Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

ESTIMACIÓN DE POSICIÓN BASADA EN DEEP LEARNING CON SENSORES AMBIENTALES UWB Y DISPOSITIVOS MÓVILES

Alumno/a: **Navas Damas, Manuel**

Tutor/a: Dr. D. Javier Medina Quero
Dª. Aurora Polo Rodríguez
Dpto.: Informática

Febrero, 2023

Universidad de Jaén

Departamento de Informática

Dr. D. **Javier Medina Quero** tutor del Proyecto Fin de Máster titulado: **Estimación de posición basada en Deep Learning con sensores ambientales UWB y dispositivos móviles**, que presenta **Manuel Navas Damas**, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén

Jaén, febrero de 2023

El alumno:

NAVAS DAMAS MANUEL - 77384996R

Digitally signed by NAVAS DAMAS MANUEL - 77384996R
Date: 2023.02.14 00:08:00 +01'00'

Manuel Navas Damas

Los tutores:

MEDINA QUERO JAVIER - 75131484Y

Firmado digitalmente por MEDINA QUERO JAVIER - 75131484Y
Fecha: 2023.02.14 09:08:26 +01'00'

Javier Medina Quero

POLO RODRIGUEZ AURORA - 77647378E

Firmado digitalmente por POLO RODRIGUEZ AURORA - 77647378E
Fecha: 2023.02.14 11:11:22 +01'00'

Aurora Polo Rodríguez

A mi familia, por confiar siempre en mí, y en especial a mi madre María Teresa por su amor incondicional.

A mis tutores, porque gracias a su dedicación, sus consejos y su cercanía este camino ha sido mucho más llevadero, y en especial a Javier por el apoyo que me ha prestado desde el primer día.

A todos aquellos con quien he compartido mi experiencia en este Máster, tanto profesores como compañeros.

Gracias.

Índice

1 Introducción 11
1.1 Motivación 11
1.2 Objetivos 17
1.3 Propuesta 17
1.4 Estructura de la memoria 19
2 Estado del arte 20
2.1 Conceptos previos 20
2.2 Trabajos relacionados 33
3 Métodos 42
3.1 Arquitectura general 42
3.2 Dispositivos baliza UWB 43
3.3 Aplicación móvil y framework UWB 47
3.4 Servidor REST 65
3.5 Modelo de Deep Learning 75
4 Validación 92
4.1 Entornos de prueba 92
4.2 Resultados 95
4.3 Discusión 99
5 Conclusiones y trabajos futuros 101
6 Referencias 104

Índice de tablas

Tabla 1. *Comparativa de especificaciones entre las tecnologías UWB, WiFi y BLE* 23
Tabla 2. *Lista de smartphones con compatibilidad con la tecnología UWB (Fuente: Wikiwand.com)* 24
Tabla 3. *Especificaciones del dispositivo de desarrollo en iOS* 51
Tabla 4. *Resumen de errores absolutos en la estimación por entorno, coordenadas y promedio* 100

Índice de figuras

Figura 1. *Evolución de la población mundial mayor de 65 años ("World Population Prospects 2022," 2022)*..... 12
Figura 2. *Evolución de la esperanza de vida en diferentes regiones y a nivel mundial* ("World Population Prospects 2022," 2022) 12
Figura 3. *Porcentaje de personas de 65 años o más que viven solas, por país o zona de residencia.* ("World Population Ageing 2020 Highlights," 2020)..... 13
Figura 4. *Mapa de RSS empleado con un algoritmo fingerprinting para localización en interiores* (Díaz Gonzalez, 2017) 15
Figura 5. *Ejemplo de despliegue del sistema de localización en interiores basado en Deep Learning con balizas UWB y dispositivo móvil* 18
Figura 6. *Método Angle-of-Arrival (AOA) (Alarifi et al., 2016)* 25
Figura 7. *Método Time-of-Arrival (TOA) (Alarifi et al., 2016)*..... 25
Figura 8. *Método Time Difference-of-Arrival (TDOA) (Alarifi et al., 2016)* 26
Figura 9. *Localización basada en RSSI (Zafari et al., 2019)*..... 27
Figura 10. *Estimación de la posición mediante trilateración a) con información perfecta e invariable (teórica) b) con perturbaciones (real) (Cook et al., 2005)* 28
Figura 11. *Relación entre inteligencia artificial, aprendizaje automático y aprendizaje profundo (Kelleher, 2019)*..... 30
Figura 12. *Esquema de una red neuronal (Kelleher, 2019)*..... 31
Figura 13. *Arquitectura general de la propuesta de localización en interiores basada en Deep Learning con sensores ambientales UWB*..... 42
Figura 14. *Kit de desarrollo de balizas UWB Estimote (izquierda) y contenido del kit (derecha).* 44
Figura 15. *Comparación del tamaño de una baliza UWB de Estimote con una moneda de 1€.*..... 46
Figura 16. *Esquema interno de una baliza UWB de Estimote.* 46
Figura 17. *Airtag es uno de los primeros dispositivos comerciales que utilizan la tecnología UWB para localizar objetos interiores (Kateliev, 2021)*..... 49
Figura 18. *Interfaz gráfica del IDE Xcode empleado para el desarrollo de la aplicación iOS.* 51
Figura 19. *GUI de la aplicación iOS (izquierda) y esquema del domicilio sensorizado (derecha).* 52

Figura 20. *Interfaz de usuario de la aplicación iOS durante una sesión activa* 54
Figura 21. *Se proporciona el delegado en la inicialización del EstimoteUWBManager* 57
Figura 22. *Protocolo de búsqueda y conexión con las balizas UWB del SDK.* 58
Figura 23. *Mensajes de descubrimiento y conexión con las balizas UWB por parte del dispositivo móvil.* 58
Figura 24. *Función didUpdatePosition() que permite obtener los datos UWB desde las balizas desplegadas en el entorno.* 59
Figura 25. *Modificador onReceive() y método que gestiona el envío de datos al servidor.* 61
Figura 26. *Proceso de añadir dependencias mediante el gestor de paquetes de Swift* 63
Figura 27. *Posición actual de Python como primer lenguaje más utilizado (arriba) y evolución de su uso en las última décadas (abajo)* (*TIOBE Index*, n.d.) 67
Figura 28. *Interfaz de Anaconda Navigator.* 68
Figura 29. *Endpoint /update de la API REST* 70
Figura 30. *Endpoint /open_session de la API REST* 71
Figura 31. *Envío de la petición GET al endpoint /open_session desde la app iOS* 71
Figura 32. *Endpoint /close_session de la API REST* 72
Figura 33. *Envio de la petición GET al endpoint /close_session desde la app iOS* 72
Figura 34. *Endpoint /user_pos_update de la API REST* 73
Figura 35. *Ejecución de la API REST* 73
Figura 36. *Archivo de muestras UWB* 74
Figura 37. *Fichero de etiquetas de posición con respecto a un instante de tiempo* 75
Figura 38. *Configuración de capas del modelo Deep Learning propuesto.* 83
Figura 39. *Esquema del entorno A* 93
Figura 40. *Esquema del entorno B* 95

Figura 41. *Error absoluto por coordenadas (arriba) y fichero xy.tsv (abajo) donde se almacenan las coordenadas reales junto a las estimadas por el modelo para el conjunto de datos del entorno A.*..... 96
Figura 42. *Ejemplos de representación gráfica de las coordenadas reales (azules) y estimadas por el modelo (rojas) para el entorno A.* 97
Figura 43. *Error absoluto por coordenadas (arriba) para los datos del entorno B y fichero xy.tsv (abajo) donde se almacenan las coordenadas reales junto a las estimadas por el modelo.* 98
Figura 44. *Ejemplos de representación gráfica de las coordenadas reales (azules) y estimadas por el modelo (rojas) para el entorno B.* 99

1 Introducción

En este capítulo introductorio se presenta el trabajo Fin de Máster "*Estimación de posición basada en Deep Learning con sensores ambientales UWB y dispositivos móviles*", donde se presenta un diseño de sistema de localización en interiores basado en el uso de sensores con tecnología de banda ultraancha (Ultra-Wide Band, UWB) en combinación con un modelo de Deep Learning que permite estimar la posición con una precisión suficientemente buena para entornos domésticos y similares. La novedad de esta propuesta es que el dispositivo móvil es quien recoge la señal de localización de UWB haciendo uso de balizas ambientales de bajo consumo.

1.1 Motivación

El crecimiento y envejecimiento de la población mundial es una evidencia incuestionable hoy en día, así como el hecho de que la denominada tercera edad representa un segmento importante de la población que paulatinamente va a continuar en aumento. Desde los años 1950 se calcula un incremento del 5% de las personas mayores de 65 años sobre el porcentaje de la población mundial (ver Figura 1), situándose hoy en día cerca del 10 % ("World Population Prospects 2022," 2022). Esta tendencia se prevé que continúe aumentando durante los próximos años, puesto que el envejecimiento de la población es mayor en la actualidad que en años anteriores y la esperanza de vida se ha visto incrementada constantemente (ver Figura 2) gracias a la mejora de los servicios sanitarios y a los avances de la ciencia y la tecnología aplicados al bienestar de la sociedad.

Estos datos son aún más notables si nos limitamos a los países desarrollados. Sin ir más lejos, en España se pronostica que para el año 2050 el 44,1% de las personas tendrán más de 60 años (Otero et al., 2004) y de igual forma ocurrirá, por ejemplo, en Estados Unidos con un 32% esperado (Puyol, 2012).

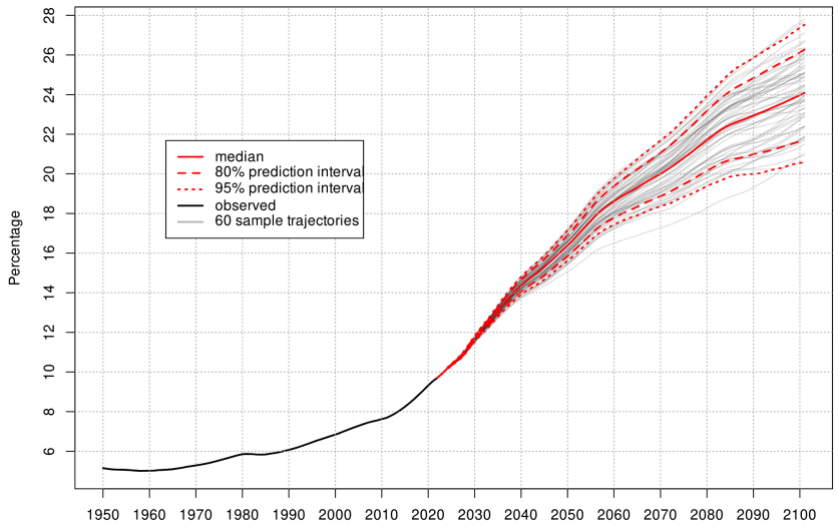


Figura 1. *Evolución de la población mundial mayor de 65 años ("World Population Prospects 2022," 2022)*

	Life expectancy at birth (years)								
	1990			*2021*			*2050*		
Region	*Males*	*Females*	*Both sexes*	*Males*	*Females*	*Both sexes*	*Males*	*Females*	*Both sexes*
World	**61.5**	**66.5**	**64.0**	**68.4**	**73.8**	**71.0**	**74.8**	**79.8**	**77.2**
Sub-Saharan Africa	47.3	51.2	49.2	57.8	61.6	59.7	64.3	69.1	66.7
Northern Africa and Western Asia	61.7	67.0	64.3	69.7	74.8	72.1	76.0	80.8	78.3
Central and Southern Asia	58.1	59.9	58.9	65.9	69.6	67.7	74.9	79.4	77.1
Eastern and South-Eastern Asia	65.6	70.7	68.1	73.6	79.6	76.5	79.4	84.1	81.7
Latin America and the Caribbean	64.6	70.9	67.7	68.8	75.8	72.2	78.1	83.1	80.6
Australia/New Zealand	73.7	79.8	76.8	82.7	85.6	84.2	85.4	88.6	87.0
Oceania*	60.3	65.5	62.5	64.6	70.1	67.1	68.4	74.9	71.6
Europe and Northern America	69.7	77.4	73.6	73.9	80.4	77.2	81.6	86.1	83.8
Least developed countries	48.7	51.6	50.1	61.7	66.5	64.1	67.8	73.5	70.6
Landlocked developing Countries	49.0	53.5	51.2	61.0	66.5	63.7	67.4	73.4	70.3
Small island developing States	63.4	67.9	65.6	68.0	73.9	70.8	74.1	80.0	77.0

*excluding Australia and New Zealand

Figura 2. *Evolución de la esperanza de vida en diferentes regiones y a nivel mundial* ("World Population Prospects 2022," 2022)

Uno de los aspectos más importantes que se ha identificado en diversos estudios como necesidad de este segmento de la población debido a sus características es la localización en interiores. Este tipo de personas son susceptibles, debido a su edad, de sufrir diferentes patologías que hacen necesaria una atención permanente y sobre todo el seguimiento y control de su localización en cada momento. Algunas de las enfermedades más comunes en este segmento de la población son el Alzheimer o la demencia, entre otras patologías menos comunes. En todas ellas la tecnología puede ayudar para prevenir accidentes monitorizando su actividad diaria y permitiendo a los

cuidadores o personas a cargo poder responder con la mayor brevedad posible a ciertas situaciones peligrosas para la salud de estas personas mayores.

Otro factor importante de nuestra sociedad que se debe tener en cuenta es el aumento del número de personas mayores que viven solas (ver Figura 3), bien porque prefieren quedarse en casa el mayor tiempo posible en lugar de ir a una residencia (Donnelly et al., 2016) o bien porque un conjunto de esta población no tiene los recursos necesarios para asumir el coste que supone una residencia o de tener un cuidador particular en su propio domicilio.

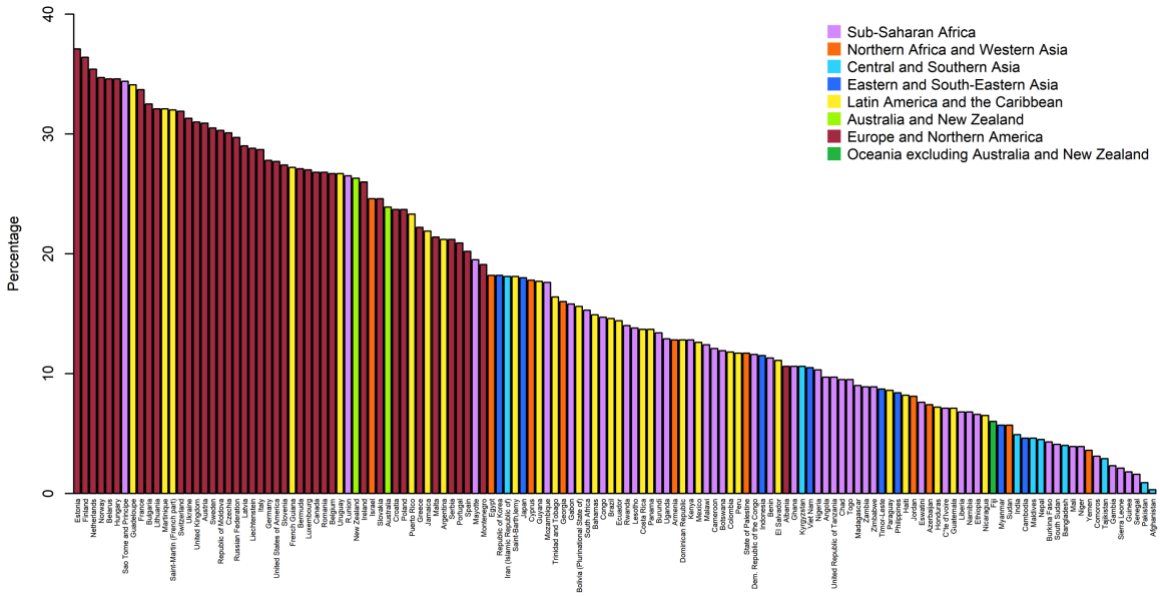


Figura 3. ***Porcentaje de personas de 65 años o más que viven solas, por país o zona de residencia.*** **("World Population Ageing 2020 Highlights," 2020)**

A raíz de todos los datos mostrados se hace evidente la necesidad de investigar nuevas y diferentes formas de cuidary otorgar autonomía a este segmento de la población, siendo la localización de mayores dentro de su lugar de residencia habitual de la forma más rápida y oportuna posible de cara a evitar cualquier riesgo o accidente. El principal problema que existe en la actualidad sobre los sistemas de localización en interiores, es que no se han encontrado tecnologías o métodos eficaces que ofrezcan una precisión en la localización lo suficientemente óptima como para permitir determinar de forma precisa la ubicación precisa de una persona dentro

de una casa o un edificio a bajo coste, larga autonomía y fácil despliegue. Es por esto que el desarrollo de nuevas propuestas y enfoques de localización en interiores es un tema de investigación abierto y de gran relevancia, que ofrece extensas posibilidades para el desarrollo de nuevas soluciones que intenten minimizar al máximo de lo posible el error de estimación en las localizaciones, ya sea mediante soluciones tecnológicas con nuevos desarrollos electrónicos o mediante la definición de métodos y técnicas científicas que mejoren la tecnología existente.

Se han realizado muchos estudios sobre este tema, la mayoría basados en el uso de tecnologías muy comunes en paradigmas como el Internet de las cosas (IoT) o los ambientes inteligentes, como son el Bluetooth Low Energy (BLE) o la red inalámbrica WiFi, así como dispositivos como smartphones, wearables y otras soluciones comerciales. Sin embargo, otras propuestas científicas han estudiado y puesto en práctica el potencial de una tecnología empleada en el pasado para aplicaciones militares, y que en muchos casos (como se pondrá de manifiesto en el apartado 2 de este trabajo) ha ofrecido resultados muy prometedores para su uso en nuevas soluciones de sistemas de posicionamiento en interiores. Esta tecnología se conoce como banda ultraancha (Ultra Wide-Band o UWB) y permite la conectividad entre dispositivos en distancias relativamente cortas de forma similar a como lo hacen el WiFi o Bluetooth, pero obteniendo un grado de precisión más elevado en comparación con ellas.

En este Trabajo de Fin de Máster se estudiarán las capacidades de la tecnología UWB integrada en un sistema de predicción inteligente para localización en interiores usando dispositivos móviles como receptores y recolectores de los datos. Hoy en día, los sistemas inteligentes que ofrecen capacidades de inteligencia artificial (IA) han tenido una mejora relevante de prestaciones gracias al aprendizaje automático (también conocido como Machine Learning). Machine learning describe la capacidad de aprender de los sistemas, dispositivos y computadores, resolver problemas y automatizar procesos de construcción de modelos analíticos para resolver las tareas asociadas. Cuando los modelos se generan a partir de datos de entrenamiento específicos de un problema hablamos de enfoques orientados a datos. Recientemente, dentro del aprendizaje automático encontramos la variante denominada Aprendizaje Profundo o Deep Learning, que se corresponde con un

concepto basado en redes neuronales artificiales y extracción automática de características. Para muchas aplicaciones, los modelos de Deep Learning superan a los modelos de aprendizaje automático superficial y a los enfoques tradicionales de análisis de datos (Janiesch et al., 2021). Con relación a los algoritmos de posicionamiento que se han estudiado con modelos de Deep Learning distinguimos entre la Trilateración y el Fingerprinting. El primero se refiere a la estimación de localización en interiores mediante procedimientos geométricos mientras que el segundo se basa en el uso de la intensidad de la señal recibida (Received Signal Strength, RSS) con algoritmos de proximidad, como el denominado k-Nearest Neighbor (kNN) (Campaña Bastidas et al., 2022).

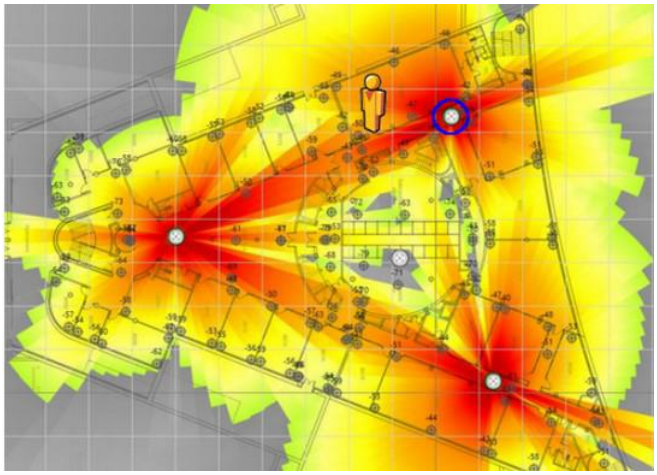


Figura 4. ***Mapa de RSS empleado con un algoritmo fingerprinting para localización en interiores*** **(Díaz Gonzalez, 2017)**

En este trabajo proponemos la implementación de un método de Fingerprinting con señales inalámbricas, que se basará en medir la intensidad de las señales procedentes de varios puntos de acceso o balizas y empleará esta información para determinar la ubicación del dispositivo que portará el usuario. Las señales que trabajan con RSS pueden ser WiFi, Bluetooth, BLE, UWB, etc. El método consta de dos fases: entrenamiento y estimación de la posición. En la fase de entrenamiento se utiliza un mapa con las intensidades de señal recibidas en diferentes puntos y para definir el posicionamiento se compara la RSSI observada en el sensor de emisión con el mapa previamente construido, utilizando algoritmos de proximidad (en nuestro caso de Deep Learning), y a continuación se estima la posición del sensor (ver Figura 4).

Durante la fase de entrenamiento que hemos mencionado se necesita emplear algún tipo de dispositivo que nos permita realizar el etiquetado de la posición del usuario en el entorno de prueba al mismo tiempo que se obtienen los datos de la intensidad de la señal de las balizas desplegadas. Habitualmente, para pruebas de laboratorio o pruebas de concepto en entornos simulados o reales, se suelen utilizar dispositivos móviles o vestibles (wearables) como los relojes o pulseras inteligentes. En este trabajo fin de máster se ha optado por desarrollar una aplicación para dispositivos móviles, en concreto para dispositivos iPhone con sistema operativo iOS.

Las razones para utilizar este tipo de dispositivos son varias. En primer lugar, destaca la gran popularidad de la que gozan los smartphones en nuestra sociedad hoy en día, por lo que el conocimiento sobre el uso de estos dispositivos en la mayoría de usuarios suele ser medio o alto, sobre todo en la población más joven. Sin embargo, es un hecho que cada vez es más alto el porcentaje de personas adultas, e incluso el de personas mayores que utilizan diariamente este tipo de dispositivos, por lo cual se está rompiendo la brecha digital que existía años atrás con este tipo de dispositivos (Busch et al., 2021). Por este motivo, resulta interesante comenzar a enfocar los estudios que tienen como objetivo a este segmento de la población teniendo en cuenta su tendencia a portar y saber utilizar en un futuro cercano este tipo de dispositivos con prácticamente las mismas facilidades que un adulto de mediana edad.

Por otro lado, la mejora en cuanto a las nuevas formas de desarrollo de software destinado a dispositivos móviles han facilitado la curva de aprendizaje para crear aplicaciones gracias a entornos de desarrollo integrados más eficaces e intuitivos (como las últimas versiones de Xcode[1] o Android Studio); o la integración de lenguajes de programación como Swift 5[2] o Kotlin 1.8 orientados al desarrollo de software para iPhone y Android respectivamente. Además, también existe una gran variedad y calidad de frameworks disponibles abiertamente para la comunidad de desarrolladores de dispositivos móviles, los cuales facilitan en gran medida muchas tareas y añaden funcionalidades muy comunes en estos desarrollos. En conjunto,

[1] https://developer.apple.com/xcode/
[2] https://www.apple.com/es/swift/

estos avances nos permiten reducir en gran medida el tiempo de desarrollo de aplicaciones que aprovechen las grandes capacidades de este tipo de dispositivos.

La integración de la intensidad de señal proporcionada por las balizas UWB junto con un modelo de Deep Learning basado en Fingerprinting proporciona al sistema una mayor fiabilidad y precisión, minimizando el error en las estimaciones hasta niveles aceptables para su aplicación en entornos reales.

1.2 Objetivos

Los objetivos de este Trabajo de Fin de Máster son los siguientes:

- Despliegue de sensores UWB ambientales y creación de una herramienta de recolección de su señal desde un dispositivo móvil con receptor UWB.
- Creación de una herramienta para el etiquetado de posición en tiempo real desde dispositivo móvil para la creación de la base de datos Fingerprinting.
- Desarrollo de casos de estudio de recolección y etiquetado de datos UWB en entornos reales.
- Creación de un modelo de Deep Learning de estimación de localización en base a la medida de UWB recibida en el móvil por cada baliza.
- Integración del modelo de Deep Learning en nodo Fog para evaluación en tiempo real.

1.3 Propuesta

Basándonos en lo anterior, el propósito de este Trabajo de Fin de Máster es la **creación de un sistema de localización de usuarios en espacios interiores mediante la integración de balizas ambientales con tecnología UWB**. Para ello, los usuarios portarán un **dispositivo móvil** que recolectará la señal de UWB de las

balizas en tiempo real, enviando estos datos a un **nodo Fog** implementado en Python, que recibirá y procesará dicha información. La comunicación entre el dispositivo móvil y el nodo se realizará utilizando el protocolo HTTP mediante la implementación de una **API REST**.

Usando la técnica de Fingerprinting, un **modelo de Deep Learning** estimará la posición del usuario en base a la señal y un etiquetado que realizará el usuario previamente durante la fase de entrenamiento mediante un dispositivo móvil. Este modelo entrenado se desplegará en el nodo Fog para proveer la localización del usuario en tiempo real dentro de un entorno inteligente, que permite preservar la privacidad de los datos en contexto local.

Este sistema de localización en interiores ofrecerá la **suficiente precisión y robustez como para poder localizar al usuario en todo momento dentro de su domicilio**. Sus posibles aplicaciones están orientadas principalmente al seguimiento y control de personas mayores o dependientes para prevenir accidentes y monitorizar su actividad diaria mediante un **sistema de bajo coste, larga autonomía, fácil despliegue y alta escalabilidad**.

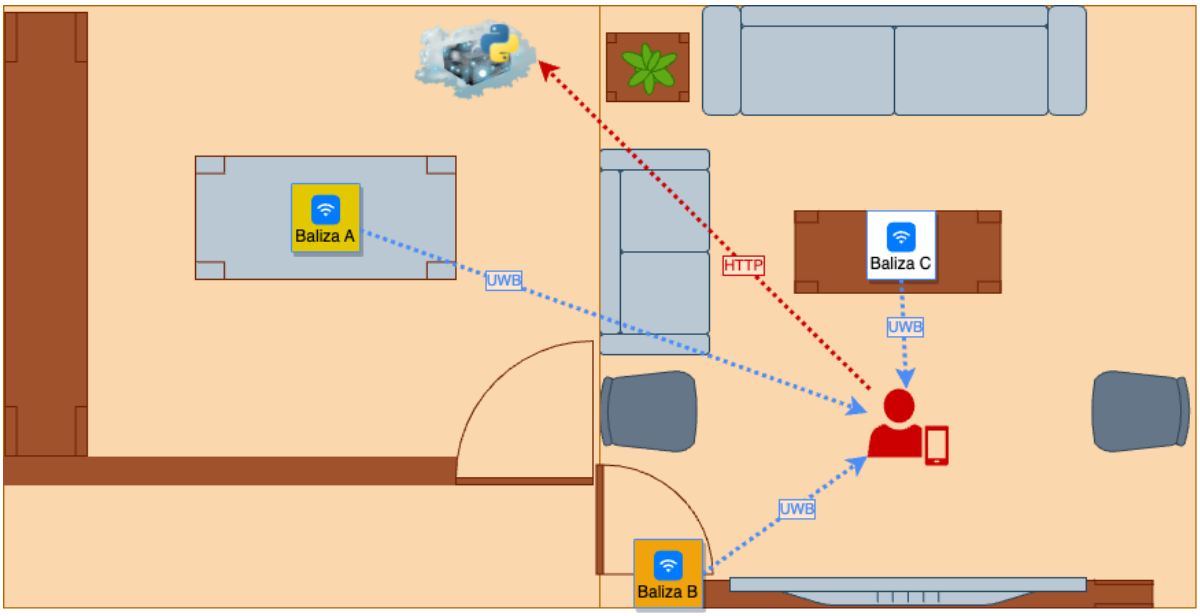


Figura 5. *Ejemplo de despliegue del sistema de localización en interiores basado en Deep Learning con balizas UWB y dispositivo móvil*

1.4 Estructura de la memoria

La estructura de esta memoria se ha dividido en los siguientes capítulos:

I. En el **capítulo 1** se ha explicado la propuesta que se plantea en este Trabajo de Fin de Máster, se ha motivado su utilidad de aplicación y los objetivos que se pretenden alcanzar con el desarrollo de este trabajo.
II. En el **capítulo 2** se describen los conceptos básicos necesarios para la comprensión del resto de la memoria y se lleva a cabo una revisión literaria sobre el contexto del trabajo.
III. En el **capítulo 3** se detallan aspectos específicos de la propuesta, tales como los sensores que se han utilizado, la creación del servidor REST y sus servicios asociados, la aplicación para dispositivo móvil y el framework UWB asociado a los sensores seleccionados y por último el modelo Deep Learning desarrollado.
IV. En el **capítulo 4** se expone la validación de la propuesta, mostrándose los resultados obtenidos y discutiendo la viabilidad de la misma para su aplicación en entornos reales.
V. En el **capítulo 5** se presentan las conclusiones y se plantean posibles mejoras y trabajos futuros.
VI. Para finalizar, en el **capítulo 6** se recogen las referencias de la presente memoria.

2 Estado del arte

En este capítulo, en primer lugar, revisaremos algunos conceptos y técnicas implementadas en la literatura científico-técnica que se utilizan a lo largo de este Trabajo Fin de Máster, puesto que su comprensión es fundamental para entender los métodos empleados y los análisis posteriores. En segundo lugar, se realizará una revisión literaria sobre trabajos previos relacionados con los sistemas de localización en interiores que han sido tenidos en cuenta como referencia para la solución propuesta en este Trabajo de Fin de Máster. Se diferenciará entre:

- Estudios sobre posicionamiento en interiores en general, donde se analizan trabajos basados en distintos protocolos, tecnologías y métodos, sin centrarnos en ninguno en particular.
- Estudios sobre posicionamiento en interiores centrados en implementaciones mediante tecnología UWB junto con distintos métodos de aproximación.

2.1 Conceptos previos

En esta sección se definen algunos conceptos importantes sobre los sistemas de posicionamiento en interiores y la tecnología de banda ultraancha (UWB).

2.1.1 Posicionamiento en interiores (Indoor Positioning, IP)

El posicionamiento en interiores define el proceso para obtener la localización de un dispositivo o usuario en un entorno o ambiente interior (Zafari et al., 2019). Esta línea de investigación ha sido ampliamente estudiada en los últimos años, principalmente en entornos de aplicación para personas mayores con la implementación de redes inalámbricas de sensores (WSN) en conjunto con entornos inteligentes. La principal razón de la proliferación a gran escala de esta tecnología se debe al desarrollo de nuevos smartphones y dispositivos wearables con capacidades de comunicación inalámbrica, donde las nuevas tecnologías son ahora más aplicables, como BLE, WiFi y UWB (Marquez, 2017).

2.1.2 Banda ultraancha (Ultra Wide-Band, UWB)

La banda ultraancha o ultra wide-band (UWB) es una tecnología de radio que puede utilizar un nivel de energía muy bajo para comunicaciones de corto alcance y gran ancho de banda en una gran parte del espectro radioeléctrico (García Gutiérrez, 2016). UWB tiene un espectro de frecuencias que va de 3,1 a 10,6 GHz, donde la señal tiene un ancho de banda fraccional superior a 0,20 y ocupa un ancho de banda superior a 500 MHz (Zafari et al., 2019).

Los sensores UWB que operan en la banda de frecuencias de 5 GHz tienen diferentes características, una de las más importantes es su capacidad para detectar no sólo los objetivos situados en la línea de visión (LOS), sino también el objetivo situado detrás de un obstáculo no metálico (escenarios sin línea de visión (NLOS)) (Sachs, 2013).

En comparación con WiFi y Bluetooth, UWB presenta varias diferencias clave en cuanto a especificaciones[3]:

- Ancho de banda: UWB tiene un ancho de banda mucho mayor que WiFi o Bluetooth. Esto le permite transmitir más datos a la vez, pero también significa que es más susceptible a las interferencias de otras fuentes.
- Alcance: UWB tiene un alcance menor que WiFi o Bluetooth. Esto se debe a que utiliza niveles de potencia más bajos y propaga su señal en un rango de frecuencias más amplio.
- Latencia: UWB tiene menor latencia que WiFi o Bluetooth. Esto se debe a que utiliza una técnica de espectro ensanchado de secuencia directa (DSSS) que permite una transmisión de datos más rápida.
- Consumo de energía: UWB consume menos energía que WiFi o Bluetooth. Esto la hace más adecuada para su uso en dispositivos alimentados por batería.

[3] Hay que tener en cuenta que existen muchas variantes de UWB, Bluetooth y WiFi, con diferentes capacidades y características. Las comparaciones mostradas son en términos generales.

- Seguridad: UWB tiene un mayor nivel de seguridad que WiFi o Bluetooth. Como utiliza un rango de frecuencias muy amplio, es más difícil para un atacante interceptar o interferir la señal.

A modo de resumen, en la siguiente tabla se recogen las principales características de UWB y se comparan con las de otras tecnologías de comunicación inalámbricas más comunes, como son WiFi y Bluetooth Low Energy.

Especificaciones	UWB	WiFi	BLE
Diseño orientado a	Entornos WPAN	Conexión a WLAN para dispositivos	Intercambio de datos en distancias cortas
Tipo de red	Malla	Estrella	Malla
Consumo de energía	Muy bajo	Alto	Bajo
Rango	1 – 10 metros	20 metros	10 – 100 metros
Rendimiento	Entre 110 y 480 Mbps.	900 Mbps	24 Mbps
Número máximo de nodos	Determinado por el ancho de banda disponible, la potencia de salida de los dispositivos y la presencia de interferencias.	Depende del router	7
Banda operativa	Entre 3.1 y 10.6 GHz	2.4 y 5 GHz	Entre 2.4 y 2.485 GHz
Estándar IEEE	IEEE 802.15.4a, IEEE 802.15.4z, IEEE 802.15.4-2011 y IEEE 802.15.6[4]	802.11	802.15.1

[4] El IEEE ha desarrollado varios estándares para la tecnología UWB. La elección de la norma depende de la aplicación específica y del equilibrio entre velocidad de transmisión de datos, consumo de energía y resistencia a las interferencias.

Espectro ensanchado	Espectro ensanchado de secuencia directa (DSSS) y Espectro ensanchado por salto de frecuencia (FHSS)	DSSS	Espectro adaptativo de salto de frecuencia (AFH)
Seguridad	Autenticación y cifrado mediante protocolos robusto como AES. Seguridad física inherente a sus características de mayor ancho de banda.	Cifrado de acceso protegido Wi-Fi (WPA2)	Confidencialidad, autenticación y derivación de claves.
Modulación	Multiplexación por división de frecuencia ortogonal (OFDM) y Multiplexación por división de frecuencia ortogonal multibanda (MB-OFDM)	Modulación por desplazamiento de fase en cuadratura (QPSK)	Modulación de desplazamiento de frecuencia gaussiana (GFSK)

Tabla 1. ***Comparativa de especificaciones entre las tecnologías UWB, WiFi y BLE***

UWB es una tecnología especialmente atractiva para la localización en interiores debido principalmente a su inmunidad frente a interferencias, al contrario que otras como WiFi o Bluetooth, y su capacidad para proporcionar una precisión muy alta. Como puntos débiles podríamos destacar que su alcance es reducido, requiere un hardware más concreto aún no popularizado, no se encuentra en un nivel de estandarización estable como otras tecnologías previas y su todavía coste elevado. En la siguiente tabla se muestran los pocos dispositivos móviles que integran chip UWB a día de hoy, como se puede observar únicamente los modelos de alta gama de los principales fabricantes ostentan este tipo de tecnología.

Marca	Dispositivo	Precio base	Lanzamiento	Modelo	Chip UWB
Apple	iPhone 11	499 €	Septiembre 2019	Todos los modelos	Apple U1
	iPhone 12	629 €	Octubre 2020	Todos los modelos	Apple U1
	iPhone 13	762 €	Septiembre 2021	Todos los modelos	Apple U1
	iPhone 14	919 €	Septiembre/Octubre 2022	Todos los modelos	Apple U1
Google	Pixel 6	777 €	Octubre 2021	Modelo Pro	Qorvo DW3720
	Pixel 7	809 €	Octubre 2022	Modelo Pro	Qorvo DW3720
	Galaxy Note 20	600 €	Agosto 2020	Modelo Ultra	NXP SR100T
	Galaxy S21	550 €	Enero 2021	Modelos Plus y Ultra	NXP SR100T
Samsung	Galaxy S22	653 €	Febrero 2022	Modelos Plus y Ultra	NXP SR100T
	Galaxy S23	1.209 €	Febrero 2023	Modelos Plus y Ultra	NXP SR100T
	Galaxy Z Fold2	779 €	Septiembre 2020	Todos los modelos	NXP SR100T
	Galaxy Z Fold3	1.091 €	Agosto 2021	Todos los modelos	NXP SR100T
	Galaxy Z Fold4	1.388 €	Agosto 2022	Todos los modelos	NXP SR100T
Xiaomi	MIX4	1.042 €	Septiembre 2021	Todos los modelos	NXP SR100T

Tabla 2. *Lista de smartphones con compatibilidad con la tecnología UWB (Fuente: Wikiwand.com)*

2.1.3 Métodos de medición de distancia para posicionamiento en interiores

Existen diferentes métodos para determinar la localización en interiores con UWB y otras tecnologías. En este punto se explican algunas de las propuestas más importantes utilizadas en diferentes estudios, que destacan por su implementación exitosa y por los buenos grados de precisión que han conseguido obtener.

- Ángulo de llegada (Angle-of-arrival, AOA)

En este método, la distancia entre los sensores o dispositivos se calcula mediante la intersección de líneas angulares para cada emisión de señal. La desventaja de este método es que a medida que aumenta la distancia entre sensores

y dispositivos la precisión que se obtiene disminuye (Gutiérrez Carrero, 2018). En la siguiente Figura 6 se muestra un esquema que ilustra el funcionamiento de este método.

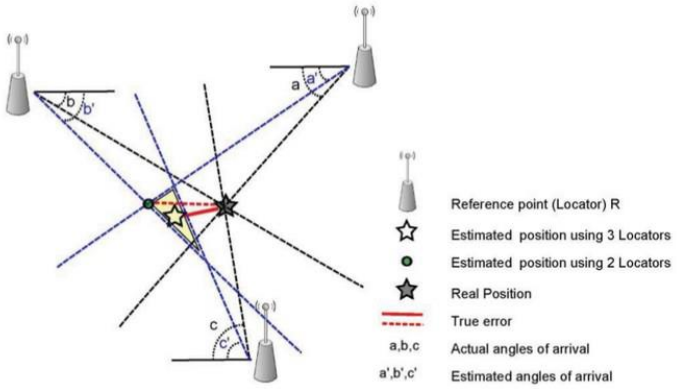


Figura 6. *Método Angle-of-Arrival (AOA) (Alarifi et al., 2016)*

➢ Tiempo de llegada (Time-of-arrival, TOA)

En este método la información de posicionamiento se obtiene a partir de la intersección de las circunferencias generadas por varios sensores de transmisión medidas a partir del tiempo de llegada de paquetes, donde sus radios representan las distancias entre los anclajes y el objetivo (ver Figura 7). Este algoritmo requiere que todos los sensores de emisión de señales estén correctamente sincronizados (Gutiérrez Carrero, 2018).

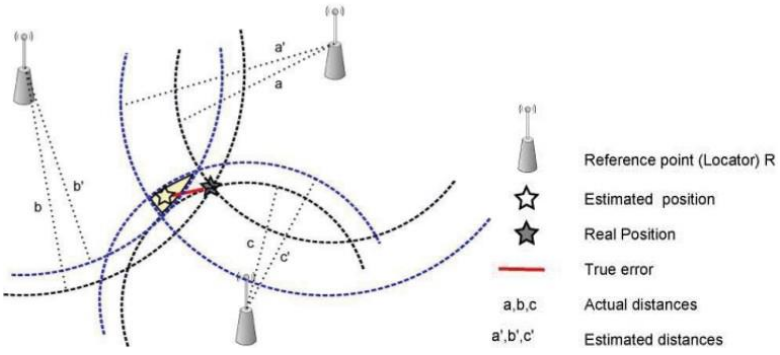


Figura 7. *Método Time-of-Arrival (TOA) (Alarifi et al., 2016)*

➢ Diferencia de tiempo de llegada (Time difference-of-arrival, TDOA)

Con este método, el posicionamiento de un sensor o dispositivo se obtiene midiendo la diferencia en el tiempo de llegada de una señal enviada por él a tres o más sensores receptores, por ejemplo, las balizas que se muestran en el esquema de la Figura 8.

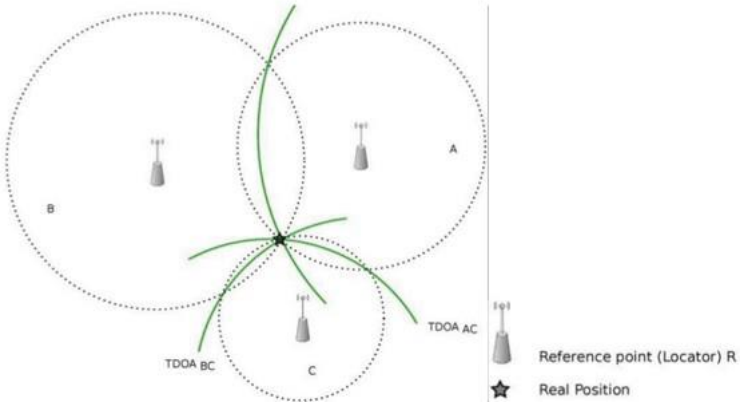


Figura 8. ***Método Time Difference-of-Arrival (TDOA) (Alarifi et al., 2016)***

➢ Indicador de intensidad de señal recibida (Received Signal Strength Indication, RSSI)

Es el método con menor rendimiento y mayor error de precisión aunque es el que requiere menor consumo de energía y recursos para implementarse. En este método, el posicionamiento se obtiene midiendo la distancia de un sensor o dispositivo con respecto a la atenuación generada por la propagación de la señal desde el sensor de transmisión hasta el sensor de recepción. La movilidad de los sensores y la variabilidad impredecible del canal de comunicación generan grandes errores de precisión (Gutiérrez Carrero, 2018).

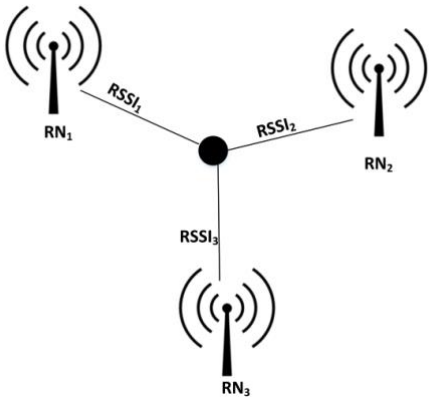


Figura 9. ***Localización basada en RSSI (Zafari et al., 2019)***

2.1.4 Algoritmos de posicionamiento en interiores

A continuación, se describen los principales tipos de algoritmos que se utilizan actualmente para la estimación de la localización en interiores, el primero de ellos se fundamenta en métodos geométricos mientras que el segundo se basa en el uso de la intensidad de la señal recibida (RSSI) junto con algoritmos de proximidad, como kNN. Estos algoritmos han destacado debido a la gran eficiencia y versatilidad que han demostrado en estudios previos.

- Trilateración y triangulación

Estos algoritmos determinan la ubicación de un sensor o dispositivo utilizando la geometría de los círculos y la medición de las distancias y ángulos de los sensores que se encuentran en el área de medición (Bagaric, 2016). Se utiliza ampliamente junto con métodos de medición que hemos mencionado anteriormente, tales como TOA, TDOA y RSSI. A nivel teórico, este método devolvería una respuesta única y exacta en la intersección de los círculos (ver Figura 10), sin embargo las señales de radio son extremadamente variables, particularmente en entornos de interior debido a las reflexiones y refracciones que se producen en obstáculos y esquinas, así como a los cambios que se produzcan en el entorno o al movimiento de personas en la zona, por lo cual lo que ocurre en la realidad es que estos círculos no intersecan en ningún punto común y la posición debe ser estimada intentando minimizar las distancias entre

la posición que se estima y todos los círculos mediante técnicas matemáticas como el método de los mínimos cuadrados (Cook et al., 2005).

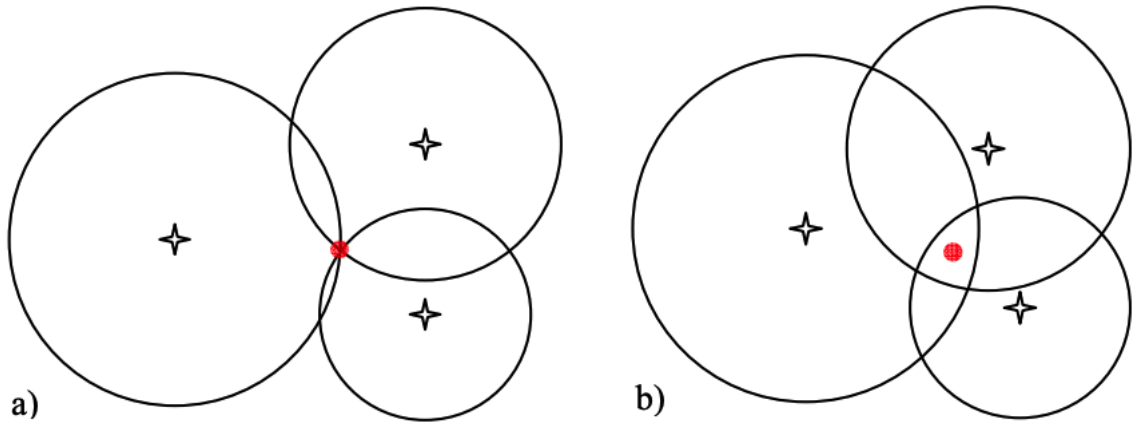


Figura 10. *Estimación de la posición mediante trilateración a) con información perfecta e invariable (teórica) b) con perturbaciones (real) (Cook et al., 2005)*

➢ Fingerprinting

El Fingerprinting de la intensidad de señal recibida es un método que se basa en la creación de una base de datos con las intensidades de señal en el escenario o área de aplicación y la estimación de la localización en base a las mismas, y el uso de un mapa de estas distribuciones para localizar una determinada muestra RSSI (Bagaric, 2016).

El algoritmo comienza recopilando una "huella digital" de la intensidad de la señal inalámbrica en varias ubicaciones del entorno. Esta huella digital es un conjunto de valores RSSI (intensidad de señal recibida) en cada ubicación, registrados desde múltiples puntos de acceso. Los valores RSSI suelen medirse en dBm (decibelios-milivatios), una unidad de medida utilizada para expresar el nivel de potencia de una señal eléctrica, como una señal de radiofrecuencia (RF), en decibelios (dB) respecto a un milivatio (mW). Esta unidad se utiliza habitualmente en telecomunicaciones e ingeniería de radiofrecuencias. Un nivel de potencia de 0 dBm equivale a 1 mW, y cada aumento de 3 dB representa una duplicación de la potencia. Por ejemplo, una señal con un nivel de potencia de 10 dBm es diez veces más potente que una señal con un nivel de potencia de 0 dBm (o 1 mW). Del mismo modo, una señal con un nivel de potencia de -10 dBm es una décima parte de potente que una señal con un nivel

de potencia de 0 dBm (o 1 mW). Una vez recogidas las huellas digitales, normalmente se almacenan en una base de datos en lo que se conoce como fase de entrenamiento.

Después en la fase de estimación, cuando un dispositivo necesita determinar su ubicación, mide los valores RSS de los puntos de acceso o balizas que puede ver y los compara con las etiquetas de la base de datos. A continuación, mediante algún algoritmo de proximidad como k-Nearest Neighbor (kNN) (Zhang & Feng, 2017), se estima la ubicación del dispositivo.

El RSS fingerprinting es un método muy utilizado para la localización en interiores porque es relativamente fácil de implementar y no requiere ningún hardware adicional en el dispositivo. Sin embargo, es sensible a los cambios en el entorno, como la adición de nuevos muebles u objetos, y a la densidad de las etiquetas recogidas durante la fase de entrenamiento.

- Técnicas híbridas

Para producir una estimación más robusta de la posición, se puede utilizar una combinación de parámetros relacionados con la posición. Un esquema híbrido de estimación de la posición proporciona heterogeneidad a las redes de sensores, en términos de sincronización temporal, capacidades de enrutamiento de los dispositivos de red y alcance de las comunicaciones (Mazhar et al., 2017). Además, en la comunicación de banda ancha de corto alcance, las técnicas híbridas proporcionan significativamente el menor error de posicionamiento posible (Alarifi et al., 2016).

2.1.5 Inteligencia artificial, aprendizaje automático y aprendizaje profundo

El aprendizaje profundo ha surgido de la investigación en inteligencia artificial y aprendizaje automático. La siguiente figura ilustra la relación entre inteligencia artificial, aprendizaje automático y aprendizaje profundo:

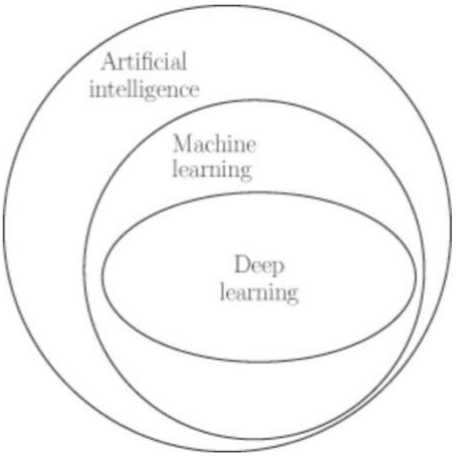


Figura 11. ***Relación entre inteligencia artificial, aprendizaje automático y aprendizaje profundo (Kelleher, 2019)***

El campo de la inteligencia artificial nació en un seminario celebrado en el Dartmouth College en el verano de 1956. En él se presentaron investigaciones sobre diversos temas, como la demostración matemática de teoremas, el procesamiento del lenguaje natural, la planificación de juegos, los programas informáticos capaces de aprender a partir de ejemplos y las redes neuronales. El campo moderno del aprendizaje automático se basa en los dos últimos temas: los ordenadores que pueden aprender a partir de ejemplos y la investigación sobre redes neuronales (Janiesch et al., 2021).

El aprendizaje automático (Machine Learning) implica el desarrollo y la evaluación de algoritmos que permiten a un ordenador extraer (o aprender) funciones a partir de un conjunto de datos (conjuntos de ejemplos). Para entender lo que significa el aprendizaje automático se debe distinguir entre:

- Un conjunto de datos, en el contexto del aprendizaje automático, es una colección de puntos de datos y/o instancias que se utiliza para entrenar y/o probar modelos de aprendizaje automático. Los puntos de datos de un conjunto de datos se organizan normalmente en características/atributos y pueden utilizarse para entrenar algoritmos que realicen predicciones, clasifiquen instancias y lleven a cabo otras tareas de aprendizaje automático. La calidad y el tamaño de un conjunto de datos pueden influir significativamente en el rendimiento de los modelos de aprendizaje

automático, por lo que la preparación y el tratamiento de los datos es un paso fundamental en el proceso de aprendizaje automático.

- Un algoritmo es un proceso o una serie de pasos que puede seguir un ordenador para resolver un problema concreto. En el contexto del aprendizaje automático, un algoritmo define un proceso para analizar un conjunto de datos e identificar patrones recurrentes en ellos.

- Una función es una correspondencia determinista entre un conjunto de valores de entrada y uno o varios valores de salida. El hecho de que el mapeo sea determinista significa que, para cualquier conjunto específico de entradas, una función siempre devolverá las mismas salidas. Una función se puede representar de muchas formas distintas, como una operación aritmética, una secuencia de reglas if-then-else u otras representaciones mucho más complejas.

Una forma de representar una función es utilizar una red neuronal. El aprendizaje profundo (Deep Learning) es el subcampo del aprendizaje automático que se centra en modelos de redes neuronales profundas. De hecho, los patrones que los algoritmos de aprendizaje profundo extraen de los conjuntos de datos son funciones que se representan como redes neuronales. La siguiente figura ilustra la estructura de una red neuronal.

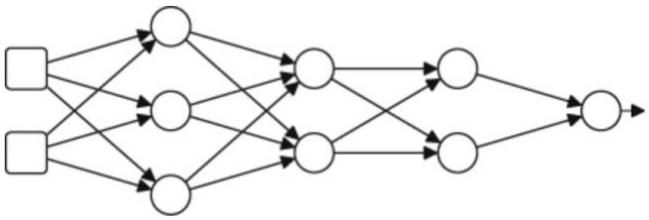


Figura 12. *Esquema de una red neuronal (Kelleher, 2019)*

Los cuadrados de la izquierda en el esquema anterior representan los datos de entrada. Después, cada uno de los círculos de esta figura se denomina neurona y cada neurona ejecuta una función: toma una serie de valores como entrada y los asigna a un valor de salida. Las flechas de la red muestran cómo las salidas de cada neurona pasan como entradas a otras neuronas. En esta red, la información fluye de

izquierda a derecha. Una red neuronal utiliza una estrategia de divide y vencerás para aprender una función: cada neurona de la red aprende una función simple, y la función global (más compleja), definida por la red, se crea combinando estas funciones más simples.

El machine learning implica un proceso de dos pasos: entrenamiento e interpretación. Durante el entrenamiento, un algoritmo de aprendizaje automático procesa un conjunto de datos y elige la función que mejor se ajusta a los patrones de los datos. La función extraída se codificará en un programa informático de una forma determinada (como reglas if-then-else o parámetros de una ecuación especificada). La función codificada se conoce como modelo, y el análisis de los datos para extraer la función suele denominarse entrenamiento del modelo. En esencia, los modelos son funciones codificadas como programas informáticos. Sin embargo, en el aprendizaje automático los conceptos de función y modelo están tan estrechamente relacionados que a menudo se omite la distinción y los términos pueden incluso utilizarse indistintamente.

En el contexto del Deep Learning, la relación entre funciones y modelos es que la función extraída de un conjunto de datos durante el entrenamiento se representa como un modelo de red neuronal y, a la inversa, un modelo de red neuronal codifica una función como un programa informático. El proceso estándar utilizado para entrenar una red neuronal consiste en comenzar el entrenamiento con una red neuronal en la que los parámetros de la red se inicializan aleatoriamente. Esta red inicializada aleatoriamente será muy imprecisa en cuanto a su capacidad para establecer la relación entre los distintos valores de entrada y los resultados previstos para los ejemplos del conjunto de datos. El proceso de entrenamiento consiste en iterar a través de los ejemplos del conjunto de datos y, para cada ejemplo, presentar los valores de entrada a la red y, a continuación, utilizar la diferencia entre la salida devuelta por la red y la salida correcta para el ejemplo que figura en el conjunto de datos, con el objetivo de actualizar los parámetros de la red de modo que se ajuste más a los datos que se han tomado como entrada.

Una vez que el algoritmo de aprendizaje automático ha encontrado una función que es lo suficientemente precisa (en términos de que las salidas que genera

coinciden con las salidas correctas que figuran en el conjunto de datos) para el problema que estamos tratando de resolver, el proceso de formación se ha completado, y el algoritmo devuelve el modelo final. Este es el punto en el que finaliza el entrenamiento en el aprendizaje automático.

Una vez finalizado el entrenamiento, el modelo queda fijado. La segunda etapa del aprendizaje automático y el aprendizaje profundo es la interpretación, es decir, cuando el modelo entrenado se aplica a nuevos ejemplos, ejemplos para los que no conocemos el valor de salida correcto y, por lo tanto, queremos que el modelo genere estimaciones o predicciones de este valor.

Algunos de los campos en los que el Deep Learning ha tenido mucho éxito hasta hoy son la clasificación de imágenes, la detección de objetos, el reconocimiento de voz, el procesamiento del lenguaje natural, los sistemas de recomendación, la robótica, la visión artificial, la sanidad y los servicios financieros.

2.2 Trabajos relacionados

Existen diversos trabajos previos relacionados con la temática de este Trabajo de Fin de Máster, especialmente aquellos que explican los métodos y técnicas utilizados para el posicionamiento en interiores. Se puede encontrar un mosaico de enfoques, donde los protocolos más utilizados son WiFi y Bluetooth, junto con métodos que ayudan a la determinación del posicionamiento (Tariq et al., 2017), como triangulación, análisis de escena, fingerprinting, localización basada en proximidad, análisis de visión, etc. Más concretamente, los estudios sobre el posicionamiento en interiores con Ultra Wide Band (UWB) son relativamente nuevos y la cantidad de trabajos que explican su funcionamiento son escasos.

En esta sección se realiza un análisis de los trabajos que fueron tomados en cuenta para la preparación de la propuesta de este Trabajo de Fin de Máster. Para llevar a cabo esta selección de los trabajos mencionados que están relacionados con la tecnología UWB se tuvieron en cuenta tres criterios:

I. que los trabajos hayan sido publicados como máximo en los últimos cinco años,
II. que los trabajos relacionen la tecnología UWB para posicionamiento y
III. que los trabajos muestren algún tipo de aplicación con UWB o IPS.

De acuerdo con lo anterior, esta sección se divide en dos grupos, trabajos sobre posicionamiento en interiores a nivel general y trabajos que relacionan la tecnología UWB con el posicionamiento en interiores.

2.2.1 Trabajos sobre posicionamiento en interiores

En este punto hemos incluido un abanico heterogéneo de trabajos que se basan en distintas tecnologías y protocolos, como WiFi y Bluetooth, combinados con métodos diferentes como triangulación, análisis de escena, localización basada en proximidad o fingerprinting entre otros.

Hasta el día de hoy, existen varios trabajos que explican el funcionamiento de los sistemas de posicionamiento en interiores en el mundo científico. Existen autores centrados en los sistemas de posicionamiento de tipo No-GPS. Por ejemplo, en el artículo de Tariq et al (Tariq et al., 2017) se presenta un estudio sobre las técnicas de localización en interiores y las técnicas híbridas de localización en interiores. El trabajo presentado es general, explicando las ventajas y desventajas de cada técnica y tecnología estudiada. Del mismo modo, existe un trabajo similar de Alvarado y Delgado (Cruz Alvarado & Sandí Delgado, 2017), donde se explican los sistemas y tecnologías que facilitan el posicionamiento en interiores.

Por otro lado, el trabajo de Zafari et al (Zafari et al., 2019), es una propuesta de referencia sobre el IPS que se centra en la evaluación de diferentes sistemas para el posicionamiento en interiores, pero desde la perspectiva de la eficiencia energética, disponibilidad, coste, rango de recepción, latencia, escalabilidad y precisión de seguimiento.

Correa et al (Correa et al., 2017), estudian el desarrollo de sistemas de posicionamiento en interiores desde el punto de vista de la infraestructura y la

metodología empleada. Principalmente, el documento relacionado habla de sistemas basados en red, sistemas basados en inercia y sistemas híbridos de posicionamiento.

Por otra parte, Bagaric (Bagaric, 2016), se centra en mejorar la estimación del posicionamiento de dispositivos móviles dentro de espacios cerrados, siendo la novedad respecto a otros trabajos similares el estudio sobre posicionamiento en interiores con el uso de tecnologías como Raspberry Pi al que se le añade un módulo receptor FM (Frecuencia Modular) en el proceso.

Dentro de los estudios generales, destacamos otros trabajos que explican las situaciones relacionadas con casos y contextos específicos como una técnica concreta o algún énfasis en una tecnología específica. Respecto a estos trabajos hemos destacado varios que se basan en el análisis del error de la medida de localización con WiFi (Basiri et al., 2017) (Guashima & Geovanny, 2013) y posicionamiento inercial (Regueiro Senderos, 2014), donde lo más relevante es la conclusión sobre la principal causa del error común en estos sistemas, que son:

i) para la tecnología WiFi la variabilidad de la señal recibida, y
ii) en los sistemas inerciales el error de precisión en el tiempo.

Concretamente, Basiri et al (Basiri et al., 2017), se centra principalmente en el sistema inercial, concluyendo que la precisión es baja, lo que significa que sigue siendo un problema abierto.

Un sitio web que trata sobre el error de posicionamiento es el de POZYX (*Pozyx Academy | How Positioning Works*, n.d.), donde se explica esta situación con dispositivos (tag), que son sensores que trabajan con tecnología UWB y determinan la posición en 3D o 2D, según la configuración de los mismos, la conclusión más importante a la que se llegó fue que es posible reducir el error de posicionamiento a aproximadamente 10 cm.

En cuanto a los trabajos que relacionan el análisis sobre técnicas específicas de posicionamiento en interiores, encontramos a Díaz (Díaz Gonzalez, 2017) entre otros. El uso de técnicas de fingerprinting con tecnología WiFi es estudiado por López (López Justicia, 2018), que concluye que esta técnica requiere la calibración y evaluación del

espacio interior. Además, con el objetivo de reducir el error, se diseñó un algoritmo denominado: Dynamic Space Warping (DSW) que calcula la posición de los dispositivos. Una desventaja de este trabajo es el error de posicionamiento obtenido, que fue de 2 metros.

En la misma línea, siguiendo con el estudio de la técnica de fingerprinting, tenemos el trabajo de Trevisan (Trevisan Troche, 2017), que trata del análisis de la afectación de la presencia de varias personas en el cálculo del posicionamiento en interiores mediante tecnología WiFi y su comparación con un sistema basado en el campo magnético. En este trabajo la principal conclusión es que ninguna de las tecnologías estudiadas ofrece las mínimas garantías de error, sin embargo, concluye que los sistemas con WiFi son mejores respecto al sistema basado en campo magnético. Pero el error sigue siendo elevado para el posicionamiento en interiores, pues incluso con WIFI, en los casos estudiados el error estimado fue de aproximadamente 2 metros.

Por otro lado, Díaz (Díaz González, 2017), se centra en el posicionamiento en interiores con Bluetooth y técnicas de trilateración. Este trabajo concluye estableciendo que la trilateración es mejor que el fingerprinting en las pruebas realizadas, ya que reduce el error, pero no presenta una medida aproximada.

Pereira y Polo (Pereira Tapiro & Polo Poveda, 2015), proponen una solución con un prototipo de localización, que funciona con Bluetooth Low Energy, RSSI y triangulación. Estas tecnologías ya han sido utilizadas anteriormente en otros estudios, y las conclusiones siguen siendo que son aptas para el uso en sistemas de localización en interiores, sin embargo, el error de precisión sigue siendo grande.

Para finalizar esta primera parte, detallamos otros trabajos que relacionan diferentes opciones para el posicionamiento en interiores. Por ejemplo Jiménez y Seco (Jiménez & Seco, 2017), se centran en sistemas de posicionamiento en interiores con teléfonos móviles, donde los autores presentan una evaluación experimental de diferentes enfoques para encontrar un objeto de interés respecto a la disposición en la que el usuario mantiene el dispositivo móvil. Islam et al (Islam et al., 2018), proponen el diseño de un esquema de posicionamiento en interiores combinando dos algoritmos:

i) una Red Neuronal Artificial (Artificial Neural Networks, ANN), y
ii) un algoritmo de lógica difusa.

La ANN trabaja con la distancia euclidiana y entrena la información obtenida con la optimización de la técnica de retropropagación por redes neuronales, generando resultados de posicionamiento para el algoritmo de lógica difusa. Islam et al (Islam et al., 2018), afirman que los resultados obtenidos tuvieron una precisión de casi el 99% para el posicionamiento.

Por último, Pabón (Pabón Dueñas, 2017), propone un trabajo sobre una red de navegación en interiores, donde el objetivo es la optimización de rutas internas de posicionamiento para cualquier tipo de persona. El trabajo trata principalmente sobre rutas de orientación para peatones, con el objetivo de ayudar a las personas para que se ubiquen correctamente dentro del edificio en el que se encuentran, pero en este trabajo no se profundiza en las técnicas o tecnologías al respecto.

2.2.2 Trabajos sobre posicionamiento en interiores con UWB

Como ya se ha comentado, la tecnología UWB (banda ultraancha) es un tipo de comunicación inalámbrica que utiliza un amplio espectro de frecuencias para transmitir datos a corta distancia. Esto permite una comunicación de alta velocidad, bajo consumo y localización precisa. Suele emplearse en aplicaciones como redes inalámbricas de área personal, audio y vídeo inalámbricos y posicionamiento en interiores (Fowler et al., 1990). En esta sección hemos recopilado algunos trabajos sobre posicionamiento en interiores utilizando UWB. Comenzamos con los trabajos que relacionan una comparación o evalúan el rendimiento de esta tecnología de comunicación inalámbrica.

Para empezar, encontramos estudios como el propuesto por Caso et al (Caso et al., 2018), donde los autores se centran en la comparación de UWB con WiFi como tecnologías de comunicación a combinar con el uso de técnicas de fingerprinting. Las conclusiones de este trabajo fueron rotundas, pues los autores destacaron que UWB supera a WIFI en este tipo de escenarios.

Barral et al (Barral et al., 2019), hablan de UWB y de una plataforma llamada PLUS RTLS, donde se utilizan principalmente las técnicas de RSS y TDOA, buscando mejorar el algoritmo de posicionamiento. Los resultados publicados son muy buenos y nuevamente la tecnología UWB demuestra que la precisión que maneja es superior a la que se obtiene mediante otras tecnologías.

Un documento interesante que trata sobre el análisis del canal UWB es el presentado por Martínez (Martínez García, 2010), donde se explica la evolución de esta tecnología, sus características y sus usos, también encontramos diferentes aplicaciones sobre UWB y la diferencia con otras tecnologías, evidenciando una vez más que UWB es una excelente opción para su aplicación en sistemas de posicionamiento en interiores.

Por otro lado, también se encuentran disponibles propuestas con aplicación específica. Por ejemplo, Gutiérrez (Gutiérrez Carrero, 2018), explica el uso de la UWB con el sistema POZYX, donde utiliza principalmente la técnica TOA. Por su parte, los resultados en este trabajo fueron satisfactorios, pero el autor expresa que esta tecnología aún necesita estudios más profundos.

En otra propuesta, los autores Kocur et al (Kocur et al., 2017) proponen un estudio sobre el seguimiento de múltiples personas en movimiento mediante sensores UWB, donde se analiza el efecto de apantallamiento mutuo y su relación con el posicionamiento en interiores utilizando sensores UWB. Los resultados presentados son muy buenos, pero los autores afirman que todavía hay espacio para más mejoras y propuestas de aplicaciones para la vida real.

Yao et al (Yao et al., 2017), se centra en la aplicación de la tecnología UWB en sistemas de posicionamiento en interiores, en combinación con la técnica de la unidad de medición inercial (Inertial Measurement Unit, IMU). Además en dicho estudio los autores añaden el filtro de Kalman para mejorar la precisión y reducir las interferencias que generan las personas en los sistemas de posicionamiento en interiores. En las diferentes pruebas y simulaciones realizadas, los autores mostraron mejoras al utilizar los métodos y tecnologías mencionadas.

Otro de los trabajos interesantes sobre seguimiento de personas en tiempo real con sistemas de posicionamiento en interiores es el presentado por Drutarovsky et al (Drutarovský et al., 2017), donde los autores discuten la arquitectura modular de sensores UWB (UWB-SN) que, según afirman, puede utilizarse para el rápido desarrollo y prueba de nuevas aplicaciones de seguimiento de personas en tiempo real.

En cuanto a los estudios de la tecnología UWB que se centran en analizar su precisión, encontramos como trabajos relevantes los mencionados por Mazhar et al (Mazhar et al., 2017) y Tiemann y Wietfeld (Tiemann & Wietfeld, 2017). Estas propuestas explican principalmente cómo los sistemas UWB pueden ser más precisos. Más concretamente, en el trabajo de Mazhar et al (Mazhar et al., 2017) se revisan los diferentes métodos, algoritmos e implementaciones con UWB, principalmente para garantizar la precisión y su comparación con otras tecnologías como WiFi, Bluetooth y ZigBee. Por su parte, Tiemann y Wietfeld[49], se centran en la precisión con UWB utilizando TDOA con vehículos aéreos no tripulados (UAV) multirrotores, siendo este último trabajo novedoso e interesante por los objetivos sobre el rendimiento con UWB y la precisión que se manejan.

En este grupo de trabajos relacionados podemos destacar también algunos trabajos sobre una tecnología que soporta UWB, denominada IEEE 802.15.4a, que es un estándar del grupo "802.15" del IEEE, donde se han incluido dos niveles físicos adicionales respecto al estándar original denominado IEEE 802.15.4, pues uno de estos niveles contiene UWB.

El primer trabajo relacionado con UWB+IEEE 802.15.4a es el de Suarez y Llano (Suárez Páez & Llano Ramírez, 2010), donde presentan una revisión sobre esta tecnología, centrándose en describir las características técnicas y funcionamiento de UWB con IEEE 802.15.4a, haciendo hincapié en describir el funcionamiento del canal IR-UWB (Impulsive Radio Ultra Wide Band). Otros trabajos interesantes que siguen este enfoque son los propuestos por De Rivaz et al (de Rivaz et al., 2007) y Karapistoli et al (Karapistoli et al., 2010), donde se detalla y explica el funcionamiento y prestaciones del modelo de canal IEEE 802.15.4a con UWB.

En cuanto a estudios específicos de posicionamiento en interiores con UWB e IEEE 802.15.4a, existen varios trabajos, que analizan las diferentes técnicas generales de este proceso, como los propuestos por García Gutiérrez (García Gutiérrez, 2016) y Lagunas et al (Lagunas et al., 2014).

En otros artículos similares, como los propuestos por Barua et al (Barua et al., 2017) o Silva & Hancke (Silva & Hancke, 2017), los autores proponen un enfoque basado en el tiempo de llegada (TOA) utilizando tecnologías UWB+IEEE 802.15.4a a 2,4 GHz para conseguir una mayor precisión en la localización.

Por otro lado, Benini et al (Benini et al., 2011) explican el uso de UWB+IEEE 802.15.4a haciendo referencia al filtro de Kalman (McElroy et al., 2014) y al sistema IMU con estas tecnologías.

Para acabar con los trabajos que estudian el modelo de canal IEEE 802.15.4a, Tiemann (Tiemann, 2016), relaciona el posicionamiento en interiores con UWB e IEEE 802.15.4a utilizando TDOA. La principal conclusión sobre este trabajo según el autor es que, aunque la precisión del sistema es suficiente para la mayoría de las aplicaciones, el rechazo multitrayectoria podrí mejorarse en futuros trabajos para permitir un funcionamiento robusto en entornos abarrotados.

Para terminar, desde nuestra Universidad los investigadores recientemente también han publicado estudios relacionados con el posicionamiento en interiores y UWB. Por ejemplo, en la propuesta de Campaña Bastidas et al (Campaña Bastidas et al., 2022) se analiza las posibilidades de UWB frente a otras tecnologías de comunicación inalámbrica y se propone un diseño de sistema de localización en interiores basado en dicha tecnología junto con técnicas de fingerprinting, mediante el cual se pudo comprobar la precisión de la tecnología UWB.

Por otro lado, en la propuesta de Polo y Medina (Polo-Rodriguez & Medina-Quero, 2022) se describe un modelo computacional basado en sensores POZYX UWB que discrimina la activación de sensores binarios en entornos con múltiples ocupantes para distinguir qué usuario interaccionó con los sensores ambientales en función de la proximidad.

Para terminar esta revisión de los trabajos relacionados con la temática del presente trabajo podemos concluir que la propuesta planteada en este trabajo supone una vertiente poco explorada de los sistemas de posicionamiento en interiores, puesto que como se ha demostrado en esta revisión, existen multitud de pruebas y estudios que indican que la tecnología UWB es interesante para estas aplicaciones, pero muy pocas propuestas combinan esta tecnología con el aprendizaje a partir de las mediciones de RSSI en un instante de tiempo con otras previamente guardadas mediante el uso de modelos de Deep Learning usando como receptores los propios dispositivos que portan los usuarios. Todo esto, unido al aspecto más novedoso: la utilización de un dispositivo móvil como herramienta de captación de datos UWB y generadora de etiquetas de la posición del usuario supone que la propuesta de este TFM sea novedosa y atractiva. Este enfoque permitirá abrir la puerta a sistemas de localización de menor coste, ya que las balizas ambientales son notablemente más baratas que las anclas UWB, así como procesar la localización de forma individual y centralizada en dispositivos de los usuarios, sin que datos sensibles naveguen a la nube.

3 Métodos

Este capítulo describe la arquitectura y el enfoque para estimar la posición del usuario en interiores, detallando el tipo de dispositivos y balizas que se van a implementar, la aplicación móvil que tendrá el papel de herramienta para la recolección de señal UWB y etiquetado de posición en tiempo real, así como el servidor que implementa una API REST con distintos servicios y, por último, el modelo de Deep Learning para la estimación de la localización.

3.1 Arquitectura general

En primer lugar, se presenta un esquema de la arquitectura que integra los distintos dispositivos y componentes de este trabajo.

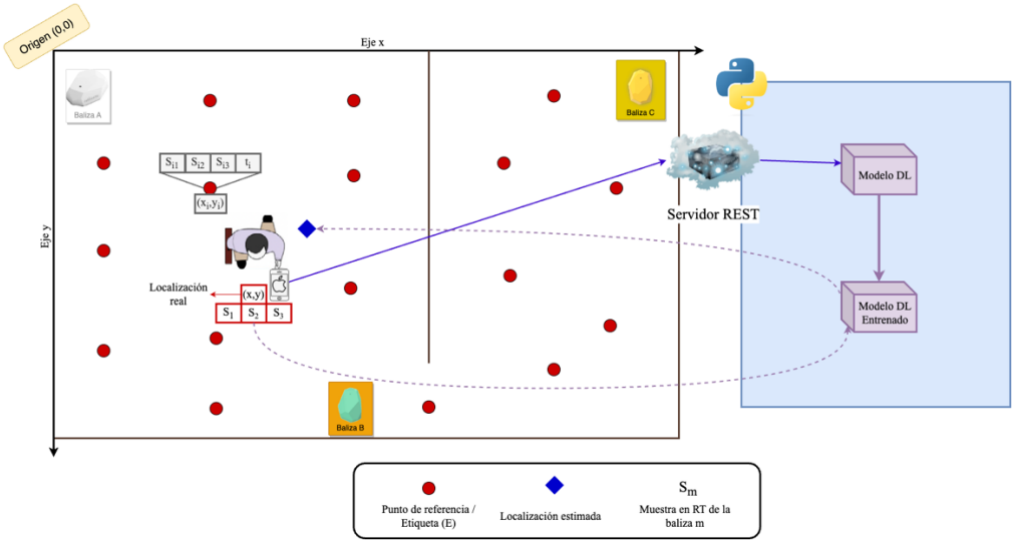


Figura 13. *Arquitectura general de la propuesta de localización en interiores basada en Deep Learning con sensores ambientales UWB*

La figura anterior refleja la esencia de esta propuesta, un sistema de localización de interiores que está formado por distintos componentes desplegados en un entorno inteligente de ámbito doméstico. Entre los distintos componentes encontramos:

- Dispositivos baliza con tecnología UWB, que se encargan de comunicarse con la herramienta de captura de datos UWB y enviarle sus datos de localización con respecto al dispositivo móvil que actúa como referencia. Como se puede observar en la figura anterior, se propone el despliegue

de tres balizas para controlar un entorno doméstico formado por dos habitaciones contiguas.

- Dispositivo móvil, que actúa como herramienta de recolección de datos y etiquetado durante el proceso de fingerprinting gracias a una aplicación desarrollada en Swift para dicho propósito. Este dispositivo será portado por el usuario en todo momento y actúa como referencia de la localización.

- Servidor REST, que recopila, procesa y almacena la información enviada desde el dispositivo móvil para generar el dataset que será empleado como entrada para el modelo de Deep Learning. Este servidor será implementado en Python y desplegará un servicio de API REST con distintos servicios o endpoints en un nodo Fog.

- Modelo Deep Learning, que será el elemento del sistema encargado de estimar la posición del usuario mediante los datos arrojados por las balizas UWB al dispositivo móvil una vez haya sido debidamente entrenado con los datos de la fase previa de etiquetado.

Durante el resto del presente capítulo se detallará cada uno de estos componentes que dan lugar a la propuesta del presente Trabajo de Fin de Máster.

3.2 Dispositivos baliza UWB

En el contexto de un sistema de posicionamiento en interiores, una baliza es un pequeño dispositivo que utiliza alguna tecnología de comunicación inalámbrica (WiFi, BLE, UWB…) para transmitir su ubicación a los dispositivos cercanos. Estos dispositivos, como los dispositivos móviles, pueden utilizar esta información de localización para determinar su propia posición dentro del edificio o zona donde se encuentran las balizas.

Las balizas pueden colocarse en lugares fijos, como paredes o techos, y utilizarse junto con otras tecnologías para proporcionar un posicionamiento más preciso en interiores. Las balizas son un elemento indispensable para el éxito del

posicionamiento, por lo que tanto la elección de un dispositivo adecuado y de calidad que actúe como balizas como su colocación en el área del edificio que se desea controlar con el sistema de posicionamiento son cruciales para el buen desempeño del mismo.

Para esta propuesta se ha seleccionado un modelo de **balizas UWB fabricado por Estimote, Inc**[5]. Esta empresa tecnológica ubicada en New York (USA) está dedicada a servicios y consultoría de TI, y entre sus múltiples líneas de negocio se encuentra la fabricación dispositivos iBeacon atractivos y muy capaces, y SDK que permiten incluso a los programadores con poca experiencia integrarlos con diferentes plataformas de aplicaciones (Android e iOS). Según los datos ofrecidos por la propia empresa, más de 50.000 desarrolladores han hecho uso de sus balizas.

Concretamente, las balizas utilizadas en la propuesta constituyen un nuevo modelo que se encuentra en estado de pre-producción, un kit de desarrollo que aún no dispone de la certificación de equipos de alcance reducido (FCC). Esta certificación consiste en la validación de que el dispositivo cumple con la norma técnica de equipos de alcance reducido y es necesaria para que este tipo de productos pueda ser comercializado. Por tanto, se trata de un modelo novedoso que únicamente está disponible con fines de desarrollo de ingeniería, demostración y/o evaluación.

Figura 14. ***Kit de desarrollo de balizas UWB Estimote (izquierda) y contenido del kit (derecha).***

[5] https://estimote.com/

Las balizas de banda ultraancha de Estimote, en esencia, son pequeños transmisores de radio alimentados por pilas que se pueden fijar a paredes u objetos. Los dispositivos cercanos habilitados para UWB pueden conectarse a estas balizas y calcular medias de localización precisas. Las especificaciones más interesantes de estas balizas son las siguientes:

- Señal de radio UWB: La última generación del chip Ultra Wideband compatible el chip U1 de Apple.
- Señal Bluetooth: Para balizamiento BLE de baja potencia, así como interconexión segura para activar la señal de radio UWB.
- Señal de radio NFC: Basta con tocar la baliza para obtener su identificador o abrir la aplicación correspondiente.
- Sensor de luz ambiental: Se puede emplear para simplificar el despliegue o para optimizar la duración de la batería fuera de un horario concreto.
- Reloj en tiempo real (RTC): Se utiliza para la seguridad avanzada o la gestión de energía basada en el tiempo y la optimización de la batería.
- 2 x Pilas AA: Esta fuente de energía es suficiente para alimentar a esta baliza 2 años en reposo o hasta 4.000 minutos de estado activo.
- Sensor inercial (IMU): Útil para pruebas y desarrollo, o para optimizar la batería de objetos móviles.
- Luz led: A modo de señal, parpadea para indicar una distancia corta, facilitando de este modo las pruebas y el desarrollo.
- Caja de silicona (cubierta exterior): Altamente duradera para despliegues en interiores y exteriores y 100% reciclable.
- Orificio para tornillo de montaje: para una estética de despliegue permanente, permite atornillar la baliza para fijarla contundentemente.

Además de Bluetooth, los smartphones y wearables modernos transmiten señales de radio de banda ultraancha, en concreto los dispositivos móviles de Apple que son compatibles con estas balizas UWB son los iPhone 11 o superiores y los Apple Watch 6 o superiores. Estos dispositivos son capaces de conectarse a estas balizas UWB compatibles y calcular medidas de localización con precisión de centímetros.

Cada baliza UWB es un diminuto computador (ver Figura 15) alimentado por batería que dispone de varios módulos de radio como BLE o UWB (ver Figura 16). A nivel de software, ejecuta un firmware inteligente que transmite periódicamente señales BLE de baja potencia. Puede funcionar con pilas pequeñas durante años.

Figura 15. ***Comparación del tamaño de una baliza UWB de Estimote con una moneda de 1€.***

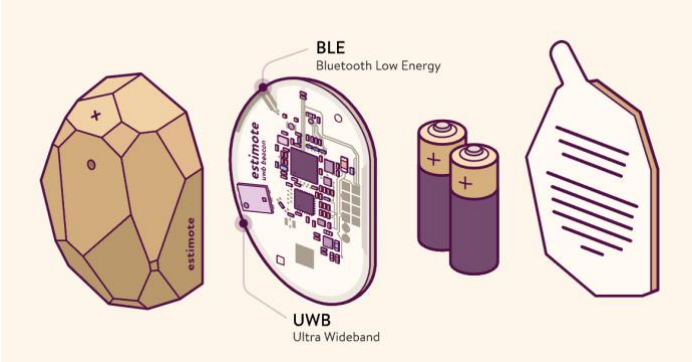


Figura 16. ***Esquema interno de una baliza UWB de Estimote.***

Los dispositivos cercanos con un chip compatible, como el último iPhone con el chip U1[6], captan estas señales BLE. Las aplicaciones compatibles del teléfono inician la localización UWB y calculan la distancia exacta entre los dispositivos. Las múltiples antenas del teléfono ayudan a calcular el ángulo de las señales.

[6] https://support.apple.com/es-es/HT212274

En cuanto a la compatibilidad con dispositivos móviles con sistema operativo Android, hoy en día ya existen en el mercado dispositivos Android con un chip UWB compatible con estas balizas UWB, sin embargo los dispositivos de Apple cuentan con el chip más avanzado y testeado con esta tecnología, por esta razón se ha optado por utilizar un dispositivo móvil iPhone para la implementación de nuestra propuesta.

Es importante destacar que las balizas de Estimote no realizan un posicionamiento 2D en interiores, si no que determinan únicamente la distancia del dispositivo móvil a cada una de ellas. El objetivo de la integración de un modelo de fingerprinting es extender dicha funcionalidad para poder realizar una estimación en interiores de la localización del usuario.

Se puede concluir que la selección de estas balizas nos permitirá realizar de forma sencilla el despliegue de sensores ambientales UWB en un entorno doméstico con un bajo coste, de forma sencilla y una muy buena autonomía, lo cual representa uno de los objetivos principales de este Trabajo de Fin de Máster.

3.3 Aplicación móvil y framework UWB

Uno de los objetivos de este Trabajo de Fin de Máster consiste en el diseño y desarrollo de una herramienta de recolección de señal UWB desde un dispositivo móvil y la creación de una herramienta de etiquetado de posición en tiempo real desde dispositivo móvil para facilitar esta tarea. Estas herramientas, además, nos permitirán conseguir otro de los objetivos marcados: la recolección de datos UWB en un entorno real, que a su vez esto nos proporcionará la pequeña base de datos que utilizaremos como conjunto de datos de entrada en el modelo de Deep Learning para estimar la posición del usuario.

Durante la fase de entrenamiento en las técnicas de fingerprinting, por "etiquetado" se entiende el proceso de asociar una posición específica a una determinada muestra de “huella digital”. En nuestra propuesta, este proceso se realiza mediante el uso de una aplicación móvil que el usuario portará durante esta fase y en la cual se permitirá asociar las señales UWB recibidas por el dispositivo en ese instante de tiempo, identificado por una marca temporal (timestamp), a la localización

precisa del usuario dentro del entorno sensorizado. Estos datos se acumulan durante pequeños intervalos de tiempo en el propio dispositivo móvil, hasta que cada cierto tiempo se envían por WiFi, mediante el protocolo HTTP, a un servidor desarrollado en Python que se encargará de procesar y almacenar permanentemente esta información en ficheros de texto.

Una vez finalizada la fase de etiquetado, las muestras capturadas se utilizan para entrenar un modelo de aprendizaje automático que permita reconocer y clasificar automáticamente nuevas "huellas digitales", que en este contexto se corresponden con la localización del usuario en el entorno sensorizado. Estudios previos han demostrado que los datos de entrenamiento utilizados deben ser diversos y lo suficientemente amplios como para garantizar que el modelo sea sólido.

Como ya se ha mencionado anteriormente, el dispositivo móvil que vamos a utilizar debe cumplir los requisitos de compatibilidad con la tecnología inalámbrica UWB. Hoy en día, existen muchos dispositivos en el mercado que disponen de esta funcionalidad, tanto basados en Android como en iOS. Sin embargo, como se ha comentado en el apartado anterior, para satisfacer los requisitos del framework que acompaña a las balizas UWB que hemos seleccionado para el despliegue, se ha escogido como dispositivo móvil un **iPhone 13 Pro** del que se disponía. Este dispositivo cuenta con una serie de características de conectividad que le permiten conectarse a distintas redes y dispositivos, lo que lo convierte en una buena opción para nuestro propósito. Algunas de las principales características de conectividad del iPhone 13 Pro son:

- Conectividad 5G: El iPhone 13 Pro es compatible con las bandas 5G sub-6GHz y mmWave, lo que permite velocidades de carga y descarga más rápidas y una mejor conectividad en zonas con cobertura 5G.

- Wi-Fi 6: El último estándar Wi-Fi, que ofrece velocidades más rápidas, un mejor rendimiento en entornos con mucha gente y una mayor eficiencia energética.

- Bluetooth 5.0: Este dispositivo móvil es compatible con la última versión de Bluetooth, que permite una transferencia de datos más rápida, un mayor alcance y una mayor eficiencia energética.

- Chip U1 UWB: Este chip es una de las últimas incorporaciones a la familia de dispositivos de Apple, e incorpora reconocimiento de señales de banda ultraancha.

Como se puede deducir, la característica más importante para nuestro objetivo es la presencia del chip U1. Este chip se encuentra disponible en todos los modelos de iPhone desde el iPhone 11 y emplea tecnología de banda ultraancha para el reconocimiento espacial, lo que permite al dispositivo móvil localizar con precisión otros dispositivos compatibles con esta tecnología. El chip habilita varias funciones, tales como sugerencias direccionales en AirDrop, mejora de la precisión de la búsqueda de dispositivos compatibles, como los AirPods Pro o los AirTags[7], habilita nuevas experiencias basadas en proximidad (por ejemplo, permite abrir un coche cerrado o abrir una cerradura inteligente), permite localizar con más precisión a dispositivos compatibles dentro de un edificio mediante la combinación del chip U1 y otros sensores, y también es utilizado para mejorar la precisión y el realismo de experiencias de realidad aumentada.

Figura 17. ***Airtag es uno de los primeros dispositivos comerciales que utilizan la tecnología UWB para localizar objetos interiores (Kateliev, 2021)***

[7] Apple AirTag es un pequeño dispositivo con forma de moneda que permite a los usuarios rastrear y localizar objetos como llaves, carteras y bolsos, utilizando la red "Find My" de Apple y la aplicación "Find My".

En este dispositivo móvil se ejecutará la herramienta de recolección de datos emitidos por las balizas UWB. Para ello se ha desarrollado una sencilla aplicación para iOS cuyos requisitos funcionales son los siguientes:

- Escaneo y detección de las balizas UWB desplegadas en el entorno.
- Conexión con las balizas y reconocimiento constante y en tiempo real de los datos emitidos por éstas.
- Almacenamiento temporal de los datos emitidos, junto con otra información adicional de interés como el identificador de la baliza que lo ha generado y del dispositivo móvil que lo ha recogido.
- Envío asíncrono y periódico de los datos almacenados de forma local a un servidor mediante HTTP.
- Comunicación con el servidor, mediante mensajes HTTP, para gestionar las sesiones de captura de datos para el etiquetado del modelo DL.
- Desarrollo de una interfaz de usuario sencilla que permita fácilmente controlar el estado de la captura de datos y generar etiquetas para la posición del usuario.

Para el desarrollo de la aplicación para iOS se ha utilizado Xcode, instalado en su última versión 14.2. **Xcode** es un entorno de desarrollo integrado (IDE) desarrollado por Apple para el desarrollo de software de macOS e iOS. Xcode incluye un editor de código fuente, un editor de interfaz gráfica de usuario y muchas otras herramientas para desarrollar software para las plataformas de Apple.

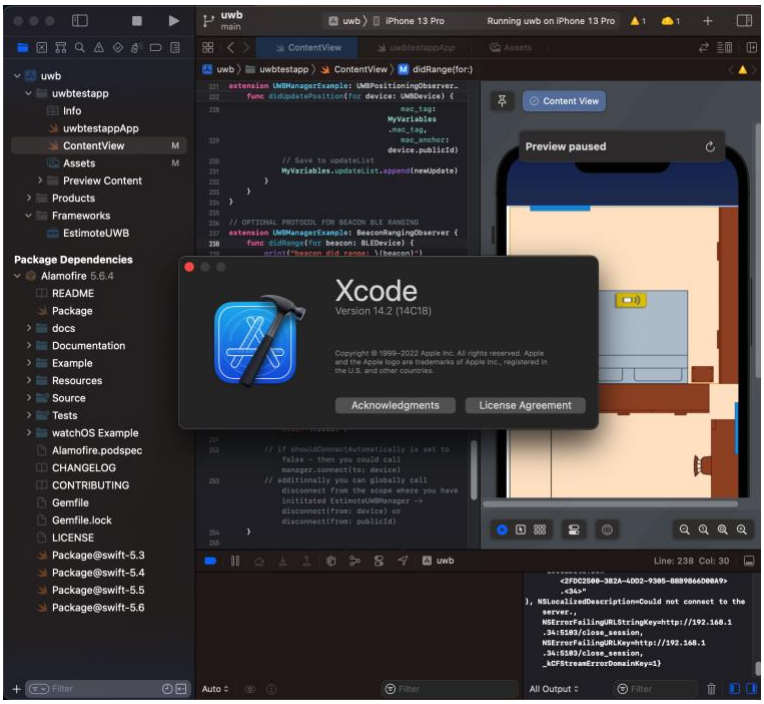


Figura 18. *Interfaz gráfica del IDE Xcode empleado para el desarrollo de la aplicación iOS.*

Debido a que este IDE sólo se encuentra disponible para los sistemas operativos de Apple, para el desarrollo de este software se ha recurrido al uso de un ordenador portátil de esta marca, cuyas características técnicas son las siguientes:

Modelo	MacBook Pro M1 13 pulgadas
Año de lanzamiento	2020
CPU	Apple M1
Memoria	8 GB
Sistema Operativo	MacOS Ventura 13.0.1

Tabla 3. *Especificaciones del dispositivo de desarrollo en iOS*

El lenguaje de programación empleado para el desarrollo ha sido **Swift**. Swift es un lenguaje de programación compilado de propósito general desarrollado por Apple Inc. para sus plataformas y Linux. Está diseñado para funcionar con los frameworks Cocoa y Cocoa Touch de Apple, y con el sistema operativo Linux. Swift pretende ser más resistente y legible que Objective-C, el antecesor lenguaje que se utilizaba para el desarrollo de aplicaciones MacOS y iOS antes de la aparición de Swift. Swift es un lenguaje de programación relativamente nuevo, surgido en 2014, pero se ha ganado popularidad rápidamente entre los desarrolladores de iOS y macOS, así como en otras

plataformas. Es un lenguaje de programación potente, de alto rendimiento y fácil de aprender.

Para comenzar el desarrollo se generó un nuevo proyecto en Xcode, seleccionando como sistema operativo objetivo "*iOS*" y como plantilla la denominada "*Single View App*" ya que para nuestro propósito sólo necesitaremos una vista principal donde mostrar el mapa del entorno que se va a sensorizar junto con los controles que nos permitan iniciar y detener la captura de datos. En la siguiente figura se muestra la interfaz gráfica de la aplicación:

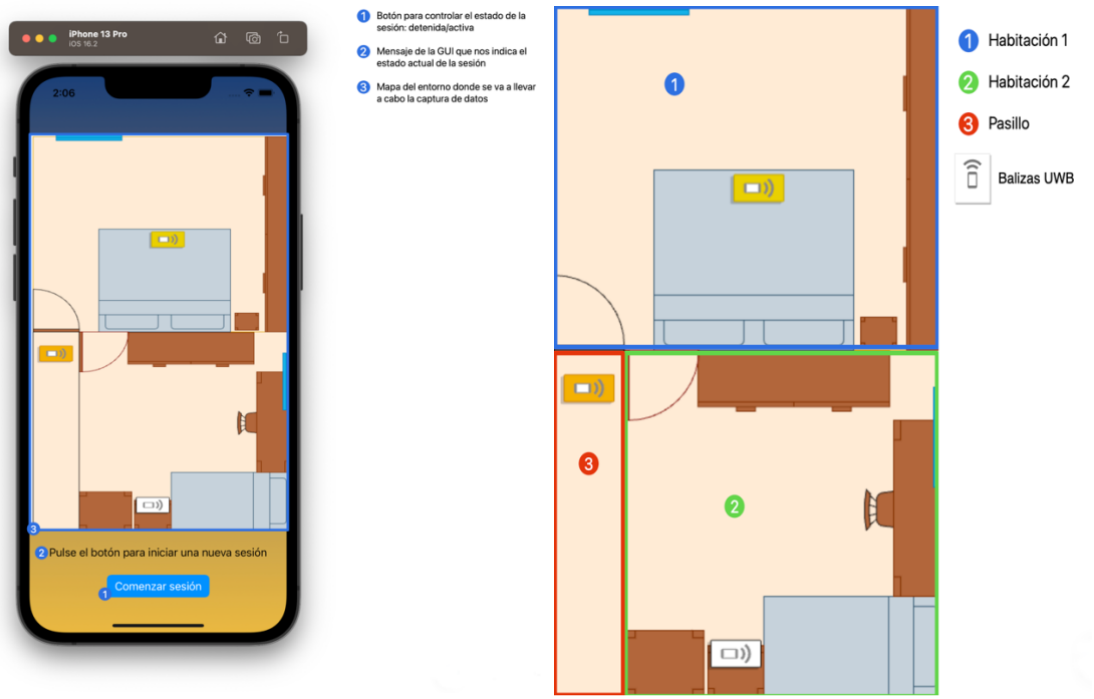


Figura 19. *GUI de la aplicación iOS (izquierda) y esquema del domicilio sensorizado (derecha).*

Como se puede observar en la figura anterior, en esta interfaz podemos diferenciar tres elementos:

- El **mapa del entorno**, que ocupa la mayor parte de la vista de la interfaz, y le muestra al usuario cual es la ubicación de las balizas.
- El **mensaje de estado de la sesión**, que nos indica en cuál de los estados se encuentra la aplicación:

a) Detenida: durante este estado no se almacenan los datos de las balizas ni se registran las pulsaciones del usuario sobre el mapa del entorno.

b) Activa: en este estado los datos recolectados desde las balizas son almacenados localmente en el dispositivo, así como las etiquetas registradas por el usuario mediante sus pulsaciones sobre el mapa del entorno. De forma continua mientras la aplicación se encuentra en este estado y con una frecuencia establecida en 10 segundos, se activa un disparador que envía los datos recopilados (a modo de procesamiento por lotes) al servidor REST para su procesamiento y persistencia. Esto se hace así por temas de eficiencia en la ejecución de la aplicación, ya que el refresco de las señales UWB emitidas por las balizas es muy alto, por lo que si se enviara un dato cada vez que se recibe desde la baliza se produciría una saturación tanto en la aplicación del dispositivo móvil como en el propio servidor.

- El **botón para controlar el estado de la sesión**, que permite tanto iniciar una nueva sesión como detener una sesión activa y realizar la correspondiente comunicación con el servidor.

La pulsación del usuario sobre el mapa durante una sesión activa, para llevar a cabo el etiquetado, registrará las coordenadas sobre el plano 2D relativas a la esquina superior izquierda de la imagen, que actúa como punto de origen (0,0), y mostrará un punto rojo en la interfaz de la aplicación, para aportar la retroalimentación necesaria al usuario, de manera que reconozca si la pulsación ha sido registrada en el lugar correcto de la imagen y así también poder ajustar al máximo la precisión de la localización de las etiquetas según su punto de vista.

Figura 20. ***Interfaz de usuario de la aplicación iOS durante una sesión activa***

Para conseguir la comunicación del dispositivo móvil con las balizas UWB se ha utilizado el **framework de Estimote** que se ofrece con el kit de desarrollo de las balizas UWB seleccionadas para la propuesta. Este SDK para iOS es un kit de desarrollo de software que permite a los desarrolladores crear aplicaciones para dispositivos iOS que pueden comunicarse y localizar balizas Estimote Ultra-Wideband (UWB). El SDK proporciona un conjunto de APIs que permiten a los desarrolladores buscar balizas cercanas, determinar la distancia a una baliza mediante distintos indicadores de posicionamiento, y seguir el movimiento de una baliza en el tiempo. Además, el SDK incluye ejemplos de código y documentación, disponibles en su repositorio oficial de GitHub[8], para el uso con otras balizas del fabricante Estimote que utilizan otras tecnologías de conexión inalámbrica, como BLE, para ayudar a los desarrolladores a empezar. Sin embargo, al ser las balizas UWB una novedad dentro de los modelos de Estimote, la documentación del SDK para dichas balizas no se encuentra disponible todavía, por lo cual hoy en día es más complicado desarrollar aplicaciones mediante este framework, pero Estimote promete que estará disponible en un futuro próximo, al igual que ocurre con el resto de sus dispositivos.

Algunas de las características de este SDK y de los dispositivos Estimote son:

[8] https://github.com/estimote

- Cálculo de distancias: APIs para medir la distancia entre el dispositivo y la baliza, permitiendo a la app determinar la proximidad del dispositivo a la baliza. Para ello usan una medida interna de filtrado de RSSI que permite realizar la estimación. El valor de la distancia al objeto se mide en metros, y en el contexto de este trabajo, destacamos que no es preciso por la orientación, bloqueo y efecto pantalla de objetos. El modelado o rectificación de la posición del usuario en base a esta medida con ruido será el objeto del modelo de localización de DL.

- Posicionamiento: APIs para determinar la ubicación del dispositivo dentro de un espacio utilizando múltiples balizas, que pueden utilizarse para crear servicios basados en la localización en interiores.

- Gestión de balizas: APIs para buscar balizas cercanas, configurar y actualizar los ajustes de una baliza, y seguir el movimiento de una baliza a lo largo del tiempo.

- Soporte de múltiples balizas simultáneas: El SDK puede detectar y rastrear múltiples balizas al mismo tiempo, por lo tanto, es ideal para la idea de nuestra propuesta.

- Baja latencia y alta precisión: El SDK aprovecha la tecnología UWB que permite proporcionar una alta precisión y baja latencia en el seguimiento de la ubicación.

- Fácil integración: El SDK es fácil de integrar con otros frameworks y librerías, lo que simplifica la incorporación de la funcionalidad UWB a las apps existentes.

Para integrar el Estimote UWB SDK en el proyecto de Xcode en primer lugar se descargó el código del SDK desde el sitio web aportado por Estimote. Una vez descargado el SDK, se puede añadir fácilmente al proyecto Xcode arrastrando el archivo del framework del SDK a la sección "*Frameworks, Libraries, and Embedded Content*" del proyecto dentro de la pestaña "*General*". Una vez que se ha añadido el SDK al proyecto, se puede importar al código de la aplicación mediante la directiva

import seguido del nombre del framework (*EstimoteUWB*) en la parte superior del archivo principal del proyecto, denominado “ContentView”.

Antes de poder utilizar el SDK, se tuvo que configurar el proyecto con las capacidades adecuadas. En la pestaña "*Signing & Capabilities*", se activó "*Location updates*" y “*Uses Bluetooth LE accessories*” dentro del grupo "*Background Modes*”.

La función "*Location updates*" de un proyecto en Xcode permite a la aplicación acceder a los datos de ubicación del dispositivo, que luego puede utilizar para proporcionar funciones basadas en la ubicación, como la cartografía y el seguimiento de la ubicación. Esta función debe estar activada para que la aplicación pueda acceder a los datos de localización y, además, debe solicitar permiso al usuario para acceder a su ubicación. De forma análoga, la función " *Uses Bluetooth LE accessories*" en un proyecto Xcode permite a la aplicación comunicarse con dispositivos Bluetooth Low Energy (BLE), como pulseras de fitness, relojes inteligentes y otros accesorios que utilizan BLE para transmitir datos. Esta función debe estar activada para que la aplicación pueda comunicarse con dispositivos BLE, y también debe solicitar permiso al usuario para acceder a las funciones Bluetooth de su dispositivo. El framework UWB hace uso de estas dos funciones, por lo tanto, deben activarse para el correcto funcionamiento de las funcionalidades que nos proporciona.

Una vez instalado, importado y configurado el framework UWB se implementó en la aplicación de la siguiente forma: Para empezar, se puede inicializar el SDK creando una nueva instancia de la clase EstimoteUWBManager y estableciendo su delegado. Una vez inicializado el SDK, se puede empezar a buscar balizas llamando al método startScanning() de la instancia EstimoteUWBManager, como se puede observar en la siguiente figura:

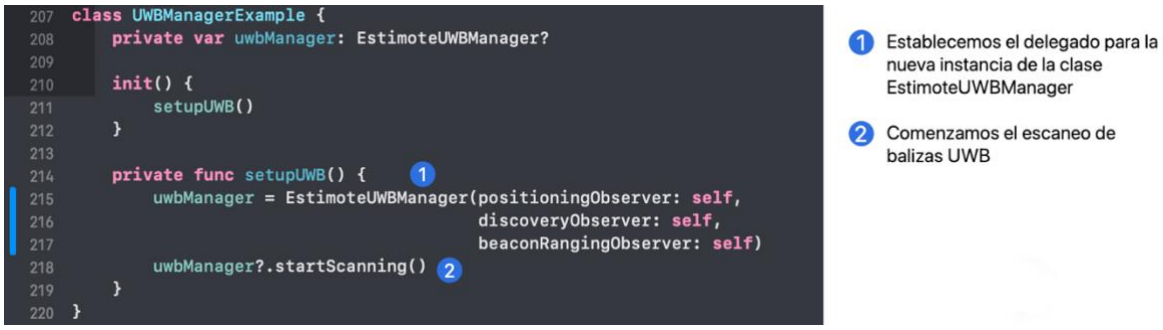


Figura 21. *Se proporciona el delegado en la inicialización del EstimoteUWBManager*

Si en algún momento queremos detener la comunicación con las balizas, se puede detener la búsqueda llamando al método stopScanning() de la instancia EstimoteUWBManager.

Mediante el protocolo UWBDiscoveryObserver se puede controlar el comportamiento de la aplicación en base a los distintos eventos que pueden ocurrir en cuanto al estado de la conexión con las balizas UWB. Se puede establecer que el framework intente establecer conexión automáticamente con las balizas UWB que vaya descubriendo, o por el contrario en redes saturadas con muchos de estos dispositivos podríamos controlar con cuales de estas balizas queremos establecer conexión y leer sus datos. Esto último se puede llevar a cabo filtrando las balizas por su identificador único denominado publicId.

También se puede controlar el comportamiento de la aplicación cuando el framework descubre una nueva baliza, mediante el método didDiscover(), así como cuando se establece una conexión de forma satisfactoria con una baliza descubierta previamente, mediante didConnect(). Del mismo modo ocurre cuando se produce una desconexión de una baliza, mediante el método didDisconnect(). Esto es algo que puede ocurrir, por ejemplo, por interferencias en la señal de radio. En tal caso el framework nos proporciona información sobre el motivo que ha causado esta desconexión y nos permitiría actuar en consecuencia. En las pruebas realizadas en esta propuesta no se han detectado apenas desconexiones de las balizas, por lo cual no se ha implementado ninguna medida en estos casos.

Otro de los eventos de interés es cuando el framework intenta establecer conexión con una baliza, pero no logra conseguir conectarse por algún motivo. En este caso, el método didFailConnect() nos informaría de este hecho y nos proporcionaría también información sobre la causa de la imposibilidad de conexión.

En la siguiente figura se muestra la implementación de estos métodos en el código de la aplicación desarrollada:

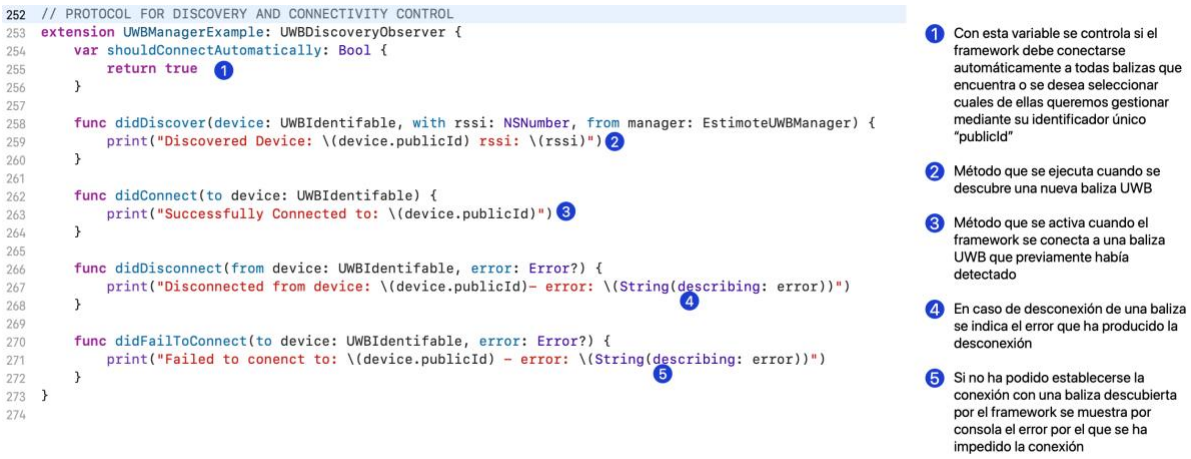


Figura 22. *Protocolo de búsqueda y conexión con las balizas UWB del SDK.*

Como se ha explicado, la aplicación al comenzar su ejecución iniciará automáticamente el escaneo de las balizas UWB disponibles en el entorno y en este caso, si tenemos conectado el dispositivo móvil al equipo de desarrollo, mostrará mensajes por la consola de Xcode similares a los que se muestran en la siguiente figura:

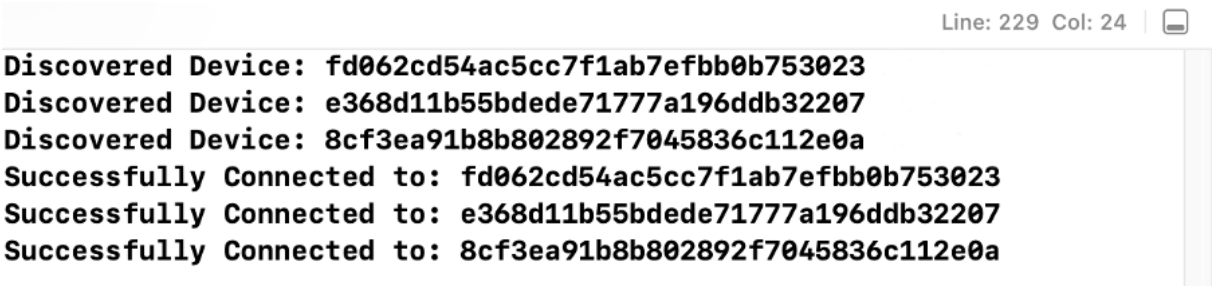


Figura 23. *Mensajes de descubrimiento y conexión con las balizas UWB por parte del dispositivo móvil.*

Una vez que se han detectado y se ha establecido conexión con las balizas, mediante el protocolo UWBPositioningObserver podremos gestionar el método que nos ofrecerá las actualizaciones de los datos enviados por cada una de las balizas a las que nos hemos conectado previamente. En la Figura 24 se muestra la implementación de la función didUpdatePosition(), mediante la cual se puede acceder al objeto de tipo *UWBDevice* que nos proporciona el SDK de Estimote, el cual dispone de diversa información acerca de la baliza UWB que envía la señal. Para el propósito de esta propuesta, de esta información únicamente se necesita obtener:

i) el **identificador de la baliza UWB** que envía esta información, lo cual se recibe a través del parámetro *publicId* de dicho objeto, y
ii) la **distancia calculada entre el dispositivo móvil y la baliza UWB**, la cual se encuentra definida en el parámetro *distance*.

Además de esto también se necesita añadir el identificador del dispositivo móvil que ha recogido esta actualización de datos de la baliza (*mac_tag*) y la marca temporal del instante de tiempo (*timestamp*) en el cual se ha leído y almacenado esta muestra de datos.

Figura 24. ***Función didUpdatePosition() que permite obtener los datos UWB desde las balizas desplegadas en el entorno.***

Los datos recopilados por el dispositivo móvil mediante la función didUpdatePosition() se envían cada 10 segundos en el servidor para ser almacenados de forma persistente. Para implementar la repetición de esta tarea se ha empleado la clase Timer de Swift. Timer es un objeto que forma parte de una API de alto nivel del lenguaje Swift que permite ejecutar un bloque de código a una hora determinada o a intervalos regulares y se utiliza habitualmente en el desarrollo de iOS para tareas como la programación de animaciones, la repetición de tareas o el seguimiento del tiempo transcurrido. En nuestro caso, el siguiente código crea un temporizador que ejecuta una tarea de forma cíclica en intervalos de 10 segundos. La tarea se ejecutará en el hilo principal y en el modo común.

```
let timer = Timer.publish(every: 10, on: .main, in: .common).autoconnect()
```

Después, para implementar la acción que se desea ejecutar cada vez que se activa el objeto *timer*, se emplea el modificador onReceive, el cual permite observar cambios en un temporizador o en otro objeto del tipo ObservableObject. En este caso, el modificador estará observando cambios en el objeto *timer*. Cuando se reciba un nuevo valor del temporizador, se ejecutará el cierre pasado como argumento al modificador onReceive, proporcionando acceso al objeto *timer*, aunque en nuestro caso la información que nos proporciona dicho objeto no se utiliza para nada pues simplemente se realiza la llamada a la función sendUpdate() para enviar al servidor en modo batch todas las muestras de datos UWB recopilados hasta el momento:

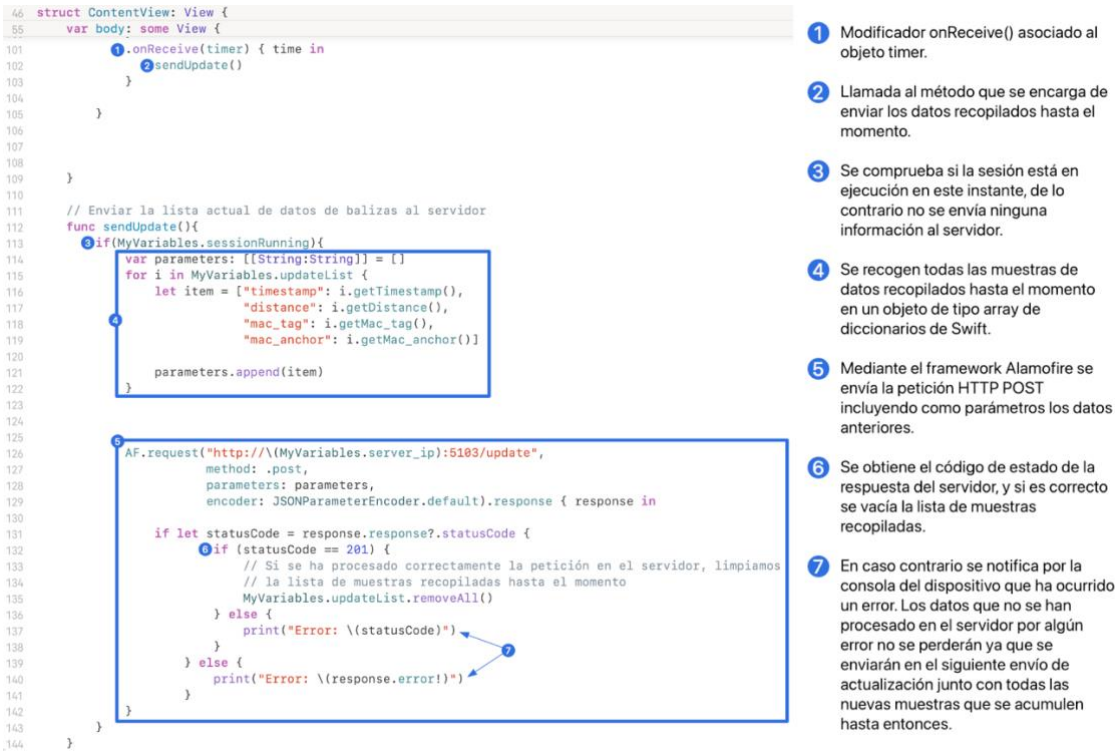


Figura 25. ***Modificador onReceive() y método que gestiona el envío de datos al servidor.***

Para llevar a cabo la comunicación entre la aplicación móvil y dicho servidor se ha optado por utilizar el protocolo HTTP. La implementación de los servicios o endpoints de la API REST de este servidor se detallará en la sección 3.4. A través de HTTP podemos realizar solicitudes a un servidor para obtener, editar, publicar y eliminar datos. Para diferenciar estas acciones existen los denominados métodos de HTTP. Estos métodos definen el tipo de acción que debe realizarse sobre el recurso especificado. La elección del método HTTP depende de la naturaleza de la solicitud y del resultado deseado. Los principales métodos son:

1. GET: El método GET se utiliza para recuperar datos de un servidor. Es el método HTTP más utilizado y sirve para solicitar un recurso o información al servidor.

2. POST: El método POST se utiliza para enviar datos al servidor. Normalmente se utiliza para enviar un formulario o enviar datos al servidor para su procesamiento.

3. PUT: El método PUT se utiliza para actualizar un recurso en el servidor. Se utiliza para sustituir un recurso existente por uno nuevo.

4. DELETE: El método DELETE se utiliza para eliminar un recurso del servidor.

5. HEAD: El método HEAD se utiliza para recuperar las cabeceras de un recurso sin el contenido del cuerpo. Se utiliza para obtener información sobre un recurso sin recuperar su contenido.

6. PATCH: El método PATCH se utiliza para actualizar un recurso en el servidor enviando sólo los cambios, en lugar del recurso completo.

7. OPTIONS: El método OPTIONS se utiliza para recuperar los métodos y protocolos soportados de un recurso. Se utiliza para comprobar las capacidades de un servidor.

En nuestra propuesta principalmente se hace uso del método POST para enviar datos. En el caso del método sendUpdate(), que se puede observar en la Figura 25, se utiliza este método para enviar el array de diccionarios en formato JSON. Para realizar las peticiones HTTP desde la aplicación para iOS se ha utilizado el framework **Alamofire**[9], un popular framework de código abierto para iOS, macOS y watchOS que facilita la realización de solicitudes de red en Swift. Algunas características clave de Alamofire incluyen:

- Servicios web mediante HTTP: proporciona una API simple y concisa para realizar solicitudes HTTP, incluyendo distintos métodos como GET, POST, PUT, DELETE, etc.

- Gestión de solicitudes y respuestas: Alamofire, que se detalla a continuación, será el framework para serializar y deserializar automáticamente los datos de respuesta en tipos de datos comunes como JSON y XML.

[9] https://github.com/Alamofire/Alamofire

- Autenticación: soporta autenticación básica HTTP, OAuth y otros métodos de autenticación.
- Carga de datos: facilita la carga de datos como imágenes y vídeos.
- Descarga de datos: puede descargar datos en segundo plano y admite la reanudación de descargas.
- Depuración: incluye potentes herramientas de depuración que permiten a los desarrolladores inspeccionar las solicitudes y respuestas de red.
- Alcance de la red: puede detectar el estado de disponibilidad de la red, comprobar si el dispositivo está conectado a Wi-Fi o celular, etc.

En general, Alamofire es un framework bien documentado, bien mantenido y ampliamente utilizado para realizar peticiones de red en Swift. En el contexto de esta propuesta, este framework nos facilitará el envío de solicitudes a los endpoints de la API REST del servidor y la gestión de las respuestas que éste devuelva. Para importar este framework al proyecto de Xcode hemos utilizado el gestor de paquetes Swift[10], una herramienta para gestionar la distribución de código Swift diseñado para integrarse con el sistema de compilación de Swift y automatizar el proceso de descarga, compilación y vinculación de dependencias. Para ello nos dirigimos al gestor de paquetes de Swift dentro de Xcode (ver Figura 26), y agregamos la nueva dependencia.

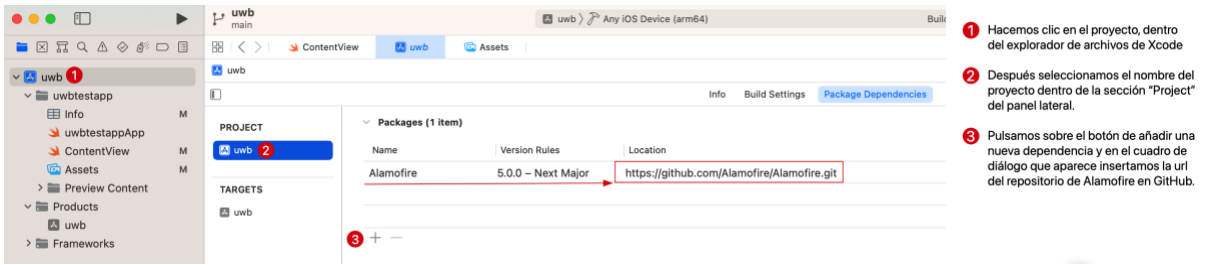


Figura 26. *Proceso de añadir dependencias mediante el gestor de paquetes de Swift*

Como se explica en la figura anterior, el gestor de paquetes nos pedirá la dirección URL del repositorio de GitHub del paquete que deseamos añadir, en este

[10] https://www.swift.org/package-manager/

caso Alamofire. Opcionalmente también nos dará opción a instalar una versión distinta a la última estable disponible, en el caso de que se necesitase recurrir concretamente a una versión más antigua. Una vez hecho esto, en nuestro código de la aplicación para iOS podremos implementar peticiones HTTP siguiendo la sintaxis de Alamofire, como se puede ver en el código que se muestra en la Figura 25:

```
import Alamofire

AF.request("http://\(MyVariables.server_ip):5103/update",
        method: .post,
        parameters: parameters,
        encoder: JSONParameterEncoder.default).response { response in

if let statusCode = response.response?.statusCode {
        if (statusCode == 201) {
            // Si se ha procesado correctamente la petición en el servidor, limpiamos
            // la lista de muestras recopiladas hasta el momento
            MyVariables.updateList.removeAll()
        } else {
            print("Error: \(statusCode)")
        }
    } else {
        print("Error: \(response.error!)")
    }
}
```

Como se desprende del ejemplo anterior, mediante AF.request() se envía una solicitud de tipo POST a la URL especificada, y el cierre response se utiliza para gestionar la respuesta del servidor. La respuesta incluye información como los objetos de solicitud y respuesta, el error (si lo hay) y los datos de respuesta (si los hay). En este caso comprobamos el código de estado de la respuesta del servidor, y en el caso de que haya sido satisfactoria procedemos a limpiar la lista de muestras recopiladas hasta el momento para no duplicar información en el siguiente envío de datos. En la próxima sección se detallarán todos los servicios o endpoints que son demandados desde el dispositivo, pero la forma de realizar las peticiones sigue siempre el esquema indicado en el ejemplo de código anterior.

La creación de una herramienta que permitirá recopilar constantemente los datos transmitidos por UWB en un entorno real, así como proporcionar la capacidad de realizar el etiquetado de la posición del usuario en tiempo real desde un dispositivo móvil era uno de los objetivos marcados para este Trabajo de Fin de Máster que se ha cumplido mediante el desarrollo de la aplicación para iOS y la implementación del SDK UWB.

3.4 Servidor REST

En la sección anterior hemos visto como se ha creado una herramienta para recopilar datos UWB y las etiquetas de posición del usuario durante las distintas sesiones. No sería abordable ni práctico almacenar todas las muestras en el propio dispositivo móvil, por lo tanto, se ha optado por utilizar una **API REST** alojada en un servidor externo al dispositivo móvil, que podría ser un computador personal, un ordenador de placa simple (SBC) tipo Raspberry Pi, un servicio cloud… En esta API se implementarán distintos servicios que nos permitirán, por un lado, manejar el estado de las sesiones de toma de muestras y por otro obtener esta información desde la aplicación móvil para catalogarla y almacenarla en archivos de texto.

Una **API REST** (Representational State Transfer) es una arquitectura de software basada en web y un método para crear servicios web que utilizan peticiones HTTP mediante los distintos métodos que se explicaron en la sección 3.3. Es un estándar para crear un conjunto de reglas y protocolos que definen cómo interactúan entre sí dos sistemas de software de forma escalable y mantenible para aplicaciones web y dispositivos móviles.

El lenguaje de programación empleado para el desarrollo de esta API ha sido **Python** (en su versión 3.9.13). Python es un lenguaje de programación interpretado de alto nivel que se utiliza para una amplia gama de tareas, como el desarrollo web, la informática científica y la inteligencia artificial, entre otras. Fue creado por Guido van Rossum y publicado por primera vez en 1991. Existen distintas y muy variadas

razones por las que Python es el lenguaje de programación más utilizado hoy en día según los datos del índice TIOBE[11]:

i) Fácil de aprender y utilizar, pues tiene una sintaxis sencilla e intuitiva que facilita el aprendizaje a los nuevos programadores.

ii) Versátil, ya que se utiliza en una amplia gama de aplicaciones, desde el desarrollo web y la computación científica hasta el análisis de datos y el aprendizaje automático.

iii) Amplia comunidad y ecosistema, contando con una comunidad de usuarios grande y activa que contribuye a su desarrollo y proporciona una gran cantidad de recursos y apoyo.

iv) Multitud de bibliotecas, lo que significa que cuenta con una amplia colección de paquetes que los desarrolladores pueden añadir a sus proyectos para resolver tareas específicas.

v) Adopción industrial, es decir, Python es ampliamente utilizado en la industria, y muchas empresas multinacionales tecnológicas confían en él para diversos proyectos.

[11] El índice TIOBE proporciona una instantánea de la popularidad actual de los lenguajes de programación y es ampliamente utilizado como referencia por desarrolladores de software, investigadores y expertos del sector.

Jan 2023	Jan 2022	Change	Programming Language	Ratings	Change
1	1		Python	16.36%	+2.78%
2	2		C	16.26%	+3.82%
3	4	↑	C++	12.91%	+4.62%
4	3	↓	Java	12.21%	+1.55%
5	5		C#	5.73%	+0.05%
6	6		Visual Basic	4.64%	-0.10%
7	7		JavaScript	2.87%	+0.78%
8	9	↑	SQL	2.50%	+0.70%
9	8	↓	Assembly language	1.60%	-0.25%
10	11	↑	PHP	1.39%	-0.00%

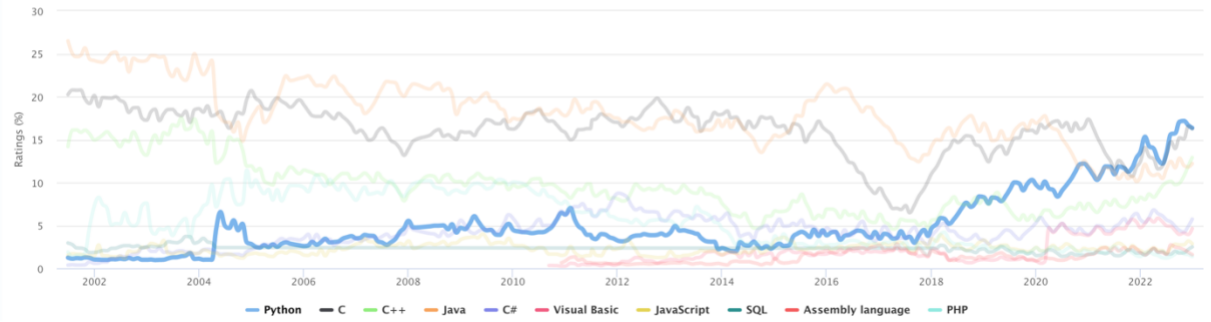


Figura 27. *Posición actual de Python como primer lenguaje más utilizado (arriba) y evolución de su uso en las última décadas (abajo)* (*TIOBE Index*, n.d.)

Para trabajar correctamente con Python, sobre todo si se encuentran en desarrollo distintos proyectos al mismo tiempo, es utilizar lo que se denominan **entornos virtuales**. Los entornos virtuales en Python son entornos aislados para proyectos Python, que permiten gestionar dependencias y paquetes de forma independiente para cada proyecto. Ayudan a prevenir conflictos entre paquetes requeridos por diferentes proyectos, creando un entorno separado para cada proyecto con su propio conjunto de dependencias. Se pueden crear, activar y gestionar entornos virtuales utilizando herramientas como *venv*, *virtualenv* o *Anaconda Navigator*.

Para nuestra propuesta se ha optado por utilizar **Anaconda Navigator** (ver Figura 28), una interfaz gráfica de usuario para la gestión de paquetes y virtuales de Python. Esta herramienta nos permite a los usuarios crear y gestionar entornos virtuales para sus proyectos Python, así como gestionar paquetes y dependencias para cada entorno.

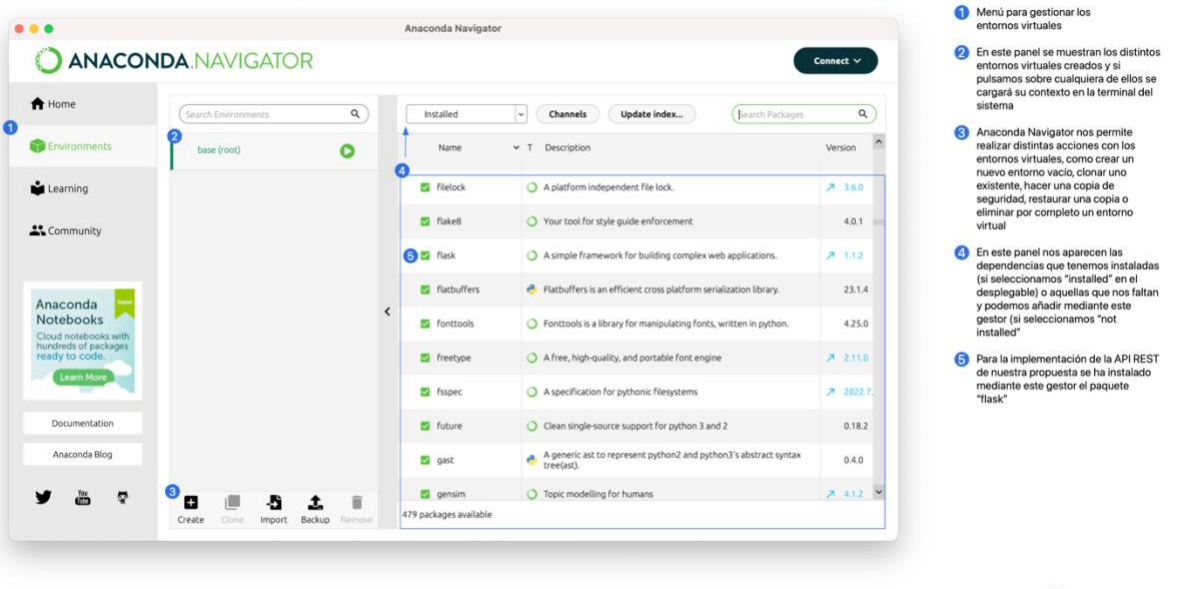


Figura 28. ***Interfaz de Anaconda Navigator.***

Como se puede comprobar en la figura anterior, para nuestro proyecto Python vamos a necesitar un paquete específico denominado *Flask*. **Flask**[12] es un popular framework microweb de código abierto escrito en Python. Es ligero, flexible y no impone una estructura de directorios específica ni requiere herramientas o bibliotecas concretas adicionales. Flask proporciona una plataforma sencilla y fácilmente extensible para crear APIs REST, y permite a los desarrolladores centrarse en escribir el código de su aplicación en lugar de preocuparse por los detalles de bajo nivel. Incluye funciones como el enrutamiento de URL, la gestión de solicitudes y los motores de plantillas, y admite extensiones de terceros para tareas comunes como la integración de bases de datos, la autenticación y la gestión de formularios. Flask puede utilizarse para crear aplicaciones pequeñas y medianas, y es adecuado tanto para la creación rápida de prototipos como para la implantación en producción, por tanto, es una forma ideal de implementar el objetivo para la propuesta de este trabajo.

Para trabajar con *Flask* en nuestro código Python, en primer lugar, debemos importar este paquete al entorno virtual, como ya se ha explicado anteriormente. Después de esto ya se podría importar los módulos necesarios para utilizar *Flask* mediante las directivas *from* e *import*:

[12] https://flask.palletsprojects.com/en/2.2.x/

```
from flask import Flask, jsonify, request
```

El siguiente paso consiste en crear una instancia de la clase *Flask*, sobre la cual se implementarán los distintos servicios. Dicha instancia se iniciará al comenzar la ejecución del hilo principal del script Python estableciendo los parámetros *host* y *port*:

```
app = Flask("restAPI")
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5103)
```

El argumento *host* se establece en '0.0.0.0', lo que significa que el servidor escuchará en todas las interfaces de red, permitiendo a los clientes externos acceder a la API. Por otro lado, el argumento *port* se establece en 5103, que especifica el número de puerto en el que escuchará el servidor.

Una vez implementada la base de la aplicación *Flask* únicamente se necesita definir los endpoints o servicios de la API mediante el modificador @app.route(). A continuación, se muestran los códigos de todos los endpoints que se han implementado en esta API.

En el endpoint */update* se reciben y procesan los datos de las muestras UWB recopiladas en la aplicación del dispositivo móvil (ver Figura 24), incluyendo:

- La marca temporal del instante de tiempo en el cual se recibió dicha muestra (*timestamp).*
- La distancia existente entre el dispositivo móvil y la baliza UWB que ha generado esta muestra (*distance*).
- El identificador del dispositivo móvil que ha recopilado esta muestra (mac_*tag*).
- El identificador de la baliza UWB que ha generado dicha muestra *(*anchor_tag*).*

Estos valores envían por lotes cada 10 segundos desde la aplicación iOS y se almacenan como una fila de datos delimitados por espacios (tipo CSV) por cada

muestra recibida, en un archivo que recibe el nombre del identificador de la sesión actual, que a su vez coincide con la marca temporal en la que se inició la sesión (ver Figura 30). Este archivo siempre se abre en el modo "*append*", para evitar sobrescribir datos anteriormente guardados en el mismo y los datos aunque se envíen en batch incluyen la huella de tiempo del dispositivo móvil cuando los recolectó evitando desplazamientos de tiempos por retardos en el envío.

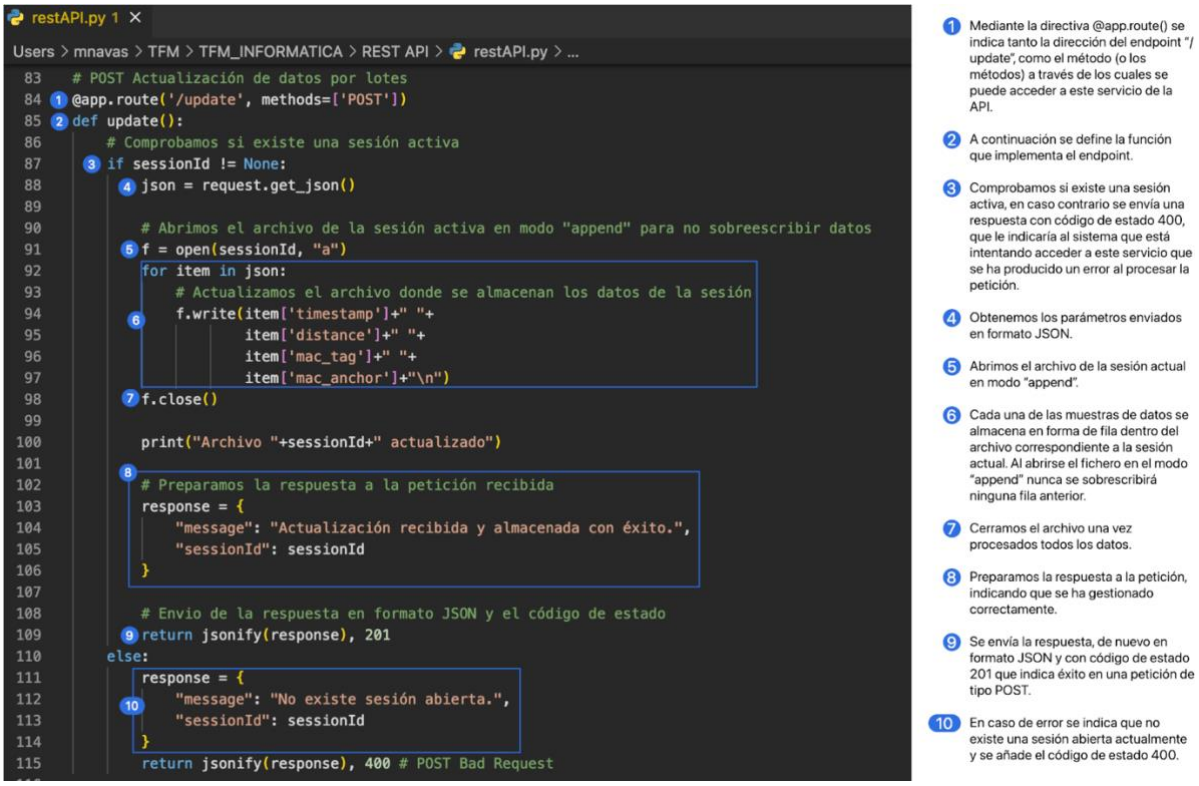


Figura 29. ***Endpoint /update de la API REST***

Como se puede comprobar en la figura anterior, una vez que se ha procesado la petición de forma correcta, se prepara la respuesta y se envía en formato JSON acompañada del código de estado 201, que indica el éxito del procesamiento de la petición.

De forma similar se implementa el endpoint */open_session*, que permitirá iniciar una nueva sesión de toma de datos (ver Figura 30). Dentro de su implementación se generará un nuevo identificador de la sesión, mediante la obtención de la marca temporal de ese instante de tiempo, y se almacenará en la variable global del script

denominada *sessionId*. Comprobar el valor de dicha variable en los distintos endpoints donde se necesite nos permitirá saber si existe una sesión abierta o no.

```python
@app.route('/open_session', methods=['GET'])  # 1
def open_session():
    global sessionId  # 2
    sessionId = str(datetime.datetime.now().timestamp())

    response = {  # 3
        "message": "Sesión iniciada con éxito.",
        "sessionId": sessionId
    }

    return jsonify(response), 200  # 4
```

1. Directiva @app.route() para establecer el endpoint /open_session mediante el método GET.
2. Tomamos la variable global del script denominada sessionId y almacenamos en ella la marca temporal del instante en el que se inicia la sesión. Esta sesión a partir de ahora se referencia por esta marca temporal y los archivos donde se almacenarán sus datos se identificarán con este dato.
3. Preparamos la respuesta a la petición.
4. Se envía la respuesta con el código de estado 200, que indica éxito en un método GET.

Figura 30. *Endpoint /open_session de la API REST*

Para realizar las peticiones desde la aplicación de iOS, como se ha explicado en el apartado anterior, se utilizará el framework Alamofire. Mediante dicho framework podremos generar y enviar las peticiones utilizando el método correspondiente, por ejemplo en la siguiente figura se detalla cómo se ha implementado la petición desde la aplicación de iOS para el endpoint */open_session*:

```
206
207     // Enviar petición al servidor para iniciar una nueva sesión
208     AF.request("http://\(MyVariables.server_ip):5103/open_session",
209                method: .get).response { response in
210         if let statusCode = response.response?.statusCode {
211             if (statusCode == 200) {
212                 // Si se ha procesado correctamente la petición en el servidor, cambiamos la GUI
213                 buttonTitle = "Terminar sesión"
214                 descriptionTitle = "Sesión en ejecución..."
215
216                 // Cambiamos la variable que nos informa del estado de la sesión
217                 MyVariables.sessionRunning = true
218             } else {
219                 print("Error: \(statusCode)")
220             }
221         } else {
222             print("Error: \(response.error!)")
223         }
224     }
```

1. Mediante Alamofire creamos la petición al endpoint /open_session, indicando tanto la dirección IP de la API REST como el puerto utilizado de escucha.
2. Indicamos que se desea realizar una petición mediante el método GET
3. Comprobamos que la API nos devuelve el código 200 indicando que se ha procesado correctamente la petición, por tanto se ha generado un nuevo identificador de sesión que se corresponde con su marca temporal.
4. Cambiamos la interfaz de usuario de la aplicación
5. Cambiamos el valor de la variable que indica a la lógica interna de la aplicación si existe una sesión en ejecución. Si tiene el valor true indica que sí hay una sesión en ejecución.

Figura 31. *Envío de la petición GET al endpoint /open_session desde la app iOS*

Otro de los endpoints que se ha implementado en la API es */close_sesion*, que en este caso nos permitirá dar por terminada una sesión en ejecución. Esto se realiza simplemente cambiando el valor de la variable sessionId a None, una constante especial en Python que representa la ausencia de un valor o un valor nulo. En este caso se ha empleado el método GET de HTTP para el endpoint, ya que no es necesario que desde el dispositivo se envíe ningún dato junto a esta petición.

```python
@app.route('/close_session', methods=['GET'])
def close_session():
    global sessionId
    sessionId = None

    response = {
            "message": "Sesión terminada con éxito.",
            "sessionId": sessionId
        }

    return jsonify(response), 200
```

1. Se cambia el valor de la variable global sessionId a None, indicando que no existe una sesión en ejecución a partir de este instante.
2. Se prepara la respuesta a la petición GET
3. Se envía la respuesta en formato JSON y código de estado 200.

Figura 32. *Endpoint /close_session de la API REST*

La implementación de la petición desde la aplicación para iOS a este endpoint se ha realizado de forma análoga a como se mostraba en la Figura 31 con el endpoint */open_session*. Cabe destacar que los cambios en la aplicación siempre se llevan a cabo únicamente cuando se recibe la respuesta por parte de la API, siempre y cuando el código de estado recibido indique que la petición se ha procesado de forma correcta desde el servidor.

```swift
func changeSessionState(){
    if(MyVariables.sessionRunning){
        // Enviar petición al servidor para terminar la sesión actual
        AF.request("http://\(MyVariables.server_ip):5103/close_session",
                   method: .get).response { response in
            if let statusCode = response.response?.statusCode {
                if (statusCode == 200) {
                    // Si se ha procesado correctamente la petición en el servidor, cambiamos la GUI
                    buttonTitle = "Comenzar sesión"
                    descriptionTitle = "Sesión terminada. Pulse el botón para iniciar una nueva sesión."

                    // Cambiamos la variable que nos informa del estado de la sesión
                    MyVariables.sessionRunning = false
                } else {
                    print("Error: \(statusCode)")
                }
            } else {
                print("Error: \(response.error!)")
            }
        }
```

1. Se comprueba si existe una sesión abierta, de lo contrario en este mismo método se realizaría la petición GET para iniciar una nueva sesión, en lugar de esta para cerrar la sesión existente.
2. Se prepara la petición mediante Alamofire.
3. Se gestiona la respuesta a la petición que nos devuelve la API. Si todo ha ido bien se vuelve a cambiar la interfaz de usuario y se cambia el valor de la variable sessionRunning.

Figura 33. *Envio de la petición GET al endpoint /close_session desde la app iOS*

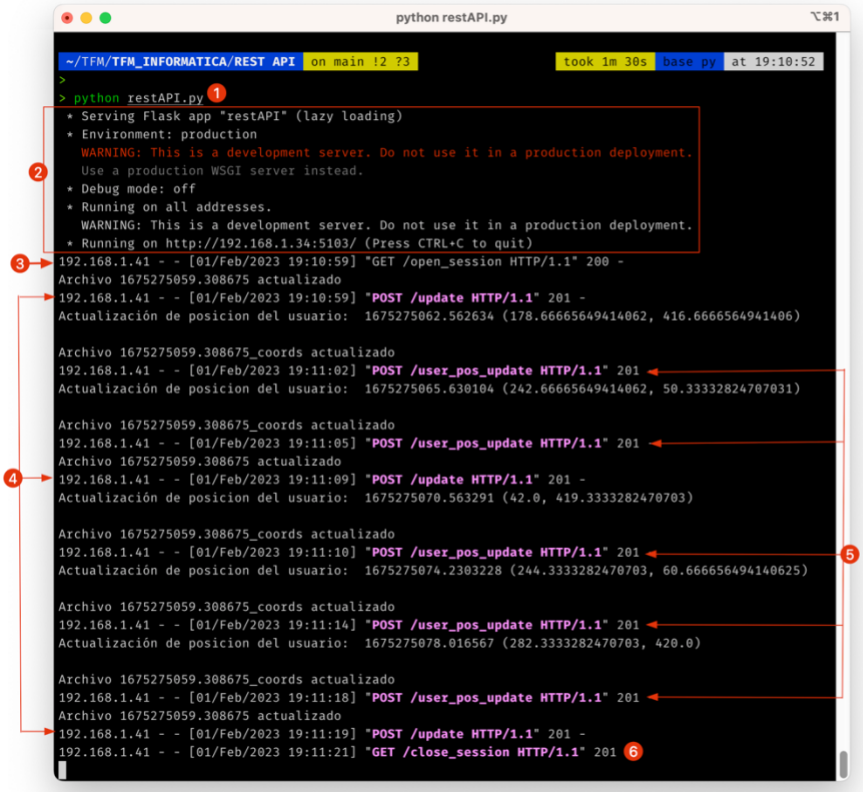
Por último, también era necesario implementar otro endpoint para capturar y procesar las etiquetas del usuario con respecto a su localización, por lo que se implementó también el endpoint */user_pos_update*. De forma similar a como se hacía en el endpoint */update*, en este se reciben los datos en formato JSON, se abre el archivo de la sesión donde se almacenan estos datos y se insertan al final de dicho archivo en una única fila y separados por espacios. En este caso los datos que se reciben son únicamente la marca de tiempo (*time*), la coordenada x (*xCoordinate*) del mapa del entorno que se muestra en la aplicación y la coordenada y (*yCoordinate*).

Por supuesto antes de eso se comprueba si existe una sesión en ejecución, a través del valor de la variable global *sessionId* como ya se ha explicado anteriormente.

```python
# POST Nueva etiqueta de la posición del usuario
@app.route('/user_pos_update', methods=['POST']) 1
def user_pos_update():
    if sessionId != None: 2
        filename = sessionId+"_coords" 3

        # Obtenemos los parámetros enviados en la petición
        json = request.get_json()

        time = json['timestamp'] 4
        xCoordinate = json['x']
        yCoordinate = json['y']

        print("Actualización de posicion del usuario: ", str(time)+" ("+str(xCoordinate)+", "+str(yCoordinate)+")\n")

        # Actualizamos el archivo donde almacenaremos los datos de la sesión
        f = open(filename, "a") 5
        f.write(str(time)+" "+
                str(xCoordinate)+" "+
                str(yCoordinate)+"\n")
        f.close()

        print("Archivo "+filename+" actualizado")

        response = { 6
            "message": "Actualización recibida y almacenada con éxito.",
            "sessionId": filename
        }
        return jsonify(response), 201 7
    else:
        print("SessionID null")
        response = {
            "message": "No existe sesión abierta.",
            "sessionId": filename
        } 8
        return jsonify(response), 400 # POST Bad Request
```

1. Directiva @app.route() para establecer el endpoint.
2. Comprobamos que exista una sesión en ejecución
3. El archivo donde se guardan los datos de la posición del usuario forma por el identificador de la sesión + el sufijo "_coords".
4. Obtenemos los datos de la petición POST.
5. Guardamos la información en el archivo correspondiente, sin sobreescribir los datos anteriores.
6. Creamos la respuesta a la petición.
7. Se devuelve la respuesta con el código de estado 201, indicando que se ha procesado satisfactoriamente.

Figura 34. *Endpoint /user_pos_update de la API REST*



1. Ejecutamos el script Python
2. Al iniciar la ejecución del script se lanza el servicio de Flask, levantando la API REST en la dirección IP privada correspondiente al dispositivo donde se está ejecutando y el puerto indicado en la instancia de la aplicación Flask
3. Se inicia una nueva sesión mediante la petición GET al endpoint / open_session
4. El envío de datos por lotes se realiza a través del endpoint /update mediante el método POST
5. Las etiquetas de la posición que realiza el usuario de forma manual se reciben de igual forma mediante petición POST, pero al endpoint / user_pos_update.
6. Mediante otra petición de tipo GET al endpoint /close_session, el dispositivo móvil indica a la API que finaliza la sesión actual, por lo tanto a partir de entonces no se recibirán más actualizaciones de muestras de datos ni etiquetas de localización.

Figura 35. *Ejecución de la API REST*

Durante la ejecución del script en Python, en la terminal del sistema se muestra el inicio de la API, la dirección IP y puertos donde se mantiene a la escucha de las peticiones, así como distintos mensajes que nos indica cuando un dispositivo está interactuando con los diferentes endpoints, como se puede observar en la Figura 35.

Los ficheros resultantes se almacenan en el servidor. Por cada sesión dispondremos de 2 ficheros distintos, uno que tendrá por nombre la marca de tiempo (*timestamp*) del instante en el que comenzó la sesión y que albergará los datos de las muestras UWB que son recopiladas y almacenadas brevemente en el dispositivo móvil, que son enviadas periódicamente hasta el servidor REST, como hemos visto en esta sección, donde finalmente se procesan y almacenan persistentemente. Estos archivos contendrán los datos de la marca temporal en la que se ha tomado el dato en el dispositivo, la distancia entre el dispositivo móvil y la baliza UWB calculada por la propia baliza, el identificador del dispositivo móvil donde se ha recopilado este dato y el identificador de la baliza UWB desde la cual se ha tomado la muestra.

En la siguiente figura se muestra un ejemplo de uno de estos archivos:

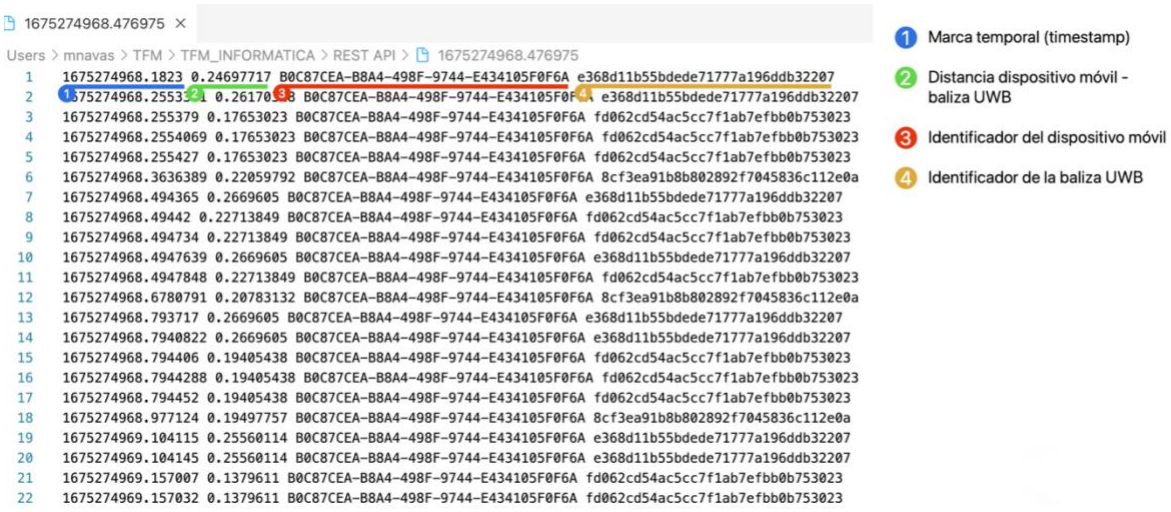


Figura 36. ***Archivo de muestras UWB***

Por otro lado, en otro archivo que tendrá por nombre la misma marca temporal y el sufijo "_coords" se almacenarán las etiquetas que el usuario realiza manualmente sobre su posición, definida por sus coordenadas (x,y) en el mapa del entorno que se

le muestra en la aplicación móvil junto con la marca temporal del instante de tiempo en el que se ha tomado dicha posición. En la siguiente figura se muestra el archivo de etiquetas asociado al fichero de muestras mostrado en la figura anterior:

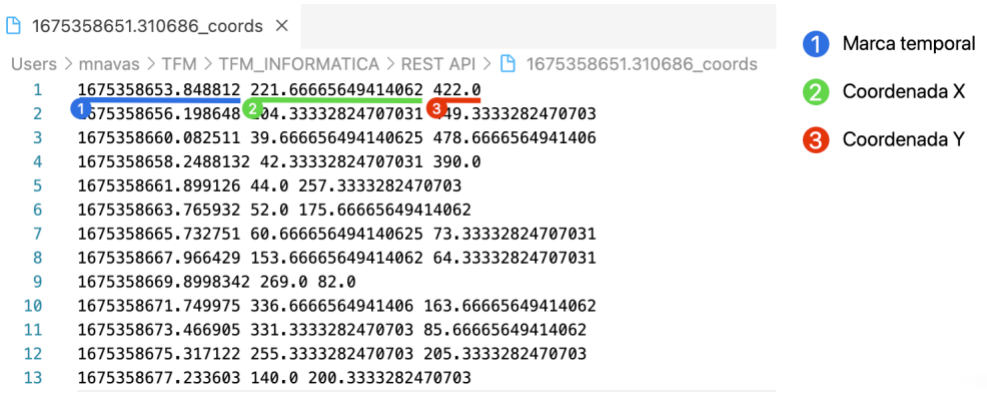


Figura 37. *Fichero de etiquetas de posición con respecto a un instante de tiempo*

Por cada sesión de toma de muestras tendremos un par de estos ficheros, que servirán de entrada para el modelo de Deep Learning que posibilitará la estimación de la localización en el entorno interior donde previamente se han tomado las distintas tomas de datos. En la siguiente sección se explicará en qué consiste el modelo desarrollado.

3.5 Modelo de Deep Learning

El aprendizaje profundo o Deep Learning está permitiendo la innovación y el cambio en todos los aspectos de nuestra vida moderna. La mayoría de los avances en inteligencia artificial que se alcanzan en la actualidad y de los que tanto se habla en los medios de comunicación se basan en el Deep Learning. Como ya se ha explicado con más detalle en la sección 2.1, Deep Learning es un subcampo de la inteligencia artificial que se centra en la creación de algoritmos inspirados en la estructura y el funcionamiento del cerebro, conocidos como redes neuronales artificiales. Estos algoritmos están diseñados para aprender de datos complejos y tomar decisiones o hacer predicciones basadas en patrones extraídos del conjunto de datos utilizado para entrenar la red neuronal artificial (el modelo). Estas redes neuronales son capaces de tomar decisiones precisas y son adecuadas

principalmente en contextos en los que los datos son complejos y se dispone de conjuntos representativos como datos de partida.

Uno de los mejores indicativos de la enorme aceleración en cuanto al rendimiento en los distintos campos en los que el Deep Learning se ha empleado hasta la actualidad es el de AlphaGo, el programa informático que fue capaz de derrotar a un jugador profesional de Go[13] en 2016 y al campeón mundial solo un año después. El éxito de este software fue muy sorprendente debido a que Go es un juego mucho más complejo que el ajedrez (hay más configuraciones de tablero posibles en Go que átomos en el universo) tanto para humanos como para ordenadores. La complejidad computacional entre el ajedrez y el Go se muestra claramente en el desfase temporal entre el desarrollo de los programas de ordenador que pudiesen competir contra humanos en ambos juegos. Los programas de ajedrez tardaron 30 años en progresar desde la competencia a nivel humano en 1967 hasta el nivel de campeón mundial en 1997, sin embargo, con el desarrollo del Deep Learning, los programas informáticos para competir contra humanos en Go sólo tardaron 7 años en pasar del nivel de aficionado avanzado al de campeón del mundo (2009-2016)(Hölldobler et al., 2017).

Debido a que el Deep Learning engloba a modelos que permiten tomar decisiones basadas en datos identificando y extrayendo patrones de grandes conjuntos de datos que mapean con precisión desde conjuntos de entradas complejas de datos, AlphaGo utilizó esta técnica para evaluar las configuraciones del tablero y decidir el siguiente movimiento a realizar. Para la propuesta de este Trabajo de Fin de Máster se empleará un modelo de Deep Learning basado en *Convolutional Neuronal Network* y *Long short-term memory*, entrenado con las distancias en bruto recogidas por las balizas UWB y la herramienta de recolección de datos (dispositivo móvil), para generar un sistema de localización en interiores sobre el mismo entorno donde se han desplegado las balizas UWB.

[13] Go es un antiguo juego de mesa chino que se juega en un tablero cuadriculado de 19x19 con piedras blancas y negras. El objetivo del juego es rodear y capturar más territorio que tu oponente. Se considera uno de los juegos más antiguos y complejos del mundo, y es popular en muchos países, sobre todo de Asia Oriental.

Para el desarrollo de este modelo de Deep Learning se utilizará nuevamente el lenguaje Python, pues es uno de los lenguajes más populares para el aprendizaje automático en general debido a las características de este lenguaje que se destacaron en la sección anterior. Una de estas características era la disponibilidad de librerías y paquetes que se pueden exportar y aportan muchas funcionalidades útiles para este tipo de desarrollos. Algunas de estas bibliotecas para computación científica, análisis de datos y aprendizaje automático incluyen:

- NumPy: proporciona soporte para matrices, que son una estructura de datos utilizada en muchas aplicaciones científicas y matemáticas. También proporciona funciones matemáticas para realizar operaciones con estas matrices, como álgebra lineal, generación de números aleatorios y transformadas de Fourier. Se utiliza ampliamente en la computación científica y el análisis de datos, y a menudo se utiliza como base para otras bibliotecas de Python en estos campos.

- SciPy: proporciona funciones para trabajar con matrices, optimización numérica, procesamiento de señales y mucho más. Está basada en NumPy y se utiliza mucho en computación científica.

- Pandas: Proporciona estructuras de datos para almacenar de forma eficiente grandes conjuntos de datos, incluidas matrices y marcos de datos, así como funciones para limpiar, transformar y analizar datos. Se utiliza ampliamente para el preprocesamiento de datos y la ingeniería de características en proyectos de aprendizaje automático y ciencia de datos.

- Matplotlib: biblioteca de visualización de datos que proporciona funciones para trazar gráficos 2D y 3D, histogramas, diagramas de dispersión, gráficos de barras, gráficos circulares, barras de error, etc. Se utiliza ampliamente para el análisis y la visualización de datos en comunidades científicas y de investigación.

- Scikit-Learn: implementa algoritmos de aprendizaje automático y herramientas para el análisis de datos. Presenta una variedad de algoritmos, incluyendo regresión, clasificación, agrupación y reducción

dimensional, así como herramientas para el preprocesamiento de datos y la evaluación de modelos.

- TensorFlow: es una biblioteca matemática simbólica y se utiliza para aplicaciones de aprendizaje automático como las redes neuronales.

- Keras: biblioteca de software de código abierto que proporciona una interfaz Python para RNA (redes neuronales artificiales). Actúa como interfaz para la biblioteca TensorFlow. Se desarrolló centrándose en permitir una experimentación rápida y fue capaz de soportar tanto redes convolucionales como redes recurrentes, así como combinaciones de ambas.

Estas bibliotecas proporcionan herramientas para la computación numérica, la manipulación y análisis de datos o la construcción de modelos de aprendizaje automático y son ampliamente utilizadas por científicos de datos, investigadores e ingenieros para resolver una variedad de tareas en estos campos.

3.5.1 Segmentación de flujo de datos

La segmentación de datos es el proceso que permite dividir un gran conjunto de datos en subconjuntos o segmentos más pequeños y manejables en función de determinadas características o atributos. El objetivo de la segmentación es alinear los datos, para comprender patrones e identificar tendencias y obtener información sobre segmentos temporales específicos de los datos que puedan servir de base para la toma de decisiones y el desarrollo de estrategias y soluciones más eficaces. En el contexto de este Trabajo de Fin de Máster, este proceso se aplica con el objetivo de proporcionar una representación sólida de los datos para homogeneizar los heterogéneos datos en bruto de los sensores UWB.

Siguiendo una definición formal, una baliza s detecta la presencia de una etiqueta e en tiempo real en forma de un par:

$$\overline{(s,e)}_i = \{s_i, t_i\},$$

Donde $(s,e)_i$ representa la distancia dispositivo móvil – baliza y t_i la marca temporal. De esta forma obtenemos un flujo de datos para una baliza y una etiqueta dadas (s,e), los cuales son definidos por:

$$\overline{S_{(s,e)}} = \{\overline{(s,e)_0}, \dots, \overline{(s,e)_\iota}\}$$

Y un valor dado en una marca temporal t_i por:

$$S_{(s,e)}(t_i) = s_i$$

Definimos varias ventanas temporales deslizantes[14] para agrupar los datos recogidos por las diferentes balizas UWB en diferentes intervalos cortos de tiempo. Las ventanas se definen con un intervalo de tiempo $W_w = [W_w^-, W_w^+]$ que selecciona las muestras de un flujo de datos $\overline{S_{(s,e)}}$ y agregan los valores $\overline{(s,e)_\iota}$ en una única medida, utilizando para ello una función de agregación:

$$T_t(S_{(s,e)}, W_w, t^*) = \bigcup_{\overline{(s,e)_i}}^{S_(s,e)} s_i, t_i \in [t^* - W_w^-, t^* - W_w^+]$$

Así, las funciones de agregación ∪ se aplican sobre los datos $(s,e)_i$ en un intervalo de tiempo $W_w = [W_w^-, W_w^+]$ en un momento dado t^*.

En concreto, las siguientes funciones de agregación que se proponen han sido media, máximo y mínimo y han sido definidas en otras propuestas como descriptores clave para flujos sensores (Lopez Medina et al., 2019).

La segmentación de la señal se realiza mediante varias ventanas temporales deslizantes de intervalos consecutivos:

$$W = [W_1^-, W_1^+], \dots, W_i = [W_i^-, W_i^+], \; W_i^- = W_{i-1}^+$$

[14] Una ventana temporal deslizante es un intervalo móvil de duración determinada que se utiliza para seleccionar datos que cambian con el tiempo, con el objetivo de obtener una mejor comprensión de los patrones subyacentes y las tendencias de los datos en flujos de tiempo real.

De este modo, obtenemos una secuencia de características a partir de flujos de datos de las balizas para cada etiqueta e, cuya forma es $|S| \; x \; |\cup| \; x \; |W|$:

$$S_e^*(t^*) \rightarrow \begin{cases} S_e^1(t^*) = T_t(S_{(S1,e)}, [W_1^-, W_1^+], t^*) \ldots, \\ \rightarrow T_t(S_{(S1,e)}, [W_i^-, W_i^+], t^*) \\ \ldots \\ S_e^s(t^*) = T_t(S_{(s,e)}, [W_1^-, W_1^+) \ldots, \\ \rightarrow T_t(S_{(s,e)}, [W_i^-, W_i^+], t^*) \end{cases}$$

3.5.2 Modelo de Deep Learning para estimar la posición

En este apartado se describe el modelo Deep Learning para estimar la posición de los habitantes de un espacio interior en tiempo real.

Primero, se integran **tres capas de una Red Neuronal Convolucional 1D** (1D Convolutional Neural Network, 1D CNN) como extractores de características espaciales. Las redes neuronales convolucionales (CNN) son un tipo de red neuronal artificial utilizada habitualmente en tareas de reconocimiento de imágenes y vídeos, que se caracterizan en utilizar una serie de capas que procesan y analizan los datos de entrada, compuestas por varias capas convolucionales y de agrupación, seguidas de una o varias capas totalmente conectadas. Las capas convolucionales aplican filtros a los datos de entrada para detectar características, mientras que las capas de agrupación reducen las dimensiones espaciales de los mapas de características generados por las capas convolucionales. Las capas totalmente conectadas utilizan la información de alto nivel para unir todas las activaciones de la red y realizar predicciones o clasificaciones.

La característica 1D indica que se trata de un tipo de CNN que opera sobre datos unidimensionales, como secuencias o series temporales. En esta propuesta se conectan tres capas convolucionales, que a su vez contiene un número de filtros variable en cada una de estas capas, concretamente **2 filtros en la primera capa**, **3 filtros en la segunda** y **3 filtros de nuevo en la tercera capa**. Estos filtros se

desplazan sobre los datos de entrada y extraen características a diferentes escalas, creando un mapa de características.

A continuación, se añaden varias capas de otra red neuronal, en este caso del tipo memoria a corto plazo de larga duración, más conocida como **LSTM** (Long Short-Term Memory). Esta red es de tipo recurrente (RNN) y se utiliza para procesar datos secuenciales. Está diseñada para superar el problema del gradiente evanescente de las RNN tradicionales mediante la introducción de celdas de memoria y puertas que controlan el flujo de información a lo largo del tiempo.

El problema del gradiente evanescente en las RNN tradicionales es un fenómeno en el que los gradientes calculados durante la retropropagación se vuelven extremadamente pequeños, lo que provoca un aprendizaje lento o ineficaz. Esto ocurre porque en las RNN tradicionales, los gradientes se multiplican repetidamente por la misma matriz de pesos durante la retropropagación a lo largo del tiempo. Esto puede hacer que la magnitud de los gradientes disminuya rápidamente, haciéndose muy pequeña y provocando que la red sea incapaz de aprender de los datos de entrada. Este problema es especialmente pronunciado cuando se procesan secuencias de gran longitud, ya que los gradientes pueden llegar a ser tan pequeños como para desaparecer, de ahí el nombre de "gradiente evanescente". La arquitectura LSTM se diseñó para resolver este problema introduciendo células de memoria y puertas que permiten pasar o bloquear gradientes de forma selectiva, evitando el problema del gradiente evanescente y permitiendo a la red aprender de secuencias largas, recordar mejor y hacer predicciones basadas en dependencias a largo plazo en la secuencia de entrada.

En esta propuesta, **dos capas de LSTM con 32 unidades** modelan las dependencias temporales de las características superiores de la CNN. La idea es que la red LSTM tome la salida de la CNN y la procese de forma que tenga en cuenta la secuencia de características, lo que permite a la red aprender y hacer predicciones basadas en las relaciones entre las características de la secuencia. Las dos capas de LSTM con 32 unidades se utilizan para capturar y procesar las dependencias temporales de las características, aumentando la capacidad de la red para modelar con precisión las relaciones y hacer predicciones basadas en los datos de entrada.

Se incluyen **capas de reducción de valores (*dropout*)** para reducir los parámetros y evitar la co-adaptación de características y el sobreajuste (Zagoruyko & Komodakis, 2017). La sobre adaptación se produce cuando un modelo se ajusta demasiado a los datos de entrenamiento, lo que da lugar a un bajo rendimiento de la estimación en datos de test o evaluación. La inclusión de estas capas de reducción de valores ayuda a evitar la sobre adaptación al obligar a la red a ser más robusta y a realizar predicciones basadas en un conjunto diverso de activaciones, en lugar de depender demasiado de un único conjunto de características. Además, estas capas también pueden ayudar a evitar la co-adaptación de características, en la que diferentes características de una capa dependen demasiado unas de otras. La combinación de redes híbridas CNN-LSTM ha sido seleccionada por haber proporcionado resultados alentadores en fingerprinting en estudios anteriores (Hoang et al., 2020).

En tercer lugar, **un modelo de perceptrón multicapa (multi-layer perceptrón, MLP) de 2 capas de 512 y 256** aprende de los patrones espaciales y temporales para proporcionar una salida, que es la regresión de las ubicaciones X e Y con una función de activación sigmoidea. Las capas del MLP procesan los datos de entrada y extraen características que captan las relaciones entre las variables de entrada. Estas características se utilizan para producir una salida, que representa la predicción o estimación de la red de la variable objetivo, en este caso las coordenadas 2D de la ubicación del habitante.

En toda la red se aplica el **aprendizaje por retro propagación basado en la métrica de pérdida del error cuadrático medio y el optimizador Adam**. La retro propagación se trata de un algoritmo de aprendizaje supervisado utilizado para actualizar los pesos de la red en función del error de predicción. Dicho error de predicción se mide utilizando una métrica de pérdida, que en este caso es el error cuadrático medio. El error cuadrático medio mide la diferencia entre los resultados previstos y los reales de la red, y es una función de pérdida habitual en los problemas de regresión. El optimizador utilizado en este caso es Adam, que es un algoritmo de optimización basado en gradientes. El propósito del optimizador es determinar la mejor forma de actualizar los pesos de la red basándose en los gradientes calculados y la métrica de pérdidas. Adam es una opción popular para los optimizadores debido

a su capacidad para adaptar la tasa de aprendizaje en función de cada parámetro, lo que puede conducir a una convergencia más rápida y un mejor rendimiento en comparación con otros algoritmos de optimización.

En la siguiente figura se describe la configuración de capas en el modelo de Deep Learning propuesto que se acaba de describir.

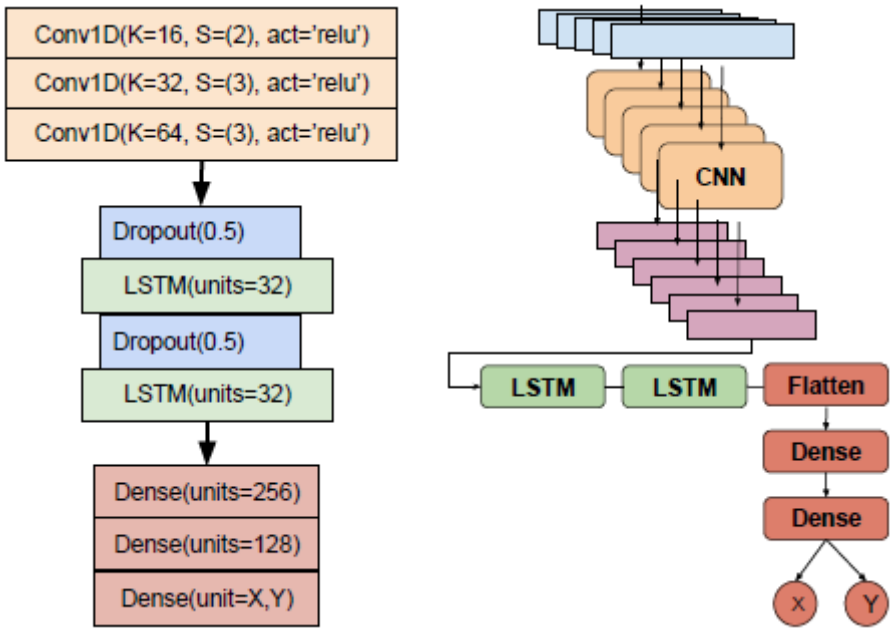


Figura 38. ***Configuración de capas del modelo Deep Learning propuesto.***

En esta propuesta evaluamos el rendimiento de CNN, LSTM y CNN+LSTM, que proporciona el resultado óptimo entre arquitecturas de modelos de Deep Learning.

3.5.3 Código del modelo Deep Learning

En esta sección se comenta brevemente los aspectos más relevantes del código que se ha implementado tanto para la segmentación de los datos de muestras UWB y coordenadas etiquetadas por el usuario como para la definición del modelo Deep Learning que se ha descrito en la sección anterior.

En primer lugar, en cuanto a la segmentación se comienza obteniendo los datos recopilados mediante el dispositivo móvil a lo largo de las distintas sesiones (tracks) de captura de datos realizadas.

```
# Datos entorno
```

```
flat="./data-iphone/"
folders=["track1","track2","track3","track4","track5"]

file_folder={}
for f in folders:
    for path in os.listdir(flat+f):
        # comprobar si la ruta actual está disponible
        if os.path.isfile(os.path.join(flat+f, path)) and not path.endswith("_coords"):
            file_folder[f]=path

print(file_folder)



# Leemos los datos de muestras UWB (tiempo, baliza y señal)
data=[]
N=0
for f in folders:
    series=read_csv(flat+f+"/"+file_folder[f], sep=" ",parse_dates=True,header=None)
    N=N+len(series.values)
    d2=series.values
    d2=np.array(d2)
    print(d2)
    data.extend(d2)

data=np.array(data)
print(len(data))
print(N)

# Leemos los datos de coordenadas etiquetadas por el usuario
dataY=[]

for f in folders:
    series=read_csv(flat+f+"/"+file_folder[f]+"_coords", sep=" ",parse_dates=True,header=None)
    for d in series.values:
        dataY.append(d)
dataY=np.array(dataY)
```

A continuación, se definen las variables de las ventanas temporales de valores “futuros” y “pasados” que segmentan los datos para crear una secuencia de valores a partir de un tiempo determinado.

```
# Ventanas temporales de valores "futuros" (los datos de distancia despues de la estimación de tiempo) hasta 12 segundos
tW2=[6,4,3,2,1,0]
# Ventanas temporales de valores "pasados" (los datos de distancia despues de la estimación de tiempo) hasta 12 segundos
tW1=[0,-1,-2,-3,-4,-6]
```

Seguidamente se leen de los datos de muestras UWB los identificadores de las distintas balizas que han sido detectadas por la herramienta de recolección de datos

del dispositivo móvil, en este caso como ya se ha explicado con anterioridad se muestran los *publicId* de las tres balizas que se han utilizado en ambos entornos.

```
# Leemos las diferentes balizas (3) que hay configuradas y desplegadas
print(data[:,4])
anchors=np.unique(data[:,4])
print(anchors)
```

Después se normalizan los valores de las distancias y se calculan el tiempo inicial y final de las muestras de datos junto con los valores mínimos, máximos y medios de las coordenadas X e Y.

```
# Normalizamos los valores de distancias
min_v=(float)(min(data[:,1]))
max_v=(float)(max(data[:,1]))
print(min_v,max_v)

# Tiempo inicial y final de los datos
min_t=(float)(min(dataY[:,0]))
max_t=(float)(max(dataY[:,0]))
print(min_t,max_t)

# Valores min max de la coordenada Y
min_y=(float)(min(dataY[:,2]))
max_y=(float)(max(dataY[:,2]))
print(min_y,max_y)

# Valores min max de la coordenada X
min_x=(float)(min(dataY[:,1]))
max_x=(float)(max(dataY[:,1]))
print(min_x,max_x)

# Esta función obtiene la media de un conjunto de señales de distancias normalizadas
def getAVG(vs):
  if (len(vs)==0):
    return 0

  return (np.mean(vs)-min_v)/(max_v-min_v)

# Esta función obtiene el máximo de un conjunto de señales de distancias normalizadas
def getMAX(vs):
  if (len(vs)==0):
    return 0

  return (max(vs)-min_v)/(max_v-min_v)

# Esta función obtiene el minimo de un cojunto de señales de distancias normalizadas
def getMIN(vs):
  if (len(vs)==0):
    return 0
```

```
return (min(vs)-min_v)/(max_v-min_v)
```

Por último, se realiza un bucle desde el tiempo inicial al final de las muestras recogidas donde por cada segundo que transcurre en ese espacio de tiempo:

i) se recoge la posición (x,y) más cercana anterior y posterior a cada instante de tiempo procesado en el bucle,

ii) se calcula el incremento de tiempo respecto al instante analizado, desde la ventana "futura" y "pasada",

iii) se descarta la muestra si el usuario la etiquetó en un tiempo superior a 10 segundos,

iv) se extraen las 3 características (máximo, mínimo y media) de entre todos los datos que se incluyen dentro del segmento establecido por cada ventana temporal,

v) añadimos los valores de entradas futuras y pasadas a la secuencia,

vi) calculamos una posición (x,y) de la imagen ponderada respecto las posiciones más cercanas anterior y posterior a este tiempo t,

vii) normalizamos los valores de la posición obtenida, y

viii) por último, añadimos la secuencia de características a la variable X junto con la posición (x,y) calculada previamente a la variable Y.

En el siguiente fragmento de código se muestra el bucle *for* donde se implementan todas estas acciones:

```
# Desde el tiempo inicial al final (en segundos) de uno en uno (con un desplazamiento en funcion del tamaño de ventana para recoger datos anteriores y futuros)
for t in np.arange(min_t-min(tW1),max_t-max(tW2),1.0):
    print(t)

    # Recogemos la posición de X,Y más cercana anterior y posterior a este tiempo t
```

```
agXY0=dataY[np.sort(np.where( (dataY[:,0] <  t)))][0][-1]
agXYNn=dataY[np.sort(np.where( (dataY[:,0] >=  t)))][0][0]
# Calculamos el incremento de tiempo respecto t, del futuro y del pasado
incT0=t-agXY0[0]
incTN=agXYNn[0]-t
print("agXY0:",agXY0, "incT0:",incT0)
print("agXYNn:",agXYNn, "incTN:",incTN)

# Importante: si el usuario no etiquetó esa muestra en un tiempo cercano (a 10 segundos), la descartamos
if(incT0>5 or incTN> 5):
    print("NOT DATA AVAILABLE!")
    continue

#valores de las x-> NO posición, los datos de entrada de distancias
xi=[]
for a in anchors:
    xirssi1=[]
    xirssi2=[]
    xirssi3=[]
    xirssi4=[]
    xirssi5=[]
    xirssi6=[]
    # Ventanas futuras
    for it in range(len(tW1)-2,-1,-1):
        t0=t+tW1[it]
        tN=t+tW1[it+1]
         # Todos los datos en ese segmento de ventana
        ag1=data[np.where((data[:,4] == a) & (data[:,0] <= t0) & (data[:,0] >=  tN))]

        # Sacamos las 3 características de esa ventana (max,min,media)
        xirssi1.append(getAVG(ag1[:,1]))
        xirssi2.append(getMAX(ag1[:,1]))
        xirssi3.append(getMIN(ag1[:,1]))

    # Ventanas pasadas
    for it in range(0,len(tW2)-1):
        t0=t+tW2[it]
        tN=t+tW2[it+1]
                # Todos los datos en ese segmento de ventana
        ag1=data[np.where((data[:,4] == a) & (data[:,0] <= t0) & (data[:,0] >=  tN))]

        # Sacamos las 3 características de esa ventana (max,min,media)
        xirssi4.append(getAVG(ag1[:,1]))
        xirssi5.append(getMAX(ag1[:,1]))
        xirssi6.append(getMIN(ag1[:,1]))

    # Añadimos los valores de entradas futuras y pasadas a la secuencia
    xi.append(xirssi1)
    xi.append(xirssi2)
    xi.append(xirssi3)
    xi.append(xirssi4)
```

```
        xi.append(xirssi5)
        xi.append(xirssi6)

    # Calculamos una posición X,Y de la imagen ponderada respecto las posiciones más
    # cercanas anterior y posterior a este tiempo t
    w0= 1-(incT0)/(incT0+incTN)
    wn= 1-(incTN)/(incT0+incTN)

    posX=(agXYNn[1]*wn+agXY0[1]*w0)
    posY=(agXYNn[2]*wn+agXY0[2]*w0)
    print("posX:",posX, "posY:",posY)

    # Normalizamos
    posX=(posX-min_x)/(max_x-min_x)
    posY=(posY-min_y)/(max_y-min_y)
    print("posX:",posX, "posY:",posY)

    # Añadimos la sequencia de caracteristicas X e Y (posX,posY de la imagen)
    print("xi.shape:",np.array(xi).shape)
    X.append(np.transpose(xi))
    Y.append([posX,posY])

    print("count:",c)

    c=c+1
    if(c>5000):
        break
```

Una vez terminado el proceso de segmentación se realiza una conversión necesaria de las secuencias de características a vectores de datos de tipo NumPy para trabajar con ellos en el modelo Deep Learning.

```
# Importante: pasar los datos a NUMPY para trabajar con el modelo de DL
X=np.array(X)
Y=np.array(Y)
print(X.shape)
print(Y.shape)
```

En cuanto a este modelo, se comienza dividiendo los datos en 10 conjuntos para implementar la validación cruzada con $k = 10$ y se inicializa el fichero xy.tsv donde escribir los resultados reales y estimados. La validación cruzada de k repeticiones es un método común utilizado en el aprendizaje automático para evaluar el rendimiento de un modelo. Consiste en dividir el conjunto de datos en k partes iguales (pliegues), en el caso de $k = 10$ utilizando k - 1 pliegues para entrenar el modelo y el pliegue restante para las pruebas. Este proceso se repite k veces y cada pliegue sirve una vez como conjunto de prueba. El rendimiento medio del modelo en los k conjuntos de

pruebas se utiliza como medida de rendimiento global del modelo. El uso de múltiples pliegues ayuda a reducir la varianza en la métrica de evaluación, proporcionando una estimación más sólida del rendimiento del modelo. En este caso, como $k = 10$ se realizan 10 pliegues y se repite el proceso 10 veces, utilizando en cada iteración uno de los 10 pliegues para las pruebas y el resto para entrenar el modelo. Al finalizar todas las iteraciones de la validación cruzada se genera el fichero denominado xy.tsv donde se almacenan las posiciones reales junto a las estimadas, y en la salida por consola del script en Python del modelo se muestra el error absoluto en cuanto a la coordenada X e Y.

```
# Dividimos los datos en K iteraciones para validación cruzada
kf = KFold(n_splits=10,shuffle=True)
kf.get_n_splits(X)



# Fichero para escribir los resultados de prediccion y reales (se sobrescribe el anterior)
f = open(flat+"xy.tsv", "w")
f.write("")
f.close()
```

Después se comienza el bucle donde se implementarán las acciones de cada iteración de la validación cruzada:

i) Separamos los datos de entrenamiento y de prueba,

ii) se genera un nuevo modelo,

iii) se añaden las 3 capas de la red neuronal convolucional de 2, 3 y 3 filtros respectivamente,

iv) se agregan 2 capas de LSTM de secuencia, junto con las capas de reducción de valores con dropout de 0.5,

v) se añade un modelo de perceptrón multicapa y se realiza regresión de dos valores (x,y),

vi) se incorpora la retro propagación basada en la métrica de pérdida del error cuadrático medio y el optimizador Adam,

vii) se itera 1000 veces,

viii) se realiza la predicción de esta iteración de la validación cruzada,

ix) se produce una desnormalización de la salida entre 0 y 1 para tener coordenadas del mundo real y

x) se calcula el error absoluto y se añade índice, posiciones reales y predichas al fichero xy.tsv.

```
# Para cada K del cross validator
for train_index, test_index in kf.split(X):
    # Datos de train y test
    print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_val = X[train_index], X[test_index]
    y_train, y_val = Y[train_index], Y[test_index]


    # Nuevo modelo
    model = keras.Sequential()

    # 3 capas de convolucion
    model.add(Conv1D(filters=16, kernel_size=2, activation='relu',
                                input_shape=(X.shape[1],X.shape[2])))
    model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))

    # 2 capas de LSTM de secuencia
    model.add(Dropout(0.5))
    model.add(LSTM(32, return_sequences=True))
    model.add(Dropout(0.5))
    model.add(LSTM(32))

    # Lo pasamos a una capa densa y hacemos regresion de dos valores (posX,posY)
    model.add(Flatten())
    model.add(Dense(512))
    model.add(Dense(256))
    model.add(Dense(2, activation="sigmoid"))

    model.compile(loss='mean_squared_error', optimizer='adam')
    model.summary()

    # Iteramos 1000 veces
    history = model.fit(x_train , y_train,
                        batch_size=64,
                        validation_data=(x_val,y_val),
                        epochs=300)
```

```
# Predecimos
yhat = model.predict(x_val)

# Desnormalizamos
x1=y_val[:,0]*(max_x-min_x)+min_x
y1=y_val[:,1]*(max_y-min_y)+min_y
x2=yhat[:,0]*(max_x-min_x)+min_x
y2=yhat[:,1]*(max_y-min_y)+min_y

# Calculamos error absoluto y ponemos indice, posiciones reales y predichas
absx=0.0
absy=0.0

f = open(flat+"xy.tsv", "a")
for i in range(0,len(x2)):
  f.write(str(test_index[i])+"\t"+str(x1[i])+"\t"+str(y1[i])+"\t"+str(x2[i])+"\t"+str(y2[i])+"\n")
            absx=absx+abs(x1[i]-x2[i])
    absy=absy+abs(y1[i]-y2[i])
  f.close()

absx=absx/(float)(len(x2))
absy=absy/(float)(len(x2))
print("absx:"+str(absx))
print("absy:"+str(absy))
```

La salida del modelo nos muestra al final el error absoluto de las coordenadas X e Y por separado, como se puede observar en la Figura 41. En el siguiente capítulo se explica cómo se han validado los resultados obtenidos por esta propuesta de sistema de localización en interiores.

En el siguiente hipervínculo se puede acceder a una carpeta compartida en Google Drive donde se ha compartido tanto el código del software desarrollado para este Trabajo de Fin de Máster como los datos obtenidos mediante las diferentes sesiones de captura en los distintos entornos.

4 Validación

En este capítulo se explicarán los dos escenarios donde se ha puesto en práctica la propuesta de este Trabajo de Fin de Máster, haciendo hincapié en aspectos como la localización, dimensión, duración de las pruebas, número de datos recogidos, etc. Después se comentarán los resultados en cada caso, atendiendo al error absoluto medio de las coordenadas X e Y del plano de cada entorno aplicando validación cruzada con 10 iteraciones. Para terminar, se mostrará un ejemplo de estimación de la localización de forma gráfica, donde se muestran los datos reales de coordenadas y las estimaciones del modelo de Deep Learning.

4.1 Entornos de prueba

Entre los distintos objetivos del Trabajo de Fin de Máster se encontraba tanto el despliegue de sensores ambientales UWB como la recolección de datos en un entorno real. El sistema propuesto está pensado para la localización en entornos de interiores, con una disposición similar a la que podríamos encontrar en cualquier domicilio particular, residencias, centros de día para personas mayores o dependientes, etc. Para llevar a cabo el despliegue de los sensores UWB y el estudio de varios casos se han utilizado dos domicilios distintos con una distribución parecida, donde destacan dos habitaciones que se encuentran separadas por una pared y un pasillo que comunica ambas zonas a través de sus respectivas puertas de acceso. A continuación, se detallan cada uno de estos entornos de prueba:

4.1.1 Entorno A

El primer entorno de pruebas donde se desplegaron las balizas UWB y se recolectaron datos se encuentra situado en la localidad de Torredonjimeno (Jaén). Se trata de una casa de tres plantas, donde concretamente se utilizó dos salas contiguas.

Una de ellas se trata de una cocina (la habitación de la izquierda en la Figura 39) que cuenta con una puerta de acceso y un ventanal que da acceso a un patio con salida al exterior del edificio. Dentro de esta sala encontramos los habituales muebles de cocina y electrodomésticos, como una vitrocerámica, un horno o un frigorífico,

todos ellos empotrados en los muebles. Además, también dispone de una mesa en el centro de la sala acompañada de cuatro sillas. En este mueble en concreto se ha colocado una de las balizas UWB, ya que ocupa un lugar central estratégico que resulta ideal para el objetivo del sistema de localización. Las medidas de las paredes que conforman la sala se indican en la propia Figura 39, siendo el área de la sala de unos 15,50 m^2.

La segunda sala de este entorno de pruebas se trata de una sala de estar, y tiene un papel más protagonista, ya que alberga dos de las balizas UWB. Una de ellas se dispuso en una mesita central, de forma análoga a la disposición que ocupó la baliza en la cocina del entorno, mientras que la otra se situó sobre el mueble del televisor de la sala, lo más cerca posible de la puerta de acceso que comunica esta sala de estar con el pasillo. Como se puede apreciar en la Figura 39 las puertas de acceso de ambas salas son contiguas, razón por la cual se seleccionó estas dos estancias del domicilio, pues son las que presentan menos obstáculos entre ambas salas contiguas. Aparte del mobiliario donde se colocaron las balizas UWB, encontramos un par de sofás, varias sillas, otra mesa auxiliar, una ventana que sale al exterior del edificio y un ventanal que da acceso al mismo patio que se mencionó en la sala anterior. Las medidas de las paredes, de nuevo, se pueden encontrar en el esquema de la Figura 39, siendo el área total de la sala aproximadamente de unos 17,96 m^2.

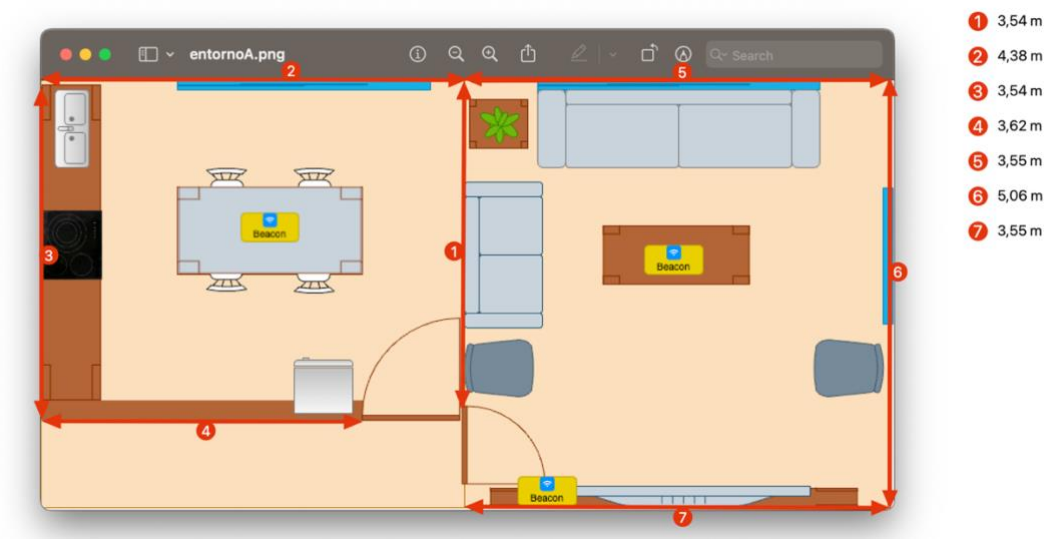


Figura 39. *Esquema del entorno A*

En este entorno de pruebas y con el despliegue que se ha detallado, se realizaron un total de 5 sesiones de toma de datos (tracks) de unos 4 – 5 minutos de duración. De cada una de estas sesiones se recolectaron miles de muestras de datos UWB desde las diferentes balizas y se realizaron un promedio de 30 etiquetas de la localización mediante la herramienta desarrollada en el dispositivo móvil. En total se obtuvieron 3639 muestras y 155 etiquetas de coordenadas.

4.1.2 Entorno B

El segundo entorno de pruebas que se ha estudiado se encuentra localizado en el municipio de Sigüenza, en la provincia de Guadalajara. Concretamente se trata de un piso de unos 100 m^2 formado por tres habitaciones, dos baños un salón y una cocina. De todas estas salas se han seleccionado las dos habitaciones contiguas, por la misma razón que en el entorno A.

Ambas habitaciones cuentan con un mobiliario similar, destacando una cama, armario y mesa auxiliar. También ambas disponen de una única puerta de acceso y una ventana como salida al exterior. En este caso, a diferencia del entorno A, se ha optado por situar una baliza UWB en cada habitación en la ubicación más céntrica posible. En una de ellas se colocó en el extremo inferior central de la cama mientras que en la segunda habitación se colocó en la mesa auxiliar pues no existía opción a colocar la baliza de forma más céntrica sin colocarla en el suelo de la sala, pero esta alternativa no resultaba práctica ni realista. La tercera baliza en este entorno de pruebas se ha colocado en una posición intermedia entre ambas salas, concretamente aprovechando un muro frontal a la puerta de acceso de ambas habitaciones, de forma que esta baliza pudiera comunicarse de la forma más equitativa posible con la herramienta de recolección de datos del dispositivo móvil en ambas salas. En la Figura 40 se puede visualizar un esquema de la disposición de las balizas junto con las medidas de las paredes de las salas del entorno, siendo el área de la sala que queda más a la izquierda en el esquema de 7,93 m^2 y el de la sala a la derecha de unos 10,68 m^2.

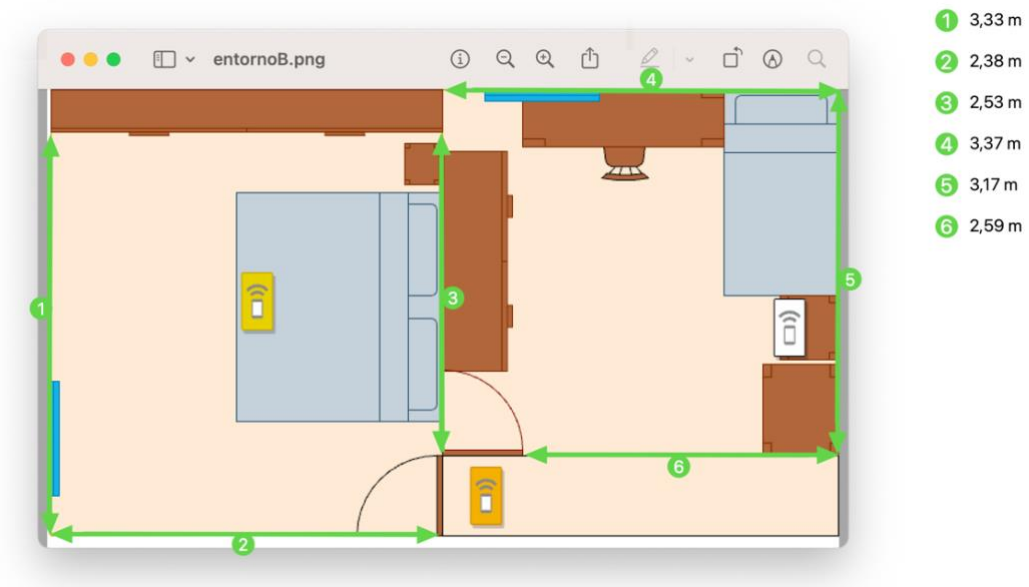


Figura 40. *Esquema del entorno B*

De forma análoga al primer entorno, para este segundo despliegue se tomaron en total otras 5 sesiones de unos 2 – 3 minutos de duración, aproximadamente la mitad de duración de las pruebas realizadas en el entorno A. Esto se debe a que se pretendía comprobar si una localización más equitativa de las balizas supondría una mejora en cuanto a los resultados, por lo cual la hipótesis fue que, si disminuíamos la duración de las sesiones, con una menor cantidad de datos podríamos obtener resultados parecidos a las pruebas realizadas en el primer entorno, a pesar de que la ubicación de las balizas UWB era más desfavorable para una de las salas. Por esta razón, como se puede intuir, para este entorno se han obtenido una cantidad inferior de muestras, en concreto unas 1917. En cuanto a las etiquetas de localización generadas mediante la aplicación del dispositivo móvil, se generaron de media entre 37 y 38 etiquetas por sesión con un total de 188 entre todos los tracks realizados.

4.2 Resultados

Los datos recogidos de cada uno de los entornos de prueba se utilizaron como conjunto de datos de entrada para el modelo de Deep Learning que se ha detallado en la sección 3.5. Para evaluar la efectividad del modelo y los datos obtenidos de las

distintas pruebas se calcula el error absoluto mediante la técnica de k-fold cross validation (validación cruzada de k iteraciones), eligiendo en este caso $k = 10$.

Para los datos del entorno A se obtiene como resultado un error absoluto en cuanto a la coordenada X de unos 46,69 píxeles (de la pantalla del dispositivo móvil) y de 44,03 píxeles en cuanto a la coordenada Y. Para interpretar este error de forma más precisa es necesario realizar la conversión de píxeles a metros, teniendo en cuenta las medidas del entorno que hemos detallado en la Figura 39, se puede calcular que cada metro del entorno se corresponde con aproximadamente 87,46 píxeles, por tanto, **el error absoluto de la coordenada X en el entorno A se corresponde con aproximadamente 0,53 m, mientras que para la coordenada Y se obtiene un error de 0,5 m**. En la siguiente figura se muestra la salida por consola del script del modelo junto con el fichero donde se han imprimido las posiciones reales junto con las estimadas.

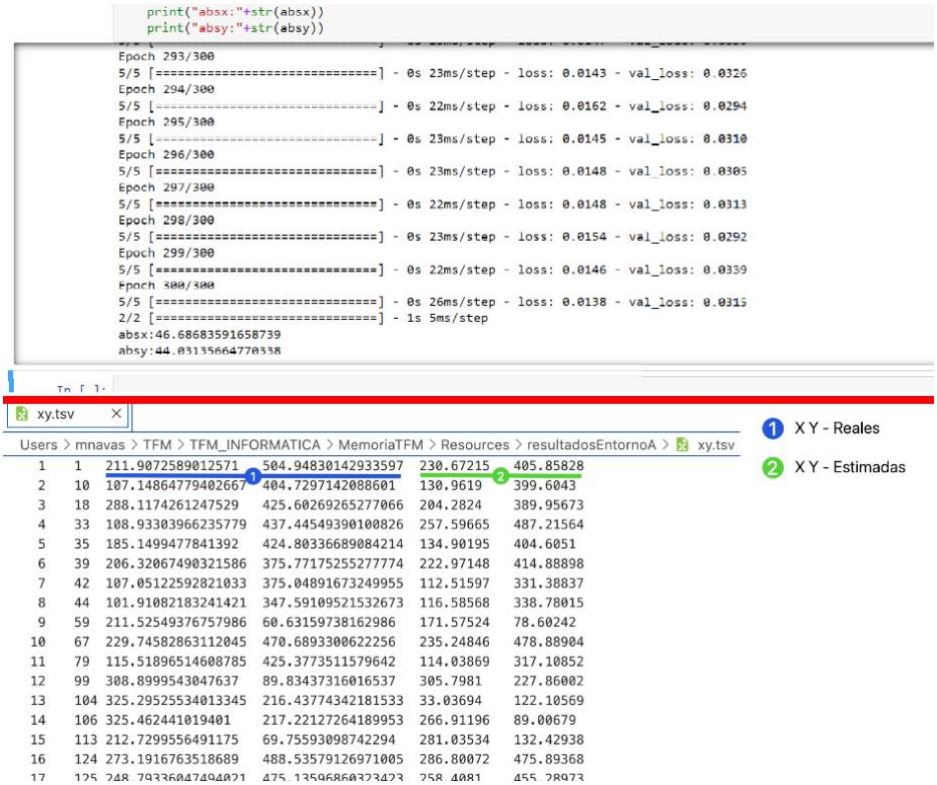


Figura 41. *Error absoluto por coordenadas (arriba) y fichero xy.tsv (abajo) donde se almacenan las coordenadas reales junto a las estimadas por el modelo para el conjunto de datos del entorno A.*

Adicionalmente se ha desarrollado un ligero script en Python con Jupyter Notebook (*IPython Notebook*) que permite visualizar gráficamente los puntos marcados mediante las coordenadas X e Y sobre el mismo plano del entorno de pruebas, de forma que podremos comparar las diferencias entre las coordenadas reales y las estimadas de los resultados obtenidos por el modelo. En la siguiente figura se muestra un par de capturas de pantalla de dos instantes donde se pueden ver que los puntos de las marcas estimadas aparecen la mayoría de las veces con buena precisión en relación con el punto de marca real.

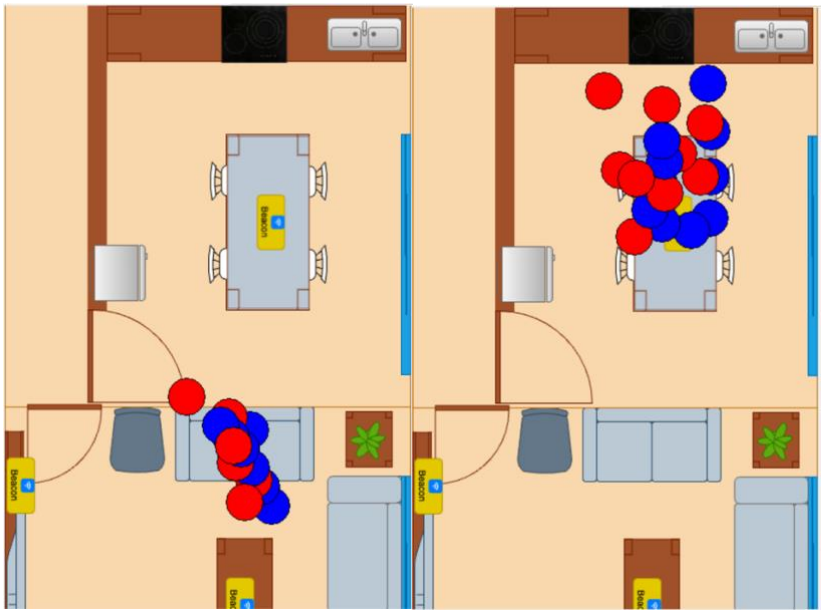


Figura 42. *Ejemplos de representación gráfica de las coordenadas reales (azules) y estimadas por el modelo (rojas) para el entorno A.*

Por otro lado, para los datos del entorno B se obtiene un error absoluto de 58,96 píxeles para la coordenada X y de 39,03 píxeles para la coordenada Y. Siguiendo la lógica empleada en el entorno A, en esta ocasión la relación píxeles por metro del entorno B se ha calculado en 91,54 píxeles/m, por tanto, los errores anteriores se traducen **a 0,64 m para la coordenada X y 0,43 m para la coordenada Y**. Como se puede observar se ha obtenido un error mayor para la coordenada X si lo comparamos con el anterior entorno (0,53 m), pero por otro lado ligeramente inferior en cuanto a la coordenada Y (0,50 m). En promedio nos da como resultado un error de 0,53 m entre ambas coordenadas, por tanto, estamos ante una precisión de estimaciones muy parecida a la obtenida en el entorno A. A continuación, se muestra la salida por

consola del script del modelo entrenado con los datos del entorno B, junto con el fichero donde se han imprimido las posiciones reales y las estimadas.

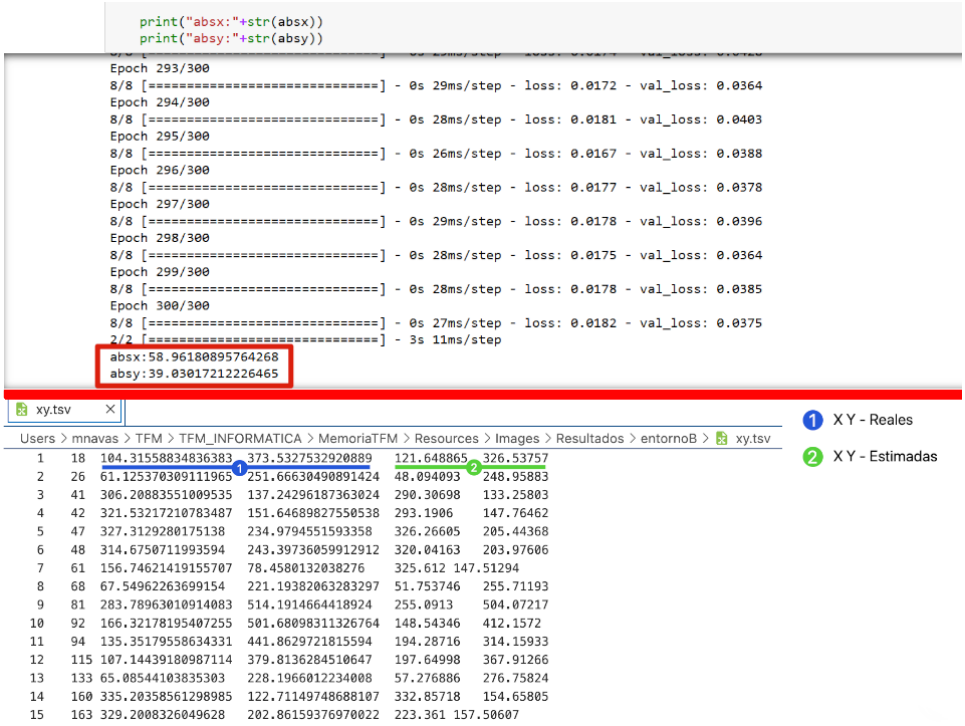


Figura 43. *Error absoluto por coordenadas (arriba) para los datos del entorno B y fichero xy.tsv (abajo) donde se almacenan las coordenadas reales junto a las estimadas por el modelo.*

A continuación, se muestran también un par de capturas de pantalla de la representación visual de los puntos reales y los estimados por el modelo para el entorno B.

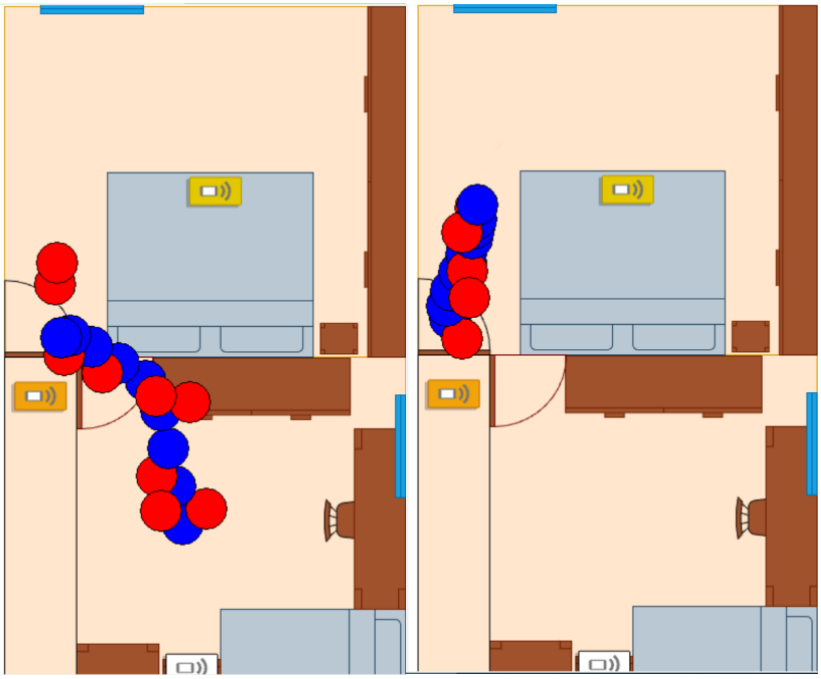


Figura 44. *Ejemplos de representación gráfica de las coordenadas reales (azules) y estimadas por el modelo (rojas) para el entorno B.*

4.3 Discusión

Los resultados de la evaluación de este modelo de Deep Learning para localización en interiores basado en sensores ambientales UWB son muy prometedores, ya que los errores absolutos obtenidos tanto en el entorno A como en el B son relativamente bajos para contar únicamente con tres balizas en áreas de unos 150 m2 en promedio. Tanto en el entorno A como en el B se han obtenido errores absolutos muy próximos a los 50 cm, algo muy satisfactorio dado el reducido despliegue necesario de dispositivos UWB utilizados, que tiene un impacto más reducido en energía y recursos que otras soluciones clásicas de UWB.

Como ya se adelantaba en la sección anterior, se ha conseguido demostrar que, a pesar de haber realizado unas pruebas de menor duración y cantidad de muestras de datos desde las balizas en el entorno B, se ha conseguido obtener unos resultados muy parecidos a los del entorno A. Esto nos indica que este sistema no se ve limitado por la calidad de los conjuntos de datos de entrada y puede ser empleado en situaciones o localizaciones donde las condiciones no sean las óptimas. Por ejemplo, en aquellas localizaciones que presenten una importante cantidad de obstáculos o donde no sea posible conseguir una buena disposición para ubicar las balizas UWB. Es importante señalar que ambos entornos se evaluaron mediante validación cruzada

con 10 iteraciones, lo que garantiza que los resultados son sólidos y no sólo el resultado de un conjunto de entrenamiento específico. En la siguiente tabla se muestra un resumen con los errores absolutos obtenidos mediante la validación cruzada.

Entorno	Error absoluto		
	X	Y	Promedio
A	0,53 m	0,50 m	0,515 m
B	0,64 m	0,43 m	0,535 m

Tabla 4. *Resumen de errores absolutos en la estimación por entorno, coordenadas y promedio*

En general, los resultados de la evaluación indican que este modelo de Deep Learning es un enfoque prometedor para la localización en interiores basada en datos obtenidos de sensores ambientales UWB. Para poder garantizar la escalabilidad y despliegue de forma óptima en cualquier entorno, son necesarias más pruebas y validaciones en diferentes entornos y escenarios para evaluar completamente las capacidades y limitaciones del modelo, así como para identificar cualquier área potencial de mejora.

Las balizas de Estimote han mostrado un buen rendimiento en su relación calidad precio, permitiendo disponer de una precisión de señal/distancia adecuada para poder reconstruir la localización en la complejidad de espacios interiores. A su vez, la integración de dispositivos móviles y ambientales se ha presentado en una arquitectura flexible que permite la multi ocupación y facilita el despliegue y el etiquetado de muestras que resulta vital en la técnica fingerprinting.

5 Conclusiones y trabajos futuros

La integración de balizas ambientales con tecnología de comunicación inalámbrica UWB junto un modelo de Deep Learning ha demostrado ser una solución viable y eficaz para conseguir una estimación lo bastante precisa de la posición en interiores para determinadas aplicaciones, sobre todo teniendo en cuenta las características que se han destacado en cuanto al coste, autonomía y escalabilidad de las balizas seleccionadas, las cuales dotan a esta propuesta de un mayor interés por sus posibilidades de uso en la vida real. Los resultados de las evaluaciones de validación cruzada que se han realizado en esta propuesta han obtenido de media un error absoluto medio aproximado de 0,50 m, un resultado muy satisfactorio para diversas aplicaciones como la localización de personas mayores o dependientes en entornos como centros de día o residencias, una de las aplicaciones más demandadas de este tipo de sistemas inteligentes. A pesar de los resultados satisfactorios obtenidos en este estudio, aún existen áreas de mejora que pueden ser exploradas en trabajos futuros. Uno de los aspectos más interesantes en este sentido sería el estudio de la aplicación del sistema en otros escenarios reales como los que se ha comentado anteriormente, para evaluar su rendimiento en situaciones de la vida real e identificar cualquier problema potencial que deba abordarse.

En cuanto a las posibles mejoras, una de las más obvias es aumentar el número de balizas UWB utilizadas en el sistema para mejorar la precisión de la estimación de la posición, además el sistema propuesto está preparado para ser escalable por lo que la integración de un mayor número de balizas Estimote sería automática. Además, en cuanto al modelo de Deep Learning, podría optimizarse para intentar reducir aún más el error. Otro aspecto a considerar es la prueba del sistema en otros tipos de entornos de prueba, ya que los dos entornos A y B que se han estudiado en este Trabajo de Fin de Máster han sido elegidos y configurados buscando la solución más óptima posible en cuanto a factores importantes como la cantidad de obstáculos o la localización de las balizas UWB con el fin de obtener los mejores resultados posibles, pero también podrían realizarse pruebas en entornos con otras características, como aquellos que puedan presentar una mayor cantidad de obstáculos, unas paredes más gruesas o mayor distancia entre las balizas, para evaluar su rendimiento en condiciones más adversas. Esto ayudaría a determinar la robustez del sistema y su

capacidad para manejar diversas condiciones en interiores, aunque los resultados obtenidos dan lugar a intuir que el sistema sería igualmente fiable en este tipo de condiciones. También podría investigarse la integración de otros sensores, como cámaras o acelerómetros, para mejorar el rendimiento general del sistema. Estos sensores podrían proporcionar información complementaria al modelo de Deep Learning que podría utilizarse para mejorar la precisión de la estimación de la posición. Además para mejorar la usabilidad del sistema, una vez superado el proceso de entrenamiento del modelo Deep Learning se podría trasladar la aplicación móvil a un dispositivo wearable como el Apple Watch, lo cual haría que a partir de ese momento sería prescindible el dispositivo móvil y la monitorización del usuario sería más cómoda por las características de este tipo de wearables.

En conclusión, este estudio ha servido para poner de manifiesto que el Deep Learning y los sensores ambientales UWB tienen un gran potencial para su uso en sistemas de estimación de posición en interiores. Además, en esta propuesta se ha añadido como novedad la inclusión de un dispositivo muy normalizado en la sociedad actual como son los dispositivos móviles en forma de herramienta de comunicación y recolección de datos UWB, lo que proporciona más facilidades de integración y uso por parte de los futuribles usuarios de este sistema. La integración de sensores adicionales, la inclusión de otros tipos de sensores ambientales, las pruebas en entornos interiores más hostiles y el despliegue en escenarios con usuarios reales son vías prometedoras para futuros trabajos de mejora del sistema.

La proliferación de dispositivos con UWB también se espera con alta expectativas de aumentar el impacto de este trabajo. En concreto la integración de Estimote en Android permitiría desplegar el modelo de Deep Learning en el dispositivo móvil, computando la localización en la capa Edge donde se recoge la propia señal y aumentando así las consideraciones de privacidad de los sistemas de localización de interiores.

Estas líneas abiertas esperan poder abordarse en un futuro proyecto de investigación que permita recolectar datos en diferentes domicilios, contando con datos de vida cotidiana y usando wearables y balizas localización que relacionen las

actividades y zonas de habitabilidad con los perfiles de los usuarios y su estado de salud.

6 Referencias

1. Alarifi, A., Al-Salman, A., Alsaleh, M., Alnafessah, A., Al-Hadhrami, S., Al-Ammar, M. A., & Al-Khalifa, H. S. (2016). Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. Sensors, 16(5), Article 5. https://doi.org/10.3390/s16050707

2. Bagaric, J. (2016). Indoor positioning system for mobile devices using radio frequency and perfect sequences [MasterThesis]. https://iconline.ipleiria.pt/handle/10400.8/2309

3. Barral, V., Rodas, J., & Escudero, C. J. (2019). Performance of a UWB location system in an indoor environment.

4. Barua, B., Kandil, N., Hakem, N., & Zaarour, N. (2017, July 14). Indoor localization with UWB and 2.4 GHz bands. https://doi.org/10.1109/APUSNCURSINRSM.2017.8072744

5. Basiri, A., Lohan, E. S., Moore, T., Winstanley, A., Peltola, P., Hill, C., Amirian, P., & Figueiredo e Silva, P. (2017). Indoor location based services challenges, requirements and usability of current solutions. Computer Science Review, 24, 1–12. https://doi.org/10.1016/j.cosrev.2017.03.002

6. Benini, A., Mancini, A., Frontoni, E., Zingaretti, P., & Longhi, S. (2011). Adaptive extended Kalman filter for indoor/outdoor localization using a 802.15.4a wireless network. 315–320.

7. Busch, P. A., Hausvik, G. I., Ropstad, O. K., & Pettersen, D. (2021). Smartphone usage among older adults. Computers in Human Behavior, 121, 106783. https://doi.org/10.1016/j.chb.2021.106783

8. Campaña Bastidas, S., Espinilla, M., & Medina, J. (2022, January 1). Review of Ultra Wide Band (UWB) for Indoor Positioning with application to the elderly. https://doi.org/10.24251/HICSS.2022.269

9. Caso, G., Le, M., De Nardis, L., & Di Benedetto, M.-G. (2018). Performance Comparison of WiFi and UWB Fingerprinting Indoor Positioning Systems. Technologies, 6, 14. https://doi.org/10.3390/technologies6010014

10. Cook, B., Buckberry, G., Scowcroft, I., Mitchell, J., & Allen, T. (2005). Indoor Location Using Trilateration Characteristics.

11. Correa, A., Barcelo, M., Morell, A., & Vicario, J. L. (2017). A Review of Pedestrian Indoor Positioning Systems for Mass Market Applications.

Sensors, 17(8), Article 8. https://doi.org/10.3390/s17081927

12. Cruz Alvarado, M. A., & Sandí Delgado, J. C. (2017). Sistemas y tecnologías que facilitan el posicionamiento Indoor. Pensamiento Actual, 17(29), 132–144.

13. de Rivaz, S., Denis, B., Pezzin, M., & Ouvry, L. (2007). Performance of IEEE 802.15.4a UWB Systems Under Multi-User Interference. 2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, 1–7. https://doi.org/10.1109/PIMRC.2007.4394401

14. Díaz Gonzalez, F. (2017). Intelligent indoor positioning system (Bachelor's thesis).

15. Donnelly, S., O'Brien, Begley, E., & Brennan, J. (2016). "I'd prefer to stay at home but I don't have a choice." Meeting Older People's Preference for Care: Policy, but what about practice?

16. Drutarovský, M., Kocur, D., Svecova, M., & Garcia, N. (2017). Real-time wireless UWB sensor network for person monitoring. 19–26. https://doi.org/10.23919/ConTEL.2017.8000034

17. Fowler, C., Entzminger, J., & Corum, J. (1990). Assessment of ultra-wideband (UWB) technology. IEEE Aerospace and Electronic Systems Magazine, 5(11), 45–49. https://doi.org/10.1109/62.63163

18. García Gutiérrez, R. (2016). Position estimation for IR-UWB systems using compressive sensing [Bachelor thesis, Universitat Politècnica de Catalunya]. https://upcommons.upc.edu/handle/2117/89818

19. Guashima, M., & Geovanny, O. (2013). LOCALIZACIÓN INDOOR MULTIMODAL. https://riunet.upv.es/handle/10251/33038

20. Gutiérrez Carrero, C. (2018, January). Seguimiento de objetos por UWB (Ultra Wide Band) [Info:eu-repo/semantics/bachelorThesis]. E.T.S. de Ingenieros Informáticos (UPM). https://oa.upm.es/49432/

21. Hoang, M. T., Yuen, B., Ren, K., Dong, X., Lu, T., Westendorp, R., & Reddy, K. (2020). A CNN-LSTM Quantifier for Single Access Point CSI Indoor Localization (arXiv:2005.06394). arXiv. https://doi.org/10.48550/arXiv.2005.06394

22. Hölldobler, S., Möhle, S., & Tigunova, A. (2017, June 16). Lessons Learned from AlphaGo.

23. Islam, A., Hossan, M. T., Chowdhury, M., & Jang, Y. M. (2018). Design of an

indoor positioning scheme using artificial intelligence algorithms. 953–956. https://doi.org/10.1109/ICOIN.2018.8343265

24. Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

25. Jiménez, A., & Seco, F. (2017, September 18). Finding objects using UWB or BLE localization technology: A museum-like use case. https://doi.org/10.1109/IPIN.2017.8115865

26. Karapistoli, E., Pavlidou, F.-N., Gragopoulos, I., & Tsetsinas, I. (2010). An overview of the IEEE 802.15.4a Standard. IEEE Communications Magazine, 48(1), 47–53. https://doi.org/10.1109/MCOM.2010.5394030

27. Kateliev, P. (2021, May 31). Apple AirTags review: The good, the bad, and the tiny. Phone Arena. https://www.phonearena.com/reviews/apple-airtags-review_id5072

28. Kelleher, J. D. (2019). Deep Learning (Illustrated edition). The MIT Press.

29. Kocur, D., Fortes, J., & Svecova, M. (2017). Multiple moving person tracking by UWB sensors: The effect of mutual shielding persons and methods reducing its impacts. EURASIP Journal on Wireless Communications and Networking, 2017, 68. https://doi.org/10.1186/s13638-017-0847-x

30. Lagunas, E., Navarro, M., Closas, P., & Najar, M. (2014, September 1). Spatial sparsity based direct positioning for IR-UWB in IEEE 802.15.4a channels. Proceedings - IEEE International Conference on Ultra-Wideband. https://doi.org/10.1109/ICUWB.2014.6958999

31. López Justicia, A. (2018). Indoor location system with multisensor device and temporary redundancy.

32. Lopez Medina, M. A., Espinilla, M., Paggetti, C., & Medina, J. (2019). Activity Recognition for IoT Devices Using Fuzzy Spatio-Temporal Features as Environmental Sensor Fusion. Sensors, 19. https://doi.org/10.3390/s19163512

33. Marquez, A. (2017). Implementation of an Autonomous Small-scale Car with Indoor Positioning using UWB and IMU. Electronic Theses and Dissertations. https://scholar.uwindsor.ca/etd/7277

34. Martínez García, A. (2010). Channel study for Ultra Wide Band systems.

35. Mazhar, F., Khan, M. G., & Sällberg, B. (2017). Precise Indoor Positioning

Using UWB: A Review of Methods, Algorithms and Implementations. Wireless Personal Communications, 97(3), 4467–4491. https://doi.org/10.1007/s11277-017-4734-x

36. McElroy, C., Neirynck, D., & Mclaughlin, M. (2014, June 14). Comparison of wireless clock synchronization algorithms for indoor location systems. https://doi.org/10.1109/ICCW.2014.6881189

37. Pabón Dueñas, A. B. (2017). Indoor navigation network for techno accessibility.

38. Pereira Tapiro, M., & Polo Poveda, W. A. (2015). Diseño e implementación de un prototipo de un sistema de localización en espacios cerrados (indoor). instname:Universidad Autónoma de Occidente. http://hdl.handle.net/10614/7972

39. Polo-Rodriguez, A., & Medina-Quero, J. (2022). Discriminating sensor activation in activity recognition within multi-occupancy environments based on nearby interaction (arXiv:2211.10355). arXiv. https://doi.org/10.48550/arXiv.2211.10355

40. Pozyx Academy | How positioning works. (n.d.). Retrieved January 28, 2023, from https://www.pozyx.io/pozyx-academy/how-does-positioning-work

41. Puyol, R. (2012). Grandes líneas de cambio demográfico. Nueva revista de política, cultura y arte, 140, 5–12.

42. Regueiro Senderos, C. (2014). Error en el posicionamiento indoor en dispositivos móviles [Master thesis, Universitat Oberta de Catalunya]. https://recercat.cat//handle/2072/238132

43. Sachs, J. (2013). Handbook of Ultra-Wideband Short-Range Sensing: Theory, Sensors, Applications. John Wiley & Sons.

44. Silva, B., & Hancke, G. P. (2017). Characterization of non-line of sight paths using 802.15.4a. 1436–1440. https://doi.org/10.1109/ICIT.2017.7915576

45. Suárez Páez, J., & Llano Ramírez, G. (2010). Revisión del estado del arte deIR-Ultra-Wideband y simulación de la respuesta impulsiva del canal IEEE802.15.4a. 6.

46. Tariq, Z. B., Cheema, D. M., Kamran, M. Z., & Naqvi, I. H. (2017). Non-GPS Positioning Systems: A Survey. ACM Computing Surveys, 50(4), 57:1-57:34. https://doi.org/10.1145/3098207

47. Tiemann, J. (2016). Wireless Positioning based on IEEE 802.15. 4a Ultra-

Wideband in Multi-User Environments. Technical Report for Collaborative Research Center SFB 876 Providing Information by Resource-Constrained Data Analysis.

48. Tiemann, J., & Wietfeld, C. (2017). Scalable and precise multi-UAV indoor navigation using TDOA-based UWB localization. 1–7. https://doi.org/10.1109/IPIN.2017.8115937

49. TIOBE Index. (n.d.). TIOBE. Retrieved January 31, 2023, from https://www.tiobe.com/tiobe-index/

50. Trevisan Troche, D. (2017). Influence of the presence of people in indoor positioning systems via Wi-Fi fingerprinting.

51. World Population Ageing 2020 Highlights. (2020). United Nations. https://www.un.org/development/desa/pd/sites/www.un.org.development.desa.pd/files/files/documents/2020/Nov/undesa_pd-2020_world_population_ageing_highlights.pdf

52. World Population Prospects 2022. (2022). United Nations. https://www.un.org/development/desa/pd/sites/www.un.org.development.desa.pd/files/wpp2022_summary_of_results.pdf

53. Yao, L., Wu, Y.-W. A., Yao, L., & Liao, Z. Z. (2017). An integrated IMU and UWB sensor based indoor positioning system. 2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 1–8. https://doi.org/10.1109/IPIN.2017.8115911

54. Zafari, F., Gkelias, A., & Leung, K. K. (2019). A Survey of Indoor Localization Systems and Technologies. IEEE Communications Surveys & Tutorials, 21(3), 2568–2599. https://doi.org/10.1109/COMST.2019.2911558

55. Zagoruyko, S., & Komodakis, N. (2017). Wide Residual Networks (arXiv:1605.07146). arXiv. https://doi.org/10.48550/arXiv.1605.07146

56. Zhang, F., & Feng, J. (2017). High-Resolution Mobile Fingerprint Matching via Deep Joint KNN-Triplet Embedding. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1), Article 1. https://doi.org/10.1609/aaai.v31i1.11088