ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Estudio de técnica de generación de modelos 3D a partir de secuencias de imágenes

(Study of Structure from Motion algorithm)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Carlos Abad García

Septiembre - 2017

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Carlos Abad García
Director del TFG: Pablo Pedro Sánchez Espeso
Título: "Estudio de técnica de generación de modelos 3D a partir de secuencias de imágenes"
Title: "Study of Structure from Motion algorithm"

Presentado a examen el día: 20 de septiembre de 2017

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Sánchez Espeso, Pablo Pedro
Secretario (Apellidos, Nombre): Juárez Crespo, José Ángel
Vocal (Apellidos, Nombre): Fernández Solórzano, Víctor

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

A mi familia, especialmente a mis padres, por el apoyo y dedicación que siempre han prestado, haciendo posible que haya llegado hasta aquí.

A mi tutor, Pablo, por la confianza depositada y descubrirme un nuevo campo, además de su labor de enseñanza durante los dos últimos cursos de la carrera.

A mis amigos, por estar ahí día tras día y siempre poder contar con ellos.

Resumen

En este trabajo se realiza un estudio de la técnica *Structure from Motion*. Para ello, se analizan todos los pasos que se han de realizar en el algoritmo, presentando un análisis detallado desde el punto de vista teórico de todos ellos. Además, se desarrolla una implementación en C++ del algoritmo y se evalúa su rendimiento. Finalmente, a partir de dicho análisis se proponen mejoras a realizar en los ámbitos de la calidad de reconstrucción, rendimiento y escalabilidad con el fin de servir como mejoras de cara a futuras implementaciones en sistemas embebidos.

Palabras clave: *Structure from Motion* (*SfM*), Reconstrucción 3D, Visión Artificial (*Computer visión*), Punto de interés (*keypoint*), Descriptor, *Bundle Adjustment*, Triangulación, Matriz de proyección, *Tracking*, *Mapping*.

Abstract

This document presents an analysis of the *Structure from Motion* technique. All the steps which take part in the algorithm are studied, presenting a detailed analysis of all of them from the theoretical point of view. On top of that, a C++ implementation is developed and evaluated. Finally, from the results of the evaluation, improvements regarding reconstruction quality, performance and scalability are proposed in order to improve future implementations in embedded systems.

Keywords: Structure from Motion (*SfM*), 3D Reconstruction, Computer vision, Keypoint, Descriptor, Bundle Adjustment, Triangulation, Projection matrix, Tracking, Mapping.

Índice

Capítulo 1 - Introducción

1.1 Introducción 1
1.2 Objetivos 3
1.3 Estructura del documento 4

Capítulo 2 - Estado del arte

2.1 Sistemas de reconstrucción y captación de imagen tridimensional 5
2.1.1 Sistemas con sensores específicos 5
LIDAR 6
Kinect: sensor con luz estructurada y cámara de tiempo de vuelo 6
2.1.2 Sistemas sin sensores de profundidad 7
Cámaras estéreo 7
Structure from Motion 8
2.2 Aplicaciones de la técnica SfM 8
2.2.1 Recuperación del patrimonio 8
2.2.2 Topografía 9
2.2.3 Medicina e Ingeniería 10
2.2.5 SLAM 11
2.3 Estimación de profundidad mediante métodos directos o basados en puntos de interés 12
2.4 Modelo de cámara 14
2.4.1 Introducción 14
2.4.2 Modelo pinhole 14
2.5 Terminología empleada en el trabajo 17

Capítulo 3 - Estudio del algoritmo SfM

3.1 Introducción 19
3.2 Detección de puntos de interés y correspondencias 21
3.2.1 ORB 22
Oriented FAST 23
Rotation-Aware BRIEF 25
3.2.2 Detección de correspondencias 26

3.3 Reconstrucción inicial 27
3.3.1 Movimiento de la cámara 27
Matriz fundamental y geometría epipolar 27
Estimación de la matriz fundamental 29
3.3.2 Triangulación 32
Estimación inicial 33
Método iterativo lineal 34
3.4 Perspective-n-Point 35
3.5 Bundle Adjustment 37

Capítulo 4 - Evaluación de la implementación

4.1 Estructura del algoritmo 41
4.2 Detalles sobre la implementación 44
4.2.1 OpenCV 44
Funciones de lectura de imágenes 44
Detección de puntos de interés y correspondencias 45
Obtención de la matriz fundamental 46
Resolución del problema Perspective-n-Point 47
4.2.2 SSBA 48
4.2.3 PCL 48
4.2.4 Búsqueda de las correspondencias entre puntos de una imagen y la nube de puntos 48
4.2.5 Mecanismos adicionales de filtrado de correspondencias 49
4.2.6 Distribución del código en ficheros 50
4.2.7 Uso de las librerías en el algoritmo 50
4.3 Evaluación 52
4.3.1 Calidad de la reconstrucción 52
Implementación 1: triangulación no iterativa y filtrado por distancia 53
Implementación 2: filtrado empleando la estimación de F 54
Implementación 3: triangulación iterativa 55
Implementación 4: reconstrucción con una mala inicialización 55
Visualización de las cámaras 55
4.3.2 Rendimiento del algoritmo 56
Número de puntos de interés empleado 56
Detección de puntos de interés 56
Detección de correspondencias 57

Estimación de la matriz fundamental 57
Triangulación 57
Búsqueda de puntos en la nube 57
Resolución de PnP 58
Bundle adjustment 58
4.3.3 Escalabilidad 59
4.3.4 Conclusión 59
4.4 Mejoras propuestas 60
4.4.1 Calidad de la reconstrucción 60
4.4.2 Rendimiento 60
Detección de correspondencias 61
Localización de puntos en la nube 61
Bundle Adjustment 61
4.4.3 Separación de tracking y mapping 62

Capítulo 5 - Conclusión

5.1 Conclusiones del trabajo 65

Bibliografía

Índice de figuras

Figura 2.1: barridos del LIDAR 6
Figura 2.2: luz estructurada en Kinect 6
Figura 2.3: cámara estéreo 7
Figura 2.4: reconstrucciones llevadas a cabo por BigSfM 9
Figura 2.5: estudio de un deslizamiento de tierras 10
Figura 2.6: reconstrucción de una vértebra 11
Figura 2.7: aplicación de un método directo 12
Figura 2.8: método basado en puntos de interés 13
Figura 2.9: geometría del modelo *pinhole* 15
Figura 2.10: relación entre las coordenadas de la cámara y el espacio 3D 16

Figura 3.1: SfM secuencial 20
Figura 3.2: SfM jerárquico 20
Figura 3.3: muestra de píxeles de *FAST* 23
Figura 3.4: geometría *epipolar* 28
Figura 3.5: las soluciones para la reconstrucción a partir de E 32
Figura 3.6: triangulación y error de reproyección 33
Figura 3.7: error de reproyección 38

Figura 4.1: resultados de la reconstrucción con la primera implementación 53
Figura 4.2: resultados de la reconstrucción con la segunda implementación 54
Figura 4.3: posición de las cámaras 55
Figura 4.4: análisis del rendimiento 58
Figura 4.5: *tracking* y *mapping* 63

Índice de tablas

Tabla 2.1 : métodos basados en puntos de interés y directos, ventajas y desventajas..... 13

Acrónimos

SfM: *Structure from motion*

SLAM*: Simultaneous Location and Mapping*

vSLAM*: Visual Simultaneous Location and Mapping*

UAV: *Unmanned Aerial Vehicle*

RANSAC: *Random sample consensus*

LIDAR: *Light Detection and Ranging o Laser Imaging Detection and Ranging*

RGB: *Red, Green, Blue*

SIFT: *Scale Invariant Feature Transform*

SURF: *Speeded-Up Robust Features*

FAST: *Features from Accelerated Segment Test*

BRIEF: *Binary Robust Independent Elementary Features*

ORB: *Oriented Fast and Rotated Brief*

SVD*: Singular Value Decomposition*

PnP*: Perspective-n-Point*

DLT: *Direct Linear Transform*

SSBA*: Simple Sparse Bundle Adjustment*

PCL: *Point Cloud Library*

Capítulo 1
Introducción

1.1 Introducción

Proporcionar a los ordenadores, robots y máquinas automatizadas la capacidad de *ver* es un problema que se consideraba prácticamente trivial en los años 60, cuando la informática comercial todavía daba sus primeros pasos. De hecho, en el MIT se propuso resolver el problema de la identificación de objetos como un proyecto de verano [40]. Sin embargo, medio siglo más tarde se trata de uno de los problemas de actualidad en el ámbito de la computación, todavía sin resolver, y estudiado en el campo de la **Visión Artificial** (*Computer Vision*). Est campo reúne técnicas basadas en numerosas disciplinas (informática, álgebra, geometría, estadística…) y se beneficia de los avances más recientes (como el aprendizaje automático, *Machine Learning*) para tratar de adquirir imágenes, procesarlas y analizarlas con el fin de extraer información de ellas como lo haría una persona.

Una de las técnicas de la visión artificial que está cobrando importancia últimamente es **SfM** (*Structure from Motion*), cuyo objetivo es reconstruir una estructura tridimensional empleando secuencias de imágenes. Esta técnica es similar al funcionamiento del cerebro humano, que también es capaz de obtener información de la profundidad a partir de la percepción del movimiento en las imágenes plasmadas en la retina. Además, resuelve el problema clásico de reconstrucción del entorno en áreas tales como fotogrametría. El algoritmo de SfM emplea correspondencias entre puntos de interés detectados en las imágenes. A partir de dichos puntos obtiene la posición de las cámaras para, posteriormente, triangular los puntos reconstruyendo la escena. La base teórica se asienta en el problema geométrico de obtener la posición de los puntos de una escena a partir de la posición de varios puntos en dos vistas de la misma. La primera solución de este problema fue propuesta en 1913 por Kruppa [2], que demostró que a partir de 5

puntos que aparecen en dos vistas es posible reconstruir la posición y orientación de las cámaras a una escala desconocida. Posteriormente, fue con el trabajo de Longuet-Higgins [5] cuando se introdujo la ***matriz esencial*** (***E***), que puede ser descompuesta para obtener la posición y orientación relativa de las cámaras, siempre y cuando se trate de cámaras calibradas. Más tarde, Quang-Tuan Luong introdujo la ***matriz fundamental*** (***F***), una generalización que refleja la ***geometría epipolar*** del problema, y puede ser estimada de manera sencilla a partir de un mínimo de 8 correspondencias entre puntos de las imágenes. Estos son los dos resultados básicos sobre los que se asienta la técnica SfM, y sobre los que se darán más detalles en el capítulo 3. Se trata de un algoritmo muy valioso, al reconstruir de forma precisa una escena y, además, determinar la posición desde la cual se han tomado las fotografías. Por ello cuenta con numerosas aplicaciones en diversos campos como la medicina, la geografía, la arqueología y la propia ingeniería. Hasta ahora, se ha basado su aplicación en reconstrucciones *offline*, es decir, realizadas una vez recogidas las imágenes y en las que el tiempo de cómputo para la reconstrucción no es vital. Esto se debe a que el paso más importante en el algoritmo es un ajuste (*bundle adjustment*) que se ha considerado siempre demasiado costoso para su aplicación en tiempo real.

Uno de los problemas tradicionales en el ámbito de la robótica (*Simultaneous Location And Mapping*, **SLAM**), consiste en elaborar un mapa del entorno a la vez que se lleva a cabo un seguimiento de la posición de un vehículo autónomo: es decir, se sitúa a un robot en un entorno completamente desconocido, y el sistema es capaz de reconstruir su entorno y conocer en dónde se encuentra. Se trata, por ello, de una técnica con enorme potencial, con gran número de posibles aplicaciones, y que permite la navegación totalmente autónoma de vehículos y robots, además de poder ser aplicada en otros campos, como la cirugía guiada por imágenes [6]. Los primeros trabajos que estudian este problema aparecen en la década de 1980 [7]. Pero no es hasta este milenio cuando ganan importancia y atención. Uno de los hitos que lo propiciaron fue la victoria de los coches autónomos STANLEY y JUNIOR (elaborados por el equipo dirigido por Sebastian

Thrun en el desafío DARPA en 2005 y que incorpora técnicas SLAM a sus sistemas de navegación autónoma. Desde entonces, distintas técnicas se han empleado, sobre todo orientadas hacia la navegación autónoma para tratar de resolver este problema. La mayoría de las técnicas se han basado en filtros y métodos estadísticos [8], junto con la inclusión de sensores más allá de la cámara (profundidad, GPS…).

Recientemente se ha realizado numerosos esfuerzos para tratar de elaborar sistemas SLAM que emplean únicamente imágenes de su entorno (vSLAM) basados en los principios empleados en SfM. Es decir, realizar una reconstrucción SfM secuencial en tiempo real obteniendo las imágenes de una secuencia de vídeo. Por lo tanto, se trata de un reto de gran actualidad, el mejorar los sistemas SfM existentes para tratar de conseguir un buen rendimiento en tiempo real y poder extender su aplicación al ámbito SLAM. De hecho, en una de las últimas conferencias internacionales de visión por computador (*ICCV 2015*) se ha dedicado un día completo a tratar exclusivamente este tema [9].

1.2 Objetivos

En este proyecto se presenta un estudio de la técnica Structure from Motion. Se analiza la base teórica del algotimo y los distintos pasos que lo integran, con el fin de servir no solo como estudio para la realización del resto del trabajo, sino como referencia de futuros desarrollos, conteniendo una explicación autosuficiente sobre la generación de un modelo en tres dimensiones del entorno a partir de una secuencia de imágenes tomadas por una única cámara.

Además, se lleva a cabo una implementación software de dicha técnica y se analiza su rendimiento. También se presentan las posibles mejoras a realizar para reducir el tiempo de ejecución y aumentar la escalabilidad del algoritmo. Para ello, se tienen en cuenta las diferentes técnicas puestas en práctica en sistemas SfM y SLAM fruto de la investigación realizada en la materia, así como propuestas realizadas a partir del análisis de la implementación.

De esta forma, se pretende que la implementación y labor de análisis realizada en este trabajo sirva como base para elaborar en el futuro un sistema SfM capaz de operar en tiempo real, una de las metas más perseguidas en el ámbito actual de la visión por computador.

1.3 Estructura del documento

El trabajo se estructura en cuatro capítulos fundamentales:

- En el **capítulo 2** se presenta el estado del arte en reconstrucción tridimensional, presentando las alternativas disponibles para captación de información tridimensional y sus aplicaciones. Además, se analizan las ventajas y desventajas del uso de métodos directos o basados en puntos de interés, y se expone el modelo de cámara empleado en el proyecto.

- En el **capítulo 3** se lleva a cabo una explicación detallada sobre los diferentes pasos que integra el algoritmo SfM, presentando su base teórica.

- En el **capítulo 4** se introduce la implementación realizada, detallando su estructura y elementos para, posteriormente, realizar un análisis de su desempeño. Finalmente, se presentan las posibles mejoras a realizar para obtener el mejor rendimiento posible.

- Para terminar, en el **capítulo 5**, se lleva a cabo una conclusión del trabajo realizado.

Capítulo 2
Estado del arte

En este capítulo se analiza el estado del arte en métodos de reconstrucción y captación de información tridimensional. Se presentan aplicaciones del algoritmo SfM, una breve aproximación al debate en torno al uso de métodos directos o basados en puntos de interés, así como el modelo de cámara que se empleará en el resto del trabajo. Para terminar, se hará una pequeña aclaración sobre la terminología empleada en el trabajo.

2.1 Sistemas de reconstrucción y captación de imagen tridimensional

A lo largo de los últimos años se han propuesto numerosas técnicas para resolver el problema de obtener un modelo tridimensional de una escena. Para ello, no solo se han empleado algoritmos basados en el procesado de imagen, sino que también se ha intentado obtener información sobre la profundidad de una escena mediante diversos sensores específicos. Por lo tanto, se va a dividir el análisis de los sistemas de reconstrucción tridimensional en 2 partes: sistemas que captan información empleando sensores específicos, y técnicas que únicamente usan imágenes.

2.1.1 Sistemas con sensores específicos

Lo más común consiste en emplear algún sensor capaz de detectar la profundidad para obtener información sobre dicho parámetro, siendo la imagen la que aporta el resto de información visual.

LIDAR

La técnica LIDAR (*Laser Imaging Detection and Ranging*) consiste en emitir un haz pulsado desde un emisor laser y medir su señal reflejada, con el fin de obtener información de profundidad midiendo el tiempo que tarda la señal en ser reflejada. Para obtener información de toda una escena se aplican distintos patrones para recorrerla. Recientemente se está aplicando en topografía para generar mapas a gran escala (con el inconveniente de la gran cantidad de datos generada), y en conducción autónoma, con el posible inconveniente de que el láser no se refleja bien en la piel humana o en superficies mojadas. Su principal ventaja es la precisión obtenida en la medida de la profundidad.

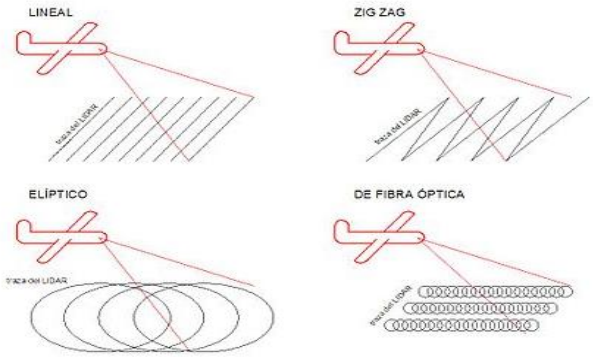

***Figura 2.1*: barridos del LIDAR**

Kinect: sensor con luz estructurada y cámara de tiempo de vuelo

Una tecnología desarrollada por Microsoft, e inicialmente orientada a su uso en videojuegos, pronto se ha convertido en un sistema empleado para el reconocimiento de gestos, siendo posible usar su principio de funcionamiento para obtener medidas de profundidad. Cuenta con tres elementos principales: una cámara RGB, un emisor de infrarrojos y una cámara infrarroja. Su funcionamiento es el siguiente: con el emisor infrarrojo se emite un patrón de puntos (todos diferentes entre sí) a toda la escena, y, aprovechando que la cámara infrarroja se encuentra desplazada con respecto al emisor y se conoce las posiciones de ambas, emplea el principio de visión estéreo y triangula la profundidad de cada punto. Esta información es incorporada a la captada por la cámara RGB tradicional. Los principales inconvenientes de esta tecnología son que únicamente funcionan en interiores (la luz solar oculta el patrón emitido), y que no es posible emplear dos cámaras iguales en la misma sala, al interferir una en el patrón de la otra.

Figura 2.2: luz estructurada en kinect

En la segunda generación de Kinect, se introduce otro mecanismo conocido como las cámaras de tiempo de vuelo. Estas cámaras emplean el mismo principio que el LIDAR: medir el tiempo

que tarda una señal emitida en rebotar y volver a la posición del emisor, calculando con esa información la distancia. La diferencia con estos sistemas reside en que, en este caso se emplea un emisor de infrarrojos que ilumina toda la escena a la vez, empleando un receptor capaz de determinar el tiempo de retraso con el que vuelve la señal para diferentes puntos de la escena. Se han desarrollado cámaras de tiempo de vuelo que emplean RF en lugar de infrarrojos, o pulsos de infrarrojos codificados para poder operar en exteriores. Sin embargo, el problema de la interferencia sigue estando presente.

Los sistemas de este tipo, que combinan cámaras y un sensor extra que aporta información sobre profundidad se suelen denominar RGB-D. El contar con este sensor facilita en gran medida la reconstrucción del entorno, empleándose este tipo de cámaras en multitud de aplicaciones SLAM y reconstrucción de estructuras, especialmente las que emplean láser (mayor precisión) en el caso de aplicaciones destinadas a recuperación del patrimonio o arqueología.

2.1.2 Sistemas sin sensores de profundidad

Cámaras estéreo

Uno de los sistemas más básicos para medir la profundidad se basa en el uso de cámaras estéreo, que emplean el mismo mecanismo que la visión humana para captar la profundidad. La aproximación asume que las imágenes de la cámara están corregidas y alineadas (imágenes equipolares), y, por lo tanto, la imagen de una línea de la cámara derecha se corresponde con la imagen de la misma línea de la cámara izquierda. Con las imágenes corregidas el algoritmo estéreo se basa en medir la diferencia de posición de cada punto entre una cámara y otra: de esta forma, los puntos que más se mueven se encontrarán más cerca de las cámaras, y cuanto menos se muevan (menor disparidad) más lejos se hallarán. Esto es posible debido a que las cámaras ven los elementos de la escena desde distintos ángulos, siendo un principio análogo al empleado en SfM. Esta tecnología surge a mediados del siglo XIX, aunque alcanza su madurez y mayor desarrollo con la llegada de las cámaras digitales. Dichas cámaras se han empleado, por ejemplo, para el estudio de otros planetas (integradas en la sonda Pathfinder), en el ámbito de la medicina (exploración), en topografía y guiado de robots/drones.

Figura 2.3*: *cámara estéreo

Structure from Motion

Por último, el objeto de este trabajo es el estudio de la técnica SfM, que se detallará en el capítulo siguiente. Se trata de la más prometedora de las mencionadas, al necesitar únicamente el movimiento de una cámara para obtener toda la información sobre el entorno. Sus principales desventajas residen en que, en general, la reconstrucción realizada no integra ninguna información sobre la escala a la que se realiza: es decir, a no ser que se cuente con una referencia conocida en la imagen, no es posible extraer información sobre las dimensiones absolutas de los elementos reconstruidos. Además, no es capaz de funcionar correctamente ante un movimiento de rotación en ausencia de traslación (rotar la cámara sobre sí misma).

A pesar de ello, se trata de un sistema en auge debido a su gran aplicabilidad tanto a teléfonos móviles como vehículos o robots. La principal ventaja reside en el uso de una sola cámara, extendiendo en gran medida la capacidad de realizar reconstrucciones del entorno, al no haber necesidad de costosos sensores extra, o la preparación de un sistema estéreo, con su correspondiente calibración y corrección de imágenes. Además, la ausencia de sensores o cámaras extra reduce en una medida notable la cantidad de datos a almacenar pasa su posterior procesado. Es por ello que su uso e importancia ha ido aumentando, conforme las técnicas empleadas para su implementación han ido mejorando su desempeño, superando los principales obstáculos iniciales, como el gran tiempo de cómputo necesario, y la menor precisión obtenida frente a los sistemas con sensores como el láser. Además, con la aparición de la realidad virtual y realidad mixta, SfM presenta una alternativa adecuada para integrar elementos del "mundo real" a escenas virtuales. A continuación, se van a presentar sus principales aplicaciones.

2.2 Aplicaciones de la técnica SfM

2.2.1 Recuperación del patrimonio

La posibilidad de obtener modelos tridimensionales a partir de imágenes ha abierto numerosas posibilidades en el ámbito de la recuperación del patrimonio. Por ejemplo, han surgido proyectos como [10, 11] que explotan la gran disponibilidad de imágenes de ciudades y monumentos que es posible obtener en internet, a partir de las fotografías tomadas por turistas. Dichos protectos emplean dichas imágenes en sistemas SfM robustos para reconstruir grandes monumentos, o incluso ciudades enteras con impresionantes resultados.

Por otra parte, también se ha aplicado SfM, por ejemplo, para realizar reconstrucciones tridimensionales de la ciudad de Palmira (Siria), recientemente destruida por DAESH, y que contiene gran cantidad de monumentos con incalculable valor histórico y cultural [12]. Gracias a la existencia de archivos fotográficos tomados en dicha ciudad, y la aplicación de técnicas SfM es posible "recuperar" una ciudad nombrada patrimonio de la humanidad en 1980. Se trata de un solo ejemplo entre numerosas iniciativas similares que se están llevando a cabo. Además, este tipo de aplicaciones se han disparado con la aparición de la tecnología de impresión 3D, permitiendo realizar modelos a escala de gran cantidad de obras que no pueden ser expuestas directamente al público. Otro ejemplo es el trabajo detallado en [13], en el que se recuperan esculturas y otros tesoros submarinos empleando fotografías realizadas por buzos.

Figura 2.4: Reconstrucciones llevadas a cabo por la herramienta BigSfM [11]. *(a) Ciudad vieja de Dubrovnik. (b) Monumentos de Roma. (c) Plaza y Basílica de San Marcos*

2.2.2 Topografía

La reconstrucción de mapas en 3D tiene numerosas aplicaciones en geografía y topología. Por ello, resulta muy útil contar con sistemas capaces de reconstruir el relieve del terreno empleando imágenes captadas por aviones, drones no tripulados (*UAV*) o incluso satélites. La aparición de

técnicas SfM ha impulsado la fotogrametría aérea, al no tener que emplear aeroplanos empleados con cámaras costosas, lo que ha permitido reducir el coste de generación de mapas. Con este fin, han surgido diferentes soluciones *software* optimizadas para este tipo de reconstrucciones, como [14], una plataforma open source impulsada por Kitware.

Además, aunque no se trate de una aplicación directa de las técnicas de SfM propiamente dicho, sí que usan sus resultados algunas herramientas usadas en la vida cotidiana de muchas personas, como lo son *google maps* y *google Street view*.

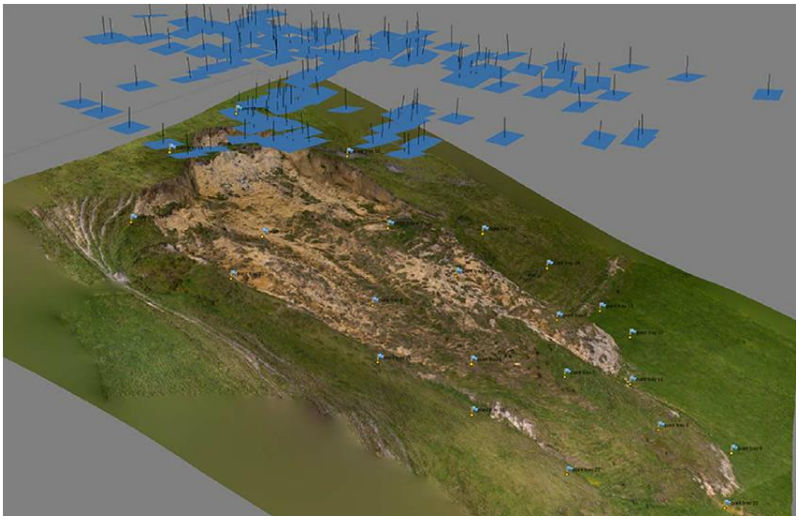

***Figura 2.5**: **Estudio de un deslizamiento de tierras** a partir de fotografías tomadas por un UAV* [15]

2.2.3 Medicina e Ingeniería

También se explota la combinación de fotografías y la impresión 3D para obtener modelos tridimensionales en la medicina e ingeniería.

En medicina, se viene empleando la impresión 3D en los últimos años para la realización de prótesis e implantes dentales, pero su uso se está extendiendo, habiendo incluso llegado a sustituir huesos o partes de órganos. SfM puede ser usada para generar modelos a bajo coste de partes de la anatomía humana a partir de secuencias de imágenes, tanto para fines médicos como educativos [16].

Figura 2.6: Reconstrucción de una vértebra [16]

Las mismas aplicaciones se pueden encontrar en el ámbito de la ingeniería, con la capacidad de obtener modelos de piezas sobre los que trabajar en cualquier momento, de forma rápida, barata y sencilla. Además, otra aplicación en el campo de la ingeniería es la capacidad de introducir modelos 3D en realidad aumentada o mixta, facilitando el entrenamiento de personal o el análisis de modelos y entornos.

2.2.5 SLAM

Hay que comentar que SfM y SLAM, a pesar de ser dos técnicas diferentes, resuelven problemas similares: la localización y mapeado simultáneo del entorlo (SLAM), y la obtención de la estructura tridimensional de una escena, y la posición desde la que se toman las imágenes (SfM). SLAM consiste en una solución en tiempo real empleando información que proviene de distintos sensores, mientras que SfM emplea únicamente imágenes, y no tiene por qué resolverse en tiempo real. Sin embargo, el vSLAM (SLAM empleando únicamente imágenes) puede ser resuelto empleando una variante de la técnica SfM que incorpore las imágenes de forma secuencial[1] y se lleve a cabo en tiempo real. vSLAM permite el guiado de drones, robots y vehículos autónomos. Sin embargo, la tecnología no está lo suficientemente desarrollada para integrarse en productos destinados al consumo, al contrario que en el resto de aplicaciones. Es por ello que se está llevando a cabo un gran esfuerzo para mejorar las técnicas actuales de SfM.

[1] Ver apartado 3.1

2.3 Estimación de profundidad mediante métodos directos o basados en puntos de interés

Existen dos aproximaciones para identificar el mismo punto en dos imágenes, usadas tanto en SfM como en SLAM para determinar la posición de un punto en 3 dimensiones: los métodos directos y los basados en puntos de interés

Los **métodos directos** emplean la intensidad de los píxeles, comparándolas y obteniendo las partes de una imagen que se corresponden con la otra, buscando minimizar el error fotogramétrico y creando un mapa de profundidad semi-denso. El error fotogramétrico mide la diferencia entre las intensidades de una imagen y otra desplazada sobre ella, de tal forma que las partes en común identificadas coincidan. Su principal ventaja reside en que se aprovecha más información de la imagen, y se obtienen reconstrucciones bastante más densas (con más puntos). Se emplea sobre todo en sistemas SLAM, a pesar de que existen métodos SfM que hacen uso de ellos.

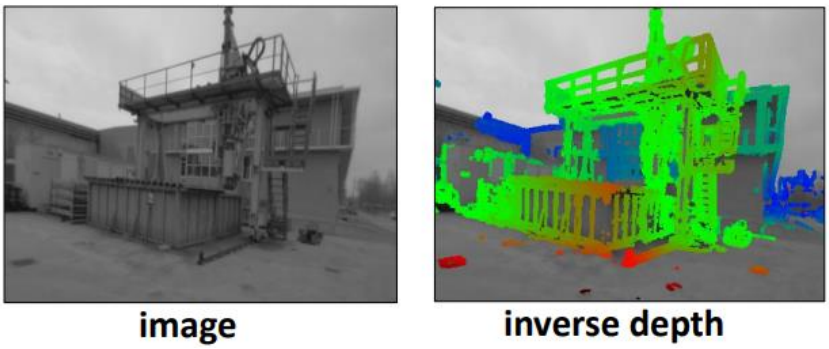

Figura 2.7: aplicación de un método directo [17]

Los métodos **basados en puntos de interés** buscan ciertos puntos de las imágenes que aportan más información (bordes, esquinas, zonas con textura…), y sus correspondencias en la otra imagen, tratando de minimizar en la reconstrucción el error de reproyección (distancia entre el punto 3D reconstruido y sus correspondencias en las imágenes). Además, se obtiene una estimación del movimiento de

Figura 2.8: *nube de puntos resultante de la aplicación de un método basado en* ***puntos de interés*** [18]

la cámara a partir de dichas correspondencias. Estos métodos proporcionan resultados menos densos (menor nube de puntos). Su principal ventaja es su mayor velocidad de cómputo, y su mayor precisión a la hora de obtener la posición de la cámara. Se emplea tanto en SfM como en SLAM, donde compite con los métodos directos.

A continuación, se presenta una tabla que recoge las principales ventajas y desventajas de cada uno de ellos.

Basados en puntos de interés	**Directos**
Únicamente pueden emplear y reconstruir puntos de interés	Emplean y reconstruyen toda la imagen
Más rápidos	Más lentos, aunque se paralelizan mejor
Flexible: se pueden eliminar puntos que no encajan en el modelo *a posteriori*	Inflexible: es complicado eliminar elementos reconstruidos anteriormente.
Robusto ante inconsistencias en el modelo (*rolling shutter*)	Poco robusto ante inconsistencias en el modelo
Detección de puntos de interés basada en información menos completa	Decisión de la alineación entre imágenes basada en información más completa
No necesitan buena inicialización	Necesitan una buena inicialización

Tabla 2.1: *métodos basados en puntos de interés y directos, ventajas y desventajas* [17]

En este trabajo, se ha decidido emplear métodos basados en puntos de interés, al considerarse que es necesario trabajar con el método que proporciona mayor velocidad de cómputo (principal problema de SfM), y siendo de vital importancia la precisión a la hora de seguir el movimiento

de la cámara. Además, a partir de las nubes de puntos generadas es posible elaborar un mallado y texturizado para obtener una reconstrucción densa *a posteriori* si es necesario.

2.4 Modelo de cámara

2.4.1 Introducción

En el ámbito de la visión artificial, se puede considerar a la cámara como un dispositivo que proyecta y mapea el mundo tridimensional en una imagen bidimensional. Existen varios modelos que recogen esta transformación, dependiendo de las características de la cámara empleada. El modelo más sencillo es el modelo *pinhole*, en el que se considera que todos los rayos de luz incidentes se proyectan en la imagen pasando a través de un único punto (centro). Este modelo es válido para cámaras con muy poca distorsión (la cual no se recoge en el modelo) y en las que los píxeles que conforman el sensor son cuadrados. A partir de él, se definen modelos más complejos que añaden diferentes parámetros: desde distintas dimensiones horizontal y vertical de los píxeles (válido para las cámaras CMOS/CCD si tienen un objetivo con lentes que introducen poca distorsión), hasta modelos en los que se reflejan las distorsiones provocadas por distintas lentes de apertura finita.

A partir del modelo *pinhole* es posible reflejar con suficiente precisión la transformación entre el mundo 3D y la imagen 2D, por lo que es el más utilizado en visión artificial con lentes estándar, y el que se emplea en este trabajo.

2.4.2 Modelo *pinhole*

La principal característica de este modelo es que la apertura de la cámara se reduce a un único punto (centro). Se realiza una proyección central de los puntos del espacio (3D) a un plano "imagen" (2D). Cada punto del espacio se proyecta a través de la línea que lo une con el centro

de la imagen, y las coordenadas del punto en la imagen son las del punto en el que dicha recta intersecta al plano imagen.

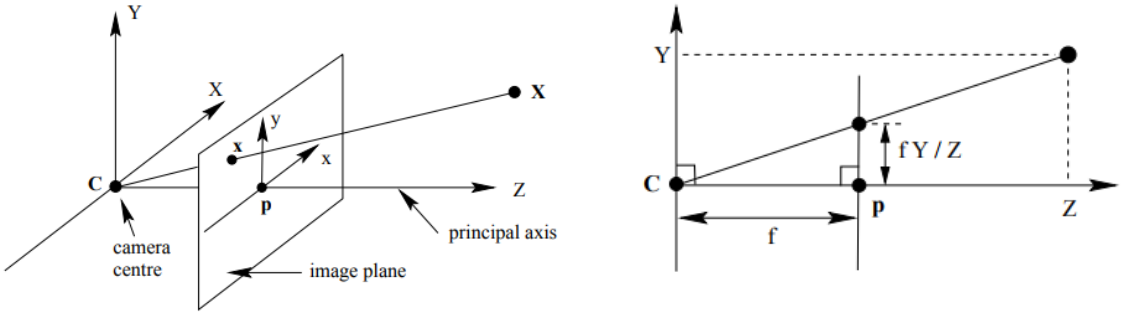

***Figura 2.9: Geometría del modelo Pinhole**: relación entre las coordenadas de la cámara y de la imagen* [1]

La distancia que separa al centro de la cámara del plano imagen se define como la distancia focal *f*. Empleando triángulos semejantes, es posible relacionar el punto en el espacio (3D) con el punto en la imagen (2D) de manera sencilla:

1. Proyectando a distancia focal: $(\mathrm{X_c}, \mathrm{Y_c}, \mathrm{Z_c})^{\mathrm{T}} \mapsto \left(\frac{f \cdot \mathrm{X_c}}{\mathrm{Z_c}}, \frac{f \cdot \mathrm{Y_c}}{\mathrm{Z_c}}, f\right)^{\mathrm{T}}$

2. Mapeando $\mathbb{R}^3 \mapsto \mathbb{R}^2$: $(\mathrm{X_c}, \mathrm{Y_c}, \mathrm{Z_c})^{\mathrm{T}} \mapsto (f\mathrm{X_c}/\mathrm{Z_c}, f\mathrm{Y_c}/\mathrm{Z_c})^{\mathrm{T}}$

Expresado matricialmente, si las coordenadas del mundo tridimensional y de la imagen se expresan como coordenadas homogéneas[2]:

$$\begin{pmatrix} \mathrm{X_c} \\ \mathrm{Y_c} \\ \mathrm{Z_c} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f\mathrm{X_c} \\ f\mathrm{Y_c} \\ \mathrm{Z_c} \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} \mathrm{X_c} \\ \mathrm{Y_c} \\ \mathrm{Z_c} \\ 1 \end{pmatrix}$$

Hasta ahora, se ha tenido en cuenta que el origen de coordenadas de la imagen es su centro (punto principal), lo cual no tiene por qué ser así. En general se toma como origen la esquina inferior izquierda de la imagen, lo que aboga a trasladar la ecuación anterior:

[2] Coordenadas empleadas por lo general en un espacio proyectivo, del que el espacio Euclídeo es un subespacio. Un punto en el espacio euclídeo n-dimensional se expresa con un vector de n+1 elementos definido hasta cierta escala. Multiplicar el vector por un factor distinto de cero no cambiará su valor. Para obtener el equivalente Euclídeo, basta con dividir todos los elementos por el elemento N+1 $\neq 0$.

$$\begin{pmatrix} fX_c + Z_c p_x \\ fY_c + Z_c p_y \\ Z_c \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \cdot \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \mathbf{K} \cdot [\mathrm{I}|0] \cdot \mathbf{X_c}, \qquad \mathbf{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix}$$

Siendo la matriz K la matriz de calibración de la cámara (o de parámetros intrínsecos).

El caso contemplado hasta ahora sitúa al centro de la cámara en el origen de coordenadas, y la cámara está apuntando en la dirección positiva del eje Z. En general, la cámara sobre la que se deseen proyectar los puntos del espacio estará situada en un punto distinto al origen. Para solucionarlo, se introduce un desplazamiento que transforma las coordenadas del origen en las del espacio tridimensional que tiene su origen en el centro de la cámara (*coordenadas de la cámara*, $\boldsymbol{X_c}$). Para ello se precisa una matriz de rotación (**R**) y un vector de traslación (**t**) que sitúan a la cámara en las coordenadas originales:

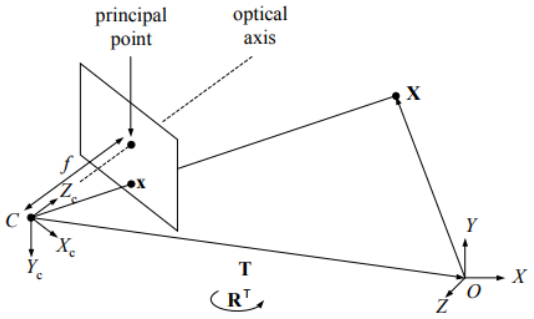

Figura 2.10: Relación entre las coordenadas de la cámara y el espacio tridimensional [1]

$$\begin{pmatrix} u \\ v \\ Z_c \end{pmatrix} = \mathbf{K} \cdot [\mathbf{I}|\mathbf{0}] \cdot [\mathbf{R}|\mathbf{t}] \cdot \mathbf{X} \rightarrow \boxed{\mathbf{u} = \mathbf{K} \cdot [\mathbf{R}|\mathbf{t}] \cdot \mathbf{X} = \mathbf{P} \cdot \mathrm{X}}$$

La ecuación anterior defina la matriz de proyección, **P**, que relaciona las coordenadas del espacio tridimensional (**X**), con su posición en la imagen (**u**). Tal y como se recoge en [1], esta transformación tiene una formulación alternativa, que contiene las coordenadas del centro de la cámara expresadas en las coordenadas del espacio tridimensional original (C):

$$\mathbf{u} = \mathbf{K} \cdot \mathbf{R} \cdot [\mathbf{I}|-\mathbf{C}] \cdot \mathbf{X}$$

Comparando esta ecuación con la anterior:

$$\mathbf{K} \cdot \mathbf{R} \cdot [\mathbf{I}|-\mathbf{C}] \cdot \mathbf{X} = \mathbf{K} \cdot [\mathbf{R}|\mathbf{t}] \cdot \mathbf{X} \rightarrow \mathbf{t} = -\mathbf{R} \cdot \mathbf{C}$$

Teniendo en cuenta que **R** es ortogonal (su inversa es igual a su traspuesta):

$$\mathbf{t} = -\mathbf{R} \cdot \mathbf{C} \rightarrow \mathbf{C} = -\mathbf{R}^{\mathbf{T}} \cdot \mathbf{t}$$

Por lo tanto, de la matriz de proyección de la cámara se obtiene de manera sencilla su posición.

Cabe comentar que las coordenadas de la imagen (**u**) están expresadas en unidades del espacio tridimensional. Para poder usar como unidad los píxeles de la cámara (lo que en el resto del trabajo se denominará **x**) basta sencillamente con multiplicar la matriz **K** por una matriz diagonal diag(m_x, m_y, 1), siendo m_x y m_y los píxeles por unidad de distancia en la dirección x e y (dichas distancias se expresan normalmente mm, la unidad en la que normalmente está expresada la focal) para recoger la posibilidad de píxeles no cuadrados.

A lo largo del trabajo se hará referencia en numerosas ocasiones a la matriz de proyección (**P**), a la de parámetros intrínsecos o de calibración (**K**), y a la de parámetros extrínsecos (**[R|t]**), que depende de la matriz de rotación **R** y del vector de traslación **t**. Además, hay que comentar que se empleará la técnica SfM asumiendo que se conoce la matriz **K**, es decir, que se conoce el modelo de cámara con el que se están tomando las fotografías, al pretender servir como base para un futuro sistema operando en tiempo real.

2.5 Terminología empleada en el trabajo

Antes de comenzar con el desarrollo teórico del trabajo, conviene aclarar dos términos que se van a emplear en numerosas ocasiones:

- **Correspondencia:** identificación del mismo punto de interés en dos imágenes. Es el resultado del procedimiento de *matching*. A partir de un punto de interés de una imagen, su correspondencia con otra imagen se forma con el punto de interés más parecido en la otra imagen. Para determinar cuál es el punto que más se parece se lleva a cabo una comparación de los descriptores[3] de los puntos. También se aplica a la identificación del punto de interés en una imagen y un punto ya reconstruido.

[3] Ver apartado 3.2

- Posiciones de la **vista, imagen o cámara:** en numerosas ocasiones se van a emplear estos tres términos para referirse a la posición en el espacio desde la que se ha tomado una fotografía empleada en la reconstrucción.

Capítulo 3
Estudio del algoritmo SfM

3.1 Introducción

Como se ha comentado en la introducción, *Structure from Motion* es la técnica que permite, a partir de la identificación del mismo conjunto de puntos (correspondencias) entre dos o más imágenes estimar la posición de las cámaras que han tomado las imágenes, así como la situación en el espacio (coordenadas 3D) de los puntos tomados como referencia. Para ello, aprovecha la variación espacial de los puntos comunes entre una imagen y la siguiente para obtener la posición desde la que se han tomado las imágenes. Posteriormente, se determina la posición de los puntos mediante triangulación para, finalmente, realizar un ajuste del resultado que minimice el error de reproyección. Los resultados obtenidos no tienen información alguna sobre la posición absoluta de ninguno de sus elementos: tan solo se obtienen posiciones relativas, por lo que si se realizase alguna transformación afín[4] sobre la posición de las cámaras el resultado obtenido sería el mismo. De esta forma, se obtiene un resultado con una escala arbitraria, que determina las posiciones relativas de las cámaras y los puntos seleccionados como referencia. Esto es un problema si se desea realizar medidas de dimensiones absolutas a partir de esta técnica, que obligaría a disponer de una referencia con dimensiones absolutas conocidas en las imágenes.

[4] Las transformaciones afines son aquellas que conservan las relaciones de colinealidad (y coplanaridad) entre puntos, es decir, puntos que recaían sobre una misma línea (o sobre un mismo plano) antes de la transformación, son preservados tras una transformación afín. Ejemplos de transformaciones agines son el escalado y el desplazamiento.

Al ser un método que tradicionalmente se realiza *offline*, es decir, conociendo todas las imágenes de antemano, han surgido diferentes variantes de SfM en cuanto a la forma de incorporar las sucesivas vistas a la escena reconstruida.

La variante más intuitiva es la **secuencial**. En ella se realiza una primera reconstrucción inicial, y a continuación se van añadiendo imágenes en el mismo orden en el que se tomaron, empleando las correspondencias entre los puntos ya reconstruidos y los de la nueva imagen para determinar la posición de las nuevas cámaras. Su principal desventaja es la aparición de un error acumulativo conocido como *drift*, que consiste en la aparición de diferentes reconstrucciones en 3D del mismo punto 2D, debidas a la acumulación de errores en la determinación de la posición de las cámaras.

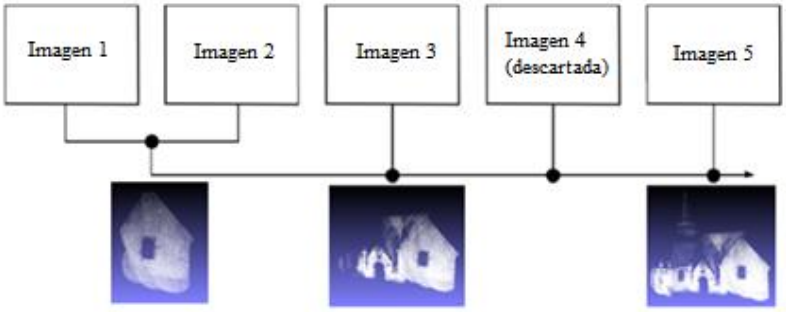

Figura 3.1: SfM secuencial [19]

Sin embargo, al tener que realizarse un ajuste tras cada incorporación para paliar el error mencionado este método es lento y poco preciso. Si se tienen disponibles las imágenes de antemano una alternativa es utilizar métodos **jerárquicos** que tratan de minimizar el número de ajustes a realizar. Para ello, se realizan reconstrucciones a partir de pares de vistas adyacentes, que se van procesando de manera jerárquica hasta obtener la escena final. Este proceso no es trivial, por estar la reconstrucción de cada par a una escala arbitraria, y con posición relativa desconocida, por lo que se emplean los puntos tridimensionales de cada par de vistas para orientar y escalar las reconstrucciones parciales (se buscan correspondencias y se hacen coincidir).

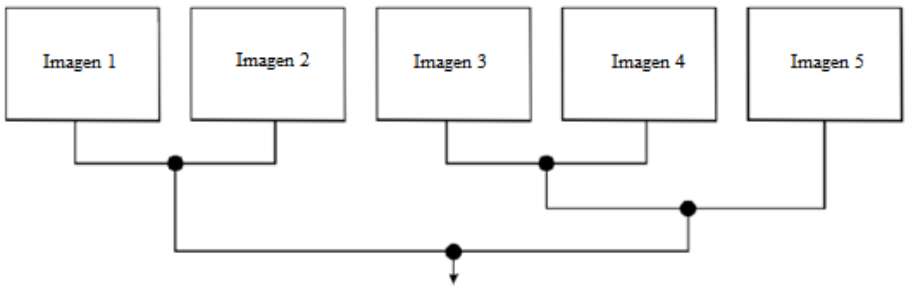

Figura 3.2: SfM jerárquico [19]

Para terminar, en [1] se presenta un método de **factorización** para optimizar el algoritmo de SfM. Consiste en tratar de estimar todas las posiciones de las cámaras al mismo tiempo. A partir de todas las rotaciones y traslaciones relativas obtenidas para cada par de imágenes, se obtienen las rotaciones y traslaciones globales mediante métodos denominados *rotation average* y *traslation average*. Finalmente, se triangulan todos los puntos en el espacio al mismo tiempo, siendo necesario un único ajuste final.

En este trabajo se va a presentar la aproximación secuencial a la reconstrucción, al ser la única de las aproximaciones presentadas que permite el trabajo en tiempo real, ya que no requiere el conocimiento de todas las vistas de antemano.

3.2 Detección de puntos de interés y correspondencias

Para llevar a cabo la reconstrucción de la escena, SfM emplea como principal fuente de información correspondencias entre imágenes. Como se ha mencionado en el capítulo 2, se va a estudiar un sistema basado en ***puntos de interés*** (***keypoints***) que recogen las características relevantes de la imagen. Estos puntos han de ser seleccionados cuidadosamente, ya que es vital para obtener una reconstrucción correcta que sean identificados adecuadamente en el resto de imágenes. Los puntos de interés se almacenan mediante ***descriptores***, que son elementos que almacenan la información sobre el entorno de los puntos de interés, y que posteriormente, son comparados entre sí para determinar las correspondencias (***matching***). En resumen, la técnica identifica puntos de interés en la imagen y los modela mediante descriptores que posteriormente serán utilizados para identificar al mismo punto en otra imagen.

Históricamente, se ha comprobado que funcionan mejor aquellos puntos de interés alrededor de los cuales hay mayor diferencia de intensidad en los píxeles (mayor gradiente). Esto se debe a que suelen representar bordes de elementos dentro de la imagen. Además, es interesante que sean invariantes frente a la rotación (como, por ejemplo, esquinas [20]) y a la escala, ya que es habitual encontrar transformaciones de este tipo entre una imagen y otra.

Existen numerosas aproximaciones para obtener los puntos de interés y formar sus descriptores. Posiblemente la más empleada sea SIFT, propuesta en 2004 por David Lowe [21] y que todavía se emplea en gran número de aplicaciones [10,11] debido a la gran calidad de los resultados obtenidos. Este algoritmo emplea una diferencia de gaussianas (*DoG*), que consiste en aplicar dos filtros gaussianos a la misma imagen, uno con escala (σ) y el otro con menor escala.

A continuación, se restan los resultados para obtener la "diferencia de gaussianas". Esta operación se realiza a lo largo de todo un espacio de escalas, reduciendo el tamaño y resolución de las imágenes. Al final se obtiene una respuesta muy alta en los cambios de intensidad, tomando como puntos de interés los máximos y mínimos de la *DoG*, y descartando aquellos puntos que pertenezcan a un borde de la imagen (difíciles de identificar en el *matching*). De esta forma, al haber computado los puntos de interés a distintas escalas, se consigue que sean invariantes a dicha transformación. Además, para hacerlos invariantes a la rotación, se asigna una rotación de referencia o canónica a cada punto de interés en función del valor de los píxeles de su entorno. Por último, cabe comentar que, durante el *matching*, la técnica introduce el "*ratio test*", que consiste en rechazar aquellas correspondencias en las que la diferencia entre los descriptores de la mejor correspondencia y la segunda mejor es pequeña (existe ambigüedad).

A pesar de existir una versión de SIFT acelerada para GPUs con un rendimiento notable [22], han surgido otros detectores de puntos de interés y descriptores que están orientados a sistemas que funcionan en tiempo real. Esto se debe a que multitud de los dispositivos en los que se desea emplear dicha técnica no están equipados con una GPU (drones, sistemas embebidos…). Como ejemplo de técnicas alternativas se puede citar SURF (detector y descriptor), FAST (detector) y BRIEF (descriptor). El que mejor rendimiento ha demostrado es **ORB** [23] (el comportamiento en tiempo real de ORB en implementaciones sin GPU es superior a SIFT sobre GPUs), obteniendo resultados de calidad equiparable a SIFT. ORB (*Oriented Fast and Rotated Brief*) es el detector de puntos de interés y descriptor empleado en este trabajo, y su funcionamiento se detalla a continuación.

3.2.1 ORB

Se trata de un detector de puntos de interés y descriptor binario[5] invariante a la rotación y resistente al ruido. Se basa en el detector de puntos de interés FAST y el descriptor BRIEF, añadiendo una orientación canónica (un ángulo que no depende de la orientación con la que aparece un punto de interés en la imagen, y es igual para los mismos puntos de interés) a los puntos de interés obtenidos. De esta forma, a la hora de obtener el descriptor para un punto de interés a partir de los valores de su entorno, primero se orienta con respecto a la orientación

[5] Parámetros que identifican a un punto de interés con un vector de un determinado número de bits obtenidos a partir de comparaciones de intensidades de píxeles de su entorno. Se caracterizan por ser obtenidos y comparados más rápido que los descriptores tradicionales.

canónica. Así, el descriptor asociado a un punto de interés no dependerá de la orientación con la que aparece en la imagen.

Oriented FAST

FAST (*Features from Accelerated Segment Test*) se trata de un método para obtener puntos de interés introducido en [24]. Para determinar si un pixel *p* es un punto de interés, se procede de la siguiente manera:

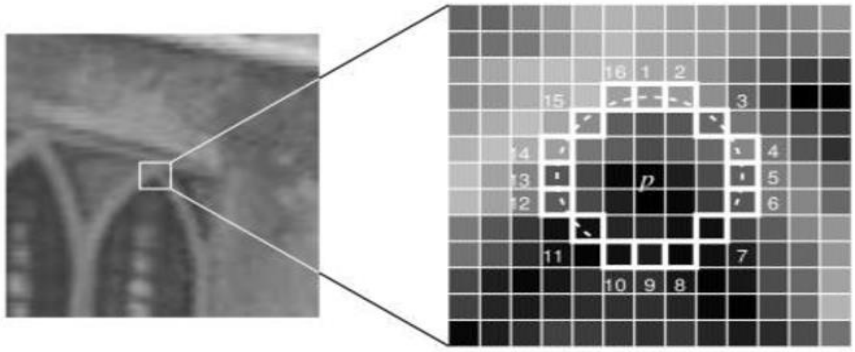

Figura 3.3: muestra de píxeles de FAST [24]

Se toma una muestra de 16 píxeles alrededor de *p* tal y como se ilustra en la figura anterior. Se define un umbral de intensidad *t*, y se comprueba qué píxeles de la muestra tienen una intensidad mayor que I(*p*) + *t* o menor que I(*p*) – *t*. En caso afirmativo, se dirá que son píxeles válidos de la muestra. Se considerará *p* un punto de interés si 9 o más píxeles adyacentes (por ejemplo, desde los píxeles 3 a 11) de la muestra tomada son válidos. En la técnica original se propone un *test* para rechazar un gran número de posibles puntos de interés de forma rápida. El test consiste en examinar los píxeles en las posiciones 1 y 9 de la muestra: si son píxeles válidos, se miden los píxeles en las posiciones 5 y 13. En caso de que al menos de 3 de los 4 píxeles medidos sean válidos, es posible que haya un grupo de 9 píxeles adyacentes válidos en la muestra, y se examina el resto de píxeles. En caso contrario, se descarta *p* como posible punto de interés.

El *test* realizado no es óptimo, al descartar el conocimiento obtenido sobre los 4 puntos analizados si se ha de comprobar el resto de la muestra. Además, con el orden y posición en el que se realiza el *test* se asume de manera implícita la apariencia del punto de interés (se espera que las posiciones 1, 9, 3, y 5 son las que más información proporcionan sobre si *p* es un punto de interés, cuando no siempre tiene por qué ser así), de forma que en algunos casos el *test* sea más rápido que en otros. Otro problema del detector consiste en que al comprobar los píxeles adyacentes a un punto de interés (por ejemplo, el píxel que está sobre *p* en la figura 3.3),

probablemente el algoritmo los detecte también como puntos de interés, al estar rodeados por una muestra muy similar, surgiendo una gran cantidad de puntos de interés concentrados en las mismas partes de la imagen. Para mejorar el algoritmo en [25] se introduce una aproximación basada en *machine learning* de FAST. Para ello, se genera un árbol de decisión empleando el algoritmo ID3, basado en la aplicación de FAST en unas imágenes de entrenamiento. Dicho entrenamiento obtiene los puntos de la muestra que proporcionan más información sobre si *p* es un punto de interés o no. Para aplicar el algoritmo una vez entrenado, se procede a realizar el *test* escogiendo la posición de los 4 puntos a comprobar (previamente, 1, 9, 3 y 5) como aquellos que con el entrenamiento han resultado proporcionar más información. Además, se aplica la técnica *non-maximal supression* para evitar obtener demasiados puntos de interés adyacentes. Dicha técnica consiste en otorgar una puntuación a cada punto de interés *p* como la suma de diferencias absolutas entre la intensidad de *p* y los píxeles de la muestra asociada. De esta forma, si hay dos puntos de interés adyacentes, se descarta el que tenga una puntuación inferior.

ORB emplea la técnica FAST original, al carecer de un entrenamiento previo para funcionar. Sin embargo, suple sus carencias aportando tres elementos adicionales:

- **Una puntuación basada en el filtro detector de esquinas de Harris:** permite ordenar los puntos de interés en función de su respuesta al filtro, para obtener los *N* mejores puntos de interés, actuando de forma similar a la técnica *non-maximal supression*, y además eliminando puntos de interés localizados en bordes (detectados por FAST, pero con baja puntuación Harris). Para ello, se ajusta el umbral FAST (*t*) de manera automática para que proporcione más de *N* puntos de interés, y se guardan como válidos únicamente los *N* con mayor puntuación.

- **Un esquema de escala piramidal**: al igual que en SIFT, ORB detecta puntos de interés a distintas escalas, y proporciona un peso a la puntuación Harris de cada punto de interés basado en la escala en la que se haya detectado.

- **Una orientación basada en el *centroide* de intensidad:** la principal aportación, permite obtener puntos de interés invariantes a la rotación. Para ello, mide el *centroide* de intensidad para cada punto de interés, que proporciona un vector que une un punto de interés con el *centroide*, que consiste en un equivalente al centro de masa para los valores de intensidad de la imagen. Para obtenerlo, se calcula el *momento* de un parche (en este trabajo, una ventana de 31x31 píxeles) alrededor de la imagen, definido como:

$$m_{pq} = \sum_{x,y} x^p \cdot y^q \cdot I(x,y)$$

Siendo el centroide:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)$$

Y el ángulo del vector que une el punto de interés con el centroide:

$$\theta = \text{atan2}(m_{01}, m_{10})$$

Obteniéndose una orientación canónica para cada punto de interés, que no depende de la orientación con la que aparezca en la imagen.

De esta forma, con unas pequeñas mejoras a FAST se obtiene un detector de puntos de interés notablemente más estable y todavía muy rápido.

Rotation-Aware BRIEF

El descriptor BRIEF se trata de una descripción del entorno de **p** (parche de 31x31 píxeles) obtenido a partir de una imagen suavizada empleando *tests* binarios de intensidad, definidos como:

$$\tau(p; x, y) \coloneqq \begin{cases} 1 : p(x) < p(y) \\ 0 : p(x) \geq p(y) \end{cases}$$

Con p(x) la intensidad de **p** en el punto **x**. De esta forma, el descriptor para cada punto de interés es un vector de *n* = 256 *tests* binarios:

$$f_n(p) \coloneqq \sum_{1 \leq i \leq n} 2^{i-1} \cdot \tau(p; x, y)$$

Donde la elección de los pares x e y donde se realizan los *tests* binarios constituyen el ***patrón de test***. En [26] se comprueba empíricamente que el mejor patrón consiste en distribuir los pares aleatoriamente de acuerdo a una distribución gaussiana alrededor del centro del parche (más puntos comprobados en el centro del parche). Así, el descriptor asociado a un punto de interés se construye con 256 comparaciones de intensidad de pares de píxeles aleatorios en una ventana de 31x31 píxeles alrededor del punto de interés.

Para incorporar la rotación obtenida en la detección de puntos de interés, se propone "rotar" el patrón de *test* multiplicando la matriz que conforma los pares de *test* por una matriz de rotación correspondiente al ángulo θ. De esta forma, se construye una tabla de búsqueda (*lookup table*) con el patrón original BRIEF a emplear y los valores de los pares de *test* para ángulos discretizados de 12º. De esta forma, se obtienen descriptores iguales para el mismo punto de interés, aunque haya una rotación entre una imagen y la otra.

Una de las características que hacen a BRIEF un buen descriptor es que cada valor tiene una media próxima a 0.5 (casi el mismo número de 1s y 0s), y la máxima varianza posible para dicha media (0.25), provocando que el descriptor sea más discriminativo, al responder de forma más variada ante distintas entradas (es decir, los vectores binarios de puntos de interés distintos son más distintos entre sí). Sin embargo, al introducir la información de la rotación en el descriptor, la media se aleja de 0.5 y empeora la varianza. Para solucionarlo y, además, reducir la correlación entre descriptores, en [23] proponen un método de entrenamiento para obtener un buen patrón de test. Dicho entrenamiento consiste en realizar todos los posibles pares de test binarios para 300.000 puntos de interés (teniendo en cuenta su rotación) obtenidos del *dataset* PASCAL [27]. Para ello se realizan todos los *tests* binarios para cada parche correspondiente a un punto de interés, y se ordenan los *tests* en función de su distancia a una media de 0.5 (al realizarlo en todos los parches, en la mitad el resultado del *test* binario ha sido 1, y en la otra mitad 0), y se van añadiendo en orden al patrón final siempre y cuando guarden poca correlación con el resto (su resultado en cada parche no sea similar a los ya incluidos), hasta tener un patrón de 256 *tests*. Dicho patrón obtiene características incluso mejores que las del descriptor BRIEF original a pesar de incorporar la información de rotación de los puntos de interés, encontrándose integrado en la implementación utilizada en este trabajo.

3.2.2 Detección de correspondencias

Una vez se ha obtenido los puntos de interés se procede a buscar correspondencias entre imágenes. Se realiza siempre entre pares de imágenes. Al emplearse descriptores binarios para almacenar la información sobre los puntos de interés, se tiene una métrica muy sencilla para calcular la similitud que hay entre dos puntos de interés: la *distancia de Hamming* (número de bits distintos entre un descriptor y otro). De esta forma, únicamente con realizar la suma de la operación XOR entre ambos descriptores, se obtiene una métrica eficaz para calcular cómo se parece un punto de interés a otro. Por lo tanto, en una aproximación sencilla a la detección de correspondencias, basta con comparar un descriptor en una imagen con todos los descriptores en la otra y tomar como correspondencia el que tenga menor *distancia de Hamming*. Para filtrar

falsas correspondencias, además, se lleva a cabo el *ratio test* introducido en SIFT. En el capítulo 4 se introducirán las otras dos técnicas empleadas para filtrar las correspondencias incorrectas, así como otras aproximaciones para obtener las correspondencias, siendo la aquí introducida la que se emplea en la implementación.

3.3 Reconstrucción inicial

3.3.1 Movimiento de la cámara

El primer paso a la hora de implementar un sistema SfM es obtener el movimiento entre las cámaras, es decir, la rotación y traslación que reflejan el movimiento desde la primera cámara a la segunda, así como su calibración (en este caso ya conocida), reflejado en la matriz de proyección. Como ya se ha explicado, no se obtiene una reconstrucción de la escena con coordenadas a nivel absoluto (no es posible conocer las "unidades" en las que están expresadas dichas coordenadas, ni su posición absoluta en el "mundo" tridimensional), por lo que se trabaja a escala arbitraria. Por lo tanto, para facilitar el problema, la primera cámara se sitúa en el origen de coordenadas.

El algoritmo parte de las correspondencias de puntos obtenidas entre las dos imágenes durante el *matching*. A partir de estas correspondencias, se va a obtener la posición y orientación relativa de las cámaras empleando la ***matriz fundamental*** (F) y la ***matriz esencial*** (**E**). La conversión de una a otra es muy sencilla, permitiendo la estimación de la primera descomponer la segunda para obtener la posición y orientación relativa de las cámaras.

Matriz fundamental y geometría epipolar

La geometría *epipolar* recoge la geometría *proyectiva* entre dos vistas, y es encapsulada por la matriz fundamental. Se basa en que, dado un punto en el espacio del que se conoce su posición en una vista, su posición en la otra vista está limitada a una *línea epipolar*, tal y como se refleja en la figura de la página siguiente.

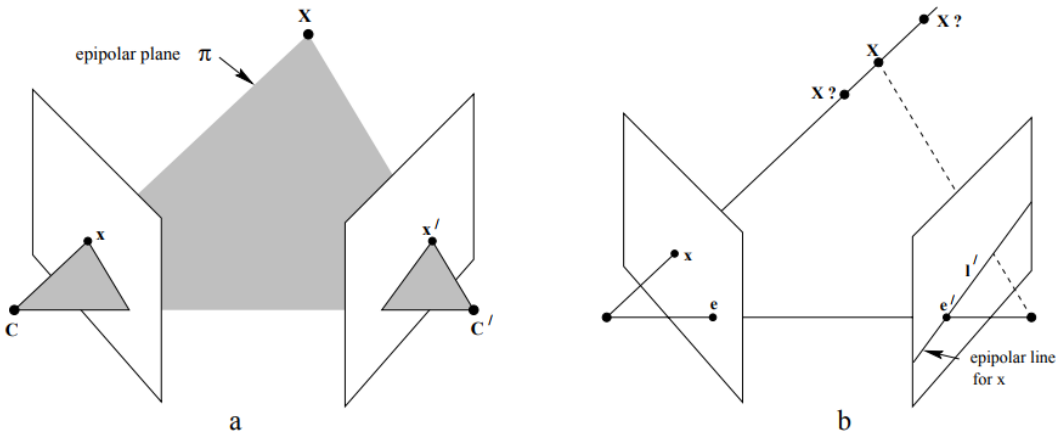

Figura 3.4: Geometría epipolar: *(a) Plano epipolar. (b) Línea epipolar*

Para cada punto que se desea proyectar: el ***plano epipolar*** es el plano que contiene los centros de las cámaras, el punto en el espacio que se desea proyectar y las proyecciones de dicho punto en ambas imágenes. Los ***puntos epipolares*** (***e*** y ***e'***) son los puntos de las imágenes en los que son cortadas por la línea que une los centros de las cámaras. La ***línea epipolar*** (***l'***) es la proyección en la segunda imagen de la línea que proyecta el punto tridimensional en la primera imagen. Por lo tanto, el punto del espacio que se está proyectando, se proyectará en la segunda imagen en algún punto de la línea *epipolar*. Se puede ver esta restricción como un mapeo, de forma que para cada punto **x** en una imagen, existe una línea *epipolar* **l'** en la otra vista en la que se encontrará **x'**. Este mapeo es el que realiza la matriz fundamental, una matriz 3x3 de rango 2 que mapea un punto de un espacio bidimensional (la primera imagen) a uno unidimensional (la línea *epipolar* de la segunda imagen). si **x** y **x'** son la proyección del punto **X** en ambas imágenes en coordenadas homogéneas, al estar **x'** en **l'** se cumple:

$$\mathbf{l}' = \mathbf{F} \cdot \mathbf{x} \rightarrow \boxed{\mathbf{x}'^{\mathbf{T}} \cdot \mathbf{F} \cdot \mathbf{x} = 0} = \mathbf{x}'^{\mathbf{T}} \cdot \mathbf{l}'$$

La relación en el recuadro es fundamental, ya que es la que nos permite computar la matriz fundamental a partir de correspondencias entre puntos de ambas imágenes. Además, hay que destacar que, en dicha relación las coordenadas de los puntos de la imagen están expresados en píxeles.

Estimación de la matriz fundamental

Del apartado anterior, se concluye que para cada correspondencia entre imágenes se verifica:

$$[\mathrm{x}' \quad \mathrm{y}' \quad 1] \cdot \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ 1 \end{bmatrix} = 0$$

Para *n* correspondencias, dicha ecuación se puede reformular como:

$$\begin{bmatrix} x_1' \cdot x_1 & x_1' \cdot y_1 & x_1' & y_1' \cdot x_1 & y_1' \cdot y_1' & y_1' & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n' \cdot x_n & x_n' \cdot y_n & x_n' & y_n' \cdot x_n & y_n' \cdot y_n' & y_n' & x_n & y_n & 1 \end{bmatrix} \cdot \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0 \rightarrow \mathbf{A} \cdot \mathbf{f} = 0$$

Como se puede observar, se trata de un sistema de ecuaciones homogéneo. Además, **f** solo puede ser determinado a una escala arbitraria ya que la solución es invariante si se multiplica la matriz **f** por una constante (factor de escala). Para que exista una solución distinta de la trivial (**f** = 0), la matriz **A** tiene que tener un rango de 8 o menor, y se obtiene una solución única (obtenida con métodos lineales) cuando **A** tiene rango 8 (asumiendo que un valor de **f** vale 1, es decir, escalando **f** por 1). Si el rango de la matriz es 9 (caso probable debido al ruido en las coordenadas de los puntos), es posible obtener una solución aproximada (mínimo error cuadrático) correspondiente al vector singular asociado al autovalor más pequeño de A [1], es decir, la última columna de V en la descomposición de valores singulares (SVD) de A.

En este trabajo se estima la matriz fundamental empleando el método *RANSAC*, ya que es uno de los algoritmos más robustos para este propósito. Como entrada se emplean los puntos de interés obtenidos en ambas imágenes (para los que se ha encontrado correspondencia) En dicho algoritmo se va resolviendo el sistema de ecuaciones para correspondencias de 8 puntos escogidos aleatoriamente, definiendo la solución a cada sistema una matriz **F**. Para cada punto de la primera imagen, empleando la **F** obtenida se calcula la línea *epipolar* correspondiente en la segunda imagen, y se obtiene la distancia *d* del punto en la segunda imagen a la línea *epipolar* correspondiente. Los *inliers* (puntos que encajan en el modelo obtenido) serán aquellos puntos con *d* menor que un umbral. Se define como matriz fundamental **F** la matriz obtenida con mayor número de *inliers* asociados. Se presenta un esquema del algoritmo en la página siguiente.

Estimación de la matriz fundamental
Entradas: puntos de interés de las dos imágenes y correspondencias obtenidas
Salidas: matriz fundamental estimada

1. N inicial = número total de puntos empleados para la estimación

2. Estimación robusta usando RANSAC: mientras *sample_count* sea menor que N

2.1 Seleccionar un conjunto aleatorio de 8 correspondencias y obtener la matriz fundamental F.
2.2 Calcular para cada correspondencia la distancia *d* de la línea epipolar obtenida con F al punto correspondiente.
2.3 Calcular el número de *inliers*. Se define como *inlier* aquellos puntos con $d < 0.5$ píxeles.
2.4 Se define $N = \log(1-0.99) / \log(1-(1-\varepsilon)^7)$, siendo ε la proporción de *outliers*.
2.5 Incrementar *sample_count* en 1.

3. Se escoge la matriz F con el mayor número de *inliers*.

Matriz esencial

Una vez se ha obtenido la matriz fundamental, para obtener la matriz de proyección de la segunda cámara se ha de partir de la *matriz esencial* (**E**). Dicha matriz es una especialización de la matriz fundamental, en la que las coordenadas de las imágenes ya no están expresadas en píxeles, sino que se expresan en coordenadas normalizadas[6]. La ecuación que define la matriz esencial (**E**) es:

$$\mathbf{u'^T} \cdot \mathbf{E} \cdot \mathbf{u} = 0 \rightarrow \mathbf{u'^T} \cdot \mathbf{K'^{-T}} \cdot \mathbf{E} \cdot \mathrm{K}^{-1} \cdot \mathbf{u} = 0 \rightarrow \mathbf{E} = \mathbf{K'^T} \cdot \mathbf{F} \cdot \mathbf{K}$$

Su principal característica consiste en que en una matriz esencial dos de sus valores singulares han de ser iguales entre sí y el tercero 0, por lo que su determinante es igual a 0.

Extracción de la posición de las cámaras

A partir de la matriz esencial es posible extraer las matrices de proyección de las cámaras mediante un procedimiento que proporciona cuatro posibles soluciones a una escala arbitraria.

[6] Dada una matriz de proyección P = K[R|t], el punto del espacio X en coordenadas normalizadas es $x_n = [R|t]X$, de forma que $x_c = K^{-1}x$. Es decir, las coordinadas que tendría el punto si la matriz intrínseca de la cámara fuera la matriz identidad.

Reconstrucción inicial

Como ya se ha mencionado, se va a asumir que la primera cámara está situada en el origen de coordenadas. Es en este paso cuando se introduce la escala desconocida en el vector de traslación que relaciona ambas cámaras y, por lo tanto, de la reconstrucción.

Es posible descomponer la matriz esencial en dos matrices: una primera matriz *antisimétrica* que corresponde a la traslación, y otra matriz *ortonormal* correspondiente a la rotación, siempre y cuando se cumpla la condición mencionada con respecto a sus *autovalores*.

$$\mathbf{E} = [\mathbf{t}]_{\mathbf{x}}{}^{7} \cdot \mathbf{R} = \mathbf{S} \cdot \mathbf{R}$$

Definiendo las matrices **W** y **Z** (*ortogonal* y *antisimétrica* respectivamente) y la descomposición en valores singulare (SVD) de **E**, en [28] se demuestra que existen únicamente dos posibles factorizaciones de E como S·R:

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathrm{SVD}(\mathbf{E}) = \mathbf{U} \cdot \mathrm{diag}(1,1,0) \cdot \mathbf{V}^{\mathrm{T}}$$

$$\mathbf{S} = \mathbf{U} \cdot \mathbf{Z} \cdot \mathbf{U}^{\mathbf{T}}, \qquad \mathbf{R} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\mathbf{T}} \text{ ó } \mathbf{U} \cdot \mathbf{W}^{\mathbf{T}} \cdot \mathbf{V}^{\mathbf{T}}$$

La factorización determina el vector de traslación de la matriz de proyección (**t**) a una escala arbitraria (los *autovalores* de **E** se suponen la unidad, es decir, como si se normalizasen por su valor, que será desconocido) a partir de $\mathbf{S} = [\mathbf{t}]_{\mathbf{x}}$. De esta relación se deduce el valor de **t** a falta de su signo:

$$\mathbf{t} = \mathbf{U} \cdot (0,0,1)^{\mathrm{T}} = \mathrm{u}_3$$

Por lo tanto, dada una matriz esencial **E** y la matriz de proyección de la primera cámara situada en el origen de coordenadas, existen 4 posibles rotaciones y traslaciones para la segunda cámara:

$$\mathbf{R}' = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\mathbf{T}}, \mathbf{t}' = +\mathrm{u}_3 \quad \text{ó} \quad \mathbf{R}' = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\mathbf{T}}, \mathbf{t}' = -\mathrm{u}_3 \quad \text{ó} \quad \mathbf{R}' = \mathbf{U} \cdot \mathbf{W}^{\mathbf{T}} \cdot \mathbf{V}^{\mathbf{T}}, \mathbf{t}' = +\mathrm{u}_3$$

$$\text{ó } \mathbf{R}' = \mathbf{U} \cdot \mathbf{W}^{\mathbf{T}} \cdot \mathbf{V}^{\mathbf{T}}, \mathbf{t}' = -\mathrm{u}_3$$

---

[7] Si $t = [t_x, t_y, t_z]$, $[t]_z = \begin{bmatrix} 0 & -t_x & t_y \\ t_x & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}$

Estas soluciones tienen una interpretación simétrica sencilla si se representan en el espacio. Al cambiar el vector de traslación de signo la posición de las cámaras se invierte (izquierda y derecha en la siguiente figura). Entre las dos posibles matrices de rotación existe una rotación de 180º con respecto al eje que une las dos cámaras (arriba y abajo en la siguiente figura).

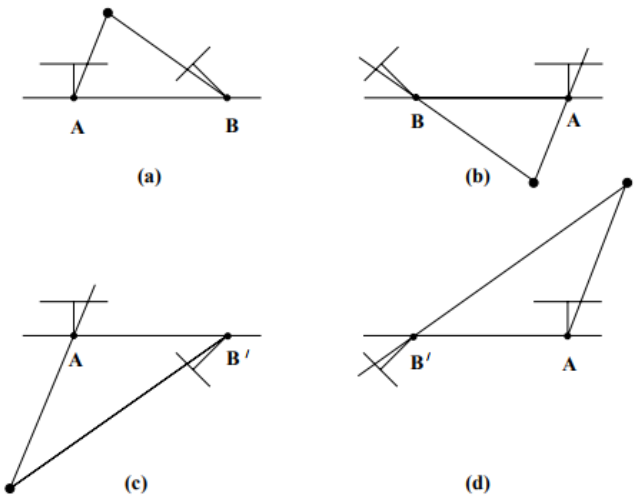

Figura 3.5: las soluciones para la reconstrucción a partir de E [1]

Para terminar, se ha de comprobar en qué solución los puntos reconstruidos mediante la triangulación (explicada a continuación) se encuentran delante de ambas cámaras. Para ello, se proyectan empleando las matrices de proyección obtenidas, y se toman aquellos con un valor de z (tercera coordenada del punto en la imagen resultante) positivo.

3.3.2 Triangulación

Una vez obtenidas las matrices de proyección de ambas cámaras (y, en consecuencia, su posición), es posible triangular los puntos obtenidos como correspondencias. Idealmente, los puntos 3D deberían estar en la intersección de las rectas que pasan por el centro de la cámara y la proyección del punto correspondiente. Es decir, se cumple $x = P \cdot X$, $x' = P' \cdot X$. Sin embargo, en imágenes con algo de ruido (por ejemplo, debido a su compresión), es bastante probable que dichas rectas no se corten, tal y como se muestra en la figura:

Reconstrucción inicial

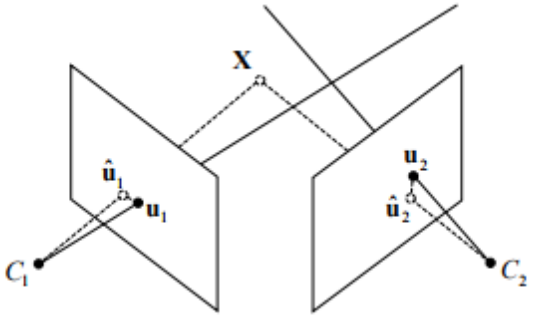

***Figura 3.6: triangulación y error de reproyección**. Las proyecciones en el espacio del mismo punto en ambas imágenes no coinciden. Se busca la solución con el menor error de reproyección* [1]

Por lo tanto, se trata de obtener el punto **X** del espacio tal que, cuando se proyecta en ambas imágenes, la distancia entre su proyección y los puntos de la correspondencia inicial es mínima (mínimo error de reproyección). Existen diversos métodos para resolver este problema. En [29], se lleva a cabo una revisión de los distintos métodos, estableciendo como el más rápido y óptimo el método iterativo lineal de mínimos cuadrados (siempre y cuando se conozcan los parámetros intrínsecos de la cámara, como es normalmente el caso). Este método comienza con una primera estimación lineal, para, posteriormente ir adaptando los pesos de las ecuaciones lineales para que se correspondan con los errores de reproyección.

Estimación inicial

La primera estimación consiste realiza una primera triangulación lineal que proporciona una solución óptima de mínimos cuadrados. Se supone:

$$\mathbf{u} = \mathbf{P} \cdot \mathbf{X}, \qquad \mathbf{u} = w \cdot (u, v, 1)^T$$

Donde (u,v) son las coordenadas normalizadas del punto observado en la imagen, y **u** sus coordenadas homogéneas, siendo w un factor de escala desconocido. Llamando $\mathbf{p}_i^T$ a la fila i-ésima de P:

$$w \cdot u = \mathbf{p}_1^T \cdot \mathbf{X}, \qquad w \cdot v = \mathbf{p}_2^T \cdot \mathbf{X}, \qquad w = \mathbf{p}_3^T \cdot \mathbf{X}$$

Eliminando w con la tercera ecuación:

$$u \cdot \mathbf{p}_3^T \cdot \mathbf{X} = \mathbf{p}_1^T \cdot \mathbf{X}, \qquad v \cdot \mathbf{p}_3^T \cdot \mathbf{X} = \mathbf{p}_2^T \cdot \mathbf{X}$$

Empleando dos imágenes, se obtiene un total de 4 ecuaciones lineales en función de las coordenadas de **X**. En forma matricial, se escribe $\mathbf{A} \cdot \mathbf{X} = 0$, definiendo un sistema de ecuaciones homogéneo:

$$\begin{bmatrix} u \cdot \mathbf{p_3^T} - \mathbf{p_1^T} \\ v \cdot \mathbf{p_3^T} - \mathbf{p_2^T} \\ u' \cdot \mathbf{p_3^T} - \mathbf{p_1^T} \\ v' \cdot \mathbf{p_3^T} - \mathbf{p_2^T} \end{bmatrix} \cdot \mathbf{X} = 0$$

Imponiendo la condición de que un elemento de **X** sea 1 (coordenadas homogéneas), este sistema se convierte en un sistema del tipo **A**·**X** = **B**, siendo **A** una matriz 4x3, **X** un vector 3x1 y **B** una matriz 4x1:

$$\begin{bmatrix} u \cdot p_{20} - p_{00} & u \cdot p_{21} - p_{01} & u \cdot p_{22} - p_{02} \\ v \cdot p_{20} - p_{10} & v \cdot p_{21} - p_{11} & v \cdot p_{22} - p_{12} \\ u' \cdot p'_{20} - p'_{00} & u' \cdot p'_{21} - p'_{01} & u' \cdot p'_{22} - p'_{02} \\ v' \cdot p'_{20} - p'_{10} & v' \cdot p'_{21} - p'_{11} & v' \cdot p'_{22} - p'_{12} \end{bmatrix} \cdot \mathbf{X} = -\begin{bmatrix} u \cdot p_{23} - p_{03} \\ v \cdot p_{23} - p_{13} \\ u' \cdot p'_{23} - p'_{03} \\ v' \cdot p'_{23} - p'_{13} \end{bmatrix}$$

Resolviendo este sistema de ecuaciones mediante la descomposición SVD se obtiene una solución aproximada de mínimo error cuadrático para **X**.

Método iterativo lineal

La estimación inicial realizada en el apartado anterior proporciona una solución aceptable, pero no muy precisa. Esto se debe a que se minimiza ||**A**·**X**||, ecuación que carece de significado geométrico, y no una función de coste adecuada. Además, multiplicar cualquiera de las ecuaciones por alguna escala modificará la solución. Esta propiedad se aprovecha para aplicar distintos pesos a las ecuaciones, de forma que el valor minimizado coincida con el error de reproyección.

Partiendo de la ecuación:

$$u \cdot \mathbf{p_3^T} \cdot \mathbf{X} = \mathbf{p_1^T} \cdot \mathbf{X}$$

El punto **X** obtenido no satisface esta ecuación de forma exacta, sino que habrá un error ε:

$$\varepsilon = u \cdot \mathbf{p_3^T} \cdot \mathbf{X} - \mathbf{p_1^T} \cdot \mathbf{X}$$

Sin embargo, el valor que se desea minimizar es la diferencia entre la posición del punto en la imagen y la proyección de **X**, que viene dado por:

$$\varepsilon' = \varepsilon / \mathbf{p_3^T} \cdot \mathbf{X} = u - \mathbf{p_1^T} \cdot \mathbf{X} / \mathbf{p_3^T} \cdot \mathbf{X}$$

Por lo tanto, si se multiplica a la ecuación original por un factor 1/w, siendo w = $\mathbf{p_3^T \cdot X}$, el error cometido coincide con el valor que se desea minimizar. Para la segunda imagen, se ha de

aplicar el factor 1/w', con w' = $\mathbf{p'}_3^T \cdot \mathbf{X}$. Los pesos dependen del valor de **X**, que no conoceremos hasta que se resuelva el sistema de ecuaciones. Por lo tanto, se procede de manera iterativa para adaptar los pesos, comenzando por unos valores de w = w' = 1, y posteriormente multiplicando por los factores w y w' calculados con la **X** obtenida en la iteración anterior. En unas pocas iteraciones (en [1] se proponen 10) el método converge, habiéndose obtenido la **X** con menor error de reproyección.

Una vez determinados los puntos se completa la reconstrucción inicial. Como resultado se han obtenido las posiciones relativas de las cámaras y de los puntos de interés comunes a ambas imágenes en el espacio a una escala arbitraria. Estos puntos serán empleados como base para incorporar el resto de vistas y puntos 3D del resto de imágenes.

3.4 Perspective-*n*-Point

Una vez se ha obtenido una reconstrucción inicial hay que buscar un mecanismo para incorporar nuevas vistas a ella. Una primera idea podría ser el emplear el mismo método que se ha utilizado para obtener la reconstrucción inicial, pero utilizado únicamente con la segunda imagen empleada para la reconstrucción y una nueva vista (de forma que se *solapan* vistas en sucesivas reconstrucciones). Sin embargo, como se ha detallado en el apartado anterior, la aproximación inicial asume que una de las cámaras está en el origen de coordenadas, y el algoritmo obtiene el cambio de posición relativo entre ambas cámaras, con una traslación a una escala arbitraria. Por lo tanto, si se llevase a cabo una nueva reconstrucción con la misma técnica que la inicial, no sería posible incorporarla a la reconstrucción anterior de manera eficaz, ya que no se conoce en qué escala ha sido obtenida la nueva reconstrucción con respecto a la anterior.

No obstante, es posible resolver este problema si se tiene en cuenta que se conocen las posiciones en el espacio de los puntos obtenidos en la reconstrucción inicial, y que es posible identificar dichos puntos con sus proyecciones en la nueva imagen. Dicha información se puede emplear para determinar la magnitud de la traslación de la nueva cámara haciendo coincidir la proyección de un punto 3D conocido con su posición en la imagen. Sin embargo, se trata de un método poco preciso y muy sensible a errores. La solución más habitual consiste en obtener la nueva posición de la cámara calibrada a partir de *n* puntos de los que se conoce su posición en el espacio (en las coordenadas de la escena ya reconstruida) y su proyección en la nueva imagen: es lo que se conoce como *Perspective-n-Point*. Esta técnica se puede formular como:

$$s \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Donde (x,y) son las coordenadas en la imagen, (x,y,z) las coordenadas del punto 3D en el sistema de coordenadas de la escena reconstruida, K la matriz de parámetros intrínsecos y *s* el factor de escala desconocido para la transformación.

Este es un problema casi lineal para la matriz de parámetros extrínsecos, a excepción de la existencia del factor de escala *s*. Para resolver este tipo de problemas en [1] se propone emplear la transformación *DLT* (*Direct linear transform*), que proporciona una solución aproximada del problema, seguida por una optimización empleando el algoritmo *Levenberg-Marquardt*. Esta aproximación también permite encontrar una solución en el caso de no conocer K. Sin embargo, es posible encontrar otras soluciones más óptimas para este problema.

Por ejemplo, una alternativa muy empleada es P3P [30] (*perspective-three-point*) en combinación con RANSAC que proporciona 4 soluciones con únicamente tres correspondencias (tres puntos de interés identificados en ambas imágenes). Dichas correspondencias posteriormente son comprobadas de manera similar a las soluciones aportadas por la descomposición de la matriz esencial. En este trabajo se ha escogido emplear EP*n*P, *Efficient perspective-n-point* (desarrollado en [31]), y que presenta una solución para el problema general en el que *n* es mayor que 3. Se trata de una solución no-iterativa, válida para un *n* grande, y que proporciona una solución igual de precisa que otras alternativas, siendo notablemente más rápido, de acuerdo con las pruebas llevadas a cabo en [31].

EP*n*P se basa en expresar cada uno de los *n* puntos como una suma ponderada (con unos pesos conocidos) de 4 puntos de control virtuales (c_j), que se convierten en las incógnitas del problema:

$$p_i^w = \sum_{j=1}^{4} \propto_{ij} \cdot c_j^w, \quad p_i^c = \sum_{j=1}^{4} \propto_{ij} \cdot c_j^c, \quad \sum_{j=1}^{4} \propto_{ij} = 1$$

El superíndice *w* indica que se refiere a los puntos expresados en las coordenadas de la escena ya reconstruida, y el superíndice *c* son los expresados en coordenadas de la cámara de la nueva imagen. La ecuación que define el problema P*n*P se reformula como:

$$s_i \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_c \\ 0 & f_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \cdot \sum_{j=1}^{4} \propto_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix}$$

Donde los parámetros que no se conocen del sistema son las 12 coordenadas de los puntos de control (x_j^c, y_j^c, z_j^c) y los *n* parámetros s_i. Teniendo en cuenta la ecuación definida por la última fila de K, se obtienen dos ecuaciones para cada uno de los *n* puntos:

$$\sum_{j=1}^{4} \propto_{ij} \cdot f_x \cdot x_j^c + \propto_{ij} \cdot (x_c - x_i) \cdot z_j^c = 0$$

$$\sum_{j=1}^{4} \propto_{ij} \cdot f_y \cdot y_j^c + \propto_{ij} \cdot (y_c - y_i) \cdot z_j^c = 0$$

Dichas ecuaciones ya no dependen de los parámetros de escalado *s* y se pueden resolver con métodos lineales. Una vez se han obtenido los valores de los puntos de control, se calculan los valores de los puntos de referencia expresados en las coordenadas de la nueva cámara, por lo que se pueden obtener los valores de **R** y **t** que minimizan el error resolviendo por métodos lineales el siguiente sistema:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Con este método, es posible ir obteniendo las posiciones de las nuevas cámaras expresadas en el mismo sistema de coordenadas que el resto de la escena previamente reconstruida. Con dichas posiciones, se procede a triangular las correspondencias encontradas entre la nueva imagen y las previamente reconstruidas.

3.5 Bundle Adjustment

A lo largo de este capítulo se han presentado distintas técnicas para obtener las posiciones de las cámaras y los puntos en el espacio. Muchas de ellas no ofrecen soluciones exactas, sino que se emplean soluciones con el mínimo error posible. Sin embargo, los errores cometidos en los distintos pasos de la reconstrucción se van acumulando al añadir vistas, provocando que el error final pueda ser bastante grande.

Por lo tanto, se hace necesario introducir un paso de ajuste o refinamiento de la reconstrucción, en el que se emplean todos los datos obtenidos hasta el momento (posiciones de las cámaras y puntos en el espacio, así como posiciones de los puntos dentro de las imágenes), y se trata de ajustarlos a un modelo óptimo. Para ello se lleva a cabo una optimización que minimiza el error

de reproyección, que recoge los parámetros de entrada (puntos del espacio, X, matrices de proyección de las cámaras, P, y puntos en las imágenes, x). Con *m* puntos en el espacio obtenidos con *n* vistas, y siendo $w_{ij} = 1$ si el punto *j* aparece en la imagen *i*:

$$\varepsilon(\mathbf{P}, \mathbf{X}, \mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} \cdot \left\| \mathbf{x_{ij}} - \mathbf{P_i} \cdot \mathbf{X_j} \right\|$$

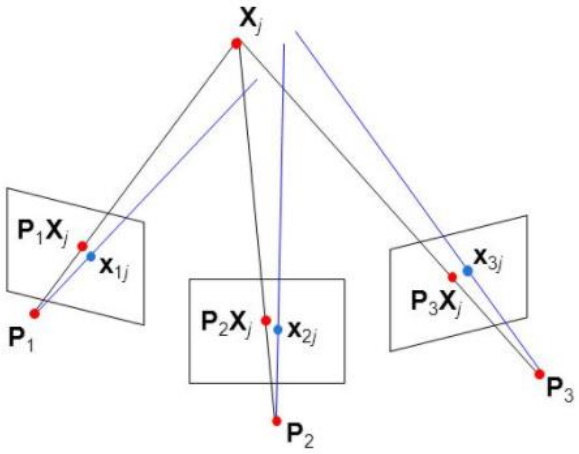

Figura 3.7: Error de reproyección. *Al proyectar un punto X conocido en el espacio con una matriz P, el error de reproyección se define como la distancia sobre la imagen del punto proyectado al punto original* [41]

El error de reproyección se modela como una suma de cuadrados de una función no lineal que depende de múltiples variables. Por lo tanto, se ha de resolver empleando un algoritmo de mínimos cuadrados no lineal. Existen diversas alternativas, siendo el algoritmo de Levenberg-Marquardt el que mejor funciona. Se trata de un algoritmo iterativo que encuentra el mínimo local de una función del tipo indicado. Dicha técnica funciona empleando una combinación de los algoritmos de descenso por gradiente y el método de Gauss-Newton: cuando el estado actual de la función se encuentra lejos de un mínimo el algoritmo funciona como el método de descenso por gradiente (lento, pero garantiza convergencia). Cuando se acerca a un mínimo local, se emplea el método de Gauss-Newton, permitiendo una convergencia rápida (principal inconveniente del método de descenso por gradiente cuando se acerca a un mínimo).

Llamando $\mathbf{p}$ al vector de parámetros, y $\mathbf{x_e} = f(\mathbf{p})$al vector de medidas estimado, se parte de unos parámetros iniciales $\mathbf{p_0}$ y unas medidas $\mathbf{x}$, y se busca obtener el vector $\mathbf{p}$ que minimiza $\varepsilon^T \cdot \varepsilon$, siendo $\varepsilon = \mathbf{x} - \mathbf{x_e}$ (error de reproyección). Para ello, *f* se aproxima de la siguiente manera:

$$f(\boldsymbol{p} + \delta_p) \approx f(\boldsymbol{p}) + \boldsymbol{J} \cdot \delta_p$$

Donde **J** es el Jacobiano de *f*, y δ_p es el del paso que minimiza la cantidad $\|\mathbf{x} - f(\mathbf{p} + \delta_p)\|$. De esta forma, se procede iterativamente hasta que el sistema converge en un valor **p** para el cual $\varepsilon^T \cdot \varepsilon$ es mínimo. En cada iteración del algoritmo, se calcula δ_p a partir de la siguiente ecuación:

$$\mathbf{N} \cdot \delta p = \mathbf{J}^T \cdot \boldsymbol{\varepsilon}, \qquad N = \mathbf{J}^T \cdot \mathbf{J} + \mu \mathbf{I}, \qquad \mu > 0$$

Analizando sus elementos, se tiene que $\mathbf{J}^T \cdot \varepsilon$ coincide con la dirección del descenso por gradiente. Además, se altera la diagonal de **N** en un proceso conocido como *damping*. Si el valor de δ_p obtenido reduce el error, se procede a la siguiente iteración reduciendo μ. En caso contrario, se aumenta el valor de μ y se obtiene un nuevo valor de δ_p, repitiendo el proceso hasta que se encuentra un valor que reduce el error, y pasando entonces a la siguiente iteración. Esta modificación a la diagonal de **N** provoca que, si μ se fija a un valor alto, la matriz se aproxima a una matriz diagonal y, en consecuencia, δ_p a la dirección del descenso por gradiente, reduciendo además su magnitud para asegurar que no se tomen pasos demasiado grandes. Por otra parte, si μ toma un valor bajo, el paso se aproxima al empleado en el método de Gauss-Newton (minimizador local de un modelo cuadrático). De esta forma, Levenberg-Marquardt implementa un método adaptativo que reúne los beneficios tanto de Gauss-Newton como del descenso del gradiente.

Para terminar, hay que resaltar que el *bundle adjustment* es probablemente el paso más importante del algoritmo, ya que permite reducir el error de manera significativa. Esto es básico para el correcto funcionamiento de SfM, al ser muy sensible a la acumulación de errores. Este ajuste se lleva a cabo tras la incorporación de una nueva vista para evitar la acumulación de errores. Además, el éxito de este ajuste depende en gran medida de una buena inicialización, por lo que es importante haber obtenido las posiciones iniciales de los puntos y cámaras con la mayor precisión posible.

Capítulo 4
Evaluación de la implementación

La implementación realizada del algoritmo SfM está orientada a establecer una base capaz de evaluar la técnica SfM empleando métodos óptimos para cada paso. La implementación puede ser modificada *a posteriori* para elaborar un sistema vSLAM capaz de operar en tiempo real. En este capítulo se va a detallar la implementación realizada, analizar su rendimiento paso por paso e identificar alternativas y posibles mejoras.

4.1 Estructura del algoritmo

Como se comenta en la introducción del capítulo 3, se ha implementado un SfM con estrategia secuencial, en la que no se conocen de antemano las imágenes o vistas a procesar, sino que se van incorporando *frame* por *frame* conforme se van capturando. Este factor será el que fije la estructura general del programa. En contexto, el sistema funciona como un ciclo continuo, en el que van llegando imágenes y se incorporan a la estructura previamente procesada. Dicho ciclo consta de dos partes principales: un primer bloque para realizar la inicialización, y otro para ir incorporando vistas sucesivamente.

De esta forma, el sistema realiza una reconstrucción inicial tal y como se ha comentado en el capítulo 2. En dicha reconstrucción se extraen puntos de interés de las primeras dos vistas, se identifican las correspondencias y, a partir de ellas se fija la primera cámara en el origen de coordenadas y se obtiene la posición relativa de la segunda cámara. A continuación, se triangulan los puntos, y se comprueba si la reconstrucción es correcta (error de reproyección bajo). En caso negativo, se descartan los resultados obtenidos, y se trata de realizar una nueva inicialización con

el siguiente *frame* y la segunda imagen empleada en la iteración anterior, repitiendo los mismos pasos hasta que se consiga una estructura inicial precisa, de vital importancia para el correcto funcionamiento del algoritmo.

En caso de que la reconstrucción inicial sea correcta, se realiza un ajuste de la estructura inicial, y se pasa a la siguiente iteración del ciclo, tratando de incorporar una nueva vista utilizando la técnica P*n*P. Para ello, en primer lugar, se buscan puntos de interés en la nueva imagen, y para cada uno de ellos se determina su correspondencia con los puntos de interés de imágenes anteriores. A partir de su índice en el conjunto de puntos de interés de imágenes anteriores, se realiza una búsqueda en los puntos ya reconstruidos (en los que previamente se ha almacenado los índices de los puntos empleados para su triangulación), obteniendo (si el punto ya estaba reconstruido) las coordenadas de ese punto en el espacio. A continuación, se resuelve el problema P*n*P, obteniendo la posición de la nueva cámara. Por último, se lleva a cabo una triangulación de las correspondencias obtenidas entre la nueva imagen y las anteriores y, si el resultado es bueno, se incorporan a la nube de puntos los nuevos valores y se realiza un nuevo ajuste. En caso de haberse obtenido una mala triangulación (alto error de reproyección) se descartan los puntos obtenidos y se pasa a la siguiente iteración.

En la siguiente página se detalla de forma esquemática la estructura completa. En negrita al final de algunos pasos se indica la sección del capítulo 3 en la que se detalla la técnica empleada. En el siguiente apartado se presentan los detalles de la implementación del algoritmo.

Estructura del algoritmo

SfM: Estructura completa del algoritmo

Entradas: Matriz de parámetros intrínsecos e imágenes a procesar

Salidas: Nube de puntos con la reconstrucción y posiciones de las cámaras

1. Leer imagen inicial

2. Obtener puntos de interés y descriptores de imagen inicial **(3.2.1)**

3. Mientras haya nuevas imágenes

3.1 Leer nueva imagen, extraer puntos de interés y descriptores **(3.2.1)**

3.2 Si la reconstrucción inicial no es válida

3.2.1 Obtener correspondencias con la imagen anterior **(3.2.2)**

3.2.2 Computar la matriz fundamental con esquema RANSAC **(3.3.1)**

3.2.3 Obtener la matriz esencial. Realizar la descomposición SVD y obtener las 4 posibles soluciones. **(3.3.1)**

3.2.4 Triangular las correspondencias empleando la matriz P obtenida con una solución y comprobar el resultado (puntos frente a la cámara). **(3.3.2)**

3.2.5 Si la solución comprobada no posiciona los puntos frente a la cámara

3.2.5.1 Probar otra matriz P. Triangular y comprobar el resultado. Si es erróneo, seguir probando hasta haber empleado las 4 soluciones.

3.2.5.2 Si ninguna de las 4 soluciones posiciona los puntos frente a la cámara, pasar a la siguiente iteración de 3.

3.2.6 Si la solución posiciona los puntos frente a la cámara

3.2.6.1 Si el error de reproyección es bajo, incorporar puntos a la nube de puntos. Incorporar vistas a la lista. Reconstrucción inicial válida.

3.2.6.2 En caso contrario descartar puntos triangulados y pasar a la siguiente iteración de 3.

3.2.6.2 Realizar ajuste. **(3.5)**

3.3 En caso contrario

3.3.1 Para todas las vistas previas

3.3.1.1 Obtener correspondencias con la imagen actual. (**3.2.2**)

3.3.1.2 Localizar correspondencias en la nube de puntos.

3.3.2 Resolver P*n*P con esquema RANSAC. Obtener matriz P **(3.4)**

3.3.3 Para todas las vistas previas

3.3.3.1 Triangular correspondencias con la imagen actual **(3.3.2)**

3.3.3.2 Si el error de reproyección es bajo, incorporar a la nube de puntos.
Vista válida.

3.3.3.3 En caso contrario, descartar puntos triangulados.

3.3.4 Si la vista es válida, incorporar imagen a la lista y realizar ajuste. **(3.5)**

4. Visualizar nube de puntos obtenida.

4.2 Detalles sobre la implementación

En este apartado se va a detallar la implementación realizada. Se comenzará por presentar el lenguaje y librerías empleados, a continuación, se comentarán otras técnicas implementadas. Por último, se indicará la funcionalidad implementada en los distintos ficheros de los que consta el código. El código ha sido desarrollado en C++. Se ha escogido este lenguaje debido a su gran versatilidad y facilidad para la integración de librerías como OpenCV. Además, se emplea la librería SSBA (*Simple Sparse Bundle* Adjustment [32]) para realizar el ajuste, y PCL (*Point Cloud Library* [33]) para la visualización de nubes de puntos. Para facilitar la compilación de estas librerías se ha escogido trabajar sobre el sistema operativo Linux, concretamente, sobre la distribución CentOS 7. A continuación se presentan más detalles sobre las librerías empleadas.

4.2.1 OpenCV

Se trata de una librería *Open Source* destinada a proporcionar multitud de recursos para el manejo de imágenes, así como numerosas implementaciones de algoritmos de visión por computador con el fin de desarrollar programas relativamente complejos. En el caso concreto de este programa, se han empleado funciones de la librería OpenCV para leer imágenes o secuencias de vídeo, para realizar la detección de puntos de interés, de correspondencias, obtención de la matriz fundamental y resolución del problema P*n*P. Además, se emplean clases de OpenCV para almacenar imágenes o elementos de la misma. Se han utilizado la clase *Mat* (matriz, sirve además para almacenar imágenes), *Vector* (matriz unidimensional, cuenta con numerosos métodos para facilitar el manejo de memoria con este tipo de variables), y algunos tipos para representar puntos de interés, correspondencias y coordenadas.

Funciones de lectura de imágenes

OpenCV incluye la clase *VideoCapture*, que permite leer secuencias de *frames* provenientes de una cámara conectada al dispositivo en el que se ejecute, un archivo de vídeo o incluso una serie de imágenes nombradas adecuadamente. En este programa se emplea dicha clase para ir leyendo las imágenes de distintas entradas. Para ello, se instancia un objeto de esta clase y, cada vez que se desea extraer un *frame* (una vez por iteración del programa), se emplea el operador de direccionamiento >> para guardarlo en un objeto de la clase *Mat*.

Detección de puntos de interés y correspondencias

Uno de los puntos fuertes de OpenCV es que ofrece interfaces para la detección sencilla de puntos de interés. En este trabajo se ha empleado la clase ORB para detectar los puntos de interés, y *BFMatcher* para las correspondencias.

ORB es una clase en la que se definen los parámetros del algoritmo mediante el constructor. Para esta implementación se han escogido los parámetros considerados óptimos en [23]:

```
static Ptr<ORB> cv::ORB::create(int nfeatures = 10000, float scaleFactor=1.2f, int nlevels=8,
int edgeThreshold=31, int firstLevel=0, int WTA_K=2, int scoreType=ORB::HARRIS_SCORE, int
patchSize=31)
```

En la llamada a la función, *nfeatures* es el número de puntos de interés máximo que se desea detectar. Se corresponde con el valor de N mencionado en el apartado 3.2.1. Para este trabajo se ha escogido 10000 puntos, un número alto al haberse trabajado con imágenes con alta resolución. Así, el umbral *t* empleado en FAST se adaptará hasta obtener más de N puntos, ordenándolos por su puntuación *Harris* y escogiendo como puntos de interés los N primeros.

El factor de escala (*scaleFactor*) fija el valor en el que se reduce la imagen en cada etapa de la pirámide de escalas, de forma que en cada nivel se aplica a la imagen un filtro gaussiano de varianza 1.2 veces mayor a la empleada en el nivel anterior. El número de niveles de la pirámide de escalas viene dado por *nlevels*.

EdgeThreshold fija la distancia al borde de la imagen a partir de la que se buscarán puntos de interés, ya que, en el caso de detectar un punto de interés demasiado cerca del borde podría no ser posible aplicar el patrón de *tests* binarios a la totalidad de su entorno (parte se saldrían de la imagen). *PatchSize* es el tamaño del parche empleado en el descriptor (el área en el que se realizan los *tests* binarios). En este caso, ambos valores se fijarán a 31 píxeles, ya que se recomienda que tengan el mismo valor.

WTA_K fija el número de elementos que se comparan en cada *test* para elaborar el descriptor. En este caso, como ya se comentó en el capítulo 2, se emplean *tests binarios*, por lo que su valor será de 2. En cuanto a *firstLevel*, es un valor que en la referencia de OpenCV indican que ha de ser 0. Por último, *scoreType* indica la forma en la que se puntuarán los distintos puntos de interés detectados, en este caso, empleando el detector de esquinas *Harris*, tal y como se indica en [23].

Una vez inicializado, cuenta con los métodos *detect*, que permite encontrar puntos de interés, y *compute*, que los procesa y elabora sus descriptores, en ambos casos funcionando como se

cuenta en el capítulo 3 con la detección de *features* FAST mejorada, y la elaboración de los descriptores BRIEF incluyendo información sobre la rotación.

```
orb.detect(Mat frame, vector<KeyPoint> keypoints);

orb.compute(Mat frame, vector<KeyPoint> keypoints, Mat descriptors);
```

BFMatcher es un detector de correspondencias del tipo *brute force* implementado en OpenCV. Esta clase de detectores compara cada descriptor de la primera imagen con **todos** los descriptores de la segunda. Existe una alternativa empleando *Fast Approximate Nearest Neighbor Search*, basada en *hashing* y destinada a comparar el descriptor desde una imagen a todo un set de imágenes previas. No es útil en este caso, ya que se desea realizar un *matching* imagen a imagen, al ser deseable que se detecte un punto de interés de la imagen que se está procesando en más de una de las imágenes anteriores. De esta forma, al constructor de *BFMatcher* únicamente es necesario indicarle la métrica para realizar la detección de correspondencias. En este caso, al emplear descriptores binarios, se trata de la distancia de *Hamming*.

```
BFMatcher matcher(NORM_HAMMING, CrossCheck = false);
```

Para realizar la detección se emplea el método *knnMatch*, que permite detectar las *k* mejores correspondencias de un descriptor en la otra imagen. Se escoge un *k* de 2, para llevar a cabo el *ratio test* de [21] con el fin de eliminar correspondencias ambiguas.

```
matcher.knnMatch(Mat descriptors1, Mat descriptors2, vector<DMatch> matches, int k=2);
```

La función devuelve las correspondencias en un vector de elementos de la clase *DMatch*, que, para cada correspondencia, almacena un índice de la situación los puntos de interés en los vectores de puntos de interés empleados. Esto permite, por ejemplo, alinear los puntos de interés. Tanto los índices como los puntos de interés de todas las imágenes son almacenados durante toda la ejecución del programa, ya que serán empleados, por ejemplo, para realizar triangulaciones con futuras vistas.

Obtención de la matriz fundamental

OpenCV cuenta con una función que permite computar la matriz fundamental a partir de correspondencias entre puntos de ambas imágenes. El método implementado es el detallado en el capítulo 3. Utiliza un esquema RANSAC en el que se va resolviendo la matriz fundamental empleando 8 correspondencias, y analizando el número de puntos de interés que se ajustan a ese modelo. La función de OpenCV es:

```
Mat findFundamentalMat(InputArray points1, InputArray points2, int method=FM_RANSAC, double
param1=0.5, double param2=0.99, OutputArray mask=inliers )
```

Detalles sobre la implementación

Los parámetros de entrada son dos vectores con las coordenadas de los puntos de interés, alineados de forma que el punto de interés del elemento *i* del vector 1 corresponda al del elemento *i* del vector 2. Para ello, se ordenan empleando las correspondencias obtenidas previamente. Además, existen dos parámetros de entrada para el uso de *RANSAC*: el primer parámetro es la máxima distancia permitida de un punto de la segunda imagen a la línea *epipolar* definida por su correspondencia en la primera imagen (en este caso, medio píxel) para ser considerado un *inlier* (en el apartado 3.3.1 se denomina a esta distancia *d*). El segundo parámetro indica el nivel de confianza de que la matriz obtenida es correcta, y se emplea en el cálculo adaptativo del número de iteraciones a realizar en RANSAC. Por último, se obtiene una máscara: un vector de la misma longitud que los puntos de interés (y correspondencias) empleados para obtener la matriz fundamental, cuyo valor será de 1 en los puntos considerados *inliers* para la matriz fundamental estimada, y 0 en caso contrario.

La función retorna la matriz fundamental obtenida.

Resolución del problema Perspective-n-Point

El último recurso de OpenCV empleado en el trabajo se trata de la implementación de la solución del problema P*n*P. La llamada a la función se efectúa de la siguiente manera:

```
solvePnPRansac(vector<Point3f> pointcloud_aligned, vector<Point2f> framePoints, Mat_<float> K,
Mat distCoef, Mat r, Mat t, bool useExtrinsicGuess=false, int itCount=1000, float
reprojError=0.5, float conf=0.99, OutputArray inliers, int flags=CV_EPNP);
```

Esta función permite, a partir de las correspondencias entre los puntos en la nueva imagen (*framePoints*), y los puntos previamente reconstruidos del espacio[8] (*pointcloud_aligned*), obtener el vector de traslación y la matriz de rotación que definen la posición de la cámara en la nueva vista. Para ello, las entradas de la función son dos vectores que contienen dichos elementos alineados, de forma que el punto en la posición *i* del vector de puntos en el espacio se corresponda con el punto en la posición *i* en el vector con los puntos de interés de la imagen. Además, se introduce la matriz de parámetros intrínsecos, K, y los coeficientes de distorsión (en este caso, vacíos, al emplear una cámara sin distorsión). Como parámetros de la función, existe la posibilidad de emplear una estimación inicial de R y t (*useExtrinsicGuess*), no empleada en este caso. El número máximo de iteraciones se ajusta a 1000, el error de reproyección máximo para considerar un punto un *inlier* será de 0.5 píxeles, el nivel de confianza de la estimación 0.99, y el método empleado será EP*n*P (el indicado en el capítulo 2). Como salidas, se obtienen el vector de

[8] Más adelante se indica cómo se obtiene dicha correspondencia

traslación, un vector de rotación (se transforma en matriz empleando la función *Rodrigues*), y otro vector con los índices de los puntos de la imagen actual que se han considerado *inliers* en la estimación.

4.2.2 SSBA

Para la realización del *Bundle Adjustment* se ha escogido emplear la librería *Simple Sparse Bundle Adjustment*, que implementa una variante poco densa (*sparse*) para la resolución del algoritmo de Levenberg-Marquardt. Esto se debe a que, como se indica en [1], al construir el Jacobiano (**J**) en el desarrollo del algoritmo para la minimización del error de reproyección se obtiene una matriz poco densa. De esta forma, es posible compactar las ecuaciones del algoritmo, de forma que su resolución sea mucho más eficiente y menos compleja, tal y como se detalla en [34]. La librería empleada incluye una implementación robusta de este método [32], y es sencillo implementarla en el código. Para llevar a cabo el ajuste, únicamente se ha de adaptar los datos obtenidos hasta el momento a los empleados por la librería (por ejemplo, de *keypoints* a coordenadas), y emplear la clase *CommonInternalsMetricBundleOptimizer*. Entre otras cosas, es posible escoger si se desea que se ajuste la matriz de parámetros intrínsecos, si se desean introducir parámetros de distorsión… Como se cuenta con cámaras calibradas, y sin distorsión se escoge no realizar un ajuste sobre los parámetros intrínsecos de las cámaras.

4.2.3 PCL

La última librería empleada es *Point Cloud Library* [33]. Esta librería proporciona desde herramientas sencillas para visualizar nubes de puntos, hasta métodos estadísticos para filtrar puntos incorrectos, funciones para generar mallados, modelado y texturas. Además, se encuentra en parte integrada con OpenCV, permitiendo trabajar con sus estructuras de datos. En este trabajo se emplea únicamente para llevar a cabo la visualización de la nube de puntos final. Para ello, basta con guardar las coordenadas de la nube en un vector de puntos tridimensionales, y guardarlos en un objeto de la clase *PointCloud*. Una vez poblada, se instancia un visualizador y se indica que muestre la nube de puntos obtenida.

4.2.4 Búsqueda de las correspondencias entre puntos de una imagen y la nube de puntos

Obtener las correspondencias entre los puntos de una imagen y los ya triangulados en el espacio es sencillo, pero no trivial. Para solucionarlo, se ha optado por guardar cada punto en el

espacio en una estructura (*Point3D*), en la que se recogen las coordenadas de dicho punto y un vector, de longitud igual al número de vistas incorporadas hasta el momento. El valor de cada posición en el vector será de -1 si ese punto no se ha triangulado a partir de la vista correspondiente a la posición indicada, o el índice del punto de interés empleado para su reconstrucción si se ha hecho a partir de un punto de esa vista. Por ejemplo, un -1 en la segunda posición indica que ese punto no ha sido reconstruido a partir de un punto de interés que aparezca en la segunda imagen procesada. Un valor distinto de -1 en la quinta posición indica que ese punto se ha reconstruido a partir del punto de interés indicado por el valor, que es el índice del punto de interés en la quinta imagen.

De esta forma, para obtener las correspondencias entre los puntos de una imagen y la nube de puntos, en primer lugar, se obtienen las correspondencias entre las dos imágenes. A continuación, para cada correspondencia, se busca en la nube de puntos el índice del punto de interés de la correspondencia en la imagen previa (asegurando que no se haya utilizado previamente). Si se encuentra, se asocia el punto de la nube en el que se ha encontrado con el punto de la imagen que se está procesando para la posterior resolución de P*n*P. Además, se añade al punto de la nube el índice del punto de interés en la nueva vista incorporada. Esta búsqueda de puntos de la nube se realiza para todas las imágenes previas con las que tenga un número significativo de correspondencias. Esta técnica se aplica en el punto 3.3.1.2 de la estructura del algoritmo presentada en el apartado 4.1.

4.2.5 Mecanismos adicionales de filtrado de correspondencias

Una vez obtenidas las correspondencias, a pesar de haber superado el *ratio test*, todavía quedan numerosas correspondencias erróneas que pueden provocar un incorrecto funcionamiento del algoritmo. Para solucionarlo, en este trabajo se proponen dos mecanismos adicionales de filtrado de correspondencias:

Un control de distancia, en el que una vez encontradas las correspondencias, se comprueba que la distancia del punto de interés en una imagen y la otra sea menor que 300 píxeles (asumiéndose un movimiento de cámara relativamente pequeño entre imágenes) para eliminar falsos positivos.

Otro mecanismo que se emplea para filtrar correspondencias consiste en emplear el cálculo de la matriz fundamental. Tras obtener las correspondencias, se realiza el cálculo de la matriz fundamental con un error de reproyección tolerado de 0.5 píxeles. De esta forma, se modela la geometría entre las dos cámaras de manera fiable, y se obtiene un vector con los elementos

empleados para el cálculo que se consideran *inliers*, es decir, que se adaptan al modelo. Por lo tanto, permite eliminar las correspondencias que no se han considerado "aptas" para el modelo. En el caso de la reconstrucción inicial, tras este filtrado se realiza una nueva estimación de la matriz fundamental con un error de reproyección tolerado todavía menor, de la que se extraerá la posición relativa de las cámaras iniciales. En el caso de que se esté incorporando una nueva vista a la reconstrucción, se realiza un cálculo de la matriz fundamental para cada par de imágenes, y se buscan únicamente en la nube de puntos aquellas correspondencias que se hayan ajustado al modelo. Esto es posible gracias al poco coste asociado con computar una matriz fundamental, como se verá en el siguiente apartado. Dichos mecanismos se aplican en los puntos 3.2.1 y 3.3.1.1 de la estructura del código presentada en el apartado 4.1.

4.2.6 Distribución del código en ficheros

El algoritmo detallado se encuentra separado en distintos ficheros:

En los ficheros *feature_matcher.cpp* (85 líneas) y *feature_matcher.h* (33 líneas) se encuentra el código correspondiente a las funciones encargadas de detectar los puntos de interés en las imágenes y encontrar las correspondencias.

En los ficheros *SfM.cpp* (321 líneas) y *SfM.hpp* (53 líneas) se encuentran el código correspondiente a las funciones encargadas de la triangulación, la estimación de la matriz fundamental y la descomposición de la matriz esencial.

En los ficheros *bundle_adjust.cpp* (158 líneas) y *bundle_adjust.h* (13 líneas) se encuentran las instrucciones necesarias para transformar la estructura de datos del algoritmo a la necesaria para realizar el ajuste, y la llamada a la función que realiza el ajuste (librería SSBA). En los ficheros *visualization.cpp* (68 líneas) y *visualization.h* (26 líneas) está el código necesario para llevar a cabo la visualización de la nube de puntos al terminar el código.

Por último, en el fichero *main.cpp* (314 líneas) contiene el código correspondiente a la estructura presentada en el apartado 4.1.

4.2.7 Uso de las librerías en el algoritmo

Para terminar, se presenta la estructura introducida en el apartado 4.1 a la que se ha añadido las librerías empleadas en cada caso (líneas en rojo). Las líneas en negro se corresponden al código propio desarrollado en el trabajo.

SfM: Estructura completa del algoritmo

Entradas: Matriz de parámetros intrínsecos e imágenes a procesar

Salidas: Nube de puntos con la reconstrucción y posiciones de las cámaras

1. Leer imagen inicial (OpenCV)

2. Obtener puntos de interés y descriptores de imagen inicial (parcialmente OpenCV)

3. Mientras haya nuevas imágenes

3.1 Leer nueva imagen, extraer puntos de interés y descriptores **(**parcialmente OpenCV)

3.2 Si la reconstrucción inicial no es válida

3.2.1 Obtener correspondencias con la imagen anterior (parcialmente OpenCV)

3.2.2 Computar la matriz fundamental con esquema RANSAC (OpenCV)

3.2.3 Obtener la matriz esencial. Realizar la descomposición SVD y obtener las 4 posibles soluciones. **(3.3.1)**

3.2.4 Triangular las correspondencias empleando la matriz P obtenida con una solución y comprobar el resultado (puntos frente a la cámara). **(3.3.2)**

3.2.5 Si la solución comprobada no posiciona los puntos frente a la cámara

3.2.5.1 Probar otra matriz P. Triangular y comprobar el resultado. Si es erróneo, seguir probando hasta haber empleado las 4 soluciones.

3.2.5.2 Si ninguna de las 4 soluciones posiciona los puntos frente a la cámara, pasar a la siguiente iteración de 3.

3.2.6 Si la solución posiciona los puntos frente a la cámara

3.2.6.1 Si el error de reproyección es bajo, incorporar puntos a la nube de puntos. Incorporar vistas a la lista. Reconstrucción inicial válida.

3.2.6.2 En caso contrario descartar puntos triangulados y pasar a la siguiente iteración de 3.

3.2.6.2 Realizar ajuste. (parcialmente SSBA)

3.3 En caso contrario

3.3.1 Para todas las vistas previas

3.3.1.1 Obtener correspondencias con la imagen actual. (parcialmente OpenCV)

3.3.1.2 Localizar correspondencias en la nube de puntos.

3.3.2 Resolver P*n*P con esquema RANSAC. Obtener matriz P **(**OpenCV)

3.3.3 Para todas las vistas previas

3.3.3.1 Triangular correspondencias con la imagen actual **(3.3.2)**

3.3.3.2 Si el error de reproyección es bajo, incorporar a la nube de puntos.
Vista válida.

3.3.3.3 En caso contrario, descartar puntos triangulados.

3.3.4 Si la vista es válida, incorporar imagen a la lista y realizar ajuste. (parcialmente SSBA)

4. Visualizar nube de puntos obtenida (PCL)

4.3 Evaluación

Una vez presentados todos los conceptos relativos tanto a la base teórica como a la implementación, se procede a evaluar el desempeño del algoritmo. Para ello, se va a estudiar la calidad de la reconstrucción, rendimiento y escalabilidad de la implementación, con el fin de estudiar los puntos débiles del algoritmo y cuellos de botella, de cara a proponer mejoras para su futura implementación. Además, se va a comentar cómo afectan distintas alternativas en la implementación y parámetros comentados. En concreto, se va a probar cómo afecta el uso del filtrado de correspondencias empleando la matriz fundamental frente a una selección basada en la distancia máxima entre correspondencias. También se analizará la diferencia entre emplear como técnica de triangulación la técnica iterativa presentada en el capítulo 2, o emplear como triangulación la inicialización de esta técnica (resolución de un único sistema). Por último, se estudiará cómo el variar el número de puntos de interés detectados en cada imagen afecta al resultado.

Las pruebas se realizan sobre un set de 32 fotos de la estatua de Leonardo Torres Quevedo situada en la Plaza de la Ciencia. Dicho set consta de fotos de alta resolución (4368x2912 píxeles, sensiblemente superior a lo que en vídeo serían imágenes 4K) que rodean toda la estatua. Además, es interesante el empleo de este set de fotos debido a que la posición desde la que se realiza la última fotografía coincide prácticamente con la de la primera, por lo que es un desafío para el programa es corregir el error acumulado a lo largo de toda la reconstrucción para poder triangular correctamente los últimos puntos, y posicionar ambas cámaras en la misma localización. Las imágenes han pasado previamente por una máscara que "recorta" el *background*, con el fin de visualizar de manera sencilla el resultado.

4.3.1 Calidad de la reconstrucción

En este apartado se va a evaluar la calidad de la reconstrucción obtenida: es decir, si los puntos reconstruidos se corresponden con la escena de forma fidedigna, la cantidad de errores cometidos, y los resultados de reconstrucción con la cámara. Para ello, se va a inspeccionar la nube de puntos resultantes (cantidad de puntos reconstruidos erróneos, que no pertenecen a la figura) y comparar las posiciones obtenidas de las cámaras.

Implementación 1: triangulación no iterativa y filtrado por distancia

La primera aproximación fue *a priori* la menos sofisticada. El método de triangulación (usado en los puntos 3.2.4 y 3.3.3.1 de la estructura del algoritmo) es lineal de mínimos cuadrados (sobre el papel, inferior a su equivalente iterativo en calidad de la reconstrucción). El filtrado de correspondencias incorrectas (realizado en los puntos 3.2.1 y 3.3.1.1 de la estructura del algoritmo) consiste en admitir una distancia máxima entre dichas correspondencias de 300 píxeles, y, además, se tolera como error de reproyección máximo 10 píxeles en la triangulación.

Observando la nube de puntos resultante, se comprueba que, pese a que el error de reproyección calculado es bajo, se ha producido una serie de errores en la nube de puntos al cerrar el bucle (volver a la posición inicial de la cámara). Esto se debe a que, debido al error acumulado en la posición de las cámaras, algunos puntos triangulados con los primeros *frames* y los últimos aparecen en posiciones erróneas. Atendiendo a las posiciones de las cámaras, antes del ajuste final se comprueba cómo la posición final tiene un pequeño error con respecto a la inicial ([-0.02387; -0.23911; -0.03898] frente a [0,0,0]) que, al realizar el ajuste se soluciona ([-0.16686; -0.13512; 2.33375] frente a [-0. 16238; -0.13419; 2.35631]).

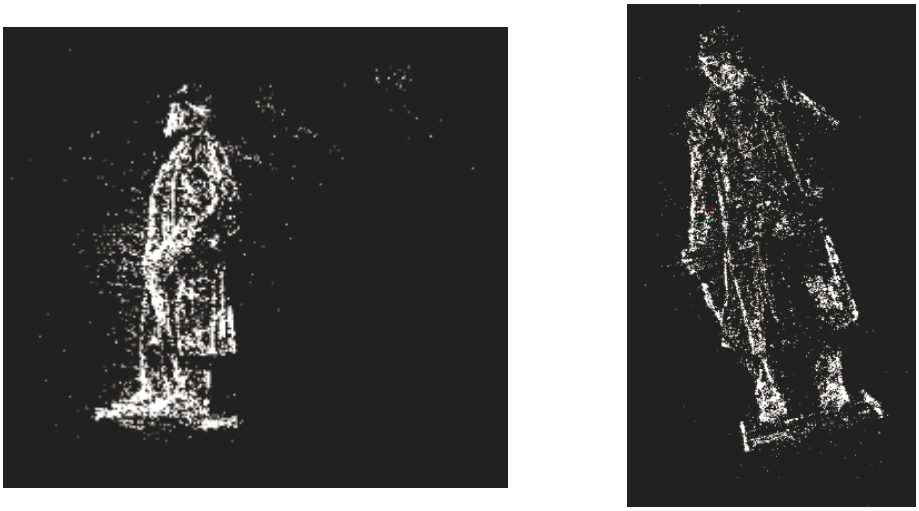

Figura 4.1: Resultados de la reconstrucción con la primera implementación.

De estos resultados se extraen dos importantes conclusiones: en primer lugar, que el algoritmo es capaz de determinar correctamente que la posición de la última imagen es prácticamente coincidente con la primera, llevando a cabo con éxito el cierre del lazo de captura de imágenes. Por otra parte, hay que comentar que las coordenadas obtenidas no tienen ningún valor en

términos absolutos. Al realizar el ajuste final, se desplaza el punto inicial del origen, y se re-escala la reconstrucción completa, de forma que el resultado final es correcto, pero no se puede extraer información sobre la posición absoluta de la reconstrucción.

Implementación 2: filtrado empleando la estimación de F

Para tratar de mejorar los resultados obtenidos, se trata de sustituir el método mediante el cual se filtran las correspondencias. En los puntos 3.2.1 y 3.3.1.1 de la estructura del algoritmo se pasa de emplear un filtrado teniendo en cuenta la distancia entre las correspondencias a emplear una estimación de la matriz fundamental entre cada par de imágenes, y únicamente conservar las correspondencias empleadas como *inliers* en la estimación (como se comenta en el apartado 4.2.5). El resto de parámetros se mantienen con respecto a la prueba anterior.

Al implementarse este nuevo filtro, se rechazan todas las correspondencias que estén a más de 0.5 píxeles de la línea *epipolar* definida por el punto en la otra imagen. De esta forma, se trata de un filtro notablemente más restrictivo que el anterior. En consecuencia, se elimina gran cantidad de puntos erróneos. El resultado obtenido es notablemente mejor, con muchos menos errores visibles alrededor de la estatua. La reconstrucción es capaz de posicionar todas las cámaras de manera adecuada, coincidiendo otra vez la posición de la última con la primera. Además, al contar con menos correspondencias, y, en consecuencia, una reconstrucción menos, se pasa de tardar 23 minutos 48 segundos a 15 minutos 59 segundos en procesar la totalidad de las imágenes. Esta drástica mejora se debe, además, a que este método de filtrado es menos costoso *per se*.

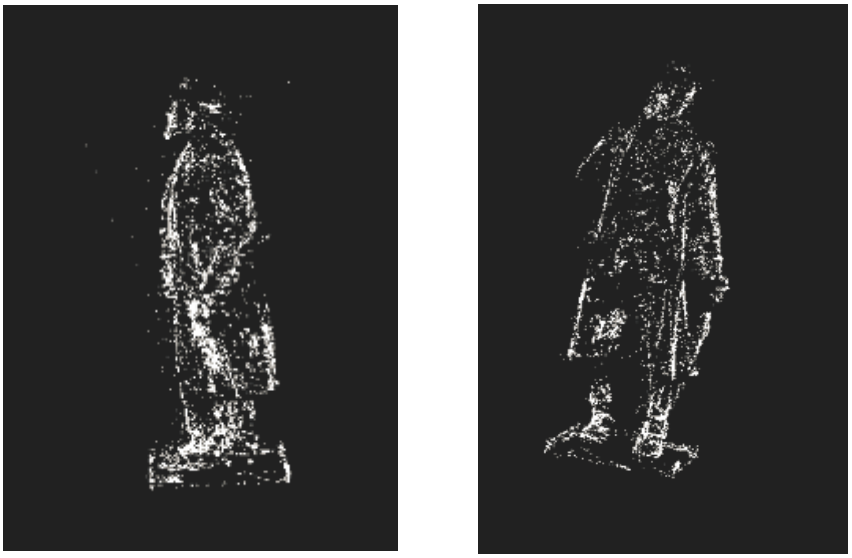

Figura 4.2: resultado de la reconstrucción con la segunda implementación

Implementación 3: triangulación iterativa

Por último, se trata de mejorar el resultado introduciendo la triangulación iterativa en los puntos 3.2.4 y 3.3.3.1. Sobre el papel, se debería obtener una triangulación más eficaz con prácticamente el mismo tiempo empleado en triangular cada punto. En la práctica, el tiempo de ejecución obtenido es bueno, incrementando el total en únicamente 2 segundos (unos 0.06 segundos más por imagen). Sin embargo, los resultados obtenidos son exactamente idénticos al caso no iterativo, por lo que se escoge emplear el método más sencillo a igualdad de resultados.

Implementación 4: reconstrucción con una mala inicialización

En todos los casos, el programa ha sido capaz de reconstruir la posición de todas las cámaras. Sin embargo, esto no siempre va a ser posible en un entorno más "realista". Para comprobar que es capaz de llevar un desempeño correcto pese a que no se reconstruyen todas las cámaras, se intenta realizar una inicialización variando el error de reproyección tolerado para la estimación de la matriz fundamental de medio píxel a 0.01 píxeles. Esta condición provoca que con las dos primeras imágenes no sea posible llevar a cabo la reconstrucción inicial. Aun así, empleando las siguientes imágenes es capaz de completar la reconstrucción de manera adecuada.

Visualización de las cámaras

Por último, se representan las posiciones de las cámaras para visualizar los resultados obtenidos, comprobando que el algoritmo ha obtenido sus posiciones correctamente.

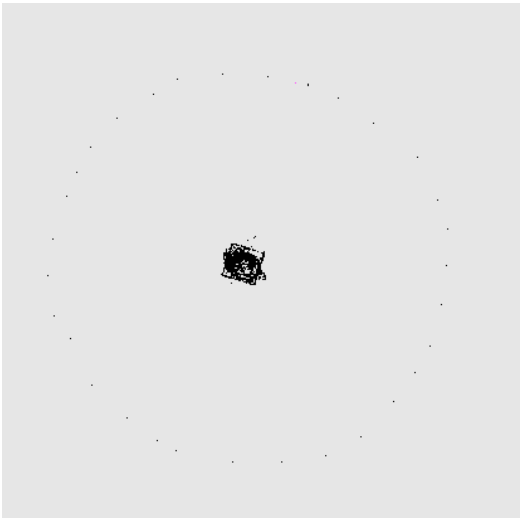

Figura 4.3: *visualización de la* ***posición de las cámaras*** *empleadas para la reconstrucción*

4.3.2 Rendimiento del algoritmo

Se va a estudiar el rendimiento de las distintas etapas del algoritmo, atendiendo, principalmente, al tiempo invertido en realizar cada parte. Cabe comentar que, a excepción de las funciones de OpenCV implementadas, ninguna parte del código está paralelizada, por lo que se ejecutará en un único hilo y de manera "secuencial". Esto permitirá identificar los cuellos de botella de manera más sencilla, con el fin de determinar posteriormente una paralelización eficaz.

Número de puntos de interés empleado

Se ha probado la ejecución del algoritmo con distintos números de puntos de interés detectados en cada imagen. Hay que comentar que el número de puntos de interés a emplear depende, en gran medida, de la resolución de la imagen. Así, en [35], por ejemplo, proponen emplear 1000 puntos de interés para imágenes VGA (0.3 megapíxeles), y 2000 para imágenes de 0.5 megapíxeles. Al contar con imágenes de 12.7 megapíxeles, se ha de emplear una cantidad notablemente superior. En primer lugar, se probó con 10000 puntos, obteniendo resultados bastante densos. En pruebas posteriores se ha ensayado con diferentes números de puntos entre 5000 y 10000. Como es obvio, al disminuir el número de puntos mejora sensiblemente la velocidad de ejecución, al tener que determinar menos correspondencias, y reconstruir menos puntos. Con las pruebas se ha determinado que el número óptimo de puntos de interés se encuentra en un término medio: empleando 7500 puntos se consigue reducir el tiempo de detección de correspondencias notablemente (de 1.8 segundos por cada par de imágenes a 1 segundo), manteniendo un número lo suficientemente alto de puntos de interés como para que al filtrar los incorrectos se reconstruyan suficientes puntos en el espacio para asegurar la estabilidad y robustez del algoritmo. De esta forma, se reduce el número de puntos en la nube aproximadamente a la mitad (de 50.000 a 23.000), manteniendo la calidad de la reconstrucción y la correcta determinación de las posiciones de las cámaras. El tiempo total se ve reducido de 15 minutos 59 segundos a 9 minutos y 56 segundos, una gran mejora debida a la cantidad de veces a lo largo del algoritmo que se realiza este procedimiento.

Detección de puntos de interés

El primer paso con cada nueva imagen consiste en detectar sus puntos de interés. El uso de ORB permite que se realice con rapidez, en tiempos ligeramente inferiores a 0.3 segundos. Hay que tener en cuenta que se están empleando imágenes de muy alta resolución, por lo que se trata de un tiempo bastante corto, y es difícil realizar optimizaciones en este apartado.

Detección de correspondencias

Se trata de uno de los pasos más costosos del algoritmo, al tener que comparar una gran cantidad de descriptores. A pesar de que las comparaciones son rápidas (únicamente se ha de computar la distancia de Hamming), se realiza gran cantidad de ellas con esta aproximación (cada punto de interés se compara en todos los puntos de interés de la otra imagen, por lo que se realizan 7500·7500 comparaciones), por lo que se invierte 1 segundo en la detección de correspondencias por cada par de imágenes. Además, al tener que obtener las correspondencias entre cada nueva imagen y todas las anteriores, el tiempo por *frame* invertido en detectar correspondencias crece linealmente.

Estimación de la matriz fundamental

En cuanto al rendimiento presentado por la estimación de la matriz fundamental, presenta un tiempo de ejecución prácticamente despreciable frente a los tiempos comentados. Se debe a la buena implementación realizada con OpenCV, tardando un tiempo de unos 0,1 milisegundos por estimación.

Triangulación

Al emplearse una triangulación no iterativa, y basada en resolver un sistema de ecuaciones lineal, cada triangulación es prácticamente instantánea, siendo el tiempo invertido en triangular unos 500 puntos entre dos imágenes de 4 milisegundos. Sin embargo, normalmente se triangulan puntos de más de 2 vistas en cada iteración, llegando a ser de 9 milisegundos para 4 vistas en el peor caso, y una media de 5 ms para 3 vistas.

Búsqueda de puntos en la nube

Se trata de otro de los grandes cuellos de botella en el algoritmo. Se realiza una búsqueda en toda la nube de puntos por cada correspondencia detectada en cada imagen. Teniendo en cuenta que se buscan correspondencias entre cada nueva imagen y todas las anteriores, es una de las partes del algoritmo en la que más tiempo se invierte, creciendo también con el número de imágenes, y llegando a ser de 9 segundos.

Resolución de PnP

Al igual que la obtención de la matriz fundamental, se trata de un problema que se resuelve en un espacio muy corto de tiempo, y que se realiza una vez por cada nuevo *frame*, empleándose 50 milisegundos, por lo que se trata de una de las partes más eficientes del algoritmo.

Bundle adjustment

El ajuste realizado al final de cada iteración es vital para el correcto funcionamiento del algoritmo, pero, conforme aumenta la cantidad de datos procesada, también se convierte en una parte costosa, llegando a invertirse hasta 2.2 segundos en la última iteración. Cabe comentar que el tiempo invertido depende en gran parte de la velocidad con la que converge el algoritmo de Levenberg-Marquardt, no solo dependiendo de la cantidad de datos empleada, lo que provoca una notable variabilidad en el tiempo invertido en resolver este paso.

Se presenta una gráfica realizada para visualizar el tiempo empleado con cada fase, y cómo evoluciona conforme aumenta el número de imágenes, representando aquellos tiempos que varían con el número de imágenes de forma significativa. Se puede apreciar el crecimiento lineal del tiempo invertido en la detección de correspondencias, y que conlleva la mayor parte del tiempo de procesado dedicado a cada *frame*. Por lo demás, se observa cómo el tiempo destinado al ajuste tiene una tendencia creciente, aunque con gran variablidad, similar al caso del tiempo de localización de puntos 2d en la nube.

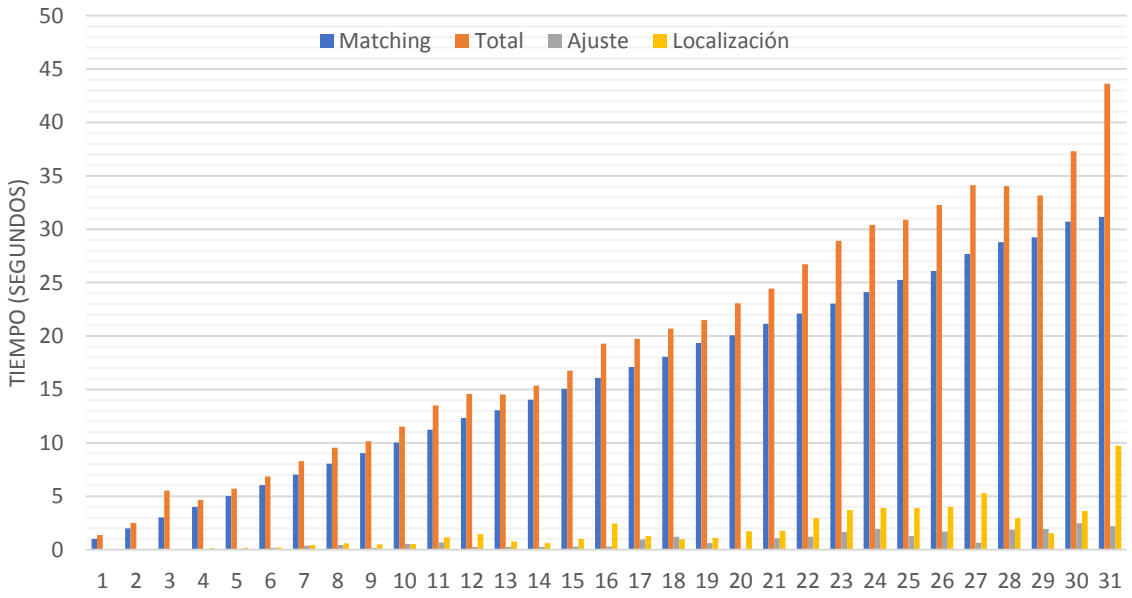

***Figura 4.4*: análisis del rendimiento.** *Variación de los tiempos invertidos en la detección de correspondencias, ajuste y tiempo total por frame.*

4.3.3 Escalabilidad

Uno de los principales problemas de la implementación es su escalabilidad. Para hacer posible la identificación de una vista en la que aparecen elementos previamente computados, como sería el ejemplo de un coche que da la vuelta a una manzana y vuelve a circular por la posición inicial, o como sucede con el set de imágenes empleados para evaluar la implementación, es necesario guardar empleando algún mecanismo información sobre las vistas previamente procesadas.

Para ello, en esta implementación se opta por guardar los puntos de interés de imágenes previas, y, cada vez que llega una nueva imagen, compararla con la totalidad de puntos de interés procesados hasta el momento. Se trata del procedimiento "tradicional", conocido por tener una complejidad $O(n^4)$, siendo n el número de imágenes a procesar [36]. Este mecanismo no solo es costoso a nivel de memoria consumida (se guarda una gran cantidad de datos), sino que también tiene un coste elevado en cuanto al tiempo consumido para procesar cada imagen, al tener que comprobar multitud de correspondencias, siendo uno de los pasos que más tiempo conlleva. De esta forma, se trata de una aproximación válida para un número de imágenes relativamente reducido (menor que 50 imágenes de alta resolución). Sin embargo, si se desea implementar con objeto de resolver el problema de SLAM, se ha de emplear otro mecanismo menos costoso, al tener que procesar una gran cantidad de imágenes para funcionar en tiempo real.

4.3.4 Conclusión

La principal conclusión extraída de la evaluación realizada es que, junto con el *bundle adjustment*, el paso más importante del algoritmo es la detección de puntos de interés en una imagen y su posición en otras imágenes (correspondencia). Se ha comprobado que la variación del número de puntos de interés detectados tiene un impacto enorme en las prestaciones del algoritmo. Además, la aparición de correspondencias erróneas tiene mucho más impacto en la calidad de la reconstrucción obtenida que el método de triangulación utilizado (llegando incluso a no poderse realizar si no se detectan suficientes correspondencias de calidad). Por último, la detección de correspondencias es el principal cuello de botella en cuanto a escalabilidad y rendimiento del algoritmo.

4.4 Mejoras propuestas

Tras analizar el desempeño del algoritmo en distintos apartados, se va a presentar mejoras basadas tanto en el análisis realizado, como en las aplicadas en otros sistemas SfM o SLAM.

4.4.1 Calidad de la reconstrucción

Se trata quizás del aspecto en el que menor margen de mejora hay, o en el que menos interesa mejorar de cara al desarrollo de un sistema SLAM. Esto se debe a que, en la mayor parte de los casos, las técnicas que obtienen mejores resultados parten de un set de imágenes conocido de antemano. Así, por ejemplo, en [10], uno de los sistemas SfM más robustos y empleados, orientado a la reconstrucción de imágenes desordenadas obtenidas de internet, realizan una correspondencia entre todas las imágenes disponibles, además de una homografía para asegurar que **no** es posible modelar el movimiento entre ambas cámaras únicamente mediante ella para evitar casos *degenerados*. Se escoge el par de imágenes con mayor número de correspondencias, y se va incorporando cámaras a la reconstrucción, empezando por aquellas en las que se observan más puntos provenientes de las correspondencias obtenidas en las imágenes iniciales. Se obtiene una reconstrucción mucho más robusta que en nuestro caso, pero, obviamente, únicamente es aplicable a una reconstrucción *offline*.

Sin embargo, analizando el comportamiento del algoritmo ORB se comprobó que, en muchas ocasiones, la inicialización tarda mucho en ser realizada adecuadamente. Dicho sistema emplea únicamente las dos últimas imágenes obtenidas para tratar de establecer una escena inicial. Se podría mejorar esta técnica empleando un esquema intermedio entre el que emplea Bundler y el algoritmo implementado en este trabajo, guardando únicamente los puntos de interés de los *frames* y sus correspondencias hasta que se realice una reconstrucción inicial exitosa, empleando para ello los pares de imágenes más adecuados.

4.4.2 Rendimiento

El apartado en el que se pueden presentar más mejoras es en el rendimiento. Atendiendo a los resultados presentados a la evaluación, existen tres partes del algoritmo en las que se ha de mejorar de forma notable el rendimiento, de cara a una futura operación en tiempo real. Se trata de la detección de correspondencias, el mecanismo para localizar puntos 2d en la nube de puntos y el *bundle adjustment*. A continuación, se presentarán ideas para mejorar el rendimiento del algoritmo en estos tres aspectos

Detección de correspondencias

Se trata de la parte del algoritmo con mayor margen de mejora. El comparar todos los puntos de interés de una imagen con los de la otra se trata de una aproximación demasiado costosa. Al tratarse de un sistema destinado a SLAM, es seguro suponer que la variación entre la posición de un punto de interés en un *frame* y ese mismo punto en el siguiente tendrá poca variación. Por lo tanto, una aproximación que aceleraría de forma notable la detección de correspondencias consistiría en realizar una búsqueda guiada únicamente en un entorno pequeño alrededor de la posición de cada punto de interés en el *frame* anterior. En ORB-SLAM [35], con una filosofía similar, se propone realizar una búsqueda guiada empleando un modelo de velocidad constante a partir del movimiento anterior. De esta forma, solo se compararía una pequeña cantidad de puntos, agilizando notablemente la detección de correspondencias. Además, si para realizarla se divide la imagen en celdas, sería posible paralelizar el proceso de forma eficaz (detectando correspondencias entre celdas, entendidas como imágenes más pequeñas reflejando una parte del *frame* global).

Localización de puntos en la nube

La principal mejora que se puede realizar en este apartado consiste en tratar de reducir el número de vistas en las que se ha de buscar correspondencias (*ver parte de escalabilidad*). Otra opción más óptima consistiría en añadir a cada elemento de la nube de puntos el descriptor asociado al punto de interés, de forma que, en lugar de tener que obtener todas las correspondencias entre imágenes es posible obtener correspondencias directamente con los puntos del espacio. En ORB-SLAM proponen escoger el descriptor con la menor distancia de Hamming al resto de descriptores asociados a los puntos de interés que se han empleado para reconstruir el punto 3D.

Bundle Adjustment

Tradicionalmente, se trata del paso más costoso de SfM, y por el que se ha considerado hasta los últimos años que SfM no era válido para aplicaciones SLAM. La librería implementada en este código presenta una solución del *bundle adjustment* bastante rápida. Una primera alternativa consiste en emplear una librería más potente, como Ceres, un *solver* no lineal creado en google [37]. Sin embargo, no se ha de esperar una mejora muy destacable. En el siguiente apartado se sugieren mejoras para el rendimiento y escalabilidad, con un notable impacto en la velocidad necesaria para el ajuste, sin necesidad de modificarlo.

4.4.3 Separación de *tracking* y *mapping*

A continuación, se va a aclarar el paradigma actual en los sistemas vSLAM basados en detección de puntos de interés: la separación de los procesos de *tracking* y *mapping*.

Se conoce como *tracking* al procedimiento por el cual se obtiene la posición de la cámara en el espacio, y por *mapping* a la reconstrucción de los puntos de la imagen en el espacio. Esta separación fue introducida en PTAM [38], siendo el primer sistema SLAM basado en una técnica del tipo SfM en ofrecer un buen rendimiento operando en tiempo real. Desde entonces, se trata de la fórmula empleada en la mayor parte de sistemas de este tipo.

Básicamente el sistema cuenta con dos hilos (*threads*) corriendo en paralelo. El primer *thread* lleva a cabo el *tracking*, es decir, se encarga de procesar cada nueva vista obteniendo sus puntos de interés y, a partir de correspondencias ya sea con otra imagen (reconstrucción inicial) o con los puntos reconstruidos por el *mapping*, se lleva a cabo una estimación de la posición de la cámara. A continuación, se decide si la nueva imagen aporta la suficiente información novedosa al mapa previamente generado o no. En caso afirmativo, se decide que la nueva imagen es un *keyframe*. Por último, se realiza una versión ligera del *bundle adjustment*, consistente en ajustar únicamente la posición de la cámara.

El segundo *thread* se encarga de realizar el procesado de los *keyframes*, elaborando el *mapping* a partir de ellos. De esta forma, se toma como entrada los puntos de interés y posición de la cámara obtenidos, y se lleva a cabo la triangulación de los nuevos puntos, seguida de un ajuste global. Esto permite que, a pesar de que el ajuste tarda más de lo necesario para implementarlo directamente en tiempo real, al únicamente tener que ser realizado para las *keyframes*, que llegarán cada cierto número de *frames* normales.

De esta forma, se consigue obtener un *tracking* continuo de la cámara, sabiendo en todo momento dónde se encuentra con un coste mucho menor, al no realizar una triangulación de nuevos puntos con cada nueva imagen. La reducción del número de puntos reconstruidos, evitando reconstruir nuevos puntos que no aportan información relevante sobre el entorno permite, además, reducir el tiempo empleado durante el propio *tracking* ya que, al contarse con nubes de puntos menos densas, se ha de encontrar muchas menos correspondencias.

La dificultad de dicha técnica reside en seleccionar los *keyframes* adecuados, de forma que se cuente con suficiente información sobre el entorno para localizar la cámara con precisión en todo momento y que, además, el sistema sea capaz de responder ante movimientos más impredecibles.

Mejoras propuestas

La aproximación básica propuesta en PTAM consiste en establecer una distancia mínima recorrida entre *keyframes*, lo que no asegura un suficiente cambio visual. Para solucionarlo, en ORB-SLAM proponen insertar un nuevo *frame* como *keyframe* cada vez que el sistema haya procesado un mínimo de 20 *frames* con respecto a la última *keyframe* o inicialización (asegurando que el anterior mapeado haya terminado), y que, además, la nueva vista contenga más de un 10% de puntos de interés que no aparezcan en los *frames* empleados para obtener su posición. De esta forma, trata de introducir *keyframes* lo más rápido posible, para, posteriormente, eliminar los que ofrezcan información redundante empleando un grafo de *covisibilidad*, en el que los *keyframes* están conectados con nexos cuyo peso es función del número de correspondencias entre ambos.

Hay que destacar que, para optimizar todavía más la técnica, se propone introducir un *mapa local*, de forma que en cada nuevo *frame* se realiza el *tracking* únicamente con los *keyframes* que contienen la información sobre el entorno de la cámara en ese momento. Para ello, se usa información elaborada con el grafo de *covisibilidad*.

Además, en ORB-SLAM se propone el uso de un tercer *thread* para el cierre de bucles, con el uso de otro grafo (*esencial*) para detectar su presencia. Dicho grafo también es empleado, junto con un sistema de BoW (*Bag of Words*), que agrupa los puntos de interés en *clusters*, de forma que sean fáciles y rápidos de identificar para relocalizar la cámara si se pierde el *tracking*.

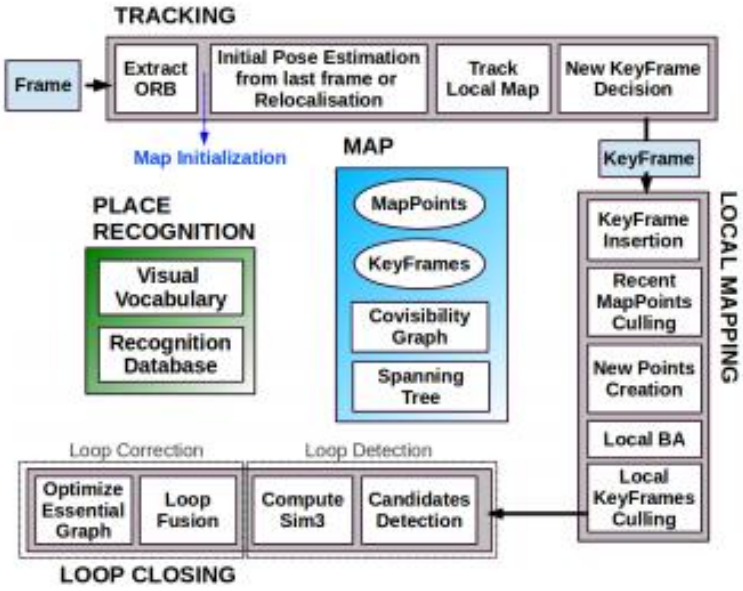

Figura 4.5: Tracking y Mapping. *Separación en hilos propuesta en ORB-SLAM* [35]

Para terminar, es posible aplicar técnicas de *machine learning* más avanzadas que la BoW para mejorar el desempeño del algoritmo, como el uso de redes neuronales entrenadas durante el funcionamiento para obtener una relocalización mucho más eficaz, propuesto en [39]. Dichas técnicas todavía no han alcanzado un rendimiento lo suficientemente rápido como para ser aplicadas en tiempo real, pero son objeto de numerosas investigaciones actualmente en el campo de la visión artificial, por lo que es de esperar que en un futuro no muy lejano se incorporen estrategias de este tipo para mejorar el desempeño de sistemas SLAM.

Otra conclusión derivada del análisis reside en que la triangulación tiene un coste computacional realmente bajo. Por lo tanto, otra propuesta que, a pesar de que aumenta notablemente el consumo de memoria puede mejorar en gran medida los resultados visuales de la reconstrucción sería el uso de dos mapas en paralelo. Un primer mapa construido a partir de las *keyframes* que se use para el *tracking*, junto a otro mapa más denso, resultado de obtener las correspondencias del *frame* actual con el *keyframe* más cercano y llevar a cabo su triangulación. Esta aproximación sería posible para un sistema en el que se disponga de otro *thread* para la realización de esta tarea.

Capítulo 5
Conclusión

5.1 Conclusiones del trabajo

En este trabajo se ha analizado la situación actual de las técnicas de reconstrucción tridimensional, presentando el estado del arte en la materia, y las diferentes alternativas empleadas habitualmente. Se ha comentado la importancia de la técnica de SfM (*Structure from Motion*), con sus principales ventajas y aplicaciones, realizando posteriormente un análisis detallado del método de SfM secuencial. Dicho análisis se ha realizado presentando técnicas avanzadas y parte del estado del arte en cada paso, con todos los instrumentos necesarios para desarrollar un sistema de SfM completo.

También se ha realizado una implementación de un sistema SfM como el descrito en el estudio del algoritmo, detallando los pormenores de dicha implementación. Una vez desarrollada, se lleva a cabo un análisis de la calidad de la reconstrucción, el rendimiento de cada técnica implementada, y de su escalabilidad. Para terminar, se presentan las mejoras a realizar en el algoritmo fruto del análisis realizado, con el fin de servir como base para futuras implementaciones de de SfM orientado a un funcionamiento en tiempo real en sistemas embebidos. De la realización de este trabajo, se extraen cuatro principales conclusiones:

1. La técnica de *Structure from Motion*, así como las técnicas vSLAM inspiradas en ella son uno de los temas candentes en visión artificial, siendo objeto de continuas innovaciones y gran cantidad de esfuerzos para mejorar los algoritmos existentes. Esto se debe a su gran cantidad de potenciales aplicaciones y las enormes posibilidades de un sistema SfM robusto y capaz de funcionar en tiempo real.

2. Los pasos más importantes del algoritmo son la detección de puntos de interés y correspondencias. Como se ha comprobado en la evaluación, son las técnicas con mayor impacto tanto en la calidad de la reconstrucción como en el rendimiento. El número de

puntos de interés detectados en cada imagen condiciona de forma muy importante el rendimiento, y la calidad de la reconstrucción depende de que las correspondencias de puntos de interés entre imágenes sean determinadas correctamente.

3. El principal cuello de botella del algoritmo es la detección de correspondencias entre imágenes cuando ya existen puntos reconstruidos. Esto es debido a la cantidad de comparaciones que se han de realizar y que crecen de manera no lineal con el número de imágenes procesadas. Por ello, no es escalable tener en cuenta todas las imágenes para reconstruir.

4. Para elaborar un sistema SfM capaz de operar en tiempo real y escalar acorde a la cantidad de imágenes a procesar que ello conlleva se ha de emplear aproximaciones como la separación del algoritmo en dos hilos (*tracking* y *mapping)*. Esta aproximación mapea únicamente los *keyframes* (imágenes que contienen nueva información sobre el entorno), reduciendo el número de imágenes cuyos puntos se triangulan en el espacio. De esta forma, con cada nueva imagen se ha de obtener un número de correspondencias mucho menor (se buscan solamente en los *keyframes* en lugar de en todas las imágenes), reduciendo en gran medida el uso de memoria y el tiempo de cómputo.

Bibliografía

[1] Hartley, R; Zisserman, A., *Multiple view geometry in computer vision*, 2nd ed. Cambridge: Cambridge University Press, 2005.

[2] Robertson, D.P.; Cipolla, R. *Structure from Motion*, In Varga, M., editors, *Practical Image Processing and Computer Vision*, John Wiley, 2009.

[3] Baggio, Daniel Lélis. *Mastering OpenCV with practical computer vision projects*. Packt Publishing Ltd, 2012.

[5] Longuet-Higgins, H. C. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133-135, 191

[6] Mountney, Peter; Yang, Guang-Zhong. Motion compensated SLAM for image guided surgery. *Medical Image Computing and Computer-Assisted Intervention*–MICCAI 2010, 2010, p. 496-504.

[7] Smith, Randall C.; Cheeseman, Peter. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 1986, vol. 5, no 4, p. 56-68.

[8] Cadena, Cesar, et al. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 2016, vol. 32, no 6, p. 1309-1332.

[9] Malisiewicz, Tomasz (Wednesday, January 13, 2016). *The Future of Real-Time SLAM and Deep Learning vs SLAM* [Online]. Available: http://www.computervisionblog.com/2016/01/why-slam-matters-future-of-real-time.html

[10] Snavely, Noah; Seitz, Steven M; Szeliski, Richard. Photo Tourism: Exploring image collections in 3D. *ACM Transactions on Graphics* (Proceedings of SIGGRAPH 2006), 2006.

[11] Agarwal L, Sameer, et al. Building rome in a day. *Communications of the ACM*, 2011, vol. 54, no 10, p. 105-112.

[12] Nodjimbadem, Katie (March 2016) *The Heroic Effort to Digitally Reconstruct Lost Monuments* [Online]. Available: http://www.smithsonianmag.com/history/heroic-effort-digitally-reconstruct-lost-monuments-180958098/

[13] Autodesk ReMake Blog (September 28, 2016) *Sensing Shipwrecks* [Online]. Available: https://autodeskremake.squarespace.com/blog/2016/8/25/sensing-shipwrecks

[14] Map-Tk software. https://github.com/Kitware/maptk

[15] Lucieer, Arko; Jong, Steven M. de; Turner, Darren. Mapping landslide displacements using Structure from Motion (SfM) and image correlation of multi-temporal UAV photography. *Progress in Physical Geography*, 2014, vol. 38, no 1, p. 97-116.

[16] Autodesk ReMake Blog (January 03, 2017) *No bones about it* [Online]. Available: https://autodeskremake.squarespace.com/blog/2017/1/3/reality-capture-in-biomedicine

[17] Engel, Jakob; Schöps, Thomas; Cremers, Daniel. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*. Springer, Cham, 2014. p. 834-849.

[18] Mur Artal, Raúl, "Should we still do sparse-feature based SLAM?" In *ICCV*, Santiago, Chile, 2015.

[19] Moulon, Pierre; Leotta, Mark, "SfM Pipelines" In *CVPR*, Boston, MA, 2015

[20] Harris, Chris; Stephens, Mike. A combined corner and edge detector. In *Alvey vision conference*. 1988. p. 10.5244.

[21] Lowe, David G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004, vol. 60, no 2, p. 91-110.

[22] Wu, Changchang. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)(2007). URL *http://cs. unc. edu/~ ccwu/siftgpu*, 2011.

[23] Rublee, Ethan, et al. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on. IEEE*, 2011. p. 2564-2571.

[24] Rosten, Edward; Drummond, Tom. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, 2006, p. 430-443.

[25] Rosten, Edward; Porter, Reid; Drummond, Tom. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 2010, vol. 32, no 1, p. 105-119.

[26] Calonder, Michael, et al. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, 2010, p. 778-792.

[27] Pascal VOC project. http://host.robots.ox.ac.uk/pascal/VOC/

[28] Hartley R.I. (1992) Estimation of relative camera positions for uncalibrated cameras. In *Sandini G. (eds) Computer Vision* — ECCV'92. ECCV 1992. Lecture Notes in Computer Science, vol 588. Springer, Berlin, Heidelberg

[29] Hartley, Richard I.; Sturm, Peter. Triangulation. *Computer vision and image understanding*, 1997, vol. 68, no 2, p. 146-157.

[30] Gao, Xiao-Shan; Hou, Xiao-Rong; Tang, Jianliang; Cheng, Hang-Fei. Complete Solution Classification for the Perspective-Three-Point Problem". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003. 25 (8): 930–943

[31] Lepetit, V.; Moreno-Noguer, F.; Fua, P. EPnP: An Accurate O(n) Solution to the PnP Problem, in *International Journal Of Computer Vision*, 2009, vol. 81, p. 155-166.

[32] Zach, Christopher. Robust bundle adjustment revisited. In *European Conference on Computer Vision*. Springer, Cham, 2014. p. 772-787.

[33] Point Cloud Library. http://pointclouds.org/

[34] Lourakis, Manolis IA; Argyros, Antonis A. SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software* (TOMS), 2009, vol. 36, no 1, p. 2.

[35] Mur-Artal, Raul; Montiel, J. M. M.; Tardós, Juan D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 2015, vol. 31, no. 5, pp. 1147-1163.

[36] Wu, Changchang. Towards linear-time incremental structure from motion. In *3DTV-Conference, 2013 International Conference on. IEEE*, 2013. p. 127-134.

[37] Agarwal, Sameer, et al. Ceres solver. 2012.

[38] Klein, Georg; Murray, David. Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on. IEEE*, 2007. p. 225-234.

[39] Chen, Zetao, et al. Convolutional neural network-based place recognition. *arXiv preprint* arXiv:1411.1509, 2014.

[40] Papert, Seymour A. The summer vision project. *Memo AI group*. 1966.

[41] Fergus, Rob (2012). Lecture 6: Multi-view Stereo & Structure from Motion [Online]. Available: http://cs.nyu.edu/~fergus/teaching/vision_2012/6_Multiview_SfM.pdf