se trabaja con la distancia desde un punto fijo hasta el emisor, sino que se trabaja con una distancia relativa así que se emplea una técnica diferente denominada multilateración. En la figura 2.4 tenemos una ilustración sobre como se puede ubicar un objeto usando la medida del TDoA.

b) Distancia por intensidad de la señal *RSS*

Conocida la potencia de la señal emitida, se mide la potencia con la que llega la señal al receptor, y se calcula la distancia suponiendo conocida la atenuación que ha sufrido en su recorrido por el canal. Este es el tipo de medida más usado para la implementación de sistemas *ILS* actualmente, en gran parte debido a su facilidad y bajo coste. Es por ello que esta medida será la considerada en las simulaciones y algoritmos que se desarrollarán en este trabajo, por lo que realizaremos un análisis más profundo de ella en el siguiente apartado.

Por ahora es suficiente mencionar que hay numerosos estudios actuales que se

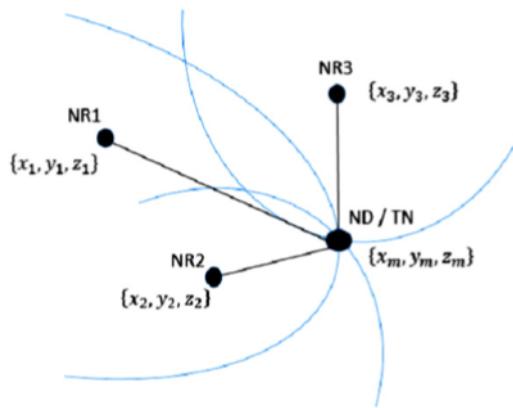


Figura 2.4: Localización empleando la medida del TDoA

Fuente: [9]

enfocan en determinar con una precisión elevada la atenuación de la señal conforme esta recorre el canal y que la *ecuación 1* presentada en la *sección 2.3* es la que establece una relación matemática entre la potencia emitida y recibida de una señal, por lo que es donde se implementa la atenuación.

### c) Distancia por proximidad

Esta es la técnica más simple para determinar la distancia puesto que se basa en la activación de un sensor cuando el objeto se encuentra en su rango de alcance. Se necesita un número grande de sensores para poder estimar la ubicación con una precisión aceptable.

También se le denomina medida binaria puesto que si un sensor devuelve un valor lógico positivo indica proximidad del objeto y si devuelve un valor lógico negativo indica que el objeto no se encuentra en el área definida por su alcance, siendo esta la única información que obtenemos por parte de los sensores, que serían los nodos fijos de posición conocida.

Ubicando los sensores en un mapa formando una malla, podremos estimar la posición como una casilla de dicha malla en función de los sensores que presenten una salida lógica positiva. La precisión del sistema viene directamente relacionado con la cantidad de sensores que ubiquemos y el rango de alcance de estos.

## 3. Perfilado de RSS

Vimos que la medida de la potencia RSS se usa para estimar distancias entre el objeto y los nodos de referencia para luego realizar la localización. Se necesita implementar alguna técnica de trilateración o derivada de ella para realizar la localización con las distancias calculadas. Esto consiste en estimar la posición del objeto como el punto de cruce entre tres circunferencias de radio la distancia calculada. Más adelante se pondrá énfasis en las técnicas de localización empleadas usando como medida la distancia estimada entre el emisor y el receptor puesto que será la opción que escogeremos para el desarrollo de las simulaciones de este trabajo. Véan en la *figura 2.5* una ilustración de cómo funciona la trilateración, se aprecia la necesidad de al menos tres receptores para poder realizar la ubicación.

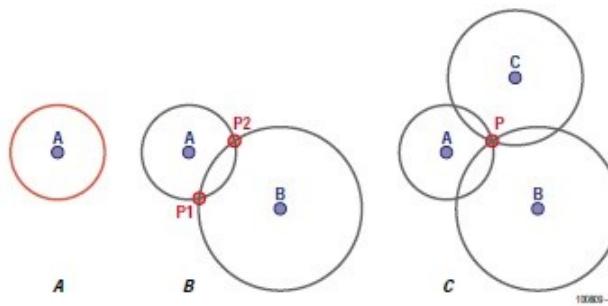


Figura 2.5: Localización por trilateración

Fuente: [16]

La estimación de las distancias no es tan precisa como se necesite, pues como veremos la atenuación de la señal es un elemento dinámico que cambia constantemente en función de muchas condiciones externas, por ello, diseñar un modelo matemático para realizar los cálculos es complicado.

Para resolver este problema se han diseñado las técnicas de perfilado que consisten en realizar mediciones experimentales de las atenuaciones de la señal en el lugar donde se quiera implementar el sistema de localización y realizar un mapa en el cual se almacenan los datos de las atenuaciones por zonas. De esta forma, cuando se desea medir la distancia de un emisor a un receptor, suponiendo conocida la posición de uno de ellos podremos usar el valor de atenuación experimental más reciente, cometiendo de esta manera un menor error en la estimación de la distancia.

Dichas medidas experimentales se pueden realizar tanto de manera presencial, realizando los experimentos personalmente, o bien se pueden instalar unos dispositivos fijos, que formarían parte de los nodos del sistema, que se encarguen de tomar las medidas constantemente proporcionando el valor de atenuación más preciso en cada momento. Vemos en [4] un ejemplo práctico de este método en el cual dividen el mapa en una cuadricula en la cual se anotan las atenuaciones experimentales que realizaron para luego poder usar dicha información al realizar los experimentos.

## 2.2. Tecnologías para la implementación de sistemas ILS en interiores

En esta sección comentaremos las distintas tecnologías disponibles en el mercado para la implementación de localización de objetos en interiores. La intención es proporcionar una visión general al lector sobre que tipo de dispositivos permiten realizar las mediciones comentadas en la sección. En este trabajo no se propone el diseño de un sistema concreto por lo tanto no se profundizará en ninguna de las tecnologías que se listan a continuación, sino más bien se explicará su funcionamiento de manera general usando la información que pueden leer en los documentos de referencia, [23], [13] y [3].

### 1. Wireless Local Area Network *WLAN* (IEEE 802.11)

Popularmente conocido como *WiFi*, éste es un estándar que opera en la banda de frecuencias *ISM Industrial, Scientific and Medical* de 2.4 GHz. Es el estándar usado ampliamente para conectar a internet los diferentes dispositivos inalámbricos, es por ello una tecnología interesante para implementar un sistema *ILS* puesto que se haría uso de una infraestructura ya existente. Por eso es la tecnología más estudiada a nivel académico para implementar este tipo de sistemas.

Su alcance se encuentra entre los 50 y los 100 metros y la medida más habitual que se emplea para realizar la localización es la potencia de la señal *RSSI* por la facilidad que ofrece el protocolo de obtener el valor de la potencia.

Un gran problema de esta tecnología es que las redes implementadas se usan para comunicaciones, por lo que el ancho de banda y la frecuencia de muestreo están pensadas para maximizar el alcance y la capacidad de transmisión de datos, así, obligando la implementación de unos algoritmos muy eficientes para mejorar la precisión, que suele estar entre los 3 y los 10 metros dependiendo de la ubicación y cantidad de repetidores.

### 2. Bluetooth (IEEE 802.15.1)

También opera en la banda de frecuencias *ISM* de 2.4 GHz, sin embargo tiene una capacidad de transmisión y rango inferiores, alcanza una distancia de entre 10-15 metros.

El aspecto positivo de este estándar es que fue concebido para consumir muy poca potencia, por lo que se puede implementar en dispositivos pequeños que presenten una alta duración de la batería.

Al igual que con la tecnología *WiFi* la medida más habitual para realizar la localización es la potencia *RSSI*. La precisión alcanzada se encuentra alrededor de los 3 metros.

### 3. ZigBee (IEEE 802.15.4)

Es la tecnología más usada para la implementación de sistemas *ILS*. El estándar define los niveles más altos de la pila de protocolos y se usa en redes inalámbricas de sensores centrándose en el bajo coste y la eficiencia energética lo que se ve penalizado con velocidades de transmisión pequeñas.

El rango de alcance suele estar entre los 20 y los 30 metros en interiores y, al igual que las tecnologías anteriores que usan la banda *ISM* de 2.4 GHz, la característica de la señal más usada para realizar localización es la potencia *RSSI*.

Otros aspectos que destacan es la imposibilidad de realizar una comunicación inalámbrica directamente entre los dispositivos ZigBee y los dispositivos electrónicos persona-

les como los móviles inteligentes, puesto que no incluye ningún hardware que cumpla con el estándar IEEE 802.15.4, y la facilidad de que las señales sufran interferencias puesto que ZigBee opera en la banda *ISM* sin licencia.

#### 4. Radio Frequency Identification Device (*RFID*)

Esta tecnología fue diseñada para transferir y almacenar datos usando transmisión electromagnética a cualquier circuito compatible. Su funcionamiento se basa en una serie de etiquetas *RFID*, que transmiten información con un formato previamente definido por un protocolo, y un dispositivo lector que tiene la capacidad de decodificar.

Se diferencian dos tipos de tecnologías *RFID*, siendo estas la pasiva y la activa. Ambas pueden operar en un amplio rango de frecuencias, desde bajas frecuencias hasta las microondas.

- a) En la activa, las etiquetas necesitan una fuente de alimentación y pueden alcanzar unos rangos de cientos de metros. Son una gran opción para localización a grandes distancias gracias a su alcance y tienen un bajo coste.
- b) En la pasiva, las etiquetas no necesitan una fuente de alimentación sin embargo, el alcance es muy limitado de tan solo 1 o 2 metros. Se han diseñado sobre todo para reemplazar a los códigos de barra.

#### 5. Ultra Wideband (*UWB*)

Es una tecnología de comunicación nueva que se basa en el envío de pulsos ultra cortos, con una duración inferior a  $1\text{ ns}$ . A diferencia de los sistemas *RFID* convencionales que transmiten en una única banda de frecuencias, los sistemas *UWB* transmite una única señal a diferentes frecuencias simultáneamente, desde los 3.1 a los 10.6 GHz.

Una gran ventaja es que la transmisión de la señal *UWB* es vista como ruido por otras tecnologías inalámbricas. Además, la potencia de la señal transmitida es pequeña por lo que el consumo de las etiquetas *UWB* también lo es, inferior al de las etiquetas *RFID* activas. Además, la corta duración de los pulsos de señal hacen que esta tecnología se vea menos afectada por las refracciones que pueda sufrir la señal al rebotar.

La característica ideal a medir para implementar un sistema *ILS* con esta tecnología sería el tiempo de llegada, *ToF*, *RToF* o *TDoA*. El rango de alcance se encuentra próximo a los 30 metros y en experimentos realizados se ha conseguido realizar la localización de un objeto con una precisión de entre 15 y 30 centímetros.

#### 6. Visible Light Communication (*VLC*)

Se trata de una tecnología emergente para alcanzar transferencias de datos a altas velocidades. Esta tecnología usa el espectro de luz visible entre 400 y 800 THz que se modula y emite gracias a los diodos emisores de luz *LED Light Emitting Diodes* que tiene la capacidad de parpadear a altas frecuencias para realizar la transmisión de los datos. Se precisan de sensores que detecten los pulsos de luz emitidos por los *LEDs*, usualmente fotodiodos puesto que presentan un tiempo de respuesta muy pequeño.

La característica más conveniente a medir de la señal es el ángulo de llegada *AoA*, puesto que se trata de un foco de luz. Un gran ventaja es que al trabajar con la luz es menos susceptible a interferencias y presenta un alcance muy elevado.

El gran inconveniente de esta tecnología es que se necesita línea de visión directa

entre el emisor y el receptor, puesto que de otra forma la lectura del ángulo de llegada *AoA* será imprecisa.

### 7. Infrarrojos (*IR*)

Esta tecnología es similar a la transmisión de datos por *VLC* sin embargo emplea diodos *LED* que emiten ondas de infrarrojos, justo por debajo del espectro visible. Como resultado, es una tecnología mucho menos intrusiva que la *VLC*, pero sufre de la misma limitación de necesitar línea de visión directa para tener una buena estimación del *AoA*.

### 8. Señal Acústica

Esta tecnología se basa en usar los micrófonos de los dispositivos móviles para capturar señales acústicas emitidas por unos nodos fijos de posición conocida y estimar la posición del celular con respecto a estos.

De forma general, estas señales acústicas tienen modulada la hora de transmisión por lo que se mediría el tiempo de transmisión *ToF* para estimar la distancia del móvil al nodo fijo.

A parte de los problemas de sincronización horaria, esta tecnología presenta un problema de contaminación acústica, la necesidad de instalar una infraestructura nueva y decremento en la duración de las baterías de los teléfonos por la necesidad de estar constantemente refrescando llegadas de la señal.

### 9. Ultrasonido

Es igual a la tecnología que emplea una señal acústica, por lo que se basa en medir el tiempo de propagación *ToF*, pero transmitiendo ondas de ultrasonido ( $> 20KHz$ ). Tiene el problema añadido de la velocidad de propagación de la onda de ultrasónica se ve afectada en más proporción por las condiciones atmosféricas, como la humedad y la temperatura. Esto se puede contrarrestar instalando sensores de temperatura y de humedad para corregir el factor de propagación empleado en los cálculos, sin embargo, esto involucra una mayor infraestructura a instalar y un poder de computación superior.

Habiendo visto las tecnologías disponibles en el mercado, mencionamos las principales métricas que se deben tener en cuenta al decidir cual usar para la implementación de un sistema *ILS*, las cuales son: **la precisión, el coste, la eficiencia energética, el rango alcanzable, la facilidad de ampliación y la robustez.**

Así, realizamos la *tabla 2.1* como en [3] para facilitar la clasificación y comparación de las mismas

Comparando las distintas posibilidades vemos que todas las tecnologías comúnmente usadas para la implementación de este tipo de sistemas ofrecen la posibilidad de usar como medida la *RSSI* de la señal. Podemos afirmar que la potencia de la señal es la medida predominante entre las diferentes tecnologías.

Aunque las tecnologías que trabajan con ondas sonoras o lumínicas presentan ventajas como mayor precisión y mejores rangos, en sus respectivos casos, dichas tecnologías requieren la instalación de nueva infraestructura y no son fácilmente escalables. Además presentan poca robustez pues las ondas acústicas son sensibles a las condiciones atmosféricas, mientras que las lumínicas requieren línea de visión directa entre el emisor y el receptor. Todos

| Tecnología  | Medida                     | Precisión | Coste | Eficiencia | Rango      | Ampliación | Robustez |
|-------------|----------------------------|-----------|-------|------------|------------|------------|----------|
| WLAN        | RSSI,ToF,<br>RToF,TDoF,AoA | 3-10 m    | Bajo  | Media      | 50-100 m   | Alta       | Alta     |
| Bluettooth  | RSSI,ToF                   | 1-5 m     | Medio | Elevada    | 10-15 m    | Alta       | Alta     |
| ZigBee      | RSSI                       | 1-10 m    | Medio | Alta       | 10-30 m    | Media      | Media    |
| RFID        | RSSI                       | 1-5 m     | Bajo  | Alta       | 1 – 2m*    | Media      | Media    |
| UWB         | RSSI,ToF,TDoA              | 0.1-1 m   | Alto  | Alta       | 20-30 m    | Baja       | Alta     |
| VLC         | AoA                        | 0.1-0.3 m | Bajo  | Alta       | 5-10 m     | Baja       | Baja     |
| IR          | AoA                        | 0.1-0.3 m | Bajo  | Alta       | 20-25 m    | Baja       | Baja     |
| Acústico    | ToF                        | 0.1-0.5 m | Bajo  | Media      | 1-10 Km    | Media      | Baja     |
| Ultrasonido | ToF                        | 0.1-0.5 m | Alto  | Media      | 0.1-1.5 Km | Baja       | Baja     |

\*Considerando el uso de la tecnología RFID pasiva

Tabla 2.1: Comparativa de las tecnologías disponibles para implementar localización

estos factores hacen que se prefieran las otras tecnologías para la implementación de los sistemas *ILS*, dejando las lumínicas y acústicas para localización en exteriores y submarinas.

Visto esto, analizamos el objetivo que se persigue en la realización de este trabajo, que es determinar cómo se realiza la localización de un objeto o persona en el interior de un edificio y qué factores son los más importantes a tener en cuenta.

Guiándonos por la temática principal de los artículos académicos publicados sobre esta temática y justificado con la comparación realizada en la tabla superior, escogemos estudiar con mayor profundidad las técnicas de localización por *RSSI* ya que se puede implementar con las tecnologías más comunes y es la medida que usa la gran parte de los sistemas *ILS*.

## 2.3. Pérdida de potencia de la señal de radio *RSSI* (*Radio Signal Strength Indicator*)

En esta sección tratamos de comprender mejor qué es la *RSSI* de una señal, porqué y cómo se puede usar dicha medida para determinar una distancia.

*RSSI* es una medida que indica la potencia de una señal de radio recibida. Viendo la descripción realizada en [1] se define como el voltaje medido por un circuito instalado en el receptor. Por ejemplo, es la medida que emplean los dispositivos para indicar cuanta señal WiFi está llegando.

El circuito necesario para medir la potencia *RSSI* es simple, de pequeñas dimensiones, barato y no requiere un consumo de potencia adicional para el dispositivo en el que se instala, por ello esta tecnología es muy favorable para realizar localización.

Para calcular la distancia hacia el punto donde se emitió la señal usando la medida de *RSSI* se necesita de un modelo matemático de la propagación de las ondas radio en el interior del edificio. Este modelo incluye varios parámetros entre los cuales la distancia existente entre emisor y receptor, por lo que se han de conocer todos los demás parámetros para poder despejar y calcular la distancia.

El principal problema es que esto requiere estimaciones muy precisas y no es tolerable a cambios en la sensibilidad del receptor, la potencia de emisión, las características del canal y en la orientación. Así, la exactitud de la distancia calculada está directamente relacionada con la precisión que se tenga al sustituir los parámetros en el modelo matemático empleado.

En [1] se menciona que para el modelado de la propagación de una señal de radio se han de tener en cuenta tres modelos distintos los cuales incluyen, **modelado de la atenuación en el espacio libre**, **modelado de la atenuación por reflexiones** y **modelado de la atenuación a largas distancias**.

Veremos en el capítulo siguiente que la ubicación del objeto desconocido se realizará suponiendo que los tres nodos más próximos son aquellos que reciben una potencia de señal *RSSI* mayor. Este hecho implica que no se consideran las reflexiones que puedan haber sufrido las señales. A casos prácticos del sistema real, esto significa que además de tres nodos lo suficientemente cerca como para no considerar la atenuación por largas distancias, supondremos que dichos nodos se encuentran en línea de visión directa con el objeto, por lo que en el intercambio de señales no se producirán atenuaciones.

Habiendo despreciado el efecto de dos modelos de atenuación comentados, procedemos a analizar el modelo de la atenuación en el espacio libre pues es el que se suele emplear en este contexto de localización.

Como se indica en [21], la *ecuación 2.1* que modela las pérdidas en el espacio libre deriva de la fórmula de transmisión de Friis y define una relación entre la potencia emitida y recibida de una onda de radio entre dos antenas suponiendo que el canal recorrido es.

$$\frac{P_r}{P_t} = D_t D_r \left( \frac{\lambda}{4\pi d} \right)^2 \quad (2.1)$$

En donde:

- $P_r \rightarrow$  Potencia Recibida  
 $P_t \rightarrow$  Potencia Transmitida  
 $D_t \rightarrow$  Directividad antena transmisora  
 $D_r \rightarrow$  Directividad antena receptora  
 $\lambda \rightarrow$  Longitud de onda de la señal  
 $\alpha \rightarrow$  Coeficiente de pérdidas del canal

Partiendo de la *ecuación 2.1* ponemos de manera explícita la dependencia con la distancia  $d$  ya que será el parámetro clave en el sistema de localización basado en *RSSI*, resulta:

$$P_r = P_t D_t D_r \left( \frac{\lambda}{4\pi} \right)^2 \left( \frac{1}{d} \right)^\alpha$$

A la variable  $\alpha$  se le define como el exponente de pérdidas del canal, y es un valor negativo que reduce la señal conforme  $d$  incrementa. Esto representa la atenuación de la señal por el canal, conforme más largo es el recorrido, más potencia se pierde.

Realizamos una última asignación en la ecuación para obtener el modelo matemático simplificado de las pérdidas del canal. Se trata de realizar la asignación  $K = D_t D_r \left( \frac{\lambda}{4\pi} \right)^2$  donde  $K$  es un valor constante adimensional que depende de las características físicas de las antenas del transmisor y el receptor, de la sensibilidad del receptor y de las variaciones que presente el emisor en la transmisión.

Finalmente llegamos a la *ecuación 2.2* que es el modelo matemático simplificado de las pérdidas que sufre la señal en su paso por el canal. Este será el modelo que consideraremos al realizar las simulaciones en el *capítulo 3*.

$$P_r = P_t K \left( \frac{1}{d} \right)^\alpha \quad (2.2)$$

El coeficiente de pérdidas del canal  $\alpha$  es la gran incógnita desconocida que se ha de determinar experimentalmente en la implementación de los sistemas *ILS*.

El aspecto práctico es muy simple de comprender. Varios factores del ambiente en el que ubiquemos el sistema harán que la atenuación de la señal sea distinta a la atenuación en el espacio libre. Como el factor  $K$  incluye todas las variaciones que se deban a características técnicas de la electrónica usada, las variaciones que se produzcan debido al ambiente se deberán ver reflejadas en el coeficiente  $\alpha$ . Así, si realizamos una medida de *RSSI* experimental a una distancia conocida, conoceremos todas las variables de la *ecuación 2.2* excepto  $\alpha$ . Procedemos a despejar  $\alpha$  de la ecuación y tenemos el coeficiente de atenuación del canal en el que se ha realizado el experimento.

Lo que no se comenta en el párrafo superior es que el impacto que tiene el ambiente sobre la atenuación de la señal no es constante. Es decir, el coeficiente  $\alpha$  es distinto para el mismo lugar, en distintos espacios temporales. Es más, dentro de una misma habitación obtendremos diferentes medidas para  $\alpha$  en función de posición en la cual nos ubiquemos. La medición de la *RSSI* se ve afectada por factores como la temperatura, nivel de la batería, interferencias, humedad. Todos estos factores son variables y dificultan la determinación de  $\alpha$ . Existen muchos artículos académicos e investigaciones científicas intentan determinar el coeficiente de pérdidas del canal para las distintas situaciones ambientales y técnicas.

Por ejemplo, [18], enfoca su estudio directamente en la manera de transmitir la *RSSI*. Estudian el efecto que tiene en la lectura factores como la cantidad de lecturas por distancia, otros dispositivos trabajando en la misma frecuencia, la temperatura, la sensibilidad del transpondedor, la longitud del paquete enviado, los nodos cercanos, el nivel de carga de la batería y otras condiciones medioambientales. Tras ello proponen varios modelos matemáticos y realizan experimentos para determinar la mejoría con respecto al modelo simplificado de la ecuación 2.

En [22] se propone un modelo empírico basado en el modelo aquí descrito y escogen un valor de  $\alpha$  según las medidas estadísticas, como la media y la desviación estándar, de varios experimentos que se realizan.

En [7] proponen una técnica que llaman VariLoc basada en balizas multi-rango. En vez de transmitir siempre a la misma potencia, los emisores varían constantemente la potencia de la señal emitida así que, en vez de usar la *RSSI* de la señal, se usa la sensibilidad del receptor para realizar la estimación de las perdidas del canal. Es una técnica interesante sobre todo porque ofrece resultados positivos incluso cuando las lecturas de *RSSI* no son estables.

Repasando los artículos de la biografía y realizando una búsqueda verán que la gran mayoría de estudios e investigaciones realizadas sobre la localización por *RSSI* se basan en la correcta determinación de las pérdidas que sufre la señal y el desarrollo de algoritmos eficientes para reducir los errores a su mínimo nivel.

Es aquí cuando entra la duda sobre qué tan importante es tener una correcta estimación de tal coeficiente  $\alpha$ , ¿ hasta dónde llegan los efectos de una mala estimación ?. No es una cuestión cuya respuesta se pueda encontrar fácilmente en los artículos publicados hasta la fecha.

Esta es la aportación que diferencia este trabajo de los demás artículos en este ámbito. En las simulaciones del *capítulo 3* se estudiará el impacto que tiene escoger un valor de  $\alpha$  erróneo al realizar los cálculos para determinar la ubicación del objeto.

## 2.4. Técnicas de localización por RSSI

Sabemos que los sistemas *ILS* se basan en dispositivos emisores/receptores y un sistema de procesamiento para analizar los datos. A dichos emisores/receptores les llamamos nodos y sabemos que se necesita conocer la posición de algunos de ellos en todo momento para poder realizar la localización, es por ello que se dejan fijos.

Sabemos además que tipo de tecnologías se pueden usar para la implementación del sistema y lo que es más, debido a las ventajas que ofrecen los sistemas que intercambian la información usando señales de radio, hemos decidido indagar más en el indicador de potencia de la señal.

Así, hemos visto que usando dicha medida llamada *RSSI* podemos determinar la distancia que hay entre un nodo de posición conocida *normalmente un nodo fijo* y un nodo de posición desconocida, *usualmente un nodo móvil*.

Como se decía al inicio de este apartado, el objetivo de un sistema *ILS* es determinar la posición de un objeto ubicado en el interior de un edificio.

En esta sección veremos como se puede usar la información de la distancia hacia un nodo fijo para determinar la posición espacial de un nodo móvil cuya posición desconocemos.

A aspectos prácticos, la utilidad de estos sistemas es determinar la ubicación de los objetos en el plano de un edificio, es por ello que enfocaremos el posicionamiento en un mapa de dos dimensiones, siendo este el espacio de trabajo.

Así, recopilando la información que pueden leer en [6], [19], [10] y [17] veamos las distintas técnicas que se emplean para determinar la posición de un objeto en un plano conociendo las distancias hacia los nodos fijos.

- Triangularización

Sabiendo cuales son los tres nodos más cercanos, esta técnica consiste en unirlos con líneas rectas formándose así un triángulo de vértices los tres puntos fijos.

La posición del objeto se estima como el baricentro del triángulo, (*también llamado centroide*), como ven en la *figura 2.6*.

Esta técnica tan solo usa la información de la *RSSI* para determinar los nodos más cercanos, pero no es necesario el cálculo de la distancia, pues solo se necesita conocer su posición.

El gran inconveniente es que la precisión que ofrece esta técnica viene ligada a la cantidad y cercanía de los puntos fijos. En concreto, indiferentemente de la posición del objeto dentro del área ocupada por el triángulo, la posición que se estima es la misma.

El cálculo matemático es muy simple, suponiendo que los nodos nodos fijos son los puntos  $a = (x_a, y_a)$ ,  $b = (x_b, y_b)$  y  $c = (x_c, y_c)$  las coordenadas del baricentro  $o = (x_o, y_o)$  se computa sumando las coordenadas  $x$  e  $y$  de cada vértice y dividiendo por 3, así:

$$x_o = \frac{x_a + x_b + x_c}{3}$$

$$y_o = \frac{y_a + y_b + y_c}{3}$$

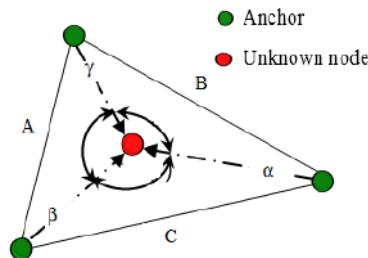


Figura 2.6: Triangularización

Fuente: [19]

- Trilateración

Una forma simple de mejorar la precisión de la técnica de triangularización es usar la medida de *RSSI* para determinar las distancias hacia los tres nodos fijos más cercanos y trazar tres círculos de centro la posición del nodo fijo y de radio la distancia hacia el nodo móvil.

Esta técnica, ilustrada en la *figura 2.7*, se llama trilateración y estima la posición del objeto como el punto de cruce entre las tres circunferencias.

La resolución matemática de las coordenadas  $(x, y)$  se basa en la resolución del sistema de ecuaciones formado por las *ecuaciones 2.3 a 2.5*.

$$(x - x_1)^2 + (y - y_1)^2 = d_1^2 \quad (2.3)$$

$$(x - x_2)^2 + (y - y_2)^2 = d_2^2 \quad (2.4)$$

$$(x - x_3)^2 + (y - y_3)^2 = d_3^2 \quad (2.5)$$

Donde las coordenadas  $(x, y)$  corresponden al punto de cruce entre las tres circunferencias, las coordenadas  $(x_j, y_j)$  con  $j = 1, 2, 3$  se corresponden al centro de las circunferencias, es decir, son las coordenadas de los nodos fijos considerados y las distancias  $d_j$  con  $j = 1, 2, 3$  son las distancias desde el punto de cruce hasta el centro de cada respectiva circunferencia.

En primera instancia este método parece ser el que proporciona una localización óptima, puesto que nos ofrece exactamente la posición del objeto.

Sin embargo, el problema reside en que las medida de las distancias  $d_j$  no son exactas, sino que presentan errores debido a los factores vistos en el apartado superior. Aún así, se puede resolver el sistema de ecuaciones usando técnicas de aproximación por métodos numéricos.

El problema de dichas técnicas es que requieren de cálculos extensos que necesitan un poder de procesamiento que crece exponencialmente con la rapidez que necesitemos que se realice. Esto encarece mucho el sistema, por lo que se han desarrollado otras técnicas derivada de esta para la determinar la ubicación del objeto.

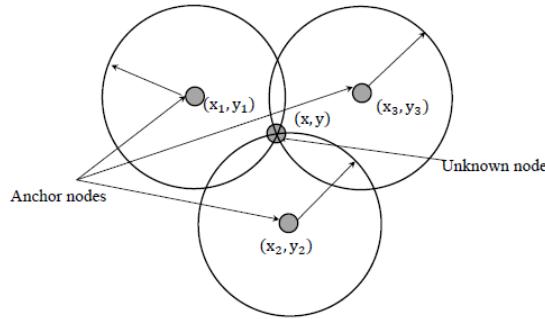


Figura 2.7: Punto de cruce entre tres círculos

Fuente: [15]

- Técnica Max-Min

Como se muestra en la *figura 2.8*, esta técnica consiste en determinar los cuadrados en cuales se encuentran circunscritas las circunferencias de *triangulación* y estimar la posición del objeto como el centro del área que se forma debido al solape de estos tres cuadrados.

Esta técnica adquiere el nombre de *Max-Min* porque las coordenadas del nodo móvil es el centro del área formado por las posiciones  $x_{max}$ ,  $x_{min}$ ,  $y_{max}$  e  $y_{min}$  que indican las posiciones  $x$  e  $y$  extremas del área de solape.

$$x_{min} = \max(x_1 - d_1, x_2 - d_2, x_3 - d_3)$$

$$x_{max} = \min(x_1 + d_1, x_2 + d_2, x_3 + d_3)$$

$$y_{min} = \max(y_1 - d_1, y_2 - d_2, y_3 - d_3)$$

$$y_{max} = \min(y_1 + d_1, y_2 + d_2, y_3 + d_3)$$

Donde  $x_j$  e  $y_j$  con  $j = 1, 2, 3$  son las coordenadas de los nodos fijos y  $d_j$  con  $j = 1, 2, 3$  son las distancias estimadas desde el nodo móvil hasta los respectivos nodos fijos. Así, las coordenadas del nodo móvil  $(x_t, y_t)$  se calcula según las *ecuaciones 2.6 y 2.7* respectivamente.

$$x_t = \frac{(x_{min} + x_{max})}{2} \quad (2.6)$$

$$y_t = \frac{(y_{min} + y_{max})}{2} \quad (2.7)$$

El problema que representa esta técnica es que dependiendo de los errores que tengamos en el cálculo de las distancias es posible que las áreas de dichos cuadrados no solapen, por lo que no se podrá calcular la posición del objeto.

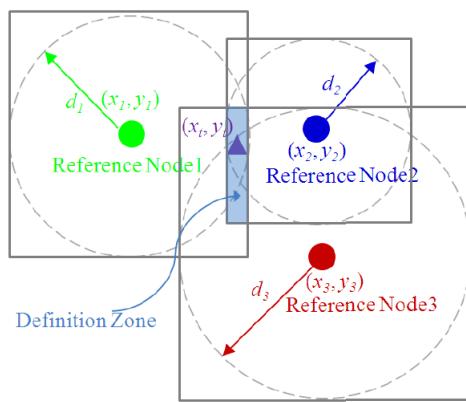


Figura 2.8: Técnica Min-Max

Fuente: [17]

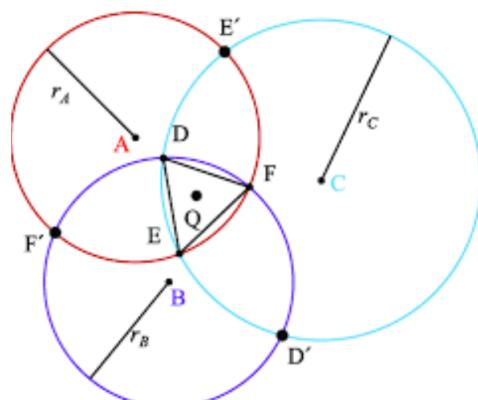


Figura 2.9: Técnica de la centroide

Fuente: [6]

#### ■ Técnica de la Centroide

Esta técnica se ha desarrollado durante la elaboración de este trabajo y surgió debido a la geometría presente cuando se produce el cruce de tres circunferencias.

Si dos circunferencias presentan seis puntos de cruce en común, se formará un área justo en el centro de las tres circunferencias.

Ese área bien se puede formar por el solapamiento de las tres circunferencias, o bien se puede formar debido al hueco que se forma entre ellas, véase la *figura 2.9*, en la que se muestra uno de los casos que pueden darse por el cruce de las circunferencias.

En ambas situaciones se da la posibilidad de dibujar un triángulo inscrito en dicho área, por lo que la localización consistiría en ubicar al objeto como el baricentro o centroide de dicho triángulo.

# Capítulo 3

## Técnicas de localización propuestas

Habiendo adquirido los conceptos teóricos necesarios para entender el funcionamiento de un sistema *ILS*, comenzamos con este capítulo el aspecto práctico del *TFG*.

Resaltamos que se ha decidido estudiar las técnicas de localización empleando la *RSSI* de la señal como medida. Veremos los problemas que puedan surgir al intentar ubicar un objeto de posición desconocida usando la distancia estimada desde el mismo hacia los nodos fijos. Para ello se han diseñado varios *scripts* de *MatLab* que nos ayudarán a determinar el impacto que tiene equivocarse en el cálculo de dicha distancia.

Vimos que la correcta estimación de las pérdidas que sufre la señal con su paso por el canal es un campo de estudio de elevado interés, como lo demuestran las recientes publicaciones académicas que tratan sobre la implementación de sistemas *ILS*.

En la "sección 2.3" vimos que el principal objetivo de los artículos publicados es obtener valores precisos para el coeficiente de atenuación del canal  $\alpha$  con el desarrollo de nuevas técnicas. Luego realizan experimentos comparando y analizando los resultados obtenidos.

Es importante acentuar que el enfoque de este trabajo es determinar las consecuencias de tener una mala estimación de  $\alpha$ , no tratamos de estimar un valor para dicho coeficiente. En este capítulo se analizan los errores que pueden haber en la localización en función del error que se comete al asumir una atenuación determinada de la señal.

Nuestro objetivo es cuestionar la necesidad de desarrollar técnicas y sistemas complejos para conocer el canal adecuadamente. ¿ Con qué precisión es necesario conocer el canal para obtener unas prestaciones determinadas del sistema de localización ?

### 3.1. Parámetros considerados

La determinación de la distancia se hará siguiendo el modelo matemático simplificado de la atenuación de la señal visto en la *ecuación 2.2*.

$$P_r = P_t K \left(\frac{1}{d}\right)^\alpha$$

Para determinar la distancia entre un emisor y un receptor se han de conocer todas las variables de la ecuación mencionada, así, la única variable desconocida de la ecuación es la  $d$ . Se ha comentado que la constante  $K$  es un valor en el cual se incluyen los efectos provocados por las antenas y otras características eléctricas de los dispositivos emisores y receptores.

Cómo en este trabajo no se tratará de determinar ninguna tecnología en concreto como la más adecuada, sin más bien entender que sucede si hay una variación en la atenuación usada para hacer los cálculos con respecto a la atenuación real del ambiente, supondremos algunas idealizaciones que excluirán el uso de determinados factores habituales en investigaciones de este ámbito, incluyendo todos los factores que afecten a la constante  $K$ .

Como es una constante, se podría determinar  $K$  de manera experimental y esta no variará con el incremento de la distancia. En los cálculos que realizaremos en los algoritmos planteados en la secciones posteriores supondremos que el valor de esta constante es unitario,  $K = 1$ , para así apreciaremos mejor los efectos de un error entre el valor de  $\alpha$  usado para los cálculos y el valor real.

Con esta suposición, el modelo matemático quedará simplificando de la manera indicada por la *ecuación 3.1*, que será la que empleada para los cálculos de las distancias en las simulaciones realizadas en los *capítulos 3 y 4*.

$$P_r = P_t \left(\frac{1}{d}\right)^\alpha = P_t d^{-\alpha} \quad (3.1)$$

Consideraremos que todos los factores que producen una atenuación de la señal están incluidas en el valor del coeficiente de atenuación. En cuanto al valor de este coeficiente, para poder determinar el impacto de una asignación errónea se le dará un valor fijo, al que llamaremos  $\alpha_s$  puesto que es un valor supuesto, en contraste con el valor real  $\alpha$  que es el valor con el cual se atenuará la señal en la realidad.

Para la elección de un valor de  $\alpha_s$  nos basamos en el trabajo realizado por [14], donde se trata de estimar valores para este coeficiente en varios ambientes. La *figura 3.1* se muestra los resultados obtenidos experimentalmente por el equipo de investigación en distintos canales, esto es a distintas frecuencias, para la atenuación en el espacio libre. Se aprecia que de manera general el valor de  $\alpha$  incrementa conforme aumenta la distancia entre el emisor y el receptor, pasando por un periodo de inflexión en el cual las curvas oscilan suavemente entre los dos y los nueve metros de distancia.

Según la evolución de las gráficas de la figura asignaremos un valor de dos unidades para el coeficiente  $\alpha_s$ , puesto que es una valor medio válido para todos los canales. Así, en las simulaciones realizadas en este trabajo, se considerará que la señal decrece de forma cuadrática con la distancia por lo que el coeficiente de atenuación supuesto será  $\alpha_s = 2$ .

A parte de los valores vistos en la *ecuación 3.1*, existen más factores que se han de tener en cuenta al realizar un sistema de localización en interiores. Para determinar cuales son consultamos el manual de uso o características de un sistema emisor/receptor, en concreto se trata de la ficha de especificaciones de la placa Digi XBee S2C que pueden ver en el

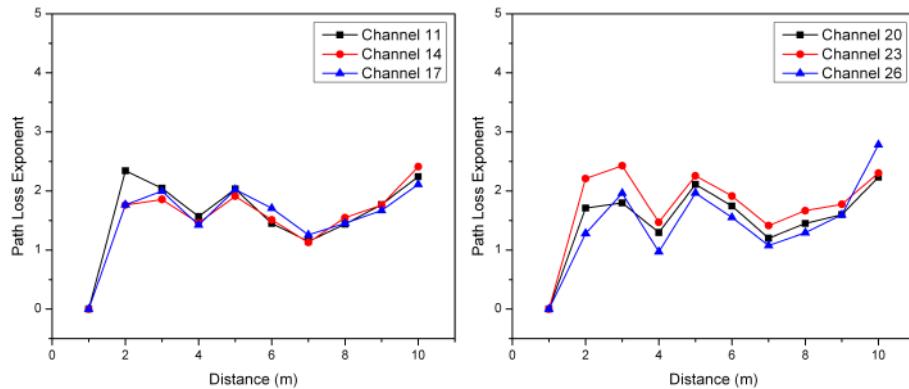


Figura 3.1: Coeficiente de atenuación según la distancia entre emisor y receptor, para distintas frecuencias

Fuente: [14]

anexo B. De todas las características incluidas, las siguientes afectan directamente a la transmisión/recepción:

- Velocidad de Transmisión
- Rango de alcance en Interiores y Exteriores Éste se encuentra marcado con un asterisco donde podemos leer: *El rango que figura es estimado. El rango real variará en función de la potencia de alimentación, orientación del transmisor y receptor, altura de las antenas, fuentes de interferencia y objetos entre el transmisor y receptor incluyendo estructuras internas y externas tales como paredes, árboles, edificios y demás.*
- Potencia de transmisión
- Sensibilidad del receptor

El alcance de la comunicación viene determinado por la máxima distancia que es capaz de recorrer la señal emitida antes de llegar al receptor con una intensidad superior a la sensibilidad del mismo. Hay dos factores fundamentales que afecten a dicha distancia, la atenuación que se producen por el camino *reducen la intensidad de la señal* y las interferencias que pueda haber *modifican la amplitud de la señal complicando su de-codificación*. Vimos en el *capítulo 2* que el tratamiento de las interferencias es un tema de gran envergadura que requiere un estudio por separado por lo que en el presente documento no se considerarán la existencias de las mismas.

La velocidad de transmisión de los datos es un factor que afecta directamente a la implementación de un sistema de localización *ILS* puesto que determina cuantas veces por unidad de tiempo será posible refrescar la posición de un nodo móvil. Sin embargo, esto es función de la tecnología empleada, por lo que dicho aspecto técnico no se considerará a lo largo de este trabajo. Se asumirá que la velocidad de transmisión es suficiente para que no afecte al proceso de estimación de la distancia.

En cuanto a la sensibilidad del receptor, este valor indica el umbral mínimo que debe de tener la señal para poder ser analizada por el receptor. Afecta directamente al rango de alcance puesto que si la sensibilidad es mala, la señal no puede sufrir una gran atenuación para seguir siendo decodificable, por lo que la distancia que puede recorrer es menor. En este trabajo deseamos ver el impacto de la atenuación en función de la distancia, por lo

que supondremos que todos los nodos son capaces de detectar y decodificar la señal.

En este trabajo se pretende determinar la relación entre la distancia emisor-receptor y un error en la atenuación de la señal, es por ello que se considerará una potencia de transmisión lo suficientemente grande para asegurar que se puedan realizar cálculos incluso con los nodos más lejanos.

Dicho de otra forma, suponemos que la señal presenta una potencia lo suficientemente grande para llegar hasta el nodo más lejano indiferentemente de cual sea la atenuación.

## 3.2. Descripción de los algoritmos usados

En esta sección expondremos el funcionamiento lógico de los algoritmos de una manera gráfica y simple de entender, evitando en lo posible la introducción de código en las explicaciones. Visite los anexos para visualizar los "scripts" completos. Aparte del conocimiento de los algoritmos empleados, otro objetivo que se pretende con este apartado es transmitir al lector las dudas a resolver y el objetivo de haber desarrollado los algoritmos.

En este *TFG* se han empleado tres "scripts" distintos para la realización de las simulaciones, cada uno de ellos diseñado para proporcionar respuestas a las dudas que se plantean. Este capítulo tan solo conlleva el uso de dos de ellos, *ErrorsVsAlpha* y *ErrorsPerDensity* cuyo código completo pueden ver en el anexo *anexo a*, aquí procederemos a explicar su estructura y funcionamiento general.

- *ErrorsVsAlpha*

Este *script* se ha desarrollado para determinar el efecto de una estimación errónea en la atenuación de la señal al recorrer el canal. Se compone de dos partes, una para determinar los efectos que tiene un  $\alpha_s \neq \alpha$  en la distancia calculada desde el emisor al receptor y otra parte para determinar cuál es el efecto que tiene en la posición estimada. Las *figuras 3.2 y 3.3* son una ilustración del funcionamiento de este *script*. Se explicarán todas las funciones que aparecen y de las cuales se hacen uso justo después.

En la *figura 3.2* vemos la estrategia usada para determinar el efecto que tiene sobre la distancia el cometer un error al suponer  $\alpha_s$ .

Se asumen dos posiciones conocidas en el espacio *KnownPositionA* que sería el emisor y *KnownPositionB* que simularía al receptor. Se determina la distancia real entre estos para poder calcular así la diferencia entre la distancia real  $d$  y la calculada usando la información de RSSI  $d_e$ . Se define el array  $\epsilon = [0, 0,1, 0,2, \dots, 1,4, 1,5]$  cuyos valores serán los errores que supondremos que se cometan al usar  $\alpha_s$ . Es decir, el coeficiente  $\alpha$  real, el que atenúa a la señal, será  $\alpha = \alpha_s(1 \pm \epsilon)$ . Se usa un bucle para poder determinar el error cometido al calcular  $d_e$  para cada uno de los valores del vector  $\epsilon$ . Dentro del bucle se calcula el coeficiente  $\alpha$  real  $\alpha = \alpha_s(1 \pm \epsilon)$  y se calcula la potencia de la señal que llegaría al emisor usando la *ecuación 3.1* y el valor  $\alpha$ . Tras obtener el valor de  $P_r$  volvemos a aplicar la *ecuación 3.1* para obtener esta vez el valor de  $d_e$ , usando el coeficiente  $\alpha_s$ . Como el coeficiente  $\alpha$  usado al calcular la atenuación de la señal, es distinto al valor del coeficiente  $\alpha_s$  usado al calcular la distancia  $d_e$ , ésta última será distinta a la distancia real existente  $d$ , debido al error  $\epsilon$ . Así, el último paso consistiría en determinar el error cometido en el cálculo de la distancia, comparando  $d$  con  $d_e$ .

La *figura 3.3* muestra la estrategia usada para determinar el efecto de la aplicación de  $\epsilon$  en la posición que se estima para el objeto.

En este caso se han de asumir cuatro posiciones conocidas en el plano, que corresponden con los tres nodos fijos *FixedNode1* ... *3* y la propia posición del objeto, para

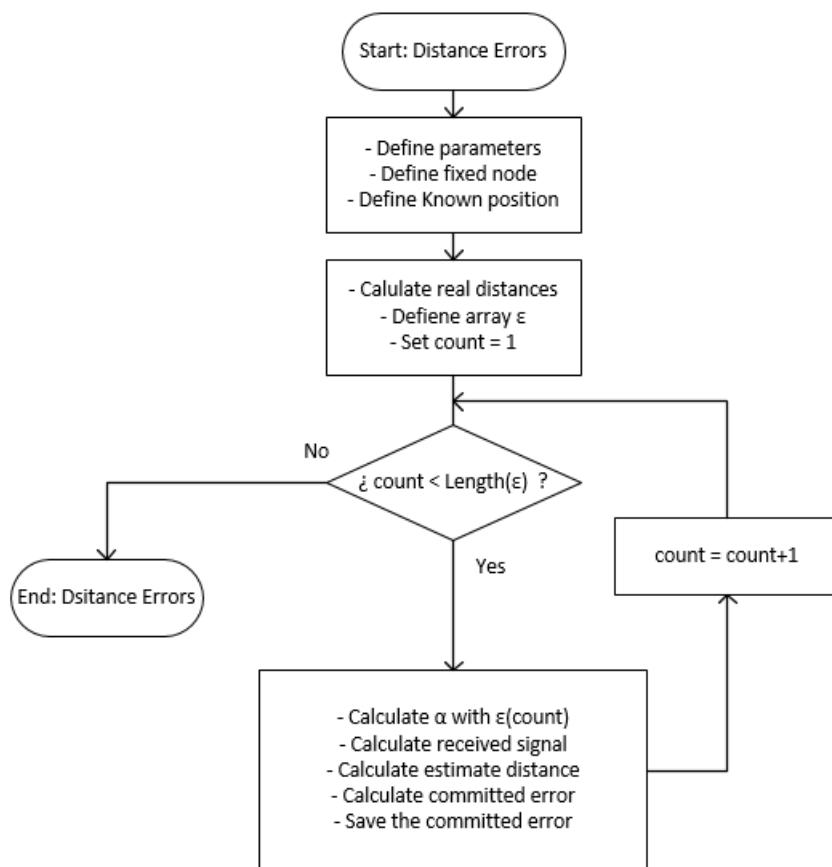


Figura 3.2: Determinar error en la estimación de la distancia



Figura 3.3: Determinar error en estimación de la posición

poder comparar la posición estimada con la real.

Se calcula la distancia real existente desde el objeto a cada uno de los nodos fijos y se dejan los mismos valores de error definidos por el vector  $\epsilon$ , por lo tanto, se vuelve a emplear un bucle que haga una iteración por cada valor del vector. Tras calcular  $\alpha$ , se computa la señal que llegaría a cada uno de los nodos fijos y se calcula la distancia estimada  $d_e$  a cada uno de ellos.

Tras ello se implementan los métodos de localización vistos en la capítulo dos, esto se hace llamando a la función *CalculateLocalization*. Más adelante hablaremos sobre el algoritmo de localización y cómo se ha implementado el mismo. Por último, se calcula el error que se ha producido en la estimación de la posición como la distancia en línea recta desde la posición real hasta la posición estimada.

#### ■ *ErrorsPerDensity*

Este *script* se ha diseñado para determinar el efecto de variar la cantidad de nodos fijos por unidad de área. Dentro del mismo se realizan dos operaciones importantes, obtener los errores cometidos al estimar la posición del objeto en función de la cantidad de nodos fijos que haya en un mismo área y acotar dicho error para una cantidad de nodos establecida. Las *figuras 3.4 y 3.5* muestra la estructura general de este *script*.

En la *figura 3.4* vemos la técnica empleada para computar los errores que se producen en un mapa de dimensiones pre-establecidas en función de la cantidad de nodos fijos equidistantes existentes. Tras introducir las dimensiones del plano se define el array *NumberOfFixedPoints* cuyos valores representan la cantidad de nodos fijos equidistantes que se desean tener en el mapa. Usaremos un bucle para calcular los errores en función de cada valor de dicho array.

Para que los errores obtenidos tengan validez se ha de realizar la simulación para varias posiciones del mapa. La constante *NumberOfTrackingPoints* define el número de puntos que ubicaremos en el mapa para proceder a estimar su posición y luego calcular el error. La constante *minAllowableDistance* define la distancia mínima que debe haber entre los puntos y los nodos fijos, para distribuirlos por todo el área del mapa.

En cada iteración del bucle principal se generan los nodos fijos equidistantes usando la función *GenerateEquidistantPoints* y se generan puntos aleatoriamente distribuidos por el mapa usando la función *GenerateTrackingPoints*. Estos últimos se consideran las posiciones de los nodos móviles que debemos localizar, así que entramos en un bucle secundario en el cual se calculan las posiciones estimadas de cada uno de los puntos distribuidos aleatoriamente. Se emplea la función *GetClosestPoints* y se aplica el algoritmo de localización con la función *CalculateLocalization*.

El objetivo final es determinar los errores en función de la densidad de puntos en el área, así que se usa la función *CalculateErrorsPerDensity* para determinar el error máximo, medio y mínimo en la estimación de la posición de los puntos aleatorios.

Tras esto, se vuelve repetir el proceso haciendo una nueva iteración del bucle principal incrementando la cantidad de nodos fijos equidistantes en el plano.

Un aspecto importante de este algoritmo reside en la capacidad de la función *GetClosestPoints* de implementar un error  $\epsilon$  en el coeficiente de atenuación  $\alpha = \alpha_s(1 \pm \epsilon)$ . Dicho valor de  $\epsilon$ , como veremos en la descripción de la función, será aleatorio dentro de un rango determinado, consiguiendo de esta forma determinar los errores en la posición en función de la densidad de nodos suponiendo que  $\alpha_s$  varía un tanto por

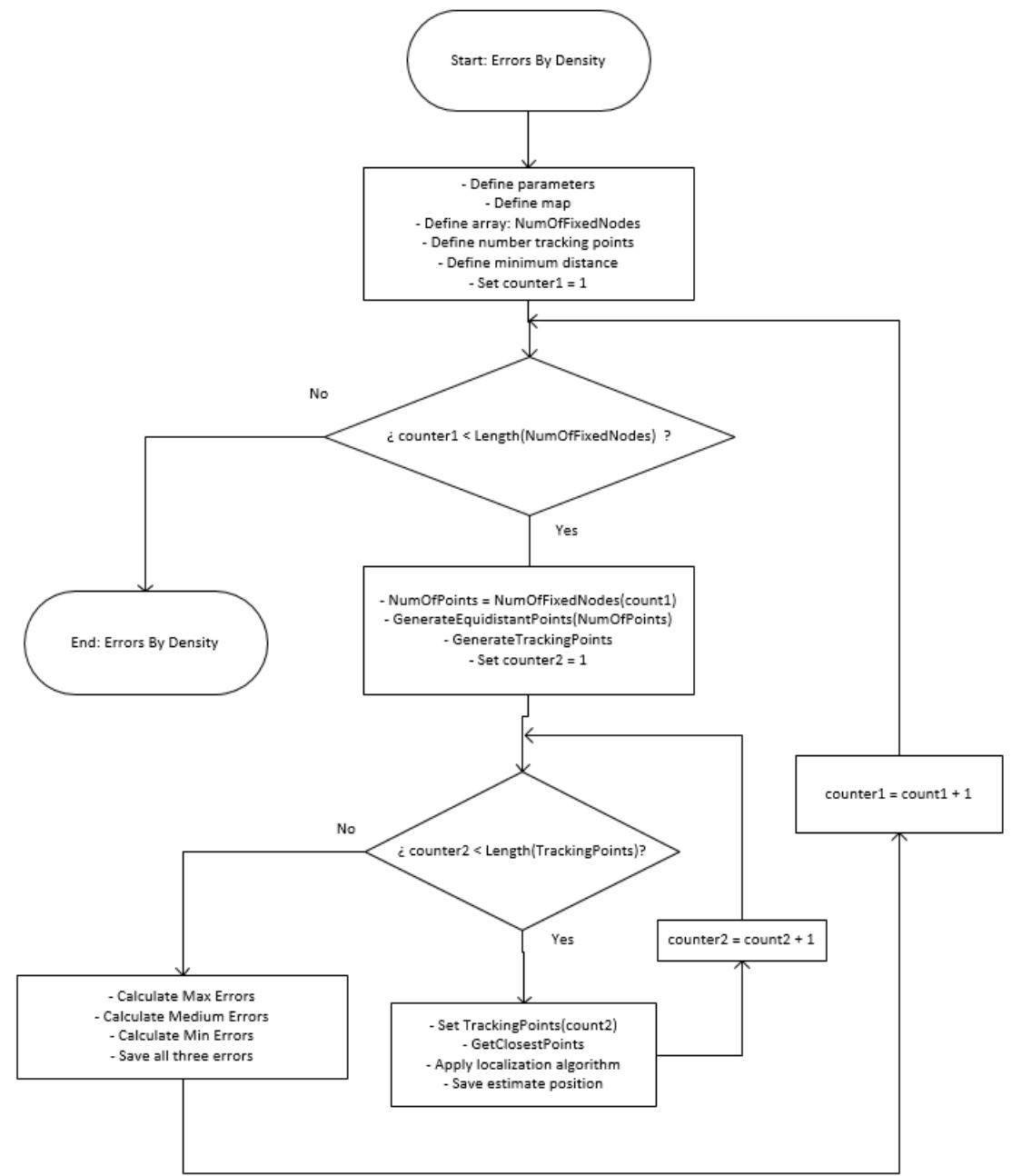


Figura 3.4: Determinar errores en función de la densidad de nodos

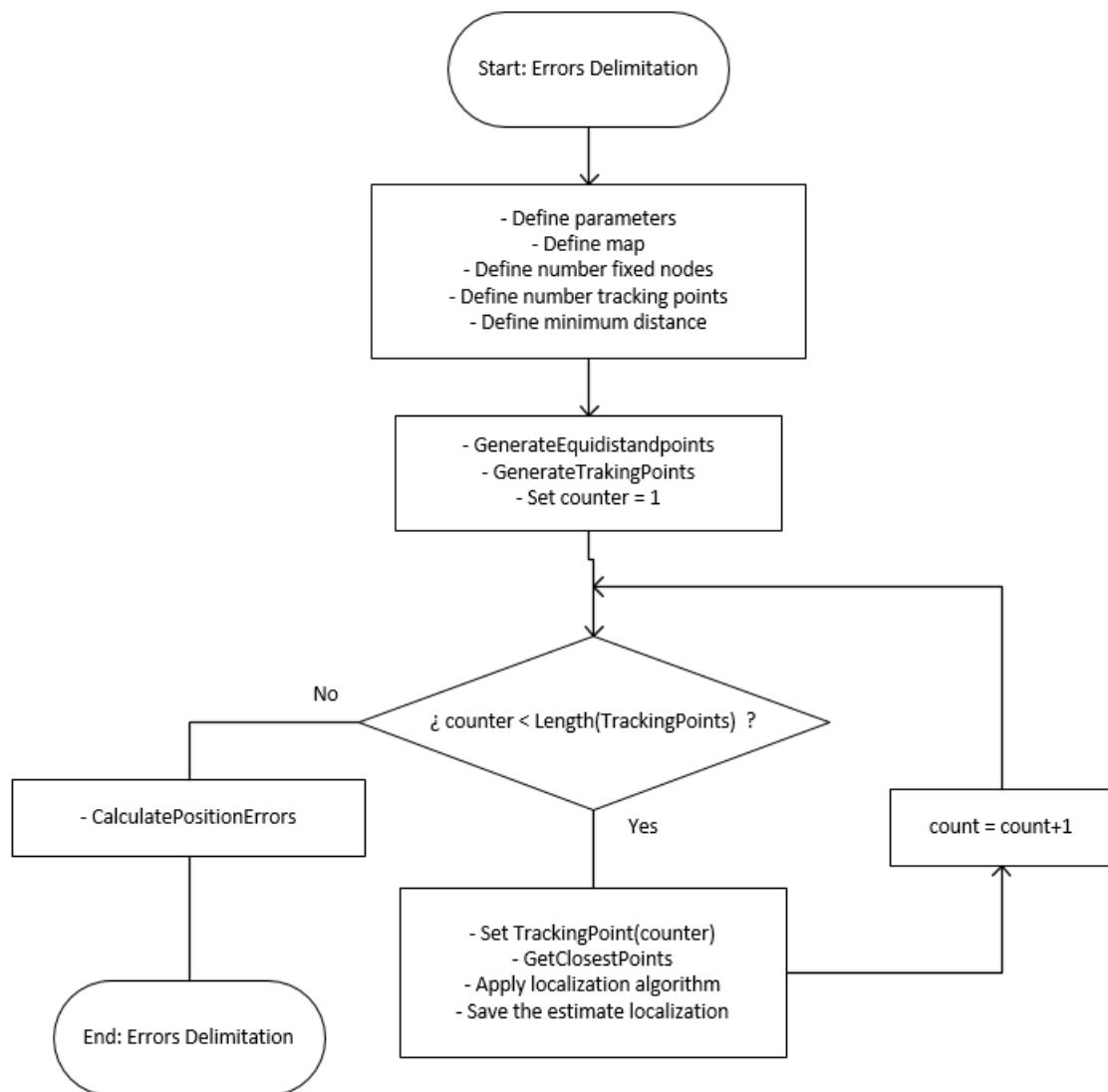


Figura 3.5: Delimitar errores para una densidad de nodos específica

ciento establecido.

La figura 3.5) muestra la estrategia usada para la delimitación del error. La diferencia reside en que se usa una cantidad constante de nodos fijos equidistantes, por lo que tan solo se emplea un bucle en el cual se estima la posición de cada uno de los puntos aleatorios generados.

Tras esto se computan los errores cometidos al estimar la posición de cada nodo móvil, haciendo uso de la función *CalculatePositionsErrors*.

Cada uno de estos *scripts* se apoyan en llamadas a varias funciones auxiliares, las cuales implementan las funcionalidades básicas como el algoritmo de localización, por ejemplo. La utilidad de estas funciones se puede encuadrar en cuatro categorías, *la generación de nodos, el cálculo de distancias, la estimación de la posición y el cálculo de errores*. Tras haber visto el funcionamiento principal de los algoritmos, procedamos viendo la estructura lógica de las funciones auxiliares que se usan. Gran parte de las funciones vistas en los *scripts* hacen uso de otras funciones adicionales. Estas también se definirán a continuación. Se definirán en función de la categoría y seguiremos un orden que facilite la compresión del lector acompañados de esquemas gráficos cuando así sea necesario.

Para favorecer el entendimiento de las funciones, estas se nombrarán indicando los valores que estas requieren como entrada y los valores que devuelve. Sin embargo, es importante clarificar que una salida/entrada nombrada en la explicación se puede equiparar con varias variables en el código. Esto se hace por simplificación, por ejemplo, al indicar como entrada *DimensionesMapa*, esto se corresponde a cuatro valores distintos (*x-mínimo, x-máximo, y-mínimo, y-máximo*) en el código.

La implementación de cada función descrita a continuación se puede visitar en el *anexo A*.

#### ▪ Generación de nodos

- **GenerateEquidistantPoints**

Entradas: *DimensionesMapa*, *NúmeroDePuntos*

Salidas: *NodosFijosX* *NodosFijosY*

Esta función genera puntos uniformemente distribuidos en un mapa de dos dimensiones devolviendo dos vectores, uno con las coordenadas de los puntos en el eje de abscisas *NodosFijosX* y el otro con las coordenadas en el eje de ordenadas *NodosFijosY*. Como parámetros de entrada se le han de pasar las dimensiones del mapa en el cual se desea generar los puntos y el número de puntos que se requieren.

Consiste en generar una cuadricula sobre el mapa con el mismo número de cuadrados que el numero de puntos a generar para luego colocar un punto en el centro de cada cuadrado.

De todos los divisores de *NúmeroDePuntos* se selecciona el intermedio, o el inmediatamente por encima en caso de que la cantidad de divisores sea par. Se realizan tantas separaciones equiespaciadas en el eje x como el divisor seleccionado y se realizan tantas separaciones equiespaciadas en el eje y de manera que al multiplicar ambas cantidades de separaciones se obtenga la cantidad de

puntos que se desea generar. Por ejemplo, si deseamos colocar 20 nodos fijos, como la cantidad de divisores es par [1 2 4 5 10 20] se seleccionará el 5 como el inmediatamente por encima de la mitad. Así, se realizarán 5 separaciones en el eje x, y 4 separaciones en el eje y puesto que  $4 \cdot 5 = 20$

Por último, se generan los vectores con las coordenadas  $x, y$  de los nodos fijos, considerando que cada nodo se colocará en el punto intermedio de cada cuadrado de la cuadrícula.

- **GenerateTrackingPoints**

Entradas: DimensionesMapa, PosicionesPuntosFijos, NúmeroDePuntos, DistanciaMínima

Salidas: PuntosX, PuntosY

Esta función genera puntos aleatoriamente distribuidos en el plano. Esos puntos se consideran los nodos móviles del sistema *ILS* que procederemos a estimar su posición.

Como salida devuelve dos vectores, uno con las coordenadas de los puntos en el eje x y el otro con las coordenadas de los puntos eje y. Como entrada, aparte de las dimensiones del mapa y del número de puntos que se desea generar, se requiere indicar una distancia mínima a mantener entre los puntos en todo momento, es por ello que también se le indica la ubicación de los nodos fijos.

Su lógica es muy simple, se emplean dos funciones *RandomGeneratorX* y *RandomGeneratorY* para generar las coordenadas aleatorias de un punto dentro del rango determinado por las dimensiones del mapa. Tras la generación de este punto se comprueba que la distancia con los demás puntos generados y con los nodos fijos es superior a la distancia mínima indicada.

Si se supera esta condición se introducirán las coordenadas del punto aleatorio en los vectores de salida *PuntosX*, *PuntosY*. En caso de que no se supere la condición, se genera otro punto aleatorio, así hasta que se generen tantos puntos como los indicados por *NúmeroDePuntos*.

- **Calculo de distancias**

- **GetDistancesToFixedPoints**

Entradas: NodosFijosX, NodosFijosY, PosicionX, PosiciónY

Salidas: Distancias

Esta función calcula la distancia que hay desde un punto en el mapa hacia todos los nodos fijos existentes.

Como entrada requiere las coordenadas de los nodos fijos y las coordenadas del nodo móvil *PosiciónX* y *PosiciónY* para el cual se quieren calcular las distancias. Como salida esta función devuelve un vector con todas las distancias calculadas.

Su lógica consiste en aplicar la ley de Pitágoras, como pueden ver en la para cada uno de los puntos fijos, así, si las coordenadas del nodo fijo en cuestión son  $(x_f, y_f)$  y las coordenadas del nodo móvil son  $(x_m, y_m)$  la distancia existente entre estos dos será  $d$ :

$$d = \sqrt{(x_f - x_m)^2 + (y_f - y_m)^2}$$

- GetClosestPoints

Entradas: Parametros, PosicionesNodosFijos, PosiciónObjeto

Salidas: NodosCercanos, DistanciasReales, DistanciasEstimadas

Esta función es uno de los grandes pilares de este trabajo. Se encarga de determinar cuales son los tres nodos fijos más cercanos al objeto y devuelve el valor de las distancias estimadas entre dichos nodos y el objeto usando el coeficiente de atenuación  $\alpha_s$ . Se usa el modelo matemático de la *ecuación 3.1* para calcular las distancias.

Como salida devuelve dos vectores que contienen las coordenadas de los tres puntos más cercanos al objeto, un vector que contiene las distancias estimadas entre cada uno de los tres puntos fijos y el objeto a localizar y adicionalmente devuelve un vector con la distancia exacta, para posibilitar el cálculo de los errores.

Como parámetros de entrada se le han de pasar la intensidad de señal emitida, el factor de atenuación  $\alpha_s$ , la posición de todos los nodos fijos y las coordenadas del objeto que deseamos localizar.

En caso de que se conozca exactamente la atenuación de la señal, es decir  $\alpha = \alpha_s$ , la distancia estimada será igual a la existente. En caso contrario, la exactitud de la distancia obtenida está directamente relacionada con la diferencia que hay entre  $\alpha$  y  $\alpha_s$ . Si  $\alpha > \alpha_s$  la señal sufrirá una mayor atenuación de lo que se prevé, así, la distancia estimada será superior a la distancia real. En el caso contrario, si  $\alpha < \alpha_s$  la distancia estimada será inferior a la distancia real.

Para simular un coeficiente  $\alpha_s \neq \alpha$  se vuelve a usar un error  $\epsilon$  al que se le asignará un número aleatorio dentro de un rango  $\epsilon = aleatorio[minimo, maximo]$ , así que se podrá calcular la potencia recibida usando el coeficiente  $\alpha = \alpha_s(1 \pm \epsilon)$  en la *ecuación 3.1*.

Tras realizar dicho cálculo para todos los nodos fijos existentes, se seleccionan aquellos tres para cuales la potencia de la señal es mayor, siendo esos los considerados como los más cercanos. Para computar la distancia estimada se vuelve a usar la *ecuación 3.1* pero esta vez despejando  $d$  y usando los valores de  $P_t$  calculado antes y para el coeficiente de atenuación el valor de  $\alpha_s$ .

De esta forma la distancia obtenida al resolver la ecuación sera distinta de la distancia real existente en función de la diferencia que haya entre  $\alpha$  y  $\alpha_s$ . Recordemos que esta diferencia se debe a la aplicación de  $\epsilon$ , que varía aleatoriamente dentro de un rango, así que podremos relacionar los errores que se producen en la posición en función del porcentaje de variación de  $\alpha_s$ . Vean la *figura 3.6* como aclaración del funcionamiento de esta función.

Figura 3.6: Estructura lógica de *GetClosestPoints*

- Estimación de la posición

- CalculateLocalizationTheory

Entradas: NodosCercanos, Distancias

Salidas: PosiciónObjeto

Las funciones denominadas CalculateLocalization... serán las encargadas de determinar la ubicación del objeto en función de los parámetros que reciban por entrada. En este caso, se le ha añadido la terminación *Theory* para indicar que esta función es una implementación directa de las técnicas de localización comentadas en el *capítulo 2*. En concreto se calcula la ubicación del objeto usando tres de los métodos descritos , trilateración, método max min y triangularización.

Como entradas necesita conocer la posición de los tres nodos más cercanos y las distancias hacia estos. Como salida nos devuelve las coordenadas de la ubicación estimada para el objeto. La aplicación de los tres métodos se realiza de forma gradual, en función de la precisión que podamos obtener.

Primero se forman las tres circunferencias de centro los nodos fijos y radio la distancia estimada hasta el objeto y se calculan los puntos de cruce entre ellas. Si hay alguna circunferencia que no cruza con las otras, no queda otra opción que realizar triangularización, es decir, dibujar un triángulo cuyos vértices son los puntos fijos y determinar como posición del objeto la centroide o baricentro de dicho triángulo.

Si todas las circunferencias cruzan entre ellas se intenta determinar la existencia de un punto de cruce común entre las tres. Para ello se compara el radio de una de las circunferencias con la distancia entre el centro de la misma y los dos puntos de corte entre las otras dos. Si una de dichas distancias coincide con el radio de la circunferencia, permitiendo una pequeña tolerancia, entonces se considera que tenemos un punto de cruce entre las tres circunferencias por lo que dicho punto será el de la ubicación del objeto a localizar.

Por último, si todas las circunferencias se cruzan entre sí, pero no hay un punto de cruce común a las tres claro porque no se cumple con la condición de desviación mínima admisible, se emplea el método Max-Min que consiste en determinar el centro del área que se forma por el solapamiento de los tres cuadrados en los que se encuentra incrustada la circunferencia.

- MakeCirclesCross

Entradas: Circunferencias

Salidas: Circunferencias

Como se ha visto en el capítulo dos, usar triangularización tiene el inconveniente de que para todo el área del mapa comprendida por el triángulo se estima la misma posición para el objeto, el baricentro, lo que afecta a la precisión. Esta función se ha desarrollado para evitar realizar triangularización.

Consiste en comprobar si las circunferencias formadas por los nodos fijos y las distancias estimadas se cruzan entre sí, e incrementar el radio de las circunferencias en incrementos muy pequeños en caso contrario hasta conseguir que

todas se crucen entre sí.

Como parámetros de entrada se le pasan directamente las circunferencias en forma de vectores, cada vector conteniendo las coordenadas del centro de la circunferencia y el radio de esta. Esta función modifica los radios de las circunferencias si es necesario y devuelve los vectores modificados.

Para el incremento de los radios se sigue un algoritmo diseñado para obtener la mayor precisión posible al incrementar los radios, el cual se basa en tres pasos:

- Paso 1: Si ninguna circunferencia se cruza y mientras esto se siga cumpliendo, se incrementa el radio de la circunferencia más pequeña en todo momento hasta conseguir que al menos dos de las circunferencias se crucen. Al aumentando el radio de la más pequeña tratamos de minimizar el impacto.
- Paso 2: En caso de que sólo haya dos circunferencias que se crucen, determinar cuales son e incrementar el radio de la que no se cruza con ninguna hasta que se cruce con al menos una de las otras dos. Al igual que antes, se trata de minimizar el impacto variando el radio de la circunferencia que no cruza con ninguna.
- Paso 3: Asumiendo que al menos dos circunferencias cruzan entre sí y que la tercera cruza con al menos una de esas dos, tan solo queda la posibilidad de que haya dos circunferencias que no crucen entre sí por lo que se ha de determinar cuales son esas dos y se incrementa el radio de la más pequeña.

Una vez finalizada la implementación del algoritmo obtenemos las tres circunferencias modificadas de manera que siempre se cruzarán entre sí, por lo que se puede proceder a estimar la posición sin necesidad de realizar triangularización.

#### • CalculateLocalizationMaxMin

Entradas: Circunferencias

Salidas: PosiciónObjeto

Esta función calcula la ubicación del objeto implementando el método Max-Min directamente en el caso de que no se considere la existencia de un punto de cruce común entre las tres circunferencias.

Al aplicar este método de localización no se tiene en cuenta la posibilidad de que no exista un solapamiento de área, así que se ha de usar la función *MakeCirclesCross* previamente para asegurar el cruce de las circunferencias entre sí.

#### • CalculateLocalizationCentroid

Entradas: Circunferencias

Salidas: PosiciónObjeto

A diferencia de la función anterior, esta función aplica el método de la centroide en caso de que no exista un punto de cruce común.

Tampoco se tiene en cuenta la no existencia de todos los puntos de cruce entre las circunferencias, imposibilitando la formación de los triángulos, así que se ha

de usar la función *MakeCirclesCross* previamente para asegurar una correcta implementación de esta función.

- **CalculateLocalization**

Entradas: Circunferencias

Salidas: PosiciónObjeto

Esta función es la que implementa el método de localización por preferencia. Se trata de una combinación entre las funciones *CalculateLocalizationMaxMin* y *CalculateLocalizationCentroid*. Prácticamente lo que hace es usar el método Min-Max para ciertas situaciones y el método de la centroide para otras. En la siguiente sección, cuando veamos los resultados, entenderemos la motivación que tuvimos por la cual se ha decidido desarrollar esta función.

La decisión entre usar uno de los métodos o el otro se hace en base a la cantidad que se incrementan las circunferencias. Se define un incremento umbral para el radio de las circunferencias y se compara con el incremento que se ha practicado.

Si el radio de alguna de las circunferencias ha sufrido un incremento superior a dicho umbral se implementa el método Max-Min. Si el incremento es menor que dicho umbral se implementa el método de la centroide para realizar la ubicación.

Veremos en las simulaciones que el método de la centroide ofrece mejores resultados cuando las circunferencias se encuentran próximas, sin embargo, si hay alguna muy lejana presenta mejores resultados el método Max-Min. Es por ello que será ésta la función que se implementará cuando realicemos las simulaciones de los errores por densidad de nodos.

Por último, faltaría comentar que en caso de que todas las circunferencias se crucen pero no exista un punto de cruce común a las tres, se aplicará el método de la centroide.

- **GetRectanglesArea**

Entradas: Circunferencias

Salidas: Área

Esta función determina el área de solape que se forma debido al cruce de los rectángulos que tienen inscrita a las circunferencias. Así, como entrada recibe las tres circunferencias y devuelve el área de solape como salida.

Las cuatro esquinas de un cuadrado que circumscribe a una circunferencia de radio  $r$  y centro  $(x, y)$  vienen definidas por las coordenadas  $(x - r, y + r)$   $(x - r, y - r)$   $(x + r, y - r)$   $(x + r, y + r)$ .

Usando la función *polyshape* que viene implementada en *MatLab* se genera el cuadrado circunscrito de cada una de las circunferencias. Por último, se determina el área de solape entre los tres cuadrados usando la función *intersect* que también viene por defecto con el programa.

- **ApplyCenterOfArea**

Entradas: Área

Salidas: Posición Objeto

Esta función determina las coordenadas del punto central del área rectangular que devuelve la función anterior, por lo tanto estas funciones se han de usar en conjunto.

Como dicho área se formará por el solape de tres cuadrados, siempre tendrá una forma rectangular. Para determinar las coordenadas  $(x_c, y_c)$  del punto central del área de un rectángulo cuyos puntos extremos son  $(x_{min}, y_{max})$ ,  $(x_{min}, y_{min})$ ,  $(x_{max}, y_{min})$  y  $(x_{max}, y_{max})$ . Se aplican las ecuaciones:

$$x_c = x_{min} + \left( \frac{x_{max} - x_{min}}{2} \right)$$

$$y_c = y_{min} + \left( \frac{y_{max} - y_{min}}{2} \right)$$

- **ApplyTriangleMethod**

Entradas: Circunferencias

Salidas: Posición Objeto

Esta función es la que implementa el método de la centroide, es decir calcula la posición del objeto como el baricentro del triángulo de menor área de los formados por los seis puntos de cruce entre las tres circunferencias.

Para ello primero se forman todos los triángulos posibles usando la función *polyshape*, siendo el área una de las propiedades a las cuales podemos acceder, introducimos todos los valores en un array y comprobando que posición de éste se corresponde con el mínimo.

Por último, se calculan las coordenadas del baricentro del menor triángulo con la función *centroid* que viene implementada en *MatLab*

## ■ Cálculo de errores

- **CalculatePositionErrors**

Entradas: PosicionesReales, PosicionesEstimadas

Salidas: Errores Absolutos

Esta función calcula el error absoluto que hay entre la posición real del objeto y la posición estimada.

Tiene la característica de que se computa el error para varias posiciones a la vez, siendo útil para calcular el error a lo largo de una trayectoria, por ejemplo. Como parámetros de entrada recibe en vectores las coordenadas de las posiciones reales y las coordenadas de las posiciones estimadas. Esos vectores pueden tener dimensión uno, por lo que la función también es válida para calcular el error de tan sólo una posición. Devuelve un vector con los errores calculados para cada una de las posiciones.

Funciona aplicando el teorema de Pitágoras para cada posición, así, dada la

posición real del objeto  $(x_r, y_r)$  y la posición estimada  $(x_e, y_e)$  se calcula el error absoluto cometido conforme:

$$\text{Error} = \sqrt[2]{(x_r - x_e)^2 + (y_r - y_e)^2}$$

- **CalculateErrorsPerDensity**

Entradas: PosicionesReales, PosicionesEstimadas

Salidas: Error Mínimo, Máximo y Medio

Esta función se usa para calcular el error máximo, medio y mínimo para una serie de posiciones reales y estimadas que se pasen como entradas.

Para ello hace uso de la función anterior *CalculatePositionsErrors*, de hecho tiene los mismos parámetros de entrada. Tras determinar los errores absolutos cometidos para cada posición, computa el valor máximo, el valor medio y el valor mínimo de todos ellos. Son esos valores los que devuelve en variables separadas.

- **CalculateRelativeError**

Entradas: Valor Exacto, Errores Absolutos

Salidas: Errores Relativos

Esta función calcula los errores relativos cometidos, en tanto por ciento, respecto al valor preciso.

Como parámetros de entrada requiere los valores precisos y los errores absolutos. Conociendo dichos datos se calcula el error relativo con:

$$\text{Error}_{\text{relativo}} = \frac{\text{Error}_{\text{absoluto}}}{\text{Valor Preciso}} * 100$$

Habiendo visto ya el desarrollo de los algoritmos para realizar las simulaciones y el cómo se implementa la localización con el uso de las funciones explicadas, es momento de que veamos los resultados obtenidos.

### 3.3. Presentación de resultados

El primer paso que debemos dar para calcular la ubicación de un objeto es determinar la distancia hacia los nodos. Suponiendo conocidas la potencia emitida y la atenuación de la señal por el canal, tan solo debemos despejar la distancia en la **ecuación 3.1**. El principal problema reside en que la atenuación de la señal es dinámica como vemos en los experimentos realizados por [14], es por ello muy complicado elegir con precisión un valor de  $\alpha_s$ . Para solucionar este problema podríamos instalar un sistema adicional que esté calculando constantemente un valor para  $\alpha_s$ , sin embargo, esto incrementaría el coste del sistema *ILS* y aumentaría la necesidad energética.

En las simulaciones que se realizarán en esta sección englobaremos la incertidumbre asociada al canal dentro del parámetro  $\alpha$ , asumiendo que lo conocemos de manera imprecisa.

La primera prueba consiste en determinar cual es el error cometido en el cálculo de la distancia si la atenuación de la señal  $\alpha$  es inferior al valor considerado  $\alpha_s$ . A modo de ejemplo, colocamos dos puntos en el plano, el punto uno que será considerado el emisor colocado en la posición  $(x_1, y_1) = (5, 5)$  y el punto dos que se considerará el receptor colocado en la posición  $(x_2, y_2) = (20, 20)$ . De la misma manera, repetimos el experimento incrementando la distancia entre los dos puntos, colocando el punto dos en la posición  $(x_2, y_2) = (50, 50)$ . De esta forma es posible comparar los resultados cuando el emisor y receptor se encuentran cerca o lejos.

Usando el algoritmo para determinar el error en la distancia del *script Errors Vs Aplha*, calculamos el valor de  $\alpha = \alpha_s(1 - \epsilon)$  con  $\epsilon$  representando la diferencia entre  $\alpha$  y  $\alpha_s$  en %. Suponiendo que el emisor transmite una señal hacia el receptor, se calcula la atenuación que esta sufre usando el valor de  $\alpha$  para luego estimar la distancia entre los dos puntos usando el valor de  $\alpha_s$ . Esto se realiza para cada uno de los valores del vector  $\epsilon$ .

Como la distancia entre los dos puntos es conocida, en concreto de 15 metros, podremos calcular cual es el error absoluto y relativo que se ha cometido debido a que  $\alpha$  sea diferente a  $\alpha_s$ .

El error que se comete en la distancia será el que determine que las circunferencias usadas para la implementación del algoritmo de localización se crucen o no, es por ello de interés conocer qué sucede con dicho error si la distancia entre el emisor y el receptor incrementa. Representamos gráficamente tanto los errores absolutos como relativos en la *figura 3.7 a) y b)*.

Tal y como esperamos, el error cometido en la estimación de la distancia aumenta conforme incrementa la diferencia entre  $\alpha$  y  $\alpha_s$ . Un hecho interesante es ver que al pasar de un 100 % en  $\epsilon$  el error absoluto es prácticamente igual a la distancia a estimar. Viendo la gráfica del error relativo, *figura 3.7 b)*, se aprecia que conforme la diferencia entre coeficientes se aproxima al 100 %, también lo hacen los errores cometidos, sin embargo, en ningún momento se podrá superar el 100 % de error relativo. Esto quiere decir que el error máximo que podemos cometer, considerando una atenuación para la señal superior a la existente, equivalente a la distancia que hay entre el emisor y el receptor.

Esto adquiere sentido puesto que al considerar en todo momento una atenuación superior a la real, la distancia estimada será inferior a la existente. Así, como máximo podremos obtener en los cálculos una distancia igual a la existente y como por definición la distancia

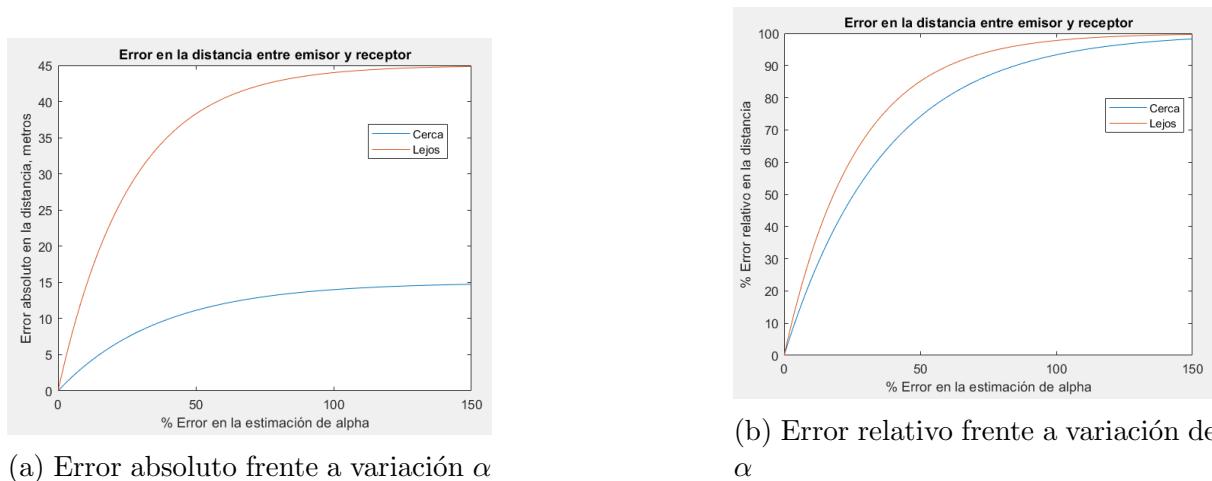


Figura 3.7: Error cometido al calcular la distancia entre emisor y receptor

no puede adquirir valores negativos, la mínima distancia que se podría estimar sería cero. Esto, aparte de indicar que el máximo error absoluto a cometer es igual a la distancia existente entre los dos puntos, confirma el hecho de que el error sea mayor conforme aumenta la distancia entre ellos.

Un factor que preocupa es el incremento exponencial que presenta la curva del error relativo. Analizando el trabajo realizado por [14], vemos que obtienen unas lecturas de  $\alpha$  comprendidas entre 1,8 y 1,95 para línea de visión directa, donde el valor teórico recomendado  $\alpha_s$  es de 2. Esto quiere decir que la diferencia entre ambos coeficientes es inferior al 10 %, lo que nos hace determinar que la zona de interés se encuentra para un  $\epsilon$  entre 0 % y 20 %.

En la *figura 3.8* se enfocan las gráficas anteriores para unos valores del *eje x* comprendidos en el el intervalo [0, 20] %. Vemos que el error relativo se triplica conforme  $\epsilon$  incrementa. Es decir, cometemos un error del 5 % al considerar  $\alpha_s$ , tendremos un 15 % de error en la distancia. Además, se aprecia que el error relativo es mayor en la situación de los nodos más alejados. Para un 10 % en  $\epsilon$ , tenemos un error relativo poco superior al 20 % si los puntos se encuentran cerca, mientra que el error relativo es del 30 % si los puntos están lejos uno de otro.

En la práctica, esto indica que las circunferencias que usaremos para realizar el algoritmo de localización podrán variar su radio en un 25 % respecto al valor real. Un dato preocupante.

Veamos ahora el caso de suponer un coeficiente  $\alpha_s$  inferior a  $\alpha$ . Es decir, repetimos el experimento anterior exactamente con los mismos datos, sin embargo esta vez aplicamos la ecuación  $\alpha = \alpha_s(1 + \epsilon)$ , nótese que esta vez se suma  $\epsilon$  en vez de restarlo.

A diferencia del caso anterior, esta vez no existiría un límite en el error cometido puesto que al ser  $\alpha$  superior a  $\alpha_s$ , la atenuación de la señal será mayor que la considerada, lo que indica que la distancia calculada será superior a la real. No existe un límite por encima que indique un valor máximo de la distancia estimada.

En el caso práctico dicho límite lo representarían las características del canal. Teóricamente el coeficiente  $\alpha$  podría ser infinitamente superior al valor de  $\alpha_s$ , es decir, se podrían hacer los cálculos para un coeficiente  $\alpha$  cada vez más grande, la distancia estimada incrementaría continuamente, por lo que el error incrementaría junto con esta.

Sin embargo, si recordamos las gráficas de la *figura 3.1*, en las mediciones experimentales

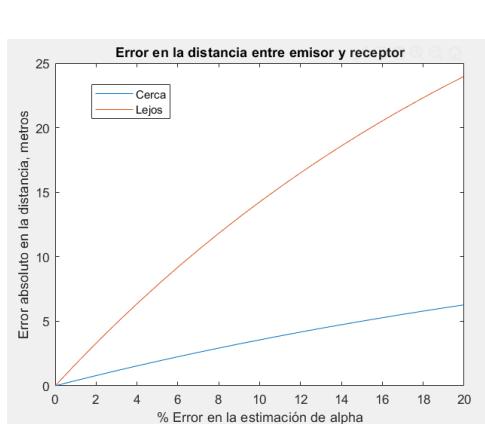
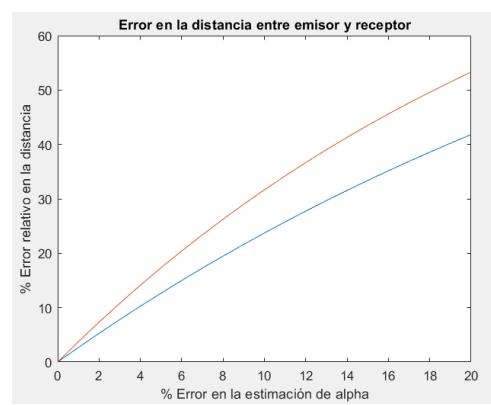
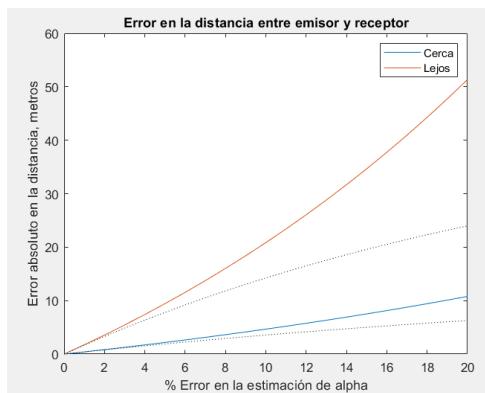
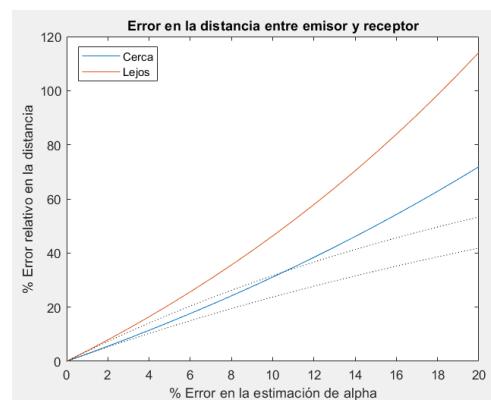
(a) Error absoluto frente a variación  $\alpha$ (b) Error relativo frente a variación de  $\alpha$ Figura 3.8: Error cometido al calcular la distancia considerando  $\epsilon$  del 20 %(a) Error absoluto frente a variación  $\alpha$ (b) Error relativo frente a variación de  $\alpha$ 

Figura 3.9: Error cometido al calcular la distancia entre emisor y receptor

realizadas el valor del coeficiente  $\alpha$  no ha subido más allá del 2,5 para ninguna de las situaciones, eso en ambientes propensos a atenuar más la señal, donde no siempre se dispone de línea de visión directa. Ese sería el límite físico superior para la atenuación, en ese edificio.

Debido a esto, al presentar los resultados de este segundo experimento se grafican solo entre los valores de interés,  $\epsilon$  entre el 0 % y el 20 %. A los resultados, que ven en la figura 3.9, se les ha añadido las líneas negras de puntos que se corresponden con los resultados obtenidos en la simulación anterior, para facilitar una comparación.

Analizando las gráficas se confirma el hecho de que en este caso el error no tiene un límite superior, puesto que tanto la curva del error absoluto como la del error relativo presentan una forma exponencial positiva. De hecho, el error relativo supera el umbral del 100 % cuando en el caso anterior eso no era posible.

La primera conclusión es que los errores cometidos al considerar un  $\alpha_s$  inferior a  $\alpha$  son superiores a los errores cometidos al considerar un  $\alpha_s$  superior que  $\alpha$ . En la figura 3.9 a) vemos que esta diferencia se agranda con el incremento de la distancia.

Cuando el emisor se encuentra cerca del receptor, los errores cometidos se asimilan en gran parte, pues ambas curvas tienen un comportamiento similar y la diferencia máxima entre ellas es de aproximadamente 2.5 metros.

En caso de que el emisor se encuentre a gran distancia del receptor, la separación de las

curvas empieza a notarse excesiva a partir del 8% en  $\epsilon$ . La diferencia en este caso es tal que para un  $\epsilon$  de 20% se alcanza una separación superior a los 25 metros.

La situación empeora si miramos los resultados de la *figura 3.9 b)* pues los errores relativos están muy por encima en comparación a la situación de la simulación anterior. Esto se debe a que no existe un límite superior para el error relativo, como el limitado por la máxima distancia entre los nodos visto anteriormente.

Para un valor de  $\epsilon$  del 10% se comete un 30% de error en caso de que los puntos se encuentren cerca, y se comete aproximadamente un 50% de error en la estimación de la distancia si los puntos están lejos unos de otros.

En definitiva, si no querremos que el error absoluto en la estimación de la distancia se dispare, los nodos han de estar razonablemente cercanos.

Tras la realización de estas primeras simulaciones, sacamos dos conclusiones importantes. La de sentido común es que debemos cuidar la distancia entre el emisor y el receptor puesto que a mayor esa distancia, mayor error podemos cometer. La segunda es que es preferible considerar un  $\alpha_s$  elevado a uno conservativo para el modelado de la atenuación de la señal. Aquí se presenta una línea crítica, tampoco podemos considerar un  $\alpha_s$  muy por encima a  $\alpha$  puesto que tendríamos un sistema que trabajaría con grandes errores en condiciones normales. Si consideramos un  $\alpha_s = 3$  para que esté siempre por debajo del  $\alpha = 2,5$  que es el máximo experimentado, tendríamos un sistema que en condiciones normales,  $\alpha = 2$ , trabajaría con un  $\epsilon$  del 50%. Sería inviable.

Sin embargo, como el error relativo es siempre superior en el caso de que  $\alpha_s < \alpha$ , si tuviésemos un 10% de diferencia entre ambos coeficientes, es preferible que esa diferencia sea para el caso  $\alpha_s > \alpha$ .

Esto hace que entendamos la complicación que conlleva implementar un sistema de localización en interiores, sobre todo porque las aplicaciones que tienen estos sistemas requieren usualmente unas precisiones elevadas. Pensemos en la localización de personas en un centro comercial. Si deseamos saber que escaparte les llama más la atención quizás nos baste con tener un error máximo de 2 metros, sin embargo, si deseamos saber que sección de ese escaparte ha llamado más atención *moda femenina o masculina, por ejemplo*, necesitamos tener una precisión de medio metro al menos. Como se verá más adelante, esto al final determinará el número de nodos fijos del sistema.

Comprobado el gran impacto que tiene la consideración del coeficiente  $\alpha_s$ , nos preguntamos como se verán afectados los métodos de localización vistos en el segundo capítulo. Así que procedemos con las simulaciones usando la segunda parte del *script ErrorsInAlphaha*.

Para ello necesitamos colocar tres nodos fijos, ya que son necesarios para el funcionamiento de los algoritmos de localización. Definimos dichos nodos de manera que formen un triángulo equilátero, en las posiciones que se listan a continuación.

- Nodo fijo 1:  $(x_1, y_1) = (0, 0)$
- Nodo fijo 2:  $(x_2, y_2) = (5, 10)$
- Nodo fijo 3:  $(x_3, y_3) = (10, 0)$

La idea es determinar el error que se produce al estimar la posición de un objeto en función de la diferencia que haya entre  $\alpha$  y  $\alpha_S$ , es decir,  $\epsilon$ .

Deseamos determinar ese error a distintas posiciones para el objeto, así que se definen varias posiciones para el objeto y se realizará la estimación de cada una de ellas. En concreto, las posiciones que vamos a estimar son las listadas tras este párrafo. En la *figura 3.10* presentamos una ilustración con la distribución de los puntos en el mapa, marcando con círculos los nodos fijos y con cruces las distintas posiciones del objeto.

- Objeto en centro:  $(x_o, y_o) = (5, 5)$
- Objeto en un lateral:  $(x_o, y_o) = (2, 5)$
- Objeto en una esquina:  $(x_o, y_o) = (10, 10)$
- Objeto muy cerca de un nodo fijo:  $(x_o, y_o) = (9, 1)$

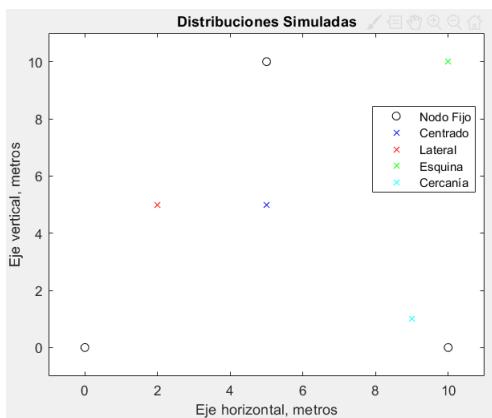


Figura 3.10: Posiciones consideradas para los nodos

El objetivo es estimar la ubicación del objeto cuando éste se encuentra en cada una de las posiciones indicadas, usando las técnicas de localización vistas en el capítulo dos. Para conseguir esto, aplicamos la función correspondiente *CalculateLocalizationTheory*, *CalculateLocalizationMaxMin* y *CalculateLocalizationCentroid* al realizar el método de localización en la segunda parte del *script*, con el objetivo de comparar la precisión de cada uno de los métodos.

En la primera simulación se le asignan unos valores a  $\epsilon$  comprendidos entre el 0 % y el 50 % y se considera un  $\alpha_s$  superior a  $\alpha$ , por lo se hace la asignación  $\alpha = \alpha_s(1 - \epsilon)$  para cada uno de los valores de  $\epsilon$ .

Se realiza una iteración del algoritmo por cada una de las posiciones del objeto. Como resultado se obtendrá el error absoluto, en metros, entre la posición real y la posición estimada, para cada una de las posiciones del objeto. Se muestran todos los resultados en una misma figura para poder comparar los errores en función de la posición.

En la *figura 3.11* se ven los resultados obtenidos para cada uno de los métodos de localización implementados.

Al usar el método de la centroide y el método MaxMin se ha debido de usar la función *MakeCirclesCross* previamente para asegurar que todas las circunferencias se cruzan entre sí. Es importante mencionar que el incremento aplicado al radio de la circunferencia es de 1 centímetro, 0.01 metro, para evitar aumentar los radios más de lo estrictamente necesario para que crucen todas entre sí.

Otro parámetro importante de mencionar, y que es común a todos los métodos de localización implementados, es la tolerancia admisible al comprobar la existencia de un punto

de cruce común entre las tres circunferencias. Recordemos que dicha comprobación se hace comparando el radio de una circunferencia con la distancia desde su punto central hacia los puntos de cruce entre las otras dos circunferencias.

Tras la realización de varios experimentos, se ha decidido permitir una tolerancia de 30 centímetros, puesto que para valores superiores obtenemos errores más grandes con este método que con los otros dos.

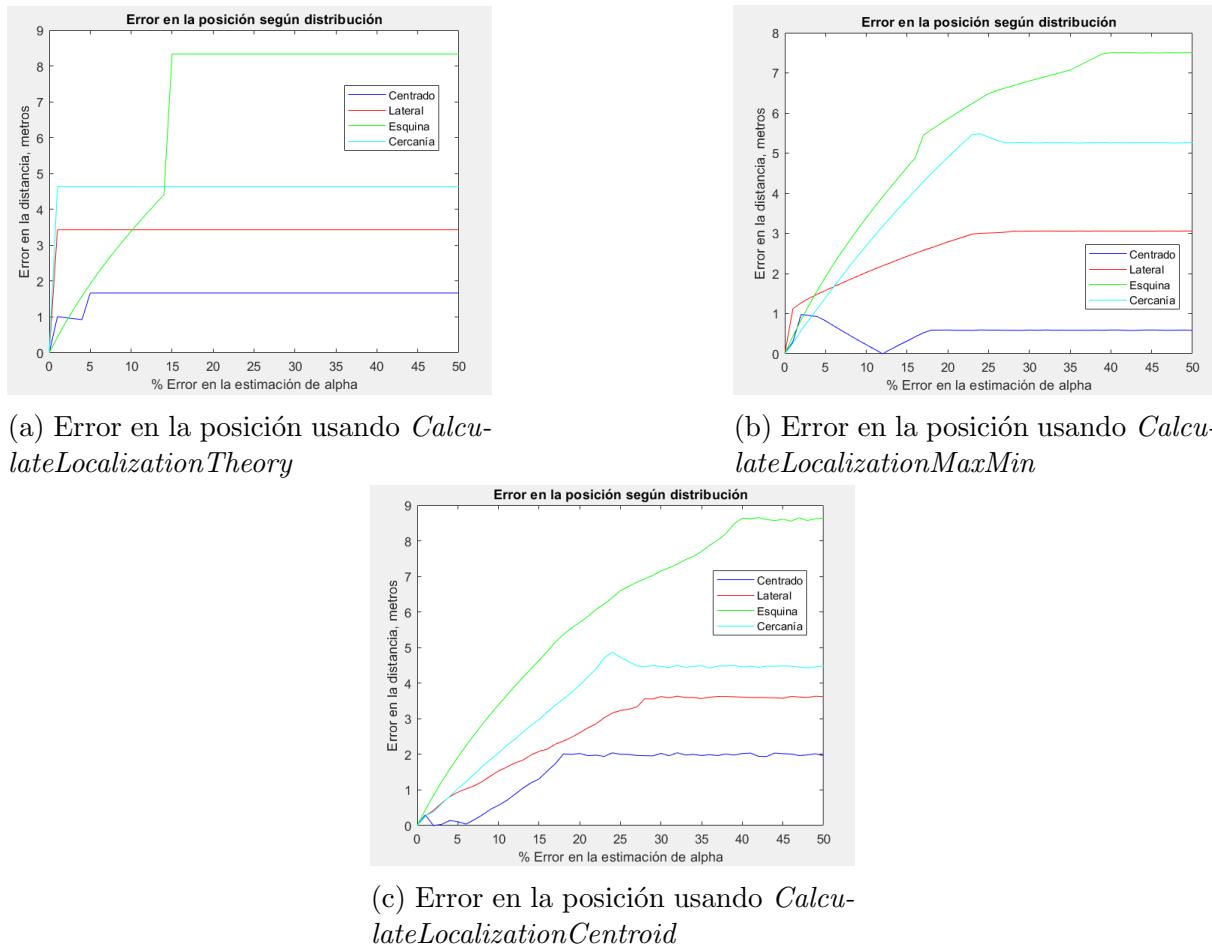


Figura 3.11: Error en la posición frente error en  $\alpha$

La figura 3.11 a) muestra la salida de aplicar la función *CalculateLocalizationTheory*. En esta figura se aprecian muy bien las variaciones en los métodos de localización usados, puesto que la evolución de los errores sigue líneas rectas para todas las posiciones. Vemos que siempre llega un punto a partir del cual el error es constante, esto se debe a que a partir de dicha  $\epsilon$  el algoritmo realiza triangulación, por lo que estima siempre la misma posición para el objeto. Así la distancia entre el objeto y la posición estimada, en este caso el baricentro del triángulo, será siempre la misma.

Además, determinamos que es en los cambios de pendiente de las curvas cuando se produce la transición de un método de localización a otro. Así, para la curva azul correspondiente a la posición del objeto centrada, vemos que para un  $\epsilon$  pequeño, cerca del 2%, hay un cambio de curvatura que se corresponde con el momento en el que ya no se cumple con la tolerancia admisible, por lo que se realiza el método Max-Min. La triangularización en este caso comienza cuando  $\epsilon$  alcanza un valor cercano al 5%, por lo que se interpreta que es en ese momento cuando las circunferencias dejan de cruzarse.

Destaca el error cometido cuando el objeto se encuentra en la esquina, pues es allí cuando se produce el valor máximo. Esto se justifica por la distancia entre el objeto y el nodo fijo de la posición inferior izquierda, puesto que dicha distancia es la máxima de entre todos los casos. Es más, vemos que el máximo valor de los errores está definido por la distancia existente entre el objeto y uno de los nodos fijos, puesto que el segundo, tercer y cuarto error máximo se producen para la posición cercana a un nodo, lateral y centrada respectivamente. Así, podemos concluir que el máximo error cometido se produce en el caso en el cual existe una mayor distancia entre el objeto y uno de los nodos fijos, indiferentemente de dónde se ubiquen los otros dos nodos fijos.

Esta relación se sigue cumpliendo con la implementación de las funciones *CalculateLocalizationMaxMin* y *CalculateLocalizationCentroid*, las gráficas de las *figuras 3.11 b) y c)* respectivamente. En estos casos no se realiza triangularización sino que se aumentan las distancias hasta conseguir que las circunferencias crucen.

Es más, realizando una comparación se determina que para una gran distancia entre el objeto y uno de los nodos fijos el error es similar independientemente del método de localización usado. En las tres gráficas, las curvas para la posición *cercanía* y *esquina* tiene un comportamiento similar. Un incremento progresivo hasta alcanzar el valor máximo. El máximo es algo menor para la curva *esquina* con el método Max-Min, sin embargo no es una diferencia notable.

La diferencia entre los distintos métodos se encuentra cuando la distancia del objeto a los nodos es pequeña, es decir, para las curvas de *centrado* y *lateral*. En ambos casos el error máximo cometido disminuye respecto a implementar triangularización. Viendo que ambas funciones *CalculateLocalizationMaxMin* y *CalculateLocalizationCentroid* ofrecen mejores resultados que *CalcuateLocalizationTheory* procedemos a comparar estos métodos.

Resulta interesante ver que el método Max-Min ofrece mejores resultados para valores de  $\epsilon$  elevados mientras que el método de la centroide presenta mejores resultados para  $\epsilon$  inferiores.

Centrándonos en los valores pequeños de  $\epsilon$ , entre el 0 % y el 15 % y en la posición central del objeto, ambos métodos devuelven errores nulos aunque la diferencia entre  $\alpha$  y  $\alpha_s$  aumente.

Este hecho puede llamar la atención a primera vista, sin embargo se debe a las condiciones impuestas en la simulación. Es preferible ver la *figura 3.12* previamente para comprender con facilidad la explicación que se ofrece en el siguiente párrafo.

En la figura, el asterisco verde representa la posición del objeto, el asterisco rojo representa la posición estimada por el método Max-Min y el rombo representa la posición estimada por el método de la centroide. Se representan las circunferencias, la posición del objeto y la posición estimada para cuatro valores distintos de  $\epsilon$ , 3 %, 10 %, 16 % y 30 %.

Para un valor de  $\epsilon$  pequeño, *figura 3.12 a)*, la estimación determinada por el método de la centroide es prácticamente exacta, por lo tanto el error es muy pequeño, casi nulo. Por el contrario, la estimación realizada por el método Max-Min se encuentra muy por encima de la posición real, por lo que presenta un error elevado.

Al incrementar el valor de  $\epsilon$ , *figura 3.12 b)*, la circunferencia superior deja de cruzar con las otras dos por lo que se incrementa su radio formando una nueva circunferencia, la definida en rojo. En esta situación vemos que la posición real se encuentra justo entre

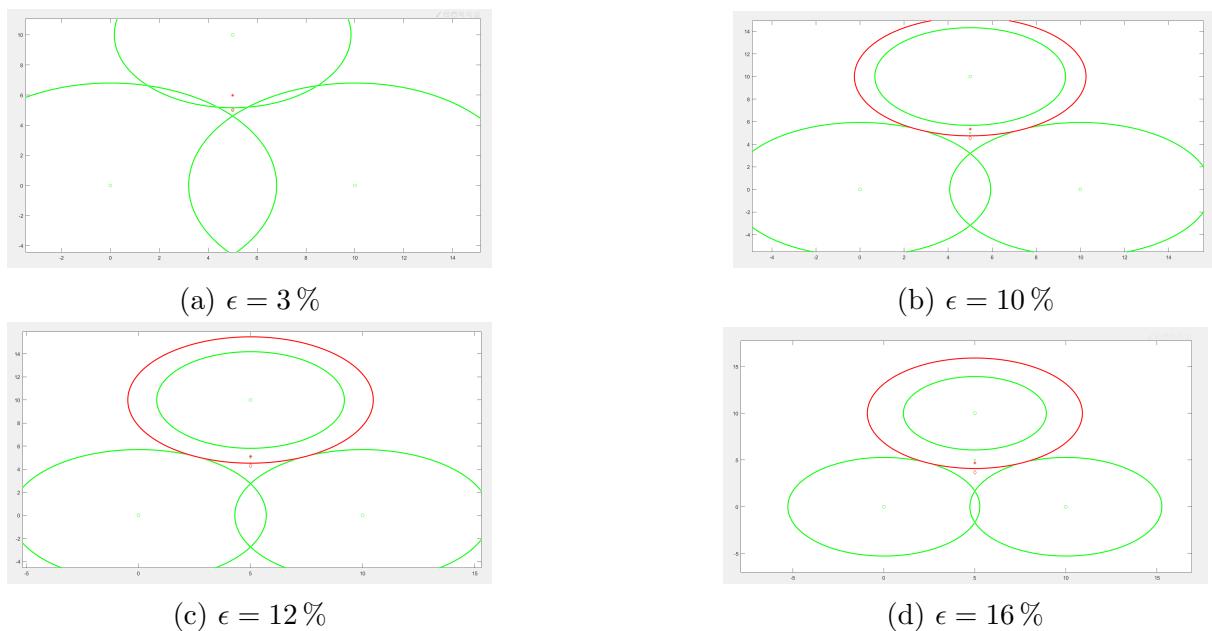


Figura 3.12: Circunferencias y estimación de la posición para distintos valores de  $\epsilon$  en la posición central

las posiciones estimadas por los dos métodos aplicados. Esto quiere decir que el error cometido por el método Max-Min ha disminuido, puesto que la posición estimada se ha aproximado a la real, mientras que el error cometido por el método de la centroide ha aumentado debido a que la posición estimada se ha alejado de la real.

Este efecto se produce debido a que estamos considerando en todo momento un  $\alpha_s$  superior al real, por lo que las distancias estimadas serán siempre menores que las reales. Para un  $\epsilon = 12\%$ , *figura 3.12 c)*, se ha de incrementar el radio de la circunferencia superior de tal manera que la posición estimada por el método Max-Min es prácticamente igual a la posición real, el error es prácticamente nulo, mientras que la posición estimada por el método de la centroide se aleja de la posición real hacia abajo.

A partir de este punto, conforme  $\epsilon$  incrementa, *figura 3.12 d)*, las posiciones estimadas se irán desplazando hacia abajo, alejándose de la posición real, pero ahora el error cometido por el método Max-Min es inferior al error cometido por el método de la centroide.

Se entiende ahora el motivo de tener valores nulos de error, para valores de  $\epsilon$  distintos de cero. Definido en una frase, es debido a que la posición estimada inicial se encuentra por encima de la posición real y se va desplazando hacia abajo conforme  $\epsilon$  aumenta. Por lo tanto, habrá algún valor de  $\epsilon$  para el cual la posición real será igual a la estimada, siendo así el error nulo.

Otro aspecto que merece la pena resaltar es que el error se mantenga constante para valores elevados de  $\epsilon$ , ya que en este caso no estamos realizando triangulación, es decir, no estamos estimando la misma posición siempre. Esto ocurre debido a una combinación de las condiciones de simulación junto con el algoritmo usado para el incremento del radio. Debido a que el objeto y los nodos fijos se encuentran en la misma posición siempre, la distancia estimada disminuye de forma proporcional en todos los casos. Como ven en la *figura 3.13*, para valores elevados  $\epsilon$  el incremento que se le ha de hacer al radio de las circunferencias es muy grande. Como el algoritmo de incremento de radios lo hace de forma proporcional, y la disminución de los radios con el incremento de  $\epsilon$  también ocurre de forma proporcional, llega un momento en el cual las circunferencias son tan pequeñas que al incrementarlas siempre se cruzarán en el mismo punto, debido a la proporcionalidad

del sistema. Así, la posición estimada por ambos algoritmos será siempre la misma, por lo que el error se mantendrá constante también.

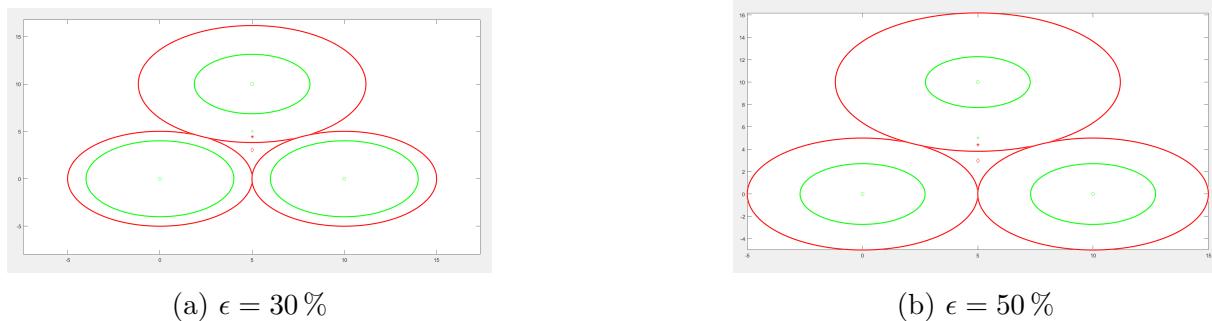


Figura 3.13: Circunferencias y posiciones estimadas para valores de  $\epsilon$  elevados

Habiendo entendido el porqué en algunas situaciones es mejor usar el método Max-Min mientras que en otras es mejor usar el método de la centroide, decidimos realizar un nuevo algoritmo que aplique el método que mejor se adapte en cada situación. Para ello se ha desarrollado la función *CalculateLocalization*.

En las representaciones de la *figura 3.12* vimos que la posición real del objeto se encuentra justo en medio de las posiciones estimadas por los métodos Max-Min y Centroide para un valor de  $\epsilon$  cercano al 10 %. A partir de allí el error cometido por el método Min-Max es menor que el error cometido por el método de la centroide.

Así, para poder alternar entre un método y otro usamos el incremento del radio de una de las circunferencias como variable de transición. Para un  $\epsilon = 10\%$ , el incremento que se le realiza al radio de la circunferencia superior es de 0.92 metros. Así, determinamos como valor de transición entre un método y otro un incremento del radio de alguna de las circunferencias superior a 0.95 metros, para incluir pequeñas variaciones que se puedan producir.

En otras palabras, si el incremento que se le realizan a las circunferencias es inferior a 0.95 metros, se empleará el método de la centroide. Por el contrario, si dicho incremento es superior a los 0.95 metros, se usará el método Max-Min.

Las simulaciones para comprobar la eficacia de este método consiste en compararlo con los dos métodos en los que se basa, para cada posición del objeto. Así, en las mismas condiciones descritas en la simulación anterior, implementamos como algoritmo de localización la función *CalculateLocalization* para cuando el objeto se encuentre en las posiciones central, lateral, esquina y cercanía.

Se muestran los resultados en la *figura 3.14*. Para poder realizar una comparación entre los tres métodos, se han pintado las tres curvas en la misma gráfica, cada curva corresponde con un método. En la figura tienen cuatro gráficas, una por cada posición del objeto, de esta forma se pueden interpretar los resultados de una manera más simple. Tengan en cuenta que el método implementado por la función *CalculateLocalization* es una combinación entre los dos métodos, Max-Min y Centroide, así que su curva sobrescribirá a una de las otras en todo momento.

Vemos que en la mayoría de los casos la línea verde correspondiente a los errores cometidos por *CalculateLocalization* representa el valor mínimo de error que se puede cometer para un valor de  $\epsilon$  dado.

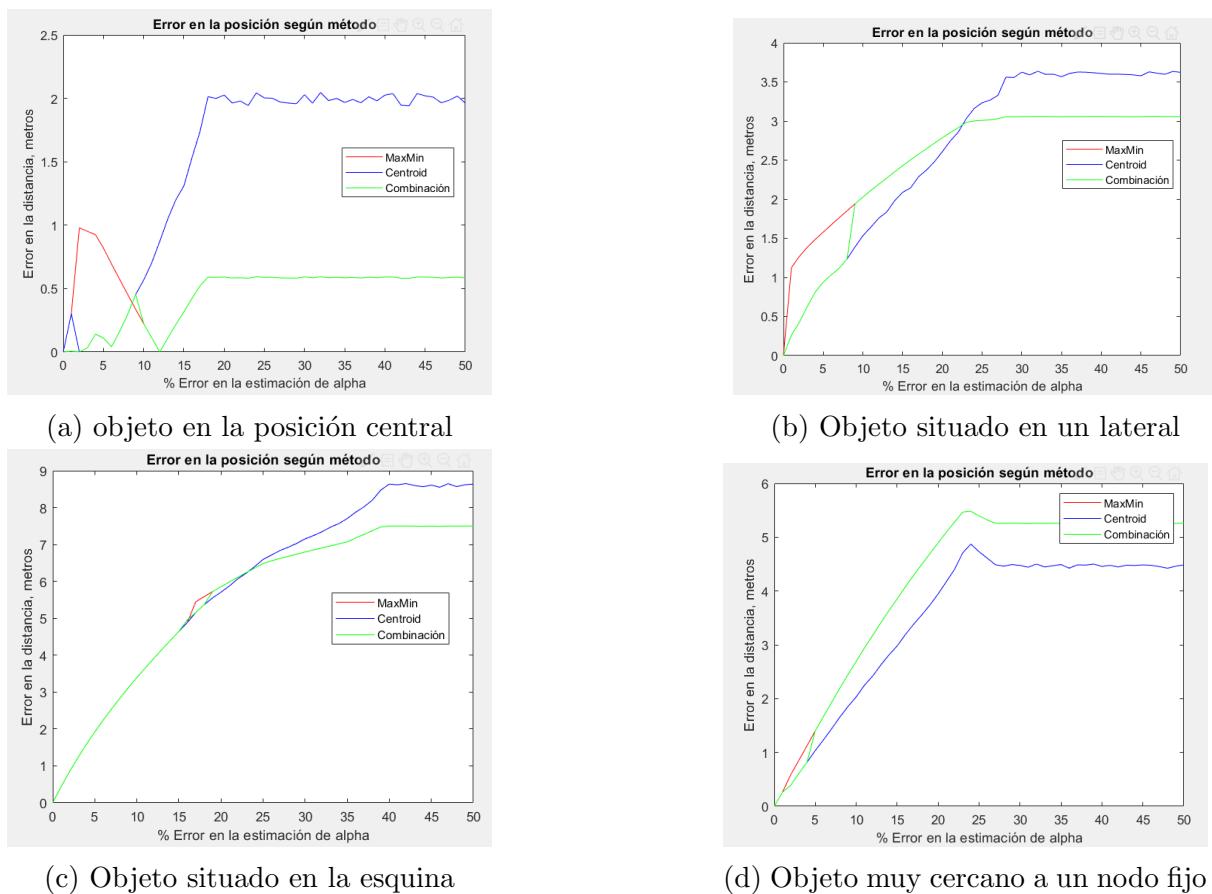


Figura 3.14: Comparación entre *CalculateLocalization*, *CalculateLocalizationMaxMin* y *CalculateLocalizationCentroid*

La gran ventaja de este algoritmo se aprecia en la figura 3.14 a) en el caso de que el objeto se ubique en la posición central. Sobre todo para los valores de  $\epsilon$  que nos son de especial interés, comprendidos entre 0 % y el 20 %, evitamos tener errores de un metro aproximadamente, para pasar a tener unos errores de unos cuantos centímetros. Esta diferencia es tal que puede hacer que el sistema a implementar sea válido o no.

Lo mismo sucede para valores de  $\epsilon$  grandes, donde vemos que el error del método de la centroide incrementa hasta los 2 metros mientras que el error del método Max-Min, que es el usado por la función *CalculateLocalization* en estos casos, se estabiliza en un valor máximo próximo al medio metro.

En resumen, para pequeñas diferencias entre  $\alpha$  y  $\alpha_s$  pasamos de tener errores de un metro, a tener errores entre los 10 y los 30 centímetros. Para grandes diferencias entre  $\alpha$  y  $\alpha_s$  pasamos de tener un error de dos metros, a tener un error de medio metro. Sin duda, es mejor la implementación del método de localización con la función *CalculateLocalization* en comparación los otros dos métodos por separado.

La gran excepción de este resultado sucede cuando el objeto se ubica en la cercanía de un nodo fijo. En este caso el error cometido por el método Max-Min a altos valores de  $\epsilon$  es superior al error cometido por el método de la centroide. Como *CalculateLocalization* emplea el método Max-Min en esos casos, el error cometido no es el menor posible. Aun así, para valores de  $\epsilon$  comprendidos entre el 0 % y el 20 % la diferencia entre los errores es pequeña.

No olvidemos que los errores tan elevados en las posiciones extremas son debidas a la lejanía del objeto con algún nodo fijo. No hay manera algorítmica de mejorar dichos

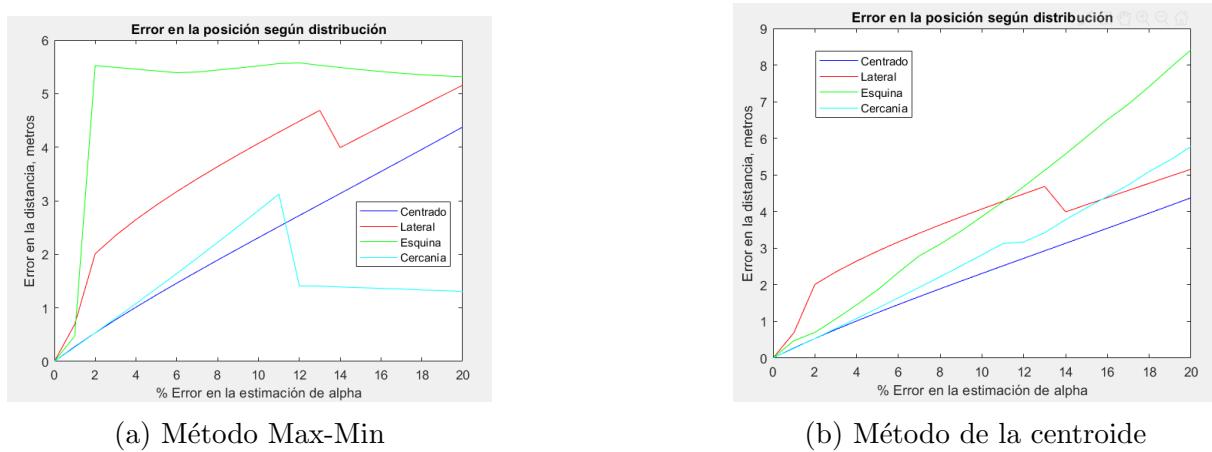


Figura 3.15: Errores cometidos en la posición al considerar valores de  $\alpha_s$  inferiores a  $\alpha$

errores. Las dos soluciones posibles en tales casos son, o bien procurar que nunca haya una gran distancia entre el objeto y los nodos fijos, o bien conseguir que  $\alpha_s$  sea igual a  $\alpha$ . Esto último resulta complicado pues, vemos en la figura 3.14 c), si el objeto está colocado en la esquina donde la distancia con uno de los nodos fijos es máxima, tan sólo un 10 % de variación entre  $\alpha$  y  $\alpha_s$  provoca un error de 3 metros en la posición.

En todas las simulaciones se ha considerado el caso de considerar un  $\alpha_s$  superior a  $\alpha$ . En este caso las distancias estimadas son menores que las reales, así que siempre habrá que aumentar las circunferencias.

Veamos el caso en el cual consideramos un  $\alpha_s$  inferior al valor real. Las distancias estimadas serán superiores a las existentes, por lo que las circunferencias en este caso siempre se cruzarán entre ellas, así que no hará falta incrementar ningún radio.

Para determinar qué método es mejor emplear en este caso, representamos gráficamente los errores cometidos por el método Max-Min y el método de la centroide para cada una de las posiciones del objeto. En la figura 3.15 se muestran los resultados. Vemos que para ambos métodos de localización el error crece linealmente cuando el objeto se encuentra en la posición central o en un lateral.

La diferencia reside cuando el objeto se encuentra en una esquina o en la posición muy cercana a otro nodo. En este caso vemos que el error incrementa de forma abrupta con el método Max-Min, para luego mantenerse constante, mientras que usando el método de la centroide sigue incrementado de forma lineal suavemente.

El error máximo que se comete con el método Max-Min es inferior, sin embargo dicho error máximo es para valores de  $\epsilon$  grandes. Con el método de la centroide se cometan errores inferiores en todo momento, para valores de  $\epsilon$  comprendidos entre el 0 % y el 15 %. Así, concluimos que es preferible usar el método de la centroide si los círculos se cruzan entre sí, frente al método Max-Min. Si consultan el código de la función *CalculateLocalization* en el anexo a, verán que primero se comprueba si las tres circunferencias cruzan entre sí, y en caso afirmativo, aplica directamente el método de la centroide. He aquí el motivo de dicha implementación.

La principal conclusión que sacamos tras la realización de las simulaciones anteriores es que **el nodo fijo más alejado es el que determina el error que se comete**.

Por lo tanto, **la distancia entre los nodos es el factor que más afecta al error**

**cometido.**

Entonces nos surge la pregunta, ¿ cuántos nodos hace falta colocar en un determinado plano para conseguir que el error sea pequeño?

Esa es la pregunta que se trata de responder con las simulaciones diseñadas en el *script ErrorsPerDensity*.

Un aspecto crítico de un sistema *ILS* es que el objeto se puede encontrar en cualquier posición del mapa. No es posible conocer con antelación la geometría que formarán los tres puntos más cercanos.

La función *GenerateTrackingPoints* es la que genera esta incertidumbre.

Con la primera simulación de este *script* se trata de determinar cuantos nodos fijos por unidad de área se han de colocar para conseguir que el error no alcance los valores vistos con el objeto en la esquina en las simulaciones anteriores. En resumen, tratamos de conseguir una densidad de nodos adecuada.

Para ello definimos un mapa cuyas dimensiones son de  $50 \times 50m^2$  y usamos la función *GenerateEquidistantPoints* para colocar los nodos fijos uniformemente distribuidos en el mapa. Tras ello se colocan puntos distribuidos aleatoriamente en el mapa, para los que estimaremos su posición. Por último calcularemos el error máximo, mínimo y medio que se ha cometido en la estimación de la posición de todos los puntos aleatorios.

Se usarán cuatro intervalos distintos para el valor de  $\epsilon$ . En los experimentos realizados por [14] la diferencia habitual entre los valores experimentales de  $\alpha$  y los teóricos está en un 10 %, así que en los cuatro casos de la simulación consideremos una incertidumbre del 20 %, 15 %, 10 % y el 5 %.

Recordamos que a efectos prácticos de esta simulación, el valor de  $\epsilon$  se genera de manera aleatoria entre un máximo y un mínimo. Así, para que la simulación abarque ambos casos,  $\alpha_s$  superior a  $\alpha$  y  $\alpha_s$  inferior a  $\alpha$ , el rango de valores para  $\epsilon$  son de  $[-0,1, 0,1]$ ,  $[-0,075, 0,075]$ ,  $[-0,05, 0,05]$  y  $[-0,25, 0,025]$ .

Como resultado de la simulación obtendremos los errores máximos, mínimos y medios cometidos al estimar la posición de todos los puntos aleatorios, para cada una de las incertidumbres consideradas. La *figura 3.16* vemos los resultados para cada rango de valores para  $\epsilon$ .

El primer aspecto que destaca en las gráficas de la figura es que el error máximo no varía en ninguno de los cuatro casos. Esto se debe a que es suficiente con la existencia de un punto situado en una posición comprometida, como en una esquina del mapa dejando todos los nodos fijos detrás, para que dicho error tenga un valor grande.

El haber representado los errores máximos en las gráficas, imposibilita apreciar el tamaño de los errores mínimos y medios, puesto que la diferencia de valor es muy grande. En cuanto a los errores mínimos, estos no deben de preocuparnos puesto que las curvas están muy cercanas al cero.

Así que nos concentraremos en los valores de error medio, vean la *figura 3.17*. Como intuiríamos, los errores decrecen conforme aumenta la densidad de nodos, de la misma forma que decrecen con la disminución de  $\epsilon$ . Destaca el hecho que en todos los casos la pendiente de las curvas se aproxima a cero a partir de los 100 nodos fijos, de hecho, incluso para un valor elevado de incertidumbre, como lo es un 20 %, el error medio se

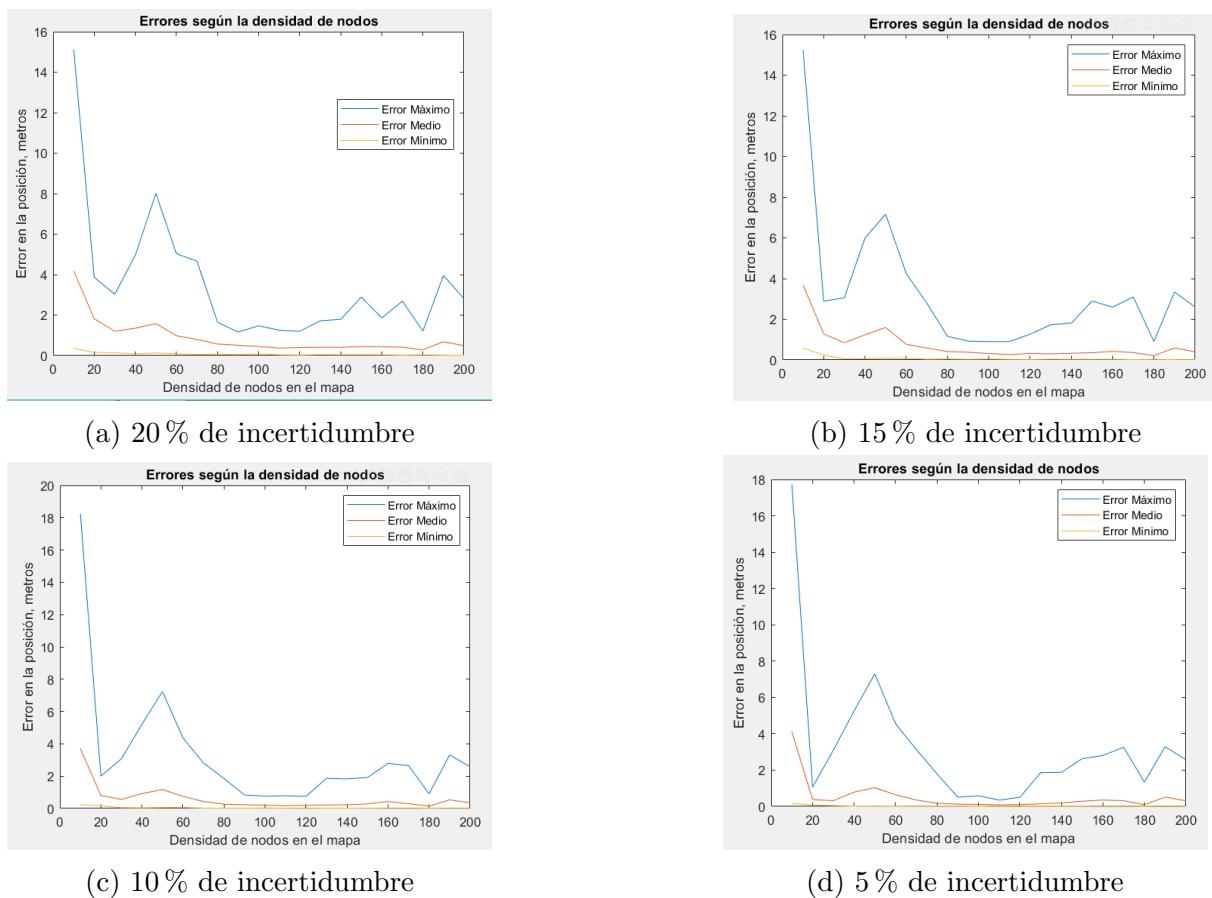


Figura 3.16: Errores máximos, mínimos y medios en función de la densidad de nodos existente

encuentra cercano a los 0.5 metros. Para valores de  $\epsilon$  que oscilen un 10 %, los más habituales en condiciones normales, el error medio se encuentra por debajo de los 0.3 metros.

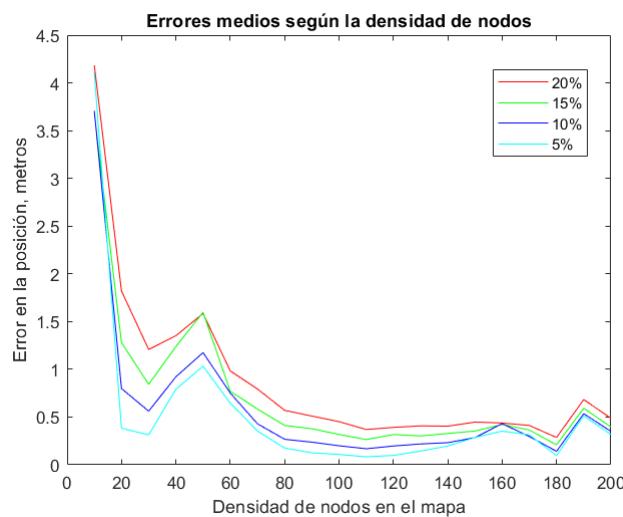


Figura 3.17: Errores medios en función de la densidad de nodos y  $\epsilon$

Los resultados obtenidos son muy prometedores. El mapa usado para esta simulación tiene un área de  $2500m^2$ . Que a partir de los 100 nodos fijos los errores se mantengan constantes indica que la densidad de nodos a partir de la cual no es económico

eficiente aumentar el número de puntos fijos es de *1nodo* cada  $25m^2$ . En esta situación la separación máxima entre nodos sería de  $\sqrt{5^2 + 5^2} = 7$  metros, una distancia a la que pueden llegar tecnologías baratas y de bajo consumo como el *bluetooth*.

Es más, en dichas condiciones incluso con una gran incertidumbre, del 20 %, el error ronda los 0.5 metros, una precisión suficiente para la implementación de un sistema *ILS*. Si consideramos una incertidumbre menor, como la de un 10 %, conseguimos dicho error de 0,5 metros con tan sólo 60 nodos fijos en el mapa, es decir, *1nodo* cada  $40m^2$ .

Otro detalle interesante es que para una incertidumbre del 10 %, se mantiene el error muy por debajo de los 0.5 metros. Por lo que en ese caso obtendríamos un sistema que trabaja con una precisión muy elevada. En conclusión, no es necesario conocer la atenuación con una exactitud elevada para conseguir una precisión más que razonable.

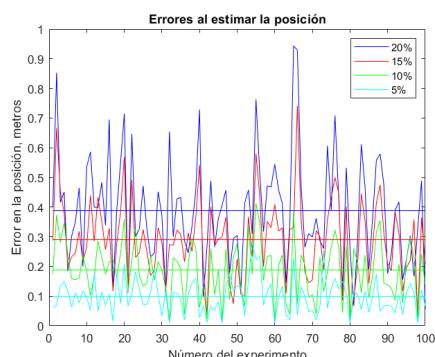
Comprobemos los resultados obtenidos haciendo uso de la segunda parte del *script ErrorsPerDensity*. Para ello se ha diseñado un mapa cuadrado de  $400m^2$  con un nodo cada  $25m^2$ . Generamos puntos aleatorios dentro del perímetro y procedemos a realizar su localización. Obtenemos los errores cometidos al estimar la posición de cada uno de los puntos aleatorios. De esta manera podremos representar todos los errores en una gráfica y determinar el rango en el que se encuentran.

Para corroborar los resultados obtenidos arriba, volemos a realizar el experimento duplicando la cantidad de nodos fijos en el mapa.

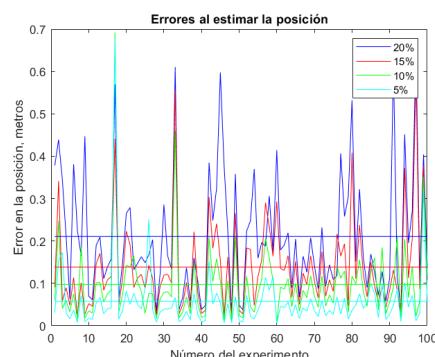
Se representan los resultados en la *figura 3.18*. Se muestran sobre un mismo gráfico los errores cometidos a varias incertidumbres, las líneas continuas representan los errores medido para su respectiva incertidumbre.

Es interesante observar que colocar el doble de nodos, *figura 3.18 b)*, hace que los errores se estabilicen frente a grandes valores de  $\epsilon$ . Vemos que todos los errores medios cometidos se encuentran muy cerca unos de otros, habiendo una diferencia de 10cm entre el error medio cometido con una incertidumbre del 20 % y una del 5 %.

Este resultado confirma el hecho de que la distancia hacia los nodos afecta en mayor medida al error cometido que una mala consideración de  $\alpha$ .



(a) Colocando un nodo cada  $25m^2$



(b) Colocando el doble de nodos

Figura 3.18: Errores cometidos al estimar las posiciones en el mapa de  $400m^2$

No se confirma el hecho de que a partir de un nodo fijo cada  $25m^2$  el error no disminuye, puesto que en realidad colocar el doble de nodos ha reducido el error medio cometido a aproximadamente la mitad. En el caso caso de la *figura 3.18 a)* tenemos un error medio de  $0,4m$  para una incertidumbre del 20 %, mientras que en el caso de la *figura 3.18 b)* el

error es cercano a los  $0,2m$ .

En este caso habría que determinar si es necesaria una precisión de  $20cm$  y habría que determinar la viabilidad de colocar el doble de nodos. Harían falta 1 nodo fijo cada  $12m^2$ , esa cantidad supone instalar un dispositivo cada 3 metros dentro del edificio.

Concluimos que **incrementar la densidad de nodos reduce el error a costa de aumentar el coste del despliegue del sistema ILS**. Habrá que llegar a un compromiso teniendo en cuenta la precisión requerida y el presupuesto disponible.

A diferencia, colocar 1 nodo cada  $25m^2$  si es viable, y confirmamos el hecho de que se consigue una buena precisión incluso a grande incertidumbres. Es más, considerando una incertidumbre del 10 % como la obtenida en los documentos de la bibliografía, el sistema *ILS* tendría un error medio de  $20cm$ . Dicha precisión es más que suficiente para las aplicaciones habituales que tienen este tipo de sistemas.

Procedamos con las conclusiones de las simulaciones realizadas durante la elaboración de este capítulo.

La más importante es que **la distancia prevalece sobre el conocimiento de  $\alpha$** , veíamos en la *figura 3.11* que los errores máximos al estimar la posición ocurren en situaciones en las que el objeto se encuentra alejado de alguno de los nodos fijos.

Este hecho se confirma con los resultados obtenidos en las *figuras 3.17*, puesto que el error decrece abruptamente conforme incrementa la densidad de nodos fijos en el mapa, indiferentemente de la incertidumbre.

Por último, en la *figura 3.18* se confirma la posibilidad de tener una precisión adecuada para la implementación de un sistema *ILS* sin conocer la atenuación de la señal por el canal con mucha exactitud.

# Capítulo 4

## Aplicación práctica

Ya hemos determinado cómo se realiza la localización en interiores, qué variables se han de tener en cuenta y qué efectos tienen una mala estimación de estas variables en el sistema final.

Como continuación del trabajo realizado en el capítulo anterior, en este capítulo trataremos de comprobar la veracidad de todas las conclusiones que se han realizado.

Para ello simularemos la implementación de un sistema *ILS* real. Como cada vez es más evidente la revolución en el mercado de los vehículos, estudiaremos el comportamiento de estos sistemas en un aparcamiento. Una aplicación con alta probabilidad de que se implemente en la mayoría de los parkings públicos, sobre todo con la llegada del vehículo autónomo.

Siendo este el caso, nos encontramos con la primera cuestión que deseamos resolver, y es que en un aparcamiento un vehículo se encuentra en constante movimiento. Hay varios factores que se pueden tener en cuenta al diseñar un sistema de localización en este caso. En este trabajo no entraremos en analizar aspectos técnicos como la rapidez con la cual nuestro sistema es capaz de determinar una posición o la existencia de varios objetos que se necesitan localizar a la vez. Al fin y al cabo realizar un seguimiento continuo de varios objetos móviles se simplifica en aplicar el algoritmo de localización varias veces en un intervalo de tiempo muy corto y analizar todas las posiciones obtenidas, así podemos definir la trayectoria seguida para cada objeto.

La capacidad de un sistema de desarrollar esas funciones viene determinada por aspectos tecnológicos como el hardware empleado, el software, la cantidad de servidores instalados y más matices que no forman parte de la línea de estudio que conlleva este trabajo.

La intención de este capítulo es determinar cuanta precisión se puede obtener según la información vista en los capítulos superiores y determinar si es suficiente para localizar un vehículo en un aparcamiento, suponiendo una incertidumbre en la atenuación de la señal por el canal.



Figura 4.1: Complejo residencial diseñado en Bordeaux

Fuente: [8]

## 4.1. Contexto, preparación del experimento y algoritmo

La mayoría de los países desarrollados, entre ellos España, han puesto ya fecha a la prohibición de los vehículos por combustión. Con la llegada de los vehículos eléctricos se abrirán nuevas posibilidades de comunicación entre los mismos y se expandirá el área de la conducción autónoma gracias a la tecnología que los vehículos permiten implementar. Realizar un seguimiento y el control de un motor eléctrico es mucho más simple comparado con el control y el seguimiento de un motor de combustión.

Siendo esto un tema de estudio aparte, la breve explicación dada sirve para justificar la elección de un aparcamiento pues considero de gran utilidad que los vehículos autónomos se comuniquen con un sistema de gestión al abandonar la vía pública para entrar en un edificio, de modo general un aparcamiento.

En todo caso, esto requiere de un sistema de localización para motorizar la ubicación de los vehículos que están dentro. Es por ello que ha surgido la curiosidad sobre qué resultados obtendríamos al implementar un sistema *ILS* en un aparcamiento, determinar cuantos nodos fijos serían suficientes para conseguir la precisión necesaria.

Para facilitar el diseño del aparcamiento, se ha usado el plano de una de las plantas del aparcamiento diseñado en Brodeaux por el estudio de arquitectura *Brisac Gonzales*. La información de todo el proyecto junto a todos los planos es de libre acceso y pueden encontrarla en [8]. Se trata de un complejo residencial con un diseño único muy interesante en el cual las casas se encuentran ubicadas encima del aparcamiento. Este es un edificio de 4 plantas que tiene 106 plazas por planta. La planta baja del complejo, con acceso a la carretera, está dedicada a locales comerciales.

Vean en la *figura 4.1* una ilustración del proyecto comentado.

En el caso de este trabajo se realizará la simulación sobre tan solo una de las plantas del aparcamiento. Como no se incluyen medidas en los planos, suponemos que las plazas cumplen con los estándares mínimos por lo que cada plaza tendrá 2.5 metros de ancho y 5 metros de largo. Viendo en la "*figura 4.2*" un esquema del plano del aparcamiento, constatamos que tiene un largo de 36 plazas distribuidas en batería, así que podemos decir que el eje horizontal de nuestro mapa tiene un largo de  $36\text{plazas} * 2,5(\text{m}/\text{anchoplaza}) = 90\text{metros}$ . En cuanto al ancho del aparcamiento, apreciamos que el carril de circulación tiene el mismo ancho que el largo de una plaza de aparcamiento. Así, se deduce una capacidad de 6 plazas a lo ancho del aparcamiento, por lo que se considerará una dimensión de

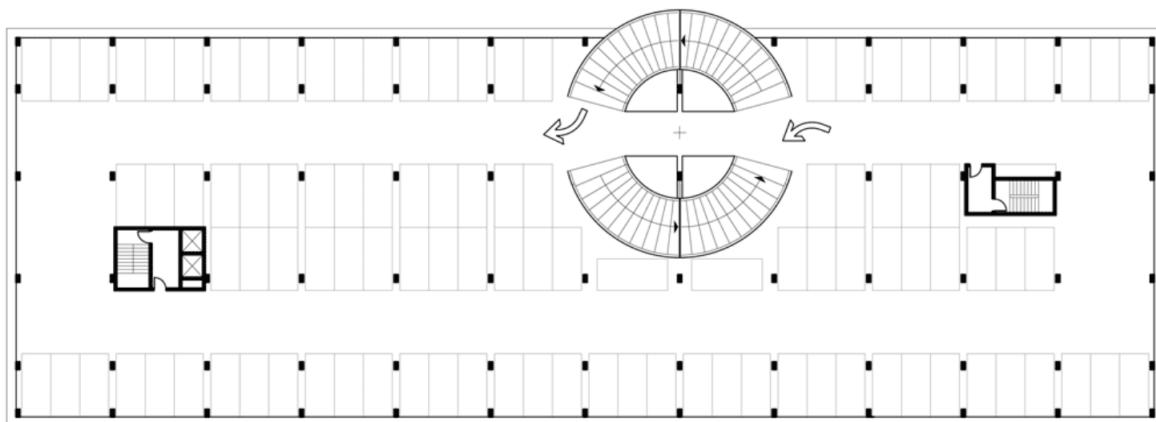


Figura 4.2: Esquema del aparcamiento

$$6 \text{plazas} * 5(\text{m/largoplaza}) = 30\text{m}.$$

La superficie útil calculada para el aparcamiento es de  $2700\text{m}^2$ .

Haciendo una pequeña lectura de los artículos [11] y [5], encontraremos muchos factores que se han de tener en cuenta antes de implementar la localización de vehículos en aparcamientos.

Aquí nos centramos en determinar cual es la densidad de nodos óptima para que nos permita realizar la estimación de la trayectoria de un vehículo con una precisión suficiente, determinar los errores que se cometan en la estimación de la posición y comparar los resultados con los obtenidos en el apartado anterior.

Las dimensiones del mapa se han escalado en *MatLab* de forma que cada unidad en uno de los ejes equivale a un metro en el mapa. Se hace uso del *script ParkingTrajectory* el cual realiza dos funciones, determinar la trayectoria de un vehículo y calcular los errores que se han cometido a lo largo de la misma. En la *figura 4.3* pueden ver la estructura lógica que se sigue.

Primero se definen manualmente trayectorias dentro del parking que simulan vehículos que entran y van a aparcar. Las dimensiones del aparcamiento, como hemos dicho, un largo de 90 metros en el eje horizontal y un ancho de 30 metros en el eje vertical. Se hace uso de la función *GenerateEquidistantPoints* para generar puntos fijos uniformemente distribuidos y se procede con la estimación de la posición para cada punto definido por las trayectorias.

Por último se calculan los errores en cada estimación de posición realizada y se grafican los resultados.

La *figura 4.4* muestra las dos trayectorias que se han diseñado, de tal forma que se cubran todos los posibles puntos con gran posibilidad de generar un gran error.

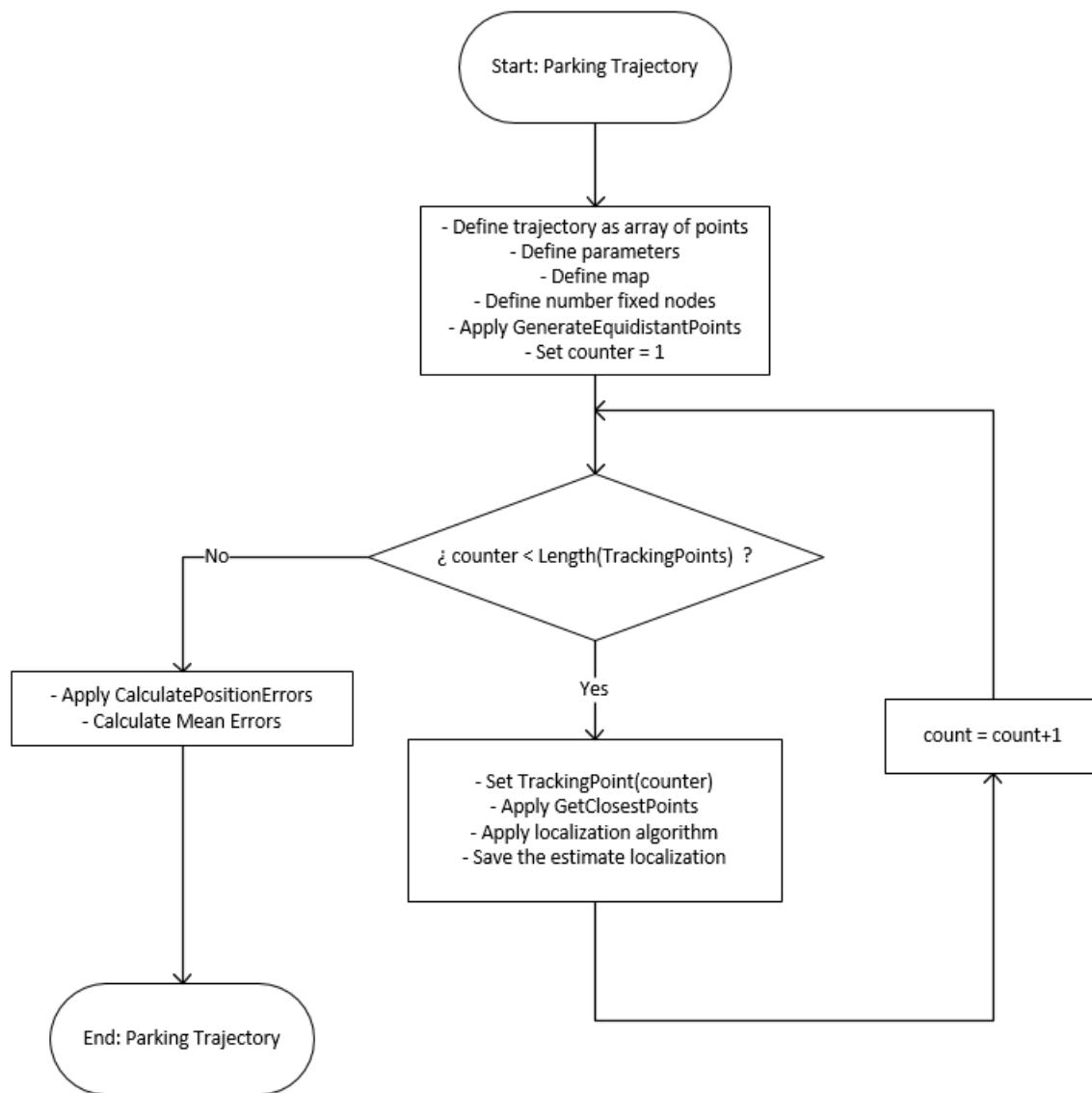
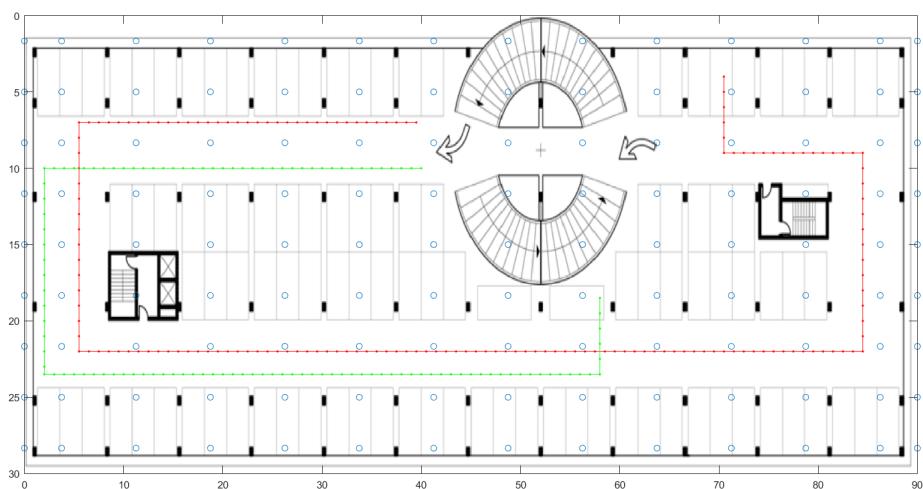
Figura 4.3: Estructura lógica de *ParkingTrajectory*

Figura 4.4: Trayectorías definidas líneas continuas

## 4.2. Presentación de los resultados

Hemos visto en el capítulo anterior que la incertidumbre máxima habitual que se suele tener es del orden del 10 %. Es por ello que se ha decidido realizar las simulaciones para dos valores de incertidumbre, 15 % y 10 %.

Comencemos analizando la diferencia entre el algoritmo de localización de aumento de radios, implementado con la función *CalculateLocalization*, y el algoritmo de localización que realiza triangulación, implementado con la función *CalculateLocalizationTheory*. La *figura 4.5* muestra las dos trayectorias simuladas *A* y *B* junto a sus respectivas estimaciones, para una incertidumbre del 15 %. Siguiendo con los resultados obtenidos en el capítulo anterior, colocamos un nodo cada  $25m^2$ , por lo que en este mapa de  $2700m^2$  se colocan 108 nodos fijos distribuidos uniformemente mediante la función *GenerateEquidistantPoints*. En adición a estos 108 nodos, se han añadido nodos en las paredes laterales de manera que queden alineados con los demás. Esto se ha hecho así para asegurar que un vehículo siempre estará rodeado de nodos fijos por todas las direcciones.

Se aprecia una mejor precisión en la *figura 4.5 b)* para los desplazamientos horizontales en ambas trayectorias, lo que justifica que se obtienen mejores resultados con el algoritmo nuevo diseñado que incrementa los radios de los círculos.

Las peores estimaciones se presentan en los desplazamientos verticales de la *trayectoria A*. Por la figura, intuimos que se debe a la posición del coche respecto de los nodos fijos, ya que en comparación, la estimación durante el desplazamiento vertical de la *trayectoria B* es muy precisa.

Comparando las diferencias entre ambas trayectorias, vemos que los nodos fijos más cercanos de la *trayectoria A* quedan todos desplazados a un lado del punto de muestra. Por el contrario, cualquier punto de muestra de la *trayectoria B* se encuentra rodeado de los puntos fijos más cercanos.

Este hecho es lo que se pretendía evitar al haber incluido los nodos fijos pegados a la pared, que ningún vehículo tenga todos los nodos fijos más cercanos desplazados a un mismo lateral.

Este resultado pone en evidencia lo importante que es la posición del vehículo respecto a los nodos fijos.

Aún así, con ambos métodos de localización la precisión media parece ser suficiente para un vehículo en un garaje. Representamos los errores cometidos a lo largo de la trayectorias en la *figura 4.6*.

Vemos en la *figura 4.6 a)* que el error medio ronda los 1,5metros para ambas trayectorias en caso de realizar triangulación. Si se aplica el método de incremento de radios, el error medio baja, siendo la diferencia más notable la correspondiente a la *trayectoria B*, donde el error medio se reduce por debajo de la unidad, mientras que el error de la *trayectoria A* se mantiene por encima de la unidad, alrededor de los 1,2metros. Esta diferencia en la reducción del error es debida a que durante los desplazamientos verticales la estimación fue igual de imprecisa para la *trayectoria A* usando ambos métodos de localización, mientras que para la *trayectoria B* se aprecia un gran incremento de precisión en todo momento.

Aunque los errores medios son razonables, la volatilidad del error representa un problema pues ambas trayectorias pueden superar los 5metros de error, con ambos métodos. Volviendo a la *figura 4.5*, cuando la *trayectoria A* entra en su aparcamiento vemos que

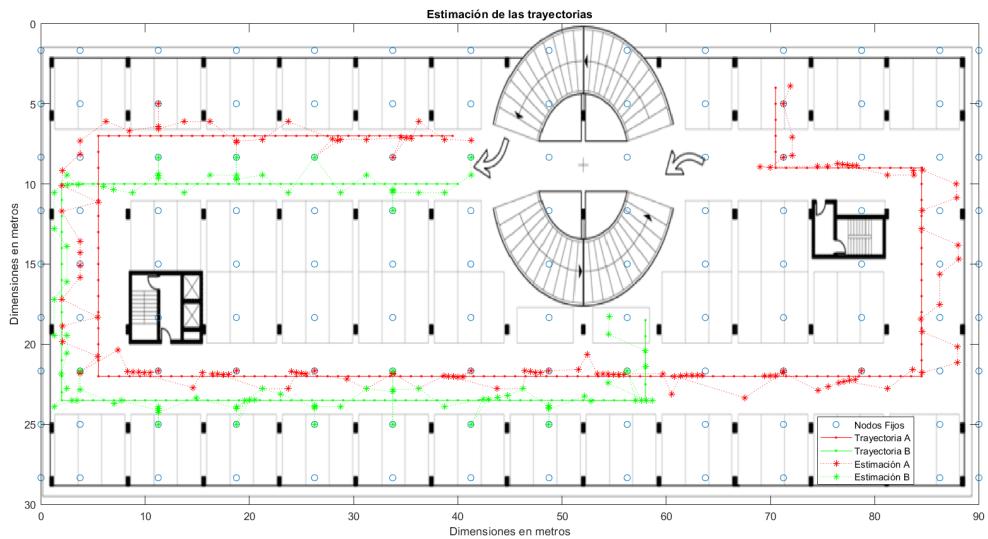
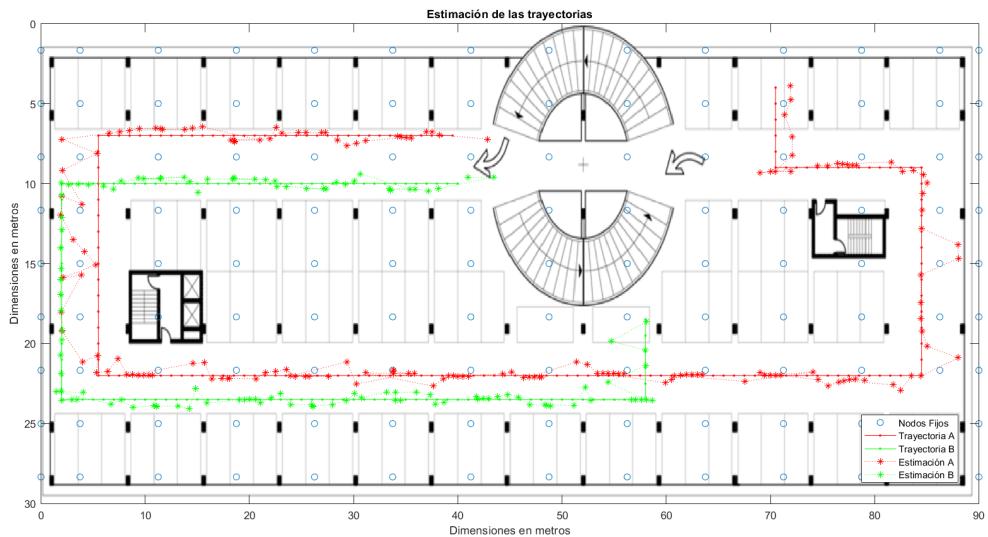
(a) Usando *CalculateLocalizationTheory*(b) Usando *CalculateLocalization*

Figura 4.5: Trayectorias y estimación con un incertidumbre del 15 % con 108 nodos fijos en el garaje

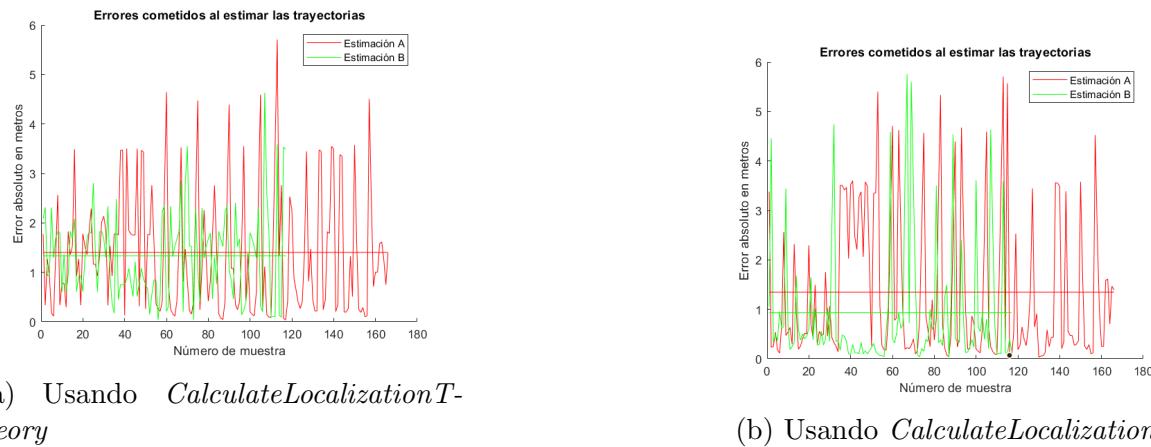


Figura 4.6: Errores cometidos durante la estimación con una incertidumbre del 15 % y 108 nodos fijos en el garaje

la posición estimada está desplazada una plaza a la izquierda. No sucede lo mismo con la *trayectoria B*, por el hecho de que la plaza de aparcamiento es horizontal en vez de vertical, ya que el error si es grande. Si el error cometido en la posición final de esta trayectoria fuese hacia la derecha en vez de hacia la izquierda, pueden ver en la *figura 4.5 a)* que la posición estimada estaría desplazada dos plazas hacia la izquierda.

Se entiende que esta volatilidad del error es debida a la posición del vehículo respecto a los nodos más cercanos. Es decir, se produce debido a que ocurre lo mismo que en los desplazamientos verticales de la *trayectoria A*.

Sin embargo, la posición del vehículo en un garaje puede ser cualquiera, exceptuando las columnas y detrás de las paredes. Usando sólo la medición de la *RSSI* es imposible saber cuales son los tres nodos más cercanos que además envuelvan al vehículo.

Para este caso, la única forma que hay de solucionar el problema es aumentando la densidad de nodos.

Por ello, incrementamos en un 50 % la cantidad de nodos fijos en el garaje, colocando en total 162 nodos fijos más los nodos extra añadidos en las paredes. Volvemos a realizar la simulación manteniendo una incertidumbre del 15 % y analizamos las diferencias.

Vemos en la *figura 4.7* una gran mejoría de la trayectoria estimada respecto al caso anterior. Usando el algoritmo que incrementa las circunferencias se consigue una trayectoria estimada prácticamente igual a la real.

La precisión también ha mejorado en el caso de aplicar triangularización, pues al haber más nodos, en más triángulos se puede dividir el mapa. Sin embargo, durante el recorrido horizontal de ambas trayectorias se ve como se estima la misma posición para varias muestras, lo que incrementa los errores cometidos.

Se muestran en la *figura 4.8* los errores que se cometen a lo largo de la estimación de la trayectoria en este caso.

Mientras que el error medio rondaba los 1,5metros con 108 nodos fijos colocados, incrementando dicha cantidad en un 50 % se consigue reducir el error medio en un porcentaje aún mayor, puesto que en este caso se cometen errores del rango del 0,5metros.

En concreto, los errores medios usando triangulación se encuentran poco por encima de dicha cifra y los errores medios cometidos por el método que incrementa los círculos está

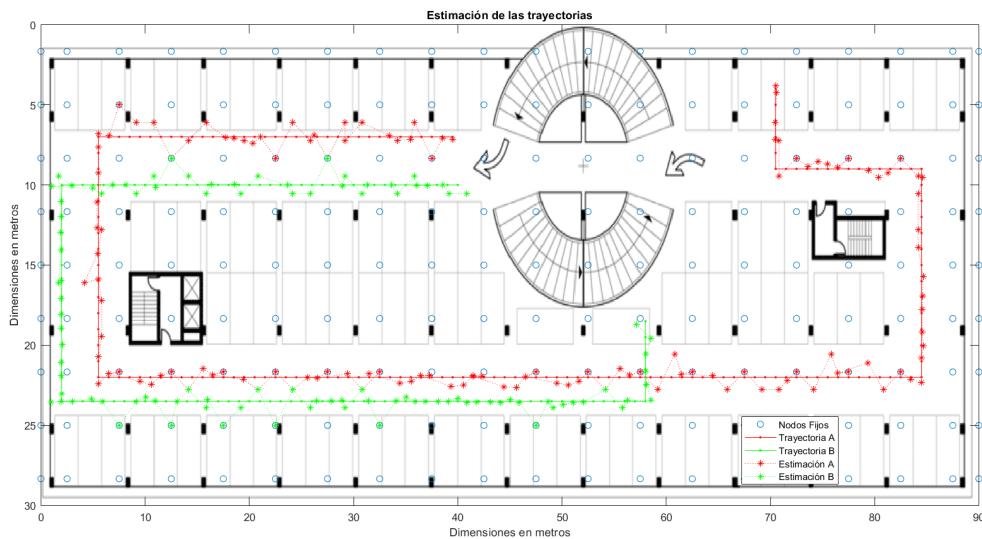
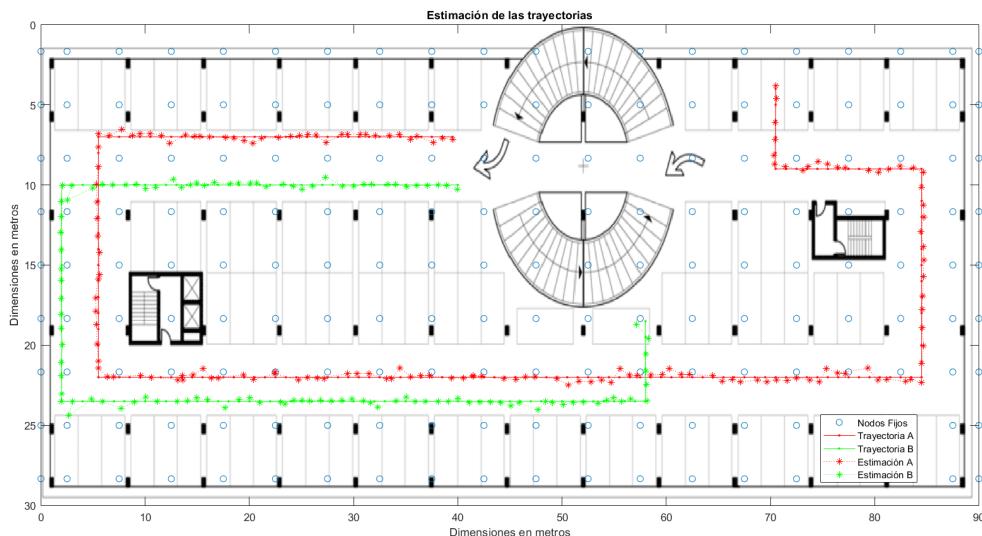
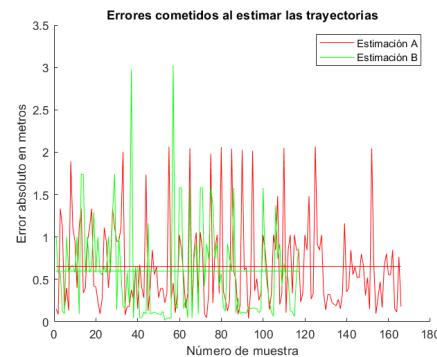
(a) Usando *CalculateLocalizationTheory*(b) Usando *CalculateLocalization*

Figura 4.7: Trayectorias y estimación con una incertidumbre del 15 % con 162 nodos fijos en el garaje

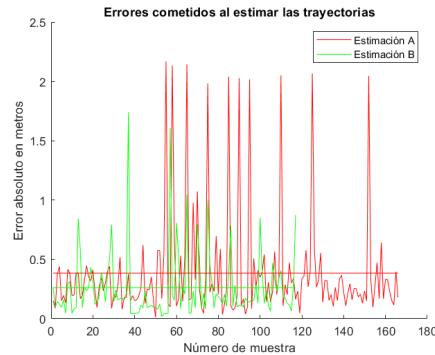
poco por debajo de dicha cifra.

El resultado más interesante es que se ha reducido la volatilidad. La función *CalcuateLocalizationTheory* tiene un error máximo cerca de los *3metros*, una cantidad inferior a los *5metros* de antes. El error máximo implementando la función *CalculateLocalization* supera por muy poco los *2metros*.

Esos son valores que permiten realizar la localización del vehículo con precisión, en todo momento. Por lo que consideramos que estos son los nodos fijos necesarios dentro del aparcamiento para conseguir implementar un sistema *ILS* que funcione correctamente. Si en un mapa de  $2700m^2$  se colocan 164 nodos fijos, es el equivalente de colocar 1 nodo cada  $16m^2$ . por lo que esta es la densidad de nodos necesaria para implementar el sistema en un garaje.



(a) Usando *CalculateLocalizationTheory*



(b) Usando *CalculateLocalization*

Figura 4.8: Errores cometidos durante la estimación con una incertidumbre del 15 % y 164 nodos fijos en el garaje

Habiendo comprobado que es mejor el algoritmo que incrementa el radio de las circunferencias y determinada la densidad de nodos necesaria, veamos las diferencias que se presentan con un incertidumbre más baja, en concreto más cercana al caso real, del 10 %. En la *figura 4.9* se representan los errores a lo largo de la estimación de la trayectoria teniendo en cuenta las cuatro posibilidades. Incertidumbre del 15 % y densidades de  $\frac{1\text{nodo}}{25m^2}$  y  $\frac{1\text{nodo}}{16m^2}$  para las *figuras 4.9 a) y b)* respectivamente mientras que incertidumbre del 10 % y densidades de  $\frac{1\text{nodo}}{25m^2}$  y  $\frac{1\text{nodo}}{16m^2}$  para las respectivas *figuras 4.9 c) y d)*.

Haciendo una comparación de los resultados obtenidos es fácil realizar que el factor que más influye es la densidad de nodos, en comparación a la incertidumbre, puesto que los errores obtenidos en la *figura 4.9 c)* están muy por encima de los errores cometidos en la *figura 4.9 b)* aunque la incertidumbre en esta es superior.

La influencia de la incertidumbre se aprecia al comparar la *figura 4.9 a) con c)* y la *figura 4.9 b) con d)*. En ambas ocasiones el error medio de ambas trayectorias disminuye ligeramente. Sin embargo, el error máximo no se ve afectado, por lo que la volatilidad del error es muy elevado para el sistema *ILS* teniendo una densidad de  $\frac{1\text{nodo}}{25m^2}$  incluso con una incertidumbre usual alrededor del 10 %.

La nueva conclusión que se obtiene revisando las simulaciones realizadas en esta sección es la importancia de determinar las posiciones conflictivas, las que tienen un error máximo, y reducir dicho máximo error a un valor adecuado para el sistema.

En el *capítulo 3*, al determinar la densidad de nodos necesaria, lo hicimos mediante la curva del error medio. Al implementar dicha densidad en una aplicación real hemos visto que hay ciertas posiciones conflictivas, que son en las que el error es máximo, para las cuales nuestro sistema no funciona adecuadamente.

La solución tomada ha sido la de aumentar la densidad de nodos de forma que al menos dos de los nodos fijos que se usan para localizar estén a diferentes laterales del objeto. Esta solución no es única, se podría usar la densidad inicial de  $\frac{1\text{nodo}}{25m^2}$  que ofrece un error medio aceptable e implementar otros sistemas para resolver las posiciones conflictivas, bajando el error medio.

Sin embargo, esto supone implementar un sistema adicional que se encargue de elegir los nodos fijos para los cuales se calcule la distancia.

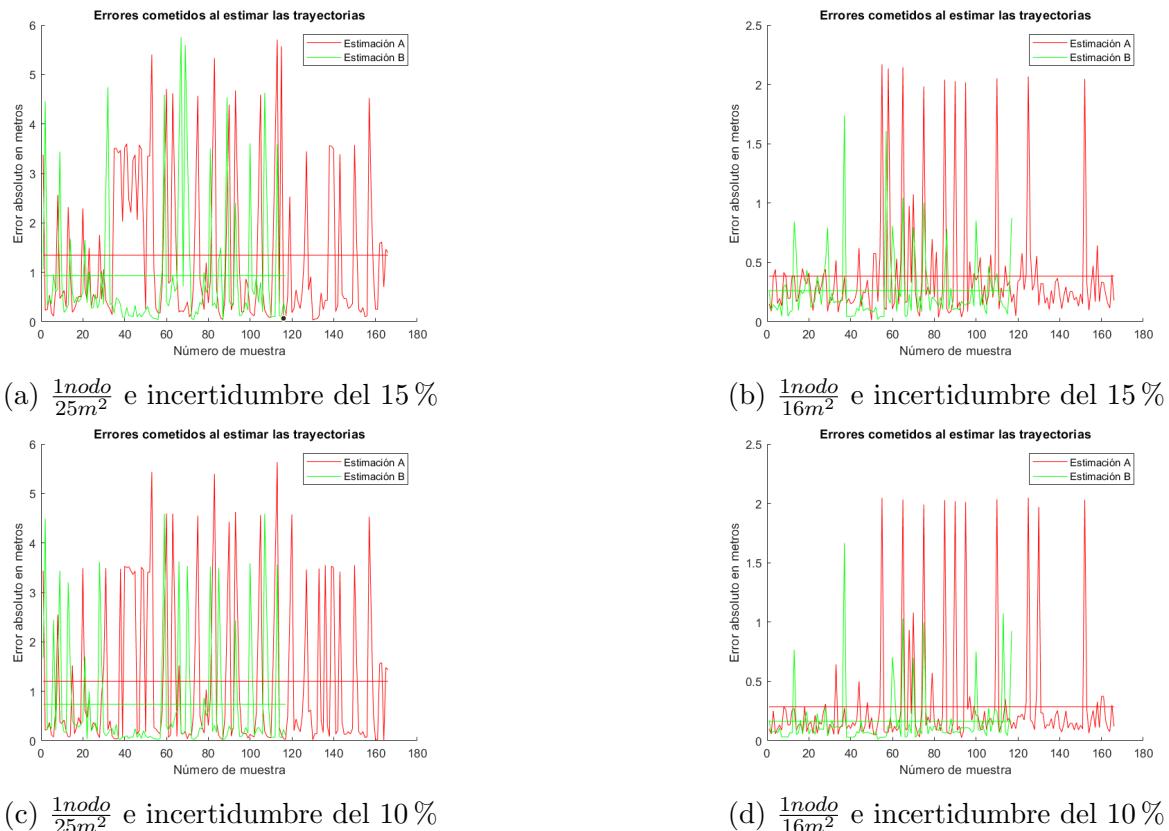


Figura 4.9: Errores durante la estimación de las trayectorias a distintas incertidumbres y densidades de nodos

### 4.3. Estimación de costes

Para concluir con este capítulo práctico, veremos el precio de implementar el sistema *ILS* en el aparcamiento explicado en la sección superior. Para ello se considerará por separado los materiales y la mano de obra.

El sistema sigue una topología en *cluster*, donde los nodos coordinadores serán módulos *XBee S2C 802.15.4*. Para poder intercomunicar estos módulos entre sí será necesario usar un hardware de control y procesamiento, para el cual se ha elegido la placa *Arduino UNO* por ser de código abierto lo que abarata costes. Para poder conectar los módulos XBee con la placa Arduino se ha de comprar un hardware intermedio denominado *Shield XBee para Arduino*.

El Arduino se ha de conectar a la toma de corriente eléctrica. En el *anexo B* vemos que la alimentación recomendada para estas placas está entre 7V y 12V así que se usara el transformador de corriente *HLK-PM12 AC/DC 220V/12V* para alimentar a cada nodo fijo.

Así mismo, será necesaria la instalación de cableado para la alimentación de todo el hardware. Como línea general de alimentación se usará un cable negro *H07V-K* de  $1,5mm^2$  de sección que permite su uso para aplicaciones de hasta 2300 watios. Para unir todos los nodos fijos se necesitarían 870m de cable, lo que redondearemos a 900m por las posibles ocurrencias inesperadas. Para unir el transformador con la placa Arduino, se usará un cable unipolar negro de  $0,5mm^2$  de sección *H05V-K* del cual se comprará 100m puesto que es la cantidad mínima.

Los nodos finales serán los vehículos dentro del aparcamiento, que estarán emitiendo

continuamente una señal la cual los nodos fijos determinarán su potencia para implementar la localización. Dichos nodos finales, que serán los objetos que el sistema ha de detectar, se implementarán como tarjetas que se le proporciona al vehículo con la entrada en el aparcamiento. Para conseguir que el dispositivo emisor sea tan pequeño, se hará uso del chip *ATSAMR21E16A* fabricado por *Microchip* que cumple con el estándar *IEEE 802.15.4* que usa los módulos XBEE y tiene un transmisor de RF a  $2,4GHz$  integrado. Vemos en la ficha de datos de este dispositivo, *anexo B*, que necesita una alimentación de entre  $1,8V$  y  $3V$  por lo que se usarán pilas de botón de  $3V$ , en concreto *Eunicell CR2032*. Para conectar este microchip con la pila se necesitará de un circuito integrado, que en este caso ha de ser fino y flexible. Suponiendo que el tamaño del circuito impreso tiene la dimensión de una tarjeta de crédito  $50x80mm$ , haremos los pedidos a la empresa *PCB4MAKER* que ofrecen presupuesto en función de las dimensiones del circuito que se desee imprimir.

Por último, como nodo coordinador principal se usará un equipo de sobremesa *PcCom Basic Office Pro* que tiene un procesador *Intel Core i5-7400* con  $8GB$  de memoria ram y  $240GB$  de disco duro SSD. Toda la programación se realizaría usando el lenguaje Python, puesto que así las herramientas de programación son de código abierto, lo que abarata costes. Este equipo se complementará con el pack de teclado y ratón *Logitech Desktop MK120* junto con un monitor *BenQ GL2580H* de  $24,5$  pulgadas.

El precio total de todos estos materiales, con las cantidades estimadas e incluyendo el I.V.A puede verse en la *tabla 4.1*. Se han considerado 180 nodos fijos, equivalentes a los 162 colocados equidistantemente en el mapa más los 18 colocados en las paredes laterales. Como el aparcamiento tiene 106 plazas, se han considerado 106 nodos móviles o finales. El precio total de todos los materiales estaría muy cerca de los **10500 €**.

| Producto                           | Unidades   | Precio              | Total             |
|------------------------------------|------------|---------------------|-------------------|
| XBee S2C 2.4GHz                    | 180 unids. | 14 €/unid           | 2664 €            |
| Arduino Uno                        | 180 unids. | 17 €/unid.          | 3060 €            |
| Shield XBee V <sub>2</sub> Arduino | 180 unids. | 11.29 €/unid.       | 2032.2 €          |
| Unzib HLK-PM12 220V/12V            | 180 unids. | 3 €/unid.           | 540 €             |
| Cable H07V-K 1,5mm <sup>2</sup>    | 900 metros | 11.95 €/100m        | 107.55 €          |
| Cable H05V-K 0,5mm <sup>2</sup>    | 100 metros | 11.01 €/100m        | 11.01 €           |
| Microchip ATSAMR21E16A             | 106 unids. | 2.9 €/unid.         | 307.4 €           |
| PCB Flexible 50x85mm               | 110 unids. | 107.69 €/10unids.   | 1184.59 €         |
| Eunicell CR2032                    | 106 unids. | 0.18 €/unid.        | 19.08 €           |
| PcCom Basic Office Pro             | 1 unid.    | 397.64 €/unid.      | 397.64 €          |
| Logithec Desktop MK120             | 1 unid.    | 15.99 €/unid.       | 15.99 €           |
| BenQ GL2580H 24.5"                 | 1 unid.    | 128 €/unid.         | 128 €             |
|                                    |            | <b>Precio Total</b> | <b>10467.46 €</b> |

Tabla 4.1: Precios de los materiales

En cuanto a la mano de obra, para estimar su coste se han dividido las tareas de forma que se realice una por día, considerando un día de trabajo equivalente a una jornada laboral de 8 horas. Se considerará que el trabajo lo realiza un ingeniero técnico industrial, que conlleva un coste de  $36 €/h$ , junto a un equipo de 2 ayudantes cuales cobrarán  $15 €/h$  brutos cada uno. Así, el coste de cada hora de trabajo será de  $66 €$ . Vean en la *tabla*

| Descripción tarea              | Horas | Coste        | Total         |
|--------------------------------|-------|--------------|---------------|
| Instalación Eléctrica          | 8 h   | 66 €/h       | 528 €         |
| Intalación de los nodos fijos  | 8 h   | 66 €/h       | 528 €         |
| Adaptación sistema de tarjetas | 8 h   | 66 €/h       | 528 €         |
| Configurar Sistema             | 8 h   | 66 €/h       | 528 €         |
| Pruebas y Ajustes              | 8 h   | 66 €/h       | 528 €         |
|                                |       | Precio Total | <b>3168 €</b> |

Tabla 4.2: Coste de la mano de obra

4.2 la subdivisión de tareas y el coste de cada una. El coste total de la mano de obra es próximo a los **3200 €**.

Considerando los costes redondeados, el precio total para la implementación de este sistema *ILS* en el garaje sería de 13700 € el cual también redondeamos para incluir posibles gastos inesperados.

En resumen, instalar un sistema *ILS* en el garaje de 2700m<sup>2</sup> con 106 plazas de aparcamiento y colocando una densidad de nodos de  $\frac{1 \text{nodo}}{16 \text{m}^2}$ , conlleva un coste aproximado de **14000 €**.

Hemos visto que dicha densidad de nodos ofrece una precisión suficiente para poder definir la trayectoria del vehículo e indicar en qué plaza se ha aparcado este. El coste puede parecer caro para una comunidad de vecinos, sin embargo, no sería un gasto elevado para una obra de uso público como un aparcamiento central, un centro comercial o un aeropuerto. Siendo estos los sitios de interés para la colocación de estos sistemas.

Ahora bien, habría que realizar un estudio y ver si conviene conviene colocar tantos nodos fijos a lo largo del garaje, puesto que el coste de los dispositivos receptores, XBee, Arduino y Shield es de 7756€, más de la mitad del presupuesto.

En unos aparcamientos públicos podríamos reducir el número de nodos fijos y complementar la falta de precisión usando el sistema de indicación de plaza ocupada como complemento. Así, cuando el vehículo aparcá y el sistema *ILS* indica una plaza distinta a la real, dicha información se contrasta con el sensor de plaza ocupada, corrigiendo la posición del vehículo. Esto reduciría el coste de implementación en más de un 25 % y seguiría ofreciendo una precisión suficiente, pues vimos en la figura 4.5 que la precisión con 108 nodos fijos implementados era lo suficientemente buena como para predecir la trayectoria, sólo que la volatilidad es tan elevada que no se puede asegurar una estimación correcta del sitio donde se ha aparcado.

El problema de este sistema serían si dos vehículos aparcán a la vez en plazas muy cercanas, una cosa muy poco probable en un aparcamiento, por lo que considero que obtendríamos unos resultados muy positivos.

Por último, hemos de hacer énfasis en que el coste estimado en esta sección es, como su nombre indica, una tasación aproximada en la cual no se han tomado en cuenta muchos factores que sí se deberían tomar para la presentación de un presupuesto. Por ejemplo, se ha dado por hecho que el cable de 1,5mm<sup>2</sup> aguanta la potencia suficiente para la instalación, puesto que el consumo de los nodos fijos es muy pequeño en comparación con los puntos de luz para lo que está pensado dicho cable. Aún así, se ha de justificar

con los cálculos pertinentes para la presentación de un proyecto real.

Lo mismo sucede con la elección del ordenador. Se da por hecho que la potencia de procesamiento del procesador es suficiente, sin embargo, para ello habría que ver las características del mismo y calcular si tiene la capacidad de refrescar las posiciones de los vehículos tantas veces como sea necesario.

Esto, a su vez, viene determinado por la cantidad de tráfico que hay dentro del garaje usualmente, y así, infinidad de factores que afectan y son dependientes de otros no considerados al hacer el cálculo de los costes, puesto que este es un trabajo de investigación sobre el funcionamiento de la localización a través de potencia de la señal de radio *RSSI*.

# Capítulo 5

## Conclusiones

Una vez conocidas todas las tecnologías disponibles para la implementación de un sistema *ILS* se determina que dentro de edificios lo más preferible es implementar un algoritmo de localización por *RSSI*, ya que este ofrece una mayor inmunidad a las reflexiones, no necesita de un array de antenas, tampoco necesita determinar el ángulo de lлагada y no necesita una sincronización de los relojes internos de los dispositivos. Además, es una medida común a todas las tecnologías que trabajen con ondas de radio, que casualmente son las tecnologías de comunicación más habituales entre los dispositivos.

La gran incógnita aparece cuando detectamos la necesidad de conocer las pérdidas que sufre la señal en su recorrido para poder calcular la distancia. Dichas pérdidas son dinámicas y dependen de factores no constantes con el tiempo. Vimos que la mayoría de artículos de investigación en este campo tratan sobre como mejorar la estimación del coeficiente de pérdidas, siendo el artículo más reciente hasta la fecha el publicado por [7] en Abril de este mismo año 2019.

Así que decidimos determinar la importancia de la estimación de dicho coeficiente.

### 5.1. Principales Aprendizajes

Comparando los distintos algoritmos de localización vimos que el factor que más afecta al error cometido al estimar la posición de un objeto es la distancia que hay entre este y el nodo fijo más alejado.

Según los resultados obtenidos en la *sección 3.2* indiferentemente de cual sea el algoritmo de localización empleado, la magnitud del error queda limitada por la distancia al nodo fijo más lejano.

Los resultados de las simulaciones en el *capítulo 3* muestran que esto es cierto incluso si dos de los nodos se encuentran pegados al objeto. Es preferible que los tres nodos más cercanos se encuentren a 5 metros cada uno, que tener dos nodos a 1 metro y e tercero a 10 metros.

Además, descubrimos que es preferible asignar un valor alto a  $\alpha_s$  ya que el error crece más lentamente, sin embargo, realizar esto supone tener un sistema que trabaja con error desde el primer momento, por lo que surgió la cuestión de la densidad de nodos.

Al principio daba la impresión de que era suficiente con una densidad de  $\frac{1nodo}{25m^2}$  puesto que los errores se estabilizan a partir de ese punto. Sin embargo, con la realización de la aplicación práctica en el *capítulo 4* determinamos la existencia de puntos conflictivos,

puntos para los cuales el error es máximo, que pueden hacer que el sistema *ISL* no sea práctico en algunas ocasiones, así que concluimos que es importante acotar también el error máximo dentro de un rango aceptable, no solo el medio.

Así, se determina que para la aplicación ejemplo del garaje se necesita una densidad de  $\frac{1\text{nodo}}{16m^2}$  para que la determinación de la trayectoria sea precisa en todo momento.

Para el caso del supermercado, si se precisa conocer en todo momento el pasillo en el que se encuentra el cliente, dicha densidad ha de aumentar, puesto que obtener error máximo cercano a los 2 metros como en el aparcamiento no es una precisión suficiente. Sin embargo, dicha aplicación no es tan crítica como determinar en qué sitio se encuentra el coche en un aparcamiento, lo peor que puede pasar es que la publicidad que llega en el móvil sea de un producto que se encuentre en el pasillo de al lado. En todo caso, respondiendo a la pregunta de la introducción, no es suficiente con la cantidad de repetidores WiFi instalados ya que ningún edificio tiene instalado un repetidor cada  $16m^2$ , sería económicamente ineficiente. En este caso, sería necesario complementar las técnicas de localización basadas en *RSSI* con otras alternativas, como por ejemplo las basadas en el tiempo de propagación.

Concluyendo con todo el trabajo realizado, vimos que implementar un sistema *ILS* que tenga la densidad de nodos necesaria es lo suficientemente caro, incluso empleando tecnología barata, como para plantearse estudiar métodos y alternativas que aumenten la precisión reduciendo la cantidad de nodos fijos a instalar, ya que estos suponen un gran porcentaje del sistema.

Viendo los resultados de las simulaciones en el *capítulo 4* se concluye que escoger los tres nodos fijos tan sólo por el nivel de *RSSI* que reciben es problemático, ya que en ocasiones, los tres nodos más próximos al objeto se encuentran desplazados a un mismo lateral, situación en la cual la precisión del sistema es muy reducida.

Por último, respondemos a la cuestión principal planteada en este *TFG* confirmando la necesidad de estudiar técnicas alternativas para estos sistemas.

## 5.2. Líneas futuras

Visto que tener una densidad de nodos elevada es costoso, sin embargo, el factor más importante para la determinación del error es la distancia y la posición relativa entre el objeto y los nodos fijos, sería de interés estudiar la combinación de dos métodos diferentes para la implementación del sistema *ILS*.

En el caso del garaje podría usarse el sistema de detección de plaza ocupada, ya instalado, de la manera que se ha propuesto en el *capítulo 4*. Para un centro comercial, se podría estudiar la implementación del método *ToF* con los repetidores *WiFi* junto con la instalación de unos nodos fijos que usen la potencia de la señal *RSSI*, ya que los dispositivos móviles hoy en día se sincronizan todos a la misma hora indicada por un servidor a través de internet.

Otra línea de estudio interesante sería determinar el impacto que tendría realizar un perfilado de *RSSI*, separando el mapa en cuadrículas y considerando un valor de atenuación diferente para cada cuadrícula. Como el trabajo de separar el mapa en cuadriculas ya se realiza al considerar una densidad de nodos, se podría determinar experimentalmente un valor de  $\alpha_s$  para cada superficie que englobe un nodo, así, si consideramos una densidad de  $\frac{1\text{nodo}}{16m^2}$  determinaríamos experimentalmente un valor de  $\alpha_s$  para cada  $16m^2$  del mapa.

Por último, como se ha visto que la posición del objeto con respecto a los nodos que lo rodean también es relevante, sería de interés estudiar como afecta al error las distintas formas de colocar los nodos, para determinar si hay alguna estructura determinada que nos permita reducir la cantidad de nodos fijos manteniendo el error.

Una forma de realizar este trabajo seria dividir el mapa en formas geométricas distintas al rectángulo, por ejemplo hexágonos para dividirlo en forma de panel de abejas, y colocar un nodo fijo en el centro de cada hexágono. En este caso también se debería estudiar la distribución habitual del mobiliario y determinar las zonas donde más frecuentemente se encuentran los objetos, puesto que así podríamos determinar zonas del mapa que necesiten una densidad de nodos fijos superior, implementando así sistemas *ILS* con una densidad de nodos variable por zonas.

# Bibliografía

- [1] O. G. Adewumi, K. Djouani y A. M. Kurien. “RSSI based indoor and outdoor distance estimation for localization in WSN”. En: *2013 IEEE International Conference on Industrial Technology (ICIT)*. 2013, págs. 1534-1539. DOI: 10.1109/ICIT.2013.6505900.
- [2] Amit Ajmeri. *System Integration: Field Wireless Networks*. Ed. por ISA Publications. 28 de nov. de 2012. URL: <https://www.isa.org/standards-publications/isa-publications/intech-magazine/2013/december/field-wireless-networks/>.
- [3] Ahmed Azeez Khudhair y col. “Wireless Indoor Localization Systems and Techniques: Survey and Comparative Study”. En: *Indonesian Journal of Electrical Engineering and Computer Science* 3 (ago. de 2016), págs. 392-409. DOI: 10.11591/ijeecs.v3.i2.pp392-409.
- [4] Bouzouane A. Bouchard B. et al. Bilodeau JS. “An experimental comparative study of RSSI-based positioning algorithms for passive RFID localization in smart environments”. En: *Journal Ambient Intell Human Comput* (jun. de 2017). ISSN: 1868-5145. DOI: <https://doi.org/10.1007/s12652-017-0531-3>.
- [5] Hee Chien Yee y Yusnita Rahayu. “Monitoring Parking Space Availability via Zigbee Technology”. En: *International Journal of Future Computer and Communication* 3 (dic. de 2014), págs. 377-380. DOI: 10.7763/IJFCC.2014.V3.331.
- [6] Jinze Du. “Indoor localiation techniques for wireless sensor networks”. Tesis doct. Universite de Nantes, feb. de 2018. URL: <https://hal.archives-ouvertes.fr/tel-01709236>.
- [7] M. Golestanian y C. Poellabauer. “VariLoc: Path Loss Exponent Estimation and Localization Using Multi-Range Beacons”. En: *IEEE Communications Letters* 23.4 (2019), págs. 724-727. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2019.2903042.
- [8] Alyn Griffiths. *Bordeaux car park by Brisac Gonzalez*. Ed. por dezeen. 14 de feb. de 2014. URL: <https://www.dezeen.com/2014/02/07/car-park-with-apartments-on-its-roof-by-brisac-gonzalez/>.
- [9] F. Gustafsson y F. Gunnarsson. “Positioning using time-difference of arrival measurements”. En: *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03)*. Vol. 6. 2003, págs. VI-553. DOI: 10.1109/ICASSP.2003.1201741.
- [10] Guangjie Han y col. “Localization algorithms of Wireless Sensor Networks: a survey”. En: *Telecommunication Systems* 52.4 (2013), págs. 2419-2436. ISSN: 1572-9451. DOI: 10.1007/s11235-011-9564-7. URL: <https://doi.org/10.1007/s11235-011-9564-7>.
- [11] E. Karbab y col. “Car park management with networked wireless sensors and active RFID”. En: *2015 IEEE International Conference on Electro/Information Technology (EIT)*. 2015, págs. 373-378. DOI: 10.1109/EIT.2015.7293372.

- [12] Anup Kumar Paul y Takuro Sato. “Localization in Wireless Sensor Networks: A Survey on Algorithms, Measurement Techniques, Applications and Challenges”. En: *Journal of Sensor and Actuator Networks* 6 (oct. de 2017), pág. 24. DOI: 10.3390/jsan6040024.
- [13] H. Liu y col. “Survey of Wireless Indoor Positioning Techniques and Systems”. En: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (2007), págs. 1067-1080. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.905750.
- [14] Jorge Miranda y col. “Path Loss Exponent Analysis in Wireless Sensor Networks: Experimental Evaluation”. En: jul. de 2013. DOI: 10.1109/INDIN.2013.6622857.
- [15] Cheriet Mohammed El Amine. “Localization in a Wireless Sensor Network based on RSSI and a decision tree”. En: *Przeglad Elektrotechniczny* 2013 (dic. de 2013), pág. 121. URL: [https://www.researchgate.net/publication/278682889\\_Localization\\_in\\_a\\_Wireless\\_Sensor\\_Network\\_based\\_on\\_RSSI\\_and\\_a\\_decision\\_tree](https://www.researchgate.net/publication/278682889_Localization_in_a_Wireless_Sensor_Network_based_on_RSSI_and_a_decision_tree).
- [16] Javier Silva Ortiz. *La Navegación por Satélite*. Ed. por Revista Tino. 30 de nov. de 2017. URL: <https://revista.jovenclub.cu/la-navegacion-por-satelite-satellite-navigation/>.
- [17] B. Rattanalert y col. “Problem investigation of min-max method for RSSI based indoor localization”. En: *2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2015, págs. 1-5. DOI: 10.1109/ECTICON.2015.7207057.
- [18] Ruben Sandoval, Antonio-Javier Garcia-Sanchez y Joan Garcia-Haro. “Improving RSSI-Based Path-Loss Models Accuracy for Critical Infrastructures: A Smart Grid Substation Case-Study”. En: *IEEE Transactions on Industrial Informatics* PP (nov. de 2017), págs. 1-1. DOI: 10.1109/TII.2017.2774838.
- [19] Souhila Silmi y col. *Localization Algorithms for Wireless Sensor Networks: A Review*. Inf. téc. Inderscience Enterprises, 2012.
- [20] Statista. *Número de dispositivos interconectados online en todo el mundo 2012-2020 en miles de millones*. Ed. por Statista. 2 de ago. de 2015. URL: <https://es.statista.com/estadisticas/638100/internet-de-las-cosas-numero-de-dispositivos-conectados-en-todo-el-mundo--2020/>.
- [21] Murat Torlak. *Path Loss*. Inf. téc. Univesity of Texas as Dallas, 2008. URL: <https://www.utdallas.edu/~torlak/courses/ee4367/lectureradio.pdf>.
- [22] J YogeshwariK y Gopalakrishna Poornima. “Empirical Approach to Determine Path-Loss Exponent for Indoor LOS Environment”. En: 2017. URL: <https://www.semanticscholar.org/paper/Empirical-Approach-to-Determine-Path-Loss-Exponent-YogeshwariK-Poornima/f63e05674b90aba073ad8000a499c898c79d531d>.
- [23] Faheem Safari, Athanasios Gkelias y Kin K. Leung. “A Survey of Indoor Localization Systems and Technologies”. En: *CoRR* abs/1709.01015 (2017). arXiv: 1709.01015. URL: <http://arxiv.org/abs/1709.01015>.

# Apéndice A

## Códigos de *MatLab*

### A.1. *Script ErrorsVsAlpha*

```
1 %% Distance Errors
2
3 %Definicion de parametros
4 syms estimate;
5 EmitedSignal = 5000;
6 TheoricAlpha = -2;
7 ErrorsInAlpha = (0:0.01:1.5);
8 FixedPointX = 50; % Close: 20 y Far: 50
9 FixedPointY = 50; % 20 y 50
10 KnownPositionX = 5;
11 KnownPositionY = 5;
12
13 %Calculo de la distancia en linea recta
14 Distance = FixedPointX-KnownPositionX;
15
16 % Calculo de las distancias estimadas
17 DistanceErrors = zeros(1,length(ErrorsInAlpha));
18 for i=1:1:length(ErrorsInAlpha)
19     RealAlpha = TheoricAlpha*(1+ErrorsInAlpha(i));
20     ReceivedSignal = EmitedSignal*(Distance^(RealAlpha));
21     eqn = ReceivedSignal == EmitedSignal*(estimate)^(TheoricAlpha);
22     EstimateDistance = double(solve(eqn, estimate));
23     EstimateDistance = EstimateDistance(EstimateDistance >= 0);
24     DistanceError = abs(Distance-EstimateDistance);
25     DistanceErrors(i) = DistanceError;
26 end
27
28 % Preparacion y representacion de resultados
29 Distances = zeros(length(DistanceErrors),1);
30 Distances(:)= Distance;
31
32 DistanceErrorsClose = DistanceErrors;
33 RelativeErrorsClose = CalculateRelativeErrors(Distances,
DistanceErrorsClose);
```

```
34 DistanceErrorsClosePlus = DistanceErrors;
35 RelativeErrorsClosePlus = CalculateRelativeErrors(Distances,
36     DistanceErrorsClosePlus);
37
38 DistanceErrorsFar = DistanceErrors;
39 RelativeErrorsFar = CalculateRelativeErrors(Distances,
40     DistanceErrorsFar);
41 DistanceErrorsFarPlus = DistanceErrors;
42 RelativeErrorsFarPlus = CalculateRelativeErrors(Distances,
43     DistanceErrorsFarPlus);
44 plot(ErrorsInAlpha*100, DistanceErrorsClose);
45 hold on;
46 plot(ErrorsInAlpha*100, DistanceErrorsFar);
47 title('Error en la distancia entre emisor y receptor');
48 xlabel('% Error en la estimacion de alpha');
49 ylabel('Error absoluto en la distancia, metros');
50 legend('Cerca', 'Lejos');
51
52 plot(ErrorsInAlpha(1:21)*100, DistanceErrorsClosePlus(1:21));
53 hold on;
54 plot(ErrorsInAlpha(1:21)*100, DistanceErrorsFarPlus(1:21));
55 title('Error en la distancia entre emisor y receptor');
56 xlabel('% Error en la estimacion de alpha');
57 ylabel('Error absoluto en la distancia, metros');
58 legend('Cerca', 'Lejos', 'AutoUpdate', 'off');
59 plot(ErrorsInAlpha(1:21)*100, DistanceErrorsClose(1:21), 'k:');
60 plot(ErrorsInAlpha(1:21)*100, DistanceErrorsFar(1:21), 'k:');
61
62 plot(ErrorsInAlpha*100, RelativeErrorsClose);
63 hold on;
64 plot(ErrorsInAlpha*100, RelativeErrorsFar);
65 title('Error en la distancia entre emisor y receptor');
66 xlabel('% Error en la estimacion de alpha');
67 ylabel('% Error relativo en la distancia');
68 legend('Cerca', 'Lejos');
69
70 plot(ErrorsInAlpha(1:21)*100, RelativeErrorsClosePlus(1:21));
71 hold on;
72 plot(ErrorsInAlpha(1:21)*100, RelativeErrorsFarPlus(1:21));
73 title('Error en la distancia entre emisor y receptor');
74 xlabel('% Error en la estimacion de alpha');
75 ylabel('% Error relativo en la distancia');
76 legend('Cerca', 'Lejos', 'AutoUpdate', 'off');
77 plot(ErrorsInAlpha(1:21)*100, RelativeErrorsClose(1:21), 'k:');
78 plot(ErrorsInAlpha(1:21)*100, RelativeErrorsFar(1:21), 'k:');
```

79  
80

```

81
82
83 %% Position Erros
84
85 % Definici n de parametros
86 syms estimate;
87 EmitedSignal = 5000;
88 TheoricAlpha = -2;
89 ErrorsInAlpha = (0:0.01:0.2);
90 FixedPointsX = [0 5 10]; % Close: [0 5 10] Far: [0 10 20]
91 FixedPointsY = [0 10 0]; % Close: [0 10 0] Far: [0 20 0]
92 KnownPositionX = 9; % Middle 5 ; Side 2 ; Corner 10 ; Near 9
93 ;
94 KnownPositionY = 1; % Middle 5 ; Side 5 ; Corner 10 ; Near 1
95 ;
96 % Calculo de las distancias
97 Distances = GetDistancesToFixedPoints(FixedPointsX, FixedPointsY
98 , KnownPositionX, KnownPositionY);
99
100 % Calculo de las posiciones estimadas y los errores cometidos
101 PositionErrors = zeros(1,length(ErrorsInAlpha));
102 EstimatePositionsX = zeros(1,length(ErrorsInAlpha));
103 EstimatePositionsY = zeros(1,length(ErrorsInAlpha));
104 for i=1:1:length(ErrorsInAlpha)
105     fprintf('Processing %d of %d.....',i,length(ErrorsInAlpha));
106     RealAlpha = TheoricAlpha*(1+ErrorsInAlpha(i));
107     ReceivedSignals = zeros(1,3);
108     for e=1:1:3
109         ReceivedSignal= EmitedSignal*(Distances(e)^(RealAlpha));
110         ReceivedSignals(e) = ReceivedSignal;
111     end
112     EstimateDistances = zeros(1,3);
113     for e=1:1:3
114         eqn = ReceivedSignals(e) == EmitedSignal*(estimate)^(TheoricAlpha);
115         EstimateDistance = double(solve(eqn, estimate));
116         EstimateDistance = EstimateDistance(EstimateDistance >= 0);
117         EstimateDistances(e) = EstimateDistance;
118     end
119
120 % Comentando y descomentando las funciones , se imlementan
121 % diferentes
122 % metodos de localizaci
123 Circle1 = [FixedPointsX(1) FixedPointsY(1)
124             EstimateDistances(1)];
124 Circle2 = [FixedPointsX(2) FixedPointsY(2)

```

```

        EstimateDistances(2) ];
125    Circle3 = [ FixedPointsX(3) FixedPointsY(3)
                  EstimateDistances(3) ];

126
127
128    % [ EstimatePositionX , EstimatePositionY ] =
129    % CalculateLocalizationTheory (FixedPointsX , FixedPointsY ,
130    % EstimateDistances);

131
132    % [ EstimatePositionX , EstimatePositionY ] =
133    % CalculateLocalizationCentroid (Circle1 , Circle2 , Circle3)
134    ;
135
136
137    EstimatePositionsX(i) = EstimatePositionX;
138    EstimatePositionsY(i) = EstimatePositionY;

139
140    PositionError = sqrt((KnownPositionX-EstimatePositionX)^2+
141                          KnownPositionY-EstimatePositionY)^2);
142    PositionErrors(i) = PositionError;
143    fprintf('Done.\n');

144 end

145
146 % Comparacin seg n la posicion
147 Center =PositionErrors;
148 Side = PositionErrors;
149 Corner = PositionErrors;
150 Near = PositionErrors;

151
152 plot(ErrorsInAlpha*100,Center,'b');
153 hold on;
154 plot(ErrorsInAlpha*100,Side,'r');
155 plot(ErrorsInAlpha*100,Corner,'g');
156 plot(ErrorsInAlpha*100,Near,'c');
157 title('Error en la posicin seg n distribucin');
158 xlabel('% Error en la estimacin de alpha');
159 ylabel('Error en la distancia , metros');
160 legend('Centrado' , 'Lateral' , 'Esquina' , 'Cercana')

161
162 % Comparacin seg n el mtodo empleado
163 ErrorsMaxMin = PositionErrors;
164 ErrorsCentroid = PositionErrors;
165 ErrorsLocalization = PositionErrors;

```

```

166
167 plot(ErrorsInAlpha*100,ErrorsMaxMin,'r');
168 hold on;
169 plot(ErrorsInAlpha*100,ErrorsCentroid,'b');
170 plot(ErrorsInAlpha*100,ErrorsLocalization,'g');
171 title('Error en la posicin segn mtodo');
172 xlabel('% Error en la estimacin de alpha');
173 ylabel('Error en la distancia, metros');
174 legend('MaxMin','Centroid','Combinacin')

```

## A.2. Script ErrorsPerDensity

```

1 %% ErrorsByDensity
2
3
4 % Parmetros
5 x_min = 0;
6 x_max = 50;
7 y_min = 0;
8 y_max = 50;
9 EmittedSignal = 5000;
10 LossFactor = -2;
11
12 % Definicin de los puntos fijos y mviles
13 NumberOfFixedPoints = [10 20 30 40 50 60 70 80 90 100 110 120
14 130 140 150 160 170 180 190 200];
15 NumberOfTrackingPoints = 100;
16 minAllowableTrackingPointsDistance = 0.5;
17
18 % Determinacin de los errores segn la densidad de nodos
19 MaxPositionsErrors = zeros(1, length(NumberOfFixedPoints));
20 MeanPositionsErrors = zeros(1, length(NumberOfFixedPoints));
21 MinPositionsErrors = zeros(1, length(NumberOfFixedPoints));
22 for i=1:1:length(NumberOfFixedPoints)
23 fprintf('Distribution with %d fixed points from %d', i*10, length(
24 NumberOfFixedPoints)*10);
25 [FixedPointsX, FixedPointsY] = GenerateEquidistantPoints(x_min,
26 x_max, y_min, y_max, NumberOfFixedPoints(i));
27 [TrackPointsX, TrackPointsY] = GenerateTrackingPoints(5, 45, 5,
28 45, FixedPointsX, FixedPointsY, NumberOfTrackingPoints,
29 minAllowableTrackingPointsDistance);
30 EstimateTrackPointsX = zeros(1, length(TrackPointsX));
31 EstimateTrackPointsY = zeros(1, length(TrackPointsY));
32
33 for e=1:1:length(TrackPointsX)
34 fprintf('Processing position %d of %d....', e, length(
35 TrackPointsX));
36 TrackPointX = TrackPointsX(e);
37 TrackPointY = TrackPointsY(e);
38 [ClosestPointsX, ClosestPointsY, ~, EstimateDistances] =

```

```

        GetClosestPoints (EmittedSignal , LossFactor ,
    FixedPointsX , FixedPointsY , TrackPointX , TrackPointY)
    ;
33 Circle1 = [ ClosestPointsX (1) ClosestPointsY (1)
    EstimateDistances (1) ];
34 Circle2 = [ ClosestPointsX (2) ClosestPointsY (2)
    EstimateDistances (2) ];
35 Circle3 = [ ClosestPointsX (3) ClosestPointsY (3)
    EstimateDistances (3) ];
36 [ EstimatePositionX , EstimatePositionY ] =
    CalculateLocalization (Circle1 , Circle2 , Circle3 );
37 EstimateTrackPointsX (e) = EstimatePositionX ;
38 EstimateTrackPointsY (e) = EstimatePositionY ;
39 fprintf( 'Done.\n' );
40 end
41
42 [ MaxPositionError , MeanPositionError , MinPositionError ] =
    CalculateErrorsPerDensity (TrackPointsX , TrackPointsY ,
    EstimateTrackPointsX , EstimateTrackPointsY );
43
44 MaxPositionsErrors ( i ) = MaxPositionError ;
45 MeanPositionsErrors ( i ) = MeanPositionError ;
46 MinPositionsErrors ( i ) = MinPositionError ;
47
48 end
49
50 % Representacion de resultados
51 plot(FixedPointsX , FixedPointsY , 'o')
52 hold on;
53 plot(TrackPointsX , TrackPointsY , 'x')
54 axis([ x_min , x_max , y_min , y_max ]);
55
56 figure(EquidistantPlot)
57 plot(NumberOfFixedPoints , MaxPositionsErrors);
58 hold on;
59 plot(NumberOfFixedPoints , MeanPositionsErrors);
60 hold on;
61 plot(NumberOfFixedPoints , MinPositionsErrors);
62 title('Error medio seg n la densidad de nodos');
63 xlabel('Densidad de nodos en el mapa');
64 ylabel('Error en la posici n , metros');
65 legend('Error Mximo' , 'Error Medio' , 'Error Mnimo')

66
67 % Restore data from plot
68 h = findobj(gca , 'Type' , 'line ')
69 NodosFijos=get(h , 'Xdata ')
70 MediumErrors5=get(h , 'Ydata ')
71
72 h = findobj(gca , 'Type' , 'line ')
73 MediumErrors10=get(h , 'Ydata ')

```

```

74
75 h = findobj(gca , 'Type' , 'line ')
76 MediumErrors15=get(h , 'Ydata ')
77
78 h = findobj(gca , 'Type' , 'line ')
79 MediumErrors20=get(h , 'Ydata ')
80
81 plot(NodosFijos , MediumErrors20 , 'r ')
82 hold on;
83 plot(NodosFijos , MediumErrors15 , 'g ')
84 plot(NodosFijos , MediumErrors10 , 'b ')
85 plot(NodosFijos , MediumErrors5 , 'c ')
86 title('Errores medios seg n la densidad de nodos ');
87 xlabel('Densidad de nodos en el mapa ');
88 ylabel('Error en la posicin , metros ');
89 legend('20%' , '15%' , '10%' , '5%')
90
91 %% Delimitacion de errors
92
93 % Par metros
94 x_min = 0;
95 x_max = 20;
96 y_min = 0;
97 y_max = 20;
98 EmittedSignal = 5000;
99 LossFactor = -2;
100
101 % Generacin de puntos fijos
102 NumberOfFixedPoints = 30; % 8, 16, 32
103 [ FixedPointsX , FixedPointsY ] = GenerateEquidistantPoints(x_min ,
    x_max , y_min , y_max , NumberOfFixedPoints );
104
105 % Generacion de puntos miles
106 NumberOfTrackingPoints = 100;
107 minAllowableTrackingPointsDistance = 0.5;
108 [ TrackPointsX , TrackPointsY ] = GenerateTrackingPoints(5 , 15 , 5 ,
    15 , FixedPointsX , FixedPointsY , NumberOfTrackingPoints ,
    minAllowableTrackingPointsDistance );
109
110 % Estimacin de la posicin de cada punto m vil
111 EstimateTrackPointsX = zeros(1 , NumberOfTrackingPoints );
112 EstimateTrackPointsY = zeros(1 , NumberOfTrackingPoints );
113 hold off
114
115 for i=1:length(TrackPointsX)
116     fprintf('Processing %d of %d.....' , i , length(TrackPointsX
        ));
117     TrackPointX = TrackPointsX(i);
118     TrackPointY = TrackPointsY(i);
119     [ ClosestPointsX , ClosestPointsY , ~ , EstimateDistances ] =

```

```

        GetClosestPoints (EmittedSignal , LossFactor ,
    FixedPointsX , FixedPointsY , TrackPointX , TrackPointY)
    ;
120 Circle1 = [ ClosestPointsX (1) ClosestPointsY (1)
    EstimateDistances(1) ];
121 Circle2 = [ ClosestPointsX (2) ClosestPointsY (2)
    EstimateDistances(2) ];
122 Circle3 = [ ClosestPointsX (3) ClosestPointsY (3)
    EstimateDistances(3) ];
123 [ EstimatePositionX , EstimatePositionY ] =
    CalculateLocalization (Circle1 , Circle2 , Circle3 );
124 EstimateTrackPointsX (i) = EstimatePositionX ;
125 EstimateTrackPointsY (i) = EstimatePositionY ;
126 fprintf ('Done.\n');
127 end
128 % Calculo de los errores en funcin de la incertidumbre
129 PositionErrors20 = CalculatePositionErrors (TrackPointsX ,
    TrackPointsY , EstimateTrackPointsX , EstimateTrackPointsY );
130 MeanErrors20 = mean (PositionErrors20);
131 MeanErrors20 = repelem (MeanErrors20 , length (PositionErrors20));
132
133 PositionErrors15 = CalculatePositionErrors (TrackPointsX ,
    TrackPointsY , EstimateTrackPointsX , EstimateTrackPointsY );
134 MeanErrors15 = mean (PositionErrors15);
135 MeanErrors15 = repelem (MeanErrors15 , length (PositionErrors15));
136
137 PositionErrors10 = CalculatePositionErrors (TrackPointsX ,
    TrackPointsY , EstimateTrackPointsX , EstimateTrackPointsY );
138 MeanErrors10 = mean (PositionErrors10);
139 MeanErrors10 = repelem (MeanErrors10 , length (PositionErrors10));
140
141 PositionErrors5 = CalculatePositionErrors (TrackPointsX ,
    TrackPointsY , EstimateTrackPointsX , EstimateTrackPointsY );
142 MeanErrors5 = mean (PositionErrors5);
143 MeanErrors5 = repelem (MeanErrors5 , length (PositionErrors5));
144
145 % Representacin de resultados
146 figure (EquidistantPlot)
147
148 plot ((1:1:NumberOfTrackingPoints) , PositionErrors20 , 'b');
149 hold on;
150 plot ((1:1:NumberOfTrackingPoints) , PositionErrors15 , 'r');
151 plot ((1:1:NumberOfTrackingPoints) , PositionErrors10 , 'g');
152 plot ((1:1:NumberOfTrackingPoints) , PositionErrors5 , 'c');
153 legend ('20%', '15%', '10%', '5%', 'AutoUpdate' , 'off')
154 plot ((1:1:NumberOfTrackingPoints) , MeanErrors20 , 'b');
155 plot ((1:1:NumberOfTrackingPoints) , MeanErrors15 , 'r');
156 plot ((1:1:NumberOfTrackingPoints) , MeanErrors10 , 'g');
157 plot ((1:1:NumberOfTrackingPoints) , MeanErrors5 , 'c');
158

```

```

159 title('Errores al estimar la posicin');
160 xlabel('Nmero del experimento');
161 ylabel('Error en la posicin , metros');
162
163 figure(MovedPlot)
164 plot((1:1:NumberOfTrackingPoints), PositionErrors);

```

### A.3. *Script ParkingTrajectory*

```

1
2 %% Defining Paths
3
4 % First Path
5 x1=[39.5:-1:5.5]; y1=[7]; y1=repelem(y1, length(x1));
6 y2=[7:1:22]; x2=[5.5]; x2=repelem(x2, length(y2));
7 x3=[5.5:1:84.5]; y3=[22]; y3=repelem(y3, length(x3));
8 y4=[22:-1:9]; x4=[84.5]; x4 = repelem(x4, length(y4));
9 x5=[84.5:-1:70.5]; y5=[9]; y5=repelem(y5, length(x5));
10 y6=[9:-1:4]; x6=[70.5]; x6=repelem(x6, length(y6));
11 FirstTrajectoryX=[x1 x2 x3 x4 x5 x6];
12 FirstTrajectoryY=[y1 y2 y3 y4 y5 y6];
13
14 % Second Path
15 x1=[40:-1:2]; y1=[10]; y1=repelem(y1, length(x1));
16 y2=[10:1:23]; y2=[y2 23.5]; x2=[2]; x2=repelem(x2, length(y2));
17 x3=[2:1:58]; y3=[23.5]; y3=repelem(y3, length(x3));
18 y4=[23.5:-1:18.5]; x4=[58]; x4=repelem(x4, length(y4));
19 SecondTrajectoryX=[x1 x2 x3 x4];
20 SecondTrajectoryY=[y1 y2 y3 y4];
21
22 %% Defining Measures and variables
23 x_min = 0;
24 x_max = 90;
25 y_min = 0;
26 y_max = 30;
27
28 % EmitterPower
29 EmittedSignal = 5000;
30
31 % Emitter Loss Factor
32 LossFactor = -2;
33
34 % Fixed nodes in Parking
35 NumberOfFixedPoints = 108; % 54 108 162
36 [FixedPointsX, FixedPointsY] = GenerateEquidistantPoints(x_min,
   x_max, y_min, y_max, NumberOfFixedPoints);
37
38 ErrorsTrajectory = figure;
39 ResultsPlot = figure;
40

```

```

41 figure(ResultsPlot);
42 parking = imread('ParkingScheme.png');
43 image([x_min x_max], [y_min y_max], parking);
44 hold on
45 plot(FixedPointsX, FixedPointsY, 'o');
46 plot(FirstTrajectoryX, FirstTrajectoryY, 'r.-');
47 plot(SecondTrajectoryX, SecondTrajectoryY, 'g.-');
48
49
50
51 %% Performing Localization
52 TrajectoryX = SecondTrajectoryX; %SecondTrajectoryX
      FirstTrajectoryX
53 TrajectoryY = SecondTrajectoryY; %SecondTrajectoryY
      FirstTrajectoryY
54
55 EstimateTrajectoryX = zeros(1, length(TrajectoryX));
56 EstimateTrajectoryY = zeros(1, length(TrajectoryY));
57
58 for i=1:1:length(TrajectoryX)
59 fprintf('Processing %d of %d....', i, length(TrajectoryX));
60 TrackPointX = TrajectoryX(i);
61 TrackPointY = TrajectoryY(i);
62 [ClosestPointsX, ClosestPointsY, ~, EstimateDistances] =
      GetClosestPoints (EmittedSignal, LossFactor, FixedPointsX,
      FixedPointsY, TrackPointX, TrackPointY);
63
64 % [EstimatePositionX, EstimatePositionY] =
      CalculateLocalizationTheory (ClosestPointsX, ClosestPointsY,
      EstimateDistances);
65
66 Circle1 = [ClosestPointsX(1) ClosestPointsY(1) EstimateDistances
      (1)];
67 Circle2 = [ClosestPointsX(2) ClosestPointsY(2) EstimateDistances
      (2)];
68 Circle3 = [ClosestPointsX(3) ClosestPointsY(3) EstimateDistances
      (3)];
69
70 [EstimatePositionX, EstimatePositionY] = CalculateLocalization (
      Circle1, Circle2, Circle3);
71
72 EstimateTrajectoryX(i) = EstimatePositionX;
73 EstimateTrajectoryY(i) = EstimatePositionY;
74 fprintf('Done.\n');
75 end
76
77 PositionErrorsA = CalculatePositionErrors(TrajectoryX,
      TrajectoryY, EstimateTrajectoryX, EstimateTrajectoryY) ;
78 MediumErrorA = mean(PositionErrorsA);
79 MediumErrorA = repelem(MediumErrorA, length(PositionErrorsA));

```

```

80 XAxisLengthA = (1:1:length(EstimateTrajectoryX));
81
82 PositionErrorsB = CalculatePositionErrors(TrajectoryX,
     TrajectoryY, EstimateTrajectoryX, EstimateTrajectoryY) ;
83 MediumErrorB = mean(PositionErrorsB);
84 MediumErrorB = repelem(MediumErrorB, length(PositionErrorsB));
85 XAxisLengthB = (1:1:length(EstimateTrajectoryX));
86
87 %% Plotting results
88
89 figure(ResultsPlot);
90 hold on;
91 plot(EstimateTrajectoryX, EstimateTrajectoryY, 'r*:' ); % ,
92     LineWidth' , 5
93 title('Estimacin de las trayectorias');
94 xlabel('Dimensiones en metros');
95 ylabel('Dimensiones en metros');
96 legend('Nodos Fijos', 'Trayectoria A', 'Trayectoria B', ,
97     'Estimacin A', 'Estimacin B');
98
99
100 figure(ERRORsTrajectory);
101 hold on;
102 plot(XAxisLengthA, PositionErrorsA, 'r');
103 plot(XAxisLengthB, PositionErrorsB, 'g');
104 legend('Estimacin A', 'Estimacin B', 'AutoUpdate', 'off');
105 plot(XAxisLengthA, MediumErrorA, 'r')
106 plot(XAxisLengthB, MediumErrorB, 'g')
107 title('Errores cometidos al estimar las trayectorias');
108 xlabel('Nmero de muestra');
109 ylabel('Error absoluto en metros');

```

## A.4. Funciones de generación de nodos

### A.4.1. *Function GenerateEquidistantPoints*

```

1 function [FixedPointsX, FixedPointsY] =
2     GenerateEquidistantPoints (x_min, x_max, y_min, y_max,
3         NumberofPoints)
4
5 NumDivisors = divisors(NumberOfPoints);
6 DivisorSelected = round((length(NumDivisors)/2));
7 Ydivisor = NumDivisors(DivisorSelected);
8 Xdivisor = NumberofPoints/Ydivisor;
9
10 SeparationPointsX = linspace(x_min, x_max, Xdivisor+1);
11 SeparationPointsY = linspace(y_min, y_max, Ydivisor+1);
12
13 FixedPointsY = zeros(1,length(SeparationPointsY))*length(
14     SeparationPointsY));

```

```

12     sumTermY=0;
13     for i=2:1:length(SeparationPointsY)
14         FixedPointsY(i-1) = (((SeparationPointsY(i)-
15             SeparationPointsY(i-1))/2)+sumTermY);
16         sumTermY = sumTermY + (SeparationPointsY(i)-
17             SeparationPointsY(i-1));
18     end
19     FixedPointsY = FixedPointsY(FixedPointsY~=0);
20     FixedPointsY = repmat(FixedPointsY, Xdivisor+2);
21     FixedPointsY = FixedPointsY(1:1,:);

22     disalignment = 0.001;
23     for i=1:1:length(FixedPointsY)
24         disalignment = disalignment*(-1);
25         FixedPointsY(i) = FixedPointsY(i)+disalignment;
26     end

27     FixedPointsX = zeros(1,length(SeparationPointsX)*length(
28         SeparationPointsX));
29     sumTermX=0;
30     for i=2:1:length(SeparationPointsX)
31         FixedPointsX(i-1) = (((SeparationPointsX(i)-
32             SeparationPointsX(i-1))/2)+sumTermX);
33         sumTermX = sumTermX+(SeparationPointsX(i)-SeparationPointsX(
34             i-1));
35     end
36     FixedPointsX = FixedPointsX(FixedPointsX~=0);
37     FixedPointsX = [x_min FixedPointsX x_max];
38     FixedPointsX = repelem(FixedPointsX, Ydivisor);

39     disalignment = 0.001;
40     for i=1:1:length(FixedPointsX)
41         disalignment = disalignment*(-1);
42         FixedPointsX(i) = FixedPointsX(i)+disalignment;
43     end
44 end

```

### A.4.2. *Function GenerateTrackingPoints*

```

1 % This function generates the point whose position shoud be
2     calculated
3 function [TrackPointsX, TrackPointsY] = GenerateTrackingPoints(
4     x_min, x_max, y_min, y_max, FixedPointsX, FixedPointsY,
5     NumberOfPoints, minAllowableDistance)

6 % Defining random generator with seed in order to be able to
7     repeat the experiment
8 rng(1, 'twister');

```

```

7      % Inline function , generates a random number between [ a and
8          b ]
9      RandomGeneratorX = @(x_min,x_max) (x_max-x_min)*rand ()+x_min
10         ;
11      RandomGeneratorY = @(y_min,y_max) (y_max-y_min)*rand ()+y_min
12         ;
13
14
15      % Array that have the positions of the tracking points
16      TrackPointsX = [] ;
17      TrackPointsY = [] ;
18
19
20      while length(TrackPointsX) < NumberOfPoints
21          % Generating Oone point
22          RandomPointX = RandomGeneratorX(x_min,x_max) ;
23          RandomPointY = RandomGeneratorY(y_min,y_max) ;
24
25          % Checking that the car was places in a corect position ,
26          % not to close from columns
27          Distances = GetDistancesToFixedPoints(FixedPointsX ,
28                                              FixedPointsY , RandomPointX , RandomPointY) ;
29          % Getting the samllest distance between al of them
30          minDistance = min(Distances) ;
31
32
33      if minDistance >= minAllowableDistance
34          TrackPointsX = [TrackPointsX RandomPointX] ;
35          TrackPointsY = [TrackPointsY RandomPointY] ;
36      end
37
38
39      end
40
41
42
43  end

```

## A.5. Funciones para el clculo de distancias

### A.5.1. *Function GetDistancesToFixedPoints*

```

1  % this function calculates the distance between a array of
2  % points and a reference point
3  function [Distances] = GetDistancesToFixedPoints(FixedPointsX ,
4                                              FixedPointsY , PointX , PointY)
5
6      Distances = zeros(1, length(FixedPointsX)) ;
7      for i=1:1:length(FixedPointsX)
8          Distances(i) = sqrt((FixedPointsX(i)-PointX)^2+
9                               FixedPointsY(i)-PointY)^2) ;
10     end
11
12
13  end

```

### A.5.2. *Function GetClosestPoints*

```

1 % This function returns the position and the distance for the
2 % three closest points to the point we track
3 function [ClosestPointsX, ClosestPointsY, ClosestDistances,
4 EstimateDistances] = GetClosestPoints (EmittedSignal,
5 LossFactor, FixedPointsX, FixedPointsY, TrackPointX,
6 TrackPointY)
7
8 syms estimate;
9
10 % Computing Distances between car and rest
11 Distances = GetDistancesToFixedPoints(FixedPointsX,
12 FixedPointsY, TrackPointX, TrackPointY);
13 Distances(Distances==0) = 1;
14
15 % Adding Kind of Error to loss factor , let 's say 10%, from
16 % -5% to +5%
17 rng(1, 'twister');
18 RandomGenerator = @(min,max) (max-min)*rand() + min;
19
20 % Computing the loss of signal from the car to the posts
21 LossFactors = zeros(1, length(Distances));
22 ReceivedSignals = zeros(1, length(Distances));
23 for i=1:length(Distances)
24     Error = RandomGenerator(-0.05,0.05);
25     RealLossFactor = LossFactors*(1+Error);
26     LossFactors(i) = RealLossFactor;
27     ReceivedSignal = EmittedSignal*(Distances(i)^(RealLossFactor));
28     ReceivedSignals(i) = ReceivedSignal;
29 end
30 SortedSignals = sort(ReceivedSignals, 'descend');
31
32 % Getting the indexes of the closest columns
33 Indexes = zeros(1,3);
34 for i=1:3
35     Index = find(ReceivedSignals == SortedSignals(i));
36     Indexes(i) = Index;
37 end
38
39 % Finally , we found what three columns are closer to the car
40 % broda
41 ClosestPointsX = zeros(1,3);
42 ClosestPointsY = zeros(1,3);
43 for i=1:3
44     ClosestPointsX(i) = FixedPointsX(Indexes(i));
45     ClosestPointsY(i) = FixedPointsY(Indexes(i));
46 end

```

```

41     ClosestDistances = zeros(1,3);
42     for i=1:1:3
43         ClosestDistances(i) = Distances(Indexes(i));
44     end
45
46 % This is not used, made just in case we could use this data
47 % for something
47 FactorsUsedClosestDistances = zeros(1,3);
48     for i=1:1:3
49         FactorsUsedClosestDistances(i) = LossFactors(Indexes(i))
50         ;
51     end
52
53 % Adding The Error To the distances ass well
54 EstimateDistances = zeros(1,3);
55     for i=1:1:3
56         eqn = SortedSignals(i) == EmittedSignal*(estimate)^(LossFactor);
57         EstimateDistance = double(solve(eqn, estimate));
58         EstimateDistance = EstimateDistance(EstimateDistance >= 0);
59         EstimateDistances(i) = EstimateDistance;
60     end
61
62 end

```

## A.6. Funciones para la estimación de la posición

### A.6.1. *Function CalculateLocalizationTheory*

```

1 % This functions returns the position of the object, based on
2 % distance and closest fixed points
3
4 function [PositionX, PositionY] = CalculateLocalizationTheory (
5     ClosestPointsX, ClosestPointsY, ClosestDistances)
6
7 Circle1 = [ClosestPointsX(1) ClosestPointsY(1) ClosestDistances
8     (1)];
9 Circle2 = [ClosestPointsX(2) ClosestPointsY(2) ClosestDistances
10    (2)];
11 Circle3 = [ClosestPointsX(3) ClosestPointsY(3) ClosestDistances
12    (3)];
13
14 admissibleDeviation = 0.3;
15
16 [Xcross12, Ycross12] = circcirc(Circle1(1), Circle1(2), Circle1(3),
17     Circle2(1), Circle2(2), Circle2(3));
18 [Xcross13, Ycross13] = circcirc(Circle1(1), Circle1(2), Circle1(3),
19     Circle3(1), Circle3(2), Circle3(3));

```

```

13 [ Xcross23 , Ycross23 ] = circcirc( Circle2(1) , Circle2(2) , Circle2(3) ,
14 Circle3(1) , Circle3(2) , Circle3(3) ) ;
15
16 if( ( isnan( Xcross12(1) ) && isnan( Xcross12(2) ) ) || ( isnan(
17 Xcross13(1) ) && isnan( Xcross13(2) ) ) || ( isnan( Xcross23(1)
18 ) && isnan( Xcross23(2) ) ) )
% If this happens, we do the triangularization
19 x1 = Circle1(1) ;
20 y1 = Circle1(2) ;
21 x2 = Circle2(1) ;
22 y2 = Circle2(2) ;
23 x3 = Circle3(1) ;
24 y3 = Circle3(2) ;
25 Triangle = polyshape([x1 x2 x3],[y1 y2 y3]) ;
26 [PositionX , PositionY] = centroid(Triangle) ;
27 else
28
29 x1 = Xcross12(1) ;
30 y1 = Ycross12(1) ;
31 x2 = Xcross12(2) ;
32 y2 = Ycross12(2) ;
33 dx1= abs( Circle3(1)-x1) ;
34 dy1 = abs( Circle3(2)-y1) ;
35 distance1 = sqrt((dx1)^2+(dy1)^2) ;
36
37 dx2= abs( Circle3(1)-x2) ;
38 dy2 = abs( Circle3(2)-y2) ;
39 distance2 = sqrt((dx2)^2+(dy2)^2) ;
40
41
42 if ( distance1-Circle3(3) ) < admissibleDeviation
43 PositionX = x1;
44 PositionY = y1;
45 elseif ( distance2-Circle3(3) ) < admissibleDeviation
46 PositionX = x2;
47 PositionY = y2;
48 else
% Here we should Apply the max min method
49 x1 = Circle1(1) ;
50 y1 = Circle1(2) ;
51 r1 = Circle1(3) ;
52 x2 = Circle2(1) ;
53 y2 = Circle2(2) ;
54 r2 = Circle2(3) ;
55 x3 = Circle3(1) ;
56 y3 = Circle3(2) ;
57 r3 = Circle3(3) ;
58 Xmin = max([x1-r1 , x2-r2 , x3-r3]) ;
59

```

```

60      Xmax = min([x1+r1, x2+r2, x3+r3]);
61      Ymin = max([y1-r1, y2-r2, y3-r3]);
62      Ymax = min([y1+r1, y2+r2, y3+r3]);
63
64      PositionX = (Xmin+Xmax)/2;
65      PositionY = (Ymin+Ymax)/2;
66  end
67
68
69 end

```

### A.6.2. *Function MakeCirclesCross*

```

1 function [Circle1, Circle2, Circle3] = MakeCirclesCross (Circle1
, Circle2, Circle3)
2
3 [Xcross12, ~] = circcirc(Circle1(1), Circle1(2), Circle1(3), Circle2
(1), Circle2(2), Circle2(3));
4 [Xcross13, ~] = circcirc(Circle1(1), Circle1(2), Circle1(3), Circle3
(1), Circle3(2), Circle3(3));
5 [Xcross23, ~] = circcirc(Circle2(1), Circle2(2), Circle2(3), Circle3
(1), Circle3(2), Circle3(3));
6 increment = 0.01;
7
8 while ( (isnan(Xcross12(1)) && isnan(Xcross12(2))) || (isnan
(Xcross13(1)) && isnan(Xcross13(2))) || (isnan(Xcross23
(1)) && isnan(Xcross23(2))) )
9
10 %None of them ; inrement the smaller
11 if ( (isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
isnan(Xcross13(1)) && isnan(Xcross13(2))) && (isnan(
Xcross23(1)) && isnan(Xcross23(2))) )
12
13 while( (isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
(isnan(Xcross13(1)) && isnan(Xcross13(2))) &&
(isnan(Xcross23(1)) && isnan(Xcross23(2))) )
14     CircleRadios = [Circle1(3) Circle2(3) Circle3(3)
];
15     [~, SmallerCircle] = min(CircleRadios);
16     switch SmallerCircle
17         case 1
18             Circle1(3) = Circle1(3)+increment;
19         case 2
20             Circle2(3) = Circle2(3)+increment;
21         case 3
22             Circle3(3) = Circle3(3)+increment;
23     end
24     [Xcross12, ~] = circcirc(Circle1(1), Circle1(2),
Circle1(3), Circle2(1), Circle2(2), Circle2(3));
25     [Xcross13, ~] = circcirc(Circle1(1), Circle1(2),
Circle1(3), Circle3(1), Circle3(2), Circle3(3));

```

```

26      [ Xcross23 ,~] = circcirc( Circle2(1) ,Circle2(2) ,
27          Circle2(3) ,Circle3(1) ,Circle3(2) ,Circle3(3)) ;
28      end
29
30      %1-2 and 1-3 ; 2-3 yes ; increment 1
31      elseif ( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
32          isnan(Xcross13(1)) && isnan(Xcross13(2))) )
33
34          while( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
35              ( isnan(Xcross13(1)) && isnan(Xcross13(2))) )
36              Circle1(3) = Circle1(3)+increment;
37              [ Xcross12 ,~] = circcirc( Circle1(1) ,Circle1(2) ,
38                  Circle1(3) ,Circle2(1) ,Circle2(2) ,Circle2(3)) ;
39              [ Xcross13 ,~] = circcirc( Circle1(1) ,Circle1(2) ,
40                  Circle1(3) ,Circle3(1) ,Circle3(2) ,Circle3(3)) ;
41          end
42
43      %1-2 and 2-3 ; 1-3 yes ; increment 2
44      elseif ( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
45          isnan(Xcross23(1)) && isnan(Xcross23(2))) )
46
47          while( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) &&
48              ( isnan(Xcross23(1)) && isnan(Xcross23(2))) )
49              Circle2(3) = Circle2(3)+increment;
50              [ Xcross12 ,~] = circcirc( Circle1(1) ,Circle1(2) ,
51                  Circle1(3) ,Circle2(1) ,Circle2(2) ,Circle2(3)) ;
52              [ Xcross23 ,~] = circcirc( Circle2(1) ,Circle2(2) ,
53                  Circle2(3) ,Circle3(1) ,Circle3(2) ,Circle3(3)) ;
54          end
55
56      %1-2
57      elseif ( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) )
58
59          while( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) )
60              CircleRadios = [ Circle1(3) Circle2(3) ];
61              [~, SmallerCircle] = min( CircleRadios );
62

```

```

63         switch SmallerCircle
64             case 1
65                 Circle1(3) = Circle1(3)+increment;
66             case 2
67                 Circle2(3) = Circle2(3)+increment;
68             end
69
70     [ Xcross12 ,~ ] = circcirc( Circle1(1) , Circle1(2) ,
71                               Circle1(3) , Circle2(1) , Circle2(2) , Circle2(3) );
72
73
74 %1-3
75 elseif ( ( isnan(Xcross13(1)) && isnan(Xcross13(2)) ) )
76
77 while ( ( isnan(Xcross13(1)) && isnan(Xcross13(2)) ) )
78     CircleRadios = [ Circle1(3) Circle3(3) ];
79     [~, SmallerCircle] = min(CircleRadios);
80
81     switch SmallerCircle
82         case 1
83             Circle1(3) = Circle1(3)+increment;
84         case 2
85             Circle3(3) = Circle3(3)+increment;
86     end
87
88     [ Xcross13 ,~ ] = circcirc( Circle1(1) , Circle1(2) ,
89                               Circle1(3) , Circle3(1) , Circle3(2) , Circle3(3) );
90
91
92 %2-3
93 elseif ( ( isnan(Xcross23(1)) && isnan(Xcross23(2)) ) )
94
95 while ( ( isnan(Xcross23(1)) && isnan(Xcross23(2)) ) )
96     CircleRadios = [ Circle2(3) Circle3(3) ];
97     [~, SmallerCircle] = min(CircleRadios);
98
99     switch SmallerCircle
100        case 1
101            Circle2(3) = Circle2(3)+increment;
102        case 2
103            Circle3(3) = Circle3(3)+increment;
104        end
105
106     [ Xcross23 ,~ ] = circcirc( Circle2(1) , Circle2(2) ,
107                               Circle2(3) , Circle3(1) , Circle3(2) , Circle3(3) );
108
109

```

```

110     end
111     [ Xcross12 ,~] = circcirc( Circle1(1) ,Circle1(2) ,Circle1(3)
112         ,Circle2(1) ,Circle2(2) ,Circle2(3));
113     [ Xcross13 ,~] = circcirc( Circle1(1) ,Circle1(2) ,Circle1(3)
114         ,Circle3(1) ,Circle3(2) ,Circle3(3));
115     [ Xcross23 ,~] = circcirc( Circle2(1) ,Circle2(2) ,Circle2(3)
116         ,Circle3(1) ,Circle3(2) ,Circle3(3));
117     end
118 end

```

### A.6.3. Function CalculateLocalizationMaxMin

```

1 function [ PositionX , PositionY ] = CalculateLocalizationMaxMin (
2     Circle1 , Circle2 , Circle3 )
3
4     admissibleError = 0.5;
5
6     [ Xcross12 , Ycross12 ] = circcirc( Circle1(1) ,Circle1(2) ,Circle1
7         (3) ,Circle2(1) ,Circle2(2) ,Circle2(3));
8     [ Xcross13 ,~] = circcirc( Circle1(1) ,Circle1(2) ,Circle1(3) ,
9         Circle3(1) ,Circle3(2) ,Circle3(3));
10    [ Xcross23 ,~] = circcirc( Circle2(1) ,Circle2(2) ,Circle2(3) ,
11        Circle3(1) ,Circle3(2) ,Circle3(3));
12
13    x1 = Xcross12(1);
14    y1 = Ycross12(1);
15    x2 = Xcross12(2);
16    y2 = Ycross12(2);
17
18    dx1= abs( Circle3(1)-x1);
19    dy1 = abs( Circle3(2)-y1);
20    distance1 = sqrt( (dx1)^2+(dy1)^2);
21
22    dx2= abs( Circle3(1)-x2);
23    dy2 = abs( Circle3(2)-y2);
24    distance2 = sqrt( (dx2)^2+(dy2)^2);
25
26    if ( distance1-Circle3(3) ) < admissibleError
27        PositionX = x1;
28        PositionY = y1;
29    elseif ( distance2-Circle3(3) ) < admissibleError
30        PositionX = x2;
31        PositionY = y2;
32    else
33
34        if ( ( isnan(Xcross12(1)) && isnan(Xcross12(2))) || (
35            isnan(Xcross13(1)) && isnan(Xcross13(2))) || (isnan(
36            Xcross23(1)) && isnan(Xcross23(2))) )
37            [ Circle1 , Circle2 , Circle3 ] = MakeCirclesCross (
38                Circle1 , Circle2 , Circle3 );
39        end

```

```

33 [ Area] = GetRectanglesArea( Circle1 , Circle2 , Circle3 );
34
35 if( isempty( Area) )
36     [ PositionX , PositionY ] = ApplyTriangleMethod( Circle1
37         , Circle2 , Circle3 );
38 else
39     [ PositionX , PositionY ] = ApplyCenterOfArea( Area );
40 end
41 end
42
43 end

```

#### A.6.4. *Function CalculateLocalizationCentroid*

```

1 function [ PositionX , PositionY ] = CalculateLocalizationCentroid
2     ( Circle1 , Circle2 , Circle3 )
3
4     admissibleError = 0.5;
5
6     [ Xcross12 , Ycross12 ] = circcirc( Circle1(1) , Circle1(2) , Circle1
7         (3) , Circle2(1) , Circle2(2) , Circle2(3) );
8     [ Xcross13 , ~ ] = circcirc( Circle1(1) , Circle1(2) , Circle1(3) ,
9         Circle3(1) , Circle3(2) , Circle3(3) );
10    [ Xcross23 , ~ ] = circcirc( Circle2(1) , Circle2(2) , Circle2(3) ,
11        Circle3(1) , Circle3(2) , Circle3(3) );
12
13    x1 = Xcross12(1);
14    y1 = Ycross12(1);
15    x2 = Xcross12(2);
16    y2 = Ycross12(2);
17
18    dx1= abs( Circle3(1)-x1);
19    dy1 = abs( Circle3(2)-y1);
20    distance1 = sqrt( (dx1)^2+(dy1)^2 );
21
22    dx2= abs( Circle3(1)-x2);
23    dy2 = abs( Circle3(2)-y2);
24    distance2 = sqrt( (dx2)^2+(dy2)^2 );
25
26    if ( distance1-Circle3(3) ) < admissibleError
27        PositionX = x1;
28        PositionY = y1;
29    elseif ( distance2-Circle3(3) ) < admissibleError
30        PositionX = x2;
31        PositionY = y2;
32    else
33
34        if ( ( isnan( Xcross12(1) ) && isnan( Xcross12(2) ) ) || (
35            isnan( Xcross13(1) ) && isnan( Xcross13(2) ) ) || ( isnan(
36            Xcross23(1) ) && isnan( Xcross23(2) ) ) )

```

```

31 [ Circle1 , Circle2 , Circle3 ] = MakeCirclesCross (
32   Circle1 , Circle2 , Circle3 );
33 end
34 [ PositionX , PositionY ] = ApplyTriangleMethod( Circle1 ,
35   Circle2 , Circle3 );
36 end

```

### A.6.5. *Function* CalculateLocalization

```

1 function [PositionX, PositionY] = CalculateLocalization ( Circle1
, Circle2 , Circle3 )
2
3 admissibleDeviation = 0.3;
4 admissibleIncrement = 0.95;
5
6 [ Xcross12 , Ycross12 ] = circloc ( Circle1(1) , Circle1(2) , Circle1
(3) , Circle2(1) , Circle2(2) , Circle2(3) );
7 [ Xcross13 , ~ ] = circloc ( Circle1(1) , Circle1(2) , Circle1(3) ,
Circle3(1) , Circle3(2) , Circle3(3) );
8 [ Xcross23 , ~ ] = circloc ( Circle2(1) , Circle2(2) , Circle2(3) ,
Circle3(1) , Circle3(2) , Circle3(3) );
9
10 x1 = Xcross12(1);
11 y1 = Ycross12(1);
12 x2 = Xcross12(2);
13 y2 = Ycross12(2);
14
15 dx1= abs ( Circle3(1)-x1);
16 dy1 = abs ( Circle3(2)-y1);
17 distance1 = sqrt ((dx1)^2+(dy1)^2);
18
19 dx2= abs ( Circle3(1)-x2);
20 dy2 = abs ( Circle3(2)-y2);
21 distance2 = sqrt ((dx2)^2+(dy2)^2);
22
23 if abs ((distance1-Circle3(3))) < admissibleDeviation
24 PositionX = x1;
25 PositionY = y1;
26 elseif abs ((distance2-Circle3(3))) < admissibleDeviation
27 PositionX = x2;
28 PositionY = y2;
29 else
30
31 if ( (isnan (Xcross12(1)) && isnan (Xcross12(2))) || (isnan
(Xcross13(1)) && isnan (Xcross13(2))) || (isnan
(Xcross23(1)) && isnan (Xcross23(2))) )
32 [ Circle1b , Circle2b , Circle3b ] = MakeCirclesCross (
Circle1 , Circle2 , Circle3 );
33
```

```

34      if( ((Circle1b(3)-Circle1(3))>admissibleIncrement)
35          || ((Circle2b(3)-Circle2(3))>admissibleIncrement
36          || ((Circle3b(3)-Circle3(3))>admissibleIncrement)
37          ) )
38          [Area] = GetRectanglesArea(Circle1b,
39                                      Circle2b, Circle3b);
40          if( isempty(Area) )
41              [PositionX, PositionY] =
42                  ApplyTriangleMethod(Circle1b,
43                                      Circle2b, Circle3b);
44          else
45              [PositionX, PositionY] =
46                  ApplyCenterOfArea(Area);
47          end
48      else
49          [PositionX, PositionY] = ApplyTriangleMethod(
50                                      Circle1b, Circle2b, Circle3b);
51      end
52  end
53 end

```

### A.6.6. Function GetRectanglesArea

```

1 function [Area] = GetRectanglesArea (Circle1, Circle2, Circle3)
2
3 % First square
4 x1_1= Circle1(1)-Circle1(3);
5 x2_1= Circle1(1)+Circle1(3);
6 y1_1= Circle1(2)+Circle1(3);
7 y2_1= Circle1(2)-Circle1(3);
8 Xcoordinates1 = [x1_1, x1_1, x2_1, x2_1];
9 Ycoordinates1 = [y1_1, y2_1, y2_1, y1_1];
10 Rectangle1 = polyshape(Xcoordinates1, Ycoordinates1);
11
12 %Second Square
13 x1_2= Circle2(1)-Circle2(3);
14 x2_2= Circle2(1)+Circle2(3);
15 y1_2= Circle2(2)+Circle2(3);
16 y2_2= Circle2(2)-Circle2(3);
17 Xcoordinates2 = [x1_2, x1_2, x2_2, x2_2];
18 Ycoordinates2 = [y1_2, y2_2, y2_2, y1_2];
19 Rectangle2 = polyshape(Xcoordinates2, Ycoordinates2);
20
21 %Third Square
22 x1_3= Circle3(1)-Circle3(3);
23 x2_3= Circle3(1)+Circle3(3);
24 y1_3= Circle3(2)+Circle3(3);

```

```

25 y2_3= Circle3(2)-Circle3(3);
26 Xcoordinates3 = [x1_3 , x1_3 , x2_3 , x2_3];
27 Ycoordinates3 = [y1_3 , y2_3 , y2_3 , y1_3];
28 Rectangle3 = polyshape(Xcoordinates3 , Ycoordinates3);
29
30 Rectangles = [Rectangle1 , Rectangle2 , Rectangle3];
31 Area = intersect(Rectangles);
32 Area = Area.Vertices;
33
34 end

```

### A.6.7. *Function ApplyCenterOfArea*

```

1 function [PositionX , PositionY] = ApplyCenterOfArea (Area)
2
3 Xcoordinates = Area(1:4);
4 Ycoordinates = Area (5:8);
5
6 PositionX = min(Xcoordinates)+(max(Xcoordinates)-min(
    Xcoordinates))/2;
7 PositionY = min(Ycoordinates)+(max(Ycoordinates)-min(
    Ycoordinates))/2;
8
9 end

```

### A.6.8. *Function ApplyTriangleMethod*

```

1 function [PositionX , PositionY] = ApplyTriangleMethod (Circle1 ,
    Circle2 , Circle3)
2
3 [Xcross12 , Ycross12] = circcirc(Circle1(1) , Circle1(2) , Circle1(3) ,
    Circle2(1) , Circle2(2) , Circle2(3));
4 [Xcross13 , Ycross13] = circcirc(Circle1(1) , Circle1(2) , Circle1(3) ,
    Circle3(1) , Circle3(2) , Circle3(3));
5 [Xcross23 , Ycross23] = circcirc(Circle2(1) , Circle2(2) , Circle2(3) ,
    Circle3(1) , Circle3(2) , Circle3(3));
6
7 PointA1 = [Xcross12(1) , Ycross12(1)];
8 PointB1 = [Xcross12(2) , Ycross12(2)];
9 PointA2 = [Xcross13(1) , Ycross13(1)];
10 PointB2 = [Xcross13(2) , Ycross13(2)];
11 PointA3 = [Xcross23(1) , Ycross23(1)];
12 PointB3 = [Xcross23(2) , Ycross23(2)];
13
14 Triangle1 = polyshape([PointA1(1) PointA2(1) PointA3(1)] , [
    PointA1(2) PointA2(2) PointA3(2)]);
15 Triangle2 = polyshape([PointA1(1) PointA2(1) PointB3(1)] , [
    PointA1(2) PointA2(2) PointB3(2)]);
16 Triangle3 = polyshape([PointA1(1) PointB2(1) PointA3(1)] , [
    PointA1(2) PointB2(2) PointA3(2)]);
17 Triangle4 = polyshape([PointA1(1) PointB2(1) PointB3(1)] , [

```

```

    PointA1(2) PointB2(2) PointB3(2)]) ;
18 Triangle5 = polyshape([ PointB1(1) PointA2(1) PointA3(1)],[
    PointB1(2) PointA2(2) PointA3(2)]);
19 Triangle6 = polyshape([ PointB1(1) PointA2(1) PointB3(1)],[
    PointB1(2) PointA2(2) PointB3(2)]);
20 Triangle7 = polyshape([ PointB1(1) PointB2(1) PointA3(1)],[
    PointB1(2) PointB2(2) PointA3(2)]);
21 Triangle8 = polyshape([ PointB1(1) PointB2(1) PointB3(1)],[
    PointB1(2) PointB2(2) PointB3(2)]);

22
23 Areas= [ area(Triangle1), area(Triangle2), area(Triangle3), area(
    Triangle4), area(Triangle5), area(Triangle6), area(Triangle7),
    , area(Triangle8)];
24 [~,index] = min(Areas);

25
26     switch (index)
27         case 1
28             [ PositionX , PositionY ] = centroid(Triangle1);
29         case 2
30             [ PositionX , PositionY ] = centroid(Triangle2);
31         case 3
32             [ PositionX , PositionY ] = centroid(Triangle3);
33         case 4
34             [ PositionX , PositionY ] = centroid(Triangle4);
35         case 5
36             [ PositionX , PositionY ] = centroid(Triangle5);
37         case 6
38             [ PositionX , PositionY ] = centroid(Triangle6);
39         case 7
40             [ PositionX , PositionY ] = centroid(Triangle7);
41         case 8
42             [ PositionX , PositionY ] = centroid(Triangle8);
43     end
44
45 end

```

## A.7. Funciones para el cálculo de errores

### A.7.1. *Function CalculatePositionErrors*

```

1 function [ PositionErrors ] = CalculatePositionErrors ( PositionsX ,
    PositionsY , EstimatePositionsX , EstimatePositionsY )
2
3 PositionErrors = zeros(1,length( PositionsX ));
4
5 for i=1:length( PositionsX )
6     PositionError = sqrt((EstimatePositionsX(i)-PositionsX(i)
7         )^2+(EstimatePositionsY(i)-PositionsY(i))^2);
8     PositionErrors(i) = PositionError;
9 end

```

```
9  
10  
11 end
```

### A.7.2. *Function CalculateErrorsPerDensity*

```
1 function [ MaxPositionError , MeanPositionError , MinPositionError ]  
2     = CalculateErrorsPerDensity ( TrackPointsX , TrackPointsY ,  
3         EstimateTrackPointsX , EstimateTrackPointsY )  
4  
5 PositionErrors = CalculatePositionErrors( TrackPointsX ,  
6     TrackPointsY , EstimateTrackPointsX , EstimateTrackPointsY );  
7  
8 MaxPositionError = max( PositionErrors );  
9 MeanPositionError = mean( PositionErrors );  
10 MinPositionError = min( PositionErrors );  
11 end
```

### A.7.3. *Function CalculateRelativeErrors*

```
1 function [ RelativeErrors ] = CalculateRelativeErrors (   
2     AccurateValues , ValueErrors )  
3  
4     RelativeErrors = zeros(1,length( AccurateValues ));  
5  
6     for i=1:length( AccurateValues )  
7         RelativeError = ( ValueErrors( i )/AccurateValues( i ))  
8             *100;  
9         RelativeErrors( i ) = RelativeError ;  
10    end  
11 end
```

## **Apéndice B**

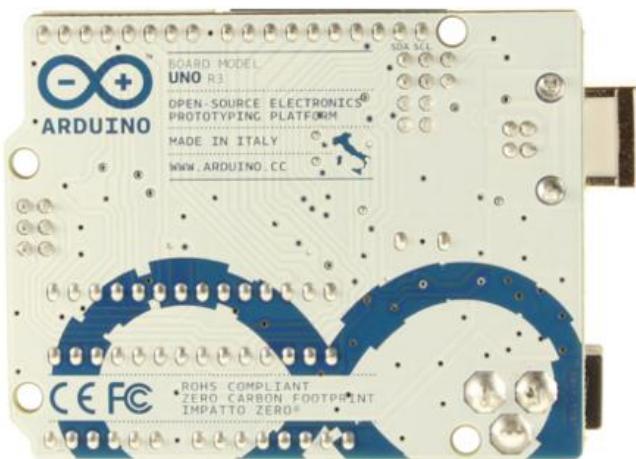
### **Fichas de datos de los dispositivos**

#### **B.1. Ficha de datos del Arduino**

# Arduino Uno



Arduino Uno R3 Front



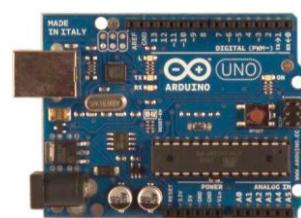
Arduino Uno R3 Back



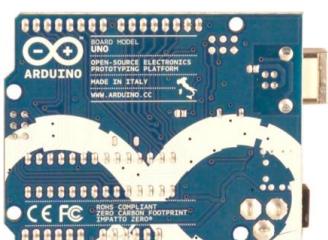
Arduino Uno R2 Front



Arduino Uno SMD



Arduino Uno Front



Arduino Uno Back

## Overview

The Arduino Uno is a microcontroller board based on the ATmega328 ([datasheet](#)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.

Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the [index of Arduino boards](#).

## Summary

|                             |           |
|-----------------------------|-----------|
| Microcontroller             | ATmega328 |
| Operating Voltage           | 5V        |
| Input Voltage (recommended) | 7-12V     |

|                         |  |
|-------------------------|--|
| Input Voltage (limits)  | 6-20V  |
| Digital I/O Pins        | 14 (of which 6 provide PWM output)                   |
| Analog Input Pins       | 6  |
| DC Current per I/O Pin  | 40 mA  |
| DC Current for 3.3V Pin | 50 mA  |
| Flash Memory            | 32 KB (ATmega328) of which 0.5 KB used by bootloader |
| SRAM                    | 2 KB (ATmega328)                                     |
| EEPROM                  | 1 KB (ATmega328)                                     |
| Clock Speed             | 16 MHz   |

## Schematic & Reference Design

EAGLE files: [arduino-uno-Rev3-reference-design.zip](#) (NOTE: works with Eagle 6.0 and newer)

Schematic: [arduino-uno-Rev3-schematic.pdf](#)

**Note:** The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

## Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

## Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the [EEPROM library](#)).

## Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode\(\)](#), [digitalWrite\(\)](#), and [digitalRead\(\)](#) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the [attachInterrupt\(\)](#) function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite\(\)](#) function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the [SPI library](#).
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though it is possible to change the upper end of their range using the AREF pin and the [analogReference\(\)](#) function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the [Wire library](#).

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with [analogReference\(\)](#).
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the [mapping between Arduino pins and ATmega328 ports](#). The mapping for the Atmega8, 168, and 328 is identical.

## Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, [on Windows, a .inf file is required](#). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A [SoftwareSerial library](#) allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a [Wire library](#) to simplify use of the I2C bus; see the [documentation](#) for details. For SPI communication, use the [SPI library](#).

## Programming

The Arduino Uno can be programmed with the Arduino software ([download](#)). Select "Arduino Uno" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the [reference](#) and [tutorials](#).

The ATmega328 on the Arduino Uno comes preburned with a [bootloader](#) that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol ([reference](#), [C header files](#)).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see [these instructions](#) for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use [Atmel's FLIP software](#) (Windows) or the [DFU programmer](#) (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See [this user-contributed tutorial](#) for more information.

## Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see [this forum thread](#) for details.

## USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

## Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

## B.2. Ficha de datos de XBee



EMBEDDED RF  
MODULES FOR OEMS



# DIGI XBEE® S2C 802.15.4 RF MODULES

Low-cost, easy-to-deploy modules provide critical end-point connectivity to devices and sensors

Digi XBee RF modules provide OEMs with a common footprint shared by multiple platforms, including multipoint and ZigBee/Mesh topologies, and both 2.4 GHz and 900 MHz solutions. OEMs deploying the Digi XBee can substitute one Digi XBee for another, depending upon dynamic application needs, with minimal development, reduced risk and shorter time-to-market.

Digi XBee 802.15.4 RF modules are ideal for applications requiring low latency and predictable communication timing. Providing quick, robust communication in point-to-point, peer-to-peer, and multipoint/star configurations, Digi XBee 802.15.4 products enable robust end-point connectivity with ease. Whether deployed as a pure cable replacement for simple serial

communication, or as part of a more complex hub-and-spoke network of sensors, Digi XBee 802.15.4 RF modules maximize performance and ease of development.

Digi XBee 802.15.4 modules seamlessly interface with compatible gateways, device adapters and range extenders, providing developers with true beyond-the-horizon connectivity.

The updated Digi XBee S2C 802.15.4 module is built with the SiliconLabs EM357 SoC and offers improved power consumption, support for over-the-air firmware updates, and provides an upgrade path to DigiMesh® or ZigBee® mesh protocols if desired.

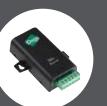
## BENEFITS

- Simple, out-of-the-box RF communications, no configuration needed
- Point-to-multipoint network topology
- 2.4 GHz for worldwide deployment
- Common Digi XBee footprint for a variety of RF modules
- Industry leading sleep current of sub 1uA
- Firmware upgrades via UART, SPI or over the air
- Migratable to DigiMesh and ZigBee PRO protocols and vice-versa

## RELATED PRODUCTS



ConnectPort®  
X4/X4H Gateways



XBee®  
Adapters



XCTU

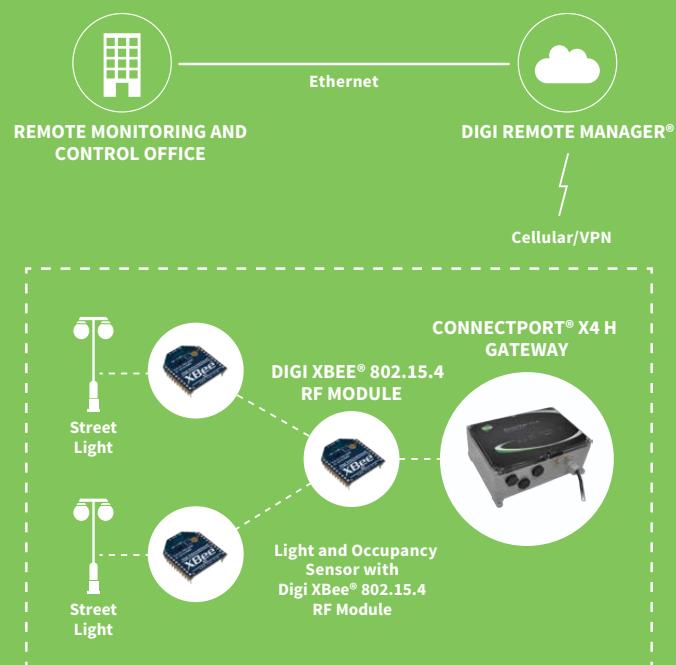


Digi Remote  
Manager®



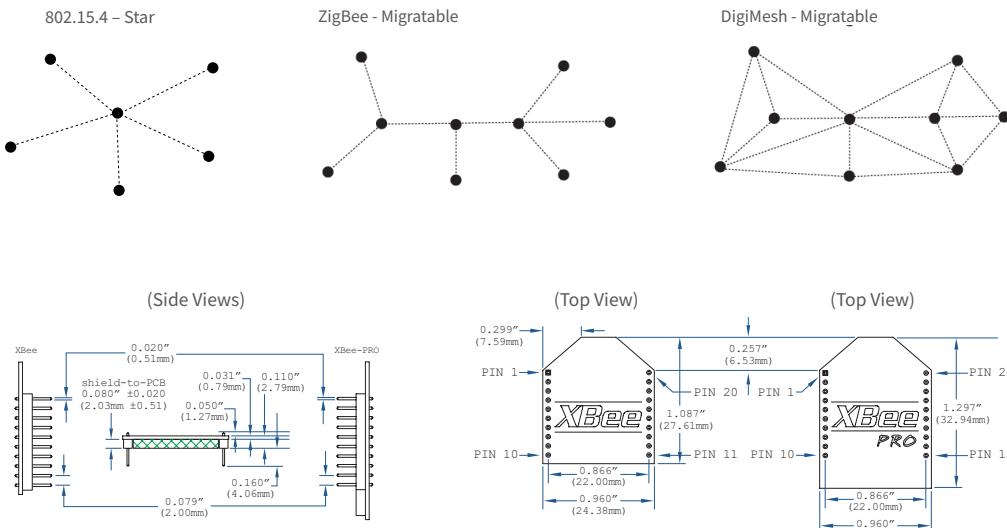
Development  
Kits

## APPLICATION EXAMPLE



| SPECIFICATIONS                    |  | Digi XBee® S2C 802.15.4  | Digi XBee-PRO® S2C 802.15.4 |
|-----------------------------------|--|--|-----------------------------|
| <b>PERFORMANCE</b>                |  |  |                             |
| TRANSCEIVER CHIPSET               | Silicon Labs EM357 SoC   |  |                             |
| DATA RATE                         | RF 250 Kbps, Serial up to 1 Mbps   |  |                             |
| INDOOR/URBAN RANGE                | 200 ft (60 m)  | 300 ft (90 m)  |                             |
| OUTDOOR/RF LINE-OF-SIGHT RANGE    | 4000 ft (1200 m)   | 2 miles (3200 m)   |                             |
| TRANSMIT POWER                    | 3.1 mW (+5 dBm) / 6.3 mW (+8 dBm)<br>boost mode  | 63 mW (+18 dBm)  |                             |
| RECEIVER SENSITIVITY (1% PER)     | -100 dBm / -102 dBm boost mode   | -101 dBm   |                             |
| <b>FEATURES</b>                   |  |  |                             |
| SERIAL DATA INTERFACE             | UART, SPI  |  |                             |
| CONFIGURATION METHOD              | API or AT commands, local or over-the-air (OTA)  |  |                             |
| FREQUENCY BAND                    | ISM 2.4 GHz  |  |                             |
| FORM FACTOR                       | Through-Hole, Surface Mount  |  |                             |
| HARDWARE                          | S2C  |  |                             |
| ADC INPUTS                        | (4) 10-bit ADC inputs  |  |                             |
| DIGITAL I/O                       | 15   |  |                             |
| ANTENNA OPTIONS                   | Through-Hole: PCB Antenna, U.FL Connector, RP-SMA Connector, or Integrated Wire<br>SMT: RF Pad, PCB Antenna, or U.FL Connector |  |                             |
| OPERATING TEMPERATURE             | -40° C to +85° C   |  |                             |
| DIMENSIONS (L X W X H) AND WEIGHT | Through-Hole: 0.960 x 1.087 in (2.438 x 2.761 cm)<br>SMT: 0.866 x 1.33 x 0.120 in (2.199 x 3.4 x 0.305 cm)                     | Through-Hole: 0.960 x 1.297 in (2.438 x 3.294 cm)<br>SMT: 0.866 x 1.33 x 0.120 in (2.199 x 3.4 x 0.305 cm) |                             |
| <b>NETWORKING AND SECURITY</b>    |  |  |                             |
| PROTOCOL                          | Digi XBee 802.15.4 (Proprietary 802.15.4)  |  |                             |
| UPDATABLE TO DIGIMESH PROTOCOL    | Yes  |  |                             |
| UPDATABLE TO ZIGBEE PROTOCOL      | Yes  |  |                             |
| INTERFERENCE IMMUNITY             | DSSS (Direct Sequence Spread Spectrum)   |  |                             |
| ENCRYPTION                        | 128-bit AES  |  |                             |
| RELIABLE PACKET DELIVERY          | Retries/Acknowledgements   |  |                             |
| IDS                               | PAN ID and addresses, cluster IDs and endpoints (optional)   |  |                             |
| CHANNELS                          | 16 channels  | 15 channels  |                             |
| <b>POWER REQUIREMENTS</b>         |  |  |                             |
| SUPPLY VOLTAGE                    | 2.1 to 3.6V  | 2.7 to 3.6V  |                             |
| TRANSMIT CURRENT                  | 33 mA @ 3.3 VDC / 45 mA boost mode   | 120 mA @ 3.3 VDC   |                             |
| RECEIVE CURRENT                   | 28 mA @ 3.3 VDC / 31 mA boost mode   | 31 mA @ 3.3 VDC  |                             |
| POWER-DOWN CURRENT                | <1 µA @ 25° C  | <1 µA @ 25° C  |                             |
| <b>REGULATORY APPROVALS</b>       |  |  |                             |
| FCC, IC (NORTH AMERICA)           | Yes  | Yes  |                             |
| ETSI (EUROPE)                     | Yes  | No   |                             |
| RCM (AUSTRALIA AND NEW ZEALAND)   | No (Coming soon)   | No (Coming soon)   |                             |
| TELEC (JAPAN)                     | No (Coming soon)   | No (Coming soon)   |                             |

| PART NUMBERS    | DESCRIPTION   |
|-----------------|---|
| KIT             |   |
| XKB2-A2T-WWC    | Wireless Connectivity Kit with Digi XBee 802.15.4 (S2C)       |
| MODULES         |   |
| XB24CAWIT-001   | Digi XBee 802.15.4 through-hole module w/ wire antenna        |
| XB24CAPIT-001   | Digi XBee 802.15.4 through-hole module w/ PCB antenna         |
| XB24CAUIT-001   | Digi XBee 802.15.4 through-hole module w/ U.fl connector      |
| XB24CASIT-001   | Digi XBee 802.15.4 through-hole module w/ RPSMA connector     |
| XB24CAPIIS-001  | Digi XBee 802.15.4 SMT module w/ PCB antenna                  |
| XB24CAUIS-001   | Digi XBee 802.15.4 SMT module w/ U.fl connector               |
| XB24CARIS-001   | Digi XBee 802.15.4 SMT module w/ RF Pad connector             |
| XBP24CAWIT-001  | Digi XBee-PRO 802.15.4 through-hole module w/ wire antenna    |
| XBP24CAPIT-001  | Digi XBee-PRO 802.15.4 through-hole module w/ PCB antenna     |
| XBP24CAUIT-001  | Digi XBee-PRO 802.15.4 through-hole module w/ U.fl connector  |
| XBP24CASIT-001  | Digi XBee-PRO 802.15.4 through-hole module w/ RPSMA connector |
| XBP24CAPIIS-001 | Digi XBee-PRO 802.15.4 SMT module w/ PCB antenna              |
| XBP24CAUIS-001  | Digi XBee-PRO 802.15.4 SMT module w/ U.fl connector           |
| XBP24CARIS-001  | Digi XBee-PRO 802.15.4 SMT module w/ RF Pad connector         |



**DIGI SERVICE AND SUPPORT** / You can purchase with confidence knowing that Digi is always available to serve you with expert technical support and our industry leading warranty. For detailed information visit [www.digi.com/support](http://www.digi.com/support).

© 1996-2017 Digi International Inc. All rights reserved.  
All trademarks are the property of their respective owners.

**DIGI INTERNATIONAL WORLDWIDE HQ**  
877-912-3444 / 952-912-3444 / [www.digi.com](http://www.digi.com)

**DIGI INTERNATIONAL GERMANY**  
+49-89-540-428-0

**DIGI INTERNATIONAL SINGAPORE**  
+65-6213-5380

**DIGI INTERNATIONAL CHINA**  
+86-21-50492199 / [www.digi.com.cn](http://www.digi.com.cn)

91003287  
A4/717  
**DIGI INTERNATIONAL JAPAN**  
+81-3-5428-0261 / [www.digi-intl.co.jp](http://www.digi-intl.co.jp)

