Escuela de Ingenierías Industriales
Universidad de Málaga

TRABAJO FIN DE GRADO

Creación de mapas topológicos por apariencia visual mediante corte de grafos. Aplicación al reconocimiento de lugares previamente visitados

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Málaga, 2023 Ángel Montejo Quesada

Universidad de Málaga
Escuela de Ingenierías Industriales

Creación de mapas topológicos por apariencia visual mediante corte de grafos. Aplicación al reconocimiento de lugares previamente visitados

Autor
Ángel Montejo Quesada

Tutores
Javier Gonzalez Jimenez

Departamento: Ingeniería de Sistemas y Automatización

Titulación: Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Palabras clave: Topología de lugares, Mapa de Apariencia, Localización basada en Apariencia, Reconocimiento Visual de Lugares, descriptores de apariencia, Grafo de Apariencia

Resumen:
La Localización basada en Apariencia (AbL) es una estratégia de Localización Visual que consiste en estimar la posición y orientación de un agente a partir de imágenes de un entorno previamente visitado. En esta técnica, se representa el entorno mediante un Mapa de Apariencia (AM), el cual consiste en una colección de imágenes geolocalizadas que se codifican empleando vectores descriptores de apariencia.
Uno de los desafios más relevantes en esta técnica es conocido como *Perceptual Aliasing*, fenómeno en el cual dos imágenes que se encuentran distantes en el espacio se codifican en vectores descriptores similares debido a la similitud visual entre ellas, lo que puede conducir a errores en la localización. En este trabajo se presenta una estrategia para crear topologías de lugares que eviten este fenómeno. La propuesta se basa en la construcción de un Grafo de Apariencia a partir de los elementos del AM, sobre el que se aplica una bipartición recursiva, generando grupos de elementos similares en apariencia. Esta topología sienta las bases para aplicaciones de Localización basada en Apariencia, donde se evita el fenómeno del *Perceptual Aliasing*, logrando un mayor rendimiento en la localización.

University of Málaga
Industrial Engineering School

Creation of Topological maps by visual appearance bu cutting graphs. Application to the recognition of previously visited places

Author
Ángel Montejo Quesada

Supervisors
Javier Gonzalez Jimenez

Department: Systems and Automation Engineering

Degree: Degree in Electronics, Robotic and Mechatronic Engineering

Keywords: Topology of places, Appearance Maps, Appearance-based Localization, Visual Place Recognition, image descriptors, Appearance Graphs

Abstract:
The task of Appearance-based Localization (AbL) falls within the domain of Visual Localization and entails the estimation of an agent's position and orientation using images of previously visited environments. This technique employs an Appearance Map (AM) as the representation of the world, which consists of a collection of geolocated images encoded with appearance descriptors.
A significant challenge encountered in this approach is referred to as Perceptual Aliasing, whereby images that are visually similar result in similar appearance descriptors despite being located at different positions in space. Consequently, this phenomenon gives rise to localization errors. In this letter, we propose a novel strategy for addressing this issue by constructing a topology of places that mitigates Perceptual Aliasing.
The proposed methodology involves constructing an Appearance Graph from the elements in the AM and subsequently applying a recursive bisection algorithm to create groups of elements exhibiting similar appearances. This topological structure serves as a foundation for the development of AbL applications, with the primary objective of enhancing localization performance by effectively circumventing the limitations imposed by Perceptual Aliasing.

Agradecimientos

Me gustaría dedicar este trabajo a mi familia, por todo el amor, apoyo y confianza que me han regalado todo este tiempo, incluso en los momentos más complicados.

También a Elena y a Paco, por quererme y estar ahí a jornada completa.

Este TFG ha sido parcialmente financiado por:

- Grupo de investigación MAPIR.

Índice

Parte I

Introducción

Capítulo 1

Introducción

Contenido

Sinopsis

En este primer capítulo se expone la motivación que impulsa la concepción de este Trabajo de Fin de Grado, acompañada de una breve introducción. A continuación, se describen los antecedentes y el contexto bajo el que se ha realizado y se exponen los objetivos que se pretenden lograr en éste. Por último, se realiza una descripción de la estructura de la memoria.

1.1. Motivación

La robótica móvil es una de las áreas más activas y prometedoras en la investigación robótica moderna. En este campo, para que un robot opere de manera autónoma en un entorno dinámico es fundamental que sepa localizarse a sí mismo. El problema de la localización ha sido un pilar fundamental para el desarrollo de esta tecnología, siendo a día de hoy un importante nicho de investigación.

En los últimos años, se han desarrollado varios métodos de Localización Visual (Visual Localization, VL) más precisos y confiables que permiten a los robots estimar su posición y orientación (denominada *pose*) en tiempo real a partir de información visual. Entre ellos podemos distinguir dos categorías: métodos basados en extracción y seguimiento de landmarks 3D, y métodos basados en modelado de la apariencia del entorno [1].

De estas dos posibilidades para VL, la alternativa más utilizada se está basada en construir un mapa de marcas o *landmarks* 3D. La proyección de estas marcas en una imagen deben de coincidir con sus observaciones. La minimización del error entre ambas (observaciones y proyecciones) permite obtener la pose de la camara. Este enfoque se sustenta en el modelo matemático de la cámara, que emplea geometría proyectiva para determinar la proyección de marcas en la cámara como función de la pose de la cámara. Este enfoque asume que los puntos de referencia del entorno pueden ser identificados con precisión en la imagen y que su ubicación se puede inferir combinando la lectura de múltiples sensores. Sin embargo, la transformación de las lecturas de los sensores (3D) a información geométrica (2D) es, en general, compleja y propensa a errores [2] y requieren una gran capacidad de cómputo [3]. Además la características se representan por medio de descriptores locales que no son invariantes a cambios en la iluminación [4], lo que hace a estos métodos poco robustos a cambios ambientales y de iluminación

En este trabajo nos centramos en el segundo de los enfoques de VL, denominados Localización por Apariencia, o Appearance-based Localization (AbL) [5] [6]. Esta técnica consiste en estimar la pose de la cámara a partir de la apariencia del entorno, medida mediante un vector descriptor que codifica la imagen en una dimensión reducida. Típicamente estos descriptores son entrenados para ser invariantes ante cambios en la iluminación [7] lo que lo convierte en una alternativa más robusta que los landmarks 3D.

De acuerdo con este paradigma, el entorno se representa como un conjunto reducido de descriptores geolocalizados [8] [9] al que se denomina Mapa de Apariencia o Appearance Map, *AM*. Así evita una representación 3D explícita del entorno, lo que mejora la eficiencia computacional y la robustez a la variabilidad de la iluminiacion

de la escena, ya que se evitan los descriptores locales que son muy sensibles a estos cambios. Sin embargo, esta variante de VL presenta algunos inconvenientes.

La principal limitación de los sistemas AbL radica en su baja precisión métrica, significativamente inferior a la de las técnicas basadas en mapas 3D. Como se defiende en [10] la apariencia visual está pobremente relacionada con la estructura geométrica del entorno, lo que se traduce en una disminución significativa en el rendimiento métrico. Sin embargo, la apariencia visual ha demostrado ser una representación especialmente adecuada en tareas de localización topológica [11] [12] [13], lo que se conoce como reconocimiento visual de lugares (Visual Place Recognition, *VPR*). En VPR no se busca estimar la pose de una imagen, sino asignarla al elemento más visualmente similar en el mapa de apariencia (AM) para asi estimar el lugar desde donde fué captada.

En el presente Trabajo de Fin de Grado (TFG) se propone un método para crear una topología de lugares de apariencia similar, adecuada para el problema de *VPR*. La premisa consiste en que cada elemento dentro de la topología se corresponde con un lugar, esto es, una región específica del espacio de poses del *AM,* desde la que se observa una escena común, aunque sea de forma parcial. La idea es agrupar los elementos del AM que tengan un descriptor de apariencia similar.

Es importante considerar que los elementos del entorno pueden aparecer repetidos, lo que implica que es probable encontrar descriptores de apariencia muy similares en el AM, a pesar de estar asociados a poses muy distantes en el espacio. Este fenómeno, conocido como *Perceptual Aliasing* (*PA*), plantea el desafío de contar con un indicador que permita determinar si la región observada es realmente la misma. Para abordar esta cuestión, se puede utilizar la pose asociada a cada descriptor en el *AM*. Sin embargo, la distancia entre poses está pobremente relacionada con la similitud en apariencia, dando lugar a agrupaciones disfuncionales que no son representativas en términos de apariencia o pose.

Por este motivo, este TFG desarrolla un método novedoso que agrupa los elementos del *AM* combinando tanto la similitud en apariencia como el grado de solapamiento del campo de visión de las cámaras, al que se denomina co-visibilidad. Específicamente, el método propuesto construye un grafo, al que denominamos Grafo de Apariencia (*GA*), donde cada nodo representa un elemento del *AM* (es decir, un par descriptor-pose), y cada arco entre nodos indica tanto el grado de similitud en apariencia como el grado de co-visibilidad entre las imágenes. Posteriormente, el Grafo de Apariencia (*GA*) es particionado de manera recursiva mediante una técnica de bipartición, lo que resulta en una topología de lugares compuesta por los diferentes grupos de nodos obtenidos.

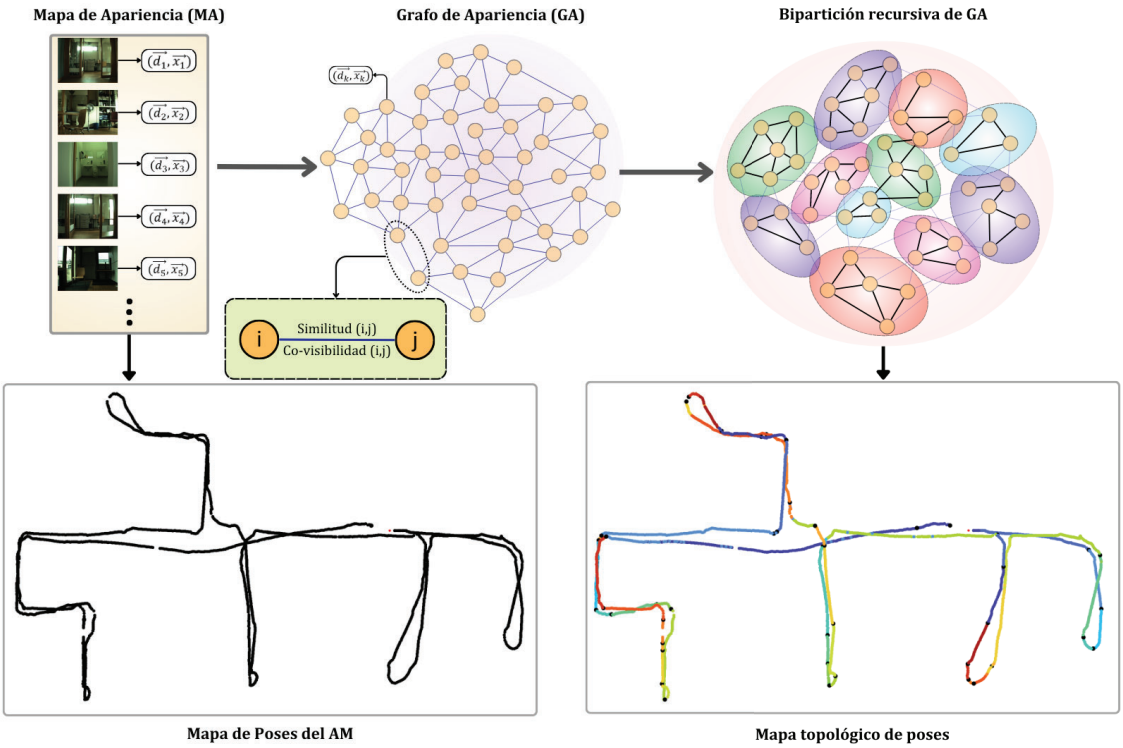


Figura 1.1: Esquema general del procedimiento de obtención de la topología de lugares propuesta en este TFG. El Mapa de Apariencia (AM) se representa como un conjunto de elementos desordenados que consisten en pares descriptor-pose $(\vec{d}_i, \vec{x}_i)$. En primer lugar se construye el Grafo de Apariencia (GA) donde cada nodo es un elemento del AM, se evalúa la similitud y co-visibilidad para establecer el peso del arco entre nodos. Por último, se aplica la bipartición recursiva del GA, dando lugar a una topología de lugares.

1.2. Contexto de realización del TFG

Este Trabajo de Fin de Grado (TFG) se ha desarrollado dentro del entorno del equipo de investigación Machine Perception and Intelligent Robotics (MAPIR) que forma parte del departamento de Ingeniería de Sistemas y Automatización de la Universidad de Málaga. En concreto, forma parte de la línea de investigación en Visión por Computador como resultado natural de las conclusiones en previas investigaciones del equipo.

Este TFG se fundamenta a partir de 2 trabajos previos de este equipo de investigación: la Tesis Doctoral titulada 'Appearance-based Localization' realizada por D. Alberto Jaenal Gálvez [10] y el artículo 'Subjective Local Maps for Hybrid

Metric-Topological SLAM' por J.L Blanco [14].

En [10] se nos describe el marco general del problema de Appearance-based Localization haciendo énfasis en sus ventajas frente a métodos tradicionales y en sus limitaciones. A partir de [14] rescatamos una herramienta de división de un grafo para crear grupos de nodos (imágenes en nuestro caso) que comparten una apariencia similar, y que empleamos aquí para la construcción del AM.

El presente trabajo pretende establecer unas bases para futuros proyectos del grupo de investigación en esta línea de trabajo, cuyo propósito es llevar a cabo una localización fotométrica jerárquica con un primer módulo de VPR, que atañe a este trabajo, y un segundo módulo de localización, de refinamiento de la pose.

1.3. Objetivo y alcance

El presente TFG se ha centrado en el estudio del problema de la Localización Visual para robots móviles a partir de la apariencia de la escena y, específicamente, para la creación de una topología de lugares que posibilite esta localización. La estrategia global que se plantea es llevar a cabo una localización topométrica, esto es, en dos niveles: la localización topológica del lugar donde se encuentra la cámara (problema de Visual Place Recognition, VPR) y a continuación realizar un refinamiento de la pose dentro de este lugar.

Para crear el mapa topológico de lugares en base a la apariencia similar, se propone un método de agrupacion de las vistas basado en el corte normalizado de grafos. Así, los nodos de este grafo son las observaciones de la escena, y el peso de los arcos que los conectan es el parecido en apariencia, de acuerdo con algun descriptor global de la imagen. Para facilitar el algoritmo de corte de grafo se desarrolla una test de co-visibilidad entre observaciones, que acota la conexion de los nodos del grafo a aquellos con un solape alto del campo de visión de las cámaras. Por tanto, el alcance del proyecto incluye los siguientes objetivos:

- Estudio del Estado del Arte, describiendo los principales problemas que presenta la Localizacion por Apariencia, para aportar una estratégia que mejore dichos problemas.
- Dado un Mapa de Apariencia, construir un Grafo de Apariencia (GA) que modele tanto el parecido de la apariencia como la co-visibilidad de la observaciones. Una parte esencial de esta parte es desarrollar un método que estime la co-visibilidad entre imágenes.
- Dado un Grafo de Apariencia (GA), implementar el Corte Normalizado del mismo que dé como resultado una topología adecuada para el *VPR*. Puesto

que el corte de grafo es binario, se ha de estudiar como aplicar la recursividad y hasta que nivel.

- Evaluar este método como un sistema de *VPR*.

De entre todos estos desarrollos, destacamos dos contribuciones principales de este TFG: El método para medir la co-visibilidad de dos observaciones, y el algoritmo de bipartición recursiva del Grafo de Apariencia.

1.4. Estructura de la memoria

La presente memoria se compone de 7 capítulos, incluyendo esta Introducción. El Capítulo 2 se dedica a introducir los fundamentos que sustentan esta propuesta, llevando a cabo una revisión del Estado del Arte y presentando las herramientas utilizadas en el desarrollo del trabajo.

En el Capítulo 3 se expone el planteamiento de la topología de lugares, definiendo de manera formal diversos aspectos relacionados con la Localización basada en Apariencia. Así mismo y se introduce el concepto de Grafo de Apariencia y la estrategia para construir la mencionada topología de lugares

El Capítulo 4 detalla el proceso de construcción del Grafo de Apariencia, enfatizando en la definición y el procedimiento de detección de la co-visibilidad y similitud sobre un Mapa de Apariencia.

En el capítulo 5 se describe el procedimiento de bipartición recursiva del Grafo de Apariencia, dando lugar a la topología de lugares. Se presentan las técnicas sobre las que se fundamenta esta estratégia.

El Capítulo 6 se centra en la evaluación de la metodología propuesta mediante una serie de pruebas en las que se analiza por separado la construcción del Grafo de Apariencia y la topología de lugares obtenida.

En el capítulo 7, se presentan las conclusiones derivadas de este trabajo y se plantean posibles líneas futuras de investigación a partir del mismo.

Capítulo 2

Fundamentos y Herramientas

Contenido

Sinopsis

En este capítulo se exponen las herramientas fundamentales en las cuales se basa este trabajo. Específicamente, se lleva a cabo un análisis del Estado del Arte en lo que respecta a descriptores de apariencia y Localización basada en Apariencia.

Además, se destaca la geometría epipolar como una herramienta esencial para el desarrollo de esta investigación. Por último, se introduce la biblioteca OpenCV, la cual implementa la teoría que sustenta la geometría epipolar.

2.1. Localización basada en Apariencia

La Localización basada en Apariencia (Appearance-based Localization, AbL) es una técnica de localización utilizada para estimar la pose de un agente en un entorno a través del análisis de las características visuales de la escena que observa, evitando la representación explícita 3D de los elementos de la escena. Esta técnica se fundamenta en la idea de que la pose de una cámara se puede inferir a partir de las características visuales o apariencia del entorno, medida mediante un vector descriptor que codifica la imagen en una dimensión reducida. Típicamente estos descriptores son entrenados para ser invariantes ante cambios en la iluminación [7] lo que lo convierte en una alternativa más robusta que los landmarks 3D.

En consecuencia, se puede entender un sistema de AbL como la combinación de tres procesos, como se ilustra en la Figura 2.3. La extracción de la descripción (probablemente utilizando una CNN), la representación de la escena a través de un mapa compuesto de pares pose-descriptores, que dan lugar a un espacio topológico, y la localización en el espacio de una imagen de consulta.

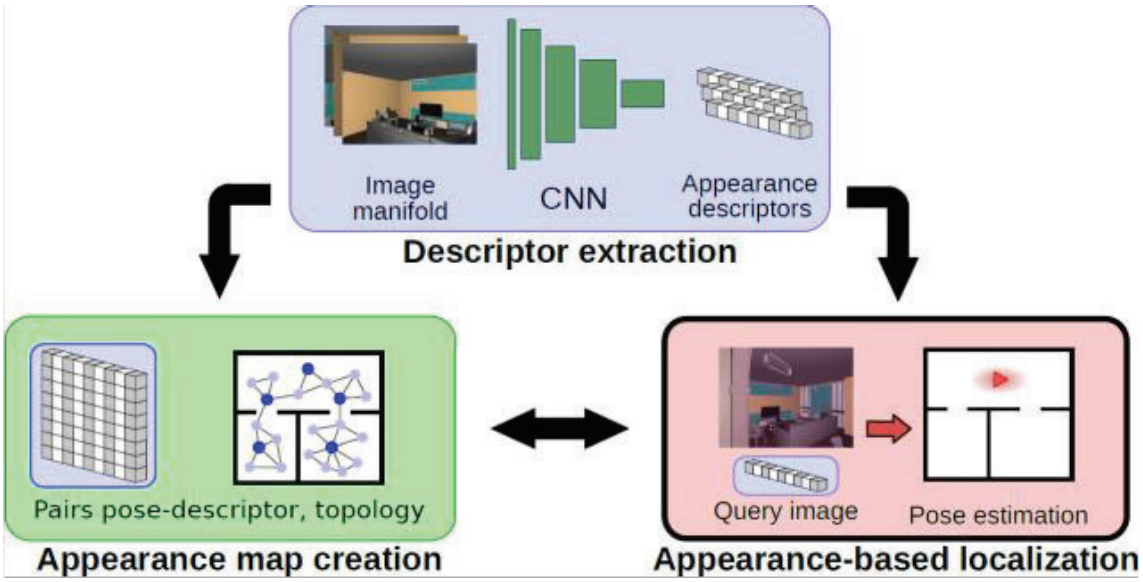


Figura 2.1: Los 3 ingredientes principales en un problema de AbL.

En este contexto, se define un criterio para la localización de una imagen de consulta que se basa en comparar el vector descriptor de consulta con los descriptores guardados en el Mapa de Apariencia (AM). La composición e interpretación adecuada del AM resulta fundamental para un proceso de localización eficaz. El presente trabajo se centra en la creación de una topología de lugares para interpretar el AM a fin de lograr una localización óptima.

En los últimos años el AbL has sido un área de gran interes en el campo de la robótica y la visión por computador, dando lugar a varias propuestas que sugieren distintos enfoques para trabajar con el AM. Algunos enfoques optan por aprender el mapa completo del entorno empleando, en [15] [16] emplean el aprendizaje por transferencia con CNN para aprender una función no lineal que mapea las características visuales de la imágen con las poses de la cámara. En [17] [18] se plantea el uso de descriptores visuales basados en CNN para crear mapas densos donde la localización se lleva a cabo empleando un filtro de partículas para estimar la pose dentro del marco de un proceso gaussiano.

Algunas alternativas abogan por el emparejamiento de secuencias de imágenes ordenadas [19] [20] [21] en lugar de usar imágenes individuales, empleando la cuantización escalar para generar representaciones compactas y eficientes de los lugares. En otros enfoques se basan en el agrupamiento de las imágenes a partir del algoritmo de k-means en base a su apariencia y pose [22] [23] como técnica para abstraer la base de datos de imágenes georeferenciadas en una representación simplificada basada en clusters.

Los métodos mencionados aportan un gran avance en el campo del VL, sin embargo todos presentan limitaciones:

- Las técnicas que abordan la apariencia como información individual por imagen [15] [16] [17] [18] pueden enfrentar una complejidad computacional significativa al manejar grandes bases de datos y operar en tiempo real. Esta complejidad se ve acentuada en particular para las técnicas que hacen uso de redes neuronales convolucionales (CNN).

- Las técnicas basadas en secuencias de imágenes [19] [20] [21] requieren que las imágenes estén ordenadas y sean dependien en gran medida de la velocidad y frecuencia de muestreo de la cámara. Esto implica una baja robustez y poca flexibilidad en tareas de modificación o ampliación del mapa.

- En contraposición, las técnicas de clusterización [22] [23] ofrecen una ventaja comptuacional frente a los anteriores. Sin embargo, al estar basadas en técnicas de k-means requieren de una cantidad predefinida de grupos para realizar la clasificación, lo que implica una gran desventaja al adaptarlo a diversos entornos.

Debido a esto, los mapas más comúnmente utilizados en AbL no garantizan una distribución eficiente de sus muestras, lo que puede tener un efecto negativo en el rendimiento de la localización y limitar su aplicabilidad, así como aumentar su costo computacional.

2.2. Descriptores de la imagen

La extracción de la información visual más relevante en una escena dada ha planteado un desafío significativo en el campo de la visión por computador. El procesamiento directo de imágenes es computacionalmente costoso debiado a la alta dimensionalidad de los datos, por ello en los ultimos años se han desarollado diversas técnicas para sintetizar la informacón visual más relevante de una imágen a fin de reducir su dimensionalidad. Se conoce como descriptores de una imágen, son vectores, o conjuntos de vectores, que representan caracteristicas en la escena (texturas, formas, colores, partrones, etc.). En su fundamento, se tratan de una trasnformación matemática entre el espacio vectorial de las imágenes I y el espacio del descriptor D tal que $D \ll I$.

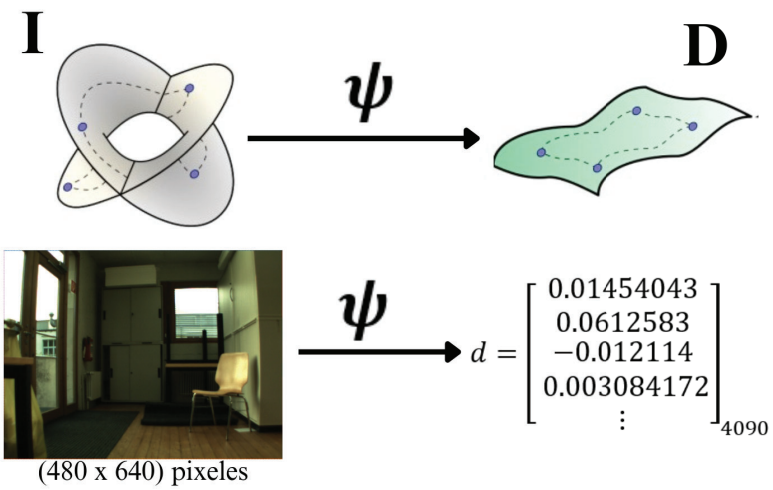


Figura 2.2: Función de reducción de dimensionalidad entre el espacio imagen $I = \{307200\}$ y el espacio de descriptores $D = R\{4096\}$ para un descriptor global.

Al analizar y comparar estos descriptores, se pueden detectar similitudes, realizar clasificaciones y extraer información semántica de las imágenes. En el contexto del AbL el termino 'apariencia' de la escena hace referencia a la sintetización de la información visual por medio de estos descriptores.

Típicamente, estos descriptores están diseñados para ser invariantes ante cambios en la apariencia, como la iluminación, las condiciones meteorológicas y la perspectiva de la escena, lo que los hace especialmente adecuados para tareas de recuperación de imágenes(*Image Retrieval*) para *VPR*. Estos descriptores pueden clasificarse en dos grupos: descriptores locales y descriptores globales.

Descriptores locales

Los descriptores locales de la imagen son conjuntos de vectores numéricos que se utiliza para representar características visuales específicas de una región de la imagen. Estas características pueden ser detectadas mediante algoritmos de procesamiento de imágenes, como la detección de bordes, esquinas, puntos de interés, entre otros [24]. Algunos ejemplos populares de estos descriptores en la literatura incluyen *SIFT* [25] [26], SURF [27] y ORB [28].

En este trabajo, emplearemos el descriptor *SIFT* (Scale-Invariant Feature Transform) debido a su rendimiento superior en comparación con otros descriptores locales, como se ha demostrado [24]. *SIFT* se clasifica como un detector de manchas o *blob detector*, donde cada vector describe una característica local (una mancha de color) en la imagen, representando la orientación y magnitud de los gradientes de la intensidad de los píxeles en el entorno de vecindad de la mancha. La principal ventaja de *SIFT* es su invariancia a la escala y la rotación de las características, que se logra aplicando varios filtros gaussianos en una pirámide gaussiana para detectar manchas de diferentes tamaños en la escena. Posteriormente, se aplica una rotación y una normalización a cada mancha para lograr la invariancia en cuanto a la orientación e iluminación. Para obtener una explicación más detallada, se pueden consultar los trabajos citados en [25] [26].

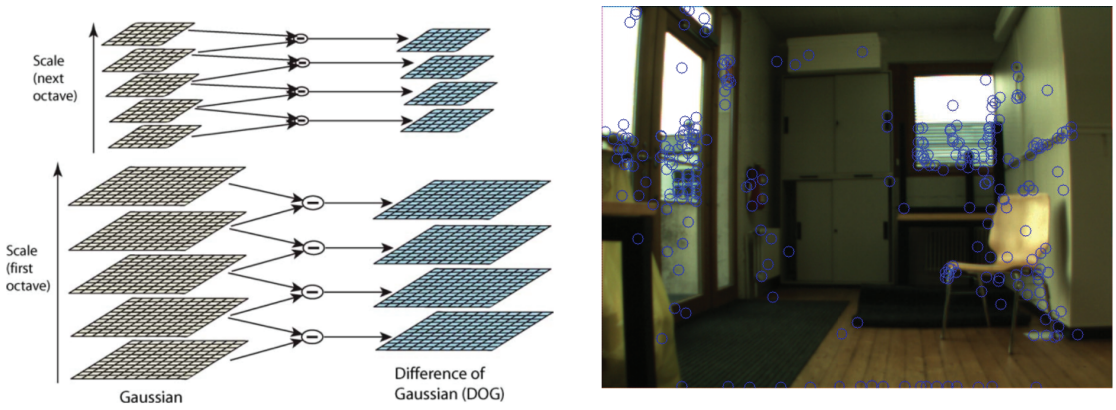


Figura 2.3: Ejemplo de detección de features por *SIFT* por el método de la piramide gaussiana [25]. Se ilustra el resultado de la detección de *SIFT* para una imagen real.

Otra ventaja significativa al trabajar con *SIFT*, y en general con descriptores locales, es que además de obtener un conjunto de vectores descriptores que caracterizan las manchas de la imagen, también obtenemos un conjunto de vectores que representan la ubicación del centroide de cada mancha dentro del plano imagen, conocidos como *keypoints*. Esta información geométrica resulta de gran interés para la investigación en curso, ya que representa la proyección de un elemento de la

escena 3D en el plano de la imagen. A partir de esta información podemos deducir relaciones geométricas entre pares de imágenes, como se detalla en la sección 2.4.

Descriptores Globales:

A diferencia de los descriptores locales, un descriptor global de la imagen busca una representación compacta de la escena en un solo vector. Estos descriptores globales son muy útiles en tareas de comparación, búsqueda en grandes bases de datos y recuperación de imágenes (*Image Retrieval*) como estratégias para el *VPR*.

Normalmente, los descriptores globales utilizados en la localización se basan en el enfoque de la bolsa de palabras (BoW, Bag of Words) [29] [30]. Este método implica la discretización del espacio de los descriptores locales, asignando cada descriptor a una palabra de un vocabulario visual para un conjunto de descriptores locales de una imagen. Se crea un histograma de frecuencia que representa la apariencia global de la imagen. Existen variantes mejoradas de este método que extraen información adicional del proceso de clasificación de los descriptores, como VLAD (Vector of Locally Aggregated Descriptors) [31] o FV (Fisher Vector) [32].

Recientemente, han surgido enfoques que proponen el uso de redes neuronales convolucionales (CNN, Convolutional Neural Networks), como ImRet [33] o *NetVLAD* [7] , los cuales han demostrado una mejora en eficiencia en comparación con las técnicas basadas en BoW. En concreto, *NetVLAD* nace como una mejora del método VLAD, que logra una mayor robustez frente a cambios de iluminación, escala y rotación [7].Consiste en una CNN entrenada para *VPR* cuya principal componente es una capa generalizada basada en VLAD.

En el presente estudio, hemos empleado *NetVLAD* como el descriptor global de apariencia de las imágenes. En consecuencia, el Mapa de Apariencia sobre el que se contruye el Grafo de Apariencia consiste en un conjunto de elementos que constan de un vector *NetVLAD* etiquetado con una pose (consulte el capítulo 3).

2.3. Visual Place Recognition

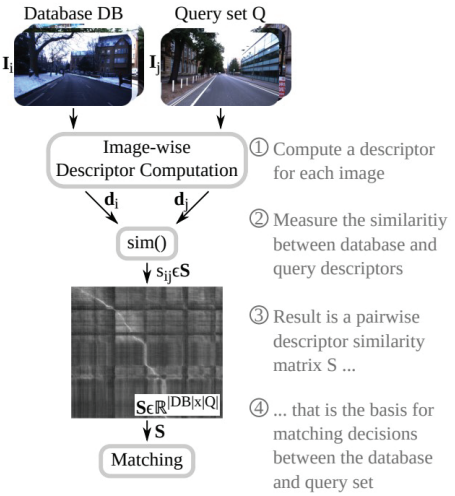
El Reconocimiento Visual de Lugares, o Visual Place Recognition (*VPR*) [34] [35], es la versión más simple de AbL. Consiste en la localización topológica de una cámara en el entorno. El *VPR* [36] [37] busca encontrar el lugar mas similar para una cierta imagen de consulta, entendiendo un *lugar* como una región del entorno. Para ello, se trabaja sobre un dataset compusto por imágenes geolocalizadas (AM), codificadas por un descriptor global [11] [29] [7], con secuencias de imágenes [21] o con un conjunto de elementos desordenados [38].

Dada la baja precisión métrica de los sistemas de AbL, en las últimas décadas muchos enfoques basados en la apariencia han optado por ignorar sus capacidades

métricas y centrarse en la localización topológica [19] [39] [40]. De hecho la apariencia visual ha demostrado ser una representación especialmente adecuada para tareas topológicas [11] [12] [13], como el reconocimiento visual de lugares (Visual Place Recognition, *VPR*). Estos enfoques no buscan estimar la pose de una imagen de consulta, sino asignarla al elemento más similar visualmente en el Mapa de Apariencia (AM). Consideran la apariencia como una variable discreta con valores individuales para cada elemento dando solución al problema del *Image Retrieval*.

Típicamente el *VPR* se trata de un problema de *Image Retrieval* [41] [35], encontrar imágenes relevantes entre una base de datos de imágenes para una consulta dada. Por lo general, el *VPR* trata de encontrar las imágenes del AM más similares a la consulta, devolviendo el lugar al que perteneces, en teoría, el lugar desde el que fue tomada la imagen de consulta. En este contexto, se considera la apariencia como una variable discreta con valores individuales para cada elemento.

Figura 2.4: Pasos y componentes clave de *VPR* como un problema de *Image Retrieval* [35].



Normalmente, se emplea un criterio de similitud para designar un lugar, considerando exclusivamente su aparaiencia y sin tomar en cuenta el aspecto espacial de *VPR*, por lo que no se logra abordar el problema del *Perceptual Aliasing* (lugares distantes en pose pero que comparten una apariencia similar). Esto deriva en una pobre estimación de la pose.

Tradicionalemente, este problema se resuelve definiendo una topología adicional sobre el Mapa de Apariencia, por ejemplo con en dataset secuenciales [42] [11]. Para definir esta topología se busca crear regiones sobre el AM que sean similares en apariencia y donde, idealmente, también son cercanas en términos de pose. En este trabajo, se propone una topología de lugares compuestos de conjuntos de elementos desordenados del Mapa de Apariencia similares en apariencia, donde además se tiene en cuenta la co-visibilidad de la escena, como se explica en detalle en el 4. En la sección 6 se implementa un sistema de *VPR* simple para evaluar la topología presentada.

2.4. Geometría Epipolar

La geometría epipolar es un concepto fundamental en el campo de la visión 3D para describir las restricciones entre las poses de dos vistas (imágenes) de una escena [43]. Tradicionalmente se emplea en diversas tareas como la autocalibración de cámaras [44], la reconstrucción 3D de una escena a partir de dos o más imágenes [45], o la creación de panorámicas mediante la técnica de image stitching [46]. En nuestro estudio, lo que resulta interesante de esta técnica es que se basa en la suposición de que las dos imágenes relacionadas representan una escena común. En este artículo, exploramos esta característica para determinar cuando un par de imágenes son co-visibles, o en otras palabras, representan la misma escena.

Marco teórico

La geometría epipolar describe las relaciones geométricas entre los puntos tridimensionales de una escena y sus proyecciones bidimensionales en dos planos de imágenes con diferentes puntos de vista. Esta metodología asume que el modelo de la cámara se puede aproximar mediante el modelo de cámara pinhole. En la Figura 2.5, se ilustra un caso base para analizar los principales componentes de la geometría epipolar. Se representan la intersección de los planos de imagen I e I' con el plano que une un punto $\mathbf{X}$ de la escena tridimensional ($\mathbf{X} \in SE(3)$) [47] y sus proyecciones bidimensionales $\mathbf{x}, \mathbf{x'} \in Im(2)$ en las imágenes.

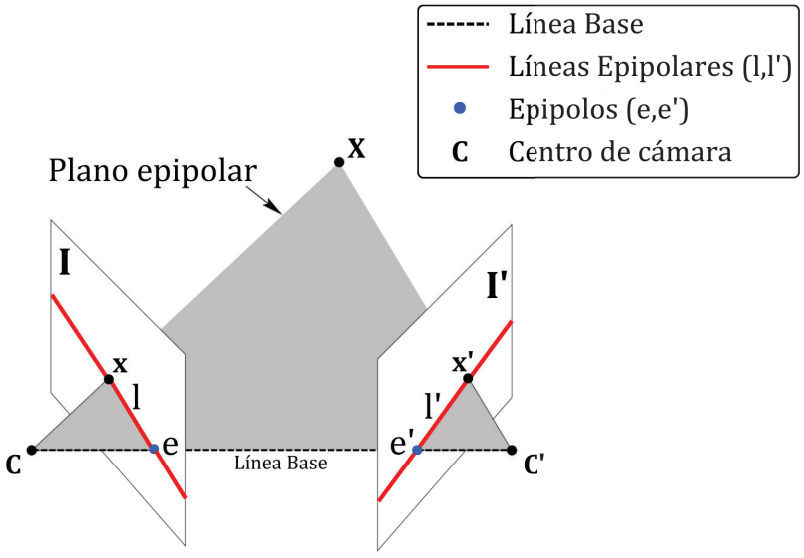


Figura 2.5: Caso base de la Geometría Epipolar, se representan los elementos relevantes para entender el fundamento de esta geometría [43].

Se define una línea imaginaria que une los dos centros de las cámaras, conocida como línea base. Como se muestra en la Figura 2.5, el punto **X** y la línea base forman un plano en el espacio, denominado plano epipolar, al cual también pertenecen las proyecciones (**x**, **x'**). En la Figura 2.6, se presenta un caso más realista en el que se representan varios puntos de la escena. Podemos observar que se define un plano epipolar para cada punto tridimensional representado, generando así un haz de planos que tienen como centro la línea base.

La intersección del plano epipolar con el plano de imagen resulta en una línea epipolar. Cada plano epipolar i corta los planos de imagen I e I', generando las líneas epipolares **li** y **l'i**. Todas las líneas epipolares de una imagen se intersectan en el epipolo, ya que pertenece a la línea base, que es el centro del haz de planos.

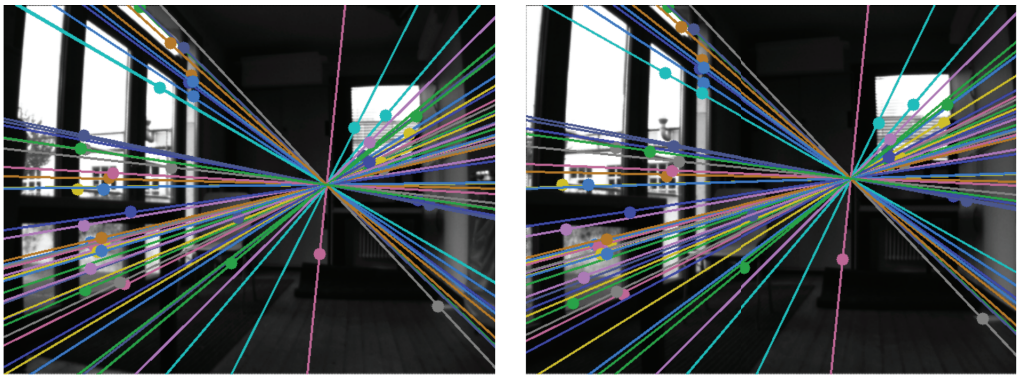


Figura 2.6: Caso real de lineas epipolares como intersecciones del haz de planos epipolares con las imágenes. Se muestran por correspondencia de colores las líneas epipolares relativas.

En conclusión, para cada punto **X** en la escena y sus proyecciones **x** y **x'**, se puede establecer una relación a través de un plano epipolar de manera que cada punto $\mathbf{x} \in I$ existe una línea epipolar correspondiente **l'** $\in I'$. En este sentido, la línea epipolar **l'** es la proyección del rayo que une el punto **x** con el centro de la cámara c sobre el plano imagen I'. Dada esta relación coplanar entre los puntos, se cumple que el punto **x'** pertenece a la línea epipolar **l'** correspondiente a **x** y viceversa. Una vez analizado el caso base, procederemos a introducir la matriz fundamental.

Matriz Fundamental

La Matriz Fundamental [43] es la representación algebraica de la geometría epipolar que establece una relación entre los puntos y líneas epipolares en ambas imágenes. Se trata de una matriz 3x3, con rango 2, que no depende de la estructura de la

escena y se puede calcular a partir de las correspondencias de puntos, sin necesidad de conocer los parámetros internos de las cámaras ni sus poses relativas.

Existe una transformación que mapea un punto **x** con su respectiva línea epipolar **l'**, al igual que un punto **x'** con su línea epipolar **l**. Esta transformación puede ser expresada matricialmente mediante la matriz fundamental F, como se muestra en la ecuación 2.1:

$$\begin{aligned} F \cdot x &= l' \\ F^T \cdot x' &= l \end{aligned} \tag{2.1}$$

La característica principal de la matriz fundamental es que impone la condición de correspondencia, también conocida como restricción epipolar. Para dos puntos correspondientes, **x** y **x'**, donde F es la matriz fundamental que los relaciona, se cumple que **x'** $\in$ **l'**$|F \cdot x = l'$. Dado que el producto escalar entre dos vectores ortogonales es cero, se cumple que $(x')^T \cdot l = 0$. Basándonos en esta condición, podemos derivar la restricción epipolar como se muestra en la ecuación 2.2. Si esta condición no se cumple, los puntos no son correspondientes, es decir, no representan el mismo punto **X** en la escena.

$$(x')^T \cdot l' = (x')^T \cdot F \cdot x = 0 \tag{2.2}$$

Esta restricción proporciona una herramienta poderosa para relacionar geométricamente pares de puntos en imágenes. En aplicaciones que emplean la geometría epipolar [46], se parte de un conjunto de pares de puntos relacionados mediante técnicas de coincidencia de descriptores locales y se emplea la matriz fundamental para filtrar estos pares y encontrar correlaciones auténticas entre puntos, conocidos como inliers. Este proceso se conoce como filtrado de outliers o malas correspondencias entre puntos.

En nuestra propuesta planteamos un enfoque similar en un problema de filtrado de outliers para determinar si un par de imágenes representan un mismo lugar. Como se mencionó anteriormente, la geometría epipolar asume que las imágenes de entrada representan una escena común. En este sentido, si no es posible establecer la correspondencia epipolar proporcionada por la matriz fundamental entre las imágenes, podemos concluir que dichas imágenes no representan la misma escena. Como se explica en el capítulo 4, aprovechamos esta condición para inferir información geométrica adicional que relaciona los pares de imágenes en nuestro dataset.

Otra característica de notable interés de la matriz fundamental radica en su capacidad de inferir la pose relativa (posición y orientación) entre los centros de

las cámaras que capturan las imágenes correspondientes. Este aspecto plantea una valioso nicho de investigación para desarrollar estrategias de Localización basada en Apariencia que estimen la pose de un agente a partir de las matrices fundamentales que lo relacionan con los elementos de la topología de lugares propuesta, tal y como se discute en el capítulo 7.

2.5. OpenCV

OpenCV [48] es una destacada librería de código abierto ampliamente reconocida en el ámbito de la visión por computadora. Esta librería ofrece una gran multitud de métodos y algoritmos para diversas tareas. Está disponible para la mayoría de sistemas operativos y puede utilizarse con distintos lenguajes de programación, entre ellos, Python, C++ y Java.

En el desarrollo de este proyecto, se emplea la librería OpenCV para el procesamiento de construcción del Grafo de Apariencia (GA). En concreto, se emplea en el procesamiento de detección de la co-visibilidad entre imágenes, como se explica en detalle en el capítulo 4. A continuación, se enumeran las herramientas más relevantes proporcionadas por esta biblioteca:

- ***SIFT* detector** [49]: Esta clase permite el cálculo de descriptores *SIFT* a partir de una imagen en escala de grises y cuenta con diversas funciones relativas a este descriptor local. En nuestro trabajo, utilizamos esta clase para obtener conjuntos de vectores descriptores de características ($\vec{d}$) y vectores de keypoints ($\vec{k}$) para una imagen dada.

- ***Brutal-Forcer Matcher*** [50]: Esta clase implementa un sistema simple de detección de correspondencias entre conjuntos de descriptores locales. El algoritmo compara un descriptor de características del primer conjunto con todos los descriptores del segundo conjunto utilizando algún cálculo de distancias y devuelve el más cercano. En nuestra propuesta, aplicamos el algoritmo Brutal-Force Matcher con el parámetro de cross-validation, lo que implica que solo se devuelven las parejas más similares entre todos los descriptores de los dos conjuntos. El resultado del algoritmo proporciona parejas de descriptores locales en dos imágenes, lo que nos sirve como un primer paso para establecer correspondencias geométricas entre keypoints.

- **Matriz Fundamental con *RANSAC*** [51]: OpenCV también implementa un algoritmo que permite el cálculo de la geometría epipolar entre dos imágenes, donde se devuelve la matriz fundamental resultante aplicada a un conjunto de correspondencias entre keypoints. Este procedimiento se explica en detalle

en la sección 2.4. Uno de los parámetros que toma esta función permite establecer el método de obtención de la matriz fundamental. En nuestro caso, aplicamos el método de *RANSAC*, que es un algoritmo iterativo que estima una matriz fundamental para un conjunto aleatorio de correspondencias en cada iteración. Se calcula el porcentaje de representación de cada matriz obtenida sobre el conjunto completo de correspondencias, asignando una puntuación en función del número de correspondencias representadas. Se devuelve la matriz fundamental que represente el mayor número de correspondencias, es decir, aquella que mejor se ajuste a la mayoría. Este algoritmo para el cálculo de la matriz fundamental resulta fundamental en el procedimiento de detección de co-visibilidad, como se explica en el capítulo 4.

2.6. Conclusiones del capítulo

En este capítulo se ha llevado a cabo una exhaustiva revisión del Estado del Arte que sustenta este trabajo, así como de las herramientas empleadas en el mismo.

A lo largo de la presente memoria, se hará referencia a este capítulo para obtener una comprensión más profunda de los elementos involucrados en la construcción de la topología de lugares propuesta.

Parte II

Desarrollo del proyecto

Capítulo 3

Topología de lugares sobre Mapas de Apariencia

Contenido

Sinopsis

En este capítulo se expone la estrategia general propuesta en este trabajo para construir la topología de lugares. Se presenta una definición formal del Mapa de Apariencia (AM), introduciendo el concepto de variedad de apariencia como una estrategia para inferir información cinemática basada en la apariencia.

A continuación, se describe el Grafo de Apariencia (GA) como una transformación de los elementos del AM, y se introduce la técnica de Corte Normalizado para realizar una bipartición recursiva del GA, lo cual resulta en la generación de una topología de lugares. Por último, se describe una estrategia de Localización Topométrica que opera en la mencionada topología de lugares, proporcionando un contexto aplicativo para su utilización.

3.1. Planteamiento: Mapas de Apariencia

En este trabajo, se ha referido el concepto de Mapa de Apariencia (AM) como un conjunto de imágenes geolocalizadas de un entorno previamente visitado. Se propone una formulación donde se define $AM = \{(\vec{x_i}, \vec{d_i})|i = 0 \dots m\}$ como un conjunto de m pares compuestos por un descriptor global [7], $\vec{d_i} \in \mathbb{R}^d$, y la pose de la cámara, $\vec{xi} \in SE(2)$[47], desde la cual se capturó la imagen.

Este trabajo se inspira en trabajos previo [52] [23] donde la apariencia de todas las imágenes del entorno se modela en una variedad o *manifold* de apariencia, denominada D. Dicha variedad se refiere a una estructura topológica [53] de dimensión n en un espacio topológico abstracto localmente euclidiano. En nuestra definición de AM propuesta, el *manifold* de apariencia se define a partir del conjunto de todos los vectores descriptores globales, $\vec{di} \in \mathbb{R}^d$, donde cada descriptor representa un punto en la superficie $D = \{(\vec{di})\}$, tal que que $D \in \mathbb{R}^d$.

Este enfoque plantea que la geometría intrínseca en D está determinada por un conjunto de parámetros de articulación con un número limitado de grados de libertad, los cuales determinan la información cinemática que se muestra en el conjunto de imágenes [53] [54] [55]. En este sentido, se pueden inferir diferentes perspectivas o momentos en una escena dada a partir de la geometría de D, es decir, a partir de la apariencia. Sin embargo, trabajar con la apariencia presenta diversas limitaciones.

Se han propuesto enfoques para inferir la pose de la cámara a partir de la geometría intrínseca del *manifold* [52] [56] [57]. No obstante, en la mayoría de las aplicaciones reales, la relación entre la apariencia y la pose es altamente no lineal como se defiende en [58], lo que dificulta la recuperación de la geometría intrínseca de D.

Figura 3.1: Ejemplo ilustrativo del comportamiento de la apariencia a lo largo de un pasillo. La apariencia es dependiente de la pose, y se mantiene estable en el pasillo, el cual posee una estructura uniforme. Sin embargo, experimenta variaciones significativas al transicionar entre habitaciones debido a las oclusiones provocadas por la proximidad de objetos como puertas y paredes. [10]

Además, las condiciones variables del entorno dificultan la relación entre la pose y la apariencia. Los cambios en la iluminación y la perspectiva no representan un problema, ya que los descriptores utilizados para construir D están diseñados para ser invariantes a este tipo de cambios. Sin embargo, como se ilustra en la Figura 3.1, la propia estructura del entorno puede derivar en una relación deficiente entre la apariencia y la pose debido principalmente a la presencia de oclusiones y al fenómeno del aliasing perceptual o *Perceptual Aliasing* (*PA*).

Las oclusiones se producen cuando hay elementos que bloquean la visión de otras áreas del entorno (como las puertas en la Figura 3.1), lo cual resulta en cambios abruptos en la apariencia sin una variación significativa en la pose. Esto se traduce en bordes afilados sobre la variedad D que introducen discontinuidades en la superficie.

Por otro lado, el *Perceptual Aliasing* (*PA*) [34] ocurre cuando existen áreas del entorno que son visualmente similares pero no están cercanas en términos de pose, lo que da lugar a intersecciones sobre la variedad D. Este fenómeno representa una gran dificultad en la mayoría de las aplicaciones de *AbL*, ya que tiende a asignar como similares imágenes que se alejan significativamente del ground-truth de la pose de consulta, lo que reduce la eficiencia de la localización.

Figura 3.2: Ejemplo de *Perceptual Aliasing* (*PA*) representado por imágenes de dos pasillos diferentes. Se puede apreciar que la apariencia general es muy similar entre las imágenes, a pesar de tratarse de imagenes distantes en términos de pose.

En conclusión, modelar la apariencia como un *manifold* resulta ser una herramienta útil para inferir información cinemática a partir de una secuencia de imágenes. No obstante, plantea un desafío matemático complejo debido a que la relación entre la pose y la apariencia no es lineal en aplicaciones realistas, a lo que se suman los fenómenos de oclusión y *PA*.

En nuestra propuesta, se plantea una topología de lugares para segmentar distin-

tas regiones en sobre el *manifold* (D) a fin de obtener lugares similares en apariencia. Estos lugares se componen de conjuntos de elementos del Mapa de Apariencia, clasificados de manera que se minimicen los efectos negativos derivados de los fenómenos mencionados.

3.2. Descripción de la topología de lugares

Nuestra propuesta surge como respuesta a la considerable complejidad asociada al problema de la localización en una variedad de apariencia D de alta dimensionalidad, específicamente centrándose en el problema del *Perceptual Aliasing PA*. Presentamos una topología para crear grupos de elementos del Mapa de Apariencia (AM) mediante la división de D en diferentes regiones, utilizando un criterio de similaridad y co-visibilidad.

Partimos de la definición del AM como un conjunto de elementos que consta de un descriptor global ($\vec{d}$) y una pose ($\vec{x}$). Introducimos el concepto de Grafo de Apariencia (GA) como un grafo ponderado no direccional (ver Figura 3.3). Se define $GA = \{V, E, W\}$ donde V es el conjunto de nodos, E es el conjunto de aristas o arcos entre nodos y W es la matriz de adyacencia ($|V|$x$|V|$) cuadrada y simétrica. Cada nodo en V se corresponde con un elemento del AM, es decir, $V = AM = \{(\vec{xi}, \vec{di})|i = 0...m\}$. El peso de las aristas E que conectan los nodos se determina en función de la similitud y la co-visibilidad entre dichos elementos. Como se detalla en la sección 4, la co-visibilidad es una buena métrica para inferir el solapamiento de los conos de visión de las cámaras dadas dos imágenes y es fundamental para evitar el PA en la agrupación. La matriz W almacena la relación entre los nodos del grafo, donde el elemento w_{ij} es el peso del arco que conecta los nodos i y j.

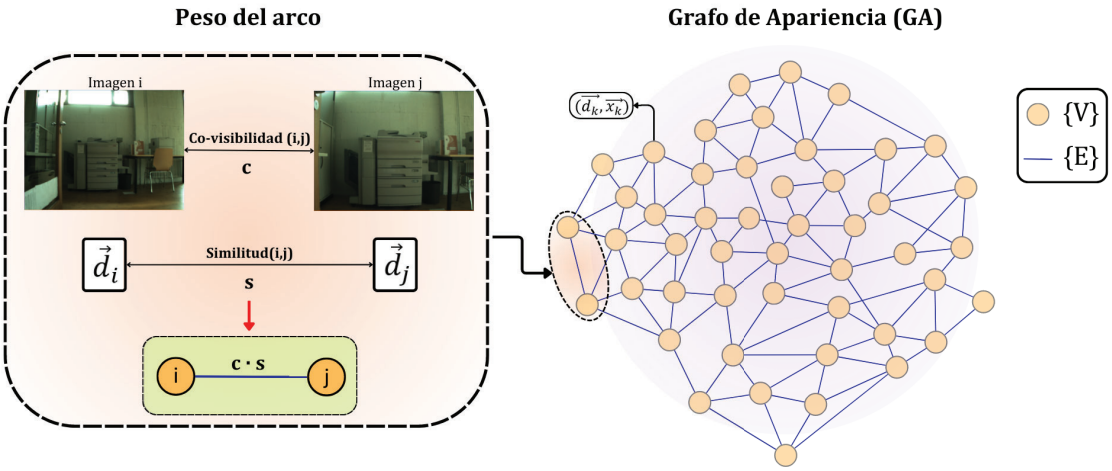


Figura 3.3: Representación del Grafo de Apariencia (GA) donde se ilustra el cálculo del peso de arco entre nodos.

En este contexto, definir una topología de lugares para de agrupar los elementos del AM se convierte en un problema de corte de grafo. En nuestra propuesta, utilizamos el Corte Normalizado de Grafo (*Normalized Cut Graph*) [59], una herramienta global utilizada para segmentar grafos. El objetivo de este criterio es dividir el grafo en dos grupos disjuntos, A y B, maximizando la disimilitud total entre los grupos y la similitud total dentro de los grupos. Este enfoque se basa en la bipartición espectral de grafos [60], que parte del estudio de los autovectores de la matriz de adyacencia para subdividir el grafo de manera que el valor del corte sea mínimo. Estos conceptos se explican en la sección 5.

La topología de lugares se obtiene al aplicar la bipartición recursiva al GA. El resultado es un conjunto de subgrafos que representan conjuntos de elementos del AM tal que $GA = \{Cj | j = 0 \ldots n\}$, donde n es el número de grupos obtenidos y $Cj = \{(\vec{x}, \vec{d})\}$ formado por elementos del AM similares y co-visibles. Por lo tanto, se obtiene un mapa topológico representado por el Grafo de Apariencia particionado (ver Figura 3.4).

Una de las ventajas de este método radica en que el número de grupos obtenidos depende exclusivamente de los parámetros de configuración del método y las características visuales del entorno, lo que supone una mejora contra métodos de agrupación basados en k-means [22] [23] que requieren de un número predeterminado de grupos. Además, es compatible con secuencias de datos desordenados.

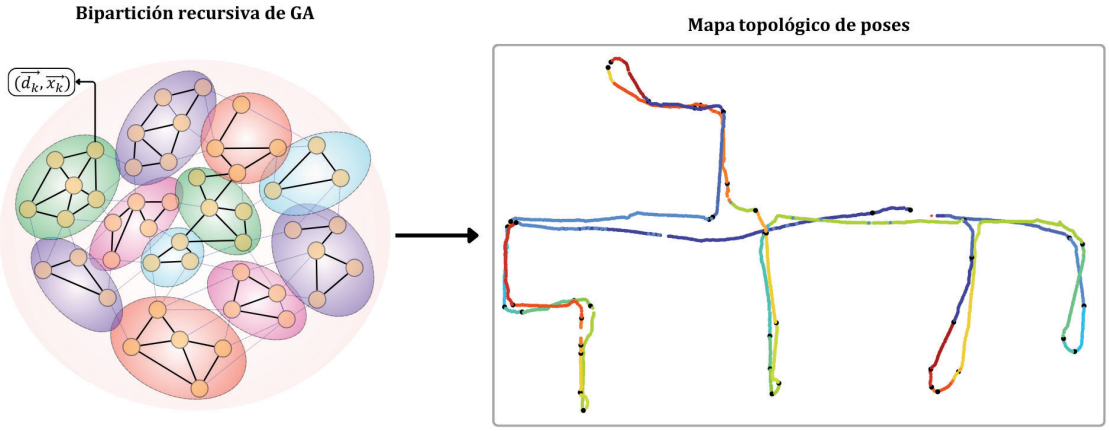


Figura 3.4: Bipartición recursiva del Grafo de Apariencia, el resultado es un conjunto de subgrafos donde cada uno representa un lugar del mapa topológico.

Es importante destacar que, aunque se dispone de las poses de los elementos del AM, el criterio para la agrupación se basa exclusivamente en las características visuales de las imágenes. El objetivo de adoptar este enfoque es minimizar la agrupación errónea de elementos en términos de pose, basándonos únicamente en la información visual del entorno.

En anteriores aproximaciones de este método, se combinan la similitud entre descriptores y la distancia entre poses para conformar el *GA* y aplicar el Corte Normalizado. Esto resultaba en una clasificación robusta frente al problema del *Perceptual Aliasing*, pero con un rendimiento deficiente en la regresión de la pose. Esto se debe a que, como se discute en la sección anterior, la distancia entre poses y la similitud en apariencia son dos magnitudes con una relación no lineal y dificiles de combinar.

La idea principal que defiende este trabajo es que la mejor manera de agrupar los elementos del Mapa de Apariencia, con el fin de minimizar el error de localización, en base a la estructura natural del entorno y considerar sus posibles irregularidades en apariencia.

3.3. Contexto: Localización Topométrica

Esta topología de lugares surge como base para un método de Localización Topométrica dentro del marco de la Localización basada en Apariencia (*AbL*). Este sistema cuenta con dos módulos principales. El primer módulo es un sistema de VPR que infiere un lugar o regiones en D. El segundo módulo, denominado módulo de Localización, utiliza información de las poses de los elementos dentro de la región para mejorar la precisión al inferir una pose de consulta. Ambos módulos operan de

manera secuencial (ver Figura 3.5) sobre un mapa topológico común representado por el Grafo de Apariencia. Esto permite obtener lo mejor de ambos modelos de localización, donde primero se estima un lugar o región el *GA* y posteriormente se refina un pose a partir de estos elementos específicos.

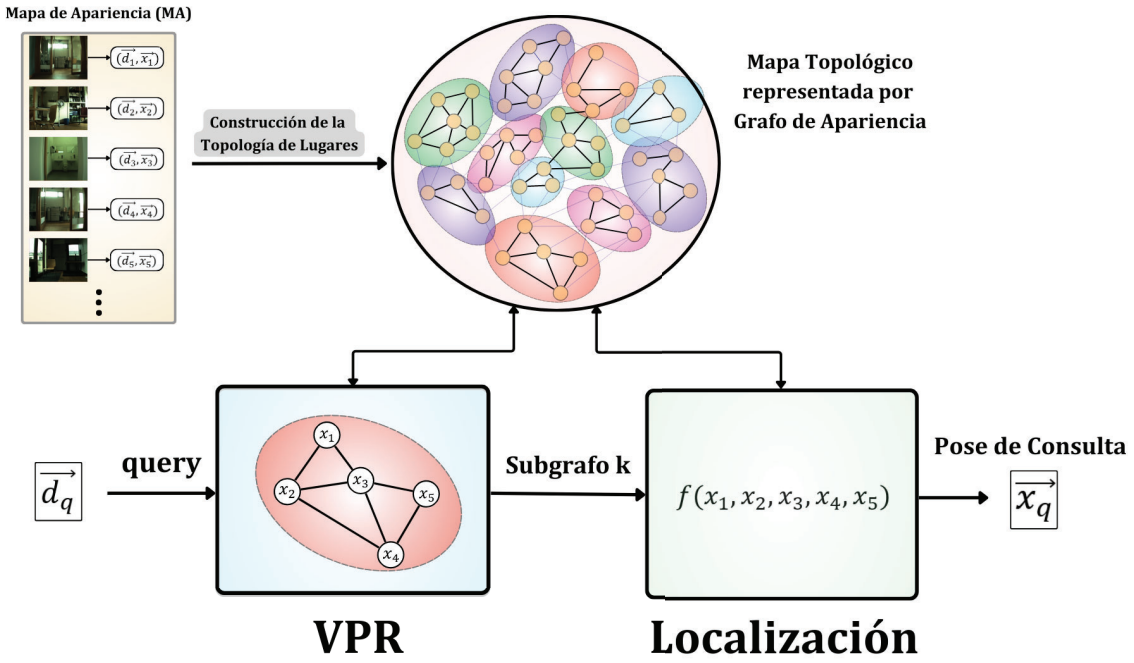


Figura 3.5: Esquema del sistema de Localización Topométrica, los módulos operan de manera secuencial sobre un mapa topológico comun representado por el Grafo de Apariencia. En el sistema de *VPR* se infiere un lugar (subgrafo k) a partir del descriptor de consulta $\vec{d}_q$. Luego, el sistema de Localización infiere la pose de consulta $\vec{x_q}$ a partir de las poses del subgrafo.

3.4. Conclusiones del capítulo

En este capítulo se presenta una estrategia para crear una topología de lugares que permita la clasificación de los elementos del Mapa de Apariencia (AM). En primer lugar, se describe el concepto de *manifold* de apariencia como estrategia para inferir información espacial a partir de la apariencia, destacando la complejidad matemática y analizando los problemas que surgen dada la baja relación entre la apariencia y la pose en el problema de localización.

Se propone el Grafo de Apariencia (GA) como una representación del AM basado en la similitud y co-visibilidad de sus elementos. Para definir la topología, se

presenta el Corte Normalizado de Grafo, una herramienta que permite dividir el grafo en regiones disjuntas. Este enfoque garantiza que las regiones resultantes estén libres de *Perceptual Aliasing* (*PA*), un fenómeno que puede introducir errores en la localización al asociar erróneamente imágenes visualmente similares pero lejanas en términos de pose.

Por consiguiente, nuestro método presenta una serie de ventajas:

- Las regiones resultantes de nuestra topología son localmente euclideas en D, por lo que se pueden realizar operaciones euclideas con la apariencia de estas regiones. Además dichas regiones están exentas de *Perceptual Aliasing* (*PA*) dada la relación geométrica entre elementos.

- La herramienta de Corte de Grafo permite la agrupación de elementos desordenados en su origen. Además el número de clusters obtenidos depende exclusivamente de las características visuales del mapa, no necesita asignación previa. Esto hace al método facilmente adaptable a todo tipo de escenarios, pudiendo incluso ampliar o modificar un mapa existente.

- Esta topología de lugares sienta la base para lograr una localización métrica eficiente en el contexto del modelo de Localización Topométrica descrito. Dado que las regiones están exentas de *PA*, se asume que dentro de cada región, las imágenes similaries en apariencia también son cercanas en términos de pose.

Capítulo 4

Construcción del Grafo de Apariencia

Contenido

Sinopsis

En este capítulo se realiza una definición formal del Grafo de Apariencia (GA), estableciendo las características que deben cumplirse para aplicar el método de Corte Normalizado de grafo. Se hace hincapié en la obtención de las matrices de similitud y co-visibilidad, proporcionando detalles sobre su fundamentación teórica e implementación práctica. Por último, se presentan los resultados obtenidos al calcular estas matrices sobre un Mapa de Apariencia específico.

4.1. Introducción

Previamente, hemos introducido el concepto de Grafo de Apariencia (GA) como un grafo ponderado no dirigido. Un grafo ponderado es aquel en el cual las aristas que conectan los nodos poseen un valor de peso asociado [61]. Al ser no dirigido, implica que las aristas no tienen una dirección definida, es decir, representan relaciones simétricas [62].

Formalmente, definimos $GA = \{V, E, W\}$ donde V es el conjunto de nodos del grafo, y cada nodo es una tupla que contiene un descriptor y una pose. Por lo tanto, $V = \{(\vec{xi}, \vec{di}) | i = 0 \dots m\}$. El conjunto E representa las aristas ponderadas que conectan los nodos. En nuestra propuesta, el peso de las aristas se encuentra acotado en el rango [0,1], donde 0 indica la ausencia de conexión entre nodos y 1 indica una conexión máxima. Por último, W es la matriz de adyacencia del grafo, la cual es cuadrada ($|V| \text{x} |V|$) y simétrica debido a que el grafo es no direccional. Cada elemento w_{ij} de la matriz W corresponde al peso de la arista que une los nodos i y j. Como se mencionó previamente, este valor está acotado en el rango [0,1].

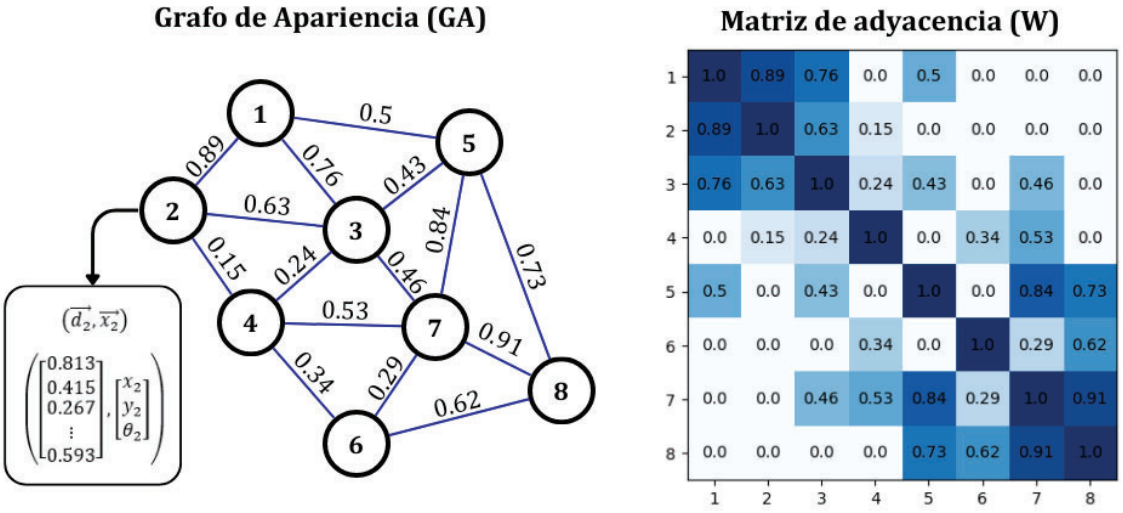


Figura 4.1: Ejemplo de Grafo de Apariencia (GA) con matriz de adyacencia.

Para poder aplicar el método de bipartición espectral de grafos, como se explica en el capítulo 5, el GA debe cumplir con ciertas características [63]. En primer lugar, el grafo debe ser no dirigido y ponderado, con valores reales. Además, los elementos de la diagonal principal deben tener el valor máximo, que en nuestro caso es 1. Por otro lado, el GA debe ser un grafo conexo, lo que significa que existe un camino que une cualquier par de nodos en el grafo. Se entiende como camino a una secuencia de nodos conectados por aristas. Esto implica que no puede haber nodos aislados ni

subgrafos no conexos dentro del *GA*.

Como se ha mencionado anteriormente, el peso que relaciona dos nodos se basa en las características visuales de las imágenes que representan, específicamente la similitud y la co-visibilidad. La similitud es un criterio clásico en *Visual Place Recognition* (*VPR)* que se utiliza para comparar imágenes en función de sus descriptores globales, permitiendo el emparejamiento o matching de imágenes para un problema de *Image Retrieval*.

La co-visibilidad entre imágenes relaciona geométricamente las características locales de las imágenes, a fin de evitar el fenómeno del *Perceptual Aliasing* (*PA*). Se estudia la posición de los descriptores locales en cada imagen y se busca una relación basada en la geometría epipolar para determinar si las imágenes presentan solape en el cono de visión de las cámaras, es decir, obserban una escena común.

Se evalúan la similaridad y co-visibilidad para cada par de elementos en V. Esto resulta en las matrices de similitud, S, y co-visibilidad, C, donde cada elemento $s(i,j)$, $c(i,j)$ representan la similitud y covisibilidad entre los elementos i y j respectivamente. El GA se construye calculando la matriz de adyacencia W como la multiplicación elemento a elementos de las matrices S y C:

$$w(i,j) = c(i,j) \cdot s(i,j)$$

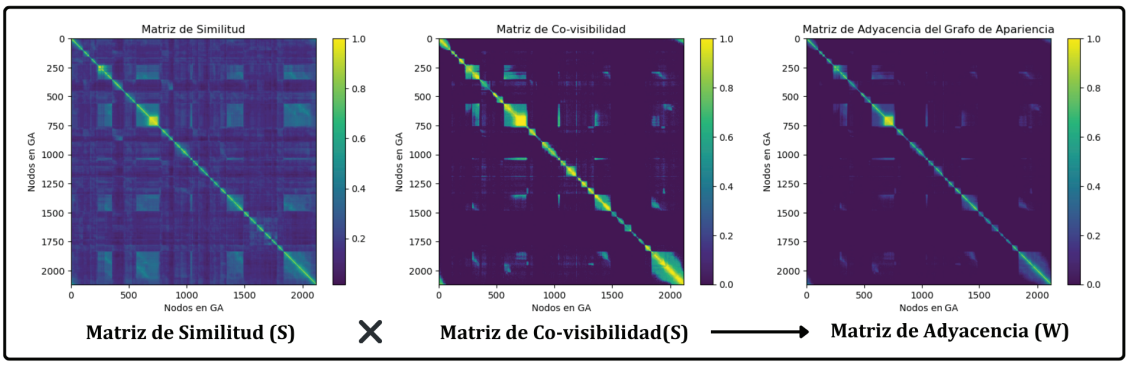


Figura 4.2: Ejemplo de matrices de co-visibilidad (C, similitud$S$ y adyacencia W.

4.2. Matriz de Similitud

La similitud entre pares de imágenes es un criterio ampliamente utilizado en *Image Retrieval* [7] [33], que permite el emparejado o clasificación de imágenes en base a su apariencia visual. En este contexto, la apariencia de una imágen se representa mediante un descriptor global numérico, como se detalla en la sección 2.2. Se asume que dos apariencias similares se codifican en vectores numéricamente parecidos, es decir puntos próximos en el espacio vectorial del descriptor. En este sentido, la similitud entre imágenes se evalúa como una distancia entre sus descriptores globales.

Existen diversas métricas para medir la distancia entre vectores, como la distancia euclidiana o la distancia de Manhattan. En nuestro caso, la métrica que ha demostrado ofrecer los mejores resultados es la similitud del coseno [64]. En un espacio vectorial donde se define el producto escalar, esta métrica evalúa la distancia angular entre dos vectores, basándose en el valor del coseno del ángulo que forman.

$$\cos(\theta) = \frac{\langle A, B \rangle}{||A|| \cdot ||B||} \tag{4.1}$$

Para modelar la apariencia de las imágenes, en este trabajo se emplea el descriptor global *NetVLAD*, como se define en la sección2.2. Como se indica en el artículo original [7], el espacio vectorial *NetVLAD* es positivo normalizado, lo que implica que la norma de todos los vectores sea igual a 1 y que todos los elementos de los vectores sean no negativos. En este espacio, los valores que puede tomar la similaridad del coseno quedan acotados en el rango [0,1], donde 0 representa dos vectores ortogonales (totalmente diferentes) y 1 indica dos vectores superpuestos (idénticos).

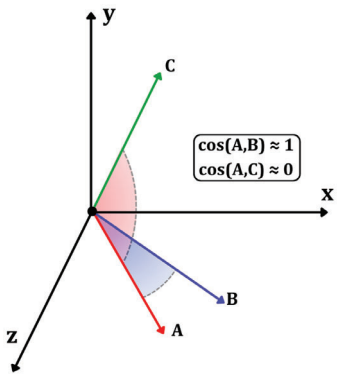


Figura 4.3: Ejemplo simplificado del espacio vectorial del descriptor global con $\mathbb{R}^3$. Se ilustra dos ejemplos de la similaridad del coseno para vectores similares (A,B) y vectores no similares (A,C).

Para cada par de nodos en V, se evalúa la similitud del coseno entre sus descriptores globales. El resultado se organiza en la matriz de similitud S (ver Figura 4.4), donde cada componente *s(i,j)* es el resultado de la ecuación 4.1 entre los descriptores i y j.

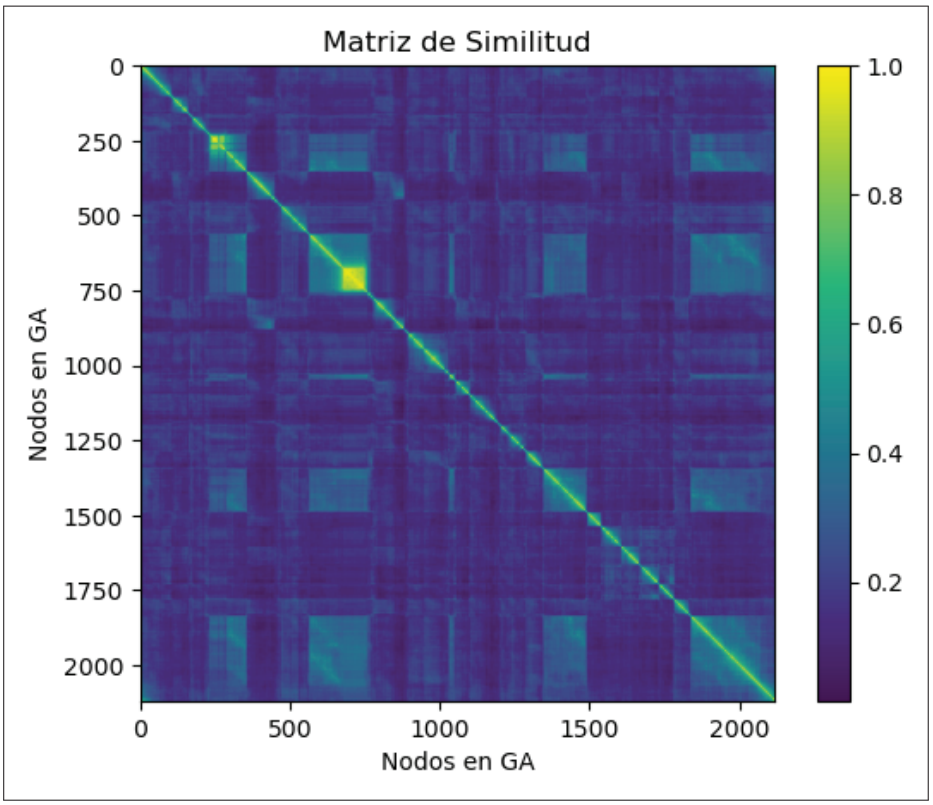


Figura 4.4: Matriz de similitud obtenida para $AM = \{Seq2\_cloudy1\}$. (ver sección 6.1)

Incorporar esta información durante la creación de la topología es fundamental para el proceso de *VPR*, ya que los grupos resultantes son similares en términos de sus vectores descriptores. Esto permite caracterizar al conjunto de forma sencilla a partir del promedio de sus descriptores, lo que a su vez simplifica la asignación de una consulta a un grupo. En la sección de resultados de este capítulo analizamos la matriz de similitud de la Figura 4.4.

4.3. Matriz de Co-visibilidad

4.3.1. Definición

La co-visibilidad entre imágenes es un concepto introducido en el presente *TFG* que define la situación en la cual un par de imágenes representan total o parcialmente una misma escena. Un ejemplo ilustrativo se muestra en la Figura 4.5, donde se observa la misma escena desde distintas perspectivas. En otras disciplinas, como la fotometría, este concepto es conocido como *solapamiento visual*. Esta noción puede ser entendida como el área de solapamiento formada entre los conos de visión de las dos cámaras y el fondo de la escena, y puede ser cuantificada en términos de píxeles o porcentaje de solapamiento.

Previamente, hemos mencionado la co-visibilidad como una métrica de la relación geométrica entre las características locales en dos imágenes. Esta técnica se fundamenta en la geometría epipolar [43], la cual se describe en detalle en el capítulo 2.4. La geometría epipolar establece una relación entre los puntos de interés en dos imágenes que se supone representan una misma escena. Por lo general, estos puntos de interés se obtienen a partir de descriptores locales de la imagen (consultar sección 2.2). La co-visibilidad entre las imágenes se establece al comprobar la existencia de esta relación.

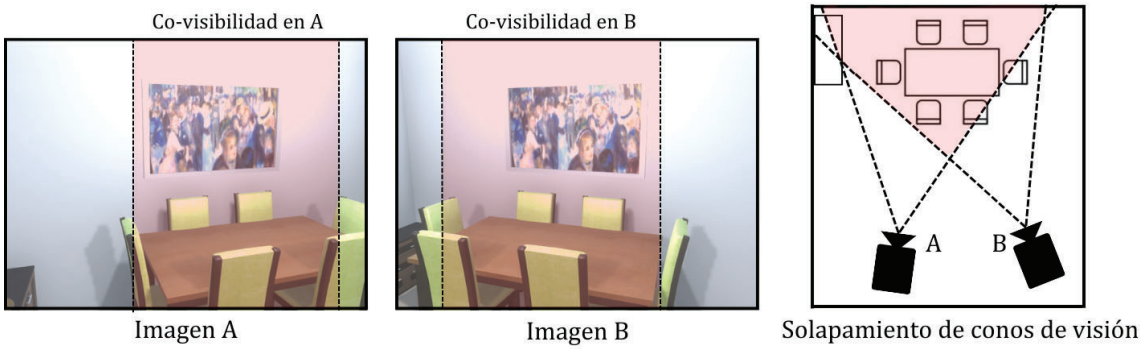


Figura 4.5: Ejemplo de imágenes co-visibiles con área de solapamiento entre conos de visión. Se observa que una porción de la imagen A está presente en la imagen B y viceversa, el porcentaje de co-visibilidad no suele ser simétrico. Imágenes originales de Robot@VirtualHome [65].

Como se detalla posteriormente en este capítulo, se mide la co-visibilidad para cada par de imágenes resultando en un valor dentro del rango [0,1] que representa el porcentaje de solapamiento entre las imágenes.Un valor de 0 indica que las imágenes no son co-visibles, mientras que un valor de 1 indica que las imágenes están completamente solapadas. El resultado se almacena en la matriz de co-visibilidad

(ver Figura 4.6). Por restricciones de diseño la matriz debe ser simétrica, por lo que se toma como valor de co-visibilidad el porcentaje de solapamiento mayor entre las dos imágenes.

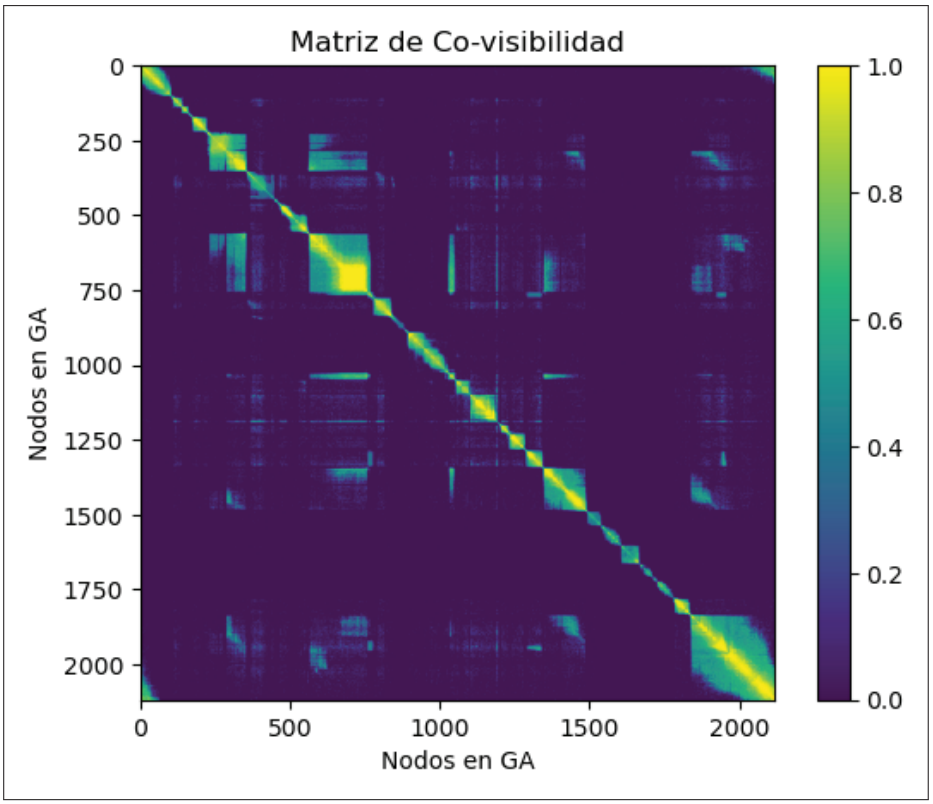


Figura 4.6: Ejemplo de matriz de co-visibilidad resultante para $AM = \{Seq2\_cloudy1\}$. (ver sección 6.1)

La co-visibilidad nace como una solución al problema del *Perceptual Aliasing* (*PA*). Los descriptores globales presentan la desventaja de perder la información geométrica de la estructura del entorno y la distribución de las características locales en la escena al codificar las imágenes en un solo vector, lo cual contribuye a la aparición del *PA*. Este fenómeno se da cuando dos imágenes son visualmente similares pero no son cercanas en cuanto a su pose [34]. En términos de vectores descriptores, el *PA* se da cuando dos vectores son cercanos en el espacio vectorial del descriptor, pero corresponden a poses distintas.

Figura 4.7: Ejemplo de *Perceptual Aliasing* (*PA*) representado por imágenes de dos pasillos diferentes. A simple vista, la apariencia general de las imágenes es muy similar, por lo que es necesario examinar los detalles locales para verificar que no corresponden al mismo lugar.

Como ejemplo ilustrativo de *Perceptual Aliasing*, se considera la situación mostrada en la imagen 4.7, donde las dos imágenes parecen haber sido capturadas desde el mismo lugar. En términos generales, las apariencias son realmente similares; sin embargo, al examinar los detalles de la escena (características locales), es posible determinar que estas imágenes no pertenecen al mismo lugar.

En nuestra propuesta, defendemos que al recuperar la información geométrica que relaciona las imágenes por medio de la co-viibilidad, es posible evitar el problema del *PA*.

4.3.2. Implementación

La estrategia propuesta para determinar la co-visibilidad entre imágenes se fundamenta en el uso de la geometría epipolar [44] [43], una disciplina ampliamente utilizada en la visión estéreo y la creación de imágenes panorámicas (*image stitching*) [46]. En nuestra propuesta, la relevancia de esta herramienta radica en que presupone que las imágenes relacionadas representan una escena común. Por lo tanto, se puede inferir que dos imágenes son co-visibles si es posible establecer una relación geométrica epipolar consistente entre ellas.

La geometría epipolar describe las relaciones geométricas entre los puntos tridimensionales de una escena y sus proyecciones bidimensionales en los dos planos de imagen, conocidas como características (*features*). Estas *features* suelen ser representadas por un conjunto de puntos clave (*keypoints*), que son ubicaciones de

interés en el plano de imagen, junto con un conjunto de vectores descriptores locales que representan cada *keypoint* (véase la sección 2.2). La matriz fundamental [43] es la representación algebraica de la geometría epipolar (consultar la sección 2.4) y relaciona los pares de *keypoints* en ambas imágenes. Los pares de puntos clave representados por la matriz fundamental cumplen con la restricción epipolar, que establece una relación geométrica entre ellos. La detección de co-visibilidad se basa en encontrar una matriz fundamental que sea consistente para la mayoría de las *keypoints*. Estos conceptos, junto con otros que se emplean en esta sección (*BFM* [49], *RANSAC* [66]), se explican detalladamente en el capítulo 2.

A continuación, se describe el proceso de detección de la co-visibilidad, en el Anexo A se muestra el código que implementa este proceso. En esencia, se trata de un problema de correspondencia entre *features*, en el cual se eliminan los *outliers* o correspondencias incorrectas, con el fin de identificar la co-visibilidad. La explicación se ilustra con imágenes originales de COLD dataset [67] (ver sección 6.1). Es importante destacar que todos los elementos de la diagonal principal de la matriz de co-visibilidad toman el valor de 1, ya que una imagen es completamente co-visible consigo misma.

- Paso 1: Extracción de descritores locales SIFT [25] de las imágenes en crudo utilizando el método proporcionado por OpenCV [49]. Esto produce dos conjuntos de características: $f1 = \{k1, d1\}$ y $f2 = \{k2, d2\}$, donde *k1, k2* representan conjuntos de *keypoints* en cada imágen y *d1, d2* son los vectores descriptores de las *features*.

Figura 4.8: *Keypoints* sobre las imágenes originales, marcados en azul

- Paso 2: Se establece la correspondencia entre las características mediante la aplicación del método *Brutal-force Matcher* (*BFM*) [50]. Esta técnica evalúa

la similtud entre los descriptores *d1* y *d2* y establece un conjunto de parejas de *features* o *matches*. Si el número de *matches* no supera un cierto umbral, se determina que no existe co-visibilidad.

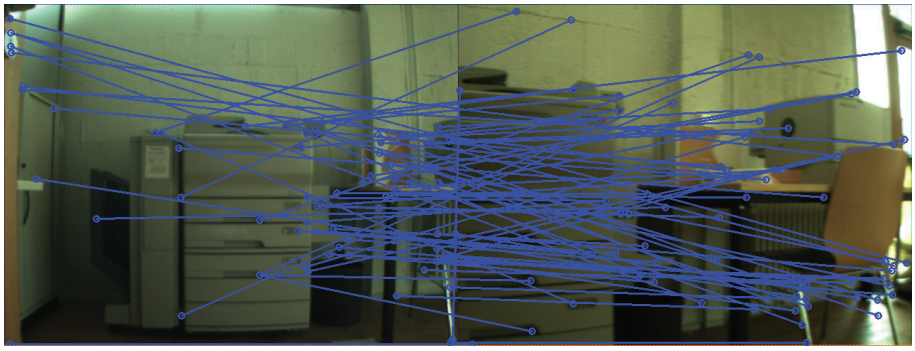
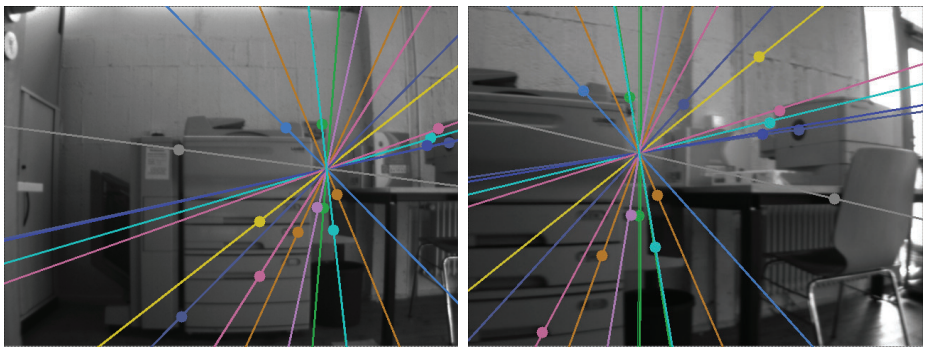


Figura 4.9: *Matching* de *keypoints* con *Brutal-forcer Matcher*. Se representa cada parejas unidas por una línea azul.

- Paso 3: Filtado de *outliers* generados por *BFM*. Se calcula la matriz fundamental utilizando el método proporcionado por OpenCV [51]], aplicando el algoritmo *RANSAC*. *RANSAC* (*Random Sample Consensus*) [66] es un método iterativo que estima una matriz fundamental en cada iteración para un subconjunto aleatorio de los datos de entrada (pares de *keypoints*). *RANSAC* devuelve el modelo que mejor representa a la mayoría de los datos.



a b

Figura 4.10: Representación de la correspondencia epipolar dada por la matriz fundamental. Se muestran por colores los pares de *keypoints* con sus respectivas lineas epipolares. La interesección de todas las líneas nos muestra la ubicación del epipolo, la poryección de la pose de la cámara en la imagen a sobre el plano imagen b. (ver capítulo 2.4

- Paso 4: Determinación de la covisibilidad. Los matches que estén representadas por la matriz fundamental cumplen con la restricción epipolar y se consideran inliers, mientras que los matches que no son representadas por la matriz se consideran outliers. Si el número de inliers supera un cierto umbral μ, se concluye que existe co-visibilidad entre las imágenes y se procede a calcular el porcentaje de solapamiento. La determinación de este umbral μ se realiza de forma heurística, como se demuestra en la sección 6.2.1 para un resultado óptimo este umbral debe pertenecer al rango [15,30].

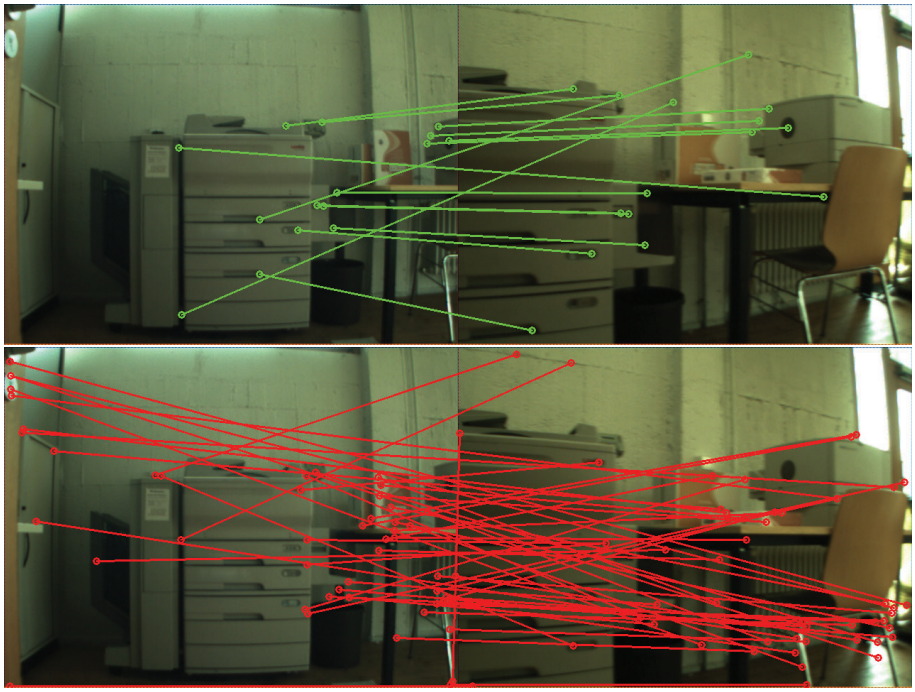
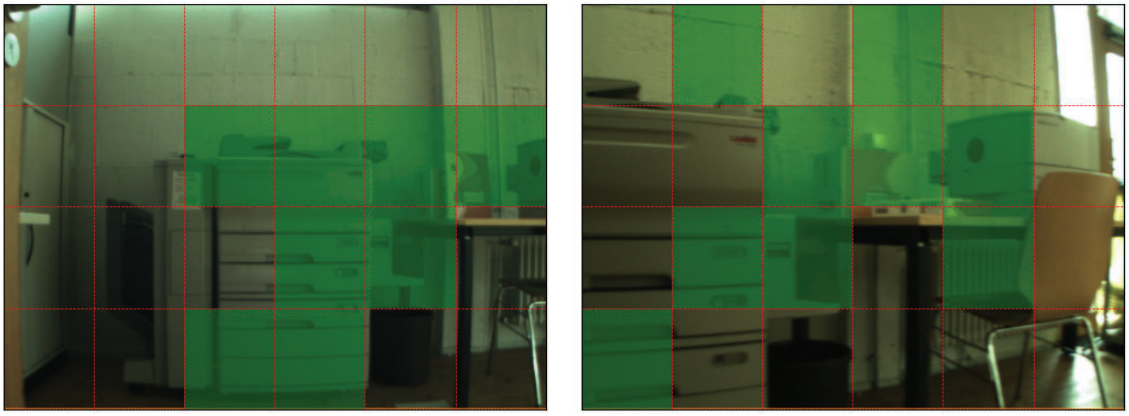


Figura 4.11: Representación del filtrado de *outliers* con la matriz fundamental para $\mu = 20$. En verde se representan los *inliers*, parejas representadas por la matriz fundamental. En rojo se representan los *outliers*.

- Paso 5: Determinar el porcentaje de solapamiento en cada imagen. Si se determina que las imágenes son co-visibles, se aplica una celda de ocupación marcando como ocupadas aquellas celdas que contienen uno o más inliers. El área de ocupación se calcula dividiendo el número de celdas ocupadas por el número total de celdas, lo que proporciona un resultado en el rango de [0, 1].



a b

Figura 4.12: Celda de ocupación sobre las imágenes originales, se marcan en verde las celdas ocupadas.

Con el fin de obtener una medida simétrica para ambas imágenes, se establece el valor de co-visibilidad como el área de ocupación más grande entre las imágenes. En consecuencia $co - visibilidad(i, j) = co - visibilidad(j, i) = c(i, j)$, dando lugar a una matriz de similaridad simétrica (ver Figura 4.2). En la Figura 4.13 se ilustra el proceso descrito para un ejemplo donde no existe covisibilidad.

Figura 4.13: Pasos para determinar co-visibilidad en un caso práctico donde no se da la co-visibilidad.

La principal desventaja de este método radica en el elevado tiempo de ejecución requerido. El análisis del tiempo de ejecución del código presente en el Anexo A arroja un orden de ejecución de $O(n^2)$, donde n representa el número de imágenes de entrada. Para el dataset *COLD Seq2_ cloudy1* [67], que cuenta con más de 2.000 imágenes, el tiempo de ejecución aumenta a varias horas para calcular la matriz de co-visibilidad. En el capítulo 7 se propone como objetivo de líneas futuras conseguir la optimización de este algoritmo.

4.4. Resultados

En esta sección se presenta el resultado de la construcción del Grafo de Apariencia (GA) siguiendo el procedimiento descrito en este capítulo. Se lleva a cabo un análisis de las matrices de co-visibilidad y similitud que forman la matriz de adya-

cencia del (GA). Estas matrices se construyen a partir del recorrido *Seq2_ cloudy1* del dataset *COLD* dataset [67] (ver sección 6.1), el cual representa el Mapa de Apariencia (AM) sobre el que se construye el GA. Consiste en un conjunto de imágenes geolocalizadas que recorren una trayectoria sobre un mapa real, en consecuencia las imágenes están ordenadas siguiendo el recorrido. Esto facilita la comprensión visual de las matrices resultantes. No obstante, como se abordará más adelante, este la topología de lugares que se presenta en este trabajo también es válida para secuencias de imágenes desordenadas.

El proceso de construcción del GA se describe en el código presente en el Anexo A, donde se muestra la construcción de las matrices de similaridad y co-visibilidad. En las Figuras que se ilustran en esta sección se presentan las matrices resultantes de este código aplicado al AM descrito. Este contiene un total de 2119, lo que determina la dimensión de las matrices (2119x2119). En las matrices presentadas, cada fila y columna representan un elemento del AM, asignando correspondencias de similitud o co-visibilidad entre ellos.

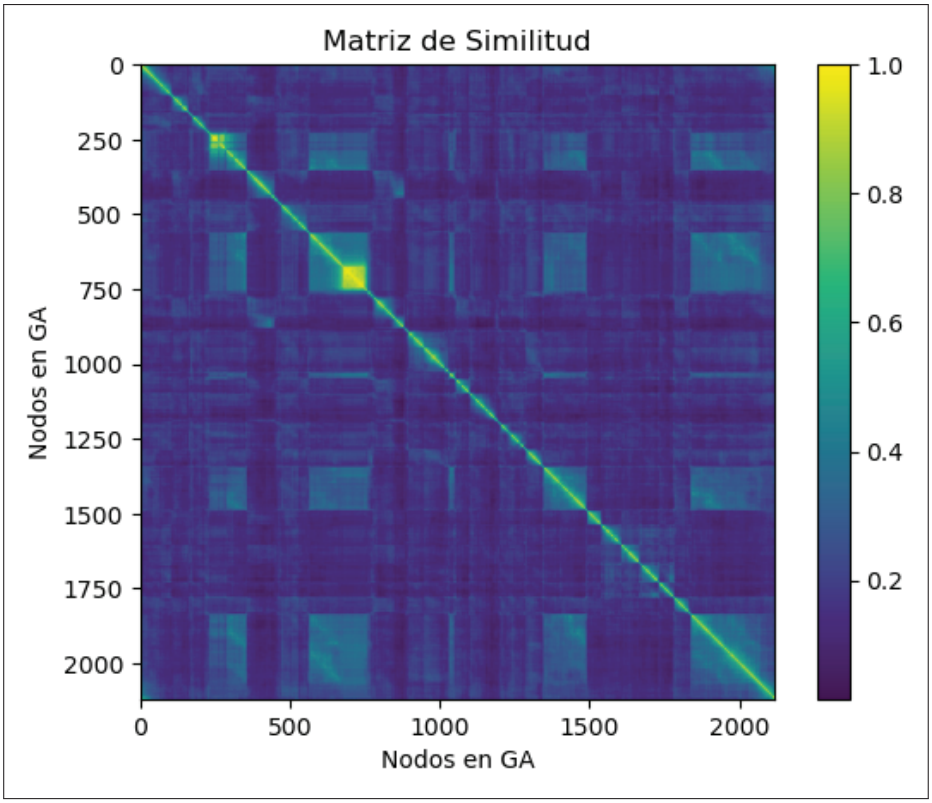


Figura 4.14: Matriz de similitud obtenida para $AM = \{Seq2\_cloudy1\}$.

En la Figura 4.14 se muestra la matriz de similitud, donde los valores de mayor correspondencia se encuentran cerca de la diagonal principal. Esto es correcto, ya

que se trata de un AM ordenado en el que se visitan los distintos lugares de forma secuencial. En un caso ideal para este dataset ordenado, solo se encontrarían similitudes en grupos cercanos a la diagonal principal, ya que los elementos alejados de esta no se corresponden con elementos cercanos en términos de pose.

Sin embargo, se encuentra correspondencia entre un gran número de elementos del AM que no pertenecen a la misma región, ya que se observan valores destacados en toda la matriz, lejos de la diagonal principal. En las correlaciones erróneas entre elementos se puede apreciar el fenómeno del Perceptual Aliasing, zonas distintas del mapa son asignadas como similares por la matriz de similitud. Por lo tanto, utilizar solamente esta matriz para construir la topología no es una opción adecuada.

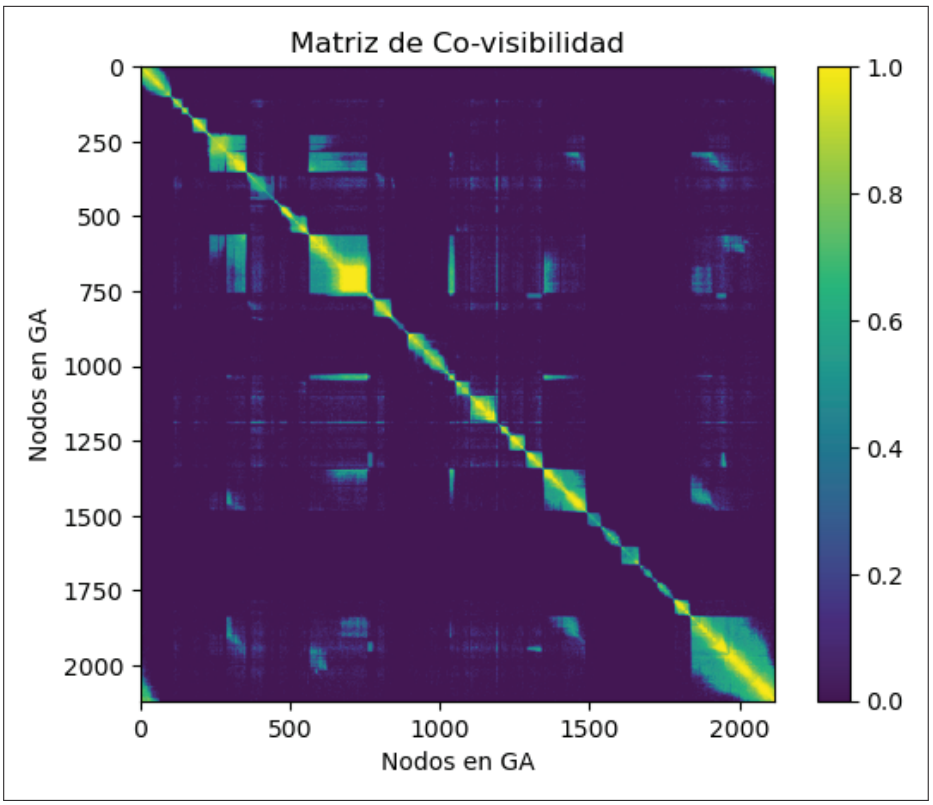


Figura 4.15: Matriz de co-visibilidad obtenida para el $AM = \{Seq2\_cloudy1\}$.

En la Figura 4.15 se presenta la matriz de co-visibilidad para el mismo Mapa de Apariencia (AM), utilizando un valor de μ=20. Se puede observar que la mayoría de las correlaciones con un valor alto de co-visibilidad se encuentran cerca de la diagonal principal. El resto de elementos de la matriz, fuera de la diagonal principal, toman valor cero en su gran mayoría. Esto es correcto, ya que, como hemos comentado, al tratarse de un dataset ordenado, los elementos distantes de la diagonal principal

representan lugares lejanos en términos de pose. Si se compara con la matriz de similitud, los distintos lugares están mucho mejor diferenciados en la matriz de co-visibilidad, formando submatrices cuadradas en torno a la diagonal principal. En la matriz de similitud también se pueden observar estas submatrices, pero están más difuminadas debido al fenómeno del *Perceptual Aliasing* (*PA*).

Es importante destacar que el algoritmo también encuentra co-visibilidad entre pares de elementos alejados de la diagonal principal. Esto puede ocurrir al revisitar alguna zona previamente vista, en el ejemplo del dataset *Seq2_cloudy1* de cOLD [67] empleado se revisita varias veces el mismo pasillo.

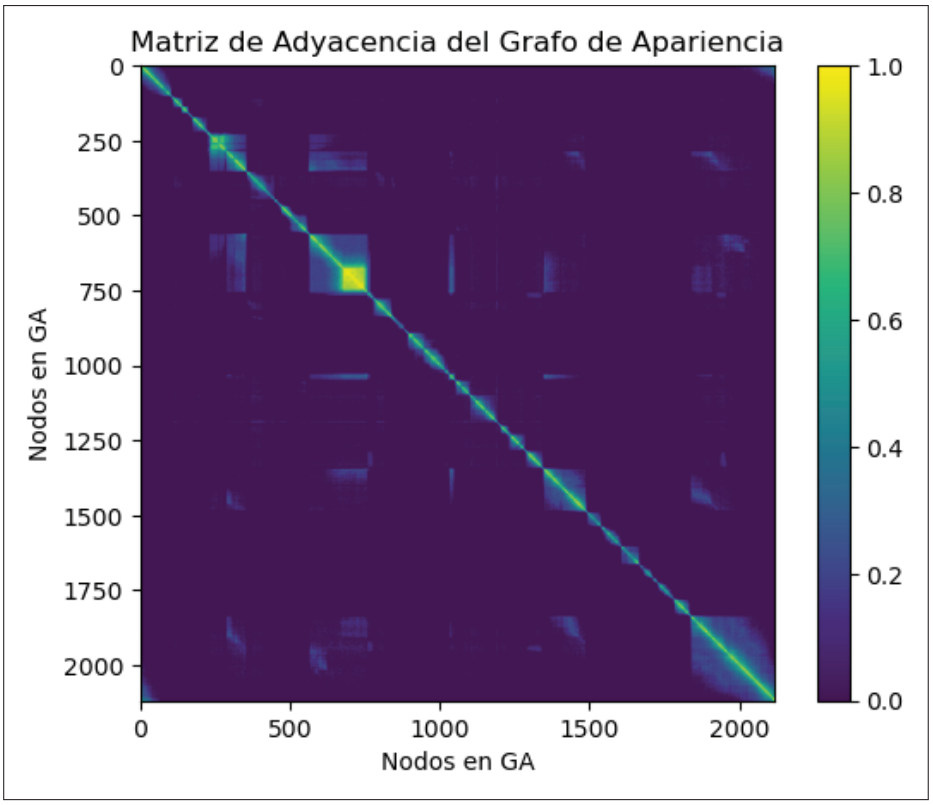


Figura 4.16: Matriz de adyacencia del *GA* construido sobre el $AM = \{Seq2\_cloudy1\}$.

Por último, en la Figura 4.16 se representa la matriz de adyacencia del Grafo de Apariencia (*GA*), que es el resultado de multiplicar elemento a elemento las matrices de similitud y co-visibilidad. En esta matriz se pueden apreciar claramente las representaciones de los distintos lugares del AM como submatrices marcadas sobre la diagonal principal. El resultado es un Grafo de Apariencia donde la mayoría de los nodos no están conectados, es decir, los elementos fuera de la diagonal principal

tienen un valor nulo, mientras que los grupos de nodos con un índice secuencial (diagonal principal) presentan una alta conectividad.

En la construcción de la matriz de adyacencia, se eliminan los casos de *Parceptual Alasing* (*PA*) presentes en la matriz de similitud multiplicando por elementos con valor nulo en la matriz de co-visibilidad. Por lo tanto, los nodos del *GA* se relacionan en base a la similitud entre apariencias y al grado de co-visibilidad entre las imágenes, evitando las relaciones de nodos que presenten *PA*. Es importante destacar que el valor del peso entre los arcos que conectan nodos es menor que en las matrices de origen, debido a que se multiplican los valores de co-visibilidad y similitud de los arcos, que toman valores menores a 1. En conclusión, se obtiene una relación más débil entre nodos, pero mucho más robusta y precisa, lo que permite evitar en gran medida el problema del *Perceptual Aliasing* en la agrupación.

El único parámetro variable en el procedimiento de construcción del Grafo de Apariencia es el umbral de número de inliers necesarios para determinar la co-visibilidad (μ), como se explica en la sección anterior. Este parámetro determina cómo de restrictivo es el cálculo de la co-visibilidad e influye en la matriz resultante con un mayor o menor número de imágenes co-visibles. En la sección 6.2 se evalúa el rendimiento de la matriz de co-visibilidad en función de este parámetro.

La matriz de adyacencia es la única representación del Grafo de Apariencia (*GA*) necesaria, ya que, como se explica en el capítulo 5, el método de corte normalizado opera solamente sobre esta matriz para obtener la topología de lugares. En el capítulo 6 se estudia la eficiencia del método de detección de co-visibilidad en términos de precisión y sensibilidad, así como la influencia del parámetro μ sobre la matriz de co-visibilidad resultante.

4.5. Conclusiones del capítulo

En este capítulo hemos introducido el concepto de Grafo de Apariencia (*GA*) como una representación del Mapa de Apariencia (*AM*) que permite establecer una topología de lugares sobre el mismo aplicando la bipartición recursiva del mismo. Se trata de un grafo ponderado no direccional, donde el peso entre nodos se basa en la similitud y co-visibilidad entre las imágenes que representan. También se detallan las características que debe cumplir *GA* para aplicar el método de bipartición espectral.

La similitud se emplea para relacionar la apariencia entre imágenes, mientras que la co-visibilidad se utiliza para abordar el problema del *Perceptual Aliasing* (*PA*). En este capitulo se define la co-visibilidad como un proceso de detección de correspondencias y la estimación de una matriz fundamental, lo que permite establecer relaciones geométricas entre las características locales de las imágenes. La principal desventaja que presenta este método es el elevado tiempo de ejecución

que requiere, siendo este un aspecto a mejorar.

De esta manera se combinan las ventajas de los descriptores globales y locales, obteniendo una relación entre elementos del AM (nodos del grafo) robusta frente al *PA*. Al considerar tanto la similitud de apariencia como la información espacial, nuestro enfoque mejora la precisión para sistemas de Localización basada en Apariencia.

Capítulo 5

Corte Normalizado del Grafo de Apariencia

Contenido

Sinopsis

Este capítulo se centra en la definición del Corte Normalizado de Grafo (*Ncut*) como estratégia a seguir para definir una topología de lugares sobre el Mapa de Apariencia (AM), subdividiendo el Grafo de Apariencia (GA). Una de las principales ventajas del *Ncut* es su capacidad para generar grupos balanceados de nodos altamente interconectados. Se presenta una metodología computacionalmente eficiente basada en la Bipartición Espectral del grafo para dividir el GA, además de una extensión de este método para obtener varios grupos.

5.1. Corte Normalizado de Grafo

El Corte Normalizado de grafo fue introducido por Shi y Malik en su trabajo [59], con el propósito de proporcionar un marco matemático coherente para abordar el problema del agrupamiento o *clustering* de grafos, en contraste con las técnicas anteriores que dependían principalmente de enfoques heurísticos. En esta sección se procede a definir y justificar matemáticamente esta herramienta.

Sea un grafo $G = \{V, E, W\}$, como el definido en el capítulo 4, puede ser dividido en dos grupos disjuntos, A y $\bar{A}$, tal que $A \cup \bar{A} = V$ y $A \cap \bar{A} = \emptyset$, simplemente eliminando las aristas que conectan las dos partes. El grado de disimilaridad entre las dos partes se calcula típicamente mediante la definición estándar de corte (cut), que es la suma total de los arcos eliminados:

$$cut(A, \bar{A}) = \sum_{u \in A, v \in \bar{A}} w(u, v) \tag{5.1}$$

La bipartición óptima de un grafo es aquella que minimiza este valor ($min\text{-}cut$) entre todos los posibles biparticiones. Sin embargo, el criterio del corte mínimo tiende a dividir grupos pequeños de nodos aislados en el grafo, lo cual puede resultar en divisiones poco eficaces. La Figura 5.1 ilustra uno de estos casos, donde se puede observar que los cortes que dividen el nodo n1 o n2 tienen un valor muy pequeño a pesar de no ser el corte más óptimo.

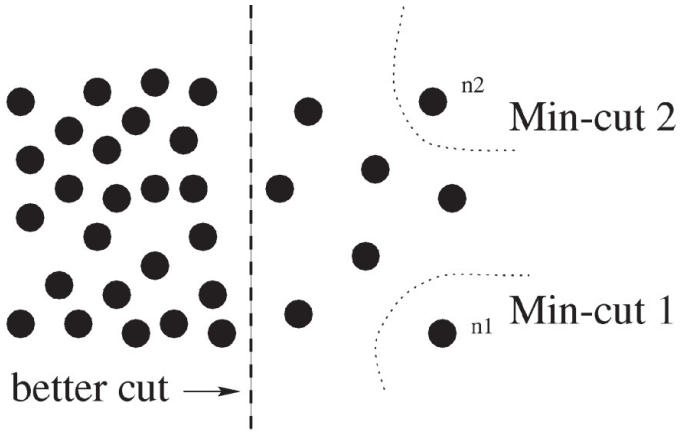


Figura 5.1: Representación del Corte mínimo (min-cut) de un grafo junto con el corte óptimo. Sobre el conjunto de nodos se representa el peso de los arcos inversamente proporcional a la distancia entre los nodos. [59].

Con el fin de evitar esta tendencia de particionar conjuntos pequeños de nodos, en [59] se introduce el Corte Normalizado (*Ncut*). En lugar de evaluar únicamente el peso de los arcos que conectan las dos particiones, esta medida calcula el valor del corte ponderado por la conexión total de todos los nodos de cada grupo con el grafo completo:

$$Ncut(A, \bar{A}) = \frac{cut(A, \bar{A}}{assoc(A, V)} + \frac{cut(A, \bar{A})}{assoc(\bar{A}, V)} \tag{5.2}$$

El término *assoc(A, V)* hace referencia a la asociación del subgrafo A con el grafo completo V. Es una medida de la cohesión entre grupos, es decir, la 'fuerza' de conexión entre los dos conjuntos disjuntos. Se puede calcular de la siguiente manera

$$assoc(A, V) = \sum_{u \in A, t \in V} w(u, t) \tag{5.3}$$

Para entender mejor estos conceptos, se ilustra la Figura 23:

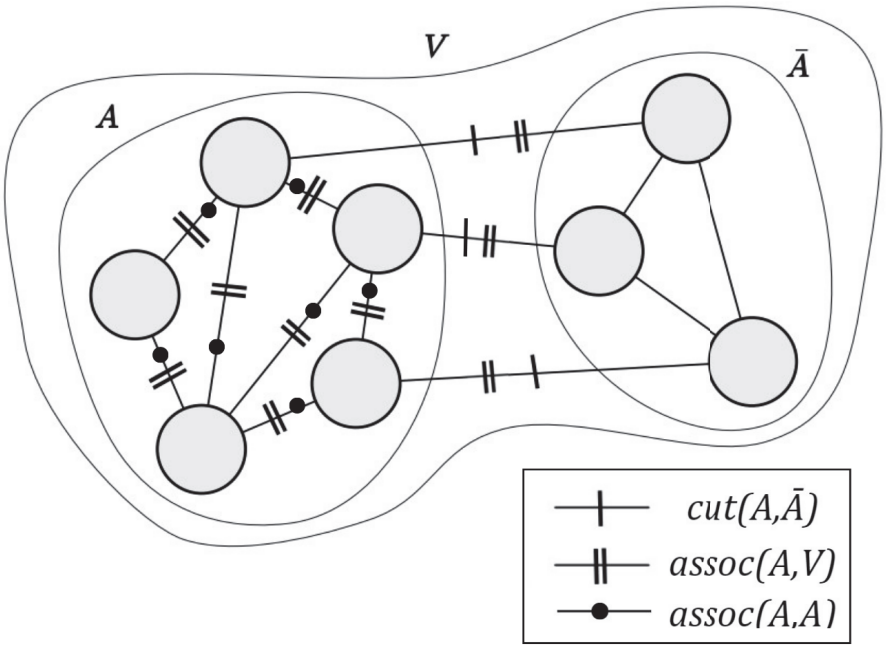


Figura 5.2: Ejemplo ilustrativo del corte (Eq.5.1) y asociación (Eq.5.3) [14].

De la Figura 5.2 podemos inducir la siguiente relación:

$$assoc(A, V) = cut(A, \bar{A}) + assoc(A, A) \tag{5.4}$$

Con esta definición de la disociación entre grupos, el valor del corte que divide grupos de nodos aislados casi seguramente será un gran porcentaje de la conexión total de estos conjuntos con todos los nodos. En consecuencia el valor del *Ncut* no será pequeño. Por lo tanto, el criterio adoptado para dividir el grafo consiste en minimizar el *Ncut*, dando lugar a subgrafos con un balance entre cohesión entre grupos y cohesión dentro de los grupos.

Pasamos a derivar el rango de valores que puede tomar *Ncut*. El valor mínimo que puede tomar el corte entre grafos es cero, en el caso en que no haya conexión entre los dos grupos, por lo tanto, el valor mínimo de *Ncut* es también cero. Para el máximo valor del corte, cuando los nodos de ambos grupos están conectados únicamente con los nodos del otro grupo, el valor de *assoc(A,A)* y *assoc($\bar{A}$,$\bar{A}$)* es cero. Teniendo esto en cuenta se puede derivar el valor máximo de *Ncut* a partir de las ecuaciones 5.2 y 5.4:

$$assoc(A, V)_{min} = assoc(\bar{A}, V)_{min} = cut(A, \bar{A})$$

$$Ncut(A, \bar{A})_{\text{máx}} = max\left(\frac{cut(A, \bar{A})}{assoc(A, V)}\right) + max\left(\frac{cut(A, \bar{A})}{assoc(\bar{A}, V)}\right) =$$

$$= \left(\frac{cut(A,\bar{A})}{cut(A,\bar{A})}\right) + \left(\frac{cut(A,\bar{A})}{cut(A,\bar{A})}\right) = 2$$

Por lo tanto, el rango de valores de *Ncut* pertenece al intervalo [0,2].

Como se discute en el trabajo de Shi y Malik [59], encontrar la bisección exacta que minimice *Ncut* es un problema computacionalmente intratable, siendo un problema *NP-completo*. En su estudio, proponen una solución aproximada basada en la descomposición espectral que produce cortes cercanos al valor óptimo. Este método consiste en la bipartición espectral de grafos, que se resume en el siguiente apartado.

5.2. Bipartición Espectral de grafos

El método de bipartición espectral de grafos es la técnica presentada en [59] para estimar la división del grafo que minimice el valor de *Ncut*. Se basa en la descomposición espectral de la matriz Laplaciana del grafo [68]. Es importante destacar que el método de bipartición espectral proporciona una aproximación del corte óptimo, ya que se omiten ciertas restricciones durante su desarrollo para hacerlo computacionalmente posible. En esta sección, presentamos una explicación simplificada de esta herramienta, refiriendo al lector al trabajo original [59] para una comprensión más completa.

Sea nuestro grafo $GA = \{V, E, W\}$ con las características descritas en el capítulo 4, donde W es la matriz de adyacencia, se considera una partición del conjunto de nodos V en dos grupos A y $\bar{A}$. Se define el vector indicarcador $\mathbf{x}$ de dimensión $|V|$, donde cada elemento $\mathbf{x}_i$ toma los valores 1 o -1 dependiendo de si el nodo i pertenece al grupo A o $\bar{A}$. Además, definimos el vector $\vec{d}$ de dimensión $|V|$, donde cada elemento es la suma de los pesos de los arcos incidentes en el nodo i, es decir, $di = \sum_j w(i,j), \forall j$. Construimos la matriz D, que tiene los valores de d en su diagonal. Como se demuestra detalladamente en [59], el valor que minimiza el *Ncut* se obtiene al reformular la ecuación 5.2 como:

$$argmin_x Ncut(x) = argmin_y \frac{y^T(D-W)y}{y^T Dy} \tag{5.5}$$

donde $y = (1+x) - b(1-x)$, un vector de dimensión $|V|$. En la ecuación anterior 5.5, el término $\mathbf{1}$ representa un vector Nx*1* donde todos los elementos son iguales a 1 y el término b se calcula del siguiente modo

$$b = \frac{\sum_{xi>0} d_i}{\sum_{xi<0} d_i}$$

Idealmente, el vector $\vec{y}$ debería tomar sólo dos valores discretos $y(i)\{1, -b\}$, ya que $\mathbf{x}_i$ solo puede tomar los valores 1 o -1. Sin embargo, esta restricción se flexibiliza con el fin de permitir que los elementos del vector y puedan tomar cualquier valor real. En consecuencia, la ecuación 5.5 deja de ser discreta y se puede minimizar resolviendo el sistema de autovalores:

$$(D-W)y = \lambda Dy \tag{5.6}$$

donde el término $L(V) = (D-W)$ es comúnmente conocida como la matriz Laplaciana del grafo [68].

La expresión 5.6 puede reescribirse como un sistema de autovalores estándar aplicando el cambio de variable $z = Dy$:

$$D^{-1/2}(D - W)D^{-1/2}z = \lambda z \tag{5.7}$$

Como se demuestra en los trabajos previos [14] [59], el autovalor más pequeño *z0* de la ecuación 5.7 toma valor cero. En el sistema original 5.6, esto se traduce en que el autovalor más pequeño es *y0=1*. Como se justifica en [59], dado que la fracción expresada en 5.5 es un cociente de Rayleigh [60], y sus autovectores son ortogonales, minimizar las ecuaciones 5.6 y 5.7 es equivalente a encontrar el segundo autovalor más pequeño. En consecuencia, resolver el mínimo *Ncut* expresado en 5.5 es equivalente a encontrar el segundo autovalor más pequeño, $\vec{y1}$, de 5.6.

La implicación de permitir que el vector $\vec{y}$ tome más de dos valores discretos radica en que, de igual manera, las componentes del autovector $\vec{y1}$ no se restringen únicamente a dos valores, sino que pueden tomar cualquier número real. Esto complica el criterio de la bipartición espectral, sin embargo, en la mayoría de los casos, este autovector proporciona una distinción clara entre los dos grupos de nodos A y $\bar{A}$. Como se propone en [14] existen tres criterios para asignar cada nodo a su grupo, en nuestra aproximación aplicamos el primer criterio. Consiste en observar el signo de cada elemento del autovector $\vec{y1}$, tal que si el elemento es $\vec{y1}_i$ es positivo o negativo, se asigna el nodo i al grupo A o $\bar{A}$ respectivamente.

5.3. Bipartición recuersiva del Grafo de Apariencia

El método de bipartición del grafo presentado ofrece una solución para dividir el grafo en dos partes a fin de obtener el valor que minimice el *Ncut*. No obstante, es necesario generalizar este método para dividir el grafo en múltiples grupos o subgrafos, a fin de poder definir una topología con varios lugares. Una estrategia sencilla propuesta en [14] [59] consiste en aplicar recursivamente la bipartición a cada uno de los subgrafos generados, obteniendo así múltiples grupos divididos por el valor que minimice el *Ncut* en cada iteración.

Surge entonces la interrogante de cuándo detener la reclusión, en otras palabras, cuando se obtiene un grupo óptimo. Para establecer este criterio se evalúa el valor del *Ncut* obtenido en cada bipartición, un buena métrica para intuir la calidad del corte. El *Ncut* mide la cohesión intergrupal de los subgrafos resultantes, inversamente escalada por la cohesión intragrupal dando un resultado acotado entre 0 y 2. En este sentido, valores cercanos a cero indican una conexión débil entre los grupos (una buena partición), mientras que valores cercanos a 2 indican que los grupos están más fuertemente conectados entre sí que consigo mismos y la partición no debería realizarse.

El método original propuesto en [59] propone establecer un umbral $\tau \in [0, 2]$ en base al cual evaluar el valor del *Ncut* para decidir si realizar o no la bipartición. Típicamente este valor se elige de forma heurística [69] [70], en la práctica se considera que un valor en el rango [0.2.1] dará buenos resultados. De esta manera, la recursión se detiene en el momento que la bipartición de un grupo supera el umbral impuesto

Sin embargo, trabajando con el Grafo de Apariencia (*GA*), encontramos que este criterio tiende a generar malos resultados. Como se explica en el capítulo 4, los pesos de los arcos entre nodos del *GA* están determinados por la similitud y la co-vibilidad. Estas medidas exhiben una alta variabilidad en ciertos conjuntos de imágenes de la escena, lo que da lugar a partes del grafo con una conexión igualmente variable. En consecuencia, al detener la bipartición recursiva al superar el umbral, se obtienen dos situaciones inconcluyentes en función del valor del umbral, como se ilustra en la Figura 5.3.

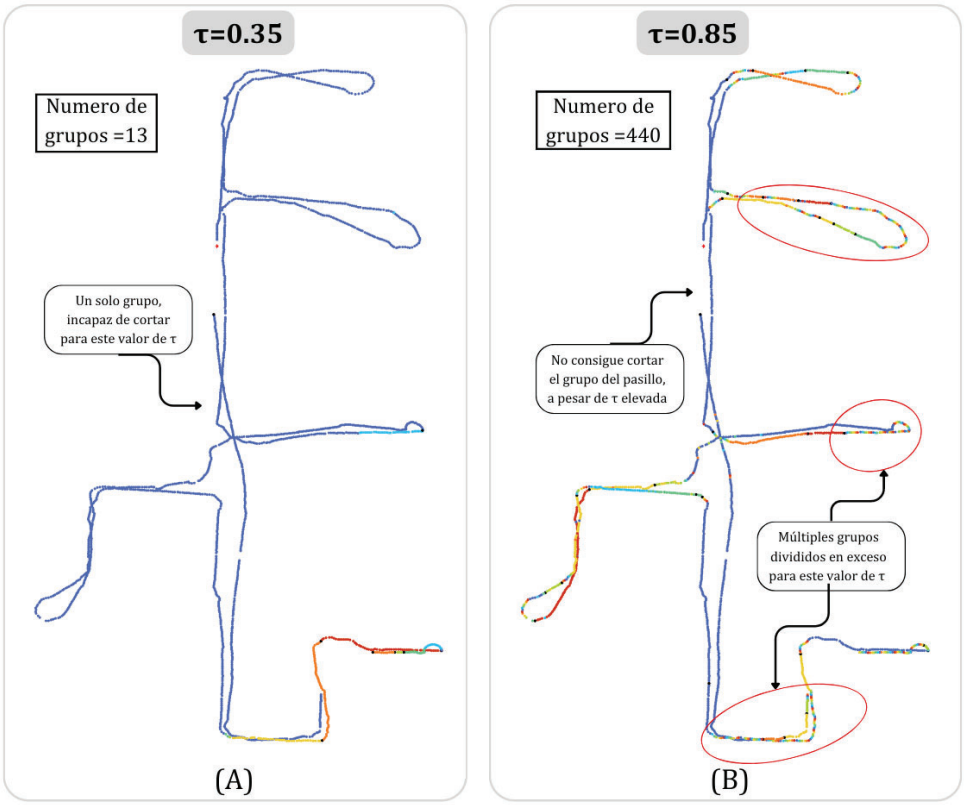


Figura 5.3: Se presentan dos ejemplos que ilustran los casos problemáticos al aplicar la bipartición recursiva original [59]. Se muestran por colores los grupos obtenidos de la bipartición recursiva.

En la imágen A se aplica un umbral bajo ($\tau = 0{,}35$) y se logran algunos grupos optimos. Sin embargo, la bipartición recursiva no logra cortar el mapa por completo debido a la presencia de zonas fuertemente conectadas. La recursión se detiene antes de cortarlo. En la imagen B, se emplea un umbral alto ($\tau = 0{,}85$), sin embargo no se logra cortar el grupo correspondiente al pasillo, nuevamente, la recursión se detiene antes de completar el corte. Además, las zonas menos conectadas (indicadas por círculos en rojo) se subdividen excesivamente debido al umbral alto utilizado.

En conclusión, deterner la recursión cuando se alcance un umbral τ no es efectivo ya que no es posible encontrar un umbral que corte de manera óptima el Grafo de Apariencia.

Para afrontar este problema, este estudio propone invertir la recursión y evaluar el valor del *Ncut* una vez se han establecido todos los grupos, modificando el criterio para detener la recursión. Esta propuesta se inspira en la naturaleza de los algoritmos recursivos basados en árboles binarios [71], estructuras de datos donde la información se organiza en hojas de tal manera que cada nodo puede tener un hijo izquierdo y un hijo derecho.

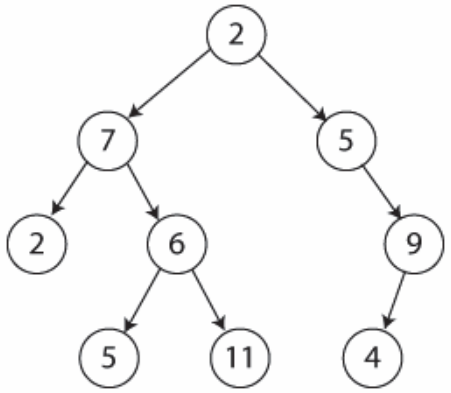


Figura 5.4: Ejemplo de árbol binario sencillo de tamaño 9. [71].

Nuestra propuesta plantea un árbol binario como el que muestra la Figura 5.5, en el cual cada hoja contiene un conjunto de nodos del grafo, V, mientras que sus hijos izquierdo y derecho contienen los nodos de los grupos A y $\bar{A}$ resultantes de la bipartición. Además, cada hoja almacena el valor de *Ncut* obtenido al dividir el grafo V en los grupos A y $\bar{A}$.

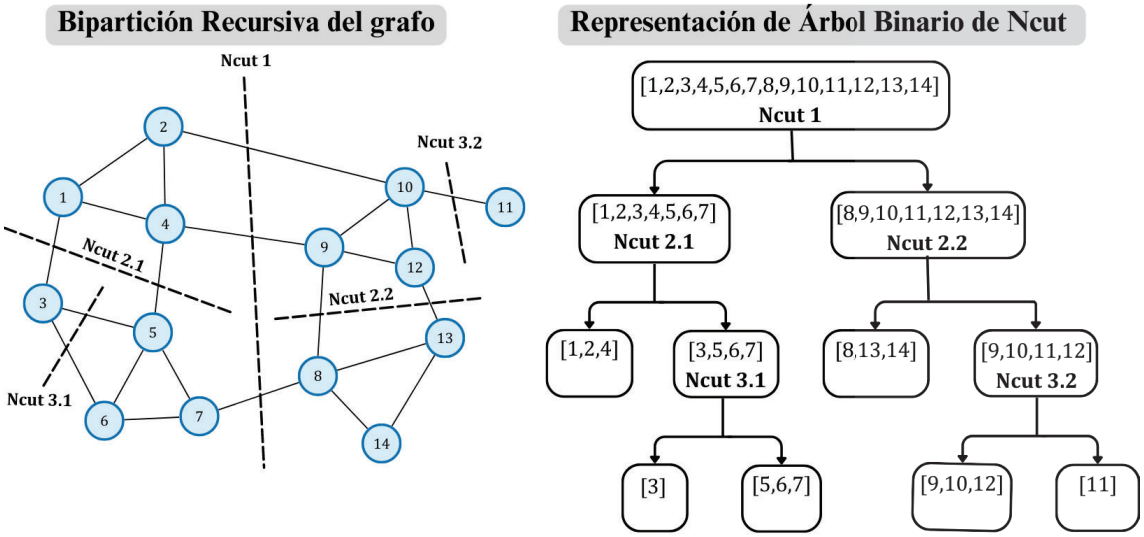


Figura 5.5: Representación de la bipartición recursiva y construcción del árbol binario.

Se aplica la bipartición espectral recursiva del Grafo de Apariencia (GA), calculando el valor de $Ncut$ en cada bipartición y construyendo el árbol binario. La recursión continúa hasta que el número de nodos en la hoja sea igual o inferior a tres, dando lugar al GA dividido en grupos de no más de tres nodos. Posteriormente, al deshacer la recursión (sentido ascendente del árbol binario) se evalúa el valor del $Ncut$ en cada hoja. Si dicho valor es inferior al umbral fijo τ se realiza la bipartición.

Es importante destacar que la bipartición real no se realiza hasta que no se evalúan los valores de $Ncut$ por lo que, en la bipartición recursiva simplemente se obtienen candidatos a subgrafos. Cuando se realiza laa bipartición real sobre GA es equivalente a eliminar los arcos que unían a los dos subgrafos candidatos. El resultado es una transformación sobre el Grafo de Apariencia, de forma que compone por los diversos subgrafos no conexos entre sí que representan los elementos de la topología de lugares.

5.4. Implementación

En esta sección se describen los detalles de implementación de la bipartición recursiva utilizando el método del Corte Normalizado de Grafo. El código correspondiente se encuentra en el Anexo C, proporcionando una visión más detallada.

Antes de aplicar la bipartición recursiva al Grafo de Apariencia (GA), es necesario verificar que cumpla con las características descritas en el capítulo 4. Específicamen-

te, debemos asegurarnos de que el *GA* sea un grafo conexo, es decir, que no contenga subgrafos no conexos ni nodos aislados. Para identificar estas situaciones, se utiliza el algoritmo de búsqueda no informada conocido como *Deep-first Search* (*DFS*) [72]. Este algoritmo es ampliamente utilizado en teoría de grafos para recorrer de manera ordenada todos los nodos de un grafo. En nuestra implementación, se utiliza una versión recursiva del algoritmo *DFS* que recorre los nodos conectados, un ejemplo explicativo se ilustra en el algoritmo 1. Esto se logra examinando los elementos distintos de cero en cada fila de la matriz de adyacencia (W), que representa las conexiones para un nodo específico. El algoritmo *DFS* se ejecuta para cada nodo encontrado, registrando los índices de los nodos visitados en cada iteración.

Al comparar los nodos visitados por el algoritmo *DFS* con los nodos reales del grafo, es posible determinar la existencia de subgrafos no conexos. En caso de encontrar nodos no visitados, se ejecuta de manera iterativa el algoritmo *DFS* para cada uno de ellos, obteniendo así un conjunto de subgrafos independientes internamente conectados. Se ilustra un ejemplo en el algoritmo 2, los nodos aislados se eliminan de la matriz de adyacencia. Una vez identificados los subgrafos, es necesario aplicar recursivamente el algoritmo de corte normalizado a cada uno de ellos, tratándolos como grafos independientes. Para lograr esto, se extraen submatrices de la matriz de pesos que corresponden a los nodos (filas y columnas) de cada subgrafo.

Algorithm 1: Función Deep-first Search

```
def DFS(visited, W, node):
    if node not in visited:
        visited += node # Actualiza lista de nodos visitados
        # Se buscan nodos conectados en W:
        connected_nodes= arg-where{ W(nodes,:) > 0}
        for index in connected_nodes:
            DFS(visited, W, index)     # Llamada recursiva

return visited

```

Algorithm 2: Función Check-connectivity

```
def Check_connectivity(W):
    visited = []
    saved=[]
    idx=0
    while length(visited) < length(node_list):
        DFS(visited, W, idx)    #Actualiza nodos visitados
        new_subgraph={visited}-{saved}  #Elimina nodos ya guardados
        #Se guarda el nuevo subgrafo
        miss={node_list}-{visited}
        if length(miss) > 0:
            # Indice de grafo no conexo para nueva busqueda
            idx = miss[0]

return subgraphs

```

Pasamos ahora a la implementación de la bipartición recursiva. Como se mencionó en la sección anterior, nuestra propuesta se inspira en la estructura de los árboles binarios. Sin embargo, no es necesario implementar explícitamente esta estructura de datos, ya que podemos aprovechar la naturaleza recursiva para realizar todos los cálculos necesarios. En el algoritmo 5 se muestra la implementación de la función denominada 'clustering'. Esta función recibe como entrada el umbral para evaluar el valor de *Ncut* y una lista de índices que representan los nodos. Durante las operaciones, se extraen submatrices de la matriz de adyacencia utilizando la lista de índices.

En primer lugar, se aplica la bipartición espectral del grafo de entrada para obtener los candidatos a clusters (A y B), se representa en el algoritmo 3. A continuación, se calcula el valor de *Ncut* de esta bipartición como se muestra en el algoritmo 4, el cual se almacena. A continuación, se realiza una llamada recursiva a la función hasta que el número de nodos en el grafo de entrada sea menor a 3, dividiendo así todo el grafo en clusters de no más de 3 nodos. Una vez finalizada la recursión, se ejecuta el resto de la función. Se evalúa el valor de *Ncut* en cada bipartición obtenida, y si este valor es menor al umbral establecido, se guardan las listas de índices correspondientes. Es importante destacar que se verifica que el tamaño de estas listas sea igual o mayor a 5 antes de guardarlas para evitar grupos excesivamente pequeños. Estos índices (listas de nodos) se eliminan del grafo original con el fin de evitar la duplicación de nodos.

Algorithm 3: Bipartición espectral

```
def SpectralBisection(W, graph_idx):
    # Submatriz de adyacencia para este grafo:
    sub_W = W(graph_idx,:graph_idx)
    # Calcula matriz D con la suma de las filas de la submatriz:
    D = diag{sub_W.sum(filas)}
    L = sub_W-D

    # Autovalores y aoutovectores de matriz Laplaciana:
    eigval, eigvec = eigsystem(L)
    # Biparticion optima a partir del segundo autovalor minimo:
    partition = eigvec(min{eigval}[1])

    # Clasificamos los nodos segun los valores de este autovector:
    for i in range(length(partition):
        if partition[i] > 0: A += graph_idx[i]
        if partition[i] < 0: B += graph_idx[i]
return A, B

```

Algorithm 4: Corte Normalizado

```
def Ncut(W, graph_idx, A, B):
    # Calcula valor del corte entre subgrafos A y B:
    cut = sum{W(A,:B)}     # Suma de arcos entre A y B
    # Calcula valor de asociacion:
    assocA = sum{W(A,:graph_idx)} # Suma de arcos entre A y el grafo
    assocB = sum{W(B,:graph_idx)} # Suma de arcos entre B y el grafo
    # Calcula Ncut:
    Ncut = (cut/assocA) + (cut/assocB)

return Ncut

```

Algorithm 5: Bipartición Recursiva

```
def clustering(W, graph_idx, threshold):
    if length(graph_idx) > 2:
        # Biparticion espectral, se obtienen subgrafos A y B
        A, B = SpectralBisection(W, graph_idx)
        # Ncut de biparticion:
        Ncut_value=Ncut(W, graph_idx, cluster_A, cluster_B)

        # Llamada recursiva, se actualizan los subgrafos
        A=clustering(A, threshold)
        B=clustering(B, threshold)

        # Evaluacion del corte tras la recusion:
        if Ncut_value < threshold:
            if length(A) > 5:
                # Guarda subgrafo A
                # Elimina nodos guardados de grafo original:
                graph_idx={graph_idx}-{A}
            if length(B) > 5:
                # Guarda subgrafo B
                # Elimina nodos guardados de grafo original:
                graph_idx={graph_idx}-{B}

return graph_idx

```

5.5. Resultados

En esta sección, se presenta el resultado de aplicar la topología de lugares propuesta en este estudio a un Mapa de Apariencia (AM) específico. En particular, se examina el caso en el que el AM corresponde al dataset *Seq2_ cloudy1* [67] (ver sección 6.1). En primer lugar, se construye el Grafo de Apariencia (GA) a partir del AM, siguiendo el procedimiento descrito en el capítulo 4. Luego, se aplica la bipartición recursiva sobre la matriz de adyacencia del GA, como se describe en este capítulo.

En el Anexo C se incluye el código relativo a la creación de la topología de lugares utilizando las matrices de similitud y co-visibilidad que conforman el GA. La Figura 5.6 muestra el mapa topológico resultante, donde cada elemento del AM se posiciona en su ubicación correspondiente. Se utiliza una codificación de colores para representar los diferentes grupos obtenidos. Asimismo, en lfa Figura 5.7 se muestra la matriz de adyacencia original y la matriz de adyacencia resultante de aplicar la bipartición recursiva, donde se muestran los grupos obtenidos por colores relativos a los colores de los grupos en el mapa.

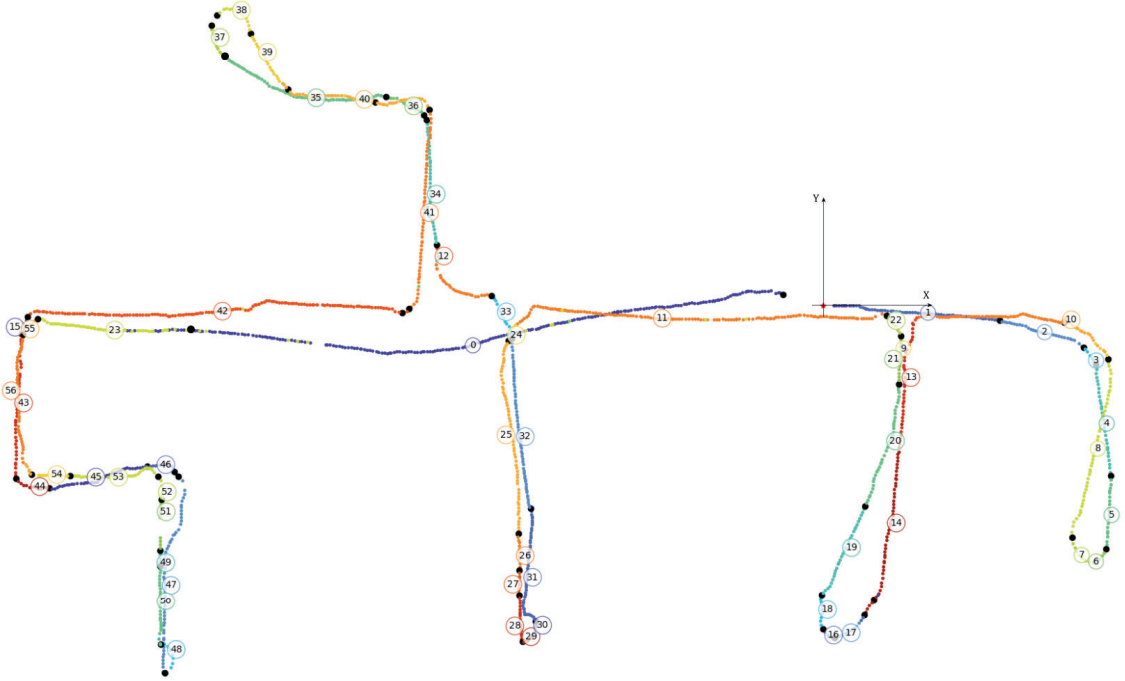


Figura 5.6: Mapa topológico resultado de aplicar la bipartición con $\tau = 0{,}5$ recursiva para el dataset *Seq2_ cloudy1* de *COLD*. Cada punto en el mapa representa un elemento del Mapa de Apariencia en su posición correspondiente. Los grupos identificados se representan por colores junto con el índice del grupo.

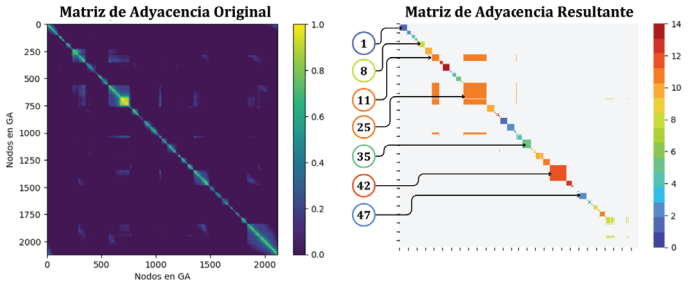


Figura 5.7: Representación de la matriz de adyacencia original del Grafo de Apariencia y la matriz de adyacencia resultante obtenida al aplicar la técnica de bipartición recursiva. Los grupos identificados se representan mediante la codificación de colores usada en 5.6. Se puede apreciar que, tras la bipartición recursiva se eliminan los arcos entre los distintos subgrafos, por lo que se eliminan estos valores de la matriz de adyacencia resultante.

Se puede observar en la Figura 5.6 que, en general, los grupos obtenidos muestran una notable ausencia de Perceptual Aliasing (*PA*). Es decir, los grupos o lugares obtenidos están bien acotados en términos de pose, sin clasificaciones erróneas. En casos en los que la secuencia presenta una alta rotación, el método tiende a identificar varios grupos más pequeños debido a los cambios bruscos en la apariencia. En la Figura 5.8 se muestra el mapa topológico con algunas fotos de los distintos grupos obtenidos.

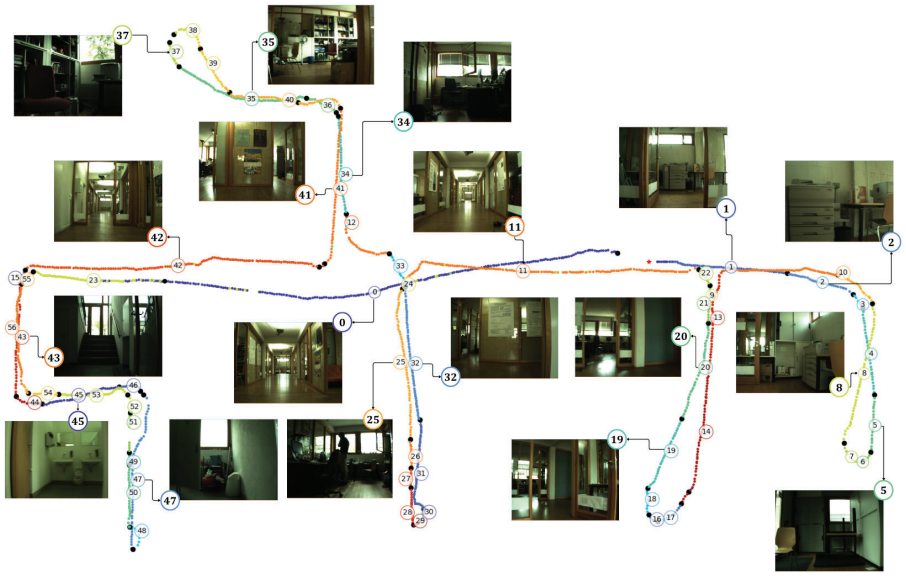


Figura 5.8: Representación del mapa topológico donde se ilustran algunas fotos pertenecientes a los grupos.

Como se mencionó anteriormente en este capítulo, el método de Corte Normalizado requiere un parámetro τ, que determina el valor óptimo de *Ncut* para realizar la bipartición del sobre *GA*. La Figura 5.9 muestra un ejemplo de la bipartición del grafo para varios valores de τ.

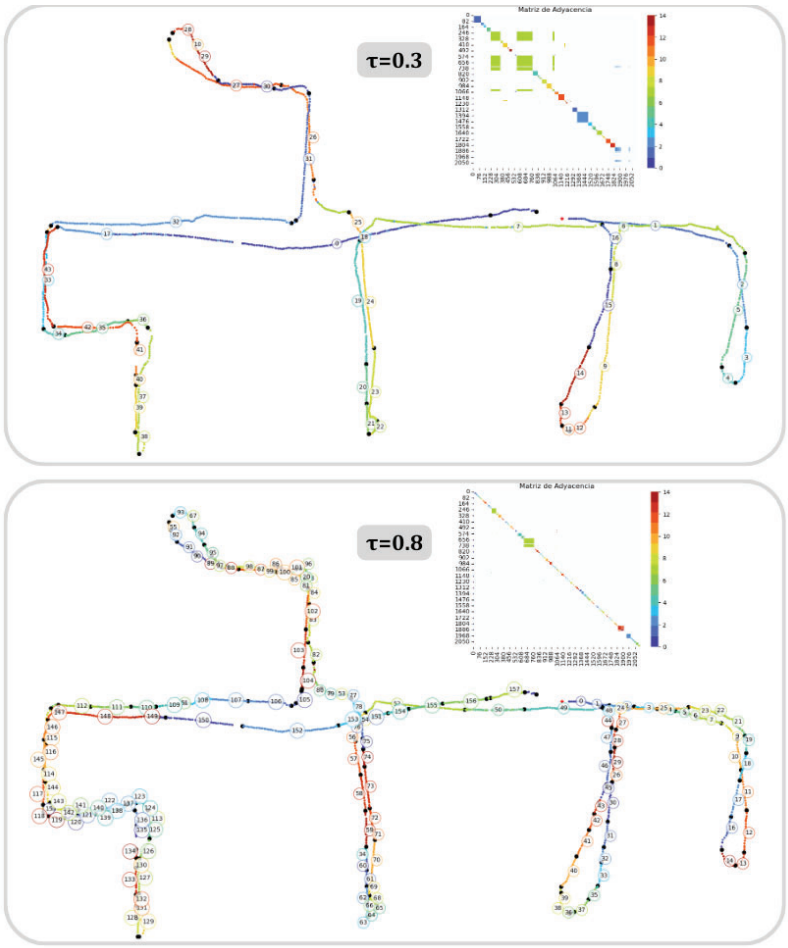


Figura 5.9: Resultado de la bipartición recursiva sobre *GA* para $\tau = 0{,}3$ y $\tau = 0{,}8$

Este parámetro indica el nivel de restricción del método al realizar el corte en el grafo. Un valor bajo de tau resulta en un menor número de grupos con un mayor número de elementos en cada grupo, ya que el *Ncut* de los grupos no cortados supera el umbral. Por otro lado, un valor alto de τ conduce a una bipartición más permisiva, lo que resulta en un mayor número de grupos con menos elementos, ya que el valor de *Ncut* de estos grupos está por debajo del umbral.

Aunque el conjunto de datos utilizado presenta una secuencia ordenada de imágenes, nuestro método es aplicable también a conjuntos desordenados de imágenes. Como ejemplo de ello, en la Figura 5.10 se ilustra el resultado de construir el *GA* y el mapa topológico utilizando la secuencia *Seq2_ cloudy1*, donde previamente se han desordenado las imágenes.

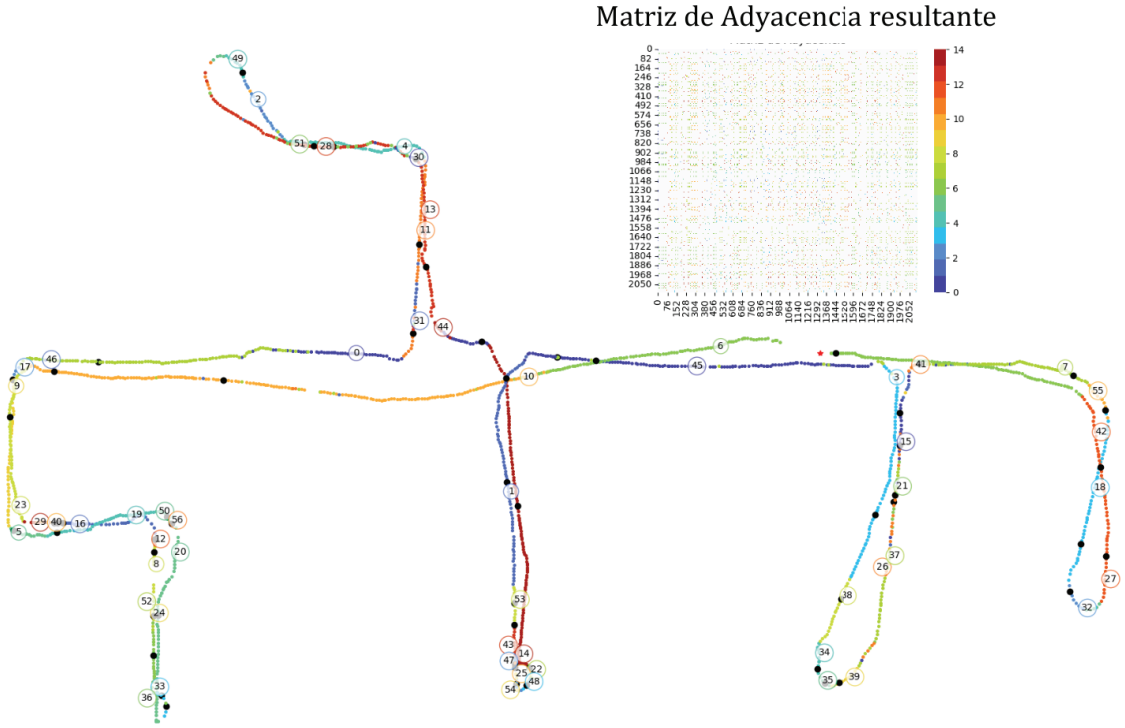


Figura 5.10: Mapa topológico y matriz de adyacencia resultante para un dataset desordenado en su origen. En la matriz de adyacencia, se observa que los nodos dentro de los grupos no siguen un patrón de ordenación, como sucede en los ejemplos anteriores.

5.6. Conclusiones del capítulo

En este capítulo se ha presentado la estrategia de bipartición recursiva del Grafo de Apariencia (*GA*) con el objetivo de definir una topología de lugares sobre el mismo. En cada bipartición, se trata de conseguir el valor mínimo de Corte Normalizado (*Ncut*), concepto definido en este capítulo. El *Ncut* evalúa la disimilitud entre subgrupos de un grafo normalizada por la asociación de cada subgrupo con el grafo completo, lo que lo convierte en una métrica adecuada para evaluar la calidad del corte del grafo.

Se presenta la bipartición espectral de grafos como un método para aproximar el corte de gráfico que minimice *Ncut*. Consiste en una técnica que se basa en los autovalores de la matriz Laplaciana del GA, de forma que el autovector asociado al segundo menor autovalor es el que minimiza el *Ncut*. El resultado de la bipartición son dos grafos no conexos donde los distintos nodos se clasifican en función de los valores de este autovector.

Se introduce una solución para generalizar el método de bipartición espectral de grafos a fin de obtener múltiples grupos a partir del GA. Esta técnica se basa en aplicar el algoritmo de bipartición de forma recursiva sobre los multiples subgrupos obtenidos. En este trabajo, se propone una modificación del método original que responde mejor a las características del Grafo de Apariencia (GA). Se realiza la bipartición del grafo completo buscando el mínimo *Ncut* en cada corte, hasta obtener grupos de tamaño mínimo. Luego, se determina si se debe realizar el corte en función del valor de *Ncut*, si queda por debajo del umbral τ se considera un corte óptimo. Cada subgrafo obtenido es inconexo con el resto, ya que cuando se realiza una bipartición se eliminan los arcos que unen los subgrafos.

En conclusión, la topología propuesta nace de biparticionar recursivamente el Grafo de Apariencia de forma que se minimice el valor del Corte Normalizado (*Ncut*). La topología de lugares se representa por medio del Grafo de Apariencia cortado, donde los elementos de esta topología son los multiples grupos de nodos o subgrafos no conexos obtenidos, que representan distintos lugares del Mapa de Apariencia con una apariencia similar, donde además se evita el fenómeno del *Perceptual Aliasing*.

Parte III

Conclusiones

Capítulo 6

Evaluación

Contenido

Sinopsis

En este capítulo se exponen los métodos utilizados para evaluar el rendimiento de los algoritmos propuestos. El capítulo se divide en dos secciones, la primera se dedica a la evaluación de la co-visibilidad para la construcción del Grafo de Apariencia (GA), mientras que la segunda se centra en la evaluación del mapa topológico aplicado a un sistema de Visual Place Recognition (VPR).

En cada sección se describen los criterios de evaluación empleados, así como su propósito. Los parámetros de ambos sistemas son evaluados en función de estos criterios, y se analizan los resultados obtenidos.

6.1. Dataset

En la evaluación presentada en este capítulo, se ha utilizado un datasets de dominio público para validar el modelo topológico propuesto. Se trata de *COLD* dataset [67] que contiene distintas secuencias de conjuntos de imágenes con pares descriptor-pose asociados, capturados en un entorno interior por un robot móvil.

El dataset *COLD* consiste en imágenes reales de una oficina (*Freiburg*, parte A) capturadas a una frecuencia de 5 Hz. El conjunto de datos incluye múltiples secuencias que recorren la oficina visitando diferentes lugares en condiciones de iluminación variadas, como días nublados, soleados y de noche con iluminación artificial.

Para la construcción de la topología de lugares evaluada en este capítulo, se utiliza la secuencia *Seq2_ cloudy1* de *COLD*, que consta de más de 2,000 imágenes geolocalizadas capturadas en un día nublado. Este conjunto de datos representa el Mapa de Apariencia (AM) sobre el cual construimos el Grafo de Apariencia (GA) y posteriormente la topología de lugares.

Para evaluar esta topología, se implementa un sistema de Visual Place Recognition (VPR) simple por medio del cual se localizan un conjunto de consultas en el mapa topológico. Las consultas se obtienen a partir de las secuencias *Seq1_ cloudy1*, *Seq2_ night1*, y *Seq2_ sunny1* de *COLD*, que se relacionan con distintas secuencias de la oficina para las condiciones lumínicas que las denominan. Se ha garantizado que estos conjuntos de datos visiten únicamente los lugares representados en el mapa topológico, con el fin de asegurar una evaluación eficiente del sistema de *VPR*.

Para evaluar la detección de co-visibilidad, se utilizan conjuntos de datos controlados obtenidos a partir de las secuencias *Seq2_ cloudy1*, *Seq2_ night1*, y *Seq2_ sunny1*. Estos conjuntos de datos controlados se seleccionan manualmente para obtener el valor real de co-visibilidad entre imágenes y así evaluar el método propuesto. Como se explica en la siguiente sección, es conveniente obtener el valor real de co-visibilidad (o *ground truth*) entre imágenes de esta manera, ya que soluciones basadas en la pose no son concluyentes. En la Figura 6.1 se muestra un ejemplo del conjunto de datos controlados correspondiente a la secuencia *Seq2_ cloudy1*.

De este dataset también se han tomado varias imágenes para ilustrar ejemplos a lo largo de la presente memoria.

6.2. Evaluación de la Co-visibilidad

6.2.1. Descripción

En esta sección se describe el proceso de evaluación de la medida de co-visibilidad, como elemento fundamental para la creación del Grafo de Apariencia *GA*. Para ello, se utiliza un dataset controlado obtenido a partir de los dataset *Seq2_ cloudy1*, *Seq2_ night1* y *Seq2_ sunny1* perteneciente a *COLD*, como se relata en la sección anterior. Este dataset consiste en imágenes indexadas sobre las cuales se conoce que pares son co-visibles, es decir, se dispone del valor real de la co-visibilidad o *ground truth*. Se muestra un ejemplo de uno de los datasets utilizados en la Figura 6.1. El código relevante a esta prueba de evaluación se expone en el Anexo B.

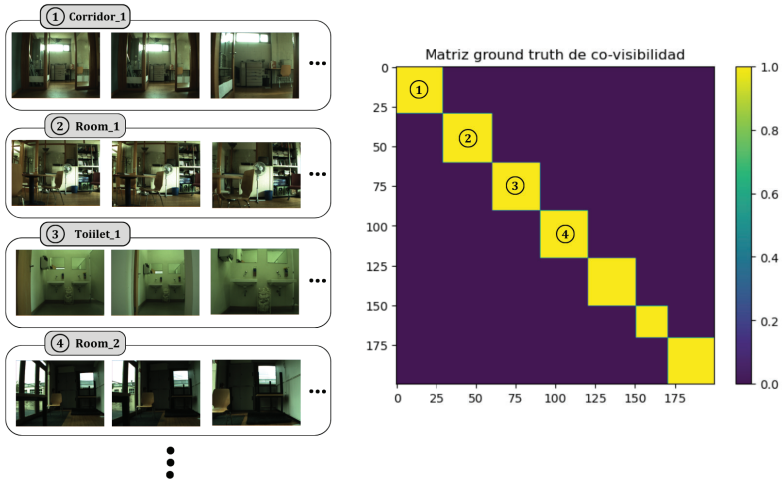


Figura 6.1: Representación del dataset controlado para *Seq2_ cloudy1* de *COLD*.

Se representa la matriz de co-visibilidad real, a la que haremos referencia como matriz de *ground truth* (GT). Se observa que los grupos en la matriz están claramente delimitados ya que han sido etiquetados como lugares distintos. Se evalúa el sistema comparando la matriz de co-visibilidad obtenida con esta matriz.

Es importante destacar que la matriz de *ground truth* es binaria, ya que solo toma dos valores (0 o 1). En consecuencia, se evalúa únicamente la detección de la co-visibilidad entre pares de imágenes, sin considerar el porcentaje de solapamiento. Para ello, se transforma la matriz de adyacencia W del *GA* de tal manera que $\forall w(i,j) > 0 \longrightarrow w(i,j) = 1$.

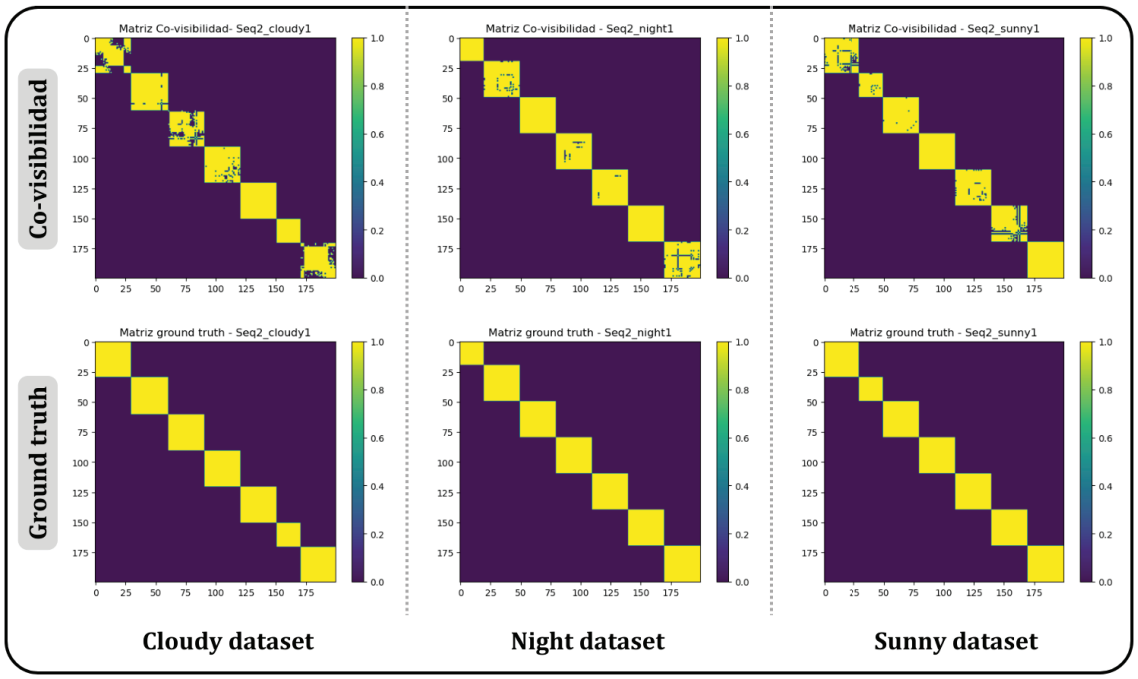


Figura 6.2: Representación de la matriz de *ground truth* y la matriz de co-visibilidad binaria ($\mu = 20$) para cada uno de los dataset controlados.

La medida de la co-visibilidad se evalúa en términos de precisión y sensibilidad (*precision and recall*). Para ello, se comparan elemento por elemento la matriz de co-visibilidad predicha por el algoritmo (C) y la matriz de *ground truth* (GT). Dado que se trata de una clasificación binaria, se derivan cuatro posibles situaciones:

- Verdadero Positivo (TP): Predicción positiva de co-visibilidad correcta.

$$TP+ = 1 \longleftrightarrow C(i,j) = GT(i,j) = 1$$

- Verdadero Negativo (TN): Predicción negativa de co-visibilidad correcta.

$$TN+ = 1 \longleftrightarrow C(i,j) = GT(i,j) = 0$$

- Falso Positivo (FP): Predicción positiva de co-visibilidad incorrecta.

$$FP+ = 1 \longleftrightarrow C(i,j) = 1 \neq GT(i,j) = 0$$

- Falso Negativo (FN): Predicción negativa de co-visibilidad incorrecta.

$$FN+ = 1 \longleftrightarrow C(i,j) = 0 \neq GT(i,j) = 1$$

Se comparan los elementos de ambas matrices y se cuentan cuántas veces se da cada uno de estos casos. Estos valores se utilizan para calcular la precisión y la sensibilidad del sistema, como se indica en las ecuaciones 6.1. En el caso ideal, todos los elementos en ambas matrices toman el mismo valor, lo que implica que FP=0 y FN=0.

$$\begin{aligned} P &= \frac{TP}{TP+FP} \\ R &= \frac{TP}{TP+FN} \end{aligned} \tag{6.1}$$

La precisión indica qué proporción de las predicciones positivas son realmente positivas, la sensibilidad evalúa qué proporción de los casos positivos reales son correctamente identificados. En la siguiente sección, se presentan los resultados para las tres secuencias estudiadas. Se lleva a cabo un estudio del parámetro μ, que actúa como umbral para determinar el número de inliers, en términos de precisión y sensibilidad.

Es importante señalar que, de hecho, se cuenta con las poses de las imágenes comparadas, lo que sugiere una alternativa más directa para el sistema de evaluación de la co-visibilidad en términos de su cercanía en pose. Sin embargo, nuestro enfoque evita utilizar la pose para estimar el *ground truth* de la co-visibilidad, debido a que la pose y la apariencia en la escena suelen tener una relación débil. Esto se debe, entre otros factores, al problema de las oclusiones en el entorno, donde dos imágenes cercanas en pose experimentan cambios abruptos en su apariencia. Por ello se prefiere escoger manualmente los casos de co-visibilidad a fin de tener una representación auténtica del *ground truth*.

6.2.2. Resultados

En esta sección se presentan los resultados de la evaluación de la medida de co-visibilidad. Se lleva a cabo un estudio del parámetro μ, que representa el umbral necesario para determinar la co-visibilidad entre imágenes (ver capitulo 4). Se realiza un análisis detallado de cómo el valor de μ afecta la matriz de co-visibilidad resultante en términos de precisión y sensibilidad, tal como se ha explicado en la sección anterior.

Un umbral μ bajo implica que la detección de co-visibilidad es más flexible, ya que requiere un menor número de inliers entre pares de imágenes, lo que conduce a un incremento en la detección de pares co-visibles. Por el contrario, a medida que este umbral se incrementa, el método se vuelve más restrictivo, lo que aumenta la precisión del método, pero también puede dar lugar a clasificaciones negativas incorrectas (Falsos Negativos).

En el proceso de evaluación, se calcula una matriz de co-visibilidad para distintos valores del umbral μ, en el rango de [5,60] aumentando en pasos de 5 unidades. A continuación, se compara la matriz obtenida con la matriz de *ground truth* y se calcula la precisión y sensibilidad correspondientes a cada umbral. Este proceso se repite para cada uno de los tres conjuntos de datos controlados definidos previamente (véase Figura 6.2).

- **Curva de Precisión**

En la Figura 6.3 muestra la gráfica de precisión para los diferentes valores de μ. Se puede observar que para los tres datasets se obtienen resultados de precisión muy similares, por lo que se solapan las curvas. En los tres casos se alcanza una precisión máxima de 1 al superar el valor de μ=15. Esto implica que el algoritmo no comete ningún error de clasificación positiva, es decir, no se producen Falsos Positivos. Por lo tanto, sobre el conjunto de datos controlados, todas las predicciones positivas realizadas por el método son correctas a partir de dicho umbral.

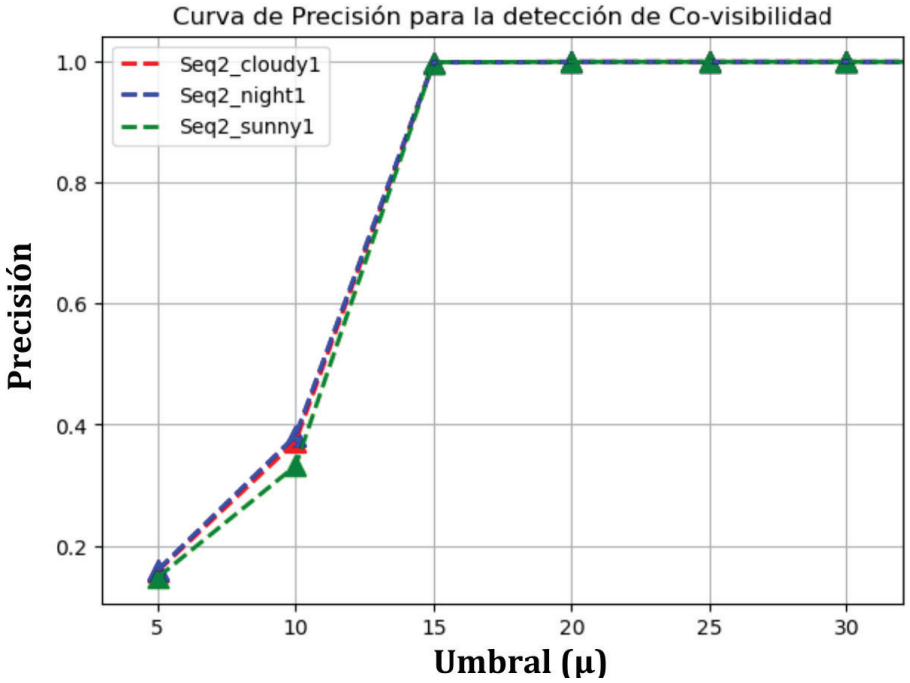


Figura 6.3: Resultado del cálculo de la precisión para la detección de co-visibilidad iterando sobre el parámetro μ.

- **Curva de Sensibilidad**

La sensibilidad (*recall*) proporciona información sobre la capacidad del método para detectar todos los casos positivos reales. La Figura 6.4 muestra la gráfica de

sensibilidad para diferentes valores del parámetro μ. Se observa que, para los tres conjuntos de datos empleados, la sensibilidad es máxima con el valor mínimo de umbral, cuando la detección de co-visibilidad es más permisiva. Esto implica que el número de Falsos Negativos es muy reducido, ya que casi todas las imágenes se predicen correctamente como co-visibles.

La sensibilidad disminuye a medida que se incrementa el umbral, dada la aparición de Falsos Positivos que reducen la exhaustividad del método. Sin embargo, para μ=15, cuando se alcanza la máxima precisión (FP=0), se obtiene un valor de sensibilidad superior al 90 % para los tres casos.

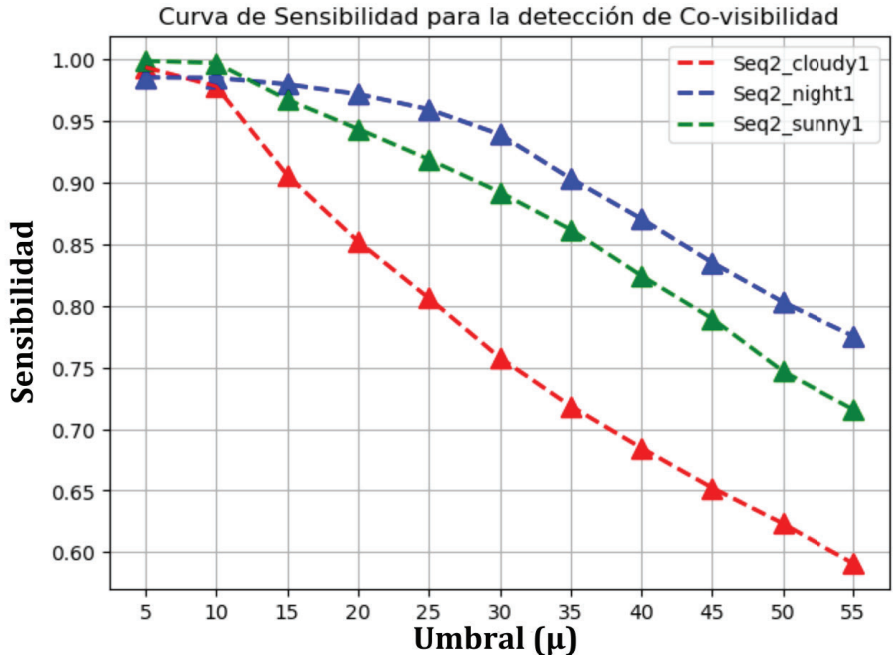


Figura 6.4: Resultado del cálculo de la sensibilidad para la detección de co-visibilidad iterando sobre el parámetro μ.

De las curvas de precisión y sensibilidad presentadas podemos concluir que el rango de valores óptimo para el parámetro μ se encuentra entre [15, 30], donde estas variables alcanzan un mayor rendimiento conjuntamente.

- **Curva de Precisión-Sensibilidad**

Las dos curvas presentadas anteriormente muestran buenos resultados del método para según qué valor del umbral. Lo ideal es alcanzar un equilibrio entre precisión y sensibilidad, es decir, entre número de Falsos Positivos y Falsos Negativos. Esta relación es inversamente proporcional: cuanto más exigente sea la medida de co-visibilidad (para valores altos de μ), mayor será la tasa de detecciones negativas

incorrectas. Por otro lado, si el método es más flexible (para valores bajos de μ), se vuelve menos preciso debido al aumento en el número de falsos positivos.

Con el fin de entender esta relación, se presenta la curva de precisión y sensibilidad (*Precision-Recall Curve, PRC*) en la Figura 6.5. Esta curva muestra la relación entre estas dos variables para diferentes umbrales de decisión μ. El área bajo la curva (AUC) de esta métrica es una medida de la calidad global del sistema, donde valores cercanos a 1 (área del 100 %) indican una alta precisión y sensibilidad. En la Figura 6.5 se muestra el AUC para los tres conjuntos de datos empleados.

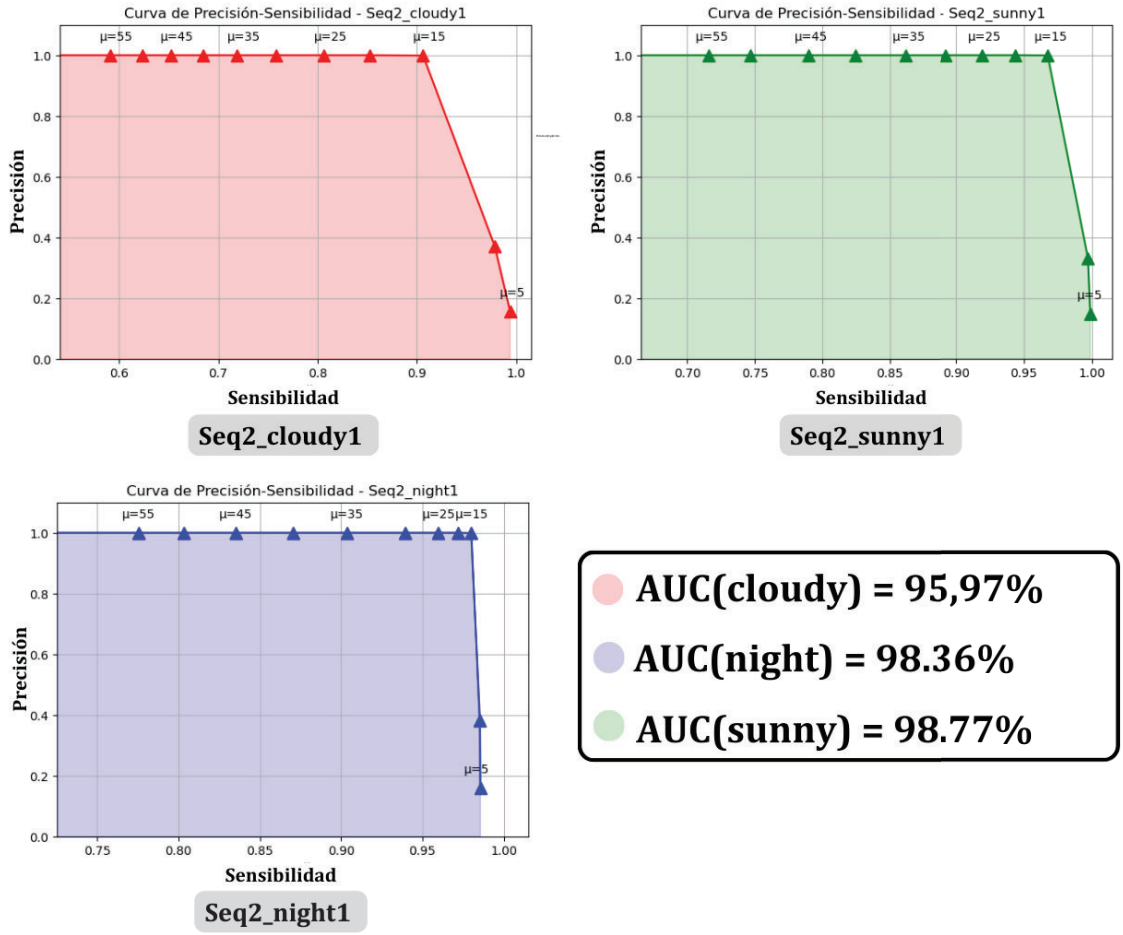


Figura 6.5: Curvas de Precisión y Sensibilidad sobre los dataset controlados. Se indica el área bajo la cuva (AUC) para cada gráfica.

Los resultados de precisión y sensibilidad obtenidos para evaluar la co-visibilidad son altamente prometedores en los tres dataset utilizados. Se puede concluir que el

método implementado es robusto en términos de precisión, ya que descarta adecuadamente los Falsos Negativos. Además, el área bajo la curva en los tres casos es cercana a 1, lo que indica que el sistema es altamente efectivo en términos globales.

Adicionalmente, se demuestra que el cálculo de la co-visibilidad no se ve afectado por las condiciones lumínicas variables, como las condiciones (*cloudy*, *night*, *sunny*). Esto se debe a que se basa en el descriptor *SIFT*, que es invariante a este tipo de cambios en la apariencia.

Es importante tener en cuenta que esta evaluación se ha realizado en un dataset donde se controlan los casos de co-visibilidad. Sin embargo, cabe preguntarse cómo se comportaría el sistema en un dataset no controlado. En el capitulo 7 se incluye esta como una de los objetivos de las líneas futuras de esta investigación.

En las siguientes secciones de este capítulo, se evaluarán diferentes aspectos de la topología de lugares utilizando un dataset no controlado. A partir de estos resultados, podemos inferir que la medida de co-visibilidad no difiere significativamente de los resultados obtenidos con este dataset.

6.3. Perceptual Aliasing en la Topología de Lugares

6.3.1. Puntuación de Silueta

En esta sección se describe el proceso de evaluación de la topología de lugares propuesta en base a la detección de Perceptual Aliasing (*PA*) en la clasificación de los elementos del Mapa de Apariencia (*AM*). El *PA* ocurre cuando se clasifica un elemento del *AM*, en función de su apariencia, dentro de un grupo distante en términos de pose. Una métrica que propone este trabajo para detectar la presencia de *PA* en nuestra topología de lugares consiste en la puntuación de silueta (*silhouette score*, *SH*).

La puntuación de silueta [73] [74] es una técnica ampliamente utilizada en el campo del machine learning para interpretar y validar la consistencia de algoritmos de clasificación. Esta técnica proporciona una representación gráfica que evalúa la calidad de la clasificación de un conjunto de elementos, considerando la pertenencia de cada elemento dentro de su grupo en comparación con otros grupos. Para determinar el grado de pertenencia es necesario definir una métrica para evaluar la distancia entre los elementos y los grupos formados, distinta a la empleada para realizar la clasificación. El valor de *SH* varía entre [-1,1], donde valores positivos indican una buena clasificación del elemento en su grupo correspondiente, mientras que valores negativos indican lo contrario, es decir, que el elemento es más cercano a otros grupos que al suyo propio.

La relevancia de esta técnica radica en que se evalúa la calidad de la clasificación mediante una métrica distinta a la empleada en el proceso de clasificación en sí. En nuestra evaluación, se busca determinar si los elementos del Grafo de Apariencia (GA) están correctamente clasificados en términos de pose. Para lograr esto, se requiere de una métrica que permita calcular la distancia entre una pose y la distribución de poses de los grupos en el mapa topológico (GA) en el espacio de poses $SE(2) = \{x, y, \theta\}$ [47].

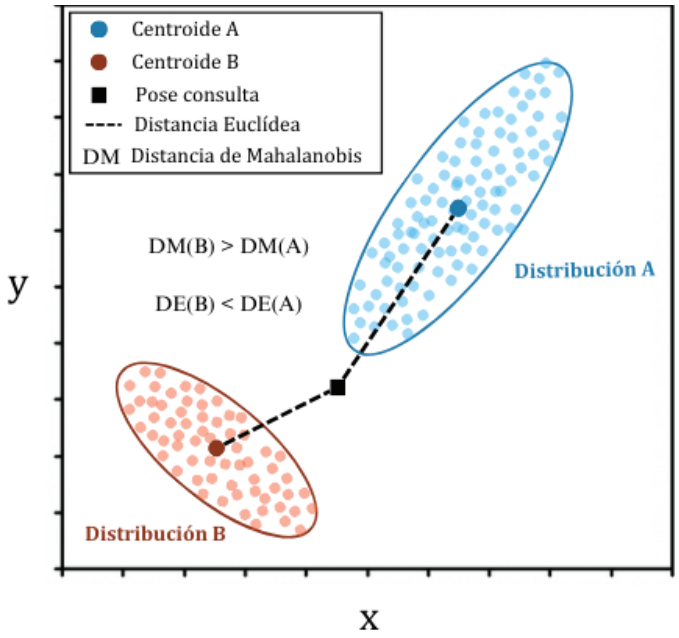


Figura 6.6: Distancia entre un punto y una distribución: comparación de la distancia euclídea y la distancia de Mahalanobis. Se ilustra una situación en la que la pose de consulta se encuentra más cerca del centroide B según la distancia euclidiana, pero más cerca de la distribución A si se considera la dispersión de las distribuciones.

Una opción sería calcular la distancia euclidiana de las poses a los centroides de los grupos. Sin embargo, en este contexto, esta opción no resulta representativa, ya que los centroides no capturan la naturaleza de las distribuciones, como se puede observar en la Figura 6.6. La métrica que mejor se ajusta a nuestras necesidades es la distancia de Mahalanobis [47] [75], la cual evalúa la distancia entre una pose y una distribución teniendo en cuenta la correlación entre las variables dentro de dicha distribución. La correlación entre variables se obtiene a partir de la matriz de covarianza que caracteriza la distribución [76] . La ecuación 6.2 muestra la fórmula de la distancia de Mahalanobis, donde $\vec{x}$ representa la pose de consulta, $\vec{c}$ representa el centroide de la distribución, y Σ representa la matriz de covarianza.

$$dist_{Mah}(\vec{x}, \{A\}) = \sqrt{(\vec{x} - \vec{c}_A)^T (\Sigma_A)^{-1} (\vec{x} - \vec{c}_A)} =$$

$$= \left(\left(\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_\theta \end{bmatrix}_A \right)^T \cdot \begin{pmatrix} \sigma_x & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta \end{pmatrix}_A^{-1} \cdot \left(\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_\theta \end{bmatrix}_A \right) \right)^{-1/2} \tag{6.2}$$

La distancia de Mahalanobis presenta una gran ventaja al trabajar con poses, ya que relaciona de manera adimensional las variables traslacionales con las variables angulares al normalizarlas en base a su covarianza, eliminando así las unidades de las variables.

El cálculo del porcentaje de silueta para un elemento i del GA se basa en calcular la distancia de Mahalanobis de la pose $\vec{x_i}$ a todos los grupos del mapa topológico (GA), es decir, a las distribuciones de pose. Luego, se seleccionan los grupos A y B de GA, donde B es el grupo distinto de A con la menor distancia de Mahalanobis, y A es el grupo al que pertenece el elemento i. Denotando a y b como las distancias de Mahalanobis a los grupos A y B, respectivamente, el valor de SH se calcula de la siguiente manera:

$$SH = \frac{b - a}{max(a, b)} \tag{6.3}$$

A partir del valor de SH para cada elemento del GA, se puede determinar la presencia de *Perceptual Aliasing* (PA). Si SH toma un valor negativo, indica que la pose $\vec{x_i}$ está más cercana a la distribución B, lo que implica una clasificación deficiente en términos de pose, en otras palabras, la presencia de PA. En el caso contrario, cuando SH toma un valor positivo, significa que la pose $\vec{x_i}$ está más cercana a la distribución a la que pertenece que a cualquier otra, lo que indica una clasificación correcta en términos de pose. En la Figura 6.7 se muestra el resultado del porcentaje de silueta sobre el mapa topológico de ejemplo. En el Anexo D se muestra el código que implementa este sistema de evaluación.

6.3.2. Resultados

En esta sección se exponen los resultados de la evaluación de la presencia de Perceptual Aliasing (PA) en relación a la topología de lugares presentada. Específicamente, se realiza una evaluación sobre un mapa topológico o Grafo de Apariencia (GA) dado construido a partir de la secuencia *Seq2_ cloudy1* de *COLD* [67] dataset, donde se aplica un valor de $\tau = 0{,}4$ para aplicar la bipartición recursiva de GA.

Para llevar a cabo esta evaluación, se calcula la puntuación de silueta (*SH*) para cada elemento del *GA*, de acuerdo con lo descrito en la sección anterior. Se obtiene un valor de *SH* para cada elemento del mapa, que puede ser positivo si el elemento está correctamente clasificado, o negativo si el elemento está mal clasificado en términos de pose, indicando así la presencia de *PA*. Con el fin de representar estos resultados, se elabora un diagrama de barras que divide el rango de valores de *SH* en intervalos. Se cuenta el número de valores de *SH* que pertenecen a cada intervalo y se representa esta cantidad en el diagrama. El resultado se ilustra en la Figura 6.7 correspondientes a un mapa topológico obtenido para $\tau = 0{,}3$.

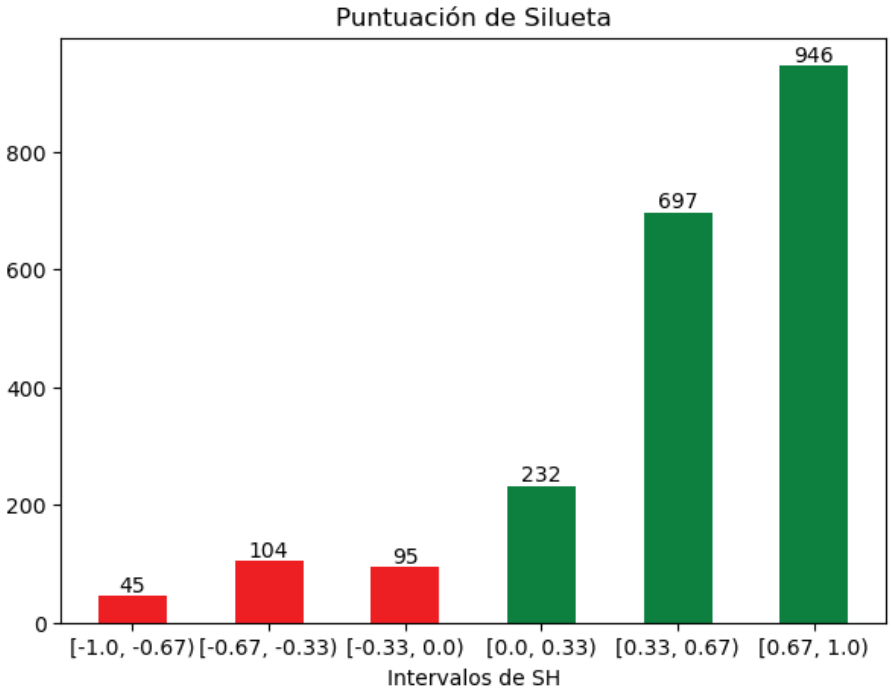


Figura 6.7: Representación del diagrama de barras que muestra los valores de la puntuación de silueta (SH). En color verde se visualizan los elementos correctamente clasificados (1875), los cuales presentan valores positivos de SH, mientras que en color rojo se representan los elementos clasificados de manera incorrecta (244), los cuales exhiben valores negativos de SH.

En el diagrama se observa que el número de elementos correctamente clasificados es significativamente mayor que el número de elementos con *Perceptual Aliasing*, donde el intervalo con mas númerado de elementos es el que toma una mayor putuación de SH. Específicamente, se identifica que un 88.49 % de los elementos están bien clasificados, mientras que un 11.51 % de los elementos presentan clasificaciones erróneas. En conclusión, se puede afirmar que la topología de lugares propuesta en

este estudio evita en gran medida el fenómeno del *Perceptual Aliasing*.

En la Figura 6.8 se muestra un análisis del parámetro τ, que representa el umbral de decisión utilizado en la bipartición recursiva del Grafo de Apariencia. Se realiza un cálculo del porcentaje de elementos correctamente clasificados en la topología de lugares para diferentes valores de τ, basándose en los valores de puntuación de silueta obtenidos.

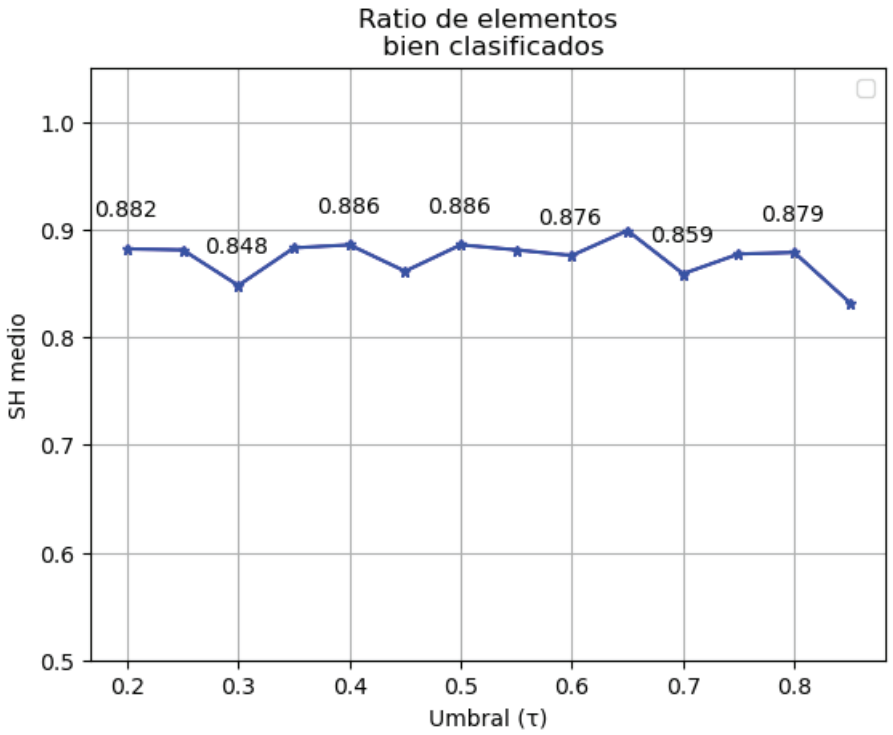


Figura 6.8: Representación de los porcentajes de elementos bien clasificación para distintos valores del parámetro τ. Se calcula este porcentaje a partir del valor de SH medio de los elementos.

Se puede apreciar que el porcentaje de elementos correctamente clasificados no experimenta cambios significativos para los diferentes valores de τ. Esto permite concluir que la naturaleza del Grafo de Apariencia presentado en este trabajo previene el fenómeno de *Perceptual Aliasing*, independientemente de la configuración específica de la bipartición recursiva.

6.4. Rendimiento del VPR sobre la Topología de Lugares

6.4.1. Descripción del sistema de VPR

En esta sección, se describe el sistema de *Visual Place Recognition* (*VPR*) implementado para evaluar la topología de lugares propuesta en este estudio. Se trata de un sistema simple, en el cual se asigna a una consulta dada q, el lugar más similar dentro del mapa topológico. En el Anexo D se muestra el código que implementa este sistema de *VPR*. Para esta evaluación, hemos construido un Grafo de Apariencia (*GA*) a partir del conjunto de datos *Seq2_ cloudy1* del dataset *COLD* [67], que representa el mapa topológico sobre el cual buscamos localizar un conjunto de consultas. Se presenta un ejemplo del resultado obtenido por el sistema de *VPR* sobre el mapa topológico en la Figura 6.9. El conjunto de consultas utilizado en este estudio se compone de 1000 elementos seleccionados de manera aleatoria a partir del datasets *Seq1_ cloudy1*.

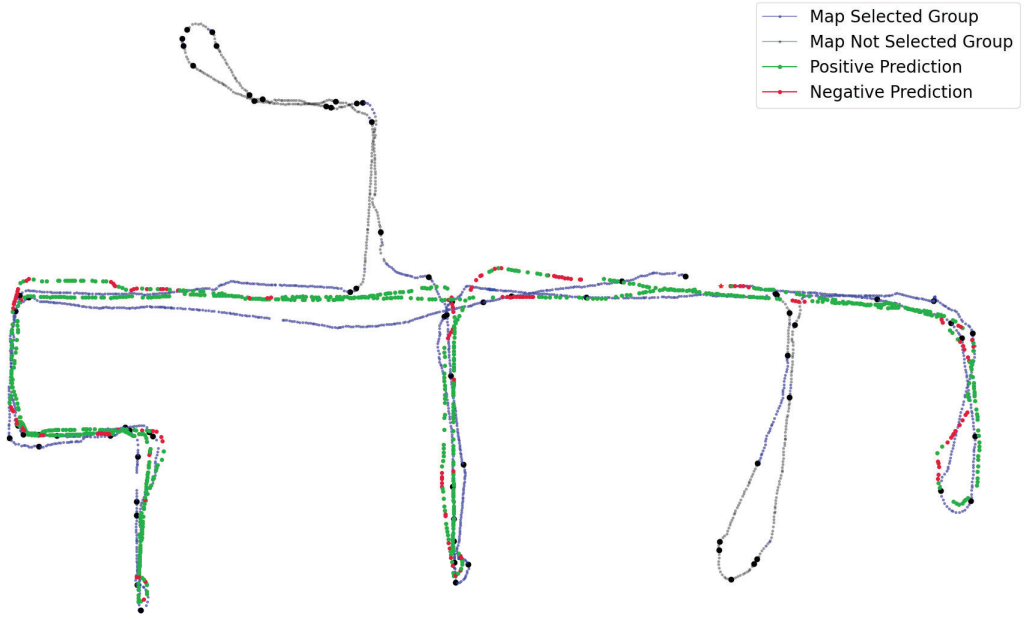


Figura 6.9: Representación del mapa topológico construido a partir de *Seq2_ cloudy1* sobre el que se ilustran las consltas, obtenidas a partir de *Seq1_ cloudy1* [67]. Los elementos del mapa se representan en azul o gris, según se trate de un lugar seleccionado o no. Las consultas aparecen o bien en color verde (para asignaciones correctas) o en rojo (para asignaciones incorrectas) las consultas.

El mapa topológico se define como $GA = \{C_j | j = 0...m\}$, donde m es el número de grupos obtenidos y cada grupo contiene elementos formados por pares descriptor-pose, tal que $Cj = \{(\vec{d_u}, \vec{p_u} | u \in C_j\}$. Del mismo modo, el conjunto de consultas $q = \{(\vec{v_i}, \vec{x_i})\}$ está compuesto por pares descriptor-pose.

En primer lugar, se calcula el vector descriptor global promedio de los elementos de cada grupo, es decir, $\vec{d_j} = \{mean(\vec{d_u}) | u \in C_j\}$, que representa el centroide de los clústeres en términos de apariencia. A continuación, se calcula la distancia entre el vector descriptor de la consulta, $\vec{v_i}$, y cada centroide, $\vec{d_j}$, basándose en la similitud del coseno, y se asigna la consulta al grupo cuyo centroide se encuentra a la menor distancia.

Para corroborar los resultados de la predicción del lugar, se utiliza la información de la pose de la consulta. Se calcula el incremento de pose en *SE(2)* [47] entre cada consulta $\vec{x_i}$ y todas las poses del *GA*. El grupo que contiene el elemento de *GA* más cercano a la consulta en términos de pose se asigna como el grupo auténtico (*ground truth*) de la consulta.

En las siguientes secciones, se presentarán las métricas empleadas para evaluar diversos aspectos del sistema de *VPR*. Se basan en la matriz de confusión multiclase [77], que permite calcular la precisión y sensibilidad del sistema a distintos niveles.

6.4.2. Matriz de confusión multiclase

La matriz de confusión [77] es una técnica de evaluación ampliamente utilizada en algoritmos de clasificación, especialmente en el campo del machine learning. Consiste en una tabla que muestra visualmente el rendimiento del algoritmo al comparar los resultados de clasificación predichos con la clase real, o ground truth (GT), de cada consulta. En la Figura 6.10 se ilustra un ejemplo de matriz de confusión simple, utilizada en clasificaciones binarias, donde se presentan las siguientes situaciones descritas en la sección 6.2.1: Verdaderos Positivos (TP), Verdaderos Negativos (TN), Falsos Positivos (FP) y Falsos Negativos (FN).

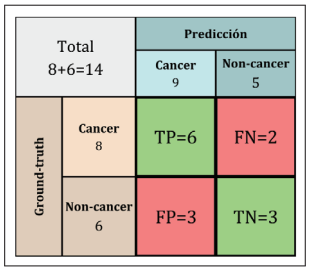


Figura 6.10: Ejemplo de Matriz de Confusión simple para evaluación de un algortimo de detección de cancer a partir de imágenes.

En nuestro caso, el mapa topológico consta de varios lugares a los que asignar una consulta, por lo tanto, utilizamos la matriz de confusión multiclase (CMM), que es una generalización de la matriz de confusión simple. Esta matriz tiene dimensiones $|m|$x$|m|$, donde m es el número de clases. Cada fila de CMM se corresponden con una clase del GT, mientras que cada la columna representa una clase predicha por el algoritmo. En este sentido, si el GT de una consulta pertenece al lugar j, pero el algoritmo predice que pertenece al lugar i, entonces el elemento en la celda $CMM(j, i)$ se incrementa en 1.

En la diagonal de la matriz se almacenan los casos de TP para cada clase, mientras que los demás elementos de la matriz representan los casos en los que el algoritmo comete errores. En un escenario ideal donde el algoritmo predice siempre la clase correcta, CMM solo tendrá valores positivos en la diagonal principal y el resto de las celdas serán 0.

En la Figura 6.11 se representa la matriz de confusión multiclase obtenida para el ejemplo de VPR presentado en la Figura 6.9. El mapa topológico sobre el que se opera cuenta con un total de 74 lugares, es decir, 74 clases.

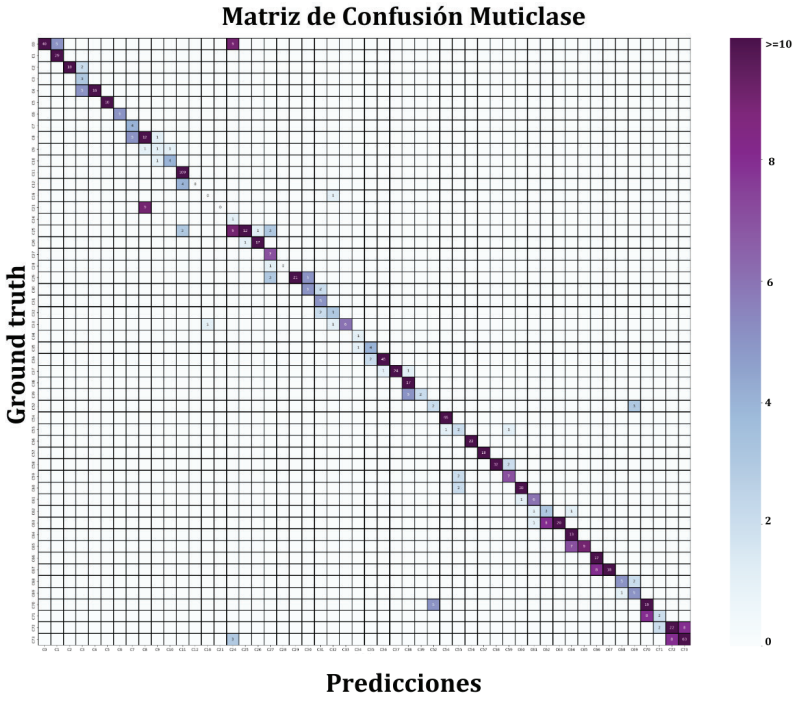


Figura 6.11: Matriz de confusión multiclase para el ejemplo de VPR presentado. Las casillas resaltadas en la matriz muestran un valor correspondiente a la escala de colores ilustrada, mientras que las casillas con un valor de 0 se representan en blanco.

La CMM presentadatiene un tamaño considerable debido a que corresponde a una clasificación de 74 clases. Es por ello que no es posible apreciar en detalle el valor de cada casilla. Sin embargo se puede observar que la mayoría de elementos en la matriz de confusión multiclase (CMM) se concentran en la diagonal principal, la cual exhibe colores más intensos, lo que indica un alto valor de Verdaderos Positivos (TP). Sin embargo, también se observan casos en los que las predicciones no son acertadas, representados por las casillas resaltadas que se encuentran fuera de la diagonal principal. Estas casillas resaltadas indican la existencia de Falsos Positivos (FP) y Falsos Negativos (FN) en la predicción.

A partir de esta matriz, se puede inferir resultados de precisión y sensibilidad del sistema. La ventaja es que nos permite evaluar estos parámetros tanto a nivel general como a nivel de grupos específicos. A continuación se describen, estos dos procesos.

6.4.3. Resultados

En esta sección, se presentan los resultados de la evaluación del sistema de *VPR* descrito, utilizando la matriz de confusión multiclase. Se analizan la precisión y la sensibilidad del sistema, tanto a nivel de grupos como a nivel general. A continuación, se detallan los cálculos necesarios para obtener estos valores.

Se llevaron a cabo tres pruebas para cada conjunto de consultas obtenidos a partir de los datasets *Seq1_ cloudy1*, *Seq2_ night1* y *Seq2_ sunny1* (ver sección 6.1). El mapa topológico se contruye a partir del dataset *Seq2_ cloudy1*.

Precisión y Sensibilidad a nivel de clases

Cada fila y columna de *CMM* representa una clase específica, lo que nos permite calcular la precisión y sensibilidad obtenidas para cada clase al observar la información de la fila y columna correspondiente. Dada una clase i sobre *CMM*, se calcula los valores de *TP*, *FP* y *FN* a partir de la fila i y la columna i de la matriz de confusión de la siguiente manera:

$$
\begin{aligned}
TP_i &= CMM(i,i) \\
FP_i &= \sum_{m}^{j=0} CMM(i,j) - CMM(i,i) \\
FN_i &= \sum_{m}^{j=0} CMM(j,i) - CMM(i,i) \\
TN_i &= \sum_{m}^{k=0}(\sum_{m}^{j=0} CMM(k,j)) \quad | \quad k \neq i, j \neq i
\end{aligned}
\tag{6.4}
$$

El valor de TP_i para cada clase corresponde al valor en la diagonal principal de esa clase. Los valores de FP_i y FN_i se calculan sumando los valores de las columnas o filas respectivamente y restando el valor de la diagonal principal. El valor TN_i representa las veces en las que el algoritmo determina correctamente que la consulta no pertenece a la clase i, por lo que se calcula como la suma de todas las filas y columnas de CMM distintas de i.

Una vez que se han obtenido estos valores, se puede calcular la precisión y el recall para cada clase utilizando las ecuaciones 6.1. Un ejemplo de los resultados obtenidos con esta evaluación se presentan en la Figura 6.12:

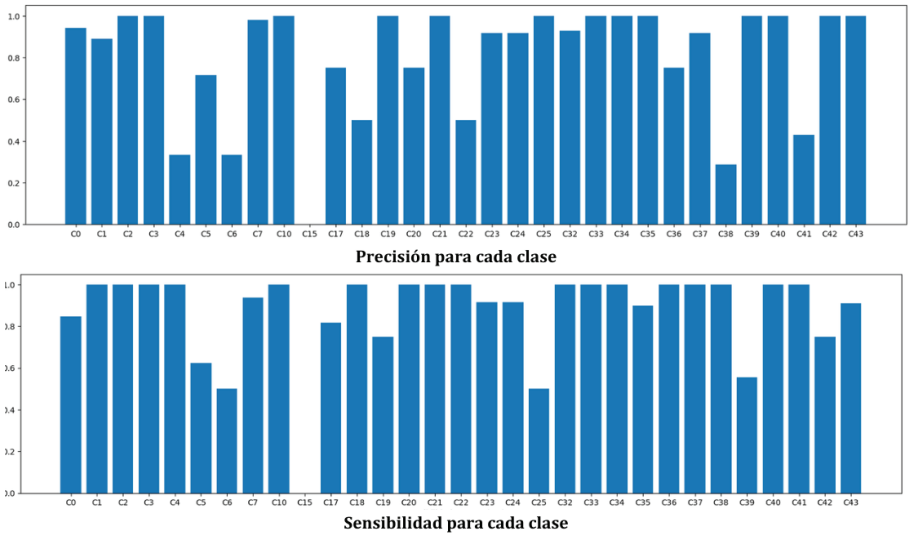


Figura 6.12: Representación de los diagrama de barras para la precisión y sensibilidad por grupos para un mapa topológico con $\tau = 0{,}3$ con consultas tomadas de *Seq1_ cloud1*. En esta representación, únicamente se incluyen aquellos grupos que han sido seleccionados ya sea por la predicción del sistema o por el *ground truth* de la consulta.

Se observa que existen numerosos grupos que muestran un valor máximo de precisión y sensibilidad (*C2*, *C10*, *C21*, etc.), lo que indica que todas las predicciones del algoritmo son correctas para dichos grupos. También se identifican un casos (*C15*) en el que el algoritmo no logra acertar la clase real de la consulta en ningun caso. En estos grupos, el valor de Verdaderos Positivos (*TP*) es cero, lo que resulta en una precisión y sensibilidad nulas.

En base a esta representación gráfica, se concluye que para el ejemplo analizado, el sistema alcanza un rendimiento óptimo en términos de precisión y sensibilidad para la mayoría de los grupos.

A partir de esta gráfica podemos obtener un valor de precisió y sensibilidad media para todos los gruster haciendo el promedio de estos valores. Para el caso de la Figura 6.12 se obtiene una precisión media de 80.11 % y una sensibilidad media del 86.85 % a nivel de grupos. Se presenta en la Figura 6.13 un estudio de la precisión y sensibilidad a nivel de grupos media para distintos valores del parámetro τ. Este valor determina el umbral de decisión a partir del cual el valor *Ncut* de una bipartición en el Grafo de Apariencia (*GA*) se considera óptima. En consecuencia, afecta directamente al tamaño y número de grupos obtenidos (ver sección 5). Para valores de τ bajos se obtienen un menor número de grupos con un número elevado de elementos por grupo, mientras que para valores grandes se obtiene un número mayor de grupos con menos elementos por grupo.

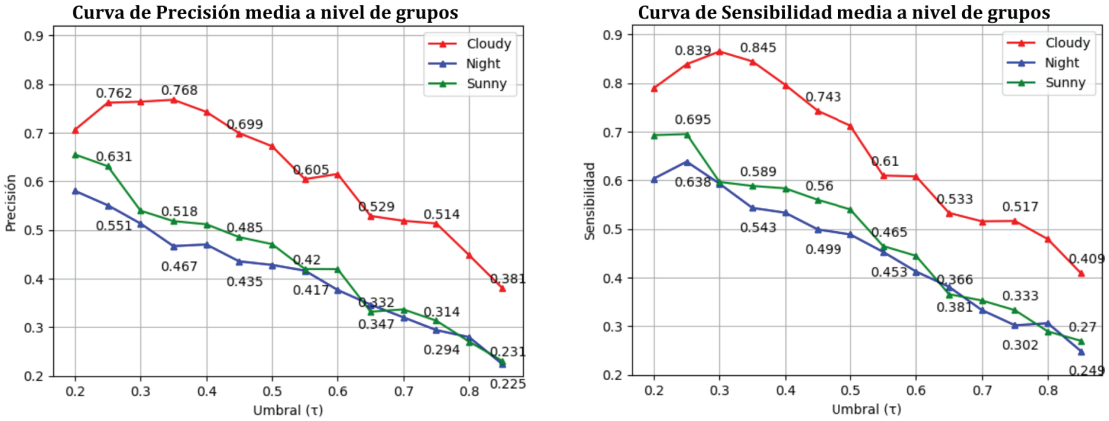


Figura 6.13: Representación de las curvas de precisión y sensibilidad medias a nivel de grupos sobre los datasets de consulta.

En la Figura se puede apreciar que el dataset que consigue mejor desempeño es el relativo a *Seq1_ cloudy1* [67]. Se asume que esto se debe a que el mapa topológico está construido bajo las mismas condiciones de apariencia. Los resultados más destacados en cuanto a precisión y sensibilidad se encuentran en el rango de valores [0.2, 0.5] para el parámetro τ.

Se puede notar que tanto la precisión como la sensibilidad disminuyen a medida que aumenta el valor de τ. Esto se debe a que se generan un mayor número de grupos en el mapa topológico, lo cual aumenta las posibilidades de error en la clasificación. Ademas no se tiene en cuenta el número de consultas que se asigna a cada grupo a la hora de calcular los valores medios. Por lo tanto, esta métrica no representa la calidad global del sistema, simplemente proporciona una idea del rendimiento del algoritmo en términos de precisión y sensibilidad a nivel de grupos para distintos valores de τ.

Precisión, Sensibilidad y Exactitud general

Operando sobre todos los valores de *CMM*, podemos obtener una evaluación general en términos de precisión, sensibilidad y exactitud del sistema. En este caso, debemos calcular los valores de *TP*, *TN*, *FP* y *FN* relativos a todos los grupos. Se puede calcular a partir de las ecuaciones 6.6, sumando el resultado para todas las clases:

$$\begin{aligned} TP &= \sum_{i=0}^{m} TP_i = \sum_{i=0}^{m} CMM(i,i) \\ FP &= \sum_{i=0}^{m} FP_i = \sum_{i=0}^{m} (\sum_{m}^{j=0} CMM(i,j) - CMM(i,i)) \\ FN &= \sum_{i=0}^{m} FN_i = \sum_{i=0}^{m} (\sum_{m}^{j=0} CMM(j,i) - CMM(i,i)) \\ TN &= \sum_{i=0}^{m} TN_i = \sum_{i=0}^{m} (\sum_{m}^{k=0} (\sum_{m}^{j=0} CMM(k,j)) \quad | \quad k \neq i, j \neq i) \end{aligned} \tag{6.5}$$

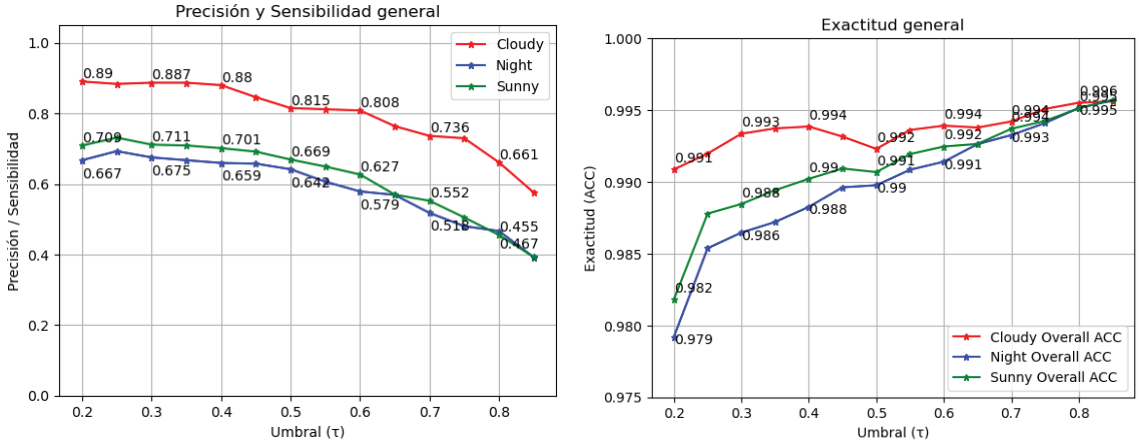
Es importante resaltar que en la matriz de confusión multiclase (CMM), la suma de todos los valores de las filas es igual a la suma de todos los valores de las columnas. En consecuencia, en la ecuación 6.5, el valor de los Falsos Positivos (FP) es igual al valor de los Falsos Negativos (FN). Como resultado, el valor de precisión y sensibilidad descrito en la ecuación 6.1 se iguala en la evaluación general del sistema. La exactitud general (ACC) se calcula de la siguiente manera:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.6}$$

La exactitud es una medida que evalúa la proporción de predicciones correctas realizadas por el algoritmo. Da una visión general del rendimiento global del sistema,

ya que considera tanto las predicciones positivas como las negativas.

A continuación, se presenta el resultado de la evaluación del sistema de *VPR*. Se calcula el valor de precisión, sensibilidad y exactitud generales aplicando el sistema de *VPR* para diferentes mapas topológicos, donde se toman varios valores del umbral τ. Este umbral determina el valor óptimo de *Ncut* necesario para realizar una bipartición efectiva, como se describe en el capítulo 5. En consecuencia, influye en el número de grupos y su tamaño. En este sentido, valores altos de τ generan numerosos grupos con pocos elementos en cada uno, mientras que valores bajos producen un menor número de grupos con más elementos. En la Figura 6.14, se muestra la curva de precisión, sensibilidad y exactitud general para los tres dataset de consulta descritos previamente.



a b

Figura 6.14: Curva de Precisión/Sensibilidad y Exactitud generales para el sistema de *VPR*.

En general, se obtienen buenos resultados para los tres conjuntos de datos utilizados, siendo especialmente destacables los valores elevados de exactitud. Se observa que el rendimiento del dataset *Seq1_ cloudy1* es superior en ambas gráficas, obteniendo resultados superiores al 80 % en precisión y sensibilidad, así como una exactitud por encima del 99 %.

Se observar que el desempeño del dataset *Seq1_ cloudy1* es el mejor en ambas gráficas, que obtiene resultados por encima del 80 % en precisión y sensibilidad y por encima del 99 % en exactitud. Por lo general, se obtienen buenos resultados para los tres datasets empleados, donde destaca sobre todo los valores altos de exactitud.

Se observa que tanto la precisión como la sensibilidad generales disminuyen a medida que aumenta el umbral. Esto se debe a que, al haber más grupos con menos

elementos, es más probable que el algoritmo de *VPR* cometa errores en la clasificación.

Por otro lado, la curva de exactitud sigue una tendencia inversa, presentando valores mínimos para umbrales bajos y valores máximos para umbrales altos. Esto se debe a que, a medida que aumenta el número de grupos, se puntúan más casos de Verdadero Negativo (TN) en la matriz de confusión multiclase, lo que resulta en una mayor exactitud. Cuando hay muy pocos grupos, los errores cometidos por el algoritmo tienen un mayor impacto en el valor de exactitud.

Idealmente, se busca alcanzar un equilibrio entre el número de grupos obtenidos y el número de elementos en cada grupo, para obtener valores promedio de precisión, sensibilidad y exactitud general. A partir de las gráficas presentadas, se concluye que el umbral óptimo del valor τ se encuentra en el intervalo [0.2, 0.5].

6.5. Conclusiones del capítulo

En este capítulo se han presentado los métodos utilizados para evaluar las herramientas propuestas en este trabajo. En primer lugar, se describe el dataset *COLD* y los diversos enfoques utilizados para llevar a cabo las pruebas. Se evalúa por separado la creación del Grafo de Apariencia (*GA*), que se refiere a la construcción de la matriz de co-visibilidad, y la topología de lugares obtenida, donde se evalúa la presencia de *Perceptual Aliasing* (*PA*) y el rendimiento del mapa topológico aplicado a un sistema de *Visual Place Recognition* (*VPR*).

En la evaluación de la co-visibilidad, se determina la precisión y sensibilidad presentadas por el método para distintos valores del umbral μ. Los resultados concluyen que el método alcanza valores más altos de precisión y sensibilidad en el rango de valores [15, 30] de μ. Es destacable el buen rendimiento del parámetro de precisión, que alcanza su máximo a partir de μ=15, lo que indica que el método no comete errores de clasificación positiva. A partir de estos valores, se calcula la curva de precisión-sensibilidad y se obtiene el área bajo esta curva (AUC), que es una medida de la calidad global del sistema. Se obtienen valores superiores al 90 % para los datasets utilizados, lo que indica un rendimiento destacable de la herramienta de detección de co-visibilidad.

Es importante destacar que estos resultados están sujetos a la calidad del conjunto de datos empleados, el cual se compone de tres datasets controlados en los que se seleccionan conjuntos de fotos que se sabe que son co-visibles. Esto se hace debido a la imposibilidad de determinar el valor verdadero o *ground truth* de la co-visibilidad a partir de otra información, como la pose. En consecuencia, estos resultados están sujetos a condiciones claras de co-visibilidad, quedando pendiente evaluar el método para un dataset más complejo donde se conozca el valor de *ground truth* .

La evaluación de la topología de lugares propuesta en este trabajo se hace en dos partes. En primer lugar, se evalúa la presencia de *Perceptual Aliasing* (*PA*) en los distintos lugares en la topología. Para ello se define la herramienta de puntuación de silueta (*SH*) que se fundamenta en el cálculo de la distancia de Mahalanobis para cada elemento de un mapa topológico dado, con el fin de determinar si está bien clasificada en su grupo en términos de pose.

Como resultado de esta prueba, se presenta un ejemplo de diagrama de barras que muestra el número de elementos clasificados correctamente e incorrectamente en función de su valor de *SH*. También se realiza un estudio del parámetro τ para analizar cómo afecta al valor de *SH* medio, el resultado mostraba que el porcentaje de elementos bien clasificados no depende de este parámetro. Se concluye que la topología de lugares propuesta evita en gran medida el *PA* gracias a las propiedades del Grafo de Apariencia.

Por último se evalúa el rendimiento de un sistema de *VPR* simple sobre un mapa topológico dado. Se describe el sistema implementado y se introduce la matriz de confusión multiclase como método de evaluación para determinar la eficacia en la clasificación de lugares. A partir de esta matriz, se obtienen dos métricas: una relativa a la precisión y sensibilidad a nivel de grupos, y otra relativa a la precisión, sensibilidad y exactitud globales del método.

Para la evaluación a nivel de grupos, se presenta un ejemplo con un diagrama de barras que muestra la precisión y sensibilidad obtenidas en cada grupo de un mapa topológico dado. A partir de esta métrica, se calcula el valor medio de precisión y sensibilidad a nivel de grupos, y se realiza un estudio del parámetro τ de la bipartición recursiva para comparar diferentes valores de precisión y sensibilidad promedio a nivel de clúster.

A partir de la matriz de confusión multiclase, también se obtienen los valores relativos a la precisión, sensibilidad y exactitud globales del sistema. Nuevamente, se evalúan estos parámetros para diferentes mapas topológicos variando el parámetro τ. La conclusión obtenida de esta evaluación es que el rango de valores óptimo para este parámetro es [0.2, 0.5], donde se obtienen valores superiores al 80 % en precisión y sensibilidad globales, y superiores al 99 % en términos de exactitud para el mejor caso.

En conclusión, las pruebas de evaluación descritas en este capítulo proporcionan resultados destacables en la detección de la co-visibilidad y la presencia de Perceptual Aliasing en el mapa topológico. En cuanto al rendimiento del sistema de *VPR*, se obtienen valores aceptables en términos de precisión y sensibilidad tanto a nivel de grupos como en general. Por lo tanto, se cumplen los objetivos propuestos para este trabajo.

Capítulo 7

Conclusiones

Contenido

Sinopsis

En este capítulo se presentan las conclusiones derivadas de la topología de lugares desarrollada en este trabajo, haciendo enfasis en las ventajas y limitaciones del método, y se destacan los resultados obtenidos en la evaluación. Además, se exploran los posibles trabajos futuros que surgen de esta investigación.

7.1. Conclusiones

En el presente TFG, se ha desarrollado un estrategia para crear una topología de lugares similares en apariencia para sistemas de Localización basada en Apariencia (*AbL*). Tras su implementación y evaluación, se han extraído una serie de conclusiones:

- Nuestra propuesta de topología de lugares sienta las bases para algoritmos de *AbL*, donde la representación del entorno consiste en un Grafo de Apariencia donde los nodos contienen información de apariencia y pose. La topología surge al dividir este grafo en distintos lugares donde el criterio de la clasificación se basa en la similitud en apariencia y en la detección de la co-visibilidad entre elementos del Mapa de Apariencia (*AM*). Este Grafo de Apariencia es una representación del entorno simple y fácil de operar, evitando así la representación 3D del entorno, lo que hace más eficiente la tarea de localización.

- Nuestra propuesta para crear topologías de lugares es adecuada para conjuntos de datos desordenados en su origen, lo que significa que las imágenes del Mapa de Apariencia no necesitan seguir un orden específico. Además, el sistema presentado no requiere un número predeterminado de grupos para clasificar los elementos del Mapa de Apariencia; sino que el número de grupos obtenidos depende de la naturaleza de los datos y los parámetros de configuración. Esto proporciona flexibilidad y facilidad de aplicación en diversos entornos, lo que supone una ventaja significativa en comparación con otros sistemas de *AbL*.

- Se aborda de manera satisfactoria el problema del Perceptual Aliasing (*PA*), que representa una limitación importante en la mayoría de los sistemas de *AbL* basados en la similitud entre descriptores de imágenes para la localización. Este problema se refiere a la clasificación errónea de elementos del Mapa de Apariencia en términos de pose, debido a una alta similitud en sus descriptores de apariencia. La solución propuesta consiste en la detección de co-visibilidad entre imágenes, que es un parámetro fundamental en la obtención de los lugares de la topología. Los resultados de evaluación de la topología de lugares demuestran que se evita en gran medida el fenómeno del *PA*, es decir, la mayoría de los elementos del Mapa de Apariencia se clasifican correctamente en lugares en términos de pose y apariencia.

- Se presenta la detección de co-visibilidad como una herramienta para relacionar imágenes geométricamente y determinar la observación de una escena común como herramienta principal para evitar el Perceptual Aliasing. Se obtienen resultados prometedores en las pruebas de evaluación realizadas, logrando la máxima precisión. Se concluye que este sistema es robusto frente a errores de

clasificación positiva, aunque presenta cierto margen de error en la clasificación negativa.

- Se evalúa el rendimiento de la topología de lugares propuesta para un sistema de Visual Place Recognition (*VPR*) simple. Los resultados obtenidos son considerablemente buenos en términos de precisión, sensibilidad y exactitud del sistema, lo que demuestra que este sistema es fácilmente aplicable para tareas de *VPR*.

- Una de las limitaciones fundamentales de esta propuesta radica en la necesidad de contar con un mapa de apariencia (AM) de imágenes geolocalizadas, lo cual normalmente requiere de un robot móvil para construir este conjunto de datos.

- Se ha comprobado que el sistema de detección de co-visibilidad reduce notablemente su rendimiento en entornos sin detalles, texturas o elementos característicos, como entornos virtuales simulados. En ausencia de elementos característicos detectables, el número de descriptores locales obtenidos es menor de lo esperado, lo que disminuye la eficacia de la detección de co-visibilidad, ya que este método se basa en la detección de estos descriptores. Sin embargo, esta limitación no se presenta en entornos reales, ya que suelen contar con numerosos matices y detalles que caracterizan los diferentes lugares. En conclusión, el método de detección de co-visibilidad opera de manera adecuada principalmente en entornos reales.

En resumen, este TFG ha desarrollado una estrategia para crear una topología de lugares similares en apariencia para sistemas de Localización basada en Apariencia. La propuesta ha demostrado ser efectiva en la representación del mundo a través de un Grafo de Apariencia y en la detección de co-visibilidad para evitar el Perceptual Aliasing. Además, se ha evaluado su rendimiento en tareas de Visual Place Recognition, obteniendo buenos resultados en términos de precisión, sensibilidad y exactitud. Aunque existen limitaciones relacionadas con la necesidad de un mapa de apariencia geolocalizado y el rendimiento en entornos sin elementos característicos, esta propuesta cumple con los objetivos planteados y muestra su potencial en aplicaciones de Localización basadas en Apariencia.

7.2. Líneas Futuras

Tras la finalización del presente TFG, se proponen una serie de líneas de investigación para dar continuidad al trabajo realizado. Estas propuestas se detallan a continuación:

- Implementación de la topología de lugares en un sistema de Visual Place Recognition (VPR) para una aplicación real práctica, seguida de su posterior evaluación y análisis de resultados.

- Evaluación de la detección de co-visibilidad en un dataset más amplio que incluya casos de co-visibilidad no ideales, y exploración de posibles estrategias para reducir el tiempo de ejecución de este proceso.

- Con la topología de lugares propuesta se logra una clasificación precisa de los elementos del Mapa de Apariencia en términos de apariencia y pose, evitando el fenómeno del Perceptual Aliasing. En consecuencia, se propone la implementación de un sistema de Localización basada en Apariencia (AbL) que permita una localización métrica óptima utilizando la topología de lugares propuesta, como se describe en la sección 3.3.

- Realizar un estudio exhaustivo sobre el potencial del Grafo de Apariencia para tareas de localización métrica. Como se mencionó en la sección 2.4, es posible obtener la traslación y rotación relativas entre las poses de las cámaras a partir de la matriz fundamental que une dos imágenes co-visibles. La inclusión de esta información en el Grafo de Apariencia establece una propuesta de localización métrica que permite inferir las poses a partir de las matrices fundamentales de los elementos del mapa que presentan similitud cercana.

Parte IV

Apéndices

Apéndice A

Construcción del Grafo de Apariencia - Jupyter Notebook

Sinopsis

En el presente apéndice se presenta el código correspondiente a la generación del Grafo de Apariencia. Específicamente, se lleva a cabo la implementación de las matrices de similitud y co-visibilidad, así como la creación de un archivo que contiene información relevante sobre el Grafo de Apariencia (GA).

El código se encuentra en un entorno de trabajo Jupyter Notebook [78], donde se han dispuesto las diferentes funciones en celdas de código independientes. Entre estas celdas de código, se incluye texto explicativo que facilita la comprensión del proceso. Es importante destacar que el lenguaje de programación utilizado es Python

Construcción del Grafo de Apariencia

Este código respone al proceso de creación de un Grafo de Apariencia a partir de un Mapa de Apariencia como un dataset. Este dataset está compuesto por imágenes geolocalizadas, representadas por medio de un descriptor de apariencia.

En este código se calculan las mastrices de similitud (S) y co-visibilidad (C), junto con un fichero json con la información de poses. Estos elementos se emplean en el código 'Corte_Normalizado' para elavorar el Grafo de Apariencia.

Librerías

In [ ]:
```
import json
import h5py
import cv2 as cv
import os
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
```

Matriz de Co-visibilidad - Funciones

En esta parte se definen las dos funciones para el cálculo de la matriz de co-visibilidad.

- La función 'CheckCovisibilidad' detecta de co-visibilidad entre dos imágenes.
- La función 'occupancy' calcula el porcentaje de co-visibilidad una vez detectada.

In [ ]:
```
def CheckCovisibilidad(img1,img2, thr):
# Esta función detecta la presencia de co-visibilidad a partir de dos imágenes en escala de grises.
# Se calcula los descriptores SIFT de las imágenes y se emparejan con el método Brutal-Forcer Matcher, creando correspondencias e
# A partir de estas correspondencias se calcula la Matriz Fundamental con el algoritmo de RANSAC.
# Se determina la existencia de co-visibilidad si el número de inliers de la matriz fundamental supera el umbral trh.
#   -Input:
#       img1, img2  ---> Imágenes a comparar en escala de grises.
#       thr         ---> Umbral para determinar la co-visibilidad.
#   -Return:
```



```
#       Cov_ENABLE  ---> Booleano indica la presencia de co-visibilidad.
#       pts1, pts2  ---> Listas con posiciones sobre los planos imágen de inliers emparejados.

    #Return variables:
    Cov_ENABLE= False   #Covisibilidad negativa por defecto
    pts1=[]
    pts2=[]

    # SIFT descriptor class:
    sift=cv.xfeatures2d.SIFT_create()

    # Calcula descriptores SIFT de las imágenes, devuelve un conjunto de descriptores de características (d)
    # y posiciones en el plano imágen, keypoints (kp)
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)

    #Emparejamiento de descriptores por el método Brutal-Forcer-Matcher (BFM)
    if len(kp1) > 0 and len(kp2) > 0:

        bf = cv.BFMatcher(cv.NORM_L2, crossCheck = True)    # Clase BFM con correlación cruzada positiva (crossCheck=True).
        matches = bf.match(des1,des2)                       # Emparejamiento (matching) de descriptores de características (d).
        matches=sorted(matches, key= lambda x:x.distance)   # Se ordenan los matches de menor a mayor distancia entre pares de d
        if len(matches)>100: matches=matches[:100]          # Se opera con los 100 matches con menor distancia, las mejores corr

        # Obtenemos keypoints con correspondencia:
        for mat in matches:
            pts1.append(kp1[mat.queryIdx].pt)
            pts2.append(kp2[mat.trainIdx].pt)

        pts1=np.int32(np.array(pts1))   # Convertimos a np array
        pts2=np.int32(np.array(pts2))

        # Calculo de la Matriz Fundamental (F) con algoritmo de RANSAC
        if len(pts1)>=50:
            F, mask = cv.findFundamentalMat(pts1,pts2,cv.FM_RANSAC,3,0.99,10) # Calculo Matriz Fundamental, nº iteraciones en alg

            # Seleccionamos inliers, pares de keypoints representados por la Matriz Fundamental obtenida
            pts1 = pts1[mask.ravel()==1]
            pts2 = pts2[mask.ravel()==1]

            # Detección de co-visibilidad en base al umbral de número de inliers
            if len(pts1)>thr:
                Cov_ENABLE=True
```

```
    return Cov_ENABLE, pts1, pts2
```

In [ ]:
```
def occupancy(pts1,pts2,cell_x,cell_y,size):
# Esta función calcula el porcentaje de co-visibilidad entre dos imágenes a partir de la posición de los inliers.
# Se define una regilla de ocupación para cada imágen, las celdas que contengan uno o mas inliers se marcan como ocupadas.
# El porcentaje de ocupación se calcula como el número de celdas ocupadas entre el número total de celdas.
# Se escoge el mayor valor de ocupación entre las imágenes para establecer el porcentaje de co-visibilidad para obtener una matr
#   -Input:
#       pts1, pts2      ---> Listas de inliers, posiciones de características sobre el plano imágen.
#       cell_x, cell_y  ---> Tamaño de la celda de ocupación
#       size            ---> Tamaño de las imágenes originales (x,y)
#   -Return:
#       w               ---> Porcentaje de ocupación, valor de co-visibilidad del arco que une los nodos.

    # Definimos celdas de ocupación para cada imágen:
    occupancy1=np.array([[0 for i in range(cell_y)] for j in range(cell_x)])
    occupancy2=np.array([[0 for i in range(cell_y)] for j in range(cell_x)])

    # Recorremos listas de inliers y marcamos las celdas ocupadas
    iter=len(pts1)
    for i in range(iter):
        #Occupancy 1:
        x1=pts1[i][0]   # Posición x en plano imágen.
        y1=pts1[i][1]   # Posición y en plano imágen.
        # Ajustamos al tamaño de la celda de ocupación:
        x1_idx=np.int32((x1*cell_x)/size[1])
        y1_idx=np.int32((y1*cell_y)/size[0])
        # Marcamos la celda como ocupada:
        occupancy1[x1_idx][y1_idx]=1

        #Occupancy 2:
        x2=pts2[i][0]   # Posición x en plano imágen.
        y2=pts2[i][1]   # Posición y en plano imágen.
        # Ajustamos al tamaño de la celda de ocupación:
        x2_idx=np.int32((x2*cell_x)/size[1])
        y2_idx=np.int32((y2*cell_y)/size[0])
        # Marcamos la celda como ocupada:
        occupancy2[x2_idx][y2_idx]=1

    # Suma de las celdas ocupadas:
    sum1=0
    sum2=0
```



```
    for i in range(cell_y):
        for j in range(cell_x):
            if(occupancy1[j][i]==1):
                sum1+=1
            if(occupancy2[j][i]==1):
                sum2+=1

    # Calculo del porcentaje de ocupación como el número de celdas ocupadas entre le número total de celdas:
    occupancy_size=cell_x*cell_y
    w1=np.float64(sum1/occupancy_size)
    w2=np.float64(sum2/occupancy_size)

    # Seleccionamos el valor máximo de ocupación como porcentaje de co-visibilidad entre las imágenes.
    w=np.max([w1,w2])   # Se hace para conseguir una matriz de adyacencia simétrica para el Grafo de Apariencia.

    return w
```

Matriz de Similitud - Funciones

En esta parte se define la función que calcula la matriz de similitud y la función que carga los descriptores globales.

In [ ]:
```
def my_similarity(data_set,sim_f):
# Esta función calcula la matriz de similitud para un conjunto de descriptores globales de la imágen.
# Se selecciona por orden todos los pares de descriptores del dataset y se evalúa la similitud con la función 'sim_f'
#   -Input:
#       data_Set    ---> Dataset del mapa de apariencia, se seleccionan los descriptores globales.
#       sim_f       ---> Función de similaridad para el cálculo de distancia entre descriptores.
#   -Return:
#       sim_matrix  ---> Matriz de similitud

    # Selección de descriptores globales del dataset:
    features=np.array(data_set['features'])
    size=len(features)  # Número de descriptores

    # Calculo de número de iteraciones para mostrar proceso
    n_iter=(size**2 +size)/2
    cont=0

    # Inicializar matriz de similitud cuadrada a partir del número de descriptores
    sim=[[0 for i in range(size)] for j in range(size)]
```

```
    # Bucle compara pares de descriptores:
    for i in range(0,size):
        for j in range(i,size):
            # Se aplica la función de similitud indicada
            similarity=sim_f(features[i],features[j])

            # Guarda resultado en matriz de similitud simétrica
            sim[i][j]=similarity
            sim[j][i]=similarity

            del similarity

            # Print proceso
            print('Process: ',np.round(((100*cont)/n_iter),2),' %',end='\r')
            cont+=1

    return np.array(sim)
```

In [ ]:
```
def get_descriptor(general_path):
    # Función para leer los descriptores globales de un fichero h5py
    descriptors=[]
    for suffix in os.listdir(general_path):
        f=h5py.File(general_path+suffix,'r')
        descriptors+=f['features']
    return np.array(descriptors)
```

Testing

Cargar Dataset

Esta parte genera un fichero .json que representa el Grafo de Apariencia.

Guarda información del nombre de las imágenes, poses, descriptores de apariencia y poses odométricas.

In [ ]:
```
# Indicar ruta con fichero json del dataset
dataset_path=".../map/"

# Diccionario para guardar dataset:
```



```
map_dataset=dict(im_paths=[], poses=[], odom_poses=[], features=[])

# Cargar poses y paths:
for json_suffix in os.listdir(dataset_path+"Poses/"):    # Carpeta 'Poses' del dataset_path
    json_file=json.load(open(dataset_path+"Poses/"+json_suffix))
    samples=len(json_file['im_paths'])
    map_dataset['im_paths']+=json_file['im_paths']
    map_dataset['poses']+=json_file['poses']
    map_dataset['odom_poses']+=json_file['odom_poses']

# Cargar descriptores de la imagen:
descr_path=dataset_path+'Descriptors/'+'netVLAD/'       # Carpeta 'Descriptors' del dataset_path
features= get_descriptor(descr_path).tolist()           # Función para obtener descriptores de la imágen
map_dataset['features']+=features                       # Guardar descriptores en 'map_dataset'

print('Tamaño del dataset: ', len(map_dataset['features']))
```

```
Tamaño del dataset:  2146
```

Cálculo de Matriz de Similitud

In [ ]:
```
# Generar fichero .npy para almacenar la matriz de similitud
similitud_path='.../Utils/sim_matrix.npy'
if not os.path.exists(similitud_path):
    with open(similitud_path,'x') as fp:
        pass
else:
    fdelet=open(similitud_path,'w')
    fdelet.close()
```

In [ ]:
```
# Definición de función de similitud, usamos la similitud del coseno:
cos_sim=lambda x,y:np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))

# Calculamos matriz de similaridad:
sim_matrix=my_similarity(map_dataset,cos_sim)

# Guardar la matriz de similitud:
np.save(similitud_path, sim_matrix)
```

```
Process:  99.99  %
```

Cálculo de Matriz de Co-visibilidad

In [ ]:
```
# Generar fichero .npy para almacenar la matriz de co-visibilidad
covisibilidad_path='.../Utils/cov_matrix.npy'
if not os.path.exists(covisibilidad_path):
    with open(covisibilidad_path,'x') as fp:
        pass
else:
    fdelet=open(covisibilidad_path,'w')
    fdelet.close()
```

In [ ]:
```
# Indicar ruta de las imágenes del dataset
img_path= ".../map/Images"

# Guardamos lista con nombres de las imágenes
im_dir=[]
for suffix in sorted(os.listdir(img_path)):
    im_dir.append(suffix)

im_size=len(im_dir) # Número de imágenes
```

Inicializamos matrices de co-visibilidad binaria y de pesos.

In [ ]:
```
overlap=np.zeros([im_size, im_size], dtype=np.bool) # Matriz de co-visibilidad binaria, se guarda el resultado de la deteccion d
cov_matrix=np.zeros([im_size, im_size], dtype=np.float64)    # Matriz de co-visibilidad real, se guarda el porcentaje de co-visi
```

Calculo de la matriz de co-visibilidad.

Se calcula el porcentaje de co-visibilidad para todos los pares de imágenes del dataset.

In [ ]:
```
# Calculo del número de iteraciones para mostrar la evolución del proceso
total_iter=(np.math.factorial(im_size)/(np.math.factorial(2)*np.math.factorial(im_size-2)))+im_size
cont_iter=0

# Fijar valor del umbral de número de inliers para determinar la co-visibilidad
thr=20

# Bucle que compara pares de imágenes:
```



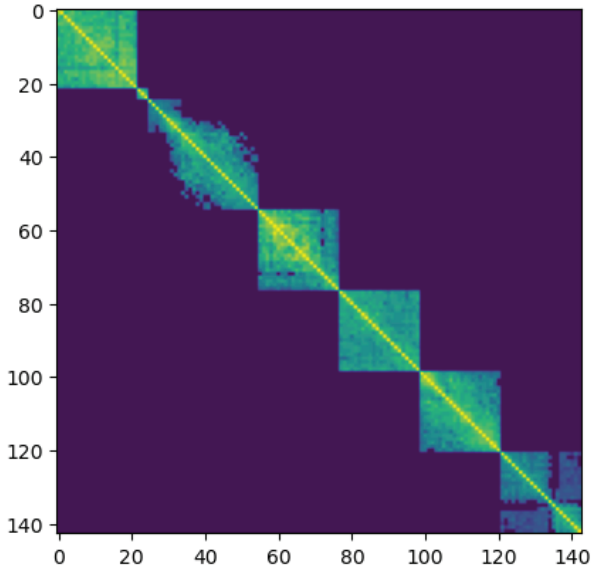
```
for i in range(im_size):
    # Lectura de imágen 1:
    suffix1=im_dir[i]
    im1=cv.imread(img_path+'/'+suffix1)

    for j in range(i,im_size):
        # Lectura de imágen 2:
        suffix2=im_dir[j]
        im2=cv.imread(img_path+'/'+suffix2)

        if i==j:    # Si las imágenes a comparar son la misma (mismo indice) directamente se determina co-visibilidad = 1
            overlap[i][j]=True
            cov_matrix[i][j]=1

        else:
            # Detección de co-visibilidad:
            ovlp,pts1,pts2=CheckCovisibilidad(im1,im2,thr)
            overlap[i, j]=ovlp
            overlap[j, i]=ovlp

            # Si se detecta la co-visibilidad se procede al cálculo del porcentaje de ocupación:
            if(ovlp):
                tam=im1.shape   # Tamaño de imágen original
                cell_x=6        # Tamaño en x de celda de ocupación
                cell_y=np.int32((tam[0]/tam[1])*cell_x) # Tamaño en y de celda de ocupación, relativo a dimensiones de la imágen

                # Calculo de porcente de ocupación
                cov_ratio=occupancy(pts1,pts2,cell_x,cell_y,tam)

                # Guarda porcentaje de ocupación en matriz de co-visibilidad simétrica
                cov_matrix[i, j]=cov_ratio
                cov_matrix[j, i]=cov_ratio

                del cov_ratio

            # Si no se detecta la co-visibilidad se asigna valor 0:
            else:
                cov_matrix[i, j]=0
                cov_matrix[j, i]=0

            del ovlp, pts1,pts2

        # Print porcentaje de ejecución:
        cont_iter+=1
```

```
        if j%10==0: print('Proceso: ',np.round((cont_iter/total_iter)*100,decimals=6),'%',end='\r')


    # Guardar matriz de co-visibilidad:
    np.save(covisibilidad_path,cov_matrix)
```

```
Proceso:  99.951437 %
```

In [ ]:
```
w=np.load(covisibilidad_path)

plt.imshow(w)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1c7463e4d48>
```



Post-procesamiento

En esta parte se eliminan los nodos aislados de la matriz de co-visibilidad. Son nodos no conectados a ningun otro nodo, no necesarios para el mapa topológico.

Se eliminan tambien de la matriz de similitud y del dataset del mapa con las poses. Se guardan las versiones definitivas.

In [ ]:
```
# Se determina los índices cuyas filas suman un valor mayor que 1 en la matriz de co-visibilidad, lo que indica que están conecte
indx = np.argwhere(cov_matrix.sum(0) > 1).reshape([-1])
tam=cov_matrix.shape[0]
print('Hay %d indices malos. Limpiamos' % (tam - len(indx)))

# Se guardan los índices de nodos buenos en las matrices
cov_matrix = cov_matrix[indx][:, indx]
sim_matrix = sim_matrix[indx][:, indx]

# Guardamos los indices buenos en el dataset
save_dataset=dict(im_paths=[], poses=[], odom_poses=[], features=[])    # Nuevo diccionario para guardar mapa definitivo
for key in map_dataset.keys():
    arr=np.array(map_dataset[key])
    arr=arr[indx]                   # Seleccion de elementos con índices buenos
    save_dataset[key]=arr.tolist()  # Guardamos listas en nuevo diccionario
```

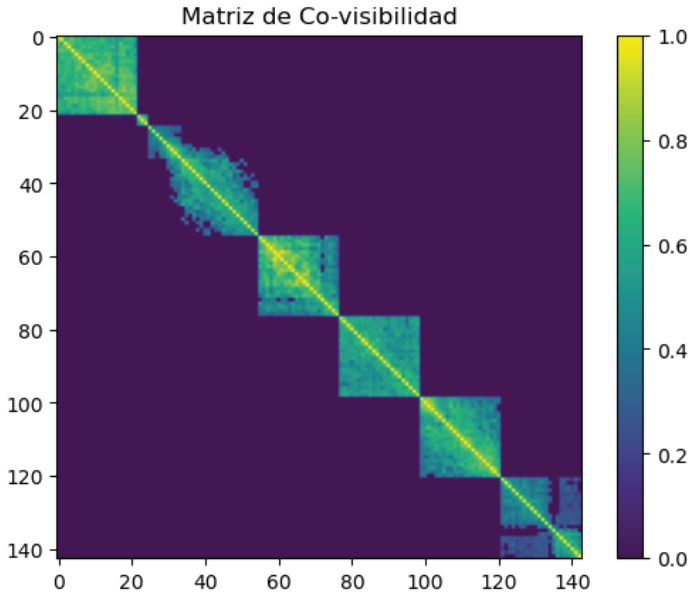
```
Hay 0 indices malos. Limpiamos
```

Guardamos resultado

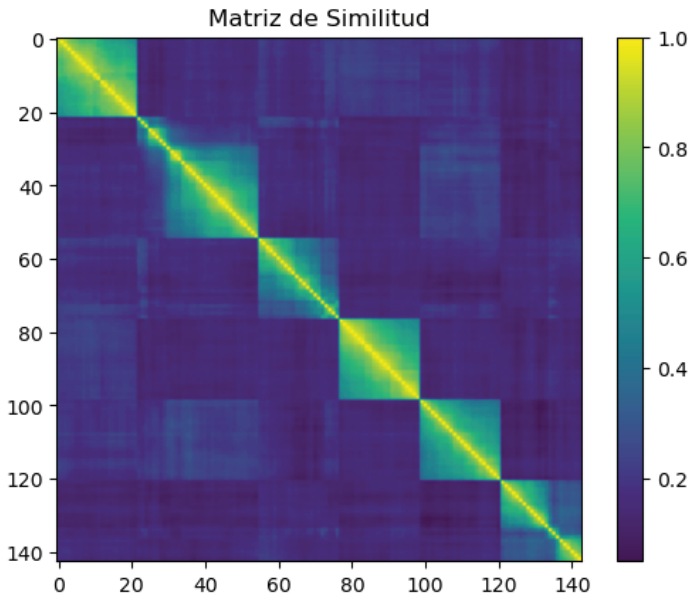
In [ ]:
```
save_json=".../Utils/Grafo_Apariencia.json"

with open(save_json, "w") as outfile:
    json.dump(save_dataset, outfile)

np.save(covisibilidad_path,cov_matrix)
np.save(similitud_path,sim_matrix)
```

Show matrix

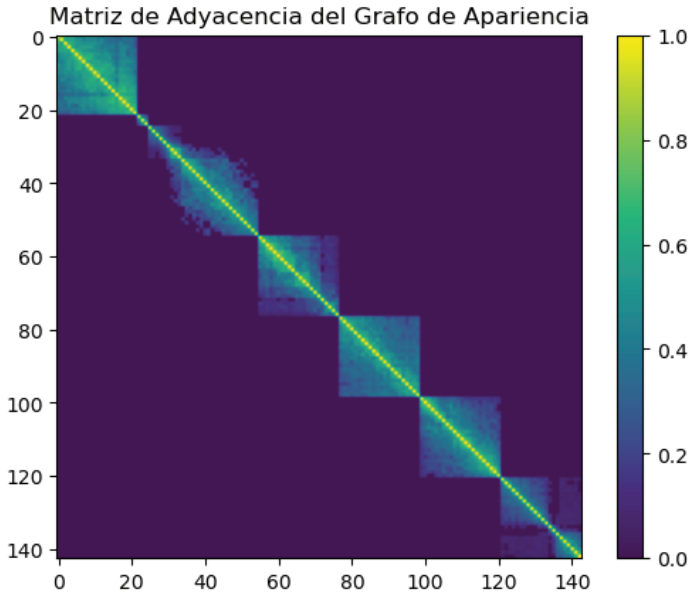
In [ ]:
```
plt.imshow(cov_matrix)
plt.title('Matriz de Co-visibilidad')
plt.colorbar()
plt.show()
```



In [ ]:
```
plt.imshow(sim_matrix)
plt.title('Matriz de Similitud')
plt.colorbar()
plt.show()
```

In [ ]:
```
plt.imshow(cov_matrix*sim_matrix)
plt.title('Matriz de Adyacencia del Grafo de Apariencia')
plt.colorbar()
plt.show()
```


Matriz de Adyacencia del Grafo de Apariencia
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140
1.0
0.8
0.6
0.4
0.2
0.0

Apéndice B

Evaluación de la Co-visibilidad - Jupyter Notebook

Sinopsis

En este apéndice se expone el código relacionado con el método de evaluación de la detección de co-visibilidad. Se incluyen diversas funciones destinadas a evaluar la precisión y la sensibilidad en un procedimiento iterativo para el cálculo de múltiples matrices de co-visibilidad, variando el umbral de decisión (μ).

El código se encuentra en un entorno de trabajo Jupyter Notebook [78], donde se han dispuesto las diferentes funciones en celdas de código independientes. Entre estas celdas de código, se incluye texto explicativo que facilita la comprensión del proceso. Es importante destacar que el lenguaje de programación utilizado es Python.

Co-visibilidad Evaluation

En este código se implementan la evaluación del sistema de detección de co-visibilidad propuesto. Se compara la estimación de co-visibilidad con el valor de ground-truth.

Librerías

In [ ]:
```
import json
import cv2 as cv
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import auc
```

Matriz de co-visibilidad estimada

Estas funciones calculan la matriz de co-visibilidad con el método propuesto.

In [ ]:
```
def occupancy(pts1,pts2,cell_x,cell_y,size):
# Esta función calcula el porcentaje de co-visibilidad entre dos imágenes a partir de la posición de los inliers.
# Se define una regilla de ocupación para cada imágen, las celdas que contengan uno o mas inliers se marcan como ocupadas.
# El porcentaje de ocupación se calcula como el número de celdas ocupadas entre el número total de celdas.
# Se escoge el mayor valor de ocupación entre las imágenes para establecer el porcentaje de co-visibilidad para obtener una matr
#   -Input:
#       pts1, pts2      ---> Listas de inliers, posiciones de características sobre el plano imágen.
#       cell_x, cell_y  ---> Tamaño de la celda de ocupación
#       size            ---> Tamaño de las imágenes originales (x,y)
#   -Return:
#       w               ---> Porcentaje de ocupación, valor de co-visibilidad del arco que une los nodos.

    # Definimos celdas de ocupación para cada imágen:
    occupancy1=np.array([[0 for i in range(cell_y)] for j in range(cell_x)])
    occupancy2=np.array([[0 for i in range(cell_y)] for j in range(cell_x)])

    # Recorremos listas de inliers y marcamos las celdas ocupadas
    iter=len(pts1)
    for i in range(iter):
```



```
        #Occupancy 1:
        x1=pts1[i][0]   # Posición x en plano imágen.
        y1=pts1[i][1]   # Posición y en plano imágen.
        # Ajustamos al tamaño de la celda de ocupación:
        x1_idx=np.int32((x1*cell_x)/size[1])
        y1_idx=np.int32((y1*cell_y)/size[0])
        # Marcamos la celda como ocupada:
        occupancy1[x1_idx][y1_idx]=1

        #Occupancy 2:
        x2=pts2[i][0]   # Posición x en plano imágen.
        y2=pts2[i][1]   # Posición y en plano imágen.
        # Ajustamos al tamaño de la celda de ocupación:
        x2_idx=np.int32((x2*cell_x)/size[1])
        y2_idx=np.int32((y2*cell_y)/size[0])
        # Marcamos la celda como ocupada:
        occupancy2[x2_idx][y2_idx]=1

    # Suma de las celdas ocupadas:
    sum1=0
    sum2=0
    for i in range(cell_y):
        for j in range(cell_x):
            if(occupancy1[j][i]==1):
                sum1+=1
            if(occupancy2[j][i]==1):
                sum2+=1

    # Calculo del porcentaje de ocupación como el número de celdas ocupadas entre le número total de celdas:
    occupancy_size=cell_x*cell_y
    w1=np.float64(sum1/occupancy_size)
    w2=np.float64(sum2/occupancy_size)

    # Seleccionamos el valor máximo de ocupación como porcentaje de co-visibilidad entre las imágenes.
    w=np.max([w1,w2])   # Se hace para conseguir una matriz de adyacencia simétrica para el Grafo de Apariencia.

    return w
```

In [ ]:
```
def overlap_matrix(im_dir,im_path,umb_F):
    # Esta función calcula la matriz de co-visibilidad para un valor de umbral 'umb_F' a partir de un lista de nombres de imágene
    # Se lleva a cabo el proceso de detección de co-visibilidad con todos los pares de imágenes en la lista-
    #   -Input:
    #       im_dir          ---> Lista con nombres de imágenes
```

```
#       im_path         ---> Directorio de las imágenes
#       umb_F           ---> Umbral de número de inliers para determinar la existencia de co-visibilidad
#   -Return:
#       overlap         ---> Matriz de co-visibilidad binaria, guarda True o False según se detecte la co-visibilidad
#       covisibilidad   ---> Matriz de co-visibilidad real con el porcentaje de solapamiento entre imágenes

# Calculo del número de iteraciones para mostrar la evolución del proceso
im_size=len(im_dir)
total_iter=(im_size**2 + im_size)/2 #(np.math.factorial(im_size)/(np.math.factorial(2)*np.math.factorial(im_size-2)))+im_size
cont_iter=0


# Return matrix:
overlap=np.array([[0 for i in range(im_size)] for j in range(im_size)]) # Matriz de co-visibilidad binaria matrix, se guarda
covisibilidad=np.array([[0 for i in range(im_size)] for j in range(im_size)])       # Matriz de co-visibilidad real, se guar

# SIFT descriptor class:
sift=cv.xfeatures2d.SIFT_create()

# Bucle que compara pares de imágenes:
for i in range(im_size):
    # Carga imagen 1:
    suffix1=im_dir[i]
    im1=cv.imread(im_path+'/'+suffix1)

    # Calcula descriptor SIFT para imagen 1, devuelve descriptores de características (d) y keypoints (kp)
    kp1, des1 = sift.detectAndCompute(im1,None)

    for j in range(i,im_size):
            # Carga imagen 2:
            suffix2=im_dir[j]
            im2=cv.imread(im_path+'/'+suffix2)

            # Si son la misma imagen (i==j) asigna co-visibilidad igual a 1
            if i==j:
                overlap[i][j]=True
                covisibilidad[i][j]=1

            else:
                # Calcula descriptor SIFT para imagen 1, devuelve descriptores de características (d) y keypoints (kp)
                kp2, des2 = sift.detectAndCompute(im2,None)

                #Emparejamiento de descriptores por el método Brutal-Forcer-Matcher (BFM)
                if len(kp1) > 0 and len(kp2) > 0:
```



```
                    # BFMatcher
                    bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)      # Clase BFM con correlación cruzada positiva (crossCh
                    matches = bf.match(des1,des2)                       # Emparejamiento (matching) de descriptores de caract
                    matches=sorted(matches, key= lambda x:x.distance)   # Se ordenan los matches de menor a mayor distancia (
                    if len(matches)>100: matches=matches[:100]          # Se opera con los 100 matches con menor distancia, (

                    # Obtenemos keypoints con correspondencia:
                    pts1=[]
                    pts2=[]

                    for mat in matches:
                        pts1.append(kp1[mat.queryIdx].pt)
                        pts2.append(kp2[mat.trainIdx].pt)

                    pts1=np.int32(np.array(pts1))
                    pts2=np.int32(np.array(pts2))


                     # Calculo de la Matriz Fundamental (F) con algoritmo de RANSAC
                    if len(pts1)>=50:
                        F, mask = cv.findFundamentalMat(pts1,pts2,cv.FM_RANSAC,3,0.99,10) # Calculo Matriz Fundamental, nº i

                        # Seleccionamos inliers, pares de keypoints representados por la Matriz Fundamental obtenida
                        pts1 = pts1[mask.ravel()==1]
                        pts2 = pts2[mask.ravel()==1]

                        # Detección de co-visibilidad en base al umbral de número de inliers
                        if len(pts1)>umb_F:
                            OVERLAP_ENABLE=True

                            # Guardamos la co-visibilidad positiva
                            overlap[i, j]=True
                            overlap[j, i]=True

                            # Calculamos el porcentaje de co-visibilidad
                            tam=im1.shape
                            # Tamaño de celda de ocupación
                            cell_x=4
                            cell_y=np.int32((tam[0]/tam[1])*cell_x)

                            # Calculo de porcente de ocupación
                            cov_ratio=occupancy(pts1,pts2,cell_x,cell_y,tam)
                            # Guarda porcentaje de ocupación en matriz de co-visibilidad simétrica
                            covisibilidad[i, j]=cov_ratio
```

```
                                covisibilidad[j, i]=cov_ratio

                                del cov_ratio

                                del pts1,pts2

                # Print porcentaje de ejecución:
                cont_iter+=1
                if j%10==0: print('Proceso: ',np.round((cont_iter/total_iter)*100,decimals=6),'%',end='\r')

    return overlap, covisibilidad
```

Matriz de Co-visibilidad ground-truth

Esta matriz guarda los valores reales de co-visibilidad entre pares de imágenes del dataset.

In [ ]:
```
def gt_matrix(im_path):
    # Esta función calcula la matriz de co-visibilidad de valores verdaderos a partir del nombre de los elementos del dataset or
    # En este dataset, imágenes co-visibles se guardan con un nombre característico del lugar que representan.

    im_dir=sorted(os.listdir(im_path))  # Lista de nombre de las imágenes
    mat_size=len(im_dir)

    # Inicialización de la matriz de ground-truth con el tamaño de la lista de imágenes
    ground_truth=[[0 for i in range(mat_size)] for j in range(mat_size)]

    # Se recorre la lista de imágenes en un bucle anidado para comparar sus nombres:
    for i, suffix1 in enumerate(im_dir):
        for j, suffix2 in enumerate(im_dir):
            # Si el nombre de las imágenes son el mismo se determina co-visibilidad 1
            if suffix1 == suffix2: ground_truth[i][j]=1

            else:
                # Se obtiene la etiqueta de lugar de los nombres de las imágenes
                label1=suffix1.partition("_")[0]
                label2=suffix2.partition("_")[0]
                # Si las etiquetas coinciden se marcan como co-visibles:
                if label1 == label2: ground_truth[i][j]=1

    return ground_truth
```


Evaluation - Precision and Recall

En esta parte se implementa la función que calcula la precisión y sensibilidad de una matriz de co-visibilidad estimada comparandola con la matriz de ground-truth.

In [ ]:
```
def eval_PR(covisibilidad, ground_truth):
# Esta función compara elemento a elemento los valores de la matriz de co-visibilidad estimada y la matriz de ground-truth.
# Según la predicción sea correcta o no se determinan 4 situaciones: True Positive, False Positive y False Negative.
# En función de estos valores se calcula la precisión y sensibilidad
#   -Input:
#       covisibilidad  ---> Matriz de co-visibilidad estimada
#       ground_truth   ---> Matriz de co-visibilidad ground-truth
#   -Return:
#       Precision      ---> Precisión obtenida por la matriz de co-visibilidad estimada
#       Recall         ---> Sensibilidad obtenida por la matriz de co-visibilidad estimada

    # Inicialización de variables
    TP=0    # True Positive: La estimación de co-visibilidad positiva es correcta
    FP=0    # False Positive: La estimación de co-visibilidad positiva es incorrecta
    FN=0    # False Negative: La estimación de co-visibilidad negativa es incorrecta

    # Se recorren las filas y columnas de las matrices de co-visibilidad y ground truth
    for i in range(covisibilidad.shape[0]):
        for j in range(covisibilidad.shape[1]):
            # Se determinan los casos de TP, FP y FN:
            if covisibilidad[i][j] == True and ground_truth[i][j]==True: TP+=1
            if covisibilidad[i][j] == False and ground_truth[i][j]==True: FN+=1
            if covisibilidad[i][j] == True and ground_truth[i][j]==False: FP+=1

    # Se calcula los valores de precisión y sensibilidad
    Precision = TP/(TP+FP)
    Recall = TP/(TP+FN)

    return Precision, Recall
```

Evaluation de Co-visibilidad

En esta parte se realiza un barrido del umbral que determina el número de inliers del proceso de detección de co-visibilidad.

Se estudian los valores de Precisión y Sensibilidad para cada matriz obtenida.

In [ ]:
```
# Definimos rango de valores
trh_F=np.arange(2,40,2) #[10,20,30,40,50,100,200]
trh_F=[int(i) for i in trh_F]

# Directorios:
im_path=".../map/Images/Test_"             # Directorio de con imágenes del dataset controlado
save_im_path=".../Covisibilidad_trhVariable/"  # Directorio para guardar las matrices de co-visibilidad obtenidas
seq_arr=["cloudy","night","sunny"]      # Distintas secuencias del dataset controlado para la evaluación

# Se itera sobre los distintos datasets
for seq in seq_arr:
    # Inicializa listas de precisión y sensibilidad para los distintos valores del umbral
    P_hist=[]
    R_hist=[]

    # Se calcula una matriz de co-visibilidad para cada valor del rango de valores del parámetro
    for i in range(len(trh_F)):
        # Lista con nombres de imágenes
        im_dir=sorted(os.listdir(im_path+seq))

        # Indica proceso de ejecución:
        print('Epoch: ',i,' / ',len(trh_F))

        # Calculo de matrices de co-visibilidad estimada y ground-truth
        overlap,w, match_arr=overlap_matrix(im_dir,im_path+seq,trh_F[i])    # Se calcula la matriz de co-visibilidad para cada u
        ground_truth=gt_matrix(im_path+seq)

        # Guarda la matriz de co-visibilidad obtenida
        fig_path=save_im_path+"Fig_COLD_"+seq+"_ovl_Thr"+str(trh_F[i])+"version2.npy"
        np.save(fig_path,w)

        # Calculo de la precisión y sensibilidad para la matriz de co-visibilidad obtenida
        P, R = eval_PR(overlap,ground_truth)

        # Guarda el resultado de la evaluación para cada umbral
        P_hist.append(P)
        R_hist.append(R)

    # Guarda las listas de precisión y sensibilidad para cada dataset empleado
    save_dict={}
    save_dict.update({"Precision":P_hist})
    save_dict.update({"Recall":R_hist})
```



```
    save_dict.update({"Trheshold_cov":trh_F})

    save_json=".../Utils/Cov_PR_COLD_"+seq+"version2.json"
    with open(save_json, "w") as outfile:
        json.dump(save_dict, outfile)
```

Plot resultados - Curva PR

Se plotean las curvas de precisión y sensibilidad para los distintos valores del umbral sobre los tres datasets empleados

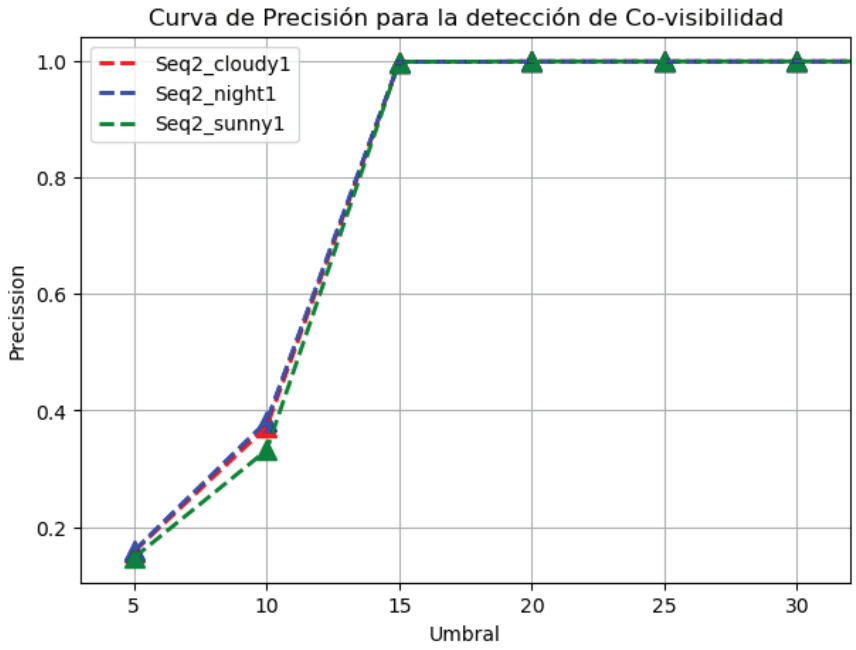
In [ ]:
```
cloudy_eval=json.load(open(".../Utils/Cov_PR_COLD_cloudy.json"))
night_eval=json.load(open(".../Utils/Cov_PR_COLD_night.json"))
sunny_eval=json.load(open(".../Utils/Cov_PR_COLD_sunny.json"))

Trh=cloudy_eval["Trheshold_cov"]
n=len(Trh)
cloudy_P=cloudy_eval["Precision"]
cloudy_R=cloudy_eval["Recall"]
night_P=night_eval["Precision"]
night_R=night_eval["Recall"]
sunny_P=sunny_eval["Precision"]
sunny_R=sunny_eval["Recall"]
```

Curva de Precisión

In [ ]:
```
# Crear figure:
plt.figure(figsize=[7,5])
plt_curve=plt.axes()

# Recorremos las listas de precisión para cada valor del umbral:
for k in range(len(Trh)):
    # Plot puntos en curva de precisión:
    plt_curve.plot(Trh[k],cloudy_P[k],marker='^',markersize=10,color='r') #,label=my_label[k])
    plt_curve.plot(Trh[k],night_P[k],marker='^',markersize=10,color='b') #,label=my_label[k])
    plt_curve.plot(Trh[k],sunny_P[k],marker='^',markersize=10,color='g') #,label=my_label[k])

# Plot curvas de precisión:
plt_curve.plot(Trh,cloudy_P,color='r', linewidth=2,linestyle='--',label='Seq2_cloudy1')
plt_curve.plot(Trh,night_P,color='b', linewidth=2,linestyle='--',label='Seq2_night1')
```

```
plt_curve.plot(Trh,sunny_P,color='g', linewidth=2,linestyle='--',label='Seq2_sunny1')

# Show plot:
plt.legend()
plt.xlim(3,32)
plt.grid()
plt.xlabel('Umbral')
plt.ylabel('Precission')
plt.title("Curva de Precisión para la detección de Co-visibilidad")
```

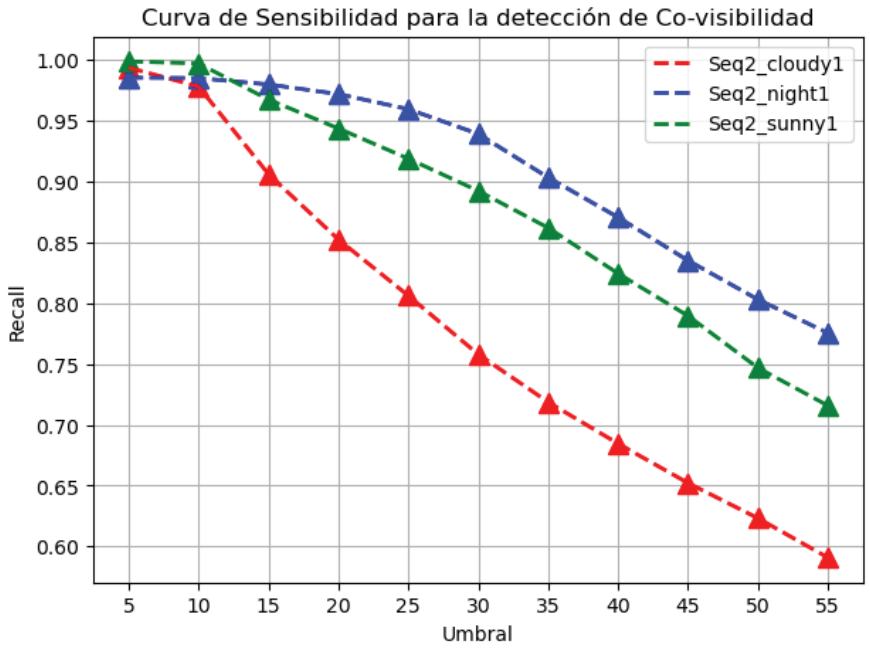
Out[ ]: Text(0.5, 1.0, 'Curva de Precisión para la detección de Co-visibilidad')



Curva de Sensibilidad

In [ ]:
```
# Crear figura
plt.figure(figsize=[7,5])
plt_curve=plt.axes()

# Recorremos las listas de precisión para cada valor del umbral:
for k in range(len(Trh)):
    # Plot puntos en curva de sensibilidad:
    plt_curve.plot(Trh[k],cloudy_R[k],marker='^',markersize=10,color='r') #,label=my_label[k])
    plt_curve.plot(Trh[k],night_R[k],marker='^',markersize=10,color='b') #,label=my_label[k])
    plt_curve.plot(Trh[k],sunny_R[k],marker='^',markersize=10,color='g') #,label=my_label[k])

# Plot curvas de sensibilidad
plt_curve.plot(Trh,cloudy_R,color='r',linewidth=2,linestyle='--',label='Seq2_cloudy1')
plt_curve.plot(Trh,night_R,color='b',linewidth=2,linestyle='--',label='Seq2_night1')
plt_curve.plot(Trh,sunny_R,color='g',linewidth=2,linestyle='--',label='Seq2_sunny1')

# Show plot
plt.legend()
plt.title("Curva de Sensibilidad para la detección de Co-visibilidad")
plt.xticks(Trh, Trh)
plt.grid()
plt.xlabel('Umbral')
plt.ylabel('Recall')
```

Out[ ]: Text(0, 0.5, 'Recall')



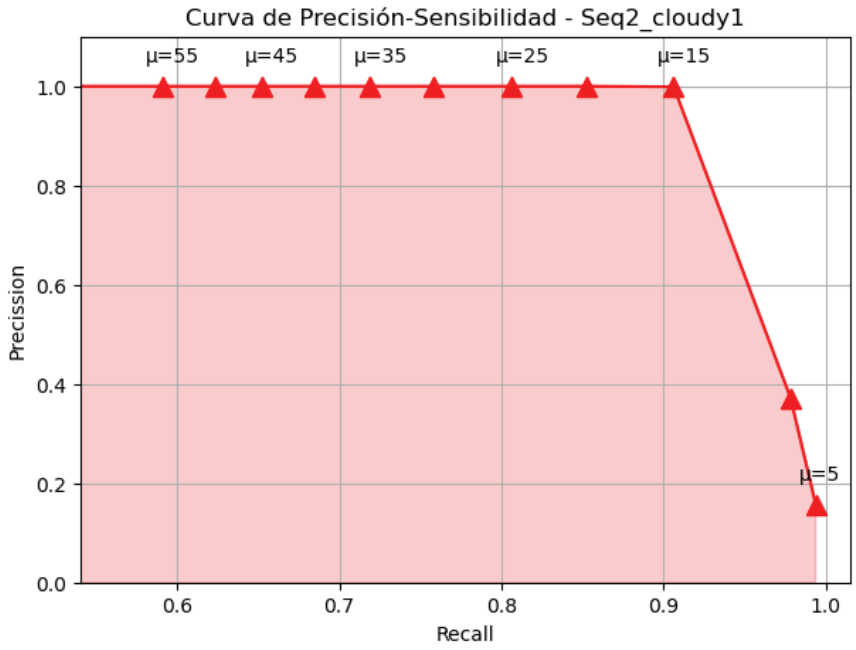
Curva de Precisión-Sensibilidad

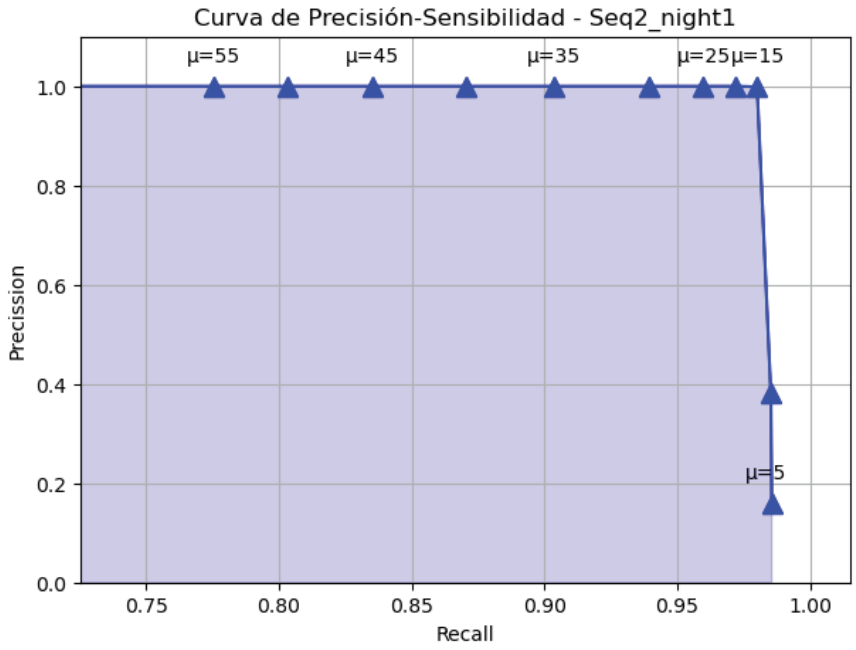
In [ ]:
```
# En esta parte se plotean las curvas de precisión-sensibilidad para los 3 dataset por separado

# Listas de Precisión y Sensibilidad para los tres dataset
Precision=[cloudy_P,night_P,sunny_P]
Recall=[cloudy_R,night_R,sunny_R]

my_color=['r','b','g']  # Definimos colores del plot:
my_label=['Seq2_cloudy1','Seq2_night1','Seq2_sunny1']   # Definimos etiquetas para el plot

for i in range(3):
    # Creamos la figura, una por cada dataset
```

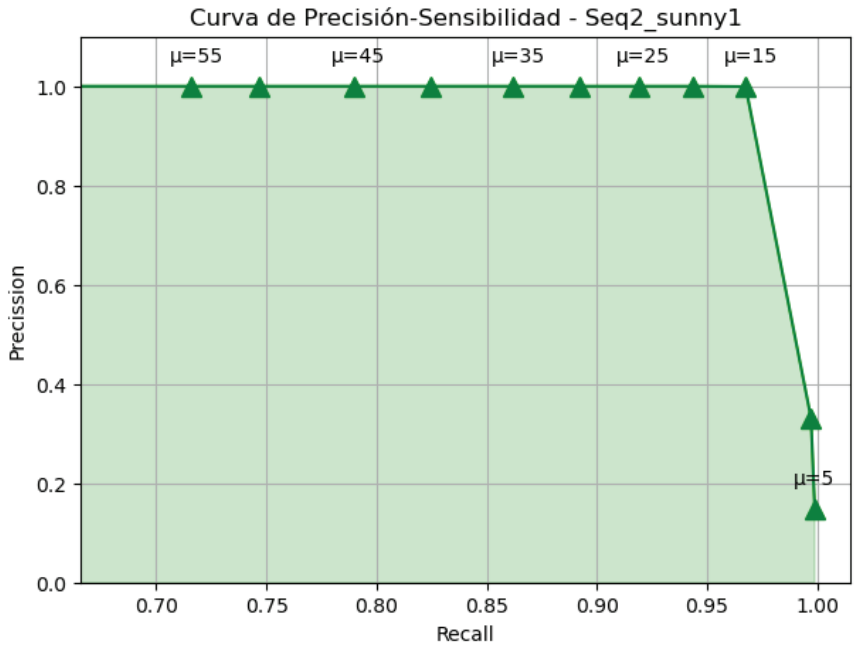


```
    plt.figure(figsize=[7,5])
    plt_curve=plt.axes()

    # Copiamos listas de Precisión y Sensibilidad
    P=np.copy(Precision[i]).tolist()
    R=np.copy(Recall[i]).tolist()

    # Indexamos el valor (P=1, R=0) para calcular correctamente el área bajo la curva
    P.extend([float(1)])
    R.extend([float(0)])

    # Plot puntos de precisión y sensibilidad
    for k in range(len(Trh)):
        # Plot puntos
        plt_curve.plot(R[k],P[k],marker='^',markersize=10,color=my_color[i])
        # Plot text con el valor del umbral en índices pares
        if (k%2)== 0: plt_curve.text(R[k]-0.01,P[k]+0.05,"μ="+str(Trh[k]))

    # Plot curva de precisión-sensibilidad
    plt_curve.plot(R,P,color=my_color[i],label=my_label[i])
    plt.fill_between(R,P,color=my_color[i],alpha=0.2)  # Área bajo la curva sobreada

    # Calculamos el área bajo la curva
    auc_score = auc(R, P)

    # Show plot:
    plt.grid()
    plt.ylim(0,1.1)
    plt.xlim(R[-2]-0.05,1.015)
    plt.xlabel('Recall')
    plt.ylabel('Precission')
    plt.title("Curva de Precisión-Sensibilidad - "+my_label[i])

    # Print área bajo la curva:
    print('AUC para '+my_label[i]+': ',auc_score)
```

```
AUC para Seq2_cloudy1:  0.9596975523326542
AUC para Seq2_night1:  0.9836591501628735
AUC para Seq2_sunny1:  0.9877299145450056
```


Curva de Precisión-Sensibilidad - Seq2_cloudy1
μ=55
μ=45
μ=35
μ=25
μ=15
μ=5
Precission
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.6
0.7
0.8
0.9
1.0


Curva de Precisión-Sensibilidad - Seq2_night1
μ=55
μ=45
μ=35
μ=25μ=15
μ=5
Precission
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.75
0.80
0.85
0.90
0.95
1.00


Curva de Precisión-Sensibilidad - Seq2_sunny1
μ=55
μ=45
μ=35
μ=25
μ=15
μ=5
Precission
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.70
0.75
0.80
0.85
0.90
0.95
1.00

Apéndice C

Bipartición Recursiva del Grafo de Apariencia - Jupyter Notebook

Sinopsis

En este apéndice se ilustra el código relativo a la Bipartición Recursiva del Grafo de Apariencia. Se implementa una clase que construye el Grafo de Apariencia con la infromación de los ficheros relativos a las matrices de similitud, co-visibilidad y poses (ver A). Esta clase engloba diversas funciones relacionadas con la bipartición espectral de grafos y el cálculo del valor de Ncut, entre otras operaciones relevantes.

El código se encuentra en un entorno de trabajo Jupyter Notebook [78], donde se han dispuesto las diferentes funciones en celdas de código independientes. Entre estas celdas de código, se incluye texto explicativo que facilita la comprensión del proceso. Es importante destacar que el lenguaje de programación utilizado es Python.

Normalized Cut Graph

En este código se implementa la bipartición recursiva del Grafo de Apariencia a fin de obtener una topología de lugares sobre un Mapa de Apariencia.

Se implimenta una clase denominada 'Appearance_graph' que construye el Grafo de Apariencia, dentro de esta clase se implementan los algoritmos necesarios para computar la bipartición recursiva.

Librerías

In [ ]:
```
import json
import matplotlib.pyplot as plt
import os
import shutil
import numpy as np
from PIL import Image
import random
import seaborn as sb
from adjustText import adjust_text

from Utils import geometry2 as geo
from Utils.load_utils import load_sequence
from Utils import load_utils
```

Función Auxiliar: Deep-first Search

In [ ]:
```
def dfs(visited, W, index=0):
# Esta función recorre el Grafo de Apariencia buscando los nodos conectados al nodo 'index'.
# Es una función recursiva que se llama a si misma por cada nodo nuevo no visitado.
#   -Input:
#       visited     ---> Lista de nodos visitados, originalmente vacía
#       W           ---> Matriz de adyacencia binaria del Grafo de Apariencia
#       index       ---> Índice del nodo a partir del cual se inicia la busqueda
#   -Return:
#       visited     ---> Lista de nodos visitados modificada (por referencia)
```



```
    if index not in visited:    # Comprueba si el nodo 'index' ha sido visitado previamente
        visited.append(index)   # Actualiza lista de nodos visitados con nuevo nodo
        adj_idx=np.argwhere(W[index,:]==1)  # Busca los nodos conectados a 'index' en la fila correspondiente de la matriz de adj
                                            # Valores iguales a 1 sobre la matriz indica una conexión al nodo 'index'.
        adj_idx=np.reshape(adj_idx,[-1])

        # Recorre lista de nodos conectados
        for neighbour in adj_idx:
            dfs(visited,W,neighbour)    # Llamada recursica a DFS por cada nodo conectado a 'index'
```

Clase: Grafo de Apariencia

En esta clase se contruye el grafo de apariencia a partir de las matrices de co-visibilidad, similaridad y el dataset json con las poses y los descriptores de apariencia.

Se implementa el Corte Normalizado de Grafo, la bipartición recursiva y otras funciones auxiliares (plot map, folders, ...)

In [ ]:
```
class weighted_graph:
    def __init__(self, cov_matrix, sim_matrix, data_set, db_name): # Inicialización de clase

        # Se guarda el dataset del Grafo de Apariencia:
        self.data_set=data_set
        self.db=db_name         # Nombre del dataset

        # Calcula la matriz de adyacencia del Grafo de Apariencia, multiplica elemento a elemento
        # las matrices de co_visibilidad y similitud.
        self.weights=cov_matrix*sim_matrix

        # Crea lista de subgrafos, originalmente 1 lista con todos los nodos del Grafo de Apariencia
        self.subgraphs=[[x for x in range(len(self.weights))]] #Clusters list

        # Se comprueba la conectividad del grafo
        self.check_connectivity()


    def check_connectivity(self):
    # Esta función comprueba la conectividad del grafo original ya que debe ser conexo en todos sus nodos.
    # Se emplea el algoritmo Deep-first Search (DFS). Los subgrafos encontrados se guardan en la lista de subgrafos.

        # Calcula matriz de adyacencia binaria para el algoritmo de DFS
```

```
        adjacent_graph=np.abs(np.sign(self.weights))

        #Lista de nodos del Grafo de Apariencia:
        tam=adjacent_graph.shape[0]
        nodes=np.arange(tam,dtype=int)

        # Inicialización de variables:
        visited=[]      # Lista de nodos visitados
        saved=[]          # Lista de nodos visitados ya guardados
        idx=0           # Inidice del nodo para el algoritmo DFS

        # Ejecución algoritmo DPS
        dfs(visited, adjacent_graph, idx)   # Actualiza lsita de nodos visitados

        # Si el número de nodos visitados es menor al de nodos en el Grafo de Apariencia exiten subgrafos no conexos
        if (len(visited)<tam):

            print('Unconnected graphs founds - Splitting')
            self.subgraphs.pop(0)       # Se elimina lista original de nodos

            while len(visited)<tam:     # Bucle while hasta visitar todos los nodos
                # Nuevo subgrafo son los índices visitados que no han sido guardado
                new_subgraph=np.setdiff1d(visited,saved)
                new_subgraph=np.array(new_subgraph).tolist()

                # Guardamos nuevo subgrafo en lista de subgrafos
                self.subgraphs.append(new_subgraph)

                # Calculamos nodos que faltan por visitar
                miss= np.setdiff1d(nodes,visited)

                if len(miss)>0:  # Si aun faltan nodos por visitar
                    idx=miss[0]     # Nuevo índice para buscar subgrafos con DFS
                    saved=np.copy(visited)    # Actualiza lista de nodos guardados

                    # Ejecución del algoritmo DFS con el nuevo nodo
                    dfs(visited, adjacent_graph, idx)

            # Guarda último subgrafo:
            new_subgraph=np.setdiff1d(visited,saved)
            new_subgraph=np.array(new_subgraph).tolist()
            self.subgraphs.append(new_subgraph)

        print('Nº of inconected inside graphs: ', len(self.subgraphs))
```



```
    def cut(self,subgraphA,subgraphB):
        # Función para calcular el valor del corte entre los subgrafos A y B.
        # Se calcula el corte como la suma del peso de los arcos que unen los subgrafos en la matriz de adyacencia
        return self.weights[subgraphA][:,subgraphB].sum()

    def assoc(self,graph,subgraph):
        # Función para calcular el valor de la asociación entre un subgrafo y el grafo completo
        # Se calcula como la suma del peso de los arcos que unen los nodos del subgrafos con el resto del grafo en la matriz de
        return self.weights[subgraph][:,graph].sum()

    def ncut(self,graph,subgraphA,subgraphB):
        # Función para calcular el valor del corte normalizado entre los subgrafos A y B.
        # Se aplica la formula del corte normalizado, llamando a las funciones 'cut' y 'assoc'
        return (self.cut(subgraphA,subgraphB)/self.assoc(graph,subgraphA))+(self.cut(subgraphA,subgraphB)/self.assoc(graph,subgra

    def spectralbisection(self,graph_indices):
        # Función que implementa la bipartición espectral del grafo.
        # Se opera sobre la matriz de adyacencia, escogiendo subgrupos relativos a los nodos del grafo con el que se opera.
        # self.weights[graph_indices][:,graph_indices] --> Selección de submatriz para los nodos seleccionados.

        # Matriz D, en su diagonal toma el valor de la suma de las filas en la matriz de adyacencia.
        D=np.diag(self.weights[graph_indices][:,graph_indices].sum(0))

        # Matriz Laplaciana, se calcula como la resta entre D y la matriz de adyacencia del grafo
        Laplacian=np.array(D-self.weights[graph_indices][:,graph_indices],dtype=np.float64)

        # Calculo de autovalores y acutovectores de la matriz Laplaciana
        eigenvalues, eigenvectors=np.linalg.eigh(Laplacian)  # This solves standard and generalized eigenvalue problems
        partition=eigenvectors[:, np.argsort(eigenvalues)[1]] # Se selecciona el eutovector asociado al segundo menor autovalor

        # Se generan dos grupos clasificando los nodos segun el valor positivo o negativo del elemento en el vector de partición
        return [graph_indices[i] for (i, x) in enumerate(partition) if x < 0], \
                    [graph_indices[i] for (i, x) in enumerate(partition) if x >= 0]

    def sort_clusters(self):
        # Función para ordenar los subgrafos obtenidos
        # Se recorre la lista de subgrafo y se intercambia su lugar para que los primeros índices sigan una ordenación natural

        n_clusters=len(self.subgraphs)
        for i in range(n_clusters):
            for j in range(0,n_clusters-i-1):
                if self.subgraphs[j][0]>self.subgraphs[j+1][0]: # Si el primer nodo guardado en j es mayor que el primer nodo gu
                    self.subgraphs[j],self.subgraphs[j+1]=self.subgraphs[j+1],self.subgraphs[j] # Intercambio de posición
```

```
        return self.subgraphs


    def clusters_recursive(self, graph, thr):
    # Función auxiliar recursiva para la bipartición generalizada del grafo
    #   -Input:
    #       graph   ---> Lista de nodos del grafo con el que se opera
    #       thr     ---> Umbral de decisión para evaluar el valor Ncut de cada bipartición
    #   -Return:
    #       graph   ---> Lista de nodos del grafo modificada tras realizar la bipartición

        cut_value=None  # Valor de Ncut por defecto

        if graph is not None and len(graph) > 1 :   # Comprueba tamaño de lista de nodos
            # Bipartición del grafo:
            group_A,group_B=self.spectralbisection(graph)

            if (group_A != [] and group_B !=[]):    # Comprueba subgrafos no son listas vacías para evitar errores

                # Calculo de valor de Ncut entre subgrafos
                cut_value=self.ncut(graph,group_A,group_B)

                # Llamada recursiva para los subgrafos A y B obtenidos
                # Se devuelve los grafos modificados tras la bipartición
                cluster_A = self.clusters_recursive(group_A,thr)
                cluster_B = self.clusters_recursive(group_B,thr)

                # Calcula de nuevo la lista de nodos como la unión de las subgrafos A y B.
                # Esto se hace para asegurar no guardar nodos repetidos que hayan sido previamente guardados.
                graph=np.union1d(cluster_A,cluster_B).tolist()
                graph=[int(x) for x in graph]

                # Se evalúa valor del Ncut para la bipartición de A y B:
                if cut_value < thr:
                    if len(cluster_A)>5:    # Se guarda el subgrafo si contiene mas de 5 nodos, para evitar grupos muy pequeños
                        self.subgraphs.append(cluster_A)                # Indexa el nuevo subgrafo
                        graph=np.setdiff1d(graph,cluster_A).tolist()    # Elimina nodos guardados del grafo original

                    if len(cluster_B)>5:    # Se guarda el subgrafo si contiene mas de 5 nodos, para evitar grupos muy pequeños
                        self.subgraphs.append(cluster_B)                # Indexa el nuevo subgrafo
                        graph=np.setdiff1d(graph,cluster_B).tolist()    # Elimina nodos guardados del grafo original

        # Se devuelve la lista de nodos modificada, donde se han eliminado los nodos previamente guardados
        return graph
```



```


    def clusters(self, sup_thr=0.4):
        # Esta función implementa la bipartición recursiva del grafo llamando a la función 'clusters_recursive' para
        # cada subgrafo no conexo encontrado en el procedimiento de 'check_connectivity'.

        # Copia lista de subgrafos para operar con ella
        subgraph_copy=self.subgraphs.copy()
        self.subgraphs=[]   # Elimina lista de subgrafos original

       # Recorre los subgrafos no conexos encontrados por la función 'check_connectivity'
        for i in range(len(subgraph_copy)):
            # Función recursiva genera subgrafos:
            graph = self.clusters_recursive(subgraph_copy[i], sup_thr) # Devuelve el último grafo que no ha sido cortado por la f

            #Si el número de elementos del último grafo no cortado es mayor a 5 se guarda como nuevo subgrafo
            if len(graph) > 5: self.subgraphs.append(graph)

        self.sort_clusters()    # Función ordena el conjunto de subgrafos
        #self.make_gifs()       # Función genera gifs con las imágenes de cada subgrafo
        #self.folders()         # Función guarda en carpetas las imágenes de cada subgrafo
        return self.subgraphs


    def classification(self):
        # Función que devuelve un vector de clasificación con el índice de el subgrafo al que pertenece cada nodo

        # Inicializa el vector de clasificación:
        classif=[0 for i in range(len(self.data_set['poses']))]

        for i,cluster in enumerate(self.subgraphs): # Recorre conjunto de subgrafos
            for index in cluster:
                classif[index]=i    # Asigna a la posición del nodo 'index' el valor del subgrafo al que pertenece
        return classif

    def create_map(self, map_path):
        # Función para guardar el Grafo de Apariencia (mapa topológico) con la información de los subgrafos

        # Nuevo diccionario
        map={}

        map['im_paths']=self.data_set['im_paths']           # Nombre de las imágenes
        map['poses']=self.data_set['poses']                 # Poses
```

```
        map['odom_poses']=self.data_set['odom_poses']       # Poses odométricas
        map['features']=self.data_set['features']           # Descriptores de la imágen
        map['clusters_list']=self.subgraphs                 # Lista de subgrafos como lista de índices de los elementos
        map['clusters_idx']=self.classification()           # Vector de clasificación de los nodos

        # Guarda fichero .json
        with open(map_path, "w") as outfile:
            json.dump(map, outfile)

        return map

    def plot_map(self,save=False):
        # Función para mostrar el mapa topológico obtenido tras la bipartición recursiva.
        # Se plotea cada elemento del mapa en su posición (x,y) correspondiente, asignando un código de color a cada grupo para
        # Además se muestra la matriz de adyacencia resultante de la bipartición recusiva usando el mismo código de colores

        # Generamos paleta de color:
        n=15    # Número de colores
        my_colors=sb.color_palette('turbo',n)
        random.shuffle(my_colors)

        # Procesamiento de las poses del Grafo de Apariencia en SE(2):
        train_dataset = load_utils.load_sequence(self.data_set,verbose=False)
        poses = train_dataset['poses']  # Guardamos poses del mapa

        # Creamos figura:
        plt.figure(figsize=(23,14))
        plt.plot(0,0,marker='*',color='red')    # Se indica el origen (0,0) del mapa

        texts=[]    # Array de texto sobre la imágen

        n_clusters=len(self.subgraphs)
        for i in range(n_clusters):     # Recorre el conjunto de subgrafos

            # Copia de subgrafo para evitar modificar valores
            subgraph=np.copy(self.subgraphs[i])
            idx=np.round(len(subgraph)/2)   # Índice medio del subgrafo

            # Posición de elemento medio en subgrafo
            x_midle=[]
            y_midle=[]

            for k,item in enumerate(self.subgraphs[i]):     # Recorre nodos del subgrafo
                # Obtiene posición del nodo:
```



```
                translation=geo.SE2Poses.t(poses[item])
                new_x=translation[0]
                new_y=translation[1]

                # Plot de pose del nodo, color determinado por el índice del subgrafo
                plt.plot(new_x,new_y,markersize=3,marker='o',color=my_colors[i%n])

                # Guarda la posición del elemento central del subgrafo para mostrar el texto
                if idx == k:
                    x_midle.append(new_x)
                    y_midle.append(new_y)

            # Plot de última pose del subgrafo en negro, delimitación visual de los distintos grupos
            plt.plot(new_x,new_y,markersize=5,marker='o',color='black')

            # Array de texto. Se plotea el índice del grupo en la posición central del grupo
            texts.append(plt.text(x_midle[-1],y_midle[-1], str(i), color='black', bbox=dict(facecolor='w',alpha=0.75, edgecolor=

        # Mostrar el plot:
        plt.axis('off')
        # Función evita superposición en los textos:
        adjust_text(texts, x=x_midle, y=y_midle, autoalign='x', only_move={'points':'xy', 'text':'xy','object':'xy'}, force_poin
        plt.show()


        # En esta parte del código se calcula la matriz de adyacencia resultante de la bipartición con
        # el mismo código de colores que se emplea en el mapa

        # Inicializamos matriz de colores:
        tam=len(self.weights)
        color_mat=np.zeros(tam**2).reshape([tam,-1])

        for i,cl in enumerate(self.subgraphs):  # Se recorre el conjunto de subgrafos
            for itemi in cl:
                for itemj in cl:
                    # Se asigna el mismo color que en el mapa para la matriz de colores simétrica
                    color_mat[itemi,itemj]=i%15
                    color_mat[itemj,itemi]=i%15

        # Se muestra la matriz de colores resultante
        sb.heatmap(color_mat,cmap=my_colors)
        plt.title('Matriz de Adyacencia')
        plt.xlabel('Nodos en GA')
```

```
        plt.ylabel('Nodos en GA')


    def make_gifs(self):
        # Función para crear gifs animados con el conjunto de fotos de cada subgrafo.

        clusters_path=".../Clusters_gifs" # Dirección para guardar gifs
        # Crear carpeta
        if os.path.exists(clusters_path):
            shutil.rmtree(clusters_path)   #Deletes the content of the folder
        os.makedirs(clusters_path)

        # Crear los gifs:
        n_clusters=len(self.subgraphs)
        pic_path='.../Images/'  # Directorio de imágenes del dataset

        for i in range(n_clusters): # Recorre conjunto de subgrafos
            images=[]
            for j in self.subgraphs[i]:
                images.append(Image.open(pic_path+self.data_set['im_paths'][j]))    # Lista con imágenes del subgrafo

            # Guardar lista de imágenes como un gif
            images[0].save(clusters_path+'/Cluster '+str(i+1)+'.gif',
               save_all=True,
               append_images=images[1:],
               duration=100,
               loop=1)

    def folders(self):
        #Función para crear carpetas con imágenes de subgrafos

        clusters_path=".../Clusters_folder" # Dirección para guardar carpetas de imágenes

        # Crear carpeta:
        if not os.path.exists(clusters_path):
            os.makedirs(clusters_path)
        shutil.rmtree(clusters_path)   #Deletes the content of the folder

        # Crear carpetas con imágenes de subgrafos
        pic_path='.../Images/'  # Directorio de imágenes del dataset
        n_clusters=len(self.subgraphs)
        for i in range(n_clusters): # Recorre conjunto de subgrafos
            try:
                # Crea carpeta para subgrafo
```



```
                new_path=clusters_path+"/Cluster_"+str(i+1)
                if not os.path.exists(new_path):
                    os.makedirs(new_path)

                # Guarda imágenes del subgrafo
                for item in self.subgraphs[i]:
                    original=pic_path+self.data_set['im_paths'][item]
                    target=new_path+"/"+self.data_set['im_paths'][item]
                    shutil.copyfile(original, target)

            except OSError:
                print ('Error: Creating folder. ' +  new_path)
```

Testing

Input data

Cargar las matrices de co-visibilidad, similitud y fichero de poses y descriptores.

In [ ]:
```
map_path='.../Utils/Grafo_apariencia.json'
similtud_path='.../Utils/sim_matrix.npy'
covisibilidad_path='.../Utils/cov_matrix.npy'

# Cargar matrices de co-visibilidad y similitud
sim_matrix = np.load(similtud_path)
cov_matrix = np.load(covisibilidad_path)


# Cargar mapa de poses
map_dataset = json.load(open(map_path))
```

Desordenar datos - Ejecutar la siguiente celda solo en caso de querer desordenar los datos en su origen

In [ ]:
```
# Generar orden de índices aleatorio:
tam=cov_matrix.shape[0]
inds = np.arange(tam)
np.random.shuffle(inds)
```

```
# Desordenar matrices y dataset de acuerdo a estos índices
mess_cov_matrix=cov_matrix[inds][:,inds]
mess_sim_matrix=sim_matrix[inds][:,inds]
mess_dataset=dict(im_paths=[], poses=[], odom_poses=[], features=[], seq=[], test=[])
for key in map_dataset.keys():
    arr=np.array(map_dataset[key])
    arr=arr[inds]
    mess_dataset[key]=arr.tolist()
```

Crear Grafo de Apariencia

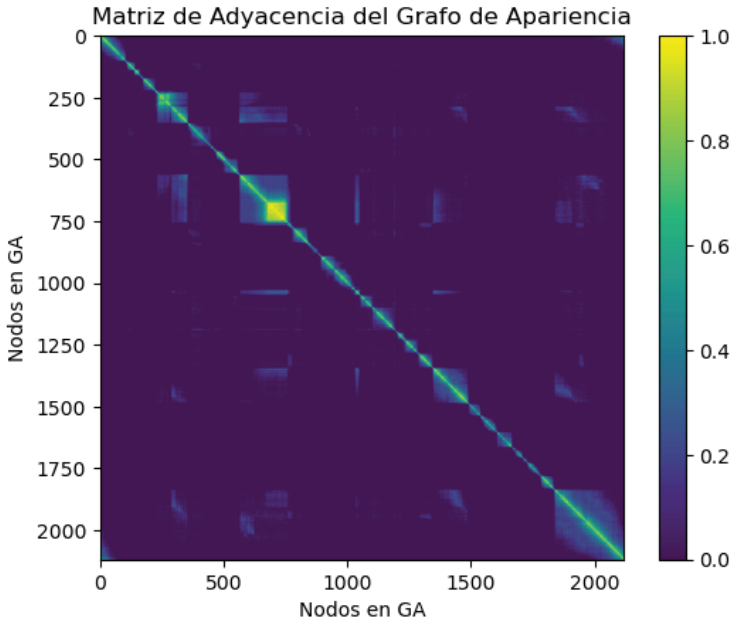
In [ ]:
```
graph=weighted_graph(cov_matrix,sim_matrix,map_dataset,"COLD")
```

```
Unconnected graphs founds - Splitting
Nº of inconected inside graphs:  5
```

Show matriz de adyacencia

In [ ]:
```
plt.imshow(graph.weights)
plt.colorbar()
plt.title('Matriz de Adyacencia del Grafo de Apariencia')
plt.xlabel('Nodos en GA')
plt.ylabel('Nodos en GA')
```

Out[ ]:
```
Text(0, 0.5, 'Nodos en GA')
```

Bipartición recuersiva del Grafo de Apariencia

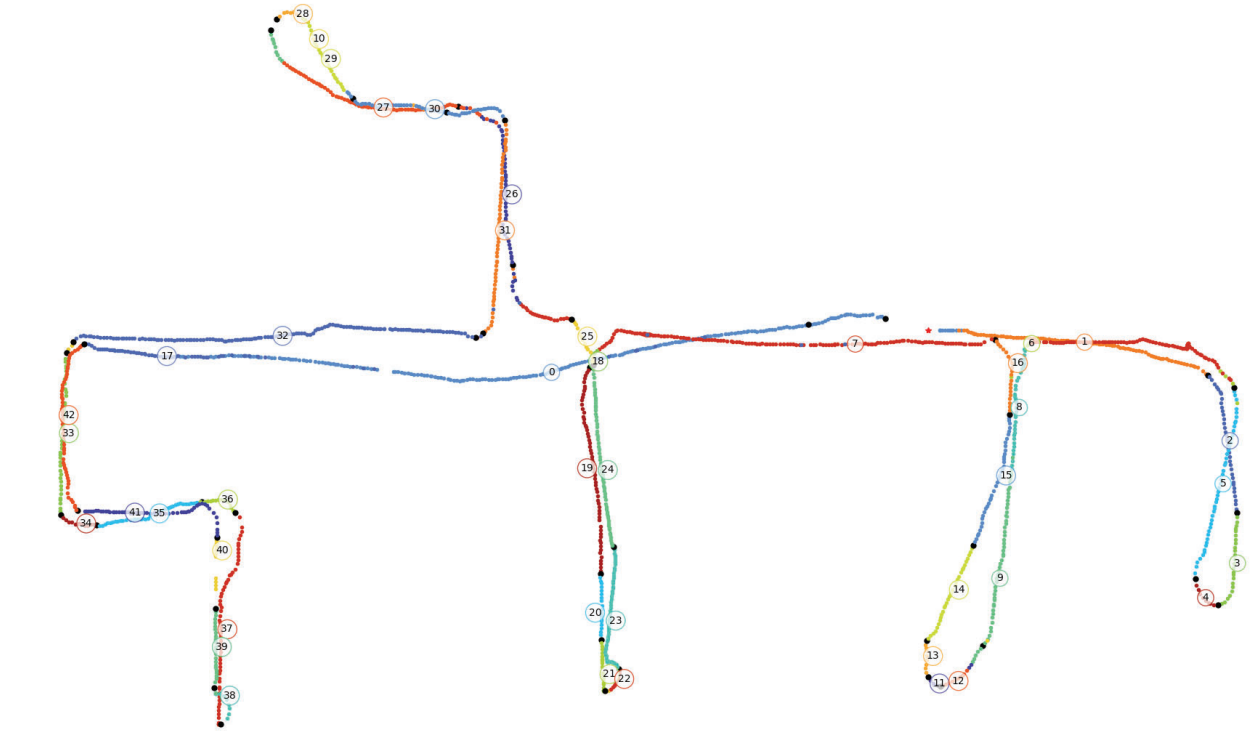
In [ ]:
```
clusters =graph.clusters(sup_thr=0.3)
```

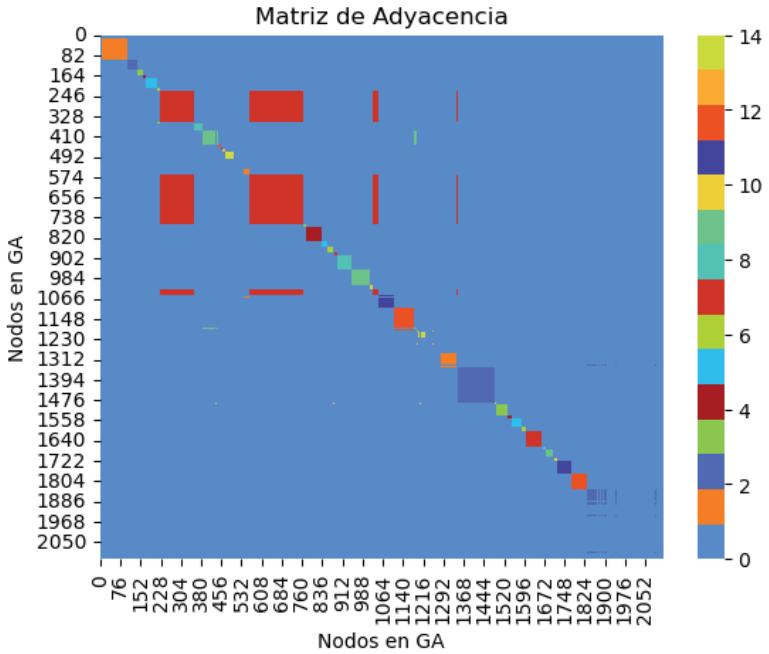
Guardar Grafo de Apariencia resultante tras la bipartición

In [ ]:
```
map_path='.../Utils/Topological_map.json'
map = graph.create_map(map_path)
```

Plot mapa topológico y matriz de adyacencia por colores

In [ ]:
```
graph.plot_map(save=False)
```





Apéndice D

Evaluación de la Topología de Lugares - Jupyter Notebook

Sinopsis

En este apéndice se incluye el código correspondiente a la implementación de los métodos de evaluación de la topología de lugares propuesta en este trabajo. Como parte de ello, se implementa un sistema de Visual Place Recognition (VPR) y se incorporan diversas funciones destinadas a medir diferentes aspectos de esta topología. Además, se presentan los resultados obtenidos tras la evaluación.

El código se encuentra en un entorno de trabajo Jupyter Notebook [78], donde se han dispuesto las diferentes funciones en celdas de código independientes. Entre estas celdas de código, se incluye texto explicativo que facilita la comprensión del proceso. Es importante destacar que el lenguaje de programación utilizado es Python.

Visual Place Recognition + Evaluación de Corte de Grafo

En este código se implementa el sistema de Visual Place Recognition (VPR) y las criterios para evaluar la topología de lugares resultante de la bipartición recursiva del grafo.

Librerías

In [ ]:
```
import h5py
import json
import numpy as np
import os
import matplotlib.pyplot as plt
from Utils import geometry2 as geo
from Utils import load_utils
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sns
```

Funciones auxiliares

In [ ]:
```
def get_descriptor(general_path):
    # Función para leer los descriptores globales de un fichero h5py
    descriptors=[]
    for suffix in os.listdir(general_path):
        f=h5py.File(general_path+suffix,'r')
        descriptors+=f['features']
    return np.array(descriptors)
```

In [ ]:
```
def pose_mean_cov(map_dataset):
    # Función para calcular la media y la matriz de covarianza de las distribuciones de poses de los grupos.
    # Tambien calcula la media de los descriptores de apariencia para cada grupo, denominadas centroides.

    # Inicialización de variables
    mean_pose=[]
    mean_feats=[]
    cov_pose=[]
```



```
    # Se recorren los grupos de la topología
    for cl in map_dataset['clusters_list']:
        poses=map_dataset['poses'][cl]  # Media en pose del grupo
        feats=map_dataset['feats'][cl]  # Media en apariencia del grupo, centroide

        # Calcula la pose y la matriz de covarianza en SE(2)
        poses_mean, poses_cov = geo.get_mean_covariance(poses)
        feats_mean=np.mean(feats,axis=0)

        # Guarda los resultados en las listas:
        mean_pose.append(poses_mean)
        mean_feats.append(feats_mean)
        cov_pose.append(poses_cov)

        del poses_mean, poses_cov, feats_mean

    # Guarda el resultado en el diccionario del mapa topológico:
    map_dataset['poses_mean']=mean_pose
    map_dataset['feats_mean']=mean_feats
    map_dataset['poses_cov']=cov_pose

    return mean_pose, cov_pose
```

APPEARANCE DISTANCE

En esta parte se implementa los criterios para determinar la distancia entre descriptores de la imágen, se emplea en el proceso de VPR para asignar una consulta a un grupo.

Funciones para calcula la distancia en apariencia entre consultas y lugares del mapa topológico

In [ ]:
```
cos_sim=lambda x,y:np.dot(x,y)/(np.linalg.norm(x)*np.linalg.norm(y))    # Similitud del coseno
euc_sim= lambda x,y: np.linalg.norm(np.array(x)-np.array(y))            # Distancia euclídea
```

Criterio de la similitud del coseno

In [ ]:
```
def app_dist_cos(query_descriptors,map_descriptors):
    distances=[[cos_sim(query_descriptors[j],map_descriptors[i])for i in range(len(map_descriptors))]for j in range(len(query_de
    return np.array(distances)
```

Criterio de la distancia euclídea mínima

In [ ]:
```
def app_dist_euc(query_descriptors,map_descriptors):
    distances=[[euc_sim(query_descriptors[j],map_descriptors[i])for i in range(len(map_descriptors))]for j in range(len(query_des
    return np.array(distances)
```

Criterion 1: Most similar centroid

Este criterio mide la distancia entre cada consulta y los centroides de los grupos. Devuelve el grupo al que pertenece la consulta en base a la distancia mínima/máxima a los centroides.

In [ ]:
```
def most_similar_centroid(query_descriptors, map_dataset, criterion='COS'):

    if criterion == 'COS':      # Similitud del coseno
        distances=app_dist_cos(query_descriptors,map_dataset['feats_mean']) # Calcula similitud del coseno para todas las consul
        return np.argmax(np.array(distances),axis=1)    # El valor máximo de la similitud del coseno para cada consulta nos da e

    elif criterion == 'EUC':    # Distancia Euclídea
        distances=app_dist_euc(query_descriptors,map_dataset['feats_mean']) # Calculo de la distancia euclidea para todas las co
        return np.argmin(np.array(distances),axis=1)    # El valor mínimo de la distnacia euclíde para cada consulta nos da el g

    else:
        print('CRITERION ERROR: not valid')
```

Criterion 2: k-most similar images

Este criterio mide la distancia en apariencia para todas los elementos del mapa topológico, devuelve las k mas cercanas

Este criterio se uso en la investigación pero finalmente fue descartado de los resultados finales.

In [ ]:
```
def k_similar(query_descriptors, map_dataset, k, criterion='COS'):

    if criterion=='COS':     # Similitud del coseno
        distances=app_dist_cos(query_descriptors,map_dataset['feats']) # Calcula similitud del coseno para todas las consultas c
        idx=np.argsort(np.array(distances),axis=1)                     # Ordena las distancias de menor a mayor
        img=idx[:][-k:]      # Se eligen los k elementos más cercanos
```



```
        # Se calcula los grupos a los que pertenecen los k elementos más cercanos
        tam=len(query_descriptors)
        cl=np.array([[map_dataset['clusters_idx'][idx[i][-j]] for j in range(1,k+1)] for i in range(tam)])

        return cl, img

    elif criterion=='EUC':   # Distancia Euclídea
        distances=app_dist_euc(query_descriptors,map_dataset['feats']) # Calcula la distancia euclida para todas las consultas c
        idx=np.argsort(np.array(distances),axis=1)                     # Ordena las distancias de menor a mayor
        img=idx[:,:k]        # Se eligen los k elementos más cercanos

        # Se calcula los grupos a los que pertenecen los k elementos más cercanos
        tam=len(query_descriptors)
        cl=np.array([[map_dataset['clusters_idx'][idx[i][j]] for j in range(k)] for i in range(tam)])  # Array with the cluster

        return cl, img

    else:
        print('CRITERION ERROR: not valid')
```

POSE DISTANCE

En este apartado se implementan los distintos criterios para determinar la distancia en pose entre las consultas y los grupos del mapa.

Criterion 1: Mode of K-neighbors SE2 Pose

Este criterio mide la distancia en SE(2) desde las poses de la consulta hasta cada pose del mapa y calcula el K vecino más cercano. Devuelve los grupos a los que pertenecen la mayoría de los k-vecinos.

In [ ]:
```
def get_mode(List):
    # Función para calcular la moda en una lista de grupos del mapa topológico
    aux=np.zeros(max(List)+1).tolist()
    for item in List:
        aux[item]+=1
    return np.argmax(aux)
```

In [ ]:
```
def mode_K_Neighbours(query_poses, map_dataset, thr_ang, k=5):
    # Array de respuesta, con los grupos a los que pertenecen cada consulta
    res=[]
```

```
    # Combina las poses del mapa en una clase de Se(2)
    map_poses=geo.combine(map_dataset['poses'])

    for i in range(len(query_poses)):
        pq=query_poses[i]   # Pose de consulta

        # Calcula la composición de poses de cada consulta a todos las poses del mapa
        incre= pq / map_poses
        dist=[np.linalg.norm(incre.t()[i]) for i in range(len(incre))]  # Extrae la translación
        ang_dist=(incre.R().as_euler('xyz'))[:,2]                       # Estrae la rotación

        # Ordena las distancias de menor a mayor
        sort_idx=np.argsort(dist)

        assert len(dist) > k

        # Bucle para calcular los grupos de los k elementos más cercanos dentro del umbral de distancia angular
        temp_idx=[]
        for j in sort_idx:
            if np.abs(ang_dist[j]) < thr_ang:               # Distancia angular menor al umbral determinado
                cluster_idx=map_dataset['clusters_idx'][j]  # Índice del grupo al que pertenecen los k elementos
                temp_idx.append(cluster_idx)
            if len(temp_idx)>=k: break  # Cuando el número de índices supera el valor k se detiene el bucle

        # Llama a la función 'get_mode' para calcular la moda de los k índices de grupos para cada consulta
        res.append(get_mode(temp_idx))

    return res
```

Criterion 2: Most nearest image

Este criterio devuelve el grupo cuyo elemento tiene la menor distancia en SE(2) para cada consulta, considerandolo el grupo más cercano.

In [ ]:
```
def nearest(query_poses, map_dataset, thr_ang=0.3):
    # Array de respuesta, con los grupos a los que pertenecen cada consulta
    res=[]

    # Combina las poses del mapa en una clase de Se(2)
    map_poses=geo.combine(map_dataset['poses'])
```



```
    # Para cada consulta:
    for i in range(len(query_poses)):
        pq=query_poses[i]   # Pose de Consulta

        # Calcula composición de poses entre la consulta y los
        incre= pq / map_poses
        dist=[np.linalg.norm(incre.t()[i]) for i in range(len(incre))]  # Extrae la translación
        ang_dist=(incre.R().as_euler('xyz'))[:,2]                       # Extrae la rotación

        # Ordena distancias de menor a mayor
        sort_idx=np.argsort(dist)

        # Bucle para obtener el grupo de la topología con elemento mas cercano a la consulta
        for j in sort_idx:
            if np.abs(ang_dist[j]) < thr_ang:               # Asegurar que cumple el umbral en la distancia angular
                res.append(map_dataset['clusters_idx'][j])  # Para cada consulta se asigna el grupo del mapa cuyo elemento este
                break

    return res
```

Criterion 3: Mahalanobis

Este criterio implementa la distancia de Mahalanobis entre un conjunto de consultas y las distribuciones de poses de los elementos de la topología.

Para ello se emplea la media y la matriz de covarianza de cada distribución.

In [ ]:
```
def mahdist_to_cluster(query_dataset,map_dataset,select_idx):
    mahdist=[]

    # Se recorre el array de poses de consultas:
    for pose in query_dataset['poses'][select_idx]:
        distances=[]
        # Se calcula la distancia de Mahalanobis de la pose de consutla a cada grupo con la libería geometry2
        for idx in range(len(map_dataset['poses_mean'])):
            distances.append(geo.multivariate_mahalannobis(pose,
                                                           map_dataset['poses_mean'][idx],
                                                           map_dataset['poses_cov'][idx]))
        mahdist.append(np.array(distances).T)

    return np.array(mahdist)
```

CRITERIOS DE EVALUACIÓN

En esta parte se implementan los dos criterios de evaluación empleados para evaluar la topología de lugares propuesta.

Criterion 1: Matriz de Confusión Multiclase

Con este criterio se calcula la Matriz de Confusión Multiclase para el sistema de VPR.

De ella se extra la precisión y sensibilidad tanto a nivel de grupos como a nivel general.

In [ ]:
```
def cm_analysis(y_true, y_pred, labels, ymap=None, figsize=(10,10)):
#   Función para plotear matriz de confusión multiclase con anotaciones.
#   - Input:
#         y_true    ---> Valor verdadero (ground-truth) de las consultas
#         y_pred    ---> Predicción de grupos de las consutlas
#         labels    ---> Etiquetas de las clases de las consutlas, array de string
#         figsize   ---> Tamaño del plot

    # Calcula la matriz de confusión multiclase con el método de sklearn
    cm = confusion_matrix(y_true, y_pred)
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
```



```
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]

                annot[i, j] = '%d' % (c)
            elif c == 0:
                annot[i, j] = ''
            else:
                annot[i, j] = '%d' % (c)
    cm = pd.DataFrame(cm, index=labels, columns=labels)
    cm.index.name = 'Ground truth'
    cm.columns.name = 'Predicción'
    fig, ax = plt.subplots(figsize=figsize)
    sns.heatmap(cm, annot=annot, fmt='', ax=ax,cmap='BuPu',vmin=0, vmax=10,linewidths=1,linecolor='black')
    plt.title("Matriz de confusión multiclase")
```

In [ ]:
```
def confusion_mat(query_GT, query_predict, n_clusters):
#   Calcula la matriz de confusión multiclase para todos los grupos del mapa topológico
#   Resultado distinto del método de cm de sklearn, útil para evaluación de Precisión y Sensibilidad a nivel de grupos
#   - Input:
#         query_GT          ---> Valor verdadero (ground-truth) de las consultas
#         query_predict     ---> Predicción de grupos de las consutlas
#         n_cluster         ---> Número de grupos totales del mapa topológico
#   - Output:
#         confm             ---> Matriz de confusión multiclase

    # Inicializa la matriz de confusión multiclase
    tam=n_clusters # Número de grupos totales
    confm=np.reshape(np.zeros(tam**2),[tam,tam])

    # Rellena casillas de matriz de confusión multiclase
    for i in range(len(query_GT)):
        confm[query_GT[i],np.array(query_predict)[i]]+=1

    return confm
```

In [ ]:
```
def cluster_eval(query_dataset, map_dataset, select_idx, remove=True):
#   Función que implementa la evaluación de la precisión y sensibilidad del sistema de VPR a nivel de grupos
#   - Input:
#         query_dataset ---> Dataset de consulta
```

```
#         map_dataset   ---> Dataset del mapa topológico
#         select_idx    ---> Índices de conjuntos de consultas seleccionados sobre para la evaluación
#         remove        ---> Booleano para determinar si se eliminan los grupos no seleccionados en la consulta
#   - Output:
#         confm         ---> Matriz de confusión multiclase


    # Crear etiquetas para el plot
    n_clusters=len(map_dataset['clusters_list'])
    label=['C%s'%(i) for i in range(n_clusters)]

    # Calcular matriz de ground-truth (m x n) en base a la pose más cercana en el mapa   / m= nº de querys, n= nº de grupos
    query_GT=nearest(query_dataset['poses'][select_idx],map_dataset)

    # Calcula matriz de predicciones (mxn) para las consultas en base a la apariencia
    query_predict=most_similar_centroid(query_dataset['feats'][select_idx],map_dataset)

    # Calcula matriz de confusión multiclase
    confm=confusion_mat(query_GT,query_predict,n_clusters)

    # Calcula número de True Positive, False Positive y False Negative
    TP=np.diag(confm)                       # TP = Valor en la diagonal
    FP=confm.sum(axis=0) - np.diag(confm)   # FP = Suma de filas menos valor de TP
    FN=confm.sum(axis=1) - np.diag(confm)   # FN = Suma de columnas menos valor de TP

    # Calcula precisión y sensibilidad
    epsilon=1e-10   # Sumado para evitar divisiones por 0
    P=TP/(TP+FP+epsilon)
    R=TP/(TP+FN+epsilon)

    # Elimina los grupos que no se seleccionan en la clasificación para representación visual más clara
    if remove:
        idx=np.arange(n_clusters)
        good_idx = np.union1d(np.argwhere(confm.sum(0) > 0).reshape([-1]), np.argwhere(confm.sum(1) > 0).reshape([-1]))
        assert len(idx)==confm.shape[0]

        P=P[good_idx]
        R=R[good_idx]
        label=['C%s'%(i) for i in good_idx]

    return (label,P), (label,R)
```


In [ ]:

```
def general_eval(query_dataset, map_dataset, select_idx):
#   Función que implementa la evaluación de la precisión, sensibilidad y exactitud del sistema de VPR a nivel general
#   - Input:
#         query_dataset ---> Dataset de consulta
#         map_dataset   ---> Dataset del mapa topológico
#         select_idx    ---> Índices de conjuntos de consultas seleccionados sobre para la evaluación
#   - Output:
#         confm         ---> Matriz de confusión multiclase

    # Calcular matriz de ground-truth (m x n) en base a la pose más cercana en el mapa   / m= nº de querys, n= nº de grupos
    query_GT=nearest(query_dataset['poses'][select_idx],map_dataset)

    # Calcula matriz de predicciones (mxn) para las consultas en base a la apariencia
    query_predict=most_similar_centroid(query_dataset['feats'][select_idx],map_dataset)

    # Calcula matriz de confusión multiclase con el método de sklearn
    confm=confusion_matrix(query_GT,query_predict)

    # Calcula número de True Positive, True Negative, False Positive y False Negative
    TP = np.diag(confm).sum(0)          # TP = Suma de la diagonal de matriz de confusión multiclase
    FP = confm.sum(axis=0).sum() - TP   # FP = Suma de todas las filas de la matriz de confusión multiclase menos el valor de TP
    FN = confm.sum(axis=1).sum() - TP   # FN = Suna de todas las columnas de la matriz de confusión multiclase menos el valor de
    TN = 0                              # TN = Suma de todas las filas y columnas distintas de la clase sobre la que se calcula.
    for i in range(confm.shape[0]):     # Se suma el valor para todas las clases
        for k in range(confm.shape[0]):
            for j in range(confm.shape[1]):
                if k!=i and j!=i :
                    TN+=confm[k][j]

    # Calcula la precisión, sensibilidad y exactitud generales:
    P=TP/(TP+FP)
    R=TP/(TP+FN)
    ACC = (TP+TN)/(TP+TN+FP+FN)

    return P, R, ACC
```

Criterion 2: Silhouette score

Con este criterio se calcula la puntuación de silueta para todos los elementos del mapa topológico a fin de determinar la existencia de Perceptual Aliasing en la clasificación. Se basa en el cálculo de la distancia de Mahalanobis

In [ ]:
```
def sh_bar(score,xlabel,k=4):
#   Función que plotea el diagrama de barras para visualizar el resultado de la puntuación de silueta
#   - Input:
#         score            ---> Array con valores de SH (puntuación de silueta)
#         xlabel           ---> Titulo del plot
#         k                ---> Numero de intervalos del rango [-1,1] para crear el diagrama de barras
#   - Output:
#         (label, graph)   ---> Etiquetas para el diagrama de barras y diagrama de barras

    assert k>1

    # Ordenamos el array de score:
    sorted_arr = np.sort(score)
    n = len(sorted_arr)

    #Dividimos el rango [-1,1] en k partes iguales
    bin_edges = np.linspace(-1,1,k+1).tolist()
    print('Bin edges: ',bin_edges)
    graph = np.zeros(k,dtype=int)

    # Incrementamos para cada valor de score el intervalo correspondiente
    for i in range(n):
        j = np.searchsorted(bin_edges, sorted_arr[i], side='right')-1
        graph[j] += 1

    # Creamos el diagrama de barras:
    label=['[%s, %s)'%(np.round(bin_edges[i],2),np.round(bin_edges[i+1],2)) for i in range(k)]
    # Colores rojo y verde para clasificaciones erroneas y correctas
    my_color=['red' for i in range(len(bin_edges)) if bin_edges[i]<0]
    my_color.extend(['green' for i in range(len(bin_edges)-1) if bin_edges[i]>=0])

    # Show plot
    fig,ax=plt.subplots(figsize=(7,5))
    bars=plt.bar(label,graph,width=0.5,color=my_color)
    ax.bar_label(bars)
    plt.title(xlabel)
    plt.xlabel('Intervalos de SH')
    plt.show()

    return (label,graph)
```


In [ ]:
```
def map_pose_silhouette_score(map_dataset, select_idx):
#   Esta función calcula la puntuación de silueta para los elementos del mapa topológico.
#   Para cada elemento, calcula la distancia de Mahalanobis de su pose a todos las distribuciones de pose del mapa. La puntuació
#   comparando las distancias de cada elemento con su grupo y el grupo más cercano distinto del suyo.
#   - Input:
#         map_dataset      ---> Dataset del mapa topológico
#         select_idx       ---> Índices seleccionados del mapa, por lo general cogemos todos los índices
#   - Output:
#         score            ---> Array con puntuación de silueta para cada elemento del mapa

    # Inicialización de array de SH:
    score=np.zeros(len(select_idx))

    # Calcula distancia de Mahalanobis los elementos del mapa seleccionados y todas las distribuciones de pose del mapa
    mahdist=mahdist_to_cluster(map_dataset,map_dataset,select_idx)


    for i in range(len(select_idx)):
        # Selección de distancia de Mahalanobis del elemento a su grupo: map_dataset['clusters_idx'][select_idx[i]]
        a=mahdist[i][map_dataset['clusters_idx'][select_idx[i]]]

        # Eliminamos distancia de Mahalanobis del grupo al que pertenece el elemento
        dist=np.copy(mahdist[i])
        dist=np.delete(dist,map_dataset['clusters_idx'][select_idx[i]])

        # Selecciona la distancia de Mahalanobis del elemento al grupo mas cercano distinto al suyo
        sort_dist=np.sort(dist)
        b=sort_dist[0]

        # Formula de puntuación de silueta:
        score[i]=(b-a)/np.max([a,b])

    return score
```

Testing

In [ ]:
```
# Directorios del mapa topológico
map_json=".../Utils/Topological_map.json"
```

Generamos un diccionario con la información del dataset de consulta.

In [ ]:
```
# Creación del diccionario:
query_dict=dict(im_paths=[], poses=[], odom_poses=[], features=[], seq=[], test=[])

query_path='.../query/'    # Directorio de dataset de consulta

# Carga poses y paths:
for json_suffix in os.listdir(query_path+"Poses/"):
    json_file=json.load(open(query_path+"Poses/"+json_suffix))
    samples=len(json_file['im_paths'])
    query_dict['im_paths']+=json_file['im_paths']
    query_dict['poses']+=json_file['poses']
    query_dict['odom_poses']+=json_file['odom_poses']
    query_dict['seq']+=[json_suffix]*samples

# Cargar descriptores de la imágen de consultas
descr_path=query_path+'Descriptors/'
features= get_descriptor(descr_path).tolist()
query_dict['features']+=features
```

In [ ]:
```
# Procesamos las poses de consultas para representarlas en el espacio SE(2)
query_dataset = load_utils.load_sequence(query_dict,verbose=False)
pose_query = query_dataset['poses']
feats_query = query_dataset['feats']
```

Seleccionamos índices del dataset de consulta para la evaluación

In [ ]:
```
tam=len(pose_query)
select_idx=np.arange(0,tam,5)  #Consultas seleccionadas

pose_test=pose_query[select_idx]
feats_test= feats_query[select_idx]
```

Procesamos información de pose del mapa topológico en SE(2) y se extrae información

In [ ]:
```
map_dataset = load_utils.load_sequence(json.load(open(map_json)),verbose=False) # Procesar poses en SE(2)

pose_map = geo.combine(map_dataset['poses'])
feats_map = map_dataset['feats']
```



```
clusters_map = map_dataset['clusters_list']     # Lista de grupos
cl_clasif = map_dataset['clusters_idx']         # Vector de clasificación de elementos por grupos
```

Calcular la media de los descriptores y la media y la matriz de covarianza de las poses por grupos

In [ ]:
```
pose_mean, pose_cov = pose_mean_cov(map_dataset)
```

Calculamos Puntuación de Silueta para el Mapa Topológico

In [ ]:
```
tam=len(pose_map)
map_select_idx=np.arange(tam)  #Seleccionamos todos los elementos el mapa

# Llamada a la función de puntuación de silueta
map_sh_score=map_pose_silhouette_score(map_dataset, map_select_idx)
```

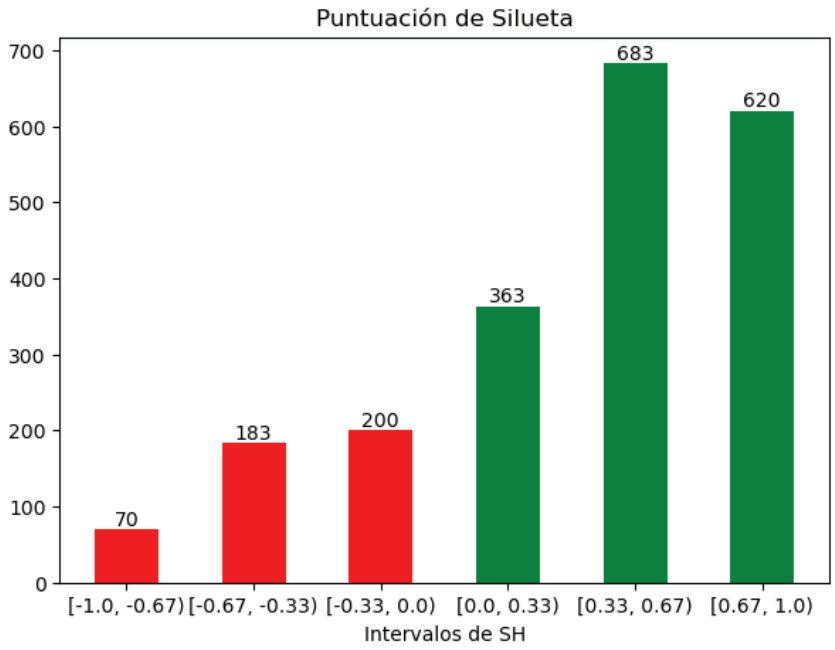
Se calcula el porcentaje de elementos bien clasificados, evaluando el signo de la puntuación de silueta

In [ ]:
```
map_sh_score_mean=(np.sign(map_sh_score)).mean()
```

Plot del diagrama de barras de la puntuación de silueta

In [ ]:
```
plt.figure(figsize=[10,6])

map_pose_sh_diagram=sh_bar(map_sh_score,xlabel='Puntuación de Silueta',k=6)
```

```
Bin edges:  [-1.0, -0.6666666666666667, -0.33333333333333337, 0.0, 0.33333333333333326, 0.6666666666666665, 1.0]
<Figure size 1000x600 with 0 Axes>
```



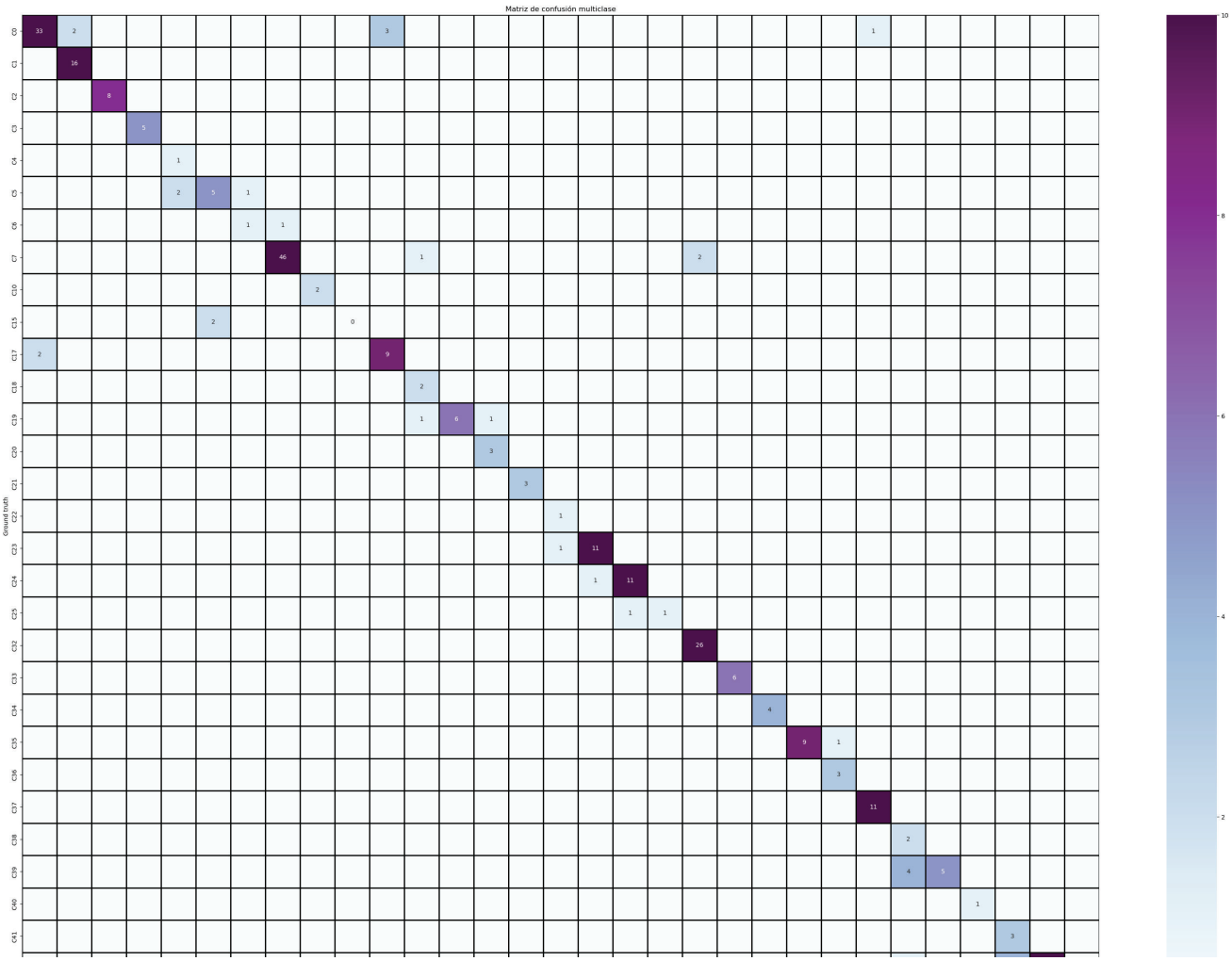
Evaluación del sistema de VPR

Cálculo y plot de la matriz de confusión multiclase

In [ ]:
```
# Calcular matriz de ground-truth (m x n) en base a la pose más cercana en el mapa   / m= nº de querys, n= nº de grupos
query_GT=nearest(query_dataset['poses'][select_idx],map_dataset)

# Calcula matriz de predicciones (mxn) para las consultas en base a la apariencia
query_predict=most_similar_centroid(query_dataset['feats'][select_idx],map_dataset)
query_predict=query_predict.tolist()

# Lista de grupos visitados para las etiquetas sobre la matriz de confusión multiclase
```



```
visited_clusters=np.unique(np.union1d(query_GT,query_predict))
my_labels=["C"+str(i) for i in visited_clusters]

# Plot matriz de confusión multiclase
cm_analysis(query_GT,query_predict,my_labels,figsize=(40,30))
```



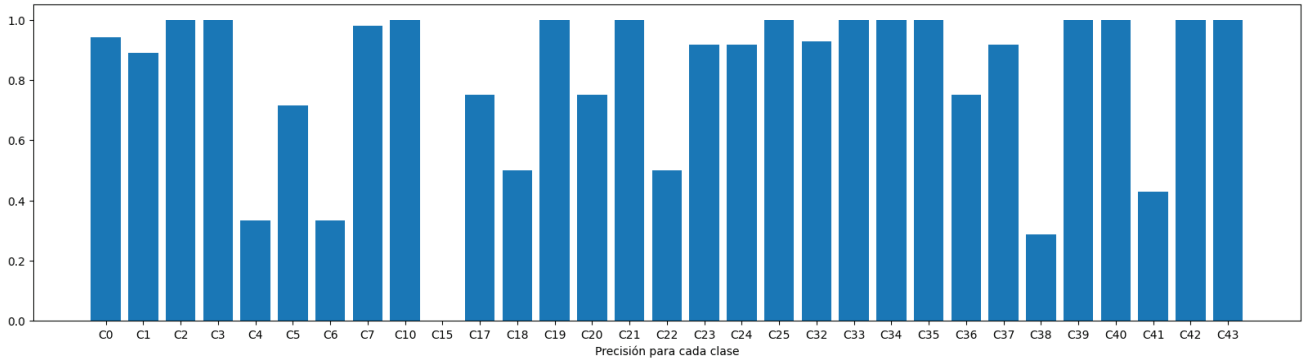
Precision y Sensibilidad a nivel de grupos

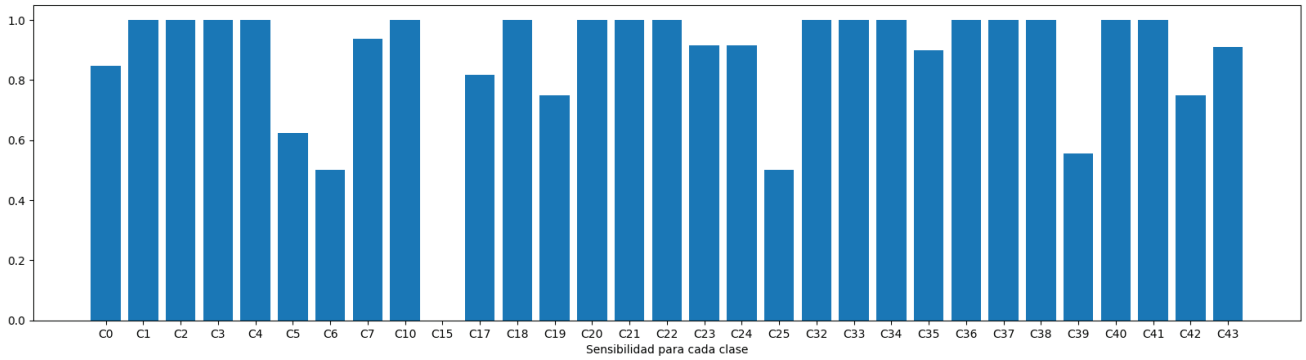
Llamamos a la función

In [ ]:
```
P, R= cluster_eval(query_dataset, map_dataset, select_idx, remove=True)
```

Plot del diagrama de barras

In [ ]:
```
plt.figure(figsize=[20,5])
plt.bar(P[0],P[1],width=0.8)
plt.xlabel('Precisión para cada clase')
plt.show()

plt.figure(figsize=[20,5])
plt.bar(R[0],R[1],width=0.8)
plt.xlabel('Sensibilidad para cada clase')
plt.show()
```





Calculamos la media de la precisión y sensibilidad a nivel de grupos-

In [ ]:
```
P_mean=np.mean(P[1])
R_mean=np.mean(R[1])
```

Precision, Sensibilidad y Exactitud a nivel general

Llamamos a la función

In [ ]:
```
P_, R_, ACC_ =general_eval(query_dataset, map_dataset,select_idx)
```

Resultados

In [ ]:
```
print('Mean Precission in clusters: ',100*P_mean,'%')
print('Mean Recall in clusters: ',100*R_mean,'%')
print('Precision general: ', 100*P_,'%')
print('Recall general: ',100*R_,'%')
print('Accuracy general: ',100*ACC_,'%')
print('Error :',100*(1-ACC_),'%')
```



```
Mean Precission in clusters:  80.11057728783581 %
Mean Recall in clusters:  86.85835797329572 %
Precision general:  88.6986301369863 %
Recall general:  88.6986301369863 %
Accuracy general:  99.27087936367653 %
Error : 0.7291206363234659 %
```

Resultado general de la evaluación

In [ ]:
```
print('Media SH score intra map: ',map_sh_score_mean,' ~ ',100*((map_sh_score_mean+1)/2),'%')
print('Mean Precission in clusters: ',100*P_mean,'%')
print('Mean Recall in clusters: ',100*R_mean,'%')
print('Precision general: ', 100*P_,'%')
print('Recall general: ',100*R_,'%')
print('Accuracy general: ',100*ACC_,'%')
print('Error :',100*(1-ACC_),'%')
```

```
Media SH score intra map:  0.5724398301085417  ~  78.62199150542708 %
Mean Precission in clusters:  80.11057728783581 %
Mean Recall in clusters:  86.85835797329572 %
Precision general:  88.6986301369863 %
Recall general:  88.6986301369863 %
Accuracy general:  99.27087936367653 %
Error : 0.7291206363234659 %
```

Plot VPR map

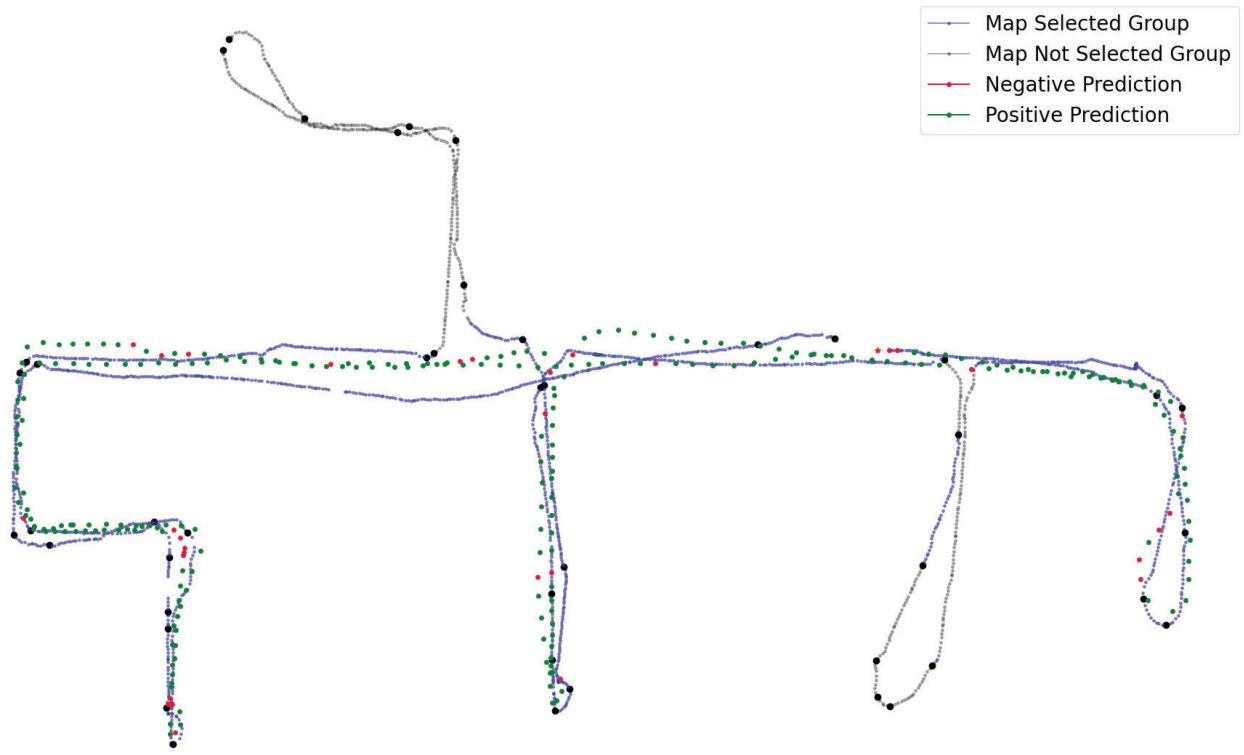
En esta sección se plotea el mapa topológico y las consultas, indicando si estan bien o mal clasificadas.

In [ ]:
```
# Calcular matriz de ground-truth (m x n) en base a la pose más cercana en el mapa   / m= nº de querys, n= nº de grupos
query_GT=nearest(query_dataset['poses'][select_idx],map_dataset)

# Calcula matriz de predicciones (mxn) para las consultas en base a la apariencia
query_predict=most_similar_centroid(query_dataset['feats'][select_idx],map_dataset)
query_predict=query_predict.tolist()

# Lista de grupos a los que se asigna una consulta
visited_clusters=np.unique(np.union1d(query_GT,query_predict))
```

Ploteamos el mapa

In [ ]:
```
# Creamos la figura:
fig=plt.figure(figsize=(23,14))
ax = fig.add_subplot(111)
plt.plot(0,0,marker='*',color='red')    # Marcamos el origen del mapa (0,0)

# Plot del mapa topológico
n_clusters=len(clusters_map)
for i in range(n_clusters):
    subgraph=np.copy(clusters_map[i])   # Copia del grupo

    for k,item in enumerate(clusters_map[i]):   # Recorremos grupos
        # Extraemos posición de elementos:
        translation=geo.SE2Poses.t(pose_map[item])
        new_x=translation[0]
        new_y=translation[1]
        # Si es un grupo visitado se plotea en azul, si no en gris:
        if i in visited_clusters:
            plt.plot(new_x,new_y,markersize=2.5,marker='o',color='b',alpha=0.4,label="Map Selected Group")
        else:
            plt.plot(new_x,new_y,markersize=2.5,marker='o',color='black',alpha=0.3,label="Map Not Selected Group")
    # Ploteamos un punto negro para delimitar los grupos
    plt.plot(new_x,new_y,markersize=6,marker='o',color='black')


# Plot de consultas:
pose_consulta=[geo.SE2Poses.t(pose_test[j]) for j in range(len(pose_test))] # Poses de consutlas

for j in range(len(pose_consulta)):
    # Si la predicción es correcta se plotea la posición en verde
    if query_GT[j] == query_predict[j]:
        plt.plot(pose_consulta[j][0],pose_consulta[j][1],markersize=4,marker='o',color='g',label='Positive Prediction')

    # Si la predicción no es correcta se plotea la posición en rojo
    else:
        plt.plot(pose_consulta[j][0],pose_consulta[j][1],markersize=4,marker='o',color='#DC143C',label="Negative Prediction")

# Show plot
ax.legend()
handles, labels = ax.get_legend_handles_labels()
lgd = dict(zip(labels, handles))
ax.legend(lgd.values(), lgd.keys(),fontsize="20")
plt.axis('off')
```


Out[ ]:
```
(-16.24954285, 6.53292785, -5.6359072, 4.6513412)
```



Bibliografía

[1] J. Porta y B. Krose, «Appearance-based concurrent map building and localization using a multi-hypotheses tracker,» en *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 4, 2004, 3424-3429 vol.4. DOI: 10.1109/IROS.2004.1389946.

[2] T. Sattler, B. Leibe y L. Kobbelt, «Fast image-based localization using direct 2D-to-3D matching,» en *2011 International Conference on Computer Vision*, 2011, págs. 667-674. DOI: 10.1109/ICCV.2011.6126302.

[3] F. Camposeco, A. Cohen, M. Pollefeys y T. Sattler, «Hybrid Scene Compression for Visual Localization,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2019.

[4] C. Valgren y A. J. Lilienthal, «SIFT, SURF seasons: Appearance-based long-term localization in outdoor environments,» *Robotics and Autonomous Systems*, vol. 58, n.º 2, págs. 149-156, 2010, Selected papers from the 2007 European Conference on Mobile Robots (ECMR '07), ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2009.09.010. dirección: https://www.sciencedirect.com/science/article/pii/S0921889009001493.

[5] O. Vysotska y C. Stachniss, «Lazy Data Association For Image Sequences Matching Under Substantial Appearance Changes,» *IEEE Robotics and Automation Letters*, vol. 1, n.º 1, págs. 213-220, 2016. DOI: 10.1109/LRA.2015.2512936.

[6] M. Lopez Antequera et al., «Técnicas de visión por computador para calibración, localización y reconocimiento.,» 2021.

[7] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla y J. Sivic, «NetVLAD: CNN architecture for weakly supervised place recognition,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 5297-5307.

[8] A. Torii, J. Sivic, T. Pajdla y M. Okutomi, «Visual place recognition with repetitive structures,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, págs. 883-890.

[9] D. Anguelov, C. Dulong, D. Filip et al., «Google street view: Capturing the world at street level,» *Computer*, vol. 43, n.º 6, págs. 32-38, 2010.

[10] A. J. Gálvez, «Appearance-based Localization,» PhD thesis, Departamento de Sistemas y Autom
atica. Universidad de M
alaga, 2023.

[11] D. Gálvez-López y J. D. Tardos, «Bags of binary words for fast place recognition in image sequences,» *IEEE Transactions on Robotics*, vol. 28, n.º 5, págs. 1188-1197, 2012.

[12] M. Cummins y P. Newman, «FAB-MAP: Probabilistic localization and mapping in the space of appearance,» *The International Journal of Robotics Research*, vol. 27, n.º 6, págs. 647-665, 2008.

[13] Y. Latif, G. Huang, J. Leonard y J. Neira, «Sparse optimization for robust and efficient loop closing,» *Robotics and Autonomous Systems*, vol. 93, págs. 13-26, 2017.

[14] J.-L. Blanco, J. González y J.-A. Fernández-Madrigal, «Subjective local maps for hybrid metric-topological SLAM,» *Robotics and Autonomous Systems*, vol. 57, n.º 1, págs. 64-74, 2009.

[15] A. Kendall, M. Grimes y R. Cipolla, «Posenet: A convolutional network for real-time 6-dof camera relocalization,» en *Proceedings of the IEEE international conference on computer vision*, 2015, págs. 2938-2946.

[16] S. Brahmbhatt, J. Gu, K. Kim, J. Hays y J. Kautz, «Geometry-aware learning of maps for camera localization,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, págs. 2616-2625.

[17] T. Sattler, M. Havlena, K. Schindler y M. Pollefeys, «Large-scale location recognition and the geometric burstiness problem,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 1582-1590.

[18] M. Lopez-Antequera, N. Petkov y J. Gonzalez-Jimenez, «City-scale continuous visual localization,» en *2017 European Conference on Mobile Robots (ECMR)*, IEEE, 2017, págs. 1-6.

[19] M. J. Milford y G. F. Wyeth, «SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights,» en *2012 IEEE international conference on robotics and automation*, IEEE, 2012, págs. 1643-1649.

[20] S. Garg y M. Milford, «Fast, compact and highly scalable visual place recognition through sequence-based matching of overloaded representations,» en *2020 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2020, págs. 3341-3348.

[21] S. Garg y M. Milford, «Seqnet: Learning descriptors for sequence-based hierarchical place recognition,» *IEEE Robotics and Automation Letters*, vol. 6, n.º 3, págs. 4305-4312, 2021.

[22] M. Volkov, G. Rosman, D. Feldman, J. W. Fisher y D. Rus, «Coresets for visual summarization with applications to loop closure,» en *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, págs. 3638-3645.

[23] A. Jaenal, F.-A. Moreno y J. Gonzalez-Jimenez, «Unsupervised appearance map abstraction for indoor Visual Place Recognition with mobile robots,» *IEEE Robotics and Automation Letters*, vol. 7, n.º 3, págs. 8495-8501, 2022.

[24] K. Mikolajczyk y C. Schmid, «A performance evaluation of local descriptors,» *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, n.º 10, págs. 1615-1630, 2005.

[25] D. G. Lowe, «Distinctive image features from scale-invariant keypoints,» *International journal of computer vision*, vol. 60, págs. 91-110, 2004.

[26] H. Bay, T. Tuytelaars y L. Van Gool, «SIFTR(Scale Invariant Feature Transform),» *Lecture notes in computer science*, vol. 3951, págs. 404-417, 2006.

[27] H. Bay, T. Tuytelaars y L. Van Gool, «Surf: Speeded up robust features,» *Lecture notes in computer science*, vol. 3951, págs. 404-417, 2006.

[28] E. Rublee, V. Rabaud, K. Konolige y G. Bradski, «ORB: An efficient alternative to SIFT or SURF,» en *2011 International conference on computer vision*, Ieee, 2011, págs. 2564-2571.

[29] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi y T. Pajdla, «24/7 place recognition by view synthesis,» en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, págs. 1808-1817.

[30] G. Csurka, C. Dance, L. Fan, J. Willamowski y C. Bray, «Visual categorization with bags of keypoints,» en *Workshop on statistical learning in computer vision, ECCV*, Prague, vol. 1, 2004, págs. 1-2.

[31] H. Jégou, M. Douze, C. Schmid y P. Pérez, «Aggregating local descriptors into a compact image representation,» en *2010 IEEE computer society conference on computer vision and pattern recognition*, IEEE, 2010, págs. 3304-3311.

[32] F. Perronnin y C. Dance, «Fisher kernels on visual vocabularies for image categorization,» en *2007 IEEE conference on computer vision and pattern recognition*, IEEE, 2007, págs. 1-8.

[33] F. Radenović, G. Tolias y O. Chum, «Fine-tuning CNN image retrieval with no human annotation,» *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, n.º 7, págs. 1655-1668, 2018.

[34] S. Lowry, N. Sünderhauf, P. Newman et al., «Visual place recognition: A survey,» *ieee transactions on robotics*, vol. 32, n.º 1, págs. 1-19, 2015.

[35] S. Schubert, P. Neubert, S. Garg, M. Milford y T. Fischer, «Visual Place Recognition: A Tutorial,» *arXiv preprint arXiv:2303.03281*, 2023.

[36] S. Lowry, N. Sünderhauf, P. Newman et al., «Visual place recognition: A survey,» *ieee transactions on robotics*, vol. 32, n.º 1, págs. 1-19, 2015.

[37] C. Masone y B. Caputo, «A survey on deep visual place recognition,» *IEEE Access*, vol. 9, págs. 19 516-19 547, 2021.

[38] H. Taira, M. Okutomi, T. Sattler et al., «InLoc: Indoor visual localization with dense matching and view synthesis,» en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, págs. 7199-7209.

[39] T. Naseer, L. Spinello, W. Burgard y C. Stachniss, «Robust visual robot localization across seasons using network flows,» en *Proceedings of the AAAI conference on artificial intelligence*, vol. 28, 2014.

[40] M. Xu, T. Fischer, N. Sünderhauf y M. Milford, «Probabilistic appearance-invariant topometric localization with new place awareness,» *IEEE Robotics and Automation Letters*, vol. 6, n.º 4, págs. 6985-6992, 2021.

[41] M. Xu, N. Snderhauf y M. Milford, «Probabilistic visual place recognition for hierarchical localization,» *IEEE Robotics and Automation Letters*, vol. 6, n.º 2, págs. 311-318, 2020.

[42] M. J. Milford y G. F. Wyeth, «SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights,» en *2012 IEEE international conference on robotics and automation*, IEEE, 2012, págs. 1643-1649.

[43] R. Hartley y A. Zisserman, «Epipolar Geometry and the Fundamental Matrix,» en *Multiple View Geometry in Computer Vision*, 2.ª ed. Cambridge University Press, 2004, págs. 239-261. DOI: 10.1017/CBO9780511811685.014.

[44] *Epipolar geometry - Wikipedia*. dirección: https://en.wikipedia.org/wiki/Epipolar_geometry.

[45] E. A. Cárdenas Quiroga, L. Y. Morales Martın y A. Ussa Caycedo, «La estereoscopıa, métodos y aplicaciones en diferentes áreas del conocimiento,» *Revista Cientıfica General José Marıa Córdova*, vol. 13, n.º 16, págs. 201-219, 2015.

[46] T. Liao y N. Li, «Natural Image Stitching Using Depth Maps,» *arXiv preprint arXiv:2202.06276*, 2022.

[47] J. L. Blanco-Claraco, *A tutorial on* **SE**(3) *transformation parameterizations and on-manifold optimization*, 2022. arXiv: 2103.15980 [cs.RO].

[48] G. Bradski, «The openCV library.,» *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, n.º 11, págs. 120-123, 2000.

[49] *OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)*. dirección: `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`.

[50] *OpenCV: Feature Matching*. dirección: `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`.

[51] *OpenCV: Epipolar Geometry*. dirección: `https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html`.

[52] J. L. Crowley y F. Pourraz, «Continuity properties of the appearance manifold for mobile robot position estimation,» *Image and Vision Computing*, vol. 19, n.º 11, págs. 741-752, 2001.

[53] J. Lee, *Introduction to topological manifolds*. Springer Science & Business Media, 2010, vol. 202.

[54] A. C. Sankaranarayanan, C. Hegde, S. Nagaraj y R. G. Baraniuk, «Go with the flow: Optical flow-based transport operators for image manifolds,» en *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2011, págs. 1824-1831.

[55] K.-C. Lee, J. Ho, M.-H. Yang y D. Kriegman, «Video-based face recognition using probabilistic appearance manifolds,» en *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, IEEE, vol. 1, 2003, págs. I-I.

[56] A. Jaenal, F.-A. Moreno y J. Gonzalez-Jimenez, «Appearance-based sequential robot localization using a patchwise approximation of a descriptor manifold,» *Sensors*, vol. 21, n.º 7, pág. 2483, 2021.

[57] A. Jaenal, F.-A. Moreno y J. Gonzalez-Jimenez, «Appearance-based sequential robot localization using a patchwise approximation of a descriptor manifold,» *Sensors*, vol. 21, n.º 7, pág. 2483, 2021.

[58] S. Luengo, A. Jaenal, F. A. Moreno, A. J. Gonzalez-Jimenez et al., «Dimensionality Reduction in images for Appearance-based camera Localization,» 2022.

[59] J. Shi y J. Malik, «Normalized cuts and image segmentation,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, n.º 8, págs. 888-905, 2000. DOI: `10.1109/34.868688`.

[60] B. Mohar, Y. Alavi, G. Chartrand y O. Oellermann, «The Laplacian spectrum of graphs,» *Graph theory, combinatorics, and applications*, vol. 2, n.º 871-898, pág. 12, 1991.

[61] *Grafo ponderado - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Grafo_ponderado.

[62] *Grafo no dirigido - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Grafo_no_dirigido.

[63] A. Álvarez Suárez et al., «Estudio de limitaciones teóricas y aproximaciones computacionales al problema del coloreado espectral de grafos,» 2016.

[64] *Similitud coseno - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Similitud_coseno.

[65] D. Fernandez-Chaves, J. R. Ruiz-Sarmiento, N. Petkov y J. González-Jiménez, «Robot@VirtualHome dataset,» jul. de 2021. DOI: 10.5281/ZENODO.4610098. dirección: https://doi.org/10.5281/zenodo.4610098#.ZHaNBBYknCI.mendeley.

[66] *RANSAC - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/RANSAC.

[67] A. Pronobis y B. Caputo, «COLD: The CoSy localization database,» *The International Journal of Robotics Research*, vol. 28, n.º 5, págs. 588-594, 2009.

[68] *Matriz laplaciana - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Matriz_laplaciana.

[69] J. Shi y J. Malik, «Normalized cuts and image segmentation,» *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, n.º 8, págs. 888-905, 2000.

[70] O. Veksler, «Image segmentation by nested cuts,» en *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, IEEE, vol. 1, 2000, págs. 339-344.

[71] *Árbol binario - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/%C3%81rbol_binario.

[72] *Depth-first search - Wikipedia*. dirección: https://en.wikipedia.org/wiki/Depth-first_search.

[73] *Silhouette (clustering) - Wikipedia*. dirección: https://en.wikipedia.org/wiki/Silhouette_%28clustering%29.

[74] P. J. Rousseeuw, «Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,» *Journal of computational and applied mathematics*, vol. 20, págs. 53-65, 1987.

[75] *Distancia de Mahalanobis - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Distancia_de_Mahalanobis.

[76] *Matriz de covarianza - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/wiki/Matriz_de_covarianza.

[77] *Confusion matrix - Wikipedia*. dirección: https://en.wikipedia.org/wiki/Confusion_matrix.

[78] *Project Jupyter | Home*. dirección: https://jupyter.org/.