Tesis Doctoral
Doctorado en Ingeniería - Sistemas e Informática

SLAM Monocular en Tiempo Real

Real-Time Monocular SLAM

Trabajo presentado por

Gustavo Alonso Acosta Amaya

Director
Prof. Jovani Alberto Jiménez Builes, PhD.

Comité Evaluador:
Prof. José Gabriel Hoyos, PhD. Universidad del Quindío
Juan Diego Lemos, PhD. Universidad de Antioquia
Prof. Víctor Hugo Jaramillo, PhD. Universidad EIA

Universidad Nacional de Colombia
2019

Declaración

Yo declaro que el contenido y organización de esta tesis son producto de mi trabajo original y no compromete en modo alguno los derechos de terceros. En su desarrollo he enumerado todas las fuentes de información y observado los principios éticos establecidos por la institución.

Gustavo Alonso Acosta Amaya
2019

* Este trabajo se presenta como requisito parcial para optar al título de Doctor en Ingeniería de la Facultad de Minas de la Universidad Nacional de Colombia.

Quiero dedicar el esfuerzo y empeño puestos en este trabajo a mi amada familia,

A mis hijos Andrés Felipe, Johan Esteban y Tomás Daniel

A mi esposa Ana Gil y mi nieto Juanjo

A mis padres y hermanos, junto con "Agaíta"

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi director de tesis, Profesor Jovani Alberto Jiménez Builes PhD., por sus valiosos aportes, respaldo y guía permanente. A las personas que me han alentado a culminar este trabajo; familiares, amigos y colegas. De manera especial al Politécnico Colombiano Jaime Isaza Cadavid y mis estudiantes del programa de Ingeniería en Instrumentación y Control.

Resumen

Una de las tareas fundamentales que debe poder ejecutar un robot móvil es la navegación autónoma de su entorno de trabajo, para lo que se requiere de un modelo del entorno o mapa y un método para estimar la localización en éste. Sin embargo son numerosas las situaciones en las que no se dispone, *a priori*, de una representación del entorno, por ejemplo en labores de búsqueda, rescate, exploración planetaria, exploración oceánica y minería subterránea.

En tales circunstancias se deberán resolver de manera simultánea ambos problemas, la localización y el mapeo. En efecto, la estimación de la localización requiere de un mapa, y a su vez, para elaborar un mapa es necesario establecer una localización con relación a un modelo. La solución simultánea de estos dos problemas se conoce en robótica como SLAM (*Simultaneous Localization and Mapping*).

Desde su formulación hace más de treinta años, la comunidad científica vinculada a la robótica ha invertido esfuerzo y energía en la solución del SLAM. En la actualidad se considera un componente fundamental de los sistemas robóticos, permitiéndoles realizar tareas más complejas y por tanto, otorgándoles mayores niveles de autonomía.

Si bien el SLAM bidimensional para entornos interiores de pequeña escala se considera un problema resuelto con el que se obtienen resultados consistentes, una vez se extiende a la estimación y reconstrucción tridimensional o a entornos de grandes dimensiones, nuevos retos de investigación surgen de inmediato.

Para el caso particular del SLAM tridimensional con sensores visuales, algunos de los nuevos problemas que deben ser resueltos son: mayor complejidad y costo computacional debido al gran volumen de datos a procesar, errores debidos a la baja resolución de los sensores, cambios de iluminación en el entorno, superficies con falta de textura e imágenes borrosas por movimientos rápidos de la cámara.

En esta tesis se adelanta un estudio sistemático y riguroso del SLAM, desde su formulación y métodos de solución, hasta la evaluación de algunos de los algoritmos SLAM de código abierto mas recientes. De manera particular se aborda el problema del SLAM monocular en tiempo real y se conducen experimentos en entornos interiores con un sistema robótico especialmente diseñado para tal fin. Las principales contribuciones de este trabajo son:

- El estudio sistemático y exhaustivo del SLAM, desde su formulación hasta los métodos de solución más representativos (Capítulo 2).
- La formulación de un método de evaluación de los algoritmos SLAM con base en dos métricas (MeC y MoC) que consideran la calidad de los mapas producidos (Capítulo 3).
- El diseño y construcción de un sistema robótico totalmente compatible con ROS (*Robot Operating System*) para la validación experimental conducida en esta tesis y la investigación y desarrollo de aplicaciones (Capítulo 3).
- El estudio riguroso de los algoritmos *visual SLAM* que hacen parte del estado del arte actual y, de manera particular, los métodos relativos a los problemas SfM (*Structure from Motion*) y el SLAM monocular en tiempo real (Capítulos 4 y 5).

Palabras clave: SLAM monocular, SLAM visual, Robótica móvil, Filtro de Kalman extendido, Filtro de partículas, Localización robótica, Mapeo robótico, ROS, SfM, Reconstrucción 3D.

Abstract

One of the fundamental tasks that a mobile robot must be able to execute is the autonomous navigation of its working environment, for which a model of the environment or map is required and a method to estimate the location in it. However, there are numerous situations in which there is not available, *a priori*, a representation of the environment, for example in search and rescue, planetary exploration, ocean exploration and underground mining.

In that circumstances, both problems, location and mapping must be solved simultaneously. In effect, location estimation requires a map, and in turn, to create a map it is necessary to establish a location in relation to a model. The simultaneous solution of these two problems is knows in robotics as SLAM (*Simultaneous Localization and Mapping*).

Since its formulation more than thirty years ago, the research community linked to robotics has invested effort and energy in the solution of SLAM. Nowadays, it is considered a fundamental component of robotic systems, allowing them to perform more complex tasks and therefore, giving them greater levels of autonomy.

Although the two-dimensional SLAM for small-scale indoor environments is considered a solved problem, with consistent results, when it extends to three-dimensional estimation and reconstruction or to large-scale environments, new research challenges emerge immediately.

Particularly, for the three-dimensional SLAM with only visual sensors, some of the new problems that must be solved are: greater complexity and computational cost due to the large volume of data to be processed, errors due to low resolution of the sensors, lighting changes in the environment, surfaces with lack of texture and blurry images due to rapid movements of the camera.

In this thesis a systematic and rigorous study of the SLAM is carried out, from its formulation and solution methods, until the evaluation of some of the most recent open source SLAM algorithms. In particular, the monocular SLAM problem in real time is addressed and experiments are conducted in indoor environments with a robotic system specially designed for this purpose. The main contributions of this work are:

- A systematic and exhaustive study of SLAM, from its formulation to the most representative solution methods, is conducted (Chapter 2).
- The formulation of a method for evaluating SLAM algorithms based on two metrics (MeC y MoC) that consider the quality of the maps produced (Chapter 3).
- The design and construction of a robotic system with full compatibility with ROS (*Robot Operating System*) for the experimental validation conducted in this thesis and research and development of applications (Chapter 3).
- The rigorous study of visual SLAM algorithms that are part of the current state of the art and, in particular, the methods related to SfM (Structure from Motion) and real-time monocular SLAM (Chapters 4 and 5).

Keywords: monocular SLAM, visual SLAM, Mobile robotics, Extended Kalman filter, Particle filter, Robotic localization, Robotic mapping, ROS, SfM, 3D reconstruction.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

SLAM monocular en tiempo real

1.1. Introducción

Algunas de las tareas que definen a un robot móvil como una entidad autónoma son la navegación segura de su entorno (*obstacle avoidance*) y el planeamiento de trayectorias. Para llevar a cabo una ejecución exitosa de estas tareas, el robot debe estar en capacidad de estimar de manera confiable su localización $(x, y, \theta)^T$ y elaborar representaciones relativamente precisas de su entorno (*mapping*).

Los métodos clásicos empleados en la solución del SLAM se basan en el uso de técnicas probabilísticas, en las cuales, los parámetros y las medidas se tratan como variables aleatorias, y la incertidumbre en la estimación de la localización se modela mediante funciones de densidad de probabilidad (*probability density functions PDF*).

Al problema de determinar la estructura tridimensional de un objeto a partir del análisis de un conjunto finito de imágenes de una escena, capturadas a intervalos regulares de tiempo, se le conoce como *Structure from Motion (SfM)*. El incremento en la capacidad de cómputo de los sistemas microprocesados y el desarrollo de algoritmos más eficientes, han permitido el surgimiento en años recientes de sistemas robóticos, sistemas de computación vestible, sistemas de realidad aumentada y aplicaciones de mapeo virtual basadas en el paradigma SfM [2].

En este trabajo se tratarán los problemas de la localización y el mapeo considerando como sensor principal una cámara estándar y el empleo de técnicas probabilísticas para tratar con la incertidumbre y el error asociado a los sistemas de percepción.

1.2. Motivación

En este trabajo se aborda el problema del SLAM desde un contexto diferente a la aproximación clásica basada en el uso filtros probabilísticos para estimar la localización de robots y marcas (*landmarks*) en un entorno de trabajo.

La solución clásica del SLAM requiere de diferentes fuentes de información o sensores para reducir la incertidumbre en la estimación de la localización. Usualmente se emplean sistemas odométricos para estimar la localización del robot y se mejora esta estimación mediante medidas exteroceptivas que permiten determinar la localización de marcas presentes en el entorno [3].

En este trabajo se propone estimar la posición de un robot móvil con base en características extraídas de un conjunto finito de imágenes suministrada por una cámara estándar montada sobre el robot y elaborar representaciones tridimensionales de los sitios visitados.

Al problema de estimar incrementalmente el movimiento de una cámara y la estructura de la escena partir de un conjunto de imágenes proporcionadas por la cámara se le denomina SLAM Monocular o SfM incremental. Se propone usar técnicas probabilísticas para modelar la incertidumbre asociada a la medida de características extraídas del conjunto de imágenes.

1.3. Marco teórico

Una de las tendencias actuales en el ámbito de la robótica móvil ha sido el empleo de dispositivos de visión como sistema de percepción principal para dar solución a los problemas de localización, mapeo y navegación de robots móviles [2], en lugar de la aproximación clásica de combinar sistemas de rango láser para la medida de distancias (*Laser Range Finder*) con mapas bidimensionales del entorno.

Varios autores han propuesto técnicas de procesamiento diferentes a la tradicional aproximación basada en la identificación y reconstrucción de características. Civera *et al.* en [4] presentan un método basado en una combinación de técnicas de visión de bajo costo computacional a las que denominan "*lightweight vision*", con las que realizan el mapeo y navegación autónoma de un robot móvil en entornos interiores de gran escala. Emplean una combinación de odometría y un anillo de tres cámaras

para la evasión de obstáculos, el mapeo de espacio libre y la detección de bucles cerrados (*loop closure detection)*.

Tal como se establece en [3] y [5], es posible determinar el movimiento propio (*egomotion*) de un sensor que se mueve a través de una escena a partir de múltiples medidas de una misma entidad extraídas de un conjunto finito de observaciones. Se considera la escena como estática. La extracción de múltiples medidas permite también extraer características geométricas de la escena.

El uso de cámaras estándar como sistema de percepción principal para abordar los problemas de navegación y mapeo en robótica móvil presenta nuevos retos, como la detección robusta de características, la asociación de datos y la eficiencia computacional en la estimación de estados (*espacio de configuración*) en entornos de grandes dimensiones [3].

Es bien conocido que los algoritmos SLAM basados en filtros probabilísticos se implementan en la actualidad de manera rutinaria en un buen número de robots comerciales y experimentales [6]. Algunos autores consideran el problema del SLAM 2D con lidar para entornos interiores de dimensiones reducidas un problema ya resuelto [7], no siendo así cuando se consideran entornos interiores y exteriores de grandes dimensiones o sistemas que emplean información visual.

Solo recientemente la comunidad científica viene llevando a cabo importantes esfuerzos para desplazar aquellos sensores tradicionalmente empleados para resolver el SLAM por cámaras. Las cámaras ofrecen ventajas como bajo costo, tamaño y peso reducidos y alta disponibilidad. Son dispositivos no invasivos, proveen gran cantidad de información, y constituyen sistemas altamente biomiméticos e intuitivos por cuanto la visión constituye el principal sistema de navegación para el hombre y para muchas especies animales [6].

Ante la dificultad de extraer características de imágenes individuales que permitan obtener mapas confiables de grandes dimensiones en tiempo real a las ratas de muestreo de las cámaras actuales, algunos investigadores [6], [4] han propuesto resolver los problemas de reconstrucción mediante medidas de características extraídas de pequeños conjuntos de imágenes, técnica conocida como *Structure of Motion (SfM)*.

1.3.1. Visual SLAM

El mapeo y localización en entornos 3D, en los cuales se pueden llegar a considerar hasta seis grados de libertad (6DOF), constituye un problema en el cual el tipo de sensores predominantes son las cámaras. Esto se debe al hecho de que las cámaras proveen un gran volumen de información que permite la elaboración de representaciones 3D del entorno. La apariencia visual de estas representaciones, resulta ser más significativa e intuitiva desde la perspectiva del ser humano, que aquellas obtenidas de datos suministrados por sensores de rango láser y sonares [8].

El término *Visual SLAM* o *Monocular SLAM* se refiere al empleo de una cámara como único sensor en la solución del SLAM, sin considerar esquemas de control activo sobre el movimiento de la cámara. Por varias décadas los métodos tradicionales para estimar la posición de un robot y actualizar el mapa del entorno se basaron en el uso de filtros secuenciales probabilísticos, que permiten ajustar la función de densidad de probabilidad empleada para estimar la localización del robot con base en observaciones exteroceptivas realizadas sobre el entorno [9].

Los métodos basados en filtros probabilísticos emplean una combinación de diferentes tipos de sensores como láser, sonar, odómetros y cámaras para resolver el problema del SLAM, mientras que con el SLAM monocular es posible emplear una sola cámara para determinar la localización de un robot y construir mapas del entorno en tiempo real.

No obstante, y aparte de las evidentes ventajas de usar un único sensor, el SLAM monocular plantea una serie de desafíos significativos como son la dificultad para predecir el movimiento de la cámara entre imágenes consecutivas (*frames*), el procesamiento de imágenes en tiempo real a altas ratas de muestreo del sensor (*frames per second, fps)* y el hecho de que la información de profundidad no es directamente observable a partir de las imágenes suministradas por una sola cámara. La información de profundidad, a no ser que se emplee una cámara RGB-D, solo puede ser establecida a partir de los cambios de una o varias características en una secuencia de imágenes [9], [10].

Una contribución significativa, propuesta por Davison en [8], y que denominó MonoSLAM, consistió en adaptar la aproximación probabilística EKF (*Extended Kalman Filter*) al problema del SLAM monocular para estimar simultáneamente el

estado de la cámara y las características de interés en una secuencia de imágenes, junto con la incertidumbre en la estimación de tales estados [10], [11].

Uno de los problemas reconocidos al emplear el estimador de estado EKF, son los errores acumulativos introducidos por la linealización. La incertidumbre en los estados de la cámara crece respecto al marco de referencia, a medida que la cámara se desplaza. Este incremento de la incertidumbre invalida la linealización llevada a cabo por el estimador EKF [12], [13]. La estimación mejora significativamente cuando el marco de referencia se desplaza al centro de la cámara, básicamente debido a la cercanía de los elementos de interés en la imagen en comparación con un marco de referencia externo [13].

Como se aprecia en la figura 1.1, se requieren múltiples medidas de distancia (posiblemente provistas por un lidar) para estimar la posición de la marca (*landmark*), la cual se localiza en la intersección de los arcos obtenidos a partir de cada medida. De manera análoga, para estimar la posición de la marca con base en una cámara se requieren múltiples medidas, sin embargo en vista de que estas no proveen información de profundidad, entonces se emplean múltiples medidas de proyección de la marca sobre la escena [14].

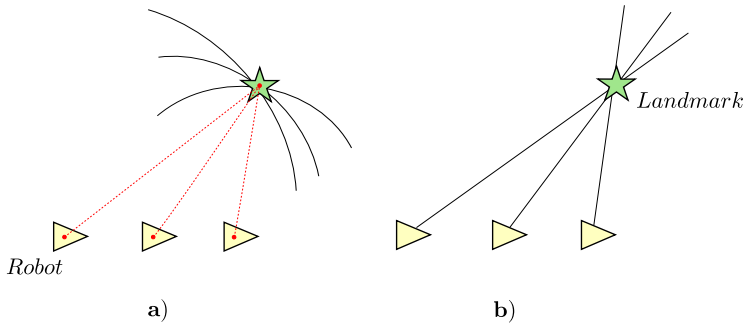


Figura 1.1 Múltiples observaciones parciales permiten estimar la localización de la marca a partir de medidas de **a**) distancia y **b**) de proyección de la marca sobre la escena.

En virtud de que una observación parcial no permite determinar la posición de la marca, el robot debe desplazarse para recopilar información desde diferentes puntos, por lo que el desplazamiento del robot resulta esencial en la elaboración de mapas basados en SfM.

Con base en las imágenes captadas por una cámara, es posible llevar a cabo medidas sobre la proyección de diferentes objetos sobre la escena y modelar la

incertidumbre en la medida mediante funciones de densidad de probabilidad. El estado actual de la cámara y el entorno corresponde a un vector $\mathbf{x}$ en el cual se almacenan parámetros de la cámara y del mapa. Cada proyección *i* medible sobre la escena, la cual se denomina característica (*feature* o *landmark*), da origen a un vector de parámetros $\mathbf{z_i}$ al que se asocia una función de probabilidad condicional $\mathbf{p}(\mathbf{z_i} \mid \mathbf{x})$ que modela la incertidumbre en el proceso de medida [15].

Cuando una nueva imagen es obtenida, es posible proyectar la distribución de probabilidad del vector $\mathbf{x}$ sobre el espacio de características y estimar la localización de aquellas características de interés mediante la ecuación 1.1:

$$p(\mathbf{z}_T) = \int p(\mathbf{z}_T \mid \mathbf{x}) p(\mathbf{x}) d\mathbf{x}, \tag{1.1}$$

donde $\mathbf{z}_T = (z_1\ z_2\ \ldots)^T$ es el vector de características que contiene además la función de probabilidad condicional $p(\mathbf{z}_T \mid \mathbf{x})$.

1.4. Planteamiento del problema

El SLAM, o problema de la construcción de modelos del mundo y la estimación simultánea de la localización en dichos modelos mediante uno o varios robots autónomos, ha sido un problema que ha despertado un notable interés en la comunidad científica. Se busca obtener mapas precisos del entorno que permitan estimar de manera confiable la localización del robot, sin embargo una de las dificultades inherentes al SLAM es el de la incertidumbre relativa al desplazamiento del robot, la cual se debe a fenómenos de deslizamientos y asimetrías en los dispositivos de locomoción y la incertidumbre en las medidas del entorno, como por ejemplo la distancia a obstáculos circundantes.

La aproximación clásica a la solución del SLAM ha sido el empleo de una combinación de diferentes dispositivos de percepción como láser, sonar, GPS, odometría y unidades inerciales de medida, entre otros, junto con métodos probabilísticos para la estimación de los espacios de estado del robot y el entorno.

Si bien el problema del SLAM se considera totalmente resuelto cuando se elaboran representaciones 2D de entornos interiores de dimensiones reducidas, sigue presentando retos importantes cuando se considera el mapeo de entornos interiores y

exteriores de grandes dimensiones y la construcción de mapas tridimensionales con base en un único sensor. Específicamente, cuando se aborda la solución del SLAM 3D mediante el empleo de una cámara estándar como sensor principal, surgen nuevos problemas que deben ser solucionados:

- Para la estimación y seguimiento de la localización de una cámara estándar, de peso y dimensiones reducidas, se requieren seis grados de libertad *(6DOF)*, tres para la posición *(x, y, z)* y tres para la orientación *(pitch, yaw, roll)*.
- Simultáneamente a la estimación de la localización, se debe determinar la estructura 3D del entorno circundante.
- El mapeo y localización se deben llevar a cabo en tiempo real, lo que para una cámara estándar corresponde a un rata de 30 *fps*.

Una aproximación a la solución del SLAM monocular consiste en aprovechar los avances y desarrollos llevados a cabo en el campo de la visión por computador, algoritmos SfM, SLAM visual, estimadores de estado y la reconstrucción tridimensional de objetos.

1.5. Preguntas de investigación

Enmarcado en el problema previamente identificado se formula la pregunta de investigación:

¿Cómo integrar algunos de los métodos SfM, visual SLAM y la estimación de la localización en una plataforma móvil que permita construir representaciones tridimensionales del entorno mediante un único sensor correspondiente a una cámara estándar?

A partir de la pregunta formulada, surgen interrogantes complementarios como:

- ¿Qué arquitectura de hardware y software se puede implementar para procesar la información proporcionada por una cámara a una rata de 30 *fps*?
- ¿Cuales deberán ser las características de la plataforma móvil mediante la cual se pretende explorar un entorno de trabajo?

- ¿Qué estrategias de exploración se pueden implementar para recorrer el entorno de trabajo?
- ¿Qué algoritmos emplear para extraer la información de interés de un conjunto finito de imágenes suministradas por una cámara?
- ¿Qué algoritmos emplear en la construcción de representaciones tridimensionales del entorno?

1.6. Hipótesis

Con base en el planteamiento de la pregunta de investigación principal se establece la siguiente hipótesis:

El uso métodos SfM y visual SLAM para la extracción de características de un conjunto finito de imágenes proporcionadas por una única cámara estándar, en combinación de métodos probabilísticos para el modelamiento de la incertidumbre en la medida, constituyen una alternativa de solución al problema del SLAM.

1.7. Objetivos

1.7.1. Objetivo general

Aplicar métodos para la extracción de información de un conjunto finito de imágenes proporcionados por una cámara monocular y métodos probabilísticos de estimación de la localización para la elaboración de representaciones tridimensionales de entornos de trabajo.

1.7.2. Objetivos específicos

- Estudiar los métodos probabilísticos clásicos para solucionar el problema del SLAM.
- Estudiar los métodos SfM y visual SLAM para la extracción de información de imágenes proporcionadas por una cámara monocular estándar.

- Integrar en una plataforma móvil algunos desarrollos en el campo de la visión computacional y un método de estimación de localización que permitan construir representaciones del entorno próximo.
- Validar a nivel experimental un método propuesto para la elaboración tridimensional de representaciones del entorno.

1.8. Metodología de trabajo

Se propone una metodología basada en el empleo de bases de datos y el diseño de pruebas experimentales que permitan recabar datos para la validación de los algoritmos de estimación de posición y mapeo del entorno. Se asume la evidencia teórica y experimental como aspecto orientador de los procesos de razonamiento que permiten adquirir una mayor comprensión de los fenómenos asociados al objeto de estudio. Se aborda la resolución de los problemas con base en los niveles de comprensión alcanzados.

Capítulo 2

Planteamiento y solución del SLAM

2.1. Introducción

Uno de los mayores retos que enfrenta la investigación en robótica es el de incorporar diferentes niveles de autonomía a los sistemas robóticos, entendiéndose autonomía como la capacidad de un sistema para llevar a cabo sus propios procesos y operaciones sin ningún tipo de intervención humana.

La navegación es probablemente el problema más estrechamente vinculado a la autonomía de los robots móviles, el cual a su vez puede dividirse en cinco subproblemas, cada uno de los cuales otorga un nivel de abstracción jerárquico en lo relativo a la autonomía de los sistemas robóticos, son estos: el mapeo del entorno, la localización, la planificación y la generación y seguimiento de trayectorias [16], [17].

En robótica móvil, el término "*mapeo*" hace referencia a la construcción de mapas o representaciones del entorno. Sin embargo, a diferencia de lo que usualmente las personas consideran como mapas, desde la perpectiva de la robótica un mapa no es más que una estrutura de datos que asocia a un conjunto de medidas (procesadas o en bruto) un conjunto de ubicaciones. La utilidad de las representaciones computacionales radica en el hecho de que le permiten a los robots establecer su localización con relación a su entorno de trabajo, con lo cual logran incrementar notablemente sus niveles de autonomía y robustez en la navegación [18].

Por otra parte, el problema de la localización busca dar respuesta a la pregunta *¿Donde me encuentro?* y se refiere a la capacidad de un robot móvil para establecer su ubicación en el mundo haciendo uso de su sistema de percepción [19]. Una vez el robot establece su localización, puede decidir a donde dirigirse o que hacer. Los términos *ubicación*, *pose* y *localización* se refieren a las coordenadas y orientación de un robot respecto a un sistema global de representación del espacio [20].

El mapeo y la localización son dos problemas estrechamente relacionados. Cuando no se dispone de una representación *a priori* del entorno, ésta deberá ser construida de manera autónoma por el robot. Esto supone un problema nada trivial, dado que para poder construir modelos del entorno se debe poder establecer una localización precisa con relación a un modelo del que aún no se dispone. De igual modo, para poder obtener una buena estimación de la localización se requiere de un mapa.

Por consiguiente, cuando un robot se enfrenta a un entorno desconocido, tendrá que resolver simultáneamente dos problemas que no pueden ser resueltos de manera independiente, la localización y el mapeo. Estos dos problemas se encuentran tan fuertemente correlacionados que no es posible establecer cual de ellos es causa y cual efecto. La necesidad de resolver concurrentemente ambos problemas da origen al problema de la *Localización y Mapeo Simultáneo* o *SLAM*, por sus siglas en inglés.

El nivel de autonomía de un robot que incorpora módulos para la estimación de la localización y el mapeo se evidencia en su capacidad de afrontar exitosamente situaciones complejas como:

- Navegación segura de entornos desconocidos con elaboración y actualización de modelos computacionales
- Navegación en entornos dinámicos en el que personas u objetos se desplazan o alteran su posición
- Planeamiento y seguimiento de trayectorias con evasión de obstáculos imprevistos o no modelados previamente

Normalmente, la información espacial se deriva de la percepción directa del entorno a través de sensores exteroceptivos. Además, los sensores propioceptivos, como la odometría o las unidades de medición inercial proporcionan información sobre el cambio de ubicación dentro del entorno. Sin embargo, existen otras alternativas para la adquisición información espacial, incluidas representaciones externas

como planos, bocetos o descripciones escritas, así como la comunicación directa con otros robots o con seres humanos [16], [18].

2.2. Tipos de Mapas

Una primera clasificación taxonómica de las representaciones computacionales usadas en robótica móvil, se establece entre mediados de la década de 1980 e inicios de la década de 1990. Durante este período dos paradigmas dominaron la escena de la investigación en el mapeo de entornos: los mapas métricos y los topológicos [21].

En las representaciones métricas se almacenan propiedades geométricas del entorno, siendo una de las primeras representaciones de este tipo el algoritmo de ocupación de cuadrículas o celdas propuesto por Moravec y Elfes [22]. Usualmente un mapa de ocupación de celdas representa el entorno como un bloque de cuadrículas a las que se asocia un valor de probabilidad de ocupación de cada celda. La información espacial para la actualización del valor de ocupación de cada celda se deriva de los sensores exteroceptivos del robot, mientras que la información sobre el cambio de ubicación del robot en su entorno se obtiene de la odometría.

Por otro lado las representaciones topológicas describen el entorno como una lista de sitios (*nodes*) con características relevantes conectados por arcos o aristas (*edges*). Se suele asociar a los arcos información relativa a la navegación entre nodos, comúnmente geométrica, por lo que la línea divisoria entre mapas métricos o topológicos se ha ido difuminando con el tiempo. En la figura 2.1 se aprecian dos mapas, uno métrico y otro topológico.

2.3. Formulación del problema del SLAM

Los primeros trabajos relativos al SLAM, en los que se aborda de manera sistemática el problema, se remontan mediados e inicios de las décadas de 1980 y 1990 respectivamente. Autores como Smith, Cheeseman y Shelf [23], [24], Crowley [25], Chatila y Laumond [26], Leonard y Durrant-Whyte [27] incorporaron métodos probabilísticos para tratar la incertidumbre sensorial y mejorar la estimación conjunta tanto de la posición del robot como de las marcas del entorno.

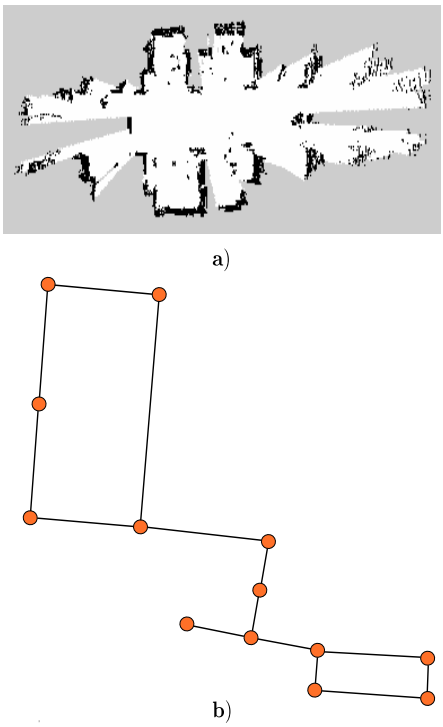


Figura 2.1 Dos tipos de modelos del entorno, **a**) un mapa métrico y **b**) uno topológico.

El SLAM puede considerase como un proceso en el cual un vehículo autónomo ubicado en una localización desconocida, en un sitio desconocido, describe (*mapea*) su entorno usando únicamente observaciones relativas a su localización y estima su trayectoria con relación a las observaciones realizadas [28]. El modelo del entorno o mapa, se construye incrementalmente con base en la información provista por los sensores propio y exteroceptivos con los que cuenta el robot. Tanto la posición del robot como la posición de las características observadas se estiman empleando algún tipo de estimador, por ejemplo un filtro de Kalman, uno de partículas o un filtro H_∞ [29].

En la formulación clásica del SLAM se suele considerar un escenario como el que se presenta en la figura 2.2. Un vehículo autónomo (*robot*), del cual se conoce su modelo cinemático, es ubicado en una posición y entorno desconocidos. Mediante sus sensores, el robot identifica y mide la distancia relativa a un número desconocido de características. A medida que navega su entorno, las distancias a características previamente detectadas y aquellas nuevas, son actualizadas por el módulo de percepción.

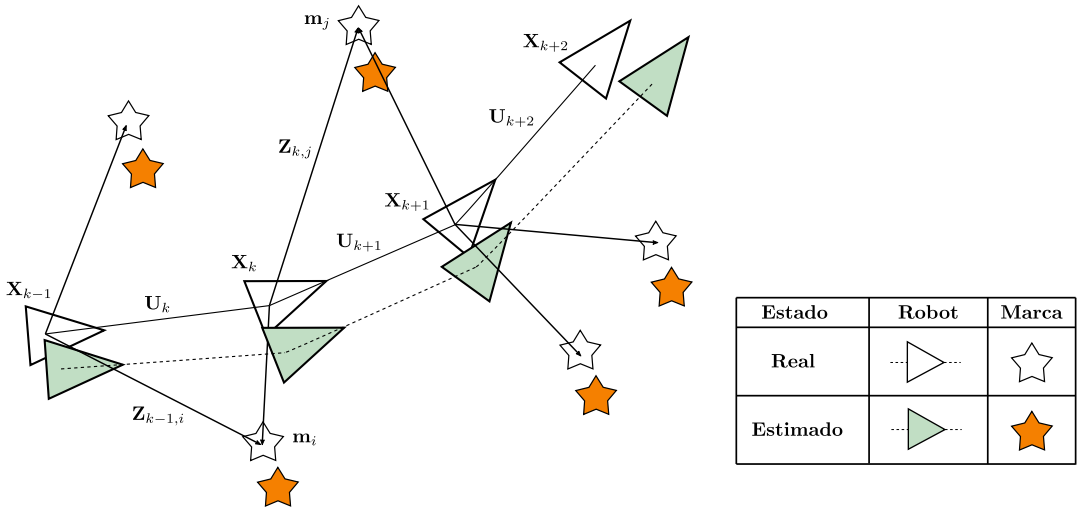


Figura 2.2 Problema del SLAM, en el que se debe llevar a cabo una estimación simultánea del estado del robot y de las características del entorno [1].

Con relación a la figura 2.2 se definen, en cada instante de tiempo k, las siguientes cantidades:

- $\mathbf{x}_k$: vector de estado que describe la localización y orientación del robot
- $\mathbf{u}_k$: vector de control. Una vez se aplica la acción de control en el instante de tiempo $k-1$, el robot es conducido del estado $\mathbf{x}_{k-1}$ al estado $\mathbf{x}_k$
- $\mathbf{m}_i$: vector que contiene la localización de la *i-ésima* característica. Se asume que la localización de las características es invariante en el tiempo
- $\mathbf{z}_{ik}$: observación o medida de la *i-ésima* característica tomada en el instante k por el sistema de percepción del robot desde la posición $\mathbf{x}_k$
- $\mathbf{z}_k$: una observación genérica efectuada en el instante de tiempo k

Así mismo se definen los siguientes conjuntos:

- Conjunto de los vectores de estado o posiciones del robot hasta el instante k:

$$\mathbf{X}^k = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k\} = \{\mathbf{X}^{k-1}, \mathbf{x}_k\} \tag{2.1}$$

- Conjunto de los vectores de control enviados al robot hasta el instante k:

$$\mathbf{U}^k = \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\} = \{\mathbf{U}^{k-1}, \mathbf{u}_k\} \tag{2.2}$$

- Mapa del entorno que contiene una lista de características estáticas:

$$\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_n\} \tag{2.3}$$

- El conjunto de todas las observaciones o medidas hasta el instante k:

$$\mathbf{Z}^k = \{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_k\} = \{\mathbf{Z}^{k-1}, \mathbf{z}_k\} \tag{2.4}$$

Se asume que se cumplen las siguientes hipótesis: 1) no se dispone de información previa sobre la ubicación de las marcas $\mathbf{m}$ del entorno, 2) la posición inicial $\mathbf{x}_0$ es conocida y, 3) también se conocen la secuencia de control $\mathbf{U}^k$ y las medidas $\mathbf{Z}^k$. Siendo así, el SLAM busca determinar en cada instante de tiempo k tanto la posición $\mathbf{x_k}$ del robot como la de las marcas observadas $\mathbf{m}$, conociendo las medidas $\mathbf{Z}^k$ y el conjunto de comandos $\mathbf{U}^k$ aplicados al robot.

2.3.1. El problema de la localización

Consiste en estimar la posición de un robot a partir del histórico de comandos (ecuación 2.2), el conocimiento previo de su entorno (ecuación 2.3) y el histórico de las observaciones realizadas (ecuación 2.4). Analíticamente, el problema de la localización consiste en estimar la distribución de probabilidad dada por la ecuación 2.5 [30].

$$P(\mathbf{x}_k \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{m}) \tag{2.5}$$

Una estimación robusta o *localización global* se consigue al incorporar todo el histórico de comandos y observaciones en el cálculo de la probabilidad de la ecuación 2.5, sin embargo la complejidad computacional se incrementa notablemente. Una simplificación frecuente consiste en usar los datos del instante previo $k-1$ para estimar la posición en el instante k. Esto se conoce como una *localización local* y se expresa analíticamente mediante la ecuación 2.6.

$$P(\mathbf{x}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^{k-1}, \mathbf{m}) \tag{2.6}$$

Si bien de esta manera se simplifica notablemente la complejidad del algoritmo, se genera el riesgo de que la localización estimada difiera significativamente de la localización real del robot.

2.3.2. El problema del mapeo

El problema del mapeo consiste en construir un modelo del entorno haciendo uso de los históricos de las posiciones del robot (ecuación 2.1), las observaciones (ecuación 2.4) y los comandos aplicados al robot (ecuación 2.2). La formulación analítica del problema del mapeo se expresa mediante la ecuación 2.7.

$$P(\mathbf{m} \mid \mathbf{X}^k, \mathbf{Z}^k, \mathbf{U}^k) \tag{2.7}$$

2.3.3. Formulación probabilística del SLAM

La ecuación 2.8 corresponde a la definición probabilística del problema del SLAM.

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) \tag{2.8}$$

En general, la solución es de naturaleza iterativa o recurrente. Se comienza con la distribución de probabilidad $P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^{k-1})$ en el instante $k-1$ y el cálculo de la distribución *a posteriori* $P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k)$ a partir de $\mathbf{Z}^k$, $\mathbf{U}^k$ y el teorema de Bayes [31]. Para el cálculo de la distribución es necesario conocer las cantidades:

$$P(\mathbf{Z}^k \mid \mathbf{X}^k, \mathbf{m}) \tag{2.9}$$

$$P(\mathbf{X}^k \mid \mathbf{X}^{k-1}, \mathbf{U}^k) \tag{2.10}$$

La ecuación 2.9 se conoce como el *modelo de observación* y define la probabilidad de realizar una medida $\mathbf{Z}^k$ cuando se conoce el estado $\mathbf{X}^k$ del robot y el mapa $\mathbf{m}$ del entorno. Resulta razonable asumir que una vez definidos el estado del robot y el mapa, las observaciones son condicionalmente independientes entre sí y solo dependen de $\mathbf{X}^k$ y $\mathbf{m}$ [31], esto es:

$$P(\mathbf{Z}^k \mid \mathbf{X}^k, \mathbf{m}) \;=\; \prod_{i=1}^{k} P(\mathbf{z}_i \mid \mathbf{X}^k, \mathbf{m}) \;=\; \prod_{i=1}^{k} P(\mathbf{z}_i \mid \mathbf{x}_i, \mathbf{m}) \tag{2.11}$$

De otra parte, la ecuación 2.10 se denomina *modelo de movimiento* y permite predecir el estado $\mathbf{X}^k$ del sistema. Puesto que la estimación del nuevo estado solo

depende del estado previo $\mathbf{X}^{k-1}$ y del comando de control $\mathbf{U}^k$ aplicado al robot, entonces se dice que la transición de estados corresponde a un proceso de *Markov*.

El problema del SLAM (ecuación 2.8) se puede resolver de manera recursiva en dos partes, una que permite actualizar la posición del vehículo (*robot móvil*) y la otra el mapa, para esto se usan las definiciones y modelos discutidos anteriormente, el teorema de Bayes y la regla de la cadena para la probabilidad condicional.

- *Actualización de la observación*:
Inicialmente se expande la distribución conjunta del estado del robot, el mapa y la observación en términos del estado del vehículo y el mapa:

$$\begin{aligned} P(\mathbf{x}_k, \mathbf{m}, \mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) &= \\ P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{z}_k, \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) * P(\mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) &= \\ P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) * P(\mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) &= \\ P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) * P(\mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k) & \end{aligned} \tag{2.12}$$

Siendo el último factor en la ecuación 2.12 independiente del estado del vehículo, entonces se puede omitir el término $\mathbf{x}_0$. Ahora, expandiendo la distribución conjunta en términos de la observación se llega a:

$$\begin{aligned} P(\mathbf{x}_k, \mathbf{m}, \mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) &= \\ P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}, \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) * P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) &= \\ P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) * P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) & \end{aligned} \tag{2.13}$$

Donde el último término en la ecuación 2.13 se obtiene al considerar las asunciones relativas a la ecuación 2.11. Al igualar y despejar las ecuaciones 2.12 y 2.13 se obtiene:

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) = \frac{P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) * P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0)}{P(\mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k)} \tag{2.14}$$

Puesto que el denominador de la ecuación 2.14 es independiente tanto de la posición del robot como del mapa, entonces puede considerarse como una constante

de normalización $\eta = \dfrac{1}{P(\mathbf{z}_k \mid \mathbf{Z}^{k-1}, \mathbf{U}^k)}$, que únicamente depende de los modelos de observación y movimiento.

- *Actualización de la posición*:
El segundo factor del numerador en la ecuación 2.14, se puede expresar alternativamente en términos del modelo de movimiento del vehículo, la probabilidad posterior conjunta desde el instante $k-1$ (*proceso de Márkov*) y el teorema de la probabilidad total:

$$\begin{aligned} P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) = \int P(\mathbf{x}_k, \mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) d\mathbf{x}_{k-1} \quad = \\ \int [P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{m}, \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0) * P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^k, \mathbf{x}_0)] d\mathbf{x}_{k-1} \quad = \\ \int [P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) * P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^{k-1}, \mathbf{x}_0)] d\mathbf{x}_{k-1} \end{aligned} \tag{2.15}$$

Donde el resultado final obtenido en la ecuación 2.15, es consecuencia de las asunciones de independencia del modelo de movimiento con relación al mapa y las observaciones y su dependencia con respecto a las entradas de control. Sustituyendo la ecuación 2.15 en la 2.14 se obtiene:

$$\begin{aligned} P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) \;=\; \eta * P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) * \\ \int [P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) * P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}^{k-1}, \mathbf{U}^{k-1}, \mathbf{x}_0)] d\mathbf{x}_{k-1} \end{aligned} \tag{2.16}$$

La ecuación 2.16 corresponde a una expresión recursiva para el cálculo de la estimación conjunta *a posteriori* para el estado del robot y el mapa con base en las observaciones y las entradas de control y es función a su vez del modelo de movimiento $P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k)$ y del modelo de observación $P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m})$. La figura 2.3 explica la solución recursiva del SLAM, donde se involucran variables probabilísticas medibles y no medibles.

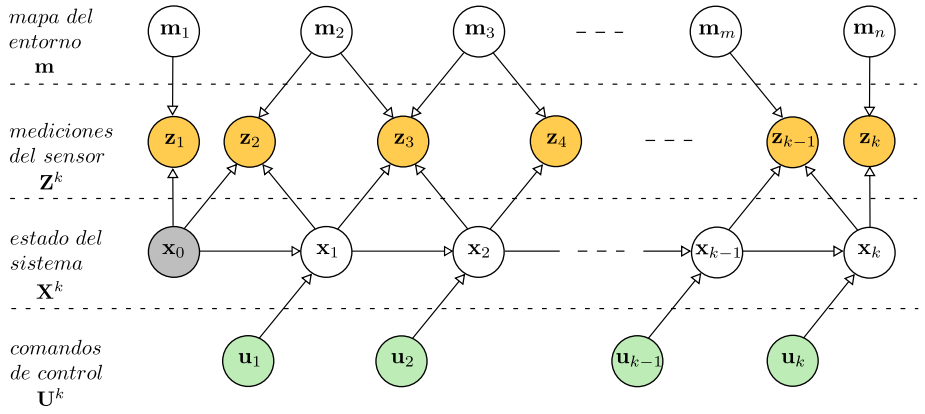


Figura 2.3 Red Dinámica Bayesiana (*RDB*) asociada al SLAM. Los nodos coloreados se refieren a la variables observables (aquellas medibles directamente) y los blancos a las no observables (aquellas que se deben estimar). El grafo muestra las dependencias entre el movimiento del robot, su estado, las mediciones de los sensores y los objetos estacionarios.

2.3.4. Algoritmos de solución del SLAM

Desde su formulación, un sinnúmero de aproximaciones para dar solución al SLAM han sido planteadas, siendo los métodos estadísticos los más relevantes. Los algoritmos disponibles admiten una amplia variedad taxonómica basada en diferentes criterios, como por ejemplo el tipo de sensores, el tipo de representación del entorno y los métodos de estimación para determinar la localización del robot y las marcas.

De acuerdo al principio de transducción o método de detección, los sensores para SLAM se clasifican en sensores acústicos, sistemas láser y sensores de visión [32], [33]. Si bien son más adecuados para entornos subacuáticos, los sensores acústicos también son frecuentemente usados en interiores, debido principalmente a su bajo costo.

En [34], se lleva a cabo una fusión sensorial de sensores ultrasónicos y una brújula magnética digital para establecer la posición de objetos en el entorno próximo a un robot. Se emplean sensores giroscópicos y un codificador para determinar la posición del vehículo. El SLAM se resuelve para entornos interiores de dimensiones reducidas. Por otro lado, una caracterización experimental de sensores ultrasónicos y un algoritmo probabilístico basado en filtros de partículas es usado en [35] para estimar la localización de un robot móvil en su entorno.

El sensor láser es probablemente el tipo de sensor más popular para resolver el problema del SLAM, principalmente por su robustez tanto en entornos interiores como exteriores [33]. Un algoritmo SLAM 2D basado en sensor láser, que modifica y optimiza el algoritmo TinySLAM para robots terrestres, haciéndolo adecuado para su uso en micro-vehículos aéreos es propuesto en [36]. Así mismo en este trabajo, se presenta una extensión al mapeo 3D. Un método basado en Graph-SLAM para la estimación de la localización de un robot en interiores se describe en [37]. Para tal fin, se emplean datos de odometría e información de características extraídas del flujo de datos de un sensor láser.

En años recientes los sensores de visión han experimentado un notable auge, no sólo en la solución del SLAM, sino también en aplicaciones de navegación autónoma de vehículos. Entre sus ventajas se destacan el gran volumen de información visual que proporcionan aún siendo sensores pasivos, además son livianos, de dimensiones reducidas y bajo costo. Sin embargo, debido a la gran cantidad de datos que proveen, los algoritmos para la extracción de información de interés imponen cargas computacionales elevadas a los dispositivos de procesamiento. Un algoritmo SLAM de gran escala en tiempo real para cámaras estéreo (*LSD-SLAM*) es propuesto en [38]. En [6] se demuestra que es posible resolver la localización y mapeo en tiempo real con una sola cámara que se desplaza libremente en el entorno, y que constituye, la única fuente de datos disponible.

En cuanto a los métodos estadísticos, los algoritmos SLAM se pueden clasificar de acuerdo al método de filtrado considerado en la estimación del estado del sistema (*robot móvil*). Los más ampliamente usados son el *EKF* (*Extended Kalman Filter*) y el filtro de partículas (*Particles Filter*) [39].

Filtro de Kalman extendido (*EKF-SLAM)*

El filtro de Kalman es un algoritmo para la estimación del estado oculto de un sistema dinámico lineal a partir de variables de estado observables (*medidas*) que se encuentran sometidas a ruido blanco Gaussiano. Puesto que el filtro de Kalman solo aplica para sistemas lineales, su variante el EKF, se ha constituido en uno de los métodos de uso más extendido en la solución de problemas de estimación probabilística para sistemas no lineales [40].

En el algoritmo SLAM basado en EKF (*EKF-SLAM*), el modelo de movimiento del robot se describe en términos de su modelo cinemático sometido a ruido no correlacionado de media cero de acuerdo a la ecuación 2.17.

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \iff \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \omega_k$$

$$\omega_k \sim \mathcal{N}(0, \mathbf{Q}_k) \tag{2.17}$$

$$P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) \iff \mathbf{z}_k = h(\mathbf{x}_k, \mathbf{m}) + \vartheta_k$$

$$\vartheta_k \sim \mathcal{N}(0, \mathbf{R}_k) \tag{2.18}$$

Donde $f(\cdot)$ describe el modelo cinemático del vehículo, $\mathbf{u}_k$ el comando de movimiento aplicado en el instante $\mathbf{x}_{k-1}$ y ω_k las perturbaciones en el movimiento del robot, que se modelan como ruido Gaussiano de media cero y covarianza $\mathbf{Q}_k$.

Por otra parte en la ecuación 2.18, $h(\cdot)$ corresponde al modelo de observación que da cuenta de las medidas realizadas por el vehículo desde la posición $\mathbf{x}_k$. El error en la medida también se modela como ruido Gaussiano de media cero y covarianza $\mathbf{R}_k$. En las ecuaciones 2.17 y 2.18 se asume que el ruido en el estado del sistema y las observaciones son independientes entre sí [41].

La simplicidad del método EKF-SLAM para aproximar el estado real del sistema (robot y marcas), se fundamenta en el hecho de que el producto y la convolución de distribuciones Gaussianas dan como resultado Gaussianas. Siendo así, si los modelos de observación (ecuación 2.9) y de movimiento (ecuación 2.10) se asumen como Gausssianos, entonces las etapas de *actualización* (ecuación 2.13) y de predicción (ecuación 2.15) dan como resultado distribuciones Gausssianas. Por lo tanto el método EKF-SLAM resuelve la ecuación 2.16 únicamente en términos de la media estimada y la covarianza de la distribución conjunta $P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0)$.

Los principales problemas que reporta la literatura con respecto al método EKF-SLAM tiene que ver con su convergencia, costo computacional y asociación de datos.

Convergencia: el método aplica para modelos de movimiento y observación "*ligeramente no lineales*" en los que aplica una aproximación lineal de primer orden. La convergencia del método se ve notablemente comprometida para modelos altamente no lineales, obteniéndose soluciones inconsistentes del SLAM [42].

Costo computacional: con cada observación, la totalidad de las marcas y la matriz de covarianza conjunta deben ser actualizadas. Esto se traduce en que la complejidad algorítmica del EKF-SLAM se incrementa cuadráticamente con el número de marcas m en el mapa, lo que se denota como $\mathcal{O}(m^2)$.

Una estrategia para reducir el costo computacional, consiste en generar pequeños mapas locales (*sub-mapping*) que posteriormente se combinan para generar un mapa final del entorno, tal y como se propone en [43], [44] y [45].

Asociación de datos: el método resulta ser bastante vulnerable al problema *loop-closure* o asociación de datos, que consiste en determinar si las observaciones actuales se corresponden con observaciones anteriores de sitios ya visitados, lo cual permite incrementar la probabilidad de la localización del robot.

El problema se torna aún más complejo cuando las características identificadas difieren de simples puntos (p. ej. esquinas o paredes) y, como resulta ser habitual, son percibidas desde diferentes puntos de vista [1].

Filtro de Partículas

Un método que aplica para sistemas no lineales y que ha ganado popularidad en años recientes es el *Filtro de Particulas* (PF). Este método permite resolver problemas de estimación no lineales que involucran ruido no Gaussiano o multimodal en el modelo de movimiento [46], [47].

Propuesto inicialmente por Montemerlo [48], y también conocido como método recursivo de muestreo de *Monte-Carlo*, el PF permite reducir la complejidad computacional del EKF-SLAM factorizando la ecuación 2.8 de la siguiente manera:

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) = \underbrace{P(\mathbf{x}_k \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0)}_{est.\ de\ tray.\ (PF)} \underbrace{\prod_{i=1}^{M} P(\mathbf{m} \mid \mathbf{x}_k, \mathbf{Z}^k)}_{est.\ de\ marcas\ (EKF)}, \tag{2.19}$$

donde M corresponde al número de marcas observadas hasta el momento actual.

La ecuación 2.19 permite desacoplar el SLAM: una vez conocida la trayectoria del robot (*estimada con el PF*), cada marca observada puede ser procesada individualmente, como una actualización de la medida desde una posición conocida mediante un EKF [1]. Al algoritmo propuesto en la ecuación 2.19 se le conoce como FastSLAM.

El concepto básico del PF consiste en aproximar la función de densidad de probabilidad de la trayectoria del robot mediante N hipótesis (*partículas*) de acuerdo a la expresión:

$$P(\mathbf{x}_k \mid \mathbf{Z}^k, \mathbf{U}^k, \mathbf{x}_0) \approx \sum_{i=1}^{N} \omega^{(i)} \delta(x_k - x_k{}^{(i)}) \tag{2.20}$$

En la ecuación 2.20, cada partícula $\{x_k^i\}_{i=1}^N$ se pondera mediante un peso $\omega^{(i)}$ acorde a la probabilidad de las medidas y se localiza mediante la función Delta de Dirac en $\delta(x_k - x_k{}^{(i)})$. En general, puesto que no es posible extraer muestras directamente de la distribución *a posteriori*, entonces las muestras se extraen de una distribución más simple $q(x_k^{(i)} \mid Z^k, U^k, x_0)$ denominada *distribución propuesta* o *densidad de importancia* [49], [47].

Como se ha mencionado, el filtro de partículas constituye el núcleo del algoritmo FastSLAM, que actualmente cuenta con dos versiones, FastSLAM 1.0 y FastSLAM 2.0, siendo esta última la más eficiente [1]. Algunas de las ventajas más relevantes del FastSLAM son: 1) aproximación no lineal para el modelo de movimiento, 2) complejidad algorítmica menor a la del EKF-SLAM, la cual se puede mejorar hasta $\mathcal{O}(N\log(M))$ y 3) el algoritmo lineal Gaussiano FastSLAM con $N = 1$ partículas converge al mapa real si se observan todas las marcas una infinidad de veces y si la ubicación de al menos una de ellas se conoce de antemano [50].

Dos problemas significativos del PF son la *degeneración de partículas* y el *empobrecimiento de la muestra*. Con relación al primero, se sabe que después de pocas iteraciones, el peso se concentra en unas cuantas partículas, mientras que la gran mayoría tendrán pesos despreciables. Esto conlleva a una degeneración de la muestra, requiriendo de un gran esfuerzo computacional para la actualización de las contribuciones de las partículas con pesos casi nulos. El problema se ha enfrentado llevando a cabo un proceso de *re-muestreo* (*resampling*), en el que aquellas partículas con pesos muy bajos son descartadas, mientras que aquellas con altos pesos se mantienen y multiplican [51], [52].

No obstante, el re-muestreo conduce a que la mayoría de las nuevas partículas sean descendientes de solo unas cuantas, precisamente aquellas con los pesos más significativos, mientras que las demás no generarán descendencia. Esto provoca una falta de diversidad de partículas o empobrecimiento de la muestra, que conlleva a que las partes más antiguas de la trayectoria se hagan deterministas y se genere una pérdida de información de correlación entre las marcas observadas anteriormente [51].

La solución empleada para resolver el dilema *degeneración-empobrecimiento* consiste en llevar a cabo el re-muestreo solo en pasos selectivos, cuando la diversidad de partículas es inferior a un umbral predeterminado. Para tal fin se ha establecido el criterio de *Tamaño de Muestra Efectivo* o *ESS* (*Effective Sample Size*), el cual viene dado por [53]:

$$ESS = \left(\sum_{i=1}^{N_t} \left(\omega^{(i)} \right)^2 \right)^{-1} \tag{2.21}$$

La ecuación 2.21 es considerada como una medida de la degeneración de la muestra en donde *ESS* toma valores en el rango$1 \leq ESS \leq N_t$. El re-muestreo se hace solo cuando *ESS* es menor a un umbral N_H, p. ej. $N_H = N_t/2$.

GraphSLAM

Una representación alternativa de la RDB de la figura 2.3 es la denominada *Belief Net* (BN) o *Graph-based* (GB), que corresponde básicamente a un modelo gráfico en el que se representa un conjunto de variables aleatorias y sus dependencias condicionales y que resalta la estructura espacial subyacente al SLAM [54].

El proceso de construcción de una BN que representa de manera explícita las dependencias de las variables consideradas en el SLAM (trayectoria del robot, localización de marcas y observaciones), y que utiliza toda la historia del proceso para optimizar dicha representación se conoce como método *GraphSLAM*.

El modelo obtenido al aplicar el método GraphSLAM corresponde a una BN (diagrama *pose-graph*) en el que las posiciones en la trayectoria del robot $(\mathbf{X}_k, \mathbf{m})$ se representan como nodos. Las observaciones $\mathbf{Z}^k$ o la odometría se representan como arcos entre nodos y corresponden a las mediciones en bruto (*raw measurements*) proporcionadas por los sistemas de percepción del robot [54], [55].

Sea $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ una notación para un grafo como el de la figura 2.4, donde $\mathcal{V} = \{\mathbf{x}_i\}$ y $\mathcal{E} = \{\mathbf{z}_{ij}\}$ son respectivamente los conjuntos de nodos (*vértices*) y arcos usados para la construcción de tal representación. Dadas dos posiciones consecutivas $\mathbf{x}_i$, $\mathbf{x}_j$ en la trayectoria del robot, con $\mathbf{z}_{ij}$ denotando las medidas de odometría entre ambos nodos y $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ la predicción o transformación (cambios en desplazamiento y dirección de un nodo a otro), se define una función de error mediante 2.22 [56]:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{e}_{ij}(\mathbf{x}) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j) \tag{2.22}$$

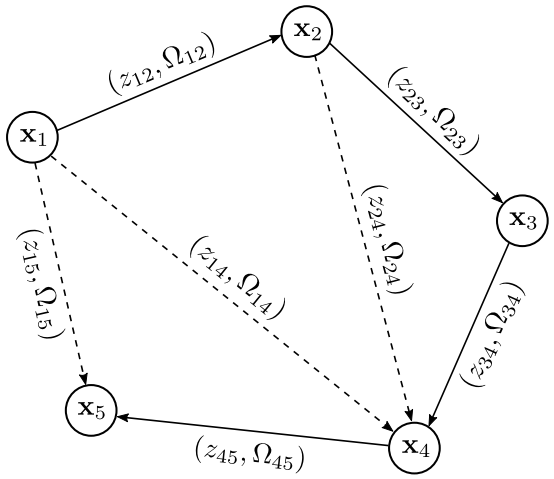


Figura 2.4 GraphSLAM. Ejemplo de representación *pose-graph* o grafo de posiciones. Cada nodo corresponde a una posición del robot. Posiciones contiguas se unen mediante un arco que modela las restricciones espaciales entre posiciones del robot surgidas de las medidas. Las líneas continuas denotan medidas secuenciales y las discontinuas representan restricciones de cierre de bucle.

La ecuación 2.22 calcula la diferencia entre la observación esperada y la observación real captada por el robot. En tal sentido GraphSLAM puede ser formulado como un problema de optimización cuyo objetivo es encontrar una configuración de nodos $\mathbf{x}^*$ que minimice el error cuadrático $\mathbf{F}(\mathbf{x})$ de todas las observaciones dado el conjunto $\mathcal{V}$ [57], [58], esto es:

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{V}} \underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{F}_{ij}}$$

$$\mathbf{x}^* = \arg\min_x \mathbf{F}(\mathbf{x}) \tag{2.23}$$

donde el término $\mathbf{\Omega}_{ij}$ en la ecuación 2.23 se conoce como la *matriz de información* y representa el ruido Gaussiano en la medida del sensor entre los nodos $\mathbf{x}_i$ y $\mathbf{x}_j$. Una vez reducido a un problema de mínimos cuadrados con restricciones, GraphSLAM se puede resolver empleando métodos estándar de optimización como Gauss-Newton o Levenberg-Marquardt, los cuales se basan en una linealización local iterativa [59], [55].

Una de las ventajas de emplear métodos gráficos para solucionar el SLAM, es que los datos adquiridos desde el estado inicial se usan en la optimización global, lo que no sucede con los métodos de filtrado (EKF y partículas), en los que la información de estados anteriores es resumida en una densidad de probabilidad marginal que resulta de la asunción de modelo de Márkov para el proceso. De esta manera, no es posible corregir los errores introducidos por la linealización en estados previos puesto que la "*memoria*" del proceso se pierde (*propiedad de Márkov*).

Por otro lado, la principal dificultad que se reporta para los métodos GraphSLAM es que la optimización global de toda la información pasada del proceso (*sus datos históricos*), incrementa notablemente el costo computacional y la complejidad del problema. Al respecto se han propuesto algunas soluciones consistentes en limitar uniformemente el número de posiciones con respecto al tiempo o la distancia recorrida o marginalizar aquellas nuevas posiciones que resultan muy cercanas a posiciones anteriores [60], [61].

2.4. Materiales y métodos

Previamente, tanto la formulación probabilística como los métodos de solución del SLAM fueron discutidos (secciones 2.3.3 y 2.3.4). En esta sección se presentarán los materiales, métodos, configuración y diseño de la plataforma robótica con la que se llevarán a cabo los experimentos de validación.

La investigación en robótica y el desarrollo de sistemas robóticos han experimentado un significativo crecimiento en años recientes, debido en parte al surgimiento de nuevos métodos de modelamiento, estimación y control, y a los avances tecnológicos

en computación, sensórica, mecatrónica y comunicaciones. Tal panorama propicia el surgimiento de nuevas aplicaciones de la robótica en diversos ámbitos de la vida cotidiana [62], [63], y también la necesidad de contar con herramientas versátiles y potentes de diseño, simulación y desarrollo de sistemas robóticos funcionales.

En la actualidad los robots son considerados sistemas distribuidos complejos que incluyen una amplia variedad de componentes de hardware (HW) y software (SW), razón por la cual en este trabajo se ha optado por incorporar el mayor grado de modularidad posible en las fases de diseño e implementación del sistema robótico experimental (SRE). Si bien el diseño modular plantea problemas adicionales de integración, comunicación e interoperatividad, estos pueden ser resueltos usando una capa intermedia de software o *middleware* para sistemas robóticos distribuidos [64].

Así mismo, como herramienta de verificación de la eficiencia, seguridad y robustez de nuevos algoritmos, los simuladores juegan un papel clave en la investigación en robótica, principalmente en aplicaciones donde los robots deben interactuar de manera estrecha con las personas, como son los robots de asistencia, educativos y logísticos para entrega de mercancías [65], [66].

En cuanto al hardware, un buen número de compañías alrededor del mundo (Clearpath Robotics, Fetch Robotics, PAL robotics, Hokuyo Automatic, etc.), desarrollan y proveen una amplia gama de plataformas robóticas y sistemas de sensórica, percepción y actuación, para la investigación, aprendizaje y desarrollo de aplicaciones industriales y comerciales compatibles con diferentes middlewares y simuladores.

En este trabajo se usan las siguientes herramientas de software y hardware para llevar a cabo los experimentos de validación:

- **Robot Operating System** (*ROS*): middleware que ha ganado mucha aceptación en entornos académicos e industriales. Se ha posicionado, *de facto*, como un estándar para el desarrollo de aplicaciones robóticas [67], [68], [69].

- **Simuladores**: una herramienta de simulación compatible con el middleware ROS. Las pruebas preliminares de los algoritmos de teleoperación a implementar se llevarán a cabo con Gazebo, un simulador 3D que requiere de un PC con procesamiento gráfico de altas prestaciones, como el que soportan las GPUs (*Graphics Processing Unit*) [70], [71].

- **Turtlebot2**: plataforma de hardware abierto para la investigación, prueba y desarrollo de métodos, algoritmos y aplicaciones en robótica [67], [68], [72]. Cuenta con los sistemas de percepción, control y actuación necesarios para iniciar con el desarrollo de aplicaciones soportadas por ROS. Algunas modificaciones de bajo costo fueron diseñadas para mejorar el procesamiento y autonomía de la plataforma [73], estas se documentan en secciones posteriores.

- **Procesamiento y comunicaciones**: se dispuso de dos equipos de computo, uno de altas prestaciones denominado *Workstation*, el cual es usado para la supervisión del sistema mediante interfaces gráficas de usuario (*GUI*) y para la ejecución de procesos de alto costo computacional. El otro equipo, de menores prestaciones y denominado *Robot PC*, fue incorporado al robot móvil para ejecutar procesos de menor costo computacional, principalmente aquellos asociados a la propiocepción, exterocepción y control de efectores del robot TurtleBot2. La conectividad de ambos equipos se hace a través de un *router*. Las característica del hardware utilizado son:

  - Workstation: computador ASUS de la serie N56J con procesador Intel® Core™ i7-4700HQ CPU @ 2.4GHz × 8, 8 GB de RAM, Disco Duro de 1 TB y GPU NVIDIA® GEFORCE GTX® 760M

  - Robot PC: Mini-ordenador Intel® NUC6i3SYH (`19V DC @ 3.43A`) con procesador Intel® Core™ i3-6100U CPU @ 2.3GHz × 4, 4 GB de RAM, Disco Duro de 500 GB

  - Enrutador: TP-Link 3G/4G TL-MR3420

  - Cámara USB: Logitech® C270 con captura de vídeo de hasta 1280 x 720 píxeles @ 30fps

  Sistema operativo y versión ROS: en ambos equipos se instaló la distribución Linux Ubuntu 16.04.4 LTS (Xenial) de 64 bits y ROS Kinetic Kame.

2.4.1. Robot Operating System (ROS)

ROS es un middleware de código abierto para el desarrollo de aplicaciones robóticas que proporciona un entorno de programación distribuido, tanto para plataformas reales como simuladas y que aporta abstracción de hardware, control de bajo nivel, transferencia de mensajes entre procesos y gestión de paquetes de software [74]. ROS fue concebido teniendo en cuenta los siguientes criterios [75]:

- **a) Peer-to-Peer**: procesos conectados en tiempo de ejecución sin clientes ni servidores fijos. Los procesos se comportan a la vez como clientes y servidores.
- **b) Procesamiento distribuido**: los procesos (*nodos*) pueden correr en *hosts* diferentes y comunicarse a través de la red.
- **c) Sistema liviano**: ROS provee un mecanismo de empaquetamiento liviano para bibliotecas de terceras fuentes (*third-party libraries*).
- **d) Multi-lenguaje**: si bien ROS soporta diferentes lenguajes como Octave, LISP, Lua y Java, los APIs (*Application Programming Interface*) con mejor soporte son C++ (*roscpp*) y python (*rospy*).
- **e) Libre y de código abierto**: la totalidad del código fuente de ROS es de dominio público.
- **f) Basado en herramientas**: el diseño de ROS se basa en un *micro-kernel* en el que se usa un gran número de pequeñas herramientas para construir y ejecutar diversos componentes de ROS, en oposición a un entorno monolítico de desarrollo y ejecución.

La arquitectura de ROS fue diseñada y organizada en tres componentes denominados *niveles de conceptos* [1], que se muestran en la figura 2.5 y se explican a continuación.

[1] http://wiki.ros.org/ROS/Concepts

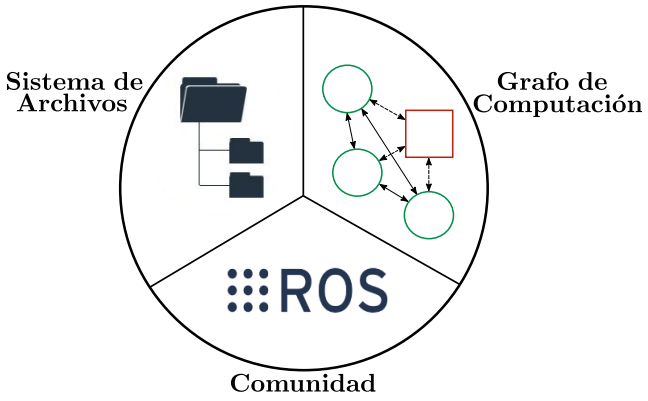


Figura 2.5 La arquitectura del *middleware* ROS incorpora tres niveles de conceptos: sistema de archivos, grafo de computación y comunidad.

Sistema de archivos

Este nivel permite centralizar el proceso de construcción de proyectos, al tiempo que brinda un alto grado de flexibilidad mediante herramientas para descentralizar sus dependencias. El sistema de archivos de ROS incorpora diferentes recursos organizados en árboles jerárquicos de archivos en disco (*directorios*), tal como se aprecia en la figura 2.6.

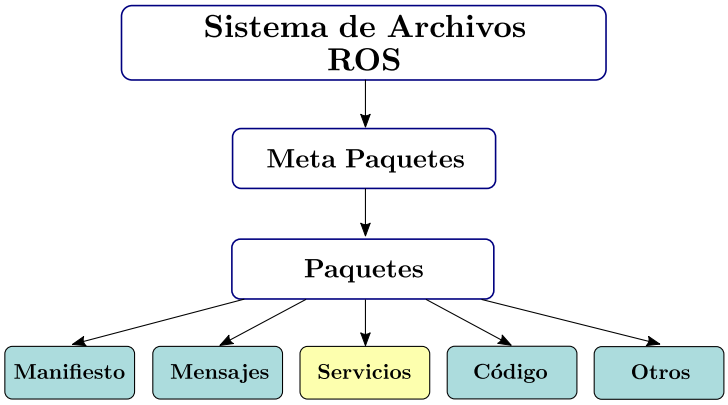


Figura 2.6 Nivel de sistema de archivos ROS. Organiza los recursos en árboles de directorios.

- **Meta paquetes**: agrupan una colección de paquetes de propósito especial. También se les suele denominar pilas (*stacks*).

- **Paquetes**: unidad principal para la organización del software en ROS. Usualmente contienen ejecutables (*nodos*), librerías, archivos de configuración y demás objetos que se organizan conjuntamente como una sola unidad.
- **Manifiesto**: el manifiesto de un paquete corresponde a un archivo `package.xml` que proporciona información relativa al paquete, autores, licencia, dependencias, banderas de compilación y demás.
- **Mensajes**: estructuras de datos que se envían de un proceso a otro.
- **Servicios**: define las estructuras de solicitud de datos y respuesta de los servicios proporcionados por cada proceso ROS.
- **Código**: archivos de código fuente y scripts en los qe se codifican los procesos ROS.

Grafo de computación

La computación en ROS se lleva a cabo mediante una red de procesos interconectados (*nodos*) que procesan y proveen datos de diferentes maneras. Los conceptos básicos del nivel de grafo de computación se observan en la figura 2.7.

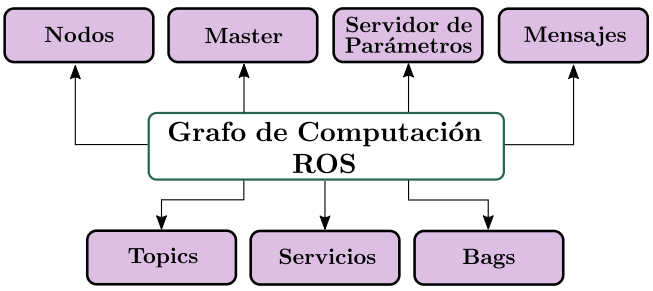


Figura 2.7 Conceptos del grafo de computación ROS.

- **Nodos**: son los procesos que llevan a cabo la computación en ROS. Se encuentran interconectados en red (*peer-to-peer*) mediante un modelo publicador/suscriptor. ROS promueve la "*modularidad de grano fino*", en la cual un sistema esta compuesto de un buen número de nodos simples.
- **Máster**: provee registro de nombres y búsqueda. Sin el máster los nodos de la red no podrían encontrase para intercambiar mensajes o invocar servicios.

Para sistemas robóticos distribuidos, el máster se deberá ejecutar solo en un computador, los nodos restantes se pueden ejecutar en otras máquinas y deberán poderse comunicar con el máster.

- **Servidor de parámetros**: Los nodos pueden almacenar variables en el servidor de parámetros y establecer también su nivel de privacidad. Si el parámetro (*variable*) tiene un alcance global, entonces puede ser accesado por los demás nodos de la red. Actualmente el servidor de parámetros hace parte del máster.

- **Mensajes**: los nodos se comunican entre sí usando mensajes. Los mensajes son estructuras de datos con campos de diferentes tipos que pueden ser accesados por los nodos.

- **Topics**: ROS gestiona los mensajes a través de nombres. Cuando un nodo esta enviando datos (*publicador*), se dice que esta publicando un topic. Un nodo suscrito a este topic (*suscriptor*) recibe el mensaje y puede acceder los datos contenidos en él. Los nombres asignados a los topics deben ser únicos para evitar conflictos en la red. En general, los nodos no saben de la existencia de otros nodos, puesto que la comunicación se da a través de un mecanismo de publicación/suscripción. Un esquema tal permite desacoplar la producción y consumo de información en la red, mejorando la modularidad y robustez del sistema.

- **Servicios**: el modelo publicador/suscriptor es un paradigma de comunicación muy flexible, sin embargo cuando un nodo publica un mensaje, éste podría ir dirigido a muchos nodos, lo que resulta no ser tan eficiente cuando se quiere hacer una petición a un nodo en particular. ROS provee un mecanismo más eficiente para los requerimientos/replicas entre nodos y es a través de los servicios.

- **Bags**: son archivos con extensión *.bag* usados para salvar y reproducir mensajes de datos ROS. Los *bags* constituyen un importante mecanismo para almacenar datos de sensores y órdenes enviadas a los actuadores de un sistema robótico, facilitando así el desarrollo y prueba de algoritmos.

Un ejemplo de grafo de computación ROS se presenta en la figura 2.8. Los nodos (*procesos en ejecución*) se encuentran encerrados por óvalos y los topics por rectángulos. El nodo `/usb_cam` publica mensajes en los topics `/usb_cam/image_raw`

y `/usb_cam/camera_info`. El nodo `/image_viewer` se encuentra suscrito al topic `/usb_cam/image_rect`. El nodo `/usb_cam/image_proc` es tanto publicador como suscriptor.

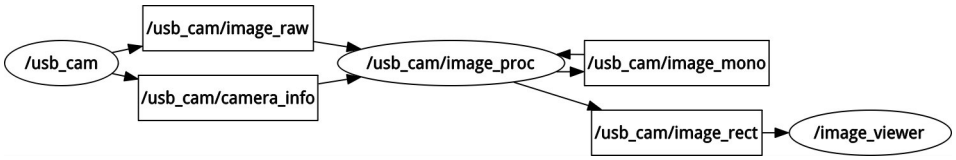


Figura 2.8 Ejemplo de grafo de computación de una red ROS.

Comunidad ROS

El concepto de comunidad ROS corresponde a los recursos con que cuenta la comunidad de desarrolladores y usuarios de ROS para intercambiar software y conocimiento. Estos recursos son:

- **Distribuciones ROS**: colección de meta paquetes organizados en versiones de fácil instalación.
- **Repositorios**: ROS depende de una red federada de repositorios de código en la que diferentes instituciones pueden desarrollar y liberar sus propios componentes de software.
- **ROS wiki**: foro principal para la documentación de información relativa a ROS. Cualquier persona se puede registrar y obtener una cuenta para contribuir con documentación, proporcionar correcciones o actualizaciones y escribir tutoriales.
- **Sistema de gestión de errores**: pueden hacer uso de este recurso, quienes han detectado errores de software o necesitan agregar nuevas funcionalidades a éste.
- **Listas de correo**: es el principal canal de comunicación para la actualización de ROS. Funge también como foro de preguntas sobre ROS.
- **ROS answers**: recurso web que ayuda a la formulación de preguntas relacionadas con ROS. Una vez se publica una duda en este sitio, otros usuarios de ROS la pueden ver y sugerir soluciones.

- **Blog**: el blog de ROS se mantiene actualizado con noticias, fotos y videos relacionados con la comunidad ROS [2].

2.4.2. Plataforma robótica experimental - RobSLAM

El prototipo de robot móvil experimental, de aquí en adelante denominado *RobSLAM*, se desarrolló a partir del robot personal Turtlebot2, una plataforma de código abierto soportada por ROS (figura 2.9). El robot incorpora un sensor exteroceptivo RGB-D (*Kinect V1*), bandejas y soportes que permiten anexar nuevo equipamiento y, como componente principal, la base móvil *iClebo Kobuki*. Esta integra sistemas de sensórica propioceptiva, procesamiento y locomoción.

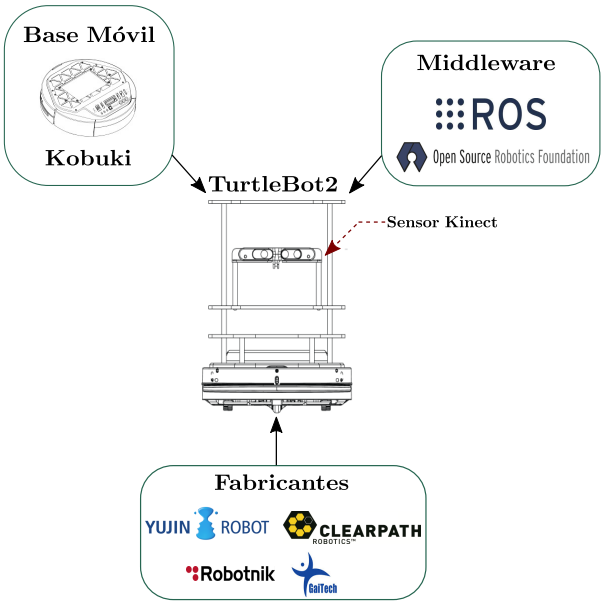


Figura 2.9 TurtleBot2: plataforma base del sistema experimental **RobSLAM**.

De la plataforma original se removió la bandeja intermedia para reducir la altura del robot y mantener bajo su centro de gravedad. Quedaron disponibles cuatro separadores de aluminio de `13 mm` de diámetro por `51 mm` de alto, cuya longitud fue extendida mediante cilindros perforados de acrílico de `20 mm` de altura. Estos separadores se usaron para anclar el mini-ordenador NUC (Robot PC) a la parte inferior de la bandeja superior, para ello se dispuso de una base rectangular de

[2] http://www.ros.org/news

soporte de 130 × 150 mm con sus tornillos de fijación. En la figura 2.10 se presenta la arquitectura hardware del sistema RobSLAM.

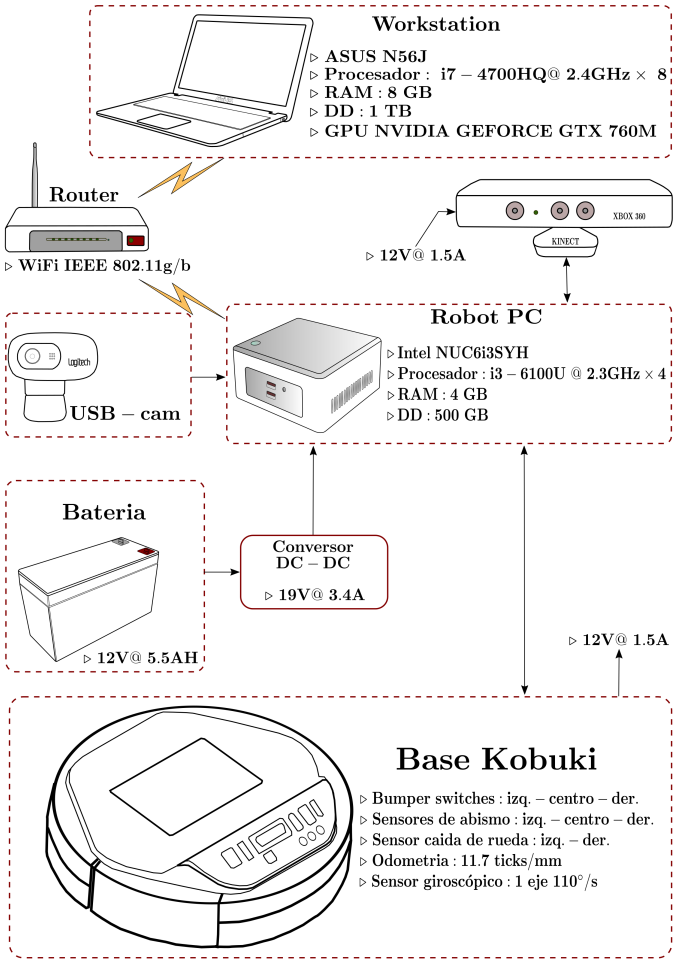
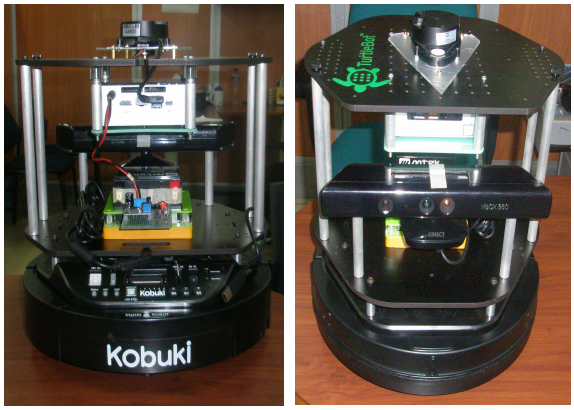


Figura 2.10 Arquitectura hardware del sistema robótico RobSLAM.

Puesto que la tensión de 19.1V DC @ 2A, disponible en uno de los conectores de la base Kobuki resultó insuficiente para la alimentar el mini-ordenador NUC, se diseñó un sistema de alimentación regulado que incluye una batería de 12V @ 5.5AH, un conversor DC/DC para elevar el voltaje a 19.0V DC y una estructura de fijación para impedir el desplazamiento de la batería. El sistema se ancló en la bandeja inferior del robot permitiendo mantener bajo su centro de gravedad.

Tal diseño provee una alimentación adecuada para el Robot PC y soporta la operación continua, hasta por tres horas y media en condiciones de plena carga de la

batería, del sistema, tiempo suficiente para adelantar los experimentos propuestos y recabar datos para el análisis. En la figura 2.11 se puede observar la apariencia definitiva del robot móvil.



a) *b)*

Figura 2.11 Componente "*robot móvil*" del sistema RobSLAM. **a**) Vista posterior donde se aprecia el sistema de alimentación regulado, y **b**) vista anterior, donde se observa el Robot PC y el sensor Kinect.

2.4.3. Respuesta a preguntas de investigación

A continuación se da respuesta a algunas de las preguntas de investigación formuladas en la sección 1.5 de este trabajo:

- ¿Qué arquitectura de hardware y software se puede implementar para procesar la información proporcionada por una cámara a una rata de 30 *fps*?

  Respuesta: Una alternativa eficiente y de bajo costo para resolver el SLAM y la navegación de robots móviles en entornos interiores, consiste en emplear una base móvil con tracción diferencial que integre tanto mecanismos para el control de efectores como para el acceso a información propioceptiva.

  En general las tareas de SLAM y navegación requieren ser implementadas en casi todo robot de servicios con capacidad de carga reducida, sin que ello implique un incremento significativo de los costos de diseño e implementación.

En este trabajo se optó por utilizar la plataforma robótica `TurtleBot2`, un robot de bajo costo para interiores cuya principal componente es una plataforma robótica diferencial con capacidad de carga de hasta 5 *kg*, suficiente para soportar los componentes tecnológicos adicionales que fueron agregados al sistema.

Se emplearon los soportes y bandejas apilables provistas por el fabricante para incorporar hardware de procesamiento y percepción adicionales, de tal manera que se puedan correr sin mayores dificultades los más recientes algoritmos SLAM desarrollados por la comunidad científica.

Una característica del sistema RobSLAM que cabe resaltar, es su total integración con ROS, uno de los *middleware* de mayor crecimiento y penetración en la investigación y desarrollo de la robótica en la actualidad.

ROS es compatible con librerías representativas de visión por computador (**OpenCV**), procesamiento de datos tridimensionales e imágenes de profundidad (**PCL**), planificación de movimientos (**MoveIt!**), simuladores 2D y 3D (**Stage**, **Gazebo**) y librerías cliente para el desarrollo de nodos en C++ (**roscpp**) y python (**rospy**).

La arquitectura de software es soportada por el *middleware* ROS, que dispone de paquetes de mapeo y navegación (`gmapping`), localización (`AMCL`), drivers para cámaras y sensores RGB-D (`usb_cam`, `openni`), paquetes de conversión de imágenes de profundidad a medidas láser (`depthimage_to_laserscan`), de conversión de imágenes OpenCV a imágenes ROS y viceversa (`cv_bridge`), API para el desarrollo de nodos en python, tanto para la operación y monitoreo del robot, como para la evaluación de los algoritmos SLAM considerados.

Las librerías OpenCV, PCL y algoritmos SLAM 3D compatibles con ROS (p. ej. `ORB_SLAM`, `RTAB-Map`) permiten procesar secuencias de imágenes en tiempo real provistas por una cámara USB a 30 *fps* y elaborar mapas 3D de los entornos interiores explorados.

- ¿Cuales deberán ser las características de la plataforma móvil mediante la cual se pretende explorar un entorno de trabajo?

Respuesta: Las principales características o requerimientos considerados en el proceso de diseño del sistema RobSLAM fueron los siguientes:

Maniobrabilidad: una característica considerada al inicio del proceso de diseño del sistema robótico fue su maniobrabilidad en recintos interiores, caracterizados por pasillos y accesos estrechos, espacios de pequeñas y medianas dimensiones con alta circulación de personas. Una configuración simple y de bajo costo, capaz de cumplir con estos requisitos es la tracción diferencial, incorporada en la base móvil Kobuki seleccionada para la implementación del sistema.

Estabilidad: para robots orientados al trabajo en interiores, una de las configuraciones que mayor estabilidad brinda es la diferencial, con una o dos ruedas de apoyo. En el caso particular del sistema RobSLAM, para el cual se espera que cuente a futuro con capacidades de manipulación de objetos al incorporarle un brazo robótico, la estabilidad la proporcionan dos ruedas de soporte localizadas en las partes anterior y posterior de la base móvil.

Controlabilidad: uno de los primeros requerimientos establecidos para el sistema RobSLAM, es que no se tuviese que resolver el problema del control de movimientos del robot en el bajo nivel. En la base Kobuki la tarjeta de control **OpenCR** implementa los controladores PID para los motores DC de las ruedas de tracción. A la base solo se envían los comandos (consignas) de velocidad lineal y angular que se requieran.

Modularidad: RobSLAM fue diseñado teniendo en cuenta criterios de modularidad, a nivel de hardware y de software. Ambos criterios se cumplieron al seleccionar una base móvil de hardware libre y abierto y un *middleware* de código abierto que provee un alto grado de abstracción de hardware.

Accesibilidad: el sistema RobSLAM debería poder suministrar una puerta de acceso rápido a personal no experto, no solo a nuevas tecnologías, sino a métodos y algoritmos propios de la robótica. Este requerimiento fue cumplido gracias a la adopción del middleware ROS que provee alto grado de integración

con herramientas de software y hardware de última generación.

Otras características: también fueron tenidas en cuenta características adicionales como el costo y las dimensiones del sistema. TurtleBot 2 es una plataforma robótica básica de bajo costo (algo más de USD $1000) y dimensiones reducidas,[3] que lo hacen idóneo para trabajar en entornos interiores pequeños y medianos, en aplicaciones que no demanden elevadas capacidades de carga como son la robótica de servicios y de asistencia personal.

2.5. Conclusiones del capítulo

En este capítulo se presentó un estudio sistemático de la formulación probabilística del problema del SLAM y su solución recursiva, hasta llegar a su formulación como una Red Dinámica Bayesiana (*RDB*) de estimación probabilística. Se discutieron los paradigmas mas representativos para la solución del SLAM, iniciando con la aproximación clásica del filtro de Kalman Extendido, hasta los más recientes métodos basados en Filtros de Partículas y Grafos Probabilísticos para el modelamiento de las variables aleatorias y sus dependencias condicionales.

La forma en que se presentó la formulación y los métodos de solución del SLAM, contribuye a una apropiación conceptual y procedimental rápida, rigurosa y concisa por parte del lector interesado, de tal forma que le permite concentrar sus esfuerzos en la selección y uso de aquellas herramientas tecnológicas más convenientes para el desarrollo de sus propias aplicaciones robóticas.

En la sección de Materiales y Métodos se expusieron rigurosamente las herramientas de hardware y software que se emplearan en los experimentos de validación adelantados en capítulos posteriores. Cabe destacar la presentación detallada de las características del software empleado en el desarrollo de las aplicaciones robóticas, específicamente el middleware ROS y su arquitectura de niveles conceptuales.

Se discutió en detalle la arquitectura hardware del sistema propuesto, la cual integra diferentes subsistemas que permiten incrementar el nivel de autonomía energética y de procesamiento del sistema robótico experimental diseñado en este trabajo (RobSLAM).

[3] https://www.clearpathrobotics.com/turtlebot-2-open-source-robot/

En tal sentido, en este capítulo se contribuye a que el lector disponga de un referente que le permita abordar, desde una perspectiva holística, el problema de diseño e implementación de sistemas robóticos modernos, con aplicaciones que podrían ir desde la asistencia y cuidado personal, los servicios de mensajería y entrega de suministros, hasta el apoyo en actividades de búsqueda, exploración y rescate.

Capítulo 3

Experimentos SLAM

En este capítulo se conducirán experimentos de mapeo, localización y navegación en entornos interiores que permitan validar algunos los métodos expuestos en la sección 2.3.4 (*Algoritmos de solución del SLAM*). Para tal fin se empleará el sistema RobSLAM y se desarrollarán algoritmos de teleoperación para la exploración del entorno que se pretende mapear.

3.1. Mapeo con teleoperación

3.1.1. Sistema RobSLAM

En la sección 2.4.2 (*Plataforma robótica experimental - RobSLAM*) se describió la arquitectura hardware de RobSLAM, un sistema diseñado para la investigación en mapeo, localización y navegación autónoma de vehículos, así como para el desarrollo de aplicaciones en robótica de servicios y asistencial. El sistema incorpora una base móvil Kobuki [1] con dos ruedas de tracción en configuración diferencial y dos ruedas de apoyo o *caster balls* (ver figuras 2.10 y 2.11). Por su tipo de tracción, a estos robots se les conoce también como 2WMR (*two wheeled mobile robots*).

Las especificaciones más significativas de la base Kobuki se presentan en el cuadro 3.1.

[1] http://en.yujinrobot.com/rd-platform

Cuadro 3.1 Especificaciones de la base Kobuki (Yujin Robot, Co., Ltd.).

Mecánica	
Velocidad traslacional máxima	70 cm/s
Velocidad rotacional máxima	$180^\circ/s$
Capacidad de carga útil	5 kg
Altura máx. de obstáculo que puede escalar	12 mm
Profundidad máx. de bache que puede superar	5 cm
Tiempo esperado de operación	3 $horas$
Tiempo esperado de carga de batería	1,5 $horas$
Diámetro de las ruedas de tracción	7,6 cm
Distancia entre puntos de apoyo de las ruedas motrices	23 cm
Propiocepción	
Odometría lineal (encoders en ruedas de tracción)	11,7 $ticks/mm$
Odometría angular (sensor giroscópico de un eje)	$110^\circ/s$
Sensores de choque	$izq.$, $centro$, $der.$
Sensores de abismo	$izq.$, $centro$, $der.$
Sensores de caída de rueda	$izq.$, $der.$

En cuanto al sistema de percepción exteroceptivo, RobSLAM cuenta con una cámara USB, un sensor RGB-D (*Kinect*) y un escáner láser de bajo costo. Las especificaciones de estos dispositivos se resumen en el cuadro 3.2.

3.1.2. Modelo cinemático

El modelado cinemático se refiere al estudio del movimiento de un sistema mecánico sin considerar las fuerzas y torques involucrados. Para un 2WMR, el modelo cinemático permite expresar sus velocidades lineal y angular como funciones de las velocidades de sus ruedas de tracción y su geometría [76], [77], [78].

Una trayectoria curvilínea sin deslizamiento, se obtiene cuando el robot gira alrededor de un punto externo ubicado sobre el eje común a sus ruedas de tracción. Este punto, conocido como *Centro Instantáneo de Curvatura* (*CIC*), se irá desplazando a medida que las velocidades de las ruedas motrices van cambiando. De esta manera es posible establecer diferentes trayectorias o recorridos para el robot.

Cuadro 3.2 Especificaciones de los sensores exteroceptivos del sistema RobSLAM.

Cámara	
Resolución	1280×720
Conectividad	USB 2.0
Sensor Kinect	
Resolución cámara RGB	640×480 @30 fps
Resolución cámara de profundidad	320×240
Campo de visión horizontal	57°
Campo de visión vertical	43°
Profundidad máx. (*distancia*)	$\simeq 4{,}5\ m$
Profundidad mín. (*distancia*)	$0{,}4\ m$
Escáner Láser *LDS-01*	
Fuente de luz	diodo láser, $\lambda = 785\ nm$
Rango	$120 \sim 3500\ mm$
Exactitud en la medida ($120mm \sim 499mm$)	$\pm 15\ mm$
Exactitud en la medida ($500mm \sim 3500mm$)	$\pm 5{,}0\,\%$
Rata de escaneo	$300 \pm 10\ rpm$
Campo de visión horizontal	360°
Resolución angular	1°

Como se observa en la figura 3.1, en cada instante de tiempo las ruedas derecha e izquierda describen curvas alrededor del CIC a la misma velocidad angular $\omega = d\psi/dt$. El cuadro 3.3 muestra los símbolos usados. Las velocidades lineales de las ruedas de tracción vienen dadas por:

$$v_L = \omega(R + \frac{L}{2}) \tag{3.1}$$

$$v_R = \omega(R - \frac{L}{2}), \tag{3.2}$$

donde R es la distancia entre el CIC y el punto medio C_R entre ambas ruedas.

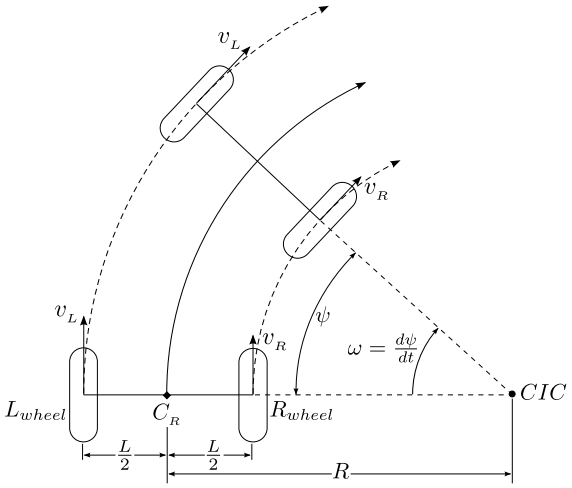


Figura 3.1 Cinemática de un robot móvil diferencial.

Cuadro 3.3 Parámetros y variables del modelo cinemático de un robot diferencial.

Parámetro/Variable	Descripción
L	Dist. entre puntos de apoyo de las ruedas, 0,23 m
R	Radio instantáneo de curvatura $[m]$
C_R	Punto medio entre las ruedas motrices 0,115 m
ψ	Ángulo de orientación del robot $[rad]$
ω	Vel. angular del robot $[rad \cdot s^{-1}]$
v_L	Vel. lineal rueda izquierda $[m \cdot s^{-1}]$
v_R	Vel. lineal rueda derecha $[m \cdot s^{-1}]$
v_{CR}	Vel. lineal del C_R $[m \cdot s^{-1}]$

La velocidad angular del robot se obtiene al restar de la ecuación 3.1 la 3.2:

$$\omega = \frac{v_L - v_R}{L} \tag{3.3}$$

Si estas ecuaciones se suman y se reemplaza 3.3 en el resultado, se obtiene:

$$R = \frac{v_L + v_R}{v_L - v_R} \cdot \frac{L}{2} \tag{3.4}$$

Las ecuaciones 3.3 y 3.4 permiten calcular la velocidad angular del robot y el radio instantáneo de curvatura R como funciones de las velocidades lineales de las ruedas y su separación L. Estas velocidades pueden ser fácilmente establecidas mediante la odometría lineal del robot.

Finalmente, la velocidad lineal del punto medio C_R corresponde al promedio de las velocidades de las ruedas:

$$v_{CR} = \omega R = \frac{v_L + v_R}{2} \tag{3.5}$$

Las expresiones 3.3 a 3.5, dan cuenta de los desplazamientos básicos que pueden realizar los robots diferenciales, los cuales se resumen en el cuadro 3.4 y que permiten sintetizar trayectorias más complejas.

Cuadro 3.4 Movimientos básicos de los 2WMR.

Condición	Velocidades/Radio	Desplazamiento
$V_L = V_R = v$	$w = 0,\ v_{CR} = v,\ R = \infty$	Adelante
$V_L = V_R = -v$	$w = 0,\ v_{CR} = -v,\ R = -\infty$	Atrás
$V_L = -V_R$	$w = -\frac{2V_L}{L},\ v_{CR} = 0,\ R = 0$	Contra horario
$V_R = -V_L$	$w = \frac{2V_L}{L},\ v_{CR} = 0,\ R = 0$	Horario
$V_L = v,\ V_R = 0$	$w = \frac{v}{L},\ v_{CR} = \frac{v}{2},\ R = \frac{L}{2}$	Der. R mín.
$V_L = 0,\ V_R = v$	$w = \frac{-v}{L},\ v_{CR} = \frac{v}{2},\ R = \frac{-L}{2}$	Izq. R mín.
$V_L > V_R$	$w \neq 0,\ v_{CR} \neq 0,\ R > \frac{L}{2}$	Giro a derecha
$V_L < V_R$	$w \neq 0,\ v_{CR} \neq 0,\ R < \frac{-L}{2}$	Giro a izquierda

3.1.3. Componentes de software

A continuación se resumen los módulos de ROS empleados para implementar la aplicación de mapeo 2D de un entorno interior de oficinas.

- **turtlebot_bringup**: lanza los drivers que permiten acceder a la información sensorial y comandar los actuadores del robot móvil.
- **gmapping**[2]: implementa de manera eficiente un filtro de partículas *Rao-Blackwellized* para resolver el SLAM. La disponibilidad de un *wrapper* ROS para gmapping ha popularizado su uso[3], permitiendo la construcción de mapas de celdas de ocupación a partir de las medidas proporcionadas por un láser y la odometría del robot [79], [80].
- **rtab_map**[4] (*Real-Time Appearance-Based Mapping*): método graph-slam RGB-D que combina un algoritmo Bayesiano de detección de bucles cerrados y una optimizador de grafo (ver figura 2.4) para resolver el SLAM, a la vez que implementa una gestión eficiente de la memoria que le permite cumplir requerimientos de tiempo real [81].
- **turtlebot_rviz_launchers**: lanzadores de `rviz`[5] para la visualización del robot y la salida de sus sensores.
- **rosserial_python**: implementa el protocolo *rosserial* para la comunicación con dispositivos tales como sistemas embebidos a través del puerto serial del computador. Para este experimento se emplea en la captación de datos del dispositivo de teleoperación diseñado (hardware que se describe más adelante).
- **teleop_joy.py**: script que lanza el nodo de teleoperación diseñado para conducir el robot con un dispositivo tipo *joystick*.

3.1.4. Configuración de red del sistema RobSLAM

Como se observa en la figura 3.2, se emplea un router para configurar una red de área local que soporta la interconectividad de procesos ROS que, si bien

[2] https://openslam-org.github.io/
[3] http://wiki.ros.org/gmapping
[4] https://github.com/introlab/rtabmap
[5] herramienta de visualización 3D para ROS

pueden correr en equipos de computo diferentes, hacen parte de un único grafo de computación. Las direcciones IP asignadas a cada máquina se pueden establecer mediante el uso del comando de línea `ifconfig` y el programa de monitoreo de redes `nmap`[6]. En la figura 3.2 se aprecian las direcciones IP asignadas a los equipos Workstation y Robot PC.

En una red como la que implementa el sistema RobSLAM, el nodo maestro ROS deberá correr en el computador que controla la base móvil Kobuki, en este caso el Robot PC. También se recomienda ejecutar en esta máquina aquellos nodos que procesan información sensorial y generan acciones de control directa sobre los efectores. Por otro lado, en el computador remoto (Workstation) se sugiere ejecutar procesos de monitoreo y visualización, como aquellos en los que suelen intervenir operadores humanos.

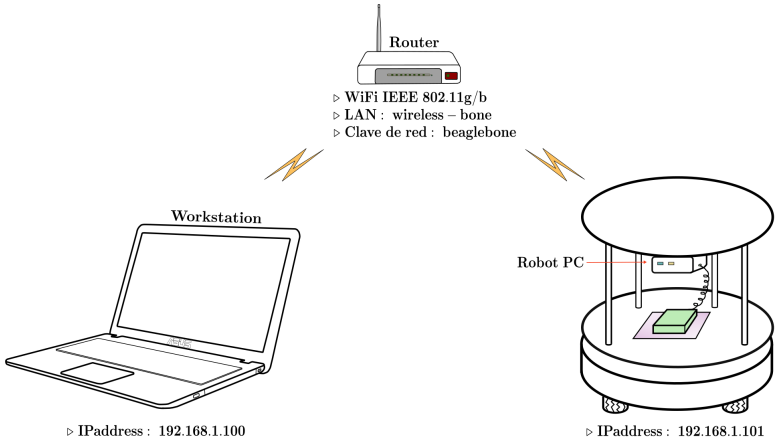


Figura 3.2 Red ROS correspondiente al sistema RobSLAM. Los procesos pueden ser ejecutados en máquinas diferentes (computación distribuida) y acceder a los mensajes a través de una red de área local.

Para configurar la red del sistema RobSLAM, se debe editar el archivo `.bashrc` de configuración de la terminal de comandos de ambos computadores, indicando en el archivo del Robot PC, que es en esta máquina donde se deberá correr el ROS Máster. Las siguientes líneas se agregan al final de estos archivos:

- Archivo `.bashrc` del Workstation:

```
# dirección IP del computador donde correrá el ROS Master
export ROS_MASTER_URI=http://192.168.1.101:11311
```

[6]https://nmap.org/

```
# dirección IP del Host ASUS
export ROS_HOSTNAME=192.168.1.100
```

- Archivo `.bashrc` del Robot PC:

```
# Robot PC como ROS Master
export ROS_MASTER_URI=http://localhost:11311
# dirección IP del Host NUC
export ROS_HOSTNAME=192.168.1.101
```

Una vez efectuados los cambios, toda terminal de comandos deberá cerrarse y abrir otras para que los cambios sean incorporados en estas nuevas terminales.

3.1.5. Módulo de teleoperación

La ejecución del algoritmo SLAM requiere de una exploración sistemática del entorno. Para tal fin, se diseñó un dispositivo de teleoperación que recoge la información procedente de un control analógico de bajo costo tipo joystick, y la transfiere vía puerto serial al Workstation del sistema RobSLAM (figura 2.10). Un script de python fue desarrollado para procesar esta información y generar comandos suaves de velocidad para el robot móvil.

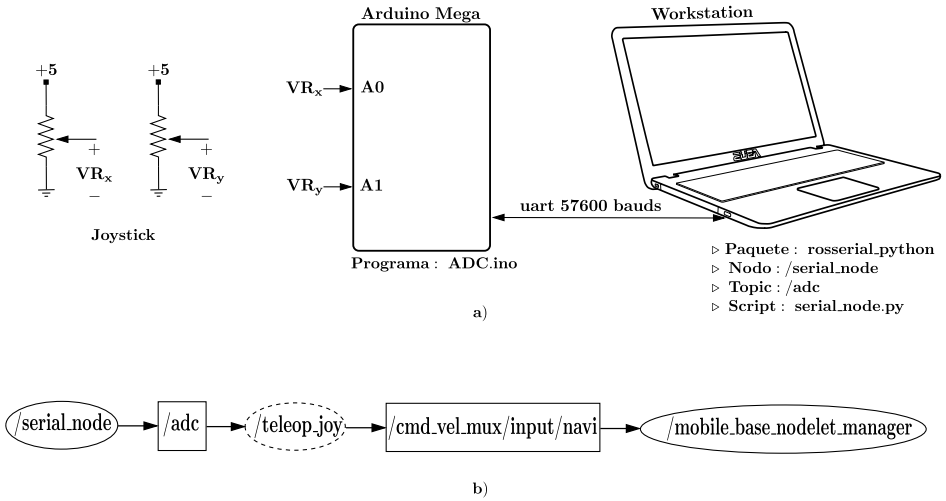


Figura 3.3 Hardware y software de teleoperación: a) el movimiento del joystick genera los voltajes variables VR_x y VR_y que son codificados y transferidos por el Arduino al Workstation. En éste se corre el nodo de teleoperación b) con el que se generan comandos de velocidad para el robot móvil.

El dispositivo de teleoperación y su conexión al Workstation se aprecia en la figura 3.3 a). Las señales analógicas VR_x y VR_y, procedentes del joystick, se conectan a las entradas ADC (*analog-to-digital conversion*) $A0$ y $A1$ del sistema embebido Arduino Mega. Los códigos digitales obtenidos son transferidos al Workstation, aprovechando así la capacidad de abstracción de hardware que brinda el middleware ROS.

En el sistema embebido se graba el programa `ADC.ino`, disponible como ejemplo en la librería `ros_lib` del Arduino IDE. El programa ha sido levemente modificado para monitorear solo dos entradas analógicas de las seis que monitorea originalmente.

En el Workstation se ejecuta el nodo `serial_node`, que hace parte del paquete `rosserial_python` y que recibe los códigos digitales enviados por el Arduino. El nodo publica mensajes tipo `rosserial_arduino/Adc` en el topic `adc`.

En la figura 3.3 b) se aprecia el grafo de computación correspondiente a la teleoperación del robot. El nodo `/teleop_joy` se suscribe al topic `/adc`, procesa los mensajes allí publicados y genera mensajes de velocidad `geometry_msgs/Twist` que publica en el topic `cmd_vel_mux/input/navi` para dirigir el desplazamiento del robot.

Como es de suponer, la rata de publicación de los mensajes de velocidad enviados al robot, viene dada por la rata de publicación de mensajes en el topic `/adc`, que es de aproximadamente $f = 291\ Hz$.

En la figura 3.4 se observan los códigos que se obtienen al desplazar el joystick de extremo a extremo a lo largo de los ejes X y Y. En la posición de reposo (centro) se presenta un pequeño desfase con respecto al código teórico que debería obtenerse (`joy_code_x = 512`, `joy_code_y = 512`), que se debe principalmente a pequeñas alinealidades asociadas a los potenciómetros y las tolerancias asumidas en el proceso de fabricación del dispositivo.

El nodo de teleoperación emplea los códigos digitales para generar los perfiles de velocidad lineal y angular (figura 3.4) caracterizados por las funciones de transferencia dadas por las ecuaciones 3.6 y 3.7 respectivamente.

$$v = \begin{cases} \frac{0{,}4C_x - 168}{420}, & \text{si } 0 \leq C_x < 420\ (retroceso) \\ 0, & \text{si } 420 \leq C_x < 700\ (paro) \\ \frac{0{,}4C_x - 280}{323}, & \text{si } 700 \leq C_x \leq 1023\ (avance) \end{cases} \tag{3.6}$$

$$\omega = \begin{cases} \frac{168-0{,}4C_y}{420}, & \text{si } 0 \leq C_y < 420 \ (giro\ antihorario) \\ 0, & \text{si } 420 \leq C_y < 700 \ (paro) \\ \frac{280-0{,}4C_y}{323}, & \text{si } 700 \leq C_y \leq 1023 \ (giro\ horario), \end{cases} \tag{3.7}$$

en donde C_x y C_y son los códigos que resultan del desplazamiento del joystick en los ejes X y Y respectivamente.

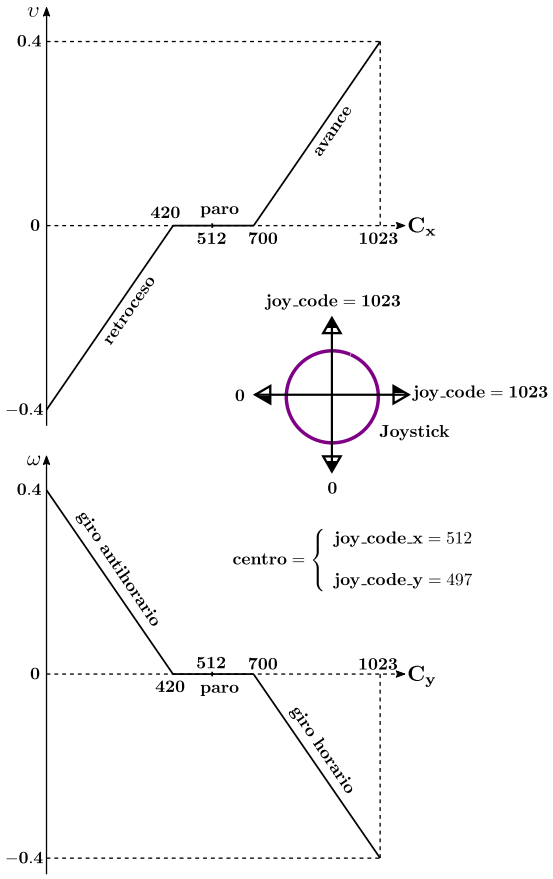


Figura 3.4 Códigos digitales que resultan del accionamiento del joystick y perfiles de velocidad lineal v y angular ω que implementa el nodo de teleoperación.

Los perfiles propuestos permitieron obtener rampas suaves de aceleración durante la teleoperación del robot, tal y como se puede apreciar en la figura 3.5. La gráficas se obtuvieron después de realizar pruebas de teleoperación por diez segundos en un entorno interior de oficinas. Se aprecia como la velocidad en el avance se va incrementando hasta alcanzar el valor máximo de 0,4 cm/s, la cual se mantiene mientras el joystick se encuentra en su posición tope hacia adelante. Posteriormente,

la velocidad va disminuyendo linealmente hasta la detención del robot, cuando se lleva el joystick a su posición de equilibrio (centro).

Algo similar sucede durante el retroceso del robot, en el que se alcanza la velocidad límite de $-0{,}4\ cm/s$ cuando se lleva el joystick a su posición tope hacia atrás. La velocidad va disminuyendo a medida que se acciona el joystick hacia su posición de equilibrio, hasta que el robot se detiene.

Cabe destacar que, de acuerdo a los perfiles de velocidad diseñados (ecuaciones 3.6 y 3.7), se cuenta con un rango de posiciones del joystick cercanas a su posición de equilibrio para las cuales no se generan comandos de velocidad. De esta forma se evitan accionamientos involuntarios del robot cuando se presentan manipulaciones no intencionadas del joystick.

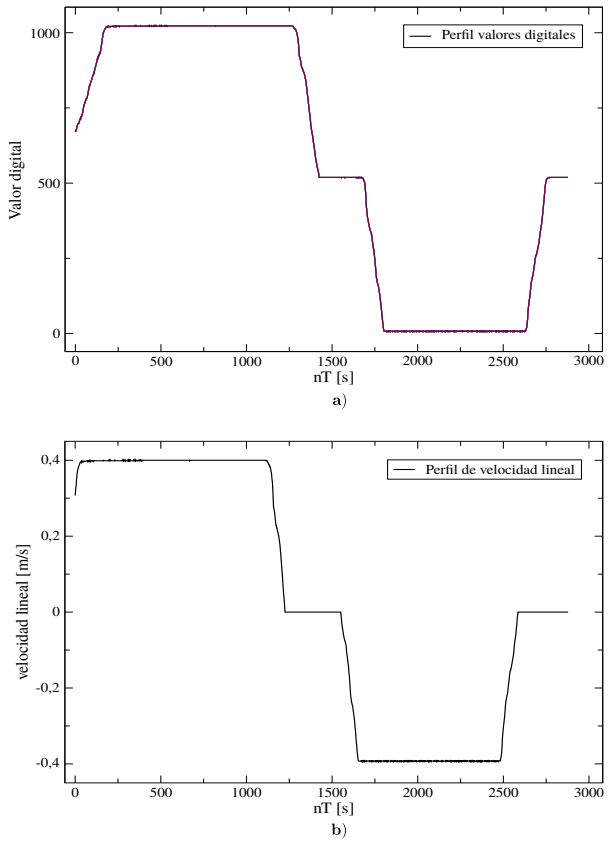


Figura 3.5 Resultados de una prueba de teleoperación del robot. La duración de la prueba fue de aproximadamente 10 segundos, con avance, detención y retroceso.

3.1.6. Experimentos SLAM con gmapping

El sistema RobSLAM se probó en un entorno interior de oficinas cuyo plano se observa en la figura 3.6. El espacio consta de 14 módulos de $2{,}44m \times 1{,}75m$ localizados a lado y lado de un corredor central en el que se localizan dos vigas. Cada cubículo se denota con un literal (de la **A** a la **N**), solo se dispone de una puerta de acceso localizada en uno de los extremos del recinto, el cual presenta alta circulación de personas durante buena parte del día. También se indica la posición inicial del robot y el recorrido diseñado para la exploración del entorno.

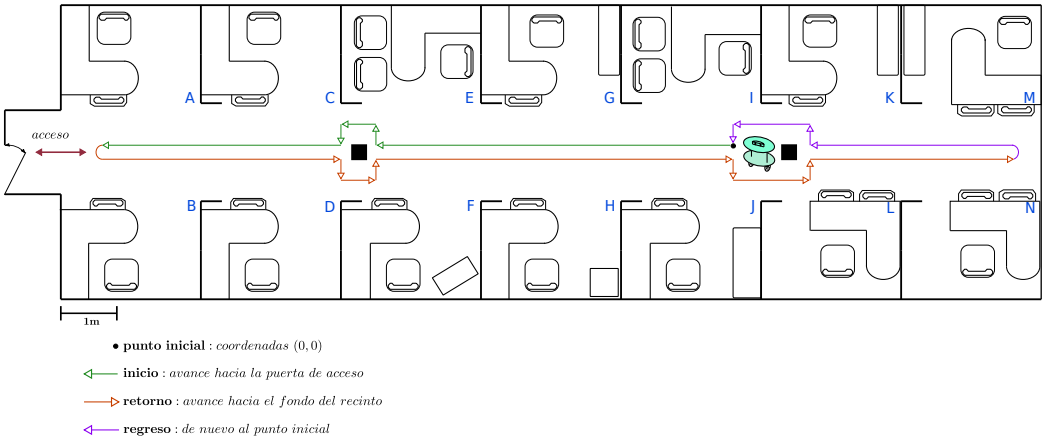


Figura 3.6 Plano arquitectónico del entorno experimental y recorrido diseñado para la teleoperación del robot.

La secuencia de pasos para los experimentos SLAM con `gmapping` es la siguiente:

Paso 1 Configurar parámetros de la red ROS en los archivos `.bashrc` de los computadores Workstation y Robot PC (ver sección 3.1.4)

Paso 2 Establecer la conexión inalámbrica entre ambos computadores a través de la red de área local `wireless-bone` provista por el router TP-LINK

Paso 3 Abrir en el Workstation dos terminales para el acceso remoto vía `SSH` del Robot PC:

WS-T1: `$ssh -C -Y turtlebot@192.168.1.101`

WS-T2: `$ssh -C -Y turtlebot@192.168.1.101`

donde WS-T1 y WS-T2 son terminales de comandos abiertas en el Workstation. Una vez se ingresa la clave del equipo remoto (Robot PC), las terminales se renombran como RPC-T1 y RPC-T2, en referencia al computador remoto.

Paso 4 Lanzar en la terminal RPC-T1 los nodos de inicialización del robot móvil:

RPC-T1: `$roslaunch turtlebot_bringup minimal.launch`

Paso 5 Localizar el robot en una posición y orientación convenientes antes de lanzar los nodos SLAM. En caso de usar un sensor RGB-D como escáner láser, entonces se deberá exportar la variable de entorno correspondiente:

RPC-T2: `$export TURTLEBOT_3D_SENSOR=kinect`

RPC-T2: `$roslaunch turtlebot_navigation gmapping_de\`
`mo.launch`

Paso 6 Conviene visualizar en el Workstation el proceso de construcción del mapa a medida que se explora el entorno, para ello se usa el visor `rviz` de ROS:

WS-T3: `$roslaunch turtlebot_rviz_launchers view_navi\`
`gation.launch`

Paso 7 Conectar el hardware de teleoperación al Workstation PC, habilitar el puerto serial y correr el nodo de implementación del protocolo `rosserial`:

WS-T4: `$sudo chmod 777 /dev/ttyUSB0`
WS-T4: `$rosrun rosserial_python serial_node.py /dev\`
`/ttyUSB0`

Paso 8 Lanzar el nodo de teleoperación en el Workstation PC:
WS-T5: `$rosrun odom_ros teleop_joy.py`

Paso 9 Teleoperar la plataforma móvil a través del entorno

Paso 10 (Opcional) También es posible verificar el mapeo desde una terminal del Workstation:

WS-T6: `$rostopic echo /map/data -n1`

Paso 11 Salvar el mapa una vez concluida la exploración del entorno, para ello se ingresa en una terminal del Workstation el comando:

WS-T7: `rosrun map_server map_saver -f ofic_map`

La ejecución de este comando crea en el directorio actual los archivos `ofic_map.pgm` y `ofic_map.yaml`. El primero corresponde al mapa 2D del entorno y el segundo a los meta-datos asociados al mapa.

Paso 12 (Opcional) Se puede usar un visor de imágenes como por ejemplo `eog` para darle un vistazo al mapa construido:

WS-T7: `$eog ofic_map.pgm`

Durante su construcción el mapa es publicado en el topic `/map` con el tipo de mensajes `nav_msgs/OccupancyGrid` a una rata promedio de 0,068 s. La probabilidad de ocupación de cada celda esta en el rango `[0, 100]`, donde los valores extremos se interpretan como:

- **0**: espacio completamente libre
- **100**: espacio completamente ocupado
- **-1**: valor especial que denota desconocimiento de la probabilidad de ocupación de la celda, caso que se da cuando parte del espacio aún no ha sido explorado por el robot

Múltiples experimentos de mapeo con `gmapping` fueron realizados. En la figura 3.7 se aprecian dos de los mapas elaborados, los cuales se han sobrepuesto al plano arquitectónico del recinto.

En general, ambas representaciones reflejan las características geométricas del entorno, aunque, como es de esperarse, su aspecto dista de ser el que comúnmente se asocia a los mapas cartográficos o planos arquitectónicos.

En el mapa de la figura 3.7 a) se observa una desviación pronunciada de los cubículos **M** y **N**, localizados al fondo del salón, con relación al plano arquitectónico. De otro lado, en el mapa de la figura 3.7 b) tal desviación es considerablemente menor. Conviene anotar que este fue el último sector en ser explorado durante la teleoperación del robot, pudiendo ser causa de esta distorsión los errores odométricos acumulados.

En ambos mapas, o bien las paredes coinciden con el plano, o bien se localizan hacia su interior o su exterior. También se puede observar como el contorno de los objetos (paredes, columnas, divisiones, mobiliario, etc.) no aparecen como líneas bien definidas sino más bien como líneas difuminadas, producto de los métodos probabilísticos usados para modelar la incertidumbre en los comandos y medidas.

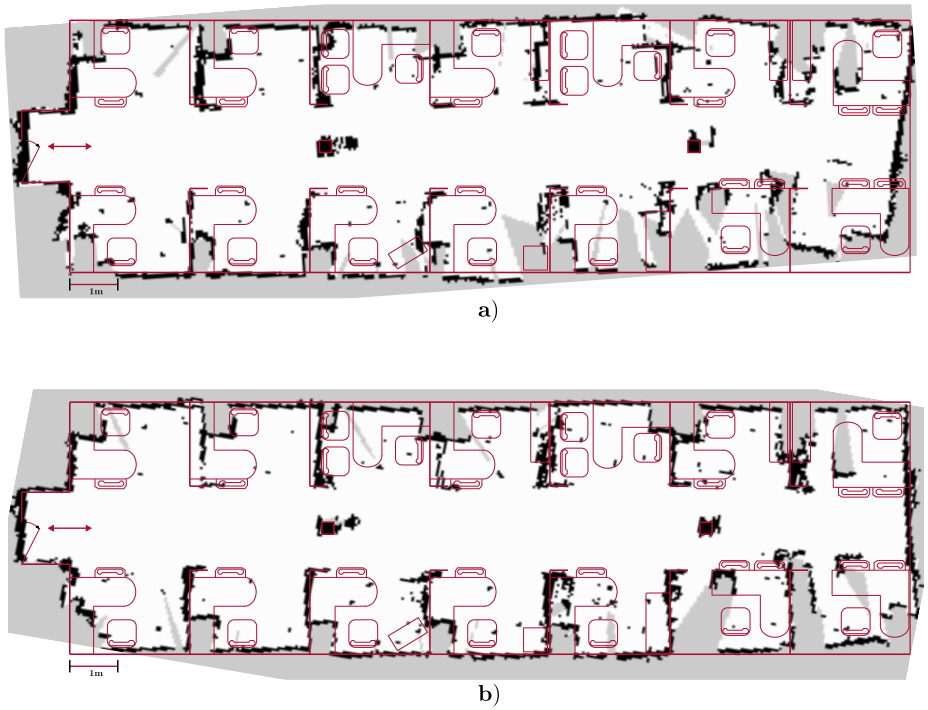


Figura 3.7 Dos mapas obtenidos en pruebas experimentales se han sobrepuesto al plano arquitectónico del entorno. En **a**) las paredes que delimitan los cubículos **M** y **N** presentan un desfase con respecto al plano. En **b**) este sector del mapa se ajusta de mejor forma al plano.

Como se mencionó anteriormente, del proceso de mapeo se obtienen dos archivos de salida:

- Archivo `.pgm`: imagen del mapa elaborado en la que los píxeles más claros (blancos) corresponden a espacio libre de obstáculos, los más oscuros (negros) a espacio ocupado y aquellos entre claro y oscuro (grises) denotan un grado de ocupación desconocido. En tal caso no se puede afirmar si dicho espacio se encuentra o no ocupado

- Archivo `.yaml`: describe meta-datos del mapa, incluyendo el nombre del archivo de la imagen del mapa

Por ejemplo, el contenido del archivo `ofic_nav4.yaml` correspondiente al mapa de la figura 3.7 b) es el siguiente:

```
image: ofic_nav4.pgm
resolution: 0.050000
origin: [-12.200000, -12.200000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

donde cada campo tiene el siguiente significado:

- `image`: ruta (*path*) de la imagen que contiene los datos de ocupación

- `resolution`: resolución del mapa en $[m/pixel]$. Un píxel en la imagen corresponde a 0,05 m

- `origin`: localización en el mapa del píxel en la esquina inferior izquierda. Sus coordenadas son (x, y, yaw), donde yaw corresponde al ángulo de orientación del robot (ángulo ψ en la figura 3.1)

- `negate`: bandera de inversión de la semántica de ocupación de las celdas en el mapa. Cuando esta bandera es cero, el espacio libre se representa con píxeles blancos y el ocupado con píxeles negros. Si es uno se da lo contrario

- `occupied_thresh`: los píxeles con un valor de probabilidad mayor a este valor se consideran completamente ocupados y se colorean de negro (si `negate` = 0)

- `free_thresh`: los píxeles con un valor de probabilidad menor a este valor se consideran completamente libres y se colorean de blanco (si `negate` = 0)

En la figura 3.8 se presentan cuatro de los mapas elaborados con el sistema RobSLAM empleando el algoritmo `gmapping`. Se observa la similitud en las trayectorias estimadas durante la exploración del entorno, lo que permite poder aplicar métricas para evaluar la calidad de los mapas obtenidos (ver sección 3.1.8). La notación `g_mpX` se refiere a un mapa elaborado con `gmapping`, donde `X` es el número del mapa obtenido del experimento correspondiente.

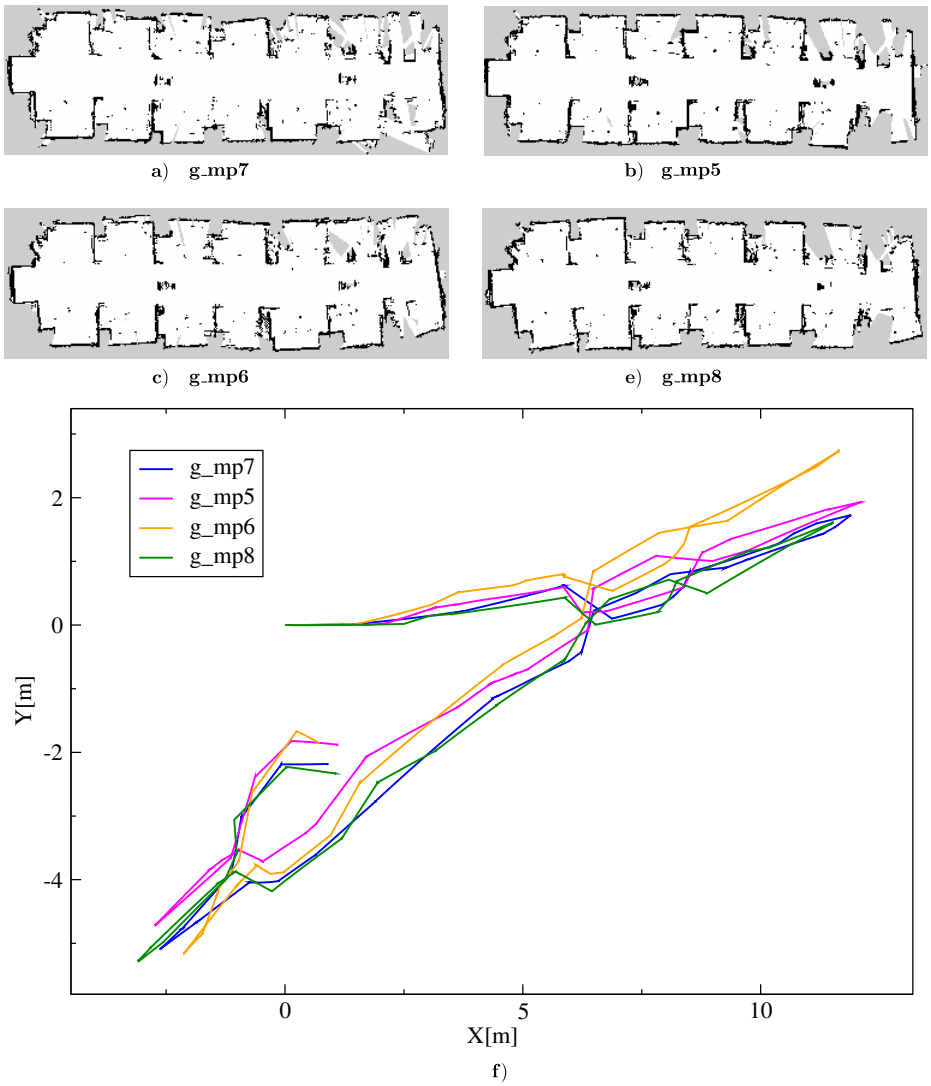


Figura 3.8 Mapas elaborados con el sistema RobSLAM, gmapping y las trayectorias estimadas del robot durante la exploración del entorno.

3.1.7. Experimentos SLAM con RTAB_map

Mientras que en la sección anterior (3.1.6) se empleó `gmapping`, un método SLAM basado en el filtro de partículas *Rao-Blackwellized*, en ésta se empleará como alternativa el método `RTAB_map`, un algoritmo graph SLAM visual basado en la detección de bucles cerrados (ver sección 2.3.4) [82].

`RTAB_map` hace uso de "*palabras visuales*", o características extraídas de una secuencia de imágenes RGB, para estimar la localización del sensor (cámara) con respecto a dichas imágenes. Para ello emplea el detector y descriptor `SURF` (*Speeded Up Robust Features*). También hace uso del algoritmo *bag-of- words* en el que las palabras visuales son almacenadas en un "*diccionario visual*" para establecer si la ubicación actual ha sido previamente visitada o no [81], [83].

Cabe destacar que los bucles cerrados pueden ser detectados en un mapa o en mapas diferentes, esto gracias a que `RTAB-map` provee capacidades de mapeo de entornos de grandes dimensiones a partir de múltiples sesiones. Así mismo, la integración de `RTAB-map` con ROS es soportada por el paquete `rtabmap_ros`, con el que se obtienen mapas 3D en forma de nube de puntos, odometría visual y mapas 2D de celdas de ocupación haciendo uso de sensores RGB-D o cámaras estéreo [83].

Experimentos de mapeo 2D y 3D fueron conducidos con el sistema RobSLAM en el entorno interior de oficinas descrito anteriormente (sección 3.1.6, figura 3.6) haciendo uso de `rtabmap_ros`. Partiendo de la misma posición inicial del robot, se empleó la siguiente secuencia de pasos:

Paso 1 Configurar parámetros de la red ROS en los archivos `.bashrc` de los computadores Workstation y Robot PC (ver sección 3.1.4)

Paso 2 Establecer la conexión inalámbrica entre ambos computadores a través de la red de área local `wireless-bone`, soportada por el router TP-LINK

Paso 3 Abrir en el Workstation dos terminales para el acceso remoto vía `SSH` del Robot PC:

WS-T1: `$ssh -C -Y turtlebot@192.168.1.101`
WS-T2: `$ssh -C -Y turtlebot@192.168.1.101`

donde WS-T1 y WS-T2 son terminales de comandos abiertas en el

Workstation. Una vez se ingresa la clave del equipo remoto (Robot PC), estas terminales se renombran como RPC-T1 y RPC-T2, en alusión al computador remoto.

Paso 4 Lanzar en la terminal RPC-T1 los nodos de inicialización del robot móvil:

RPC-T1: `$roslaunch turtlebot_bringup minimal.launch`

Paso 5 Localizar el robot en una posición y orientación convenientes antes de lanzar los nodos SLAM. En caso de usar un sensor RGB-D como escáner láser, se deberá exportar la variable de entorno correspondiente:

RPC-T1: `$export TURTLEBOT_3D_SENSOR=kinect`
RPC-T1: `$roslaunch rtabmap_ros demo_turtlebot_mapping\ .launch args:="--delete_db_on_start"`

Paso 6 Conviene visualizar en el Workstation el proceso de construcción del mapa a medida que se explora el entorno, para ello conviene usar el visor `rviz` de ROS:

WS-T3: `$roslaunch rtabmap_ros demo_turtlebot_rviz\ .launch`

Paso 7 Conectar el hardware de teleoperación al Workstation PC, habilitar el puerto serial y correr el nodo de implementación del protocolo `rosserial`:

WS-T4: `$sudo chmod 777 /dev/ttyUSB0`
WS-T4: `$rosrun rosserial_python serial_node.py /dev\ /ttyUSB0`

Paso 8 Lanzar el nodo de teleoperación en el Workstation PC:

WS-T5: `$rosrun odom_ros teleop_joy.py`

Paso 9 Teleoperar la plataforma móvil a través del entorno

Paso 10 (Opcional) También es posible verificar el mapeo desde una terminal del Workstation:

WS-T6: `$rostopic echo /map/data -n1`

Paso 11 Una vez concluida la exploración del entorno, el mapa puede ser salvado en el Workstation ingresando el comando:

WS-T7: `rosrun map_server map_saver -f ofic_map`

La ejecución de este comando crea en el directorio actual los archivos `ofic_map.pgm` y `ofic_map.yaml`. El primero corresponde al mapa 2D del entorno y el segundo a los meta-datos asociados

Paso 12 (Opcional) Se puede usar un visor de imágenes como por ejemplo `eog` para darle un vistazo al mapa construido:

WS-T7: `$eog ofic_map.pgm`

Uno de los mapas obtenidos a partir de los experimentos adelantados con `RTAB-map` se observa en la figura 3.9, donde luce superpuesto al plano arquitectónico del recinto. A primera vista se aprecia que los contornos de los objetos se presentan menos definidos cuando se les compara con los mapas elaborados con `gmapping` (ver figura 3.7). A tal efecto se le denominará, en adelante, difuminación o desenfoque (*"blurry effect"*).

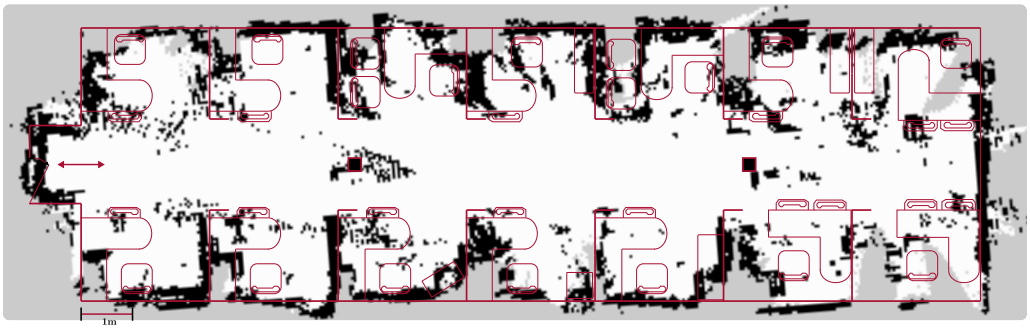
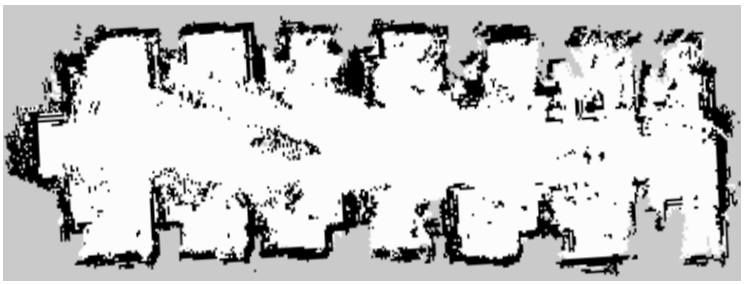


Figura 3.9 Mapa obtenido de uno de los experimentos conducidos con RTAB-map.

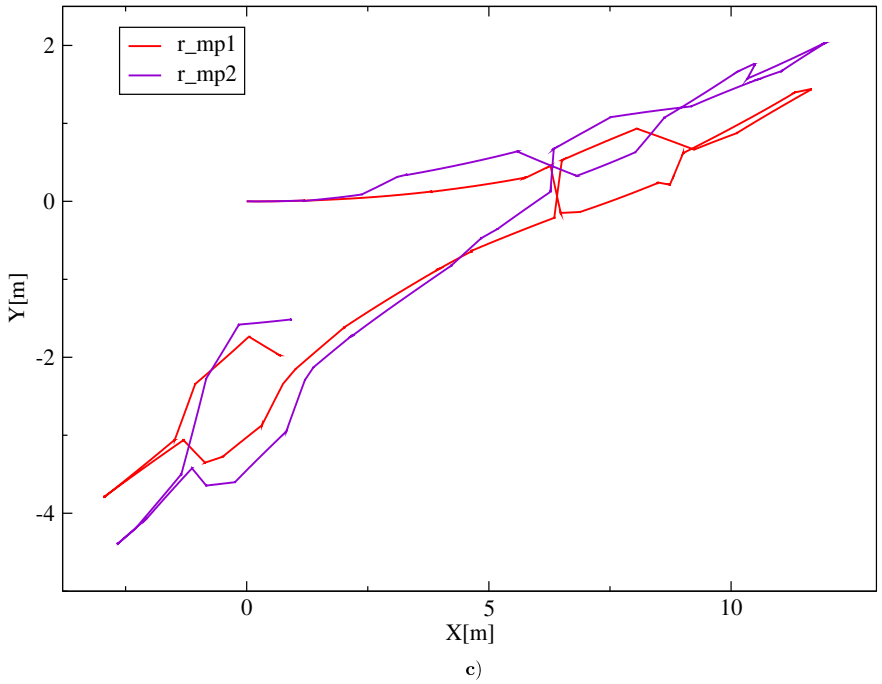
Ejemplos de otros dos mapas realizados con `rtab-map` se exponen en la figura 3.10, junto con las trayectorias estimadas en cada recorrido. La notación `r_mpX` se

refiere a un mapa elaborado con el RTAB-map, donde X es el número del mapa obtenido del experimento correspondiente.



a) r_mp1

b) r_mp1



c)

Figura 3.10 Mapas obtenidos de los experimentos conducidos con el algoritmo RTAB-map (figuras **a**) y **b**). En **c**) se muestran las trayectorias estimadas del robot durante su teleoperación.

3.1.8. Métricas y análisis de resultados

Diferentes métricas para evaluar la calidad de los algoritmos SLAM han sido propuestas, no obstante tal evaluación sigue presentando desafíos importantes debido, principalmente, a las múltiples variantes que pueden ser consideradas a la hora de proponer una métrica.

En [84] una métrica de evaluación se establece a partir de la calidad de los mapas obtenidos. Esta se determina con base en el error entre el mapa generado y las dimensiones reales del entorno. Estas dimensiones se establecen bien sea a partir de mediciones de campo o con base en la disponibilidad de planos precisos del entorno. En la práctica, las verdaderas dimensiones de los entornos en que transcurre nuestra vida cotidiana están raramente disponibles, debido a que resultan imposibles de determinar o porque acarrean costos excesivos [85].

En vista de que la trayectoria seguida por un robot durante la exploración del entorno resulta más fácil de establecer que sus dimensiones, las métricas basadas en la comparación de las trayectorias reales y las estimadas son en la actualidad las de uso más extendido [86], [87].

Otros criterios considerados en la evaluación de los métodos SLAM son [88]:

- Tiempo de procesamiento
- Asignación de recursos (procesadores, memoria, etc.) [89]
- Precisión en la localización [90]
- Calidad del mapa generado [91]
- Robustez al ruido en la odometría [92]
- Robustez al ruido en las medidas de distancia (escáner láser, RGB-D) [92]
- Capacidad de tratar con entornos desordenados y saturados de objetos
- Capacidad para mapear correctamente bucles cerrados

Algunas métricas para la evaluación de algoritmos SLAM 2D focalizadas en la calidad del mapa son descritas en [93]. En los métodos propuestos no se requiere contar, *a prori*, con mapas de dimensiones reales (*ground truth maps*) o trayectorias

reales seguidas por el robot (*ground truth trajectories*). Estas razones soportaron la adopción de algunas de estas métricas como mecanismo de evaluación de los algoritmos SLAM considerados.

La primera métrica, basada en la proporción de celdas ocupadas y celdas libres, permite establecer el grado de difuminación del mapa. Mientras mayor sea esta proporción, menor será la calidad del mapa y menos eficiente será el algoritmo SLAM que lo produjo.

Una segunda métrica se basa en el conteo de esquinas, según la cual, a mayor número de estas menor será la calidad del mapa generado. En efecto, de no presentarse pérdida de información sensorial ni fallos en los algoritmos, el número de esquinas reales presentes en el entorno se verá reflejado en el mapa, en caso contrario aparecerán esquinas adicionales por duplicación de paredes, áreas cerradas que aparecen como abiertas y discrepancias en la trayectoria real versus la estimada [93].

Como se observa en la figura 3.11, los algoritmos de implementación de ambas métricas constan de dos etapas. La primera corresponde a un preprocesamiento en la cual, o bien se remueve ruido del mapa o bien se realza la información de interés contenida en él. La segunda etapa (*procesamiento*) permite extraer información cuantitativa relativa a cada métrica en particular.

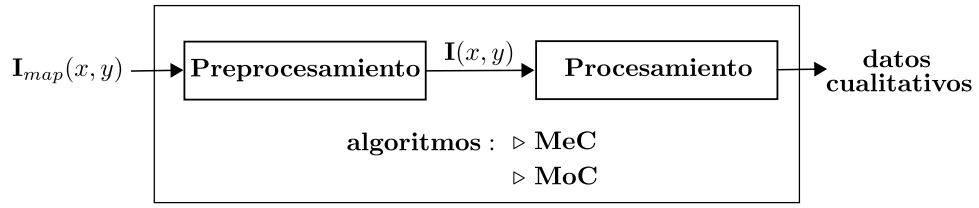


Figura 3.11 El mapa $\mathbf{I}_{map}(x,y)$ es procesado por los algoritmos que implementan las métricas **MeC** y **MoC** con que se evalúan los algoritmos SLAM considerados, $\mathbf{I}(x,y)$ es la versión filtrada de $\mathbf{I}_{map}(x,y)$.

En este trabajo se proponen dos modificaciones significativas relacionadas con las métricas consideradas. Con respecto a la métrica basada en la proporción de celdas ocupadas y libres, en adelante denominada MeC (***Me**tric-of-**C**ells*), el preprocesamiento busca descartar falsas celdas ocupadas producto del ruido en los sensores, el cual finalmente es transferido a los mapas (figura 3.11). Mientras que Filatov *et al.*

en [93] llevan a cabo un preprocesamiento mediante una umbralización con respecto al valor medio de todas las celdas (*píxeles*) en el mapa, en este trabajo se implementa a través de un filtro de mediana (*median filter*), que mantiene los detalles de contraste al no difuminar los bordes. Este tipo de filtro fuerza los puntos de la imagen con valores de intensidad muy diferentes a los puntos circundantes, a adoptar valores más próximos a sus vecinos, de modo que se eliminan picos de intensidad que aparecen en zonas aisladas de la imagen.

En cuanto a la métrica basada en el número de esquinas detectables en el mapa, nombrada en adelante como MoC (***M**etric-**o**f-**C**orners*), el preprocesamiento en [93] se implementa con el operador LoG (***L**aplacian-**o**f-**G**auss*). Este operador corresponde a un filtro de dos etapas, primero se suaviza la imagen empleando un filtro Gaussiano y posteriormente uno derivativo (Laplaciano), que permite determinar áreas con cambios bruscos de intensidad en la imagen (contornos). El suavizado es requerido ya que los filtros Laplacianos resultan ser muy sensibles al ruido. El operador LoG se describe mediante la ecuación 3.8:

$$\mathbf{LoG}(x,y) = -\frac{1}{\pi\sigma^4}\left(1-\frac{x^2-y^2}{2\sigma^2}\right)e^{-\frac{x^2-y^2}{2\sigma^2}} \tag{3.8}$$

En este trabajo el preprocesamiento para la MoC se implementa con un filtro bilateral que permite remover ruido en el mapa a la vez que preserva los bordes. Este filtro remplaza el valor de intensidad de cada píxel en la imagen por una media ponderada de píxeles circundantes (*vecindario*). El peso se puede establecer con base en una distribución Gaussiana.

En el cuadro 3.5 se establecen las diferencias de la implementación en [93] con las métricas MeC y MoC propuestas en este trabajo. En la etapa de procesamiento se emplean los mismos métodos, un conteo separado de celdas libres y ocupadas para la MeC y un detector de esquinas Harris para la MoC.

El algoritmo 1 implementa la métrica MeC. Para su cálculo se diseñó un nodo ROS en el que la imagen del mapa primero se suaviza aplicando un filtro de mediana con *kernel* de 3×3. De esta forma se remueve gran parte de ruido en la imagen afectando lo menos posible las regiones verdaderamente ocupadas por paredes y divisiones. Posteriormente el arreglo bidimensional correspondiente a la imagen filtrada ($\mathbf{I}(x,y)$ en la figura 3.11), es convertido en una estructura ROS de tipo `sensor_msgs/Image`, cuyo campo "`.data`", corresponde a un arreglo unidi-

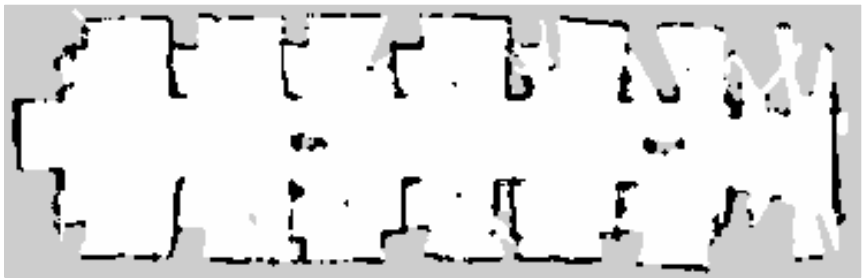
mensional de tipo `str`, el cual es fácilmente iterable mediante un ciclo `for` simple con el cual se clasifican las celdas.

Cuadro 3.5 Detalles de implementación de las métricas MeC y MoC en la evaluación de algoritmos SLAM.

	Preprocesamiento MeC	Preprocesamiento MoC
Filatov *et al.*	Umbralización con el valor medio de todas las celdas	LoG (*Laplace of Gaussian*)
Acosta	El valor de cada celda (píxel) se reemplaza por la mediana del vecindario (*Median Filter*)	Filtro Bilateral (*mantiene los bordes*)

La figura 3.12) muestra el resultado del preprocesamiento en el algoritmo de implementación de la MeC.

a)



b)

Figura 3.12 Al mapa $\mathbf{I}_{map}(x,y)$ en **a**) se aplica un filtro de mediana con *kernel* de 3×3 para remover ruido, como por ejemplo el que se resalta con rectángulos de colores. En **b**) el mapa $\mathbf{I}(x,y)$ se ve más "*limpio*" que el original.

ALGORITMO 1: Cálculo de Métrica MeC

Input: mapa como parámetro en la línea de comandos

Output: proporción de celdas libres y ocupadas en el mapa

1. Leer parámetro de la línea de comandos (*map_file.pgm*);

if $map\_file.pgm$ **then**
  $\mathbf{I}_{map}(x,y) \leftarrow map\_file.pgm$
else
  Desplegar mensaje de error y salir;
end

2. Suavizar imagen aplicando filtro de mediana:

$\mathbf{I}(x,y) \leftarrow \mathbf{T}[\mathbf{I}_{map}(x,y)]$;

3. Crear objeto `cv_bridge` para convertir imagen OpenCV en imagen ROS:

`puente_ros = CvBridge();`

4. Crear estructura de datos ROS tipo `sensor_msgs/Image` a partir de imagen OpenCV:

ros_image = `puente_ros.cv2_to_imgmsg(`$\mathbf{I}(x,y)$`);`

5. Discriminar los tres tipos de celdas (*ocupadas, libres, indeterminadas*):

$valor\_ocupada = 0$;
$valor\_libre = 254$;
$valor\_indet = 205$;

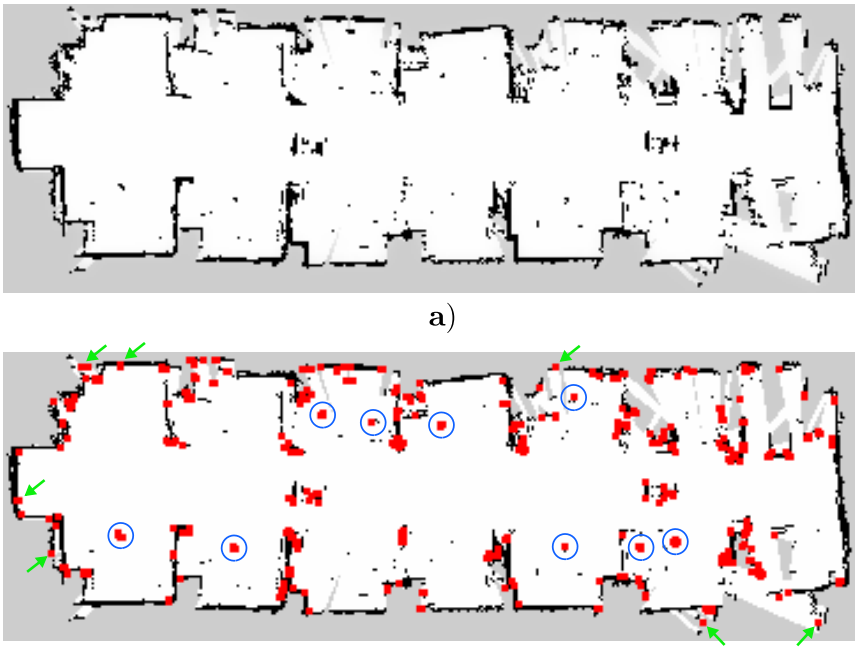
$nro\_ocupadas = 0$;
$nro\_libres = 0$;
$nro\_indet = 0$;

foreach *entrada i en ros_image.data* **do**
  if $ros\_image.data[i] == valor\_ocupada$ **then**
    $nro\_ocupadas = nro\_ocupadas + 1$;
  else if $ros\_image.data[i] == valor\_libre$ **then**
    $nro\_ocupadas = nro\_ocupadas + 1$;
  else
    $nro\_indet = nro\_indet + 1$;
  end
end

6. Devolver proporción de celdas ocupadas: *prop_occ*

La implementación de la métrica MoC basada en esquinas se presenta en el algoritmo 2. Inicialmente se aplica un filtro bilateral para remover ruido a la vez que se mantiene la estructura del mapa (bordes). El resultado es un mapa con fronteras bien delimitadas al que se puede aplicar el operador de `Harris` para la detección de esquinas. Finalmente se lleva a cabo una dilatación de la imagen para marcar las esquinas detectadas y se establece un nivel de umbral para contabilizar las esquinas.

Las esquinas detectadas por el algoritmo MoC para uno de los mapas elaborados se aprecian en la figura 3.13. Se encierran en círculos azules algunas esquinas no estructurales generadas por artefactos que no corresponden a paredes, como pueden ser patas de sillas y mesas. Así mismo se indica con flechas verdes la ubicación de algunas falsas esquinas.



a)

b)

Figura 3.13 El mapa $\mathbf{I}(x,y)$ en **a**) se obtuvo después de filtrar el mapa original mediante un filtro bilateral que remueve ruido y conserva los contornos. En **b**) se aprecian las esquinas detectadas por el método MoC (pequeños cuadros rojos). Se han encerrado en círculos algunas esquinas generadas por artefactos que no corresponden a paredes (p. ej. sillas y mesas) y se indican con flechas verdes algunas "*falsas esquinas*".

ALGORITMO 2: Cálculo de Métrica MoC

Input: mapa como parámetro en la línea de comandos

Output: número de esquinas en el mapa

1. Leer parámetro de la línea de comandos (*map_file.pgm*);

if $map\_file.pgm$ **then**
  $\mathbf{I}_{map}(x,y) \leftarrow map\_file.pgm$
else
  Desplegar mensaje de error y salir;
end

2. Aplicar filtro bilateral para remover ruido en el mapa manteniendo bordes:

Diámetro del vecindario de píxeles para filtrado fuerte: $d = 15$;
$\mathbf{I}(x,y) \leftarrow Bilateral\_Filter[\mathbf{I}_{map}(x,y), d]$;

3. Aplicar operador de `Harris` para detección de esquinas en imagen:

$\mathbf{I}_{esquinas}(x,y) \leftarrow Harris[\mathbf{I}(x,y)]$;

4. Dilatación de imagen para conteo esquinas:

$\mathbf{I}_{dilate}(x,y) \leftarrow Dilate[\mathbf{I}(x,y)]$;
$nro\_esquinas = 0$;
$thres\_esquinas = 0{,}525 * max(\mathbf{I}_{dilate}(x,y))$; // umbral para conteo
$h = nro\_filas[\mathbf{I}_{dilate}(x,y)]$;
$w = nro\_filas[\mathbf{I}_{dilate}(x,y)]$;

for $i = 0$ **to** $h-1$ **do**
  for $j = 0$ **to** $w-1$ **do**
    if $\mathbf{I}_{dilate}(h,w) > thres\_esquinas$ **then**
      $nro\_esquinas = nro\_esquinas + 1$;
    end
  end
end

5. Despliega mapa con esquinas demarcadas;

6. Devolver número de esquinas: *nro_esquinas*

En el cuadro 3.6 se evalúan ocho mapas elaborados con el algoritmo `gmapping`. Mientras que la mejor MeC la presenta el mapa $g\_mp7.pgm$, la mejor MoC corresponde al mapa $g\_mp3.pgm$. También se observa que con base al puesto ocupado

por cada mapa en ambas métricas, el mejor mapa es *g_map*7.*pgm*, que presenta la menor proporción de celdas ocupadas y el segundo más bajo número de esquinas.

Le siguen en su orden los mapas *g_mp*5, *g_mp*6, *g_mp*1 y *g_mp*3, que de acuerdo a los lugares que ocupan en el cuadro 3.6, se pueden considerar como de calidad intermedia. Finalmente, los de menor calidad resultan ser los mapas *g_mp*4.*pgm*, *g_mp*2.*pgm* y *g_mp*8.*pgm*, siendo este último el de menor calidad con respecto a las métricas propuestas en este trabajo.

Cuadro 3.6 Calidad de los mapas elaborados con gmapping de acuerdo con los criterios de evaluación MeC y MoC.

Mapa	**MeC**	**MoC**	**Puesto MeC**	**Puesto MoC**
g_mp1.pgm	0.0083474864	234	2	6
g_mp2.pgm	0.0104268391	216	8	4
g_mp3.pgm	0.0096944173	177	7	1
g_mp4.pgm	0.0090362548	249	4	7
g_mp5.pgm	0.0091894531	190	5	3
g_mp6.pgm	0.0090299479	222	3	5
g_mp7.pgm	0.0071166992	180	1	2
g_mp8.pgm	0.0095703125	260	6	8

En el cuadro 3.7 se ordenan los mapas considerados de acuerdo a los criterios cuantitativos de evaluación MeC y MoC, siendo el primero el mejor mapa elaborado.

Cuadro 3.7 Clasificación de los mapas elaborados con gmapping.

Mapa	**Clasificación**
g_mp7.pgm	1
g_mp5.pgm	2
g_mp6.pgm	3
g_mp1.pgm	4
g_mp3.pgm	5
g_mp4.pgm	6
g_mp2.pgm	7
g_mp8.pgm	8

Para los mapas elaborados con el algoritmo `RTAB-map`, el resultado de la aplicación de las métricas se observa en la tabla 3.8

Cuadro 3.8 Calidad de los mapas elaborados con RTAB-Map de acuerdo con los criterios de evaluación MeC y MoC

Mapa	**MeC**	**MoC**	**Puesto MeC**	**Puesto MoC**
r_mp1.pgm	0.0287981371	372	1	4
r_mp2.pgm	0.0321693755	398	5	6
r_mp3.pgm	0.0303139744	441	2	8
r_mp4.pgm	0.0303271176	295	3	2
r_mp5.pgm	0.0312273590	273	4	1
r_mp6.pgm	0.0336942018	351	7	3
r_mp7.pgm	0.0330023752	376	6	5
r_mp8.pgm	0.0338742141	427	8	7

En el cuadro 3.9 se establece una clasificación de los mapas elaborados con el algoritmo `RTAB-Map`. De acuerdo a las métricas consideradas el mejor mapa es `r_mp4.pgm`, que se localiza en el tercer y segundo puesto de la clasificación MeC y MoC respectivamente. Los mapas `r_mp1.pgm` y `r_mp5.pgm` están prácticamente empatados, no obstante se clasifica primero aquel que presenta menos esquinas, ya que se asume que probablemente contenga menos cantidad de "*falsas esquinas*". Finalmente el mapa `r_mp8.pgm` es el que presenta las métricas más desfavorables.

Cuadro 3.9 Clasificación de los mapas elaborados con RTAB-Map.

Mapa	**Clasificación**
r_mp4.pgm	1
r_mp5.pgm	2
r_mp1.pgm	3
r_mp6.pgm	4
r_mp3.pgm	5
r_mp7.pgm	6
r_mp2.pgm	7
r_mp8.pgm	8

El mosaico de mapas de la figura 3.14 presenta los mejores y peores mapas elaborados con cada uno de los algoritmos considerados y, justo debajo, se indican las esquinas detectadas por el algoritmo MoC.

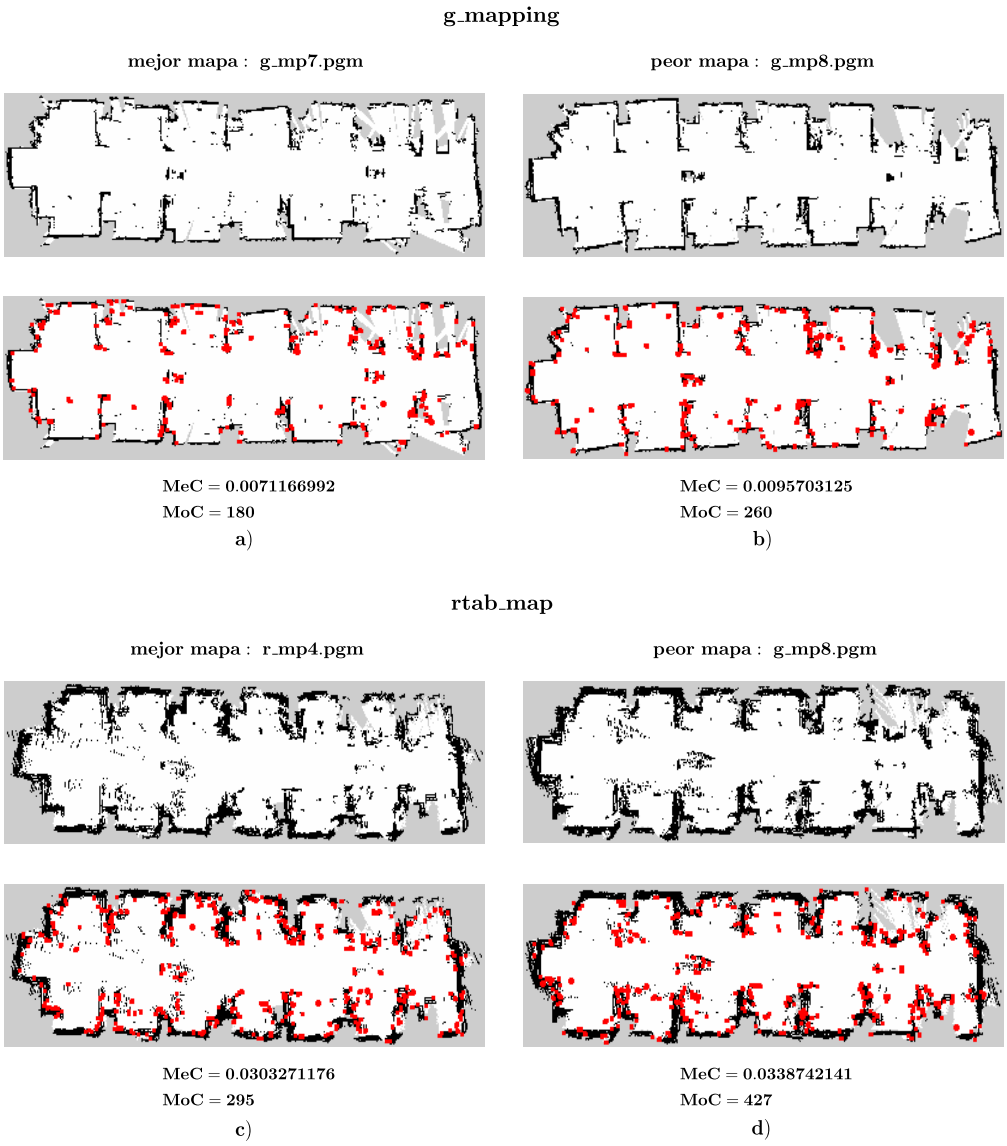


Figura 3.14 Comparativa de los mejores y peores mapas para el mismo espacio de oficinas. Los mapas en las figuras **a**) y **b**) fueron elaborados con **gmapping**, mientras que los mapas de las figuras **c**) y **d**) se construyeron con **rtab_map**.

De acuerdo con las métricas establecidas, los mejores mapas siempre se obtuvieron al emplear `gmapping`, cuyo peor mapa (`g_mp8.pgm`) es incluso de mejor calidad que el mejor mapa elaborado con `rtab_map` (`r_mp4.pgm`). Resulta notable el alto

grado de difuminación que se observa en los mapas elaborados con el algoritmo `rtab_map` (figuras 3.14 **c**) y **d**)), en contraste con los bordes mejor definidos que resultan de emplear el algoritmo `gmapping` (figuras 3.14 **a**) y **b**)).

3.1.9. Respuesta a preguntas de investigación

A continuación se da respuesta a algunas de las preguntas de investigación formuladas en la sección 1.5 de este trabajo:

- ¿Qué estrategias de exploración se pueden implementar para recorrer el entorno de trabajo?

  Respuesta: Dos estrategias pueden ser consideradas a la hora de implementar la exploración de entornos interiores para resolver el SLAM. La primera consiste en una navegación reactiva basada en comportamientos simples como *evadir* obstáculos, *bordear* paredes o *avanzar* a través de espacio libre, entre otros. Tal aproximación constituye, *per se*, un área de investigación en robótica móvil conocida como "*Robótica Basada en Comportamientos*".

  La principal ventaja de esta estrategia radica en el hecho de que el robot quedaría habilitado para explorar su entorno de manera autónoma, no obstante resulta ser la solución con mayor grado de complejidad.

  La segunda alternativa es mucho más simple de implementar y consiste en la teleoperación del robot móvil a través del entorno. Su principal desventaja se debe al hecho de requiere de un teleoperador humano.

  En vista de que es el SLAM y no la navegación autónoma el tema principal de investigación en este trabajo, se optó por la solución más simple, una teleoperación inalámbrica del robot desde el computador de supervisión del sistema RobSLAM (figura 3.3).

3.2. Conclusiones del Capítulo

En este capítulo se condujeron experimentos de validación de dos de los métodos más recientes para la solución del SLAM, cuya discusión teórica se presentó en la sección 2.3.4 (*Algoritmos de solución del SLAM*).

Para tal fin se diseñó e implementó el sistema RobSLAM, un prototipo para la investigación y desarrollo de aplicaciones robóticas totalmente compatible con ROS. El sistema permitió resolver el SLAM para entornos interiores, posibilitando el monitoreo permanente del proceso y la teleoperación inalámbrica del robot móvil desde una estación remota (Workstation PC).

Se alcanzó una autonomía energética de hasta tres horas y media para el robot móvil, uno de los componentes del sistema RobSLAM, al incorporar un sistema regulado de alimentación compuesto de un batería de `12V`, `5.5Ah` y un conversor `DC/DC`. De esta manera se consiguió suplir la demanda de energía de la unidad de procesamiento del robot móvil. En efecto, debido a los altos requerimientos de procesamiento que impone la solución del SLAM, fue necesario incluir en el diseño del robot móvil un mini-ordenador de altas prestaciones, de aquí la necesidad de contar con un mayor suministro de energía que el provisto por la batería con la que viene equipada la base móvil Kobuki (`14.8V`, `2200mAh`).

Un sistema simple de teleoperación para el robot móvil se diseñó empleando un microcontrolador Arduino Mega al que se conecta un joystick de dos grados de libertad. El microcontrolador corre un nodo de supervisión del joystick y envía, a través del puerto serial, la información pertinente al Workstation PC. Un nodo diseñado en python, que se ejecuta en el Workstation, se encarga de procesar los mensajes procedentes del hardware de teleoperación y generar comandos de velocidad con transiciones suaves que son enviados al robot móvil.

Las curvas de aceleración propuestas en las ecuaciones 3.6 y 3.7 permitieron una teleoperación suave del robot móvil a través del entorno. Sin embargo, un tiempo para el ajuste de ecuaciones, adaptación y entrenamiento del personal de teleoperación fue requerido antes de poder iniciar las pruebas SLAM.

En este trabajo se propusieron dos variantes a las métricas consideradas en [93] como método de evaluación de los algoritmos SLAM considerados (ver cuadro 3.5). Las métricas, denominadas MeC y MoC, mostraron una mejor consistencia y robustez del algoritmo `gamapping` que el de su contraparte `RTAB-Map`. La principal

ventaja de las métricas consideradas consiste en que no es necesario disponer, *a priori*, de mapas de dimensiones reales o a escala del entorno y tampoco de las trayectorias reales. La eficiencia de los algoritmos SLAM pudo determinarse con base en criterios relativos a la calidad de los mapas elaborados por cada una de las alternativas consideradas.

Cabe anotar que los métodos MeC y MoC aquí propuestos, no consideran la similitud morfológica de los mapas con relación al entorno real como un criterio de evaluación de los algoritmos SLAM. Esto explica porque los mejores mapas no necesariamente coinciden con aquellos que mas se “asemejan” al entorno real.

Capítulo 4

Reconstrucción tridimensional del entorno

En sus inicios, el problema del SLAM se formuló para incrementar la autonomía de robots móviles que operaban principalmente en entornos interiores. Se procesaba la información provista por sensores propioceptivos como encoders rotativos y unidades inerciales de medida (IMU *Inertial Measurement Units*), y sensores de distancia basados en láser y ultrasonido. Sin embargo, este tipo de dispositivos proporcionan ninguna o muy poca información tridimensional (3D) del entorno [94].

En años recientes, la solución del SLAM haciendo solamente uso de sensores de visión o cámaras, se ha posicionado como un tema principal de investigación en la robótica, en virtud de que este tipo de sensores ofrecen una serie de ventajas como peso y dimensiones reducidas, bajo costo, bajo consumo (por ser sensores pasivos), operación tanto en interiores como exteriores y porque proveen información rica y densa de la escena [95].

Por otra parte su uso introduce otro tipo de problemas, como una mayor complejidad y costo computacional debido al gran volumen de información a procesar y errores en los datos debido a baja resolución del dispositivo, cambios de iluminación en el entorno, superficies con falta de textura e imágenes borrosas por movimientos rápidos de la cámara [96].

En este capítulo se aborda el problema de la estimación simultánea del modelo 3D del entorno, y la localización relativa del sensor (o robot) en dicho modelo. A este problema se le denomina *SLAM visual* o *vSLAM*, y hace parte del estado del arte de la

investigación en el campo de la robótica. Para el estudio del SLAM visual llevado a cabo en este trabajo, se discuten inicialmente aquellos elementos de fundamentación teórico-conceptual asociados al problema, tales como el modelamiento y calibración del sensor, la geometría epipolar o de vista múltiple, el modelo de movimiento de la cámara y la reconstrucción 3D de la escena.

De manera particular se discute el algoritmo SfM (*Structure from Motion*) incremental, el cual provee las bases para solucionar el SLAM monocular en tiempo real [97], [98]. Así mismo se revisan los algoritmos más representativos del SLAM visual.

4.1. Reconstrucción 3D

En esta sección se discuten los principios matemáticos asociados a la reconstrucción 3D. Para entender mas fácilmente como los objetos tridimensionales presentes en una escena pueden reconstruirse mediante una secuencia de imágenes bi-dimensionales (2D), resulta conveniente conocer como se da el proceso inverso, es decir, como se forman las imágenes 2D en una cámara.

Para tal fin, primero se discute el modelo de cámara que será usado en lo que resta de este trabajo. Seguidamente se formulan los problemas de reconstrucción, estimación de la posición y se discute el método BA (*Bundle Adjustment*) para reducir el error en la estimación de la reconstrucción 3D de la escena y el desplazamiento 3D de la cámara.

4.1.1. Modelo de Cámara Estenopeica

El modelo del proceso de formación de imágenes en una cámara se puede plantear al considerar una cámara estenopeica. En su forma más simple, una cámara estenopeica no es más que una caja sellada en la que la luz solo puede ingresar a través de un orificio de tamaño muy reducido denominado estenopo [1] (*pinhole* en inglés) [99]. Cabe mencionar que ningún elemento óptico o lente es incorporado al estenopo, en adelante denominado *Centro de Proyección*.

[1] Del griego $\sigma\tau\acute{\epsilon}\nu\omega$ (steno) que significa estrecho y ó$\pi\acute{\eta}$ (ope) que significa apertura.

En la figura 4.1 se aprecia como los rayos de luz que pasan a través del centro de proyección forman una imagen 2D del entorno exterior 3D, denominado escena [100]. La imagen proyectada en la superficie posterior de la cámara, denominada "*plano de la imagen*", aparece invertida o en "*negativo*", a una distancia focal f del centro de proyección.

Con el fin de evitar las imágenes negativas, resulta conveniente considerar un plano virtual paralelo al plano de la imagen, localizado entre la escena y el centro de proyección a la misma distancia focal f.

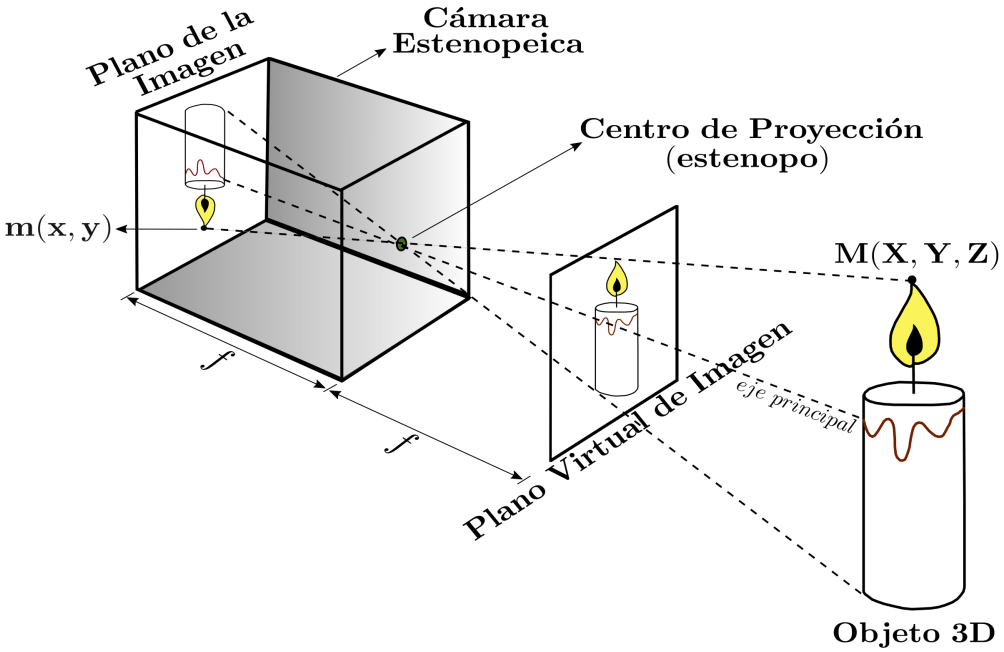


Figura 4.1 Cámara estenopeica: una *imagen negativa* (invertida) de la escena 3D se forma en la cara posterior de cámara cuando los rayos de luz ingresan por el centro de proyección (estenopo). Una imagen positiva se obtiene en el plano virtual de la imagen, localizado entre el centro de proyección y la escena, a una distancia f. El eje principal de la cámara corresponde a la línea que pasa por el centro del estenopo y que es ortogonal al plano de la imagen.

La construcción de un modelo matemático del proceso de formación de imagen, se facilita notablemente si se incorpora un sistema de coordenadas ortonormal de mano derecha, cuyo origen coincida en el centro de proyección. Además el eje principal de la cámara se debe hacer coincidir con el eje Z del sistema de coordenadas.

Tal y como se muestra en la figura 4.2 (arriba), el sistema de coordenadas 3D vinculado a la cámara, induce un sistema 2D en el plano de la imagen. La intersección de los ejes u y v en el plano, cuya ecuación viene dada por $Z = f$, se denomina el *punto principal* **p** [100].

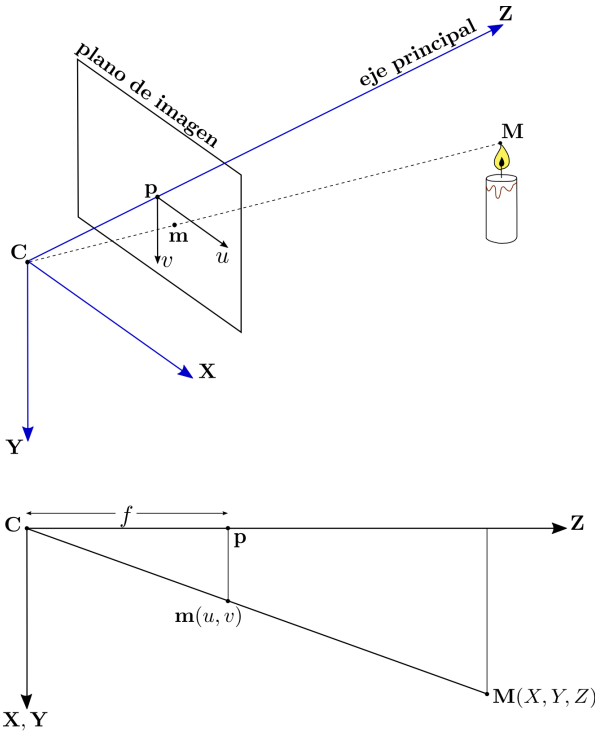


Figura 4.2 Geometría de la cámara estenopeica.

Mientras que el punto **M** tiene coordenadas (X,Y,Z) en $\mathbb{R}^3$, el punto **m**, por el teorema fundamental de semejanza de triángulos (figura 4.2 abajo), tendrá coordenadas (u,v) en $\mathbb{R}^2$ dadas por la ecuación 4.1:

$$u = f\,\frac{X}{Z}$$

$$v = f\,\frac{Y}{Z}, \tag{4.1}$$

donde f es la distancia focal.

Las coordenadas métricas u, v dadas por la ecuación 4.1, deberán ser transformadas a coordenadas de píxeles [2], el sistema coordenado de las imágenes digitales que proveen los sensores de visión en la actualidad.

Tal y como se muestra en la figura 4.3, los píxeles se organizan en una rejilla rectangular de filas y columnas, a semejanza de una matriz, donde cada entrada se asocia a la coordenada de un píxel. Se observa como el origen de coordenadas de píxel (sistema **x**-**y**) se localiza en la esquina superior izquierda, en lugar del punto **p**.

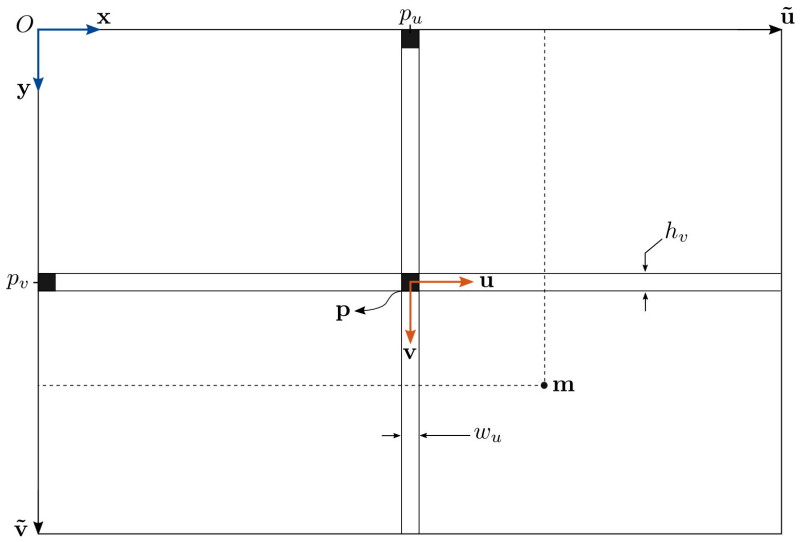


Figura 4.3 Transformación de coordenadas métricas a coordenadas de píxel en el sistema **x**-**y**.

Por lo tanto, para establecer las coordenadas de **m** en el sistema **x**-**y** se deberán llevar cabo las siguientes transformaciones [100]:

- Las coordenadas métricas u, v han de ser transformadas en coordenadas métricas $\tilde{u}, \tilde{v}$ (figura 4.3)
- Las coordenadas métricas $\tilde{u}, \tilde{v}$ se deberán transformar en coordenadas de píxeles relativas al sistema **x**-**y**

Como se nota en la figura 4.3, en el sistema **ũ**-**ṽ** el punto **m** tendrá coordenadas

[2] Un píxel es la menor unidad homogénea en blanco y negro, escala de grises o color que compone una imagen digital

$$\tilde{u} = f\,\frac{X}{Z} + p_u$$

$$\tilde{v} = f\,\frac{Y}{Z} + p_v \tag{4.2}$$

En la ecuación 4.2 las coordenadas aún se expresan en unidades métricas, siendo (p_u, p_v) las coordenadas del punto principal **p** referidas al nuevo sistema. La conversión al sistema **x**-**y** de imagen digital se obtiene dividiendo las coordenadas en la ecuación 4.2 por las dimensiones de cada píxel.

Siendo w_u, h_v el ancho y alto de cada píxel expresado en unidades métricas, y m_u, m_v sus respectivos inversos multiplicativos, entonces las coordenadas (x, y) de la proyección bi-dimensional **m** del punto **M** en la escena (figura 4.2), vendrían dadas por la ecuación 4.3 [100].

$$x = m_u\,\left(f\,\frac{X}{Z} + p_u\right)$$

$$y = m_v\,\left(f\,\frac{Y}{Z} + p_v\right) \tag{4.3}$$

La distancia focal de la cámara expresada en dimensiones de píxel en la imagen (sistema **x**-**y**) viene dada por los productos $\alpha_x = f m_u$ y $\alpha_y = f m_v$. Similarmente, el punto principal **p**, en dimensiones de píxel, tendrá coordenadas (p_x, p_y) dadas por $p_x = m_u p_u$ y $p_y = m_v p_v$ respectivamente [101]. De esta manera la ecuación 4.3 puede reescribirse como:

$$Zx = \alpha_x X + Z p_x$$

$$Zy = \alpha_y Y + Z p_y \tag{4.4}$$

Puesto que el punto **m** se puede representar en coordenadas extendidas de píxel mediante el vector columna $\mathbf{m} = (x, y, 1)^T$, entonces la ecuación 4.4 se puede escribir equivalentemente como:

$$Z\mathbf{m} = Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{4.5}$$

donde la matriz triangular superior de 3×3 en la ecuación 4.5 incorpora cinco parámetros *intrínsecos* que son propios de la cámara y se denota como:

$$K = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

La matriz K (ecuación 4.6), conocida como *matriz de parámetros intrínsecos* o simplemente *matriz de calibración*, representa la proyección de una escena 3D en unidades métricas en una imagen digital 2D en unidades de píxeles [102].

Una forma más general de la matriz de calibración se presenta en la ecuación 4.7, donde se incorpora el parámetro s o *factor de asimetría*. Este parámetro deberá ser diferente de cero cuando los ejes x e y en el sensor CCD (charge-coupled device) de la cámara no son perpendiculares entre sí, es decir cuando los píxeles no son rectangulares sino que presentan formas de romboides [101].

$$K = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

Los demás parámetros intrínsecos incluidos en la matriz K (4.7) son:

- $\alpha_x = m_u f$: longitud focal en número de píxeles en dirección horizontal (**x**)
- $\alpha_y = m_v f$: longitud focal en número de píxeles en dirección vertical (**y**)
- $m_u = \frac{1}{w_u}$: ancho de píxel `[m/píxel]`
- $m_v = \frac{1}{h_v}$: alto de píxel `[m/píxel]`
- $(p_x, p_y)^T$: coordenadas en píxeles del punto principal

Los parámetros contenidos en la matriz K se pueden obtener a través de un proceso de calibración [102]. La ecuación 4.5, usada para mapear un punto $\mathbf{M} =$

$(X,Y,Z)^T$ del espacio de coordenadas del mundo al sistema de coordenadas de píxel de la imagen digital, se puede expresar en notación compacta como:

$$\rho\mathbf{m} = \mathbf{KM}, \tag{4.8}$$

donde $\rho \in \mathbb{R}^+$ representa la distancia o profundidad del punto $\mathbf{M}$ en la escena con relación a la cámara; en efecto, de las ecuaciones 4.5 y 4.8 se tiene que $\rho = Z$. Al representar el punto $\mathbf{M}$ de la escena como un vector homogéneo $\mathbf{M} = (X,Y,Z,1)^T$, el modelo de la cámara se podrá expresar como una transformación lineal de sus coordenadas homogéneas:

$$\rho \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.9}$$

La ecuación 4.9 se puede escribir de manera compacta como $\rho\mathbf{m} = \mathbf{K}[\mathbf{I}|\mathbf{0}]\mathbf{M}$, donde el centro de la cámara coincide con el origen del sistema de coordenadas.

Cuando los objetos en la escena deben ser representados con respecto a un marco de referencia no centrado en la cámara, conocido como "*sistema de coordenadas del mundo*", se hace necesario introducir una transformación para su conversión en coordenadas de la cámara. Tal transformación requiere de una matriz de rotación $\mathbf{R}$ y un vector de traslación $\mathbf{t}$, que se combinan en la llamada "*matriz de parámetros extrínsecos*", una matriz de transformación homogénea que viene dada por la ecuación 4.10 [103].

$$[\mathbf{R} \mid \mathbf{t}] = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3{}^T & 1 \end{bmatrix} \in \mathbb{R}^{3\times 4}, \tag{4.10}$$

donde $\mathbf{0}_3{}^T$ es una submatriz de transformación de perspectiva nula y el uno en la entrada inferior derecha representa un escalado global unitario.

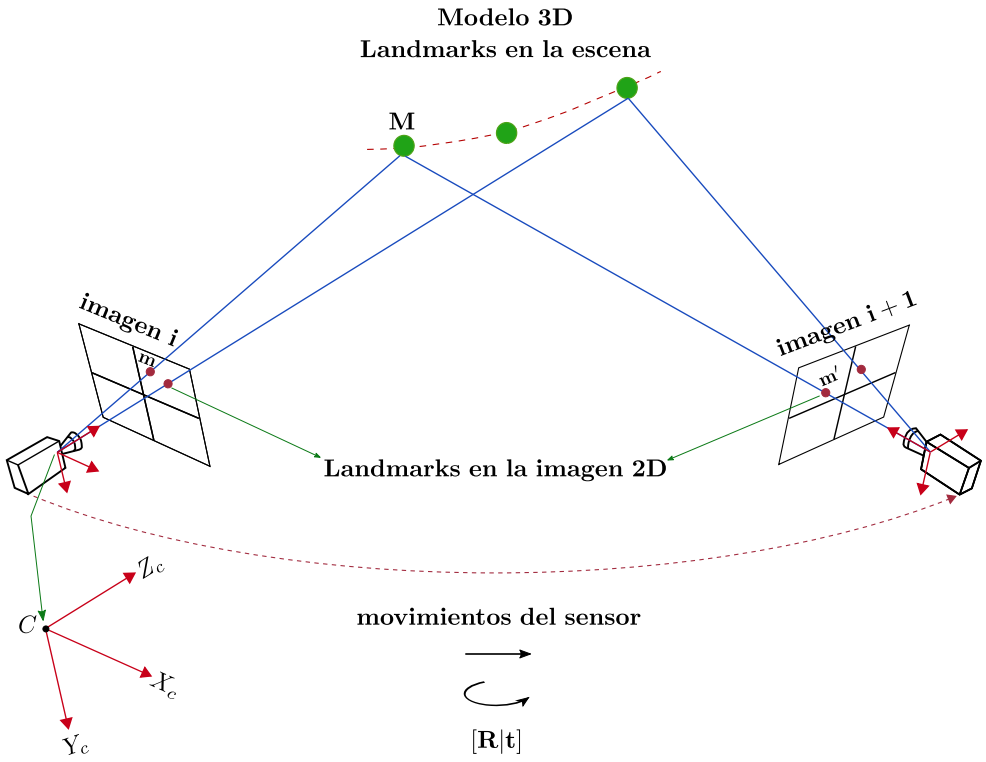


Figura 4.4 Relación entre las coordenadas de la cámara y del espacio tridimensional. La orientación y posición de la cámara en la escena vienen dadas por la matriz $[\mathbf{R}|\mathbf{t}]$. Se muestra en detalle las coordenadas del centro de la cámara C en términos del sistema de coordenadas del mundo.

En la figura 4.4 se observa como las coordenadas del mundo se relacionan con las coordenadas de la imagen. La ecuación 4.11 describe tal relación a partir de la *matriz de proyección* $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$, que combina los parámetros intrínsecos y extrínsecos de la cámara.

$$\rho\mathbf{m} = \mathbf{K} \cdot [\mathbf{I} \mid \mathbf{0}] \cdot [\mathbf{R} \mid \mathbf{t}] \cdot \mathbf{M} = \mathbf{K} \cdot [\mathbf{R} \mid \mathbf{t}] \cdot \mathbf{M} = \mathbf{P} \cdot \mathbf{M} \tag{4.11}$$

Una formulación alternativa, en la cual se consideran las coordenadas del centro de la cámara con respecto a las del mundo (figura 4.4), se presenta en [101], y se expresa mediante la ecuación 4.12.

$$\rho\mathbf{m} = \mathbf{K} \cdot \mathbf{R} \cdot [\mathbf{I}| - \mathbf{C}] \cdot \mathbf{M} \tag{4.12}$$

Al igualar las ecuaciones 4.11 y 4.12 se obtiene el resultado que se indica en la ecuación 4.13.

$$\mathbf{K}\cdot\mathbf{R}\cdot[\mathbf{I}|-\mathbf{C}]\cdot\mathbf{M}=\mathbf{K}\cdot[\mathbf{R}\mid\mathbf{t}]\cdot\mathbf{M}\Rightarrow\mathbf{t}=-\mathbf{R}\cdot\mathbf{C} \tag{4.13}$$

Puesto que la inversa de una matriz ortogonal (**R**) es igual a su transpuesta, entonces al despejar **C** de la ecuación 4.13 se obtiene:

$$\mathbf{t}=-\mathbf{R}\cdot\mathbf{C}\Rightarrow\mathbf{C}=-\mathbf{R}^{\mathbf{T}}\cdot\mathbf{t} \tag{4.14}$$

El resultado obtenido en la ecuación 4.13 establece que una vez se conoce la matriz de proyección de la cámara, entonces es posible determinar su posición.

4.1.2. Geometría epipolar

Se denomina *geometría epipolar*, a las relaciones geométricas que surgen de la intersección entre los planos de dos imágenes de la misma escena, capturadas desde diferentes ubicaciones, con el conjunto de planos que tienen como base la recta (*baseline*) que une los centros ópticos de las cámaras.

Sean $\mathbf{m}_1$ y $\mathbf{m}_2$ las proyecciones, a través de los centros ópticos C_1 y C_2, del mismo punto **M** en la escena sobre los planos de imagen I_{m1} e I_{m2} respectivamente (figura 4.5). Se definen las siguientes relaciones entre puntos correspondientes [103], [104]:

Definición **4.1.** *Al plano que forman el punto* **M** *en el espacio tridimensional y los centros C_1 y C_2, se le denomina "plano epipolar". Para una escena con múltiples puntos se tiene un conjunto de planos, uno por cada punto* $\mathbf{M}_i$.

Definición **4.2.** *La intersección de la línea que une los centros C_1 y C_2 (baseline en la figura 4.5) determina los puntos e_1 y e_2, denominados epipolos, los cuales en coordenadas homogéneas se expresan como:* $\mathbf{e}_1=[e_{1x}\quad e_{1y}\quad 1]^T$, $\mathbf{e}_2=[e_{2x}\quad e_{2y}\quad 1]^T$.

Definición **4.3.** *La intersección del plano epipolar* $\Pi_{\mathbf{M}}$ *con los planos de imagen I_{m1} e I_{m2}, define las líneas epipolares l_1 y l_2 respectivamente. Todas las líneas epipolares pasan a través del epipolo.*

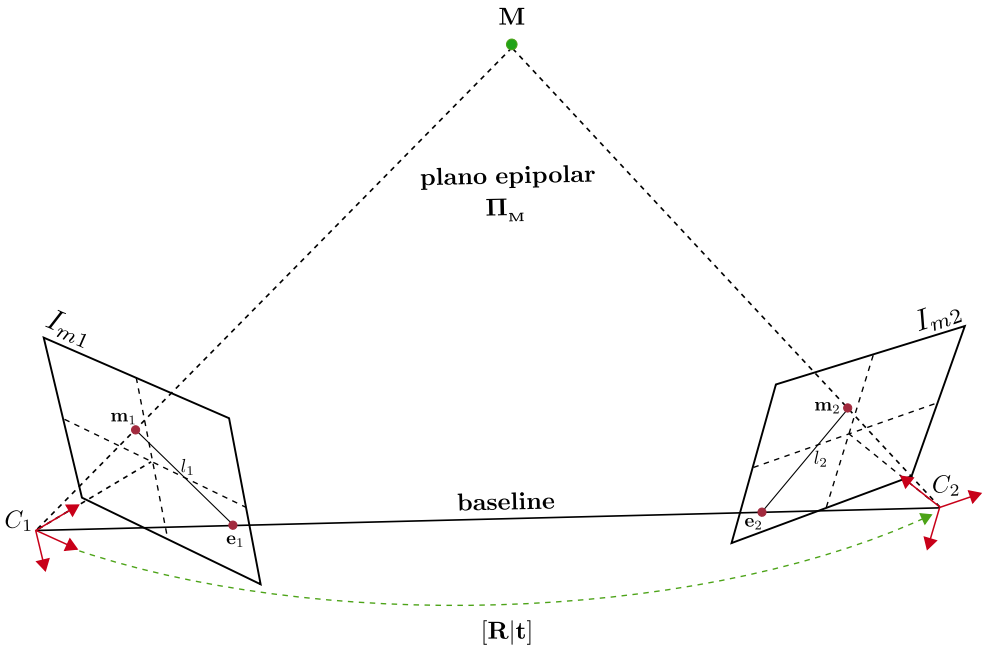


Figura 4.5 Elementos principales de la *geometría epipolar*.

Posiblemente la denominada *restricción epipolar* sea la más importante propiedad de la geometría epipolar. Establece que, conocidos los centros de proyección C_1 y C_2 a partir de la calibración de la cámara, la proyección $\mathbf{m_1}$ en la imagen I_{m1} delimita el espacio de búsqueda de la proyección $\mathbf{m_2}$ en la imagen I_{m2} a su línea epipolar l_2 [105].

Por otra parte, la *matriz fundamental* $\mathbf{F}$ es el principal parámetro de la geometría epipolar, en virtud de que incorpora el grueso de la información relativa a la orientación y posición $[\mathbf{R}|\mathbf{t}]$ entre dos imágenes. En términos de esta matriz, la restricción epipolar se puede formular con base en el teorema 4.1.

Teorema **4.1.** *Dadas dos imágenes del mismo punto* $\mathbf{M}$ *en el espacio tridimensional, ambas caracterizadas por su orientación y posición relativa* $[\mathbf{R}|\mathbf{t}]$ *y sus parámetros internos de calibración* $\mathbf{K}_1$ *y* $\mathbf{K}_2$*, y siendo* $\mathbf{m}_1$ *y* $\mathbf{m}_2$ *las proyecciones de* $\mathbf{M}$ *en los correspondientes planos de imagen* $\mathbf{I}_{m1}$ *y* $\mathbf{I}_{m2}$*, la restricción epipolar puede expresarse como:*

$$\mathbf{m}_2{}^T \mathbf{F} \mathbf{m}_1 = 0 \tag{4.15}$$

donde $\mathbf{F} = \mathbf{K}_2{}^{-T} [\mathbf{t}]_\times \mathbf{R} \mathbf{K}_1{}^{-1}$ *y* $[\mathbf{t}]_\times$ *es una matriz simétrica de* 3×3 *asociada a* $\mathbf{t}$*.*

De acuerdo a lo anterior, la restricción epipolar ofrece ventajas en la implementación computacional de algoritmos para el seguimiento de características en una secuencia de imágenes por las siguientes razones:

- Por cada punto de interés en una imagen, la búsqueda del punto correspondiente en una segunda imagen se restringe a la línea epipolar correspondiente, es decir, a una búsqueda unidimensional en lugar de una bidimensional.
- Alternativamente, dado un conjunto de correspondencias, las restricciones epipolares permiten identificar puntos atípicos (*outliers*) que deberán ser descartados.

4.1.3. Algoritmo SfM

Se denomina *Structure from Motion* (SfM) al proceso de estimación tridimensional de estructuras a patir de secuencias de imágenes capturadas desde ubicaciones diferentes con una o varias cámaras [106].

De la aplicación de los algoritmos SfM se obtiene: 1) los parámetros de la cámara de cada imagen y 2) un conjunto de puntos 3D visibles en las imágenes, los cuales se codifican como *trayectorias* (tracks). Se define *trayectoria*, como las coordenadas 3D de un punto reconstruido de la escena, junto con su lista de coordenadas 2D correspondientes en un subconjunto de imágenes de interés [107].

En años recientes, los algoritmos SfM han cobrado notable interés en la comunidad científica vinculada a la robótica. Esto gracias a que sus algoritmos más recientes han podido ser adaptados para resolver el SLAM en tiempo real, permitiendo obtener reconstrucciones densas del entorno sólo con el uso de cámaras de bajo costo [108], [109].

En la figura 4.6 se presenta la estructura de procesamiento genérica (*pipeline*) de los algoritmos SfM actuales [107], [110].

Figura 4.6 Etapas principales de la cadena de procesamiento **SfM** (*pipeline*).

Se identifican las siguientes etapas:

- **Extracción 2D**: la extracción de características en imágenes considera dos operaciones, detección y descripción. Los *detectores* permiten establecer características relevantes a partir de gradientes representativos de intensidad en los píxeles de la imagen, siendo los más utilizados aquellos que presentan mayor robustez a la rotación y escalamiento de las imágenes.

  Por otra parte, los *descriptores* capturan y sintetizan la información en el vecindario local de píxeles alrededor de las características, facilitando así la asociación entre imágenes.

  En el cuadro 4.1 se resumen las principales ventajas y desventajas de tres de los detectores y descriptores de uso más extendido en la solución del SLAM con base en algoritmos SfM [111], [112], [113], [114].

Cuadro 4.1 Detectores y descriptores usados en SfM.

Detector/Descriptor	**Ventaja**	**Desventaja**
SIFT	robustez a la rotación y escalamiento	alto costo computacional
SURF	bajo costo computacional	pobre manejo de cambios de puntos de vista e iluminación
ORB	invarianza a la rotación y robustez al ruido	sensible a variaciones en la escala

- **Asociación 2D**: en esta etapa los descriptores son comparados con el fin de establecer la correspondencia (*matching*) entre características detectadas en imágenes diferentes. Dadas dos imágenes, la asociación 2D permite obtener un conjunto de pares $(x_i, y_i) \leftrightarrow (x_j, y_j)$ donde, el primer término corresponde a la *i_ésima* característica detectada en la primera imagen, y el segundo, a su correspondiente en la segunda imagen.

- **Generación de trayectorias**: Aquí se verifica que las características correspondientes se asignen efectivamente al mismo punto de la escena, para lo cual,

se estima una transformación basada en geometría proyectiva que mapee los puntos de interés (características) entre imágenes [106].

Dependiendo de la configuración espacial de un par de imágenes, diferentes estrategias son aplicables para describir su relación geométrica: *i*) una homografía [3] para describir la transformación de una cámara con movimiento rotacional puro o que captura escenas planas y *ii*) la geometría epipolar que describe la relación de una cámara en movimiento a través de su matriz fundamental **F** (o su matriz esencial **E**) [106].

Cuando una transformación válida mapea un número suficiente de características entre imágenes, entonces se consideran validadas geométricamente. Para reducir el número de asociaciones incorrectas entre características, se suele emplear un algoritmo de estimación robusta como RANSAC (*Random Sample Consensus*) [107], [115]. La salida de esta etapa se conoce como *grafo de la escena*, cuyos nodos corresponden a las imágenes y los arcos o bordes a los pares verificados entre imágenes.

- **Modelado y refinación SfM**: en los dos últimos componentes de procesamiento que se muestran en la figura 4.6 se incorporan los procesos de inicialización, registro de imagen, triangulación y *Bundle Adjustment-BA*.

  La inicialización crea un mapa inicial a partir de la reconstrucción de dos vistas, puesto que la profundidad no puede ser establecida a partir de un imagen individual. La selección de este par de vistas resulta crítica, ya que la reconstrucción 3D podría no recuperarse como producto de una mala inicialización [116], [106].

  El registro de imágenes es el proceso de hacer coincidir, alinear y superponer dos o más imágenes de una escena capturadas desde ubicaciones diferentes [111], mientras que la triangulación permite reconstruir la geometría 3D a partir de la correspondencia de píxeles en varias imágenes [107], [117].

  Finalmente el modelo es refinado mediante una optimización no lineal iterativa o **BA**, cuyo objetivo es minimizar la distancia entre la retroproyección del modelo tridimensional y los puntos asociados en la imagen [118].

[3] Transformación proyectiva para establecer la correspondencia entre figuras geométricas planas presentes en dos imágenes.

4.2. Algoritmos visual SLAM

Dos métodos principales constituyen el estado del arte del SLAM visual, los basados en características (*Feature-based*) y los directos. En esta sección se discuten algunos de los algoritmos SLAM visual (también conocidos como *vSLAM*) más representativos [119].

4.2.1. Algoritmos indirectos o basados en características

Los métodos basados en características contemplan dos variantes, los que emplean filtros y los que implementan una optimización no lineal (p. ej. **BA**). Se introdujeron para reducir la complejidad computacional requerida en el procesamiento píxel a píxel, al considerar únicamente un conjunto de características sobresalientes y su seguimiento en el flujo normal de imágenes (*tracking*) [95].

MonoSLAM

La primera aproximación vSLAM monocular fue propuesta por Davison en [8] y es considerada como un método representativo de los algoritmos basados en filtros. En esta aproximación el movimiento de la cámara y la estructura 3D del entorno se estiman simultáneamente usando un EKF. Su principal desventaja es el incremento del costo computacional en proporción al tamaño del entorno, que obedece al crecimiento del vector de estado a medida que el número de características se va incrementando [94].

PTAM

Es un algoritmo SLAM monocular que ejecuta en paralelo los procesos de estimación de la posición 3D de una cámara (*tracking*) y el mapeo de puntos en la escena. El mapa no se actualiza con cada imagen capturada (*frame*), sino que se desacopla de este flujo de datos seleccionando heurísticamente imágenes clave denominadas *keyframes*. El método *keyframe* mantiene las bondades del algoritmo de optimización **BA**, con lo que el procesador cuenta con más tiempo entre *keyframes* para una mejor elaboración del mapa [97], [120].

ORB-SLAM

Usa el descriptor ORB para llevar a cabo el seguimiento de características. Posee funcionalidades adicionales a PTAM como son la detección de bucles cerrados y un método de optimización del grafo de posicionamiento global [119].

El sistema se compone de tres módulos que se ejecutan como hilos paralelos [86]:

- Un hilo de seguimiento (*tracking*) que localiza la cámara en cada nueva imagen y decide cuando se necesario insertar una *keyframe*.
- Un hilo de mapeo local que procesa las *keyframes* nuevas y desarrolla una optimización **BA** para refinar la reconstrucción en las proximidades de la cámara.
- Un hilo de búsqueda de bucles cerrados que considera cada nueva *keyframe* disponible.

ORB-SLAM2

Si bien el problema vSLAM puede resolverse usando como único sensor una cámara de bajo costo, liviana y de tamaño reducido, el SLAM monocular puede llegar a presentar algunos inconvenientes.

Puesto que con una sola cámara la profundidad no puede ser directamente observable, entonces la escala real del mapa y la trayectoria estimada serán desconocidas. Adicionalmente, la localización del sistema requiere de múltiples imágenes o métodos de filtrado para producir el mapa inicial, ya que no es posible llevar a acabo una triangulación con solo una imagen. Finalmente, se sabe que el SLAM monocular puede llegar a fallar cuando se realizan movimientos puramente rotacionales.

Estos inconvenientes son sorteados al usar sensores que permiten medir directamente la profundidad de los puntos de la escena. Estas razones motivaron a Mur-Artal *et al.* a incorporar nuevas funcionalidades al sistema mono SLAM que inicialmente desarrollaron en [116]. En [121] proponen **ORB-SLAM2**, un sistema cuyas principales contribuciones son:

- Primer sistema SLAM de código abierto que soporta sensores monoculares, estéreo, y RGB-D. Además incluye detección de bucles cerrados, relocalización y rautilización de mapas.
- Incremento de la precisión cuando se usa una cámara RGB-D y una optimización **BA**, en comparación con los métodos ICB (*Iterative Closets Points*) [122].
- Al usar medidas estereoscópicas cercanas y lejanas, en combinación con observaciones monoculares, los resultados estéreo obtenidos son más precisos que los métodos estéreo directos de última generación.
- Incorpora un modo de localización liviano que hace un reuso efectivo del mapa con el mapeo deshabilitado.

RTAB-Map

RTAB-Map (*Real-time Appearance-Based Mapping*) es un algoritmo graph SLAM visual (ver sección 3.1.3) basado en la detección de trayectorias cerradas. El detector usa un algoritmo *bag-of-words* basado en el descriptor SURF. Cada vez que una trayectoria cerrada es detectada, un nuevo arco (*edge*) es incorporado al grafo de posición (ver figura 2.4) de la cámara [40].

Con cada adición de un arco, el grafo de posición o mapa es refinado haciendo uso de **TORO**[4] (*Tree-based netwORk Optimizer*) [83], [81], un optimizador basado en gradiente descendente que distribuye los errores residuales a través del grafo de posiciones [123], [124].

4.2.2. Algoritmos directos

Los algoritmos SLAM directos resuelven el problema conjunto de estimación del movimiento de la cámara y la reconstrucción estructural del entorno considerando las imágenes completas, obviando así la detección y correspondencia de características entre imágenes. Algunos de los métodos más representativos del SLAM directo son:

[4]https://openslam-org.github.io/toro.html

DTAM

El primer algoritmo SLAM directo publicado fue DTAM (*Dense Tracking and Mapping*) [109]. Si bien no implementa la detección de bucles cerrados y una optimización global, introduce el seguimiento a partir de la minimización del error fotométrico entre *keyframes*[119]. DTAM incorpora los siguientes componentes [94]:

- Inicialización del mapa con base en mediciones estéreo
- El movimiento de la cámara se estima mediante generación de vistas sintéticas a partir del mapa reconstruido
- La información de profundidad se estima para cada píxel aplicando procesamiento estéreo multi-línea (*multi-baseline stereo*).

LSD-SLAM

LSD-SLAM (*Large-scale Direct Monocular*) utiliza la alineación directa de la imagen junto con una estimación de mapas de profundidad semi-densos basada en filtros. En este método el mapa global obtenido corresponde a un grafo de posición en el cual las *keyframes* se asumen como vértices y las transformaciones de similitud 3D como bordes [125], [38].

LSD-SLAM emplea una optimización del grafo que permite de manera explícita la corrección de la desviación de escala y la detección de bucles cerrados en tiempo real. El algoritmo corre tres procesos en paralelo después de que la inicialización tiene lugar: seguimiento, estimación de mapas de profundidad y optimización [126].

DSO

El método *Direct Sparse Odometry* DSO, combina un modelo probabilístico en que se minimiza el error fotométrico, con una optimización conjunta de todos los parámetros del modelo, incluyendo la geometría que se representa como profundidad inversa en un marco de referencia y el movimiento de la cámara [127].

DSO permite obtener una estimación robusta de la trayectoria. Si bien es un método directo, genera mapas dispersos 3D con el fin de cumplir requerimientos de tiempo real [119].

4.2.3. Respuesta a Preguntas de Investigación

A continuación se da respuesta a una de las preguntas de investigación formuladas en la sección 1.5 de este trabajo:

- ¿Qué algoritmos emplear para extraer la información de interés de un conjunto finito de imágenes suministradas por una cámara?

  Respuesta: Los algoritmos vSLAM buscan aprovechar los grandes volúmenes de información que proveen las secuencias de imágenes. De acuerdo con la manera en que estos datos son usados, tales algoritmos se pueden clasificar como *dispersos* o *densos*, e *indirectos* o *directos*.

  Con respecto a la primera clasificación, mientras que los sistemas dispersos emplean un conjunto reducido de píxeles en la imagen, los sistemas densos usan todos o la gran mayoría de los píxeles disponibles.

  Puesto que cada una de estas aproximaciones emplea un número diferente de píxeles y regiones en la imagen, los mapas producidos resultan ser notablemente diferentes. Los mapas generados por los algoritmos dispersos son básicamente representaciones aproximadas de la escena en forma de nubes de puntos. Son usadas principalmente para realizar el seguimiento (*tracking*) de las posiciones de la cámara.

  Por otra parte, los sistemas densos entregan representaciones más detalladas de la escena (*mapas densos*) que requieren un hardware de mayores prestaciones y, en el caso de los sistemas de última generación, resulta indispensable disponer de una GPU (*Graphics Processing Unit*).

  En cuanto al segundo criterio de clasificación (*indirectos/directos*), los primeros, también conocidos como basados en características (*Feature-based*), buscan extraer y seguir algunas características relevantes de la secuencia de imágenes, y emplearlas para estimar la trayectoria de la cámara y elaborar una representación tridimensional del entorno. Las características van desde

geometrías simples como aristas o esquinas, hasta descriptores de características mas sofisticados como SIFT, SURF, ORB, etc. En contraste, los métodos directos operan sobre las intensidades de la totalidad de píxeles en la imagen, en lugar regiones particulares (subconjuntos de píxeles).

Otro aspecto a resaltar en lo que respecta a la clasificación de los métodos vSLAM, es que la extracción de características en los sistemas indirectos puede requerir de una considerable cantidad de tiempo. Los métodos directos, al no requerir de una etapa de extracción, disponen de este tiempo para llevar a cabo otro tipo de computación, de tal manera que pueden mantener las ratas de procesamiento de los métodos indirectos.

Finalmente se debe mencionar que los algoritmos basados en características son más robustos a los cambios de iluminación que su contraparte, debido a que no utilizan de manera directa las intensidades de los píxeles.

Con base en las razones anteriormente expuestas, en este trabajo se ha optado por emplear algoritmos indirectos para la reconstrucción de la estructura tridimensional del entorno y la estima de la trayectoria de la cámara.

4.3. Conclusiones del Capítulo

En este capítulo se condujo un estudio sistemático y conciso de los problemas de estimación tridimensional de estructuras (SfM) y el SLAM visual. Inicialmente se discute el modelo de cámara estenopeica (*pinhole*), pasando por las matrices de parámetros intrínsecos $\mathbf{K}$ y extrínsecos $[\mathbf{R}|\mathbf{t}]$, hasta llegar a la geometría epipolar.Una vez establecidos los fundamentos matemáticos asociados al problema SfM, se discute la estructura genérica de procesamiento que permite resolverlo.

Posteriormente, y una vez establecida la relación entre el SfM y el SLAM, se discuten algunos de los métodos que constituyen el estado del arte del SLAM visual en tiempo real y se estableció su clasificación desde dos puntos de vista: 1) de acuerdo a los tipos de mapas elaborados (*dispersos* o *densos*), y 2) de acuerdo a la manera en que procesan los datos (extracción de características o toda la imagen). Gracias a ello fue posible dar respuesta a una de las preguntas de investigación formuladas y seleccionar los algoritmos vSLAM con los que se condujeron los experimentos del siguiente capítulo.

Fue precisamente el desarrollo conceptual claro y conciso de los problemas SfM y vSLAM, el principal aporte de este capítulo, que extiende y complementa el estudio del SLAM presentado en el capítulo 2 y lo sitúa en la actual frontera del conocimiento.

Capítulo 5

Experimentos SLAM monocular

En este capítulo se conducen experimentos de SLAM visual empleando dos de los métodos mas representativos del estado del arte actual. Para la selección, se consideraron diferentes criterios que permitieron la clasificación de los algoritmos de acuerdo a las diferentes aproximaciones que actualmente se emplean en la solución del SLAM visual.

Después de llevar a cabo la comparativa de los métodos considerados, se profundiza en el estudio de las dos alternativas seleccionadas. Posteriormente se describen los experimentos conducidos y se ponderan sus ventajas y desventajas. Finalmente se presentan las conclusiones del capítulo.

5.1. Comparativa de algoritmos vSLAM

En el cuadro 5.1 se presentan algunos de los algoritmos visual SLAM más populares, los cuales se clasifican de acuerdo al tipo de sensores visuales que soportan y al tipo de detector/descriptor que utilizan. Se referencia así mismo el artículo más representativo de cada una de estas aproximaciones.

MonoSLAM fue el primer algoritmo SLAM visual en tiempo real en emplear una cámara como único sensor. En este método se elaboran mapas probabilísticos dispersos basados en características, estimándose los estados de la cámara, las características y las incertidumbres asociadas.

PTAM es un algoritmo monocular también usado en realidad aumentada (RA). Frecuentemente no entrega mapas completos, en su lugar elabora pequeños mapas que incluyen un solo objeto físico del entorno.

Tanto MonoSLAM como PTAM fueron desarrollados en la década pasada y actualmente se consideran algo longevos, motivo por el cual no fueron considerados a la hora de conducir los experimentos.

Cuadro 5.1 Clasificación de algoritmos vSLAM. N.A.: No Aplica, no usa detector/descriptor por tratarse de un método directo.

Algoritmo	**Sensores**	**Detector/Descriptor**	**Referencia**
MonoSLAM	mono	Shi-Tomasi	[6]
PTAM	mono	FAST	[120]
DSO	mono, estéreo	N.A. grad. de inten.	[127]
ORB-SLAM2	mono, estéreo, RGB-D	ORB	[121]
LSD-SLAM	mono	N.A., semi-denso	[125]
RTAB-Map	estéreo, RGB-D	SURF, bag-of-words	[128]
DTAM	mono	N.A., pixel-wise	[109]

La figura 5.1 sintetiza una clasificación basada en el método de extracción de información de la secuencia de imágenes y el tipo de mapa producido. En los mapas dispersos se consideran pequeños subconjuntos de píxeles, precisamente aquellos asociados a las características detectadas y a su vecindario. En contraste, los mapas densos usan la mayoría o la totalidad de los píxeles en la imagen.

Dos de los algoritmos vSLAM más representativos para la elaboración de mapas densos son DTAM y RTAB-Map. Mientras que RTAB-Map se clasifica como un método indirecto, DTAM se considera uno directo. En los métodos indirectos primero se extraen características de interés de la imagen y a partir de ellas se localiza la cámara y se construye el mapa. Por otro lado en los métodos directos se considera la intensidad de los píxeles sin considerar una etapa preliminar de extracción de características.

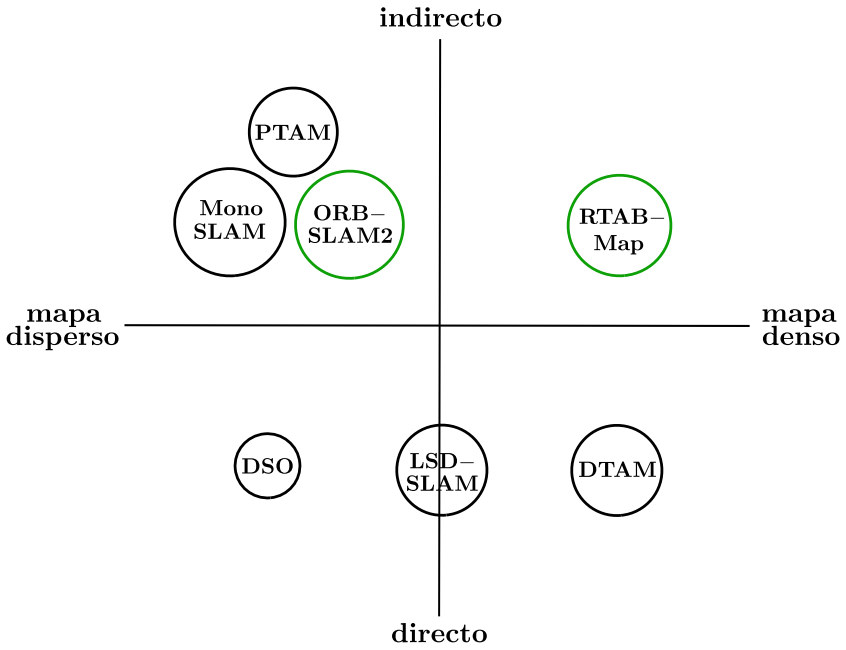


Figura 5.1 Algunos de los métodos SLAM visual más reconocidos.

Un aspecto importante a considerar es que DTAM está diseñado para escenas o espacios pequeños y localizados [129], mientras que RTAB-Map soporta mapeo 2D y 3D de grandes dimensiones y largo plazo.

Otro de los métodos directos que cabe destacar es LSD-SLAM, el cual considera áreas con mayor gradiente de intensidad que las consideradas por DTAM. En tal sentido, LSD-SLAM ignora regiones con baja o ninguna textura para las cuales resulta difícil estimar la profundidad. Por esta razón se considera un método semi-denso [94].

Un algoritmo recientemente propuesto es DSO. En esta aproximación se busca minimizar el error fotométrico del conjunto de píxeles entre imágenes, en lugar de minimizar el error geométrico de retroproyección. DSO lleva a cabo una optimización conjunta de probabilidad de todos los parámetros del modelo [127], [38].

ORB-SLAM2 es una extensión de su antecesor ORB-SLAM [116] que soporta, no sólo cámaras monoculares, sino también estéreo y RGB-D (ver tabla 5.1). ORB-SLAM2 incluye reuso de mapas, detección de trayectos cerrados y capacidades de relocalización. Opera en tiempo real en sistemas de procesamiento estándar y en una amplia variedad de entornos, desde pequeños recintos interiores con secuencias cortas de imágenes capturadas con una cámara RGB portátil, pasando por entor-

nos industriales con secuencias captadas desde drones, hasta grandes secuencias obtenidas desde automóviles que circulan por entornos urbanos [121].

Con base en lo anteriormente expuesto, en los experimentos de este capítulo se usarán los métodos ORB-SLAM2 (componente monocular) y RTAB-Map (componente RGB-D), que se resaltan con círculos verdes en la figura 5.1. Para ello se tuvo en cuenta que:

- Ambos métodos hacen parte del estado del arte actual en lo que al SLAM visual se refiere
- DSO también hace parte de los algoritmos más recientes, sin embargo no lleva a cabo procesos de optimización global y detección de bucles cerrados, ambos soportados por ORB-SLAM2 y RTAB-Map
- DTAM aplica solo para entornos localizados y de dimensiones reducidas, mientras que ORB-SLAM2 y RTAB-Map permiten resolver el SLAM tanto para entornos reducidos como de gran escala
- MonoSLAM no lleva a cabo optimización global y, al igual que PTAM, tampoco realiza la detección de bucles cerrados. Además estos métodos datan de los años 2003 y 2007 respectivamente. El desarrollo de RTAB-Map se remonta al año 2013 y su extensión (usada en este trabajo) al 2017. Por otra parte ORB-SLAM data del año 2014 y su extensión ORB-SLAM2 (empleada en este trabajo) se remonta al 2016
- Si bien el desarrollo de LSD-SLAM se remonta al año 2013, se considera un sistema semi-denso, ubicado a medio camino entre los paradigmas disperso y denso

5.1.1. Experimento *hand-held* ORB-SLAM2

En la figura 5.2 se aprecia el entorno de prueba para el algoritmo ORB-SLAM2 (monocular) junto con sus dimensiones. Corresponde al escritorio localizado en el cubículo **J** del recinto de oficinas considerado anteriormente (ver figura 3.6). Se indican algunas de las posiciones que puede adoptar la cámara en su recorrido.

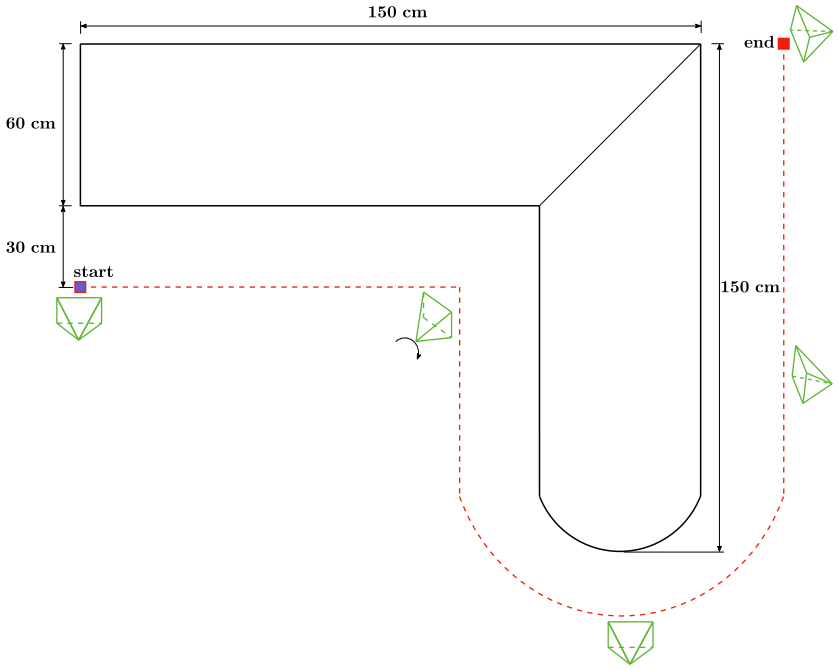


Figura 5.2 Entorno de escritorio en "L" para experimentos con el algoritmo ORB-SLAM2 en su componente monocular. Se indican algunas de las posiciones que puede llegar a adoptar la cámara (en color verde) durante recorridos que se aproximan a la trayectoria considerada en color naranja (línea punteada).

El experimento se conduce sosteniendo la cámara con una mano y siguiendo una trayectoria próxima a la que se indica con color naranja en la figura 5.2. Este tipo de mapeo se conoce en la literatura como *hand-held reconstruction* [130], [131]. Durante el recorrido se buscó que la cámara apuntase en todo momento al escritorio, a una altura aproximada de 130 *cm* con respecto al piso. Sobre el escritorio se dispuso diferentes objetos que le brindaron al algoritmo la posibilidad de detectar y seguir diferentes características.

Se diseñó un recorrido de ida y vuelta, comenzando en el punto etiquetado como *start*, en el extremo izquierdo de la figura 5.2 y avanzando hasta el punto denotado como *end*, para finalmente, retornar al punto inicial.

En la figura 5.3 se observa el entorno de trabajo para el experimento. Se encuentra conformado por un escritorio y los objetos que sobre él se encuentran, cabe mencionar algunos como: teléfono, computador NUC, botella de agua, computador ASUS, teclado, patrón de tablero de ajedrez para calibración de cámaras y robot móvil (en color rojo).

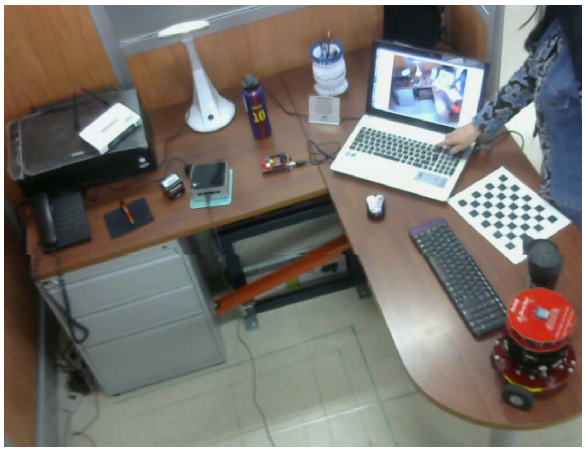


Figura 5.3 Los objetos dispuestos sobre el escritorio (entorno de trabajo) ofrecen diferentes posibilidades de detección y seguimiento de características.

La configuración experimental para un mapeo *hand-held* del entorno considerado se observa en la figura 5.4. En este caso no se requiere de la plataforma móvil puesto que la cámara se desplaza manualmente sobre la escena.

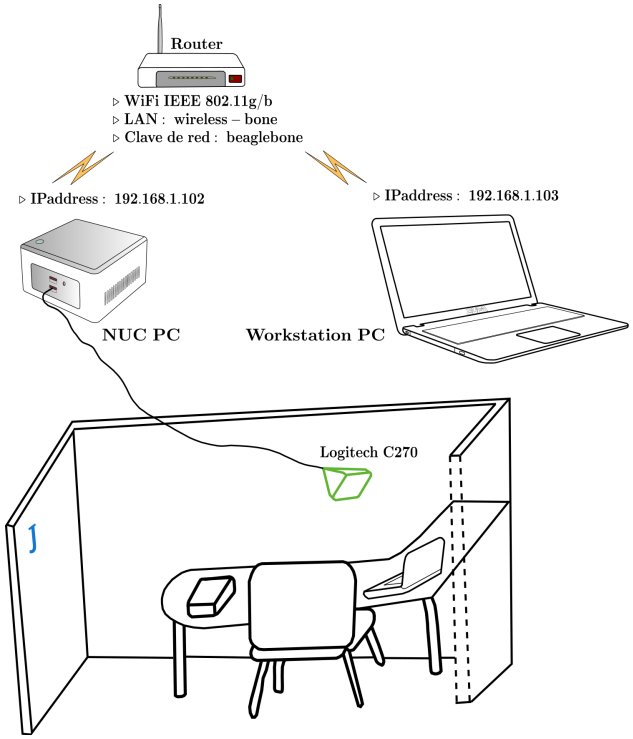


Figura 5.4 Configuración experimental para un mapeo tipo *hand-held* con el método ORB-SLAM2.

En la figura 5.5 se puede apreciar el mapa entregado por el algoritmo ORB-SLAM2 empleando su componente monocular y desarrollado en [121]. La librería puede ser compilada con o sin ROS [1], en este trabajo se usa en combinación con este *middleware*.

Para ejecutar ORB-SLAM2 con ROS se procede a abrir desde el Worksation PC tres terminales para el acceso remoto del NUC PC mediante `SSH`, de esta manera es posible supervisar el proceso desde el Worksation PC (ver figura 5.4). Posteriormente se ingresan los siguientes comandos en las terminales:

T1: `$roscore`

T2: `$rosrun usb_cam usb_cam_node usb_cam/image_raw:=/camera/ \`
`image_raw`

T3: `rosrun ORB_SLAM2 Mono \`
`/home/turtlebot/ORB_SLAM2/Vocabulary/ORBvoc.txt \`
`/home/turtlebot/ORB_SLAM2/Examples/ROS/ORB_SLAM2/Asus.yaml`

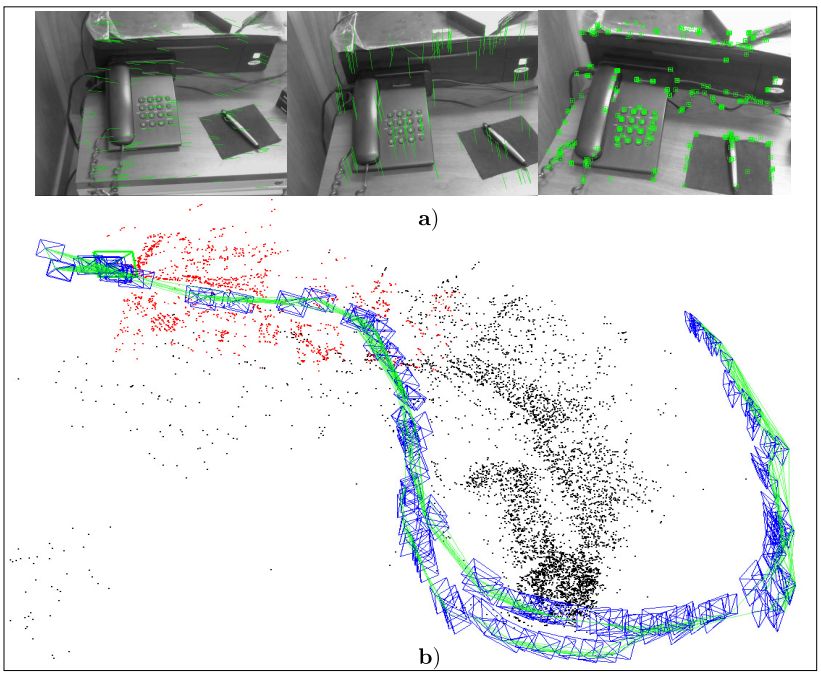


Figura 5.5 Mapa disperso elaborado con ORB-SLAM2. En **a**) se observa la detección de características previas a la inicialización del mapa (segmentos de recta en color verde) y su seguimiento después de la elaborar un primer mapa (pequeños rombos encerrados por cuadrados). En **b**) se aprecia la nube de puntos dispersa correspondiente al mapa.

[1] https://github.com/raulmur/ORB_SLAM2

Los dos primeros comandos no revisten dificultades. El primero ejecuta el *ROS Máster* y el segundo lanza el nodo `usb_cam_node`, un driver de ROS para cámaras USB, en este caso el dispositivo Logitech C270 que se muestra en la figura 5.4.

El último comando lanza el nodo `Mono` del paquete `ORB_SLAM2` con el cual se inicia el mapeo del entorno a partir de la secuencia de imágenes provista por la cámara. Se implementa así un proceso de SLAM monocular en tiempo real. El comando requiere de dos parámetros. El primero indica la localización del vocabulario de palabras visuales contenido en el archivo `ORBvoc.txt` (*bag-of-words*) y el segundo la ubicación del archivo `ASUS.yaml` que contiene los parámetros de la cámara.

El mapeo inicia extrayendo características ORB de la secuencia de imágenes, en procura de construir un mapa inicial (*map initialization*) en el cual estimar la posición de la cámara. En las dos primeras imágenes de la figura 5.5a) (esquina superior izquierda) se observa como el sistema detecta un conjunto de características relevantes del entorno, lo cual se indica mediante los segmentos de recta de color verde sobre los objetos observados. En esta etapa la GUI (Graphical User Interface) del sistema coloca el mensaje "`TRYING TO INITIALIZE`" en la ventana del flujo de imágenes.

Una vez el algoritmo consigue construir un mapa inicial, se procede a estimar la localización de la cámara en él. El sistema se encuentra ahora en modo SLAM, lo que se indica al colocar el mensaje "`SLAM MODE`" en la ventana de flujo de imágenes. Adicionalmente, la señalización de características cambia a pequeños rombos encerrados por cuadrados, como se observa en la tercera imagen de la figura 5.5a).

Cada punto p_j de la nube de características ORB almacena [116]:

- Su posición tridimensional $\mathbf{x}_{wj}$ en el sistema de coordenadas del mundo
- La dirección $\mathbf{n}_j$ de la cámara, que corresponde al vector unitario medio extraído a partir de los rayos que unen el punto con el centro óptico de las *keyframes* observadas
- Un descriptor $\mathbf{D}_i$ representativo, correspondiente al descriptor ORB cuya distancia *hamming* es la mínima entre todas las *keyframes* donde el punto es observado

- Las distancias máxima y mínima (d_{max}, d_{min}) desde las cuales el punto puede ser observado, de acuerdo con los límites de invarianza de escala de las características ORB

Cada *keyframe* almacena [116]:

- La posición $\mathbf{T}_{iw}$, la cual es una transformación de cuerpo rígido de los puntos del mundo al sistema de coordenadas de la cámara
- Los parámetros intrínsecos de la cámara, que incluyen la distancia focal y las coordenadas del punto principal
- Todas las características ORB extraídas de la imagen, asociadas o no a un punto del mapa y cuyas coordenadas no están distorsionadas o, en caso de estarlo, se dispone de un modelo de distorsión

La figura 5.6 resume lo anteriormente expuesto. Las coordenadas $\mathbf{x}_{wj}$ de los puntos rojos y negros que hacen parte de la nube. Los puntos rojos corresponden al mapa local actual y los negros al mapa global. Las *keyframes* aparecen en azul y las trayectorias estimadas $\mathbf{T}_{iw}$ de la cámara en verde.

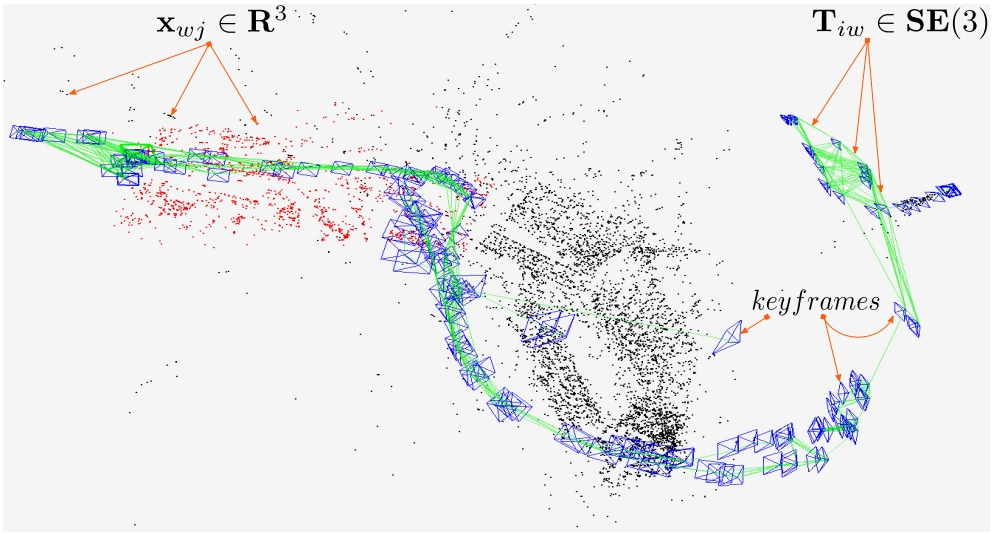


Figura 5.6 Interpretación de un mapa disperso elaborado con ORB-SLAM2.

ORBSLAM2 es un método GraphSLAM (sección 2.3.4) en el que se lleva a cabo una detección de bucles cerrados con base en la optimización del grafo de posiciones

(figura 2.4). El problema de optimización de la estructura 3D, la posición de la cámara y sus parámetros intrínsecos se soluciona empleando BA (sección 4.1.3). En el grafo de posición las *keyframes* corresponden a los nodos y las trayectorias de la cámara a los arcos (figura 5.6).

En la figura 5.7 se establece la correspondencia de algunos objetos en la escena con la nube de puntos dispersa o mapa.

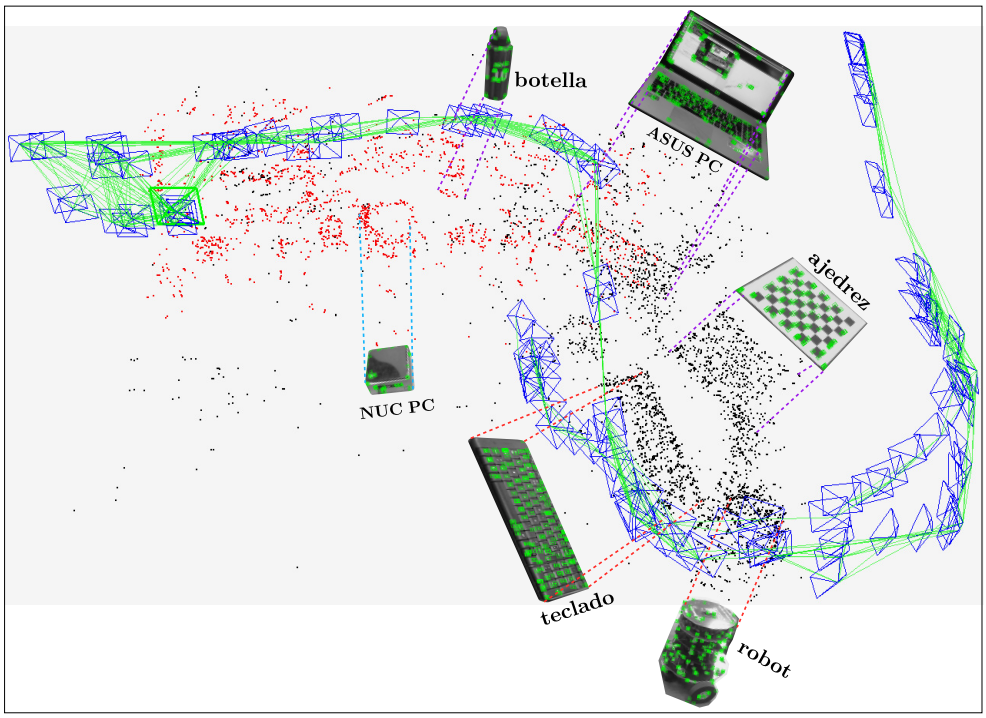


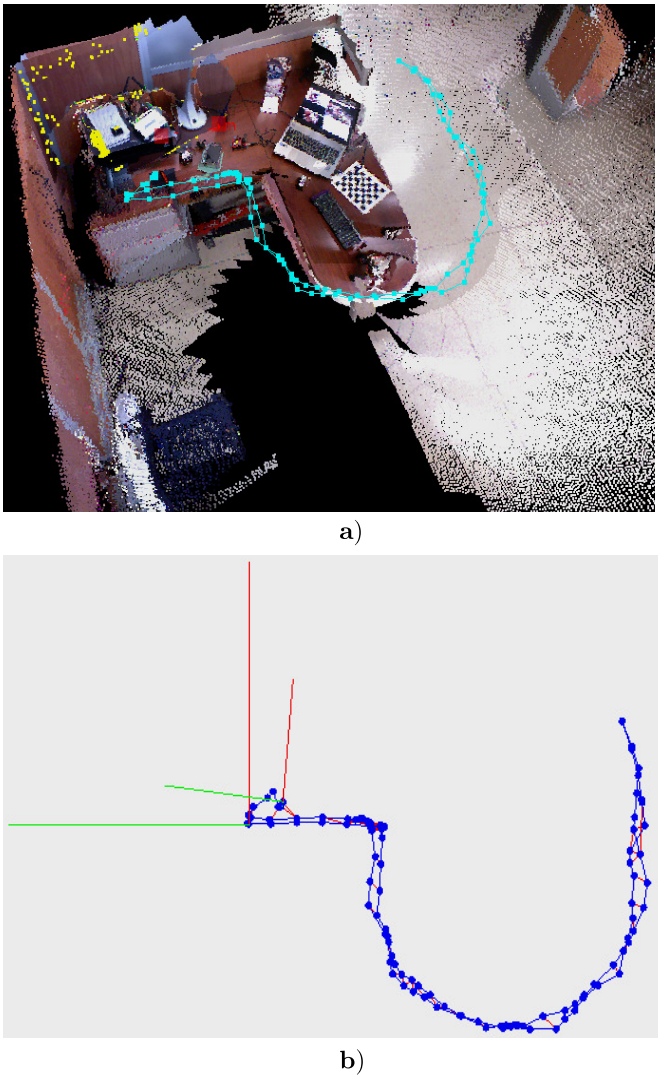
Figura 5.7 Identificación de objetos en la nube de puntos ORB.

5.1.2. Experimento *hand-held* RTAB-Map

El mismo entorno de escritorio en "*L*" (figura 5.2) fue considerado para realizar experimentos de mapeo *hand-held* con RTAB-Map. En esta caso se empleó un sensor *Kinect*, que entrega tanto imágenes de profundidad como RGB.

En la figura 5.8 a) se observa uno de los mapas 3D elaborados, se destaca la nube de puntos que otorga una apariencia realista al modelo. El grafo de posiciones (*pose graph*) se muestra en la figura 5.8 b) junto a las coordenadas inicial y final del sistema vinculado al sensor.

Los nodos del grafo almacenan la odometría de la cámara, junto con información visual de imágenes de profundidad, imágenes RGB y las *visual words* (características SURF cuantizadas en un diccionario visual incremental), usadas por el algoritmo de detección de bucles cerrados [132]. El algoritmo permite establecer coincidencias entre la ubicación actual de la cámara y ubicaciones previas de sitios ya visitados empleando el método *bag-of-words* [133], discutido anteriormente en la sección 3.1.7.



a)

b)

Figura 5.8 Mapa elaborado con RTAB-Map. En **a**) la nube de puntos densa otorga una apariencia realista al modelo. El grafo de posiciones en **b**) muestra los bordes o trayectoria estimada de la cámara (*segmentos de recta en azul*), los nodos (*puntos azules*) y la detección de bucles cerrados (*segmentos de recta en rojo*).

En la figura 5.9 se aprecia la odometría visual y la detección de bucles cerrados que implementa el método RTAB-Map.

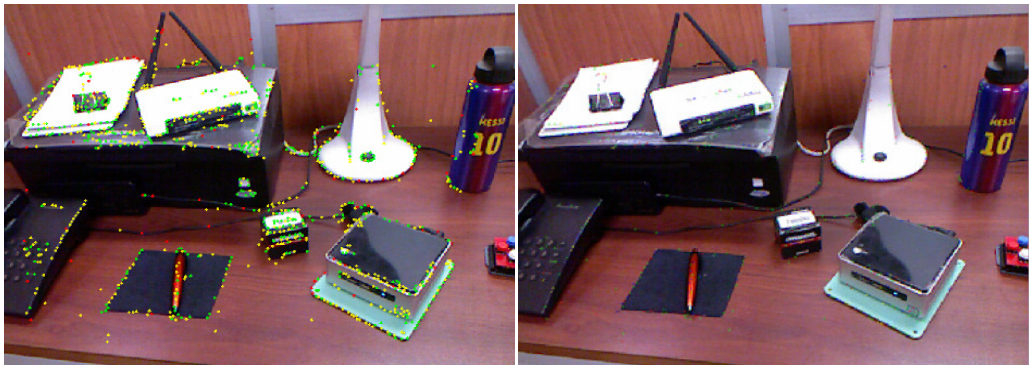


Figura 5.9 Odometría visual (*izq.*) y detección de bucles cerrados (*der.*) llevados a cabo por el sistema RTAB-Map.

5.1.3. Experimento de mapeo disperso con el sistema RobSLAM

En la figura 5.10 se aprecia el entorno de trabajo para los experimentos de mapeo vSLAM llevados a cabo con el sistema RobSLAM. Se consideraron los cubículos H y J del recinto de la figura 3.6.

Se indica en verde el recorrido diseñado, el cual consta de tres tramos. El primero de 0,86 *m* con orientación de 30° con respecto a la paralela al borde que delimita los módulos. El segundo, paralelo a ésta y a una distancia de 1,02 *m* tiene una longitud de 2,55 *m*. Por último, un tercer tramo que es simétrico con el primero y de la misma longitud.

Los sistemas monoculares estiman la profundidad al triangular puntos claves correspondientes en imágenes consecutivas, por lo que son requeridos desplazamientos laterales de la cámara que generen líneas base de longitud suficiente para el cálculo de profundidad (ver figura 4.5). Por tal motivo, para desplazamientos únicamente hacia adelante o hacia atrás tienden a fallar [134]. Con el fin de permitir desplazamientos laterales, la cámara fue localizada en el costado derecho del robot móvil, tal y como se muestra en la figura 5.10.

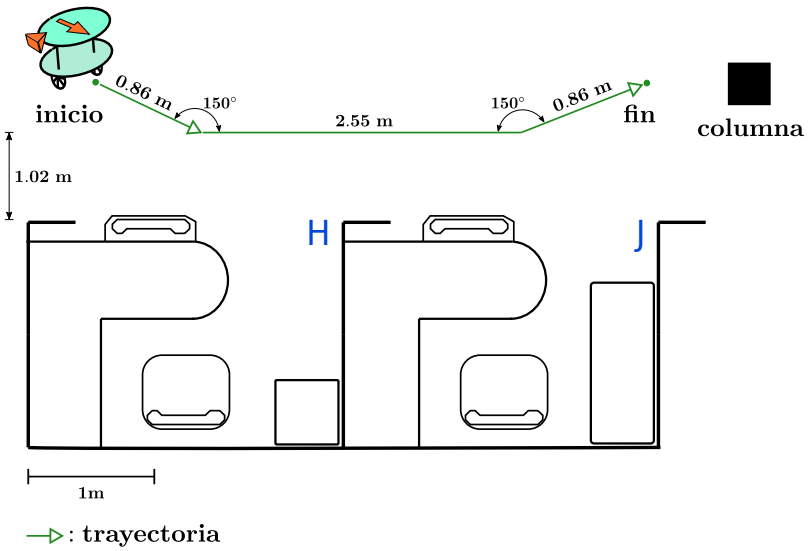


Figura 5.10 Trayectoria para mapeo con ORB-SLAM2. La longitud total del recorrido es de 4.27m.

El robot fue teleoperado a lo largo de la trayectoria mostrada en la figura 5.10 haciendo uso del hardware y software diseñado en la sección 3.1.5 (Módulo de teleoperación).

En la figura 5.11 se muestran algunas etapas del proceso de construcción del mapa con ORB-SLAM2. En la figura 5.11 **a**) se observa el inicio de la construcción del mapa (derecha), siendo necesario incorporar una impresión del tipo *patrón de ajedrez* a la primera división observada por la cámara (izquierda).

De hecho, ante la falta de textura de las divisiones encontradas en el entorno, fue necesario localizar en varias de ellas este tipo de patrones, so pena de que el algoritmo no identificase un número suficiente de características para llevar a cabo el seguimiento (*tracking*).

Conforme se avanza en la teleoperación del robot móvil, se observa como se va estimando simultáneamente la trayectoria de la cámara y el mapa 3D (figuras 5.11 **b**) y **c**)). A medida que se van dejando atrás algunas características, otras nuevas van siendo incorporadas para su seguimiento. Los sitios ya visitados y cuyas características no hacen parte de la escena actual, aparecen en color negro en la nube de puntos, y pasan a ser parte del mapa global. Los puntos rojos en la nube, hacen parte de las características del mapa local que se encuentra en proceso de elaboración.

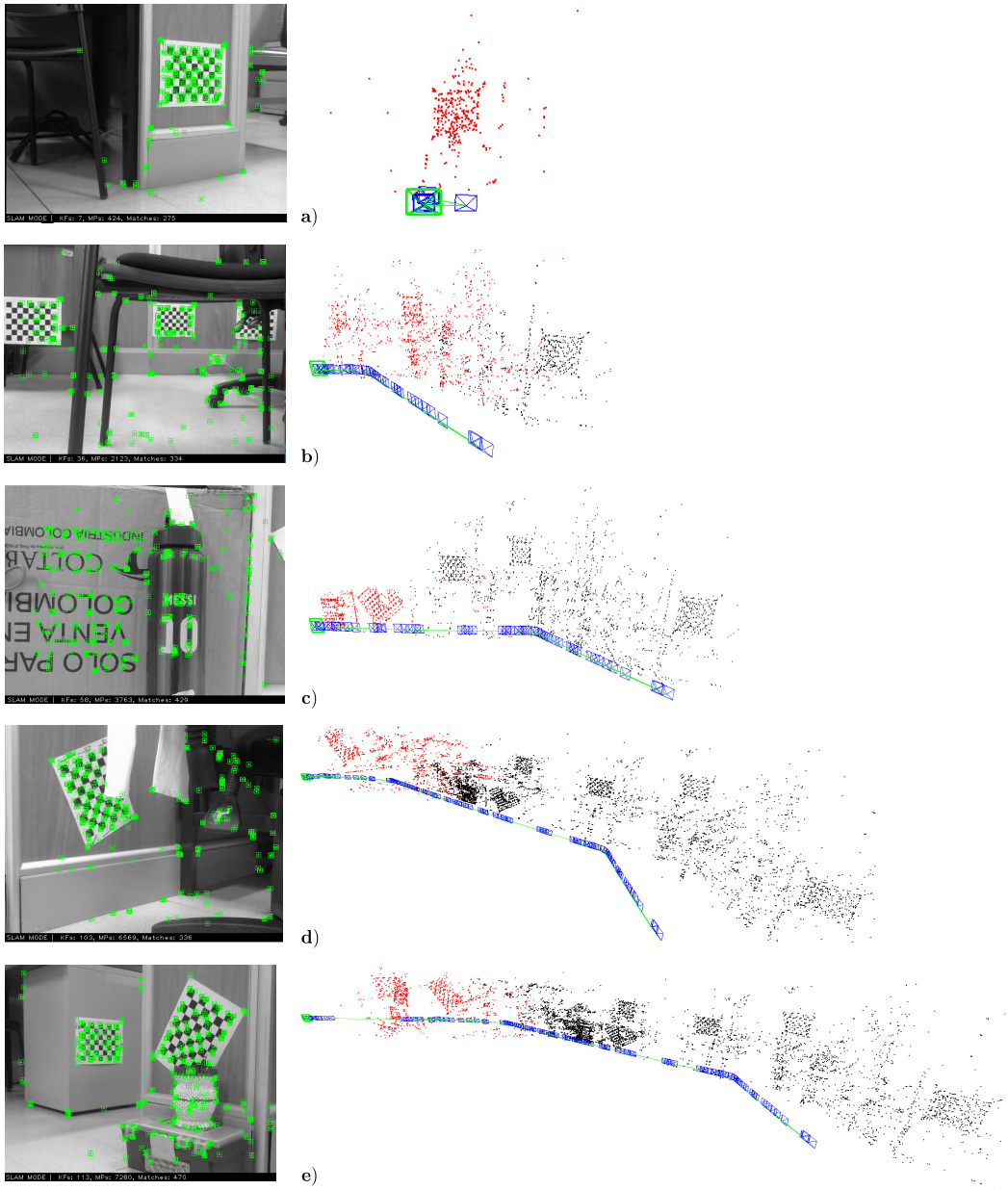


Figura 5.11 Diferentes etapas en el proceso de construcción de un mapa 3D con el sistema RobSLAM y el algoritmo monocular en tiempo real ORB-SLAM2. Nótese la incorporación de patrones tipo *tablero de ajedrez* para agregar textura a las divisiones modulares del recinto.

Las figuras 5.11 **d**) y **e**)) muestran el proceso de mapeo y localización casi al final del recorrido. Se observa como la estimación de la trayectoria de la cámara se ajusta al recorrido diseñado y como la reconstrucción 3D refleja, en términos generales, la estructura geométrica del entorno.

El mapa final se aprecia en la figura 5.12, donde se muestra además un patrón de tablero de ajedrez adherido a un escritorio para agregar textura a la escena.

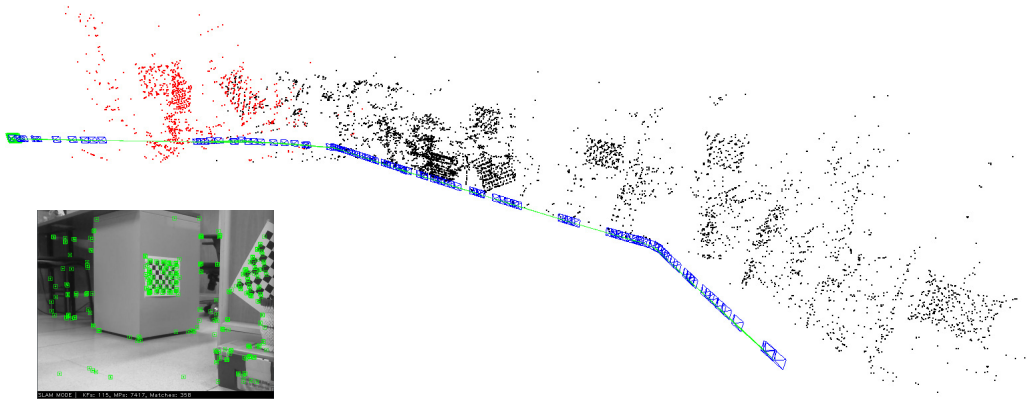


Figura 5.12 Mapa final producido por ORB-SLAM2 al final del recorrido .

Se puede observar en la figura 5.13, que si bien la trayectoria estimada de la cámara se aproxima a la odometría mecánica del robot, tal estimación no corresponde a la escala absoluta del recorrido.

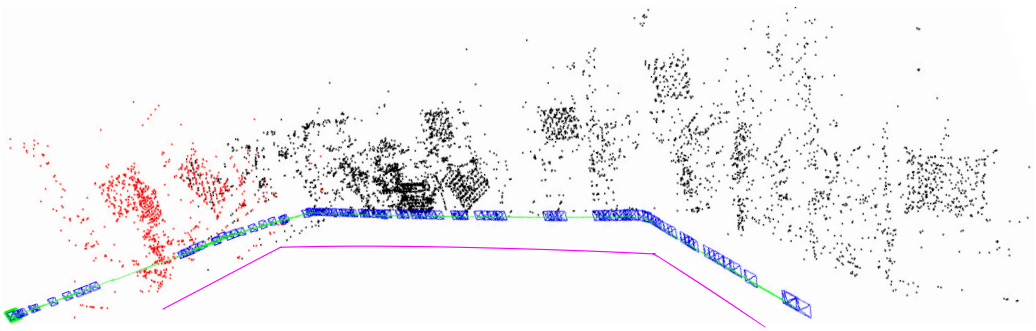


Figura 5.13 Mapa final obtenido con ORB-SLAM2.

5.1.4. Respuesta a Preguntas de Investigación

A continuación se da respuesta a la última pregunta de investigación formulada en la sección 1.5 de este trabajo:

- ¿Qué algoritmos emplear en la construcción de representaciones tridimensionales del entorno?

Respuesta: La comparación de algoritmos vSLAM de última generación conducida en la sección 5.1, permitió establecer que dos algoritmos convenientes para resolver los problema de la construcción de representaciones tridimensionales del entorno y la estimación de la trayectoria del sensor son ORB-SLAM2 y RTAB-Map.

Particularmente ORB-SLAM2 resultó ser el más adecuado, por tratarse de un algoritmo capaz resolver el SLAM en tiempo real empleando una cámara USB como único sensor, el cual ofrece las ventajas de ser un dispositivo de bajo costo, liviano, de dimensiones reducidas y bajo consumo de energía.

5.2. Conclusiones del Capítulo

En este capítulo se consideraron algunos de los algoritmos que constituyen el estado del arte actual del SLAM visual. La revisión y comparación sistemática de varios de los más populares algoritmos de código abierto disponibles para resolver el problema del SLAM, condujo a la selección de los métodos ORB-SLAM2 y RTAB-Map.

Ambos algoritmos se emplearon en experimentos de mapeo *hand-held*, en los que el sensor es desplazado manualmente por la escena que se desea reconstruir, en este caso un entorno interior de oficina con un escritorio en "L", el cual proporcionó un número suficiente y variado de características para mantener la odometría visual y poder elaborar su modelo tridimensional.

Un experimento adicional empleando el sistema RobSLAM y el algoritmo ORB-SLAM2 fue conducido en un recinto de oficinas. Se efectuó una teleoperación del robot a lo largo de dos módulos contiguos con presencia de artefactos como sillas, mesas y divisiones con enchape de madera. En este caso, se evidenció una falta de textura, principalmente en las divisiones, que provocó una frecuente pérdida de la odometría visual y un fallo en el experimento, aún al hacer retroceder el robot para tratar de recuperar la odometría.

Una solución al problema de la falta de textura, consistió en adherir a las divisiones un patrón de tablero de ajedrez, como los usados en la calibración de cámaras. Si bien se mejoró el seguimiento de características y la odometría visual, se detectó un segundo problema, consistente en que los tableros observados a mitad del recorrido

fueron considerados como sitios previamente visitados. En efecto, tableros observados al inicio del recorrido y tableros observados posteriormente fueron considerados bucles cerrados por el proceso paralelo de detección de bucles de ORB-SLAM2. Este problema llevó a obtener *falsas* estimas de localización de la cámara y mapas que no reflejaban las propiedades geométricas del entorno.

El problema de la falta de variedad en las características incorporadas al entorno fue resuelto girando algunos de los tableros y agregando nuevos objetos a la escena. Con ello, finalmente se pudieron obtener mapas tridimensionales aceptables del entorno explorado.

Tanto ORB-SLAM2 como RTAB-Map son algoritmos GraphSLAM (ver sección 2.3.4, *Algoritmos de solución del SLAM*), de los cuales se obtienen dos cosas: 1) una nube de puntos, y 2) un grafo de posiciones. Pero, mientras que ORB-SLAM2 elabora mapas dispersos, RTAB-Map entrega una nube densa de puntos, lo que implica la necesidad de un hardware de mayores prestaciones.

Las principales ventajas de ORB-SLAM2 son: 1) el mapa se puede inicializar solamente con el primer par de imágenes, 2) es globalmente consistente, sin embargo el cierre de bucles puede resultar en saltos de marcada extensión en la estimación de la posición. Como desventajas se pueden mencionar: 1) se requiere de un vocabulario *bag-of-words* entrenado, 2) el modo de localización requiere de un mapa integrado y depende considerablemente de la relocalización y 3) la escala absoluta es observable para las componentes estéreo y RGB-D del algoritmo pero se desconoce para la monocular.

Capítulo 6

Conclusiones Generales

La creciente demanda mundial de robots de servicios, logísticos, de asistencia personal y de compañía, así como la necesidad de incrementar sus niveles de autonomía, han hecho del SLAM una pieza fundamental de la robótica. Además sus métodos se han extendido a la conducción autónoma de todo tipo de vehículos: terrestres, aéreos y acuáticos.

Esta tesis se enfocó en el estudio del problema del mapeo y localización simultanea en entornos desconocidos o SLAM. Desde su formulación y métodos de solución, hasta la evaluación de su desempeño. Para tal fin se propuso un método que cuantifica la calidad de los mapas elaborados con base en dos métricas, MeC y MoC. La primera basada en la proporción de celdas libres y ocupadas y la segunda en le número de esquinas detectadas en el mapa.

Así mismo, un sistema robótico (RobSLAM) fue diseñado y construido para la validación experimental conducida en este trabajo. El sistema es lo suficientemente versátil para apoyar procesos de investigación y desarrollo de aplicaciones robóticas, ofrece buena autonomía energética y capacidades extendidas de procesamiento y exterocepción. El sistema aprovecha las características de abstracción de hardware y software que ofrece ROS, uno de los *middlewares* de más rápido crecimiento y penetración en la comunidad robótica.

Un análisis de los aspectos teóricos más relevantes de los problemas SfM (*Structure from Motion*) y SLAM visual fue conducido. Posteriormente se llevó a cabo un estudio comparativo de los algoritmos más representativos del estado del arte actual para la solución del SLAM visual. Como resultado del estudio, los algoritmos RTAB-

Map y ORB-SLAM2 fueron seleccionados para conducir experimentos SLAM en un entorno interior de oficinas. Mientras que RTAB-Map permite construir mapas densos de nube de puntos, ORB-SLAM2 entrega representaciones dispersas del entorno. Este último algoritmo se probó con el sistema RobSLAM para dar solución al problema del SLAM monocular en tiempo real. Las capacidades de procesamiento y sensórica incorporadas al sistema, permitieron resolver el problema en los entornos experimentales considerados.

Bibliografía

[1] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robot. Automat. Mag*, 13(2):99–110, 2006.

[2] G. Carrera, A. Angeli, and A.J. Davison. Lightweight slam and navigation with a multi-camera rig. In *5th European Conference on Mobile Robots (ECMR-2011)*, pages 77–82, 2011.

[3] R.A. Newcombe and A.J. Davison. Live dense reconstruction with a single moving camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1498–1505, 2010.

[4] J. Civera, D.R. Bueno, A.J. Davison, and J.M.M. Montiel. Camera self-calibration for sequential bayesian structure from motion. In *Proceedings of International Conference on Robotics and Automation*, pages 403–408, 2009.

[5] J. Neira, A.J. Davison, and J.J. Leonard. Guest editorial-special issue on visual slam. *IEEE Trans. Robot.*, 24(5):929–931, 2008.

[6] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[7] C. Hertzberg, R. Wagner, O. Birbach, T. Hammer, and U Frese. Experiences in building a visual slam system from open source components. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2644–2651, 2011.

[8] A.J. Davison. Real-time simultaneous localization and mapping with a single camera. In *Proceedings of the IEEE International Conference on Computer Vision ICCV*, pages 1403–1410, 2003.

[9] P. Tobias. *Towards Dense Visual SLAM*. PhD thesis, Universidad Técnica de Dresden, 2011.

[10] S. Lovegrove. *Parametric Dense Visual SLAM*. PhD thesis, Department of Computing, Imperial College London, 2011.

[11] K. Konolige and M. Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24:1066–1077, 2008.

[12] J. Civera, O.G. Grasa, A.J. Davison, and J.M.M. Montiel. 1-point ransac for ekf-based structure from motion: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[13] J. Castellanos, J. Neira, and J. Tardós. Limits to the consistency of ekf-based slam. In *IFAC Symposium on Intelligent Autonomous Vehicles*, pages 1–6, 2004.

[14] H. Huang. *Bearing-only SLAM, A Vision-Based Navigation System for Autonomous Robots*. PhD thesis, Faculty of Information Technology, Queensland University of Technology, 2008.

[15] M. Chli and A.J. Davison. Active matching for visual tracking. *Robotics and Autonomous Systems*, 57(12):1173–1187, 2009.

[16] R.A. Romero, E. Prestes, F. Osório, and D. Wolf, editors. *Robótica Móvel*. LTC, 2014.

[17] A. Corominas Murtra, E. Trulls, O. Sandoval-Torres, J. Pérez-Ibarz, D. Vasquez, J.M. Mirats-Tur, M. Ferrer, and A. Sanfeliu. Autonomous navigation for urban service mobile robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4141–4146, 2010.

[18] A. Kelly. *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge University Press, 2014.

[19] F. Martín, V. Matellán, P. Barrera, and J.M. Cañas. Localization of legged robots combining a fuzzy-markov method and a population of extended kalman filters. *Robot. Auton. Syst.*, 55(12):870–880, 2007.

[20] R. Negenborn. Robot localization and kalman filters–on finding your position in a noisy world. Master's thesis, Utrecht University, Utrecht, 2003.

[21] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[22] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, 1985.

[23] R.C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[24] R. Smith, M. Shelf, and P. Cheeseman. Stochastic map for uncertain spatial relationships. In *4th. International Symposium on Robotics Research*, pages 1–10, 1987.

[25] J. Crowley. World modeling and position estimation for a mobile robot using ultrasonics ranging. In *1989 International Conference on Robotics and Automation, Robotics and Automation*, pages 1574–1579, 1989.

[26] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 138–143, 1985.

[27] J.J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS 91*, pages 1442–1447, 1991.

[28] M.W.M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[29] N.A. Othman and H. Ahmad. Examining the eigenvalues effect to the computational cost in mobile robot simultaneous localization and mapping. *Computers and Electrical Engineering*, 56:659–673, 2016.

[30] O. El Hamzaoui. *Localisation et Cartographie Simultanées pour un robot mobile équipé d'un laser à balayage: CoreSLAM*. PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 2012.

[31] H. Durrant-Whyte, S. Majumder, S. Thrun, M. de Battista, and S. Scheding. A bayesian algorithm for simultaneous localisation and map building. In *The 10th International Symposium on Robotics Research, ISSR'01*, volume 6, pages 49–66, 2001.

[32] A. Ajay and D. Venkataraman. A survey on sensing methods and feature extraction algorithms for slam problem. *International Journal of Computer Science, Engineering and Applications*, 3(1):59–63, 2013.

[33] T.J. Chong, X.J. Tang, C.H. Leng, M. Yogeswaran, O.E. Ng, and Y.Z. Chong. Sensor technologies and simultaneous localization and mapping (slam). In *IEEE International Symposium on Robotics and Intelligent Sensors, IEEE IRIS 2015*, volume 76, pages 174–179, 2015.

[34] S. Jung, J. Kim, and S. Kim. Simultaneous localization and mapping of a wheel-based autonomous vehicle with ultrasonic sensors. *Artificial Life and Robotics*, 4(2):186–190, 2009.

[35] A. Burguera, Y. González, and O. Gabriel. Sonar sensor models and their application to mobile robot localization. *Sensors*, 9(12):10217–10243, 2009.

[36] C. Doer, G. Scholz, and G. F. Trommer. Indoor laser-based slam for micro aerial vehicles. *Gyroscopy and Navigation*, 8(3):181–186, 2017.

[37] W. Lin, J. Hu, H. Xu, C. Ye, X. Ye, and Z. Li. Graph-based slam in indoor environment using corner feature from laser sensor. *2017 32nd Youth Academic Annual Conference of Chinese Association of Automation*, pages 1211–1216, 2017.

[38] J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942, 2015.

[39] I. Rekleitis. A particle filter tutorial for mobile robot localization, technical report tr-cim-04-02. Technical report, Centre for Intelligent Machines, McGill University, Montreal, Québec, Canada, 2004.

[40] Z. Kong and Q. Lu. A brief review of simultaneous localization and mapping. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 5517–5522, 2017.

[41] Manigandan Nagarajan Santhanakrishnan, John Bosco Balaguru Rayappan, and Ramkumar Kannan. Implementation of extended kalman filter-based simultaneous localization and mapping: a point feature approach. *Sadhana*, 42(9):1495–1504, 2017.

[42] F. Servant. *Localisation et cartographie simultanées en vision monoculaire et en temps réel basé sur les structures planes*. PhD thesis, Université Rennes 1, 2009.

[43] L.M. Paz. *Divide and Conquer: ekf slam in O(n)*. PhD thesis, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 2008.

[44] P. Pinies and J.D. Tardós. Large-scale slam building conditionally independent local maps: Application to monocular vision. *IEEE TRANSACTIONS ON ROBOTICS*, 24(5):1094–1106, 2008.

[45] J. Aulinas, X. Lladó, J. Salvi, and S.Y. Petillot. Slam base selective submap joining for the victoria park dataset. In *7th IFAC Symposium on Intelligent Autonomous Vehicles (IFAC-IAV)*, pages 1–6, 2010.

[46] J. Nordh. pyparticleest: A python framework for particle-based estimation methods. *Journal of Statistical Software*, 78(1):1–25, 2017.

[47] B.G. Sileshi. *Algorithmic and architectural optimization techniques in particle filtering for FPGA-Based navigation applications*. PhD thesis, Departamento de Microelectrónica y Sistemas Electrónicos, Universidad Autónoma de Barcelona, 2016.

[48] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fast-slam: A factored solution to the simultaneous localization and mapping problem. In *Proc. AAAI Nat. Conf. Artif. Intell.*, pages 593–598, 2002.

[49] R. Havangi, M. Nekoui, and M. Teshnehlab. Improved fastslam framework using soft computing. *Turkish Journal of Electrical Engineering and Computer Sciences*, 20(1):25–46, 2012.

[50] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. 18th Int. Joint Conf. Art. Intell. IJCAI 2003*, pages 1151–1156, 2003.

[51] T. Li, S. Sun, T.P. Sattar, and J.M. Corchado. Fight sample degeneracy and impoverishment in particle filters: A review of intelligent approaches. *Expert Systems with applications*, 41:3944–3954, 2014.

[52] M. Ades and P.J. Van Leeuwen. An exploration of the equivalent weights particle filter. *Quarterly Journal of the Royal Meteorological Society*, 139:820–840, 2013.

[53] X. Wang, T. Li, S. Sun, and J.M. Corchado. A survey of recent advances in particle filters and remaining challenges for multitarget tracking. *Sensors*, 17(12):1–21, 2017.

[54] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[55] M. Zhang, Ling. Pei, and X. Deng. Graphslam-based crowdsourcing framework for indoor wi-fi fingerprinting. In *Proc. Fourth Int. Conf. on Ubiquitous Positioning, Indoor Navigation and Location Based Services IEEE UPINLBS-2016*, pages 61–66, 2016.

[56] J. Kim, J. Cheng, and H. Shim. Efficient graph-slam optimization using unit dual-quaternions. In *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 34–39, 2015.

[57] A. Annaiyan, M.A. Olivares-Mendez, and H. Voos. Real-time graph-based slam in unknown environments using a small uav. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1118–1123, 2017.

[58] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Trans. on Intel. Vehicles*, 2(3):194–220, 2017.

[59] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, May 2011*, pages 3607–3613, 2011.

[60] J. Vallvé, J. Solà, and J. Andrade-Cetto. Graph slam sparsification with populated topologies using factor descent optimization. *IEEE Robotics and Automation Letters*, 3(2):1322–1329, 2018.

[61] R. Appel, H. Folmer, J. Kuper, R. Wester, and J. Broenink. Design-time improvement using a functional approach to specify graphslam with deterministic performance on an fpga. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 797–803, 2017.

[62] V. Scilimati, A. Petitti, P. Boccadoro, R. Colella, D. Di Paola, A. Milella, and L.A. Grieco. Industrial internet of things at work: a case study analysis in robotic-aided environmental monitoring. *IET Wireless Sensor Systems*, 7(5):155–162, 2017.

[63] M. Shneier and R. Bostelman. Literature review of mobile robots for manufacturing. Technical report, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, USA, 2015.

[64] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. Middleware for robotics: A survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742, 2008.

[65] A. Staranowicz and G.L. Mariottini. A survey and comparison of commercial and open-source robotic simulator software. In *Proceedings of the 4th International Conference on Pervasive Technologies Related to Assistive Environments*, pages 56:1–56:8, 2011.

[66] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache. Simulation environment for mobile robots testing using ros and gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101, 2016.

[67] R. Halder, J. Proença, N. Macedo, and A. Santos. Formal verification of ros-based robotic applications using timed-automata. In *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*, pages 44–50, 2017.

[68] A. Koubâa, M.F. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud. Turtlebot at office: A service-oriented software architecture for personal assistant robots using ros. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 270–276, 2016.

[69] R. Walenta, T. Schellekens, A. Ferrein, and S. Schiffer. A decentralised system approach for controlling agvs with ros. In *2017 IEEE AFRICON*, pages 1436–1441, 2017.

[70] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, 2004.

[71] I. Afanasyev, A. Sagitov, and E. Magid. Ros-based slam for a gazebo-simulated mobile robot in image-based 3d model of indoor environment. In *Lecture notes in computer science*, pages 273–283, 2015.

[72] Y. Quiñonez, I. Tostado, and C. Burgueño. Application of evolutionary techniques and computer vision for the autonomous robots navigation using a turtlebot 2. *RISTI - Revista Ibérica de Sistemas y Tecnologías de la Información*, (E3):93–105, 2015.

[73] E. Eaton, C. Mucchiani, M. Mohan, D. Isele, J.M. Luna, and C. Clingerman. Design of a low-cost platform for autonomous mobile service robots. In *Workshop on Autonomous Mobile Service Robots, IJCAI (2016)*, pages 1–7, 2016.

[74] F.M. Noori, D. Portugal, R.P. Rocha, and M.S. Couceiro. On 3d simulators for multi-robot systems in ros: Morse or gazebo? In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 19–24, 2017.

[75] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *Open-source software workshop of the Int. Conf. on Robotics and Automation*, pages 1–6, 2009.

[76] G.A. Acosta, J.A. Saldarriaga, and J.A. Jiménez. A performance evaluation approach for embedded controllers of mobile robots. In *Proceedings of the XVII Latin American Conference in Atomatic Control, CLCA2016*, pages 207–212, 2016.

[77] R. Dhaouadi and A.A. Hatab. Dynamic modelling of differential-drive mobile robots using lagrange and newton-euler methodologies: A unified framework. *Advances in Robotics and Automation*, 2(2):1–7, 2013.

[78] A. Barrera. *Advances in Robot Navigation*. InTech, 2011.

[79] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:34–46, 2007.

[80] D. Turnage. Simulation results for localization and mapping algorithms. In *2016 Winter Simulation Conference (WSC)*, pages 3040–3051, 2016.

[81] N. Altuntaş, E. Uslu, F. Çakmak, M.F. Amasyali, and S. Yavuz. Comparison of 3-dimensional slam systems: Rtab-map vs. kintinuous. In *2nd International Conference on Computer Science and Engineering, UBMK 2017*, pages 99–103, 2017.

[82] M. Labbé and F. Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, 2014.

[83] I.Z. Ibragimov and I.M. Afanasyev. Comparison of ros-based visual slam methods in homogeneous indoor environment. In *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–6, 2017.

[84] J.M. Santos, D. Portugal, and R.P. Rocha. An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, pages 1–6, 2013.

[85] O. Wulf, A. Nuchter, J. Hertzberg, and B. Wagner. Ground truth evaluation of large urban 6d slam. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 650–657, 2007.

[86] A. Huletski, D. Kartashov, and K. Krinkin. Evaluation of the modern visual slam methods. In *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, pages 19–25, 2015.

[87] A. Huletski, D. Kartashov, and K. Krinkin. Tinyslam improvements for indoor navigation. In *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 493–498, 2016.

[88] R. Jaulmes, E. Molin, and J. Obriet-Leclef. Towards a quantitative evaluation of simultaneous localization and mapping methods. In *4th National Conference of Control Architecture of Robots, Tolouse-France*, pages 1–10, 2009.

[89] B.M.F. Da Silva, R.S. Xavier, T.P. do Nascimento, and L.M.G. Gonsalves. Experimental evaluation of ros compatible slam algorithms for rgb-d sensors. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pages 1–6, 2017.

[90] B. Vincke, A. Elouardi, and A. Lambert. Design and evaluation of an embedded system based slam applications. In *2010 IEEE/SICE International Symposium on System Integration*, pages 224–229, 2010.

[91] X.S. Le, L. Fabresse, N. Bouraqadi, and G. Lozenguez. Evaluation of out-of-the-box ros 2d slams for autonomous exploration of unknown indoor environments. In *11th International Conference on Intelligent Robotics and Applications, ICIRA 2018*, pages 283–296, 2018.

[92] L. Fang, A. Fisher, S. Kiss, J. Kennedy, C. Nagahawatte, R. Clothier, and J. Palmer. Comparative evaluation of time-of-flight depth-imaging sensors for mapping and slam applications. In *Proceedings of the Australian Robotics and Automation Association*, pages 1—-7, 2016.

[93] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan. 2d slam quality evaluation methods. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 120–126, 2017.

[94] T. Taketomi, H. Uchiyama, and S. Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16–26, 2017.

[95] G. Younes, D. Asmar, E. Shammas, and J. Zelek. Keyframe-based monocular slam: design, survey, and future directions. *Robotics and Autonomous Systems*, 98:67–88, 2017.

[96] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J.M. Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1):55–81, 2012.

[97] H. Strasdat, J.M.M. Montiel, and A. Davison. Real time monocular slam: Why filter? In *IEEE Int. Conf. Robotics and Automation*, pages 2657—-2664, 2010.

[98] Z. Cui. *Global Structure-from-Motion and Its Application*. PhD thesis, School of Computing Science, Faculty of Applied Sciences, Simon Fraser University, 2017.

[99] A. Terrón. *Luz impresa luz grabada. Fotografía estenopeica y planchas de fotopolímero*. PhD thesis, Facultad de Bellas Artes, Universidad de Granada, 2015.

[100] T. Moons, L. Van Gool, and M. Vergauwen. 3d reconstruction from multiple images, part 1: Principles. *Foundations and Trends in Computer Graphics and Vision*, 4(4):287–398, 2008.

[101] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[102] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An invitation to 3-D vision – From images to models*. Springer Verlag, 2003.

[103] G.L. Mariottini and D. Prattichizzo. Egt for multiple view geometry and visual servoing. *IEEE Robotics and Automation Magazine*, 12(4):26–39, 2005.

[104] P.L. Pari. *Control Visual Basado en Características de un Sistema Articulado. Estimación del Jacobiano de la Imagen Utilizando Múltiples Vistas*. PhD thesis, Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid, 2007.

[105] R. Klette. *Concise Computer Vision–An Introduction into Theory and Algorithms*. Springer, 2014.

[106] J.L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4114, 2016.

[107] Y. Furukawa and C. Hernández. Multi-view stereo: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 9(1–2):1–148, 2015.

[108] H. Zhu, Y. Nie, T. Yue, and X. Cao. The role of prior in image based 3d modeling: a survey. *Frontiers of Computer Science*, 11(2):175–191, 2017.

[109] R.A. Newcombe, S.J Lovegrove, and A.J. Davison. Dtam: dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision, ICCV'2011*, pages 2320—-2327, 2011.

[110] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao. Robust monocular slam in dynamic environments. In *2013 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2013*, pages 209–218, 2013.

[111] S.A.K. Tareen and Z. Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10, 2018.

[112] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:105–119, 2008.

[113] H. Bay, A. Ess, T. Tuytelaars, and L. van Gool. Speeded-up robust features (surf). *CVIU*, 110(3):346—359, 2014.

[114] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *9th European Conference on Computer Vision, (ECCV'06)*, pages 430–443, 2006.

[115] P. Musialski, P. Wonka, D.G. Aliaga, M. Wimmer, L. Gool, and W. Purgathofer. A survey of urban reconstruction. *Computer Graphics Forum*, pages 146–177, 2013.

[116] R. Mur-Artal, J.M.M. Montiel, and J.D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147––1163, 2015.

[117] M. Farenza, A. Fusiello, and R. Gherardi. Structure-and-motion pipeline on a hierarchical cluster tree. In *IEEE International Workshop on 3-D Digital Imaging and Modeling*, pages 1489–1496, 2009.

[118] T.A.N. Pire. *Localización y mapeo simultáneos mediante el uso de un sistema de visión estéreo*. PhD thesis, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, 2017.

[119] S. Milz, G. Arbeiter, C. Witt, B. Abdallah, and S. Yogamani. Visual slam for automated driving: Exploring the applications of deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 247–257, 2018.

[120] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, pages 225–234, 2007.

[121] R. Mur-Artal and J.D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Trans. Robot*, 33(5):1255–1262, 2017.

[122] E. Mendes, P. Koch, and S. Lacroix. Icp-based pose-graph slam. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–200, 2016.

[123] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. *Robotics : Science and Systems III*, pages 65–72, 2008.

[124] L. Carlone, R. Aragues, J.A. Castellanos, and B. Bona. A fast and accurate approximation for planar pose graph optimization. *The International Journal of Robotics Research*, pages 965—-987, 2014.

[125] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *Proc. Eur. Conf. Comput. Vis.*, pages 834—-849, 2014.

[126] E. López, S. García, R. Barea, L.M. Bergasa, E.J. Molinos, R. Arroyo, E. Romera, and S. Pardo. A multi-sensorial simultaneous localization and mapping (slam) system for low-cost micro aerial vehicles in gps-denied environments. *Sensors*, 17(4):1–27, 2017.

[127] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018.

[128] M. Labbé and F. Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, pages 1–40, 2018.

[129] S. Krig. *Computer vision metrics: survey, taxonomy, and analysis*. Apress Open, 2014.

[130] B. Ummenhofer and T. Brox. Dense 3d reconstruction with a hand-held camera. In *Lecture Notes in Computer Science*, pages 103–112, 2012.

[131] M. Senthilvel, R.K. Soman, and K. Varghese. Comparison of handheld devices for 3d reconstruction in construction. In *Proceedings of the 34th. International Symposium on Automation and Robotics in Construction, (ISARC 2017)*, pages 1–8, 2017.

[132] W.G. Aguilar, G.A. Rodríguez, L. Álvarez, S. Sandoval, F. Quisaguano, and A. Limaico. Real-time 3d modeling with a rgb-d camera and on-board processing. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 410–419, 2017.

[133] K. Liu and R. Mulky. Enabling autonomous navigation for affordable scooters. *Sensors*, 18(6):1–21, 2018.

[134] J. Frost. *Robust and Scalable Visual Simultaneous Localization and Mapping in Indoor Environments using RGBD Cameras*. PhD thesis, Department of Computer Sciences, University of Lübeck, 2017.