Diseño Distribuido de un Sistema de Mapeo y Localización basado en visión para un Robot Móvil

Tesina de Grado

Licenciatura en Ciencias de la Computación

Autor: Mauro De Croce
Legajo: D-2170/9

Director: Taihú Pire
Co-Director: Federico Bergero

3 de mayo de 2018

Índice general

I Preliminares **3**

1. Introducción **4**
1.1. Motivación 5
1.2. Objetivo 6
1.3. Trabajos Relacionados 7

2. Conceptos Previos **9**
2.1. Visión Robótica 9
2.2. SLAM 9
2.3. ROS 12
2.4. S-PTAM 13
2.4.1. Arquitectura general 14
2.4.2. Estructuras de datos 15
2.4.3. Tracker 16
2.4.4. Mapper 18

II Aportes originales **20**

3. Diseño distribuido de S-PTAM **21**
3.1. Arquitectura distribuida 21
3.1.1. Identificación de objetos del mapa 22
3.1.2. Estructura de mensajes 25
3.2. Tracker 28
3.3. Mapper 29
3.4. Comunicación entre los nodos 29

4. Experimentos **31**
4.1. Plataforma de Benchmark 31
4.2. Datasets 32
4.3. Configuraciones de DS-PTAM 33
4.4. Resultados 35

III Conclusiones y Trabajo Futuro **42**

5. Conclusiones y Trabajo Futuro **43**

Resumen

En este trabajo se presenta una arquitectura distribuida para el sistema de SLAM estéreo S-PTAM. Esta arquitectura está desarrollada sobre el framework ROS, separando las tareas de localización y mapeo en dos nodos independientes de ROS. El sistema de SLAM distribuido resulta ideal para robots móviles con bajo poder de cómputo ya que permite ejecutar el módulo de localización *onboard* y el módulo de mapeo -que tiene mayor costo computacional- en una estación base, aprovechando al máximo el procesamiento disponible *onboard*. La arquitectura propuesta fue implementada en base al código monolítico de S-PTAM original y luego validada a través de distintos experimentos sobre *datasets* públicos. Los resultados obtenidos muestran la factibilidad de la arquitectura distribuida propuesta, su correcta implementación y los beneficios de distribuir la carga computacional en varias computadoras.

Parte I

Preliminares

Capítulo 1

Introducción

La robótica es una de las áreas tecnológicas más populares en la actualidad, fundamentalmente porque sus desarrollos han tenido un impacto destacado en la vida cotidiana de las personas. Tal es así que es común pensar hoy en día a humanos y robots realizando tareas de forma cooperativa, no sólo en el sector industrial sino también en ámbitos hogareños y en terrenos hostiles o de difícil acceso para humanos. Podemos enumerar así aplicaciones de las más variadas, entre ellas se encuentran algunas complejas, como la asistencia de salud, la búsqueda y rescate de personas, la exploración espacial. También podemos mencionar aquellas relacionadas con el confort personal como los robots de limpieza, automóviles autónomos o aquellas destinadas a alivianar la carga de una tarea pesada, tal es el caso de las maquinarias agrícolas. La robótica móvil toma un papel preponderante para que estas aplicaciones puedan cumplir sus objetivos de manera satisfactoria.

Para que pueda navegar por el entorno un robot móvil, es fundamental que conozca su posición y orientación (pose) a cada momento y que cuente con una representación del entorno (mapa) que se encuentra transitando. En robótica estos problemas suelen abordarse de manera simultánea, y el problema se conoce con el nombre de SLAM (*Simultaneous Localization and Mapping*) [1, 2, 3], que en español significa Localización y Mapeo en Simultáneo. En este tipo de problema se plantea la necesidad de construir un mapa del entorno (*mapping*), a la vez que se estima la pose del robot al desplazarse dentro del mismo (*localization*), tomando las señales de sus sensores o dispositivos incorporados.

Existen distintos tipos de sensores que pueden utilizarse para resolver el problema de SLAM: cámaras monoculares y estéreo, sonares, láser, GPS y sensores RGB-D entre otros. Sin embargo, cada sensor tiene sus limitaciones, por ejemplo un sonar permite detectar si hay objetos a una cierta distancia, pero no es posible determinar de qué objeto se trata. En el caso de los sensores GPS (*Global Positioning System*) sólo funcionan en entornos

exteriores (*outdoor*), e incluso pueden fallar en lugares donde la señal no es suficientemente buena. Los sensores láser permiten estimar la profundidad del entorno con gran precisión, pero suelen ser costosos, poco portables y no cuentan con un amplio rango de alcance. Por último, los sensores RGB-D tipo *Kinect* utilizan un sensor infrarrojo para estimar la profundidad de la escena observada y por ende se ven afectados por la luz solar, lo que impide un buen funcionamiento en ambientes exteriores.

En contrapartida y a diferencia de los sensores antes mencionados, las cámaras cuentan con gran portabilidad, bajo costo, y al mismo tiempo proveen abundante información de la escena que se está observando, además pueden ser utilizadas en ambientes tanto interiores como exteriores. Todo esto hace que las cámaras sean uno de los sensores más utilizados para resolver el problema de SLAM en la actualidad. Dentro de las soluciones basadas en cámaras podemos encontrar aquellas que utilizan dispositivos monoculares o estéreo. La principal ventaja que tienen los sistemas estéreo sobre los sistemas monoculares es que permiten realizar una reconstrucción 3D a escala de la escena observada. Las cámaras monoculares si bien permiten realizar una reconstrucción 3D de la escena, no es posible determinar la escala de la misma.

Actualmente existen numerosos enfoques para resolver el problema de SLAM mediante el uso de visión (SLAM Visual) como por ejemplo: PTAM [4], ORB-SLAM2 [5], LSD-SLAM [6], ScaViSLAM [7], SVO [8] y S-LSD-SLAM [9], en particular en esta tesis se trabaja sobre el sistema de SLAM S-PTAM (*Stereo Parallel Tracking and Mapping*) presentado en [10] ya que es uno de los sistemas referentes del estado del arte y es de código abierto.

Parte de esta tesina fue publicada en el trabajo **Diseño Distribuido de un Sistema de Mapeo y Localización basado en visión para un Robot Móvil** en las IX Jornadas Argentinas de Robótica [11].

1.1. Motivación

Los sistemas SLAM tienen un alto costo computacional debido a la necesidad de procesar gran cantidad de información en tiempo real. Actualmente, los robots móviles cuentan con una capacidad de procesamiento *onboard* limitada dada por la naturaleza de los dispositivos que los conforman. Cabe aclarar que dependiendo del entorno y la tarea que deba realizar el robot, el hardware de procesamiento deberá satisfacer restricciones de peso, tamaño y consumo eléctrico. Un exceso en cualquiera de estas magnitudes podría comprometer la capacidad de navegación del robot. Por ello, resulta conveniente distribuir la carga de procesamiento apoyándose en distintos dispositivos externos.

Considerando que S-PTAM es un sistema monolítico y que se identifican varias funcionalidades que pueden dividirse en componentes independientes

interconectados, en este trabajo se plantea un rediseño que permita hacer una distribución del procesamiento en varios dispositivos aprovechando así las características individuales de cada uno y potenciando el accionar conjunto de los mismos, sin detrimento de los resultados obtenidos.

1.2. Objetivo

El objetivo de esta tesina es desarrollar una versión distribuida del sistema de SLAM Visual S-PTAM de tal manera que se permita correr el módulo de localización (*tracking*) en la unidad de procesamiento *onboard* del robot, y separadamente el módulo de mantenimiento del mapa (*mapping*) en una estación base. De esta manera, el módulo de localización dispone de manera exclusiva de la capacidad de procesamiento del robot, permitiendo procesar una mayor cantidad de imágenes provistas por la cámara, obteniendo de esta forma un sistema más preciso y robusto a movimientos bruscos del robot. Asimismo, el módulo de mapeo que demanda un gran costo computacional, al correr en una estación remota puede ejecutarse en una computadora de mayor capacidad de cómputo de la que usualmente se encuentra en un robot móvil.

S-PTAM fue rediseñado de tal manera que pueda correr la parte de localización y mapeo de manera separada en dos computadoras (*onboard* y estación base). Para esto, se planteó un diseño modular basado en eventos que utilice memoria distribuida en vez de memoria compartida.

Al trabajar con memoria compartida se cuenta con la ventaja de que los datos almacenados en la misma pueden ser accedidos por todos los hilos involucrados, lo que la convierte en una forma rápida y eficaz de compartir información entre ellos. Por otro lado, hay que llevar un estricto control de sincronización en el caso de que se requiera acceder o modificar simultáneamente un recurso en memoria compartida por parte de dos o más hilos lo cual es complejo y costoso ya que involucra bloquear los recursos compartidos. En contrapartida, si se opta por utilizar memoria distribuida, cada proceso tendrá su propia memoria local, pero para compartir información con otro proceso será necesario recurrir al mecanismos de IPC (*Interprocess Communication*) como paso de mensajes, lo que puede llegar a impactar en la performance de todo el sistema, obligando a mantener consistente la información entre cada proceso.

En este trabajo se plantea la necesidad de disponer en cada módulo (*tracker* y *mapper*) de una copia del mapa, de manera que cada uno de ellos pueda trabajar de forma independiente, y para lograr la consistencia—entre las copias del mapa—será necesario sincronizarlos ante cada cambio que se produzca, ya sea en uno u otro. Para mantener consistentes los mapas se utilizaron mecanismos basados en el paradigma que ofrece ROS en materia de mensajes. Se crearon estructuras de mensajes de manera de incluir la mayor

cantidad de información posible con el objetivo de minimizar la cantidad de mensajes, y por ende reducir el tráfico y el costo computacional que implica esta sincronización.

1.3. Trabajos Relacionados

Los problemas SLAM son un caso de estudio desde hace ya algunas décadas y muchos trabajos fueron realizados en pos de obtener soluciones basadas en sistemas distribuidos, algunos mediante la utilización de *frameworks* desarrollados para tal fin como es el caso de [12] y de [13]. El primero está orientado a disminuir la complejidad de las comunicaciones dentro de la red para facilitar las implementaciones de soluciones SLAM distribuidas utilizando ROS como plataforma subyacente. Para ello, aprovecha la abstracción de mensajes provista por ROS, como así también su sistema de estadísticas, lo que le permite generar un reporte del consumo de recursos de red utilizado por cada nodo de ejecución. Mientras que el segundo propone que los datos, como el mapa y otras variables, sean almacenados en servidores externos ubicados en la nube. Esto libera de carga computacional al robot, que sólo debe encargarse de realizar el trackeo de las imágenes, aunque requiere una conexión a *Internet* mediante *WiFi* para los dispositivos.

Actualmente, la creciente utilización de vehículos aéreos en diversas aplicaciones ha desencadenado numerosos estudios sobre este tipo de plataforma. Al igual que todos los robots móviles, los vehículos aéreos (comúnmente llamados *drones*) también requieren de soluciones SLAM para su funcionamiento autónomo, tal es el caso de [14, 15] que proponen el uso de *drones* para obtener, mediante la aplicación del paradigma de PTAM [4], un mapa de forma colaborativa. Por otro lado, teniendo en cuenta la limitación de peso que los *drones* pueden transportar en comparación con los robots terrestres, el procesamiento distribuido toma un valor fundamental ya que permite balancear tanto carga como procesamiento en dispositivos externos y de esta manera liberar al robot de muchas responsabilidades. Encontramos un ejemplo de ello en [16] donde se incorpora el uso de una estación central que sincroniza múltiples vehículos, genera un mapa por cada uno de ellos y luego realiza la unión de los fragmentos solapados obteniendo así un mapa general.

También podemos mencionar el desarrollo planteado en [17], que describe un framework en el que se incorporan técnicas de SLAM monocular basadas en PTAM, que le permiten a múltiples vehículos aéreos la posibilidad de evitar obstáculos durante el proceso de exploración del entorno. Para este desarrollo se utilizó ROS, donde cada componente como los *drivers* de los vehículos, funcionalidades de trackeo y de mantenimiento del mapa, han sido implementados como nodos independientes aprovechando la arquitectura distribuida de ROS, pudiendo así ejecutarlos en distintos procesos con el fin

de agregar escalabilidad al framework.

Por otro lado, en [18] se presenta un novedoso sistema distribuido de reconstrucción volumétrica de objetos conformado por un vehículo aéreo que envía datos de las imágenes capturadas por sus cámaras a un servidor de gran capacidad de procesamiento. Este servidor se encargará de procesar dichas imágenes para llevar a cabo la tarea de reconstrucción. Además, permite al usuario hacer uso de una *tablet* para visualizar los resultados en tiempo real, brindando también la posibilidad de corregir resultados mientras se desarrolla el proceso.

Uno de los métodos de SLAM que ha tenido mayor impacto en la robótica autónoma fue PTAM [4], a pesar de que en sus inicios haya sido concebido para resolver problemas en el ámbito de la Realidad Aumentada (*Augmented Reality*) en lugares reducidos. Este método plantea un nuevo paradigma que se aplica a problemas de SLAM Visual en el que se propone separar las tareas de localización y mapeo en hilos de ejecución independientes optimizando el rendimiento en procesadores de múltiples núcleos. En particular, esta propuesta innovadora fue utilizada en S-PTAM [10] pero abordando el problema de SLAM para entornos de grandes dimensiones. Cabe señalar que anteriormente los paradigmas utilizados para encontrar soluciones a los problemas de SLAM eran basados en filtros como por ejemplo *Extended Kalman Filter* [19] y *Particle Filter* [20].

Muchos de los desarrollos que forman parte del estado del arte en el área de sistemas SLAM Visuales distribuidos abordan el problema de la colaboración entre dispositivos para la conformación de un mapa más completo del entorno, la mayoría de ellos trabaja sobre vehículos aéreos o dispositivos móviles, como *tablets* o celulares [14, 15, 16, 17, 18].

Este trabajo, a diferencia de abordar el problema de construir un mapa más detallado del entorno, se enfoca en obtener un sistema de SLAM distribuido capaz de localizar un robot móvil en tiempo real. El sistema de SLAM propuesto se apoya tanto en el hardware de procesamiento *onboard* del robot como en otros dispositivos externos, considerando el poder de cómputo de cada uno para poder asignarle una tarea acorde a su capacidad de procesamiento.

Capítulo 2

Conceptos Previos

En este capítulo se introducen los conceptos necesarios que se utilizarán en el resto de la tesina. Primero se hace una reseña de visión robótica y sistemas SLAM. Luego una introducción al framework ROS y finalmente se describe la arquitectura del sistema S-PTAM.

2.1. Visión Robótica

La visión robótica es un campo de investigación que combina el uso de cámaras y algoritmos computacionales para permitir a los robots procesar información visual de su entorno. Es decir, que además de procesar las imágenes obtenidas a partir de los sensores del robot, extrayendo de ellas la información de interés, la visión robótica considera aspectos relacionados al robot como sus movimientos, su estabilidad y su posición para dotarlo de la capacidad de obtener una noción, lo más certera posible, del entorno que lo rodea. De esta manera, un robot móvil podría percibir su entorno, determinando por ejemplo qué objetos se encuentran enfrente de él y así planear una trayectoria que se encuentre libre de obstáculos. En particular, las cámaras estéreo imitan la funcionalidad de los ojos humanos permitiendo reconstruir la escena 3D que se está observando.

2.2. SLAM

Se denomina SLAM [1, 2, 3] (*Simultaneous Localization and Mapping*) al problema en que un robot móvil debe realizar una representación el entorno (mapa) al mismo tiempo que estima su pose (localización). SLAM es uno de los principales temas de investigación en la robótica móvil dado que permite que un robot pueda navegar en un entorno desconocido y en zonas donde no es posible contar con un sistema de posicionamiento global (GPS). El robot hace uso de diversos sensores como cámaras, lásers y sonares para

detectar marcas naturales del entorno (*landmarks*) y estimar su posición y orientación referente a estas.

Se puede ver un ejemplo en la Figura 2.1 que muestra la posición y orientación real del robot (triángulos grises), las posiciones reales de los *landmarks* (estrellas blancas) y la visualización de éstos por parte del robot a lo largo de su trayectoria (flechas rojas). Por otro lado, se puede observar también la trayectoria estimada (línea punteada) que calcula el robot a partir de las posiciones estimadas de las *landmarks* (estrellas azules) que fueran detectadas durante el recorrido.

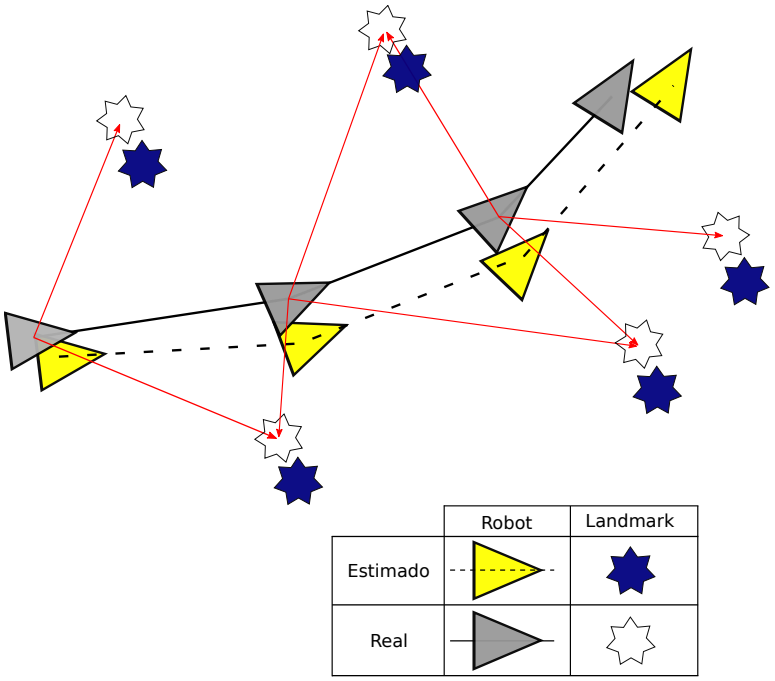


Figura 2.1: Ejemplo de estimación de pose mediante detección de *landmarks*.

Los métodos de SLAM Visual utilizan cámaras como sensores para obtener información del entorno. Existen diferentes enfoques a la hora de realizar el procesamiento de las imágenes, en particular los sistemas de SLAM que han tomado mayor relevancia son aquellos basados en el paradigma de extracción de *features*. Este paradigma es el utilizado en este trabajo y se tomará como recurso pedagógico para el entendimiento del concepto de método de SLAM Visual.

Los sistemas de SLAM Visual basados en *features* reciben las imágenes capturadas por las cámaras y en ellas se identifican marcas naturales como bordes y esquinas de los objetos (ver Figura 2.2), que son percibidos como zonas de alto gradiente presentes en la imagen. Estas marcas destacadas reciben el nombre de *features* y su extracción es un paso fundamental en el proceso, ya que de esta manera se reduce la cantidad de información que se

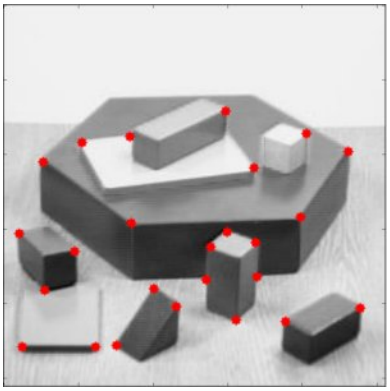


Figura 2.2: Detección de esquinas aplicado en diversos objetos

tiene de cada imagen, considerando sólo aquellos puntos que son fácilmente reconocibles de una imagen a otra.

El proceso de extracción de *features* consiste en primer lugar en la etapa de detección de esquinas o puntos salientes (llamados puntos claves o en inglés *key points*) y luego la etapa de descripción en la que se calcula un vector gradiente con la información que rodea al punto en cuestión. Este vector recibe el nombre de *descriptor* y permite identificarlo unívocamente a partir de su información característica con el objetivo de poder reconocerlo entre distintas imágenes.

Dadas dos imágenes de una misma escena es posible estimar la posición 3D de cada punto observado. En particular, los métodos de SLAM Visual basados en *features* realizan dicha reconstrucción 3D a partir de la triangulación de los *features* extraídos en cada una de las imágenes, ver Figura 2.3. Cada punto 3D reconstruido recibe el nombre de *Map Point* y el conjunto de ellos forma parte de lo que se denomina mapa (representación 3D del entorno).

Dicho mapa aumentará de tamaño a medida que se le incorporen los nuevos *Map Points* durante el resto del proceso y será utilizado para estimar la localización de robot durante la navegación del mismo. Para realizar la localización, en una primera etapa se realiza la búsqueda de correspondencias 3D-2D entre los *Map Points* del mapa y los *features* extraídos de las imágenes adquiridas. Luego estas correspondencias 3D-2D son utilizadas para estimar efectivamente la pose del robot.

Un método de SLAM Visual puede resumirse como un método iterativo e incremental en el que al recibir una imagen (o más) se realizan las siguientes tareas:

1. Se extraen los *features* de la imagen asociados a las marcas naturales del entorno (*landmarks*).

2. A partir de los *features* extraídos se generan los puntos del mapa (*Map*

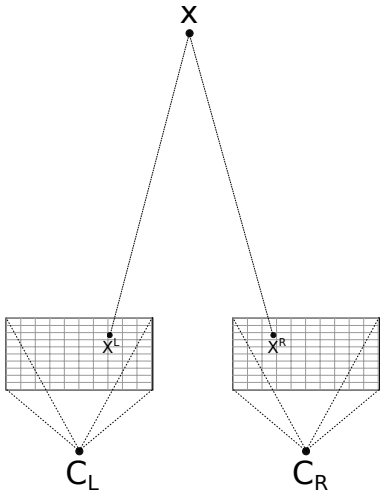


Figura 2.3: Triangulación de punto 3D. Considerando que C_L y C_R son los centros focales de cada cámara, X^L y X^R son las observaciones del punto en el espacio X sobre el plano izquierdo y derecho respectivamente.

Points).

3. Se estima la pose del robot realizando una correspondencia entre los datos sensados y la información del mapa generado.

4. Se ajustan las poses estimadas (a lo largo de la trayectoria del robot) y los *Map Points* de manera conjunta, teniendo en cuenta la observaciones de los mismos.

Finalizada esta iteración el método queda a la espera del siguiente par de imágenes.

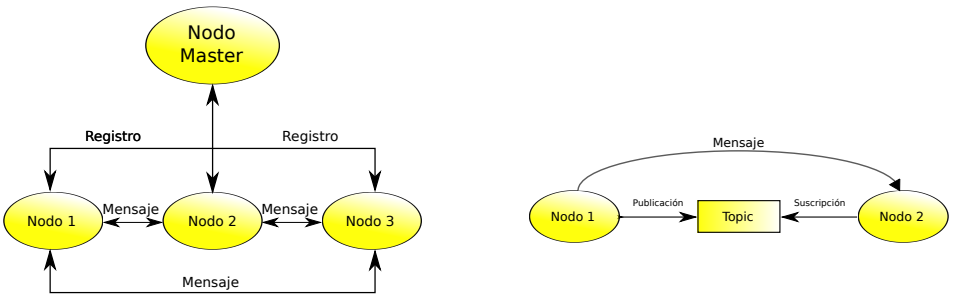
2.3. ROS

ROS [21] (*Robot Operating System*) es por excelencia la herramienta utilizada en desarrollos robóticos a nivel mundial. Fue desarrollado en 2007 por ingenieros de la Universidad de Stanford de Estados Unidos para dar soporte a sus proyectos, pero desde entonces llegó a convertirse en un estándar en el campo de la robótica. El uso de ROS en el ámbito de la investigación crece día a día, ya que cuenta con soporte para C++ y *Python* motivando a una gran comunidad de desarrolladores que continuamente enriquecen sus funcionalidades con nuevos paquetes.

ROS es en realidad un *framework* de desarrollo, pero tanto su nombre como su popularidad se deben a que provee los servicios estándar de un sistema operativo tales como abstracción de *hardware*, control de dispositivos

de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes, lo que facilita muchísimo las tareas de desarrollo. Además, ROS es *software* libre bajo términos de licencia BSD, lo que permite libertad para su uso comercial y académico.

La estructura modular de ROS permite que cada aplicación pueda ser ejecutada, de manera independiente, dentro de una unidad de ejecución llamada nodo. En particular, existe un nodo llamado *master* que funciona como nodo principal proporcionando el registro de nombres de nodos y permitiendo que los nodos sean capaces de encontrar al resto de los nodos, intercambiar mensajes o invocar servicios. La Figura 2.4 muestra el esquema de su arquitectura. ROS provee un sistema de canales de comunicación entre los nodos llamados *topics*. Los nodos pueden enviar mensajes publicando en *topics* o recibir mensajes mediante la suscripción a los mismos. Cabe aclarar que el sistema de *topics* es asíncrono, es decir que un nodo que publica un mensaje no puede saber si otro lo recibió. Estas características tendrán fundamental importancia en la nueva arquitectura propuesta.



(a) (b)

Figura 2.4: Esquema de la Arquitectura de ROS: a) vinculación de nodos con *master* y servicios disponibles, b) envío de mensajes entre nodos mediante publicación y suscripción al *topic* correspondiente.

2.4. S-PTAM

S-PTAM (*Stereo Parallel Tracking and Mapping*) es un método que aborda el problema de SLAM basándose en visión estéreo. Fue desarrollado con el objetivo de ser ejecutado en tiempo real en ambientes de grandes dimensiones obteniendo una estimación precisa de la pose del robot mientras realiza una construcción del mapa del entorno que lo rodea.

Siguiendo el enfoque de PTAM [4] (*Parallel Tracking and Mapping*) y con el fin de aprovechar el poder de los procesadores multinúcleos, en S-PTAM se dividen las tareas de localización y mapeo en hilos de ejecución independientes, logrando de esta manera un muy buen desempeño en comparación

con otros métodos de SLAM actuales.

Al comenzar su ejecución, el método parte con las cámaras situadas en el origen de coordenadas del mundo y sin un mapa de su entorno. Luego, al recibir el primer par de *Stereo Frames* capturado por las cámaras, el mapa es inicializado mediante la triangulación de los puntos detectados en las imágenes de cada uno de los *frames*, llamados *Map Points*. A partir de entonces, los *Stereo Frames* recibidos serán localizados al estimar la pose de la cámara en base al mapa actual. Esta tarea se logra proyectando los puntos 3D sobre el plano de cada imagen para luego buscar las correspondencias con los *features* extraídos en la misma comparando cada uno de sus descriptores. Dichas correspondencias son llamadas correspondencias 2D-3D ya que representan la observación de un punto en el espacio por parte de las cámaras. Además, un *Stereo Frame* puede ser seleccionado para ser agregado al mapa y en consecuencia recibirá el nombre de *Key Frame*. Esta selección se lleva a cabo mediante una heurística y tiene el fin de agregar al mapa sólo aquellos *Stereo Frames* que aporten una cantidad de información significativa del entorno con el fin de no generar un mapa excesivamente grande, que desaproveche espacio en memoria e implique un tiempo excesivo a la hora de ser procesado. En particular, el *Stereo Frame* que inicia el mapa es considerado como un *Key Frame*. Entonces, a medida que el robot avance en su recorrido obtendrá nuevas imágenes estéreo que serán procesadas con el método descripto, de esta manera se cuenta con un mapa incremental que aumentará de tamaño en cada iteración del proceso.

Al mismo tiempo y en forma paralela a *Tracker*, se inicia el hilo que se encarga de realizar el refinamiento del mapa realizando tareas como el ajuste de poses de la cámara y de *Map Points*. Este refinamiento se lleva a cabo mediante el método de *Bundle Adjustment* [22]. También este hilo busca nuevas asociaciones entre *Key Frames* y *Map Points* para aumentar las restricciones entre ellos. Además, realiza la eliminación tanto de *Map Points* como de *Measurements* considerados espurios o inconsistentes por el método de *Bundle Adjustment*.

2.4.1. Arquitectura general

La Figura 2.5 muestra un diagrama de la estructura general de S-PTAM, la cual consta de dos funcionalidades principales: *Tracking* que recibe la información de los puntos extraídos de las imágenes estéreo y a partir de ellos calcula la pose del robot y crea los puntos que luego formarán parte del mapa, *Local Mapping* (o *Mapper*) que realiza las optimizaciones sobre el mapa (*refinement*), establece nuevas conexiones entre nuevos puntos del mapa y *Key Frames* que fueron previamente agregados, y además se encarga de eliminar aquellos puntos que son considerados erróneos. Otra parte fundamental es el mapa que se construye durante el proceso. El mismo se almacena en memoria compartida y es accedido y modificado por sendos

módulos.

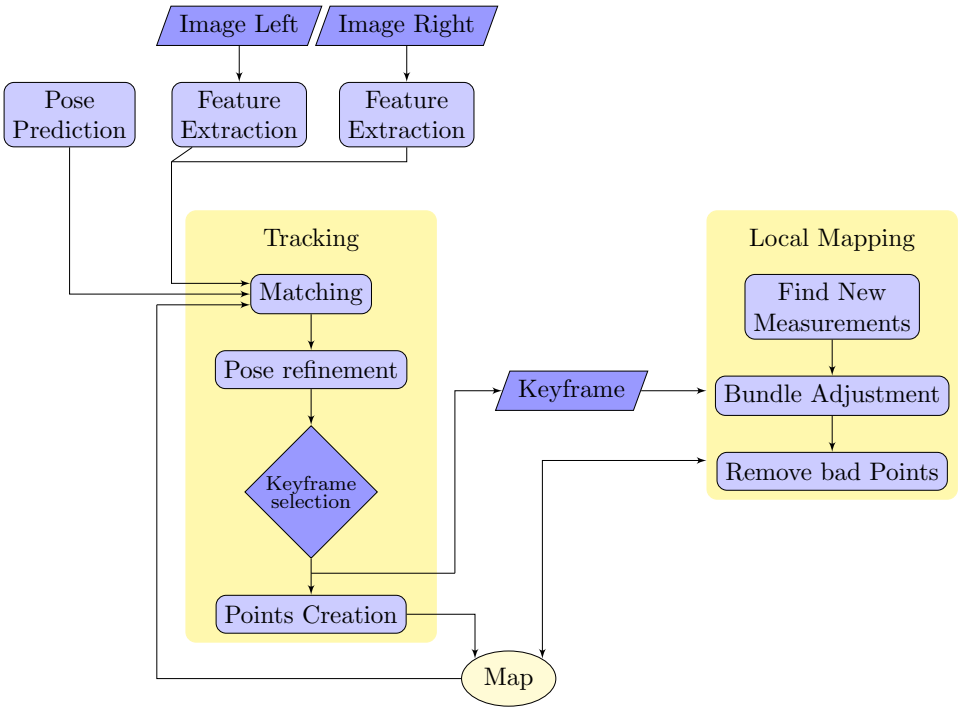


Figura 2.5: Diagrama de estructura general de S-PTAM [10].

2.4.2. Estructuras de datos

El mapa construido por S-PTAM es representado internamente como un grafo bipartito (un ejemplo del mismo se puede apreciar en la Figura 2.6) donde los nodos se corresponden con los *Key Frames* y los *Map Points* obtenidos durante el proceso; y las aristas representan las observaciones de los *Map Points* desde los *Key Frames* correspondientes, es decir que una arista que vincula un *Map Point* con un *Key Frame* indica que ese *Map Point* fue observado desde dicho *Key Frame*, estas observaciones reciben el nombre de *Measurements* o correspondencias 2D-3D.

Esta representación del mapa permite encontrar fácilmente el conjunto de *Key Frames* desde los cuales se visualiza un *Map Point* determinado, este conjunto recibe el nombre de *Key Frames* de covisibilidad y sirve tanto para determinar el conjunto de *Map Points* que son altamente probables de ser observados en cada de *Stereo Frame* de entrada como para el refinamiento del mapa local del robot llevado acabo por el *Local Mapping*.

Por otro lado, los *Key Frames* son *Stereo Frames* que han sido seleccionados utilizando heurísticas determinadas. Actualmente, existen muchas heurísitcas de selección para determinar si un *Stereo Frame* debe ser un *Key*

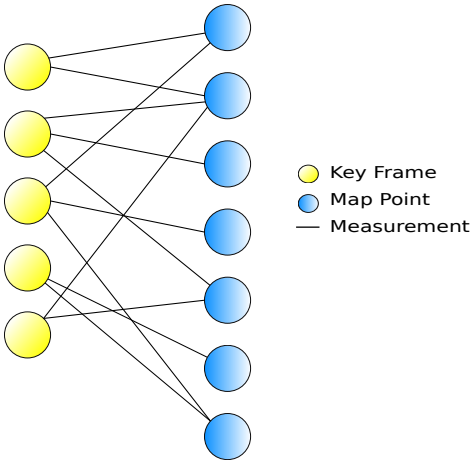


Figura 2.6: Ejemplo de grafo bipartito que representa al mapa reconstruido por S-PTAM.

Frame, ejemplos de algunos criterios utilizados son: la distancia euclídea al *Key Frame* más cercano en el mapa, la cantidad de información aportada por dicho *Key Frame* al agregarlo al mapa o el porcentaje de *Map Points* observados en comparación con el número de *Map Points* observados por el *Key Frame* más cercano. Cabe destacar que es esta última técnica mencionada la que se utiliza en este estudio.

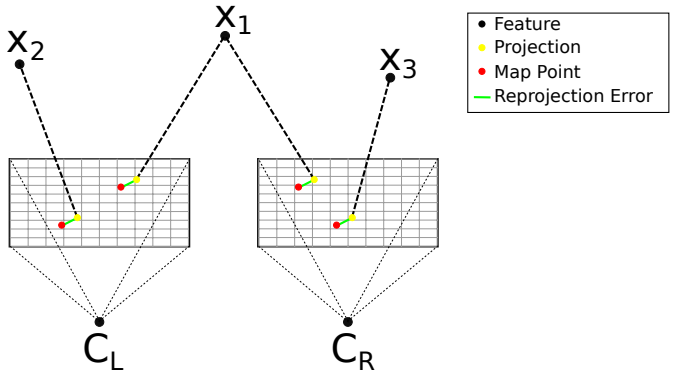
Internamente cada *Key Frame* contiene información de la posición y la orientación de la cámara al momento de ser capturado, los *features* que han sido detectados en los *frames* izquierdo y derecho; y las mediciones 2D-3D resultantes de las observaciones de los puntos del mapa en cada imagen.

Por último, los *Map Points* contienen información referente a su ubicación en el espacio (vector posición), el vector normal a la última cámara que lo observó, el descriptor (obtenido del feature de la imagen izquierda) que permite identificarlo unívocamente, y los *measurements* que lo vinculan con los *Key Frames* que lo observan.

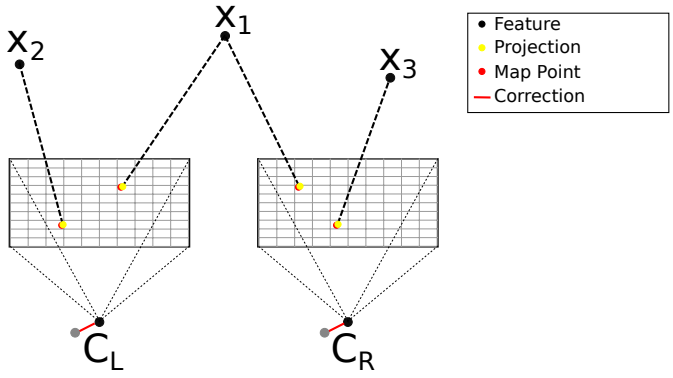
2.4.3. Tracker

Una de las funcionalidades principales del método S-PTAM es aquella que estima, a través de la determinación de la pose de cada *Stereo Frame*, la pose de las cámaras y por ende la del robot. Para ello, por cada par de imágenes estéreo que recibe extrae sus *features*, es decir sus puntos salientes y establece asociaciones 2D-3D entre ellos y los *Map Points* del mapa que posee hasta el momento. Para llevar a cabo esta tarea, proyecta los *Map Points* en cada uno de los planos de las imágenes y busca en cada imagen el *feature* correspondiente. Para asegurarse de que la asociación es la correcta se realiza una comparación entre el descriptor del *feature* propuesto y el

descriptor del *Map Point*, entonces si la distancia entre los descriptores es menor a un umbral dado, se establece dicha correspondencia. El conjunto de todas las correspondencias es utilizado para refinar la pose actual mediante la minimización del *error de reproyección*. En la Figura 2.7 se muestra el proceso de estimación de la pose durante la etapa de *tracking*. Una vez estimada la pose de la cámara, espera por un nuevo *Stereo Frame* que desencadene otra iteración del proceso. Cabe destacar que una vez que un *Stereo Frame* es procesado, el nodo *Tracker* determina, mediante la utilización de una heurística, si este es seleccionado como *Key Frame* para ser agregado al mapa.



(a)



(b)

Figura 2.7: Ajuste de la pose de la cámara mediante reducción del error de reproyección: a) se proyecta cada *Map Point* sobre los planos de la imagen estéreo, y se buscan correspondencia entre los *features* cercanos, b) se corrige la pose de la cámara reduciendo la distancia entre los *features* de la imagen y las proyecciones del *Map Point* correspondiente.

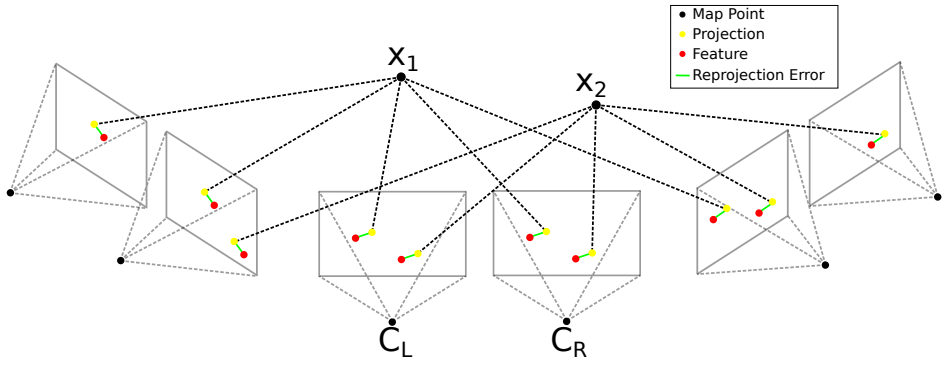
Sumado a dichas tareas, y a diferencia de muchas soluciones SLAM, *Tracker* también se encarga de triangular nuevos puntos del mapa a partir de las imágenes estéreo y de agregar al mapa los *Key Frames* liberando así a

otras partes como el *Mapper* de esta responsabilidad. Además, esto permite que *Tracker* disponga de los nuevos puntos de manera inmediata, sin esperar a que otros procesos realicen la tarea de ampliar el mapa.

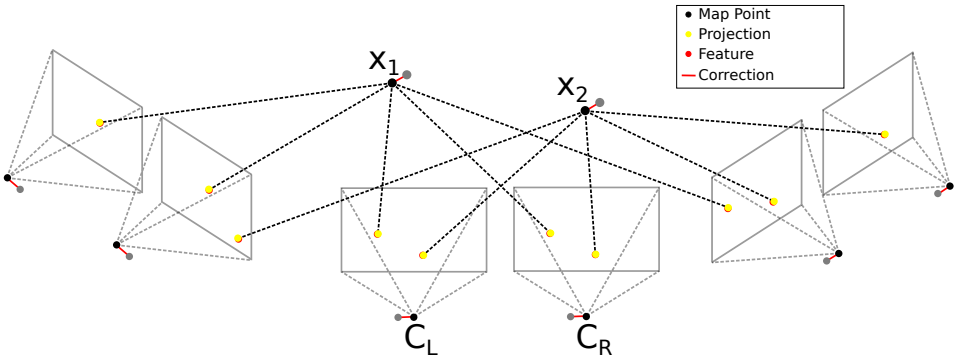
2.4.4. Mapper

Luego de que *Tracker* agregue un *Key Frame* y sus respectivos *Map Points* al mapa, *Mapper* aporta mayor robustez buscando nuevas correspondencias entre los nuevos *Key Frames* y los *Map Points* del mapa. Además, lleva a cabo la etapa de refinamiento del mismo. Para esto, se utiliza el método de optimización no lineal *Bundle Adjustment* [22], que ajusta un subconjunto de *Key Frames* cercanos al último *Key Frame* incorporado al mapa y los *Map Points* que en ellos se observan. Este subconjunto de *Key Frames* y *Map Points* se denomina *mapa local*. De esta manera se obtiene una reconstrucción más precisa del entorno. El ajuste consiste en refinar las poses de dichos elementos mediante la minimización del error de reproyección. Podemos encontrar un ejemplo de la aplicación del *local Bundle Adjustment* en la Figura 2.8.

Una vez optimizado el mapa, se eliminan aquellas observaciones que son marcadas como espurias (*outliers*). Luego, aquellos *Map Points* que no cuenten con observaciones asociadas serán removidos del mapa.



(a)



(b)

Figura 2.8: Ejemplo de aplicación de *Bundle Adjustment* en el que dos puntos del espacio son observados desde tres vistas distintas: a) los *Map Points* X_1 y X_2 son proyectados sobre los planos de las tres diferentes vistas y estas proyecciones son comparadas con los features correspondientes en cada plano, b) la pose de los *Map Points* y de las cámaras son ajustadas minimizando el error de reproyección entre los *features* de las imágenes y las proyecciones del *Map Point* correspondiente.

Parte II

Aportes originales

Capítulo 3

Diseño distribuido de S-PTAM

En este capítulo se detallan las tareas llevadas a cabo en el presente trabajo. Inicialmente se parte del desarrollo obtenido en S-PTAM (*Stereo Parallel Tracking and Mapping*), el cual es un método que aborda el problema de SLAM basándose en visión estéreo siguiendo el enfoque propuesto por PTAM. Teniendo en cuenta que S-PTAM es un sistema monolítico que divide las tareas de localización y mapeo en distintos hilos de ejecución, se propone un nuevo diseño que permita la ejecución de manera distribuida de sus componentes. De esta manera se podría lograr mayor versatilidad a la hora de contar con distintos tipos de hardware con distintas capacidades de procesamiento. Entonces, un robot móvil podría ejecutar parte del procesamiento correspondiente al módulo de localización (*tracking*) de manera *onboard* y por otro lado ejecutar el módulo de mantenimiento del mapa (*mapping*) en un dispositivo externo que disponga mayor poder de procesamiento, ya que esta tarea es la de mayor costo computacional. De esta manera, cada módulo dispondrá de manera exclusiva de toda la capacidad de procesamiento del dispositivo en el que se ejecute.

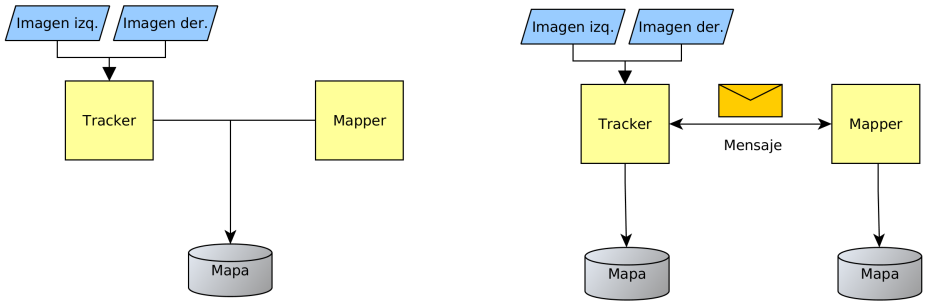
3.1. Arquitectura distribuida

Actualmente, el diseño de S-PTAM hace uso de memoria compartida en el que existe un único mapa sobre el cual trabaja todo el sistema. En base a este planteo se identifican las funcionalidades predominantes y se rediseña la arquitectura para que sean ejecutadas en módulos independientes.

En la arquitectura propuesta, cada uno de estos módulos será implementado como un nodo ROS distinto, que podrá interactuar con otros mediante el sistema de mensaje que proporciona este *framework* (presentado en la sección 2.3). La comparación entre dichas arquitecturas puede apreciarse en la Figura 3.1.

Esta nueva arquitectura trae aparejadas ventajas y desventajas. Entre las ventajas se destaca la posibilidad de que se realicen nuevos desarrollos que impliquen mejoras en alguno de los componentes de manera independiente. Esto posibilita, por ejemplo que si se desarrolla una nueva versión del nodo *Mapper*, ésta pueda ser ejecutada sin modificar el código del nodo *Tracker*. El otro punto importante es que provee tolerancia a fallos para los casos concretos en los que alguno de los componentes detenga su ejecución y ello no provoque la detención del resto del sistema.

Entre las desventajas se puede mencionar la duplicación de la memoria utilizada comparado con el caso en que se realice la ejecución del método en una única computadora. Por otro lado, otra desventaja es que se cuenta con dos mapas, uno para el nodo *Tracker* y otro para el nodo *Mapper*, que pueden ser inconsistentes en algunos momentos durante el proceso. Provocando en el peor de lo casos que el mapa del nodo *Tracker* no incorpore las mejoras realizadas por el nodo *Mapper* haciendo que la localización de la cámara se vea empobrecida.



(a) Arquitectura original. (b) Arquitectura propuesta.

Figura 3.1: Comparación entre la arquitectura de S-PTAM original y la arquitectura distribuida propuesta. En la arquitectura distribuida tanto el *Tracker* como el *Mapper* tienen una instancia del Mapa, y la comunicación entre ambos nodos se realiza a través de mensajes de ROS.

3.1.1. Identificación de objetos del mapa

Al proponer un funcionamiento distribuido será necesario contar con una copia del mapa actualizada en cada uno de los nodos y será fundamental que ambos mapas se encuentren sincronizados durante todo el proceso para mantener consistencia sobre el resultado obtenido. Para lograrlo, será necesario que cada nodo informe sobre los cambios que han sido realizados sobre su copia del mapa para que el otro nodo impacte esos cambios en el propio. En el diseño actual de S-PTAM los elementos del mapa tales como *Key Frames*, *Map Points* y *Measurements* pueden ser accedidos mediante el uso de

punteros debido a que se encuentran en espacios de memoria compartida. En cambio, en el diseño propuesto es necesario desarrollar un mecanismo de identificación de objetos que se encuentran en distintos espacios de memoria.

Es así que se decide incorporar identificadores en los objetos que forman parte del mapa y que son creados y modificados de forma dinámica durante el proceso. Estos identificadores se corresponderán, en cada nodo, al mismo objeto del mapa (cada uno con direcciones distintas de memoria), es decir que se agregan variables dentro de la declaración de cada clase con el fin de poder identificarlos y establecer una correspondencia con la dirección de memoria y el objeto almacenado durante la ejecución.

Para abstraer este comportamiento de correspondencia se recurre a la creación de la clase *Translations*, que internamente consta de un vector de punteros a objetos de tamaño fijo (previamente determinado[1]) y métodos que permiten asignar sus valores y obtener el objeto apuntado a partir de un identificador dado, en este caso el *id* es utilizado para identificar y referenciar unívocamente cada objeto. Concretamente, se define el puntero kf_j para denotar el *Key Frame* con $id = j$ y el puntero mp_i para el *Map Point* cuyo $id = i$. Cada uno de estos punteros es almacenado en su vector correspondiente. Es decir, que en definitiva cada vector contendrá en la posición *n-ésima* el puntero correspondiente al elemento con $id = n$.

Entonces, teniendo una instancia de esta clase para *Key Frames* y otra para *Map Points*, tanto *Tracker* como *Mapper* pueden hacer referencia a un objeto determinado sin importar que cada nodo lo tenga almacenado en un espacio de memoria distinta y de esta forma se provee un mecanismo para comunicarle al otro nodo sobre cambios en sus atributos de una manera clara y consistente.

La Figura 3.2 muestra la evolución de los objetos *Translations* desde el inicio de la ejecución, en la que las estructuras de cada nodo se encuentran vacías, particularmente se utiliza el vector llamado KFs para los *Key Frames* y el vector MPs para *Map Points*. Luego de recibir un *Stereo Frame* y seleccionarlo como *Key Frame*, se almacenan los datos correspondientes en las estructuras del nodo *Tracker*, dichos cambios se indican en color gris, es decir que se agregará en la posición 0 del vector KFs de *Tracker* un puntero a la dirección de memoria donde estará alojado el objeto correspondiente al *Key Frame* con $id = 0$, por otro lado se almacenarán en las entradas del vector MPs tantos punteros como *Map Points* hayan sido generados a partir de este *Key Frame* de manera que en la posición i de este vector se encuentre el puntero al *Map Point* que posea $id = i$. Una vez asignado todos los valores correspondientes al *Key Frame* procesado, *Tracker* conformará y enviará un mensaje a *Mapper* para que este replique toda la información en sus estructuras y actualice su propio mapa. Posteriormente, el nodo *Map-*

[1] en los experimentos se utilizó un tamaño igual a 5000 para el vector de *Key Frames* y 2000000 para el vector de *Map Points*

per realizará el refinamiento de su propio mapa mediante el método *Bundle Adjustment*, dicha operación realizará ajustes y modificaciones sobre la pose de los *Key Frames* y los *Map Points*, luego estas modificaciones serán comunicadas a *Tracker* mediante un mensaje con el fin de que este nodo aplique dichos cambios sobre los objetos que han sufrido modificaciones (indicados en la imagen en color celeste). A partir de entonces este proceso se repite de forma iterativa y de esta manera ambos nodos lograrán consistencia entre sus mapas.

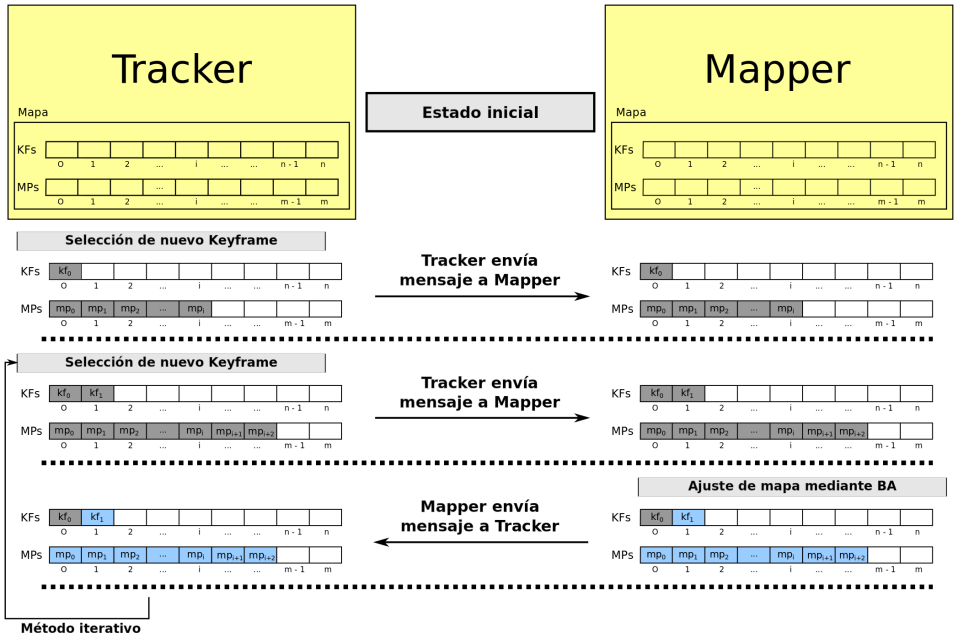


Figura 3.2: Modificaciones de los objetos *Translations* durante el inicio de la ejecución.

Durante el proceso existen momentos en los cuales los mapas de cada nodo no se encuentran completamente sincronizados, tal es el caso en el que uno de los nodos realiza modificaciones sobre su propio mapa pero todavía no le ha informado dichos cambios al otro nodo mediante el envío de un mensaje. Esto hace que haya pequeños instantes de inconsistencia entre la información que contiene el mapa de cada uno de los nodos. En el caso de estudio que este trabajo aborda, esa inconsistencia no genera grandes dificultades ni errores, ya que los mismos no son de gran magnitud y por ende no tienen influencia sobre el resultado obtenido. En el caso concreto en el que el nodo *Tracker* se encuentra realizando la etapa de localización, y el refinamiento del mapa por parte del nodo *Mapper* aún no se ha reflejado en el mapa de *Tracker*, la localización se habrá realizado utilizando un mapa que no se encuentra completamente actualizado, haciendo que la precisión de localización se vea afectada. Se observa que este comportamiento es el

esperado al plantear un diseño distribuido en el que se prioriza el proceso de localización de manera tal que éste nunca deba bloquearse a la espera de nuevas actualizaciones del mapa.

3.1.2. Estructura de mensajes

Para lograr comunicación entre los nodos se propone una estructura de mensajes definida en base a los estándares de ROS, los cuales incluyen tipos de dato como enteros, booleanos, coma flotante, caracteres, cadenas, etc. Además, las estructuras pueden conformarse con vectores anidados de estos tipos primitivos mencionados.

Una representación de la estructura completa de los mensajes utilizados en este trabajo puede verse en la Figura 3.3. En dicha figura, cada caja representa un tipo de mensaje distinto. En el encabezado de cada una de las cajas se especifica el nombre del mensaje, en su interior se determina por cada línea el tipo y el nombre de cada uno de los objetos contenidos en dicho mensaje. Cada tipo de objeto podrá hacer referencia a un tipo básico o a otra estructura de mensaje con el fin de incluir y anidar mensajes abstrayendo sus estructuras. Siguiendo estos lineamientos se crearon distintos tipos de mensajes que permiten establecer la comunicación entre los nodos de ROS manteniendo en todo momento la información contenida en cada elemento propio de S-PTAM.

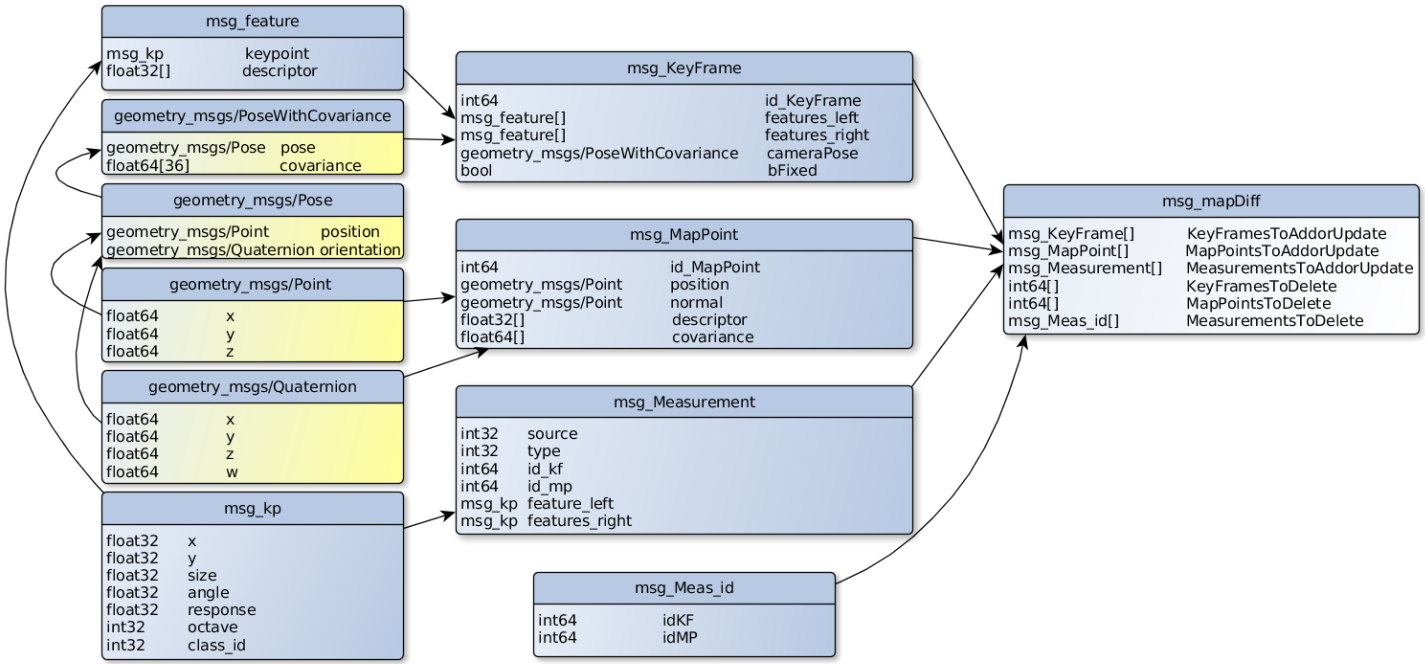


Figura 3.3: Esquema de estructuras de mensajes utilizada.

Para explicarlo de manera incremental, se identifica a los mensajes primitivos de ROS con color amarillo, mientras que las cajas de color azul corresponden a los mensajes creados en este trabajo. En particular se observa que el mensaje *geometry_msgs/Point* contiene 3 objetos de tipo *float64* en su estructura, estos objetos representan las coordenadas de un punto en el espacio a partir de los valores *x, y, z* correspondientes a cada eje coordenado. A su vez, el mensaje de tipo primitivo *geometry_msgs/Quaternion* contiene 4 valores también de tipo *float64* que sirven para representar la orientación de un objeto en un espacio tridimensional, este mensaje junto a *geometry_msgs/Point* son los componentes del mensaje *geometry_msgs/Pose*. De esta forma se construyen las estructuras de los mensajes.

Uno de los mensajes principales es *msg_KeyFrame*. El mismo contiene, como su nombre lo indica, la información referida a un *Key Frame*. En particular, su identificador es representado por el objeto de tipo *int64* y nombre *id_KeyFrame* (que es usado como subíndice a la hora de definir el puntero que será almacenado en la instancia *Translations* de cada nodo, como se describió previamente). Además, contiene dos vectores de mensajes de tipo *msg_feature* llamados *features_left* y *features_right* que representan al vector que contiene los *features* dectados en el *frame* izquierdo y al vector con los *features* del *frame* derecho respectivamente. También contiene datos sobre la pose de la cámara (*cameraPose*) y un valor booleano (*bFixed*) que determina si dicho *Key Frame* puede ser ajustado mediante el *Bundle Adjustment* del nodo *Mapper*.

Por su parte, *msg_MapPoint* contiene datos como el identificador (*id_MapPoint*, que también es utilizado por la instancia de *Translations*), la posición del punto en el espacio (*position*), la normal, los descriptores y la covarianza del *Map Point* en cuestión.

A su vez, el mensaje *msg_Measurements* contiene información respecto a la fuente (*source*) de la medición, es decir que especifica si una medición corresponde a un *Key Frame* que da origen a un nuevo *Map Point* del mapa. En dicho caso se especifica que resulta de una triangulación y se le asigna al campo *source* la palabra clave *triangulation*. Si la medición corresponde a un *Key Frame* que observa un *Map Point* generado a partir de un *Key Frame* anterior, el campo *source* es instanciado con la palabra clave *tracker*. En su defecto, si la medición corresponde a un *Key Frame* que observa algún *Map Point* que fue creado por un *Key Frame* recibido posteriormente, se la especifica como una re-observación y al campo *source* se le asigna la palabra clave *refind*. También se indica si la medición es de tipo (*type*) monocular o estéreo y los identificadores (*id_kf* y *id_mp*) del *Key Frame* y del *Map Point* involucrados. Por último se incluyen en el mensaje los *features* del lado izquierdo y derecho.

Finalmente, la estructura del mensaje principal utilizada en este trabajo se define en *msg_mapDiff* y es utilizada, tanto por el nodo *Tracker* como por el nodo *Mapper*, para informar sobre alguna actualización en el ma-

pa. En su estructura incluye estructuras de otros mensajes, tal es el caso de *KeyFramesToAddorUpdate* que es de tipo *msg_KeyFrame[]* y representa un vector de estructuras correspondiente al mensaje *msg_KeyFrame* que hace referencia a los *Key Frames* que fueron agregados o modificados y cuyos cambios deben ser informados al otro nodo. Análogamente, se incluyen vectores estructuras de los mensajes *msg_MapPoint* y *msg_meas* llamados *MapPointsToAddorUpdate* y *MeasurementsToAddorUpdate* respectivamente y que representan los *Map Points* y *Measurements* que han sido agregados o modificados recientemente. Además, también se incluyen en este mensaje estructuras para identificar tanto los *Key Frames*, *Map Points* y *Measurements* a eliminar del mapa en caso de ser necesario, esto se realiza mediante un vector de identificadores, los cuales constan de elementos de tipo *long int* para el caso de los *Key Frames* y de los *Map Points*, mientras que para identificar a los *Measurements* se utiliza un vector de mensajes de tipo msg_Meas_id que abstrae su estructura y consta de los identificadores del *Key Frame* y del *Map Point* involucrados en el *Measurement* a eliminar.

3.2. Tracker

Con el objetivo de lograr un procesamiento distribuido, y considerando que S-PTAM dispone de un único mapa compartido, se decide incorporar como primera medida una instancia del mapa al nodo *Tracker*. A su vez, también se incorporan las funciones y métodos que le permiten agregar, modificar y eliminar *Key Frames*, *Map Points* y *Measurements* del mapa[2]. Se puede observar que de esta manera el nodo *Tracker* queda compuesto por distintas funcionalidades que le permiten ser utilizado como un módulo independiente para resolver problemas de localización. Debido a que al recibir las imágenes estéreo provenientes de la cámara del robot móvil, puede procesar sus *frames* para luego crear y actualizar su instancia del mapa del entorno que lo rodea. De esta forma, se logra una independencia del nodo *Tracker* respecto del nodo *Mapper*, pudiendo ser ejecutado sin necesidad de que *Mapper* esté ejecutándose simultáneamente. Esta característica se constituye como uno de los principales aportes de este trabajo, ya que hasta el momento no era posible llevarla a cabo con el diseño monolítico actual de S-PTAM.

Si bien los resultados son más precisos al ejecutar ambos nodos concurrentemente, poder utilizar el nodo *Tracker* de forma independiente permite, por ejemplo, ejecutar una versión simplificada de S-PTAM en dispositivos que no tengan gran capacidad de cálculo.

[2]Las funcionalidades de modificación y eliminación serán llevadas a cabo por el nodo *Tracker* sólo cuando el nodo *Mapper* le indique que debe actualizar su mapa.

3.3. Mapper

El nodo *Mapper* constituye el otro componente fundamental de S-PTAM, el cual se acopla al nodo *Tracker* para trabajar de forma colaborativa. Está implementado como un nodo ROS y cuenta, al igual que *Tracker*, con una instancia del mapa y las funciones y métodos para poder agregar, modificar y eliminar *Key Frames*, *Map Points* y *Measurements* del mismo. Dicho mapa será una copia del mapa que contiene *Tracker* y será ajustado continuamente mediante el método de optimización *Bundle Adjustment*. Cada modificación será comunicada a *Tracker* para que actualice su propio mapa a fin de lograr consistencia entre ellos.

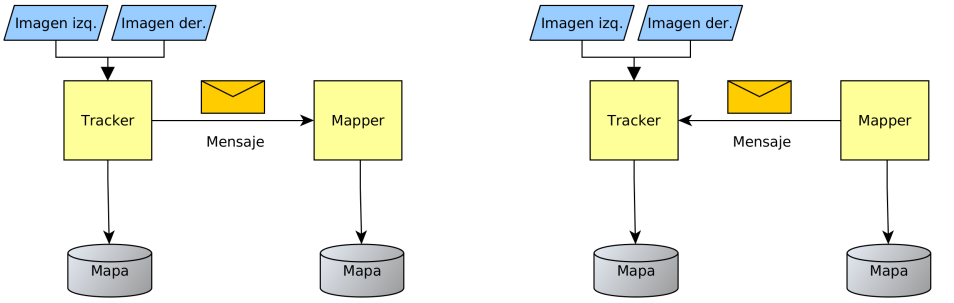
Es posible observar que el diseño propuesto en este trabajo permite distintas configuraciones en las cuales el módulo *Mapper* puede no formar parte de las mismas, siendo *Tracker* el único módulo indispensable ya que mantiene la localización del robot. Luego, opcionalmente es posible incorporar el módulo *Mapper* para que realice la optimización del mapa brindando mayor precisión al método. Además es posible obtener una configuración en la que múltiples instancias de *Mapper* se ejecuten de manera simultánea. Esta última configuración no fue tenida en cuenta en el desarrollo de este trabajo, sin embargo se deja planteada como un posible trabajo futuro.

3.4. Comunicación entre los nodos

La comunicación entre los nodos se lleva a cabo mediante la utilización del sistema de comunicación y gestión de mensajes provisto por ROS el cual facilita esta tarea a partir de canales de información entre nodos llamados *topics*. Para enviar mensajes a través de un *topic*, un nodo ROS deberá publicar en él, mientras que para recibir un mensajes publicado en un *topic* deberá suscribirse al mismo. Cabe destacar que el nodo emisor no conoce quienes son los nodos receptores de su mensaje, es decir que estos son unidireccionales y asíncronos (difusión).

Entonces *Tracker* y *Mapper* se comunican utilizando la estructura de mensajes en la que se incluye la información relacionada con los cambios en el mapa, es decir que cuando *Tracker* agrega elementos como *Key Frames*, *Map Points* y *Measurements* genera un mensaje, respetando la estructura de la Figura 3.3, en la que se incluye toda la información necesaria para que el nodo *Mapper* pueda agregar esos objetos en su propio mapa. Como contrapartida, cuando el nodo *Mapper* modifica su mapa al realizar el refinamiento del mismo, le comunica a *Tracker* dichos cambios utilizando el mencionado sistema de mensajes, incluyendo en ellos sólo información relevante para que *Tracker* pueda reflejar en su mapa dichas modificaciones y así evitar un mensaje que contenga información redundante e impacte negativamente en la *performance* del sistema. Por una cuestión de simplicidad

en el diseño, se decidió utilizar la misma estructura de mensajes, provista en la Figura 3.3, tanto para los que fueran enviados por *Tracker* como por *Mapper*. La Figura 3.4 muestra los dos casos posibles de comunicación entre los nodos.



(a) *Tracker* envía mensaje con actualizaciones del mapa a *Mapper*.

(b) *Mapper* envía mensaje con actualizaciones del mapa a *Tracker*.

Figura 3.4: Mensajes entre *Tracker* y *Mapper*.

Para facilitar el manejo de los mensajes se implementó la librería *messages_handler*, la cual contiene funciones que convierten elementos de S-PTAM en mensajes ROS y viceversa. Es decir, que para crear un mensaje la función tomará como argumento objetos de C++ (que representan los elementos que S-PTAM utiliza internamente) y devolverá el mensaje correspondiente siguiendo la estructura definida para tal fin. En contrapartida, la función inversa tomará como argumento un mensaje y retornará un objeto utilizando para ello los constructores de la clase adecuada.

El código de esta tesina está publicado como código libre y se encuentra disponible en `https://github.com/CIFASIS/distributed-sptam`.

Capítulo 4

Experimentos

En el capítulo anterior se presentó el diseño distribuido de S-PTAM, llamado a partir de ahora DS-PTAM, que permite ejecutarlo no sólo de manera distribuida sino que también bajo distintas configuraciones que requieren de menor capacidad computacional.

En este capítulo se analizan los resultados obtenidos al ejecutar la versión distribuida de S-PTAM desarrollada en este trabajo. En dicho análisis, se contrastan los resultados obtenidos con los de la versión original de S-PTAM utilizando como referencia el *ground-truth* (trayectoria real realizada por el *robot*) disponible para cada experimento.

4.1. Plataforma de Benchmark

El trabajo fue desarrollado utilizando el lenguaje de programación C++ sobre el *framework* ROS, particularmente en su versión Kinetic Kame [23]. El sistema operativo utilizado fue Ubuntu 16.04 de 64 bits. Se realizaron tres tipos de experimentos:

- En el primero de ellos se utilizó una PC con procesador Intel Core i3-4130 de 3.40 GHz de 4 núcleos y 8 GB de RAM. Este experimento plantea evaluar el desempeño del sistema propuesto en un *hardware* de alto rendimiento.
- En el segundo caso se realizaron experimentos utilizando *hardware* de menor capacidad de cómputo. En este casos se utilizó un equipo con procesador *Dual-Core* de 3.4 GHz y de 4 GB de memoria RAM.
- En el tercer caso se realizaron experimentos utilizando *hardware* de aún menor capacidad de cómputo, para simular el procesamiento *onboard* con el que cuentan hoy en día los vehículos móviles. En este caso se utilizó un equipo con procesador *Dual-Core* 2.4 GHz y 4 GB de memoria RAM.

En todos los tipos de experimentos en los que se utilizó arquitectura distribuida, se designó como estación base una *notebook* con procesador Intel Core i3-3217U de 1.80 GHz de 4 núcleos, 8 GB de RAM con sistema operativo UbuntuMATE 16.04 de 64 bits. En los experimentos iniciales se asignó la computadora de mayor poder de cálculo al Tracker ya que éste debe correr en tiempo real y en los casos de uso el nodo Mapper no se vió exigido.

Ambos equipos fueron conectados mediante una red cableada para simular la distribución de cómputo entre el robot móvil (PC) y la estación base (*notebook*). Aunque en el caso ideal al ejecutar DS-PTAM sobre un robot móvil sería conectar ambas computadoras mediante una red inalámbrica, se puede notar que el tráfico en la misma no es significativo ya que **no** se envían las imágenes, sino que sólo se envía la información de los *features* extraídos en las imágenes junto las modificaciones realizadas al mapa.

En los casos donde no se utilizó este tipo de arquitectura, se utilizó sólo la PC para ejecutar el sistema.

4.2. Datasets

Para llevar a cabo los experimentos se utilizaron *datasets* de dominio público ampliamente usados en el campo de la robótica móvil para evaluar sistemas de SLAM basados en visión. Cada uno de los *datasets* dispone de información precisa del estado del *robot* a cada instante, conocida como *ground-truth*, que es obtenida a partir de distintos sensores como *GPS* e IMU (*Inertial Measurement Unit*). Esta información permite realizar la comparación y evaluación de los distintos métodos de SLAM.

En primer lugar se utilizó la secuencia 00 del *dataset* KITTI [24], la misma consiste en un conjunto de imágenes estéreo capturadas por un par de cámaras montadas sobre un vehículo que transita un ambiente urbano (entorno *outdoor*). La trayectoria realizada por el vehículo es de 4 km aproximadamente. Las imágenes cuentan con una resolución de 1344×391 píxeles y son obtenidas a un *frame rate* de 10 Hz. La Figura 4.1 muestra el vehículo y los sensores incorporados en él para realizar el *dataset* KITTI, mientras que la Figura 4.2 muestra algunas imágenes tomadas por la cámara durante el recorrido.

Para evaluar el sistema en entornos *indoor*, se utilizó el *dataset* desarrollado en el contexto de la *European Robotics Challenge* (EuRoC) [25]. En particular, se utilizó la secuencia *Machine Hall 01*, la cual consiste en una secuencia de imágenes tomadas por un *Micro Aerial Vehicle* (MAV), que se desplaza dentro de una sala de máquinas industriales durante 182 segundos y recorre una trayectoria de 80.6 metros. Las imágenes cuentan con una resolución de 752×480 píxeles y son obtenidas a un *frame rate* de 20 Hz. La Figura 4.3 muestra el *drone* utilizado para realizar el *dataset*

Figura 4.1: Vehículo y sensores utilizados en el *dataset* KITTI.

Figura 4.2: Capturas de la secuencia *00* del *dataset* KITTI.

EuRoC, mientras que la Figura 4.4 muestra algunas imágenes tomadas por la cámara durante el recorrido. Este *dataset* refleja el caso ideal en el que el diseño distribuido de S-PTAM cobra mayor sentido ya que las limitaciones impuestas por este tipo de vehículos hacen evidente la necesidad de reducir el equipamiento destinado al cómputo montado sobre el mismo.

4.3. Configuraciones de DS-PTAM

El sistema DS-PTAM da origen a nuevas configuraciones que no eran posibles en la versión original. Las mismas se detallan a continuación:

Figura 4.3: MAV utilizado para realizar el *dataset* EuRoC.

Figura 4.4: Capturas de la secuencia *Machine Hall 01* del *dataset* EuRoC.

- DS-PTAM (Tracker): Ejecución del nodo Tracker únicamente.
- DS-PTAM (Tracker + Mapper): Ejecución de los nodos Tracker y Mapper en la misma PC.
- DS-PTAM: Ejecución de los nodos Tracker y Mapper de forma distribuida.

La primera de las configuraciones planteadas corresponde a los casos en los que no se cuenta con dispositivos de alto poder de cómputo ni con una estación base que pueda realizar la tarea computacional más costosa como es el refinamiento del mapa, la cual es realizada por el nodo *Mapper*. Para llevar a cabo el experimento con esta configuración se utilizó la PC de escritorio,

en la que se ejecutó tanto el nodo principal de ROS (nodo *master*) y el nodo *Tracker*. Cabe destacar que para cada uno de los experimentos realizados la reproducción del *dataset* se llevó cabo en la PC de escritorio.

La segunda configuración es similar a la anterior con la diferencia que se incorpora la ejecución del nodo *Mapper* en la misma PC que ejecuta nodo principal de ROS (nodo *master*) y el nodo *Tracker*. De esta manera se contará con un mapa refinado y por ende los resultados de localización serán más precisos, en este caso la PC realizará las tareas tanto de localización como de mapeo. Cabe aclarar que en este caso el paso de mensaje se transforma en una copia local de memoria de un proceso a otro.

Por último, la configuración distribuida plantea la ejecución del nodo principal de ROS (nodo *master*) y el nodo *Tracker* en la PC mientras que la ejecución del nodo *Mapper* se lleva a cabo en la *notebook*. Cabe aclarar que es necesario lograr la comunicación de ambos equipos a través de la red y para ello ROS provee mecanismos que involucran variables de entorno del sistema operativo como son *ROS_IP* y *ROS_MASTER_URI* las cuales deben ser correctamente asignadas. Concretamente, *ROS_IP* contiene la dirección IP del equipo que ejecuta algún nodo ROS y que debe ser identificado dentro de la red, por otro lado *ROS_MASTER_URI* le indica a los nodos ROS la dirección IP y el puerto de escucha del equipo donde se está ejecutando el nodo principal de ROS (nodo *master*), dicho nodo provee la comunicación con el resto de los nodos y permite el envío de mensajes entre ellos.

4.4. Resultados

Las Figuras 4.5 y 4.6 muestran la comparación entre las trayectorias estimadas por el sistema propuesto para distintas configuraciones. Se utilizaron las configuraciones DS-PTAM (Tracker), DS-PTAM (Tracker + Mapper), DS-PTAM y S-PTAM original contrastando sus trayectorias contra la trayectoria real realizada por el vehículo (*ground-truth*) tanto para los casos en los que se utilizó el *dataset* KITTI como EuRoC.

En ambos casos el *ground-truth* es representado en color negro. Mientras que el color celeste representa la trayectoria para la ejecución del nodo *Tracker* en la PC, dicha configuración es nomenclada como DS-PTAM (Tracker). En color marrón se representa la trayectoria para la configuración correspondiente a la ejecución de ambos nodos en la misma PC, identificada como DS-PTAM (Tracker + Mapper). Por otro lado, en color rojo se muestra la trayectoria para la ejecución de S-PTAM original en la PC. Finalmente en color verde, la ejecución distribuida de los nodos en la que el nodo *Tracker* fue ejecutado en la PC mientras que el nodo *Mapper* se ejecutó en la *notebook*, dicha configuración es referenciada en la figura como DS-PTAM.

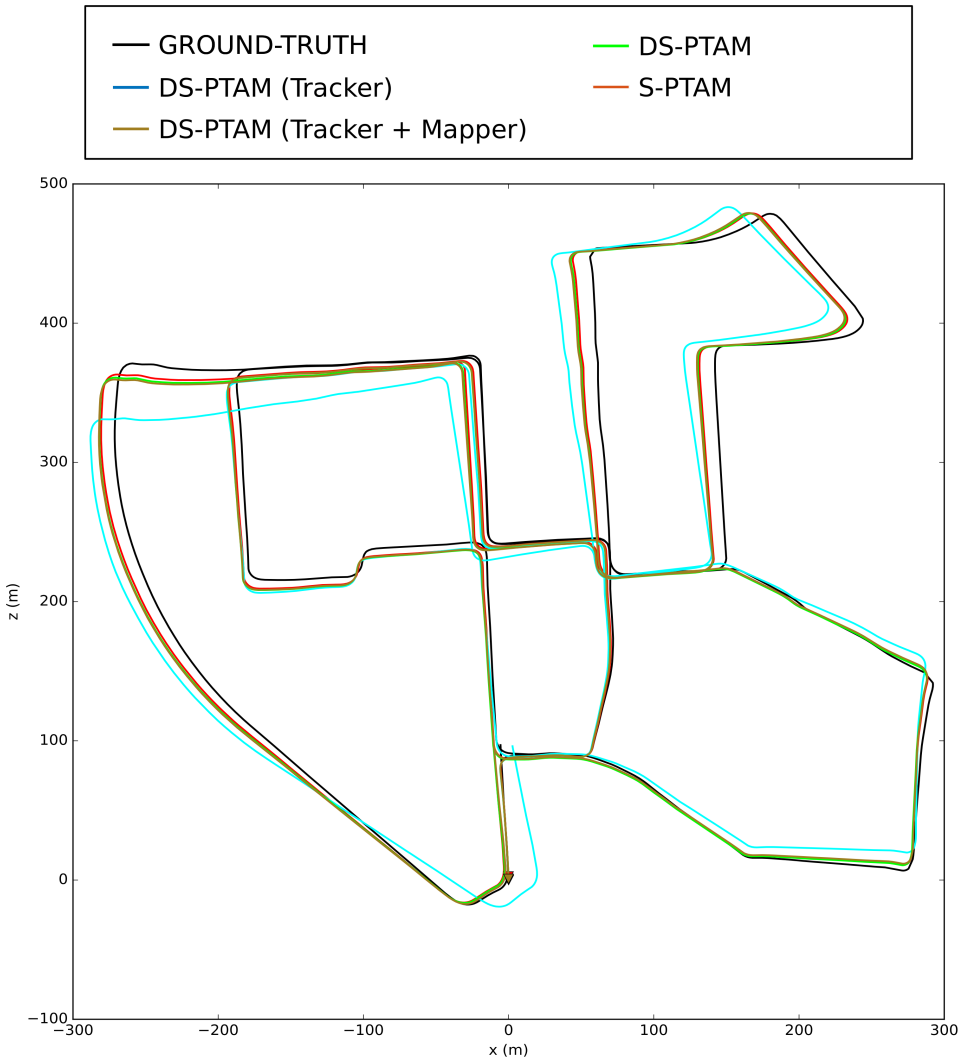


Figura 4.5: Comparación de las trayectorias estimadas para las configuraciones DS-PTAM (Tracker), DS-PTAM (Tracker + Mapper), DS-PTAM y S-PTAM original sobre la secuencia 00 del *dataset* KITTI.

Para poder evaluar el sistema propuesto se realiza un análisis de los errores de traslación y rotación relativos y absolutos. Además se analizan los valores máximos de cada uno de ellos y de RMSE (*Root Mean Square Error*) de los errores relativos. Para denotar dichos errores se utiliza la siguiente notación: t indica error de traslación, mientras que r hace mención al error de rotación. t^i_{abs} es la diferencia de traslación del *frame* i obtenida en el experimento respecto del *frame* i del *ground-truth*. Mientras que t^i_{rel} es la diferencia de traslación entre el *frame* i y el *frame* $i+1$ del experimento menos la diferencia entre el *frame* i y el *frame* $i+1$ del *ground-truth*.

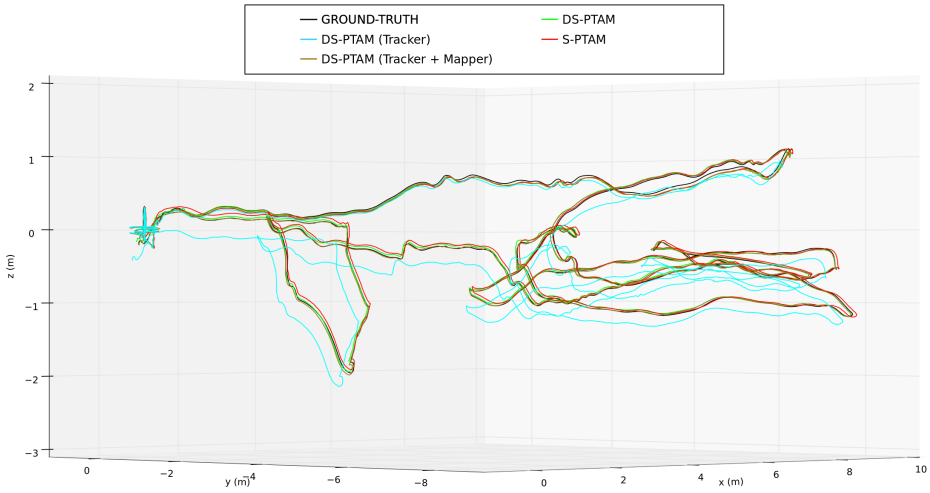


Figura 4.6: Comparación de las trayectorias estimadas para las configuraciones DS-PTAM (Tracker), DS-PTAM (Tracker + Mapper), DS-PTAM y S-PTAM original sobre la secuencia *Machine Hall 01* del *dataset* EuRoC.

De manera análoga se definen r^i_{abs} y r^i_{rel} para los errores de rotación. El cálculo de los errores absolutos se realiza de la siguiente manera:

$$x_{abs} = x_{gt} \ominus x_{exp},$$

donde x_{gt} es el valor obtenido del *ground-truth* y x_{exp} es el obtenido por el método correspondiente de SLAM. Mientras que los errores relativos se calculan mediante la siguiente fórmula:

$$x_{rel} = x'_{gt} \ominus x'_{exp},$$

donde $x' = [x_1 \ominus x_2, x_2 \ominus x_3, \dots]$. En el caso de la traslación el símbolo $\ominus$ representa la distancia euclídea mientras que para la rotación representa la diferencia como la definida en [26].

A continuación se muestran las tablas que detallan los valores arrojados por cada uno de los experimentos, en los cuales se utilizaron distintas configuraciones sobre los dos *datasets* empleados. Los subíndices *abs*, *rel* y *RMSE* hacen referencia a los errores absolutos, relativos y a la raíz cuadrada del error cuadrático medio, respectivamente. Mientras que *Max* hace referencia a los valores máximos obtenidos para cada tipo de error. En la última fila de cada una de las tablas se muestran los tiempos de procesamiento en promedio que demora el método de localización en procesar un par de imágenes estéreo para cada una de las ejecuciones. Se toma esta tarea como punto de comparación dado que la localización debe correr en tiempo

real en un sistema SLAM. En cada una de las tablas se resaltan —si los hubiera— los valores mínimos destacados por fila.

En la Tabla 4.1, que representa los resultados obtenidos de los experimentos realizados sobre el *dataset* KITTI, se puede observar que los máximos errores absolutos de traslación y de rotación para la configuración del nodo *Tracker* ejecutado de manera individual son muy superiores a los valores obtenidos para el resto de los casos. Este comportamiento se debe a que el resto de las configuraciones cuentan con la funcionalidad de refinamiento de mapa haciendo que cuenten con una representación más precisa del entorno, implicando una mejora en la localización. Para el caso de DS-PTAM y S-PTAM se observan resultados competitivos, es decir sus comportamientos son muy similares. A esto último se puede agregar que DS-PTAM tiene la ventaja de poder ejecutarse de forma distribuida.

	DS-PTAM (Tracker)	DS-PTAM (Tracker + Mapper)	DS-PTAM	S-PTAM
Max t_{abs} (m)	48,028	17,043	16,643	**14,576**
Max t_{rel} (m)	**0,299**	0,312	0,307	0,312
t_{RMSE} (m)	0,030	0,029	0,030	0,029
Max r_{abs} (deg)	11,792	9,026	8,971	**8,789**
Max r_{rel} (deg)	2,197	2,195	**2,186**	2,200
r_{RMSE} (deg)	0,119	0,120	0,119	0,120
Tracking (s)	0,044	0,049	0,044	0,044

Tabla 4.1: Comparación de errores y tiempos de localización promedio para los experimentos realizados sobre el *dataset* KITTI. La ejecución fue llevada a cabo utilizando un equipo con procesador *Quad-Core* de 3.40 GHz y 8 GB de memoria RAM.

En materia de tiempos, la configuración DS-PTAM (Tracker + Mapper) lleva un mayor costo computacional demorando más tiempo que el resto de los experimentos. Esto se debe a la complejidad extra requerida por la arquitectura distribuida que no está siendo aprovechada al correr *Tracker* y *Mapper* en una misma PC.

En la Tabla 4.2 se muestran los resultados obtenidos para los experimentos realizados sobre el *dataset* EuRoC. Se puede observar que S-PTAM obtiene los mejores resultados. Esto se debe a que S-PTAM, al trabajar con memoria compartida, no cuenta con la sobrecarga que trae aparejada la arquitectura distribuida en materia de mensajes y sincronización de datos. Cabe destacar que la configuración DS-PTAM (Tracker) emplea una cantidad de tiempo mayor que S-PTAM para estimar la localización. Aunque este comportamiento es exactamente el opuesto al esperado se debe a que esta configuración trabaja sobre un mapa que no está siendo optimizado a lo largo de la ejecución (debido a la ausencia del nodo *Mapper*) haciendo

que la tarea de localización demore más tiempo.

	DS-PTAM (Tracker)	DS-PTAM (Tracker + Mapper)	DS-PTAM	S-PTAM
Max t_{abs} (m)	0.340	0.257	0.266	**0.223**
Max t_{rel} (m)	0.05	0.05	0.05	0.05
t_{RMSE} (m)	0.005	0.005	0.005	**0.004**
Max r_{abs} (deg)	4.67	3.40	3.21	**2.85**
Max r_{rel} (deg)	1.04	0.84	1.12	**0.82**
r_{RMSE} (deg)	0.28	0.28	0.28	**0.22**
Tracking (s)	0.0392	0.0423	0.0380	**0.0379**

Tabla 4.2: Comparación de errores y tiempos de localización promedio para los experimentos realizados sobre el *dataset* EuRoC. La ejecución fue llevada a cabo utilizando un equipo con procesador *Quad-Core* de 3.40 GHz y 8 GB de memoria RAM.

Debido a que en el experimento anterior se utilizó un *hardware* cuya capacidad de cómputo es mucho mayor a la que disponen los *drones* hoy en día, se presenta un nuevo experimento sobre el *dataset* EuRoC en el que se utiliza un *hardware* de menor capacidad. En particular, se utiliza un procesador *Dual-Core* de 3.4 GHz y 4 GB de memoria RAM para simular la capacidad de procesamiento del *robot* móvil. Para la estación base se utiliza un procesador *Quad-Core* 1.80 GHz y 8 GB de memoria RAM.

En la Tabla 4.3 se muestran los resultados obtenidos para S-PTAM y para cada una de las configuraciones de DS-PTAM. Se puede observar que la *performance* de DS-PTAM (Tracker + Mapper) y de S-PTAM se ven fuertemente afectadas por la limitación de *hardware*, mientras que la *performance* de las configuraciones DS-PTAM (Tracker) y DS-PTAM se mantienen con respecto a los resultados del experimento anterior. El deterioro en la *performance* para las configuraciones DS-PTAM (Tracker + Mapper) y S-PTAM se debe mayormente a la pérdida de *frames* que sufre el sistema, esto es, el ritmo de generación de los *frames* es mayor al ritmo de procesamiento en este *hardware* limitado. Por otro lado, el nodo *Mapper* no llegará a ajustar el mapa en tiempo y forma para que el nodo *Tracker* pueda estimar la pose de manera precisa. Cabe mencionar que DS-PTAM obtiene en general los mejores resultados como es esperado.

Finalmente se realizó un experimento que consiste en ejecutar el sistema utilizando un *hardware* todavía más limitado, en concreto se utilizó un procesador *Dual-Core* de 2.4 GHz y 4 GB de memoria RAM para simular la capacidad de procesamiento del robot móvil. La Tabla 4.4 muestra los valores obtenidos tanto para los errores de localización como para los tiempos promedio. Cabe destacar que para este experimento las configuraciones correspondientes a DS-PTAM (Tracker + Mapper) y S-PTAM no lograron

	DS-PTAM (Tracker)	DS-PTAM (Tracker + Mapper)	DS-PTAM	S-PTAM
Max t_{abs} (m)	0.417	0.718	**0.253**	0.441
Max t_{rel} (m)	**0.059**	0.331	0.061	0.248
t_{RMSE} (m)	0.006	0.021	**0.005**	0.015
Max r_{abs} (deg)	4.708	10.773	**3.259**	9.004
Max r_{rel} (deg)	**1.271**	5.387	1.337	4.992
r_{RMSE} (deg)	0.286	0.383	**0.283**	0.387
Tracking (s)	0.067	0.097	**0.065**	0.096

Tabla 4.3: Comparación de errores y tiempos de localización promedio para los experimentos realizados sobre el *dataset* EuRoC. La ejecución fue llevada a cabo utilizando un equipo con procesador *Dual-Core* de 3.4 GHz y 4 GB de memoria RAM.

completar la tarea de localización, perdiendo la estimación de pose del robot a los pocos segundos de iniciado el experimento por lo cual no son reportados en la tabla. Esta pérdida de localización se debe a que dichas configuraciones requieren de una mayor potencia de procesamiento. Este es un resultado esperable que alienta el uso de un sistema distribuido para el caso en el que se disponga de un *hardware* con poca capacidad de cómputo. Por último, se puede mencionar que tanto DS-PTAM (Tracker) como DS-PTAM presentan una *performance* similar.

	DS-PTAM (Tracker)	DS-PTAM
Max t_{abs} (m)	0,515	**0,444**
Max t_{rel} (m)	0,303	**0,221**
t_{RMSE} (m)	0,018	**0,012**
Max r_{abs} (deg)	**6,641**	7,398
Max r_{rel} (deg)	**2,209**	4,039
r_{RMSE} (deg)	**0,318**	0,337
Tracking (s)	0.097	**0.089**

Tabla 4.4: Comparación de errores y tiempos de localización promedio para los experimentos realizados sobre el *dataset* EuRoC. La ejecución fue llevada a cabo utilizando un equipo con procesador *Dual-Core* de 2.4 GHz y 4 GB de memoria RAM.

En la Figura 4.7 se muestran las trayectorias para la configuración distribuida de DS-PTAM en los distintos tipos de *hardware* utilizados. Se observa que la ejecución que utiliza el menor poder de cómputo es la que mayor error presenta, mientras que para los casos de mayor capacidad de procesamiento:

alto (*Quad-Core* 3.4 GHz - 8 GB de RAM) y medio (*Dual-Core* 3.4 GHz - 4 GB de RAM) se aprecia un error menor, obteniendo trayectorias muy similares entre sí.

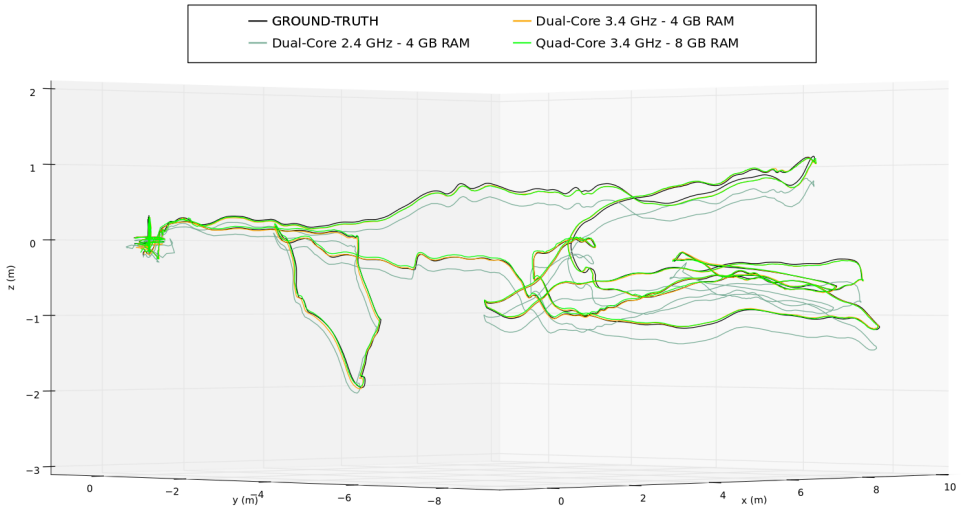


Figura 4.7: Comparación de las trayectorias estimadas para la configuración DS-PTAM distribuida en los distintos tipos de *hardware* utilizados para la secuencia *Machine Hall 01* del *dataset* EuRoC.

Parte III

Conclusiones y Trabajo Futuro

Capítulo 5

Conclusiones y Trabajo Futuro

En este trabajo se presentó un sistema de SLAM distribuido, llamado DS-PTAM. Inicialmente se partió del desarrollo de S-PTAM (*Stereo Parallel Tracking and Mapping*) [10], el cual es un método que aborda el problema de SLAM basándose en visión estéreo siguiendo el enfoque propuesto por PTAM [4].

Teniendo en cuenta que S-PTAM es un sistema monolítico que divide las tareas de localización y mapeo en distintos hilos de ejecución, se propuso un nuevo diseño que permite la ejecución de manera distribuida de sus componentes. De esta manera se puede lograr mayor versatilidad a la hora de contar con distintos tipos de hardware con capacidades distintas de procesamiento. Esta nueva arquitectura permite entonces que un robot móvil pueda ejecutar parte del procesamiento correspondiente al módulo de localización (*tracking*) de manera *onboard* y por otro lado ejecutar el módulo de mantenimiento del mapa (*mapping*) en un dispositivo externo que disponga de mayor poder de procesamiento, ya que esta tarea es la de mayor costo computacional. De esta manera, cada módulo dispondrá exclusivamente de toda la capacidad de procesamiento del dispositivo en el que se ejecute.

La arquitectura distribuida permite también numerosas configuraciones, como ejecutar sólo el nodo *Tracker*, el nodo *Tracker* junto con el nodo *Mapper* o incluso configuraciones más avanzadas, como por ejemplo tener más de un nodo *Mapper* analizando cada uno una sección distinta del mapa o compartir los nodos *Mapper* entre varios robots.

Se expusieron experimentos (utilizando *datasets* de dominio público) para validar la precisión y la *performance* del método propuesto. En dichos experimentos se evaluaron aspectos relacionados a las trayectorias estimadas, los errores de localización y los tiempos obtenidos en cada uno ellos. A partir de estos resultados se corroboran los beneficios de distribuir la carga computacional en varias computadoras, principalmente en aquellos casos en

los que se involucran vehículos móviles con bajo poder de cómputo, como es el caso de los *drones*. Debido a que en aquellos experimentos en los que se usó *hardware* muy potente los resultados del sistema distribuido no resultaban ventajosos en comparación con el sistema monolítico original se decidió realizar los experimentos con un *hardware* limitado (computacionalmente hablando). Los resultados fueron mucho más alentadores dejando en evidencia los beneficios de este diseño y justificando la complejidad agregada para trabajar bajo una arquitectura distribuida cuando se cuenta con una baja capacidad de cómputo.

Por otro lado, a lo largo del desarrollo de este trabajo se identificaron situaciones en las que se pueden aplicar mejoras, es por ello que se proponen algunas de ellas como trabajo futuro.

Una primera línea de trabajo sería adaptar el módulo de *Loop Closing* recientemente planteado en [26] a la versión de S-PTAM distribuida aquí propuesta. Esto permitiría que el robot pueda identificar que cierto lugar ya fue previamente visitado durante su recorrido y en consecuencia realizar un cierre de ciclo que le permita reducir el error global obtenido por el sistema.

Otra alternativa, que en principio no sigue la línea del paradigma propuesto en este trabajo pero que puede ser interesante de abordar, es la de plantear que el mapa se genere y actualice de forma colaborativa por parte de un conjunto de robots. Al tener desacoplado las tareas de localización y mapeo esto se vuelve más simple.

También puede resultar interesante desarrollar una interfaz para la gestión del mapa, definiendo una interfaz o protocolo que establezca las operaciones que se pueden realizar sobre el mapa. Un primer esbozo de esto se implementó en este trabajo mediante el intercambio de mensajes entre *Tracker-Mapper* aunque no se ha definido una interfaz común.

Por último, pero no menos interesante, se propone la creación de un nodo que se encargue de la extracción de los *features* en las imágenes que capturan las cámaras de robot, liberando de esta manera al nodo *Tracker* de una tarea computacionalmente costosa.

Bibliografía

[1] H. Durrant-Whyte, T. Bailey, Simultaneous localization and mapping: part i, IEEE Robotics Automation Magazine 13 (2) (2006) 99–110. doi:10.1109/MRA.2006.1638022.

[2] T. Bailey, H. Durrant-Whyte, Simultaneous localization and mapping (SLAM): part II, IEEE Robotics Automation Magazine 13 (3) (2006) 108–117. doi:10.1109/MRA.2006.1678144.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J. Leonard, Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age, IEEE Transactions on Robotics 32 (6) (2016) 1309–1332. doi:10.1109/TRO.2016.2624754.

[4] G. Klein, D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, in: Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR), IEEE Computer Society, Washington, DC, USA, 2007, pp. 1–10. doi:10.1109/ISMAR.2007.4538852.
URL http://dx.doi.org/10.1109/ISMAR.2007.4538852

[5] R. Mur-Artal, J. D. Tardós, ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras, CoRR abs/1610.06475.
URL http://arxiv.org/abs/1610.06475

[6] J. Engel, T. Schöps, D. Cremers, LSD-SLAM: Large-Scale Direct Monocular SLAM, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), Proceedings of the European Conference on Computer Vision (ECCV), Springer International Publishing, Cham, 2014, pp. 834–849. doi:10.1007/978-3-319-10605-2_54.
URL http://dx.doi.org/10.1007/978-3-319-10605-2_54

[7] H. Strasdat, A. J. Davison, J. M. M. Montiel, K. Konolige, Double window optimisation for constant time visual slam., in: ICCV, IEEE Computer Society, 2011, pp. 2352–2359.
URL http://dblp.uni-trier.de/db/conf/iccv/iccv2011.html#StrasdatDMK11

[8] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, D. Scaramuzza, Svo: Semidirect visual odometry for monocular and multicamera systems., IEEE Trans. Robotics 33 (2) (2017) 249–265.
URL http://dblp.uni-trier.de/db/journals/trob/trob33.html#ForsterZGWS17

[9] J. Engel, J. Stückler, D. Cremers, Large-scale direct slam with stereo cameras, in: Proceedings of the IEEE International Conference on Intelligent Robots

and Systems (IROS), IEEE, 2015, pp. 1935–1942. doi:10.1109/IROS.2015.7353631.
URL http://dx.doi.org/10.1109/IROS.2015.7353631

[10] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, J. J. Berlles, Stereo parallel tracking and mapping for robot localization, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 1373–1378.

[11] M. De Croce, T. Pire, F. Bergero, Diseño Distribuido de un Sistema de Mapeo y Localización basado en visión para un Robot Móvil, in: Anales de las IX Jornadas Argentinas de Robótica, 2017, pp. 116–121.

[12] R. Gamage, M. Tuceryan, An experimental distributed framework for distributed simultaneous localization and mapping, in: Electro Information Technology (EIT), 2016 IEEE International Conference on, IEEE, 2016, pp. 0665–0667.

[13] L. Riazuelo, J. Civera, J. M. M. Montiel, C2tam: A cloud framework for cooperative tracking and mapping. 62 (2014) 401–413.
URL http://dblp.uni-trier.de/db/journals/ras/ras62.html#RiazueloCM14

[14] R. Williams, B. Konev, F. Coenen, Scalable distributed collaborative tracking and mapping with micro aerial vehicles., in: IROS, IEEE, 2015, pp. 3092–3097.
URL http://dblp.uni-trier.de/db/conf/iros/iros2015.html#WilliamsKC15

[15] N. Mahdoui, E. Natalizio, V. Frémont, Multi-uavs network communication study for distributed visual simultaneous localization and mapping., in: ICNC, IEEE Computer Society, 2016, pp. 1–5.
URL http://dblp.uni-trier.de/db/conf/iccnc/iccnc2016.html#MahdouiNF16

[16] C. Forster, S. Lynen, L. Kneip, D. Scaramuzza, Collaborative monocular slam with multiple micro aerial vehicles., in: IROS, IEEE, 2013, pp. 3962–3970.
URL http://dblp.uni-trier.de/db/conf/iros/iros2013.html#ForsterLKS13

[17] R. Williams, B. Konev, F. Coenen, Multi-agent environment exploration with ar.drones., in: TAROS, Vol. 8717 of Lecture Notes in Computer Science, Springer, 2014, pp. 60–71.
URL http://dblp.uni-trier.de/db/conf/taros/taros2014.html#WilliamsKC14

[18] H. Bischof, Dense reconstruction on-the-fly, in: Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), CVPR '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 1450–1457.
URL http://dl.acm.org/citation.cfm?id=2354409.2355079

[19] A. H. Jazwinski, Stochastic processes and filtering theory, Mathematics in science and engineering, Academic press, New York, 1970, uKM.
URL http://opac.inria.fr/record=b1078357

[20] N. Metropolis, S. Ulam, The Monte Carlo Method, Journal of the American Statistical Association 44 (247) (1949) 335–341, pMID: 18139350.

arXiv:http://www.tandfonline.com/doi/pdf/10.1080/01621459.1949.10483310, doi:10.1080/01621459.1949.10483310.
URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1949.10483310

[21] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: an open-source Robot Operating System, in: ICRA Workshop on Open Source Software, 2009.

[22] B. Triggs, P. F. McLauchlan, R. I. Hartley, A. W. Fitzgibbon, Bundle Adjustment — A Modern Synthesis, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, Ch. 21, pp. 298–372. doi:10.1007/3-540-44480-7_21.
URL http://dx.doi.org/10.1007/3-540-44480-7_21

[23] ROS Kinetic Kame, http://wiki.ros.org/kinetic, desarrollada: 23 de mayo del 2016.

[24] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, Vision Meets Robotics: The KITTI Dataset, The International Journal of Robotics Research, IJRR 32 (11) (2013) 1231–1237. doi:10.1177/0278364913491297.
URL http://dx.doi.org/10.1177/0278364913491297

[25] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, R. Siegwart, The euroc micro aerial vehicle datasets, The International Journal of Robotics ResearcharXiv:http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html, doi:10.1177/0278364915620033.
URL http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[26] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, J. Jacobo Berlles, S-PTAM: Stereo Parallel Tracking and Mapping, Robotics and Autonomous Systems (RAS) 93 (2017) 27 – 42. doi:10.1016/j.robot.2017.03.019.