MÁSTER UNIVERSITARIO OFICIAL
EN VISIÓN ARTIFICIAL

Curso académico 2015/2016

Trabajo Fin de Máster

Análisis de algoritmos de VisualSLAM: un entorno integral para su evaluación

Autor: Victor Arribas Raigadas
Tutor: José María Cañas Plaza

Agradecimientos

En primer lugar me gustaría dar las gracias a todos los profesores del Máster. En especial a mi Tutor José María, el cual alimentó mi pasión por la robótica en tercero de carrera y sin sus revisiones y consejos en la realización de esta memoria la calidad de la misma sería muy inferior.

De forma anecdótica, agradecer también a Juan Antonio su insistencia explicando la transformada de Radón, la cual ha servido de epifanía en la comprensión de los algoritmos.

Por supuesto, agradecer a mi familia su apoyo y comprensión, no sólo a lo largo de este proyecto, sino también a lo largo de mi vida. En especial agradecer a mi padre y a mi novia sus contribuciones a esta memoria planteando otros enfoques y cazando gazapos.

Por último quisiera dedicar unas palabras a mi novia Raquel, cuyos consejos y cabeza fría han sido indispensables durante el desarrollo de este trabajo siendo el principal punto de apoyo en los momentos duros.

Resumen

Con los mercados de los *smartphones* y más recientemente los *drones* apuntando hacia aplicaciones que usan y/o requieren cámaras como elementos indispensables, el abanico de posibilidades tecnológicas crece, y crece el interés en un problema transversal: VisualSLAM. Todo cuerpo móvil con raciocinio tiene la necesidad de saber dónde está y qué es lo que le rodea (SLAM) para poder moverse. Cuando esto se realiza con el sentido de la vista, o en este caso con una cámara, se habla de Localización y Mapeado Simultáneos mediante Visión.

Los resultados de esta línea de investigación son prometedores y permiten aplicaciones prácticas, como el robot aspirador Roomba 980. Estas aplicaciones aumentan la utilidad de este área, contribuyendo a la proliferación de nuevas técnicas, algoritmos y variantes no sólo con cámaras convencionales, también con los nuevos sensores RGBD.

En este Trabajo de Fin de Máster se presenta un entorno integral que permita un análisis y una rica evaluación cuantitativa de las diversas soluciones de visualSLAM existentes y de las que están por venir. Para ello, se ha desarrollado un cauce de evaluación completo que simplifica el proceso de evaluación armonizando cada uno de los elementos implicados. Este cauce se apoya, como estándar que permite comparativas justas, en las bases de datos internacionales (*datasets*).

Para realizar un análisis y evaluación ricos se ha diseñado una herramienta de análisis que resolverá los problemas de ajuste temporal y espacial entre la estimación dada por los algoritmos de visualSLAM y la información de verdad proporcionada por los *datasets* y entregará diversas medidas de error. Esta herramienta ha culminado con el diseño de diversas métricas que permiten una rica evaluación. Una de ellas soportará la tarea de discernir qué algoritmo de visualSLAM se comporta mejor.

El entorno ha sido empleado para evaluar diversos algoritmos de VisualSLAM escogidos por su relevancia, incluyendo el estudio teórico de los mismos.

Palabras clave: *VisualSLAM, cauce de evaluación, métricas de evaluación, herramienta de análisis, framework integral.*

Índice general

CAPÍTULO

1 Introducción

En este capítulo se va a presentar el marco en el que se sitúa el presente Trabajo de Fin de Máster.

1.1. Visión artificial y robótica

El campo de la visión artificial es cada vez más prominente debido a la mejora y abaratamiento constante del hardware. Las cámaras se han convertido en sensores muy asequibles y se encuentran presentes en todas partes. En parte, gracias a la expansión del mercado del smartphone. Este dispositivo inteligente no solo ha contribuido al abaratamiento y miniaturización de las cámaras, sino que también ha hecho lo propio con la CPU y la capacidad de procesamiento embebido. En definitiva, se ha ido contando con procesadores y sensores más potentes y de menor consumo que han copado el mercado sin perder su vinculación con los smartphones.

Aquí nos encontramos por ejemplo con las SmartTV, televisores inteligentes que integran algoritmos de generación de 3D a partir de 2D mediante técnicas de flujo óptico y segmentación de planos, o seguimiento facial para videollamadas con realidad aumentada [Figura 1.1a].

Como ejemplo de esta simbiosis, la cual se ha fortalecido por la moda del *selfie*, se encuentra *selfie mirror*[1] [Figura 1.1b], un espejo inteligente conectado a las redes sociales y a los dispositivos inteligentes de la casa capaz de controlarlos, pero donde destaca su capacidad de reconocimiento visual para la vigilancia.
Por todo ello, la visión artificial será una de las capacidades sensoriales más reiteradas en "*el internet de las cosas*", siendo integrada en múltiples dispositivos empotrados.

A parte de este prometedor futuro, la visión artificial tiene un amplio presente y pasado. Aquí nos encontramos el reconocimiento OCR para procesar documentos o reconocer matrículas de coches para automatizar diversas comprobaciones, como el estado de la ITV, o el correcto pago en zonas de aparcamiento modernizadas [Figura 1.1c].

Otro área en el que ha entrado recientemente la visión artificial es el deporte. Ahí una de las aplicaciones más reconocidas es el ojo de halcón en tenis [Figura 1.1d], cuyo uso se ha extendido al cálculo de estadísticas en pases y saques. También en otros deportes, donde en España destaca MediaPro en el cálculo de estadísticas de los jugadores de fútbol o reconstrucción 3D asistida para verificar los fuera de juego.

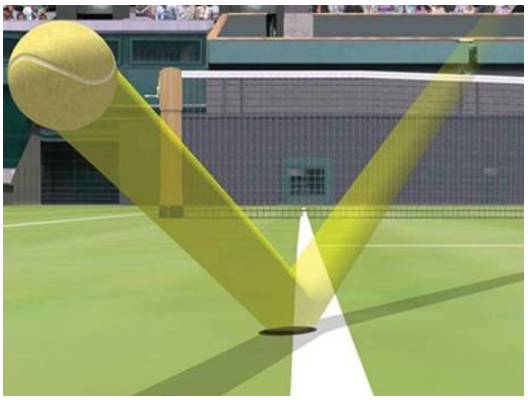
[1] `https://www.indiegogo.com/projects/selfiemirror-the-first-smart-mirror`

Figura 1.1: Ejemplos de aplicaciones de la visión artificial.

(a) Realidad aumentada con seguimiento (*Hangouts*).

(b) Selfie mirror, espejo con su cámara y HUD de control (izquierda) y ejemplo de aplicabilidad en reconocimiento y vigilancia (derecha).

(c) Reconocimiento de matrículas.



(d) Ojo de halcón.

Entre sus otras muchas aplicaciones, se puede destacar la video-vigilancia para el seguimiento de personas o la detección de comportamientos y artefactos extraños. Una de las más clásicas es el reconocimiento facial y de sonrisa en las cámaras fotográficas.

Otro de los campos en los que se usa visión artificial es en robótica. La robótica es una disciplina con más bagaje y que en campos como la automatización industrial ya alcanzó su madurez, cuyo ejemplo más claro son los brazos robóticos.

Acercándonos a la robótica móvil, en los últimos años se ha visto copada por la aparición de los drones de entretenimiento, pequeños vehículos aéreos no tripulados que navegan por control remoto o de forma autónoma. Esta nueva tecnología se está aplicando a muchas áreas, aunque con el nivel de madurez tecnológica actual, su aplicación sólo será viable en algunas aplicaciones concretas.

(a) Nixie.

(b) Phantom3 de DJI.

Figura 1.2: Ejemplos de UAVs de entretenimiento.

En el ámbito de la robótica, un punto de interés se encuentra en el control autónomo, donde se necesita integrar técnicas de SLAM[2] para realizar el control autónomo y el seguimiento. Como ejemplo de uno de estos escenarios se encuentra *nixie*[3] [Figura 1.2a], un mini drone *wearable* equipado con un Intel Edison que se lanza, te saca una foto o vídeo y vuelve a tu mano. Este mini drone, en estado de prototipo, sólo cuenta con dos sensores: IMU y cámara RGB. Para la navegación autónoma se apoyará en la información de la cámara entrando en el ámbito del visualSLAM.

1.2. VisualSLAM

Se denomina visualSLAM a los algoritmos que resuelven, o pretenden resolver, el problema del SLAM usando únicamente cámaras como medio sensorial.

Una de las cualidades de visualSLAM con respecto a otras técnicas que emplean sensores específicos de profundidad, comúnmente denominados RGB-D, es que no se ve limitado por el reducido rango de trabajo de este tipo de sensores. Lo cual permite reconstruir tanto escenarios muy compactos como abiertos.
Esto se debe en parte a la incapacidad de determinar la escala exacta del mapa, es decir, la reconstrucción es completa salvo factor de escala; siendo esta libertad de escala uno de los factores que permite su aplicación a escenarios de diverso tamaño.

Por contra, la indeterminación de la escala puede suponer que la metodología interna de clasificación pueda estar mal definida. Como ejemplos de este caso se encuentran los métodos de detección de espúreos por umbral (RANSAC) y la norma de Huber.
En casos donde la determinación de la escala es crucial, como en odometría, estos algoritmos se suelen inicializar con un patrón de tamaño conocido.

VisualSLAM comprende a los algoritmos que emplean cámaras en escala de grises o en color, así como configuraciones de par estéreo o de una única cámara. En este último caso, el par estéreo se simula obteniendo el paralaje del movimiento.

En este trabajo, teniendo en cuenta las restricciones de aplicabilidad final, solo se contemplarán los algoritmos que emplean una sola cámara, es decir, *(visual) monocular SLAM*, que no debe confundirse con el término *MonoSLAM* acuñado por 1.2.2. Esta sección está destinada a servir como punto de partida y narrar las dos técnicas precursoras[4] en el ámbito del visualSLAM: MonoSLAM y PTAM. Como veremos, la primera técnica proviene de los modelos bayesianos y los filtros de Kalman, mientras que la segunda proviene de las técnicas de optimización empleadas en el SfM (*Structure from Motion*). Son dos formas de afrontar el mismo problema, las cuales han sido planteadas desde la comunidad robótica y la comunidad de visión artificial respectivamente.

1.2.1. Structure from Motion

El SfM consiste en la extracción de la información tridimensional a partir de múltiples vistas de un objeto a través de la fotogrametría. Para ello se emplean técnicas de optimización como el ajuste de haces donde se busca la configuración de puntos 3D y posiciones de cámara que minimizan en error de reproyección.
Por su naturaleza de procesamiento por bloques, se enmarca dentro del procesamiento *offline* y es ideal para tratar secuencias de diferentes fuentes. Sin embargo, no escala para esce-

[2] SLAM: Simultaneous Location And Mapping

[3] `https://flynixie.com/, https://youtu.be/_VFsdPAoI1g` (se recomienda ver el vídeo)

[4] no fueron los primeros, y varios detalles característicos son tomados de trabajos anteriores. Sin embargo son tomados de este modo por suponer un punto de inflexión en el estado del arte.

narios arbitrariamente grandes en tiempo real.

Para un robot, que necesita realizar una acción en cada paso del bucle de control, basta con tener una ubicación de su localización en lugar de una reconstrucción completa y detallada del entorno. Esto se apoya en la naturaleza secuencial del problema, donde por lo tanto se pueden aplicar las técnicas de filtrado clásicas de SLAM.

1.2.2. MonoSLAM

MonoSLAM [DRMS07], Monocular Simultaneous Location And Mapping, es un método de SLAM que propone su resolución con el uso de una sola cámara. Sus inicios se remontan a 2003 [Dav03], donde se propone el uso de este tipo de sensores no solo en el campo del SfM, sino también en el de SLAM.

El método propone el uso de un filtro extendido de Kalman (EKF) por cada punto 3D del mapa que se está generando además del elemental para estimar la posición de la cámara. En cada filtro, los puntos 2D sobre la imagen conforman el modelo de observación. Como peculiaridad, se define un único filtro que incluye la localización de la cámara así como todos los puntos 3D del mapa, en consecuencia , el tamaño del vector de estado cambia a lo largo del tiempo. La motivación de este modelo único es la propia naturaleza restrictiva del movimiento de la cámara, de modo que una única matriz de covarianzas absorbe y representa mejor la correlación de los puntos obteniendo una representación consistente del mapa capaz de realizar cierres de bucle.

El método requiere tres puntos para su inicialización. Para ello se emplea un patrón conocido que permite determinar la escala y las unidades métricas del movimiento ya que de otro modo no podría emplearse como información alternativa o complementaria a la odometría.

Posteriormente, Civera [CDM08] [CGDM10] introdujo dos mejoras sustanciales al método. En primer lugar, se encuentra la parametrización inversa de la distancia. Esta permite la inclusión de puntos 3D en el mapa en el momento en el que son vistos en lugar de ser retrasada hasta que el paralaje de la sucesivas observaciones hicieran converger el modelo de incertidumbre del punto. Unido a la capacidad de representar puntos en el infinito lo convierten en un método robusto ante rotaciones.

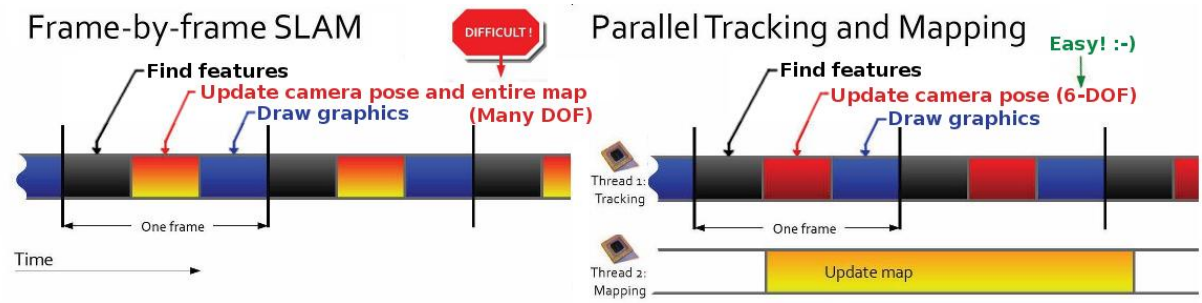
> Los puntos en el infinito son útiles especialmente en escenas al aire libre. Este tipo de puntos son tenidos muy en cuenta en SfM.
> Por ejemplo, una estrella en el firmamento se observa en el mismo lugar aunque se recorran kilómetros. Es un punto que no exhibe paralaje y que no puede ser empleado para estimar la traslación, sin embargo es el candidato perfecto para representar las rotaciones.
> En una escena donde todos los puntos característicos se encuentran en el infinito, el método se convierte en una brújula visual en tiempo real.

La segunda mejora es *1-Point RANSAC*, un método de detección de espúreos más rápido que JCBB (*Joint Compatibility Branch and Bound*).

1.2.3. PTAM

PTAM, Parallel Tracking And Mapping [KM07], es un método publicado pocos meses después de la presentación oficial de MonoSLAM y que presenta una aproximación diametralmente opuesta. Su planteamiento original está enfocado a la realidad aumentada en entornos cerrados y pequeños.

En lugar de emplear técnicas de filtrado como MonoSLAM, las cuales son sensibles a movimientos bruscos, utiliza las técnicas de optimización de SfM. Estas técnicas son muy pesadas computacionalmente, por lo que para llevarlas a tiempo real se separan la estimación de movimiento (*tracking*) y la construcción del mapa (*mapping*) en dos hilos independientes.



Esta separación permite al hilo de mapping elevar la cota de tiempo real a miles de puntos frente a MonoSLAM, cuya cota está en cientos. Además, si ambos procesos están separados, el tracking ya no estará probabilísticamente asociado al mapa, por lo que se puede emplear arbitrariamente cualquier método de tracking robusto. Del mismo modo, la desligadura de ambos procesos implica que ya no existe la exigencia de evaluar el mapa con cada nuevo fotograma, por tanto se podrá ahorrar en cómputo cuando los sucesivos fotogramas tengan información redundante. Esto último permite introducir el término de *keyframe*, que define a aquellos fotogramas con alta disparidad entre ellos creando un balance entre información redundante para la optimización del mapa e información nueva para el emparejamiento en tracking.

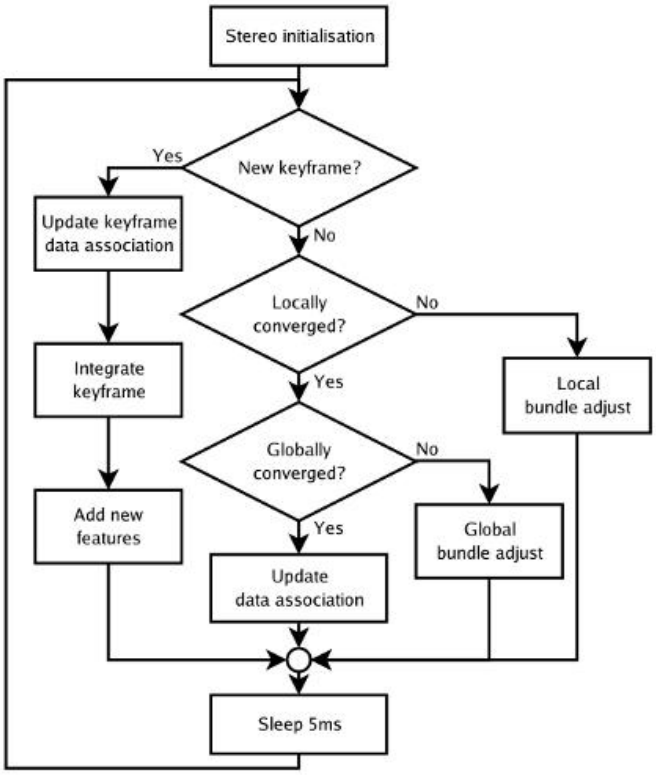


Figura 1.3: PTAM: vista global del algoritmo.

Por contra, el aumento en el intervalo temporal impide el emparejamiento a través de parches "grandes", por lo que se utilizan en su lugar parches pequeños junto con la restricción epipolar para realizar la búsqueda.

Para el tracking se ha elegido una metodología *coarse-to-fine* de dos pasos de optimización. En primer lugar se realiza la inicialización, donde se estima una pose tentativa a partir del modelo de movimiento (que asume velocidad constante). En segundo lugar se proyectan los puntos del mapa de los cuales se buscan 50 emparejamientos en los niveles pequeños de

la pirámide de la imagen. Estos serán empleados para el refinamiento grosero de la estimación de la pose minimizando el error de reproyección. Con esta nueva estimación se vuelven a proyectar los puntos del mapa tomando esta vez hasta 1000 emparejamientos para realizar un refinamiento más fino.

Para que el mapping pueda ir a tiempo real, se realiza la optimización por ajuste de haces con los últimos N *keyframes*. La optimización local debe realizarse antes de la llegada de un nuevo *keyframe*, lo que finalmente permite un tiempo de cómputo mayor al ajustarse al lapso temporal entre *keyframes* y no entre fotogramas.
Cuando el hilo esté ocioso, se programará la optimización completa del mapa, la cual se verá interrumpida si llega información nueva.

La inicialización del mapa se realiza a través del método de cinco puntos [Nis04] que permite la obtención del movimiento entre dos imágenes calibradas a partir de cinco emparejamientos.

Una de las tesis empleadas por el autor frente a MonoSLAM es que los métodos de filtrado pueden aprovechar la información de odometría como solución inicial al aplicarse a robots con ruedas, sin embargo, con cámaras movidas por una mano esta información no existe perdiendo dicha ventaja. Además, los posibles movimientos bruscos generarán errores de asociación para los cuales el método no es lo suficientemente robusto para soportar entornos de realidad aumentada.

1.2.4. Aplicaciones de visualSLAM

Las técnicas de visualSLAM, tanto las basadas en filtrado como las basadas en optimización, están siendo empleadas en diversos ámbitos, las cuales se pueden categorizar en dos grupos atendiendo a los dos modos de actuación posibles:

a. Procesamiento *off-line*: las imágenes o vídeo tomadas, con o sin información de posición, se procesan por lotes mediante técnicas de fotogrametría para obtener la ubicación de cada imagen y una reconstrucción tridimensional de la escena.
   En este caso se enmarca Pix4D[5], donde en la figura 1.4 se puede ver el resultado de realizar este procesado a partir de imágenes tomadas desde un UAV como el Phantom3, considerado el estándar de facto en fotografía aérea.
   Esta aplicación se puede considerar como referencia del *estado-del-arte* actual, cuyo antecedente más importante es *PhotoTourism* [SSS06], proyecto de Microsoft presentado[6] en el SIGGRAPH de 2006.

b. Procesamiento *on-line*: es la clave para aportar este tipo de información sensorial al bucle de control en la navegación autónoma y en la realidad aumentada. Aquí se puede enmarcar el *Proyecto Tango* de Google[7], que emplea un sensor RGB-D para componer en tiempo real un mapa tridimensional de la escena desde un dispositivo móvil. Se espera la llegada del primer *smartphone* con tecnología *Tango* de la mano del fabricante Lenovo.

[5] `https://pix4d.com/` (empresa fundada en 2011)
[6] `https://vimeo.com/30584674`
[7] https://developers.google.com/tango/

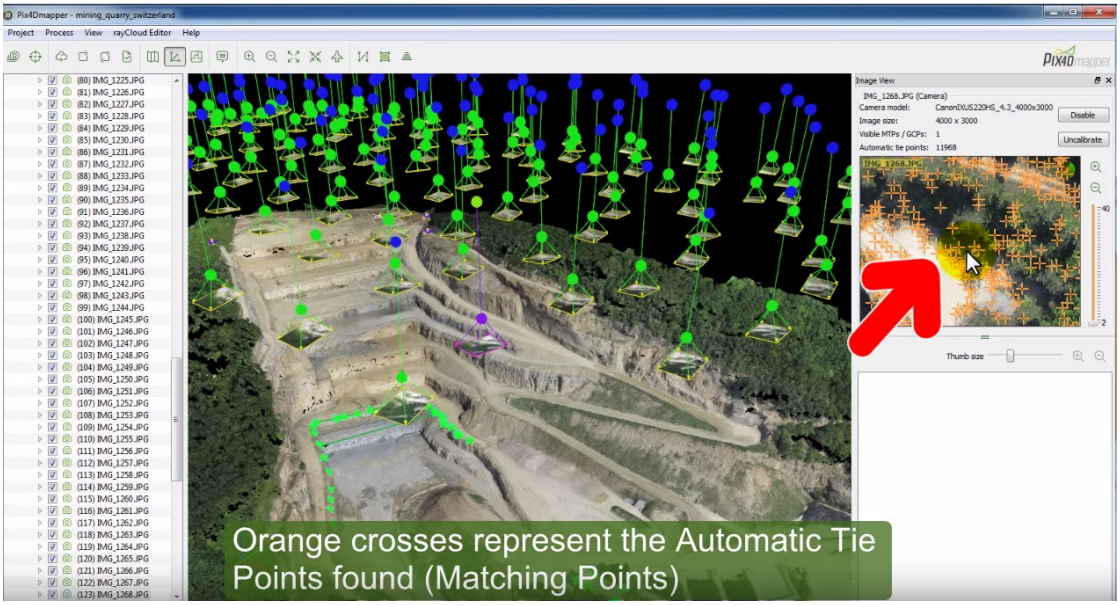


Figura 1.4: 416 imágenes de explotacion a cielo abierto tomadas con UAV y procesadas con Pix4DMapper.

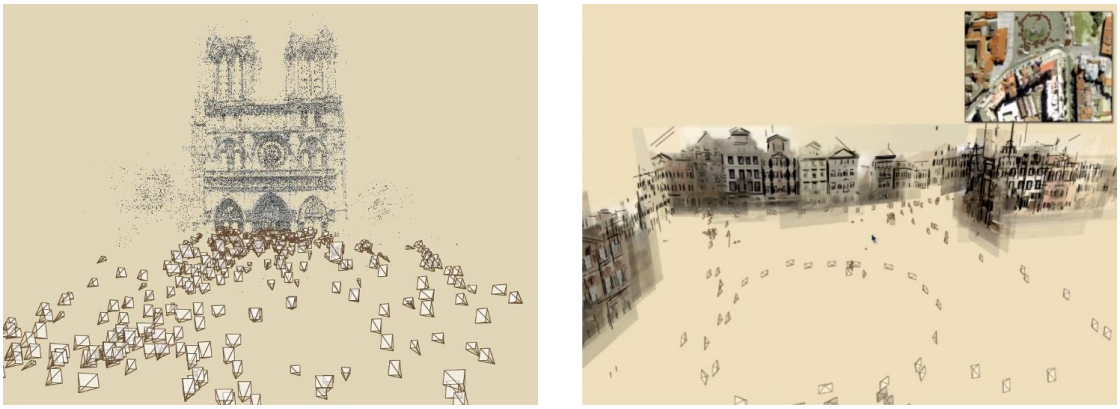


Figura 1.5: Photo Tourism, mapa de los puntos característicos (izquierda) y pseudo-reconstruccion por *image stitching* (derecha).

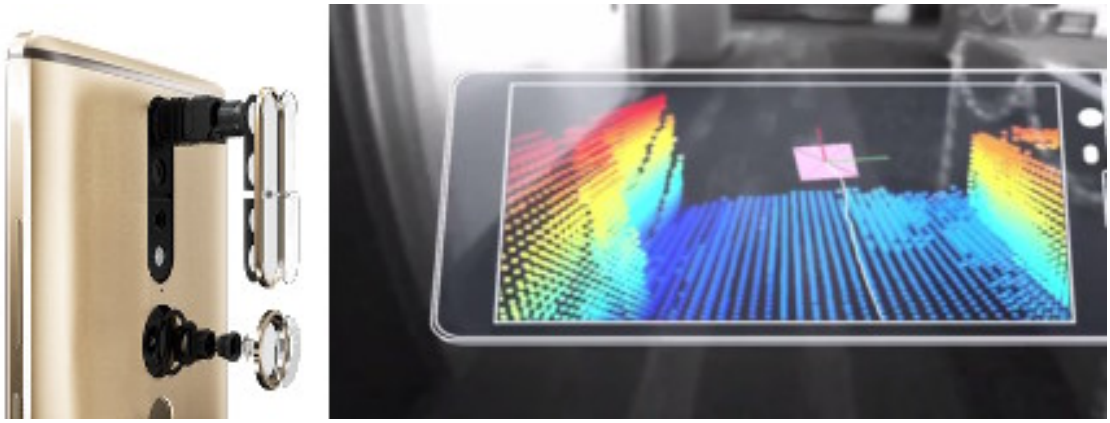


Figura 1.6: Proyecto Tango, sensor específico (izquierda) y recreación (derecha).

Sin embargo, las aplicaciones más destacan son las aspiradoras robóticas que emplean visión, como la Dyson Eye 360 y la Roomba 980. Se trata de dos productos que ya han llegado al mercado (2016) y que reflejan la madurez que están alcanzando tanto las técnicas como la tecnología.

Figura 1.7: Aspiradoras robóticas. Dyson Eye 360 (izquierda) y Roomba 980 (derecha).

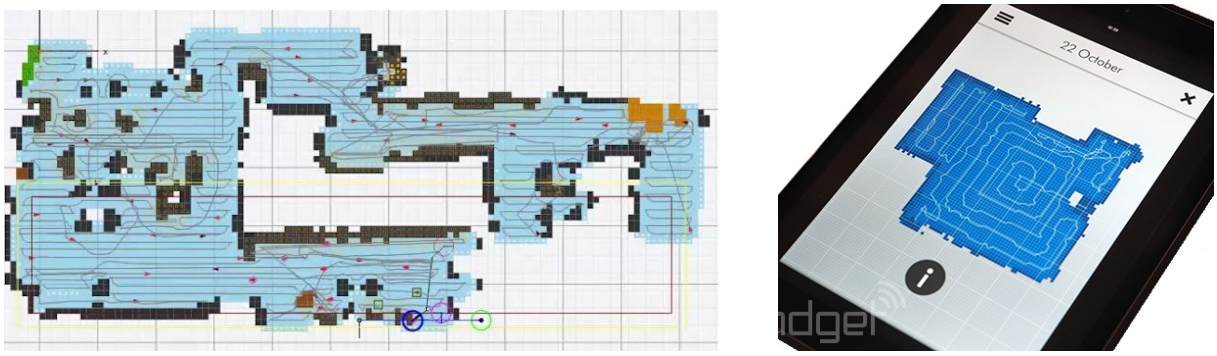


Figura 1.8: Mapa de la casa generado por Roomba 980 (izquierda) y Dyson Eye 360 (derecha).

1.3. VisualSLAM en el Laboratorio de Robótica

En el Laboratorio de Robótica de la Universidad Rey Juan Carlos se han desarrollado varios trabajos en el ámbito del visualSLAM, los cuales son los antecedentes directos de este TFM.

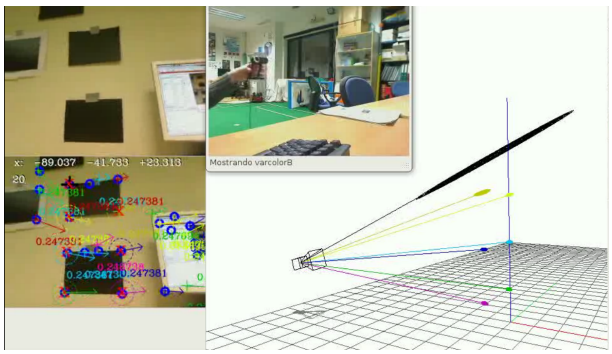
Tomando los más relevantes, podemos destacar en primer lugar el trabajo de López Ramos [LR10], donde se afronta la implementación del método clásico de visualSLAM, MonoSLAM, un método que estima en tiempo real la posición y orientación de una cámara móvil en un entorno estático utilizando imágenes exclusivamente [Figura 1.9a].

El segundo, realizado por Hernández Cordero [HC14], afronta la implementación de dos técnicas de visualSLAM, DLT y una modificación de PTAM basada en el código original, con el objetivo de implementar un sistema de autolocalización sin balizas que pudiera ser empleado para la Realidad Aumentada tanto en PC como en una plataforma móvil (dispositivo Android) [Figura 1.9b]. Con este trabajo se aporta además un componente de JdeRobot que brinda estos dos métodos y el de MonoSLAM de López Ramos al entorno de desarrollo del Laboratorio de Robótica permitiendo seleccionar cualquiera de ellos.

El tercero, realizado por San Román [SRL15], afronta el problema de la Odometría Visual sin necesidad de mapa previo empleando para ello la matriz fundamental [Figura 1.9c].

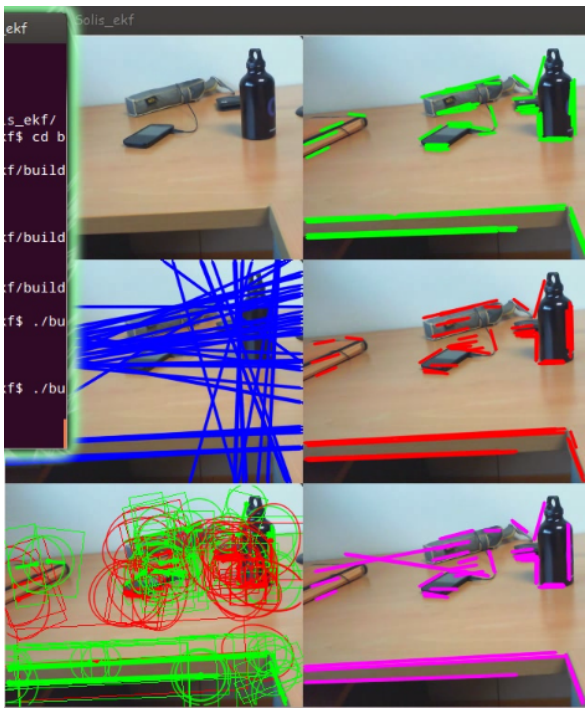
Se pueden destacar otros trabajos de visualSLAM que emplean sensores RGBD, como el Kinect de Microsoft. En este encuadre se encuentran trabajos como el de Martín Organista [MO14], que afronta el problema de la Odometría Visual apoyándose en información de profundidad.

Asimismo, el primer hito en esta dirección se puede ubicar en 2005, con el trabajo de López Fernández [LF05], en el que se desarrolló un algoritmo de autolocalización probabilística para un robot Pioneer en el que se contaba con un mapa del entorno previamente generado empleando para ello filtros de partículas como aproximación no densa de MonteCarlo.



(a) 2010, López Ramos

(b) 2014, Hernández Cordero



(c) 2015, San Román

Figura 1.9: visualSLAM en el Laboratorio de Robótica.

Para este trabajo, y tomando como referencia Pix4D, se empezó a ahondar en la aplicación de las técnicas de visualSLAM para realizar mapas desde dispositivos embebidos como los drones, donde se imponen restricciones de peso, consumo y reducción del *hardware*. Este es un escenario aún no resuelto donde se emplean diferentes técnicas y concesiones para afrontarlo.

En el caso de *Pix4D*, la reconstrucción permite la inspección con una precisión bastante razonable[8], pero esta es realizada de manera *off-line*, donde el tiempo de cómputo no es una limitación. Además requiere el uso de GPS, marcas, referencias y ajuste fino manual para llegar a esos niveles de precisión.

En el caso de *Tango*, se emplea un sensor específico de profundidad, por tanto no es aplicable al mercado actual de teléfonos y UAVs. Además, el peso adicional que supone un hardware no específico a la plataforma de implantación no es despreciable en el caso de los UAV.

En el caso de Roomba 980, y por ende Dyson Eye 360, no es una solución puramente basada en visión. Emplea la información de odometría para mejorar y complementar las estimaciones visuales.

Por todo ello se plantearon unas simples preguntas: ¿se puede afrontar esta tarea con técnicas únicamente visuales, es decir, VisualSLAM?, ¿qué nivel de precisión ofrecen?, ¿el tiempo de cómputo permite su aplicación en tiempo real?.

En responder a estas preguntas se encuentra enmarcado el trabajo que aquí se presenta. Para ello se creará una herramienta que permita el análisis cuantitativo de los algoritmos de

[8]https://support.pix4d.com/hc/en-us/articles/202558889-Accuracy-of-Pix4Dmapper-Outputs

VisualSLAM, todo ello envuelto en un marco de evaluación que también ha sido necesario diseñar prácticamente desde cero.

En esta memoria se reflejan los puntos más relevantes de esta tarea, quedando articulados en diez capítulos. Tras haber realizado en este primer capítulo una breve introducción, en el capítulo segundo, *Objetivos*, se definirán los hitos y la ruta a seguir para la consecución de este trabajo. En este capítulo quedará definido el objetivo principal y la temática de este trabajo: el diseño de un entorno para la evaluación de los algoritmos de VisualSLAM. El tercer capítulo, *Diseño*, está orientado a ofrecer una vista panorámica de la solución planteada, poniendo el foco de interés en el cauce de evaluación. El cuarto capítulo, *Algoritmos de VisualSLAM*, está destinado al estudio teórico y comprensión de diversos algoritmos de VisualSLAM. En el quinto capítulo, *Implementaciones de VisualSLAM*, se hablará de las adaptaciones realizadas a las implementaciones de los algoritmos estudiados en el capítulo anterior, poniéndolas en contexto dentro del entorno de evaluación definido. En el capítulo sexto, *Datasets*, se explicará la importancia de utilizar fuentes de datos conocidas para la evaluación de los algoritmos de VisualSLAM y se presentarán cinco bases de datos internacionales. El séptimo capítulo, *Librería de Análisis*, es el encargado de describir el núcleo de este proyecto, incluyendo las diversas consideraciones a tener en cuenta para afrontar con éxito su implementación. En el capítulo octavo, *Herramienta de análisis*, se definirán las métricas y se detallará el proceso de evaluación, haciendo tangible la herramienta de análisis empleada en el proceso de evaluación. El capítulo noveno, *Evaluación*, está destinado a reflejar parte del proceso de evaluación realizado, condensándolo en dos conjuntos de evaluaciones suficientemente representativos. Finalmente, en el décimo capítulo, *Conclusiones*, se valorará el trabajo expuesto a lo largo de los capítulos anteriores.

CAPÍTULO

2 Objetivos

Con el ámbito del problema más acotado, en este capítulo se procederá a formalizar los objetivos, requisitos y la metodología de trabajo.

2.1. Objetivos

El objetivo de este Trabajo de Fin de Máster consiste en el desarrollo de una herramienta de análisis que incorpore métricas de calidad para la evaluación cuantitativa de diversos algoritmos de visualSLAM mediante el uso de bases de datos internacionales. Para ello hemos articulado este objetivo global en varios subobjetivos o tareas:

1. Realizar un estudio de las diferentes técnicas y algoritmos para afrontar el problema de VisualSLAM, así como sus implementaciones disponibles.

2. Estudiar y adaptar las bases de datos internacionales (*datasets*).

3. Diseñar una herramienta que incorpore métricas de calidad y permita la evaluación cuantitativa de algoritmos de VisualSLAM.

4. Desarrollar un entorno que simplifique el proceso de evaluación y la realización de cada una de las tareas de las que se compone, actuando como nexo de unión y estandarizando la información requerida y contemplada en cada paso.

5. Analizar varios algoritmos de visualSLAM con la herramienta creada y alguna de las bases de datos estudiadas y concluir, en vista de los resultados, cuál es el mejor algoritmo para su posterior uso.

Para la realización del objetivo (1) será necesario acudir al estado del arte y ver cuáles son las tendencias y metodologías que se están empleando en la actualidad. Este estudio se podrá ver en el capítulo 4. En segundo lugar se procederá a estudiar y adaptar las implementaciones existentes para poder realizar la evaluación de los mismos. De este modo, estaremos hablando de los objetivos (1.1) y (1.2).

El objetivo (2) implicará el estudio de diversos *datasets*, incluyendo qué tipo de escenarios proporcionan, a qué tipo de pruebas están enfocados, que información presentan y en qué formato se provee dicha información.

El objetivo (3) es el objetivo principal de este trabajo. Implicará la creación de una herramienta que permita analizar y evaluar de forma individual cada algoritmo de visualSLAM así como la creación de la herramienta que permita la comparación entre ellos. De este modo, podemos dividir el objetivo (3) en dos objetivos más finos:

3.1 Crear la herramienta que permita el análisis puntual que cada algoritmo.

3.2 Diseñar unas métricas de evaluación y cuantificación a partir del análisis realizado.

En este documento, veremos el desarrollo de los objetivos (3.1) y (3.2) en los capítulos 7 y 8 respectivamente.

El objetivo (4) denota la necesidad de realizar un ecosistema completo que permita encajar todas las piezas simplificando las tareas repetitivas y mecánicas. Se trata de una cuestión elemental al realizar una evaluación, donde se entiende que el proceso de análisis y evaluación debe ser repetible para ofrecer la posibilidad de refutar las conclusiones dadas.
Una de las piezas de este entorno será la herramienta planteada en el objetivo (3).

Finalmente, el objetivo (5) implicará el uso de la herramienta desarrollada para realizar la evaluación de los algoritmos de visualSLAM seleccionados. Asimismo, implicará la elección de un algoritmo para su posterior uso. Además actuará como prueba de certificación del objetivo (2). Es decir, si se puede afirmar taxativamente que un algoritmo es mejor para un conjunto de pruebas y criterios definidos, entonces la viabilidad y utilidad de la herramienta habrá quedado probada.

2.2. Requisitos

Los requisitos iniciales, así como los identificados durante las iteraciones del proceso de desarrollo, se describen a continuación, los cuales se pueden dividir en tres grupos.

En primer lugar se contemplan los requisitos globales que se deben cumplir, conformando el primer grupo:

R1. La comparativa debe restringirse a los algoritmos de VisualSLAM, es decir, aquellos cuya percepción es únicamente visual a través de una cámara. De este modo descartamos entre otros aquellos que emplean información de profundidad, denominados RGB-D.

R2. Debe restringirse a los algoritmos de VisualSLAM que empleen una sola cámara.

R3. Debe ser extensible a nuevos algoritmos de VisualSLAM.

R4. Debe usar conjuntos de datos internacionales.

R5. Debe ser extensible a nuevos conjuntos de datos.

R6. Facilidad de uso.

En segundo lugar, y haciendo una iteración más profunda de la especificación de requisitos de algunos objetivos, nos encontramos con los requisitos específicos del objetivo (3):

R7. Debe ofrecer una única métrica cuantitativa para la comparación final entre los algoritmos de VisualSLAM.

R8. Debe ofrecer medidas estándar y a un nivel de detalle más bajo que permitan un análisis más profundo del algoritmo.

R9. Debe ofrecer resultados fácilmente legibles y reconocibles para el usuario.

R10. Debe ofrecer resultados gráficos.

Asimismo, atendiendo a los detalles de implementación, los requisitos específicos del objetivo (4) son:

R11. *Ejecución desatendida*: la ejecución en segundo plano y sin interacción con el usuario es una prioridad para agilizar el proceso de evaluación y de pruebas.

R12. *Entorno virtual*: con él se persigue acabar con cualquier daño colateral que deje inservible el sistema. De este modo las incompatibilidades entre librerías o versiones no afectarán al éxito en la obtención de resultados.

R13. *Entorno repetible*: tener un entorno controlado y determinista es indispensable para asegurar no sólo la repetitibilidad de los resultados sino incluso la capacidad de poder ejecutar el algoritmo.

R14. *Automatización*: la automatización es un requisito indispensable para obtener resultados de calidad. Poder revaluar un algoritmo sin tener que realizar cada paso del proceso a mano evita la tentación de usar datos antiguos y posibilita la realización de más pruebas y ajustes.

R15. *Ejecución fraccionada*: el entorno de evaluación incluirá etapas heterogéneas con diferentes necesidades. Fragmentar cada una de estas tareas aportará mayor flexibilidad al software.

R16. *Ejecución por lotes*: la ejecución por lotes simplificará el proceso de evaluación complementando los requisitos R14 y R15.

2.3. Metodología y plan de trabajo

Como metodología no se ha seguido a rajatabla ninguna concreta, aunque se han seguido las premisas del modelo de ciclo de vida en espiral. Esto es una de las primeras lecciones que se aprenden de las metodologías ágiles, la metodología debe adaptarse al proyecto y no al revés. Dicha premisa cobra más importancia en un proyecto realizado por una sola persona.
Por consiguiente, y enmarcándolo dentro de la metodología en espiral, primero se ha realizado un estudio previo muy amplio que sirve para obtener una visión completa del problema y detectar puntos críticos, y luego se ha ido acotando hasta llegar al desarrollo principal, el cual ha sido revisado y validado en cada iteración. Esta amplitud inicial es importante ya que no sólo nos permite avanzar en todas las vías en paralelo, sino porque ofrece una prueba de concepto para las ramificaciones que se han paralizado en favor del desarrollo troncal.

El proceso de desarrollo ha sido supervisado por el tutor mediante tres herramientas de trabajo: reuniones semanales, definición de hitos y diario de trabajo.
Durante las reuniones se debían definir varios hitos de corto o medio plazo en los que se trabajaría esa semana. Este progreso se puede ver en la página web habilitada para tal uso: `http://jderobot.org/Varribas-tfm`
Así mismo, el código fuente desarrollado puede encontrarse en:
`https://gitlab.com/varribas-pfm` y `https://gitlab.jderobot.org/varribas-pfm`.

El plan de trabajo sería dividido en cinco hitos. El primero implicaría el aprendizaje del entorno de desarrollo del Laboratorio de Robótica de la URJC. El segundo hito sería el estudio teórico de los algoritmos de VisualSLAM así como probar sus implementaciones. El

tercer hito sería el estudio de las bases de datos internacionales y la selección de una de ellas. El cuarto consistiría en el diseño de la herramienta de comparación. El quinto sería la realización de la comparativa y las conclusiones finales.

CAPÍTULO

3 Entorno de evaluación

En este capítulo se resumen las decisiones que han servido para dar forma a la arquitectura general de este proyecto, conformando el denominado cauce de evaluación.

Para obtener una vista general se introducirá en primer lugar el cauce de evaluación, presentando un análisis del problema y de cómo debe afrontarse. Tras ello se resumirán las decisiones de diseño. Por último, se presentará cómo se ha llevado acabo esta arquitectura, dando cuerpo físico a la arquitectura definida.

3.1. Definición del cauce de evaluación

La obtención de resultados es el corazón de este proyecto y es el que marca la arquitectura implicada para ser llevada acabo con éxito. Dichos resultados no son sólo los valores puntuales sobre los que se realizarán las conclusiones, sino todos y cada uno de los datos generados por cada etapa del proceso.
A este proceso con un fin bien definido se le ha denominado cauce de evaluación, el cual queda ilustrado en la figura 3.1 y se sustenta sobre tres pilares:

- datasets
- ejecución
- análisis

El icono ⟲ indica que se trata de una etapa cuyos datos son volubles y que es sensible a ser re-evaluada. Por tanto se han realizado herramientas para su automatización y ejecución en bloque.

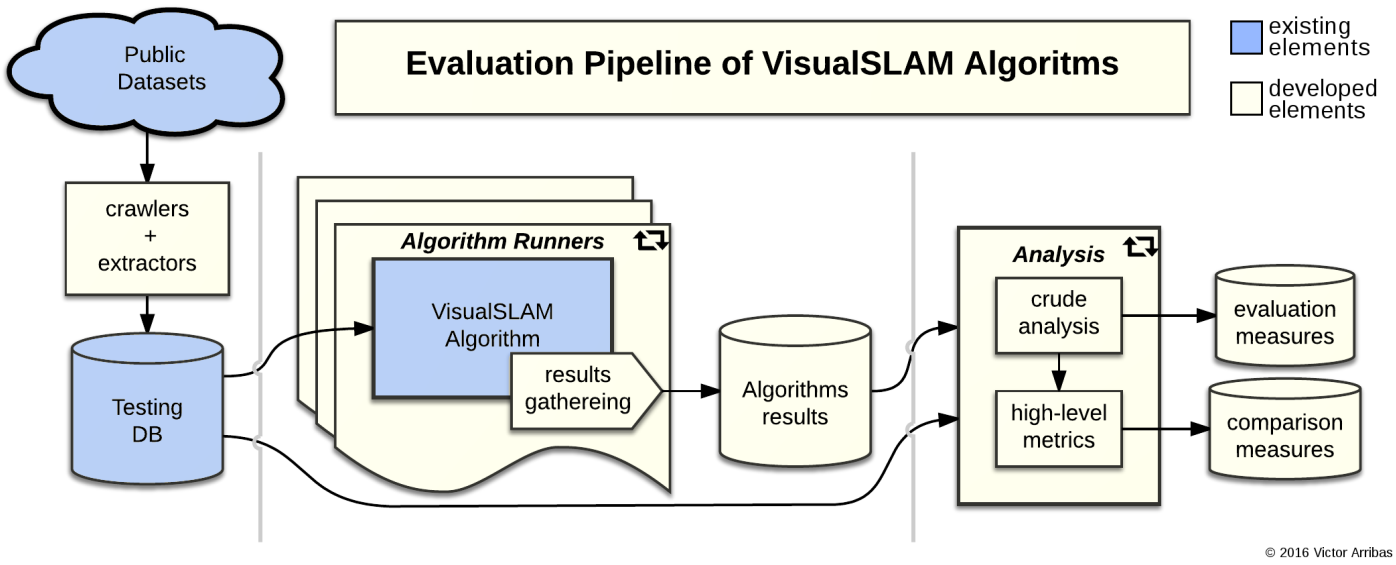


Figura 3.1: Cauce de evaluación, vista global del sistema.

3.1.1. Obtención de los Datasets

Los *datasets* son los datos de confianza. Estos datos alimentarán y caracterizarán el cauce de evaluación de los algoritmos del VisualSLAM, siendo descritos en el capítulo 6.

Este pilar queda completado por tres tareas:

1. Selección del *dataset*
   En esta primera etapa es necesario plantearse cuánto se adecúa el *dataset* a nuestro problema. Qué datos de entrada proporciona, qué métricas de evaluación permite sustentar sobre él, qué casos y escenarios aborda, cuál es su grado de completitud, cuáles son sus limitaciones y cuál es el coste de integrarlo en el cauce de evaluación.

2. Obtención del *dataset*
   Aunque esta tarea suene trivial, suele tener implicaciones no previstas. Las dos más comunes son el tiempo y el espacio.

   Los *datasets* son colecciones de datos tremendamente pesados. Por ejemplo, el dataset de TUM son 50GB en su formato comprimido, y el de VaFRIC son 45GB también comprimido. Con estos números, necesitaríamos más de 100GB por cada *dataset*.

   Como efecto colateral a estas magnitudes de datos, tenemos la limitación de ancho de banda, siendo el caso común de 300Kb/s de pico. De este modo, necesitaríamos 48 horas ininterrumpidas de descarga a velocidad máxima para obtener un único *dataset*. Como es lógico, se han empleado ciertos trucos para que la realización de esta tarea fuera cómoda y robusta ante caídas o pérdidas.

3. Ajuste del *dataset*
   Por último, nos encontraremos con posibles incompatibilidades entre formatos. En algunos casos bastará con renombramientos y reorganización, y en otros exigirá una conversión de tipos.

3.1.2. Ejecución de los algoritmos de VisualSLAM

Se entiende como ejecución a todo el proceso que abarca la ejecución de un algoritmo de VisualSLAM, desde la configuración del entorno hasta la recolección de los datos generados por el mismo.
En el desarrollo esta etapa del cauce nos encontraremos ante las siguientes tareas:

1. Configuración del entorno.
2. Compilación del algoritmo de visualSLAM.
3. Adaptaciones para conectar los datos de entrada.
4. Adaptaciones para guardar los resultados.
5. Ejecución del algoritmo de visualSLAM.
6. Recopilación de todos los datos generados con un orden y nombre adecuados.

3.1.3. Ejecución del análisis

Esta etapa se encarga de simplificar el proceso mecánico necesario para obtener los resultados del análisis y de la evaluación de cada algoritmo. Para llevarla a cabo deberemos disponer de la información de verdad de los *datasets* así como de las estimaciones realizadas por los algoritmos de visualSLAM. Partiendo de esos datos de entrada las tareas a realizar son:

1. Llevar las dos fuentes de datos a un marco comparable.
2. Analizar el error con respecto a la información de verdad.
3. Asignar una puntuación a la estimación.
4. Generar resultados gráficos.

Se trata de la etapa más importante ya que sobre este pilar se sustentan las conclusiones de este proyecto. Esta etapa implica el uso de la herramienta de análisis desarrollada también en este proyecto, la cual encontraremos descrita a lo largo de los capítulos 7 y 8 así como en el anexo A. Para formalizarla no sólo bastará con diseñar y crear los programas que permitan realizar el análisis, sino que además habrá que definir cómo se quiere realizar el análisis, qué se va a medir y cuáles serán los criterios de evaluación.

3.2. Diseño del cauce de evaluación

Las elecciones de diseño del cauce de evaluación se agrupan en cuatro frentes. Por un lado será necesario definir cómo se formalizará cada una de las etapas, por el otro habrá que definir cuál será el nexo de unión y el formato transversal al cauce. En este apartado se recogen solo las más relevantes.

Con respecto a las decisiones globales que caracterizan todo el cauce se pueden destacar tres elecciones de diseño fundamentales.

La primera es relativa al formato elegido para la entrada y salida de la librería de análisis (capítulo 7). Se elegirá un formato de ocho campos ($timestamp,\ tx, ty, tz, qx, qy, qz, qw$), el cual nos permitirá abordar el problema de la sincronización y constituye una representación compacta e inequívoca.

La segunda es relativa a la interfaz de intercomunicación entre los *datasets* y los algoritmos de VisualSLAM. Para ello, tras analizar los pros y los contras y el estado de soporte actual,

se empleará ROS, que nos permitirá delegar en él las decisiones y tareas de acople entre los diferentes elementos. De este modo, en lugar de definir interfaces y formatos propios, se empleará un framework muy reconocido y estándar de facto actual, por lo que nos beneficiaremos del soporte que ofrece la comunidad y brindaremos este mismo soporte para que este trabajo pueda ser utilizado con la misma transparencia.

La tercera decisión de diseño ha sido la modularización y fragmentación de las etapas del cauce y de los elementos de los que se compone cada una, reflejada en los requisitos R15 y R16. La intención es generar pequeñas herramientas y pasos que nos permitan realizar bucles pequeños.
Si el diseño fuese monolítico, sería necesario ejecutar el cauce completo perdiendo flexibilidad. Supongamos que queremos probar varias configuraciones de un algoritmo. Con una arquitectura monolítica deberemos realizar, por cada uno de los casos que proporciona el *dataset*, todas las etapas del cauce y así sucesivamente para cada configuración. Esta cuestión se vuelve extremadamente importante durante el desarrollo, ya que las mejoras, pruebas y la corrección de errores se verían ralentizados por la ejecución del cauce completo cuando solo se está afectando a una parte.
Por ese motivo se ha fraccionado cada parte, utilizando para la comunicación entre cada una de ellas el formato y *framework* de comunicación mencionados anteriormente.

Con respecto a la etapa de ejecución de los algoritmos de visualSLAM, su diseño implicaría la definición, y por consiguiente el cumplimiento, de cuatro de los seis requisitos específicos presentados en el capítulo 2.

Para ello se han empleado dos tecnologías de virtualización livianas. La primera, *CoW*, cumple el requisito R11 permitiendo emplear todo el potencial hardware del equipo anfitrión. La segunda, Docker, cumplimenta los requisitos R11, R12 y R13, además del R14 en términos de despliegue.

Para la ejecución de esta etapa y cumplir con el objetivo R14, se han definido tres estratos de ejecución. La capa de alto nivel se encarga de la ejecución por lotes procesando todos los casos de un *dataset* abstrayendo su configuración y rutas. Incluye soporte de listas blancas y negras para analizar únicamente los casos de interés. La capa de nivel intermedio, que corresponde con la capa superior del envoltorio de ejecución, es la encargada de definir los parámetros exactos que serán pasados al algoritmo de visualSLAM. El parámetro más importante es el archivo de calibración de la cámara, que puede variar dentro de un mismo *dataset*. La capa inferior corresponde al envoltorio de ejecución. Se encarga de normalizar la ejecución del algoritmo de visualSLAM definiendo una API común, encargándose de la conversión y recopilación de los datos a evaluar.

En el Anexo B se describen estas dos tecnologías y los detalles de implementación del envoltorio de ejecución.

3.3. Implantación del cauce de evaluación

Una vista más concreta del cauce puede obtenerse a través del diagrama de despliegue presentado en la figura 3.2. En este diagrama se representa cómo se ha realizado la ejecución del cauce de evaluación, la cual se encuentra dividida en dos partes. En un lado (derecha) se encuentra la ejecución de cada uno de los algoritmos de VisualSLAM. Esta tarea se ha realizado en un servidor dedicado, de modo que los requisitos de almacenamiento y dependencias software no se transfieren a la etapa de análisis.

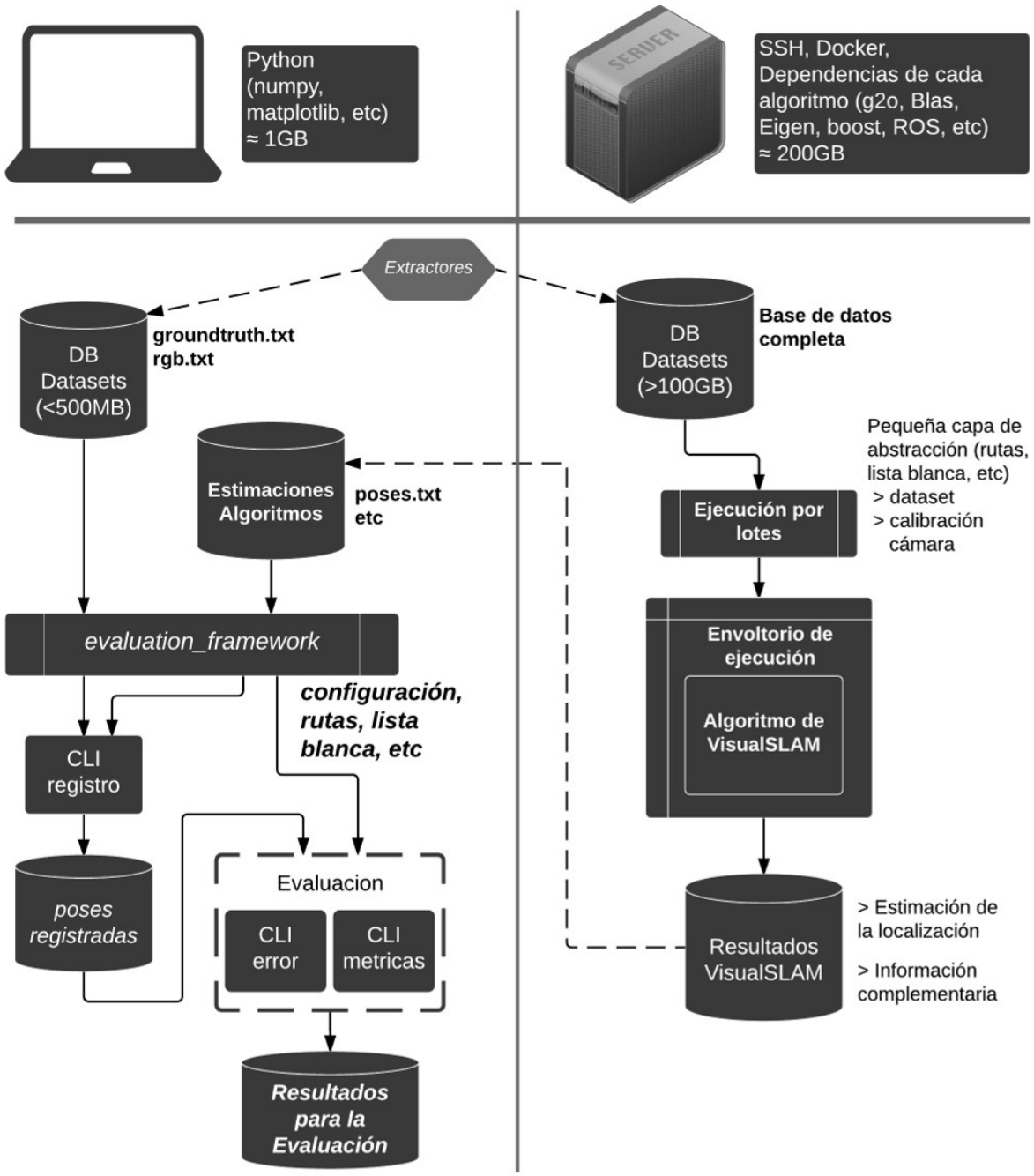


Figura 3.2: Cauce de evaluación: Diagrama de despliegue.

En el otro lado (izquierda) se encuentra la ejecución de la etapa análisis. Esta se plantea como una caja negra, de tal forma que todo el trabajo realizado en local se resume en el siguiente fragmento de código:

```
1 git clone evaluation_framework
2 cd evaluation_framework && source init.env
3
4 cd <empty dir>
5 export DATASET_DIR=$DATASET_TUM_DIR
6 export RESULTS_DIR=$DATASET_TUM_DIR/results.d/algorithm-X
7
8 analysis_error.sh
9 analysis_metrics.sh
```

En las líneas 1 y 2 se presenta el entorno de evaluación el cual se configura a través de *init.env* para poder tener las herramientas disponibles en el *$PATH*.
En las líneas 5 y 6 quedan reflejados los requisitos para poder realizar el análisis de un algoritmo concreto. Además, estos resultados se podrán generar en un directorio independiente para evitar cualquier tipo de contaminación de los datos originales.
Finalmente, las líneas 8 y 9 muestran los dos comandos de alto nivel que bastan para realizar el análisis:

- Con *analysis_error.sh* se generan las medidas de error contempladas. Por ejemplo nos encontraremos con el estudio del error de traslación y de rotación junto con la rectificación de la estimación y la transformación correspondiente que permiten obtener dichas medidas, el estudio y validación del marco temporal de sincronización, etc.
- Con *analysis_metrics.sh* se generan los resultados de las métricas definidas en el capítulo 8, donde se condensan los valores más importantes dados de forma directa o indirecta por el análisis anterior.

Destacar que ambos análisis evalúan todos los escenarios del *dataset* salvo que se especifique lo contrario, generando tanto los resultados numéricos como gráficos.

CAPÍTULO

4 Algoritmos de VisualSLAM

En este capítulo se estudiarán los algoritmos de VisualSLAM escogidos por su relevancia. Para afrontar este estudio antes será necesario definir algunos conceptos básicos.

4.1. Conceptos básicos

Los algoritmos que vamos a estudiar presentan partes comunes y partes distintivas tanto a alto como a bajo nivel, por ello, antes de entrar a fondo en cada una de ellos se realizará una introducción de los conceptos básicos, la matemática asociada y su notación.

4.1.1. Espacios de transformación geométricos

A lo largo de los artículos veremos la notación $SE(3)$, $SO(3)$, $sim(3)$, etc. En este apartado se arroja un poco de luz acerca de esta terminología y su contexto.

Transformación Rígida *(rigbody transform)*
Para este ámbito, la transformación rígida es aquella que solo se compone de una rotación y una traslación. Es decir, sin reflexiones. Por ello estamos ante una transformación rígida adecuada o propia, también denominada rototraslación. Este tipo de transformación es la empleada en cinemática, y pertenece al grupo euclídeo $SE(3)$.

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

Transformación de similaridad
Es el supergrupo de la transformación rígida que incluye un cambio de escala, siendo su notación:

$$\begin{pmatrix} sR & t \\ 0 & 1 \end{pmatrix}$$

Esta extensión sobre la transformación rígida permitirá un mejor registro en el estudio cinemático del espacio proyectivo.

Notación
Las transformaciones son la base de la geometría euclidiana. Estas se encuentran agrupadas según sus características.
Las transformaciones rígidas conforman el grupo $E(3)$, mientras que las rígidas adecuadas conforman el grupo $SE(3)$. Si la transformación no incluye traslación, pertenecerá al grupo $SO(3)$, donde $O(n)$ corresponde al grupo ortogonal (donde la inversa es igual que la traspuesta).

Si materializamos la formulación a través del Álgebra de Lie, la nomenclatura sufre una pequeña variación para constatar en qué espacio algebraico se está trabajando. Así tendremos $se(3)$ como la representación en Álgebra de Lie de $SE(3)$. Y $sim(3)$ como grupo de similaridad [$se(3)$ más escala].

4.1.2. Fotometría y error fotométrico

La fotometría es la ciencia que se encarga de la medida de la luz tal y como el ojo humano la percibe. Para nuestro ámbito, se empleará para el registro entre diferentes imágenes en escala de grises capturadas por la cámara.

La fotometría permite la comparación entre imágenes de forma exacta si y solo si cumplen ciertas restricciones:

- Iluminación constante.
- Superficies lamberianas.

Asumiendo errores debido a las posibles oclusiones y a la resolución, podremos aplicar esta métrica como una simple resta píxel a píxel.

$$Ia_(u,v) - I_b(u',v') \tag{4.1}$$

Dos imágenes se considerarán alineadas si su error fotométrico es mínimo.

4.1.3. Error de reproyección

El error de reproyección es la medida que se emplea en SfM para optimizar las posiciones de las cámaras y los puntos de la escena.

Se trata de una medida que depende únicamente de la geometría de la escena y del tipo de cámaras (modelo de cámara). Sin embargo, la automatización de la selección de los puntos y su emparejamiento emplea técnicas basadas en apariencia así como descriptores que normalmente se apoyan sobre la fotometría.

El error de reproyección es una medida en el dominio de la imagen y vine dado por la ecuación (4.2), donde π representa el modelo de proyección de la cámara.

$$\begin{aligned} e_i = \quad & \left\| \mathbf{u_i} - \pi(T_{cw}, \mathbf{p_i}) \right\| \\ & u_i \in \mathbb{R}^2 \\ & p_i \in \mathbb{R}^3 \\ & T_{cw} \in SE(3) \end{aligned} \tag{4.2}$$

4.1.4. Métodos densos, no densos y basados en características

Métodos basados en características. Se basan en la extracción y emparejamiento de ciertas características puntuales de la imagen. Aceptan un alto grado de desplazamiento entre fotogramas si el descriptor es lo suficientemente invariante. Sin embargo, este grado de libertad, introduce falsos emparejamientos y problemas con las oclusiones.

Métodos densos. Es una forma de definir a los métodos directos, los cuales extraen la información de movimiento directamente de los valores de intensidad de la imagen. Son computacionalmente más pesados, aunque evitan las etapas de extracción y emparejamiento, por lo que todo el esfuerzo se concentra en el cálculo de la transformación. Exigen variaciones de movimiento muy reducidas, por lo que son técnicas idóneas para aplicaciones en tiempo real, o correctamente explicado, con una tasa de refresco elevada. Al trabajar con la totalidad de la imagen, permiten la reconstrucción densa, además de lidiar de forma inherente con texturas de patrón regular.

Métodos no densos. Son aquellos que se apoyan, al igual que los métodos densos, en el estudio fotométrico de la imagen. Sin embargo trabajan a nivel local o sobre un subconjunto de la imagen; véase como ejemplo, aquellas regiones con alta derivada espacial. Por tanto, la reconstrucción de la escena tampoco es densa.

4.2. DTAM

DTAM, Dense Tracking And Mapping [NLD11], es un método de reconstrucción denso que emplea el *error fotométrico* para poder trabajar en el dominio de la imagen directamente. Por ello estaríamos hablando de un método directo y denso.

Para la reconstrucción del mapa emplea una metodología basada en la *transformada de Radón*, con una relación $1:m$ entre una imagen maestra, denominada *keyframe*, y sus esclavas.

Para la localización emplea un refinamiento a dos niveles a través de dos optimizaciones no lineales de Lucas Kanade. El primero en un espacio de transformación limitado, y el segundo en los seis grados de libertad.

A continuación se entrará en detalle en ambos puntos. Sin embargo debido a la complejidad de las fórmulas y las convenciones tomadas se incluye como preámbulo una pequeña información de apoyo[1] para la correcta comprensión de las mismas:

Información de apoyo

Pose de la cámara con respecto al mundo

$$T_{wc} = \begin{pmatrix} R_{wc} & c_w \\ 0^T & 1 \end{pmatrix}$$

Datos asociados a un keyframe *r*

- $\mathbf{I_r}$ - imagen en RGB $\Omega \rightarrow R^3$
- $\mathbf{T_{rw}}$ - pose $SE(3)$
- $\mathbf{C_r}$ - volumen de coste (o error) R^3
- $\xi_\mathbf{r}$ - mapa de profundidad $\Omega \rightarrow R$

mapeo

- **u** - *coordenadas de textura* $u =< u, v > \epsilon\Omega$ (≡ coordenadas x,y de la imagen digital)
- **m** - conjunto de imágenes asociadas a un *keyframe*. $I(r) = I_1 ... I_m$

4.2.1. Reconstrucción *(mapping)*

La reconstrucción sigue un método de minimización de energía global para estimar $\xi_\mathbf{r}$ de forma iterativa. Para poder estimar $\xi_\mathbf{r}$, se construye lo que denominan volumen de coste, $\mathbf{C_r}$.
El volumen de coste se encuentra discretizado en cubos tal y como se muestra en la figura 4.1. Cada cubo – unidad dimensional en el mundo 3D con el mismo píxel asociado – se proyecta en cada una de las imágenes esclavas. Si la resta entre ambos píxeles es cero (error fotométrico nulo), se considerará como la configuración de proyección correcta.

$$\mathbf{C}_r(\mathbf{u}, d) = \frac{1}{|\mathcal{I}(r)|} \sum_{m \in \mathcal{I}(r)} \| \rho_r \left(\mathbf{I}_m, \mathbf{u}, d \right) \|_1$$

$$\rho_r \left(\mathbf{I}_m, \mathbf{u}, d \right) = \mathbf{I}_r \left(\mathbf{u} \right) - \mathbf{I}_m \left(\pi \left(\mathrm{KT}_{mr} \pi^{-1} \left(\mathbf{u}, d \right) \right) \right)$$

Por tanto, el valor de cada cubo es el error acumulado de proyección, es decir, el error de reconstrucción de la transformada de Radón. El cual está normalizado para soportar una construcción comparable e incremental del mismo.

[1] Nota al pie: la información de apoyo, es decir, chuletas, pueden no parecer demasiado representativas, pero el que baje hasta el artículo original las agradecerá.

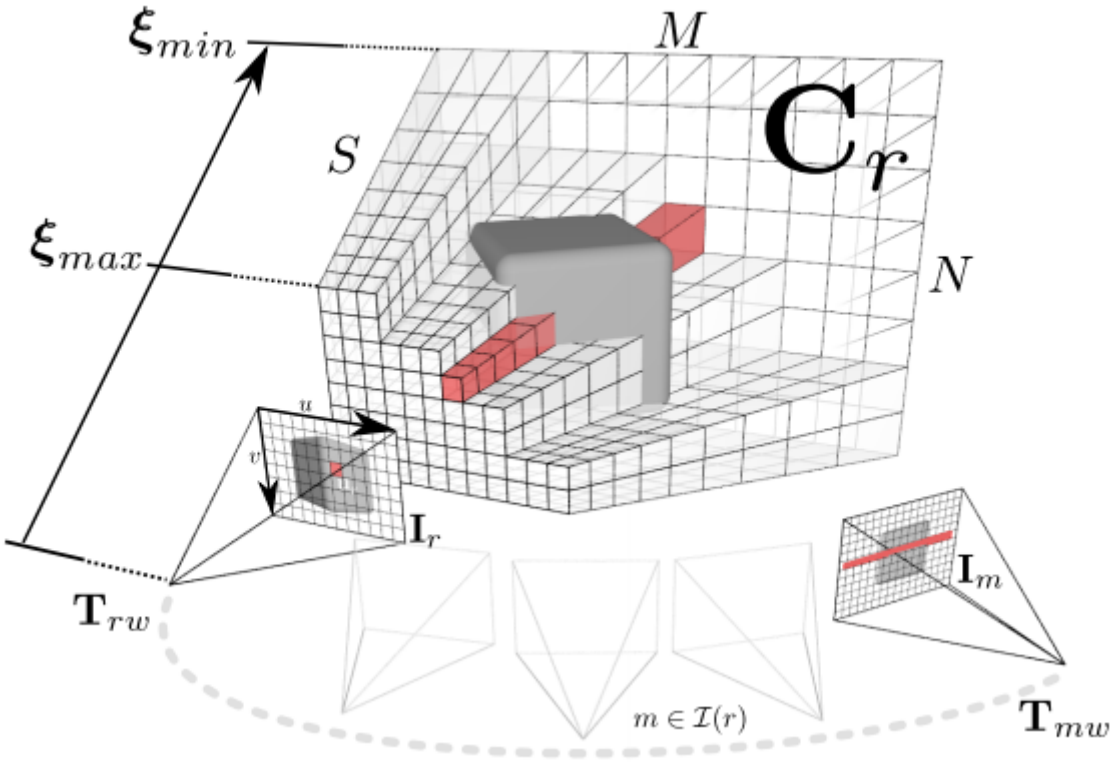


Figura 4.1: DTAM: volumen de coste

El cómputo de dicho volumen de coste se embarca dentro del problema de minimización. Para ello es necesario aplicar un regularizante. La componente regularizante se compone de dos factores:

Norma de Huber:
$$\|x\|_\epsilon = \begin{cases} \frac{\|x\|_2^2}{2\epsilon} & \text{if } \|x\|_2 \leq \epsilon \\ \|x\|_1 - \frac{\epsilon}{2} & \text{otherwise} \end{cases}$$

+

Suma Pesada:
$$g(\mathbf{u}) = e^{-\alpha \|\nabla \mathbf{I}_r(\mathbf{u})\|_2^\beta}$$

De este modo, la fórmula resultante queda de la siguiente manera:

$$E_{\boldsymbol{\xi}} = \int_\Omega \left\{ \underbrace{g(\mathbf{u})}_{\text{Modificador de peso}} \underbrace{\|\nabla \boldsymbol{\xi}(\mathbf{u})\|_\epsilon}_{\text{Norma de Huber sobre el gradiente}} + \underbrace{\lambda \mathbf{C}(\mathbf{u}, \boldsymbol{\xi}(\mathbf{u}))}_{\text{Coste de reconstrucción}} \right\} \mathrm{d}\mathbf{u} \,.$$

Modificador de peso *para reducir la fuerza de regularización sobre los bordes*

Norma de Huber sobre el gradiente *como forma de calcular la variacióntotal (TV)*

Coste de reconstrucción *error fotométrico*

4.2.2. Localización *(tracking)*

Acudiendo a [LD10], podemos ver que la localización se resuelve por medio de una formulación alternativa a EKF utilizando ESM (Minimización Eficiente de Segundo orden).
La localización llevaba mucho tiempo resolviéndose a través de un filtro de Kalman, el cual es un proceso secuencial. Desde la irrupción de PTAM, el tratamiento desacoplado y la paralelización han sido la norma. Precisamente ESM es un método que permite su resolución en paralelo, véase [Mal04].
De este modo, la estimación de la pose se realiza primero siguiendo el método propuesto en [LD10], resolviendo iterativamente por Newton el siguiente problema:

$$f(x) = \frac{1}{2}\|d(x)\|^2$$

$$d_{p_r}(x) = \mathcal{I}^l\left(\mathtt{H}\left(\hat{\mathtt{R}}^{lr}\mathtt{R}^{lr}(x)\right)p_r\right) - \mathcal{I}^r\left(p_r\right)$$

$$x_0 = -(J^{\mathrm{T}}J)^{-1}J^{\mathrm{T}}f(0).$$

Y finalmente afinar la optimización resolviendo:

$$F(\boldsymbol{\psi}) = \frac{1}{2}\sum_{\mathbf{u}\in\Omega}\left(f_{\mathbf{u}}\left(\boldsymbol{\psi}\right)\right)^2 = \frac{1}{2}\|f(\boldsymbol{\psi})\|_2^2,$$

$$f_{\mathbf{u}}(\boldsymbol{\psi}) = \mathbf{I}_l\left(\pi\left(\mathrm{KT}_{lv}(\boldsymbol{\psi})\pi^{-1}\left(\mathbf{u},\xi_v\left(\mathbf{u}\right)\right)\right)\right) - \mathbf{I}_v\left(\mathbf{u}\right)$$

4.2.3. Resumen

DTAM es un método que emplea el error fotométrico de manera integral siendo el único factor de minimización tenido en cuenta tanto para la localización como para la reconstrucción.
Destacar que, aunque ambas etapas se encuentran intercaladas, las dos necesitan partir de un modelo previo. Por ello es un método que requiere una inicialización previa al flujo propuesto. No se detalla cuál es la inicialización más allá de mencionar que se trataría de un método de disparidad estéreo basado en características. Por tanto, podemos intuir que la inicialización es realizada por PTAM.
Por último, recalcar que las condiciones de tiempo real auguradas son resultado de trasladar la resolución del volumen de coste a GPU.

4.3. SVO

SVO [FPS14], Fast Semi-Direct Monocular Visual Odometry, es un método híbrido que conjuga la minimización sobre el error fotométrico con el empleo de características.
Los autores de este artículo han hecho un doble trabajo excepcional en su redacción. En primer lugar, incluyen un resumen comparativo de los método directos y los basados en características. Esta comparativa ha sido tomada como referencia para su explicación en esta memoria[2]. En segundo lugar, referencian sin tapujos en qué trabajos se han basado e inspirado, lo cual es de agradecer frente a otras formas de redactar más oscuras.
De este modo, podemos obtener una vista general del método enunciando que se trata de un método que combina múltiples facetas[3]:

- *parallel tracking and mapping (reference: PTAM)*
- *keyframe selection*
- *many feature tracking*
- *direct aligment via photometric error*

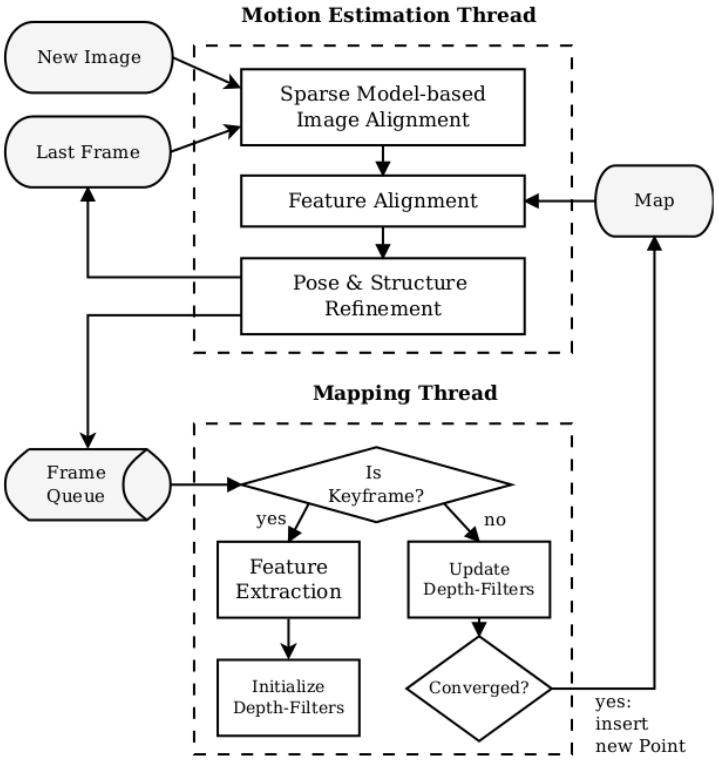


Fig. 1: Tracking and mapping pipeline

De igual modo, quedan aquí presentadas las tres contribuciones principales:

- Método híbrido de estimación de movimiento (*semi-direct approach*).
- Transformada de Radón sobre puntos clave (*semi-dense, sparse depth calculation*).
- Modelo bayesiano para modelar la inhomogeneidad del eje de profundidad (*probabilistic mapping*).

Información de apoyo

- $\mathbf{I_k}$ - imagen en RGB en el instante k, $\Omega \rightarrow R^3$
- $\pi|\pi^{-1}$ - proyección y retro-proyección
- $\mathbf{u}$ - coordenadas de textura/de la imagen digital
- $\bar{\mathbf{R}}$ - en conjunto de $\mathbf{u}$ cuya profundidad es conocida en el instante $k-1$

4.3.1. Modelo híbrido

La hibridación de ambos métodos tiene como objetivo suplir las carencias del método basado en características.

Los métodos basados en características requieren tres etapas: selección, caracterización

[2] sección 4.1.4
[3] Para no perder la trazabilidad con el artículo y sus referencias éstas han sido citadas en inglés

y emparejamiento. La caracterización es muy costosa ya que generar una descripción lo suficientemente invariante tiene una carga computacional considerable, la cual se transfiere a la etapa de emparejamiento.
Asumiendo que el desplazamiento entre *frames* es muy pequeño, y que en la selección de los puntos característicos se exige un mínimo de distancia entre ellos se puede enunciar que:

« Para cada característica en una imagen, su vecino más cercano en la otra imagen es su emparejamiento. »

De este modo, realizando una minimización del error fotométrico para el área circundante se obtiene el emparejamiento entre las características de dos *frames* eliminando las dos etapas más pesadas.
Este proceso se constata mediante la selección de características con FAST, parches de región de 4x4 y evaluación del error fotométrico a varios niveles de la pirámide de la imagen.

4.3.2. Estimación de movimiento *(tracking)*

La estimación del movimiento se realiza en tres pasos incrementales de optimización:

(1) Error fotométrico semi-denso

En primer lugar, se aproxima la transformación [en el espacio $se(3)$] que minimiza el error fotométrico para las zonas de la imagen con profundidad conocida tal y como se observa en la siguiente ecuación:

$$\mathbf{T}_{k,k-1} = \arg\min_{\mathbf{T}_{k,k-1}} \frac{1}{2} \sum_{i \in \bar{\mathcal{R}}} \| \, \delta\mathbf{I}(\mathbf{T}_{k,k-1}, \mathbf{u}_i) \, \|^2 .$$

$$\delta I(\mathbf{T}, \mathbf{u}) = I_k\Big(\pi\big(\mathbf{T} \cdot \pi^{-1}(\mathbf{u}, d_\mathbf{u})\big)\Big) - I_{k-1}(\mathbf{u}) \quad \forall \, \mathbf{u} \in \bar{\mathcal{R}},$$

$$\bar{\mathcal{R}} = \big\{ \mathbf{u} \;\big|\; \mathbf{u} \in \mathcal{R}_{k-1} \;\wedge\; \pi\big(\mathbf{T} \cdot \pi^{-1}(\mathbf{u}, d_\mathbf{u})\big) \in \Omega_k \big\}.$$

Esta ecuación no lineal se resuelve por el método iterativo de Gauss-Newton.

(2) Refinamiento individual parche a parche

Se trata de un *paso de relajación* que viola las restricciones de la epipolar para mejorar la correlación parche a parche. Este proceso lo podemos explicar en tres simples pasos:

I: para cada punto 3D visible en la imagen,

II: se escoge el *keyframe* que mejor se adapta,

III: se optimiza individualmente la transformación afín que minimiza el error fotométrico

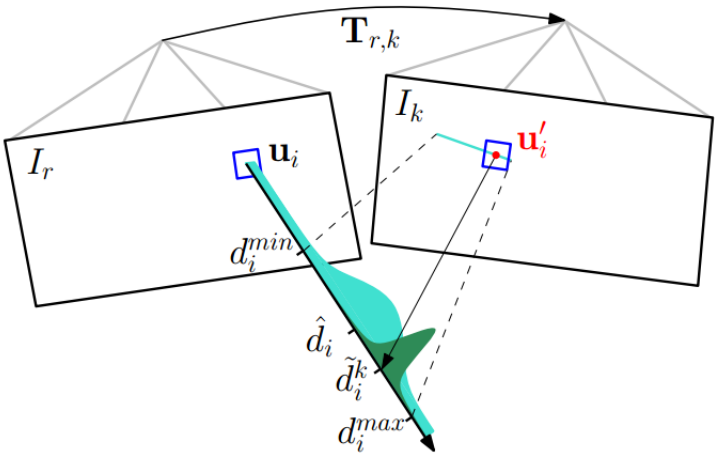
(3) Error de re-proyección

En este último paso se realiza la minimización del error de reproyección clásica de los métodos basados en características. El objetivo es corregir los residuos que genera el paso anterior, los cuales pueden involucrar la pérdida de la ortogonalidad.

4.3.3. Reconstrucción *(mapping)*

El mapa se compone de un conjunto de *keyframes* acompañados de su pose y de N parches.
Los parches son los puntos de FAST con profundidad modelada por una *fdp*. Ésta es definida por un modelo de probabilidad mixto: una componente normal y otra uniforme.
De este modo los outliers quedan caracterizados por la componente uniforme mientras que las medidas correctas se ajustarán al modelo gaussiano en torno al valor de profundidad real.

$$p(\tilde{d}_i^k|d_i,\rho_i) = \rho_i \mathcal{N}\left(\tilde{d}_i^k \middle| d_i, \tau_i^2\right) + (1-\rho_i)\mathcal{U}\left(\tilde{d}_i^k \middle| d_i^{\min}, d_i^{\max}\right)$$



Cuando la incertidumbre de un parche decae, éste es añadido al mapa. Cada nuevo *frame* tiene la posibilidad de convertirse en *keyframe* si diverge lo suficiente del resto. En cuyo caso reemplazará al *keyframe* más alejado.

4.3.4. Resumen

Se trata de un método focalizado en el problema de la localización. Sus autores sitúan el dominio de aplicación de este método a micro-UAVs, donde el cómputo es muy reducido. Se habla de 55fps en una CPU embebida que escalan a 300fps en un sobremesa.

Comparado con DTAM, cuenta con una reconstrucción mucho más pobre, la cual está potenciada por el descarte de los *keyframes* para ser constante en memoria y poder acomodarse a los recursos reducidos de un hardware embebido.

4.4. LSD-SLAM

LSD-SLAM, Large-Scale Direct Monocular SLAM [ESC14], se puede resumir como un método de reconstrucción directo semi-denso que integra probabilidad en la estimación del mapa de profundidad, siendo esta última característica una de las dos aportaciones del método propuesto.
Agradecer en este caso también la claridad con la que los autores describen el método, incluyendo la matemática "elemental" necesaria para comprenderlo, donde se incluye la minimización del error fotométrico a través de la Jacobiana; la primera de las dos formas vistas para su resolución.

Tanto la estimación de movimiento como la construcción del mapa emplean el error fotométrico, donde su diferencia lógica es que el primero trabaja en $se(3)$ mientras que el segundo trabaja en $sim(3)$. Además, en ambos se integra la varianza de la profundidad dentro de la norma de Huber, lo que permite atribuir (pesar con) diferente criterio a las diferentes regiones de la imagen. Esto se sustenta sobre la siguiente premisa:

« En RGB-D la varianza en profundidad se mantiene constante en todo el rango. En RGB no y, por tanto, en modelar esta varianza está la diferencia. »

A continuación se presenta una pequeña aclaración de notación que permitirá la comprensión del resto de la explicación:

Información de apoyo

- $\xi_{\mathbf{ji}}$ representa a la transformación que mueve un punto del fotograma i al fotograma j y pertenecerá a $se(3)$ o $sim(3)$ según el contexto.
- $(.)_{\mathbf{jki}}$ representa al k-ésimo candidato j que se quiere emparejar con i en el grafo de conexión de *keyframes*, denotando al arco entre ellos.
- $\omega(\mathbf{p_i}, \mathbf{D}(\mathbf{p_i}), \xi)$ es la proyección del punto p_i transformado.
- $\mathbf{r_p}$ corresponde al residuo del error fotométrico, siendo $\sigma^{\mathbf{2}}_{\mathbf{r_p}}$ su varianza.
- $\mathbf{r_d}$ corresponde al residuo del error en profundidad.

Datos asociados a un *keyframe* κ_i

- $\mathbf{I_i}$ - imagen en RGB $\Omega \rightarrow R^3$
- $\mathbf{D_i}$ - mapa de profundidad $\Omega_i^* \rightarrow R^+$, donde Ω_i^* es el subconjunto de Ω que presenta un alto gradiente.
- $\mathbf{V_i}$ - varianza de la profundidad $\Omega_i^* \rightarrow R^+$
- $\xi_{\mathbf{ji}}$ - pose entre *keyframes*.
- $\Sigma_{\mathbf{ji}}$ - covarianza de la relación entre *keyframes*.

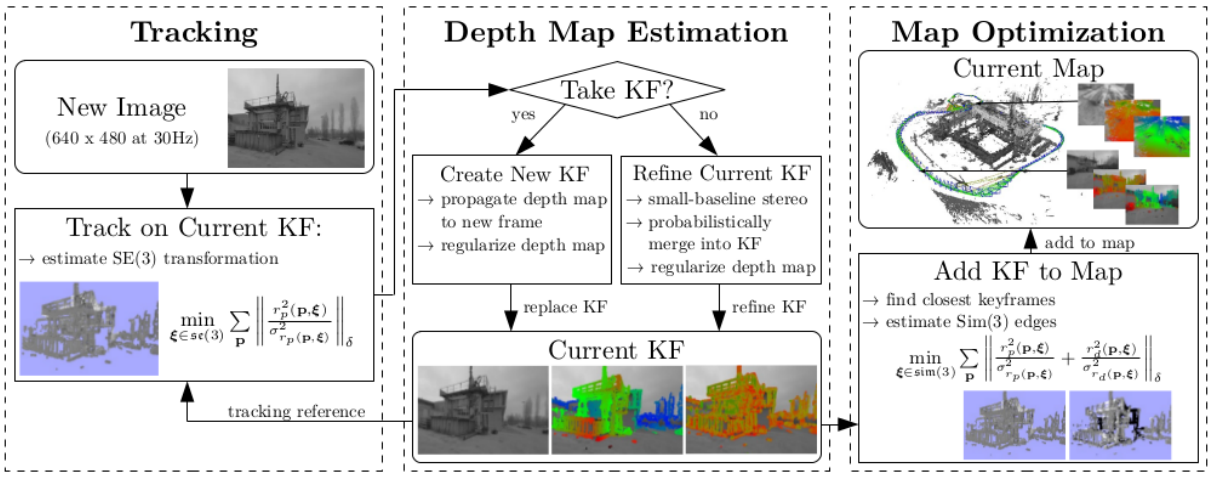


Figura 4.2: LSD-SLAM, visión global del algoritmo.

4.4.1. Localización *(tracking)*

La estimación localización se realiza en el espacio $se(3)$ a través de la resolución de la minimización del error fotométrico por el método de Gauss-Newton (y la Jacobiana). Esta minimización incluye el matiz de estar normalizada sobre la varianza, al igual que el mapa de profundidad.

$$E_p(\boldsymbol{\xi}_{ji}) = \sum_{\mathbf{p}\in\Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p},\boldsymbol{\xi}_{ji})}{\sigma^2_{r_p(\mathbf{p},\boldsymbol{\xi}_{ji})}} \right\|_\delta$$

$$\text{with} \quad r_p(\mathbf{p},\boldsymbol{\xi}_{ji}) := I_i(\mathbf{p}) - I_j(\omega(\mathbf{p}, D_i(\mathbf{p}), \boldsymbol{\xi}_{ji}))$$

$$\sigma^2_{r_p(\mathbf{p},\boldsymbol{\xi}_{ji})} := 2\sigma_I^2 + \left(\frac{\partial r_p(\mathbf{p},\boldsymbol{\xi}_{ji})}{\partial D_i(\mathbf{p})}\right)^2 V_i(\mathbf{p})$$

$$\delta\boldsymbol{\xi}^{(n)} = -(\mathbf{J}^T\mathbf{W}\mathbf{J})^{-1}\mathbf{J}^T\mathbf{W}r(\boldsymbol{\xi}^{(n)}) \;\text{ with }\; \mathbf{J} = \left.\frac{\partial \mathbf{r}(\boldsymbol{\epsilon}\circ\boldsymbol{\xi}^{(n)})}{\partial\boldsymbol{\epsilon}}\right|_{\boldsymbol{\epsilon}=0}$$

A través de esta formulación, se extrae la matriz de covarianzas de ξ, Σ_{ji}, que corresponde a la inversa de la Hessiana de la última iteración $(J^T W J)^{-1}$. Esta matriz no es perfecta, siendo aproximación inferior, pero suficiente.

4.4.2. Reconstrucción *(mapping)*

El mapa se compone de *keyframes*, κ_i, donde el último *keyframe* adquirido es el único que se emplea para la estimación de movimiento.
Un fotograma se establece como *keyframe* si su desplazamiento relativo, $dist(\xi_{ji}) = \xi_{ji}^T W \xi_{ji}$, supera un umbral. En caso contrario el fotograma se emplea para refinar el mapa de profundidad del *keyframe* actual.
Cuando se establece un nuevo *keyframe*, a parte de varios procesos de refinamiento, se normaliza el mapa de profundidad para que su media sea uno. El escalado resultante es el que se empleará como inicialización en el salto de $se(3)$ a $sim(3)$. La estimación de la pose relativa, entre el nuevo *keyframe* y el anterior se realiza a través de la minimización del siguiente funcional de energía.

$$E(\boldsymbol{\xi}_{ji}) := \sum_{\mathbf{p}\in\Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p},\boldsymbol{\xi}_{ji})}{\sigma^2_{r_p(\mathbf{p},\boldsymbol{\xi}_{ji})}} + \frac{r_d^2(\mathbf{p},\boldsymbol{\xi}_{ji})}{\sigma^2_{r_d(\mathbf{p},\boldsymbol{\xi}_{ji})}} \right\|_\delta ,$$

$$r_d(\mathbf{p},\boldsymbol{\xi}_{ji}) := [\mathbf{p}']_3 - D_j([\mathbf{p}']_{1,2})$$

$$\sigma^2_{r_d(\mathbf{p},\boldsymbol{\xi}_{ji})} := V_j([\mathbf{p}']_{1,2}) \left(\frac{\partial r_d(\mathbf{p},\boldsymbol{\xi}_{ji})}{\partial D_j([\mathbf{p}']_{1,2})} \right)^2 + V_i(\mathbf{p}) \left(\frac{\partial r_d(\mathbf{p},\boldsymbol{\xi}_{ji})}{\partial D_i(\mathbf{p})} \right)^2$$

Este funcional permite realizar tracking en $sim(3)$, donde se emplea el mapa de profundidad para asegurar la convergencia.

Por último, se realiza la optimización del mapa a través de la librería *g2o* la cual viene formalizada por la siguiente ecuación.

$$E(\boldsymbol{\xi}_{W1}\dots\boldsymbol{\xi}_{Wn}) := \sum_{(\boldsymbol{\xi}_{ji},\boldsymbol{\Sigma}_{ji})\in\mathcal{E}} (\boldsymbol{\xi}_{ji}\circ\boldsymbol{\xi}_{Wi}^{-1}\circ\boldsymbol{\xi}_{Wj})^T \boldsymbol{\Sigma}_{ji}^{-1} (\boldsymbol{\xi}_{ji}\circ\boldsymbol{\xi}_{Wi}^{-1}\circ\boldsymbol{\xi}_{Wj}).$$

4.4.3. Cierre de bucle

Para el cierre de bucle se agrega una verificación mutua entre los dos *keyframes* implicados para evitar falsos positivos. Se estima la transformación relativa en las dos direcciones y se comprueba si son estadísticamente similares.

$$e(\boldsymbol{\xi}_{j_k i},\boldsymbol{\xi}_{ij_k}) := (\boldsymbol{\xi}_{j_k i}\circ\boldsymbol{\xi}_{ij_k})^T \left(\boldsymbol{\Sigma}_{j_k i} + \mathrm{Adj}_{j_k i}\boldsymbol{\Sigma}_{ij_k}\mathrm{Adj}_{j_k i}^T \right)^{-1} (\boldsymbol{\xi}_{j_k i}\circ\boldsymbol{\xi}_{ij_k})$$

4.4.4. Inicialización

Con este método se propone una inicialización propioefectiva que no requiere el uso de métodos externos. Este consiste en una inicialización aleatoria del mapa de profundidad con una varianza arbitrariamente grande, donde los sucesivos fotogramas harán converger dicho mapa.
El método presume de poder inicializarse con movimientos únicamente rotacionales.

4.4.5. Resumen

En este método se propone un alineamiento de la imagen en el espacio $sim(3)$ eficiente en memoria que no requiere el uso de GPU como DTAM. Así mismo, propone un modelado probabilístico de la no homogeneidad de la profundidad con respecto a los métodos basados en RGB-D.

El salto a $sim(3)$ y la metodología incremental para construir el mapa permiten, con respecto a SVO y a métodos basados en características, la construcción eficiente de mapas grandes.

Por último, el mecanismo de inicialización no basado en PTAM lo hace robusto ante escenarios con movimientos rotacionales.

4.5. ORB-SLAM

ORB-SLAM [MAT14b] [MAMT15], es un método basado en características cuya contribución se encuentra en la interpretación del mapa. Puede verse como una versión de PTAM que incluye cierre de bucle[4], aunque obviamente no es tan simple. Sin embargo esto nos da pie para omitir las cuestiones relativas a PTAM permitiendo una descripción de más alto nivel del método.

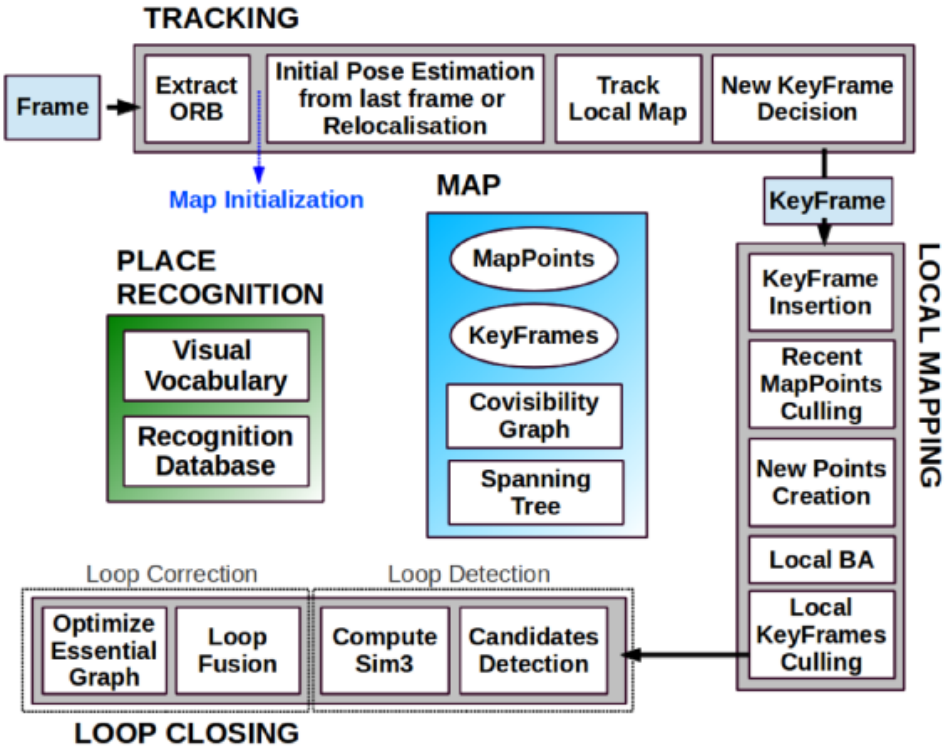


Figura 4.3: ORB-SLAM: vista global del algoritmo.

Información de apoyo
Datos asociados a un punto 3D (en el mapa)

- un parche plano.
- $\vec{\mathbf{n}}$ - la normal del parche, calculada como la media de todas sus vistas, es decir, la media de los rayos de retroproyección.
- un conjunto de asociaciones ORB a varios *keyframes*.
- un descriptor ORB unificado.
- $[\mathbf{d_{min}}, \mathbf{d_{max}}]$ - rango de distancias en los que se ha visto el punto.

En términos generales, los puntos claves del método se pueden reducir a cuatro: base de datos, grafo de covisibilidad, mapa local y metodología de relocalización.

Como base de datos que permita el reconocimiento se utiliza una bolsa de palabras binarias (BoW2), cuyo contenido son los descriptores ORB ordenados en base a dos índices: su importancia dentro de cada *keyframe*, su puntuación obtenida sobre un conjunto de datos de entrenamiento previamente calculado. De este modo se mejora la eficiencia en la búsqueda de emparejamientos.

[4] es una conclusión a la que se llegó en este proyecto, pero que también es afirmada por los autores en [MAT14a]

El grafo de covisibilidad es una de las tres partes del mapa, siendo las otras dos los *keyframes* y los puntos 3D. Por cada arco entre dos nodos (*keyframes*) se define la fortaleza de la relación como el número de puntos del mapa comunes. Este número debe superar un umbral para que exista dicha relación.
Además, este valor será empleado en varios puntos del algoritmo: como apoyo para la base de datos, para definir el mapa local y como verificador del cierre de bucle.

El mapa local, que correspondería en PTAM con la optimización local por ajuste de haces y permite la exploración continua, se formaliza de otra manera. En lugar de emplear sistemáticamente los últimos *k keyframes*, se utiliza la información del grafo de covisibilidad obteniendo un conjunto de **keyframes** más representativo.
Además, este mismo paso se realiza en el tracking, realizando la estimación de la pose con los mejores *keyframes* del grafo en lugar de con el de menor paralaje. Corresponde con el segundo paso de la metodología *coarse-to-fine* para estimar la pose. El gran número de puntos obtenidos se puede reducir con las dos siguientes verificaciones gracias a que ya contamos con una estimación inicial. De este modo los puntos de desechan si:

a. el ángulo del rayo con respecto a $\vec{\mathbf{n}}$ supera un umbral (45^o).

b. la distancia estimada está fuera del rango $[\mathbf{d_{min}}, \mathbf{d_{max}}]$.

Para realizar la relocalización, el fotograma actual se transforma a una bolsa de palabras y se busca en la base de datos su mejor emparejamiento. Sin embargo, en lugar de buscar un único *keyframe*, se utilizan los pesos de los arcos del grafo de covisibilidad para realizar un ranking grupal, donde la puntuación viene dada por la suma ponderada de un *keyframe* y sus vecinos. Esto permite revolver múltiples hipótesis, siendo seleccionado el *keyframe* de mayor peso dentro del grupo victorioso.

La validación del cierre de bucle también se apoya en el grafo de covisibilidad, reduciendo la probabilidad de falsos positivos y permitiendo la propagación omnidireccional de la corrección en las etapas de optimización del mapa local y global.

4.5.1. Resumen

Este método emplea descriptores ORB y y una base de datos con capacidad de reconocimiento (BoW2) para otorgar semántica al mapa. Esto permite no sólo relocalizarse, sino también reutilizar el mapa incluso con una cámara diferente tal y como enuncia su autor.
Esto último potencia la detección de cierre de bucle, la cual con respecto a otras técnicas que se basan sólo en la apariencia, no requiere que los dos *keyframes* sean prácticamente idénticos. De este modo se pueden detectar cierres de bucle aunque la escala (distancia) sea dispar.

Por completitud, decir que la inicialización se realiza buscando la matriz fundamental o la homografía (en caso de escenas coplanares) con ocho puntos mediante RANSAC para obtener un mapa inicial y empezar el seguimiento.

Tanto este método como LSD-SLAM emplean el grafo de conectividad entre *keyframes* para definir un criterio más robusto ante los falsos positivos frente a otros métodos que se limitan a un criterio temporal.

CAPÍTULO

5 Implementaciones de VisualSLAM

Esta sección está destinada a los detalles de las implementaciones de los algoritmos de VisualSLAM anteriormente descritos con vistas a incorporarlos al cauce de evaluación. Para cada uno de ellos se hará referencia al código empleado así como un breve resumen de los problemas encontrados y los cambios que han sido necesarios para poder probarlos.

Para ello será necesario realizar, en primer lugar, una pequeña introducción sobre ROS y los aspectos a los que se hace referencia. ROS es un framework de comunicación centralizado basado en el modelo de publicación/subscripción. Cada componente o programa se denomina *Nodo*, y queda registrado a través de un espacio de nombres.
La comunicación se realiza a través de *topics* y *servicios* mediante envío de mensajes. Los mensajes pueden ser etiquetados con una marca de tiempo, lo cual permitirá la agrupación de datos de diferentes fuentes así como la interpolación de la información cinemática para aumentar su precisión. Esto último lo realiza el *framework* de manera automática y transparente a través del topic */tf*.
Una de sus herramientas es *rosbag*, que permite grabar los mensajes de los *topics* en un fichero especial, denominado *bag file*, para su posterior reproducción.

Así mismo, habrá que definir cuáles son las entradas y salidas concretas. Las entradas de cada algoritmo se resumen en dos. Por un lado una secuencia de vídeo, la cual está contenida dentro de un *bag file*, y por otro la información de captura relativa a dicha secuencia, es decir, los parámetros intrínsecos de la cámara, así como información complementaria como la tasa de fotogramas. La estructura exacta de los *bag file* depende de quién haya realizado la grabación, aunque suelen presentar el mismo patrón al seguir las convenciones de ROS. Un ejemplo de esta estructura se puede encontrar en la sección 6.3 del capítulo 6. La tasa de fotogramas de la secuencia queda implícitamente definida por la marca de tiempo asociada a cada fotograma. Así mismo, los parámetros intrínsecos pueden definirse de manera estándar a través del *topic camera_info*.

La salida de cada algoritmo se resume en una: la estimación de la posición y orientación de la cámara. La metodología estándar para este tipo de problemas es realizar una estimación por cada por cada fotograma de la secuencia de vídeo, a la que se le aplica la marca de tiempo del fotograma para mantener su relación. Esta estimación será almacenada en un fichero llamado *poses.txt*. No obstante, también se recogerá información complementaria como la frecuencia de trabajo del algoritmo de VisualSLAM (fichero *fps.txt*).
Para procesar la salida se emplearán, preferiblemente, *nodos* ROS creados para tal propósito,

los cuales han sido diseñados para permitir su reutilización. En algunos casos será necesario publicar la información pertinente, mientras que en otros bastará con suscribirse a ella.

Por último, decir que para todos los algoritmos se ha planteado la supresión de la interfaz gráfica. De este modo los algoritmos podrán ser ejecutados en modo *headless*, o modo consola, permitiendo su ejecución remota.

5.1. DTAM

Por desgracia, el código fuente de esta implementación está guardado con mucho recelo. Aunque se encontró una aproximación[1], es incompleta y ha dado errores de segmentación, por lo que finalmente fue descartada y no podrá ser incluido en la comparativa.

5.2. SVO

Los autores de SVO ofrecen su implementación de forma libre en la plataforma GitHub[2]. Esta implementación ha sido la más compleja de compilar ya que la ruptura de APIs obliga a buscar la última versión funcional en el historial de versiones. Pasado ese escollo, es la que mejor publica su información a través de ROS, por lo que todo el tratamiento de la información necesaria ha podido realizarse de manera externa tal y como ilustra la figura 5.1. Como nota final avisar que exige conocimientos avanzados de la API de línea de comandos y de configuración de ROS.
En la sección B.3 del Anexo B se adjunta el código del envoltorio responsable de levantar y configurar los nodos y realizar las conexiones reflejadas en la figura 5.1.

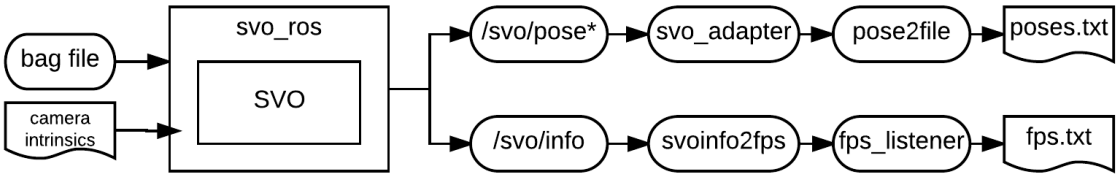


Figura 5.1: SVO: esquema de conexión elemental

[1] `https://github.com/anuranbaka/OpenDTAM`
[2] `https://github.com/uzh-rpg/rpg_svo`

5.3. LSD-SLAM

Los autores de LSD-SLAM ofrecen su implementación de forma libre en la plataforma GitHub[3]. El código fuente ya ofrece una capa de abstracción sobre ROS, sin embargo se ha empleado un *fork*[4] que integra el nuevo sistema de compilación (*catkin*).

Con respecto a los cambios realizados, ninguno de ellos es crítico. Sin embargo, es digno de mención el planteamiento de la interfaz gráfica ya que define una arquitectura de entrada/salida desacoplada. Este planteamiento ha permitido una supresión elegante de la interfaz gráfica[5], la cual tiene dos puntos de conexión:

ImageDisplay

Esta clase actúa como punto de entrada único para realizar las tareas de visualización. Esto significa que en tiempo de compilación uno podría elegir entre OpenCV, Qt o Ninguno, añadiendo al programa la capacidad de cambiar en frío entre proveedores de interfaz gráfica.
Esta posibilidad es la que uno se ha imaginado, ya que el código real está bastante ligado a la API de OpenCV, por lo que realizar una versión en Qt con widgets sería un poco complicado.

La motivación de hacer un proveedor vacío es que este no requiere tocar ni una línea de código y puede ser inyectado añadiendo una compilación condicional a través de CMake.

Output3DWrapper

Esta clase actúa como interfaz para tratar con varios frameworks de comunicación. La implementación lo emplea de manera opcional, por lo que puede deshabilitarse de forma segura a través de un simple *nullptr*.

En la figura 5.2 se muestra el esquema de ejecución. Destacar y agradecer que la configuración de la cámara se realiza de forma transparente e interna a través de la subscripción a la información de la cámara.

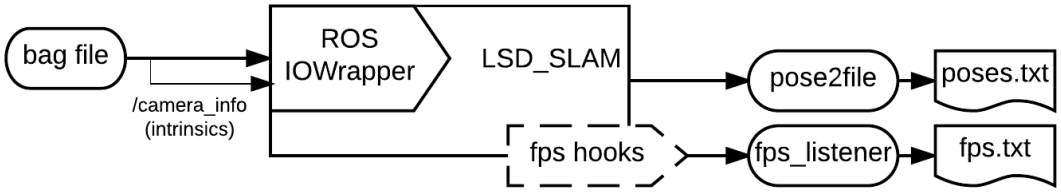


Figura 5.2: LSD_SLAM: esquema de conexión elemental

[3] `https://github.com/tum-vision/lsd_slam` por JakobEngel
[4] `https://github.com/icoderaven/lsd_slam`
[5] `https://github.com/varhub/lsd_slam/tree/option-to-disable-UI-from-cmake`

5.4. ORBSLAM2

El autor de ORBSLAM presentó recientemente una nueva versión[6], la cual motivó su inclusión en esta memoria. Incluye múltiples mejoras, donde destaca la capacidad de trabajar en modo estéreo y mejoras en la forma de realizar la inicialización. Sin embargo, al ser un código nuevo, no está exento de errores. A resaltar uno de los encontrados[7], aunque es preciso decir que sin las mejoras y correcciones por parte de la comunidad, la consecución del envoltorio hubiera sido mucho más compleja. En la figura 5.3 se presenta el esquema de ejecución, donde se puede ver la pseudo-solución al peor fallo detectado[8], ya que bloquea el algoritmo entrando en el problema de la no terminación.

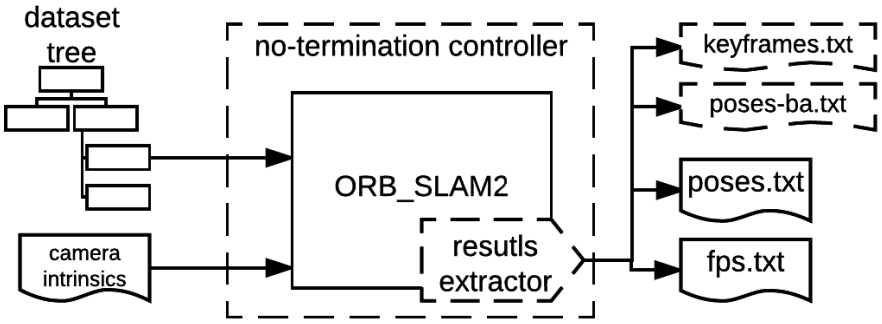


Figura 5.3: ORB_SLAM2: esquema de conexión elemental

Al contar con el problema de la no terminación, se vuelve imprescindible la captura de la salida estándar del algoritmo. Aquí nos encontramos con un *handicap* al usar nodos ROS. ROS tiene su propio sistema de registro de los mensajes de información y depuración, por lo que no está diseñado para hacer fácil el acceso a la salida estándar del algoritmo.
Para afrontar esa limitación y buscar el problema subyacente, se decidió prescindir del envoltorio para ROS. De este modo, la recopilación y guardado de la salida del algoritmo sería realizada manualmente dentro del código.

[6] `https://github.com/raulmur/ORB_SLAM2`
[7] `https://github.com/raulmur/ORB_SLAM2/pull/62`
[8] `https://github.com/raulmur/ORB_SLAM2/issues/63`

CAPÍTULO

6 Datasets

Un *dataset* es un conjunto de casos de uso, reales o sintéticos, que permiten la evaluación sistemática y repetible de un algoritmo.
Los *datasets*, dentro de las limitaciones técnicas y prácticas, suelen proveer un gran abanico de los casos que se le pueden plantear resolver al algoritmo. Este abanico incluye desde casos base – simples, cómodos y cortos para ayudar a la mejora constante de un algoritmo en desarrollo – así como otros más complejos destinados a probar las situaciones más complicadas y/o las más típicas.
El objetivo final de un *dataset* es otorgar una medida cualitativa de calidad del algoritmo de modo que dicha medida sea extrapolable al mundo real.

Un *dataset* no sólo es capturar un caso de uso. También implica obtener toda información complementaria del mismo que sea necesaria para establecer criterios y medidas de calidad y adecuación. Además, esta información complementaria debe ser correcta y precisa. Por todo ello, se agradece a la comunidad científica la liberación de sus *datasets*, los cuales permiten la evaluación de algoritmos con un alto grado de fiabilidad.
Estos *datasets* públicos, realizados por entidades de renombre, son considerados por parte de la comunidad científica como las bases de datos internacionales sobre las que se deben medir las nuevas propuestas y algoritmos de VisualSLAM.

A modo de resumen se expone una lista de los beneficios de usar *datasets* frente a un test directo:

1. Evaluación sistemática y repetible.
2. Proveen información de verdad absoluta para una evaluación cuantitativa.
3. Proveen los casos de uso más representativos.
4. Proveen casos de uso focalizados en ciertas características.
5. Permite la evaluación cualitativa y cuantitativa entre dos algoritmos.

Además, los *datasets* públicos, suelen estar orientados a la comparación y mejora constante del estado del arte, por tanto nos encontraremos:

1. Sistemas de validación cruzada para asegurar una puntuación de calidad no sesgada.
2. Comparativa de todos los algoritmos presentes y pasados del estado del arte.
3. Una vista global de cuál es la cota y norma alcanzadas en un área determinada.

Destacar que la creación de un *dataset* que sirva para la medición es una tarea muy compleja. Basta con acudir a las notas de revisión de algunos *datasets* para ver que éstos mismos no están exentos de errores.

```
Kitti Changelog:
09.02.2015: We have fixed some bugs in the ground truth of the road
            segmentation benchmark and updated the data, devkit and
            results.
31.10.2013: The pose files for the odometry benchmark have been
            replaced with a properly interpolated (subsampled) version
            which doesn't exhibit artefacts when computing velocities
            from the poses.
```

Por ello, no se debe volcar el cien por cien de la certeza en los *datasets* y descartar, siempre en último término, que posibles anomalías en el comportamiento o los resultados se deben a algún error en los mismos.

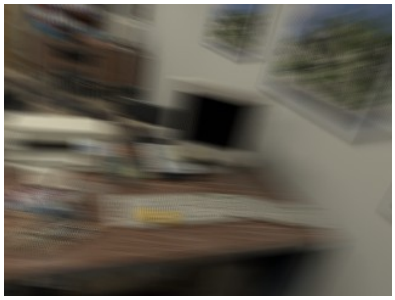
Para el problema de VisualSLAM, los *datasets* que se han tenido en consideración por su relevancia han sido los siguientes[1]:

6.1. VaFRIC

VaFRIC[2] [HNAD12] es un *dataset* orientado a la Odometría Visual y que se ha planteado para responder a la pregunta: ¿aumentar la frecuencia de captura es siempre mejor?. Esta pregunta viene respaldada por un proyecto de seguimiento en tiempo real de la Universidad de Tokio que emplea una cámara especial que trabaja a 1000Hz. En el trabajo realizado se expone esta tesis tomando como referencia el mercado actual de cámaras.

Se trata de un *dataset* sintético que emplea el programa POV-Ray[3] para generar secuencias a partir de modelos 3D de interiores al que se le ha provisto de ruido en sus datos para adecuarlo a las condiciones naturales del mundo real.
Se caracteriza por presentar la misma escena a diferente frecuencia de muestreo (FPS), de tal modo que permita obtener las cotas inferior y superior de trabajo del algoritmo evaluado. Las frecuencias de trabajo van desde los 20Hz hasta los 200Hz. Ofrece dos versiones, una foto-realista que incluye ruido, desenfoque por movimiento y simulación del tiempo de exposición, y otra puramente virtual sin todos estos artefactos.

(a) variante pura (*ray-tracing*)



(b) variante fotorealista

[1] nótese que se le ha otorgado un alias identificativo a cada *dataset*. El objetivo es que sea simple y fácil de recordar aunque no sea la nomenclatura más correcta.

[2] `https://www.doc.ic.ac.uk/~ahanda/VaFRIC/test_datasets.html`

[3] `http://www.povray.org`

La información de cada secuencia se compone de los pares:

- Imagen RGB *(640x480)*.
- Posición y orientación de la cámara *[groundtruth]*.

Lo interesante de este *dataset* es la posibilidad de usarlo para determinar empíricamente las cotas de trabajo del algoritmo además de ser el precursor del *dataset* ICL-NUIM.

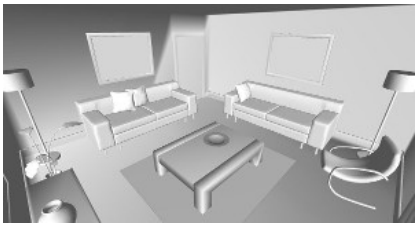
Tabla 6.1: VaFRIC: Enlaces relevantes

Datasets	`https://www.doc.ic.ac.uk/~ahanda/VaFRIC/test_datasets.html`
Conocimientos prácticos	`https://www.doc.ic.ac.uk/~ahanda/VaFRIC/codes.html`

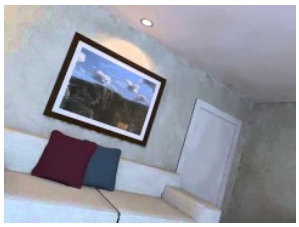
6.2. ICL-NUIM

ICL-NUIM [HWMD14] es un *dataset* orientado a la Odometría Visual, Reconstrucción y Visual SLAM creado entre la colaboración del Colegio Imperial de Londres y Maynooth, Universidad Nacional de Irlanda.

Se trata de un *dataset* sintético generado a partir de modelos 3D de interiores al que se le ha provisto de ruido en sus datos para adecuarlo a las condiciones naturales del mundo real.
Se compone de ocho secuencias, correspondientes a cuatro casos de movimiento en dos escenarios diferentes. Además, se pone a disposición una versión del *dataset* compatible con el formato definido por TUM.



(a) Living Room, modelo 3D



(b) secuencia LR kt0

(c) secuencia Office kt0

La información de cada secuencia consta de:

- Imágenes RGB *(640x480)*.
- Información de profundidad asociada a la imagen *[groundtruth]*.
- Localización de la cámara *[groundtruth]*.

Destacar que los autores publican no sólo el *dataset*, sino también el *pipeline* para que cualquier investigador genere sus propias escenas.
Asimismo, este *dataset* se proporciona en un formato compatible con TUM a parte del suyo propio.

Tabla 6.2: ICL-NUIM: Enlaces relevantes

Datasets	`https://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html`

codes.html (tabla 6.1) es una página que contiene información muy útil y que debe visitarse incluso aunque solo se usen otros *datasets*.

6.3. TUM

TUM [SEE+12] es un *dataset* de VisualSLAM creado por el Grupo de Visión Computacional de la Universidad Técnica de Múnich (TUM).

Consta de un conjunto de 48 secuencias que abarcan un amplio abanico de casos de uso, todas ellas en interior. Cada uno de los escenarios se focaliza en una prueba concreta, incluyendo escenas con movimiento longitudinal o angular, con y sin textura, etc.

(a) sensor Kinect

(b) freiburg2 xyz

(c) freiburg3 teddy

(d) freiburg3 nostructure_texture_far

En la tabla adjunta podemos ver una descripción del conjunto de casos que presentaba el *dataset* en el momento de su presentación.

Sequence Name	Duration [s]	Avg. Trans. Vel. [m/s]	Avg. Rot. Vel. [deg/s]
Testing and Debugging			
fr1/xyz	30	0.24	8.92
fr1/rpy	28	0.06	50.15
fr2/xyz	123	0.06	1.72
fr2/rpy	110	0.01	5.77
Handheld SLAM			
fr1/360	29	0.21	41.60
fr1/floor	50	0.26	15.07
fr1/desk	23	0.41	23.33
fr1/desk2	25	0.43	29.31
fr1/room	49	0.33	29.88
fr2/360_hemisphere	91	0.16	20.57
fr2/360_kidnap	48	0.30	13.43
fr2/desk	99	0.19	6.34
fr2/desk_with_person	142	0.12	5.34
fr2/large_no_loop	112	0.24	15.09
fr2/large_with_loop	173	0.23	17.21
Robot SLAM			
fr2/pioneer_360	73	0.23	12.05
fr2/pioneer_slam	156	0.26	13.38
fr2/pioneer_slam2	116	0.19	12.21
fr2/pioneer_slam3	112	0.16	12.34

Figura 6.4: Descripción de los *datasets* de TUM 2011

A día de hoy se han introducido nuevos casos conformando un *dataset* mucho más completo. Entre ellos se incluyen casos que se focalizan en la existencia o no de textura y/o estructura, en el tratamiento de objetos dinámicos o que presentan un escenario más rico en detalles para evaluar la reconstrucción.

Así mismo, se han agregado casos orientados a la calibración para ofrecer un cauce común en la determinación de los parámetros intrínsecos de la cámara y casos de validación que permiten evaluar el algoritmo realizado y ser presentado en la web oficial de forma objetiva.

La información sensorial consta de:

- Imágen RGB tomada por un sensor Kinect *(640x480@30fps rolling shutter)*.
- Información de profundidad tomada por un sensor Kinect.
- Localización con un sistema de MotionCapture *[groundtruth]*.

El *dataset* TUM presenta un formato dual, a continuación se explican cada uno de los elementos de los que se componen estos dos formatos:

- **Árbol de ficheros (.tgz)**
  - **groundtruth.txt** - fichero con la pose de referencia de la cámara. El formato es:
    `timestamp px py pz qx qy qz qw`
    Donde el *timestamp* corresponde al instante de tiempo en el que se adquirió la pose por parte de un dispositivo de Motion Capture.
  - **rgb.txt** - fichero que lista a modo de tupla la ruta de cada fotograma de la secuencia de video y su instante de adquisición. Su formato es:
    `timestamp frame_path`
  - **rgb/** - directorio donde se encuentran todas las imágenes RGB.
  - **depth.txt** - fichero que lista la ruta de cada imagen (o mapa) de profundidad y su tiempo de adquisición. Su formato es:
    `timestamp frame_path`
  - **depth/** - directorio donde se encuentran todas las imágenes de profundidad.

- **ROS bag (.bag)**
  - **/camera/rgb/image_color** - *topic* donde los fotogramas de la secuencia de vídeo son publicados.
  - **/camera/rgb/camera_info** - *topic* donde se publica los parámetros de calibración de la cámara.
  - **/camera/depth/image_color** - *topic* donde los mapas de profundidad son publicados.
  - **/camera/rgb/camera_info** - *topic* donde se publica los parámetros de calibración de profundidad (cámara IR).
  - **/tf** - *topic* donde se publica el *groundtruth* como una "transformación entre marcos (sistemas) de referencia".

Destacar que el formato *.tgz* carece de los parámetros de calibración de la cámara.

Las marcas de tiempo de las tres fuentes de datos no están sincronizadas debido a:

1. La pose está gobernada por su propio sensor de captura.
2. Las imágenes de vídeo y de profundidad pertenecen al mismo dispositivo, Kinect, sin embargo carece de disparador hardware.

Esto significa que los datos presentan tres desfases y dos periodos diferentes, asumiendo que la información del Kinect se recupere por pares. Las tres marcas de tiempo son recogidas en los ficheros *groundtruth.txt*, *rgb.txt* y *depth.txt* respectivamente.

Esta situación se aplica al *dataset* de TUM al emplear estos dos dispositivos de captura. En el caso de ICL-NUIM, al ser un *dataset* sintético, las tres fuentes de datos vienen emparejadas uno a uno, donde la marca de tiempo es un índice secuencial por motivos de compatibilidad.

Tabla 6.3: TUM: Enlaces relevantes

Datasets	`https://vision.in.tum.de/data/datasets/rgbd-dataset/download`
Formato	`https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats`
Herramientas de evaluación	`https://vision.in.tum.de/data/datasets/rgbd-dataset/tools`

6.4. KITTI

KITTI [GLU12] [GLSU13] es un "benchmark suite" para visión, cuyas siglas derivan de los dos institutos implicados en su creación: el Instituto Tecnológico de Karlsruhe y el Instituto Tecnológico de Toyota.

Provee múltiples *datasets* con escenas de largo recorrido en ciudad tomados por un vehículo. Cada *dataset* se centra una faceta en particular del problema de la conducción autónoma: flujo óptico, detección de carretera, *tracking*, reconocimiento y etiquetado, etc.

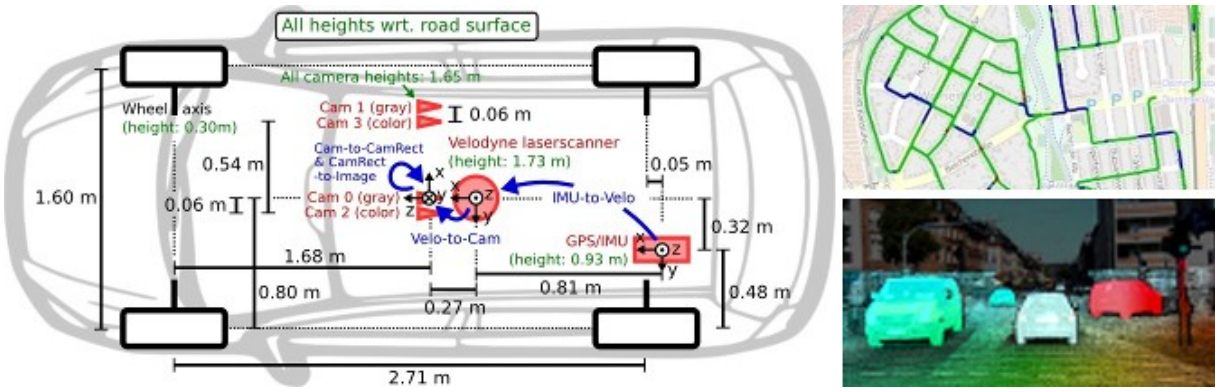


Figura 6.5: Especificación sensorial (izquierda). Escenario de los *datasets* (derecha).

La información sensorial consta de:

- Imágenes RGB formando un par estéreo *(1384x1032@15fps global shutter)*.
- Imágenes en escala de grises formando un par estéreo.
- Información de profundidad tomada con un sensor láser Velodyne *[groundtruth]*.
- Localización con un sistema GPS *[groundtruth]*.

Tabla 6.4: KITTI: Enlaces relevantes

Dataset	`http://www.cvlibs.net/datasets/kitti/eval_odometry.php`
Especificación sensorial	`http://www.cvlibs.net/datasets/kitti/setup.php`
Herramienta de evaluación	`http://kitti.is.tue.mpg.de/kitti/devkit_odometry.zip`
Herramienta de ayuda	`https://github.com/utiasSTARS/pykitti` (Junio 2016)

6.5. MRPT

Se trata de un conglomerado de *datasets* creado por la Universidad de Málaga. Contiene *datasets* propios así como otros tomados prestados de otras fuentes. Todos ellos bajo un formato común: RawLog.

(a) Intel 2003 (b) MonoSLAM (Davison) (c) Málaga 2013 Urban (d) Málaga 2009[4]

Este *dataset* es una apuesta muy interesante porque provee un formato único para acceder a diversos *datasets*, entre ellos se encuentra TUM 2011. En contrapartida, es un formato mucho más complejo al ser genérico y estar formalizado para ser compatible con las redes bayesianas. En cualquier caso se anuncia que cuenta con las herramientas necesarias para su manipulación. Existen dos modos de trabajo:

a) Código C++ integrado en el algoritmo.

b) Nodo ROS delegando la tarea de acople al estándar de facto en Robótica.

Tabla 6.5: MRPT: Enlaces relevantes

Datasets	`http://www.mrpt.org/robotics_datasets`
Formato	`http://www.mrpt.org/Rawlog_Format`
Nodo ROS	`http://wiki.ros.org/mrpt_rawlog`

Tras estudiar y valorar cada uno de los *datasets* se decidiría que el *dataset* que sería empleado para el primer prototipo sería el de TUM. Para entender esta decisión es necesario realizar una pequeña recapitulación.

En primer lugar, dentro de la heterogeneidad de cada *dataset*, se puede decir que KITTI y MRPT están enfocados al problema de la conducción autónoma, donde los casos que presentan están grabados mediante un coche equipado con un diverso y completo grupo de sensores. Así mismo, TUM e ICL-NUIM presentan escenarios de interior donde la cámara se mueve manualmente. Este tipo de movimiento, así como las posibles rotaciones realizadas, se asemeja más a la cinemática que puede presentar un *drone*, por lo que es más deseado.

En segundo lugar, el formato definido en TUM es más compacto y suficiente para el problema que se quiere tratar. Además, ICL-NUIM presenta compatibilidad con TUM, lo que permitiría soportar ambos *datasets* de forma transparente.

Por último, decir que TUM y MRPT soportan ROS, de modo que definir la interfaz de interconexión entre los *datasets* y los algoritmos de VisualSLAM dentro del marco de ROS ofrecería mayor flexibilidad, además de desacoplarnos del formato físico de TUM.

Esta decisión sería tomada en paralelo junto con la formalización del cauce de evaluación presentado en el capítulo 3 y el diseño de la librería de análisis expuesto en el capítulo 7. Persiguiendo de este modo una mayor compatibilidad del cauce de evaluación y una herramienta genérica. En base a ello, se ha desarrollado el diagrama de compatibilidad entre

[4] [BMG09]

datasets expuesto en la figura 6.7, donde los componentes representados en azul oscuro están pendientes de desarrollo.

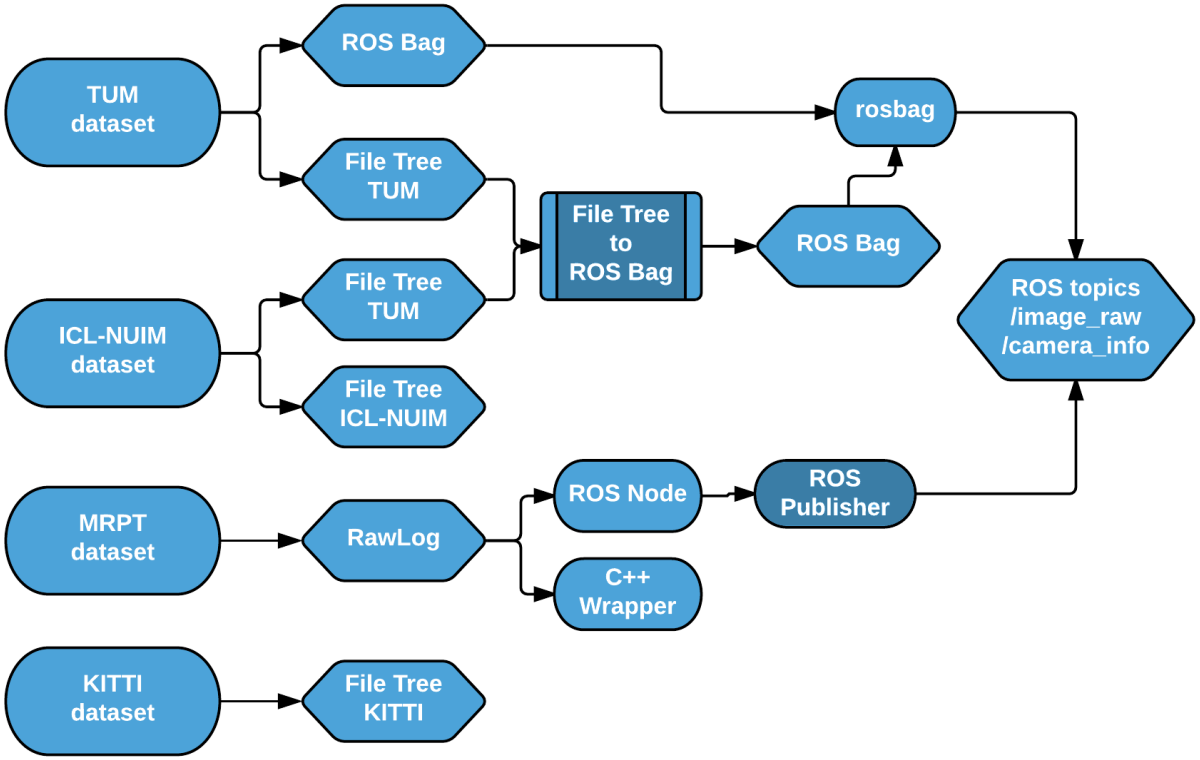


Figura 6.7: Diagrama de compatibilidad de formatos entre *datasets*.

CAPÍTULO

7 Librería de Análisis

7.1. Análisis

En esta sección introduciremos cuáles son los datos que deberá analizar la herramienta que vamos a diseñar, así como la problemática que presentan dichos datos. Por cada problemática, se presentarán las alternativas o vías de actuación eligiendo aquella que parece más razonable.

7.1.1. Objeto de análisis

La evaluación de los algoritmos de VisualSLAM consta de dos partes. En primer lugar, la evaluación de la calidad de la estimación del movimiento. En segundo lugar, la calidad de mapa reconstruido. El objeto de estudio de este proyecto será la calidad de la *localización* de los algoritmos de VisualSLAM. Para ello nos apoyaremos en la información de verdad absoluta que proveen los *datasets* estudiados en el capítulo 6 y que se encuentra ejemplificada la sección 6.3 de ese capítulo.

La información de localización consta de una posición y una orientación en el espacio tridimensional, las cuales representan la ubicación de la cámara y su orientación en un instante de tiempo determinado. A esta conjunción de posición más orientación es denominada *Pose*, y a la secuencia de poses la denominaremos *Track*.

Gracias a las dos fuentes de datos, la verdad absoluta proporcionada por el dataset y la estimación de localización por parte del algoritmo de VisualSLAM, podremos realizar el análisis comparativo entre ellas permitiendo la extracción de estadísticas, errores y puntuaciones.

7.1.2. El problema de la sincronización I: marco de trabajo

En el apartado anterior hemos asumido que las dos fuentes de datos son comparables, lo cual puede no ser verdad. Veámoslo en más detalle.

Los datos recibidos por los algoritmos de VisualSLAM pueden tener como origen varios sensores diferentes. Cada sensor tiene su propia frecuencia de trabajo y sin un *trigger hardware* los datos recopilados presentarán un decalaje u *offset*. Para los algoritmos que se van a analizar en este trabajo tiene una importancia menor, ya que acorde a los requisitos R1 y R2, los algoritmos presentan una única fuente de datos: la imagen de la cámara. Sin embargo, esta afirmación debe tomarse con cuidado ya que la primera mejora de estos algoritmos

suele involucrar un sensor inercial, o IMU, que permita hacer una mejor inicialización o su fusión a posteriori.

No obstante, la verdad absoluta es recopilada por otro sensor, y como es una información desconocida para el algoritmo de VisualSLAM, jamás podrá presentar una sincronización salvo que se realice de forma externa.
Esta sincronización externa se plantea comúnmente de dos formas:

A) Sincronización 1:1
   Este tipo de sincronización implica un emparejamiento uno a uno. De este modo, por cada fotograma del vídeo de entrada existirá una estimación del algoritmo de VisualSLAM. Esta es la estrategia que se ha establecido como norma en las estimaciones en tiempo real y es factible cuando el algoritmo es liviano computacionalmente.

B) Sincronización temporal
   El caso anterior no se puede practicar con información sensorial proveniente de diferentes sensores. En este caso, la vía planteada es la definición de un marco de referencia temporal común a todos los sensores, de modo que quede reflejado el periodo y desfase de cada sensor.

Ambas formas son acumulativas, siendo la sincronización temporal el mecanismo de sincronización presente en todos los datasets estudiados en el capítulo 6. Por ello, la herramienta de análisis adoptará dicho mecanismo como condición necesaria.

7.1.3. El problema de la sincronización II: dominio y frecuencia portadora

Hasta ahora hemos viso cuál es el mecanismo que nos permite realizar la sincronización, pero no el método que lo lleva a cabo. Este método será la interpolación.

La interpolación depende del tipo de datos que se estén tratando, y no es siempre aplicable. Por tanto podemos clasificar la sincronización según si acepta o no interpolación:

- Emparejamiento en el dominio de origen: es el caso en el que los datos permiten la interpolación. Es decir, existe una transformación temporal en el dominio de los datos que permita inferir el intervalo continuo a partir de dos muestras discretas.
- Emparejamiento en el dominio de resultados: es el caso en el que los datos no permiten la interpolación. En este caso se contempla y modela el desfase, siendo este corregido a posteriori en un dominio que sí permita la transformación.

Dado que nuestros datos admiten emparejamiento en el origen, sólo falta escoger el periodo portador de la totalidad de los datos sensoriales. Aquí nos encontramos tres opciones:

- Emparejamiento al máximo periodo (o mínina frecuencia):
  A priori es el menos deseado ya que se pierde densidad en los datos, derivando en perdida de datos de entrenamiento en escenarios de machine learning, o en la diferencia entre una evolución no lineal o lineal en cinemática.
  Sin embargo, es el que genera una medida de error más optimista. Esto se debe a que en promedio genera un error de curvatura menor[1].

[1] existe error de curvatura si y solo si la frecuencia subyacente es superior a la de muestreo. Más adelante se explica este término.

- Emparejamiento al mínimo periodo (o máxima frecuencia):
  Produce un mayor número de datos sintéticos. Esta propiedad deseable tiene las siguientes contraindicaciones: maximiza el error de curvatura, genera datos no presentes en el caso de estudio y sesga el estudio estadístico.

- Emparejamiento a un periodo arbitrario constante:
  Esta elección permite aplicar teoría estadística elemental de forma correcta.

7.1.4. El problema de la interpolación: método de interpolación

Existen múltiples técnicas de interpolación, lineal, cúbica, etc. Cada una presenta unas cualidades que pueden ser deseables o no para nuestro ámbito. A continuación se presenta la motivación que ha llevado a la elección de la interpolación lineal.

Tratándose del muestreo de un movimiento, realizar una interpolación suave, por ejemplo a través de una spline, sería más que plausible. Sin embargo, ello implica imponer un requisito que el problema no cumple: *derivada contínua*.
Con este conocimiento en mente, la única posibilidad correcta de realizar algún tipo de suavizado sería caracterizando los umbrales de movimiento del objeto. De ese modo, se obtendría cierto grado de suavidad respetando la discontinuidad de la derivada en los cambios de dirección bruscos.

Estas consideraciones cobran aún más importancia cuando se aplica sobre la orientación. La orientación está definida como un cuaternión por sus múltiples ventajas. En este espacio, podemos plantear los tres tipos de interpolación más comunes. Cada una cumple ciertas propiedades tal y como se ilustra en la siguiente tabla[2]:

	commutative	**constant velocity**	**torque-minimal**
quaternion slerp	No!	Yes	Yes
quaternion nlerp	Yes	No!	Yes
log-quaternion lerp	Yes	Yes	No!

Tabla 7.1: Tres métodos de interpolación de rotaciones y las propiedades que satisfacen

En el campo de la animación, estas diferencias son tenidas muy en cuenta. En el contexto actual requiere saber qué propiedades cumplen los datos y cuáles son las que se requiere que cumpla la interpolación.
En el peor de los casos, una mala elección del método puede desembocar en la divergencia de los datos, lo que en último término puede llevar a casos no comparables, y por tanto la comparativa entre ellos sería incorrecta.

Al haber escogido la interpolación lineal entre puntos se asume constancia en velocidad. En consecuencia, debemos mantener la coherencia eligiendo el método que ofrece la misma propiedad.

[2] tabla cortesía de [LEF95], una de las dos fuentes más concisas junto con [Kei11]

Error de curvatura
Se define error de curvatura al área que encierra el arco entre los dos puntos de un segmento. En las ilustraciones 7.1 y 7.2 se ejemplifica los dos tipos de error de curvatura que se pueden dar. Están representados con los colores azul y rojo y tienen una tasa de muestreo diez veces inferior a la de la curva ideal representada en negro.
En estos dos casos anteriores, el error introducido se debe al remuestreo a una frecuencia superior. Sin embargo este no es el único caso. Todo remuestreo a otra frecuencia aunque sea menor implicará un incremento del error no presente en los datos iniciales.
En la figura 7.3 se presenta un remuestreo sobre la figura 7.1 donde para dos casos la nueva muestra se ubica en el peor caso posible.

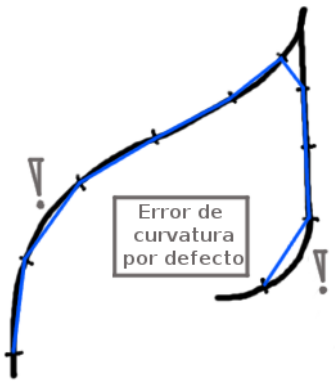


Figura 7.1: Ejemplificación de una interpolación lineal

Figura 7.2: Ejemplificación de una interpolación cúbica (exageración).

Figura 7.3: Ejemplo de remuestreo sobre la figura 7.1

A modo de resumen, se enumeran las vías tomadas:

- Sincronización por marco temporal.
- Emparejamiento en el dominio de origen.
- Emparejamiento sobre el mayor periodo.
- Interpolación lineal para las posiciones y lineal-esférica para las orientaciones.

7.1.5. Marco espacial

Toda la problemática anterior está ligada al ajuste temporal entre dos fuentes de datos. Sin embargo, atendiendo específicamente a la información de localización y a cómo la proveen los algoritmos de VisualSLAM nos encontraremos ante un nuevo problema: los dos tracks están representados en sistemas de coordenadas diferentes.
Recordemos que una de las limitaciones de estos algoritmos es que son agnósticos ante la escala a menos que utilice algún artificio en su inicialización. Esto mismo ocurre con el origen de coordenadas, que salvo configuración explícita se asume relativo al dispositivo de captura.

Ambos hechos implican que existe una disparidad entre ambos tracks, la cual puede ser en traslación, en rotación y/o en escala. Por tanto será necesario realizar un ajuste espacial.

7.2. Diseño

Para crear la librería de análisis se ha optado por un diseño modular. Para ello se ha segmentado en 4 + 2 módulos. En la ilustración 7.4 se puede ver una imagen general de este programa concreto en el que quedan resumidos sus puntos clave. A continuación describiremos brevemente cada uno de ellos:

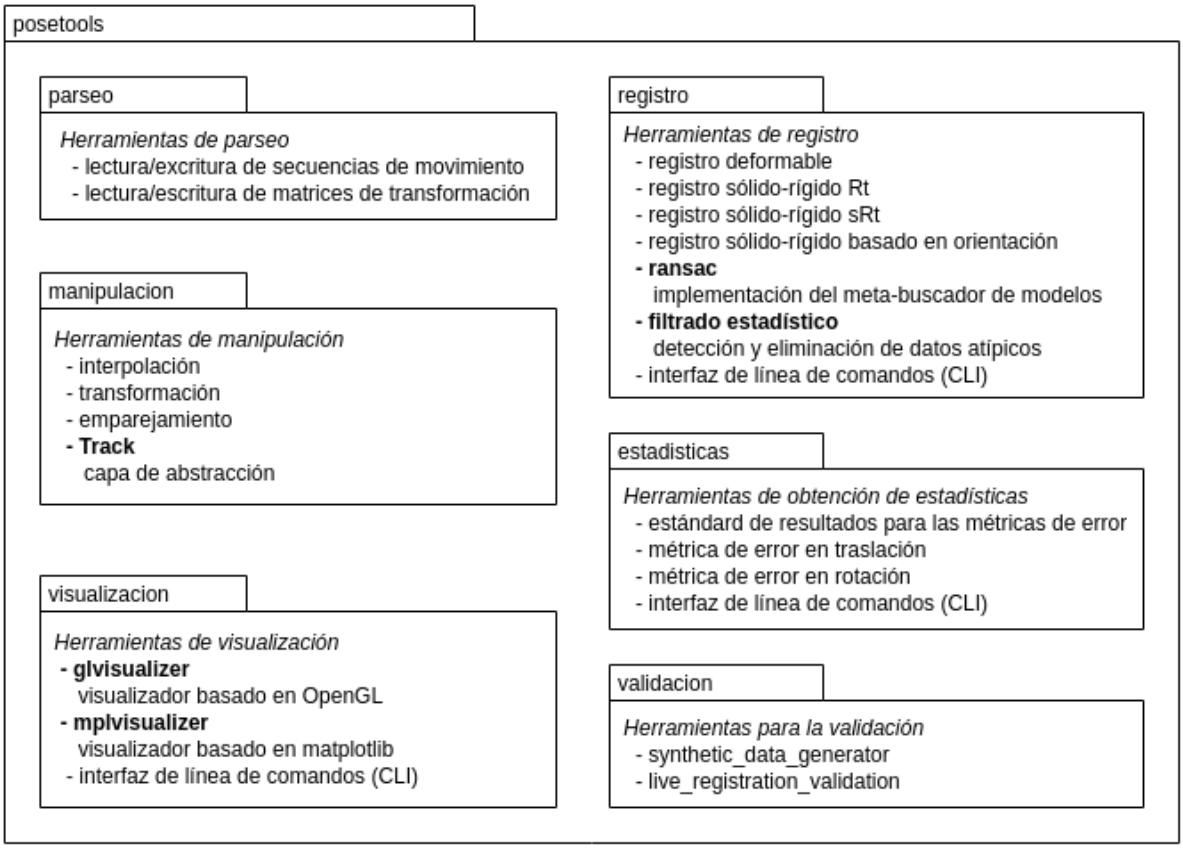


Figura 7.4: Diagrama de paquetes

Adquisición (parseo)
Este módulo se encargará de cargar los ficheros de datos y salvar los cambios realizados. De este modo definimos la frontera que permite separar y desacoplar el formato de entrada de la representación interna.
Finalmente, nos permitirá crecer en compatibilidad con otros formatos.

Manipulación
Bajo este módulo se situarán todas las operaciones que implican alterar y manipular los datos, que van desde la simple selección de segmentos del *track* hasta la interpolación, el remuestreo y la aplicación de las transformaciones geométricas.

Registro
El registro entre dos *tracks* es el elemento central de esta herramienta. Es el que se encarga de realizar el ajuste espacial entre dos *tracks*.

Visualización
Bajo este módulo se situará el código que permite la visualización de los tracks. En el plan

de trabajo ha sido el segundo módulo en desarrollarse ya que no solo es una herramienta de usuario, sino que permite detectar de forma gráfica si la lógica implementada es incorrecta.

Estadísticas
Este módulo surgió como necesidad en las iteraciones posteriores de desarrollo. En ella se han situado las métricas y los estudios estadísticos.
El objetivo de este módulo no ha sido sólo la reutilización de código común, sino definir unos estándares y una plataforma común que soporte la diversidad de implementaciones plausibles para el módulo de registro.

Validación
Este módulo constituye un *test* del sistema. Las partes más críticas han sido validadas con pruebas puntuales o con definición de tests unitarios. Sin embargo, la complejidad en la verificación del módulo de registro tuvo como consecuencia inevitable la definición de un marco de pruebas único que asegurase la integridad del test y de los módulos de manipulación y registro.

Con la arquitectura elemental planteada de forma abstracta, en las siguientes secciones se procederá a darle forma y contenido.

7.3. Adquisición

Definir qué formato será en que deban presentar los datos a tratar caracterizará todo el proyecto. No sólo atañe al formato de entrada y salida, sino que también podrá afectar a la representación interna del programa.
Existen varias alternativas cada una con sus ventajas y desventajas. A continuación se enumeran algunas de las propuestas.

La primera opción es codificarlo como *un punto 3D más un cuaternión*. Esta predisposición se debe a la experiencia previa con ROS, donde se emplea este formato. Su representación requiere 7 variables y permite trabajar ambos valores de forma independiente.
El orden y la semántica en la aplicación de ambos factores lleva a resultados dispares, por tanto es necesario definir dicha semántica. Siguiendo el mismo esquema que ROS, primero se aplica la traslación, de modo que ésta queda representada en el sistema de referencia maestro o canónico. En segundo lugar se aplica la rotación, pero con el único objetivo de obtener la base de coordenadas que definirá el subespacio en ese marco de referencia.
Por lo que si no existen puntos definidos en ese marco de referencia, bastará con aplicar la traslación.

La segunda opción es trabajar directamente con la *matriz de transformación*. Al tratarse de una transformación rígida, basta con 11 parámetros, la matriz de 3x3 que define la transformación más el vector de traslación. Este formato tiene la ventaja de que no necesita ninguna conversión para poder encadenar transformaciones.

Existe una representación compacta de una rotación denominada *rotation axis*. Es una representación de 3 valores que definen el eje de rotación, siendo su ratio de magnitud con respecto al vector unitario la magnitud de la rotación. Sólo requiere 6 valores, sin embargo requiere su traducción a otro formato para ser aplicado.

Por mencionar una última, se podrían emplear los 6 *parámetros de la transformación rígida*. Sin embargo, esta representación suele ser una de las primeras en ser descartadas

debido a que requiere el conocimiento explícito de en qué orden se aplican las rotaciones, XYZ o ZYX.

Tras analizar los pros y los contras, se ha escogido como formato la tupla <*punto 3D + cuaternion*> por las siguientes razones fundamentales:

1. Permite una representación compacta y directa.
2. Las variables están desacopladas.
3. No es necesario definir en qué orden se aplican las rotaciones.
4. No sufre de gimball lock.
5. Es una representación inequívoca.

Además, es el formato de texto elegido por TUM que consta de los 8 campos: $timestamp$, $tx, ty, tz, qx, qy, qz, qw$. Por lo que podremos beneficiarnos de tener una representación interna y externa común y compatible con TUM y ROS.

La única discrepancia posible sería el orden en que se define el cuaternión, w, x, y, z ó x, y, z, w. La primera coincide con su representación matemática más común, sin embargo, en el ámbito informático la representación suele ser la segunda por cuestiones de alineamiento en memoria y retrocompatibilidad con las librerías matemáticas multidimensionales.

7.4. Manipulación

Todas las decisiones teóricas de diseño relativas a la manipulación del track han sido presentadas con anterioridad[3]. Del mismo modo ha quedado definida su representación[4]. Por lo tanto, en esta sección solo tendremos que definir cuál será el diseño que hará efectiva la lógica que rige este módulo. Para ello se ha decidido hacer un diseño centralizado en el *objeto Track*, de modo que se pueda emplear la notación ".", pero cuya lógica se trata de manera exógena con pequeños módulos tal y como se presenta en el diagrama de clases de la figura 7.5.

Dicha clase implementa toda la lógica elemental necesaria para su manipulación:

- Corregir el posible decalaje u offset temporal.
- Recortar a un marco de tiempo.
- Remuestrear a una frecuencia arbitraria no homogénea.
- Interpolar la información posición y orientación.
- Cargar el track y salvarlo a un fichero.
- Aplicar una transformación geométrica.

[3] secciones 7.1.2, 7.1.3, 7.1.4 y 7.1.5
[4] sección 7.3

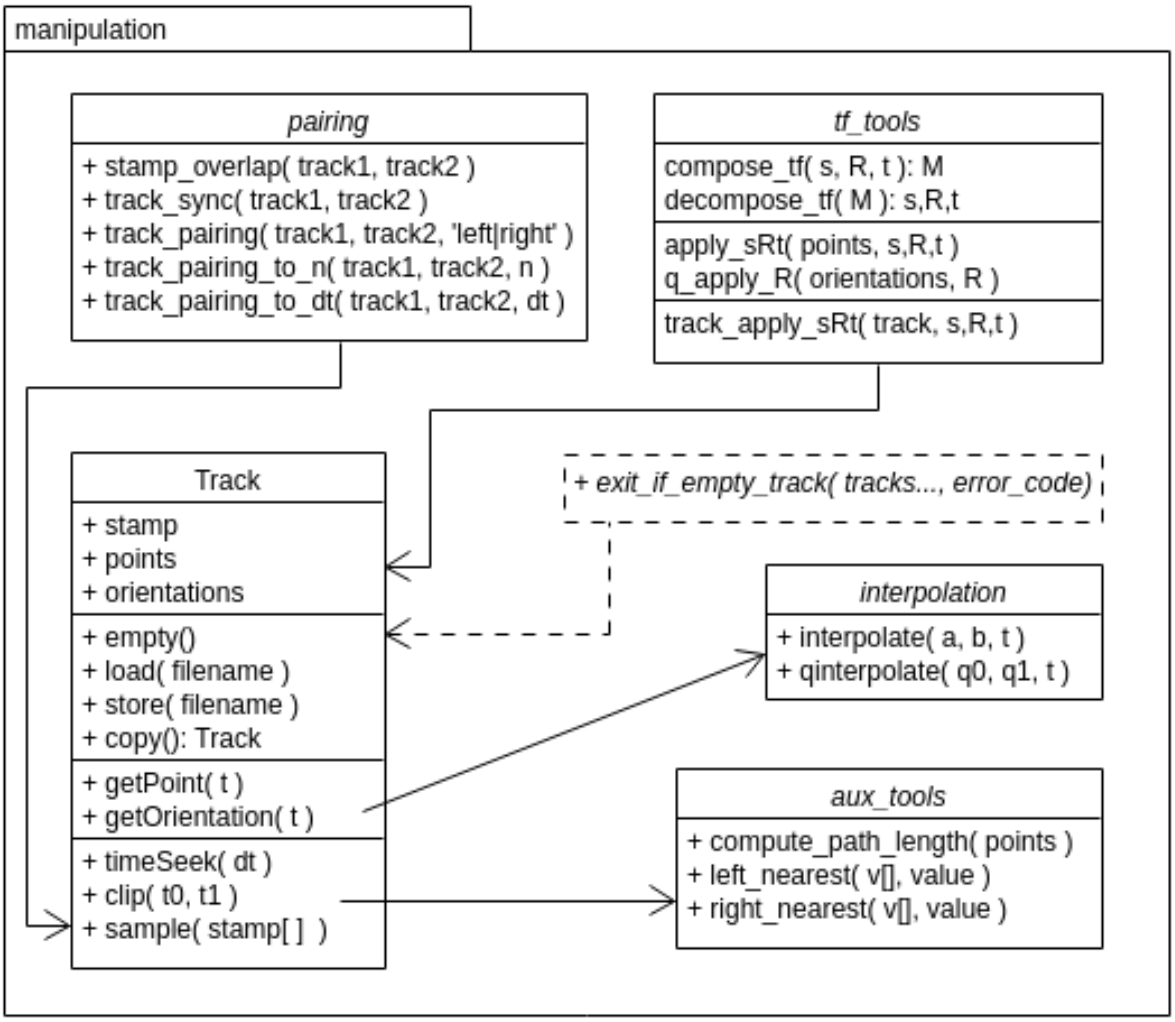


Figura 7.5: Diagrama de clases del módulo de manipulación

7.5. Registro

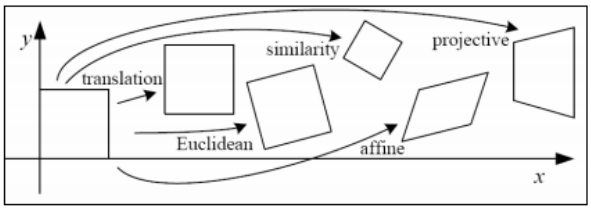
El módulo de registro es el encargado de encontrar la transformación que corrige la disparidad existente entre los sistemas de referencia de dos tracks. A la estimación de esta transformación entre dos conjuntos de datos se le denomina registro, el cual formalizaremos en las siguientes secciones.

7.5.1. Estimación de la transformación rígida

En este apartado se explicará el método de estimación de la transformación rígida empleado. Para ello lo dividiremos en tres apartados, correspondiendo la tercera a la explicación matemática del método.

Introducción

El algoritmo de registro es el encargado de encontrar la aplicación que permite mapear los datos entre los dos espacios. Esto se traduce en encontrar la transformación geométrica que permite transformar los puntos dados en un sistema de referencia arbitrario al sistema de referencia definido por el *groundtruth*.



Informalmente, se trata de obtener los 15 valores de la matriz de transformación (7.1). No obstante el problema es más acotado. Buscamos una transformación de similaridad, es decir, una transformación euclídea, también denominada rígida, más un cambio de escala. Por tanto, podemos reducir el problema a 11 valores (7.2).

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & 1 \end{pmatrix} \qquad (7.1) \qquad\qquad \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & 1 \end{pmatrix} \qquad (7.2)$$

Sin embargo, encontrar la solución a este problemas no es trivial. El espacio de las transformaciones geométricas tiene una problemática y limitaciones asociadas:

- *Variables acopladas*
  Las 11 variables que definen una transformación no son linealmente independientes, sino que son una combinación de los parámetros que definen una transformación.
- *Variables subyacentes*
  Los parámetros reales de una transformación, como los ángulos de rotación, la escala, etc se encuentran combinadas bajo las 11 variables, con excepción de la traslación. Bajo estas 11 variables la formalización, o extracción, de dichos parámetros conforma en algunos casos un sistema de ecuaciones no lineal. Este hecho nos impedirá resolver el problema con un método de optimización total lineal.
- *Espacio no derivativo*
  Estamos ante un espacio no derivativo, por lo tanto no se pueden aplicar métodos de descenso de gradiente propiamente dichos. A pesar de ello, se pueden llegar a aplicar métodos de descenso de gradiente basados en una función heurística, de modo que dicho descenso se realiza en un espacio artificial alternativo. En este caso, para cada iteración se deberían explorar un total de 14 direcciones y sus combinaciones.

Precisamente, el hecho de restringir el problema a una transformación rígida de 7 parámetros (3 ángulos, 3 rotaciones y 1 de escala) es lo que impide resolver el problema por optimización total. A parte de estos 7 parámetros, las 11 variables a optimizar modelan el espacio afín, donde se añade un parámetro más: *el zizallado*. Por tanto, estaríamos incumplimiento una de las restricciones del problema.

Sin embargo, fraccionando el problema podremos replantearlo como un problema de optimización lineal tal y como veremos en la siguiente sección. Evitando de este modo algoritmos iterativos consiguiendo un reducido coste computacional.

Antecedentes

Acudiendo a la literatura, en [ELF97][5] podemos encontrar una comparativa de cuatro de los algoritmos más populares para la estimación de una transformación rígida 3D.
Los algoritmos comparados son SVD (descomposición matricial), OM (Matrices ortonormales), UQ (unit quaternions) y DQ (dual quaternions). En este documento se llega a la conclusión de que el método SVD [Hor87], aunque no es el más rápido, es el más estable, por lo que es el candidato escogido para realizar el registro.

Método de orientación absoluta

El método propuesto por [Hor87] trunca el problema en dos partes.

1. En primer lugar, estima la traslación en base al centroide de los datos.
2. En segundo lugar, estima la rotación mediante minimización del residuo por SVD.

Para ello el problema se ha relajado asumiendo la siguiente premisa:
Las dos colecciones de puntos a comparar están en un espacio cerrado y son lo suficientemente representativas como para que sus centroides sean equivalentes. De este modo, la estimación de la traslación queda resuelta apriori.

Además el método tiene dos generalizaciones:

a. Estimación de la escala: está basado en la proyección radial, es decir, en el ratio de productos escalares.
b. Contribución asimétrica de los puntos: los emparejamientos de puntos pueden tener un peso asociado permitiendo lógica borrosa y la inclusión de ransac de manera implícita.

A continuación se describe la matemática involucrada:

Sean r_l y r_r dos colecciones de puntos representadas en sistemas de coordenadas diferentes tomadas como izquierda y derecha respectivamente. Sea $r_{l,i}, r_{r,i}$ el emparejamiento en i de dichas colecciones.
Tendremos que $r_r = sR(r_l) + r_0$ denota la transformación del sistema de coordenadas de la izquierda al de la derecha, con s como factor de escala, R como rotación y r_0 como traslación. De modo que los parámetros s, R y r_0 son los que minimizan el error residual $e_i = r_{r,i} - sR(r_l, i) + r_0$, ya que para un conjunto de datos imperfecto, la ecuación $r_{r,i} - sR(r_l, i) + r_0 = 0$ no se satisface.

[5] también disponible en su trabajo previo [LEF95]

$$\sum_{i=1}^{n} \|e_i\|^2 = \sum_{i=1}^{n} \left\| r_{r,i} - sR(r_l, i) + r_0 \right\|^2 \tag{7.3}$$

$$\sum_{i=1}^{n} \left\| r_{r,i} - sR(r_l, i) \right\|^2 - 2r_0 \cdot \sum_{i=1}^{n} \left\| r_{r,i} - sR(r_l, i) + r_0 \right\|^2 + n \cdot \|r_0\|^2 \tag{7.4}$$

Sea 7.4 la ecuación expandida de 7.3, el primer término se queda solo si $r0 = 0$. Por ello, ahora se procede a desacoplar la traslación utilizando la premisa del centroide.

Sean $\bar{r}_l, \bar{r}_r$ los centroides de ambos conjuntos 7.5.
Sean $r'_{r,i} = r_{r,i} - \bar{r}_r$ y $r'_{l,i} = r_{l,i} - \bar{r}_l$ los conjuntos centrados.
Se obtiene que $r'_0 = 0$ para la ecuación 7.6, resultando en 7.7.

$$\bar{r}_l = \frac{1}{n} \sum_{i=1}^{n} r_{l,i} \qquad \bar{r}_l = \frac{1}{n} \sum_{i=1}^{n} r_{l,i} \tag{7.5}$$

$$e'_i = r'_{r,i} - sR(r'_l, i) + r'_0 \tag{7.6}$$

$$\sum_{i=1}^{n} = \left\| r'_{r,i} - sR(r'_l, i) \right\|^2 \tag{7.7}$$

Por último, para calcular la escala, s, se expande otra vez 7.7 teniendo en cuenta que la rotación no afecta a la magnitud de los vectores, es decir, $\left\| R(r'_{l,i}) \right\|^2 = \left\| r'_{l,i} \right\|^2$, resultando en la ecuación 7.8

$$s = \left(\sum_{i=1}^{n} \left\| r'_{r,i} \right\|^2 \Big/ \sum_{i=1}^{n} \left\| r'_{l,i} \right\|^2 \right)^{1/2} \tag{7.8}$$

Como nota final, destacar que el orden de las ecuaciones se puede invertir si los datos respetan ciertas propiedades y la formulación de 7.7 sin el factor de escala es equivalente. En ese caso, la escala se calcularía a posteriori 7.9, que es un paso intermedio en el desarrollo de 7.8.

$$s = \sum_{i=1}^{n} r'_{r,i} \cdot R(r\,l, i) \Big/ \sum_{i=1}^{n} \left\| r'_{l,i} \right\|^2 \tag{7.9}$$

La diferencia entre las fórmulas (7.9) y (7.8) se puede observar mejor en la figura 7.6.

I. El cálculo de la escala a priori (7.9), emplea las magnitudes de los vectores radiales $\vec{v}_i = |p_i - c|$.

II. El cálculo de la escala a posteriori (7.8), realiza la proyección del punto estimado p'_i sobre p_i.

En el segundo caso, los errores de estimación de la rotación, es decir, el residuo $\theta_r = \theta - \theta'$, se transfieren a la escala. Sin embargo, permite estimar la escala de forma aislada por cada eje. En el primer caso no se puede dar, ya que al no compartir la misma base no se puede descomponer el vector.

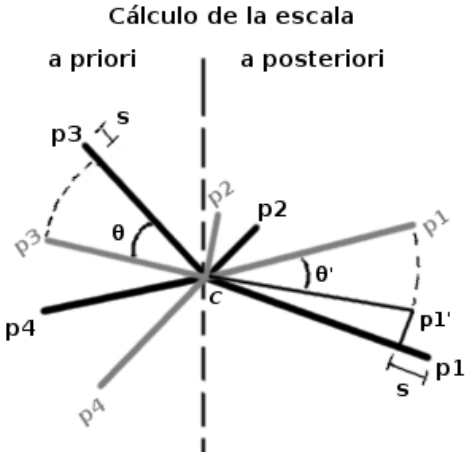


Figura 7.6: Métodos a priori y a posteriori de cálculo de la escala

7.5.2. Estimación robusta

Debido a que la estimación proporcionada por los algoritmos de VisualSLAM no es perfecta, ésta contará con errores de precisión y además con errores de estimación. Este segundo caso es el problemático ya que implica la aparición de datos que no se ajustan a la información de localización subyacente, es decir, datos espúreos.

Los datos espúreos o atípicos pervierten el modelo estimado por métodos que asumen que todos los puntos pertenecen al modelo como en el caso de 7.5.1. Por ello es necesario la inclusión de algún método que otorgue resistencia ante este tipo de situaciones.
Para llevar a cabo esta tarea se han planteado dos opciones, una que detecta y descarta los datos atípicos de la distribución subyacente a los datos y otra que emplea técnicas de consenso para definir y diferenciar el modelo subyacente.

7.5.2.1. Algoritmo de detección de espúreos

Esta primera técnica se basa en la teoría estadística para la detección de los datos atípicos. Existen múltiples opciones para detectar datos atípicos [Wik16c], pero se ha elegido la conjunción de estos dos métodos:

a) Distancia de Mahalanobis [Wik16b]

b) IQD (Interquartile Deviation Method) [Ora12] [Wik16a]

La distancia de Mahalanobis ha sido escogida por dos factores:

1. Implica la misma formulación, es decir, cálculo de covarianzas, para calcular la distancia y la mejor solución por mínimos cuadrados.
2. Explica mejor la no uniformidad del eje de profundidad con respeto a los otros dos ejes.

Del mismo modo, IQD o método de desviación intercuartílico ha sido elegido frente a un método más simple como la desviación estándar (STD) porque este segundo se ve tremendamente afectado por los datos atípicos de magnitud infinita.
Además IQD nos permite diferenciar los datos atípicos en dos grupos:

- atípicos de pequeña magnitud (IQD=1.5): se deben a errores de precisión en el *tracking*.
- atípicos de gran magnitud (IQD=3): son los errores de estimación en el *tracking*. No están acotados pudiendo alejarse en varios órdenes de magnitud.

Por motivos de comparativas y casos especiales donde la deformación del espacio euclídeo no sea deseable, se ha planteado la posibilidad de realizar la distancia de Mahalanobis o la euclídea de forma indistinta. A continuación se presenta el pseudocódigo de los pasos

que implica este método:

```
1. Model estimation
2A. Do mahalanobis distance
    2.1. Compute mean and covariance
    2.2. Compute distances
2B. Compute euclidean distance
3. Use IQD to determine outliers
4. Drop out of bounds values
```

7.5.2.2. RANSAC

RANSAC es un método iterativo no determinista para estimar el modelo subyacente a un conjunto de datos que contiene atípicos. Ha sido estudiado en el Máster de Visión Artificial [AB15], aunque para el lector, dos rápidas referencias bibliográficas son [Der10] y [Zul09].

En términos generales, el pseudocódigo del método está representado en 7.1, donde nos encontraremos tres parámetros libres: N, K, T.

Código 7.1: Pseudocódigo de RANSAC

```
1. Select randomly the minimum number of points required to determine the
   model parameters.
2. Solve for the parameters of the model.
3. Determine how many points from the set of all points fit with a
   predefined tolerance E (ConsensusSet).
4. If the fraction of the number of inliers over the total number points in
   the set exceeds a predefined threshold K, re-estimate the model
   parameters using all the identified inliers and terminate.
5. Otherwise, repeat steps 1 through 4 (maximum of N times).
```

El parámetro N es el más fácil de determinar ya que es un parámetro que no depende de los datos *per se* y cuyo valor viene dado por la fórmula 7.10, donde p es el grado de certeza que se quiere imponer, m es el número de datos necesarios para construir el modelo y v es la probabilidad de seleccionar un outlier.

$$N = log(1-p)/log(1-(1-v)^m) \tag{7.10}$$

El parámetro K puede establecerse con la fórmula 7.11 si el porcentaje de outliers, v, se conoce a priori, donde n es el tamaño de los datos. En ese caso se define además la confianza en el consenso a través del parámetro q. Si la confianza es máxima se define $q = 1$, y su valor será menor si el ruido existente en los datos reduce la capacidad del consenso.

$$K = \lceil (1-v) \cdot n \cdot q \rceil \tag{7.11}$$

Por último el parámetro T, que es el umbral para determinar si un dato es inlier y por tanto constituye el consenso, es dependiente a la magnitud de los datos y al ruido que los acompaña. Suele ser un valor definido empíricamente aunque existen métodos para definirlo de forma implícita.
Este parámetro se verá más adelante cuando se hablen de las implementaciones concretas realizadas.

3-RANSAC es una formulación genérica del algoritmo RANSAC en tres puntos que surgió como necesidad tras implementar RANSAC clásico con el objetivo de soportar una implementación genérica que admitiera la inclusión de las diferentes variantes propuestas y que fuera completamente ajena al tratamiento de los datos. Su única restricción es que está planteado como un método de minimización en un dominio positivo, siendo el cero el valor de calidad supremo. De este modo, la interacción de RANSAC con los datos se ha extraído según una taxonomía propia en tres puntos de actuación diferentes:

- Inferencia (*f_inference*): encierra el método de inferencia del modelo.
- Comparación (*f_compare*): encierra la lógica que permite comparar la adecuación del modelo.
- Aptitud (*f_fitness*): encierra la lógica de puntaje de un modelo.

Matemáticamente podemos formular 3-RANSAC:
Sean a y b dos conjuntos y exista una aplicación A donde $b = A(a)$.
Sea $a^{(i)}$ un subconjunto aleatorio de a.
Denotando ínf $f(x)$ como el ínfimo de la función $f(x)$, tal que ínf$(f(x)) = \sup(1/f(x))$.
Entonces:

$$\begin{aligned} A^* &= \inf_{A^* \in \Omega} f(d(a, b, A^*)) \\ A_i^* &= g(a^{(i)}, b^{(i)}) \in \Omega \\ a^{(i)} &\subset a, \quad b^{(i)} \subset b \end{aligned} \tag{7.12}$$

En la literatura, normalmente nos encontraremos dicha lógica dividida en dos partes: la parte de inferencia *f_inference*, que se corresponde con el paso 2, y la de evaluación *f_fitness(f_compare(x))*, que se corresponde con el paso 3. Esta división en dos es suficiente, sin embargo la motivación de realizar esta partición en tres componentes es doble:

1. El primer objetivo es reforzar la coherencia y destacar los posibles errores de concepto al emplear el meta-buscador.
2. El segundo objetivo es la reutilización, minimizar el código repetido y otorgar a cada función una semántica más rica y ajustada.

Veámoslo con un ejemplo:

```
f_inference = rigid_transform_solver.rigid_transform_sRt(_from, _to)
f_compare = rigid_transform_solver.calculate_error(_from, _to, *sRt, mode='
    dist')
f_fitness = ransac.instance_fitness_saturated_error(threshold=0.100)
```

En este fragmento de código podemos ver que:

1. Estamos aproximando el modelo con una transformación rígida.
2. La función de comparación es la encargada de realizar el mapeo de los datos. Es decir, aplicar el modelo inferido a los datos para que sean comparables.
3. La función de comparación elige en qué espacio se comparan ambos conjuntos de datos.

4. La función de fitness es la que absorbe el concepto de umbral de ransac. Por tanto, las dos anteriores son completamente ajenas a su implicación en ransac.
5. La función de fitness es ajena a los datos y al modelo, lo que permite implementar cualquier método de evaluación y ser aplicados a diferentes datos y modelos.

Gracias a ello, la función de fitness puede absorber las dos variantes elementales de evaluación: conteo de inliers 7.13 y suma truncada del error 7.14.

$$f(x) = \begin{cases} 1 & , \quad x_i \leq \varepsilon \\ 0 & , \quad x_i > \varepsilon \end{cases} \qquad (7.13) \qquad f(x) = \begin{cases} x_i & , \quad x_i \leq \varepsilon \\ \varepsilon & , \quad x_i > \varepsilon \end{cases} \qquad (7.14)$$

Formalización y variantes de la función de aptitud

Habiendo presentado RANSAC, es el momento de definir cuál será la lógica concreta bajo las tres funciones definidas:

- Como función de inferencia se empleará el método definido en el capítulo 7.5.1.
- Como función de comparación se empleará (7.15), es decir, la distancia euclídea entre cada par de puntos al ser lo que se está minimizando en la función de inferencia.
- Como función de aptitud, hemos visto (7.13) y (7.14), donde emplearemos (7.16) de las variantes que definiremos a continuación.

$$d(a,b,M)_i = \begin{array}{l} \|M \cdot a_i - b_i\|_2 \\ a_i, b_i \in \mathbb{R}^3 \\ M \in SE(3) \end{array} \qquad (7.15)$$

La función (7.13) corresponde a la planteada en el método original de RANSAC [FB81], donde si $f(x) > t$ entonces se ha alcanzado una solución satisfactoria. Sin embargo su formalización se deja bastante abierta al igual que en otros puntos críticos del método donde ya se plantean métodos y alternativas para mejorar sus resultados.

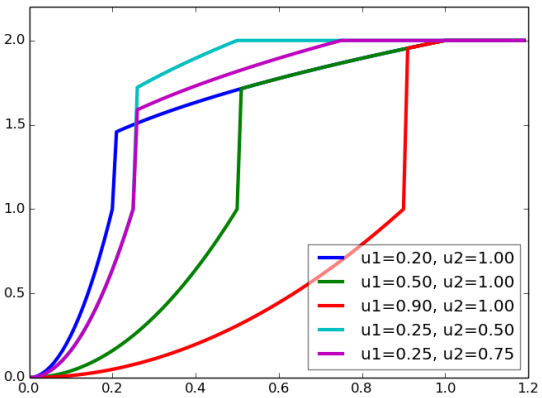
Esto ha dado pié a otros criterios, como por ejemplo (7.14), el cual arregla el primer problema que plantea la función original. Al tratarse de una función binaria, cualquier información adicional de la que dispongamos se pierde. De modo que tiene un comportamiento *FIFO* que hace que se descarten soluciones con el mismo número de inliers aunque sean de mayor calidad.
Por ese motivo en (7.14) se efectúa una ponderación de los inliers que permite la ordenación de soluciones de igual calidad binaria. En definitiva, es un método más informado.

Con esto *in mente*, podemos plantear una variante cuya ponderación sea más representativa para el problema que hemos definido. Así surge (7.16), donde la ponderación es exponencial para que la ordenación sea afectada no solo individualmente, sino también colectivamente. Además normaliza en rango y en número de elementos para obtener un valor que sea representativo puntualmente. De este modo, un valor de 1 reflejaría que no se ha obtenido un modelo que satisfaga las condiciones impuestas, siendo 0 la perfección.

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\lfloor d(i) \rfloor_u}{u}\right)^2 \quad , \quad \begin{array}{l} d(i) \in [0,\infty) \\ u \in (0,\infty) \\ f(x) \in [0,1] \end{array} \qquad (7.16)$$

Como nota final, destacar que el Tutor me hizo conocedor de que el umbral que se define se puede desdoblar, de modo que el dominio de la función no tiene por qué estar acotado por el valor de tolerancia del error.
Esto permite tener un límite duro, que será el que se tenga en cuenta en el criterio de parada y un límite blando que agrega un grado más de ordenación que será útil si el método llega a su límite de iteraciones.

De este modo, y tomando como base (7.16), se llegó a la función (7.17) que, sin ser pretencioso, me gustaría tildar de estado del arte. En ella podemos ver un primer tramo no lineal que fomenta la ordenación colectiva. El segundo tramo corresponde a la penalización que separa la frontera entre inlier y outlier. El último tramo es una aproximación no asintótica al límite del dominio.



$$f(x) = \begin{cases} (x/u_1)^2 & , \quad x \leq u_1 \quad \in [0,1] \\ 1 + \sqrt{x/u_2} & , \quad x \leq u_2 \quad \in [1,2] \\ 2 & , \quad x > u_2 \end{cases} \tag{7.17}$$

En esta sección se han descrito los elementos que conforman la solución elegida para el módulo de registro. De este modo, el método de registro realizará en primer lugar un filtrado de espúreos mediante el mecanismo presentado en 7.5.2.1 (IQD+Mahalanobis) para reducir la cota de número de ejecuciones de RANSAC. Tras ello se realiza la búsqueda del modelo subyacente con RANSAC empleando como función de inferencia el método presentado en 7.5.1 (Método de orientación absoluta), como función de comparación (7.15) (distancia euclídea entre puntos) y como función de aptitud (7.16).

Esta elección se apoya en un estudio empírico realizado en este mismo trabajo para determinar cuál es el método de registro de los que ofrece la librearía realizada que mejor se comporta en nuestro escenario. Este estudio puede encontrarse en el Anexo A.

7.6. Visualización

La visualización ha sido llevada a cabo empleando dos librerías diferentes, OpenGL (glvisualizer) y matplotlib (mplvisualizer), cada una con sus pros y sus contras.

OpenGL
Apoyarse en OpenGL implica realizar a mano todos los componentes gráficos, desde el pintado del track hasta la leyenda incluyendo la lógica que rige los movimientos de la cámara. Ello ha sido realizado a través de los binding de OpenGl y glut.

matplotlib
Python cuenta con una librería muy completa para la visualización de todo tipo de gráficas llamada matplotlib.
En términos profanos se puede decir que matplotlib es la API de generación de gráficas de Matlab llevada a Python. Esto se debe a que se diseñó para que científicos y personas no versadas en programación pudieran migrar de Matlab a Python de forma cómoda.

La ventaja de matplotlib es que deja resuelta toda la lógica de visualización. Sin embargo su capacidad de escalar y renderizar artefactos complejos es limitada llegando a perder su capacidad de renderizacion en tiempo real y afectando a la interacción con el usuario.

La ventaja de OpenGL es que ofrece libertad total y para cosas complejas y que requieran una cámara de movimiento libre es la alternativa a seguir. En dicho caso se insta a emplear una librería de más alto nivel como pygame o pcl.

Finalmente, en las figuras 7.7 y 7.8 se muestran los resultados de ambas herramientas de visualización. En ellas se muestran varios *tracks* escogidos arbitrariamente. Cada color representa un *track*, donde su secuencia de posiciones se muestra en forma de línea y la orientación de cada posición se muestra como un segmento representando el eje de profundidad.

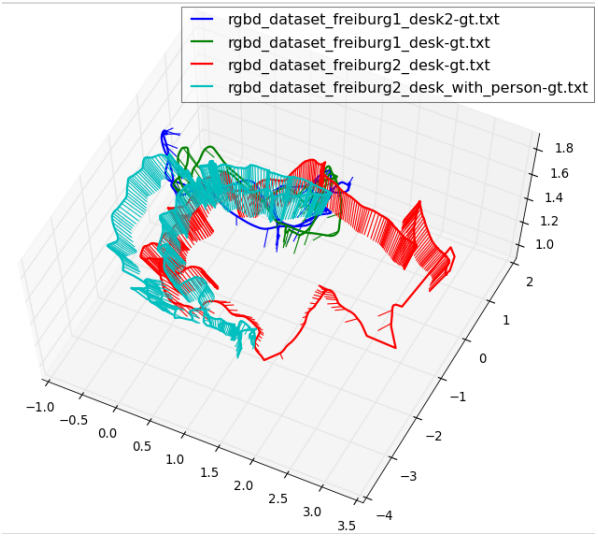


Figura 7.7: mplvisualizer

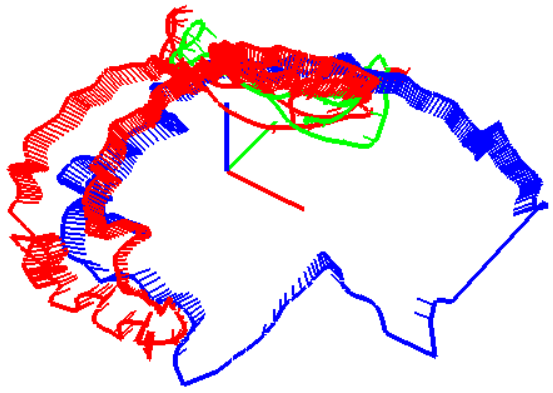


Figura 7.8: glvisualizer

7.7. Estadísticas

Este módulo ha sido diseñado para estandarizar y exponer las métricas de error usadas por la propia librería. Además define facilidades para su evaluación estadística. El núcleo de estas simplificaciones se presenta en las figuras 7.9 y 7.10.

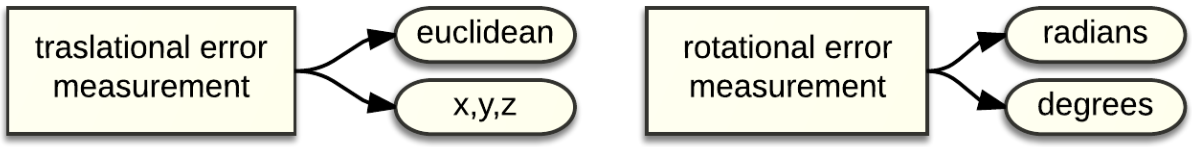


Figura 7.9: Representación de las opciones para medir el error

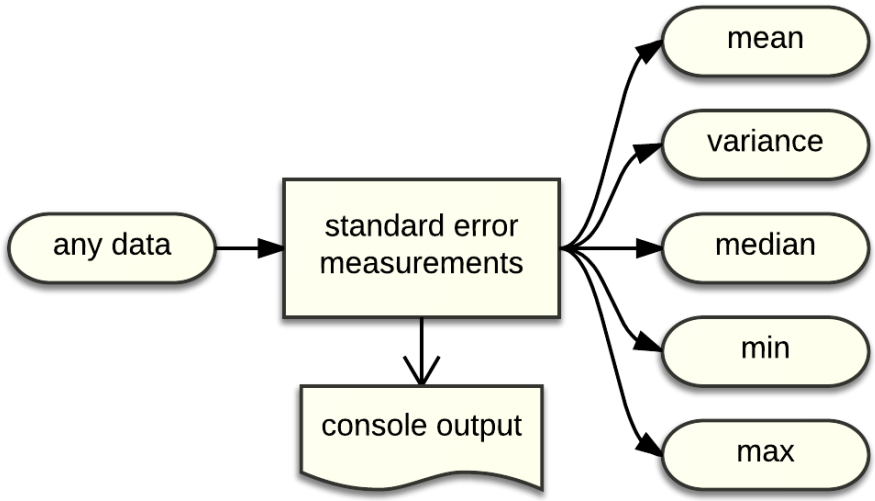


Figura 7.10: API para la obtención de estadísticas

Gracias a esta decisión las aplicaciones que realicemos después podrán beneficiarse de esta capa de abstracción. De este modo se podrá asumir que siempre existirán esas cinco métricas estadísticas cuando se proceda a hacer un análisis del error.
Además, como se puede intuir en la representación, incluye un método para poder imprimir los valores de forma bonita y elegante sin tener que repetir ese código una y otra vez.

7.7.1. Validación

El último de los módulos de la librería de análisis es el de validación. La validación de la implementación ha sido aplicada a varios niveles y en varias etapas del proceso, ya sea con datos estáticos bien definidos o con datos dinámicos que cumplen ciertas restricciones.
Este módulo aglutina y unifica la definición de los datos dinámicos, dando un contexto único y una API para su posterior uso. Por lo que estamos ante una librería para generar datos sintéticos de forma aleatoria.

7.7.1.1. Datos sintéticos

Los datos sintéticos son datos diseñados artificialmente cuyo objetivo es proveer un entorno completamente controlado y ajustable, pudiendo hacer hincapié en ciertos aspectos del espacio sensorial. Estos datos pueden ser empleados como test unitarios para asegurar la correctitud del algoritmo así como su comportamiento en sucesivas mejoras y revisiones.

La colección más importante de datos sintéticos es la que incluye ruido, ya que es la que permitirá validar el módulo de registro. Esto requiere definir un modelo de ruido controlado. Para ello se han implementado tres tipos de ruido:

- **Ruido blanco o gaussiano**
  Se trata del modelo de ruido por excelencia al ser el que nos encontramos en los procesos naturales. Se caracteriza por ser una señal no correlada cuya función de densidad responde a una distribución normal. Se ha implementado como una distribución normal de varianza ajustable.
- **Ruido por datos espúreos**
  Este tipo de ruido sirve para modelar datos que no pertenecen al modelo. Se ha implementado como un offset de magnitud definida dentro de un rango. Su magnitud concreta queda definida por una distribución uniforme, así como su dirección.
- **Ruido cósmico**
  Este tipo de ruido se caracteriza por ser homogéneo en todo el dominio. También se le denomina ruido térmico. Se ha implementado como una distribución uniforme dentro del espacio de trabajo definido.

La diferencia entre el ruido blanco y cósmico es únicamente semántica. Ambos son señales no correladas con *psd* plana. Sin embargo se aplican de forma diferente.

I. El ruido blanco se aplica sobre los datos, alterando su percepción.

II. El ruido cósmico se aplica sobre el dominio y tiene la capacidad de hacer aflorar nuevos datos, por tanto puede verse como una forma estocástica de crear espúreos.

Para los tests con datos sintéticos solo van a emplearse el ruido gaussiano y datos espúreos debido a que son condición suficiente para tener un modelo de ruido completo, acotado y controlado. Su implementación ha sido llevada acabo mediante el uso de cuaterniones y su definición se ha realizado siguiendo una serie de reglas para asemejarnos con los futuros datos reales:

- sucesión de puntos en el espacio.
- distancia entre puntos arbitraria pero acotada.
- sin restricciones de orientación.
- variación de orientación acotada y con capacidad de generar lineas rectas.

7.7.1.2. Test del sistema

La constatación de este módulo puede encontrarse en el test del sistema denominado *live_registration_validation*, el cual actúa como herramienta de validación interactiva del módulo de registro.
En ella se realiza una experimentación no exhaustiva en la que se generan tracks aleatorios y tras aplicar transformaciones y ruido se comprueba si el algoritmo de registro es capaz de recuperar la transformación aplicada. Podemos ver un ejemplo de la ejecución de este programa en el diario web [6] [7].

[6] `http://jderobot.org/Varribas-tfm/posetools/live_algorithm_validation2`
[7] `http://jderobot.org/store/varribas/uploads/recursos-memoria/live2-gaussian-ls.mp4`

7.8. Interfaz de línea de comandos

En el momento de escribir esta memoria las APIs de alto nivel identificadas, y que por tanto serán usadas por el usuario así como por el capítulo 9, son tres:

```
mplvisualizer.py [-h] [-k ORIENTATION] [--names]
                      output track_files [track_files ...]

positional arguments:
  output                plotting output. Filename to save it or '-' to show
                        it.
  track_files           track's filenames
optional arguments:
  -k ORIENTATION, --orientation ORIENTATION
                        -1 to disable it. show also orientation each k points.
  --names               legend will display track filename
```

```
track_register.py [-h]    [--store_tf STORE_TF]
                          [--method]
                          [--right_timestamp] [--right2left]
                          [--plot PLOT]
                           file_left file_right file_out

positional arguments:
  file_left             track filename
  file_right            track filename
  file_out              filename where registered track will be stored

optional arguments:
  -h, --help            show this help message and exit
  --store_tf STORE_TF   store guess transformation to given filename (use .tf
                        as preferred extension)
  --method              registration method to use
  --right_timestamp     by default timestamp is taken from left. Active to
                        reverse it
  --right2left          by default registration direction is left to right.
                        Active to reverse it
  --plot PLOT           enable plotting. Save to file of display it
```

```
error_measurement.py [-h] [-dt DT] [-N N]
                           [--use_tf USE_TF] [--write_to WRITE_TO]
                           [--right_timestamp] [--right2left]
                            file_left file_right

positional arguments:
  file_left             a track
  file_right            another track

optional arguments:
  -h, --help            show this help message and exit
  -dt DT                measure error each dt (in time)
  -N N                  measure error in only N points (time equidistant)
  --use_tf USE_TF       load predefined transform. Otherwais program will guess
                        it.
  --write_to WRITE_TO   file where write measurement results.
  --right_timestamp     by default, timestamp is taken from left track. Set to
                        reverse it.
  --right2left          by default, registration direction is left-to-right.
                        Set to reverse it.
```

CAPÍTULO

8 Herramienta de análisis

Gracias a la formalización realizada en los capítulos 7 y A, ya contamos con las herramientas necesarias para realizar la evaluación de los algoritmos de VisualSLAM.
En este capítulo definiremos la semántica y los criterios de evaluación que regirán las conclusiones. Para ello necesitamos tres elementos: la metodología o arquitectura de evaluación, las métricas que emplearemos para la evaluación, y el medio físico para presentar los resultados.

8.1. Arquitectura de evaluación

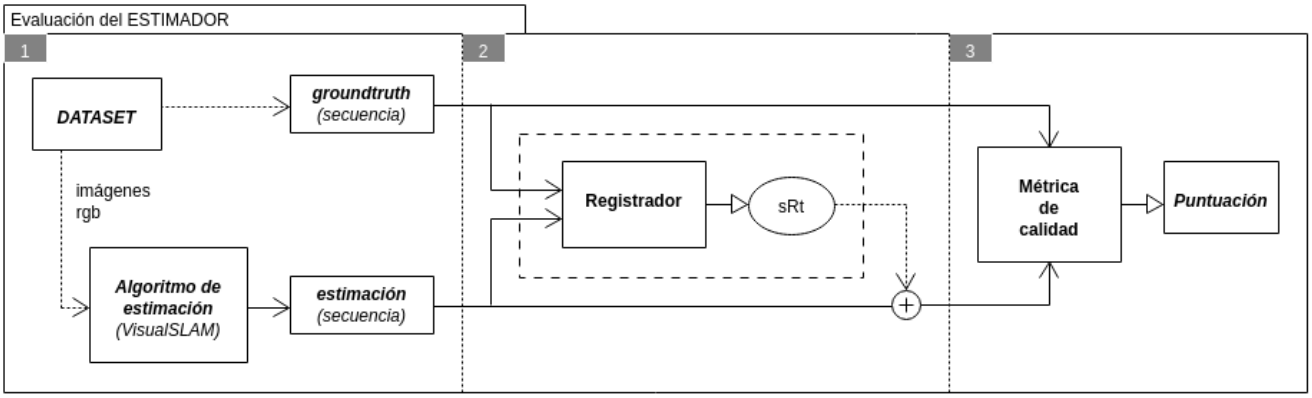


Figura 8.1: Evaluación del estimador

La ilustración 8.1 presenta la arquitectura de evaluación definida. Esta se encuentra dividida en tres partes:

(1) La primera etapa comprende la ejecución de un algoritmo de VisualSLAM, del cual tomaremos su estimación.

(2) La segunda etapa comprende el ajuste temporal y espacial descritos en el capítulo 7, de este modo obtendremos dos fuentes comparables de la información de la localización.

(3) La tercera etapa implica la evaluación de ambas fuentes, de donde obtendremos la métrica de comparación (requisito R4) además de otras métricas que nos ofrezcan una visión más informada (requisito R4.1).

8.2. Métricas establecidas

Este capítulo está destinado a la formalización de las métricas que se emplearán en la evaluación de los algoritmos de VisualSLAM.

M1. **Porcentaje de tiempo traqueado**
Es una medida que se ha planteado para el descarte. De este modo podremos determinar si una secuencia es conflictiva a priori, ya que solo necesita las marcas de tiempo para calcular su valor. Además, al tratarlo como un porcentaje, no dependeremos del número de muestras que presente cada secuencia, de modo que obtendremos una métrica relativa capaz de ser extrapolada a otras secuencias.

M2. **Porcentaje de tiempo traqueado con un mínimo de calidad**
Es el planteamiento a posteriori de la métrica anterior. Para esta es necesario realizar el registro entre la estimación y el groundtruth, ya que el dominio de trabajo es el error cometido punto a punto. Así podremos determinar cuántas estimaciones cumplen el criterio de precisión impuesto.

M3. **Métrica de calidad propia**
El valor de esta métrica es el que usaremos para determinar si un algoritmo es mejor que otro, el cual va de 0 a 5 siendo 5 la mejor puntuación posible. Se trata de la tercera revisión en el planteamiento de una métrica que satisfaga los criterios de calidad preconcebidos y que sea comparable.

M4. **Error euclídeo medio**
Es una métrica que no puede faltar ya que es la métrica estándar en este tipo de evaluaciones. La encontraremos en otras publicaciones así como en los rankings que proveen algunos datasets.

M5. **Tiempo de inicialización**
Con esta métrica obtendremos el retraso entre el inicio de la secuencia y la primera estimación.

M6. **Tiempo hasta convergencia**
Esta métrica complementa a la anterior. Entrega el retraso con respecto a la primera estimación por debajo de los criterios de calidad. Es decir, el retraso hasta que la estimación alcanza la precisión deseada.

Como simplificación de las métricas (M1) y (M2) podemos reducirlas a un simple conteo de elementos si las marcas de tiempo están sujetas a un periodo constante y existe una relación uno a uno entre los datos de entrada y la estimación. Esta simplicidad es la potencia de ambas métricas.

Para las métricas (M2), (M3) y (M6) es necesario definir unos valores mínimos de calidad. En caso de que ningún algoritmo cumpla las restricciones mínimas impuestas, se procederá a relajar las métricas definidas para ver en qué cota se mueve el estado del arte. Este suceso podrá identificarse gracias a la conjunción de dos métricas antecesoras de (M3).
Los valores de umbral o calidad elegidos serán a priori $u_1 = 1cm$ y $u_2 = 10cm$. En caso de que sean demasiado estrictos serán rebajados a $5/10cm$ o $5/20cm$, siendo la tolerancia de error máxima que se propone para considerar que una reconstrucción sea lo suficientemente buena para plantear la inspección encima de ese método.

Tras describir brevemente las métricas, a continuación se procede a su formalización.

La métrica (M1) viene dada por la ecuación (8.1), donde $GT = \{gt_1..gt_m\}$ es el conjunto de muestras de la secuencia de referencia y $E = \{e_1..e_n\}$ es el conjunto de estimaciones.

$$\frac{|E|}{|GT|} = \frac{n}{m} \tag{8.1}$$

La métrica (M2) viene dada por la ecuación (8.2), donde $u(x)$ es la función escalón (8.3)[(7.13)] y $d(a,b)$ la función de distancia (8.3)[(7.15)], siendo e^* las estimaciones de E registradas, E^*, y $gt^{(n)}$ sus emparejamientos.

$$\frac{\sum_{i=1}^{n} u(x_i)}{n}, x_i = d(e_i^*, gt_i^{(n)}) \tag{8.2}$$

$$u(x) = \begin{cases} 1 & , \quad x \leq u \\ 0 & , \quad x > u \end{cases} \tag{8.3}$$

$$d(a,b) = \|a-b\|_2 \tag{8.4}$$

La métrica (M4) viene dada por la ecuación (8.5).

$$\frac{1}{n}\sum_{i=1}^{n} d(e_i^*, gt_i^{(n)}) \tag{8.5}$$

La métrica (M5) viene dada por la ecuación (8.6), donde st_gt y st_e representan al *timestamp* de las secuencias de referencia y de estimación respectivamente.

$$t = st\_gt_1^{(n)} - st\_e_1 \tag{8.6}$$

La métrica (M6) viene dada por la ecuación (8.7).

$$\begin{aligned} t = \quad & st\_gt_k^{(n)} - st\_e_k \\ & k = \text{ínf}_k\, u(d(e_k^*, gt_k^{(n)}) \\ & k \in 1..n \end{aligned} \tag{8.7}$$

Por último, la métrica (M3)[1] viene dada por la ecuación (8.9) normalizada mediante (8.8) para obtener valores comparables para el mismo dataset.

$$\frac{1}{m}\sum_{i=1}^{n} f(d(e_i^*, gt_i^{(n)})) \quad (8.8)$$

$$f(x) = \begin{cases} 5-(x/u_1)^{\frac{1}{2}} & , \quad x \leq u_1 \quad \in [4,5] \\ 4\cdot(1-(\frac{x-u_1}{u_2-u_1})^2) & , \quad x \leq u_2 \quad \in [0,4] \\ 0 & , \quad x > u_2 \end{cases} \quad (8.9)$$

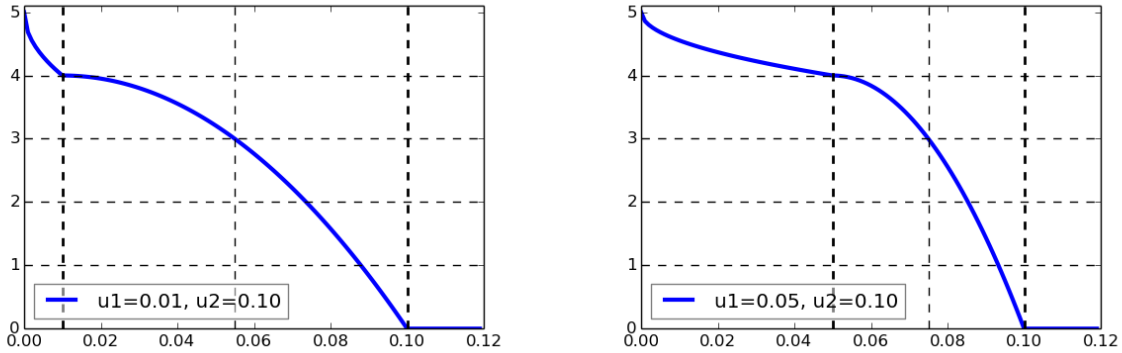


Figura 8.2: Representación de la métrica (M3) para dos criterios de precisión diferentes.

Se trata de una métrica positiva, es decir, su peor valor es el cero, como (M1) y (M2). Es el resultado de invertir y la función de aptitud (7.17) descrita en el capítulo 7.5.2.2. Esta cumple las siguientes propiedades deseables:

- Elitismo en el rango [4,5] para discriminar los métodos con mayor precisión que el requerido.
- Densidad de la puntuación concentrada entorno al umbral de precisión u_1.
- La función vale 3 si la media de puntuaciones está entre los umbrales u_1 y u_2.
  $si \quad x = u1 + \frac{u_2-u_1}{2} \rightarrow f(x) = 3$

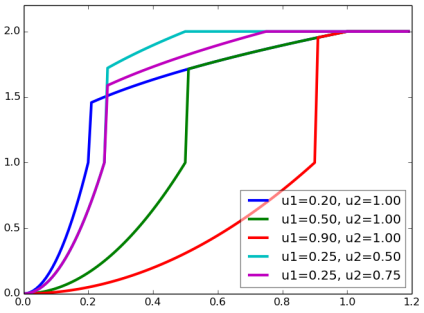


Figura 8.3: Función de aptitud exponencial de doble umbral con penalización.

De este modo hemos conseguido un sistema de puntuación donde u_1 representa la precisión que debe presentar la estimación y u_2 el error de precisión tolerable. Así, un valor de cero implicaría que el algoritmo de VisualSLAM no cumple los requisitos, un valor de cuatro sería el valor óptimo, de modo que cualquier valor superior implica que el algoritmo supera los requisitos de precisión. Por último, debido a que la puntuación se ha normalizado entre cero y cinco donde se entiende que tres es el valor de aprobación en términos enteros, este se ha situado en el punto medio entre ambos umbrales.

[1]Nota para el lector: uno es libre de utilizar esta métrica siempre y cuando cite este trabajo y su autor de forma clara, respetando la licencia y los derechos de autor.

8.3. Presentación de los datos

La presentación de los datos se ha decidido realizar de forma dual, tanto en formato de texto plano como gráfico. Para la presentación gráfica se ha usado la librería *matplotlib*. A continuación se muestran las tres gráficas que se generan:

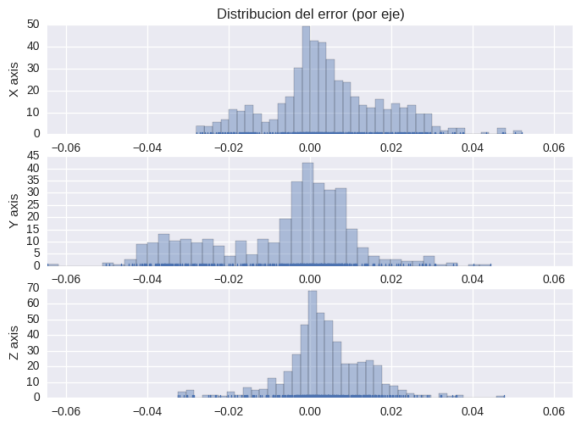


Figura 8.4: Error longitudinal en los tres ejes (freiburg1_desk, ORBSLAM2).

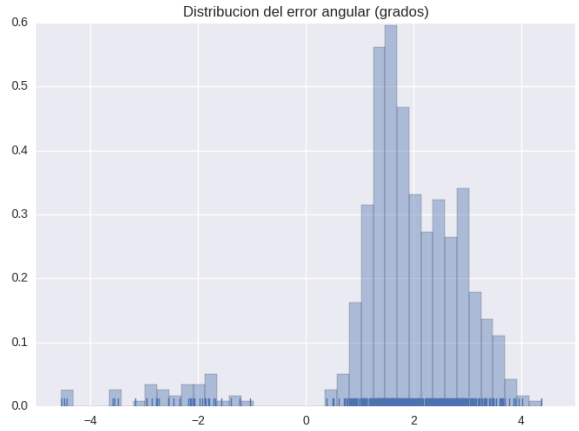


Figura 8.5: Error en orientación (freiburg1_desk, ORBSLAM2).

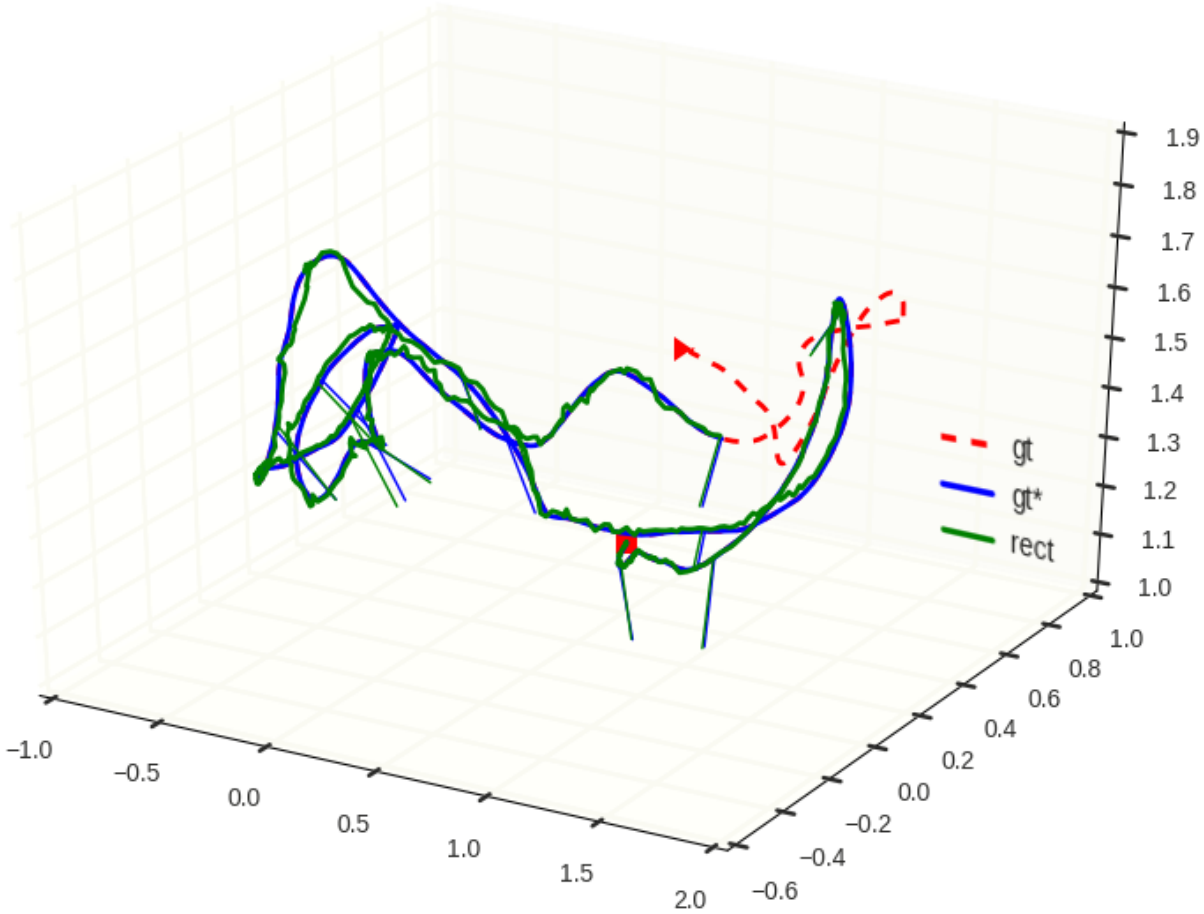


Figura 8.6: Representación 3D (freiburg1_desk, ORBSLAM2).

- La gráfica 8.4 presenta la distribución del error longitudinal en cada uno de los ejes.
- La gráfica 8.5 presenta la distribución del error en orientación.

- La gráfica 8.6 realiza una representación 3D de los dos tracks. Sirve para obtener una primera visión de la estimación donde podemos ver cuán bien se ajusta así como retraso en la inicialización o pérdidas.
  En rojo se representa la secuencia completa, comenzando en el símbolo de *start* '▷' y acabando en el de *stop* '□'. En azul se representa la estimación registrada y en verde el *groundtruth* emparejado. Ambos presentan tanto la información de posición como de orientación.

8.4. Interfaz de línea de comandos

Para esta herramienta se han definido dos puntos de entrada desde la línea de comandos. Estos serán empleados por el cauce de evaluación descrito e el capítulo 3.

```
error_plotting.py [-h] [-R REGISTER_METHOD] [--use_tf USE_TF]
                       [--plot_prefix PLOT_PREFIX]
                       track_1 track_gt

positional arguments:
  track_1               registered track unless other specified.
  track_gt              groundtruth track.

optional arguments:
  -h, --help            show this help message and exit
  -R REGISTER_METHOD, --register_method REGISTER_METHOD
                        use a registration method.
  --use_tf USE_TF       use precomputed transform.
  --plot_prefix PLOT_PREFIX
                        show plots or store them under passed prefix.
```

```
varribas_metrics.py [-h] [--write_to WRITE_TO]
                         track_registered track_gt gt_stamp

positional arguments:
  track_registered      registered track to evaluate.
  track_gt              groundtruth track.
  gt_stamp              groundtruth stamp (use track_gt by default).

optional arguments:
  -h, --help            show this help message and exit
  --write_to WRITE_TO   file where write measurement results.
```

CAPÍTULO

9 Evaluación

En este capítulo realizaremos la evaluación de los algoritmos de VisualSLAM con el entorno y la herramienta presentados en los capítulos anteriores. Estas han sido reducidas a dos para acotar su extensión en este documento. En primer lugar procederemos a su formalización.

El objeto de estudio de este trabajo serán los algoritmos de VisualSLAM: SVO, LSD_SLAM y ORB_SLAM2. Todos ellos parametrizados con sus opciones por defecto salvo SVO que está configurado en su perfil preciso. Estos serán sometidos a varios escenarios del *dataset* de TUM seleccionará a priori.

9.1. Evaluación panorámica

Para esta primera evaluación se seleccionaron a priori seis escenas del *dataset* TUM que en conjunto suponen un test bastante completo y variado. Estas escenas son:

(a) freiburg2_xyz: secuencia de depuración con movimientos longitudinales. Tiene una cinemática más restrictiva que freiburg1_xyz y será el caso base que nos permita detectar problemas o descartar un algoritmo.

(b) freiburg2_rpy: secuencia de depuración con movimientos de rotación. Tiene una cinemática más restrictiva que freiburg1_rpy y actuará como segundo caso base. Se incluye para focalizarnos en el problema del paralaje y de la homografía.

(c) freiburg1_floor: secuencia que enfoca el suelo como un FPV[1]. Es una escena con predominancia planar con textura y estructura.

(d) freiburg3_nostructure_notexture_far: secuencia que carece de estructura y textura. Se espera que sólo los métodos densos puedan trabajar en este escenario.

(e) freiburg3_structure_texture_near: la secuencia más fácil de todas con respecto a complejidad visual. Escenario con varios planos con alta textura y extructura.

(f) freiburg3_teddy: secuencia orientada a la reconstrucción. Vista de 360º a dos distancias.

[1] First-Person-View, referido a a cámara frontal de los drones (jerga en carreras de drones).

A continuación se presentan los resultados de los tres algoritmos sobre los seis casos de estudio. En primer lugar se realizará el enfrentamiento cuantitativo a través de las tablas 9.1 y 9.2. Tras ello se presentarán los resultados gráficos de cada uno de ellos exponiendo sus peculiaridades.

	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(a)	99.95 %	51.51 / 98.17 %	3.850	3.852	0.020833 m	0.064139 s	31.137813 s
(b)	76.93 %	28.53 / 93.32 %	2.595	3.373	0.038668 m	0.069706 s	71.315586 s
(c)	90.74 %	0.00 / 30.79 %	0.717	0.790	0.600450 m	0.063871 s	-
(d)	9.28 %	29.55 / 100.00 %	0.287	3.091	0.039115 m	0.119837 s	0.907968 s
(e)	99.73 %	0.00 / 33.21 %	0.840	0.843	0.582144 m	0.112608 s	-
(f)	65.17 %	12.48 / 85.03 %	1.630	2.501	0.063437 m	0.141302 s	15.645384 s

Tabla 9.1: Métricas obtenidas para LSD_SLAM

	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(a)	98.69 %	64.02 / 99.97 %	4.123	4.178	0.012438 m	0.631945 s	1.631906 s
(e)	93.54 %	27.82 / 100.00 %	3.539	3.783	0.026740 m	2.372932 s	20.837131 s
(f)	95.97 %	36.98 / 98.75 %	3.705	3.860	0.021763 m	0.141302 s	15.208976 s

Tabla 9.2: Métricas obtenidas para ORB_SLAM

	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(e)	9.65 %	40.57 / 83.02 %	0.292	3.030	0.044893 m	3.476925 s	3.644883 s

Tabla 9.3: Métricas obtenidas para SVO

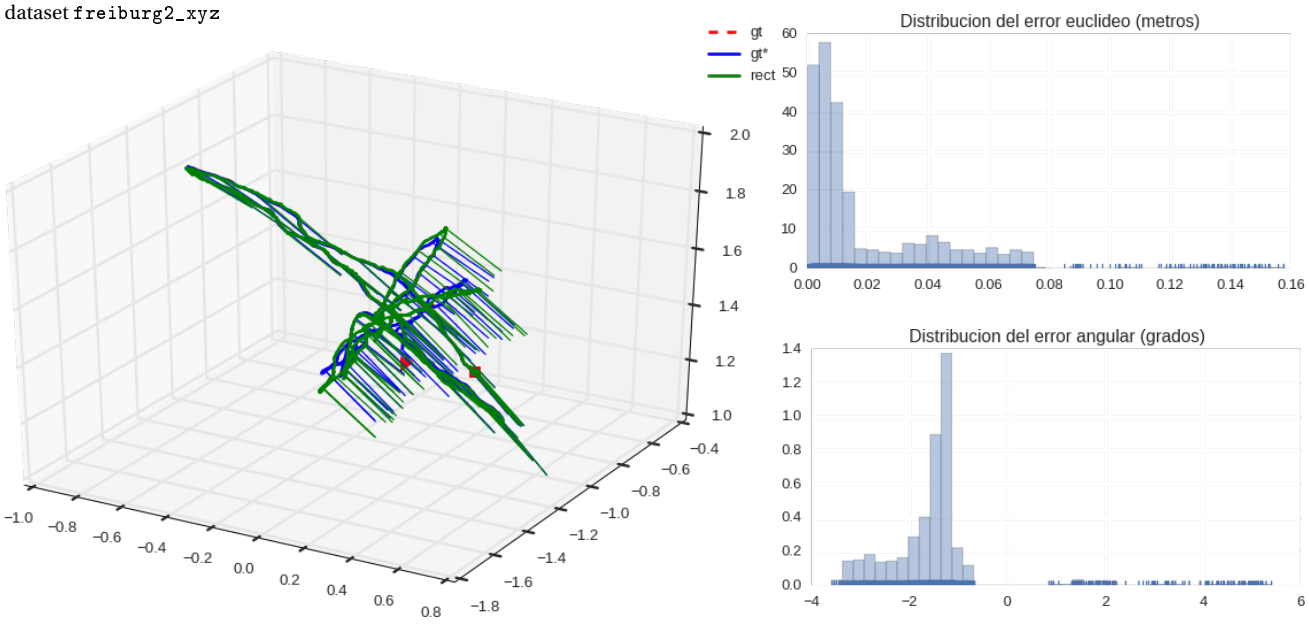
A la luz de los resultados, podemos decir que ORB_SLAM tiene una precisión mayor. La métrica (M3) se encuentra entorno al 4 y su tiempo de convergencia es menor.
Por contra también vemos cómo LSD_SLAM es capaz de inicializarse en escenarios donde ORB_SLAM no puede, presentando una capacidad de inicialización más rápida.

La métrica (M2) se encuentra desdoblada en los dos umbrales de precisión definidos.

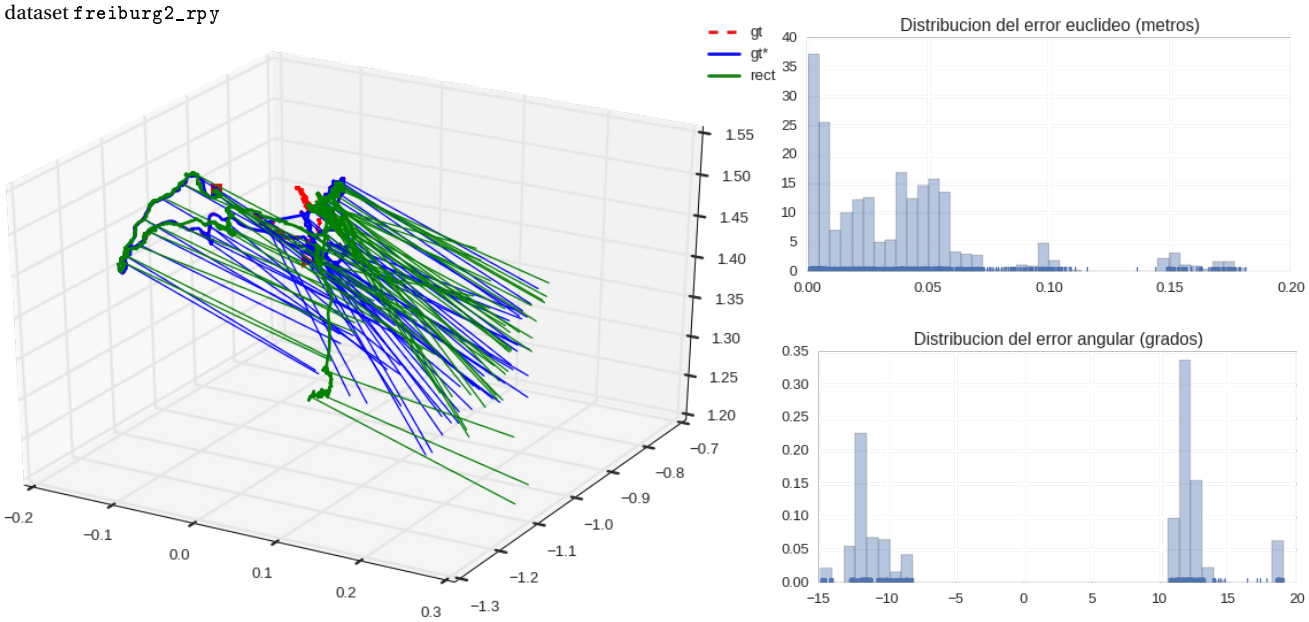
La métrica (M3*) corresponde a la denormalización sobre m, o normalización en (M1), para obtener un valor que dependa de la precisión del propio algoritmo.

9.1.1. LSD_SLAM

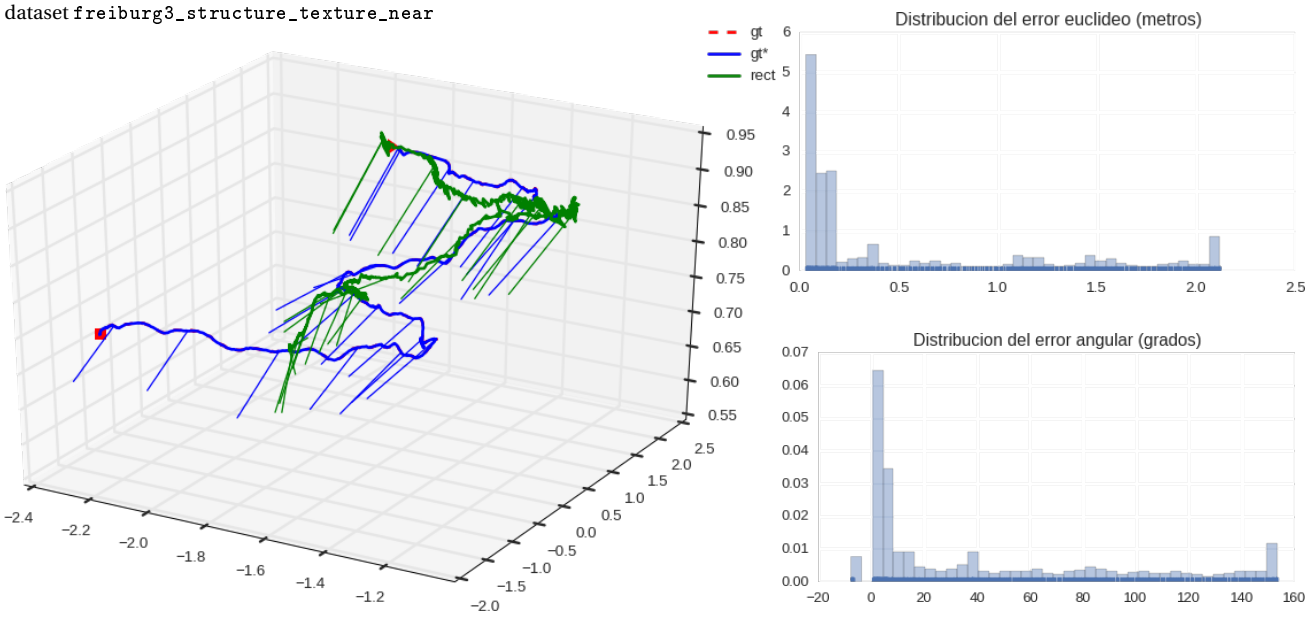
dataset `freiburg2_xyz`



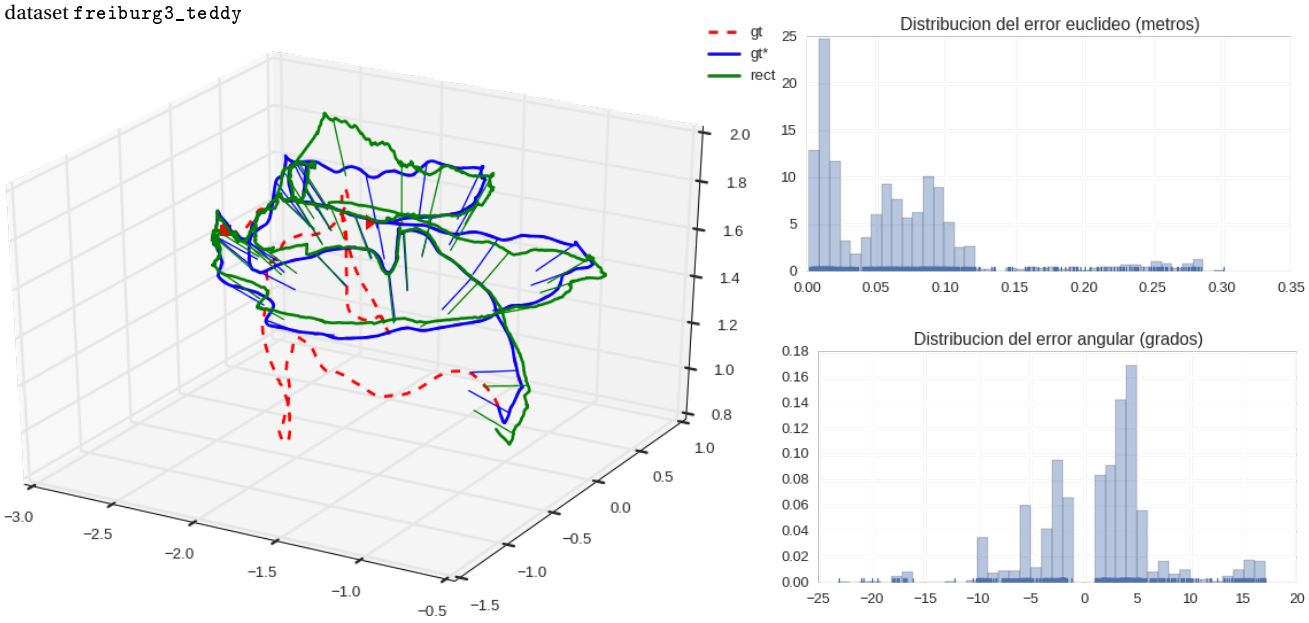
dataset `freiburg2_rpy`



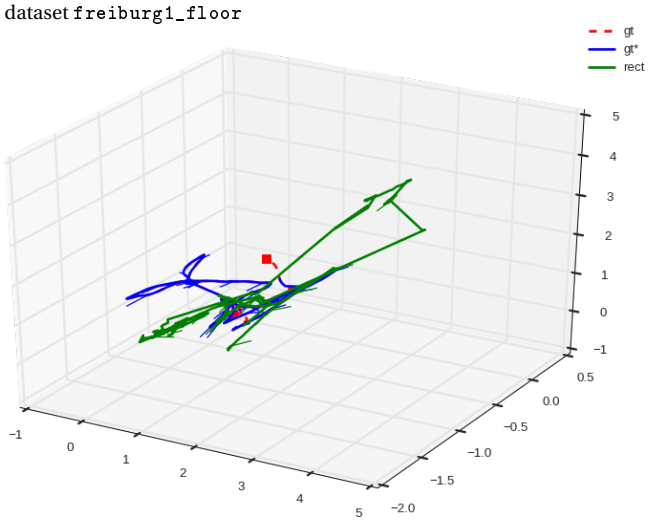
dataset `freiburg3_structure_texture_near`



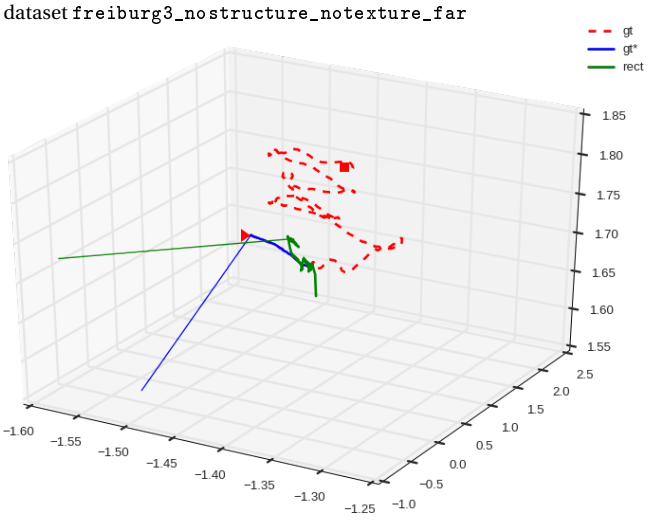
dataset `freiburg3_teddy`



dataset `freiburg1_floor`



dataset `freiburg3_nostructure_notexture_far`

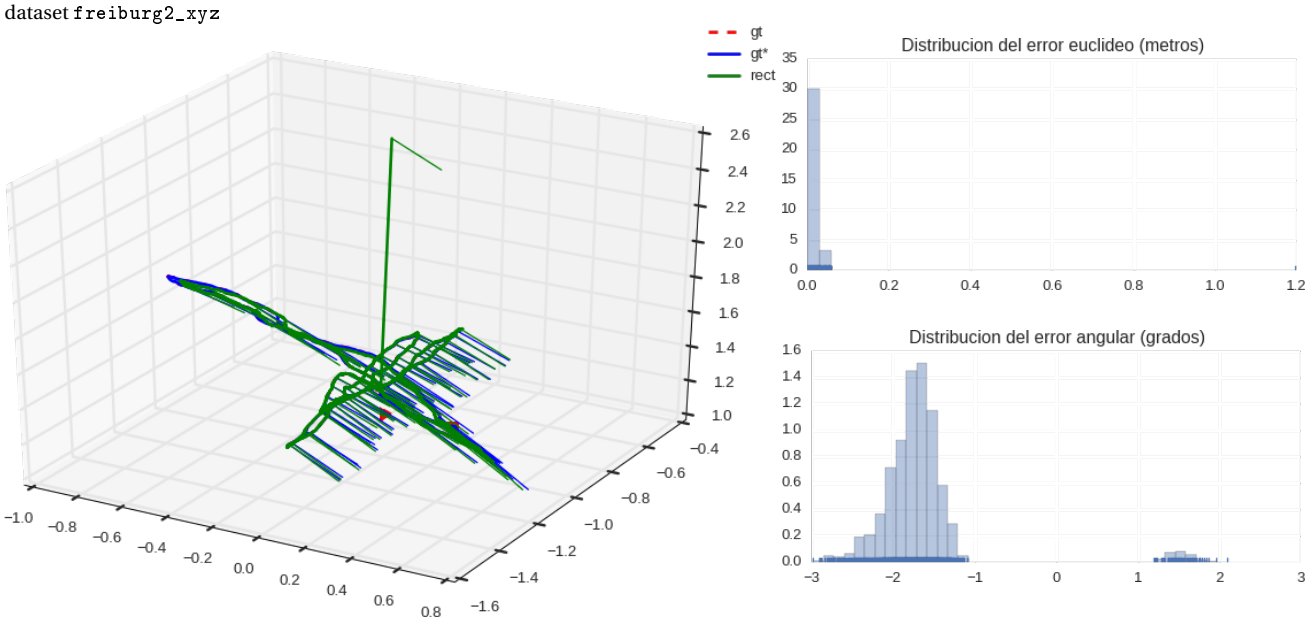


Este algoritmo se comporta muy bien para los casos (a) y (b). En el caso de (e) vemos cómo en los fotogramas más homogéneos la estimación se ha degradado corrompiendo el resto de su estimación. En el caso (f) su estimación se ha interrumpido al 65 % sin capacidad de recuperación.

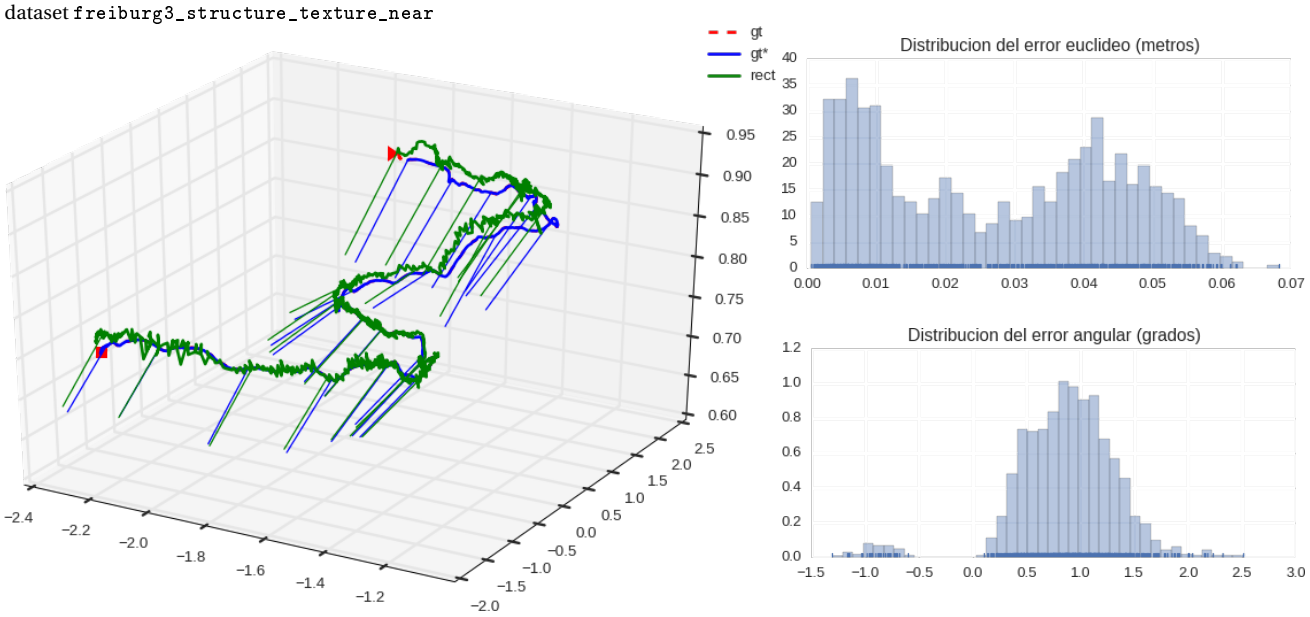
Por último vemos cómo para los escenarios más problemáticos, (c) y (d), aunque es capaz de inicializarse en seguida pierde el *tracking*.

9.1.2. ORB_SLAM2

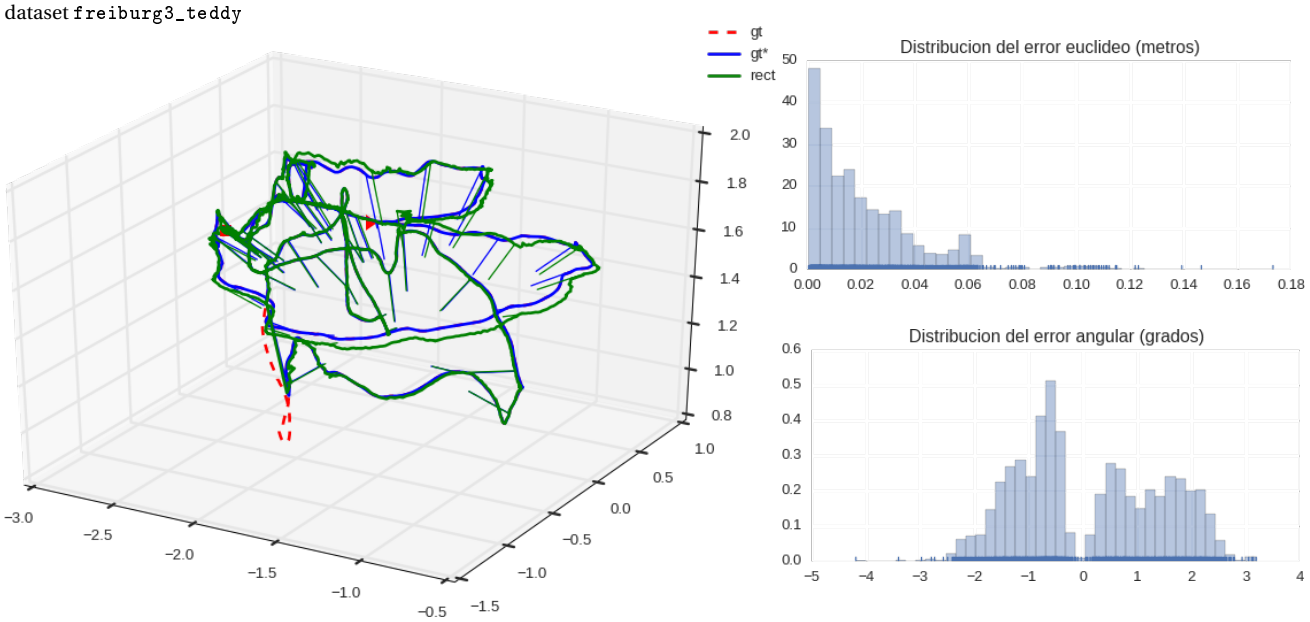
dataset `freiburg2_xyz`



dataset `freiburg3_structure_texture_near`



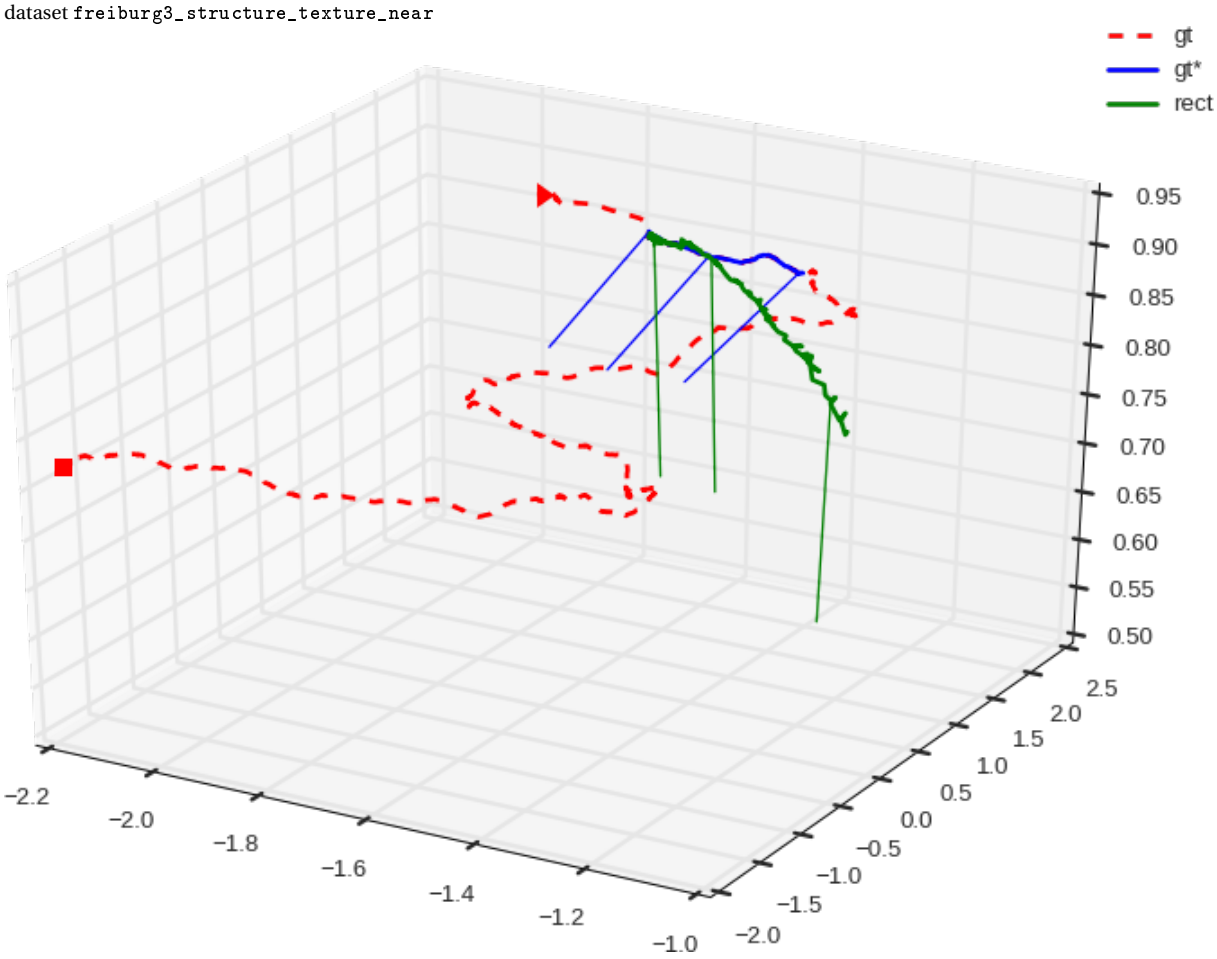
dataset `freiburg3_teddy`



Este algoritmo se ha visto superado por los casos (b), (c) y (d). Sin embargo, para los otros tres casos presenta unos resultados muy precisos con salvedad de algún dato espúreo para (a).
En (e) y (f) vemos cómo la capacidad de ORB_SLAM para relocalizarse e interpretar el mapa le permite recuperarse de las pérdidas. Para (f), vemos que es más robusto ante pérdidas que LSD_SLAM, pero cuando ocurre es capaz de recuperarse si se mantiene el foco de atención. Para (e), vemos cómo en los fotrogramas más homogéneos, el registro de modelo único presenta un pequeño punto de inflexión en dicho punto.

9.1.3. SVO

dataset `freiburg3_structure_texture_near`



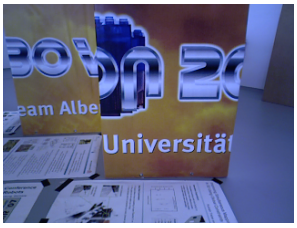
Este algoritmo no ha conseguido inicializarse salvo para (e), donde su estimación no es representativa. Esta tendencia se mantiene para el dataset completo de TUM, por lo que no será tenido en cuenta en la evaluación.

Sin embargo sirve para ilustrar una de las limitaciones del método de estimación de la transformación rígida descrito en 7.5.1. Estas limitaciones están relacionadas con la morfología de la colección de puntos, los cuales si se encuentran degenerados sobre una o dos dimensiones la estimación de la rotación puede no ser correcta al no ser unívoca. En (e) se representa este extremo, donde la orientación presenta un error sistemático de 90º.

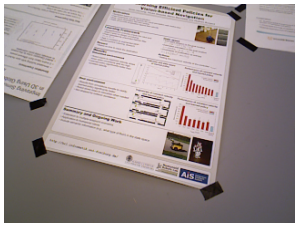
9.2. Evaluación en escenarios favorables

La preselección anterior nos ha permitido ver a grandes rasgos el comportamiento de cada algoritmo. Para completar el estudio basado en la métrica (M3) se ha decidido realizar una evaluación más simple y homogénea. Esta vez, los escenarios escogidos han sido los cuatro que por sus propiedades han sido considerado los más favorables, es decir, aquellos con mucha textura y un espacio de trabajo cerrado.

(a) freiburg3 structure_texture_near



(b) freiburg3 structure_texture_far



(c) freiburg3 nostructure_texture_near_loop

(d) freiburg3 nostructure_texture_far

Se trata de de una selección semi-ciega debido al estudio del caso anterior. En las tablas 9.4, 9.5 y 9.6 se presentan los resultados de cada uno de los algoritmos.

	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(a)	99.73 %	0.00 / 44.25 %	1.312	1.316	0.596036 m	0.112608 s	-
(b)	99.68 %	0.00 / 42.25 %	1.236	1.240	0.726186 m	0.119363 s	-
(c)	99.76 %	5.54 / 66.87 %	2.134	2.139	0.111610 m	0.109426 s	6.575135 s
(d)	81.72 %	0.00 / 51.32 %	1.347	1.648	0.363564 m	0.119177 s	-

Tabla 9.4: Métricas obtenidas para LSD_SLAM

	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(a)	93.54 %	26.65 / 100.00 %	3.636	3.888	0.022974 m	2.372932 s	20.673011 s
(b)	96.80 %	44.05 / 100.00 %	3.945	4.075	0.012056 m	1.007970 s	2.043998 s
(c)	96.85 %	27.32 / 100.00 %	3.830	3.955	0.018787 m	1.776134 s	11.296084 s

Tabla 9.5: Métricas obtenidas para ORB_SLAM

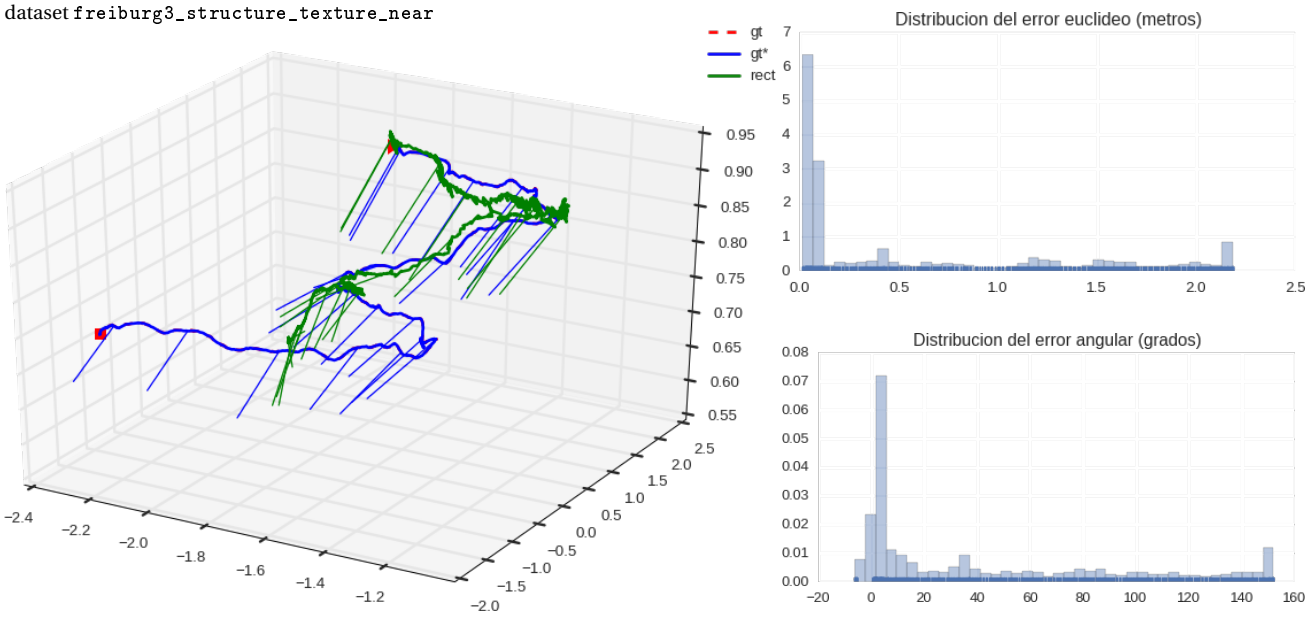
	(M1)	(M2)	**(M3)**	(M3*)	(M4)	(M5)	(M6)
(a)	9.65 %	41.51 / 86.79 %	0.300	3.106	0.042142 m	3.476925 s	3.644883 s
(b)	10.87 %	18.63 / 82.35 %	0.296	2.725	0.053412 m	1.812182 s	2.883893 s
(c)	25.80 %	4.38 / 44.24 %	0.378	1.463	0.151351 m	2.619017 s	6.439067 s
(d)	31.40 %	9.59 / 100.00 %	1.109	3.532	0.035285 m	2.747982 s	3.119825 s

Tabla 9.6: Métricas obtenidas para SVO

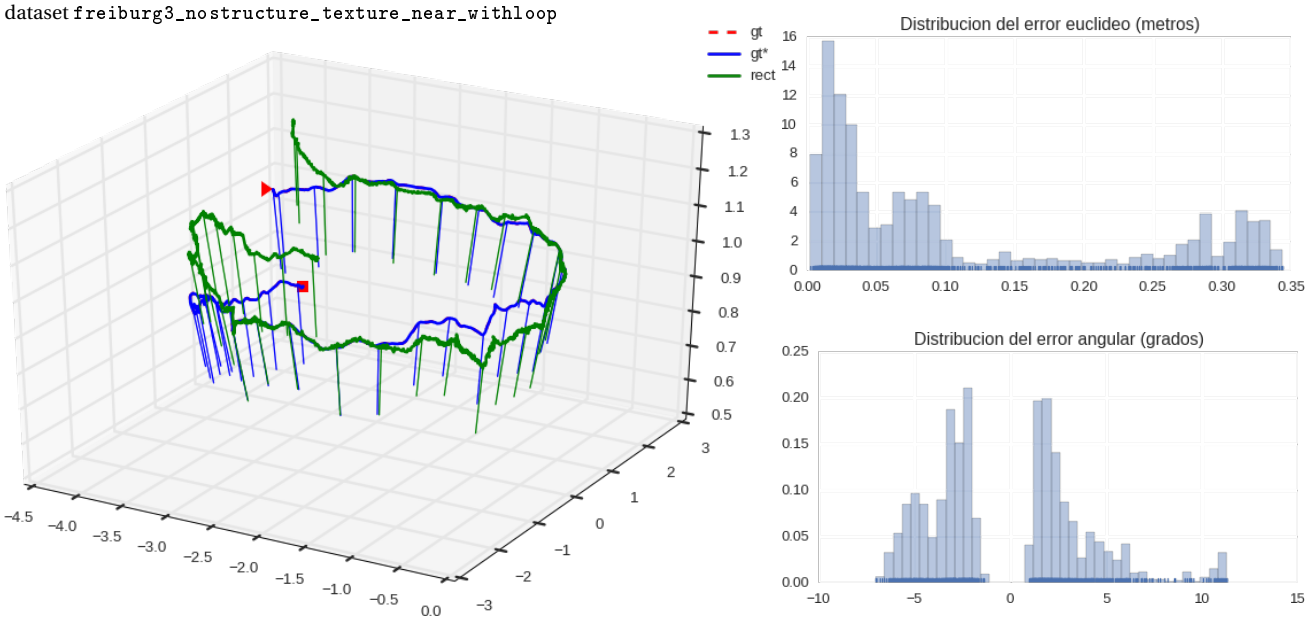
A la luz de los resultados podemos determinar que ORB_SLAM es el algoritmo que mejor rendimiento ofrece en términos de localización y precisión para escenarios con alta textura no homogénea. Como excepción se sitúa el escenario (d), donde una percepción más distante del texto lo convierte en un escenario de textura más homogénea con respecto a (b) y (c). Esto implica que las características ORB sean menos discriminantes.
Así mismo, la completitud de LSD_SLAM se ve eclipsada por su menor rendimiento.

9.2.1. LSD_SLAM

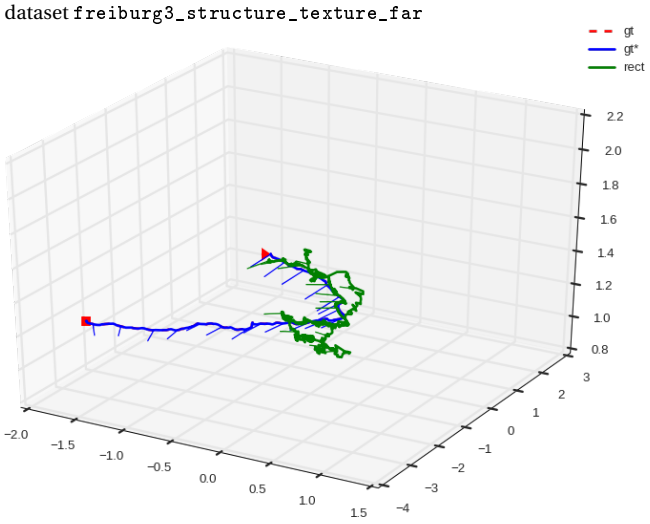
dataset `freiburg3_structure_texture_near`



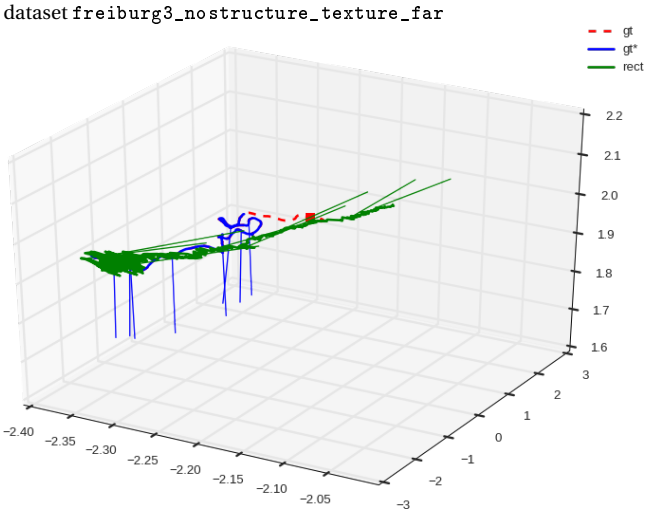
dataset `freiburg3_nostructure_texture_near_withloop`



dataset `freiburg3_structure_texture_far`

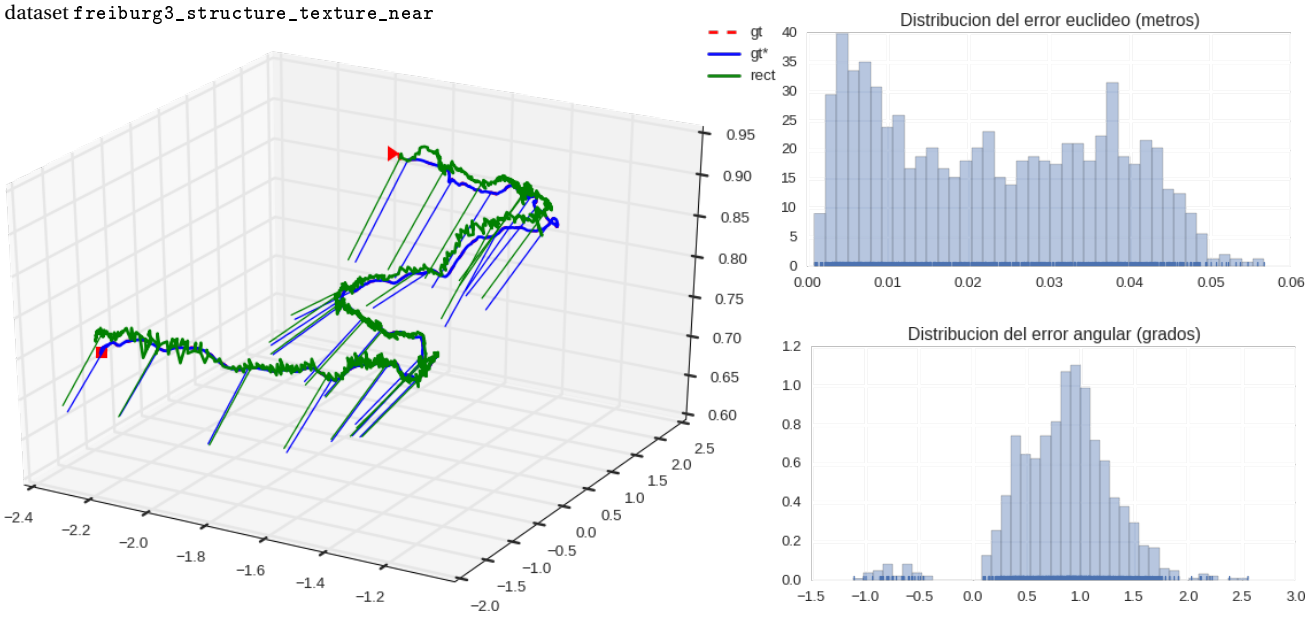


dataset `freiburg3_nostructure_texture_far`

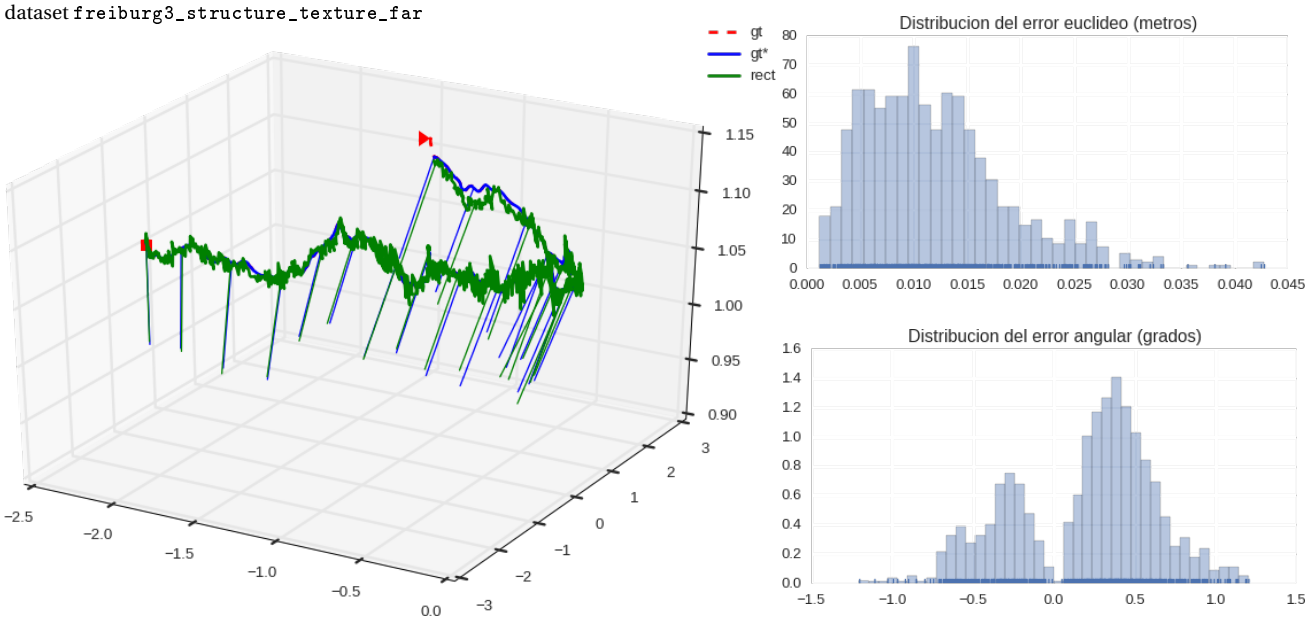


9.2.2. ORB_SLAM2

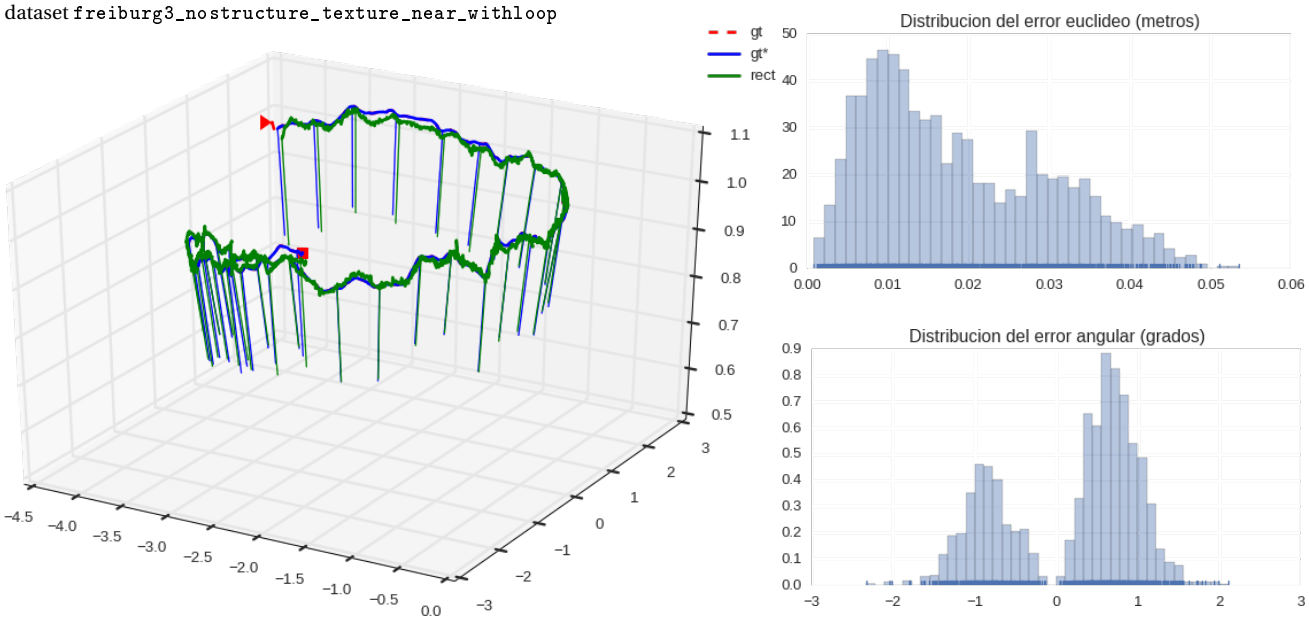
dataset `freiburg3_structure_texture_near`



dataset `freiburg3_structure_texture_far`



dataset `freiburg3_nostructure_texture_near_withloop`



9.3. Conclusiones

Tras analizar los resultados de las dos evaluaciones aquí presentadas de los algoritmos de VisualSLAM podemos realizar las siguientes conclusiones.

En primer lugar decir que ORB_SLAM2 es el algoritmo que mejor rendimiento presenta tal y como indica la métrica (M3). Acudiendo a la tabla 9.5 podemos ver que aunque su tiempo de inicialización es mayor [métricas (M1) y (M5)], la estimación de la posición es mucho más precisa [métricas (M2) y (M3*)]. Esta situación no es extraña, ya que como discuten varios autores, la calidad de la localización de los algoritmos basados en características suele ser superior a los de los métodos directos por la propia naturaleza del problema.

Asimismo, los resultados más pobres de los dos métodos directos están condicionados en parte por el propio *dataset* empleado en su evaluación. El sensor Kinect es de tipo *rolling shutter*, es decir, la imagen es sensible a deformaciones y cizallados debidos al movimiento. Esta característica penaliza a los métodos directos, mientras que los basados en características pueden trabajar con este tipo de sensores si los parches que definen las características son lo suficientemente pequeños.

Dentro de este caso se encuentra SVO, el cual, según sus propios autores, se limita a cámaras *global shutter* enfocadas directamente hacia el suelo[2]. El *dataset* TUM sólo presenta escenas cuya dinámica de movimiento implica una cámara frontal. Esto se extiende a ICL-NUIM así como a los otros *datasets* enfocados a la conducción autónoma. Por tanto, ninguno de los *datasets* contemplados se enmarcan dentro del dominio de actuación de SVO. Sin embargo, ICL-NUIM cumple con el primer requisito ya que sus imágenes son sintéticas y no presentan este tipo de artefactos.

El uso del *dataset* ICL-NUIM, así como el de RPG se planteó para ofrecer una visión más completa. Sin embargo esta idea fue finalmente descartada, así como la inclusión del resto de los escenarios de TUM, en favor de la brevedad. El objetivo de este proyecto no es realizar una evaluación exhaustiva del comportamiento de cada algoritmo ante los diferentes escenarios que se les pueden plantear, sino el de presentar una herramienta y una métrica que sirvan como apoyo en la elección de un algoritmo de VisualSLAM. Por ello esta labor se considera un trabajo futuro.

Por último, destacar que las conclusiones y resultados están más ligadas a las implementaciones de los algoritmos que a los propios algoritmos *per se*. Siendo un factor determinante el correcto ajuste y parametrización de cada algoritmo.

[2] véase `http://jderobot.org/Varribas-tfm/SLAM_algorithms_review#Conclusions`

CAPÍTULO

10 Conclusiones

Con este capítulo se da cierre al presente documento y al desarrollo del Trabajo de Fin de Máster. En él se presentarán las conclusiones, repasando el cumplimiento de los objetivos marcados, resumiendo el código realizado y los conocimientos adquiridos, y presentando los posibles trabajos futuros derivados de la consecución del mismo.

10.1. Conclusiones

En primer lugar se revisa el cumplimiento de los objetivos y de los requisitos definidos en el capítulo 2.

- El primer objetivo consistía en el estudio y adaptación de diversos algoritmos de VisualSLAM.

  - Con respecto al estudio teórico de las diferentes técnicas y algoritmos de VisualSLAM, cumpliendo con los requisitos R1 y R2 se han estudiado cuatro algoritmos diferentes. Dos de ellos (DTAM y LSD-SLAM) emplean métodos directos, otro (ORB-LAM) basado en características, siendo el restante (SVO) un híbrido que intercala el error fotométrico con el de reproyección. Así mismo, tenemos dos métodos enfocados en la reconstrucción densa del mapa (DTAM y LSD-SLAM) y otro enfocado a la odometría visual (SVO). Un algoritmo que requiere GPU (DTAM) y otro que puede correr en el hardware embebido de un minidrone (SVO). Por mencionar una última distinción, dos de ellos (LSD-SLAM y ORB-SLAM) contemplan el cierre de bucle. Uno de ellos (LSD-SLAM) emplea métodos estadísticos (la matriz de covarianzas) para caracterizar la fortaleza de la relación entre *keyframes*, mientras que el otro (ORB-SLAM) emplea conteo de características. Así mismo, ORB-SLAM integra capacidad de reconocimiento semántico dentro del mapa para poder relocalizarse y detectar cierres de bucle con mayor deriva.

  - Para realizar la evaluación de los mismos, se ha partido de implementaciones existentes de LSD-SLAM, SVO y ORB-SLAM2 acotando el esfuerzo necesario para la consecución de este trabajo. Sobre estas implementaciones ha sido necesario realizar algunas adaptaciones y corrección de errores. Para evitar problemas de compatibilidad cada implementación ha sido encerrada en un contexto propio.

También se han definido los mecanismos de interacción y comunicación para integrarlos dentro del cauce de evaluación.

- El segundo objetivo consistía en el estudio y adaptación de los datasets destinados al problema del visualSLAM. Para ello se buscaron diversas bases de datos, seleccionándolas por su relevancia. Tras estudiar cada una de ellas y ver las capacidades de algunas de las mismas para realizar estudios y análisis enfocados en un criterio concreto, se escogió el dataset TUM. Esto es debido a la adecuación de los escenarios de prueba que provee y al presentar de forma nativa un formato ROS similar al escogido para el cauce de evaluación.

- El tercer objetivo se encuentra dividido en dos subobjetivos abarcando la mayor parte del desarrollo realizado en este trabajo.

  - El primer subobjetivo consistía en la realización de una herramienta que permitiera el análisis de cada algoritmo. Este objetivo se ha cumplido realizando no solo una herramienta puntual para tal objetivo, sino una librería que ofrece una mayor capacidad y que permitirá a cualquier futuro desarrollador abstraerse de los problemas relativos al ajuste temporal y espacial.
    Se han adoptado diversos criterios para realizar el análisis, otorgando a la librería un alto potencial y flexibilidad. A este respecto, la única limitación de la misma se encuentra en el método de ajuste espacial, para el cual se ha descartado la información relativa a la orientación y se ve afectado en los casos peculiares en los que el conjunto de datos se encuentran degenerados en alguna de las dimensiones del espacio tridimensional.

    La contribución principal de esta herramienta con respecto a la herramienta de evaluación que ofrece TUM es la inclusión del ajuste temporal. En la herramienta que aquí se presenta se realiza la interpolación de las posiciones de verdad absoluta para medir y caracterizar el error en el mismo instante de tiempo. En TUM se mide este error únicamente para las estimaciones próximas a la información de verdad absoluta, pero sin modelar ni contemplar el desfase temporal existente.

    Destacar que la librería, y por extensión la herramienta, cumple el requisito R3. Donde la capacidad de extenderse a nuevos algoritmos no sólo cubre los de VisualSLAM, sino cualquier algoritmo de SLAM y SfM que entregue la información de localización en un formato compatible.

    Con respecto al requisito R6, no se cuenta con información objetiva al respecto ya que, lamentablemente, no ha habido posibilidad de recibir realimentación de otros usuarios. Sin embargo, en mi humilde opinión, creo que es un requisito completamente satisfecho y que se puede ver en la interfaz de línea de comandos y en el uso de la herramienta dentro del cauce de evaluación.

  - El segundo subobjetivo consistía en el diseño de las métricas para la evaluación y cuantificación de la calidad de los algoritmos de VisualSLAM. Estas métricas se encuentran formalizadas en la sección 8.2 del capítulo 8, donde se ha definido la métrica principal (M3) cumpliendo con el requisito R7. Esta métrica propia se presenta como un sistema de calificación para la calidad de *localización* de los algoritmos de SLAM orientado a su aplicación en escenarios reales, donde la importancia viene dada por su precisión teniendo en cuenta su comportamiento global. Se trata de una métrica comparable solamente condicionada por el sesgo semántico entre escenarios.

A parte de las seis métricas establecidas, la evaluación incluye el estudio estadístico del error de traslación y de rotación, así como las métricas entregadas por la herramienta de evaluación de TUM, satisfaciendo el requisito R8 en su totalidad. Esta información se ofrece cumpliendo los requisitos R9 y R10.

- El cuarto objetivo consistía en la definición y construcción de un entorno completo que simplificase el proceso de evaluación de los algoritmos de visualSLAM. Este entorno ha sido formalizado diseñando un cauce de evaluación, incluyendo su arquitectura, etapas e interfaces de comunicación.
  Tras evaluar el entorno definido, se puede afirmar que cumple los requisitos impuestos R3, R4 y R5. Sin embargo, con respecto al requisito R6, éste solo se cumple en el "lado cliente", es decir, en la evaluación de los algoritmos de VisualSLAM dejando al margen cómo se obtienen sus estimaciones.

  Además, se ha realizado un diseño del cauce de evaluación, especialmente en sus detalles de implementación a bajo nivel, que ha permitido la re-evaluación completa de un algoritmo cuando se han presentado diferentes inconvenientes: pérdida de datos, *datasets* corruptos, errores propios y ajenos, etc, contribuyendo a una mejor calidad del software, de las comparativas y de las conclusiones de este trabajo. Cumpliendo con los requisitos R11, R14, R15 y R16.

- El quinto objetivo consistía en analizar las fortalezas y debilidades de los cuatro algoritmos estudiados para su posterior uso. Atendiendo a la evaluación y conclusiones realizadas, el algoritmo de VisualSLAM que mejor rendimiento presenta es ORB-SLAM, siendo el único que cumple con el criterio de precisión establecido en 1cm. Su virtud es su capacidad para relocalizarse e incluso interpretar mapas precalculados o capturados con otro dispositivo diferente, aunque viene acompañado de un coste computacional mayor. Dejando a un lado las restricciones hardware, sería el candidato ideal de los cuatro contemplados para navegación con *drones*, enjambres y flotas heterogéneas.

En segundo lugar, y al margen de los objetivos, se ha visto la importancia de los datos sintéticos, así como las capacidades de cada uno de los *datasets* públicos estudiados para realizar estudios y análisis enfocados en un criterio concreto. Se ha realizado una formulación de tres puntos de RANSAC (3-RANSAC) que ha simplificado la elaboración tanto teórica como práctica de la parte correspondiente al ajuste espacial. Asimismo, todo el desarrollo e investigación realizados han permitido obtener una visión global y clara del problema.

Por último, la necesidad de realizar este entorno de evaluación ha quedado recientemente validada. A menos de un mes de la defensa de este trabajo, desde la Universidad de Toronto se ha presentado una herramienta de ayuda, *pykitti*, que facilita el uso de la base de datos KITTI. Equiparándola al trabajo que aquí se presenta, se limitaría a los módulos específicos *parse* y *tf_tools*.

10.1.1. Desarrollo software

El desarrollo software implicado en este proyecto es bastante heterogéneo y ha implicado el desarrollo e interacción de diversos componentes. A continuación se describen los cinco componentes principales:

- **dataset_evaluation_framework**
  Se trata del entorno de evaluación de alto nivel reflejado en la sección 3.3 del capítulo 3. Se encarga de realizar las tareas de configuración y simplificación, como la ejecución por lotes. Es un *framework* de composición, es decir, se compone de diversas herramientas y éste hace uso de ellas.
- **dataset_crawler**
  Es un conjunto de herramientas encargadas de la descarga y extracción inteligente del contenido de los diferentes *datasets*.
- **visualslam_evaluation**
  Es la herramienta de evaluación de alto nivel. Incluye las métricas definidas en 8.2.
- **posetools**
  Es la herramienta de análisis y evaluación de bajo nivel. Se encuentra formalizada como una librería de análisis, exponiendo su funcionalidad tanto a través de una API de desarrollo como a través de una interfaz de línea de comandos. Gracias a esta segunda, se puede emplear como un programa más, siendo su *CLI* explotada por el entorno de evaluación.
- **runners**
  Engloba el código *Shell* asociado a la etapa de ejecución del cauce de evaluación. Aglomera los envoltorios de bajo y alto nivel de cada uno de los algoritmos de visualSLAM evaluados.

Asimismo, se han desarrollado herramientas y programas de menor relevancia, de las cuales se destacan las dos herramientas secundarias más importantes:

- **posetools_evaluation**
  Se trata de una pequeña herramienta empleada para seleccionar el mejor método de registro para la librería de análisis. Esta herramienta define un entorno y una metodología de evaluación extensible a cualquier método futuro.
- **docker_toolkit**
  Se trata de un conjunto de herramientas desarrolladas para simplificar y completar el uso de Docker.

En la tabla 10.1 se hace una cuantificación del código desarrollado, que supera las 8.000 líneas de código efectivo[1] , de las cuales 4.359 corresponden a la librería de análisis. El código está sujeto, casi en su totalidad, a la licencia GPL versión 3 y puede encontrarse en: `https://gitlab.com/groups/varribas-pfm`.

Componente	Lenguaje	Líneas de código
datasets_evaluation_framework	Shell	232
datasets_crawler	Shell	105
visualslam_evaluation	Python	634
posetools	Python	4359[2]
posetools_evaluation	Python	1087

Componente	Lenguaje	Líneas de código
runners	Shell	338
pose2file	Python	87
eval_timestamp	Shell	120
docker_toolkit	Shell	388

Tabla 10.1: Estadísticas del código empleado en este trabajo.

[1] líneas de código real, excluyendo comentarios y líneas en blanco
[2] 5.750 líneas incluyendo comentarios y líneas en blanco

10.1.2. Conocimientos adquiridos

Los conocimientos adquiridos, o que se han necesitado adquirir, para la realización de este proyecto han sido muy numerosos y diversos. En primer lugar, y atendiendo a los conocimientos teóricos, se ha entendido a grandes rasgos la metodología y la matemática subyacente de los algoritmos de VisualSLAM, comprendiendo sus puntos débiles. Se han adquirido conocimientos elementales sobre el álgebra de Lie. Se ha adquirido la capacidad de trabajar con cuaterniones en sus aplicaciones más simples, como la codificación de rotaciones en el espacio 3D e interpolación de las mismas.

Con respecto a los conocimientos prácticos o más ingenieriles, se ha profundizado en ciertas facetas de ROS, se ha aprendido Docker a un nivel avanzado creando un pequeño *toolkit*, se ha creado una metodología *CoW* basada en *schroot* y se ha mejorado en Python[3], usando por ejemplo decoradores y funciones lambda.

En un contexto menos cercano, se pueden destacar otros conocimientos adquiridos dentro del marco del Trabajo de Fin de Máster. Incluyendo el aprendizaje de JdeRobot, la plataforma de desarrollo del Laboratorio de Robótica de la URJC, o el aprendizaje de ZeroC Ice y CMake, de los cuales no se tenía ningún conocimiento previo. Así como el uso de Gazebo, tanto a nivel de usuario como de desarrollador.

10.2. Trabajo futuro

Tras la finalización de este trabajo, los posibles trabajos futuros se abren en varios frentes. En este apartado se recogen algunos de ellos.

La línea más natural de continuación de este trabajo es extenderlo incorporando más bases de datos y más algoritmos de visualSLAM. Esta podría extenderse a otros algoritmos que por ejemplo empleen sensores de profundidad como el planteado dentro del *Proyecto Tango*.

Con respecto al trabajo realizado, hemos visto que el método de registro tiene algunas limitaciones. Una mejora sería la el uso de otro método en busca de sobrepasar estas limitaciones.

Asimismo, en algunos capítulos se ha hecho referencia al estudio de la frecuencia de trabajo de los algoritmos de VisualSLAM. Éste ha sido omitido debido a diversos problemas y comportamientos erráticos que implican que este estudio no sea representativo. Un posible trabajo futuro sería la realización de este estudio, incluyendo el *profiling* del algoritmo completo y sus múltiples hilos en lugar de limitarse al del hilo de *tracking*.

Con esta primera herramienta hemos abordado exclusivamente la medición de la calidad de la localización. Como trabajo futuro sería interesante evaluar la calidad de la reconstrucción generada por los algoritmos de VisualSLAM estudiados.

[3]lenguaje aprendido de forma autodidacta en el Máster de Visión Artificial.

APÉNDICE

A Algoritmos de registro: estudio de las las alternativas

En el capítulo 7.5.1 vimos un método de estimación de la transformación rígida. Así mismo, en 7.5.2 vimos dos alternativas para lidiar con los datos atípicos. Finalmente en 7.5.2.2 vimos que se podía definir un gran número de variantes para solventar el problema del registro y por tanto del ajuste espacial dentro de nuestra herramienta de análisis de algoritmos de visualSLAM.

En este capítulo se estudiarán estas variantes, de modo que podamos determinar cuál es la configuración que mejor responde a nuestro ámbito.

La realización de este apartado se hará de forma externa a la librería explicada en el capítulo 7. Esto nos permitirá identificar la API y aumentar la complejidad de la evaluación. De este modo, el módulo que aquí se describe podrá ser autocontenido, incluyendo los ficheros auxiliares y los escenarios no aleatorios necesarios para la evaluación, sin afectar a la librería en sí misma[1]. Por todo ello, estaríamos hablando del primer programa o módulo desarrollado sobre la librería.

Para realizar esta tarea, primero es necesario definir cuál será la metodología para realizar la evaluación de los algoritmos de registro. Esta es definida y queda acotada en los siguientes apartados.

En esencia probaremos los distintos algoritmos de registro con secuencias sintéticas a las que aplicaremos una transformación en $SIM(3)$ (compuesta por una rotación, una traslación y una escala) y a las que añadiremos ruido. De todas ellas tendremos, por lo tanto, la verdad absoluta (secuencia original) y la secuencia transformada. Los algoritmos de registro estimarán esa transformación en $SIM(3)$ lidiando mejor o peor con las diferentes magnitudes de ruido existentes.

A.1. Arquitectura de evaluación

El objetivo es evaluar el comportamiento robusto o no de los diferentes algoritmos registradores ante el ruido, tanto de espúreos como sistemáticos, que se presentan en la secuencia de poses reales y estimadas. La ilustración A.1 presenta la arquitectura de evaluación definida. Esta se encuentra dividida en tres partes:

[1] esta restricción está impuesta por el mecanismo de control de versiones de Git y su efecto memoria.

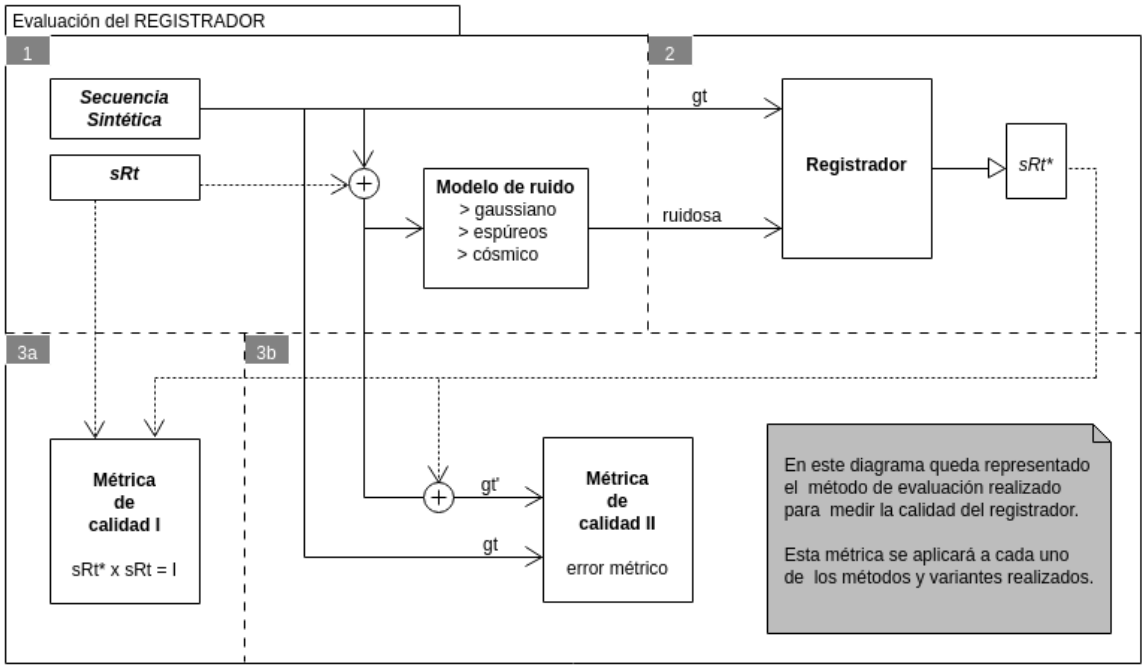


Figura A.1: Evaluación del registrador

(1) La primera etapa se encarga de la generación de los datos. Implica la generación de una secuencia sintética y la transformación a inferir, además de un modelo de ruido que le ponga las cosas complicadas al registrador y nos permita trasladar la secuencia a un entorno más realista.

(2) La segunda etapa es la estimación de la transformación. Esta debe ser tratada como una caja negra.

(3) La tercera etapa es la métrica de calidad, la cual será responsable de elegir cuantitativamente qué registrador es mejor. En ella se muestran dos métricas, la primera que mide cuan cerca nos hemos quedado de la solución en el espacio de transformación (3a) y la segunda que lo mide sobre la secuencia (3b).

A.2. Secuencias sintéticas de poses y magnitud del ruido introducido

Como primer paso de la etapa de evaluación es necesario definir y cuantificar el ruido que se va a introducir en las muestras. Esto choca con respecto a las condiciones mínimas necesarias para la verificación de la corrección descrita en el apartado 7.7.1.2 ya que en ese caso estamos empleando los datos sintéticos como datos de entrenamiento para elegir el registrador que mejor se adapte a nuestro entorno. Esto implica definir las unidades de las muestras así como las del ruido.
Por ese motivo, para esta evaluación se incluyen los valores exactos escogidos con objetivo de que queden documentados.

Para la elaboración del *track* se ha decidido crear una secuencia de 100 puntos cuya distancia entre puntos varía según una distribución uniforme entre 1cm y 50cm; otorgando una distancia media de 25m. La inclusión de espúreos se rige por una distribución uniforme entre 1m y 10m. El ruido blanco introducido está escalado a nivel de decímetros y variará su desviación típica entre 0 y 2. Los valores concretos quedan reflejados en las siguientes tablas:

std	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.2	1.4	1.6	1.8	2.0
media	0.016	0.031	0.047	0.063	0.080	0.095	0.111	0.128	0.143	0.159	0.192	0.222	0.256	0.288	0.319
std	0.006	0.013	0.020	0.028	0.33	0.039	0.046	0.054	0.063	0.067	0.081	0.93	0.107	0.121	0.133
max	0.046	0.102	0.144	0.222	0.246	0.295	0.316	0.388	0.418	0.447	0.573	0.644	0.711	0.916	0.968

Tabla A.1: Magnitud del ruido gaussiano. Cabecera: desviación estándar por eje (en decímetros) del ruido. Cuerpo: estudio estadístico medido en distancia euclídea (en metros).

Track	
número de puntos	100
longitud media	25.170 m

Tabla A.2: Estudio de la longitud del track para 1000 casos

Datos espúreos	
media	5.542 m
min	1.000 m
max	9.999 m

Tabla A.3: Magnitud datos espúreos para 1000 casos

A.3. Medidas de calidad del algoritmo registrador

Para efectuar la evaluación de los algoritmos registradores se han definido dos métricas de error:

a) **Error residual de transformación**: el objetivo de los algoritmos registradores es el de encontrar la transformación inversa a la aplicada a la secuencia sintética. De este modo, se puede evaluar la calidad de la misma atendiendo al residuo de su producto, es decir, cuán lejos se está de la identidad.

$$error = \left\| I^* - I \right\|, \quad I^* = sRT^* \cdot sRt$$

b) **Error métrico**: al estar emparejando puntos tridimensionales y buscar la transformación que minimiza el error euclídeo, o error métrico, podemos emplear esta misma métrica para la evaluación.

$$error = \frac{1}{N} \cdot \sum_{i=1}^{N} \left\| p_i^* - p_i \right\|$$

La realización de la evaluación se ha hecho efectiva a través de dos aproximaciones que plantean dos metodologías de evaluación levemente diferentes:

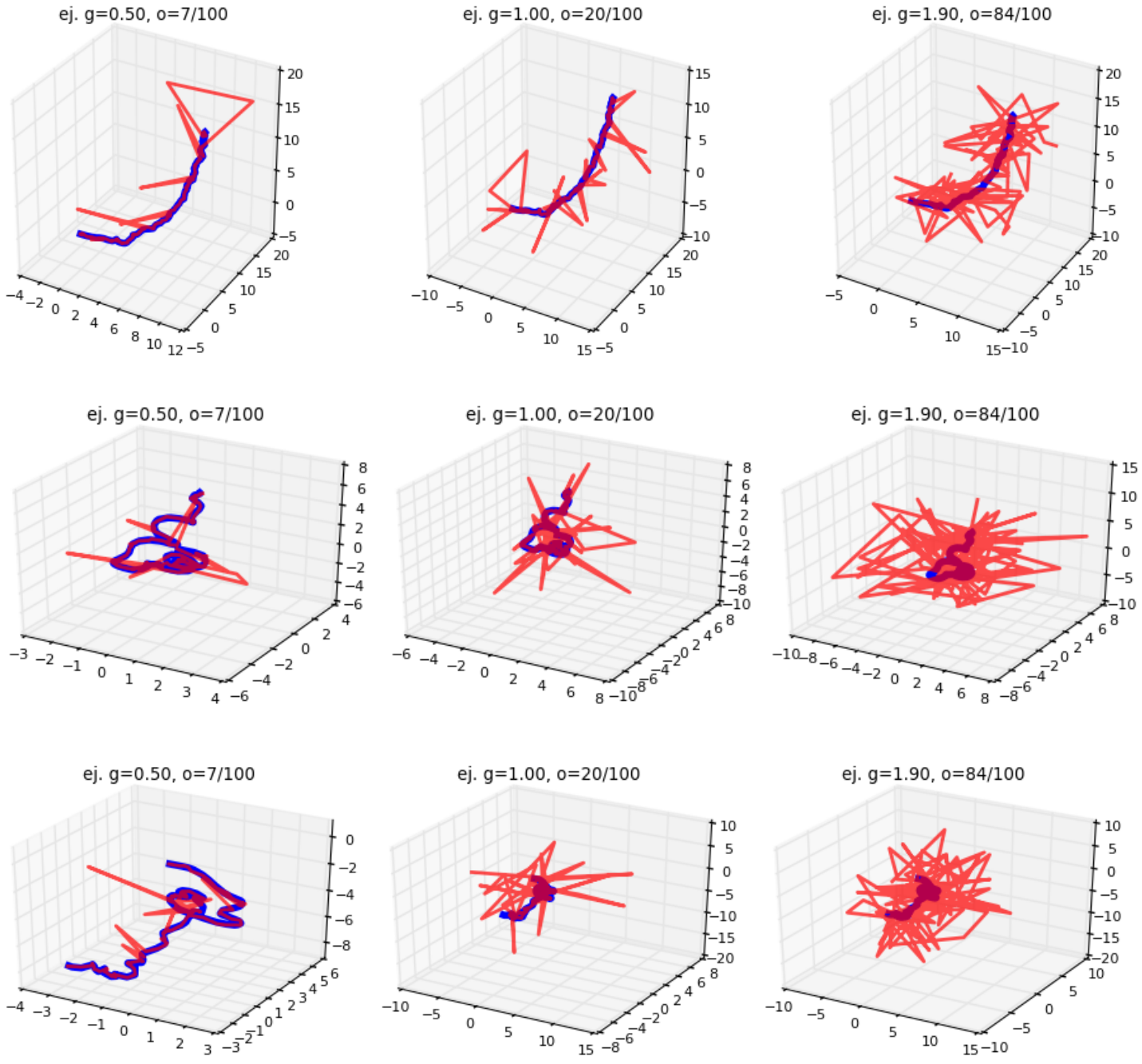
A. Planteamiento gráfico: se trata de un estudio no exhaustivo que plantea una evaluación cualitativa visual. Es no exhaustivo porque el experimento queda definido por un único track y una transformación.
Ambos pueden ser aleatorios o definidos por el usuario, siendo el segundo caso el relevante para este planteamiento.
El estudio se hace en los dos modos de ruido: ruido blanco y espúreos.

   - desviaciones del ruido blanco empleadas: $x \in [0,2], \Delta x = 0{,}2$
   - porcentaje de espúreos empleados: $y = 0,2,4,6,8,10,15,20,25,30,40,50,65,80,95$

B. Planteamiento numérico: se trata de un estudio exhaustivo que somete los algoritmos a evaluar a N secuencias y M transformaciones, resultando en una complejidad final de $N \cdot M$. Los valores empleados han sido 100 y 20 respectivamente.
Este método emplea las dos métricas definidas con anterioridad, las cuales son evaluadas para cada par $< x, y >$ de la superficie de error.

Esta dualidad definida también tiene una diferenciación en las restricciones que debe cumplir el método. Para B, cuyos resultados deben ser numéricamente estables y representativos, todos los métodos deben devolver siempre una hipótesis por muy mala que sea. Para ello se empleará (Alg-1)[2] como caso peor. De forma antagónica, A se ha formalizado para que permita la negación de hipótesis. De este modo podemos ver el dominio de trabajo del método.

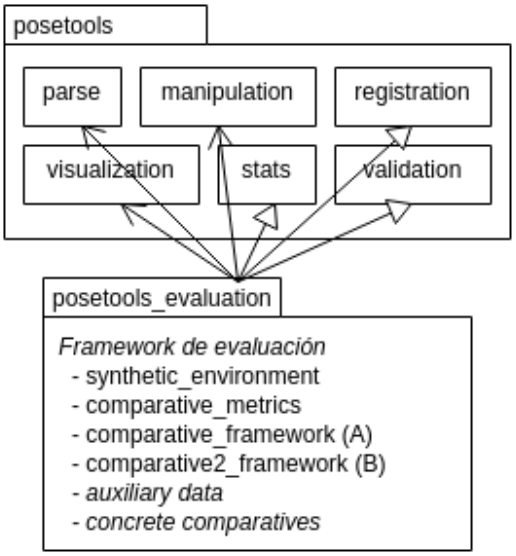
A continuación se presentan algunos ejemplos de las secuencias empleadas en el planteamiento A, donde queda reflejada la magnitud del ruido definida en la sección A.2. En azul se representa la secuencia sintética sin ruido y en rojo ésta misma secuencia tras aplicar el modelo de ruido.



[2] será definido más adelante, sección A.5

A.4. Implementación

La implementación de este módulo se ha apoyado en la librería llegando a emplear todos y cada uno de sus módulos tal y como se puede observar en la figura de la derecha. El módulo más importante ha sido el de validación, donde el componente *live_registration_validation* ha sido el precursor de este componente de evaluación. Por ejemplo, si atendemos a la figura A.1, *live_registration_validation* involucra las etapas 1, 2 y 3a, sólo que se trata como una caja blanca. Para concluir, se presenta el fragmento de código que ejecuta un experimento, donde podemos ver que basta pasarle los métodos desarrollados junto con sus nombres.




```
if 1:
    names = ['LS', 'LS*', 'LS**', 'gRANSAC (5cm)', 'gRANSAC (5cm) + LS', 'gRANSAC (25%)']
    methods = [LS, LS_, LS__, RANSAC_5cm, RANSAC_5cm_LS, RANSAC_adaptative25]

if __name__ == '__main__':
    argv = ['', 'data/track_1.txt']
    framework = comparative_framework

    framework.names = names
    framework.methods = methods
    framework.main(argv)
```

Además, gracias a los "trucos" que se hacen entre bambalinas, la definición de estos métodos queda bastante simplificada.

```
f_fitness_5cm  = ransac.instance_fitness_saturated_error(threshold=0.05)
f_fitness_5cm2 = instance_fitness_error_transfer_function(threshold=0.05)

RANSAC_5cm_LS = run_RANSAC(new_RANSAC(f_fitness_5cm, LS_init=True))
RANSAC_5cm_ = run_RANSAC(new_RANSAC(f_fitness_5cm), f_mod=data_refinement_1)

LS  = rigid_transform_sRt
LS_ = lambda _from,_to: rigid_transform_sRt(*data_refinement_1(_from,_to))
```

Como podemos ver, *new_RANSAC* abstrae los parámetros de inicialización de ransac, de modo que todos comparten los mismos parámetros para asegurar una comparativa justa y actualizada. Además *run_RANSAC* nos permite inyectar lógica de manipulación de los datos a priori, en ese caso, la supresión de espúreos.

A.5. Algoritmos de registro a evaluar

En esta sección se listan los candidatos que optan a ser el algoritmo de registro definido por la herramienta así como aquellos auxiliares más relevantes que se han empleado para realizar la comparativa.

Alg-1) **LS**: método básico. Solución por mínimos cuadrados planteada en 7.5.1.

Alg-2) **LS***: LS con el paso previo de refinamiento propuesto en 7.5.2.1 para la eliminación de espúreos.

Alg-3) **LS****: como LS*, solo que el refinamiento de los datos se realiza de forma iterativa y progresiva hasta que se considera que todos los datos pertenecen al modelo.

Alg-4) **RANSAC**: método de ransac formalizado en 7.5.2.2 donde la función de aptitud es (7.14) con un umbral definido en 5cm.
Nomenclatura: gRANSAC (5cm)

Alg-5) **RANSAC adaptativo**: ransac con función de aptitud (7.14) cuyo umbral viene dado por el primer cuartil (percentil 25) de la función (7.15) aplicada sobre 7.5.1.
Nomenclatura: gRANSAC (25 %)

Alg-6) **RANSAC adaptativo***: como el anterior pero con un paso de refinamiento.
Nomenclatura: gRANSAC* (25 %)

Alg-7) **RANSAC exponencial**: ransac con función de aptitud (7.16)
Nomenclatura: gRANSAC^2 (5cm)

Alg-8) **RANSAC sin consenso**: método propuesto por el Tutor que elimina la necesidad de realizar la clasificación entre inlier y outlier permitiendo una función de aptitud más laxa. Emplea la función (7.17), que es doblemente informada y acotada.
Nomenclatura: ncRANSAC (5cm/1m)

Alg-9) **RANSAC exponencial con contribucion de outliers**: ransac con función de aptitud (7.17). Se trata de (7.16) más una ponderación acotada de los outliers. Su única diferencia con 8 es que realiza el paso de suavizado.
Nomenclatura: $\text{gRANSAC}^{2,2}$ (5cm/1m)

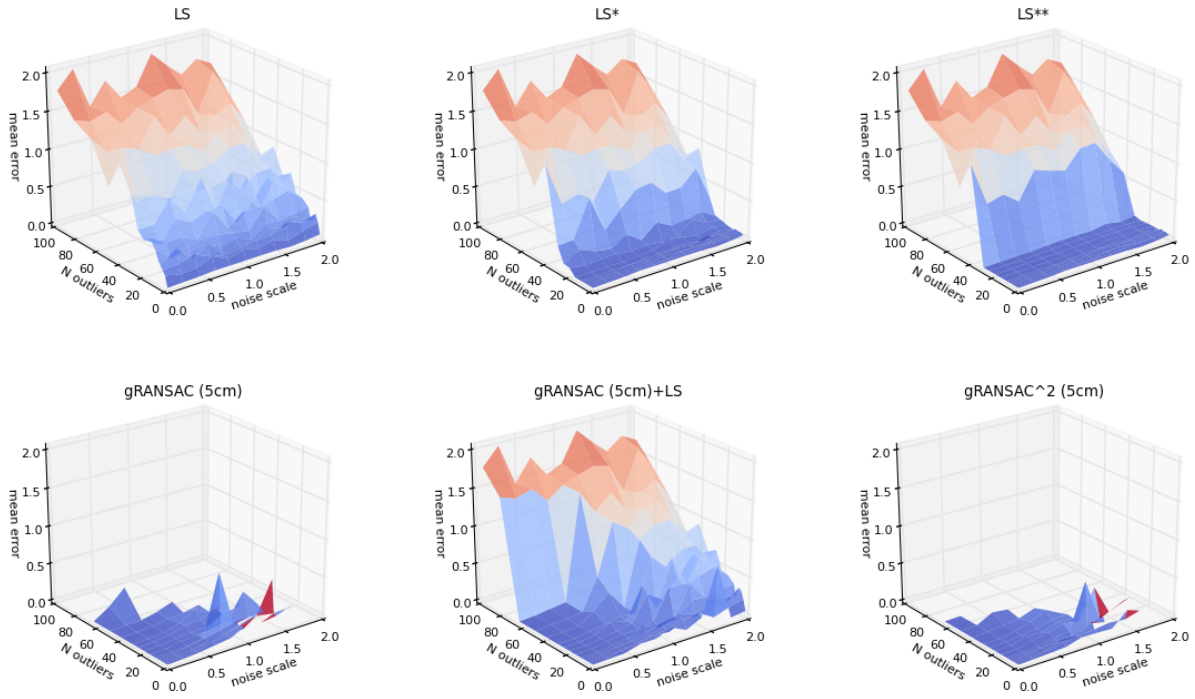
Alg-10) **RANSAC exponencial adaptativo***: fusión de los casos (Alg-6) y (Alg-7).
Nomenclatura: gRANSAC^{2}* (25 %)

La lista presentada no es exhaustiva, ya que se han omitido otras parametrizaciones por motivos de longitud y espacio.

Nota sobre la nomenclatura: gRANSAC es el nombre de la clase que implementa 3-RANSAC

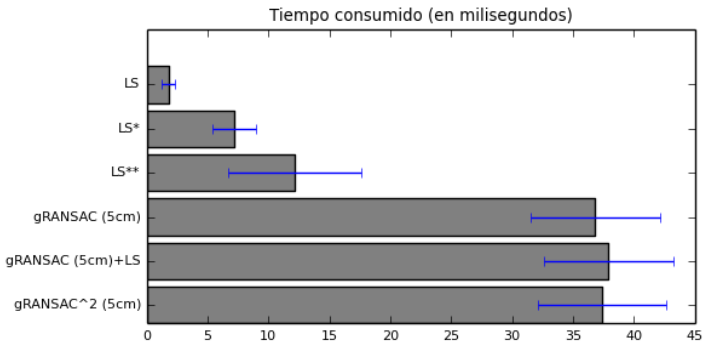
A.6. Resultados

Caso A-I
En este caso quedan reflejadas las tres variantes de mínimos cuadrados: (Alg-1), (Alg-2) y (Alg-3). Así como las variantes de ransac: (Alg-4) y (Alg-7).

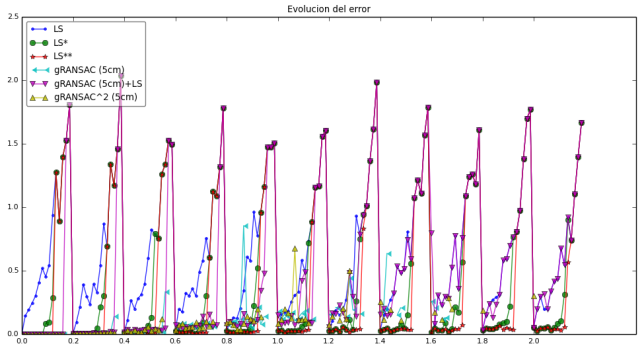


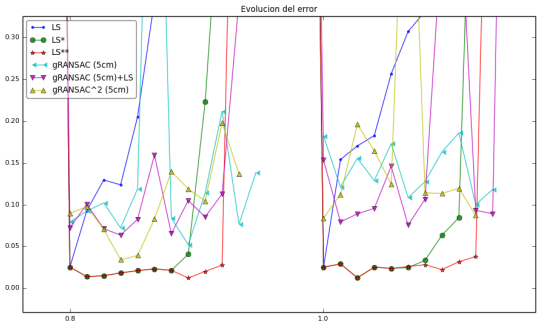
Como podemos ver, el algoritmo de detección de espúreos nos da un cierto grado de resistencia cercanos al 10 % y 30 % respectivamente. Aunque muy inferior al que ofrece RANSAC.
Además se presenta el comportamiento natural del algoritmo de RANSAC implementado, donde no devuelve ningún modelo si los datos no satisfacen las restricciones impuestas. Es decir, para las hipótesis planteadas ninguna ha conseguido que al menos 4 puntos tengan un error teórico inferior a 5cm. En la gráfica central inferior se presenta la conjunción de (Alg-1) y (Alg-4) para ofrecer una comparativa más cómoda.

De forma testimonial se incluye en la ilustración de la derecha el tiempo de ejecución de cada método. Al ser una característica no resolutiva no se volverá a incluir. Basta con decir que (Alg-5) es el más pesado al incluir un cómputo similar al del paso de refinamiento presente en (Alg-2).



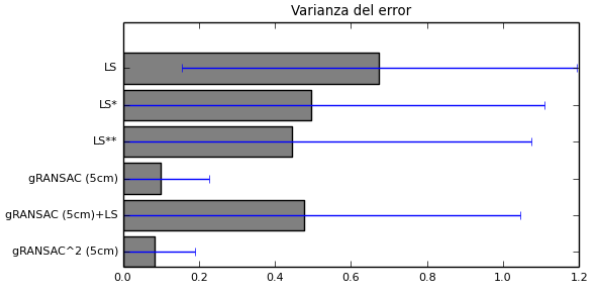
Los comportamientos que se describen son aquellos que se han presentado de forma sistemática. Como es obvio, este documento solo presenta un caso modelo para dar apoyo gráfico a los razonamientos dados.





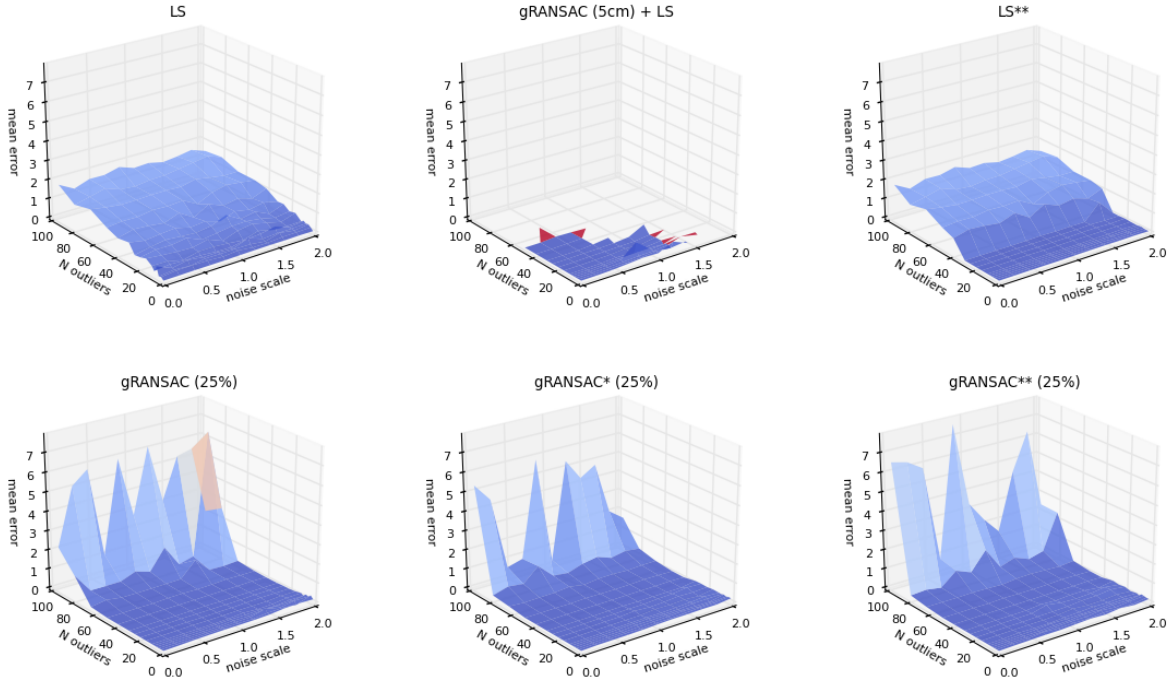
Al desdoblar la superficie y presentarla de forma lineal, esta permite visualizar la evolución del error ante un entorno cada vez más hostil. Aquí podemos ver que (Alg-2) y (Alg-3) logran cotas inferiores antes de llegar a su punto de ruptura.

También cómo la diferencia entre (Alg-4) y (Alg-7) es muy reducida.
En la imagen de la derecha se condensan dichos valores. En ella podemos ver cómo la variante exponencial ofrece un rendimiento levemente superior aunque sistemático, de modo que la ordenación debida por la ponderación causa los efectos deseados.



Caso A-II

En ese caso se agregan (Alg-5) y (Alg-6) tomando (Alg-4) y (Alg-3) como referencias.
Aquí no solo se estudiará la capacidad de outliers adaptativo, sino también se incluirá cómo influye el refinamiento de datos en el algoritmo de RANSAC y qué implicaciones tiene.

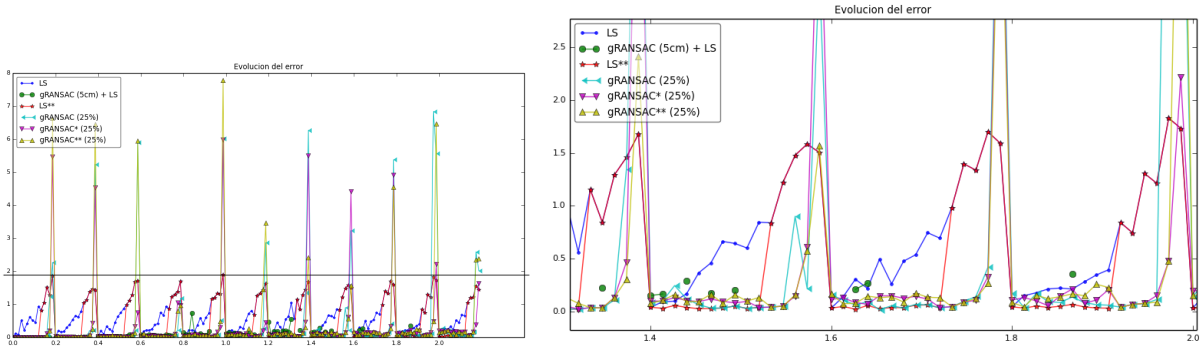


Como se puede observar, el error tiende a infinito cuando el porcentaje de *outliers* es elevado. Concretamente cuando es superior al 75 % de los datos.

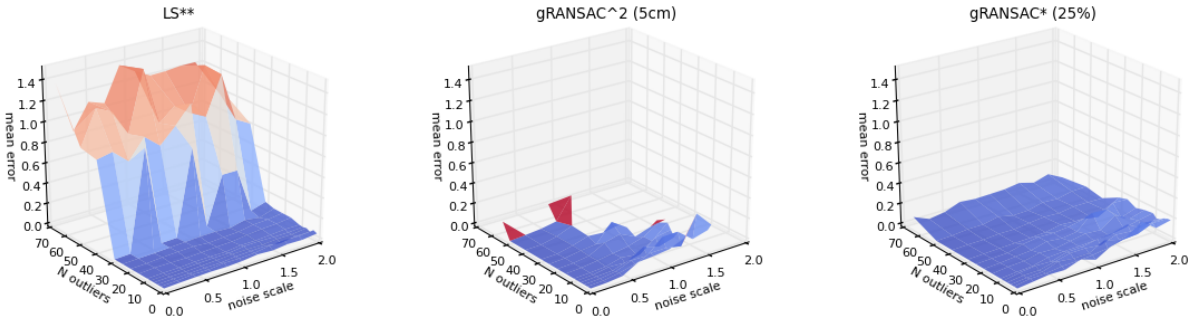
Esto se debe a que aunque en este método no haya que definir un umbral explícitamente, sí necesita a priori una certeza de cuántos datos pertenecen al modelo. De este modo, (Alg-5) exige que al menos un 25 % de los datos sean *inliers*. En caso contrario devuelve una solución

extremadamente mala.
Esto es una desventaja con respecto a (Alg-7), que nos asegura unos criterios de calidad.

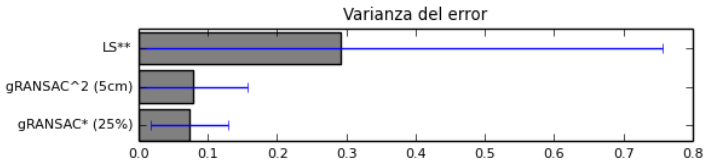
En la gráfica de evolución del error se pueden ver mejor dicho salto en magnitud, donde la linea negra horizontal refleja la cota superior de (Alg-3).



Si trabajamos en los límites de estabilidad del algoritmo (Alg-5), los resultados se vuelven mucho más favorables.

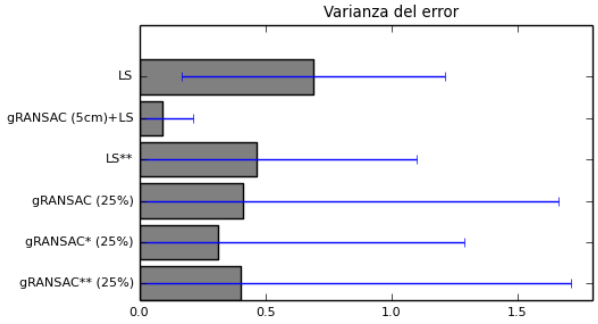


Con un porcentaje de datos espúreos de hasta el 70 %, el método (Alg-6) tiene un rendimiento superior a (Alg-7).



Destacar que la métrica de la derecha tiene efecto sólo donde el método ratifica la existencia de un modelo. Mientras que el rendimiento de (Alg-7) está medido sobre un dominio más reducido, para (Alg-6) incluye todo el dominio. Eso significa que ofrece un rendimiento mejor incluso incluyendo casos más hostiles.

Por último, se discute sobre el grado de refinamiento, que será más fácil de ver en los tests de tipo B.

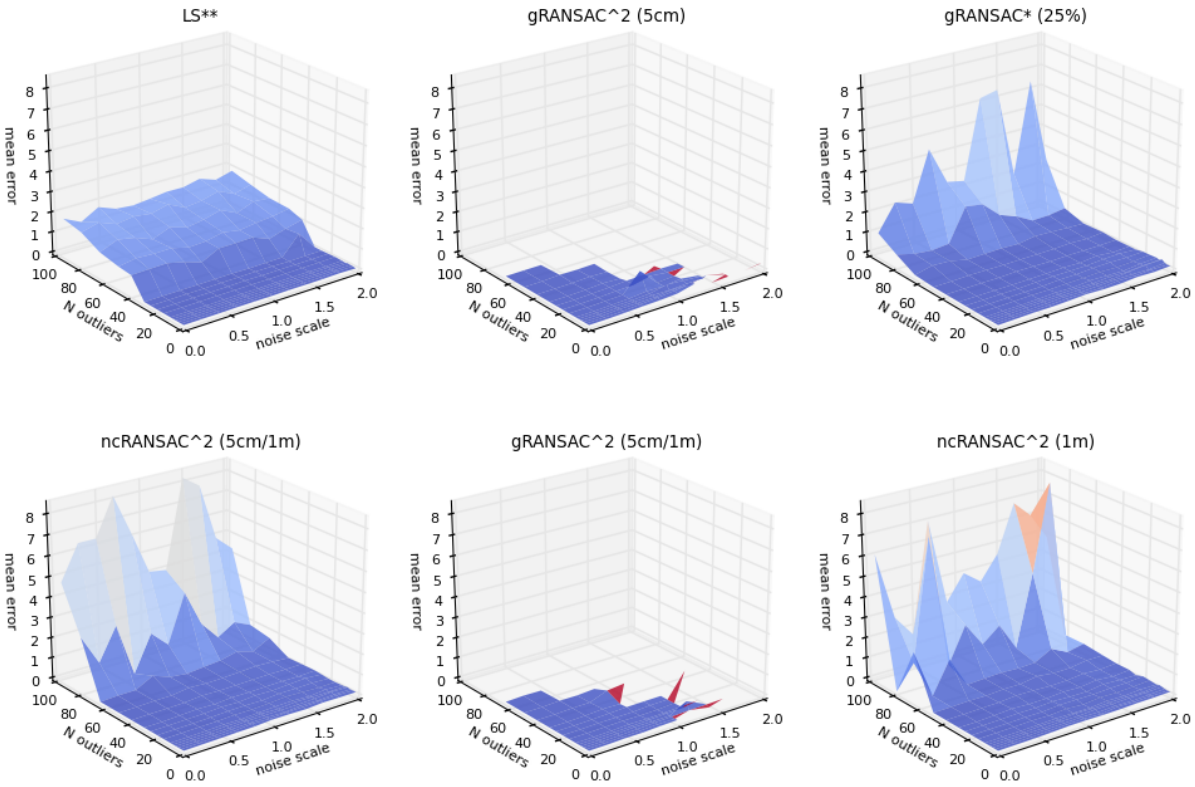


El refinamiento no influye de forma directa a inliers. Afecta de forma indirecta pudiéndose reducir a un pro y a un contra:

- *pro:* la reducción de *outliers* reduce el número de iteraciones necesarias para obtener el mismo grado de certeza.
- *contra:* un refinamiento excesivo puede eliminar *inliers* si el porcentaje de *outliers* es muy elevado o están degenerados en otro modelo.

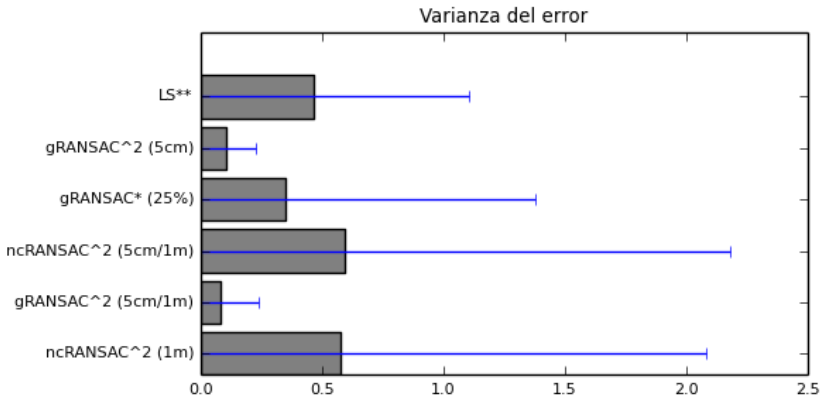
Caso A-III

En este caso se agregan (Alg-8) y (Alg-9) usando como base (Alg-7) y (Alg-6).



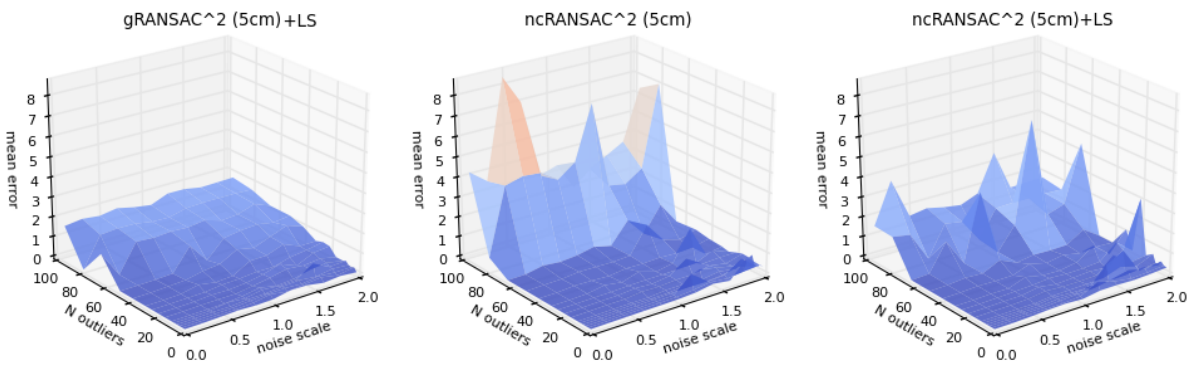
En esta gráfica podemos ver cómo (Alg-8) tiene un comportamiento muy similar a (Alg-6). Además se ha incluido en la parte inferior derecha la versión primordial de (Alg-8). Esta es completamente ajena a la clasificación y por tanto al umbral de *inliers*, por lo que emplea la función de aptitud (7.16).

Si atendemos a la gráfica de la derecha podemos ver cómo la supresión del consenso empeora el rendimiento de forma considerable. Recordemos que la diferencia entre (Alg-8) y (Alg-9) es únicamente la realización del paso de suavizado el cual exige la clasificación entre *inliers* y *outliers*.

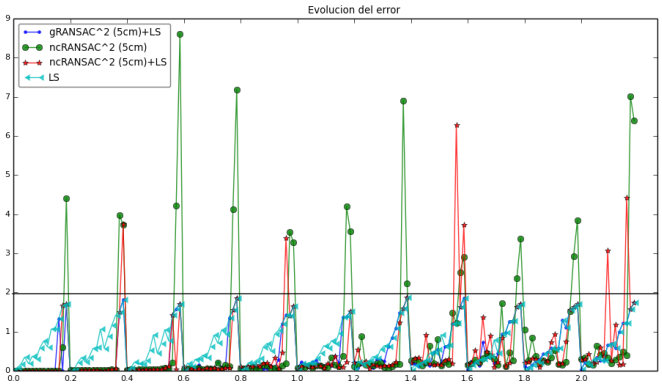


Esto puede verse mejor en el siguiente experimento donde se emplea la misma función de aptitud y valor numérico. La gráfica de la izquierda (a) corresponde a (Alg-7) con una inicialización previa realizada por (Alg-1). La gráfica de la derecha (c) corresponde a la misma configuración pero sin consenso. Por último, la central (b) sirve como puente con el caso de estudio anterior.

(a) (b) (c)

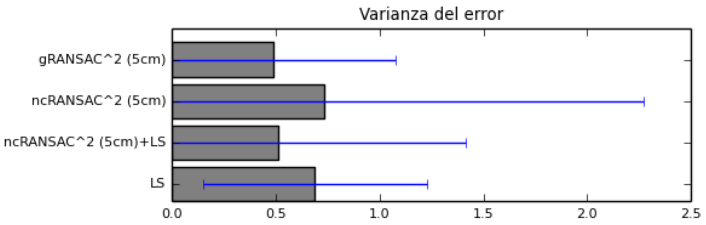


En ella podemos ver cómo (c) empeora la cota superior recomendada por (Alg-1). La diferencia con respecto a (Alg-5) es que aquí el valor de umbral no se ve afectado ni aumentado por el ruido presente en los datos. En su lugar, la pérdida de calidad se debe a que se ha encontrado un conjunto cuyo modelo es mejor que aquel formado por la totalidad de los datos, pero donde los elementos de este conjunto no son representativos debido al ruido.



Este es el motivo por el que (a) se mantiene en la cota superior, situación que solo será cierta si el ruido presente en los datos sigue una distribución simétrica donde realizar el suavizado con un filtro de media anula el ruido.

Por completitud, se presenta la gráfica que condensa el error y sirve para complementar la anteriormente presentada.



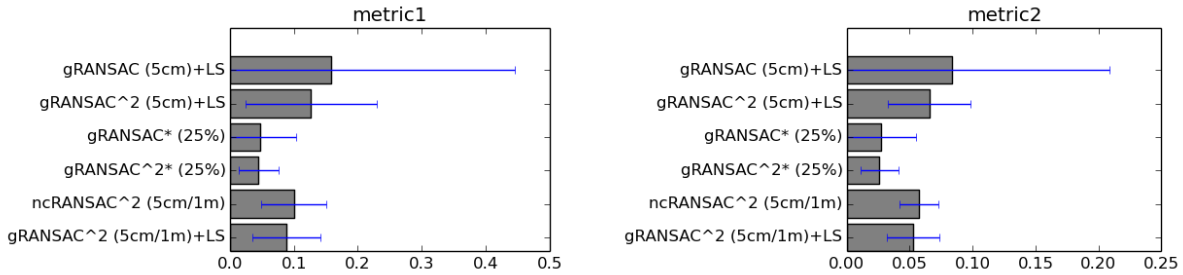
Caso B-I

En este primer caso no se va a aplicar ninguna transformación. Es decir, se va a emplear la identidad. De este modo, el experimento se ceñirá a 100 tracks. Además se ha definido un ruido blanco de 0.5 de varianza y un 50 % de datos espúreos.

Con esta configuración se han evaluado los algoritmos (Alg-4), (Alg-6), (Alg-7), (Alg-8), (Alg-9) y (Alg-10), cuyos resultados serán presentados a continuación.
Sin embargo, antes de ver los resultados es necesario realizar dos apuntes:

- Los algoritmos (Alg-1), (Alg-2) y (Alg-3) han sido descartados de esta comparativa debido a que se situaban a dos órdenes de magnitud del resto, por lo que no eran representativos.
- A todos los algoritmos que pueden rechazar la existencia del modelo se les a añadido una inicialización con (Alg-1). De este modo a través del sufijo *+LS* podemos reconocer dichos algoritmos.

		metric1		metric2	
name	alias	mean	std	mean	std
gRANSAC (5cm)+LS	Alg-4 +LS	1,582e-01	2,871e-01	8,346e-02	1,255e-01
gRANSAC^2 (5cm)+LS	Alg-7 +LS	1,271e-01	1,028e-01	6,551e-02	3,297e-02
gRANSAC* (25 %)	Alg-6	4,760e-02	5,572e-02	2,717e-02	2,768e-02
gRANSAC^2* (25 %)	Alg-10	4,498e-02	3,158e-02	2,585e-02	1,513e-02
ncRANSAC^2 (5cm/1m)	Alg-8	1,003e-01	5,103e-02	5,746e-02	1,564e-02
gRANSAC^2 (5cm/1m)+LS	Alg-9 +LS	8,793e-02	5,315e-02	5,298e-02	2,077e-02

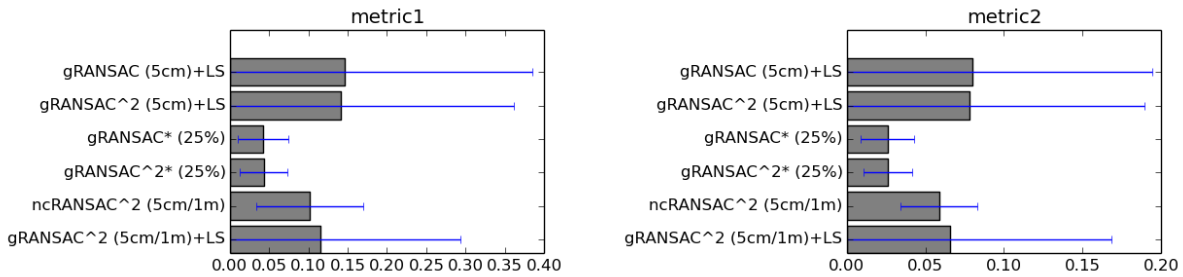


Como podemos observar, (Alg-10) es el que mejor rendimiento presenta junto con (Alg-6).

Caso B-II

Este segundo caso es equivalente al primero solo que se aplicarán transformaciones aleatorias, concretamente 20 por cada track, de modo que estaríamos realizando 2.000 tests a cada método.

		metric1		metric2	
name	alias	mean	std	mean	std
gRANSAC (5cm)+LS	Alg-4 +LS	1,460e-01	2,390e-01	7,987e-02	1,147e-01
gRANSAC^2 (5cm)+LS	Alg-7 +LS	1,415e-01	2,201e-01	7,810e-02	1,117e-01
gRANSAC* (25 %)	Alg-6	4,253e-02	3,232e-02	2,604e-02	1,702e-02
gRANSAC^2* (25 %)	Alg-10	4,302e-02	3,039e-02	2,631e-02	1,553e-02
ncRANSAC^2 (5cm/1m)	Alg-8	1,019e-01	6,832e-02	5,863e-02	2,425e-02
gRANSAC^2 (5cm/1m)+LS	Alg-9 +LS	1,149e-01	1,782e-01	6,556e-02	1,031e-01



Aquí también podemos ver cómo se mantiene la tendencia. Sin embargo en términos de minimización del error (Alg-6) y (Alg-10) se invierten los papeles siendo el primero el de menor error. Esto mismo no ocurre con la varianza.

A.7. Conclusiones

A la vista de los resultados presentados, así como de todos los realizados fuera de esta memoria y que se han intentado dejar representados en los cinco casos relatados, podemos concluir que el mejor método o mejor combinación es: *RANSAC exponencial adaptativo con un paso de refinamiento*. Por tanto será el método predefinido del módulo de registro de la librería para el análisis de los algoritmos de visualSLAM.

Vinculándolo a lo descrito con anterioridad en esta memoria es la confluencia de:

- Método básico: estimación de la transformación rígida 7.5.1
- Resistencia a espúreos I: RANSAC 7.5.2.2
  - Umbral adaptativo.
  - Función de aptitud (7.16)
- Resistencia a espúreos II: IQD+Mahalanobis 7.5.2.1

Número de datos del modelo preliminar de RANSAC
Quizá uno de los parámetros más simples y que por tanto suele omitirse al formalizar el método. Fue denotado como m en (7.10) y corresponde al paso 1 descrito en 7.1.

El número mínimo de puntos para definir una transformación rígida es tres. Sin embargo, para las implementaciones realizadas se ha definido un valor de cuatro, es decir, $3+1$. De este modo el cuarto punto actúa como un consenso a priori.

Si los tres puntos seleccionados son *outliers*, estos pertenecerán al modelo que los contiene. Sin embargo, si el modelo se genera con cuatro *outliers*, estos ya no pertenecerán al modelo. Esto se debe a que la solución al problema sobredeterminado es una interpolación de los cuatro modelos formados tres a tres. Así el modelo obtenido podrá caer en un espacio vacío.

Este consenso a priori acota el caso peor cuando el número de iteraciones no es suficiente siendo (Alg-8) el método más beneficiado por esta elección.

Como es lógico, se han probado otros valores más elevados. Estos proporcionan un mínimo de calidad y suavidad siendo un caso extremo del tratamiento del umbral K descrito en el paso 4 del método de RANSAC.

APÉNDICE

B Etapa de Ejecución: detalles de implementación

En este anexo se concretan algunos de los detalles de la etapa de ejecución, ya que estos, por motivos de extensión y por su menor relevancia han quedado fuera de esta memoria.
En primer lugar se realizará una breve introducción de las tecnologías empleadas para la implantación de la parte correspondiente a la ejecución de los algoritmos de VisualSLAM. Tras ello, se incluye un fragmento del código responsable de la ejecución de dichos algoritmos.

B.1. CoW

Copy-on-Write es una metodología por la cual se mantiene un sistema de versiones a través de la duplicación de los archivos y/o bloques en lugar de su sobre-escritura. Existen múltiples formas de implantar este sistema según los requisitos de versionado. En este caso, donde sólo nos interesa guardar el estado actual del sistema, podemos emplear *LVM snapshots* o *UnionFS*.

Se decidió el uso de *UnionFS* por su experiencia con distribuciones *Live* de Ubuntu, cuya gestión se simplifica otra herramienta, *schroot*, también conocida con anterioridad.

Con esta doble conjunción tenemos un sistema de capas de reducido tamaño que puede aprovechar el 100 % de los recursos hardware del ordenador.
El método planteado tiene sus pros y sus contras. Para simplificar, hace fácil lo que en Docker es difícil, y viceversa.

B.2. Docker

Docker es un sistema de virtualización basado en contenedores que emplea *debootstrap* y *UnionFS*. Está enfocado a la realización de una tarea por contenedor, aunque se puede llegar a configurar un entorno gráfico y de desarrollo.

Ha sido elegido por ofrecer un entorno autocontenido y repetible, pero en mayor medida porque la experiencia obtenida en esta herramienta ha llevado a la consecución de un pequeño *toolkit* que simplifica el proceso de creación de contenedores. Como ejemplo, se adjunta a continuación la definición de alto nivel de un contenedor, que tras su compilación superará las 150 líneas.
Esta metodología permite crear el entorno de ejecución de cada uno de los algoritmos cambiando, si fuera necesario, las líneas 7, 8 o 9. De lo contrario sería necesario duplicar todas las líneas comunes llevando en último término a la obsolescencia del código y la ausencia de la corrección de los errores detectados.

```
1  FROM ubuntu:14.04
2  MAINTAINER Victor Arribas <v.arribas.urjc@gmail.com>
3
4  RUN apt-get update
5  include(fragments/apt-fast)
6
7  # ENV ROS_PACKAGES_EXTRA "libsuitesparse-dev ros-indigo-libg2o"
8  include(fragments/ros-indigo)
9  include(orb_slam2_ros)
10
11 include(fragments/varribas-devel-toolkit)
12 include(fragments/varribas-devel-toolkit-x11)
13 include(fragments/user-layer)
```

B.3. Envoltorio de ejecución

El envoltorio de ejecución es una de las tarea compleja ya que exige aglutinar un gran conocimiento en el que se deben buscar interfaces, patrones comunes y puntos críticos en pos de una arquitectura genérica que permita la reutilización y sustitución transparente de los diversos elementos y módulos detectados.

Como es lógico, esta transparencia no se ha obtenido, pero se ha sembrado la semilla para alcanzarlo. Como ejemplo se presenta la capa inferior que permite ejecutar la implementación de SVO. En este fragmento de código podremos ver:

1. **API posiblemente identificada para cualquier algoritmo.** Esta se reduce a tres elementos: secuencia de entrada, archivo de calibración de la cámara y directorio de salida. La definición de la salida es un punto crítico ya que el número de artefactos generados puede ser indeterminado.
2. **Complejidad en el tratamiento de la entrada.** Esta queda reflejada a lo largo de las líneas 44-57, donde se está ofreciendo compatibilidad con los datasets de RPG[1] y TUM.
3. **Entendimiento avanzado de ROS.** Levantar tantos nodos de ROS tiene su misterio. Además, su configuración en frío o en caliente exige un conocimiento avanzado de la API de línea de comandos de ROS.
4. **Capa atada a la implementación.** Como se puede ver, este primer envoltorio está completamente acoplado a la implementación que se quiere poder ejecutar. Por tanto, se necesitará una por cada algoritmo. Sin embargo, ha sido formalizada en su mínima expresión delegando en la capa superior las cuestiones genéricas.
5. ***sigterm().*** Diseño propio para la terminación gentil de un proceso. Permite al proceso una terminación ordenada, pero asegura su exterminio en caso de que el proceso no responda.
6. **Modo pánico.** La realización de los envoltorios ha sido realizada en modo pánico. Es decir, con los *flags* "`set -e`" y "`set -u`" que aseguran que todos y cada uno de los comandos se han completado sin errores.

[1] `rpg.ifi.uzh.ch/datasets`

Código B.1: runner_svo_bag.sh

```
#!/bin/sh
#
# This file is part of runners
# https://gitlab.com/varribas-pfm/runners
#
# Copyright (c) 2016
# Author: Victor Arribas <v.arribas.urjc@gmail.com>
# License: GPLv3 <http://www.gnu.org/licenses/gpl-3.0.html>
# Scope: Master Thesis in Computer Vision <https://gitlab.com/groups/
    varribas-pfm>
#
# Usage:
# runner_svo_bag <bag file> <calibration file> [output prefix]
# Example:
# ./runner_svo_bag.sh /media/datasets/dataset-tum/
    rgbd_dataset_freiburg1_xyz.bag "$(rospack find svo_ros)/param/
    camera_tum1.yaml" /tmp/svo
# ./runner_svo_bag.sh /media/datasets/svo/airground_rig_s3_2013-03-18_21
    -38-48.bag "$(rospack find svo_ros)/param/camera_pinhole.yaml" /tmp/svo

set -e
set -u

# global config
dataset=${1} #-/media/datasets/dataset-tum/rgbd_dataset_freiburg1_xyz.bag}
calibration=${2} #-camera_tum1.yaml} [1,2,3]
output_directory=${3-$PWD}

dataname=$(basename $dataset)
dataname=${dataname%.bag}

poses=${output_directory}/${dataname}-poses.txt
ofps=${output_directory}/${dataname}-fps.txt
logfile=${output_directory}/${dataname}.log


echo "Power␣up␣nodes"
rosrun pose2file pose4tum.py _topic:=/svo/pose4tum _file:=$poses &
pid_pose2file=$!
rosrun pose2file posecov2pose.py _in_topic:=/svo/pose _out_topic:=/svo/
    pose4tum &
pid_marker2pose=$!

rosrun pose2file svo_info2fps.py in_topic:=/svo/info out_topic:=/svo/fps &
pid_svofps=$!
rosrun pose2file fps_listener.py _topic:=/svo/fps _file:=$ofps &
pid_fps=$!

# Patch for TUM datasets
if echo "$dataname" | grep -q 'rgbd_dataset'; then
        TUM_PATCH=1
        CAM_TOPIC=/camera/rgb/image_mono
```

```
48          BAG_ARGS=''
49          BAG_POST_ARGS="/camera/rgb/image_color:=/camera/rgb/image_raw"
50          ROS_NAMESPACE=camera/rgb rosrun image_proc image_proc &
51          pid_image_proc=$!
52  else
53          TUM_PATCH=0
54          CAM_TOPIC=/camera/image_raw
55          BAG_ARGS=''
56          BAG_POST_ARGS=''
57  fi
58
59  # SVO
60  ROS_NAMESPACE=svo rosparam load $calibration
61  ROS_NAMESPACE=svo rosparam load  "$(rospack␣find␣svo_ros)/param/vo_accurate
        .yaml"
62  #ROS_NAMESPACE=svo rosparam set 'init_min_inliers' 30
63  rosrun svo_ros vo \
64    _cam_topic:=$CAM_TOPIC \
65    _accept_console_user_input:=false \
66    </dev/null &
67  ret=$?
68  pid_svo_ros=$!
69  sleep 3
70  echo "SVO␣status:" $?
71
72  sigterm(){
73          local signals='-2 -15 -9'
74          for signal in $signals
75          do
76                  if ps | grep -q $1; then
77                          kill $signal $1
78                          sleep 2
79                  else
80                          break
81                  fi
82          done
83  }
84
85  echo "Execute␣dataset␣$dataname"
86  rosbag play --quiet --clock $BAG_ARGS $dataset $BAG_POST_ARGS &
87  wait $!
88  sleep 3
89  sigterm $pid_svo_ros
90  if [ $TUM_PATCH -eq 1 ]; then
91          sigterm $pid_image_proc
92  fi
93  sigterm $pid_marker2pose
94  sigterm $pid_pose2file
95  sigterm $pid_svofps
96  sigterm $pid_fps
97  echo "Dataset␣$dataname␣completed"
98  ps
99  echo
```

Bibliografía

[AB15] Víctor Arribas and Javier Benito. Presentación RANSAC. `https://goo.gl/uyTZJv`, 2015. Máster en Visión Artificial, URJC.

[Arr15] Víctor Arribas. Quaternions, una rápida introducción. `https://goo.gl/I39nol`, 2015. Máster en Visión Artificial, URJC.

[Blo04] Jonathan Blow. Understanding slerp, then not using it. *Game Developer Magazine*, 2004.

[BMG09] José-Luis Blanco, Francisco-Angel Moreno, and Javier González. A collection of outdoor robotic datasets with centimeter-accuracy ground truth ‡. *Autonomous Robots*, 27(4):327–351, November 2009.

[CDM08] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse depth parametrization for monocular slam †. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.

[CGDM10] Javier Civera, Oscar G Grasa, Andrew J Davison, and JMM Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry ‡. *Journal of Field Robotics*, 27(5):609–631, 2010.

[Dav03] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera †. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410. IEEE, 2003.

[Der10] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4:2–3, 2010.

[DKL98] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation* †. Datalogisk Institut, Københavns Universitet, 1998.

[DRMS07] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[ELF97] David W Eggert, Adele Lorusso, and Robert B Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5-6):272–290, 1997.

[ESC13] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera †. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1449–1456, 2013.

[ESC14] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FPS14] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset †. *The International Journal of Robotics Research*, page 0278364913491297, 2013.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite †. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[Han05] Andrew J Hanson. Visualizing quaternions †. In *ACM SIGGRAPH 2005 Courses*, page 1. ACM, 2005.

[HC14] Alejandro Hernández Cordero. Autolocalización visual aplicada a la realidad aumentada †. Master's thesis, Universidad Rey Juan Carlos, 2014. `http://jderobot.org/Ahcorde-tfm`.

[HNAD12] Ankur Handa, Richard A Newcombe, Adrien Angeli, and Andrew J Davison. Real-time camera tracking: When is high frame-rate best? †. In *European Conference on Computer Vision*, pages 222–235. Springer, 2012.

[Hor87] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[HWMD14] Ankish Handa, Thomas Whelan, John McDonald, and Andrew J Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *Robotics and automation (ICRA), 2014 IEEE international conference on*, pages 1524–1531. IEEE, 2014.

[Kei11] KeithM. Math Magician - Lerp, Slerp, and Nlerp. `https://keithmaggio.wordpress.com/2011/02/15/math-magician-lerp-slerp-and-nlerp/`, 2011.

[KM07] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[LD10] Steven Lovegrove and Andrew J Davison. Real-time spherical mosaicing using whole image alignment †. In *Computer Vision–ECCV 2010*, pages 73–86. Springer, 2010.

[LEF95] Adele Lorusso, David W Eggert, and Robert B Fisher. *A comparison of four algorithms for estimating 3-D rigid transformations*. University of Edinburgh, Department of Artificial Intelligence, 1995.

[LF05] José Alberto López Fernández. Localización de un robot con visión local ‡. Master's thesis, Universidad Rey Juan Carlos, 2005.

[LR10] Luis Miguel López Ramos. Autolocalización en tiempo real mediante seguimiento visual monocular †. Master's thesis, Universidad Rey Juan Carlos, 2010. `http://jderobot.org/Lm.lopez-pfc-teleco`.

[Mal04] Ezio Malis. Improving vision-based control using efficient second-order minimization techniques †. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 2, pages 1843–1848. IEEE, 2004.

[MAMT15] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system †. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015.

[MAT14a] Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based slam †. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 846–853. IEEE, 2014.

[MAT14b] Raúl Mur-Artal and Juan D Tardos. ORB-SLAM: tracking and mapping recognizable features. In *Proceeding of Robotics: Science and Systems (RSS) Workshop on Multi View Geometry in Robotics*, 2014.

[MF14] Alberto Martín Florido. Navegación visual en un cuadricóptero para el seguimiento de objetos †. Master's thesis, Universidad Rey Juan Carlos, 2014.

[MO14] Daniel Martín Organista. Odometría visual con sensores RGBD ‡. Master's thesis, Universidad Rey Juan Carlos, 2014.

[Nis04] David Nistér. An efficient solution to the five-point relative pose problem †. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004.

[NLD11] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[Ora12] EPM Information Development Team Oracle. Outlier detection methods. `https://docs.oracle.com/cd/E17236_01/epm.1112/cb_statistical/frameset.htm?ch07s02s10s01.html`, 2012.

[SEE+12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[SRL15] Ignacio San Román Lana. Odometría visual 3D para autolocalización de una cámara móvil en tiempo real †. Master's thesis, Universidad Rey Juan Carlos, 2015. `http://jderobot.org/Isanroman-tfm`.

[SSS06] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3D [‡]. In *SIGGRAPH Conference Proceedings*, pages 835–846. ACM Press, 2006. http://research.microsoft.com/en-us/um/redmond/groups/ivm/PhotoTours/.

[Wik16a] WikiHow. Cómo calcular datos atípicos. http://es.wikihow.com/calcular-datos-atípicos, 2016.

[Wik16b] Wikipedia. Mahalanobis distance. https://en.wikipedia.org/wiki/Mahalanobis_distance, 2016.

[Wik16c] Wikipedia. Outliers and detection methods. https://en.wikipedia.org/wiki/Outlier, 2016.

[Zul09] Marco Zuliani. RANSAC for Dummies [†]. *Vision Research Lab, University of California, Santa Barbara*, 2009.