ULPGC
Universidad de
Las Palmas de
Gran Canaria

Escuela de
Ingeniería Informática

Sistema de procesamiento de datos para la mejora del posicionamiento en interiores: integración con plataforma robótica de toma de datos Bluetooth Low Energy.

Trabajo de Fin de Grado

Autor:

Simon Teodor Rusu Stratulat

Tutores:

Alexis Quesada Arencibia

Gabriele Salvatore De Blasio

Julio de 2023

Grado en Ingeniería Informática

Universidad de Las Palmas de Gran Canaria

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a todo el equipo del Instituto Universitario de Cibernética, Empresa y Sociedad por el apoyo ofrecido durante la realización del presente trabajo, en especial a mis tutores Gabriele Salvatore de Blasio y Alexis Quesada Arencibia.

También agradecer a mi familia por brindarme su respaldo durante todo el transcurso de toda esta etapa y a los profesores del Grado de Ingeniería de la Universidad de Las Palmas de Gran Canaria por el inmenso esfuerzo que realizan año tras año, formando y capacitando a futuros profesionales capaces de lograr grandes hazañas.

Sin la implicación y dedicación de todos ellos, este proyecto no podría haber sido posible.

Índice

Índice de figuras

Resumen

El trabajo propuesto es una continuación del proyecto *Robomap*, enfocado en el desarrollo de IPS (*Sistemas de Posicionamiento en Interiores*) mediante tecnologías Bluetooth y Wi-Fi, por el IUCES (*Instituto Universitario de Cibernética, Empresa y Sociedad*). El propósito de este proyecto es desarrollar una herramienta que permita agilizar y optimizar el procesamiento de los datos obtenidos por la plataforma robótica *Robomap* facilitando la incorporación y pruebas de nuevas metodologías de posicionamiento en el futuro. Para ello, se realiza la migración de las utilidades de procesamiento a la herramienta, al mismo tiempo que se integran nuevos métodos IPS. Además, se incorporan gráficas estadísticas para comparar en cada método aplicado, valores reales y obtenidos, evaluando precisión y exactitud.

Abstract

The proposed work is a continuation of the *Robomap* project, focused on the development of IPS (*Indoor Positioning Systems*) using Bluetooth and Wi-Fi technologies, by the IUCES (*University Institute of Cybernetics, Business, and Society*). The purpose of this project is to develop a tool that streamlines and optimizes the processing of data obtained by the *Robomap* robotic platform, facilitating the incorporation and testing of new positioning methodologies in the future. To achieve this, the processing utilities are migrated to the tool, while new IPS methods are integrated. Additionally, statistical graphs are included to compare real and obtained values for each applied method, evaluating precision and accuracy.

1. Introducción

Los Sistemas de Posicionamiento en Interiores o IPS (*Indoor Positioning Systems*) permiten localizar personas u objetos dentro de los espacios interiores: viviendas, instalaciones, plataformas de transporte público, etc., en donde la señal GPS (*Global Positioning System*) es de baja intensidad o inexistente. Para ello, este sistema propone el despliegue de una infraestructura de dispositivos emisores y receptores que llevan a cabo un procedimiento de recopilación de información sobre los niveles de intensidad de las señales recibidas y así posicionar de forma precisa una determinada entidad [1].

Las aplicaciones de los IPS son diversas y abarcan diferentes áreas: orientación dentro de edificios o espacios cerrados, como hospitales, museos o aeropuertos; mejora de la experiencia del cliente en establecimientos o tiendas por medio de notificaciones basadas en su historial de compras o localización; evolución de la gestión logística, rastreo de recursos y optimización de rutas comerciales; seguimiento y localización de personas en tiempo real para casos de evacuación o ayuda a los equipos de seguridad; optimización en el uso de espacios y ocupación de edificios [2].

Existen numerosas soluciones tecnológicas utilizadas en el posicionamiento *indoor*, clasificadas en: ópticas, magnéticas, acústicas o de radiofrecuencia. Entre estas categorías, las tecnologías de radiofrecuencia son especialmente relevantes y ampliamente utilizadas en el presente trabajo. En particular, las tecnologías de banda estrecha BLE (*Bluethooth Low Energy*) y Wi-Fi. Si bien ambas están en constante desarrollo, BLE proporciona una mayor precisión a la hora de realizar un posicionamiento, pero tiene un menor alcance y necesita de un mayor número de emisores de señal [3].

Para realizar el posicionamiento de objetos en una escena es necesario utilizar medidas o señales procedentes de varias fuentes emisoras y así calcular la posición aproximada de una entidad en un tiempo determinado. Una vez se han obtenido las mediciones recolectadas de estas fuentes, se hace uso de diferentes métodos para determinar la posición. Estos se clasifican en dos categorías principales: los

deterministas, que incluyen la triangulación y la trilateración; y los basados en muestreo, como el *Fingerprinting* [3].

En particular, en el proyecto actual se ha hecho hincapié en el uso del método de *Fingerprinting* para el posicionamiento en interiores. Este método se fundamenta en la comparación de las mediciones de señales inalámbricas actuales con una serie de patrones almacenados previamente en una base de datos. Tras realizar este proceso comparativo, se deduce la posición más probable del objeto en la escena. *Fingerprinting* tiene la capacidad de superar las limitaciones ocasionadas por obstáculos o interferencias, lo cual lo convierten en una opción sólida respecto a otras alternativas [3].

Este método consta de dos fases: de calibrado y de posicionamiento [4]. Durante la etapa de calibración, se establece una correspondencia entre las coordenadas cartesianas previamente conocidas del receptor en diferentes puntos específicos del entorno y la media de intensidad de señal recibida, conocida como RSSI (*Received Signal Strength Indicator*). Las coordenadas reales del receptor en el escenario son adquiridas a través de un sistema adicional de posicionamiento, el cual se emplea como referencia para evaluar la precisión y exactitud del método utilizado. La fase de posicionamiento se basa en determinar las coordenadas exactas de un objeto o receptor utilizando las potencias de las señales que se reciben de distintos emisores. Posteriormente, se comparan estas señales con las mediciones realizadas durante la fase de calibración [4].

Durante la fase de calibración del método de *Fingerprinting*, la colecta de datos se centra exclusivamente en tomar datos procedentes de los puntos de referencia. Por otro lado, durante la fase de posicionamiento se recolectan datos en los puntos aleatorios. Recopilar datos en puntos aleatorios, simula un escenario mucho más realista donde se puede evaluar la capacidad del método para realizar predicciones más precisas en ubicaciones no mapeadas anteriormente. Este procedimiento se beneficia de una muestra amplia para la calibración, así como de una muestra más pequeña pero aleatoria para la fase de posicionamiento. Esta combinación mejora significativamente la capacidad del método para generar predicciones precisas en todo el entorno.

En el presente proyecto, se emplea el procedimiento predictivo WkNN (*Weighted k-Nearest Neighbor*) para llevar a cabo la estimación de las coordenadas en la fase de posicionamiento. El método WkNN se basa en la selección de los vecinos más cercanos mediante la asignación de pesos ponderados según su cercanía y potencia de señal. Esto permite obtener una predicción precisa de las coordenadas del objeto en el momento de posicionamiento [4].

En este caso, para la recepción de señales, se hace uso del estándar BLE, el cual se basa en una serie de dispositivos emisores conocidos como beacons o balizas BLE. Estos dispositivos están diseñados para transmitir paquetes de datos en intervalos de tiempo regulares y configurables. Los beacons transmiten señales a través de protocolos específicos, siendo los más comunes iBeacon (*Apple*) y Eddystone (*Google*) [5]. En cuanto a los canales utilizados por lo beacons BLE, los seleccionados son los canales 37, 38 y 39, que están reservados para transmisiones de beacons y ofrecen un nivel bajo de interferencia con otros dispositivos Bluetooth [4].

El enfoque principal de este proyecto se centra en la etapa de procesamiento de la base de datos recolectada por *Robomap* tras finalizar la campaña de recopilación de datos. Debido a la extensión y alcance del proyecto general, en etapas anteriores no fue posible abordar en detalle la fase de procesamiento de datos. Sin embargo, es importante destacar que esta etapa es un componente crítico en la cadena de trabajo, ya que transforma los datos brutos en información útil y facilita el análisis posterior.

Esta fase de procesamiento se caracteriza por ser un proceso prolongado que tiene como finalidad representar los datos generados en gráficas comparativas, esto permite un análisis más profundo de la metodología empleada para realizar el posicionamiento.

La rutina de procesamiento actual es un proceso costoso, que se extiende en el tiempo y requiere de la ejecución de múltiples herramientas y la intervención manual del usuario en cada etapa. Esta complejidad y carga de trabajo significativa pueden llevar a errores y retrasos en la generación de resultados, además de requerir un esfuerzo adicional para analizar y comprender los datos de manera efectiva.

2. Estado del arte

La plataforma robótica *Robomap* (Figura 1) constituye un sistema autónomo que permite la captura automatizada de datos, mejorando considerablemente la eficiencia y rapidez de la fase de calibrado. Una vez completada esta etapa, se deben llevar a cabo una serie de procedimientos para acometer la fase de posicionamiento. Estos procedimientos permiten la transformación de los datos brutos obtenidos durante la calibración y posibilitan el análisis comparativo de diversas metodologías de posicionamiento, con el objetivo de identificar aquellas que arrojen los resultados más favorables.

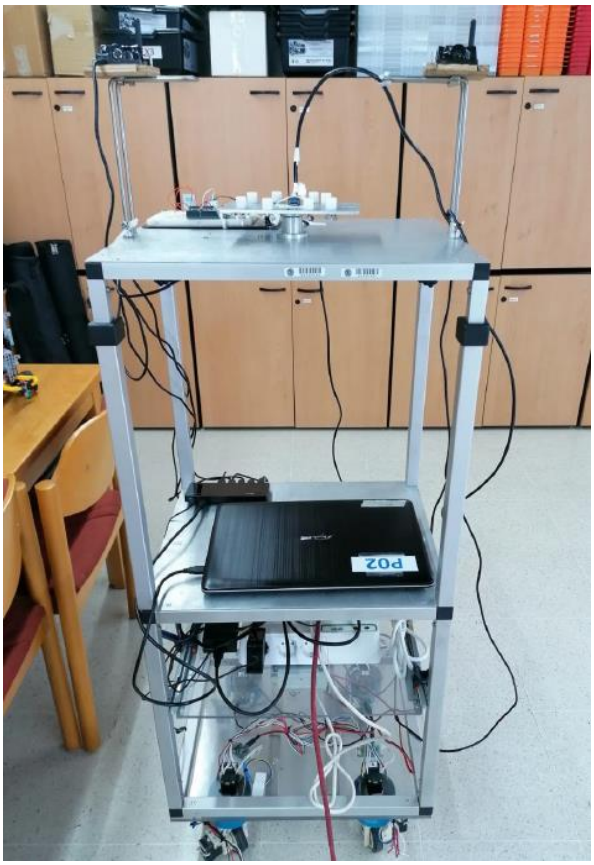

Figura 1: Plataforma robótica *Robomap*

2.1. Metodología de recopilación de datos y captura de información

Tal y como se ha introducido, la plataforma robótica *Robomap* se basa en el estándar BLE para la recepción de señales, aprovechando una serie de dispositivos emisores conocidos como beacons (Figura 2). Estos beacons emiten una señal de radio continua o periódica en una frecuencia específica, permitiendo la transmisión de información en tiempo real. Los dispositivos receptores, como *dongles* USB receptores de señales BLE (Figura 3), smartphones o tabletas, pueden captar e interpretar estas señales siempre que se encuentren dentro del rango de alcance de los beacons [6]. En el contexto de este trabajo, las señales emitidas por los beacons son recibidas directamente por un *dongle* USB conectado a una computadora que controla la plataforma robótica [4].

Figura 2: Beacons BLE

Figura 3: *Dongle Nrf51 Nordic Semiconductor*

Las capturas de los paquetes BLE se realizan tanto por la herramienta Wireshark [7] como en su versión por consola Tshark [8].

Wireshark es un analizador de protocolos utilizado principalmente para el análisis y la solución de problemas en redes de comunicaciones. Ofrece una interfaz gráfica con numerosas opciones de organización incluyendo un extenso lenguaje para el filtrado de datos [7]. Su combinación con Tshark hace de esta herramienta una opción ideal para realizar capturas de señales mediante la tecnología BLE [8].

2.2. Creación de la base de datos SQLite para la recopilación de datos

Tras finalizar la recogida de señales, Robomap hace uso de un servicio propietario denominado "*Script Service BBDD*", que encapsula en una base de datos SQLite (*.sqlite3*) todas las señales obtenidas por Wireshark. En la Figura 4 podemos observar el modelado de la base de datos.

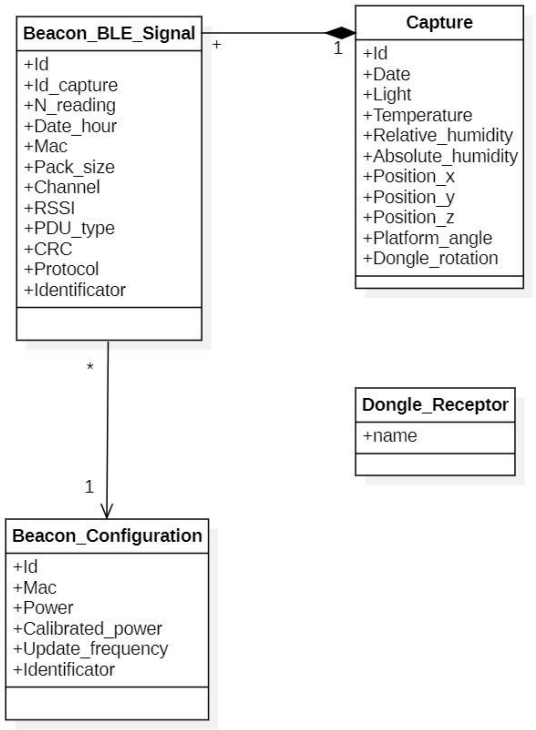

Figura 4: Diseño de base de datos de *Robomap*

2.3. Herramientas de conversión de datos mediante scripts en Python y Java

En este punto comienza la fase de preparación de los datos necesarios para el procesado de las señales BLE. En el proceso de exportación de las señales, se emplean varios scripts que automatizan las etapas necesarias para convertir la base de datos inicial (*.sqlite3*) en ficheros organizados por puntos físicos de la compaña en el formato adecuado.

Inicialmente, se utiliza un script en Python (*exportCSVfromDB.py*) que accede a la base de datos y ejecuta consultas SQL para obtener las señales en un punto físico específico, junto con el ángulo de rotación del dongle receptor utilizado en la campaña. Estas señales se convierten a formato .csv, y el script organiza todas las señales en carpetas por puntos físicos recorridos durante la campaña (Figura 5).

Figura 5: Salida de script *exportCSVfromDB.py*

Posteriormente, se ejecuta un script en Java (*exportXLSXfromCSV.java*) que recorre todos los ficheros .csv generados previamente y los transforma en ficheros finales en formato .xlsx. Estos ficheros contienen todas las señales de un punto organizadas por canal, protocolo y ángulo de rotación del *dongle* receptor BLE utilizado en la plataforma robótica (Figura 6).

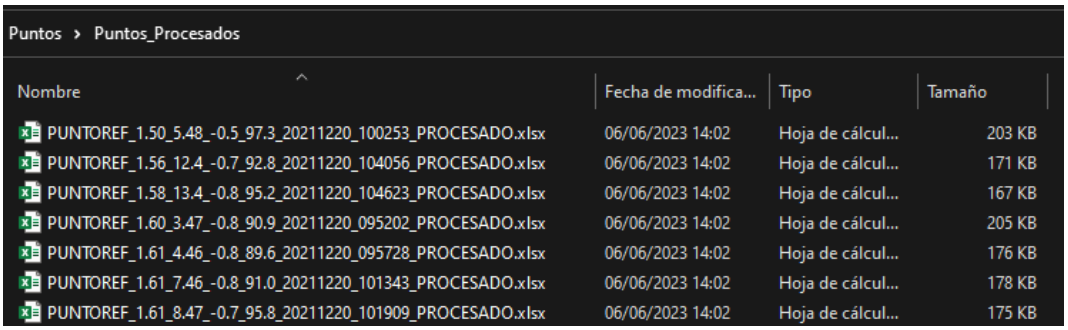

Figura 6: Salida de *exportXLSXfromCSV.java*

Finalmente, los ficheros .xlsx obtenidos se utilizan como parámetros de entrada en el algoritmo de posicionamiento *Fingerprinting* (*WkNN*) desarrollado en Matlab [9].

2.4. Importación de datos a Matlab y aplicación de algoritmos de posicionamiento

Matlab (del acrónimo, *Matrix Laboratory*) es un entorno de programación y software utilizado ampliamente en el ámbito académico y de investigación [10]. Cuenta con su propio entorno de desarrollo integrado (*IDE*) [11] y un conjunto de bibliotecas especializadas.

Debido a su potencia computacional y capacidad de manejo de datos, Matlab se utiliza ampliamente en el procesamiento de métodos de posicionamiento en interiores.

En particular, se aplica el método conocido como WkNN. Este método se basa en la comparación de las señales recibidas de diferentes emisores para estimar la ubicación de un objeto o individuo en un entorno interior. Utilizando algoritmos de clasificación y ponderación, el WkNN asigna diferentes pesos a las señales recibidas según su proximidad y precisión. A continuación, se calcula una ubicación estimada basada en las señales ponderadas. Esta aproximación ha demostrado ser notablemente efectiva para superar las restricciones ocasionadas por obstáculos físicos e interferencias de radio, lo que la convierte en una opción altamente atractiva para el posicionamiento en espacios interiores [12].

Una vez obtenidos los ficheros .xslx con todas las señales categorizadas de un determinado punto, se importan a Matlab por medio de un script que recoge

los datos y aplica el método WkNN obteniendo como salida las coordenadas estimadas del punto en el espacio Posteriormente, se calcula el error promedio como una medida de la diferencia entre las coordenadas reales y las estimadas.

El error promedio dictamina la exactitud de los datos y se suele mostrar por medio de tablas como la observada a continuación [13]. La Figura 7 ejemplifica el incremento que experimenta la precisión al pasar de 8 muestras a 16, esto se traduce en un aumento de entre 10 y 30 cm según la orientación en la toma de datos [4].

	8				16			
Prot.-Chann.	N	E	S	W	N	E	S	W
iB37	1.7	1.3	1.4	1.6	1.4	1.2	1.3	1.6
	1.7	1.2	1.3	1.5	1.4	1.1	1.3	1.6
Ed37	1.6	1.2	1.4	1.6	1.6	1.3	1.3	1.5
	1.6	1.2	1.4	1.6	1.5	1.2	1.3	1.5
iB38	1.8	1.3	1.5	1.6	1.6	1.3	1.3	1.5
	1.8	1.3	1.5	1.5	1.7	1.3	1.4	1.5
Ed38	1.8	1.2	1.5	1.6	1.6	1.2	1.4	1.6
	1.8	1.2	1.5	1.5	1.7	1.2	1.4	1.6
iB39	1.6	1.5	1.4	1.5	1.5	1.5	1.3	1.4
	1.6	1.4	1.4	1.5	1.6	1.4	1.3	1.4
Ed39	1.6	1.4	1.3	1.6	1.5	1.5	1.2	1.5
	1.5	1.5	1.4	1.5	1.5	1.4	1.2	1.5

Figura 7: Tabla de exactitud. Extraído de [4].

Por otra parte, se calcula la función de distribución acumulada o CDF (*Cumulative Distibution Function*). Este cálculo se lleva a cabo mediante la ordenación de menor a mayor de los errores promedios obtenidos en los diferentes puntos aleatorios. El cálculo del error generalmente se realiza utilizando alguna métrica de distancia, como la euclidiana [14]. En la Figura 8 se muestran ejemplos de gráficas de precisión por medio de la representación de la CDF [13].

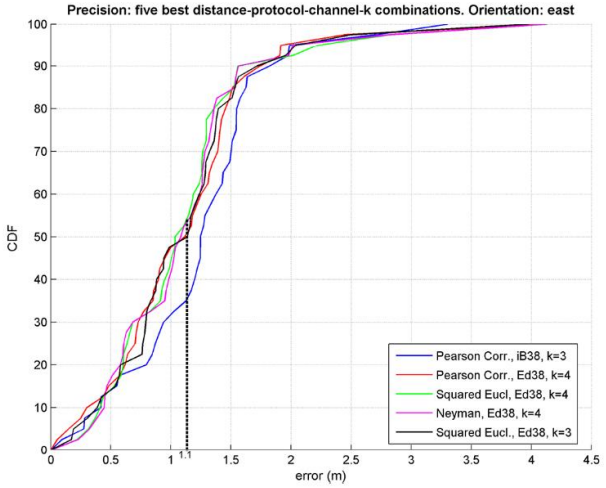

Figura 8: Gráfica de precisión. Extraído de [4].

Una vez completado todo el procedimiento de procesamiento de datos y obtenidas las coordenadas predichas, se puede realizar una evaluación de la precisión y exactitud del método de posicionamiento WkNN. Para ello, se utilizan las gráficas de precision y exactitud como salidas finales. Estas desempeñan un papel crucial en la evaluación y validación del método de posicionamiento implementado. Proporcionan información valiosa sobre cómo se comporta el algoritmo en diferentes condiciones y áreas geográficas. Además, permiten identificar posibles patrones en los errores de posicionamiento, ayudando a identificar áreas de mejora y permiten refinar el algoritmo aplicado.

3. Objetivos

En este apartado, se presentan los objetivos que se pretenden alcanzar a través del desarrollo de la herramienta denominada *IPSCampaignManager*.

- Desarrollar un sistema que reemplace el procedimiento actual de análisis de datos, el cual involucra la utilización de varios scripts para exportar los datos de posicionamiento desde la base de datos propietaria de *Robomap* hacia Matlab. El sistema propuesto busca mejorar y automatizar este proceso, facilitando la extracción y procesamiento de los datos de posicionamiento de manera más eficiente y precisa.
- Migrar toda la etapa de procesamiento actual a un nuevo enfoque que simplifique y mejore la eficiencia del mismo. Este nuevo enfoque tiene como objetivo reducir significativamente los tiempos de procesamiento, así como mejorar la transparencia en el almacenamiento de los resultados para el usuario. Se busca implementar un sistema que permita almacenar los resultados de la forma más organizada y accesible posible.
- Además de la implementación del método WkNN, se propone la integración de nuevos métodos de posicionamiento en el sistema. Se pretende habilitar la comparación entre estos métodos, permitiendo la aplicación dinámica de cada uno con sus respectivos hiperparámetros. Además, se persigue desarrollar el sistema de forma escalable, de modo

que en el futuro sea sencilla la incorporación de nuevos métodos de procesamiento.

- Facilitar el despliegue del software mediante una arquitectura basada en contenedores. Esta implementación permitirá que la migración de la herramienta de un PC a otro sea prácticamente transparente para el usuario, sin la necesidad de instalar librerías, herramientas o SDK adicionales para hacerla compatible con el nuevo entorno.

Por otra parte, es posible definir objetivos adicionales que se centren en el proceso de desarrollo desde la perspectiva del estudiante.

- Contribuir al desarrollo de un proyecto académico consolidado, aportando conocimientos y experiencia para su mejora y evolución.
- Fomentar el aprendizaje y la adquisición de conocimientos específicos en el campo de los Sistemas de Posicionamiento en Interiores (*IPS*).
- Desarrollar un sistema de talla profesional, haciendo uso de las últimas tecnologías del sector y aplicando buenas prácticas en el desarrollo de código limpio.

4. Justificación de competencias específicas

FB1: Capacidad para la resolución de los problemas matemáticos que puedan plantearse en la ingeniería. Aptitud para aplicar los conocimientos sobre: álgebra lineal, cálculo diferencial e integral, métodos numéricos, algorítmica numérica, estadística y optimización.

En el desarrollo de los algoritmos de posicionamiento IPS, se han utilizado y manipulado sistemas matriciales para representar y procesar los datos de señales recibidas. Esto implica aplicar conocimientos de álgebra lineal. Además, la preparación de los datos y la aplicación de métodos como WkNN, SVR, NuSVR y LinearSVR, por medio de librerías especializadas o el cálculo de la distancia euclidiana requiere de un dominio de las técnicas matemáticas correspondientes.

FB4: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación de la ingeniería.

Durante el proceso de desarrollo, se realiza el manejo de ficheros, interactuando con el sistema operativo para realizar operaciones de lectura, escritura y manipulación de datos. Además, se trabaja con bases de datos relacionales avanzadas, aplicando conceptos como la normalización, diseño de tablas y relaciones, además de realizar consultas múltiples para extraer información relevante.

FB5: Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de la programación, y su aplicación para la resolución de problemas propios de la ingeniería.

Esto se ha conseguido por medio de la propuesta de una arquitectura de desarrollo basada en contenedores Docker. Los contenedores se ejecutan de forma independiente, virtualizados sobre el sistema operativo, y están interconectados mediante su propia red. Esta estructura permite el traspaso de información entre los contenedores de manera transparente para el usuario.

CI1: Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Esto ha sido posible gracias a las buenas prácticas de desarrollo software, siguiendo estándares de calidad y seguridad, como se evidencia en el cuidadoso filtrado de los parámetros introducidos por el usuario al dar de alta una nueva campaña o aplicar los métodos de posicionamiento deseados. Esta medida garantiza que el sistema esté protegido y no se vea afectado en caso de recibir datos inesperados o incorrectos.

CI6: Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.

Se ha aplicado un enfoque algorítmico sólido para diseñar multitud de soluciones a problemas durante el desarrollo del sistema. Se han desarrollado funciones reutilizables que desempeñan un papel fundamental al reducir considerablemente la

carga del sistema y garantizar un rendimiento óptimo. Estas funciones están cuidadosamente diseñadas para abordar los problemas planteados sin generar efectos secundarios que puedan afectar negativamente el rendimiento general del sistema.

CI7: Conocimiento, diseño y utilización de forma eficiente de los tipos y estructuras de datos más adecuados a la resolución de un problema.

Se ha trabajado con una amplia variedad de estructuras de datos, como matrices, listas, objetos JSON, números enteros y en punto flotante, y cadenas de texto. Se ha prestado especial atención a la conversión y transformación de estos datos, asegurando su correcto almacenamiento en la base de datos SQLite y su posterior recuperación en un formato adecuado para un procesamiento óptimo.

CI8: Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

La consecución de un sistema compuesto por un backend con una API RESTful que proporciona los servicios y la lógica de negocio necesarios, así como un frontend o aplicación web que interactúa con el usuario. Esta arquitectura robusta y eficiente permite un intercambio fluido de información entre ambas partes del sistema, asegurando su correcto funcionamiento y una experiencia de usuario satisfactoria. Para ello, se ha realizado un análisis exhaustivo de las tecnologías disponibles para cada componente del sistema, seleccionando cuidadosamente aquellas que mejor se adaptan a los requisitos y objetivos del proyecto.

CI13: Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los sistemas de información, incluidos los basados en web.

En este proyecto, se ha utilizado la base de datos SQLite como medio de almacenamiento de los datos generados por el sistema. Además, se ha implementado una estrategia eficiente para el manejo de imágenes, evitando almacenarlas directamente en la base de datos y en su lugar comprimiéndolas y almacenándolas en un directorio específico en el propio backend.

CI17: Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.

Se ha otorgado especial cuidado al diseño de la interfaz de usuario, buscando que sea intuitiva, fácil de usar y accesible para los usuarios. Se ha seguido un enfoque de simplicidad, evitando sobrecargar la interfaz con información innecesaria y además se ha reducido el número de vistas necesarias para completar tareas. Esto se ha logrado mediante una cuidadosa organización y presentación de la información, priorizando la eficiencia y la usabilidad del sistema.

TI3: Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas.

Para obtener esta competencia se ha seguido una metodología de desarrollo basada en el diseño de prototipos. Esta metodología ha permitido iterar sobre el diseño de la arquitectura software y realizar ajustes continuos en función de las necesidades y preferencias de los usuarios.

5. Aportaciones al entorno socioeconómico, técnico o científico

El sistema desarrollado otorga importantes aportaciones al ámbito académico y científico. *IPSCampaignManager* ofrece la posibilidad de simplificar y reducir las tareas relacionadas con el procesamiento de datos, la aplicación de métodos de posicionamiento en interiores y la posterior comparación de datos estadísticos mediante la generación dinámica de gráficas. Esta optimización de procesos permitirá realizar avances significativos en el proyecto *Robomap*, ya que se ha desarrollado una herramienta que facilita la exploración y análisis de sus datos, que suponen un gran volumen de información pendiente de procesar.

Al simplificar las tareas de procesamiento y análisis de datos, el sistema brinda a los investigadores y profesionales una herramienta eficiente y confiable para llevar a cabo estudios en el campo de los Sistemas de Posicionamiento en Interiores. La capacidad de comparar diferentes métodos de posicionamiento, junto con la generación de datos estadísticos, proporciona información valiosa para evaluar el rendimiento y la precisión de los algoritmos utilizados.

El sistema también destaca por su implementación basada en las últimas tecnologías, lo que le confiere una gran flexibilidad y escalabilidad. Gracias a esto, el sistema es capaz de adaptarse y actualizarse fácilmente para satisfacer las necesidades futuras. Su diseño modular permite una integración nativa con otras aplicaciones desarrolladas posteriormente.

Por último, el *Backend* de la API desarrollada tiene la capacidad de ser reutilizado por múltiples aplicaciones, lo que permite nutrir de datos y funcionalidades a diferentes sistemas de forma simultánea. Esto se logra a través de la implementación de *endpoints* [15] que pueden ser consumidos por diversas aplicaciones de manera independiente.

Gracias a esta arquitectura flexible, se evita la congestión o bloqueo de la API al ejecutar simultáneamente el código necesario para dar un servicio eficiente y fiable.

6. Metodología y planificación

Durante el desarrollo de la herramienta se hace uso de una metodología ágil de desarrollo por prototipos, esta se caracteriza por dotar al desarrollo software de un enfoque iterativo e incremental [16].

La metodología se inicia con la recopilación y análisis de requisitos, seguido del desarrollo y prototipado de la herramienta. En esta etapa, se elabora un prototipo funcional que cumpla con los requisitos preestablecidos. Lo siguiente es mostrar el prototipo al cliente y recibir la retroalimentación correspondiente, en caso de requerirlo se vuelve a la fase de diseño y se construye un nuevo prototipo que satisfaga los nuevos requisitos del cliente. Este proceso se repite hasta que se obtiene un producto final de ingeniería (Véase la Figura 9).

Figura 9: Ciclo de vida de modelo por prototipos. Extraído de [16].

La elección de esta metodología se basó en la falta de claridad en los requisitos iniciales del sistema y los posibles cambios que pudieran surgir en etapas más avanzadas del desarrollo. Además, el uso de esta metodología posibilita que el cliente, en este caso, el grupo de investigación del IUCES a cargo del proyecto pueda evaluar y probar las nuevas versiones de la herramienta en un transcurso de tiempo reducido. Esto permite entender bien el problema y poder dirigir el desarrollo hacia la dirección deseada sin la necesidad de invertir mucho tiempo y esfuerzo en la especificación de requisitos.

En relación con la planificación establecida en el Trabajo Fin de Titulación TFT-01, se ha decidido asignar un tiempo adicional al inicialmente previsto con el fin de

garantizar un desarrollo exhaustivo y de calidad que contribuya a una mejora significativa del proyecto a largo plazo. Esta decisión se fundamenta en la intención de proporcionar un aporte significativo y relevante al proyecto, abordando de manera rigurosa cada uno de los aspectos a desarrollar y finalizando todas las tareas que surgieron a raíz de los requerimientos presentados por el equipo de investigación del IUCES encargado del proyecto.

7. Tecnologías y herramientas

En el marco de este trabajo, se ha llevado a cabo un exhaustivo análisis con el propósito de seleccionar las tecnologías y herramientas más adecuadas para el desarrollo del proyecto presentado. Este proceso de selección se basó en una cuidadosa evaluación de los requisitos y objetivos establecidos, con el fin de identificar aquellas soluciones que mejor se ajustaran a las necesidades especificadas.

7.1. Docker

Docker es una tecnología de organización de contenedores que permite elaborar y utilizar contenedores de Linux, los contenedores pueden ser considerados como máquinas virtuales muy livianas y modulares. Esta herramienta proporciona una forma limpia y eficiente de ejecutar servicios y aplicaciones al hacer uso del kernel de Linux y aprovechar su tecnología para ejecutar procesos de forma aislada y segura [17].

Figura 10: Logo de Docker

Una de las grandes ventajas de Docker es que permite realizar un despliegue instantáneo de aplicaciones complejas, esto ocasiona que el desarrollo software pueda ser migrado entre diversos computadores con entornos completamente distintos sin necesidad de instalar dependencias, librerías o SDK (*Kit de Desarrollo de Software*) [18] adicionales.

Algunos de los conceptos fundamentales de Docker son los siguientes:

- *Docker Desktop*: aplicación de escritorio que administra Docker y permite a los usuarios utilizarlo en su entorno de desarrollo local. Cuenta con una interfaz gráfica que simplifica su uso.
- *Docker Hub*: repositorio en línea que permite a los usuarios compartir y descargar imágenes predefinidas de contenedores.
- *Dockerfile*: archivo de texto que contiene una serie de instrucciones para construir una imagen de un contenedor.
- *Docker Compose*: herramienta que permite definir y administrar aplicaciones de múltiples contenedores, este archivo incluye la definición de los servicios, las redes y los volúmenes necesarios para interconectar los contenedores.

7.2. Python

Python ha sido elegido como el lenguaje de programación principal para realizar el desarrollo del *Backend* de la aplicación. Una de las razones clave para esta elección fue mantener la consistencia con el código fuente existente de *Robomap*, que también está escrito en Python. Al utilizar el mismo lenguaje en ambos proyectos, se busca maximizar la compatibilidad e interoperabilidad entre las aplicaciones, lo que facilita la integración de funcionalidades y el intercambio de datos entre ellas.

Figura 11: Logo de Python

Además, Python es ampliamente reconocido y valorado en el ámbito de los desarrollos matemáticos y la aplicación de métodos de posicionamiento en interiores, ya que cuenta con bibliotecas y paquetes especializados en matemáticas y ciencias de datos. Algunos ejemplos notables incluyen NumPY, scikit-learn o SciPy, estas bibliotecas han simplificado de forma significativa el desarrollo y han permitido que el sistema sea altamente eficiente.

Dentro del contexto de desarrollo de Python se han utilizado las siguientes utilidades:

- PyPi (*Python Package Index*) es un repositorio de software en línea que contiene una gran colección de paquetes de dependencias y bibliotecas de Python disponibles para su descarga e instalación [19]. Otorga una fácil instalación y gestión de paquetes, esto simplifica el proceso de configuración del entorno de desarrollo y reduce la carga de trabajo relacionada con la instalación manual de paquetes y sus dependencias.

Figura 12: Logo de PyPi

- Flask es un *framework* web ligero y flexible para Python que se utiliza para elaborar aplicaciones web bajo el patrón MVC [20]. Posee una baja curva de aprendizaje y gracias a su sintaxis clara y sencilla facilita el desarrollo de aplicaciones web. Flask es una excelente opción para crear APIs (*Interfaces de Programación de Aplicaciones*) [21]. Este *framework* ofrece un sistema de enrutamiento sencillo y flexible que permite definir las URL y acciones asociadas a cada una de ellas, esto facilita el manejo de los diferentes tipos de peticiones HTTP, como *get*, *post*, *put* y *delete*.

Figura 13: Logo de Flask

- SQLAlchemy es un ORM (*Object-relational mapping*) en Python que proporciona una interfaz de programación para interactuar con bases de datos relacionales de forma sencilla y eficiente. Al utilizarse junto con Flask, SQLAlchemy ofrece numerosas ventajas: permite interactuar con bases de datos utilizando objetos y métodos en lugar de sentencias de consulta SQL; permite integrarse con una amplia variedad de motores de bases de datos como SQLite, MySQL o PostgreSQL; ofrece soporte multiconsulta en bases de datos relacionales y permite utilizar filtros, ordenamientos y otras funcionalidades para generar consultas potentes y optimizadas [22].

Figura 14: Logo de SQLAlchemy

7.3. JavaScript

Figura 15: Logo de JavaScript

JavaScript ha sido seleccionado como el lenguaje de programación principal para el desarrollo del *Frontend* de la aplicación. Las razones son evidentes: posee una amplia compatibilidad con los principales navegadores web, además de ejecutarse en prácticamente cualquier dispositivo y plataforma, lo que garantiza que sea plenamente accesible para la mayoría de los usuarios. Por otra parte, es totalmente compatible con la arquitectura *API RESTful*, utilizada en el *Backend*, esto supone que puede realizar solicitudes HTTP a través de AJAX o utilizar bibliotecas para realizar llamadas a la API y consumir respuestas en formato JSON.

Es bien conocido que Javascript destaca por su gran comunidad de usuarios y la abundancia de recursos disponibles, por lo que esto proporciona una ventaja significativa durante el desarrollo de aplicaciones. Existe una gran cantidad de soluciones y enfoques previamente definidos para resolver todo tipo de problemáticas comunes.

Dentro del contexto de desarrollo de JavaScript se han utilizado las siguientes utilidades:

Figura 16: Logo de Node.js

- Node.js es un entorno de ejecución JavaScript que permite su ejecución en el lado del servidor. Es capaz de manejar una gran cantidad de solicitudes concurrentes de forma eficiente y además que cuenta con un gestor de paquetes llamado npm (*Node Package Manager*), que es una de las mayores fortalezas de la plataforma. Esto acelera el desarrollo de aplicaciones al proporcionar soluciones preparadas para usar y fomentar la reutilización de código [23].

- Angular es un *framework* de desarrollo para JavaScript que utiliza un lenguaje de programación denominado Typescript (basado en Javascript) y también es compatible con el preprocesador de estilos Sass (*Syntactically Awesome Style Sheets*) utilizado para dar formato al contenido web [24]. Angular posee una arquitectura basada en componentes, lo que facilita la organización y reutilización del código y además propicia que el desarrollo final sea modular y mantenible.
  Uno de sus puntos fuertes es su biblioteca Angular Material, utilizada durante todo el desarrollo del proyecto. Esta cuenta con una amplia variedad de componentes reutilizables listos para incorporar en la web, como barras de navegación, diálogos o formularios avanzados.

Figura 17: Logo de Angular

- Bulma es un *framework* CSS que proporciona un conjunto de estilos y componentes predefinidos para el desarrollo de interfaces de usuario [25]. Su objetivo principal es adaptar el entorno web a una amplia variedad de tamaños de pantallas, lo que significa que las interfaces desarrolladas con Bulma se verán y funcionarán de manera óptima en diferentes tipos de dispositivos.

Figura 18: Logo de Bulma

- Integraciones con diferentes librerías externas para dotar de funcionalidad y pulir la interfaz de usuario, entre las cuales destacan: *ngx-pagination* (paginación de elementos); *ng2-charts* (control y visualización de gráficas dinámicas); *ngx-owl-carousel-o* (control y visualización de carrusel de imágenes); *ngx-toastr* (visualización de notificaciones).

7.4. SQLite

SQLite es una base de datos relacional muy ligera y versátil, utilizada en aplicaciones y proyectos de tamaño pequeño a mediano debido a su escaso consumo de recursos [26]. Dado que *Robomap* ya utilizaba una base de datos SQLite3 en su implementación, se optó por mantener esa elección por motivos de compatibilidad. Al admitir la importación directa de campañas de recogida de datos provenientes de *Robomap*, se asegura una transición más suave y una mayor compatibilidad con el código existente. Esto significa que no es necesario realizar cambios significativos en la estructura de la base de datos.

Figura 19: Logo de SQLite

Entre sus características destacan: su capacidad de manejar grandes volúmenes de datos, esto es especialmente importante cuando se trabaja con campañas que contienen decenas de miles de registros; la base de datos es embebida, por lo que es totalmente portable y no se requiere una instalación externa de un servidor de bases de datos; es completamente compatible con Python y SQLAlchemy, lo que la convierte en una opción ideal para integrarse sin problemas con la herramienta final.

7.5 Visual Studio Code

Visual Studio Code ha sido el entorno de desarrollo que se ha empleado a la hora de desarrollar el código fuente. Además de proporcionar una interfaz intuitiva y personalizable, cuenta con un ecosistema de extensiones muy variado, lo que permite ampliar sus funcionalidades y adaptarlo a diferentes lenguajes de programación [27].

Figura 20: Logo de Visual Studio Code

Posee una integración fluida con diversas utilidades, como el control de versiones (*Git*), depuración de código, múltiples tipos de terminales integradas, además de la capacidad de utilizar servicios en la nube.

7.6 Git y Github

Dentro de Visual Studio Code, se ha implementado Git como el sistema de control de versiones por defecto. Esto permite gestionar fácilmente las versiones del código y realizar un seguimiento de los cambios realizados. Además, el proceso de subida del código se realiza por medio del repositorio Github, lo que facilita la colaboración y el trabajo en equipo [28].

Figura 21: Logos de Git y Github

8. Análisis y diseño

En este apartado nos centraremos en discutir el análisis realizado en el contexto del proyecto y mostrar el diseño propuesto. Esta ha sido una etapa crucial en el desarrollo del software, ya que permite comprender y definir claramente los requisitos, objetivos y divisar las dificultades del proyecto.

8.1. Requisitos de la herramienta

1. Requisitos funcionales:
   1.1. El sistema debe permitir la subida de campañas de recogidas de datos de *Robomap*, permitiendo a los usuarios asociar archivos de configuración e imágenes a cada campaña.
   1.2. El sistema debe proporcionar la capacidad de eliminar campañas con la opción de pedir confirmación al usuario antes de eliminar definitivamente los datos asociados.
   1.3. El sistema debe permitir la extracción automática de la información de campaña desde la base de datos propietaria de *Robomap* y reflejarlos en una pantalla de visualización de detalles, junto a sus ficheros de configuración e imágenes.

1.4. El sistema debe simplificar el proceso de aplicación de métodos de posicionamiento, incorporando el método WkNN desarrollado en Matlab.

1.5. El sistema debe ser capaz de proporcionar información en tiempo real sobre el estado y progreso de la aplicación de cada método de posicionamiento.

1.6. El sistema debe almacenar los resultados de manera organizada en una base de datos, asegurando su integridad y facilitando el acceso a los usuarios.

1.7. El sistema debe permitir la aplicación dinámica de nuevos métodos de posicionamiento, incluyendo la posibilidad de ajustar los hiperparámetros correspondientes a cada método.

1.8. El sistema debe ser escalable, permitiendo la fácil incorporación de nuevos métodos de procesamiento en el futuro.

1.9. El sistema debe manejar los datos de entrada y los tipos de manera adecuada para prevenir errores de usuario al introducirlos.

1.10. El sistema debe ofrecer la capacidad de comparar y exportar gráficas dinámicas de precisión y exactitud de diferentes métodos de posicionamiento.

1.11. El sistema debe mantener las mismas configuraciones de beacons para diferentes campañas, dando la posibilidad de poder asociar en un futuro múltiples grupos de configuraciones únicas a la misma campaña.

2. Requisitos no funcionales:

2.1. El sistema debe tener un rendimiento óptimo, reduciendo significativamente los tiempos de procesamiento en comparación con el procedimiento actual.

2.2. El sistema debe ser transparente en el almacenamiento de los resultados, garantizando la integridad y trazabilidad de la información.

2.3. El sistema debe ser compatible con diferentes entornos de implementación, utilizando una arquitectura que facilite su despliegue en diferentes sistemas sin la necesidad de instalar librerías o herramientas adicionales.

3. Requisitos de usabilidad:
    3.1. El sistema debe tener una interfaz de usuario intuitiva y fácil de navegar.
    3.2. El sistema debe proporcionar retroalimentación clara y mostrar mensajes de error comprensibles para el usuario.
    3.3. El sistema debe permitir la personalización de las preferencias del usuario, como la configuración de los métodos de posicionamiento o el filtrado por parámetros en el momento de generar gráficas.
    3.4. El sistema debe contar con una documentación detallada que explique su instalación, funcionamiento y cualquier información relevante respecto al mismo.

8.2. Diagrama de casos de uso

El sistema *IPSCampaignManager* presenta el siguiente diagrama de casos de uso, el cual describe las interacciones entre los actores y el sistema (Véase la Figura 22).

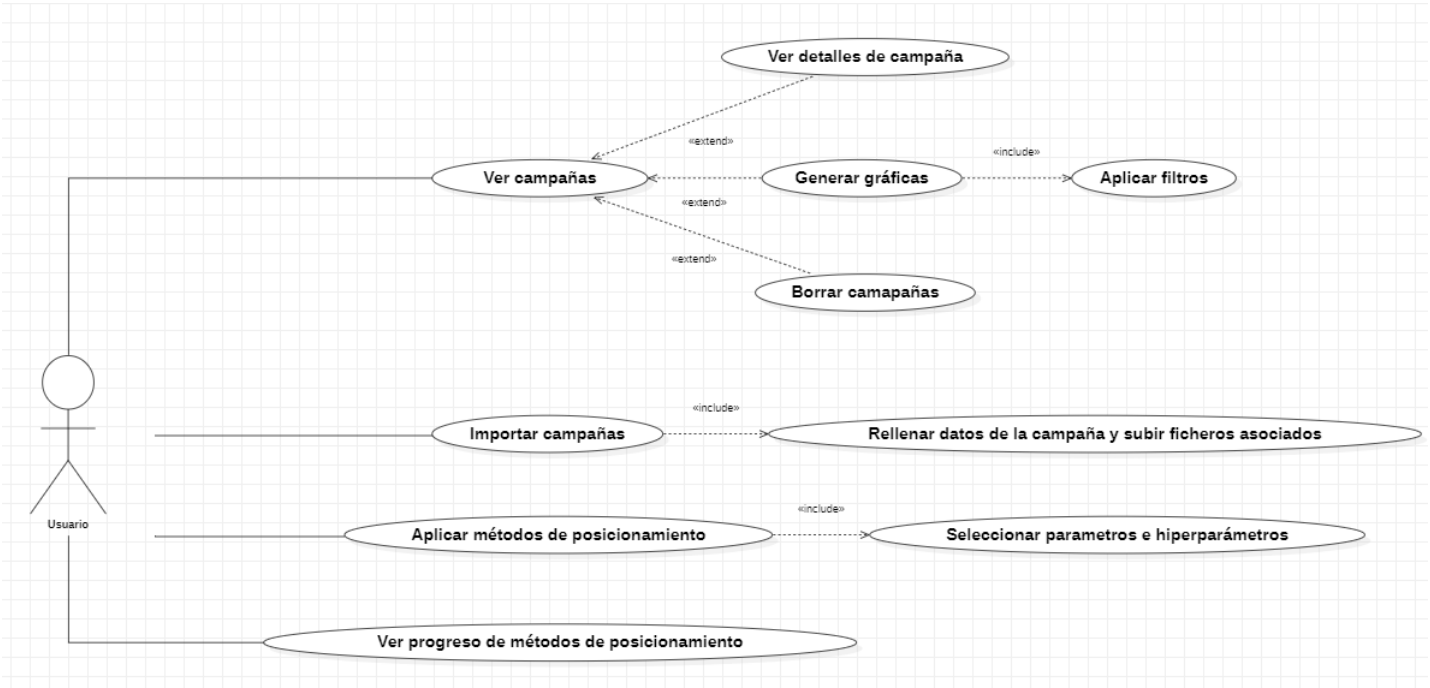

Figura 22: Diagrama de casos de uso

8.3. Modelo de la base de datos

El sistema *IPSCampaignManager* presenta el siguiente diagrama de base de datos. El modelo de la base de datos define la estructura lógica de la base de datos y establece las relaciones entre las entidades, atributos y tablas que la componen (Véase la Figura 23).

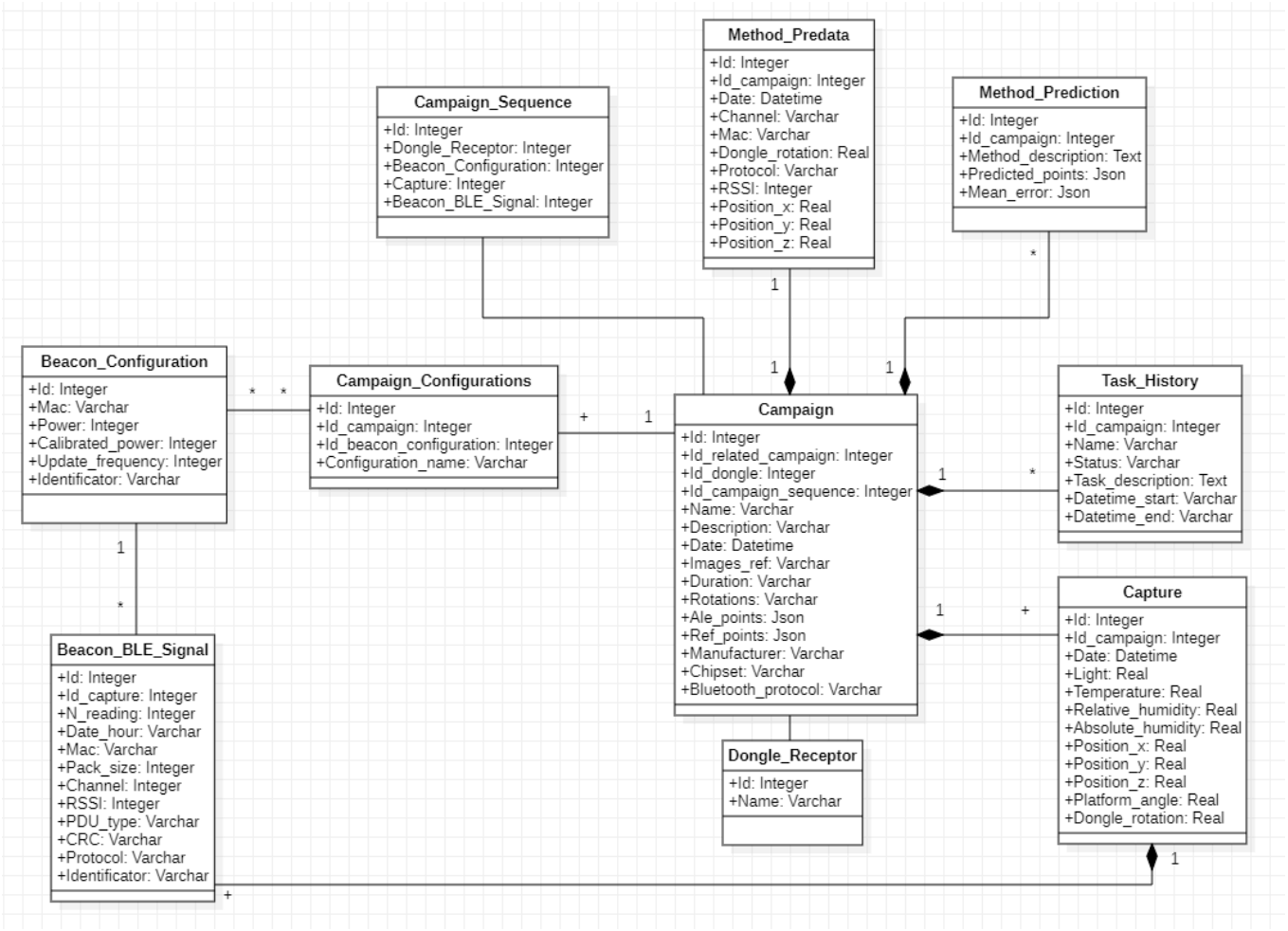

Figura 23: Diagrama de entidades de base de datos

La base de datos consta de las siguientes 10 tablas:

- *Campaign*: representa una campaña de recogida de datos.
- *Capture*: representa cada una de las capturas de datos realizadas durante una campaña.
- *Beacon_BLE_Signal*: representa cada una de las señales BLE de una captura de datos.

- *Beacon_Configuration*: representa los parámetros de configuración de una señal proveniente de un beacon. Las configuraciones son únicas y no dependen de ninguna campaña.
- *Campaign_Configurations*: representa la correspondencia entre campañas y configuraciones de beacons.
- *Task_History*: representa el histórico de tareas relacionadas con la aplicación de métodos de posicionamiento.
- *Method_Predata*: representa la preparación de los datos previa a la aplicación de los métodos de posicionamiento.
- *Method_Prediction*: representa la aplicación de las predicciones a los métodos de posicionamiento.
- *Campaign_Sequence*: representa el número de registros obtenidos de la campaña de *Robomap*.
- *Dongle_Receptor*: representa los dongles receptores encargados de capturar las señales BLE durante la campaña de recopilación de datos. Los dongles son únicos y no dependen de ninguna campaña.

Las relaciones entre las entidades de la base de datos son las siguientes:

- *Campaign-Dongle_Receptor*: cada campaña se asocia a un dongle receptor de señales BLE.
- *Campaign-Capture*: cada campaña tiene una o más capturas.
- *Campaign-Task_History*: cada campaña puede tener muchos históricos de tareas.
- *Campaign-Campaign_Configurations*: cada campaña tiene una o más correspondencias entre campaña y configuraciones de campaña.
- *Campaign-Campaign_Sequence*: cada campaña tiene una secuencia de campaña.
- *Campaign-Method_Predata*: cada campaña tiene una preparación de datos.
- *Campaign-Method_Prediction:* cada campaña puede tener muchas predicciones de métodos de posicionamiento.

- *Beacon_Configuration-Campaign_Configurations*: muchas configuraciones de beacons pueden ir asociadas a muchas configuraciones de campaña.
- *Beacon_BLE_Signal-Beacon_Configuration*: cada señal BLE de beacon tiene una configuración de beacon.
- *Capture-Beacon_BLE_Signal*: cada captura puede tener una o más señales BLE de beacon.

8.4. Diseño de la interfaz de usuario

La metodología de desarrollo por prototipos desempeñó un papel fundamental en el diseño de la interfaz de usuario, ya que permitió dar al desarrollo un enfoque colaborativo. A lo largo del proceso, se presentaron diferentes versiones de prototipos al IUCES, quienes proporcionaron retroalimentación constante.

En el proceso de diseño de la interfaz de usuario, se ha adoptado un enfoque en el que se ha prescindido de la creación de mockups y, en su lugar, se han desarrollado las vistas directamente en los prototipos. Esta metodología ha permitido una iteración ágil y continua de los diseños hasta alcanzar los resultados deseados. Podemos observar las diferentes pantallas de la aplicación web en las (Figuras 24-31).

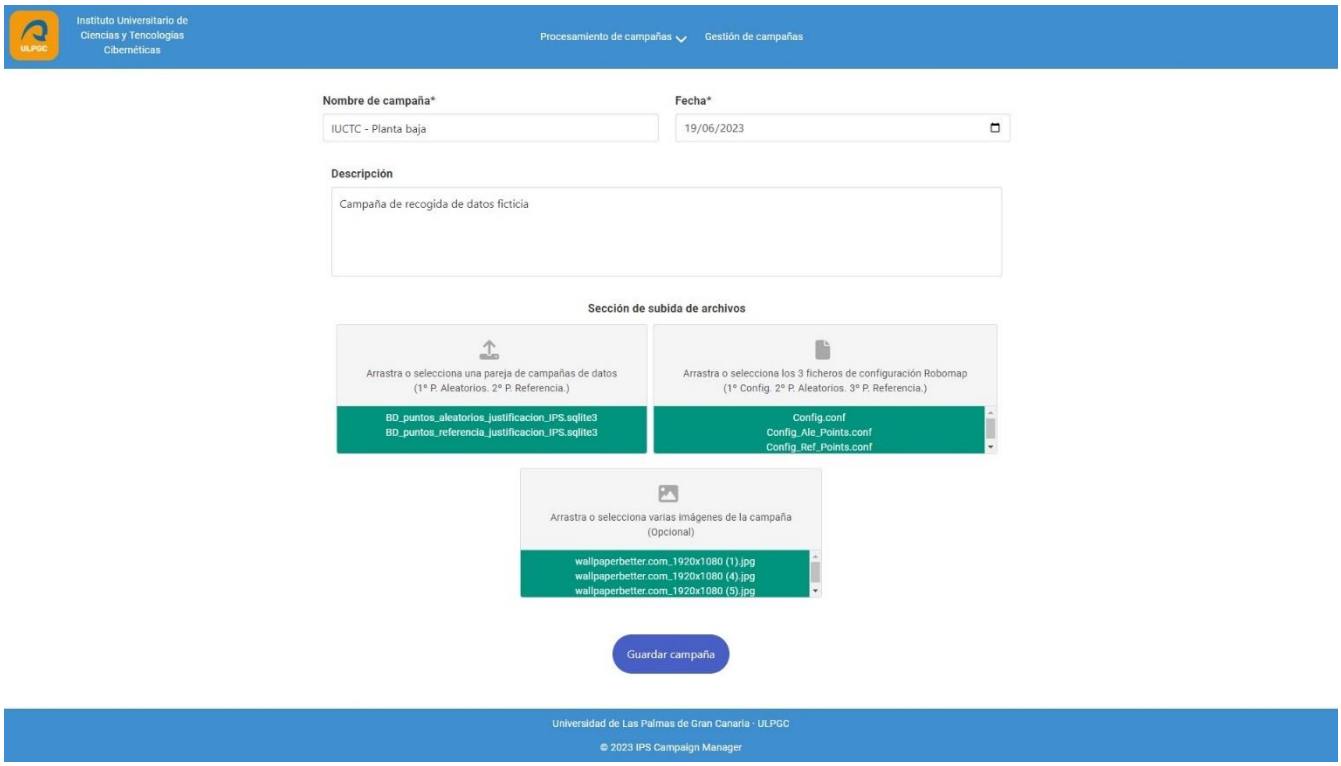

Figura 24: Pantalla de subida de campaña

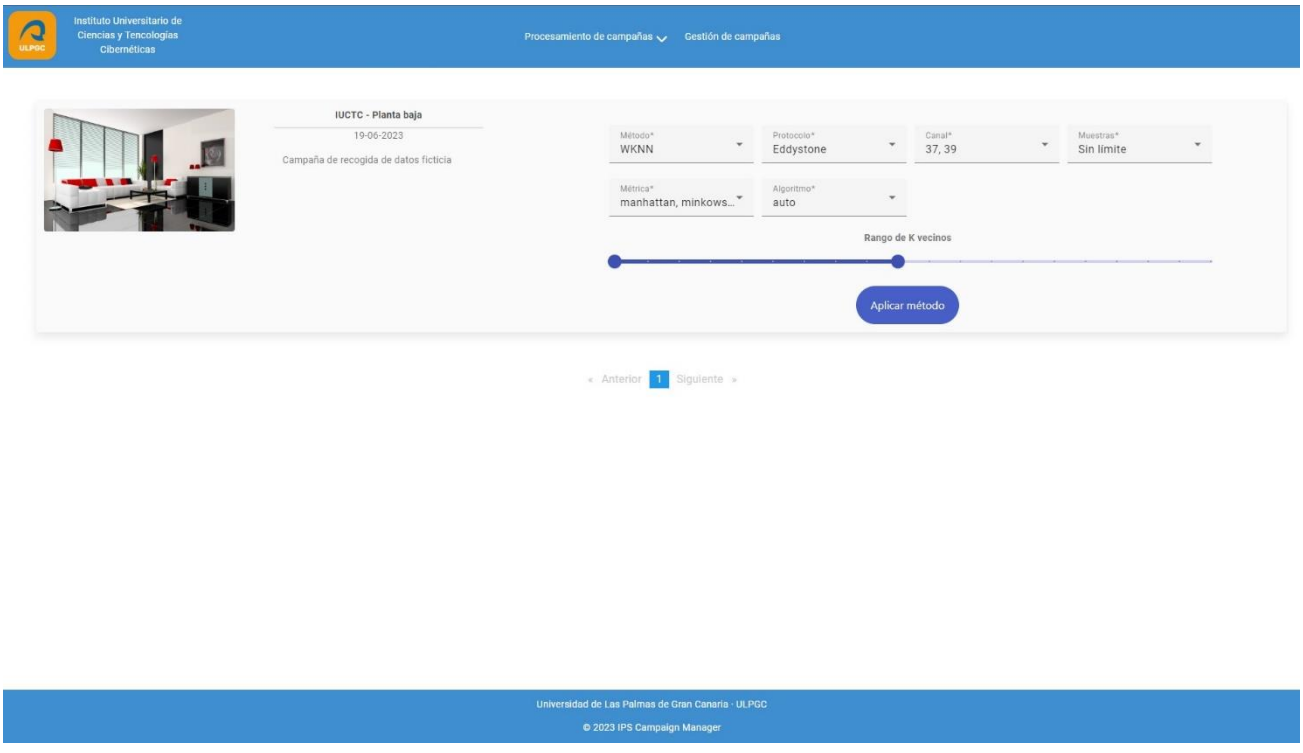

Figura 25: Pantalla de aplicación de método de posicionamiento

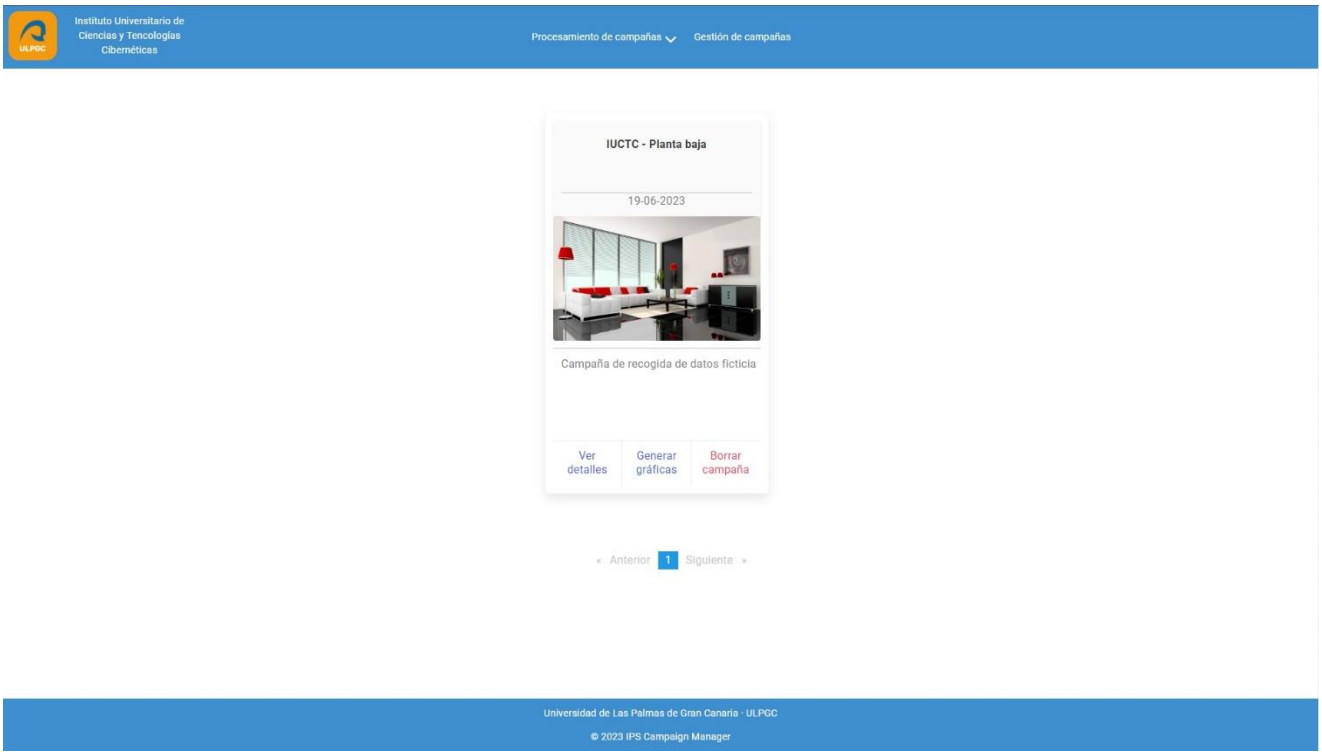

Figura 26: Pantalla de gestión de campañas

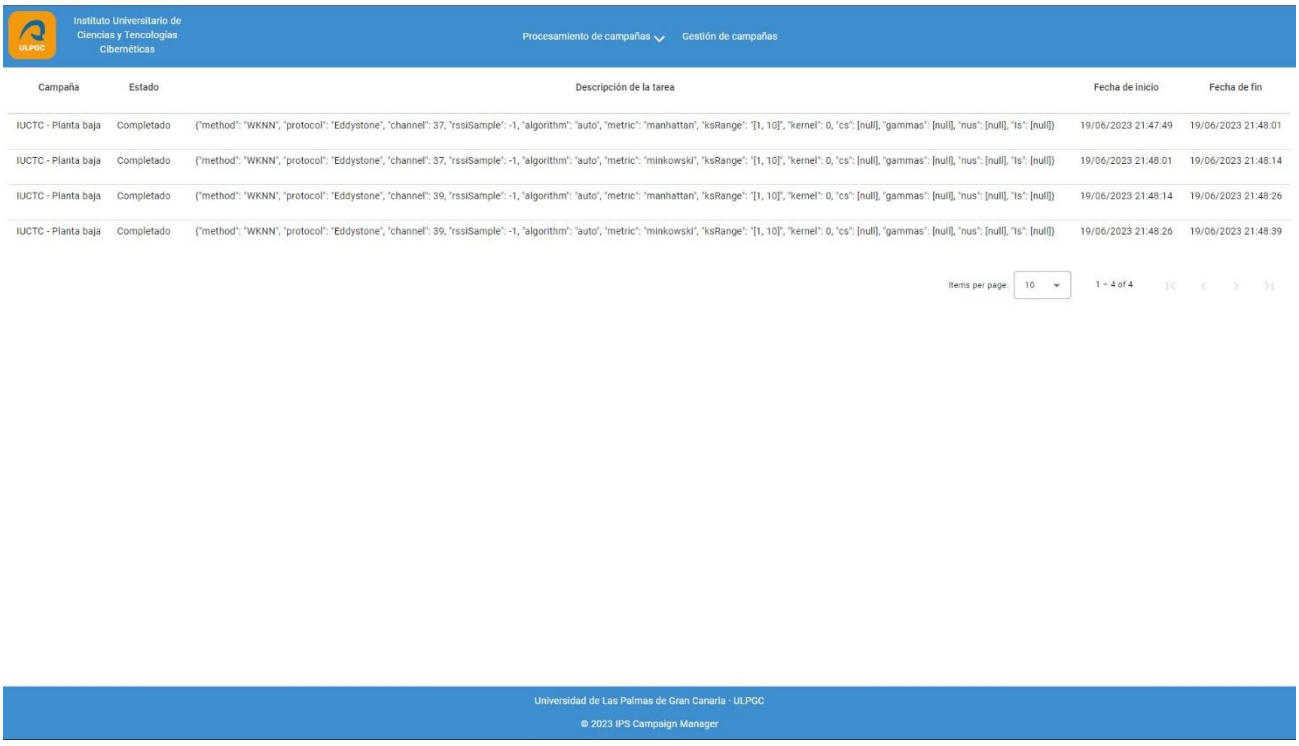

Figura 27: Pantalla de estado de procesamiento

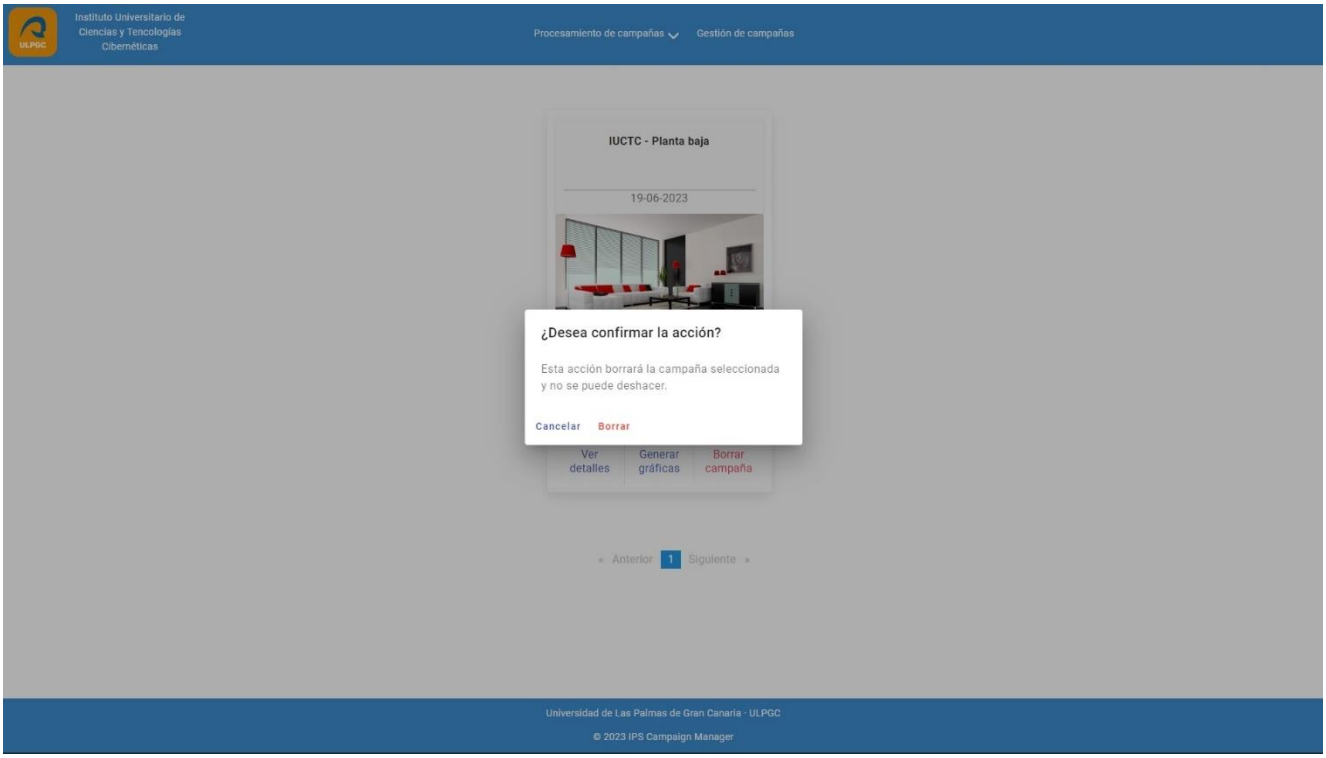

Figura 28: Diálogo de confirmación de borrado de campaña

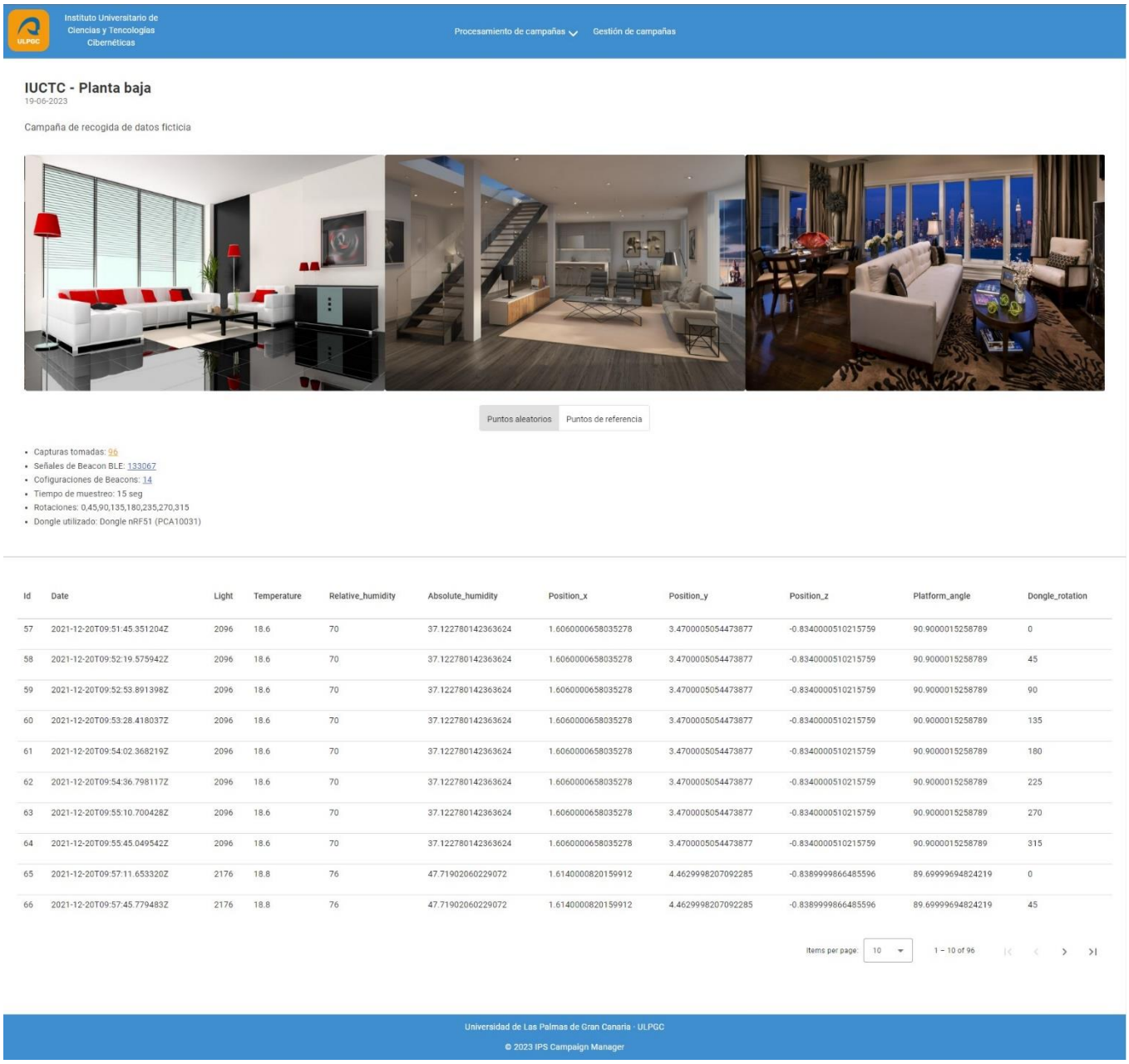

Figura 29: Pantalla de detalles de campaña

Figura 30: Pantalla de generación de gráfica de precisión

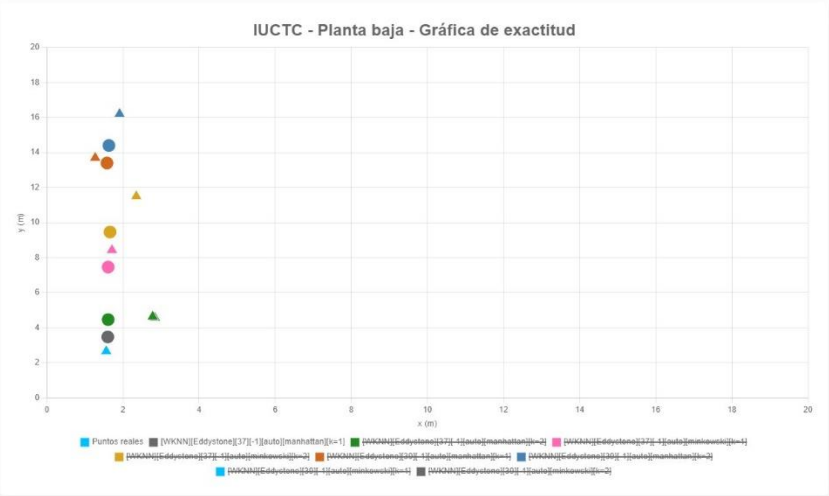

Figura 31: Gráfica de exactitud

9. Arquitectura de software

En este apartado, se va a profundizar en un aspecto fundamental del desarrollo de aplicaciones: la arquitectura del sistema. La arquitectura de software desempeña un papel de suma importancia al proporcionar una estructura sólida y bien organizada para el diseño y la implementación de sistemas de software.

9.1. Despliegue de aplicaciones dentro de contenedores en Docker

Para lograr un despliegue eficiente de contenedores en Docker, es necesario tener previamente instalado *Docker Desktop* en el entorno de desarrollo. Por otra parte, es necesario contar con un fichero *Dockerfile* que contenga la configuración específica para cada contenedor. Este fichero estará ubicado en la raíz de los proyectos, tanto para el *Backend* (Figura 32) como para el *Frontend* (Figura 33) de la aplicación.

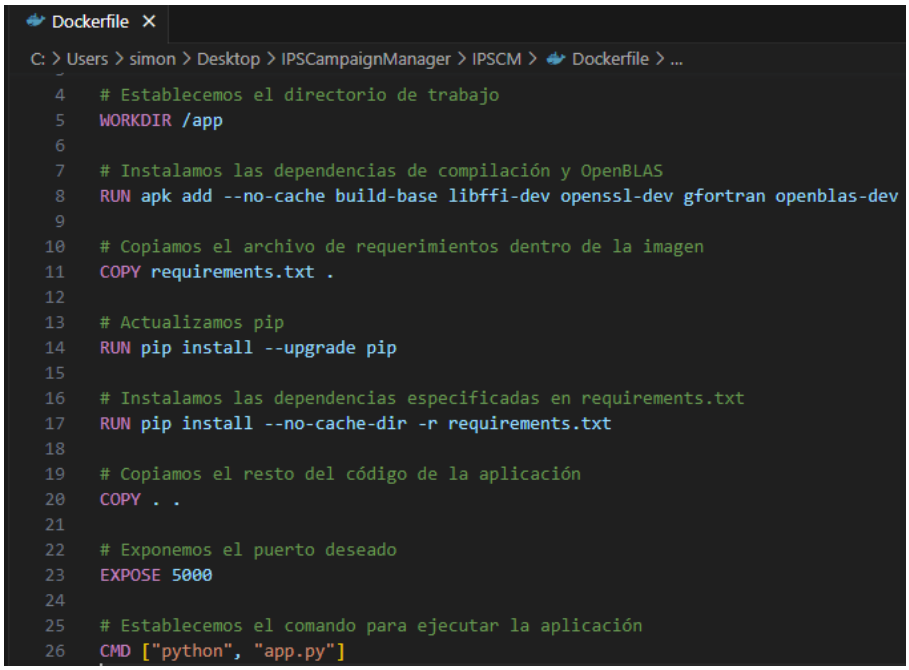

Figura 32: *DockerFile* de aplicación *Backend*

```dockerfile
# Primera Etapa: Construir la aplicación Angular
FROM node:latest as node

RUN mkdir -p /app
WORKDIR /app

# Copiar el archivo package.json e instalar dependencias
COPY package.json /app
RUN npm install

# Copiar el resto de los archivos y construir la aplicación Angular
COPY . /app

# Ajustar el presupuesto en el archivo angular.json
RUN sed -i 's/"maximumError": "1mb"/"maximumError": "2mb"/' /app/angular.json

RUN npm run build --prod

# Segunda Etapa: Configurar el servidor NGINX
FROM nginx:latest

# Copiar los archivos de la primera etapa al directorio del servidor NGINX
COPY --from=node /app/dist/ipscm.web-client /usr/share/nginx/html
```

Figura 33: *Dockerfile* de aplicación *Frontend*

Tras definir ambos ficheros *Dockerfile*, es necesario generar un nuevo fichero *Docker Compose* (Figura 34), para gestionar y desplegar ambos contenedores de forma conjunta.

```yaml
version: '1'
name: ipscm
services:
  server:
    image: simonrusu97/ipscmback
    build:
      context: ./IPSCM
      dockerfile: Dockerfile
    ports:
      - "5000:5000"

  client:
    image: simonrusu97/ipscmfront
    build:
      context: ./IPSCM.WebClient
      dockerfile: Dockerfile
    ports:
      - "4200:80"
```

Figura 34: Fichero *Docker Compose*

A continuación, se procede a ejecutar por consola el comando *"docker-compose up*" en el directorio donde se encuentre el fichero *Docker Compose* para generar las imágenes de los contenedores en *Docker Desktop* (Figura 35). Además de eso, *Docker Compose* crea y ensambla de forma automática los contenedores a partir de esas imágenes (Figura 36).

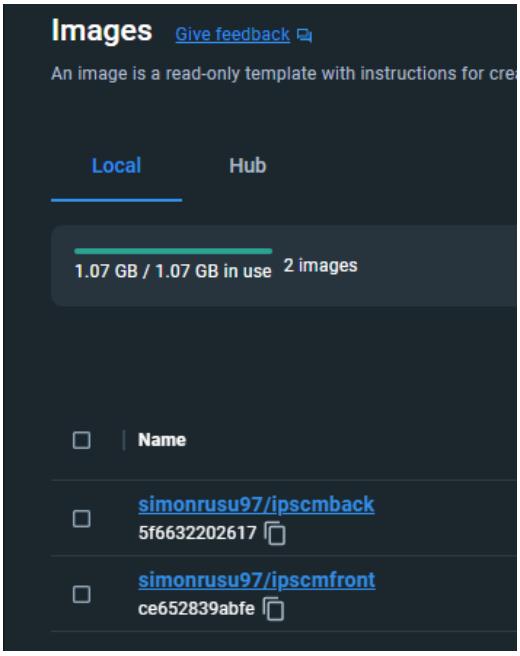

Figura 36: Imágenes generadas en *Docker Desktop*

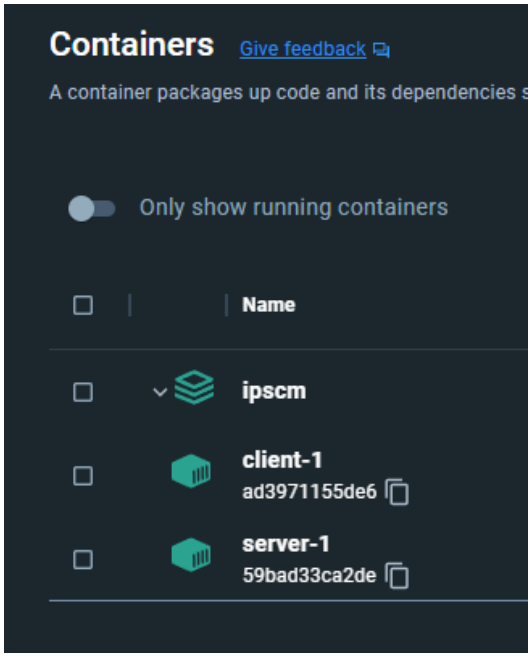

Figura 35: Contenedores generados en *Docker Desktop*

Opcionalmente, es posible subir cada una de las imágenes generadas al repositorio de imágenes *Docker Hub* a partir del menu "*Actions*" del panel de Imágenes de *Docker Desktop* (Figura 37). Esto facilita el acceso a las imágenes por parte de otros usuarios, quienes podrán descargarlas desde el repositorio y realizar un despliegue instantáneo en sus propias máquinas. Para ello hay que modificar el nombre de usuario de las imágenes en el fichero *docker-compose.yaml* por el que se utilice en *Docker Desktop*.

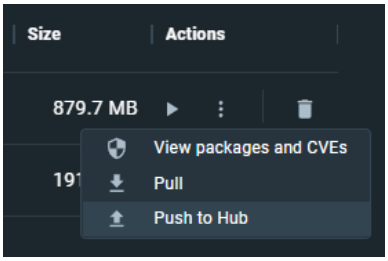

Figura 37: Subida de imagen a *Docker Hub*

9.2. Diseño arquitectónico del Backend

Para el diseño del *Backend*, se ha utilizado una arquitectura basada en Web API adaptada a las necesidades específicas del proyecto (Figura 38). El principal propósito del *Backend* es proporcionar datos al *Frontend* y permitir la comunicación entre los componentes del sistema.

En esta arquitectura podemos definir los siguientes elementos:

- *Models*: proporcionan la estructura de las entidades en la base de datos. Cada modelo representa una tabla en la base de datos y define los campos y relaciones entre ellos. Los modelos cuentan con una función de serializado que devuelve un diccionario con la representación serializada de sus datos, esto permite convertir los objetos a formato JSON para su posterior envío a través de la API.
- *Controllers*: son responsables de proporcionar funcionalidad a los modelos definidos. Su objetivo principal es facilitar la manipulación y gestión de los datos a través de SQLAlchemy, Estos permiten realizar diversas operaciones, como consulta, creación, actualización y eliminación de datos.
- *Services*: desempeñan un papel importante en el sistema al proporcionar soporte adicional para realizar operaciones de procesamiento de datos e interacción entre diferentes controladores. Estos servicios complementan la funcionalidad de los controladores y permiten realizar tareas más complejas y especializadas.
- Fichero *routes.py*: contiene la definición de los recursos utilizados por *Flask-Restful* para gestionar las llamadas a las funciones que forman parte de la API. Cada recurso representa una ruta y define los métodos HTTP que pueden ser utilizados para interactuar con la API.
- Fichero *config.py*: es responsable de la configuración de la aplicación y la conexión con la base de datos SQLite. También se encarga de generar la estructura de directorios necesaria para almacenar las imágenes comprimidas utilizadas en la aplicación.

- Fichero *app.py*: es el punto de entrada principal de la aplicación. Además de su inicialización, se encarga de agregar los recursos de la API, proporcionando rutas para cada uno de ellos.
- Directorio *db*: almacena la base de datos "*IPSCM.sqlite3"*, utilizada por la aplicación. Además, dentro de este directorio, se encuentra la carpeta "*campaign_images*", la cual se utiliza para almacenar los archivos comprimidos de las imágenes correspondientes a cada campaña.

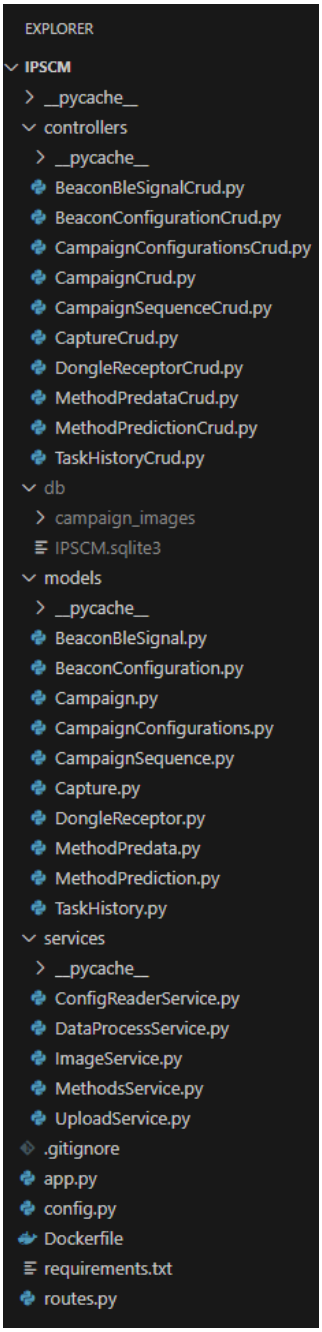

Figura 38: Diseño arquitectónico del *Backend*

9.3. Diseño arquitectónico del Frontend

El *Frontend* Angular ofrece una arquitectura basada en componentes reutilizables que permiten desarrollar interfaces de usuario de forma eficiente y modular (Figura 39). También cuenta con la capacidad de implementar enrutamientos eficientes entre diversas vistas y componentes, lo que facilita la navegación dentro de la aplicación y permite la creación de servicios para ofrecer funcionalidades específicas según los requisitos del proyecto.

La aplicación se compone de los siguientes elementos:

- Directorio *src/app*: es el directorio principal donde se almacenan los archivos principales de configuración y todos los componentes creados para la aplicación.
    - *header*: contiene el componente del encabezado de la aplicación.
    - *footer*: contiene el componente del pie de página de la aplicación.
    - *campaigns*: contiene el componente de la vista de campañas.
    - *campaign-card*: contiene el componente que representa cada elemento de campaña.
    - *campaing-details*: contiene el componente de la vista de detalles de campaña.
    - *confirmation-dialog*: contiene el dialogo de confirmación de borrado de campaña.
    - *ips-methods*: contiene el componente de la vista de aplicación de métodos IPS.
    - *ips-campaign-card*: contiene el componente que representa cada elemento de campaña a la cual se aplicará el método IPS.
    - *task-history*: contiene el componente de la vista del histórico de tareas.
    - *upload-campaign*: contiene el componente de la vista de subida de campaña.

- *graphics*: contiene el componente de la vista de generación de gráficas.
- *services*: contiene el servicio "*api.service.ts"* utilizado para la comunicación con el *Backend*.
- *app-routing.module.ts*: define las rutas y la configuración de enrutamiento de la aplicación. Se utiliza para establecer las rutas y vincularlas con los componentes correspondientes.
- *app.component.html*: contiene la plantilla HTML asociada al componente raíz de la aplicación.
- *app.component.sass*: contiene los estilos CSS del componente raíz de la aplicación.
- *app.component.spec.ts*: contiene las pruebas unitarias para el componente raíz de la aplicación.
- *app.component.ts*: contiene la lógica y el comportamiento asociados al componente principal de la aplicación.
- *app.module.ts*: es el punto de entrada principal de Angular. Se importan, configuran y definen todos los componentes, servicios, directivas y otros módulos utilizados en la aplicación.
- *env.ts*: define las variables de entorno y configuraciones de la aplicación.

- Directorio src/assets: se almacenan los recursos estáticos, como fuentes "*fonts*", imágenes "*img*" y estilos "*styles*".
- Fichero *src/favicon.ico:* icono de la aplicación que se muestra en la pestaña del navegador.
- Fichero *src/index.html*: define la estructura básica de la página y actúa como la entrada inicial para la aplicación.
- Fichero *src/main.ts*: archivo responsable de arrancar la aplicación mediante la inicialización del módulo principal de la aplicación (*AppModule*).
- Fichero *src/styles.sass*: en este archivo se definen los estilos globales que afectan a toda la aplicación.

- Fichero *angular.json*: fichero de configuración principal de Angular. Contiene información sobre la estructura del proyecto, configuraciones de generación de código, opciones de compilación, etc.
- Fichero *tsconfig.json*: archivo de configuración del compilador de TypeScript. Proporciona instrucciones al compilador sobre como compilar el código Typescript en JavaScript válido.
- Directorio *node_modules*: contiene todas las dependencias, módulos de Node.js y paquetes de terceros necesarios para la construcción y ejecución de la aplicación Angular. Los paquetes se instalan por la consola de comandos, mediante "*npm install*".

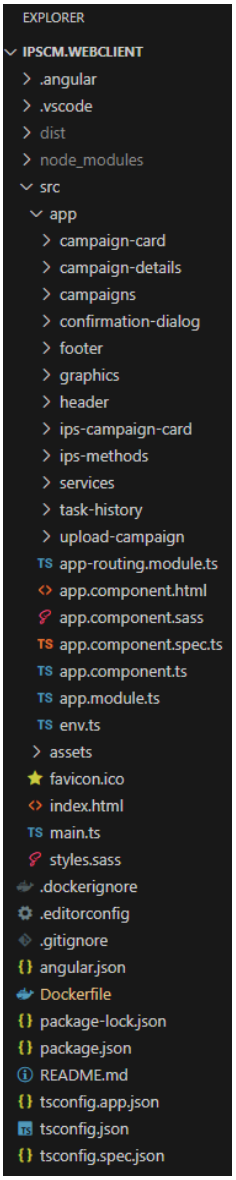

Figura 39: Diseño arquitectónico del *Frontend*

10. Implementación y funcionalidades

En este capítulo, se explicará en profundidad la implementación de las funcionalidades del sistema, otorgando especial atención a la comunicación y el intercambio de información entre el *Backend* y el *Frontend*.

10.1. Importación de campañas

Al importar una campaña a través de la interfaz de usuario, se accede al componente del *Frontend* denominado *upload-campaign*. Este componente muestra un formulario en el que se deben completar varios campos, entre los cuales se encuentran: nombre de campaña, fecha, descripción, archivos de bases de datos, ficheros de configuración e imágenes de campaña.

Al subir la campaña, los datos se encapsulan y se envían al *Backend* (Figura 40) por medio de la función correspondiente de la API (Figura 41).

```
uploadCampaign(): void {
  const formData = new FormData();

  formData.append('name', this.uploadForm.get('name')?.value);
  formData.append('date', this.uploadForm.get('date')?.value);
  formData.append('description', this.uploadForm.get('description')?.value);

  if (this.selectedImages) {
    for (let i = 0; i < this.selectedImages.length; i++) {
      formData.append('images', this.selectedImages[i]);
    }
  }

  if (this.firstSelectedFile && this.secondSelectedFile) {

    formData.append('files', this.firstSelectedFile);
    formData.append('files', this.secondSelectedFile);
  }

  if (this.selectedConfFiles){
    for (let i = 0; i < this.selectedConfFiles.length; i++) {
      formData.append('confs', this.selectedConfFiles[i]);
    }
  }
  this.router.navigate(['/campaigns']);
  this.toastr.warning('Conectando con el servidor...', 'Operación en curso');
  this.apiService.uploadCampaign(formData).pipe(
    tap(response => {
      setTimeout(() => {
        this.toastr.clear();
        this.toastr.success('¡La campaña se ha guardado correctamente!', 'Operación completada', { timeOut: 2000 });
        this.router.navigateByUrl('/', { skipLocationChange: true }).then(() => {
          this.router.navigate(['/campaigns'])
        });
      }, 0);
    }),
    catchError(error => {
      setTimeout(() => {
        this.toastr.clear();
        this.toastr.error('Ocurrió un error al cargar la campaña', 'Operación no completada', { timeOut: 2000 });
      }, 0);
      return of(null);
```

Figura 40: Encapsulamiento y envío de datos al *Backend*.

```typescript
uploadCampaign(formData: FormData): Observable<any> {
  return this.http.post(this.baseUrl + 'uploadCampaign', formData);
}
```

Figura 41: Función *uploadCampaign()*

Tras realizar la llamada a la función de tipo *post uploadCampaign* se mandan los datos al *Backend* y se procede con su procesamiento (Figura 42).

```python
class UploadCampaign(Resource):
    def post(self):
        name = request.form.get('name')
        date = request.form.get('date')
        imagesRef = None

        description = request.form.get('description')
        if description=='null':
            description = "No hay descripción"

        if 'images' in request.files:
            images = request.files.getlist('images')
            imagesRef = generateZipImages(images)

        files = request.files.getlist('files')
        confs = request.files.getlist('confs')

        return uploadCampaign(name, date, description, imagesRef, files, confs)
```

Figura 42: Función *UploadCampaign()*

En el caso de las imágenes, se realiza una llamada a la función *generateZipImages()* (Figura 42) del servicio *ImageService.py* y se guarda la ruta del comprimido.

```python
def generateZipImages(images):
    ranFolder = str(uuid4())
    zipRef = ranFolder + '.zip'
    imagesFolder = os.path.join(pathlib.Path(__file__).parent.parent.resolve(), imagesDir, ranFolder)
    os.makedirs(imagesFolder)

    for i in images:
        i.save(os.path.join(imagesFolder, secure_filename(str(uuid4()) + os.path.splitext(i.filename)[1])))

    zip_filename = f"{imagesDir}/{ranFolder}.zip"
    with zipfile.ZipFile(zip_filename, "w") as zip:
        for image_name in os.listdir(imagesFolder):
            zip.write(os.path.join(imagesFolder, image_name), arcname=image_name)

    shutil.rmtree(os.path.join(imagesDir, ranFolder))
    return zipRef
```

Figura 43: Función *generateZipImages()*

A continuación, se realiza una llamada a la función *uploadCampign()* perteneciente al servicio de subida de campaña *UploadService.py*. Esta función realiza múltiples llamadas a los servicios y a los controladores de las diferentes entidades involucradas en la generación de una campaña (Figura 44).

```
def uploadCampaign(name, date, description, imagesRef, files, confs):

    relatedCampaignId = None
    images = None

    filename = secure_filename("auxDB.sqlite3")

    campaignParams = readBLEConf(confs[0])
    alePointsJson = readAlePointsConf(confs[1])
    refPointsJson = readRefPointsConf(confs[2])

    for i,data in enumerate(files):
        files[i].save(os.path.join('db', filename))
        lastBeaconConfName = getLastInsertedBeaconConfName()

        if(i == 1):
            relatedCampaignId = getLastInsertedCampaignId()
            images = imagesRef

        createCampaignSequence()
        createDongleReceptor()
        createCampaign(name, date, description, images, relatedCampaignId, campaignParams, alePointsJson, refPointsJson)

        lastCampaignId = getLastInsertedCampaignId()
        beaconConfName = getNextBeaconConfName(lastCampaignId, lastBeaconConfName)

        createBeaconBleSignal()
        createCapture()

        createBeaconConfiguration(lastCampaignId, beaconConfName)
        generatePredata(lastCampaignId)

        os.remove('db/auxDB.sqlite3')
```

Figura 44: Función *uploadCampaign()*

Cuando se genera una nueva campaña, se generan dos conexiones de base de datos (Figura 45). Estas conexiones son utilizadas para interactuar con las bases de datos relacionadas con la gestión de las campañas.

La primera instancia genera una base de datos auxiliar *"auxDB.sqlite3"* para cada campaña de nueva importación. Esta conexión se utiliza para acceder y realizar consultas sobre las bases de datos de la campaña importada.

La segunda instancia de conexión hace referencia a la conexión con la base de datos principal de *IPSCampaignManager*. Esta conexión se utiliza para transferir los datos de la primera instancia, es decir, los datos de la campaña importada, a la base de datos principal.

```
app.config["SQLALCHEMY_DATABASE_URI"] = f"sqlite:///{basedir / 'db' / 'IPSCM.sqlite3'}"
app.config["SQLALCHEMY_BINDS"] = {
    "auxDB": f"sqlite:///{basedir / 'db' /  'auxDB.sqlite3'}"
}
```

Figura 45: Instancias de conexión de bases de datos

El siguiente paso es obtener los ficheros de configuración que han sido incluidos en la subida de campaña, extraer su información e incluirla en la campaña de datos.

Esto se consigue por medio del servicio *ConfigReaderService.py*, este servicio consta de tres funciones: *readBLEConf()* (Figura 46) , *readAlePointsConf()* (Figura 47) y *readRefPointsConf()* (Figura 48).

```
def readBLEConf(file):
    values = []

    content = file.read()
    content = content.decode('utf-8')
    config = configparser.ConfigParser()
    config.read_string(content)
    duration = config.get('BLE', 'Duration')
    rotations = config.get('Orientator', 'Rotations')

    values.append(duration)
    values.append(rotations)

    return values
```

Figura 47: Función de lectura de Configuración BLE

```
def readAlePointsConf(file):
    content = file.read()
    content = content.decode('utf-8')
    config = configparser.ConfigParser()
    config.read_string(content)

    points = {}
    for section in config.sections():
        if section.startswith('POINT_'):
            point = {}
            point['X'] = config.get(section, 'X')
            point['Y'] = config.get(section, 'Y')
            point['Z'] = config.get(section, 'Z')
            point['Rotation'] = config.getfloat(section, 'Rotation')
            points[section] = point

    return json.dumps(points)
```

Figura 46: Función de lectura de puntos aleatorios

```python
def readRefPointsConf(file):
    content = file.read()
    content = content.decode('utf-8')
    config = configparser.ConfigParser()
    config.read_string(content)

    points = {}
    for section in config.sections():
        if section.startswith('POINT_'):
            point = {}
            point['X'] = config.get(section, 'X')
            point['Y'] = config.get(section, 'Y')
            point['Z'] = config.get(section, 'Z')
            rotations = config.get(section, 'Rotations')
            point['Rotations'] = [float(rotation) for rotation in rotations.split(',')]
            points[section] = point

    return json.dumps(points)
```

Figura 48: Función de lectura de puntos de referencia

Una vez que se han obtenido todos los datos necesarios, el siguiente paso es recorrer cada uno de los archivos importados. Es importante tener en cuenta que el orden de subida de archivos desde la interfaz de usuario debe ser respetado.

El primer archivo que se recorre corresponde a la base de datos de puntos aleatorios, mientras que el segundo archivo se refiere a la base de datos de puntos de referencia.

Cada par de bases de datos está asociado a través del campo *relatedCampaignId* (Figura 44). Durante la segunda iteración del proceso, se obtiene la campaña previamente registrada y se guarda su identificador de campaña *campaignId*. Además, en esta iteración se guarda la referencia a las imágenes asociadas a la campaña. Es crucial respetar el orden de generación de entidades, ya que existen múltiples dependencias entre todas las entidades involucradas en el proceso.

Adicionalmente, según figura en los requisitos funcionales del sistema, se ha generado una función que permitirá asociar diferentes grupos de configuraciones de beacons a una misma campaña. En la actualidad, *Robomap* no es capaz de generar campañas con múltiples configuraciones y reflejarlas en los archivos de base de datos que se obtienen tras la campaña de datos.

Por lo tanto, se ha definido la función *getNextBeaconConfName()* (Figura 49), esta función pretende diferenciar las configuraciones de campaña por su denominación.

El formato del nombre de configuración seleccionado es *"___Configuración__XX"*, este enfoque es susceptible de cambiar en futuras iteraciones de la herramienta.

```python
def getNextBeaconConfName(lastCampaignId, lastBeaconConfName):
    if lastBeaconConfName is not None:
        beaconConf = lastBeaconConfName.split('_')[:-1]
        try:
            index = int(lastBeaconConfName.split('_')[-1])
            index += 1
            if index >= 10:
                beaconConf.append(str(index))
            else:
                beaconConf.append('0' + str(index))
        except:
            index = lastCampaignId
            beaconConf.append('0' + str(index))

        return '_'.join(beaconConf)
    else:
        return "___Configuración___00"
```

Figura 49: Función *getNextBeaconConfName()*

La última entidad en generarse es *MethodPredata* mediante el método *generatePredata()* del servicio *DataProcessService.py* (Figura 50). La función de esta entidad es reorganizar los datos obtenidos de las diferentes entidades de las campañas para darles un formato óptimo, lo que resulta en una reducción considerable en el tiempo necesario para aplicar los métodos de procesamiento.

```python
def generatePredata(campaignId):
    captureIds = getCaptureIdsByCampaignId(campaignId)
    filteredCapture = getCapturesById(captureIds)

    batch_data = []

    for capture in filteredCapture:
        filteredSignals = getBeaconBleSignalById(capture.Id)
        for signal in filteredSignals:
            batch_data.append((campaignId, capture.Date, signal.Channel, signal.Mac, capture.Dongle_rotation, signal.Protocol,
                               signal.RSSI, capture.Position_x, capture.Position_y, capture.Position_z))

    createMethodPredataBatch(batch_data)
```

Figura 50: Función *generatePredata()*

La función *generatePredata()* filtra la entidad *Capture* por su identificador. A partir de estas consultas, se obtienen las señales BLE asociadas a cada captura. El propósito final de esta función es combinar y dar de alta

conjuntamente las capturas y las señales en una única entidad. Dado que puede haber una gran cantidad de registros a dar de alta, se utiliza un enfoque de alta por lotes para mejorar la eficiencia del proceso.

Para finalizar el proceso de subida de la campaña, se realiza la eliminación de los archivos de bases de datos auxiliares generados durante el proceso, específicamente los archivos llamados "*auxDB.sqlite3*".

10.2. Aplicación de métodos IPS

A la hora de aplicar un método IPS a través de la interfaz de usuario, se accede al componente del *Frontend* denominado *ips-campaign-card.*

En el componente, el usuario realiza el filtrado de parámetros e hiperparámetros mediante las opciones dinámicas de selección. En la Figura 51 podemos observar las variables utilizadas en el componente.

```
export class IpsCampaignCardComponent {
  @Input() campaign!: any;
  campaignImages: string[] = [];
  methods: string[] = ["WKNN", "SVR", "NuSVR", "LinearSVR"];
  protocols: string[] = ["Eddystone", "iBeacon"];
  channels: string[] = ["37", "38", "39"];
  samples = Array.from({length: 20}, (_, index) => index + 1);
  selectAllProtocolsCheck: boolean = false;
  selectAllChannelsCheck: boolean = false;
  selectAllMetricsCheck: boolean = false;
  selectAllAlgorithmsCheck: boolean = false;

  metricParams: string[] = ["manhattan", "chebyshev", "minkowski"]
  algorithmParams: string[] = ["brute", "ball_tree", "kd_tree", "auto"]
  kernelParams: string[] = ["linear", "poly", "sigmoid", "rbf"]
  gammaParams: string[] = ["auto", "scale", "range"]
  WKNNparams: boolean = false;
  SVRparams: boolean = false;
  NuSVRparams: boolean = false;
  LinearSVRparams: boolean = false;
  SVRgammaRange: boolean = false;
  NuSVRgammaRange: boolean = false;


  selectedMethod: any;
  form!: FormGroup;
```

Figura 51: Declaración e inicialización de variables en el componente *ips-campaign-card*

Al seleccionar un determinado método, se aplican los validadores de formulario correspondientes según los hiperparámetros de cada método. Para ello se hace uso de la función *onMethodSelected()* (Figura 52).

```
onMethodSelected(method: string){
    this.selectedMethod = method;
    this.validateAndClearFields();
    this.WKNNparams = false;
    this.SVRparams = false;
    this.NuSVRparams = false;
    this.LinearSVRparams = false;
    this.SVRgammaRange = false;
    this.NuSVRgammaRange = false;

    switch(this.selectedMethod){
      case "WKNN":
        this.WKNNparams = true;
        this.form.get('cStep')?.setValidators(null);
        this.form.get('nuStep')?.setValidators(null);
        this.form.get('iStep')?.setValidators(null);
        this.form.get('gStep')?.setValidators(null);
        this.form.get('gamma')?.setValidators(null);
        this.form.get('kernels')?.setValidators(null);
        break;
      case "SVR":
        this.SVRparams = true;
        this.form.get('algorithms')?.setValidators(null);
        this.form.get('metrics')?.setValidators(null);
        this.form.get('iStep')?.setValidators(null);
        this.form.get('nuStep')?.setValidators(null);
        this.form.get('gStep')?.setValidators(null);
        break;
      case "NuSVR":
        this.NuSVRparams = true;
        this.form.get('algorithms')?.setValidators(null);
        this.form.get('metrics')?.setValidators(null);
        this.form.get('iStep')?.setValidators(null);
        this.form.get('gStep')?.setValidators(null);
        break;
      case "LinearSVR":
        this.LinearSVRparams = true;
        this.form.get('algorithms')?.setValidators(null);
        this.form.get('metrics')?.setValidators(null);
        this.form.get('gStep')?.setValidators(null);
        this.form.get('gamma')?.setValidators(null);
        this.form.get('nuStep')?.setValidators(null);
        this.form.get('kernels')?.setValidators(null);
        break;
    }
    Object.keys(this.form?.controls).forEach(key => {
      this.form?.get(key)?.updateValueAndValidity();
```

Figura 52: Validación de formulario del método seleccionado

Una vez se hayan elegido los parámetros e hiperparámetros deseados se selecciona el boton *"Aplicar método"* y la acción desencadena la llamada a la función *applyMethodBtn()*. Esta función empaqueta el método seleccionado junto a sus hiperparámetros (Figura 53) para posteriormente enviarlos al *Backend* (Figura 54) por medio de la llamada de la función correspondiente del Servicio de la API (Figura 55).

```
switch(this.selectedMethod){

  case "WKNN":
      metric = this.form.get('metrics')?.value;
      algorithm = this.form.get('algorithms')?.value;
      kRangeStart = this.form.get('kRangeStart')?.value;
      kRangeEnd = this.form.get('kRangeEnd')?.value;

      dataPackage['metrics'] = metric;
      dataPackage['algorithms'] = algorithm;
      dataPackage['kRangeStart'] = kRangeStart;
      dataPackage['kRangeEnd'] = kRangeEnd;
      break;
```

Figura 53: Empaquetamiento de datos de la función *applyMethodBtn()*

```
formData.append('params', JSON.stringify(dataPackage));

this.toastr.clear();
this.toastr.warning('Conectando con el servidor...', 'Operación en curso', { timeOut: 2000 });

this.apiService.applyMethod(formData).pipe(
  catchError(error => {
    setTimeout(() => {
      this.toastr.clear();
      this.toastr.error('Ocurrió un error durante la aplicación del método', 'Operación no completada', { timeOut: 2000 });
    }, 0);
    return of(null);
  })
).subscribe();
```

Figura 54: Envío de datos al *Backend*

```
applyMethod(formData: FormData): Observable<any> {
  return this.http.post(this.baseUrl + 'applyMethod', formData);
}
```

Figura 55: Función *applyMethod()*

En la parte del *Backend*, los datos se reciben por medio de la función de tipo *post DataProcessing()* (Figura 56). En esta función, los parámetros, que llegan en formato JSON se vuelven a convertir en objetos.

```
class DataProcessing(Resource):
    def post(self):
        params = request.form.get('params')
        data =  json.loads(params)
        return dataProcessing(data)
```

Figura 56: Función *DataProcessing()*

Esta función llama a su vez a la función *dataProcessing()* del servicio *DataProcessService.py* para proceder con el procesamiento del método (Figura 57). En primer lugar, se desempaquetan los valores recibidos de manera que puedan ser utilizados adecuadamente en el procesamiento. A continuación, se obtienen los puntos aleatorios y de referencia de la campaña en la que se va a ejecutar el procesamiento y por último se obtienen las coordenadas únicas asociadas a dichos puntos (Figura 58).

```
def dataProcessing(data):
    campaignId = data['campaign']
    campaignName = data['campaignName']
    method = data['method']
    protocols = data['protocols']
    channels = data['channels']
    rssiSamples = data['sample']
    metrics = data['metrics']
    algorithms = data['algorithms']
    kernels = data['kernels']
    cs = data['cValues']
    gammas = data['gValues']
    nus = data['nuValues']
    Is = data['iValues']
    ksRange = []
    ksRange.append(data['kRangeStart'])

    configPoints = getPointsByCampaignId(campaignId)

    ref_points_conf = configPoints['Ref_points']
    ale_points_conf = configPoints['Ale_points']

    ref_points = getUniqueCoordinatesByCampaignId(campaignId, len(ref_points_conf))
    ale_points = getUniqueCoordinatesByCampaignId(campaignId-1, len(ale_points_conf))

    if(len(ref_points)) < data['kRangeEnd']:
        ksRange.append(len(ref_points))
    else:
        ksRange.append(data['kRangeEnd'])

    ksRangeJson = json.dumps(ksRange)
```

Figura 57: Función *dataProcessing()*: parte 1

```python
def getUniqueCoordinatesByCampaignId(campaignId, limit):
    unique_coordinates = db.session.query(
        MethodPredata.Position_x,
        MethodPredata.Position_y,
        MethodPredata.Position_z
    ).filter(MethodPredata.Id_campaign == campaignId).distinct().limit(limit).all()

    unique_coordinates = [
        {'x': x, 'y': y, 'z': z}
        for x, y, z in unique_coordinates
    ]

    return unique_coordinates
```

Figura 58: Función *getUniqueCoordinatesByCampaignId()*

Dado que las campañas se encuentran de forma correlativa, se puede establecer que el id *campaignId-1* hace referencia a la campaña anterior, la cual contiene los puntos aleatorios necesarios para el procesamiento. Por otro lado, la campaña actual con el *campaignId* correspondiente contiene los puntos de referencia.

Retomando la función *dataProcessing()*, una vez obtenidos todos los parámetros necesarios se procede a recorrer cada uno de ellos para aplicarlo en caso de que el usuario lo hubiera seleccionado (Figura 59)

```python
for protocol in protocols:
    aleBeaconMacs, refBeaconMacs = processBeaconMacs(campaignId, protocol)
    for channel in channels:
        for metric in metrics:
            for algorithm in algorithms:
                for kernel in kernels:

                    taskId = taskToJson(method, protocol, channel, rssiSamples, algorithm, metric, ksRangeJson, kernel, cs, gammas, nus, Is)

                    if not checkExistingMethodPrediction(campaignId, taskId):

                        datetime_start = datetime.now()
                        createTaskHistory(campaignId, campaignName, "Obteniendo matriz de puntos de referencia...", taskId, datetime_start)

                        refRSSIMatrix = getRefPointsMatrix(ref_points, ref_points_conf, refBeaconMacs, campaignId, protocol, channel, rssiSamples)

                        updateTaskHistory(campaignId, taskId, "Obteniendo matriz de puntos aleatorios...")

                        aleRSSIMatrix= getAlePointsMatrix(ale_points, ale_points_conf, aleBeaconMacs, campaignId-1, protocol, channel, rssiSamples)

                        updateTaskHistory(campaignId, taskId, "Aplicando método de posicionamiento...")

                        predicted_points = applyMethod(refRSSIMatrix, aleRSSIMatrix, method, metric, algorithm, ksRange, kernel, cs, gammas, nus, Is, taskId)

                        mean_error = calculate_error(aleRSSIMatrix, predicted_points)

                        createMethodPrediction(campaignId, taskId, json.dumps(predicted_points), json.dumps(mean_error))

                        datetime_end = datetime.now()
                        updateTaskHistory(campaignId, taskId, "Completado", datetime_end)
```

Figura 59: Función *dataProcessing()*: parte 2

En primer lugar, se realiza el procesamiento de las MAC de los beacons que se utilizaron en la campaña de recopilación de puntos aleatorios y puntos de referencia (Figura 60). Este procesamiento es necesario debido a una observación durante el desarrollo, que indica que algunos beacons asociados a la configuración de *Robomap* pueden no tener señales BLE asociadas. Si un beacon está presente en la configuración de *Robomap,* pero no tiene señales BLE asociadas debe ser excluido ya que podría generar problemas o resultados incorrectos durante el procesamiento de los datos.

```
def processBeaconMacs(campaignId, protocol):
    aleBeaconMacs = getBeaconConfigurationByCampaignId(campaignId-1)
    refBeaconMacs = getBeaconConfigurationByCampaignId(campaignId)

    aleBeaconsProcessed = []
    refBeaconsProcessed = []

    for mac in aleBeaconMacs:
        if existsBeaconWithProtocol(campaignId-1, mac, protocol) == True:
            aleBeaconsProcessed.append(mac)

    for mac in refBeaconMacs:
        if existsBeaconWithProtocol(campaignId, mac, protocol) == True:
            refBeaconsProcessed.append(mac)

    return aleBeaconsProcessed, refBeaconsProcessed
```

Figura 60: Función *processBeaconMacs()*

Si bien, en la funcion *processingData()* se está haciendo uso de un bucle de 5 niveles, se ha prestado especial atención para garantizar que esta condición no afecte negativamente el procesamiento de los datos. En caso de que uno de los parámetros no sea seleccionado, se rellena con un elemento nulo, lo que permite que el bucle funcione de manera apropiada y continúe el procesamiento de los demás parámetros.

Lo siguiente es serializar los parámetros para asociarlos a la tarea que se está llevando a cabo, esto se realiza por medio de la función *taskToJson()*(Figura 61).

```python
def taskToJson(methods, protocols, channels, rssiSamples, algorithm, metric, ksRange, kernel, cs, gammas, nus, Is):
    taskDescription = json.dumps({
    'method': methods,
    'protocol': protocols,
    'channel': channels,
    'rssiSample': rssiSamples,
    'algorithm': algorithm,
    'metric': metric,
    'ksRange': ksRange,
    'kernel': kernel,
    'cs': cs,
    'gammas': gammas,
    'nus': nus,
    'Is': Is
    })

    return taskDescription
```

Figura 61: Función *taskToJson()*

Se comprueba que el procesamiento de los datos dado el conjunto de parámetros no haya sido generado anteriormente mediante la función *checkExistingMethodPrediction()*.

Lo siguiente es generar las matrices RSSI de puntos de referencia y aleatorios. Esto se consigue por medio de las funciones *getRefPointsMatrix()* y *getAlePointsMatrix(),* podemos observar la lógica en las Figuras 62 y 63 respectivamente.

```python
def getRefPointsMatrix(points, points_conf, beaconMacs, campaignId, protocol, channel, rssiSamples):
    coordinates = 3
    cols = len(beaconMacs) + coordinates
    rows = len(points)

    refPointsMatrix = np.full((rows, cols), -np.inf)

    for i, (point, point_values) in enumerate(zip(points, points_conf.values())):
        x = float(point_values['X'])
        y = float(point_values['Y'])
        z = float(point_values['Z'])
        rotations = point_values['Rotations']

        for j, mac in enumerate(beaconMacs):
            max_samples = []

            for k, rotation in enumerate(rotations):
                matchSamples = getFilteredPredataSamples(campaignId, rssiSamples, float(rotation), mac, protocol, channel, point['x'], point['y'], point['z'])

                if matchSamples:
                    max_sample = max(matchSamples, key=lambda sample: sample.RSSI)
                    max_samples.append(max_sample.RSSI)


            avg_max_sample = sum(max_samples) / len(max_samples)
            refPointsMatrix[i,j] = avg_max_sample

        refPointsMatrix[i,cols - 3] = x
        refPointsMatrix[i,cols - 2] = y
        refPointsMatrix[i,cols - 1] = z
```

Figura 62: Función *getRefPointsMatrix()*

En *getRefPointsMatrix()* se genera una matriz con el valor medio de los valores máximos por orientación para cada uno de los beacons que han intervenido en la captura de datos y para cada uno de los puntos de referencia recopilados. Las tres columnas finales contienen las coordenadas reales (X, Y, Z) de los puntos de referencia.

```python
def getAlePointsMatrix(points, points_conf, beaconMacs, campaignId, protocol, channel, rssiSamples):

    coordinates = 3
    cols = len(beaconMacs) + coordinates
    rows = len(points)

    alePointsMatrix = np.full((rows, cols), -np.inf)

    for i, (point, point_values) in enumerate(zip(points, points_conf.values())):
        x = float(point_values['X'])
        y = float(point_values['Y'])
        z = float(point_values['Z'])
        rotation = float(point_values['Rotation'])


        for j, mac in enumerate(beaconMacs):
            matchSamples = getFilteredPredataSamples(campaignId, rssiSamples, rotation, mac, protocol, channel, point['x'], point['y'], point['z'])

            if matchSamples:
                max_sample = max(matchSamples, key=lambda sample: sample.RSSI)
                alePointsMatrix[i,j] = max_sample.RSSI

        alePointsMatrix[i,cols - 3] = x
        alePointsMatrix[i,cols - 2] = y
        alePointsMatrix[i,cols - 1] = z

    return alePointsMatrix
```

Figura 63: Función *getAlePointsMatrix()*

Por otra parte, en *getAlePointsMatrix()* se genera una matriz con el valor máximo para la orientación en la que se haya tomado la captura (generalmente orientación Este) para cada uno de los beacons que han intervenido en la captura de datos y para cada uno de los puntos aleatorios recopilados. Las tres columnas finales contienen las coordenadas reales (X, Y, Z) de los puntos aleatorios.

Seguidamente, se aplica el método seleccionado haciendo uso del servicio *MethodsService.py* (Figura 64).

```python
def applyMethod(data_train, data_test, method, metrics, algorithms, ks_range, kernel, cs, gammas, nus, Is, description):
    match method:
        case "WKNN":
            return applyWKNNmethod(data_train, data_test, metrics, algorithms, ks_range, description)
        case "SVR":
            return applySVRmethod(data_train, data_test, kernel, cs, gammas, description)
        case "NuSVR":
            return applyNuSVRmethod(data_train, data_test, kernel, cs, gammas, nus, description)
        case "LinearSVR":
            return applyLinearSVRmethod(data_train, data_test, Is, cs, description)
```

Figura 64: Función *applyMethod()*

A continuación, se presentan los métodos de posicionamiento implementados. Cada método recibe como parámetros la matriz de RSSI aleatorios como datos de entrenamiento, la matriz de RSSI de referencia como datos de prueba y los parámetros específicos de cada método, además de la descripción de la tarea para asignarla como clave en el diccionario de valores generado. Para ello se hace uso de la función *getKeyDescription()* (Figura 69).

- El método WkNN *(Weighted k-Nearest Neighbors)* (Figura 65) se basa en asignar pesos a los vecinos más cercanos. En la función se extraen las características y las etiquetas, se crea un modelo WkNN para cada valor de k, se realizan las predicciones correspondientes y finalmente se almacenan los resultados en un diccionario donde cada clave es la descripción de los hiperparámetros utilizados.

```
def applyWKNNmethod(data_train, data_test, metric, algorithm, ks_range, description):
    keyName = getKeyDescription(description)

    X_train = data_train[:, :-3]
    y_train = data_train[:, -3:]

    results = {}

    for i in range(ks_range[0], ks_range[1] + 1):
        regressor = KNeighborsRegressor(n_neighbors=i, weights=lambda distances: 1 / distances,
                                        metric=metric, algorithm=algorithm)
        regressor.fit(X_train, y_train)

        X_test = data_test[:, :-3]
        y_pred = regressor.predict(X_test).tolist()
        newKeyName = keyName + f'[k={str(i)}]'
        results[newKeyName] = y_pred

    return results
```

Figura 65: Implementación del método WkNN

- El método SVR *(Support Vector Regression)* (Figura 66) se basa en la utilización de vectores de soporte no lineales para realizar la regresión. El objetivo de SVR es encontrar la función que minimice el error dentro de un margen definido. En primer lugar, la función extrae las características y las etiquetas, se crea un modelo SVR para cada combinación de parámetros (kernel, C y gammas) y se realiza un *StandardScaler* para normalizar las características del conjunto de entrenamiento. En siguiente lugar, se utiliza el *MultiOutputRegressor* para tratar con múltiples etiquetas de salida, luego se realizan las predicciones utilizando el modelo

entrenado en el conjunto de prueba y finalmente se almacenan los resultados en un diccionario.

```python
def applySVRmethod(data_train, data_test, kernel, cs, gammas, description):
    keyName = getKeyDescription(description)

    X_train = data_train[:, :-3]
    y_train = data_train[:, -3:]

    results = {}

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(data_test[:, :-3])

    for c in cs:
        if isinstance(gammas, list):
            for gamma in gammas:
                model = SVR(kernel=kernel, C=c, gamma=gamma)
                wrapper = MultiOutputRegressor(model).fit(X_train, y_train)

                y_pred = wrapper.predict(X_test)
                y_pred_list = y_pred.tolist()

                newKeyName = keyName + f'[k={f"C{c}_G{gamma}"}]'
                results[newKeyName] = y_pred_list
        else:
            model = SVR(kernel=kernel, C=c, gamma=gammas)
            wrapper = MultiOutputRegressor(model).fit(X_train, y_train)

            y_pred = wrapper.predict(X_test)
            y_pred_list = y_pred.tolist()

            newKeyName = keyName + f'[k={f"C{c}_G{gamma}"}]'
            results[newKeyName] = y_pred_list

    return results
```

Figura 66: Implementación del método SVR

- El método NuSVR (*Nu Support Vector Regression*) es similar a SVR al utilizar vectores de soporte no lineales para realizar la regresión, sin embargo, cuenta además con un parámetro de soporte adicional llamado Nu, utilizado para manejar el margen de tolerancia. De cara a la lógica, es necesario agregar un bucle adicional que permita recorrer cada uno de los Nu enviados como parámetros.

```python
def applyNuSVRmethod(data_train, data_test, kernel, cs, gammas, nus, description):
    keyName = getKeyDescription(description)

    X_train = data_train[:, :-3]
    y_train = data_train[:, -3:]

    results = {}

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(data_test[:, :-3])

    for nu in nus:
        for c in cs:
            if isinstance(gammas, list):
                for gamma in gammas:
                    model = NuSVR(nu=nu, kernel=kernel, C=c, gamma=gamma)
                    wrapper = MultiOutputRegressor(model).fit(X_train, y_train)

                    y_pred = wrapper.predict(X_test)
                    y_pred_list = y_pred.tolist()

                    newKeyName = keyName + f'[k={f"nu{nu}_C{c}_G{gamma}"}]'
                    results[newKeyName] = y_pred_list
            else:
                model = NuSVR(nu=nu, kernel=kernel, C=c, gamma=gammas)
                wrapper = MultiOutputRegressor(model).fit(X_train, y_train)

                y_pred = wrapper.predict(X_test)
                y_pred_list = y_pred.tolist()
                newKeyName = keyName + f'[k={f"nu{nu}_C{c}_G{gamma}"}]'
                results[newKeyName] = y_pred_list

    return results
```

Figura 67: Implementación del método NuSVR

- El método LinearSVR (Figura 68), al contrario que en SVR o NuSVR, utiliza una regresión lineal mediante vectores de soporte para generar la predicción. Por lo tanto, este método no cuenta con kernel, parámetros gamma o parámetros Nu. Se introduce un nuevo parámetro I para controlar el número de iteraciones de la regresión.

```python
def applyLinearSVRmethod(data_train, data_test, Is, cs, description):
    keyName = getKeyDescription(description)

    X_train = data_train[:, :-3]
    y_train = data_train[:, -3:]

    results = {}

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(data_test[:, :-3])


    for i in Is:
        for c in cs:
                model = LinearSVR(C=c, max_iter=i, random_state=0)
                wrapper = MultiOutputRegressor(model).fit(X_train, y_train)

                y_pred = wrapper.predict(X_test)
                y_pred_list = y_pred.tolist()

                newKeyName = keyName + f'[k={f"i{i}_C{c}"}]'
                results[newKeyName] = y_pred_list

    return results
```

Figura 69: Implementación del método LinearSVR

```python
def getKeyDescription(description):
    key = json.loads(description)
    keyName = ''

    for field, value in key.items():
        if value != [None] and value != 0 and field != 'ksRange' and field != 'cs' and field != 'Is' and field != 'nus' and field != 'gammas':
            keyName += f'[{value}]'

    return keyName
```

Figura 68: Función *getKeyDescription()*

Acto seguido, se obtiene el error y el error medio entre la matriz de RSSI aleatorios y los puntos obtenidos a partir de la ejecución del método IPS haciendo uso de la función *calculateError()* (Figura 70).

```python
def calculateError(aleMatrix, results):

    y_test = aleMatrix[:, -3:]
    errors = {}

    for k, y_pred in results.items():
        error = np.sqrt(np.sum((y_test - y_pred) ** 2, axis=1))
        mean_error_rounded = round(np.mean(error), 3)
        newKeyName = f'{str(k)}[Mean={str(mean_error_rounded)}]'
        errors[newKeyName] = error

    errorsList = {k: sorted(v) for k, v in errors.items()}

    return errorsList
```

Figura 70: Función *calculateError()*

Por último, se dan de alta los resultados en la entidad *MethodPrediction* mediante la función *createMethodPrediction()* (Figura 71).

```python
def createMethodPrediction(campaignId, method_description, predicted_points, mean_error):

    methodPrediction = MethodPrediction()
    methodPrediction.Id_campaign = campaignId
    methodPrediction.Method_description = method_description
    methodPrediction.Predicted_points = predicted_points
    methodPrediction.Mean_error = mean_error

    db.session.add(methodPrediction)

    db.session.commit()
    db.session.remove()
```

Figura 71: Función *createMethodPrediction()*

10.3. Estado de procesamiento

El estado de procesamiento de la aplicación de un método de posicionamiento se encuentra en el componente *task-history* del *Frontend*.

Este componente llama al método del servicio de la API correspondiente tal y como vemos en la Figura 72.

```typescript
constructor(private apiService: ApiService, private cdr: ChangeDetectorRef) { }
ngOnInit(){
  this.apiService.pollTaskHistory().subscribe(data => {
    const objectsArray: any[] = [...Object.values(data)];
    this.historyItemDataSource.data = objectsArray;
    this.historyTasks = objectsArray;
  });
}
```

Figura 72: Función *pollTaskHistory()*

Esta función se encarga de llamar cada 5 segundos a la API y obtener todas las tareas actualizadas (Figura 73). Esto permite obtener actualizaciones en tiempo real sin necesidad de recargar la vista correspondiente.

```
getTaskHistory(){
  return this.http.get(this.baseUrl + "taskHistory");
}

pollTaskHistory(): Observable<any> {
  return timer(0, 5000).pipe(
    takeUntil(this.destroy$),
    switchMap(() => {
      if (this.router.url === '/task-history') {
        return this.getTaskHistory();
      } else {
        this.destroy$.next(null);
        return EMPTY;
      }
    })
  );
}
```

Figura 73: Funciones *pollTaskHistory()* y *getTaskHistory()*

La actualización del estado de una tarea ocurre en el *Backend*, específicamente en el archivo *DataProcessService.py*. Durante el procesamiento del método, se utilizan las funciónes *createTaskHistory()* y *updateTaskHistory()* para realizar una serie de actualizaciones que van reflejando el estado actual de la tarea. Esta funcionalidad permite al usuario verificar el progreso de la tarea en tiempo real. (Figura 74).

```
datetime_start = datetime.now()
createTaskHistory(campaignId, campaignName, "Obteniendo matriz de puntos de referencia...", taskId, datetime_start)

refRSSIMatrix = getRefPointsMatrix(ref_points, ref_points_conf, refBeaconMacs, campaignId, protocol, channel, rssiSamples)

updateTaskHistory(campaignId, taskId, "Obteniendo matriz de puntos aleatorios...")

aleRSSIMatrix= getAlePointsMatrix(ale_points, ale_points_conf, aleBeaconMacs, campaignId-1, protocol, channel, rssiSamples)

updateTaskHistory(campaignId, taskId, "Aplicando método de posicionamiento...")

predicted_points = applyMethod(refRSSIMatrix, aleRSSIMatrix, method, metric, algorithm, ksRange, kernel, cs, gammas, nus, Is, taskId)

mean_error = calculateError(aleRSSIMatrix, predicted_points)

createMethodPrediction(campaignId, taskId, json.dumps(predicted_points), json.dumps(mean_error))

datetime_end = datetime.now()
updateTaskHistory(campaignId, taskId, "Completado", datetime_end)
```

Figura 74: Funciónes *createTaskHistory()* y *updateTaskHistory()*

10.4. Gestión de campañas

Para visualizar las campañas guardadas, se accede al componente del *Frontend* denominado *campaigns*, el cual contiene los elementos individuales de cada campaña definidos por *campaign-card*.

En la figura 75 podemos observar las opciones disponibles de cada *campaign-card. "Ver detalles*" enruta hacia el componente *campaign-details*, *"Generar gráficas"* enruta hacia el componente *graphics* y *"Borrar campaña"* hace uso de la función *deleteconfirmDialog()* para borrar una campaña.

```
<footer class="card-footer">
    <a routerLink="/campaign-details" [queryPara
    <a routerLink="/graphics" [queryParams]="{ i
    <a (click)="deleteConfirmDialog()" class="ca
</footer>
```

Figura 75: Opciones disponibles de *campaign-card*

La función *deleteConfirmDialog()* hace una llamada al servicio de la API para realizar el borrado de la campaña (Figura 76). Esta función contiene la llamada a la API llamada *deleteCampaignById()* (Figura 77).

```
deleteConfirmDialog(){
  const dialogRef = this.dialog.open(ConfirmationDialogComponent, {
    width: '400px',
    data: 'Esta acción borrará la campaña seleccionada y no se puede deshacer.'
  })

  dialogRef.afterClosed().subscribe(res =>{
    if(res){
      this.apiService.deleteCampaignById(this.campaign.Id).pipe(
        tap(() => {
            this.toastr.success('¡La campaña se ha borrado correctamente!', 'Operación completada', { timeOut: 2000 });
            this.router.navigateByUrl('/', { skipLocationChange: true }).then(() => {
              this.router.navigate(['/campaigns'])
            });
          }),
          catchError(() => {
            this.toastr.error('Ha ocurrido un error durante el borrado', 'Operación no completada', { timeOut: 2000 });
            return of(null);
          })
        ).subscribe();
    }
  })
}
```

Figura 76: Función *deleteConfirmDialog()*

```
deleteCampaignById(id: any){
  return this.http.delete(this.baseUrl + `deleteCampaign/${id}`);
}
```

Figura 77: Función *deleteCampaignById()*

En el *Backend*, encontramos la función *deleteCampaignById()* (Figura 78) de tipo *delete*. Esta función se encarga de realizar una consulta basada en el identificador de la campaña para verificar su existencia, así como la existencia de la campaña relacionada. En primer lugar, se realizan múltiples llamadas a las funciones de borrado de las diferentes entidades que dependen de una campaña. En siguiente lugar, si existiera, se borra el comprimido de imágenes asociado a la campaña con la función *deleteZipImages()* (Figura 79). Por último, se libera el espacio en la base de datos que ocupaban los registros eliminados mediante el comando de SQLite *"Vacuum"*.

```
def deleteCampaignById(campaignId):
    campaign = Campaign().query.filter_by(Id=campaignId).first()
    relatedCampaign = Campaign().query.filter_by(Id=campaign.Id_related_campaign).first()

    if campaign:
        deleteCampaignSequenceByCampaignId(campaignId)
        deleteCampaignSequenceByCampaignId(relatedCampaign.Id)

        deleteBeaconBleSignalByCampaignId(campaignId)
        deleteBeaconBleSignalByCampaignId(relatedCampaign.Id)

        deleteCaptureByCampaignId(campaignId)
        deleteCaptureByCampaignId(relatedCampaign.Id)

        deleteConfigurationsByCampaignId(campaignId)
        deleteConfigurationsByCampaignId(relatedCampaign.Id)

        deleteMethodPredataByCampaignId(campaignId)
        deleteMethodPredataByCampaignId(relatedCampaign.Id)

        deleteMethodPredictionByCampaignId(campaignId)

        deleteTaskHistoryByCampaignId(campaignId)

        if campaign.Images_ref is not None:
            deleteZipImages(campaign.Images_ref)

        db.session.delete(campaign)
        db.session.delete(relatedCampaign)

        db.session.commit()
        db.session.execute('VACUUM')

        return {'message': 'Campaign successfully deleted'}, 200
    else:
        return {'message', 'Campaign not found'}, 404
```

Figura 78: Función *deleteCampaignById()*

```python
def deleteZipImages(zip):
    os.remove(os.path.join(imagesDir, zip))
```

Figura 79: Función *deleteZipImages()*

10.5. Visualización de detalles

Para visualizar los detalles de una campaña se accede al componente *campaign-details* del *Frontend*. Dado un *id* de campaña, se obtienen todos sus detalles y los de su campaña asociada. En primer lugar, se obtienen sus imágenes por medio de la función del API *getCampaignImagesById()* (Figura 80).

```javascript
apiService.getCampaignImagesById(this.campaign.Id).subscribe((response: Blob) => {
  if(response != null){
    const zipImages = new JSZip();
    zipImages.loadAsync(response).then(images =>{
      let count = 0;
      Object.keys(images.files).forEach(filename => {
        images.files[filename].async('base64').then(imageData => {
          const imageUrl = '' + imageData;
          if (count < 3 && Object.keys(images.files).length == 1) {
            for (let i = 0; i < 2; i++) {
              this.campaignImages.push(imageUrl);
              count++;
            }
          } else {
            this.campaignImages.push(imageUrl);
            count++;
          }
        })
      })
    })
  }
```

Figura 80: Función *getCampaignImagesById()*

Al solicitarse las imágenes el *Backend* hace uso de la función *sendZipImages()* para mandar el comprimido según la referencia de imágenes guardada en la campaña.

```python
def sendZipImages(imagesRef):
    image_zip = imagesDir + '/' + imagesRef

    response = send_file(
        image_zip,
        mimetype='application/zip',
        as_attachment=True,
        download_name=imagesRef
    )
    return response
```

Figura 81: Función *sendZipImages()*

Por otra parte, se recopilan los datos de las entidades asociadas a la campaña (Figura 82) y a su campaña asociada (Figura 83) y se mapean por lotes para poder visualizarse en el *Frontend* correctamente. Dado el gran volumen de trabajo involucrado en la recopilación y procesamiento de datos, se ha implementado un icono de carga que proporciona retroalimentación visual al usuario. Este icono de carga aparece durante unos pocos segundos para indicar al usuario que el sistema está ocupado realizando tareas intensivas.

```
campaignData(): Observable<any> {
  return forkJoin([
      this.apiService.getDongleName(this.campaign.Id_dongle),
      this.apiService.getCampaignSequence(this.campaign.Id_campaign_sequence),
      this.apiService.getCaptures(this.campaign.Id),
      this.apiService.getSignals(this.campaign.Id),
      this.apiService.getConfigs(this.campaign.Id),
  ]).pipe(
      map(([dongleName, campaignSequence, captures, signals, configs]) => {
          return {
              dongleName,
              campaignSequence,
              captures,
              signals,
              configs
          };
      })
    );
}
```

Figura 82: Función *campaignData()*

```
relatedCampaignData(): Observable<any> {
  return this.apiService.getRelatedCampaignById(this.campaign.Id_related_campaign).pipe(
    switchMap((relatedCampaign: any) => {
      return forkJoin([
        this.apiService.getDongleName(relatedCampaign.Id_dongle),
        this.apiService.getCampaignSequence(relatedCampaign.Id_campaign_sequence),
        this.apiService.getCaptures(relatedCampaign.Id),
        this.apiService.getSignals(relatedCampaign.Id),
        this.apiService.getConfigs(relatedCampaign.Id)
      ]).pipe(
        map(([dongleName, campaignSequence, captures, signals, configs]) => {
          relatedCampaign.dongleName = dongleName;
          relatedCampaign.campaignSequence = campaignSequence;
          relatedCampaign.captures = captures;
          relatedCampaign.signals = signals;
          relatedCampaign.configs = configs;
          return relatedCampaign;
        })
      );
    })
  );
}
```

Figura 83: Función *relatedCampaignData()*

10.6. Generación de gráficas

Para visualizar y generar gráficas de precisión y exactitud hay que acceder al componente *graphics* del *Frontend.*

En primer lugar, se realiza una consulta al servicio de la API para obtener todas las predicciones o aplicaciones de métodos de procesamiento (Figura 84).

```
this.apiService.getMethodPredictionsById(this.campaignId).subscribe(res=> {

  if(res != 0){
    this.campaignMethodPrediction = res
  }
})
```

Figura 84: Función *getMethodPredictionById()*

En siguiente lugar, se aplican los filtros de los parámetros e hiperparámetros para obtener los datos deseados de forma dinámica según el tipo de grafica seleccionada.

Cuando el usuario aplica los filtros y selecciona la opción de insertar datos en la gráfica se realiza una llamada a las funciones *insertPrecisionData()* o *insertAccuracyData()* según corresponda (Figura 85).

```
insertPrecisionData(){
  this.totalSeries = [];
  this.onFilterSelected('precision');
}

insertAccuracyData(){
  this.onFilterSelected('accuracy');
}
```

Figura 85: Funciones *insertPrecisionData()* e *insertAccuracyData()*

Por una parte, la función *onFilterSelected()* prepara los datos de filtrado dinámico para aplicarlos a los datos recibidos desde el *Backend* (Figura 86).

```
onFilterSelected(chartTipe:string){
  this.filteredData = [...this.campaignMethodPrediction];

  this.filteredData = this.filterData(this.form.get('methods')?.value, "method", this.filteredData);
  this.filteredData = this.filterData(this.form.get('protocols')?.value, "protocol", this.filteredData);
  this.filteredData = this.filterData(this.form.get('channels')?.value, "channel", this.filteredData);

  switch(this.selectedMethod){
    case "WKNN":
      this.filteredData = this.filterData(this.form.get('metrics')?.value, "metric", this.filteredData);
      this.filteredData = this.filterData(this.form.get('algorithms')?.value, "algorithm", this.filteredData);
      if(chartTipe == 'precision'){
        this.filteredData = this.filterRange(this.filteredData, "WKNN", "Mean_error");
      }else{
        this.filteredData = this.filterRange(this.filteredData, "WKNN", "Predicted_points");
      }
      break;
```

Figura 86: Función *onFilterSelected()*: parte 1

Las funciones *filterData()* (Figura 87) y *filterRange()* (Figura 88) se utilizan para aplicar filtros a los hiperparámetros del método, así como a aquellos hiperparámetros que requieran un filtrado basado en un rango específico. Estos filtros permiten controlar y ajustar los parámetros de acuerdo con las propiedades del método y los requisitos de filtrado establecidos.

```
filterData(filters: any, property: string, data: any[]): any[]{
  const filterArray = Array.isArray(filters) ? filters : [filters];
  return data.filter((object:any) =>{
    const methodDescription = JSON.parse(object.Method_description);
    return filterArray.some((filter: string) => filter === methodDescription[property].toString());
  })
}
```

Figura 87: Función *filterData()*

```
filterRange(data: any[], rangeType: string, atribute: string): any[] {

  return data.map((object: any) => {
    const kRangeStart = this.form.get('kRangeStart')?.value;
    const kRangeEnd = this.form.get('kRangeEnd')?.value;
    const cRangeStart = this.form.get('cRangeStart')?.value;
    const cRangeEnd = this.form.get('cRangeEnd')?.value;
    const gRangeStart = this.form.get('gRangeStart')?.value;
    const gRangeEnd = this.form.get('gRangeEnd')?.value;
    const gamma = this.form.get('gamma')?.value;
    const nuRangeStart = this.form.get('nuRangeStart')?.value;
    const nuRangeEnd = this.form.get('nuRangeEnd')?.value;
    const iRangeStart = this.form.get('iRangeStart')?.value;
    const iRangeEnd = this.form.get('iRangeEnd')?.value;

    const clonedObject = { ...object }
    const objParam = JSON.parse(clonedObject[atribute]);
    const filteredParam: any = {};

    Object.entries(objParam).forEach(([key, value]) => {
      switch(rangeType){
        case 'WKNN':

          const matchesWKNN = parseInt(key.match(/k=(\d+)/)?.[1] || "", 10);

          if (matchesWKNN >= kRangeStart && matchesWKNN <= kRangeEnd) {
            filteredParam[key] = value;
          }
          break;
```

Figura 88: Función *filterRange()*

Los filtros se aplican gracias a que se preparan adecuadamente los datos durante la fase de aplicación del método de posicionamiento. Para ello, se utiliza una estructura de datos basada en diccionarios de pares *clave-valor*. La clave se utiliza para identificar los filtros necesarios para categorizar cada valor mostrado en los campos *"Predicted_points"* y *"Mean_error"* de la entidad *"Method_Prediction"*.

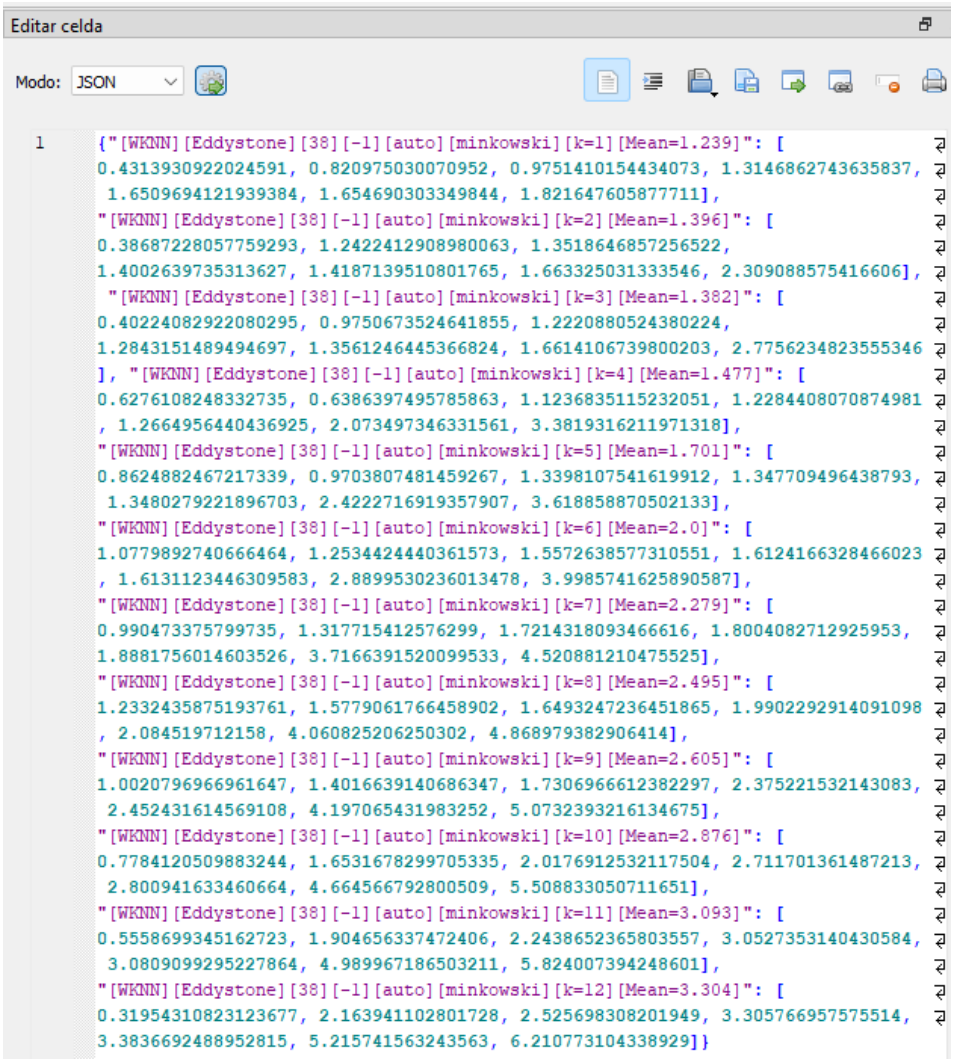

Figura 89: Estrctura de datos del campo *Mean_*error de la entidad *Method_Prediction*

Tras aplicar los filtros necesarios a cada conjunto de datos, la función *onFilterSelected()* (Figura 90) realiza una llamada a las funciones *parsePrecisionData()* (Figura 91) o *parseAccuracyData()* (Figura 92)

respectivamente, que da el formato esperado por la librería de generación de gráficas y esto permite que se puedan rellenar con los datos obtenidos.

```
if(chartTipe === 'precision'){
  let parsedData = this.parsePrecisionData();
  this.fillPrecisionData(parsedData);
}
else {
  let parsedData = this.parseAccuracyData();
  this.fillAccuracyData(parsedData);
}
```

Figura 91: Función *onFilterSelected():* parte 2

```
parsePrecisionData() {
  let parsedData = this.filteredData.map((item:any) => {
      let meanErrors = JSON.parse(item.Mean_error);

      let data = Object.keys(meanErrors).map(key => ({
          key: key,
          meanError: meanErrors[key]
      }
      ));

      const cdfValues = this.calculateCDF(data);

      return {
          ...item,
          PrecisionData: data,
          CdfValues: cdfValues
      };
  });

  return parsedData;
}
```

Figura 92: Función *parsePrecisionData()*

```
parseAccuracyData() {
  let parsedData = this.filteredData.map((item: any) => {
    let predictedPoints = JSON.parse(item.Predicted_points);

    let predPoints = Object.entries(predictedPoints).map(([key, points]: [string, any]) => ({
      key: key,
      values: points.map(([x, y]: [number, number]) => ({ x, y })),
    }));

    let aleatoryPoints = this.alePoints ? Object.values(this.alePoints).map((point: any) => ({
      x: parseFloat(point.X),
      y: parseFloat(point.Y),
    })) : [];


    return {
      ...item,
      PredictedData: predPoints,
      AleatoryData: aleatoryPoints,
    };
  });

  return parsedData;
}
```

Figura 90: Función *parseAccuracyData()*

En el caso de la función *parsePrecisionData(),* se obtiene la función de distribución acumulada o CDF (*Cumulative Distribution Function*) y los datos se insertan dentro del campo *"CdfValues"*. Esto se consigue por medio de la función *calculateCDF() (Figura 93).* La CDF representa la probabilidad acumulada de que una variable aleatoria sea menor o igual a un determinado valor.

```
calculateCDF(data: any) {
  const cdfData = [];

  for (let i = 0; i < data.length; i++) {
      let meanErrors = data[i].meanError;
      let cdf:any = [];
      let totalCount = meanErrors.length;
      let cumulativeCount = 0;

      meanErrors.forEach((errorValue:any, index:any) => {
          cumulativeCount += 1;
          const probability = cumulativeCount / totalCount;
          cdf.push({ x: errorValue, y: probability });
      });

      cdf.unshift({ x: 0, y: 0 });

      cdfData.push({
          key: data[i].key,
          cdf: cdf
      });
  }

  return cdfData;
```

Figura 93: Función *calculateCDF()*

Por último, se llega a las funciones *fillPrecisionData()* y *fillAccuracyData()* referenciadas en la Figura 92.

La función *fillPrecisionData()* (Figura 94) se encarga de recorrer los datos filtrados e incorporarlos a la gráfica correspondiente. En primer lugar, se añade un color aleatorio a cada una de las series para diferenciarlas.

Por otro lado, se limita el volcado de datos a un máximo de 30 series para evitar problemas de rendimiento y saturación de datos, esto solo ocurre si los filtros son demasiado amplios.

Durante el desarrollo, se han definido dos listas de series: *totalSeries* y *selectedSeries*. Estas listas se utilizan para organizar y gestionar las series de datos en las gráficas.

La lista *totalSeries* representa todas las series de datos que se obtienen después de aplicar el filtrado correspondiente y contiene todas las series que coinciden con los criterios de búsqueda aplicados cada vez.

Por otro lado, la lista *selectedSeries* se refiere a las series de datos que han sido seleccionadas por el usuario y se muestran en la gráfica. Esta lista contiene un subconjunto de las series presentes en *totalSeries* y se utiliza para determinar qué series deben mostrarse visualmente al usuario. La selección de series se realiza por medio de la función *onClick()* ubicada en la configuración de la gráfica de precisión (Figura 94).

```
onClick: (event, legendItem, legend) => {

    const index = legendItem.datasetIndex;

    const chart = legend.chart;
    const dataset = chart.data.datasets[index||0];

    if (!this.selectedSeries.includes(dataset)) {
      dataset.hidden = false;
      this.selectedSeries.push(dataset);
    }
    else {
      let foundIndex = this.selectedSeries.findIndex((data:any) => data === dataset);
      if (foundIndex !== -1) {
          this.selectedSeries.splice(foundIndex, 1);
      }
      dataset.hidden = true;
    }

    chart.update(this.selectedSeries);
}
```

Figura 94: Función *onClick()*

Por lo tanto, lo siguiente que se realiza en la función *fillPrecisionData()* es filtrar los datos de *selectedSeries* que ya estén seleccionados por el usuario, y así evitar que puedan haber duplicados en *totalSeries* (Figura 95).

```
fillPrecisionData(filteredData:any){
  let labels = [];

  for(let i = 0; i < filteredData.length; i++){
    let item = filteredData[i];
    let cdfValues = item.CdfValues;

    for(let j = 0; j < cdfValues.length; j++){
        let data = cdfValues[j];
        let meanErrors = data.cdf;

        let color = this.getRandomColor();
        let backgroundColor = color + '0.2';

        let dataset = {
            data: meanErrors,
            label: data.key,
            backgroundColor: backgroundColor,
            borderColor: color,
            pointBackgroundColor: color,
            pointBorderColor: '#fff',
            pointHoverBackgroundColor: '#fff',
            pointHoverBorderColor: color,
            fill: false,
            hidden: true
        }

        for (let selected of this.selectedSeries){
          if (!this.totalSeries.includes(selected) && this.totalSeries.length < 30) {
              this.totalSeries.push(selected);
          }
        }

        if(this.totalSeries.length < 30) {
          this.totalSeries.push(dataset);
        }
        else {
            this.toastr.warning('No se han cargado todas las series. Ajusta tus selecciones
            break;
        }

        const filteredSeries = this.totalSeries.reduce((acc:any, curr:any) => {
          const existingItem = acc.find((item:any) => item.label === curr.label);

          if (!existingItem) {
            acc.push(curr);
          }
```

Figura 95: Función *fillPrecisionData()*

La función *fillAccuracyData()* realiza un procesamiento ligeramente diferente al comentado anteriormente ya que debe incluir un *dataset* de los puntos reales (aleatorios) y compararlos con los *dataset* de puntos que se han obtenido

tras la aplicación de cada método de posicionamiento. Al igual que en el caso anterior, se establece un límite de volcado de 30 series para evitar problemas de rendimiento (Figura 96).

```
fillAccuracyData(filteredData:any){
  let colors = ["#00BFFF", "#696969", "#228B22", "#FF69B4", "#DAA520", "#D2691E", "#4682B4"];

  let datasets = [];
  let labels = [];


  let aleatoryPoints = filteredData[0].AleatoryData;
  let pointColors_ale = aleatoryPoints.map((_:any, index:any) => colors[index % colors.length]);

  let dataset_ale = {
    data: aleatoryPoints,
    label: "Puntos reales",
    backgroundColor: pointColors_ale,
    borderColor: pointColors_ale,
    pointBackgroundColor: pointColors_ale,
    pointBorderColor: '#fff',
    pointHoverBackgroundColor: '#fff',
    pointHoverBorderColor: pointColors_ale,
    fill: false,
    hidden: false,
    showLine: false,
    pointStyle: 'triangle',
    pointRadius: 10
  }

  datasets.push(dataset_ale);

  for(let i = 0; i < filteredData.length; i++){
    let item = filteredData[i];
    let predictedValues = item.PredictedData;

    for(let j = 0; j < predictedValues.length; j++){
        let data = predictedValues[j];
        let predPoints = data.values;

        let pointColors = predPoints.map((_:any, index:any) => colors[index % colors.length]);
        let dataset_pred = {
            data: predPoints,
            label: data.key,
            backgroundColor: pointColors,
            borderColor: pointColors,
            pointBackgroundColor: pointColors,
            pointBorderColor: '#fff',
            pointHoverBackgroundColor: '#fff',
            pointHoverBorderColor: pointColors,
            fill: false,
            hidden: true,
```

Figura 96: Función *fillAccuracyData()*

11. Pruebas y resultados

En el siguiente apartado, se presentan las pruebas realizadas para verificar el correcto funcionamiento de la herramienta, así como una breve comparativa de métodos de posicionamiento en interiores basada en los resultados obtenidos. Estas pruebas se llevaron a cabo para evaluar el rendimiento y la precisión de los diferentes métodos de posicionamiento implementados.

11.1. Validación de la herramienta

Para validar el procedimiento implementado en la herramienta *IPSCampaignManager*, se contó con la valiosa colaboración del personal del IUCES. Su apoyo fue fundamental para obtener los datos necesarios y corroborar la precisión y exactitud de los resultados obtenidos.

La validación se realizó en base a una campaña realizada por *Robomap* con anterioridad denominada *BASE_DATOS_JUSTIFICACIÓN_ROBOMAP*.

El IUCES proporcionó los datos obtenidos en forma de dos archivos: "*coordenadas_PR_PA_BEACONS.xlsx"* y "*BD_SIMON.xlsx"*.

A continuación, se detallan las hojas de cálculo presentes en el archivo *"coordenadas_PR_PA_BEACONS.xlsx"*:

- *Coord-PR*: Contiene las coordenadas de los puntos de referencia utilizados en el proceso. Estas coordenadas representan ubicaciones específicas en el entorno interior que se utilizan como referencia para el posicionamiento.
- *Coord-PA*: En esta hoja de cálculo se encuentran las coordenadas de los puntos aleatorios. Estos puntos se utilizan como datos de referencia para evaluar el rendimiento de los métodos de posicionamiento implementados.
- *Coord-Beacons*: Aquí se registran las coordenadas de los beacons utilizados durante el proceso.
- *distancias_PR-PA*: Esta hoja de cálculo presenta las distancias entre los puntos de referencia y los puntos aleatorios. Estas

distancias se utilizan para evaluar la precisión del posicionamiento y comparar los resultados obtenidos con las distancias reales.

- *distancias_PR_BEACONS*: En esta hoja se registran las distancias entre los puntos de referencia y los beacons. Estas distancias son utilizadas para calibrar y ajustar los métodos de posicionamiento en función de las distancias reales.
- *distancias_PA_BEACONS*: Aquí se encuentran las distancias entre los puntos aleatorios y los beacons. Estas distancias también se utilizan para calibrar y ajustar los métodos de posicionamiento.

Por otra parte, estas son las hojas de cálculo presentes en el fichero *"BD_SIMON.xlsx":*

- *PR*: Se muestran los datos de los puntos de referencia en relación con los beacons. Para cada beacon y punto de referencia, se muestra el valor medio de los valores máximos de RSSI (solo para iBeacon canal 37) en las 8 orientaciones utilizadas. Las dos últimas columnas contienen las coordenadas de cada punto de referencia proporcionadas por *Robomap*.
- *PA*: Se muestran los datos de los puntos aleatorios en relación con los beacons. Para cada beacon y punto aleatorio, se muestra el valor máximo de RSSI (solo para iBeacon canal 37) en la orientación utilizada (90º = Norte). Las dos últimas columnas contienen las coordenadas de cada punto aleatorio proporcionadas por *ROBOMAP*.
- *distancias*: Se calculan las distancias euclidianas en el espacio de la señal. Proporciona información sobre las distancias entre los puntos de referencia, puntos aleatorios y beacons.
- *PA1* a *PA7*: Estas hojas de cálculo implementan manualmente el método WkNN para los puntos aleatorios utilizando diferentes valores de k (número de vecinos) y pesos.
- *EXACT-PREC*: Se obtiene la exactitud (error promedio) y la precisión (CDF) del método WkNN.

Por lo tanto, los parámetros e hiperparámetros utilizados para la validación de resultados fueron los siguientes:

- Método: WkNN
- Protocolo: iBeacon
- Canal: 37
- Muestras: Sin límite (parámetro -1)
- Métrica: minkowski
- Algoritmo: auto
- Rango de K vecinos: 1 a 12

Tras comparar los resultados proporcionados por el IUCES (Figura 97) y los resultados obtenidos mediante *IPSCampaignManager* (Figura 98), podemos confirmar que las pruebas han sido exitosas. Los datos coinciden de manera muy precisa, a excepción de algunas discrepancias por el número de decimales seleccionados.

Hay que destacar que *IPSCampaignManager* ordena automáticamente los puntos aleatorios en función de su error y agrega un nuevo campo en la clave del diccionario llamado *"Mean"* que representa el promedio de la serie.

PA1	PA2	PA3	PA4	PA5	PA6	PA7	promedio	k
0,82	1,65	1,65	0,98	2,15	0,43	1,82	1,36	1
1,24	1,39	1,36	1,40	1,73	0,37	2,27	1,39	2
1,65	1,28	1,23	0,98	1,37	0,42	2,65	1,37	3
2,05	1,27	1,21	1,30	1,69	0,78	3,04	1,62	4
2,41	1,36	1,29	1,01	1,25	1,06	3,42	1,69	5
2,91	1,52	1,46	0,73	1,07	1,30	3,75	1,82	6
3,19	1,74	1,68	1,00	1,35	1,54	4,07	2,08	7
3,85	1,99	1,94	1,24	1,66	1,77	4,38	2,40	8
4,16	2,37	2,32	1,02	1,37	2,01	4,70	2,56	9
4,69	2,61	2,56	0,77	1,65	2,35	5,14	2,82	10
4,95	2,98	2,93	0,58	1,96	2,61	5,46	3,07	11
5,21	3,20	3,15	0,35	2,29	2,82	5,70	3,25	12

Figura 97: Hoja *EXACT-PREC* de *BD_SIMON.xlsx*

```
{"[WKNN][iBeacon][37][-1][auto][minkowski][k=1][Mean=1.358]": [
0.4313930922024591, 0.820975030070952, 0.9751410154434073, 1.6509694121939384,
 1.6546903033498441, 1.821647605877711, 2.153531982581173],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=2][Mean=1.394]": [
0.37240309194083493, 1.2411216040928172, 1.3617157821782568,
1.3882875609272867, 1.4006103098296665, 1.7262597546491212, 2.268478232908191
], "[WKNN][iBeacon][37][-1][auto][minkowski][k=3][Mean=1.369]": [
0.4230179940539563, 0.9848281774802764, 1.2293709925965834, 1.2813258645767942
, 1.3710462150530338, 1.6433291193670305, 2.6532329009877818],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=4][Mean=1.619]": [
0.7831465957252345, 1.2048966720570682, 1.2670560667103608, 1.3105435295174979
, 1.6857656835119148, 2.0439698924453733, 3.0371016134197375],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=5][Mean=1.687]": [
1.0262984080335034, 1.056076300878396, 1.2458787411691943, 1.2915807259938499,
 1.3587187775427658, 2.4126474558343323, 3.4192752367342014],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=6][Mean=1.823]": [
0.7447937972016969, 1.0736406497231863, 1.3033536339732128, 1.4552002457691533
, 1.5205666622170202, 2.9073186636309285, 3.753771902929213],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=7][Mean=2.084]": [
1.0150961435713366, 1.3487737656503285, 1.5373847423882254, 1.6834955113845287
, 1.7428183450691863, 3.19059686815754, 4.069642197910212],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=8][Mean=2.407]": [
1.251272850294735, 1.6575634282454268, 1.7659993069884081, 1.9422025859831171,
 1.9950609212573849, 3.8529066825700236, 4.383788095280208],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=9][Mean=2.567]": [
1.0313624486121458, 1.3733933062775086, 2.0053360557798334, 2.3270417610508543
, 2.3789200233822942, 4.159174966388082, 4.696445153641906],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=10][Mean=2.83]": [
0.7772029815988407, 1.6649426924661184, 2.3480144044413302, 2.5703704709084434
, 2.6198023827866863, 4.691566815311088, 5.135574798923891],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=11][Mean=3.062]": [
0.5891950532158767, 1.970667455453252, 2.5794337919120993, 2.9379444763617695,
 2.9840297302785075, 4.951165520231503, 5.418662346363328],
"[WKNN][iBeacon][37][-1][auto][minkowski][k=12][Mean=3.255]": [
0.3581539350829652, 2.299513577849147, 2.824681049458158, 3.160365214026323,
3.207892779638153, 5.219186185578282, 5.714864166119052]}
```


Figura 98: Campo *Mean_error* de la entidad *Method_Prediction*

11.2. Análisis y comparativa de métodos IPS

En esta sección se realizará una breve comparativa entre los diferentes métodos aplicados en *IPSCampaignManager*.

Para ello se van a comparar gráficamente la exactitud y la precisión de los métodos implementados: WkNN (Figura 99), SVR (Figura 100), NuSVR (Figura 101) y LinearSVR (Figura 102).

Los criterios de la aplicación de métodos IPS son los siguientes:

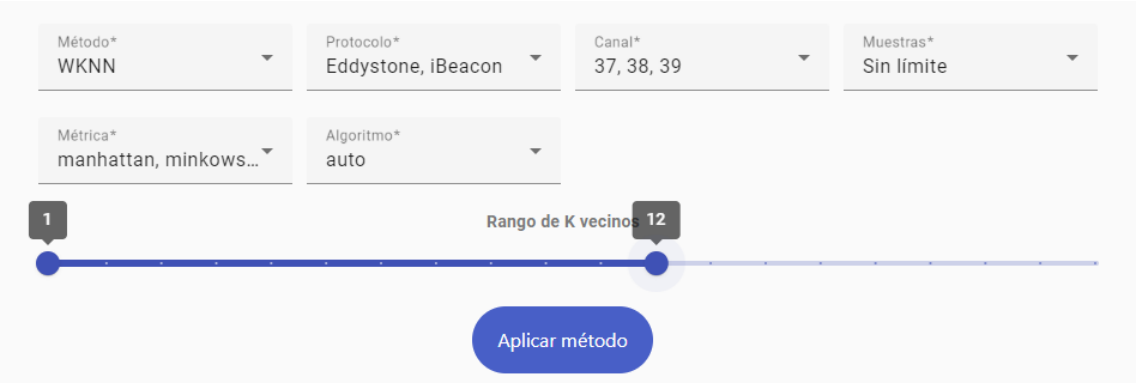

Figura 99: Aplicación del método WkNN

Figura 101: Aplicación del método SVR

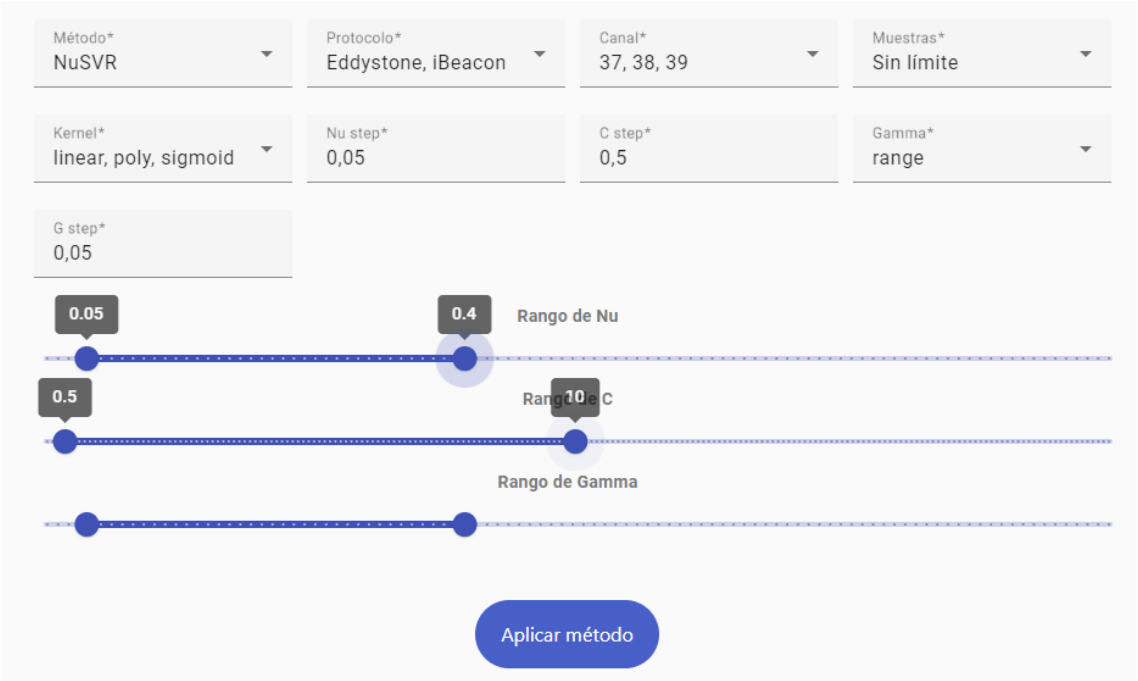

Figura 100: Aplicación de método NuSVR

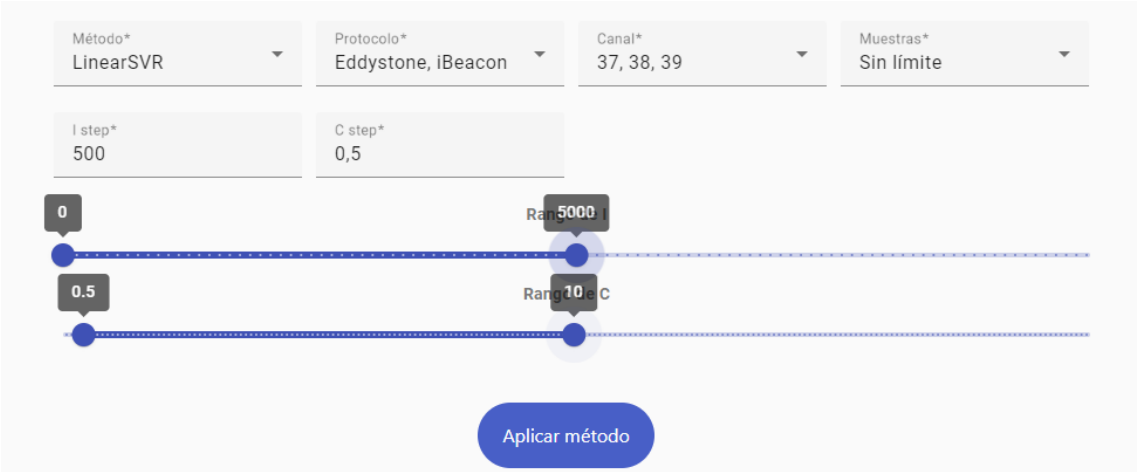

Figura 102: Aplicación del método LinearSVR

Si bien se han obtenido resultados satisfactorios en las pruebas realizadas, es importante tener en cuenta que no se ha llevado a cabo una comparación rigurosa entre todas las variantes de métodos y sus hiperparámetros. Esto significa que existen posibilidades de encontrar configuraciones aún mejores para el posicionamiento en interiores.

En particular, el método WKNN (Figura 103) ha demostrado ser el que proporciona los mejores resultados hasta el momento.

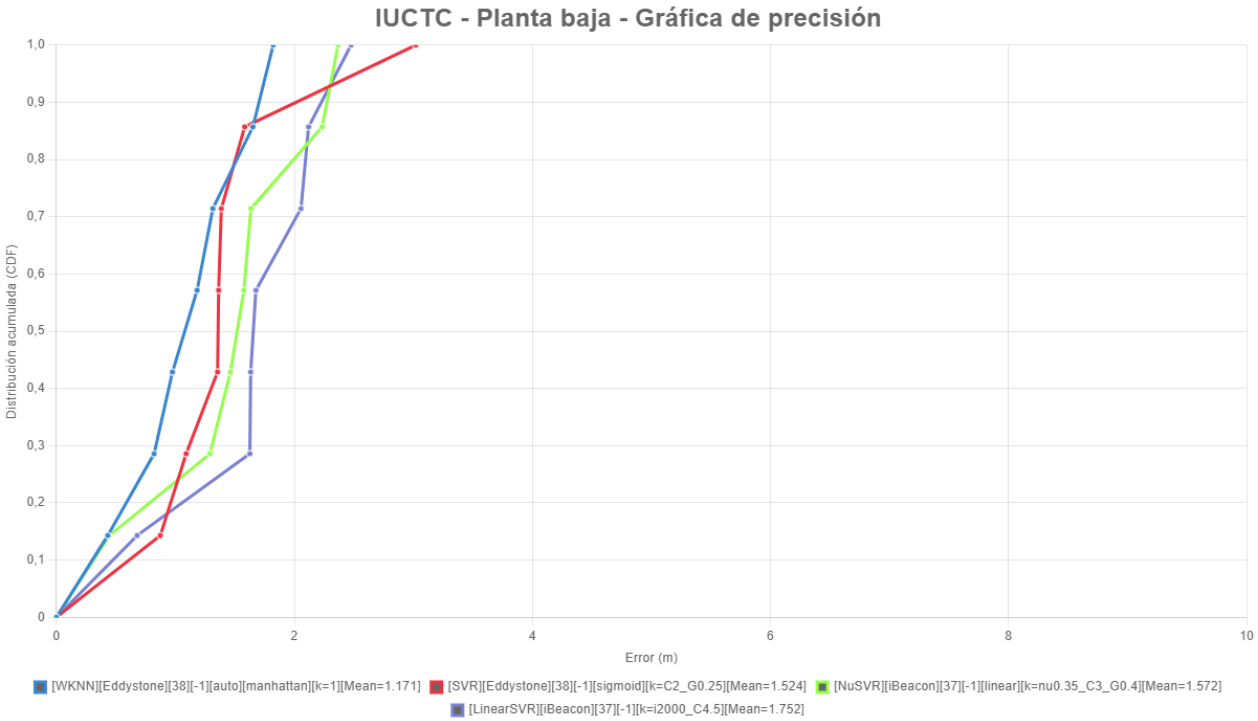

Figura 103: Gráfica de precisión global

A continuación, es posible observar las gráficas de exactitud entre los puntos reales y los referentes a cada uno de los métodos seleccionados: WkNN (Figura 104), SVR (Figura 105), NuSVR (Figura 106) y LinearSVR (Figura 107).

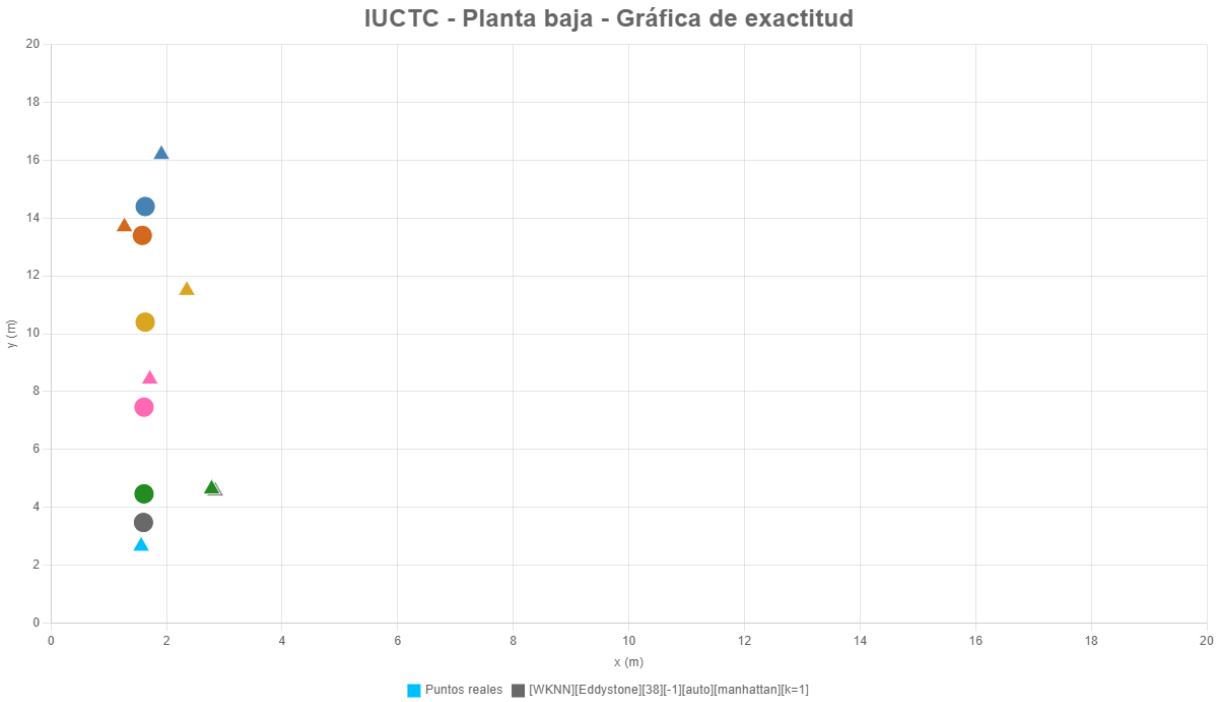

Figura 104: Gráfica de exactitud WkNN

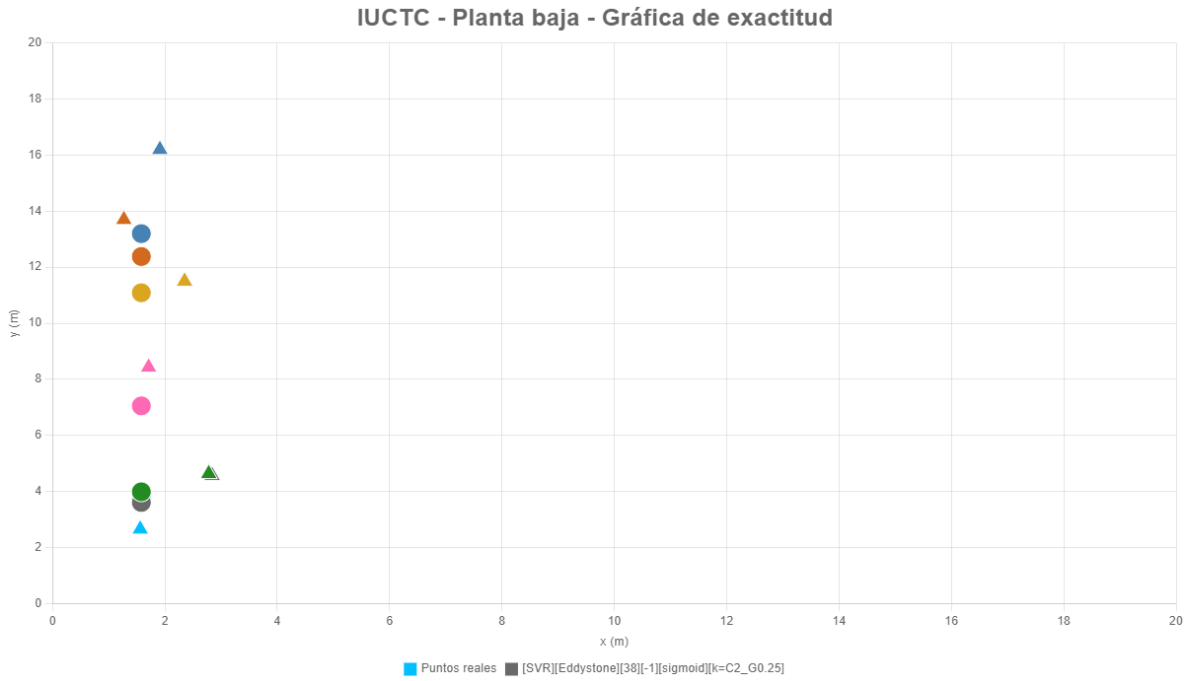

Figura 105: Gráfica de exactitud SVR

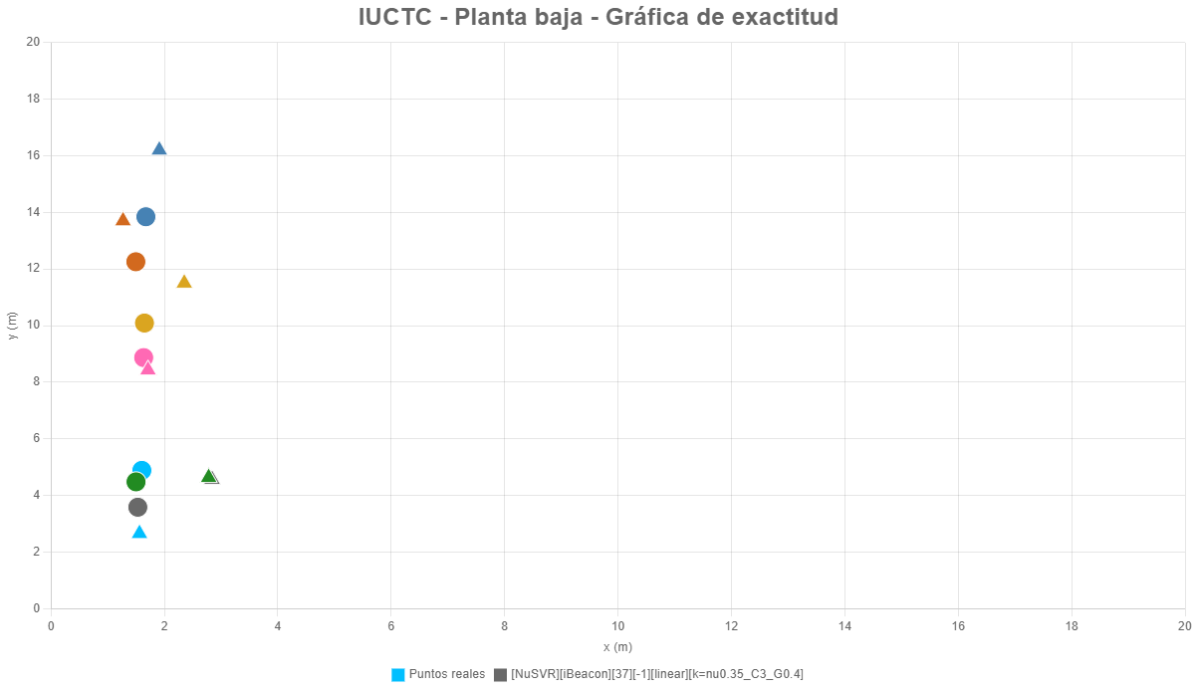

Figura 106: Gráfica de exactitud NuSVR

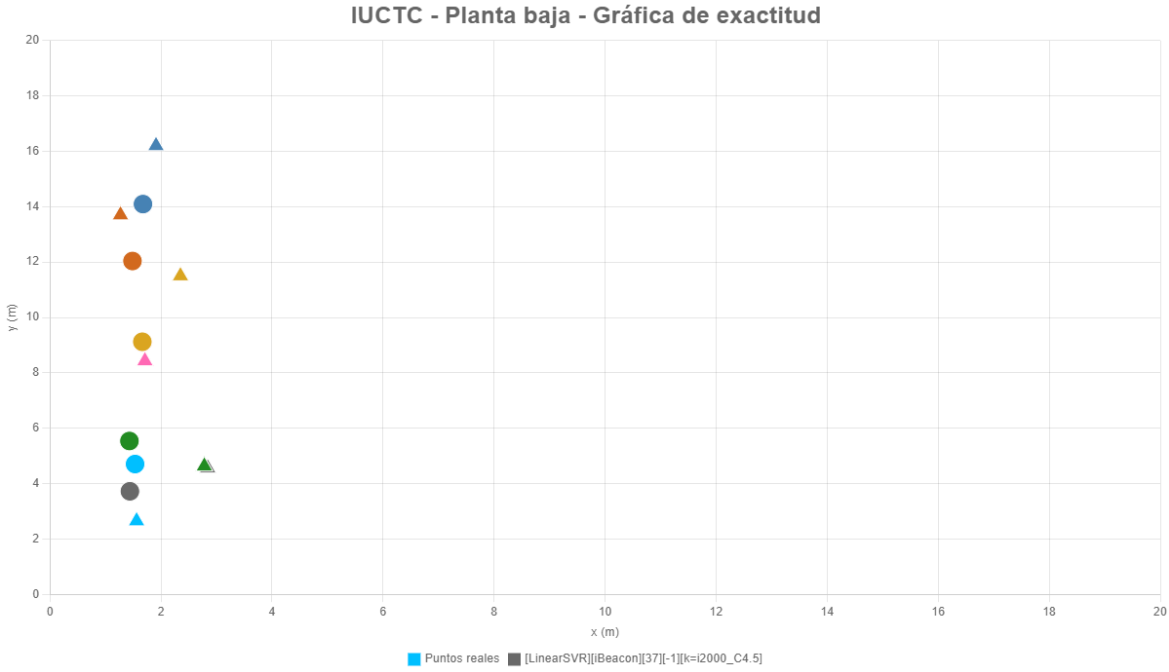

Figura 107: Gráfica de exactitud LinearSVR

12. Conclusiones y aspectos futuros

12.1. Conclusiones

Tras la finalización de este trabajo, es posible afirmar que se han cumplido los objetivos propuestos inicialmente. La nueva herramienta, *IPSCampaignManager*, está completamente operativa y ha pasado a ser parte del proyecto *Robomap*. El desarrollo de esta herramienta ha sido todo un desafío, tanto por la complejidad del procesamiento de la información requerido como por la necesidad de cumplir con los requisitos establecidos por el IUCES.

A lo largo de este proyecto, se ha adquirido un conocimiento profundo sobre cómo construir desde cero un sistema robusto, eficiente y bien organizado. Esto ha tenido un impacto positivo en el aprendizaje personal y ha permitido desarrollar habilidades en programación de software de manera significativa.

Uno de los mayores logros obtenidos es poder contribuir activamente al proyecto *Robomap*. El esfuerzo y entusiasmo invertidos en el desarrollo de la herramienta tienen un propósito importante al brindar una solución completa y beneficiosa para el proyecto en su totalidad. Esto permite que el proyecto siga avanzando y evolucionando, construyendo sobre la base sólida establecida por la herramienta desarrollada.

En definitiva, ha sido un verdadero orgullo formar parte de este proyecto y contribuir al desarrollo de *Robomap*. A lo largo de este camino, se han adquirido conocimientos valiosos y los desafíos enfrentados nos han permitido crecer tanto a nivel profesional como personal.

12.2. Aspectos futuros

La herramienta es altamente estable y sienta las bases para futuros desarrollos que podrían agregar nuevas funcionalidades de gran interés:

- Mejoras en las gráficas interactivas que se obtienen como resultado final tras aplicar diferentes métodos IPS: agregar segmentos de unión entre los puntos reales y los predichos en las gráficas de precisión; ordenar valores tras aplicar filtros según su error promedio y facilitar la búsqueda de mejores resultados; destacar los puntos con menor error promedio, etc.
- Potenciación de los métodos de posicionamiento actuales mediante técnicas de regularización y optimización de hiperparámetros.
- Exploración de nuevos métodos o variantes que puedan mejorar el rendimiento de los sistemas de posicionamiento, como por ejemplo indagar en el uso de redes neuronales para mejorar las predicciones en el procesamiento de datos.
- Posibles mejoras de la interfaz web: otorgar funcionalidad para seleccionar una imagen por defecto en las campañas; dar la posibilidad de asignar denominaciones a grupos de configuraciones de beacons para facilitar su identificación; editar los datos generales de la campaña; dar capacidad para crear configuraciones IPS predeterminadas y aplicar de forma más rápida grupos de métodos de posicionamiento conocidos a nuevas campañas.

13. Fuentes de información

[1] *Situm*, “Sistemas de localización en interiores”, 24 de febrero de 2022, [En línea]. Disponible en: https://situm.com/es/blog/posicionamiento-en-interiores/sistemas-de-localizacion-en-interiores/. Accedido: febrero de 2023.

[2] *Telefónica Empresas*, “Principales aplicaciones de las soluciones de localización en interiores”, 5 de agosto de 2014, [En línea]. Disponible en: https://www.telefonicaempresas.es/grandes-empresas/blog/principales-aplicaciones-de-las-soluciones-de-localizacion-en-interiores/. Accedido: febrero de 2023.

[3] *Unigis*, “Posicionamiento Indoor”, 9 de julio de 2018, [En línea]. Disponible en: https://www.unigis.es/posicionamiento-indoor/. Accedido: febrero de 2023.

[4] G. De Blasio, A. Quesada-Arencibia, C. R. García, J. C. Rodríguez-Rodríguez and R. Moreno-Díaz, "A Protocol-Channel-Based Indoor Positioning Performance Study for Bluetooth Low Energy," in *IEEE Access, vol. 6, pp. 33440-33450, 2018*, [En línea]. Disponible en: https://ieeexplore.ieee.org/document/8360100. Accedido: febrero de 2023.

[5] *Beaconstac*, “Eddystone and iBeacon”, [En línea]. Disponible en: https://www.beaconstac.com/ibeacon-and-eddystone. Accedido: marzo de 2023.

[6] *BorealTech*, “Beacons BLE: información clave sobre esta tecnología”, 28 de septiembre de 2021, [En línea]. Disponible en: https://borealtech.com/beacons-ble-informacion-clave-sobre-esta-tecnologia/. Accedido: mayo de 2023.

[7] *Wireshark*, [En línea]. Disponible en: https://www.wireshark.org/. Accedido: junio de 2023.

[8] *Wireshark*, “Tshark Manual Page”, [En línea]. Disponible en: https://www.wireshark.org/docs/man-pages/tshark.html. Accedido: junio de 2023.

[9] *MathWorks*, [En línea]. Disponible en: https://www.mathworks.com/help/matlab/. Accedido: junio de 2023.

[10] *Criptobyte*, “¿Qué es Matlab y para qué sirve?”, 23 de octubre de 2022, [En línea]. Disponible en: https://www.criptobyte.com/que-es-matlab-y-para-que-sirve/. Accedido: junio de 2023.

[11] *Red Hat*, “¿Qué es y para qué sirve un IDE?”, 20 de enero de 2023, [En línea]. Disponible en: https://www.redhat.com/es/topics/middleware/what-is-ide. Accedido: junio de 2023.

[12] Xuesheng Peng, Ruizhi Chen, Kengen Yu, Feng Ye and Weixing Xue, “An Improved Weighted K-Nearest Neighbor Algorithm for Indoor Localization” in *MDPI AG, 2020* [En línea]. Disponible en: https://www.researchgate.net/publication/347614407_An_Improved_Weighted_K-Nearest_Neighbor_Algorithm_for_Indoor_Localization. Accedido: abril de 2023.

[13] *Wikipedia*, “Precisión y exactitud”, [En línea]. Disponible en: https://es.wikipedia.org/wiki/Precisi%C3%B3n_y_exactitud. Accedido: junio de 2023.

[14] *Ecured*, “Distancia euclídea”, [En línea]. Disponible en: https://www.ecured.cu/Distancia_eucl%C3%ADdea. Accedido: abril de 2023.

[15] *TechTarget*, “API endpoint”, ocubre de 2021, [En línea]. Disponible en: https://www.techtarget.com/searchapparchitecture/definition/API-endpoint. Accedido: febrero de 2023.

[16] *LA REPÚBLICA INDEPENDIENTE DE TU SOFTWARE*, “Ciclo de vida PROTOTIPADO”, [En línea]. Disponible en: http://is3ados.blogspot.com/2008/11/los-ciclos-de-vida-del-software-ii.html. Accedido: marzo de 2023.

[17] *Red Hat*, “¿Qué es Docker?”, 20 de enero de 2023, [En línea]. Disponible en: https://www.redhat.com/es/topics/containers/what-is-docker?sc_cid=7013a0000026OSAAA2&gclid=CjwKCAjw-b-kBhB-EiwA4fvKrM0j0GBk-mOYzJU8n6RVlAJoddIJsinNq5rgXd1VHxKKhNdPt3kavhoCj0sQAvD_BwE&gclsrc=aw.ds. Accedido: febrero de 2023.

[18] *Profesional Review*, “¿Qué es un SDK? Conoce las herramientas de trabajo de los desarrolladores”, 2 de enero de 2021, [En línea]. Disponible en: https://www.profesionalreview.com/2021/01/02/sdk/. Accedido: mayo de 2023.

[19] *KeepCoding*, “¿Qué es PyPi?”, 11 de mayo de 2023, [En línea]. Disponible en: https://keepcoding.io/blog/que-es-pypi/. Accedido: febrero de 2023.

[20] *OpenWebinars*, “Qué es Flask”, 17 de noviembre de 2017, [En línea]. Disponible en: https://openwebinars.net/blog/que-es-flask/. Accedido: febrero de 2023.

[21] *MuleSoft*, “¿Qué es una API?”, [En línea]. Disponible en: https://www.mulesoft.com/es/resources/api/what-is-an-api. Accedido: febrero de 2023.

[22] *SQLAlchemy*, “ORM”, [En línea]. Disponible en: https://docs.sqlalchemy.org/en/20/orm/. Accedido: marzo de 2023.

[23] *OpenWebinars*, “Qué es NodeJS y para qué sirve”, 4 de septiembre de 2019, [En línea]. Disponible en: https://openwebinars.net/blog/que-es-nodejs/. Accedido: febrero de 2023.

[24] *HubSpot*, “¿Qué es Angular?”, [En línea]. Disponible en: https://blog.hubspot.es/website/que-es-angular. Accedido: febrero de 2023.

[25] *HubSpot*, “Bulma CSS”, 7 de diciembre de 2022, [En línea]. Disponible en: https://blog.hubspot.com/website/bulma-css. Accedido: marzo de 2023.

[26] *KeepCoding*, “¿Qué es SQLite?”, 14 de marzo de 2023, [En línea]. Disponible en: https://keepcoding.io/blog/que-es-sqlite/. Accedido: febrero de 2023.

[27] *Microsoft*, “Visual Studio”, [En línea]. Disponible en: https://visualstudio.microsoft.com/es/. Accedido: febrero de 2023.

[28] *Kinsta*, “Git vs GitHub: ¿Cuál es la Diferencia y cómo Empezar?”, [En línea]. Disponible en: https://kinsta.com/es/base-de-conocimiento/git-vs-github/. Accedido: junio de 2023.

[29] *Docker*, “Install Docker Desktop on Windows”, [En línea]. Disponible en: https://docs.docker.com/desktop/install/windows-install/. Accedido: febrero de 2023.

14. Anexos

14.1. Manual de instalación

Para realizar la instalación de la herramienta *IPSCampaignManager* se pueden seguir los siguientes pasos.

1. Instalar Docker Desktop en la máquina cliente [29].
2. Abrir el cuadro de búsqueda e introducir el usuario "simonrusu97".
3. Seleccionar la pestaña *"Images"* (Figura 108).

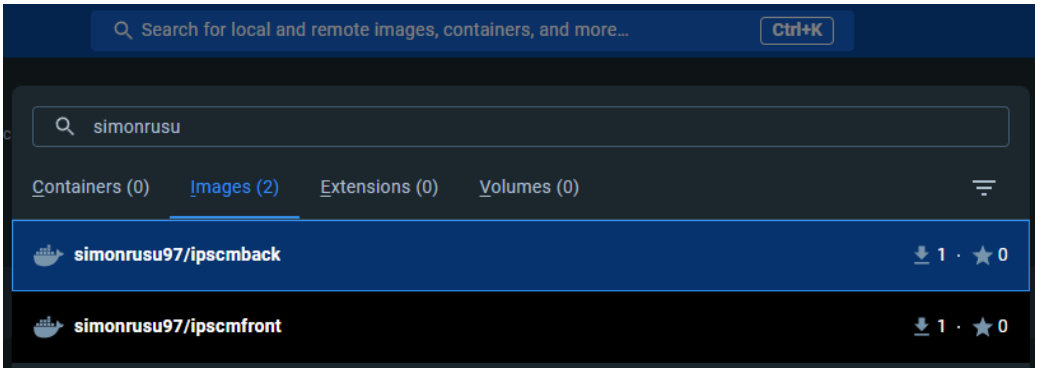

Figura 108: Búsqueda de imágenes en Docker Desktop

4. Descargar las imágenes "*ipscmback*" e "*ipscmfront*" mediante la opción "*pull*" (Figura 109).

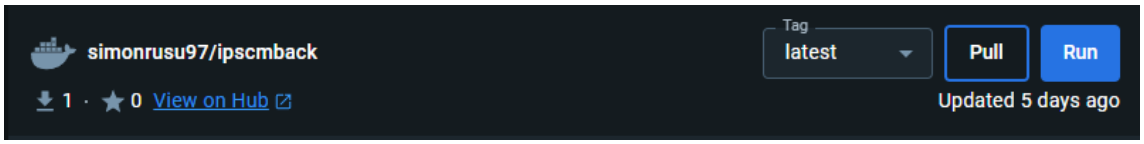

Figura 109: Descarga de imagen en Docker Desktop

5. Abir la pestaña *"Images"* del menú principal y seleccionar la opción "*Run*" para cada una de las imágenes (Figura 110).

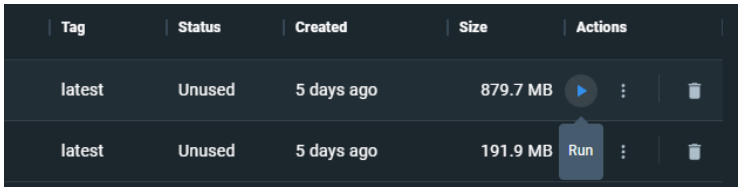

Figura 110: Ejecución de imagen en Docker Desktop

6. Desplegar la pestaña *"Optional settings"*, introducir un nombre de contenedor en el *"Container name"* e introducir el puerto recomendado en *"Host port"* (Figura 111).

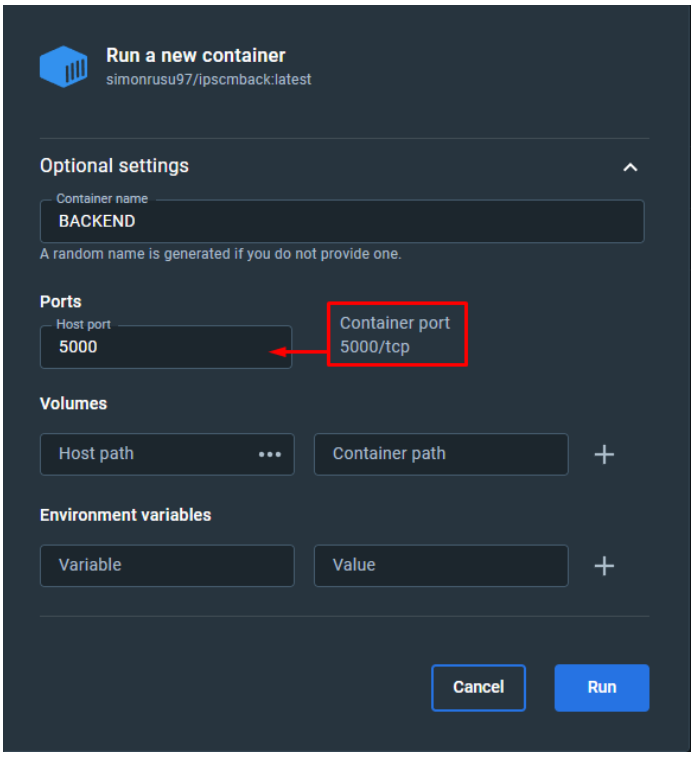

Figura 111: Opciones adicionales de ejecución de imagen en Docker Desktop

7. Repetir el proceso para la otra imagen.
8. Abrir la pestaña *"Containers"* y pinchar encima de la opcion *"Port(s)"* del contenedor asociado a la imagen "*simonrusu97/ipscmfront:latest"* (Figura 112). Este enlace abre la ruta "*localhost"* en nuestro navegador con la página principal del frontend de *IPSCampaignManager.*

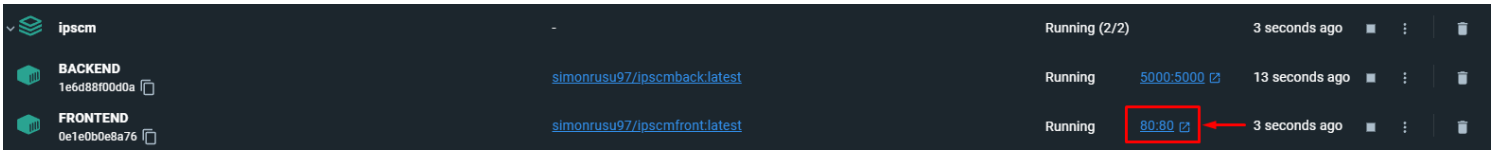

Figura 112: Enlace directo a la aplicación *IPSCampaignManager*

9. En caso de querer parar la aplicación hay que seleccionar la opción *"Stop"* de cada uno de los contenedores (Figura 113). Para reanudar la ejecución seleccionamos la opción *"Start"* de cada contenedor.

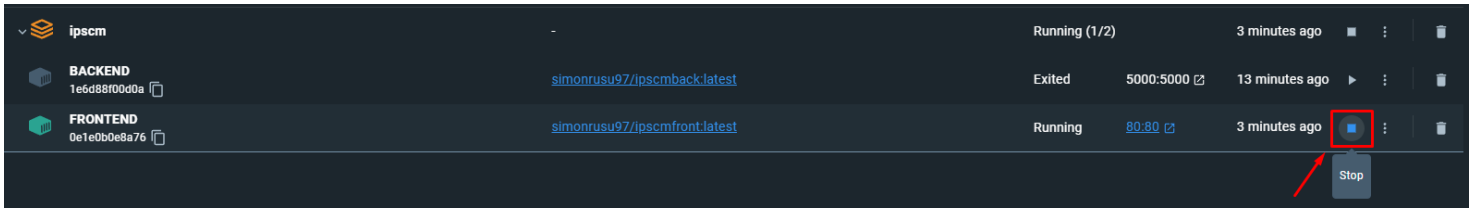

Figura 113: Parar/reanudar un contenedor en Docker Desktop

14.2. Manual de usuario

14.2.1. Importar campaña

Para importar una campaña debemos seleccionar la opción *"Importar campaña"* desde el menú de navegación (Figura 114).

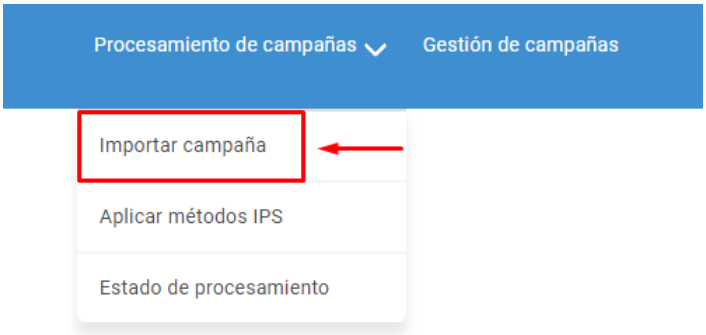

Figura 114: Selección de opción "*Importar campaña*"

Una vez en la pantalla de subida de campaña (Figura 117) debemos cumplimentar una serie de campos:

- Nombre de campaña (obligatorio): denominación que se le va a dar a la campaña.
- Fecha (obligatorio): fecha asociada a la campaña. Se selecciona el día actual de forma predeterminada.
- Descripción: descripción que se asocia a la campaña.

- Sección de subida de archivos: los archivos pueden arrastrarse a la sección de subida correspondiente o se pueden seleccionar desde el explorador de archivos.
    - Pareja de campaña de datos (obligatorio): deben subirse de forma ordenada la base de datos de puntos aleatorios y posteriormente la base de datos de puntos de referencia.
    - Ficheros de configuración de Robomap (obligatorio): deben subirse de forma ordenada el fichero de configuración de *Robomap*, el fichero de configuración con los puntos aleatorios y, por último, el fichero de configuración con los puntos de referencia.
    - Imágenes: conjunto de imágenes asociadas a la campaña.

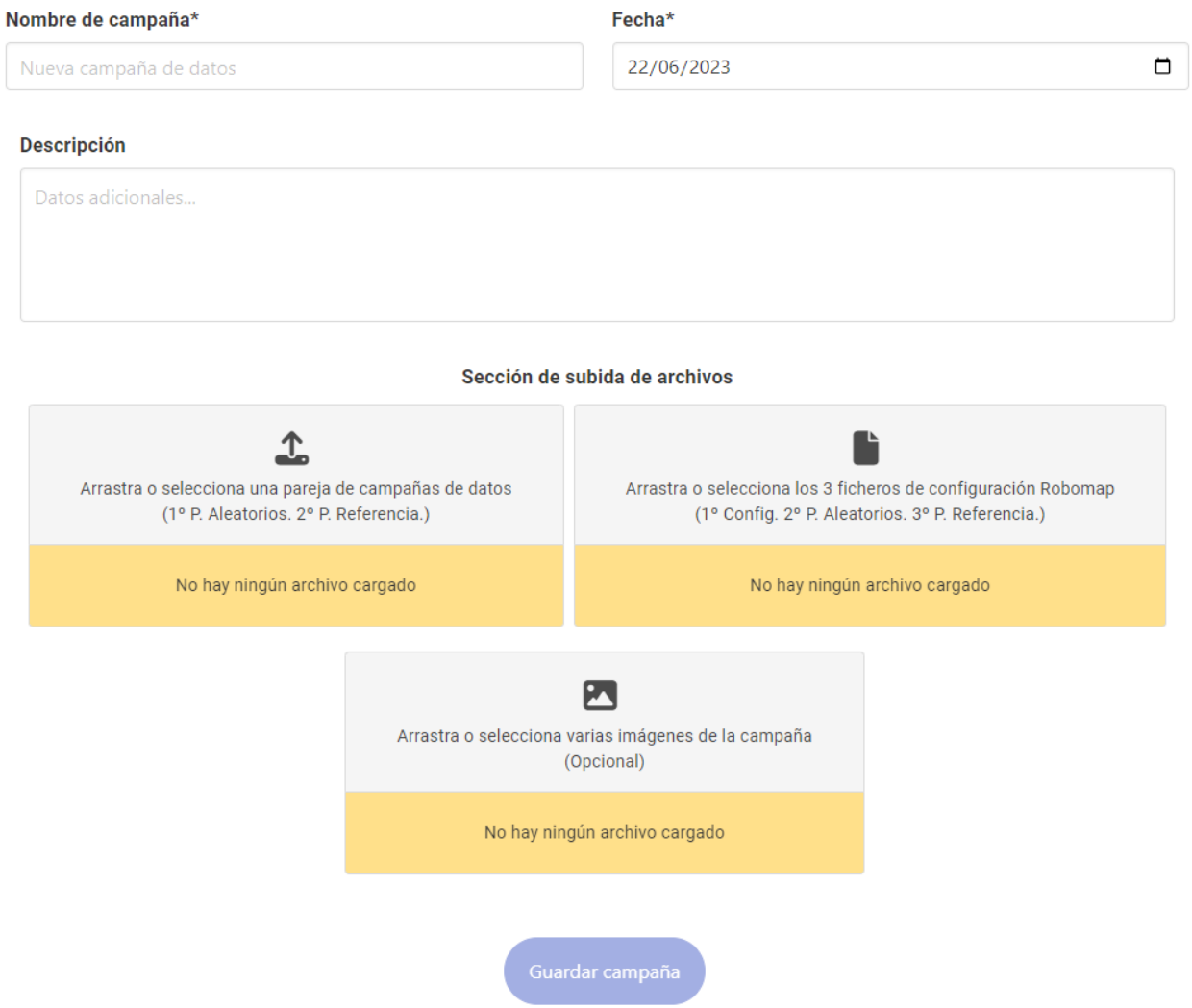

Figura 115: Pantalla de subida de campaña

Una vez que se hayan seleccionado los archivos en la sección de carga, el sistema mostrará una lista de los archivos que se subirán a la herramienta (Figura 118).

Figura 116: Archivos seleccionados

Todas las secciones de subida poseen validación de formato y en caso de intentar subir un archivo diferente al solicitado mostrará un mensaje de error (Figura 117).

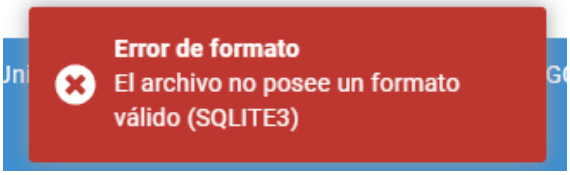

Figura 117: Validación de formato de sección de archivos

Tras seleccionar la opción *"Guardar campaña"* el sistema mandará los datos al servidor y se mostrará un mensaje de carga (Figura 118). En cuanto finalice la subida de campaña se mostrará un mensaje de éxito (Figura 119), en caso contrario recibiremos un mensaje de error (Figura 120).

Nota: no es posible importar múltiples campañas de forma simultánea.

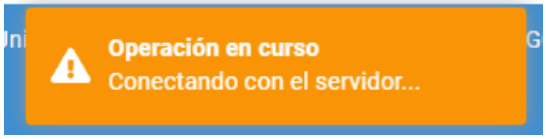

Figura 120: Mensaje de operación en curso

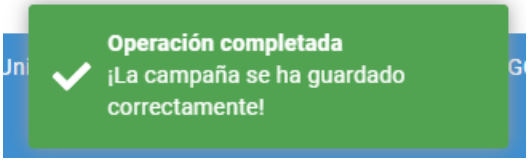

Figura 119: Mensaje de operación completada

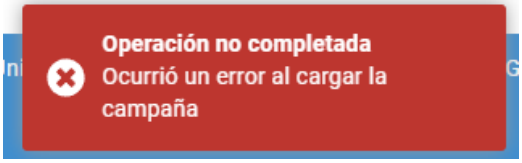

Figura 118: Mensaje de operación no completada

La herramienta nos redirigirá a la sección de gestión de campañas y podemos observar la nueva compaña de datos (Figura 121).

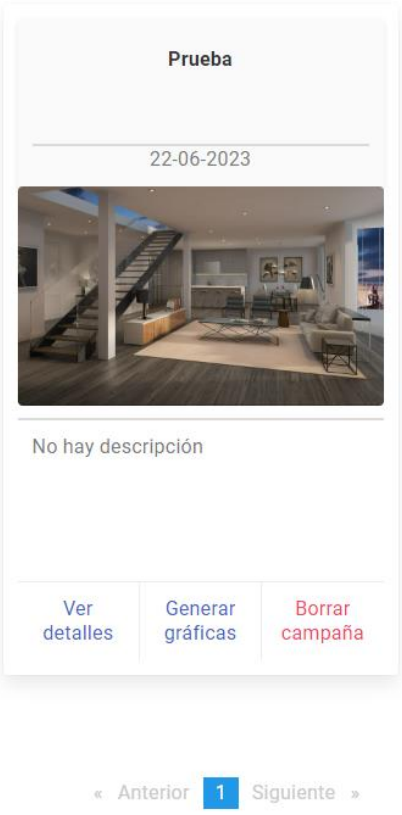

Figura 121: Nueva campaña generada

14.2.2. Aplicar métodos IPS

Para aplicar un nuevo método IPS seleccionamos la opción "Aplicar método IPS" desde el menú de navegación (Figura 122).

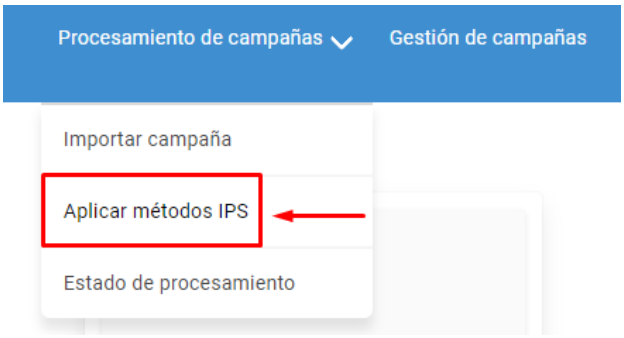

Figura 122: Selección de opción "*Aplicar métodos IPS*"

Una vez estemos en la pantalla correspondiente (Figura 123), podemos elegir entre aplicar diferentes métodos e hiperparámetros.

- Método: WkNN, SVR, NuSVR o LinearSVR
- Protocolo: Eddystone o iBeacon
- Canal: 37, 38 y 39
- Muestras: entre 1 y 20 muestras o sin límite

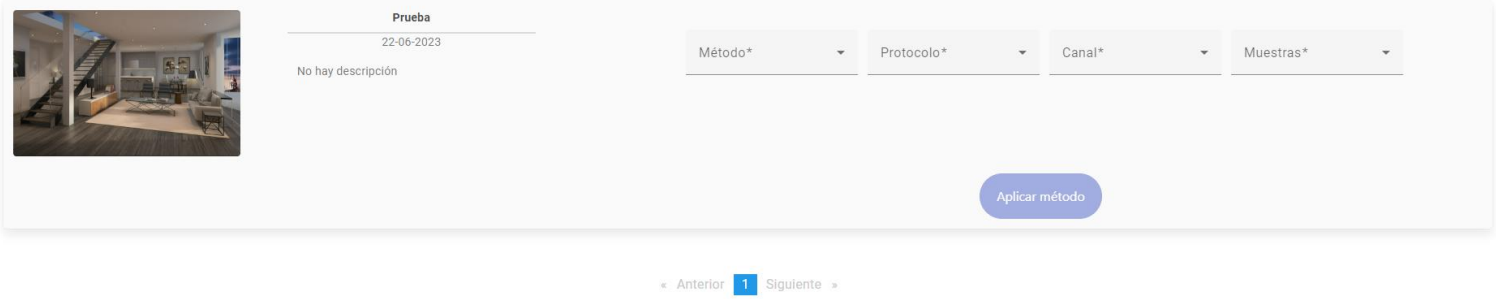

Figura 123: Pantalla de selección de aplicación de método IPS

En caso de seleccionar un determinado método, se muestran los campos con sus hiperparámetros asociados, entre los cuales se encuentran:

WKNN:

- Métrica: manhattan, chebyshev o minkowski
- Algoritmo: brute, ball_tree, kd_tree o auto
- Rango de K vecinos: entre 1 y 20

SVR:

- Kernel: linear, poly, sigmoid o rbf
- C step: valor numérico
- Rango de C: entre 0.1 y 20
- Gamma: auto, scale o range
    - G step: valor numérico
    - Rango de Gamma: entre 0.01 y 1

NuSVR:

- Kernel: linear, poly, sigmoid o rbf
- Nu step: valor numérico
- Rango de Nu: entre 0.01 y 1
- C step: valor numérico
- Rango de C: entre 0.1 y 20
- Gamma: auto, scale o range
    - G step: valor numérico
    - Rango de Gamma: entre 0.01 y 1

LinearSVR:

- I step: valor numérico
- Rango de I: entre 0 y 10000
- C step: valor numérico
- Rango de C: entre 0.1 y 20

Es posible aplicar múltiples métodos de procesamiento de forma simultánea, además el sistema evita aplicar un método con el mismo conjunto de hiperparámetros si ya fue aplicado con anterioridad.

Nota: En caso de aplicar demasiados métodos de forma simultánea, el sistema puede priorizar el procesamiento de datos, lo que podría resultar en una disminución en la capacidad de servir datos a la aplicación Web de manera inmediata.

14.2.3. Ver estado de procesamiento

El usuario tiene la posibilidad de ver en tiempo real el estado de procesamiento de cada método IPS.

Para acceder a la sección correspondiente seleccionamos la opción "Estado de procesamiento" desde el menú de navegación (Figura 124).

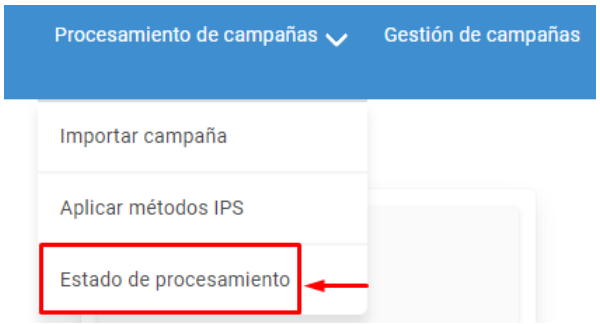

Figura 124: Selección de opción "*Estado de procesamiento*"

Al acceder a la pantalla podemos observar una tabla con los siguientes campos (Figura 125):

- Campaña: nombre de la campaña
- Estado: muestra el estado en el que se encuentra la aplicación del método.
- Descripción de la tarea: muestra los detalles del método aplicado.
- Fecha de inicio: fecha y hora de inicio de la tarea.
- Fecha de fin: fecha y hora de fin de la tarea.

Campaña	Estado	Descripción de la tarea	Fecha de inicio	Fecha de fin
Prueba	Completado	{"method": "SVR", "protocol": "Eddystone", "channel": 37, "rssiSample": -1, "algorithm": 0, "metric": 0, "ksRange": "[0, 0]", "kernel": "linear", "cs": [0.1, 0.5, 0.9, 1.3, 1.7, 2.1, 2.5, 2.9, 3.3, 3.7, 4.1], "gammas": [0.01, 0.06, 0.11, 0.16, 0.21], "nus": [null], "Is": [null]}	22/06/2023 13:43:51	22/06/2023 13:44:03
Prueba	Completado	{"method": "SVR", "protocol": "Eddystone", "channel": 38, "rssiSample": -1, "algorithm": 0, "metric": 0, "ksRange": "[0, 0]", "kernel": "linear", "cs": [0.1, 0.5, 0.9, 1.3, 1.7, 2.1, 2.5, 2.9, 3.3, 3.7, 4.1], "gammas": [0.01, 0.06, 0.11, 0.16, 0.21], "nus": [null], "Is": [null]}	22/06/2023 13:44:03	22/06/2023 13:44:15
Prueba	Obteniendo matriz de puntos de referencia...	{"method": "SVR", "protocol": "Eddystone", "channel": 39, "rssiSample": -1, "algorithm": 0, "metric": 0, "ksRange": "[0, 0]", "kernel": "linear", "cs": [0.1, 0.5, 0.9, 1.3, 1.7, 2.1, 2.5, 2.9, 3.3, 3.7, 4.1], "gammas": [0.01, 0.06, 0.11, 0.16, 0.21], "nus": [null], "Is": [null]}	22/06/2023 13:44:15	

Figura 125: Tareas de procesamiento

14.2.4. Gestión de campañas

El usuario puede ver y gestionar las campañas que se hayan importado, para ello debe acceder a la opción "Gestión de campañas" desde el menú de navegación (Figura 126).

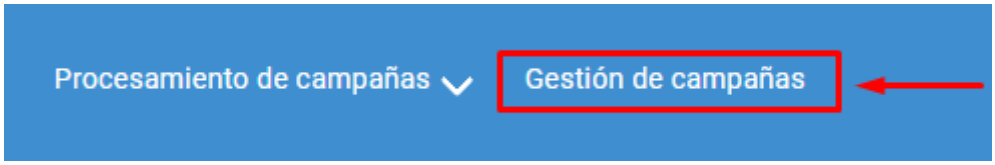

Figura 126: Selección de opción "*Gestión de campañas"*

Una campaña permite ver sus detalles, generar gráficas o eliminarla (Figura 127).

Figura 127: Visualización de campaña

En caso de seleccionar la opción "Borrar campaña", se abrirá un diálogo de confirmación para confirmar la acción (Figura 128).

Figura 128: Diálogo de confirmación de borrado de campaña

Al seleccionar la opción “Ver detales” podemos observar todos los detalles referentes a la campaña (Figura 129).

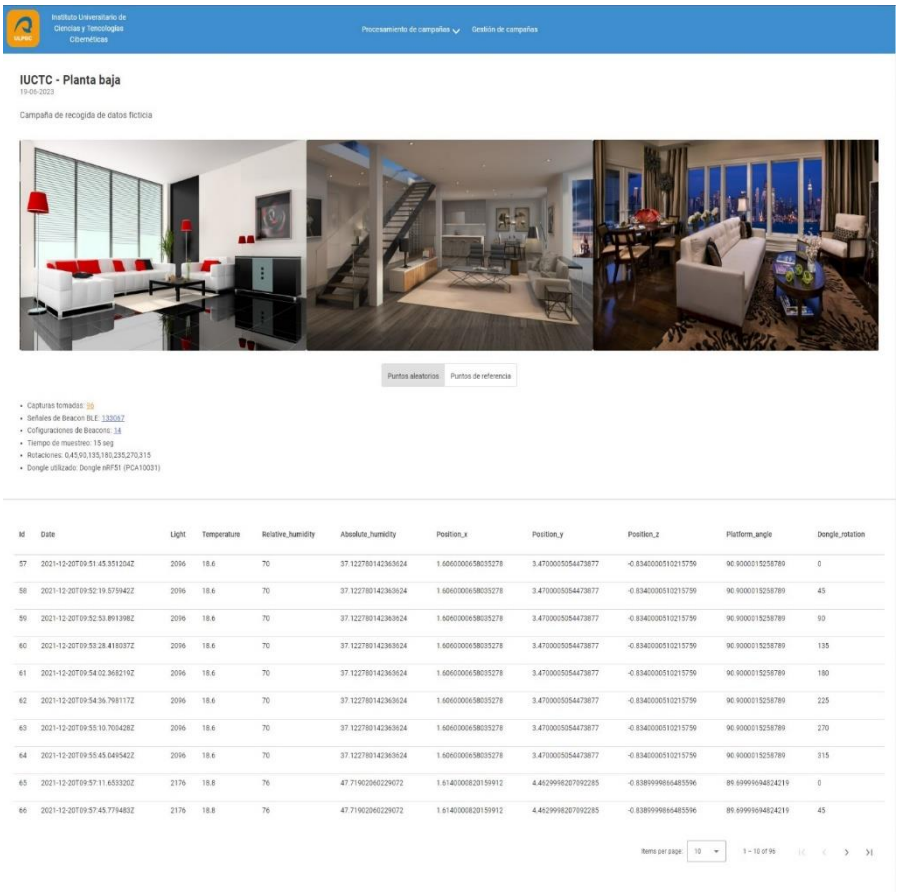

Figura 129: Detalles de campaña

El carrusel de imágenes es completamente funcional y permite navegar entre las diferentes imágenes de la campaña, tanto arrastrando el puntero hacia los extremos del carrusel como mediante los iconos inferiores de desplazamiento (Figura 130).

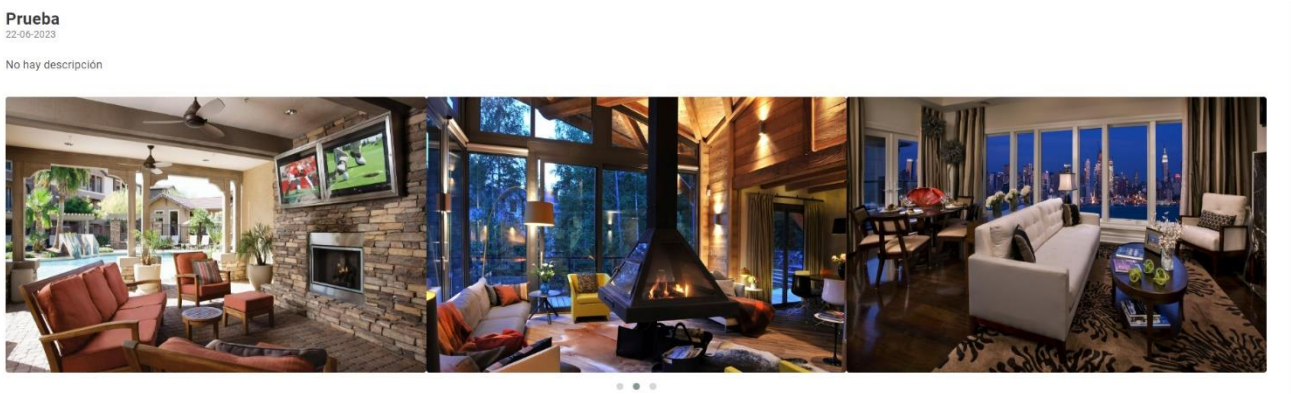

Figura 130: Carrusel de imágenes de campaña

Por otra parte, tenemos la posibilidad de volcar la información de la campaña de puntos aleatorios o puntos de referencia.

También es posible seleccionar que datos se van a previsualizar en la tabla inferior, pudiendo elegir entre: capturas tomadas, señales de Beacon BLE o configuraciones de Beacons (Figura 131).

Figura 131: Opciones de selección de datos de campaña

Al seleccionar la opción “Generar gráficas” se nos muestra la pantalla de generación de gráficas. Podemos seleccionar generar dos tipos de gráficas: de precisión y exactitud (Figura 132).

Figura 132: Tipos de gráficas

En la parte superior es posible aplicar filtros para volcar determinadas series de datos (Figura 133).

Figura 133: Filtros de series para el método WKNN

Si se aplican filtros con poca restricción, es posible que se genere un gran volumen de resultados en función de los métodos aplicados anteriormente. Para evitar esta situación, el sistema cuenta con una limitación de inserción de 30 series de forma simultánea. Si se supera este límite, se mostrará un mensaje de advertencia. En este caso, es necesario reducir el número de resultados restringiendo los criterios de búsqueda para obtener una cantidad más manejable de series. Esto permite obtener resultados más relevantes y evita una sobrecarga de información para el usuario (Figura 134).

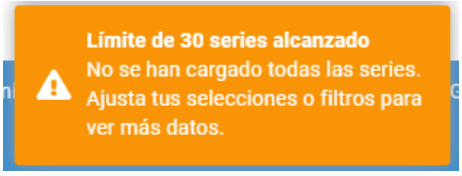

Figura 134: Mensaje de advertencia por limitación de series

Tras volcar el primer grupo de series, estas aparecerán deshabilitadas en la zona de la leyenda de la gráfica (Figura 135).

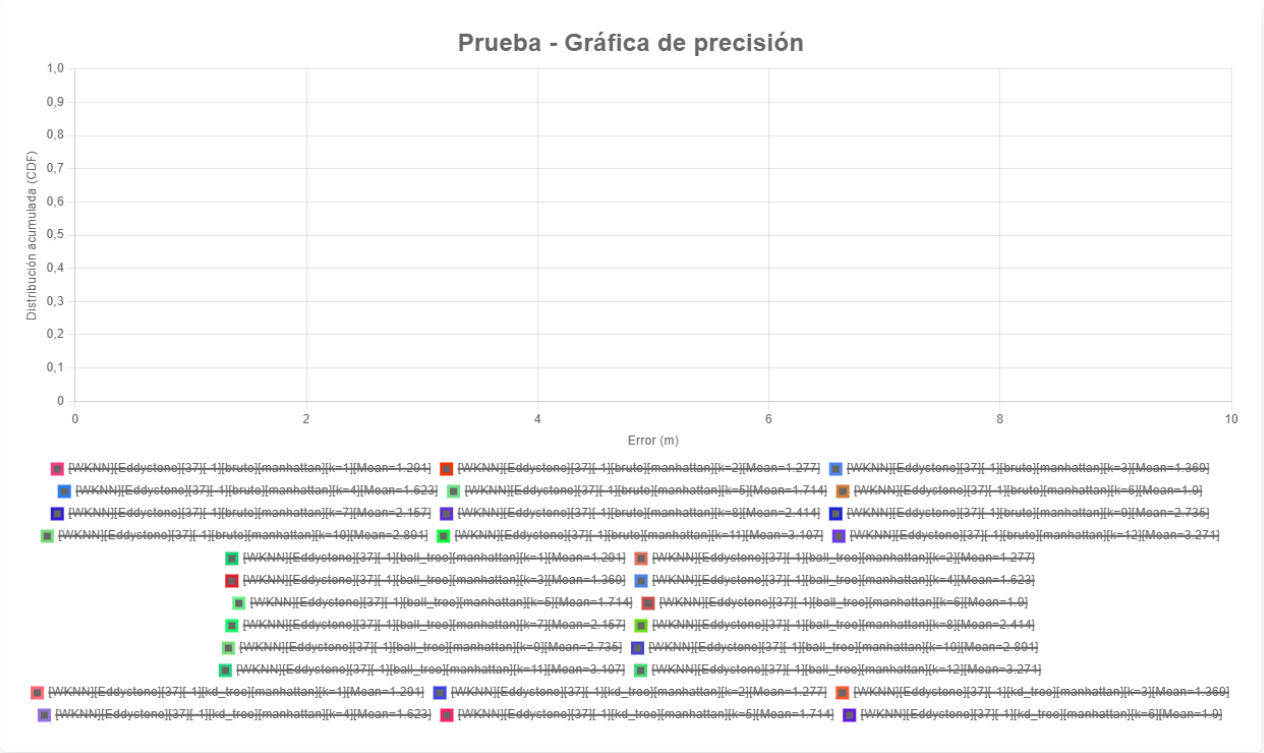

Figura 135: Volcado de series en gráfica de precisión

El sistema permite al usuario seleccionar las series que desee visualizar en la gráfica mediante un mecanismo de selección interactiva (Figura 136). Este sistema permite al usuario seleccionar series y acumularlas en cada aplicación de filtrado que realice, con lo cual se da la posibilidad de hacer una comparación dinámica entre todos los métodos existentes.

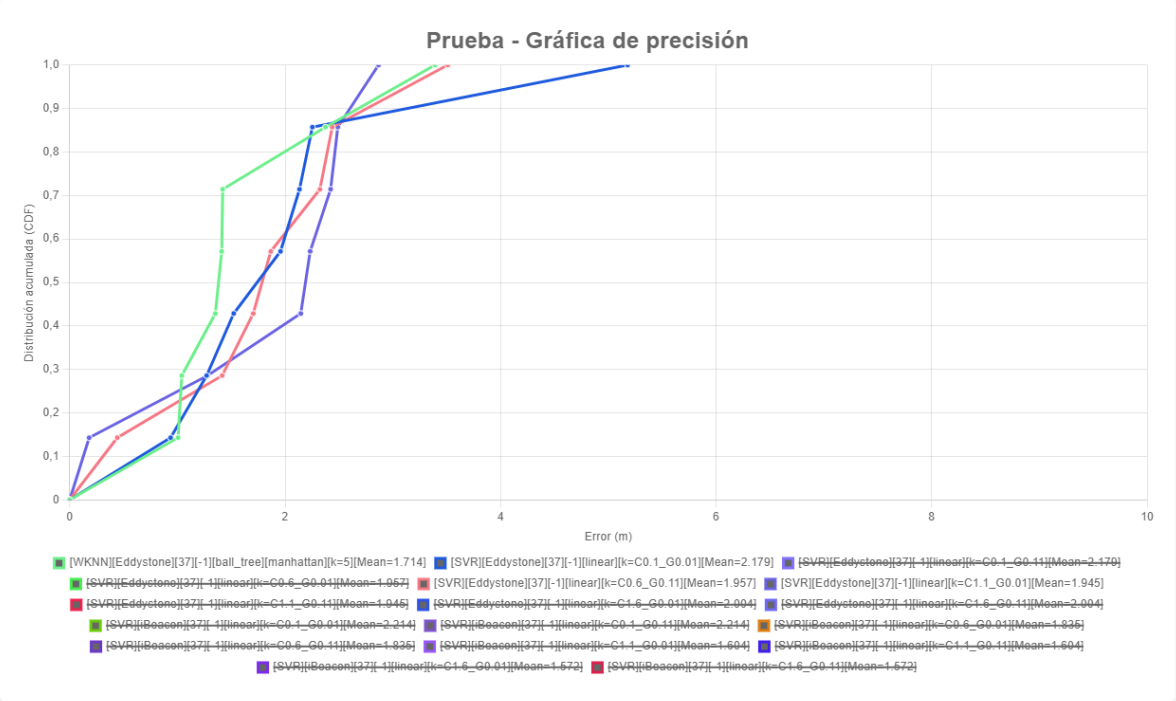

Figura 136: Comparación entre diferentes métodos

Por otra parte, la gráfica de exactitud muestra de forma automática los puntos reales de la serie y los puntos predichos, lo que facilita una comparación visual de la precisión del método utilizado (Figura 137).

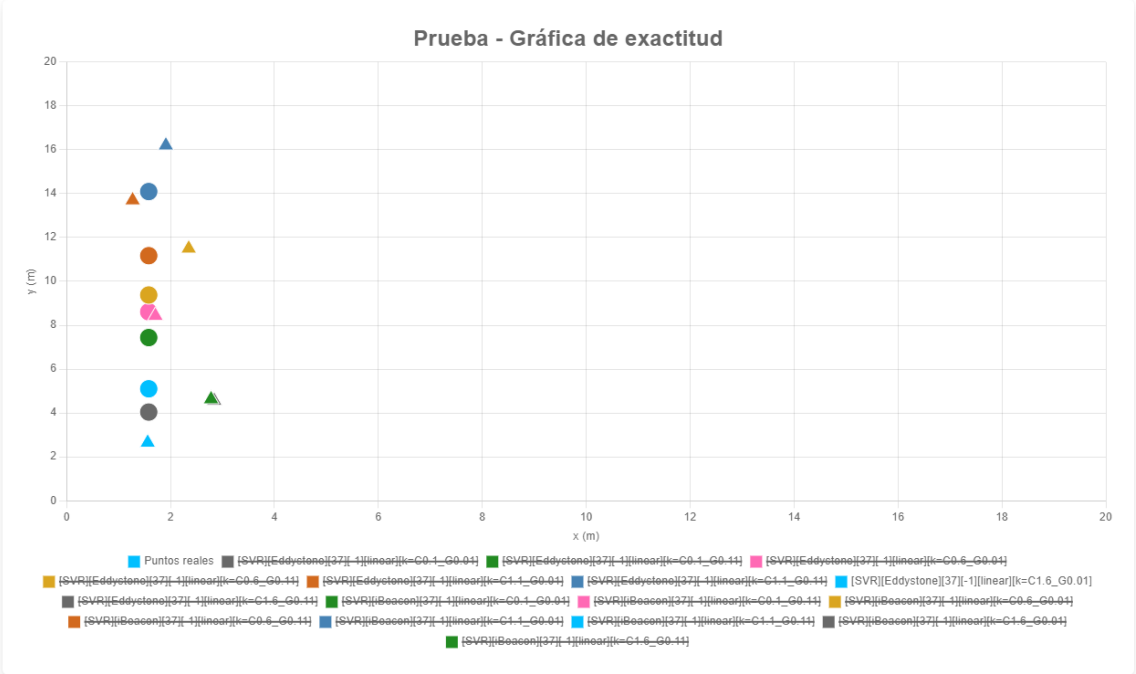

Figura 137: Comparación entre puntos reales y predichos

Tanto en las gráficas de precisión como en las de exactitud, al colocar el cursor sobre un punto específico, se muestra el valor correspondiente de dicho punto dentro de la serie (Figura 138).

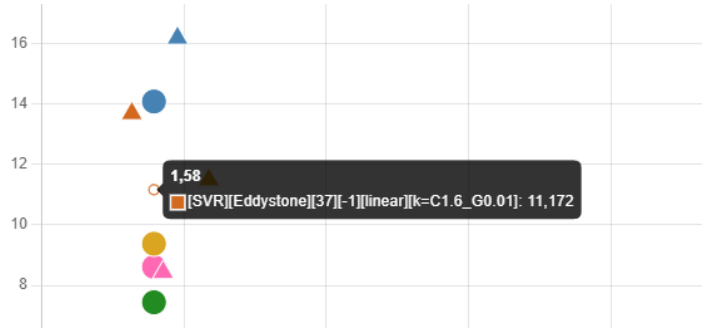

Figura 138: Etiqueta con información del punto

Por último, las gráficas se pueden exportar haciendo clic derecho sobre la gráfica y seleccionando la opción "Guardar imagen como..." (Figura 139).

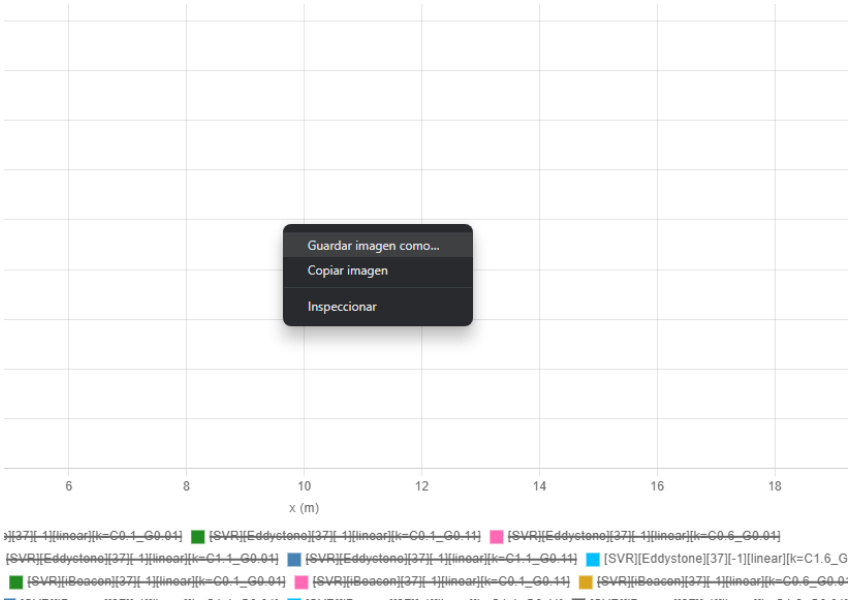

Figura 139: Exportación de gráficas

14.3. Exportación de base de datos de IPSCampaignManager

En caso de necesitar acceder a la base de datos de la herramienta es posible hacerlo mediante los siguientes pasos:

1. Acceder a la pestaña *"Containers"* y seleccionar la opción *"View files"* del menu contectual del contenedor asociado a la imagen "*simonrusu/ipscmback:latest"* (Figura 140).

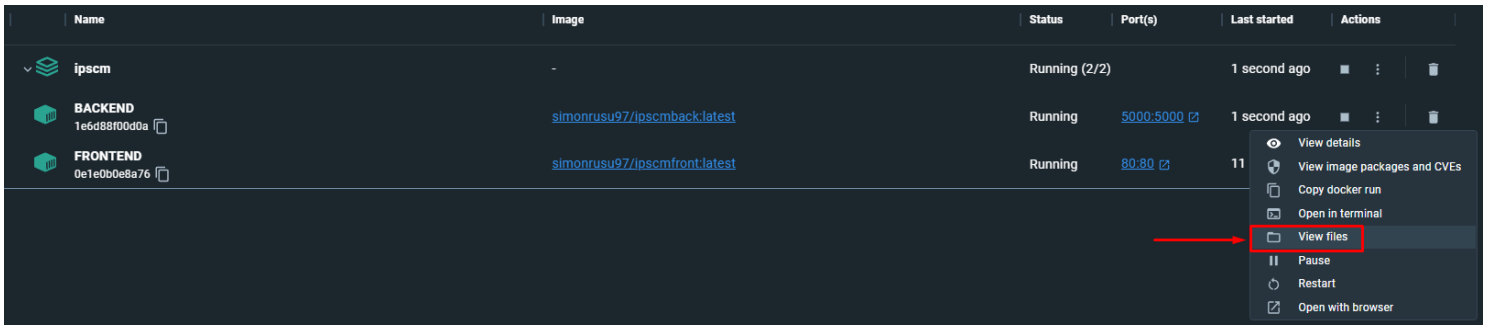

Figura 140: Opción *view files* del Contenedor en Docker Desktop

2. Acceder a la ruta *"/app/db/IPSCM.sqlite3"* y seleccionar la opción *"Save"* para guardar el fichero (Figura 141).

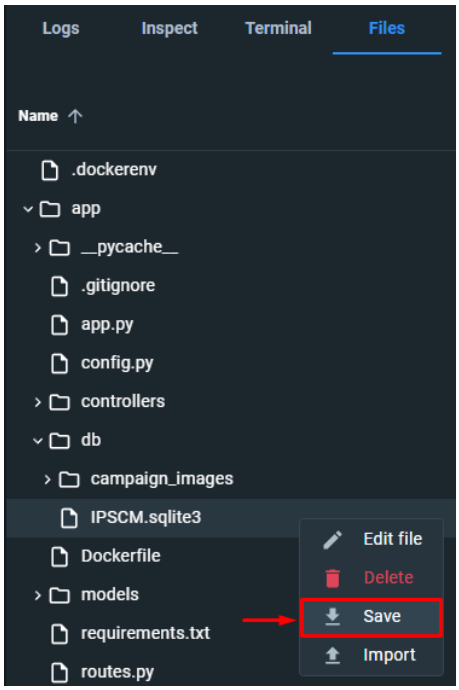

Figura 141: Guardado de archivo en Docker Desktop

14.4. Formato de ficheros de configuración

Al realizar una subida de campaña se deben proporcionar tres ficheros de configuración adaptados a la herramienta *IPSCampaignManager*. Estos ficheros contienen la siguiente estructura:

- *Config.conf* (Figura 142): utilizado para recopilar la duración de la recogida de datos y las rotaciones configuradas en *Robomap*.

```
[BLE]
Port = 9000
Address = localhost
LogFile = ./logs/logBLE.txt
Duration = 15

[Orientator]
Port = 32444
Address = localhost
LogFile = ./logs/logOrientador.txt
Rotations = 0,45,90,135,180,235,270,315

[Weather]
Port = 32443
Address = localhost
LogFile = ./logs/logWeather.txt

[Robo Map]
Port = 1998
Address = localhost
LogFile = ./logs/logRoboMap.txt
```

Figura 142: Fichero de configuración *Config.conf*

- *Config_Ale_Points.conf* (Figura 143): se utiliza para indicar los puntos aleatorios que se midieron durante la toma de datos y sus rotaciones.

```
[POINT_0]
X = 1.56
Y = 2.65
Z = 0
Rotation = 90.0

[POINT_1]
X = 2.84
Y = 4.56
Z = 0
Rotation = 90.0

[POINT_2]
X = 2.78
Y = 4.63
Z = 0
Rotation = 90.0

[POINT_3]
X = 1.71
Y = 8.43
Z = 0
Rotation = 90.0
```

Figura 143: Fichero de configuración *Config_Ale_Points.conf*

- *Config__Ref_Points.conf* (Figura 144): se utiliza para indicar los puntos de referencia que se midieron durante la toma de datos y sus rotaciones.

```
[POINT_0]
X = 1.60
Y = 3.47
Z = 0
Rotations = 0.0,45.0,90.0,135.0,180.0,225.0,270.0,315.0

[POINT_1]
X = 1.61
Y = 4.46
Z = 0
Rotations = 0.0,45.0,90.0,135.0,180.0,225.0,270.0,315.0

[POINT_2]
X = 1.50
Y = 5.48
Z = 0
Rotations = 0.0,45.0,90.0,135.0,180.0,225.0,270.0,315.0

[POINT_3]
X = 1.63
Y = 6.46
Z = 0
Rotations = 0.0,45.0,90.0,135.0,180.0,225.0,270.0,315.0
```

Figura 144: Fichero de configuración *Config_Ref_Points.conf*