Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Implementación Android de algoritmos basados en información heurística y técnicas *Map Matching* para la calibración de balizas US

Autor: Rubén Cervigón Rey

Tutora: Mª del Carmen Pérez Rubio

Cotutor: David Gualda Gómez

2019

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Máster Universitario en Ingeniería Industrial

Trabajo Fin de Máster

Implementación Android de algoritmos basados en información heurística y técnicas *Map Matching* para la calibración de balizas US

Autor: Rubén Cervigón Rey

Directores: Mª del Carmen Pérez Rubio

David Gualda Gómez

Tribunal:

Presidente: Dª. Ana Jiménez Martín

Vocal 1º: D. Antonio de Vicente Rodríguez

Vocal 2º: Dª. Mª del Carmen Pérez Rubio

CALIFICACIÓN: .....

FECHA: .....

A los míos. . .

. . . en especial por los que faltan, para que donde quiera que estéis, os sintáis orgullosos de mí.

"La muerte no existe, la gente sólo muere cuando la olvidan"
Isabel Allende

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

Para empezar, me gustaría agradecer a Jesús Ureña Ureña, por haberme brindado la oportunidad de realizar este trabajo dentro de un grupo de investigación magnífico y del que salgo totalmente satisfecho por el trabajo realizado y todo lo aprendido. Gracias a mi tutora, Mª del Carmen Pérez Rubio, y a mi cotutor, David Gualda Gómez, por la forma en la que me han ido guiando, ayudando a resolver todo tipo de dudas y aportarme sugerencias a lo largo de la elaboración de este trabajo. Nunca os podré agradecer todo lo que me habéis enseñado en mis dos etapas en el grupo.

En segundo lugar, agradecer a todos los integrantes del grupo de investigación GEINTRA y en concreto al grupo US&RF, por estos ocho meses increíbles en los que he adquirido un montón de cosas de cada uno de ellos, no únicamente en aspectos técnicos, sino también en aspectos como la amistad, el trabajo en equipo o el esfuerzo de superación, valores que considero fundamentales.

Por último, que no por eso menos importante, dar las gracias a mi madre, hermana y amigos que han estado conmigo durante estos intensos años universitarios, compartiendo buenos momentos y sobre todo ayudándome a superar aquellos no tan buenos. Sin vosotros todo lo conseguido no hubiera sido posible.

Resumen

Este Trabajo Fin de Máster desarrolla e implementa sobre una plataforma Android un algoritmo para la calibración tanto de motas ultrasónicas individuales como de balizas ultrasónicas LOCATE-US, modelo diseñado en el grupo GEINTRA y que consta de 5 transductores ultrasónicos. La herramienta permite calibrar la posición de balizas situadas tanto en el techo como en las paredes y aunque se ha desarrollado pensando en su uso con tecnología ultrasónica, puede utilizarse con otro tipo de balizas como radiofrecuencia, infrarrojos, etc.

El fundamento de la calibración es la estimación de la posición de la baliza con respecto a la posición de referencia del sistema. La calibración propuesta consiste en el uso de las distancias de la baliza de estudio a las paredes cercanas medidas mediante un medidor laser. La información de dichas distancias, el mapa en formato vectorial y otros datos heurísticos (como la región de ubicación aproximada de la baliza, la orientación aproximada de las medidas y las características de las paredes de las cuales se han obtenido la distancia de la baliza a ellas) son las entradas del algoritmo propuesto que obtiene la mejor estimación de la posición de la baliza. Dado que no hay una solución cerrada al problema, se ha implementado una minimización numérica basada en el uso de algoritmos genéticos (GA) y de búsqueda armónica (HS). La propuesta se ha validado con simulaciones y experimentos reales, obteniendo correctamente la posición de las balizas incluso en entornos complejos, con paredes que presentan un cierto grado de curvatura.

Palabras Clave: Calibración; ULPS; GA; HS; Android.

Abstract

This study presents an algorithm developed on an Android-based platform for calibrating the position of beacons which are placed both on the ceiling and on the wall of an indoor environment. Specifically, the beacon structure to be calibrated was developed in the GEINTRA group and called LOCATE-US. It consists of five ultrasonic transducers distributed around a square structure of 70.7x70.7cm. The application also allows to calibrate individual ultrasonic spots.

The notion of calibration is to estimate the position of a beacon related to a known reference system of a map. The calibration proposal consists of using several distances from the beacon to the neighbor walls measured by a laser meter. The information of these distances, the map in vector format and other heuristic data (such as the approximate localization region of the position of the beacon, the approximate orientation of the measurements and the features of the walls from which the laser meter is projected) are the inputs of the proposed algorithm that obtains the best estimation of the beacon's position. Due to the fact that there is not an analytical solution, we have implemented a numerical minimization based on the use of a Genetic Algorithm (GA) and a Harmony Search (HS) methods. The proposal has been validated with simulations and real experiments, obtaining the position of the beacon with great accuracy.

Keywords: Calibration; ULPS; GA; HS; Android.

Índice

Índice de figuras

Índice de tablas

Glosario de acrónimos y símbolos

BW	Ancho de banda de ajuste al tono.
GA	Algoritmo Genético (*Genetic Algorithm*).
GNSS	Sistemas de Navegación Global por Satélite.
GPS	Sistema de Posicionamiento Global.
HMCR	Tasa de consideración de la memoria armónica.
HM	Memoria Armónica.
HMS	Tamaño de la memoria armónica.
HS	Algoritmo Búsqueda Armónica (*Harmony Search*).
IA	Inteligencia Artificial.
IHS	Algoritmo Búsqueda Armónica mejorada (*Improve Harmony Search*).
IR	Radiación Infrarroja.
LBS	Servicios Basados en Localización.
LPS	Sistemas de Posicionamiento Local.
NI	Número de improvisaciones/iteraciones.
PAR	Tasa de ajuste al tono.
RF	Radio-Frecuencia.
TDOA	Diferencia tiempo de vuelo (*Time Difference Of Arrival*).
U-LPS	Sistemas de Posicionamiento Local Ultrasónico.
US	Señales Ultrasónicas.

1. Introducción

a. Contexto y objetivos del Trabajo Fin de Máster

La creación de espacios dotados de "inteligencia ambiental" es factible hoy en día gracias a los avances en sensores, comunicaciones inalámbricas y capacidad de computación de dispositivos móviles. En los espacios inteligentes es fundamental su capacidad para detectar la presencia de agentes (personas, vehículos autónomos, robots, drones) y ofrecer servicios basados en su localización (LBS). El enorme potencial de desarrollo de los LBS en dispositivos portables, tipo *smartphones* o *tablets*, ha supuesto una revolución e impulso para la investigación en sistemas de posicionamiento y navegación. Aun con una extensa labor investigadora previa, los sistemas de posicionamiento local (LPS) existentes no han alcanzado las prestaciones de disponibilidad, precisión y robustez de los sistemas satelitales para exteriores [Lopes, 2014].

Hoy en día los Sistemas de Navegación Global por Satélite (GNSS), como el GPS estadounidense, el GLONASS ruso o el programa GALILEO europeo, proporcionan datos de posición con precisiones del orden de 10m, permitiendo la ubicación en espacios exteriores, así como trazar diferentes rutas para llegar a otras ubicaciones. Sin embargo, estos sistemas no son adecuados para aplicaciones interiores, ya que por lo general estas señales no son lo suficientemente potentes para penetrar paredes y techos, además de proporcionar precisiones inferiores a las requeridas en este tipo de ámbitos (precisiones del orden de centímetros).

Dado que gran parte de nuestro tiempo lo pasamos precisamente en espacios interiores, resultan comprensibles los esfuerzos actuales en el desarrollo de tecnologías de localización precisa (centimétrica, fundamental en aplicaciones de robótica móvil) de agentes móviles dada la variedad de aplicaciones que se podrían ofrecer a partir del posicionamiento de los mismos: navegación guiada en interiores de edificios (aeropuertos, hospitales, fábricas…), aplicaciones de realidad aumentada para turismo cultural, juegos, estudio de puntos de interés y patrones de movimiento con objeto de mejorar los servicios o inserción de publicidad personalizada [Filonenko, 2013].

En la actualidad, los sistemas de posicionamiento en interiores (IPS) se basan en diferentes tecnologías como la radiofrecuencia (RF) [Bahl, 2000][Niu, 2015][Xiao, 2014][Belmonte, 2018], infrarrojos (IR) [Aitenbichler, 2003][Gorostiza, 2011], cámaras [Li, 2010][Opdenbosch, 2014] o ultrasonidos (US) [Ureña, 2007][Ureña, 2015], entre otros. Cabe destacar que frente a tecnologías basadas en señales de oportunidad (tipo WiFi) están otras, como los ultrasonidos o las cámaras, basadas en infraestructura externa situada en los espacios interiores (techos, paredes, columnas, etc.), que por lo general alcanzan precisiones mejores.

En este sentido, el grupo de investigación GEINTRA (Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte, http://www.geintra-uah.org/) y en concreto el subgrupo US&RF, en el que se enmarca el presente Trabajo Fin de Máster (TFM), plantea un Sistema de Posicionamiento Local Ultrasónico (U-LPS, al que se le ha llamado LOCATE-US) para conseguir una localización precisa de elementos móviles en entornos interiores, como se muestra en la siguiente figura.

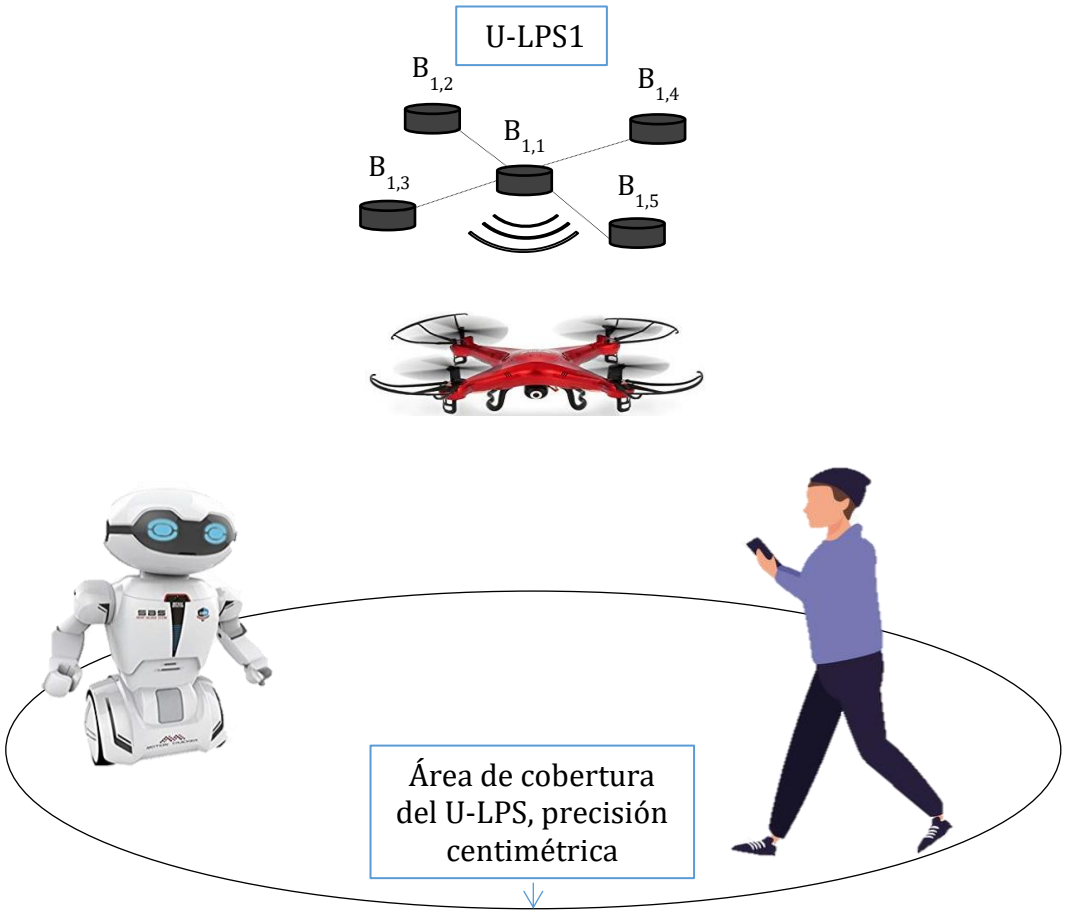


Figura 1: Sistema de Posicionamiento Local Ultrasónico (LOCATE-US) propuesto por el grupo GEINTRA US&RF.

Este sistema propone la localización centimétrica en entornos interiores extensos mediante el procesamiento de señales ultrasónicas (US) emitidas por un conjunto de sistemas U-LPSs ubicados en el entorno [Pérez, 2016]. Con respecto a trabajos anteriores, destacar las ventajas que se consiguen con esta propuesta, como la utilización únicamente de señales ultrasónicas frente a trabajos basados en señales de radio frecuencia (RF), que consiguen menores precisiones.

Más en concreto, el Sistema de Posicionamiento Local Ultrasónico aquí presente está basado en el procesamiento de las señales ultrasónicas, que requiere la correlación con los códigos emitidos por las balizas y la obtención de las diferencias de distancias a partir de las diferencias de tiempo de vuelo (TDOA, *Time Difference Of Arrival*) de las señales emitidas por cada uno de los cinco emisores que lo componen ($B_{1,1}$, $B_{1,2}$, $B_{1,3}$, $B_{1,4}$, $B_{1,5}$, expuestos en la Figura 1), ya que las diferencias de distancias son directamente las diferencias de tiempos de llegada multiplicadas por un factor que relaciona la velocidad del sonido en el aire y la frecuencia de muestreo de las señales recibidas. Esta técnica se basa en medir las diferencias de tiempo existentes entre las señales recibidas tomando una de ellas como referencia. Estas diferencias de distancia obtenidas a través de los US son la diferencia de las distancias euclídeas entre la posición del agente móvil y la de los transductores del U-LPS, por lo que a partir del conocimiento de las mismas y de la posición exacta de uno de los emisores de la baliza (por geometría se conoce la posición del resto de emisores) se puede realizar el posicionamiento.

Cuando esos sistemas son instalados, es necesario por tanto conocer la posición del elemento IPS (en nuestro caso los transductores/emisores de la baliza) con respecto a la posición de referencia del sistema (a ese proceso se le denomina calibración). En esa idea va dirigida una de las líneas de trabajo del proyecto TARSIUS (en el que se enmarca el presente Trabajo Fin de Máster), que aborda la mejora y el robustecimiento de las debilidades actuales de los sistemas de posicionamiento local (LPS) y su uso en aplicaciones de alto nivel con demanda futura. La idea es mejorar la modularidad, coste, facilidad de despliegue y mantenimiento de las tecnologías LPS.

Específicamente, la realización de este TFM se centra en la estimación automática de la posición de las balizas en un dispositivo portable. Para ello plantea un sistema de autocalibración (estimación automática de la posición de las balizas) de balizas, mediante el desarrollo y optimización de algoritmos de autocalibración y su implantación sobre plataforma Android, proporcionando al usuario una aplicación basada en los mapas de Google que obtenga la posición de la baliza ultrasónica, a partir de unos datos de entrada introducidos por el usuario. Esta implementación sobre plataforma Android es uno de los incentivos de este trabajo frente a estudios anteriores implementados sobre otras plataformas, como Matlab [Gualda, 2019] debido al amplio abanico de opciones que se abren al tener el algoritmo en funcionamiento en un dispositivo móvil a tiempo real.

b. Revisión de sistemas para la calibración de balizas y propuesta planteada

Este trabajo propone una aplicación sobre plataforma Android para la calibración de la posición bien de motas individuales como el conjunto de transductores de balizas LOCATE-US situadas tanto en el techo como en las paredes de espacios interiores.

El propósito del conocimiento de la posición de las balizas es para usar esa información en situaciones de posicionamiento en interiores, en las cuales las balizas pueden usarse en diferentes tecnologías como radiofrecuencia, infrarrojos, ultrasonidos, etc. En el proyecto TARSIUS en el que se enmarca el presente TFM, se necesita conocer la posición de las balizas que componen un conjunto de sistemas de posicionamiento local ultrasónicos situados en diferentes puntos estratégicos del edificio (como pueden ser zonas en las que se produzcan giros en las trayectorias de los usuarios, para corregir la posición. Así, en este TFM se plantea una aplicación Android que permita la cómoda calibración de las balizas desde un dispositivo portable (móvil, tableta o similar).

La calibración se puede llevar a cabo de manera manual usando una plomada láser para determinar la proyección vertical de las balizas en el suelo y resolviendo la posición 2D de las balizas mediante varias distancias del punto proyectado a posiciones de referencia conocidas del espacio interior en el que se encuentre, como

esquinas o puntos específicos de la pared. Las distancias de un punto a otro del espacio bien sean paredes o techo, se suelen obtener con un medidor láser. Otra forma de obtener la posición es mediante el posicionamiento inverso, es decir, se miden distancias a puntos conocidos del espacio y se obtiene la posición de las balizas utilizando un algoritmo de localización para resolver el sistema de ecuaciones no lineal [Mahajan, 1999]. El problema principal de este método es que se requiere el conocimiento preciso de un alto número de puntos en el espacio para obtener una correcta obtención de la posición.

Otros métodos más autónomos se basan en el uso de un robot móvil (MR) con un sistema de cálculo para calibración de varias balizas que componen una estructura [Ureña, 2011]. Sin embargo, cuando el área de localización es grande y hay algunas estructuras de balizas para cubrir completamente la zona [Ruiz, 2013], el esfuerzo de calibrar todas las balizas es bastante alto, y más en el caso de que la posición de los nodos sea dinámica y pueda cambiar a lo largo del tiempo. En [Ureña, 2015] la solución es el empleo de un MR para autocalibrar todas las estructuras, pero el problema es que el error relacionado con la posición de las balizas se incrementa acumulativamente cuando el MR se desplaza entre sistemas de localización.

Este TFM utilizará una calibración *Map Matching* de las balizas. El fundamento de este tipo de calibración radica en la combinación de medidas y toda la información heurística relacionada con el mapa para la estimación de la posición de las balizas.

Este método consiste en el uso de las distancias tomadas desde la proyección vertical de uno de los transductores de la baliza en el suelo a las paredes cercanas (tomadas con un medidor laser a partir de las proyecciones horizontales tanto frontal como laterales de la plomada). Dichas distancias, la información del mapa (información vectorial de las paredes en las cuales se proyecta tanto frontal como lateralmente la plomada láser, en este caso un mapa en formato XML, desarrollado previamente en [García, 2011]) y otros datos heurísticos (como la región aproximada donde se ubica la baliza y la orientación aproximada del láser, que es el ángulo formado entre la pared 1 y el semieje positivo de abscisas del edificio) son las entradas del algoritmo propuesto que obtiene la mejor aproximación de la posición de la baliza.

Dado que para este problema no hay una solución analítica, se propone la implementación de un método de minimización numérica, mediante el uso de algoritmos metaheurísticos, para la estimación de la misma.

Una vez estimada la posición del transductor a partir del cual se han tomado las medidas de las distancias (en la aplicación se ha programado para que ese transductor sea el central de la baliza) con el método de minimización numérica elegido, la posición del resto de transductores de la baliza se obtendrá por geometría al conocer las dimensiones de la misma (para más información acerca de las balizas ultrasónicas, del sistema LOCATE-US, con las que se ha trabajado en el presente TFM véase el apartado de descripción del material utilizado).

c. Revisión a los algoritmos metaheurísticos

En programación se dice que un algoritmo es heurístico cuando la solución no se determina en forma directa, sino mediante ensayos, pruebas y reensayos. Más en concreto en [Zanakins, 1981] se describe un heurístico como un "procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido".

Los métodos heurísticos se utilizan, por ejemplo, cuando no existe un método exacto de resolución, cuando existe un método exacto que consume mucho tiempo para ofrecer la solución óptima, cuando existen limitaciones de tiempo o como paso intermedio para obtener una solución inicial para la aplicación de otra técnica.

Dentro de este área, el término metaheurístico lo introdujo Glover (1986) al definir una clase de algoritmos de aproximación que combinan métodos heurísticos tradicionales con estrategias eficientes de exploración del espacio de búsqueda [Blum, 2003].

Con la introducción de las técnicas de optimización metaheurísticas [Glover, 1986], se originó un nuevo campo de investigación en el área del tratamiento del cálculo. Las técnicas metaheurísticas constituyen estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales, con un alto rendimiento, y que comprenden un conjunto de técnicas iterativas, generalmente estocásticas, las

cuales actúan sobre una población de individuos o agentes simples que son evolucionados o modificados mediante una serie de reglas previamente especificadas [Melian, 2003].

La metaheurística realiza una búsqueda más minuciosa que la que ofrece el método heurístico (aunque tampoco garantiza la obtención del óptimo del problema considerado) y es realmente útil en problemas de optimización combinatoria, problemas en las que las variables de decisión están acotadas y el espacio de soluciones se encuentra formado por ordenaciones de valores de dichas variables.

Las técnicas de optimización metaheurísticas incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otras. Algunos ejemplos de dichas técnicas son: Colonia de Hormigas [Allaoua, 2009], Evolución Diferencial [Vaisakh, 2008], Algoritmo Genético (GA) [Holland, 1975], Búsqueda Tabú [Abido, 2002], enjambre de Partículas [Abido, 2002] y Algoritmo de Búsqueda Armónica (HS) [Zong, 2001][Wang, 2010].

La lógica de las diferentes técnicas metaheurísticas es similar: el punto de partida es una solución (o conjunto de soluciones) que típicamente no es óptima. A partir de ella se obtienen otras parecidas, de entre las cuales se elige una que satisface algún criterio (que determina a esa solución mejor que al resto), a partir de la cual comienza de nuevo el proceso. Este proceso se detiene cuando se cumple alguna condición establecida previamente.

Destacar que todas las técnicas metaheurísticas tienen las siguientes características:

- Son ciegas, no saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos (es decir, se trata de procesos de búsqueda en los que cada nueva solución no es necesariamente mejor, en términos de la función objetivo, que la inmediata anterior). Algunas veces

aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.

- Son relativamente sencillos. Todo lo que se necesita es una representación adecuada del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para evaluar el campo de soluciones.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.
- La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma (en general, no lo será).

Aunque las soluciones que ofrecen las técnicas metaheurísticas no son las óptimas y, en general, ni siquiera es posible conocer la proximidad de las soluciones al óptimo, permiten estudiar problemas de gran complejidad de una manera sencilla y obtener soluciones suficientemente buenas en tiempos razonables. Además, destacan por su gran flexibilidad frente a otros métodos, lo que permite usarlos para abordar una amplia gama de problemas.

En este TFM se van a utilizar algoritmos genéticos (GA) y algoritmos de búsqueda armónica (HS). Se han elegido estos métodos de minimización, debido a que uno es el más extendido en la literatura (GA) y el otro el que obtiene resultados más precisos (HS), pero se podrían haber usado otro tipo de algoritmos que también son capaces de resolver este tipo de problemas. Destacar que, para la obtención de una posición lo más fidedigna posible a la realidad, es más importante la disposición de datos correctos a la entrada del algoritmo (los valores de las distancias a las paredes y la orientación de la plomada láser, y la selección sobre el mapa de la región aproximada y de las paredes en las cuales ha proyectado el láser) que el propio algoritmo utilizado en sí.

d. Estructura del documento

Este documento está dividido en 11 capítulos, organizados como se explica a continuación:

El capítulo 2 expone la base teórica sobre la que se ha basado este trabajo (algoritmos metaheurísticos); Las principales características del material utilizado se describen en el capítulo 3; El capítulo 4 explica todo lo relacionado con la programación de la aplicación Android desarrollada; En el capítulo 5 se evalúan los algoritmos propuestos con diferentes pruebas experimentales; Finalmente para acabar con la parte de la memoria del proyecto se presentan, en el Capítulo 6, las conclusiones obtenidas con la realización del mismo.

En los últimos capítulos, se describen otras cuestiones del proyecto como las partes más importantes del código desarrollado (Capítulo 7: Planos), una serie de condiciones para el funcionamiento correcto de la aplicación (Capítulo 8: Pliego de condiciones), el coste (Capítulo 9: Presupuesto), el manual de usuario de la aplicación (Capítulo 10) y la bibliografía (Capítulo 11).

2. Base teórica

a. Descripción del algoritmo genético

La Inteligencia artificial (IA) es el concepto según el cual las máquinas tienden a pensar y adaptarse como los seres humanos, y se centra en la creación de programas que pueden mostrar comportamientos considerados inteligentes. Los investigadores en el campo de la Inteligencia Artificial han tratado de recrear, con algoritmos y fórmulas matemáticas, el funcionamiento del cerebro humano. Si bien la Inteligencia Artificial ha sido capaz de resolver problemas como jugar al ajedrez, optimizar el tráfico aéreo y dirigir la compraventa de acciones de bolsa, a lo largo de los años ha ido degenerando y alejándose de su premisa inicial. Hoy en día esta Inteligencia Artificial *mainstream* tiene más que ver con la Minería de Datos, la Robótica y la Teoría de Grafos que con la neurología [Robologs, 2015].

En contraposición a la IA *mainstream*, durante la década de los ochenta surgieron nuevas técnicas como la ingeniería neuromórfica, la inteligencia de enjambre o la robótica evolutiva. La Inteligencia Artificial Bioinspirada se basó en ellas y cuestionó la validez de los métodos hasta entonces utilizados para intentar construir máquinas que emulasen a organismos, sistemas y procesos biológicos que durante miles de años han permitido a los organismos vivos adaptarse y sobrevivir, siendo técnicas especialmente eficaces para resolver problemas mal definidos, trabajar con información corrupta o adaptarse a entornos cambiantes.

Los Algoritmos Genéticos o Evolutivos probablemente sean una de las ideas más intuitivas del campo de la IA, ya que se inspiran en la evolución natural para solucionar problemas de optimización que de otra forma serían mucho más tediosos.

Estos algoritmos simulan la selección natural y la adaptación encontrada en la naturaleza. Es decir, trabajan con una población de individuos, cada uno de los cuales representa una solución al problema dado. A cada individuo se le asigna un valor de aptitud (llamado *fitness*), relacionado con la adaptación de dicha solución al problema. Cuanto mayor sea la adaptación de un individuo, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse y por tanto de que

su material genético se propague en sucesivas generaciones, combinando su material genético con otro individuo seleccionado de igual forma. Favoreciendo el cruce de los individuos mejor adaptados, se consigue focalizar las posibles soluciones a las áreas más prometedoras del espacio de búsqueda. Esta combinación producirá nuevos individuos (descendientes de los anteriores), que compartirán algunas de las características de sus padres. De esta manera, a través de los denominados operadores genéticos (selección, recombinación y mutación) se produce una nueva población de posibles soluciones, las cuales, por lo general, cada vez serán mejores. Este proceso continúa hasta que se cumple algún criterio de parada establecido.

A continuación, en la Figura 2 se representa un ejemplo ilustrativo del funcionamiento del algoritmo genético, en el cual se pretende la búsqueda de una estrella naranja, a partir de una población inicial.

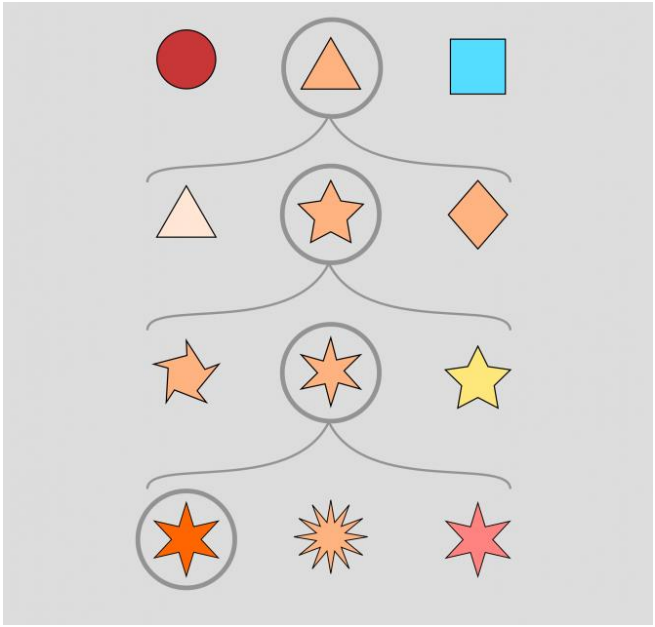


Figura 2: Ejemplo ilustrativo del funcionamiento de un Algoritmo Genético [Robologs, 2015].

En la imagen se puede observar como el algoritmo genético va seleccionando la figura que más se parece a la que se busca, creando en cada generación una nueva población de individuos con los rasgos de los cromosomas más aptos de la generación anterior, acercándose cada vez más a su objetivo.

La estructura de un Algoritmo Genético es un proceso iterativo que actúa sobre una población de individuos. Cualquier algoritmo genético puede resumirse en una serie de pasos. A continuación, se muestra en la Figura 3 un esquema con las diferentes etapas del algoritmo genético.

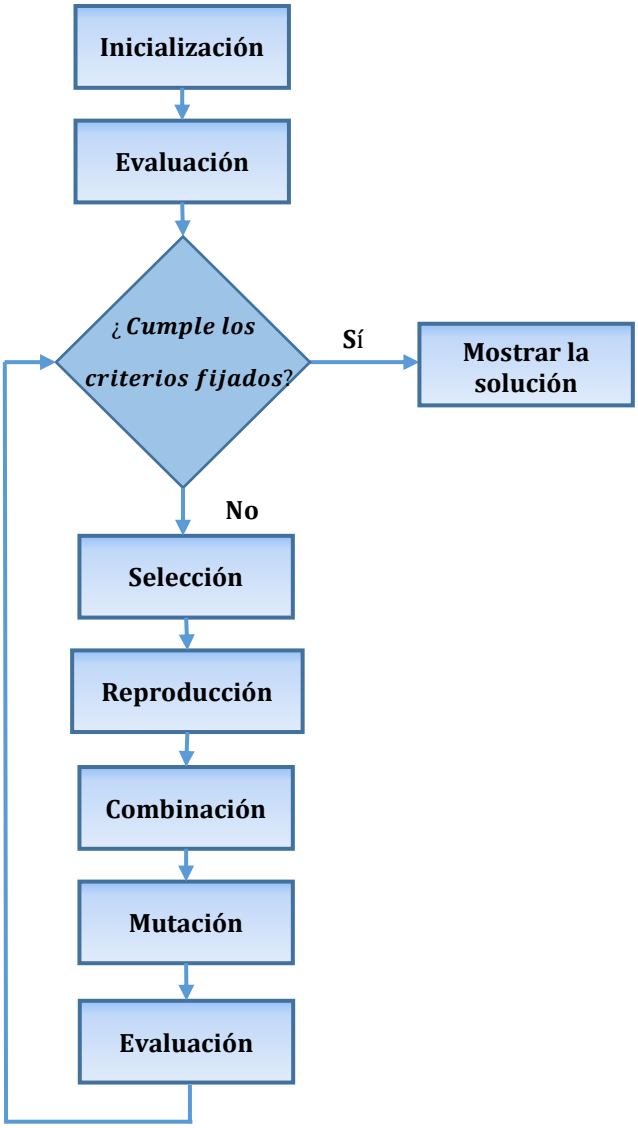


Figura 3: Diagrama de flujo del funcionamiento del Algoritmo Genético.

Una vez mostrado el esquema con los pasos generales del algoritmo genético, se procede a profundizar más cada uno de ellos.

- **Inicialización**:

Consiste en la búsqueda de parámetros del algoritmo y la creación de la primera generación de cromosomas (soluciones posibles al problema en cuestión).

- **Evaluación (Función *Fitness*):**

La Función de *Fitness* evalúa cada uno de los individuos de la población y les asigna un valor numérico según su calidad. Este valor numérico se llama fitness y sirve para determinar las mejores soluciones de dicha generación.

Si alguna de esas soluciones cumple con los criterios del algoritmo se termina el mismo mostrando el mejor cromosoma (mejor solución). Si no se satisfacen los términos marcados, el siguiente paso es el de la selección de los cromosomas para la creación de una nueva generación.

- **Selección y Reproducción:**

Hay varias formas de seleccionar los individuos para crear una nueva población. Algunas de las más utilizadas son:

  - Selección proporcional: cuanto más alto sea el *fitness* de un individuo, mayor será la probabilidad de que pase a la siguiente generación.
  - Selección por torneo: se eligen varios individuos al azar de la población y se compara su fitness. El individuo con el fitness más alto pasará a la siguiente generación. Esto se hace en ocasiones en las que es interesante escoger soluciones que no son tan buenas para poder mantener una buena variabilidad genética entre los individuos. Así al mezclar el material genético se exploran soluciones muy distintas.
  - Selección por rango: los individuos se ordenan en función de su fitness, y la probabilidad que tienen de reproducirse va según su posición.

Otro término importante a la hora de la reproducción, además de la elección del método de selección de los individuos “padres”, es el porcentaje de los mismos que son seleccionados. Ese valor, se denomina presión e indica que cuanto más alta sea, menos individuos serán seleccionados para crear la siguiente generación.

- ***Crossover* y Mutación:**

Una vez se han seleccionado los mejores individuos de la población, se aplican varios operadores genéticos (recombinación y mutación) para generar la nueva población de soluciones.

La recombinación, o más conocida como *Crossover* consiste en mezclar el material genético de los individuos seleccionados de la población anterior para crear los nuevos individuos (véase la Figura 4, donde se ilustra la explicación del entrecruzamiento cromosómico desarrollada por Thomas Hunt en 1916).

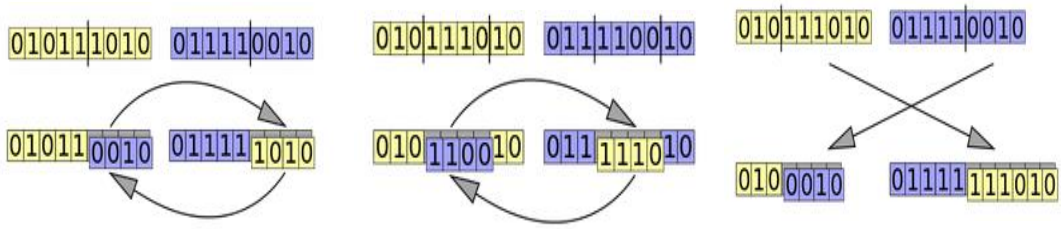


Figura 4: Ejemplo de cómo se realiza el crossover de cromosomas [Fisher, 1997].

Sin embargo, el *crossover* no ayuda a encontrar nuevas soluciones al problema, es por eso que además también se muta el material genético de los nuevos individuos. La mutación consiste en alterar mediante pequeñas variaciones al azar el valor de uno o más genes del cromosoma, consiguiendo mantener la diversidad de una población a la siguiente.

Se recomienda que en la nueva población haya una copia exacta, sin modificar ni cruzar, de cada uno de los individuos elegidos durante la selección. De esta forma, si por alguna razón la nueva población de individuos resulta ser peor que la anterior, por lo menos no se pierden soluciones que hasta entonces eran buenas.

Una vez aplicados los operadores genéticos, la nueva generación de cromosomas es evaluada y se comprueba si cumple los criterios exigidos. En caso de que no los cumpla, la generación de nuevos cromosomas continua con los pasos descritos anteriormente (véase Figura 3, diagrama de flujo del GA) hasta que el criterio de fin se verifique, momento en el que el algoritmo mostrará la mejor solución del problema.

Para finalizar con la descripción de los algoritmos genéticos se muestra en la Figura 5 un ejemplo sencillo con la idea seguida en el presente TFM para la implementación de los algoritmos genéticos en nuestro problema en cuestión.

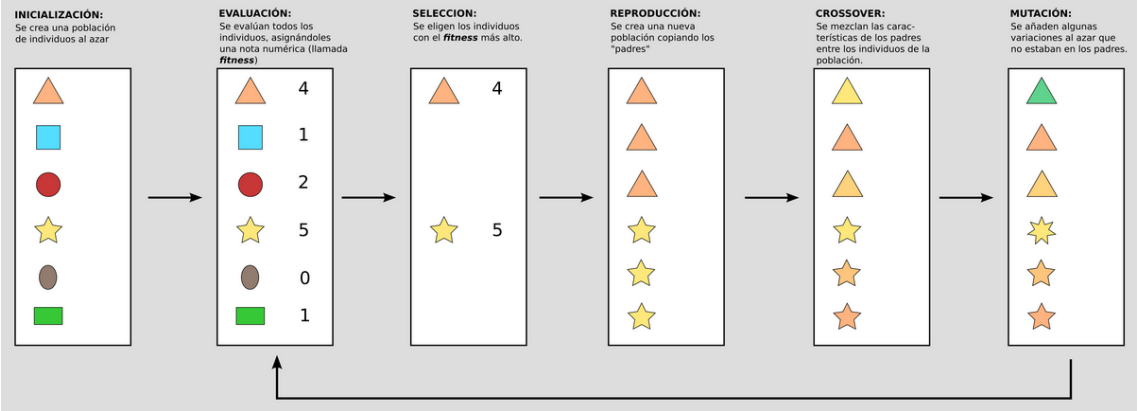


Figura 5: Ejemplo del funcionamiento de un Algoritmo Genético [Robologs, 2015].

b. Descripción del algoritmo de búsqueda armónica

El algoritmo de búsqueda armónica (HS) es una técnica metaheurística basada en población, e inspirada originalmente en el proceso de improvisación y afinación llevado a cabo por un conjunto o agrupación de músicos en la búsqueda de la armonía musical [Cobos, 2011].

La improvisación musical es un proceso en el cual la experiencia y el conocimiento previo de las armonías aportan calidad de la pieza que se está tocando. Por tanto, cuando un músico está improvisando, realiza las siguientes acciones:

- Toca alguna melodía conocida que ha aprendido anteriormente.
- Toca algo parecido a la melodía anteriormente mencionada, ajustándola poco a poco al tono deseado.
- Compone una nueva melodía basándose en sus conocimientos musicales para seleccionar nuevas notas aleatoriamente.

La optimización es uno de los campos más importantes en ciencia e ingeniería. Cuando se habla de minimizar tiempo, costes, materiales o espacio, se hace referencia al objetivo principal de todo proyecto: optimizar. Este aspecto tan importante hoy en día es objeto de investigación y desarrollo en el campo científico, con lo cual se busca mejorar aspectos tan importantes en la industria como lo son la

productividad y el costo, y el tiempo de procesamiento y la reducción de espacio en el ámbito computacional.

El proceso de improvisación que lleva a cabo el algoritmo HS para optimizar una función hace de este una poderosa herramienta de programación para la resolución de múltiples problemas de optimización.

Entre las ventajas de los algoritmos armónicos frente a otros algoritmos evolutivos destacan las siguientes: no requiere cálculos complejos, obvia óptimos locales y puede manejar variables discretas y continuas.

Con respecto a los algoritmos genéticos (el otro algoritmo metaheurístico desarrollado en este TFM y el más desarrollado en la literatura), destaca que, al no realizar cálculos matemáticos complejos, el procesamiento es mucho más rápido, lo cual hace que el tiempo de convergencia del algoritmo marque la diferencia entre los diferentes algoritmos metaheurísticos desarrollados.

Dada la versatilidad que ofrece este algoritmo, ha permitido que se hayan utilizado para una gran variedad de problemas tales como el diseño de una red de distribución de agua, el diseño estructural, problemas relacionados con el transporte (rutas de vehículos) o la solución al juego del sudoku, entre otros.

Es decir, en este método el proceso de improvisación de una banda de música se usa como la base para la resolución de problemas de optimización, siendo el objetivo encontrar el perfecto estado de armonía.

La función objetivo representaría al público, las variables que se pretenden estimar a cada músico, el rango musical de cada instrumento es análogo al rango de valores de dichas variables y la armonía musical en un tiempo específico es equivalente a una solución de una iteración determinada. El conjunto de soluciones se almacena en la memoria armónica (HM) con un tamaño previamente configurado, lo que es equivalente a la población de los algoritmos genéticos.

Desde un punto de vista algorítmico, la Búsqueda Armónica es un proceso iterativo y estocástico que combina de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda, construyendo en cada iteración,

un vector solución siguiendo tres pautas básicas: búsqueda en la memoria armónica, ajuste por improvisación y, finalmente, búsqueda aleatoria.

Una vez hecha una breve descripción del algoritmo, se procede a representar y explicar los pasos del mismo.

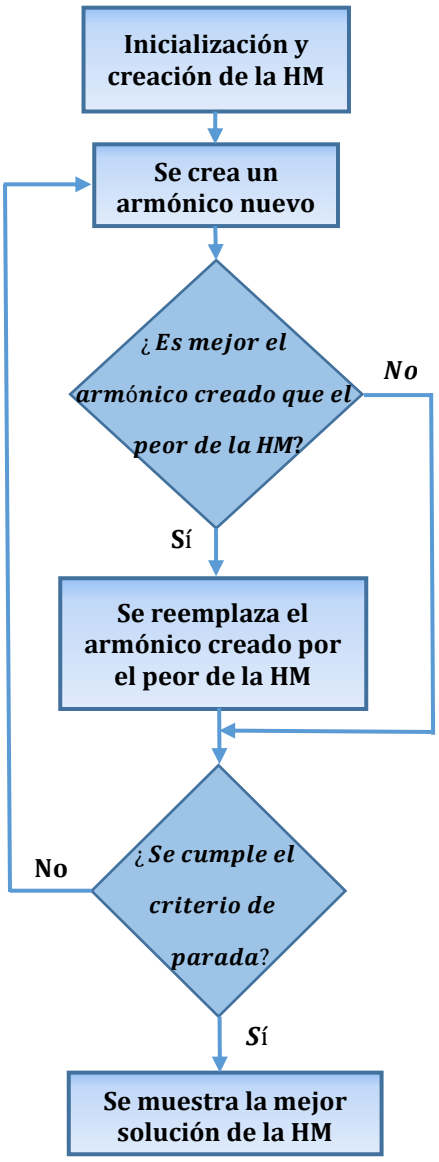


Figura 6: Diagrama de flujo del funcionamiento del algoritmo de búsqueda armónica.

- **Inicialización de los parámetros del HS**

Consiste en la configuración de los parámetros del algoritmo, entre los que destacan:

  - HMS: tamaño de la memoria armónica.

- HMCR: tasa de consideración de la memoria armónica.
- PAR: tasa de ajuste de tono.
- BW: ancho de banda de ajuste al tono.
- NI: número de improvisaciones.

- **Inicialización de la memoria armónica**

Una vez que los parámetros de inicialización son configurados, la primera generación de armónicos (posibles soluciones del problema) son generadas desde una distribución uniforme en los rangos [LBi, UBi] (siendo LBi y UBi, el límite de decisión más bajo y más alto de cada variable), y almacenadas en la memoria armónica (HM) de acuerdo al tamaño previamente fijado (HMS).

Es decir, la iniciación aleatoria de la memoria armónica se hará siguiendo la siguiente ecuación.

$$\boldsymbol{x_i^j} = LB_i + r * [UB_i - LB_i\,] \tag{2.1}$$

Siendo *j* el número de solución del HM (1…HMS), *i* el número de la variable a estimar (1...3, en nuestro caso) y r un número aleatorio de distribución uniforme (U~(0,1)).

- **Improvisación de una nueva armonía**

Se genera una nueva armonía (también llamada improvisación) basada en varios parámetros como el *Harmony Memory Considering Rate* (HMCR), que tiene valores entre 0 y 1, que controla la influencia de los armónicos en HM para generar el nuevo (si HMCR es 0, el nuevo armónico es completamente aleatorio), el *Pitching Adjust Rate* (PAR) que determina la probabilidad de que un armónico del HM sea mutado y el ancho de banda (*Bandwidth*, BW), que es el cambio límite de ajuste.

- **Actualización de la memoria armónica**

La nueva armonía reemplazará la peor armonía almacenada en la memoria armónica si el valor de aptitud del vector armónico actual, medido en términos de la función objetivo es mejor que el de la peor armonía.

- **Verificación del criterio de parada**

Si el número máximo de improvisaciones (NI) es alcanzado, el algoritmo proporciona la mejor solución del HM. Si no, se genera una nueva armonía de nuevo de acuerdo a los parámetros previamente descritos y el proceso de evaluación (improvisación de un nuevo armónico y actualización de la memoria) se repite.

Por lo general, el algoritmo HS es poco sensible a los parámetros, por esto, el algoritmo no requiere de una afinación exhaustiva de los parámetros para obtener buenas soluciones [Cobos, 2011]. A pesar de lo anterior, es preciso destacar que los parámetros HMCR y PAR ayudan al método en la búsqueda de mejores soluciones. Los parámetros PAR y BW tienen un profundo efecto en el desempeño del algoritmo y es por esto que el ajuste de estos dos parámetros ha sido muy importante para la mejora del algoritmo con el paso de los años.

El HS ha sido aplicado en diversos problemas reales y sobre el cual se han realizado gran variedad de transformaciones, obteniendo resultados satisfactorios [Zong, 2009][Yang, 2009], lo que motivó a continuar la investigación sobre este algoritmo.

Entre las transformaciones o hibridaciones que ha sufrido HS destaca principalmente la búsqueda armónica mejorada (IHS). Este algoritmo, propuesto en 2007 por Mahdavi et al [Mahdavi, 2007], emplea un novedoso método para la generación de nuevos vectores solución basado en el ajuste dinámico de los parámetros PAR y BW, logrando con esto mejorar la precisión y la velocidad de convergencia.

Esta variante de la búsqueda armónica modifica el paso de generación de una nueva armonía del algoritmo original (en la propuesta original PAR y BW parámetros fijos). PAR y BW cambian dinámicamente con el número de iteraciones y se calculan con las fórmulas mostradas a continuación.

$$PAR = PAR_{min} + \frac{(PAR_{max} - PAR_{min})}{NI - 1} * (iter - 1) \tag{2.2}$$

Siendo el PAR la tasa de ajuste del tono para cada iteración, PAR_{min} la tasa mínima de ajuste del tono, PAR_{max} la tasa máxima de ajuste del tono, NI el número máximo de improvisaciones e $iter$ el número de iteración actual.

$$bw = bw_{max} * e^{(c*(iter-1))} \quad (2.3)$$

Siendo bw el ancho de banda de tasa de ajuste del tono para cada iteración, bw_{min} el ancho de banda mínimo de tasa de ajuste del tono y bw_{max} el ancho de banda mínimo de tasa de ajuste del tono. Por último, el coeficiente c sigue la siguiente expresión:

$$c = \frac{\ln(bw_{min}/bw_{max})}{NI - 1} \quad (2.4)$$

Con estas modificaciones el parámetro PAR crece linealmente con el número de improvisaciones, mientras que BW decrece exponencialmente, consiguiendo que con el paso de las iteraciones se dé menor peso a la aleatoriedad haciendo un ajuste más preciso en torno a la solución generada, ya que al generar una variable de la solución si un número aleatorio es menor o igual que el PAR se realiza el ajuste, siendo dicho ajuste el valor generado de esa solución más ó menos el BW (en función si se ajusta por arriba o por abajo, misma probabilidad para ambos casos). Con este cambio en los parámetros, el algoritmo de búsqueda armónica mejorado (IHS) logra mejorar el rendimiento del algoritmo convencional (HS), ya que encuentra mejores soluciones tanto a nivel local como global en un tiempo de convergencia menor. De ahí que en el presente TFM se haya implementado directamente el algoritmo de búsqueda armónica mejorado.

c. Particularización de los algoritmos metaheurísticos para la estimación de la posición 2D y la orientación de la baliza US

Una vez expuesta una pequeña introducción sobre los algoritmos metaheurísticos que se van a utilizar, se particularizan para nuestro caso en concreto.

El requerimiento principal para la calibración de una baliza usando el método propuesto en el presente TFM es la disposición del mapa (en formato vectorial) del entorno donde las balizas van a ser instaladas. Para testear el algoritmo implementado se va a utilizar el mapa de la Escuela Politécnica Superior de la Universidad de Alcalá en formato XML desarrollado en [García, 2011].

A continuación, en la Figura 7 se muestra una parte de dicho mapa en el cual se han llevado a cabo las pruebas experimentales realizadas y que corresponde a la tercera planta de la Escuela.

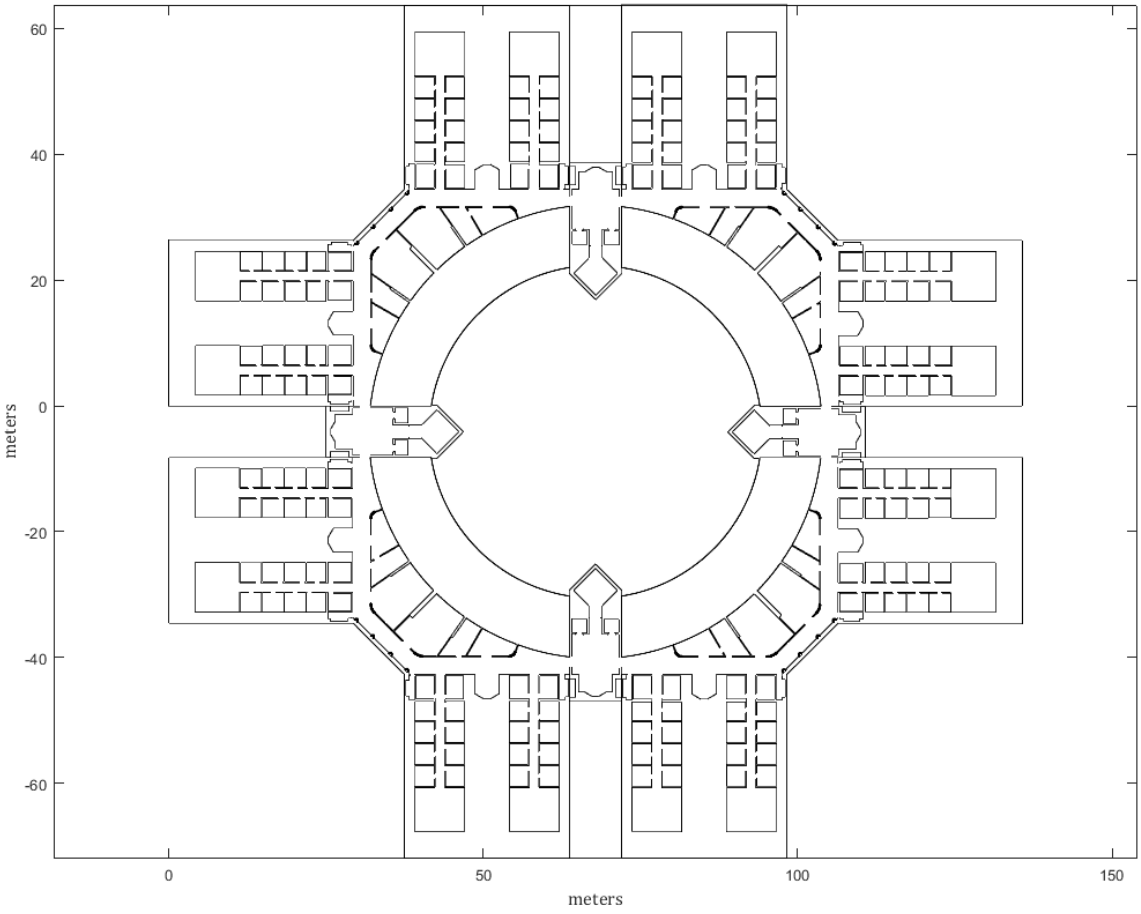


Figura 7: Plano en formato XML de la tercera planta de la Escuela Politécnica Superior de la Universidad de Alcalá.

Los algoritmos metaheurísticos se utilizan para la estimación de la posición del transductor central de la baliza, que consta de 5 transductores ultrasónicos, 4 ubicados en los vértices de una estructura cuadrada de 70.7 cm de lado y el otro situado en el centro de la estructura. Para ello y según el método propuesto en este TFM, se necesita la siguiente información de entrada: distancias desde la baliza a las paredes, información heurística (región aproximada en la que pueda estar la baliza, orientación aproximada de las medidas (de la plomada láser)) y del mapa (información vectorial de las paredes de las cuales se tienen distancias). En la Figura 8 se representa un diagrama 3D de la propuesta planteada en este trabajo.

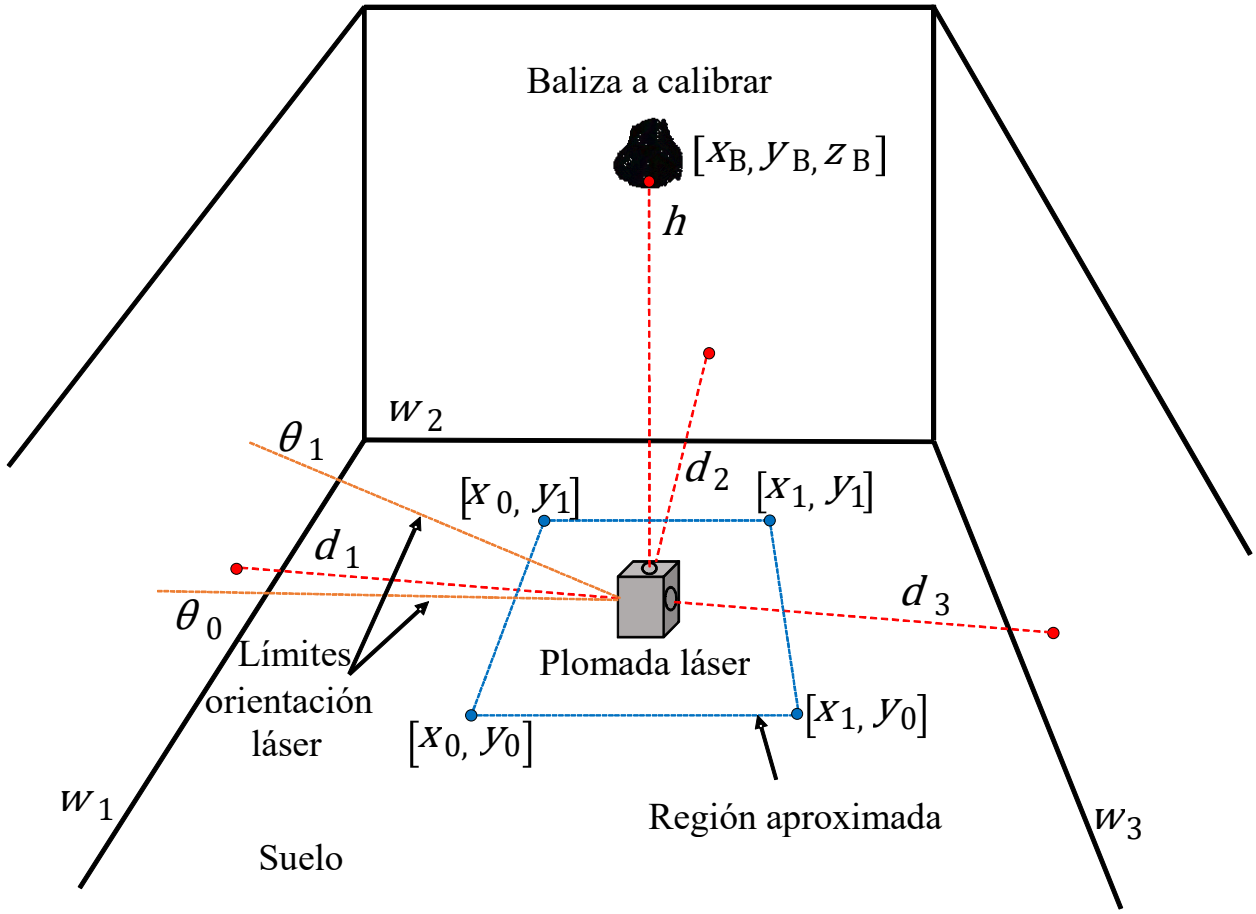


Figura 8: Esquema general del sistema planteado.

Como se puede observar en la ilustración, la plomada láser se sitúa en el suelo en la proyección vertical del transductor central de la baliza en el mismo. Las proyecciones horizontales de la plomada a cada pared se miden mediante el medidor láser, obteniendo las distancias a cada pared. En el esquema mostrado hay tres proyecciones horizontales (d_1, d_2 y d_3) basándonos en las proyecciones que proporciona la plomada láser utilizada en este TFM (para más información acerca de la plomada utilizada véase el apartado de descripción de material utilizado), pero el número de distancias de la estructura láser a las paredes puede ser variable como muestra la siguiente expresión:

$$\boldsymbol{D} = \begin{bmatrix} d_1 \\ \vdots \\ d_n \\ \vdots \\ d_N \end{bmatrix} \tag{2.5}$$

Siendo N el número de distancias proporcionadas por el usuario, valor que debe coincidir con el número de paredes de las cuales se tiene información, ya que cada distancia va asociada a su pared correspondiente (d_n-> w_n).

La información heurística relacionada con la región aproximada de la posición de la baliza, la orientación aproximada de la plomada láser y la información de las paredes en la cuales impactan las proyecciones de la plomada se detallan en los siguientes párrafos.

La región rectangular aproximada en la cual se ubica la baliza viene definida como:

$$\boldsymbol{R} = [x_0 \;\; x_1 \;\; y_0 \;\; y_1]^T \tag{2.6}$$

La orientación de la plomada respecto a la referencia del mapa (semieje positivo del mapa en formato XML) será otra de las variables que se obtendrán de nuestro algoritmo, proporcionando a la entrada un rango aproximado de la misma, como representa el vector *Angle Bounds*, **A**:

$$\boldsymbol{A} = [\theta_0 \;\; \theta_1]^T \tag{2.7}$$

Por último, el vector **W** representa la información de las paredes en formato vectorial de las cuales se tiene información (distancias medidas por el medidor láser de la plomada a ellas).

$$\boldsymbol{W} = \begin{Bmatrix} w_1 \\ \vdots \\ w_n \\ \vdots \\ w_N \end{Bmatrix} \tag{2.8}$$

Todos estos datos de entrada se introducen tanto al algoritmo genético, como al de búsqueda armónica que son capaces de obtener la estimación de la posición del transductor de la baliza desde el cual se han dado las distancias a las paredes (posición en *x*, *y* de la baliza y orientación de las medidas), representada por el vector $\widehat{\mathbf{B}}$, de acuerdo a la minimización de la función Aptitud ***f*** (*fitness function*), que es la encargada de evaluar las posibles soluciones del algoritmo metaheurístico en cuestión.

$$\widehat{\mathbf{B}} = [\hat{x}_B \ \hat{y}_B \ \hat{\theta}_B]^T \tag{2.9}$$

$$\boldsymbol{f} = \frac{1}{N} * \sum_{i=1}^{i=N} |d_i - \hat{d}_i| \tag{2.10}$$

Es decir, el algoritmo va recorriendo las posiciones de la rejilla (región aproximada) quedándose con la posición que menor error en la función aptitud muestre, con el mínimo absoluto. El mínimo absoluto se refiere a de todas las posibles combinaciones de valores de estudio de la rejilla, va calculando en cada posible combinación las distancias a las paredes ($\hat{d}_i$) desde la posición estimada ($\hat{x}_B, \hat{y}_B$) a las paredes de estudio con la orientación estimada de las medidas ($\hat{\theta}_B$), quedándose con la combinación que menor error da respecto a la observación (valor introducido por el usuario, el obtenido con el medidor laser, d_i).

Como resumen, se representa en la Figura 9 el diagrama de bloques del sistema propuesto en el presente TFM para la calibración de balizas US.

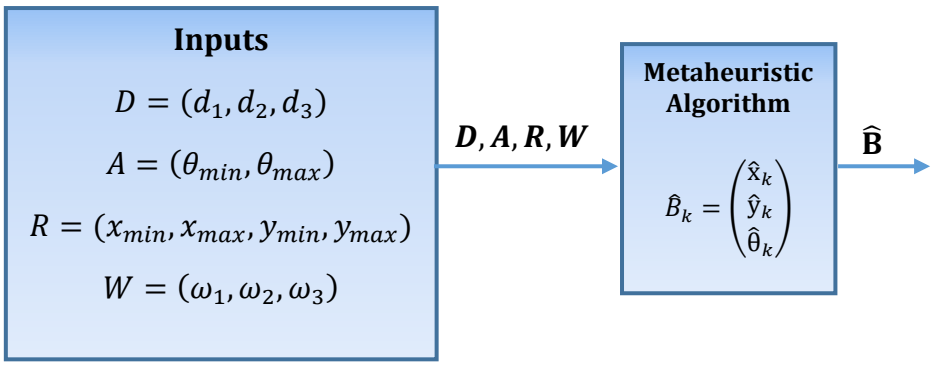


Figura 9: Diagrama de bloques del sistema propuesto.

Siendo k el número de iteración del ciclo evolutivo del algoritmo metaheurístico, es decir, $\widehat{\mathbf{B}}_{\boldsymbol{k}}$ es una posible combinación de nuestra solución, siendo la salida del algoritmo ($\widehat{\mathbf{B}}$) la mejor solución de todas las estudiadas en el número de ciclos determinado.

Por último, resaltar que este procedimiento también se podría realizar evaluando todas las posibles combinaciones de las soluciones, obteniendo el mínimo absoluto con una probabilidad del 100%, pero el número de operaciones a realizar sería muy elevado de cara a que este proceso se pretende ejecutar en tiempo real. De ahí la

importancia del uso de algoritmos metaheurísticos, que consigue trabajar con una carga computacional mucho menor obteniendo una probabilidad cercana al 100% de encontrar el mínimo absoluto, ya que este tipo de algoritmos evalúa únicamente las combinaciones con mayor probabilidad de contener al mínimo (de ahí que tenga menor número de combinaciones, se ahorra aquellas zonas con baja probabilidad, ganando rapidez de cómputo), mientras que el método basado en fuerza bruta, no tiene forma de restringir donde puede estar el mínimo y evalúa todas las posibles combinaciones de la solución.

Por lo general, los algoritmos metaheurísticos utilizan en torno al 0.01% de las combinaciones totales. En este trabajo se propone utilizar en torno al 1% del total, es decir se evaluará un mayor número de combinaciones que las necesarias, siendo más conservadores, para asegurarnos que la solución sea lo más fiable posible (no sea un falso mínimo). Además, se usará un número de combinaciones totales dinámico que dependerá de la bondad de los valores introducidos (de *x* e *y*, región aproximada, y de la orientación, *Angle bounds*), para que indistintamente del valor de los mismos la solución obtenida sea correcta. Es decir, cuanto más elevados sean los rangos introducidos por el usuario, mayor número de combinaciones realizará el algoritmo, no dando pie a que, si los datos introducidos por el usuario no son muy concretos, al algoritmo no le dé tiempo a converger en la solución óptima.

Con estas especificaciones propuestas, el algoritmo tardará en ejecutarse del orden de un minuto en un caso de resolución compleja, tiempo por otro lado totalmente asumible para ejecutar en tiempo real, ya que la calibración de la baliza es un procedimiento que sólo se realiza una vez.

3. Descripción del material utilizado

En esta sección se describen las principales características del material utilizado para la realización del TFM, recogidas en la Tabla 1 expuesta a continuación:

Equipo	Descripción
Plomada Láser	Leica Lino P5, proyecta cinco rayos láser, los cuales emiten en ángulos rectos entre sí (véase la figura mostrada a continuación). Alcance de 30m con una precisión de ±1.5mm. 
Medidor Láser	Suaoki D5, medidor láser de distancias de hasta 20m, con una tolerancia en las medidas de ±2mm. 

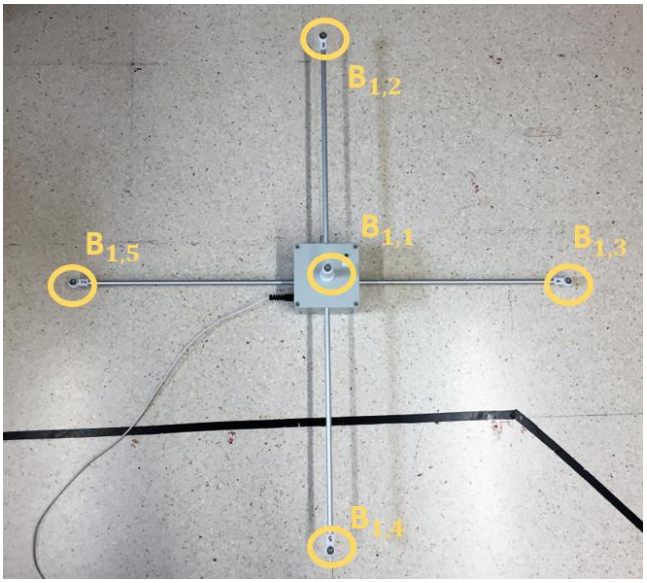
Tablet	Samsung Galaxy Tab S3, procesador de 64 bits, 4 núcleos a 2.15GHz y 4 núcleos a 1.6GHz, 4GB de RAM y 32GB de ROM. Sistema operativo Android de versión 7.0 Nougat.
Matlab	Herramienta de software matemático que ofrece un entorno de desarrollo con un lenguaje de programación propio (lenguaje M).
Android Studio	Entorno de desarrollo aplicaciones sobre plataforma Android.
Ordenador	Intel Core i7 2.93Ghz, 3GB de RAM y procesador de 64 bits.
U-LPS	Sistema de posicionamiento local ultrasónico constituido por cinco transductores ultrasónicos Prowave 328St160 [Ureña, 2018], distribuidos en una estructura cuadrada de 70.7cm de lado y con una distancia entre el transductor central y el resto de 0.5m, capaces de cubrir una superficie aproximada de 30m^2 instalados en el techo a una altura de 3,5m. La FPGA proporciona para todos los U-LPS un enlace inalámbrico a un PC externo, para configurar fácilmente la frecuencia de muestreo, las secuencias de codificación y el esquema de modulación. Estos parámetros pueden modificarse en tiempo de ejecución, aunque en la práctica cada U-LPS sólo se configura una vez en una fase de configuración. 

Tabla 1: Equipos y software utilizados en el TFM.

4. Desarrollo de la aplicación

a. Introducción

En este TFM se ha realizado una aplicación sobre plataforma Android, que permitirá la estimación automática de la posición de los transductores de una baliza de posicionamiento local.

En el desarrollo de la aplicación se han diferenciado las siguientes partes que se han tenido que ir implementando para el correcto funcionamiento de la misma, que se expondrán en las secciones posteriores del capítulo, siendo las que se exponen a continuación:

- Programación para el manejo de datos del mapa en formato vectorial.
- Programación de la conversión de información en formato latitud/longitud a formato vectorial.
- Programación para la fusión de la información de los mapas en formato vectorial del edificio con los de Google Maps.
- Programación para la visualización sobre Google Maps.
- Programación del algoritmo de búsqueda de la pared más cercana.
- Programación de los algoritmos metaheurísticos.
- Programación del algoritmo de obtención de la posición 3D de los transductores para la baliza colocada en la pared

Antes de empezar a explicar las secciones expuestas con más detalle, se va a hacer una breve introducción a Android y su lenguaje de programación, ya que será útil para entender posteriormente por qué se ha programado así la aplicación.

Lo primero que se hace a la hora de crear una aplicación Android es elegir la mínima versión (*minimun SDK*) en la cual se puede ejecutar la app y cuál es la versión objetivo (*target SDK version*), para la que está prevista que se use la aplicación. Lo normal, es utilizar la mínima versión posible, para que así la app pueda ser accesible al mayor número de usuarios (se ha utilizado la API 15-Android 4.0.3, IceCreamSandwich-, que a día de escritura del presente TFM es compatible con prácticamente el 100% de dispositivos). De versión objetivo se ha escogido la API

27 (Android 8.1, Oreo-*Go Edition*), que es la última versión disponible (es recomendable que sea lo mayor posible, para que depure fallos posibles en versiones anteriores).

Cada proyecto Android contiene un descriptor de la aplicación o *manifest*, que es donde se programan los permisos de la aplicación (para acceder y poder cambiar el estado del wifi del dispositivo), el icono de la aplicación, y las diferentes actividades del proyecto (entre ellas se configura cuál es la actividad de arranque al iniciar la app (categoría *Launcher*), el resto se catalogan dentro de la categoría *Default*); ficheros para construir el módulo/proyecto, *Gradle Scripts*, donde se configura la versión objetivo para compilar el programa, la versión mínima y donde se añaden las librerías externas a Android utilizadas. En este proyecto se ha utilizado las siguientes librerías externas: *Jcoord 1.0* que permite convertir entre coordenadas latitud-longitud (WG S84) y coordenadas UTM, y viceversa, necesaria a la hora de obtener/ representar información en Google Maps; y JGAP que contiene una serie de métodos que permiten la implementación de algoritmos genéticos en Java. Por último, y lo más importante de un proyecto Android son las actividades (*activities*), que están conformadas por dos partes: la parte lógica y la parte gráfica. La parte lógica es el código fuente, un archivo .java que es una clase, que al ser una actividad extiende de la clase *Activity*, y en la cual se programa todo lo relacionado con la funcionalidad para la cual se realiza la app (algoritmos, funcionalidad de los botones, etc.). La parte gráfica es un XML que tiene todos los elementos que se ven por pantalla como iconos, botones, texto, etc.; es decir se encarga del diseño de las diferentes pantallas de la aplicación (*layouts*).

Cabe destacar el ciclo de vida de una actividad ya que las actividades son las que realmente controlan el ciclo de vida de las aplicaciones. Una actividad puede tener 4 estados:

- Activa (*running*), actividad visible, tiene el foco.
- Visible (*paused*), actividad visible pero no tiene el foco, por ejemplo, cuando se pasa a otra actividad que no ocupa toda la pantalla.
- Parada (*stopped*), actividad no visible, tapado por completo.

- Destruida (*destroyed*), cuando la actividad termina o es finalizada por el sistema.

En resumen, el ciclo de vida de una actividad se ilustra en la Figura 10:

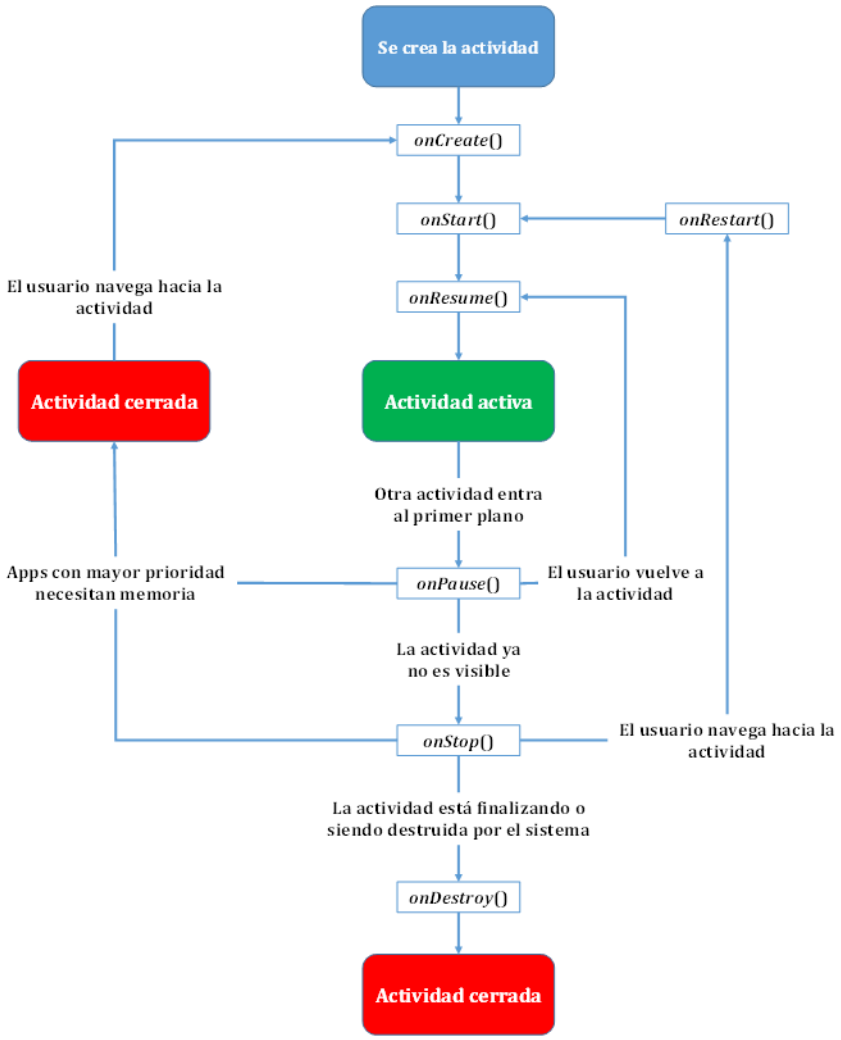


Figura 10: Ciclo de vida de una actividad Android.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad y en los cuales se permitirá programar el código. Los diferentes métodos que capturan estos eventos son:

- *onCreate*(): se utiliza para programar las inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos.
- *onStart*(): indica que la actividad está a punto de ser mostrada.
- *onResume*(): cuando va a comenzar a interactuar la actividad con el usuario. Sirve para lanzar animaciones, música, etc.

- *onPause*(): antes de lanzar la actividad a segundo plano. Se suele utilizar para detener animaciones, música y almacenar datos.
- *onStop*(): la actividad deja de ser visible para el usuario, si hay poca memoria, puede que se destruya la actividad sin pasar por este método.
- *onRestart*(): actividad vuelve a ser representada tras *onStop*.
- *onDestroy*(): se llama antes de que la actividad sea destruida totalmente. Al igual que el método *onStop*, si hay muy poca memoria puede que ni se ejecute.

Para poder manejar variables en varias clases sin tener que usar constructores (función dentro de una clase con su mismo nombre, que permite utilizar las variables que se introducen como entradas a la clase), se ha creado una especie de nube, una clase (llamada *Link_data*) en la cual almacenar las variables, para que las variables declaradas ahí puedan ser accesibles desde todas las clases.

Esta aplicación desarrollada implementa una serie de algoritmos, que conllevan un procesado interno que el usuario no observa. En Android, hay un problema a la hora de realizar tareas que se extienden en el tiempo, ya que a partir de los 5s de duración de un proceso en la tarea principal muestra automáticamente un diálogo AMR, de que la app no responde. En la tarea principal se ejecutan todas las operaciones que gestionan la interfaz de usuario de la aplicación, además de las actividades, servicios y *broadcast receivers*. Es por ello que cualquier operación costosa/larga que se realice en el hilo principal, va a bloquear la ejecución del resto de componentes de la aplicación, y por supuesto también la interfaz, produciendo al usuario un efecto evidente de lentitud, bloqueo o mal funcionamiento en general, algo que se debería evitar a toda costa, ya que podría suponer la desinstalación de la aplicación por parte del usuario. La solución para evitar este suceso es utilizar tareas asíncronas (hilos), consiguiendo que las tareas pesadas se ejecuten en un hilo aparte, evitando el bloqueo de la interfaz.

Hay dos formas de trabajar con hilos, mediante *Thread* o mediante *AsyncTask*. Para esta app se han utilizado los dos tipos, debido a las prestaciones que nos ofrecen en las diferentes casuísticas.

La clase *Thread* es una clase extendida a hilo que en su interior contiene una función *run*, en la cual se introduce todo el código a ejecutar cuando el hilo una vez creado,

se arranque (*hilo.start*()). Al utilizar este tipo de hilos, hay más opciones a la hora de controlar el hilo (mediante comandos como: *hilo.join*(), *hilo.Run*(), *hilo.getPriority*(), etc.), de ahí que se utilice sobre todo en operaciones secuenciales ya que permite controlar la prioridad de ejecución. Este hilo no es capaz de interaccionar con el interfaz de usuario (principal), por lo que para visualizar algo por pantalla desde un hilo se deberá utilizar el manejador del hilo (*Handler*). En nuestro caso se ha utilizado esta clase para realizar la lectura de la información del mapa en formato vectorial, hilo que no necesita interacción alguna con el hilo interfaz de usuario, ya que se encarga en segundo plano de procesar toda la información del archivo '.xml' para a posteriori poder realizar operaciones con la información del mismo.

La otra forma de trabajar con hilos es mediante *AsyncTask*, que es la solución que recomienda Android Studio. La clase *AsyncTask* es una tarea asíncrona definida en las siguientes funciones: *onPreExexute*(), *doInBackground*(Params…), *onCancelled*(), *onProgressUpdate*(Progress…) y *onPostExecute*(Result…). Este tipo de hilo no necesita el *Handler*, ya que puede interactuar con el interfaz de usuario tanto en el método *onProgressUpdate* (durante la ejecución) como en el *onPostExecute* (tras la ejecución). El método *doInBackground* sería el equivalente al *run* de la clase *Thread*, dónde se introduce todo el código a ejecutar en el hilo. Esta clase se ha utilizado en la ejecución del algoritmo de calibración de las balizas (que realiza la estimación del transductor central con el método de minimización seleccionado y a posteriori obtiene a partir del mismo la posición del resto de transductores), que es un proceso que en función de los datos de entrada puede durar hasta el orden de minutos, para poder proyectar en pantalla un *progress dialog* que permitiera mostrar al usuario en tiempo real el porcentaje de cómputo procesado por el algoritmo. Al ser un hilo que se ejecuta en segundo plano esto permitiría al usuario en el caso de una tarea de larga duración, que pudiera utilizar otras funcionalidades/aplicaciones del dispositivo, mientras espera la finalización de la tarea.

Por último, para acabar con la introducción, decir que la aplicación desarrollada está fundamentada en la utilización de Google Maps, más en concreto se ha usado una tecnología desarrollada por Google unos años atrás, llamada *Indoor Maps*. Dicha tecnología, que está incluida en la API de Google Maps para Android e Ios, ha implementado mapas interiores de algunos edificios (como universidades,

hospitales o aeropuertos) y permite al usuario obtener automáticamente información del interior de los mismos al hacer *zoom in* sobre Google Maps en el edificio en cuestión, en nuestro caso la Escuela Politécnica Superior de la Universidad de Alcalá (véase la figura 11). Se utilizará el mapa interior del edificio tanto para que el usuario pueda introducir en el mismo la información necesaria para poder ejecutar el algoritmo metaheurístico (región aproximada y paredes en las que ha proyectado la plomada láser) como para representar la posición de los transductores de la baliza, una vez el algoritmo haya finalizado.

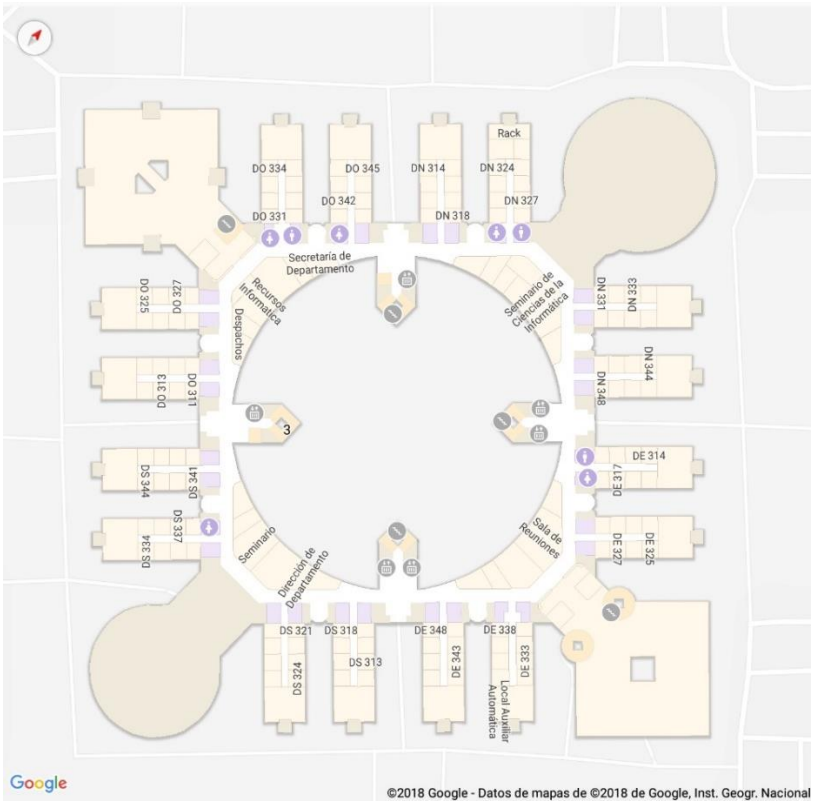


Figura 11: Plano en Google Maps de la tercera planta de la Escuela Politécnica Superior de la Universidad de Alcalá.

Destacar que para poder hacer uso de estas funcionalidades en una aplicación (SDK de Maps para Android) hay que tener habilitados, en la opción del SDK *Manager* de Android Studio, los *Google Play Services* (se recomienda tener la versión más actualizada) y solicitar permiso a Google. Para ello se debe obtener una clave API que luego se puede agregar a la aplicación móvil. La clave API (API *key*) se usa para rastrear las solicitudes de API asociadas con su proyecto para su uso y facturación (en el caso de que la aplicación haga más de un número determinado de solicitudes por segundo habría que pagar para poder hacer uso de la API).

Para administrar proyectos, obtener la clave API para los mismos y agregar restricciones, hay que utilizar la consola de *Google Cloud Platform*. La restricción de clave es una forma de garantizar la seguridad del proyecto, para evitar plagios. Es decir, se le puede agregar al paquete de proyecto la huella digital de un ordenador, y en el caso de que se intente ejecutar el proyecto desde otro ordenador, no dará a esa aplicación descargada las funcionalidades de la API (al hacer la restricción sobre la SDK de Maps no permitiría la visualización de los mapas). En nuestro caso, como es un proyecto de investigación el cuál puede ser utilizado por varios programadores, no se ha hecho uso de esta opción.

Una vez conseguida la API *key*, para poder finalizar el procedimiento para utilizar los mapas de Google en la aplicación, habría que proceder a agregar la clave a la app, para ello en el *manifest* se añadiría dentro del elemento <*application*> el siguiente elemento secundario (sustituyendo la clave API obtenida en el atributo *value*):

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY"/>
```

Figura 12: Declaración API key en el manifest.

b. Programación para el manejo de datos en formato vectorial

El requerimiento principal para la calibración de una baliza usando el método propuesto en el presente TFM es la disposición de la información del mapa del entorno de trabajo. Como bien se ha ido explicando en las diferentes secciones, el trabajo se fundamenta en el uso de los mapas de Google debido a la gran diversidad de funciones que estos permiten a la hora de su utilización, además de su rápida ejecución en tiempo real. A pesar de ello, estos mapas tienen diferentes limitaciones, entre las que destaca el no poder hacer uso de la información contenida en los mismos, dejando únicamente trabajar sobre ellos. Para testear el algoritmo implementado se va a utilizar la información del mapa de la Escuela Politécnica Superior de la Universidad de Alcalá en formato XML que se ha desarrollado anteriormente en el grupo de investigación [García, 2011].

El lenguaje XML, acrónimo anglosajón de la palabra *eXtensible Markup Language* (lenguaje de etiquetado extensible), consiste en jerarquizar y estructurar la información definiendo etiquetas en función al tipo de dato que se está describiendo. Este lenguaje contiene una serie de reglas, pautas o convenciones para planificar formatos de texto para tales datos, de manera que produzcan archivos que sean fácilmente generados y leídos por un ordenador, inequívocos y que eviten los problemas más comunes como la falta de extensibilidad, la falta de interoperabilidad entre plataformas o la falta de soporte para universalizar su tratamiento. La principal ventaja de este tipo de archivos es que, al tener todo clasificado, a la hora de buscar un dato concreto el proceso es mucho más rápido, ya que eres capaz de discriminar muchos datos, ganando en tiempo de ejecución.

El etiquetado del archivo de estudio sigue el siguiente árbol jerárquico:

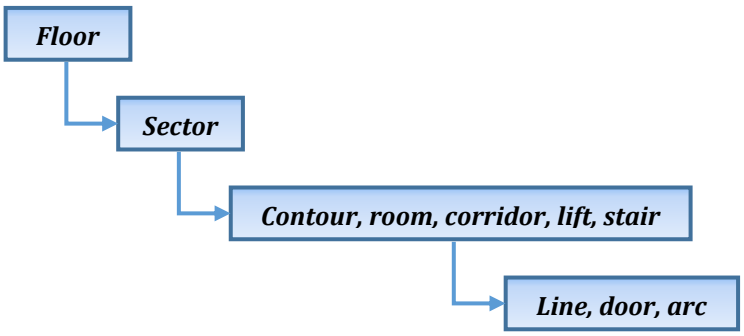


Figura 13: Estructura jerárquica del archivo XML.

En un primer nivel se distingue la planta del edificio (*floor*), seguido por la zona de la planta descrita (EPS está divida en 4 sectores, correspondientes a los puntos cardinales, además de las zonas entre sectores). En un tercer nivel se contemplan las siguientes estancias en el edicio: el contorno del mismo (*contour*), las salas (*room,* que a su vez se etiquetan en diferentes tipos: baños, oficinas/despachos o laboratorios entre otros), pasillos (*corridor*), ascensores(*lift*) y escaleras (*stair*). En el último nivel se distinguen paredes lisas (*line*, descritas con 4 atributos: punto (x,y) del punto de origen como del punto final), curvas (*arc*, descritos con 8 atributos: punto de origen, centro y fin, en coordenadas x e y, radio y rotación) y puertas (*door*, descritas con 5 atributos: punto (x,y) del punto de origen como del punto final y tipo).

A continuación, se expone un extracto del archivo XML del mapa, para ilustrar como es la información con la que se va a tratar.

```
<?xml version="1.0" encoding="utf-8"?>
      <floor number="3">
          <sector name="OESTE">
              <contour>
                  <line x_o="0" y_o="0" x_d="29,34" y_d="0" />
                  <door x_o="29,34" y_o="0" x_d="32,09" y_d="0"
                  type="0"/>
                  <line x_or="32,09" y_o="0" x_d="41,76" y_d="0"/>
                  <arc x_or="41,76" y_o="0" x_d="63,75" y_d="21,99"
                  r_x="67,9" r_y="-4,15" radio="26,47" turn="1"/>
                  <line x_o="63,75" y_o="21,99" x_d="63,75"
                  y_d="31,70"/>
                  <door x_o="63,75" y_o="31,70" x_d="63,75"
                  y_d="34,49" type="0"/>
                  <line x_o="63,75" y_o="34,49" x_d="63,75"
                  y_d="63,74"/>
                  <line x_o="63,75" y_o="63,74" x_d="37,47"
                  y_d="63,74"/>
                  <line x_o="37,47" y_o="63,74" x_d="37,47"
                  y_d="34,43"/>
                  <line x_o="37,47" y_o="34,43" x_d="29,31"
                  y_d="26,28"/>
                  <line x_o="29,31" y_o="26,28" x_d="0" y_d="26,28"/>
                  <line x_o="0" y_o="26,28" x_d="0" y_d="0"/>
              </contour>
              <room name="F31 OESTE" type="laboratory">
                  <line x_o="4,29" y_o="1,69" x_d="11,32"
                  y_d="1,69" />
                  <line x_o="11,32" y_o="1,69" x_d="11,32"
                  y_d="4,89" />
                  <door x_o="11,32" y_o="4,89" x_d="11,32" y_d="6,39"
                  type="0" />
                  <line x_o="11,32" y_o="6,39" x_d="11,32"
                  y_d="9,59" />
                  <line x_o="11,32" y_o="9,59" x_d="4,29"
                  y_d="9,59" />
                  <line x_o="4,29" y_o="9,59" x_d="4,29"
                  y_d="1,69" />
              </room>

              .
              .
              .

          </sector>
      </floor>
```

Una vez descrito el formato del archivo con el que se va a trabajar, destacar que para poder trabajar con los valores del mismo hay que realizar un *parser* (análisis sintáctico) del archivo.

El *parser* es la herramienta principal de cualquier aplicación XML, ya que este proceso no únicamente permite comprobar si nuestros documentos están bien formados o son válidos, sino que también permite incorporarlos a nuestras aplicaciones, de manera que estas puedan manipular y trabajar con los datos de los documentos XML.

Principalmente se puede hacer un *parser* de un documento XML de dos modos: usando SAX o DOM. Los grandes rasgos en el funcionamiento de cada modo son los siguientes:

- DOM, *Document Object Model*: crea un árbol jerárquico en memoria que contiene todo el documento XML (independientemente de lo que necesite el usuario), y con él en memoria espera a que el usuario le solicite algo del mismo.
- SAX, *Simple API for XML*: se usa para realizar un recorrido secuencial de los elementos del documento XML, es decir, va tratando la información a la vez que la va leyendo.

En nuestra aplicación en concreto, el XML se va a utilizar principalmente para obtener los atributos de las paredes y a partir de los mismos y del punto seleccionado por el usuario, detectar la pared más cercana a dicho punto. Es decir, habrá que realizar un barrido de todas las paredes de la zona de interés hasta encontrar la pared más cercana. Cómo es un proceso que mínimo para cada ejecución del algoritmo se tiene que realizar tres veces (número de paredes que hay que introducir según el método propuesto), se ha decidido no utilizar el *parser* SAX, ya que, con este tipo, habría que estar continuamente leyendo el XML con el incremento en el tiempo de ejecución que conlleva.

Por tanto, para que afecte lo menos posible en el rendimiento de la aplicación, se ha propuesto un parseo del XML mediante el modo DOM utilizando el paquete Java org.w3c.dom, para leer únicamente una vez el XML (que se hará en un hilo en la pantalla de arranque de la aplicación, tardando en torno a 35ms) a pesar de un gasto en memoria (ya que con este tipo de parseo se almacenará en una variable (de la clase *Element* de dicho paquete, que se ha definido como rootMapa en la clase *Link_data*, para que pueda ser accesible desde cualquier parte del programa) todos

los atributos del mapa, pudiendo acceder a cualquier parte del documento XML repetidamente haciendo uso de la misma).

Una vez que se tiene en una variable en memoria los datos del archivo, ya se puede hacer uso de ellos para la aplicación que se requiera.

Por ejemplo, se puede buscar en concreto una de las etiquetas del XML, con la sentencia expuesta a continuación:

NodeList items = Link_data.rootMapa.getElementsByTagName("Nombre etiqueta a buscar (floor, sector, room, line, etc.)**");**

La variable ítems sería una lista de elementos de la clase Node, que contendría todas las etiquetas encontradas de ese tipo. Una vez que se tiene un ítem, es decir un nodo de la lista, se podría obtener información de él tanto aguas arriba como aguas abajo. Por ejemplo, si la etiqueta buscada hubiese sido de tipo "line", se podría obtener información de sus nodos superiores en el árbol jerárquico (si está asociado a una sala, pasillo, etc. o en qué sector, planta se encuentra), mientras que, al no tener nodos inferiores, lo único que se podría obtener de él serían sus atributos. Con esto sería muy sencillo discriminar datos de cara a conseguir un mejor tiempo de procesamiento, por ejemplo, no es normal que se instalen balizas en un ascensor, por lo que podrías detectar si la pared de estudio pertenece a un ascensor y directamente no estudiarla.

Los principales métodos utilizados para el manejo de los datos del XML han sido los siguientes:

- **getParentNode();** -> para posicionarte en el nodo superior al de estudio.
- **getAttributes();** -> para posicionarte en los atributos del nodo en el que estés.
- **getNodeValue();** -> Cada dato almacenado en un XML, lleva asociado un nombre y un valor, para obtener el valor sería el método nombrado, mientras que para obtener el nombre se utilizaría el método *getNodeName*(). Destacar que el XML tiene formato texto, por lo que, si quieres operar con ese valor, lo tienes que pasar a formato numérico (bien a float, *Float.parseFloat*("nº"), o a entero *Integer.parseInt*("nº")).

c. Programación para la visualización sobre Google Maps

Una vez que se tiene configurado el proyecto para el uso de los mapas de Google, la API de Google Maps dará la opción de hacer uso e interactuar con el mapa a través de las diferentes librerías, clases y métodos que ofrece.

El método fundamental que hay que añadir en cada una de las actividades que van a hacer uso de los mapas, es el llamado *OnMapReadyCallback*. Con la implementación de este método se permite hacer uso de la función *onMapReady*, que es en la cual se vincula una variable con el mapa, permitiendo interactuar con el mismo a posteriori a partir de dicha variable. Por ejemplo, se puede realizar *zoom in* automático en la zona del mapa deseada (utilizando la clase *CameraPosition* y el método *animateCamera*) o detectar edificios en las coordenadas a las que está enfocando la cámara, y en el caso de que lo detecte, acceder al número de plantas que este tiene, pudiendo seleccionar la planta a visualizar (usando las clases *IndoorLevel* y *IndoorBuilding*).

Las principales funcionalidades de los mapas de Google utilizadas en este trabajo y que han sido cruciales para poder llevar a cabo la aplicación, se expresan a continuación:

- *setOnMapLongClickListener*: este método permite capturar eventos en las coordenadas en formato latitud longitud (que es con el que trabaja Google Maps) al realizar un *Long Click*, es decir, un click pronunciado sobre el mapa. Se ha realizado esta opción y no el click normal, para cerciorarnos de que los eventos capturados sean realmente de interés, y no capturar otro tipo de eventos, ya que con el método *setOnMapClickListener* se capturaría cualquier mínimo roce en la pantalla. Esta funcionalidad se ha utilizado para poder obtener las entradas del algoritmo metaheurístico tanto de la región aproximada dónde se encuentra el transductor central (una vez que el usuario pulse el botón '*Add approximately region*' se capturarán los eventos *LongClick*, y una vez se hayan capturado cuatro eventos de este tipo, se generará la región a partir de los valores *x* e *y* máximos y mínimos obtenidos) como para la selección de las paredes láser (ya que una vez que el usuario pulse el botón '*Selection of the laser projected walls*' se capturarán los eventos

LongClick y a partir de esa posición obtenida se buscará la pared más cercana).

- *addMarker*: este método además de añadir un marcador en el mapa en la posición indicada ofrece una serie de opciones en el mismo como seleccionar que sea visible, la posición donde se desea ubicar, si se le desea añadir un título, elegir su color, e incluso elegir otro icono generado por el usuario. Se utilizan marcadores para mostrar dónde ha pulsado el usuario a la hora de generar la región y para representar la posición del transductor central, cuando se pide sólo representar dicha posición.
- *addPolyline*: este método permite dibujar una recta entre dos puntos dados (cada punto especificado en coordenadas LatLng). Principalmente se ha utilizado para poder representar la región aproximada seleccionada y las paredes rectas seleccionadas.
- *addCircle*: este método representa círculos a partir de unos datos dados (centro del círculo y radio) y ha servido para visualizar en el mapa la posición en planta de los transductores de la baliza calculados.

A pesar de las numerosas ventajas que tiene el trabajar con una plataforma como Google Maps, a la hora de trabajar sobre dicho interfaz existen aún bastantes limitaciones, sobre todo a la hora de intentar plasmar algo sobre Maps. En nuestro caso se necesita poder pintar arcos (para representar las paredes curvas) y escribir (para indicar en la selección de las paredes cada pared seleccionada a que distancia correspondía), tareas que han sido necesarias abordar.

Como ya se ha comentado anteriormente, la API de Google puede dibujar tanto líneas como circunferencias, sin embargo, no dispone de métodos para representar arcos. Para ello, se ha tenido que crear una función que permita representarlos a partir de sus parámetros, tomando como referencia la información de los arcos proveniente del XML (en cual un arco se describe con 8 atributos: punto de origen, centro y fin, en coordenadas x e y, radio y rotación). Esta función calcula el ángulo entre los extremos del arco mediante la clase *SphericalUtil* y lo procesa en función del valor de *rotation* (orientación del arco). Una vez que tiene el valor del ángulo "procesado", a partir del centro y origen y hasta el punto final del arco, se segmenta el mismo en una serie de polilíneas, las cuales si se pueden representar.

Para poder mostrar textos sobre Google Maps, se ha tenido que desarrollar otra función, ya que la API no tiene método para hacerlo. Esta función que se ha implementado permite escribir sobre el mapa de Google, centrando el texto respecto a dos puntos conocidos, siendo el procedimiento de la misma el siguiente:

- Se obtiene la posición en formato LatLng de los 2 puntos.
- Se crea un *layout* que contenga un *textView* en su interior.
- Se escribe con el método *setText* sobre el *textView* generado.
- Se crea un *bitmap* con el *layout* generado, siendo el *bitmap* un objeto tipo que tiene implementado bastantes métodos para poder trabajar con *ImageView* y *layouts*, entre otros.
- Por último, se coloca el *bitmap* del *layout* con el texto, en el punto medio de los 2 introducidos y se añade como si fuera un *marker* (utilizando las propiedades del bitmap para insertar el *textView* como una "imagen", que será el icono del marcador).

Por último, para acabar con esta sección se expone un ejemplo en el cual se visualiza el resultado obtenido al hacer uso de las funciones desarrolladas tanto para poder dibujar arcos como escribir textos sobre el mapa de Google Maps.

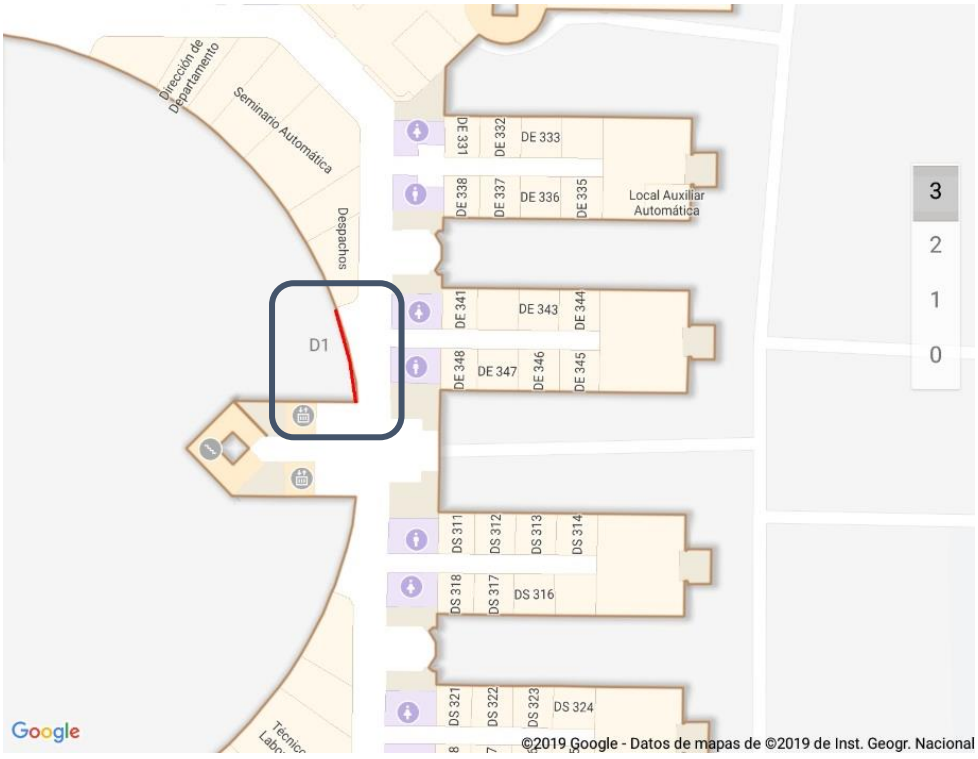


Figura 14: Ejemplo funciones desarrolladas para visualizar en Google Maps.

d. Programación de la conversión de formato LatLng a formato vectorial

La aplicación desarrollada captura los eventos generados sobre el mapa en el formato utilizado por Google, coordenadas latitud longitud, pero sin embargo para facilitar su desarrollo y ya que la información del mapa está en formato vectorial (*x*,*y*), la algoritmia relacionada con los mapas se ha desarrollado utilizando los datos en formato vectorial y una vez que se obtienen resultados correspondientes al sistema de referencia vectorial se pasan a formato latitud longitud para poder representarlos sobre Google Maps.

Para la conversión de las coordenadas *x* e *y* a formato LatLng, se ha utilizado una función realizada previamente en el grupo de investigación. Esa función transforma las coordenadas *x* e *y* a coordenadas UTM (sistema de coordenadas universal transversal de Mercator). Una vez se tienen las coordenadas UTM, se convierten a formato latitud longitud mediante la librería Jcoord.

En la aplicación desarrollada es de vital importancia que el usuario interactué con el mapa de Google para facilitar información de cara a la calibración de la baliza (región aproximada, selección de las paredes proyectadas por la plomada), por lo que es crucial poder convertir esa información capturada en LatLng a XML para poder manipular de manera más sencilla esa información.

Se ha desarrollado una función en Java que sea capaz de transformar coordenadas latitud longitud a coordenadas en formato vectorial. En primer lugar, se convierten las coordenadas a formato UTM mediante la librería Jcoord. Una vez se tienen las coordenadas en formato UTM se transforman a coordenadas XML mediante un procedimiento basado en una rotación (para conseguir la orientación del eje de coordenadas del archivo XML (véase la figura adjunta a continuación) con respecto a la orientación del sistema de referencia del mapa de Google) y una posterior traslación (que tiene que ver con la diferencia en coordenadas UTM entre el punto capturado y el punto de origen del eje de coordenadas).

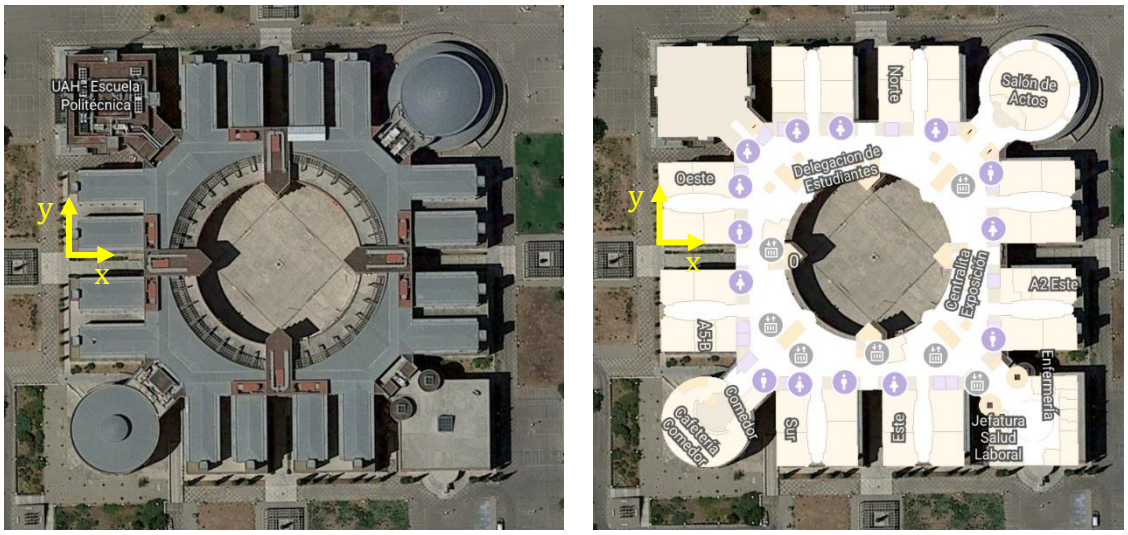


Figura 15: Eje coordenadas del mapa en formato vectorial, en vista satélite (izda) y con mapa interior (dcha).

Para validar la función desarrollada se ha realizado un barrido del edificio en cuestión, comprobando que para diferentes valores el algoritmo converge en una solución óptima. La prueba consistió en hacer un estudio por toda la superficie del edificio (cómo se puede observar en la figura expuesta a continuación), obteniendo las coordenadas en formato latitud longitud de los diferentes puntos y observando que al realizar el proceso inverso no difería del valor real.

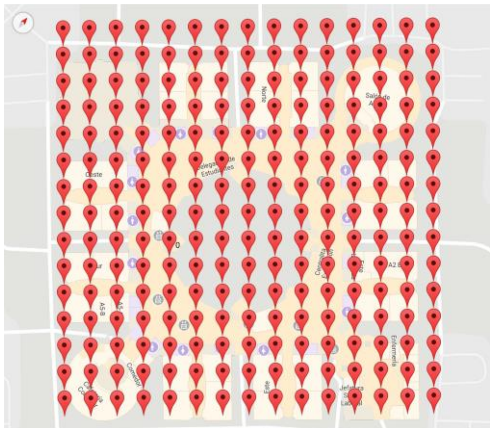


Figura 16: Puntos de estudio para la validación de la función de conversión.

e. Programación para la fusión de la información de los mapas en formato vectorial del edificio con los Google Maps

Como bien se ha explicado en secciones anteriores, al no poder utilizar la información de los mapas de Google, se ha utilizado la información del mapa desarrollada en un proyecto previo [García, 2011].

A continuación, se procede a exponer un resumen del funcionamiento de la aplicación según los algoritmos expuestos anteriormente en un caso concreto, por ejemplo, al seleccionar el usuario las paredes dónde ha proyectado el láser.

1. Se captura el evento generado en el mapa gracias al método *setOnMapLongClickListener* de Google Maps.
2. Se pasa esa información capturada del formato de trabajo de Google Maps (LatLng) a formato vectorial, mediante las funciones de conversión de coordenadas implementadas.
3. Al punto capturado en formato vectorial se le aplica el algoritmo desarrollado para la detección de la pared más cercana.
4. Una vez se tiene la pared más cercana y en consecuencia los atributos de la misma (obtenidos de la información del mapa vectorial), se convierten los datos a formato Google Maps (coordenadas latitud longitud) y se representa sobre el mapa dicha información de la pared.

A la hora de representar sobre el mapa de Google se observó que existían discrepancias entre los datos de Google y los del mapa del edificio, como se puede observar en la imagen expuesta a continuación:

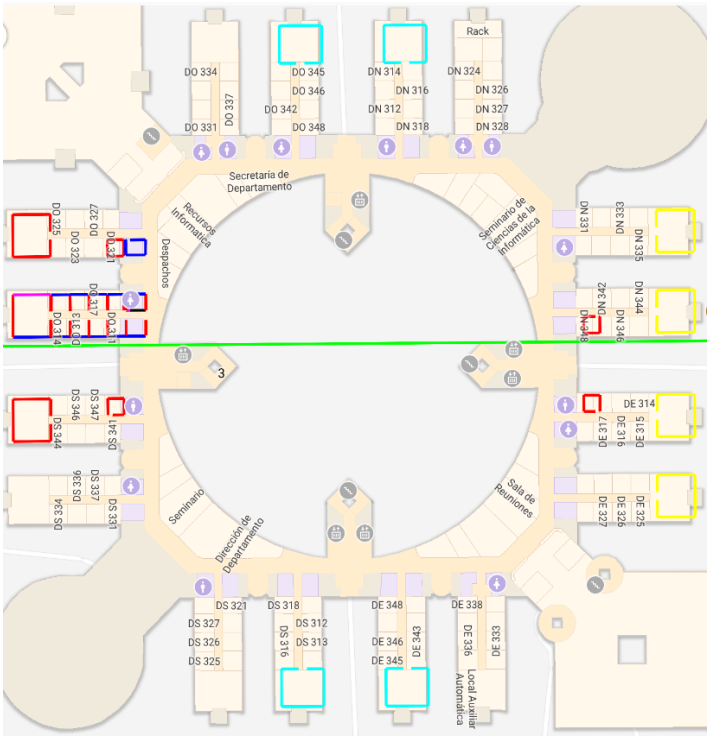


Figura 17: Prueba realizada para verificar discrepancias en la representación.

La figura 17 ilustra una prueba realizada en la cual se representaron en diferentes colores (rojo, azul, cian y amarillo) sobre el mapa de Google las paredes de varias salas del edificio según la información del mapa en formato vectorial. En esta figura se puede observar la falta de concordancia entre la información de ambos sistemas, que además se acentúa a la hora que se va aumentando el valor en el eje *x* (línea verde), cómo se puede observar a la derecha de la imagen con varias salas que incluso exceden los límites del edificio fijados por Google.

Para que coincida la información del mapa en Google Maps con la información que tenemos en formato vectorial, se han tenido que adaptar las funciones que trabajan con la conversión de datos tanto de formato LatLng a XML como al revés, para que la representación de los datos de trabajo a la hora de utilizarlos sobre Google Maps sea lo más fidedigna posible. Se ha hecho un barrido a lo largo del mapa obteniendo a partir de zonas identificadas, la diferencia en formato vectorial entre los valores en formato vectorial del mapa y los que deberían ser para que la representación sobre Google Maps fuera correcta.

Una vez que tenemos los errores que se van generando, y al observar que existe una correspondencia entre el error y la posición en el eje, se realiza una regresión lineal para cuantificar la modificación que habría que realizarles a los datos para representarlos adecuadamente.

A continuación, se expone una de las regresiones obtenidas, más en concreto la correspondiente para modificar el valor de la coordenada *x*.

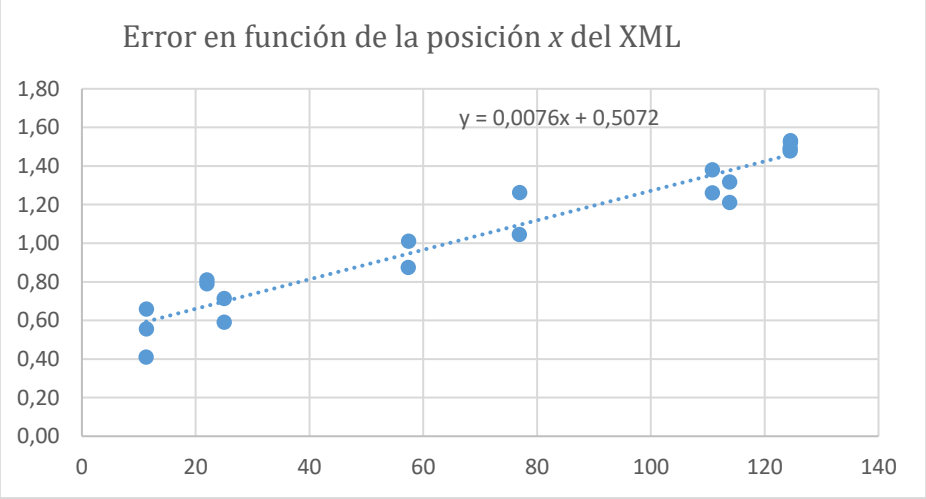


Figura 18: Regresión lineal del error obtenido en función de la coordenada x.

Por último, se expone una figura en la cual se representa toda la información del mapa en formato vectorial sobre el mapa de Google, tras las modificaciones realizadas.

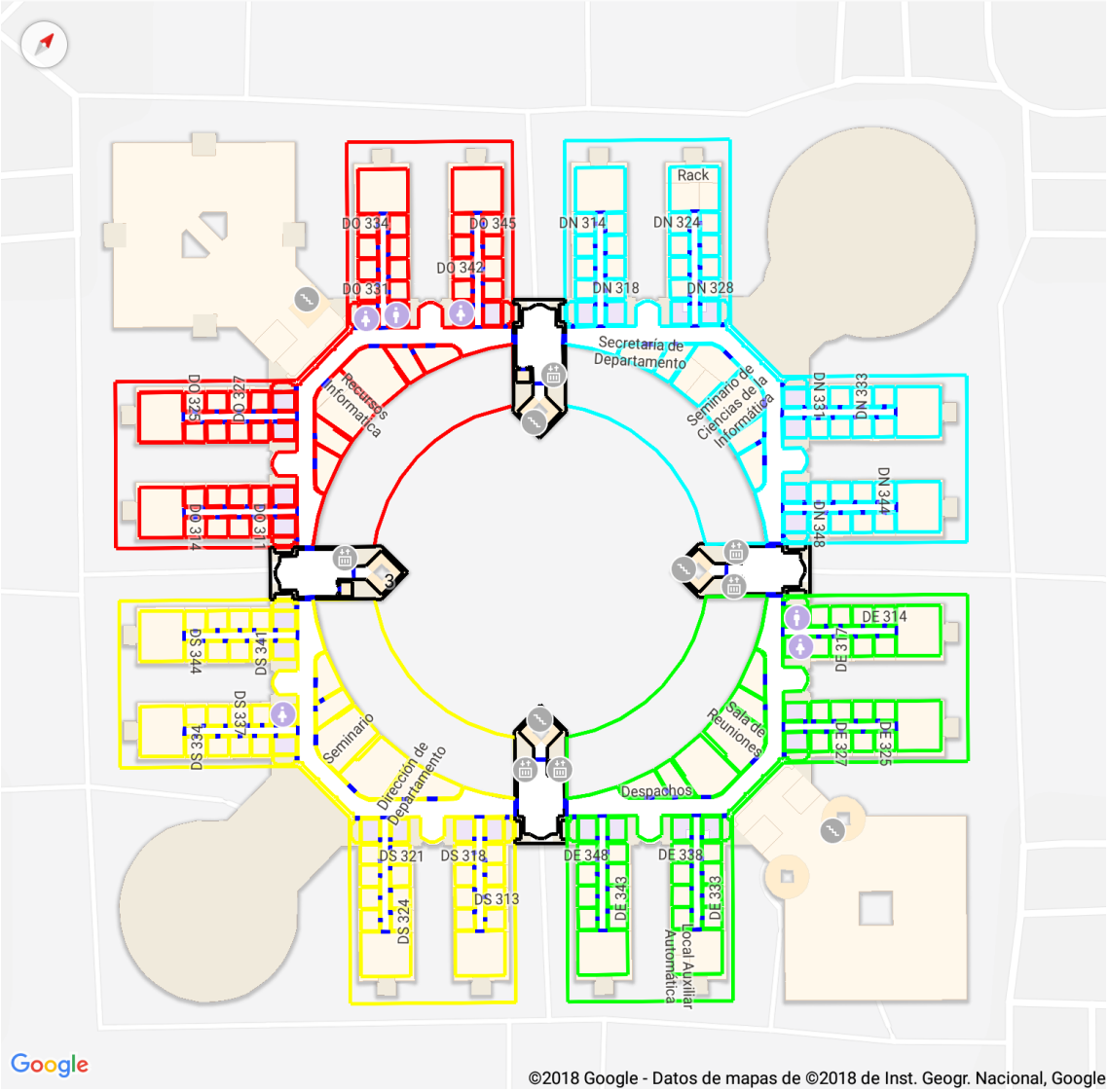


Figura 19: Representación de la información del mapa en formato vectorial sobre el mapa de Google.

Destacar la importancia de haber conseguido una correcta representación, ya que en una aplicación como la desarrollada en este trabajo la interacción con el mapa es fundamental (tanto para la elección/representación de la región aproximada y las paredes en las cuales proyecta el láser, como a la hora de representar la posición final de la baliza calibrada).

f. Programación del algoritmo de búsqueda de la pared más cercana

El algoritmo de búsqueda de la pared más próxima consiste en la detección de la pared más cercana detectada a un punto determinado introducido por el usuario.

El desarrollo de esta función ha sido necesario para que el usuario pudiera indicar sobre el mapa a qué paredes hacían referencia los valores introducidos sobre las distancias medidas a las paredes en las que había proyectado el láser.

A partir del punto seleccionado por el usuario ya convertido a formato vectorial, se propone hacer un barrido de las diferentes paredes del edificio, calculando la distancia entre el punto y las paredes, detectando la distancia mínima y representando sobre el mapa de Google la pared que hace referencia a esa distancia.

Para calcular la distancia entre un punto y una pared, se ha propuesto proyectar desde el punto de estudio una serie de líneas, en el caso de que alguna línea intersecte con la pared, se obtiene el módulo del vector entre el punto y la intersección, que será la distancia buscada.

En el edificio existen tanto paredes lisas como curvas, por lo tanto, se han tenido que estudiar dos tipos de intersecciones: entre dos rectas (caso pared lisa) y entre recta y curva (arco/circunferencia, caso pared curva).

- **Intersección entre dos rectas**

La ecuación general de una recta sigue la siguiente estructura:

$$Ax + By = \mathrm{E} \tag{4.1}$$

A partir de los atributos de una pared lisa, punto de inicio (*x1*,*y1*) y final (*x2*,*y2*) de la recta que la describe, se obtienen los parámetros de la ecuación general de la siguiente forma:

$$\begin{gathered} A = \mathrm{y2} - \mathrm{y1} \\ B = x1 - x2 \\ E = x1 * (y2 - y1) - y1 * (x2 - x1) \end{gathered} \tag{4.2}$$

El punto de intersección entre dos rectas se calcularía mediante el siguiente sistema de ecuaciones:

$$\begin{cases} Ax + By = E \\ Cx + Dy = F \end{cases} \tag{4.3}$$

Este sistema de ecuaciones es lineal y asumiendo que las rectas son linealmente independientes (no son paralelas, es decir el determinante de la matriz AB-CD es distinto de cero) se puede resolver mediante la regla de Cramer.

$$x = \begin{vmatrix} E & B \\ F & D \end{vmatrix} \Big/ \begin{vmatrix} A & B \\ C & D \end{vmatrix} = \frac{E * D - B * F}{A * D - B * C} \tag{4.4}$$

$$y = \begin{vmatrix} A & E \\ C & F \end{vmatrix} \Big/ \begin{vmatrix} A & B \\ C & D \end{vmatrix} = \frac{A * F - E * C}{A * D - B * C} \tag{4.5}$$

Como tanto las paredes como las líneas de intersección se han representado como rectas y las mismas no están delimitadas, una vez obtenido el punto de intersección en caso de que lo hubiera, habrá que comprobar que ese punto es correcto. Para ello se han realizado dos filtrados, uno para comprobar que el punto pertenece al tramo a la semirrecta de corte y otro para verificar que pertenece al tramo de la pared.

Para la verificación del método implementado se ha realizado una prueba en la cual se pretende obtener la pared de una sala más cercana a un punto situado en el interior de la misma (sala determinada por 5 paredes lisas). A continuación, se procede a exponer la prueba realizada y los resultados obtenidos.

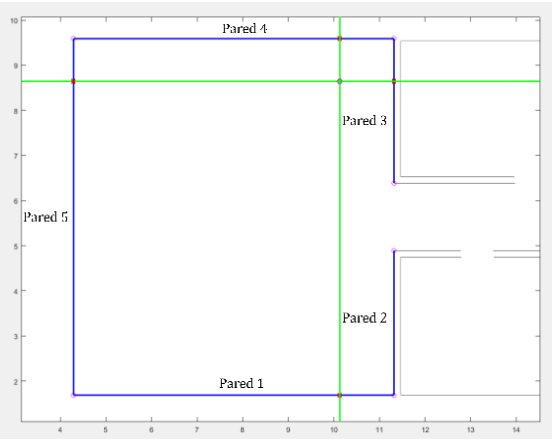


Figura 20: Prueba verificación detección pared lisa más cercana (I).

En primer lugar, se proyectan las líneas (representadas en verde en la figura ilustrada) desde el punto de estudio, obteniendo la intersección de las mismas con las paredes de la sala (en este caso intersecta con todas excepto con la pared 2, como se puede observar en la figura expuesta anteriormente).

Cada vez que se da una intersección que cumple con los filtros estipulados se calcula la distancia, y se comprueba su valor con el de la pared más cercana hasta ese momento, almacenando los atributos de la pared en el caso de que su distancia al punto fuera menor a la registrada. Destacar que la distancia entre dos puntos se calcula haciendo el módulo del vector que forman entre ellos.

$$dist(P,Q) = \sqrt{(P_x - Q_x)^2 + \left(P_y - Q_y\right)^2} \tag{4.6}$$

Una vez recorridas todas las paredes de estudio se representaría la pared más cercana al punto marcado por el usuario.

```
Punto marcado por el usuario: (10.132627,8.646259)
Distancia a la pared[1]= 6.956259 m.
Distancia a la pared[3]= 1.187373 m.
Distancia a la pared[4]= 0.943741 m.
Distancia a la pared[5]= 5.842627 m.
Hay 4 puntos de intersección.
La pared más cercana al punto introducido es la pared [4] con una
distancia de 0.943741 m.
```

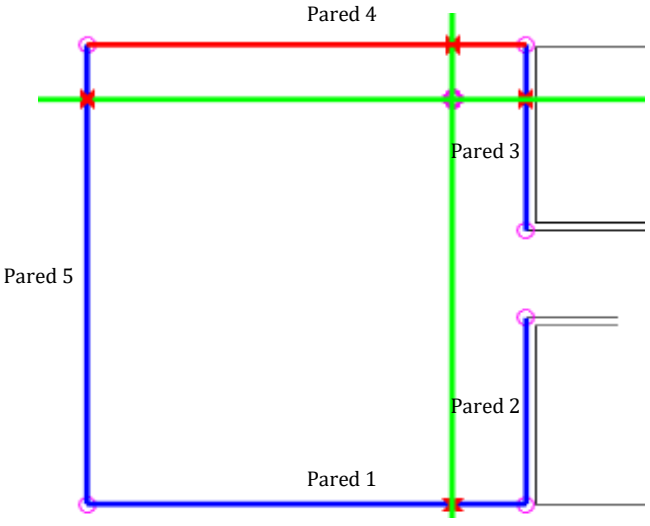


Figura 21: Prueba verificación detección pared lisa más cercana (II).

- **Intersección entre recta y circunferencia**

Una circunferencia está descrita por la siguiente ecuación:

$$(x - C_x)^2 + \left(y - C_y\right)^2 = r^2 \tag{4.7}$$

Siendo C el centro de la circunferencia y r el radio de la misma, todos ellos valores que se pueden obtener del XML, dónde las paredes curvas se identifican con la etiqueta 'arc'.

El punto de intersección entre la recta de intersección y la circunferencia que sigue el arco se calcularía mediante el siguiente sistema de ecuaciones:

$$\begin{cases} Ax + By = E \\ x^2 - 2 * C_x * x + y^2 - 2 * C_y * y = r^2 - {C_x}^2 - {C_y}^2 \end{cases} \tag{4.8}$$

Este sistema de ecuaciones es no lineal, por lo que su solución no es tan trivial como en el caso anterior. Para facilitar la implementación de este sistema en Java se ha propuesto desarrollar el sistema de manera teórica, hasta obtener la ecuación de segundo grado que englobe a las dos ecuaciones y esta ya implementarla en el programa.

La ecuación de segundo grado hacia la que tendería el sistema sigue la siguiente forma:

$$ax^2 + bx + c = 0 \tag{4.9}$$

Que desarrollada sería:

$$x = \frac{-b \pm \sqrt{b^2 - 4 * a * c}}{2 * a} \tag{4.10}$$

La resolución de este sistema al ser de segundo grado nos podría dar ninguna, una o dos soluciones en función de los valores de los parámetros. Estas tres posibles soluciones tienen que ver directamente con las posiciones relativas que se pueden dar entre una recta y una circunferencia.

- Si $(b^2 - 4 * a * c) < 0$, no habría solución es decir no habría intersección ninguna entre la recta y la circunferencia.

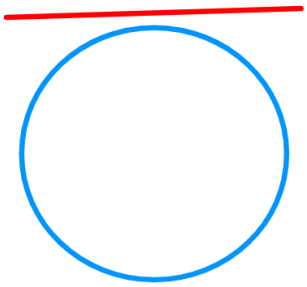


Figura 22: Posición relativa entre recta y circunferencia (I).

- Si $(b^2 - 4 * a * c) = 0$, habría solución doble es decir habría un único punto de intersección entre la recta y la circunferencia.

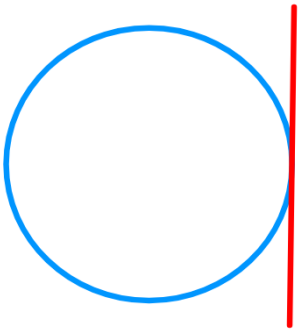


Figura 23: Posición relativa entre recta y circunferencia (II).

- Si $(b^2 - 4 * a * c) > 0$, habría dos soluciones posibles es decir habría dos puntos de intersección entre la recta y la circunferencia.

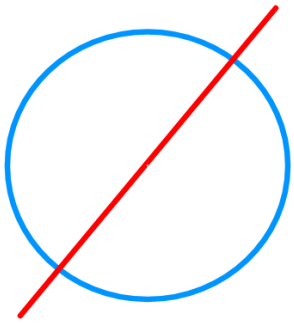


Figura 24: Posición relativa entre recta y circunferencia (III).

En el caso de que al resolver el sistema tuviéramos algún punto de intersección, para comprobar que ese punto es correcto, se realizarán dos filtrados, uno para comprobar que el punto pertenece al tramo a la semirrecta de corte y otro para verificar que pertenece al tramo de la circunferencia delimitado por el arco (región delimitada por los extremos del arco, valores definidos en el XML).

En el caso de que existiera un punto de intersección con una distancia al punto menor que alguna obtenida anteriormente, se almacenaría el valor de la pared, haciendo hincapié si este mínimo obtenido hace referencia a una pared lisa o curva, de cara a la representación futura sobre el mapa.

Por último, se exponen un par de ejemplos del funcionamiento del método propuesto para obtener la intersección entre una recta y una circunferencia en las paredes curvas del despacho 0301.

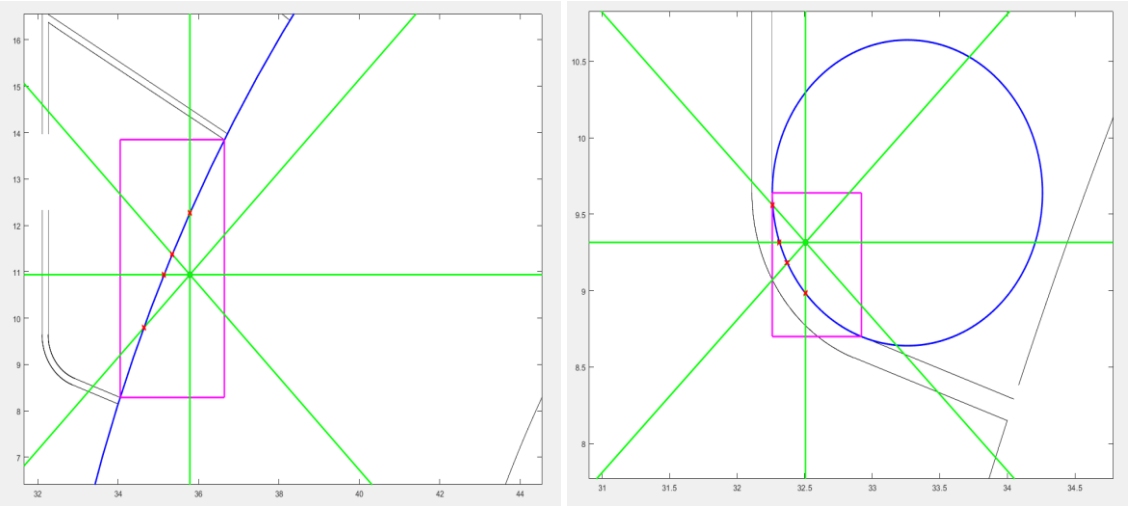


Figura 25: Prueba verificación detección pared curva más cercana.

En ambas imágenes ilustradas se puede observar como se obtienen las intersecciones que corresponden a la pared curva de estudio (delimitada por el área descrita en magenta), a partir de las líneas tiradas (representadas en verde) desde un punto determinado, que sería el seleccionado por el usuario sobre el mapa.

Destacar que este algoritmo no busca obtener el valor exacto de la distancia mínima, sino que pretende de manera rápida obtener la pared más cercana a un punto, por lo que para esta aplicación el método propuesto es completamente válido.

g. Programación de los algoritmos metaheurísticos

En este apartado se procede a explicar la implementación de los algoritmos metaheurísticos expuestos en el capítulo de la base teórica.

Antes de pasar a exponer las características específicas de cada método, se explicarán los aspectos que ambos algoritmos tienen en común.

Los algoritmos metaheurísticos poblacionales son capaces de converger a soluciones cercanas al óptimo mediante la evaluación de un reducido grupo de soluciones.

Según la bibliografía [Gualda, 2019], el tamaño adecuado del conjunto de soluciones que habría que estudiar en cada iteración sería de 10 veces el número de variables que quieras estimar. En este caso, al querer estimar la posición en formato vectorial de la posición central de la baliza y la orientación de la plomada láser, tendríamos tres variables a estimar por lo que el conjunto de posibles soluciones a tratar en cada solución sería de 30.

Destacar que las soluciones se generan dentro de los posibles márgenes introducidos por el usuario en las diferentes entradas del algoritmo y que el primer conjunto de soluciones creadas se genera de manera totalmente aleatoria dentro de esos posibles límites. Por lo tanto, para que la solución final del algoritmo no dependa de lo bien o lo mal que haya introducido el usuario los datos de entrada ni de la suerte tenida con la población inicial generada, se ha decidido que el número de combinaciones totales sea un valor dinámico, proporcional a los *bounds* introducidos por el usuario. De esta forma se busca que ante cualquier casuística posible el algoritmo sea capaz de converger en una solución adecuada.

La bondad de las combinaciones evaluadas se realiza con la función aptitud que calculará, para cada posible combinación de solución, las distancias a las paredes ($\hat{\mathrm{d}}_i$) desde la posición estimada ($\hat{\mathrm{x}}_B, \hat{\mathrm{y}}_B$) a las paredes de estudio con la orientación estimada de las medidas ($\hat{\theta}_B$). Cuanto menor error haya respecto a la observación (valor introducido por el usuario, el obtenido con el medidor laser, d_i), mejor será la solución.

No existe ningún criterio para determinar el número de iteraciones totales que deben realizarse, aunque una de las principales ventajas de este algoritmo tiene que ver con esta variable, ya que los algoritmos metaheurísticos son capaces de evaluar únicamente las combinaciones con mayor probabilidad de contener el mínimo error. Es decir, se ahorra estudiar zonas con baja probabilidad, ganando rapidez de cómputo, obteniendo el mínimo absoluto con una probabilidad cercana al 100%, mientras que utilizando el método de fuerza bruta, no hay forma de restringir donde

puede estar el mínimo e itera tantas veces como posibles combinaciones haya. Esta característica de los algoritmos metaheurísticos ha sido fundamental a la hora de su elección para el desarrollo de esta aplicación, ya que se pretende ejecutar en tiempo real.

Por lo general, en la literatura los algoritmos metaheurísticos utilizan en torno al 0.5% de las combinaciones totales (asegurando incluso algún decimal de precisión). En este trabajo se propone utilizar en torno al 5% del total, es decir se evaluará un mayor número de combinaciones que las necesarias, siendo más conservadores, para asegurarnos que la solución sea lo más fiable posible (no sea un falso mínimo). Con el porcentaje de combinaciones totales seleccionado, el proceso de calibración tardará en torno a 1 minuto, tiempo por otro lado totalmente asumible, ya que la calibración de la baliza sólo se realiza cuando se modifica la posición de la misma (algo no habitual).

i. Algoritmo genético

La implementación de este algoritmo se ha llevado a cabo gracias al uso de la librería JGAP (*Java Genetic Algorithms Package*), la cuál como cualquier librería externa ha de introducirse en la carpeta del proyecto en la pestaña libs de la carpeta app y añadirse su dependencia correspondiente y sincronizar el proyecto, para que la ejecute el compilador y sea capaz de utilizar sus funcionalidades en el programa.

Para estar familiarizados con el lenguaje utilizado en esta sección destacar que se llama *población* al conjunto de soluciones (*Genotype*), *individuo/cromosoma* a cada una de las soluciones (*IChromosome*) y *gen* a cada variable de la solución (*Gene*).

JGAP es un framework capaz de simular un ambiente para que se desarrolle y sobreviva la mejor solución, haciendo uso principalmente de los siguientes métodos:

- *randomInitialGenotype()*: crea soluciones aleatoriamente, tantas como las prefijadas con el método *setPopulationSize()*.
- *evolve()*: aplica operadores genéticos (mutación y combinación) para que surjan nuevas generaciones de posibles soluciones, tantas como número de iteraciones haya determinadas.

- *evaluate()*: función de ajuste que pone a prueba cada solución (*Fitness Function*). Les asigna un valor un valor numérico en función de su calidad, llamado fitness que es inversamente proporcional al error obtenido con la función aptitud. Los más adaptados de cada generación son los que tendrán más opciones de reproducirse, de cara a generaciones posteriores.
- *getFittestChromosome()*: retorna el individuo mejor adaptado de la población.

Destacar que los métodos de la librería JGAP deben programarse dentro de una función extendida de una excepción o en una excepción sin más, ya que si no se producen errores en la ejecución.

ii. Algoritmo de búsqueda armónica

Este algoritmo está menos extendido en la literatura que los algoritmos genéticos, de ahí que para su implementación se haya tenido que desarrollar el mismo desde cero, ya que no se ha encontrado una librería ya creada.

Para su desarrollo se ha seguido el esquema expuesto en la Figura 7, destacando en su implementación dos aspectos: la configuración de los parámetros del algoritmo y la manipulación del array de la memoria armónica.

Una vez expuesto que según la bibliografía el tamaño de la memoria armónica en nuestro caso debe ser de 30 posibles soluciones, se procede a explicar que valores se han seleccionado para el resto de los parámetros del algoritmo.

El HMCR (tasa de consideración de la memoria armónica) es una variable que, en función de su valor, considera más válidos o menos los valores de la memoria armónica de cara a la generación de nuevos armónicos. Es decir, si el parámetro HMCR vale 0, directamente el nuevo armónico será un valor aleatorio, mientras que, si vale un valor más cercano a 1, se iría fijando más en los valores de la memoria de cara a la generación de los armónicos. En nuestro caso se ha seleccionado un valor de 0.9.

Los parámetros PAR (tasa de ajuste de tono) y bw (ancho de banda de ajuste de tono) al haber implementado el algoritmo de búsqueda armónica mejorado serán

parámetros dinámicos, por lo que de ellos se determinan los límites que puede tomar el parámetro. El parámetro PAR crece linealmente con el número de improvisaciones, mientras que bw decrece exponencialmente, consiguiendo que con el paso de las iteraciones se realice un ajuste más preciso para la generación de nuevos armónicos en torno a las soluciones ya generadas, dando cada vez menos peso a la aleatoriedad, ya que al generar una variable de la solución si un número aleatorio es menor o igual que el PAR se realiza el ajuste ("mutación"), siendo dicho ajuste el valor generado de esa solución más o menos el BW (en función si se ajusta por arriba o por abajo, misma probabilidad para ambos casos). Los valores fijados para los parámetros PAR_{min}, PAR_{max}, bw_{min} y bw_{max} han sido 0.1, 0.5, 0.01 y 0.05 respectivamente.

En cuanto a la manipulación de la memoria armónica (HM), destacar, que en este algoritmo es fundamental que la HM esté ordenada, ya que interesan de la misma el valor con mejor y peor aptitud. Al ser un proceso que se va a realizar muchas veces durante la ejecución del programa (tantas como iteraciones), se intentará buscar la forma más eficiente posible desde el punto de vista computacional para el desarrollo de la misma. De ahí que se haya utilizado el método *Arrays.sort* de Java, que incluye en su interior una función comparadora (*compare*). Este procedimiento es capaz de ordenar la matriz de la memoria armónica (matriz 30x4, 30 filas debido al número de posibles soluciones y 4 debido a las variables de la solución, x, y y θ, y al valor aptitud de cada solución) en función de la columna aptitud.

iii. Comparativa y verificación de los algoritmos metaheurísticos implementados

En este apartado se exponen cómo y qué resultados se han obtenido de las pruebas realizadas para la verificación de los algoritmos desarrollados sobre plataforma Android.

Las pruebas han consistido en la aplicación del algoritmo metaheurístico a partir de unas entradas determinadas obtenidas con medidas reales.

En primer lugar, se explicará cómo se han obtenido las distancias del transductor central a la pared:

- Se sitúa la plomada debajo del transductor de estudio y se fija la plomada en el punto en el que la proyección vertical de la plomada coincida con el transductor.
- Se marcan las proyecciones horizontales en las paredes.
- Estimar aproximadamente el rango en el que podría estar la orientación de la plomada láser.
- Se mide con el medidor láser desde dónde estaba situada la plomada hasta las proyecciones obtenidas, teniendo especial cuidado en poner el medidor a la altura en la que tiene el láser la plomada, para que la medida sea lo más paralela posible, introduciendo así el mínimo error.

Con esto ya tendríamos las medidas a las tres paredes donde el láser ha proyectado y la orientación aproximada de la plomada.

Para obtener la posición real del transductor y poder hacer luego la comparativa del valor obtenido en el algoritmo con un dato real, habría que medir desde la proyección vertical del transductor la distancia a dos puntos conocidos en el mapa y luego geometricamente obtener la posición de ese punto. Para intentar introducir el mínimo error posible en las medidas, debe asegurarse que el medidor láser proyecte en la pared y no en cualquier elemento estético (como pudiera ser un rodapié), ya que esto implicaría un error acumulativo en el resultado final.

En cuanto a esta prueba, el entorno de trabajo corresponde al laboratorio F31 Oeste de la tercera planta de la EPS UAH, siendo el área aproximada dónde se encontraría el transductor y las paredes proyectadas las expuestas a continuación:

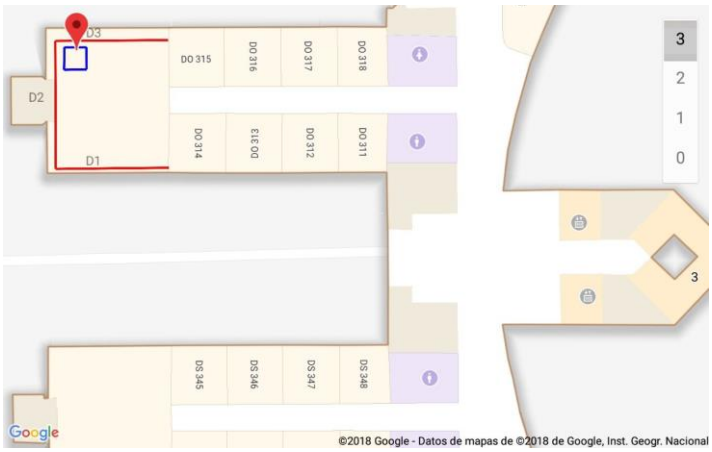


Figura 26: Entorno de trabajo pruebas verificación algoritmos metaheurísticos.

En la imagen expuesta se ha dibujado también la posición calculada teóricamente, valor que en condiciones normales de funcionamiento de la aplicación no se tendrían (valor a calcular mediante el algoritmo), para describir mejor el entorno de la prueba.

La primera prueba de estudio se realizó para una orientación aproximada de la plomada de en torno a -90º con respecto al semieje positivo del sistema de referencia del mapa, y con unas proyecciones a las paredes de 7.239, 1.315 y 0.611m para D1, D2 y D3, representadas en rojo, verde y azul respectivamente, como se puede observar en la siguiente figura:

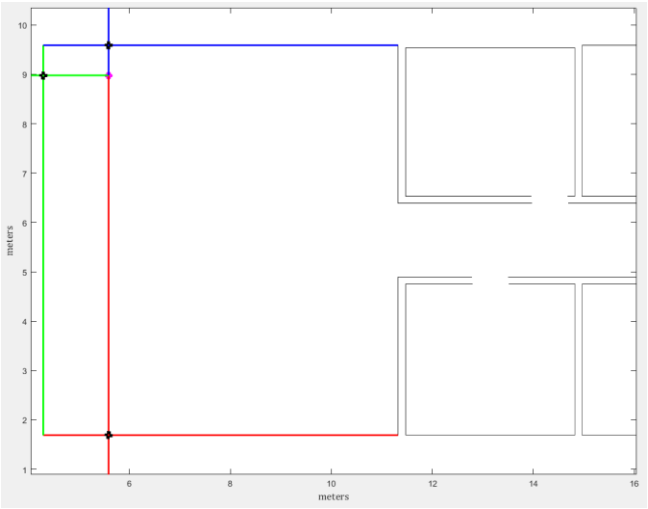


Figura 27: Proyecciones plomada láser Prueba I para la verificación de los algoritmos metaheurísticos.

La comparativa de los resultados obtenidos en ambos algoritmos para un 1% de las combinaciones totales es el siguiente:

	Resultado teórico	**Resultado obtenido con GA**	**Resultado obtenido con HS**
X [m]	5.585	5.602973470032881	5.60501054359651
Y [m]	8.98	8.971377204140587	8.976752429345005
θ (º)	≃90 [85,95]	-89.944	-89.9934
Tiempo de procesamiento [ms]	X	183	234

Tabla 2: Comparativa Prueba I para la verificación de los algoritmos metaheurísticos.

Destacar que como se expresó en capítulos anteriores, el algoritmo de búsqueda armónica obtiene valores más cercanos al valor teórico, sin embargo, hay algo que no se adecua con lo dicho anteriormente, el tiempo de procesamiento.

Aunque el algoritmo de búsqueda armónica sea en cuanto a algoritmia menos complejo con respecto al genético, hay dos causas principales de que el algoritmo genético tarde menos en converger:

- Para el desarrollo del algoritmo genético se ha utilizado una librería de Java externa, mientras que el de búsqueda armónica sí se ha tenido que implementar desde cero sobre plataforma Android, por lo que, aunque se haya implementado intentando hacerlo lo mejor y más eficiente posible, el otro algoritmo está mucho más depurado.
- Aunque se haya ejecutado en ambos algoritmos las mismas condiciones de entrada, la comparativa entre ambos no es totalmente equitativa, ya que según como está programado, el algoritmo genético es capaz de descartar sin tener que evaluar algunas soluciones de la población, a las cuales considera directamente no aptas, ganando eficiencia de computo.

En cuanto al error obtenido destacar que además del error intrínseco debido a la toma de medidas, el error también depende en gran parte del número de combinaciones de estudio del algoritmo.

Para demostrar el error debido al número de combinaciones se ha realizado un estudio de los resultados obtenidos en el algoritmo genético variando para las entradas de estudio de la prueba I el número de evoluciones del mismo (destacar que cada evolución, conlleva a una nueva generación, es decir a 30 nuevas combinaciones posibles, se ha realizado así ya que, aunque número de combinaciones totales no se puede controlar al hacer uso de la librería JGAP, el número de evoluciones sí). Se ha obtenido el error entre los valores teóricos y los obtenidos con el algoritmo, realizando el proceso 10 veces para cada número de evoluciones determinado obteniendo los siguientes valores:

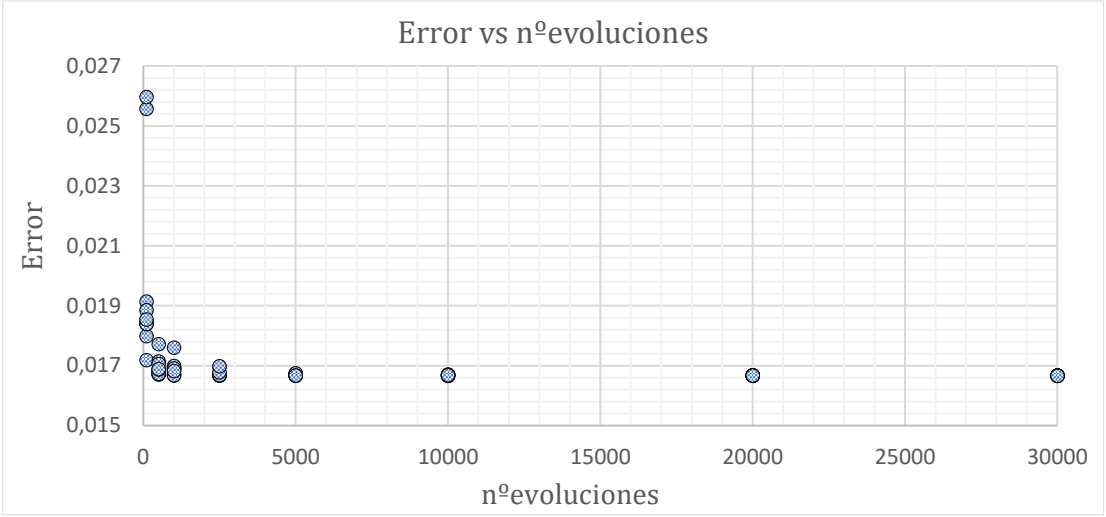


Figura 28: Prueba comparación error obtenido en función del número de combinaciones realizadas.

Nºevoluciones	**TEjecución medio (ms)**	**Error medio**
100	150,4	0,019854873
500	575,7	0,016950067
1000	999,1	0,0169127
2500	2336,4	0,016761551
5000	4451,2	0,016693354
10000	9206,6	0,016681007
20000	21486,2	0,016673753
30000	32415,5	0,016670621
100000	135891,8	0,016669484

Tabla 3: Comparativa error obtenido en función del número de combinaciones realizadas.

Como se puede observar en el gráfico de dispersión, a menor número de combinaciones posibles, el resultado obtenido depende más de la población inicial obtenida, mientras a mayor número de combinaciones, el algoritmo es capaz de converger en soluciones óptimas independientemente de la población inicial. Por tanto, los resultados obtenidos verifican lo explicado en esta sección sobre la importancia de haber programado el número de combinaciones totales de manera dinámica, para que independientemente de la población generada y de la bondad de las entradas introducidas por el usuario, el propio algoritmo sea capaz de obtener resultados fiables.

Destacar que hay que llegar a una relación de compromiso entre el número de combinaciones totales (que establece el tiempo de ejecución aproximado del algoritmo) y el error obtenido, ya que llegado a un punto el propio error existente se debe al error introducido por las propias entradas, por lo que no merece la pena seguir incrementando el número de combinaciones, ya que no implica mejora en los resultados obtenidos. Por ejemplo, este error se puede observar en los resultados medios obtenidos entre el caso de 30000 y 100000 evoluciones, ya que implica un tiempo de ejecución mucho mayor, para una diferencia en el error del orden de la millonésima (10E-6). En nuestro caso esa relación de compromiso entre los resultados obtenidos y el tiempo de ejecución se ha establecido en que el número de iteraciones del algoritmo corresponda en un 5% de las posibles combinaciones totales.

Para finalizar la verificación de la robustez y la comprobación del funcionamiento de los algoritmos desarrollados, se ha realizado otra prueba colocando la baliza en la misma proyección vertical del transductor, cambiando la orientación de la plomada láser, obteniendo unas proyecciones a las paredes de 7.246, 1.35 y 0.626 metros para D1, D2 y D3 respectivamente, como se puede observar en la siguiente figura:

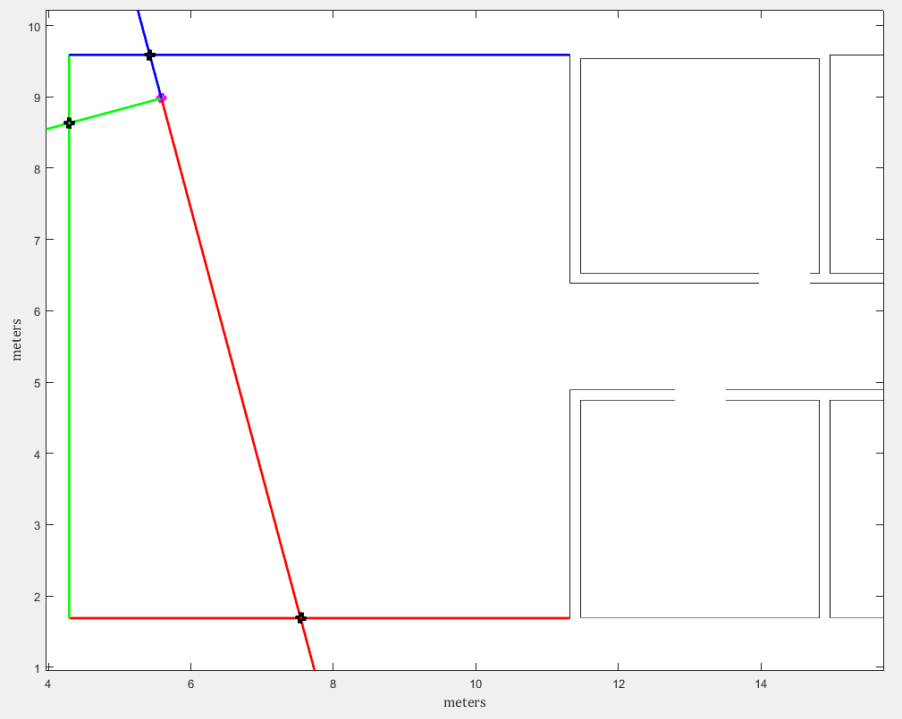


Figura 29: Proyecciones plomada láser Prueba II de la verificación de los algoritmos metaheurísticos.

Siendo la comparativa de los resultados obtenidos en ambos algoritmos para un 1% de las combinaciones totales la siguiente:

	Resultado teórico	**Resultado obtenido con GA**	**Resultado obtenido con HS**
X [m]	5.585	5.60837972257908	5.61461464662601
Y [m]	8.98	8.966515068670148	8.975861675708853
θ (º)	≃ [70,90]	-78.5491	-78.8399
Tiempo de procesamiento [ms]	X	245	412

Tabla 4: Comparativa Prueba II para la verificación de los algoritmos metaheurísticos.

Como rasgo más característico de esta prueba respecto a la anterior, decir que los tiempos de procesamiento se han incrementado, y esto se debe a que el rango de la posible orientación de la plomada ha aumentado. Al estar programado de manera dinámica y tener unas entradas más distantes, habrá mayor número de combinaciones de estudio, para que el algoritmo aun así sea capaz de converger en una solución precisa.

En cuanto al error obtenido en esta prueba, se ha realizado para un número de evoluciones determinado (30000) la diferencia entre los valores teóricos y en los que convergía el algoritmo, realizando el proceso 10 veces, obteniendo un error medio en esas pruebas de 2.076E-5. Destacar que es un valor bastante inferior al obtenido en la prueba I (error medio de 1.67E-2) y esto al realizar el mismo número de evoluciones y entorno de trabajo ambos, se debe a las entradas introducidas por el usuario. Se ha querido mostrar este caso, para representar la importancia a la hora de obtener las entradas del algoritmo (principalmente las distancias medidas), ya que cómo se ha demostrado en este apartado, tal y como se han programado los algoritmos metaheurísticos son capaces de obtener la solución óptima, por lo que es de vital importancia introducir el mínimo error posible con las entradas introducidas para intentar corromper lo mínimo los resultados obtenidos respecto a los valores reales.

h. Programación del algoritmo para la obtención de la posición de los transductores

Una vez que se ha ejecutado el algoritmo metaheurístico y se tiene la posición del transductor central, se exponen los resultados obtenidos en los correspondientes *textView* del apartado de *Outputs* y se abren dos formas diferentes de representación en función del modo de calibración que se está realizando (baliza situada en el techo o en la pared).

Destacar que las capturas que se van a exponer a lo largo de la explicación del algoritmo corresponden a figuras obtenidas en el Matlab, que es en el software en el que se han desarrollado los algoritmos.

i. Representación de la posición de la baliza en el techo

Se le permite al usuario, mediante un *Alert Dialog*, la opción de poder visualizar únicamente el transductor central o representar la posición de todos ellos.

Al pulsar la opción de la posición del transductor central únicamente, se representa un marcador en la posición del mismo, como se observa en la figura siguiente:

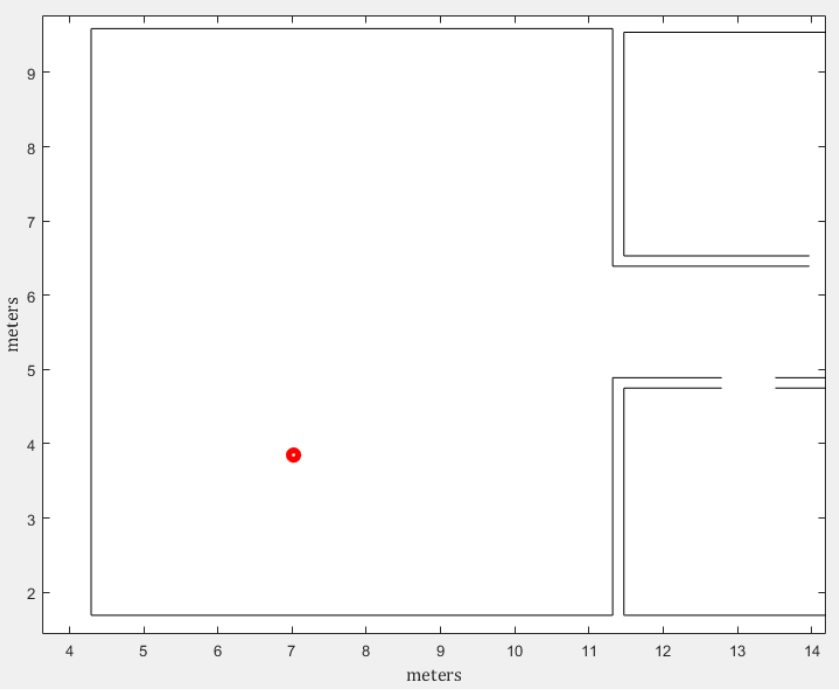


Figura 30: Representación del transductor central con la baliza en el techo.

Si se pulsa a la opción de representar los 5 transductores de la baliza, se abre otro *Dialog*, solicitando información adicional para poder realizar dicha representación. Al saber la posición del transductor central y la geometría de la baliza, se puede obtener la posición de los 4 transductores no centrales a partir de información de uno de ellos. La información adicional que se ha requerido es que el usuario indique el ángulo aproximado que forma el vector entre el transductor dos y el central con respecto al semieje de abscisas del mapa. A continuación, se expone un ejemplo en el cual, a partir de la misma posición del transductor expuesta en la figura anterior, se intenta representar la posición del resto de transductores dando un ángulo de 135º del transductor 2 respecto al semieje positivo del eje en formato vectorial.

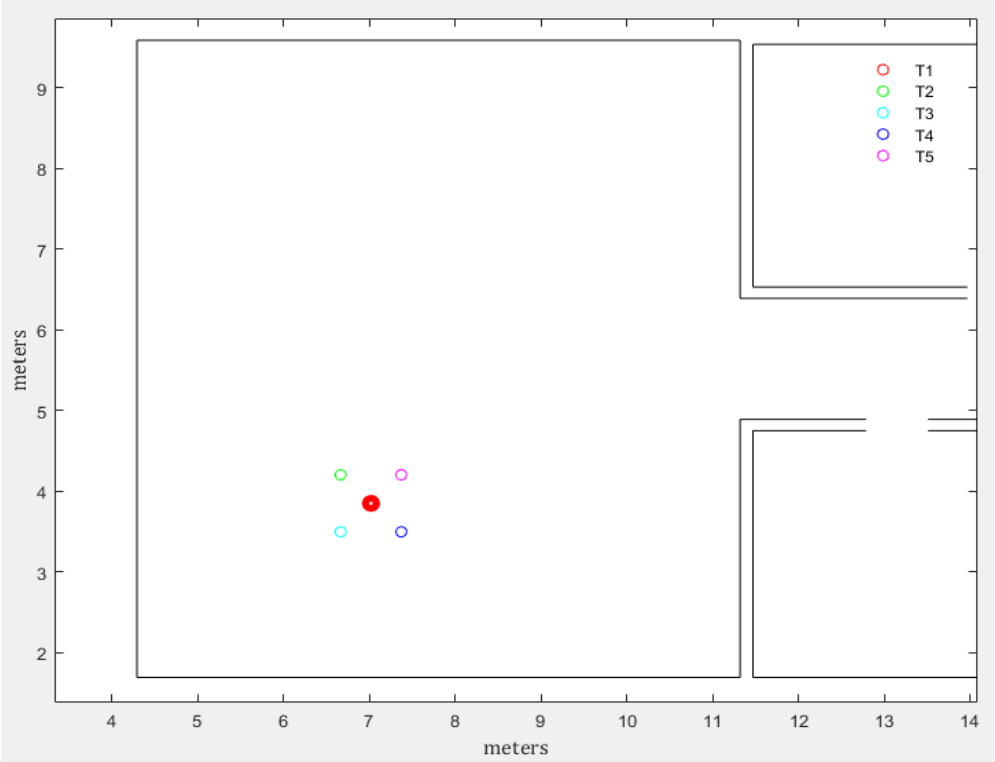


Figura 31: Representación transductores-baliza en el techo.

La posición de los transductores obtenida a partir de la información facilitada sería la siguiente:

```
Posición transductor 1 (rojo) = [7.018500, 3.848700].
Posición transductor 2 (verde) = [6.664947, 4.202253].
Posición transductor 3 (cyan) = [6.664947, 3.495147].
Posición transductor 4 (azul) = [7.372053, 3.495147].
Posición transductor 5 (magenta) = [7.372053, 4.202253].
```

Destacar que, en este modo de calibración, se obtienen la posición 2D de los transductores. No se obtiene la posición 3D, debido a que en el techo obtener la posición tridimensional sería muy sencillo, ya que consistiría únicamente en medir la altura desde el suelo de cada uno de los transductores. De esta forma se evita introducir un posible error a la hora de estimar las posiciones, que pudiera corromper los resultados obtenidos con el algoritmo metaheurístico.

ii. Representación de la posición de la baliza en la pared

En el caso de calibración de la posición de los transductores de la baliza cuando está situada en la pared, no es para nada trivial la obtención de ninguna de las coordenadas de los transductores, por lo que directamente se calcularán las coordenadas 3D de todos ellos. Para ello además de las entradas necesarias para realizar el algoritmo metaheurístico, se han de introducir adicionalmente la altura del transductor central y el ángulo del transductor 2 respecto al suelo.

El procedimiento desarrollado para la obtención de dichas coordenadas es el siguiente:

1. Se obtiene la **posición 2D del transductor central** (T_1) mediante el algoritmo metaheurístico.

Figura 32: Obtención coordenadas de los transductores - baliza colocada en la pared (I).

2. Se busca la **pared más cercana a dicho punto**. Por la geometría de la baliza, la pared más cercana a la posición del transductor central siempre va a ser la pared en la que esté instalada la baliza. Se detecta la pared más cercana, almacenando sus atributos.

Figura 33: Obtención coordenadas de los transductores - baliza colocada en la pared (II).

3. Al conocer la recta de la pared dónde está instalada la baliza y las coordenadas de T_1, se puede obtener la **proyección de T_1 en la pared**, ya que será la intersección entre la recta de la pared y la recta con vector director perpendicular a la pared y que pasa por T_1.

Figura 34: Obtención coordenadas de los transductores - baliza colocada en la pared (III).

4. Una vez tenemos la proyección del transductor central en la pared, intentamos obtener las **proyecciones del resto de transductores**. Para ello se ha propuesto el siguiente desarrollo:
    - Se obtiene el ángulo formado por la pared y el eje *x*.
    - Se identifica el tipo de pared dónde está instalada la baliza con el vector director de la misma, haciendo distinción entre paredes de pendiente infinita o cero (verticales o horizontales), paredes de pendiente positiva y paredes de pendiente negativa.
    - Se procesa el ángulo de rotación en función de la posición del usuario frente a la pared (que se corresponde con el vector formado entre la posición y la proyección de T_1) para que la representación de las proyecciones concuerde con la información introducida por el usuario del ángulo que forma T_2 respecto al suelo.

Figura 35: Obtención coordenadas de los transductores - baliza colocada en la pared (IV).

Destacar que para este ejemplo propuesto el ángulo del transductor dos respecto a la pared era de 60º y una altura de T_1 de 2m, es decir, la vista que tenía el usuario de la baliza a calibrar en alzado seguiría la siguiente forma:

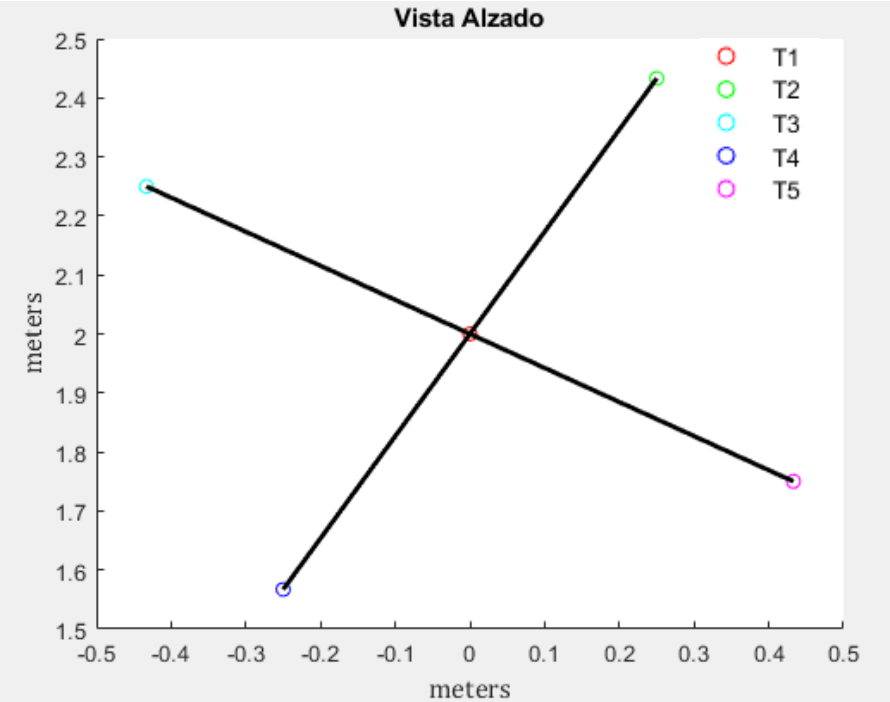


Figura 36: Obtención coordenadas de los transductores - baliza colocada en la pared (V).

En función del ángulo que forma el ángulo en la pared se puede obtener geométricamente la posición de las balizas en un plano recto. Las proyecciones de los transductores deben estar contenidas en la pared (no tiene por qué ser un plano recto) y dependen de la ubicación del transductor central, de ahí el algoritmo desarrollado, que permite la obtención de un ángulo de rotación que permita una correcta representación en las diferentes casuísticas posibles.

A continuación, se expone la representación en planta de las proyecciones de las coordenadas de los transductores en la pared en dos casos con planos de trabajo más complejos:

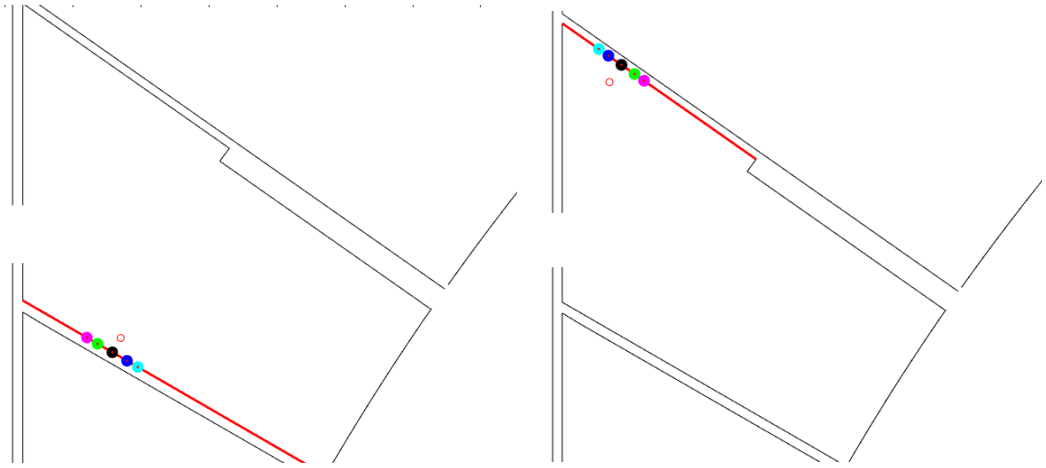


Figura 37: Obtención coordenadas de los transductores - baliza colocada en la pared (VI).

Como se puede observar en los ejemplos mostrados, el algoritmo propuesto sigue funcionando correctamente ante otro tipo de casuísticas, con lo que queda verificado el código desarrollado.

5. Una vez se tienen las proyecciones de cada transductor en la pared y sabiendo la geometría de la baliza, se realiza la traslación de estos a sus planos de trabajo, obteniendo las **coordenadas del resto de transductores**. El plano de trabajo del transductor central se encuentra a una distancia de 14 y 8cm respecto a los planos de trabajo de T_2-T_4 y T_3-T_5 respectivamente. El procedimiento utilizado ha sido el siguiente:
   - Se crea una recta paralela a la pared de instalación de la baliza y que pasa por el transductor central.
   - A partir de esta recta se generan dos rectas paralelas a ella y a la distancia de los planos de trabajo de los otros transductores.
     La distancia entre rectas paralelas, que sigue la siguiente fórmula:

$$dist(r,s) = \frac{|C - C'|}{\sqrt{A^2 + B^2}} \tag{4.11}$$

     Siendo A, B, C y C' los parámetros de las ecuaciones generales de las rectas paralelas:

$$\begin{cases} r: Ax + By + C = 0 \\ s: Ax + By + C' = 0 \end{cases} \quad (4.12)$$

- Se proyectarán rectas perpendiculares a la pared y que pasen por las coordenadas de las proyecciones de los transductores.
- Las intersecciones obtenidas de esas rectas con los planos de trabajo correspondientes serán las coordenadas de los diferentes transductores.

La representación final obtenida sería el siguiente:

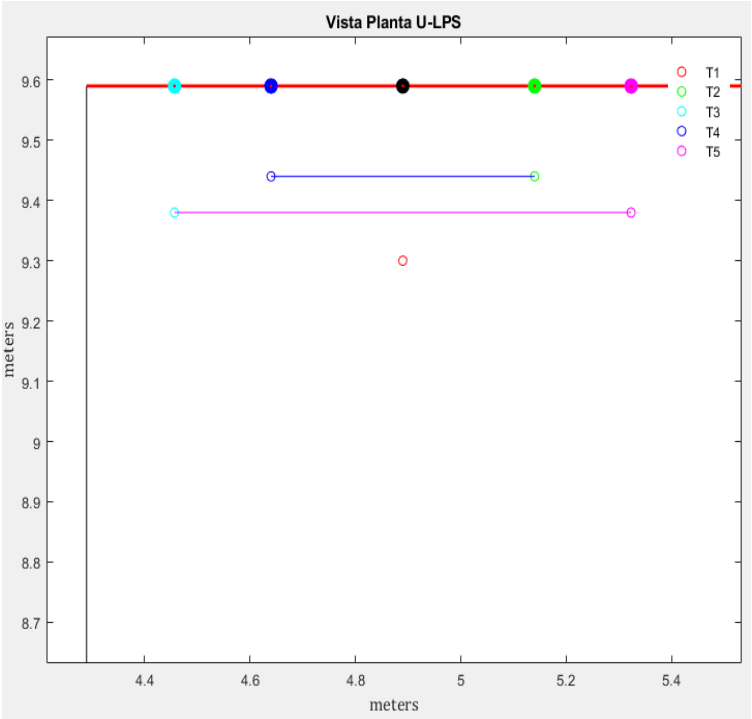


Figura 38: Obtención coordenadas de los transductores - baliza colocada en la pared (VII).

La posición de los transductores obtenida a partir de la información facilitada sería la siguiente:

```
Posición transductor 1 (rojo) = [4.8900, 9.3000, 2.0000].
Posición transductor 2 (verde) = [5.1400, 9.4400, 2.4330].
Posición transductor 3 (cyan) = [4.4570, 9.3800, 2.2500].
Posición transductor 4 (azul) = [4.6400, 9.4400, 1.5670].
Posición transductor 5 (magenta) = [5.3230, 9.3800, 1.7500].
```

Para finalizar la verificación del algoritmo desarrollado se expone dos ejemplos más en la posición estudio, pero para diferentes ángulos respecto al suelo (180º y 225º).

- Ángulo T_2 respecto al suelo de 180º

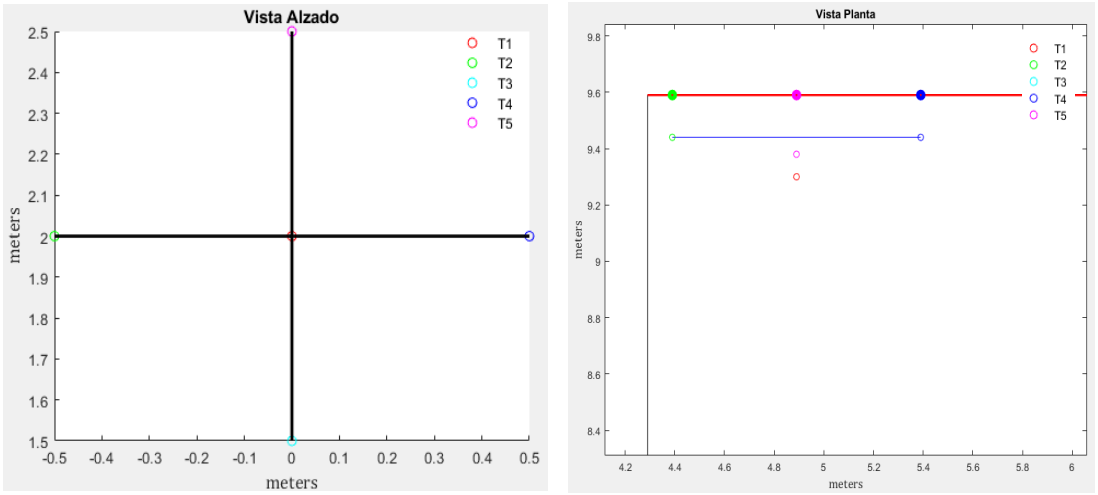


Figura 39: Representación Prueba I calibración en pared.

```
Posición transductor 1 (rojo) = [4.8900, 9.3000, 2.0000].
Posición transductor 2 (verde) = [4.3900, 9.4400, 2.0000].
Posición transductor 3 (cyan) = [4.8900, 9.3800, 1.5000].
Posición transductor 4 (azul) = [5.3900, 9.4400, 2.0000].
Posición transductor 5 (magenta) = [4.8900, 9.3800, 2.5000].
```

- Ángulo T_2 respecto al suelo de 225º

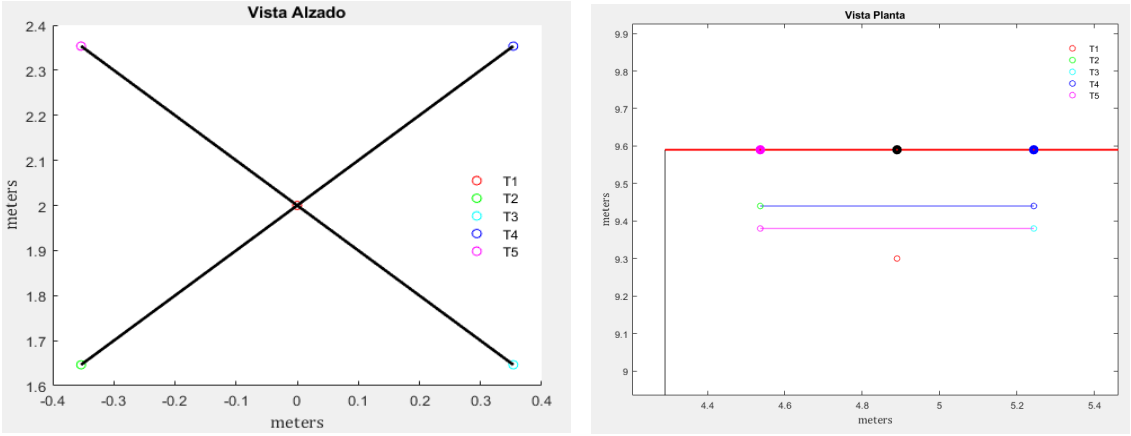


Figura 40: Representación Prueba II calibración en pared.

```
Posición transductor 1 (rojo) = [4.8900, 9.3000, 2.0000].
Posición transductor 2 (verde) = [4.5364, 9.4400, 1.6464].
Posición transductor 3 (cyan) = [5.2436, 9.3800, 1.6464].
Posición transductor 4 (azul) = [5.2436, 9.4400, 2.3536].
Posición transductor 5 (magenta) = [4.5364, 9.3800, 2.3536].
```

iii. Visualización final de los resultados obtenidos

Una vez verificados estos algoritmos desarrollados se han implementado sobre plataforma Android y se han representado sobre Google Maps.

La vista de Google Maps es en planta, por tanto, para representar de manera más visual la solución en el modo de calibración con la baliza en la pared se ha propuesto adjuntar también una vista del alzado, como se ilustraba en Matlab. Para implementar la vista del alzado en Android se ha utilizado la clase Canvas, que facilita una superficie en la que se puede dibujar, disponiendo una serie de métodos que permiten representar líneas, círculos, texto, etc.

Para finalizar con la sección se exponen los dos ejemplos con los que se ha desarrollado el algoritmo cómo quedaría su representación final sobre la aplicación Android.

Destacar que para la representación se han seleccionado las medidas reales de la baliza, por eso y aun usando el mayor zoom disponible en Google Maps, a la hora de representar los transductores, vendría bien algo más de definición en el mapa.

- Caso baliza en el techo: en Matlab figuras representación transductor central y transductores-baliza en el techo

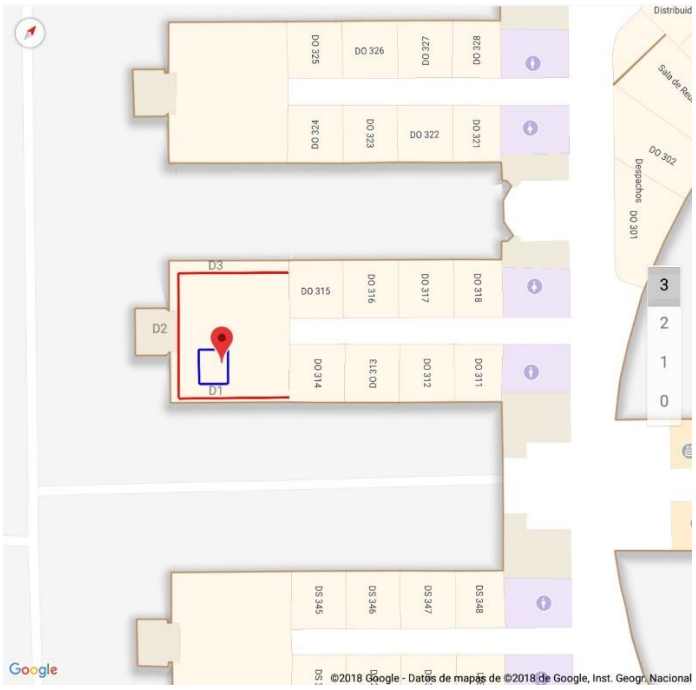


Figura 41: Representación Android transductor central- baliza en el techo.

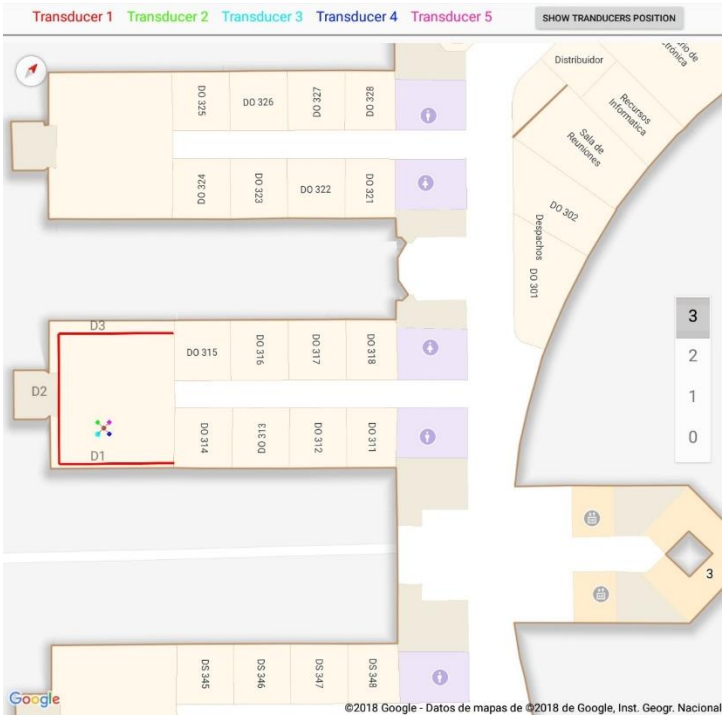


Figura 42: Representación Android transductores- baliza en el techo.

- Caso baliza en la pared: en Matlab figuras obtención coordenadas de los transductores - baliza colocada en la pared (V y VII).

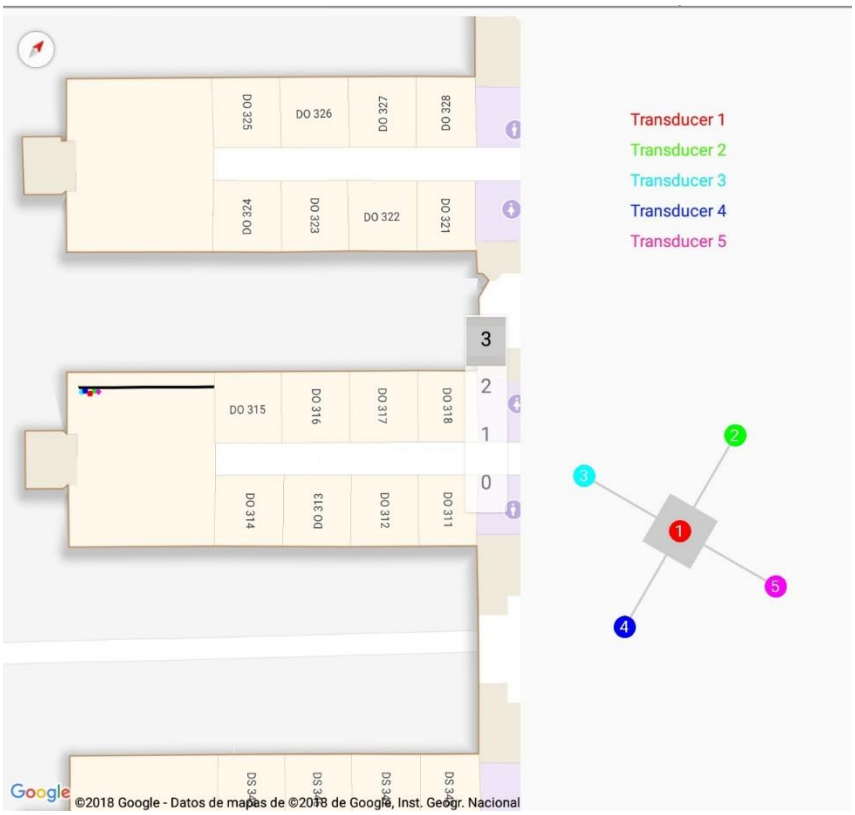


Figura 43: Representación Android coordenadas de los transductores - baliza colocada en la pared.

i. Diagrama de flujo de la aplicación

Para finalizar con el apartado del desarrollo de la aplicación se expone el diagrama de flujo de la app, en el que se entrelazan los diferentes bloques de los que se han hablado en esta sección.

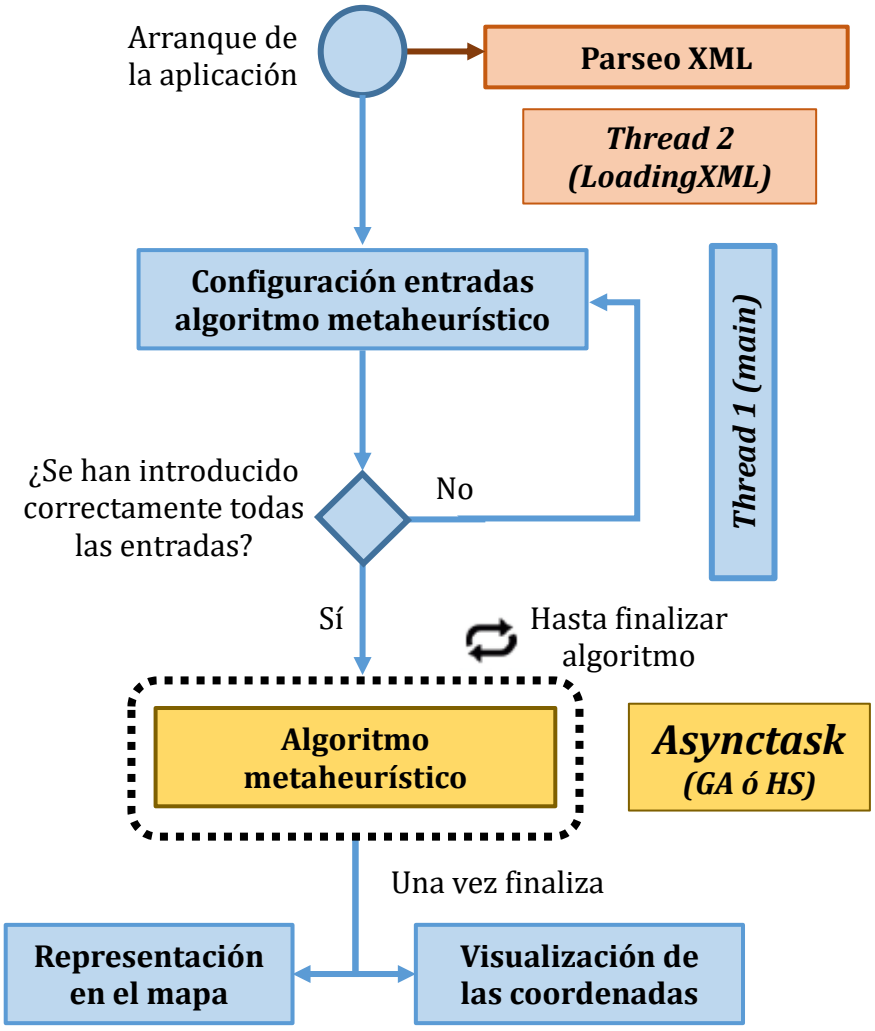


Figura 44: Diagrama de flujo de la aplicación.

En este diagrama se diferencia entre las tareas realizadas por los tres hilos principales del programa. Dichos hilos son los siguientes:

- Hilo *main*, representado en azul, hilo que más recursos dispone, encargado de atender los eventos de los distintos componentes. Este hilo ejecuta los métodos *onCreate*(), etc, es decir controla todas las actividades y servicios. Cuando la aplicación ha de realizar trabajo intensivo como respuesta a una interacción de usuario, hay que tener cuidado porque es posible que la

aplicación no responda de forma adecuada, ya que al realizar una tarea desde el hilo principal este queda bloqueado a la espera de que termine el proceso, sin poder utilizar el resto de las funcionalidades de la aplicación. Aunque pudiera darse el caso de que el proceso sea tan rápido que no dé la sensación al usuario de que la aplicación se ha bloqueado, para asegurarnos se han creado una serie de hilos que realicen ese tipo de trabajos. Además, como se expuso en apartados anteriores, si el hilo principal es bloqueado más de 5 segundos, el sistema mostrará un cuadro de dialogo al usuario "La aplicación no responde" para que el usuario decida si quieres esperar o detener la aplicación, algo que se debe evitar de todos modos en el desarrollo de la aplicación.

- Hilo *LoadingXML* (clase Thread), representado en marrón, se encarga del parseo de la información del mapa en formato vectorial. Este hilo se ejecuta en la pantalla de inicio de la aplicación, y se ha programado de forma que el hilo principal espere y no pase a otra pantalla hasta que el proceso no haya finalizado, para asegurarnos que el proceso se ha ejecutado correctamente, y así poder utilizar a posteriori los valores parseados sin problema.
- Hilo que ejecuta el algoritmo metaheurístico seleccionado (clase AsyncTask), representado en dorado, se encarga de ejecutar todas las posibles combinaciones determinadas hasta que converge en una solución óptima. Este hilo ejecuta en segundo plano (*background*) la parte iterativa del algoritmo, mientras que en el *onPostExecute* realiza la visualización de los resultados tanto en el mapa como numéricamente. Por tanto, todas las tareas se ejecutan en un sólo hilo, para evitar errores derivados de ejecuciones en paralelo (principal ventaja de programarlo mediante un AsyncTask).

  Cuando el proceso es secuencial es de vital importancia realizarlo en un único hilo, ya que en android no se tiene la capacidad de administrar los recursos, es el propio sistema el que los otorga, por lo que hacerlo así es la única forma de asegurar que una vez que Android dé los recursos, se ejecuta de seguido.

  Haciéndolo de forma segmentada, te arriesgas a que si a la hora de administrar recursos al acabar el primer hilo y el sistema no le da paso al siguiente y se lo da a cualquier otro proceso del teléfono (como una llamada entrante, por ejemplo), retrasaría la ejecución del algoritmo.

5. Pruebas experimentales finales

En esta sección se van a mostrar varias pruebas realizadas con la aplicación desarrollada para observar los resultados obtenidos y comprobar la fiabilidad del algoritmo implementado.

Se introducirán en la aplicación una serie de entradas, de las cuales se sabe el resultado final debido a pruebas experimentales obtenidas anteriormente, para poder verificar los resultados obtenidos. Al verificar las pruebas con una serie de valores obtenidos experimentalmente, habrá un *ground truth* intrínseco en los resultados.

Se ejecutará el algoritmo en tiempo real en el dispositivo portable (Samsung Galaxy Tab S3), proporcionando una solución en un tiempo mayor o menor en función de la bondad de las entradas.

a. Descripción del entorno de pruebas y resultados

Para mostrar la robustez del algoritmo se han probado varios casos en habitáculos completamente diferentes del edificio politécnico de la Universidad de Alcalá dónde ya hay balizas instaladas y por tanto ya hay una calibración realizada en las mismas, por lo que se conoce su posición, es decir un dato con el cual comparar los resultados obtenidos con el algoritmo desarrollado.

Estas pruebas se basan en la obtención de la posición central de la baliza mediante el algoritmo metaheurístico, ya que es el cuello de botella de la aplicación. La obtención de la posición del resto de transductores bien en el techo o en la pared se basa en algoritmos que toman como punto de partida la posición del transductor central.

A continuación, se procede a explicar las dos pruebas realizadas para observar la versatilidad del algoritmo, una en la cual todas las medidas a las paredes correspondían a paredes rectas y otra en la cual también había medidas correspondientes a paredes curvas.

- **Calibración con un entorno de paredes rectas.**

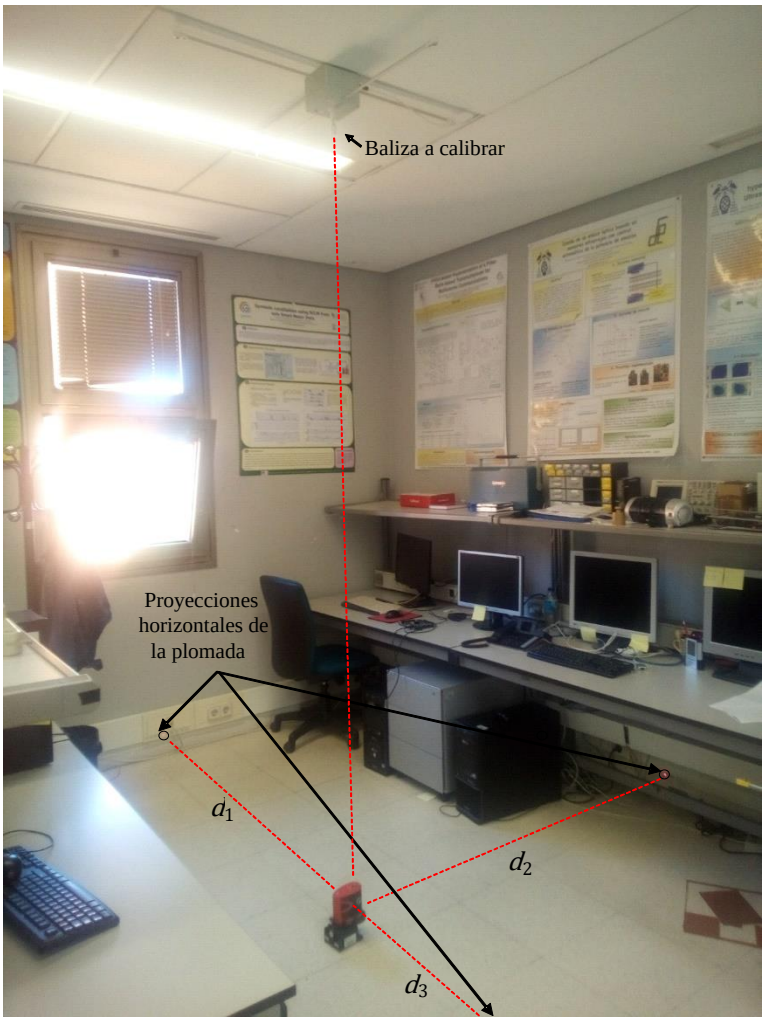


Figura 45: Posición real de la baliza, prueba I.

En esta figura se observa la plomada láser situada en el suelo en la proyección vertical del transductor central. Esta plomada proporciona tres proyecciones horizontales de las cuales se obtendrá su medida mediante un medidor láser, siendo los valores obtenidos una de las entradas al algoritmo metaheurístico.

A continuación, se muestra la pantalla principal del programa tras la calibración realizada, como queda la representación de la posición obtenida en Google Maps y la comparativa entre los valores obtenidos con la aplicación y los valores que se obtuvieron en la calibración al instalar la baliza.

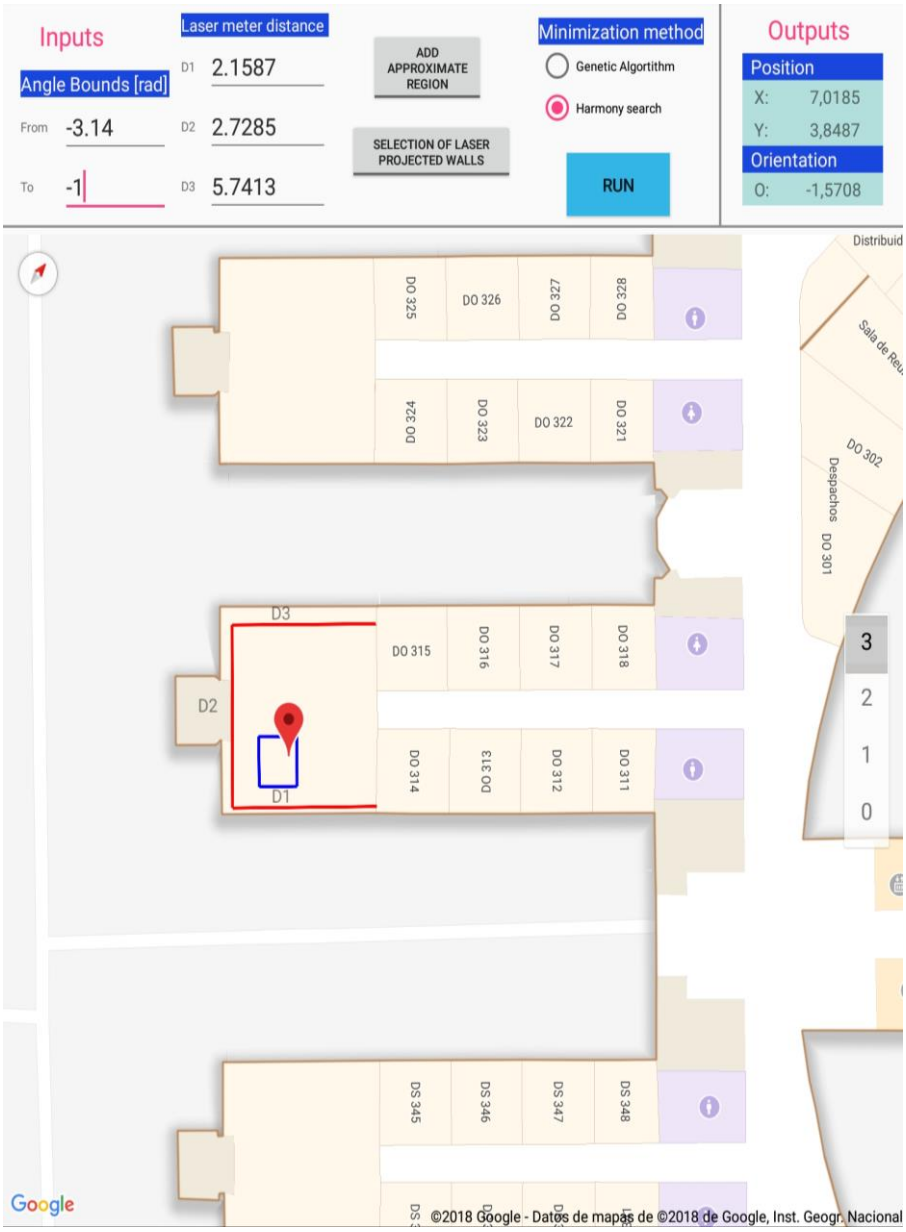


Figura 46: Posición estimada de la baliza, prueba I.

	X [m]	Y [m]	θ [rad]
Valores de referencia	7.0185	3.8487	-1.5708
Valores obtenidos en la app con GA	7.0178	3.8488	-1.5658
Valores obtenidos en la app con HS	7.0184	3.8487	-1.5708

Tabla 5: Comparativa de la prueba I.

Diferencia GA (%): 0,0258

Diferencia HS (%): 7,98e-5

Destacar que los valores de referencia son los valores obtenidos a la hora de instalar la baliza, con una calibración manual. Es decir, se mide desde la posición del transductor a calibrar la distancia a puntos conocidos en el mapa, calculando a partir de esas distancias de manera geométrica la posición del transductor.

Para que la comparativa entre los algoritmos metaheurísticos sea justa, se ha tenido que ajustar de manera experimental el número de iteraciones (estudio de las posibles soluciones) para que sea el mismo o similar en ambos algoritmos, ya que como se comentó en los capítulos anteriores, la librería de Java utilizada para el algoritmo genético es capaz de despreciar posibles soluciones de una población sin tener que estudiarlas.

Se puede observar que el algoritmo de búsqueda armónica realiza las estimaciones dando resultados más parecidos a los de referencia que el algoritmo genético, cumpliendo con lo descrito en la bibliografía.

Por otro lado, remarcar que, aunque se ha calculado la diferencia entre los resultados obtenidos con los algoritmos metaheurísticos y los valores de referencia, este dato no es el error de la posición obtenida con respecto a la realidad, sino la diferencia que se obtiene al realizar la estimación de la posición con los algoritmos metaheurísticos.

Esa diferencia (media entre la diferencia entre el valor calculado y el de referencia en todos los elementos de la solución) es bastante pequeña, siendo el error que se comete un valor mayor, ya que contemplaría los errores debido a las tolerancias de los aparatos utilizados (plomada y medidor) y el error propio que comete el usuario al utilizar los aparatos para realizar las medidas.

La tolerancia de la plomada y del medidor láser es de ±1.5 mm y ±2 mm respectivamente, por lo que el *ground truth* aproximado del sistema es del orden de mm, algo totalmente asumible en un sistema de posicionamiento en interiores, ya que la posición de los transductores es necesaria para luego en función de la posición del usuario y la cobertura que recibe de los transductores hacer el posicionamiento relativo.

- **Calibración con un entorno de paredes rectas y curvas.**

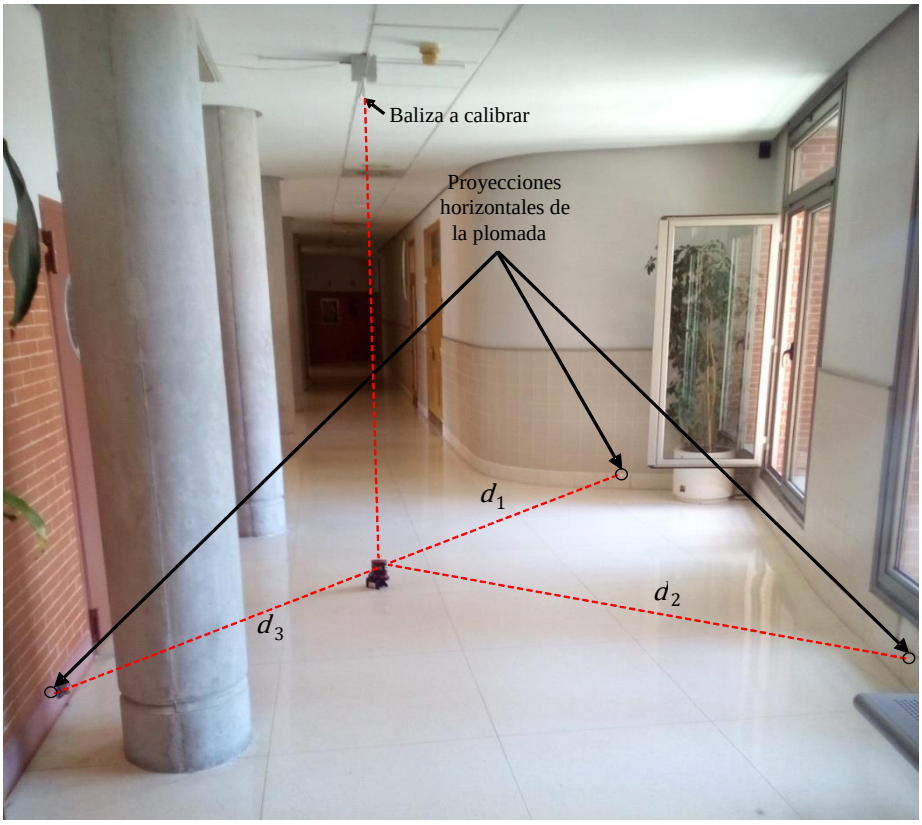


Figura 47: Posición real de la baliza, prueba II.

A continuación, se muestra la representación de la posición relativa en Google Maps y la comparativa entre los valores obtenidos con la aplicación y los reales.

	X [m]	Y [m]	θ [rad]
Valores de referencia	7.0185	3.8487	-1.5708
Valores obtenidos en la app con GA	7.0178	3.8488	-1.5658
Valores obtenidos en la app con HS	7.0184	3.8487	-1.5708

Tabla 6: Comparativa de la prueba II.

Diferencia GA (%): 0,645

Diferencia HS (%): 0.0311

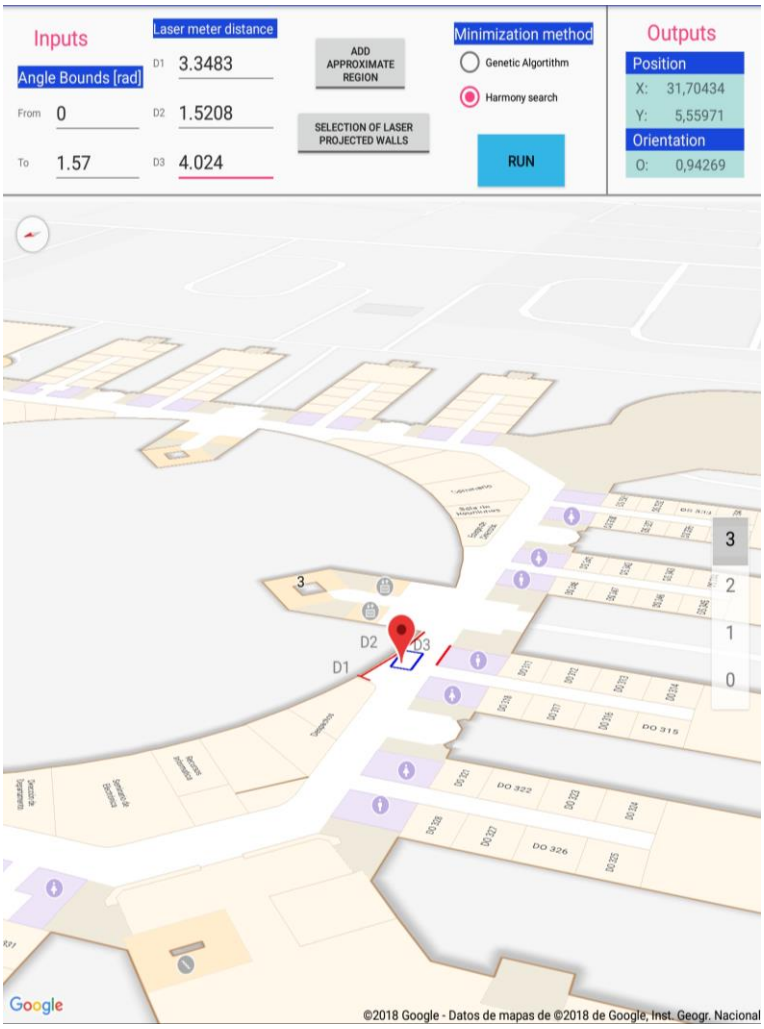


Figura 48: Posición estimada de la baliza, prueba II.

En comparación con la otra prueba se puede observar como las diferencias entre lo obtenido y la referencia son mayores para el mismo número de iteraciones en ambos casos. Este aumento radica principalmente en que, al haber paredes curvas, el caso es de más compleja resolución.

Como conclusión, expresar que aun trabajando en entornos desfavorables el algoritmo desarrollado presenta un resultado totalmente aceptable, radicando el tiempo de ejecución principalmente en las entradas del sistema, ya que se ha programado la aplicación de tal forma que independientemente de la bondad de las entradas introducidas por el usuario, el algoritmo converja a una solución adecuada. Los algoritmos desarrollados se han desarrollado de la forma lo más eficiente posible, para que en cualquiera de los casos el procesamiento pueda ejecutarse en el dispositivo portable en tiempo real.

6. Conclusiones y trabajos futuros

Este trabajo fin de máster tenía como objetivo principal desarrollar una aplicación que fuera capaz de estimar automáticamente la posición de balizas de posicionamiento local ultrasónico, para facilitar la implementación de estos sistemas.

Para llegar a este objetivo final se han tenido que ir alcanzando una serie de hitos. Por un lado, se ha tenido que programar la algoritmia necesaria para poder trabajar de manera simultánea con la información del mapa en formato vectorial y la información del mapa de Google. Se han desarrollado funciones e implementado clases y métodos ya existentes sobre plataforma Android para ser capaces de manipular datos de un fichero en formato vectorial, convertir esos datos al formato en el que trabaja Google maps y fusionarlos de tal forma que la información en formato vectorial tenga una representación fidedigna sobre los mapas de Google, para que ambos sistemas tengan las mismas referencias en el mapa.

Por otro lado, se ha desarrollado la algoritmia correspondiente para conseguir la detección de la pared más cercana a partir de un punto introducido por el usuario, la obtención de la posición central de la baliza mediante varios algoritmos metaheurísticos y la posición 3D de los transductores.

Por último, se ha buscado la forma de crear un interfaz que sea sencillo y cómodo desde el punto de vista del usuario. Para ello se ha hecho especial hincapié a la hora de visualizar los resultados, desarrollando funciones capaces de representar métodos que no tiene Maps habilitados (cómo escribir sobre el Mapa de Google o dibujar curvas) o utilizando la clase Canvas para complementar la vista en planta que proporciona Google. Este logro ha sido de vital importancia ya que, aunque la mayor parte de este TFM reside en la algoritmia, disponer de una interfaz gráfica vistosa y útil resulta un elemento diferenciador de cara a la exposición del producto.

En definitiva, se ha desarrollado una aplicación llamada '*Map Matching Beacons Calibration*' la cual es capaz de obtener y visualizar la posición de los transductores de balizas colocadas tanto en la pared como en el techo, cumpliendo el hito fijado dentro del proyecto TARSIUS, en el que se encuentra enmarcado el presente TFM.

7. Planos

En esta sección se van a mostrar algunos extractos de las partes más importantes del código desarrollado sobre plataforma Android.

- **Algoritmo genético**

Se adjunta el código desarrollado en el presente TFM para realizar el algoritmo genético. Destacar que para su implementación se ha utilizado la librería JGAP.

```
Configuration.reset();

Configuration conf = new
DefaultConfiguration();conf.setPreservFittestIndividual(true);

FitnessFunction myFunc = new
fitnessFunction_GA(DWalls,Link_data.Array_Wall);
conf.setFitnessFunction(myFunc);

Gene[] sampleGenes = new Gene[3];
sampleGenes[0] = new DoubleGene(conf, UmbralMin_Region[0],
UmbralMax_Region[0]);
sampleGenes[1] = new DoubleGene(conf, UmbralMin_Region[1],
UmbralMax_Region[1]);
sampleGenes[2] = new DoubleGene(conf, AngleBound[0], AngleBound[1]);

IChromosome sampleChromosome = new Chromosome(conf, sampleGenes);
conf.setSampleChromosome(sampleChromosome);

conf.setPopulationSize(30);

Genotype Poblacion = Genotype.randomInitialGenotype(conf);

double num_combinaciones_brute_force=floor((UmbralMax_Region[0]-
UmbralMin_Region[0])*100)*floor((UmbralMax_Region[1]-
UmbralMin_Region[1])*100)*floor(abs(AngleBound[1]-
AngleBound[0])*100);
double porcentaje_combinaciones=0.05;
int MAX_EVOLUCIONES_PERMITIDAS = (int)
floor(porcentaje_combinaciones*num_combinaciones_brute_force)/27;
/*Para que la comparación con HS sea justa. GA genera como mucho 30
sol por iteración mientras HS solo 1*/

for (int i = 0; i < MAX_EVOLUCIONES_PERMITIDAS; i++) {
        Poblacion.evolve();
}

BestSolution=Poblacion.getFittestChromosome();

//Aquí ya se tendrían los valores de X,Y, orientación (O),
almacenados en las variables: X; Y; O
X=(double) BestSolution.getGene(0).getAllele();
Y=(double) BestSolution.getGene(1).getAllele();
O=(double) BestSolution.getGene(2).getAllele();
```

- **Algoritmo de búsqueda armónica**

El código desarrollado para la realización del algoritmo ha sido el siguiente:

```
/** Random initialization of Harmony Memory (HM)*/

for (int i=0;i<HMS;i++){
    for (int j=0;j<NVAR;j++){
        matrix[i][j]=PVB[j][0]+(PVB[j][1]-PVB[j][0])*random();
        PossibleSolution[j]=matrix[i][j];
    }
    funcionAptitud = new

fitnessFunction_HS(PossibleSolution,DWalls,Link_data.Array_Wall);
    matrix[i][3]=funcionAptitud.calculate();
}

Arrays.sort(matrix, new Comparator<double[]>() {
    @Override
    public int compare(double[] entry1, double[] entry2) {
        int Sort_index=3;//columna por la cual se quiere ordenar
        double value1 = entry1[Sort_index];
        double value2 = entry2[Sort_index];
        return Double.compare(value1,value2);
    }
});

/** Harmony Search*/
for(int it=0;it<NI;it++){
    PAR=PARmin+(it-1)*(PARmax-PARmin)/(NI-1);
    bwRange=bwmax*exp((it-1)*coefbw);

    for(int j=0;j<NVAR;j++){ //se genera variable del armónico
        if(random()>=HMCR){
            PossibleSolution[j]=PVB[j][0]+(PVB[j][1]
               -PVB[j][0])*random();
        }else { //si no se genera aleatorio, se tiene en cuenta HM
            int pos_random=(int)floor(HMS*random());
            PossibleSolution[j]=matrix[pos_random][j];

             //Se realiza ajuste de tono
            if(random()<=PAR){
                double Sol_adjusted;

                if(random()<=0.5){
                  Sol_adjusted=PossibleSolution[j]+bwRange*random();
                  if(PVB[j][1]>Sol_adjusted){
                      PossibleSolution[j]=Sol_adjusted;
                  }
                }else{
                  Sol_adjusted=PossibleSolution[j]-bwRange*random();
                  if(PVB[j][0]<Sol_adjusted){
                      PossibleSolution[j]=Sol_adjusted;
                  }
                }
            }
        }
    }
```

```
    /**Aquí ya tendríamos el nuevo armónico generado. Por tanto lo
evaluamos.*/
    funcionAptitud = new
fitnessFunction_HS(PossibleSolution,DWalls,Link_data.Array_Wall);
    double error_PossibleSolution=funcionAptitud.calculate();

    if(error_PossibleSolution<matrix[29][3]){
        matrix[29][0]=PossibleSolution[0];
        matrix[29][1]=PossibleSolution[1];
        matrix[29][2]=PossibleSolution[2];
        matrix[29][3]=error_PossibleSolution;


        Arrays.sort(matrix, new Comparator<double[]>() {
            @Override
            public int compare(double[] entry1, double[] entry2) {
                int Sort_index=3;
                double value1 = entry1[Sort_index];
                double value2 = entry2[Sort_index];
                return Double.compare(value1,value2);
            }
        });
    }

}

/**Aquí ya se habría realizado el número de iteraciones
determinadas.*/

/** La mejor solución del algoritmo se corresponderá a la primera
fila de la memoria armónica (valor con menor error matrix[0][3])*/
X= matrix[0][0];
Y= matrix[0][1];
O= matrix[0][2];
```

Destacar que tanto el código de los algoritmos genéticos como el de búsqueda armónica se introducirá dentro de la sentencia *try* de un bloque *try-catch*, para que en caso de que surgiera alguna excepción a lo largo del transcurso del algoritmo, se procese directamente lo programado en el *catch*.

Esto se realiza para evitar que en caso de que haya algún problema, no se cierre automáticamente la aplicación, que es lo que ocurriría en caso de no poner este bloque.

En la clausula *catch*, se le mandará al usuario un mensaje por pantalla (*Alert Dialog*) para que corra de nuevo el algoritmo, ya que este no se ha podido ejecutar correctamente.

- **Función aptitud (*Fitness function*)**

Esta parte se centrará en las clases creada para evaluar cada posible combinación generada en los algoritmos metaheurísticos.

Se han tenido que generar dos clases diferentes para implementar esta función, aunque en cuanto al procesamiento de las posibles soluciones, se evalúan de la misma forma en ambas clases, debido a las peculiaridades que hay que realizar al utilizar la librería externa de los algoritmos genéticos: tiene que ser una clase extendida a *FitnessFunction* y ha de devolver el valor aptitud (*fitness*) en el método *evaluate()*.

En cuanto a la función aptitud generada en el algoritmo de búsqueda armónica, se ha creado una clase llamada fitnessFunction_HS, que contiene en su interior un constructor, método con el mismo nombre de la clase en el cual se inicializan las variables de la clase con las introducidas en la misma; y un método, llamado calculate, en el cual se realizan las operaciones correspondientes a la obtención de la bondad de la solución, devolviendo el error (inverso del valor aptitud) entre la solución estudiada y los datos introducidos por el usuario.

El código desarrollado es el siguiente:

```
public class fitnessFunction_HS{

    public double[] LaserMeterDistance;
    public Float[][] Pared;
    public double[] chromosome;

    public fitnessFunction_HS(double[] Solucion, double []
DistanceWalls, Float[][] Paredes) {

        Pared = null;
        LaserMeterDistance=null;
        chromosome=null;

        LaserMeterDistance=new double[DistanceWalls.length];
        LaserMeterDistance= DistanceWalls;

        Pared=new Float[Paredes.length][Paredes[0].length];
        Pared=Paredes;

        chromosome=new double[Solucion.length];
        chromosome=Solucion;
    }
```

```
public double calculate() {

    /**El método calculate es el método que se debe sobrecargar para
que devuelva el valor de aptitud asociado al cromosoma que se recibe
por parámetro.
 */

//Combinación de estudio: chromosome[0] (X), chromosome[1] (Y),
chromosome[2] (Orientacion);

    /**Definimos los puntos de las rectas de las proyecciones
        de la plomada*/
    int max_range=10;
    double[][] points_laser = new double[][]{
            {chromosome[0]+(max_range*cos(chromosome[2])),
               chromosome[1]+(max_range*sin(chromosome[2]))},
            {chromosome[0]+(max_range*cos(chromosome[2]-Math.PI/2)),
               chromosome[1]+(max_range*sin(chromosome[2]-
               Math.PI/2))},
            {chromosome[0]-(max_range*cos(chromosome[2])),
               chromosome[1]-(max_range*sin(chromosome[2]))},};

    double [] dist=new double[3]; //Array de las distancias
estimadas para cada cromosoma

    for(int i=0;i<3;i++) //recorremos las paredes
    {
        if(Pared[i][0]==0){//pared i es una pared recta

            Float x1, y1, x2, y2, a, b, e;
            x1 = Pared[i][1];
            y1 = Pared[i][2];
            x2 = Pared[i][3];
            y2 = Pared[i][4];
            a = (y2 - y1);
            b = -(x2 - x1);
            e = x1 * (y2 - y1) - y1 * (x2 - x1);

            Float x3, y3, x4, y4, c, d, f;
            x3 = (float) points_laser[i][0];
            y3 = (float) points_laser[i][1];
            x4 = (float) chromosome[0];
            y4 = (float) chromosome[1];
            c = (y4 - y3);
            d = -(x4 - x3);
            f = x3 * (y4 - y3) - y3 * (x4 - x3);

            if ((a * d) - (c * b) != 0) {
                //no son paralelas, se puede aplicar Cramer
                //calculamos punto de corte con las paredes->
               (xInterseccion,yInterseccion)
                double xInt =((e * d) - (b * f)) / ((a * d)-(c* b));
                double yInt = ((a * f) - (e * c)) / ((a * d)-(c*b));

                dist[i] = sqrt(pow(xInt-x4, 2) + pow(yInt - y4, 2));
            }
        }
```

```
        else if (Pared[i][0]==1){ //pared i es una pared curva

             //Ecuacion de la circunferencia: x^2+(-2*a)*x+y^2
             +(-2*b)*y=(r^2-a^2-b^2)
             Float A, B, C;
             A=-2*Pared[i][5];B=-2*Pared[i][6];
             C=(float)(+pow(Pared[i][5],2)+pow(Pared[i][6], 2)
             -pow(Pared[i][7], 2));

             Double[] xInt_arc=new Double[2];
             Double[] yInt_arc=new Double[2];

             Float x3, y3, x4, y4;
                x3 = (float) points_laser[i][0];
                y3 = (float) points_laser[i][1];
                x4 = (float) chromosome[0];
                y4 = (float) chromosome[1];

            Float m, n,a,b,c;
            int nPuntosInterseccion=0;

         if((x3-x4)<=3e-2&&(x3-x4)>=-3e-2){
                //semirecta vertical; (x4-x3)~=0; x=0*y+n
                m=(x4-x3)/(y4-y3); //0
                n=x3-m*y3;

                /*ecuación de la forma-> a*y^2+b*y+c=0;
                a,b,c variables de la ecuacion de 2°grado*/
                a=1f;
                b=B;
                c=(float)pow(n,2)+n*A+C;

                //Resolución ecuacion de segundo grado
                if((pow(b,2)-4*a*c)<0){
                        //No hay solucion (nPuntosInterseccion=0)
                        nPuntosInterseccion=0;
                }
                else if((pow(b,2)-4*a*c)==0){
                        //Hay solución doble (nPuntosInterseccion=1)
                        nPuntosInterseccion=1;
                        yInt_arc[0]=(double)-b/(2*a);
                        xInt_arc[0]=(double) n;
                }
                else if((pow(b,2)-4*a*c)>0){
                        //Hay soluciones diferentes
                        (nPuntosInterseccion=2)
                        nPuntosInterseccion=2;
                        yInt_arc[0]=(-b+sqrt(pow(b,2)-4*a*c))/(2*a);
                        xInt_arc[0]=(double)n;
                        yInt_arc[1]=(-b-sqrt(pow(b,2)-4*a*c))/(2*a);
                        xInt_arc[1]=(double)n;
                }
        }
```

```
        else{ //línea horizontal o con pendiente normal
                    m=(y4-y3)/(x4-x3);
                    n=y3-m*x3;

                    //ecuacion de la forma-> a*x^2+b*x+c=0
                    a=1+(float)pow(m,2);
                    b=A+2*m*n+B*m;
                    c=(float)pow(n,2)+n*B+C;

                    //Resolucion ecuacion de segundo grado
                    if((pow(b,2)-4*a*c)<0){
                        //No hay solucion (nPuntosInterseccion=0)
                        nPuntosInterseccion=0;
                    }
                    else if((pow(b,2)-4*a*c)==0){
                        //Hay solucion doble (nPuntosInterseccion=1)
                        nPuntosInterseccion=1;
                        xInt_arc[0]=(double)-b/(2*a);
                        yInt_arc[0]= m*xInt_arc[0]+n;
                    }
                    else if((pow(b,2)-4*a*c)>0){
                        /*Hay soluciones diferentes
                        (nPuntosInterseccion=2)*/
                        nPuntosInterseccion=2;
                        xInt_arc[0]=(-b+sqrt(pow(b,2)-4*a*c))/(2*a);
                        yInt_arc[0]= m*xInt_arc[0]+n;
                        xInt_arc[1]=(-b-sqrt(pow(b,2)-4*a*c))/(2*a);
                        yInt_arc[1]= m*xInt_arc[1]+n;
                    }
                }

                double dist_Int_min=1e8;

                for(int j=0;j<nPuntosInterseccion;j++){
                    double distInt = sqrt(pow(xInt_arc[j] - x4, 2)
                                    + pow(yInt_arc[j] - y4, 2));
                    if (distInt < dist_Int_min) {
                        dist_Int_min=distInt;
                        dist[i]=dist_Int_min;
                    }
                }
            }
        }

        double mean_error=((abs(dist[0]-LaserMeterDistance[0]))
        +(abs(dist[1]-LaserMeterDistance[1]))
        +(abs(dist[2]-LaserMeterDistance[2])))/3;

        return mean_error;
        /**A menor error, mejor fitness, mejor aptitud, mejor es la
        combinación estudiada (fitness=1/mean_error)*/

    }

}
```

8. Pliego de condiciones

El presente pliego de condiciones tiene por objeto definir las condiciones para las que se ha creado la aplicación, no asegurando el correcto funcionamiento de la misma si no se siguen dichas condiciones. Las condiciones para las cuales se ha originado la app se describen a continuación.

- Aplicación diseñada para smartphones Android de versión 4.0.3 o superior.
- Programación para la obtención de la posición teniendo en cuenta las dimensiones del Sistema de Posicionamiento Local Ultrasónico LOCATE-US propuesto por el grupo GEINTRA US&RF. Más en concreto se ha desarrollado para calibrar balizas de este tipo ubicadas en el techo o en la pared.
- Se precisa acceso a Internet a la hora de visualizar en Google Maps, ya que sin Internet no se cargan los mapas. Aunque se podría programar automáticamente el encendido del wifi sin el permiso del usuario, para tratar de ser lo más cuidadosos posibles con la intimidad del usuario, se ha creado una alerta que sale al arranque de la aplicación si no se tiene el Internet activado, que pide permiso al individuo para que autorice la activación del WiFi. Dicho diálogo de alerta es como el mostrado en la figura expuesta a continuación:

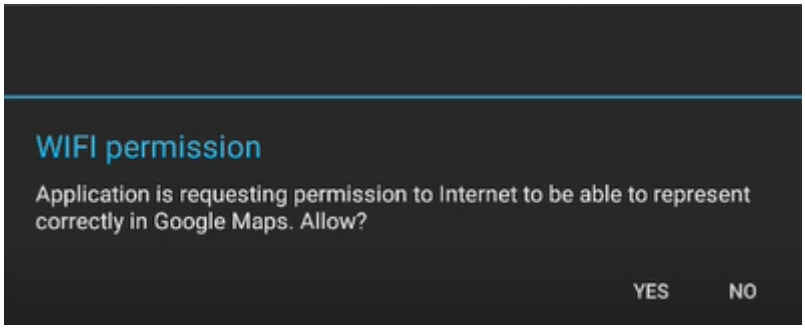


Figura 49: Permiso a acceder al Wifi del dispositivo portable.

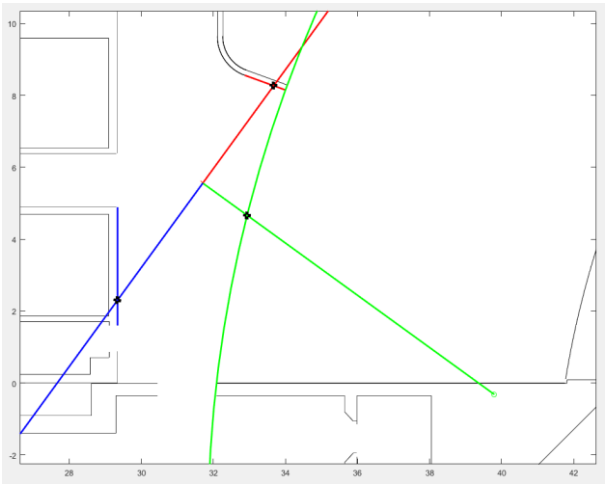
- La aplicación se ha creado utilizando una API *key* de Google Maps para poder hacer uso de los servicios de Google. Aunque a priori este servicio es gratuito, si la aplicación excediera de un número de consultas por segundo, QPS, que va entre las 50 y las 500, en función de la API de Google utilizada (*direction, geocoding, Dynamic maps, Street view, roads,* etc.) se tendría que pagar por hacer uso del servicio. Si no se tuviera vinculada ninguna forma de pago,

asociada a la credencial de la aplicación, la API cesaría sus funciones cuando se excedieran dichos límites.

- Aplicación diseñada para el Edificio Politécnico de la Universidad Superior. Para otro edificio harían falta los datos del mismo en formato XML, almacenarlos en el proyecto Android de la aplicación y cargarlo durante la ejecución del programa.
- Se requiere una correcta introducción de las entradas del algoritmo para un correcto funcionamiento del mismo, ya que esta aplicación se basa en un algoritmo de optimización, es decir, obtiene el resultado final a partir de una serie de entradas iniciales. Por lo que, si de entrada se introducen valores erróneos, se introducirá un error que se derivará al producto final.

  Destacar una de las entradas en concreto, la de la selección de la pared, ya que el XML del edificio de estudio detecta una pared como dos líneas (muro exterior e interior), por lo que es de vital importancia seleccionar justo la parte de muro a la cual se toman las medidas ya que, si no, se estará cometiendo un error en el resultado equivalente al espesor de la pared. Es decir, la no elección de la pared correcta deriva a un error, que hace que la solución ideal a la que debería converger el algoritmo tenga un fitness no tan bueno, lo que conlleva a que el algoritmo converja en soluciones reales peores con fitness superiores al de la solución ideal.

  A continuación, se muestra una prueba de los resultados obtenidos en la función *fitness* a partir de una posición conocida, eligiendo en primer lugar la pared correcta y luego seleccionando la otra parte de la pared.



```
mean_error =

   4.5294e-04


fitness =

   2.2078e+03

>> dist

dist =

    3.3483    1.5196    4.0241
```

Figura 50: Prueba selección pared correcta.

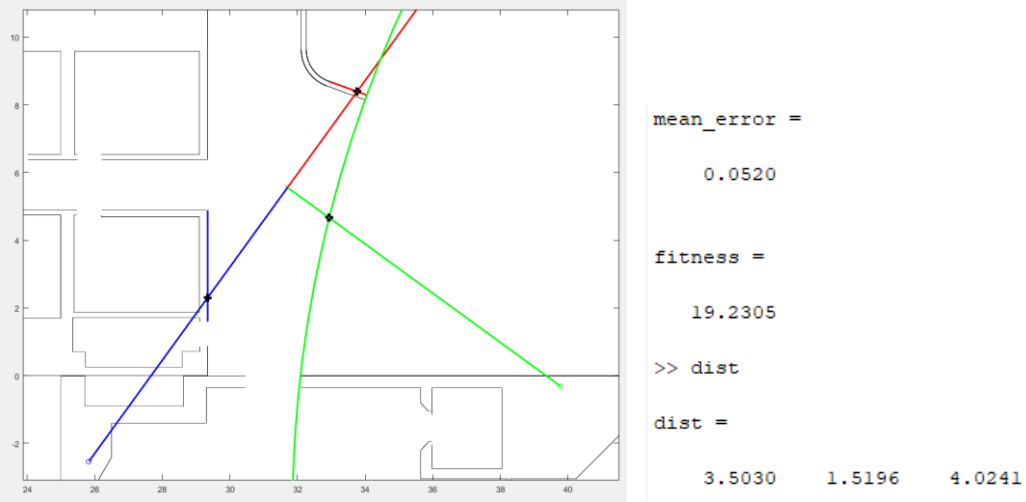


Figura 51: Prueba selección pared incorrecta.

Se observa como d1, tiene un valor que difiere al que se ha medido (3.3483m), debido a que se ha cogido la pared de estudio, pero la parte del muro más lejana, de ahí que la distancia obtenida sea la obtenida anteriormente más el espesor del muro.

9. Presupuesto

El presupuesto que se va a exponer hace referencia tanto a los costes directos debidos a las horas de trabajo como programador (el propio alumno) como a los costes del material necesario para su desarrollo. Sin embargo, no se han tenido en cuenta los tiempos de formación e investigación para el desarrollo del mismo, ni el empleado en la redacción del mismo. Dicho presupuesto del proyecto resulta sólo orientativo y no está pensado en vistas a una posible comercialización y por tanto amortización de la herramienta creada. En este caso la aplicación desarrollada, satisface una necesidad planteada por el Departamento de Electrónica de la Universidad de Alcalá de Henares en el seno del proyecto TARSIUS.

a. Coste del material utilizado

En esta sección se muestran los costes debidos a los elementos físicos utilizados en este proyecto (sin el IVA incluido en los precios), computando sólo la amortización en los meses de utilización.

Equipo	Precio (€)	Periodo amortización (años)	Usos (meses)	Coste de amortización (€)
Tablet Samsung Galaxy Tab S3	425	3	8	94.44
PC Intel i7, 3GB RAM	800	3	8	177.77
Total	**1315**	**-**	**-**	**272.21**

Tabla 7: Coste del material utilizado.

b. Costes directos de la aplicación

Para obtener los costes de programación (tanto en Android como en Matlab), se ha asumido un coste estándar como "programador" de 50 €/h. El desglose de las horas de programación se ha dividido en los grandes bloques del proyecto, englobando en cada uno de ellos desde su creación hasta su funcionamiento, y se muestra en la siguiente tabla:

Bloque del proyecto	Tiempo empleado (horas)	Coste (€)
Manejo dajos en formato XML	80	4000
Conversión formato LatLng a XML	40	2000
Fusión información de los mapas	20	1000
Detección pared más cercana	60	3000
Algoritmos metaheurísticos	300	15000
Algoritmo para obtener la posición de los transductores	200	10000
Batería de pruebas	200	10000
Diseño del interfaz	250	12500
	Total	**57500**

Tabla 8: Costes directos de la aplicación.

c. Costes indirectos de la aplicación

En este apartado se reflejan los costes indirectos, entendiendo por estos costes como los que suponen las licencias de los programas necesarios para el desarrollo del proyecto.

Licencia	Coste (€)	Periodo amortización (años)	Usos (meses)	Coste de amortización (€)
Matlab r2018b	2000	2	8	666.66
Android Studio 3.1.2	0	2	8	0
Microsoft Office 2016	450	2	8	150
Total	**2450**	-	-	**816.66**

Tabla 9: Costes indirectos de la aplicación.

d. Coste del proyecto

Concepto	Coste íntegro del proyecto (€)	Coste de amortización (€)
Coste del material utilizado	1315	272.21
Costes directos de la aplicación	57500	57500
Costes indirectos de la aplicación	2450	816.66
Total (sin IVA)	**61265**	**58588.87**
Total (con IVA del 21%)	**74130.65**	**70892.53**

Tabla 10: Coste del proyecto.

Por tanto, el importe estimado del proyecto en la duración del mismo asciende a la cantidad de:

Setenta mil ochocientos noventa y dos euros y cincuenta y tres céntimos

Guadalajara, a 13 de Junio de 2019.

Firmado: Rubén Cervigón Rey

Ingeniero Industrial.

10. Manual de usuario

a. Descripción de la aplicación

La aplicación **Map Matching Beacons Calibration** es una aplicación para smartphones de versión 4.0.3 o superior (a día de hoy compatible con prácticamente el 100% de los dispositivos) que permite la calibración de la posición de balizas, es decir, la obtención de la posición y su posterior visualización en Google Maps.

b. Descarga de la aplicación

Esta aplicación se puede instalar a través de la plataforma de desarrollo de aplicaciones Android (Android Studio) con el programa desarrollado en este TFM. Será tan sencillo como abrir el programa con Android Studio y compilar y ejecutar el programa, a través del botón ***Run* 'app'**, como se aprecia en la imagen adjuntada a continuación.

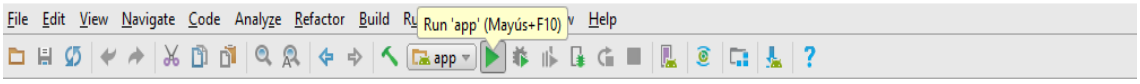


Figura 52: Instalación de la aplicación (I)

Una vez seleccionada esta opción, aparecerá una pantalla para elegir en qué dispositivo de los conectados al ordenador vía USB cargar el programa, como se puede observar en la siguiente imagen:

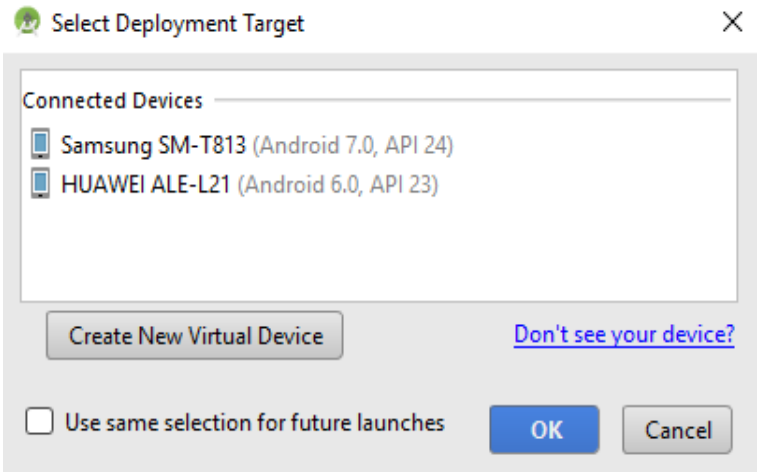


Figura 53: Instalación de la aplicación (II).

Se selecciona el dispositivo deseado, se pulsa **OK** y automáticamente se instala la aplicación en dicho dispositivo.

En la siguiente imagen se ilustra una captura del menú de aplicaciones del dispositivo en el que se ha desarrollado la aplicación tras descargar la aplicación.

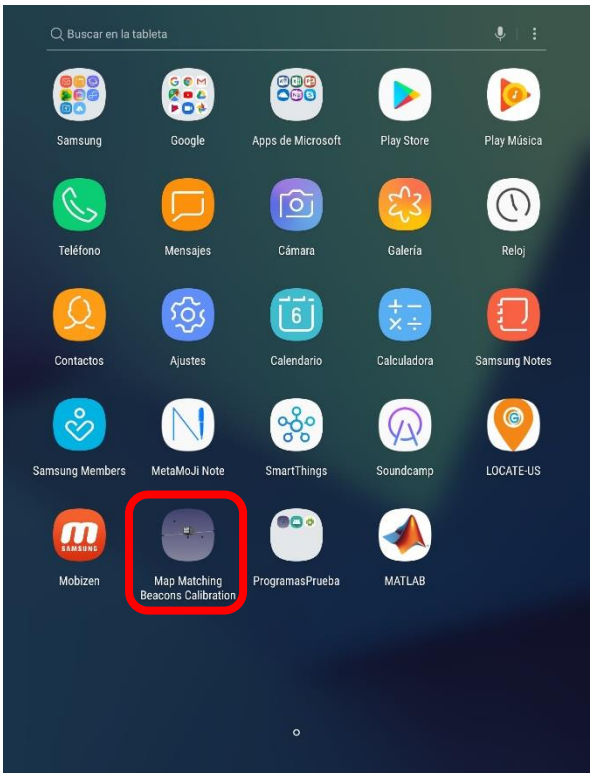


Figura 54: Instalación de la aplicación (III).

c. Navegación por la aplicación

i. Interfaz de arranque

Figura 55: Interfaz de arranque (I).

Se ha programado una pantalla de arranque a la aplicación, de duración 2 segundos, que sirva como presentación de la aplicación que se va a abrir.

Al finalizar este periodo la aplicación solicita que le digas en qué posición se encuentra la baliza a calibrar, para dirigirte a su interfaz específico.

Figura 56: Interfaz de arranque (II).

ii. Pantalla principal

En esta pantalla el usuario podrá realizar la calibración de la baliza una vez introduzca las entradas pertinentes, bien en el modo '*Calibration of a beacon placed on the ceiling*' o '*Calibration of a beacon placed on the wall*'.

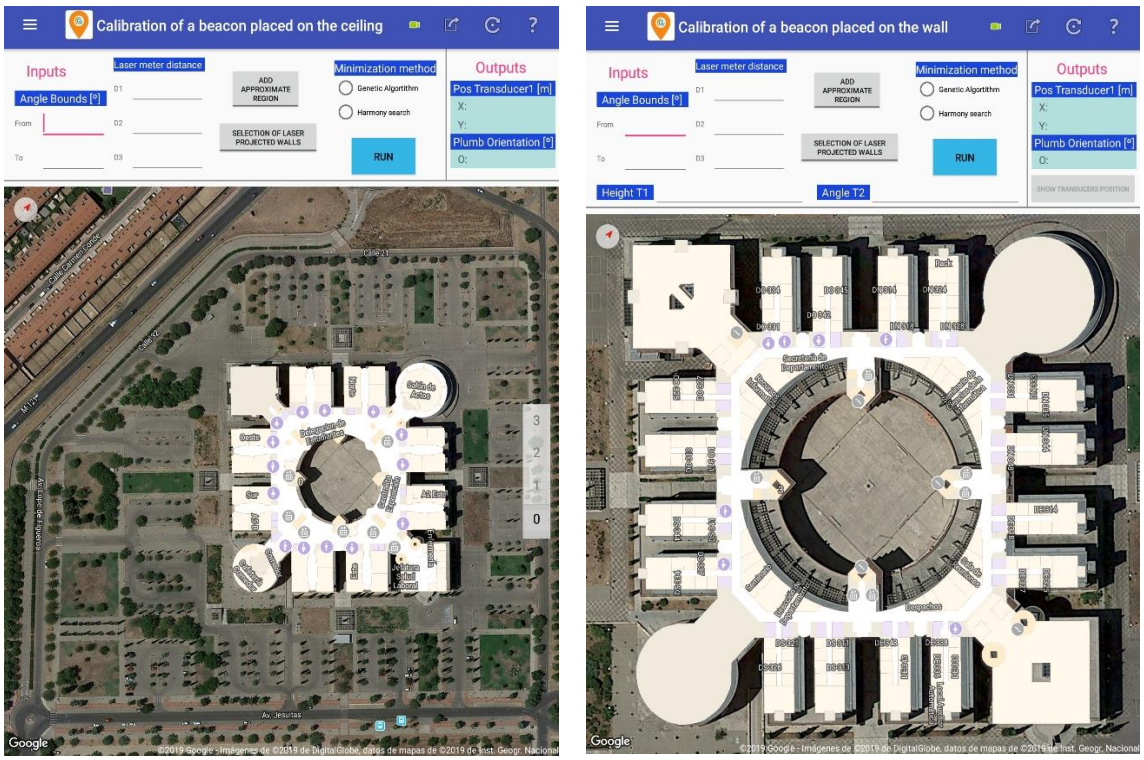


Figura 57: Pantalla principal antes de comenzar una prueba. Calibración de la baliza en el techo(a) o en la pared (b).

Una vez se han introducido todas las entradas y se presiona el botón RUN, se ejecutará el algoritmo seleccionado en el caso de que no hubiera ningún dato erróneo, en cuyo caso la aplicación enviará un mensaje al usuario diciéndole donde está el problema.

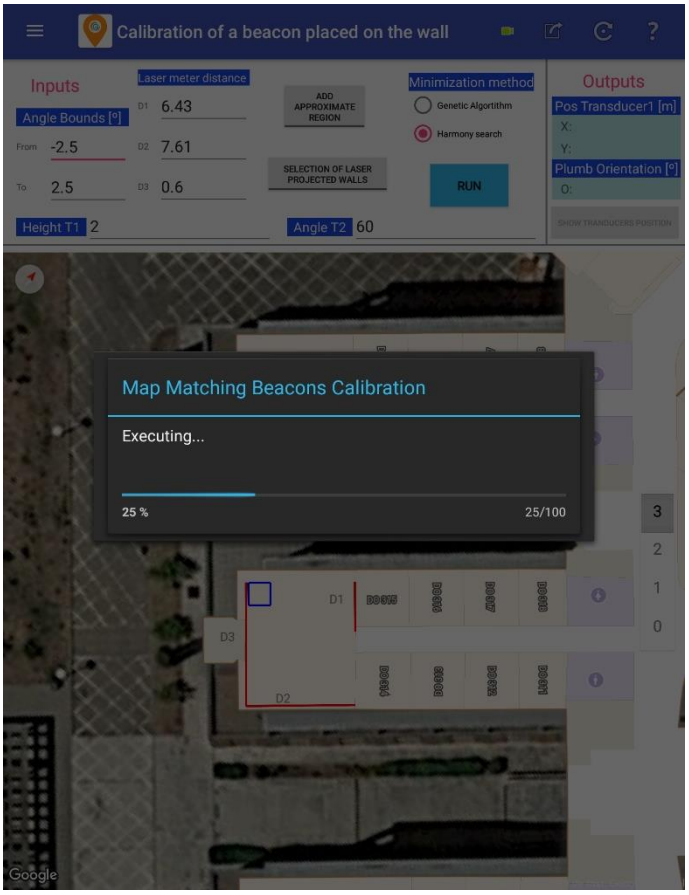


Figura 58: Pantalla principal durante la ejecución.

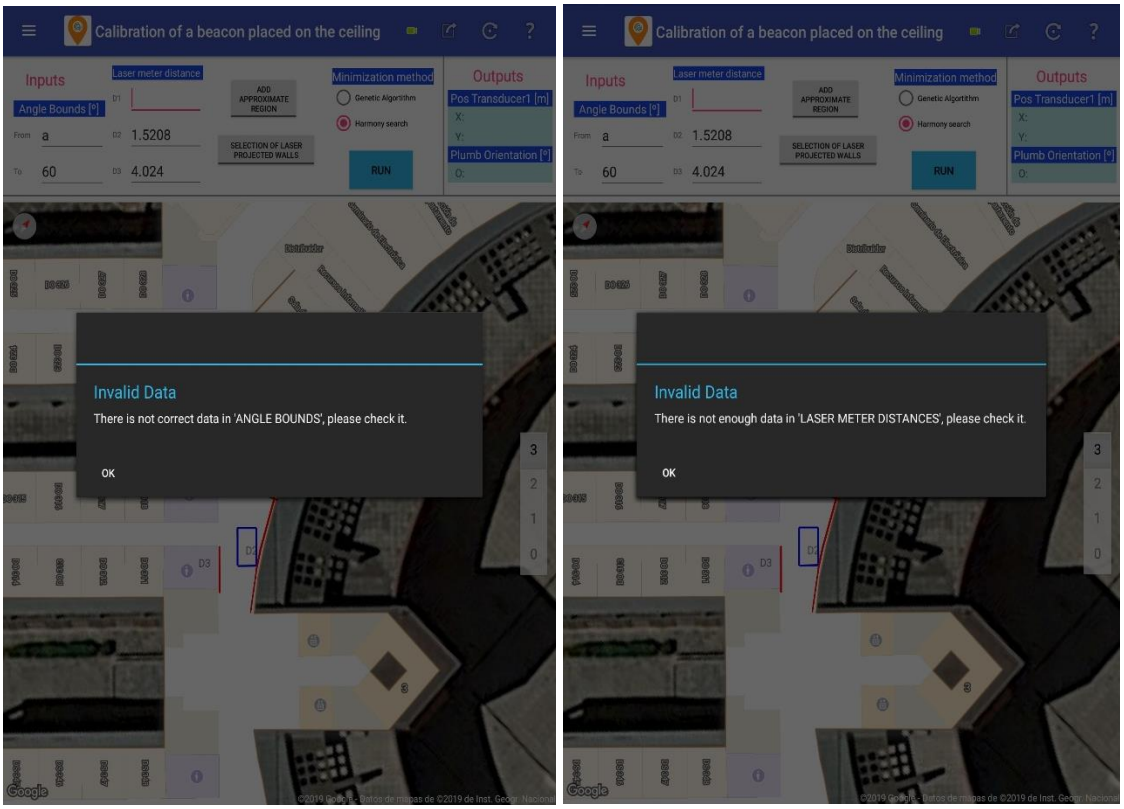


Figura 59: Pantalla principal ante errores en las entradas. (a) Dato erróneo. (b) Casilla vacía.

Al acabar la ejecución del algoritmo se visualizará de manera gráfica y numérica (a través del botón ***SHOW TRANSDUCER POSITION***) la posición de los transductores de la baliza.

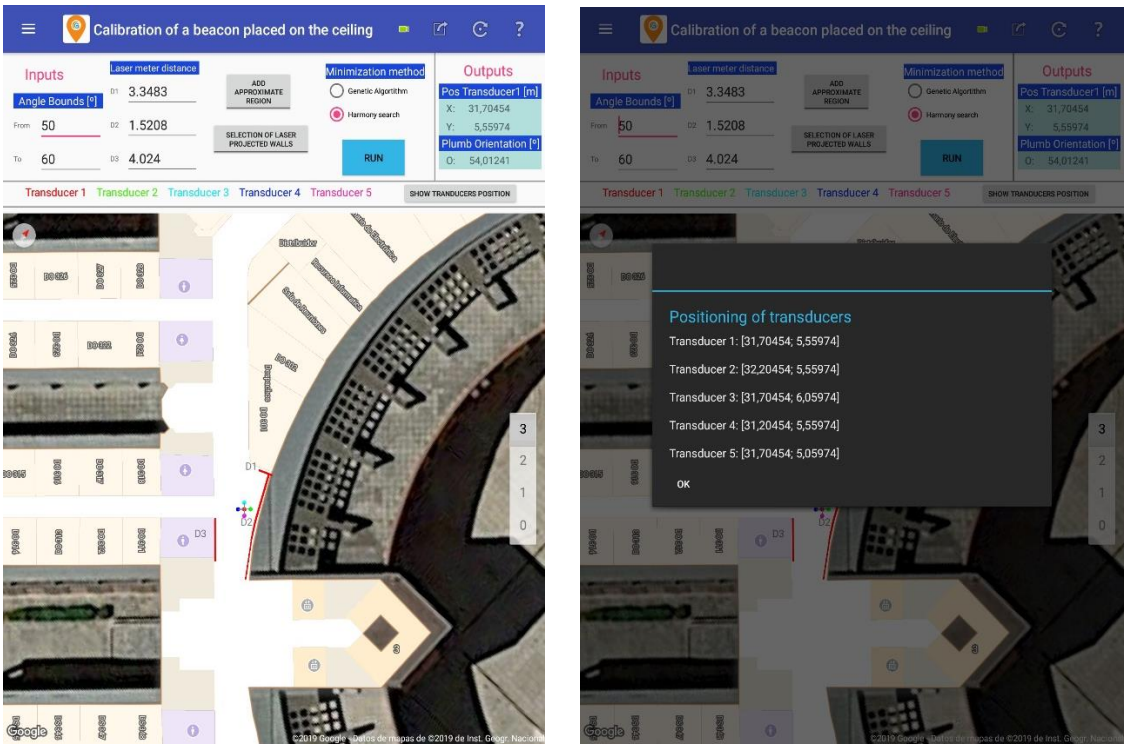


Figura 60: Pantalla principal tras finalizar la prueba 'Beacon on the ceiling'.

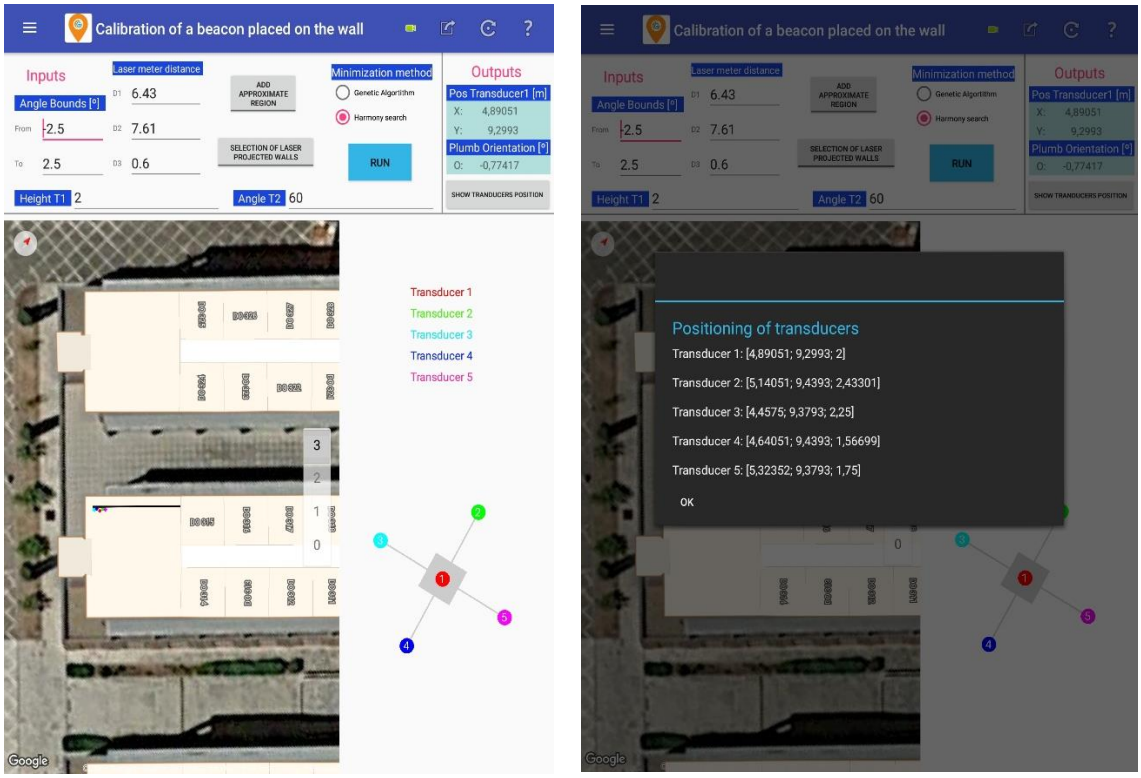
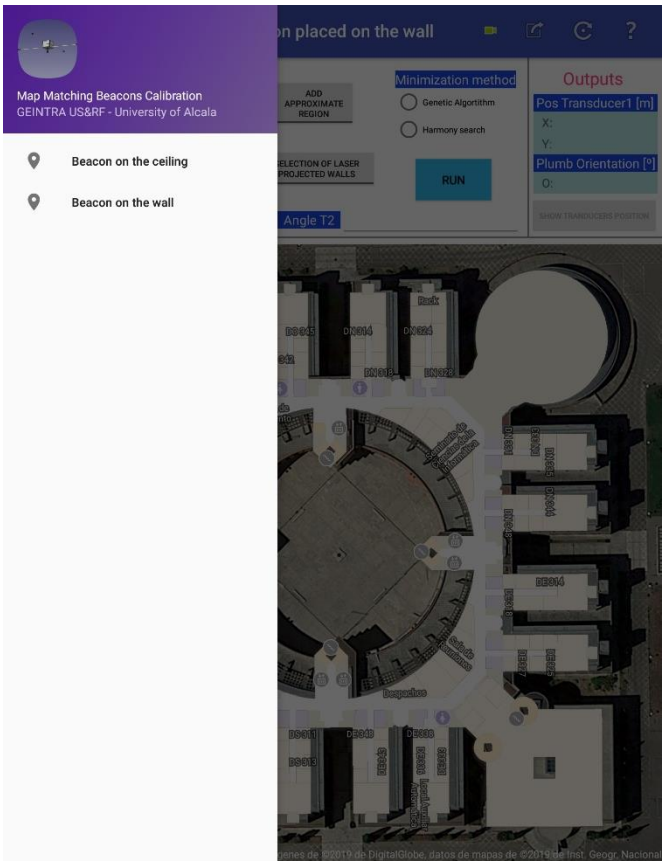


Figura 61: Pantalla principal tras finalizar la prueba 'Beacon on the wall'.

iii. Transición entre pantallas



Se ha programado un menú deslizante (que se despliega al pulsar el icono situado en la parte superior izquierda de las pantallas, como se puede observar enmarcado en las imágenes del final de la página) que permita pasar de la calibración de balizas en el techo a la pared (pulsando el ítem *Beacon on the wall*) o viceversa (pulsando *Beacon on the ceiling*).

Figura 62: Transición entre pantallas (I).

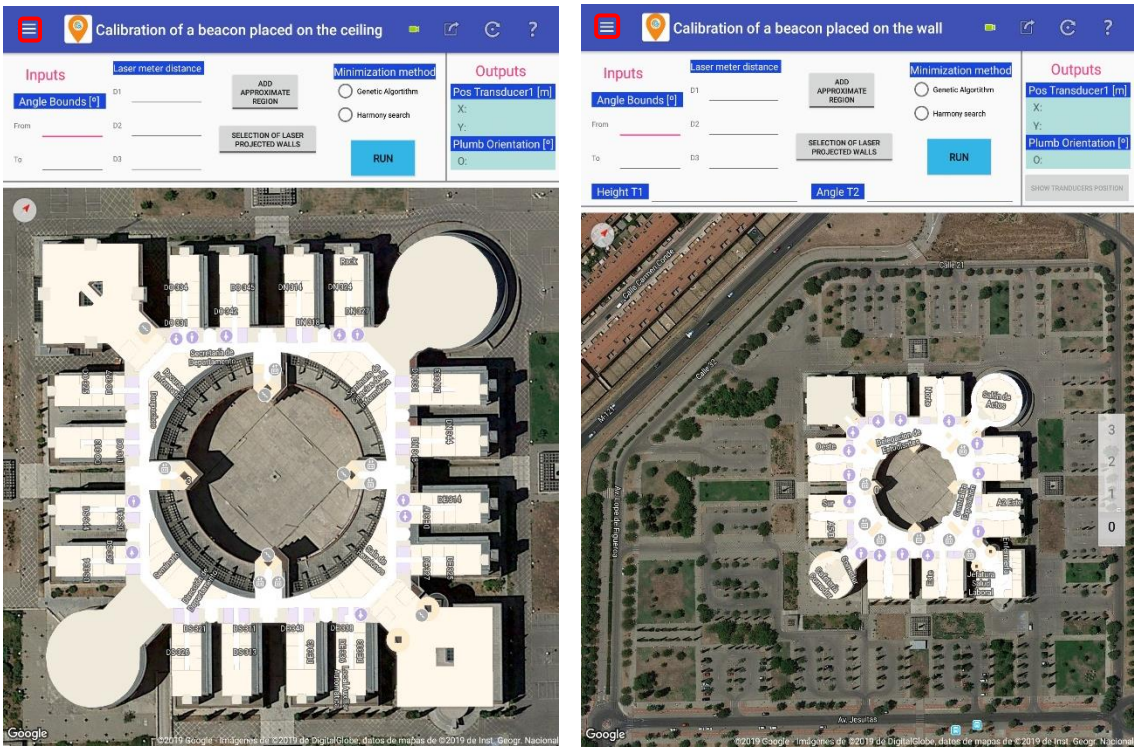
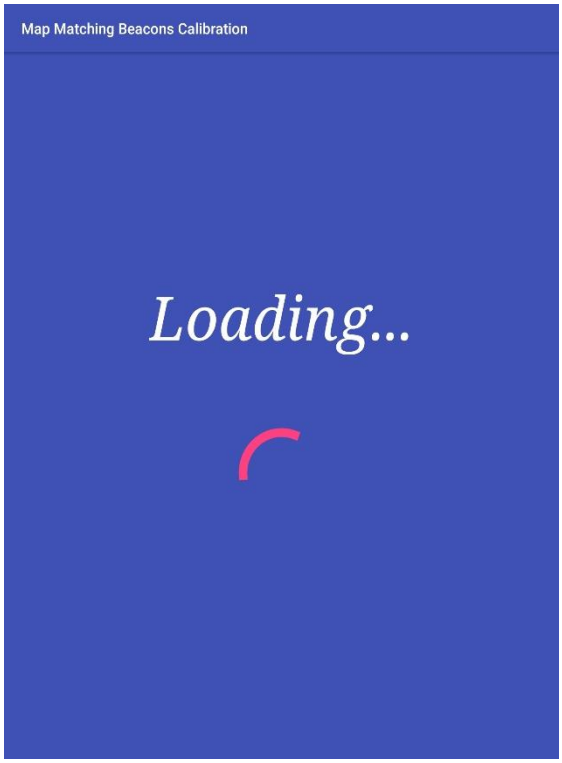


Figura 63: Transición entre pantallas (II).



Para realizar correctamente la transición entre las actividades que manejan los dos modos de calibración se ha tenido que programar una ventana intermedia, que garantice que en ningún momento hay dos Mapas de Google activos a la vez.

Figura 64: Transición entre pantallas (III).

Si no se utiliza esta pantalla intermedia lo que ocurre al intentar hacer la transición entre actividades, es que como al finalizar una (mediante el método *finish*), se realizaba el método *intent* para cargar la otra (algo que se produce de manera automática), se producía un momento al cargar la nueva actividad en el que Google detecta que intentas abrir una actividad que contiene un *fragment* Maps cuando ya hay otra activa que también lo tiene (método *finish* no se realiza de manera instantánea, tarda algo de tiempo (del orden de ms)), por lo que no te da los recursos del mapa de Google a la ventana a la que intentas pasar.

No es el caso, pero, aunque se intentara hacer uso de dos mapas de Google de manera simultánea para ofrecer más servicios al usuario en la aplicación, no se puede, ya que Google lo entiende como una estrategia para no sobrepasar las solicitaciones por segundo límite.

Al hacer uso de esta pantalla de transición, se cerciora de que se cierre perfectamente la actividad en la que se está y se pueda dar paso a la actividad a la que se quiere ir, pudiendo hacer uso en ambas de los servicios de Google.

iv. Botones de ayuda

Se han programado una serie de botones en la parte de la superior de la pantalla, más en concreto en la *ActionBar* de la actividad, para facilitar al máximo el uso de la aplicación y están enfocados hacia un usuario sin experiencia en el manejo de la app.

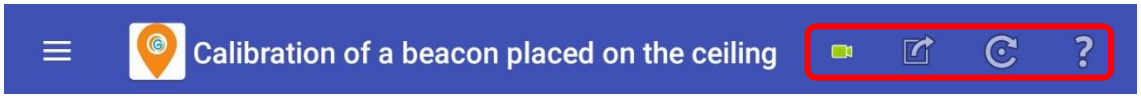


Figura 65: Botones de ayuda de la pantalla principal

- Botón ***Tutorial***: da acceso a un video tutorial del manejo de la aplicación desde el arranque para el modo de calibración en la pantalla en la que se esté.

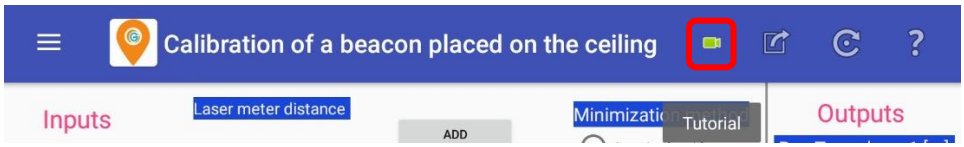


Figura 66: Botón de ayuda 'Tutorial'.

- Botón ***Test Button***: introduce automáticamente unas entradas de un ejemplo concreto, para realizar una demo rápida de cómo funciona la app a partir de unas entradas adecuadas.

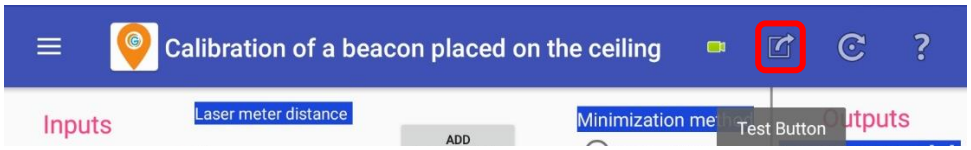


Figura 67: Botón de ayuda 'Test Button'.

- Botón ***Screen Cleaner***: resetea todos los datos de la pantalla (entradas, resultados anteriores).

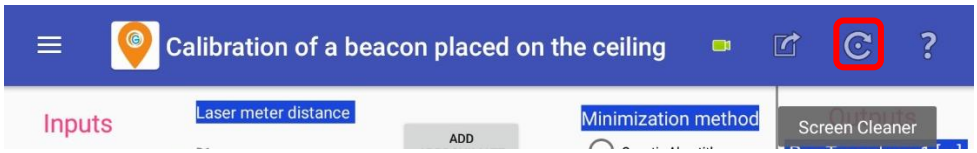


Figura 68: Botón de ayuda 'Screen Cleaner'.

- Botón ***User Guide***: manual de usuario deslizable del funcionamiento del modo de calibración en el que se esté.

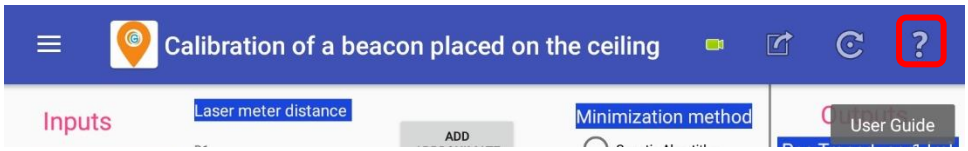


Figura 69: Botón de ayuda 'User Guide'.

11. Bibliografía

[Abido, 2002] Abido, M., "Optimal Power Flow using Tabu Search Algorithm", Electric Power Components and Systems, vol. 30, pp. 469 – 483, 2002.

[Abido, 2002] Abido, M., "Optimal Power Flow using Particle Swarm optimization", Electrical Power and Energy Systems, N◦24, pp. 563 – 571, 2002.

[Aitenbichler, 2003] E. Aitenbichler, M. Muhlhauser, "An IR local positioning system for smart items and devices", 23rd International Conference on Distributed Computing Systems Workshops, 2003.

[Allaoua, 2009] Allaoua, B., Laoufi, A., "Optimal Power Flow Solution Using Ant Manners for Electrical Network", Advances in Electrical and Computer Engineering, vol. 9, N◦1, pp. 34 – 40, 2009.

[Bahl, 2000] P. Bahl, V. N. Padmanabhan, "RADAR: an in-building RF-based user location and tracking system", Proceedings of the IEEE Conference on Computer Communications, vol. 2, pp. 775–784, 2000.

[Belmonte, 2018] O. Belmonte, R. Montoliu, J. Torres-Sospedra, E. Sansano-Sansano, D. Chia-Aguilar, "A radiosity-based method to avoid calibration for indoor positioning systems", Expert Systems with Applications, vol. 105, pp. 89–101, 2018.

[Blum, 2003] Blum, C., Roli A. (2003), "Metaheuristics in combinatorial optimization: Overview and conceptual comparison.", ACM Computing Surveys, vol. 35, No. 3, p. 268-308.

[Cobos, 2011] C. Cobos, M.S, J. Pérez, D. Estupiñan, "A survey of harmony search", Revista Avances en Sistemas e Informática, Medellín, vol. 28 nº2, Julio, 2011.

[Filonenko, 2013] V. Filonenko, C. Cullen, J. D. Carswell, "Indoor positioning for smartphones using asynchronous ultrasound trilateration," ISPRS International Journal of Geo-Information, vol. 2, pp. 598-620, 2013.

[Fisher, 1997] R. A. Fisher, G. R. De Beer, "Thomas Hunt Morgan. 1866-1945", Obituary Notices of Fellows of the Royal Society 5, 451-466, 1997.

[García, 2011] J. C. García, M. Marrón, F. Paulo, T. Bastos, S. Palazuelos, J. L. Sánchez, D. Gualda, "Proposal for a Building Accessibility Service (BASE)", Everyday Technology for Independence and Care (AAATE), Maastricht, vol. 29, pp. 393-400, 2011.

[Glover, 1986] Glover, F., "Future paths for integer programming and links to artificial intelligence", Computersand Operations Research, N◦13, pp. 533 – 549, 1986.

[Gorostiza, 2011] E. M. Gorostiza, J. L. Lázaro Galilea, F. J. Meca Meca, D. Salido Monzú, F. Espinosa Zapata, L. Pallarés Puerto, "Infrared Sensor System for Mobile-Robot Positioning in Intelligent Spaces", Sensors, vol.11, pp. 5416–5438.

[Gualda, 2019] D. Gualda, J. Ureña, J. Alcalá, and C. Santos, "Calibration of Beacons for Indoor Environments based on a Digital Map and Heuristic Information", Sensors, vol. 19, no. 670, pp 1 - 17, 2019.

[Holland, 1975] J. H. Holland, "Adaptation in Natural and Artificial Systems", Ann Arbor: The University of Michigan Press, 1975.

[Li, 2010] X. Li, J. Wang, A. Olesk, N. Knight, W. Ding, "Indoor positioning within a single camera and 3D maps", Ubiquitous Positioning Indoor Navigation and Location Based Service, Kirkkonummi, pp. 1–9, 2010.

[Lopes, 2014] S. I. Lopes, J. M. N. Vieria, J. Reis, D. Alburqueque, "Accurate Smartphone indoor positioning using WSN infrastructure and non-invasive audio for TDoA estimation," Pervasive and Mobile Computing, pp. 1-18, 2014.

[Mahajan, 1999] A. Mahajan, F. Figueroa, "An automatic self-installation and calibration method for a 3D position sensing system using ultrasonics", Robotics and Autonomous Systems, vol. 28, pp. 281-294, 1999.

[Mahdavi, 2009] M. Mahdavi, M. Fesanghary, E. Damangir, "An improved harmony search algorithm for solving optimization problems. Applied Mathematics and Computation", pp. 1567-1579, 2007.

[Melian, 2003] Melian, B., Moreno, J.A., Moreno, J.M., "Metaheuristics: A global view", Revista Iberoamericana de Inteligencia Artificial, N◦19, pp. 7 – 28, 2003.

[Niu, 2015] J. Niu, B. Wang, L. Cheng, J. J. P. C. Rodrigues, "WicLoc: An indoor localization system based on WiFi fingerprints and crowdsourcing", IEEE International Conference on Communications, London, pp. 3008–3013, 2015.

[Opdenbosch, 2014] D. Van Opdenbosch, G. Schroth, R. Huitl, S. Hilsenbeck, A. Garcea, E. Steinbach, "Camera-based indoor positioning using scalable streaming of compressed binary image signatures", IEEE International Conference on Image Processing (ICIP), Paris, pp. 2804–2808, 2014.

[Pérez, 2016] M. C. Pérez, D. Gualda, J. M. Villadangos, J. Ureña, P. Pajuelo, E. Díaz, E. García, "Android application for indoor positioning of mobile devices using ultrasonic signals", Proceedings of the 2016 International Conference on Indoor Positioning and Indoor Navigation, Octubre, 2016.

[Robologs, 2015] Algoritmos genéticos. Página web: https://robologs.net (visto a día 17 de Junio de 2019).

[Ruiz, 2013] D. Ruiz, E. Garcia, J. Urena, D. de Diego, D. Gualda, J. C. Garcia, "Extensive Ultrasonic Local Positioning System for navigating with mobile robots", Workshop on Positioning, Navigation and Communication (WPNC), Desdren, pp. 1-6, 2013.

[Ureña, 2007] J. Ureña, A. Hernández, A. Jiménez, J. M. Villadangos, M. Mazo, J. C. García, J. J. García, F. J. Álvarez, C. De Marziani, M. C. Pérez, J. A. Jiménez, A. R. Jiménez, F. Seco, "Advanced sensorial system for an acoustic LPS, Microprocessors and Microsystems", vol. 31, pp. 393-401, 2007.

[Ureña, 2011] J. Ureña, D. Ruiz, J. C. García, J. J. García, A. Hernández, M. C. Pérez, "LPS self-calibration method using a mobile robot", Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Hangzhou, pp. 1-6, 2011.

[Ureña, 2015] J. Ureña, D. Gualda, A. Hernández, E. García, J. M. Villadangos, M. Carmen Pérez, J. C. García, J. J. García, A. Jiménez, "Ultrasonic local positioning system for mobile robot navigation: From low to high level processing", Proceedings of the IEEE International Conference on Industrial Technology (ICIT), Sevilla, pp. 3440–3445, 2015.

[Ureña, 2018] J. Ureña, A. Hernández, J. J. García, J. M. Villadangos, M. C. Pérez, D. Gualda, F. J. Álvarez, T. Aguilera, "Acoustic local positioning with encoded emission beacons" Proceedings of the IEEE, vol. 106, no. 6, Junio 2018.

[Vaisakh, 2008] Vaisakh, K., Srinivas, L., "Differential Evolution Approach for Optimal Power Flow Solution", Journal of Theoretical and Applied Information Technology, vol. 2, pp. 261 – 268, 2008.

[Wang, 2010] C.-M. Wang, Y.-F. Huang, "Self-adaptive harmony search algorithm for optimization", Expert Systems with Applications, vol. 37, pp. 2826–2837, 2010.

[Xiao, 2014] Ting-Ting Xiao, Xing-Yu Liao, Ke Hu, Min Yu, "Study of fingerprint location algorithm based on WiFi technology for indoor localization", 10th International Conference on Wireless Communications, Networking and Mobile Computing, Beijing, pp. 604–608, 2014.

[Yang, 2009] Yang, X.-S, "Harmony Search as a Metaheuristic Algorithm in Music-Inspired Harmony search Algorithm", Springer Berlin, Heidelberg, pp. 1-14. 2009.

[Zanakins, 1981] Zanakins, S. H., Evans, J. R., "Heuristic 'Optimization': Why, When and How to Use It", Interfaces, vol. 11, 1981.

[Zong, 2001] Zong Woo Geem, Joong Hoon Kim, G. V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search", Simulation, vol. 76, pp. 60–68, 2001.

[Zong, 2009] Zong Woo Geem, "Music- Inspired Harmony Search Algorithm: Theory and Applicationes", pp. 206, 2009.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá