ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

Estudio de la aplicación de Machine Learning a técnicas de posicionamiento en interiores

(Study of the application of Machine Learning to indoor positioning techniques)

Para acceder al Título de

Máster Universitario en Ingeniería de Telecomunicación

Autor: Javier Gómez Ortiz

Octubre - 2018

Agradecimientos

"Los finales no se pueden prever,
simplemente suceden"
Javier Krahe.

Debo agradecer, en primer lugar, el inquebrantable apoyo de mi familia. De mis abuelos que aún hoy me acompañan cada día y que les habría encantado ver hasta dónde hemos llegado. Debo agradecer la infinita preocupación de mis padres y la acomplejante diligencia de mi hermano.

Debo agradecer, por supuesto, a mis compañeros, a los amigos que, sin duda, hacen parecer corto el camino.

Finalmente agradezco a la escuela pública, a la que mis padres confiaron mi educación. Y, por supuesto, a sus trabajadores. Me gustaría, en concreto, reconocer la ayuda de Alberto en este trabajo, en el que no dudó en adentrarse a pesar de ser un tema nuevo para ambos.

Gracias a todos.

Resumen

En este trabajo se utilizan las técnicas de aprendizaje autónomo como solución al problema de la localización en interiores. Este problema ha sido estudiado previamente en numerosos proyectos, siguiendo soluciones consideradas como tradicionales, pero sin conseguir aportar resultados satisfactorios. Es por ello que, tras realizar un estudio de las diferentes técnicas de inteligencia articial y aprendizaje autónomo que existen actualmente, se ha seleccionado aquella que aporta mejores resultados. El método así propuesto ha sido probado haciendo uso del conjunto de datos y resultados obtenidos mediante técnicas clásicas de trilateración. El análisis de los resultados obtenidos permite asegurar una mejora en las estimaciones de posición, además de otras ventajas desde el punto de vista de despliegue e implementación.

Palabras clave: Aprendizaje autónomo, redes neuronales, inteligencia artificial, conjunto de datos, entrenamiento, evaluación.

Abstract

In the present paper, machine learning techniques are used as a solution to the indoor location problem. This same problem has been studied previously in many projects, following solutions considered as traditional, but without achieving any satisfactory result. Consequently, after researching the different artificial intelligence and machine learning techniques that currently exist, the one that provides the best results has been selected. The method proposed has been tested using the dataset and results obtained by classical trilateration techniques. The analysis of the obtained results ensures an improvement in the indoor positioning, as well as other advantages from the point of view of deployment and implementation.

Keywords: Machine learning, neural networks, artificial intelligence, dataset, training, testing.

Índice

1 Introducción **11**
1.1 Motivación 11
1.2 Objetivos 13

2 Introducción al problema **14**
2.1 Funcionamiento del posicionamiento en exteriores 14
2.2 Técnicas tradicionales de posicionamiento en interiores 15

3 Inteligencia Artificial y *Machine Learning* **18**
3.1 Inteligencia Artificial Estrecha 18
3.2 Inteligencia Artificial Amplia 19
3.3 Machine Learning o Aprendizaje Autónomo 19
3.4 Aprendizaje no supervisado 21
3.5 Aprendizaje por refuerzo 22
3.6 Aprendizaje Supervisado 24

4 Modelos de Aprendizaje Autónomo **26**
4.1 Algoritmos genéticos 26
4.2 Redes Neuronales 27

5 Desarrollo Práctico **29**
5.1 Escenario de partida 29
5.1.1 Análisis de Resultados 32
5.2 Procesamiento de los datos 35
5.3 Entorno de desarrollo de Redes Neuronales 37
5.3.1 Tensorflow 38
5.3.2 Keras 39
5.4 Funcionamiento de la Red Neuronal 39
5.5 Resultados 45
5.5.1 Resultados Primer Escenario 45
5.5.2 Resultados Segundo Escenario 46
5.6 Análisis de un Nuevo Escenario 51
5.6.1 Uso de Beacons WiFi 51
5.6.2 Uso de Beacons BlueTooth 53
5.6.3 Uso Combinado Beacons WiFi y BlueTooth 57

6 Conclusiones y líneas futuras **59**
6.1 Conclusiones 59
6.2 Líneas futuras 59

Referencias **61**

Índice de figuras

1 Pasillo Planta -2, edificio I+D+i 29
2 Sala Multiusos, edificio I+D+i 30
3 Esquema primer escenario, Pasillo 31
4 Esquema segundo escenario, Sala 32
5 Esquema primer escenario, Pasillo 33
6 Esquema segundo escenario, Sala 34
7 Flujo de Trabajo 40
8 Evolución de la tasa de acierto 47
9 Evolución de la función de pérdida 47
10 Distribución de los errores 48
11 Matriz de Confusión del escenario 2 50
12 Escenario Planta -2 51
13 Evolución de la tasa de acierto con Beacons WiFi 52
14 Evolución de la función de pérdida con Beacons WiFi 53
15 Evolución de la tasa de acierto en el escenario 3 con beacons BlueTooth . . 54
16 Evolución de la función de pérdida en el escenario 3 con beacons BlueTooth 54
17 Matriz de Confusión del escenario 3 con beacons BlueTooth 56
18 Matriz de Confusión del escenario 3 con beacons WiFi y BlueTooth 58

Índice de cuadros

1 Ejemplo estructura de los datos 33
2 Ejemplo estructura de los datos 35
3 Resultados Primer Escenario 45
4 Resultados Segundo Escenario 46

1 Introducción

El 4 de agosto de 1957 la Unión Soviética consiguió, por primera vez en la historia de la humanidad, poner un satélite artificial, el *Sputnik I*, en la órbita terrestre. Con este hito comenzó lo que se denominó la "carrera espacial", que enfrentó a las dos grandes potencias del pasado siglo y que, pese a culminar con la llegada de la misión *Apolo XI* a la superficie lunar, inició un nuevo campo de investigación e innovación que aún hoy sigue explorándose.

Esta era espacial ha traído consigo numerosos avances en múltiples campos, desde la astrofísica hasta la botánica. Otro ejemplo de estos avances que, no sólo han experimentado las agencias espaciales o militares, sino que han transcendido al ámbito civil son los dispositivos de posicionamiento global que comienzan a desarrollarse en la década de los sesenta, y que progresivamente fueron llegando al uso comercial. Sin embargo, no es hasta su incorporación en los teléfonos inteligentes cuando su uso realmente se democratizó.

Actualmente dichos dispositivos posibilitan no sólo localizar la posición actual de un usuario, o establecer la mejor ruta en tiempo real hacia un destino dado, sino que se usan en una multiplicidad de campos, desde el ocio hasta las actividades deportivas, bien por parte de aficionados o por deportistas de élite.

Dichos avances y tecnologías, sin embargo, no han sido capaces de facilitar, hasta la fecha, un sistema de posicionamiento en interiores fiable y eficaz que podría tener también múltiples usos y finalidades, ya fueran comerciales, turísticas o de gestión y prevención de riesgos.

Algunas de las aplicaciones prácticas que están a la espera del desarrollo de tecnologías de posicionamiento en interiores podrían ser la localización de personas o vehículos en grandes centros comerciales, museos o centros polideportivos, la coordinación de actividades grupales, la obtención de datos de flujos de tráfico humano para su posterior explotación o detectar y trasladar el movimiento realizado a videojuegos de realidad virtual.

1.1 Motivación

Tradicionalmente se ha intentado abordar el problema de la localización en interiores mediante el posicionamiento de balizas que emitieran una señal con una potencia conocida desde una ubicación, también conocida, para así tratar de obtener la posición del usuario mediante técnicas de trilateración. Estas técnicas, al igual que otras soluciones han resultado infructuosos por diferentes motivos. Generalmente no se llegan a modelos lo

suficientemente precisos ya que el entorno es extremadamente adverso. Dicha adversidad se debe a las múltiples reflexiones, refacciones y demás efectos indeseados que sufren las señales electromagnéticas. Sin embargo, aunque se consiguiera una precisión aceptable, en términos prácticos seguirían sin ser viables, ya que dichos algoritmos requieren de una infraestructura no siempre portable. Además, con asiduidad, estos algoritmos provocan un alto consumo energético, inviable para los dispositivos móviles, ya que requieren mucha complejidad computacional y acceder en todo momento al receptor, bien sea WiFi o BlueTooth.

Los dispositivos móviles actuales cuentan con una gran cantidad de sensores que permiten la obtención de una ingente cantidad de datos. Además el exponencial aumento de la potencia de cálculo que han experimentado dichos dispositivos en la última década, así como el uso de aplicaciones web, hacen posible la gestión y análisis de dichos datos, y permiten la exploración de nuevas técnicas que mitigan los efectos anteriores.

Así mismo, en los últimos años, se ha experimentado un auge en las técnicas de aprendizaje autónomo y de la inteligencia artificial. Numerosas empresas están invirtiendo sus esfuerzos, su tiempo y su dinero, en la investigación de dichas tecnologías que permiten hallar soluciones a problemas complejos, reduciendo los requisitos computacionales.

Partiendo de estas premisas, en este trabajo se pretende estudiar esta nueva era de algoritmos de aprendizaje autónomo, para aplicarlos sobre un conjunto de datos previamente obtenidos, y así evaluar si resulta posible, y práctica su implementación, en este caso, para solucionar el problema del posicionamiento en interiores.

El factor clave a la hora de determinar el desempeño del algoritmo diseñado será la tasa de acierto que consiga en la etapa de evaluación tras haber sido entrenado. Esta tasa de acierto se comparará con la obtenida a través de diferentes técnicas que podrían entenderse como tradicionales, como puede ser la trilateración.

Al tratarse de algoritmos mucho más refinados y potentes se espera que el resultado sea significativamente mejor que con las técnicas tradicionales.

1.2 Objetivos

Teniendo todo lo anterior en cuenta, se procede a detallar los objetivos de partida del presente trabajo.

En primer lugar, partiendo del desconocimiento de las técnicas de *machine learning* se pretende realizar una aproximación a las mismas para que puedan ser aplicadas en el ámbito de este trabajo. Para ello se investigan sus diferentes vertientes, para así decidir qué modelo o algoritmo se ajusta mejor a las necesidades propias de la localización en interiores.

Por otro lado, se parte de un estudio previo [1] en el que se aplican técnicas simples de trilateración para intentar resolver el posicionamiento en interiores. Haciendo uso de los mismos datos se espera mejorar de manera significativa los resultados existentes.

Finalmente, tras la verificación teórica de los resultados se pretende realizar un nuevo estudio de campo, con nuevas medidas que permitan evaluar el uso combinado de señales WiFi y BlueTooth, o si por el contrario existe una clara comparativa entre ambas tecnologías.

2 Introducción al problema

En el presente capítulo se define cuál es el estado del arte en los sistemas de posicionamiento globales, y se analizan los motivos por los cuales dichas soluciones no son extrapolables al posicionamiento en interiores.

2.1 Funcionamiento del posicionamiento en exteriores

Los sistemas de posicionamiento global actuales, surgen en el apogeo de la carrera espacial. Como tantos otros avances, dichos sistemas, nacieron impulsados por motivaciones bélicas, la Guerra Fría en este caso. Tanto los Estados Unidos como la Unión Soviética, desarrollaron sus propios sistemas de posicionamiento, lo que finalmente fue el *GPS*[2] en el caso del primero; y lo que se denominó *GLONASS* en el caso de la URSS. Al ser el *GPS* el más conocido, y ya que ambos funcionan de manera similar, en este documento se describe el funcionamiento del *GPS*.

El sistema de posicionamiento global, GPS, es capaz de determinar la posición de un objeto, persona o vehículo, en cualquier lugar de la Tierra. Para ello hace uso de una red de satélites que orbitan el planeta en una órbita inferior a la geoestacionaria. Para poder dar cobertura universal, se cuenta con un número de satélites, entre los 24 y los 32, que orbitan la Tierra con un periodo de aproximadamente 12 horas.

Para poder determinar su posición, un equipo debe detectar al menos cuatro satélites, de los cuales obtendrá información temporal así como tramas de datos. El equipo usará esta información temporal para sincronizar su reloj interno, y así dilucidar cuánto tiempo tardan en llegar las señales de los distintos satélites. Una vez conocidas dichas latencias, se utiliza una técnica de trilateración para determinar la distancia desde el equipo a cada uno de los satélites logrando así determinar la posición en tres dimensiones.

Si bien este sistema es robusto y funciona a la perfección en exteriores, llegando a precisiones del orden de los centímetros, se torna inservible cuando se pretende utilizar en interiores. Esto es debido a diferentes factores. En primer lugar la señal recibida de los diferentes satélites, ya débil de por sí debido a la elevada distancia que los separa del receptor, se ve atenuada aún más por las paredes, cristales y demás mobiliario. A esto se suman las reflexiones y refracciones que sufren las señales y, por si fuera poco, no todas las señales de los satélites se ven afectadas de igual manera, lo que penaliza en exceso el comportamiento del sistema.

2.2 Técnicas tradicionales de posicionamiento en interiores

Siendo inviable la utilización de los sistemas actuales de posicionamiento global, se podría pensar en el desarrollo de infraestructuras internas que funcionen de manera análoga al GPS, que tan buenos resultados da en exteriores.

Para ello se deberían tener ciertas baliza, en lugares concretos que emitieran, además de su posición, información temporal. El dispositivo capta las señales de diferentes balizas pero, por desgracia, es incapaz de determinar una latencia significativa, ya que la señal llega prácticamente de manera instantánea debido a las reducidas distancias y a la velocidad de las ondas electromagnéticas.

Además, estas señales pueden no llegar en línea recta, sino tras múltiples reflexiones, lo que representa un aumento significativo de la latencia y por tanto un error a la hora de determinar la posición.

Si bien queda descartada la solución de adaptar las técnicas utilizadas por los sistemas GPS para el posicionamiento en exteriores, las técnicas de trilateración, al menos en teoría, son una alternativa, ya que tienen en cuenta la atenuación que sufre la potencia emitida o recibida y no el retraso temporal.

Suponiendo que el medio es isótropo y que no existe ninguna razón por la que una señal emitida en cierta frecuencia se atenúe más que otra emitida en esa misma frecuencia, se puede suponer que, si se tienen balizas situadas en el interior de un edificio, un equipo que detecte dichas señales debe ser capaz de determinar su posición en función de la distancia a cada una de las balizas. Para determinar la distancia, en este caso se utiliza, como se ha mencionado, no la latencia temporal, sino la atenuación detectada.

Este tipo de aproximación se ha estudiado en profundidad en anteriores Trabajos Fin de Grado [3] [4], así como en diferentes publicaciones. Los resultados, sin embargo, no resultan todo lo satisfactorios que cabría esperar. En el presente documento se ha definido este tipo de técnicas como el estado del arte del posicionamiento en interiores debido a su fácil implementación y al acceso a estudios anteriores. Por lo tanto son estos resultados los que se pretenden superar.

Alternativamente, en los últimos años han surgido nuevas técnicas de posicionamiento en interiores. Con el auge de las tecnologías de realidad virtual y realidad aumentada se han redoblado los esfuerzos por desarrollar sistemas capaces de captar la posición y el movimiento de equipos, para así trasladarlos a los videojuegos. Una de estas experiencias

fue probada en un TFG anterior [5], con resultados totalmente dependientes de la robustez del posicionamiento.

Quizá no es el ejemplo más famoso de realidad virtual, pero sin duda el *"HTC Vive"* [6] ha conseguido uno de los mejores sistemas de seguimiento del movimiento. Para conseguirlo tanto las gafas de realidad virtual, como los controles que se usan con las manos, cuentan además de con acelerómetro y giroscopio, con una serie de sensores ópticos capaces de captar la luz infrarroja que emiten las denominadas *"sensor boxes"*. Estas cajas se deben situar en esquinas opuestas de la sala en la que se desea experimentar con la realidad virtual, de forma que iluminen todo el espacio.

Una vez preparada la sala y dispuestas las *sensor boxes*, estas actúan como faros emitiendo pulsos de luz infrarroja con una frecuencia de 60 pulsos por segundo. Tras cada pulso, un láser hace un barrido por la sala como si de un radar se tratase. Las gafas y controles, por su parte, detectan cuándo son iluminadas por la luz infrarroja, y comienzan un contador que se detiene cuando detectan el láser. Trabajando con estos datos el sistema es capaz de determinar la posición del usuario con gran precisión, lo que permite transmitir fidedignamente su movimiento a la realidad virtual.

Este tipo de aproximaciones, aunque son realmente precisas, no resultan prácticas a gran escala ya que se debe implementar una gran infraestructura, tanto por parte del edificio, que se debería equipar con las *sensor boxes*, como de los posibles usuarios, que deberían llevar encima un equipo receptor capaz de detectar la luz infrarroja y los láseres. Por si fuera poco, estos sistemas ven gravemente perjudicado su funcionamiento cuando aparecen objetos inesperados en la sala. Queda, por lo tanto, descartada esta solución.

Otra aproximación al problema es la tomada por Google en sus sistemas Android. Cualquiera que tenga un dispositivo con dicho sistema operativo habrá recibido notificaciones, cuando está en un centro comercial, por ejemplo, del establecimiento en el que ha parado a comprar, o de la cafetería en la que se ha sentado a tomar algo. Si bien no es público el funcionamiento concreto del algoritmo de posicionamiento usado por Google, de algún modo tiene que hacer uso de las huellas que dejan las señales WiFi. Por lo tanto, sabrá posicionarte en cierto establecimiento según la señal WiFi que detecte con más potencia, ya que presumiblemente será la del local en cuestión.

Este sistema resulta eficaz para los intereses de Google: vender anuncios de locales o establecimientos que frecuenta el usuario, y notificarle cuando se encuentre cerca de uno de ellos. Pero no es el posicionamiento al que se pretende aspirar, que sería el análogo del GPS para espacios interiores. Esta solución resulta interesante sin embargo ya que no

exige ningún tipo de infraestructura ni por parte de los establecimientos ni del usuario. Tan solo hace uso de las señales que se encuentran y de un teléfono inteligente.

Se ahondará, por tanto, en este tipo de soluciones, que se podrían entender como pasivas, ya que se limitan a "escuchar" las señales del entorno y procesarlas para tratar de obtener información valiosa. En este trabajo se propone hacer uso de algoritmos de inteligencia artificial y de aprendizaje autónomo para intentar extraer la información deseada de las señales captadas.

3 Inteligencia Artificial y *Machine Learning*

Recientemente los términos de inteligencia artificial y *machine learning* han cobrado gran relevancia, tanto a nivel mediático como académico, con la publicación de multitud de estudios y artículos científicos. Este gran auge de la inteligencia artificial viene, sin duda, impulsado por el aumento de la potencia computacional, así como por la popularización de lenguajes de programación y la aparición y mejora de *frameworks* orientados a tal fin.

Antes de comenzar a desarrollar dichos conceptos y sus diferentes ramificaciones, se debe definir el concepto de inteligencia artificial. Se entiende por inteligencia artificial la simulación, por parte de algoritmos, de procesos inherentes a la mente humana, principalmente aprendizaje y razonamiento.

A partir de esta definición surgen dos clasificaciones de inteligencia artificial, cuyos términos en inglés son *Soft Artificial Intelligence* y *Hard Artificial Intelligence*. Una traducción literal sería inteligencia artificial fuerte y floja, aunque en este documento se referirá a ellas como inteligencia artificial estrecha y amplia o completa.

3.1 Inteligencia Artificial Estrecha

Dentro de esta clasificación se encuentran todos aquellos algoritmos de inteligencia artificial programados para resolver una tarea concreta o un conjunto de ellas. Es decir, todos los algoritmos actualmente más conocidos y que se utilizan en el día a día, tanto por las grandes empresas como por usuarios particulares, para: reconocimiento de imágenes, clasificación, sugerencias basadas en gustos, reconocimiento de voz, y un largo etcétera. Básicamente, todos los avances realizados en el campo de la inteligencia artificial hasta el momento se limitan a esta clasificación.

Es importante destacar, cuando se habla de estos algoritmos, que su funcionamiento se basa en el procesado de unos datos de entrada para generar la salida deseada y, lo que es más importante, que no son capaces de realizar ninguna otra función que no sea procesar dichos datos. La *inteligencia* por tanto, reside en el periodo de aprendizaje.

Durante el periodo de aprendizaje o la etapa de entrenamiento, a los algoritmos se les proporciona unos datos de entrada así como la salida que deben generar dichos datos. Con esta información el algoritmo ha de ser capaz de *aprender*, es decir, encontrar la función matemática que mejor se ajuste para que, tras concluir el entrenamiento, cuando se proporcionen datos reales, este algoritmo pueda encontrar una salida de acuerdo con

las expectativas. Existen diferentes problemas en esta etapa que se deben tener en cuenta y que se abordan en apartados posteriores.

3.2 Inteligencia Artificial Amplia

Una inteligencia artificial amplia es toda aquella inteligencia artificial que no está restringida a un campo de acción, sino que es capaz de razonar, pensar o aprender, de manera análoga a como lo hace una persona. En teoría, es aquella que puede incluso tomar consciencia de sí misma, y es por ello que se denomina singularidad al punto en el que una inteligencia artificial completa es capaz de aprender a mejorarse a sí misma, aumentando así, sus capacidades de manera exponencial.

No existe hasta el momento ninguna constancia de que esto sea posible o viable en un futuro próximo. Existe la creencia de que con la llegada de la computación cuántica se podría llegar a avances en estos campos, pero no se sabe nada con certeza.

Una de las ideas que está sobre la mesa es la simulación de un cerebro real. Es decir, se plantea que si se pudiera simular de manera fidedigna el comportamiento de un cerebro humano, quizá surgieran automáticamente dichos conceptos de consciencia, intuición o creatividad.

Como todo esto son suposiciones y se aleja bastante del ámbito de este trabajo, simplemente resaltar que de aquí en adelante, cuando aparezca el término inteligencia artificial, se estará refiriendo a la inteligencia artificial estrecha.

3.3 Machine Learning o Aprendizaje Autónomo

El *machine learning* o aprendizaje autónomo es una rama de la inteligencia artificial, que comprende todos aquellos algoritmos capaces de, a partir de ejemplos concretos, hallar reglas o patrones que resuelvan la generalidad del problema. Su uso es conocido en los motores de búsqueda, en los videojuegos o en ejemplos tan divergentes como la detección de fraude, reconocimiento de números y letras manuscritas, o predicciones basadas en gustos. Todos estos problemas, sin embargo, han sido definidos dentro de nueve grupos, tal y como se puede ver en [7], los cuales son: clasificación, regresión, identificación de similitudes, *"clustering"*, agrupación de co-ocurrencias, *"profiling"*, predicción de vínculos, reducción de datos y modelado causal.

Clasificación: Este conjunto de problemas tiene como objetivo determinar a qué grupo pertenece un dato o conjunto de datos basándose en lo que se ha aprendido previamente de datos anteriores. Un ejemplo práctico es un algoritmo capaz de dilucidar en qué grupo de consumo puede estar un usuario basándose en su edad, nacionalidad, lugar de residencia, etcétera. Con esos datos y comparándolo con usuarios anteriores, se le podría clasificar dentro de un grupo concreto.

Regresión: Los problemas de regresión son similares a los de clasificación, con la diferencia que, en este caso, la salida es un número y no una clase. Se enmarcarían en este subconjunto, por ejemplo, los algoritmos capaces de predecir el precio del alquiler de una vivienda basándose en diferentes variables.

Identificación de similitudes: El estado del arte de los sistemas de recomendación se basa en este tipo de problemas, en los cuales se pretende encontrar similitudes entre los usuarios, basándose en búsquedas previas, productos consumidos, etc.

Clustering: Similar al conjunto anterior, este tipo de problemas pretende agrupar datos o usuarios, aunque en este caso, sin un propósito concreto o definido.

Agrupación de co-ocurrencias: En este caso lo que se pretende es identificar datos que sucedan simultáneamente. Es interesante, por ejemplo, ser capaz de averiguar qué productos se compran a la vez para así crear una promoción. También se puede predecir un suceso basándose en otro que suele ocurrir de manera conjunta con este.

Profiling: Es posible estimar o predecir comportamientos o resultados en base a los datos previos. Los problemas de profiling tratan de resolver dichos problemas. Son muy utilizados por las entidades bancarias para evitar el fraude. Con este tipo de técnicas son capaces de predecir el comportamiento habitual de clientes para bloquear transacciones sospechosas, o que discuerdan con las predicciones previas.

Predicción de vínculos: Nuevamente estas técnicas son muy utilizadas en la actualidad. Se busca la conexión entre conjuntos de datos. El claro ejemplo es cuando una red social recomienda amistades a sus usuarios basándose en el conjunto de amigos que tienen en común.

Reducción de datos: Los datos son muy importantes en la actualidad, sin embargo, en ocasiones existen demasiados como para ser utilizados. Los algoritmos de reducción de datos pretenden llegar a un compromiso óptimo entre la reducción de los datos y la información perdida por dicha reducción. Este tipo de técnicas son imprescindibles debido

a la limitación de de los recursos frente al incremento del volumen de los datos.

Modelado Causal: Estas técnicas son capaces de encontrar relaciones de causalidad entre un conjunto de datos. Una empresa puede estar interesada en utilizar esta clase de algoritmos para evaluar si sus campañas de publicidad son efectivas o por el contrario no existe una relación causal entre los nuevos clientes y la publicidad.

Como se puede ver existen nueve subconjuntos que ni mucho menos son disjuntos entre sí. A la hora de definir el problema que se desea plantear, se debe orientar de una manera o de otra para enmarcarlo en uno de esos conjuntos, y así utilizar los algoritmos más apropiados.

Se puede simplificar esta clasificación en tan solo cinco enunciados a solucionar.

- Determinar si un dato o conjunto es de la clase A o de la clase B. Algoritmos de clasificación
- Identificar datos que no sean típicos o esperados. Algoritmos de detección de anomalías.
- Determinar un valor numérico en base a datos conocidos. Algoritmos de regresión.
- Determinar la estructura de los datos. Algoritmos de clustering.
- Determinar el comportamiento en base a datos. Algoritmos de aprendizaje por refuerzo.

Una vez vistos los diferentes tipos de problemas que se abordan con la inteligencia artificial y con los algoritmos de aprendizaje autónomo, se debe incidir en el hecho de que existen distintos tipos de algoritmos de aprendizaje autónomo, estos son: el aprendizaje no supervisado, el aprendizaje por refuerzo y el aprendizaje supervisado.

3.4 Aprendizaje no supervisado

En este tipo de algoritmos no se cuenta con datos que estén etiquetados, es decir, se desconoce el grupo a qué grupo pertenecen. Lo que se pretende, es encontrar ciertas estructuras que sean capaces de caracterizarlos. Este tipo de aprendizaje es muy utilizado

en los problemas de clustering, donde se deben agrupar los datos, aunque muchas veces dichas agrupaciones no tengan un significado real. Como ya es conocido, correlación no implica causalidad y en muchos casos se pueden dar falsos positivos, es decir, hallar estructuras entre datos que no infieran ninguna relación real. Sin embargo, en ocasiones se es capaz, utilizando estos algoritmos, de hallar relaciones que se pasarían por alto con técnicas tradicionales de análisis. Estos algoritmos también son útiles en la agrupación de co-ocurrencias y los problemas de profiling.

Desde el punto de vista de su aplicabilidad, estos algoritmos resultan especialmente sencillos de utilizar, ya que no requieren que personas cataloguen o etiqueten los datos, una tarea que puede ser especialmente pesada si se pretende hacer manualmente.

3.5 Aprendizaje por refuerzo

Si bien es cierto que los dos grandes métodos de aprendizaje son el supervisado y el no supervisado, existe un tercero que aun menos estudiado merece ser mentado. Este es el aprendizaje por refuerzo, donde se proporciona al algoritmo ciertos datos en el periodo de entrenamiento y, tras determinar una respuesta a dichos datos, recibe una retroalimentación o evaluación sobre cuán buena ha sido tal respuesta.

Este tipo de aprendizaje, que podría entenderse como similar al aprendizaje supervisado, difiere del mismo ya que no se le proporciona en ningún momento la respuesta correcta, por el contrario, simplemente se indica la distancia o error cometido.

Quizá los ejemplos más famosos de aprendizaje por refuerzo son los módulos de análisis de ajedrez. Recientemente la empresa, perteneciente a Google, dedicada a la investigación de inteligencia artificial, DeepMind [8], publicó el resultado de su proyecto *AlphaZero* [9]. En dicho documento se muestra cómo AlphaZero fue capaz, mediante algoritmos de aprendizaje por refuerzo, de llegar a dominar el ajedrez desde cero, es decir, con el único conocimiento previo de las reglas de juego. Llegando, por tanto, sin ningún tipo de ayuda humana, a jugar al ajedrez, el shogi y el Go de manera casi perfecta. Siendo capaz, no sólo de superar el nivel de los mejores jugadores humanos, algo que ya eran capaces de hacer algoritmos tradicionales, sino también superar a los más potentes módulos de análisis de cada juego.

Para entender cómo ha sido esto posible, se debe primero analizar cómo funcionan los módulos tradicionales, lo que aún hoy, se considera el estado del arte, ya que, hasta la fecha, DeepMind no ha hecho públicos sus algoritmos. Como los módulos de los distintos

juegos son semejantes, se procede a explicar únicamente el funcionamiento de *Stockfish* [10] el módulo de análisis de ajedrez más potente hasta la fecha y que es *OpenSource*.

Se considera que toda partida de ajedrez se divide en tres fases; en la primera de ellas, la apertura, el jugador ha de desarrollar sus piezas y enrocar su rey. Tras esto se llega al medio juego, donde se suceden las capturas e intercambios de piezas, lo que desemboca en la tercera fase, el final, donde ya quedan pocas piezas sobre el tablero. Partiendo de esta división tradicional, *Stockfish* juega la apertura con una extensa base de datos que contiene una infinidad de posiciones con sus correspondientes evaluaciones. En esta fase del juego el módulo procura seguir dicha base de datos hasta llegar a una posición de medio juego ventajosa.

Una vez en el medio juego, así como en el final, los sistemas de evaluación, creación del árbol de variables, y la poda del mismo cobran importancia.

En primer lugar, el sistema de evaluación comprende tanto los conceptos estratégicos como tácticos de la posiciónń que está siendo evaluada. Dicho sistema se programa y configura de forma manual o tradicional. Estas funciones de evaluación son por tanto lineales, y podrían ser superadas por redes neuronales profundas, que aportan más grados de libertad al problema. Por otra parte, a la hora de crear y explorar el árbol de jugadas se utiliza un algoritmo *minimax*, es decir buscar el movimiento que reporte mayor ganancia al jugador uno, sabiendo que el adversario va a elegir el movimiento que reporte la menor ganancia a dicho jugador. Para reducir el árbol de búsquedas se utiliza una poda *Alpha-Beta*[1].

Por el contrario, lo que consiguió DeepMind con AlphaZero, fue que tanto el sistema de evaluación de las posiciones, como la búsqueda y poda dentro del árbol de variables, no sólo se realizaran de manera autónoma, sino que el algoritmo aprendiera a calibrar dichos sistemas por si mismo basándose únicamente en las reglas de juego y del número típico de movimientos que dura una partida.

Con estos conocimientos, el algoritmo entrenó jugando contra sí mismo durante nueve horas y fue capaz de mejorar a partir del aprendizaje por refuerzo hasta superar, tras el periodo de entrenamiento, el nivel de cualquier Gran Maestro de cada una de las diferentes disciplinas: ajedrez, shogi y go. Para evaluar su nivel de juego, se enfrentó a *AlphaZero* contra *Stockfish*, derrotándolo claramente.

[1]Podado Alpha-Beta: `http://shelf2.library.cmu.edu/Tech/17700646.pdf`

Para concluir con este tipo de aprendizaje se debe mencionar que su gran ventaja es que resulta muy eficiente cuando no se conoce la respuesta correcta, ya sea en juegos extremadamente complejos como los mencionados anteriormente, o en otro tipo de problemas de complejidad *NP* y *NP-Hard*. Los algoritmos de aprendizaje por refuerzo son capaces, en estas situaciones, de encontrar soluciones subóptimas en un tiempo razonable tras superar la etapa de entrenamiento.

3.6 Aprendizaje Supervisado

El tercer y último método de aprendizaje autónomo que existe, y que se detalla a continuación, es el denominado aprendizaje supervisado. Poseyendo ya las nociones básicas del aprendizaje no supervisado, se puede describir como la antítesis de este. Es decir, aquellos algoritmos en los que, durante el periodo de entrenamiento, se proporcionan los datos de entrada junto con la salida que se debe generar. Cuando los datos van acompañados de la salida correcta se dice que los datos están etiquetados (o *labelled* en inglés).

El aprendizaje supervisado se torna muy útil en labores de predicción de resultados en base a datos históricos. Se puede utilizar, por poner un ejemplo, para clasificar los correos electrónicos y que se almacenen en la bandeja de entrada o por el contrario en la de no deseados. Para ello el algoritmo contaría con las direcciones de origen de los correos, los cuerpos de los mismos, asuntos y demás características, como por ejemplo, si el usuario calificó ese correo como no deseado. Con esos datos el algoritmo puede aprender del comportamiento del usuario, para así catalogar los correos electrónicos.

Este trabajo parte de un *Dataset*, o conjunto de muestras, que incluye los instantes temporales en que han sido tomadas dichas muestras, un identificador de la baliza emisora de la señal y la potencia recibida. Dichos datos se encuentran ya etiquetados, por lo que además de la información anterior, se cuenta con la posición en que se ha realizado cada captura.

En posteriores capítulos, cuando se detallen los aspectos prácticos de este trabajo, se profundizará más en la estructura del Dataset y en las modificaciones que se han realizado sobre el mismo para su posterior utilización.

Contando con ese conjunto muestral correctamente etiquetado, se ha optado por hacer uso del aprendizaje supervisado para el desarrollo del trabajo. La idea no dista demasiado del ejemplo del correo electrónico. Se dispone de unos datos que se clasifican en un total de 45 posiciones, desde la posición 0 a la posición 44. El objetivo del algoritmo es aprender

cómo clasificar los datos para que, una vez la etapa de entrenamiento o aprendizaje haya concluido, poder discernir la posición del usuario en base a las potencias recibidas de las distintas balizas.

Una vez decidido el tipo de aprendizaje que se va a emplear queda por decidir cómo ha de modelarse el algoritmo, ya que, al igual que pasa con los métodos de entrenamiento, existen diferentes modelos que permiten a las máquinas aprender de manera autónoma.

4 Modelos de Aprendizaje Autónomo

Quizá el modelo más conocido de aprendizaje autónomo es la red neuronal (*Neural Network* en inglés). Sin duda, esta herramienta es extremadamente potente y flexible, por lo que es capaz de adaptarse a multitud de problemas con gran tasa de éxito. Todo esto ha llevado a las redes neuronales a colocarse a la cabeza del modelado de problemas de aprendizaje autónomo. Debido a sus numerosas ventajas y a su gran polivalencia ha sido seleccionado en el presente trabajo, no sólo para abordar el problema de la localización en interiores, sino como ejemplo de uno de los múltiples usos que se le podría dar a dicha herramienta.

Existen sin embargo, otros modelos quizá menos conocidos, pero que también se están utilizando y que merecen ser, al menos, mencionados.

4.1 Algoritmos genéticos

Muchos de los modelos utilizados, como el caso de las redes neuronales o los algoritmos genéticos, están inspirados y pretenden emular comportamientos naturales o biológicos. Los algoritmos genéticos tienen origen en la década de 1970. Uno de sus precursores fue John Henry Holland, que propuso un artículo [11] donde detallaba cómo este tipo de algoritmos evolutivos era capaz de resolver problemas cuya solución yaciera en dominios de búsqueda muy extensos. Proponiendo incluso, que dichos algoritmos son capaces de encontrar soluciones a problemas que los diseñadores del propio algoritmo ni siquiera comprenden completamente. Los algoritmos genéticos han sido utilizados desde hace años en problemas complejos de optimización, donde no resultaba posible encontrar una solución óptima en un tiempo polinomial.

Los algoritmos genéticos se diseñan con un objetivo, dar solución a un problema. Cuentan con numerosas características, cada una de ellas representadas como un gen, y la combinación de todas ellas es el genotipo de la solución. En cada generación o época[2], que es el término que más ha transcendido a otros modelos, se hacen mutar distintos genes para después evaluar los diferentes genotipos, para así decidir cuales de ellos resultan más prometedores. Para esta labor, generalmente, es utilizado el aprendizaje por refuerzo que se ha visto anteriormente. Al finalizar una generación, y antes de comenzar la siguiente, se cruzan los mejores genotipos para tratar de seguir mejorando la solución. De manera aleatoria, es posible generar mutaciones, para así variar el dominio de búsqueda, lo que

[2]El término época se ha generalizado hasta significar cada paso completo del Dataset de entrenamiento

permite ampliar el mismo y salvar mínimos relativos o locales de la función de evaluación.

Uno de los problemas que subyace a los algoritmos genéticos, así como a las redes neuronales y otro tipo de modelos, es la elevada carga computacional que necesitan dichos modelos para converger. Otro gran problema es que no se puede conocer, a priori, si el algoritmo en sí va a encontrar una solución cerrada, o si por el contrario no convergerá. Esto resulta ser un grave problema, ya que puede suponer una enorme pérdida de tiempo y de recursos. Pese a estas dificultades, estos modelos, como se ha visto, han encontrado soluciones a múltiples problemas y son el estado del arte en multitud de campos.

4.2 Redes Neuronales

Otro de los múltiples modelos inspirados en la naturaleza son las redes neuronales, las cuales tratan de emular el comportamiento de las neuronas del cerebro. Aunque los primeros conceptos teóricos en los que se basan dichas redes datan de la década de 1940, no ha sido hasta estos últimos años cuando se han podido implementar de manera satisfactoria, ya que demandan una enorme potencia computacional.

Existen tantos tipos de redes neuronales como aplicaciones que puedan resolver: La red neuronal tradicional, la red neuronal convolucional, o modelos en que dos redes compiten entre sí para aumentar la robustez y fiabilidad de la solución final, son sólo algunos ejemplos. Estas últimas redes se denominan GAN [12] (*Generative Adversarial Network*) y resultan muy útiles en ciertos problemas.

Sin embargo, pese a la gran variedad de modelos, todos ellos se basan en dos conceptos muy simples: neuronas, las cuales son la unidad básica de procesado, y capas, que agrupan un conjunto de neuronas. Las redes neuronales están formadas por una capa de entrada que, generalmente, tiene tantas neuronas como características [3] se quieran introducir en la red, una capa de salida, que al igual que en la entrada, el número de neuronas depende de las características de la solución; y un número de capas intermedias u ocultas que puede ser mayor o menor según la complejidad del problema.

Las neuronas de una capa se conectan a algunas de las neuronas de las capas anterior y posterior si la red no está densamente conectada. De estarlo, todas las neuronas de una capa estarán conectadas con todas las neuronas de las capas contiguas.

[3] En la literatura es utilizado el término *features* para referirse a dichas características.

Viendo las neuronas como cajas negras, las cuales reciben una entrada y generan una salida, el periodo de entrenamiento no afecta a las neuronas en sí, sino que actualiza reiteradamente los pesos de las conexiones entre neuronas. Estos pesos oscilan en valores reales entre 0 y 1 dando mayor peso a las relaciones más prometedoras, o llegando a desconectar neuronas entre sí, según van progresando las épocas del entrenamiento.

En los siguientes capítulos se describe el desarrollo práctico del trabajo, profundizando más en los conceptos ya mencionados, así como en otros más abstractos, que todavía no han sido definidos, tales como las funciones de coste, que caracterizan el entrenamiento, o las funciones de activación de las capas, por poner dos ejemplos.

5 Desarrollo Práctico

A continuación, una vez estudiados los conceptos teóricos que sustentan los algoritmos de aprendizaje autónomo; se procede a detallar cómo se han utilizado dichos conocimientos para desarrollar una herramienta de localización en interiores.

5.1 Escenario de partida

Como se ha mencionado ya, este trabajo presenta otro enfoque diferente al utilizado en un Trabajo de Fin de Máster anterior [1] para así llegar a nuevas soluciones más prometedoras.

La situación de partida está formada por dos escenarios de medidas, el primero de ellos, se encuentra localizado en uno de los pasillos de la planta -2 del edificio I+D+i de la Universidad de Cantabria. El otro escenario se trata de la Sala Multiusos situada en el mismo edificio y en el mismo piso que el pasillo anterior. A continuación se muestran imágenes, Figuras [1] y [2], de dichos escenarios:

Figura 1: Pasillo Planta -2, edificio I+D+i

Figura 2: Sala Multiusos, edificio I+D+i

Como se puede apreciar, ambos escenarios cuentan con particularidades que los diferencian y que pueden aportar riqueza a los datos obtenidos.

El primero de los escenarios que se analiza es el pasillo. Este pasillo está dividió en cuarenta y cuatro casillas o celdas, las cuales son las posiciones a estimar. En cinco de esas celdas se colocan las correspondientes balizas BlueTooth. Aunque, como se explicará posteriormente, no es necesario el conocimiento de la posición de las balizas para la solución que se propone en el presente trabajo, sí que lo era para el trabajo anterior, ya que aquel algoritmo depende del conocimiento de dichas posiciones.

Otra de las peculiaridades del pasillo es la existencia de columnas a lo largo del mismo, tal y como se observa en la Figura [1], las cuales afectan, como se desprende de los estudios de dicho trabajo, al diagrama de radiación de las balizas. Otro factor a estudiar será, por lo tanto, la robustez de la solución que se propone ante obstáculos inmóviles en el terreno.

La [3] muestra una vista de planta del pasillo con las correspondientes celdas, así como las posiciones de las balizas y las columnas existentes.

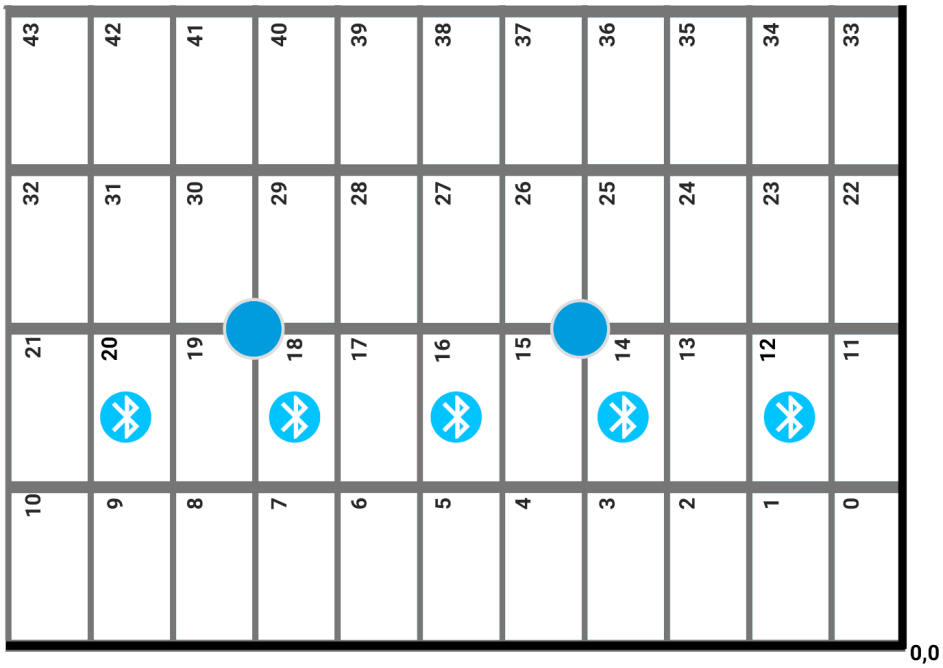

Figura 3: Esquema primer escenario, Pasillo

En cuanto al segundo escenario, se sitúa en la Sala Multiusos de la planta -2. Esta sala está dividida en cuarenta y cinco celdas, colocando en ocho de las mismas las correspondientes balizas BlueTooth. Además, estas balizas se colgaron del techo para procurar una menor interferencia de mesas, sillas y demás mobiliario, así como de los posibles usuarios.

En cada una de las casillas así definidas se llevan a cabo medidas, en el primer escenario durante treinta segundos por cada posición, mientras que en este segundo escenario el tiempo de escaneo es de cuarenta y cinco segundos por cada posición.

Al igual que en el caso anterior, el esquema de este segundo escenario es el que se muestra en la Figura [4].

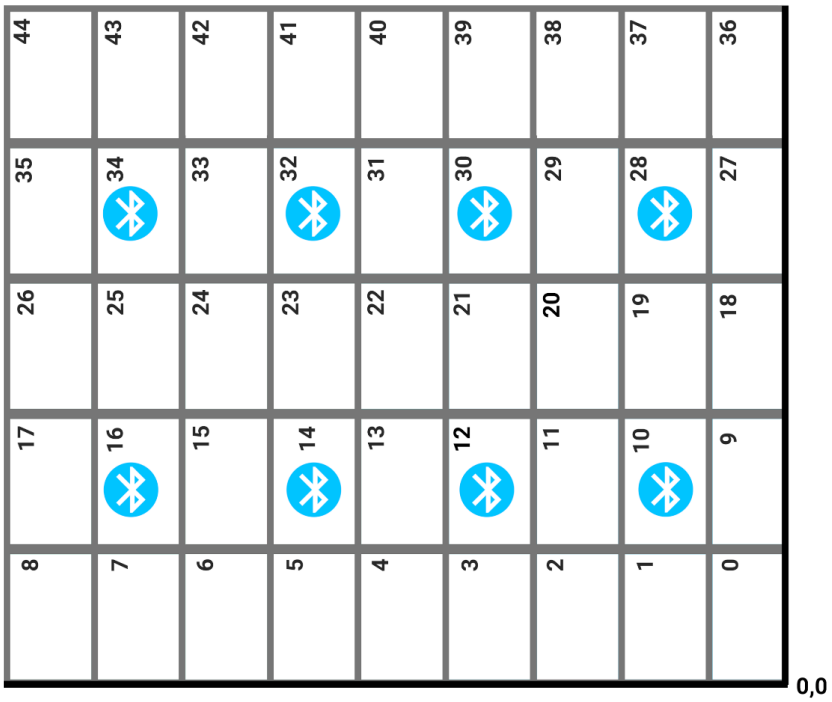

Figura 4: Esquema segundo escenario, Sala

5.1.1 Análisis de Resultados

Una vez conocidos los escenarios de este apartado, se detalla la estructura del conjunto de datos obtenido, así como los resultados alcanzados en el trabajo [1].

Los datos de ambos escenarios están almacenados en sendos archivos con formato "csv" y siendo cada fila la información procesada de un *Beacon* recibido. Cada una de esas filas incluye los datos de su identificador mac, la potencia recibida, la posición en la que se realizó la captura y el instante temporal en que dicha captura fue realizada. En la siguiente Tabla [1], se ejemplifica la estructura de los datos de únicamente dos beacons.

MAC	RSSI (dBm)	Posición	Marca Temporal
C8:FD:19:13:67:F2	-98	0	1497457681768
C8:FD:19:13:43:ED	-83	0	1497457681772

Cuadro 1: Ejemplo estructura de los datos

Conocida la estructura de los datos, se procede a detallar los resultados que fueron obtenidos en el pasado trabajo y que serán el punto de comparación con las nuevas técnicas que se implementan en este documento.

Resultados Primer Escenario

A la hora de analizar los resultados del primer escenario, existen varias características que se han de tener en consideración. La primera de ellas es que no se logró posicionar al usuario con precisión de celda. Sino que hubo que dividir el pasillo de forma longitudinal en cuatro posiciones. Por lo que el posicionamiento es básicamente unidimensional, es decir, a lo largo del pasillo pero no a lo ancho. En la Figura [5] se muestra la agrupación de las celdas por colores.

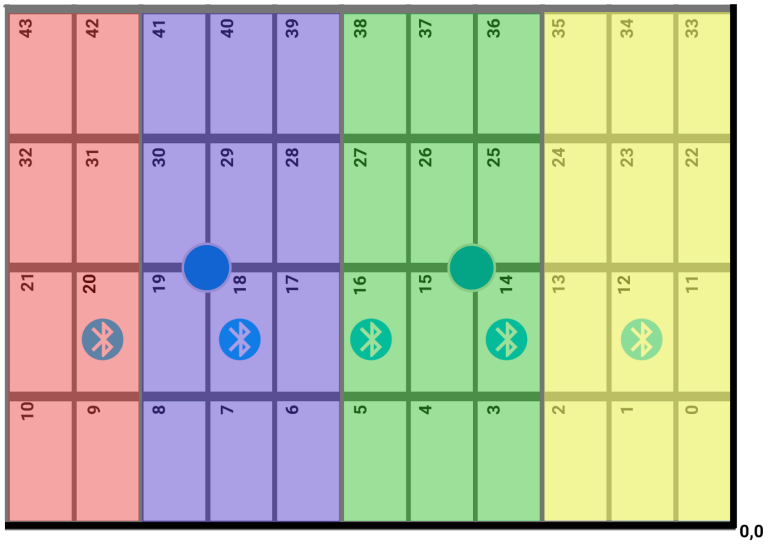

Figura 5: Esquema primer escenario, Pasillo

Otra de las cuestiones que resultaron interesantes fue que los mejores resultados fueron obtenidos cuando la potencia emitida por las balizas BlueTooth era $6dBm$, es decir, la máxima.

Finalmente se ha de tener en cuenta que los resultados obtenidos no corresponden a una única captura en un instante de tiempo sino que se debía permanecer capturando durante veinte segundos.

Teniendo en cuenta todo lo anterior, se obtuvo una tasa de acierto de un 81,82 %.

Resultados Segundo Escenario

En el segundo escenario, la sala, se pretende posicionar al usuario bidimensionalmente, para ello, se divide dicha sala en cuatro cuadrantes. Tal y como se muestra en la Figura [6] entre los distintos cuadrantes simbolizados con colores existe un área de separación para facilitar la labor del decisor.

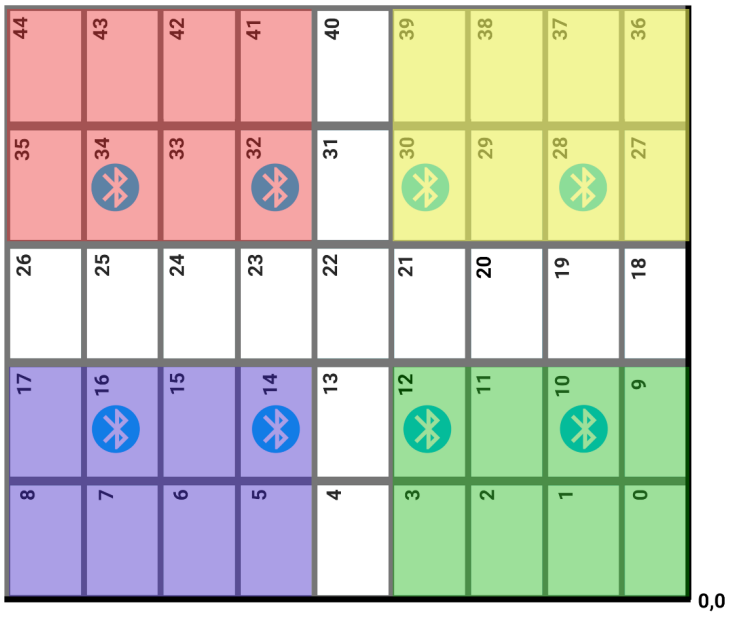

Figura 6: Esquema segundo escenario, Sala

Al igual que en el escenario anterior, los mejores resultados se obtuvieron realizando una captura durante veinte segundos y con las balizas emitiendo a máxima potencia, $6dBm$. Los resultados, sin embargo, fueron significativamente peores que en el primer escenario; siendo, en este caso, la tasa de acierto de tan solo un 59,38 %.

5.2 Procesamiento de los datos

Una vez analizados los resultados de partida, se procede a la definición de un algoritmo basado en aprendizaje supervisado. Para ello, primero es necesario analizar el conjunto de datos del que se dispone y ver cuál es la mejor forma de introducir dichos datos en la red neuronal que se pretende crear.

Con las nociones del comportamiento básico de una red neuronal mencionadas en el capítulo 4.2, se sabe que toda red neuronal debe tener una capa de entrada y otra de salida, así como otras intermedias que ahora no son relevantes.

Tras estudiar el problema, se llegó a la conclusión que lo que mejores resultados reporta no es alimentar al algoritmo como estaban distribuidos los datos originariamente, es decir, con una sola captura por fila. La mejor opción es colocar en una sola fila tantas columnas como balizas BlueTooth, y en cada una de las columnas almacenar la potencia recibida de dichas balizas. Dichas columnas representarán cada una de las características (*features*) de los datos de entrada. Por lo tanto, para el primer escenario se cuenta con cinco columnas, una por baliza.

También se debe proporcionar la información de la posición en que se han tomado dichas capturas, por lo que se añade una columna más a la fila. El resultado pasa por modificar la estructura referenciada en la Tabla [1]. Para ello se ha programado un *script* en *Python* que desecha la información que no se necesita, la marca temporal, y reformatea los datos necesarios para que sigan la estructura de la Tabla [2].

RSSI 1	RSSI 2	RSSI 3	RSSI 4	RSSI 5	Posición
-93	-83	-98	-92	-97	0
-86	-99	-98	-95	-90	5

Cuadro 2: Ejemplo estructura de los datos

Con esto se consigue un conjunto de datos con el que es más fácil trabajar, ya que cada

fila se corresponde con las medidas concretas de una posición en un instante de tiempo, y cuenta con la información conjunta de las intensidades de todas las balizas.

Para procesar los datos del segundo escenario se procedió de manera análoga. El único cambio relevante es que en lugar de utilizar filas de seis columnas, las cinco correspondientes a las señales de las diferentes balizas y la que determina la posición, se utilizan nueve columnas, al haber ocho balizas.

Una vez los datos están con el formato deseado todavía quedan dos procesos que realizar. Estos procesos son comunes a ambos escenarios por lo que se explican de manera conjunta.

Tras introducir al algoritmo los datos etiquetados, es necesario poder evaluar el algoritmo una vez haya superado la etapa de entrenamiento o aprendizaje. Para ello se debe dividir el conjunto de datos para que existan muestras que se utilicen para entrenar y otras que el algoritmo no conozca y que se usen para evaluar el comportamiento final del mismo.

Para dividir el conjunto de datos, en primer lugar se aleatoriza el orden de las filas y posteriormente se divide dedicando el 85 % de las muestras o filas al periodo de aprendizaje, y reservando el 15 % restante para evaluar los resultados.

Finalmente se debe recordar que, en última instancia, este problema a ojos del algoritmo es un problema de categorización. Se tienen unos datos a la entrada que corresponden con una de las cuarenta y cuatro o cuarenta y cinco posiciones, según el escenario. El algoritmo debe ser capaz de establecer en qué categoría o posición está cada dato de entrada.

Debido al comportamiento de las redes neuronales, la posición o clase no se debe determinar como un número decimal, sino como un vector de longitud igual al número total de clases. Las clases son las posiciones en el presente problema. Por lo tanto, para representar la posición X, el vector tendrá valor '1' en el índice X y valor '0' en el resto de índices. Este tipo de codificación se denomina *One Hot Encoding*.

Una vez se tienen los datos procesados, se dividen las entradas (las columnas correspondientes a las potencias recibidas de las balizas) de las salidas (el vector que determina la posición). Obteniendo así cuatro conjuntos de datos:

- train_X: Corresponde a los datos de entrada del algoritmo en la etapa de entrenamiento.

- train_Y: Corresponde a las etiquetas de los datos de entrada utilizados en la etapa de entrenamiento.
- test_X: Corresponde a los datos de entrada de la etapa de evaluación.
- test_Y: Corresponde a las etiquetas de los datos entrada de la etapa de evaluación.

5.3 Entorno de desarrollo de Redes Neuronales

Tras procesar los datos para facilitar su uso y para optimizar los resultados del algoritmo, es necesario diseñar y programar la red neuronal que deberá aportar una solución novedosa y eficaz al problema de la localización en interiores.

Para la programación de la red neuronal se ha utilizado el lenguaje de programación *Python*, el *Framework Tensorflow*[13] y la *API Keras*[14]. Así mismo, se han utilizados diversos paquetes de Python, tales como *Numpy* o *matplotlib*, tanto para el procesado de los datos como para la representación de los resultados. A continuación se detalla qué es y en qué se basa cada una de las herramientas mencionadas.

En primer lugar Python [15] es un lenguaje de programación que está cobrando gran relevancia en prácticamente todos los ámbitos de desarrollo de software. Esto es debido, entre otros factores, a su gran flexibilidad y escalabilidad. Concretamente en el mundo del aprendizaje autónomo y la inteligencia artificial es, posiblemente, el lenguaje más utilizado junto a R [16], que es un lenguaje orientado al análisis de datos y a la estadística.

Una de las numerosas bondades de Python es la posibilidad de "importar paquetes". Esto quiere decir que se pueden utilizar módulos de software desarrollados por terceros, que aumentan o facilitan las tareas que se pueden realizar. Estos paquetes se pueden asemejar a las librerías de C, aunque la importación y el uso de los paquetes en Python resulta mucho más sencillo. Para poder utilizar (importar) un paquete, primero se debe instalar, para ello se utiliza un gestor de paquetes denominado *Pip* (acrónimo de *Pip instalador de Paquetes*). Una vez el paquete está instalado, se puede importar mediante la instrucción: *import nombre_del_paquete*.

Entre los paquetes que se han utilizado en este proyecto destacan principalmente dos, el framework Tensorflow y una API de alto nivel para facilitar su uso denominada Keras. Sin embargo, también se han utilizado otros paquetes como Numpy o Pandas para facilitar el trabajo con vectores, matrices y tensores; o Matplotlib para realizar las gráficas.

5.3.1 Tensorflow

Tensorflow es un framework desarrollado por Google y orientado al desarrollo de algoritmos de aprendizaje autónomo. Una de las grandes ventajas que aporta es su flexibilidad, ya que un mismo código, con apenas cambios, puede ejecutarse sobre sistemas móviles, entornos de escritorio o máquinas multigpu. Incluso es posible su ejecución en entornos distribuidos con cientos de máquinas. Tensorflow permite crear modelos para la etapa de entrenamiento y evaluación de algoritmos, que abarcan desde la regresión lineal hasta *Deep Neural Networks* o redes neuronales profundas y *Convolutional Neural Networks*. Estas últimas resultan especialmente útiles en problemas de reconocimiento o procesado de imágenes.

Como se ha comentado previamente, Tensorflow permite ejecutarse en multitud de dispositivos. En el caso que concierne a este trabajo se optó por ejecutar en un ordenador personal que cuenta con una GPU, o tarjeta gráfica, Nvidia GTX 1050, lo que supone una gran ventaja frente a la ejecución en un procesador tradicional o CPU. Para entender por qué una tarjeta gráfica puede realizar este tipo de trabajo a una velocidad significativamente superior a lo que lo haría un procesador, se debe comprender el funcionamiento de las mismas. La labor de una tarjeta gráfica es procesar los bits que conforman una imagen. Si por ejemplo, se tratase de una imagen a resolución FullHD (1920x1080 píxeles) se debe operar, prácticamente en tiempo real con más de dos millones de píxeles. Para ello, las tarjetas gráficas han evolucionado paralelizando las operaciones. En la actualidad, una tarjeta gráfica como la que se ha usado para este trabajo, cuenta con unos 800 procesadores (*Cuda Cores* [17]) muy optimizados para realizar operaciones aritméticas básicas de forma concurrente.

Si se compara este enorme número de procesadores con los núcleos que conforman un procesador actual, se ve que difieren en dos órdenes de magnitud en favor de los núcleos Cuda, por lo que la diferencia de tiempo que conlleva realizar operaciones paralelas es abismal.

Volviendo al funcionamiento de una red neuronal, se concluye que están formadas por un gran número de neuronas, que será mayor o menor dependiendo de la complejidad de la propia red, y que dichas neuronas únicamente realizan operaciones aritméticas básicas. Es por ello que si se pudieran volcar dichas neuronas en los múltiples núcleos cuda, se podría paralelizar su ejecución ahorrando una inmensa cantidad de tiempo.

Tensorflow permite esto de manera muy sencilla. Únicamente se debe instalar el paquete *Tensorflow-gpu*, tener actualizados los controladores de la tarjeta gráfica y el *CUDA*

Toolkit instalado. Cumpliendo los anteriores requisitos, Tensorflow detectará la tarjeta gráfica y ejecutará sobre ella los algoritmos que se describan.

5.3.2 Keras

Si bien Tensorflow es una herramienta realmente útil, puede resultar algo complicada de utilizar, o al menos tediosa. Debido a la numerosa cantidad de posibilidades que ofrece, realizar operaciones sencillas, como definir una capa dentro de una red neuronal, puede suponer varias decenas de líneas de código. Keras resuelve estos problemas aportando una API de alto nivel capaz de ejecutarse tanto sobre Tensorflow, como sobre otros frameworks (CNTK o Theano), facilitando las tareas de definición y creación de algoritmos.

Que sea sencillo de utilizar no quiere decir, necesariamente, que sea menos potente. El hecho es que Keras se adapta perfectamente tanto a los requisitos de este trabajo como a otros mucho más complejos. Así resulta una herramienta perfecta para los desarrolladores que desean utilizar alguno de los frameworks mencionados anteriormente sin necesidad de un conocimiento exhaustivo de los mismos.

Finalmente, para destacar la relevancia de Keras en los entornos de desarrollo, se puede buscar en *GitHub* la palabra "keras", lo que retornará un total de cerca de 15000 repositorios. Esos 15000 repositorios suponen la tercera parte del número de repositorios que se encuentran si se busca el término "tensorflow". Keras, por lo tanto representa un sector realmente significativo del desarrollo de algoritmos de aprendizaje autónomo.

5.4 Funcionamiento de la Red Neuronal

A modo de resumen, se muestra una imagen, Figura [7], que representa el flujo de trabajo que se ha llevado a cabo, donde se aprecian la ingesta de datos, el preprocesado de los mismos, y las etapas de entrenamiento y evaluación. Tras concluir dichos procesos llegaría la etapa de despliegue del algoritmo, al ser este un trabajo de corte más teórico se ha prescindido de dicha etapa.

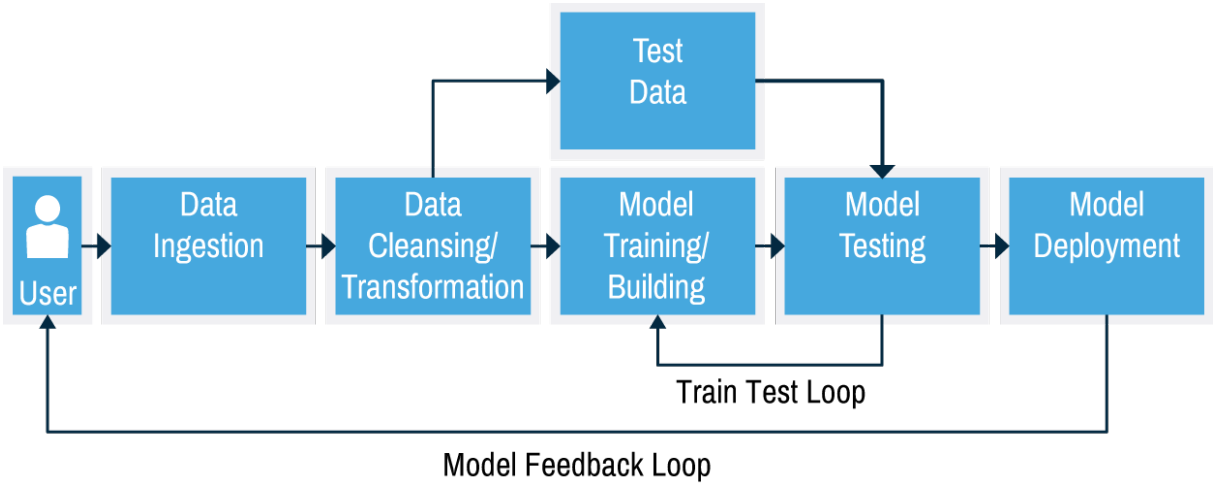

Figura 7: Flujo de Trabajo

Vistas las herramientas que se han utilizado, se procede a la explicación de la red neuronal. Se detalla cómo ha sido la implementación del diseño inicial, cómo se han introducido los datos a la entrada y cómo se ha evaluado la precisión de la salida.

En primer lugar se importaron los paquetes necesarios, como se puede observar en el siguiente fragmento de código:

```
import os
import numpy as np
import matplotlib.pylab as plt
import keras
from keras.models import Sequential
from keras.layers import Dense, AlphaDropout, Dropout, LeakyReLU
from keras.models import model_from_json
from keras.utils import plot_model
from keras import regularizers
```

A continuación se cargan los datos de entrenamiento. Estos datos son los que han sido ya formateados y guardados como *Numpy arrays*. Tras cargarlos, se almacena en dos variables el número de balizas BlueTooth, así como el número total de posiciones. En el siguiente fragmento de código se muestra el proceso anteriormente descrito.

```
# Cargar datos de entrenamiento
X = np.load("dataset/pasillo/train_X.npy")
Y = np.load("dataset/pasillo/train_Y.npy")

#Número de entradas (Los diferentes RSSI de Beacons bluetooth)
num_input = X.shape[1]       #Número de beacons BlueTooth
num_classes = Y.shape[1]     #Número de posiciones 0, 1, 2, ..., 43
```

Teniendo ya los datos cargados, así como la información del número de balizas y de posiciones, se procede a crear el modelo de red neuronal. Para ello se definen las diferentes capas que conforman la propia red. En primer lugar se crea el modelo y se añade una capa de entrada que tiene tantas neuronas como número de balizas. Además se incluyen unos penalizadores que intervienen en la etapa de entrenamiento, tanto en los parámetros de la red, sus neuronas, como en su función de activación.

```
# Crear modelo de red neuronal
model = Sequential()

#Capa de entrada
model.add(Dense(num_input, input_dim=num_input,
            kernel_regularizer=regularizers.l2(0.001),
            activity_regularizer=regularizers.l1(0.001)))
```

También se puede observar que la capa tiene el modificador *"Dense"*. Esta característica es común a todas las capas, lo que significa que cada una de las neuronas de una capa está conectada con todas las neuronas de las capas adyacentes.

A continuación, se definen las capas ocultas que serán *"Dense"* y tendrán respectivamente 512, 1024 y nuevamente 512 neuronas. Las dimensiones de dichas capas se han obtenido tras probar repetidamente diferentes configuraciones, siendo la mencionada anteriormente la que mejores resultados ha reportado. Además de estas capas se añade entre ellas capas *Dropout*[4] y *LeakyReLu*[5] cuya función es aumentar la robustez del algoritmo. El resto de capas utilizan como función de activación la ya nombrada ReLu.

[4]Dropout: Consiste en, arbitrariamente, reducir a cero la entrada de las neuronas. `https://keras.io/layers/core/#dropout`

[5]LeakyReLu: Versión modificada de *ReLu, Rectified Linear Unit* que añade un gradiente cuando la unidad no está activa. `https://keras.io/layers/advanced-activations/#leakyrelu`

```
#Capas Ocultas
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(LeakyReLU(alpha=0.6))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.4))
model.add(LeakyReLU(alpha=0.6))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(LeakyReLU(alpha=0.6))
```

Tradicionalmente, en las capas ocultas, se han usado funciones de rectificación o activación como la Sigmoid $f(x) = sigmoid(x) = \frac{1}{1+e^{-x}}$, siendo x la entrada a la neurona. Sin embargo, actualmente en las redes neuronales profundas se está optando por utilizar la función ReLu que se asemeja más al comportamiento de una neurona en el cuerpo humano. Esta función vale cero cuando el valor de la entrada es negativo y el propio valor de entrada cuando es positivo, se define según la siguiente ecuación: $f(x) = max(x, 0)$.

A diferencia de las capas ocultas, en las capas de salida de los problemas de clasificación binarios sí que funciona muy bien la activación sigmoid. En el caso que atañe a este trabajo, sin embargo, no se puede utilizar ya que la clasificación que se pretende realizar dista de ser binaria, ya que existen más de cuarenta posibles catalogaciones. El equivalente a la función sigmoid en estos casos es la denominada *Softmax*: $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}$. La salida de esta función es la salida de la red neuronal y dicta la probabilidad entre 0,0 y 1,0 de que cada una de las posiciones sea la correcta. Finalmente, la posición con una mayor probabilidad será la estimación de la posición del usuario.

Finalmente, para describir la capa de salida se debe indicar que la dimensión de dicha capa es igual a la del número de posiciones o clases.

```
#Capa de salida
model.add(Dense(num_classes, activation='softmax'))
```

Una vez definidas las propiedades de la red neuronal, se procede a compilarla, determinando así la función de coste, el optimizador y, de manera opcional, una métrica que se utiliza para ver la evolución de la precisión de la red mientras entrena.

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['acc'])
```

La función de pérdidas es la denominada *Categorical Crossentropy*, que representa la entropía cruzada entre la predicción realizada y el valor real. Por otro lado, el optimizador utilizado se llama *adam, Adaptive Gradient Algorithm*. Lo que hace este algoritmo es acortar los tiempos de entrenamiento utilizando un gradiente adaptativo, mayor al principio, lo que permite el rápido aprendizaje, y menor cuando se va acercando a la solución, para evitar saltar a otro dominio de búsqueda.

Una vez definido el modelo de red neuronal que se va a utilizar, se crea un histórico donde se guarda la evolución de la tasa de acierto y de los resultados de la función de pérdida, a medida que va progresando en el entrenamiento. Finalmente, se da comienzo al entrenamiento, estableciendo los datos de entrada, el número de épocas de las que constará dicho entrenamiento, y el número de muestras que se procesarán a la vez. Adicionalmente se establece la opción *"shuffle"* para que al principio de cada época se cambie el orden de los datos de entrada. También se reserva un 10 % de las muestras para comprobar la evolución.

```
class AccuracyHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.acc = []

    def on_epoch_end(self, batch, logs={}):
        self.acc.append(logs.get('acc'))

history = AccuracyHistory()

# Entrenar el modelo
history = model.fit(X, Y, epochs=200, batch_size=64, shuffle=True,
                    validation_split = 0.1)
```

Finalmente se realiza una primera evaluación del modelo entrenado, utilizando los propios datos de entrenamiento. Es por esto, que la tasa de acierto seguramente sea mayor que si se utilizan datos nuevos. Sin embargo, si se ha conseguido un buen entrenamiento, esta penalización será marginal, aunque pueden existir problemas si se entrena poco o si por el contrario se sobreentrena. En ambos casos el algoritmo no responde adecuadamente a nuevos datos, bien porque no ha aprendido lo suficiente, o porque se ha ajustado demasiado a las muestras dadas. A continuación se muestra el código previamente descrito:

```
# Evaluar el modelo
resultados = model.evaluate(X, Y)
```

```
# Serializar modelo a JSON
model_json = model.to_json()
with open("model/pasillo/model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model/pasillo/model.h5")
print("Saved model to disk")
```

En el código anterior se muestra cómo se guarda el modelo tras superar la etapa de entrenamiento. Para ello, lo que se guarda es tanto la estructura de la red como los pesos de las conexiones entre neuronas. Este modelo guardado se puede cargar en cualquier momento, y bien utilizarse para realizar ya las clasificaciones, o para volver a pasar por el entrenamiento si así se desease.

Para finalizar, se han creado dos gráficas que muestran la evolución de la precisión y del resultado de la función de pérdida a lo largo del entrenamiento:

```
# Listar los datos en "history"
print(history.history.keys())
# Pintar evolución de la precisión (tasa de acierto)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Evolución del Modelo')
plt.ylabel('Tasa de acierto')
plt.xlabel('Épocas')
plt.legend(['Entrenamiento', 'Evaluación'], loc='upper left')
plt.show()

# Pintar evolución de la pérdida
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Evolución del Modelo')
plt.ylabel('Función de pérdida')
plt.xlabel('Épocas')
plt.legend(['Entrenamiento', 'Evaluación'], loc='upper left')
plt.show()
```

Con esto, ya está lista la red neuronal y se puede entrenar para evaluar su respuesta ante el problema de la localización en interiores. Al estar parametrizado el número de

balizas y de posiciones, el mismo código vale para los dos escenarios, e incluso para otros posibles problemas.

5.5 Resultados

Teniendo presentes los resultados correspondientes a anteriores técnicas, que han sido recopilados en el capítulo 5.1.1, se procede a analizar los resultados proporcionados por la red neuronal. Para ello, nuevamente se diferenciarán los dos escenarios considerados.

5.5.1 Resultados Primer Escenario

Recapitulando, el primer escenario corresponde al pasillo, en el cuál hay tan solo cinco balizas BlueTooth y 44 posiciones. En este primer escenario se muestran los resultados correspondientes a las medidas realizadas con una potencia de transmisión de $0dBm$, $3dBm$ y $6dBm$. Se prescinde de usar las medidas realizadas con menos potencia, ya que se cuenta con muy pocas medidas.

A continuación se muestra la Tabla [3] donde se recogen los resultados proporcionados. Se debe tener en cuenta que estos resultados tienen la precisión máxima, es decir, sólo se computa como acierto si la posición estimada es exactamente la misma posición que la real.

Potencia Transmitida	Número de Medidas	Aciertos	Fallos	Precisión (%)
$0dBm$	390	295	95	75,64 %
$3dBm$	592	418	174	70,60 %
$6dBm$	531	360	171	67,79 %

Cuadro 3: Resultados Primer Escenario

Se puede comprobar que, incluso en el peor caso, la precisión obtenida es inmensamente superior que la obtenida en el estudio original.

Cabe remarcar que en el anterior trabajo no fue posible posicionar a un usuario con la precisión de una celda, sino que fue necesario hacer agrupaciones de celdas para lograr los resultados recogidos en 5.1.1.

Otro de los datos realmente interesantes, es que la forma de analizar la precisión es comparar las predicciones que otorga la red neuronal a las 390, 592 y 531 medidas con las posiciones reales. No es necesario por lo tanto escuchar durante 20 segundos para determinar la posición de un usuario, sino que es suficiente con escuchar una señal de cada baliza.

5.5.2 Resultados Segundo Escenario

A continuación se recogen los resultados del segundo escenario. En este caso se estudian cuatro casos diferentes, según la potencia transmitida por las balizas BlueTooth. En la Tabla [4] se recogen estos resultados:

Potencia Transmitida	Número de Medidas	Aciertos	Fallos	Precisión (%)
$-3dBm$	52	41	9	87,84 %
$0dBm$	628	556	72	88,53 %
$3dBm$	664	575	89	86,59 %
$6dBm$	618	518	100	83,81 %

Cuadro 4: Resultados Segundo Escenario

Como se puede observar, los resultados en este caso resultan aún mejores que los del primer escenario. Probablemente sea porque había más balizas transmisoras. Otro dato interesante a remarcar, es que, en ambos escenarios, la mejora en la tasa de acierto es inversamente proporcional al aumento de potencia transmitida (Salvando cuando se envían $-3dBm$ que no se reciben suficientes datos como para que el resultado sea significativo). Este hecho se contradice con el trabajo original, donde el aumento de dicha potencia de transmisión mejoraba la respuesta del algoritmo.

Finalmente, para poder ver cómo evoluciona la red neuronal a medida que entrena, se muestran las siguientes figuras, Figura [8] y Figura [9]. Que representan la evolución de la tasa de acierto (Figura [8]) y el valor de la función de coste (Figura [9]) en función de las épocas. Ambas figuras corresponden al segundo escenario, aunque las del primero son análogas a las mismas.

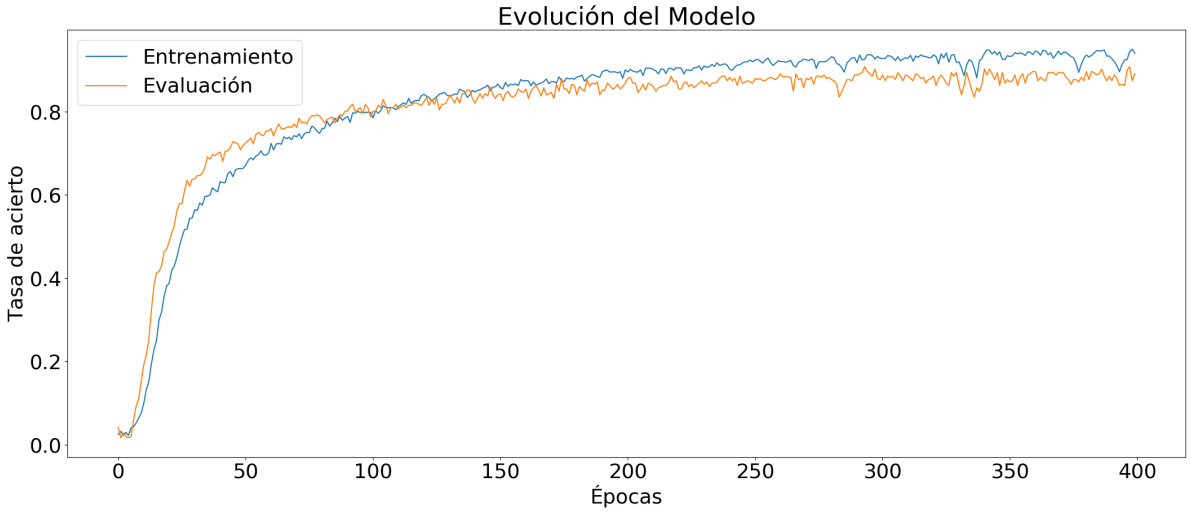

Figura 8: Evolución de la tasa de acierto

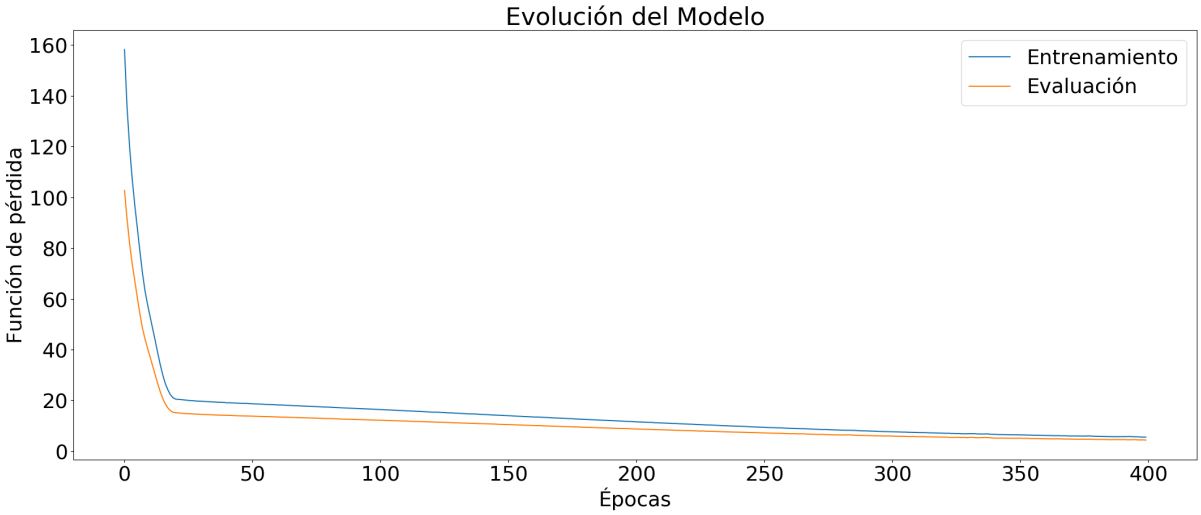

Figura 9: Evolución de la función de pérdida

Como se puede ver, el aumento de precisión y la disminución de la función de pérdida es enorme durante las primeras épocas y luego se va refinando poco a poco. Esto se debe al optimizador elegido que utiliza un gradiente grande al principio y menor a medida que van pasando las épocas.

Igualmente se aprecian ligeros empeoramientos en la tasa de acierto debido a que el algoritmo intenta modificar los pesos de las conexiones para tratar de hallar nuevos dominios de búsqueda.

Aparte de estos resultados, se ha realizado un pequeño programa para analizar la distribución de los errores, para estudiar si los errores se concentran en uno de los ejes, X o Y. También se ha estudiado si los errores se concentran en posiciones cercanas a la real, o si se encuentran en posiciones alejadas.

Para entender los mapas de calor de la Figura [10] se debe explicar que se han renombrado las posiciones en función de sus cordenadas X e Y. Posteriormente, se ha realizado la resta de las posiciones reales y las predichas por el algoritmo. Se han descartado todas aquellas posiciones que resultaban correctas y se han representado los errores en función de cuánto se alejaban.

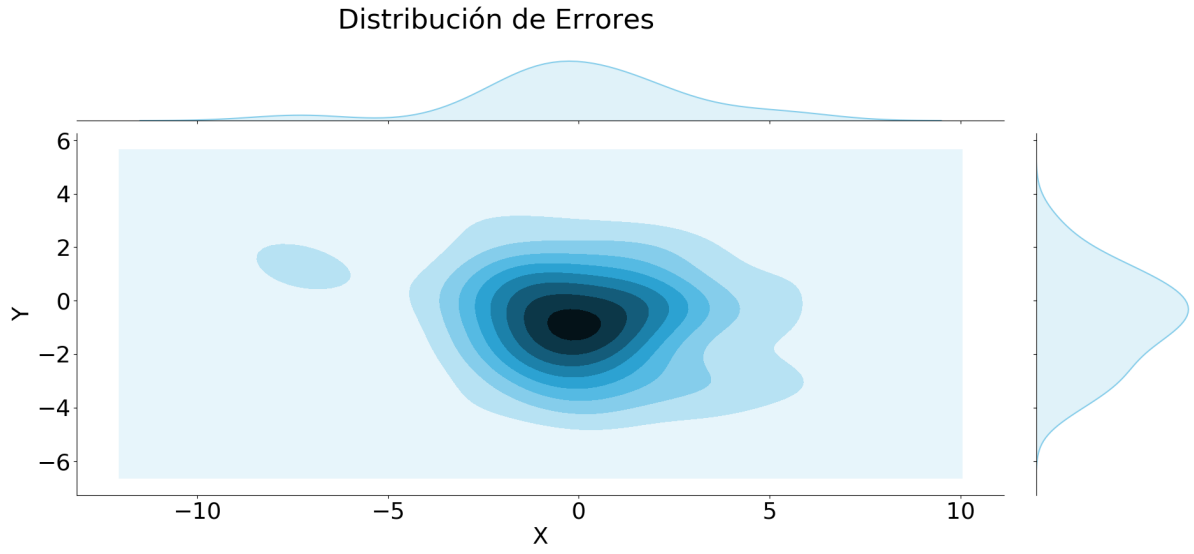

Figura 10: Distribución de los errores

Como se puede observar, la mayoría de los errores se encuentran en las posiciones adyacentes a la posición real, $(0, 0)$, por lo que el algoritmo parece comportarse de manera aceptable incluso cuando falla.

Cabe mencionar que en las pruebas que se ha realizado, la distribución de los errores no parece depender de los ejes X o Y, sino que el número de fallos en ambos ejes es similar. Hecho que se puede ver claramente en la gráfica. Por último decir que el gráfico de la Figura [10] corresponde a una evaluación de la respuesta del algoritmo al segundo escenario, pero que en ambos escenarios se comporta de manera similar.

Finalmente se muestra otro gráfico interesante. La Figura [11] representa una matriz de confusión. En ella se muestran las posiciones predichas por el algoritmo, eje Y frente a las posiciones reales, eje X. En dicha matriz se puede observar que la inmensa mayoría de las posiciones predichas coinciden con la posición real, por lo que están en la diagonal. También se observa que se han evaluado todas las posiciones aunque no de manera equitativa. Para facilitar su visualización se muestra la matriz como mapa de calor.

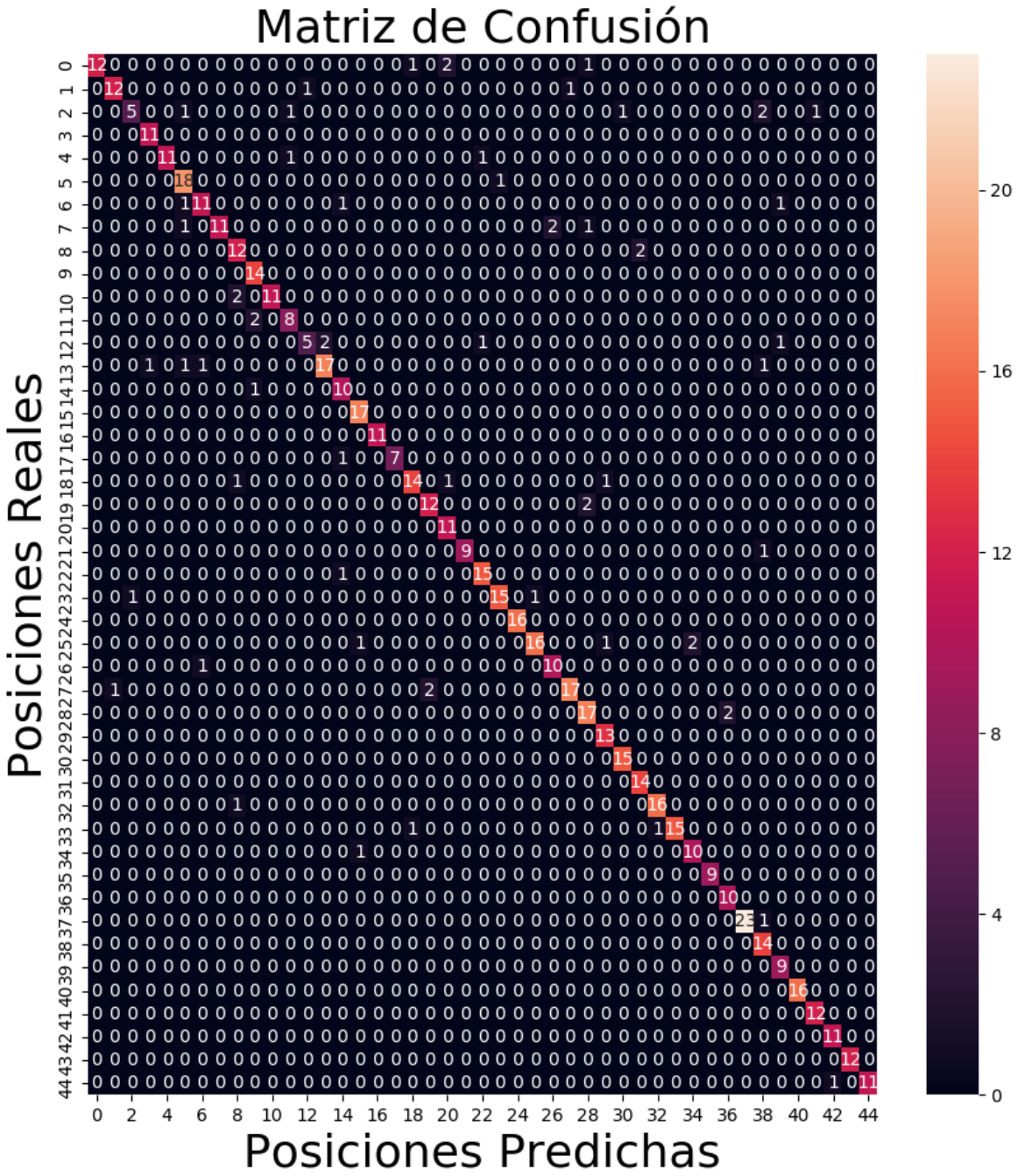

Figura 11: Matriz de Confusión del escenario 2

5.6 Análisis de un Nuevo Escenario

Visto el éxito obtenido con los datos existentes, se ha procedido a utilizar la misma red neuronal para analizar un nuevo Dataset que mezclara beacons WiFi con las señales de balizas BlueTooth para evaluar tres situaciones.

- Uso exclusivo de beacons WiFi
- Uso exclusivo de balizas BlueTooth
- Uso combinado de ambas señales

El objetivo es estudiar si aporta algo la utilización de diferentes señales, o si por el contrario, la ganancia que se puede extraer es marginal.

5.6.1 Uso de Beacons WiFi

Antes de comenzar, se describe la distribución física de este nuevo escenario. En este caso se ha optado por realizar las medidas a lo largo del pasillo de la planta -2 del edificio de I+D+i de la Universidad de Cantabria. El objetivo es ver si el algoritmo es capaz de situar al usuario a lo largo del pasillo. Los puntos en que se han realizado las mediciones, así como la ubicación de las balizas BlueTooth que se han usado en el presente escenario están representados en la siguiente figura, Figura [12].

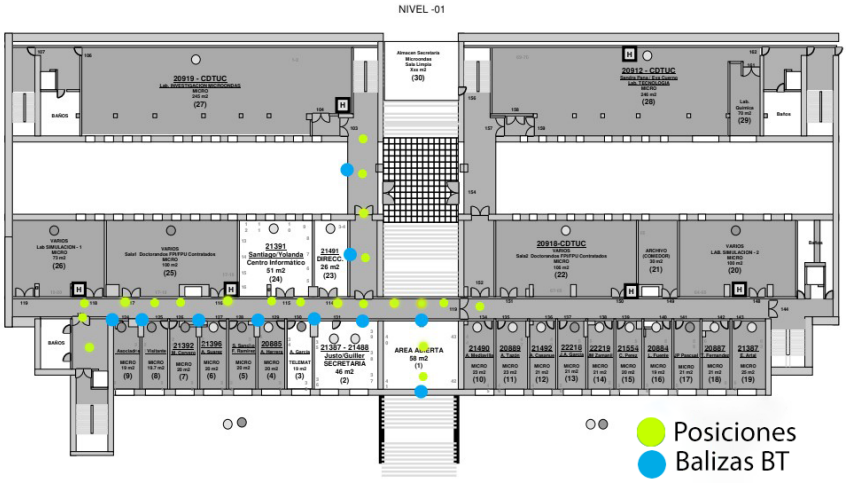

Figura 12: Escenario Planta -2

Para este primer análisis, sin embargo, únicamente se han utilizado las señales WiFi emitidas por los puntos de acceso situados en el piso -2 del edificio de I+D+i. No se ha contado con un conocimiento ni de cuántos puntos de acceso hay, ni de su posición, ni de la potencia con la que emiten. Para realizar las capturas se ha programado pequeño script 6.2. Una vez obtenidos los datos se han reformateado análogamente al proceso descrito en el capítulo 5.2.

Teniendo los datos ya formateados, se han introducido a la red neuronal, la cuál ha sido modificada, reduciendo el número de neuronas en cada capa oculta a 128, 256 y 128. La razón de esta reducción de dimensiones es que los datos adquiridos son menos diversos y que existen menos posiciones (un total de 20), por lo que una red neuronal mayor puede tener problemas de sobreentrenamiento. A continuación se mostraran las figuras, Figura [13] y Figura [14], que representan la evolución del algoritmo.

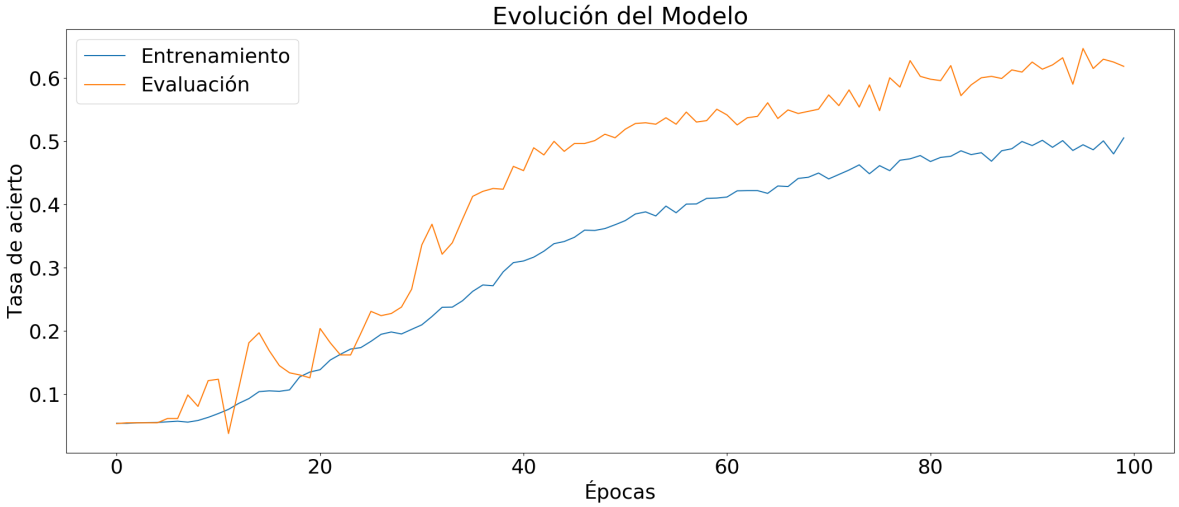

Figura 13: Evolución de la tasa de acierto con Beacons WiFi

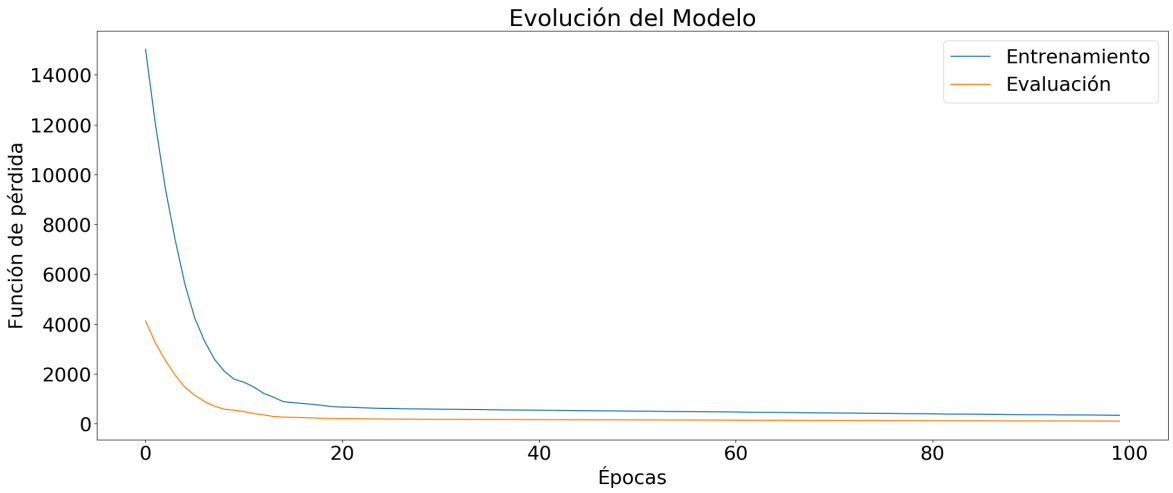

Figura 14: Evolución de la función de pérdida con Beacons WiFi

Como se ve en el eje X, también se ha reducido el número de épocas porque de seguir con las 400 iniciales se producía el sobreentrenamiento. Esto es debido, como ya se ha comentado, a que el Dataset no es realmente rico, ya que muchas medidas se repiten y existe poca diversidad.

Con estos datos se obtiene una tasa de acierto superior al 60 %, que si bien es inferior a los resultados obtenidos con las balizas BlueTooth, en este caso no ha sido necesario realizar ningún despliegue de infraestructura, sino que han sido utilizados únicamente los puntos de acceso que estaban previamente instalados. Lo cual es un punto que se debe tener en consideración.

5.6.2 Uso de Beacons BlueTooth

Para realizar esta nueva medición, se han colocado un total de 10 balizas BlueTooth a lo largo del pasillo descrito anteriormente, Figura [12]. Una vez colocadas, se ha procedido a realizar las capturas en cada una de las posiciones. Al igual que sucedía con las señales WiFi, es necesario reformatear los datos obtenidos para que estos puedan alimentar la red neuronal. Una vez completados estos procesos se inicia la etapa de entrenamiento. Dicha etapa concluye con los siguientes resultados, Figura [15] y Figura [16].

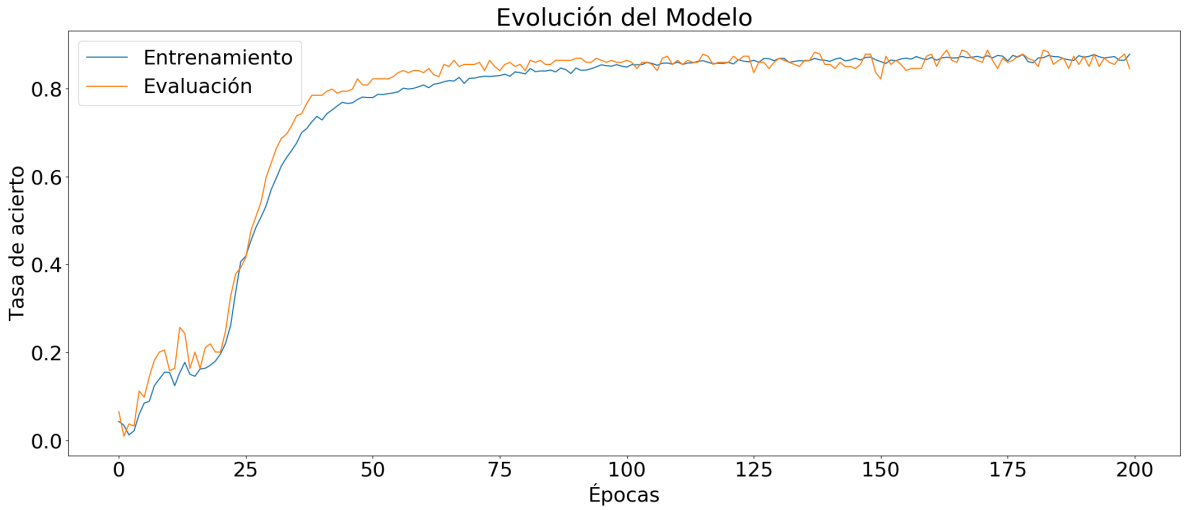

Figura 15: Evolución de la tasa de acierto en el escenario 3 con beacons BlueTooth

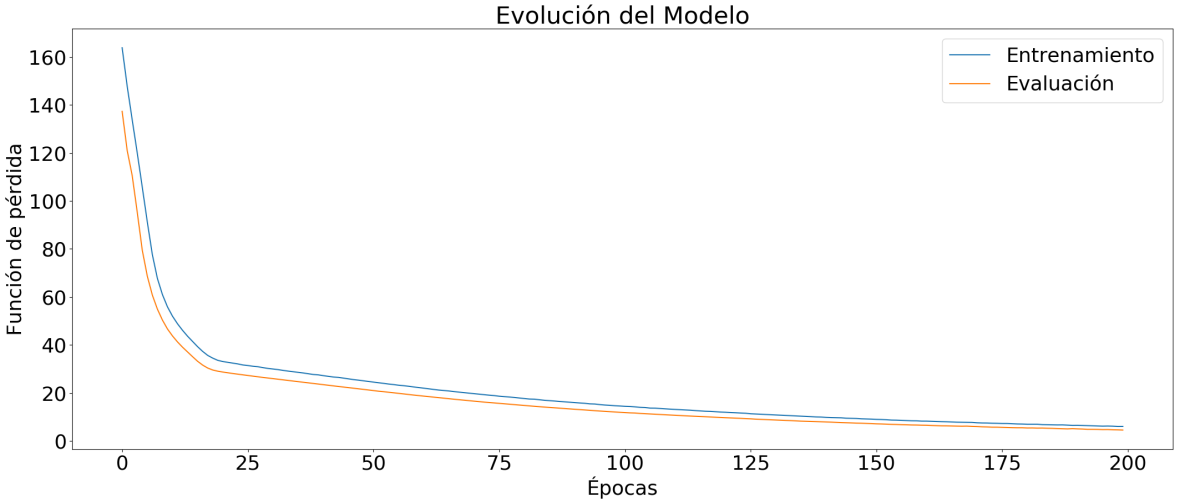

Figura 16: Evolución de la función de pérdida en el escenario 3 con beacons BlueTooth

Se puede apreciar en la Figura [15] que los datos emitidos por las balizas BlueTooth permiten a la red neuronal obtener unos resultados mucho más precisos. Se observa que la precisión, *Accuracy*, es más alta. También es interesante estudiar cómo la evolución de la precisión es mucho más consistente que en el caso de las balizas WiFi. De esto se puede inferir que los datos capturados con las balizas BlueTooth son mucho más ricos que los

emitidos por los puntos de acceso WiFi, y que la red neuronal es capaz de explotar esa riqueza para mejorar la precisión.

Estas mejoras se resumen en los resultados finales, aportando una precisión del $91,14\,\%$ con 360 estimaciones correctas de un total de 395. Sin embargo, para analizar mejor los resultados obtenidos, y ver dónde y cómo se equivoca el algoritmo, se muestra nuevamente una matriz de confusión. En ella se puede observar que el algoritmo se comporta de manera excelente ya que, aún cuando falla, lo hace mayoritariamente con las posiciones adyacentes a la real. En este caso la matriz de confusión es más fiable que en los anteriores escenarios ya que las posiciones son secuenciales. Esto quiere decir que la posición x se encuentra entre $x-1$ y $x+1$.

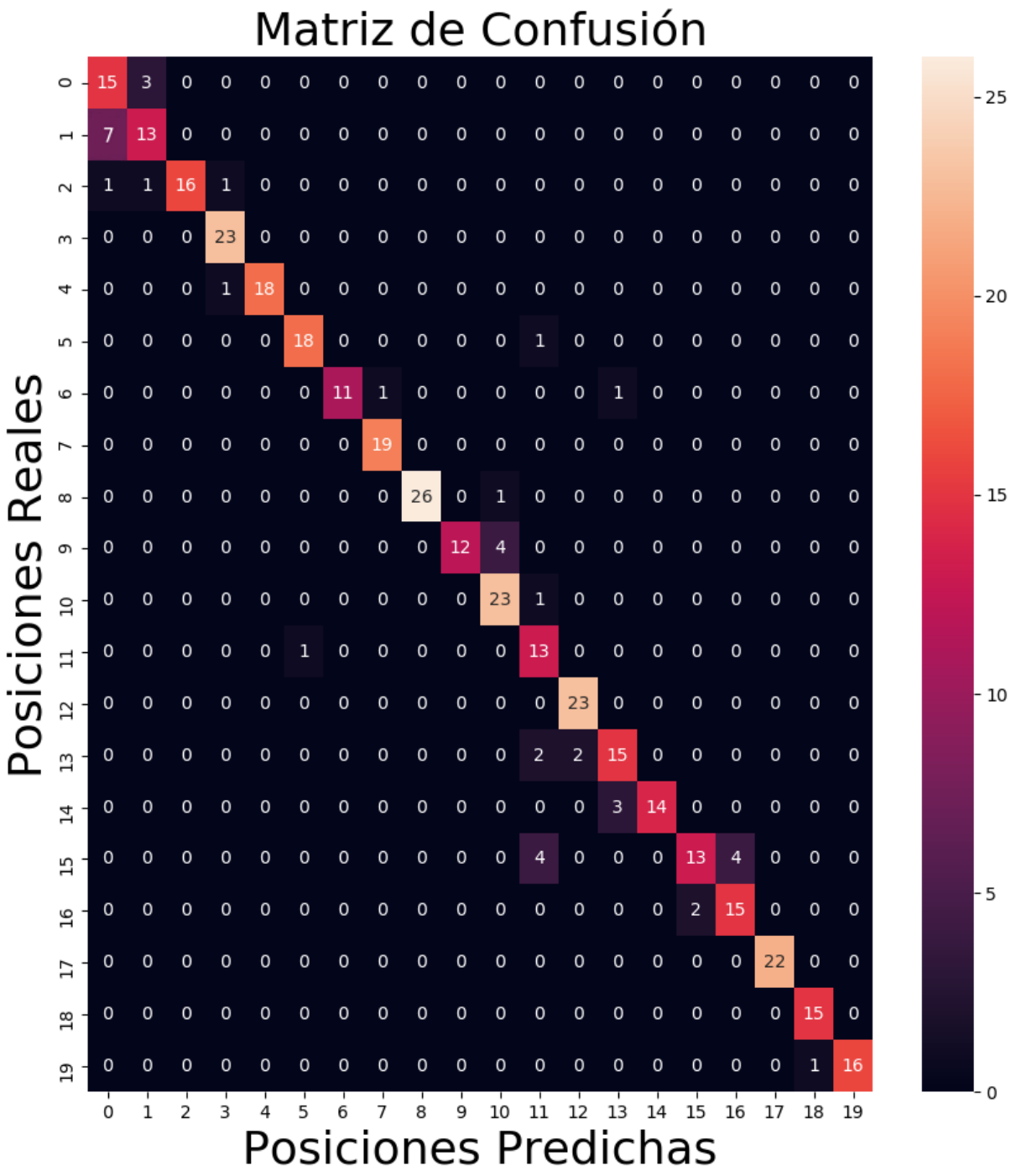

Figura 17: Matriz de Confusión del escenario 3 con beacons BlueTooth

Como se observa en la matriz de confusión la posición que más valores erroneos tiene es la posición cero. Esto es debido a que es el punto más alejado de las balizas, por lo que recibe muy pocos datos. Aún así los resultados en dicha posición son aceptables.

5.6.3 Uso Combinado Beacons WiFi y BlueTooth

Finalmente, como último escenario de este documento, se ha elaborado un Dataset combinado formado por las señales recibidas de WiFi y BlueTooth. Se pretende demostrar, que aunque las señales WiFi aportan mucha menos información que las emitidas por las balizas BlueTooth, la información combinada de ambas puede llevar a mejoras en el rendimiento del algoritmo. Se debe recalcar, sin embargo, que en caso de existir alguna mejora esta será marginal, ya que, el algoritmo actualmente cuenta con una tasa de acierto superior al 90 %.

Tras alimentar el algoritmo con el nuevo conjunto de datos, no se detectan cambios significativos respecto al uso exclusivo de beacons BlueTooth, ni en la evolución de la tasa de acierto, ni en el valor de la función de coste.

Sin embargo, al proceder a la etapa de evaluación, el algoritmo sí que reporta un mejor resultado. Sobre un total de 367 muestras, logra acertar 350, lo que equivale a una tasa de acierto de un $95,37\,\%$. Llegando así a superar en más de un $4\,\%$ los resultados anteriores.

Finalmente, se procede a observar la matriz de confusión para ver si se puede extraer alguna conclusión adicional, Figura [18].

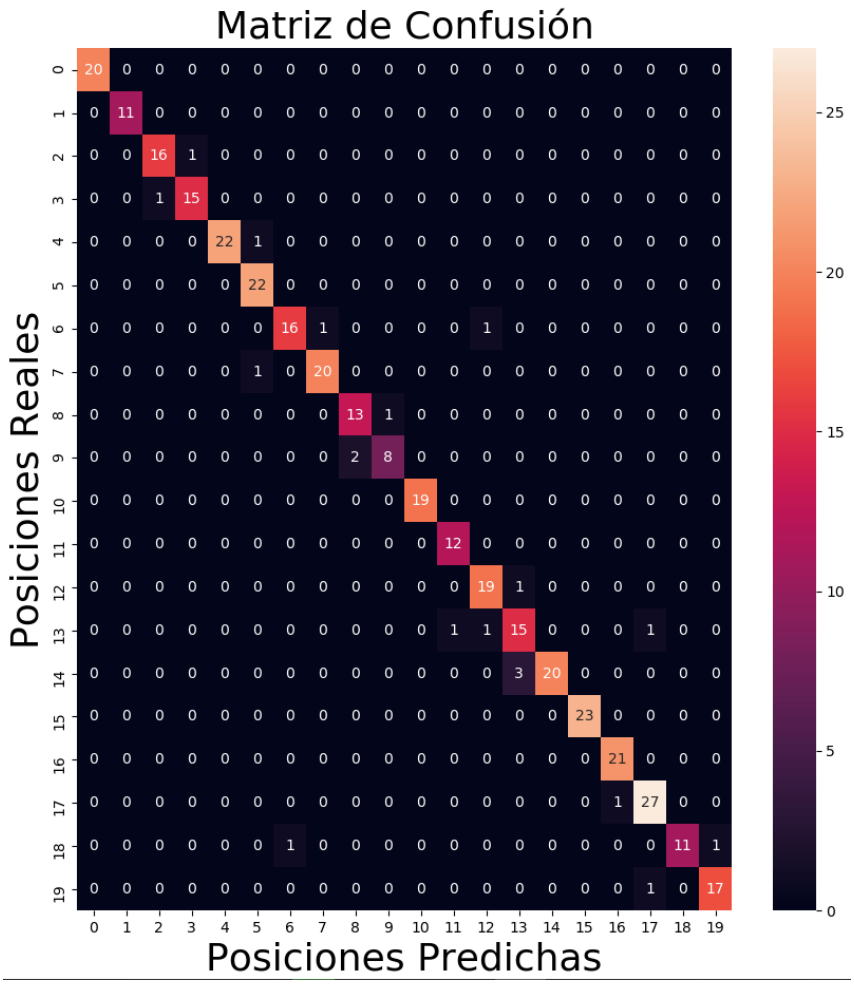

Figura 18: Matriz de Confusión del escenario 3 con beacons WiFi y BlueTooth

Tal y como se puede apreciar, se ha obtenido una gran ganancia en la estimación de la posición 0, que era la que peores resultados ofrecía en el apartado anterior. Sin duda, las señales recibidas por los puntos de acceso han ayudado a refinar la estimación en dicha posición. Adicionalmente, se observa que incluso cuando el algoritmo no es capaz de determinar la posición correctamente, el error se aproxima más a la posición real, por lo que el rendimiento general del algoritmo se ve beneficiado al combinar ambas señales.

6 Conclusiones y líneas futuras

A continuación se procede a detallar las conclusiones que se han obtenido tras la finalización del presente trabajo. También se propondrán una serie de cuestiones que pueden resultar interesantes de cara a futuros trabajos.

6.1 Conclusiones

Más allá de los resultados concretos obtenidos, este trabajo demuestra que las herramientas de aprendizaje autónomo pueden ser la solución para un gran abanico de problemas que permanecen abiertos desde hace tiempo y que técnicas tradicionales no han conseguido resolver, como es el caso de la localización en interiores.

Se ha concluido que la combinación de balizas BlueTooth y técnicas de aprendizaje autónomo otorgan un gran rendimiento, con tasas de éxito superiores al 95 %, en el posicionamiento en interiores. Además el coste de despliegue de la infraestructura no es excesivamente alto.

Alternativamente, se han obtenido resultados aceptables con el uso exclusivo de beacons WiFi. Esta solución, debería por tanto, tenerse en cuenta cuando se cuente con un número elevado de puntos de acceso ya desplegados y los requisitos de precisión no sean excesivamente elevados.

Respecto al uso combinado de las señales WiFi y BlueTooth, arroja resultados satisfactorios. En primer lugar, ha quedado demostrado que ambas señales pueden complementarse para mejorar la precisión donde había pocas señales BlueTooth. Pero también esta combinación ha demostrado su eficacia ayudando a reducir la varianza del error entorno a la posición real.

Finalmente, el coste computacional de efectuar el posicionamiento, una vez completado el entrenamiento, es bajo. Por lo tanto, resulta posible su despliegue, bien mediante aplicación móvil, o mediante servicio web.

6.2 Líneas futuras

La realización de este trabajo ha sido, principalmente, un estudio de las técnicas de aprendizaje autónomo, y una aplicación práctica de las mismas para aportar una solución al

problema de la localización en interiores.

En primer lugar, puede resultar interesante realizar pruebas con múltiples usuarios que, simultáneamente, realicen peticiones a un servidor que almacene el modelo de la red neuronal, ya entrenada, para ver así si las posiciones que retorna a cada usuario son o no acertadas y si las interferencias de esos usuarios pueden hacer caer el rendimiento del algoritmo.

Los resultados expuestos en 5.6.3 sugieren que el uso combinado de distintas señales mejora aún cuando una de ellas no aporta gran información. Este hecho permite que futuras investigaciones indaguen sobre la mejor forma de combinar dichas señales u otras. En esta linea, el análisis combinado de balizas en diferentes frecuencias es posible que reporte mejores resultados.

También puede resultar interesante el estudio sobre el posible impacto en el rendimiento del algoritmo del posicionamiento físico de las balizas, bien sean BlueTooth, WiFi o de otro tipo.

Respecto al tratamiento de los datos, se puede analizar si un preprocesado más intensivo de los mismos puede facilitar la etapa de entrenamiento y mejorar así el rendimiento del algoritmo. Técnicas de *Data Augmentation*, o el uso de redes GAN pueden aportar resultados más consistentes.

Finalmente, en cuanto a implementaciones prácticas se refiere, este tipo de sensores pueden desplegarse, sin un alto coste,en centros comerciales para obtener flujos de tráfico o afluencia. También está justificado su uso en aparcamientos, lo que puede permitir encontrar fácilmente la ubicación del vehículo o de las entradas o salidas.

Referencias

[1] A. Gozalo Madrazo. Estudio y desarrollo de una solución efectiva para el problema del posicionamiento en interiores. Master's thesis, Octubre 2017.

[2] *Global Positioning System Standard Positioning Service Signal Specification*. U.S. Gobernment, 1995.

[3] M. Higuera Fuentevilla. Sistemas de geolocalización wifi. Technical report, Cantabria, España, 2010.

[4] J. Carrasco Alonso. Desarrollo de una herramienta para la localización de dispositivos en entornos inalámbricos, 8 2016.

[5] P. Fernández Fernández. Aplicación de tecnologías de realidad virtual en el posicionamiento en interiores. Technical report, Cantabria, España, 2016.

[6] Htc vive. `https://www.vive.com`. Último acceso: 17-07-2018.

[7] Las 9 tareas del Machine Learning. `https://blogthinkbig.com/tareas-machine-learning`. Último acceso: 17-07-2018.

[8] DeepMind. `https://deepmind.com/`. Último acceso: 17-07-2018.

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 12 2017.

[10] Official Stockfish Repository. `https://github.com/official-stockfish/Stockfish`. Último acceso: 17-07-2018.

[11] J. H. Holland. Genetic algorithms computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. *Scientific American*, 267(1):66–73, 07 1992.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software disponible en tensorflow.org.

[14] F. Chollet et al. Keras. `https://keras.io`, 2015.

[15] G. Rossum. Python reference manual. Technical report, Amsterdam, Paises Bajos, 1995.

[16] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[17] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

Anexo I: captura_wifi.py

```
#!/usr/bin/env python3

import csv
import sys, os
import pandas as pd
import numpy as np
from scapy.all import *

ap_list = []
potencias = {}

dataframes = []
maxpos = 44

def packetHandler(pkt):
    if pkt.haslayer(Dot11):
        if pkt.type == 0 and pkt.subtype == 8:
            try:
                extra = pkt.notdecoded
                rssi = -(256-ord(extra[-4:-3]))
            except:
                rssi = -100
            if pkt.addr2 not in ap_list:
                ap_list.append(pkt.addr2)
                potencias[pkt.addr2]=[]
                print ("AP MAC: %s with SSID: %s" %(pkt.addr2, pkt.info))
            potencias[pkt.addr2].append(rssi)

pos = int(sys.argv[1])

if (pos >= 0) and (pos < maxpos):
    sniff(iface="mon0", prn = packetHandler,timeout=60)

    for beacon in ap_list:
        dataframes.append(pd.DataFrame(potencias[beacon],columns=[beacon]))

    df = pd.concat(dataframes, axis=1)
    df.fillna(-120, inplace=True)
```

```
    df['Pos'] = pd.Series(pos*np.ones(len(df[ap_list[0]])), index=df.index)

    df.to_csv("./Datos/medidas{}.csv".format(pos), sep='\t')
    print("Datos correctamente guardados.")
else:
    print("No se ha introducido un valor de posición correcto")
```