

**MÁSTER UNIVERSITARIO
EN VISION ARTIFICIAL**

Curso Académico 2018/2019

Trabajo Fin de Master

Herramienta de evaluación cuantitativa de
algoritmos Visual SLAM

Autor: Elías Barcia Mejias

Tutores: José María Cañas Plaza , Eduardo Perdices García

Resumen

Actualmente la investigación y desarrollo en robótica móvil está en pleno auge. Los robots modernos están equipados con múltiples sensores y uno de los más utilizados son las cámaras, ya que permiten al robot captar en imágenes todo el entorno que le rodea. En contra partida, el procesado de imágenes conlleva una carga notable de CPU debido a la enorme cantidad de información que puede aportar cada imagen.

Una de las funcionalidades más importantes que se persigue, es que los robots móviles puedan desplazarse por su entorno y navegar desde la posición A a la posición B de forma autónoma. Esta tarea no resulta muy complicada en entornos estructurados, donde el robot conoce de antemano el terreno por el que se mueve o sabe de la existencia de alguna baliza que le dé pistas de su posición. Pero en entornos no estructurados, donde el robot desconoce por completo el terreno, carece de mapas y no existe a priori ningún tipo de marca o baliza que pueda guiar al robot, la navegación resulta mucho más compleja.

En exteriores, podríamos guiar al robot mediante GPS, pero la señal GPS no llega con la suficiente potencia a todas partes. Por ejemplo, en interiores de edificios o en zonas subterráneas, o mejor aún, imaginemos que enviásemos el robot a explorar la superficie del planeta Marte, donde la señal GPS es inexistente. ¿Cómo se las arreglaría el robot para desplazarse por el terreno de forma autónoma sin perderse?

Hoy en día ya existe una familia de técnicas que permite al robot navegar de manera autónoma por zonas desconocidas para él, esta técnica se llama VisualSLAM.

Visual SLAM (*Simultaneous Localization and Mapping*) es una técnica utilizada principalmente con robots móviles y que aporta al robot la capacidad de autolocalizarse y generar mapas del entorno que le rodea en tiempo real. Gracias a ese mapa y principalmente a esa autolocalización se pueden utilizar las técnicas de navegación autónoma, que requieren inevitablemente de una estimación de posición propia fiable. VisualSLAM básicamente se comporta como una caja negra que procesa las imágenes en secuencia captadas por una o varias cámaras. A partir de esas imágenes el robot es capaz de obtener su posición 3D en

el mundo que le rodea. De esta forma el robot podrá desplazarse en su entorno de forma autónoma sin perderse.

El robot debe contar con una capacidad de cálculo suficiente que le permita ejecutar el software de procesado de imágenes y, al mismo tiempo, realizar la generación de mapas. Estas tareas requieren ser ejecutadas en tiempo real, unos 30 fotogramas por segundo. Es posible utilizar la técnica de VisualSLAM hoy en día en pequeños dispositivos gracias al aumento de su potencia de computación.

Dependiendo del tipo de cámaras con las que esté equipado el robot, tendrá mayor o menor capacidad de ejecutar VisualSLAM. Como mínimo, el robot debe tener una cámara RGB, muy común en los drones, aunque también puede tener 2 cámaras estéreo que le ayudarán a representar el entorno en 3D con mayor fiabilidad. Otras cámaras, como las RGBD se ayudan de un sensor de profundidad que también capacita al robot para representar en tres dimensiones el mundo que les rodea con mayor robustez y precisión.

Los algoritmos de Visual SLAM se componen de varias fases, dependiendo de como esté diseñada cada fase el algoritmo proporcionará una posición y generará un mapa con mayor o menor exactitud o con mayor o menor velocidad. Por tanto sería conveniente disponer de alguna herramienta que permita comparar cuantitativamente la calidad o los errores de precisión de cada algoritmo de tal forma que nos ayude a encontrar algoritmos óptimos que nos proporcionen la posición y la creación de mapas con gran precisión y velocidad. En este proyecto se ha creado la herramienta SLAM TestBed, que ha sido diseñada para que mida el error de la misma trayectoria calculada por distintos algoritmos.

Índice general

1. Introducción	1
1.1. Visión Artificial	2
1.1.1. Bloques habituales en sistemas de Visión Artificial	2
1.1.2. Aplicaciones de Visión Artificial	4
1.2. Visual SLAM	5
1.2.1. Aplicaciones en Visual SLAM	6
1.2.2. Conceptos	12
1.2.3. Problemas abiertos en Visual SLAM	13
1.3. Estructura del documento	14
2. Objetivos	16
2.1. Objetivos	16
2.2. Requisitos	17
2.3. Metodología y plan de trabajo	18
3. Estado del Arte	20
3.1. Algoritmos de Visual SLAM	20
3.1.1. Algoritmo MonoSLAM	20
3.1.2. Algoritmo PTAM	22
3.1.3. Algoritmo ORB-SLAM	23
3.1.4. DSO y LDSO	24
3.1.5. Comparativa de los algoritmos más representativos	27
3.2. Herramientas y datasets para evaluar algoritmos SLAM	28

4. Herramienta SLAMTestbed	34
4.1. Diseño	34
4.2. Estimador PCA y Cálculo de Escala	38
4.3. Estimador Offset Temporal	39
4.4. Interpolador temporal	41
4.5. Registro Espacial	42
4.6. Cálculo de Estadísticas	44
4.7. Interfaz Gráfico de Usuario	45
5. Validación experimental	48
5.1. Módulo Transformador	49
5.2. Pruebas de transformaciones individuales	50
5.3. Pruebas de transformaciones en parejas	54
5.4. Pruebas de transformaciones combinadas	66
6. Conclusiones	71
6.1. Conclusiones	71
6.2. Líneas Futuras	72
Bibliografía	74

Capítulo 1

Introducción

El presente TFM se encuadra dentro de la Visión Artificial y en particular en el campo de la Autolocalización Visual, que tiene numerosas aplicaciones entre las que destacan la robótica y la realidad aumentada. Actualmente la investigación y desarrollo en robótica móvil está en pleno auge. Los robots modernos están equipados con múltiples sensores y uno de los más utilizados son las cámaras, ya que permiten al robot captar en imágenes todo el entorno que le rodea. En contra partida, el procesado de imágenes conlleva una carga notable de CPU debido a la enorme cantidad de información que puede aportar cada imagen.

Una de las funcionalidades más importantes que se persigue, es que los robots móviles puedan desplazarse por su entorno y navegar desde la posición A a la posición B de forma autónoma y para ello una habilidad fundamental es la Autolocalización. Esta tarea no resulta muy complicada en entornos estructurados, donde el robot conoce de antemano el terreno por el que se mueve o sabe de la existencia de alguna baliza que le dé pistas de su posición. Pero en entornos no estructurados, donde el robot desconoce por completo el terreno, carece de mapas y no existe a priori ningún tipo de marca o baliza que pueda guiar al robot, la Autolocalización resulta mucho más compleja.

Visual SLAM (*Simultaneous Localization and Mapping*) es una técnica utilizada principalmente con robots móviles equipados con cámaras y que aporta al robot la capacidad de autolocalizarse y generar mapas del entorno que le rodea en tiempo real gracias al procesamiento de imágenes. Gracias a ese mapa y principalmente a esa autolocalización se pueden utilizar las técnicas de navegación autónoma, que requieren inevitablemente de una estimación de posición propia fiable.

El robot debe contar con una capacidad de cálculo suficiente que le permita ejecutar

el software de procesado de imágenes y al mismo tiempo realizar la generación de mapas. Estas tareas requieren ser ejecutadas en tiempo real, unos 30 fotogramas por segundo.

1.1. Visión Artificial

La Visión Artificial es una rama científico técnica creada para extraer y procesar información a partir de imágenes.

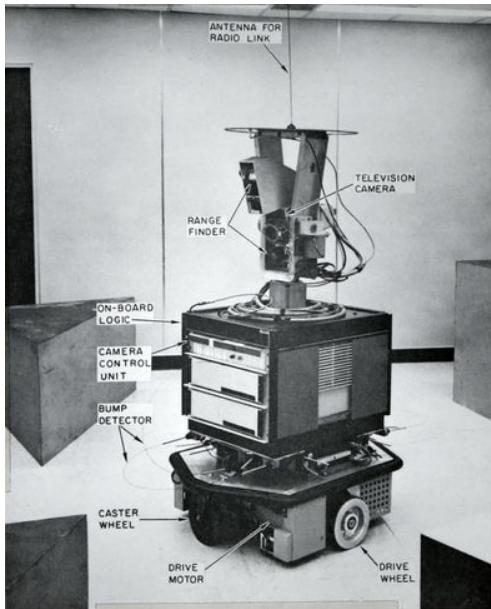
Sus inicios, se remontan hacia mediados del siglo pasado, cuando en 1961 Larry Roberts desarrolló en la universidad un programa que era capaz de ver una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva, para ello tuvo que utilizar los dos componentes principales de un sistema de visión artificial , una cámara y necesariamente un ordenador. Pero debido a la alta complejidad de las tareas de visión artificial y a la primitiva capacidad de proceso de las computadoras de la época, los resultados en la investigación sobre visión artificial fueron muy limitados y podemos decir que la evolución de la visión artificial ha ido ligada a los avances en la computación. Con la aparición de microprocesadores más rápidos, el aumento exponencial de la memoria y la creación y mejora de los algoritmos se han ido consiguiendo mejores resultados hasta crear sistemas de visión artificial que son capaces de operar en tiempo real, permitiendo que un automóvil sea capaz de conducir de forma autónoma, o que un robot sea capaz de agarrar una pelota cuando se la lanzamos.

1.1.1. Bloques habituales en sistemas de Visión Artificial

La mayoría de sistemas de visión artificial o visión por computador están compuestos por dos elementos principales: El sistema de adquisición de imágenes y el sistema de procesado de imágenes. El primero se compone de la iluminación, captura de imágenes y sistemas de adquisición de señales. El segundo implementa los algoritmos de visión que procesan las imágenes para extraer información de ellas.

- **El sistema de iluminación:** Está compuesto por todos los elementos que iluminan los objetos con cualquier tipo de radiación electromagnética. Como ejemplo de estos artefactos podríamos citar la luz natural del sol, o la luz artificial proporcionada por lámparas, lasers o leds.

- **El sistema de captura de imagen:** Transforma en señales eléctricas la luz que se refleja en los objetos. El elemento más utilizado son las cámaras, tanto de espectro visible como de espectro invisible.



(a)

Figura 1.1: Robot Shakey. En los albores de la Visión Artificial

- El sistema de adquisición de señales: Las imágenes capturadas por las cámaras se utilizan para generar señales de vídeo. Su principal objetivo es enviar la señal de vídeo a la entrada de datos del ordenador.

- Sistema de procesamiento: Suele ser un ordenador u otro dispositivo con capacidad de computación que implementa los algoritmos necesarios para procesar la imagen digital y para elaborar la información requerida por el sistema de visión artificial

El sistema de procesamiento de imágenes suele componerse de las siguientes fases:

1. Preproceso: Durante esta fase la imagen puede ser adaptada para extraer mejor la información requerida por los algoritmos o métodos usados. El principal objetivo de esta fase es obtener una mejor calidad de la imagen de entrada , usando técnicas como filtrados de ruidos, convolución, resaltado de bordes etc
2. Segmentación: En esta fase la imagen es dividida en áreas de interés. Por ejemplo diferenciando objetos cuadrados de objetos esféricos o seleccionando las líneas de la carretera obviando el resto de la imagen. Para este propósito se pueden utilizar varias técnicas: umbrales, discontinuidades, crecimiento de regiones, filtros de color, detección de movimiento, etc
3. Clasificación: Una vez la imagen ha sido dividida por regiones de interés (Regions of Interest), se pueden extraer las características específicas de cada una. Esto puede

realizarse por análisis morfológico , por texturas o usando técnicas de clasificación de color.

- **Sistemas Periféricos:** Se trata de los elementos receptores de información, incluyendo monitores, dispositivos que usan la información para tomar decisiones etc.

1.1.2. Aplicaciones de Visión Artificial

Hoy en día, las aplicaciones de la visión por computador están creciendo muy deprisa, debido a la disponibilidad de hardware barato que es capaz de ejecutar complejos algoritmos de visión artificial en un tiempo razonable. Por ejemplo, podemos encontrar aplicaciones en robótica para vehículos no tripulados, en medicina la visión artificial ya ayuda en diagnósticos mediante análisis de imágenes de los pacientes (cáncer, enfermedades degenerativas , etc), en astronomía ayuda a generar imágenes de mayor calidad etc.

Actualmente, la visión artificial se utiliza en muchos procesos científicos, industriales y militares, por ejemplo para el reconocimiento de objetos o en el seguimiento de éstos:

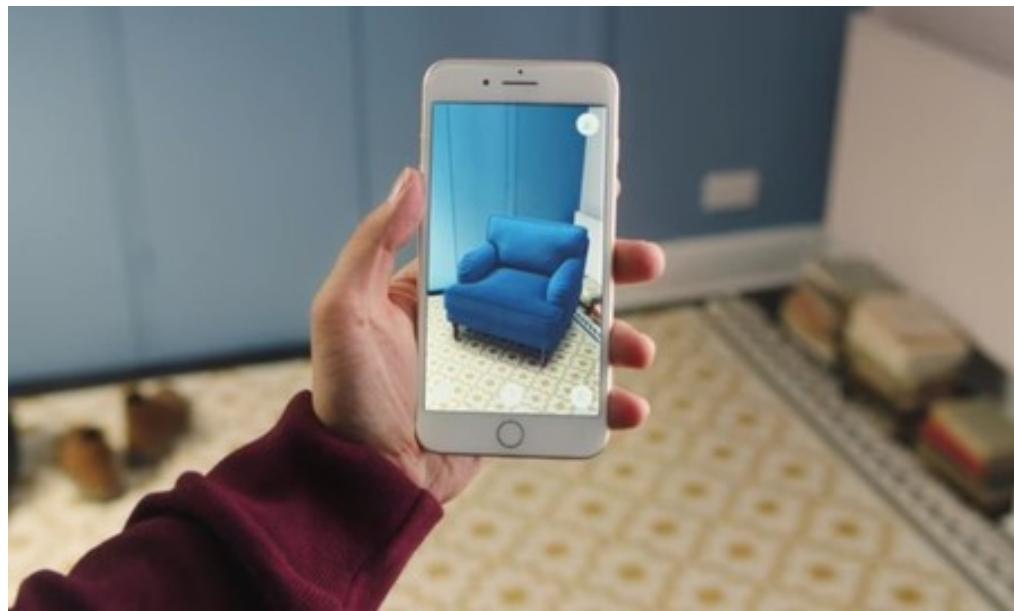
Reconocimiento de objetos: Se trata de buscar unas propiedades concretas de un determinado objeto (forma, color o cualquier otro patrón) en una imagen para determinar si un objeto se encuentra o no en ella. Por ejemplo mediante la obtención de píxeles característicos que destaque en la imagen o utilizándose técnicas de Deep Learning con redes neuronales.

Seguimiento de objetos: Tras ser detectado, se pueden realizar tareas de seguimiento de un objeto. Podrá efectuarse dicho seguimiento teniendo en cuenta sus propiedades (texturas, bordes, etc) o analizando su desplazamiento respecto a imágenes anteriores.

Reconocimiento de caracteres (OCR): Una de las aplicaciones más populares para la visión artificial es el reconocimiento de caracteres (OCR). El propósito de estos sistemas son la identificación de caracteres, por ejemplo hay aplicaciones que te permiten sacar una foto a la lista de componentes de un producto envasado y la aplicación gracias al OCR puede revisar todos los ingredientes del alimento y avisar si el producto contiene algún elemento al que pueda ser alérgico el usuario, como por ejemplo el gluten.

Aplicaciones de realidad aumentada: gracias a la Visión Artificial es posible capturar imágenes del mundo real y procesarlas añadiendo la posibilidad de introducir en estas imágenes elementos artificiales nuevos (gráficos 3D) que encajarían en las dimensiones de las imágenes capturadas. Por ejemplo podríamos captar imágenes de una habitación de

la casa a través de la cámara del móvil y mediante realidad aumentada añadir mobiliario nuevo virtual que sólo sería observable desde la pantalla del smartphone.



(a)

Figura 1.2: Ejemplo de realidad aumentada, el sillón sólo existe en la pantalla del smartphone

1.2. Visual SLAM

Visual Simultaneous Localization And Mapping (Localización y Mapeo Visual Simultáneo). Se refiere al proceso de estimar la posición y orientación de una cámara con respecto al mundo que le rodea, mientras que simultáneamente mapea el entorno que rodea a la cámara gracias a un procedimiento que analiza y extrae información de las imágenes capturadas por dicha cámara.

Visual SLAM es un tipo específico de sistema SLAM que se basa en algoritmos de visión 3D para realizar tareas de autolocalización y mapeo cuando ni la localización de la posición de la cámara ni el entorno son conocidos.

La mayoría de los sistemas SLAM funcionan estimando la posición de un conjunto de puntos en varias imágenes sucesivas y así triangular la posición 3D, mientras que simultáneamente utiliza esta información para dar un posición aproximada de la cámara. Básicamente, el objetivo de estos sistemas es hacer un mapa de sus alrededores de su localización para así poder realizar tareas de navegación por el entorno.

Esto es posible con una sola cámara. Mientras existan un número suficiente de puntos que puedan ser seguidos a través de varios fotogramas, tanto la orientación del sensor de orientación como la estructura física del entorno pueden ser rápidamente estimados.

Todos los sistemas Visual SLAM están constantemente trabajando para minimizar el error de reprojeción, o la diferencia entre los puntos reales y los puntos proyectados, para ello suele utilizar una solución llamada *Bundle Adjustment*.

1.2.1. Aplicaciones en Visual SLAM

Visual SLAM es todavía una tecnología emergente, pero con muchísimo potencial. Es una parte importante en aplicaciones de realidad aumentada, ya que esta tecnología es capaz de mapear el mundo físico con gran exactitud. También se utilizará en una gran variedad de robots, por ejemplo, los robots que se envían a Marte utilizan sistemas de Visual SLAM para navegar de forma autónoma. De la misma manera drones y robots en agricultura pueden utilizar esta misma tecnología para moverse por campos de cultivo, incluso los vehículos autónomos podrían utilizar sistemas Visual SLAM para mapear y entender el mundo a su alrededor. Otro gran potencial del Visual SLAM es que permite reemplazar el GPS en ciertos entornos, ya que el GPS no es muy útil en interiores y en grandes ciudades, donde el GPS tiene una exactitud de metros mientras que con Visual SLAM no existirían estos problemas y además tiene mayor exactitud. A continuación se expondrán varios ejemplos de aplicaciones, desde teléfonos móviles hasta robots aspiradora.

1. Proyecto Tango:

El proyecto Tango es un proyecto colaborativo que trata de equipar a los smartphones y Tablets con sistema operativo Android la capacidad de medir la profundidad a la que se encuentra cada píxel de las imágenes capturadas por la cámara. Para ello los dispositivos compatibles con Tango dispondrán de 2 cámaras, una cámara RGB y otra que captura la profundidad, así el smartphone es capaz de construir un mapa en 3D del entorno (Figura 1.3(c)). Las posibilidades que ofrecerán este tipo de dispositivos serán muy variadas, desde medir las dimensiones de la habitación, hasta lo más útil como guiar a personas con discapacidades visuales en el interior de edificios. Pero también tendrá utilidades para el entretenimiento como convertir una habitación en el escenario de un juego mediante realidad aumentada.

Al ser una tecnología nueva aún no hay un elevado número de dispositivos que lo



Figura 1.3: El primer smartphone compatible con Tango de Lenovo(a). El primer Smartphone compatible con Tango y DayDream de ASUS (b). Esquema de prototipo de smartphone Tango (c).

soporten. De momento existen 2 móviles compatibles con Tango¹, el Lenovo Phab 2 pro (Figura 1.3(a)) y el Asus Zenfone AR (Figura 1.3(b)). En el caso del Zenfone AR estará equipado con 3 cámaras traseras, una para seguir objetos (motion tracking), otra para detectar profundidad y otra de alta resolución de 23 MP. Con estas 3 cámaras el smartphone podrá crear una modelo tridimensional del entorno y seguir su movimiento. La cámara de localización permitirá al ZenFone conocer su posición 3D en todo momento mientras se mueve por el entorno. La cámara de profundidad está equipada con un proyector de Infrarrojos que le permite medir distancias hasta los objetos en el mundo real.

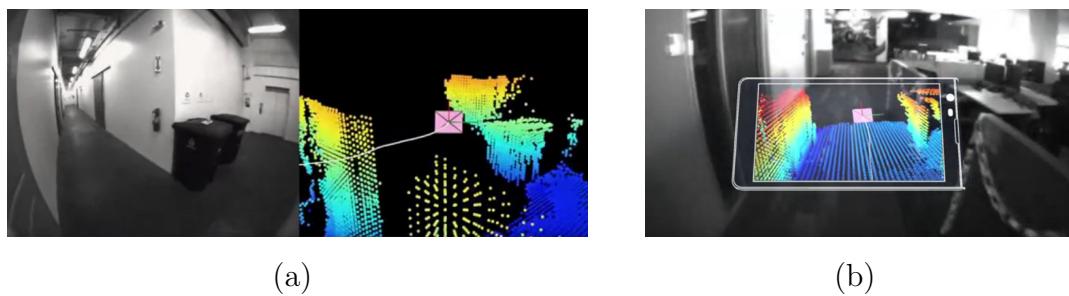


Figura 1.4: Generación de mapa 1 (d). Generación de mapa 2 (e)

2. Magic Plan:

Magic Plan es una aplicación que permite de forma interactiva obtener planos de

¹<https://get.google.com/tango/>

habitaciones o del interior de un edificio, utilizando para ello la cámara de nuestra tablet o smartphone, sólo es necesario sacar fotos. Esta aplicación es gratuita, aunque si se desea obtener el plano en formato digital (pdf, jpg, csv y otros) será necesario pagar una pequeña cantidad de dinero. Es muy sencilla de utilizar y en cuestión de minutos se obtiene un plano fiable sin necesidad de medir, dibujar, mover muebles y sin necesidad de ser un experto. La aplicación utiliza técnicas de Visual SLAM y se apoya también en la información de los giroscopios de los dispositivos. Es compatible con Android y dispositivos Apple.

En el caso de Android, actualmente la última versión es compatible con el sistema Tango, por tanto el procedimiento de captura es mucho más sencillo, robusto y preciso ya que permite detectar con mayor exactitud todas las paredes de la habitación, visualizarlas en 3D y aplicar realidad aumentada.

3. Pix4D:

Pix4D² es un software especializado en fotogrametría. Permite la posibilidad de generar mapas 2D y 3D desde fotografías. Las imágenes pueden ser transmitidas vía wireless a Pix4DDim para procesarlas y convertirlas a mapas 2D y 3D. Posteriormente esta información será accesible desde la nube para poder analizarlas y compartirlas. Pix4D permite crear mapas con exactitud a partir de fotografías de interiores, también tiene aplicaciones en minería para medir superficies y volúmenes (Figura 1.5(a)) de minas a cielo abierto, incluso se utiliza con finalidades forenses para recrear en 3D escenarios de accidentes, que posteriormente pueden ser analizadas con todo detalle. También tiene aplicaciones en la agricultura para obtener mapas de cosechas utilizando la información que proporcionan las cámaras especiales como la Parrot Sequoia (Figura 1.5(b)). Con la aplicación Pix4DCapture podremos controlar un dron desde nuestro smartphone para que genere un mapa. El dron puede volar de forma autónoma siguiendo algunas de las trayectoria de vuelo que trae por defecto el producto o también puede generar el mapa mientras lo teledirigimos.

²<https://pix4d.com/>

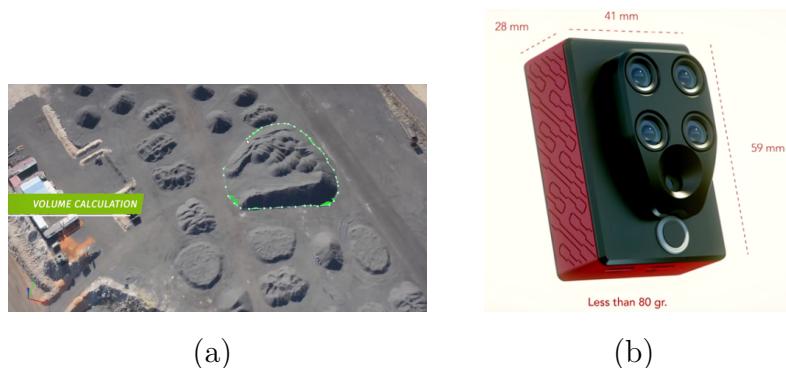


Figura 1.5: Pix4D cálculo de volumen(a). Cámara multiespectral Parrot Sequoia (b)

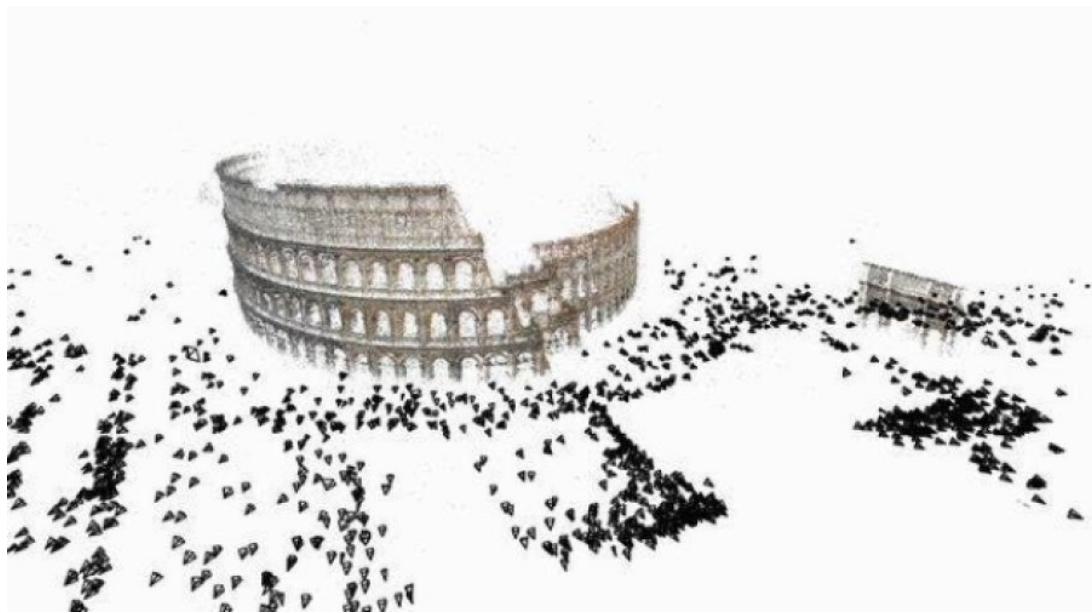
4. Photo Tourism:

PhotoTourism o Photo Synth es un software inicialmente creado por la universidad de Washington en colaboración con Microsoft. Es un sistema que toma grupos de conjuntos de fotografías disponibles online sobre un lugar en concreto, normalmente sobre un monumento turístico mundialmente conocido (como NotreDame, el Coliseo (Figura 1.6(a)), La Fontana de Trevi) y es capaz de reconstruir puntos 3D de los monumentos y también calcular o estimar la posición de la cámara desde donde se tomaron las fotografías. Proporciona una nueva forma de navegar a través de fotografías de un destino turístico y una nueva forma de hacer visitas virtuales a monumentos. Este sistema utiliza la técnica de *Structure From Motion* SFM. SFM encuentra coincidencias de puntos característicos entre distintas fotografías de un mismo lugar y que han sido tomadas desde distintos puntos de vista y así es capaz de calcular la localización 3D de dichos puntos característicos y también la localización 3D desde donde se tomaron las fotografías. A diferencia de Visual SLAM, el procesamiento de estas fotografía es offline, sin necesidad de tiempo real, por lo que pueden ser ejecutadas desde un PC que por lo general tiene una capacidad de computación mucho mayor que una tablet o teléfono móvil.

5. Canvas y el sensor Structure:

Canvas³ es una herramienta de escaneo 3D enfocada a profesionales de la construcción o incluso aficionados al bricolaje en casa. La aplicación se ayuda del sensor de profundidad Structure. Este sensor se acopla en la parte trasera de un Ipad. Canvas permite obtener los planos en 3D de cualquier habitación de una manera fácil y sencilla, simplemente tendremos que pasear el Ipad equipado con el sensor Structure

³<https://canvas.io/>



(a)

Figura 1.6: Recreación del Coliseo de Roma generado con Photo Tourism.

⁴ alrededor de la habitación y podremos ver como el mapa 3D comienza a generarse en tiempo real. El sensor (Figura 1.7(a)) toma miles de medidas de profundidad que utilizará para generar el plano tridimensional. Los planos son almacenados en el Ipad y pueden ser consultados de manera interactiva posteriormente. Además permite que los planos generados sean convertidos a ficheros CAD.



(a)

Figura 1.7: El sensor de profundidad Structure para Ipad.

6. Robot Aspirador:

Recientemente ha entrado en los hogares el uso de Visual SLAM

⁴<https://structure.io/>

gracias a los últimos modelos de aspiradora equipados con cámaras. Estos aspiradores robotizados disponen de cámaras que le permiten obtener un mapa de la habitación o planta del edificio y gracias a este mapa son capaces de aspirar toda la superficie del suelo de la habitación de manera eficiente, sin dejar ninguna zona de la planta sin limpiar. Además están equipados con sensores de proximidad, que les permiten esquivar obstáculos y aunque tengan que modificar su recorrido momentáneamente son capaces de seguir limpiando ya que pueden utilizar el mapa para continuar su ruta. Entre los distintos aspiradores estarían: Aspirador Dyson 360 Eye (Figura 1.8(a))⁵, Aspirador Roomba 966 (Figura 1.8(b))⁶, Aspirador LG-Hombot ⁷

Tanto el modelo de Dyson como Roomba utilizan una cámara de 360 grados, en cambio el modelo de LG utiliza una doble cámara, y es capaz de aspirar la casa incluso en la oscuridad.

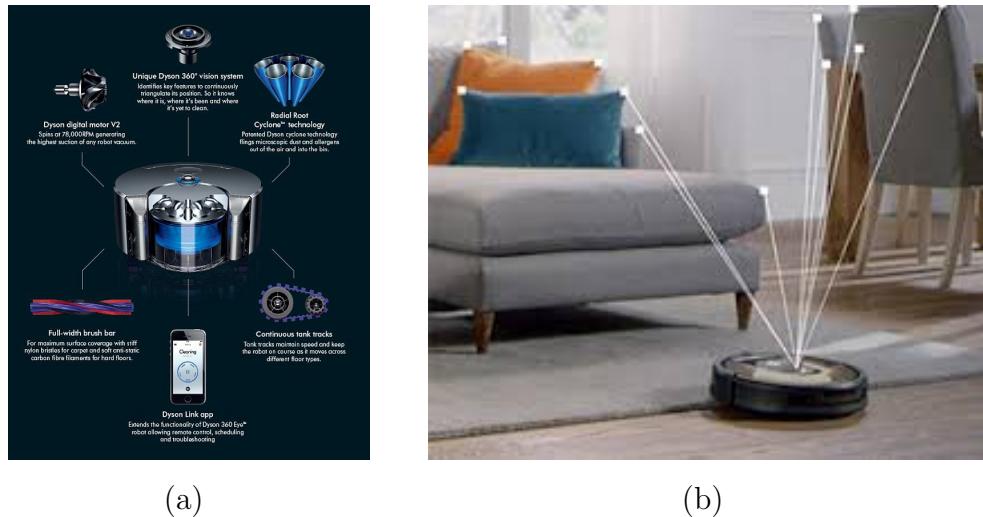


Figura 1.8: Robot Dyson 360 Eye (a) Robot Roomba 966 (b)

7. Drones:

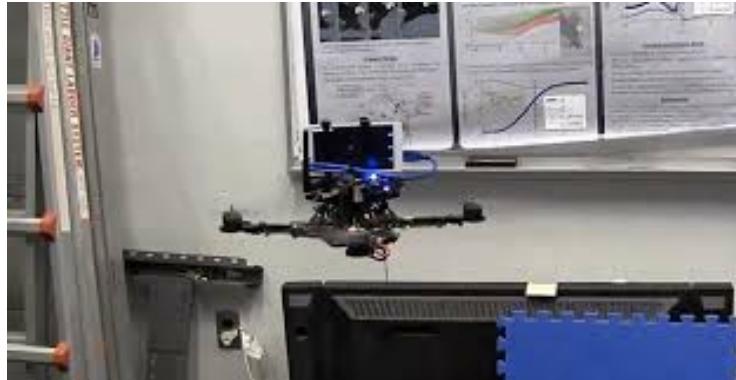
Por último no podemos olvidar los drones, robots voladores equipados con cámara que también pueden obtener mapas de su entorno con Visual SLAM. Existen también proyectos para equipar a drones con dispositivos compatibles con Tango para que sean capaces de obtener mapas de interiores con mayor precisión, robustez y velocidad ⁸.

⁵<http://www.dyson.com>

⁶<http://www.irobot.es/robots-domesticos/aspiracion>

⁷<http://www.lg.com/es/aspiradoras/lg-VR64702LVMT>

⁸<http://spectrum.ieee.org/automaton/robotics/drones/autonomous-quadrotor-flight-based-on-google-project-tango>



(a)

Figura 1.9: Dron equipado con dispositivo compatible con Tango

1.2.2. Conceptos

Conviene explicar una serie de conceptos relacionados con Visual SLAM ya que aparecerán más adelante cuando expliquemos en profundidad los algoritmos más importantes que existen hoy en día para localización visual.

Calidad: La calidad del algoritmo de Visual SLAM dependerá de tres factores. La eficiencia temporal, la precisión espacial en la estimación de la posición y la robustez del algoritmo.

Eficiencia: Mediremos la eficiencia como el tiempo de ejecución de cada iteración del algoritmo. Los algoritmos deberán ser capaces de procesar al menos 30 fotogramas por segundo

Precisión: El error lineal y error angular entre la posición estimada y la posición lineal determinará la precisión. Mediremos el error lineal como la distancia euclídea entre las dos posiciones, mientras que el error angular vendrá determinado por la diferencia entre las 2 orientaciones.

Robustez: Diremos que el algoritmo de localización es robusto siempre que pueda seguir funcionando con normalidad tras encontrarse con una situación imprevista (como una oclusión, secuestros, movimiento de objetos en la escena, mala calidad de imagen, etc)

Oclusiones: Cuando la cámara del robot esté tapada parcial o totalmente se producirá una oclusión, por tanto no se podrá extraer información en la región de la imagen ocluida. Los algoritmos deben estar preparados para cuando existan oclusiones.

Secuestros: Se producirá un secuestro cuando la cámara o robot sea movida

deliberadamente por un tercero, de tal forma que los cálculos de la posición anterior ya no sean válidos. Los algoritmos deberían detectar secuestros e intentar localizarse desde su nueva posición.

Localización Absoluta: se produce cuando tenemos un mapa conocido, si estimamos la posición del robot dentro del mapa en coordenadas respecto del origen de coordenadas de ese mapa.

Localización Incremental: En entornos con mapa desconocido, la localización del robot se establecerá de forma incremental respecto de posiciones previas pasadas (por ejemplo en el instante anterior), lo que dará lugar a un error en la localización incremental que aumentará con el tiempo.

Cierre de bucle: Se produce cuando el robot vuelva a pasar por una zona del mundo que ya haya visitado anteriormente. Si se vuelve a pasar por el punto de origen se puede determinar el error que se ha producido comparando la posición real en el origen con la estimada por el algoritmo de localización.

Relocalización: Si tras un secuestro, el robot consigue recuperarse y estima correctamente la posición absoluta del robot dentro del mapa.

1.2.3. Problemas abiertos en Visual SLAM

Actualmente las técnicas de Visual SLAM presentan algunos problemas o inconvenientes que todavía son difíciles de sortear en la práctica. En esta sección presentaremos algunos de ellos:

1. Inicialización del Mapa:

Si en Visual SLAM queremos conseguir una estimación lo más exacta posible de la posición de la cámara es necesario contar con una buena inicialización del Mapa. Se debe contar con un sistema de coordenadas globales definido, y se tomarán puntos de referencia del entorno como puntos iniciales del mapa en el sistema global de coordenadas. Utilizando este método podemos inicializar Visual SLAM en un sistema de coordenadas global en la tierra. La transformación de estos puntos iniciales al sistema de referencia global se realizará mediante homografía.

Objetos de referencia como objetos 3D también se han utilizado para obtener un sistema global de coordenadas, posiciones iniciales de la cámara son estimadas gracias al seguimiento de objetos de referencia. En Mono SLAM (cap 3,sec 1), por ejemplo,

se utilizan al menos 4 puntos 3D (que se corresponden con las esquinas de un folio) como objetos de referencia.

2. Ambigüedad en la escala:

En algunas aplicaciones con Visual SLAM se necesita información de escala absoluta. Para obtener una referencia de escala absoluta se pueden utilizar zonas de la anatomía del usuario, como la cara, su mano o el propio cuerpo u otros objetos de tamaño conocido. En todos estos métodos se asume que entre personas la diferencia de tamaño es mínima para dichas partes del cuerpo. Se han dado otras aproximaciones como utilizar algunos de los sensores con los que ya están equipados la mayoría smartphones tales como acelerómetros, giroscopios y sensores magnéticos. Para eliminar el ruido de estos sensores se utiliza una técnica de filtro de dominio de frecuencia.

3. Dificultad para operar en entornos con pocas texturas:

Visual SLAM utiliza el emparejamiento de píxeles o puntos característicos entre varios frames consecutivos. El emparejamiento suele fijarse en esquinas, bordes o puntos distintivos que fácilmente podrán localizarse entre frames. Pero cuando en el entorno hay pocas texturas o presenta una alta monotonía de texturas, el emparejamiento es difícil de realizar ya que un punto en un fotograma podría corresponder con N puntos en el siguiente fotograma y por tanto se dispararía el error de posición. Quizá este sea uno de los problemas más difíciles de solucionar con Visual SLAM [Takafumi Taketomi, 2017].

1.3. Estructura del documento

El presente documento está dividido en 6 secciones y trata de describir este TFM, cuya temática es la creación de una herramienta de evaluación de algoritmos de Visual SLAM. El primer capítulo es una introducción a la Visión Artificial y Visual SLAM, donde se explica en que consiste Visual SLAM, sus aplicaciones y cuales son las principales dificultades que debemos solventar a la hora de implementar un algoritmo de Visual SLAM. El capítulo 2 será una descripción detallada de los principales objetivos del TFM. El capítulo 3 estará dedicado al Estado del Arte de los algoritmos Visual SLAM, y también se expondrán las herramientas más conocidas hoy en día para la evaluación de algoritmos Visual SLAM. En el capítulo 4 hablaremos de SLAMTestBed, una herramienta software creada precisamente en este TFM para evaluación de algoritmos Visual SLAM. En el capítulo 5 se expondrán los distintos

experimentos y tests realizados para evaluar la herramienta SLAM TestBed. Y por último, el capítulo 6, donde se muestran las conclusiones del TFM.

Capítulo 2

Objetivos

En el siguiente capítulo se detallarán las metas concretas que se pretenden alcanzar en este Trabajo Fin de Máster.

2.1. Objetivos

Desde el nacimiento de Visual SLAM, se están desarrollando algoritmos que permitan a los robots estimar su posición, o con una cámara localizar una imagen en tiempo real en 3D para aplicaciones de realidad aumentada. Estos algoritmos son complejos y están compuestos de múltiples etapas, a menudo un ligero cambio en alguna de estas etapas puede hacer que los resultados del algoritmo mejoren significativamente o por el contrario empeoren. Sería muy útil y conveniente contar con una herramienta que permitiese analizar o estimar la bondad de los nuevos algoritmos visual SLAM.

El objetivo principal de este proyecto es presentar una herramienta que permitan comparar cuantitativamente el rendimiento de los algoritmos Visual SLAM y la calidad de sus estimaciones. Esta herramienta permitirá comparar el rendimiento de los diferentes algoritmos de Visual SLAM así como estudiar más fácilmente que modificaciones o mejoras se podrían aplicar en los algoritmos ya existentes.

En este TFM nos hemos centrado en medir la exactitud de las estimaciones de posición sin tener en cuenta la generación de mapas, es decir se ha puesto toda la atención en comparar las mediciones de *tracking* dejando el *mapping* para futuros proyectos.

Entenderemos que un algoritmo A es mejor que otro B cuando el algoritmo A sea capaz de estimar la posición con mayor exactitud que otro algoritmo B en el menor tiempo posible y de manera robusta. La herramienta a desarrollar medirá la precisión de las estimaciones

de posición y la agilidad entendiéndose esta como el tiempo de proceso dedicado a realizar dichas estimaciones de posición.

Típicamente esta herramienta permitirá comparar dos secuencias de posiciones 3D, la estimada por un algoritmo de Visual SLAM y la secuencia de posiciones verdaderas. La diferencia entre ambas mide el error del algoritmo SLAM a la hora de estimar correctamente las posiciones de la cámara. Este objetivo principal lo hemos articulado en los siguientes subobjetivos:

1. **Conseguir un Registrador Espacial:** Este registrador espacial debe permitir registrar 2 secuencias de puntos 3D en el espacio, estimando rotación, traslación y escala para poder realizar el registro espacial entre las 2 nubes de puntos.
2. **Registrador Temporal:** Que permita sincronizar las 2 secuencias de puntos 3D. Para ello deberemos ajustar los 2 secuencias a la misma frecuencia de muestreo, (utilizaremos interpolación) y también calcular el offset temporal entre ambas. El offset temporal sería la diferencia de tiempo entre 2 secuencias.
3. **La herramienta de medición se validará experimentalmente:** Para comprobar que funciona correctamente realizaremos experimentos donde comprobaremos el error estimado entre la verdad absoluta (groundtruth) y una secuencia transformada de la que se conoce exactamente el registro temporal y espacial. Se creará un Módulo de Transformación que permitirá realizar tanto alteraciones temporales como espaciales a una secuencia dada generando como resultado una nueva secuencia transformada.

2.2. Requisitos

La solución que se desarrolle para alcanzar los objetivos deberá satisfacer además estos requisitos:

R1. La herramienta debe restringirse a la comparación de algoritmos Visual SLAM que utilizan como sensor de visión una cámara RGB. Así quedan descartados los algoritmos que emplean cámaras RGBD.

R2. Se restringirá solo a los algoritmos Visual SLAM que empleen *una sola* cámara.

R3. La herramienta debe ser aplicable a nuevos algoritmos de Visual SLAM.

R4. Facilidad de uso.

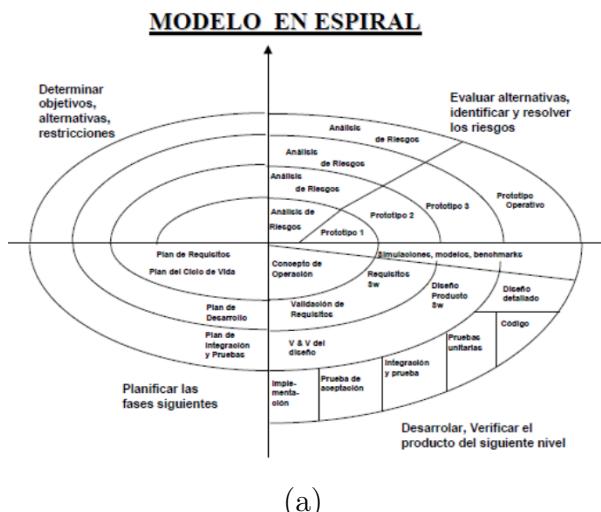
R5. Debe ofrecer una única métrica cuantitativa para la comparación entre varios algoritmos de Visual SLAM.

R6. Debe ofrecer resultados fácilmente legibles y reconocibles para el usuario.

R7. El software generado debe ser abierto y estar públicamente disponible.

2.3. Metodología y plan de trabajo

En cuanto a la metodología utilizada para desarrollar este TFM, se ha seguido el modelo de ciclo de vida en espiral. Una de las primeras lecciones que se aprenden de las metodologías ágiles es que la metodología debe adaptarse al proyecto y no al revés. Dicha premisa cobra más importancia en un proyecto realizado por una sola persona.



(a)

Figura 2.1: Inicialización de Mono SLAM con 4 puntos conocidos.

El modelo en espiral es un modelo iterativo, donde cada ciclo viene a representar una fase del proyecto software. Podemos diferenciar 4 fases dentro de cada ciclo del modelo en espiral

-Determinar los Objetivos: Determinaremos que metas se deben conseguir en cada iteración, teniendo en cuenta los objetivos del proyecto.

-Evaluación de alternativas: Evaluar las distintas alternativas para alcanzar las metas que se han establecido en la fase anterior, utilizando distintos puntos de vista.

-Desarrollo y Evaluación: Una vez seleccionada la mejor alternativa, se diseña y desarrolla el producto y por último se realizarán pruebas para testear su funcionamiento.

-Planificación: Teniendo en cuenta los resultados de las pruebas realizadas, se planificará la siguiente iteración revisando posibles errores cometidos en iteraciones anteriores y se comienza con una nuevo ciclo en espiral.

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 de Visual SLAM y detectar puntos críticos. Segundo se ha ido acotando hasta el primer prototipo del desarrollo principal, el cual ha sido revisado y validado en varias iteraciones. Este prototipo 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 los tutores mediante tres herramientas: 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:¹

Así mismo, el código fuente desarrollado puede encontrarse en² y la memoria en³

¹<https://jderobot.org/Elias-tfm>

²<https://github.com/JdeRobot/slam-Testbed>

³<https://github.com/RoboticsURJC-students/2017-tfm-elias-barcia>

Capítulo 3

Estado del Arte

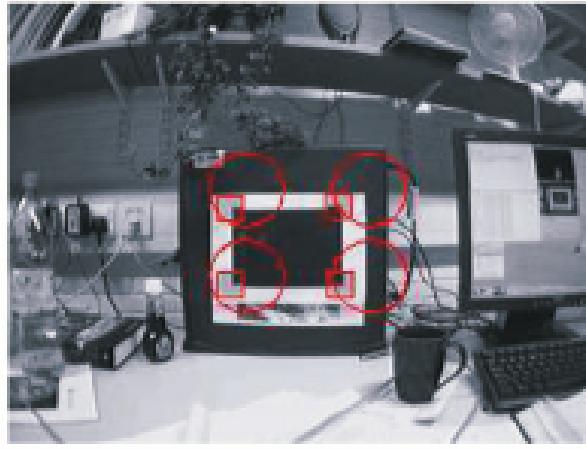
En este capítulo explicaremos varios de los algoritmos SLAM más significativos, como MonoSLAM, PTAM, ORB-SLAM, DSO/LDSO. También se describirán brevemente las distintas herramientas que existen en la actualidad para comparar las estimaciones de los algoritmos SLAM, comenzaremos por TUM, seguido de *rgb trajectory evaluation*, describiremos la herramienta SLAMBENCH y por último daremos algunos detalles sobre *The Kitti Vision Benchmark Suite*.

3.1. Algoritmos de Visual SLAM

En la actualidad existen múltiples algoritmos de Visual SLAM, en este apartado describiremos 4 de los algoritmos mas importantes en Visual SLAM, que destacan sobre los demás por su influencia en el estado del arte. Los dos primeros, MonoSLAM Y PTAM, fueron los primeros algoritmos que permitieron utilizar este tipo de técnicas en tiempo real, mientras que los dos últimos, ORB-SLAM y LDSO son los más utilizados en la actualidad por su precisión, eficiencia y robustez.

3.1.1. Algoritmo MonoSLAM

El algoritmo de MonoSLAM (*Monocular SLAM*) [Davison *et al.*, 2007] utiliza solamente una cámara RGB para la localización y mapeo de entornos desconocidos. Fue desarrollado entre los años 2002 y 2007 por Andrew Robinson. Para estimar la posición de la cámara utiliza un Filtro Extendido de Kalman (EKF) y la posición de una serie de puntos 3D. Este método requiere de una inicialización con al menos 4 puntos 3D conocidos que utilizará para calcular la posición de la cámara y la generación de nuevos puntos para el mapa.



(a)

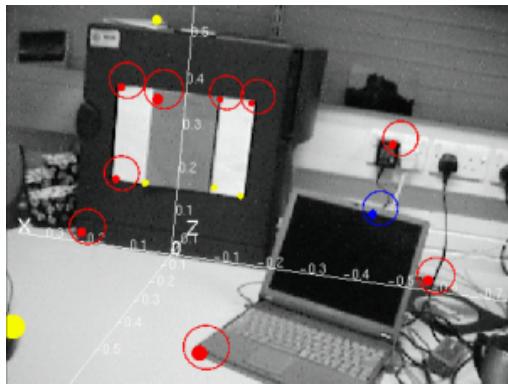
Figura 3.1: Inicialización de MonoSLAM con 4 puntos conocidos.

El EKF, tiene un vector de estado compuesto de posición, orientación y velocidad de la cámara además de las coordenadas 3D de los puntos conocidos en un cierto momento, esto implica que el vector de estado irá aumentando de tamaño a medida que vayamos descubriendo nuevos puntos 3D. El modelo de observación estará compuesto de las proyecciones de cada uno de los puntos 3D en el plano imagen.

El uso de un EKF es apropiado ya que se realizan iteraciones cada pocos milisegundos, y en intervalos de tiempo tan pequeños, el sistema puede aproximarse a un sistema lineal, mejorando la estimación cuanto mayor sea la frecuencia de muestreo. En cada iteración se hace una detección de puntos de interés (obtenidos mediante FAST [Trajkovic and Hedley, 1998]) en la imagen actual de entrada, y obtendremos una serie de puntos que serán candidatos a formar parte del vector de estado con los puntos que queremos seguir. Estos candidatos deberán ser filtrados, pues alguno puede ser un falso positivo. Se utilizará una función de divergencia ZMSSD (*Zero Mean Sum of Squared Differences*) entre parches para determinar si el candidato es aceptable o no. Al utilizar sólo parches de unos pocos píxeles alrededor del candidato, el algoritmo optimiza el cómputo, ya que no requiere procesar toda la imagen.

Aún así, es posible que se acepten puntos candidatos que no sean apropiados. Para tratar de eliminar estos falsos positivos, [Civera *et al.*, 2010] propuso una alternativa conocida como 1-Point RANSAC que descarta los puntos cuya posición en la imagen no es coherente con el resto de puntos. MonoSLAM sólo es recomendable para mapas con pocos puntos. Algunas de las desventajas de MonoSLAM son gran sensibilidad a movimientos bruscos y por tanto difícilmente podrá recuperarse de un **secuestro** y si la hipótesis de partida no

es correcta el filtro podría desestabilizarse y no llegar nunca a aproximar razonablemente el vector de estado.



(a)

Figura 3.2: Ejemplo de puntos característicos tomados con MonoSLAM.

3.1.2. Algoritmo PTAM

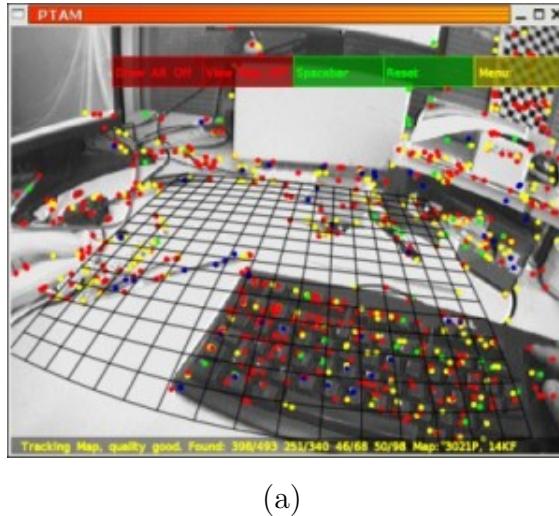
PTAM son las siglas de *Parallel Tracking and Mapping*. Este algoritmo fue creado en 2007 por George Klein [Klein and Murray, 2007]. La diferencia principal con MonoSLAM es que utiliza 2 hilos, uno para calcular el posicionamiento de la cámara (*Tracking*) y el segundo para la generación del mapa (*Mapping*). Esta separación en dos hilos de ejecución se debe a que el *Tracking* necesita ser calculado en tiempo real para obtener una localización precisa, mientras que el *Mapping* puede demorarse más tiempo sin perjudicar a la localización de la cámara.

Otro de los elementos clave de este algoritmo son los *Keyframes* o fotogramas clave. Se genera un nuevo *Keyframe* a medida que la cámara se va desplazando, utilizándose tanto para la localización como para ir generando el mapa de puntos.

Este algoritmo es recomendable para mapas con elevado número de puntos, siendo además capaz de recuperarse fácilmente de un secuestro. Al igual que MonoSLAM, extrae los puntos de interés mediante extracción de características. Para extraer esos puntos se realizará una subdivisión de la imagen a distintas resoluciones, normalmente 4 niveles,(lo que se conoce como pirámide de la imagen) y se pasará un filtro FAST sobre esta pirámide para detectar los puntos más característicos de la imagen. Cada *Keyframe* que se genera, contiene la imagen captada junto su pirámide y sus puntos de interés detectados. Cuando añadimos un *Keyframe*, se intenta localizar en este *Keyframe* los puntos que ya se encuentran en el mapa, en caso de no localizarlos se añaden nuevos puntos al

mapa. Mientras no se añadan *Keyframes*, se intentará mejorar el mapa con los *Keyframes* disponibles optimizando el mapa con *Bundle Adjustment* [Konolige and Agrawal, 2008].

En la actualidad, este algoritmo sólo es adecuado para entornos reducidos, pudiendo utilizarse para pequeñas aplicaciones de realidad aumentada.



(a)

Figura 3.3: Nube de puntos característicos tomados con PTAM.

3.1.3. Algoritmo ORB-SLAM

La característica principal de este algoritmo es la utilización de descriptores ORB para emparejar los puntos en las imágenes. La ventaja principal de estos descriptores es que son más fiables que los parches tradicionales y por tanto permiten obtener mapas robustos y precisos tanto en escenarios de grandes dimensiones como en zonas pequeñas, sin embargo, para su funcionamiento en tiempo real, requiere la utilización de ordenadores con alta capacidad de proceso [Mur-Artal *et al.*, 2015]. Este algoritmo puede ser utilizado con una cámara, con dos cámaras en estéreo, e incluso con cámaras de profundidad RGBD. ORB-SLAM permite tanto relocalizarse como realizar cierres de bucle, para lo que usa un modelo de bolsa de palabras[Gálvez-López and Tardos, 2012]. Otra de las novedades de este algoritmo es que utiliza 3 hilos, el primero para *Tracking*, el segundo para *Mapping* y un tercero para detectar cierres de bucle.

En el hilo de *Tracking*, se trata de calcular la posición actual a partir de los emparejamientos encontrados de los puntos 3D en el fotograma anterior, para ello utilizará los descriptores ORB. En caso de perdida, el robot podrá relocalizarse gracias a un modelo de bolsa de palabras que le permitirá encontrar *Keyframes* candidatos que concuerden con

la observación actual (Figura 3.4(a)).

En el hilo de *Mapping*, se inicializarán 2 mapas, uno por homografía y el segundo mediante una matriz fundamental. Los 2 mapas recibirán una puntuación y se elegirá como candidato para inicializar el mapa aquel que obtenga mayor puntuación. Cuando ya se dispone del mapa inicial, se procesan los *Keyframes* creando nuevos puntos 3D y se optimiza localmente el mapa mediante *Bundle Adjustment*. A su vez se genera un grafo donde cada *Keyframe* se conecta con otros *Keyframes* de su alrededor siempre y cuando estos *Keyframe* tengan suficientes puntos 3D en común. Este grafo permite la eliminación de *Keyframe* redundantes (Figura 3.4(b)).

En el hilo de *Looping*, se comprobará si se ha producido un cierre de bucle. Utilizando el grafo de *Keyframe* conectados y el modelo de bolsa de palabras se intenta encontrar *Keyframe* candidatos que tengan una apariencia similar a la imagen actual.

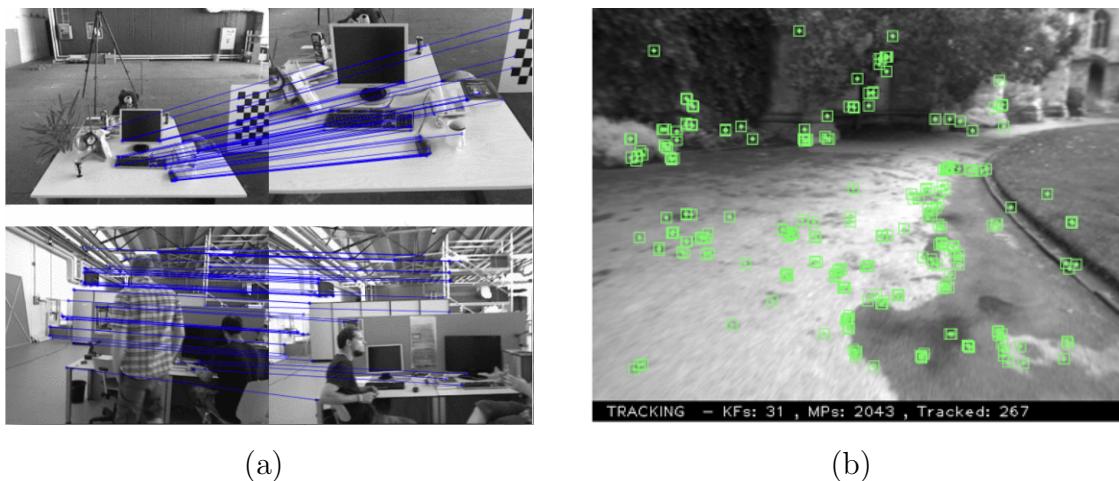


Figura 3.4: Localización de puntos característicos en 2 imágenes con ORB

3.1.4. DSO y LDSO

DSO: Direct Sparse Model. Al contrario de los métodos anteriores, basados en puntos característicos, está basado en optimizaciones continuas del error fotométrico sobre una ventana de fotogramas recientes[Engel *et al.*, 2016].

El inicio del Tracking, cuando se crea un nuevo *Keyframe*, todos los puntos activos son proyectados en el y ligeramente dilatados, creando así un mapa de profundidad semi denso. Nuevos fotogramas son creados con respecto a este fotograma utilizando alineamiento directo de 2 fotogramas, una pirámide multi escala y un modelo de movimiento constante a

inicializar. La creación de *Keyframes* es similar a ORB-SLAM, existiendo 3 criterios para determinar cuando se necesita un nuevo *Keyframe*.

1. Se creará un nuevo *Keyframe* (Figura 3.5(a)) cuando la imagen de entrada cambie notablemente con respecto al último *Keyframe*, esto se medirá con la diferencias de medias al cuadrado entre los píxeles.
2. La traslación de la cámara causa oclusiones y des-oclusiones, lo cual indica que se deben generar nuevos *Keyframes*
3. Si el tiempo de exposición de la cámara cambia significativamente, se deberá tomar un nuevo *Keyframe*. Esto se mide por el factor de brillo relativo entre 2 fotogramas.

En cuanto al rechazo de *Keyframes*, sigue la siguiente estrategia. Sean $I_1 \dots I_n$ un conjunto de *Keyframes* activos, siendo I_1 el más nuevo y I_n el más antiguo

1. Siempre se mantendrán los dos últimos *Keyframes* (I_1 e I_2)
2. Frames con menos del 5 % de sus puntos visibles en I_1 son descartados.
3. Si mas de N fotogramas están activos, se descartan (exceptuando I_1 e I_2) aquel que maximice un marcador de distancia $d(i,j)$ donde $d(i,j)$ es la distancia Euclídea entre *Keyframes* I_1 e I_j

Sobre el tratamiento de los puntos, siempre se tratará de mantener un numero fijo de puntos activos repartidos de forma uniforme entre el espacio y los fotogramas activos. En un primer paso, se identifican N_p puntos candidatos en cada nuevo *Keyframe*. Los puntos candidatos no son inmediatamente sumados a la optimización, sino que son localizados individualmente en sucesivos fotogramas generando una primera estimación del valor de profundidad que servirá como inicialización.

En cuanto a la selección de puntos candidatos, se intentará seleccionar aquellos puntos que están bien distribuidos en la imagen y tienen un valor elevado de gradiente con respecto a sus alrededores. Para obtener una distribución uniforme de puntos sobre la imagen, esta se divide en bloques de dxd , de cada bloque se elegirá el píxel con el mayor gradiente siempre y cuando supere un umbral, de lo contrario no se selecciona el píxel de ese bloque. Los puntos candidatos son localizados en los siguientes fotogramas utilizando una búsqueda sobre la línea epipolar minimizando el error fotométrico. Una vez hallamos encontrado las coincidencias preparamos un valor de profundidad y la varianza asociada que se utilizará para restringir el intervalo de búsqueda en los fotogramas siguientes.

Por último, cuando un conjunto de puntos antiguos son marginados, nuevos puntos candidatos son activados para remplazarlos, siempre intentando mantener una distribución uniforme de puntos por toda la imagen¹.

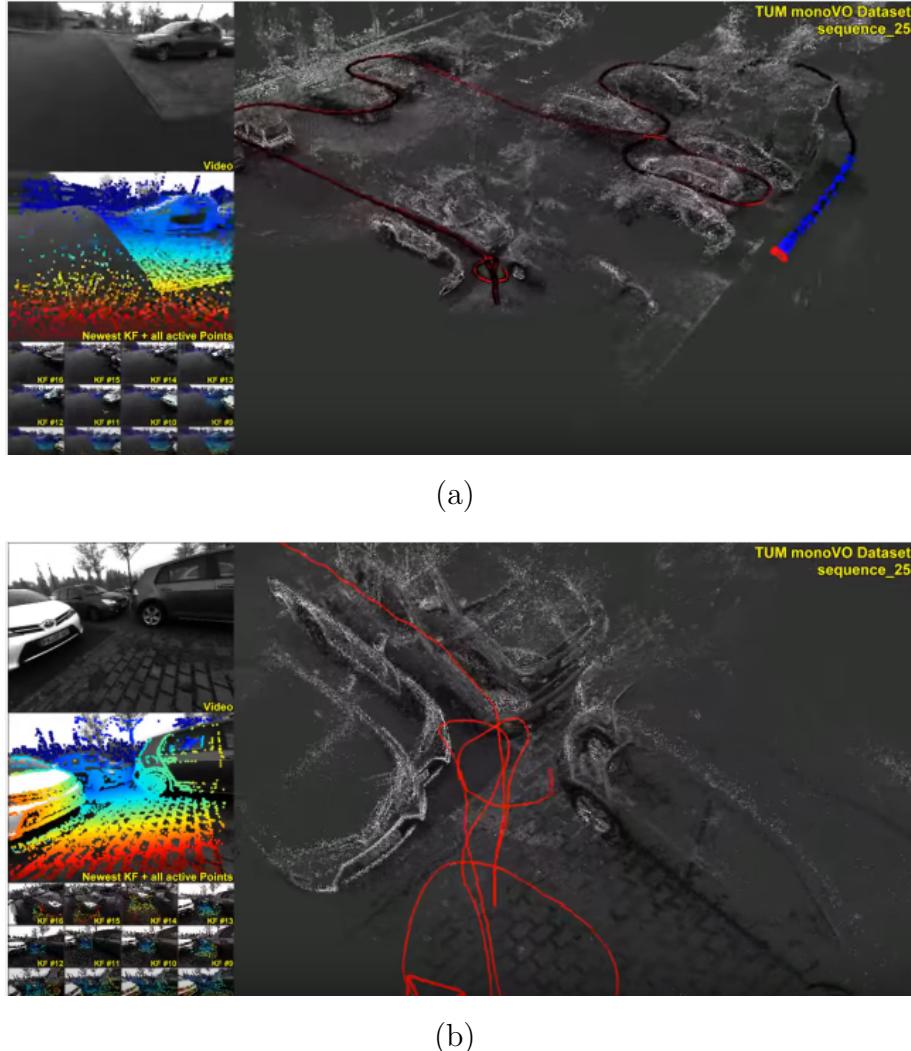


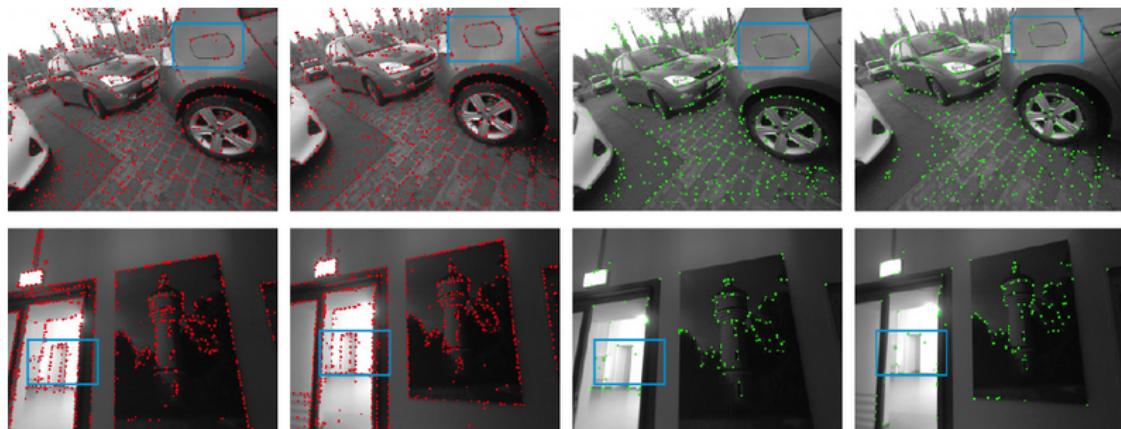
Figura 3.5: Mapa generado con DSO (a) Ligero error en la posición al volver al punto de partida (b).

LDSO: Direct Sparse Odometry with Loop Closure. Este método es una extensión del algoritmo DSO, incorporando detección de cierre de bucle y optimización de posición y mapeo. Al ser un método directo, DSO puede utilizar cualquier píxel de la imagen con suficiente gradiente de intensidad, lo cual lo hace más robusto incluso en áreas donde apenas se pueden obtener puntos característicos. LDSO mantiene esta robustez, mientras que al mismo tiempo asegura la repetibilidad sobre alguno de esos puntos prestando más

¹<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7898369>

atención sobre esquinas características en el proceso de *Tracking*. Estas repetibilidad de puntos característicos permite detectar de forma fiable los candidatos de cierres de bucle utilizando la técnica basada en características de *bag-of-words* de forma similar a ORB-SLAM. Los candidatos a cierre de bucle son verificados geométricamente minimizando en conjunto errores geométricos 2D y 3D.

Para la selección de puntos de características repetibles, LDSO utiliza un grid dinámico para detectar suficientes puntos incluso en entornos con pocas texturas. Se toma un número determinado de puntos, de los cuales algunos son esquinas de los que se calculan los descriptores ORB y los empaquetamos en BoW. El algoritmo utiliza ambos tipos de puntos de esquinas y no esquinas para realizar el *Tracking*, mientras que solo los puntos que son esquina se utilizan para el cierre de bucle. De esta forma, podemos hallar la posición en entornos con pocas texturas y también encontrar características comunes entre Keyframes si lo necesitásemos.



(a)

Figura 3.6: diferencias entre puntos escogidos con DSO y LDSO.

3.1.5. Comparativa de los algoritmos más representativos

A continuación se presenta una tabla que muestra las características principales de cada algoritmo. Esta tabla es similar a la que aparece en [Perdices García, 2017] pero en este caso se ha añadido el algoritmo LDSO.

Funcionalidad	Mono-SLAM	PTAM	ORB-SLAM	LDSO
Probabilístico	Sí	No	No	No
Hilos de ejecución	1	2	3	3
Emparejamien-to	Parches	Parches	ORB	Métodos directos
Puntos 3D con incerti-dumbre	No	No	No	Sí
Mapa inicial	Dado	Homograf.	Homog. /Matriz F.	Incerti-dumbre
<i>Keyframes</i>	No	Sí	Sí	Sí
Puntos en mapa	Cientos	Miles	Miles	Miles
Mapa denso	No	No	No	Semi-denso
Gestión de mapas grandes	No	No	Sí	Sí
Relocalización	No	Sí	Sí	Sí
Rechazos de espurios	No	No	Sí	No
Cierre de bucle	No	No	Sí	Sí

3.2. Herramientas y datasets para evaluar algoritmos SLAM

En este apartado trataremos varias herramientas que podemos utilizar para hacer comparaciones de rendimiento (*benchmarking*) sobre los resultados de algoritmos SLAM y evaluar y comparar dichos algoritmos.

1. Computer Vision Group TUM

Este grupo proporciona varias herramientas para evaluar el rendimiento de algoritmos VSLAM², permite evaluar trayectorias y compararlas con la trayectoria *ground truth*.[Sturm *et al.*, 2012] Utiliza principalmente 2 métodos, el error absoluto de la trayectoria *absolute trajectory error (ATE)* y el error relativo a la posición *relative pose error (RPE)*. Podemos encontrar en la web HERE scripts descargables para ambas métricas.

Las trayectorias que vayan a ser evaluadas deben ser almacenadas en un archivo de texto, donde cada línea contendrá una única posición con el siguiente formato('timestamp tx ty tz qx qy qz qw')

Donde el campo *timestamp*, proporciona el número de segundos, tx ty tz, dan la posición de la cámara con respecto al origen de coordenadas del mundo real, y el vector qx qy qz qw, dan la orientación de la cámara con respecto al origen de coordenadas del mundo real en formato de quaternionio.

Absolute Trajectory Error (ATE): El error absoluto de trayectoria mide la diferencia que existe entre cada punto de la trayectoria estimada y la trayectoria verdadera. Como paso de preproceso se realiza una asociación entre la posición estimada con la posición verdadera utilizando el emparejamiento por *timestamps* o marcas de tiempo. Tras esta asociación, se alinea la trayectoria real con la estimada usando SVD (*Singular Value Decomposition*). Por último, calcula la diferencia entre cada par de posiciones y devuelve valores estadísticos como la media, mediana y desviación estandar de estas diferencias. También es posible obtener gráficos con las dos trayectorias.

Relative Pose Error (RPE): La herramienta proporciona un script en python (*evaluatrpe.py*) que obtiene el error entre el movimiento relativo entre pares de sellos temporales (*timestamps*). Por defecto, el script calcula el error entre todos los pares de *timestamps* del fichero de trayectoria. Como el número de pares de *timestamps* en la trayectoria estimada es cuadrático se pueden poner cotas con un número máximo de pares de *timestamps*. Opcionalmente, también se puede elegir usar un tamaño de ventana fijo. En este caso, cada posición en la trayectoria estimada es asociado con posteriores posiciones dependiendo del tamaño de ventana y unidad. Esta técnica de evaluación es util para calcular el desvio o deriva.

2. Trajectory Evaluation Toolbox for Visual(-inertial) Odometry

²<https://vision.in.tum.de/data/datasets/rbgd-dataset/tools>

Esta herramienta está desarrollada en python e incluye :

métodos de alineamiento de trayectorias para diferentes modalidades de sensores y métricas de error tales como ATE y *Relative Odometry Error*. [Zhang and Scaramuzza, 2018]

El software ha sido diseñado para uso fácil. Dados dos ficheros de texto donde especificaremos la trayectoria estimada y la verdadera, la herramienta establece automáticamente el emparejamiento de tiempos, realiza alineamiento de trayectoria y calcula distintos errores métricos con una línea de comandos. También se puede utilizar para comparar diferentes algoritmos con múltiples *datasets*. Para que la aplicación pudiese ser utilizada con diferentes formatos, también se proporcionan varios scripts para convertir otros formatos conocidos (e.g., EuRoC, rosbag) al formato utilizado por la herramienta. El formato utilizado para los ficheros de datos es el siguiente: `timestamp tx ty tz qx qy qz qw`, al igual que en la herramienta SLAMTestBed

3. SLAMBENCH

SLAMBench es una herramienta de la Universidad de Edimburgo, creada para evaluar sistemas SLAM sobre un conjunto extensible de datasets y métricas. [Bodin *et al.*, 2018] Actualmente soporta 8 tipos distintos de algoritmos (densos, semi-densos y escasos) y 3 *dataSets*. Es una herramienta que permite la reproducibilidad de los resultados para los sistemas SLAM actuales y posibilita la integración y evaluación de nuevos resultados SLAM.

SLAMBench soporta varios algoritmos diferentes. Entre los algoritmos escasos o poco densos soportaría MonoSLAM, PTAM ,OKVIS y ORB-SLAM2. Entre los métodos densos se soportan KinectFusion, InfiniTAM, ElasticFusion, LSD-SLAM.

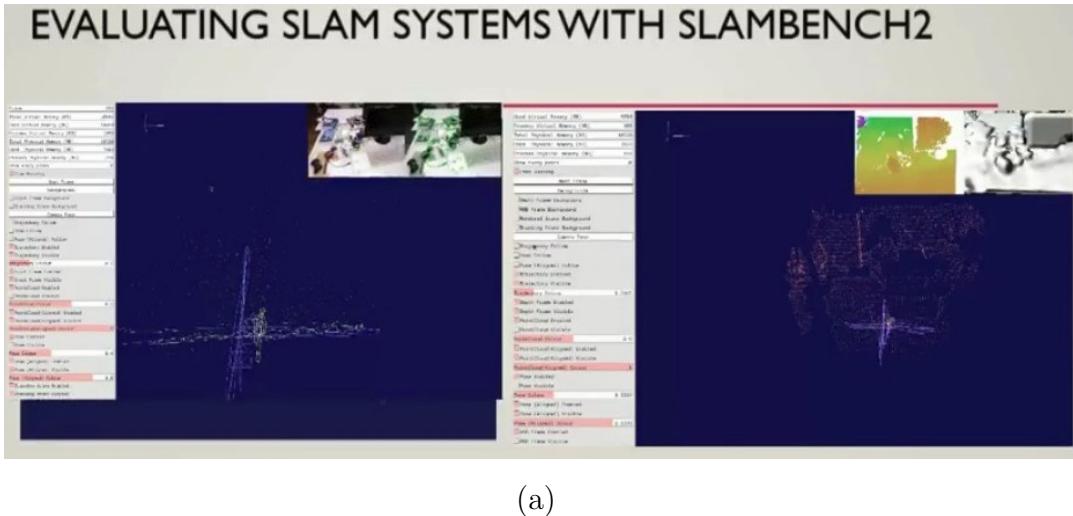


Figura 3.7: captura de la herramienta SLAMBENCH2.

Para medir el rendimiento de algoritmos SLAM se puede utilizar un *framework* para cuantificar la calidad de los resultados teniendo en cuenta exactitud, tiempo de ejecución, uso de memoria y consumo de energía. Esta información puede ser visualizada gracias a su interfaz gráfico. Además SLAMBench ofrece una plataforma con grandes posibilidades para investigaciones futuras, ya sea para diseño de algoritmos como optimizaciones a nivel de implementación. Es una herramienta multiplataforma y se puede utilizar en PCs de sobremesa, portátiles y móviles. Algunos *benchmarks* se han obtenido con Ubuntu, OS y Android. También puede ser utilizado con CUDA. SLAMBench proporciona, entre otras métricas, medidas de exactitud del algoritmo utilizado. Las medidas de exactitud son determinadas comparando datos estimados con los datos *groundtruth*. Absolute Trajectory Error (ATE) y Relative Pose Error RPE son utilizadas para medir la exactitud de la trayectoria. Estas métricas de trayectoria junto con una métrica de mapeo, proporcionan comparaciones cuantitativas para varios algoritmos.

ATE y RPE son calculadas en tiempo de ejecución, con un alineamiento mínimo entre la primera posición más cercana de *groundtruth* y la posición estimada (en términos de *timestamp*). Ya *off-line*, técnicas más complejas de alineamiento pueden ser usadas para comparar técnicas densas y semidensas cuando el mapeo de escalas no funciona. RER (*Reconstruction Error*) se calcula mediante la ejecución del algoritmo *Iterative Closest Point (ICP)* de los modelos de la nube de puntos de la reconstrucción y del *groundtruth*. Como este proceso consume mucha CPU, esta evaluación es también ejecutada *off-line*.

Otras métricas que proporciona SLAMBench es el consumo de energía ,utilización de memoria, velocidad de proceso por fotograma.

SLAMBench también permite elegir entre diferentes sistemas de interfaz de usuario. Métricas de evaluación pueden ser cambiadas y customizadas, así como el interfaz de usuario gráfico (GUI), mientras mantiene independencia de los datasets y algoritmos. Por ejemplo, el visualizador nativo está basado en la librería Pangolin, puede ser reemplazado por un visualizador ROS.

SLAMBench está siendo una herramienta muy útil en robótica y Sistemas de Realidad Aumentada (AR). Aunque un gran número de algoritmos SLAM han sido presentados, no se ha investigado lo suficiente para tratar de unificar el interface de estos algoritmos, o realizar comparaciones de todas sus capacidades en conjunto. Esto presenta un problema ya que diferentes aplicaciones SLAM pueden tener diferentes requisitos funcionales y no funcionales. Por ejemplo, una solución para Realidad Aumentada desarrollada para móviles tendría que optimizar el consumo de energía, mientras que otra solución diseñada para vehículos de navegación autónoma estaría enfocada a funcionar con la mayor exactitud posible. SLAMBench2 es una herramienta de evaluación que compararía sistemas SLAM actuales y futuros, utilizando una lista extensible de datasets, mientras utiliza una lista comparable de métricas de rendimiento. SLAMBench2 es un software que está disponible de manera pública.³

4. The Kitti Vision Benchmark Suite

Este entorno de aplicaciones que utilizan mapas y secuencias grabadas desde la plataforma de coches autónomos Annieway [Stiller *et al.*, 2008] para crear nuevos desafíos o retos en las tareas comparativas o *benchmarking* de Visual SLAM [Geiger *et al.*, 2012], y están investigando en varios campos como: vision estéreo, flujo óptico, odometría, detección de objetos en 3D y seguimiento de objetos 3D. La exactitud de los conjuntos de datos verdadero es medida gracias al scanner láser *Velodyne* y a los sistemas de localización GPS con los que van equipados sus coches autónomos. Los datasets han sido grabados en la ciudad de Karlsruhe.

Además de proporcionar todos los datos en formato *raw*, para cada uno de sus *benchmarks*, también proporcionan una métrica de evaluación y una web de evaluación. En experimentos preliminares se ha comprobado que métodos que obtienen una puntuación alta en algunos *benchmarks*, cuando son aplicados al mundo

³<https://github.com/pamela-project/slambench2>

real obtienen unos resultados por debajo de la media. El objetivo es reducir esta tendencia y completar los benchmarks existentes proporcionando benchmarks en el mundo real con dificultades novedosas para la comunidad.

Un ejemplo podría ser el benchmark de odometría, que consiste en 22 secuencias estereo, grabadas en formato *png*. En el dataset se proporcionan 11 secuencias con trayectorias verdaderas para entrenar y 11 secuencias sin anotaciones verdaderas para evaluar. Para este benchmark se pueden proporcionar resultados usando una cámara o un sistema de cámaras estereo. La única restricción que se impone es que el método debe ser totalmente automático (no se permite el etiquetado manual de cierre de bucle) y que el mismo conjunto de parámetros es usado para todas las secuencias.

Para todas las secuencias de test, su evaluador estima los errores de traslación y rotación. En una tabla de evaluación se establece un ranking de métodos de acuerdo con la media de esos valores, donde los errores son medidos en porcentaje para la traslación y en grados por metro para la rotación.



(a)

Figura 3.8: Coche autónomo Annieway utilizado con Kitti.

Capítulo 4

Herramienta SLAMTestbed

En este capítulo se detalla el diseño y la implementación de SLAMTestbed, una aplicación herramienta diseñada y creada para comparar cuantitativamente algoritmos SLAM. El diseño de algoritmos SLAM, y de Visual SLAM en particular es todavía una disciplina abierta en periodo de expansión y cuenta cada vez con un mayor número de aplicaciones reales, entre las que destaca la navegación de vehículos autónomos o las aplicaciones de realidad aumentada. Existe un gran número de algoritmos, por lo que se necesitan herramientas que permitan medir la precisión, robustez y velocidad de cada algoritmo.

Comenzaremos explicando el diseño de la herramienta SLAMTestbed desde un punto de vista de Caja Negra, explicando sus entradas y sus salidas. Se continuará con la explicación de su diagrama de bloques y se finalizará profundizando en cada uno de los componentes principales de la herramienta, es decir los distintos módulos que permiten calcular el Registro Espacial (Escala, Traslación y Rotación) y el Registro Temporal (interpolación de frecuencias y cálculo de *Offset* o Desplazamiento Temporal) entre dos secuencias de posiciones y orientaciones 3D.

4.1. Diseño

En esta sección no entraremos en los detalles de la implementación de la herramienta SLAMTestbed, si no que lo trataremos como una *caja negra*. En la entrada del sistema tendremos dos secuencias de puntos 3D. Las secuencias procesados por esta herramienta serán ficheros de texto cuyos registros tendrán los siguientes 8 campos:

Timestamp, X, Y, Z, qx,qy,qz,qw

Uno de los *dataset* será la verdad absoluta, y el segundo dataset será la posición y orientación en 3D obtenidos tras aplicar un algoritmo Visual SLAM, correspondiendo cada registro del *dataset* con una posición de la cámara. Una vez procesados los dos *datasets* por la herramienta, obtendremos como salida, las transformaciones estimadas por la herramientas entre la verdad absoluta y las posiciones y orientaciones calculadas por el algoritmo de SLAM. Además, se obtendrá un conjunto de estadísticos que miden el error cometido en las estimaciones de SLAM y así poder medir la precisión de los algoritmos.

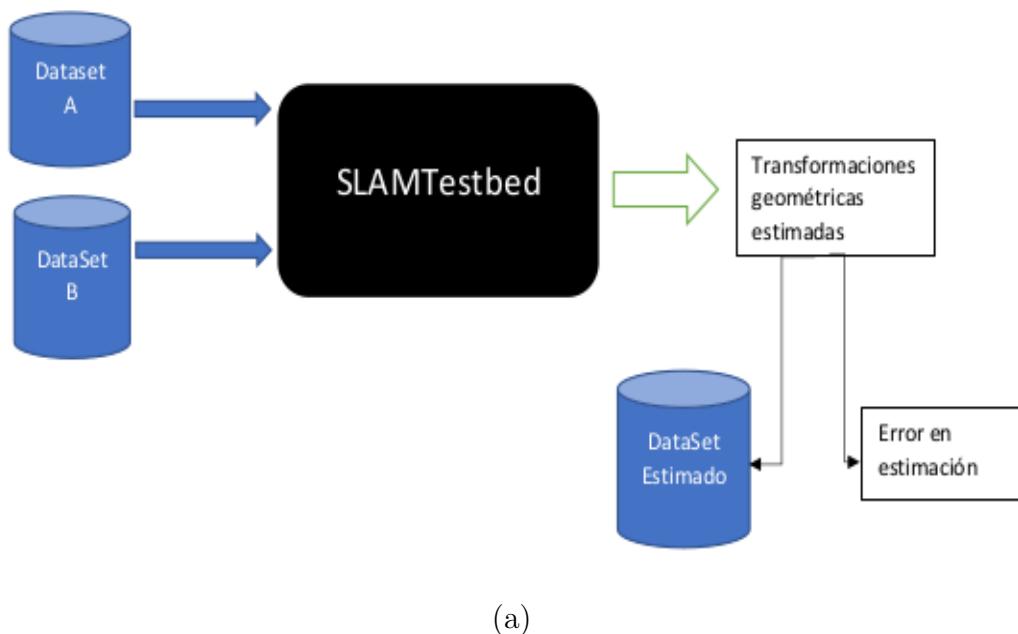


Figura 4.1: El diseño de Caja Negra de la herramienta SLAMTestbed.

Una vez explicadas las entradas y salidas de la aplicación ahora explicaremos con más detalle la implementación de la herramienta SLAMTestbed.

El objetivo principal de la herramienta desarrollada es calcular el error existente entre una secuencia con las posiciones y orientaciones 3D verdaderas (dataset A) y la trayectoria calculada por el algoritmo de SLAM (dataSetB). Para que esto sea posible, necesitaremos antes eliminar algunas variables que no permiten comparar directamente las dos trayectorias, como son la escala, el offset temporal y la transformación en 3D entre ellas. Por ello, necesitaremos calcular un nuevo dataset (dataset estimado), que sea comparable con el datasetA.

Las principales funciones o módulos utilizados para obtener el dataset estimado son:

Cálculo de PCA :

El análisis de componentes principales (o PCA), nos permitirá reducir los dos *dataSets* a sus componentes principales, lo que posibilita estimar la escala y el offset existente entre ellos.

Estimación de Escala :

Estima la diferencia de escala entre los dos *datasets* a partir de los datos proporcionados en el cálculo de componentes principales.

Estimación de Offset temporal :

Con este módulo podremos hallar la diferencia entre marcas de tiempos de los 2 datasets, ya que pueden haber comenzado en periodos de tiempo distintos.

Interpolación para igualar frecuencias de muestreo :

Con la interpolación podremos igualar en frecuencias los dos datasets, en caso de que éstas sean distintas, que es lo habitual por que el algoritmo SLAM genera estimaciones a diferente ritmo del que vienen las muestras de la secuencia con verdad absoluta.

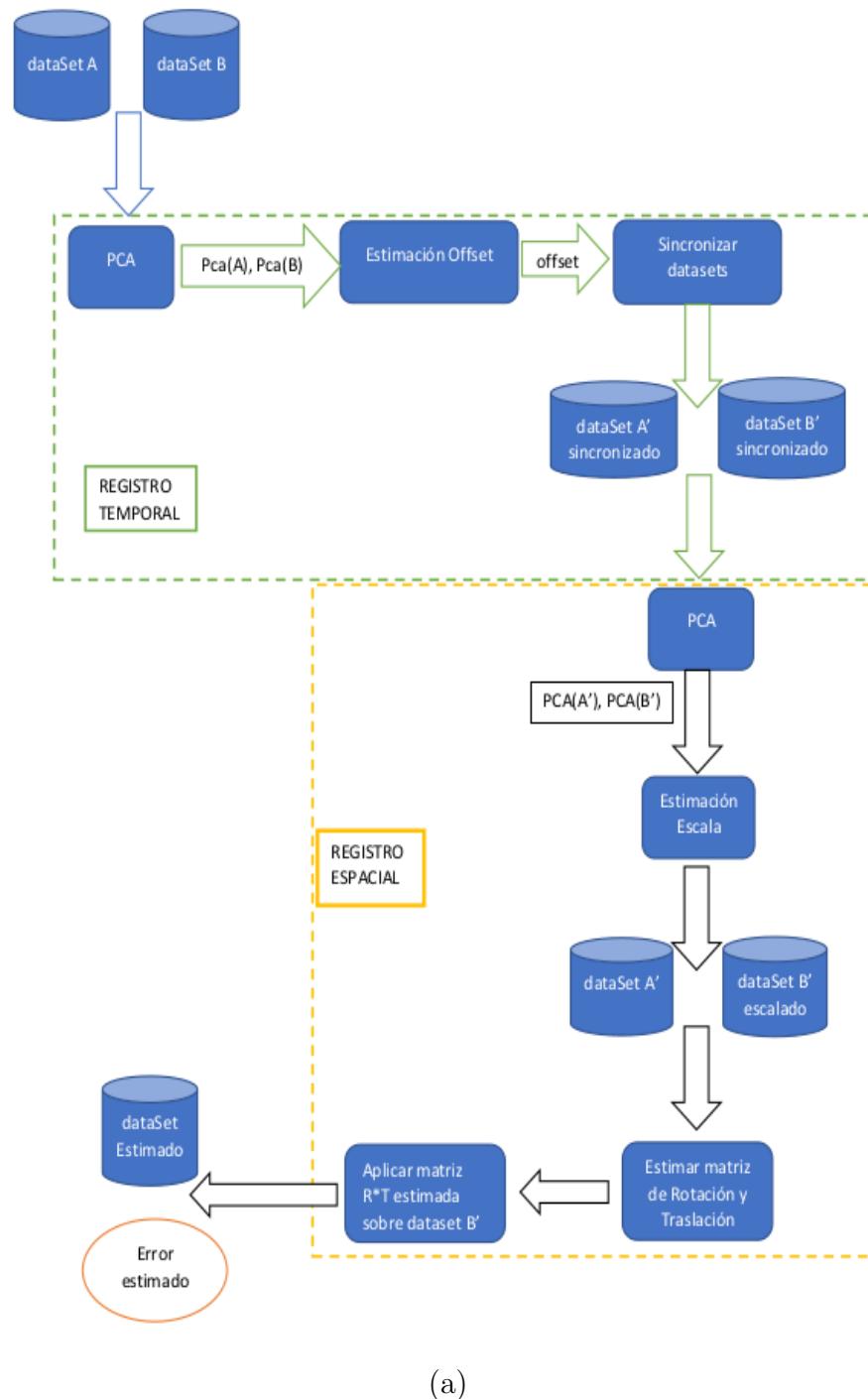
Operaciones de Registro para estimar la Rotación y Traslación :

Permitirá estimar las traslación y rotación existentes entre el datasetA y el datasetB y llevarlas así al mismo sistema de referencia espacial, donde ya son directamente comparables.

La explicación al flujo de datos seguido en la Figura 4.2 es el siguiente:

Primero se calcula la descomposición en componentes principales (PCA) para cada uno de los *datasets* obteniendo como resultado *dataSetA'* y *dataSetB'*. Posteriormente se estima el *offset* o desplazamiento de tiempo que existe entre las 2 secuencias de datos o *datasets*. Corrigiendo el offset en el *datasetB*, sumando el valor del *offset* calculado a los valores de tiempo del *datasetB* así las dos secuencias tienen ya el mismo origen de tiempos. Una vez tengamos calculado el offset, unificaremos las frecuencias de los 2 datasets mediante interpolación. En este proceso de interpolación se generarán nuevas muestras para los *datasets'*. Cuando tengamos unificada la frecuencia para los dos datasets, obtendremos de nuevo PCA de ambos datasets y podremos obtener la escala. Posteriormente estimaremos las transformaciones de rotación y traslación para pasar del *datasetA* al *datasetB*. Aplicaremos dichas transformaciones sobre el *datasetA*, para obtener un nuevo dataset, el *dataset Estimado* que pintaremos en pantalla de color rojo.

A continuación se explicarán con más detalle cada uno de estos módulos.



(a)

Figura 4.2: Diagrama de bloques de la herramienta SLAMTestbed

4.2. Estimador PCA y Cálculo de Escala

En análisis de componentes principales (*PCA* según sus siglas en inglés) es el algoritmo más utilizado para reducir las dimensiones de un conjunto de datos. Como resultado de aplicar PCA sobre un conjunto de datos, obtendremos un nuevo conjunto de datos en términos de nuevas variables no correlacionadas, que denominaremos componentes principales.

Las componentes principales se ordenarán por su Varianza, de esta forma la primera componente principal será la que mayor varianza tenga e identificará un eje. Un segundo eje ortogonal al primero vendrá definido por la segunda componente principal e identificará la segunda mayor varianza. Finalmente obtendremos un tercer eje ortogonal a los dos primeros, definido por la tercera componente principal que identificará la tercera mayor varianza.

En nuestro caso, este análisis nos permitirá obtener el tamaño de cada dataset en cada una de sus componentes (x,y,z). Este análisis es necesario puesto que los ejes de las dos secuencias de entrada no tienen porqué coincidir. Por ello, con los valores obtenidos con PCA podremos generar dos nuevos datasets cuyos ejes sean comparables, es decir que estén alineados. A partir de los nuevos datasets generados podremos calcular la diferencia de tiempos entre los dos datasets y posteriormente la escala entre ambos.

Se implementan dos métodos de cálculo de PCA, el primero es realizando una descomposición Eigen *eigenDecomposition*, el segundo utilizando el método SVD (utilizando la librería Eigen), que describiremos a continuación en pseudocódigo.

```
# Obtener el valor medio de cada componente
mX = media (dataset.x)
mY = media (dataset.y)
mZ = media (dataset.z)

#restar la media a cada componente x, y z

dataset2.x = dataset.x - mX
dataset2.y = dataset.y - mY
dataset2.z = dataset.z - mZ

#Calcular el nuevo dataset
```

```

cov = obtenerMatrizCovarianza( dataset2 )
svd = CalcularSVD( cov )
pca = Matriz V del resultado svd
datasetResultado = dataset * Matriz ( svd . V )

```

Una vez que los ejes de ambos datasets sean comparables, podremos calcular la diferencia de escala entre los dos datasets, lo que permitirá igualar la escala entre los datasets obteniendo así el primer paso del registro espacial. El algoritmo utilizado está basado en la utilización de los valores singulares de cada dataset, como se muestra en el siguiente pseudocódigo:

```

S1 = svd( dataSet1 ).valoresSingulares
S2 = svd( dataSet2 ).valoresSingulares
scalaX = S2(0) / S1(0)
scalaY = S2(1) / S1(1)
scalaZ = S2(2) / S1(2)

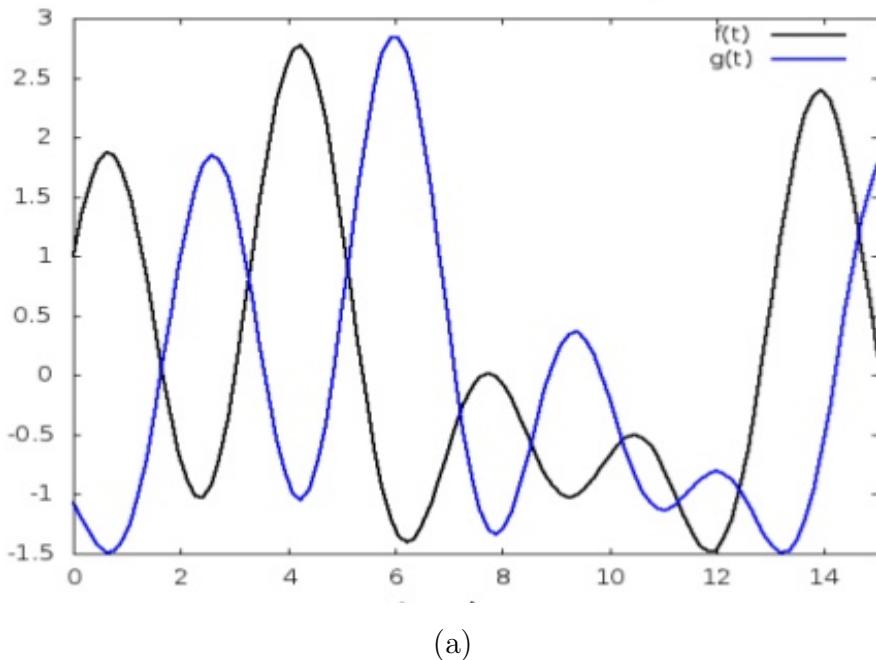
```

4.3. Estimador Offset Temporal

Este módulo es necesario puesto que los datasets pueden tener un desfase en su inicio, incluso aunque hayan sido obtenidos a partir de la misma secuencia, debido a la propia implementación de los algoritmos SLAM. El método principal utilizado para estimar el offset se realiza mediante el cálculo de la Correlación Cruzada. Este método conlleva una gran carga para la CPU ya que incluye múltiples cálculos para interpolar y calcular la correlación cruzada entre los dos datasets. Este algoritmo es iterativo y utilizaremos un dataset como base de tiempo fijo y otro dataset B que se deslizará en el tiempo desde $-T$ hasta $+T$ en pequeños incrementos de tiempo, pasos o *steps* en cada iteración, calculando para cada paso la correlación cruzada entre el dataset deslizante y el dataset original (que se mantiene fijo en el tiempo). En cada iteración del algoritmo se calculará la correlación para un paso determinado, almacenando la memoria el paso cuya correlación sea mayor, lo que nos indicará el offset temporal entre las dos secuencias. Para ello calcularemos la interpolación para los puntos 3D orientados que estén dentro del rango temporal de los dos datasets. Una vez interpolados los puntos 3D comunes de las dos series temporales tendremos dos series nuevas, para las cuales calcularemos la correlación cruzada aplicando el algoritmo clásico para dos series temporales. Pero antes, y como cada serie temporal

tiene 3 coordenadas (x, y, z para cada punto 3d), debemos transformar estas 3 coordenadas en un sólo valor para cada punto 3D de cada serie, para ello tendremos que calcular la distancia al origen de todo punto 3D, y por tanto la correlación cruzada se calculará sobre los 2 datasets convertidos a distancias al origen de coordenadas. Para un punto 3d (x, y, z), la distancia 3d respecto al origen se calculará:

$$\sqrt{(x - 0)^2 + (y - 0)^2 + (z - 0)^2}$$



(a)

Figura 4.3: Gráfico que muestra el desfase del offset entre dos series temporales y el resultado de la correlación cruzada.

```

dataA(d) = CalcularDistanciaAlOrigen( dataSetA(x,y,z) )
dataB(d) = CalcularDistanciaAlOrigen( dataSetB(x,y,z) )
mA = CalcularMedia(dataA)
mB = CalcularMedia(dataB)

```

Para cada fila

$$sx = dataA(i) - mA$$

$$sy = dataB(i) - mB$$

```

denom = sqrt( sx*sy );
step = 0.01
Desde i=-T hasta i = T
    Si ( dataB.t < dataB.t )
        j=j+1
    continuar

    sino si ( dataB.t > dataA.t )
        i=i+1
    continuar

    sino si ( dataB.t == dataA.t )
        sxy=(dataA -mA) * (dataB-mB)
        i++
        j++

r = (sxy) / denom;
r= fabs(r);

```

4.4. Interpolador temporal

El módulo de interpolación se utiliza para sincronizar a igual frecuencia 2 secuencias, cuyas frecuencias de muestreo no sean iguales, algo que es común entre los algoritmos de SLAM. En este proceso de interpolación se generarán nuevos registros en el dataset a partir de la interpolación de muestras entre dos marcas de tiempo. Este interpolador puede configurarse para que tenga distintos comportamientos:

1. **Interpolación a la frecuencia menor** de los dos datasets. Aunque se generarán nuevos registros sincronizados por interpolación, en general se perderán registros del dataset de mayor frecuencia.

2. **Interpolación a la frecuencia mayor** de los dos datasets. En este caso se generarán nuevos registros para el dataset de menor frecuencia.
3. **Interpolación a frecuencia común**, en este caso, los dos datasets se deberán sincronizar a la frecuencia deseada por el usuario.

En los tres casos, la interpolación se realiza sobre las secuencias temporales de los 2 datasets, pero además se calcula la interpolación para las coordenadas X,Y,Z de cada punto 3D y valores (qx,qy,qz,qw) de cada cuaternio. La función *slerp* de la librería Eigen nos permite calcular la interpolación entre cuaternios, viene de la abreviatura de la definición en inglés *Spherical Linear intERPolation*, creada para animaciones de rotaciones 3D utilizando cuaternios.

La fórmula para la interpolación lineal utilizada sería:

$$y - y2 = (t - t2) * \frac{y3 - y2}{t3 - t2}$$

Donde t sería el nuevo valor de tiempo para el cual deberíamos interpolar los nuevos valores de las coordenadas X,Y,Z. Los valores t3,y3 hacen referencia al valor de la secuencia en t+1 Los valores t2,y2 hacen referencia al valor de la secuencia en t-1

4.5. Registro Espacial

Una vez modificados los dos datasets para que tengan la misma escala y la misma frecuencia de muestreo y el mismo origen temporal, el siguiente paso es obtener la rotación y la traslación que existe entre ellos. Esta matriz servirá para transformar el segundo dataset, y así poder calcular el error existente en la trayectoria obtenida por el algoritmo de SLAM. Los pasos necesarios para realizar este registro espacial son los siguientes:

Hallar la matrices de rotación y traslación .

El algoritmo desarrollado para esta funcionalidad sería el siguiente:

```
# Hallar los centroides para las coordenadas X,Y,Z
centA = centroides (A)
centB = centroides (B)

# Restar los centroides a sus respectivas matrices
```

```

A= A-centA
B= B-centB

# Calcular el producto de las matrices
H = A.traspuesta * B

# Obtener valores singulares de la matriz H
S,U,V = svd(H)

# Obtener la matriz de Rotacion
R = V*U.traspuesta

# Obtener la matriz de Traslacion
t= -R * centA + centB;

```

Transformar un dataset aplicando las matrices de Rotación y Traslación estimadas

Este método tomará un dataset, lo multiplicará por la matriz de rotación y le sumará la matriz de traslación.

Transformar los cuaternines aplicando la matriz Rotación.

Con este método conseguimos transformar los cuaternios del dataSetA al dataSetB. Para ello convertiremos la matriz de rotación en un cuaternion que llamaremos "q". Por último multiplicaremos cada cuaternion p de la siguiente forma.

$$q * p * q.inverso$$

Estimar las matrices de rotación y traslación mediante la técnica de RANSAC. Estimaremos las matrices de Rotación y Traslación utilizando los datos de los 2 datasets, eliminando primero los puntos espúreos (outliers) mediante el algoritmo RANSAC [Fischler and Bolles, 1981]. Se ha elegido RANSAC por que permite estimar la matriz $R*T$ entre ambas secuencias con robustez, de modo que la estimación es fiable.

El algoritmo seguido ha sido el siguiente:

```

Mientras n < MaxIteraciones hacer
    Seleccionar n inliers
    Estimar una primera matriz de R*T

```

```

    aplicar R*T sobre subconjunto NoInliers
    Si el error < threshhold
        agregar candidato a inliers
        aplicar RT sobre inliers , NoInliers
        medir el error
        seleccionar menor error
        incrementar iteraccion
    Devolveremos las matrices de R*T con menor error

```

4.6. Cálculo de Estadísticas

Este bloque mide las diferencias entre dos secuencias pero cuando una de llas es la de posiciones verdaderas y la otra es de posiciones estimadas, entonces esas diferencias se interpretan correctamente como el error en las posiciones estimadas.

Para medir el error cometido entre el dataset con posiciones orientadas verdaderas y el dataset Estimado utilizaremos el módulo de Cálculo de Estadísticas. Este módulo permite calcular distintos estadísticos, como el error medio, mediano, máximo y mínimo, así como el Error Cuadrático Medio (Root Mean Square Error). Cuanto menores sean estos valores, mejor será la estimación.

De entre estos estadísticos, el que mejor determina la calidad de la trayectoria estimada por los algoritmos de SLAM es el error cuadrático medio, puesto que penaliza los errores demasiado grandes más que otros estadísticos, algo que es determinante en los algoritmos de SLAM. Cuanto más próximo a 0 sea el valor del RMSE, mejor será la estimación. Si conseguimos un RMSE con valor 0 significa que los datos estimados coinciden con los datos que hemos tomado como verdad absoluta o *groundtruth*.

El RMSE se calcula de la siguiente forma:

$$RMSE = \sqrt{\sum (g - e)^2 / N}$$

donde g es el dataset *groundTruth*

e es el dataset Estimado

N es el número de elementos de los datasets

Otro valor de medida de error calculado es la distancia angular entre 2 rotaciones, que nos sirve para medir el grado de error entre la orientación real y la orientación estimada,

para ello convertimos la matriz de rotación estimada en un cuaternion usando la librería Eigen.

Los pasos para medir este error angular sería:

```
convertir matriz de rotacion en cuaternion q
convertir matriz de rotacion estimada en cuaternion p
normalizar cuaternios q y p
myAngularDistance=p.angularDistance(q);
```

En caso de no conocer la matriz de rotación de la secuencia verdadera, el vector q será igual a (1,0,0,0).

4.7. Interfaz Gráfico de Usuario

Para la herramienta SLAMTestbed también se ha diseñado y creado un Interfaz de Usuario Gráfico (*Graphic User Interface*) que permita pintar los puntos 3D de cada *dataset* en pantalla, así como invocar comandos (a través de clicks de ratón) para ejecutar los distintos módulos que contiene la herramienta.

Esto permite medir la exactitud de los resultados de la aplicación de un algoritmo con los resultados de otro algoritmo o comparar varios resultados de un mismo algoritmo en el que se han aplicado distintos parámetros de ejecución.

Este interfaz gráfico de usuario de 3 dimensiones ha sido desarrollado en C++ con el entorno QT , OpenGL y la librería Eigen.

La librería Eigen¹ es una librería *Open Source* realizada en C++, para cálculos de Álgebra lineal, operaciones con Matrices y vectores, Transformaciones Geométricas.

OpenGL² es un API estandar y se ha utilizado en este proyecto para todo el tratamiento en gráficos 3D, pero apenas proporciona soporte para GUI, es por ello, por lo que se decidió incorporar como herramienta de desarrollo el *framework* QT.

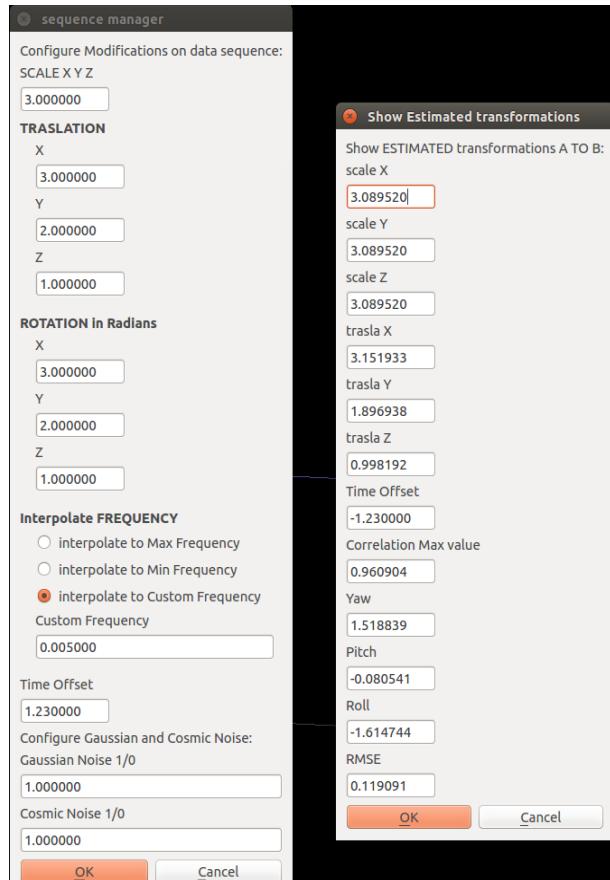
QT³ es un framework de desarrollo en C++, con el cual se ha podido diseñar y desarrollar el interfaz de usuario de SLAMTestbed. Además, contiene el módulo QTOpenGL que permite integrar facilmente código OpenGL con aplicaciones QT.

¹<https://eigen.tuxfamily.org/dox/GettingStarted.html>

²<https://www.opengl.org/>

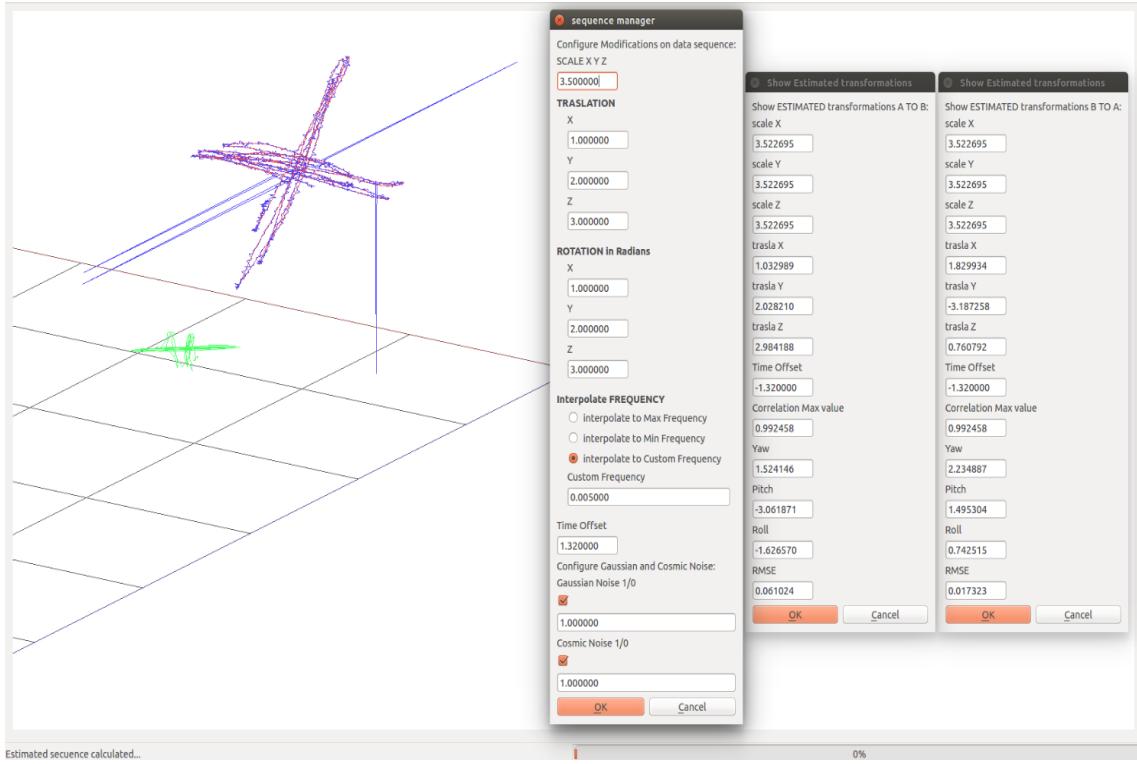
³<https://www.qt.io/developers/>

La aplicación software muestra al usuario un interfaz gráfico que permite leer un archivo de datos con puntos 3D y mostrarlo en pantalla como una nube de puntos 3D (Figura 4.5). Este archivo podríamos denominarlo groundTruth. Con el ratón podremos girar en 3 dimensiones la nube de puntos, acercarnos (Zoom in) o alejarnos (Zoom Out), así como leer un segundo conjunto de puntos 3D y estimar las transformaciones (rotaciones, traslaciones, escala etc) que existen entre ellos. El conjunto de datos resultante tras aplicar las estimaciones será visualizado en pantalla como otra nube de puntos 3D.



(a)

Figura 4.4: Transformaciones realizadas frente Transformaciones estimadas



(a)

Figura 4.5: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

En la pantalla gráfica podremos ver tres datasets. Los puntos 3D del primer dataset *groundtruth* se pintarán en color verde, los puntos del dataset a evaluar o dataset transformado serán de color azul, y por último en color rojo estarán los puntos 3D del dataset estimado. En el caso de que el error entre los dos datasets sea pequeño, los puntos 3D correspondientes al dataset estimado solaparán con los puntos correspondientes a la verdad absoluta, lo que significará que la trayectoria calculada por el algoritmo de SLAM es cercana a la realidad y en este caso el punto estimado (en color rojo) quedará invisible. Además el GUI también muestra los valores de transformación estimadas (Figura 4.4)

Capítulo 5

Validación experimental

En este capítulo mostraremos las pruebas realizadas con la herramienta SLAMTestBed.

Para probar experimentalmente SLAMtestbed hemos construido una herramienta adicional, el transformador, que desde una secuencia original de posiciones orientadas 3D genera una secuencia transformada aplicando varias transformaciones parametrizadas con valores conocidos. Este transformador incluye: traslaciones, rotaciones, cambio de escala, offsets temporales, cambio en frecuencias de muestreo, así como ruidos gaussiano o esporádico. Hemos generado secuencias controladas con unas u otras transformaciones conocidas y las hemos introducido en SLAMtestbed junto a la secuencia original. De este modo, comparando las transformaciones estimadas por SLAMtestbed con las reales hemos verificado que SLAMtestbed las estimaba correctamente, validando así experimentalmente su buen funcionamiento.

Para comprobar los resultados podremos utilizar la ventana que muestra el resultado de las estimaciones una vez han terminado los cálculos. Gracias a la visualización gráfica de los datasets (incluido el dataset estimado) la evaluación de los resultados será más sencilla y fiable, ya que si los puntos 3D del dataset estimado no se aproximan al dataset destino, el desajuste entre los puntos 3d de ambos dataset será perceptible a simple vista. Por otro lado tendremos el indicador RMSE, que cuanto más próximo sea su valor a 0 mejor será la estimación realizada.

A continuación mostraremos las pruebas realizadas para estimar las transformaciones para convertir el dataset A en el datasetB y viceversa, donde el datasetB se obtiene como resultados de aplicar el módulo transformador sobre el datasetA. Se han realizado varias pruebas, en cada una de ellas se han activado un subconjunto de parámetros del módulo transformador, es decir que en una prueba se habrá modificado sólo la escala y un traslación, en otras pruebas se habrá realizado cambios en traslación y rotación , etc.

En cada captura de pantalla aparecerá de forma gráfica , el dataset original (en color verde), el dataset transformado (en azul), y el dataset estimado (en rojo), junto con otras 3 pantallas que indicarán:

1. los valores de los parámetros de las transformaciones realizadas
2. valores de los resultados de transformación de dataset A a dataset B
3. valores de los resultados de transformación de dataset B a dataset A

5.1. Módulo Transformador

La herramienta SLAMtestbed poseé varios módulos accesibles desde el interfaz gráfico. Entre estos módulos podríamos destacar el **Módulo Transformador** que permite realizar transformaciones sobre el conjunto de puntos 3D orientados de una secuencia de entrada, de tal forma que se obtendrá como resultado una segunda secuencia transformada. De esta forma se han podido realizar pruebas para comprobar que los cálculos estimados de Traslación, Rotación y Escala son fiables y tienen un error mínimo.

Los diversos parámetros del módulo transformador son accesibles desde interfaz gráfico de la pantalla. Las transformaciones permitidas por la herramienta son:

-Escala. Permite modificar los datos de entrada a nivel de escala. La escala siempre será mayor que cero y se admitirán números reales. Por defecto tendrá el valor de 1.

-Traslaciones. Se pueden definir traslaciones sobre cada uno de los 3 ejes de coordenadas. La traslación admite números reales positivos y negativos.

-Rotaciones. Permite definir rotaciones sobre cada uno de los 3 ejes de coordenadas. El valor de cada rotación se insertará en radianes. Los valores admitidos son números reales tanto positivos como negativos.

-Offset de tiempo. Con el offset de tiempo podremos introducir un desplazamiento o *gap* en los valores de sello temporal o *timestamp* del fichero de entrada . La exactitud del offset será de centésimas, es decir con 2 decimales.

-Cambio de frecuencia: ón nos permitirá ajustar a la misma frecuencia los 2 datasets. Se podrán realizar 3 tipos de interpolación de los datos.

1. Interpolación a la frecuencia máxima
2. Interpolación a la frecuencia mínima

3. Interpolación a frecuencia personalizada

-Ruido Gaussiano: Una de las transformaciones que podremos aplicar sobre la secuencia de entrada de puntos 3D orientados es un ruido gaussiano a los datos transformados. Sólo se añadiría el ruido Gaussiano a las coordenadas X,Y,Z de cada punto 3D.

-Ruido Cósmico: Otra transformación a aplicar sobre los datos transformados es la incorporación del ruido aleatorio, de intensidad y frecuencia modulables. Cada cierto tiempo aleatorio, con cierta probabilidad, se altera un punto orientado 3D cambiando sus valores de posición y orientación.

5.2. Pruebas de transformaciones individuales

En este apartado mostraremos imágenes del módulo GUI, donde se han realizado varias transformaciones individuales y se han estimado dichas transformaciones con SLAMtestbed. En todas las transformaciones se ha realizado la interpolación de frecuencias al valor 0.05 segundos para ambos datasets, es decir que se toma 20Hz como frecuencia deseada a la que se llevan las dos secuencias de entrada.

El gráfico superior muestra los resultados de la estimación tras una transformación de un cambio de escala. Se aprecia en la captura de pantalla, como el dataset transformado es más grande que el dataset original. Como se puede observar el RMSE es 0.0 y el valor de la escala estimada coincide con el valor de la transformación de escala, en este caso es 2.0

La captura de pantalla superior muestra los resultados de la estimación tras realizar una traslación en el dataset original. Como puede observarse el RMSE es 0.0, y gráficamente el dataset transformado y estimado coinciden en cada posición 3D. Es por este motivo por el que hay ausencia de puntos rojos en la representación 3D. Hay que recordar que el dataset estimado se presenta con puntos rojos. Cuando la estimación no es buena , el dataset transformado y estimado no coincidirán y podremos ver en pantalla los puntos rojos , allí donde precisamente no coincidan dataset estimado. En cuanto a los valores de la traslación estimada puede verse que coinciden con la transformación realizada.

En la captura de pantalla anterior, podremos observar el dataset original en verde, y el dataset transformado tras aplicar una rotación. La rotación se especifica en radianes. Como puede observarse, el error (RMSE) es mínimo 0.00001. Se puede comprobar que aunque la

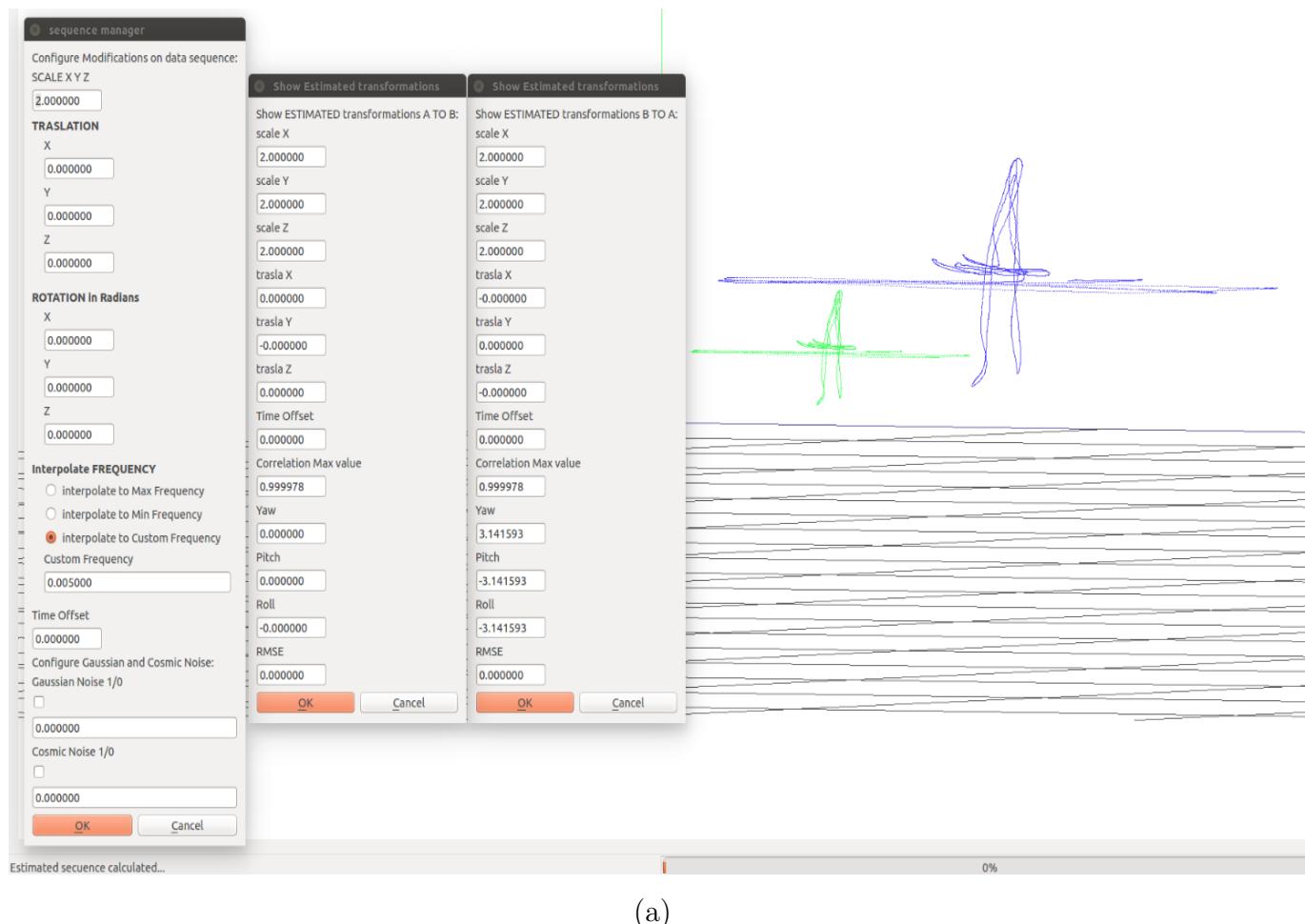


Figura 5.1: Gráfico que muestra los resultados de la estimación tras una transformación de un cambio de escala.

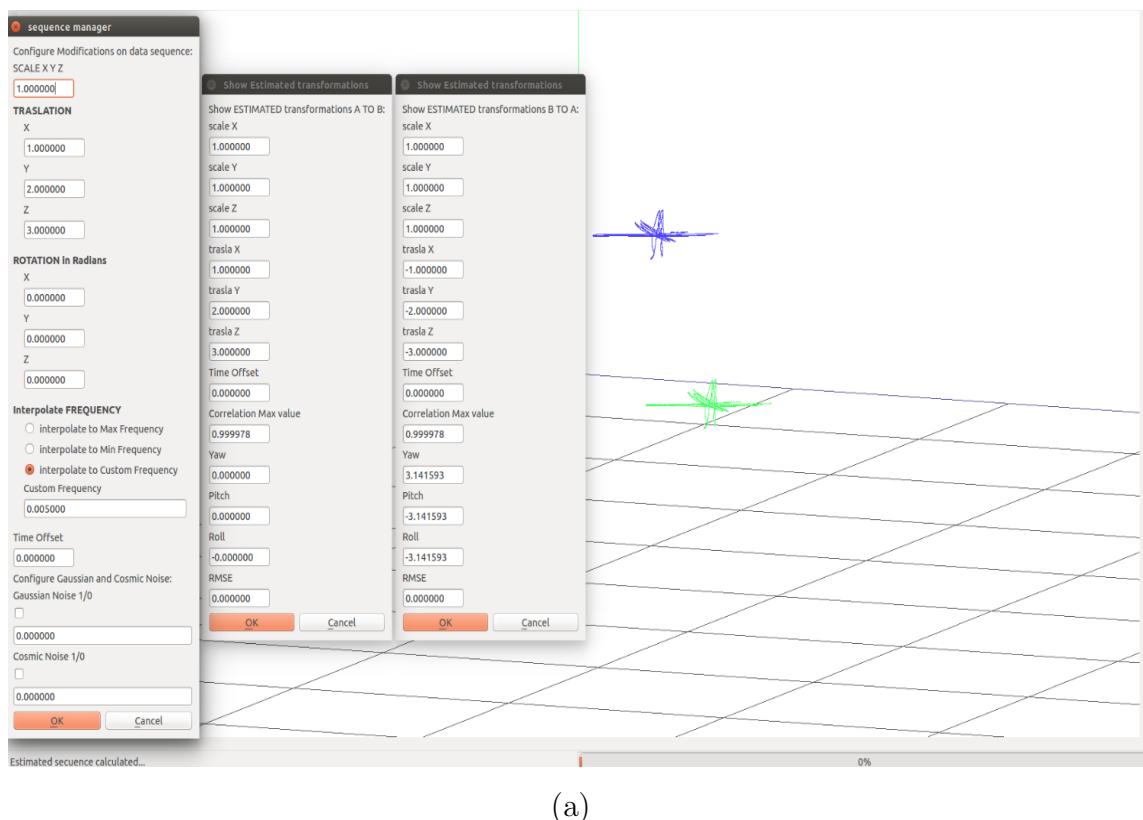


Figura 5.2: Gráfico que muestra los resultados de la estimación de un cambio de traslación.

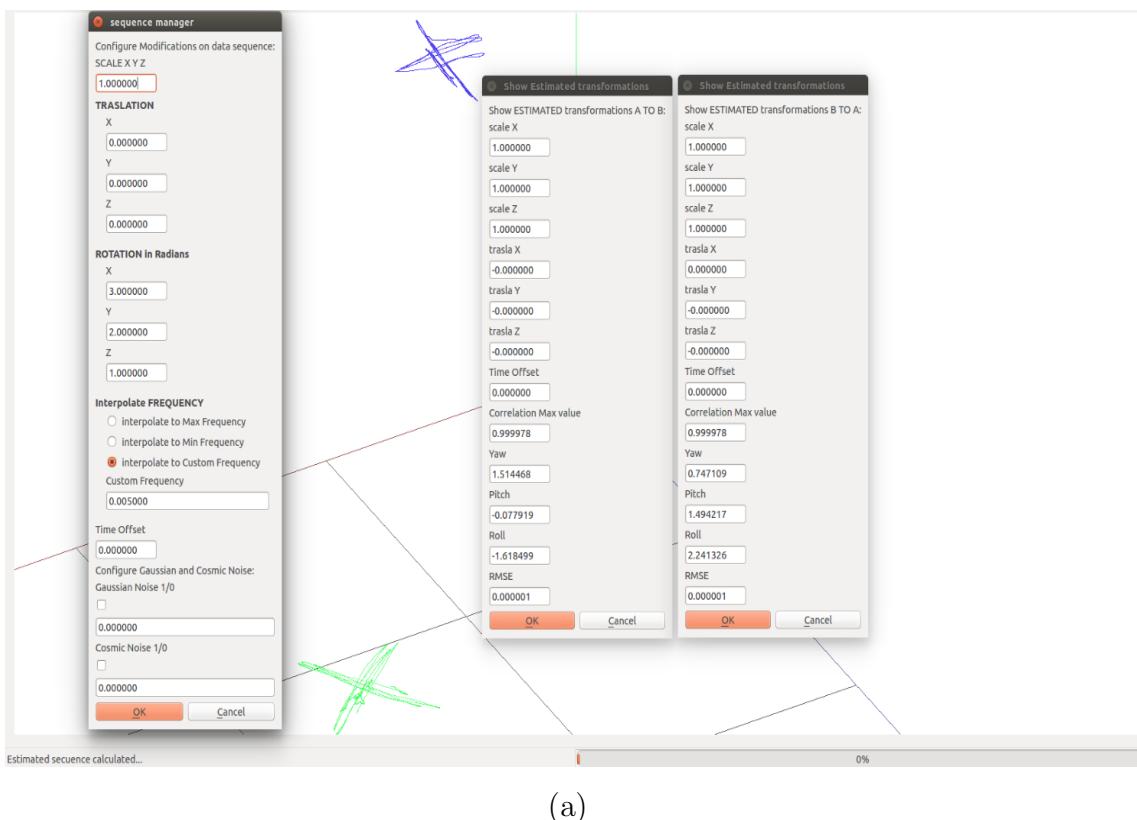
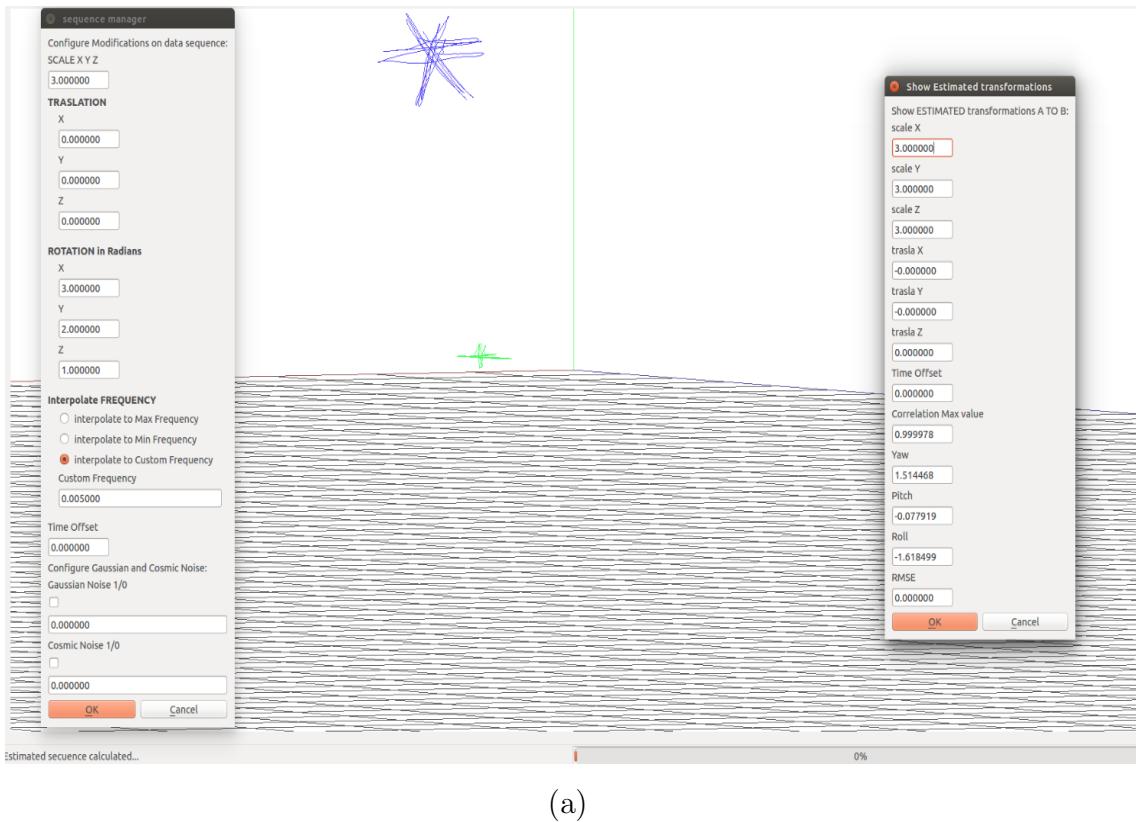


Figura 5.3: Gráfico que muestra los resultados de la estimación de un cambio de rotación.



(a)

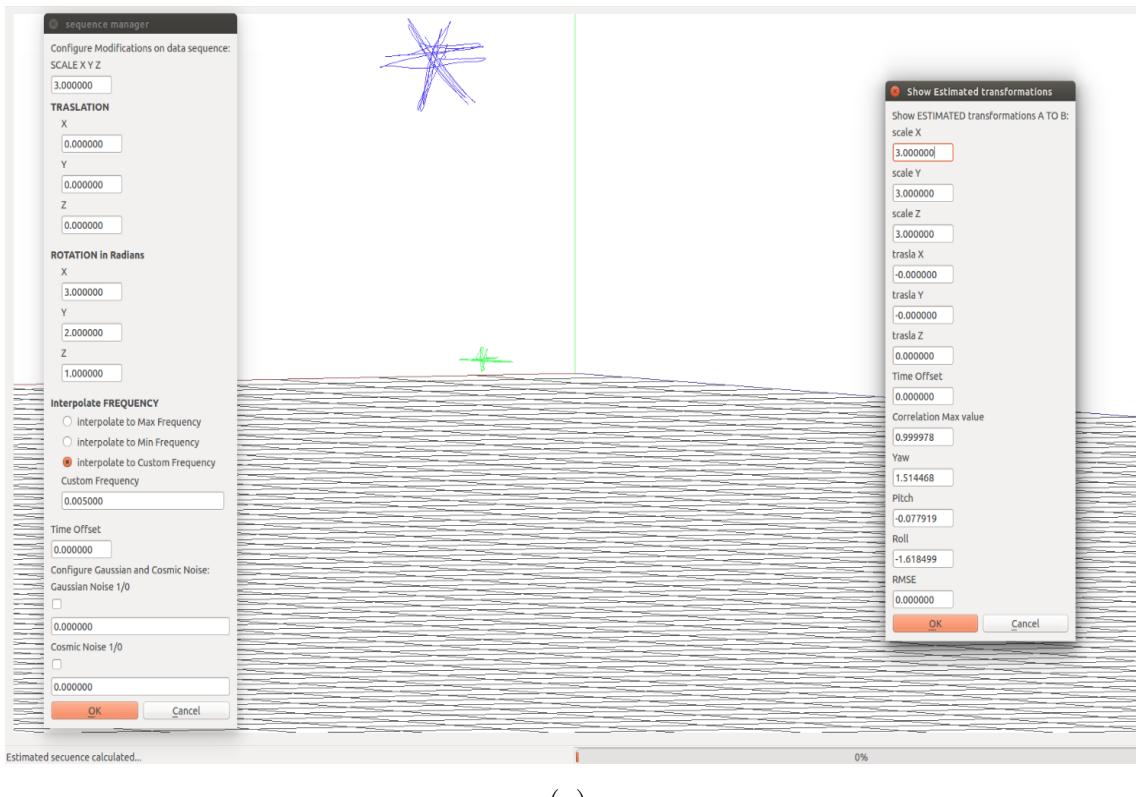
Figura 5.4: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

rotación realizada en los ejes X,Y,Z en la transformación ha sido respectivamente de 3.0, 2.0 y 1.0 radianes, en la estimación estos valores no coinciden, se ha aplicado otra rotación equivalente en radianes.

5.3. Pruebas de transformaciones en parejas

En este apartado, mostraremos imágenes del módulo GUI, donde se han realizado varias pruebas de transformaciones de parámetros en parejas y se han estimado dichas transformaciones. En todas las transformaciones se ha realizado la interpolación de frecuencias al valor 0.05 para ambos datasets.

En la imagen superior, se muestra en color verde, el dataset groundtruth, y en azul el mismo dataset tras realizarle un par de transformaciones en escala y traslación. En la ventana de la derecha se muestran los resultados de la transformación estimada, que coincide con la transformación realizada, además el RMSE es 0. El dataset estimado , también se pinta en color rojo, pero en este caso la estimación es tan buena que coincide con



(a)

Figura 5.5: Gráfico que muestra los resultados de la estimación de un cambio de escala y rotación.

el dataset transformado y no se aprecia ningún punto rojo. En este caso la transformación realizada es de escala y translación. Se puede comprobar que la estimación realizada es correcta ya que coinciden los valores de la escala y translación estimados.

En la imagen superior podemos ver los resultados de aplicar otras 2 transformaciones sobre el dataset original. En este caso se trata de una transformación en escala y rotación. Tras realizar la estimación, el error medido entre el dataset estimado y el dataset transformado es 0.0 . Por tanto podemos decir que la estimación es exacta a la transformación realizada sobre el dataset original.

En este caso, al dataset original (color verde) se le ha aplicado un par de transformaciones (en escala y con offset), dando lugar al conjunto azul. En este caso tambien la estimación es buena , ya que el error RMSE da como resultado 0.0 y el cálculo del offset tambien coincide con el valor del offset que se ha aplicado en la transformación, salvo con signo negativo.

En la imagen superior, se ha aplicado sobre el dataset original un cambio de escala y además se le ha añadido ruido Gaussiano, como puede verse en el dataset resultante (color

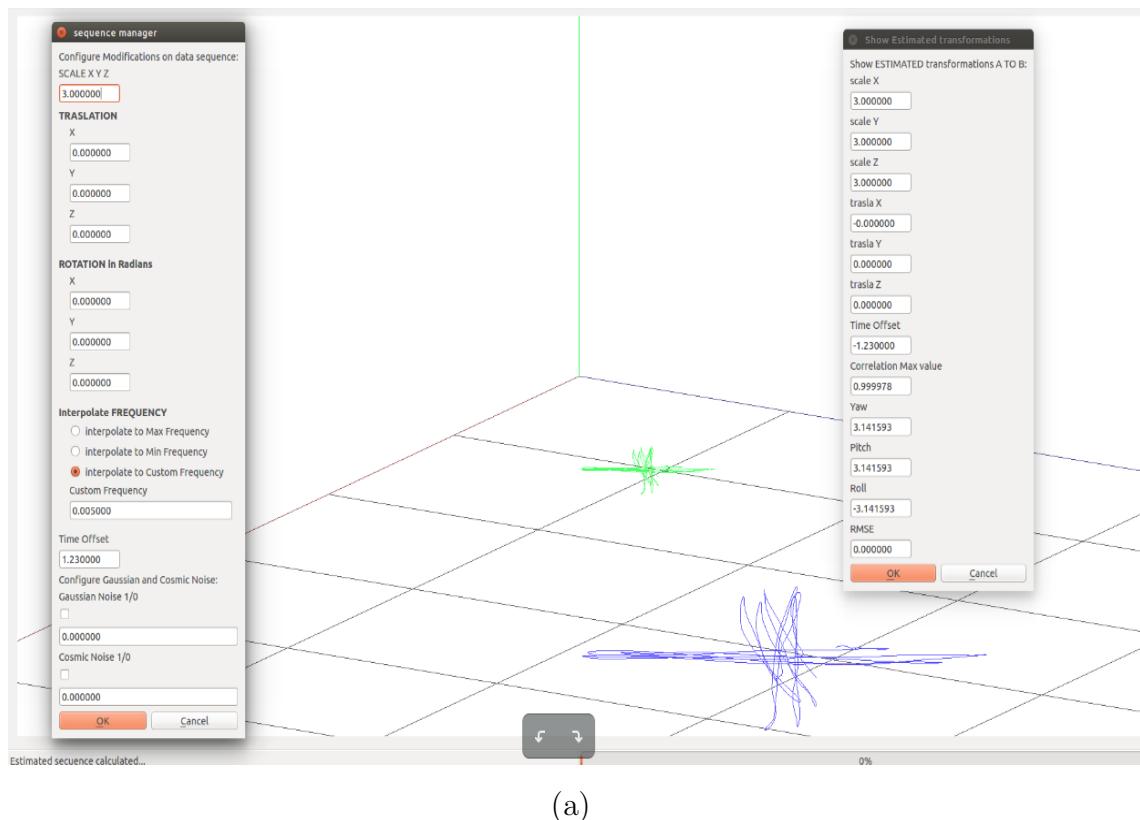


Figura 5.6: Gráfico que muestra los resultados de la estimación de un cambio de escala y offset.

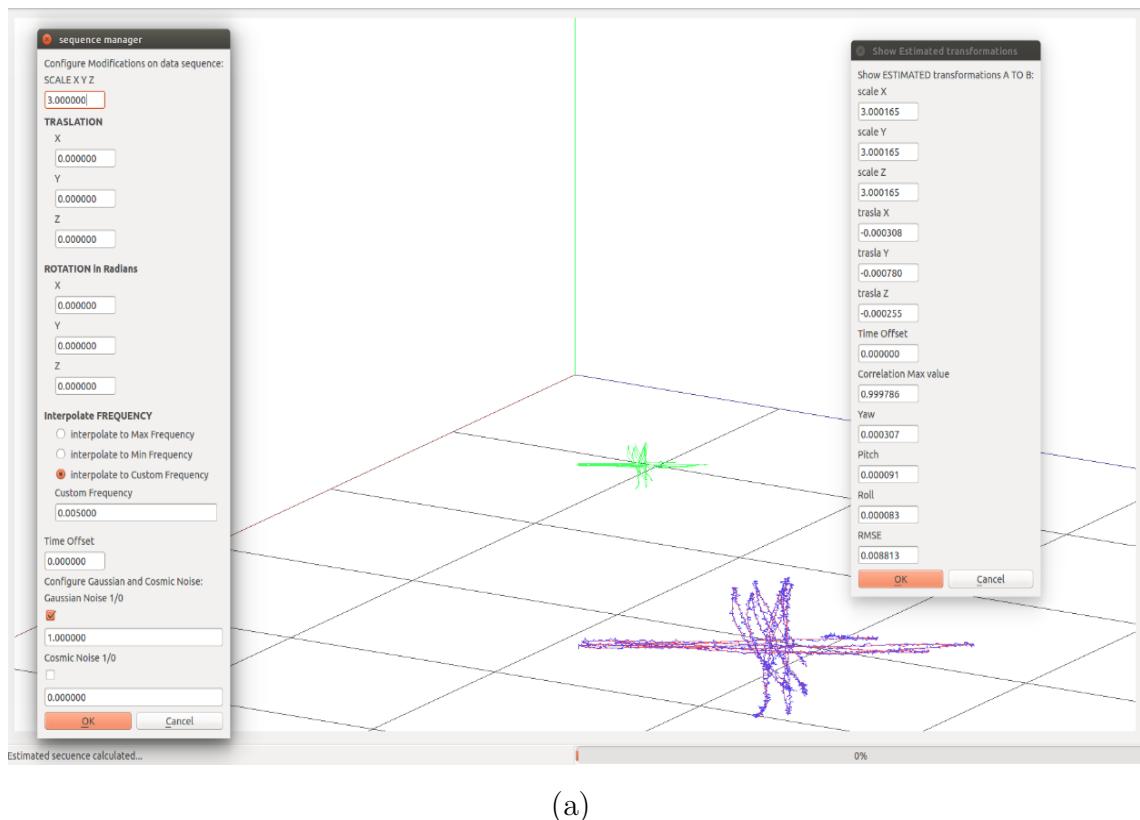
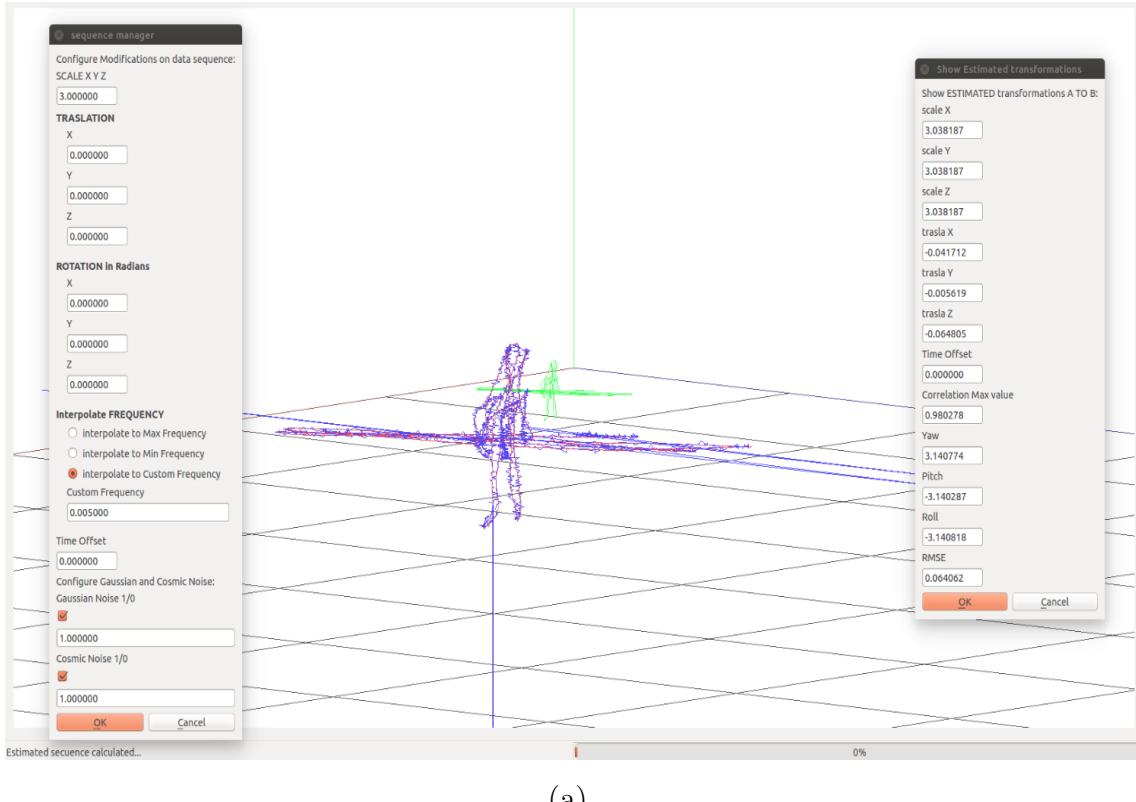


Figura 5.7: Gráfico que muestra los resultados de la estimación de un cambio de escala y ruido Gaussiano .



(a)

Figura 5.8: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

azul). Las estimaciones obtenidas son buenas, pero el error RMSE calculado entre el dataset transformado y el dataset estimando ya no es 0.0, sino 0.008. Esta falta de precisión puede apreciarse a simple vista, ya que como el dataset estimado carece de ruido gaussiano, los puntos del dataset estimado no coinciden con los puntos del dataset transformado, y por tanto ahora si podemos ver buena parte de los puntos del dataset estimado (color rojo).

En este caso, se ha aplicado sobre el dataset original un cambio de escala más ruido gaussiano y cósmico. Al tener ahora ruido cósmico, en algunos puntos del dataset transformado aparecen unos valores muy superiores a la media, estos valores desorbitados harán que aumente notablemente el error entre el dataset transformado y el dataset estimado. Para esta experimento , el valor RMSE es de 0.06. Aunque puede observarse que aún así, la posición de los puntos del dataset estimado coincide con la posición de los puntos del dataset transformado.

En este test, se han aplicado 2 transformaciones sobre el dataset original, una traslación y una rotación. La estimación de la traslación y la rotación coinciden con los valores de las transformaciones. El error calculado entre el dataset estimado y el dataset transformado

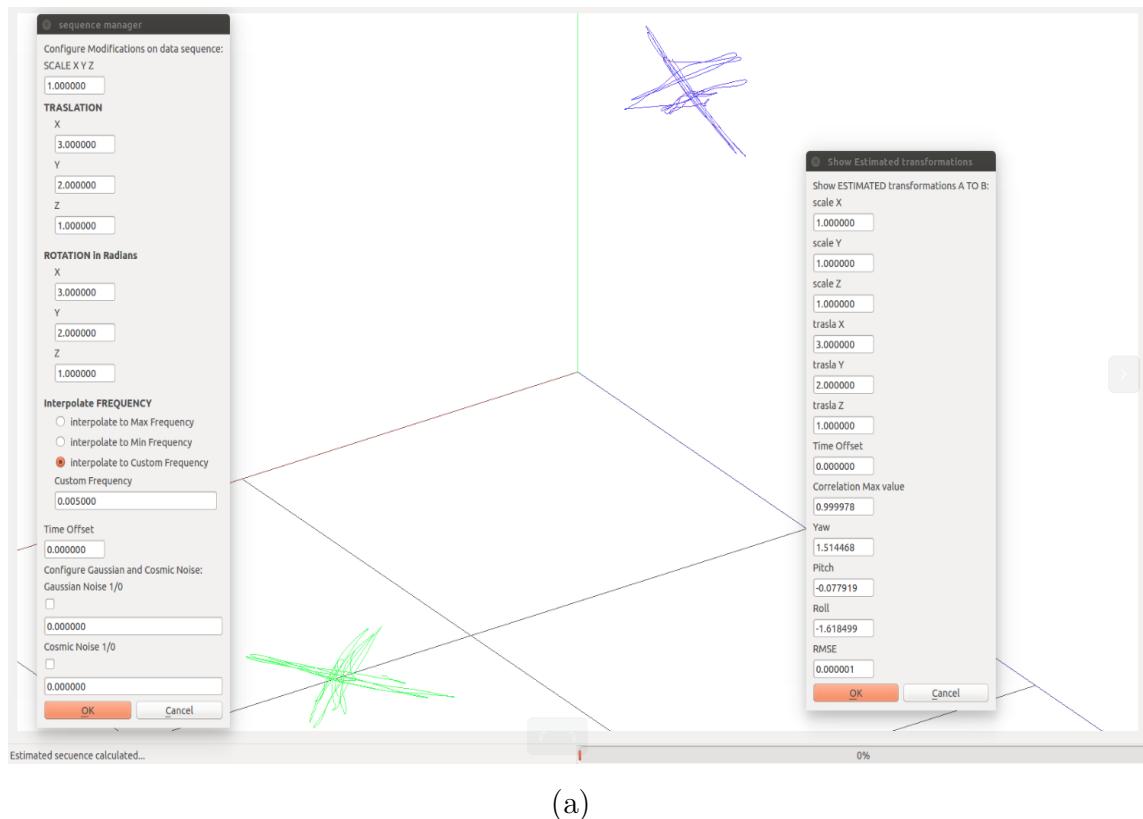
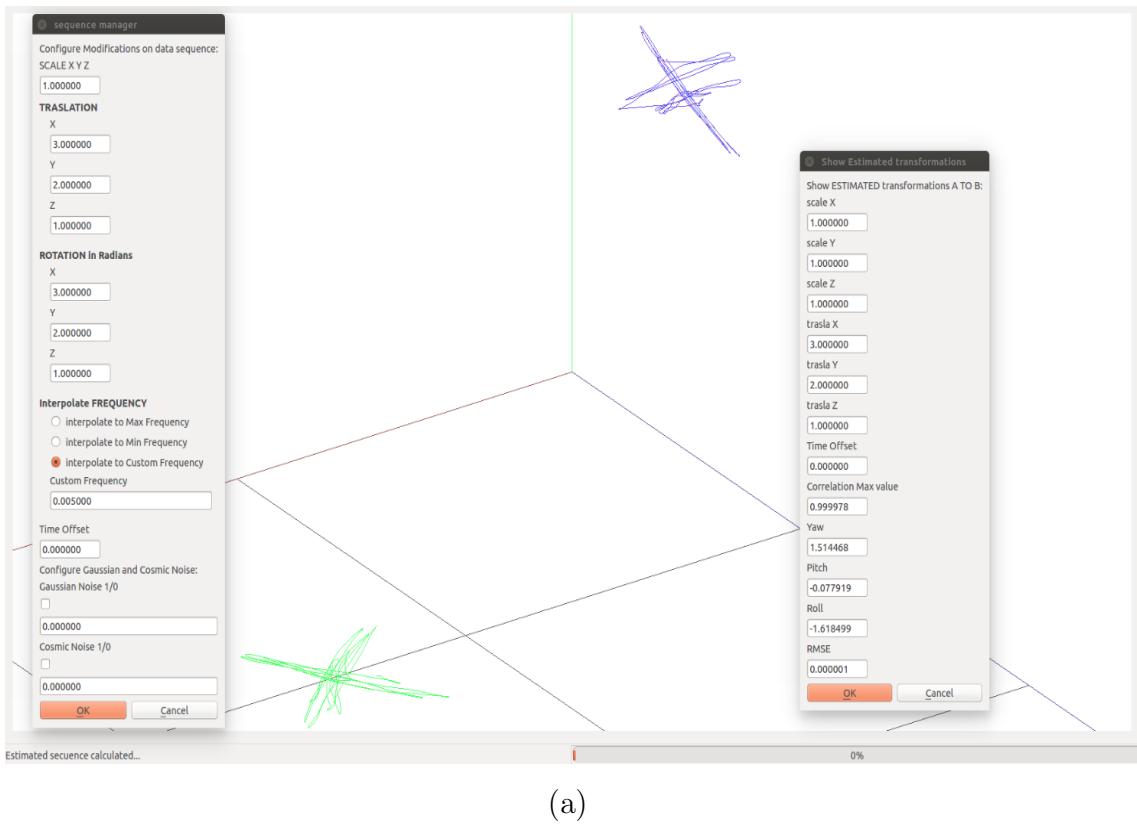


Figura 5.9: Gráfico que muestra los resultados de la estimación de un cambio de traslación y rotación .



(a)

Figura 5.10: Gráfico que muestra los resultados de la estimación de un cambio de translación y offset .

es 0.000001. Podríamos considerarlo como 0.0

En este caso, al dataset original (color verde) se le ha aplicado un par de transformaciones (en translación y con offset), dando lugar al dataset transformado (conjunto de puntos de color azul). En este caso tambien la estimación es buena , ya que el error RMSE da como resultado 0.0 y el cálculo del offset tambien coincide con el valor del offset que se ha aplicado en la transformación, salvo con signo negativo.

En la imagen superior, se ha aplicado sobre el dataset original un cambio de translación y además se le ha añadido ruido Gaussiano, como puede verse en el dataset resultante (color azul). Las estimaciones obtenidas son buenas, pero el error RMSE calculado entre el dataset transformado y el dataset estimando ya no es 0.0, sino 0.008. Esta falta de precisión puede apreciarse a simple vista en el interfaz gráfico, ya que como el dataset estimado carece de ruido gaussiano , los puntos del dataset estimado no coinciden con los puntos del dataset transformado, y por tanto ahora si podemos ver buena parte de los puntos del dataset estimado (color rojo).

En este caso, se ha aplicado sobre el dataset original un cambio de translación más

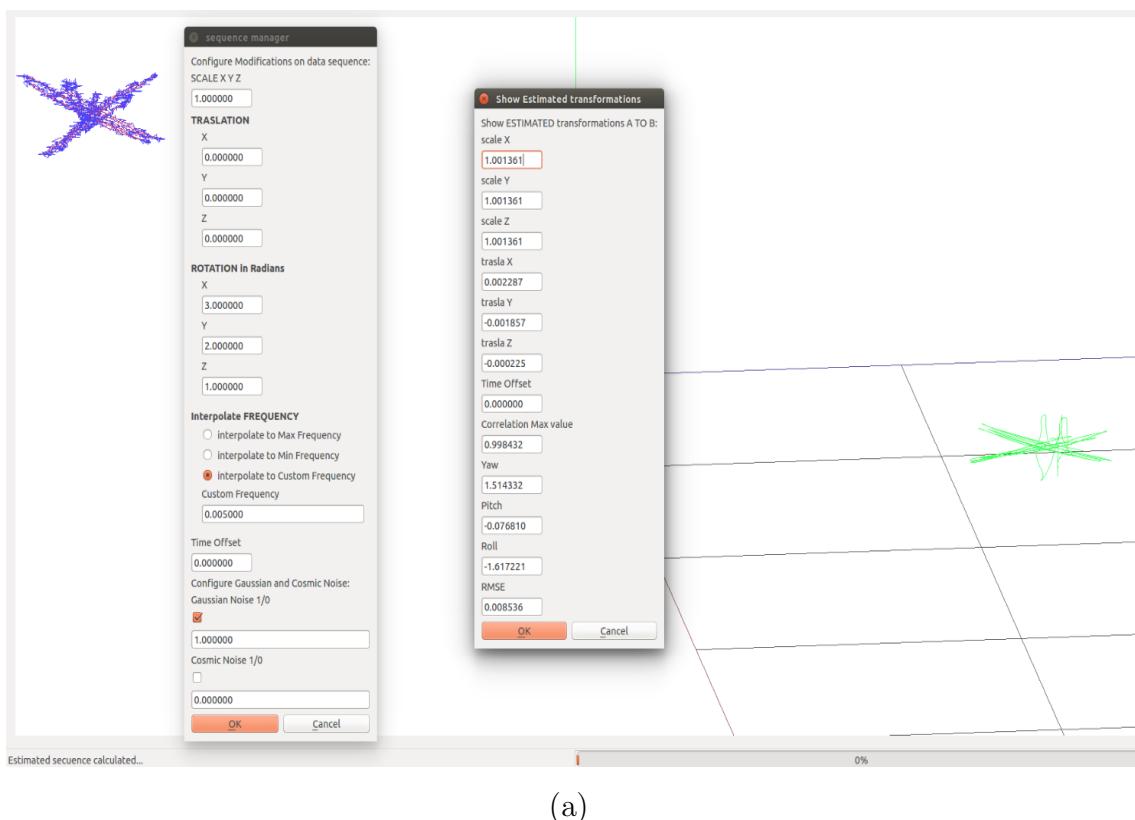


Figura 5.11: Gráfico que muestra los resultados de la estimación de un cambio de traslación y ruido gaussiano.

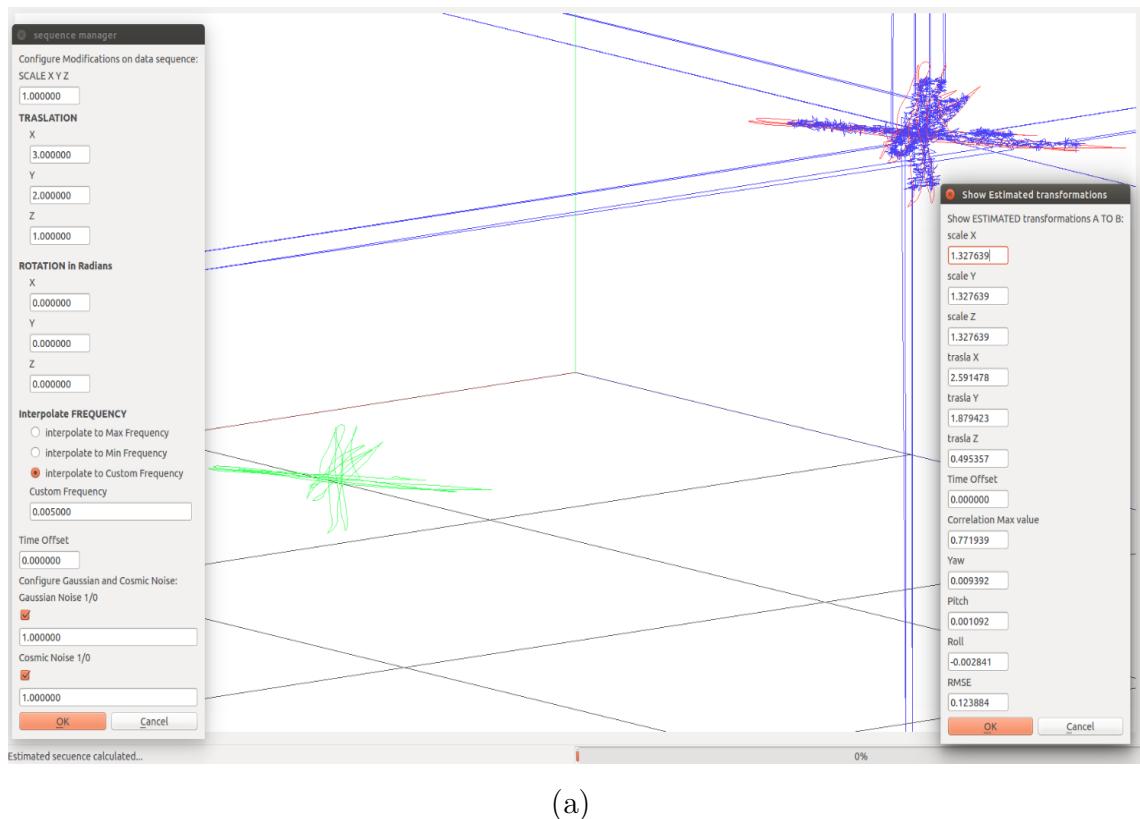
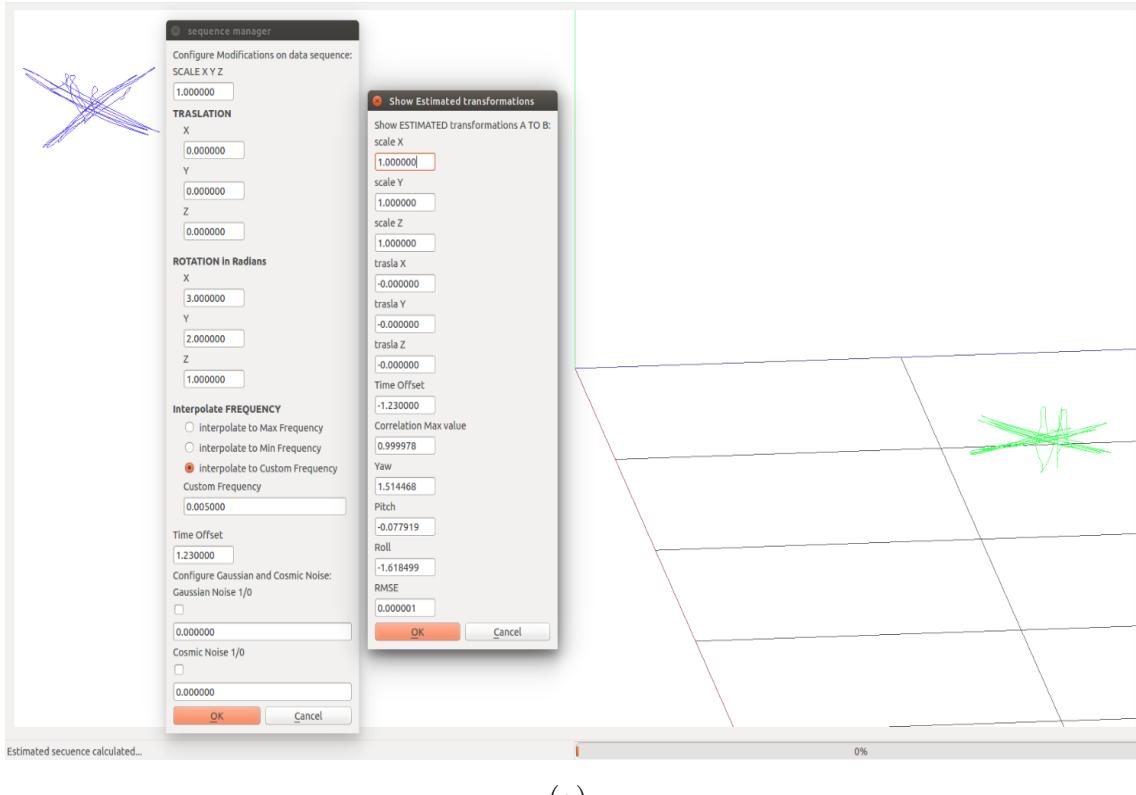


Figura 5.12: Gráfico que muestra los resultados de la estimación de un cambio de traslación y ruido cósmico y gaussiano.



(a)

Figura 5.13: Gráfico que muestra los resultados de la estimación de un cambio de rotación y offset.

ruido gaussiano y cósmico. Al tener ahora ruido cósmico, en algunos puntos del dataset transformado aparecen unos valores muy superiores al rango de valores del dataset , estos valores desorbitados harán que aumente notablemente el error entre el dataset transformado y el dataset estimado. Para esta experimento , el valor RMSE es de 0.12. Aunque puede observarse que aún así, la posición de los puntos del dataset estimado coincide con la posición de los puntos del dataset transformado.

En este caso, al dataset original (color verde) se le ha aplicado un par de transformaciones (en rotación y con offset temporal), dando lugar al dataset transformado (conjunto de puntos de color azul). En este caso tambien la estimación es buena , ya que el error RMSE da como resultado 0.0 y el cálculo del offset tambien coincide con el valor del offset que se ha aplicado en la transformación, salvo con signo negativo.

En la imagen superior, se ha aplicado sobre el dataset original un cambio de rotación y además se le ha añadido ruido Gaussiano, como puede verse en el dataset resultante (color azul). Las estimaciones obtenidas son buenas, pero el error RMSE calculado entre el dataset transformado y el dataset estimado ya no es 0.0, sino 0.008. Esta falta de precisión

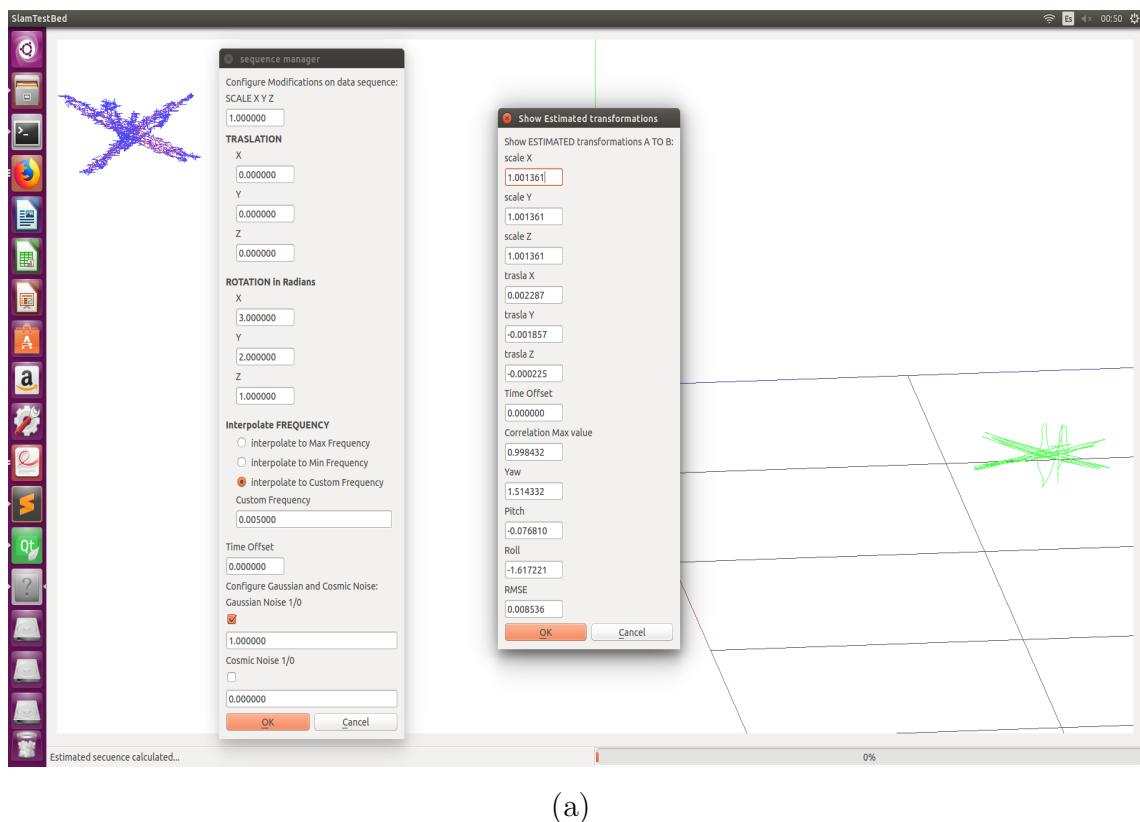
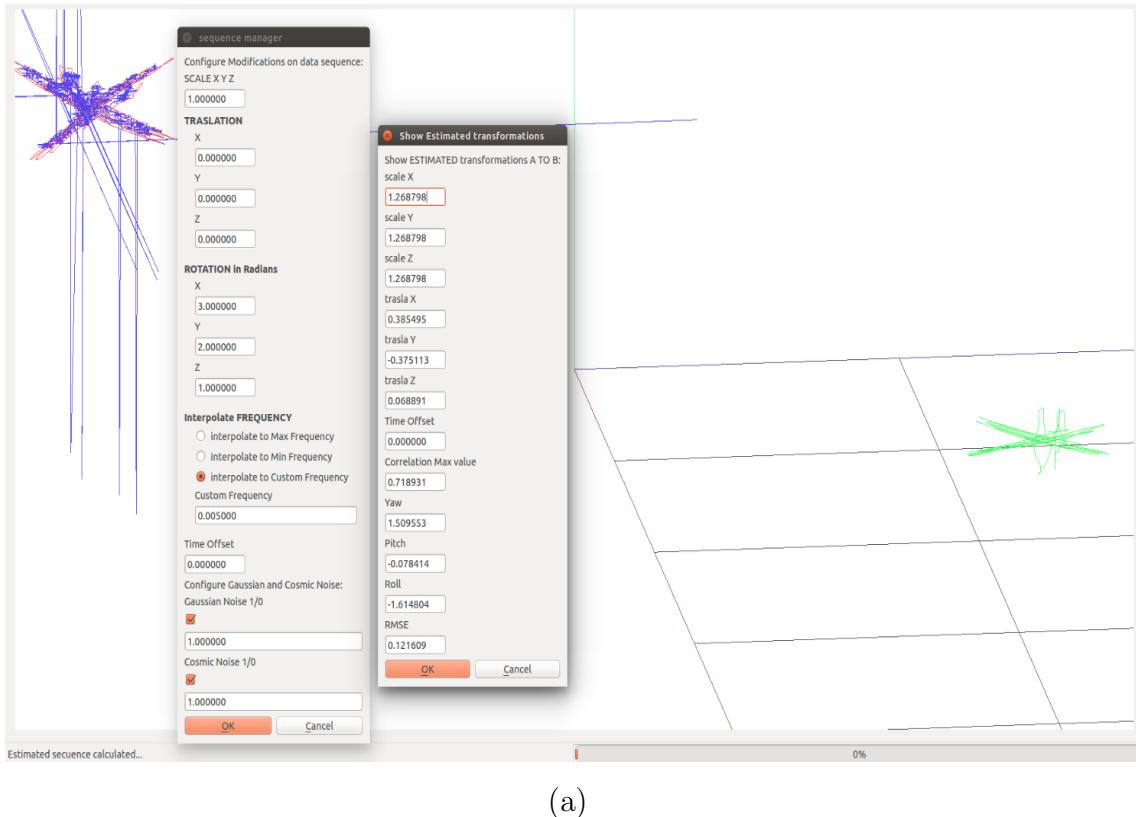


Figura 5.14: Gráfico que muestra los resultados de la estimación de un cambio de rotación y ruido gaussiano.

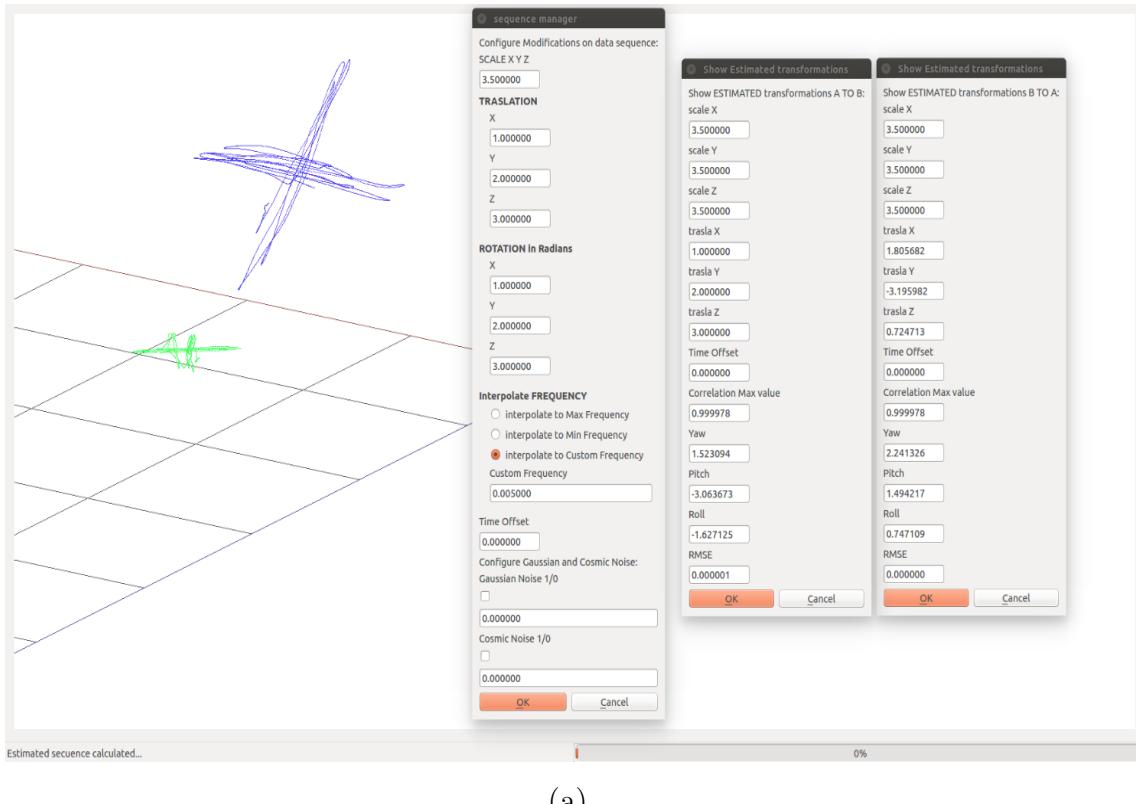


(a)

Figura 5.15: Gráfico que muestra los resultados de la estimación de un cambio de rotación y ruido cósmico y gaussiano.

puede apreciarse a simple vista en el interfaz gráfico, ya que como el dataset estimado carece de ruido gaussiano , los puntos del dataset estimado no coinciden con los puntos del dataset transformado, y por tanto ahora si podemos ver buena parte de los puntos del dataset estimado (color rojo).

En este caso, se ha aplicado sobre el dataset original un cambio de rotación más ruido gaussiano y cósmico. Al tener ahora ruido cósmico, en algunos puntos del dataset transformado aparecen unos valores muy superiores al rango de valores del dataset , estos valores desorbitados harán que aumente notablemente el error entre el dataset transformado y el dataset estimado. Para esta experimento, el valor RMSE es de 0.12. Aunque puede observarse que aún así, la posición de los puntos del dataset estimado coincide con la posición de los puntos del dataset transformado.



(a)

Figura 5.16: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

5.4. Pruebas de transformaciones combinadas

En este apartado, mostraremos imágenes del módulo GUI, donde se han realizado varias pruebas de transformaciones combinadas y se han estimado dichas transformaciones. En todas las transformaciones se ha realizado la interpolación de frecuencias al valor 0.05 para ambos datasets.

En la imagen superior se realiza la estimación de las transformaciones necesarias para pasar del datasetA al datasetB . En este caso se ha aplicado una transformación combinada de Escala, Traslación y Rotación. En verde puede verse el dataset original y en azul el dataset transformado, en rojo se aprecia el dataset estimado. Se puede ver gráficamente como el dataset estimado se ajusta con gran precisión al dataset transformado (azul). Por otra parte el RMSE entre el dataset transformado y el dataset estimado es de 0.0 Además las transformaciones se han calculado en los 2 sentidos, desde el dataSet A al dataSet B y desde el dataSetB al dataSet A.

Arriba en la imagen , se presenta el dataset original en verde, el dataset transformado

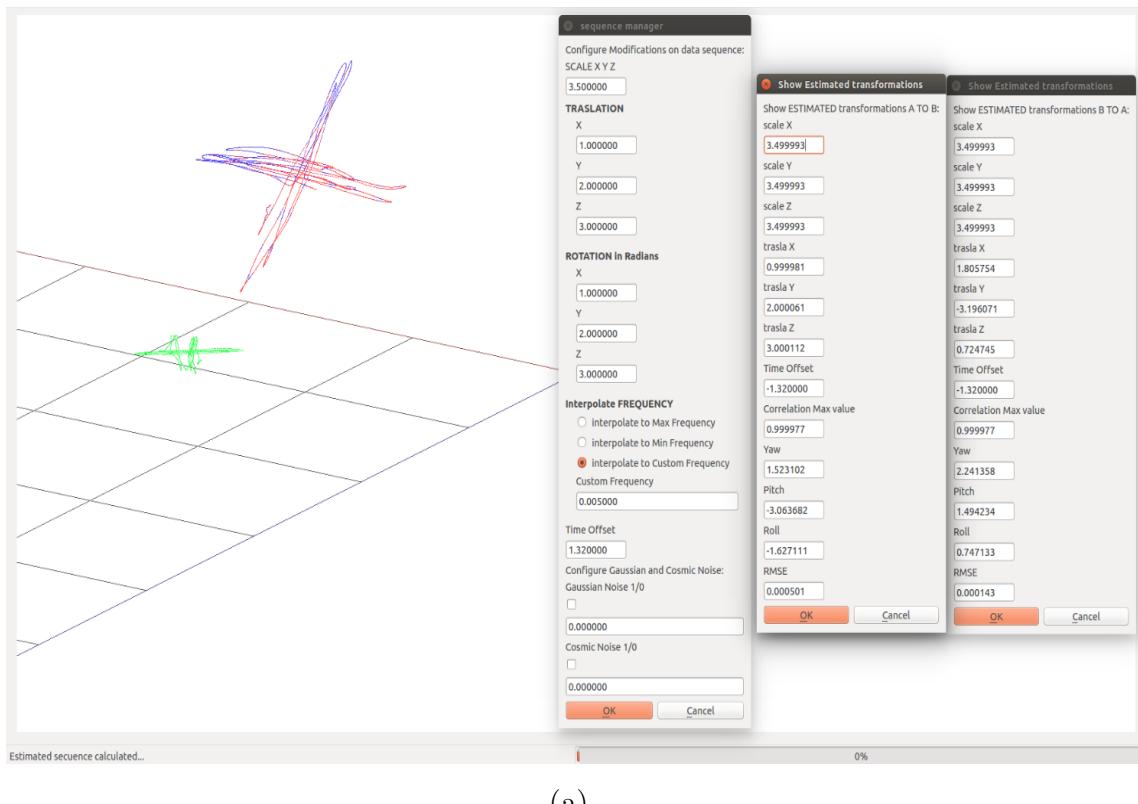
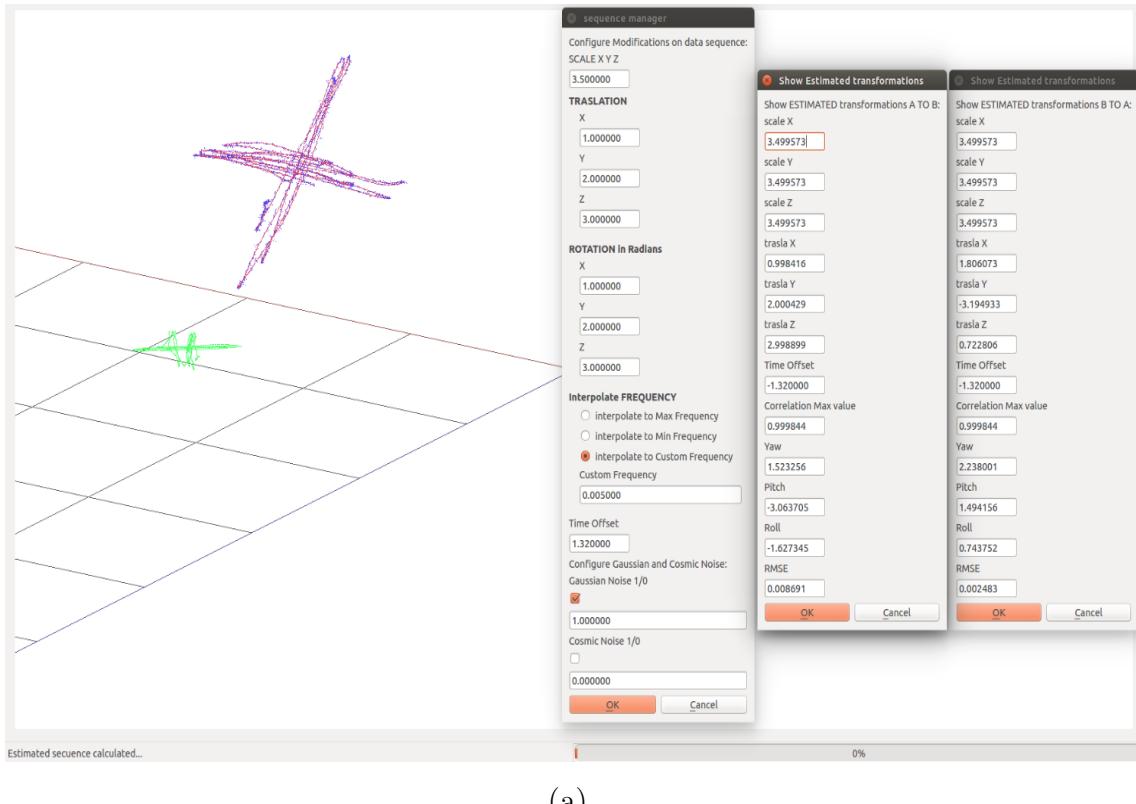


Figura 5.17: Gráfico que muestra los resultados de la estimación de un cambio de escala, traslación, rotación y offset .



(a)

Figura 5.18: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

en azul y el dataset estimado en rojo. En este caso se ha realizado la siguiente combinación de transformaciones, Escala, Traslación, Rotación y Offset. Como se puede observar en los resultados de las estimaciones, el offset es estimado con total exactitud. Sin embargo el valor del RMSE ya no es de 0.0, si no de 0.0005, aún así el error cometido sigue siendo muy bajo. Visualmente, también se puede observar que la precisión ha disminuido , ya que se aprecian más puntos rojos (dataset estimado) en el gráfico. Esto indica que la estimación sigue siendo muy buena aunque la precisión ha bajado.

En la imagen superior, podemos ver el dataset original en verde, el dataset transformado en azul y el dataset estimado en rojo. Las transformaciones realizadas en este caso han sido, Escala, Traslación, Rotación, Offset y Ruido Gaussiano. En este caso, al incluir ruido Gaussiano en la transformación del dataset original, el dataset estimado pierde precisión, y el RMSE obtenido es de 0.008. Sin embargo, la precisión aunque es menor, continúa siendo buena, y en el gráfico puede verse , como el dataset estimado concuerda de manera aceptable con el dataset estimado. Observese que en el dataset estimado no hay ruido gaussiano.

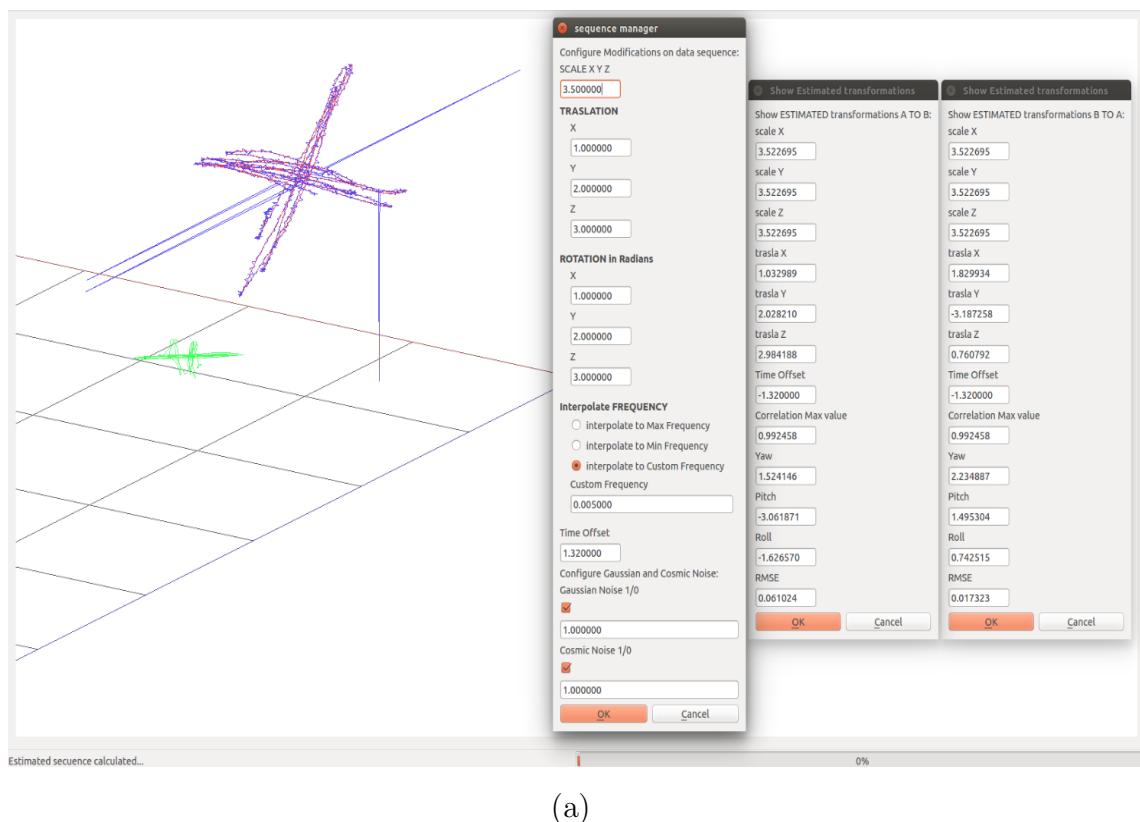
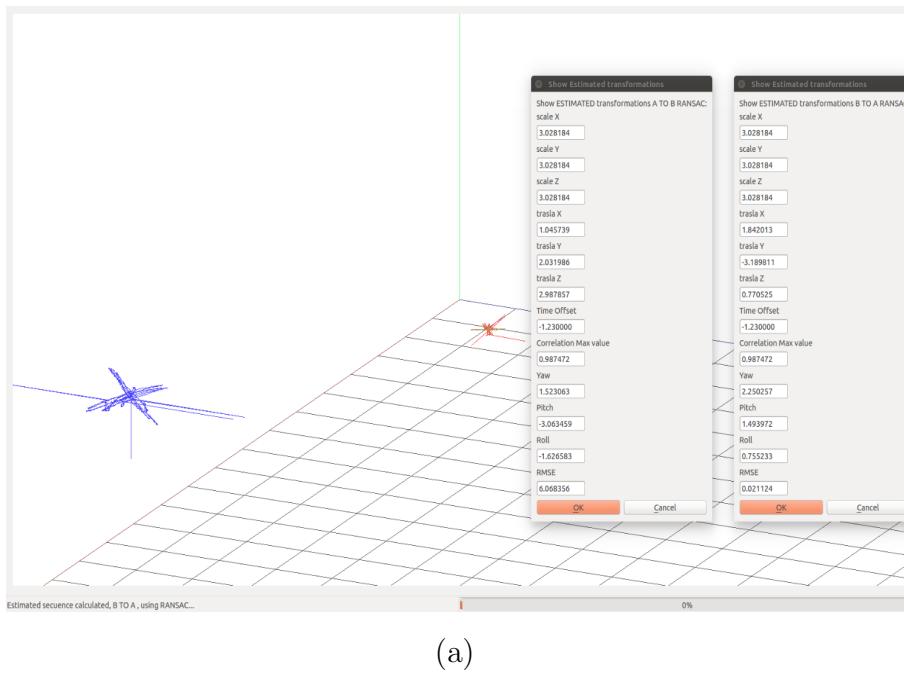


Figura 5.19: Gráfico que muestra los resultados de la estimación de un cambio de escala y traslación .

Arriba podemos ver una nueva captura de pantalla, donde se han realizado las siguientes transformaciones, Escala, Traslación, Rotación, Offset, Ruido Gaussiano y Ruido Cósmico. Como se puede observar, en las estimaciones del dataset A (verde) al dataset B (azul), la precisión ha bajado notablemente, con un valor RMSE del 0.06. Esta precisión tan baja se debe a la inclusión de ruido Cósmico, que altera notablemente los valores de X, Y , Z de algunos puntos 3D. Aún así, como puede verse en la representación gráfica del dataset Estimado (rojo) sobre el dataset Transformado (azul) se aproxima bastante a la solución óptima.



(a)

Figura 5.20: Gráfico que muestra los resultados de la estimación de la transformación del datasetA en el datasetB y viceversa, utilizando RANSAC.

En captura de pantalla superior, podremos ver una nueva transformación donde se ha aplicado transformaciones de Escala, Traslación, Rotación, Offset, Ruido Gaussiano y Cósmico y además se ha utilizado el método RANSAC para calcular el error.

Capítulo 6

Conclusiones

Se ha diseñado y programado una aplicación, llamada SLAMTestbed, que estima las relaciones entre dos secuencias de puntos 3D orientados, calculando tanto sus relaciones espaciales (qué traslación, rotación y escala hay entre ellas) como sus relaciones temporales (qué desfase temporal existe entre ellas).

Esto lo consigue incluso con dos secuencias de puntos 3D que no están expresadas en el mismo sistema de referencia espacial, ni con la misma escala, ni con la misma frecuencia de muestras, ni con el mismo origen de tiempos. Para ello, incluye varios bloques específicos de estimación encadenando diferentes estimadores, análisis matemáticos y módulos de registro.

Esta aplicación permite medir objetiva y cuantitativamente el error en las estimaciones de un algoritmo de Visual SLAM cuando se introducen sus estimaciones y la secuencia de posiciones verdaderas en la herramienta SLAMTestbed. En ese caso las diferencias entre ambas secuencias y los estadísticos calculados reflejan la calidad y la exactitud de las estimaciones del algoritmo. Además de caracterizar sus calidad objetivamente permite por tanto compararlo con otros algoritmos de Visual SLAM o diferentes variantes del mismo algoritmo.

6.1. Conclusiones

El objetivo principal de este proyecto era ofrecer una herramienta software que permita medir cuantitativamente el rendimiento de los algoritmos Visual SLAM y la calidad de sus estimaciones. Este objetivo se ha logrado con el diseño e implementación de un Registrador Temporal (subobjetivo 1) y otro Registrador Espacial (subobjetivo 2). Para crear el Registrador Temporal ha sido necesario desarrollar un módulo que calcule

el desplazamiento de tiempo entre dos secuencias (capítulo 4.3) y tambien un módulo interpolador (capítulo 4.4) capaz de sincronizar en frecuencia los dos *datasets* de entrada. Para validar el correcto funcionamiento de los Registradores Temporal y Espacial, se ha creado un módulo Transformador (subobjetivo 3) tal y como se ha comentado en el capítulo 5.1. Para validar experimentalmente estos dos módulos de registro se han hecho múltiples pruebas de distinta complejidad como se describe en el capítulo 5.2, 5.3 y 5.4.

La herramienta SLAMTestbed incorpora además un interfaz gráfico que permite al usuario visualizar en 3D las secuencias de datos así como lanzar de una forma tan sencilla como hacer click con el ratón varias operaciones sobre los datasets y visualizar los resultados (capítulo 4.7). Se debe tener en cuenta que no todas las herramientas de algoritmos SLAM que existen actualmente incorporan este interfaz gráfico, como por ejemplo TUM (capítulo 3.2), que a día de hoy sólo permite lanzar comandos desde un terminal. Esto es una ventaja para SLAMTestbed, ya que, como se ha demostrado en las pruebas, este interfaz es útil para verificar con un simple vistazo si la estimación es correcta o por el contraio se aleja de la verdad absoluta.

6.2. Líneas Futuras

En este subapartado describiremos en que líneas se debería trabajar o profundizar más en el futuro en la herramienta SLAMTestbed:

1. Uso de SLAMTestbed para comparar diferentes algoritmos del estado del arte de Visual SLAM utilizando datasets internacionales de referencia que incluyan verdad absoluta.
2. Utilización de SLAMTestbed en el TFM de Javier Martinez para medir si la introducción de sensor inercial mejora o no la calidad de las estimaciones de posición en el algoritmo SD-slam.
3. Incorporar más estadísticas y medidas de error que ayuden a evaluar la precisión y robustez del algoritmo, como por ejemplo el porcentaje de tiempo en que la diferencia entre trayectorias es menor que un umbral.
4. Escritura de un artículo científico que presente esta herramienta a la comunidad investigadora.

5. Nuevas opciones y mejoras en el interfaz de usuario para modificar parámetros como la distribución del ruido gaussiano, valores límite para el ruido cósmico, o utilización de varios hilos de proceso para que el interfaz no se quede bloqueado mientras realiza cálculos.
6. Añadir nuevos algoritmos en el módulo de Registro, como Horn, Horn + RANSAC y posibilidad de que la herramienta seleccione automáticamente el algoritmo de registro que mejores resultados obtenga.
7. Portabilidad de la herramienta a otros lenguajes de programación (python,Java) o sistemas operativos (Android)

Todo el código de la herramienta SLAMTestbed está publicado en el repositorio GitHub, lo cual abre su desarrollo y mantenimiento a la comunidad de usuarios e investigadores que quieran utilizar esta herramienta.

Bibliografía

- [Arribas, 2016] Victor Arribas. Análisis de algoritmos de visualslam: un entorno integral para su evaluación. *Trabajo Fin de Máster. Universidad Rey Juan Carlos*, 2016.
- [Bodin *et al.*, 2018] Bruno Bodin, Harry Wagstaff, Sajad Saeedi, Luigi Nardi, Emanuele Vespa, John H Mayer, Andy Nisbet, Mikel Luján, Steve Furber, Andrew J Davison, Paul H.J. Kelly, and Michael O’Boyle. Slambench2: Multi-objective head-to-head benchmarking for visual slam. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2018.
- [Civera *et al.*, 2010] 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.
- [Davison *et al.*, 2007] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6), 2007.
- [Engel *et al.*, 2014] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.
- [Engel *et al.*, 2016] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *arXiv preprint arXiv:1607.02565*, 2016.
- [Fischler and Bolles, 1981] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [Forster *et al.*, 2017] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017.

- [Gálvez-López and Tardos, 2012] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.
- [Geiger *et al.*, 2012] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [Hernández, 2014] Alejandro Hernández. Autolocalización visual aplicada a la realidad aumentada. *Trabajo Fin de Máster. Universidad Rey Juan Carlos*, 2014.
- [Jakob *et al.*, 2016] Engel Jakob, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Septiembre 2016.
- [Klein and Murray, 2007] 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.
- [Konolige and Agrawal, 2008] K. Konolige and M. Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, Oct 2008.
- [Mur-Artal *et al.*, 2015] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.
- [Newcombe *et al.*, 2011] 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.
- [Perdices García, 2017] Eduardo Perdices García. Técnicas para la localización visual robusta de robots en tiempo real con y sin mapas. *Tesis Doctoral. Universidad Rey Juan Carlos*, 2017.
- [Stiller *et al.*, 2008] Christoph Stiller, Sören Kammler, Benjamin Pitzer, Julius Ziegler, Moritz Werling, Tobias Gindel, and Daniel Jagszent. Team annieway’s autonomous system. pages 248–259, 02 2008.

- [Sturm *et al.*, 2012] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.
- [Takafumi Taketomi, 2017] Sei Ikeda Takafumi Taketomi, Hideaki Uchiyama. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, Junio 2017.
- [Trajkovic and Hedley, 1998] Miroslav Trajkovic and Mark Hedley. Fast corner detection. *Image and Vision Computing*, 16:75–87, 02 1998.
- [Zhang and Scaramuzza, 2018] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.