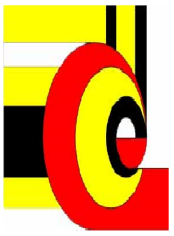


INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Localización y mapeo simultáneo en interiores mediante sensor RGBD

TESIS

QUE PARA OBTENER EL GRADO DE:

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

OMAR EDGARDO LUGO SÁNCHEZ

DIRECTOR DE TESIS:

DR. JUAN HUMBERTO SOSSA AZUELA

28 de julio de 2015

SIP-14

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 16 del mes de junio de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Localización y mapeo simultáneo en interiores mediante sensor RGBD"

Presentada por el alumno:

LUGO	**SÁNCHEZ**	**OMAR EDGARDO**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: | B | 1 | 3 | 0 | 1 | 0 | 0 |

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de Tesis

Dr. Juan Humberto Sossa Azuela

Dr. Sergio Suárez Guerra

Dr. Herón Molina Lozano

Dra. Elsa Rubio Espino

Dr. Ricardo Menchaca Méndez

Dr. Francisco Hiram Calvo Castro

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION
EN COMPUTACION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México D.F. el día 15 del mes Julio del año 2015, el (la) que suscribe Omar Edgardo Lugo Sánchez alumno (a) del Programa de Maestría en ciencias de la computación con número de registro B130100, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Juan Humberto Sossa Azuela y cede los derechos del trabajo intitulado Localización y mapeo simultáneo en interiores mediante sensor RGBD, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección lugo.oe@gmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Lugo Omar

Nombre y firma

Resumen

Para que un robot sea capaz de explorar un entorno que lo rodea, este debe de poder localizarse a si mismo y construir un mapa del ambiente al mismo tiempo. El problema recibe el nombre de localización y mapeo simultáneo (SLAM por sus siglas en inglés). En la presente tesis se resuelve este problema mediante el uso de un sensor RGB-D para percibir el mundo, él cual se opera manualmente en 6 grados de libertad. La solución del problema SLAM requiere la estimación de la posición y trayectoria del sensor en el ambiente tridimensional de forma precisa; por lo cual se construye un mapa 3D del entorno para estimar la ubicación del sensor. El mapa se construye mediante el registro consecutivo de las nubes de puntos 3D aportados por el sensor RGB-D. En esta etapa se hace uso de las características visuales SURF, y se calcula las transformaciones geómetras mediante los algoritmos RANSAC e ICP. También se realiza una corrección de inconsistencias para lograr la reconstrucción del mapa 3D de manera consistente y lograr una estimación más precisa de la trayectoria del robot. Esto se hace aplicando técnicas de optimización no lineales mediante el algoritmo de SLAM basado en grafos.

Abstract

For a robot to be able to explore an environment that surrounds it, it should be able to locate itself and build a map of the environment at the same time. The problem is called simultaneous localization and mapping (SLAM for its acronym in English). In this thesis this problem is solved, in which an RGB-D sensor to sense the world is used, and which is moved manually in 6 degrees of freedom. SLAM problem solution requires estimating the position and trajectory of the sensor in the three dimensional environment accurately. Whereby a 3D map of the environment to estimate the location of the sensor is constructed. The map is constructed by consecutive registration of 3D point cloud provided by the RGB-D sensor. At this stage the use of visual features SURF is made, and the geometric transformations are calculated using the algorithms RANSAC and ICP . A correction of inconsistencies are made in order to build betters 3D maps of the environment, and also to improve the estimation of the sensor trajectory, this is done using nonlinear optimization techniques with the graph-SLAM algorithm.

Agradecimientos

A mis padres, Ramón Efraín Lugo Sepúlveda, y Fabiola Sánchez Ley, por el apoyo incondicional que me han dada toda mi vida. A mi director de tesis el Dr. Juan Humberto Sossa Azuela por todo la ayuda brindada.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología por haberme aceptado como becario de tiempo completo para el desarrollo de la siguiente tesis y por el soporte económico otorgado durante dos años. Se agradece el apoyo brindado al IPN, a través del programa PIFI y COFAA; a la SIP-IPN a través de los proyectos: mediante de los proyectos: SIP 20121311, SIP 20131182 SIP 20131505, SIP 20131514 y a CONACyT por el proyecto 155014.

Índice general

Índice de figuras

Índice de tablas

Abreviaciones

RGB-D	**R**ed **G**reen **B**lue - **D**epth
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
RANSAC	**Ran**dom **Sa**mple **C**onsensus
ICP	**I**terative **C**losest **P**oint
SURF	**S**peeded **U**p **R**obust **F**eature
RPE	**R**elative **P**ose **E**rror
ATE	**A**solute **T**rajectory **E**error
EKF	**E**xtended **K**alman **F**ilter

Dedicado a mis padres.

Capítulo 1

Introducción

1.1. Localización y mapeo simultáneo

Unos de los principales objetivos de la robótica desde sus inicios han sido la creación de robots que puedan operar automáticamente en escenarios del mundo real, los cuales tienen un amplio rango de aplicaciones, coches de auto-conducción, vehículos aéreos no tripulados, vehículos submarinos autónomos, rovers planetarios, robots domésticos de reciente aparición, en situaciones de alto riesgo para las personas y un gran número de usos en la industria. Una de las habilidades necesarias para estos robots autónomos es que puedan saber su posición y trayectoria en el medio que interaccionan. Debido a que no se pueden hacer suposiciones sobre el medio, los robots tienen que aprender de su entorno. Esta capacidad ha sido identificado como un problema fundamental en la robótica [1]. Esta habilidad consiste en que el robot pueda construir un mapa del medio ambiente, mediante la información obtenida por sus sensores y además pueda localizarse en él. Al conjunto de algoritmos y técnicas que se tratan en este problema dual recibe el nombre de localización y mapaeo simultáneo (SLAM por sus siglas en inglés). Unos de los principales problemas que se enfrentan en SLAM es debió al ruido sensorial, limitaciones físicas presentes en los sensores, sobre todo a los errores inherentes de odometría, por lo que se utilizan modelos probabilistas para reducir estos errores inherentes y realizar mejores estimaciones.

Durante años, diferentes sensores se han utilizado para tratar el problema de SLAM, mucha de esta investigación se ha basado en el uso de sensores de escaneo de láser 2D del ambiente que los rodea, estos sensores aunque precisos son costosos[2][3]. También se ha investigado en el uso de cámaras estéreo [4], recientemente a surgido un gran interés en el uso de sensores RGB-D debido al los precios reducidos de estos sensores como es el ejemplo de la cámara KINECT [5], además los sensores que solo suelen proporcionar

datos de nubes de puntos aunque resultan ser muy apropiados para realizar registro entre estos datos, ignoran información muy útil contenida en las imágenes. Por lo que los sensores RGB-D aportan información visual muy útil que puede ser utilizada para resolver unos de los principales problemas en SLAM, que es detectar lugares visitados con anterioridad. Los sensores RGB-D son cámaras que capturan información RGB y mediante un láser pueden medir la profundidad de cada uno de los pixeles de la imagen [5]. Estos sensores han existido durante años, sin embargo, en los últimos años se han vuelto más populares debido a su accesible precio, es por eso que se ha optado por usar este sensor en este trabajo de tesis para resolver el problema de SLAM.

Se puede decir que los sistemas SLAM se componen de estos módulos principales. El primero es el encargado de realizar una estimación de la trayectoria del robot mediante la alineación de los datos adquiridos por su sensores. El módulo también se encarga de reconocer lugares visitados anteriormente y el último se encarga de realizar una optimización global de los datos para minimizar las inconsistencias inherentes al realizar las estimaciones.

1.2. Objetivo general

El objetivo que se persigue en esta tesis es el desarrollo de un sistema que permita la construcción de mapas 3D a partir de los datos captados mediante un sensor RGB-D. Este sensor se podrá mover libremente en 6 grados de liberad (SLAM 6D). Esto requiere la estimación de la posición y orientación del sensor Kinect en su entorno de forma precisa. Para atacar este problema se hace uso de la información RGB y la profundidad captadas por el sensor Kinect y se hace una primera aproximación de la trayectoria del sensor siguiendo un enfoque de alineación de nubes de puntos de forma sucesiva. Como este enfoque no es suficiente para construir mapas consistentes debido a la acumulación del error en cada una de las alineaciones consecutivas de nubes de puntos y también por el ruido inherente en la odometría y en los sensores [6], se implementa el algoritmo de SLAM basado en grafo. El cual permite dar solución al problema de cierre de ciclos y obtener mejores estimaciones de la trayectoria de la cámara y reconstrucciones más precisas del mapa 3D del ambiente.

1.3. Objetivos particulares

1. Implementar el algoritmo de SLAM basado en grafo.

2. Calcular la trayectoria del sensor Kinect al realizar alineación de nubes de puntos consecutivas.

3. Emparejamiento de características visuales 2D que se corresponden en dos imágenes consecutivas.

4. Aproximación a la pose mediante RANSAC con los emparejamientos encontrados.

5. Refinamiento de la pose por métodos iterativos con todos los puntos 3D, como ICP.

6. Detectar el paso por lugares visitados anteriormente (cierre de ciclos) y optimización del grafo de poses.

1.4. Limitaciones

En este proyecto el sensor Kinect siempre será operado manualmente y no se utilizará ninguna plataforma de robot. Los diferentes algoritmos para la detección de características visuales, estimación de poses y construcción de mapas, etc., serán implementados con bibliotecas de código abierto, pero es una de las metas de este trabajo hacer ajustes para que sirvan a nuestros propósitos. No está en el alcance de este proyecto añadir sensores adicionales o una plataforma robot, tampoco el hardware existente será modificado de ninguna manera.

El trabajo se pensó para que las mediciones se realizaran en interiores, con el fin de no exponer al sensor Kinect a demasiada radiación infrarroja, en cuyo caso las mediciones del sensor de profundidad se ven afectadas, entre otros factores que pueden afectar a las mediciones del sensor Kinect en exteriores. Cuando se recogen los datos se espera que no existan objetos en movimiento en el medio ambiente, ya que tales perturbaciones pueden afectar el algoritmo de una manera impredecible [7][8][9][10].

No habrá ninguna demanda sobre rendimiento en tiempo real en los algoritmos en este proyecto. El rendimiento en tiempo real se refiere a la capacidad de mapear una habitación en el mismo tiempo que se mueve el sensor Kinect para explorar. También significa que todos los cálculos para construir el mapa se completan en un ritmo tal que el sensor pueda moverse libremente en el medio ambiente y que todavía tenga la estimación más reciente de los alrededores. Aunque este proyecto no tiene la intención de cumplir con cualquier objetivo de rendimiento en tiempo real, se dedicó tiempo en tratar de obtener un rendimiento en tiempo real del sistema.

1.5. Organización de la tesis

Esta tesis está organizada de la siguiente manera, se presenta:

Capítulo 2: El estado del arte para el problema de SLAM.

Capítulo 3: Los principales aspectos teóricos en el problema de SLAM.

Capítulo 4: El hardware y software usados para implementar este proyecto.

Capítulo 5: Las diferentes metodologías que se usan para construir un mapa a partir de las mediciones obtenidas del sensor Kinect.

Capítulo 6: El algoritmo de extracción de características que se utilizó en este trabajo.

Capítulo 7: Como se implementó en este proyecto la construcción del grafo de poses así como la solución que se dio al problema de cierre de ciclos.

Capítulo 8: El paso adicional que se agregó en este trabajo para la eliminación de datos atípicos en dos dimensiones.

Capítulo 9: La discusión de los resultados experimentales obtenidos. Los experimentos realizados pretenden medir el desempeño del sistema implementado.

Capítulo 10: Las conclusiones a las que se llegaron y los trabajos futuros.

Capítulo 2

Estado del arte

En este capítulo se hace una revisión de las tendencias principales en el estado del arte más recientes y a través de la historia para el problema de localización y mapeo simultáneo (SLAM). Se presentan las soluciones que se han propuesto a los problemas y retos presentes en SLAM. Se prestan también los modelos más populares que se han propuesto para atacar el problema: filtro extendido de Kalman, filtro de partículas y el SLAM basado en grafo. También se discute sobre los sensores que se han empleado en las investigaciones realizadas.

2.1. Localización y mapeo simultáneo

La localización y mapeo simultáneo es el proceso mediante el cual una entidad (robot, vehículo o incluso un sensor llevado por una persona) tiene la capacidad para construir un mapa global del medio ambiente visitado y, al mismo tiempo, utilizar este mapa para deducir su propia ubicación en cualquier momento.

Con el fin de construir un mapa del entorno, la entidad debe poseer sensores que le permiten percibir y obtener mediciones de los elementos del mundo circundante. Estos sensores se clasifican en externos e internos. Entre los sensores externos es posible encontrar: sonares [11][12], láseres [13][14], cámaras [15][16] y sistemas de posicionamiento global (GPS) [17]. Todos estos sensores son ruidosos y tienen capacidades de alcance limitado. Además, solo vistas locales del entorno se pueden obtener usando los tres primeros sensores antes mencionados. Los sensores láser y el sonar permiten información precisa y muy densa de la estructura de medio ambiente. Sin embargo, tienen los siguientes problemas: no son útiles en entornos altamente desordenados o para el reconocimiento de objetos, ambos son caros, pesados y consisten de grandes piezas de equipo, por lo que

su uso es difícil para los robots humanoides o transportados por el aire. Por otro lado, un sensor GPS no funciona bien en las calles estrechas o bajo el agua.

2.2. SLAM visual

En la última década, se ve una tendencia por el uso de la visión como el único sistema de percepción sensorial externa para resolver el problema de SLAM [18][19][20][21][22]. La razón principal de esta tendencia se atribuye a la capacidad de un sistema basado en visión para obtener información de distancia y también por la información visual que aporta como el color y la textura del ambiente, dando al un robot la posibilidad de integrar otras tareas de alto nivel, como la detección y el reconocimiento de personas y lugares. Además, las cámaras son menos costosas, más accesibles, más ligeras y tienen menor consumo de energía. Desafortunadamente, puede haber inconvenientes presentes en estos sensores que pueden ocasionar errores por los siguientes motivos: insuficiente resolución de la cámara, cambios de iluminación, superficies con falta de textura, imágenes borrosas debido a movimientos rápidos de la cámara, entre otros.

Los primeros trabajos sobre la navegación visual se basaron en una configuración estéreo [23]. Las configuraciones estéreo ofrecen la ventaja de ser capaces de calcular fácilmente y con precisión las posiciones reales en 3D de los puntos de referencia que figuran en la escena, por medio de la triangulación [24], que es información de gran utilidad en el problema SLAM visual. En los trabajos [25][26][27] se representan los sistemas más eficaces de SLAM estéreo de la actualidad.

A pesar de que las muchas contribuciones que se han hecho en el área de SLAM visual, todavía hay muchos problemas. Muchos sistemas de SLAM visuales sufren de acumulación del error mientras que el entorno se está explorando (o fallan completamente), lo que conduce a estimaciones inconsistentes de la posición del robot y mapas totalmente incongruentes. Existen tres razones principales:

1. Generalmente se asume que se realiza un suave movimiento de la cámara y que habrá consistencia en la aparición de características [28][29], pero en general esto no es cierto. Los supuestos anteriores están altamente relacionados con la selección del detector de características y de la técnica de correspondencia utilizada. Esto origina una imprecisión en el cálculo de la trayectoria de la cámara por capturar imágenes con poca textura o que son borrosas debido a los movimientos rápidos de la cámara [30]. Estos fenómenos son típicos cuando la cámara es operada por una persona, robots humanoides y helicópteros de cuatro rotores, entre otros.

2. La mayoría de los investigadores asumen que los entornos a explorar son estáticos y que sólo contienen elementos fijos y rígidos, pero la realidad es que la mayoría de los entornos contienen personas y objetos en movimiento. Si esto no se considera, se generaran errores impredecibles en todo el sistema. Algunas soluciones propuestas para este problema se pueden encontrar en los trabajos [7][8][9][10].

3. El mundo es visualmente repetitivo. Hay muchas texturas similares en el ambiente, esto provoca que sea difícil reconocer un área previamente explorada y también hace difícil la tarea de SLAM en grandes extensiones.

2.3. Puntos de interés y características

Una característica se define como una parte "interesante" de una imagen. Las características se utilizan como punto de partida para muchos algoritmos de visión por computadora ya que se utilizan como principales primitivas para los algoritmos. En general, el algoritmo será sólo tan bueno como su detector de características. En el problema de SLAM visual, las características se usan para relacionar dos imágenes, tarea clave en el problema de asociación de datos. Una característica de buena calidad tiene las siguientes propiedades: debe ser notable (fácil de extraer), preciso (que se puede medir de manera exacta) e invariantes a la rotación, traslación, escala y cambios de la iluminación [15]. Por lo tanto, una característica de buena calidad tiene una apariencia similar desde diferentes puntos de vista en el espacio 3D. El proceso de extracción característica se compone de dos fases: La detección consiste en el procesamiento de la imagen para obtener una serie de elementos sobresalientes. La descripción consiste en la construcción de un vector de características basadas en la apariencia visual en la imagen. La invariancia del descriptor a los cambios en la posición y orientación permite una mejor comparación de imágenes y asociación de datos.

2.3.1. Detectores

Hay un gran número de detectores de características. Algunos ejemplos son: detector de esquinas [31], invariantes afines [32], de diferencia gaussiana utilizada en SIFT (scale invariant feature transform) Reference105, que utilizan la *Hessessiana* como en SURF (speeded up robust feature) [33], entre otros. En [34] el autor realiza una evaluación del desempeño de estos algoritmos, como resultado, se concluye que los detectores basados en la *Hessessiana* son mejores en presencia de características fuera de foco y que son más robustos ante otros factores. Se realizó otro estudio de los detectores mencionados

junto con algunos otros en [35]. En este estudio se tiene en cuenta las características de repetibilidad, precisión, robustez, eficiencia e invarianza.

2.3.2. Descriptores

En [36] aparece un estudio comparativo de diferentes algoritmos de descripción de características centrados en el problema de SLAM visual. La evaluación se basa en el número de coincidencias correctas e incorrectas que se encuentran a través de secuencias de video con cambios significativos en la escala, punto de vista y de iluminación. En este trabajo se demuestra que el descriptor SURF es superior al descriptor SIFT en términos de robustez y tiempo de cómputo. Más adelante, los autores manifiestan que SIFT no demuestra una gran estabilidad, lo que significa que muchos de los puntos de referencia detectados desde una posición particular de la cámara desaparecen cuando se mueve ligeramente. Actualmente hay muchas variantes que mejoran el rendimiento del algoritmo SIFT, por ejemplo: ASIFT, que incorpora a la invariancia a las transformaciones afines [37], BRIEF (Binary Robust Independent Elementary Features) [38], ORB, un descriptor binario rápido basado en BRIEF, pero invariante a la rotación y resistente al ruido [39], entre otros.

2.4. El problemas de la asociación de datos

La correspondencia entre imágenes consiste en la búsqueda de características extraídas por un detector en una imagen y su correspondiente característica en otra imagen. En el área de navegación de robots, la asociación de datos consiste en relacionar las mediciones del sensor con los elementos ya dentro del mapa del robot [40]. Este problema también consiste en determinar si las mediciones son espurias o pertenecen a elementos que no aparecen en el mapa. Una comparación de imágenes eficiente y una asociación de datos correcta son esenciales para una navegación con éxito, ya que los errores conducirán rápidamente a mapas incorrectos.

En los últimos años, ha habido un progreso considerable en el desarrollo de algoritmos para la búsqueda de características que son invariantes a varias transformaciones de la imagen. Muchos de estos algoritmos obtienen un descriptor para cada característica detectada y se calculan medidas de disimilitud entre los descriptores y estructuras de datos para realizar la búsqueda de parejas de forma rápida y eficiente.

Hay varias medidas de disimilitud, como la distancia euclidiana, distancia Manhattan, distancia Chi cuadrada, entre otros. Con las estructuras de datos se pueden crear otros criterios de disimilitud como los árboles ks o tablas hash [41]. Algunos ejemplos son:

1. Umbral de distancia: dos características están relacionadas si la distancia entre sus descriptores está por debajo de un umbral.

2. Vecino más cercano: A y B están relacionados si el descriptor B es el vecino más cercano del descriptor A y si la distancia entre ellos es por debajo de un umbral.

3. Relación de distancia del vecino más cercano: este enfoque es similar al vecino más cercano, excepto que el umbral se aplica a la relación de las distancias del píxel actual con el primer y segundo vecino más cercano.

El uso de descriptores de alta calidad o incluso diferentes tipos de medidas de similitud no garantizan evitar falsas correspondencias. Si se utilizan estas correspondencias dentro de un sistema SLAM, errores importantes serán generados para la estimación de la cámara y el mapa. Por lo tanto, es necesario el uso de estimadores robustos como RANSAC (Random sample consensus), PROSAC (Progressive sample consensus), entre otros, pueden manejar automáticamente falsas correspondencias. Un análisis comparativo de estos estimadores se puede encontrar en [42]. La principal diferencia entre ellos es la forma en que evalúan la calidad de un modelo. Los estimadores robustos se utilizan comúnmente para estimar los parámetros de un modelo a partir de datos que contienen valores atípicos. RANSAC estima una relación global de la adaptación de los datos y al mismo tiempo clasifica los datos bajo datos típicos (datos que son consistentes con la relación) y valores atípicos (no consistentes con la relación). Debido a la capacidad de tolerar una gran cantidad de valores atípicos, este algoritmo es una opción popular para resolver una gran variedad de problemas de estimación.

2.5. Cierre de ciclos

La detección de cierre de ciclo consiste en el reconocimiento de un lugar que ya ha sido visitado anteriormente [43]. Este problema ha sido uno de los mayores impedimentos para realizar SLAM a gran escala y recuperarse de errores críticos. A partir de este problema surge otro llamado “aliasing perceptual” [44], donde dos lugares diferentes se reconocen como el mismo. Esto representa un problema incluso cuando se usan cámaras como sensores debido a las características repetitivas del medio ambiente. Un buen método de detección de cierre de ciclos no debe devolver ningún falso positivo y debe obtener un mínimo de falsos negativos.

En [45] se presenta un método para recuperarse de los fallos de seguimiento y detectar los cierres de ciclo en el problema de SLAM visual en tiempo real. Con el fin de detectar fallos o cierres de ciclo, modelan las imágenes como una “bolsa de palabras” para encontrar los

nodos que tienen una apariencia similar a la imagen. En [46] se presenta un método para detectar cierres de ciclo bajo un esquema de filtrado Bayessiano y un método de "bolsa de palabras", donde la probabilidad de pertenecer a una escena visitada se calcula para cada imagen adquirida. [47] proponen un marco probabilístico para reconocer lugares, que utiliza sólo los datos de aspecto de la imagen. A través de la formación de un modelo generativo de la apariencia, demuestran que no sólo es posible calcular la semejanza de dos observaciones, sino también la probabilidad de que pertenezcan al mismo lugar, por lo que calculan una función de distribución de probabilidad de la posición observada.

Todas las obras de cierre de ciclos descritos anteriormente, tienen como objetivo lograr una precisión del 100 %. Esto es debido al hecho de que un solo falso positivo puede causar fallos irremediables durante la creación del mapa. En el contexto de SLAM, los falsos positivos son más graves que falsos negativos [48].

2.6. Soluciones al problema de SLAM visual

Desde la década de los noventas, los enfoques probabilísticos se han convertido en la tendencia dominante que busca resolver el problema de SLAM. Los enfoques que se convirtieron en las tres soluciones más populares son : filtro extendido de Kalman (EKF), filtro de partículas y el SLAM basado en grafo. EKF SLAM fue la primera solución dada para SLAM. Pero este método ha sido un poco impopular debido a su limitación de alta complejidad computacional. El filtro de partículas es una nueva solución para resolver el problema de SLAM y proporciona nuevas soluciones al problema de asociación de datos. La tercera forma de resolver el problema SLAM se basa en un grafo, el cual ha sido aplicada con éxito en algunos problema de SLAM. En las siguientes secciones se presenta un resumen de estas soluciones.

2.6.1. Filtro extendido de Kalman

La formulación EKF de SLAM fue la primera y ha sido por lo mismo una de las más influyentes. EKF para SLAM se presento en los trabajos [49][50][51][52], los cuales fueron los primeros trabajos en proponer el uso de un único vector de estado para estimar las ubicaciones del robot y un conjunto de características en el medio ambiente, con una matriz de covarianza asociada al error que representa la incertidumbre en estas estimaciones, incluyendo las correlaciones entre las estimaciones de la posición del robot y de la características. A medida que el robot se mueve a través de su medio ambiente, tomando mediciones, el vector de estado del sistema y la matriz de covarianza se actualizan con el

filtro extendido de Kalman [53][54][55]. Conforme se van observando nuevas características, se añaden nuevos estados al vector de estado del sistema. El gran inconveniente es que el tamaño de la matriz de covarianza del sistema crece cuadráticamente.

2.6.2. SLAM basado en grafo

Este enfoque resuelve el problema de SLAM través de la optimización no lineal. La idea de este enfoque se inspira en la representación gráfica del problema SLAM. Las técnicas basadas en grafos fueron mencionadas por primera vez en [56][57], pero solo fue hasta [6] que se desarrolló una solución funcional de este enfoque. La mayoría de las técnicas contemporáneas tratan el problema fuera de línea y abordar el problema de SLAM completo.

La idea básica de SLAM basado en grafo es la siguiente, los puntos de referencia y las posiciones del robot pueden ser considerados como nodos en un grafo. Cada par consecutivo de lugares $(X_T - 1, x_t)$ está conectado entre si por un arco que representa la información de la odometría u_t. Existen otros arcos entre los puntos de referencia m_i y posiciones del robot x_t, si en el tiempo t el robot detecta el punto de referencia i. Los arcos en este grafo son considerados como restricciones. Al ajustar estas restricciones, se obtiene una mejor estimación de las posiciones del robot o trayectoria y un mejor mapa.

2.6.3. Filtro de partículas

El tercer principal paradigma en SLAM se basa en el filtro de partículas. El filtro de partículas se remontan a [58], pero sólo en los últimos años es que han cobrado popularidad. El filtro de partículas representa la probabilidad posterior a través de un conjunto de partículas. Cada partícula se puede pensar como una conjetura concreta en cuanto a cuál puede ser el verdadero valor del estado. Mediante la recopilación de muchas de esas conjeturas en un conjunto de conjeturas, o un conjunto de partículas, el filtro de partículas captan una muestra representativa de la distribución posterior. Se ha demostrado que en condiciones controladas que se acerca a la verdadera posterior cuando el tamaño del conjunto de partículas tiende a infinito. También es una representación no paramétrica que representa distribuciones multimodales con facilidad.

Los filtros de partículas aplicables al problema de SLAM se remontan a [59][60]. El truco se introdujo en la literatura SLAM en [61], seguido de [62], que acuñó el nombre FastSLAM.

Capítulo 3

Marco teórico

En este capítulo se ofrece una introducción al problema de localización y mapeo simultáneo, más conocido en su forma abreviada como SLAM. Este aborda el problema de que un robot pueda navegar en un entorno desconocido. Durante la navegación por el medio ambiente, el robot busca adquirir un mapa del mismo y al mismo tiempo desea poder localizarse a sí mismo usando su mapa. La resolución del problema de SLAM puede ser motivada por dos razones: una podría ser por el interés de obtener un modelo detallado del entorno, la segunda, es que se podría tener interés de mantener una localización exacta de un robot en su entorno. SLAM sirve para estos dos propósitos.

Se revisa a modo de comparación para entender mejor el enfoque que se siguió en este trabajo, los tres grandes paradigmas y algoritmos que se han propuesto para atacar el problema de SLAM. Estos paradigmas han derivado en un gran número de métodos publicados a lo largo del tiempo. Primero se revisa el enfoque tradicional, que se basa en el filtro de Kalman extendido (EKF) para representar la mejor estimación de la ubicación del robot. El segundo paradigma extrae su intuición del hecho de que el problema SLAM puede ser visto como un grafo de restricciones, donde se aplica optimización no lineal para recuperar el mapa y la ubicación del robot, este es el enfoque seguido en este trabajo. Finalmente, se examina el paradigma de filtro de partículas, que se aplica estimación de la densidad no paramétrica y métodos de factorización eficientes al problema SLAM.

3.1. Definición SLAM

El problema SLAM (localización y mapeo simultáneos) se define como sigue: Un robot móvil transita un entorno desconocido, a partir de una localización con coordenadas conocidas, su movimiento es incierto, por lo que es difícil determinar sus coordenadas globales. Mientras explora, el robot puede percibir su entorno. El problema de SLAM trata

sobre de la construcción de un mapa del entorno mientras se determina simultáneamente la posición relativa del robot con respecto al mapa construido. Formalmente, SLAM se describe mejor en la terminología probabilista. Denotemos tiempo con t y la ubicación del robot por x_t. Para robots móviles en un terreno plano, x_t es por lo general un vector 3D, que consiste en dos de sus coordenadas en el plano mas un valor de rotación para su orientación. La secuencia de los lugares, o camino está dada por:

$$X_T = (x_0, x_1, x_2, ...x_T) \tag{3.1}$$

Aquí T es un tiempo finito, la ubicación inicial x_0 es conocida; las otras posiciones no pueden ser detectadas. La odometría proporciona información relativa entre dos ubicaciones consecutivas. Sea u_t que denota la odometría que caracteriza el movimiento entre el momento $t-1$ y el tiempo t; estos datos pueden ser obtenidos a partir de los codificadores de las ruedas del robot o de las acciones dadas a los motores. A continuación, la expresión 3.2 caracteriza el movimiento relativo del robot. Para movimiento libre de ruido, U_T sería suficiente para recuperar X_T de la ubicación inicial x_0. Sin embargo, las mediciones de odometría son ruidosos y las técnicas de integración difieren inevitablemente de la verdad.

$$U_T = (u_0, u_1, u_2, ...u_T) \tag{3.2}$$

Finalmente, el robot detecta objetos en el entorno. Sea M el mapa del medio ambiente. El medio ambiente puede estar compuesto de puntos de referencia, objetos, superficies, etc. y M describe sus ubicaciones. Generalmente se asume que el mapa de entorno M es invariante en el tiempo (es decir, estático).

Las mediciones del robot establecen relaciones entre las características en M y las ubicaciones X_T del robot. Sin pérdida de generalidad, suponemos que el robot toma exactamente una medición en cada punto en el tiempo, la secuencia de las mediciones está dada por:

$$Z_T = (z_0, z_1, z_2, ...z_T) \tag{3.3}$$

La figura 3.1 ilustra las variables involucradas en el problema de SLAM. Se muestra la secuencia de posiciones del robot y las mediciones de los sensores y las relaciones causales entre estas variables. Tal diagrama se conoce como modelo gráfico. Es útil en la comprensión de las dependencias en el problema de SLAM.

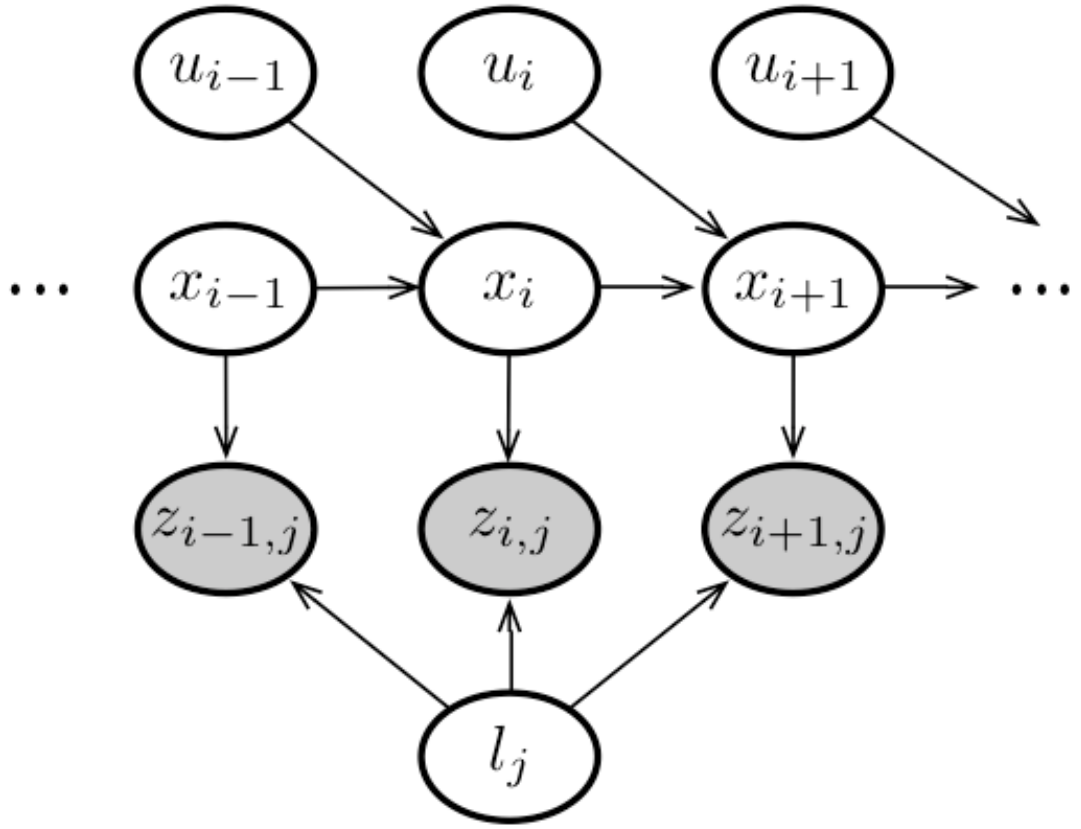


FIGURA 3.1: Los arcos indican relaciones causales y los nodos sombreados son directamente observables por el robot. En SLAM, el robot busca recuperar las variables no observables.

El problema de SLAM trata sobre la recuperación de un modelo del mundo M y la secuencia de posiciones del robot X_T a partir de los datos de odometría y de las mediciones. La literatura distingue dos formas principales del problema de SLAM, de igual importancia. Una es conocida con el nombre de SLAM completo: el cual trata de estimar la probabilidad a *posteriori* de toda la trayectoria del robot junto con el mapa y está denotado por la expresión:

$$p(X_T, m|Z_T, U_T) \tag{3.4}$$

El problema de SLAM completo consiste en calcular la probabilidad a *posteriori* de X_T y m a partir de los datos observables. Las variables Z_T y U_T son directamente observables en el robot, mientras que X_T y m son los que queremos calcular. A menudo los algoritmos para el problema de SLAM completo tratan todos los datos fuera de línea y se procesa toda la información al mismo tiempo.

La segunda forma del problema de SLAM, que es igualmente importante y es al que se le conoce como SLAM en línea. Este problema se define a través de la expresión:

$$p(x_T, m|Z_T, U_T) \tag{3.5}$$

Esta forma de SLAM busca recuperar la ubicación actual del robot, en lugar de todo el camino. Los algoritmos que abordan este problema son generalmente incrementales y procesan un elemento de datos a la vez. Típicamente este tipo de algoritmos son conocidos como filtros.

Para resolver cualquier de estas formas de SLAM, el robot debe contar con dos modelos más: un modelo matemático que relaciona las mediciones de odometría u_t con posiciones del robot $x_t - 1$ y X_t y un modelo que relacione las mediciones z_t con el medio ambiente m y la posición del robot X_t.

Es fácil plantear estos modelos matemáticos como distribuciones de probabilidad, por ejemplo: $p(x_t|x_t - 1, u_t)$ modela la distribución de probabilidad de la posición x_t si se supone que el robot comenzó en una posición conocida $x_t - 1$ y además se tienen los datos de odometría, u_t, $p(Z_T|x_t, m)$ es la probabilidad de medir Z_T si esta medición se toma en una ubicación conocida x_t y un entorno conocido m. En la realidad, en el problema de SLAM no sabemos la ubicación del robot y tampoco conocemos el medio ambiente. Sin embargo, la regla de Bayes nos ayudará con esto ya que hace uso de estas relaciones matemáticas para poder recuperar las distribuciones de probabilidad de las variables utilizando los datos observables.

La estimación a *posteriori* dada en 3.5 implica que operan en espacios de alto estado dimensional. Esto no sería manejable si el problema de SLAM no tuviera una estructura bien definida. Esta estructura surge de ciertos hechos comúnmente supuestos, como el que el mundo es estático y la suposición de Markov. Una forma conveniente para describir esta estructura es a través de la red bayesiana dinámica, representada en la figura 3.2. Una red bayesiana es un modelo de grafo que describe un proceso estocástico como un grafo dirigido, el grafo tiene un nodo para cada variable aleatoria en el proceso y una arista dirigida entre dos nodos del modelos que representa una dependencia condicional entre ellos.

En la figura 3.2, se pueden distinguir nodos azules y grises que indican las variables observadas (z_1 y u_1) y nodos blancos que son las variables ocultas. Las variables ocultas x y m modelan la trayectoria del robot y el mapa del medio ambiente. La conectividad de la red sigue un patrón recurrente que se caracteriza por el modelo de transición de estados y por el modelo de observación. El modelo de transición $p(x_t|x_{t-1}, u_t)$ está representado

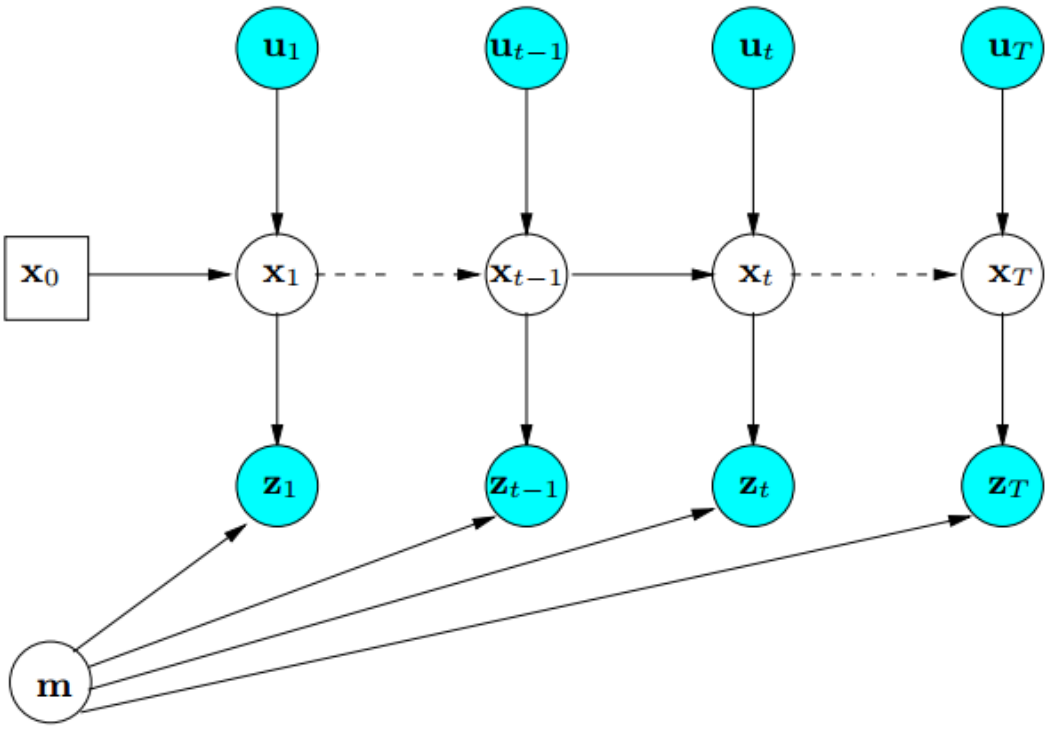


FIGURA 3.2: Red bayesiana dinámica del proceso de SLAM.

por las dos aristas que conducen a x_t y representan la probabilidad de que el robot en el tiempo t está en x_t dado que en el tiempo $t-1$ estaba en x_t y se adquirió la odometría por la medición u_t.

El modelo de observación $p(z_t|x_t, m_t)$ modela la probabilidad de realizar la observación z_t dado que el robot está en la posición x_t en el mapa. Está representado por las flechas que entran en z_t. La observación externa z_t depende sólo de la ubicación actual x_t del robot y de un mapa (estático) m. Expresar SLAM como red bayesiana destaca su estructura temporal y por lo tanto este formalismo es muy adecuado para describir procesos de filtrado que se pueden utilizar para hacer frente al problema de SLAM.

Una representación alternativa a la red bayesiana es a través de la denominada "basado en grafo" del problema de SLAM, que resalta la estructura espacial de SLAM subyacente. En el SLAM basado en grafo, las poses del robot se modelan por los nodos en un grafo y una etiqueta con su posición en el medio ambiente [6][63]. Las restricciones espaciales entre poses que resultan las observaciones z_t o a partir de mediciones de odometría u_t están codificadas en las aristas entre nodos. La idea de un algoritmo de SLAM basado en grafo es construir un grafo con las mediciones de los sensor. Cada nodo en el grafo representa una pose del robot y una medición adquirida en la pose. Una arista entre dos nodos representa una restricción espacial que relaciona las dos poses del robot.

Una restricción consiste en una distribución de probabilidad sobre las transformaciones relativas entre las dos poses. Estas transformaciones son mediciones de odometría entre las posiciones secuenciales del robot o se determinan mediante la alineación de las observaciones adquiridas en dos poses del robot. Una vez que el grafo se construye, se busca encontrar la configuración de las poses del robot que mejor satisfaga las restricciones. Así, en el SLAM basado grafo el problema se desacopla en dos tareas: la construcción del grafo de las mediciones (construcción del grafo) y determinar la configuración más probable de las poses dados las aristas del grafo (optimización del gráfo). La construcción del grafo se suele llamar "front-end" y es fuertemente dependiente del sensor, mientras que la segunda parte se llama "back-end" y se basa en una representación abstracta de los datos y la optimización de estos.

3.2. SLAM y los puntos de referencia

Una suposición común en el problema de SLAM es que el ambiente que se quiere explorar contiene puntos de referencia. Tomaremos como ejemplo un mapa de 2 dimensiones. Los puntos de referencia corresponden a puntos fácilmente reconocibles, los cuales cuando se proyecta en un mapa en dos dimensiones se caracterizan por un punto de coordenadas. En un mundo en dos dimensiones, cada punto de referencia está caracterizado por dos valores de coordenadas. De ahí que el mundo es un vector de tamaño $2N$, donde N es el número de puntos de referencia en el ambiente.

En este entorno supondremos que el robot puede detectar tres cosas: la distancia relativa a los puntos de referencia, su orientación relativa y la identidad de estos puntos de referencia. La distancia y la orientación pueden ser ruidosas, pero la identidad de los puntos de referencia se sabe perfectamente.

Para modelar una configuración de este tipo, se parte de la definición exacta de la función de medición libre de ruido. La función de medición h describe el funcionamiento de los sensores: la cual acepta como entrada una descripción del entorno m y una ubicación x_t del robot y calcula la medición:

$$h(x_t, m) \tag{3.6}$$

El modelo de medición probabilístico se deriva de esta función de medición mediante la adición de un término de ruido; se trata de una distribución de probabilidad que tiene como valor máximo $h(x_t, m)$ o sea, libre de ruido, pero permite la medición de ruido:

$$p(z_t|x_t, m)\ N(h(x_t, m), Q_t) \tag{3.7}$$

Donde N indica la distribución normal en tres dimensiones, que se centra en $h(x_t, m)$. La matriz de 3×3 Q_t es la covarianza del ruido, indexada por el tiempo.

El modelo de movimiento se deriva de un modelo cinemático del movimiento del robot. Dado el vector de ubicación $x_t - 1$ y el movimiento u_t. La cinemática nos dice como calcular x_t. Esta función se denota por g:

$$g(x_t - 1, u_t). \tag{3.8}$$

El modelo de movimiento se define entonces por una distribución normal centrada en $g(x_t - 1, u_t)$ pero sujeta a un ruido de gaussiano, donde R_t es una matriz de covarianza:

$$p(x_t|x_t - 1, u_t) = N(g(x_t - 1, u_t), R_t) \tag{3.9}$$

Con estas definiciones, tenemos todo lo que necesitamos para desarrollar un algoritmo de SLAM. Mientras que en la literatura, los problemas de puntos de referencia son los más estudiados, los algoritmos de SLAM no se limitan a éstos solamente. Sin importar qué representación del mapa se use, o qué tipo de sensor, todo algoritmo SLAM necesita una definición detallada de las características de m, o sea, el modelo de medición $p(z_t|x_t, m)$ y el modelo de movimiento $p(x_t|x_t - 1, u_t)$.

3.3. Diferentes formas de SLAM

El problema de SLAM se puede distinguir a lo largo de un número de diferentes dimensiones. La mayoría de los trabajos de investigación importantes identifican el tipo de problema de SLAM al hacer las suposiciones explícitas, por ejemplo, si trata de un SLAM completo contra uno en línea. Las distinciones comunes del problema de SLAM son las siguientes:

3.3.1. Volumétrico o basado en características

En el SLAM volumétrico, el mapa se muestrea a una resolución lo suficientemente alta como para permitir la reconstrucción fotorrealista del medio ambiente. El mapa m en el SLAM volumétrico es por lo general de bastante alta dimensión, con el inconveniente de

que el cómputo puede ser un poco complicado. En el SLAM basado en características se extrae características medidas por sensor. El mapa se compone entonces sólo de características. El ejemplo de puntos de referencia anterior es un ejemplo de SLAM basado en características. Éstas técnicas tienden a ser más eficientes, pero sus resultados pueden ser inferiores al SLAM volumétrico debido al hecho de que la extracción de características descarta información de las mediciones del sensor.

3.3.2. Topológico o métricas

Algunas técnicas de mapeo recuperan sólo una descripción cualitativa del medio ambiente, lo que caracteriza lugares básicos. Tales métodos son conocidos como topológicos. Un mapa topológico podría definirse sobre un conjunto de lugares distintos y un conjunto de relaciones cualitativas entre estos lugares (por ejemplo, el lugar A es adyacente a otro B). Los métodos de SLAM que utilizan métricas proporcionan información métrica entre la relación de los lugares. En los últimos años, los métodos topológicos han pasado de moda, a pesar de una amplia evidencia de que los humanos suelen utilizar información topológica para la navegación.

3.3.3. Correspondencias conocidas o desconocidas

El problema de correspondencia consiste en relacionar la identidad de las cosas detectadas a otras detectadas anteriormente. En el ejemplo anterior de los puntos de referencia, se asumió que la identidad de puntos de referencia era conocida. Algunos algoritmos de SLAM hacen tal suposición, mientras que otros no lo hacen. Los algoritmos que no hacen esta suposición proporcionan mecanismos especiales para la estimación de la correspondencia de características con características previamente observadas en el mapa. El problema de la estimación de correspondencias se conoce como el problema de asociación de datos y es uno de los problemas más difíciles en SLAM.

3.3.4. Estático o dinámico

Los algoritmos estáticos SLAM suponen que el medio ambiente no cambia con el tiempo. Los métodos dinámicos permiten cambios en el entorno. La gran mayoría de la literatura sobre SLAM asume ambientes estáticos; los efectos dinámicos a menudo son tratados igual que los valores atípicos en las mediciones.

3.3.5. Incertidumbre pequeña o grande

Los problemas de SLAM se distinguen por el grado de incertidumbre de la locación que pueden manejar. Los algoritmos de SLAM más simples permiten sólo pequeños errores en la estimación de localización. Son buenos para situaciones en las que un robot va por un camino que no se cruza consigo mismo y luego regresa por el mismo camino. En muchos entornos, es posible llegar al mismo lugar desde múltiples direcciones. Aquí, el robot puede acumular una gran cantidad de incertidumbre. Este problema se conoce como el problema de "cierre de ciclos". Cuando se cierra un ciclo, la incertidumbre disminuye. La capacidad de cerrar los ciclos es una característica clave de los algoritmos de SLAM de hoy en día.

3.4. Los tres paradigmas principales en SLAM

Esta sección revisa tres los paradigmas principales en SLAM. El primero, conocido como filtro extendido de Kalman (EKF), es históricamente el más antiguo, pero ha perdido popularidad debido a sus propiedades computacionales limitantes. El segundo, se basa en representaciones por grafos, se aplica con éxito mediante métodos de optimización no lineal al problema de SLAM y se ha convertido en el paradigma principal para resolver el problema de SLAM completo. Este método es el que se implementa en este trabajo. El tercero y último método utiliza técnicas de filtrado estadísticos no paramétricas conocido como filtro de partículas, el cual es un método popular para SLAM en línea.

3.4.1. Filtro extendido de Kalman

Este enfoque asume una representación métrica del medio ambiente basado en características, en el que los objetos pueden ser representados de manera efectiva como puntos en un espacio de parámetros apropiados. La posición del robot y las ubicaciones de las características forman una red de relaciones espaciales inciertas.

El algoritmo EKF representa la estimación del robot con gaussiana multivariante:

$$p(x_t, m|Z_T, U_T) \sim N(m_T, \Sigma_t). \tag{3.10}$$

El vector de alta dimensión μ_t contiene la mejor estimación del robot de su propia ubicación y la ubicación de las características del entorno. En un ambiente 3D, la dimensión de m_t sería $3 + 3N$ ya que necesitamos tres variables que representan la ubicación del robot y $3N$ variables para los puntos de referencia de N en el mapa.

La matriz Σ_t es la covarianza de la evaluación del robot de su error esperado de la conjetura m_t. Como una matriz cúbica, Σ_t es de tamaño $(3+3N) \times (3+3N) \times (3+3N)$. En SLAM, esta matriz es por lo general claramente no dispersa. Los elementos fuera de la diagonal capturan las correlaciones en las estimaciones de diferentes variables. Correlaciones no nulas aparecen porque la ubicación del robot es incierta y como resultado las localizaciones de los puntos de referencia en los mapas son inciertas. La importancia de mantener los elementos fuera de la diagonal es una de las propiedades fundamentales de EKF SLAM [64].

El algoritmo EKF SLAM se deriva fácilmente de nuestro ejemplo de puntos de referencia. Supongamos que la función de movimiento g 3.6 y la función de medición h 3.8 fueron lineales en sus argumentos. Entonces se parecería mejor el filtro Kalman y sería aplicable. EKF SLAM linealiza las funciones g y h utilizando series de Taylor, EKF SLAM no es otra cosa mas que la aplicación del modelo estándar de EKF al problema SLAM en línea.

La figura 3.3 ilustra el algoritmo de EKF SLAM . El robot navega desde una posición inicial que sirve como el origen de su sistema de coordenadas. Mientras se mueve, la incertidumbre de su propia posición aumenta, como lo indica las elipses de incertidumbre de diámetro creciente. También detecta lugares de referencia y los mapea con una incertidumbre que combina la incertidumbre de medición fija con la creciente incertidumbre de la posición. Como resultado de ello, la incertidumbre en los lugares de referencia crece con el tiempo. La transición interesante se ilustra en la figura 3.3: Aquí el robot observa el punto de referencia que vio en el comienzo mismo de la construcción del mapa y cuya ubicación es relativamente bien conocida. A través de esta observación, el error de la posición del robot se reduce, como se indica en la figura 3.3. notese la muy pequeña elipse de error para la posición final del robot. Esta observación también reduce la incertidumbre para otros puntos de referencia en el mapa. Este fenómeno surge de una correlación que se expresa en la matriz de covarianza de la probabilidad a *posteriori* gaussiana. Dado que la mayoría de la incertidumbre en las primeas estimaciones de los puntos de referencia es causada por la posición del robot y como esta misma incertidumbre persiste en el tiempo, las estimaciones de localización de los puntos de referencia están correlacionados. Cuando se obtiene información sobre la posición del robot, esta información se extiende a puntos de referencia previamente observados. Este efecto es probablemente la característica más importante en este modelo SLAM [64]. La información que ayuda a localizar el robot se propaga a través del mapa y como resultado mejora la localización de otros puntos de referencia.

EKF SLAM también aborda el problema de asociación de datos. Si se desconoce la identidad de las características observadas, la idea básica de EKF resulta inaplicable.

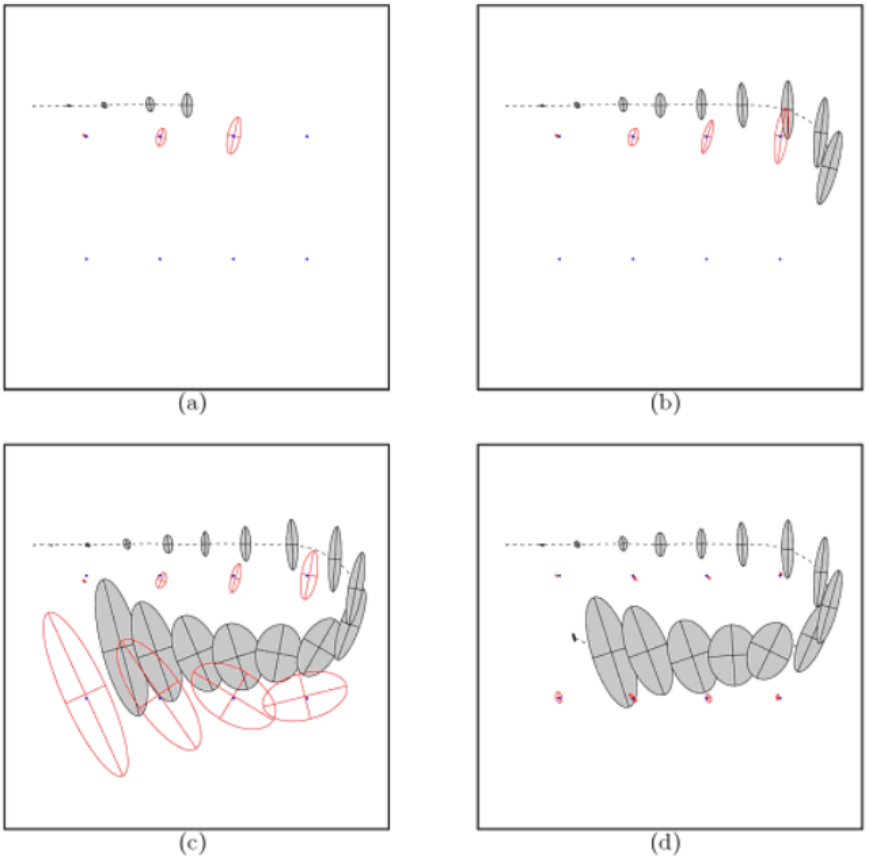


FIGURA 3.3: La trayectoria del robot es la línea punteada y la estimaciones de su propia posición son las elipses sombreadas. Se muestran ocho puntos de referencia distinguibles de la ubicación desconocida como pequeños puntos y las estimaciones de sus localización se muestran como elipses blancas. En (a-c) la incertidumbre de la posición del robot va en aumento, al igual que la incertidumbre acerca de los puntos de referencia que se encuentra. En (d) el robot detecta de nuevo el primer punto de referencia que detectó anteriormente y en consecuencia la incertidumbre de todos los puntos de referencia disminuye, como también la incertidumbre de su posición actual.

La solución calcula la asociación de datos más probable cuando se observa un punto de referencia. Esto se hace generalmente basado en la proximidad: ¿Cuál de los puntos de referencia en el mapa es más probable que corresponde al punto de referencia recientemente observado? El cálculo de proximidad considera el ruido de medición y la incertidumbre en la estimación real de la posición y la métrica utilizada en este cálculo se conoce como la distancia de Mahalanobis, que es una distancia cuadrática ponderada. Para minimizar las posibilidades de falsas asociaciones de datos, muchas implementaciones usan características visibles para distinguir puntos de referencia individuales y asocian grupos de puntos de referencia observados simultáneamente [65][66]. Implementaciones modernas también mantienen una lista provisional puntos de referencia y sólo añaden puntos de referencia al mapa interno cuando se han observado con la suficiente frecuencia [67][68][69]. Con una adecuada utilización de puntos de referencia y cuidadosa implementación de la etapa de asociación de datos, EKF SLAM se convierte en un

poderoso método para SLAM basado en características (punto de referencia).

3.4.2. SLAM basada en grafo

La idea básica de SLAM basado en grafo es la siguiente: los puntos de referencia y las posiciones del robot pueden ser considerados como nodos en un grafo. Cada par consecutivo de lugares $(X_T - 1, x_t)$ está conectado entre sí por un arco que representa la información de la odometría u_t. Existen otros arcos entre los puntos de referencia m_i y posiciones del robot x_t, si en el tiempo t el robot detecta el punto de referencia i. Los arcos en este grafo son considerados como restricciones blandas. Al ajustar estas restricciones, se obtiene una mejor estimación de las posiciones del robot o trayectoria y un mejor mapa.

La construcción del grafo se muestra en la figura 3.4. En el instante $t = 1$, el robot detecta el punto de referencia m_1, así que este añade un arco en el grafo entre x_1 y m_1. Si se almacena toda esta información en un formato de matriz, se puede apreciar que es similar a una ecuación cuadrática que define las restricciones, como se muestra en el lado derecho de la figura 3.4.

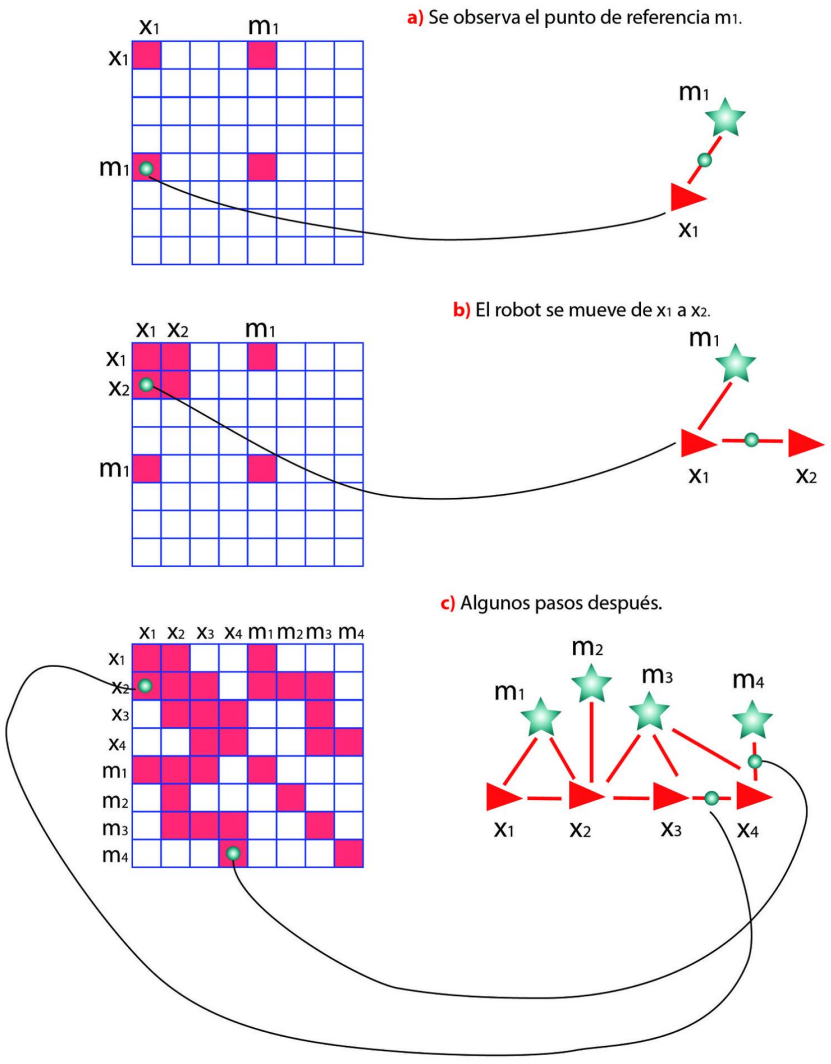


FIGURA 3.4: La imagen muestra como se construye el grafo, el cual está representado por una matriz.

Si el robot se mueve, la odometría u_2 agregará un arco entre los nodos X_1 y X_2, como se muestra en la figura 3.4. El grafo se construye al aplicar consecutivamente estos dos pasos básicos, el cual como se puede apreciar, va aumentado de tamaño el grafo. En la mayoría de los casos, cada nodo sólo está conectado a un pequeño número de otros nodos. El número de restricciones en el grafo en el peor de los casos es lineal en el tiempo transcurrido y en el número de nodos en el grafo. De esta forma el enfoque de SLAM basado en grafo construye un problema de estimación simplificado mediante la abstracción de las mediciones de los sensor. Estas mediciones se sustituyen por las aristas en el grafo las cuales pueden ser vistas como "mediciones virtuales". Las aristas entre dos nodos se asignan con una distribución de probabilidad sobre las ubicaciones relativas de

las dos poses, las cuales están condicionadas por las mediciones mutuas. En general, el modelo de observación $p(z_t|x_t, m_t)$ es multimodal y por lo tanto la suposición gaussiana no se sostiene. Esto significa que una sola observación z_t podría resultar en múltiples aristas potenciales que pueden conectar diferentes nodos en el grafo y por lo tanto la conectividad en el grafo necesita ser descrita como una distribución de probabilidad. Tratar directamente con la multimodalidad supondría una explosión combinatoria de la complejidad. Por lo tanto, se debe determinar las restricciones más probables que resultan de las observaciones. Esta decisión depende de la distribución de probabilidad sobre las poses del robot. Este problema como se mencionó se conoce como problema de asociación de datos, usualmente tratado por el "front-end". Para calcular la asociación de datos correcta, el "front-end" requiere realizar una estimación consistente del condicional previo sobre la trayectoria del robot $p(x_{1:T}|z_{1:T}, u_{1:T})$. Para ello es necesario intercalar la ejecución del "front-end" y "back-end" mientras el robot explora el entorno. Por lo tanto, la precisión y la eficiencia del "back-end" es crucial para el diseño de un buen sistema de SLAM.

Si las observaciones se ven afectadas por ruido gaussiano (a nivel local) y la asociación de datos se conoce, el objetivo de un algoritmo de mapeo basado en el grafo es calcular una aproximación gaussiana a *posteriori* sobre la trayectoria del robot. Esto implica calcular la media de esta gaussiana dada la configuración de los nodos que maximiza la probabilidad de las observaciones. Una vez que se conoce esta media, la matriz de información de la gaussiana se puede obtener de una manera sencilla. En lo que sigue vamos a caracterizar la tarea de encontrar este máximo como un problema de optimización de restricciones. Un ejemplo de lo mencionado se ilustra en la figura 3.5.

Sea $x = (x_1, ..., X_T)^T$ un vector de parámetros, donde x_i describe la pose del nodo i. Sea z_{ij} y $\Omega_i j$ la media y la matriz de información de una medición virtual entre el nodo i y el nodo j. Esta medición virtual es una transformación que hace que las observaciones adquiridas en i tengan máxima superposición con las observaciones adquiridas en j. Sea $\hat{z}_{ij}(x_i, x_j)$ la predicción de una medición virtual dada una configuración de nodos x_i y x_j. Por lo general, esta predicción es la transformación relativa entre los dos nodos. Por lo tanto, la verocimilitud l_{ij} de una medición z_{ij} es:

$$l_{ij} \propto [z_{ij} - \hat{z}_{ij}(x_i, x_j)]^T \Omega_{ij} [z_{ij} - \hat{z}_{ij}(x_i, x_j)]. \tag{3.11}$$

Sea $e(x_i, x_j, z_{ij})$ una función que calcula la diferencia entre la observación esperada $\hat{z}_{ij}$ y la observación real z_{ij} recogida por el robot. Para simplificar la notación, vamos a codificar los índices de la medición con los índices de la función de error.

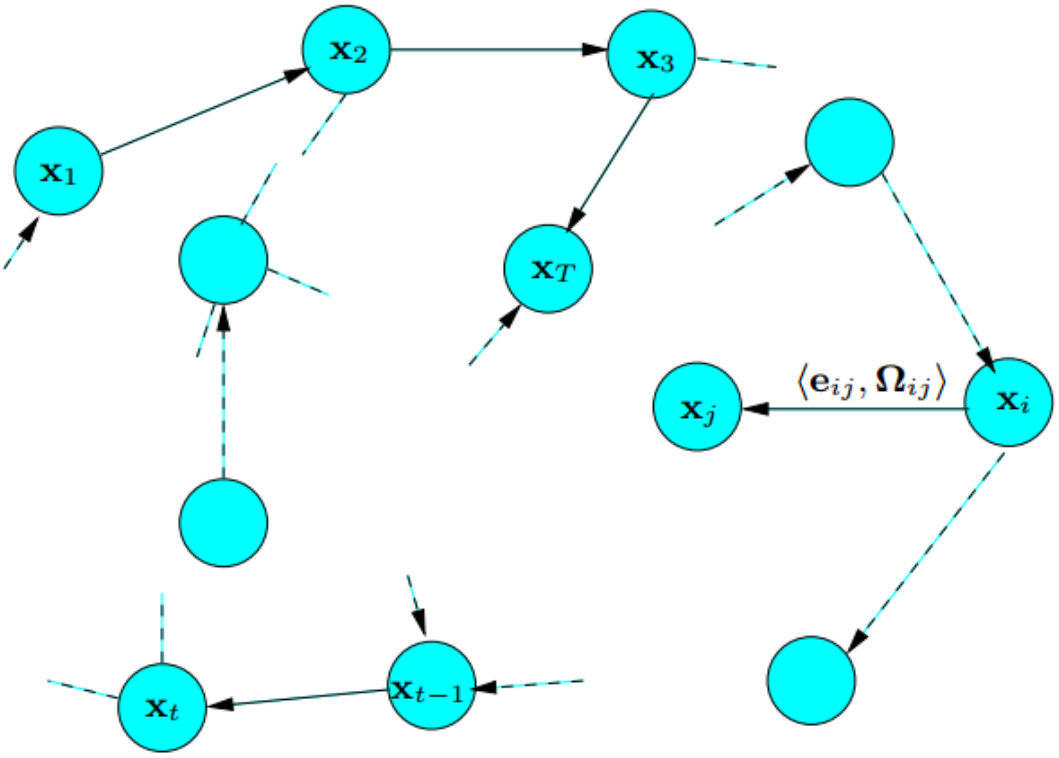


FIGURA 3.5: Cada nodo en el grafo corresponde a una pose del robot. Las poses cercanas están conectados por aristas que modelan restricciones espaciales entre las poses del robot. Las aristas $et-1$ t entre poses consecutivas modelan la odometría obtenida de las mediciones, mientras que las otras aristas representan restricciones espaciales que surgen de múltiples observaciones de la misma parte del medio ambiente.

$$e(x_i, x_j) = z_{ij} - \hat{z}_{ij}(x_i, x_j) \tag{3.12}$$

La figura 3.6 ilustra las funciones y las cantidades que juegan un papel en la definición de una arista en el grafo. Sea C el conjunto de pares de índices que indican que existe una restricción z. El objetivo de un enfoque de máxima verosimilitud es encontrar la configuración de los nodos x^* que minimiza el logaritmo negativo de verocimilitud $F(x)$ para todas las observaciones.

$$F(x) = \sum_{\langle i,j \rangle \in C} e_{ij}^T \Omega_{ij} e_{ij}, \tag{3.13}$$

Por lo tanto, se trata de resolver la siguiente ecuación:

$$x^* = argmin_x F(x). \tag{3.14}$$

Varias técnicas eficientes de optimización se pueden aplicar. Las opciones más comunes incluyen descenso de gradiente, gradiente conjugado, entre otros. La mayoría de las implementaciones de SLAM se basan en algún tipo de linealización iterativa de las funciones.

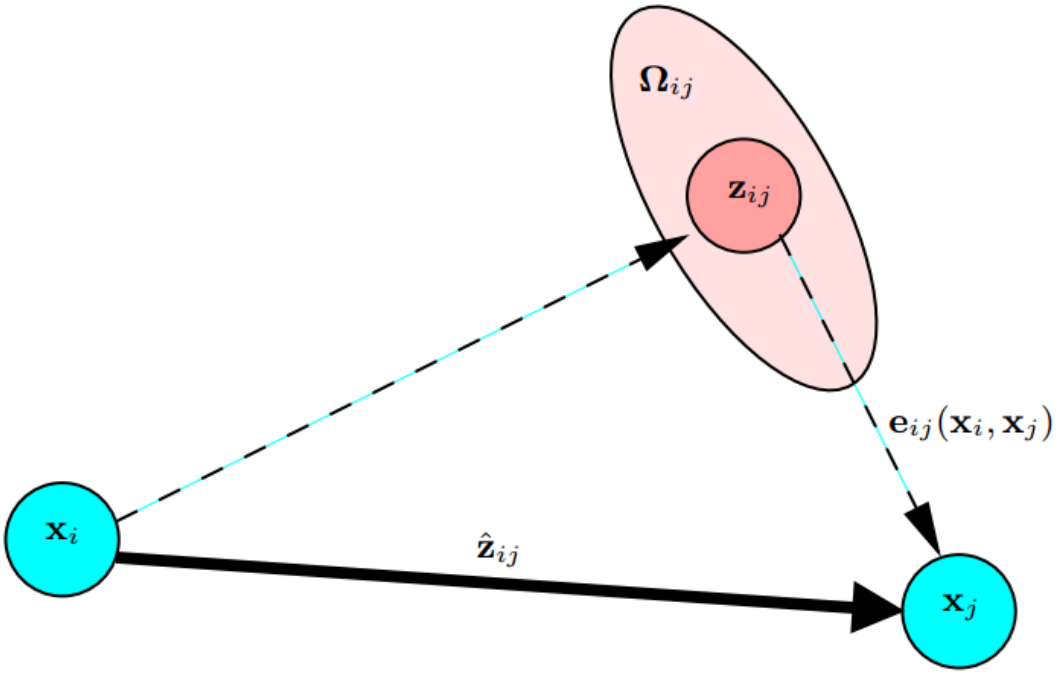


FIGURA 3.6: Aspectos de una arista que conecta el vértice x_i y el vértice x_j. Esta arista se origina de la medición z_{ij}. Desde la posición relativa de los dos nodos, es posible calcular la medición esperada z_{ij} que representa x_j visto desde el marco de x_i. El error $e_{ij}(x_i, x_j)$ depende de la diferencia entre lo esperado y la medición real. Una arista está totalmente caracterizada por su función de error $e_{ij}(x_i, x_j)$ y por la matriz de información Ω_{ij} de la medición que da cuenta de su incertidumbre.

El SLAM basado en grafo tienen la ventaja de que escala mejor en mapas de más altas dimensiones en comparación que el enfoque EKF SLAM. El factor limitante clave en EKF SLAM es la matriz de covarianza, que tiene espacio (y tiempo de actualización) cuadrática según el mapa. No existe tal limitación en SLAM basado el grafo. El tiempo de actualización del grafo es constante y la cantidad de memoria requerida es lineal. Realizar la optimización puede ser costosa, sin embargo, la búsqueda de la asociación de datos óptimos se sospecha que es un problema NP-duro, aunque en la práctica el número de asignaciones plausibles suele ser pequeña.

3.4.3. Filtro de partículas

El tercer principal paradigma en SLAM se basa en el filtro de partículas el cual se remonta a [58], pero sólo en los últimos años es que han cobrado popularidad. El filtro de

partículas representa la probabilidad a *posteriori* a través de un conjunto de partículas. Cada partícula se puede pensar como una conjetura concreta en cuanto a cuál puede ser el verdadero valor del estado. Mediante la recopilación de muchas de esas conjeturas en un conjunto de conjeturas, o un conjunto de partículas, el filtro de partículas captan una muestra representativa de la distribución *posteriori*. Se ha demostrado que en condiciones controladas que se acerca a la verdadera *posteriori* cuando el tamaño del conjunto de partículas tiende a infinito. También es una representación no paramétrica que representa distribuciones multimodales con facilidad.

El problema clave con el filtro de partículas en el contexto de SLAM es que el espacio de mapas y trayectorias de robot es enorme. Supongamos que tenemos un mapa con 5,000 características. ¿Cuántas partículas se necesitarían para llenar ese espacio? De hecho, los filtros de partículas escalan exponencialmente con la dimensión del espacio de estado subyacente. Tres o cuatro dimensiones se consideran aceptables, pero 100 dimensiones generalmente no.

El truco para hacer los filtros de partículas aplicables al problema de SLAM se remonta a [59][60]. El truco se introdujo en la literatura de SLAM en [61], seguido de [62], que acuñó el nombre FastSLAM. Primero se va a explicar el algoritmo FastSLAM básico y se mostrará la justificación de este planteamiento.

En cualquier instante en el tiempo, FastSLAM mantiene K partículas del tipo:

$$X_t^{[k]}, \mu_{t,1}^{[k]}, ..., \mu_{t,N}^{[k]} \sum_{t,1}^{[k]}, ..., \sum_{t,N}^{[k]} \tag{3.15}$$

Aquí $[k]$ es el índice de la muestra. Esta expresión indica que una partícula contiene:

- Un trayecto de la muestra $X_t^{[k]}$.
- Un conjunto N dimensional de gaussianas con media $\mu_{t,N}^{[k]}$ y varianzas $\sum_{t,N}^{[k]}$, una para cada punto de referencia en el medio ambiente.

Aquí $n(1 \leq n \leq N)$ es el índice del punto de referencia. De esto se deduce que las partículas K poseen K muestras de ruta. También se deduce que posee KN gaussianas, cada uno de las cuales modela exactamente un punto de referencia para una de las partículas.

Inicializar FastSLAM es sencillo: basta con configurar la ubicación de cada partícula del robot a sus coordenadas conocidas de inicio y poner a cero el mapa. La actualización de las partículas procede entonces de la siguiente manera:

- Cuando se recibe una lectura de odómetría, nuevas variables de localización se generan estocásticamente, una para cada una de las partículas. La distribución para generar esas partículas de localización se basa en el modelo de movimiento:

$$x_t^{[k]} \sim p(x_t | x_{t-1}^{[k]}, u^t) \tag{3.16}$$

  Aquí $x_{t-1}^{[k]}$ es la ubicación anterior, que es parte de la partícula. Esta etapa de muestreo probabilístico es fácil de implementar para cualquier robot cuya cinemática pueda ser calculada.

- Cuando se recibe una medición z_t, suceden dos cosas: primero, FastSLAM calcula para cada partícula la probabilidad de la medición z_t. Si el índice del punto de referencia detectado n. Entonces, la probabilidad deseada se define como:

$$w_t^{[k]} := N(x_t^{[k]}, \mu_{t,n}^{[k]}, \Sigma_{t,n}^{[k]}) \tag{3.17}$$

  El factor $w_t^{[k]}$ se llama el peso de importancia ya que mide la importancia de la partícula a la luz de la nueva medición del sensor. Al igual que antes, N indica la distribución normal, pero esta vez se calcula para un valor específico, z_t. Los pesos de importancia de todas las partículas se normalizaron para que sumen 1.

  A continuación, FastSLAM saca con reemplazo del conjunto de partículas existentes un conjunto de nuevas partículas. La probabilidad de sacar una partícula es su peso de importancia normalizada. Este paso se llama remuestreo. La intuición detrás del remuestreo es simple: las partículas cuya medición es más plausible tienen una mayor probabilidad de sobrevivir al proceso de remuestreo. Finalmente, FastSLAM actualiza la media $\mu_{t,n}^{[k]}$ y la covarianza $\Sigma_{t,n}^{[k]}$) del nuevo conjunto de partículas basado en el medición z_t. Esta actualización sigue las reglas de actualización estándar de EKF.

El algoritmo FastSLAM tiene una serie de propiedades notables, la primera es que resuelve tanto el SLAM completo y el problema de SLAM en línea. Cada partícula tiene una muestra de la trayectoria completa, esto se ilustra en la figura 3.7, pero la ecuación de actualización real sólo utiliza la posición más reciente. Esto hace FastSLAM un filtro, similar a EKF. La segunda propiedad, FastSLAM hace que sea fácil seguir múltiples hipótesis de asociación de datos. Es fácil tomar decisiones de asociación de datos en base a cada partícula, en lugar de tener que adoptar la misma hipótesis para todo el filtro. Y tercera, FastSLAM se puede implementar de manera muy eficiente: utilizando métodos avanzados de árbol para representar el mapa de estimaciones, la actualización se puede realizar en tiempo logarítmico según el tamaño del mapa N y lineal en el número de

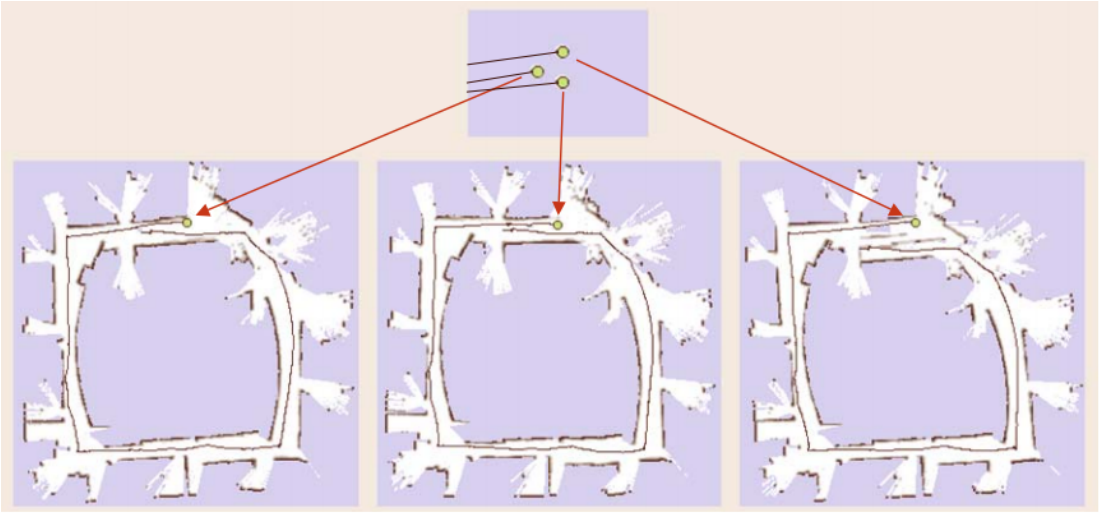


FIGURA 3.7: Cada partícula lleva su propio mapa y los pesos de importancia de las partículas se calculan en función de la probabilidad de las medidas dadas del propio mapa de la partícula.

partículas M. Estas propiedades, junto con la relativa facilidad de aplicación, han hecho de FastSLAM una opción popular.

3.5. Comparación de los tres paradigmas principales

Los tres paradigmas presentados cubren la gran mayoría de los trabajos en el campo de SLAM. Como se mencionó, EKF SLAM viene con un obstáculo computacional que plantea limitaciones de escala graves. Las extensiones más prometedores de EKF SLAM se basan en la construcción de sub-mapas locales; sin embargo, en muchos aspectos, los algoritmos resultantes se asemejan al enfoque basado en el grafo.

Los métodos que están basados en grafo abordan el problema de SLAM completo y por lo tanto, son por naturaleza, fuera de línea. Se basan en la observación de que SLAM puede ser modelado por un grafo de restricciones blandas, donde cada restricción o bien corresponde a un movimiento o un evento de medición. Debido a la disponibilidad de métodos de optimización altamente eficientes para problemas de optimización no lineales. El SLAM basado en grafos se ha convertido en el método de elección para la construcción de mapas fuera de línea a gran escala. La búsqueda de asociación de datos se incorpora muy fácilmente en el marco matemático básico y existe un gran número de técnicas de búsqueda para encontrar correspondencias adecuadas.

Los métodos de filtro de partículas evitan algunas de los problemas derivados de las correlaciones naturales en el mapa, que plagan a EKF. Mediante el muestreo de las poses del robot, los puntos de referencia individuales en el mapa se independizan y por

lo tanto están correlacionadas. La representación de partícula de FastSLAM tiene una serie de ventajas. Computacionalmente, FastSLAM se puede utilizar como un filtro y su actualización requiere tiempo lineal-logarítmico donde EKF necesitaba tiempo cuadrático. Además, FastSLAM puede distinguir de la asociación de datos, lo que hace que sea un método principal para SLAM en la asociación de datos desconocidos. En el lado negativo, el número de partículas necesarias puede crecer rápidamente a tamaños muy grandes.

3.6. Algoritmo general de SLAM

Un algoritmo de SLAM consiste esencialmente en los siguientes pasos:

- Adquisición de datos: en este paso se hacen las mediciones mediante los sensores, por ejemplo, cámaras, sonares, láser, etc. y se guardan para su posterior uso.
- Extracción de características: se extrae de los datos una serie de características únicas, que deben ser fácilmente reconocibles.
- Asociación de características: las características encontradas en las mediciones anteriores se asocian con características de la medición más reciente.
- Estimación de la pose: el cambio relativo entre los puntos de características y la posición del vehículo se utiliza para estimar la nueva posición del vehículo.
- Ajuste del Mapa: el mapa se actualizará de acuerdo con la nueva pose calculada.

Estos pasos se repiten continuamente y una trayectoria de estimaciones de posición y un mapa se construyen. La figura 3.8 ilustra el proceso. En la figura 3.8 un robot ha recibido datos de entrada y se extraen características de los datos. En el caso de SLAM visual, una características puede ser cualquier cosa que es fácilmente reconocible por un sensor visual, por ejemplo, una esquina, un borde, un punto en un color que sobresale.

El robot de la figura 3.8 extrajo tres características y se almacenan en una base de datos que contiene todas las características que se han observado hasta ahora. En la figura 3.8 el robot se ha movido hacia adelante y recibió nuevos datos. El robot extrae nuevas características de los datos y busca a través de su base de datos para ver si hay coincidencias con mediciones antiguas. Se añaden características extraídas que no se encuentran en la base de datos y las características que den una coincidencia en la base de datos se utilizan para estimar el cambio de posición del robot. Esto se hace midiendo el cambio en la distancia y el ángulo de las antiguas características con respecto a las

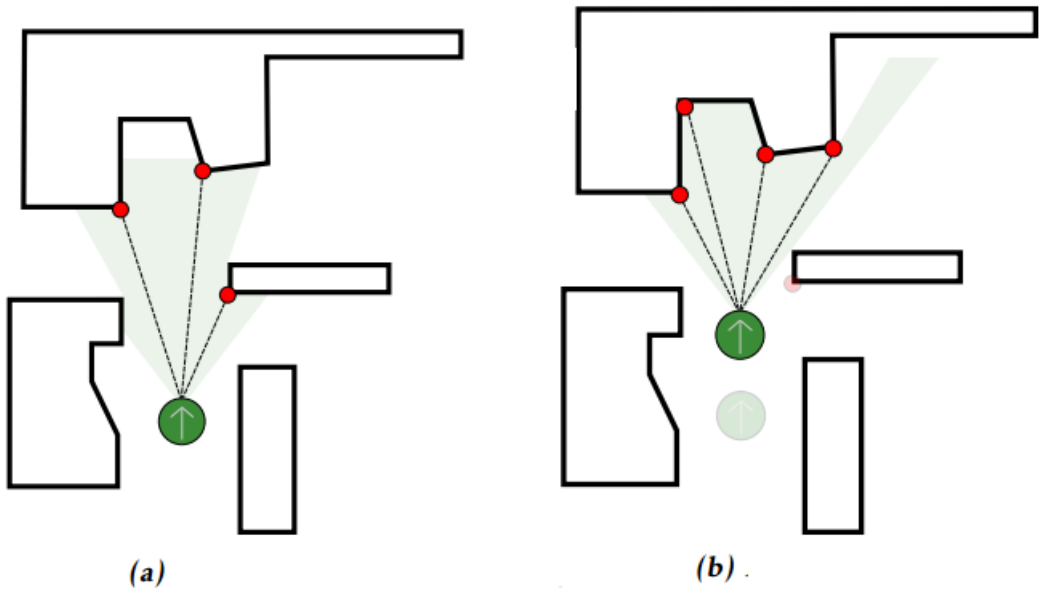


FIGURA 3.8: a) El robot detecta 3 puntos de referencia. b) El robot avanza y detecta dos de los puntos de referencia que detectó en la medición anterior y dos puntos nuevos de referencia más.

nuevas. Cuando se estima que la nueva posición del robot se utiliza esta estimación y las medidas para ajustar las posiciones de los características.

3.7. Algoritmo RBG-D SLAM

Este proyecto de tesis se basa en un algoritmo llamado RGB-D SLAM con el enfoque de grafo, el cual es un algoritmo de seis grados de libertad (x y, z, balance, cabeceo y guiñada) que realiza SLAM visual. El algoritmo utiliza tanto el color (RGB) y la información de profundidad (D), el cual es un algoritmo genérico, debido al hecho de que no contiene ningún modelo de movimiento. El algoritmo ha sido desarrollado por [70]. A continuación se presentan los pasos principales del algoritmo:

- Como un primer paso la información de profundidad y la imagen RGB se recogen con la estampa de tiempo para cada una, luego se extraen características de la imagen RGB mediante un algoritmo de extracción de características. En RGB-D SLAM se pueden usar múltiples algoritmos de extracción de características. Cada algoritmo tiene sus pros y contras en diferentes ambientes y difieren en el tiempo de cálculo y recursos [71]. El algoritmo implementado en esta tesis es SURF y se describe en el capítulo 5.

- En el siguiente paso del algoritmo, las características extraídas se proyectan a la imagen de profundidad. Este paso introduce cierta incertidumbre. Principalmente debido a la falta de coincidencia entre la profundidad y las imágenes RGB.
- Para encontrar la transformación 6D de la posición de la cámara, o asociación de información, teniendo en cuenta el ruido, se utiliza el algoritmo RANSAC y luego se refina mediante el algoritmo ICP. Las características nuevas se corresponden con las características extraídas anteriormente. En este trabajo se compara contra la última medición hecha. Tres pares de características coincidentes son seleccionados al azar y se utilizan para calcular la transformación 6D, luego todos los pares de características se evalúan por su distancia euclidiana entre sí, pares cuya distancia euclidiana está por debajo de un cierto umbral se consideran datos atípicos. Con los datos típicos se calcula una la transformación y se refina usando el algoritmo ICP.

La figura 3.9 muestra un esquema de la implementacion hecha en este trabajo de tesis de SLAM basado en grafo.

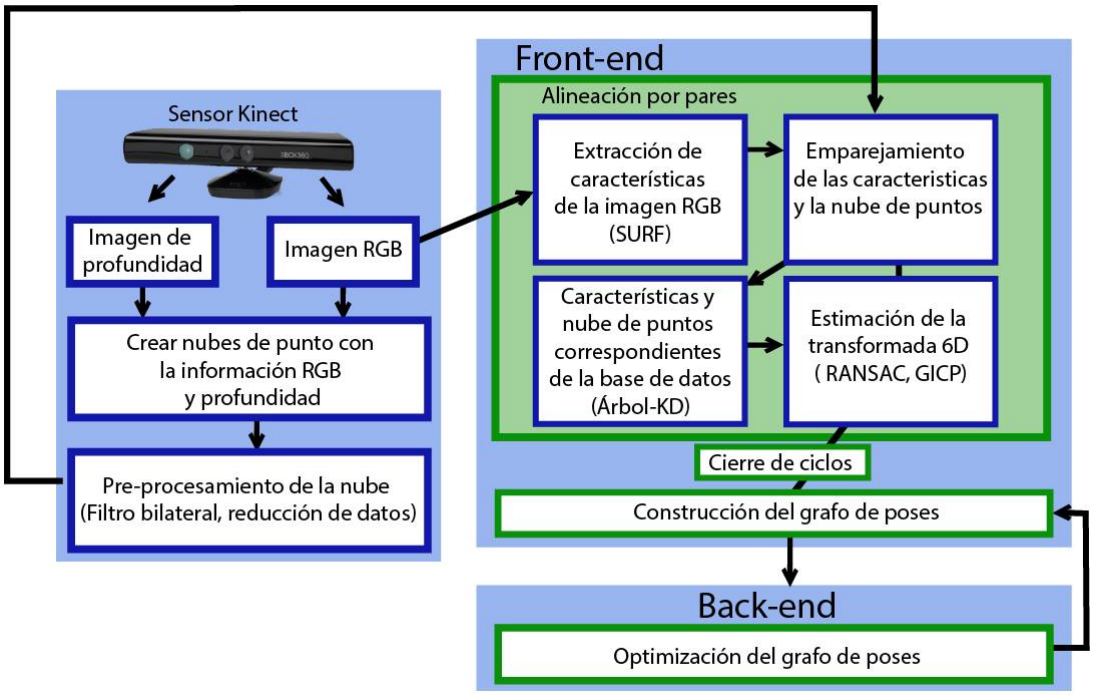


FIGURA 3.9: Esquema del algoritmo SLAM implementado en este trabajo.

3.8. Pre-procesamiento de la nube de puntos

En esta sección, se presentan los filtros usados para reducir el ruido en los datos medidos con del sensor Kinect. Antes de proceder a usar los datos de Kinect, se utilizan filtros

con el fin de aliviar los diversos tipos de ruido que este sensor produce (véase la figura 3.10). Hay varias formas en las que se puede eliminar cierto tipo de ruido en nubes de puntos [72][73]. Sin embargo, ninguno tiene en cuenta los difíciles efectos cuantitativos vistos en este tipo de datos.

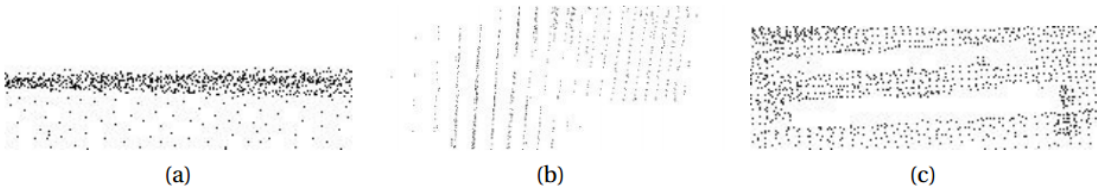


FIGURA 3.10: Varios tipos de ruido que las cámaras RGB-D producen: (a) el ruido en una superficie plana; (b) efectos de cuantificación; (c) pixeles que faltan.

Existen tres métodos principales de eliminación de ruido :

1. ***El filtro de mediana:*** presentado en [74], es uno de los filtros más simples y extendidos de procesamiento de imágenes. Es conocido por tener un buen desempeño con ruido de impulso (algunos pixeles individuales tienen valores extremos), no reduce el contraste (en comparación con los filtros basados en promedio) y es robusto a los valores atípicos. Además, es fácil de implementar y eficiente ya que requiere una sola pasada sobre la imagen. Se compone de una ventana móvil de tamaño fijo que reemplaza el pixel en el centro con la mediana dentro de la ventana.

2. ***El filtro bilateral*** es un filtro no lineal que suaviza una señal y que preserva bordes fuertes y ha sido utilizado con éxito en el campo de la visión por computadora, por bastante tiempo. La ecuación 3.18 muestra como se actualiza el pixel central de una ventana móvil de tamaño fijo. p es el pixel actual, q es un pixel en el Kernel N alrededor de p y I_p es la intensidad del pixel. Las funciones f y g miden distancias geométricas gaussianas ponderadas y la similitud fotométrica. El filtro bilateral tiende a suavizar más cuando los pixeles vecinos son similares (áreas uniformes y suavizar menos cuando hay saltos grandes, evitar suavizar los bordes). Matemáticamente, la ecuación 3.19 es similar, pero los operadores de suavizado trabajan en el dominio de la imagen a color: S es la imagen de profundidad, I es la imagen RGB y f y g son funciones gaussianas centradas en 0 y con desviaciones estándar σ_{color} y $\sigma_{profundidad}$.

$$\widetilde{I_p} = \frac{\sum_{s \in N} f(p-s) g(I_p - I_s) I_s}{\sum_{s \in N} f(p-s) g(I_p - I_s)} \tag{3.18}$$

$$\widetilde{S_p} = \frac{\sum_{s \in N} S_q f(||p-s||) g(||I_p - I_s||)}{\sum_{s \in N} f(||p-s||) g(||I_p - I_s||)} \tag{3.19}$$

En [75] se propone una una nueva forma de aproximar el filtro bilateral con el fin de hacer implementaciones rápidas. Esto se hace mediante la interpretación de la imagen como una función 3D, donde todo el filtro se convierte en una convolución en 3D. Los autores demuestran que este enfoque produce resultados muy similares a la del filtro bilateral original, pero con tiempos de ejecución mucho más rápidos. En este trabajo de tesis se aplica esta filtro con la implemetación realizada de la librería PCL.

3. ***Mínimos cuadrados móviles:*** para cada punto p de la nube, un plano local se estima mediante los vecinos y el análisis de los componentes principales (PCA) y una función 2D $g(xy)$ (generalmente de grado 2 ó 3), como en la ecuación 3.20, donde q es el origen del fotograma de referencia (es decir, el origen del plano local), n es la normal del plano y p_i es un punto en la vecindad de q. El punto inicial es entonces proyectado a esta función y enviado a la nube resultado [76].

$$\sum_{i=1}^{N}[g(x_i, y_i) - n \circ p(p_i - q)]^2 \Theta(||p_i - q||) \tag{3.20}$$

El resultado de aplicar los filtros a conjuntos de datos ruidosos se muestran en la figura 3.11.

3.9. Búsqueda de vecinos más cercanos

Varias partes de este trabajo requieren el cálculo de los vecinos más cercanos. Dado un conjunto de puntos N, $P = (p_1, p_2, ...p_N)$ y un punto q, el problema consiste en encontrar el punto P que está más cerca de q. Esto tiene usos prácticos en diversas aplicaciones y por lo tanto muchos métodos diferentes han sido desarrollados.

Una idea básica es calcular las distancias a todos los puntos en P y elegir el que esté a la distancia más corta. Esto se conoce como la búsqueda de fuerza bruta, búsqueda exhaustiva o el enfoque ingenuo. El método es simple, pero solamente escala linealmente con el número N de puntos en P. Expresado en notación O, sería $O(N)$.

Si se realizan muchas búsquedas de vecinos más cercanos dentro de P, es aconsejable representar P con una estructura que permita realizar búsquedas más eficientes.

Un idea consiste en dividir los puntos en P en grupos de acuerdo a su ubicación. La búsqueda del vecino más cercano se acelera mediante la exclusión condicional de muchos puntos y reduciendo así el número de cálculos de la distancia. Uno de estos métodos es el árbol *kd* [77], que se detalla en la siguiente sección.

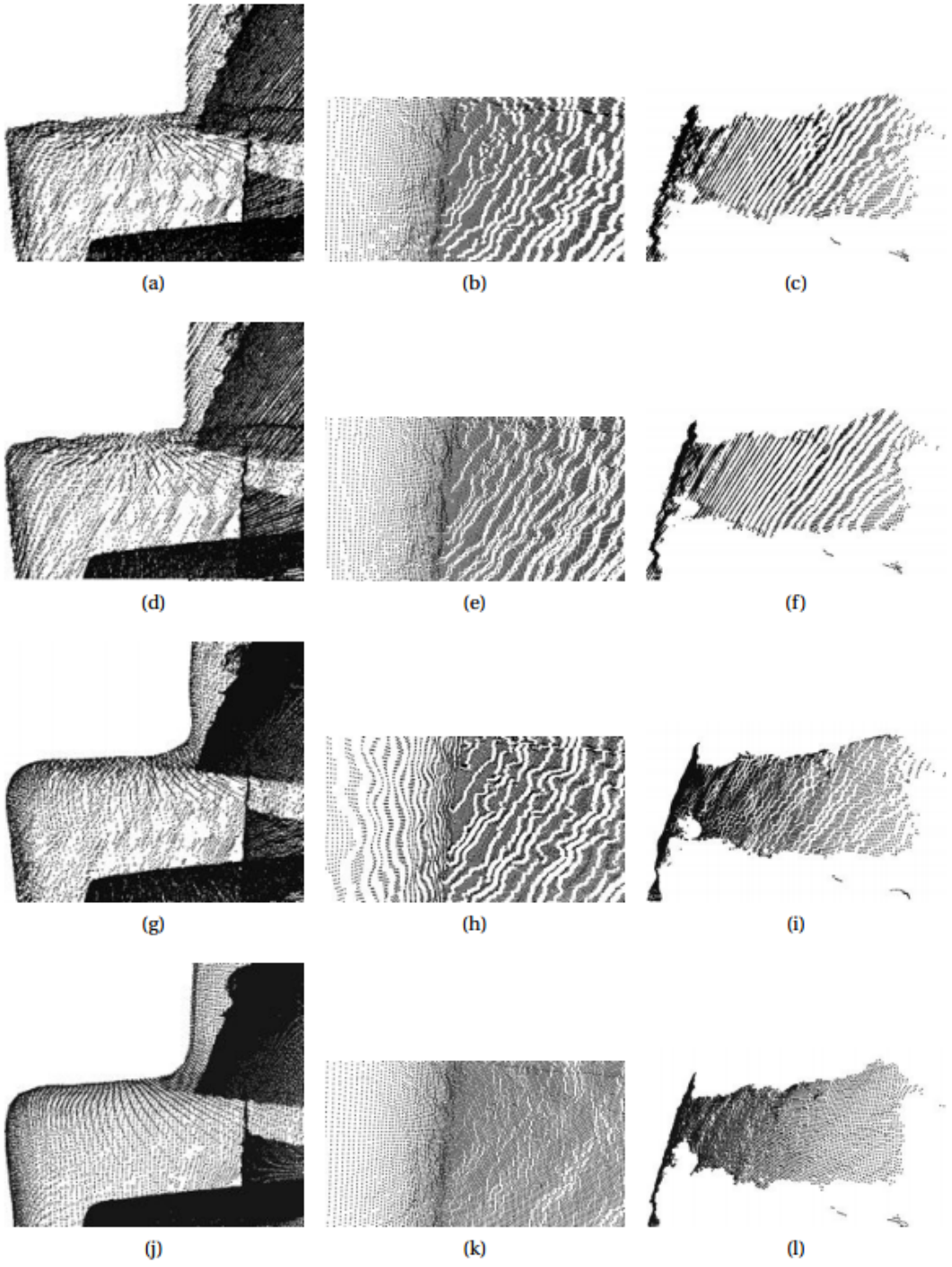


FIGURA 3.11: (a, b, c) nube de entrada sin filtrar; (d, e, f) filtro de mediana; (g, h, i) filtro MLS; (j, k, l) filtro bilateral

3.9.1. Árboles *kd*

Con el fin de acomodar los puntos en un árbol *kd*, el conjunto P se divide al encontrar la mediana de las coordenadas de todos los puntos. El punto que corresponde a la mediana se convierte en la raíz del árbol. Luego, los dos subconjuntos resultantes se dividen en base a la mediana de su segunda coordenada. El proceso es conocido como "partición binaria del espacio" y genera un árbol binario balanceado que contiene todos los puntos de P.

La figura 3.12 ilustra la creación de un árbol *kd* para puntos en dos dimensiones. Aunque es difícil de ilustrar el proceso, este puede ser fácilmente extendido para espacios de dimensiones superiores. En R^3, que es de interés en este trabajo, las nubes de puntos pueden ser divididas posteriormente en dos conjuntos de igual tamaño por planos que son ortogonales al eje X, Y y Z.

La complejidad computacional de la creación de un árbol *kd* no es mayor que $O(knlog^2N)$ [78]. Esto corresponde al ordenamiento del punto con respecto a cada uno de sus coordenadas, o sea, k veces una operación $O(nlog^2N)$ de ordenamiento.

Mediante el uso de árboles *kd* el tiempo de cálculo para la búsqueda de vecinos más cercanos en métrica euclidiana se puede reducir en gran medida. Supongamos que un árbol *kd* ha sido construido y contiene los puntos de conjunto P. Localizando el vecino más cercano a q dentro de P se puede hacer de la siguiente manera algorítmica.

1. Moverse hacia abajo en el árbol a partir de la raíz comparando las coordenadas de acuerdo con la dimensión real de división hasta que se alcanza un nodo hoja.
2. Marque el punto actual como $mejoractual$ y calcular la distancia d entre q y $mejoractual$.
3. Moverse un nivel hacia arriba en el árbol *kd* y determinar la distancia desde q para este nodo. Si la distancia es más corta que d, definir este nodo como $mejoractual$ y la distancia a esta como d.

   Si la distancia desde q hacia el plano que separa los nodos actuales es mayor que d, excluir todos los nodos del otro lado de la rama y se continúa moviéndose hacia arriba hasta llegar a la raíz.

   De lo contrario, los nodos del otro lado de la rama se exploraran de la misma manera que todo el árbol.
4. Cuando se alcanza el nodo superior y todas las ramas laterales se han buscado, se escoge el candidato más corto de la búsqueda principal y eventuales sub-búsquedas de las ramas.

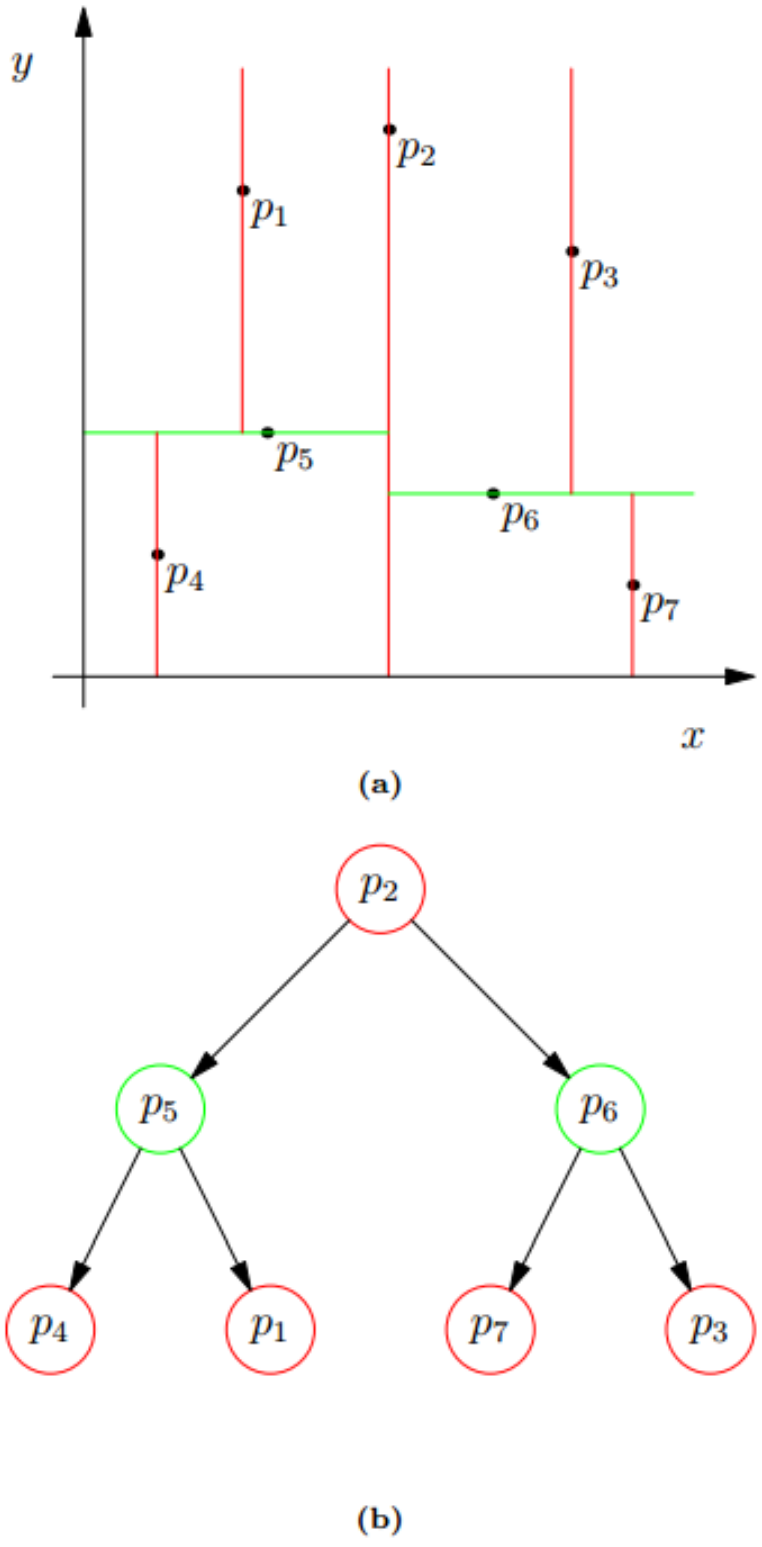


FIGURA 3.12: (a) Una nube de puntos en 2 dimensiones, la cual ha sido dividida mediante partición binaria del espacio. (b): El árbol *kd* correspondiente.

Dependiendo de la dispersión de P, piezas muy grandes o muy pequeñas de P pueden ser excluidas de la búsqueda del vecino más cercano. La razón es que si la distancia al primer nodo hoja encontrada es más corta que la distancia del plano divisor de su padre, entonces estamos en el nodo correcto y todas las demás ramas pueden ser excluidas. En este caso, sólo un cálculo de la distancia se lleva acabo y la complejidad es $O(1)$. La complejidad esperada del algoritmo de búsqueda es $O(log^2 N)$ [78]. En el peor de los casos, ni uno solo punto pudo ser excluido de la búsqueda y el algoritmo se comportará como el enfoque ingenuo, que es $O(N)$.

Existen dos modificaciones importantes al algoritmo de árbol *kd*:

- Al realizar las búsquedas de vecino k más cercano, el algoritmo de localización puede tener en cuenta no sólo el vecino más cercano, si no también un número determinado de vecinos más cercanos. En [78] demuestran que esta modificación incrementa la complejidad del algoritmo.
- Una búsqueda del vecino más cercano aproximada pretende encontrar buenos candidatos para los vecinos más cercanos mientras se mantiene una baja complejidad. En árboles *kd*, esto podría ser implementado como una simplificación del algoritmo de localización. Se atravesará el árbol y volverá al punto en el interior del nodo hoja como el vecino más cercano aproximado. Obviamente, esto tiene una complejidad de $O(1)$.

Capítulo 4

Recursos de hardware y software utilizados

Este capítulo comienza con una introducción al hardware utilizado para implementar este trabajo y continúa con una breve presentación del software y las bibliotecas de código abierto utilizadas.

4.1. Recursos de hardware

4.1.1. Kinect

Kinect para Xbox 360 (antes conocido como Project Natal) es un dispositivo sensor de entrada de movimiento, el cual se basa en el tecnología de software desarrollado por Rare y con base en la cámara de RGB-D, tecnología desarrollada por PrimeSense. La cámara Kinect es un sensor de bajo costo construido especialmente para la consola Xbox. el cual puede entregar una imagen RGB y una imagen de profundidad en paralelo.

El sensor Kinect ha sido objeto de estudio recientemente [79]. Por esa razón y por ser el sensor elegido para la solución desarrollada, en esta sección se exponen las principales características del sensor Kinect y se da una breve descripción de su funcionamiento.

4.1.1.1. Características del sensor

El sensor Kinect está provisto de una cámara RGB, un sensor de rango, un arreglo de micrófonos, una base motorizada que permite un movimiento de cabeceo y acelerómetros

que proveen información sobre la orientación del sensor. El conjunto de sensores que conforman el dispositivo Kinect se muestra en la figura 4.1.

FIGURA 4.1: Sensor Kinect

La cámara RGB tiene un flujo de datos de 8-bits con resolución VGA (640 *times* 480 píxeles) a una frecuencia de 30Hz. Por otra parte, el sensor de rango está compuesto por un proyector láser infrarrojo y un sensor CMOS monocromo. El flujo de datos del sensor de rango es de 11-bits con resolución VGA (640 × 480 píxeles) a una frecuencia de 30Hz, aunque el campo de visión es ligeramente menor que el de la cámara RGB.

El conjunto formado por el proyector infrarrojo y la cámara infrarroja proporcionan una resolución espacial de 3 mm en altura y anchura y de 1 cm en profundidad. Si bien el rango de operación está entre los 0.5 y 3.5 metros aproximadamente.

4.1.1.2. Funcionamiento del sensor de rango

La tecnología empleada en el sensor Kinect para obtener la información de profundidad está basada en la tecnología de la empresa PrimeSense conocida como "Codificación por luz". En esta sección se describe en qué consiste esta tecnología

Para conseguir las mediciones de profundidad, la cámara de rango está formada por dos componentes: un proyector de luz infrarroja y un sensor de imagen CMOS monocromo estándar como se ha explicado en la sub-sección anterior. El proyector infrarrojo emite un patrón de puntos no correlacionados sobre la escena como se aprecia en la figura 4.2. Esto quiere decir que las posiciones de los puntos proyectados no están correlacionadas al visualizarlas en un plano trasversal al eje normal al proyector.

En la descripción de la patente para la obtención de la imagen de rango usando puntos proyectados no correlacionados [80] de PrimeSense se detalla un proceso en cuatro etapas:

1. Capturar imágenes de referencia con el patrón de puntos no correlacionados a diferentes distancias.

FIGURA 4.2: Imagen IR proporcionada por el sensor Microsoft Kinect donde se aprecia el patrón IR proyectado por el emisor láser.

2. Capturar una imagen de prueba con el patrón de puntos proyectado sobre la escena.
3. Hallar el rango emparejando la imagen de prueba con las imágenes de referencia.
4. Construir el mapa 3D basándose en las compensaciones locales entre la imagen de prueba y las de referencia.

Para realizar las mediciones de profundidad de un objeto en la escena, primero se toman una serie de imágenes de referencia proyectando el patrón de puntos sobre una superficie plana a diferentes distancias conocidas en un proceso fuera de línea. Más adelante, durante el proceso en línea, el microprocesador presente en Kinect realiza una comparación de los grupos de puntos de la imagen infrarroja con cada una de las imágenes de referencia mediante correlación cruzada. Tras esta etapa, se obtiene la imagen de referencia que más se parece a la de la imagen de prueba. A continuación se calcula la compensación local de cada uno de los puntos de la imagen de prueba con los de la imagen de referencia correspondiente y mediante triangulación se obtiene el valor de profundidad para cada píxel de la imagen de rango. Finalmente, se traduce el valor de disparidad obtenido a distancia en metros.

4.1.2. Computadora de escritorio

Se usó una computadora Dell. La computadora cuenta con un procesador Intel(R) Xeon(R) CPU E5-2620 2.00GHz, 16 GB de memoria RAM y una tarjetea gráfica ATI FirePro 3800 de 512 Gb de RAM, Sistema Operativo Windows 8 de 64 bits. Se implementó todo el proyecto en esta plataforma, entre sus tareas están la de visualización de los resultados, el cómputo de todos los algoritmos implementados para llevar acabo este proyecto, así como almacenar todas las mediciones capturadas por el sensor Kinect.

4.2. Recursos de software

Este proyecto ha sido implementado gracias a la existencia de librerías de código abierto específicas para cada uno de los elementos que lo componen. Los tres bloques software sobre los que se desarrolla el proyecto son los siguientes:

- Librerías de captura de datos con el sensor Kinect.
- Librerías de procesamiento de imágenes 2D y de nubes de puntos 3D.
- Librerías de optimización de grafos para SLAM.

4.2.1. Descripción de las librerías utilizadas

A continuación se describen las características más relevantes de las librerías de software utilizadas en el desarrollo del proyecto, destacando aquellas que conforman los tres bloques descritos.

MRPT (Mobile Robot Programming Toolkit) es una biblioteca multiplataforma de código abierto y escrita en C ++ orientada a ayudar a los investigadores de robótica a diseñar e implementar algoritmos relacionados con la localización y mapeo simultáneo (SLAM), visión por computadora y la planificación de movimiento (evasión de obstáculos). Diferentes grupos de investigación han empleado MRPT para implementar proyectos reportados en algunas de las principales revistas de robótica y conferencias.

Algunos de las características incluidas:

- La visualización y manipulación de grandes conjuntos de datos.
- Algoritmos de SLAM: ICP, filtro extendido Kalman, filtro de partículas. Blackwellized-Rao y SLAM basado en grafo.

- Obtener conjuntos de datos de los sensores.

OpenCV (Open Source Computer Vision Library) es una biblioteca de visión por computadora de código abierto y software de aprendizaje automático. OpenCV fue construido para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD, OpenCV hace que sea fácil utilizar y modificar el código.

La biblioteca cuenta con más de 2,500 algoritmos optimizados, que incluye un amplio conjunto de algoritmos de visión por computadora, desde algoritmos clásicos hasta de última generación y de aprendizaje automático. Estos algoritmos se pueden utilizar para detectar y reconocer rostros, identificar objetos, clasificar las acciones humanas en videos, movimientos de cámara, extraer modelos 3D de objetos, producir nubes de puntos 3D de cámaras estéreo, encontrar imágenes similares en una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer el paisaje y establecer marcadores de realidad aumentada, etc.

PCL (Point Cloud Library) es una biblioteca de código abierto de algoritmos para tareas de procesamiento de nubes de puntos y el procesamiento de la geometría 3D, como ocurre en la visión por computadora en tres dimensiones. La biblioteca contiene algoritmos para la estimación de características, reconstrucción de la superficie, registro 3D, ajuste de modelos y segmentación. Está escrita en C ++ y liberadada bajo la licencia BSD.

Eigen es una biblioteca de C ++ de alto nivel de plantillas de encabezado para álgebra lineal, operaciones con matrices y vectoriales, solucionadores numéricos y algoritmos relacionados. Eigen es una biblioteca de código abierto con licencia en virtud de MPL2 a partir de la versión 3.1.1. Las versiones anteriores tenían licencia LGPL3 +.

Eigen es a menudo conocido por su elegante API, capacidades de matrices versátiles, fijas, dinámicas y una gama de solucionadores densos y dispersos. Para lograr un alto rendimiento, Eigen utiliza vectorización explícita para la ESS 2/3/4, ARM NEON y conjuntos de instrucciones AltiVec.

OpenNI (Open Natural Interaction) es una organización sin fines de lucro dirigida por la industria y de software de código abierto centrado en la certificación y la mejora de la interoperabilidad de las interfaces de usuario naturales e interfaces de usuario orgánicas para la interacción natural. De este proyecto se utilizan sus drivers para poder operar el sensor Kinect.

PrimeSense, que fue miembro fundador de OpenNI, pero cerró el proyecto OpenNI original cuando fue adquirida por Apple el 24 de noviembre de 2013. Desde entonces exsocios

de PrimeSense están manteniendo la versión OpenNI 2 activa como un software de código abierto.

4.2.2. Librerías de captura de datos

La implementación de este proyecto integra un interface de captura de datos. PCL ofrece acceso a los datos del sensor Kinect a través de la librería OpenNI. Este proyecto también integra la posibilidad de leer datos de un archivo en lugar de hacerlo directamente del sensor Kinect. En este caso se utiliza la interfaz de la librería MRPT para leer ficheros ".rawlog" que contienen las imágenes RGB y de rango.

4.2.3. Librerías de procesamiento de imágenes 2D y nubes de puntos 3D

La mayor parte de la funcionalidad de la aplicación se asienta sobre las librerías de procesamiento de imágenes 2D y nubes de puntos 3D. En este nivel se encuentran los algoritmos de detección y emparejamiento de características 2D, estimación y refinamiento de la transformación rígida y filtrado de nubes de puntos, etc. Para las tareas de procesamiento de imágenes 2D, se usa la librería OpenCV que es una de las alternativas más extendidas y con mayor soporte para la visión por computadora hoy día.

Se utiliza la librería PCL para procesamiento de nubes de puntos y para realizar las operaciones sobre las nubes de puntos 3D. Esta librería representa la alternativa más ampliamente utilizada y con mayor soporte de este ámbito.

4.2.4. Librerías de optimización de grafos

La construcción de mapas consistentes en una aplicación de SLAM requiere de estructuras de datos y algoritmos para la construcción y optimización del grafo de poses y restricciones espaciales. Para este apartado se utilizó el módulo de SLAM basado en grafo de la librería MRPT.

Capítulo 5

Construcción del mapa

En este capítulo se describe un método para estimar la transformación entre correspondencias de puntos 3D (RANSAC), el cual es robusto a datos atípicos, lo cual resulta de vital importancia por el ruido inherente en los sensores y por correspondencias incorrectas entre puntos 3D. Estimar la transformación rígida óptima entre dos conjuntos de puntos 3D es uno de los problemas más comunes a la hora de realizar reconstrucción de mapas 3D y representa uno de los puntos clave de la implementación realizada en este trabajo a la hora de obtener las relaciones espaciales entre los fotogramas RGB-D.

También se presentan dos algoritmos iterativos para refinar la estimación de la transformación rígida. El primero de estos algoritmos iterativos es ICP [81], un método ampliamente aplicado en el campo de la robótica móvil; el segundo de los algoritmos iterativos es GICP [82] que es una versión generalizada muy reciente de ICP que representa el estado del arte en tema de alineación de nubes de puntos.

5.1. Estimación de la transformación 3D

Dado un conjunto de correspondencias entre dos conjuntos de puntos en 3D, es posible encontrar una transformación rígida que una los dos conjuntos de puntos, es decir, la operación que proyecta cada punto de un fotograma origen hasta el punto correspondiente al fotograma destino, esto se ilustra en la figura 5.1. En nuestro caso, esta transformación es una proyección en perspectiva de 6 grados de libertad, compuesto por una matriz de rotación y un vector de traslación en 3 dimensiones. Esta transformación se puede escribir mediante el uso de una matriz de 4×4 de coordenadas homogéneas. Podemos entonces proyectar un punto P donde $P = (x, y, z, 1)^T$, simplemente mediante la aplicación de esta matriz de transformación al punto $P' = T_{tranformacion}P$

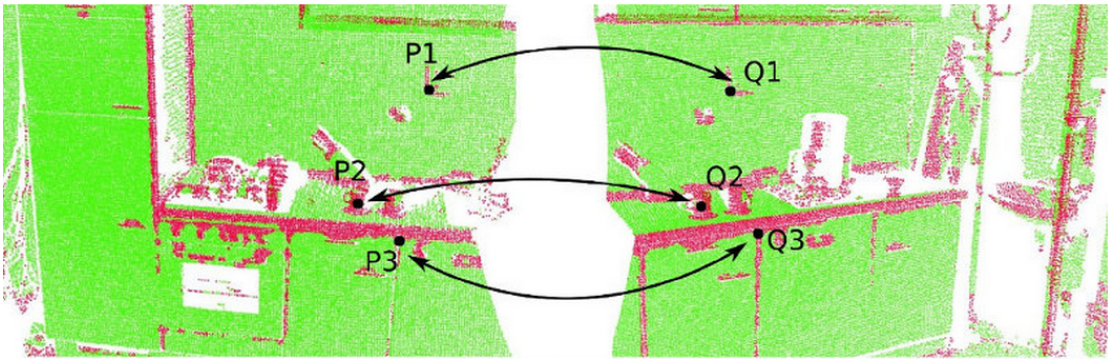


FIGURA 5.1: En la imagen de la derecha se detectan tres puntos, los cuales se vuelven a detectar en la misma área observada desde un ángulo diferente.

Teniendo en cuenta estas correspondencias entre puntos, el siguiente paso es encontrar una matriz de transformación, que dé una proyección satisfactoria para la mayoría de las correspondencias. Idealmente, podríamos simplemente calcular una transformación mediante un método de mínimos cuadrados para todos estos puntos. El número mínimo de pares necesarios para esto sería tres y cuantos más puntos, mejor definido sería el sistema. Sin embargo, debido a la incertidumbre causada por el ruido sensorial y la correspondencia incorrectas, un único punto que se estime incorrectamente podría conducir a un error significativo en la transformación final. Por lo tanto, una mejor transformación se puede encontrar iterativamente con el método RANSAC descrito en este capítulo.

5.2. Registro 3D

La mayor parte de las soluciones aportadas en el ámbito de la reconstrucción 3D con cámaras RGB-D como el sensor Kinect se basan en un proceso de reconstrucción acumulativo entre pares de fotogramas RGB-D. Este método consiste en determinar la traslación y rotación del sensor con seis grados de libertad entre un fotograma origen y otro destino. Este método de alineación de fotogramas se conoce como "alineación por pares" y es el enfoque que más se está aplicando y el cual está constituido por dos etapas:

1. Emparejamiento de características visuales y aproximación a la pose mediante RANSAC.

2. Refinamiento de la pose mediante ICP.

En los trabajos [79][70][83] se ha seguido este enfoque de aproximación visual a la pose con RANSAC y de refinamiento mediante un algoritmo ICP. Este método ha probado

ser bastante eficaz a la hora de reconstruir mapas de interiores. Por esta razón, además de haber sido el enfoque más aplicado, se ha decidido seguir los pasos de este enfoque en la implementación de este trabajo. El proceso de reconstrucción se basa en la estimación y el refinamiento de la pose relativa entre fotogramas consecutivos y para ello se sigue el enfoque de las dos etapas mencionado anteriormente.

Para realizar la alineación de los fotogramas RGB-D se utiliza el algoritmo RGB-D SLAM, cuya descripción se muestra en el pseudocódigo en la capítulo 3.

5.3. RANSAC (random sample consensus)

Como se ha explicado anteriormente la existencia de ruido en los datos es inherente en los sensores de medición, además, siempre se considera que habrá correspondencias incorrectas entre puntos 3D, esto supone un problema a la hora de estimar la transformación entre dos nubes de puntos. Por tal motivo es importante contar con un método que permita la eliminación de datos atípicos con objeto de reducir los falsos positivos.

"Random sample consensus", o RANSAC para abreviar, es un algoritmo iterativo utilizado para adaptar los parámetros de un modelo matemático a los datos experimentales. RANSAC es un método adecuado cuando un conjunto de datos contiene un alto porcentaje de los valores atípicos, es decir, mediciones que sufren de errores tan grandes que su validez es baja. El método fue presentado por primera vez a principios de los años ochenta en [84] y se sugirió para ser un método adecuado en el análisis de imágenes automatizado.

Supongamos un modelo matemático que tiene n parámetros libres que pueda ser estimado dado un conjunto de medidas P. El número de mediciones en P tiene que ser mayor que n. Sean S y T dos subconjuntos de variables diferentes de P.

Dados los supuestos el algoritmo RANSAC funciona de la siguiente manera:

- Seleccionar aleatoriamente un subconjunto de las mediciones en P y llamarlo S. Utilice S para hacer una primera estimación de los n parámetros libres del modelo.
- Use la estimación actual del modelo para seleccionar un nuevo subconjunto de puntos T a partir de las mediciones que estén dentro de una cierta tolerancia al error del modelo.
- Si T contiene más medidas que algún límite dado entonces volver a estimar los parámetros libres del modelo de acuerdo con este nuevo subgrupo. Calcular una

medida de lo bien que T y el modelo coinciden, almacenar ese valor y seleccionar un nuevo subconjunto S.

- Si T no contiene más medidas que el límite dado, seleccionar al azar a un nuevo subconjunto S de P y empezar todo de nuevo.
- Si ninguno de los subconjuntos seleccionados tiene más mediciones que el límite, se termina el algoritmo como fracaso, o si se ha alcanzado el número máximo de iteraciones.

El método se caracteriza por estos parámetros:

- La tolerancia del error utilizado para determinar cuando los datos no son parte del modelo.
- El número máximo de iteraciones del algoritmo.
- El número mínimo de mediciones en un subconjunto que se utilizará para la estimación de los parámetros.

La gran ventaja con RANSAC es la forma robusta que maneja valores atípicos en el conjunto de datos, esto se ilustra en la Figura 5.2. Los inconvenientes con RANSAC es que no garantiza alguna solución, ni que una solución dada es óptima. Además, los tres parámetros mencionados anteriormente son en gran medida específicos del problema, lo que significa que se requiere un ajuste experimental para el caso específico tratado.

A continuación se presenta el algoritmo en pseudo código 1.

En el caso de la estimación de la transformación rígida, el modelo que se trata de estimar es la matriz T de 4×4 que mejor explica la traslación y rotación de los emparejamientos de puntos 3D correspondientes a los emparejamientos 2D. Para estimar los parámetros de la transformación rígida hacen falta un mínimo de 3 puntos. Tras estimar el modelo en cada iteración, se consideran como datos típicos los emparejamientos de puntos 3D $x_i \leftrightarrow x_i^{'}$ que cumplen con que la distancia de $x_i^{'}$ a Txi es menor que un cierto umbral.

5.4. ICP (iterative closest point)

El registro óptimo de nubes de puntos es un problema importante a la que muchos autores han contribuido. Iterative Closest Point (ICP) es uno de los métodos de registro más intuitivos para hacerlo. El algoritmo ICP fue introducido en 1991 por [85] e independiente por [81] y con el tiempo fue desarrollado por varios investigadores. El objetivo general

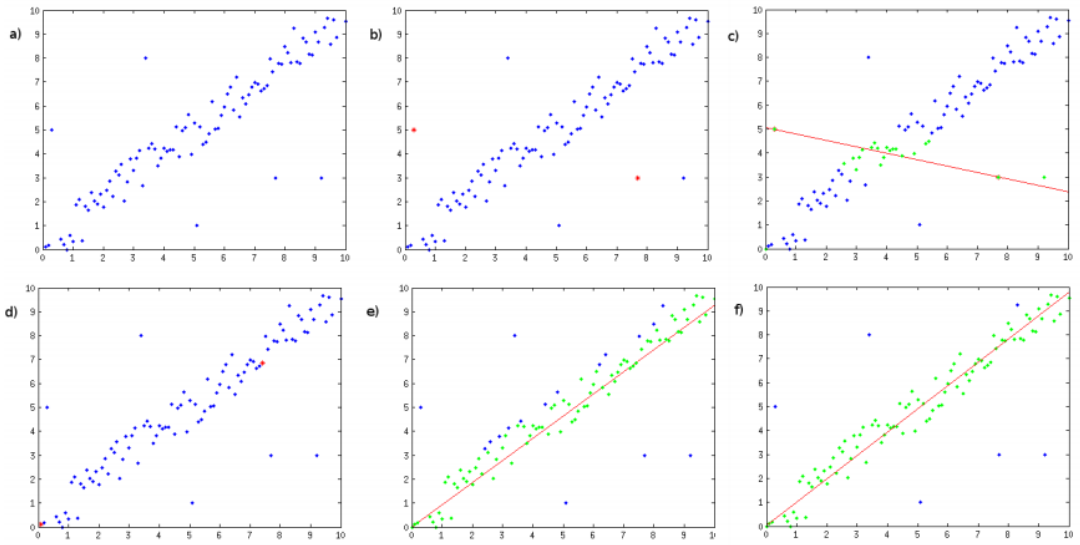


FIGURA 5.2: Algoritmo RANSAC tratando de conseguir el modelo de una línea. a) los datos sin procesar. b) Dos puntos al azar seleccionado. c) Modelo estimado con esos puntos y datos típicos. d) Otros dos puntos al azar son seleccionados. e) Modelo estimado con los nuevos puntos y datos típicos. f) Modelo estimado con los mejores datos típicos del modelo.

Algoritmo 1 General RANSAC

Entrada: Conjunto de datos de puntos.
Salida: $MejorModelo$, $MejoresCorrespondencias$.

Definir el número de iteraciones N.
para $i = 1$ hasta N **hacer**
  Muestras ← seleccionar k puntos aleatoriamente.
  Calcular *modeloactual* de las Muestras.
  $Correspondencias \leftarrow 0$.
  para Todos los puntos **hacer**
    Evaluar modeloactual para el punto y calcular su *error*
    si $error < Umbral$ **entonces**
      $Correspondencias \leftarrow Correspondencias$ + punto
    fin si
  fin para
  Contar el número de *Correspondencias* y calcular su error medio.
  si *modeloactual* es valido **entonces**
    Recalcular *modeloactual* de las *Correspondencias*
    si *modeloactual* es mejor que $MejorModelo$ **entonces**
      $MejorModelo \leftarrow modeloactual$
      $MejoresCorrespondencias \leftarrow Correspondencias$
    fin si
  fin si
fin para
devolver $MejorModelo$, $MejoresCorrespondencias$.

es aplicar una transformación a una de las nubes y que se aproxime a la otra lo más que sea posible. Llamaremos la primera serie de puntos como "puntos del modelo" y al otro conjunto como "puntos de datos". La idea es aplicar la transformación a los "puntos de datos" a fin de que éstos tengan la mejor alineación con los "puntos del modelo". En este trabajo, se denota el conjunto de "puntos del modelo" como $Q = q_1, q_2, q_3, ...q_N$ y el conjunto de "puntos del modelo" como $P = p_1, p_2, p_3, ...p_N$. Para formular una métrica de ajuste matemáticamente, una métrica de error o función objetivo E se define 5.1. Una métrica popular es la suma de errores cuadráticos, es decir, las distancias al cuadrado de los "puntos de datos" a sus vecinos más cercanos en los "puntos del modelo" después de aplicar la transformación τ.

$$E = \sum_{i=1}^{N} \|\tau(p_i) - q_i\|^2 \tag{5.1}$$

Las transformaciones pueden ser rígidas o no rígidas y la tranformación correcta debe elegirse de acuerdo con la tarea. Para este trabajo se asumen una tranformación no rígida. En el espacio 3D, tal transformación tiene seis grados de libertad, tres de rotaciones y tres de traslaciones y la función objetivo se convierte en una función de seis variables. Expresando una rotación en términos de la matriz de rotación R y una traslación con en el vector T, el problema puede ser escrito por 5.2.

$$E = \sum_{i=1}^{N} \|(R(p_i) + \vec{T}) - q_i\|^2 \tag{5.2}$$

Normalmente no se tiene información de los datos a registrarse, como cuáles tiene correspondencia. Esto quiere decir, que no sabemos qué punto q_i será el vecino más cercano a p_i después de la transformación. Esta información puede ser aproximada al realizar una búsqueda del vecino más cercano, explicada en el capítulo 3. Los métodos de minimización no lineales presentados hasta el momento y mas adelante, hábilmente eligen alguna transformación y determinar cómo se debe cambiar para disminuir E.

5.5. Algoritmo ICP

El algoritmo ICP toma dos nubes de puntos $Q = a_i$ y $P = p_i$ y una transformación inicial τ_0 que acerca la nube de puntos Q hacia P y devuelve la transformación τ que mejor alinea P con Q al resolver un problema de minimización. El algoritmo por tanto, obtiene un mínimo local por lo que puede no converger si las nubes no están suficientemente

próximas la una a la otra. En el algoritmo 2 se muestra el pseudocódigo estándar de ICP:

Algoritmo 2 Algoritmo Iterate Closest Point estándar

Entrada: Dos nubes de puntos: $Q = \{q_1, q_2, ...q_N\} P = \{p_1, p_2, ...p_N\}$
1: Conjetura inicial: Transformación τ_0.
Salida: Transformación correcta τ.
2: $\tau = \tau_0$.
3: **mientras** No ha convergido **hacer**
4: **para** $i = 1$ hasta N **hacer**
5: $d_i = EncontrarELPuntoMasCercanoEnD(\tau \cdot s_i)$
6: **si** $\|d_i - \tau \cdot s_i\| < e_max$ **entonces**
7: $w_i = 1$
8: **si no**
9: $w_i = 0$
10: **fin si**
11: **fin para**
12: $\tau = argmin_\tau\{\sum_i w_i \|\tau \cdot s_i - d_i\|^2\}$.
13: **fin mientras**

Dada una nube de puntos origen P y otra destino Q, el algoritmo ICP se puede resumir conceptualmente en tres etapas:

1. Determinar las correspondencias $q_i \leftrightarrow p_i$ entre los puntos de las nubes Q y P. Para ello se transforma cada uno de los puntos P_i de P obteniendo el punto transformado $p_i^{'}$ y se busca el vecino más cercano q_i de $p_i^{'}$ en la nube Q. Para buscar los vecinos más cercanos se utilizan árboles kd. Si la distancia de q_i a $p_i^{'}$ es mayor que un cierto umbral se desprecia dicha correspondencia.

2. Minimización: Estimar la transformación que minimiza las distancias entre las correspondencias.

3. Transformación: Aplicar la transformación estimada a la nube de puntos P para aproximarla hacia la nube Q.

Este proceso se repite iterativamente hasta que se cumplan las condiciones de convergencia como: se haya sobrepasado un determinado límite de iteraciones o que el cambio relativo en la métrica de error sea menor a un umbral. En muchos casos, el algoritmo converge muy rápidamente, sin embargo pueden surgir varios problemas:

- Múltiple mínimos locales en la métrica del error: el algoritmo puede converger hacia uno de los mínimos locales en lugar del mínimo global.

- El ruido y los valores atípicos: estos causan que la métrica de error nunca sea cero. Los valores extremos pueden causar resultados erróneos, especialmente por la ponderación del error cuadrático.
- Superposición parcial: las nubes de puntos pueden no parecerse a las mismas partes de un objeto, sin embargo, la superposición parcial debe ser necesaria.

En la figura 5.3 se muestra visualmente el proceso realizado durante una iteración de ICP.

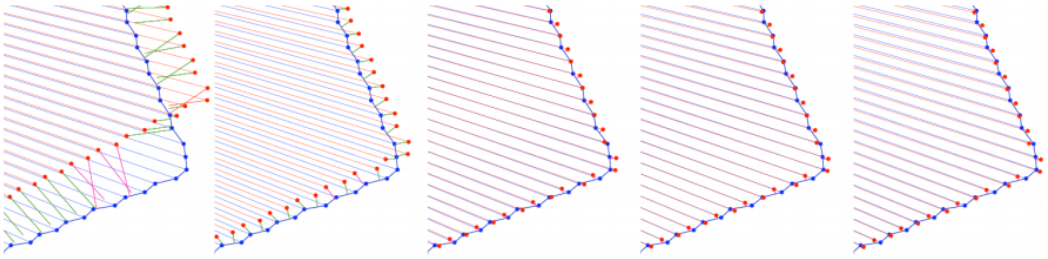


FIGURA 5.3: K-ésima iteración del algoritmo ICP.Se busca el vecino más cercano de cada punto transformado de la nube origen con T_{k1} en la nube destino dentro de un determinado radio de búsqueda. Se estima la transformación rígida T_k con las correspondencias de puntos 3D y se itera para encontrar la solución.

Muchos investigadores han propuesto soluciones para resolver los problemas del algoritmo ICP. Esto ha dado lugar a varias variantes del algoritmo, como el ICP general(GICP), que se explica más adelante. Una división de variantes ICP [86] identifican seis etapas distintas del algoritmo:

1. Selección
2. Emparejamiento
3. Ponderación
4. Rechazar
5. Métricas de error
6. Minimización de la métrica del error

En las siguientes secciones se dará una breve descripción de cada uno de estos y las estrategias seguidas según los requisitos de del problema tratado en este trabajo de tesis.

5.5.1. Selección

Puede ser beneficioso considerar sólo algunos de los "puntos del modelo" y de los datos antes de aplicar el algoritmo ICP. Por ejemplo los valores atípicos pueden ser filtrados en base a un umbral. Para reducir la complejidad computacional, la cantidad el tiempo cálculos en gran medida. La idea detrás del muestreo aleatorio es evitar cualquier sesgo hacia valores atípicos [87], en este trabajo se opto por utilizar un submuestreo uniforme [88].

5.5.2. Emparejamiento

Encontrar a los vecinos más cercanos es generalmente la etapa computacionalmente más intensiva en el algoritmo ICP. El método de fuerza bruta se limita a calcular las distancias a todos los vecino candidatos y recoger el más cercano. Este método escala de forma lineal y sólo es útil para algunas búsquedas o para datos de superiores dimensionales.

Algunas técnicas pueden ser empleadas para acelerar la búsqueda del vecino más cercano. Estos incluyen los árboles *kd* y triangulaciones de Delaunay, los cuales proporcionan una gran mejora en los tiempos de búsqueda, a expensas de algún procesamiento previo. En la mayoría de las iteraciones del algoritmo ICP, los "puntos de datos" y los "puntos del modelo" estarán bastante cercanos unos del otros. Normalmente se favorece el uso de árboles *kd* y es el que se usa para obtener un mejor desempeño en este proyecto de tesis, los cuales se explican en el capítulo 3.

5.5.3. Ponderación

A los pares de puntos emparejados se les podría aplicar una ponderación diferente en función de su compatibilidad en algún sentido. Prácticamente esto significaría multiplicar la métrica del error con un factor w específico. El factor podría basarse en la distancia, color, la curvatura contra direcciones normales tangentes, etc. En estos trabajos se mencionan algunos factores más [86][89]. En este trabajo no se usa ponderación alguna.

5.5.4. Rechazar

Los pares de puntos pueden ser rechazados después de la etapa de emparejamiento. Esto se puede hacer en una evaluación estadística de las distancias de vecinos más cercanos. Por ejemplo, los 10 % peores pares de puntos pueden ser rechazados, etc. En este proyecto se rechazaron los puntos que estuvieran a una distancia mayor a 0,001 metros.

5.5.5. Métricas de error

La métrica de error define la función objetivo que se trata de reducir al mínimo en cada iteración del algoritmo. Dos métricas comúnmente utilizados son:

La Minimización punto a punto: suma las distancias al cuadrado de los "puntos de datos" y los "puntos del modelo" y se expresa de la siguiente manera:

$$E = \sum_{i=1}^{N} \|(R(p_i) + \vec{T}) - q_i\|^2 \tag{5.3}$$

En términos físicos esta métrica puede ser visualizada como la energía potencial total de resortes unidos entre pares de puntos emparejados.

Minimización punto a plano: suma las distancias de los "puntos de datos" a los planos tangentes en el que los "puntos del modelo" coincidentes residen. Su formulación matemática es la siguiente:

$$E = \sum_{i=1}^{N} [((R(p_i) + \vec{T}) - q_i) \cdot \vec{n_i}]^2 \tag{5.4}$$

Donde $\vec{n_i}$ denota las normales tangentes estimadas en el punto i de modelo. La interpretación física podría ser la energía potencial total de resortes que son libres de moverse en las tangentes de los "puntos del modelo" y que están fijados a los "puntos de datos". La minimización punto a plano permite que las regiones planas se deslizan a lo largo de las unas de las otras, lo cual puede ser ventajoso. Otras métricas de error se pueden usar o proponer, en este trabajo se usa la métrica de punto a punto.

5.5.6. Minimización de la métrica del error

Existe una solución de forma cerrada para la minimización de la métrica de error de punto a punto [90][91][92]. La cual se basa en la descomposición del valor singular.

La métrica punto a plano sólo tiene una solución cerrada después de la linealización de la matriz de rotación R. La linealización introduce un error, pero en iteraciones finales, se espera que las rotaciones sean pequeñas y la matriz de rotación linealizado esté cerca de la real. Se propuso la minimización y deriva por [93].

5.6. GICP (generalized iterative closest point)

Generalized ICP es una variante de ICP propuesta en [82]. GICP se basa en la inclusión de un modelo probabilístico en el paso de minimización de ICP. Esta propuesta deja el resto del algoritmo ICP igual, manteniendo la simplicidad y el rendimiento de la versión estándar. Las correspondencias entre los puntos también usan la distancia euclidiana en lugar de una medida probabilística, esto permite continuar usando árboles *kd* para la búsqueda de los vecinos más cercanos.

Esta propuesta se centra en modificar el paso de minimización del algoritmo ICP, por lo que el resto del algoritmo permanece inalterado. Para mostrar la modificación introducida en GICP con respecto a ICP esta sección se centra únicamente en el paso de minimización. De esta forma se asume que ya se ha realizado la búsqueda en los vecinos más cercanos con árboles *kd* y que se dispone de dos nubes de puntos $A = (a_1, a_2, a_3, ...a_N)$ y $B = (b_1, b_2, b_3, ...b_N)$ indexadas por sus correspondencias (a_i se corresponde con b_i). También se asume que las correspondencias con $\|m_i - T \cdot b_i\| > dmax$ se han suprimido de A y de B.

En el modelo probabilístico se asume la existencia de un conjunto subyacente de puntos, $\hat{A} = (\hat{a_1}, \hat{a_2}, \hat{a_3}, ...\hat{a_N})$ y $B = (\hat{b_1}, \hat{b_2}, \hat{b_3}, ...\hat{b_N})$, que generan A y B de acuerdo con $ai \sim N(\hat{a}_i, C_i^A)$ y $b_i \sim N(\hat{b}_i, C_i^B)$. En este caso, C_i^A y C_i^B son matrices de covarianza asociadas a los puntos medidos. Si se asume correspondencia perfecta (geométricamente perfecta sin errores debidos a oclusión o muestreo) y la transformación correcta T^*, se sabe que:

$$\hat{b_i} = T * \hat{a_i} \tag{5.5}$$

Para una transformación T arbitraria, se define $d_i^{(T)} = b_i - Ta_i$ y se considera la distribución de donde proviene $d_i^{(T*)}$. Como se asume que a_i y b_i se originan por gaussianas independientes, aplicando la ecuación 5.5 se tiene que:

$$d_i^{(T*)} \sim N(\hat{b_i} - (T*)\hat{a_i}, C_i^{(B)} + (T*)C_i^A(T*)^T) = N(0, C_i^B + (T*)C_i^A(T*)^T) \tag{5.6}$$

Ahora se aplica la estimación por máxima verosimilitud para computar T iterativamente:

$$T = argmax_t \prod_i p(d_i^{(T)}) = argmax_t \sum_i log(p(d_i^{(T)})) \tag{5.7}$$

Que puede simplificarse como:

$$T = argmax_t d_i^{(T)^T} (C_i^B + TC_i^A T^T)^{-1} \tag{5.8}$$

El algoritmo ICP estándar puede verse como un caso especial si se establece lo siguiente:

$$C_i^B = IC_i^A = 0 \tag{5.9}$$

En este caso, 5.8 se convierte en:

$$T = argmax_t \sum_i d_i^{(T)^T} d_i^{(T)} = argmax_t \sum_i \| \, d_i^{(T)} \|^2 \tag{5.10}$$

Que es exactamente la fórmula de actualización de ICP estándar. Con esto definimos el marco generalizado de ICP. Sin embargo, tenemos más libertad en el modelado; ya que somos libres de elegir cualquier conjunto de covarianzas para C_i^B y C_i^A.

5.7. Estimación de las poses

Se explicó cómo determinar una transformación rígida entre un par de fotogramas. Para cada transformación, una estimación de la pose de la cámara se puede calcular. Conociendo la pose inicial, el primer paso es determinar una estimación de cualquier pose después de una sucesión de transformaciones. Teniendo en cuenta una secuencia finita de fotogramas $[fotograma_0; fotograma_n]$, sea P_k en la posición $k \in [0; N]$. Para la transformación de 6 grados, compuesta por 3 ejes de rotación y 3 de traslación, que puede ser representado por una matriz de 4×4 con coordenadas homogéneas, donde R es una matriz de rotación de 3×3 y t un vector de traslación:

$$P_k = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.11}$$

Para $i > 0$, tenemos la transformación T_{i-1}^i que une la posición P_{i-1} a la posición de P_i. Cada transformación T también puede ser representada por una matriz de 4×4 similar. Si P_0 determina la posición inicial, entonces podemos calcular la posición P_i mediante la combinación de todas las transformaciones como:

$$P_k = \prod_{i=k}^{1} T_{i-1}^{i} P_0 \tag{5.12}$$

Como el producto de matrices no es conmutativo, es esencial seguir el orden correcto al multiplicar las matrices. Por ejemplo, para la pose P_3 tenemos:

$$P_3 = T_2^3 T_1^2 T_0^1 P_0 \tag{5.13}$$

En general, la posición inicial se define por la orientación inicial de la cámara para cada eje, almacenada en la matriz de rotación inicial R_0 y el vector de traslación inicial $t_0 = (x_0, y_0, z_0)^T$, como sigue:

$$P_0 = \begin{bmatrix} R0_{11} & R0_{12} & R0_{13} & t_0 \\ R0_{21} & R0_{22} & R0_{23} & t_0 \\ R0_{31} & R0_{32} & R0_{33} & t_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.14}$$

Como una elección arbitraria, podemos decidir la posición inicial como el origen del sistema de coordenadas, que está dada por la matriz de identidad I_4:

$$P_0 = I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.15}$$

Capítulo 6

Características visuales

En este capítulo se describen SURF que constituyen el método que se han aplicado en este proyecto para la detección, extracción y descripción de las características visuales.

En el capítulo 4 se ha descrito el método que se aplica en este trabajo para obtener las medidas de odometría visual, el cual consiste en la alineación de fotogramas RGB-D consecutivos. También se motivó la inclusión de una etapa previa a la ejecución de ICP que consiste en la estimación de la transformación mediante RANSAC. Este método de inicialización de la transformación requiere correspondencias de características SURF visuales extraídas de cada imagen.

6.1. SURF (speeded-up robust features)

Es un detector y descriptor de características que se puede utilizar para tareas tales como el reconocimiento de objetos o reconstrucción 3D. Está inspirado en parte por el descriptor "scale-invariant feature transform" (SIFT). La versión estándar de SURF es varias veces más rápido que SIFT y afirman sus autores que es más robusto frente a las diferentes transformaciones de imagen que SIFT,[33].

El descriptor SURF está basado en la suma de las respuestas Haar wavelets 2D y hace un uso eficiente de las imágenes integrales. En la siguiente sección se explica la imagen integral y la utilidad que tiene para realizar operaciones de convolución de forma muy eficiente. Otra de las optimizaciones de SURF es el uso de una aproximación entera al determinante de la *Hessiana* para la detección de "objetos binarios grandes", que se pueden calcular mucho más rápido con las imágenes integrales.

El detector de SURF se basa en el determinante de la matriz $Hessiana$. Para motivar el uso de la $Hessiana$, primero considérese una función continua de dos variables $f(x, y)$. La matriz $Hessiana$, H es la matriz de las derivadas parciales de f.

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{6.1}$$

El determinante de esta matriz, más conocido como discriminaste, se calcula como:

$$det(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - (\frac{\partial^2 f}{\partial x \partial x})^2 \tag{6.2}$$

El valor del discriminaste se puede usar para clasificar el máximo o mínimo de la función mediante la prueba de la segunda derivada. Como el determinante es el producto de los autovalores de la $Hessiana$, se pueden clasificar los puntos basándose en el signo del resultado. Si el determinante es negativo, entonces los autovalores tienen signos diferentes y por tanto el punto no es un extremo local; si es positivo, entonces los dos autovalores son positivos o negativos y el punto se clasifica como extremo.

Aplicando esta teoría al dominio discreto de las imágenes, se sustituye la función continua $f(x, y)$ por la intensidad de los pixeles de la imagen $I(x, y)$. El cálculo de las derivadas parciales de $f(x, y)$ se puede sustituir por la aplicación de filtros de convolución:

$$H(X,\sigma) = \begin{bmatrix} L_x x(X\sigma) & L_x y(X\sigma) \\ L_x y(X\sigma) & L_y y(X\sigma) \end{bmatrix} \tag{6.3}$$

Para el cálculo de las derivadas parciales se pueden usar kernels muestreando las segundas derivadas de la función gaussiana, donde $L_x x(x, \sigma)$ se corresponde con la evaluación de la función $\frac{\partial^2 g(\sigma)}{\partial x^2}$. No obstante, en [33] se propuso una aproximación a estos kernels mediante filtros en caja como se puede apreciar en la figura 6.1. Los filtros en caja son filtros de convolución muy simples que pueden aplicarse de una forma muy eficiente usando la representación de imagen integral como se explica más adelante.

Para el cálculo del determinante de la $Hessiana$ con filtros cuadrados se usa una fórmula que lo aproxima bastante bien, propuesta por [33]:

$$det(Happrox) = D_x x D_y y - (0{,}9 D_x y)^2 \tag{6.4}$$

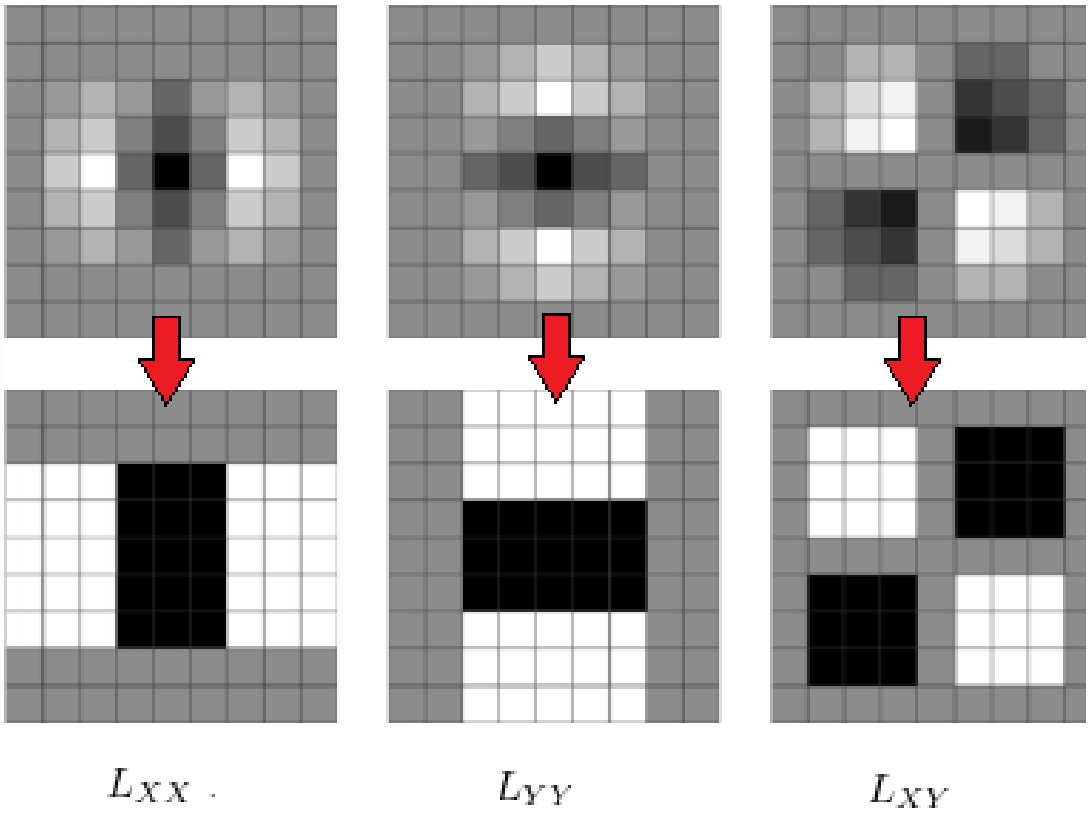


FIGURA 6.1: Los valores del los operadores L_{XX}, L_{YY} y L_{XY} se aproximan mediante respuestas de filtros cuadrados.

6.1.1. Imagen integral

La imagen integral permite el cálculo rápido de los filtros de convolución de tipo cuadrados. Un imagen integral $S(x, y)$ en una ubicación (x, y) que representa la suma de todos los pixeles en la imagen I dentro de una región rectangular formada por el origen $y(x, y)$. Ésto se define en la ecuación 6.5:

$$S(x, y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j) \tag{6.5}$$

Una vez que la imagen integral se ha calculado, solo se necesitan tres sumas para calcular la suma de las intensidades de un área rectangular, como se ilustra en la figura 6.2. Por lo tanto, el tiempo de cálculo es independiente de su tamaño.

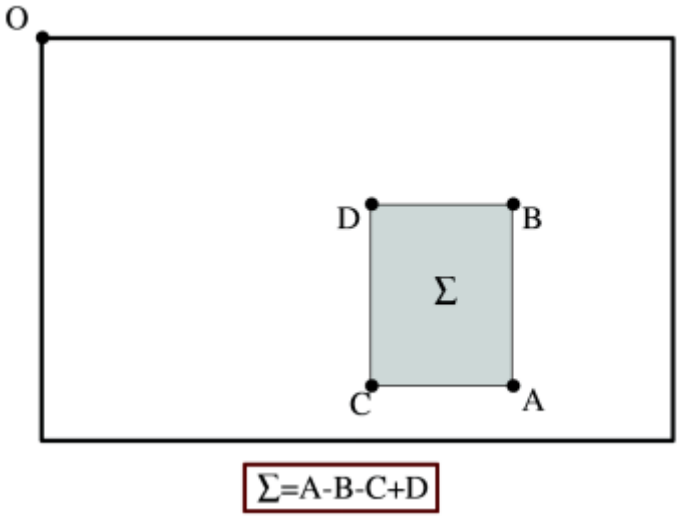


FIGURA 6.2: Usando imágenes integrales, se tarda sólo tres adiciones y cuatro accesos a memoria para calcular la suma de intensidades dentro de una región rectangular de cualquier tamaño.

6.1.2. Espacio de escalas

Para poder detectar características visuales mediante el determinante de la *Hessiana*, es necesario introducir el concepto de espacio de escalas. Un espacio de escalas es una función continua que se puede usar para encontrar extremos a través de diversas escalas. En visión por computador, un espacio de escalas típicamente se implementa como una pirámide de imágenes donde iterativamente se aplican filtros de suavizado por convolución y se reduce el tamaño de la imagen. Este es el método que se aplica en SIFT, el problema es que es computacionalmente ineficiente debido al re-escalado de las imágenes.

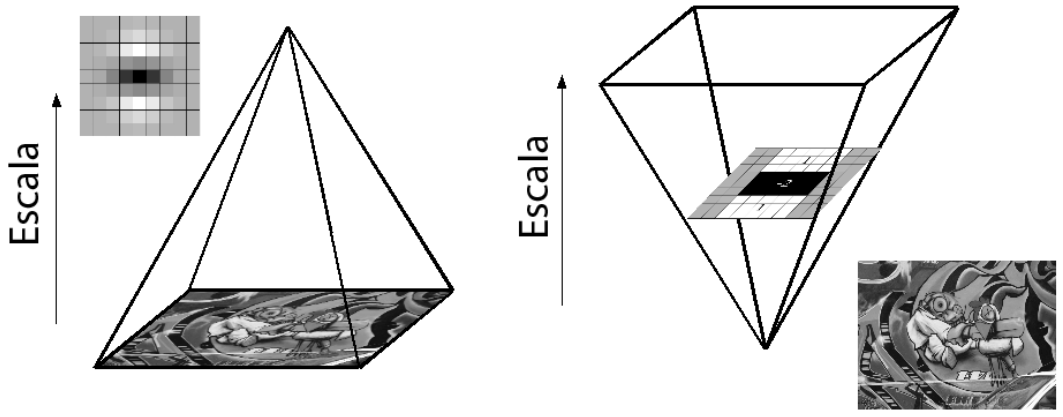


FIGURA 6.3: Izquierda: el enfoque tradicional para la construcción de un espacio de escalas. La imagen se re-escala iterativamente al tiempo que se suaviza con un filtro gaussiano de un determinado tamaño. Derecha: el enfoque que sigue SURF es que la imagen no se re-escala, sino que se realiza la convolución con filtros de tamaño cada vez mayor.

En SURF, por otra parte se aplica un concepto de espacio de escalas diferente. En este caso se parte de la base de que la construcción de los filtros cuadrados no dependen del tamaño del filtro, por lo que se crea una estructura de pirámide de filtros con los que realizar la convolución a la imagen original. Esto permite que se puedan computar varios niveles de la respuesta en el espacio de escalas simultáneamente y no iterativamente como en SIFT. La figura 6.1 muestra la diferencia entre el enfoque tradicional y el que sigue SURF para obtener el espacio de escalas. Las sucesivas capas se van obteniendo como consecuencia de aplicar gradualmente filtros mayores. En cada nueva octava, el incremento de tamaño del filtro es el resultado de doblar el incremento realizado en la octava anterior, esto se ilustra en la figura 6.4.

- Octava inicial: $9 \times 9 \xrightarrow{6} 15 \times 15 \xrightarrow{6} 21 \times 21 \xrightarrow{6} 27 \times 27$
- Siguiente octava: $15 \times 15 \xrightarrow{12} 27 \times 27 \xrightarrow{12} 39 \times 39 \xrightarrow{12} 51 \times 51$
- Siguiente octava: $27 \times 27 \xrightarrow{24} 51 \times 51 \xrightarrow{24} 75 \times 75 \xrightarrow{24} 99 \times 99$
- Y así sucesivamente...

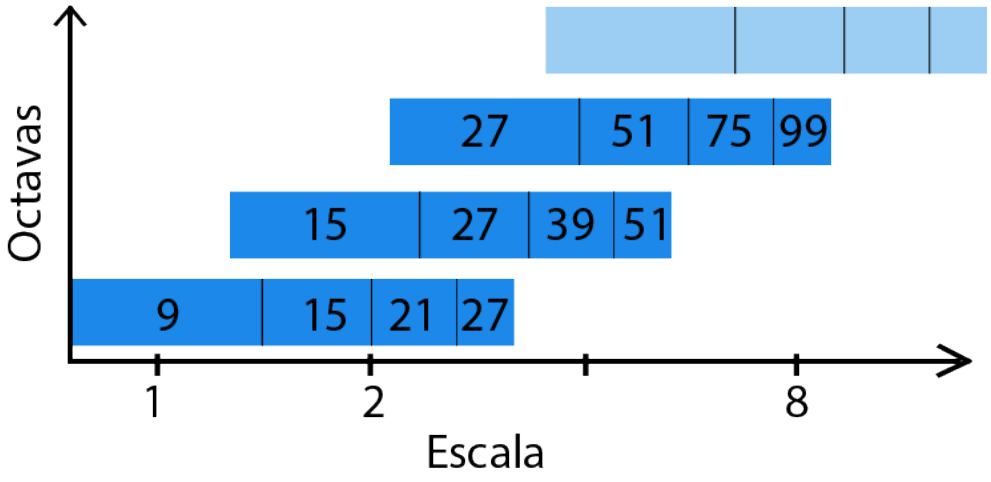


FIGURA 6.4: Representación gráfica del tamaño de los filtros usados en diferentes octavas. El eje X representa la escala y el eje Y indica la octava.

6.1.3. Detección de características

El proceso de detección de características SURF puede dividirse en tres fases:

1. Los objetos binarios grandes se filtran de forma que las características que tengan una respuesta por debajo de un cierto umbral se eliminan. Incrementar el umbral reduce el número de características detectadas, dejando únicamente las más robustas. Reducir el umbral hace que se detecten más características.

2. Se aplica supresión no maximal para encontrar un conjunto de puntos candidatos. Para hacer esto, cada pixel candidato se compara con sus 26 vecinos (8 en la escala del candidato y 9 en las escalas superior e inferior). La figura 6.5 ilustra el proceso de supresión no maximal.

3. Se interpolan los datos de la cercanía del candidato para producir una localización del pixel.

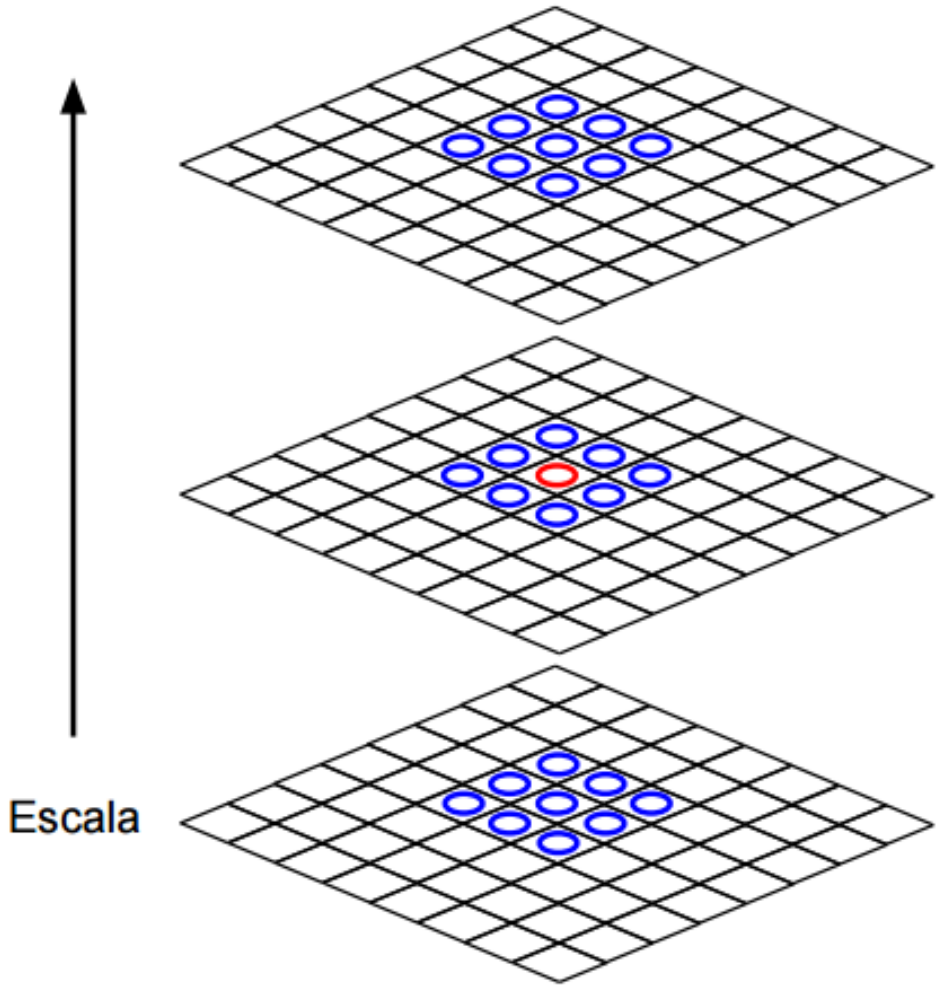


FIGURA 6.5: El pixel central se selecciona como máximo si su respuesta para el objeto binario grande es mayor que la de los pixeles de su alrededor (en su nivel y en los niveles adyacentes).

Para interpolar la localización del pixel se expresa la matriz *Hessiana* $H(x, y, \sigma)$ como su expansión de Taylor de segundo orden centrada en la característica detectada:

$$H(x) = H + \frac{\partial H^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 H}{\partial X^2} X \tag{6.6}$$

La posición interpolada del extremo $\hat{x} = (x, y, \sigma)$, se obtiene calculando la derivada de la expansión de Taylor de $H(x, y, \sigma)$ e igualando a cero:

$$\hat{x} = -\frac{\partial^2 H^{-1}}{\partial X^2} \frac{\partial H}{\partial X} \tag{6.7}$$

Para poder calcular el punto interpolado se necesita calcular la matriz 3×3 $\frac{\partial^2 H^{-1}}{\partial X^2}$ y el vector 3×1 $\frac{\partial H}{\partial X}$. Los elementos de estas matrices se computan mediante diferencias finitas en los niveles de intensidad de los pixeles, donde d_x se refiere a $\frac{\partial I}{\partial X}$ y $d_x x$ a $\frac{\partial^2 I}{\partial X^2}$ y así con las demás.

$$\frac{\partial^2 H}{\partial x^2} = \begin{bmatrix} d_{xx} & d_{yx} & d_{sx} \\ d_{xy} & d_{yy} & d_{sy} \\ d_{xs} & d_{ys} & d_{ss} \end{bmatrix} \tag{6.8}$$

$$\frac{\partial H}{\partial x} = \begin{bmatrix} d_x \\ d_y \\ d_s \end{bmatrix} \tag{6.9}$$

6.1.4. Descriptor

El cómputo de los descriptores SURF se puede dividir en dos etapas. La primera consiste en extraer la información de orientación de las características con objeto de obtener un descriptor robusto a las rotaciones. La segunda fase consiste en obtener las respuestas a los Haar wavelets que se explica más adelante y que constituyen los componentes del descriptor.

El descriptor de SURF especifica cómo se distribuyen las intensidades de los pixeles vecinos de una característica detectada por el detector rápido de *Hessiana*. Esta propuesta es similar a la de SIFT, pero el uso de las imágenes integrales, junto con el uso de los filtros conocidos como Haar wavelets, mejoran el tiempo de cómputo. Los Haar wavelets [94] son simples filtros que se pueden usar para calcular los gradientes en x e y, como se puede apreciar en la figura 6.6. De nuevo se usa la representación de imagen integral para realizar la convolución con los filtros para operar de forma eficiente.

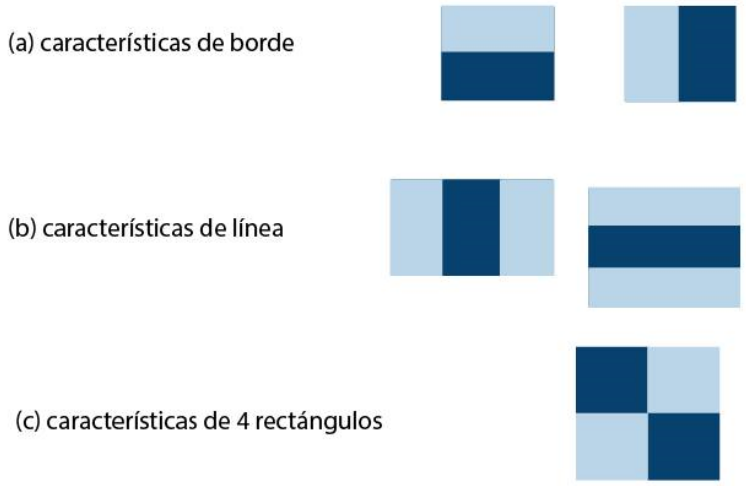


FIGURA 6.6: La imagen ilustra los diferentes tipos de Haar wavelets.

6.1.4.1. Orientación

El siguiente paso, sería la asignación de orientación. Este paso es importante puesto que es la que le otorga al descriptor invarianza ante la rotación, otorgándole a cada punto de interés una orientación, el cálculo de la orientación de la característica se puede dividir en dos tareas:

1. Computar las respuestas Haar wavelet de tamaño 4σ dentro de un radio de 6σ alrededor de la característica detectada. Las respuestas se ponderan con una gaussiana centrada en la característica con desviación típica de $2{,}5\sigma$. Las respuestas se representan como un vector de puntos 2D en el espacio, donde cada uno de estos puntos se corresponde con la dirección del vector gradiente evaluado en cada uno de los pixeles de la región circular.

2. Seleccionar la dirección predominante de las respuestas. Se rota una sección de circunferencia de $\frac{\pi}{3}$ alrededor del origen de la característica y se suman las componentes de las respuestas que quedan dentro de la sección. El vector resultante con mayor módulo representa la orientación predominante de la característica. El proceso de selección de la orientación predominante se puede apreciar en la figura 6.7.

6.1.4.2. Componentes del descriptor

La extracción de los componentes del descriptor SURF puede dividirse a su vez en dos procesos más simples:

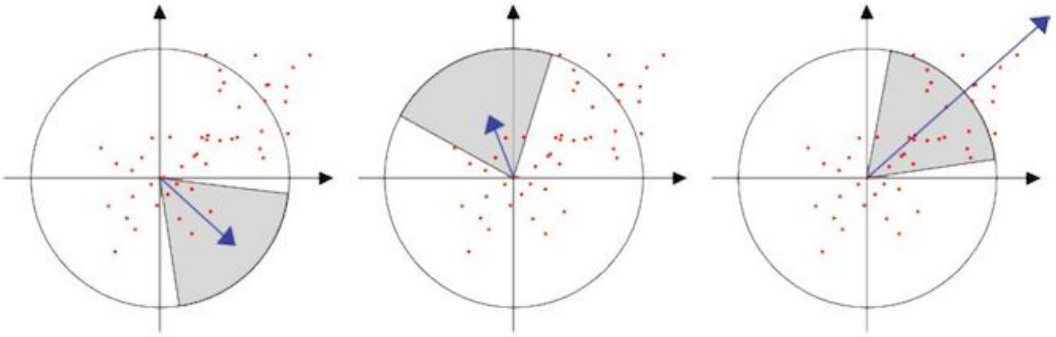


FIGURA 6.7: A medida que la sección de circunferencia rota en torno a la característica, se suman las componentes de las respuestas produciendo los vectores de dirección de la característica. El vector de mayor módulo determina la orientación predominante.

1. Construir una ventana rectangular de tamaño 20σ en torno a la característica detectada y orientada conforme a la dirección predominante de las respuestas Haar wavelets.

2. Dividir cada ventana en 4×4 sub-regiones rectangulares para obtener las respuestas Haar wavelet de cada sub-región. Dentro de cada sub-región se computa la respuesta Haar wavelet de tamaño 2σ sobre 25 puntos distribuidos de forma regular por la sub-región. Refiriéndose a las respuestas Haar wavelet en x e y como d_x y d_y, cada sub-región aporta un vector de 4 componentes y $V_{subregion} = [\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|]$. El proceso de extracción de las componentes de cada sub-región se muestra en la figura 6.8. Como cada sub-región aporta un vector de 4 componentes, el descriptor resultante es de tamaño $4 \times 4 \times 4 = 64$. Este descriptor es invariante a la rotación, escala y cambios de iluminación.

6.1.5. Emparejamiento

El Emparejamiento en SURF se lleva acabo usando el signo de la laplaciana y la distancia euclidiana. Esta mejora importante del uso del signo de la laplaciana no añade costo de cálculo, puesto que ya se calcula durante la detección. En la etapa de emparejamiento de características, sólo necesitamos comparar si las características tienen el mismo tipo de contraste, como se muestra en la figura 6.9. Esta información mínima permite hacer el emparejamiento de una forma más rápida, sin reducir el rendimiento del descriptor.

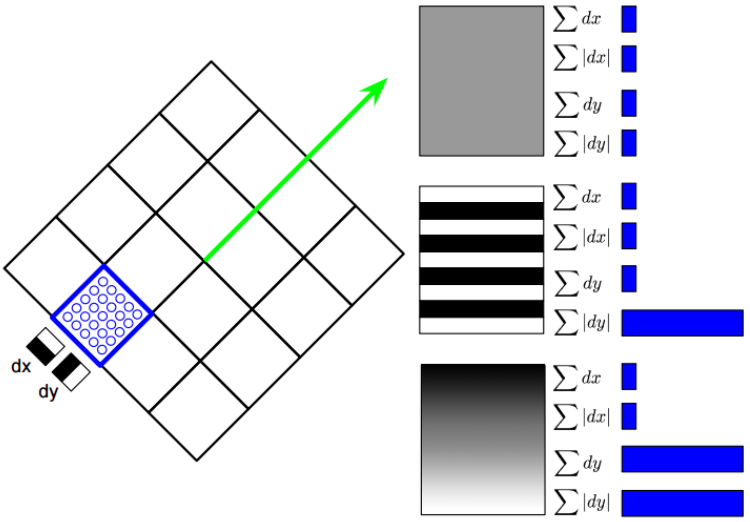


FIGURA 6.8: Izquierda: el rectángulo marcado limita las 16 sub-regiones y los 25 puntos de su interior representan los pixeles con los que se calcula la respuesta a los Haar wavelets. Como se puede apreciar, las respuestas se computan relativas a la orientación predominante de la característica. Derecha: ejemplo de las componentes del descriptor SURF calculados para sub-regiones diferentes.

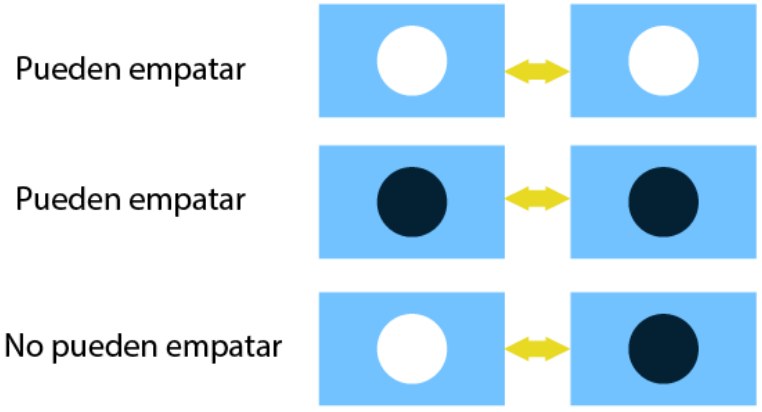


FIGURA 6.9: Solo se comparan las características si tienen el mismo tipo de contraste, este proceso ahorra tiempo de emparejamiento sin disminuir la precisión. Después de esto, si la distancia euclídea está por debajo de un umbral, las dos características de SURF se consideran como la misma.

Capítulo 7

Construcción del grafo de poses

En este capítulo se presenta la solución propuesta en este trabajo para el problema de cierre de ciclos y la construcción del grafo de poses. Un enfoque simple para un robot para determinar su posición es mediante el uso de la odometría, que estima la posición actual sobre la base de una posición de inicio. Los datos de los actuadores del robot se usa para calcular los movimientos. Por ejemplo, para una rueda sería la circunferencia de la rueda y cuantas veces la rueda ha dado vueltas. Entonces para evitar colisiones del robot podría utilizar la información de odometría.

El gran problema de este método es el error de la integración en la odometría, no importa qué tan precisa y bien calibrado sea el sistema, un pequeño error es inevitable. Y cuanto más se mueve el robot, más se agranda el error. Imagínese que un robot se mueve en un cuadrilátero, como se muestra en la figura 7.1, aún con buena odometría, el robot no va a terminar donde se cree que lo hará. La solución a este problema se conoce como cierre de ciclos, como se mencionó en el capítulo 3. Si el robot puede reconocer de alguna manera que un lugar ya ha visitado anteriormente, será capaz de comparar las posiciones calculadas y ajustar los errores que se han producido. Este es uno de los grandes desafíos en el problema de SLAM.

7.1. La acumulación del error

La pose de la cámara se estima mediante un método de alineación de nubes de puntos. Esta alineación determina una medición u observación de los puntos del fotograma destino vistos desde el fotograma origen. Es evidente que esta observación no es exacta debido al ruido del sensor, correspondencias incorrectas, alineaciones imprecisas de RANSAC e ICP o simplemente por que la odometría jamás será perfecta. En un enfoque de odometría visual, la pose global de la cámara se obtiene componiendo las poses

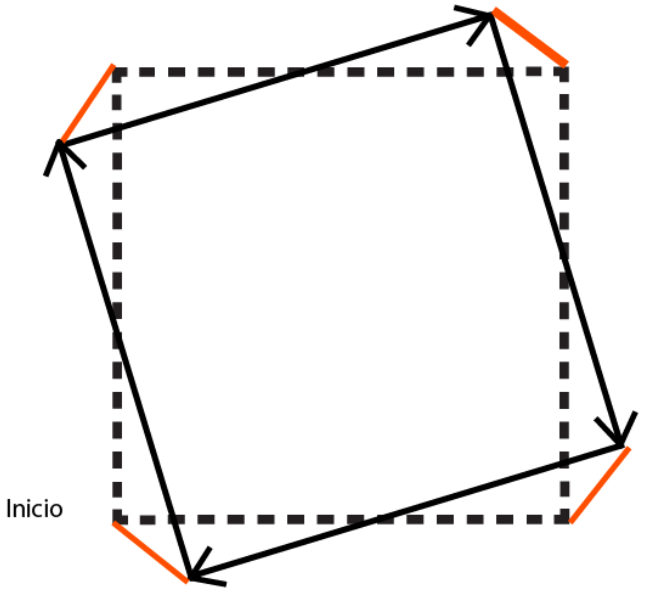


FIGURA 7.1: Un robot supone que se mueven en un cuadrado indicado por una línea de puntos. Un pequeño error y este crecerá a medida que el robot se mueva y el robot no terminará donde el supone.

relativas entre los fotogramas consecutivos, de forma que el error cometido entre cada una de estas estimaciones se acumula con cada una de las anteriores. Con el paso del tiempo y como resultado de esta acumulación del error, la trayectoria estimada se alejará cada vez más de la real, hasta el punto de que el error acumulado no es admisible y la trayectoria estimada dejará de ser útil. En la figura 7.2 se puede apreciar el efecto de la acumulación de error en la odometría visual.

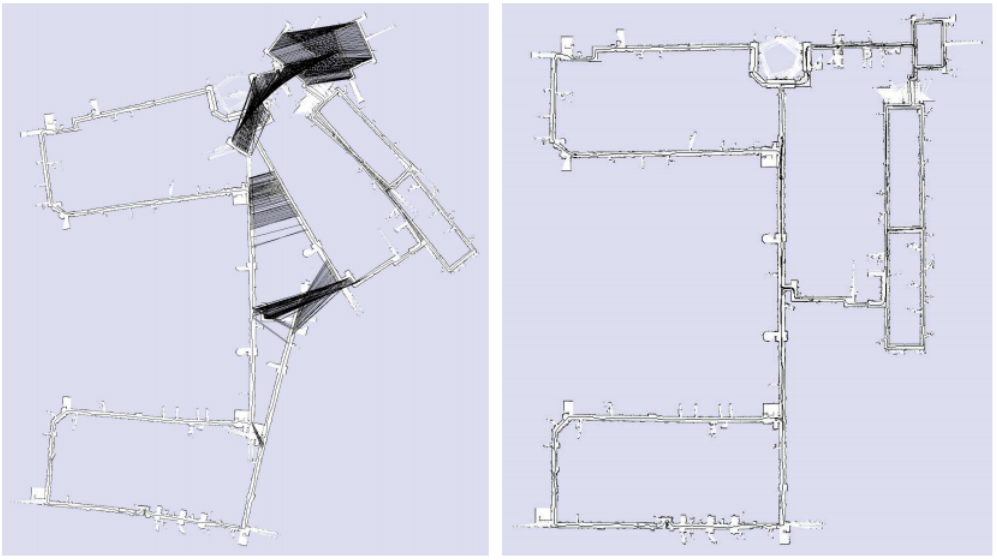


FIGURA 7.2: En la imagen de la izquierda se puede apreciar el mapa generado después de que el error se ha acumulado durante un tiempo. A la derecha, al reducir el error mediante el cierre de ciclos, se crea un un mapa mucho más consistente.

Como el problema de la acumulación de error es inherente al proceso de odometría, se utilizan formas de modelar este error para tratar de reducirlo, tal es el caso del SLAM basado en grafo como se explicó en el capítulo 3.

7.2. Contrucción del grafo y cierre de ciclos

En el capítulo 3 se explicó la estructura que tiene el grafo de poses y las restricciones espaciales, a continuación se explicará como se construye el grafo de poses para este trabajo de tesis. El problema de construir el grafo de poses no es trivial y la principal razón de ello es que es necesario detectar bucles, es decir, es necesario determinar cuándo se ha pasado por un lugar visitado anteriormente.

La detección de bucles, es un campo de investigación en el que se está trabajando para ofrecer metodologías eficientes que produzcan bucles con bajas tasas de falsos positivos [95][96]. Por tanto, la tarea de detectar bucles de forma eficiente y robusta cae fuera de los objetivos de este trabajo de tesis, principalmente por su complejidad. Por este motivo, se implementó una solución sencilla a este problema, aunque poca robusta, esta solución permite construir un grafo de poses y dar una solución al problema de acumulación del error. La idea básica de la solución propuesta es la simplificar la tarea de detección de ciclos, a la tarea de medir el número de datos típicos entre la medición actual y las mediciones anteriores.

A continuación se presenta los pasos que se siguieron en este trabajo para realizar la construcción del grafo de poses y la detección de los cierres de ciclos:

1. Calcular la pose relativa entre el fotograma actual y la pose de la fotograma anterior, si el número de datos típicos es mayor a un umbral, se agrega la nueva pose como un vértice al grafo de poses y se verificará en un paso siguiente sí existe un cierre de ciclo. En caso contrario, si el número de datos típicos es menor al umbral, se descarta la pose.

2. Se calcula la odometría visual del fotograma N actual y los fotogramas anteriores de 1 a $N-2$.

3. Si el número de emparejamientos correctos calculados mediante RANSAC es superior a un umbral, se considera que se ha vuelto a pasar por un lugar visitado anteriormente, es decir, se considera que se ha encontrado un cierre de ciclo. Se refina la transformación que relaciona las 2 poses mediante ICP y se añade la restricción espacial entre dicha pose y la actual.

4. Se calcula la odometría visual que relaciona la pose actual N con la anterior $N-1$. y se agrega la restricción espacial al grafo.

En la descripción e implementación de este método simple de detección de bucles, se usa principalmente la información de datos típicos que existen entre un fotograma a otro, el cual se calcula mediante RANSAC. El número de datos típicos nos dice simplemente cuantos emparejamientos correctos de puntos de referencia (características) se encontraron de un fotograma a otro y se utiliza para medir la relación que existe entre un fotograma a otro, entre más parecido sea un fotograma a otro, se espera que se tenga un mayor número de datos típicos.

En la implementación se utiliza un umbral para determinar si se agrega un nuevo fotograma al grafo, este tiene como idea limitar el número de fotogramas y agregar solo aquellos que aporten nueva información. Por ejemplo, puede existir el caso que se realicen múltiples mediciones desde una misma posición, las cuales no aportarían información nueva, por lo que se podrían considerar redundantes. Además, se detectarían múltiples cierres de ciclos entre estas mismas mediciones, ya que se detectaría una gran cantidad de datos típicos. La idea de aplicar este umbral es que todas esa información se pueda resumir en solo unos cuantos fotogramas y relaciones espaciales y solo agregar información nueva, de esa manera se espera reducir el tiempo de cómputo utilizado en la detección de ciclos, ya que se contaría con una fracción de la información total para detectar el cierre de ciclo.

El valor de este umbral para agregar un nuevo fotograma se determinó de forma experimental. Este umbral depende y debe ser menor que el número de características que se puedan extraer con el algoritmo de detección de características. En el caso de SURF, se debe determinar un buen valor para la *hessiana*, para que se puedan extraer un buen número de características. Si se escoge un valor muy alto, se detectarán muchos menos puntos de características aunque serán más robustos, si se utiliza un valor pequeño, se detectarán muchos más puntos de características, pero serán menos robustos. También debe tenerse en cuenta que el número de características que se pueden extraer cambia de un algoritmo de extracción a otro. Otro factor que influye para determinar el valor de este umbral es el ambiente donde se extraen las características, ya que existen escenarios que no son ricos en características y por lo tanto no se extraerán el mismo número que en otros casos.

Se utiliza un segundo umbral en la implemetación, el cual se utiliza para determinar si se ha detectado un cierre de ciclos. Determinar el valor de este umbral también depende de los factores mencionados anteriormente. Sin embargo, determinar un valor adecuado para este parámetro es de suma importancia para el desempeño de esta implemetación

de SLAM. Ya que se debe evitar cometer un gran número de errores en la detección de cierre de ciclos y de esta manera evitar un gran número de falsos positivos y/o falsos negativos. Escoger un valor inapropiado para este umbral puede provocar que se creen restricciones donde no existen, lo cual provocará mapas inconsistentes cuando se realice la optimización del grafo. Lo mismo pasa cuando no se detectan cierres de ciclos que ayudarían a reducir en gran medida el error acumulado, aunque esto no es tan grave como agregar una restricción donde no existe.

7.3. Optimización del grafo

Una vez que el grafo se ha inicializado con las poses y las restricciones de los cierres de ciclos, el grafo puede ser optimizado. Varios métodos están disponibles para el proceso de optimización. El método utilizado este trabajo es el de "Levenberg-Marquardt" con un solucionador CHOLMOD lineal. La optimización se realiza con un número predefinido de pasos. Esto se implementó mediante las librerías de MRPT. Al optimizar el grafo de poses se corrigen los vértices y se crean nuevas estimaciones de las poses de la cámara, esto se ilustra en la imagen 7.3.

7.4. Reconstrucción de la escena

Por último, una vez que las poses de cámara se determinan con una buena certeza, la reconstrucción de la escena (mapa 3D) se puede realizar. Para cada pose, una nube de puntos 3D se genera a partir de los datos RGB y profundidad. Con el fin de reconstruir toda el mapa 3D, se combinan todas las nubes de puntos desde un punto de vista único. Este proceso se llama "registro de nubes de puntos", como se explicó en el capítulo 4. Una vez que se conocen las posiciones de la cámara, cada nube de puntos se transforma mediante la proyección de todos sus puntos de acuerdo con la posición de la cámara correspondiente. Las nubes de puntos transformadas entonces se concatenan juntas para construir el mapa 3D.

Este método es simple, los principales problemas que presenta es que lleva a duplicar los puntos, por lo que puede consumir muchos recursos de memoria, además, no toma las variaciones de iluminación en cuenta. Algunas consideraciones sobre la memoria se describen en los experimentos en el capítulo 8. Este mapa 3D no es necesariamente la mejor representación de la escena, para muchas aplicaciones, sería preferible definir algunas superficies densas, como en los trabajos [97][79], pero esto requeriría un mayor estudio y procesamiento y está fuera del alcance de este trabajo.

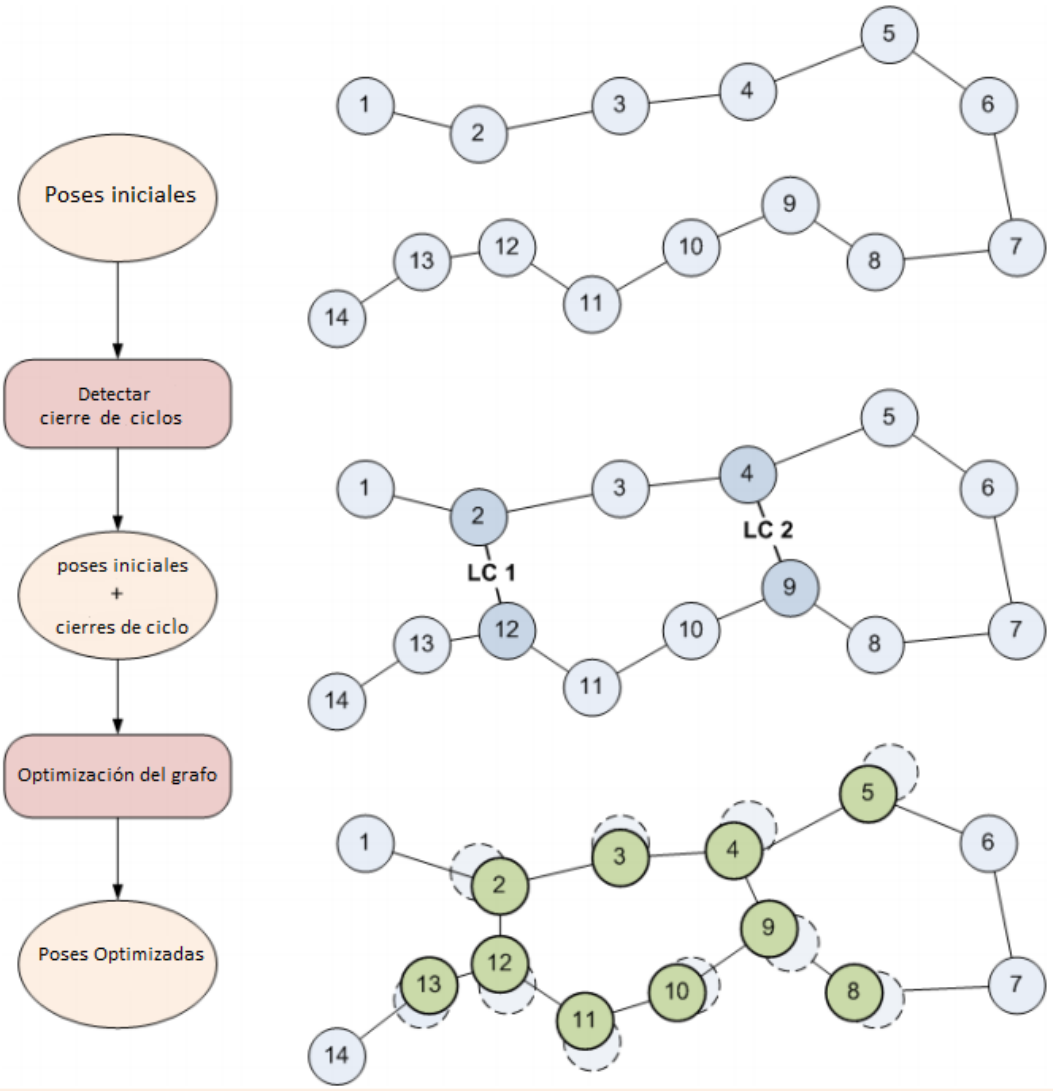


FIGURA 7.3: La figura ilustra los pases seguidos en la detección de cierre de ciclos

Capítulo 8

Eliminación de datos atípicos 2D

En este capítulo se presenta la etapa de eliminación de datos atípicos 2D mediante el cómputo de la matriz de homografía. En este capítulo se motiva la utilización de este método de eliminación de datos atípicos que son resultado del emparejamiento de características mediante arboles *kd*. Más adelante se explica el cálculo de la matriz de homografía.

8.1. El motivo de eliminar datos atípicos

En la capítulo 7 se explicó el enfoque que se siguió en la detección de cierre de ciclos en este trabajo, el cual consiste en medir el número de datos típicos resultantes de realizar el emparejamiento de características visuales mediante RANSAC. Realizar la detección de bucles basándose únicamente en el número de emparejamientos tiene una ventaja y un gran inconveniente. La principal ventaja es que es sencillo de implementar. Sin embargo, el gran inconveniente es que aparecen un gran número de falsos positivos a la hora de detectar bucles. Es decir, se detecta el paso por lugares visitados anteriormente incluso cuando no se ha pasado por un lugar conocido. La idea de eliminar datos atípicos 2D es de reducir este número de falsos positivos.

La matriz de homografía H relaciona puntos de una superficie plana proyectados sobre dos imágenes $x_i \leftrightarrow x_i^{'}$. Esta relación indica la distancia del punto Hx_i al punto $x_i^{'}$. Se pueden usar esta relación para eliminar los datos atípicos. Para eliminar datos atípicos resultantes del emparejamiento de características 2D, se consideran datos típicos a las correspondencias $x_i \leftrightarrow x_i^{'}$ cuya distancia al punto transformado $H_x i$ es menor que un cierto umbral. En la figura 8.1 se muestra el criterio para determinar las correspondencias de puntos 2D que se consideran datos típicos o atípicos.

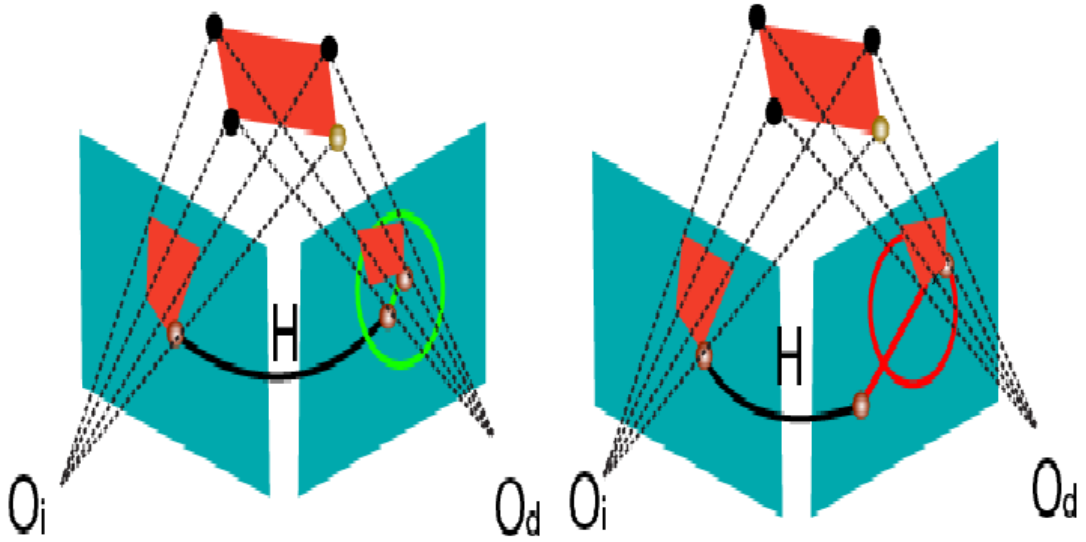


FIGURA 8.1: Eliminación de datos atípicos 2D con la matriz de homografía se basa en la distancia de $x^{'}$ a la correspondencia transformada $x_t = Hx$. Izquierda: la distancia de la correspondencia $x^{'}$ de x al punto transformado $x_t = Hx$ es menor de un cierto umbral, por lo que se considera dato típico. Derecha: la distancia de $x^{'}$ al punto transformado $x_t = Hx$ es mayor que un cierto umbral, por lo que se considera dato atípico.

8.2. La matriz de homografía

En el campo de la visión por computador, dos imágenes de una misma superficie plana están relacionadas por una homografía si se asume que se está usando el modelo de una cámara "pin-hole". De esta forma, un punto de una superficie proyectado sobre la imagen A tiene coordenadas $p_a^T = (x_a y_a 1)^T$. El mismo punto proyectado sobre la imagen B tiene coordenadas $p_b^T = (x_b y_b 1)^T = p_b^{'T}/w^{'}$. La matriz H relaciona las coordenadas de las proyecciones del punto sobre A y B de la siguiente forma:

$$P_a = \begin{pmatrix} x_a \\ y_a \\ 1 \end{pmatrix} P_b^{'} = \begin{pmatrix} w^{'}x_b \\ w^{'}y_b \\ w^{'} \end{pmatrix} H^{'} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} P_b^{'} = H \cdot P_a \tag{8.1}$$

$$p_b = p_b^{'}/w^{'} \tag{8.2}$$

8.3. Cálculo de la matriz de homografía

En esta sección se explica un método para obtener la matriz de homografía que mejor transforma las coordenadas de cuatro puntos de una imagen en la otra. Para ello se parte de cuatro pares de correspondencias 2D $x_i \leftrightarrow x_i^{'}$ (puntos en P^2 y por tanto

vectores homogéneos 3×1). Por lo que se puede encontrar la homografía H de tamaño 3×3 tal que:

$$x_i^{'} = H \cdot x_i \tag{8.3}$$

Los vectores $x_i^{'}$ y $H \cdot x_i$ no son numéricamente idénticos y pueden diferir en un factor de escala. Sin embargo, tienen la misma dirección y por tanto $x_i^{'} \times H \cdot x_i = 0$.

Escribiendo la j-ésima fila de H como h^{jT}, se tiene que:

$$H \cdot x_i = \begin{pmatrix} h^{1T} x_i \\ h^{2T} x_i \\ h^{3T} x_i \end{pmatrix} \tag{8.4}$$

Escribiendo $x_i^{'T} = (x_i^{'} y_i^{'} w_i^{'})$, el producto vectorial queda:

$$x_i^{'} \times H \cdot x_i = \begin{pmatrix} y_i^{'} h^{3T} x_i - w_i^{'} h^{2T} x_i \\ y_i^{'} h^{1T} x_i - w_i^{'} h^{3T} x_i \\ y_i^{'} h^{2T} x_i - w_i^{'} h^{12T} x_i \end{pmatrix} \tag{8.5}$$

Como $h^{jT} x_i = x_i^T h^j$, el sistema de ecuaciones $x_i^{'} \times H \cdot x_i = 0$ se puede escribir en términos de las entradas de H como:

$$A_i \cdot h = \begin{pmatrix} 0^T & -w_i^{'} x_i^T & y_i^{'} x_i^T \\ w_i^{'} x_i^T & 0^T & -x_i^{'} x_i^T \\ -y_i^{'} x_i^T & x_i^{'} x_i^T & 0^T \end{pmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0 \tag{8.6}$$

Donde A_i es una matriz 3×9 y h es un vector columna 9×1 con los elementos de H. En la ecuación anterior, la tercera fila de A_i se puede obtener sumando $x_i^{'}$ veces la primera fila e $y_i^{'}$ veces la segunda, por lo que el rango de A_i es dos. Aunque se podrían usar únicamente dos ecuaciones, se usan las tres porque si $w_i^{'} = 0$, las dos primeras ecuaciones se convierten en una. De esta forma, apilando las ecuaciones para cuatro pares de puntos se tiene:

$$Ah = 0 \tag{8.7}$$

Donde A es una matriz de 12×9 de rango 8. H está definida a una determinada escala, por lo que se escoge una arbitrariamente haciendo que $||h|| = 1$. El sistema sobredeterminado $Ah = 0$ puede no tener solución, por lo que se resuelve minimizando $||Ah||$ sujeto a que $||h|| = 1$. La restricción $||h|| = 1$ se impone para evitar la solución $h = 0$.

El algoritmo anterior permite obtener la matriz de homografía para cuatro pares de puntos $x_i \leftrightarrow x_i^{'}$. Sin embargo, en la práctica se dispone generalmente un número mayor de emparejamientos, por lo que surge la pregunta sobre cómo determinar la matriz de homografía que mejor transforma los puntos de una imagen en la otra.

El procedimiento se puede resumir de la siguiente forma:

- Escoger un conjunto de cuatro emparejamientos $x_i \leftrightarrow x_i^{'}$
- Computar la matriz de homografía con el método descrito anteriormente.
- Computar la bondad de la matriz de homografía estimada.

Este proceso se repite con varios conjuntos de emparejamientos y se selecciona la mejor H como estimación inicial. Este modelo también se pude obtener mediante RANSAC.

Capítulo 9

Resultados experimentales y discusión

En este capítulo se expondrán y discutirán los resultados obtenido el medir el desempeño de la solución implementada para el problema de SLAM. Hasta este momento se ha explicado cada una de las partes que componen la solución propuesta al problema del SLAM, sin embargo, no se ha demostrado ningún resultado que muestre el comportamiento del sistema en la práctica, en este capítulo se pretender medir el desempeño del sistema desarrollado mediante el uso de un benchmark, además se discute los resultados obtenidos.

9.1. Parámetros del sistema

Hay numerosos parámetros que pueden ser ajustados para dar resultados ligeramente mejores en relación con cada entorno. En el caso de las secuencias utilizadas, no se cambió los valores de los parámetros, esto tiene como idea mostrar la robustez en diferentes escenarios. Los valores para los parámetros más importantes se enlistan a continuación:

- Hessiana en SURF: 200
- Máximo número de iteraciones en RANSAC: 100
- Máxima distancia para ser correspondencia en RANSAC: 0.015 metros
- Máxima distancia para ser correspondencia en ICP : 0.3 metros
- Máximo número de iteraciones de ICP: 5

- Máxima distancia para la convergencia de ICP: 0.01 metros
- Número de correspondencias para nuevo fotograma clave: 100
- Número de correspondencias para cierre de ciclo: 50
- Máxima distancia de correspondencia en matriz de homografía: 0.15 metros

9.2. Visualización de los resultados

Como resultado, el programa exporta las posiciones de la cámara y el mapa representado por una nube de puntos, aunque este no es la mejor representación de un mapa 3D, es la representación con la que se trabaja en esta investigación. Por razones de rendimiento, es necesario controlar el uso de la memoria RAM, esto se logra al limitar el tamaño de la nube de puntos final. Teóricamente, cada nube de puntos puede estar compuesta de $640 \times 480 = 307,200$ puntos. Prácticamente, el número efectivo de puntos es inferior, ya que la información de profundidad no está disponible para cada punto (principalmente debido al límite del rango y por las oclusiones). Para cada pixel, 24 bits se utilizan para las posiciones (x, y, z) y 24 bits para los colores R, G, B (3×8 bits). Por lo tanto, el tamaño de una nube esta dado por $640 \times 480 \times 8 \times sizeof(float) = 9,830,400$ bytes, que son alrededor de 9.8 megabytes, aunque esta cantidad no puede parecer mucho en un principio, debe considerarse que es solo por una nube de puntos y se trabaja con múltiples nubes en memoria al vez, además que se guarda información adicional para cada nube de puntos. Por motivos de rendimiento y recursos limitados de memoria, es razonable reducir la cantidad de información. Cada nube de puntos es submuestreada haciendo uso de voxeles, que son la unidad cúbica que componen un objeto 3D, por tanto, son equivalente del los pixeles en un objeto 2D.

9.3. Evaluación del sistema

En esta sección se medirá el desempeño de sistema mediante el uso de un benchmark, el cual ofrecerá un punto de referencia completa que se utiliza para evaluar el sistema desarrollado de SLAM en este trabajo. El sistema se evaluará bajo diferentes situaciones como: imágenes borrosas causado por movimientos rápidos de la cámara, con pocas características, imágenes sin textura y mapas grandes. Se utilizan 4 secuencias del benchmark los cuales contienen estas situaciones.

Se presentarán los resultados obtenidos al aplicar la pura odometría visual, como también los resultados después de aplicar la optimización del grafo de poses. De esta manera se

espera apreciar la mejoría que se obtiene al aplicar el algoritmo de SLAM basado en grafo.

9.3.1. Benchmark utilizado

Los benchmarks apoyan en gran medida la evaluación científica y comparación objetiva de los algoritmos y su desempeño. Varios ejemplos de benchmarks de éxito en el área de visión por ordenador han demostrado que estos benchmarks y las métricas claras para evaluación pueden ayudar significativamente a empujar el estado del arte. Es por ello que se utilizará el benchmark propuesto por [98]. Este conjunto de datos ofrece un punto de referencia completa que se puede utilizar para evaluar los sistemas de SLAM y la odometría visual sobre datos RGB-D. Además, en [98] proponen también dos métricas de evaluación para su benchmark y proporcionar herramientas para su fácil evaluación.

El benchmark consta de 39 secuencias, las cuales se grabaron en dos ambientes de interiores diferentes. Cada secuencia contiene imágenes de color y profundidad, así como la trayectoria verdadera. El formato del la trayectoria verdadera es el siguiente: en cada línea en el archivo el formato es $(T, t_x, t_y, t_z, q_x, q_y, q_z, q_w)$. T es la marca de tiempo, (t_x, t_y, t_z) da la posición del centro óptico de la cámara con respecto al origen del mundo. y (q_x, q_y, q_z, q_w) dan la orientación del centro óptico de la cámara, estos dos últimos forma la unidad conocida cuaternión. Los cuaterniones proporcionan una notación matemática para representar las orientaciones y las rotaciones de objetos en tres dimensiones.

Las secuencias se proporcionan como pares de imágenes de colores y de profundidad, junto con sus respectivas marcas de tiempo, que indican el momento de su medición. Las imágenes en color se almacenan como imágenes RGB de 640×480 de 8 bits en formato PNG. Las imágenes de profundidad se almacenan como imágenes 640×480 de 16 bits monocromáticas en formato PNG.

A continuación se explican las métricas propuestas en este benchmark:

Relative pose error measures (RPE): mide la precisión local de la trayectoria durante un intervalo de tiempo fijo $\triangle$. Por lo tanto, corresponde a la deriva de la trayectoria y es particularmente útil para la evaluación de sistemas de odometría visuales. Se define la métrica en el paso del tiempo i como.

$$E_i := (Q_i^{-1} Q_i + \triangle)^{-1} (P_i^{-1} P_i + \triangle) \tag{9.1}$$

A partir de una secuencia de poses n de la cámara, se obtiene de esta manera $m = n - \triangle$ errores de pose relativos individuales a lo largo de la secuencia. A partir de estos errores,

los autores proponen "root mean squared error" (RMSE), sobre todos los índices del tiempo del componente traslacional como:

$$RMSE(E_{1:n}, \triangle) := (\frac{1}{m}\sum_{i=1}^{m} ||trans(E_i)||^2)^{1/2} \tag{9.2}$$

Donde $trans(E_i)$ se refiere a los componentes de traslación del erro relativo a la pose E_i.

Absolute trajectory error (ATE): para los sistemas de SLAM visual, la consistencia global de la trayectoria estimada es una cantidad importante. La consistencia global puede ser evaluada mediante la comparación de las distancias absolutas entre la trayectoria estimada y la trayectoria del la trayectoria verdadera, el error absoluto de la trayectoria en paso i del tiempo se calcula como:

$$F_i := Q_i^{-1} S P_i \tag{9.3}$$

Al igual que el error relativo de la pose(RSE), se evalúa el error cuadrático medio sobre todos los índices en el tiempo de los componentes de traslación, es decir:

$$RMSE(F_{1:n}, \triangle) := (\frac{1}{n}\sum_{i=1}^{n} ||trans(F_i)||^2)^{1/2} \tag{9.4}$$

9.3.2. Secuencia 1

Se probó el sistema con la secuencia $rawlog\_rgb\_dataset\_freiburg1\_xyz$ del benchmark. De la cual se puede ver una muestra de las imágenes en la tabla 9.1. De esta secuencia se puede crear un mapa 3D consistente fácilmente, ya que no cuenta con algo que provoque una mala reconstrucción. La cámara se mueve lentamente mayormente por por los ejes (x, y), pasando por los mismos lugares en repetidas ocasiones, lo que crea varios cierres de ciclos. El mapa es de tamaño pequeño y se pueden detectar suficientes características en cada imagen.

Cuadro 9.1: Muestra de imágenes de la secuencia 1.

Los resultados de la evaluación del sistema para esta secuencia se pueden observar en la tabla 9.2. Se puede apreciar en ellos que el error no es muy grande en cada métrica, por lo que se puede considerar que se pudo conseguir una estimación de la trayectoria adecuada y una reconstrucción del mapa consistente. El error después de la optmización disminuye, como se esperaba.

rawlog_rgb_dataset_freiburg1_xyz	
ATE	0.074298
ATE (Optimizado)	0.014862
RPE	0.311291903173
RPE (Optimizado)	0.29708478285

CUADRO 9.2: Los resultados de las métricas ATE y RPE para la secuencia 1.

A continuación se muestra el resultado de graficar la métrica ATE, la cual se puede apreciar en las figuras 9.2 y 9.1. En la gráfica 9.1 se puede ver que el error es notable en el cálculo de la trayectoria estimada. Después de realizar la optimización del grafo se puede apreciar como el error disminuye y que la trayectoria estimada es muy parecida a la verdadera.

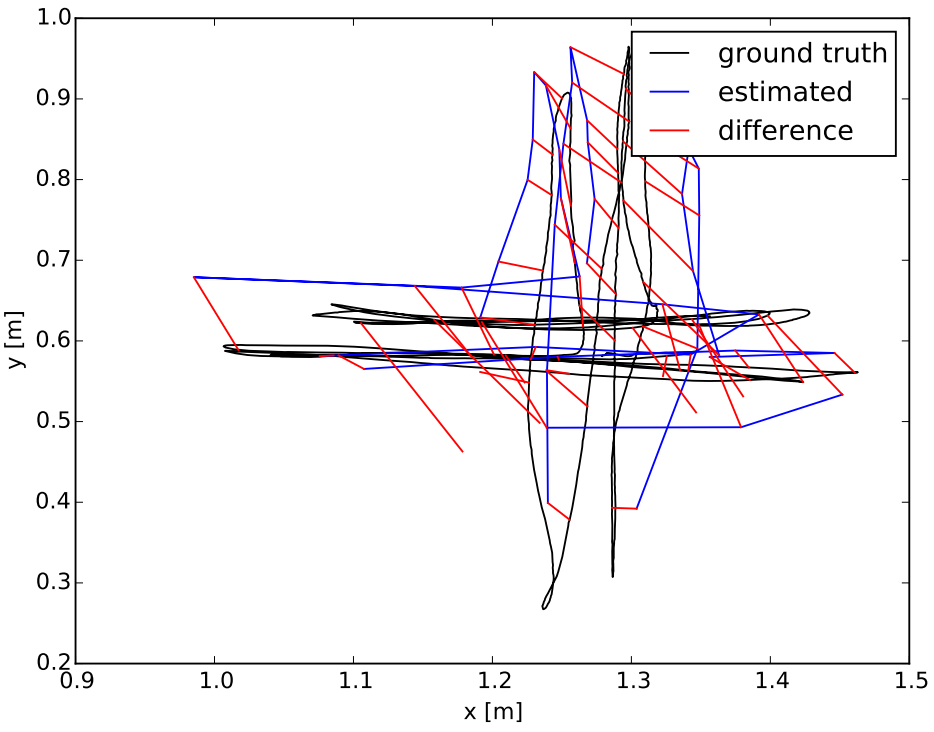


FIGURA 9.1: Diferencia entre la trayectoria estimada y la verdadera de la secuencia 1.

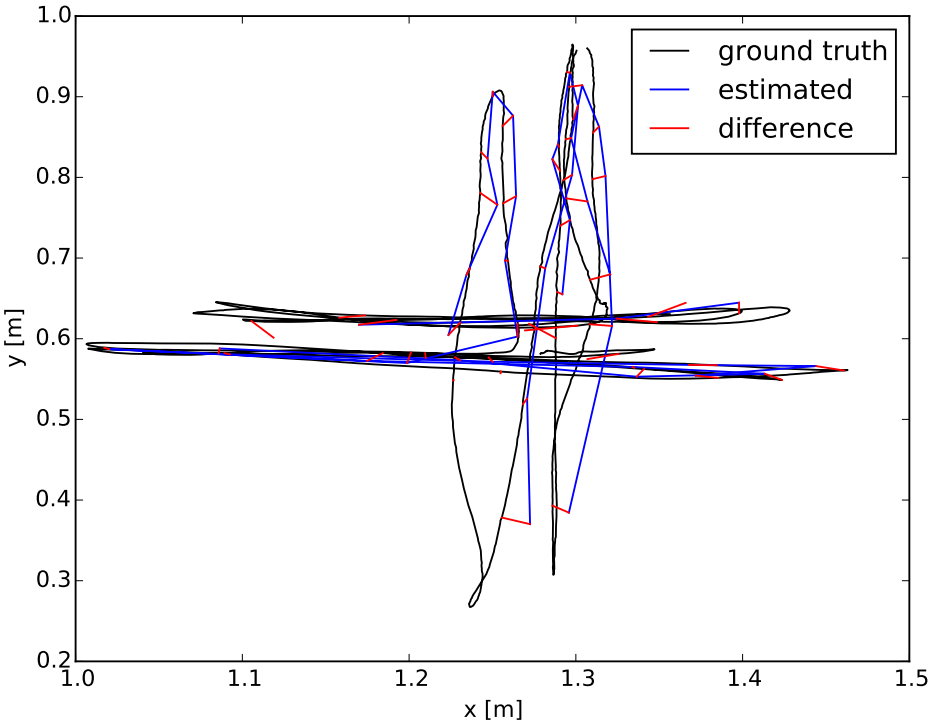


FIGURA 9.2: Diferencia entre la trayectoria estimada optimizada y la verdadera de la secuencia 1.

En las figuras 9.4 y 9.3 se muestran los mapas construidos. El primero es el mapa construido con solo la odometría visual y el segundo el obtenido al optimizar el grafo de poses. Se pueden apreciar varias inconsistencias marcadas en círculos rojos en el mapa sin optimizar, figura 9.2. En uno de los círculos se puede apreciar que aparece varias veces la misma silla, en el otro círculo se aprecia un libro en 2 lugares a la vez, esto da muestra que la acumulación del error es lo suficientemente importante como para producir estas inconsistencias. Como se esperaba, el error crece rápidamente aún para este mapa pequeño. En la figura 9.1 se puede ver que las inconsistencias anteriores han sido eliminadas una vez el mapa ha sido optimizado, lo cual ayuda obtener una mejor aproximación de la trayectoria y un mapa más consistente.

FIGURA 9.3: Mapa 3D construido de la secuencia 1. Los círculos rojos señalan inconsistencias visibles en el mapa.

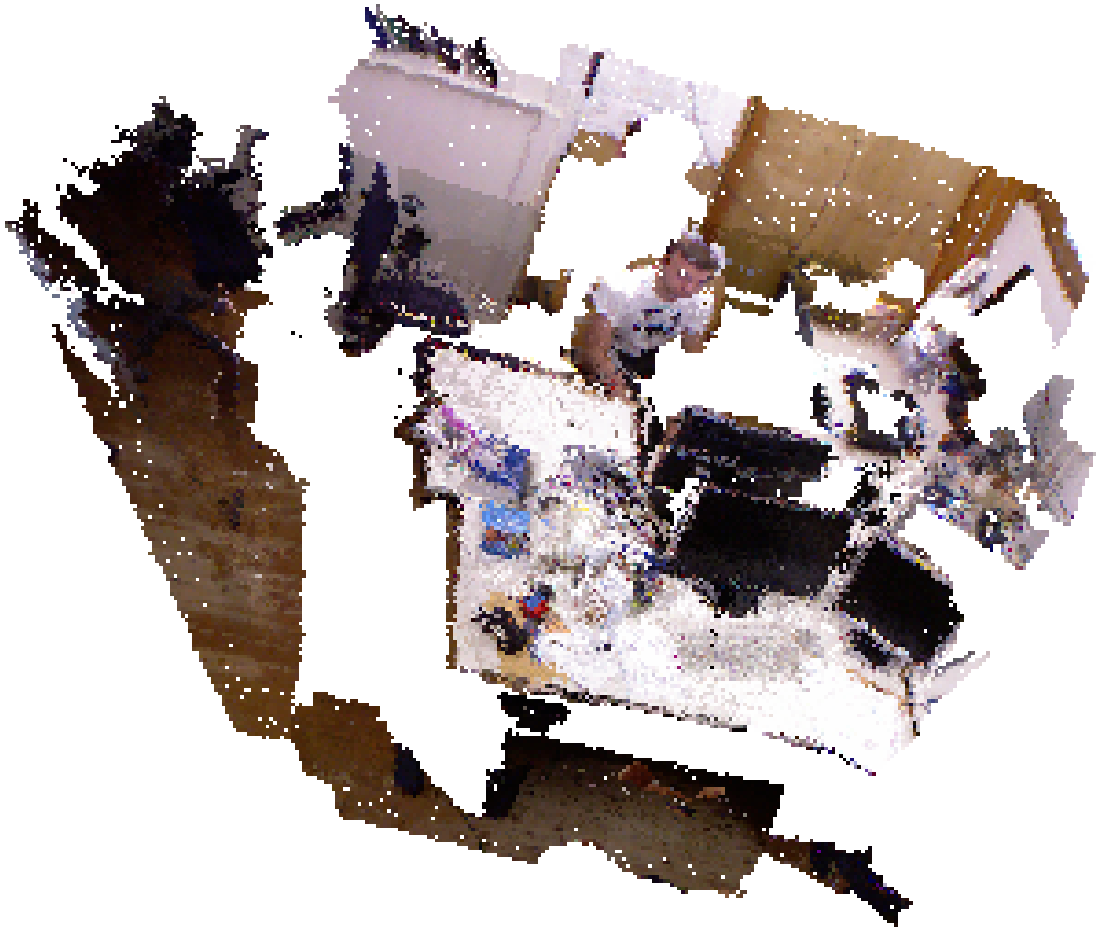


FIGURA 9.4: Mapa 3D optimizado construido de la secuencia 1.

En las figuras 9.5 se presenta el grafo de poses para esta secuencia. Se puede apreciar que existe una gran interconectividad en el grafo, ya que los vértices se conectan mediante aristas a varios vértices distintos, lo cual indica que se detectó el paso por un mismo lugar (cierre de ciclos) y se crearon estas aristas que relacionan las poses y establecen restricciones espaciales entre ellas, la cual ayuda a obtener un mejor resultado después de optimizar el grafo.

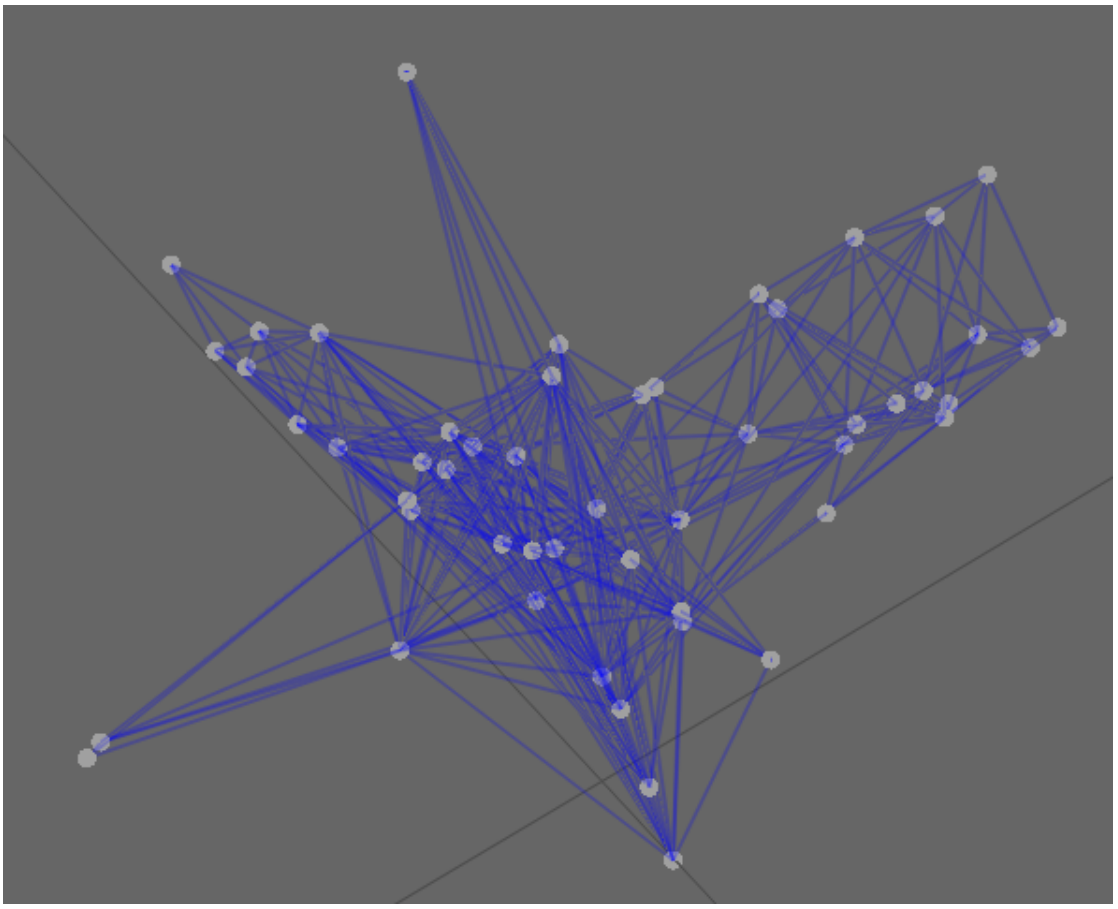


FIGURA 9.5: Cada punto en blanco representa un vértice en el grafo, que a su vez representa una pose. Las líneas azules representan vértices en el grafo, que son una representación de las restricciones espaciales entre cada pose.

9.3.3. Secuencia 2

Se probó el sistema con la secuencia $rawlog\_rgb\_dataset\_freiburg1\_desk2$ del benchmark, de la cual se puede ver una muestra de las imágenes en la tabla 9.3. La secuencia tiene la propiedad de tener muchas imágenes borrosas causado por movimientos rápidos del sensor Kinect. Además, pasa varias veces por los mismos lugares, por lo que se esperaría detectar cierres de ciclos. Los movimientos rápidos del sensor provocan que no se puede extrae puntos de características, lo que hace fallar al extractor de características. Sin características en un fotograma, no se puede encontrar correspondencias entre otros fotogramas pasados. Por lo que no se podrá realizar un emparejamiento entre los descriptores de las características entre dos fotogramas, ni encontrará una buena transformación. Por este motivo, en esta secuencia, el número de emparejamientos correctos entre pares de imágenes consecutivas después de aplicar RASNAC es mucho menor en comparación que en la secuencia 1. Todo esto provoca que se tengan estimaciones más pobres de la trayectoria y un mapa 3D menos consistente.

Cuadro 9.3: Muestra de imágenes de la secuencia 2.

Los resultados de la evaluación del sistema para esta secuencia se pueden observar en la tabla 9.4. Se puede apreciar en los resultados obtenidos que el error no es muy bueno, por lo que se puede considerar que no se pudo conseguir una estimación de la trayectoria tan adecuada y tampoco una reconstrucción del mapa 3D muy consistente. Aunque el error después de la optmización disminuye, no es suficiente como para reducirlo significativamente.

rawlog_rgb_dataset_freiburg1_xyz	
ATE	0.286861
ATE (Optimizado)	0.183815
RPE	1.45149348232
RPE (Optimizado)	1.3436163866

CUADRO 9.4: Los resultados de las métricas ATE y RPE para la secuencia 2.

A continuación se muestra el resultado de graficar la métrica ATE, la cual se puede apreciar en las figuras 9.7 y 9.6. En la gráfica 9.6 se puede ver que el error es bastante notable en el cálculo de la trayectoria estimada. Después de realizar la optimización del grafo se puede apreciar en la figura 9.7 como el error disminuye y que la trayectoria estimada se acerca más a la verdadera. Sin embargo, la diferencia entre ellas es apreciable.

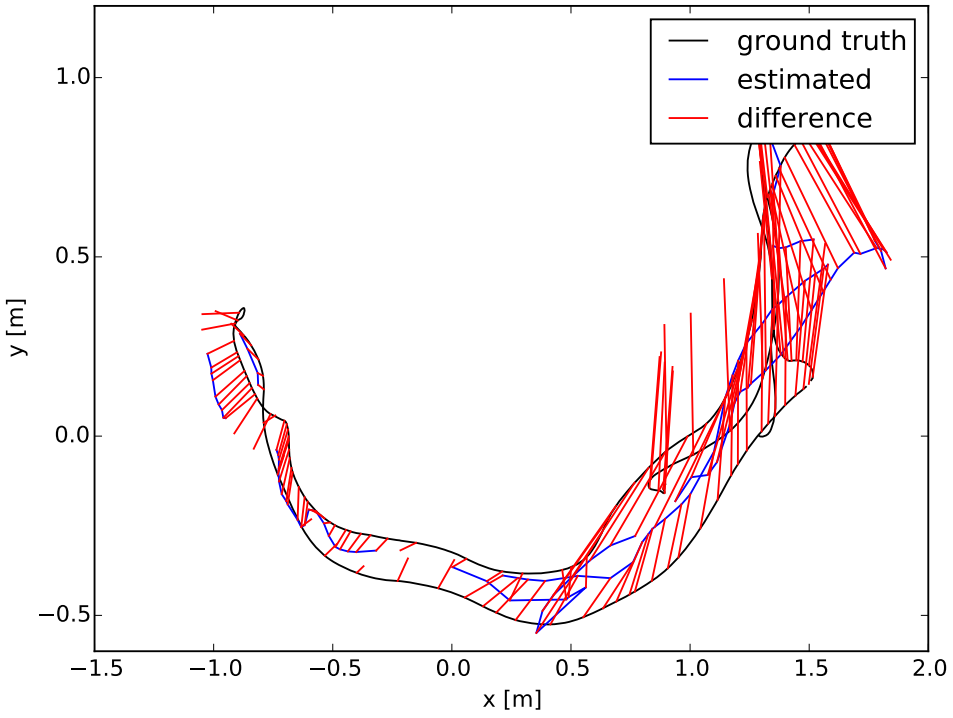


FIGURA 9.6: Diferencia entre la trayectoria estimada y la verdadera de la secuencia 2.

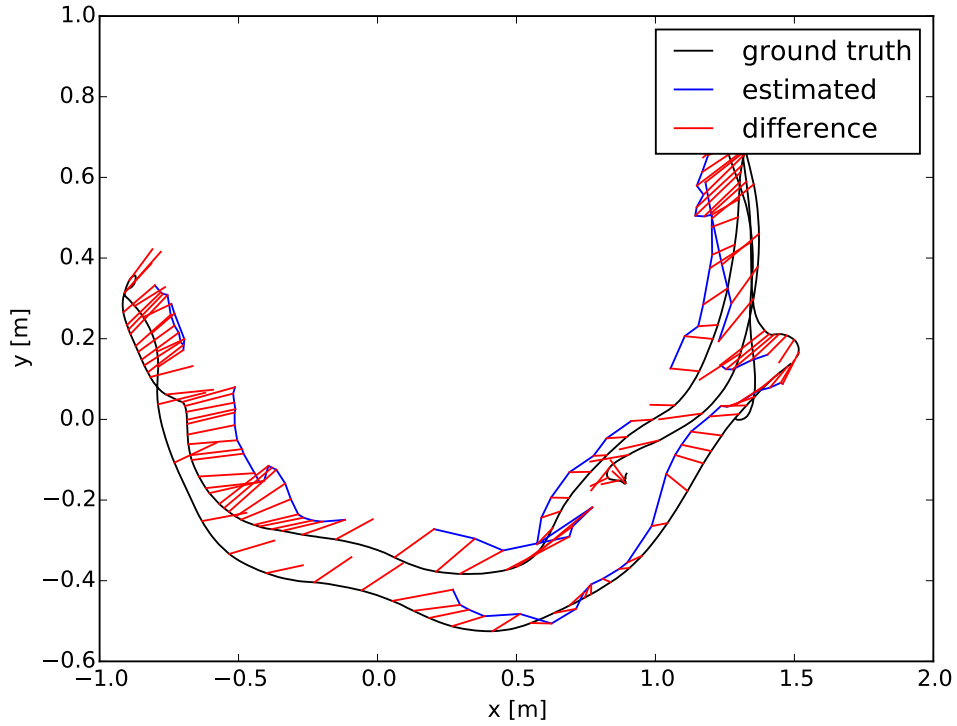


FIGURA 9.7: Diferencia entre la trayectoria estimada optimizada y la verdadera de la secuencia 2.

En las figuras 9.8 y 9.9 se muestran los mapas construidos. El primero es el mapa construido con solo la odometría visual, el segundo mapa es el resultado de optimizar el grafo de poses. Se pueden apreciar varias inconsistencias marcadas en círculos rojos en el mapa sin optimizar, figura 9.8. En uno de los círculos se puede apreciar que la mesa aparece en varios lugares a la vez, lo cual es muy notorio. En el otro círculo se aprecia que el piso está en una posición muy por arriba de la que se esperaría. Esto da muestra que la acumulación del error resultado de imágenes borrosas causado por movimientos rápidos de la cámara es lo suficientemente importante como para producir estas inconsistencias. El error crece rápidamente por que no es posible estimar una buena transformación entre poses debido a que no se pueden extraer puntos de características. En el mapa optimizado, figura 9.9 se puede ver que las inconsistencias han disminuido, pero que que aparecen nuevas inconsistencias. Se puede ver que el mapa construido después de la optimización es más consistente, aunque no lo suficiente para obtener un mapa 3D adecuado.

FIGURA 9.8: Mapa 3D construido de la secuencia 2. Los círculos rojos señalan inconsistencias visibles en el mapa.

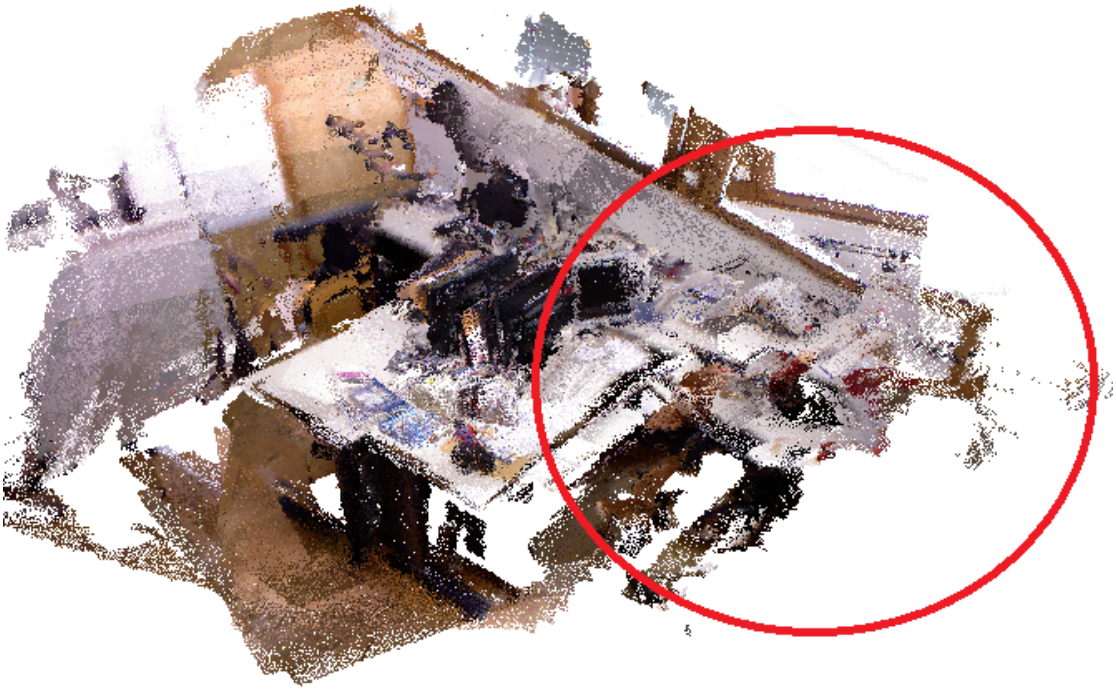


FIGURA 9.9: Mapa 3D optimizado construido de la secuencia 2. El círculo rojo señala un inconsistencia visible en el mapa.

En las figuras 9.10 se presenta el grafo de poses para esta secuencia. Se puede apreciar que no existe una gran interconectividad en el grafo, esto se debe a que no se pudo determinar muchos cierres de ciclos aún cuando la secuencia pasa por los mismos lugares en varias

ocasiones. Debido el problema de las imágenes borrosas, no se pueden determinar ciertos cierres de ciclos que ayuden a construir un mapa más consistente.

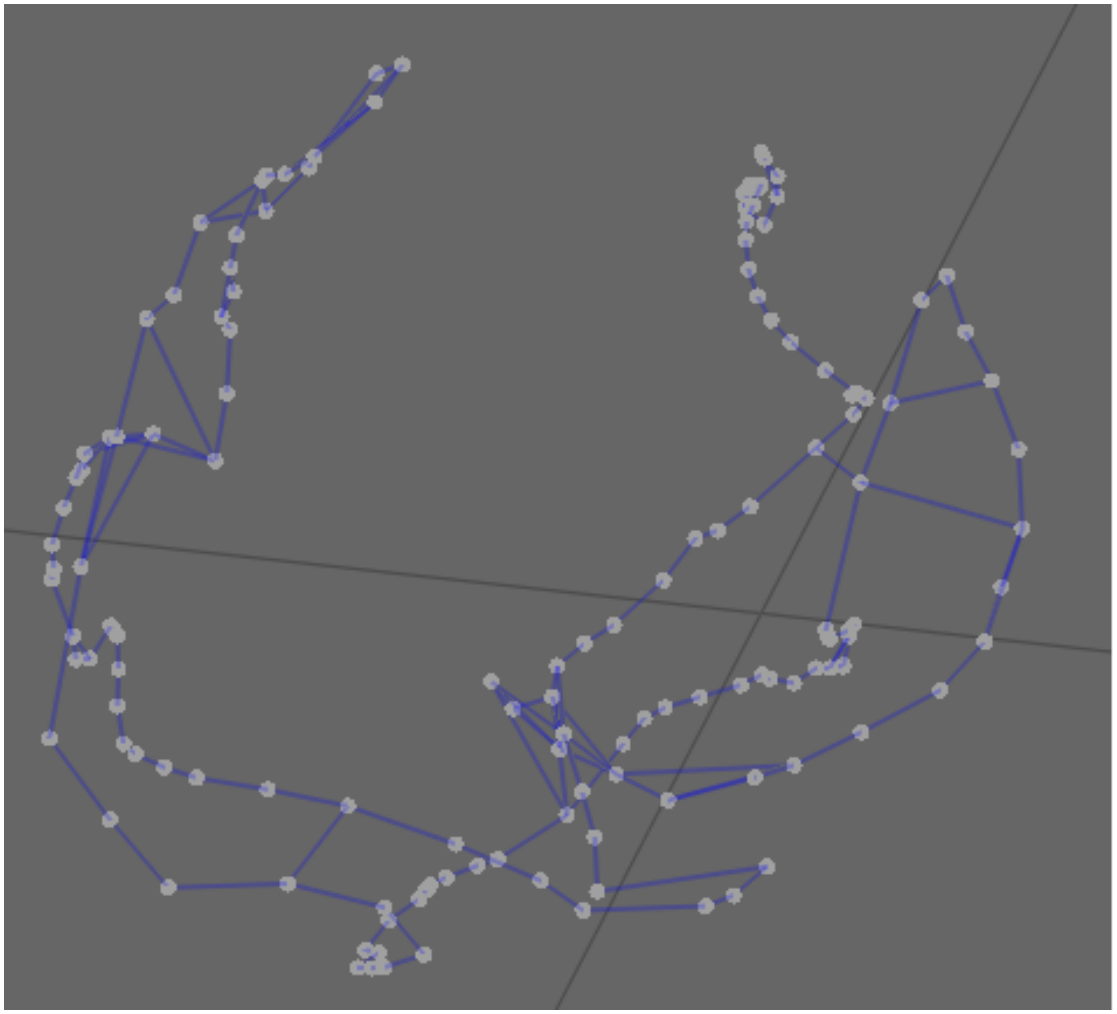
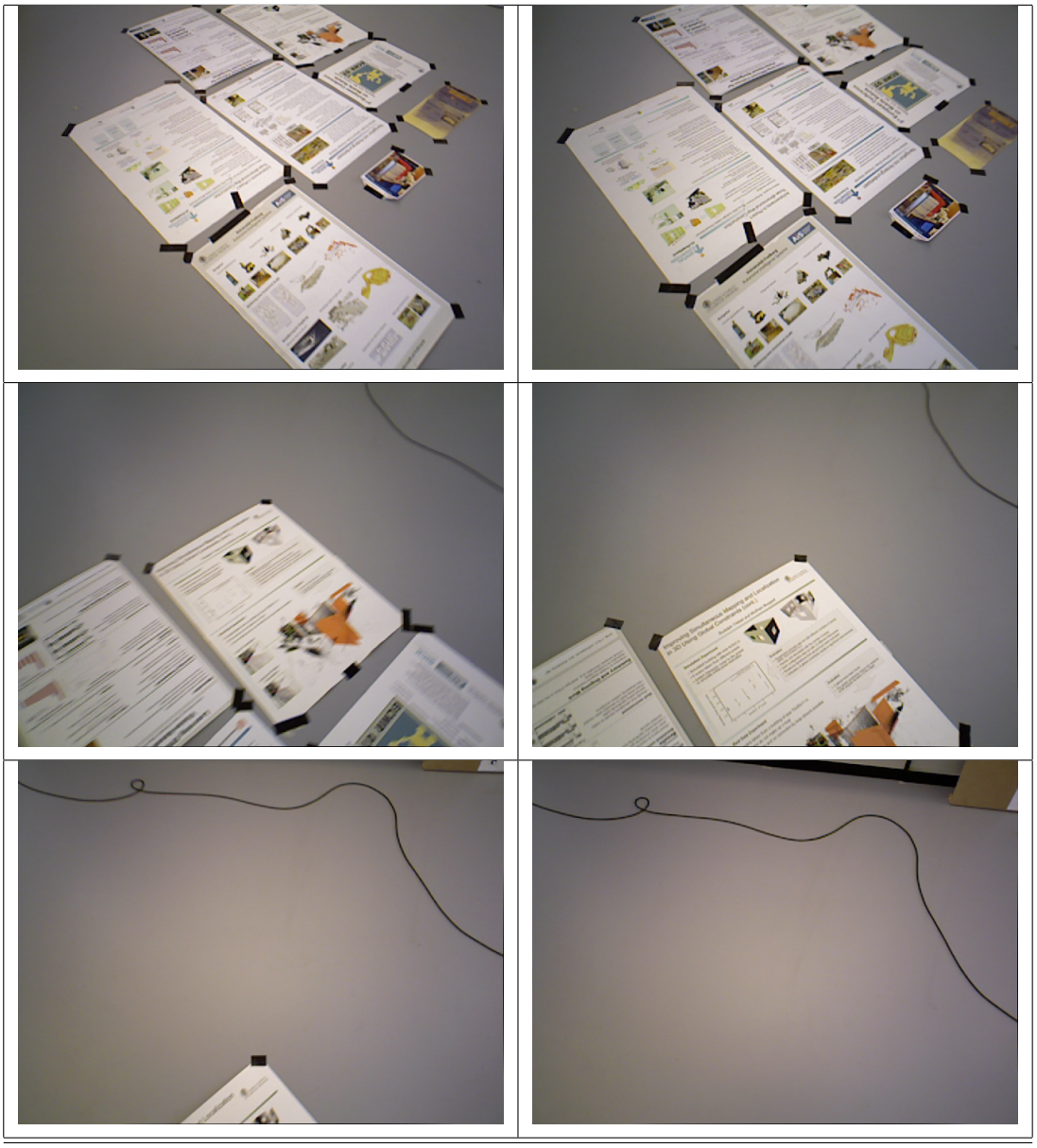


FIGURA 9.10: Grafo de poses de la secuencia 2. Cada punto en blanco representa un vértice en el grafo, que a su vez representa una pose. Las líneas azules representan vértices en el grafo, que son una representación de las restricciones espaciales entre cada vértice.

9.3.4. Secuencia 3

Se probó el sistema con la secuencia $rawlog\_rgb\_dataset\_freiburg3\_nostructure\_texture\_far$ del benchmark, de la cual se puede ver una muestra de las imágenes en la tabla 9.5. La secuencia tiene la propiedad de no tener estructura, solo son tomas de una pared. La secuencia aporta información de textura solo en una parte de la pared. El otro lado de la pared no contiene información de textura. Se pueden extraer puntos de características de la parte de la pared que tiene textura y hacer una buen reconstrucción. En la parte donde no se cuenta con textura no se puede extraer puntos de características y por

lo tanto no se puede hacer un bueno estimado de la trayectoria ni una reconstrucción adecuada del mapa 3D.



Cuadro 9.5: Muestra de imágenes de la secuencia 3.

Los resultados de la evaluación del sistema para esta secuencia se pueden observar en la tabla 9.6. Los resultados muestran que se tiene un error considerable en las métricas. Como se esperaba, el sistema se ve afectado por la falta de textura ya que no puede extraer características, tampoco puede estimar correspondencias entre las imágenes que si tienen textura y las que no. Esto se ve reflejando en resultados obtenidos.

rawlog_rgb_dataset_freiburg1_xyz	
ATE	0.635681
ATE (Optimizado)	0.465527
RPE	1.11341365485
RPE (Optimizado)	1.15395970128

CUADRO 9.6: Los resultados de las métricas ATE y RPE para la secuencia 3.

A continuación se muestra el resultado de graficar la métrica ATE, la cual se puede apreciar en la figuras 9.12 y 9.11. Se puede observar que el error es casi el mismo en las dos gráficas ya que la optimización no lo reduce en gran medida.

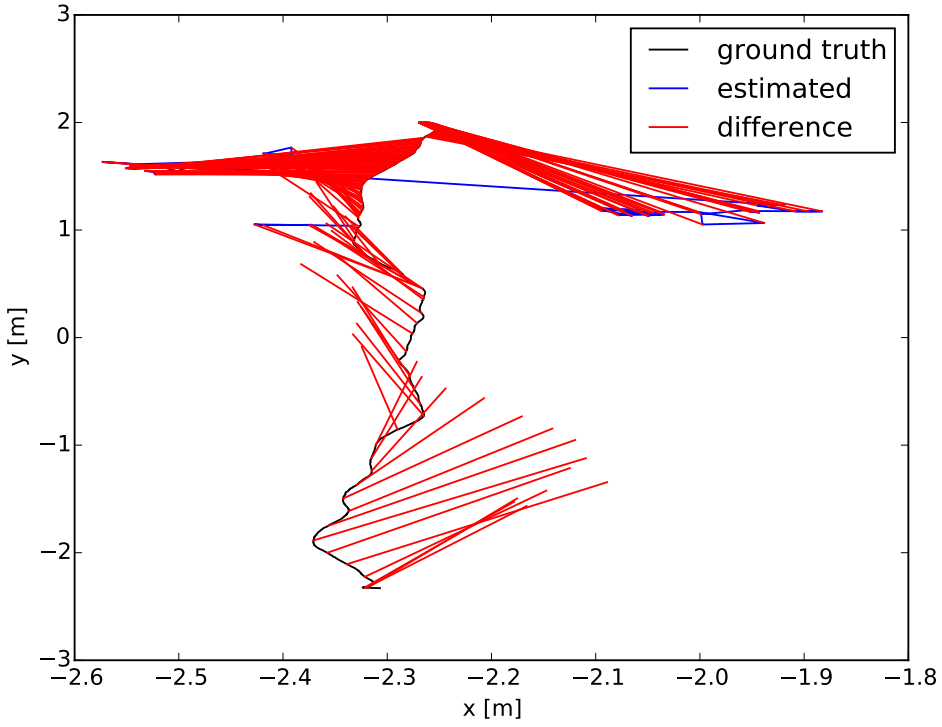


FIGURA 9.11: Diferencia entre la trayectoria estimada y la verdadera de la secuencia 3.

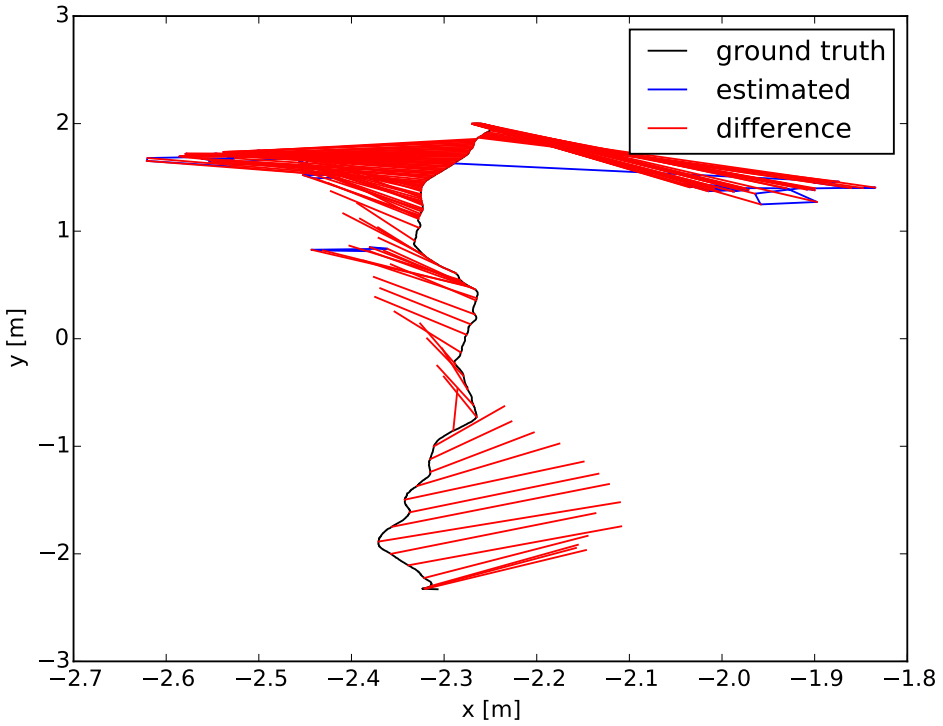


FIGURA 9.12: Diferencia entre la trayectoria estimada optimizada y la verdadera de la secuencia 3.

En las figuras 9.13 y 9.14 se muestran los mapas construidos. Se puede apreciar que los dos mapas construidos son casi el mismo o muy parecidos. Esto era de esperarse ya que en una secuencia tan sencilla la optimización del grafo no reduce mucho el error. El problema resulta de las imágenes sin textura ya que no se puede extraer características de estas y por lo tanto, no se puede calcular una transformación que relacione un par de fotogramas. Se puede observar que se puede crear una buena reconstrucción de la parte de la pared donde sí se cuenta con textura. Al llegar a la parte sin información de textura, es donde se comienza a tener problemas. Esas inconsistentes se ilustran en los círculos rojos en las figuras 9.13 y 9.14.

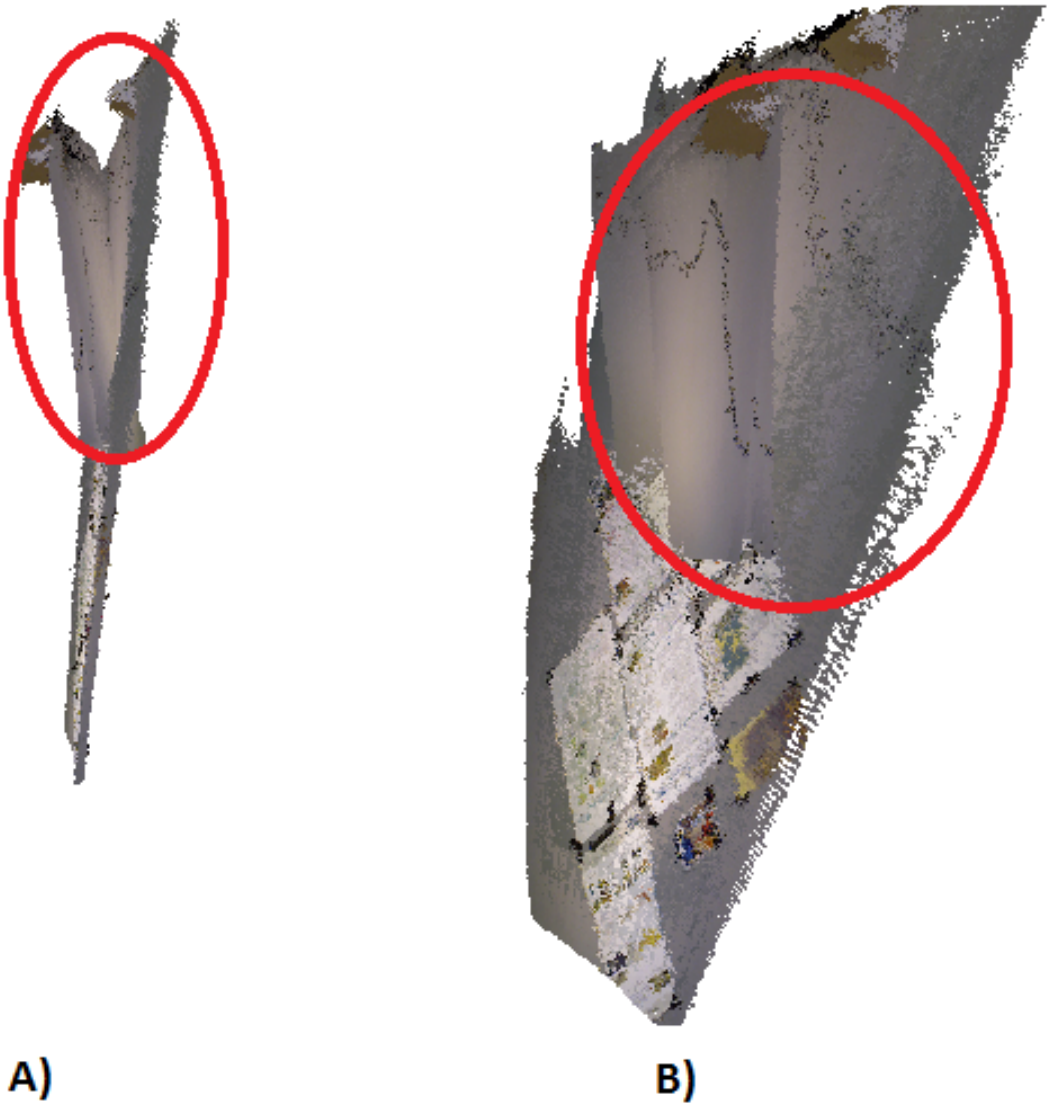


FIGURA 9.13: Mapa 3D construido de la secuencia 3. Los círculos rojos señalan inconsistencias visibles en el mapa.

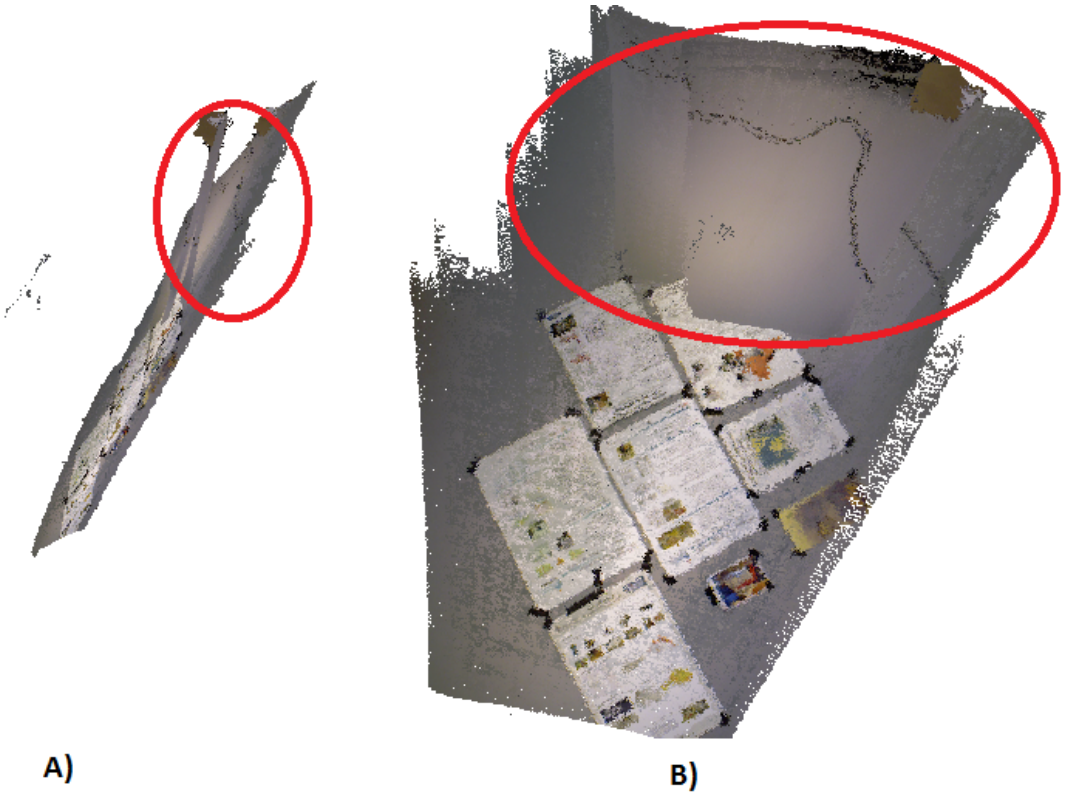


FIGURA 9.14: Mapa 3D optimizado construido de la secuencia 3. Los círculos rojos señalan inconsistencias visibles en el mapa.

En las figuras 9.15 se presenta el grafo de poses para esta secuencia. La mayoría de los vértices en la figura están interconectados. Estos vértices pertenecen a la parte donde se cuenta con información de textura. Los vértices que pertenecen a la parte sin textura están encerrados mediante un círculo rojo. Cada uno de estos vértices solo están conectados a su sucesor o antecesor mediante una sola arista. Esto se debe a que siempre se calcula una transformación que relaciona al último vértice y el que se acaba de agregar al grafo. Aún si no existe una transformación adecuada por falta de características. Esta es la razón de por que solo se tiene una arista que conecta esos pares de vértices y no existe ni un solo cierre de ciclos entre estos vértices.

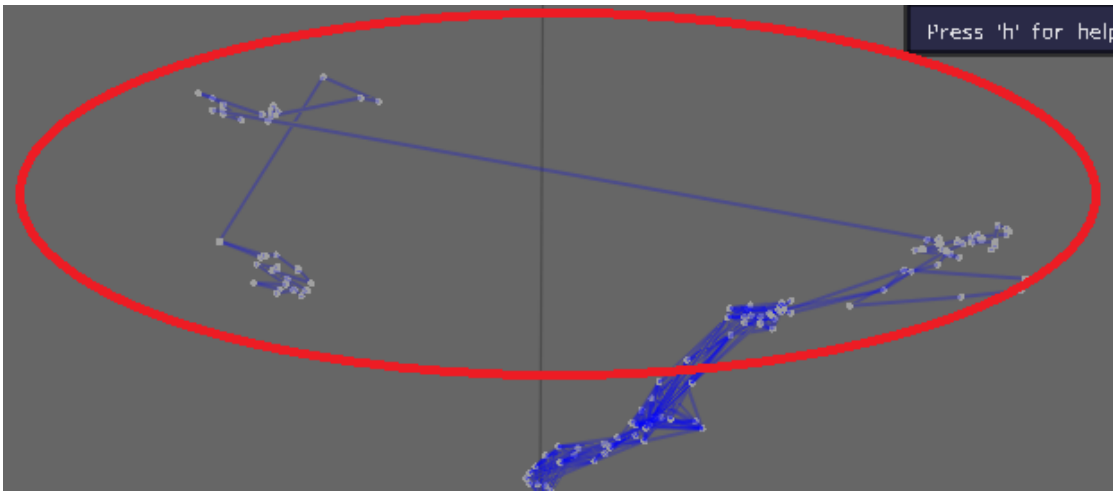
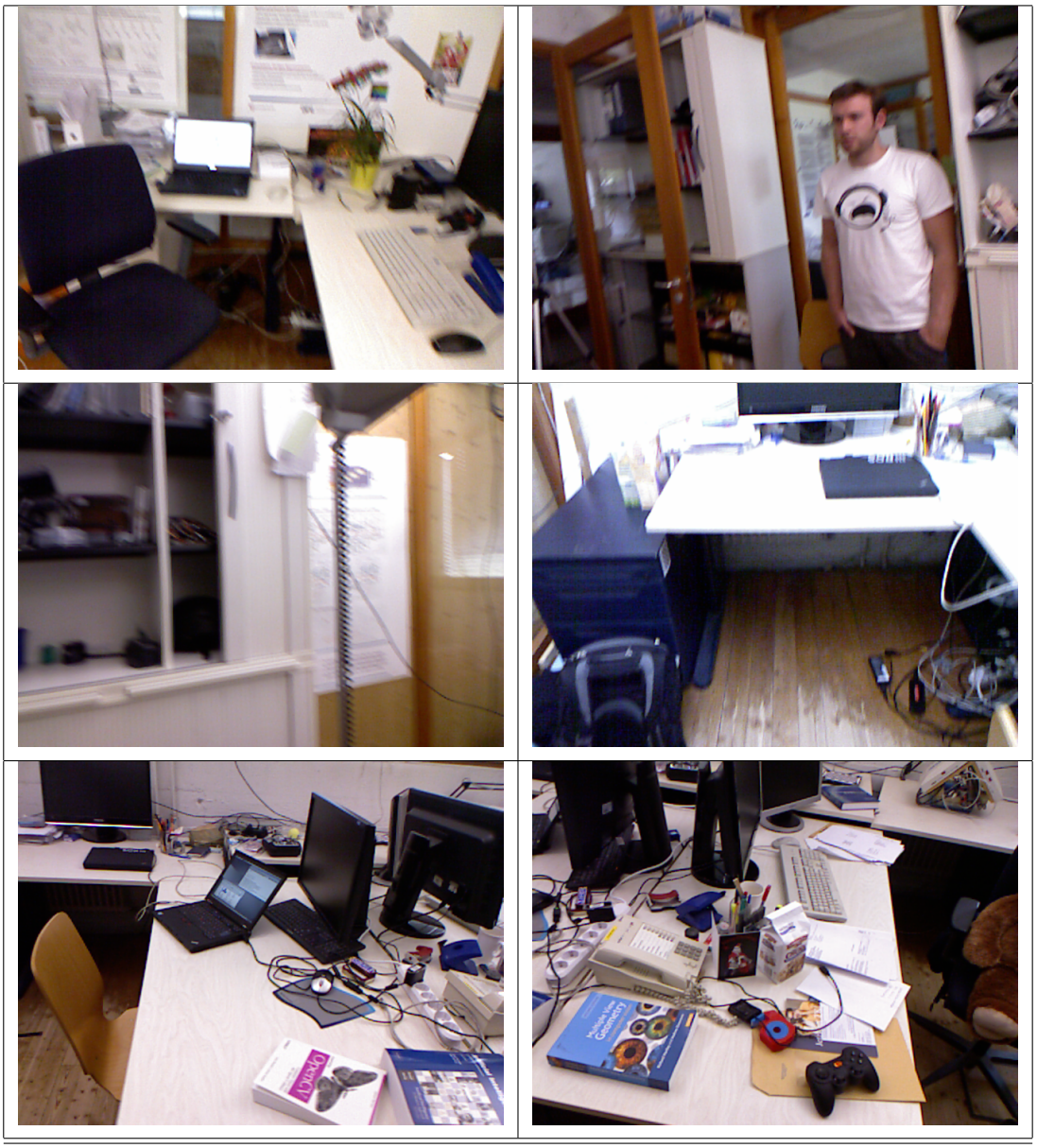


FIGURA 9.15: Grafo de poses de la secuencia 3. El círculo rojo resalta los vértices que pertenecen a la parte sin textura, cada uno de estos nodos solo están conectado entre si por una arista.

9.3.5. Secuencia 4

Se probó el sistema con la secuencia *rawlog_rgb_dataset_freiburg1_room* del benchmark, de la cual se puede ver una muestra de las imágenes en la tabla 9.7. La secuencia es mayormente tomas a 360 grados de un cuarto. La cámara se mueve la mayor parte del tiempo a una velocidad media, pero en ocasiones a velocidad rápida, lo cual puede provocar problemas en la extracción de características. Se hacen pocas tomas de lugares sin textura. La cámara pasa por lugares visitados anteriormente en algunas ocasiones, por lo que se tienen varios cierres de ciclos. La secuencia da lugar a la reconstrucción de un mapa relativamente grande. Por todas las propiedades anteriores mencionadas, está secuencia resulta un buena para probar el desempeño del sistema, con problemas como los mencionados en las secuencias 2 y 3.



CUADRO 9.7: Muestra de imágenes de la secuencia 4.

Los resultados de la evaluación del sistema para esta secuencia se pueden observar en la tabla 9.8. Los resultados muestran que no se tiene un error considerablemente alto. Se observa una disminución sustancial del error después de la optimización del grafo.

rawlog_rgb_dataset_freiburg1_room	
ATE	0.225118
ATE (Optimizado)	0.079954
RPE	0.482966610258
RPE (Optimizado)	0.407133514625

CUADRO 9.8: Los resultados de las métricas ATE y RPE para la secuencia 4.

A continuación se muestra el resultado de graficar la métrica ATE, la cual se puede apreciar en la figuras 9.17 y 9.16. En la primera gráfica se aprecia que la trayectoria estimada es similar a la verdadera, pero que existe una diferencia considerable entre las dos. Después de la optimizaicón, la trayectoria estimada es mucho más parecida a la verdadera, pero aún se puede apreciar la diferencia entre las dos en algunas partes. En algunos puntos es muy pequeña y en otros más considerable.

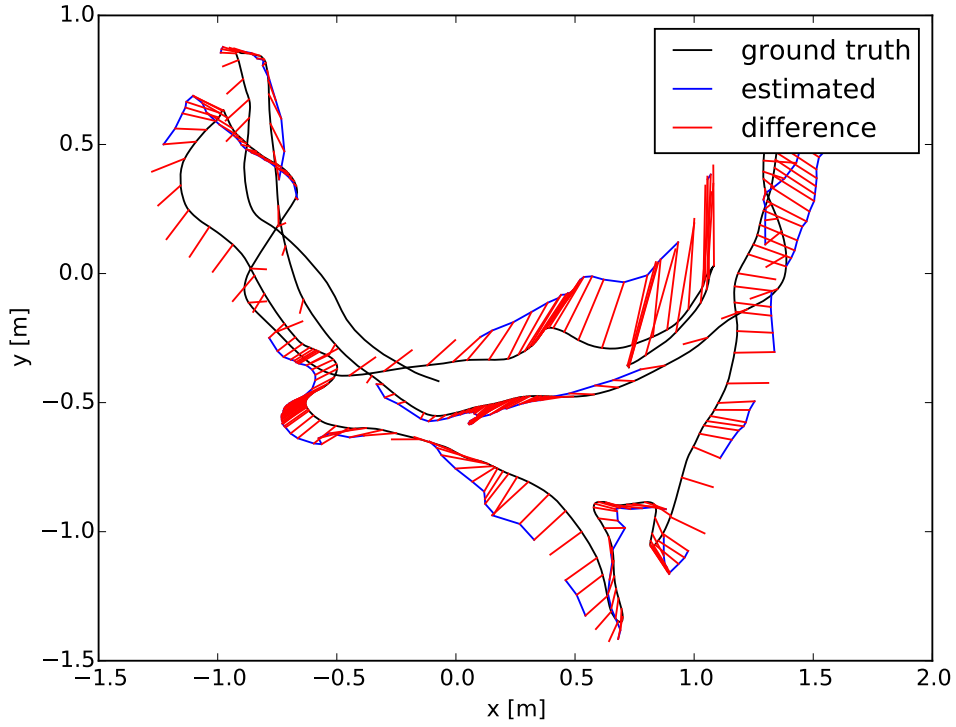


FIGURA 9.16: Diferencia entre la trayectoria estimada y la verdadera de la secuencia 4.

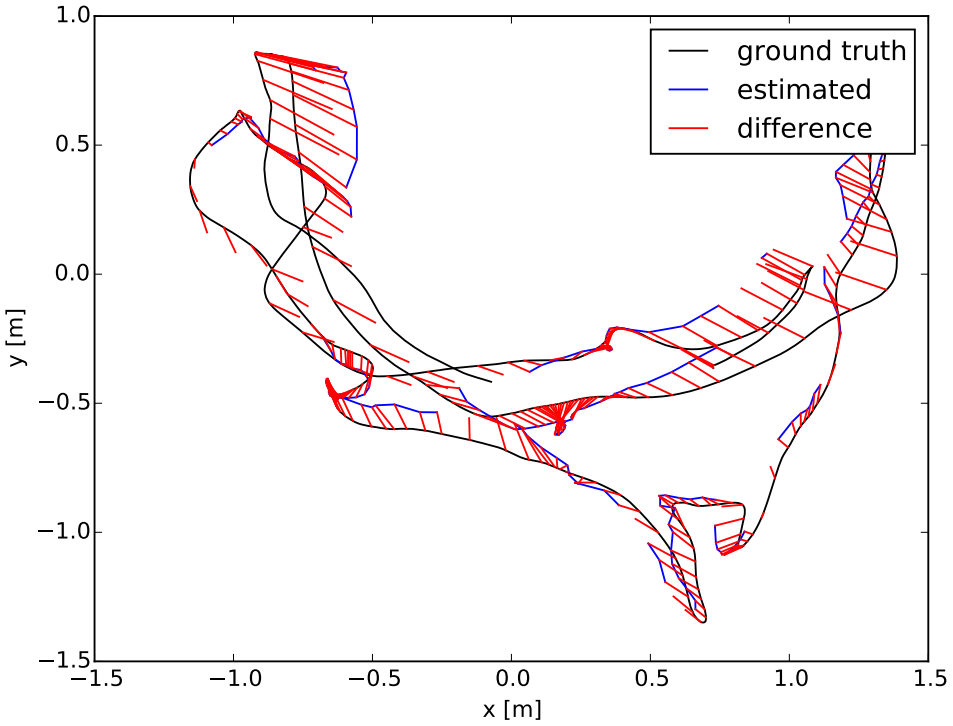


FIGURA 9.17: Diferencia entre la trayectoria estimada optimizada y la verdadera de la secuencia 4.

Las figuras 9.18 y 9.19 muestran los mapas construidos. En el primer mapa se pueden apreciar pequeñas inconsistencias que se resaltan mediante círculos rojos. En el mapa optimizado se aprecia que estas inconsistentes han disminuido. Para ser un mapa de un cuarto completo se obtuvo una buena reconstrucción.

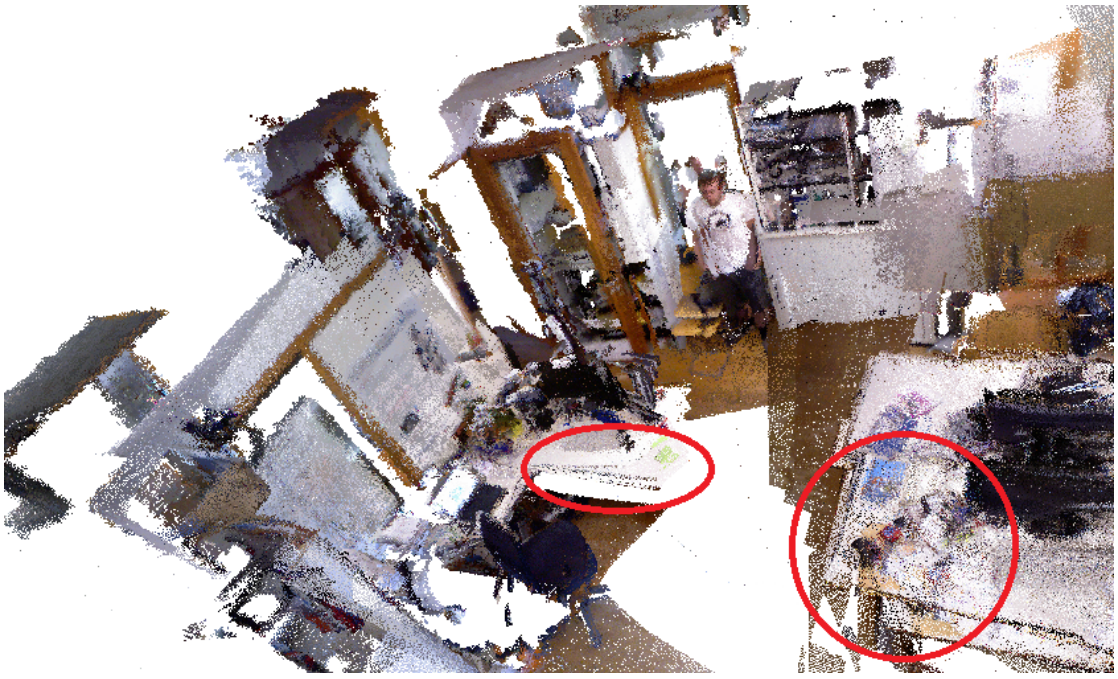


FIGURA 9.18: Mapa 3D construido de la secuencia 4. Los círculos rojos señalan inconsistencias visibles en el mapa.

FIGURA 9.19: Mapa 3D optimizado construido de la secuencia 4.

En las figura 9.20 se presenta el grafo de poses para esta secuencia. Como se mencionó al inicio de este sección, en esta secuencia se visita lugares anteriormente vistos. Al revisar el grafo de poses se comprueba que se pudieron detectar gran cantidad de cierres de ciclos en algunos lugares.

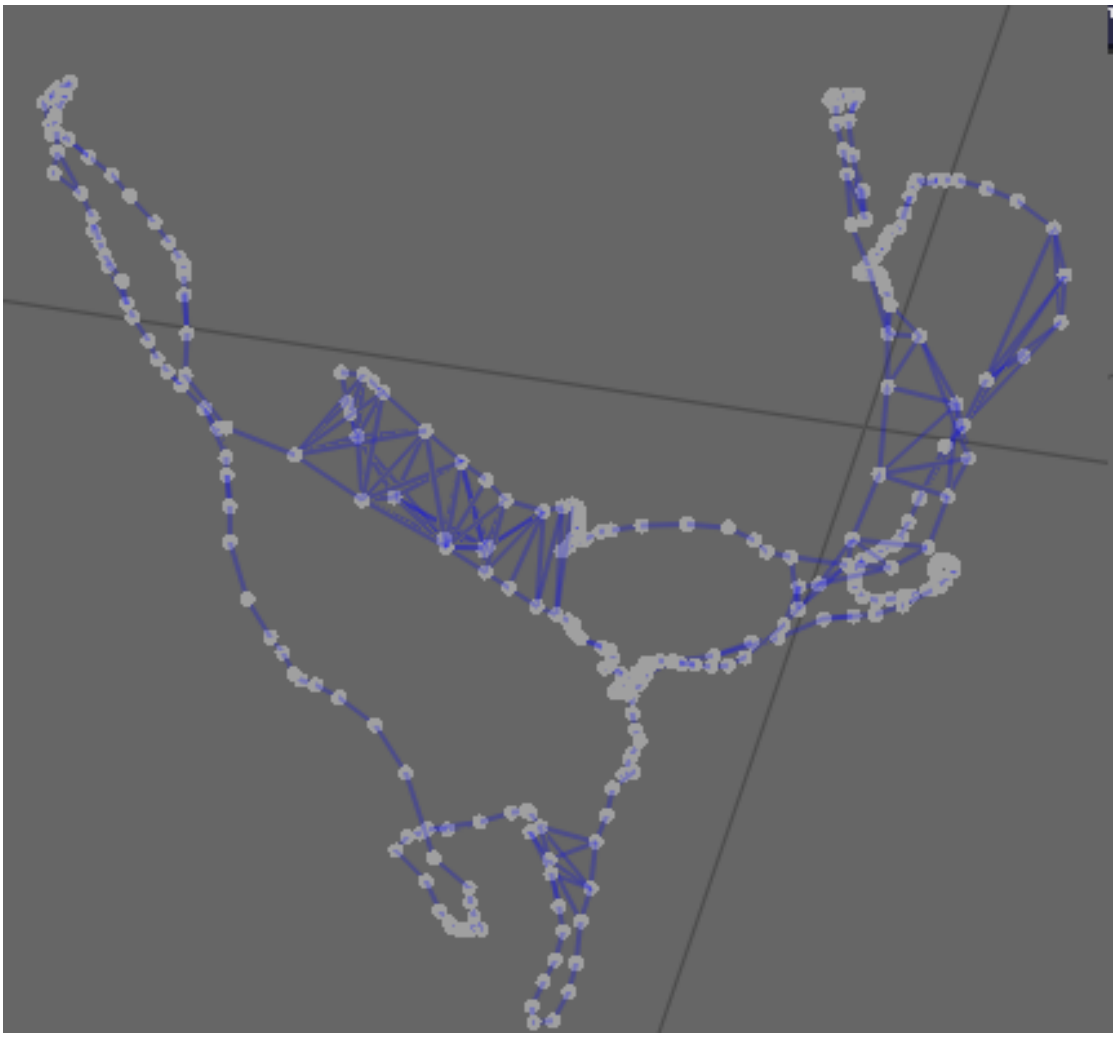


FIGURA 9.20: Grafo de poses de la secuencia 4. Hay lugares donde se detectó gran cantidad de cierres de ciclos.

Capítulo 10

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones a las que se llegaron a través de la elaboración de este trabajo de tesis. Además se resumen los resultados obtenido en el capítulo 9 y se analizan los factores que los afectan y se discuten cuestiones que podrían haber sido tratados de manera diferente. El capítulo también ofrece una lista de los trabajos futuros con tareas relevantes que podrían mejorar el rendimiento del sistema.

10.1. Conclusiones

La principal conclusión a la que se pudo llegar en este trabajo de tesis es que se logró desarrollar un sistema que permite la construcción de mapas 3D a partir de los datos captados mediante un sensor Kinect, el cual aporta información RGB-D. Además se logró obtener una buena estimación de la posición y orientación del sensor Kinect en su entorno. El cual puede moverse libremente en 6 grados de liberad. Para lograr esto, se hizo uso de la información RBG y de la profundidad captadas por el sensor para lograr una primera aproximación de la trayectoria, al seguir un enfoque de alineación de nubes de puntos de forma sucesiva, en la cual se realizó un emparejamiento de características visuales 2D SURF que se correspondían en dos imágenes consecutivas. Luego se realizó una extracción de las características 3D asociadas a los emparejamientos 2D y se realizó una aproximación a la pose mediante RANSCAC/ICP. El enfoque no fue suficiente para construir mapas consistentes debido a la acumulación del error en cada una de las alineaciones consecutivas de las nubes de puntos y también por el ruido inherente en los sensores. Es por ello que se implementó el algoritmo SLAM basado en grafo. Para el cual se dio una solución al cierre de ciclos y se obtuvo una mejor estimación en la pose del sensor Kinect y una construcción más precisa del mapa 3D del ambiente.

Teniendo en cuenta que las cámaras RGB-D pronto estarán disponibles al público a un precio bajo, un sistema de modelado basado en RGB-D potencialmente tendrá un enorme impacto en la vida cotidiana, lo que permitirá a la gente construir modelos 3D de ambientes interiores. Además, los resultados de este trabajo indican que las cámaras RGB-D podrían ser utilizadas para construir mapas y sistemas de navegación robustos. La aplicación de cámaras RGB-D podría ser un paso importante para permitir el desarrollo de plataformas de robots asequibles y útiles.

A pesar de los resultados alentadores obtenidos, nuestro sistema tiene varias deficiencias que merecen un esfuerzo a futuro. El sistema desarrollado no funciona en tiempo real, el algoritmo fue desarrollado para que solo funcione fuera de línea. Si bien el modo en línea tiene la ventaja obvia de rápidos resultados, este modo en presencia de ambientes con una falta de características únicas, tiene resultados que pueden ser subóptimos. El algoritmo fuera de línea ofrece la robustez que el algoritmo en línea carece y proporciona mapas más óptimos en la mayoría de las condiciones, al sacrificar velocidad por precisión. Se cree que una implementación que tome ventaja de los modernos y eficientes GPUs pueda lograr el aumento de velocidad para que funcione en tiempo real.

En esta tesis se muestran que los sensores RGB-D proporcionan la flexibilidad necesaria para desarrollar un algoritmo de SLAM completo basado sólo en los datos del sensor, en este caso, un sensor Kinect. Mientras que la odometría visual sufre del ruido típico, el cual es notable especialmente en los movimientos de rotación rápida debido a la falta de definición. Las técnicas de corrección de pose implementadas puede superar muchas veces ese obstáculo y proporcionar una estimación fiable de la pose a lo largo del tiempo.

Se investigó cómo se comportaba el sistema desarrollado bajo ciertas circunstancias extremas en ambientes de interior. Como por ejemplo, Se investigó cómo un ambiente de interior con pocas características reconocibles en las imágenes afectaría al rendimiento del mismo y qué ajustes se podrían sugerir para mejorar el resultado en ese entorno. Se llegó a la conclusión de que un ambiente con pocas características afectarán el desempeño de una manera negativa. Muy pocos puntos de referencia hará que el sistema pierda su trayectoria y genere mapas inconsistentes, después de lo cual, la posibilidad de una vez más conseguir una buena estimación es pequeña. El algoritmo de SLAM basado en grafos genera estimaciones decentes durante períodos de mala información visual y esto puede en algunos casos solucionar el problema. Aunque la solución aportada otorga resultados aceptables para la reconstrucción de mapas de habitaciones de tamaño pequeños, la implementación muestra resultados cada vez más pobres a medida que va creciendo el medio ambiente a explorar.

Aunque se pueden lograr reconstrucciones decentes del mapa 3D del ambiente y un buena aproximación a la trayectoria del sensor, el sensor Kinect resulta demasiado ruidoso. Esto

dificulta la tarea de atacar el problema de SLAM. Aún cuando se aplican filtros para la reducción del ruido y todos los algoritmos implementados en trabajo que tienen como finalidad solucionar el problema de SLAM. El ruido generado por el sensor Kinect sigue siendo de importancia.

La solución propuesta en este trabajo de tesis para el cierre de ciclos es altamente ineficiente y por lo tanto, consume el un gran tiempo de computo del sistema. Aunque la solución cumple con su objetivo de reducir la incertidumbre en las mediciones del sensor Kinect y ayuda a construir mapas 3D mas congruentes y realizar cálculos de la trayectoria más acertados, se debe buscar una solución mas óptima.

10.2. Trabajos futuros

Con este trabajo, se logró obtener una buena perspectiva de la problemática general de SLAM y más concretamente del SLAM visual con extracción de características. Por lo que es posible imaginar algunas mejoras en el trabajo actual, tanto en términos de calidad de los resultados y el rendimiento general.

Como hemos visto, las técnicas del estado del arte para mapas RGB-D obtienen un gran beneficio del uso de descriptores de características locales a partir de imágenes de color (bajo condiciones de iluminación controladas). Se notó en este trabajo que las características locales existentes son demasiado sensibles a la naturaleza ruidosa de las cámaras RGB-D comerciales actuales. Por lo que se plantea trabajar en el desarrollo/uso de características que se adapten a este nuevo tipo de datos obtenidos por los sensores RGB-D. También es una opción estudiar nuevos paradigmas de extracción de características como lo son las Redes neuronales convolucionales, que representa la nueva tendencia en reconocimiento de objetos.

En un sistema SLAM, la opción de ofrecer información al usuario durante el tiempo de adquisición es valiosa. Por ejemplo, informar al operador acerca de las posibles ubicaciones de la escena en la que faltan datos o donde se produjeron desajustes. Ejemplo claro de esto es cuando la imagen es borrosa por movimientos rápidos de la cámara, o por falta de textura en el entorno. Sobre esta base, se puede trabajar en un sistema que permita al usuario realizar los ajustes manuales del modelo durante la adquisición, en caso de que el registro falle.

Como se mencionó al inicio de este trabajo, una de las suposiciones más comunes que se hacen en los sistemas de SLAM es que el ambiente en el que se trabajará es estático, como se hizo en esta tesis. Es por ello que al sistema desarrollado le falta de robustez bajo la presencia de objetos en movimiento en la escena. Esto se puede resolver de una

manera sencilla si se consideran los puntos correspondientes a los objetos en movimiento como datos atípicos, aunque ésta no es una solución robusta. Se necesita una solución más robusta para este problema.

Explotar la posibilidad de modularidad. Por ejemplo, con niveles jerárquicos correspondientes a diferentes escalas para los mapas más grandes. Esta idea ya se a trabajado en otros sistemas, sobre todo aquellos que usan el enfoque EFK. La idea es que se construyan mapas más pequeños que el original, donde el error acumulado no será tan grande y luego se juntarán todos para crear el mapa total.

Se podrían utilizar sensores adicionales, en este trabajo el único sensor utilizado fue el sensor Kinect que proporciona datos RGB-D. Una posibilidad sería utilizar otros sensores y combinarlos para obtener un mejor resultado en un sistema híbrido. Diferentes sensores aportan más información que puede ser utilizada para crear un mejor grafo de poses y por lo tanto, una mejor reconstrucción del mapa 3D del ambiente.

Otra posible mejora es la implementación de ciertos algoritmos en GPU. Por ejemplo, un algoritmo de extracción de características optimizado en hardware de tarjetas gráficas podría reducir drásticamente el tiempo de cálculo, lo cual permitirá aplicaciones en tiempo real. Existen diversas implementaciones en GPU para SURF. Además, algunas soluciones aprovechan la computación paralela en plataformas específicas, tales como CUDA para las tarjetas gráficas NVIDIA. La extracción de características podría llevarse a cabo en milisegundos.

Investigar posibles ventajas con más algoritmos detectores de características. En este trabajo se optó por usar SURF, uno de los extractores mas usados y robustos en la literatura. Sin embargo, se podría probar con más u otros algoritmos extractores, o combinaciones de ellos. Por ejemplo, en este trabajo se vio que el sistema no es robusto en ambientes sin textura. Con algoritmos extractores enfocados en describir la estructura 3D del ambiente, se lograría un sistema más robusto.

Añadir un algoritmo de cierre de ciclos más óptimo y robusto. El algoritmo de cierre de ciclos utilizado en este trabajo es un poco “ad hoc”. El sistema detecta un cierre de ciclo solo si el número de correspondencias es mayor a un cierto umbral una vez que revisó todos los fotogramas almacenados pasados, lo cual es bastante costoso computacionalmente. Existe algoritmos más óptimos y robustos probados para el cierre de ciclos, por lo que se buscaría implementar alguno de ellos en el sistema. Uno de los algoritmo más eficaces para este problema es el conocido como “bolsa de imágenes”, el cual se basa en la misma idea que la clasificación de textos por el algoritmo de “bolsa de palabras”.

Como mejora final se propone añadir una representación Ocotmap. Esta representación es útil para tareas de navegación, como el cálculo de trayectorias. Esta representación consiste en identificar el espacio ocupado y libre, debido a que es necesario saber el espacio por el cual se puede navegar.

Bibliografía

[1] B. C. J.M. Frahm, P. Mordoha, "Towards urban 3d reconstruction from video," *3D Data Processing, Visualization,, Transmission, Third International Symposium*, pp. 1–8, June 1991.

[2] I. R. D. Laura A., Andrew Davison J. and J. N., "Mapping large loops with a single hand-held camera," *Robotics: Science and Systems*, 2007.

[3] D. F. Sebastian T., Wolfram B., "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," *Robotics and Automation*, p. 321–328, 200.

[4] B. W. Triebel R., "Improving simultaneous mapping and localization in 3d using global constraints," *Proceedings of the National Conference on Artificial Intelligences*, vol. 20, pp. 1330–1330, 2005.

[5] S. A. Gordon N., Salmond D., "Novel approach to nonlinear/non-gaussian bayesian state estimation," *Radar and Signal Processing*, vol. 140, p. 107–113, 1993.

[6] M. E. Lu F., "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, 1997.

[7] T. S. Wang C, Thorpe Ch, "Simultaneous localization, mapping and moving object tracking," *J Robot Res*, vol. 26, p. 889–916, 2007.

[8] M. D. Wangsiripitak S, "Avoiding moving outliers in visual slam by tracking moving objects," *Proceedings of the IEEE international conference on robotics and automation*, p. 375–380, 2009.

[9] M. D. Migliore D, Rigamonti R, "Use a single camera for simultaneous localization and mapping with mobile object tracking in dynamic environments," *CRA workshop on safe navigation in open and dynamic environments: application to autonomous vehicles*, 2009.

[10] W. C. Lin K, "Stereo-based simultaneous localization, mapping and moving object tracking," *Proceedings of the IEEE international conference on intelligent robots and systems*, p. 3975–3980, 2010.

[11] N. P. Tardós JD, Neira J, "Robust mapping and localization in indoor environments using sonar data," *Springer Tracts in Advanced Robotics*, vol. 21, p. 311–330, 2002.

[12] N. P. Tardós JD, Neira J, "Underwater slam in man-made structured environments," *Field Robot*, vol. 25, p. 898–921, 2008.

[13] H. J. Nüchter A, Lingemann K, "6d slam—3d mapping outdoor environments," *Field Robot*, vol. 254, p. 699–722, 2007.

[14] A. A. Thrun S, Montemerlo M, "Probabilistic terrain analysis for high speed desert driving," *Proceedings of robotics: science and systems*, 2006.

[15] J. I. Lemaire T, Berger, "Vision-based slam: stereo and monocular approaches," *Comput Vis*, vol. 74, p. 343–364, 2007.

[16] M. B. Bogdan R, Sundaresan A, "Leaving flatland: efficient real-time three dimensional perception and motion planning," *Field Robot. Special Issue on Three-Dimensional Mapping*, vol. 26, p. :841–862, 2009.

[17] D. H. Thrun S, Montemerlo M, "Stanley: the robot that won the darpa grand challenge," *Field Robot*, vol. 22, p. 661–692, 2005.

[18] T. J. Paz L, Piniés P and N. J, "Large-scale 6dof slam with stereo-in-hand," *IEEE Trans Robot*, vol. 24, p. 946–957, 2008.

[19] R. I. Clemente L, Davison A, ") mapping large loops with a single hand-held camera," *Proceedings of robotics: science and systems conference*, 2007.

[20] M. D. Klein G, "Parallel tracking and mapping for small ar workspaces," *Proceedings of the 6th IEEE and ACM international symposium on mixed and augmented reality*, 2007.

[21] E. F. Sáez J, "6dof entropy minimization slam," *Proceedings of the IEEE international conference on robotics and automation*, p. 1548–1555, 2006.

[22] T. Piniés P, "Large scale slam building conditionally independent local maps: application to monocular vision," *IEEE Trans Robot*, vol. 24, p. 1094–1106, 2008.

[23] S. M. M. M. Olson C, Matthies L, "Rover navigation using stereo ego-motion," *Robot Autonom Syst*, vol. 43, p. 215–229, 2003.

[24] S. P. Hartley R, "Triangulation," *Comput Vis Image Underst*, vol. 68, p. 146–157, 1997.

[25] A. M. Konolige K, "Frameslam: from bundle adjustment to real-time visual mapping," *IEEE Trans Robot*, vol. 24, p. 1066–1077, 2008.

[26] C. J. Konolige K, Bowman J, "View-based maps," *Proceedings of robotics: science and systems*, 2009.

[27] C. M. Mei C, Sibley G, "A constant-time efficient stereo slam system," *Proceedings of the British machine vision conference*, 2009.

[28] D. A, "Real-time simultaneous localisation and mapping with a single camera," *Proceedings of the IEEE international conference on computer vision*, vol. 2, p. 1403–1410, 2003.

[29] N. D, "An efficient solution to the five-point relative pose problem," *IEEE Trans Pattern Anal Mach Intell*, vol. 26, p. 756–770, 2004.

[30] C. A. Pupilli M, "Proceedings of the ieee conference on computer vision and pattern recognition," *Proceedings of robotics: science and systems conference*, vol. 1, p. 1244–1249, 2006.

[31] M. S. C. Harris, "A combined corner and edge detector," *Alvey vision conference*, p. 50–50, 1988.

[32] S. C. Mikolajcczyk K, "An affine invariant interest point detector," *Proceedings of the European conference on computer vision*, p. 128–142, 2002.

[33] L. V. G. H. Bay, T. Tuytelaars, "Surf: Speeded up robust features," *Computer Vision–ECCV*, p. 404–417, 2006.

[34] S. S. Mikolajczyk K, Tuytelaars T, "A comparision of affine region detectors," *Int J Comput Vis*, vol. 65, p. 43–72, 2005.

[35] M. K. Tuytelaars T, "Local invariant feature detectors: a survey," *Found Trends Comput Graph*, 2008.

[36] B. M. R. O. Gil A, Martínez O, "A comparative evaluation of interest point detectors and local descriptors for visual slam," *Mach Vis Appl 21*, vol. 6, p. 905–920, 2009.

[37] Y. G. Morel J, "Asift: a new framework for fully affine invariant image comparison," *SIAM J Imaging*, vol. 2, p. 438–469, 2009.

[38] L. V. Calonder M, "Brief: binary robust independent elementary features," *Proceedings of the European conference on computer vision*, 2010.

[39] K. K. G. B. E. Rublee, V. Rabaud, "Orb: an efficient alternative to sift or surf," *International Conference on Computer Vision*, 2011.

[40] T. J. Neira J, "Data association in stochastic mapping using the joint compatibility test," *Proceedings of the IEEE international conference on robotics and automation*, vol. 17, p. 890–897, 2001.

[41] D. T. Grauman K, "Pyramid match hashing: sub-linear time indexing over partial correspondences," *roceedings of the IEEE conference on computer vision and pattern recognition*, 2007.

[42] P. M. Raguram R, Frahm J, "A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus," *Proceedings of the European conference on computer vision*, p. 500–513, 2008.

[43] N. P. Ho K, "Detecting loop closure with scene sequences," *Int J Comput Vis*, vol. 74, p. 261–286, 2007.

[44] F. D. Angeli A, Doncieux S, "Real time visual loop closure detection," *Proceedings of the IEEE international conference on robotics and automation*, 2008.

[45] D. T. Eade E, "Unified loop closing and recovery for real time monocular slam," *In Proceedings of the British Machine vision conference*, 2008.

[46] F. D. Angeli A, Doncieux S, "Real time visual loop closure detection," *Proceedings of the IEEE international conference on robotics and automation*, 2008.

[47] N. P. Cummins M., "Fab-map: probabilistic localization and mapping in the space of appearance," *Int J Robot Res*, vol. 27, p. 647–665, 2008.

[48] A. H. Magnusson M., "Automatic appearance-based loop detection from 3d laser data using the normal distribution transform," *J Field Robot, Three Dimensional Mapping Part 2*, vol. 26, p. 892–914, 2009.

[49] P. S. P. Cheeseman, "On the representation and estimation of spatial uncertainty," *Robot.*, p. 56–68, 1986.

[50] C. P. Smith R., Self M., "Estimating uncertain spatial relationships in robotics," *Robotics and Automation*, vol. 4, pp. 850–850, 1987.

[51] R. C. P. Moutarlier, "An experimental system for incremental environment modeling by an autonomous mobile robot," *1st International Symposium on Experimental Robotics*, 1989.

[52] R. C. P. Moutarlier, "Stochastic multisensory data fusion for mobile robot location and environment modeling," *5th Int. Symposium on Robotics Research*, 1989.

[53] A. Jazwinsky, "Stochastic processes and filtering theory," *(Academic*, 1970.

[54] R. Kalman, "A new approach to linear filtering and prediction problems," *ASME*, p. 35–45, 1960.

[55] P. Maybeck, "The kalman filter: An introduction to concepts," *Autonomous Robot Vehicles*, 1990.

[56] S. B. J. Guivant, E. Nebot, "Autonomous navigation and map building using laser range sensors in outdoor applications," *J. Robot. Syst.*, p. 565–583, 2000.

[57] H. Durrant-Whyte, "Uncertain geometry in robotics," *Trans. Robot. Autom.*, p. 23–31, 1988.

[58] S. U. N. Metropolis, "The monte carlo method," *J. Am. Stat. Assoc.*, p. 335–341, 1949.

[59] D. Blackwell, "Conditional expectation and unbiased sequential estimation," *, Ann. Math. Statist*, p. 105–110, 1947.

[60] C. Rao, "Information and accuracy obtainable in estimation of statistical parameters," *Bull. Calcutta Math. Soc*, p. 81–91, 1945.

[61] S. R. K. Murphy, "Rao-blackwellized particle filtering for dynamic bayesian networks," *Sequential Monte Carlo Methods in Practice*, p. 499–516, 2001.

[62] D. K. B. W. M. Montemerlo, S. Thrun, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.

[63] J. D. C. P. M. M. C. V. L. P. F. K. Konolige, J. Bowman, "Viewbased maps," *International Journal of Robotics Research (IJRR)*, 2010.

[64] M. Csorba, "Simultaneous localisation and map building," *Ph.D. Thesis (University of Oxford, Oxford)*, 1997.

[65] J. T. J. Neira, "Data association in stochastic mapping using the joint compatibility test," *Trans. Robot. Autom.*, p. 890–897, 2001.

[66] J. C. J. Neira, J.D. Tardos, "Linear time vehicle relocation in slam," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003.

[67] T. Bailey, "Mobile robot localisation and mapping in extensive outdoor environments," *Ph.D. Thesis (University of Sydney, Sydney 2002)*, 2002.

[68] S. C. H. D. M. C. G. Dissanayake, P. Newman, "A solution to the simultaneous localisation and map building (slam) problem," *Trans. Robot. Autom*, p. 890–897, 2001.

[69] H. D.-W. T. B. G. Dissanayake, S.B. Williams, "Map management for efficient simultaneous localization and mapping (slam)," *Autonom.Robot*, 2002.

[70] J. H.-J. S. W. B. N. Engelhard, F. Endres, "Real-time 3d visual slam with a hand-held rgb-d camera," *RGB-D workshop on 3D perception in robotics at the European robotics forum*, 2011.

[71] O. G. Juan, "A comparison of sift, pca-sift and surf," *International Journal of Image Processing*, p. 143–152, 2009.

[72] R. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *PhD thesis Computer Science department Technische Universitat München*, 2009.

[73] L. D. Nguyen C. V., Izadi S., "Modeling kinect sensor noise for improved 3d reconstruction, tracking," *In 3D Imaging, Modeling, Processing, Visualization and Transmission*, p. 524 –530, 2012.

[74] T. J. W., "Exploratory data analysis," *Addison-Wesley*, 1977.

[75] D. F. Paris S., "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput*, p. 24–52, 2009.

[76] C. D. F. S. L. D. S. T. Alexa M., Behr J., "Computing and rendering point set surfaces," *Transactions on Visualization and Computer Graphics*, p. 3–15, 2003.

[77] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, 1975.

[78] R. A. F. J. H. Friedman, J. L. Bentley, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, 1977.

[79] E. H. X. R. D. F. P. Henry, M. Krainin, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments," *International Symposium on Experimental Robotics*, 2010.

[80] Z. Z. J. Garcia, "Range mapping using speckle decorrelation," *Patent*, 2008.

[81] N. M. P.J. Besl, "A method for registration of 3-d shapes," *Transactionson pattern analysis and machine intelligencer*, p. 239–256, 1992.

[82] S. T. A. Segal, D. Haehnel, "Generalized-icp," *Proceedings of Robotics: Science and Systems*, 2009.

[83] K. K. N. Fioraio, "Realtime visual and point cloud slam," *Computer*, p. 1–3, 2011.

[84] R. C. B. M. A. Fischler, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, p. 381–395, 1981.

[85] G. M. Y. Chen, "Object modeling by registration of multiple range images," *International Conference on Robotics and Automation,*, p. 2724–2729, 1991.

[86] M. L. S. Rusinkiewicz, "Efficient variants of the icp algorithm," *Third International Conference on 3-D Digital Imaging and Modeling*, p. 145–152, 2001.

[87] N. Y. T. Masuda, K. Sakaue, "Registration and integration of multiple range images for 3-d model construction," *13th International Conference on Pattern Recognition*, p. 879–883, 1996.

[88] R. W. T. Noah A. Smith, "Sampling uniformly from the unit simplex," *Center for Language and Speech Processing Johns Hopkins University*, 2004.

[89] R. B. G. Godin, M. Rioux, "Three-dimensional registration using range and intensity information," *The International Society for Optical Engineering*, p. 279–290, 1994.

[90] S. D. B. K. S. Arun, T. S. Huang, "Least-squares fitting of two 3-d point sets," *Transactions on Pattern Analysis and Machine Intelligence*, p. 698–700, 1987.

[91] M. H. O.D. Faugeras, "The representation, recognition, and locating of 3-d objects," *International Journal of Robotics Research*, p. 5(3):27, 1986.

[92] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A (Optics and Image Science)*, p. 629–642, 1987.

[93] G. M. Y. Chen, "Object modeling by registration of multiple range images," *International Conference on Robotics and Automation*, p. 2724–2729, 1991.

[94] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, p. 331–371, 1910.

[95] J. N. P. N. I. R. J. T. B. Williams, M. Cummins, "An imageto-map loop closing method for monocular slam," *Intelligent Robots and Systems*, 2008.

[96] T. S. K. Granstrom, "Learning to close the loop from 3d point clouds," *Intelligent Robots and Systems*, 2010.

[97] J. v. B. M. H. G. Hanspeter Pfister, Matthias Zwicker, "Surfels: surface elements as rendering primitives," *SIGGRAPH*, p. 335–342, 2000.

[98] F. E. W. B. D. C. J. Sturm, N. Engelhard, "A benchmark for the evaluation of rgb-d slam systems," *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct 2012.