UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

"EVALUACIÓN DE MÉTODOS DE *CLUSTERING* PARA LA LOCALIZACIÓN GLOBAL DE ROBOTS MÓVILES"

TRABAJO FIN DE GRADO

Febrero - 2026

AUTOR: Daniel Ferrer Amat

DIRECTOR/ES: Mónica Ballesta Galdeano

Míriam Máximo Gutiérrez

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1 Contexto robótica móvil

En la última década, la robótica móvil ha experimentado una evolución significativa, pasando de operar en entornos industriales controlados a integrarse en escenarios dinámicos y no estructurados, como ciudades, hospitales o entornos domésticos. El pilar fundamental que habilita esta autonomía es la capacidad de navegación, la cual permite a un agente inteligente desplazarse de un punto a otro de forma segura y eficiente. Sin embargo, para que un robot pueda trazar una ruta o interactuar con su entorno, primero debe resolver una interrogante crítica: determinar su ubicación precisa en el espacio.

Este desafío, conocido como localización, depende intrínsecamente de la percepción. Los robots modernos están equipados con un abanico de sensores exteroceptivos, desde cámaras RGB que emulan la visión humana hasta sensores LiDAR que capturan la geometría del entorno con precisión milimétrica. Tradicionalmente, la información capturada por estos sensores se procesaba mediante algoritmos geométricos clásicos. No obstante, el auge de la inteligencia artificial ha impulsado un cambio de paradigma hacia el uso de redes neuronales profundas (*deep learning*). Estas redes son capaces de comprimir el gran volumen de datos sensoriales en descriptores latentes: vectores matemáticos compactos que codifican la información semántica y estructural de un lugar.

Aunque la obtención de estos descriptores supone un gran avance, surge un nuevo problema: ¿cómo organizar y dar sentido a esta información sin intervención humana? Aquí es donde cobra un papel fundamental el agrupamiento no supervisado o *clustering*. En el contexto de la localización global, no basta con saber las coordenadas métricas; el robot necesita construir un mapa topológico que le permita reconocer zonas o lugares (por ejemplo, distinguir una cocina de un pasillo, o una acera de una carretera).

Las técnicas de *clustering* permiten automatizar esta categorización, agrupando descriptores similares para segmentar el mapa en regiones coherentes. Sin embargo, la elección del algoritmo adecuado no es trivial, ya que debe enfrentarse a la alta dimensionalidad de los descriptores profundos y a la variabilidad de los entornos. Por tanto, la evaluación de estos métodos se presenta como una línea de investigación esencial para dotar a los robots de una mayor autonomía y robustez en su comprensión del mundo.

1.2 Objetivo del TFG

El objetivo principal de este Trabajo de Fin de Grado es analizar, implementar y validar la eficacia de diferentes algoritmos de agrupamiento no supervisado (*clustering*) aplicados sobre descriptores latentes profundos, con el fin de determinar qué técnica ofrece mayor robustez para la localización global y la categorización de lugares en robótica móvil.

Para alcanzar este propósito general, se definen los siguientes objetivos específicos:

- **Realizar un estudio comparativo de algoritmos:** evaluar el rendimiento de un conjunto diverso de técnicas de agrupamiento, abarcando métodos basados en centroides (*k-means*), densidad (HDBSCAN, *mean shift*), jerarquía (*hierarchica*l) y grafos (*spectral clustering*), para determinar cuál se adapta mejor a la no linealidad de los datos sensoriales.

- **Evaluar la robustez en escenarios heterogéneos:** analizar el comportamiento de estos algoritmos en dos entornos claramente diferenciados y comprobar su versatilidad ante condiciones diversas:

  1. Un entorno de interiores (utilizando la base de datos ScanNet), procesando descriptores generados por la arquitectura Sonata.

  2. Un entorno de exteriores (utilizando la base de datos NCLT), procesando descriptores de nubes de puntos LiDAR obtenidos mediante MinkUNeXt.

- **Analizar la precisión de la localización:** determinar si la estimación posicional derivada de los clústeres ofrece la exactitud necesaria para sistemas de posicionamiento global. Esto implica medir no solo la calidad intrínseca del agrupamiento (homogeneidad semántica), sino también el error métrico de la ubicación estimada respecto a la trayectoria real (*ground truth*).

- **Identificar limitaciones y proponer soluciones:** detectar las carencias de los métodos de agrupamiento en cuanto a precisión métrica y establecer las bases para futuros sistemas de localización híbrida o jerárquica.

1.3 Estructura TFG

El presente documento se organiza en cinco capítulos principales, seguidos de la bibliografía y los anexos correspondientes, estructurados de la siguiente manera:

• **Capítulo 1:** Introducción. Establece el marco contextual de la robótica móvil y la localización, define los objetivos del proyecto y detalla la estructura del presente trabajo.

• **Capítulo 2:** Estado del Arte. Realiza una revisión exhaustiva de los fundamentos teóricos necesarios. Se abordan los conceptos clave de robótica móvil (locomoción, SLAM, planificación) y los sensores más empleados en el contexto de la robótica móvil. Seguido a ello, se detallan las arquitecturas de redes neuronales utilizadas para el procesamiento de nubes de puntos (MinkUNeXt y Sonata) y se profundiza en las técnicas de aprendizaje automático, con especial énfasis en los algoritmos de *clustering* evaluados y las métricas para su validación.

• **Capítulo 3:** Herramientas. Describe el entorno tecnológico y de desarrollo. Se justifican el lenguaje y las bibliotecas de Python empleadas para el preprocesamiento y el aprendizaje automático, así como el entorno de trabajo. Asimismo, se presentan las bases de datos que sustentan la experimentación.

• **Capítulo 4:** Experimentos y resultados. Constituye el núcleo práctico del trabajo, dividido en dos grandes bloques de análisis: *clustering* en entornos interiores y *clustering* en entornos exteriores. En este capítulo se exponen las pruebas realizadas y se comparan los resultados de los distintos algoritmos.

• **Capítulo 5:** Conclusiones y trabajos futuros. Recoge los hallazgos principales, destacando la eficacia del *spectral clustering*, y discute las limitaciones sobre las métricas encontradas. Finalmente, se proponen líneas de investigación futura para la mejora de la navegación.

• **Bibliografía y anexos:** recopilan las referencias académicas citadas, así como el código fuente desarrollado y los resultados complementarios relativos a la selección de hiperparámetros.

2. ESTADO DEL ARTE

2.1 Robótica móvil

La robótica móvil se define como la rama de la ingeniería dedicada al diseño de agentes mecánicos capaces de desplazarse de manera autónoma en entornos no estructurados [1]. A diferencia de los robots industriales estacionarios, los robots móviles operan en condiciones de incertidumbre y deben desenvolverse eficientemente en espacios dinámicos. Esto exige la integración de un ciclo de control robusto que permita al agente percibir, interpretar y actuar sobre el entorno de forma segura [2].

Siguiendo el marco conceptual establecido por Siegwart, Nourbakhsh y Scaramuzza [3], la autonomía en robótica móvil se sustenta en la interacción coordinada de cuatro subsistemas funcionales fundamentales:

2.1.1. Locomoción y cinemática

Este subsistema engloba tanto los mecanismos físicos de desplazamiento como los modelos matemáticos que describen el movimiento del robot. La cinemática establece la relación entre las velocidades impartidas a los actuadores y el movimiento resultante en el plano global [4].

Es importante destacar que la mayoría de los robots móviles terrestres presentan restricciones no holonómicas [5], lo que implica que no pueden desplazarse instantáneamente en cualquier dirección. Un ejemplo típico es un robot con dirección tipo automóvil, que no puede moverse lateralmente sin realizar una maniobra específica. Estas limitaciones condicionan tanto el diseño de los controladores como la planificación de trayectorias.

2.1.2. Percepción y adquisición de datos

La percepción constituye el puente entre el entorno físico y el sistema de control. Su función no se limita a captar datos, sino a interpretarlos para transformarlos en información útil. Sin embargo, ninguna medición es perfecta. La percepción es intrínsecamente parcial y ruidosa debido a factores como la resolución limitada de los sensores o las condiciones ambientales [6].

La incertidumbre inherente en los datos de los sensores hace inviable el uso de modelos deterministas y obliga a adoptar un enfoque probabilístico [7], en el que las mediciones se consideran estimaciones sujetas a error. Este paradigma resulta esencial para filtrar ruido, fusionar información sensorial y mejorar la fiabilidad del sistema.

2.1.3. Localización y mapeo (SLAM)

Para interactuar correctamente con el entorno, el robot debe resolver dos problemas interdependientes:

- **Localización:** consiste en determinar la posición y orientación (pose) del robot dentro de un marco de referencia global. Los métodos básicos, como la odometría, tienden a acumular errores progresivos debido al deslizamiento de las ruedas y otras imperfecciones mecánicas. Esta acumulación de error, conocida como deriva, provoca que la estimación se aleje gradualmente de la posición real, lo que hace necesario complementarla con observaciones externas [8].
- **Mapeo:** implica construir una representación del entorno a partir de los datos sensoriales. Una de las representaciones más utilizadas es la rejilla de ocupación, que discretiza el espacio y clasifica cada celda como libre, ocupada o desconocida [9].

Cuando el robot opera en un entorno a priori desconocido, ambos problemas se vuelven inseparables, dando lugar al desafío del SLAM (*Simultaneous Localization and Mapping*). Este problema presenta una interdependencia: para construir un mapa preciso se necesita conocer la pose exacta del robot, pero para estimar correctamente la pose se requiere un mapa fiable. El SLAM resuelve esta paradoja estimando simultáneamente ambas magnitudes mediante técnicas probabilísticas (como filtros bayesianos u optimización de grafos) que buscan maximizar la coherencia entre el movimiento medido y la percepción del entorno [10].

2.1.4. Navegación y planificación de trayectorias

Una vez que el robot conoce su ubicación y dispone de un mapa del entorno, interviene el subsistema de navegación, cuya misión crítica es guiar al agente hacia su destino de forma eficiente y segura. Este proceso se organiza en dos niveles funcionales complementarios:

- **Planificación Global (*Global Path Planning*):** opera a nivel estratégico y calcula una ruta óptima basándose en el mapa conocido. Para ello se emplean algoritmos clásicos de búsqueda en grafos, como el algoritmo A*, introducido formalmente por Hart, Nilsson y Raphael [11]. Estos métodos generan una secuencia de puntos de paso (*waypoints*) minimizando una función de coste determinada (habitualmente la distancia euclídea o el tiempo) y evitando obstáculos fijos conocidos.
- **Navegación Local y Evitación de Obstáculos:** dado que el entorno puede presentar cambios dinámicos o imprevistos, es necesaria la intervención de un

planificador local que actúe en tiempo real proporcionando un control reactivo. Mientras que el mapa global contiene información estática, los sensores del agente permiten detectar obstáculos dinámicos o modificaciones que no estaban presentes en la planificación inicial.

En este contexto, algoritmos como el *Dynamic Window Approach* (DWA), propuesto por Fox, Burgard y Thrun [12], ajustan continuamente las velocidades lineal y angular del agente para evitar colisiones inminentes. Una vez superado el obstáculo, el controlador local reorienta al robot para retomar la trayectoria definida por el planificador global.

En conjunto, este esquema de navegación híbrida integra la planificación estratégica (largo plazo) con el control reactivo (corto plazo), garantizando un desplazamiento robusto y seguro.

2.2 Sensores

Los sensores en los robots móviles son elementos fundamentales, ya que permiten que el robot pueda percibir lo que ocurre a su alrededor y conocer su propio estado interno. Gracias a ellos, el robot es capaz de tomar decisiones y actuar de manera segura y eficiente en entornos reales, que suelen ser cambiantes y poco estructurados.

Estos dispositivos se pueden clasificar en dos categorías: exteroceptivos y propioceptivos. Los sensores exteroceptivos se encargan de captar información del entorno que rodea al robot, mientras que los propioceptivos obtienen datos sobre el estado interno del propio robot.

La combinación de ambos tipos de sensores es esencial, ya que permite al robot conocer con precisión su posición y orientación (pose), además de interpretar las características del entorno en el que se mueve, lo que hace que pueda tomar buenas decisiones y desenvolverse de forma autónoma [13].

2.2.1 Encoders

Los encoders son sensores propioceptivos electromecánicos que se acoplan directamente a los ejes de los motores o de las ruedas del robot. Su propósito fundamental es transducir el movimiento mecánico angular en señales eléctricas digitales, lo que permite al sistema de control cuantificar con precisión tanto el desplazamiento como la velocidad de rotación de cada actuador.

El principio de funcionamiento de estos dispositivos (típicamente de tipo óptico) se basa

en la interrupción de un haz de luz. El sensor consta de un disco perforado o ranurado solidario al eje de la rueda, una fuente de luz (generalmente un LED infrarrojo) y un fotodetector alineado al otro lado del disco. A medida que la rueda gira, las ranuras del disco permiten o bloquean el paso de la luz alternativamente, generando en el receptor un tren de pulsos cuadrados.

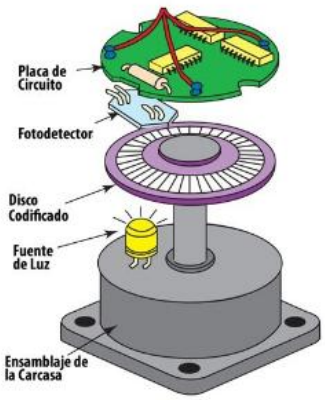

Figura 1: Despiece de los componentes de un encoder óptico incremental.

Para medir no solo la velocidad, sino también el sentido de giro, los encoders actuales incorporan dos fotodetectores que producen dos señales llamadas Canal A y Canal B. Estas señales están desplazadas entre sí 90 grados eléctricos, una disposición conocida como cuadratura. Gracias a este desfase, el microcontrolador puede identificar cuál de los dos canales cambia primero y, con ello, determinar si el robot se mueve hacia adelante o hacia atrás.

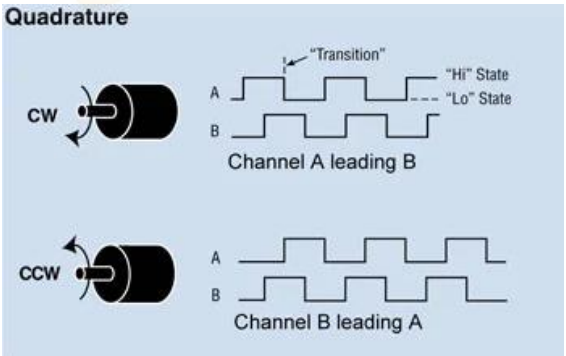

Figura 2: Diagrama de señales en cuadratura para la detección del sentido de giro.

La función principal de los encoders es realizar la odometría, permitiendo calcular la posición relativa (x, y, θ) y la velocidad del robot mediante la integración de los pulsos generados por el giro de las ruedas. Aunque estos datos son esenciales para los bucles de control (PID), este método conlleva una deriva (*drift*) o error acumulativo provocado por deslizamientos e imperfecciones del terreno. Como analizan Borenstein y Feng [14],

esta degradación progresiva de la precisión (debida a errores sistemáticos y no sistemáticos) hace imprescindible no depender únicamente de la odometría, sino fusionar sus datos con sensores externos (como LiDAR o IMU) para corregir la localización en trayectos largos.

2.2.2 Unidad de medición inercial (IMU)

La unidad de medición inercial (IMU) es un sensor que integra múltiples instrumentos de medición en un único encapsulado: un acelerómetro triaxial, un giroscopio triaxial y, frecuentemente, un magnetómetro. Su función primordial es capturar el estado dinámico del robot midiendo las fuerzas inerciales y las velocidades de rotación a las que está sometido el chasis, operando de manera independiente a su interacción con el entorno externo.

El principio de funcionamiento varía según el componente interno. El acelerómetro mide la aceleración lineal y la fuerza de la gravedad, lo que permite determinar la inclinación del robot (ángulos de *pitch* y *roll*). Por su parte, el giroscopio mide la velocidad angular o tasa de giro, esencial para conocer cambios rápidos en la orientación (*yaw*). Finalmente, el magnetómetro mide la intensidad y dirección del campo magnético terrestre, proporcionando una referencia absoluta del norte geográfico. Como detallan Barshan y Durrant-Whyte [15], la combinación de estos datos permite reconstruir la trayectoria del robot basándose en la inercia. Sin embargo, es importante considerar que estos sensores sufren de errores sistemáticos y deriva si no se calibran adecuadamente.

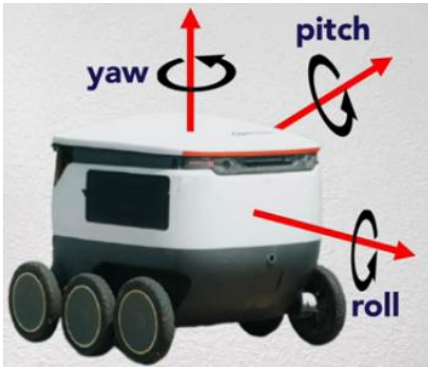

Figura 3: Representación de los ángulos de Euler (pitch, roll y yaw) que definen la orientación del robot en el espacio.

La principal ventaja de la IMU es que, a diferencia de los encoders, sus mediciones no dependen de la interacción física con el suelo. Esto la convierte en un complemento indispensable en entornos exteriores o irregulares. Cuando un robot patina o se desplaza sobre terreno accidentado, la odometría de las ruedas pierde fiabilidad,

mientras que la IMU continúa detectando la orientación real del chasis. Esto permite corregir la posición estimada mediante algoritmos de fusión sensorial, siendo el Filtro de Kalman Extendido (EKF) el estándar actual para integrar estas fuentes de información heterogéneas [16].

2.2.3 Ultrasónicos

Los sensores ultrasónicos son dispositivos exteroceptivos diseñados para medir la distancia entre el robot y los objetos de su entorno mediante el uso de ondas sonoras. A diferencia de los sensores ópticos que dependen de la luz, estos dispositivos operan emitiendo pulsos acústicos de alta frecuencia, lo que los hace inaudibles para el oído humano y les permite funcionar eficazmente independientemente de las condiciones de iluminación del entorno.

Su funcionamiento se basa en el principio de tiempo de vuelo (*time-of-flight*), que consiste en medir el tiempo que tarda un pulso acústico en viajar desde el sensor hasta un objeto y regresar. El transductor emite una onda ultrasónica que, al impactar con un obstáculo, se refleja en forma de eco hacia el origen [17]. Para obtener la distancia d, el sistema multiplica la velocidad del sonido c por el tiempo transcurrido Δt y divide el resultado entre dos, compensando el trayecto de ida y vuelta.

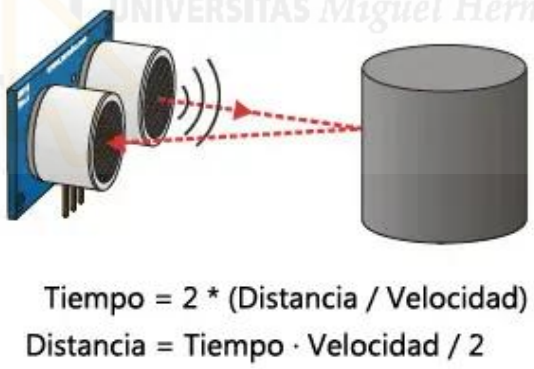

Figura 4: Principio de funcionamiento basado en el tiempo de vuelo (time-of-flight) y cálculo de la distancia.

La utilidad principal de estos sensores en la robótica móvil reside en su capacidad para complementar las limitaciones de los sistemas basados en láser. Su característica más valiosa es la capacidad de detectar materiales transparentes, ya que el sonido se refleja en estas superficies sólidas mientras que la luz las atraviesa. No obstante, su uso está limitado por la baja resolución angular y el fenómeno de la reflexión especular, un problema analizado por Borenstein y Koren [18].

2.2.4 Cámaras

Las cámaras son sensores exteroceptivos pasivos que permiten a los robots percibir el entorno capturando información visual rica sobre la escena, incluyendo la geometría, el color, la textura y la posición espacial de los objetos.

Su principio de funcionamiento se basa en la transducción optoelectrónica, análoga al sistema visual humano. La luz atraviesa un sistema de lentes y se proyecta sobre un sensor de imagen. Este sensor constituye una matriz discreta formada por millones de celdas fotosensibles (fotodiodos). Cuando los fotones inciden sobre estas celdas, se genera una carga eléctrica proporcional a la intensidad lumínica recibida (efecto fotoeléctrico). Posteriormente, un convertidor analógico-digital cuantifica esta carga, transformándola en una matriz numérica de píxeles que conforma la imagen digital procesable.

Su aplicación principal es la visión por computador, una disciplina que permite al robot no solo detectar la presencia de obstáculos, sino extraer información semántica para identificar su naturaleza y navegar de forma autónoma en entornos complejos [19]. Además, las cámaras permiten la implementación de la odometría visual. Esta técnica estima la trayectoria del robot analizando las diferencias geométricas entre fotogramas consecutivos. El algoritmo extrae puntos de alto contraste (*features*) y rastrea su vector de desplazamiento óptico. Mediante geometría proyectiva, el sistema resuelve el problema inverso para deducir la rotación y traslación que ha realizado el robot, tal y como detallan Scaramuzza y Fraundorfer [20].

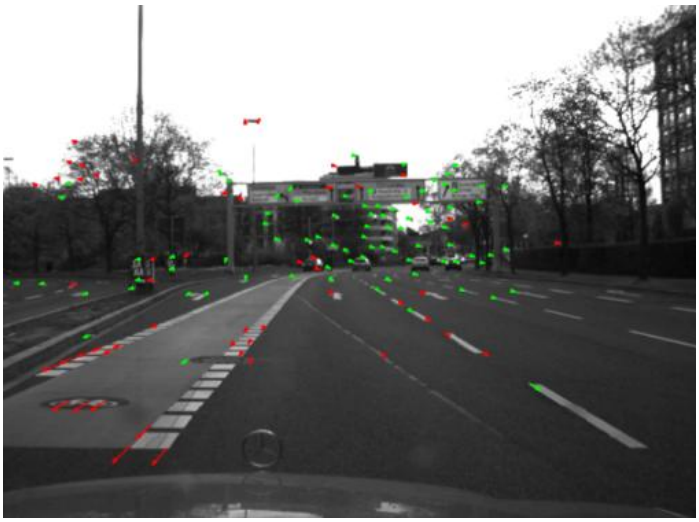

Figura 5: Ejemplo de seguimiento de características (feature tracking) en un entorno de conducción real (Dataset KITTI).

2.2.5 GPS

El Sistema de Posicionamiento Global (GPS) es un sensor exteroceptivo que permite determinar la posición absoluta del robot (latitud, longitud y altitud) en cualquier punto de la superficie terrestre. A diferencia de los sensores locales como el LiDAR o las cámaras, que perciben el entorno inmediato, el GPS contextualiza espacialmente al robot dentro de un marco de referencia global georreferenciado.

Su principio de funcionamiento se basa en la técnica de trilateración. El receptor del robot capta señales de radio emitidas simultáneamente por una constelación de satélites en órbita conocida. Cada señal incorpora una marca temporal precisa de su emisión. Al comparar este tiempo con el de recepción, el sensor calcula el tiempo de vuelo de la señal y, dado que la velocidad de propagación es la velocidad de la luz (c), determina la distancia a cada satélite. Para resolver la posición tridimensional (x, y, z) y corregir el error de sincronización del reloj interno del receptor, es matemáticamente necesario captar la señal de al menos cuatro satélites simultáneamente.

La utilidad del GPS es indispensable para la navegación en exteriores. Mientras que los encoders y la IMU proporcionan una estimación relativa que acumula error (*drift*) con el tiempo, el GPS ofrece una medida absoluta que no se degrada a largo plazo, sirviendo para corregir la posición global del robot. Sin embargo, el GPS estándar tiene un margen de error de varios metros, lo cual resulta excesivo para tareas de robótica móvil que requieren exactitud. Por ello, en aplicaciones de ingeniería se utiliza la técnica RTK (*Real-Time Kinematic*) o GPS Diferencial. Esta variante utiliza una estación base fija de coordenadas conocidas para calcular el error atmosférico y enviarlo al robot, logrando precisiones del orden de centímetros [21].

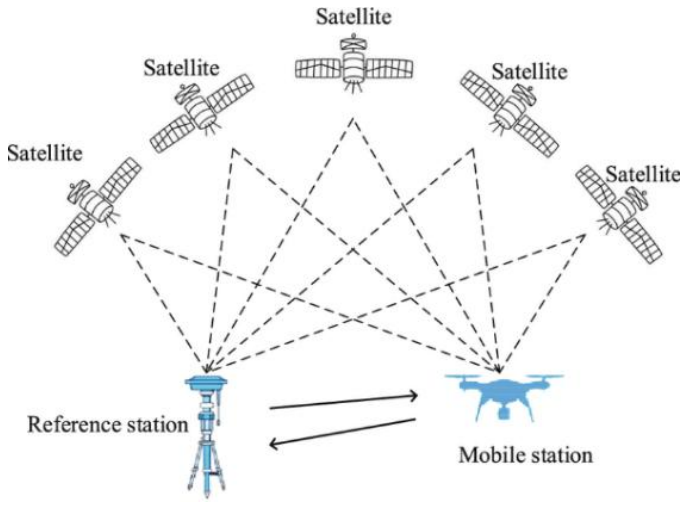

Figura 6: Esquema de funcionamiento de un sistema de posicionamiento RTK.

2.2.6 LiDAR

El *Light Detection and Ranging* (LiDAR) es un sensor activo que se ha consolidado como el estándar industrial para la percepción en robótica móvil y conducción autónoma. A diferencia de los sensores pasivos como las cámaras, que dependen de la iluminación externa, el LiDAR emite su propia energía en forma de pulsos de luz láser, lo que les confiere una robustez excepcional frente a cambios drásticos de iluminación, permitiéndole operar con igual eficacia tanto en plena oscuridad como bajo la luz solar directa.

El principio de funcionamiento fundamental de este dispositivo es la medición por tiempo de vuelo. El sistema emite un breve pulso de luz láser que viaja a través del aire hasta impactar con una superficie del entorno. En ese instante, la luz se refleja de manera difusa y una fracción de ella retorna al receptor del sensor. Un cronómetro de alta precisión mide el intervalo de tiempo Δt transcurrido entre la emisión y la recepción. Dado que la velocidad de la luz c es una constante conocida, la distancia d al objeto se calcula con precisión milimétrica mediante la ecuación:

$$d = \frac{c \cdot \Delta t}{2}$$

Para construir una representación completa del entorno y no solo medir un punto aislado, los sensores LiDAR incorporan un mecanismo de barrido, habitualmente mediante un sistema de espejos rotatorios o prismas que desvían el haz láser en 360 grados [22]. Este escaneo continuo a alta frecuencia genera millones de mediciones por segundo, resultando en una estructura de datos conocida como nube de puntos (*point cloud*). Cada punto de esta nube contiene las coordenadas espaciales exactas (x, y, z) del obstáculo detectado, y frecuentemente un valor de intensidad que indica la reflectividad del material.

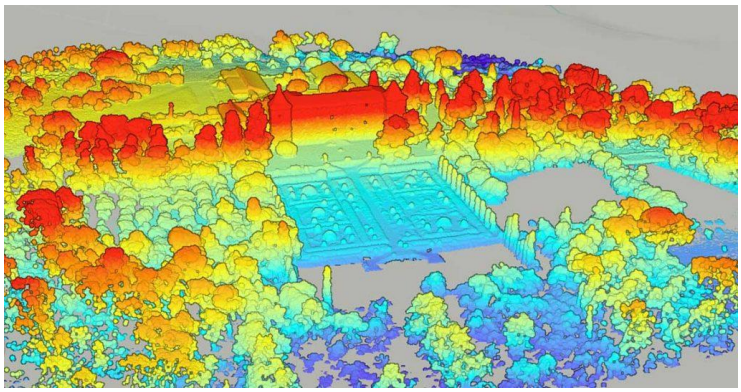

Figura 7: Representación tridimensional del entorno mediante una nube de puntos (point cloud) densa generada por un sensor LiDAR.

La principal utilidad del LiDAR es proporcionar una percepción geométrica precisa y robusta, fundamental para sistemas de navegación autónoma y para algoritmos de SLAM (*Simultaneous Localization and Mapping*) [23]. Gracias a su precisión, los robots pueden construir mapas detallados de entornos desconocidos y localizarse dentro de ellos con mínima incertidumbre, permitiendo la planificación de trayectorias y la evasión de obstáculos en tiempo real.

Asimismo, el LiDAR es idóneo para técnicas de *clustering* y análisis de estructuras espaciales, ya que ofrece una geometría explícita y constante del entorno. A diferencia de las imágenes, cuya apariencia puede variar con la iluminación o los colores, el LiDAR proporciona una representación tridimensional precisa, facilitando la identificación de patrones topológicos y la clasificación de escenas basada en la distribución espacial de los datos.

2.3 Descriptores de nubes de puntos

El análisis del entorno tridimensional (3D) se fundamenta en el uso de nubes de puntos como la representación discreta de una escena. Una nube de puntos se define como un conjunto no ordenado de vértices $P = \{p_i\}_{i=1}^{N}$, donde cada punto p_i se caracteriza primariamente por sus coordenadas espaciales (x, y, z). Frecuentemente, esta información se complementa con atributos como la intensidad o el color. Estos datos se adquieren típicamente a través de tecnologías de escaneo activo, como LiDAR, o pasivas, como los sistemas de profundidad *RGB-D*.

Para abordar tareas complejas de percepción, como la agrupación no supervisada (*clustering*), es esencial transformar los datos de coordenadas primarias en una representación más concisa y robusta: el descriptor de puntos. Un descriptor es un vector de características que codifica de manera única el contexto geométrico y estructural local de un punto dentro de su vecindad.

La calidad del descriptor determina directamente la capacidad de los algoritmos de agrupación para diferenciar estructuras. Un descriptor óptimo debe cumplir con tres propiedades esenciales:

- **Distintivo:** debe maximizar la separación vectorial entre puntos de estructuras diferentes, facilitando la delineación precisa de los grupos y mejorando métricas como el índice de rand ajustado (ARI).
- **Robusto:** debe mantener la invariancia frente al ruido del sensor, cambios de densidad u oclusiones, asegurando un alto *recall* en entornos reales.

• **Semántico:** debe contener información de alto nivel útil para la agrupación lógica, evitando depender únicamente de la geometría local simple.

Para llevar a cabo las tareas de *clustering* en este proyecto, se han evaluado y utilizado dos arquitecturas de redes neuronales 3D: MinkUNeXt y Sonata.

2.3.1 MinkUNeXt

La red MinkUNeXt [24] representa el estado del arte en eficiencia para el procesamiento de nubes de puntos a gran escala. Su funcionamiento se fundamenta en las convoluciones dispersas 3D (*3D sparse convolutions*). A diferencia de las redes tradicionales que procesan todo el volumen espacial desperdiciando recursos en el espacio vacío, MinkUNeXt realiza operaciones de cómputo únicamente en las coordenadas activas (donde existen puntos).

Estructuralmente, utiliza un diseño U-Net totalmente convolucional, modernizado con bloques residuales optimizados (*inverted residual blocks*). Esta combinación le permite extraer características profundas y contextuales de mapas masivos manteniendo un consumo de memoria mínimo y una velocidad de inferencia muy superior a la de las redes densas.

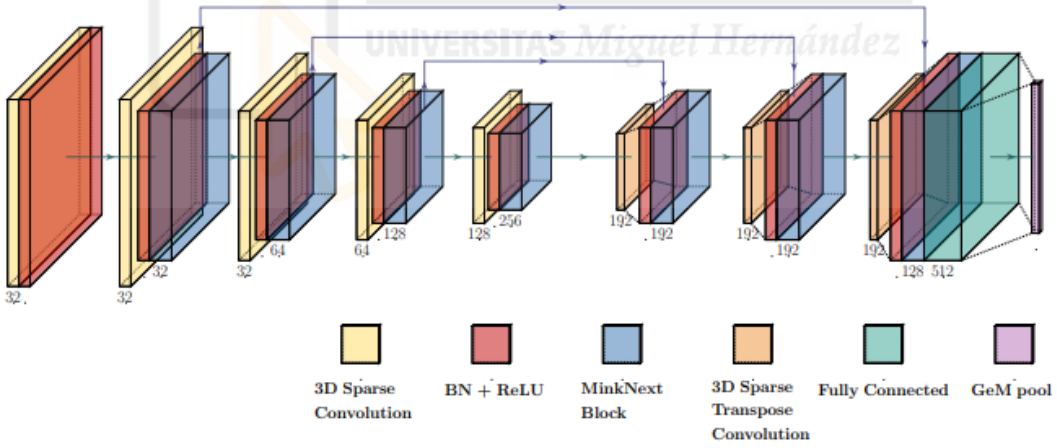

Figura 8: Arquitectura de la red MinkUNeXt.

Como se detalla en la Figura 8, la arquitectura sigue un esquema codificador-decodificador (tipo U-Net).

• **Codificador (izquierda):** reduce progresivamente la dimensionalidad espacial de la nube de puntos mediante convoluciones dispersas (bloques amarillos) y bloques MinkNeXt (bloques azules), aumentando la profundidad de las características (canales) de 32 hasta 512.

• **Decodificador (derecha):** recupera la resolución espacial utilizando convoluciones transpuestas (bloques naranjas).

• **Conexiones de salto (flechas superiores):** concatenan las características de alta resolución del codificador con las del decodificador para preservar la información geométrica detallada que suele perderse en las capas profundas.

Para finalizar el proceso y permitir el reconocimiento del lugar, la red debe convertir toda la información procesada en una firma única. Esto se logra en la última etapa mediante la capa de GeM Pool, representada en color morado. Esta operación agrega matemáticamente todas las características extraídas en un único vector numérico compacto (descriptor global). Este vector es el que finalmente se utiliza para comparar la similitud entre diferentes nubes de puntos.

2.3.2 Sonata

La red Sonata [25] opera bajo un marco de auto-destilación de puntos (*point self-distillation)* mediante aprendizaje auto-supervisado (SSL). Su objetivo principal es superar el denominado atajo geométrico (*geometric shortcut*), un problema común donde las redes aprenden solo características espaciales de bajo nivel, colapsando sus representaciones y perdiendo el significado semántico.

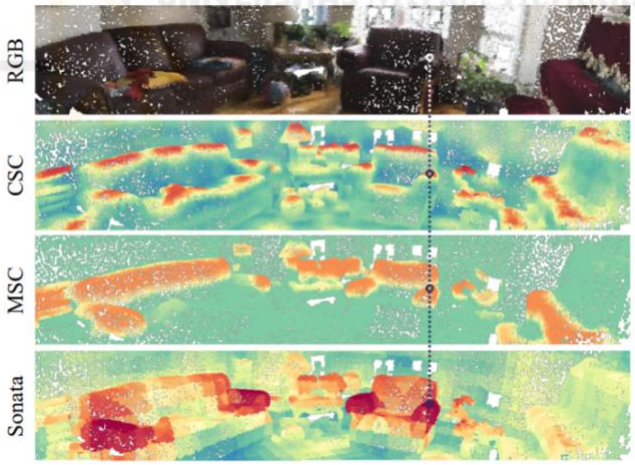

Figura 9: Comparativa visual donde la red Sonata muestra una mayor capacidad para extraer información semántica detallada frente a otros métodos.

Para combatir esto y garantizar descriptores semánticamente ricos, Sonata emplea dos estrategias clave:

• **Perturbación espacial:** aplica una fuerte perturbación (*jitter*) a las coordenadas de los puntos enmascarados, obligando a la red a aprender el contexto global en lugar de memorizar la geometría local trivial.

• **Diseño *decoder-free*:** utiliza una estructura centrada exclusivamente en el codificador (*encoder-only*), evitando la tendencia de las arquitecturas U-Net a reconstruir simplemente la geometría.

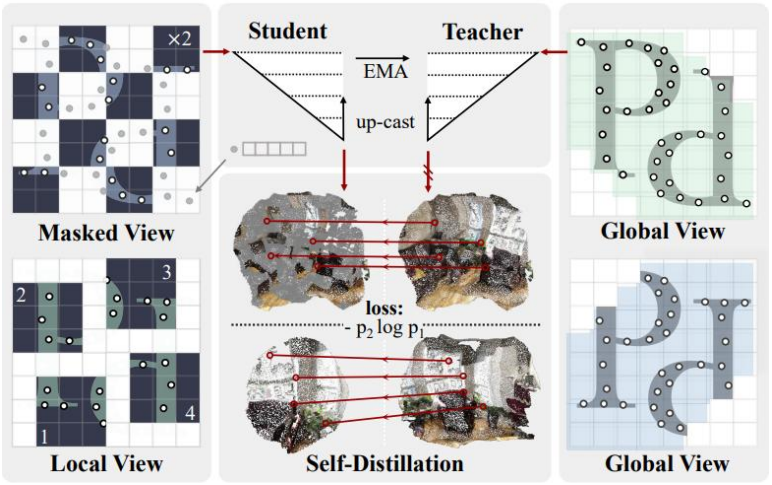

Figura 10: Arquitectura de la red Sonata.

Como se detalla en la Figura 10, el funcionamiento de Sonata se basa en un esquema de auto-destilación con dos redes paralelas que comparten la misma arquitectura de codificador (sin decodificador):

• **Rama *student* (estudiante):** se observa a la izquierda del diagrama. Esta red recibe como entrada una versión incompleta o difícil de la escena, representada en la imagen como *Masked View* (donde partes de la nube de puntos han sido ocultadas) o *Local View*. Su objetivo es generar una representación robusta capaz de inferir la información faltante.

• **Rama *teacher* (maestro):** situada a la derecha, recibe la *Global View* (la escena completa y sin cortes). Esta rama no aprende por descenso de gradiente tradicional, sino que sus pesos se actualizan progresivamente basándose en los del estudiante mediante una media exponencial móvil (EMA), lo que le permite actuar como un objetivo estable y coherente.

El mecanismo central del aprendizaje reside en la comparación ilustrada en el centro de la figura (indicada por las flechas rojas de la función de pérdida). Ambas ramas procesan sus respectivas entradas y generan un vector de características o descriptor. El sistema obliga a que el descriptor generado por el estudiante (a partir de una visión parcial) sea lo más parecido posible al descriptor generado por el maestro (que ve la escena global). De esta forma, la red aprende a generar descriptores globales que capturan la esencia

semántica y estructural del entorno, ignorando oclusiones o ruido, lo cual es fundamental para el reconocimiento de lugares en entornos complejos.

Inicialmente, se planteó el uso generalizado de la red MinkUNeXt para la obtención de descriptores tanto en interiores como en exteriores. Sin embargo, tras la aplicación de los algoritmos de agrupamiento y el análisis de las métricas de evaluación, se determinó que el desempeño de esta red en entornos interiores no alcanzaba los niveles de precisión requeridos.

Ante esta limitación, se procedió a evaluar los descriptores generados por la red Sonata. Los resultados experimentales mostraron una mejora sustancial en la clasificación, motivo por el cual se estableció finalmente el uso de Sonata para el *clustering* de interiores, manteniendo MinkUNeXt para los exteriores debido a su eficiencia.

2.4 Técnicas de aprendizaje automático

2.4.1 Definición de aprendizaje automático

El aprendizaje automático (*machine learning*) constituye una rama de la inteligencia artificial cuya finalidad es lograr que las máquinas adquieran la capacidad de aprender de su propia experiencia sin necesidad de intervención humana [26]. Mediante algoritmos específicos, los ordenadores pueden determinar cómo llevar a cabo determinadas tareas sin haber sido programados explícitamente para ello. Estos algoritmos utilizan los datos que se les proporciona para identificar patrones que generen nuevo conocimiento y, así, contribuir a la toma de mejores decisiones. En consecuencia, la finalidad principal del *machine learning* es la predicción de resultados de interés basándose en los datos de entrada.

Los algoritmos de aprendizaje automático perfeccionan sus métodos a medida que se entrenan con nuevos datos, de modo que su rendimiento depende en gran medida de la calidad y cantidad de la información disponible. Un conjunto de datos amplio y representativo favorece un aprendizaje más sólido y, en consecuencia, resultados más precisos.

2.4.2 Aprendizaje supervisado

El aprendizaje supervisado es aquel en el que el modelo se entrena a partir de un conjunto de datos previamente etiquetados, es decir, datos cuyo valor objetivo ya es conocido. Esto permite que el modelo identifique y generalice los patrones presentes en los datos, de forma que pueda realizar predicciones fundamentadas cuando se le

proporcione nueva información.

Este tipo de aprendizaje es el más utilizado para la solución de problemas de clasificación y regresión. Sus principales aplicaciones se centran en la automatización de decisiones y diagnósticos, siendo utilizado en la detección de fraudes [27], los sistemas de reconocimiento de voz [28] y procesamiento de lenguaje natural [29].

2.4.3 Aprendizaje no supervisado

El aprendizaje no supervisado es aquel en el que los datos no presentan etiquetas asociadas, de modo que el modelo debe identificar patrones subyacentes para generar sus propias categorías. Su objetivo principal es establecer relaciones y formar agrupaciones entre los datos que muestran similitudes entre sí.

Este tipo de aprendizaje es fundamental para la exploración de datos y la ingeniería de características. Sus aplicaciones más importantes incluyen el *clustering* (agrupamiento), una técnica ampliamente utilizada para la segmentación de clientes y análisis de mercados [30].

Por otro lado, destaca la detección de anomalías (*anomaly detection*), cuya capacidad para identificar comportamientos inusuales es crítica en la detección de intrusiones y ciberataques en redes [31], así como en la monitorización de maquinaria industrial para el mantenimiento predictivo [32].

2.4.4 Aprendizaje por refuerzo

El aprendizaje por refuerzo es aquel en el que un agente desarrolla la capacidad de tomar decisiones óptimas mediante la interacción continua con un entorno. Su comportamiento se orienta a través de un sistema de recompensas y penalizaciones que evalúa las acciones ejecutadas. De este modo, el agente aprende mediante un proceso de ensayo y error, experimentando acciones y ajustando su estrategia para maximizar las recompensas a largo plazo.

Este tipo de aprendizaje es el motor de la Inteligencia Artificial en entornos dinámicos y sus principales aplicaciones son la robótica inteligente y el control de movimiento [33], el desarrollo de sistemas de conducción para vehículos autónomos [34] y la optimización de estrategias en juegos complejos [35].

2.5 Técnicas de *clustering*

2.5.1 Definición de *clustering*

El *clustering* es una técnica de aprendizaje no supervisado cuyo objetivo principal consiste en agrupar un conjunto de datos en distintas categorías o clústeres basándose en la similitud inherente entre las observaciones. A diferencia del aprendizaje supervisado, en el que el modelo se entrena a partir de datos etiquetados, los algoritmos de *clustering* operan sobre datos sin etiquetar y buscan descubrir estructuras subyacentes, patrones compartidos o regiones de mayor densidad dentro de la distribución de los datos [36].

2.5.2 *K-means clustering*

El algoritmo *K-means* es una de las técnicas de agrupamiento más utilizadas y reconocidas en el ámbito del aprendizaje automático debido a su simplicidad y eficiencia computacional. Se clasifica como un método de partición basado en centroides.

El funcionamiento de este método parte de la necesidad de establecer previamente un número de grupos (*k*). Este valor debe especificarse antes de iniciar el algoritmo, aunque no necesariamente coincida con el número óptimo de clústeres, algo que posteriormente puede evaluarse mediante técnicas de validación. A partir de esta elección inicial, el algoritmo aplica un proceso iterativo orientado a minimizar la varianza *intra-cluster*. Su funcionamiento puede describirse mediante las siguientes etapas:

1. Definición de parámetros: se define el número de grupos o clústeres objetivo, denotado como *k*.

2. Inicialización: se generan aleatoriamente k centroides iniciales en el espacio de datos. Estos puntos actuarán como los prototipos representativos de cada grupo en la primera iteración.

3. Asignación: se calcula la distancia entre cada observación y los *k* centroides. Cada dato es asignado al clúster cuyo centroide es el más próximo.

4. Actualización: una vez que todos los datos han sido asignados, se recalcula la ubicación de los centroides. El nuevo centroide de cada grupo se obtiene calculando la media aritmética de todas las observaciones asignadas a dicho grupo.

5. Iteración y Convergencia: se repiten los pasos 3 y 4 de forma cíclica. El proceso termina cuando se cumple el criterio de convergencia, es decir, cuando los centroides no varían su posición significativamente entre iteraciones o las asignaciones de las observaciones ya no cambian.

No todos los conjuntos de datos son adecuados para este algoritmo. *K-means* es altamente recomendable y ofrece sus mejores resultados bajo las siguientes condiciones

- **Datos numéricos y continuos:** al basarse en cálculos de medias y distancias, funciona idealmente en espacios vectoriales numéricos.
- **Geometría convexa (esférica):** el algoritmo asume que los clústeres tienen formas globulares o esféricas. Tiene dificultades para identificar grupos con formas geométricas complejas, alargadas o entrelazadas (como medias lunas o anillos).
- **Densidad y tamaño uniforme:** funciona mejor cuando los distintos grupos tienen densidades similares y tamaños comparables.
- **Bajo nivel de ruido:** es un algoritmo sensible a los valores atípicos (*outliers*), ya que un solo dato muy alejado puede desplazar significativamente el centroide y distorsionar el agrupamiento.

Debido a su versatilidad, *k-means* se aplica en una gran variedad de campos, siendo especialmente relevante en ingeniería para:

- **Compresión de datos (cuantificación):** es la aplicación original del algoritmo. Permite reducir el tamaño de señales o imágenes sustituyendo miles de valores originales por un número pequeño de prototipos (centroides) representativos, logrando una transmisión mucho más eficiente con una pérdida de información controlada [37].
- **Detección de anomalías:** sirve para identificar fallos o intrusiones en sistemas. El algoritmo modela el comportamiento normal agrupando los datos históricos seguros; posteriormente, cualquier nuevo dato que aparezca a una distancia excesiva de estos centroides se marca automáticamente como una posible avería o ataque [38].
- **Segmentación de imágenes:** se utiliza masivamente en visión artificial para simplificar imágenes complejas. El algoritmo agrupa los píxeles con colores o intensidades similares en regiones homogéneas, facilitando el análisis posterior de la escena y la detección de objetos [39].

2.5.3 HDBSCAN *clustering*

El algoritmo HDBSCAN (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*) es una técnica de agrupamiento avanzada que surge como una extensión del algoritmo DBSCAN. Se clasifica como un método híbrido que combina el *clustering* basado en densidad con el *clustering* jerárquico.

A diferencia de los métodos de partición como *k-means*, HDBSCAN no requiere predefinir el número de clústeres (*k*). Su filosofía se basa en que los clústeres son regiones de alta densidad separadas por zonas de ruido. El único parámetro crítico que debe especificarse es el tamaño mínimo del clúster (*min_cluster_size*), lo que permite al algoritmo descubrir la estructura de los datos por sí mismo. En lugar de minimizar la varianza, el algoritmo aplica un proceso orientado a maximizar la estabilidad de los grupos.

Su funcionamiento puede describirse mediante las siguientes etapas:

1. Definición de parámetros: se define el parámetro *min_cluster_size*, que indica el número mínimo de puntos necesarios para considerar que un grupo es un clúster válido.

2. Transformación del espacio: el algoritmo calcula una métrica llamada distancia de alcanzabilidad mutua. Esto aleja matemáticamente los puntos que están en zonas de baja densidad, haciendo que el ruido sea más fácil de separar de los grupos reales.

3. Construcción de la jerarquía: se construye un árbol de expansión mínima que conecta todos los datos. A partir de él, se genera un dendrograma (árbol jerárquico) que representa todas las posibles agrupaciones conectadas a diferentes niveles de densidad.

4. Condensación del árbol: dado que la jerarquía completa es demasiado compleja, el algoritmo la simplifica. Las ramas del árbol que tienen menos puntos que el *min_cluster_size* se consideran irrelevantes y se descartan, dejando solo la estructura principal.

5. Extracción por estabilidad: en vez de cortar el árbol a una altura fija, el algoritmo selecciona los clústeres basándose en su estabilidad. Se eligen aquellos grupos que persisten (sobreviven) durante un mayor rango de densidades. Los puntos que no entran en estos grupos se etiquetan como ruido (-1).

No todos los conjuntos de datos requieren esta complejidad, pero HDBSCAN es altamente recomendable y ofrece sus mejores resultados bajo las siguientes condiciones:

• **Geometría arbitraria (no convexa):** a diferencia de *k-means,* no asume formas esféricas. Es ideal para identificar grupos con formas topológicas complejas, alargadas, anillos o espirales, ya que sigue la conectividad de los puntos en lugar de la distancia a un centro.

• **Densidad variable:** funciona excelentemente en conjuntos de datos donde existen clústeres muy densos junto a otros mucho más dispersos. Esta es la principal ventaja frente al DBSCAN clásico, que suele fallar al intentar aplicar un único umbral de densidad global a todo el conjunto de datos.

• **Presencia de ruido y *outliers*:** es un algoritmo diseñado específicamente para gestionar datos sucios. Clasifica explícitamente los valores atípicos como ruido (-1), evitando que estos puntos distorsionen la posición o forma de los clústeres principales.

• **Desconocimiento del número de grupos:** es ideal para la exploración de datos *a priori*, ya que el algoritmo infiere automáticamente la cantidad de clústeres (*k*) basándose en la estabilidad de la estructura jerárquica, eliminando la necesidad de estimar este parámetro manualmente.

Debido a su capacidad para filtrar ruido y adaptarse a la forma de los datos, HDBSCAN se aplica en campos avanzados de ingeniería, siendo especialmente relevante para:

• **Detección de anomalías (ciberseguridad/IoT):** al aislar el ruido de forma nativa, se utiliza para identificar fallos críticos o intrusiones. Permite separar el tráfico de red normal de ataques anómalos sin necesidad de etiquetas previas, basándose puramente en la desviación de densidad [40].

• **Datos geoespaciales y trayectorias:** es el estándar para el agrupamiento de ubicaciones GPS. Los métodos basados en densidad han probado ser superiores para detectar patrones de movimiento en ciudades (como rutas de reparto o flujos turísticos), ya que respetan la topografía urbana y no imponen formas circulares [41].

• **Bioinformática (genética):** Se aplica en la clasificación de secuencias de ADN y expresión génica. Dado que los datos biológicos suelen tener mucho ruido de fondo y estructuras jerárquicas naturales, este enfoque permite aislar familias de genes funcionales con mayor precisión que los métodos de partición rígida [42].

2.5.4 *Mean shift clustering*

El algoritmo *mean shift* es una técnica de agrupamiento basada en la densidad que opera bajo un principio muy intuitivo: buscar las zonas de mayor concentración de datos. A diferencia de métodos rígidos como *k-means*, este algoritmo no asume formas geométricas predefinidas ni intenta cortar el espacio. En su lugar, trata los datos como si fueran un terreno topográfico, escalando iterativamente hacia los picos o cimas donde se acumulan más puntos para definir allí los centros de los grupos. El funcionamiento de este método no requiere predefinir el número de clústeres (*k*).

Su filosofía se basa en localizar los picos o modas (zonas de máxima densidad) en la distribución de los datos. El único parámetro crítico que debe especificarse es el ancho de banda (*bandwidth*), que determina el radio del área de búsqueda. El algoritmo funciona desplazando ventanas deslizantes hacia las zonas de mayor densidad hasta que convergen en un centro estable.

Su funcionamiento puede describirse mediante las siguientes etapas:

1. Definición de parámetros: se define el parámetro de ancho de banda (*bandwidth*), que actúa como una ventana de observación. Este valor es crucial: un ancho pequeño genera muchos grupos, mientras que uno grande puede agrupar todo en un solo clúster.

2. Inicialización: se distribuyen ventanas (generalmente circulares o hiperesféricas) centradas en cada punto de datos del espacio. Cada punto es, inicialmente, un candidato a centroide.

3. Cálculo de la media: dentro de cada ventana, el algoritmo calcula la media matemática de todos los puntos que caen dentro de su radio. Si se utiliza una función *kernel*, se da más peso a los puntos cercanos al centro de la ventana.

4. Desplazamiento (*shift*): el centro de la ventana se desplaza desde su posición actual hacia la media calculada en el paso anterior. Esto hace que la ventana suba gradualmente hacia la zona donde hay más puntos (mayor densidad).

5. Iteración y convergencia: se repiten los pasos 3 y 4 de forma cíclica. El proceso termina cuando las ventanas dejan de moverse (se estancan en un pico de densidad). Las ventanas que convergen en el mismo pico se fusionan, y todos los puntos recorridos por ellas se asignan a ese clúster.

No todos los conjuntos de datos son adecuados para este algoritmo debido a que requiere muchos cálculos. Sin embargo, *mean shift* es la mejor opción bajo las siguientes condiciones:

• **No es necesario definir el número de grupos:** a diferencia de otros métodos, no hace falta adivinar ni estimar previamente el número de clústeres. El algoritmo analiza la estructura de los datos y encuentra automáticamente la cantidad de grupos necesarios.

• **Formas complejas e irregulares:** el algoritmo no se limita a encontrar grupos redondos o esféricos (como *k-means*). Es capaz de identificar y adaptarse a grupos con formas extrañas o alargadas, siempre que los datos estén concentrados.

• **Prioridad de la precisión sobre la velocidad:** debido a que su funcionamiento es

lento en bases de datos gigantes (alta complejidad computacional), se recomienda su uso en conjuntos de datos de tamaño moderado donde lo más importante es obtener un agrupamiento de alta calidad y precisión.

• **Limpieza de ruido:** es excelente para suavizar datos, ya que ignora los pequeños puntos aislados (ruido) y se centra únicamente en las zonas donde la información es densa y fiable.

Debido a que este método funciona muy bien analizando densidades visuales, se aplica casi exclusivamente en visión artificial para:

• **Segmentación de imágenes:** se utiliza para simplificar imágenes digitales, agrupando los píxeles que tienen colores parecidos para separar claramente los objetos del fondo y facilitar su análisis posterior [43].

• **Seguimiento de objetos en video (*tracking*):** permite seguir el movimiento de un objetivo (como una persona o un coche) a través de una secuencia de video, calculando en tiempo real hacia dónde se desplaza su centro de color aunque cambie de forma o tamaño [44].

• **Filtrado y suavizado:** se aplica para mejorar la calidad de las imágenes, eliminando la textura innecesaria o el grano (ruido visual) pero manteniendo nítidos los bordes y contornos de los objetos [45].

2.5.5 *Hierarchical clustering*

El algoritmo *hierarchical clustering* (agrupamiento jerárquico) es una técnica de análisis de datos que, a diferencia de *k-means* o *mean shift*, no busca simplemente dividir los datos en grupos, sino que intenta construir una jerarquía de clústeres. Se suele representar mediante una estructura de árbol llamada dendrograma. Existen dos enfoques: el divisivo (de arriba abajo) y el aglomerativo (de abajo arriba). Este último es el más común y opera bajo un principio sencillo: cada dato empieza siendo su propio grupo y se van fusionando por parejas hasta formar uno solo.

A diferencia de otros algoritmos, el funcionamiento de este método no requiere predefinir el número de clústeres antes de empezar. Su filosofía se basa en calcular todas las posibles agrupaciones. El algoritmo construye un árbol completo que va desde los datos individuales hasta un único grupo global; posteriormente, el usuario selecciona el número de clústeres óptimo cortando dicho árbol a la altura deseada.

Su funcionamiento aglomerativo puede describirse mediante las siguientes etapas:

1. **Inicialización:** se considera que cada punto de datos individual es un clúster

independiente. Si el conjunto tiene N datos, comenzamos con N grupos de tamaño uno.

2. Cálculo de similitud: se calcula la distancia entre todos los pares de clústeres activos y se almacenan en una matriz de distancias.

3. Fusión: el algoritmo identifica los dos clústeres más cercanos (similares) según el criterio de enlace (*linkage*) seleccionado y los une en un nuevo grupo único.

4. Actualización: se recalcula la distancia entre este nuevo grupo recién formado y todos los demás grupos restantes en el espacio.

5. Iteración y definición de la partición: se repiten los pasos 3 y 4 de forma cíclica. El proceso matemático termina cuando todos los datos han sido fusionados en un único gran clúster raíz. La configuración final de los grupos no es automática, sino que se obtiene estableciendo un umbral de corte en el dendrograma; la altura de este corte determina el número de clústeres resultantes en función de la similitud deseada.

No todos los conjuntos de datos son adecuados para este algoritmo debido a su alto coste de procesamiento. Sin embargo, el *hierarchical clustering* es la herramienta de referencia bajo las siguientes condiciones:

• **Visualización de la estructura:** es la mejor opción cuando se necesita entender las relaciones entre los datos. El dendrograma permite ver gráficamente la jerarquía completa y cómo se organizan los subgrupos, algo que los métodos de partición plana no ofrecen.

• **Flexibilidad con el número de grupos:** al no requerir definir la *k* inicialmente, facilita la exploración de datos desconocidos. Permite determinar la cantidad óptima de clústeres *a posteriori*, observando las distancias de unión en el árbol generado.

• **Conjuntos de datos moderados:** debido a su alto coste de cálculo, no es viable para *big data*. Se recomienda su uso en *datasets* de tamaño medio donde se prioriza la calidad y precisión del análisis sobre la velocidad de ejecución.

• **Estructuras anidadas:** a diferencia de *k-means*, este método detecta eficazmente grupos que contienen otros subgrupos en su interior (relaciones de inclusión), respetando la jerarquía natural de la información.

Debido a su capacidad para estructurar datos complejos y revelar relaciones anidadas, el agrupamiento jerárquico se aplica extensamente en ingeniería para tareas que requieren una visión taxonómica de la información. Algunas de sus aplicaciones más relevantes son:

• **Ciberseguridad (linaje de *malware*):** se utiliza para rastrear la evolución de virus y *malware*. Dado que el código malicioso suele crearse modificando versiones anteriores, este algoritmo es capaz de reconstruir el árbol genealógico (filogenia) de los ataques, permitiendo a los antivirus bloquear familias enteras de amenazas basándose en su ancestro común [46].

• **Sistemas de energía (*smart grids*):** es fundamental para el análisis de perfiles de carga eléctrica. Agrupa los datos de los contadores inteligentes construyendo una jerarquía de consumo: desde el comportamiento de un solo usuario, pasando por edificios, hasta barrios enteros. Esto permite a las eléctricas predecir picos de demanda con una granularidad que otros métodos no permiten [47].

• **Reestructuración de software:** se emplea para organizar sistemas de código legado (*legacy code*). El algoritmo agrupa automáticamente las clases y archivos que interactúan entre sí para reconstruir la arquitectura modular del software, ayudando a los ingenieros a entender programas complejos sin documentación previa [48].

2.5.6 *Spectral clustering*

El algoritmo *spectral clustering* (agrupamiento espectral) es una técnica avanzada que se aleja de los métodos tradicionales basados en distancias para adoptar un enfoque basado en la teoría de grafos. En lugar de asumir que los clústeres son formas compactas o esféricas, este método interpreta los datos como nodos de un grafo conectados por aristas, cuya fuerza depende de la similitud entre puntos. Su objetivo es realizar cortes en dicho grafo para separar los subgrupos que están fuertemente conectados entre sí y débilmente conectados con el resto.

A diferencia de los métodos geométricos, su funcionamiento se basa en el álgebra lineal, específicamente en el análisis del espectro (valores propios o *eigenvalues*) de la matriz de similitud de los datos. Esto le permite transformar el espacio original en un subespacio de menor dimensión donde los clústeres, que originalmente podrían tener formas complejas o entrelazadas, se vuelven linealmente separables y fáciles de agrupar.

Su funcionamiento puede describirse mediante las siguientes etapas:

1. Construcción del grafo de similitud: se genera una matriz de adyacencia o afinidad que representa las conexiones entre todos los puntos de datos. Generalmente se utiliza una función *kernel* (como la gaussiana o de vecinos más cercanos) para asignar un peso alto a los puntos cercanos y bajo a los lejanos.

2. Cálculo de la matriz Laplaciana: a partir de la matriz de similitud, se calcula la

matriz Laplaciana. Esta matriz es una representación matemática fundamental que condensa la topología y la estructura de conectividad del grafo.

3. Descomposición espectral (*eigenvalues*): se calculan los primeros *k* vectores propios (*eigenvectors*) de la matriz Laplaciana. Estos vectores contienen la información estructural más importante de los datos y se utilizan para proyectar los puntos originales a un nuevo espacio dimensional reducido.

4. Agrupamiento final: en este nuevo espacio transformado, los clústeres complejos se han convertido en grupos compactos y separados. Finalmente, se aplica un algoritmo estándar (como *k-means*) sobre estos vectores propios para obtener la partición definitiva.

Debido a su alto coste computacional, este algoritmo es la mejor opción bajo las siguientes condiciones:

• **Geometrías no convexas:** es muy superior a *k-means* cuando los clústeres tienen formas complejas, como anillos concéntricos, espirales o formas entrelazadas. Al basarse en la conectividad y no en la distancia al centro, detecta la continuidad de la forma sin problemas.

• **Datos basados en grafos:** es la opción natural cuando los datos no son coordenadas en un espacio, sino relaciones (redes sociales, enlaces de internet, interacciones biológicas), ya que trabaja nativamente con matrices de adyacencia.

• **Necesidad de reducción dimensional:** el algoritmo realiza implícitamente una reducción de dimensionalidad antes de agrupar. Esto lo hace robusto en situaciones donde el espacio original tiene muchas dimensiones ruidosas que dificultarían la convergencia de otros métodos.

Debido a su capacidad para particionar grafos complejos basándose en la conectividad global, este algoritmo tiene aplicaciones validadas en campos avanzados de ingeniería como:

• **Segmentación de imágenes (cortes normalizados):** es el estándar clásico en visión artificial para separar objetos del fondo. Shi y Malik introdujeron el método de *Normalized Cuts* [49], demostrando que el agrupamiento espectral puede particionar una imagen basándose en la textura y el brillo, respetando los bordes de los objetos mejor que los métodos locales.

• **Detección de comunidades en redes:** se utiliza para analizar redes sociales o de telecomunicaciones. Permite identificar cliques o comunidades de usuarios que interactúan frecuentemente entre sí, aislando subgrupos funcionales dentro de una

red masiva de conexiones [50].

• **Separación de fuentes de audio (*speaker diarization*):** en procesamiento de voz, se aplica para distinguir entre diferentes hablantes en una grabación. El algoritmo agrupa los segmentos de audio basándose en la similitud espectral de la voz, permitiendo etiquetar "quién habló cuándo" sin entrenamiento previo supervisado [51].

2.6 Métricas de evaluación

En este apartado se describen los indicadores cuantitativos seleccionados para validar los resultados del proyecto. Las métricas se presentan divididas en dos categorías: métricas externas y métricas internas, las cuales se detallan en los siguientes subapartados.

2.6.1 Métricas externas

Las métricas externas son aquellas que nos permiten evaluar la calidad del agrupamiento (*clustering*) cuando disponemos de etiquetas o una referencia conocida (*ground truth*). Lo que buscan estas métricas es medir el grado de concordancia entre los grupos generados por el algoritmo y la clasificación real de los datos, determinando si el algoritmo ha sido capaz de identificar correctamente las clases originales

En el presente trabajo, las métricas externas consideradas son el *adjusted rand index* (ARI), el *recall* y la *intersection over union* (IoU).

• ***Adjusted Rand Index* (ARI):** es una métrica utilizada para medir la similitud entre dos particiones de un conjunto de datos: la asignación de clústeres realizada por el algoritmo y la clasificación verdadera (*ground truth*). A diferencia del índice de rand simple, el ARI introduce una corrección probabilística para tener en cuenta la coincidencia que se produciría por puro azar.

Su función es crucial para garantizar que el modelo no solo agrupa correctamente una gran cantidad de puntos, sino que realmente ha descubierto la estructura de clases subyacente. La fórmula se basa en el recuento de pares de puntos que han sido clasificados de forma idéntica o diferente en ambas particiones, para luego ajustarse:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}$$

Donde:

***RI*:** la proporción de pares de elementos que han sido agrupados de la misma manera en ambas particiones (la coincidencia observada).

***E [RI]*:** el valor esperado del índice de Rand bajo una hipótesis de aleatoriedad.

El ARI resultante se encuentra en un rango de -1 a 1. Un valor de 1 indica una concordancia perfecta entre las particiones. Un valor cercano a 0 sugiere que la similitud es comparable a un resultado aleatorio, y valores negativos indican un rendimiento peor que el azar. El ARI es la métrica de elección en estudios de bioinformática [52] y teledetección [53].

• ***Recall*:** también conocido como sensibilidad, es una métrica de evaluación externa fundamental que cuantifica la capacidad de recuperación y la completitud del modelo. Mide la proporción de casos positivos reales que el algoritmo ha logrado identificar, respecto al total de casos positivos existentes.

Su importancia reside en su enfoque directo en la minimización de los falsos negativos (FN), que son los casos críticos que el modelo omite. Por ello, el *recall* es una métrica esencial en la evaluación de escenarios de alto riesgo, como el diagnóstico médico o la detección de anomalías, donde el coste de pasar por alto un caso positivo es superior al coste de una falsa alarma.

Se calcula mediante el cociente entre los verdaderos positivos (TP) y la suma de todos los casos que deberían haber sido detectados:

$$Recall = \frac{TP}{TP + FN}$$

Donde:

TP (true positive): el número de casos positivos que el algoritmo clasificó correctamente.

FN (false negative): el número de casos positivos que fueron omitidos o pasados por alto por el algoritmo.

El *recall* varía entre 0 y 1. Un valor cercano a 1 indica que el modelo es muy bueno para evitar omitir información relevante. En la literatura académica, el *recall* es prioritario en áreas críticas como la clasificación de imágenes histopatológicas en biomedicina [54] y los sistemas de detección de intrusos (IDS) en ciberseguridad [55].

• ***Intersection over Union* (IoU**): también conocido como índice de Jaccard**,** es una métrica fundamental en tareas de segmentación espacial y detección de objetos. Su

objetivo es cuantificar la precisión del solapamiento entre la región predicha por el modelo y la región verdadera (*ground truth*). A diferencia de otras métricas de conteo, el IoU penaliza severamente los desajustes en los bordes, midiendo la fidelidad geométrica del contorno detectado.

Esta métrica se calcula como la razón entre el área de la intersección y el área de la unión de las regiones predichas y reales. En términos de los componentes de la matriz de confusión, se expresa de la siguiente manera:

$$IoU = \frac{TP}{TP + FP + FN}$$

Donde:

TP (verdaderos positivos): el área de superposición correcta (intersección).

FP (falsos positivos): el área predicha por el modelo que queda fuera del objeto real.

FN (falsos negativos): el área del objeto real que el modelo no logró cubrir.

El valor resultante oscila entre 0 y 1. Un IoU de 1 representa una superposición perfecta, indicando que el contorno predicho es idéntico al contorno real. Esta métrica es crítica en aplicaciones de alta precisión, como la detección de objetos 3D en vehículos autónomos [56] y la segmentación de imágenes médicas [57].

2.6.2 Métricas internas

Las métricas internas permiten evaluar la calidad del agrupamiento (*clustering*) basándose únicamente en la información intrínseca de los datos, operando en escenarios donde no existen etiquetas predefinidas (*ground truth*). El objetivo fundamental de estas métricas es garantizar la formación de grupos bien definidos, buscando simultáneamente maximizar la compacidad (minimizar la dispersión *intra-cluster*) y maximizar la separación (aumentar la distancia *inter-cluster*) [58].

En el presente trabajo, inicialmente se planteó el uso de las siguientes métricas el método del codo, el índice de silueta y el índice de Davies-Bouldin como técnicas principales para la evaluación estructural del agrupamiento.

- **Método del codo:** es una técnica ampliamente empleada para determinar el número óptimo de clústeres (*k*) en algoritmos de partición como *k-means*. La premisa fundamental consiste en identificar el punto de equilibrio donde aumentar el número de grupos deja de aportar una reducción significativa en la varianza *intra-cluster*.

  Para cuantificar esta varianza se utiliza la suma de los errores al cuadrado (SSE),

también conocida como inercia, que mide la dispersión de los puntos respecto al centroide de su grupo asignado. Esta medida se expresa matemáticamente de la siguiente manera:

$$SSE = \sum_{i=1}^{k} \sum_{x \epsilon Ci} ||x - \mu i||^2$$

Donde:

- ***k:*** número total de clústeres.
- **C_i:** el conjunto de puntos que forman el clúster *i*.
- ***x:*** cada punto individual que pertenece al clúster *i*.
- ***μ_i:*** el centroide del clúster *i*.

En la práctica, el procedimiento requiere ejecutar el algoritmo iterativamente para un rango de valores de *k* y registrar el SSE resultante en cada caso. Al representar gráficamente estos valores (con *k* en el eje horizontal y SSE en el vertical), el objetivo es identificar el punto de inflexión donde la pendiente de la curva se suaviza drásticamente. Este punto, que visualmente se asemeja a un codo, señala el número óptimo de clústeres, indicando que añadir más divisiones fragmentaría los datos sin mejorar sustancialmente la estructura global [59].

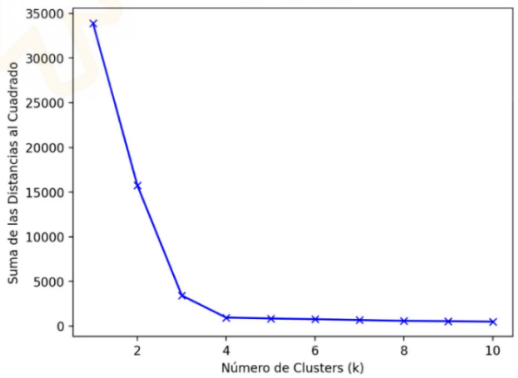

Figura 11: Representación gráfica del método del codo para determinar el número de clústeres óptimo.

• **Índice de silueta:** permite evaluar la calidad de los clústeres generados mediante algoritmos basados en la distancia euclídea, como es el caso de *k-means*. Esta medida cuantifica la relación existente entre la separación de los diferentes clústeres y la similitud entre los puntos de un mismo clúster.

Para el cálculo de este coeficiente se define el valor s_i para cada punto de la siguiente manera:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Donde:

s_i: coeficiente de silueta para el punto *i.*

a_i: distancia promedio entre el punto *i* y todos los demás puntos dentro del mismo clúster (cohesión *intra-cluster*).

b_i: distancia promedio entre el punto *i* y todos los puntos del clúster más cercano (separación *inter-cluster*).

El coeficiente resultante oscila en un rango de -1 a 1. Un valor cercano a 1 indica la mejor separación posible, situación que ocurre cuando la distancia interna a_i es mucho menor que la externa b_i. Por el contrario, un valor cercano a -1 denota una mala asignación, ocurriendo cuando la distancia externa es menor que la interna. Si se obtiene un valor cercano a 0, indica que el punto se encuentra en el límite de decisión entre dos grupos (solapamiento).

Para estimar la cantidad óptima de clústeres, se utiliza el promedio de la silueta (*average silhouette*) calculado sobre todos los puntos del conjunto de datos. El procedimiento consiste en iterar sobre un rango de valores de *k* y seleccionar aquel que maximice esta métrica, lo que garantiza grupos con alta cohesión y buena separación [60].

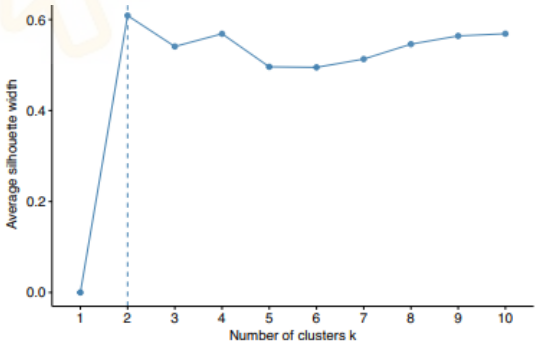

Figura 12: Gráfico para obtener el número óptimo de clústeres mediante el índice de silueta.

• **Índice de Davies-Bouldin:** es una métrica empleada para evaluar la calidad de los clústeres producidos por un algoritmo de *clustering*. La premisa fundamental de este índice consiste en validar que una agrupación de calidad debe generar clústeres que sean separados y compactos simultáneamente. Para ello, el método se basa en relacionar la dispersión dentro de los clústeres (*intra-cluster*) con la separación entre ellos (*inter-cluster*).

El índice se construye calculando el cociente entre estas dos medidas para identificar el peor escenario de similitud de cada grupo. Matemáticamente, se define de la siguiente manera:

$$DBI = \frac{1}{k}\sum_{i=1}^{k} \max_{j \neq i}\left(\frac{si + sj}{dij}\right)$$

Donde:

k: número total de clústeres.

s_i: dispersión *intra-cluster* del grupo i (mide la cercanía de los puntos dentro del grupo *i*).

s_j: dispersión *intra-cluster* del grupo j (mide la cercanía de los puntos dentro del grupo *j*, siendo $j \neq i$).

d_{ij}: separación *inter-cluster* (mide la distancia entre los centroides de los grupos *i* y *j*).

La interpretación de este índice se basa en la minimización. Una dispersión *intra-cluster* baja (puntos muy cercanos entre sí) y una dispersión *inter-cluster* alta (grupos muy alejados) reducen el numerador y aumentan el denominador, dando como resultado un valor bajo. Por lo tanto, cuando los clústeres están bien definidos, el valor del índice disminuye. Para determinar el número óptimo de clústeres, se debe ejecutar el algoritmo para diferentes valores de *k* y seleccionar aquel que ofrezca el menor valor del índice de Davies-Bouldin [61].

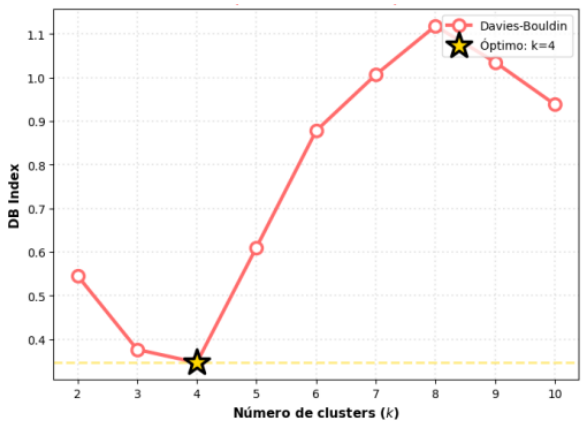

Figura 13: Gráfico de Davies-Bouldin para obtener el número óptimo de clústeres.

3. HERRAMIENTAS

En el presente capítulo se describen los recursos de software utilizados para la implementación de los algoritmos, desde el lenguaje de programación y el entorno de edición hasta las bibliotecas de procesamiento de datos. Además, se analiza la naturaleza de las bases de datos empleadas en los experimentos, detallando las características que justifican su elección para evaluar el sistema tanto en interiores como en exteriores.

3.1 Python

Para el desarrollo experimental de este trabajo se ha seleccionado Python como lenguaje de programación. La elección de esta herramienta se fundamenta en su robustez y en su posición como la herramienta de referencia dentro de los ámbitos de la ciencia de datos y el aprendizaje automático (*machine learning*).

Python [62] se caracteriza por ser un lenguaje de programación interpretado, de alto nivel y orientado a objetos, que cuenta con una semántica dinámica. Su diseño se rige por la filosofía del Zen de Python, que enfatiza la legibilidad del código y la simplicidad sintáctica, permitiendo que el desarrollo se centre en la resolución de problemas lógicos más que en la complejidad técnica del lenguaje. El uso de esta herramienta se debe a una serie de características fundamentales que la diferencian de otros entornos de programación, entre las que destacan:

- **Naturaleza interpretada:** a diferencia de los lenguajes tradicionales, Python ejecuta las instrucciones línea a línea. Esta propiedad facilita un flujo de trabajo iterativo y una depuración de errores más ágil, permitiendo validar cambios en la lógica de los algoritmos sin los tiempos de espera asociados a la compilación.
- **Semántica dinámica:** el tipado de las variables y el enlace de los objetos se resuelven en tiempo de ejecución, permitiendo una flexibilidad superior al manejar estructuras de datos heterogéneas o de tamaño variable, facilitando que el código sea más adaptable a diferentes configuraciones de entrada.
- **Gestión automática de memoria:** el lenguaje incorpora un recolector de basura (*garbage collector*) que gestiona la asignación y liberación de recursos de forma transparente al usuario. Esto minimiza la aparición de errores de bajo nivel relacionados con la memoria y permite centrar el desarrollo en la implementación de los modelos lógicos.

En consecuencia, Python constituye un entorno de experimentación integrado que permite ejecutar los algoritmos de agrupamiento, extraer métricas de rendimiento y generar representaciones gráficas. Estas capacidades son fundamentales para evaluar con precisión la eficacia del *clustering* en los experimentos realizados.

3.2 Bibliotecas

Este trabajo se apoya en un ecosistema de bibliotecas de código abierto dentro del entorno Python. Estas herramientas permiten transformar el lenguaje en una plataforma de ingeniería de datos robusta, facilitando desde la manipulación técnica de las muestras hasta la ejecución y validación de algoritmos complejos.

A continuación, se detallan las bibliotecas empleadas agrupadas por su funcionalidad en el proyecto:

3.2.1. Base computacional y manejo de datos

Estas herramientas constituyen la base computacional fundamental para el manejo de la información capturada por los sensores.

- **Pandas:** se utiliza para la gestión de *DataFrames*. Es la herramienta que permite cargar los archivos CSV de las muestras y realizar operaciones de limpieza, filtrado y organización de los datos de forma eficiente.

- **NumPy:** es el motor de cálculo numérico. Se emplea para transformar las tablas de datos en matrices multidimensionales, permitiendo realizar operaciones matemáticas rápidas, esenciales para que los algoritmos calculen distancias entre puntos.

- **SciPy (*stats.mode* y *spatial.distance.cdist*):** se utiliza el módulo *stats* en tareas de post-procesamiento para identificar la etiqueta predominante (moda). Por otro lado, *cdist* se emplea para calcular de manera eficiente la matriz de distancias entre pares de muestras.

- **Itertools:** se utiliza para la automatización de la experimentación mediante la función *product*. Esta biblioteca permite generar de forma eficiente todas las combinaciones posibles de hiperparámetros (estrategia de *Grid Search*), facilitando la ejecución sistemática de los diferentes algoritmos de *clustering* bajo múltiples configuraciones.

• **Tools (read_descriptors):** se trata de un *script* personalizado que constituye el primer paso del flujo de datos. Su función principal es la lectura del archivo CSV y la conversión de los datos crudos a formatos numéricos procesables (*arrays* de NumPy). Cabe destacar el uso del módulo re (expresiones regulares), empleado para la interpretación de las cadenas de texto de los descriptores, permitiendo transformar las secuencias de caracteres en vectores numéricos, además de extraer las etiquetas de los entornos (*scene_types*) para la posterior clasificación.

```python
import pandas as pd
import re
import numpy as np


def string2array(desc):
    data = desc.strip("[]")
    numbers = re.split(r'\s+', data)
    numbers = [num for num in numbers if num.strip()]
    array = np.array(numbers, dtype=float)
    return array

def read_descriptors(csv_file):
    csv_data = pd.read_csv(csv_file).to_dict('records')
    descriptors = np.array([string2array(str(dic["descriptor"])) for dic in csv_data])
    scenes_names = [dic["scene_name"] for dic in csv_data]
    scenes_types = [dic["scene_type"] for dic in csv_data]

    return descriptors, scenes_names, scenes_types
```

Figura 14: Código del script personalizado para extraer y formatear los datos de entrada del clustering en interiores.

3.2.2. Aprendizaje automático y preprocesamiento

Scikit-learn (sklearn) [63] es el eje central del desarrollo. Proporciona una interfaz unificada para el aprendizaje automático, dividiéndose su uso en tres fases críticas:

• **Preprocesamiento (*Normalizer*, *LabelEncoder*):** antes del agrupamiento, el *Normalizer* normaliza individualmente cada muestra para que tenga una norma unitaria. Esto proyecta los datos sobre una hiperesfera, haciendo que el agrupamiento se base en la dirección o el perfil de los datos (similitud) en lugar de en su magnitud absoluta. Por su parte, el *LabelEncoder* transforma las etiquetas textuales en valores numéricos procesables.

• **Algoritmos de *clustering*:** se implementan técnicas de distinta naturaleza para comparar su eficacia: basados en centroides (*K-Means*), en densidad (HDBSCAN, *MeanShift*) y basados en jerarquía o grafos (*AgglomerativeClustering, SpectralClustering*).

• **Métricas de evaluación:** para cuantificar el rendimiento de forma objetiva, se emplean métricas intrínsecas (*Silhouette*, Davies-Bouldin) y métricas de concordancia con la realidad (ARI, *Jaccard*, *Recall*). Además, se importa confusion_matrix para evaluar posteriormente la precisión de la clasificación.

3.2.3. Visualización

Para la interpretación visual de los resultados, se emplean las siguientes herramientas específicas:

- **Matplotlib:** es la biblioteca base para la generación de gráficos técnicos y diagramas de dispersión (*scatter plots*).
- **Seaborn:** utilizada para generar representaciones estadísticas avanzadas y estéticas, facilitando la visualización de los mapas de calor (*heatmaps*) generados a partir de las matrices de confusión.

3.3 Visual Studio Code

Para llevar a cabo la evaluación de los distintos métodos de *clustering*, así como para la tarea de localización, se ha utilizado Visual Studio Code (VS Code) [64], un editor de código fuente desarrollado por Microsoft. Se trata de una herramienta multiplataforma (disponible para Windows, macOS y Linux) que se distribuye de forma gratuita y que combina la simplicidad de un editor de código con potentes herramientas de desarrollo.

La elección de esta herramienta para el desarrollo del TFG se fundamenta en sus funcionalidades, las cuales han facilitado la implementación técnica:

- **Soporte avanzado para Python:** mediante la extensión oficial de Python, el editor habilita la funcionalidad *IntelliSense*. Esta característica ofrece autocompletado inteligente y sugerencias de parámetros en tiempo real, lo que resulta esencial al trabajar con bibliotecas complejas de análisis de datos (como scikit-learn o pandas), optimizando así la escritura de funciones y la definición de sus argumentos.
- **Terminal integrada:** VS Code permite gestionar terminales de línea de comandos directamente desde su interfaz. Esto ha facilitado la ejecución de los *scripts* de *clustering*, la instalación de dependencias (bibliotecas) y la visualización de métricas de evaluación a través de la consola, eliminando la necesidad de alternar entre diferentes ventanas o aplicaciones.
- **Depuración (*debugging*):** el editor incorpora un depurador nativo que permite ejecutar el código paso a paso y establecer puntos de interrupción (*breakpoints*). Esta característica ha sido fundamental para detectar errores lógicos en el código y validar el flujo de ejecución de los algoritmos implementados.
- **Detección de errores (*linting*):** el entorno analiza el código en tiempo real mientras se escribe, señalando errores sintácticos o advertencias de estilo previos a la

ejecución, lo que asegura un código más limpio y acorde a los estándares de Python.

En definitiva, la capacidad de Visual Studio Code para centralizar la edición, ejecución y depuración en una única interfaz unificada ha sido determinante para el desarrollo del proyecto. Esto ha permitido mantener un flujo de trabajo continuo y eficiente, reduciendo los tiempos de desarrollo y facilitando la iteración rápida en los experimentos de *clustering* y localización.

3.4 Base de datos

Para validar la eficacia de los algoritmos propuestos y asegurar que los resultados son aplicables a situaciones del mundo real, se han seleccionado dos conjuntos de datos (*datasets*) ampliamente reconocidos en la comunidad científica. La elección de estas dos bases de datos responde a la necesidad de evaluar el sistema en escenarios heterogéneos: por un lado, entornos interiores estructurados y, por otro, entornos exteriores dinámicos y de gran escala.

3.4.1 ScanNet

ScanNet [65] es una base de datos de vídeo RGB-D a gran escala que proporciona una colección exhaustiva de reconstrucciones 3D de escenas interiores. Este conjunto de datos contiene aproximadamente 2,5 millones de vistas repartidas en más de 1500 escaneos distintos, lo que lo convierte en un referente para la validación de algoritmos de comprensión de escenas y visión por computador.

A diferencia de los entornos exteriores, este conjunto de datos se centra en espacios cerrados y estructurados, ofreciendo características esenciales para la evaluación de interiores:

- **Escala y diversidad:** con más de 1500 escaneos únicos, ScanNet abarca una amplia variedad de tipos de habitaciones (oficinas, apartamentos, baños, etc.) y configuraciones de mobiliario, proporcionando una muestra representativa y densa de entornos reales.
- **Información RGB-D:** la captura se realizó utilizando sensores de profundidad *commodity*, proporcionando canales de color y profundidad alineados. Esto permite explotar tanto la apariencia visual como la geometría 3D de la escena para el análisis.
- **Verdad terreno (*ground truth*):** el *dataset* destaca por sus ricas anotaciones, que incluyen poses de cámara 3D precisas, reconstrucciones de superficie y, lo que es crítico para este estudio, el etiquetado semántico del tipo de escena.

Figura 15: Muestras del dataset ScanNet. Las imágenes muestran la segmentación semántica de diferentes estancias interiores (vista superior), donde cada color representa una categoría de objeto distinta (mobiliario, suelo, paredes, etc.), proporcionando la información de referencia (ground truth) necesaria para la evaluación.

En el contexto de este trabajo, la inclusión de ScanNet responde a dos objetivos principales:

1. **Validar el *clustering* en entornos estructurados:** permite evaluar cómo se comportan los algoritmos propuestos ante geometrías definidas y objetos estáticos típicos de interiores.

2. **Evaluación cuantitativa precisa:** gracias a las etiquetas de tipo de escena disponibles, es posible medir objetivamente la calidad de la agrupación (*clustering*) comparando los resultados del algoritmo con la clasificación real de las estancias.

3.4.2 NCLT *dataset*

El *University of Michigan North Campus Long-Term Dataset* [66] es una extensa colección de datos recopilados mediante una plataforma robótica (*segway*) que recorrió el campus de la Universidad de Michigan de manera repetida a lo largo de 15 meses.

El conjunto de datos consta de 27 sesiones de grabación (rutas) que cubren una trayectoria total de 147,4 km. Las rutas repiten consistentemente el mismo recorrido por el campus, lo que permite evaluar la localización a largo plazo revisitando los mismos lugares en diferentes momentos.

A diferencia de los entornos controlados, este conjunto de datos captura el entorno exterior en una amplia variedad de condiciones desafiantes, incluyendo:

- **Variabilidad estacional y climática:** los registros abarcan desde días soleados de verano hasta escenarios invernales con nieve acumulada, lo que introduce cambios drásticos en la apariencia visual de la escena.

• **Elementos dinámicos:** al tratarse de un campus universitario real, las grabaciones incluyen flujo constante de peatones, ciclistas y tráfico rodado, lo cual añade ruido y complejidad al análisis.

• **Verdad terreno (*ground truth*):** el *dataset* proporciona una solución de pose de alta precisión (optimizada mediante SLAM) que fusiona GPS RTK con sistemas inerciales, permitiendo conocer la ubicación exacta del robot en todo momento.

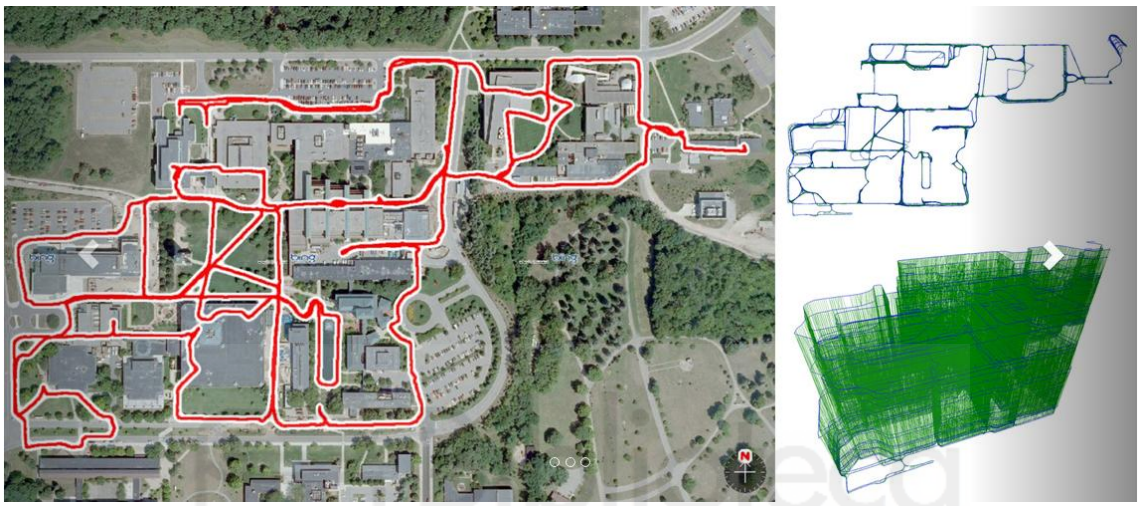

Figura 16: Visualización de las trayectorias del dataset NCLT. A la izquierda, superposición de la ruta recorrida por el robot sobre el mapa satelital del campus. A la derecha, grafo de poses que muestra la alineación espacial y la consistencia de las múltiples sesiones de grabación realizadas a lo largo de 15 meses.

En el presente trabajo, la selección de la base de datos NCLT responde a dos necesidades fundamentales:

1. Disponer de un entorno no estructurado y cambiante para validar la robustez de los algoritmos de *clustering* en exteriores.

2. Contar con datos de posicionamiento reales fiables que permitan cuantificar el error de localización obtenido mediante los métodos de *clustering* propuestos.

4. EXPERIMENTOS Y RESULTADOS

En este apartado se describen las pruebas experimentales realizadas para validar el funcionamiento y la eficacia de las técnicas de *clustering* propuestas. El objetivo principal de esta fase es evaluar el rendimiento de los algoritmos seleccionados en escenarios reales o simulados, analizando su capacidad para agrupar los datos de manera coherente bajo distintas condiciones.

Con el fin de garantizar un análisis exhaustivo, la experimentación se ha dividido en dos escenarios claramente diferenciados: entornos interiores y entornos exteriores. Esta distinción permite analizar cómo influyen las variables propias del entorno, como la densidad de obstáculos, el ruido en las mediciones o la dispersión de los datos, en la calidad de los resultados obtenidos.

No obstante, para mantener una base comparativa sólida, en ambos escenarios se han evaluado las mismas cinco técnicas, elegidas para representar distintos paradigmas de agrupamiento:

- *K-means clustering*
- HDBSCAN *clustering*
- *Mean shift clustering*
- *Hierarchical clustering*
- *Spectral clustering*

Todo el flujo de trabajo experimental se ha implementado mediante un software desarrollado en Python, el cual integra tanto la ejecución computacional de los algoritmos como el procesamiento para la obtención de resultados.

4.1 *Clustering* interior

En este apartado se presentan los experimentos realizados en entornos interiores. El objetivo principal ha sido comprobar cómo se comportan las distintas técnicas de agrupamiento cuando trabajan con datos que representan espacios cerrados, como habitaciones u oficinas, donde la distribución de los objetos es compleja.

Para realizar estas pruebas, se ha utilizado la base de datos ScanNet, que contiene una gran colección de escenas interiores escaneadas en 3D. Sin embargo, no se ha trabajado directamente con los datos brutos de dicha base.

Para poder aplicar los algoritmos de *clustering* de forma efectiva, primero se han procesado estas escenas utilizando una red neuronal llamada Sonata. La función de esta red ha sido generar los descriptores de los datos. Es decir, esta arquitectura analiza la información de ScanNet y extrae las características matemáticas más relevantes de cada punto. Posteriormente, las técnicas de *clustering* seleccionadas se ejecutan sobre estos descriptores procesados, y no sobre la nube de puntos original, lo que permite una clasificación mucho más precisa basada en las propiedades aprendidas por la red, tal y como se observa en el esquema de la Figura 17.

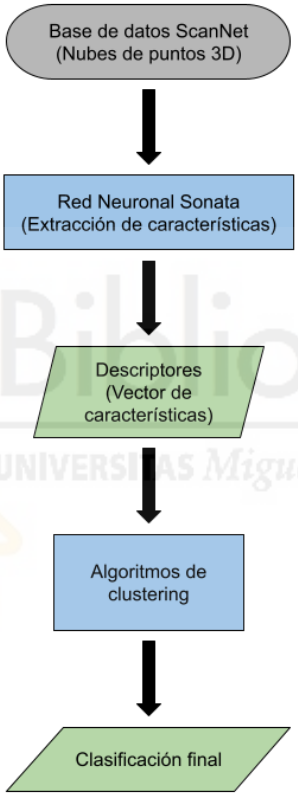

Figura 17: Diagrama del flujo de trabajo utilizado para el clustering en entornos interiores.

Para analizar la eficacia de los algoritmos de *clustering* propuestos, además de calcular distintas métricas, se ha implementado la generación de matrices de confusión para cada método. Esto es fundamental para el análisis en interiores, ya que las métricas por sí solas dan una visión general, mientras que las matrices de confusión permiten identificar exactamente dónde falla cada algoritmo, proporcionando un desglose detallado de los errores de clasificación y facilitando una interpretación más profunda de los resultados.

Un paso previo fundamental para la validación ha sido determinar el número óptimo de grupos (*k*) a identificar. Tras analizar la base de datos ScanNet, se ha observado que las escenas utilizadas se clasifican en 11 tipologías de estancias diferenciadas.

La Tabla 1 detalla estas categorías, mostrando la correspondencia entre las etiquetas originales del *dataset* y la nomenclatura utilizada en este trabajo, así como la distribución cuantitativa de las 44 escenas que componen la muestra experimental.

Tipo de estancia (ScanNet)	**Tipo de estancia (TFG)**	**Número de escenas**
Bathroom	Baño	3
Office	Oficina	6
Kitchen	Cocina	4
Bedroom / Hotel	Dormitorio	5
Hallway	Pasillo	3
Conference Room	Sala de conferencias	4
Laundry Room	Lavandería	4
Storage / Basement / Garage	Almacén	3
Bookstore / Library	Librería	4
Living room / Lounge	Sala de estar	5
Copy / Mail Room	Cuarto de reprografía	3

Tabla 1: Relación de tipologías de estancia y número de escenas utilizadas.

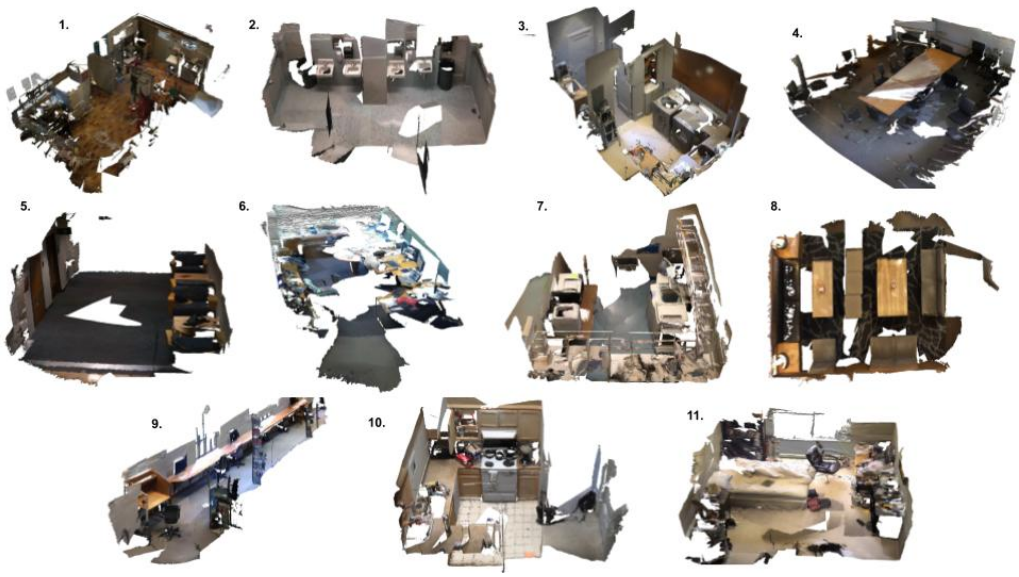

Figura 18: Muestras visuales representativas de las 11 tipologías de estancias analizadas. (1.) Almacén, (2.) Baño, (3.) Lavandería, (4.) Sala de conferencias, (5.) Pasillo, (6.) Oficina, (7.) Cuarto de reprografía, (8.) Sala de estar, (9.) Librería, (10.) Cocina, (11.) Dormitorio.

Por este motivo, para garantizar una comparativa justa y supervisada, el objetivo experimental se ha fijado en obtener una partición de k = 11 clústeres en todas las técnicas. La configuración se ha realizado de la siguiente manera según la naturaleza del algoritmo:

- **Para algoritmos paramétricos (*k-means, spectral clustering, hierarchical clustering*):** se ha introducido explícitamente el parámetro de entrada k = 11, forzando al algoritmo a dividir el espacio de descriptores en once regiones.

- **Para algoritmos basados en densidad (HDBSCAN, *mean shift*):** dado que estas técnicas no aceptan el número de clústeres como entrada directa, se ha realizado un proceso de ajuste de hiperparámetros (tales como el *bandwidth* o el tamaño mínimo de clúster) mediante un proceso iterativo, calibrándolos hasta obtener una segmentación que convergiera en 11 grupos, alineándose así con la realidad semántica del *dataset*.

Adicionalmente, se ha empleado el ARI (*Adjusted Rand Index*) como criterio de calidad para refinar esta selección. El procedimiento no se ha limitado únicamente a buscar la convergencia en k = 11, sino que se ha evaluado qué combinación de hiperparámetros maximiza dicho índice. De este modo, se ha seleccionado la mejor configuración posible, asegurando que la partición resultante no solo cumple con el número de grupos objetivo, sino que presenta la mayor fidelidad y concordancia respecto a la clasificación de referencia del *dataset*.

Dado que se dispone de las etiquetas reales (*ground truth*) de las escenas de ScanNet, la evaluación combina dos tipos de métricas para obtener un análisis más detallado:

1. Métricas externas: se han calculado el ARI (*Adjusted Rand Index*), *recall* e IOU (*Intersection over Union*). Estas métricas cuantifican el grado de coincidencia entre los clústeres generados y las etiquetas reales (*ground truth*) del *dataset*. Un valor elevado implica una alta fidelidad semántica, indicando que el algoritmo ha sido capaz de agrupar correctamente los distintos tipos de estancias.

2. Métricas internas: se emplean el coeficiente de *silhouette* y el índice Davies-Bouldin. Estas métricas no dependen de las etiquetas reales, sino que evalúan la calidad de la estructura geométrica de los grupos, analizando la relación entre la distancia *intra-cluster* (compacidad) y la distancia *inter-cluster* (separación).

Su interpretación varía según el índice: para *silhouette*, un valor elevado (cercano a 1) es deseable, ya que indica clústeres bien definidos. Por el contrario, el índice Davies-Bouldin es una métrica de minimización, donde un valor más bajo indica una mejor partición, caracterizada por una menor dispersión *intra-cluster* y mayor

distancia entre los grupos.

A continuación, se presentan los resultados obtenidos para las cinco técnicas evaluadas:

Método	Recall	ARI	IOU	Silhouette	Davies-Bouldin	Clústeres
Spectral	**0.705**	**0.549**	**0.579**	0.203	1.243	11
Hierarchical	0.656	0.434	0.528	**0.294**	0.868	11
K-Means	0.573	0.311	0.405	0.265	0.952	11
Mean Shift	0.544	0.246	0.380	0.282	**0.755**	11
HDBSCAN	0.474	0.273	0.355	0.213	1.286	11

Tabla 2: Resultados obtenidos de las distintas técnicas de clustering evaluadas en entornos interiores.

Al analizar los datos, se observa una clara superioridad del método *spectral clustering* en cuanto a fidelidad con la realidad. Este algoritmo alcanza el mejor ARI (0.549) e IOU (0.579), y destaca especialmente por obtener el mayor *recall* (0.705). Este último dato es revelador, ya que indica que esta técnica es la que mejor ha conseguido agrupar la totalidad de los descriptores pertenecientes a una misma estancia (por ejemplo, consiguiendo que todos los descriptores de baño queden en un único grupo), interpretando correctamente la complejidad generada por Sonata.

Sin embargo, se produce un fenómeno interesante al observar las métricas internas. Aunque *spectral clustering* es el que mejor acierta (métricas externas altas), obtiene el valor más bajo en *silhouette* (0.203). Por el contrario, técnicas como *mean shift* o *hierarchical* obtienen mejores valores de cohesión interna, con un mejor *silhouette* y un índice Davies-Bouldin de 0.755 para *mean shift* (Tabla 2). No obstante, esta cohesión geométrica no siempre se traduce en una mejor clasificación real: mientras que *mean shift* presenta dificultades, *hierarchical* mantiene un rendimiento aceptable con un *recall* de 0.656 (Tabla 2), evidenciando que una buena separación geométrica no garantiza la fidelidad semántica.

Esto sugiere que la distribución de los datos en el espacio de descriptores no es necesariamente esférica ni compacta (lo que favorecería a *k-means* o *mean shift*), sino que presenta estructuras topológicas complejas y no convexas. *Spectral clustering* demuestra su capacidad para adaptarse a estas geometrías irregulares y capturar

correctamente la identidad semántica de los objetos, aun cuando esto implique generar agrupaciones que penalizan la compacidad geométrica evaluada por las métricas internas.

Para complementar el análisis cuantitativo y comprender la naturaleza exacta de las asignaciones realizadas, se evalúan las matrices de confusión para las cinco técnicas. En estas representaciones visuales, el eje vertical corresponde a las clases reales (*ground truth* de las 11 estancias) y el eje horizontal a las predicciones de los algoritmos. Los valores numéricos dentro de cada celda representan la cantidad absoluta de escenas asignadas a cada clúster. Una diagonal principal con valores elevados indica un alto grado de acierto, mientras que cualquier valor situado fuera de esta diagonal representa un error de clasificación, reflejando las ocasiones en las que el algoritmo ha etiquetado incorrectamente una estancia real.

A continuación, se presentan las matrices de confusión obtenidas para cada uno de los métodos evaluados:

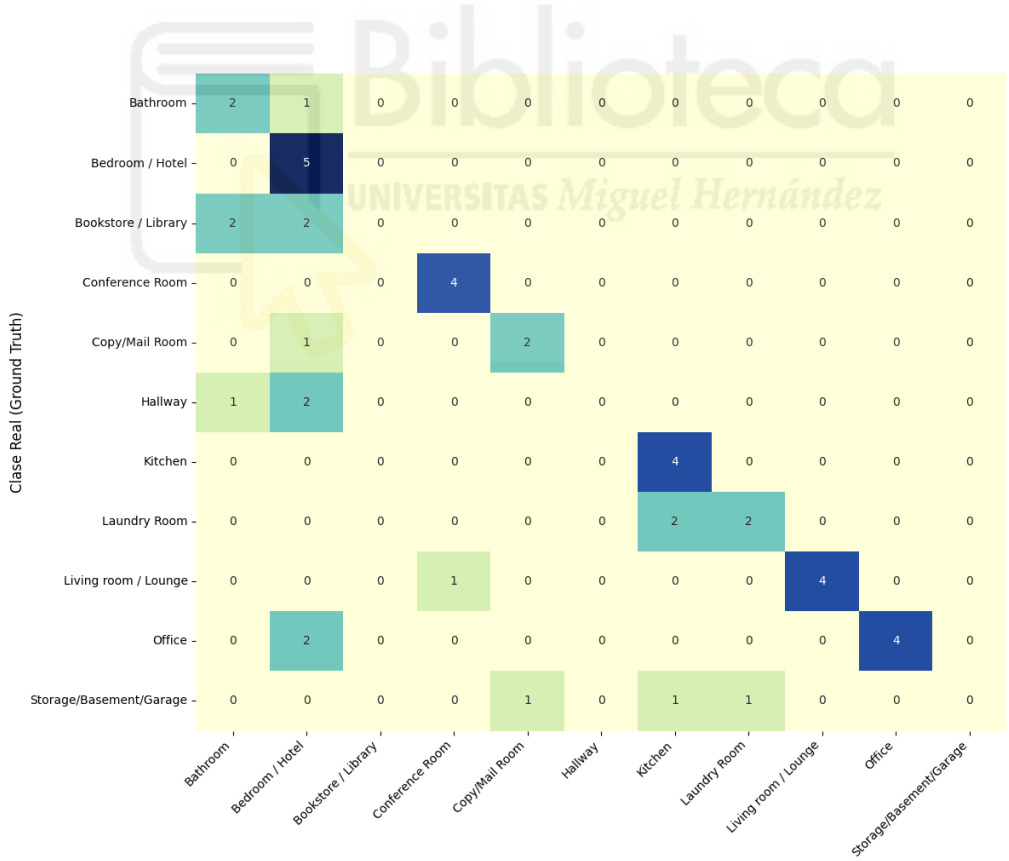

Figura 19: Matriz de confusión de los resultados obtenidos mediante k-means.

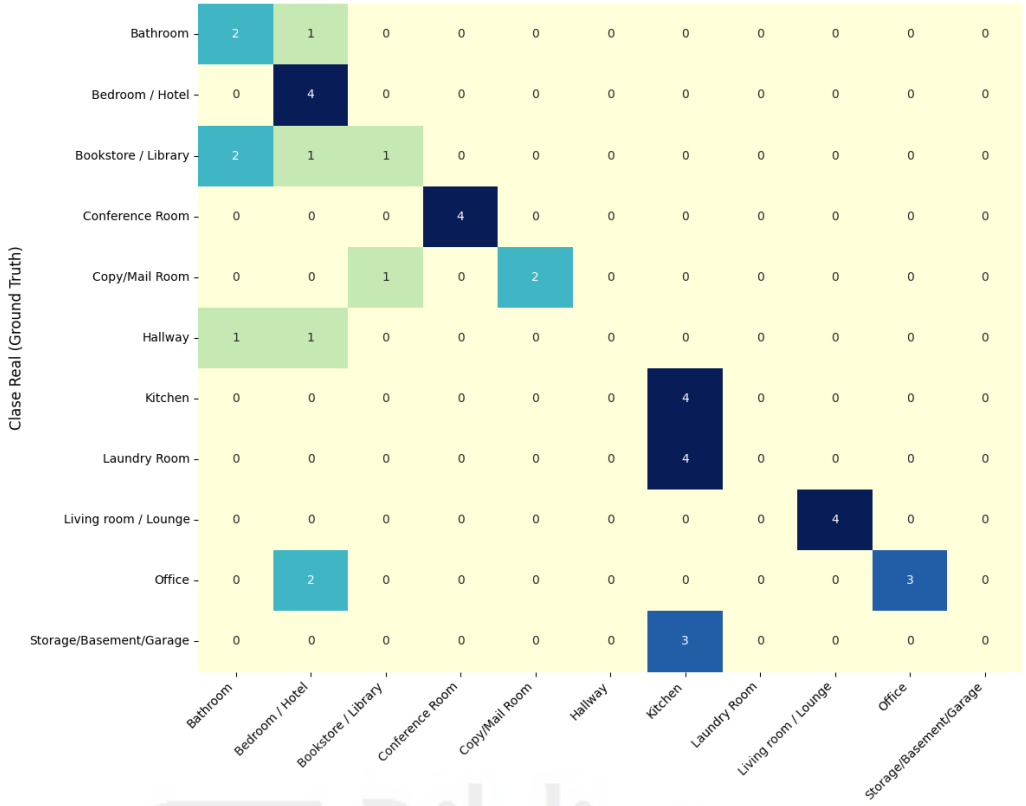

Figura 20: Matriz de confusión de los resultados obtenidos mediante HDBSCAN.

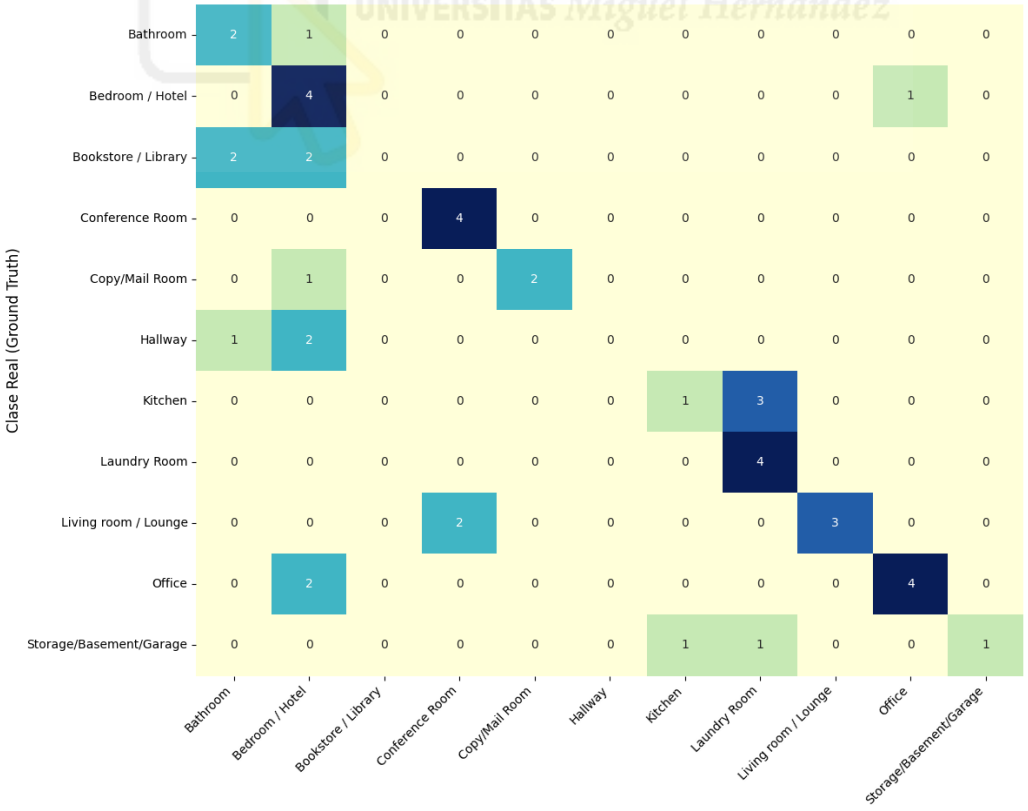

Figura 21: Matriz de confusión de los resultados obtenidos mediante mean shift.

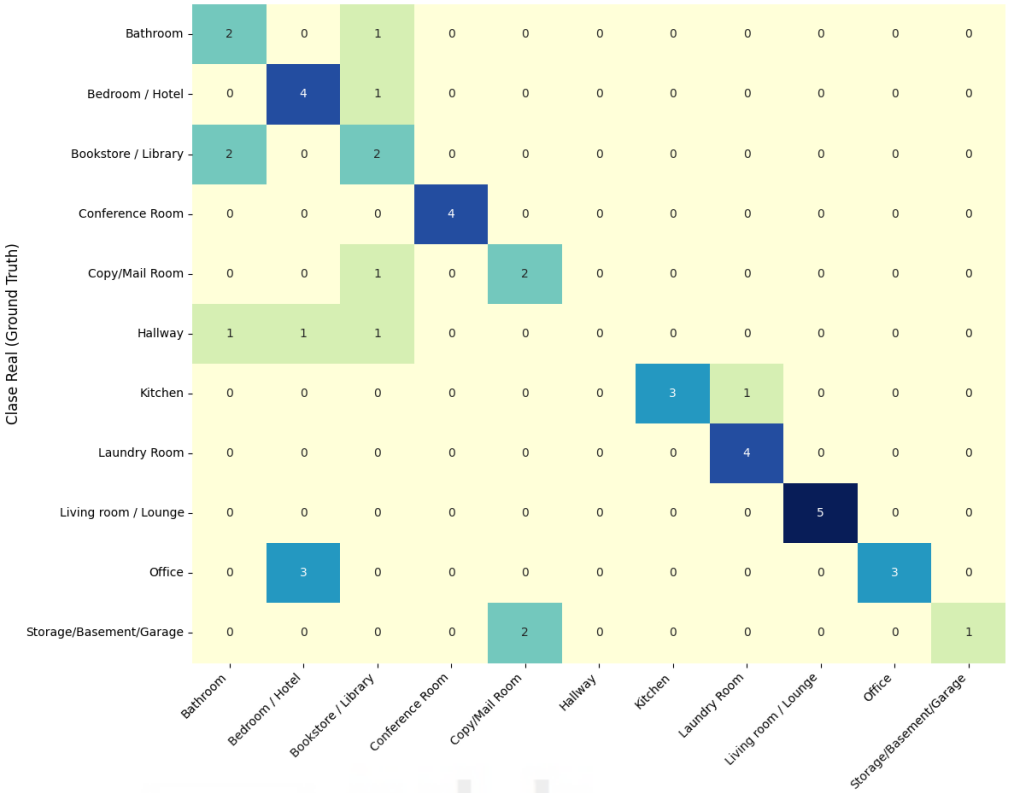

Figura 22: Matriz de confusión de los resultados obtenidos mediante hierarchical clustering.

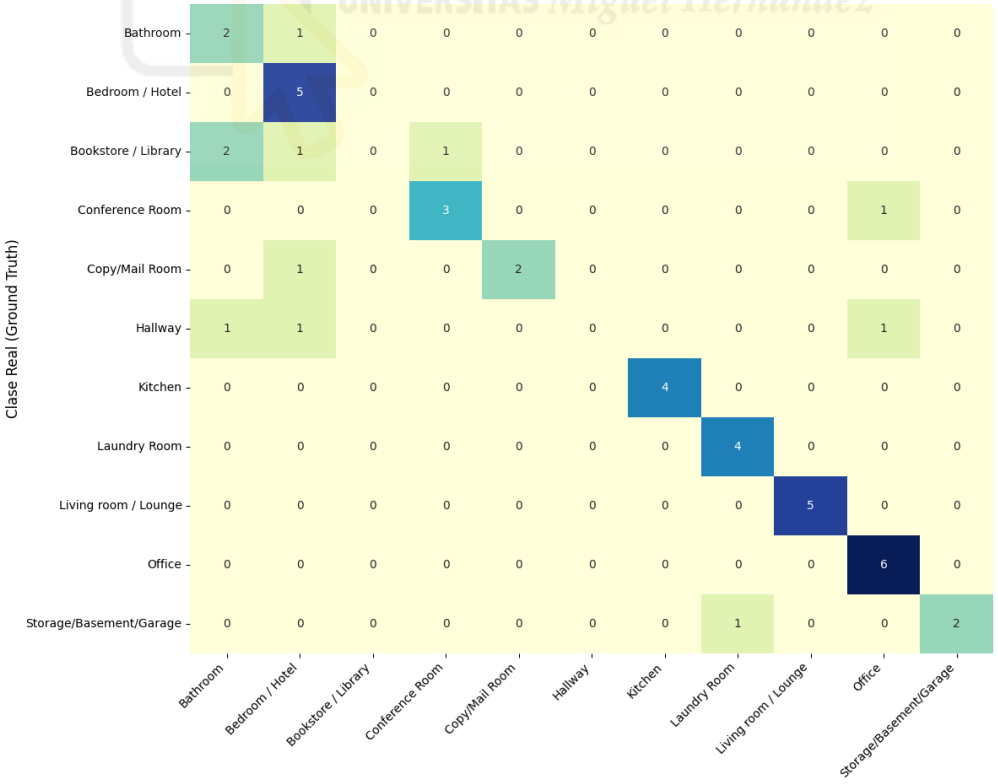

Figura 23: Matriz de confusión de los resultados obtenidos mediante spectral clustering.

A partir del análisis de estas matrices, se puede observar cómo cada algoritmo ha realizado la agrupación de los clústeres respecto a la realidad semántica del *dataset*. Esta comparativa gráfica refleja fielmente los resultados cuantitativos obtenidos anteriormente, permitiendo identificar no solo el rendimiento global, sino los patrones de error específicos de cada técnica.

Como punto de partida, se confirma visualmente en la Figura 23 que *spectral clustering* es la técnica más robusta. Al ser el método que alcanzó los valores más altos en las métricas externas, presenta la diagonal principal más definida y limpia. Esto evidencia que es la técnica que maximiza la correspondencia entre los grupos predichos y las estancias originales, logrando resultados notables como la clasificación perfecta e íntegra de estancias como oficina, sala de estar y cocina. Este logro es especialmente relevante, dado que estancias como la cocina resultaron críticas para los algoritmos de densidad (Figura 20), mientras que las oficinas generaron una notable confusión en las técnicas jerárquicas (Figura 22).

Un aspecto crítico transversal a todos los métodos es la incapacidad generalizada para identificar correctamente la clase pasillo. Se observa que los pasillos no conforman un clúster propio, sino que son sistemáticamente mal clasificados y dispersados hacia otras categorías, principalmente dormitorio y baño. Este fenómeno sugiere que, desde el punto de vista de los descriptores geométricos y de color, los pasillos carecen de una identidad distintiva y actúan como zonas de transición que los algoritmos confunden con estancias vecinas.

Por último, el análisis detallado revela dos comportamientos opuestos en el resto de técnicas. En los algoritmos de densidad (HDBSCAN y *mean shift*), se aprecia una clara tendencia a fusionar estancias funcionales distintas que comparten características similares. El caso más evidente se encuentra en la Figura 20 (HDBSCAN), donde el algoritmo ha colapsado las cocinas, lavanderías y almacenes en un único grupo.

Por el contrario, los métodos que fuerzan la partición, como *k-means* y *hierarchical clustering*, evitaron esta fusión masiva, pero mostraron menor precisión que el enfoque espectral. Mientras que *k-means* presenta una dispersión de errores más generalizada a lo largo de varias categorías (Figura 19), *hierarchical* introduce una confusión específica entre clases con mobiliario parecido, dividiendo erróneamente las muestras de oficina y mezclándolas con dormitorio (Figura 22), imprecisiones que *spectral clustering* logró resolver (Figura 23).

4.2 *Clustering* exterior

En este apartado se presentan los experimentos realizados en entornos exteriores. El objetivo principal ha sido comprobar cómo se comportan las distintas técnicas de agrupamiento cuando trabajan con datos que representan espacios urbanos abiertos y dinámicos, donde la variabilidad del entorno (cambios en la vegetación, iluminación y tránsito) introduce una complejidad diferente a la de los interiores.

Para realizar estas pruebas, se ha utilizado la base de datos NCLT (*North Campus Long-Term*), que contiene recorridos extensos escaneados en un campus universitario a lo largo de varios meses. Al igual que en el experimento anterior, no se ha trabajado directamente con los datos brutos.

Para poder aplicar los algoritmos de *clustering* de forma efectiva, se ha seguido el mismo flujo de trabajo: procesar las escenas de NCLT utilizando la red neuronal MinkUNeXt para generar los descriptores de datos. De este modo, las técnicas de *clustering* seleccionadas se ejecutan sobre las características matemáticas extraídas por esta arquitectura, permitiendo una clasificación mucho más precisa basada en las propiedades profundas aprendidas por la red y no solo en la nube de puntos original, tal y como se detalla en el esquema de la Figura 24.

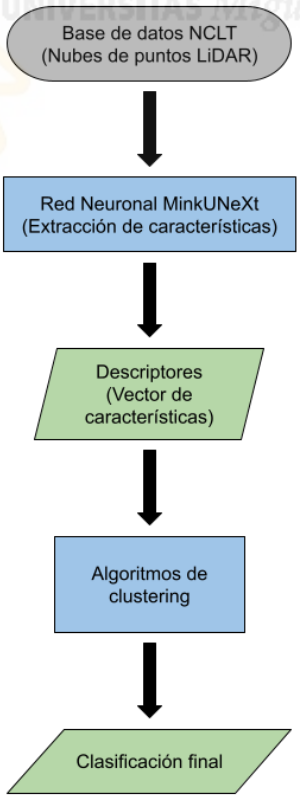

Figura 24: Diagrama del flujo de trabajo utilizado para el clustering en entornos exteriores.

A diferencia del experimento en interiores, donde se evaluaba la eficacia del agrupamiento en un escenario estático, en este caso se ha diseñado un procedimiento de validación temporal con el objetivo de verificar la robustez de los algoritmos frente al paso del tiempo. Para ello, la metodología consiste en entrenar el modelo utilizando los descriptores extraídos de las sesiones del mes de enero y, a continuación, evaluar el rendimiento del *clustering* sobre los descriptores obtenidos en sesiones posteriores (febrero y mayo), comprobando así la estabilidad de las predicciones ante cambios estacionales.

Esto permite determinar si la estructura aprendida en invierno sigue siendo válida para localizar y agrupar datos en primavera, analizando cómo afecta el cambio estacional a la consistencia del agrupamiento.

Con el fin de cuantificar la precisión del *clustering* en tareas de localización, se definió un procedimiento para calcular el error de localización utilizando los conjuntos de datos de prueba de febrero y mayo. La definición de este cálculo evolucionó durante el experimento al detectarse limitaciones en el planteamiento original.

En una primera instancia, el experimento se diseñó para utilizar los centroides de los clústeres como única referencia. La metodología consistía en calcular la distancia euclídea entre cada *scan* de prueba (correspondiente a los descriptores de febrero y mayo) y los centroides del modelo entrenado en enero, utilizando esta medida con un doble propósito: asignar el clúster basándose en la cercanía al centro y estimar la localización, asumiendo que la posición del robot coincidía con la del propio centroide.

Sin embargo, los resultados fueron excesivamente elevados. Esto se debe a que un clúster en exteriores puede abarcar una zona física muy extensa; por tanto, simplificar la posición de todo un grupo a un único punto central introducía un margen de error inaceptable.

Para solucionar este problema sin perder la eficacia del agrupamiento, se modificó el algoritmo adoptando una estrategia híbrida que distingue entre clasificación y localización:

1. Asignación de clúster: se mantiene el uso de los centroides para determinar a qué grupo pertenece el *scan* de prueba. Se calcula la distancia euclídea a todos los centros y se asigna el clúster del más cercano.

2. Cálculo del error (vecino más cercano): una vez seleccionado el clúster candidato, ya no se utiliza el centroide para localizar. En su lugar, el algoritmo accede a los descriptores de entrenamiento originales que conforman ese clúster específico y busca el descriptor más similar (*nearest neighbor*) al *scan*.

3. **Resultado:** el error de localización se calcula finalmente como la distancia euclídea mínima entre la posición real del *scan* de prueba y la de este descriptor vecino encontrado dentro del grupo.

A pesar de esta mejora técnica, se observó que los errores de localización seguían siendo muy elevados, lo que dificultaba la interpretación de los resultados. Para obtener una evaluación más robusta y operativa, se decidió incorporar dos métricas adicionales basadas en tasas de éxito:

- **Porcentaje de acierto de clúster (% Clúster Correcto):** esta métrica evalúa la fiabilidad de la primera etapa del algoritmo. Cuantifica el porcentaje de veces que el método de los centroides asigna el *scan* de prueba al grupo correcto; es decir, aquel clúster que contiene efectivamente la zona física donde se encuentra el robot. Un fallo en esta etapa implica buscar la posición en una zona errónea del mapa, haciendo imposible la localización precisa.

- **Porcentaje de casos con error < 5 metros:** dado que en exteriores el error medio puede verse distorsionado por unos pocos fallos catastróficos, este indicador cuantifica la fiabilidad del posicionamiento. Representa la frecuencia con la que el sistema logra una localización dentro del margen de tolerancia establecido (error inferior a 5 metros), descartando los casos de divergencia extrema.

El siguiente fragmento de código muestra la implementación de esta lógica: se determina el clúster, se busca el vecino más cercano dentro de él para calcular el error de localización y, finalmente, se extraen las métricas descritas anteriormente:

```
37  # FUNCIÓN DE EVALUACIÓN
38  def evaluar_configuracion(desc_train, pos_train, labels_train, desc_test, pos_test):
39      # Evitar errores de ruido
40      unique_labels = set(labels_train)
41      if -1 in unique_labels: unique_labels.remove(-1)
42      if len(unique_labels) < 2:
43          return np.nan, np.nan, np.nan
44
45      #  Construir mapa topológico (Centroides)
46      df_comb = pd.DataFrame(desc_train)
47      df_comb['cluster'] = labels_train
48      # Calcular los centroides ignorando el ruido
49      centroides = df_comb[df_comb['cluster'] != -1].groupby('cluster').mean()
50      centroides_desc = centroides.values
51      etiquetas_reales = centroides.index.values
52
53      # Asignar a cada scan el clúster más cercano
54      dists_test = cdist(desc_test, centroides_desc, metric = 'euclidean')
55      indices_cluster_asignados = np.argmin(dists_test, axis = 1)
56
57      # Comparar el scan de test(Febrero) con los descriptores del cluster seleccionado
58      pos_estimada = []
59      aciertos_cluster = 0
```

```
60
61      for i, idx_centroide in enumerate(indices_cluster_asignados):
62          label_predicha = etiquetas_reales[idx_centroide]
63
64          # Filtar los descriptores  y posiciones del cluster asignado
65          mask = (labels_train == label_predicha)
66          desc_cluster = desc_train[mask]
67          pos_cluster = pos_train[mask]
68
69          # Asignar la posición al scan de test del descriptor más parecido
70          dists = cdist([desc_test[i]], desc_cluster, metric = 'euclidean')
71          ind_vecino = np.argmin(dists)
72          pos_estimada.append(pos_cluster[ind_vecino])
73          # Carcular la distancia real del robot a todos los puntos de entrenamiento
74          dist_gt = cdist(pos_test[i].reshape(1, -1), pos_train, metric = 'euclidean')
75          # Buscamos los índices donde la localización es menor a 5m
76          ind_cercanos_gt = np.where(dist_gt[0] < 5)[0]
77
78          if len(ind_cercanos_gt) > 0:
79              labels_cercanos = labels_train[ind_cercanos_gt]
80              if label_predicha in labels_cercanos:
81                  aciertos_cluster += 1
82      pos_estimada = np.array(pos_estimada)
83      # Calculo de métricas
84      errores_loc = np.linalg.norm(pos_estimada - pos_test, axis=1)
85      # Error Medio
86      error_medio = np.mean(errores_loc)
87      # Error Mediana
88      error_mediana = np.median(errores_loc)
89      # Porcentaje Acierto Cluster
90      pct_acierto_cluster = (aciertos_cluster / len(errores_loc)) * 100
91      # 4. Porcentaje < 5m
92      pct_menor_5m = (len(errores_loc[errores_loc < 5]) / len(errores_loc)) * 100
93      # Retornamos las 4 variables
94      return pct_acierto_cluster, pct_menor_5m, error_medio, error_mediana
```

(variable) ind_cercanos_gt: NDArray[intp]

Figura 25: Código de la función que permite obtener las métricas del clustering de exteriores.

Una vez detallada la implementación técnica y establecidas las métricas de evaluación, se procedió a la ejecución de los experimentos sobre los conjuntos de datos de evaluación estacional. Con el fin de facilitar la interpretación de los datos y evitar redundancias en el análisis posterior, la exposición de los resultados se ha organizado estructurando los algoritmos según su lógica de parametrización y comportamiento operativo.

En primer lugar, se examinarán conjuntamente las técnicas de partición (*k-means y spectral clustering*) junto con el enfoque jerárquico (*hierarchical clustering*). A pesar de sus distintos fundamentos algorítmicos, estos tres métodos comparten una característica crítica para el diseño del experimento: la necesidad de predefinir el número de grupos objetivo (*k*) o el nivel de corte del dendrograma. Esta particularidad permite evaluar de forma directa y comparativa cómo influye la variación del número de clústeres en la precisión de la localización.

En cuanto a la configuración específica de estos algoritmos, cabe destacar que se han seleccionado los hiperparámetros que demostraron un rendimiento superior durante una

fase preliminar de ajuste. Concretamente, se ha empleado el criterio de enlace *ward* (*linkage* = *ward*) para el método jerárquico y la estrategia de vecinos más cercanos (*affinity* = *nearest_neighbors*) para el método espectral. La justificación empírica de esta elección, respaldada por las tablas de ejecución generadas en Python durante la optimización, se encuentra detallada en el Anexo 3.

Posteriormente, se abordarán los métodos basados en densidad, cuyo funcionamiento difiere al introducir la detección automática de grupos y la gestión del ruido.

A continuación, se presentan los resultados correspondientes a este primer bloque de métodos, donde se detalla la evolución de las métricas al variar el número de clústeres en el mes de febrero.

Método	K	Acierto (%)	Error <5 (%)
K-Means	2	97.0	35.3
K-Means	3	94.3	35.0
K-Means	4	91.7	35.0
K-Means	5	90.7	34.8
K-Means	6	87.3	34.3
K-Means	7	85.8	33.9
K-Means	8	85.0	34.5
K-Means	9	84.2	34.2
K-Means	10	84.1	34.1
K-Means	11	78.8	32.9
K-Means	12	77.0	33.2
K-Means	13	75.0	32.6
K-Means	14	76.5	32.9
K-Means	15	74.0	32.3
K-Means	16	73.6	33.1
K-Means	17	74.9	32.5
K-Means	18	71.9	32.7
K-Means	19	74.0	33.2

Tabla 3: Resultados k-means febrero.

Método	K	Linkage	Acierto (%)	Error <5 (%)
Hierarchical	2	ward	95.1	33.4
Hierarchical	3	ward	91.5	32.6
Hierarchical	4	ward	88.7	32.1
Hierarchical	5	ward	87.3	32.3
Hierarchical	6	ward	84.8	31.3
Hierarchical	7	ward	82.4	31.6
Hierarchical	8	ward	81.5	31.4
Hierarchical	9	ward	78.6	30.6
Hierarchical	10	ward	75.2	30.9
Hierarchical	11	ward	75.0	30.8
Hierarchical	12	ward	73.6	30.3
Hierarchical	13	ward	73.6	30.2
Hierarchical	14	ward	73.5	30.3
Hierarchical	15	ward	73.6	30.5
Hierarchical	16	ward	73.3	30.5
Hierarchical	17	ward	71.5	30.1
Hierarchical	18	ward	71.5	30.3
Hierarchical	19	ward	71.4	30.4

Tabla 4: Resultados hierarchical febrero.

Método	K	Affinity	Acierto (%)	Error <5 (%)
Spectral	2	nearest_neighbors	96.6	34.4
Spectral	3	nearest_neighbors	93.9	33.5
Spectral	4	nearest_neighbors	92.7	33.9
Spectral	5	nearest_neighbors	89.9	33.7
Spectral	6	nearest_neighbors	90.6	34.5
Spectral	7	nearest_neighbors	89.3	33.9
Spectral	8	nearest_neighbors	86.9	32.7
Spectral	9	nearest_neighbors	84.8	32.1
Spectral	10	nearest_neighbors	80.3	31.5
Spectral	11	nearest_neighbors	76.5	31.0
Spectral	12	nearest_neighbors	73.6	30.8
Spectral	13	nearest_neighbors	72.5	30.3
Spectral	14	nearest_neighbors	72.0	29.9
Spectral	15	nearest_neighbors	71.9	30.5
Spectral	16	nearest_neighbors	71.1	30.0
Spectral	17	nearest_neighbors	70.4	28.9
Spectral	18	nearest_neighbors	70.9	29.1
Spectral	19	nearest_neighbors	68.3	29.2

Tabla 5: Resultados spectral febrero.

Para complementar los datos numéricos expuestos anteriormente, se presentan a continuación las gráficas de resultados correspondientes al mes de febrero. Estas figuras permiten visualizar la evolución del comportamiento de los métodos *k-means*, *hierarchical* y *spectral* al variar el número de clústeres (*k*), facilitando la identificación de tendencias tanto en el porcentaje de acierto como en la precisión de la localización.

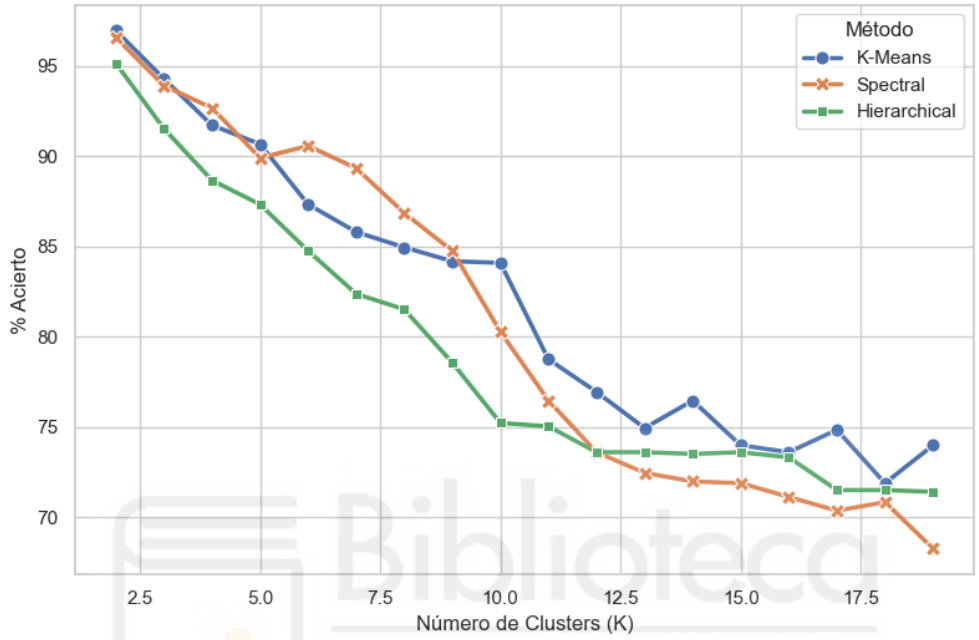

Figura 26: Gráfico de resultados del porcentaje de casos en el que el clúster asignado es correcto en el mes de febrero de los métodos de clustering k-means, hierarchical y spectral.

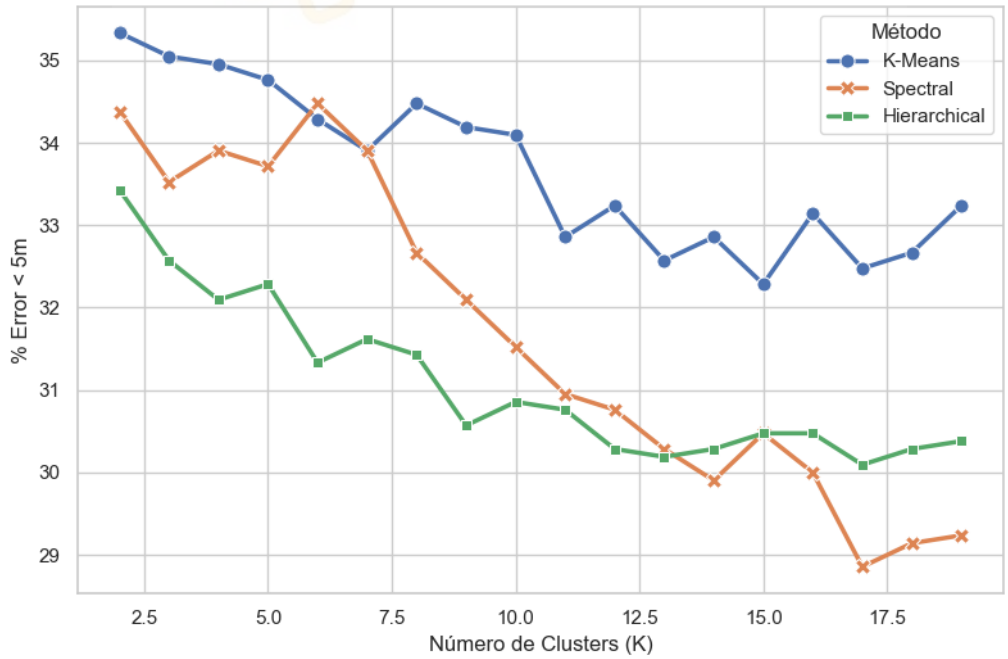

Figura 27: Gráfico de resultados del porcentaje de casos en el que el error de localización es menor a 5m en el mes de febrero de los métodos de clustering k-means, hierarchical y spectral.

A partir de los resultados representados en las tablas y gráficos, se procede a describir el comportamiento de los tres primeros algoritmos evaluados durante el mes de febrero. El análisis revela una tendencia generalizada de descenso en el rendimiento conforme aumenta la complejidad del modelo (número de clústeres), aunque con matices diferenciados según la técnica empleada.

En lo referente al porcentaje de casos en los que el clúster asignado es correcto, se observa una correlación inversa entre el parámetro *k* y la tasa de acierto, tal y como se refleja en la Figura 26. Independientemente del método, los valores máximos se obtienen cuando menor es el número de clústeres. En este punto inicial, el algoritmo *k-means* alcanza un 97.0% (Tabla 3), seguido por *spectral clustering* con un 96.6% (Tabla 5) y *hierarchical clustering* con un 95.1% (Tabla 4). A medida que el parámetro *k* aumenta, el porcentaje de casos en los que el clúster asignado es correcto disminuye progresivamente en todos los escenarios.

Por su parte, la métrica de precisión de localización, que indica el porcentaje de casos en los que el error de localización ha sido menor a 5 metros, muestra patrones de comportamiento mucho más heterogéneos (Figura 27).

- *K-means* muestra un descenso progresivo en el tramo inicial, pero pierde consistencia en la segunda mitad de la gráfica (a partir de *k*=10), donde comienza a presentar un comportamiento oscilante con subidas y bajadas constantes. En términos de magnitud absoluta, este método registró un error medio de localización que osciló entre 121.52 y 129.75 metros, con una mediana situada en el rango de 20.21 a 37.40 metros.

- *Hierarchical clustering* presenta la dinámica inversa: muestra inestabilidad en el tramo inicial (con fluctuaciones visibles entre *k*=4 y *k*=10), pero tiende a estabilizarse en el tramo final, manteniendo una línea de descenso mucho más suave y plana que *k-means* en los valores altos. Respecto a sus valores absolutos, el error de localización medio se mantuvo en un intervalo de 123.98 a 134.70 metros, mientras que la mediana varió entre 32.66 y 43.16 metros.

Finalmente, el algoritmo *spectral clustering* se desmarca de ambos comportamientos al ser el único que mejora su rendimiento tras el inicio. Tras comenzar con un 34.4%, el método evoluciona positivamente hasta alcanzar su máximo rendimiento absoluto en la configuración de *k*=6, registrando un 34.5% de precisión (Tabla 5). Este hito no solo representa su mejor resultado global, sino que en ese punto supera también a las otras alternativas. No obstante, esta ventaja desaparece en los valores más altos de *k*, donde sufre una caída acentuada y se alinea con la tendencia general de descenso que

experimentan el resto de métodos (Figura 27). En términos de magnitudes absolutas, este método obtuvo un error de localización medio que osciló entre 125.10 y 141.06 metros, mientras que la mediana sufrió una degradación significativa, pasando de 25.56 metros en el mejor caso hasta alcanzar los 63.80 metros.

Una vez analizado el comportamiento de los modelos *k-means, hierarchical y spectral* en el escenario de febrero, se presentan a continuación los resultados correspondientes al mes de mayo. Estas tablas y gráficos permitirán observar cómo varía el rendimiento de los algoritmos al enfrentarse a un entorno más cambiante y complejo.

Método	K	Acierto (%)	Error <5 (%)
K-Means	2	93.6	23.3
K-Means	3	93.0	23.3
K-Means	4	82.9	22.2
K-Means	5	83.4	22.0
K-Means	6	80.7	22.5
K-Means	7	76.2	22.7
K-Means	8	74.6	22.7
K-Means	9	74.2	22.6
K-Means	10	73.5	22.5
K-Means	11	71.4	22.0
K-Means	12	68.8	21.0
K-Means	13	69.7	22.4
K-Means	14	67.9	21.5
K-Means	15	64.8	21.8
K-Means	16	64.1	22.0
K-Means	17	61.8	21.3
K-Means	18	61.9	20.9
K-Means	19	63.5	21.2

Tabla 6: Resultados k-means mayo.

Método	K	Linkage	Acierto (%)	Error <5 (%)
Hierarchical	2	ward	94.7	23.0
Hierarchical	3	ward	91.0	21.9
Hierarchical	4	ward	88.1	22.8
Hierarchical	5	ward	78.9	21.9
Hierarchical	6	ward	76.7	21.7
Hierarchical	7	ward	73.3	21.5
Hierarchical	8	ward	72.7	21.3
Hierarchical	9	ward	70.1	20.7
Hierarchical	10	ward	66.9	20.2
Hierarchical	11	ward	66.5	19.8
Hierarchical	12	ward	64.0	19.7
Hierarchical	13	ward	64.0	19.8
Hierarchical	14	ward	63.5	20.0
Hierarchical	15	ward	63.5	20.4
Hierarchical	16	ward	63.5	20.3
Hierarchical	17	ward	62.5	20.2
Hierarchical	18	ward	61.9	20.3
Hierarchical	19	ward	61.1	20.3

Tabla 7: Resultados hierarchical mayo.

Método	K	Affinity	Acierto (%)	Error <5 (%)
Spectral	2	nearest_neighbors	88.7	22.0
Spectral	3	nearest_neighbors	94.9	22.7
Spectral	4	nearest_neighbors	83.0	21.8
Spectral	5	nearest_neighbors	83.7	23.8
Spectral	6	nearest_neighbors	81.1	22.3
Spectral	7	nearest_neighbors	76.8	21.5
Spectral	8	nearest_neighbors	72.8	20.8
Spectral	9	nearest_neighbors	72.3	21.1
Spectral	10	nearest_neighbors	70.0	20.4
Spectral	11	nearest_neighbors	68.3	20.4
Spectral	12	nearest_neighbors	64.9	19.4
Spectral	13	nearest_neighbors	65.5	19.8
Spectral	14	nearest_neighbors	64.2	19.8
Spectral	15	nearest_neighbors	65.9	21.6
Spectral	16	nearest_neighbors	65.0	20.7
Spectral	17	nearest_neighbors	63.3	20.9
Spectral	18	nearest_neighbors	63.5	21.2
Spectral	19	nearest_neighbors	60.0	20.5

Tabla 8: Resultados spectral mayo.

Seguidamente, se muestran las representaciones gráficas de estos resultados para el mes de mayo, facilitando la observación de las tendencias descritas:

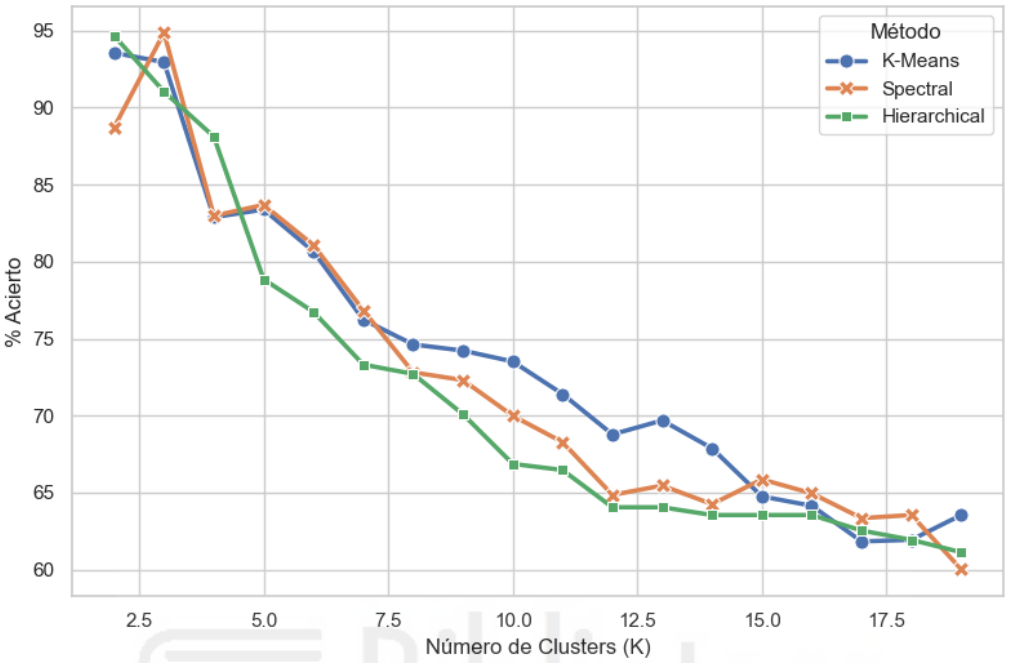

Figura 28: Gráfico de resultados del porcentaje de casos en el que el clúster asignado es correcto en el mes de mayo de los métodos de clustering k-means, hierarchical y spectral.

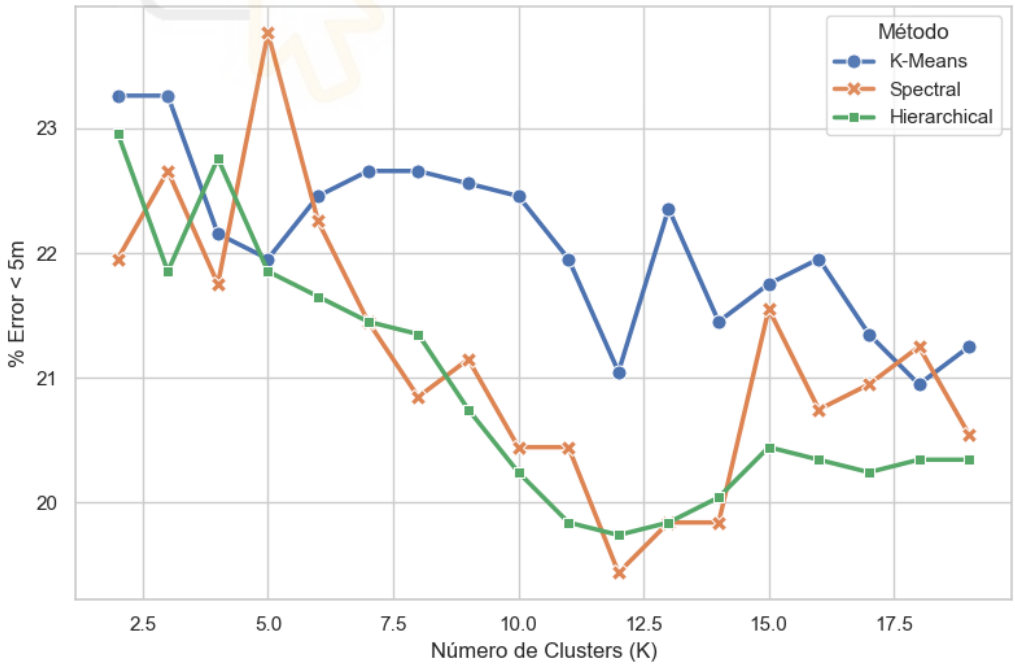

Figura 29: Gráfico de resultados del porcentaje de casos en el que el error de localización es menor a 5m en el mes de mayo de los métodos de clustering k-means, hierarchical y spectral.

En este nuevo escenario, se observa una bajada general del rendimiento en comparación con el invierno (febrero). A pesar de ello, se mantiene la misma tendencia, ya que los resultados empeoran a medida que aumenta la complejidad del modelo.

En lo referente al porcentaje de casos en los que el clúster asignado es correcto, se mantiene la correlación inversa entre el parámetro *k* y la tasa de acierto, tal y como se refleja en la Figura 28. Independientemente del método, los valores máximos se obtienen cuando menor es el número de clústeres, aunque partiendo de umbrales inferiores a los de febrero. En este punto inicial, el algoritmo *hierarchical clustering* alcanza un 94.7% (Tabla 7), seguido muy de cerca por *k-means* con un 93.6% (Tabla 6). Por el contrario, *spectral clustering* acusa la complejidad del entorno comenzando con un valor notablemente más bajo, del 88.7% (Tabla 8). A medida que el parámetro *k* aumenta, el porcentaje de acierto disminuye progresivamente en todos los escenarios.

Por su parte, la métrica de precisión de localización muestra el impacto más acusado de la estacionalidad (Figura 29). Ningún método logra superar la barrera del 24% y su evolución difiere notablemente:

- *K-means* muestra su mejor desempeño en la configuración inicial (*k*=2), alcanzando un 23.3% (Tabla 6). Sin embargo, pierde consistencia rápidamente en los tramos posteriores, donde comienza a presentar un comportamiento oscilante con subidas y bajadas constantes. En términos de magnitud absoluta, este método registró un error medio de localización que osciló entre 155.40 y 167.93 metros, con una mediana situada en el rango de 113.21 a 128.67 metros.
- *Hierarchical clustering* presenta una dinámica similar en el inicio, con un máximo del 23.0% (Tabla 7), pero sufre fluctuaciones bruscas en los primeros tramos antes de mantener una tendencia general decreciente que se acentúa en la parte central de la gráfica. Respecto a sus valores absolutos, el error de localización medio se mantuvo en un intervalo de 155.95 a 172.06 metros, mientras que la mediana varió entre 113.50 y 142.13 metros.

Por último, el algoritmo *spectral clustering* replica el patrón singular observado en invierno, siendo nuevamente el único capaz de revertir la tendencia inicial negativa. Tras comenzar con un 22.0%, el método evoluciona positivamente hasta alcanzar su máximo rendimiento absoluto en la configuración de *k*=5, donde obtiene un 23.8% de precisión (Tabla 8). En términos de magnitudes absolutas, este método obtuvo un error de localización medio que osciló entre 156.52 y 170.43 metros, mientras que la mediana varió entre 113.21 y 133.71 metros.

Completado el análisis de las técnicas particionales y jerárquicas, el estudio se centra ahora en la evaluación de los algoritmos basados en densidad: HDBSCAN y *mean shift*. A diferencia del bloque anterior, estos métodos no requieren la predefinición del número de clústeres, sino que el agrupamiento emerge de la propia concentración de los descriptores. No obstante, presentan diferencias operativas fundamentales: mientras que HDBSCAN incorpora la capacidad explícita de identificar y descartar ruido (puntos no asignables a ninguna estructura densa), *mean shift* opera convergiendo hacia los picos de densidad de la distribución.

Previo a la exposición de los resultados cuantitativos, es preciso detallar la estrategia de parametrización adoptada para controlar la sensibilidad de estos algoritmos.

En el caso de HDBSCAN, el análisis experimental se ha centrado en el barrido del hiperparámetro *min_cluster_size*. Este valor determina el número mínimo de muestras necesarias para que una agrupación de puntos sea considerada un clúster independiente; funcionalmente, este valor determina la granularidad de la segmentación, obligando al algoritmo a condensar o descartar aquellas agrupaciones que no alcanzan la masa crítica de puntos para ser validadas como clústeres.

Por su parte, para *mean shift*, la experimentación se ha focalizado en la variación del *bandwidth* (ancho de banda). Este hiperparámetro regula la granularidad del agrupamiento, es decir, define el radio de influencia bajo el cual los puntos se atraen hacia un centro común. Por tanto, un ancho de banda mayor resultará en un menor número de clústeres, fusionando estructuras cercanas, mientras que un ancho de banda menor fragmentará los datos en múltiples grupos independientes.

Cabe destacar una particularidad en la tabulación de los resultados para este método: se observará una discontinuidad del ancho de banda (*bandwidth*) en la secuencia de valores entre 0.065 y 0.084 (Tabla 10 y Tabla 12). Esta omisión es intencional, dado que las ejecuciones realizadas en ese intervalo intermedio arrojaron resultados idénticos, por lo que se ha optado por excluir estos datos redundantes de las tablas para facilitar la interpretación de los cambios realmente significativos en el comportamiento del algoritmo.

A continuación, se presentan las tablas y gráficos que recogen los resultados obtenidos por ambos métodos basados en densidad en el mes de febrero.

Método	K	Min_cluster_size	Ruido (%)	Acierto (%)	Error <5 (%)
HDBSCAN	2	2	0.2	94.5	34.2
HDBSCAN	2	3	1.7	94.0	34.0
HDBSCAN	12	4	23.2	58.1	18.8
HDBSCAN	9	5	22.5	59.3	19.2
HDBSCAN	5	6	24.4	56.5	17.0
HDBSCAN	4	7	25.1	55.9	16.7
HDBSCAN	2	8	26.4	64.7	19.7
HDBSCAN	2	9	27.5	63.4	19.1
HDBSCAN	2	10	27.0	64.4	19.9
HDBSCAN	2	11	27.5	64.3	19.6
HDBSCAN	2	12	71.5	36.0	8.9
HDBSCAN	2	13	73.9	33.0	7.8
HDBSCAN	2	14	75.8	31.0	7.2

Tabla 9: Resultados HDBSCAN febrero.

Método	K	Bandwidth	Acierto (%)	Error <5 (%)
Mean Shift	12	0.05	75.0	26.3
Mean Shift	12	0.051	76.0	26.8
Mean Shift	11	0.052	75.0	26.1
Mean Shift	9	0.053	77.6	27.6
Mean Shift	7	0.054	70.8	24.2
Mean Shift	8	0.055	74.0	25.4
Mean Shift	6	0.056	78.3	27.0
Mean Shift	6	0.057	78.1	27.0
Mean Shift	5	0.058	94.4	32.4
Mean Shift	5	0.059	94.4	32.4
Mean Shift	5	0.06	94.4	32.4
Mean Shift	5	0.061	94.5	32.4
Mean Shift	4	0.062	95.6	33.1
Mean Shift	4	0.063	95.7	33.0
Mean Shift	3	0.065	93.6	31.9
Mean Shift	2	0.084	96.3	33.0

Tabla 10: Resultados mean shift febrero.

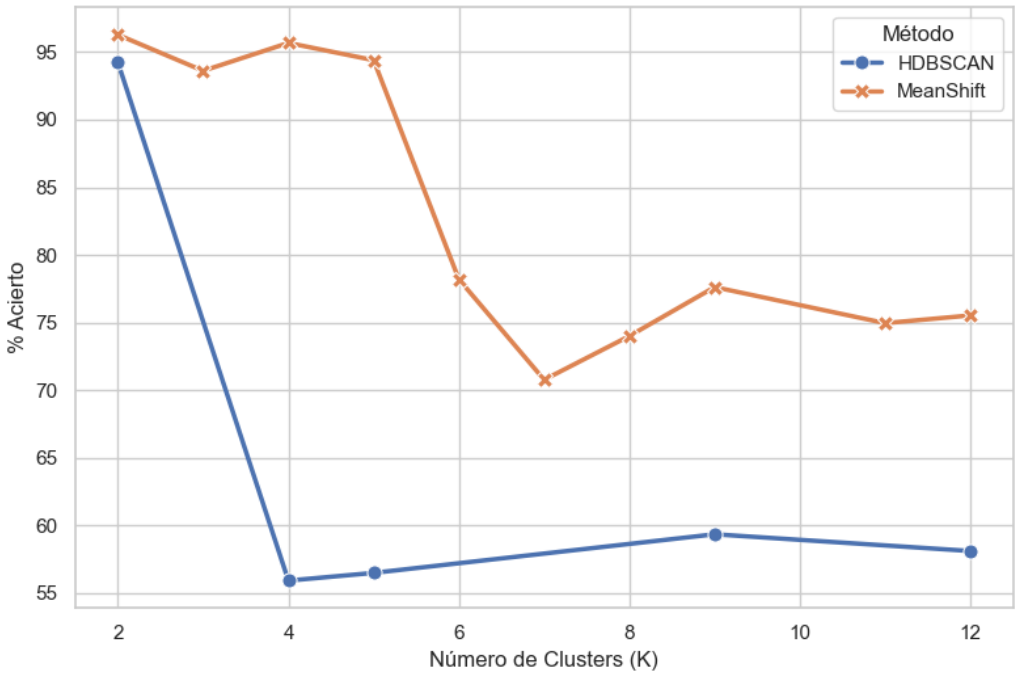

Figura 30: Gráfico de resultados del porcentaje de casos en el que el clúster asignado es correcto en el mes de febrero de los métodos HDBSCAN y mean shift.

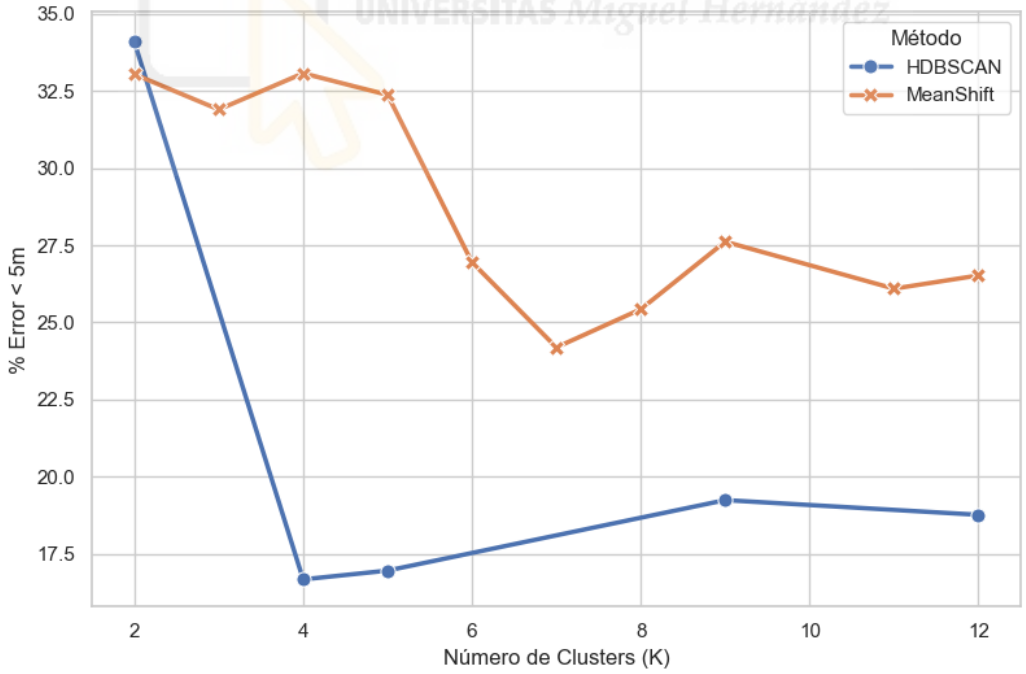

Figura 31: Gráfico de resultados del porcentaje de casos en el que el error de localización es menor a 5m en el mes de febrero de los métodos HDBSCAN y mean shift.

En este escenario de invierno, los métodos basados en densidad mantienen la misma tendencia que en el resto de técnicas, ya que los resultados empeoran a medida que aumenta el número de clústeres.

En lo referente al porcentaje de casos en los que el clúster asignado es correcto, ambos algoritmos alcanzan su máximo rendimiento con dos clústeres (*k*=2). En este punto, *mean shift* obtiene un 96.3% (Tabla 10), y HDBSCAN un 94.5% (Tabla 9).

Sin embargo, al aumentar el número de grupos, cada algoritmo evoluciona de forma distinta:

- HDBSCAN muestra un descenso gradual de la precisión a medida que aumenta el número de clústeres (de 2 a 12). Sin embargo, el factor determinante es el aumento progresivo del ruido al incrementar el parámetro *min_cluster_size*. Esto provoca que, incluso cuando el algoritmo vuelve a identificar una estructura de *k*=2 en configuraciones más exigentes, los resultados sean muy deficientes debido a la excesiva cantidad de datos clasificados como ruido (Tabla 9).
- *Mean shift* replica la tendencia del resto de métodos, ya que, al aumentar el valor de *k*, el porcentaje de acierto baja gradualmente, alejándose de los resultados del principio.

Por su parte, la métrica de precisión de localización muestra patrones de comportamiento diferenciados (Figura 31):

- HDBSCAN presenta un comportamiento más complejo marcado por la gestión del ruido. Aunque las tablas recogen el ciclo completo de experimentación, en la representación gráfica se ha omitido este tramo final para evitar distorsiones visuales y mostrar únicamente el rango operativo real del algoritmo. Analizando este rango efectivo, se observa que se produce el mismo fenómeno que en el resto de métodos. Respecto a sus valores absolutos, el error de localización medio osciló entre 139.12 y 165.65 metros, con una mediana situada en el rango de 33.89 a 145.16 metros.
- *Mean shift* también replica el patrón observado en el resto de métodos, mostrando una disminución progresiva del porcentaje de acierto a medida que aumenta la complejidad. El método parte de valores cercanos al 33.0% y desciende gradualmente hasta el 24.2% en los valores más altos (Tabla 10). En términos de magnitudes absolutas, este método registró un error medio de localización que osciló entre 126.81 y 148.02 metros, con una mediana situada en el rango de 32.45 a 107.48 metros.

A continuación, se exponen los resultados obtenidos por ambos métodos en el mes de mayo:

Método	K	Min_cluster_size	Ruido (%)	Acierto (%)	Error <5 (%)
HDBSCAN	2	2	0.2	92.4	20.9
HDBSCAN	2	3	1.7	91.6	20.6
HDBSCAN	12	4	23.2	49.6	13.3
HDBSCAN	9	5	22.5	50.2	13.3
HDBSCAN	5	6	24.4	49.0	12.2
HDBSCAN	4	7	25.1	48.3	11.8
HDBSCAN	2	8	26.4	58.7	12.3
HDBSCAN	2	9	27.5	56.9	11.7
HDBSCAN	2	10	27.0	57.8	12.3
HDBSCAN	2	11	27.5	57.9	12.1
HDBSCAN	2	12	71.5	30.0	5.5
HDBSCAN	2	13	73.9	27.7	5.4
HDBSCAN	2	14	75.8	25.9	5.3

Tabla 11: Resultados HDBSCAN mayo.

Método	K	Bandwidth	Acierto (%)	Error <5 (%)
Mean Shift	12	0.05	67.4	20.6
Mean Shift	12	0.051	68.7	20.8
Mean Shift	11	0.052	67.3	20.2
Mean Shift	9	0.053	68.3	19.3
Mean Shift	7	0.054	64.2	17.4
Mean Shift	8	0.055	66.5	18.8
Mean Shift	6	0.056	76.2	20.2
Mean Shift	6	0.057	76.2	20.2
Mean Shift	5	0.058	92.5	20.1
Mean Shift	5	0.059	92.2	20.0
Mean Shift	5	0.06	92.5	20.1
Mean Shift	5	0.061	92.6	20.0
Mean Shift	4	0.062	93.7	20.0
Mean Shift	4	0.063	93.7	19.9
Mean Shift	3	0.065	92.6	19.9
Mean Shift	2	0.084	96.5	21.7

Tabla 12: Resultados mean shift mayo.

Para complementar la información de las tablas, se adjuntan los gráficos correspondientes a los resultados obtenidos en el mes de mayo:

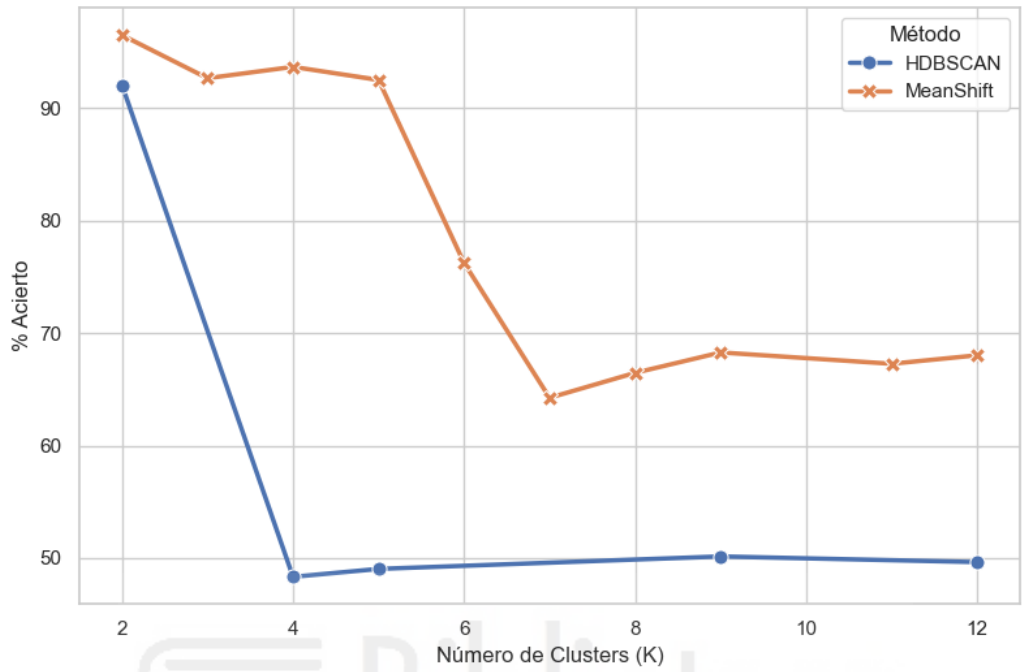

Figura 32: Gráfico de resultados del porcentaje de casos en el que el clúster asignado es correcto en el mes de mayo de los métodos HDBSCAN y mean shift.

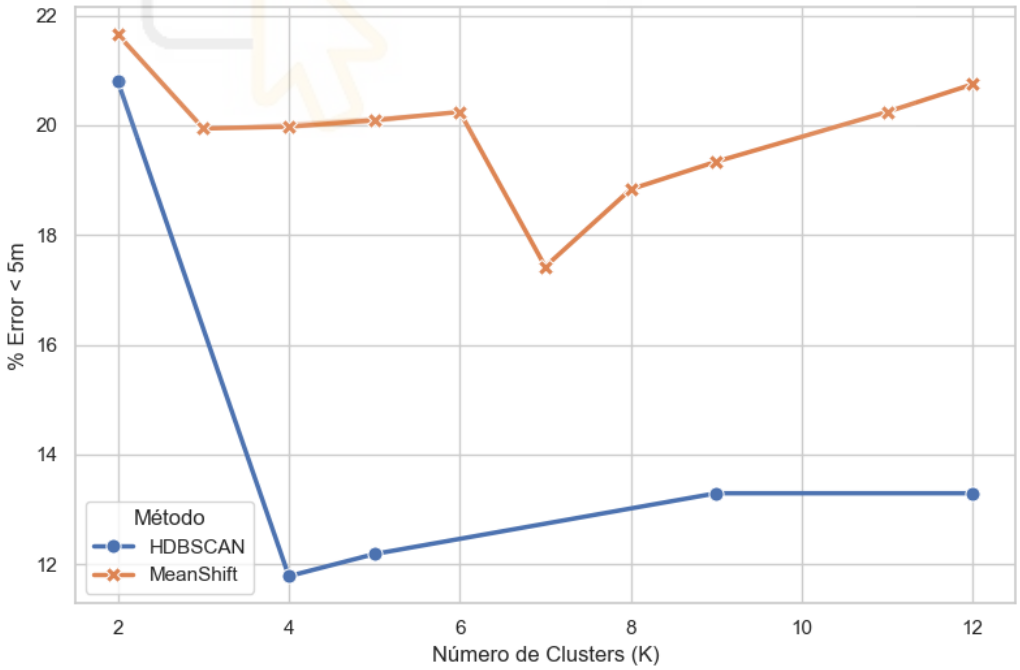

Figura 33: Gráfico de resultados del porcentaje de casos en el que el error de localización es menor a 5m en el mes de mayo de los métodos HDBSCAN y mean shift.

Al analizar los resultados obtenidos en el mes de mayo, se confirman los patrones de comportamiento observados en el resto de los algoritmos.

En cuanto a la tasa de acierto (Figura 32), ambos algoritmos revalidan que los mejores resultados se obtienen cuando menor es el número de clústeres. En este escenario, *mean shift* lidera con un 96.5% de precisión (Tabla 12), superando ligeramente al 92.4% de HDBSCAN (Tabla 11). No obstante, cada algoritmo responde de manera distinta ante el aumento de la fragmentación: mientras *mean shift* sostiene una eficacia superior al 60% incluso con 12 clústeres, HDBSCAN sufre una caída abrupta por debajo del 50% en cuanto se supera la configuración inicial, penalizado por el incremento del ruido.

Por último, la métrica de error de localización (Figura 33) refleja la mayor exigencia del entorno primaveral, con un descenso generalizado de la precisión respecto al invierno. En este rango operativo (k = 2 a 12), *mean shift* muestra gran solidez, manteniendo su porcentaje en torno al 20% en casi todas las iteraciones. Por el contrario, HDBSCAN acusa más la complejidad de la escena: parte de un 20.9% y desciende rápidamente hasta el 11.8% (Tabla 11), demostrando que, en este escenario, es menos tolerante al aumento de clústeres que el resto de las técnicas.

A la vista de estos resultados, *spectral clustering* destaca como la técnica más eficaz. Logra el mejor desempeño en mayo con un 23.8% de casos en los que el error de localización fue menor a 5 metros (Tabla 8), superando al resto de técnicas.

Además, su ventaja no es solo numérica, sino práctica. Mientras que el resto de métodos han logrado sus mejores resultados para *k*=2, *spectral clustering* alcanza su máximo rendimiento con una configuración mucho más razonable de 5 clústeres. Esto confirma su fiabilidad, ya que ofrece las mejores métricas en el mes de mayo sin la necesidad de reducir la complejidad del entorno a una simple clasificación binaria de tan solo dos grupos.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones

El objetivo de este trabajo ha consistido en analizar y comparar distintas técnicas de agrupamiento (*clustering*) para determinar cuál de ellas garantiza una mayor robustez. Tras la evaluación experimental, se concluye que el método *spectral clustering* es la técnica más eficaz, superando en estabilidad y rendimiento a métodos convencionales como *k-means o mean shift* en los dos escenarios evaluados.

Este desempeño superior se fundamenta en la capacidad del algoritmo para explotar la calidad de los espacios latentes generados por las redes neuronales específicas empleadas. En el escenario de interiores, la arquitectura Sonata (entrenada con la base de datos de ScanNet) generó descriptores semánticamente ricos que facilitaron una discriminación precisa entre estancias. Simétricamente, en exteriores, la aplicación de MinkUNeXt sobre las sesiones de NCLT logró sintetizar eficazmente la compleja geometría LiDAR. El *spectral clustering* demostró ser la técnica idónea para interpretar la estructura no lineal de estos descriptores, logrando una agrupación coherente allí donde otros métodos no lograban capturar la complejidad de los datos.

No obstante, el análisis de los resultados en el entorno exterior revela una conclusión crítica respecto a la viabilidad de la navegación autónoma basada exclusivamente en agrupamiento. Dado que los descriptores incluían información de posicionamiento, se evaluó la precisión de la tarea calculando el error métrico respecto a la posición real (*ground truth*). Los datos evidencian que, aunque el *spectral clustering* obtuvo el mejor comportamiento relativo, los errores de localización resultantes no alcanzaron los umbrales mínimos deseados para una navegación de precisión. Esto confirma que, en entornos exteriores dinámicos, el *clustering* actúa eficazmente como un sistema de localización global, pero carece de la exactitud métrica fina necesaria, validando la necesidad de integrar esta solución con las etapas de refinamiento propuestas en el siguiente subapartado.

5.2 Trabajos futuros

En lo que respecta al escenario de interiores, el sistema actual ha demostrado una gran eficacia en la agrupación semántica, logrando identificar y reunir estancias de la misma categoría, como cocinas o baños, basándose en los descriptores generados por Sonata. Esto constituye una solución robusta para la localización global, ya que permite al

agente situarse en un contexto semántico correcto. No obstante, para habilitar una navegación autónoma completa, se propone evolucionar hacia un sistema de localización jerárquica. La estrategia futura consistiría en utilizar la clasificación actual como una primera etapa para restringir el espacio de búsqueda y, posteriormente, activar un módulo de localización fina. Este segundo nivel buscaría coincidencias geométricas específicas dentro de la estancia identificada para calcular la pose exacta con 6 grados de libertad, permitiendo así una transición fluida desde una estimación general del entorno hacia una posición métrica precisa.

Por otro lado, para el escenario de exteriores sobre la base de datos NCLT, y con el objetivo de subsanar la limitación métrica descrita anteriormente, se plantea utilizar la predicción del clúster obtenida no como una posición definitiva, sino como un punto de inicialización para algoritmos específicos de refinamiento de pose. Una línea de investigación prometedora sería la integración con la Localización de Montecarlo (MCL), donde la distribución inicial de las partículas se concentraría en el área del clúster predicho en lugar de dispersarse aleatoriamente, acelerando drásticamente la convergencia hacia la posición real. Asimismo, dado que se dispone de información LiDAR, sería viable complementar este enfoque con algoritmos de registro geométrico como el *Iterative Closest Point* (ICP), los cuales alinearían la nube de puntos local con el mapa global para corregir los errores de traslación y rotación con precisión centimétrica.

6. BIBLIOGRAFÍA

[1] G. N. Desouza and A. C. Kak, "Vision for mobile robot navigation: a survey," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237-267, Feb. 2002, doi: 10.1109/34.982903.

[2] F. H. Edma, J. -H. Park and J. -S. Kim, "Performance Enhanced Risk-Aware MPPI using Gaussian Process for Robust Mobile Robot Control," in *IEEE Access*, doi: 10.1109/ACCESS.2025.3640166.

[3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed. Cambridge, MA: The MIT Press, 2011. Available: https://books.google.es/books?id=4of6AQAAQBAJ&printsec=copyright&hl=es#v=onepage&q&f=false

[4] K. M. T. M. B. Konara, M. A. A. Munasinghe, S. K. Dodampegama and Y. W. R. Amarasinghe, "Design and Analysis of an Autonomously Guided Vehicle to Minimize the Impact of COVID-19," *2020 From Innovation to Impact (FITI)*, Colombo, Sri Lanka, 2020, pp. 1-6, doi: 10.1109/FITI52050.2020.9424884.

[5] H. Hoang, A. Khoa Tran, L. Nhat Thai Tran, M. -H. Le and D. -T. Tran, "A Shortest Smooth-path Motion Planning for a Mobile Robot with Nonholonomic Constraints," *2021 International Conference on System Science and Engineering (ICSSE)*, Ho Chi Minh City, Vietnam, 2021, pp. 145-150, doi: 10.1109/ICSSE52999.2021.9538414.

[6] W. Chen and M. Tomizuka, "Estimation of load side position in indirect drive robots by sensor fusion and kalman filtering," *Proceedings of the 2010 American Control Conference*, Baltimore, MD, USA, 2010, pp. 6852-6857, doi: 10.1109/ACC.2010.5531572.

[7] H. Chang, J. Choi and M. Kim, "Experimental research of probabilistic localization of service robots using range image data and indoor GPS system," *2006 IEEE Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, 2006, pp. 1021-1027, doi: 10.1109/ETFA.2006.355414.

[8] M. Golfarelli, D. Maio and S. Rizzi, "Elastic correction of dead-reckoning errors in map building," *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, Victoria, BC, Canada, 1998, pp. 905-911 vol.2, doi: 10.1109/IROS.1998.727315.

[9] S. E. Hadji, T. H. Hing, M. S. M. Ali, M. A. Khattak and S. Kazi, "2D occupancy grid mapping with inverse range sensor model," *2015 10th Asian Control Conference*

(ASCC), Kota Kinabalu, Malaysia, 2015, pp. 1-6, doi: 10.1109/ASCC.2015.7244705.

[10] C. Cadena *et al*., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," in *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309-1332, Dec. 2016, doi: 10.1109/TRO.2016.2624754.

[11] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

[12] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.

[13] M. Kam, Xiaoxun Zhu and P. Kalata, "Sensor fusion for mobile robot navigation," in *Proceedings of the IEEE*, vol. 85, no. 1, pp. 108-119, Jan. 1997, doi: 10.1109/JPROC.1997.554212.

[14] J. Borenstein and Liqiang Feng, "Measurement and correction of systematic odometry errors in mobile robots," in *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 869-880, Dec. 1996, doi: 10.1109/70.544770.

[15] B. Barshan and H. F. Durrant-Whyte, "Inertial navigation systems for mobile robots," in *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 328-342, June 1995, doi: 10.1109/70.388775.

[16] Z. Dong, Y. Wu and J. Wang, "Kalman Filter and Extended Kalman Filter: Theory, Numerical Applications and Performance Analysis," *2025 5th International Symposium on Computer Technology and Information Science (ISCTIS)*, Xi'an, China, 2025, pp. 532-538, doi: 10.1109/ISCTIS65944.2025.11065026.

[17] A. Carullo and M. Parvis, "An ultrasonic sensor for distance measurement in automotive applications," in *IEEE Sensors Journal*, vol. 1, no. 2, pp. 143-, Aug 2001, doi: 10.1109/JSEN.2001.936931.

[18] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," in *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 213-218, April 1988, doi: 10.1109/56.2085.

[19] G. N. Desouza and A. C. Kak, "Vision for mobile robot navigation: a survey," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237-267, Feb. 2002, doi: 10.1109/34.982903.

[20] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [Tutorial]," in *IEEE Robotics*

& Automation Magazine, vol. 18, no. 4, pp. 80-92, Dec. 2011, doi: 10.1109/MRA.2011.943233.

[21] Shaoshan Liu, "Localization with Real-Time Kinematic Global Navigation Satellite System," in *Engineering Autonomous Vehicles and Robots: The DragonFly Modular-based Approach*, IEEE, 2019, pp.47-75, doi: 10.1002/9781119570516.ch5.

[22] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems," in *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50-61, July 2020, doi: 10.1109/MSP.2020.2973615.

[23] P. Y. Leong and N. S. Ahmad, "LiDAR-Based Obstacle Avoidance With Autonomous Vehicles: A Comprehensive Review," in *IEEE Access*, vol. 12, pp. 164248-164261, 2024, doi: 10.1109/ACCESS.2024.3493238.

[24] J. J. Cabrera et al., “MinkUNeXt: Point Cloud-based Large-scale Place Recognition using 3D Sparse Convolutions,” arXiv e-prints, Art. no. 2403.07593, 2024. Disponible en: https://arxiv.org/pdf/2403.07593

[25] X. Wu et al., “Sonata: Self-Supervised Learning of Reliable Point Representations,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), 2025. Disponible en: https://openaccess.thecvf.com/content/CVPR2025/papers/Wu_Sonata_Self-Supervised_Learning_of_Reliable_Point_Representations_CVPR_2025_paper.pdf

[26] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," in *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210-229, July 1959, doi: 10.1147/rd.33.0210.

[27] R. Sailusha, V. Gnaneswar, R. Ramesh and G. R. Rao, "Credit Card Fraud Detection Using Machine Learning," *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2020, pp. 1264-1270, doi: 10.1109/ICICCS48265.2020.9121114.

[28] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh and K. Shaalan, "Speech Recognition Using Deep Neural Networks: A Systematic Review," in *IEEE Access*, vol. 7, pp. 19143-19165, 2019, doi: 10.1109/ACCESS.2019.2896880.

[29] D. W. Otter, J. R. Medina and J. K. Kalita, "A Survey of the Usages of Deep Learning for Natural Language Processing," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604-624, Feb. 2021, doi: 10.1109/TNNLS.2020.2979670.

[30] T. Kansal, S. Bahuguna, V. Singh and T. Choudhury, "Customer Segmentation using

K-means Clustering," *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, Belgaum, India, 2018, pp. 135-139, doi: 10.1109/CTEMS.2018.8769171.

[31] V. Bhatia, S. Choudhary and K. R. Ramkumar, "A Comparative Study on Various Intrusion Detection Techniques Using Machine Learning and Neural Network," *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, 2020, pp. 232-236, doi: 10.1109/ICRITO48877.2020.9198008.

[32] R. C. Palsaniya, "Predictive Maintenance in Semiconductor Manufacturing - AI Machine Learning Application for Downtime Reduction," *2025 IEEE 34th International Symposium on Industrial Electronics (ISIE)*, Toronto, ON, Canada, 2025, pp. 1-5, doi: 10.1109/ISIE62713.2025.11124753.

[33] H. Nguyen and H. La, "Review of Deep Reinforcement Learning for Robot Manipulation," *2019 Third IEEE International Conference on Robotic Computing (IRC)*, Naples, Italy, 2019, pp. 590-595, doi: 10.1109/IRC.2019.00120.

[34] B. R. Kiran *et al*., "Deep Reinforcement Learning for Autonomous Driving: A Survey," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909-4926, June 2022, doi: 10.1109/TITS.2021.3054625.

[35] N. Justesen, P. Bontrager, J. Togelius and S. Risi, "Deep Learning for Video Game Playing," in *IEEE Transactions on Games*, vol. 12, no. 1, pp. 1-20, March 2020, doi: 10.1109/TG.2019.2896986.

[36] Rui Xu and D. Wunsch, "Survey of clustering algorithms," in *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645-678, May 2005, doi: 10.1109/TNN.2005.845141.

[37] J. Paek and J. Ko, "K-Means Clustering-Based Data Compression Scheme for Wireless Imaging Sensor Networks," in *IEEE Systems Journal*, vol. 11, no. 4, pp. 2652-2662, Dec. 2017, doi: 10.1109/JSYST.2015.2491359.

[38] M. Fang and F. Liu, "A Network Intrusion Detection Method based on K-Means," *CIBDA 2022; 3rd International Conference on Computer Information and Big Data Applications*, Wuhan, China, 2022, pp. 1-5. Available: https://ieeexplore.ieee.org/document/9898933?denied=

[39] T. Xing, Q. Hu, J. Li and G. Wang, "Refined SAR image segmentation algorithm based on K-means clustering," *2016 CIE International Conference on Radar (RADAR)*, Guangzhou, China, 2016, pp. 1-3, doi: 10.1109/RADAR.2016.8059487.

[40] G. D. Asyaev and A. N. Sokolov, "Building a System of Analytics of Information

Security Events in the Automated Process Control System and Incident Analysis Based on Cluster Analysis," *2025 International Russian Smart Industry Conference (SmartIndustryCon)*, Sochi, Russian Federation, 2025, pp. 584-589, doi: 10.1109/SmartIndustryCon65166.2025.10986275.

[41] R. Ibrahim and M. O. Shafiq, "Mining Trajectory Data and Identifying Patterns for Taxi Movement Trips," *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*, Berlin, Germany, 2018, pp. 130-135, doi: 10.1109/ICDIM.2018.8847135.

[42] S. Gare *et al*., "Analytics Pipeline for Visualization of Single Cell RNA Sequencing Data from Brochoaveolar Fluid in COVID-19 Patients: Assessment of Neuro Fuzzy-C-Means and HDBSCAN," *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, Glasgow, Scotland, United Kingdom, 2022, pp. 1634-1637, doi: 10.1109/EMBC48229.2022.9871686.

[43] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, May 2002, doi: 10.1109/34.1000236.

[44] D. Comaniciu, V. Ramesh and P. Meer, "Kernel-based object tracking," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564-577, May 2003, doi: 10.1109/TPAMI.2003.1195991.

[45] Z. Liu, X. Bai, C. Sun, F. Zhou and Y. Li, "Multi-Modal Ship Target Image Smoothing Based on Adaptive Mean Shift," in *IEEE Access*, vol. 6, pp. 12573-12586, 2018, doi: 10.1109/ACCESS.2018.2794141.

[46] T. He, C. Han, R. Isawa, T. Takahashi, S. Kijima and J. Takeuchi, "Scalable and Fast Algorithm for Constructing Phylogenetic Trees With Application to IoT Malware Clustering," in *IEEE Access*, vol. 11, pp. 8240-8253, 2023, doi: 10.1109/ACCESS.2023.3238711.

[47] G. Chicco, R. Napoli and F. Piglione, "Comparisons among clustering techniques for electricity customer classification," in *IEEE Transactions on Power Systems*, vol. 21, no. 2, pp. 933-940, May 2006, doi: 10.1109/TPWRS.2006.873122.

[48] O. Maqbool and H. Babri, "Hierarchical Clustering for Software Architecture Recovery," in *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759-780, Nov. 2007, doi: 10.1109/TSE.2007.70732.

[49] Jianbo Shi and J. Malik, "Normalized cuts and image segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905,

Aug. 2000, doi: 10.1109/34.868688.

[50] H. Van Lierde, T. W. S. Chow and G. Chen, "Scalable Spectral Clustering for Overlapping Community Detection in Large-Scale Networks," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 4, pp. 754-767, 1 April 2020, doi: 10.1109/TKDE.2019.2892096.

[51] T. Cheng *et al*., "Online Neural Speaker Diarization with Spectral Clustering for Meeting Scenarios," *2024 IEEE 14th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, Beijing, China, 2024, pp. 373-377, doi: 10.1109/ISCSLP63861.2024.10800669.

[52] T. Wang, B. Li and S. Nabavi, "Single-cell RNA sequencing data clustering using graph convolutional networks," *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Houston, TX, USA, 2021, pp. 2163-2170, doi: 10.1109/BIBM52615.2021.9669529.

[53] K. Taşdemir, Y. Moazzen and I. Yildirim, "An Approximate Spectral Clustering Ensemble for High Spatial Resolution Remote-Sensing Images," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 5, pp. 1996-2004, May 2015, doi: 10.1109/JSTARS.2015.2424292.

[54] D. R, K. R, P. G, P. S and S. R. A K, "Breast Cancer Classification using the Supervised Learning Algorithms," *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2021, pp. 1492-1498, doi: 10.1109/ICICCS51141.2021.9432293.

[55] L. -H. Li, R. Ahmad, W. -C. Tsai and A. K. Sharma, "A Feature Selection Based DNN for Intrusion Detection System," *2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, Seoul, Korea (South), 2021, pp. 1-8, doi: 10.1109/IMCOM51814.2021.9377405.

[56] S. A. K. Mohammed, A. H. A. Rahman, M. A. Bakar and M. Z. A. Razak, "An Efficient Intersection Over Union Algorithm with Angle Orientation for an Improved 3D Object Detection," *2024 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, Kota Kinabalu, Malaysia, 2024, pp. 312-316, doi: 10.1109/IICAIET62352.2024.10730667.

[57] X. Wu, N. Hu and P. Wang, "CSA-UNet: An Efficient Context Separable Attention UNet for Medical Image Segmentation," *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Lisbon, Portugal, 2024, pp. 2688-2693, doi: 10.1109/BIBM62325.2024.10822831.

[58] T. Li, Y. Ma and T. Endoh, "Normalization-Based Validity Index of Adaptive K-Means Clustering for Multi-Solution Application," in *IEEE Access*, vol. 8, pp. 9403-9419, 2020, doi: 10.1109/ACCESS.2020.2964763.

[59] D. Marutho, S. Hendra Handaka, E. Wijaya and Muljono, "The Determination of Cluster Number at k-Mean Using Elbow Method and Purity Evaluation on Headline News," *2018 International Seminar on Application for Technology of Information and Communication*, Semarang, Indonesia, 2018, pp. 533-538, doi: 10.1109/ISEMANTIC.2018.8549751.

[60] H. B. Tambunan, D. H. Barus, J. Hartono, A. S. Alam, D. A. Nugraha and H. H. H. Usman, "Electrical Peak Load Clustering Analysis Using K-Means Algorithm and Silhouette Coefficient," *2020 International Conference on Technology and Policy in Energy and Electric Power (ICT-PEP)*, Bandung, Indonesia, 2020, pp. 258-262, doi: 10.1109/ICT-PEP50916.2020.9249773.

[61] M. Aryuni, E. Didik Madyatmadja and E. Miranda, "Customer Segmentation in XYZ Bank Using K-Means and K-Medoids Clustering," *2018 International Conference on Information Management and Technology (ICIMTech)*, Jakarta, Indonesia, 2018, pp. 412-416, doi: 10.1109/ICIMTech.2018.8528086.

[62] https://www.python.org/

[63] https://scikit-learn.org/stable/

[64] https://code.visualstudio.com/

[65] http://www.scan-net.org/

[66] https://robots.engin.umich.edu/nclt/index.html#top

7. ANEXO 1: CÓDIGOS PARA EL *CLUSTERING* EN INTERIORES

```python
# Librerías
from tools import read_descriptors
from sklearn.preprocessing import Normalizer
from sklearn.cluster import HDBSCAN
from sklearn.metrics import adjusted_rand_score, silhouette_score
import numpy as np

# Cargar y normalizar datos
ruta = r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering interiores\Python\Parámetros_Óptimos\scannet_scene_descriptors_v3.csv"
descriptors, scenes_names, scenes_types = read_descriptors(ruta)
normal = Normalizer(norm = 'l2')
descriptors_norm = normal.fit_transform(descriptors)

# Rangos de los hiperparámetros a probar
cluster_sizes = np.arange(1, 15, 1)
min_samples_list = np.arange(1, 10, 1)

print("\n")
print(f"{'SIZES':<6} | {'SAMPLES':<7} | {'k':<5} | {'% RUIDO':<10} | {'ARI':<10} | {'SILUETA':<10}")
print("-" * 75)

# Lista para guardar los resultados
resultados = []

# Bucle de Optimización
for m_size in cluster_sizes:
    for m_samp in min_samples_list:
            try:
                hdb = HDBSCAN(min_cluster_size = m_size, min_samples = m_samp)
                labels = hdb.fit_predict(descriptors_norm)
                # Contar clusters (quitando el -1)
                unique_labels = set(labels)
                n_clusters = len(unique_labels) - (1 if -1 in labels else 0)
                # Calcular porcentaje de ruido
                perc_noise = list(labels).count(-1) / len(labels) * 100
                # Guardar solo si hay clusters reales
                if n_clusters > 1:
                    ari = adjusted_rand_score(scenes_types, labels)
                    sil = silhouette_score(descriptors_norm, labels)
                    print(f"{m_size:<6} | {m_samp:<7} | {n_clusters:<5} | {perc_noise:<9.1f}% | {ari:<10.4f} | {sil:<10.4f}")
                    resultados.append({
                        "min_cluster_size": m_size,
                        "min_samples": m_samp,
                        "k": n_clusters,
                        "ruido": perc_noise,
                        "ari": ari,
                        "sil": sil
                    })
            except Exception as e:
                pass
```

Figura 34: Código para obtener 11 clústeres en HDBSCAN.

```python
#Librerías
from tools import read_descriptors
from sklearn.preprocessing import Normalizer
from sklearn.metrics import adjusted_rand_score, silhouette_score
from sklearn.cluster import MeanShift
import numpy as np

# Cargar y normalizar datos
ruta = r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering interiores\Python\Parámetros_Óptimos\scannet_scene_descriptors_v3.csv"
descriptors, scenes_names, scenes_types = read_descriptors(ruta)
normal = Normalizer(norm = 'l2')
descriptors_normal = normal.fit_transform(descriptors)

# Rangos del hiperparámetro a probar
bandwidths = np.arange(0.0750, 0.1, 0.002)

print(f"{'BW':<10} | {'k':<5} | {'ARI':<10} | {'SILUETA':<10}")
print("-" * 45)

resultados = []

# Bucle de optimización
for bw in bandwidths:
    try:
        ms = MeanShift(bandwidth = bw, bin_seeding = False)
        ms.fit(descriptors_normal)
        n_clusters = len(ms.cluster_centers_)
        labels = ms.labels_
        if 1 < n_clusters < len(descriptors_normal):
            ari = adjusted_rand_score(scenes_types, labels)
            sil = silhouette_score(descriptors_normal, labels)
            print(f"{bw:<10.4f} | {n_clusters:<5} | {ari:<10.4f} | {sil:<10.4f}")
            resultados.append({
                'bandwidth': bw,
                'k_clusters': n_clusters,
                'ARI': ari,
                'Silhouette': sil
            })
        elif n_clusters == 1:
            print(f"{bw:<10.4f} | 1     | (Descartado: solo 1 grupo)")
        else:
            print(f"{bw:<10.4f} | {n_clusters:<5} | (Descartado: ruido)")
    except Exception as e:
        print(f"{bw:<10.4f} | ERROR | {e}")
```

Figura 35: Código para obtener 11 clústeres del método mean shift.

```python
#Librerias
from tools import read_descriptors
from sklearn.preprocessing import Normalizer
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score, silhouette_score
import pandas as pd

# Cargar y normaliza datos
ruta = r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering interiores\Python\Parámetros_Óptimos\scannet_scene_descriptors_v3.csv"
descriptors, scenes_names, scenes_types = read_descriptors(ruta)
normal = Normalizer(norm = 'l2')
descriptors_normal = normal.fit_transform(descriptors)

# Parámetros a evaluar
linkages = ["ward", "complete", "average", "single"]
metrics = ["euclidean", "manhattan", "cosine"]

# Lista para guardar los resultados
resultados = []

print("\nPROBANDO TODAS LAS COMBINACIONES PARA k = 11")
print("=============================================")

# Bucle de optimización
for link in linkages:
    for met in metrics:
        if link == "ward" and met != "euclidean":
            continue
        try:
            hierarchical = AgglomerativeClustering(n_clusters = 11, metric = met, linkage = link)
            labels = hierarchical.fit_predict(descriptors_normal)
            ari = adjusted_rand_score(scenes_types, labels)
            sil = silhouette_score(descriptors_normal, labels, metric = met)
            resultados.append({
                "Linkage": link,
                "Metric": met,
                "ARI": ari,
                "Silhouette": sil
            })
            print(f"{link:<12} | {met:<12} | {ari:.4f} | {sil:.4f}")

        except Exception as e:
            continue

# Resultados Finales
if not resultados:
    print("\n ERROR: No se generó ningún cluster válido.")
else:
    df = pd.DataFrame(resultados)
    # Mejor ARI
    best_ari = df.sort_values(by="ARI", ascending = False).iloc[0]
    # Mejor Silueta
    best_sil= df.sort_values(by="Silhouette", ascending = False).iloc[0]

    print("\n" + "="*30)
    print("MEJOR CONFIGURACIÓN ENCONTRADA")
    print("="*30)

    print(f"\n• SEGÚN ARI:")
    print(f"Linkage:    {best_ari['Linkage']}")
    print(f"Métrica:    {best_ari['Metric']}")
    print(f"ARI:        {best_ari['ARI']:.4f}")
    print(f"Silhouette: {best_ari['Silhouette']:.4f}")

    print(f"\n• SEGÚN SILUETA:")
    print(f"Linkage:    {best_sil['Linkage']}")
    print(f"Métrica:    {best_sil['Metric']}")
    print(f"ARI:        {best_sil['ARI']:.4f}")
    print(f"Silhouette: {best_sil['Silhouette']:.4f}")
```

Figura 36: Código para obtener la configuración de hiperparámetros del método hierarchical que maximiza las métricas externas.

```python
# Librerias
from tools import read_descriptors
from sklearn.preprocessing import Normalizer
from sklearn.cluster import SpectralClustering
from sklearn.metrics import adjusted_rand_score, silhouette_score
import numpy as np
import pandas as pd

# Cargar y normalizar datos
ruta = r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering interiores\Python\Parámetros_Óptimos\scannet_scene_descriptors_v3.csv"
descriptors, scenes_names, scenes_types = read_descriptors(ruta)
normal = Normalizer(norm = 'l2')
descriptors_normal = normal.fit_transform(descriptors)

# Lista para guardar los resultados
resultados = []

# Parámetros a evaluar
vecinos = np.arange(45, 60, 1)
gammas = np.arange(1,30,1)
afinidades = ["nearest_neighbors", "rbf"]
assign_labels_list = ["kmeans", "discretize"]

# Bucle de optimización
print("\n     AFINIDAD      | ASIGNACIÓN  | PARÁMETRO  |  ARI   |  SIL")
print("="*70)
for affinity in afinidades:
    for strategy in assign_labels_list:
        if affinity == "nearest_neighbors":
            lista_params = vecinos
        else:
            lista_params = gammas

        for param in lista_params:
            try:
                if affinity == "nearest_neighbors":
                    sc = SpectralClustering(
                        n_clusters = 11,
                        affinity="nearest_neighbors",
                        n_neighbors = param,
                        assign_labels = strategy,
                        random_state = 42,
                    )
                    param_str = f"NN={param}"

                else:
                    sc = SpectralClustering(
                        n_clusters = 13,
                        affinity = "rbf",
                        gamma = param,
                        assign_labels = strategy,
                        random_state = 42,
                    )
                    param_str = f"Gamma={param}"
                labels = sc.fit_predict(descriptors_normal)
                if len(set(labels)) > 1:
                    ari = adjusted_rand_score(scenes_types, labels)
                    sil = silhouette_score(descriptors_normal, labels)
                    resultados.append({
                        'Afinidad': affinity,
                        'Estrategia': strategy,
                        'Parametro': param_str,
                        'ARI': ari,
                        'Silhouette': sil
                    })
                    print(f"{affinity:<18} | {strategy:<12} | {param_str:<10} | {ari:.4f} | {sil:.4f}")

            except Exception as e:
                continue

# Resultados Finales
if not resultados:
    print("\n ERROR: No se generó ningún cluster válido.")
else:
    df = pd.DataFrame(resultados)
    # Mejor ARI
    best_ari = df.sort_values(by = 'ARI', ascending = False).iloc[0]
    # Mejor Silueta
    best_sil = df.sort_values(by = 'Silhouette', ascending = False).iloc[0]

    print("\n" + "="*60)
    print("MEJOR CONFIGURACIÓN ENCONTRADA")
    print("="*60)

    print(f"\n• SEGÚN ARI:")
    print(f"Afinidad:   {best_ari['Afinidad']}")
    print(f"Estrategia: {best_ari['Estrategia']}")
    print(f"Parámetro:  {best_ari['Parametro']}")
    print(f"ARI:        {best_ari['ARI']:.4f}")
    print(f"Silhouette: {best_ari['Silhouette']:.4f}")

    print(f"\n• SEGÚN SILUETA:")
    print("="*50)
    print(f"Afinidad:   {best_sil['Afinidad']}")
    print(f"Estrategia: {best_sil['Estrategia']}")
    print(f"Parámetro:  {best_sil['Parametro']}")
    print(f"ARI:        {best_sil['ARI']:.4f}")
    print(f"Silhouette: {best_sil['Silhouette']:.4f}")
```

Figura 37: Código para obtener la configuración de hiperparámetros del método spectral que maximiza las métricas externas.

```python
# Librerías
from tools import read_descriptors
from sklearn.preprocessing import Normalizer, LabelEncoder
from sklearn.cluster import KMeans, HDBSCAN, MeanShift, AgglomerativeClustering, SpectralClustering
from sklearn.metrics import (
    adjusted_rand_score,
    silhouette_score,
    davies_bouldin_score,
    recall_score,
    jaccard_score
)
import pandas as pd
import numpy as np
from scipy.stats import mode
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Cargar normalizar datos
ruta = r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering interiores\Python\Evaluación_Métodos\scannet_scene_descriptors_v3.csv"
descriptors, scenes_names, scenes_type = read_descriptors(ruta)
normal = Normalizer(norm ="l2")
descriptors_normal = normal.fit_transform(descriptors)

# Convertir etiquetas de texto a números
le = LabelEncoder()
scenes_type_num = le.fit_transform(scenes_type)

# Definir y ejecutar métodos de clustering
methods = {
    "K-Means": KMeans(n_clusters = 11, random_state = 42, n_init = 10),
    "HDBSCAN": HDBSCAN(min_cluster_size = 2, min_samples = 1, cluster_selection_epsilon = 0.0),
    "MeanShift": MeanShift(bandwidth = 0.077),
    "Hierarchical": AgglomerativeClustering(n_clusters = 11, linkage = "ward"),
    "Spectral": SpectralClustering(n_clusters = 11, affinity = "rbf", gamma = 24, random_state = 42, assign_labels = "discretize")
}
resultados = []
# Bucle para ejecutar los modelos y rellenar el diccionario
for name, model in methods.items():
    try:
        labels = model.fit_predict(descriptors_normal)
        # Gestión de ruido (para HDBSCAN)
        if -1 in labels:
            n_clusters_detected = len(set(labels)) - 1
        else:
            n_clusters_detected = len(set(labels))
        # Cálculo de métricas
        ari = adjusted_rand_score(scenes_type, labels)
        sil = np.nan
        db = np.nan
        if n_clusters_detected > 1 and n_clusters_detected < len(descriptors_normal):
            sil = silhouette_score(descriptors_normal, labels)
            db = davies_bouldin_score(descriptors_normal, labels)
        # Asignación por mayoría
        new_labels = np.zeros_like(labels)
        unique_clusters = np.unique(labels)
        for cluster_id in unique_clusters:
            if cluster_id == -1:
                new_labels[labels == -1] = -1
                continue
            mask = (labels == cluster_id)
            true_labels = scenes_type_num[mask]
            if len(true_labels) > 0:
                label_most_common = mode(true_labels, keepdims = True)[0][0]
                new_labels[mask] = label_most_common
        # Calculo recall e iou después de la asignación
        recall = recall_score(scenes_type_num , new_labels, average = "macro", zero_division = 0)
        iou = jaccard_score(scenes_type_num , new_labels, average = "macro", zero_division = 0)
        # Guardar resultados
        resultados.append({
            "Método": name,
            "Recall" : recall,
            "ARI" : ari,
            "IOU" : iou,
            "Silhouette": sil,
            "Davies-Bouldin": db,
            "Clusters": n_clusters_detected})
    except Exception as e:
        print(f" Error en {name}: {e}")
        resultados.append({
            "Método": name,
            "Datos": "Error",
            "ARI": np.nan,
            "Silhouette": np.nan, "Davies-Bouldin": np.nan, "Recall": np.nan, "IoU": np.nan,
            "Clusters": 0
        })
```

```python
# Mostrar tabla final
pd.set_option("display.float_format", "{:.3f}".format)
df_results = pd.DataFrame(resultados)
df_results = df_results.sort_values(by = "Recall", ascending = False)
print("\n                    RESULTADOS DE EVALUACIÓN DE CLUSTERING")
print("="*80)
# Usamos .to_string() para asegurarnos de que se imprima la tabla completa
print(df_results.to_string(index = False))
print("="*80)
# Interpretación de las métricas de evaluación
print("\nGUÍA DE INTERPRETACIÓN:")
print("  • Externas (ARI, Recall, IOU): MÁS ALTO = MEJOR (comparan con la etiquetas reales")
print("  • Internas (Silhouette: MÁS ALTO = MEJOR (mide separación / densidad)")
print("  • Internas (Davies-Bouldin): MÁS BAJO = MEJOR (mide similitud inter-cluster)")

# Generación de Matrices de Confusión
# Recuperar los nombres de las clases
nombres_clases = le.classes_
for name, model in methods.items():
    try:
        # Volver a predecir para graficar
        labels = model.fit_predict(descriptors_normal)

        # Asignamos etiquetas mediante la mayoría
        labels_aligned = np.zeros_like(labels)
        unique_clusters = np.unique(labels)
        for cluster_id in unique_clusters:
            if cluster_id == -1:
                labels_aligned[labels == -1] = -1
                continue
            mask = (labels == cluster_id)
            true_labels_in_cluster = scenes_type_num[mask]
            if len(true_labels_in_cluster) > 0:
                most_common = mode(true_labels_in_cluster, keepdims = True)[0][0]
                labels_aligned[mask] = most_common

        # Calcular la Matriz de Confusión
        matriz = confusion_matrix(scenes_type_num, labels_aligned, labels = np.unique(scenes_type_num))

        # Configurar el gráfico
        plt.figure(figsize = (12, 10))
        sns.heatmap(matriz, annot = True, fmt = "d", cmap = "YlGnBu", cbar = False, xticklabels = nombres_clases, yticklabels = nombres_clases)
        plt.title(f"Matriz de Confusión - {name}", fontsize = 16)
        plt.xlabel("Predicción del Algoritmo (Cluster asignado)", fontsize = 12)
        plt.ylabel("Clase Real (Ground Truth)", fontsize = 12)
        plt.xticks(rotation = 45, ha = "right")
        plt.yticks(rotation = 0)
        plt.tight_layout()
        plt.show()
    except Exception as e:
        print(f"No se pudo graficar {name}: {e}")
```

Figura 38: Código para evaluar el clustering en interiores.

8. ANEXO 2: CÓDIGO PARA EL *CLUSTERING* EN EXTERIORES

```python
# BIBLIOTECAS
import pandas as pd
import numpy as np
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans, HDBSCAN, MeanShift, AgglomerativeClustering, SpectralClustering
from scipy.spatial.distance import cdist
import itertools
import matplotlib.pyplot as plt
import seaborn as sns

# CONFIGURACIÓN Y CARGA DE DATOS
d_e = pd.read_csv(r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering exteriores\Evaluación métodos\nclt_descriptors_2012-01-08.csv")
d_f = pd.read_csv(r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering exteriores\Evaluación métodos\nclt_descriptors_2012-02-02_reduced.csv")
d_m = pd.read_csv(r"C:\Users\danic\OneDrive\Escritorio\TFG\Clustering exteriores\Evaluación métodos\nclt_descriptors_2012-05-11_reduced.csv")

# Convertir string a array
def conv_descriptor(desc_str):
    clean_str = desc_str.replace('[', '').replace(']', '').replace('\n', ' ')
    return np.fromstring(clean_str, sep = ' ')

# Obtener vectores
desc_ene = np.stack(d_e["descriptor"].apply(conv_descriptor).tolist())
desc_feb = np.stack(d_f["descriptor"].apply(conv_descriptor).tolist())
desc_may = np.stack(d_m["descriptor"].apply(conv_descriptor).tolist())

# Realizar normalización
normalizer = Normalizer(norm = 'l2')
desc_ene_norm = normalizer.fit_transform(desc_ene)
desc_feb_norm = normalizer.transform(desc_feb)
desc_may_norm = normalizer.transform(desc_may)

# Obtener posiciones
pos_enero = d_e[['pos_x', 'pos_y']].values
pos_feb = d_f[['pos_x', 'pos_y']].values
pos_may = d_m[['pos_x', 'pos_y']].values
# FUNCIÓN DE EVALUACIÓN
def evaluar_configuracion(desc_train, pos_train, labels_train, desc_test, pos_test):
    # Evitar errores de ruido
    unique_labels = set(labels_train)
    if -1 in unique_labels: unique_labels.remove(-1)
    if len(unique_labels) < 2:
        return np.nan, np.nan, np.nan

    #  Construir mapa topológico (Centroides)
    df_comb = pd.DataFrame(desc_train)
    df_comb['cluster'] = labels_train
    # Calcular los centroides ignorando el ruido
    centroides = df_comb[df_comb['cluster'] != -1].groupby('cluster').mean()
    centroides_desc = centroides.values
    etiquetas_reales = centroides.index.values

    # Asignar a cada scan el clúster más cercano
    dists_test = cdist(desc_test, centroides_desc, metric = 'euclidean')
    indices_cluster_asignados = np.argmin(dists_test, axis = 1)

    # Comparar el scan de test(Febrero) con los descriptores del cluster seleccionado
    pos_estimada = []
    aciertos_cluster = 0

    for i, idx_centroide in enumerate(indices_cluster_asignados):
        label_predicha = etiquetas_reales[idx_centroide]

        # Filtar los descriptores  y posiciones del cluster asignado
        mask = (labels_train == label_predicha)
        desc_cluster = desc_train[mask]
        pos_cluster = pos_train[mask]

        # Asignar la posición al scan de test del descriptor más parecido
        dists = cdist([desc_test[i]], desc_cluster, metric = 'euclidean')
        ind_vecino = np.argmin(dists)
        pos_estimada.append(pos_cluster[ind_vecino])
        # Carcular la distancia real del robot a todos los puntos de entrenamiento
        dist_gt = cdist(pos_test[i].reshape(1, -1), pos_train, metric = 'euclidean')
        # Índices de los puntos donde el error de localización es menor a 5m
        ind_cercanos_gt = np.where(dist_gt[0] < 5)[0]

        if len(ind_cercanos_gt) > 0:
            labels_cercanos = labels_train[ind_cercanos_gt]
            if label_predicha in labels_cercanos:
                aciertos_cluster += 1
    pos_estimada = np.array(pos_estimada)
    # Calculo de métricas
    errores_loc = np.linalg.norm(pos_estimada - pos_test, axis=1)
    # Error Medio
    error_medio = np.mean(errores_loc)
    # Error Mediana
    error_mediana = np.median(errores_loc)
    # Porcentaje Acierto Cluster
    pct_acierto_cluster = (aciertos_cluster / len(errores_loc)) * 100
    # 4. Porcentaje < 5m
    pct_menor_5m = (len(errores_loc[errores_loc < 5]) / len(errores_loc)) * 100
    # Retornamos las 4 variables
    return pct_acierto_cluster, pct_menor_5m, error_medio, error_mediana
```

```python
# DEFINICIÓN DEL GRID DE PARÁMETROS
param_grids = {
    "K-Means": {
        "n_clusters": np.arange(2, 20, 1),
        "n_init": [10]
    },

    "HDBSCAN": {
       "min_cluster_size": np.arange(2, 15, 1),
    },
     "MeanShift": {
        "bandwidth": np.arange(0.05, 0.086, 0.001)
    },
    "Hierarchical": {
        "n_clusters": np.arange(2, 20, 1),
        "linkage": ["ward"]
    },
    "Spectral": {
        "n_clusters": np.arange(2, 20, 1),
        "affinity": ["nearest_neighbors"]
    }
}

datos_para_exportar = []

print("\n" + "="*185)
print("                                                                RESULTADOS OBTENIDOS ")
print("="*185)

# Imprimimos la cabecera de la tabla una sola vez al principio
header_format = "{:<13} | {:<40} | {:^4} | {:^9} | {:<8} | {:<8} | {:<12} | {:<14} | {:<8} | {:<8} | {:<12} | {:<14}"
print(header_format.format("MÉTODO", "CONFIGURACIÓN", "K", "RUIDO %", "F:%OK", "F:<5m", "F:ERR_MEDIA", "F:ERR_MEDIANA", "M:%OK", "M:<5m", "M:ERR", "M:ERR_MEDIANA"))
print("-" * 185)

for metodo, params in param_grids.items():
    keys, values = zip(*params.items())
    combinaciones = [dict(zip(keys, v)) for v in itertools.product(*values)]
    mejor_config_metodo = None
    for i, config in enumerate(combinaciones):
        try:
            # Entrenar
            if metodo == "K-Means": model = KMeans(random_state = 42, **config)
            elif metodo == "MeanShift": model = MeanShift(**config, n_jobs = -1)
            elif metodo == "Hierarchical": model = AgglomerativeClustering(**config)
            elif metodo == "Spectral": model = SpectralClustering(random_state = 42, n_jobs = -1, **config)
            elif metodo == "HDBSCAN": model = HDBSCAN(**config)
            labels = model.fit_predict(desc_ene_norm)

            unique_labels_set = set(labels)
            n_clusters_reales = len(set(labels)) - (1 if -1 in set(labels) else 0)

            # Cálculo de Ruido en el caso de HDBSCAN
            n_ruido = np.sum(labels == -1)
            pct_ruido = (n_ruido / len(labels)) * 100

            # MOSTRAR RESULTADO POR PANTALLA EN TIEMPO REAL
            if not np.isnan(pct_5m_feb) and not np.isnan(pct_5m_may):
                config_str = str(config)
                if len(config_str) > 40: config_str = config_str[:37] + "..."

                print(header_format.format(
                    metodo,
                    config_str,
                    n_clusters_reales,
                    f"{pct_ruido:.1f}%",
                    f"{pct_clus_feb:.1f}%", f"{pct_5m_feb:.1f}%", f"{err_med_feb:.2f}m", f"{err_median_feb:.2f}m",
                    f"{pct_clus_may:.1f}%", f"{pct_5m_may:.1f}%", f"{err_med_may:.2f}m", f"{err_median_may:.2f}m"
                ))
                # Guardar los datos para representar los gráficos
                if n_clusters_reales > 1:
                    datos_para_exportar.append({
                        'Metodo': metodo,
                        'K': n_clusters_reales,
                        'Feb_Acierto': pct_clus_feb,
                        'Feb_Error_Menor_5': pct_5m_feb,
                        'May_Acierto': pct_clus_may,
                        'May_Error_Menor_5': pct_5m_may
                    })
        except Exception as e:
            print(header_format.format(metodo, str(config)[:40], "-", "-", "-", "-", "-", "-", "-", "-", "-", "-"))
            pass

    print("-" * 185)
```

```python
# Generación de gráficas
if len(datos_para_exportar) > 0:
    df_resultados = pd.DataFrame(datos_para_exportar)

    # Aseguramos que K sea numérico y ordenamos
    df_resultados['K'] = pd.to_numeric(df_resultados['K'])
    df_resultados = df_resultados.sort_values(by = 'K')

    # Configuración visual general
    sns.set_theme(style = "whitegrid")

    # DEFINICIÓN DE GRUPOS
    clus_k = ['K-Means', 'Spectral', 'Hierarchical']
    clus_densidad = ['HDBSCAN', 'MeanShift']

    # Filtramos los DataFrames
    df_std = df_resultados[df_resultados['Metodo'].isin(clus_k)]
    df_otros = df_resultados[df_resultados['Metodo'].isin(clus_densidad)]

    # Función auxiliar para pintar (para no repetir código 4 veces)
    def pintar_graficas(df, titulo_grupo, mes_key_acierto, mes_key_error, nombre_mes):
        if df.empty:
            print(f"No hay datos suficientes para el grupo: {titulo_grupo} en {nombre_mes}")
            return

        fig, axes = plt.subplots(1, 2, figsize=(16, 6))
        fig.suptitle(f'{titulo_grupo} - Resultados {nombre_mes.upper()}', fontsize=16, weight='bold')

        # Gráfica 1: % Acierto
        sns.lineplot(ax = axes[0], data = df, x='K', y = mes_key_acierto, hue = 'Metodo', style = 'Metodo',
                     markers = True, dashes = False, linewidth = 2.5, markersize = 8)
        axes[0].set_title(f'% Acierto de Cluster vs K ({nombre_mes})')
        axes[0].set_ylabel('% Acierto')
        axes[0].set_xlabel('Número de Clusters (K)')
        axes[0].legend(title = 'Método')

        # Gráfica 2: % Error < 5m
        sns.lineplot(ax = axes[1], data = df, x='K', y = mes_key_error, hue = 'Metodo', style = 'Metodo',
                     markers = True, dashes=False, linewidth = 2.5, markersize = 8)
        axes[1].set_title(f'% Localización < 5m vs K ({nombre_mes})')
        axes[1].set_ylabel('% Error < 5m')
        axes[1].set_xlabel('Número de Clusters (K)')
        axes[1].legend(title='Método')

        plt.tight_layout()
        plt.show()

    print("Generando gráficas...")

    # 1. GRUPO ESTÁNDAR (K-Means, Spectral, Hierarchical) - FEBRERO
    pintar_graficas(df_std, "K-Means / Spectral / Hierarchical", 'Feb_Acierto', 'Feb_Error_Menor_5', "Febrero")

    # 2. GRUPO ESTÁNDAR (K-Means, Spectral, Hierarchical) - MAYO
    pintar_graficas(df_std, "K-Means / Spectral / Hierarchical", 'May_Acierto', 'May_Error_Menor_5', "Mayo")

    # 3. GRUPO OTROS (HDBSCAN, MeanShift) - FEBRERO
    pintar_graficas(df_otros, "HDBSCAN / MeanShift", 'Feb_Acierto', 'Feb_Error_Menor_5', "Febrero")

    # 4. GRUPO OTROS (HDBSCAN, MeanShift) - MAYO
    pintar_graficas(df_otros, "HDBSCAN / MeanShift", 'May_Acierto', 'May_Error_Menor_5', "Mayo")

else:
    print("No se han generado datos válidos para graficar.")
```

Figura 39: Código para evaluar el clustering en exteriores.

9. ANEXO 3: RESULTADOS PROBANDO VARIAS CONFIGURACIONES.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Spectral	{'n_clusters': np.int64(2), 'affinity...	2	0.0%	96.6%	34.4%	125.10m	25.56m	88.7%	22.0%	161.14m	120.76m
Spectral	{'n_clusters': np.int64(3), 'affinity...	3	0.0%	93.9%	33.5%	126.60m	30.21m	94.9%	22.7%	157.02m	113.50m
Spectral	{'n_clusters': np.int64(4), 'affinity...	4	0.0%	92.7%	33.9%	126.43m	27.89m	83.0%	21.8%	160.08m	116.68m
Spectral	{'n_clusters': np.int64(5), 'affinity...	5	0.0%	89.9%	33.7%	128.84m	30.21m	83.7%	23.8%	156.52m	113.21m
Spectral	{'n_clusters': np.int64(6), 'affinity...	6	0.0%	90.6%	34.5%	126.22m	25.89m	81.1%	22.3%	160.94m	123.65m
Spectral	{'n_clusters': np.int64(7), 'affinity...	7	0.0%	89.3%	33.9%	126.85m	33.17m	76.8%	21.5%	164.05m	128.07m
Spectral	{'n_clusters': np.int64(8), 'affinity...	8	0.0%	86.9%	32.7%	129.94m	37.40m	72.8%	20.8%	164.51m	126.91m
Spectral	{'n_clusters': np.int64(9), 'affinity...	9	0.0%	84.8%	32.1%	129.80m	37.40m	72.3%	21.1%	164.98m	127.76m
Spectral	{'n_clusters': np.int64(10), 'affinit...	10	0.0%	80.3%	31.5%	133.01m	41.81m	70.0%	20.4%	168.80m	128.67m
Spectral	{'n_clusters': np.int64(11), 'affinit...	11	0.0%	76.5%	31.0%	135.31m	43.16m	68.3%	20.4%	167.42m	127.23m
Spectral	{'n_clusters': np.int64(12), 'affinit...	12	0.0%	73.6%	30.8%	134.84m	46.05m	64.9%	19.4%	170.43m	129.31m
Spectral	{'n_clusters': np.int64(13), 'affinit...	13	0.0%	72.5%	30.3%	139.35m	52.55m	65.5%	19.8%	169.91m	128.67m
Spectral	{'n_clusters': np.int64(14), 'affinit...	14	0.0%	72.0%	29.9%	139.02m	63.39m	64.2%	19.8%	170.02m	133.71m
Spectral	{'n_clusters': np.int64(15), 'affinit...	15	0.0%	71.9%	30.5%	138.37m	49.38m	65.9%	21.6%	163.13m	123.19m
Spectral	{'n_clusters': np.int64(16), 'affinit...	16	0.0%	71.1%	30.0%	138.82m	54.05m	65.0%	20.7%	163.35m	124.14m
Spectral	{'n_clusters': np.int64(17), 'affinit...	17	0.0%	70.4%	28.9%	139.72m	60.58m	63.3%	20.9%	162.49m	124.40m
Spectral	{'n_clusters': np.int64(18), 'affinit...	18	0.0%	70.9%	29.1%	137.61m	52.11m	63.5%	21.2%	161.36m	123.65m
Spectral	{'n_clusters': np.int64(19), 'affinit...	19	0.0%	68.3%	29.2%	141.06m	63.80m	60.0%	20.5%	163.16m	124.14m

Figura 40: Resultados spectral con affinity = nearest_neighbors.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Spectral	{'n_clusters': np.int64(2), 'affinity...	2	0.0%	96.6%	35.2%	121.62m	20.21m	93.4%	23.4%	155.08m	111.52m
Spectral	{'n_clusters': np.int64(3), 'affinity...	3	0.0%	89.5%	32.0%	132.23m	39.90m	90.6%	21.5%	159.09m	120.95m
Spectral	{'n_clusters': np.int64(4), 'affinity...	4	0.0%	77.2%	24.5%	144.72m	92.65m	71.9%	16.7%	167.53m	131.80m
Spectral	{'n_clusters': np.int64(5), 'affinity...	5	0.0%	75.0%	23.6%	148.76m	88.61m	71.4%	16.9%	172.82m	133.71m
Spectral	{'n_clusters': np.int64(6), 'affinity...	6	0.0%	75.1%	26.6%	137.85m	81.13m	65.9%	17.6%	173.63m	148.18m
Spectral	{'n_clusters': np.int64(7), 'affinity...	7	0.0%	67.6%	22.3%	151.63m	105.44m	60.0%	15.1%	178.33m	156.81m
Spectral	{'n_clusters': np.int64(8), 'affinity...	8	0.0%	63.7%	20.4%	169.18m	141.86m	57.3%	15.6%	186.94m	158.74m
Spectral	{'n_clusters': np.int64(9), 'affinity...	9	0.0%	60.1%	19.5%	174.27m	152.69m	55.7%	13.7%	183.87m	155.78m
Spectral	{'n_clusters': np.int64(10), 'affinit...	10	0.0%	63.4%	22.3%	158.66m	117.59m	55.9%	16.1%	183.23m	161.75m
Spectral	{'n_clusters': np.int64(11), 'affinit...	11	0.0%	57.4%	20.9%	165.42m	122.02m	54.1%	15.3%	187.71m	154.94m
Spectral	{'n_clusters': np.int64(12), 'affinit...	12	0.0%	56.8%	20.8%	165.73m	131.62m	53.6%	14.6%	195.15m	173.12m
Spectral	{'n_clusters': np.int64(13), 'affinit...	13	0.0%	56.6%	21.0%	166.95m	132.64m	54.0%	16.0%	188.67m	164.28m
Spectral	{'n_clusters': np.int64(14), 'affinit...	14	0.0%	52.7%	18.8%	168.31m	130.64m	49.7%	14.0%	193.90m	163.48m
Spectral	{'n_clusters': np.int64(15), 'affinit...	15	0.0%	51.1%	20.2%	167.74m	139.22m	48.0%	15.0%	191.95m	165.50m
Spectral	{'n_clusters': np.int64(16), 'affinit...	16	0.0%	46.3%	17.2%	180.63m	167.05m	44.8%	14.1%	194.20m	171.62m
Spectral	{'n_clusters': np.int64(17), 'affinit...	17	0.0%	52.5%	19.4%	163.13m	128.63m	46.2%	14.1%	182.32m	158.87m
Spectral	{'n_clusters': np.int64(18), 'affinit...	18	0.0%	51.1%	20.7%	161.99m	128.63m	45.9%	14.7%	202.19m	182.53m
Spectral	{'n_clusters': np.int64(19), 'affinit...	19	0.0%	51.2%	20.9%	165.25m	136.89m	45.7%	14.2%	185.52m	168.20m

Figura 41: Resultados spectral con affinity = rbf.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Hierarchical	{'n_clusters': np.int64(2), 'linkage'...	2	0.0%	95.1%	33.4%	123.98m	32.66m	94.7%	23.0%	155.95m	113.50m
Hierarchical	{'n_clusters': np.int64(3), 'linkage'...	3	0.0%	91.5%	32.6%	127.87m	37.31m	91.0%	21.9%	157.83m	121.97m
Hierarchical	{'n_clusters': np.int64(4), 'linkage'...	4	0.0%	88.7%	32.1%	126.78m	38.25m	88.1%	22.8%	157.53m	116.52m
Hierarchical	{'n_clusters': np.int64(5), 'linkage'...	5	0.0%	87.3%	32.3%	131.52m	36.80m	78.9%	21.9%	159.86m	123.74m
Hierarchical	{'n_clusters': np.int64(6), 'linkage'...	6	0.0%	84.8%	31.3%	132.15m	39.90m	76.7%	21.7%	159.79m	119.68m
Hierarchical	{'n_clusters': np.int64(7), 'linkage'...	7	0.0%	82.4%	31.6%	130.28m	36.80m	73.3%	21.5%	162.02m	120.95m
Hierarchical	{'n_clusters': np.int64(8), 'linkage'...	8	0.0%	81.5%	31.4%	131.58m	37.76m	72.7%	21.3%	161.57m	120.95m
Hierarchical	{'n_clusters': np.int64(9), 'linkage'...	9	0.0%	78.6%	30.6%	134.60m	41.73m	70.1%	20.7%	164.80m	127.61m
Hierarchical	{'n_clusters': np.int64(10), 'linkage...	10	0.0%	75.2%	30.9%	132.27m	40.73m	66.9%	20.2%	168.29m	127.23m
Hierarchical	{'n_clusters': np.int64(11), 'linkage...	11	0.0%	75.0%	30.8%	132.88m	42.32m	66.5%	19.8%	169.26m	127.81m
Hierarchical	{'n_clusters': np.int64(12), 'linkage...	12	0.0%	73.6%	30.3%	132.25m	43.16m	64.0%	19.7%	169.55m	132.83m
Hierarchical	{'n_clusters': np.int64(13), 'linkage...	13	0.0%	73.6%	30.2%	130.90m	41.69m	64.0%	19.8%	169.36m	131.80m
Hierarchical	{'n_clusters': np.int64(14), 'linkage...	14	0.0%	73.5%	30.3%	130.99m	41.73m	63.5%	20.0%	172.06m	142.13m
Hierarchical	{'n_clusters': np.int64(15), 'linkage...	15	0.0%	73.6%	30.5%	131.22m	41.69m	63.5%	20.4%	171.37m	137.72m
Hierarchical	{'n_clusters': np.int64(16), 'linkage...	16	0.0%	73.3%	30.5%	130.37m	40.50m	63.5%	20.3%	171.11m	137.72m
Hierarchical	{'n_clusters': np.int64(17), 'linkage...	17	0.0%	71.5%	30.1%	134.59m	42.40m	62.5%	20.2%	170.28m	137.12m
Hierarchical	{'n_clusters': np.int64(18), 'linkage...	18	0.0%	71.5%	30.3%	134.70m	42.36m	61.9%	20.3%	169.88m	137.72m
Hierarchical	{'n_clusters': np.int64(19), 'linkage...	19	0.0%	71.4%	30.4%	134.48m	42.65m	61.1%	20.3%	170.63m	137.72m

Figura 42: Resultados hierarchical con linkage = ward.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Hierarchical	{'n_clusters': np.int64(2), 'linkage'...	2	0.0%	94.5%	34.2%	138.60m	34.99m	93.5%	21.0%	181.35m	135.89m
Hierarchical	{'n_clusters': np.int64(3), 'linkage'...	3	0.0%	94.5%	34.3%	137.15m	35.01m	93.4%	21.2%	179.62m	136.48m
Hierarchical	{'n_clusters': np.int64(4), 'linkage'...	4	0.0%	93.0%	33.5%	124.24m	28.10m	90.2%	20.3%	169.62m	134.59m
Hierarchical	{'n_clusters': np.int64(5), 'linkage'...	5	0.0%	93.1%	33.6%	124.75m	28.10m	90.4%	20.5%	169.89m	134.34m
Hierarchical	{'n_clusters': np.int64(6), 'linkage'...	6	0.0%	92.5%	33.4%	124.52m	28.49m	89.2%	19.9%	170.12m	133.78m
Hierarchical	{'n_clusters': np.int64(7), 'linkage'...	7	0.0%	92.4%	33.4%	125.52m	29.18m	89.2%	19.9%	171.56m	135.89m
Hierarchical	{'n_clusters': np.int64(8), 'linkage'...	8	0.0%	92.2%	33.2%	135.14m	37.40m	89.1%	19.8%	180.52m	139.15m
Hierarchical	{'n_clusters': np.int64(9), 'linkage'...	9	0.0%	92.6%	33.6%	126.52m	31.59m	89.2%	19.9%	172.20m	132.83m
Hierarchical	{'n_clusters': np.int64(10), 'linkage...	10	0.0%	86.4%	29.8%	140.36m	48.28m	82.7%	16.9%	187.48m	161.90m
Hierarchical	{'n_clusters': np.int64(11), 'linkage...	11	0.0%	85.4%	29.3%	138.59m	52.62m	79.7%	15.9%	181.30m	151.07m
Hierarchical	{'n_clusters': np.int64(12), 'linkage...	12	0.0%	85.4%	29.4%	138.13m	51.52m	79.7%	16.0%	182.06m	151.84m
Hierarchical	{'n_clusters': np.int64(13), 'linkage...	13	0.0%	85.5%	29.5%	138.25m	51.52m	79.7%	16.0%	182.41m	152.30m
Hierarchical	{'n_clusters': np.int64(14), 'linkage...	14	0.0%	86.0%	29.8%	138.20m	51.52m	79.7%	15.9%	183.17m	154.86m
Hierarchical	{'n_clusters': np.int64(15), 'linkage...	15	0.0%	86.2%	29.9%	138.76m	47.98m	79.8%	16.0%	179.16m	152.30m
Hierarchical	{'n_clusters': np.int64(16), 'linkage...	16	0.0%	86.1%	29.9%	139.30m	48.70m	79.7%	16.0%	179.59m	153.74m
Hierarchical	{'n_clusters': np.int64(17), 'linkage...	17	0.0%	85.7%	29.6%	141.39m	55.92m	79.6%	16.0%	181.35m	155.78m
Hierarchical	{'n_clusters': np.int64(18), 'linkage...	18	0.0%	85.3%	29.3%	141.96m	64.47m	78.9%	15.6%	181.58m	155.78m
Hierarchical	{'n_clusters': np.int64(19), 'linkage...	19	0.0%	85.3%	29.3%	142.75m	69.31m	78.9%	15.7%	181.22m	155.11m

Figura 43: Resultados hierarchical con linkage = single.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Hierarchical	{'n_clusters': np.int64(2), 'linkage'...	2	0.0%	97.3%	33.8%	123.46m	24.97m	96.3%	21.6%	160.30m	123.65m
Hierarchical	{'n_clusters': np.int64(3), 'linkage'...	3	0.0%	91.8%	31.0%	127.94m	41.81m	89.2%	19.7%	164.46m	135.89m
Hierarchical	{'n_clusters': np.int64(4), 'linkage'...	4	0.0%	91.1%	32.4%	127.44m	36.74m	88.7%	20.4%	162.22m	127.23m
Hierarchical	{'n_clusters': np.int64(5), 'linkage'...	5	0.0%	90.9%	32.4%	127.56m	37.31m	87.9%	20.2%	163.15m	128.07m
Hierarchical	{'n_clusters': np.int64(6), 'linkage'...	6	0.0%	86.4%	32.2%	130.88m	36.32m	80.5%	21.6%	160.42m	124.00m
Hierarchical	{'n_clusters': np.int64(7), 'linkage'...	7	0.0%	86.3%	32.0%	131.34m	36.32m	80.5%	21.6%	160.42m	124.00m
Hierarchical	{'n_clusters': np.int64(8), 'linkage'...	8	0.0%	81.0%	30.3%	133.22m	41.69m	74.3%	20.9%	166.05m	128.75m
Hierarchical	{'n_clusters': np.int64(9), 'linkage'...	9	0.0%	79.5%	30.3%	133.00m	41.73m	71.2%	20.2%	166.50m	132.83m
Hierarchical	{'n_clusters': np.int64(10), 'linkage...	10	0.0%	79.3%	29.8%	133.84m	43.16m	70.2%	20.2%	165.92m	129.84m
Hierarchical	{'n_clusters': np.int64(11), 'linkage...	11	0.0%	79.4%	29.7%	133.88m	43.16m	70.1%	20.0%	166.31m	129.84m
Hierarchical	{'n_clusters': np.int64(12), 'linkage...	12	0.0%	78.5%	29.8%	133.97m	44.91m	67.6%	19.6%	169.34m	137.73m
Hierarchical	{'n_clusters': np.int64(13), 'linkage...	13	0.0%	76.8%	30.1%	135.15m	46.93m	65.1%	18.8%	169.56m	139.98m
Hierarchical	{'n_clusters': np.int64(14), 'linkage...	14	0.0%	76.8%	30.3%	135.07m	46.93m	65.0%	18.7%	169.69m	140.05m
Hierarchical	{'n_clusters': np.int64(15), 'linkage...	15	0.0%	75.0%	30.1%	135.32m	49.65m	64.4%	18.6%	170.40m	140.05m
Hierarchical	{'n_clusters': np.int64(16), 'linkage...	16	0.0%	74.8%	30.0%	135.73m	50.31m	63.6%	18.4%	170.68m	142.13m
Hierarchical	{'n_clusters': np.int64(17), 'linkage...	17	0.0%	74.5%	30.3%	135.30m	48.76m	63.1%	18.2%	171.38m	142.55m
Hierarchical	{'n_clusters': np.int64(18), 'linkage...	18	0.0%	74.3%	30.0%	135.29m	49.65m	63.0%	18.1%	172.14m	144.54m
Hierarchical	{'n_clusters': np.int64(19), 'linkage...	19	0.0%	74.8%	30.1%	134.95m	48.76m	62.8%	18.0%	171.26m	142.13m

Figura 44: Resultados hierarchical con linkage = complete.

MÉTODO	CONFIGURACIÓN	K	RUIDO %	F:%OK	F:<5m	F:ERR_MEDIA	F:ERR_MEDIANA	M:%OK	M:<5m	M:ERR	M:ERR_MEDIANA
Hierarchical	{'n_clusters': np.int64(2), 'linkage'...	2	0.0%	97.7%	34.6%	124.29m	23.30m	95.2%	21.9%	162.30m	126.24m
Hierarchical	{'n_clusters': np.int64(3), 'linkage'...	3	0.0%	95.7%	34.4%	130.48m	28.13m	92.6%	21.0%	170.18m	129.31m
Hierarchical	{'n_clusters': np.int64(4), 'linkage'...	4	0.0%	95.9%	34.3%	129.62m	27.68m	92.8%	21.1%	169.31m	129.10m
Hierarchical	{'n_clusters': np.int64(5), 'linkage'...	5	0.0%	94.8%	33.9%	129.20m	31.56m	89.7%	19.6%	170.63m	128.67m
Hierarchical	{'n_clusters': np.int64(6), 'linkage'...	6	0.0%	89.6%	30.9%	129.83m	42.32m	82.7%	19.2%	170.30m	135.89m
Hierarchical	{'n_clusters': np.int64(7), 'linkage'...	7	0.0%	89.5%	30.9%	129.72m	42.32m	82.6%	19.2%	170.27m	135.89m
Hierarchical	{'n_clusters': np.int64(8), 'linkage'...	8	0.0%	89.3%	33.6%	132.45m	33.59m	80.3%	20.7%	167.73m	128.67m
Hierarchical	{'n_clusters': np.int64(9), 'linkage'...	9	0.0%	89.3%	33.6%	132.45m	33.59m	80.3%	20.7%	167.73m	128.67m
Hierarchical	{'n_clusters': np.int64(10), 'linkage...	10	0.0%	88.9%	33.3%	132.60m	33.59m	79.3%	20.6%	168.35m	128.75m
Hierarchical	{'n_clusters': np.int64(11), 'linkage...	11	0.0%	89.0%	33.6%	131.82m	30.61m	79.4%	20.6%	167.73m	128.67m
Hierarchical	{'n_clusters': np.int64(12), 'linkage...	12	0.0%	88.7%	33.4%	132.09m	31.56m	78.9%	20.4%	168.45m	128.75m
Hierarchical	{'n_clusters': np.int64(13), 'linkage...	13	0.0%	88.7%	33.5%	132.01m	30.61m	78.7%	20.2%	166.79m	128.67m
Hierarchical	{'n_clusters': np.int64(14), 'linkage...	14	0.0%	87.9%	33.1%	132.98m	33.99m	77.3%	20.1%	167.78m	129.31m
Hierarchical	{'n_clusters': np.int64(15), 'linkage...	15	0.0%	89.2%	34.0%	131.63m	29.12m	76.2%	19.7%	169.24m	129.84m
Hierarchical	{'n_clusters': np.int64(16), 'linkage...	16	0.0%	89.2%	34.0%	131.62m	29.12m	76.0%	19.6%	169.37m	129.84m
Hierarchical	{'n_clusters': np.int64(17), 'linkage...	17	0.0%	88.1%	33.5%	132.37m	33.48m	74.0%	19.0%	170.42m	134.34m
Hierarchical	{'n_clusters': np.int64(18), 'linkage...	18	0.0%	85.3%	33.8%	129.47m	31.56m	73.3%	19.3%	168.95m	133.78m
Hierarchical	{'n_clusters': np.int64(19), 'linkage...	19	0.0%	85.2%	33.8%	129.47m	31.56m	73.2%	19.3%	168.95m	133.78m

Figura 45: Resultados hierarchical con linkage = average.