UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Corrección de la odometría visual basada en la detección de cierre de lazo"

Lidia Caramazana Zarzosa

2015

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Corrección de la odometría visual basada en la detección de cierre de lazo"

Alumno: Lidia Caramazana Zarzosa

Director: Luis Miguel Bergasa Pascual

Tribunal:

Presidente: D. Manuel Ocaña Miguel.

Vocal 1º: Dña. Cristina Losada Gutierrez.

Vocal 2º: D. Luis Miguel Bergasa Pascual.

Calificación: .....

Fecha: .....

Nuestra recompensa se encuentra en el esfuerzo y no en el resultado...
un esfuerzo total es una victoria completa.
Mahatma Gandhi.

Agradecimientos

Este Trabajo de Fin de Grado supone el broche final a cuatro años de dedicación y esfuerzo. Quiero agradecer enormemente a todas las personas que me han apoyado durante este último curso y han hecho que me fuera más fácil cumplir todas las metas propuestas. En primer lugar, agradecer a mi tutor Luis M. Bergasa por haberme dado la oportunidad de trabajar con él y su grupo de investigación *Robesafe*, por haber confiado en mí al asignarme este proyecto y por haberme ayudado en todo lo que he necesitado. Por supuesto, dar las GRACIAS (con mayúsculas) a mi compañero Roberto Arroyo, por haberse preocupado por este proyecto como si fuera suyo, por cuidar cada uno de sus detalles y por haber estado siempre dispuesto a emplear su tiempo en ayudarme. Gracias, también, a mis amigos, en especial a Alejandra, Eva y Joaquín, por estar siempre ahí, y a Gonza, cuyo apoyo ha sido imprescindible para llegar hasta aquí. Por último, dar las gracias a mi familia: a mi hermano, con el que he crecido y he compartido todo, a mi padre, porque sin él no sería hoy lo que soy y a mi madre, por ser la persona más importante y mi principal apoyo en todo.

Índice general

Índice de figuras

Índice de tablas

Resumen

En el ámbito de la robótica y la automoción resulta de interés conocer la posición que ocupa el robot en todo momento, así como la trayectoria que éste describe, haciendo uso de los sensores a bordo del mismo, para lo cual existen ya en la actualidad diferentes métodos.

Este proyecto se focaliza en el uso de cámaras como sensores de percepción del entorno y propone una metodología que permita realizar una odometría visual robusta, aplicando técnicas de corrección basadas en la detección de cierres de lazo en situaciones de localización a largo plazo.

Para ello, se va a llevar a cabo una mejora metodológica de algunas técnicas clásicas de visión artificial y se implementarán nuevos algoritmos, con el fin de corregir la deriva que implica el uso de la odometría visual en la estimación del recorrido realizado por un agente.

Se pretende obtener una estimación precisa de la posición, orientación y trayectoria seguida por un vehículo, a partir del análisis de una secuencia de imágenes adquiridas a través de un sistema estéreo de cámaras que lleva a bordo, sin tener un conocimiento previo del espacio físico en el que se encuentra, y aplicando las técnicas de corrección necesarias una vez que el vehículo recorra una zona previamente visitada.

Palabras clave: odometría visual, cierre de lazo, corrección de pose, AKAZE, LIBVISO, ABLE.

Abstract

An essential requirement in the field of robotics and automation is to know the position of a mobile robot along the time, as well as the trajectory that it describes by using on-board sensors. Nowadays, several methods exist for accomplishing this goal.

In this work, we propose a novel approach focused on the use of cameras as perception sensors of the environment, that allows to perform a robust visual odometry, where correction algorithms based on loop closure detection are applied for localization in long-term situations.

In order to satisfy the previous conditions, we carry out a methodological improvement of some classic computer vision techniques. In addition, new algorithms are implemented with the aim of correcting the drift produced in the visual odometry estimation along the traversed path.

The main objective is to obtain an accurate estimation of the position, orientation and trajectory followed by a vehicle. Sequences of images acquired by an on-board stereo camera system are analyzed without any previous knowledge about the real environment. Due to this, correction techniques are needed when a place is revisited by the vehicle.

Key words: visual odometry, loop closure detection, pose correction, AKAZE, LIBVISO, ABLE.

Resumen extendido

La motivación principal de este proyecto es la localización robusta de un vehículo en entornos exteriores utilizando únicamente cámaras como sistema de percepción. En concreto, se propone corregir el error acumulado en la estimación de la trayectoria recorrida por un móvil que proporciona la odometría visual. Para ello, se van a emplear en conjunción técnicas para la detección de cierre de lazo.

La odometría visual estima la variación en la rotación y traslación entre dos poses consecutivas de un móvil (se denomina pose al conjunto de la orientación y posición del robot) a partir de la información extraída de una secuencia de imágenes, sin tener conocimiento previo del entorno. Estimada esta variación, se conoce el movimiento relativo realizado por el robot durante un periodo de tiempo determinado. Para ello, se procesan las imágenes del entorno, capturadas por un sistema de cámaras a bordo del vehículo, siendo la única información accesible y, por tanto, la base de esta investigación.

En este proyecto, se hace uso de un sistema de cámaras estéreo, que generalmente permite obtener mejores resultados, pues la cantidad de información que se extrae es mayor que haciendo uso de una sola cámara.

Como se pretende realizar una corrección de la trayectoria recorrida a lo largo de un periodo de tiempo determinado, hay que tener en cuenta que en un escenario de localización visual a largo plazo el consumo de recursos es crítico. Además, se debe considerar la variabilidad de la apariencia visual de los elementos de la escena, ya que el entorno está expuesto a cambios físicos y ambientales (cambio en la iluminación, en la apariencia de los árboles, edificios, etc), y a la variación de elementos móviles presentes en el área de interés (vehículos estacionados, peatones, etc). Por ello, entre otros factores, es clave la elección del algoritmo de extracción de puntos de interés y el empleo de técnicas de corrección basadas en la detección de cierres de lazo. Además, también son necesarias técnicas de emparejamiento (*matching*) robusto para evitar *outliers* o falsas correspondencias que produzcan, por una parte, una mala reconstrucción del sistema estéreo y, por otra, un *tracking* o seguimiento de puntos erróneos.

La odometría visual no es una forma completamente fiable de localizar al robot, ya que se basa en fuentes de información que no son totalmente precisas. Aunque se utilicen modelos matemáticos más o menos complejos, se trabaja sobre imágenes de lugares que están sujetos a cambios o, en otros casos, sobre medidas tomadas con sensores, que tampoco están exentas de errores, y que, por tanto, conducen a una estimación de la posición del móvil que va acumulando una deriva de forma incremental. Esto se traduce en que, en trayectorias largas, la estimación de la posición por odometría deja de ser totalmente fiable, debido al ruido que se va integrando progresivamente en la predicción. Como consecuencia, se necesita explotar la información con la que se cuenta para corregir el resultado obtenido inicialmente y conseguir un sistema de localización que funcione de forma correcta.

Para ello, se necesita información adicional o complementaria que permita corregir esa deriva

acumulativa. En este caso, esta corrección se realiza en base a una restricción en la posición del robot dada por la detección de cierre de lazo. Esta detección es clave para conseguir una mayor robustez en los algoritmos de odometría visual y consiste en determinar cuándo el robot vuelve a recorrer una zona que ya había visitado previamente. Esto permite mejorar la estimación de la pose actual del robot y de su localización global. Por tanto, la finalidad del empleo de diferentes tipos de algoritmos de corrección es que la deriva que sufre la odometría visual a lo largo del tiempo pueda ser mitigada.

A continuación, se introducen los aspectos más destacados de este proyecto, que se van a describir más en detalle en capítulos posteriores:

1. Comprensión de las principales referencias del estado del arte en cuanto a odometría visual y detección de cierre de lazo.

   Aunque el término de odometría visual se acuñó en el año 2004 en un trabajo de Nister [1], la primera vez que se abordó el problema de estimar el movimiento de un robot en base a las imágenes captadas por un sistema de cámaras fue hace ya más de tres décadas, gracias a los estudios de Moravec [2] en el año 1980. Por tanto, a pesar de tratarse de un tema en proceso de investigación actualmente, las publicaciones y artículos existentes en el estado del arte proponiendo mejoras y nuevas técnicas son numerosos.

2. Implementación basada en la teoría de LIBVISO [3].

   Se ha decidido basar nuestro método en las librerías de LIBVISO, desarrolladas por Andreas Geiger, debido a las numerosas funcionalidades que ofrece, como una odometría visual estéreo o monocular, el *tracking* de puntos característicos (cámara estática y objetos en movimiento), el flujo óptico (cámara en movimiento y objetos estáticos) o el método SFM (*Structure From Motion*).

   Como en cada uno de los algoritmos de odometría visual, en LIBVISO se realizan todas las fases necesarias para la estimación de la trayectoria seguida por el vehículo, que van a ser explicadas en detalle en capítulos posteriores: adquisición de imágenes a través de un sistema de cámaras mono o estéreo previamente calibradas, búsqueda de un conjunto de datos en la imagen (puntos de interés) que la caractericen de la mejor forma posible (detección), extracción de información relevante y distintiva de cada punto de interés y de la región que lo rodea (descripción), emparejamiento entre los descriptores de cada imagen (*matching*), eliminación de *outliers* o puntos erróneos mediante el método RANSAC, y por último, estimación del movimiento mediante geometría epipolar y almacenamiento de información de interés (matrices fundamental y esencial).

   a) Se llevará a cabo el testeo del algoritmo con diferentes secuencias del dataset de KITTI [4] para la obtención de mapas, comparando el resultado de la odometría visual de Geiger con el *Ground Truth* o mapa obtenido a partir de las medidas del GPS, mostrando así de forma gráfica la deriva producida por la odometría.

3. Implementación basada en la teoría de ABLE [5] [6] [7].

   En [6] se propone un nuevo enfoque para la detección de cierre de lazo en imágenes estéreo, basado en el uso de descriptores binarios (cada elemento del descriptor es codificado en un bit). LDB [8] [9] permite realizar una descripción robusta de la imagen, basada en la intensidad y gradiente de ésta. Además, en [6] se propone también el uso de la información de disparidad entre imágenes, a través del descriptor binario D-LDB. La adición de esta información en el proceso de descripción permite reducir algunos problemas típicos en el

reconocimiento visual de lugares como el *perceptual aliasing*, o solapamiento perceptual. De esta forma, la propiedad de distintividad de LDB es superior a la de otros descriptores.

Esta técnica mejora los resultados obtenidos por otros algoritmos conocidos dentro del estado del arte de detección de cierres de lazo, como son FAB-MAP [10] o WI-SURF [11], además de tener un menor coste computacional.

a) Se testeará el algoritmo de detección de cierre de lazo, ABLE, con secuencias del dataset de KITTI de odometría visual.

b) Se implementará el descriptor binario AKAZE [12] en el algoritmo de ABLE para la obtención de resultados comparativos mediante curvas de rendimiento y otras gráficas con respecto a: SIFT [13], SURF [14], FREAK [15], LDB, BRISK [16], BRIEF [17], HOG [18] y ORB [19].

   Este punto representa una de las principales contribuciones de este proyecto. Se trata de un nuevo algoritmo de detección y descripción en espacios de escala no lineal, basado en el uso del filtro de difusión no lineal. Los métodos anteriores para detectar y describir las imágenes en espacios con diferentes niveles de escala, hacen uso de un espacio de escala gaussiano. El desenfoque gaussiano no respeta los límites naturales de los objetos, suavizando en la misma medida tanto el ruido como los detalles de la imagen, reduciendo así la precisión en la localización. En cambio, AKAZE realiza la detección y descripción de características 2D mediante el filtrado de difusión no lineal. De esta manera, se puede hacer un suavizado local y adaptativo de la imagen, reduciendo el ruido y conservando los bordes naturales de los objetos al mismo tiempo, obteniendo así mejores resultados.

c) Se integrará el descriptor AKAZE en el código de ABLE, con el fin de obtener una comparativa entre una descripción global y una descripción jerárquica o basada en el uso de un grid.

4. Integración del algoritmo para la detección de cierre de lazo en el código de odometría visual.

   Este punto representa la segunda contribución principal de nuestra investigación. Para ello, se llevarán a cabo los siguientes puntos:

   a) Detección de cierres de lazo en los mapas obtenidos con LIBVISO, basada en las matrices de similitud y distancias de *Hamming* calculadas por el algoritmo de ABLE.

   b) Almacenamiento de la información necesaria para una posterior corrección de la odometría: descripción, matrices de rotación y traslación, poses, distancias de *Hamming* y su frame correspondiente, matrices de similitud y *matching* entre descriptores.

   c) Empleo de un método de eliminación de *outliers* a la hora de concluir qué frames ya han sido visitados: RANSAC.

   d) Obtención de resultados gráficos de la detección de cierre de lazo basada en los puntos anteriormente citados.

5. Integración del algoritmo de actualización y corrección de la trayectoria.

   Tercera y última contribución de nuestro trabajo al tema de la odometría visual y objetivo final del mismo.

   Como ya se ha comentado, la odometría visual no es una forma completamente fiable de localizar al robot, ya que los cálculos se basan en fuentes de información que no son precisas. Por tanto, proporciona una forma sencilla de estimar la posición del robot, pero que va acumulando errores, tanto sistemáticos como no sistemáticos. Para solucionarlo

se proponen distintos enfoques para corregir la estimación y conseguir así un sistema de localización del móvil que funcione correctamente.

a) Corrección basada en la detección de la deriva incremental usando cierres de lazo. Una vez detectada la zona que representa el cierre de lazo, se realiza una reasignación de las poses: las nuevas se corresponden con las previamente visitadas, y las anteriores a éstas son corregidas hacia atrás, de manera proporcional a la desviación media calculada en la zona del bucle, ya que la deriva acumulada es menor en las primeras poses que en las últimas.

b) Corrección mediante el algoritmo de optimización GraphSlam [20]. Se trata de un algoritmo de optimización de la trayectoria obtenida mediante odometría visual que corrige el recorrido realizado por el robot partiendo de la zona de cierre de lazo previamente detectada. Este método resuelve por completo el problema de SLAM (Mapeado y Localización Simultáneos) [21] [22]. Para ello, representa el problema como un grafo que conduce a una suma de restricciones no lineales entre las poses que se introducen, reduciendo así el grafo a sus nodos más significativos. Dichas restricciones son optimizadas para proceder posteriormente al remapeado de la trayectoria, en base al criterio de máxima similitud y un conjunto de poses correspondientes.

c) Corrección mediante el algoritmo de optimización Levenberg-Marquardt. Se trata de un proceso iterativo de convergencia a través de la técnica de los mínimos cuadrados ponderados, basado en el método de optimización Newton-Gauss.

6. Verificación de los resultados en el dataset de KITTI y validación de la propuesta.

7. Conclusiones generales y planteamiento de trabajos futuros.

Parte I

Memoria

Capítulo 1

Introducción

En los últimos años, la estimación de la posición de un vehículo a través de sistemas basados en visión artificial es una cuestión clave en el mundo de la robótica, y más concretamente en el desarrollo de sistemas inteligentes de transporte. Esto es debido, principalmente, a que estos sistemas de visión artificial se caracterizan por un menor coste y una mayor velocidad que otros sensores comúnmente utilizados en este ámbito, como son los basados en ondas de radio o satélite, lásers y ultrasonidos.

La estimación de la posición global de un móvil (robot o vehículo) sin tener información previa del entorno en el que se encuentra, es también un requisito en muchas otras aplicaciones como son la detección de obstáculos, la conducción autónoma, o el mapeado y localización simultáneos (SLAM). Esto es precisamente el objetivo de los denominados sistemas de odometría visual, que realizan una estimación de la posición y orientación de un vehículo a partir del análisis de una secuencia de imágenes adquiridas por una cámara o sistema de cámaras que lleva a bordo, sin tener conocimiento previo del entorno. Dicho de otra manera, la odometría visual determina la variación en la rotación y traslación entre dos poses consecutivas del móvil, para así calcular cuál ha sido el movimiento realizado por este. Como el movimiento del sistema de cámaras es solidario con el movimiento del vehículo, a esta técnica también se le conoce como *ego-motion* (movimiento propio).

Atendiendo a las consideraciones previamente realizadas sobre odometría visual, se pueden ya vislumbrar algunas de sus importantes ventajas. Por una parte, estas técnicas se presentan como un buen complemento al sistema de posicionamiento global, GPS. Este sistema está fuertemente implantado en nuestra sociedad gracias a su fiabilidad y precisión en la localización global de un vehículo. Sin embargo, necesita recibir la señal radio procedente, de al menos, cuatro satélites, algo que no es posible en determinados lugares como un túnel, una zona urbana con edificios altos, o un punto negro de cobertura GPS, sitios donde el sistema de posicionamiento basado en visión artificial permanecería activo. Además, hay que tener en cuenta que la exactitud de los sistemas GPS no es siempre la deseada, sobretodo en entornos urbanos, donde la interferencia multicamino producida por los edificios puede empeorar notablemente la precisión del posicionamiento.

Aparte de las ventajas ya comentadas para sistemas de transporte terrestres, los vehículos aéreos no tripulados (UAVs) y los vehículos submarinos hacen también uso de las técnicas de odometría visual como suplemento al posicionamiento GPS, ya que la señal procedente de estos sistemas tampoco puede ser recibida convenientemente en entornos aéreos o marítimos. A su vez, existe un gran número de experiencias utilizando odometría visual en espacios interiores con resultados bastante positivos, lo que es de gran utilidad en sistemas de ayuda a personas con discapacidad, gracias a la detección de puertas, pasillos o paredes, o a la hora de generar

mapas de edificios, como hospitales o grandes centros comerciales.

Se han desarrollado también técnicas de visión por ordenador para mejorar la percepción del entorno en personas invidentes. Estos sistemas de navegación emplean comúnmente técnicas de odometría visual y utilizan cámaras estéreo que proporcionan una percepción de profundidad, empleando mapas de disparidad. También permiten la generación de una representación táctil del entorno, o una estimación simultánea de la estructura y el movimiento, haciendo uso de un sistema de cámaras fijado a un casco, que procesa las imágenes en un ordenador, como se muestra en la Figura 1.1.

Figura 1.1: Gafas biónicas basadas en visión artificial para personas con problemas de visión.

Existen incluso trabajos de odometría visual centrados en la creación de tecnología para la navegación autónoma de robots de exploración planetaria, concretamente en Marte. De hecho, los primeros trabajos de odometría visual fueron motivados por el programa de exploración *Mars Rovers*. En las primeras exploraciones se hacía uso de los clásicos sensores de odometría (véase Figura 1.2), que utilizan datos provenientes de los encoders de las ruedas del robot, las cuales tendían a resbalarse sobre la arena suelta, especialmente en superficies inclinadas, e incluso se hundían en terrenos arenosos, provocando en algunos casos una desviación de la ruta trazada sin cumplir el objetivo propuesto.

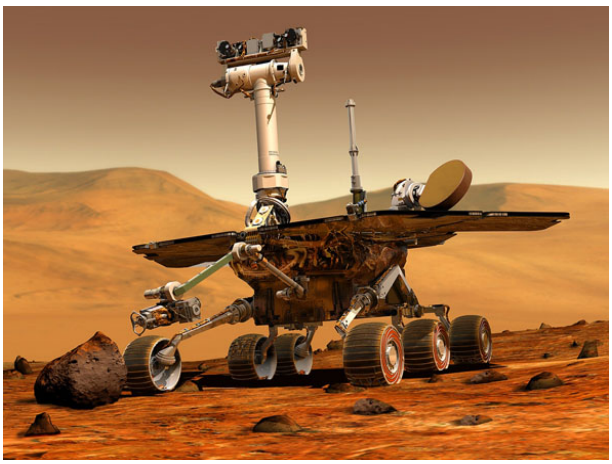


Figura 1.2: Sistema de odometría basado en encoders de las ruedas del robot, *"Curiosity"*.

Para resolver el inconveniente anteriormente comentado, la NASA incorporó en sus robots un algoritmo de odometría visual estereoscópica, de forma que el movimiento seguido por el robot

al desplazarse sobre la superficie marciana es estimado a partir del análisis de correspondencias entre puntos característicos encontrados en pares consecutivos de imágenes.

El problema tratado en nuestro proyecto es, por tanto, un tema de investigación clave y que se encuentra en pleno auge dentro del campo de la visión artificial y la robótica, ya que una corrección de la deriva en los resultados obtenidos mediante odometría visual, supondrá numerosas ventajas y avances en todas las aplicaciones citadas, y la consiguiente mejora de la calidad de vida de las personas.

1.1. Estado del Arte

1.1.1. Odometría Visual

En los últimos años se han desarrollado diferentes métodos para calcular la pose del robot, que van desde la estimación basada en marcadores visuales [23], [24], hasta sensores inerciales o GPS. Sin embargo, en la actualidad existe una tendencia cada vez mayor a tomar como base las imágenes capturadas por un sistema de cámaras que el robot lleva a bordo, que es precisamente el objetivo de la odometría visual. Este problema supone la base para numerosas aplicaciones en robótica y visión por ordenador, como puede ser la realidad aumentada o la fotogrametría [25].

En la estimación de la localización de un vehículo teniendo como única fuente de información las imágenes del entorno en el que se encuentra, existen principalmente tres opciones, dependiendo del grado de conocimiento que se tenga sobre dicho entorno:

- Los primeros métodos que se desarrollaron se basaban en la existencia de objetos conocidos en la imagen, llamados marcadores. La implementación más conocida de este tipo de *tracking* es la librería ArToolkit de la Universidad de Washington [26].
- El segundo grupo se basa en el estudio de la geometría de los objetos situados en la escena. Teniendo en cuenta que la posición de una cámara respecto a un objeto es la inversa de la posición del objeto respecto a la cámara, se pueden utilizar algoritmos de *tracking* de objetos para aplicaciones de realidad aumentada. En [27] se muestra un resumen muy amplio de las técnicas utilizadas en este tipo de *tracking*.
- En el tercer grupo se incluyen los algoritmos de tipo SFM (*Structure From Motion* [1]). Estos métodos se basan en el efecto parallax para calcular el movimiento de la cámara a lo largo de la secuencia.

Por otro lado, se puede establecer otra clasificación de los algoritmos de odometría visual: aquellos que realizan un *matching* o emparejamiento de puntos característicos entre pares de frames consecutivos [28], [29], [30] y aquellos que realizan un proceso de *tracking* sobre toda la secuencia de imágenes [31], [32], [33].

A la hora de capturar las imágenes que serán procesadas posteriormente, se puede hacer uso de un sistema de cámaras monocular [24] o estéreo. En la mayoría de los casos, el uso de un sistema estéreo permite obtener mejores resultados, pues la cantidad de información de la que se dispone es mayor [24]. Además, el problema de la ambigüedad en la escala que presentan los sistemas monoculares queda subsanado [34].

En los últimos años se están desarrollando métodos que combinan la odometría visual con el uso de otros sensores, con el fin de aumentar la precisión en sus resultados y reducir el error, un problema inherente a cualquier método incremental de estimación de la trayectoria.

Dornhege [32] añade en su método información de la velocidad, orientación y fuerza del robot, medidas mediante una IMU (Unidad de Medición Inercial). Por otro lado, Agrawal [34], [35], [31] hace uso de la información proporcionada por el GPS y los encoders de las ruedas, combinando así distintos sensores para obtener mejores resultados en la odometría visual que realiza.

En comparación con la estrategia propuesta en [31], donde se combina la odometría visual basada en el método de *bundle adjustment* con los datos obtenidos mediante GPS e IMU, el método LIBVISO [3] de Andreas Geiger realiza una estimación de las poses del robot basándose exclusivamente en información visual sobre el entorno. Esto hace que la complejidad computacional sea mucho menor y pueda ser utilizado en aplicaciones en tiempo real, razón por la que se ha elegido basar este proyecto en el concepto propuesto por este algoritmo.

1.1.2. Detección de cierre de lazo

Se han desarrollado diversos algoritmos para la detección de cierres de lazo mediante visión artificial, sobre todo a partir de la introducción de FAB-MAP (*Fast Appearance-Based MAPping*) [10]. Esta técnica está basada en lo que en visión se denomina *bag-of-words* (bolsa de palabras). FAB-MAP representa una imagen como un conjunto (o bolsa) de descriptores locales (palabras), que pertenecen a un "diccionario" previamente definido. De esta manera se obtiene un método invariante a cambios de iluminación y perspectiva.

Sin embargo, BRIEF-Gist demostró en sus resultados que un algoritmo de detección de cierre de lazo no necesita realizar una descripción compleja de las imágenes, como propone FAB-MAP. Esta teoría se refuerza en [36], donde se procesan imágenes de baja resolución y se consigue realizar un reconocimiento visual efectivo. Además, el hecho de representar las imágenes con una baja cantidad de bits (mediante descriptores binarios), permite reducir el tiempo de cómputo, ya que el proceso de *matching* de puntos característicos se realiza de una forma mucho más rápida, como se dedujo en [37].

Los descriptores binarios han ido incluyendo mejoras en sus características desde la aparición de BRIEF [17]. Descriptores como ORB [19] y BRISK [16] añaden en sus métodos, la invarianza frente a cambios de rotación y escala respectivamente.

Uno de los descriptores binarios más novedosos y atractivos es LDB [8] [9]. Éste no sólo se basa en la intensidad de las imágenes, como hace BRIEF, sino que, además, calcula el gradiente de las mismas, lo que permite obtener una descripción de la imagen mucho más precisa y realizar el proceso de *matching* en un tiempo menor que el empleado por BRIEF. Además, recientemente se ha presentado una nueva mejora sobre LDB, al añadir a este método la invarianza frente a cambios en la rotación y escala (M-LDB [38])

Por otra parte, varios trabajos del estado del arte utilizan información tridimensional de la escena para mejorar el rendimiento de los métodos de navegación basados en visión artificial, como en [39], [40], [11]. En [11], WI-SURF combina la descripción global de la imagen con información 3D procedente de un láser. FAB-MAP incorporó también en su algoritmo información espacial del entorno. Sin embargo, a pesar de que se obtiene una detección de cierre de lazo efectiva, esta asociación de información tridimensional con características visuales de la escena conlleva un alto coste computacional.

Finalmente, el método ABLE [5] [6] [7] propone una descripción binaria de la imagen, que proporciona una buena detección de cierre de lazo con bajos requerimientos de tiempo y memoria. Además, se integra la información 3D de la escena en el propio descriptor (D-LDB) a partir del cálculo de la disparidad entre imágenes. Por todo lo expuesto, en este proyecto se realiza una implementación basada en este último método.

1.2. Objetivos

Los objetivos principales de este Trabajo Fin de Grado son:

1. Detección de cierres de lazo realizando una descripción global de las imágenes mediante AKAZE. Obtención de resultados comparativos con los descriptores existentes en el estado del arte: LDB, SIFT, SURF, BRIEF, BRISK, ORB, FREAK, HOG.
2. Integración del método de detección de cierre de lazo, ABLE, en el algoritmo de odometría visual, LIBVISO. Detección de cierres de lazo en los mapas generados por la odometría visual de Andreas Geiger.
3. Corrección de la deriva acumulada en los resultados de la odometría visual. Aplicación de distintas técnicas de corrección, basadas en los cierres de lazo previamente determinados.

Capítulo 2

Método de odometría visual

2.1. Motivación

El objetivo de la odometría visual es la **estimación** de la posición **relativa** de un agente móvil, a partir de la información extraída de una secuencia de imágenes, sin tener conocimiento previo del entorno. En todas las definiciones de odometría aparecen siempre estos dos términos: *estimación* y *relativa*, y a continuación se explica el por qué.

Por una parte, se habla de **estimación** de la posición, y esto es debido a que determinar a ciencia cierta la trayectoria de un robot móvil mediante odometría es sencillamente imposible, puesto que ninguna de las técnicas existentes para calcularla tiene una precisión absoluta.

La única fuente de información que se tiene sobre el robot y su entorno es la secuencia de imágenes que éste ha ido capturando a medida que iba avanzando. Por tanto, aunque se utilicen modelos matemáticos y cálculos más o menos complejos, siempre se trabaja sobre imágenes de lugares, que están sujetos a cambios o, en otros casos, sobre medidas tomadas a partir de sensores, que tampoco están exentas de errores.

Por otra parte, otro término también presente en cualquier definición de odometría visual es **relativa**: aunque todas las posiciones sean relativas a un marco de referencia, en robótica las poses del móvil siempre se referencian a las llamadas coordenadas del mundo. Las posiciones que se dan respecto a este marco de referencia son denominadas, por norma general, posiciones absolutas. Sin embargo, las poses estimadas a partir de la odometría visual no están referenciadas a este sistema, sino que son **relativas** a nuestro punto de inicio en particular, que no tiene por qué ser conocido y que es definido por nosotros mismos, de ahí la expresión "estimación de la posición relativa de un robot", presente en la definición de odometría visual.

2.2. Explicación de la odometría visual

En este capítulo se definen cada una de las fases de las que consta la odometría visual utilizando un sistema estéreo, necesarias para estimar el movimiento del agente. En la Figura 2.1 se muestra un diagrama con los principales bloques de los que consta el proceso y que serán explicados en las siguientes secciones.

La Figura 2.2 muestra el proceso denominado *matching* o emparejamiento de los puntos característicos de la imagen (descriptores), que sirve para representar de forma gráfica el movimiento realizado por los agentes presentes en la escena: a partir de la observación de esta imagen, se

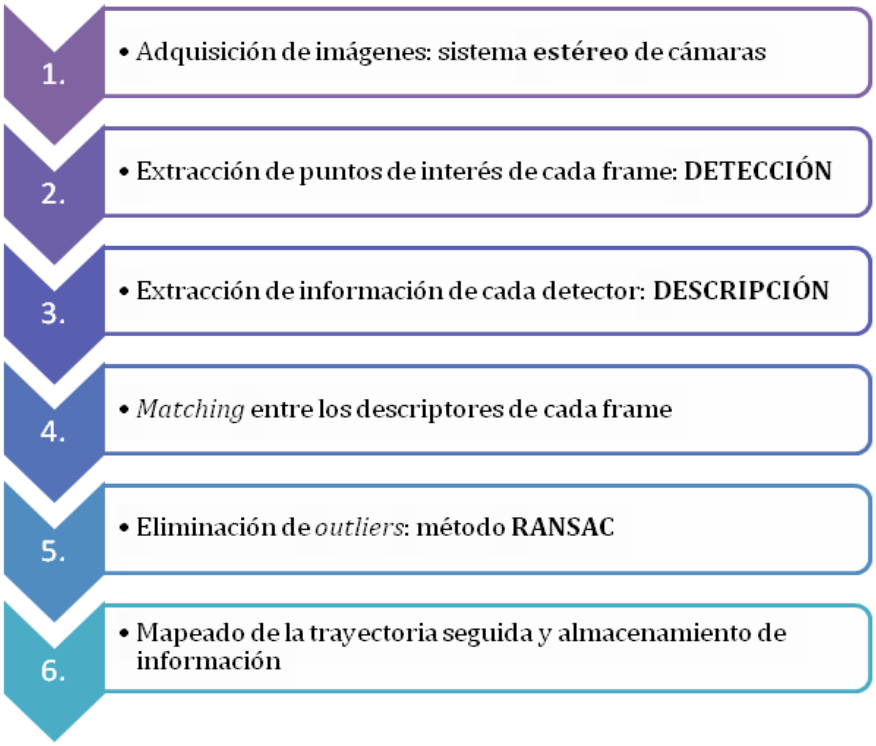


Figura 2.1: Esquema de los principales bloques de la odometría visual.

puede afirmar que los vehículos recorren la carretera de izquierda a derecha, mientras que los peatones cruzan el paso de cebra en la dirección contraria.

Figura 2.2: Representación de *matching* temporal. Imagen obtenida del artículo de Andreas Geiger.

Una vez conocidas las correspondencias entre puntos de interés, es posible estimar un modelo del movimiento. Matemáticamente, la estimación del movimiento se obtiene mediante el cálculo de la matriz esencial, usando geometría epipolar, de la que se extrae tanto la rotación como la traslación producida entre dos poses consecutivas. Esto va a ser explicado en detalle más adelante.

2.2.1. Adquisición de imágenes

Para la adquisición de imágenes se usa un sistema de visión estéreo, pues así se dispone de información tridimensional de la escena y por tanto, se obtienen resultados más estables que haciendo uso de una sola cámara.

Esta primera fase de adquisición de las imágenes es muy importante, ya que es la única fuente de información que va a ser procesada posteriormente. Por ello, la captura de las imágenes debe realizarse de la forma más eficiente posible, para que se disponga del mayor número de recursos posibles.

Se consideran como válidas las imágenes con las siguientes características:

1. Escena mayormente estática: la estimación del movimiento de un agente mediante odometría visual se basa en el hecho de que las variaciones de la posición de los puntos característicos son debidas al movimiento del sistema de cámaras, es decir, del robot. Por lo tanto, si la correspondencia se da entre puntos característicos que pertenecen a un objeto que se encontraba en movimiento a la hora de adquirir las imágenes, la anterior suposición ya no es cierta, y se estimaría un modelo del movimiento de forma errónea. De ahí la necesidad de que el entorno sea lo más estático posible.

2. Imagen con suficiente textura: la detección de puntos característicos en imágenes sucesivas es posible en aquellas que presenten variaciones significativas en cuanto a textura, ya que ésta es la que da carácter e identidad a los objetos presentes en la escena.

3. Solapamiento entre imágenes: la correspondencia entre descriptores se puede realizar si éstos están presentes en el par de imágenes tomadas de forma consecutiva.

4. Buena iluminación.

Las imágenes con las propiedades enumeradas son las óptimas para poder detectar los puntos característicos y poder realizar posteriormente las correspondencias oportunas. Ejemplo de ello se muestra en la Figura 2.3.

Figura 2.3: Ejemplo de una imagen con las propiedades requeridas para realizar la odometría visual. Figura del dataset de KITTI.

2.2.2. Detección de puntos característicos

Las técnicas de extracción de puntos de interés tienen por objeto encontrar en una imagen un conjunto de datos que la caractericen de la mejor forma posible y permitan establecer relaciones entre imágenes. Por tanto, esta es una fase esencial en la detección de cierre de lazo, puesto que la evidencia en la que se basa el método para concluir que el móvil se encuentra en esta situación es que un número significativo de puntos característicos detectados en los frames actuales coincidan con aquellos detectados en imágenes anteriores.

Además, la detección de puntos de interés, también denominados *keypoints* o *feature points*, también es necesaria a la hora de mapear la trayectoria del robot, ya que ésta se realiza en base a la rotación y traslación entre poses consecutivas, información que se extrae tras realizar el emparejamiento de puntos característicos.

La extracción de los *keypoints* permite reducir la información con la que se trabaja, y como consecuencia, el tiempo de cómputo, ya que no se procesan todos los píxeles de la imagen, sino únicamente sus puntos de interés (procesamiento *sparca*).

Los puntos de interés de la imagen suelen ser, comúnmente, esquinas, líneas, regiones o bordes. No existe una regla sobre qué tipos de puntos de interés hay que extraer, pero dependiendo del tipo de imagen conviene detectar unos u otros.

Se considera buen detector aquel que tenga las siguientes propiedades:

1. Repetibilidad: como se ha comentado, a la hora de concluir que nos encontramos ante una situación de cierre de lazo, se necesita que una gran cantidad de puntos de interés hayan sido detectados previamente en varias imágenes.
2. Precisión en la localización, tanto en posición como en escala.
3. Eficiencia computacional.
4. Robustez.
5. Invarianza a cambios geométricos y/o de iluminación.
6. Distintividad.

Existe un gran número de métodos que realizan la correspondencia de puntos a partir de la detección de **bordes**. Estos no son sensibles a cambios de intensidad, lo que representa una buena ventaja, pero dan problemas cuando aparecen otras transformaciones entre imágenes.

El algoritmo de *Canny* es usado para detectar todos los bordes existentes en una imagen, y está considerado uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución y el cálculo de la primera derivada. Los puntos de borde son zonas de píxeles en las que existe un cambio brusco de nivel de gris.

Se puede resumir el algoritmo de *Canny* en tres fases:

1. Obtención del gradiente: cálculo de la magnitud y orientación del vector gradiente en cada píxel.
2. Supresión no máxima: reducción del ancho de los bordes hasta obtener bordes de un píxel de anchura.
3. Histéresis de umbral: reducción de la posibilidad de aparición de contornos falsos realizando una histéresis basada en dos umbrales.

Figura 2.4: Ejemplo de detección de bordes en una imagen haciendo uso del método Canny.

Al encontrar bordes cercanos al umbral de detección, un pequeño cambio en la intensidad de éste podría causar un gran cambio en su topología, y por tanto, aparecerían errores de correspondencias. Para evitarlo, se puede recurrir a la detección de **esquinas**. El detector de esquinas de *Harris* es uno de los detectores de puntos de interés más comunes. Estos puntos son poco sensibles a cambios en la rotación y escala. Una esquina está caracterizada por ser una región con cambios de intensidad en diferentes direcciones, que es precisamente el principio de búsqueda de puntos de *Harris*.

Al desplazar una ventana sobre una imagen en varias direcciones se producen cambios, y en función de estos se diferencian tres tipos de regiones detectadas.

1. Región plana: no hay cambios en ninguna dirección.
2. Borde: no hay cambio en la dirección del borde.
3. Esquina: hay cambios significantes en todas las direcciones.

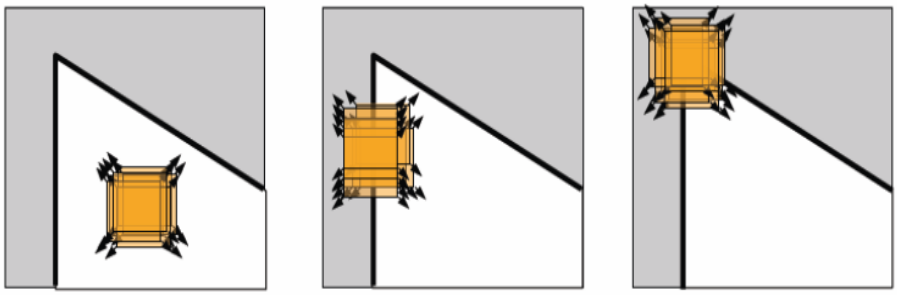


Figura 2.5: Desplazamiento de la ventana sobre una imagen para determinar si una región se trata de una esquina, borde o región plana.

En la tercera imagen de la Figura 2.5, se observa que la ventana está centrada en una esquina, por lo que habrá un cambio de intensidad notable en todas las direcciones.

Para determinar matemáticamente si se ha detectado una esquina, se desplaza la ventana un salto de *(u,v)* y se compara cada píxel antes y después del desplazamiento, sumando las diferencias de nivel de gris al cuadrado (SSD, *Sum of Squared Differences*): $E(u, v)$.

$$E(u,v) = \sum_x \sum_y w(x,y)(I(x+u, y+v) - I(x,y))^2 \tag{2.1}$$

Respecto a la fórmula anterior 2.1, comentar que $I(x,y)$ es la intensidad de la imagen original, $I(x+u, y+v)$ es la intensidad de la imagen desplazada por el vector (u,v), y $w(x,y)$ es la ventana sobre la que se realiza en análisis.

Para ventanas muy parecidas, $E(u,v)$ será muy pequeño, mientras que para ventanas muy diferentes, el error será elevado. A la hora de detectar esquinas, interesa, por tanto, que el valor de dicho error sea lo más grande posible.

Para desplazamientos pequeños, $I(x,y)$ se reemplaza por una aproximación lineal según su desarrollo de Taylor.

$$E(u,v) \approx \sum_x \sum_y w(x,y)[u^2 \frac{\delta^2 I}{\delta x^2} + 2uv \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} + v^2 \frac{\delta^2 I}{\delta y^2}] \tag{2.2}$$

Se representa la Ecuación 2.2 de forma matricial:

$$E(u,v) \approx \sum_x \sum_y [uv] w(x,y) \begin{bmatrix} \frac{\delta^2 I}{\delta x^2} & \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} \\ \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} & \frac{\delta^2 I}{\delta y^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.3}$$

Al tratarse de una suma en x e y, la Ecuación 2.3 equivale a:

$$E(u,v)[uv] \approx \sum_x \sum_y w(x,y) \begin{bmatrix} \frac{\delta^2 I}{\delta x^2} & \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} \\ \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} & \frac{\delta^2 I}{\delta y^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.4}$$

De forma resumida, la Ecuación 2.4 se expresa de la siguiente forma:

$$E(u,v)[uv] \approx [uv] M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.5}$$

$$M \approx \sum_x \sum_y w(x,y) \begin{bmatrix} \frac{\delta^2 I}{\delta x^2} & \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} \\ \frac{\delta I}{\delta x}\frac{\delta I}{\delta y} & \frac{\delta^2 I}{\delta y^2} \end{bmatrix} \tag{2.6}$$

Por tanto, los elementos de la matriz de autocorrelación, M, se obtienen al elevar al cuadrado las derivadas parciales y realizar un filtrado posteriormente. Para ello, se utiliza un filtro gaussiano, evitando así una respuesta ruidosa.

Ahora se definen λ_1 y λ_2 como los autovalores de la matriz M y se obtiene, en función de ellos, los tres tipos de regiones posibles:

1. Si los valores de λ_1 y λ_2 son pequeños, la función de autocorrelación es plana y, por tanto, la imagen tiene una intensidad constante. Se trata de una región plana.

2. Si uno de los autovalores es elevado y el otro no, la función de autocorrelación tendrá un cierto rizado y la región representa un borde.

3. Si los dos autovalores son elevados, la función contiene cambios bruscos y, por tanto, se ha detectado una esquina.

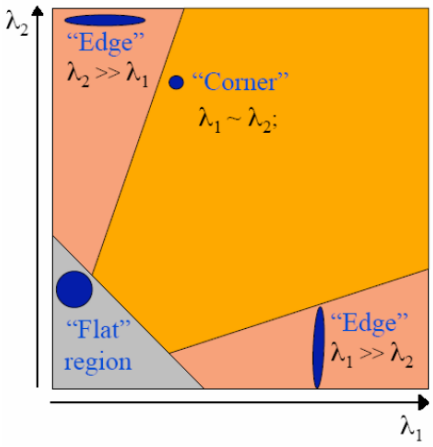


Figura 2.6: Clasificación de las regiones en función de los autovalores de la matriz M.

Los autovalores muestran los modos de variación de las direcciones principales de los gradientes. Por esta razón, cuando en una región los dos valores son elevados, se deduce que localmente existen dos direcciones importantes en los gradientes, y se trata, por tanto, de una esquina.

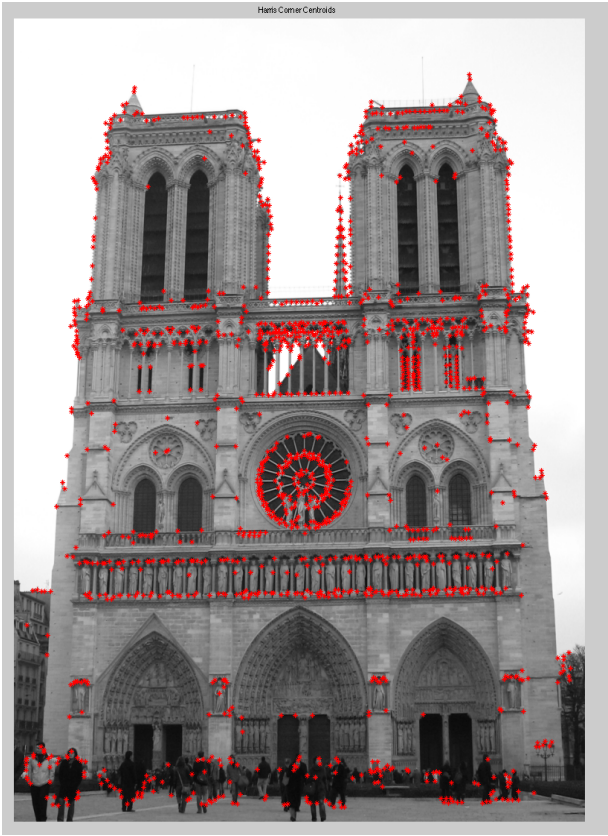


Figura 2.7: Ejemplo de la detección de esquinas Harris sobre una imagen de la catedral de Notre Dame, París.

A pesar de ser uno de los detectores más utilizados actualmente, *Harris* presenta el problema de ser computacionalmente muy costoso, ya que para determinar una esquina precisa del cálculo de las derivadas primera y segunda de cada píxel además del cálculo del determinante y traza de la matriz M. Para solventar este problema se puede hacer uso de otros detectores de

características mucho más rápidos.

Uno de los métodos de extracción de *keypoints* ampliamente utilizado es el algoritmo SIFT (*Scale Invariant Feature Transforms*), cuyo nombre es debido a que transforma los datos de la imagen en coordenadas invariantes a la escala, rotación e iluminación. SIFT trabaja en un espacio de escalas en el que se realiza un emborronamiento progresivo de la imagen. A continuación, se calcula la diferencia de gaussianas en cada escala, ya que los posibles puntos de interés son precisamente los máximos y mínimos de esas diferencias. Para conseguir una localización más precisa del punto característico se calcula su aproximación de Taylor. Además, como la diferencia de gaussianas proporciona una respuesta alta para aristas, éstas deben ser filtradas. Para ello, se calcula la matriz Hessiana para cada uno de los píxeles.

En la detección a partir de este algoritmo se pueden diferenciar cuatro fases:

1. Detección de máximos y mínimos de la escala.
2. Localización de puntos de interés.
3. Asignación de la orientación.
4. Descriptor del punto de interés, pues SIFT no sólo detecta el punto de interés, sino que también lo describe (fase que sigue a la detección en la odometría).

Por otra parte, el detector y descriptor de puntos característicos, desarrollado por Herbert Bay, TinneTuytelaars y Luc Van Gool y denominado SURF, presenta algunas ventajas sobre otros métodos, como una mejora en la velocidad de cómputo y la robustez ante transformaciones de imágenes. Estas mejoras se logran gracias a la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los *keypoints*, a la vez que se mantienen las propiedades de repetibilidad y distintividad entre distintos puntos de interés.

Las etapas de las que consta el algoritmo SURF son las siguientes:

1. Detección de puntos de interés.
2. Asignación de la orientación.
3. Extracción de los descriptores.

El detector local ORB, desarrollado por Rublee. E, Rabaud. V, Konolige. K, y Bradski. K, fue presentado como una alternativa a los detectores SIFT y SURF, siendo una de sus ventajas sobre estos su alta velocidad de cómputo. ORB hace uso del algoritmo FAST (*Features from Accelerated Segment Test*) para la detección de puntos de interés y de BRIEF (*Binary Robust Independent Elementary Features*), para la extracción de los descriptores.

Tanto FAST como BRIEF son detectores que mejoran notablemente la velocidad de cálculo, pero como inconveniente, FAST carece de un operador de orientación y BRIEF de un operador de invarianza a la rotación. Para solucionar estos problemas, ORB propone las siguientes soluciones:

1. oFAST (*FAST KeypointOrientation*), que representa al algoritmo FAST con un componente de orientación para los puntos de interés detectados.
2. rBRIEF (*Rotation-AwareBrief*), el método BRIEF al que se le añade la propiedad de la invarianza y la mejora de la velocidad de cómputo.

Por último, se encuentra el novedoso detector y descriptor AKAZE que, a diferencia del resto de algoritmos, sólo puede computar los puntos de interés detectados por él mismo. La principal ventaja de este nuevo detector frente a otros es que, mientras SIFT, SURF o FAST, realizan un filtrado gaussiano que no conserva los bordes naturales de los objetos y suaviza los detalles y el ruido en todas las escalas, AKAZE detecta y describe las características en espacios de escala no lineal. Esto permite un suavizado de la imagen localmente adaptable, difuminando pequeños detalles, pero conservando los bordes de los objetos.

Este método aumenta la repetibilidad y el carácter distintivo frente a otros detectores ya mencionados, gracias a la utilización de un filtrado de difusión no lineal. Sin embargo, la principal desventaja de AKAZE es que es computacionalmente intenso: dado que no existe ningún método analítico para resolver la ecuación de difusión no lineal, es necesario utilizar métodos numéricos para aproximar una solución. Este detector y descriptor va a ser explicado en detalle en capítulos posteriores.

2.2.3. Descripción de puntos característicos

La descripción de puntos característicos va a ser explicada en profundidad al tratar el tema de la detección de cierre de lazo, aunque también es necesario mencionarla en esta sección, puesto que se trata de una fase esencial a la hora de estimar el movimiento de un móvil mediante odometría visual.

Los parámetros calculados hasta el momento forman un sistema de coordenadas en dos dimensiones que describe localmente cada región de la imagen, y que por tanto, proporciona invarianza a esos mismos parámetros. El siguiente paso es, utilizando toda esa información, obtener un descriptor para cada zona de interés. Éste aporta robustez ante posibles variaciones de iluminación o cambios en el punto de vista en tres dimensiones.

2.2.4. Correspondencia o *matching* de puntos característicos

Después de la detección y descripción de los puntos característicos de las imágenes, el siguiente paso para la estimación del movimiento de un vehículo móvil es la correspondencia de estos puntos, de manera que se determine qué píxel de la imagen actual se corresponde con el mismo en otra imagen. El emparejamiento de puntos va ligado al proceso de descripción, pues en función de qué tipo de descriptores se empleen, se obtendrá mayor o menor número de correspondencias, existiendo, además, la probabilidad de que alguna de ellas se haga de manera errónea.

Esta etapa es clave en este proyecto, ya que el primer objetivo es la detección de cierres de lazo en los mapas obtenidos con la odometría visual, lo que se consigue cuando se realiza un número significativo de emparejamientos entre descriptores de imágenes tomadas en diferentes instantes de tiempo, con un valor de similitud más bajo que un umbral fijado (más adelante se va a explicar en qué consiste la medida de similitud). Por tanto, es de suma importancia que este emparejamiento se produzca de forma correcta, y, en caso de que no sea así, será necesario el uso de algún método que elimine falsas correspondencias, u *outliers*. En este proyecto se utiliza para ello el método RANSAC.

En la Figura 2.8 se observan correspondencias entre descriptores AKAZE de una imagen y la misma rotada (invarianza a la rotación). En este caso, la mayoría de correspondencias son correctas, aunque se puede observar algún emparejamiento erróneo, en la región donde se encuentran las letras.

En este proceso de *matching* pueden producirse emparejamientos erróneos, debido a difer-

Figura 2.8: Ejemplo de *matching* o correspondencia entre descriptores AKAZE.

entes factores como puede ser el cambio de iluminación, perspectiva o el ruido presente en la imagen. Cuanta mayor complejidad presenten los descriptores empleados en el procesamiento de las imágenes, mejores resultados se obtendrán, aunque el coste computacional también será mayor. De hecho, se ha comprobado la fiabilidad de las correspondencias obtenidas con AKAZE en la Figura 2.8, a pesar de haber realizado un cambio de perspectiva y escala en la segunda imagen y haber añadido elementos en ella, aunque, el tiempo de cómputo necesario es más elevado que haciendo uso de otros descriptores como FAST o SURF.

Por tanto, tras realizar esta etapa, se puede observar tanto la existencia de correspondencias satisfactorias, *inliers* como incorrectas, *outliers*. En imágenes más complejas que la mostrada anteriormente, no es fácil determinar a simple vista qué correspondencias son correctas y cuales no, y el hecho de trabajar con un conjunto elevado de correspondencias negativas conduce a un procesamiento incorrecto. Para evitarlo, se hace uso del citado método, RANSAC, (*RANdom SAmple Consensus*), introducido por Martin L. Fischler y Robert C. Bolles. Es un algoritmo iterativo que estima los parámetros de un modelo matemático de un conjunto de datos que contiene *outliers*.

Existen otros métodos para la estimación de estos parámetros, como el de mínimos cuadrados, que le da el mismo peso a todos los datos, por lo que la presencia de un *outlier* puede distorsionar el modelo obtenido.

La idea de RANSAC es hacer uso de la menor cantidad posible de datos para estimar el modelo y ver luego cuántos puntos se ajustan al modelo estimado. RANSAC es capaz de interpretar o filtrar datos que contienen una gran cantidad de errores. Es una técnica que está idealmente diseñada para aplicaciones de análisis de imágenes automáticas, donde la interpretación está basada en datos proveídos por detectores de puntos característicos que pueden generar errores.

De forma resumida, el algoritmo RANSAC consta de los siguientes pasos:

1. Selección aleatoria de un subconjunto de datos, en este caso, un subconjunto aleatorio de puntos característicos.
2. Generación de un modelo a partir de ese subconjunto.
3. Estudio del número de puntos que cumplen el modelo, los *inliers*.
4. Repetición de los puntos 2) y 3) un número de iteraciones determinado, y selección del modelo que tenga un número máximo de *inliers*.

En odometría visual se hace uso de este método para la generación de un modelo que no contenga *outliers*, o al menos, que contenga el menor número posible de ellos. En este caso, el modelo computado es el conjunto de la rotación y traslación entre las poses.

Para poder descartar de nuestro modelo los puntos erróneos, se computa la distancia punto-línea epipolar (aproximada con la distancia Sampson, una aproximación de primer orden), considerando *inliers*, los puntos que se encuentren a una distancia menor que un umbral determinado. Este proceso tiene un gran coste computacional y puede llegar a tomar hasta la mitad del tiempo de cómputo.

Por último, se procede a explicar el número de puntos necesarios para computar la estimación del modelo en cada iteración. Minimizar el número de puntos necesarios para calcular el modelo conlleva reducir también el número de iteraciones, que viene dado por la expresión 2.7:

$$n = \frac{log(1-p)}{log(1-(1-\epsilon)^s)} \tag{2.7}$$

Donde p representa la probabilidad de éxito, ϵ el porcentaje de valores atípicos en el conjunto de datos, en este caso, correspondencias, y n el número de datos necesarios para computar el modelo del movimiento.

En la Figura 2.9 se muestra el resultado de la aplicación del método RANSAC, usando un modelo sencillo, en este caso una línea.

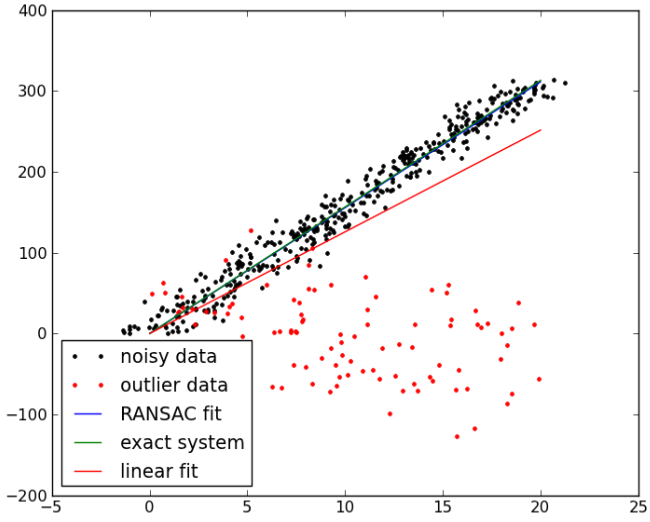


Figura 2.9: Ejemplo de aplicación del método RANSAC.

2.2.5. Geometría de la cámara

Para representar los resultados de la odometría visual se precisa de un modelo que permita pasar del mundo real en 3D al plano imagen en 2D, que es precisamente la función de una cámara.

En esta sección se explica el modelo *pin-hole*, que es el más sencillo y apropiado en visión artificial, aunque no es el único existente. Para formar el modelo *pin-hole*, se considera un centro

de proyección en el espacio, y un plano situado a una distancia f, llamada distancia focal, del centro de proyección. A este plano se le denomina plano focal o plano de la imagen.

Como se observa en la Figura 2.10, la proyección generada por la cámara será la intersección entre el plano de la imagen y la línea que une el punto a proyectar con el centro de proyección.

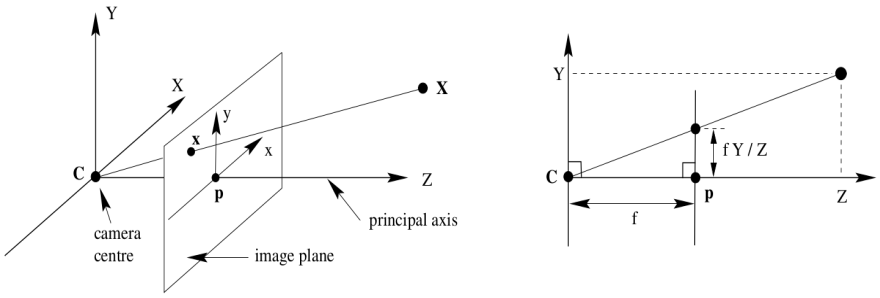


Figura 2.10: Modelo *pin-hole* de una cámara.

Al centro de proyección se le llama centro óptico. A la línea perpendicular al plano de la imagen que pasa por el centro óptico se la conoce como eje principal, y al punto de intersección entre el eje principal y el plano de la imagen se le denomina punto principal.

El principio de funcionamiento del modelo es el siguiente: los rayos salientes del objeto pasan a través de la óptica de la cámara y forman una imagen en su intersección con el plano imagen, quedando dicha imagen invertida. Esto mismo sucede en nuestra retina y es el cerebro el encargado de procesar la imagen y volverla a invertir.

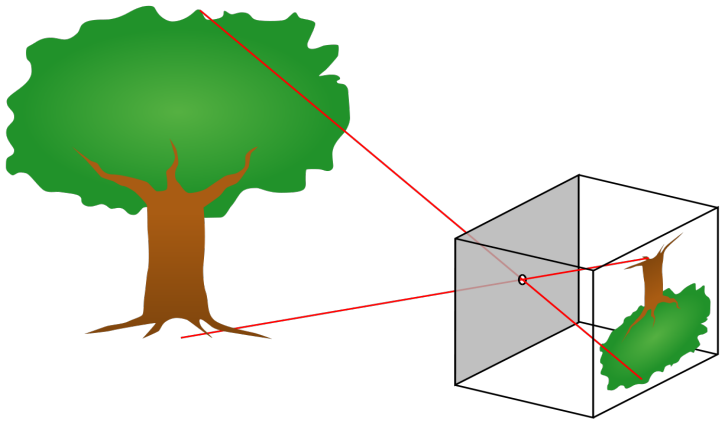


Figura 2.11: Representación de un objeto 3D en un plano mediante el modelo *pin-hole*.

En las siguientes líneas se deducen las ecuaciones que modelan este sistema de proyección.

Se representan los puntos del mundo real y de la imagen mediante vectores homogéneos y se expresa el modelo de la cámara mediante una transformación lineal entre sus coordenadas homogéneas. El modelo queda entonces expresado como el producto de las siguientes matrices:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.8}$$

Donde el vector $(x, y)^T$ son las coordenadas del punto en el plano imagen, y (X, Y, Z) las coordenadas del punto en el mundo real. La relación de equivalencia $\sim$ indica que los puntos $(x, y, 1)^T$ y $(fX, fY, Z)^T$ son iguales en el espacio proyectivo P^2, es decir, existe un escalar λ tal que $\lambda(x, y, 1) = (fX, fY, Z)$. Por lo tanto:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} \sim \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \\ 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.9}$$

Este sistema puede expresarse de forma más compacta como $x \sim P \cdot X$, siendo P la matriz de proyección de la cámara.

Este modelo es válido para un sistema de cámaras ideal pues no tiene en cuenta los factores que influyen en una cámara real como pueden ser las imperfecciones en la fabricación del sensor o de la lente, lo que distorsiona las imágenes. Estos defectos hacen que el eje principal no intersecte con el plano de la imagen en el mismo centro y que los píxeles no sean cuadrados perfectos. Por lo tanto, se tiene que trabajar con el sistema de coordenadas de la cámara real, teniendo en cuenta la posición real del punto principal (u_o, v_o) y el escalado en las direcciones x e y (α_u, α_v).

La siguiente expresión representa la transformación de las coordenadas del plano a un sistema de coordenadas homogéneas, considerando lo explicado anteriormente.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{f} \begin{bmatrix} \alpha u & 0 & x_0 \\ 0 & \alpha_v & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{cam} \\ y_{cam} \\ f \end{bmatrix} = K \begin{bmatrix} x_{cam} \\ y_{cam} \\ f \end{bmatrix} \tag{2.10}$$

Siendo K la matriz de parámetros intrínsecos de la cámara, que transforma las coordenadas en milímetros de un punto en el plano imagen (x_{cam}, y_{cam}), en coordenadas pixélicas (x, y).

Otro aspecto a tener en cuenta es que, por norma general, los puntos de una escena se representan en el sistema de coordenadas 3D y no en el de la cámara. Por lo tanto, es necesario hacer una transformación que convierta las coordenadas del mundo en coordenadas de la cámara, lo que se traduce en una rotación y traslación que se representa mediante la matriz de parámetros extrínsecos:

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.11}$$

Donde R es la matriz de rotación y t es el vector de traslación. Con toda esta información, combinando parámetros intrínsecos y extrínsecos, se puede establecer la relación entre los puntos del mundo real en 3D y en el plano imagen:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.12}$$

Esta relación se puede expresar de manera resumida con la siguiente ecuación de proyección:

$$x = PX \tag{2.13}$$

Donde, como ya se ha dicho, P se conoce como matriz de proyección. El cálculo de la matriz K se conoce como calibrado de la cámara y el cálculo de la posición se conoce como *tracking* 3D o estimación de la pose.

Por otra parte, el objetivo de la calibración es la obtención de los parámetros intrínsecos de la cámara (cuatro), más los elementos del vector de distorsión (cinco). Para realizar dicha calibración, se usa una estructura conocida, que tenga una serie de puntos fácilmente identificables (es preciso conocer las dimensiones y forma de la estructura y la distribución de los puntos). A continuación, se toman varias imágenes de la estructura desde distintos ángulos, y se calculan los parámetros intrínsecos de la cámara y la matriz de parámetros extrínsecos (orientación y traslación relativa de la estructura respecto de la cámara) de todas las imágenes.

Como patrón de calibración se puede hacer uso de un tablero de ajedrez, cuyos puntos de interés son las esquinas (MxN).

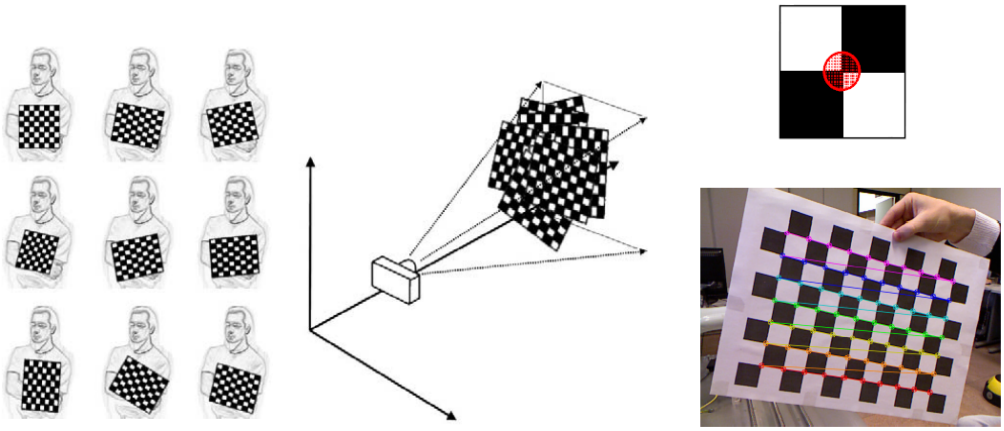


Figura 2.12: Calibración de una cámara haciendo uso de un patrón tipo tablero de ajedrez.

En la práctica, primeramente se extraen cuatro puntos de referencia del tablero, como por ejemplo los puntos mostrados en la Figura 2.13 pues, en este caso, al encontrarse en el centro del patrón, poseen mayor confiabilidad de detección.

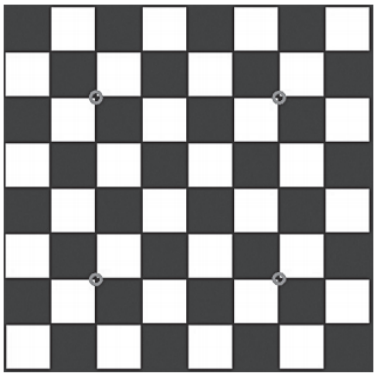


Figura 2.13: Detección de cuatro puntos de referencia en el tablero para la calibración de la cámara

Estos cuatro puntos se utilizan para estimar los parámetros intrínsecos y extrínsecos de la cámara por medio del método de calibración a partir de la superficie plana propuesto por Zhang.

La matriz de parámetros intrínsecos es calculada usando varias imágenes con diferentes posiciones del tablero y sin alterar la posición de la cámara. Esta matriz se mantiene constante mientras no se modifique el enfoque de la cámara. Una vez que la matriz de calibración intrínseca es calculada, la matriz de rotación R y el vector de traslación t se estiman para cada imagen.

En su investigación, Zhang recomienda que tanto el vector de traslación como las matrices de rotación y de calibración intrínseca se refinen utilizando regresión no lineal.

El proceso de calibración es, por tanto, previo a la realización de cualquier método de odometría visual, y en lo que respecta a este proyecto, al hacer uso de las secuencias de vídeo del dataset de KITTI, no se tuvo que realizar la calibración de las cámaras, aunque sí fue necesario indicar al algoritmo de LIBVISO el conjunto de parámetros intrínsecos y extrínsecos del sistema de cámaras utilizado.

2.2.6. Estimación del movimiento

Como se ha comentado en el estado del arte de la odometría visual, los algoritmos SFM (*Structure From Motion*) son capaces de calcular la estructura 3D de una escena a partir de varias proyecciones de la misma, es decir, a partir de una secuencia de vídeo. Estos métodos se basan en el efecto parallax para calcular el movimiento de la cámara a lo largo de la secuencia. El efecto parallax es el movimiento aparente de un objeto respecto al fondo cuando cambia el punto de vista del observador.

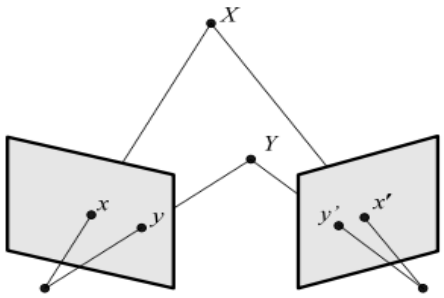


Figura 2.14: Representación efecto parallax.

Estos algoritmos se basan en el conocimiento de la geometría epipolar [41]: dos cámaras proporcionan dos proyecciones distintas de una misma escena, que están relacionadas entre sí mediante esta geometría, tal y como se muestra en la Figura 2.14. En ella se puede observar un esquema de un sistema de dos cámaras, donde se captura un punto X que se proyecta en los puntos x y x'.

La geometría epipolar queda totalmente definida por la matriz fundamental F. Esta matriz relaciona dos proyecciones de un mismo punto mediante la restricción epipolar:

$$x'^T F x = 0 \tag{2.14}$$

Siendo F la matriz fundamental, x las coordenadas de un punto característico en un frame y x' las coordenadas del mismo punto en el siguiente frame. La matriz fundamental se puede calcular a partir del *matching* entre puntos correspondientes a las dos vistas. Esta matriz es de

ocho grados de libertad, lo que significa que se necesita un mínimo de ocho correspondencias para que el sistema quede determinado.

$$Af = \begin{bmatrix} x_1x_1 & x_1y_1 & x_1 & y_1x_1 & y_1y_1 & y_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_8x_8 & x_8y_8 & x_8 & y_8x_8 & y_8y_8 & y_8 & x_8 & y_8 & 1 \end{bmatrix} f = 0. \tag{2.15}$$

Al trabajar con mediciones tomadas en imágenes reales, al igual que sucede al estimar la trayectoria de un robot, el error acumulado al medir la posición de los puntos hace que el sistema sea incompatible, por lo que se tiene que resolver con métodos de minimización. Además, es necesario utilizar el método RANSAC para eliminar *outliers* y un método de optimización para evitar problemas derivados de la existencia de ruido en las poses y así obtener una matriz fundamental válida para la estimación de la geometría.

Tras calcular la matriz fundamental se puede proceder a una reconstrucción 3D proyectiva de la escena. Esta reconstrucción es ambigua y sigue siendo válida después de aplicarle cualquier transformación en la proyección. Conociendo los parámetros intrínsecos de la cámara, se puede calcular la matriz esencial, E, a partir de la matriz fundamental, F:

$$E = K'^T F K \tag{2.16}$$

La matriz esencial permite calcular una reconstrucción métrica de la escena, que es precisamente lo que se hace a partir de la información obtenida con el método de odometría visual. Finalmente, conociendo la reconstrucción 3D y sus correspondientes proyecciones 2D se procede a la resolución de la pose de la cámara para cada frame.

2.3. Algoritmo LIBVISO

En esta sección se explica el método que se ha elegido para realizar la odometría visual en este proyecto.

Como se explica en [3], LIBVISO es un algoritmo introducido por Andreas Geiger para la estimación del movimiento de un móvil o de sus seis grados de libertad, es decir, para determinar cuál ha sido la rotación y traslación realizada en los ejes x, y y z, y para ello se hace uso de un sistema de cámaras estéreo totalmente calibrado. Por lo tanto, se necesitan como argumentos de entrada al algoritmo los parámetros intrínsecos y extrínsecos de las cámaras, que pueden ser diferentes en función de la secuencia de vídeo a procesar.

Este método está basado en la geometría trifocal entre imágenes. Inicialmente, se extraen y emparejan los puntos característicos de dos pares de imágenes consecutivas. En base a esas correspondencias, el movimiento del robot es estimado usando un tensor trifocal que relaciona los puntos de interés entre tres frames pertenecientes a la misma escena estática.

En LIBVISO se utiliza un sistema de cámaras estéreo para conseguir una mayor robustez en el algoritmo y evitar ambigüedades en la escala. Este método emplea, además, un filtro de linealización, el denominado filtro ISPKF (*Iterated Sigma Point Kalman Filter*). Por otro lado, los *outliers* son detectados a través del método RANSAC, que garantiza que los valores atípicos derivados de emparejamientos erróneos, sean rechazados antes de estimar el recorrido del móvil.

2.3.1. Modelo de la cámara

Sea K, de dimensiones 3x3, la matriz de parámetros intrínsecos de la cámara, la relación entre las coordenadas de la cámara respecto al sistema de referencia (mundo), y las coordenadas en el plano imagen vienen dadas por la Ecuación 2.17:

$$\tilde{x} = (u,v,w)^T = K \cdot X_c \tag{2.17}$$

En general, las coordenadas de la cámara y las coordenadas del mundo no están alineadas, pero se relacionan a través de un vector de traslación, t, y una matriz de rotación, R, que se corresponden precisamente con los parámetros extrínsecos de la cámara.

Dadas las coordenadas de un punto en el mundo, X_w, las coordenadas de ese punto referenciadas a la cámara, X_c, vienen dadas por la siguiente expresión:

$$X_c = R \cdot X_w + t \tag{2.18}$$

Combinando las ecuaciones 2.17 y 2.18, las coordenadas de un punto 3D en el plano imagen son:

$$\tilde{x} = P \cdot \tilde{X_w} \tag{2.19}$$

Donde P es una matriz de dimensiones 3x4, llamada matriz de proyección y es igual a:

$$P = K \cdot [R|t] \tag{2.20}$$

2.3.2. Relación entre tres imágenes

El tensor trifocal, τ, de dimensiones 3x3x3, expresa la relación entre tres imágenes de una misma escena estática y contiene la geometría proyectiva entre los diferentes puntos de vista. Además, es independiente de la estructura de la escena.

Conociendo las matrices de proyección de tres cámaras, P_A, P_B y P_C, los elementos del tensor trifocal siguen la siguiente expresión:

$$T_i^{qr} = (-1)^{i+1} \cdot det \begin{bmatrix} a^i \\ b^q \\ c^r \end{bmatrix} \tag{2.21}$$

Donde a^i representa la matriz P_A sin la fila i, y b^q y c^r representan la fila q de P_B y la fila r de P_C respectivamente.

A continuación, se hace uso de la propiedad del tensor trifocal, τ, para mapear o emparejar dos puntos característicos, x_A y x_B, correspondientes a las imágenes A y B respectivamente, en la imagen C.

En la Figura 2.15 se muestra este procedimiento de forma gráfica:

Se proyecta una línea aleatoria l_B en el mundo real 3D a través del punto x_B. Entonces, dadas la línea l_B y el tensor trifocal, τ, el punto x_C de la imagen C, que se corresponde con el

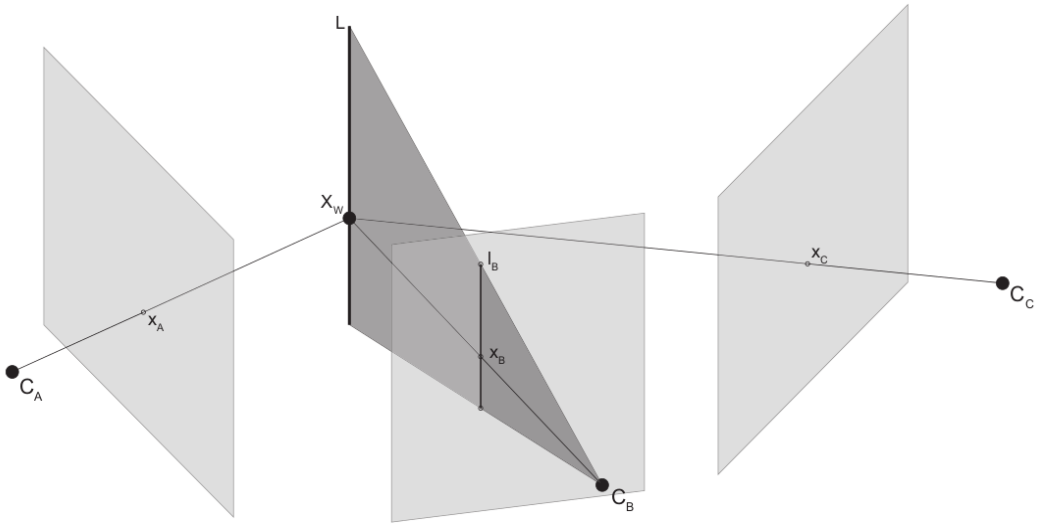


Figura 2.15: Representación de la relación entre puntos en tres imágenes.

emparejamiento de los puntos x_A y x_B en las imágenes A y B, se calcula a partir de la siguiente expresión:

$$x_C{}^k = x_A{}^k \cdot l_{B,j} \cdot T_i{}^{jk} \tag{2.22}$$

En las siguientes secciones, se explica la aplicación de la relación entre tres imágenes en la estimación del movimiento.

2.3.3. Odometría visual en LIBVISO

La estimación del movimiento relativo de la cámara se realiza, como ya se ha comentado en más de una ocasión, utilizando dos pares de imágenes estéreo consecutivas. En este proceso, también se hace uso del filtro ISPKF, que va integrando los parámetros del movimiento.

La Figura 2.16 muestra la configuración de un sistema estéreo en dos instantes de tiempo consecutivos. Se representan los planos imagen, las coordenadas de las cámaras y la orientación y traslación de éstas respecto a la cámara inferior derecha. La pose de la cámara inferior izquierda es dada por los parámetros extrínsecos de calibración (R_c, t_c), calculados respecto a la cámara inferior derecha, a partir de la estimación del movimiento, mientras que los parámetros (R_r, t_r) y (R_l, t_l) se corresponden con las poses de las cámaras derecha e izquierda, un instante de tiempo posterior (en la parte superior de la imagen), y son calculados a partir de la combinación de los parámetros extrínsecos de la cámara y la estimación del movimiento.

2.3.3.1. Parametrización del movimiento

Parametrizar el movimiento consiste en determinar la posición espacial de la cámara respecto a las coordenadas de referencia del mundo. Para ello, como se ha explicado anteriormente, se utiliza el vector de traslación, t, y la matriz de rotación, R. Esta matriz está formada por los ángulos de Euler, una concatenación de rotaciones respecto a los ejes x, y y z.

La estimación del movimiento, representado por t y R, se puede calcular en cada instante de tiempo si se conoce el vector (V_x, V_y, V_z, w_x, w_y, w_z) y el incremento de tiempo entre dos pares

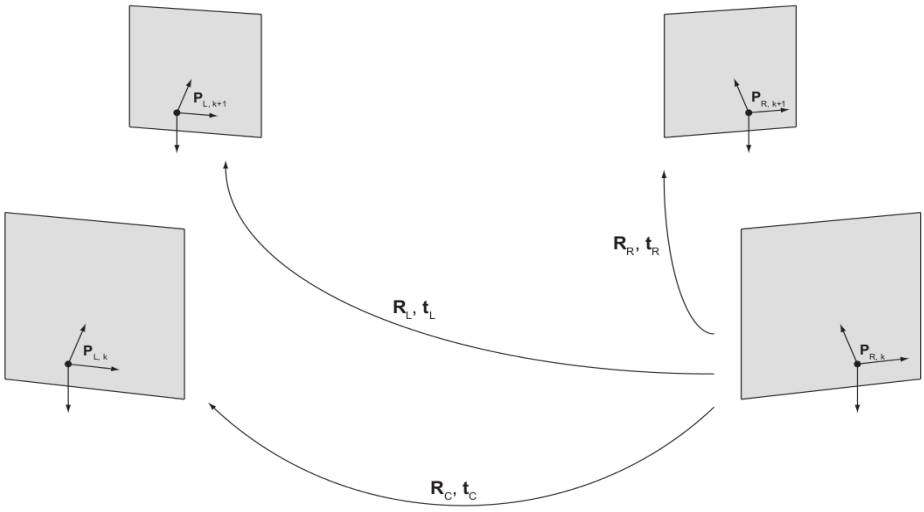


Figura 2.16: Representación de la configuración del sistema de cámaras estéreo en dos instantes de tiempo consecutivos, incluyendo las relaciones geométricas entre las imágenes.

de imágenes consecutivas.

En el vector anterior, V_i y w_i representan las velocidades lineales y angulares respectivamente. Entonces, la rotación y traslación del móvil vienen dadas por las siguientes ecuaciones:

$$t = (V_x \cdot \Delta T, V_y \cdot \Delta T, V_z \cdot \Delta T)^T \tag{2.23}$$

$$R = (w_z \cdot \Delta T, w_x \cdot \Delta T, w_y \cdot \Delta T) \tag{2.24}$$

2.3.3.2. Restricciones trifocales

Tal y como muestra la Figura 2.16, las matrices de proyección de las cuatro cámaras pueden ser estimadas si se conocen los parámetros intrínsecos y extrínsecos de las cámaras.

Si la cámara que se utiliza como referencia, en este caso, la cámara inferior derecha, tiene sus coordenadas alineadas con las coordenadas de referencia del mundo, las matrices de proyección del resto de cámaras se calculan de la siguiente forma:

$$P_{L,k} = K_L \cdot [R_c|t_c] \tag{2.25}$$

$$P_{R,k+1} = K_R \cdot [R_R|t_R] \tag{2.26}$$

$$P_{L,k+1} = K_L \cdot [R_L|t_L] \tag{2.27}$$

Donde k indica el instante de tiempo en que la imagen fue tomada.

Usando esta información, a partir de las matrices de proyección, se pueden calcular los dos tensores trifocales, τ. El primero relaciona el par de imágenes previo respecto a la imagen actual derecha, y el segundo respecto a la imagen actual izquierda:

$$\tau_R = \tau(K_R, K_L, R_c, t_c, R_R, t_R, \Delta T) \tag{2.28}$$

$$\tau_L = \tau(K_R, K_L, R_c, t_c, R_L, t_L, \Delta T) \tag{2.29}$$

El cálculo de estos dos tensores trifocales depende de la estimación del movimiento del sistema de cámaras estéreo y de los parámetros de calibración.

En cuanto a las diferentes posibilidades a la hora de llevar a cabo la detección y descripción de las imágenes, este método permite hacer uso del descriptor de Harris o realizar una descripción local mediante SIFT o SURF, descriptores muy distintivos que realizan emparejamientos robustos.

2.3.3.3. *Bucketing*

El siguiente paso es la detección y emparejamiento de puntos de interés en el par de imágenes estéreo. Una vez detectados estos puntos, se realiza un proceso de *bucketing*: se divide la imagen en varios rectángulos que no se superponen (véase la Figura 2.17) y que contienen un número máximo de puntos de interés. Este proceso ofrece varias ventajas. Por una parte, al dividir la imagen en subregiones, el número de puntos característicos en cada ventana es menor que en la imagen completa, lo que reduce la complejidad computacional del algoritmo, algo esencial en aplicaciones en tiempo real. Por otra parte, esta técnica garantiza que los puntos de interés queden distribuidos uniformemente a lo largo del eje z, lo que permite estimar de forma correcta las velocidades lineales y angulares. Además, esto implica que en escenas dinámicas, donde la mayoría de los *keypoints* dependen de objetos que se mueven, este método garantiza que todos los puntos no caigan sobre estos objetos, sino también sobre el fondo estático.

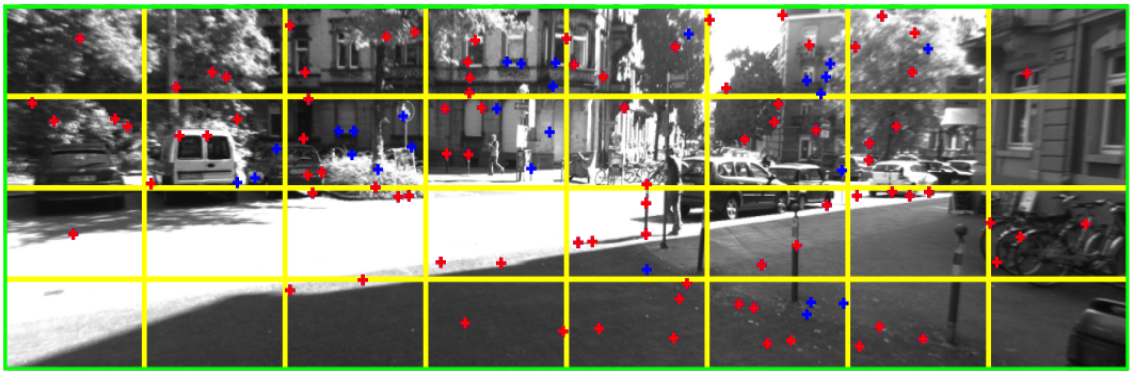


Figura 2.17: Resultado del proceso de *bucketing*.

La Figura 2.17 muestra el resultado del proceso de *bucketing*: el rectángulo verde define la región de interés y los rectángulos amarillos representan las subregiones en que se divide la imagen. Todas las cruces representan los emparejamientos encontrados en un par de imágenes estéreo: las cruces rojas son los puntos de interés seleccionados y las azules son puntos atípicos u *outliers*.

2.3.3.4. Método RANSAC

Este método ya ha sido explicado al tratar de forma general de las fases de las que consta la odometría visual, ya que, aunque existen varios métodos para rechazar puntos erróneos, este es el más extendido en la actualidad. Para llevarlo a cabo, se elige al azar unos subconjuntos de correspondencias y, en base a éstos, se estima el movimiento del móvil. Una vez que el filtro de Kalman converja, se calculan todos los *inliers* utilizando el error de reproyección euclidiana. Un punto característico se considera *inlier* si su error de reproyección euclidiana es inferior a un cierto umbral.

La técnica de *bucketing* propuesta en este algoritmo, combinada con el método RANSAC, proporciona una estimación robusta del movimiento, incluso en presencia de objetos en movimiento.

2.3.3.5. Filtro de KALMAN

El filtro de Kalman consta de una fase de predicción y otra fase de actualización. Se utiliza para estimar el estado actual de un sistema dinámico, que se supone que contiene ruido blanco de media cero. En la estimación del estado instantáneo, las mediciones a utilizar están perturbadas por este ruido.

En este proyecto, las relaciones entre el estado instantáneo (o movimiento relativo del móvil), $y = (V_x, V_y, V_z, w_x, w_y, w_z)^T$, y las mediciones (o posiciones de los puntos característicos en las imágenes actuales) son no lineales y vienen dadas por las siguientes ecuaciones respectivamente:

$$x_{R,k+1} = h_R(\tau_R, x_{R,k}, x_{L,k}) \tag{2.30}$$

$$x_{L,k+1} = h_L(\tau_L, x_{R,k}, x_{L,k}) \tag{2.31}$$

Por otro lado, las ecuaciones en el espacio de tiempo discreto del filtro de Kalman son:

$$y_{k+1} = f(y_k) + w_k \tag{2.32}$$

$$z_{k+1} = h(y_{k+1}) + v_{k+1} \tag{2.33}$$

Donde y_k es el estado del sistema en el instante de tiempo k, $f(.)$ es el sistema de ecuaciones no lineales y $h(.)$ es la ecuación de las mediciones realizadas, también no lineales.

Si se utiliza el filtro de Kalman en problemas no lineales, la linealización se suele realizar mediante una aproximación de Taylor de primer orden, lo que se conoce como filtro de Kalman extendido (EKF). Para reducir el error producido por la aproximación de Taylor, a menudo se itera la etapa de actualización (IEKF).

En LIBVISO se utiliza el denominado Filtro ISPKF (*Iterated Sigma Point Kalman Filter*), pues reduce el error de linealización y además proporciona otras ventajas respecto a otros filtros, ya que converge, aproximadamente, 60 veces más rápido que el filtro IEKF, sin necesidad de calcular derivadas. Además, converge en sólo tres iteraciones, mientras que el filtro IEKF necesita unas doscientas iteraciones para llegar a la misma solución.

2.3.4. Resultados parciales

Para la evaluación de los resultados de este algoritmo se puede hacer uso de datasets simulados o de conjuntos de datos reales. En este apartado se indican los resultados de un experimento real, haciendo uso de una secuencia de imágenes capturadas por un sistema estéreo de cámaras que el vehículo llevaba a bordo. Las imágenes fueron tomadas en un entorno urbano con elevado tráfico. El sistema de cámaras fue montado en la parte superior del coche, con una línea base de 0,7 metros.

En la Figura 2.18 se pueden observar los resultados de dos secuencias de imágenes correspondientes a trayectos de diferente longitud y velocidad. La secuencia representada en 2.18(a) supuso un reto importante debido a la vuelta de 360 grados realizada durante la maniobra de aparcamiento. Se representan los resultados del algoritmo de LIBVISO (azul) comparados

(a) Driven path for a sequence with 1200 frames.

(b) Driven path for a sequence with 1200 frames.

Figura 2.18: Resultado de algoritmo LIBVISO sobre entornos urbanos.

con la trayectoria proporcionada por las medidas de GPS (rojo). Se observa cierta deriva en la representación de la odometría visual.

Los resultados experimentales corroboran que el algoritmo propuesto obtiene una buena estimación del movimiento del vehículo en entornos urbanos, aunque siempre se acumula cierto error. Todos los resultados están comparados con el *Ground Truth*, o precisos mapas obtenidos mediante el GPS.

Como conclusión, la principal novedad del enfoque propuesto en este método es el uso del tensor trifocal entre tríos de imágenes en combinación con el método de rechazo de *outliers* RANSAC, que permite realizar una estimación del movimiento basada en el procesamiento de imágenes, sin que sea necesaria una reconstrucción 3D de la escena.

Por otro lado, la técnica de *bucketing* proporciona una buena distribución de los puntos de interés sobre la imagen, y más concretamente a lo largo del eje z, garantizando que la mayoría de éstos se sitúen sobre el fondo estático de la ventana y no sobre los objetos en movimiento. Esto permite también una estimación precisa de las velocidades angular y lineal.

2.4. Implementación de la odometría visual

En este capítulo se procede a estudiar cómo los bloques de un algoritmo de odometría visual, estudiados en el capítulo anterior, se implementan en las librerías de LIBVISO desarrolladas por Andreas Geiger, Julius Ziegler y Christoph Stiller. Estas librerías están escritas en C++, con alguna sección de código escrita con instrucciones vectoriales SSE3, con el fin de reducir el tiempo de cómputo en aquellas secciones de código críticas, como en funciones para la detección de puntos característicos, para la correspondencia de los mismos o para el cálculo de la derivada de la imagen.

Este algoritmo permite realizar una odometría visual, tanto monocular como estéreo. En este proyecto se ha utilizado el dataset de KITTI para testear los resultados, por lo que las secuencias de imágenes a procesar han sido capturadas por un sistema de cámaras estéreo, que es el tipo de odometría que se lleva a cabo. De forma esquemática, la estructura de los ficheros de la odometría visual estéreo es la que se muestra en la Figura 2.19.

Se explica, a continuación, la funcionalidad de cada uno de los ficheros:

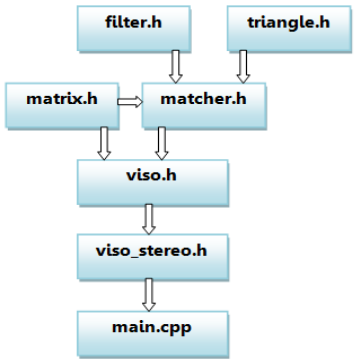


Figura 2.19: Cuerpo principal del algoritmo LIBVISO.

- Los ficheros de *filter* se encargan de procesar las imágenes del dataset, realizando filtrados mediante convoluciones y haciendo uso de filtros *kernel*. En estos ficheros, se implementa también la detección de esquinas y blobs y la derivada de la imagen, necesarias para la posterior descripción de los puntos de interés.
- Los ficheros de *triangle* realizan la triangulación de los puntos en 3D. Para ello, es necesario conocer la matriz de parámetros extrínsecos del sistema de cámaras.
- Los ficheros de *matrix* implementa numerosas funcionalidades para el trabajo con matrices, como el producto de las mismas, la descomposición en valores singulares o la extracción de valores.
- En el bloque *matcher* se realizan los procesos de detección, descripción y emparejamiento de puntos de interés.
- Al fichero principal *main*, se le pasa como argumento de entrada los parámetros de calibración de las cámaras: distancia focal, centro óptico y línea base. En él se procede a la lectura de cada par de imágenes tomadas con el sistema estéreo de cámaras. El algoritmo de odometría visual se realiza mediante la clase *VisualodometryStereo*. Es necesario configurar una serie de parámetros de cualquier objeto de esta clase:
  - Parámetros de calibración intrínsecos: línea base, centro óptico y distancia focal.
  - Parámetros para realizar el proceso de *bucketing*: número de puntos característicos por ventana y ancho y alto de la misma.
  - Parámetros para realizar el proceso de *matching*: umbrales para la detección, tamaño del grid, posible activación de refinamientos y posible reducción de la resolución de la imagen para un menor tiempo de cómputo.
  - Parámetros generales: número de iteraciones para el algoritmo de RANSAC, altura de las cámaras y pitch.

Centrándose en el algoritmo de odometría visual, cuando se ejecuta el fichero principal, se comienza la lectura de las imágenes de las cámaras derecha e izquierda, que se van cargando sucesivamente para realizar el procesamiento de las mismas por parejas. Este procesamiento se realiza con el método *process*, que devuelve a través del método *getMotion* la matriz esencial de rotación y traslación, denominada *pose*, que refleja la variación en la posición y orientación entre dos poses consecutivas del robot. El método *process* es, por tanto, el encargado de llevar a cabo todo el proceso de odometría visual, realizando todas las fases necesarias para ello.

En primer lugar, se realiza la detección y descripción de los puntos característicos mediante los métodos *computeFeatures* y *computeDescriptors*, ambos pertenecientes al método *pushBack*.

En la integración del método para la detección de cierre de lazo en el algoritmo de LIBVISO, se realizan por duplicado las fases de detección y descripción: una para obtener la odometría y la otra para la detección del bucle, como se va a explicar más adelante. El método *computeFeatures* calcula, además, las derivadas horizontal y vertical de las imágenes, necesarias para la descripción de las mismas, realizada mediante el método *computeDescriptors*.

En segundo lugar, se realiza el emparejamiento de puntos característicos mediante el método *matchFeatures*. El resultado de este proceso es proporcionado por *getMatches*, que devuelve el conjunto de correspondencias. Para finalizar la odometría, el método *updateMotion* estima el movimiento relativo entre las poses.

A continuación, se explica en detalle cada uno de los bloques que integran la odometría visual propuesta por Geiger.

2.4.1. Detección de puntos característicos

La detección de los puntos característicos de la imagen se realiza mediante el método *computeFeatures* y, para ello, se calcula la derivada de la imagen. En este caso, los puntos detectados son esquinas y blobs.

Para realizar este proceso, se procede a un filtrado de la imagen mediante un kernel, que genera una salida máxima (o mínima) en aquella región de la imagen donde se encuentre el punto característico. Aunque el proceso para detectar esquinas y blobs sea el mismo, se utilizan dos funciones diferentes.

- En el caso de la detección de esquinas, la función utilizada es **filter::checkerboard5x5()**, que implementa el filtrado de la imagen con el kernel indicado en 2.34, que calcula la derivada en las direcciones vertical y horizontal.

$$Kc = \begin{bmatrix} -1 & -1 & 0 & +1 & +1 \\ -1 & -1 & 0 & +1 & +1 \\ 0 & 0 & 0 & 0 & 0 \\ +1 & +1 & 0 & -1 & -1 \\ +1 & +1 & 0 & -1 & -1 \end{bmatrix} \tag{2.34}$$

  Mediante este kernel, se multiplica por +1 el cuadrante superior derecho y por -1 el cuadrante superior izquierdo, dando como resultado un valor máximo cuando se dé una gran diferencia entre estas regiones, lo que significa que se trata de una zona con puntos de interés. Los valores del filtro suman 0, resultado que se obtiene en las zonas más homogéneas.

- Por otra parte, la función utilizada para la detección de blobs es **filter::blob5x5()**, cuya diferencia con la función anterior radica en el uso de un kernel diferente (Ecuación 2.35):

$$Kc = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & +1 & -1 \\ -1 & +1 & +8 & +1 & -1 \\ -1 & -1 & +1 & +1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{2.35}$$

El funcionamiento del kernel es el mismo que el explicado anteriormente, con la salvedad de que en este caso se busca la detección de zonas en las que el centro de la ventana sea diferente a la zona exterior, en cuyo caso se obtiene un resultado máximo (o mínimo). Después de realizar el filtrado de la imagen, se procede a un proceso de supresión de no-máximos y no-mínimos, tras el que se obtienen los puntos característicos de la imagen.

2.4.2. Descripción de puntos característicos

Para la realización de este proceso se calculan las derivadas horizontal y vertical alrededor del punto de interés detectado anteriormente. Para ello se hace uso de la función **filter::sobel5x5()**. De forma similar a las funciones descritas en la detección, el operador Sobel realiza un filtrado mediante un kernel que genera respuesta cuando se trate de una zona con una variación significativa en la intensidad de los píxeles. La derivadas horizontal y vertical se calculan mediante los filtros kernel 2.36 y 2.37, respectivamente.

$$K_u = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} \tag{2.36}$$

$$K_v = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \tag{2.37}$$

Nuevamente, estos filtros cumplen la propiedad de salida nula en regiones homogéneas, pues sus valores suman 0. La salida del filtrado mediante K_u es máxima en las zonas donde existe una gran diferencia entre los píxeles en la dirección horizontal. En el caso del filtrado mediante K_v sucede lo mismo si la diferencia se da en la dirección vertical.

Como la detección y descripción de las imágenes se realiza en base al cálculo de la derivada, es necesario que éstas tengan suficiente textura, para que se pueda obtener suficiente información.

Posteriormente a la etapa de filtrado, se realiza una adaptación del resultado para obtener un valor con 8 bits por píxel, para después proceder a la descripción de los puntos característicos mediante el método: **Matcher::computeDescriptors()**. Esta función asigna a cada punto de interés, 16 puntos de la derivada horizontal y otros 16 de la derivada vertical, por lo que el descriptor será de tamaño 32.

2.4.3. Emparejamiento de puntos característicos

En esta fase se hace uso de tres funciones diferentes:

- En primer lugar, se comparan los descriptores para poder realizar un emparejamiento de los mismos, mediante la función **Matcher::findMatch()**. Esta comparación se realiza mediante la suma de diferencias absolutas: se compara cada píxel con sus vecinos y se computa un coste asociado a la suma de diferencias absolutas entre los descriptores. Aquel vecino que presente coste mínimo con el píxel bajo estudio es su candidato correspondiente.

Para asegurar que el emparejamiento es satisfactorio, se realiza una fase de consistencia, que consiste en comprobar que el candidato detectado también tiene un coste mínimo con el píxel estudiado en primer lugar.

- Si la fase de consistencia también da un resultado positivo, se realiza un proceso de refinamiento de las correspondencias, mediante el método **Matcher::removeOutliers()**, que realiza una primera detección de puntos erróneos.
- Para finalizar el proceso de *matching*, se realiza un último filtrado mediante un proceso de mínimos cuadrados a través del método **Matcher::refinement()**.

2.4.4. Extracción del movimiento

Se trata de la última fase de la odometría visual de LIBVISO, y consiste en el cálculo de la variación relativa entre imágenes, es decir, la rotación y traslación entre ellas.

En primer lugar, se inicializan las variables mediante el método *updateMotion*. La extracción del movimiento se realiza mediante *estimateMotion*. Este método realiza una normalización del conjunto de correspondencias determinado en la fase anterior. A continuación, se inicia una estimación de la matriz fundamental, F, mediante el método RANSAC. Por último, se calcula la matriz esencial, E, a partir de la fundamental, y se llama al método *EtoRt*, que extrae la rotación y traslación de dicha matriz, consiguiendo así el objetivo del algoritmo de odometría visual.

Capítulo 3

Método de detección de cierre de lazo

3.1. Motivación

Los sistemas de navegación basados en visión artificial usados por robots o vehículos inteligentes necesitan algoritmos eficaces para realizar un reconocimiento rápido y robusto del entorno. Como ya se ha comentado en algunas ocasiones anteriormente, la estimación de la trayectoria del móvil se realiza, en este proyecto, mediante odometría visual, un método que no es totalmente fiable pues está basado en fuentes de información que no son precisas (sistema de cámaras a bordo del vehículo). Por ello, es necesario explotar los recursos con los que se cuenta, para poder realizar una corrección del resultado que da inicialmente la odometría, que va acumulando un error debido a la integración continua de ruido que realiza. Como consecuencia, se necesita información adicional o complementaria que permita corregir esa deriva acumulativa. En este caso, esta corrección se realiza en base a una restricción en la posición del robot dada por la detección de cierre de lazo.

Esta detección es clave para conseguir una mayor robustez en los algoritmos de odometría y más concretamente en SLAM (Mapeado y Localización Simultáneos) y consiste en detectar cuándo el robot vuelve a recorrer una zona que ya había visitado previamente. Esto permite mejorar la estimación de la pose actual del robot (Figura 3.1). También es importante para determinar la localización global del vehículo y para "recuperarlo tras un secuestro" (situación en la que el robot es desplazado por algo que no es controlado por él). Por lo tanto, resolver el problema de la detección de cierre de lazo no sólo mejora los resultados del SLAM, sino que también proporciona nuevas aplicaciones en los robots móviles.

Como conclusión a la motivación de este capítulo, en nuestro objetivo principal de conseguir un método consistente, robusto y eficiente de localización del robot en tiempo real, es clave la detección de cierre de lazo en el camino recorrido por él.

3.2. Concepto general

Como se acaba de comentar, se denomina cierre de lazo a la parte del recorrido que es revisitada por un vehículo, es decir, que es transitada, al menos, en dos ocasiones diferentes. El móvil puede recorrer la zona repetidamente en la misma dirección o bien en dirección contraria, diferenciando de este modo entre cierre de lazo unidireccional y bidireccional, respectivamente.

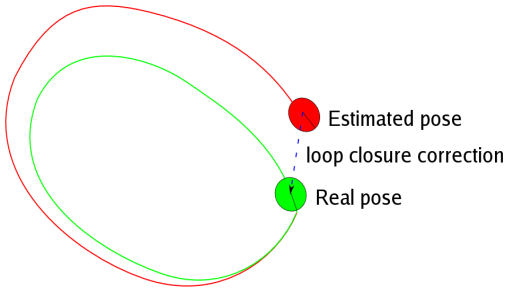


Figura 3.1: Detección de cierre de lazo y consecuente actualización de la pose actual del robot.

En este proyecto se pretende realizar la detección de cierre de lazo a partir de la odometría visual, es decir, teniendo como única información las imágenes capturadas por el sistema estéreo y sin tener conocimiento previo del entorno en el que se encuentra el agente. Para ello, son claves la fase de descripción de la imagen, el cálculo de la distancia de Hamming entre el descriptor actual y los descriptores de los frames anteriores y la estimación de la matriz de similitudes o distancias, en base a la cual se determina si se trata de una situación de cierre de lazo o no.

Existen diversas propuestas para resolver los problemas asociados usando métodos métricos, topológicos o híbridos. Un ejemplo de estos últimos se describe en [42]. Esta solución pretende construir un mapa topológico compuesto de diversos mapas métricos más sencillos. Después de esto, a medida que el vehículo explora nuevos lugares, el algoritmo decide si construir un nuevo submapa o crear una nueva relación con alguno de los mapas existentes. Los enlaces contienen las relaciones entre sistemas de coordenadas, así como las transformaciones de las incertidumbres.

En [43] se propone un SLAM 3D usando matrices de similitud SIFT, basadas en apariencia visual, que es en lo que se basa nuestro método de detección de cierre de lazo, ABLE [6], con la salvedad de que en la implementación que se ha realizado en este trabajo se realiza la descripción de la imagen mediante el nuevo descriptor binario AKAZE. Esto permite el reconocimiento de lugares previsitados que puedan ser altamente repetitivos.

El artículo presentado en [44] propone un método que es capaz de cerrar lazos muy grandes con un gran número de marcas en un entorno simulado. En él se usa un método jerárquico para representar las diferentes magnitudes probabilísticas asociadas a diversas regiones del mapa. Además, se proporcionan medios para transmitir las actualizaciones y predicciones del modelo de la parte superior a la inferior del árbol y viceversa.

Desde el desarrollo del método FAB-MAP, las técnicas de detección de cierre de lazo basadas en visión han aumentado notablemente gracias a los trabajos de investigación que se han desarrollado en los últimos años sobre este tema. Sin embargo, FAB-MAP presenta algunos inconvenientes, como la necesidad de una fase previa de entrenamiento para construir lo que en visión se conoce como *bag-of-words* del entorno, lo que hace que este método no sea el más adecuado para aplicaciones en tiempo real. Además de FAB-MAP, la detección de cierre de lazo se puede realizar mediante otros métodos, como WI-SURF, BRIEF-Gist, Filtros Bayesianos o mediante descriptores y matrices de distancia, como lo que presenta la teoría ABLE, que va a ser explicada en profundidad en las siguientes secciones.

De forma resumida, lo que ABLE pretende es detectar un bucle en base a la información almacenada en la matriz de similitud, la cual contiene el valor mínimo de las distancias de Hamming entre el descriptor de cada frame y los descriptores de todos los frames previos a

él. Estos valores son normalizados, de forma que varían dentro del intervalo [0,1], con el fin de determinar que se da una situación de cierre de lazo cuando los elementos de la matriz sean menores que un umbral fijado, cercano a 0, pues cuanto menor sea la distancia de Hamming entre dos descriptores, mayor es la probabilidad de que estén describiendo la misma imagen o escena. Sin embargo, éste no es un proceso trivial, sobre todo en situaciones de localización a largo plazo, y por ello, presenta uno de los mayores retos en el ámbito de la robótica hoy en día.

La apariencia de un mismo lugar puede variar significativamente dependiendo de la estación del año, de las condiciones meteorológicas o de la hora del día en la que nos encontremos, lo que hace que dependiendo del momento en que se graba la secuencia de vídeo de un entorno, los descriptores obtenidos de dos imágenes que representan el mismo lugar, puedan ser complicados de relacionar. Esto dificultaría enormemente la tarea de concluir que una zona ya ha sido atravesada.

En la Figura 3.2 se puede observar un ejemplo de lo explicado.

Figura 3.2: Variación de la apariencia de un lugar en función del momento del año. Imagen extraída del dataset Norland.

El dataset de Norland [45] incluye un vídeo grabado a lo largo de 728 kilómetros sobre unos raíles de tren en el norte de Noruega. En la Figura 3.2 se muestra el mismo lugar capturado en una estación del año diferente. Como se observa, hay una inmensa variación en la apariencia del entorno, que aparece totalmente cubierto de nieve en invierno, poblado de vegetación en primavera y verano o con colores rojizos y marrones, típicos del otoño.

En la Figura 3.3, se muestran otras situaciones que dificultan el reconocimiento de lugares aplicando visión artificial. De arriba a abajo, cada par de imágenes representa: a) Solapamiento perceptual entre lugares diferentes con apariencia similar. b) Luz solar a la misma hora del día en diferentes estaciones del año. c) Baja información visual en determinados lugares como túneles.

En las siguientes secciones se explican las fases de las que consta el método ABLE, que se utiliza en nuestro propósito de detectar el cierre de lazo sobre los resultados obtenidos del algoritmo LIBVISO.

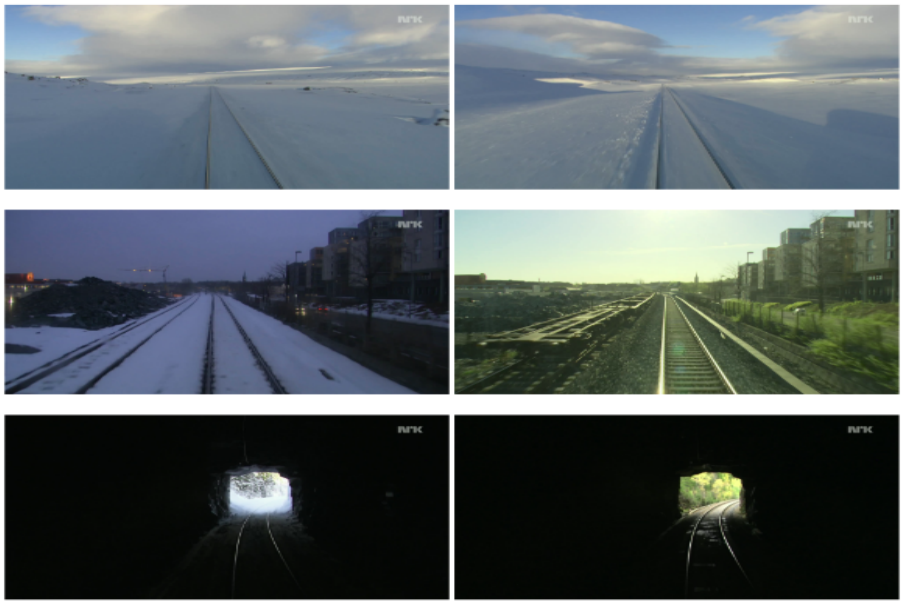


Figura 3.3: Tres ejemplos representativos de situaciones complejas para el reconocimiento visual, del dataset Norland.

3.2.1. Descripción de una imagen: tipos de descriptores

En la explicación de las distintas fases que conforman la odometría visual se citó la descripción, proceso que tiene lugar tras haber realizado la detección de la imagen.

Un buen detector de puntos de interés es capaz de localizar dichos puntos repetidamente y en distintas imágenes, a pesar del cambio de punto de perspectiva, siendo fiable ante las transformaciones que puedan tener lugar.

Una vez obtenidos los detectores de una imagen, se procede a su descripción. Un descriptor ideal proporciona de cada punto de interés, información relevante y distintiva de la región que lo rodea, de manera que la misma estructura pueda ser reconocida si es encontrada en otra imagen. Dicho de otra forma, un descriptor es un conjunto ordenado de números, expresado generalmente en forma de matriz o vector, que contiene información sobre la imagen que describe. Teóricamente se puede considerar a una imagen digital como un descriptor, pero en la práctica no se puede usar como tal debido a su enorme tamaño: una imagen de 1 Megapíxel (1 millón de píxeles) ocupa un tamaño de 3 Megabytes (cada píxel ocupa 3 bytes) y es por esta razón por la que se han desarrollado diferentes descriptores que extraen la información relevante de la imagen, para su posterior procesamiento.

3.2.1.1. Descriptores globales

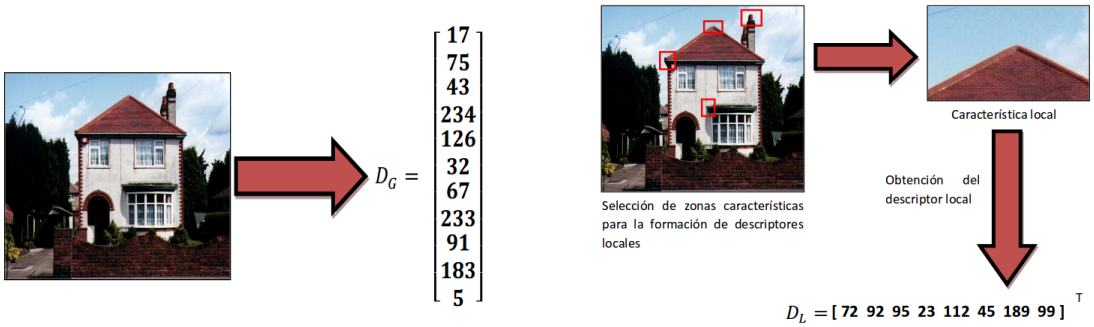
Este tipo de descriptores proporciona información sobre el conjunto de la imagen. Debido a la gran cantidad de píxeles que este descriptor debe representar, la información que se obtiene es muy genérica y suele hacer referencia a la textura o formas generales de la imagen.

GIST [46] es uno de los descriptores globales más conocidos, desarrollado por A. Oliva y A. Torralba. Con su creación se buscó un descriptor que fuera muy sencillo de obtener y que proporcionase unos ratios de acierto similares a los de otros descriptores mucho más complejos, con un coste de cómputo y almacenamiento mucho menor.

3.2.1.2. Descriptores locales

Estos descriptores, a diferencia de los globales, no describen el conjunto de toda la imagen, sino que seleccionan aquellas regiones que son más características o diferenciables. Como ya se ha comentado, lo más común es seleccionar aquellas zonas de alto gradiente, como son los bordes y esquinas presentes en la imagen.

En la Figura 3.4 se muestra la diferencia entre la descripción global y local en una imagen. En la Imagen 3.4b), la imagen se representa como un conjunto de descriptores locales, y como consecuencia, se requiere un mayor tiempo de ejecución y almacenamiento que haciendo uso de descriptores globales, como en la Figura 3.4a).



(a) Descripción global (b) Descripción local

Figura 3.4: Comparativa entre descriptor global y local en una imagen.

Como ya se ha comentado al explicar la fase de detección, en cuanto a los descriptores locales más conocidos (que a su vez son detectores) se encuentra SIFT, que propone un algoritmo para la extracción de características de una imagen añadiendo invarianza respecto a la rotación, que consigue asignando a cada uno de los puntos una orientación basada en las propiedades locales de la imagen y representando el descriptor respecto de esta orientación.

Por otra parte, SURF, que también es tanto detector como descriptor de puntos de interés, presenta como ventaja sobre otros métodos, la mejora de la velocidad de cálculo y la robustez ante transformaciones de las imágenes, reduciendo de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos.

FAST y BRIEF son algoritmos que mejoran notablemente la velocidad de cómputo, sin embargo carecen de un operador de orientación e invarianza a la rotación respectivamente.

3.2.1.3. Descriptores binarios

Cada elemento del descriptor binario es codificado en un bit (valor de 0 ó 1), de ahí su nombre. Los descriptores binarios pueden ser a su vez, locales o globales, pues esta clasificación se basa en la forma en la que se almacena la información.

La principal ventaja de este tipo de descriptores radica en la rapidez con la que un ordenador puede realizar operaciones binarias, al tiempo que se reduce significativamente el espacio necesario para almacenarlos. Como contrapartida, la información contenida en ellos es menor.

Se han producido grandes avances respecto a los descriptores binarios desde el desarrollo de BRIEF. Existen otros algoritmos como ORB y BRISK que extienden el enfoque presentado

por BRIEF, añadiendo a sus métodos la invarianza a la rotación y a los cambios en la escala respectivamente.

Uno de los métodos más recientes y atractivos para la descripción binaria de la imagen es el citado LDB, en el que se basa ABLE para realizar la detección del bucle. Este descriptor, además de basarse en el valor de la intensidad de las imágenes, como hace BRIEF, calcula también el gradiente, dividiendo la imagen en varias celdas con el fin de obtener una mayor precisión en la descripción y poder realizar la fase de *matching* de una forma más rápida. Además, como mejora a este algoritmo, se ha presentado recientemente M-LDB, un descriptor que añade a LDB la invarianza a la escala y a la rotación y D-LDB, que introduce la disparidad entre imágenes (haciendo uso de un sistema estéreo), lo que permite realizar un reconocimiento visual de lugares mucho más robusto.

Como contribución, en este proyecto se ha realizado la detección del cierre de lazo en base a la teoría de ABLE, sobre la que se ha implementado la descripción de imágenes mediante el novedoso descriptor binario AKAZE, desarrollado por Pablo F. Alcantarilla, y que va a ser explicado más adelante.

La ventaja de todos estos descriptores radica en que sus elementos son binarios, lo que permite el ahorro de un gran espacio de memoria y tiempo de cómputo. Por ejemplo, un descriptor binario de 1024 elementos ocupa un espacio de memoria de 128 bytes, mientras que un GIST de 1024 elementos, ocupa 1024 bytes.

3.2.2. Similitud entre descriptores binarios: distancia de Hamming

La distancia o medida de similitud entre dos cadenas se refiere a la cantidad de diferencias que hay entre ellas, y es utilizada para solucionar diversos problemas.

En este trabajo, se calcula la distancia entre los descriptores de una imagen; cuanto menor sea la distancia, menor será el número de bits en que difieran (más "similares" serán). Por tanto, si esta distancia es baja, la probabilidad de que estén describiendo la misma imagen y nos encontremos ante una situación de cierre de lazo será elevada.

Existen diferentes métodos para calcular la similitud entre dos descriptores, como la distancia de edición o Levenshtein, la distancia con N-Gramas, la similitud de Smith-Waterman o la distancia de Hamming, calculada en nuestro método.

La idea de similitud entre descriptores es más consistente debido a la definición de la distancia de Hamming: sean X_i y X_j dos secuencias binarias de la misma longitud $i, j = 1, ..., K$, la distancia de Hamming entre ellas es el número de símbolos en que difieren. Siendo $W(X_i)$ el peso de Hamming de una secuencia y X_i el número de unos, la distancia de Hamming $d_{i,j} = d(X_i, X_j)$ está dada por:

$$d_{i,j} = W(X_i \oplus X_j) \tag{3.1}$$

En la Figura 3.5 se muestra un ejemplo básico del cálculo de distancias de Hamming utilizando palabras de la misma longitud (4 letras).

3.2.3. Matriz de similitud

Como ya se ha comentado, se determinará que una zona forma parte de un cierre de lazo cuando la distancia de Hamming calculada entre dos descriptores sea muy baja (próxima a 0).

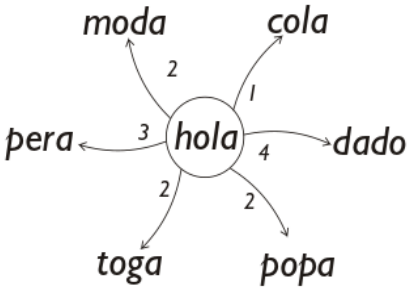


Figura 3.5: Ejemplo didáctico de distancias de Hamming entre palabras.

Por ello, cada vez que se procese una nueva imagen, se computa su descriptor y se calcula la distancia de Hamming con cada uno de los descriptores de las imágenes anteriormente procesadas, almacenando únicamente el valor mínimo de todas esas distancias en la denominada matriz de similitud.

Esta matriz se emplea, por tanto, para mostrar el grado de similitud entre un conjunto de patrones, en este caso, descriptores. Se construye una matriz, M, simétrica, de tamaño NxN, siendo N el número total de descriptores, que en el método ABLE coincide con el número de imágenes a procesar, pues se realiza una descripción global (un sólo descriptor por imagen).

Si un elemento de la matriz, $M(i, j)$, toma un valor cercano a 0, la similitud entre los descriptores i y j es muy elevada. En caso contrario, el valor de la distancia será cercano a la unidad. Esto es así ya que en ABLE, los elementos de esta matriz son normalizados, es decir, sus valores se dividen entre la máxima distancia calculada, de forma que toman un valor mínimo de 0 y máximo de 1.

A la hora de interpretar la matriz hay que tener en cuenta que la similitud entre descriptores consecutivos es muy alta y sin embargo, no representan un cierre de lazo, sino que este resultado es debido al solapamiento perceptual (dos imágenes capturadas consecutivamente representan prácticamente la misma escena y, por tanto, el descriptor de ambas es, con elevada probabilidad, el mismo punto).

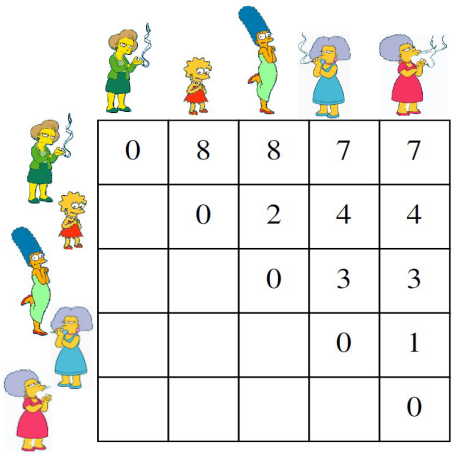
Debido al orden en que se almacenan los datos, esta matriz es simétrica respecto a su diagonal principal, cuyos elementos son iguales a 0 ya que representan la similitud de los descriptores respecto a sí mismos.

En la Figura 3.6 se muestra gráficamente la forma de una matriz de similitud. Se observa que los elementos de la diagonal principal valen 0, y que la matriz es simétrica respecto a esta diagonal. Además, cuanto más "similares" sean los personajes en apariencia, la distancia de similitud entre ellos será más baja.

3.3. Algoritmo ABLE

En [6], Roberto Arroyo presenta un método para la detección de cierre de lazo en imágenes estéreo, basada en el uso de descriptores binarios.

Este algoritmo emplea el descriptor LDB, que permite realizar una descripción robusta de la imagen, basada en la intensidad y gradiente de la misma. La propiedad de distintividad de LDB es superior a la de otros descriptores como BRIEF, basado únicamente en la diferencia de intensidad entre frames. Además, en este método también se propone el uso de la información de disparidad entre imágenes, a través del descriptor binario D-LDB. La adición de esta información



0	8	8	7	7
	0	2	4	4
		0	3	3
			0	1
				0

Figura 3.6: Ejemplo didáctico de matriz de distancias o similitudes.

en el proceso de descripción permite reducir algunos problemas típicos en el reconocimiento visual de lugares como el *perceptual aliasing*.

3.3.1. Construcción de los descriptores binarios

Este tipo de descriptores ya han sido explicados anteriormente, pero en esta sección se muestra cómo están construidos, es decir, las ecuaciones matemáticas que los modelan.

Centrándose en un *patch* o región, p, de la imagen, localizada en el punto de interés $x = (x, y)$, el test binario se define en la siguiente función:

$$\tau(p; f(i), f(j)) = \begin{cases} \text{si } f(i) < f(j) & 1 \\ \text{si } f(i) \geq f(j) & 0 \\ i \neq j; \end{cases} \tag{3.2}$$

Donde $f(i)$ es una función que devuelve la característica de un píxel en la región p, como puede ser la intensidad I de dicho píxel localizado en $x_i = (x_i, y_i)$, tal y como proporcionan los descriptores binarios BRIEF, ORB y BRISK:

$$f(i) = I(x_i) \tag{3.3}$$

La función $f(i)$ también puede proporcionar como información característica el promedio de las intensidades de varios píxeles, I_{avg}, o los gradientes Gx y Gy de una región c_i de p, tal y como proponen los descriptores binarios LDB y M-LDB:

$$f(i) = \{I_{avg}(c_i), G_x(c_i), G_y(c_i)\} \tag{3.4}$$

Con el objetivo antes mencionado de reducir el problema del solapamiento perceptual, este algoritmo propone como mejora al descriptor LDB, además del cálculo de I_{avg} y de los gradientes Gx y Gy, la disparidad entre frames, D_{avg}. Este nuevo descriptor se llama D-LDB.

$$f(i) = \{I_{avg}(c_i), G_x(c_i), G_y(c_i), D_{avg}(c_i)\} \tag{3.5}$$

Por último, el descriptor resultante, $d_n(p)$, es computado mediante un vector de n tests binarios, donde n también es la dimensión final del descriptor, ajustada empíricamente:

$$d_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p; f(i), f(j)) \tag{3.6}$$

3.3.1.1. Cálculo del descriptor binario

En la detección de cierre de lazo de ABLE se hace uso del descriptor binario LDB, por la mejora en el rendimiento que presenta respecto a otros descriptores como BRIEF gracias a la adición del cálculo del gradiente de las imagen en el proceso de descripción. Además, ABLE también permite integrar la disparidad en el proceso a través del descriptor D-LDB, que no se utiliza en la práctica en este proyecto ya que, para la detección del cierre de lazo, se realiza la descripción de una sola imagen (monocular). La disparidad es calculada mediante el algoritmo SGBM [47] (*Semi Global Block Matching*), a través de las librerías de OpenCV.

En este método no se precisa de imágenes de gran resolución para realizar un procesamiento adecuado de las mismas. En este caso, el tamaño del *patch*, p, es de 64x64 píxeles, y cualquier imagen de entrada es redimensionada a este tamaño antes de realizar la descripción. Con un tamaño menor de p se reduciría la efectividad del procesamiento y con uno mayor, no se aumentaría prácticamente la precisión.

La dimensión n del descriptor es de 256 bits (32 bytes), calculado mediante el método propuesto en [8].

El descriptor global se calcula tomando el centro de la imagen redimensionada, que es precisamente el punto de interés, sin tener en cuenta la rotación ni la escala del mismo.

Una alternativa a esta forma de calcular el descriptor, consiste en dividir la imagen en varias ventanas (mediante un *grid*), tomar como punto de interés o *keypoint* el centro de cada una de ellas y concatenar todos los descriptores para obtener uno único. ABLE permite configurar como parámetro de entrada el ancho y alto del *grid*.

3.3.1.2. *Matching* de los descriptores binarios

El objetivo de este método de detectar el cierre de lazo se puede realizar mediante la correlación de los elementos de v, un vector que contiene todos los descriptores binarios obtenidos de cada una de las imágenes procesadas, m.

La similitud entre los descriptores binarios se calcula mediante la distancia de Hamming, y esta información se almacena en la matriz M (matriz de similitud o de distancia).

$$v = (d_n(p1), d_n(p2), d_n(p3), ..., d_n(pm)) \tag{3.7}$$

$$M_{i,j} = M_{j,i} = POPCNT(v_i \oplus v_j) \tag{3.8}$$

Donde POPCNT es una instrucción vectorial de SSE4.2 que se utiliza para realizar un emparejamiento más rápido de los descriptores binarios.

3.3.2. Metodología de ABLE

La matriz M, calculada tras procesar una secuencia completa de imágenes estéreo, se normaliza según la Ecuación 3.9 :

$$M_{i,j} = \frac{M_{i,j}}{max(M)} \tag{3.9}$$

De esta forma se consigue que todos los valores de M varíen dentro del intervalo [0,1].

Posteriormente, se aplica un umbral θ a dicha matriz M con el fin de poder compararla con la matriz del *Ground Truth*, G, específica de cada secuencia y comprobar así la validez de los resultados. Una vez calculada y normalizada la matriz M, se concluye que nos encontramos ante una situación de cierre de lazo según la siguiente función:

$$Lazo = \begin{cases} \text{si } M_{i,j} < \theta & verdadero \\ \text{si } otro & falso \end{cases} \tag{3.10}$$

3.3.3. Evaluación de ABLE

Para evaluar los resultados obtenidos y poder realizar posteriormente una comparativa entre este método y otros descriptores, se obtienen las curvas de *precision-recall* a partir de la información de la matriz M. Estas curvas se utilizan para la medida del rendimiento o desempeño de los descriptores. Este enfoque se basa en el número de aciertos y fallos respecto de las correspondencias establecidas entre dos descriptores. Se calculan según las siguientes fórmulas:

$$P = \frac{tp}{tp + fp} \tag{3.11}$$

$$R = \frac{tp}{tp + fn} \tag{3.12}$$

Siendo *tp* los denominados verdaderos positivos, *fp* los falsos positivos y *fn* los falsos negativos.

Se varía linealmente el valor de θ entre 0 y 1, procesando un total de 100 valores para obtener unas curvas con suficiente información. Estas curvas van a ser mostradas con los resultados de nuestro método en el Capítulo 5, con el fin de evaluar el rendimiento de AKAZE en comparación con el resto de descriptores existentes en el estado del arte.

3.4. Implementación de la detección de cierres de lazo

En esta sección se explica la implementación de la detección de cierres de lazo mediante el método ABLE, y nuestra contribución en él, al realizar la descripción de las imágenes mediante el descriptor binario AKAZE.

En ABLE se hace uso de las librerías de OpenCV y de la librería pública de AKAZE, escritas en C++, con alguna sección de código relativa al proceso de emparejamiento o *matching*, escrita con instrucciones vectoriales SSE3, para reducir el tiempo de cómputo. El fichero principal, *test_able.cpp*, recibe como parámetro de entrada el fichero *config.txt*, donde se configuran una serie de parámetros:

1. Parámetros generales.
    a) Nombre del fichero donde se guarda la matriz de similitud en formato *txt*.
    b) Nombre del fichero donde se guarda la matriz de similitud en formato *png*.
    c) Color de la matriz de similitud.
    d) Posibilidad de mostrar las imágenes que son procesadas por el método.
2. Parámetros para la descripción y *matching*.
    a) Posibilidad de realizar una descripción global o una descripción basada en un *grid*.
    b) Posibilidad de una descripción invariante a la iluminación
    c) Número de bits del *patch*, en caso de realizar una descripción global.
    d) Tamaño del grid en el eje x (ancho de la ventana).
    e) Tamaño del grid en el eje y (alto de la ventana).
    f) Descriptor a utilizar en el proceso. En nuestro caso será AKAZE.
    g) Número de imágenes a procesar en cada iteración (monocular o estéreo).
3. Parámetros del dataset.
    a) Posibilidad de procesar una secuencia de imágenes, un vídeo o dos vídeos.
    b) Directorio donde se encuentra el dataset a testear.
    c) Formato de las imágenes del dataset. En nuestro caso, se testea el método con imágenes en formato *png*.
    d) Información adicional en caso de testear dos vídeos.
    e) Identificador o nombre de la primera imagen del dataset.
    f) Identificador o nombre de la última imagen del dataset.

Una vez configurados todos los parámetros, el fichero principal hace una llamada al método **Able::compute_ABLE()**, perteneciente a la clase *Able*, que lleva a cabo todo el proceso de detección de cierre de lazo. Una vez finalizado el algoritmo, el programa muestra los tiempos de cómputo más relevantes: tiempo total en describir todas las imágenes, tiempo total en el proceso de *matching* y tiempo medio en describir una imagen. Además, se guarda la matriz de similitud obtenida en formato *png* y en formato *txt*, para poder evaluar los resultados obtenidos con diferentes descriptores.

En la clase Able se declaran todos los parámetros anteriormente enumerados: los generales, los que configuran las fases de descripción y *matching* y los relativos al dataset. Además, también se encuentra definido el constructor **Able::Able(char* config_name)** y todos los métodos necesarios para realizar la detección de cierre de lazo, que se explican a continuación.

En el fichero *Able.cpp*, el constructor **Able::Able(char* config_name)** abre el fichero *config.txt* y lee todos los parámetros de configuración. El método **Able::global_description(Mat image)** realiza la descripción global de la imagen (obtiene un descriptor por cada imagen que procese). Para ello, se redimensiona la imagen de entrada (en escala de grises) a un tamaño de 64 píxeles y se extrae el píxel central de la imagen, que será su descriptor global.

En el fichero de configuración se permite al usuario elegir el tipo de descriptor a utilizar en el proceso. En este proyecto se han implementado las funciones del descriptor binario AKAZE, que

antes de realizar la descripción de la imagen, necesita crear un espacio de escala no lineal y convertir la imagen a un formato concreto: CV32FC1. El método **Able::global_description(Mat image)** devuelve como parámetro de salida el descriptor global de la imagen, de tipo *Mat*.

El método **Able::grid_description(Mat image)** realiza la descripción de la imagen haciendo uso de un *grid* (descripción jerárquica), de forma que la imagen se va recorriendo con una ventana, cuyo alto y ancho son definidos por el usuario. Entonces, se obtiene un descriptor por cada ventana, que se corresponde con el píxel central de esta. Nuevamente, al realizar el proceso de descripción con AKAZE, la imagen se convierte al formato CV32FC1 y se crea un espacio de escala no lineal antes de obtener los descriptores. Por último, se concatenan todos los descriptores obtenidos, de forma que se obtenga un único descriptor de tipo *Mat*.

Una vez que se obtiene el descriptor de la imagen que se esté procesando, el siguiente paso es calcular la distancia de similitud entre dicho descriptor y los previamente calculados. Esta función se realiza en el método **Able::hamming_matching(Mat desc1, Mat desc2)**. Este método es válido en caso de que se trabaje con descriptores binarios del mismo tamaño, como sucede con AKAZE. En este caso, devuelve la distancia Hamming, de tipo *int*, entre el descriptor actual y todos los obtenidos anteriormente. En caso de trabajar con descriptores vectoriales, el proceso de emparejamiento entre ellos se realiza mediante el método **Able::l2norm_matching(Mat desc1, Mat desc2)**.

Posteriormente, la menor de todas las distancias calculadas en cada iteración se almacena en la llamada matriz de similitud, que es la que se utiliza para evaluar los resultados. El método **Able::similarity_matrix_to_txt()** guarda la matriz de similitud en formato *txt* y el método **Able::similarity_matrix_to_png()** lo hace en formato *png*. Esta matriz es normalizada a través del método **Able::similarity_matrix_normalitation()**, donde cada elemento de dicha matriz es dividido por el máximo valor obtenido.

El algoritmo de detección de cierre de lazo es implementado en **Able::compute_ABLE()**, donde se integran todos los métodos explicados de forma secuencial. En él se inicializan todas las variables necesarias y se carga la secuencia de vídeo a procesar. Entonces, se realiza un bucle en el que se va leyendo cada una de las imágenes del dataset. En caso de utilizar un dataset con secuencias de vídeo estéreo, como KITTI, únicamente se procesan las imágenes capturadas por una de las dos cámaras.

Como ya se ha explicado en el capítulo anterior, la detección del cierre de lazo se determina cuando un número considerable de elementos de la matriz de similitud sean inferiores a un umbral fijado, lo que significa que una gran cantidad de puntos de interés detectados en un intervalo de frames coinciden con aquellos detectados en un intervalo de frames anteriormente procesados.

También se pueden evaluar los resultados del método de forma gráfica, a través de la matriz de similitud guardada en formato *png*, donde se representan en diferente color los frames que forman un cierre de lazo. Esto será mostrado en el Capítulo 5 de Resultados.

3.4.1. Implementación de AKAZE en la detección de cierre de lazo

El método ABLE realiza una detección de cierre de lazo haciendo uso del descriptor binario LDB, que como ya se ha explicado, se basa en el valor de intensidad y gradiente de las imágenes.

En este proyecto, se propone la implementación del nuevo descriptor binario AKAZE en este proceso de detección de cierre de lazo. Para ello, se ha hecho uso de la librería pública de AKAZE, desarrollada por Pablo F. Alcantarilla.

Los métodos anteriores para detectar y describir las características en espacios con diferentes niveles de escala, hacen uso de un espacio de escala gaussiano. Este desenfoque gaussiano no respeta los límites naturales de los objetos, suavizando en la misma medida, tanto el ruido como los detalles de la imagen, reduciendo así la precisión en la localización. En cambio, el descriptor KAZE [48] realiza la detección y descripción de características 2D mediante el filtrado de difusión no lineal. De esta manera, se hace un suavizado local y adaptativo de la imagen, reduciendo el ruido y conservando los bordes naturales de los objetos al mismo tiempo, obteniendo así mejores resultados.

El principal inconveniente de KAZE es que es computacionalmente intenso. No existen soluciones analíticas para resolver ecuaciones de difusión no lineal, por lo que se necesitan métodos numéricos para aproximar una solución. Inicialmente esto se realizaba mediante el método AOS (*Additive Operator Splitting*). Este operador es muy estable al paso del tiempo pero necesita resolver un largo sistema de ecuaciones lineales para obtener una solución, lo que hace que KAZE sea poco útil en aplicaciones en tiempo real. Por ello, como mejora a este problema se propuso posteriormente el uso de un nuevo método matemático, denominado FED (*Fast Explicit Diffusion*), que permite una construcción de un espacio de escala no lineal mucho más rápido que cualquier otro método, siendo más fácil de implementar y mucho más exacto que AOS.

La mejora previamente comentada sobre el descriptor KAZE dio lugar al nuevo descriptor AKAZE (Accelerate-KAZE), que conserva los beneficios del filtrado de difusión no lineal empleando un tiempo de cómputo mucho menor que su predecesor. Además del uso del método FED para la detección de puntos característicos, se introduce una mejora adicional: al descriptor LDB, en el que se basa KAZE, se le añade la información del gradiente de las imágenes en el espacio de escala no lineal, consiguiendo así invarianza en la escala y en la rotación, apareciendo así un nuevo descriptor denominado M-LDB, en el que se basa AKAZE para realizar las fases de detección y descripción de las imágenes.

Capítulo 4

Corrección de la odometría visual

4.1. Motivación

Como se comentó en el Capítulo 2, la odometría visual no es una forma completamente fiable de localizar a un robot, ya que basa los cálculos en fuentes de información que no son totalmente precisas. Por tanto, proporciona una forma sencilla de estimar la posición del robot, pero se van acumulando errores, tanto sistemáticos como no sistemáticos. Esto se traduce en que, en trayectorias largas, de varias decenas de metros, la estimación de la posición por odometría deja de ser fiable. Como consecuencia, se necesita buscar la forma de aprovechar esta información y corregir la estimación para conseguir un sistema de localización del móvil que funcione correctamente.

4.2. Integración de ABLE en LIBVISO

Hasta el momento se ha definido, por un lado, el algoritmo de ABLE y por otro, la odometría visual realizada por Andreas Geiger en LIBVISO. En este proyecto, como objetivo previo a la corrección de la odometría, se encuentra la detección de cierres de lazo en los mapas generados por dicha odometría, pues será a partir de esta información cuando se procederá a realizar el remapeado de la trayectoria seguida por el móvil. La integración de ambos algoritmos queda explicada a continuación:

En el fichero principal del código de LIBVISO se inicializa, en primer lugar, el método ABLE, pasando como argumento el fichero de configuración *config.txt*, para que la detección del cierre de lazo se realice con los parámetros deseados. En este caso, la detección y descripción global de cada imagen se realiza mediante AKAZE. Como segundo parámetro de entrada se introduce la secuencia de vídeo sobre la que se quiere detectar el bucle y realizar la corrección.

Como ya se ha comentado, a continuación, se pasan los parámetros de calibración de las cámaras y los parámetros más importantes de la odometría visual estéreo. Posteriormente, en cada iteración de un bucle *for* se realiza la lectura de cada par de imágenes de la secuencia elegida.

Después de realizar los pasos anteriores, centrándonos únicamente en la imagen tomada por una de las cámaras (en este caso, se elige procesar la imagen de la cámara derecha), se procede a la descripción global de la misma. Para poder hacer uso de esa información más tarde, el descriptor de cada imagen se va almacenando en el vector *updatedescriptors* mediante la función *pushBack*.

Se hace, entonces, una llamada al método *process*, al que se le pasa como argumento las imágenes capturadas por el sistema de cámaras, que tras realizar cada una de las fases propias de la odometría visual, devuelve, a través de *getMotion*, la matriz esencial de rotación y traslación, denominada *pose*.

Con el fin de poder representar los resultados de la odometría de forma gráfica, los resultados obtenidos se van guardando en diferentes ficheros *.txt*. En uno de los ficheros se almacena la matriz *pose* de dimensiones 4x4, y en otro, únicamente la parte correspondiente a la traslación, el vector (x, z), información que va a ser utilizada para representar el mapa en un plano de dos dimensiones.

A continuación, se calcula la distancia Hamming entre el descriptor de la imagen actual y cada uno de los descriptores calculados en iteraciones pasadas. De todas estas distancias calculadas, se guarda de forma progresiva en el vector *minDistance* únicamente el mínimo valor de similitud calculado entre dos imágenes, así como el frame con el que se corresponde ese mínimo valor.

Para que este método sea trasladable a cualquier secuencia de imágenes cuya odometría visual se quiera corregir, se realiza una normalización de cada valor de similitud, dividiendo el mismo entre un umbral establecido. Entonces, se procede a la detección del cierre de lazo sobre la odometría estimada. Para ello, se comprueba qué elemento guardado en el vector que contiene todos los valores de mínima similitud es menor que σ, ya que en ese caso, se concluirá que el frame actual ha sido previamente visitado, pues la similitud entre su descriptor y uno anterior es lo suficientemente baja. En caso de que ese emparejamiento se trate de un *outlier*, el método RANSAC se encarga de eliminarlo. Se almacenan de nuevo los resultados obtenidos tras este proceso, que se representarán gráficamente haciendo uso del programa de cálculo, Matlab.

A continuación, se explican las posibles correcciones de la odometría visual. Todas ellas se realizan una vez que se han detectado todos los cierres de lazo presentes en la secuencia de vídeo procesada. La primera corrección ha sido desarrollada en este trabajo en su totalidad, mientras que las otras dos están basadas en algoritmos de optimización de la trayectoria existentes en el estado del arte.

4.3. Corrección basada en la detección de la deriva incremental usando cierres de lazo

Este método de corrección ha sido implementado en este proyecto íntegramente, y se basa en la matriz de similitudes calculada por el algoritmo ABLE, pues, como ya se ha comentado, la optimización de la trayectoria se llevará a cabo una vez que se ha detectado un cierre de lazo, es decir, cuando un elemento de dicha matriz sea menor a un umbral establecido.

Este enfoque realiza diferentes correcciones de la deriva acumulada en el resultado de la odometría visual, dependiendo de la parte de la trayectoria que se pretenda reconstruir:

- Corrección de las poses pertenecientes a un cierre de lazo.
- Corrección de las poses posteriores a un cierre de lazo.
- Corrección de las poses anteriores a un cierre de lazo.

Estos tres tipos de correcciones se aplican cada vez que un nuevo cierre de lazo es detectado, pues dependiendo de la secuencia de vídeo a procesar el número de bucles que presenten será diferente.

4.3.1. Frames pertenecientes al cierre de lazo

Según la Ecuación 3.10 del método ABLE, se determina que en la trayectoria recorrida por un vehículo se da un cierre de lazo cuando algún elemento de la matriz de similitud, $M_{i,j}$ es menor que un umbral, θ. En este caso, se trata de un umbral global, estimado tras testear el algoritmo con distintas secuencias del dataset de KITTI para odometría, con el fin de encontrar un umbral trasladable a cualquier secuencia a procesar. Se recuerda que los elementos de la matriz de similitudes, M, son normalizados previamente, es decir, son divididos por el máximo valor calculado, de forma que todos ellos varían dentro del intervalo [0,1].

Una vez que se ha determinado que un frame forma parte de un cierre de lazo, es decir, su descripción representa la misma zona que otra descripción obtenida anteriormente, se procede a la corrección de su pose o posición de la cámara en el plano xz. Para ello, primero se calcula la desviación producida entre los dos puntos de interés citados:

$$\Delta x = |x(i) - x(j)| \tag{4.1}$$

$$\Delta z = |z(i) - z(j)| \tag{4.2}$$

Siendo $x(i)$ la posición actual de la cámara en el eje x y $x(j)$ la posición de la cámara en el instante de tiempo en que se extrajo el descriptor por primera vez (o el vehículo visitó la misma zona por primera vez). Análogamente, $z(i)$ y $z(j)$ representan las poses del móvil en el eje z la primera y segunda vez que atraviesa el mismo lugar, respectivamente. La desviación en el eje y no se tiene en cuenta en este problema, pues se considera que la cámara sujeta en la parte superior del vehículo se mantiene siempre a una altura similar, por lo que el error en la estimación en ese eje es despreciable.

Una vez calculadas las desviaciones producidas sobre el plano, se actualizan las poses del robot, según las siguientes ecuaciones:

$$x(i)' = x(i) + \Delta x \tag{4.3}$$

$$z(i)' = z(i) + \Delta z \tag{4.4}$$

siendo $x(i)'$ y $z(i)'$ las poses actualizadas en los ejes x y z, en la zona del cierre de lazo.

Además, se calcula la desviación media entre las poses de los descriptores extraídos en el momento actual (cuando el robot vuelve a visitar una zona por segunda vez) y las de aquéllos con los que se relacionan por estar describiendo el mismo frame. Esta información va a ser utilizada posteriormente para corregir las poses en el resto de la trayectoria.

$$x_m = \frac{\sum_{i=1}^{m} \Delta x}{m} \tag{4.5}$$

$$z_m = \frac{\sum_{i=1}^{m} \Delta z}{m} \tag{4.6}$$

siendo x_m y z_m las desviaciones medias producidas en la zona del cierre de lazo en los ejes x y z y m el número de frames que forman parte de dicha zona.

4.3.2. Frames posteriores al cierre de lazo

Como ya se ha comentado en más de una ocasión, este método corrige, en primer lugar, las poses que forman parte del cierre de lazo y, a continuación, a partir de la información extraída de dicha región, el resto de poses.

Una vez optimizada la región del bucle, las poses de la cámara en los frames posteriores se modifican según la siguiente ecuación:

$$x(i)' = x(i) + x_m \tag{4.7}$$

$$z(i)' = z(i) + z_m \tag{4.8}$$

siendo, nuevamente, $x(i)'$ y $z(i)'$ las poses corregidas.

Cabe destactar que este algoritmo se lleva a cabo cada vez que se detecta un nuevo cierre de lazo. Por lo tanto, cuanto mayor sea el número de bucles contenidos en la trayectoria que realiza el robot, más veces se ejecuta el método propuesto y, como consecuencia, más precisa es la corrección realizada, obteniendo como resultado un mapa optimizado muy próximo al mapa trazado a partir de las exactas medidas del GPS.

4.3.3. Frames anteriores al cierre de lazo

Mientras que las dos correcciones explicadas anteriormente se pueden implementar en tiempo real, pues una vez que se detecta un cierre de lazo la rectificación de las poses es inmediata, la optimización de la trayectoria que el robot ha ido recorriendo previamente a detectar dicho cierre de lazo se lleva a cabo de manera offline.

Para ello, de nuevo se hace uso de la información extraída de la región del bucle, concretamente de la desviación media producida entre todas las poses, dato que se suma, de manera proporcional, al valor de las posiciones a corregir, ya que la deriva acumulada en las poses iniciales es mucho menor que en aquellas estimadas a medida que el número de metros recorridos aumenta (cuanto mayor sea la ruta realizada por el vehículo, mayor es también el error acumulado en la estimación del movimiento).

Se calcula, en primer lugar, el factor de corrección:

$$c_x = \frac{x_m}{n} \tag{4.9}$$

$$c_z = \frac{z_m}{n} \tag{4.10}$$

siendo n el número de frames a corregir y c_x y c_z los factores de corrección a aplicar en la corrección de las poses en los ejes x y z.

Entonces, se corrige, de forma sucesiva, el valor de cada pose, comenzando con aquellas que fueron estimadas inicialmente, hasta recalcular toda la trayectoria recorrida antes de que el cierre de lazo fuera detectado:

$$x(i)' = x(i) + c_x\dot{k} \tag{4.11}$$

$$z(i)' = z(i) + c_z\dot{k} \tag{4.12}$$

siendo k una variable que se incrementa cada vez que una nueva pose es actualizada, de manera que la corrección aplicada en las primeras poses sea menor que en las últimas.

Tras la realización de esta etapa, se obtiene una reconstrucción completa del mapa proporcionado inicialmente por la odometría visual, con la que se consigue reducir de forma significativa el error acumulado en la primera estimación. Los resultados de este método se muestran gráficamente en el Capítulo 5.

4.4. Corrección basada en GraphSlam

Tanto este método como el propuesto por Levenberg-Marquardt se adaptan perfectamente al objetivo de este proyecto de corregir el resultado inicial de la odometría. Son algoritmos ya implementados y existentes en el estado del arte y se procede a su explicación teórica.

GraphSlam [20] es un algoritmo de optimización de la trayectoria obtenida mediante odometría visual. Al igual que el método anterior, puede servir para corregir el recorrido realizado por el robot partiendo de la zona de cierre de lazo previamente detectada. Utiliza la información contenida en una matriz dispersa (matriz de gran tamaño en la que la mayor parte de sus elementos son cero), obtenida tras la representación del problema como un grafo de interdependencias (dos poses están relacionadas si "ven" la misma marca o baliza).

El algoritmo resuelve el SLAM completo (localización y mapeado simultáneos) y para ello, representa el problema como un grafo que conduce a una suma de restricciones cuadráticas no lineales. Los nodos del grafo representan las poses del robot y las balizas o marcas que el sistema de cámaras captura en cada instante de tiempo, en la Figura 4.1 son (x_1, x_2, x_3, x_4) y (B_1, B_2, B_3, B_4) respectivamente. Los arcos del grafo pueden ser, bien de movimiento, que son aquellos que unen dos poses consecutivas del móvil, o bien de medida, que unen las poses con las marcas que fueron vistas desde cada posición. Además, cada línea del grafo se corresponde con una restricción no lineal, cuya suma representa un problema no lineal de mínimos cuadrados.

Las restricciones entre las poses se introducen por medio de odometría visual, reduciendo el grafo a sus nodos más significativos (denominados *keyframes*). Dichas restricciones son optimizadas para proceder posteriormente al remapeado de la trayectoria, en base al criterio de máxima similitud y un conjunto de poses correspondientes.

Para el cálculo de la probabilidad a posteriori del mapa, GraphSlam realiza una linealización del conjunto de restricciones, que da como resultado una matriz y un vector de información. Para ello se emplean técnicas estadísticas de inferencia. Debido a la baja densidad del grafo construido, dicha matriz de información es dispersa, lo que permite aplicar un algoritmo para la eliminación de variables y transformar así el grafo inicial en uno más pequeño que sólo contenga las poses del robot (Figura 4.2).

La ventaja de este algoritmo radica en la representación de la información, basada en la adición de restricciones al grafo, lo que tiene un coste computacional muy bajo y permite adquirir mapas de un orden de magnitud elevado.

GraphSlam genera mapas más fiables y precisos que otros métodos de optimización como EKF-SLAM y para ello requiere de una fase de inferencia al final del algoritmo, con el fin de recuperar el mapa y las poses (SLAM completo).

Se enumeran a continuación, las fases de las que consta este algoritmo de corrección de la odometría visual:

- Representación del problema como un grafo.
  - Los nodos representan las poses del robot y las marcas.
  - Los arcos son de movimiento y de medida.
- Construcción de la matriz de información a partir del grafo de dependencias entre poses y marcas (restricciones introducidas mediante odometría visual).
  - Almacenamiento progresivo de la información de dependencias entre distintas poses (arcos de movimiento).

  - Almacenamiento progresivo de la información de dependencias entre una marca y una o varias poses desde donde es vista (arcos de medida).

- Reducción de la matriz de información. Obtención de una matriz que sólo contenga información de los arcos de movimiento.
  - Eliminación de la información de los arcos de medida.
  - Modificación de las dependencias entre poses.
  - Adición de nuevas dependencias entre poses.

- Optimización y corrección de la trayectoria en base a la nueva matriz de información.

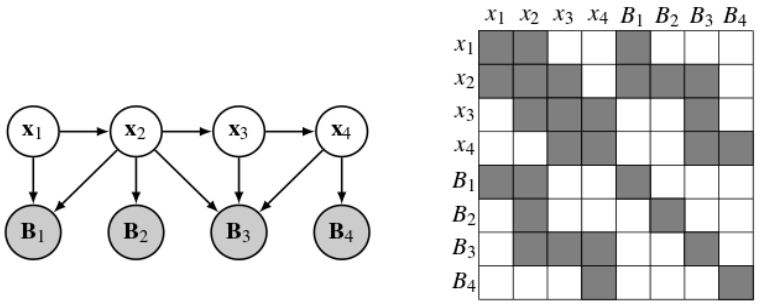


Figura 4.1: Dependencias completas.

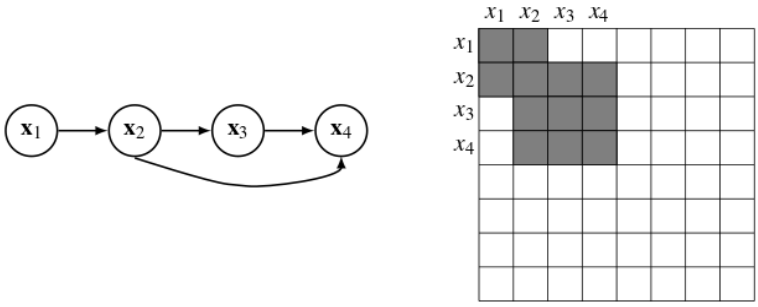


Figura 4.2: Dependencias reducidas.

4.5. Corrección basada en Levenberg-Marquardt

Se comienza esta sección explicando en qué consiste este método de optimización de forma teórica, para después proceder a su posible implementación en un problema concreto de la corrección de la odometría visual.

Levenberg-Marquardt es uno de los algoritmos de optimización más extendidos en la actualidad. Se trata de una combinación de los métodos Newton-Gauss y Descenso de gradiente, desarrollados para minimizar funciones que sean la suma de los cuadrados de otras funciones no lineales. Levenberg-Marquardt trata, por tanto, el denominado problema de mínimos cuadrados no lineales. Esto implica que la función a minimizar u optimizar sigue la ecuación:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x) \tag{4.13}$$

donde $x = (x_1, x_2, ..., x_n)$ es un vector de n elementos y r_j es una función que va de R^n a R y representa los denominados *residuos*. Se asume, además, que $m \geq n$.

Para expresar el problema de forma más sencilla, la función f se expresa como un vector de *residuos*, r: $R^n \rightarrow R$, según la ecuación:

$$r(x) = (r_1(x), r_2(x), ..., r_m(x)) \tag{4.14}$$

De esta forma, $f(x)$ se puede expresar también como: $f(x) = \frac{1}{2}||r(x)||^2$.

Una vez que se ha definido el problema, las derivadas primera (matriz Jacobiana) y segunda (matriz Hessiana) de las funciones no lineales se calculan, respectivamente, como:

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x) \nabla r_j(x) = J(x)^T r(x) \tag{4.15}$$

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) \tag{4.16}$$

La característica de los problemas de mínimos cuadrados es que, en caso de que r_j se pueda expresar como una función lineal, o que el valor de los *residuos* ($r_j(x)$) sea reducido, la matriz Hessiana se puede calcular de forma directa a partir de la matriz Jacobiana:

$$\nabla^2 f(x) = J(x)^T J(x) \tag{4.17}$$

Para resolver el problema de mínimos cuadrados no lineales existen diferentes métodos. Por una parte, el método de Descenso del gradiente es el más sencillo e intuitivo para encontrar el mínimo de la función, pero presenta importantes problemas de convergencia. Para evitarlos, se puede hacer uso de la información del gradiente o segunda derivada de la función, que es precisamente lo que emplea el método de Newton-Gauss, que resuelve la ecuación $\nabla f(x) = 0$ a partir de su aproximación de Taylor:

$\nabla f(x) = \nabla f(x_0) + (x - x_0)^T \nabla^2 f(x_0) +$ *términos de orden superior de* $(x - x_0)$.

Si se desprecian los términos de orden superior y se calcula el mínimo valor de x igualando la ecuación anterior a 0, se obtiene la expresión que modela el método de Newton-Gauss:

$$x_{i+1} = x_i - (\nabla^2 f(x_i))^{-1} \nabla f(x_i) \tag{4.18}$$

donde x_0 es sustituido por x_i y x por x_{i+1} en la aproximación de Taylor.

La ventaja de este método radica en su rapidez. Sin embargo, se trata de un método abierto, lo que quiere decir que la única manera de alcanzar la convergencia es seleccionando un valor inicial lo suficientemente cercano a la raíz buscada, algo que no siempre es posible.

Con el objetivo de explotar las ventajas de los dos métodos descritos, se propone el algoritmo de Levenberg-Marquardt, un proceso iterativo de convergencia a través de la técnica de los mínimos cuadrados ponderados, que está basado en la siguiente expresión, resultado de la combinación de los dos métodos anteriores:

$$x_{i+1} = x_i - (H + \lambda I)^{-1} \nabla f(x_i) \tag{4.19}$$

siendo H la matrix Hessiana evaluada en x_i, e I la matriz identidad. Según este método si tras una iterar esta ecuación una vez el error disminuye, significa que la hipótesis de que la

función $f(x)$ es cuadrática es correcta. Entonces, se reduce el valor de λ (normalmente en un factor de 10), con el fin de disminuir la influencia del gradiente en la Ecuación 4.19. Por tanto, lo que se busca es que el valor de λ disminuya cuando el punto x_i está cerca de la solución. Sin embargo, en caso de que el valor de λ sea muy elevado, la influencia de la matriz Hessiana es muy baja. Por ello, Marquardt propuso reemplazar en la Ecuación 4.19 la matriz identidad por la diagonal principal de la matriz Hessiana. De esta forma, cada elemento del gradiente es ponderado por el factor λ, en función de la curvatura de la función en x_i:

$$x_{i+1} = x_i - (H + \lambda diag[H])^{-1} \nabla f(x_i) \tag{4.20}$$

A continuación, se muestra cómo emplear, de forma intuitiva, el algoritmo Levenberg-Marquardt [49] en la corrección u optimización del resultado inicial proporcionado por la odometría visual.

En una odometría visual estéreo, como la que se lleva a cabo en este proyecto, es necesario un proceso de *matching* temporal, de forma que se relacione el punto característico, x, de la imagen actual con ese mismo punto en la imagen adquirida un instante de tiempo anterior, x'. La proyección de x' se expresa a partir de la matriz de homografía, o de transformación homogénea, explicada en capítulos anteriores:

$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = H(R,t) \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{4.21}$$

En esta etapa es necesario aplicar un proceso de optimización y, para ello, se hace uso del algoritmo Levenberg-Marquardt. El objetivo es minimizar la distancia de error entre el punto característico proyectado, x' y el medido, x. La función a minimizar sigue la siguiente expresión:

$$min \sum_{k=1}^{N} d(x, x') \tag{4.22}$$

Tras realizar las etapas de las que consta el método de Levenberg-Marquardt, se obtiene la matriz de rotación y el vector de traslación (pose) que optimizan la función anterior.

Generalmente, este algoritmo converge en unas 40 iteraciones, en un tiempo total de aproximadamente 45 milisegundos. Sin embargo, en la corrección de la odometría visual, el número de iteraciones necesarias depende del número de correspondencias y de la localización del punto de interés en la imagen.

Capítulo 5

Resultados

5.1. Dataset de KITTI

Tanto para obtener y evaluar los resultados del método ABLE como los de LIBVISO junto con las correcciones de la odometría visual, se ha decidido hacer uso del dataset de KITTI para odometría [4].

Este dataset está formado por 22 secuencias de vídeo que contienen un total de 44182 imágenes (39,2 km). Estas secuencias incluyen entornos urbanos con diferentes características, como el *perceptual aliasing* o solapamiento perceptual, cambios de iluminación, etc. Todas las secuencias de vídeo están grabadas con un sistema de cámaras estéreo fijado a la parte superior de un vehículo que va recorriendo las calles (véase Figura 5.1). Como ya se ha explicado, a la hora de implementar la detección de cierre de lazo en ABLE, únicamente se procesan las imágenes capturadas por una de las dos cámaras.

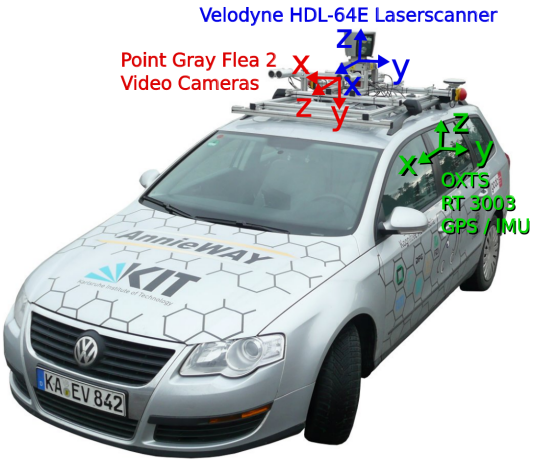


Figura 5.1: Vehículo con el que se han grabado todas las secuencias de KITTI.

Aunque este dataset no proporciona un *Ground Truth* específico para la detección de cierre de lazo, Roberto Arroyo lo incluyó en [6], y también se ha hecho uso de él en este proyecto para evaluar los resultados obtenidos con nuestro método. (Véase la Tabla 5.1). El *Ground Truth* contiene la posición y orientación exacta del vehículo, obtenidas con un sistema de posicionamiento global altamente fiable.

Centrándonos en nuestro propósito de corregir la trayectoria a partir de la detección de un

bucle, este dataset incluye 12 secuencias con cierres de lazo, 21 de los cuales son unidireccionales y 4 bidireccionales (se dice que se ha detectado un cierre de lazo bidireccional cuando la misma zona es atravesada por segunda vez en dirección contraria).

Como se comentó a la hora de hablar de la fase de adquisición de imágenes, una buena imagen de entrada al algoritmo de odometría visual debe tener suficiente textura, estar bien iluminada y representar una escena mayormente estática. Además, los sucesivos frames tienen que solaparse. Por tanto, como se muestra en la Figura 5.2, las imágenes del dataset elegido presentan características ideales para testear nuestro método y evaluar los resultados obtenidos. A modo de ejemplo, se muestran 8 frames de la secuencia 06 de KITTI: los 4 primeros (del 0 al 3) representan la misma zona que 4 los últimos (del 835 al 838), y marcan el inicio del cierre de lazo. Es decir, a partir del frame 835 el trayecto se recorre por segunda vez, por lo que las imágenes capturadas (y sus descriptores) son las mismas. Además, este dataset proporciona los parámetros extrínsecos y la matriz de calibración de las cámaras con las que fueron grabadas las secuencias de vídeo, datos necesarios para realizar la odometría visual con el algoritmo LIBVISO.

(a) Frame 00

(b) Frame 01

(c) Frame 02

(d) Frame 03

(e) Frame 835

(f) Frame 836

(g) Frame 837

(h) Frame 838

Figura 5.2: Frames representativos de la secuencia 06 del dataset de KITTI.

5.2. Resultados de odometría visual iniciales con LIBVISO

Antes de mostrar la detección de cierre de lazo sobre los mapas proporcionados por la odometría, se representa gráficamente el resultado inicial de la misma sin hacer ningún tipo de corrección, y se compara con el mapa de referencia obtenido a partir de las medidas de precisión del GPS. En la Figura 5.3 se presenta el resultado inicial de procesar la secuencia 06 del dataset de KITTI mediante el algoritmo LIBVISO. En ella se representa sobre el plano (x, z) y en color rojo el mapa que proporciona el sistema de posicionamiento global, mucho más preciso y fiable que el obtenido tras realizar la odometría visual de la secuencia, representado en color azul.

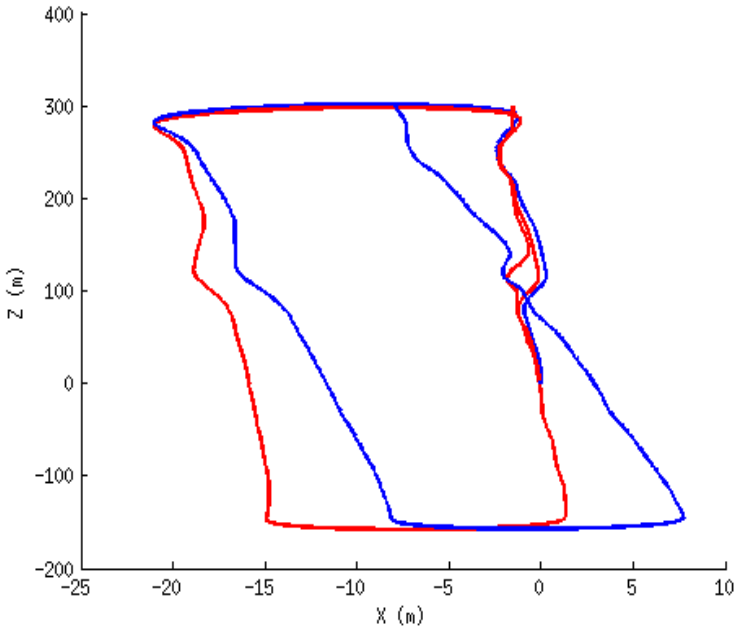


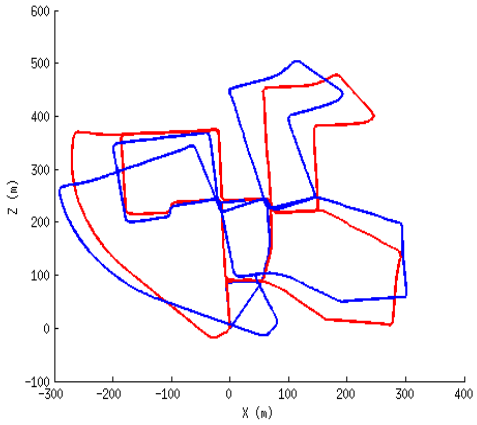
Figura 5.3: Resultado de la odometría visual no corregida en la secuencia 06 de KITTI.

El resultado de la odometría visual muestra la deriva acumulada en la estimación de la trayectoria, que se va incrementando a medida que pasa el tiempo y aumenta la distancia recorrida por el vehículo. De hecho, en la representación de la secuencia mostrada en la Figura 5.3, los últimos frames forman parte de un cierre de lazo, y, sin embargo, éste no aparece representado como tal en el mapa obtenido con el método LIBVISO. Como se describe en la Tabla 5.1, esta secuencia contiene un total de 1101 imágenes y el cierre de lazo está representado entre los intervalos de frames (000-280) y (835-1093). En la práctica, esto significa que los primeros 300 metros recorridos (aproximadamente) son atravesados de nuevo (en la misma dirección) en la última parte de la secuencia.

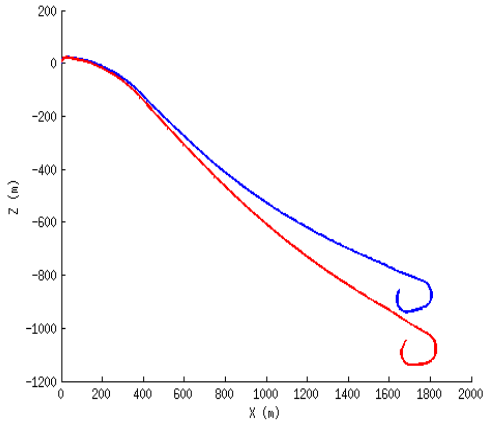
La deriva incremental que se da en el resultado de la odometría no aparece en el mapa obtenido a partir de las medidas de satélites (GPS), representado en color rojo en la Figura 5.3. Las líneas que representan la zona que es recorrida en dos ocasiones distintas, prácticamente se solapan. Además, en este ejemplo en concreto, en la zona en la que se da el cierre de lazo, el vehículo realiza un adelantamiento a una bicicleta, razón por la cual hay una pequeña desviación en la representación.

En las Figuras 5.4 y 5.5 se muestran más comparativas entre el resultado de la odometría visual sin corregir, representado en color azul, y el *Ground Truth*, o mapas obtenidos a partir de

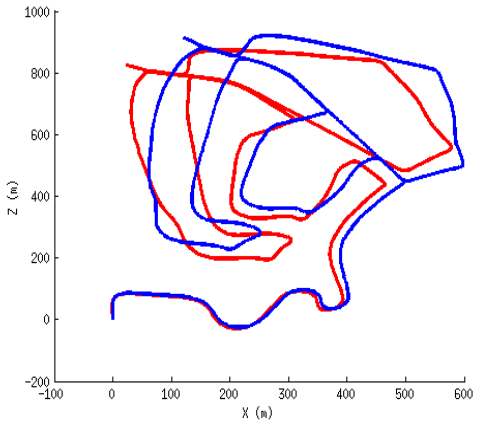
las medidas del GPS, en color rojo. De nuevo, se visualiza la deriva acumulada en la estimación de la trayectoria por odometría visual. Se trata de un error acumulativo, por lo que se va incrementando y se acentúa en recorridos más largos.



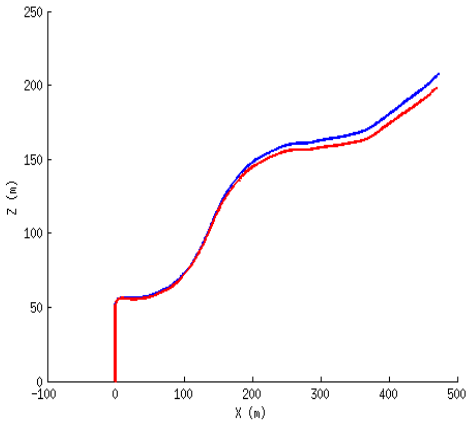
(a) Secuencia 00



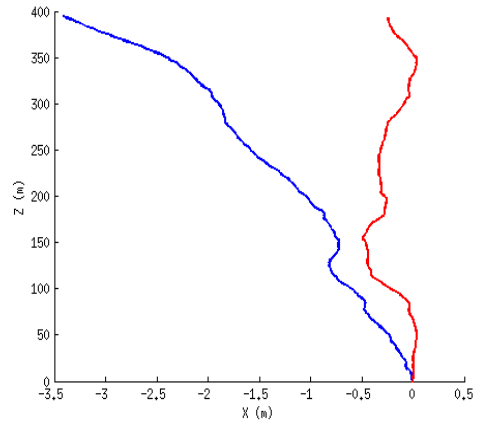
(b) Secuencia 01



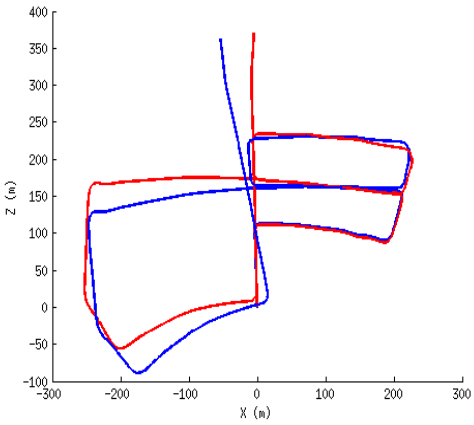
(c) Secuencia 02



(d) Secuencia 03

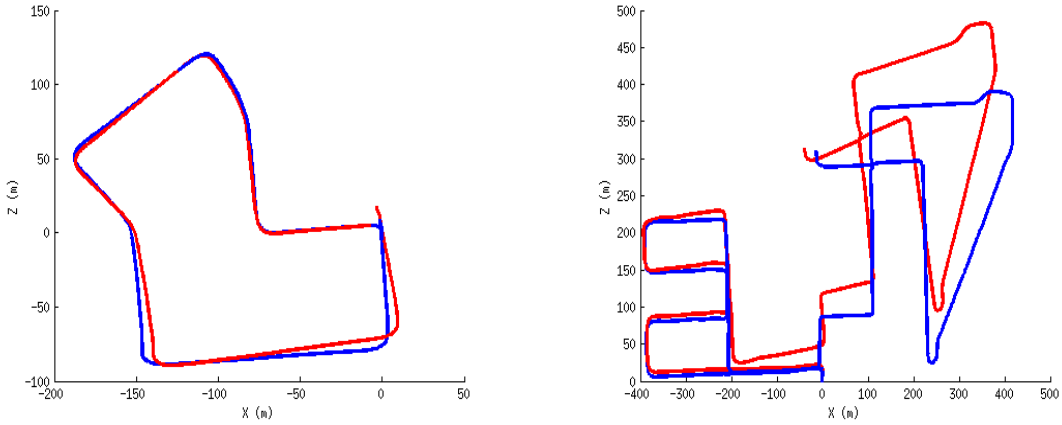


(e) Secuencia 04



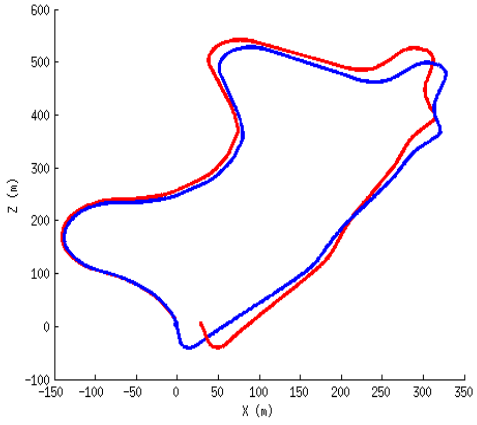
(f) Secuencia 05

Figura 5.4: Comparativa Ground Truth *vs* Odometría visual no corregida.

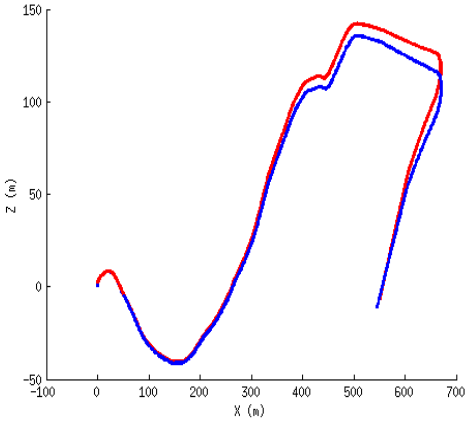


(a) Secuencia 07

(b) Secuencia 08



(c) Secuencia 09



(d) Secuencia 10

Figura 5.5: Comparativa Ground Truth *vs* Odometría visual no corregida en el resto de secuencias de KITTI.

En el dataset de KITTI para odometría, el vehículo recorre largas distancias, de varios kilómetros, por lo que la deriva acumulada es significativa y el resultado estimado no es totalmente preciso. Es en este hecho, precisamente, donde nace la necesidad de una corrección de la odometría.

Se comentan, a continuación, algunas de las características de las secuencias más representativas mostradas en las Figuras 5.4 y 5.5. Las secuencias 01, 03, 04, 07, 09 y 10 no presentan cierres de lazo, por lo que únicamente se utilizan para testear el algoritmo LIBVISO y visualizar la deriva acumulada en la estimación.

En cuanto a la deriva acumulada, comentar que el resultado inicial de la odometría visual realizada sobre las secuencias 01, 03, 07, 09 y 10 coincide prácticamente con el mapa estimado a partir de las medidas del GPS, pues, al tratarse de trayectorias cortas, apenas se acumula error en la estimación. Sin embargo, secuencias como la 00, 02 y 08 contienen varios cierres de lazo y representan trayectos complejos, de varios kilómetros y con giros bruscos, por lo que el algoritmo de odometría visual proporciona, en estos casos, un resultado con una deriva acumulada significativa.

A continuación, se muestra en la Figura 5.6 el resultado de la detección de cierre de lazo sobre la secuencia 06 de KITTI, tras realizar la integración de ABLE en el algoritmo LIBVISO. Se compara, además, con el mapa del *Ground Truth*, con el fin de poner de manifiesto la deriva acumulada. Se representa en color verde la trayectoria del vehículo estimada por la odometría visual. Como ya se ha comentado, la última parte del recorrido forma un cierre de lazo y, sin embargo, en el resultado de la odometría las líneas que representan el mismo camino visitado en dos instantes de tiempo diferentes no coinciden. Para mostrar gráficamente los resultados de la detección de cierre de lazo que realiza nuestro método, se dibuja en color amarillo las correspondencias entre pares de frames que representan el mismo lugar, pues la distancia de Hamming entre sus descriptores es muy baja. Como se indica en la Tabla 5.1, el primer frame de la secuencia (frame 0) representa el mismo lugar que el frame 835 (ambos unidos con una línea amarilla), donde comienza el cierre de lazo. El bucle finaliza en el frame 280, cuyo descriptor coincide con el obtenido de la imagen 1093.

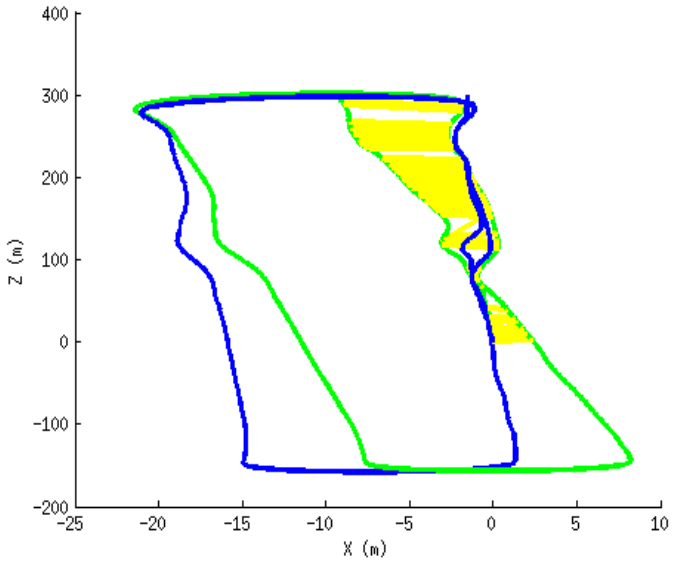


Figura 5.6: Detección de cierre de lazo sobre el resultado de la odometría visual en la secuencia 06 de KITTI *vs* mapa de GPS.

En la Figura 5.7 se representa la detección de cierre de lazo sobre otras secuencias de KITTI para odometría. Se comentan las principales características de las diferentes detecciones de cierre de lazo mostradas: se representa, nuevamente, el resultado de la odometría visual sin corregir en color verde y los cierres de lazo detectados en color amarillo, a través de líneas que unen aquellos frames cuyo descriptor describe el mismo lugar. Dependiendo de la complejidad de la secuencia procesada, la detección de los bucles se realiza de manera más o menos efectiva, pues no hay que olvidar que la odometría visual y cualquier método basado en ella proporciona resultados que no son completamente precisos. Por lo general, en todos los resultados mostrados en esta Figura, se realiza una buena detección de todos los cierres de lazo presentes.

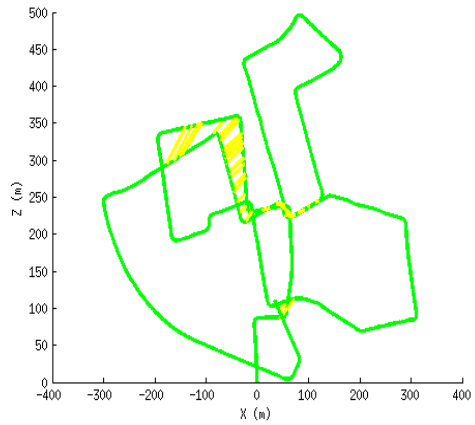
Volviendo al análisis de la secuencia 06, pues es la más representativa del problema que se está tratando, en la Figura 5.8(b) se representa una mejora introducida en nuestro método para aumentar la efectividad en la detección de los frames que forman parte de un cierre de lazo. En la Figura 5.6 se observa que un cierto número de frames que no son detectados como integrantes de un cierre de lazo (no están emparejados con ningún otro frame que represente el mismo lugar), en realidad sí lo son. Como solución a este problema, nuestro método considera que, dado un conjunto de imágenes entre las que hay frames que se han detectado como parte de un cierre de lazo, hay una elevada probabilidad de que aquellos otros frames cuya similitud no ha sido menor que el umbral fijado también formen parte de dicho lazo. Dicho de otra manera, una vez detectado un cierre de lazo, se recorren los frames correspondientes a esta zona con una ventana y, si en dicha ventana hay un número considerable de frames con una medida de similitud muy baja, el resto también son incluidos como elementos del bucle. Tras este proceso, la región del bucle queda detectada prácticamente en su totalidad.

Como pequeño apunte, en este caso la detección no es completamente exacta, ya que este método hace que un frame perteneciente a la última parte del trayecto y que ya no forma parte del cierre de lazo, también hayan sido detectados como integrantes del mismo. Esto sucede ya que los frames anteriores a él sí se correspondían con dicho lazo y, por tanto, nuestro método también lo incluye. Igualmente, este problema se soluciona haciendo uso del conocido método de eliminación de *outliers*, RANSAC, lo que permite concluir que nuestro método realiza una detección de cierre de lazo bastante precisa.

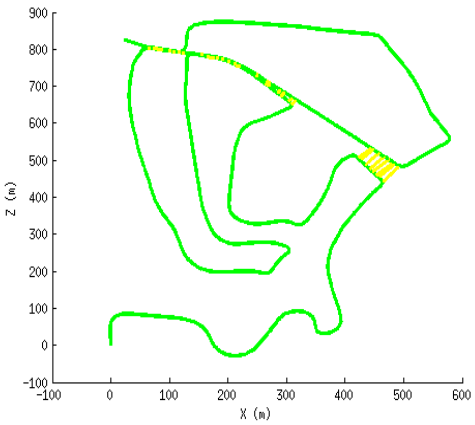
5.3. Resultados en la detección de cierre de lazo con ABLE

Para la evaluación de los resultados, el método ABLE almacena la matriz de similitud obtenida tras el procesamiento de una secuencia en formato imagen *(png)*. Esta matriz muestra gráficamente los frames donde se ha detectado un cierre de lazo. En la Figura 5.9 se muestra la matriz de similitud resultante de realizar una descripción global sobre la secuencia 06 del dataset de KITTI, mediante el descriptor binario LDB. En la imagen se representa el cierre de lazo con un color amarillo más intenso: los frames en color amarillo tienen una distancia de similitud muy baja, próxima a 0, mientras que los frames representados en color azul presentan una distancia muy elevada. Como ya se ha comentado, en la diagonal principal de esta matriz siempre aparece representado un cierre de lazo, ya que se trata de las correspondencias entre un frame y el frame obtenido un instante de tiempo posterior muy próximo, por lo que ambas imágenes representan prácticamente el mismo lugar debido al solapamiento perceptual y ABLE determina que se trata de un bucle (la distancia de Hamming entre los descriptores de dos frames sucesivos siempre va a ser muy próxima a 0).

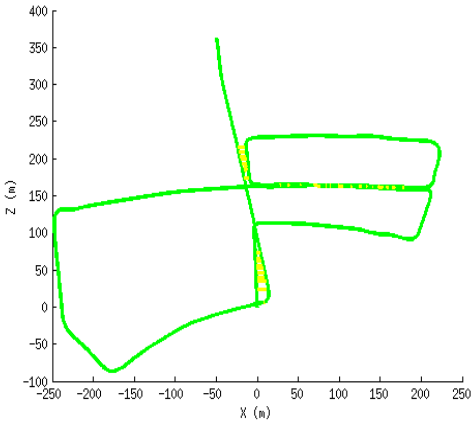
A continuación, se muestran resultados comparativos entre las diferentes descripciones realizadas. En la Figura 5.10 se representa el resultado de una detección de cierre de lazo mediante el descriptor binario AKAZE: en la Figura 5.10(a) se muestra la matriz del *Ground Truth* de la



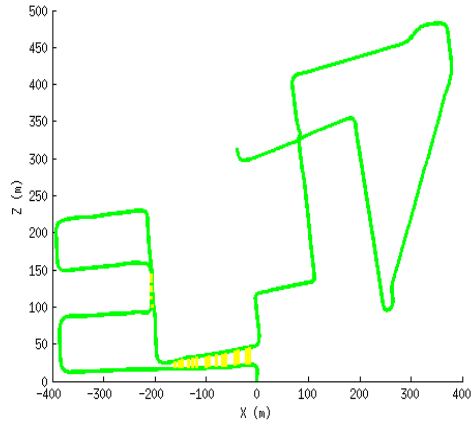
(a) Secuencia 00



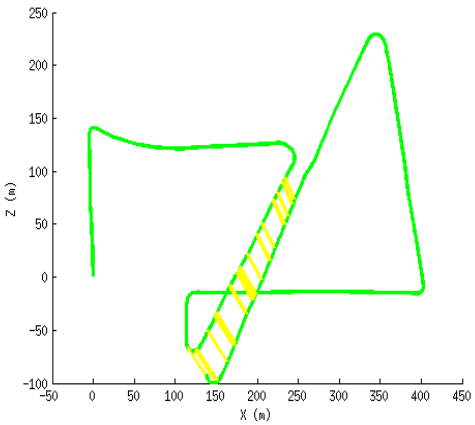
(b) Secuencia 02



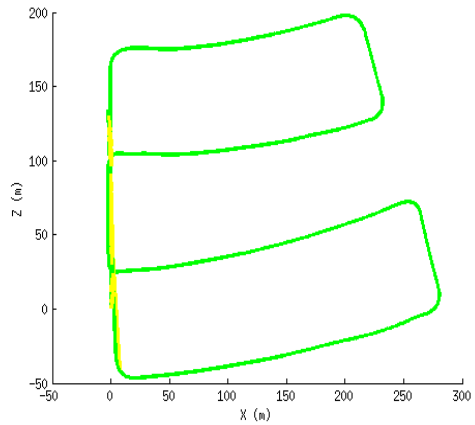
(c) Secuencia 05



(d) Secuencia 08



(e) Secuencia 13

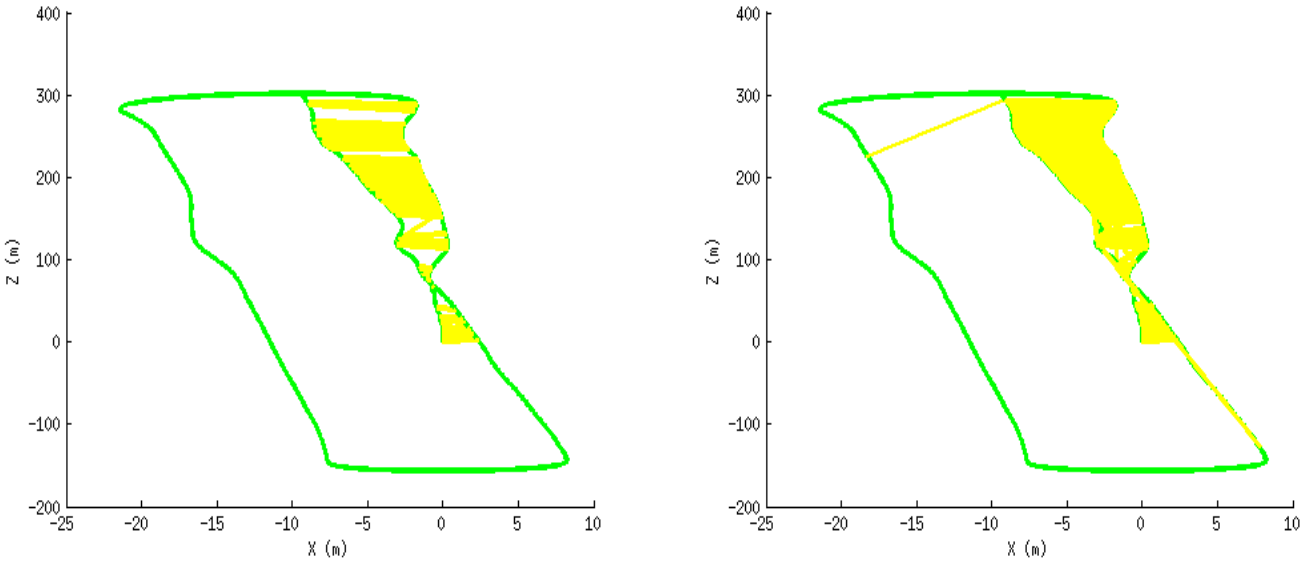


(f) Secuencia 16

Figura 5.7: Detección de cierres de lazo sobre el resultado de la odometría visual en diferentes secuencias de KITTI.

Tabla 5.1: *Ground-truth* creado para la detección de cierre de lazo en las secuencias del dataset de KITTI para odometría.

Sec.	No.Frame	Cierre de lazo unidireccional			Cierre de lazo bidireccional		
		No.	Frames iniciales	Frames Lazo	No.	Frames iniciales	Frames Lazo
00	4541	5	0000 - 0099	4451 - 4528	0		
			0122 - 0196	1570 - 1635			
			0392 - 0412	2446 - 2460			
			0392 - 0941	3398 - 3844			
			2354 - 2460	3295 - 3418			
02	4661	2	0933 - 1026	4205 - 4266	1	3332 - 3397	4566 - 4620
			1810 - 1997	4404 - 4569			
05	2761	3	0031 - 0121	2431 - 2512	0		
			0565 - 0787	1324 - 1530			
			0819 - 0885	2581 - 2627			
06	1101	1	0000 - 0280	0835 - 1093	0		
07	1101	1	0000 - 0013	1060 - 1067	0		
08	4071	0			2	0075 - 0227	1640 - 1796
						0726 - 0765	1422 - 1464
09	1591	1	0000 - 0023	1578 - 1590	0		
13	3281	4	0000 - 0138	2152 - 2316	0		
			0000 - 0089	3188 - 3280			
			0553 - 0839	1633 - 1938			
			2152 - 2264	3188 - 3280			
15	1901	1	0000 - 0086	1808 - 1900	0		
16	1731	1	0000 - 0146	1614 - 1730	1	0023 - 0079	0798 - 0843
18	1801	1	0322 - 0493	1616 - 1800	0		
19	4981	1	4246 - 4390	4812 - 4943	0		



(a) Secuencia 06 de KITTI

(b) Secuencia 06 de KITTI

Figura 5.8: Resultado comparativo al hacer uso de una ventana para incluir frames vecinos en la detección de cierre de lazo.

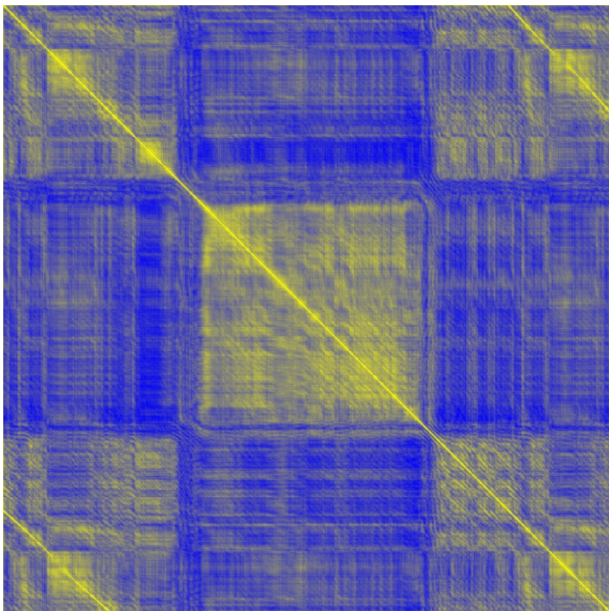
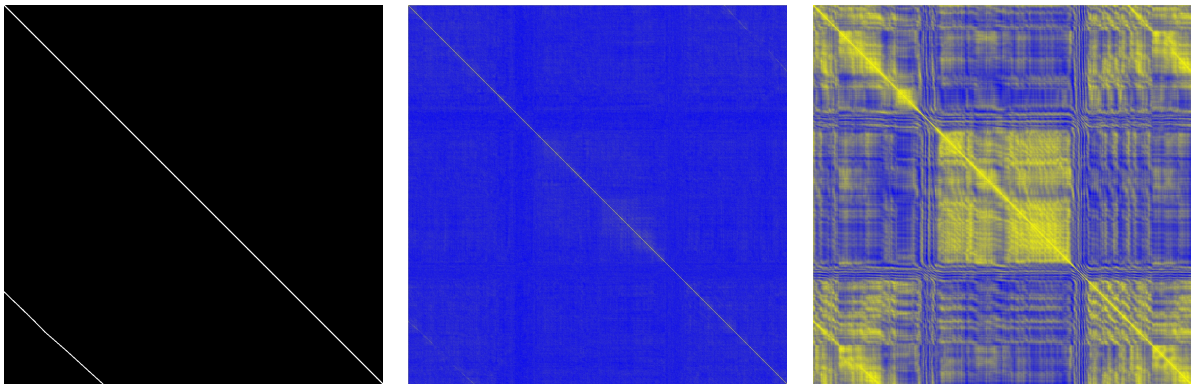


Figura 5.9: Matriz de similitud de la secuencia 06 del dataset de KITTI.

secuencia 06 de KITTI, donde los frames que forman el bucle quedan totalmente determinados en color blanco. La Figura 5.10(b) muestra el resultado de realizar la descripción de las imágenes haciendo uso de un grid de tamaño 3x1, mientras que la Figura 5.10(c) es el resultado de una descripción global de las imágenes de la secuencia.



(a) Ground Truth (b) Descripción con grid (c) Descripción global

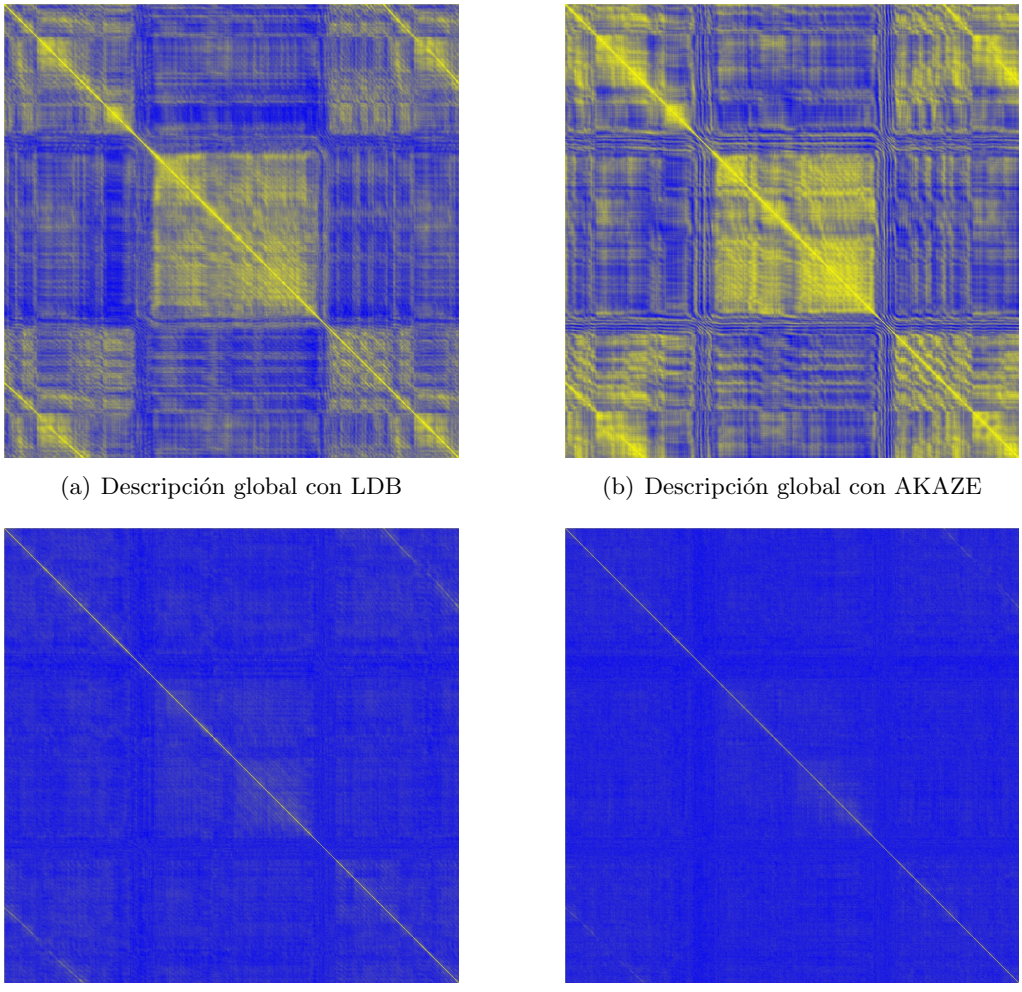
Figura 5.10: Comparativa entre descripción global y mediante grid en la secuencia 06 de KITTI.

Como se observa en la Figura 5.10, de forma general, haciendo uso del grid para realizar la descripción en vez de tomar directamente los píxeles centrales de cada imagen, se logra reducir sensiblemente el ruido, aunque se pierde efectividad a la hora de detectar correctamente el cierre de lazo. En el ejemplo concreto de la secuencia 06 de KITTI, en la matriz resultante de realizar una descripción jerárquica (con un grid de tamaño 3x1) se observa que la línea que representa la detección de cierre de lazo aparece difuminada en el centro. Como ya se ha explicado anteriormente, esto es debido al adelantamiento a una bicicleta que hace el vehículo la segunda vez que recorre la misma zona, por lo que las imágenes capturadas en esos momentos no son exactamente las mismas. Sin embargo, a pesar de lo comentado, si se realiza una descripción global de las imágenes, todos los frames que integran el cierre de lazo son detectados, a pesar

de que la representación resultante contenga mayor cantidad de ruido.

A continuación, se muestra en la Figura 5.11 una comparativa entre los resultados obtenidos al realizar una descripción mediante los descriptores binarios AKAZE y LDB. Analizando visualmente las imágenes de similitud, se observa que los resultados obtenidos con ambos descriptores son comparables (más adelante se analizarán estos mismos resultados de manera cuantitativa a través de las curvas de rendimiento).

Los resultados en la detección de cierre de lazo en la secuencia 06 de KITTI al realizar una descripción global mediante LDB son ligeramente superiores a los obtenidos mediante AKAZE. En este caso, esto es debido a que la secuencia analizada no permite explotar completamente las mejoras introducidas por AKAZE, ya que no presenta grandes cambios de escala o rotación. Por otro lado, se obtiene un resultado más efectivo con AKAZE al realizar una descripción mediante un grid, pues los píxeles de ruido que aparecen alrededor de las líneas del cierre de lazo debidos al solapamiento perceptual son reducidos casi en su totalidad.



(a) Descripción global con LDB

(b) Descripción global con AKAZE

(c) Descripción con grid con LDB

(d) Descripción con grid con AKAZE

Figura 5.11: Comparativa LDB *vs* AKAZE.

El método ABLE, permite elegir al usuario qué descriptor emplear a la hora de ejecutar el algoritmo, a través del fichero de configuración, *config.txt*. Se hace uso de esta propiedad para obtener un mayor número de resultados gráficos. En este caso, se representa en las Figura 5.12 y 5.13 una evaluación completa de los resultados obtenidos al hacer una descripción global y

jerárquica con los diferentes descriptores existentes en el estado del arte.

Al igual que se comentaba en la Figura 5.11, se observa cómo el ruido presente en las imágenes de similitud es menor al realizar la descripción a través de un grid, aunque el tiempo de cómputo y la cantidad de memoria empleados son mayores. Además, al realizar esta descripción, hay ciertas zonas del cierre de lazo que no se consiguen detectar con precisión, por lo que es menos efectiva que la global.

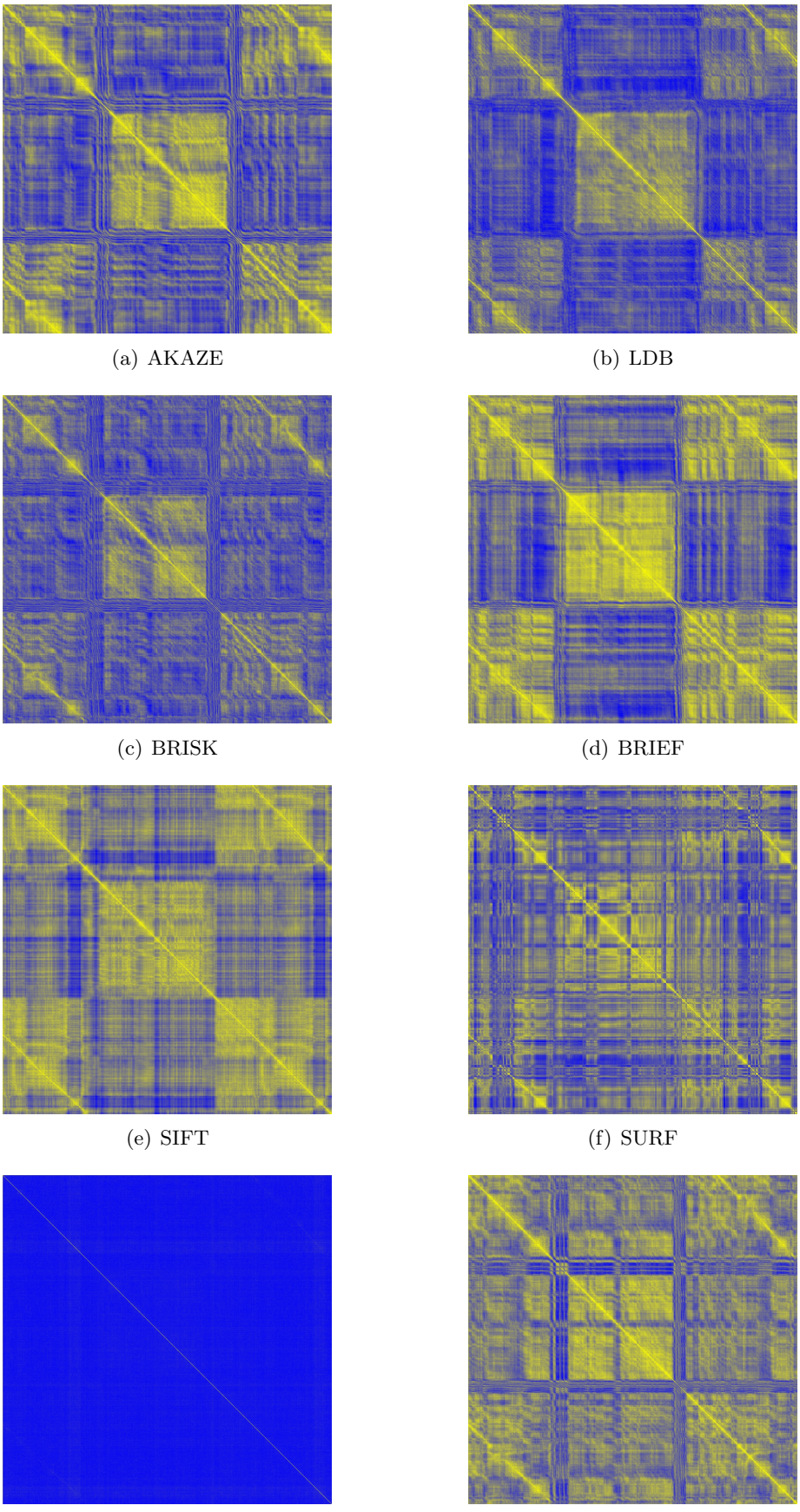
Se analiza, a continuación, la efectividad de los descriptores empleados: en el caso de la descripción realizada con grid (Figura 5.13), la detección del cierre de lazo se consigue, con mayor o menor precisión, haciendo uso de los descriptores AKAZE, LDB, SIFT, ORB y BRIEF. Por el contrario, HOG, SURF y BRISK no son capaces de detectar los frames que forman parte del bucle, pues las matrices de similitud obtenidas en estos casos no reflejan ningún frame en color amarillo.

Analizando cualitativamente la Figura 5.12, en la que se representa el resultado de realizar una descripción global, se observa que el descriptor que proporciona una matriz de similitud más pobre en cuanto a información contenida en ella es HOG, pues de nuevo no representa ningún frame en color amarillo. En cuanto al resto de descriptores, todos devuelven una imagen de similitud que representa en amarillo los frames que determinan el bucle. Observando los resultados del método al hacer uso de los diferentes descriptores se puede afirmar que la detección realizada por AKAZE y LDB es la más efectiva.

La última afirmación realizada en el párrafo anterior es corroborada a través de las curvas de rendimiento o *precision-recall* mostradas en las Figuras 5.14, 5.15 y 5.16. Para entender los resultados obtenidos en estas imágenes, se recuerda que ORB y BRISK mejoran las características de BRIEF al añadir la invarianza a la rotación y escala y que LDB no sólo usa la información de intensidad como hace BRIEF, sino que, además, calcula el gradiente de las imágenes y divide éstas en varias celdas para realizar una descripción mediante un grid, lo que permite obtener mejores resultados, como se puede observar. Además, AKAZE muestrea la imagen a describir, consiguiendo así mayor velocidad en la construcción del espacio de escala no lineal y en el proceso de detección. Este muestreo de la imagen no empeora el rendimiento del detector ya que el proceso de difusión no lineal se realiza en un radio (o vecindad) muy pequeño alrededor de cada píxel. Además, otra de las razones por las que se obtienen mejores resultados con este descriptor es que el método que emplea para resolver las ecuaciones de difusión no lineal, FED, es muy rápido y preciso. En cuanto a los detectores rápidos como ORB y HOG, se observa su bajo rendimiento, especialmente en el caso de HOG. Esto puede ser debido a que ambos se basan en la detección que realiza FAST, cuyos resultados son poco fiables en el problema de la detección de cierre de lazo. En este caso es preferible el uso de descriptores basados en el cálculo de la matriz Hessiana o de las diferencias de Gaussianas, como AKAZE o LDB, como corroboran estas curvas de rendimiento.

Centrándonos en los resultados de la implementación realizada en este proyecto, AKAZE, en su función de detección de cierre de lazo presenta un alto rendimiento al procesar aquellas secuencias más largas y complejas, que presenten un número considerable de lazos. Por ello, las curvas *precision-recall* de la secuencia 06 (Figura 5.16) muestran un bajo rendimiento de AKAZE, puesto que se trata de un recorrido sencillo, sin demasiados cambios de dirección y con un sólo cierre de lazo. Por otro lado, al evaluar las secuencias 00 y 05 (Figuras 5.15 5.14) del dataset de KITTI, más desafiantes pues presentan trayectorias complejas, con varios cierres de lazo, se observa el alto rendimiento de este descriptor.

El dataset de KITTI contiene secuencias grabadas con un sistema estéreo fijado a la parte superior del vehículo, por lo que apenas hay cambios de rotación o escala entre las imágenes.



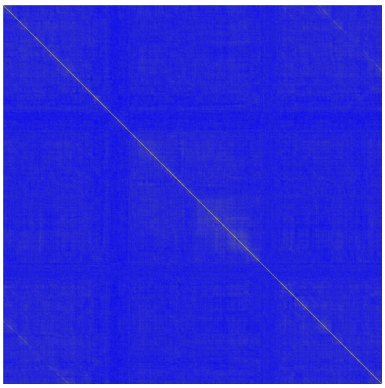
(a) AKAZE (b) LDB

(c) BRISK (d) BRIEF

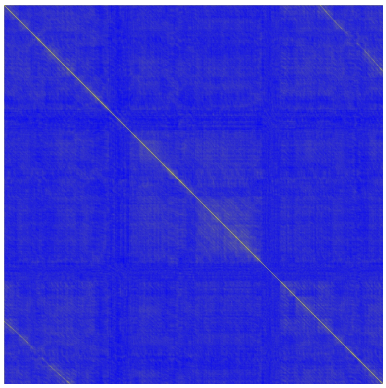
(e) SIFT (f) SURF

(g) HOG (h) ORB

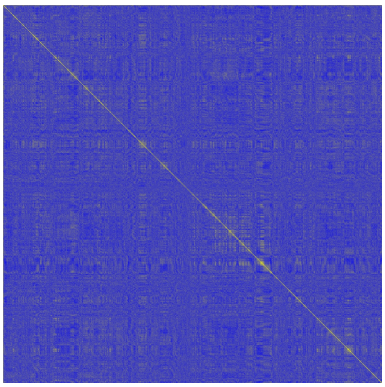
Figura 5.12: Descripción global de la secuencia 06 de KITTI empleando distintos descriptores.



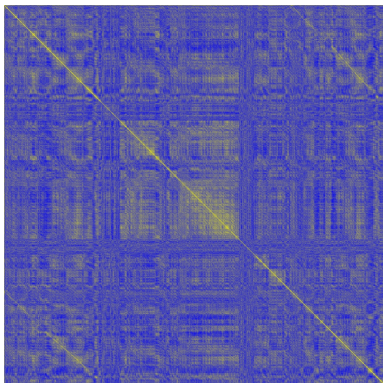
(a) AKAZE



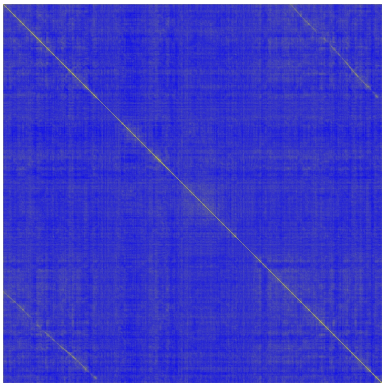
(b) LDB



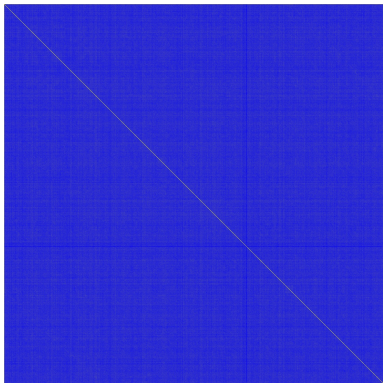
(c) BRISK



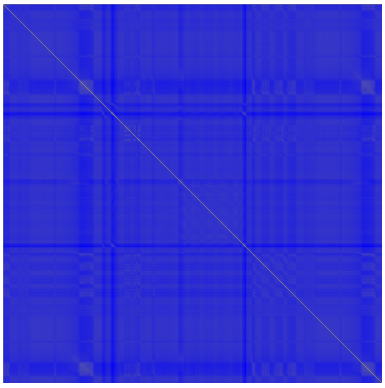
(d) BRIEF



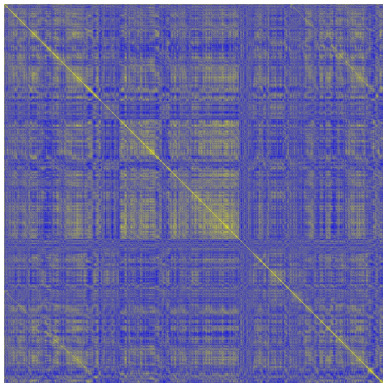
(e) SIFT



(f) SURF



(g) HOG



(h) ORB

Figura 5.13: Descripción basada en grid de la secuencia 06 de KITTI empleando distintos descriptores.

Por esta razón, el rendimiento que presenta AKAZE es similar o ligeramente superior a otros descriptores, incluso en la secuencia 06 es inferior, ya que estas secuencias no permiten explotar las mejoras introducidas por el nuevo descriptor.

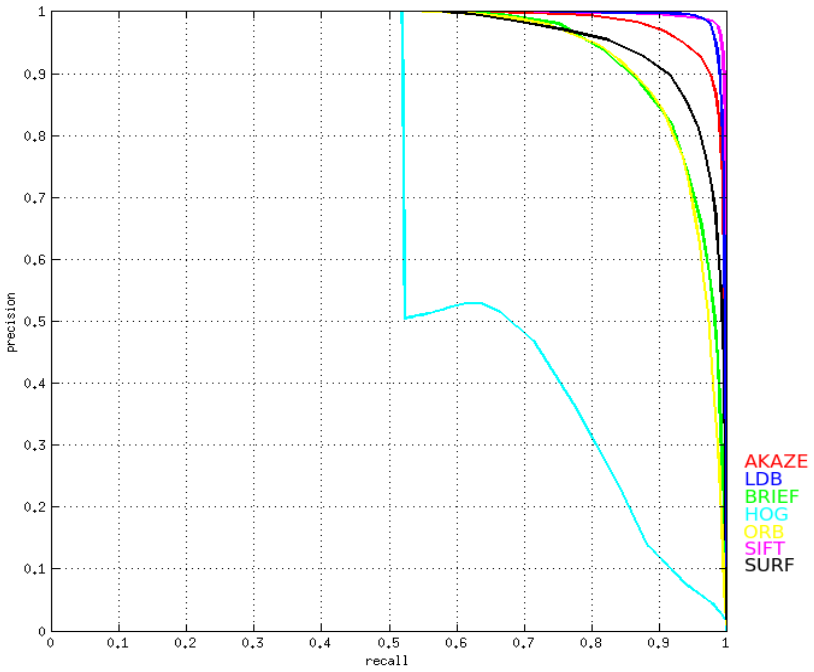


Figura 5.14: Curvas de rendimiento de la secuencia 00 de KITTI.

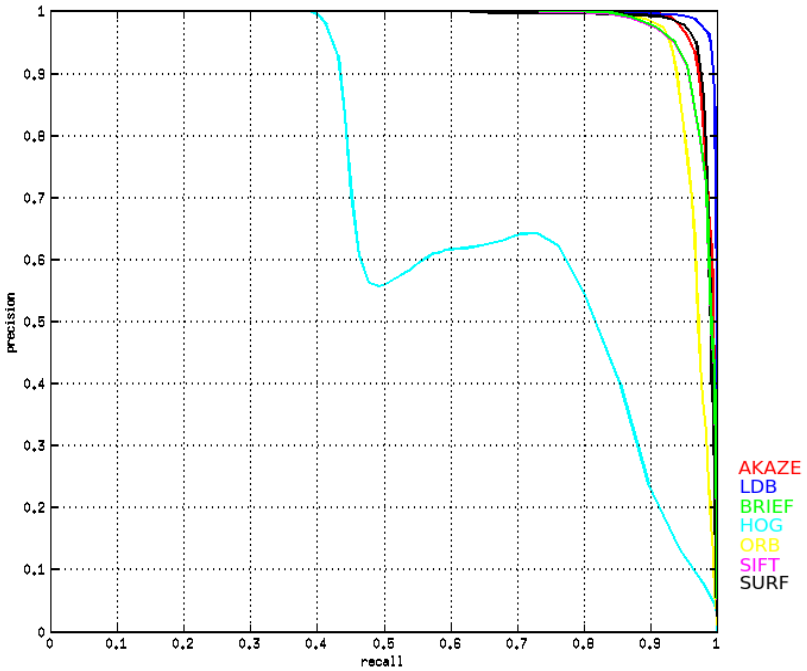


Figura 5.15: Curvas de rendimiento de la secuencia 05 de KITTI.

Para finalizar la evaluación de los resultados que proporciona ABLE, se muestra en la Tabla 5.2 el tiempo medio y total de procesamiento para cada descripción y *matching* realizados con los diferentes descriptores en la secuencia 06 del dataset de KITTI. Estos resultados han sido

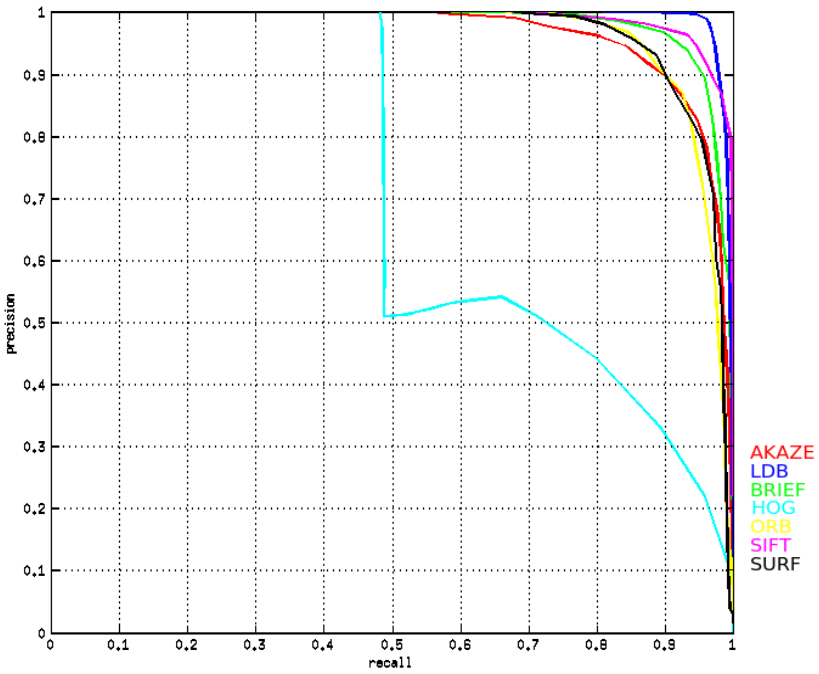


Figura 5.16: Curvas de rendimiento de la secuencia 06 de KITTI.

obtenidos a partir de un ordenador Toshiba Intel Core i5 con una CPU de 2,30 GHz y 4GB de memoria RAM.

Método ABLE	**Tiempos de cómputo**			
	Tiempo total en describir todas las imágenes	Tiempo total en emparejar todas las imágenes	Tiempo medio en describir una imagen	Tiempo medio en describir dos imágenes
BRIEF	146,72 ms	255,05 ms	1,33 $\cdot 10^{-}1$ ms	4,21 $\cdot 10^{-}4$ ms
BRISK	571,40 s	285,05 ms	519,45 ms	4,71 $\cdot 10^{-}4$ ms
ORB	401,28 ms	265,17 ms	3,64$\cdot 10^{-}1$ ms	4,38 $\cdot 10^{-}4$ ms
FREAK	69,21 s	276,05 ms	62,92 ms	4,56 $\cdot 10^{-}4$ ms
LDB	202,41 ms	359,99 ms	1,84 $\cdot 10^{-}1$ ms	5,95 $\cdot 10^{-}4$ ms
HOG	449,42 ms	3,79 s	4,08 $\cdot 10^{-}1$ ms	6,28 $\cdot 10^{-}3$ ms
SIFT	859,98 ms	1,75 s	7,78 $\cdot 10^{-}1$ ms	2,89 $\cdot 10^{-}3$ ms
SURF	393,76 ms	1,75 s	3,57 $\cdot 10^{-}1$ ms	2,89 $\cdot 10^{-}3$ ms
AKAZE	310,72 ms	275,81 ms	8,71 $\cdot 10^{-}1$ ms	4,60 $\cdot 10^{-}4$ ms

Tabla 5.2: Comparativa tiempos de cómputo de los distintos descriptores.

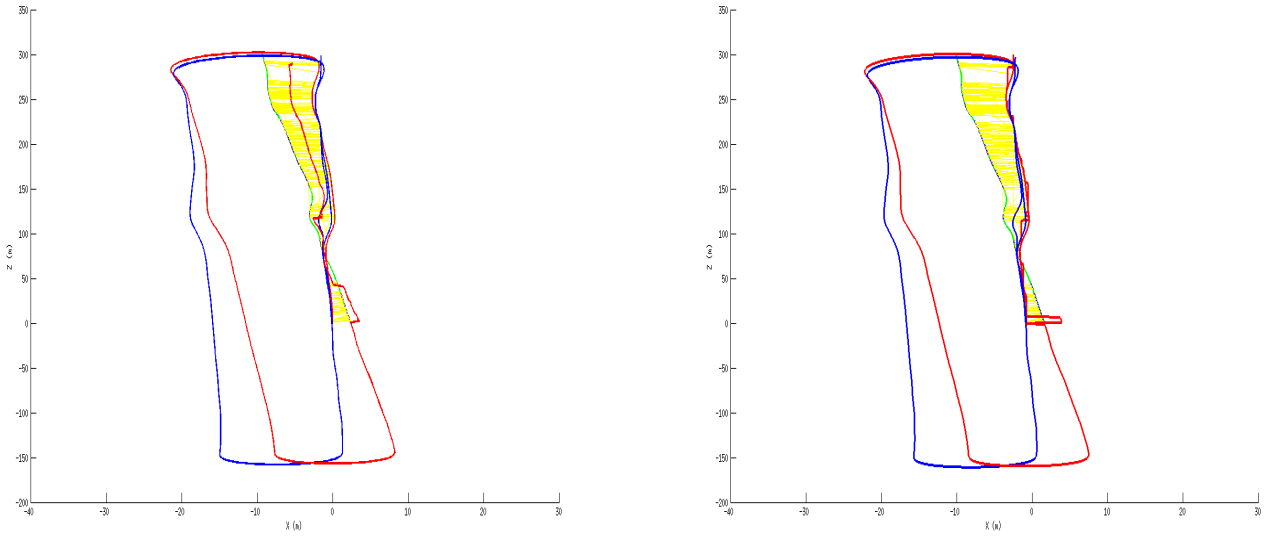
5.4. Resultados en la corrección de la odometría visual

En esta sección se muestran los resultados de la corrección de la odometría visual. Se muestra, con distintas Figuras, el proceso llevado a cabo para realizar una corrección de la trayectoria lo más efectiva posible, hasta llegar al resultado final.

En un primer intento de actualizar la trayectoria, nos centramos únicamente en la zona donde se encuentra el cierre de lazo. Las nuevas poses se corrigen en base a la desviación media

producida en dicha zona, tanto en el eje x, como en el eje z. De esta forma, como se observa en la Figura 5.17(a) sólo se consigue corregir la mitad del error (de nuevo mostramos los resultados sobre la secuencia 06 de KITTI). Dicha corrección es representada en color rojo, la primera estimación realizada por la odometría en color verde y el *Ground Truth* en azul.

Una vez que se ha corregido el error medio entre las nuevas poses y las antiguas, se procede a la corrección total del error, cuyo resultado se muestra en la Figura 5.17(b), donde en la zona del cierre de lazo, la representación de la corrección de la odometría, en color rojo, coincide con la representación del (*Ground Truth*), en color azul, consiguiendo así una buena optimización del resultado inicial de la odometría visual en la zona donde se da el bucle.



(a) Corrección del error medio en la zona del lazo

(b) Corrección del error total en la zona del lazo

Figura 5.17: Pasos previos a la corrección de la odometría visual en la secuencia 06 de KITTI.

A continuación, se muestra el resultado final de la corrección de la trayectoria completa o movimiento realizado por el vehículo a lo largo de toda la secuencia. En la anterior Figura 5.17(b), obviando la corrección realizada en la zona del cierre de lazo, la representación del resultado inicial de la odometría se solapa con la representación de la corrección, en color rojo, pues el método se centra, en primer lugar, en optimizar únicamente la zona del bucle, como ya se ha comentado. Sin embargo, en la Figura 5.18 se presenta una corrección íntegra de la odometría. Una vez calculada la desviación media producida en el lazo, se utiliza este dato para realizar la optimización de los frames iniciales, de manera que todas las poses anteriores al cierre de lazo son reubicadas en base a esa desviación, aplicándola de forma proporcional, pues la deriva acumulada en los últimos frames es mucho mayor que la acumulada al comienzo de la secuencia. Nuevamente, se representa el color rojo la corrección de la odometría visual, en color verde la estimación inicial y en color azul las medidas del GPS. Se observa una optimización efectiva de la trayectoria, mucho más precisa que el resultado inicial de la odometría. Sin embargo, aunque nuestro método consigue perfeccionar la estimación del movimiento del agente, los resultados obtenidos no son totalmente precisos (la gráfica de la corrección, en color rojo, no coincide completamente con el mapa del GPS, en azul), pues, como ya se ha ido explicando a lo largo de toda la memoria, la corrección se realiza tomando como base, únicamente, las imágenes adquiridas por el sistema de cámaras, fuente de información que no es totalmente fiable y que está sujeta a cambios. A pesar de ello, en la Figura 5.19 se muestra una comparativa entre el

resultado inicial de la odometría visual, en la Figura 5.19(a) y el resultado final de la corrección realizada, en la Figura 5.19(b), donde se pone de manifiesto la efectividad de nuestro método.

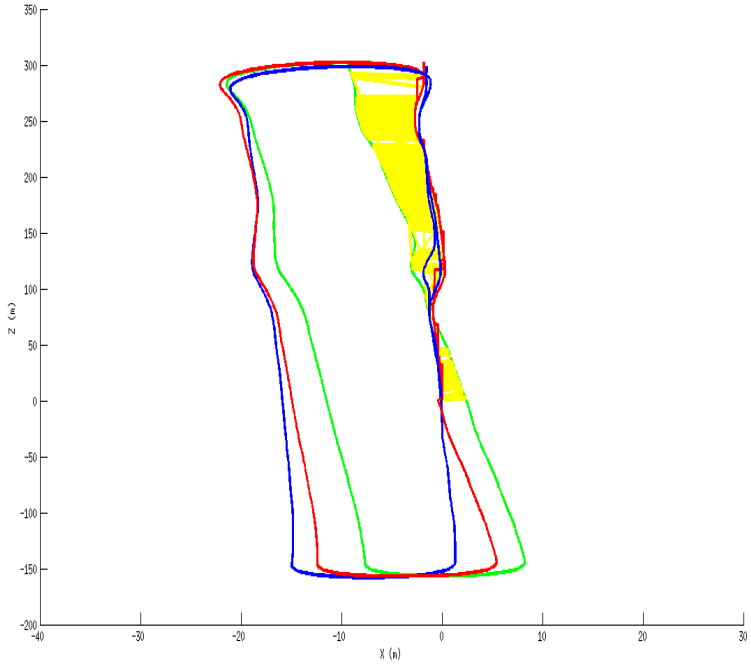
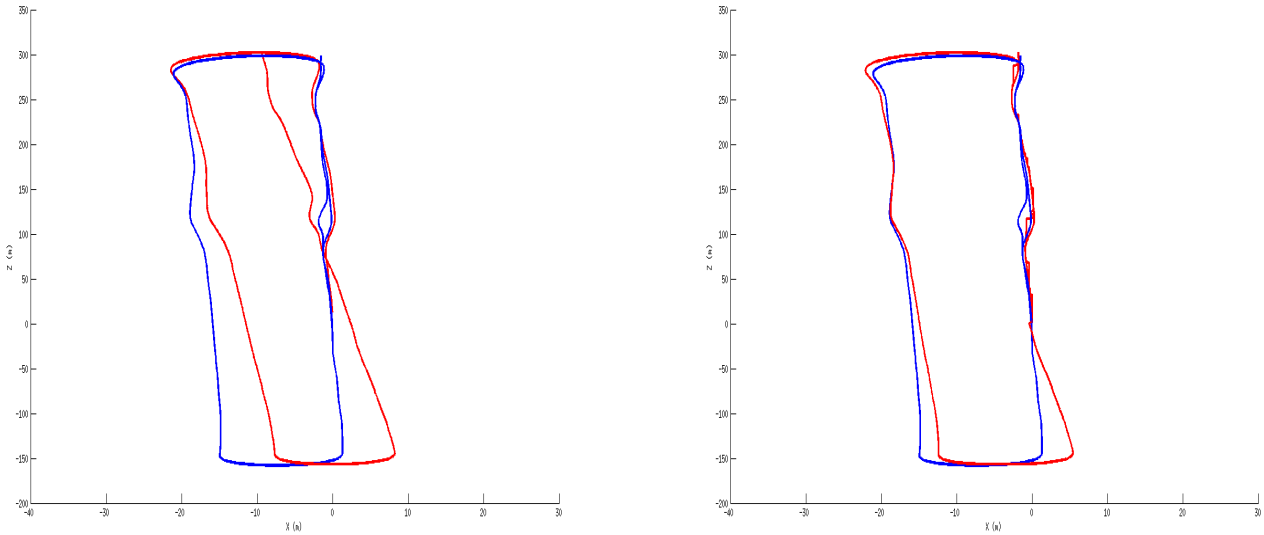


Figura 5.18: Corrección de la odometría visual en la secuencia 06 de KITTI.



(a) Odometría visual de la Secuencia 06 de KITTI vs GPS

(b) Corrección de la odometría de la Secuencia 06 de KITTI vs GPS

Figura 5.19: Resultado comparativo entre la odometría visual y la corrección de la misma a partir de la detección de cierre de lazo.

Por último, en la Figura 5.20 se muestra el resultado final del método implementado sobre

otra secuencia del dataset de KITTI, concretamente sobre la secuencia 05. En las zonas donde se encuentran los cierres de lazo (para saber qué frames forman el cierre de lazo se puede consultar la tabla 5.1), la corrección es prácticamente perfecta, pues el resultado de la optimización de la odometría visual (en color rojo) coincide con el mapa obtenido de las medidas tomadas con el GPS (en color azul). La deriva acumulada en el resto de la trayectoria (en color verde) es reducida significativamente, una vez calculada la desviación media producida en la zona del bucle. Esta desviación es ponderada y posteriormente sumada al valor de las poses, consiguiendo así una buena corrección de la estimación inicial del movimiento que realiza la odometría visual.

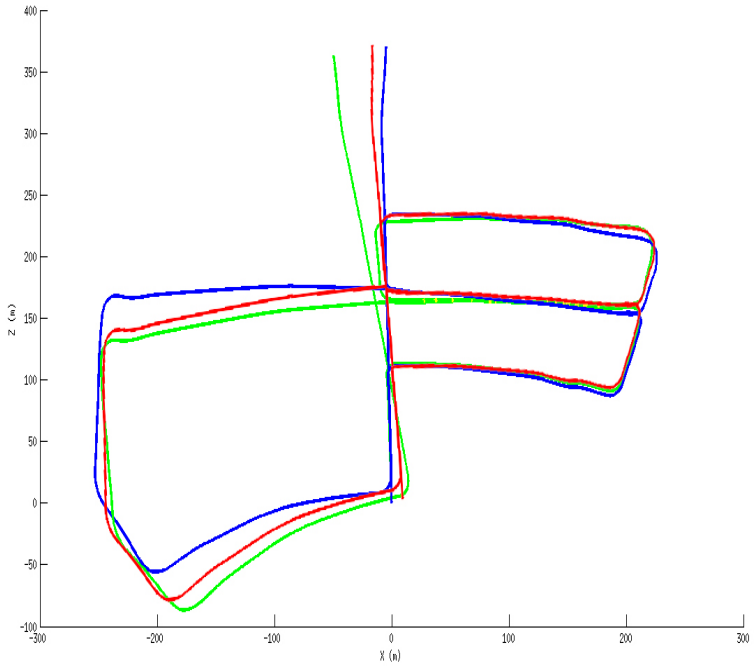


Figura 5.20: Corrección de la odometría visual en la secuencia 05 de KITTI.

Capítulo 6

Conclusiones y Trabajos Futuros

Como primera aportación, en este proyecto se ha extendido el método ABLE, y se ha demostrado cómo este enfoque realiza una detección de cierres de lazo y reconocimiento visual de lugares de forma efectiva. Para ello, este algoritmo se basa en el uso de descriptores binarios, concretamente en LDB o D-LDB (añadiendo la información de disparidad). Como contribución, se ha implementado sobre este método, la detección y descripción de características mediante AKAZE (basado en M-LDB), obteniendo mejores resultados y una precisión superior a otros algoritmos existentes en el estado del arte, como FAB-MAP, WI-SURF o BRIEF-Gist. Además, el coste computacional y consumo de recursos de memoria de ABLE es significativamente menor debido al empleo de descriptores binarios, lo que permite un rápido reconocimiento visual.

Como ya se ha comentado, AKAZE realiza el proceso de descripción de las imágenes en un espacio de escala no lineal, y añade a las características de LDB la invarianza a cambios en la rotación y en la escala. Con esto se consigue que el método de detección de cierres de lazo sea robusto en contextos en los cuales la cámara sufre cambios debidos a giros o percepción de distancias. Debido a estos beneficios, el método gana en robustez en situaciones de localización a largo plazo.

Para evaluar el rendimiento de nuestro método de detección de cierres de lazo se ha hecho uso del dataset de KITTI y del *Ground Truth* definido en [6], que ha permitido realizar una evaluación objetiva de los resultados. Como se observa en las curvas *precision-recall*, AKAZE presenta mejor rendimiento que el resto de descriptores (BRIEF, HOG, ORB, SIFT, SURF), a excepción de LDB, que en la evaluación de las secuencias de KITTI proporciona resultados comparables a los obtenidos con AKAZE pero ligeramente mejores en algunos de los casos. Esto es debido a que el dataset empleado no permite explotar completamente las mejoras introducidas por AKAZE, ya que todas las secuencias de vídeo están grabadas sin grandes cambios de escala o rotación.

Por otro lado, el algoritmo de odometría visual empleado, LIBVISO, realiza una buena estimación de la trayectoria seguida por un vehículo en entornos urbanos, gracias a la fase de actualización del filtro de Kalman que se emplea, que permite eliminar los efectos de no linealidad durante el proceso de la odometría. La principal novedad introducida por este método es el empleo del tensor trifocal entre tríos de imágenes, en combinación con el método de rechazo de *outliers*, RANSAC, que permite estimar el movimiento del móvil sin necesidad de realizar una reconstrucción 3D de la escena. Además, se calculan de forma precisa las velocidades lineal y angular del vehículo, gracias al uso de la técnica de *bucketing*, que garantiza que la mayoría de puntos característicos caigan sobre el fondo estático de la escena y no sobre los objetos en movimiento.

En nuestro propósito de corregir y mejorar el resultado inicial de la odometría visual, se ha integrado el método ABLE con el algoritmo LIBVISO, con el fin de determinar los cierres de lazo sobre los mapas resultantes de la odometría visual y realizar una corrección en base a estos cierres de lazo. Esta corrección llevada a cabo permite reducir de forma significativa el error acumulado en la estimación de la odometría, de forma que en trayectorias largas, de cientos o miles de kilómetros, esta deriva incremental puede ser compensada, y la estimación del movimiento relativo del robot es actualizada de manera más correcta. De esta forma, cada vez que el agente se encuentre en una situación de cierre de lazo, sus actuales poses son corregidas de manera inmediata, y la trayectoria seguida hasta el momento es corregida hacia atrás, de forma proporcional a la deriva acumulada. Todo esto se ha corroborado en los resultados de odometría visual presentados a lo largo de esta memoria para algunas de las secuencias del dataset de KITTI, donde se puede ver cómo las correcciones realizadas en las poses a partir de la información de cierre de lazo hacen que la trayectoria del vehículo proyectada en el mapa se acerque mucho más a la posición real que si se usase solo el método de odometría visual sin ningún tipo de corrección.

Con el fin de mejorar el método propuesto, se proponen futuras líneas de investigación derivadas de este proyecto. Por una parte, se pretende implementar este enfoque de manera totalmente online, con el fin de ir actualizando la trayectoria del robot en tiempo real, a medida que va trazando su trayectoria.

Además, otro de los objetivos futuros de nuestra investigación consiste en llevar a cabo un método todavía más robusto de localización visual a largo plazo, tema de suma importancia en el ámbito de la navegación visual actualmente. De hecho, durante el pasado año se han desarrollado varios trabajos que consideran este problema del reconocimiento visual de lugares a diferentes horas del día [50], o durante las cuatro estaciones del año [45].

Por último, se pretende implementar también en el futuro una corrección de la odometría visual en base a dos algoritmos de actualización de la trayectoria existentes en el estado del arte y comentados en esta memoria: GraphSlam y el método de optimización de Levenberg-Marquardt. Estos métodos se ajustan a nuestro problema, ya que realizan una corrección de la trayectoria que puede basarse en la detección de cierres de lazo y en restricciones dadas por la odometría visual.

Manual de Usuario

Manual

En esta sección se muestran los requisitos necesarios para hacer uso de las librerías desarrolladas en este proyecto y para la ejecución de los métodos implementados.

Para la construcción del proyecto software, o ejecutable se ha hecho uso de la herramienta CMake. Su instalación es muy sencilla y basta con ejecutar en la terminar de la consola la siguiente instrucción: *sudo apt-get install cmake*.

En el mismo directorio en el que se encuentra el fichero principal de nuestro proyecto, se crea un archivo llamado *CMakeLists.txt*. Este fichero contiene el programa que describe qué dependencias deben cumplirse y cómo se construye el proyecto. Además indica toda la información necesaria: ficheros a compilar, las librerías y ejecutables a generar, las librerías de las que depende, o los programas externos a utilizar. En él se debe especificar la versión mínima de CMake a utilizar, mediante el comando *CMAKE_MINIMUM_REQUIRED* y dar un nombre al proyecto utilizando el comando *PROJECT*. El comando *ADD_EXECUTABLE* acepta una lista de dependencias. Esto permite no sólo crear Makefiles, sino también exportar a proyectos de Eclipse u otros entornos de desarrollo. Por último, *ADD_SUBDIRECTORY* añade un nuevo subdirectorio a la lista de subdirectorios del proyecto independientemente del contenido de este.

En este proyecto se ha hecho uso de las librerías de código abierto OpenCV, tanto en el método de detección de cierre de lazo, como en la odometría visual y su corrección. Estas librerías de visión artificial fueron descargadas a través de la página web `http://www.opencv.org/`, de donde también se pueden obtener manuales para su instalación en los diferentes sistemas operativos (Linux, Windows o Mac).

Estas librerías se pueden ejecutar de dos formas diferentes: a través de Matlab, haciendo uso de los ficheros *.mex*, o bien en un entorno de C/C++, opción elegida en este proyecto. Se debe configurar el fichero *CMakeLists.txt* para enlazar de forma correcta las librerías OpenCV y poder hacer uso de ellas en los algoritmos ABLE y LIBVISO.

De forma similar, para la implementación del descriptor AKAZE en el método ABLE, se hizo uso de las librerías públicas de AKAZE, desarrolladas por Pablo F. Alcantarilla y descargadas de su página web `http://github.com/pablofdezalc/akaze/`. De nuevo, fue necesario modificar el fichero *CMakeLists.txt* para utilizar las funciones de forma adecuada. Para indicar las librerías a enlazar se hace uso del comando *TARGET_LINK_LIBRARIES*.

Una vez configurados los requisitos hardware y software necesarios, se procede a la generación del proyecto. Para ello, se ingresa en la terminal y se accede al directorio donde se encuentra el proyecto. Una vez allí, se ejecuta el comando *cmake ...* Una vez hecho esto, se generan una serie de directorios y archivos que permiten que la compilación e instalación del proyecto se realice al ejecutar los siguientes comandos:

- *make*
- *sudo make install*

Después de la generación del proyecto, basta con ejecutarlo con la siguiente instrucción:

- *.\viso2 config.txt kitti\número_secuencia*

Donde *viso2* es el propio archivo ejecutable, *config.txt* es el fichero de configuración del método ABLE y *número_secuencia* es la secuencia del dataset de KITTI con la que se quiere

testear el algoritmo de corrección de la odometría, que en este caso se encuentra en la carpeta *kitti*.

En caso de que se quiera lanzar únicamente el algoritmo de ABLE para obtener las matrices de similitud, se realizan los mismos pasos indicados anteriormente y se ejecutan en la consola las instrucciones *cmake..* y *make*, dentro del directorio donde se encuentra el fichero principal. Entonces, una vez generado el proyecto, éste se ejecuta a través de la instrucción:

- *.\test_able config.txt*

Por último, para representar gráficamente los resultados mostrados en el capítulo 5, se ha utilizado el programa de cálculo Matlab. Para la representación del *Ground Truth* o medidas del GPS se ha hecho uso del fichero *display_kitti_poses.m*, para la detección de cierres de lazo en los mapas que proporciona la odometría visual se ha empleado el código de *display_kitti_poses_loop_detection.m* y para la corrección, el archivo *display_kitti_poses_correction.m*. En los tres casos, se necesita como argumento de entrada el fichero *.txt* que contiene todas las poses del robot (bien obtenidas vía satélite para la representación del *Ground Truth* o bien a través de la odometría visual) y las distancias Hamming calculadas para cada frame.

Para obtener las curvas de rendimiento se emplean los ficheros *Precision_Recall_graphic.m* y *Precision_Recall_iter.m*. Se llama desde el espacio de trabajo a la función *Precision_Recall_graphic ('similarity.txt','matrix.png',vecindad,iteraciones);*, donde *similarity.txt* es el fichero que contiene la matriz de similitud, *matrix.png* representa el *Ground Truth* de la secuencia en formato *png* y la vecindad es el número de píxeles alrededor del píxel a evaluar (conviene usar un valor de vecindad comprendido entre 5 y 9 para obtener buenos resultados).

Pliego de condiciones

Requisitos de Hardware

- Ordenador Toshiba Intel Core i5-2410M con las siguientes características:
  - CPU de 2.30 GHz
  - Memoria RAM de 4GB DDR3 1333 MHz
  - Disco Duro de 640 GB
  - Tarjeta Gráfica de NVIDIA GeForce 315M con Tecnología CUDA

Requisitos de Software

- Sistemas operativos Linux distribución Ubuntu 14.04
- Librería de Visión Artificial OpenCV 3.1
- Librería pública de AKAZE
- Compiladores de código en C++
- Matlab R2012
- Editor de código en C++
- Editor de texto para LaTeX Kile 2.0.85

Presupuesto

Presupuesto

En esta sección se adjunta un desglose de los costes estimados para este proyecto en cuanto a hardware, software y mano de obra se refiere.

Costes de ejecución material

Costes de hardware

Concepto	Precio unidad	Cantidad	Subtotal
Ordenador Toshiba Intel Core i5	699€	1	699€

Tabla 6.1: Costes de hardware.

Costes de software

Durante la realización del proyecto se han usado distintas aplicaciones para generar el código y la documentación. Por tanto, estas licencias deben ser tenidas en cuenta en el presupuesto.

Concepto	Coste de licencia
S.O. Ubuntu 14.04	0€
Librerías OpenCV 3.1	0€
Librerías AKAZE	0€
Editor y compilador de código en C++	0€
MatLab R2012	6524,45€
Google Drive	0€
Editor de texto para LaTeX Kile	0€

Tabla 6.2: Costes de software.

Costes de personal

Para estimar el coste de personal, desglosamos las horas dedicadas al proyecto en distintas tareas. Suponemos un sueldo medio de un Ingeniero Industrial de 15 euros por hora.

Concepto	Precio/hora	Horas	Subtotal
Instalación software	15	40	600€
Comprensión del estado del arte	15	180	2700€
Implementación código ABLE	15	160	2400€
Implementación código LIBVISO	15	280	4200€
Pruebas y evaluación de resultados	15	310	4650€
Redacción de la memoria	15	230	3450€

Tabla 6.3: Costes de personal.

Costes totales

Los costes totales de ejecución material se muestran en la tabla 6.4. En ella se realiza el sumatorio de los costes de personal, hardware y software.

Concepto	Subtotal
Costes de software	6524,45€
Costes de hardware	699€
Costes de personal	18000€
Subtotal final	25223,45€

Tabla 6.4: Costes totales.

Gastos generales y beneficio industrial

Los gastos generales y el beneficio industrial son desembolsos obligatorios que proceden de la utilización de las instalaciones de trabajo y los ingresos generados a nivel de manufactura. Para estas cuentas se estima un porcentaje del 16 % sobre el coste de ejecución material. Los resultados son los que se presentan a continuación.

Concepto	Subtotal
Costes de ejecución material	25223,45€
Portentaje estimado	16 %
Subtotal final	4035,75€

Tabla 6.5: Gastos generales y beneficio industrial.

Presupuesto de ejecución por contrata

En esta tabla se realiza el sumatorio de los resultados de los costes de ejecución material, los gastos generales y el beneficio industrial.

Concepto	Subtotal
Costes de ejecución material	25223,45€
Gastos generales y beneficio industrial	4035,75€
Subtotal final	29259,20€

Tabla 6.6: Presupuesto de ejecución por contrata.

Importe total del presupuesto

Para concluir, se calcula el presupuesto final del proyecto a partir de los costes totales de ejecución por contrata más el porcentaje de IVA del 21 % aplicado.

El importe total del presupuesto asciende a: 35403,63€**(Treinta y cinco mil cuatrocientos tres con sesenta y tres euros).**

Bibliografía

[1] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition (CVPR)*, 2004, pp. 652–659.

[2] H. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover," in *Ph.D. dissertation, Stanford Univ*, 1980.

[3] B. Kitt, A. Geiger, and H. Lategahn, "Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme," in *IEEE Intelligent Vehicles Symposium (IV)*, June 2010, pp. 486–492.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research (IJRR)*, vol. 32, no. 11, pp. 1231–1237, September 2013.

[5] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, J. J. Yebes, and S. Gámez, "Bidirectional loop closure detection on panoramas for visual navigation," in *IEEE Intelligent Vehicles Symposium (IV)*, June 2014, pp. 1378–1383.

[6] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, J. J. Yebes, and S. Bronte, "Fast and effective visual place recognition using binary codes and disparity information," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2014, pp. 3089–3094.

[7] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, and E. Romera, "Towards life-long visual localization using an efficient matching of binary sequences from images," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 6328–6335.

[8] X. Yang and K. T. Cheng, "LDB: An ultra-fast feature for scalable augmented reality on mobile devices," in *International Symposium on Mixed and Augmented Reality (ISMAR)*, November 2012, pp. 49–57.

[9] ——, "Local difference binary for ultrafast and distinctive feature description," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 36, no. 1, pp. 188–194, January 2014.

[10] M. Cummins and P. Newman, "Appearance-only SLAM at large scale with FAB-MAP 2.0." *International Journal of Robotics Research (IJRR)*, vol. 30, no. 9, pp. 1100–1123, November 2010.

[11] H. Badino, D. F. Huber, and T. Kanade, "Real-time topometric localization," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 1635–1642.

[12] P. F. Alcantarilla, J. Nuevo, and A. Bartoli, "Fast explicit diffusion for accelerated features in nonlinear scale spaces," in *British Machine Vision Conference (BMVC)*, September 2013.

[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, November 2004.

[14] H. Bay, A. Ess, T. Tuytelaars, and L. van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding (CVIU)*, vol. 110, no. 3, pp. 346–359, June 2008.

[15] A. Alahi, R. tyui, R. Ortiz, and P. Vandergheynst, "FREAK: Fast retina keypoint," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, June 2012, pp. 510–517.

[16] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *International Conference on Computer Vision (ICCV)*, November 2011, pp. 2548–2555.

[17] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *European Conference on Computer Vision (ECCV)*, September 2010, pp. 778–792.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 886–893, June 2005.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *International Conference on Computer Vision (ICCV)*, November 2011, pp. 2564–2571.

[20] E. Eade and T. Drummond, "Unified loop closing and recovery for real time monocular SLAM," in *British Machine Vision Conference (BMVC)*, vol. 6, September 2008, pp. 1–10.

[21] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (SLAM): Part I The essential algorithms," *IEEE Robotics and Automation Magazine (RAM)*, vol. 13, no. 2, pp. 99–110, June 2006.

[22] T. Bailey and H. Durrant-Whyte, "Simultaneous localisation and mapping (SLAM): Part II State of the art," *IEEE Robotics and Automation Magazine (RAM)*, vol. 13, no. 3, pp. 108–117, September 2006.

[23] F. M. Amor-Martinez, A. Ruiz and A. Sanfeliu., "On board real-time pose estimation for uavs using deformable visual contour registration," in *International Conference on Robotics and Automation (ICRA)*, 2014.

[24] H. Badino, "A robust approach for ego-motion estimation using a mobile stereo platform," in *First International Workshop on Complex Motion (IWCM)*, 2004, pp. 198–208.

[25] M. McGlove and Bethel., "Manual of photogrametry," in *American Society for Photogrammetry and Remote Sensing (ASPRS)*, 2004.

[26] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *IEEE International Workshop on Augmented Reality (IWAR)*, 1999.

[27] V. Lepetit and P. Fua, "Monocular model-based 3d tracking of rigid objects: A survey." *Foundations and Trends in Computer Graphics and Vision*, vol. 1, no. 1, 2005.

[28] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 3946–3952.

[29] L. M. A. Talukder, S. Goldberg and A. Ansar, "Real-time detection of moving objects in a dynamic scene from moving robotic vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 1308–1313.

[30] A. Talukder and L. Matthies, "Real-time detection of moving objects from moving vehicles using dense stereo and optical flow," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 3718–3725.

[31] M. Agrawal and K. Konolige, "Rough terrain visual odometry," in *International Conference on Advanced Robotics (ICRA)*, 2007.

[32] C. Dornhege and A. Kleiner, "Visual odometry for tracked vehicles," in *IEEE International Workshop on Safety, Security and Rescue Robotics (IWSSRR)*, 2006.

[33] Y. C. A. Johnson, S. Goldberg and L. Matthies, "Robust and efficient stereo feature tracking for visual odometry," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 39–46.

[34] M. Agrawal and K. Konolige, "Real-time localization in outdoor environments using stereo vision and inexpensive GPS," in *International Conference on Pattern Recognition (ICPR)*, August 2006, pp. 1063–1068.

[35] K. K. M. Agrawal and R. C. Bolles, "Localization and mapping for autonomous navigation in outdoor terrains: A stereo vision approach," in *IEEE Workshop on Applications of Computer Vision (WACV)*, 2007.

[36] M. Milford, "Visual route recognition with a handful of bits," in *Robotics Science and Systems Conference (RSS)*, July 2012.

[37] M. Muja and D. G. Lowe, "Fast matching of binary features," in *Canadian Conference on Computer and Robot Vision (CRV)*, May 2012, pp. 404–410.

[38] P. F. Alcantarilla, J. Nuevo, and A. Bartoli, "Fast explicit diffusion for accelerated features in nonlinear scale spaces," in *British Machine Vision Conference (BMVC)*, September 2013.

[39] R. Paul and P. Newman, "FAB-MAP 3D: Topological mapping with spatial and visual appearance," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 2649–2656.

[40] C. Cadena, D. Gálvez-López, J. D. Tardós, and J. Neira, "Robust place recognition with stereo sequences," *IEEE Transactions on Robotics (TRO)*, vol. 28, no. 4, pp. 871–885, August 2012.

[41] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., 2004.

[42] M. C. Bosse, "Atlas: A framework for large scale automated mapping and localization," Tech. Rep., 2004.

[43] M. Cummins and P. Newman, "Probabilistic appearance based navigation and loop closing," in *IEEE International Conference on Robotics and Automation (ICRA)*, April 2007, pp. 2042–2048.

[44] U. Frese, "Closing a million-landmarks loop," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 5032–5039.

[45] N. Sünderhauf, P. Neubert, and P. Protzel, "Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons," in *Workshop on Long-Term Autonomy at the IEEE International Conference on Robotics and Automation (W-ICRA)*, May 2013.

[46] A. Oliva and A. Torralba, "Building the gist of a scene: The role of global image features in recognition," in *Visual Perception, Progress in Brain Research (PBR)*, vol. 155, December 2006, pp. 23–36.

[47] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 30, no. 2, pp. 328–341, February 2008.

[48] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "KAZE features," in *European Conference on Computer Vision (ECCV)*, vol. 7577, October 2012, pp. 214–227.

[49] A. Hernandez-Gutierrez, J. I. Nieto, T. Vidal-Calleja, and E. Nebot, "Large scale visual odometry using stereo vision," in *Australasian Conference on Robotics and Automation (ACRA)*, December 2009, pp. 2–4.

[50] M. Milford and G. F. Wyeth, "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 1643–1649.