



MÁSTER OFICIAL EN VISIÓN ARTIFICIAL

Curso Académico 2019/2020

Trabajo Fin de Máster

Visual Odometry SD-SLAM

Autor:

Javier Martínez del Río

Tutores:

José María Cañas

Diego Martín Martín

Resumen

Resumen...

Índice general

1. Introducción	1
1.1. Visión Artificial	1
1.2. Visual SLAM	3
1.2.1. Conceptos	5
1.3. Robótica	6
1.4. Estructura del documento	7
2. Objetivos	9
2.1. Descripción del problema	9
2.2. Objetivos	10
3. Estado del arte	13
3.1. MonoSLAM	13
3.2. PTAM	14
3.3. SVO	16
3.4. ORB-SLAM	17
4. Herramientas	21
4.1. SD-SLAM	21
4.2. Unidad de Medición Inercial (IMU)	23
4.3. Conjunto de datos para evaluación	24
4.4. Métricas de evaluación	28
5. Diseño e implementación	29
5.1. Introducción	29
5.2. Sistemas de referencia	32
5.3. Odometría inercial	33
5.3.1. Estimación inercial	34
5.3.2. Estimación y actualización de la escala	36
5.4. Odometría durante el estado de pérdida	38

5.5. Restauración	39
Bibliografía	43

Índice de figuras

1.1.	Comparación entre la información visual y su representación.	2
1.2.	Ejemplo de Realidad Aumentada.	4
1.3.	Reconstrucción 3D mediante <i>Structure from Motion</i>	5
1.4.	Brazo robótico industrial (a). <i>Software</i> de conducción autónoma (b).	7
2.1.	Pérdida de información de escala al proyectar objetos del mundo real (3D) al plano imagen (2D)	10
3.1.	Estimación de la posición de espacial de los puntos 3D del mapa en MonoSLAM.	14
3.2.	Puntos detectados (a) y mapa generado (b) por PTAM.	16
3.3.	Proyección de los puntos del mapa visibles desde el fotograma anterior (a) e incertidumbre en la profundidad de un punto 3D (b) en SVO.	17
3.4.	A la izquierda se muestra la localización de los puntos característicos detectados con ORB en color verde y el mapa generado a la derecha.	18
4.1.	Funcionamiento de un acelerómetro.	24
4.2.	Configuración sensorial del vehículo de KITTI.	25
4.3.	Secuencia 00 perteneciente al conjunto de datos Kitt Odometría.	26
4.4.	Secuencia 05 perteneciente al conjunto de datos Kitt Odometría.	26
4.5.	Secuencia 06 perteneciente al conjunto de datos Kitt Odometría.	27
4.6.	Secuencia 07 perteneciente al conjunto de datos Kitt Odometría.	27
5.1.	Representación del módulo de <i>Tracking</i> de SD-SLAM como una máquina de estados finitos.	30
5.2.	Representación del módulo de <i>Tracking</i> de SDIO-SLAM como una máquina de estados finita.	31
5.3.	Representación de la transformación del sistema de referencia de la IMU (izquierda) al sistema de referencia de SD-SLAM Mundo (derecha).	33
5.4.	Proceso para la estimación de la posición y orientación del sensor IMU.	34

Índice de tablas

Capítulo 1

Introducción

En este capítulo se introducirá de forma general el marco en el que se encuadra el presente trabajo de fin de máster (TFM). Concretamente, se explicará qué es la Visión Artificial y, en particular, el campo de la auto-localización visual en entornos desconocidos mediante el uso del algoritmo de SLAM. Por otra parte, se expondrá qué es la robótica, así como su fuerte relación con la Visión Artificial. Por último, se expondrá la estructura que da forma al documento.

1.1. Visión Artificial

La Visión Artificial es la rama de la Inteligencia Artificial orientada a la captura y procesamiento de imágenes. En primer lugar, la captura es posible mediante el desarrollo de sensores capaces de recibir, procesar y almacenar parte del espectro electromagnético, como la luz visible o el infrarrojo, obteniendo imágenes con distintos tipos de información. En segundo lugar, el procesamiento engloba toda la parte del desarrollo de algoritmos capaces de interpretar el contenido de dichas imágenes.

Las imágenes contienen una gran cantidad de información útil para numerosos problemas, sin embargo, extraer y procesar dicha información no es un proceso sencillo. De hecho, la propia representación de las imágenes, que suele darse en forma de matriz numérica, ya es confusa. Como se puede apreciar en la Figura 1.1, nuestros sentidos no interpretan de igual modo una imagen (izquierda) que su representación (derecha).

Los orígenes de este campo se remontan a la década de los 60, cuando se conectó por primera vez una cámara a un computador con el fin de obtener y analizar la información. Larry Roberts fue una de las primeras personas en desarrollar un experimento en el cual no solo se capturaban imágenes, sino que se analizaba su contenido (una estructura de bloques) para posteriormente reproducirlo desde otra perspectiva.

En sus inicios, las técnicas de visión artificial estuvieron limitadas por la capacidad de cómputo de los ordenadores de la época. Sin embargo, en los últimos años, con el avance de

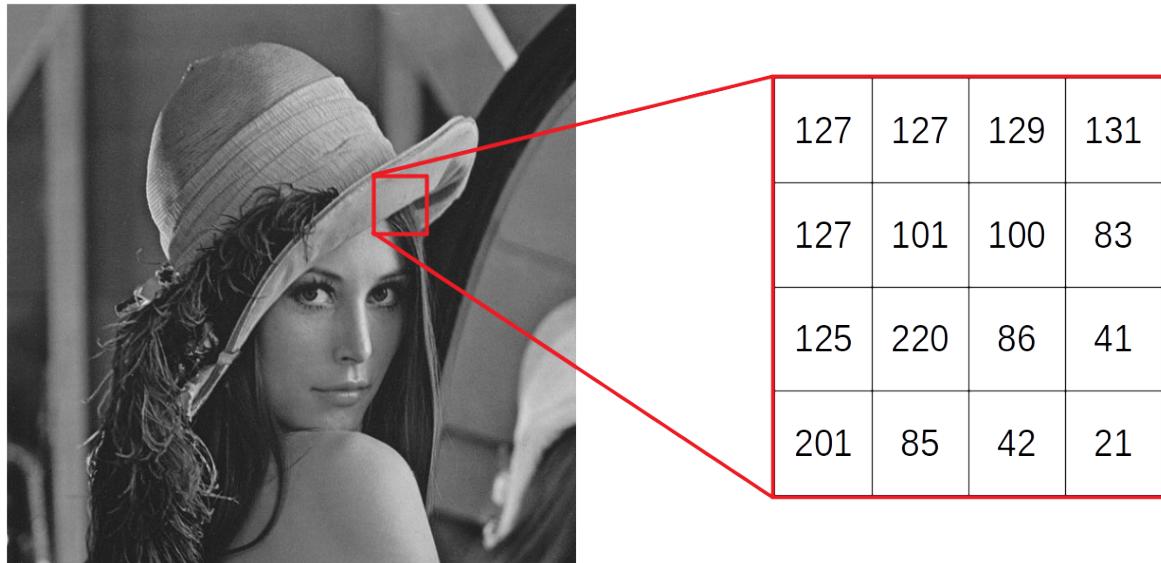


Figura 1.1: Comparación entre la información visual y su representación numérica.

la tecnología y la reducción del coste económico del *hardware*, se ha producido un gran avance en este campo. Esto ha permitido la aparición de algoritmos capaces de trabajar en tiempo real y estando presente en campos como la vigilancia, la robótica, la medicina e incluso los videojuegos. Algunas de las aplicaciones clásicas de este campo son la siguientes:

- **Detección de objetos.** Es posible identificar si un objeto se encuentra en una imagen realizando búsquedas por rasgos característicos del mismo, como la forma, el color, textura o patrones presentes en él.
- **Seguimiento de objetos.** Como extensión de la tarea anterior, es interesante poder realizar un seguimiento a un mismo objeto, de forma inequívoca, a lo largo del tiempo.
- **Reconocimiento de caracteres.** Esta técnica, conocida como OCR por sus siglas en inglés (*Optical Character Recognition*), permite la detección e identificación de los caracteres en un documento, ya sean digitales o manuscritos, con la finalidad de digitalizar el contenido de los mismos.

El rendimiento y precisión de estas tareas ha aumentado en los últimos años debido al uso de las redes neuronales, más concretamente al uso de técnicas de *Deep Learning*. La mejora ha sido tal que algunos sistemas de reconocimiento de patrones visuales obtienen tasas de error inferiores a las humanas. El primer sistema «sobrehumano» fue obtenido en 2011 en la competición de reconocimiento de señales de tráfico IJCNN, donde el sistema vencedor obtuvo una tasa de error dos veces mejor que la obtenida por sujetos de prueba humanos [30].

Sin embargo, hay campos donde las técnicas de *Deep Learning* aún no han descartado especialmente respecto a técnicas más clásicas de Visión Artificial, como por ejemplo la auto-

localización. Este problema consiste en la estimación de la ubicación de la cámara en el entorno (conocido o desconocido) que la rodea. Esta técnica es esencial para campos como la robótica, donde es imprescindible reconocer y ubicarse en el entorno para evitar choques y/o comportamientos anómalos; o al reciente campo de la realidad aumentada.

En la próxima sección se profundiza en un algoritmo de auto-localización llamado Visual SLAM.

1.2. Visual SLAM

Localización y mapeado simultáneo o SLAM por sus siglas en inglés (*Simultaneous Localization And Mapping*) es un algoritmo de auto-localización. SLAM es el término empleado para describir el proceso por el cual, a partir de información obtenida de sensores, es posible generar un mapa del entorno que lo rodea, al mismo tiempo que se localiza en él. Tiene sus orígenes en el campo de la robótica, siendo esta una de sus principales áreas de investigación, donde cobra especial importancia en robots autónomos que operan en entornos desconocidos.

Cuando el sensor principal de SLAM son una o más cámaras nos referimos al modelo como Visual SLAM. Pese a ser un algoritmo que aún está en desarrollo, dado que no hay una solución exacta para el problema que plantea, es empleado en un numerosas aplicaciones. Algunas de ellas son las siguientes:

- **Robot aspirador:** estos dispositivos autónomos son equipados con cámaras (u otros sensores similares como los LIDAR) siendo capaces de generar un mapa de los hogares navegando por ellos. La ventaja en este caso de conocer el entorno y su ubicación es la capacidad de generar y optimizar las rutas de limpieza, al mismo tiempo que evitan obstáculos en su trayectoria.
- **UAV:** los vehículos aéreos no tripulados (UAV por sus siglas en inglés) como los drones pueden emplear SLAM para visualizar el entorno que los rodea y tomar decisiones en tiempo real.
- **Realidad aumentada:** SLAM no solo tiene cabida en el mundo de la robótica, en el campo de la realidad aumentada también es posible hacer uso de este algoritmo para relacionar de un mejor modo el mundo real con el virtual. Estas aplicaciones emplean el mapa obtenido para introducir los elementos visuales de forma más realista. Esto solo es posible si se conoce la posición del sensor y el entorno. Un ejemplo de realidad aumentada puede observarse en la Figura 1.2, donde se incluye mobiliario en una habitación de forma más genuina.

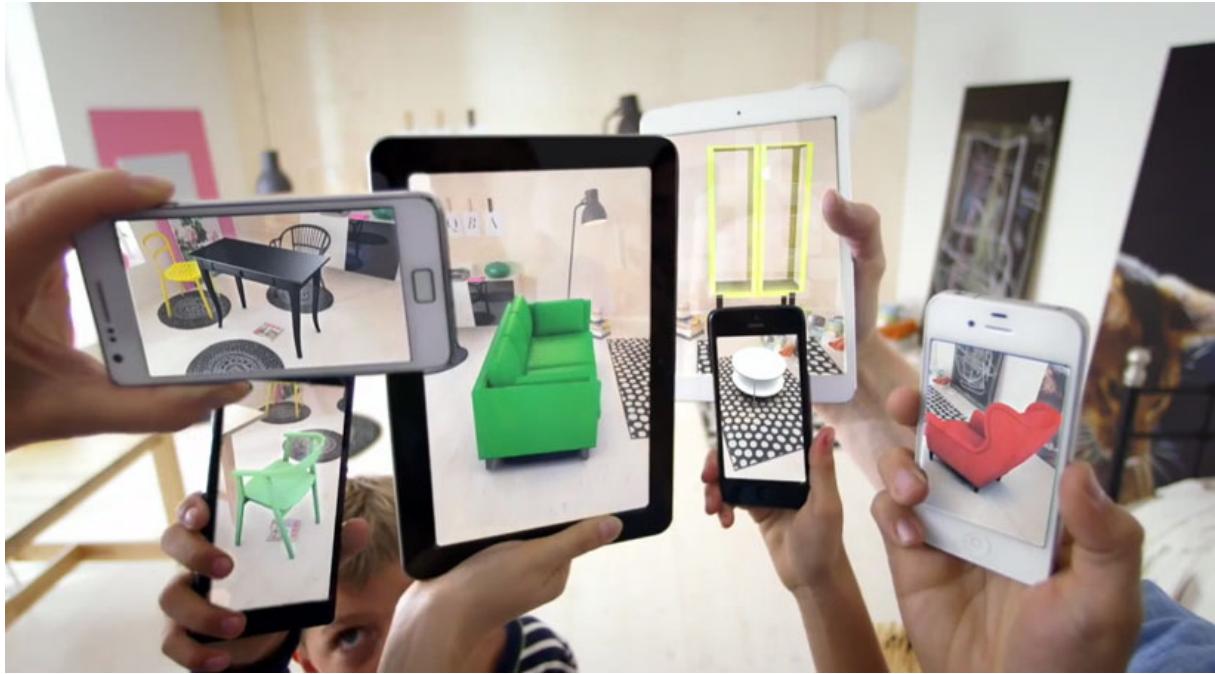


Figura 1.2: Ejemplo de aplicación de Realidad Aumentada.

Visual SLAM tiene sus raíces en la línea de investigación conocida como *Structure from motion*. Dicha línea está centrada en la reconstrucción automática de estructuras 3D a partir de un conjunto de imágenes. Este proceso se basa en la premisa de que, dadas múltiples vistas de un mismo punto tridimensional a lo largo de distintas imágenes, es posible estimar su posición en el espacio mediante el uso de triangulación. En la Figura 1.3 se puede observar un ejemplo de reconstrucción 3D a partir de 3 imágenes.

Structure from motion supuso un gran avance en las técnicas de detección de puntos característicos, también llamados puntos de interés, que serían de gran utilidad posteriormente en Visual SLAM. Estos puntos son llamados característicos porque son lo suficientemente únicos (por sus propiedades o entorno que los rodean) que los vuelven lo muy robustos de cara a ser detectados de nuevo ante cambios de iluminación o distintos ángulos de vista. Los extractores de puntos característicos más conocidos son *SIFT* [19], *SURF* [1], *FAST* [26] y *ORB* [29].

Esta técnica serviría como idea básica del funcionamiento de Visual SLAM. De forma general, la estructura a reconstruir es el entorno por donde navega la cámara. Por otra parte, el conjunto de las imágenes es generado a lo largo del tiempo, y de cada una de ellas se extraen puntos de interés para estimar su posición tridimensional, dando lugar a un mapa de puntos. Por último, se hace uso del mapa para estimar la posición de la cámara.

Esta no es la única aproximación al problema, ya que debido al avance de la tecnología, es posible emplear no sólo sistemas formados por una única cámara (modelos que denominaremos *Monoculares*), sino utilizar sistemas más complejos. Por ejemplo se puede emplear un



Figura 1.3: Reconstrucción 3D mediante *Structure from Motion*.

par estéreo, formado por dos cámaras separadas entre sí a una distancia conocida, aprovechando esta configuración para estimar la información de profundidad de toda la escena; o directamente emplear una cámara *RGBD* que ya aporta la imagen de profundidad, sin necesidad de calcularla. Todos estos modelos y sistemas tienen sus ventajas y desventajas como veremos en la sección 3.

1.2.1. Conceptos

La mayoría de algoritmos de Visual SLAM pueden clasificarse en tres categorías en función de cómo estimen la posición de la cámara. En primer lugar, existen los métodos basados en características. Estas técnicas extraen puntos de interés de las imágenes y determinan el desplazamiento minimizando el error de retro-proyección de los puntos. Por otra parte, los métodos directos emplean los valores de intensidad de los píxeles de la totalidad de la imagen para estimar el desplazamiento minimizando el error fotométrico. Y por último, los métodos híbridos hacen uso de una combinación de los dos métodos anteriores.

Indiferentemente de la clasificación a la cual pertenezca un algoritmo de Visual SLAM, todos ellos tienen asociados una serie de conceptos [11] que merece la pena detallar, dado que se hará uso de ellos a lo largo del proyecto.

Calidad: la calidad del algoritmo dependerá de la eficiencia temporal, la precisión espacial de la pose y la robustez.

Eficiencia temporal: medida como el tiempo de ejecución de cada iteración. Para considerar que el algoritmo es apto para trabajar en tiempo real deberá ser capaz de procesar al menos 30 fotogramas. (*frames*) por segundo.

Precisión de la posición: diferencia entre la pose estimada y la pose real, expresada como el error lineal y angular.

Robustez: entendida como la capacidad de recuperarse o seguir funcionando ante situaciones inesperadas, como occlusiones, imágenes borrosas, objetos dinámicos en la escena, etc.

Oclusiones: situación donde la cámara del sistema esté tapada total o parcialmente, de modo que no sea posible utilizar la totalidad de la imagen para obtener información.

Relocalización: capacidad para recuperarse de una pérdida por falta de información, siendo capaz de volver a estimar la posición de forma correcta dentro del mapa.

Cierre de bucle: capacidad de detectar cuando la cámara vuelve, tras pasar un periodo de tiempo, a una zona del mundo que ya haya visitado con anterioridad. La dificultad de esta tarea radica en cambios en la escala, iluminación, cálculo de la pose de la cámara u otros motivos similares. Por este motivo, Visual SLAM debe ser capaz de reconocer este entorno conocido y modificar su mapa para hacerlo coincidir con esta situación.

1.3. Robótica

Uno de los principales campos de investigación donde la visión artificial tiene una fuerte presencia, como se ha comentado anteriormente, es la robótica. Los robots son sistemas electromecánicos diseñados y programados con el fin de realizar unas determinadas funciones.

Para lograr este fin hacen uso de tres tipos de componentes *hardware*: actuadores, sensores y unidades de procesamiento. Los actuadores y sensores permiten interactuar y obtener información del mundo real, mientras la unidad de procesamiento actúa como el cerebro que analiza los datos proporcionados por los sensores y, en consecuencia, toma las decisiones que deben realizar en cada momento los actuadores.

Existe una gran variedad de robots desarrollados para entornos industriales, ya que tienen la capacidad de realizar el trabajo de una forma muy precisa, siendo un claro ejemplo los brazos robóticos (Figura 1.4a) utilizados en numerosas cadenas de montaje. Sin embargo, también son empleados en otros campos como la educación, defensa, medicina, la ayuda en el hogar e incluso la exploración espacial.

Uno de los sensores más utilizados en robótica son las cámaras, debido a la gran información del entorno que pueden proporcionar. Principalmente, son empleadas en robots móviles autónomos, es decir, aquellos especializados en la navegación sobre un terreno conocido, o desconocido, sin control humano.

Algunos ejemplos de estos tipos de robots pueden ser los robots aspiradora y UAVs mencionados anteriormente, o más recientemente el desarrollo de los coches autónomos. En la conducción autónoma, los vehículos son dotados de numerosos sensores entre los cuales se incluyen cámaras. A partir de las imágenes obtenidas es posible detectar los carriles de circulación, otros vehículos (Figura 1.4b) e incluso peatones.

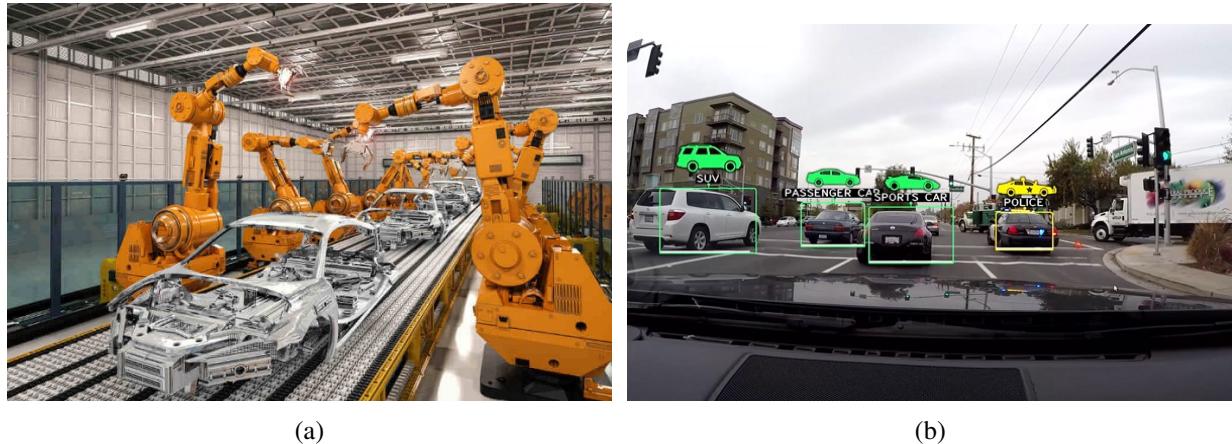


Figura 1.4: Brazo robótico industrial (a). *Software* de conducción autónoma (b).

1.4. Estructura del documento

Una vez introducidas las disciplinas en las cuales se enmarca el desarrollo del proyecto, en los siguientes capítulos se describe el trabajo realizado.

En primer lugar, se realiza un repaso al estado del arte actual de las técnicas de auto-localización en entornos desconocidos, para posteriormente plantear el objetivo del proyecto. Una vez planteados los objetivos, los capítulos 4 y 5 describirán el desarrollo y los resultados obtenidos, incluyendo la metodología empleada para la obtención de los datos. Por último, se expondrán las conclusiones finales del trabajo tras haber analizado los resultados.

Capítulo 2

Objetivos

Una vez realizada una introducción a la temática del proyecto, en este capítulo se exponen los problemas asociados a Monocular SLAM así como los objetivos que se pretenden alcanzar.

2.1. Descripción del problema

Como se ha visto en el capítulo anterior, el algoritmo de Visual SLAM puede ser empleado haciendo uso de diferentes configuraciones de cámaras: sistemas monoculares (una única cámara), sistemas estéreo (dos cámaras) o empleando cámaras de profundidad. Para el desarrollo de este proyecto nos centraremos en los sistemas monoculares, debido a que son el sistema más económico y fácil de implementar si se consigue solventar los problemas que tiene asociados (y aún sin solventar).

Debido al coste computacional, falta de información y ciertas ambigüedades inherentes a los propios sensores de adquisición monoculares, podemos encontrarnos una serie de problemas difíciles de abordar. Los principales problemas en los que se centrará el trabajo son los siguientes:

Ambigüedad en la escala. Uno de los principales problemas está asociado al proceso de construcción del mapa inicial, donde no es posible conocer la distancia real a la cual se encuentran los objetos. Este es un problema inherente de los sistemas monoculares debido a la pérdida de información 3D al momento de ser proyectada sobre el plano imagen (2 dimensiones). Es decir, un objeto en el mundo real de un determinado tamaño a una cierta distancia proyecta sobre el plano imagen la misma información que un objeto del doble de tamaño al doble de distancia. Un ejemplo visual de este problema puede observarse en la Figura 2.1. Debido a esto, los sistemas monoculares de SLAM fijan una escala arbitraria en cada proceso de inicialización.

Modelo de movimiento. El coste computacional de determinar la nueva posición de la cámara

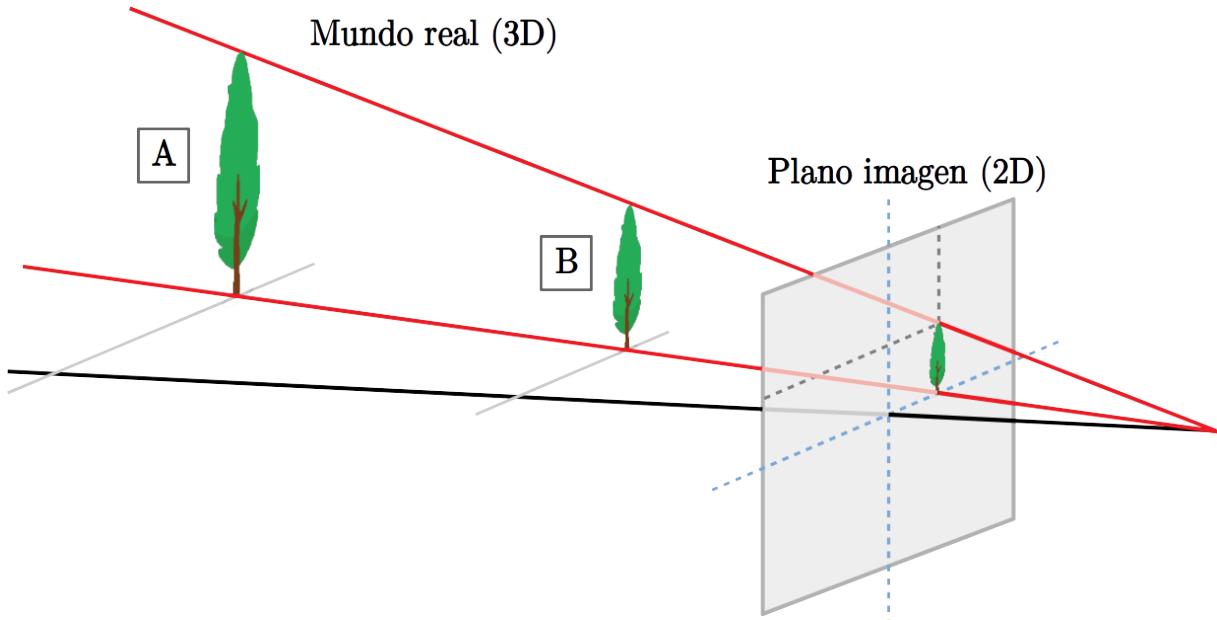


Figura 2.1: Pérdida de información de escala al proyectar objetos del mundo real (3D) al plano imagen (2D). Los árboles A y B tienen distinto tamaño y se encuentran a diferente distancia del sensor pero su proyección sobre el plano imagen es la misma.

en cada instante de tiempo es elevado. En los algoritmos basados en características este proceso suele ser iterativo, basándose en la retro-proyección y emparejamiento de los puntos visibles del mapa sobre cada nueva imagen. Con el objetivo de acelerar este proceso se emplean técnicas como la estimación de la nueva posición mediante modelos de movimiento o alineaciones de imágenes.

Recuperación ante pérdidas. Cuando la calidad visual disminuye, debido a occlusiones, bajas texturas, movimientos bruscos o robos entre otros motivos, provoca que el algoritmo entre en un estado de pérdida del cual no es capaz de recuperarse directamente.

Todos estos problemas son ampliados en la sección ??, donde se profundiza en cada uno de ellos.

2.2. Objetivos

Como se ha expuesto en la sección anterior, aún existen problemas sin resolver en el campo de Visual-SLAM Monocular, la mayoría de ellos relacionados con el desconocimiento de información del mundo real. Este proyecto trata de explorar y aportar soluciones a los problemas expuestos en el apartado anterior mediante la incorporación de sensores que proporcionen información de la realidad. En concreto se hará uso de una unidad medición iniciales o IMU

(introducidos en la sección 4.2). En los últimos años ha crecido el interés por la integración de estos sensores con los principales algoritmos de Visual SLAM [8, 25, 32].

Los objetivos concretos del proyectos, realizados sobre el algoritmo de Visual-SLAM Monocular seleccionado, son los siguientes:

- 1. Procesar y extraer información de sensores inerciales.**
- 2. Integrar y combinar la información visual junto con la proporcionada por los sensores inerciales.**
- 3. Mantener activo el hilo de *Tracking* cuando el algoritmo Visual-SLAM entre en un estado de “pérdida”.**
- 4. Validar experimentalmente las soluciones aportadas con conjuntos de datos públicos internacionales.**

Junto a estos objetivos, se fijan dos únicos requisitos: trabajar en tiempo real y no ver comprometida la robustez inicial del algoritmo.

Capítulo 3

Estado del arte

En este capítulo se explican varios de los algoritmos más significativos que abordan el problema de Visual SLAM empleando una única cámara RGB.

3.1. MonoSLAM

El primer sistema monocular de Visual SLAM fue desarrollado entre los años 2002 y 2007 por Andrew Davison et al. [4, 5, 6] y finalmente fue presentada su versión final bajo el nombre de MonoSLAM. Esta aproximación está basada en un Filtro Extendido de Kalman (en adelante EKF por sus siglas en inglés) cuyo vector estado está formado tanto por la pose de la cámara así como por cada uno de los puntos 3D que conforman el mapa.

MonoSLAM necesita un mínimo 4 puntos cuya posición en el espacio o distancias entre sí son conocidas para realizar el proceso de inicialización. para este fin se suelen emplear plantillas en forma de tablero de ajedrez como en la Figura 3.1. Con esta información tiene una estimación de profundidad y de escala inicial para poder añadir nuevas características.

Por otra parte, el modelo de predicción se basa en un modelo de movimiento de velocidad constante para la estimación la posición y orientación de la cámara. En función del movimiento de la cámara nuevos puntos son detectados mediante el operador de detección de Shi y Tomasi [17] y añadidos al mapa y al vector estado.

El modelo de observación se basa en la proyección de cada uno de los puntos 3D del mapa sobre el plano imagen teniendo en cuenta la pose de la cámara.

El principal problema de este método es que el coste computacional crece proporcionalmente al tamaño del mapa. Debido a este motivo, MonoSLAM únicamente puede trabajar en tiempo real cuando el mapa consta de menos de 100 puntos. En entornos de mayor tamaño, el vector estado del EKF se vuelve muy grande debido al número de puntos contenidos en él.

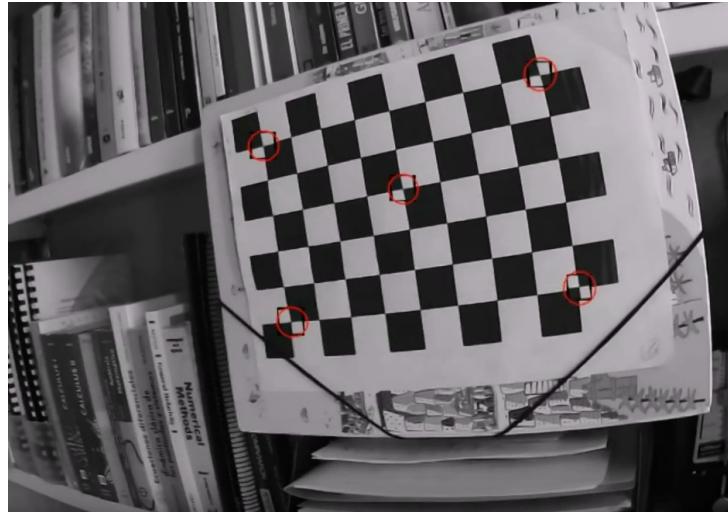


Figura 3.1: Estimación de la posición de espacial de los puntos 3D del mapa en MonoSLAM.

3.2. PTAM

Con el objetivo de solventar el problema computacional de MonoSLAM, Georg Klein y David Murray presentan en el año 2007 el algoritmo *Parallel Tracking and Mapping* [18] más conocido como PTAM por sus siglas.

La idea principal tras este proyecto es la separación del proceso de estimación de la pose de la cámara (*Tracking*) del proceso de generación del mapa (*Mapping*) en 2 hilos asíncronos de ejecución. Esto es posible debido a que solamente es necesario que funcione el proceso de *Tracking* en tiempo real, mientras que el proceso de *Mapping*, que es computacionalmente más costoso, no es necesario que se ejecute en cada iteración. Esta idea hace que sea posible trabajar con mapas de miles de puntos en lugar de cientos como hacía MonoSLAM.

Por otra parte, PTAM añade el concepto de fotograma clave (*KeyFrame* en adelante) para, en lugar de almacenar las características de todos los fotogramas en el mapa, procesar únicamente este subconjunto de fotogramas. Las condiciones para crear un nuevo *KeyFrame* pueden ser temporales (transcurrir un determinado tiempo desde que se creó el último) o espaciales (la distancia respecto al anterior es significativa o la nueva localización es de buena calidad por la cantidad de nueva información que observa).

Por último, el proceso de construcción del mapa inicial del entorno ya no precisa de información a priori, como ocurría en MonoSLAM, sino que haciendo uso de dos fotogramas que tengan un desplazamiento suficiente entre ellos aplicando el algoritmo de cinco puntos [31]. Este desplazamiento tiene que ser espacial, no basta solo con una rotación. Como se comentó en la sección 2.1 el mapa generado representará la realidad en una escala diferente a la real.

Con todos estos elementos el proceso de *Tracking* consta de los siguientes pasos:

1. Detección de características: la imagen se subdivide en distintas resoluciones (aplicando

una pirámide de imagen) y se aplica a cada una de ellas el detector de esquinas FAST [28] para detectar los puntos de interés. Un ejemplo de los puntos detectados puede observarse en la Figura 3.2a.

2. Estimación de la posición: la pose de la cámara se actualiza con un modelo de velocidad constante, al igual que se hacía en MonoSLAM.
3. Actualización de grano grueso: se selecciona un subconjunto de puntos 3D no mayor de 60 y se retro-proyectan sobre la imagen. Seguidamente, se realiza la búsqueda de su homólogo comparando parches de tamaño 8x8 en un amplio rango alrededor de la posición de la proyección. Una vez obtenidos los emparejamientos, se corrige la estimación actual de la cámara minimizando el error de retro-proyección mediante la técnica iterativa de optimización de Gauss-Newton [27].
4. Actualización de grano fino: se realiza el mismo proceso que en el caso anterior, pero en este caso se retro-proyectan hasta un total de 1000 puntos al mismo tiempo que se reduce el rango de búsqueda del homólogo. Finalmente, la pose de la cámara vuelve a ser corregida minimizando el error de retro-proyección dando lugar a la pose final.

Por otra parte, el proceso de *Mapping*, tras ser inicializado, realiza las siguientes operaciones en cada iteración:

1. Creación de *KeyFrames*: se comprueba en cada instante de tiempo si es necesario crear un nuevo *Keyframe* haciendo uso de las condiciones explicadas anteriormente.
2. Añadir puntos al mapa: en el caso de que se haya añadido un nuevo *KeyFrame*, se detectan puntos característicos en la imagen se buscan sus homólogos en los anteriores *KeyFrames*. Una vez realizados los emparejamientos, se calcula la posición 3D de cada punto mediante triangulación. Estos puntos son los que empleará el proceso de *Tracking* para corregir la estimación de la pose.
3. Optimizar el mapa: con el objetivo de refinar la ubicación de los puntos 3D que conforman al mapa se emplea el algoritmo *Boundle Adjustment* [34]. Este es un proceso demandante computacionalmente, por lo que solo se realiza cuando el hilo se encuentra desocupado.

Tras la publicación de PTAM, la mayor parte de los futuros algoritmos de Visual SLAM emplearán esta división del proceso de *Tracking* y *Mapping* en distintos hilos en sus aproximaciones.

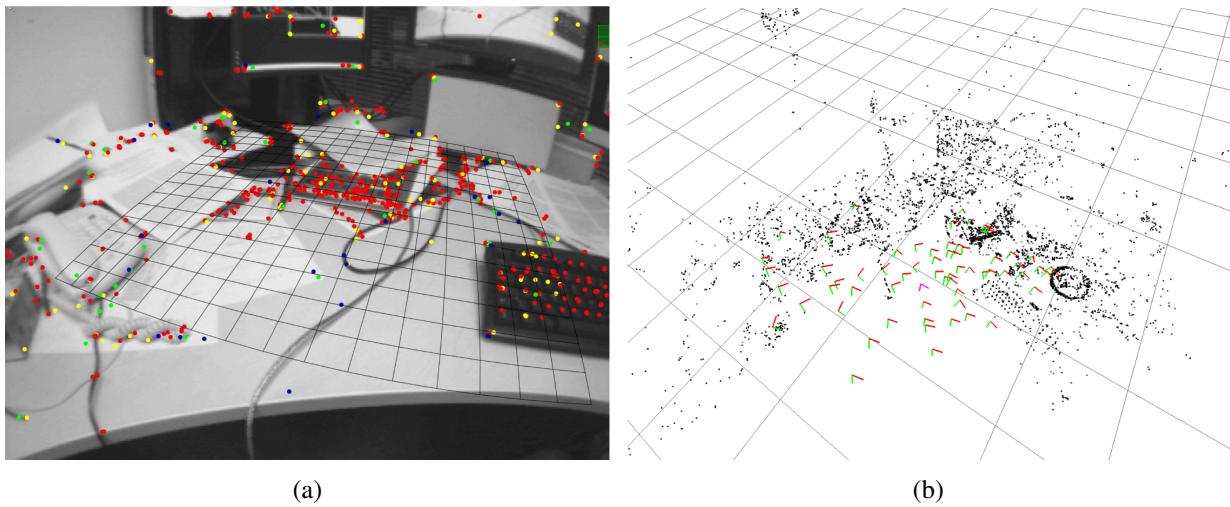


Figura 3.2: Puntos detectados (a) y mapa generado (b) por PTAM.

3.3. SVO

SVO es la abreviatura de *Fast Semi-Direct Monocular Visual Odometry* [9], algoritmo presentado por Foster et al. en el año 2014. Como se explicó en la sección 1.2.1, los algoritmos de Visual SLAM pueden emplear técnicas basadas en características (como MonoSLAM y PTAM) o técnicas directas; SVO hace uso de ambas, convirtiéndolo en un método híbrido. Este algoritmo es similar PTAM dado que separa los procesos de *Tracking* y *Mapping* en dos hilos de ejecución y la inicialización del mapa se realiza mediante homografía.

En el proceso de *Tracking* es donde podemos encontrar el uso del método híbrido para la estimación de la pose de la cámara obtenida como la minimización del error fotométrico entre cada nuevo fotograma y su anterior. Sin embargo, en lugar de calcularlo sobre la totalidad de la imagen se realiza sobre pequeños parches de tamaño 4x4. Estos parches se localizan alrededor de la proyección en el fotograma actual de los puntos del mapa visibles desde el fotograma anterior (Figura 3.3a).

Al combinar de este modo el uso de puntos de interés junto con los valores de intensidad de la imagen se reduce el coste computacional del proceso obteniendo buenos resultados. Tras obtener esta primera estimación de posición se realiza un ajuste más fino empleando parches de mayor tamaño (8x8 píxeles) para terminar de refinar la pose.

Por otra parte, el proceso de *Mapping*, al igual que PTAM, hace uso de *KeyFrames* para la construcción del mapa 3D. Sin embargo, cada nuevo punto detectado no es añadido de instantáneamente al mapa. Esto se debe a que la profundidad a la que se encuentra el punto es modelada como una distribución de probabilidad con una incertidumbre inicial muy alta.

La incertidumbre disminuye a medida que se obtienen nuevas observaciones del dicho punto en diferentes fotogramas (Figura 3.3b). Una vez que la varianza es lo suficientemente baja el

punto es añadido al mapa. Al realizar este proceso se puede tener la certeza que la posición de cada punto que es añadido es precisa, pero al mismo tiempo implica que el mapa constará de menos puntos.

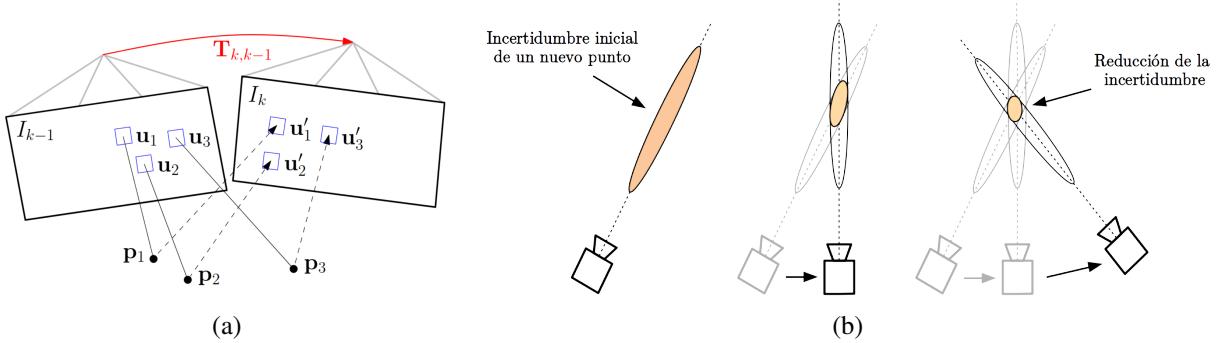


Figura 3.3: Proyección de los puntos del mapa visibles desde el fotograma anterior (a) e incertidumbre en la profundidad de un punto 3D (b) en SVO.

3.4. ORB-SLAM

ORB-SLAM es un algoritmo desarrollado en la Universidad de Zaragoza por Raúl Mur et al. presentado en el año 2015 [23] y mejorado en el año 2017 [24]. Es un método basado en características capaz de ser empleado mediante sistemas monoculares, estéreo o cámaras de profundidad RGB-D. El algoritmo toma este nombre debido a que hace uso de descriptores ORB [29] para la detección y emparejamiento de píxeles de interés.

Una de sus características principales es que, junto a los hilos de ejecución de *Tracking* y *Mapping*, añade un tercer proceso dedicado a la detección de cierres de bucle o *Looping* (explicado en la sección 1.2.1). Haciendo uso de estos tres hilos, el algoritmo puede funcionar en tiempo real siempre y cuando el procesador tenga cierta capacidad de cómputo, dado que, pese a que los descriptores ORB son mas robustos que los empleados por los anteriores algoritmos, requieren de un mayor tiempo de cómputo.

Por otra parte, el proceso de inicialización se realiza empleando el método homografía y, por otra parte, mediante la matriz fundamental haciendo uso del algoritmo de los ocho puntos [15]. Cada uno de los métodos generará un mapa diferente y se determinará de cuál de ellos hacer uso. Si la escena consta de un plano principal se escogerá el mapa obtenido por homografía, y en contrario el obtenido mediante la matriz fundamental. Este proceso permite obtener una mayor robustez en la generación del mapa inicial que será determinante en los futuros procesos.

Tanto el proceso de *Mapping* como el de *Tracking* son muy similares a los implementados en PTAM. El hilo de *Mapping* se encarga de crear *KeyFrames* cuando sea necesario, añadir

nuevos puntos 3D al mapa y optimizar el mismo haciendo uso del algoritmo de *Bundle Adjustment*. Sin embargo, en este caso, el emparejamiento de los puntos 3D se realiza mediante los descriptores ORB. Además, ORB-SLAM construye un grafo de co-visibilidad que relaciona unos *KeyFrames* con otros en función de cuantos puntos 3D son observados de forma común. Esto ayuda, entre otras cosas, a eliminar *KeyFrames* redundantes.

Al igual en los casos anteriores, el proceso de *Tracking* realiza en cada iteración una primera estimación de la posición de la cámara empleando un modelo de velocidad constante y empareja los puntos 3D visibles en el fotograma actual y anterior, calculando la posición final a partir de los emparejamientos obtenidos por triangulación. Además, incorpora un mecanismo para recuperar la posición de la cámara ante pérdidas (occlusiones, baja calidad visual, robos, etc.). Esto se consigue buscando *KeyFrames* cuya información visual concuerden con la del fotograma actual. Por tanto, es necesario que la cámara vuelva a pasar por un lugar previo almacenado en el mapa para poder relocalizarse. Para reducir el coste computacional de la obtención de los posibles *KeyFrames* candidatos se utiliza un modelo de bolsa de palabras [10]. En siguiente figura se puede observar puntos característicos detectados mediante ORB y el mapa reconstruido por ORB-SLAM.

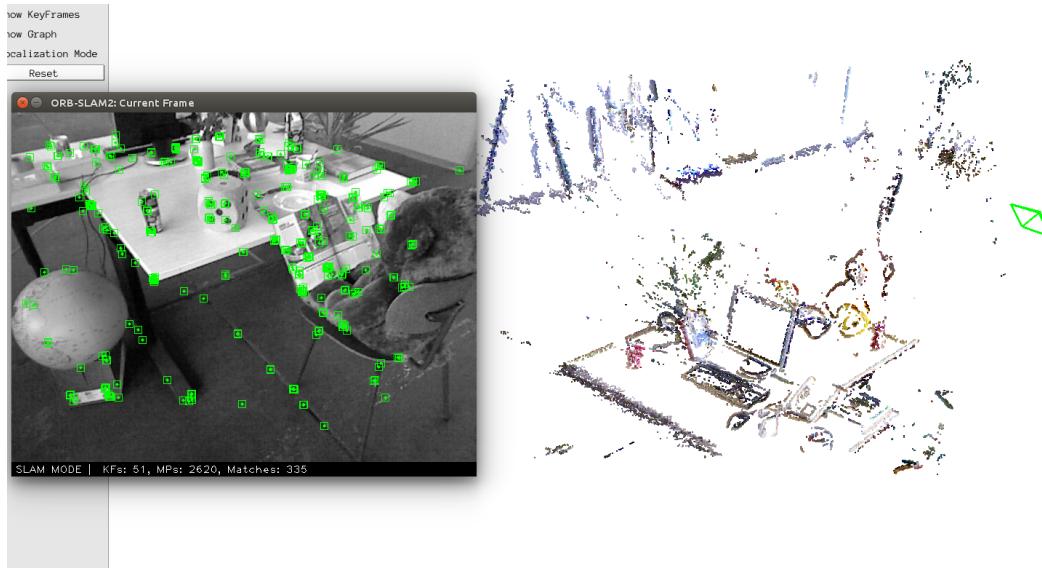


Figura 3.4: A la izquierda se muestra la localización de los puntos característicos detectados con ORB en color verde y el mapa generado a la derecha.

Por último, la función del hilo de *Looping* es la detección de posibles cierres de bucle. Funciona, igual que proceso de relocalización, haciendo uso del modelo de bolsa de palabras junto con el grafo de co-visibilidad. En el caso de detectar un cierre de bucle, se corrige la posición del fotograma actual para hacerla coincidir con el *KeyFrame* en cuestión, se eliminan posibles puntos 3D duplicados, y finalmente se corrige la posición de los *KeyFrames* restantes para hacer coincidir la trayectoria con la nueva posición.

La estructura propuesta por ORB-SLAM que otorgan una gran robustez tanto en entornos pequeños como extensos, sumado al hecho de estar liberado como un proyecto de código abierto¹, ha hecho que sea uno de los algoritmos de Visual SLAM más conocidos y utilizados hoy en día.

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

Capítulo 4

Herramientas

Una vez realizada una introducción a los algoritmos de Visual SLAM más destacados en los últimos años, en este capítulo se describen las principales tecnologías de las que se hace uso durante la realización del presente trabajo de fin de máster. También se describe el conjunto de datos y métricas seleccionadas a la hora de evaluar las soluciones propuestas.

4.1. SD-SLAM

El algoritmo de Visual SLAM seleccionado para el desarrollo del proyecto es SD-SLAM, que es el acrónimo de *Semi-Direct SLAM*, desarrollado en la Universidad Rey Juan Carlos por Eduardo Perdices como parte de su tesis doctoral [11] junto a su director José María Cañas¹ en el año 2017. SD-SLAM es el resultado de mejorar ORB-SLAM transformándolo, de un método basado en características, a uno híbrido debido a la adicción de métodos directos.

Al ser una extensión de ORB-SLAM consta de todas las características que este disponía, como el uso de 3 hilos de procesamiento (*Tracking*, *Mapping* y *Looping*), soporte para sistemas monoculares, estéreo y cámaras de profundidad RGB-D, uso de descriptores ORB y módulo de relocalización. Pero además, añade una serie de nuevas características como son, como se ha comentado antes, el uso de métodos directos en el proceso de estimación de la pose de la cámara y una nueva forma de representar los puntos 3D, como se detallará más adelante.

Donde encontramos la principal diferencia es en el hilo de *Tracking*. En ORB-SLAM este proceso consistía, en primer lugar, en una estimación de la posición basada en un modelo de velocidad constante para posteriormente realizar una corrección basada en el emparejamiento de puntos ORB minimizando el error de retro-proyección. Este último proceso es iterativo y, por tanto, si se proporciona una pose inicial cercana a la real el algoritmo convergerá en un menor tiempo.

¹<https://gsyc.urjc.es/jmplaza/>

Con el objetivo de proporcionar una mejor estimación inicial antes de comenzar el proceso de emparejamiento, SD-SLAM añade un método directo basado en la minimización del error fotométrico, similar al realizado por SVO, que denomina alineamiento de imágenes. Este proceso consisten en la proyección de los puntos 3D del fotograma anterior visibles desde el fotograma actual y construye, para cada uno de ellos, parches de tamaño 4x4 píxeles para ser comparados. Además, mientras que en SVO estos parches se transforman aplicando una matriz afín asociada al movimiento antes de ser comparados, SD-SLAM evita este paso dado que presupone que el desplazamiento entre fotogramas será pequeño y se podrá asumir que no existe una gran variación en las poses y, por consiguiente, en los parches. Finalmente, se minimiza el error fotométrico (problema no lineal de mínimos cuadrados) con el algoritmo de Gauss-Newton [27].

Este proceso de alineación de imágenes se realiza haciendo uso de una pirámide de imágenes para detectar rápidamente el desplazamiento en las imágenes de menor tamaño, para posteriormente, ser refinado a medida que se avanza por la pirámide. El resultado es una segunda estimación de pose muy rápida y cercana a la final, agilizando el proceso final de corrección (emparejamiento y minimización del error de retro-proyección). De este modo se logra una reducción en el tiempo de cómputo respecto a ORB-SLAM.

Los módulos de relocalización y detección de cierres de bucle (*Looping*) también han sido mejorados para hacer uso del alineamiento de imágenes. Estas dos operaciones se basan en la comparación del fotograma actual con un subconjunto de *KeyFrames* potencialmente similares, y por tanto, el tiempo de ejecución es dependiente del número de *KeyFrames* presentes en el mapa, dificultando su ejecución en tiempo real cuando este número crece. Al ser modificadas del mismo modo que se hizo en el proceso de *Tracking* permite reducir el tiempo de cómputo y tolerar así un mayor número de *KeyFrames*.

Por otra parte, el hilo de *Tracking* trabaja tal y como lo hacía en el algoritmo de ORB-SLAM exceptuando el modo en el que se representan los puntos 3D del mapa. La mayoría de algoritmos de Visual SLAM representan los puntos 3D mediante las coordenadas de su posición espacial en el mundo (X, Y, Z) desde un origen arbitrario (normalmente el primer *KeyFrame* del mapa). La posición 3D, como se ha comentado en numerosas ocasiones anteriormente, es calculada mediante triangulación. Este proceso es dependiente de que el ángulo que forman los rayos de retro-proyección sea suficientemente amplio; es decir, el desplazamiento (paralaje) entre las observaciones debe ser suficiente. A medida que este ángulo disminuye, el punto se aleja y su indeterminación crece, provocando potenciales errores. De hecho, existen objetos para los cuales nunca se logrará un desplazamiento suficiente para poder estimar su posición. Por ejemplo, las estrellas se podrían considerar en el “infinito” y, por tanto, no será posible obtener un paralaje suficiente para estimar su profundidad real.

SD-SLAM hace uso de la inversa de la profundidad [3] para representar los puntos 3D.

Además, la posición de estos puntos se representa como una matriz de transformación (rotación y traslación) respecto de la cámara donde fueron detectados por primera vez. De este modo, los puntos 3D están referenciados a los distintos *KeyFrames* donde se detectaron por primera vez en lugar de a un origen arbitrario. La ventaja de esta representación en comparación a la parametrización clásica es que es posible representar puntos en el infinito, dado que la inversa de la profundidad en el infinito es 0.

Una vez descritas las principales características de SD-SLAM y tras haber realizado un análisis de algunos de los principales algoritmos de Visual SLAM en el capítulo anterior, se ha decidido hacer uso de este para el desarrollo del proyecto. Los principales motivos para la elección de este algoritmo son los siguientes:

- Código abierto².
- Soporta sistemas monoculares.
- Capacidad de trabajar en tiempo real.
- Inicialización robusta (homografía y matriz fundamental).
- Incorpora un módulo para detectar cierres de bucle.
- Emplea hilos asíncronos para los módulos de *Tracking*, *Mapping* y *Looping*.
- Es capaz de representar puntos en el infinito.
- Por último, pero no menos importante, fue desarrollado en la URJC.

4.2. Unidad de Medición Inercial (IMU)

Una Unidad de Medición Inercial (IMU por sus siglas en inglés) es un dispositivo electrónico formado por distintos sensores inerciales. Típicamente constan de un acelerómetro y un giroscopio, aunque también pueden incluir un magnetómetro para ser más precisos. La IMU es capaz de proporcionar en tiempo real información de aceleración lineal y velocidad angular del sistema. Suele ser uno de los componentes principales de los sistemas de navegación inercial empleados en aparatos como aviones, submarinos, misiles o naves espaciales entre otros. Si bien las IMUs más potentes son lo suficientemente precisas como para ser utilizadas para navegación a largas distancias, las más económicas y usadas típicamente en robots y teléfonos móviles, debido a la baja calidad de los datos que proporcionan, no son suficientemente robustas como para hacer uso únicamente de ellas como sistema de navegación.

²<https://github.com/JdeRobot/SDslam>

El acelerómetro es un sensor electrónico capaz de medir las fuerzas de aceleración que actúan sobre él. Normalmente están formados por una pieza capaz de desplazarse cuando se aplican fuerzas causadas por vibraciones o un cambio de movimiento (aceleración), provocando que esta masa se desplace generando una carga eléctrica que es proporcional a la fuerza que se ejerce sobre él (Figura 4.1). La aceleración proporcionada no es lineal, ya que un acelerómetro en reposo, al detectar las fuerzas que actúan sobre él, siente el tirón de la gravedad a menos que se encuentre en caída libre.

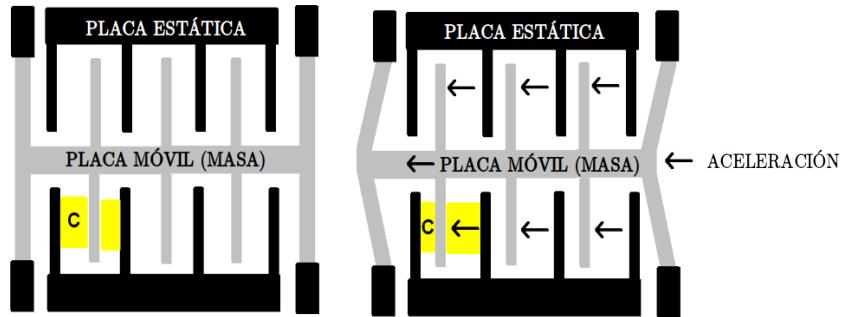


Figura 4.1: Funcionamiento de un acelerómetro.

El giroscopio es un sensor electrónico capaz de detectar las velocidades angulares. Consta de uno o varios brazos en constante vibración en un mismo sentido. Al mover este sensor, la velocidad angular que recibe el giroscopio incide sobre estos brazos alterando el sentido de la vibración. Estas variaciones son detectadas por los sensores alojados en el giroscopio y convertidas a impulsos eléctricos. Esta velocidad puede ser integrada para obtener la orientación del dispositivo.

Como se ha comentado anteriormente, una de las operaciones principales de Visual SLAM es el cálculo de la pose de la cámara, por tanto, este dispositivo tiene un gran potencial para complementar a la información visual en el cálculo de esta operación.

4.3. Conjunto de datos para evaluación

El conjunto de datos para evaluar las soluciones implementadas debe de constar de una serie de elementos imprescindibles, tales como un sistema de cámaras para proporcionar información visual, un sensor inercial y una secuencia de referencia (*Groundtruth*) para comparar con la estimación realizada por SD-SLAM.

El conjunto de datos seleccionado es el “benchmark suite” KITTI [13, 12] desarrollado de forma conjunta por el Instituto Tecnológico de Karlsruhe y el Instituto Tecnológico de Toyota. En comparación con otros conjuntos de datos, KITTI ofrece datos más realistas puesto que las secuencias están grabadas en un entorno real en lugar de ser realizados en el interior de un

laboratorio.

Las secuencias están divididas en distintas categorías, siendo la de odometría la que proporciona todas las características que necesitamos. En esta categoría los datos son tomados desde un vehículo dotado de un par estéreo de cámaras, un sensor LIDAR (del que no se hace uso en el desarrollo de este proyecto) y un sistema de navegación GPS (OXTS RT 3003) que contiene un sensor inercial. En la Figura 4.2 se puede observar la disposición de los distintos sensores. Las imágenes son tomadas a una resolución de 1392x512 píxeles a una frecuencia de 10Hz. Lamentablemente no se dispone de las especificaciones del sensor inercial, ya que este sirve de apoyo al sistema principal GPS y, por tanto, los datos proporcionados por el fabricante hacen referencia a la precisión de este último. Además de los anteriores datos, KITTI proporciona los parámetros intrínsecos y extrínsecos de cada cámara, así como la matriz de transformación rígida entre los distintos sistemas de referencia de cada sensor. Por último, mencionar que las secuencias de las que se hace uso se encuentran sincronizadas y, por tanto, únicamente dispondremos de una medida del sensor inercial por cada fotograma, pese a que este funcione a más Hz que la cámara.

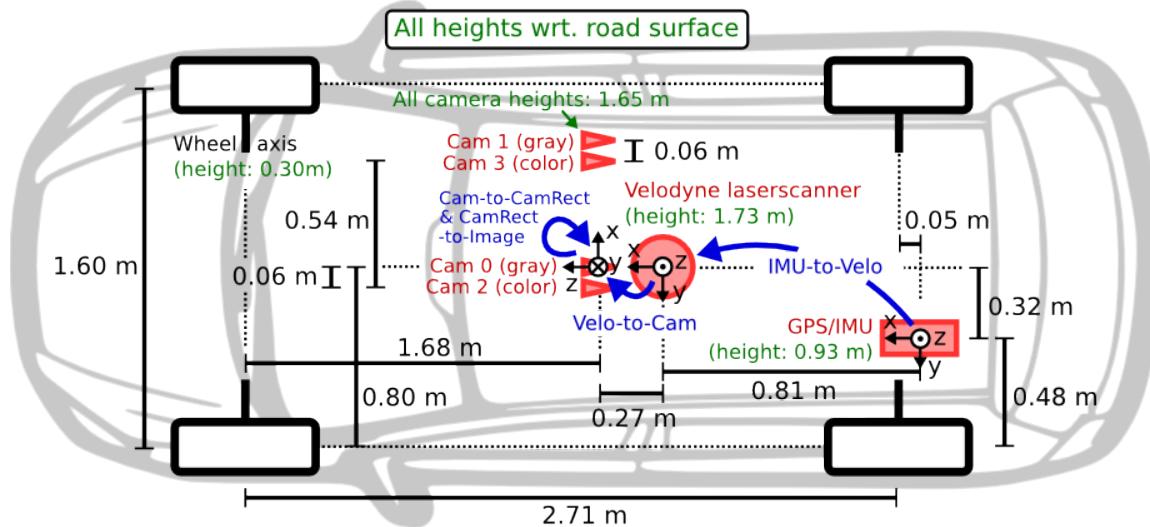


Figura 4.2: Configuración sensorial del vehículo de KITTI.

El conjunto de datos consta de 9 secuencias donde la mayoría de ellas se desarrollan en un entorno residencial. Esto quiere decir que consta elementos estáticos como edificios, vegetación o vehículos estacionados a cada lado de la calzada; y elementos dinámicos como vehículos en circulación o peatones. Debido a la larga duración de la mayoría de las secuencias se han seleccionado un subconjunto que permita cubrir una variedad de escenarios. A continuación se describen muy brevemente las secuencias seleccionadas:

- **Secuencia 00.** Secuencia con una duración de 7:50 minutos y un total de 4 cierres de bucle. Permitirá evaluar la robustez del algoritmo en un largo periodo de tiempo.

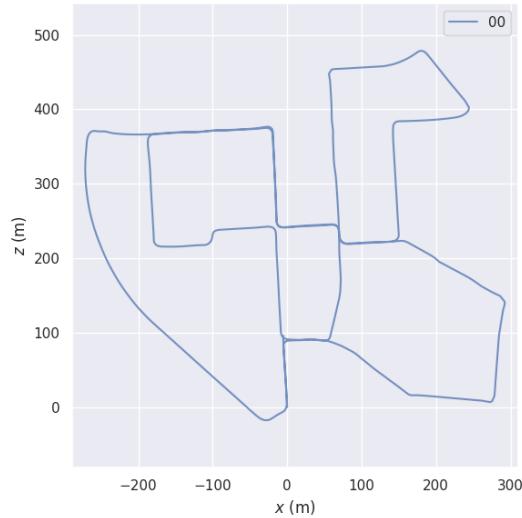


Figura 4.3: Secuencia 00 perteneciente al conjunto de datos Kitti Odometría.

- **Secuencia 05.** Similar al caso anterior, esta secuencia consta de una duración de 5 minutos con un total de 2 cierres de bucle.

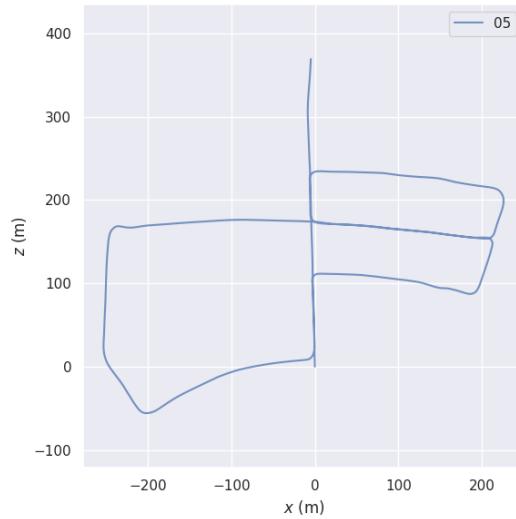


Figura 4.4: Secuencia 05 perteneciente al conjunto de datos Kitti Odometría.

- **Secuencia 06.** Trayectoria sencilla que consta de 2 rectas y dos curvas cerradas de 90° que se toman a gran velocidad. Tiene una duración de 1:30 minutos y un único cierre de bucle.

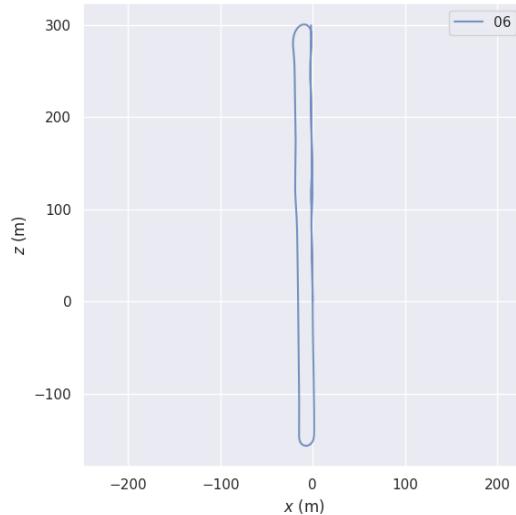


Figura 4.5: Secuencia 06 perteneciente al conjunto de datos Kitti Odometría.

- **Secuencia 07.** Secuencia con una duración de que 1:30 minutos. Consta de un cierre de bucle que une el principio y el final de toda la trayectoria y que, en la mayoría de ocasiones, no es detectado por SD-SLAM.

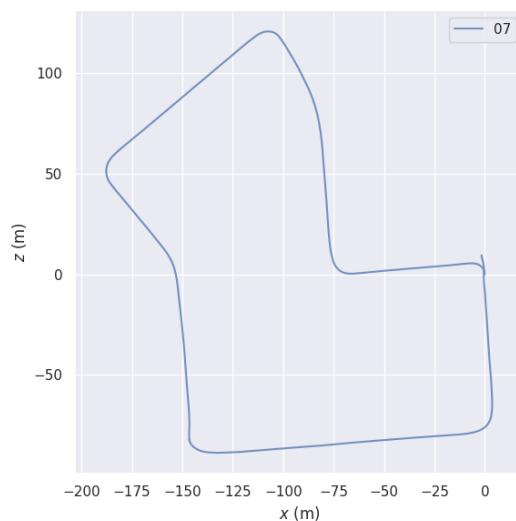


Figura 4.6: Secuencia 07 perteneciente al conjunto de datos Kitti Odometría.

4.4. Métricas de evaluación

Existen diferentes métricas para evaluar la calidad de los algoritmos de SLAM, algunas centradas en la calidad del mapa generado y otras en la trayectoria seguida por la cámara. En nuestro caso nos centraremos en estas últimas, concretamente en el error absoluto (ATE) y relativo (RPE) de las trayectorias [33]. Estas métricas requieren de una secuencia de referencia junto a la que se pretende evaluar.

El error absoluto representa la consistencia global de una trayectoria. Es calculada como la distancia geométrica entre cada una de las posiciones de ambas trayectorias. En el cálculo del error absoluto no se emplea la orientación dado que se asume que el error producido en esta se verá reflejado en la información de la posición espacial.

Normalmente no es posible comparar directamente las secuencias, dado que cada secuencia consta de sus propios sellos temporales y sistemas de referencia. Por tanto, en primero lugar, es necesario alinearlas temporalmente para obtener los pares de posiciones a evaluar. Posteriormente, en caso de ser necesario, se realiza una transformación rígida para corregir la diferencia en el sistema de referencia aplicando una rotación y traslación; esta transformación suele representarse como SE(3). Además, en los sistemas monoculares, dado que su escala es arbitraria, es necesario realizar una corrección en esta mediante una matriz de similitud; representada como Sim(3).

El error relativo, en lugar de realizar una comparación directa entre las posiciones espaciales, compara el movimiento (incremento) realizado de una posición a la siguiente. Esta métrica otorga información sobre la precisión local, siendo especialmente útil para calcular la derivación espacial o una estimación del error en orientación. En Visual SLAM este error suele ser calculado como el movimiento entre imágenes consecutivas.

Para la obtención de estos valores en los experimentos se empleará la herramienta EVO [14], que proporciona una serie de ejecutables para manipular, evaluar y comparar la trayectoria obtenida por odometría y algoritmos de SLAM. Es capaz de estimar las transformación SE(3) y Sim(3) para alinear las trayectorias empleando el algoritmo de Umeyama [35].

Capítulo 5

Diseño e implementación

En este capítulo se describen la implementación de realizada para obtener la odometría inercial y su integración en SD-SLAM, dotándolo de nuevas características que no contenía previamente.

5.1. Introducción

La unidad de medición inercial (IMU) incorpora un acelerómetro y un giróscopo que permite medir las fuerzas de aceleración y la velocidad angular en las tres dimensiones. La idea principal es emplear estos valores para obtener una estimación de posición y orientación que complementen a las estimaciones de pose realizadas por el módulo de *Tracking*. El funcionamiento original en SD-SLAM de este módulo puede ser representado por una máquina de estados finitos, como puede observarse en la Figura 5.1.

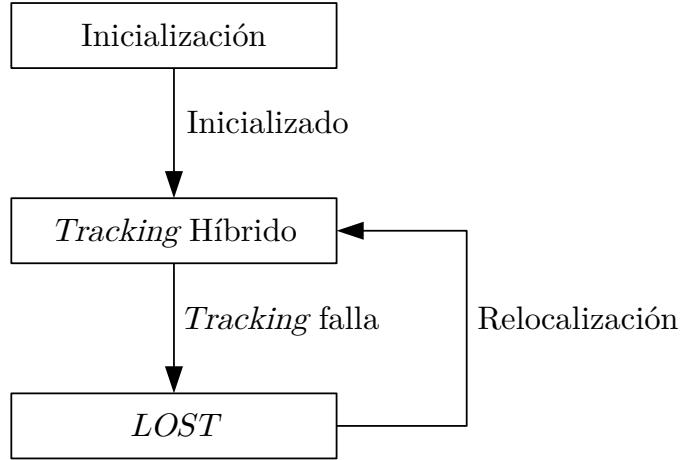


Figura 5.1: Representación del módulo de *Tracking* de SD-SLAM como una máquina de estados finitos.

En un primer momento, el algoritmo emplea un módulo de inicialización encargado de construir el mapa inicial de puntos 3D a partir de dos fotogramas. La posición de estos puntos y las poses de la cámara para ambos fotogramas es obtenida por triangulación, haciendo uso de al menos 100 características comunes detectadas por ORB. Este proceso puede realizarse mediante Homografía o Matriz Fundamental, seleccionando el modelo que menor error de retro-proyección obtenga. Debido al desconocimiento de la información de profundidad, la escala del mapa inicial es arbitraria y varía en cada ejecución.

Una vez el sistema ha sido inicializado transita al estado ***Tracking Híbrido*** encargado de realizar la estimación de la pose de la cámara cada vez que se recibe un nuevo fotograma. Este proceso se divide en varias fases, realizando en primer lugar una estimación basada en un modelo de velocidad constante, para posteriormente refinarla con la información visual aplicando el sistema híbrido de SD-SLAM (alineación de imágenes seguido de la reducción del error de retro-proyección con las características detectadas por ORB). Este proceso es satisfactorio siempre y cuando se detecten al menos 20 emparejamientos en el proceso de retro-proyección. En el caso de que este número descienda, SD-SLAM considera que la calidad visual se ha deteriorado provocando que la estimación de pose sea errónea y el algoritmo entrará en el estado de pérdida.

Durante el estado de pérdida o ***LOST***, el algoritmo tratará de relocalizarse comparando cada nuevo fotograma con aquellos *KeyFrames* visualmente similares. Cuando se detecten al menos 20 puntos comunes se considera que la cámara se encuentra en un lugar previamente visitado. Si esto llega a ocurrir, el sistema volvería a transitar al estado anterior.

Una vez resumidos los estados y módulos originales del módulo de *Tracking* de SD-SLAM,

se introducen las características introducidas en este proyecto. Las modificaciones realizadas son las siguientes:

- Se estimará una odometría inercial a partir de la información proporcionada por la IMU que complementará la odometría visual. Las poses finales obtenidas en el proceso de *Tracking Híbrido* al final de cada iteración servirán de retro-alimentación del modelo inercial para corregir las estimaciones.
- Debido a que la escala obtenida por SD-SLAM en el proceso de inicialización es arbitraria, se ampliará este módulo para que ademá estime un factor de escala λ que relacione tamaño del mundo de real con el mundo de SD-SLAM. Este es un parámetro necesario para poder hacer uso de las estimaciones iniciales.
- Mientras el algoritmo se encuentre en un estado de pérdida, se seguirá estimando la pose de la cámara mediante la odometría inercial. Esto dará lugar a un nuevo estado llamado ***Tracking Inercial***.
- Para transitar desde el nuevo estado a la odometría original de SD-SLAM, es necesario construir un nuevo mapa. Este proceso no era posible en el algoritmo original debido a la incapacidad de relacionar la posición de un nuevo mapa respecto al anterior. Con la incorporación de la odometría inercial este proceso se vuelve plausible.

Al aplicar todas las nuevas características sobre SD-SLAM, el diagrama estados original y condiciones para transitar de uno a otro será modificado como se puede observar en la Figura 5.2.

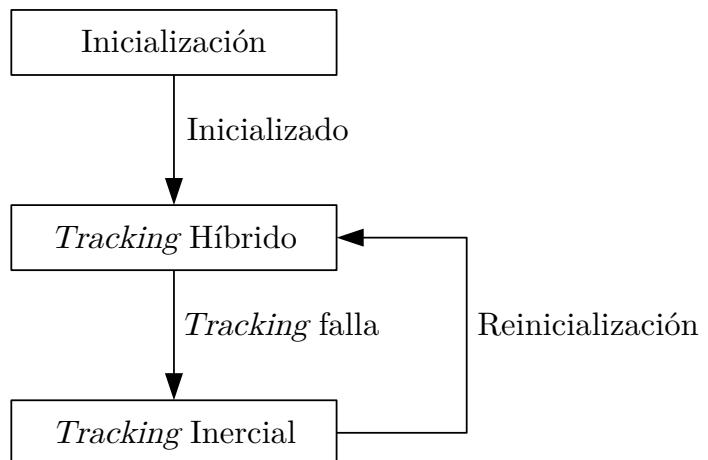


Figura 5.2: Representación del módulo de *Tracking* de SDIO-SLAM como una máquina de estados finita.

5.2. Sistemas de referencia

Antes de comenzar con las ampliaciones realizadas sobre SD-SLAM es importante definir los 3 sistemas de referencia con los que trabajaremos: Cámara, Mundo e IMU; a partir de ahora nos referiremos a ellos con los superíndices C , W y I , respectivamente.

En primer lugar, el sistema de referencia de Cámara representa la posición del mundo respecto de la pose de la cámara, es decir, el origen de coordenadas es la propia cámara. Todas las poses son expresadas mediante una matriz de coordenadas homogéneas de tamaño 4x4, formada por la orientación (matriz de rotación R de tamaño 3x3) y la posición (vector de traslación t 3x1). A diferencia de otros algoritmos de Visual SLAM que representan cada nueva pose como el desplazamiento relativo llevado a cabo respecto de la pose anterior, en SD-SLAM cada pose es independiente.

Por otra parte, el sistema de referencia de Mundo traslada el origen de coordenadas desde la cámara al origen del mundo. De este modo la pose de la cámara en cada momento es expresada como la traslación y rotación realizada desde el origen, y no viceversa como se hacía en el caso anterior. Es por este motivo que las poses de cámara (P^C) y mundo (P^W) se relacionan entre sí mediante su inversa:

$$P^W = (P^C)^{-1} \quad (5.1)$$

donde la inversa es definida como:

$$\begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix}^{-1} = \begin{pmatrix} R^T & -R^T t \\ 0^T & 1 \end{pmatrix} \quad (5.2)$$

La dirección y sentido de los ejes de la cámara representada en SD-SLAM Mundo emplea la convención EDN habitual de las cámaras, donde las coordenadas X, Y, Z representan las direcciones Este (East), Abajo (Down) y Norte (North).

Por último, las medidas iniciales se enmarcan en el sistema de referencia de la IMU. Este sistema varía en función del fabricante, aunque se suele emplear la convención ENU [2], donde las coordenadas X, Y, Z son asignadas las direcciones Este (East), Norte (North), y Arriba (Up). Indiferentemente de la convención empleada, los sistemas de referencia de la IMU y SD-SLAM Mundo pueden ser relacionados mediante una matriz de rotación que denominaremos R^{IW} .

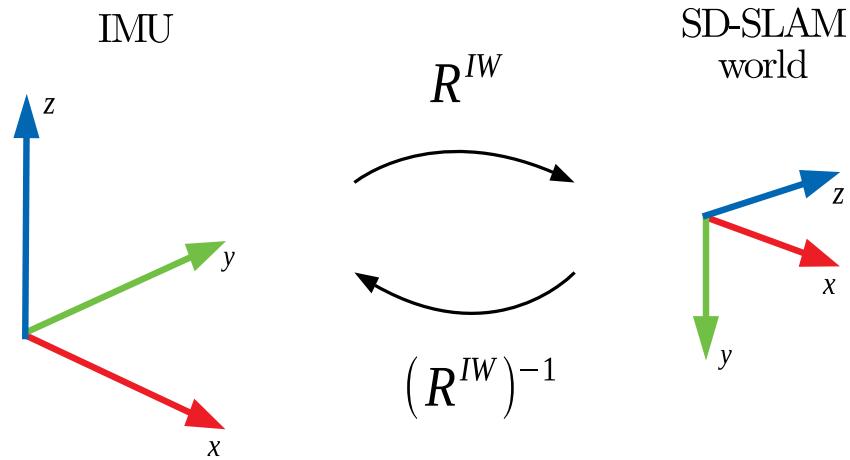


Figura 5.3: Representación de la transformación del sistema de referencia de la IMU (izquierda) al sistema de referencia de SD-SLAM Mundo (derecha).

5.3. Odometría inercial

La odometría inercial es el proceso de estimación de la posición y orientación a partir de la información proporcionada por la IMU. Estas poses servirán para complementar las estimaciones realizadas por el módulo de *Tracking*.

En primer lugar es necesario entender la naturaleza de estos valores obtenidos de los sensores iniciales. Las medidas del sensor inercial suelen ser bastante ruidosas y se suelen modelar [36] descomponiendo este error en dos tipos: uno que fluctúa rápidamente (ruido aleatorio), y otro que varía lentamente (*bias*) pues suele ser dependiente de factores como la temperatura del sensor. De este modo, las medidas pueden ser expresadas del siguiente modo:

$$\tilde{a}(t) = a(t) + b_a(t) + \eta_a(t) \quad (5.3)$$

$$\tilde{\omega}(t) = \omega(t) + b_g(t) + \eta_g(t) \quad (5.4)$$

donde:

- $\tilde{a}(t)$ es la medida de aceleración proporcionada por el acelerómetro y $a(t)$ la aceleración real a la cual es sometida, respectivamente, en el instante de tiempo t . Los valores son expresados en m/s^2 .
- $\tilde{\omega}(t)$ es la medida de velocidad angular proporcionada por el giroscopio y $\omega(t)$ la velocidad angular real a la cual es sometida el sensor, respectivamente, en el instante de tiempo t . Los valores son expresados en rad/s .

- $b_a(t)$ y $b_g(t)$ es el sesgo o *bias* que afecta al acelerómetro y giroscopio. Puede ser considerado constante pues varía muy lentamente en el tiempo.
- $\eta_a(t)$ y $\eta_g(t)$ es el ruido aleatorio que afecta al acelerómetro y al giroscopio. Normalmente es modelado como un ruido blanco gaussiano de media cero.

Además, el vector de aceleración $\tilde{\mathbf{a}}$ está sujeto a la fuerza de la gravedad constantemente. Esto implica que un acelerómetro en reposo, a menos que se encuentre en la estación internacional espacial, siempre va a estar sometido a una aceleración de 9.8 m/s^2 . Esta fuerza es repartida entre las 3 componentes de la aceleración en función de la orientación del sensor, por tanto, no es recomendable estimar la posición del sensor a partir de la medición del acelerómetro sin eliminar primero esta componente.

5.3.1. Estimación inercial

Una vez introducida la naturaleza de los datos que obtenemos de la IMU, el proceso que seguiremos para obtener la pose (posición y orientación) del sensor puede expresarse mediante el siguiente diagrama:

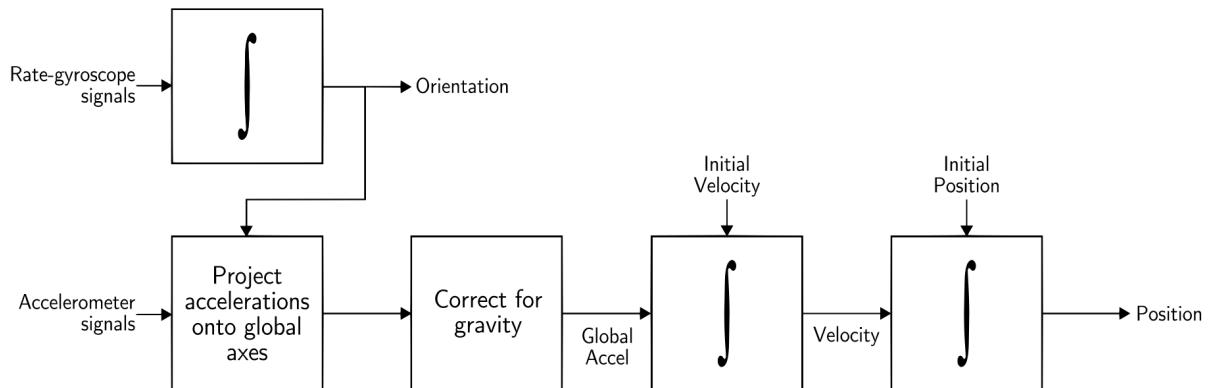


Figura 5.4: Proceso para la estimación de la posición y orientación del sensor IMU.

donde, en primer lugar, se procesa la información del giroscopio para obtener una estimación de la orientación del sensor. Posteriormente, se hace uso de esta orientación para rotar el vector de gravedad y así poder sustraer su efecto sobre las señales del acelerómetro, obteniendo así una aceleración lineal. Esta aceleración será procesada para obtener finalmente una estimación de la posición.

En la actualidad existen diversos algoritmos para obtener la orientación a partir de IMUs, como el uso de Filtros extendidos de Kalman (EKF), el algoritmo de Filtro Complementario [7] o el algoritmo de Madgwick [21]. Este último ha sido el seleccionado para este proyecto. El algoritmo de Madgwick fusiona los datos del giroscopio, acelerómetro y, opcionalmente,

magnetómetro, para estimar la orientación del sensor. Fue desarrollado en el año 2011 por Sebastian Madgwick.

En cada iteración, el algoritmo realiza una nueva estimación de orientación teniendo en cuenta la orientación anterior, el intervalo de tiempo transcurrido, las medidas de la IMU y un parámetro β . Este último se define como la ganancia del filtro y sirve para controlar la variación de la orientación de un instante de tiempo a otro. Así, unos valores bajos de β filtrarán gran parte del ruido generando transiciones suaves, mientras que valores más altos provocarán una respuesta más rápidas y ruidosas.

Por otra parte, para realizar la estimación de la posición se integra la aceleración aplicando las ecuaciones del movimiento, asumiendo que la aceleración es constante entre cada medición del acelerómetro. En primer lugar, se elimina la componente de la gravedad de la señal del acelerómetro para aislar la aceleración lineal:

$$a_t = \tilde{a}_t - R_t^S \cdot g \quad (5.5)$$

donde a_t es la aceleración lineal en el instante de tiempo t , \tilde{a}_t es la aceleración medida por el acelerómetro, R_t^I es la estimación de la orientación del sensor obtenida en el paso anterior, y g es el vector de gravedad que afecta al acelerómetro en reposo.

Una vez se dispone la aceleración lineal se estima la velocidad y, por último, la posición:

$$v_t = v_{t-\Delta t} + a_t \Delta t \quad (5.6)$$

$$s_t = s_{t-\Delta t} + v_{t-\Delta t} \Delta t + \frac{1}{2} a_t \Delta t^2 \quad (5.7)$$

donde Δt es el intervalo de tiempo transcurrido entre el instante actual t y el anterior, v_t y $v_{t-\Delta t}$ son la velocidad lineal en el instante actual y el anterior respectivamente, s_t y $s_{t-\Delta t}$ es la posición en el instante actual y el anterior respectivamente.

El resultado de este proceso es la obtención de una posición (\hat{s}_t^I) y orientación (\hat{R}_t^I) estimadas en el sistema de referencia de la IMU. Para hacer uso de ellas en SD-SLAM es necesario transformarlas a las coordenadas de Mundo. Por una parte, la orientación únicamente requiere de aplicar la matriz de transformación que relaciona ambos sistemas:

$$\hat{R}_t^W = R^{IW} \hat{R}_t^I (R^{IW})^{-1} \quad (5.8)$$

Sin embargo, la posición requiere de un escalado debido a que SD-SLAM inicializa en una escala arbitraria. Debido a este motivo, la transformación a coordenadas de mundo se realizará incrementalmente. Es decir, se define Δs_t^I como el desplazamiento realizado en el sistema inercial realizado entre los instantes de tiempo t y $t-1$ como:

$$\Delta s_t^I = \hat{s}_t^I - s_{t-1}^I \quad (5.9)$$

siendo \hat{s}_t^I la estimación realizada y s_{t-1}^I la posición en el instante de tiempo anterior. Se sigue que es posible calcular la posición en las coordenadas de mundo de la siguiente manera:

$$\hat{s}_t^W = s_{t-1}^W + R^{IW}(\lambda \Delta s_t^I) \quad (5.10)$$

donde \hat{s}_t^W es la estimación de posición inercial en coordenadas de mundo para el fotograma actual, s_{t-1}^W es la posición del fotograma anterior en coordenadas de mundo y λ el factor de escala.

Esta aproximación no es perfecta principalmente por dos motivos. En primer lugar, es dependiente de una buena estimación de orientación. Un error en esta estimación provoca una incorrecta proyección del vector de gravedad, causando que la aceleración sea integrada en una dirección errónea [16]. En segundo lugar, la estimación de la posición es obtenida a partir de una aceleración a la cual no se ha eliminado el ruido. Teniendo en cuenta estas dos consideraciones obtenemos que las estimaciones de pose tenderán a degradarse en el tiempo. Por este motivo el modelo inercial es corregido tras cada iteración visual.

El módulo de *Tracking* devuelve una pose final obtenida tras el proceso visual (reducción del error de retropoyección). Podemos entonces corregir las estimaciones realizadas por nuestro modelo. El desplazamiento Δs^W realizado entre la nueva pose y la anterior obtenidas por visión, expresado en el sistema de referencia de la IMU, es calculado como:

$$\Delta s^I = (R^{IW})^{-1} \frac{1}{\lambda} \Delta s^W \quad (5.11)$$

Con este valor se actualiza la posición estimada y con ella la velocidad:

$$s_t^I = s_{t-1}^I + \Delta s^I \quad (5.12)$$

$$v_t^I = \frac{s_t^I - s_{t-1}^I}{\Delta t} \quad (5.13)$$

Para aplicar la corrección a la orientación inercial se actualiza el valor de la misma en el filtro Madgwick, en coordenadas de IMU.

5.3.2. Estimación y actualización de la escala

Con el objetivo de hacer uso de las estimaciones inerciales en SD-SLAM Monocular es necesario estimar un factor de escala λ que relacione del mundo real con la escala de la escena visual. El proceso de inicializar los parámetros que relacionan ambos sistemas en un modelo Visual-Inercial determinará en gran medida su funcionamiento. En la literatura se ha abordado este problema con diferentes enfoques, haciendo uso de información mutua empleando Filtros de Kalman [22], planteándolo como un problema de optimización de grafos [25] o tratando de forma independiente el módulo de IMU y Visual SLAM [20]. Los sistemas Visuales-Inerciales

de SLAM no han alcanzado aún los niveles de robustez requeridos en muchos casos debido, en gran parte, a los problemas y limitaciones que presenta esta etapa.

En este proyecto, el proceso de inicialización tratarán de forma independiente ambos sistemas hasta que el algoritmo construya el mapa inicial, momento en el cual se estimará el factor escala inicial. Durante este periodo, el módulo de odometría inercial trabajará de forma paralela estimando la pose de cada nuevo fotograma que reciba SD-SLAM. La escala se calcula en el momento de creación del mapa inicial ya que es cuando se calculan las dos primeras estimaciones de pose visuales, almacenadas en los dos primeros *KeyFrames* del mapa.

Haciendo uso estos *KeyFrames*, que denominaremos 0 y 1, la ecuación 5.14 define el factor de escala λ como la relación entre las magnitudes del desplazamiento espacial realizado por el modelo inercial respecto al estimado por visión. Los *KeyFrames* son los encargados de almacenar los factores de escala y proporcionárselos al modelo de movimiento inercial.

$$\lambda = \frac{\| s_{V,1}^W - s_{V,0}^W \|}{\| s_{I,1}^I - s_{I,0}^I \|} \quad (5.14)$$

donde $s_{V,0}^W$ y $s_{V,1}^W$ son la posición estimada por Visión para los dos primeros *KeyFrames* en coordenadas de mundo, y $s_{I,0}^I$ y $s_{I,1}^I$ son la posición estimada por la odometría inercial para los mismos fotogramas en el sistema de referencia de la IMU. Para la realización de este proceso no es necesario aplicar una transformación entre los sistemas de coordenadas, dado que únicamente se hace uso de la magnitud del vector, indiferentemente de su posición espacial u orientación. Este factor de escala variará lentamente en el tiempo debido a la deriva que sufren los sistemas monoculares, por lo que es conveniente actualizarlo para ajustarse a estas. Cada vez que el algoritmo procesa un fotograma, se calcula un nuevo factor λ_i que es almacenado en un *buffer* hasta a creación de un nuevo *KeyFrame*. Al momento de actualizar el factor antiguo se ha optado por añadir un parámetro α que controle la velocidad de actualización entre el antiguo y el nuevo. La nueva escala se calcula como una ponderación entre la antigua y la escala media de los factores asociados a cada fotograma procesado desde el anterior *KeyFrame* almacenados en el *buffer*:

$$\lambda_1 = (1 - \alpha)\lambda_0 + \alpha \frac{1}{n} \sum_{i=0}^n \lambda_i \quad (5.15)$$

donde λ_0 representa el factor de escala actual, λ_1 es el nuevo factor de escala estimado, n es el número fotogramas procesados entre dos *KeyFrames*, λ_i son los factores de escala estimados para cada fotograma y α es un parámetro que toma valores comprendidos en el intervalo $[0, 1]$.

5.4. Odometría durante el estado de pérdida

En un sistema monocular, mientras este se encuentra en el estado de *pérdida*, no puede realizar ninguna de las tareas habituales del proceso de *Tracking* ni *Mapping*, debido a que ambas depende enteramente de la información visual. Con la incorporación del sensor inercial, podemos hacer uso de la odometría generada por la IMU mientras el algoritmo se encuentre en el estado de pérdida para seguir estimando la trayectoria de la cámara. La idea es permanecer en este estado hasta que la calidad visual vuelva a ser suficiente como para transitar al estado de *Tracking Híbrido*.

Durante este estado, se estima la pose de cada nuevo fotograma que recibe el algoritmo de SD-SLAM con el modelo de movimiento inercial. Debido a que esta estimación no será refinada posteriormente por la información visual junto con el mapa almacenado, los procesos de corrección la estimación inercial no se realizarán. Del mismo modo, la escala entre el sistema de referencia inercial y SD-SLAM mundo se mantendrá constante haciendo uso de la última obtenida.

Sin embargo, es necesario solventar el problema de que SD-SLAM únicamente almacena la pose de los fotogramas que considera relevantes (*KeyFrames*) y, durante el estado de pérdida, no se crea ninguno. Esto es totalmente normal ya que, a priori, no tiene sentido crearlos en un estado al cual se ha llegado por falta de información visual. Sin embargo, al realizar el seguimiento de la trayectoria con la ayuda de la odometría inercial es interesante que esta trayectoria quede almacenada en forma de *Keyframes*. En caso contrario, al momento de recuperar la calidad visual provocaría un salto espacial entre la última posición registrada y la futura posición creada al recuperarse. Además, si esta fuese almacenada, supondría una relación entre las poses de entrada y salida del estado de pérdida, permitiendo que al detectar un cierre de bucle, toda la trayectoria inercial se viese beneficiada por la corrección realizada en este proceso. Por este motivo se ha añadido el concepto de *KeyFrame Vacío*.

Los *KeyFrames* Vacíos sirven únicamente de contenedores para registrar la pose estimada por el modelo de movimiento inercial mientras el algoritmo se encuentre en el estado de pérdida. Estos son ignorados por el módulo de *Tracking*, ya que no tiene sentido que sean procesados si no constan de información visual relacionada con el mapa. Para evitar saturar la lista de *KeyFrames*, se construye 1 *KeyFrame Vacío* por cada 5 fotogramas procesados.

Normalmente, la relación entre los *KeyFrames* se realiza en base a la información visual que comparten. Por este motivo, la relación entre los *KeyFrames* Vacíos debe ser forzada, relacionándolos de forma consecutiva los unos con los otros. También es importante relacionar el último *KeyFrame* creado por visión antes de entrar en el estado de pérdida con el primer *KeyFrames* Vacío, así como del último Vacío con el primer *KeyFrame* del nuevo mapa.

5.5. Restauración

Mientras que en los sistemas estéreo o RGB-D esta recuperación se puede hacer de forma instantánea debido a que la información de profundidad de las características detectadas es conocida, no es posible hacer lo mismo en los sistemas monoculares. Para realizar la recuperación es necesario triangular la posición de las características haciendo uso de dos fotogramas con suficiente paralaje.

La calidad del mapa inicial y de la pose de los dos primeros *KeyFrames* en los algoritmos de monocular Visual SLAM es un factor crítico para las futuras características que se deseen añadir. Una mala estimación inicial conducirá a estimaciones erróneas tanto de poses como de características durante cierto periodo de tiempo hasta que el algoritmo pueda recuperarse (si es que llega a hacerlo). En consecuencia, el problema se ha formulado para hacer uso del mismo procedimiento empleado para construir el mapa inicial en SD-SLAM. El proceso implementado consta de los siguientes pasos:

1. Estimación de un nuevo mapa inicial.
2. Alineamiento de las poses de los dos primeros *KeyFrames* y puntos del nuevo mapa con la trayectoria inercial.
3. Comprobar la consistencia del nuevo mapa.
4. Ajustar la escala del nuevo mapa para hacerla coincidir con el antiguo.

El primer paso es similar al realizado en la etapa de inicialización del algoritmo. Durante este proceso, como se explicó anteriormente, se trata de estimar de la pose de los fotogramas iniciales como una transformación obtenida a partir de homografía o matriz fundamental haciendo uso de los emparejamientos obtenidos por ORB. La pose del primer *KeyFrame* se localiza centrada en el origen del sistema de referencia de Mundo y los puntos del nuevo mapa son referenciados a este *KeyFrame*.

Debido a que durante el estado de pérdida se ha estimado la trayectoria realizada por odometría inercial, el siguiente paso es el alineamiento de los dos nuevos *KeyFrames* y del nuevo mapa con esta trayectoria. La transformación a aplicar en cada estado es la siguiente:

- **Primer *KeyFrame*.** Para combinar las trayectorias es necesario que este fotograma tenga asociada la misma pose que la obtenida por odometría inercial. Teniendo en cuenta que se encuentra localizado en el origen, la transformación que hay que aplicar es equivalente a la pose inercial, por ello simplemente se sustituye la pose:

$$P_0^C = P_{I,0}^C \quad (5.16)$$

donde P_0^C es la nueva pose para el primer *KeyFrame* en el sistema de referencia de la cámara y $P_{I,0}^C$ es la pose obtenida por odometría inercial para el primer *KeyFrame*.

- **Segundo KeyFrame.** A la pose anteriormente calculada hay que añadirle el desplazamiento entre realizado entre los dos *KeyFrames*. Como el primer *keyFrame* se encontraba localizado en el origen, la pose del segundo *KeyFrame* representa el propio desplazamiento realizado entre los dos fotogramas. La aplicación de la transformación sobre una pose es calculada como:

$$P_1^C = P_{V,1}^C P_0^C \quad (5.17)$$

donde P_1^C es la nueva pose para el primer *KeyFrame* en el sistema de referencia de la cámara y $P_{V,1}^C$ es la pose obtenida en el proceso de estimación del mapa inicial para el segundo *KeyFrame*.

- **Puntos del mapa.** Los puntos 3D son expresados en el sistema de referencia de Mundo y son referenciados a la pose de la cámara que los detectó por primera vez, en este caso el primer *KeyFrame*. Por este motivo, cada uno de los puntos p del mapa recibe la misma transformación que el primer *KeyFrame*, es decir, $P_{I,0}$. Para este proceso se hace uso de las componentes de la transformación (matriz de rotación $R_{I,0}^W$ y traslación $t_{I,0}^W$) en el sistema de referencia de Mundo, dado que los puntos se encuentran en este sistema de referencia. La nueva posición de cada punto se calcula del siguiente modo:

$$p_i'^W = R_{I,0}^W p_i^W + t_{I,0}^W \quad (5.18)$$

donde p_i^W representan la posición original del punto y $p_i'^W$ la nueva para cada punto i .

En este punto ya se podría realizar la fusión del nuevo mapa con el antiguo, sin embargo, como se verá en el capítulo de ??, el proceso de la estimación del mapa inicial de SD-SLAM no está exento de estimaciones erróneas. A causa de este motivo se ha añadido un proceso para comprobar la consistencia de esta estimación. Este paso comprueba el ángulo formado por los vectores de desplazamiento de la odometría inercial entre los dos primeros *Keyframes* y las estimaciones de la inicialización calculadas en el apartado anterior. Si el ángulo es mayor a cierto umbral, se asume que las estimaciones de pose realizadas por visión son erróneas y, en consecuencia, el mapa es descartado quedando a la espera de recibir nuevos fotogramas para comenzar de nuevo el proceso.

Además, es necesario realizar un ajuste entre la escala del nuevo mapa y el antiguo. Al tratarse de un proceso de inicialización ordinario de un sistema monocular, la escala del nuevo mapa es arbitraria y no tiene porqué coincidir con la escala del mapa antiguo. No obstante,

ambos mapas están relacionados con el mundo inercial, y por tanto, se puede corregir la escala del nuevo mapa.

El objetivo es modificar la posición del segundo *KeyFrame* para que el desplazamiento realizado entre ambos *KeyFrames* coincida con la escala del mapa antiguo (que denominaremos *A*). En primer lugar, se estima el factor de escala que relaciona el nuevo mapa (que llamaremos *B*) con el sistema de referencia de la IMU empleando la ecuación 5.14 obteniendo el factor λ^B . El desplazamiento espacial llevado a cabo entre los fotogramas se transforma a la escala del sistema de referencia de la IMU aplicando $1/\lambda^B$. En este punto se puede hacer uso del último factor de escala estimado en el mapa antiguo (λ^A) para finalmente transformar el desplazamiento a la escala antigua. Este proceso se puede expresar del siguiente modo:

$$s_1^{W(A)} = s_0^W + \frac{\lambda_A}{\lambda_B} \Delta s^{W(B)} \quad (5.19)$$

donde $s_1^{W(A)}$ es la nueva posición para el segundo *KeyFrame* en coordenadas de Mundo *A*, s_0^W es la posición del primer *KeyFrame* del nuevo mapa, $\Delta s^{W(B)}$ es el desplazamiento espacial entre los dos *KeyFrames* del nuevo mapa en coordenadas de Mundo *B*.

El mismo proceso puede llevarse a cabo para escalar cada uno de los puntos del mapa. En este caso la ecuación se re formula del siguiente modo:

$$p_i^{W(A)} = s_0^W + \frac{\lambda_A}{\lambda_B} (p_i^{W(B)} - s_0^W) \quad (5.20)$$

donde $p_i^{W(B)} - s_0^W$ es el vector formado por el punto *i* y la pose de la cámara en el primer *KeyFrame*.

En este punto ya se encuentran los nuevos *KeyFrames* y puntos del mapa transformados para coincidir con la trayectoria inercial y escalados al tamaño del mapa en el momento de entrar en pérdida. Sin embargo, solo se desea trabajar con un mapa en memoria por lo que toda la información del nuevo mapa se añade al antiguo. Finalmente, se relaciona el último *KeyFrame* Vacío creado por la trayectoria inercial con el primer *KeyFrame* de este nuevo mapa. Por último, el algoritmo transita al estado de *Tracking Híbrido*.

Bibliografía

- [1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. volume 3951, pages 404–417, 07 2006.
- [2] G. Cai, B. M. Chen, and T. H. Lee. *Unmanned Rotorcraft Systems*. Springer Publishing Company, Incorporated, 2011.
- [3] J. Civera, A. J. Davison, and J. M. M. Montiel. Inverse depth parametrization for monocular slam. *IEEE Transactions on Robotics*, 24(5):932–945, 2008.
- [4] A. J. Davison. Slam with a single camera. *Concurrent Mapping and Localization for Autonomous Mobile Robots, ICRA*, 2002.
- [5] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *null*, page 1403. IEEE, 2003.
- [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] M. Euston, P. Coote, R. Mahony, J. Kim, and T. Hamel. A complementary filter for attitude estimation of a fixed-wing uav. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 340–345, 2008.
- [8] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. 07 2015.
- [9] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.
- [10] D. Galvez-Lopez and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

- [11] E. P. García. Técnicas para la localización visual robusta de robots en tiempo real con y sin mapas.
- [12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [14] M. Grupp. evo: Python package for the evaluation of odometry and slam. <https://github.com/MichaelGrupp/evo>, 2017.
- [15] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.
- [16] W. S. F. Iv, J. A. Wall, and D. Bevly. Characterization of various imu error sources and the effect on navigation performance. 2005.
- [17] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.
- [18] G. Klein and D. W. Murray. Parallel tracking and mapping for small ar workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.
- [19] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–, 11 2004.
- [20] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3923–3929, 2013.
- [21] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–7, 2011.
- [22] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572, 2007.
- [23] 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.

- [24] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.
- [25] R. Mur-Artal and J. D. Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.
- [26] N. Nain, V. Laxmi, B. Bhadviya, D. B M, and M. Ahmed. Fast feature point detector. In *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, pages 301–306, 2008.
- [27] J. Nazareth. Iterative methods for optimization by c. t. kelley. *SIAM Review*, 42:535–539, 01 2000.
- [28] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [29] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. pages 2564–2571, 11 2011.
- [30] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [31] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284 – 294, 2006.
- [32] L. Stumberg, V. Usenko, and D. Cremers. Direct sparse visual-inertial odometry using dynamic marginalization. pages 2510–2517, 05 2018.
- [33] J. Sturm, W. Burgard, and D. Cremers. Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark. In *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.
- [34] B. Triggs, P. Mclauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment - a modern synthesis. *ICCV '99 Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, pages 198–372, 01 2000.
- [35] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

- [36] O. J. Woodman. An introduction to inertial navigation. *University of Cambridge. Technical Report*, (UCAM-CL-TR-696), 2007.