

**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, RGBD como las utilizadas en el proyecto Tango 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.

El presente documento está dividido en 5 secciones y trata de describir el estado del arte de Visual SLAM. Esta primera sección o módulo es una introducción a Visual SLAM. El siguiente modulo será una descripción de las aplicaciones actuales de Visual SLAM en distintos dispositivos, desde robots aspiradora, pasando por drones y Smartphones de última generación. En el módulo 3 hablaremos de la problemática de Visual SLAM, cuales son las principales dificultades que debemos solventar a la hora de implementar un algoritmo de VisualSLAM. El punto 4 es el más extenso , contiene una breve descripción de los módulos principales que componen el algoritmo de Visual SLAM y un resumen de las técnicas actuales más conocidas de Visual SLAM. Y por último, el punto 5, donde se muestran las conclusiones.

Índice general

1. Introducción	1
1. Visión Artificial	2
2. Visual SLAM	5
2.1. Aplicaciones en VisualSLAM	6
2.2. Visual SLAM en Robótica Móvil	11
2.3. Conceptos	14
2.4. Problemas de Visual SLAM	15
2.5. Inicialización del Mapa:	15
2.6. Ambigüedad en la escala:	16
2.7. Dificultad para operar en entornos con pocas texturas:	16
2. Objetivos	17
1. Requisitos	18
3. Estado del Arte	20
3.1. MonoSLAM	20
3.2. ORB-SLAM	23
3.3. DSO y LDSO	24
4. Herramientas para realizar evaluaciones comparativas de algoritmos SLAM	28
5. Comparativa de los algoritmos más representativos	33

4. Herramienta SLAMTestBed	35
4.1. Diseño de Caja Negra	35
4.2. Diseño de caja Blanca	36
4.3. Estimador PCA y Cálculo de Escala	38
4.4. Registro Espacial	39
4.5. Estimador Offset	40
4.6. Interpolador	42
4.7. Cálculo de Estadísticas	43
4.8. GUI	43
5. Experimentos	46
5.1. Módulo Transformador	46
5.2. Pruebas de transformaciones individuales	47
5.3. Pruebas de transformaciones en parejas	51
5.4. Pruebas de transformaciones combinadas	62
6. Conclusiones	68
Bibliografía	70

Capítulo 1

Introducción

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, RGBD como las utilizadas en el proyecto Tango 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.

El presente documento está dividido en 5 secciones y trata de describir el estado del arte de Visual SLAM. Esta primera sección o módulo es una introducción a Visual SLAM. El siguiente modulo será una descripción de las aplicaciones actuales de Visual SLAM en distintos dispositivos, desde robots aspiradora, pasando por drones y Smartphones de última generación. En el módulo 3 hablaremos de la problemática de Visual SLAM, cuales son las principales dificultades que debemos solventar a la hora de implementar un algoritmo de VisualSLAM. El punto 4 es el más extenso , contiene una breve descripción de los módulos principales que componen el algoritmo de Visual SLAM y un resumen de las técnicas actuales más conocidas de Visual SLAM. Y por último, el punto 5, donde se muestran las conclusiones.

1. Visión Artificial

La Visión Artificial , es una rama científico técnica creada para extraer y procesar información a partir de imágenes, para ello genera sistemas que intentan emular el sentido de visión de los seres humanos .

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 poder 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.

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:

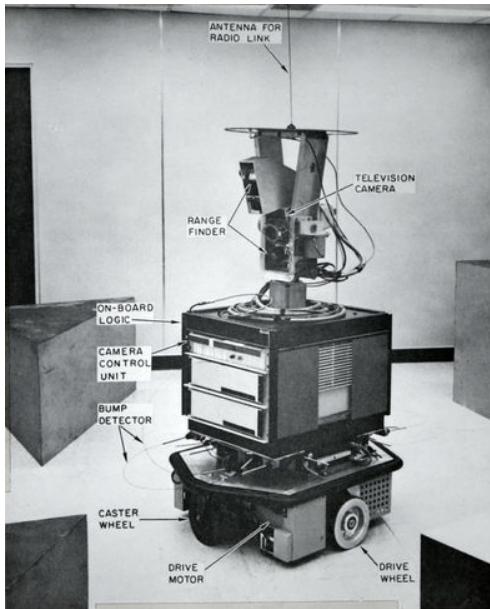
Reconocimiento de objetos: Se trata de buscar unas propiedades concretas de un determinado objeto (forma, color o cualquier otro patrón) en una imagen para determinar si un objeto se encuentra o no en ella. Por ejemplo mediante la obtención de píxeles característicos que destaque en la imagen o utilizándose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

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

- 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:

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

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

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.

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. Una de las aplicaciones más populares para la visión artificial es el reconocimiento de caracteres (OCR). Es 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.

2. Visual SLAM

Visual Simultaneous Localization And Mapping (Localización y Mapeo Visual Simultáneo), no se refiere a un algoritmo en particular o a una aplicación software. 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.

Hay varios tipos de tecnología SLAM, algunos no necesitan ni siquiera 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 una 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, al contrario de otros tipos de tecnología SLAM.

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.

Visual SLAM es todavía una tecnología emergente, pero con muchísimo potencial. Será 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 VisualSLAM 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

2.1. Aplicaciones en VisualSLAM

Hoy en día VisualSLAM ya tiene muchas aplicaciones y aún más que están por llegar en un futuro próximo, 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 1.2(c)). Los sensores del smartphone son capaces de tomar más de 250 millones de medidas 3D por segundo y con estos datos pueden construir un modelo 3D de los alrededores del teléfono. 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.



Figura 1.2: 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).

Al ser una tecnología nueva aún no hay un elevado número de dispositivos que lo soporten. De momento existen 2 móviles compatibles con Tango¹, el Lenovo Phab 2 pro (Figura 1.2(a)) y el Asus Zenfone AR (Figura 1.2(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.

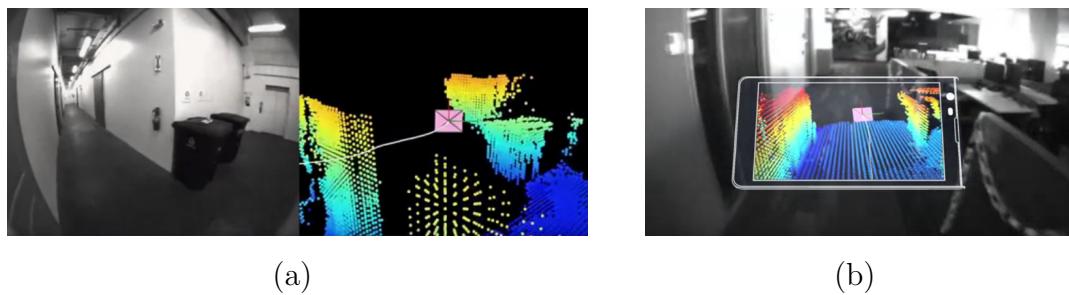


Figura 1.3: Generación de mapa 1 (d). Generación de mapa 2 (e)

2. **Magic Plan** Magic Plan es una aplicación que permite de forma interactiva obtener planos de habitaciones o del interior de un edificio, utilizando para ello la cámara

¹<https://get.google.com/tango/>

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 (Figura ??) sin necesidad de medir, dibujar, mover muebles y sin necesidad de ser un experto. La aplicación utiliza técnicas de VisualSLAM 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 1.4(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 1.4(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 (Figura ??) o también puede generar el mapa mientras lo teledirigimos.

²<https://pix4d.com/>

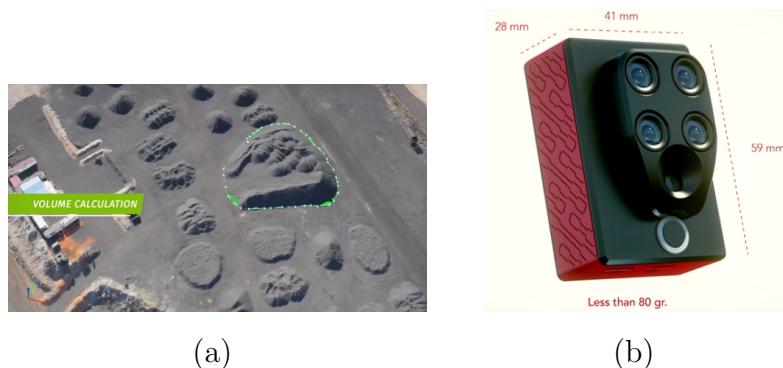
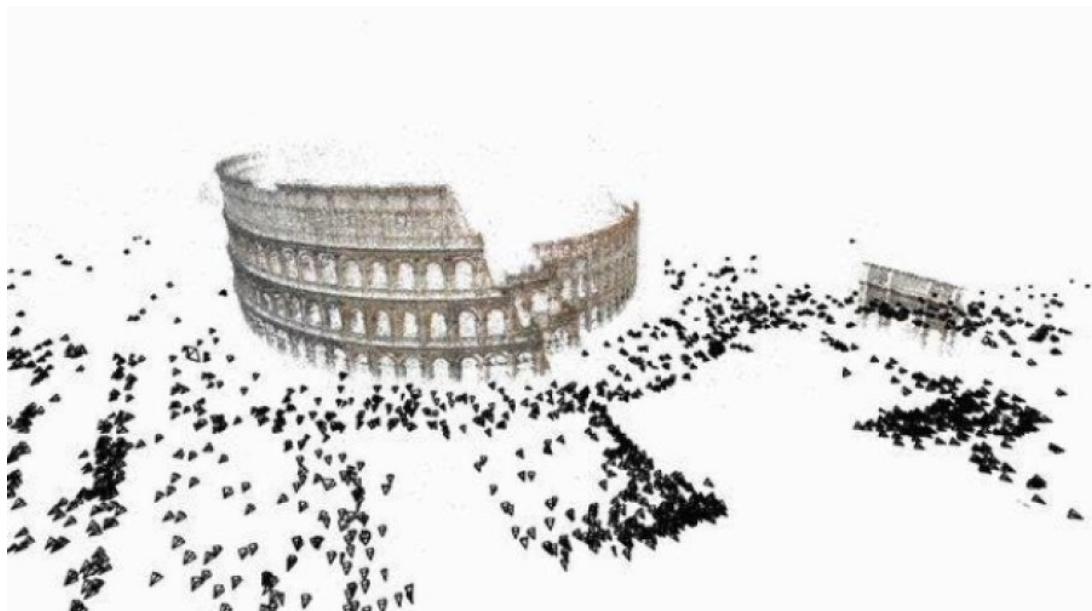


Figura 1.4: Pix4D cálculo de volumen(a). Cámara multiespectral Parrot Sequoia (b)

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 1.5(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 VisualSLAM, 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** Canvas ³ 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 ⁴ alrededor de la habitación y podremos ver como el mapa 3D comienza a generarse en tiempo real. El sensor (Figura 1.6(a)) toma miles

³<https://canvas.io/>

⁴<https://structure.io/>



(a)

Figura 1.5: Recreación del Coliseo de Roma generado con Photo Tourism.

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 1.6: El sensor de profundidad Structure para Ipad.

2.2. Visual SLAM en Robótica Móvil

Visual SLAM tiene aplicaciones directas en robótica. Un ejemplo podría ser el Robot Gita de Piaggio.

1. El robot Gita: Este novedoso robot (Figura 1.7) tiene incorporadas varios pares de cámaras estéreo, en la parte trasera y delantera. Con las imágenes captadas por estas cámaras se puede realizar VisualSLAM, además es capaz de seguir a su dueño siempre y cuando el humano lleve un cinturón con otras 2 cámaras estéreo, esta funcionalidad se consigue comparando el SLAM del robot con el SLAM captado por el cinturón. El robot dispone de un compartimento interior o maletero y tiene suficiente potencia como para poder transportar hasta 20 Kg. Podría ser de gran utilidad a la hora de ir al supermercado, ya que el robot nos seguirá transportando la compra en su interior, no necesitaremos el típico carrito, incluso nos permitiría ir al centro comercial en bicicleta.

Otra utilidad sería en el interior de un hotel, el robot podría realizar las funciones de camarero y hacer servicio de habitaciones transportando la comida directamente a las habitaciones del hotel. También podría ser un estupendo ayudante para un mecánico, ya que podría transportar la pesadas herramientas o piezas ⁵.



(a)



(b)

Figura 1.7: El cinturón con cámaras estéreo (a) La capacidad de carga del robot Gita (b)

2. Reconocimiento de Objetos: Otra utilidad de SLAM es que mejoran la capacidad de los robots móviles a la hora de reconocer objetos. Los sistemas de reconocimiento

⁵<http://spectrum.ieee.org/automaton/robotics/home-robots/piaggio-cargo-robot>

de objetos utilizarán la información proporcionada por SLAM para mejorar su capacidad de reconocimiento. La capacidad de reconocimiento será muy útil para aquellos robots que tengan que manipular objetos en su entorno. Con SLAM, los sistemas de reconocimiento pueden tomar como entradas varias imágenes desde distintos puntos de vista, por lo tanto el reconocimiento resulta más sencillo que si tuviesen tan sólo una imagen estática ⁶.

3. **Robot Aspirador:** Recientemente ha entrado en los hogares el uso de VisualSLAM 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))⁷.
- Aspirador Roomba 966 (Figura 1.8(b))⁸.
- Aspirador LG-Hombot (Figura ??)⁹.

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.

⁶<http://www.roboticsproceedings.org/rss11/p34.pdf>

⁷<http://www.dyson.com>

⁸<http://www.irobot.es/robots-domesticos/aspiracion>

⁹<http://www.lg.com/es/aspiradoras/lg-VR64702LVMT>



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

4. **Drones:** Por último no podemos olvidar los drones, robots voladores equipados con cámara que también pueden obtener mapas de su entorno con VisualSLAM. 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 ¹⁰.

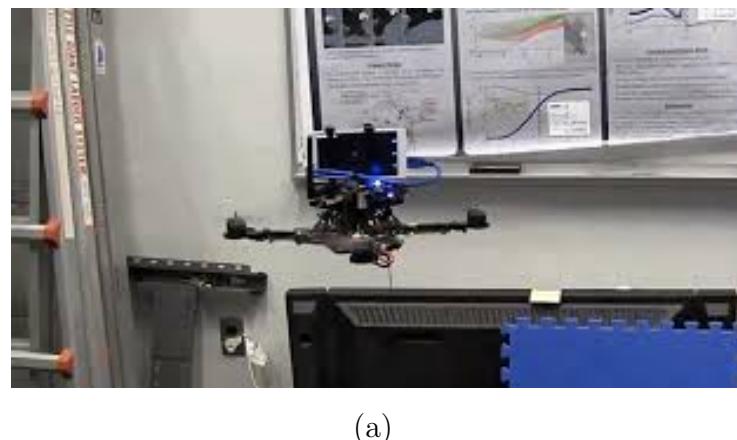


Figura 1.9: Dron equipado con dispositivo compatible con Tango

¹⁰<http://spectrum.ieee.org/automaton/robotics/drones/autonomous-quadrotor-flight-based-on-google-project-tango>

2.3. 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 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)

Hipótesis múltiple: Los algoritmos pueden manejar al mismo tiempo múltiples hipótesis candidatas como solución a la localización. Esto se producirá con frecuencia en aquellos entornos donde aparezcan simetrías o zonas que parezcan visualmente similares para el algoritmo.

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.

Dinamismo de la escena: El movimiento de los objetos en la escena suele interferir con las estimaciones de autolocalización visual. No todos los desplazamientos en las imágenes se deben entonces al movimiento propio del robot.

Cierre de bucle: Se produce cuando el robot vuelve 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.

2.4. Problemas de 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:

2.5. 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 VisualSLAM 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 MonoSLAM por ejemplo se utilizan al menos 4 puntos 3D como objetos de referencia, y la forma del objeto se usa para mejorar el mapa.

2.6. 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. 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.

2.7. 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 VisualSLAM [Takafumi Taketomi, 2017].

Capítulo 2

Objetivos

En el siguiente apartado se detallaran los objetivos que se pretenden alcanzar en este Trabajo Fin de Máster.

Desde la existencia de Visual SLAM , se están desarrollando algoritmos que permitan a los robots estimar su posición , para poder generar un mapa en 3D del entorno y así poder navegar por el espacio que le rodea. 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, por tanto el objetivo de este proyecto es presentar un conjunto de herramientas que permitan comparar el rendimiento de los algoritmos visual SLAM.

Esta herramienta permitirá comparar el rendimiento de los nuevos algoritmos de visual SLAM así como estudiar adaptaciones y mejoras en los algoritmos ya existentes.

En el marco de este TFM nos hemos centrado en una plataforma que permite 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, por lo tanto en los algoritmos de VisualSLAM se 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.

Objetivos principales: Por tanto como objetivo principal será crear una herramienta

para poder comparar 2 datasets de Visual SLAM para ello deberemos conseguir un registrador Espacial que permita registrar 2 datasets en el espacio, estimando rotación, traslación y escala para poder realizar el registro espacial entre 2 datasets

un Registrador Temporal: Con este registrador conseguiremos sincronizar los 2 datasets. Para ello deberemos ajustar los 2 datasets a la misma frecuencia de muestreo, (utilizaremos interpolación) y también calcular el offset entre los 2 datasets. El offset temporal sería la diferencia de tiempo entre 2 secuencias, Ejemplo: De una forma sencilla, si la secuencia t1 es 2,4,8 y la secuencia t2 es 3,5,7, el offset entre las 2 secuencias sería de 1 segundo.

Como objetivo secundario: Validación con Experimentos: Para comprobar que los 2 módulos funcionan correctamente, realizaremos experimentos donde mediremos el error entre el groundtruth y el nuevo dataset estimado.

Modulo de Transformación: A modo de ayuda , para poder realizar pruebas más ágiles contando con un sólo dataset, la herramienta contará con un módulo de transformación que nos permitirá realizar tanto alteraciones temporales como espaciales en un dataset, dando como resultado un nuevo dataset.

1. Requisitos

Los requisitos principales de este TFM han sido:

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

R2. Se restringirá solo a los algoritmos VisualSLAM que empleen una sola cámara.

R3. La herramienta debe ser extensible a los resultados de nuevos algoritmos de VisualSLAM.

R4. Facilidad de uso.

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

R8. Debe ofrecer resultados fácilmente legibles y reconocibles para el usuario.

2.3. Metodología y plan de trabajo

En cuanto a la metodología utilizada para desarrollar la aplicación, se ha seguido el modelo de ciclo de vida en espiral. Esto es una de las primeras lecciones que se aprenden

de las metodologías ágiles, la metodología debe adaptarse al proyecto y no al revés. Dicha premisa cobra más importancia en un proyecto realizado por una sola persona.

Por consiguiente, y enmarcándolo dentro de la metodología en espiral, primero se ha realizado un estudio previo muy amplio que sirve para obtener una visión completa del problema y detectar puntos críticos, y luego se ha ido acotando hasta llegar al desarrollo principal, el cual ha sido revisado y validado en cada iteración. Esta amplitud inicial es importante ya que no sólo nos permite avanzar en todas las vías en paralelo, sino porque ofrece una prueba de concepto para las ramificaciones que se han paralizado en favor del desarrollo troncal.

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

<https://github.com/FALTAURL>

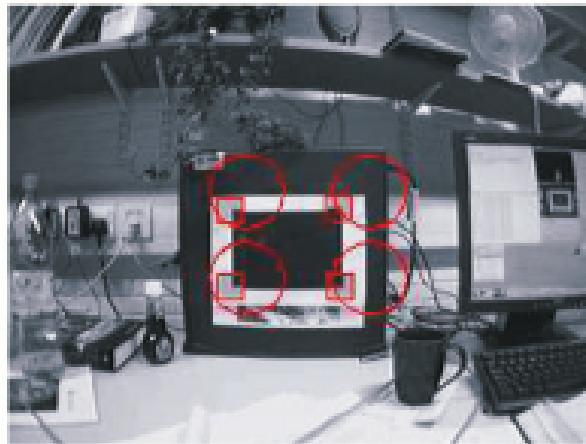
Capítulo 3

Estado del Arte

En esta sección explicaremos varios de los algoritmos SLAM más significativos , como MonoSLAM, PTAM, DTAM, SVO,LSD-SLAM,ORB-SLAM,DSO,SDVL y RGBD SLAM. Tambien 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 , también describiremos la herramienta SLAMBENCH y por último daremos algunos detalles sobre The Kitti Vision Benchmark Suite.

3.1. 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 en el año 2002 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.



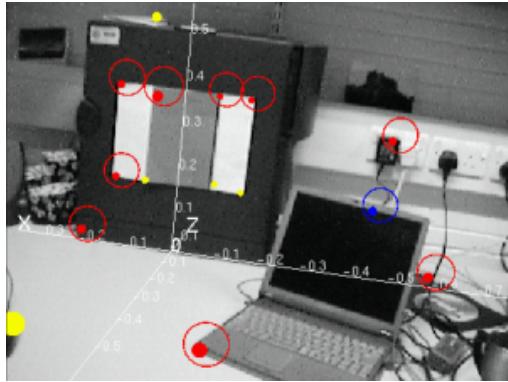
(a)

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 y además 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 por tanto en intervalos de tiempo tan pequeños, el sistema puede aproximarse a un sistema lineal. Cuantas más iteraciones o frecuencia de muestreo la estimación mejora. En cada iteración se hace una detección de puntos de interés (esquinas con FAST) en la imagen actual de entrada, y obtendremos una serie de puntos que serán candidatos a ser el vector observación de los puntos que queremos seguir. Estos candidatos deberán ser filtrados, pues alguno puede ser un falsa esquina. 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, estamos optimizando el computo 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. MonoSLAM es recomendable para mapas con pocos puntos. Es muy sensible a movimientos bruscos y por tanto difícilmente podrá recuperarse de un secuestro. Si la hipótesis de partida no es correcta el filtro podría desestabilizarse y no llegar nunca a aproximar razonablemente el vector de estado.



(a)

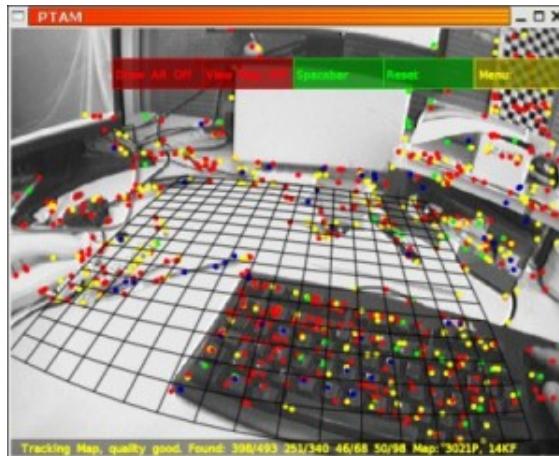
Figura 3.2: Ejemplo de puntos característicos tomados con MonoSLAM.

PTAM

Parallel Tracking and Mapping. Es un algoritmo creado en 2007 por George Klein [Klein and Murray, 2007] que también calcula el *Tracking* y el *Mapping* como en MonoSLAM pero para ello utiliza 2 *Threads*, 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.

Con las imágenes captadas en secuencia se van generando *Keyframes* o fotogramas clave. Se genera un nuevo *Keyframe* a medida que la cámara se va desplazando. Los *Keyframes* se utilizan para la localización y para ir generando el mapa de puntos. Este algoritmo es recomendable para mapas con elevado número de puntos, es capaz de recuperarse fácilmente de un secuestro, extrae los puntos de interés mediante extracción de características como en MonoSLAM y trata de emparejarlos con los puntos extraídos de las imágenes anteriores. Como extractor de características utiliza el detector FAST. 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 con *Bundle Adjustment*.

Se suele utilizar en entornos cerrados y pequeños y utiliza técnicas SFM. Muy utilizado también para realidad aumentada.



(a)

Figura 3.3: Nube de puntos característicos tomados con PTAM.

3.2. ORB-SLAM

Es un algoritmo basado en extracción y emparejamiento de píxeles característicos mediante descriptores ORB, estos descriptores 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]. Puede ser utilizado con una cámara o dos e incluso con cámaras de profundidad RGBD. Para cierres de bucle y relocalización utiliza un modelo de bolsa de palabras [Gálvez-López and Tardos, 2012]. Utiliza 3 *Threads*, el primero para *Tracking*, el segundo para *Mapping* y un tercero para detectar cierres de bucle.

En el proceso 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 proceso 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 corresponde con un vértice y un vértice estará unido a otro siempre y cuando los *Keyframe* tengan varios puntos 3D en común. Este grafo permite la eliminación de *Keyframe* redundantes (Figura 3.4(b)).

En el proceso 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.

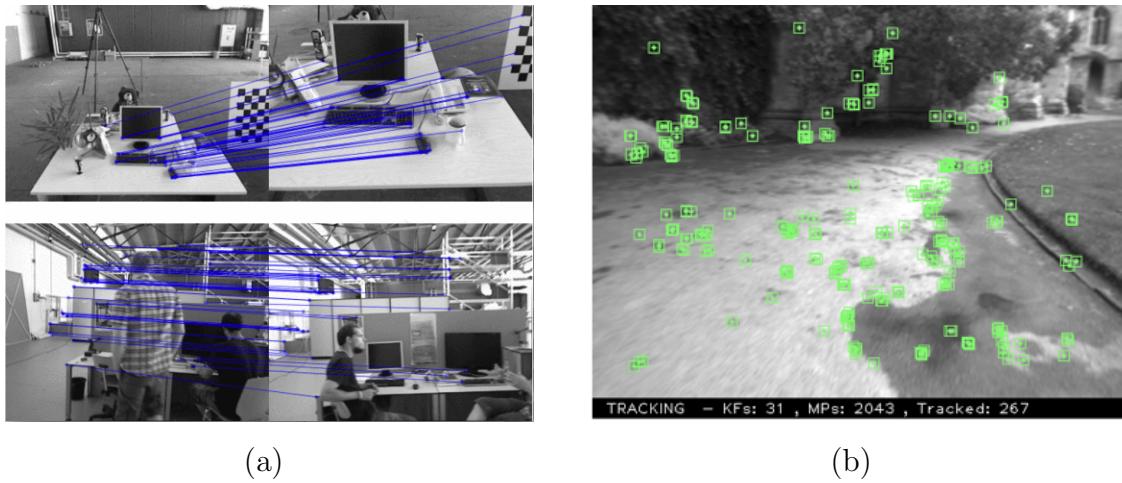


Figura 3.4: Localización de puntos característicos en 2 imágenes con ORB

3.3. DSO y LDSO

DSO: Direct Sparse Model. Está basado en optimizaciones continuas del error fotométrico sobre una ventana de frames 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 frames son creados con respecto a este frame utilizando alineamiento directo de 2 frames, una pirámide multi escala y un modelo de movimiento constante a inicializar. Para la relocalización, se podrán trazar hasta 27 rotaciones pequeñas en diferentes direcciones. Esta recuperación de posición se consigue en el nivel más pequeño de la pirámide de la imagen. La creación de *Keyframes* es similar a ORB-SLAM, existen 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 translación de la cámara causa occlusiones y des-occlusiones, 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 frames.

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 frames 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 frames 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 frames 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 $d \times d$, 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 siguientes frames 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 frames siguientes. Esta estrategia de localización está inspirada en LSD-SLAM.

Por último, la activación de puntos candidatos, 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.

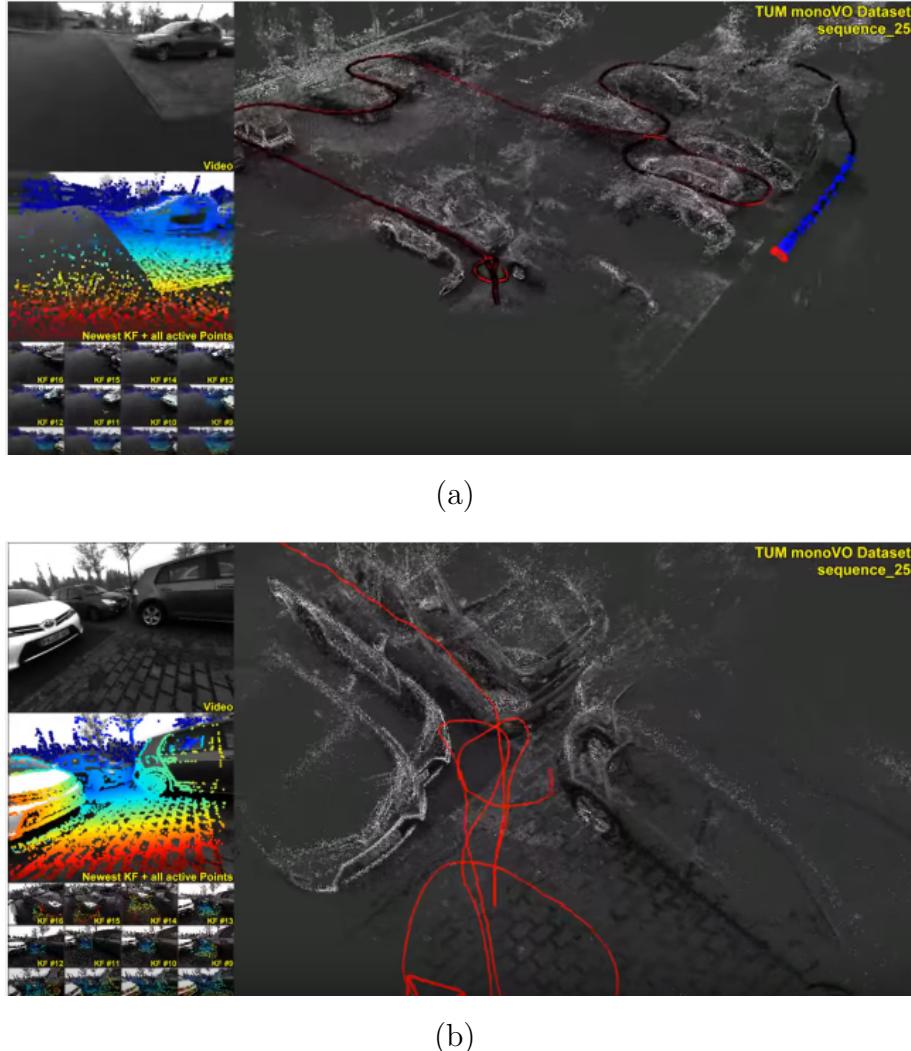
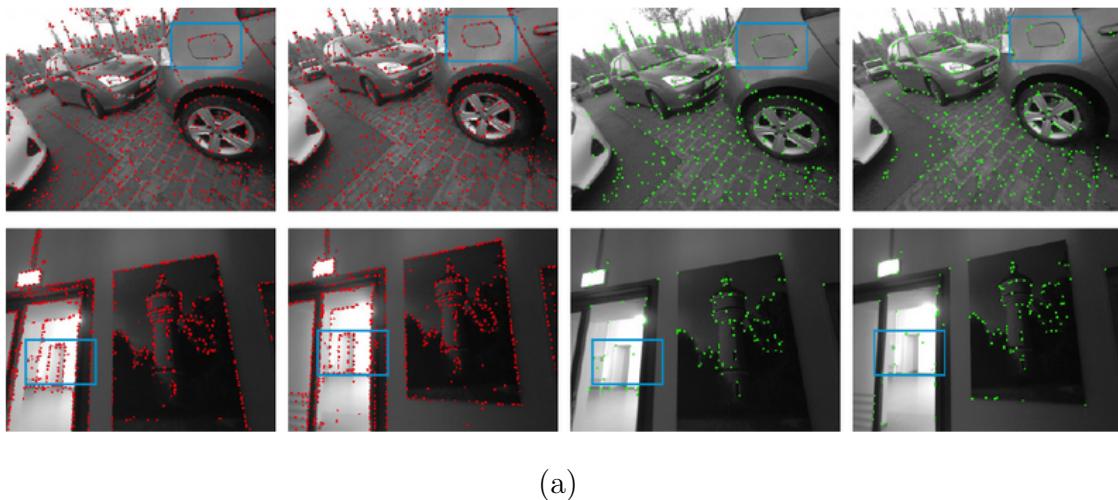


Figura 3.5: Mapa generado con DSO (a) Ligero error en la posición al volver al punto de partida (b).

LDSO: Direct Sparse Odometry with Loop Closure. Este método es una extensión del algoritmo DSO (Direct Sparse Odometry). LDSO incorpora detección de cierre de bucle y optimización de posición y mapeo. Al ser un método directo, DSO puede utilizar cualquier pixel 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

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

puntos prestando más 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. Los candidatos a cierre de bucle son verificados geométricamente y restricciones de pose relativa son estimadas minimizando en conjunto errores geométricos 2D y 3D. Selección de puntos de características Repetibles, DSO utiliza un grid dinamico para detectar suficientes pixels incluso en entornos con pocas texturas. Más específicamente, todavía se toma un numero determinado de pixeles (por defecto 2000 en DSO), de los cuales algunos son esquinas. Manteniendo el número de esquinas pequeño se calculan los descriptores ORB y los empaquetamos en BoW. El algoritmo utiliza ambos tipos de pixeles de esquinas y no esquinas para obtener el tracking, y la carga del thread para obtener el cierre de bucle permanece al mínimo. En un frame capturado mediante DSO hay pocos elementos repetibles y por tanto es difícil buscar emparejamiento de imágenes usando esos puntos para cierre bucle. Pero en LDSO se utilizan ambos esquinas y otros pixeles con alto gradiente, donde las esquinas se utilizan para construir modelos BoW (Bag of Words) y para tracking, mientras que los no esquinas sólo se utilizan para tracking. De esta forma podemos hallar la posición en entornos con pocas texturas y tambien encontrar características comunes entre keyframes si lo necesitasemos. Cada vez que calculamos descriptores ORB para cada keyframe, una base de datos BoW es construida. Los candidatos a cierre de bucle son propuestos para el keyframe actual mediante consultas a la base de datos y solo se toman aquellos que están fuera de la actual ventana. Para cada candidato intentamos machear sus características ORB a aquellos del keyframe actual, y entonces ejecutamos RANSAC PnP para computarla estimación inicial de la transformación. Despues se optimiza la transformación usando el método de Gauss-Newton mediante la minimización de restricciones geométricas 3D y 2D.



(a)

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

4. Herramientas para realizar evaluaciones comparativas de algoritmos SLAM

En este apartado trataremos sobre varias herramientas que podemos utilizar para hacer benchmarking sobre los resultados de algoritmos SLAM y evaluar y comparar dichos resultados

1. Computer Vision Group TUM

Proporciona varias herramientas para evaluar el rendimiento de algoritmos VSLAM, 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, tipo float proporciona el número de segundos , tx ty tz que son de tipo float, dan la posición del centro óptico de la cámara con respecto al origen de coordenadas del mundo real , qx qy qz qw , también de tipo float, dan la orientación del centro óptico de la cámara con respecto al origen de coordenadas del mundo real pero en formato de quaternion unidad.

Absolute Trajectory Error (ATE): El error absoluto de trayectoria mide la diferencia que existe entre cada punto de la trayectoria estimada y la trayectoria real. Como paso de preproceso se realiza una asociación entre la posición estimada con la posición ground truth 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, el ordenador 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 2 trayectorias.

Relative Pose Error (RPE): Con el script python evaluaterpe.py es posible calcular el error relativo a la posición . Este script obtiene el error entre el movimiento relativo entre pares de timestamps. Por defecto el script calcula el error entre todos los pares de timestamps del fichero de trayectoria. Como el numero de pares de timestamps en la trayectoria estimada es cuadrático se pueden poner cotas con un número máximo de pares de timestamps. Opcionalmente, tambien se puede elegir usar un tamaño de ventana fijo (-fixed delta). En este caso cada pose en la trayectoria estimada es asociado con posteriores posiciones dependiendo del tamaño de ventana (-delta) y unidad (-delta unit). Esta técnica de evaluación es util para calcular el desvio o deriva.

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

Esta herramienta está desarrollada en python e incluye : Métodos de alineamiento de trayectorias para diferentes modalidades de sensores Métricas de error tales como ATE y Relative Odometry Error. [Zhang and Scaramuzza, 2018]

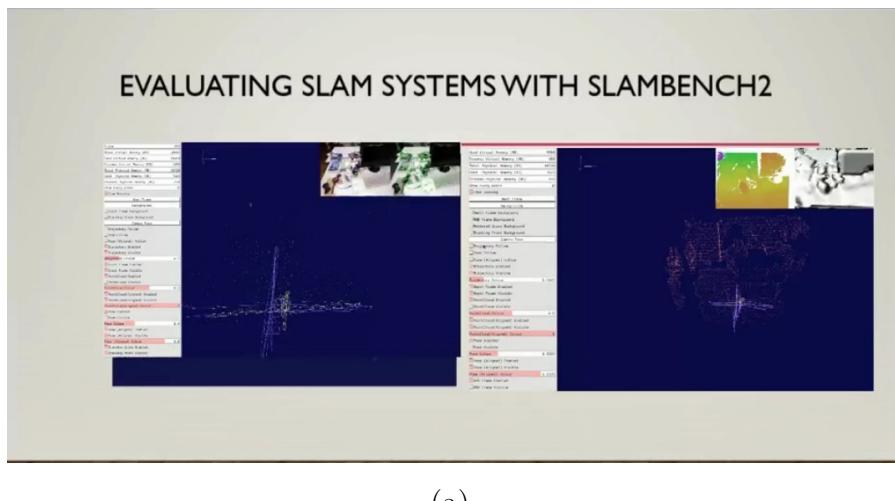
El software ha sido diseñado para uso facil. Dados 2 ficheros de texto donde especificaremos la trayectoria estimada y el groundtruth, la herramienta establece automáticamente el emparejamiento de tiempos, realiza alineamiento de trayectoria y calcula distintos errores métricos con una línea de comandos. Tambien se puede utilizar para comparar diferentes algoritmos con múltiples datasets. Para que la aplicación pudiese ser utilizada con diferentes formatos, tambien 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

3. SLAMBENCH

SLAMBench es una herramienta de la Universidad de Edimburgo, esta herramienta

ha sido creada para evaluar sistemas SLAM ya sean sistemas de código abierto o sistemas propietarios 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 reproductibilidad de los resultados para los sistemas SLAM actuales y posibilita la integración y evaluación the nuevos resultados SLAM.

SLAMBench soporta varios algoritmos diferentes. Entre los algoritmos escasos o poco densos soportaría los siguientes MonoSLAM, PTAM ,OKVIS y ORB-SLAM2. Los 2 primeros algoritmos soportan solo sistemas monoculares, OKVIS soporta sólo estereo y RGB-D y ORBSLAM2 soporta ambos tipos. Estos algoritmos son algoritmos indirectos. Entre los métodos densos se soportan 3 tipos de algoritmos, KinectFusion, InfiniTAM, ElasticFusion que son 2 recientes modelos densos. Por último LSD-SLAM es un sistema SLAM semi-denso de una sola cámara (monocular).



(a)

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 ground-truth. 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, Reconstrucción de Error , 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 ground-truth 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 se calcula mediante la ejecución del algoritmo Iterative Closest Point (ICP) de los modelos de la nube de puntos de la reconstrucción y del ground truth. 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 frame.

Interface de usuario modular: SLAMBench también permite elegir entre diferentes sistemas de interfaz de usuario. Métricas de evaluación pueden ser cambiadas y personalizadas, 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.

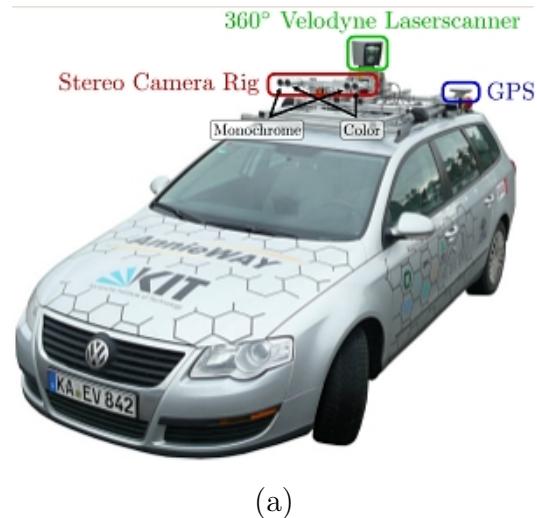
SLAMBench está siendo un componente muy importante en robótica y Sistemas de Realidad Aumentada (AR). Aunque un gran número de algoritmos SLAM han sido presentados, no se ha investigado lo suficiente para tratar de unificar el interface de estos algoritmos, o realizar comparaciones de todas sus capacidades 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 un framework de evaluación que compararía sistemas SLAM actuales y futuros, utilizando una lista extensible de datasets, mientras utiliza una lista comparable de métricas de rendimiento. Se podrían utilizar una gran variedad de algoritmos de SLAM y datasets como ElasticFusion, ORB-SLAM2, OKVIS y también se podría integrar con nuevos algoritmos y datasets. SLAMBench2 es un software que está disponible de manera

pública.

4. The Kitti Vision Benchmark Suite

Es un conjunto de aplicativos que utilizan mapas y secuencias grabados desde la plataforma de coches autónomos Annieway para crear nuevos desafíos o retos en las tareas comparativas o benchmarking de visualslam.[Geiger *et al.*, 2012] Están investigando en varios campos como: vision stereo, flujo optico, odometría, detección de objetos en 3D y seguimiento de objetos 3D. La exactitud de los conjuntos de datos ground truth es medida gracias al scanner laser 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 Karslruhe. Además de proporcionar todos los datos en formato raw, para cada uno de sus benchmarks, también proporcionan una metrica de evaluación y una web de evaluación de métricas. En experimentos preliminares se ha comprobado que métodos que obtienen una puntuación alta en algunos benchmarks, cuando son aplicados al mundo 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 stereo, grabadas en formato png. En el dataset se proporcionan 11 secuencias con trayectorias ground truth para entrenar y 11 secuencias sin ground truth 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 metodo deber 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 trasalación y en grados por metro para la rotación.



(a)

Figura 3.8: Coche autónomo Annieway utilizado con Kitti.

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 DSO.

Funcionalidad	Mono-SLAM	PTAM	ORB-SLAM	LDSO
Probabilístico	Sí	No	No	No
Hilos de ejecución	1	2	3	2
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	No
Rechazos de espurios	No	No	Sí	No
Cierre de bucle	No	No	Sí	Sí

Capítulo 4

Herramienta SLAMTestBed

En este capítulo se detallará el diseño de la herramienta SLAMTestBed, una herramienta diseñada para comparar algoritmos SLAM. El diseño de algoritmos SLAM es todavía una disciplina abierta en periodo de expansión. En la navegación autónoma de robots y drones es donde mayor importancia alcanza la utilización de los algoritmos SLAM. Pero algoritmos hay muchos y se necesitan herramientas que nos permitan medir la bondad de cada algoritmo. Comenzaremos explicando el diseño desde un punto de vista de Caja Negra, explicando las entradas y las salidas. Se continuará con la explicación del diseño en Caja Blanca y se finalizará explicando cada uno de los componentes principales de la herramienta , como 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)

4.1. Diseño de Caja Negra

En esta sección no entraremos en los detalles de la implementación de la herramienta SLAMTestBed, si no que lo trataremos como una *caja negra*. De un lado tendremos las entradas al sistema, que serán 2 datasets. Una vez procesados los 2 datasets por la caja negra, obtendremos a la salida, las transformaciones estimadas por la herramienta, junto con un dataset estimado que utilizaremos para medir el error cometido en las estimaciones.

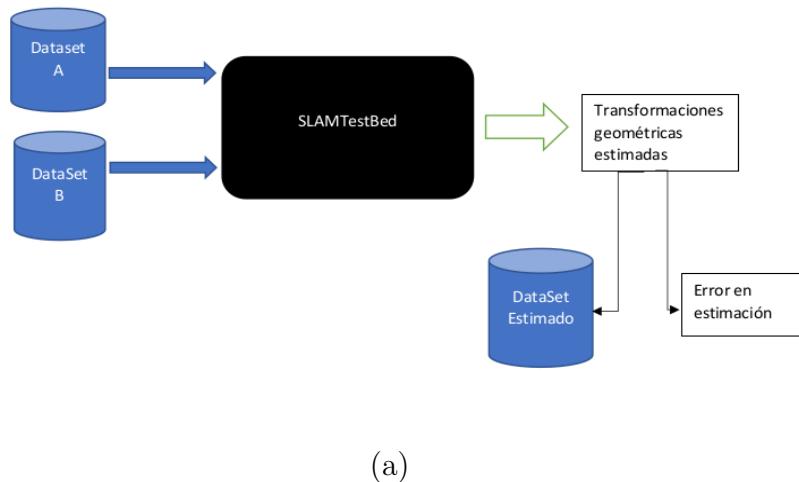


Figura 4.1: En la imagen superior, el diseño de Caja Negra de la herramienta SLAMTestBed

4.2. Diseño de caja Blanca

En esta sección explicaremos con más detalle la implementación de la herramienta SLAMTestBed. Las entradas y salidas del sistema serán las mismas que las explicadas en la sección anterior (Diseño de Caja Negra).

La explicación del algoritmo: En este apartado explicaremos los pasos seguidos para hacer los cálculos que nos permitirán estimar las transformaciones requeridas para pasar de un dataset A a otro dataSet B. Las principales funciones o módulos de los que constará la herramienta será:

Cálculo de PCA : Nos permitirá reducir el dataset a sus componentes principales.

Estimación de Escala : Estimará la diferencia de escala entre los dos datasets.

Estimación de Offset : Con este módulo podremos hallar la diferencia entre marcas de tiempos de los 2 datasets.

Interpolación para igualar frecuencias de muestreo : Con la interpolación podremos igualar en frecuencias los dos datasets.

Operaciones de Registro para estimar la Rotación y Traslación :

Permitirá estimar las traslación y rotación necesarias para pasar de un datasetA a un datasetB.

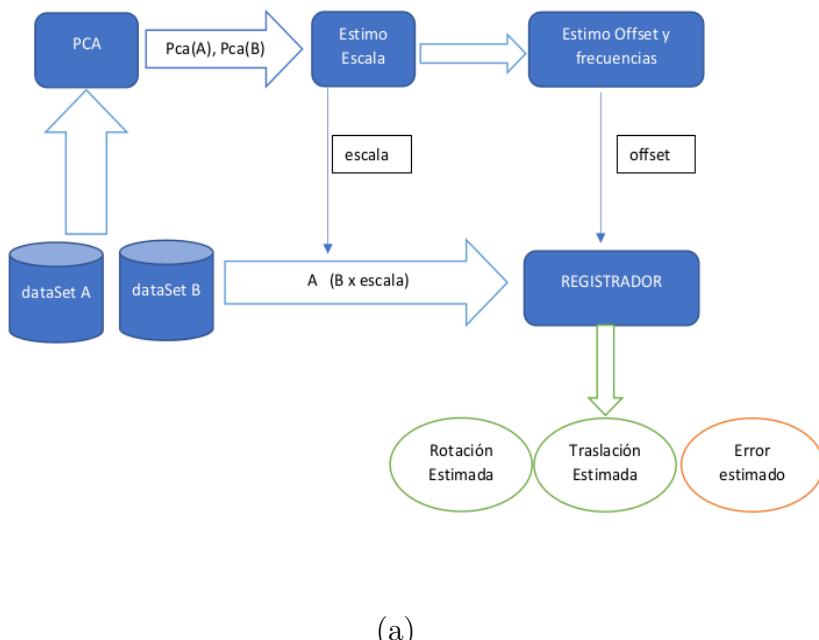


Figura 4.2: En la imagen superior, el diseño de Caja Blanca de la herramienta SLAM TestBed

Básicamente el flujo de datos que sigue la imagen superior es el siguiente:

Se calculará la descomposición en componentes principales (PCA) para cada dataset, con estos nuevos datasets, calcularemos la escala. Posteriormente estimaremos el offset de tiempo entre los dos datasets y unificaremos las frecuencias.

Con los datos `dataSetApca` y `dataSetBpca` estimaremos el offset de tiempo que exista entre las 2 secuencias de datos o datasets. Posteriormente corregiremos el offset en el dataset B, para ello sumaremos el valor del offset a los valores de tiempo del `datasetB`. Una vez tengamos calculado el offset, unificaremos las frecuencias de los 2 datasets mediante interpolación. Cuando tengamos unificada la frecuencia para los dos datasets, obtendremos de nuevo PCA de cada dataset y por último obtendremos la escala. Posteriormente estimaremos la transformaciones de rotación y traslación para pasar del `datasetA` al `datasetB`. Aplicaremos dichas transformaciones sobre el `datasetA`, que como resultado nos dará un nuevo dataset, el dataset Estimado que pintaremos en pantalla de color rojo. Más adelante se explicarán con más detalle cada uno de estos módulos.

4.3. Estimador PCA y Cálculo de Escala

Con el módulo PCA obtendremos las componentes principales de un dataset, generando un nuevo dataset.

A partir de los nuevos datasets generados podremos calcular la escala y la diferencia de tiempos entre los 2 datasets.

Se implementan 2 métodos de cálculo de PCA, el segundo utilizando el método SVD (utilizando la librería Eigen), que describiremos a continuación.

```

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

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

```

En cuanto al cálculo de la Escala, nos permitirá calcular la diferencia de escala entre los dos datasets, y así al tener los datasets en la misma escala, ya tendríamos el primer paso del registro espacial. El algoritmo utilizado estará basado en la utilización de los valores singulares de cada datasets.

```

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

```

4.4. Registro Espacial

Con el registro espacial conseguiremos estimar la matriz de rotación y traslación para aplicarlas sobre el segundo dataset, intentando conseguir la operación de registro de los 2 datasets.

Hallar la matrices de rotación y translación .

El algoritmo para esta funcionalidad sería el siguiente:

- Hallar los centroides de las coordenadas X,Y,Z de cada datos
centA = centroides (A)
centB = centroides (B)
 - Restar los centroides a sus respectivas matrices
 $A = A - \text{centA}$
 $B = B - \text{centB}$
 - Calcular el producto de las matrices
 $H = A \cdot \text{traspuesta} * B$
 - Obtener valores singulares de la matriz H
 $S, U, V = \text{svd}(H)$
 - Obtener la matriz de Rotacion
 $R = V * U \cdot \text{traspuesta}$
 - Obtener la matriz de Traslacion
 $t = -R * \text{centA} + \text{centB};$

Transformar un dataset en otro aplicando las matrices de Rotación y Traslación previamen

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

Transformar los cuaternios aplicando la matriz Rotación. Con

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 cara quaternion p de la siguiente forma. $q * p * q^{-1}$

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.

El algoritmo seguido ha sido el siguiente:

```

Mientras no alcancemos el numero maximo de iteraciones
    Seleccionar n inliers
    Estimar una primera matriz de Rotacion
    aplicar transformacion sobre un subconjunto
    Si el error < threshhold
        añadir candidato a inliers
        medir el error tras aplicar transformacion
        seleccionar menor error
        incrementar iteracion
    Devolveremos las matrices de Rotacion Traslacion que mejor se adapten
  
```

4.5. Estimador Offset

Con este módulo podremos calcular el offset o desplazamiento en el tiempo entre los 2 datasets. El método principal utilizado para calcular el offset se realiza mediante el cálculo de la Correlación Cruzada. Para ello previamente, calcularemos la distancia al origen para cada punto 3D de los 2 datasets. Para un punto 3d (x,y,z), la distancia 3d se calculará $\sqrt{(x - 0)^2 + (y - 0)^2 + (z - 0)^2}$ una vez calculadas las distancias, aplicaremos el siguiente algoritmo de cálculo de correlación cruzada

Este módulo se encarga de estimar el offset o diferencia de tiempo entre los dos datasets. Es el algoritmo más pesado ya que incluye múltiples cálculos para interpolar y calcular la correlación cruzada entre los dos datasets.

Este algoritmo será iterativo y utilizaremos un dataset como base de tiempo fijo y otro dataset B que se deslizará en el tiempo desde $-T$ hasta $+T$. El datasetB se irá deslizando en pequeños espacios de tiempo o steps en cada iteración. Para cada step calcularemos la correlación cruzada entre el dataset deslizante y el dataset original (que se mantiene fijo en el tiempo). Para un step determinado la correlación debe ser máxima entre los 2 datasets y una vez hallada la correlación máxima, el step de esa iteración nos indicará el offset. Como hemos comentado, moveremos el datasetB hasta llegar a $-T$ y por cada iteración iremos deslizando en el tiempo el datasetB un incremento de tiempo step, hasta el final del bucle

que terminaremos de deslizar datasetB hasta llegar a un tiempo $+T$. En cada iteración calcularemos la interpolación en los puntos que coincidan que estén dentro de los rangos temporales de los dos datasets. Una vez interpolados los puntos comunes de las 2 series tendremos 2 series nuevas, Para las cuales calcularemos la correlación cruzada aplicando el algoritmo típico para 2 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 calcularemos 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.

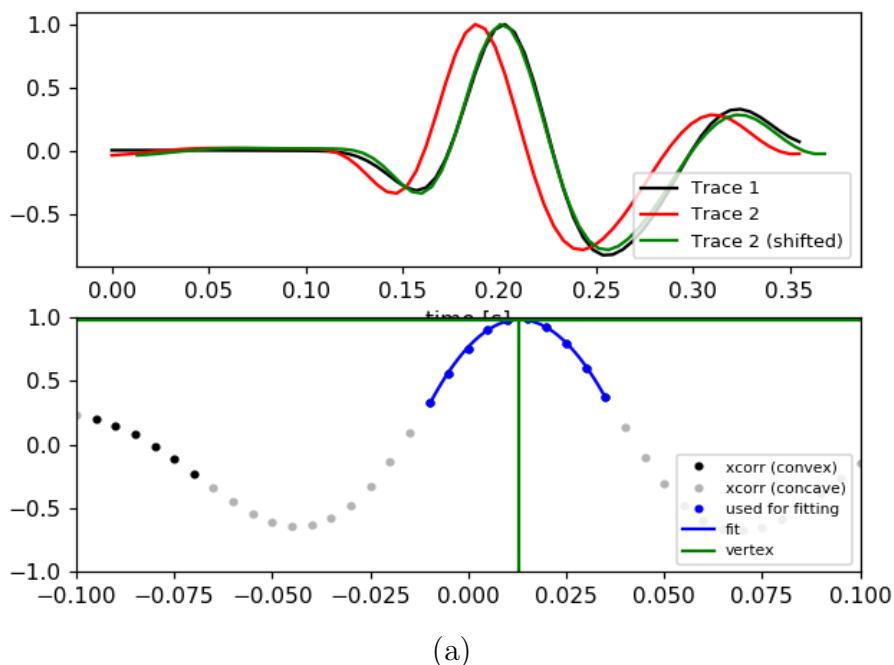


Figura 4.3: Gráfico que muestra el desfase del offset entre dos series temporales y el resultado de la correlación cruzada.

```

dataA = CalcularDistancia( dataSetA )
dataB = CalcularDistancia( dataSetB )
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 );

```

4.6. Interpolador

El módulo de interpolación se utiliza para sincronizar a igual frecuencia 2 datasets. Entre las distintas funciones , permite sincronización a la frecuencia mayor de los 2 datasets, sincronización a la frecuencia menor de los 2 datasets o incluso consigue sincronizar las 2 series a una frecuencia deseada. La interpolación se realiza sobre las secuencias temporales de los 2 datasets, pero tambien se calcula la interpolación para las coordenadas X,Y y Z de cada punto 3D y se realiza tambien interpolación para los quaternios. La función *slerp* de la librería Eigen nos permite calcular la interpolación entre quaternios. La función slerp recibe como parámetro una marca de tiempo y un quaternion.

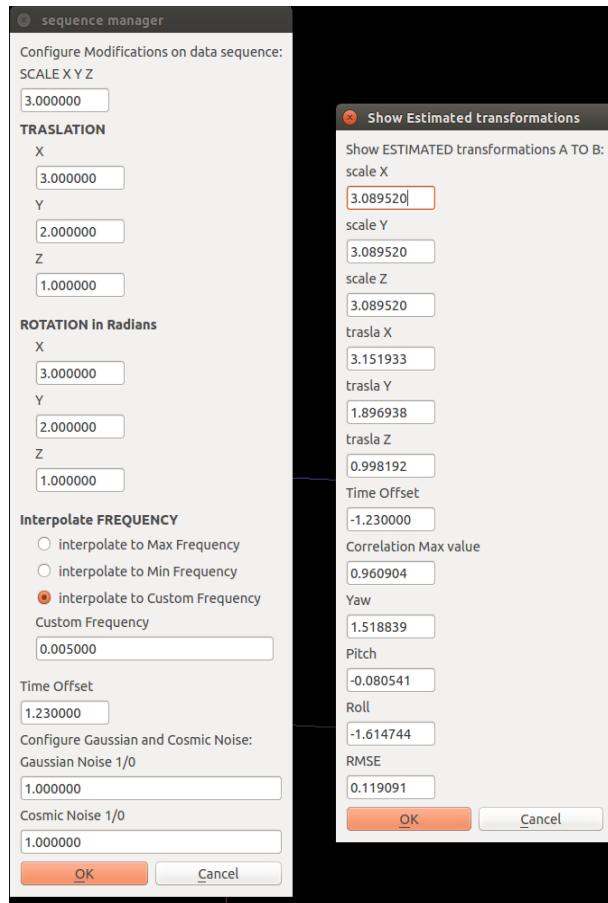
4.7. Cálculo de Estadísticas

Para medir el error cometido entre el dataset Groundtruth y el dataset Estimado utilizaremos el módulo de Cálculo de Estadísticas. Este módulo permite calcular el Error Cuadrático Medio (Root Mean Square Error) entre 2 datasets. Cuanto menor sea el RMSE , mejor será el dataset Estimado. También nos permite hallar la media, mediana , máximo y mínimo valor de un dataset.

4.8. GUI

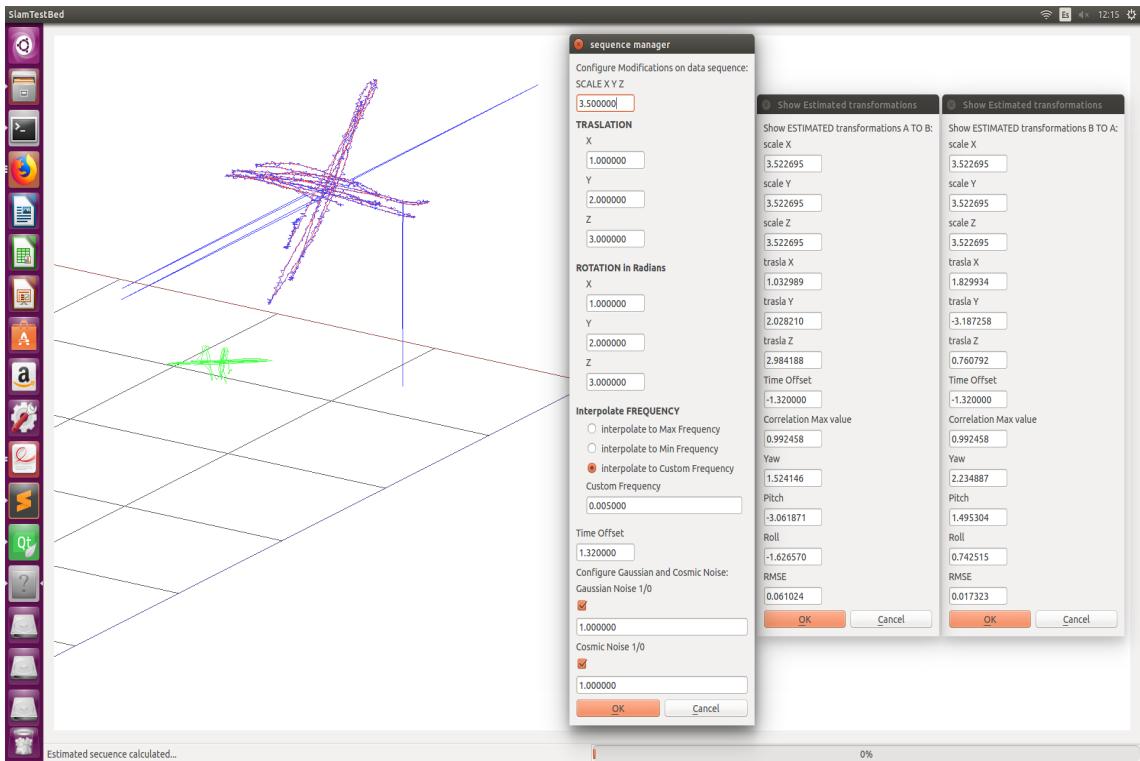
Para la herramienta SLAMTestBed tambien se ha diseñado y creado un Interfaz de Usuario Gráfico (GUI) que permita pintar los puntos 3D de cada dataset en pantalla así como invocar comandos (a través de clicks de ratón) para ejecutar los distintos módulos que contiene la herramienta SLAMTestBed. Nos permitirá 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 interfáz gráfico de usuario de 3 dimensiones ha sido desarrollado en C++ con el entorno QT y la librería Eigen. La librería Eigen es una librería Open Source realizada en C++, para cálculos de Álgebra lineal, operaciones con Matrices y vectores, Transformaciones Geométricas.

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. Este archivo podriamos denominarlo groundTruth. Con el ratón podremos girar en 3 dimensiones la nube de puntos, acercarnos a el (Zoom in) o alejarnos (Zoom Out), tambien permite leer un segundo conjunto de puntos 3D y estimar las transformaciones (rotaciones , traslaciones, escala etc) necesarias para pasar desde el conjunto de datos ground truth hasta este segundo conjunto de datos. 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 los 3 datasets. Los puntos 3D del 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.

Capítulo 5

Experimentos

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

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 verde), el dataset transformado (en azul), y el dataset estimado (en rojo), junto con otras 3 pantallas que indicarán los valores de los parámetros de las transformaciones realizadas, y otras 2 pantallas con los valores de la estimaciones calculadas (tanto si la estimación ha sido calculada del datasetA al datasetB o del datasetB al datasetA).

5.1. Módulo Transformador

Esta herramienta posee varios modulos accesibles desde el interfaz gráfico. Entre estos módulos podríamos destacar el **Módulo Transformador**. Como se ha comentado

anteriormente, el módulo Transformador permite realizar transformaciones sobre el conjunto de puntos 3D groundTruth, de tal forma que se obtendrá como resultado una segunda nube de puntos transformados. De esta forma se han podido realizar pruebas teniendo un sólo datasets para comprobar que los calculos estimados de Traslación , Rotación y escala son fiables y tienen un error mínimo.

Los diversos parámetros del módulo transformador serán accesible por el interfaz gráfico de la pantalla.

A continuación describiremos en detalle las transformaciones permitidas por la herramienta:

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

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

Rotaciones. Se podrán definir rotaciones sobre cada uno de los 3 ejes de coordenadas. El valor de cada rotación se insertará en Radianes. Los valores admitidos serán números reales tanto positivos como negativos. Para indicar rotaciones, los ángulos de rotación se incluirán en Radianes.

Offset de tiempo. Con el offset de tiempo podremos introducir un 'gap' en los valores de timestamp del fichero de entrada que más tarde podremos estimar. La exactitud del offset será de centésimas, es decir con 2 decimales.

Interpolación: El módulo de interpolación nos permitirá ajustar a la misma frecuencia los 2 datasets. Se podrán realizar 3 tipos de interpolación de los datos. Interpolación a la frecuencia máxima Interpolación a la frecuencia mínima Interpolación a frecuencia personalizada

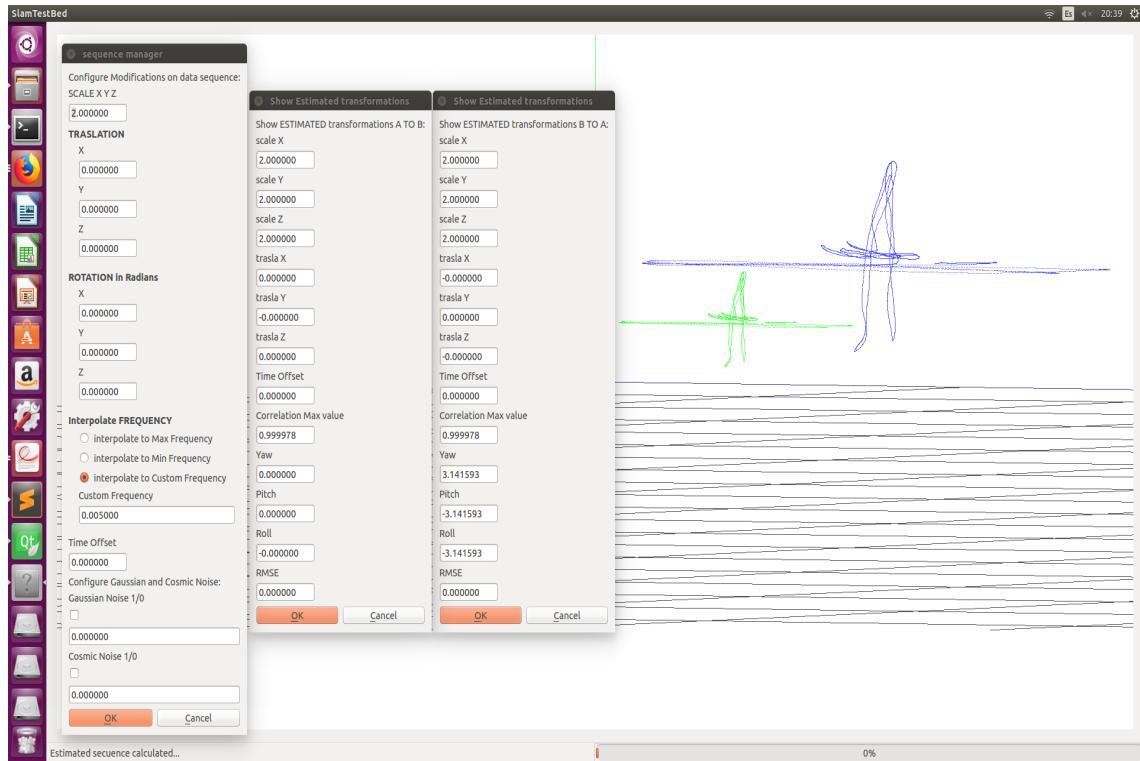
Ruido Gaussiano: Una de las transformaciones que podremos aplicar sobre el conjunto de datos del groundtruth es aplicar un ruido gaussiano a los datos transformados

Ruido Cósmico: Otra transformación a aplicar sobre los datos transformados es la incorporación del ruido cósmico

5.2. Pruebas de transformaciones individuales

En este apartado, mostraremos imágenes del módulo GUI, donde se han realizado varias pruebas de transformaciones individuales y se han estimado dichas transformaciones. En

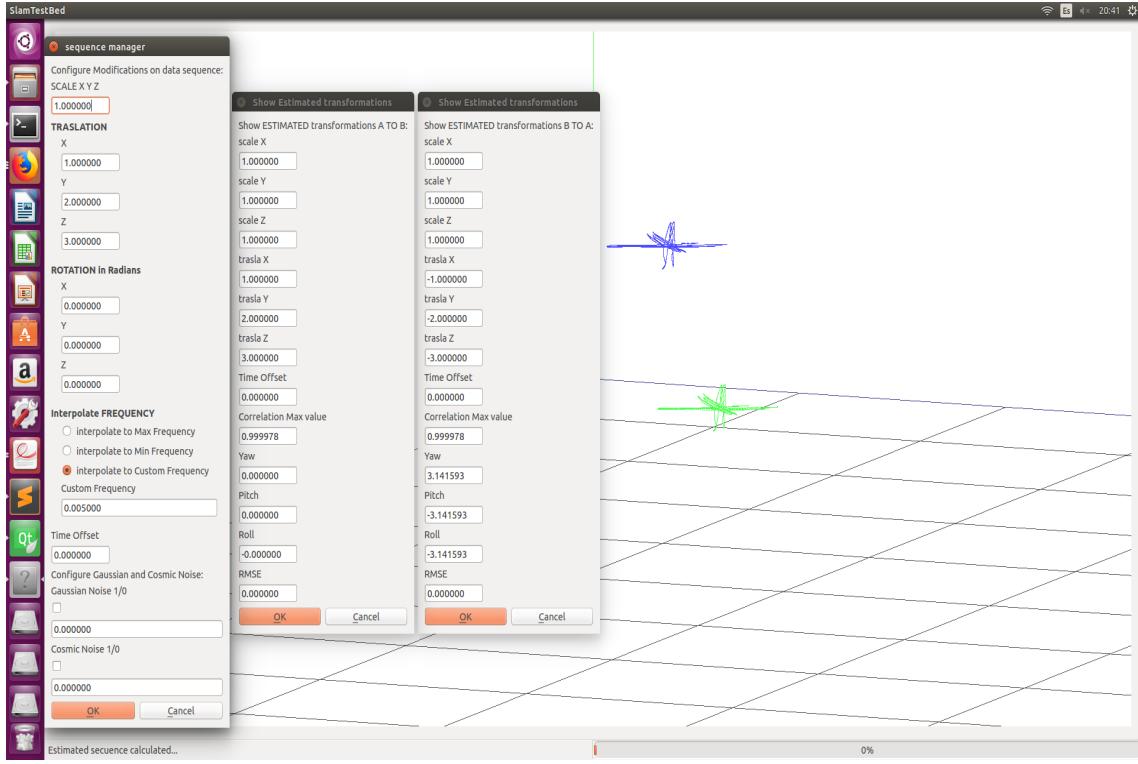
todas las transformaciones se ha realizado la interpolación de frecuencias al valor 0.05 para ambos datasets.



(a)

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

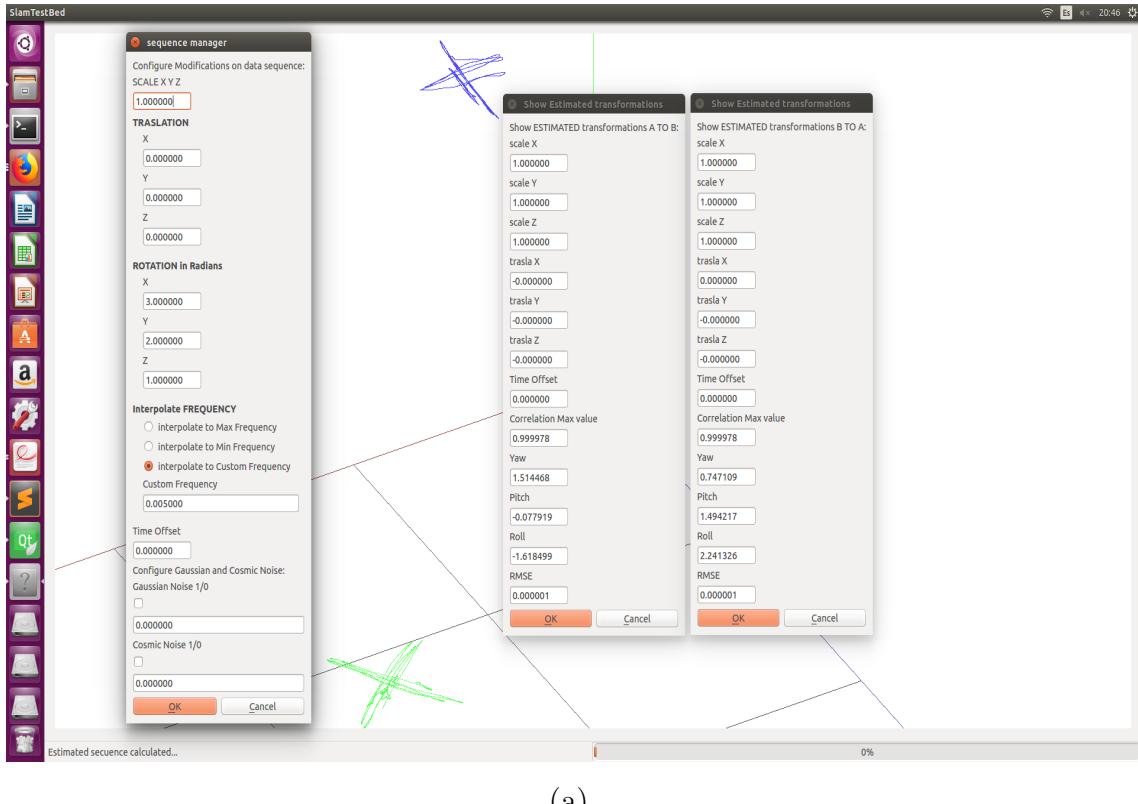
El gráfico superior muestra los resultados de la estimación tras una transformación de un cambio de escala. Se aprecia en la captura de pantalla, como el dataset transformado es más grande que el dataset original. Como se puede observar el RMSE es 0.0 y el valor de la escala estimada coincide con el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



(a)

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

La captura de pantalla superior muestra los resultados de la estimación tras realizar una translación 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 translación estimada puede verse que coinciden con la transformación realizada.

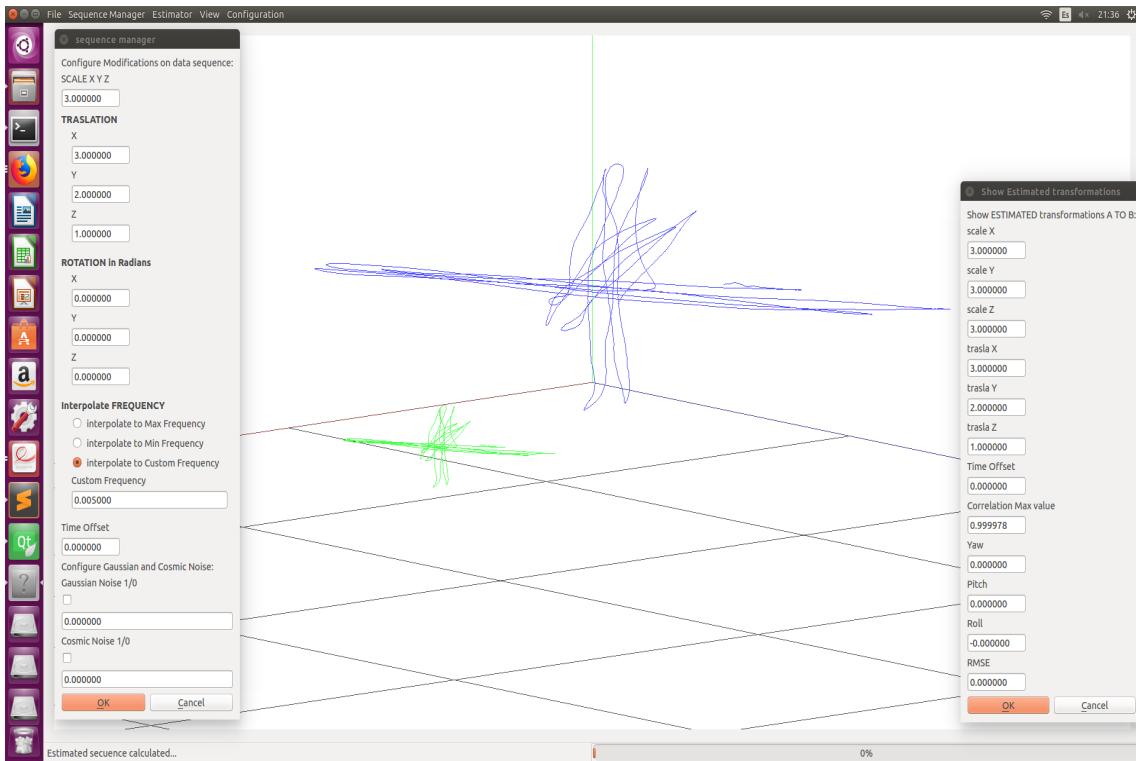


(a)

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

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 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.

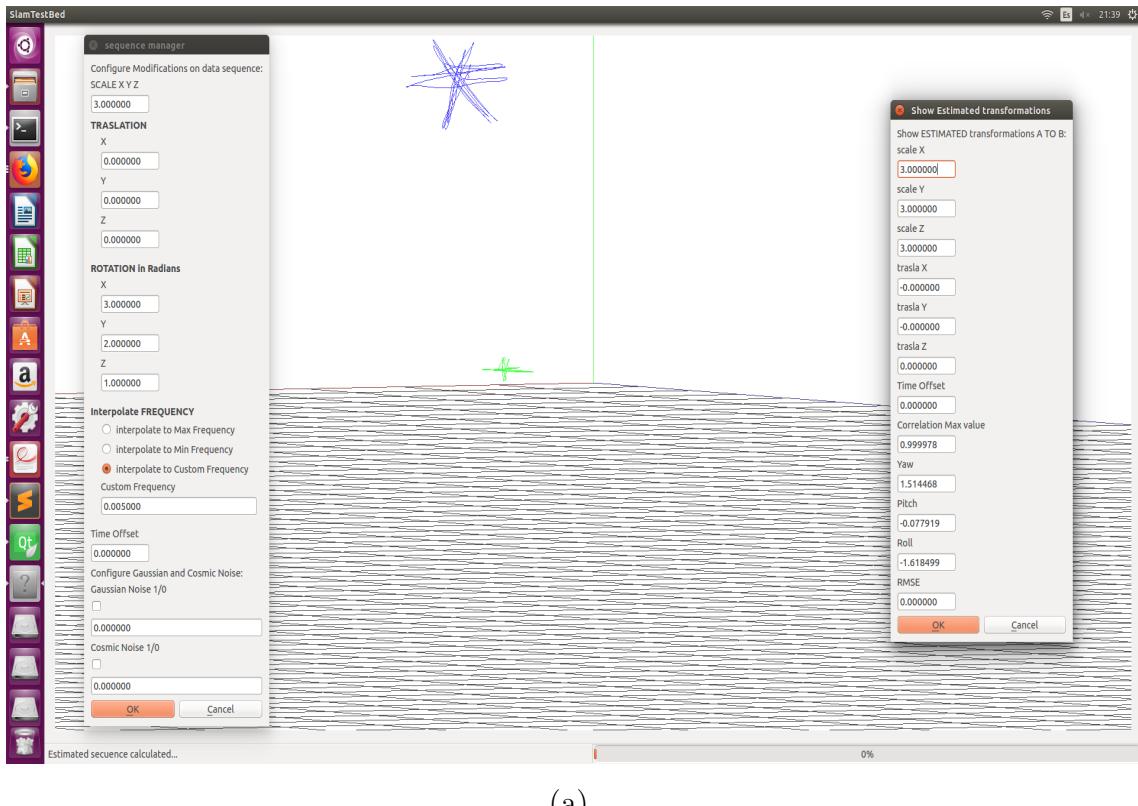
5.3. Pruebas de transformaciones en parejas



(a)

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

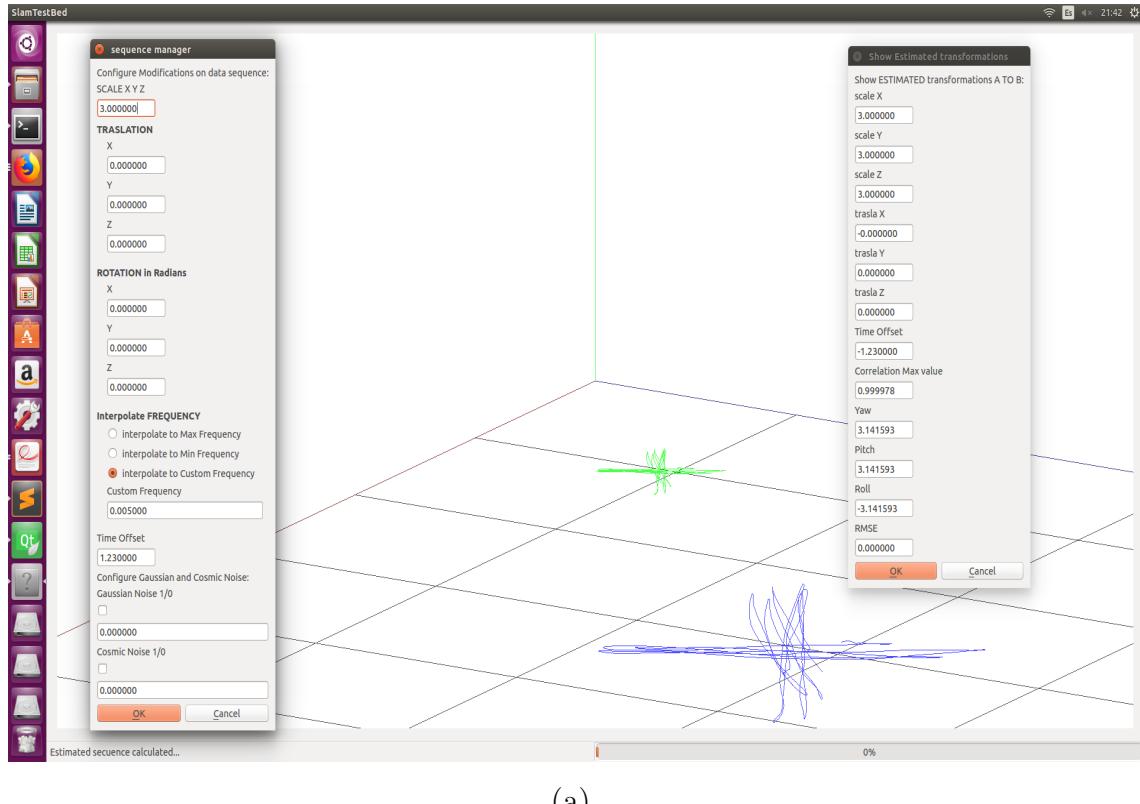
En la imagen superior, se muestra en color verde, el dataset groundtruth, y en azul el mismo dataset tras realizarle un par de transformaciones en escala y traslación. En la ventana de la derecha se muestran los resultados de la transformación estimada, que coincide con la transformación realizada, además 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 el dataset transformado y no se aprecia ningún punto rojo. En este caso la transformación realizada es de escala y traslación. Se puede comprobar que la estimación realizada es correcta ya que coinciden los valores de la escala y traslación estimados.



(a)

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

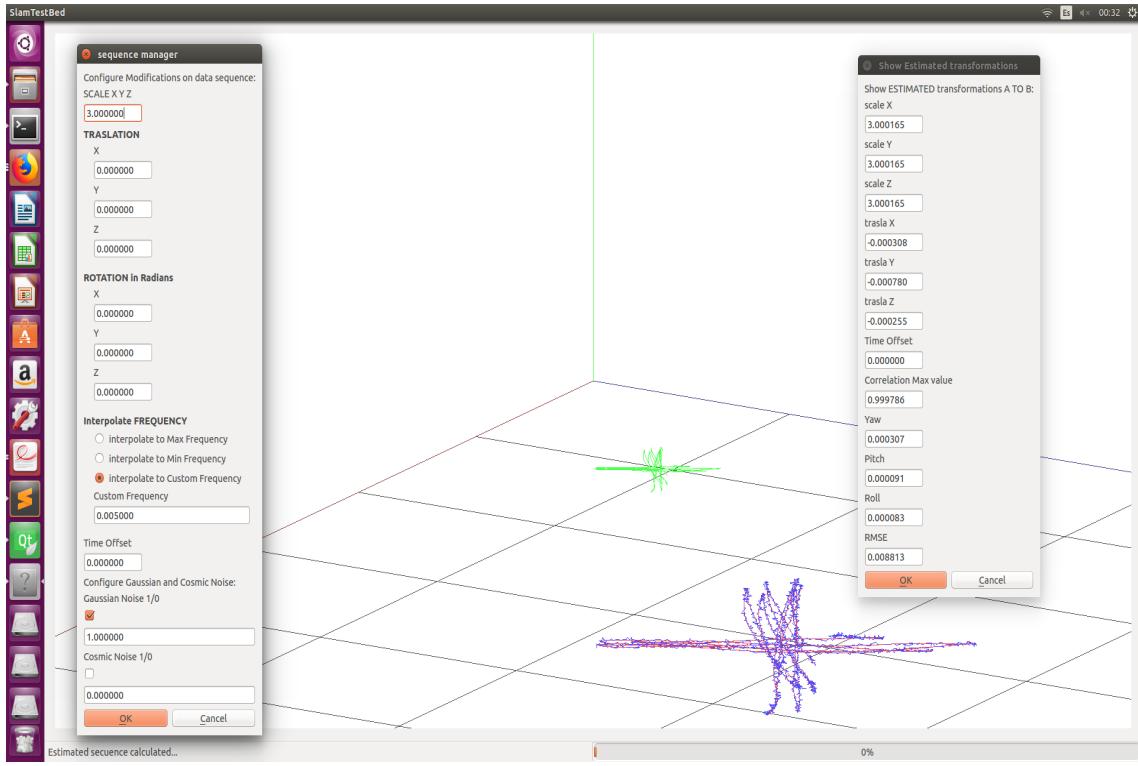
En la imagen superior podemos ver los resultados de aplicar otras 2 transformaciones sobre el dataset original. En este caso se trata de una transformación en escala y rotación. Tras realizar la estimación, el error medido entre el dataset estimado y el dataset transformado es 0.0 . Por tanto podemos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



(a)

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

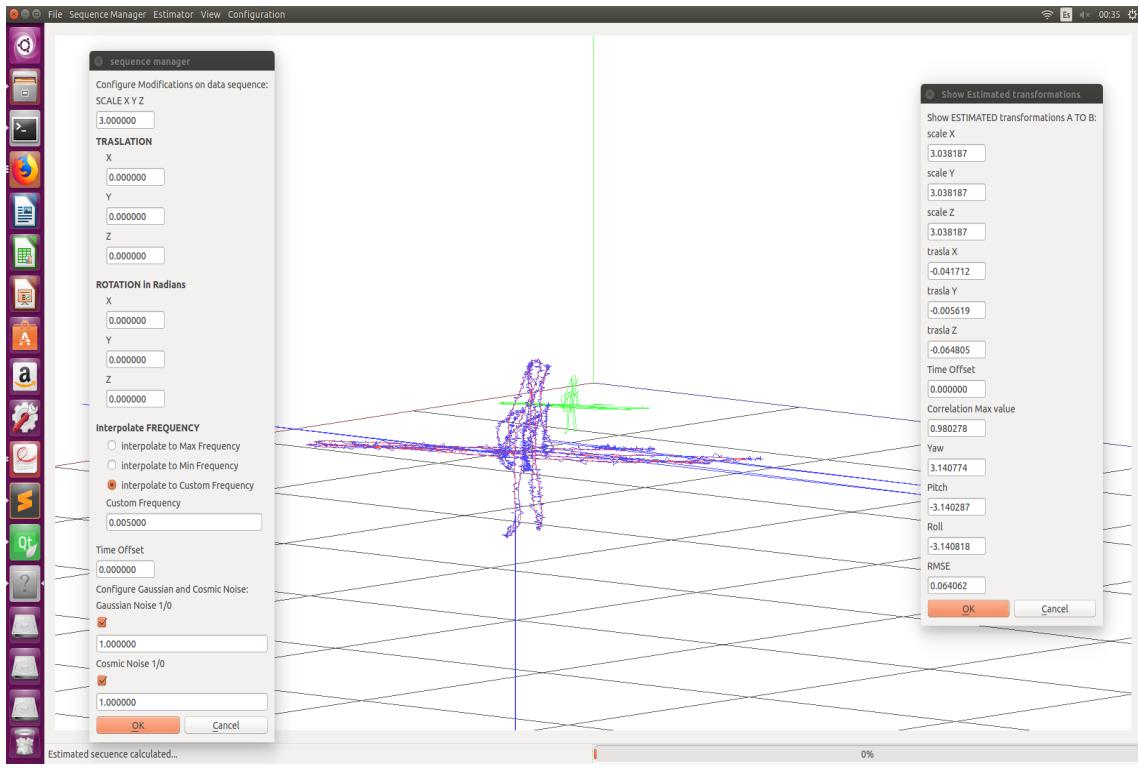
En este caso, al dataset original (color verde) se le ha aplicado un par de transformaciones (en escala y con offset), dando lugar al conjunto azul. En este caso tambien la estimación es buena , ya que el error RMSE da como resultado 0.0 y el cálculo del offset tambien coincide con el valor del offset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



(a)

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

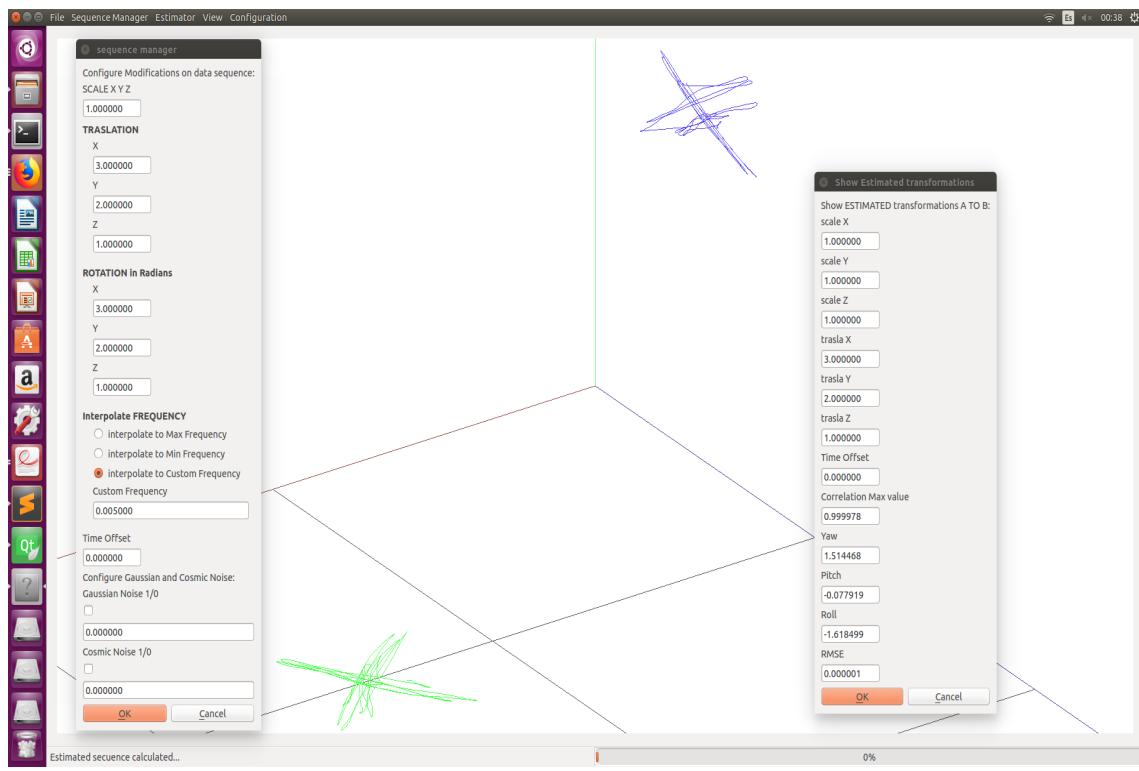
En la imagen superior, se ha aplicado sobre el dataset original un cambio de escala y además se le ha añadido ruido Gaussiano, como puede verse en el dataset resultante (color azul). Las estimaciones obtenidas son buenas, pero el error RMSE calculado entre el dataset transformado y el dataset estimado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



(a)

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

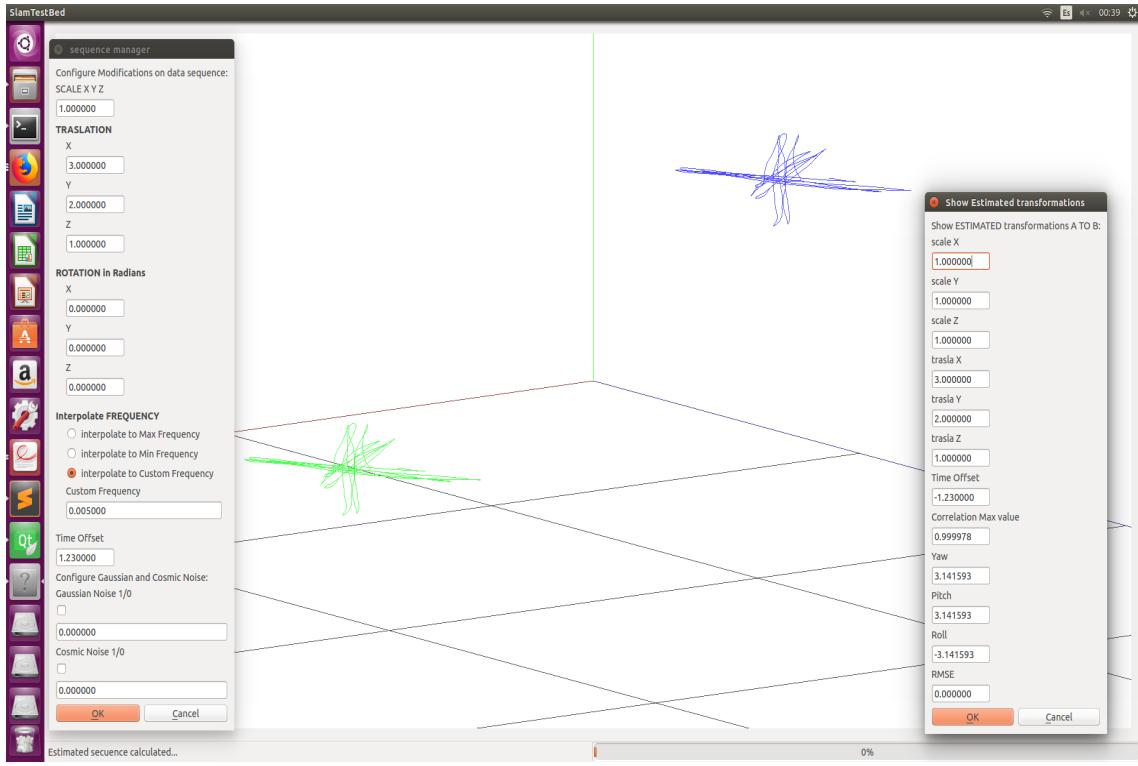
En este caso, se ha aplicado sobre el dataset original un cambio de escala más ruido gaussiano y cósmico. Al tener ahora ruido cósmico, en algunos puntos del dataset transformado aparecen unos valores muy superiores a la media, estos valores desorbitados harán que aumente notablemente el error entre 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.



(a)

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

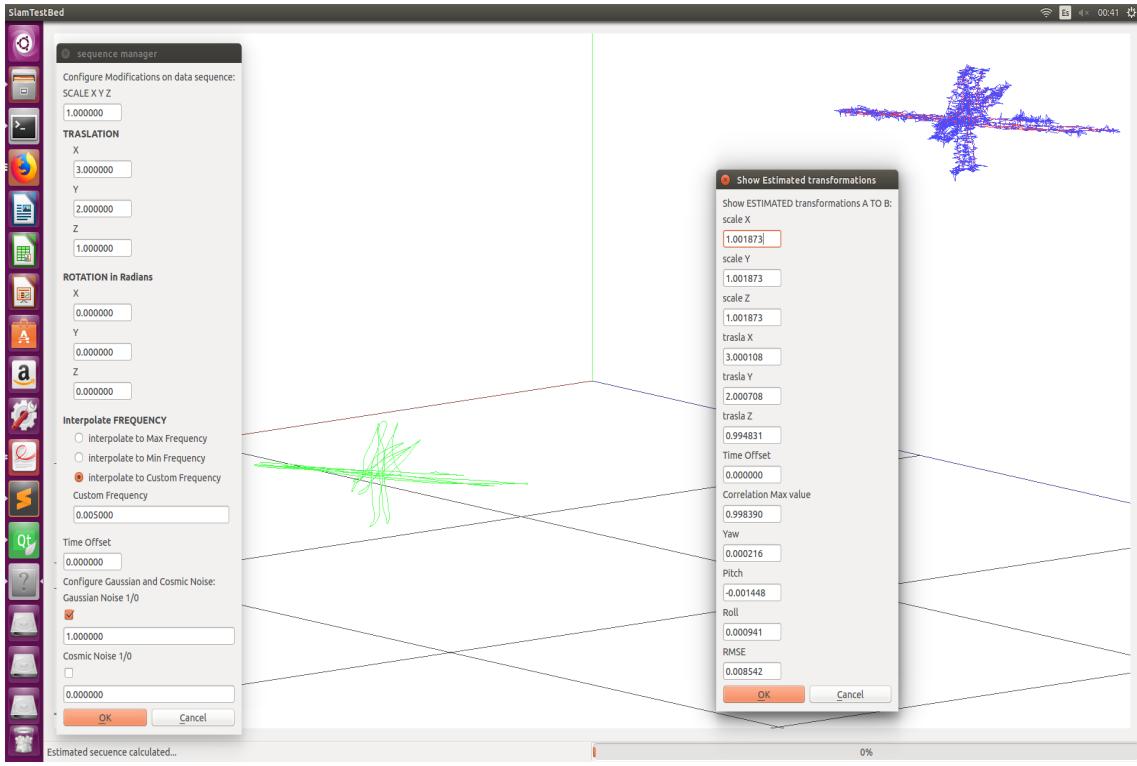
En este test, se han aplicado 2 transformaciones sobre el dataset original, una traslación y una rotación. La estimación de la traslación y la rotación coinciden con los valores de las transformaciones. El error calculado entre el dataset estimado y el dataset transformado es 0.000001. Podríamos considerarlo como 0.0



(a)

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

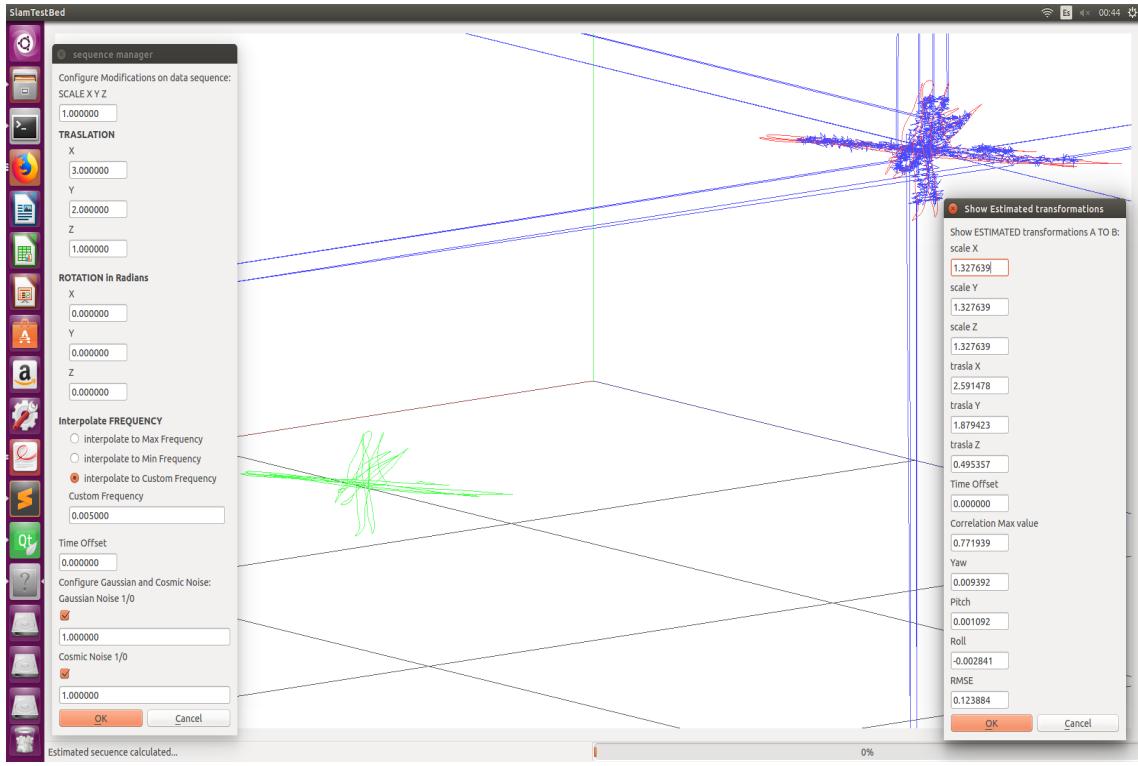
En este caso, al dataset original (color verde) se le ha aplicado un par de transformaciones (en traslación y con offset), dando lugar al dataset transformado (conjunto de puntos de color azul). En este caso tambien la estimación es buena , ya que el error RMSE da como resultado 0.0 y el cálculo del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



(a)

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

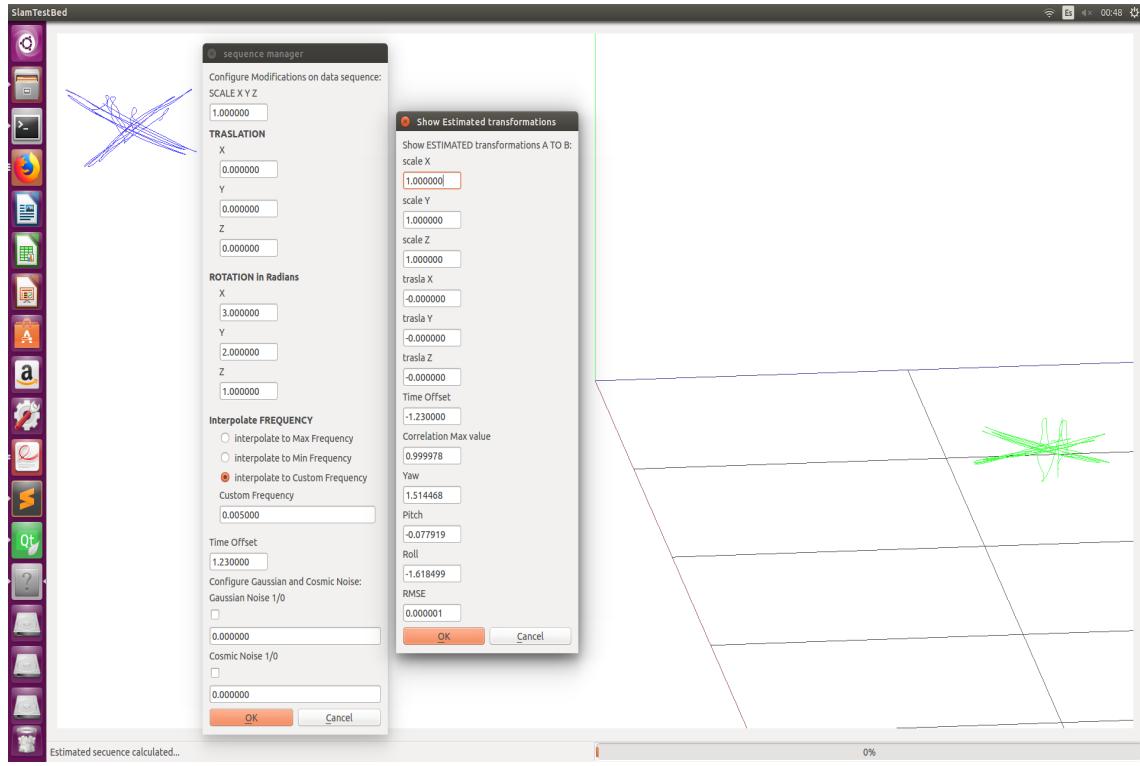
En la imagen superior, se ha aplicado sobre el dataset original un cambio de traslación y además se le ha añadido ruido Gaussiano, como puede verse en el dataset resultante (color azul). Las estimaciones obtenidas son buenas, pero el error RMSE calculado entre el dataset transformado y el dataset estimado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



(a)

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

En este caso, se ha aplicado sobre el dataset original un cambio de traslación 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 5.13: Gráfico que muestra los resultados de la estimación de un cambio de rotación y offset.

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.

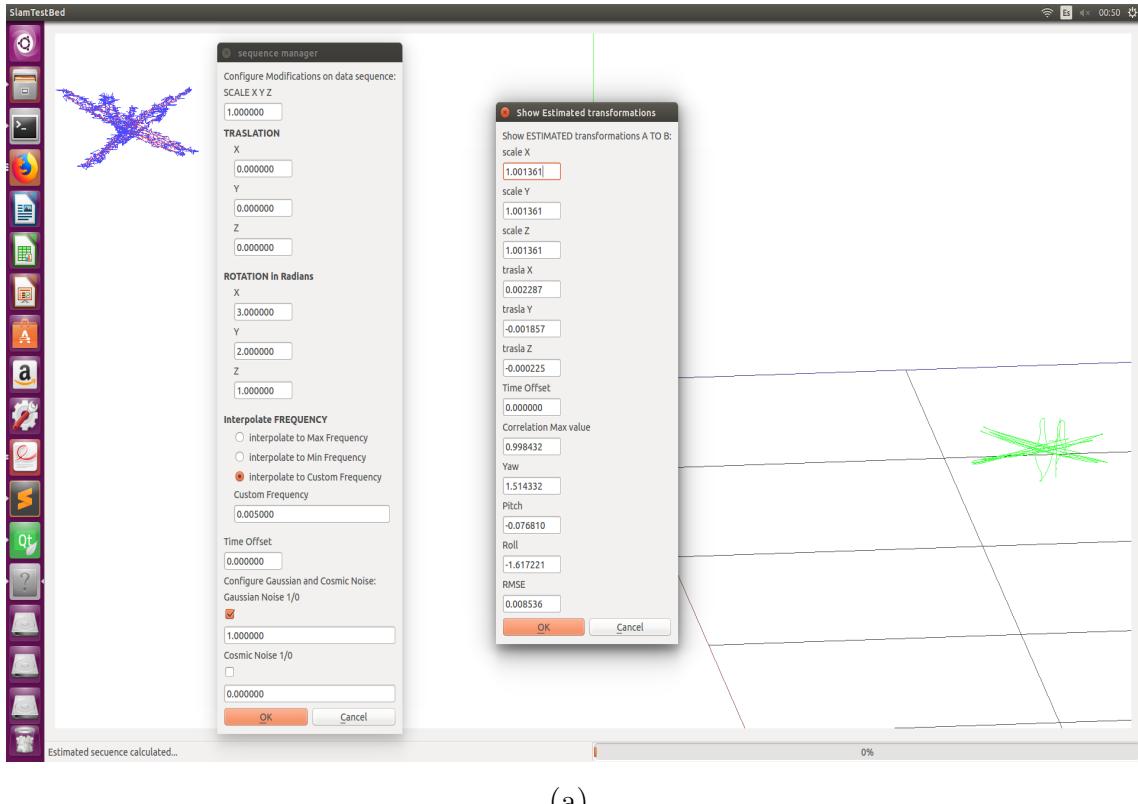
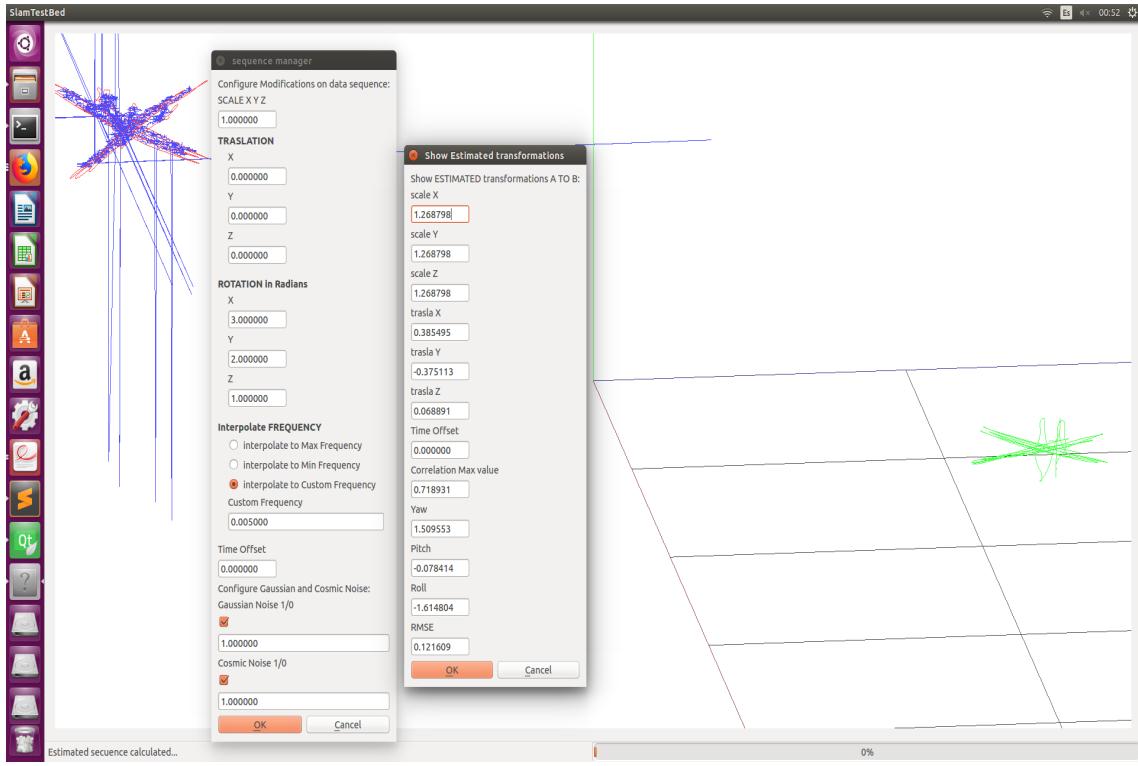


Figura 5.14: Gráfico que muestra los resultados de la estimación de un cambio de rotación y ruido gaussiano.

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 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).



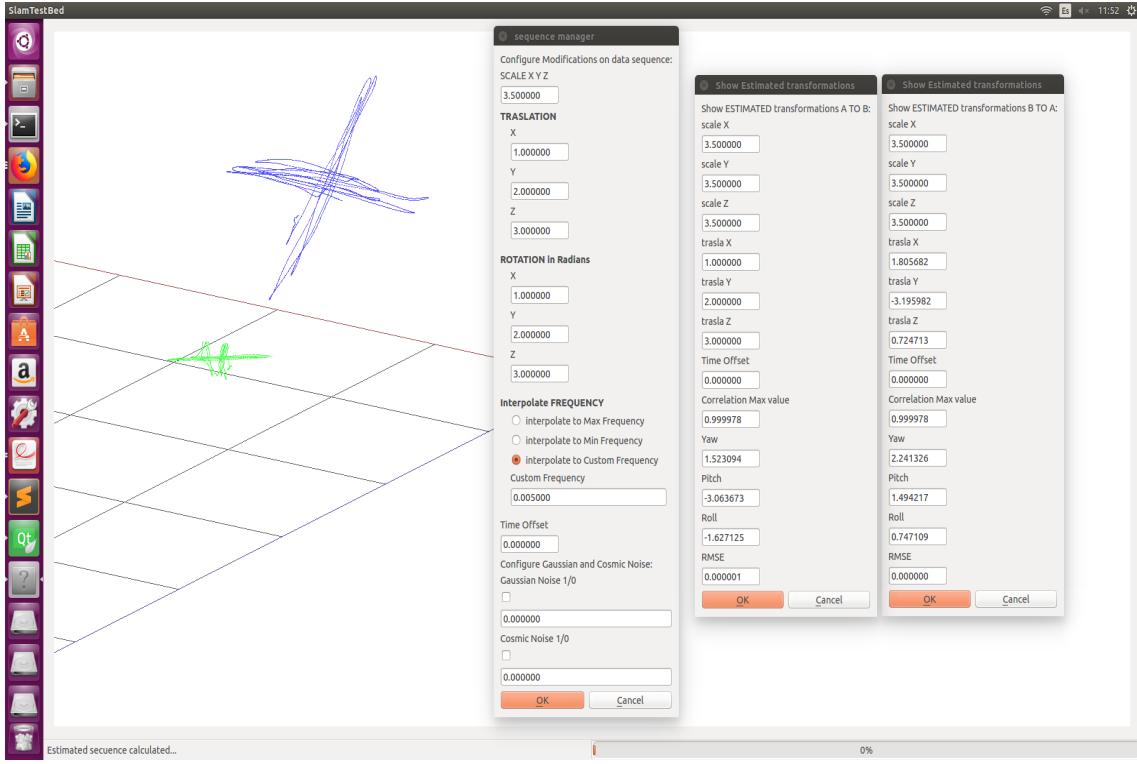
(a)

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

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.

5.4. Pruebas de transformaciones combinadas

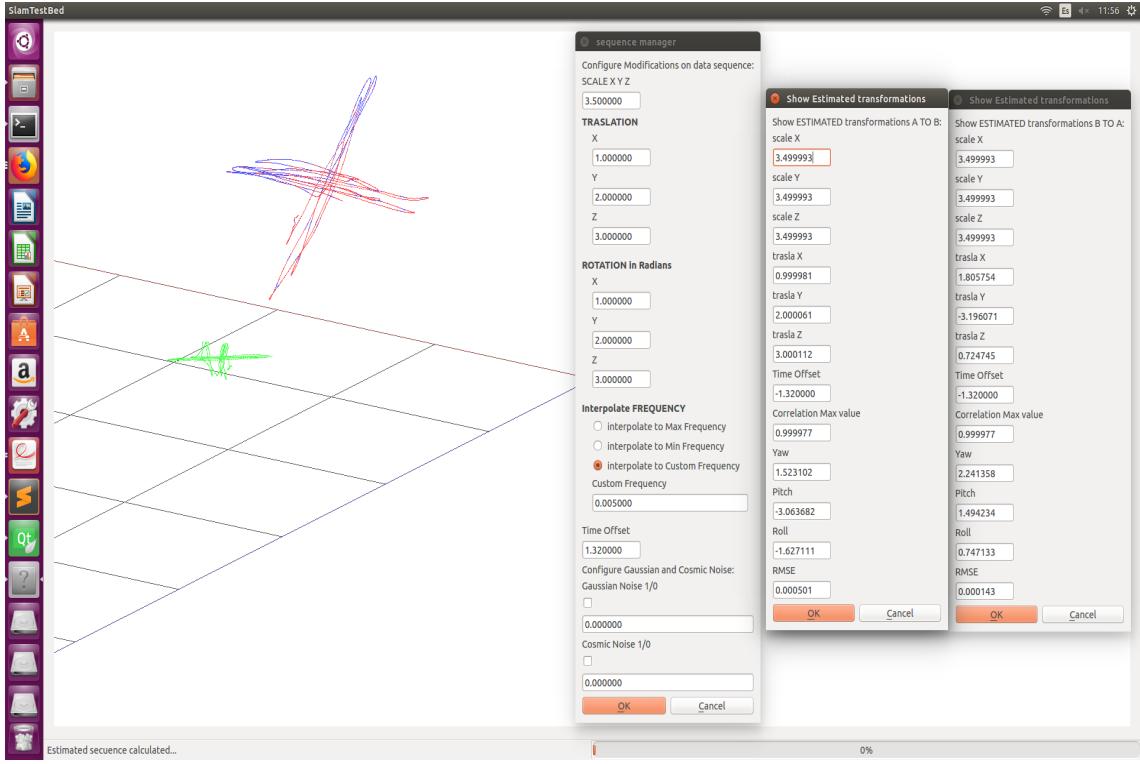
En este apartado mostraremos los resultados de las estimaciones tras realizar varias transformaciones combinadas. En todas se utiliza la interpolación de tiempo a 0.005 segundos



(a)

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

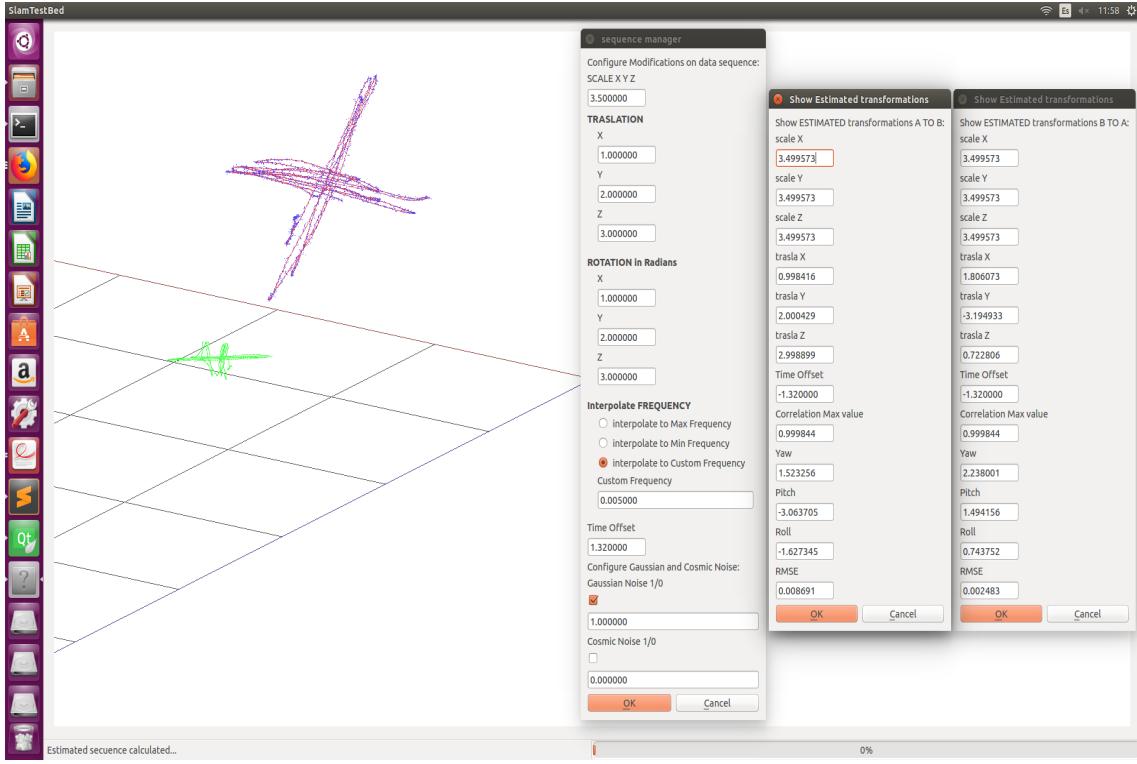
En la imagen superior se realiza la estimación de las transformaciones necesarias para pasar del datasetA al datasetB . En este caso se ha aplicado una transformación combinada de Escala, Traslación y Rotación. En verde puede verse el dataset original y en azul el dataset transformado, en rojo se aprecia 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.



(a)

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

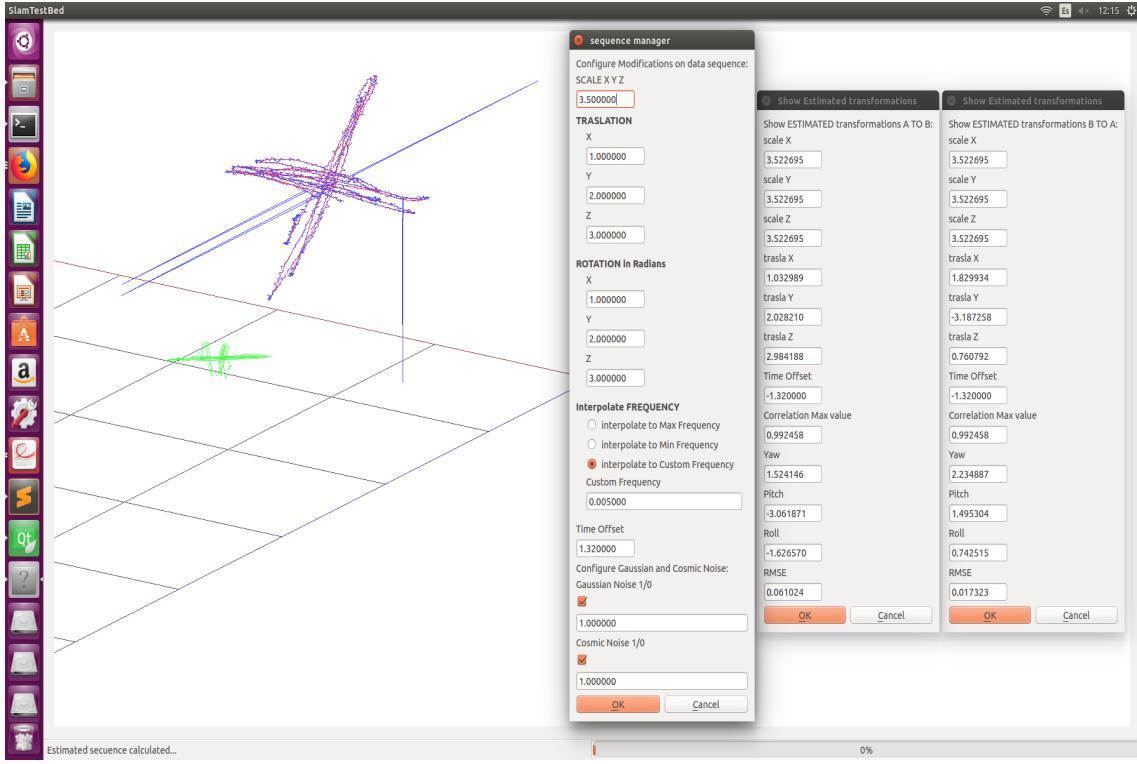
Arriba en la imagen , se presenta el dataset original en verde, el dataset transformado en azul y el dataset estimado en rojo. En este caso se ha realizado la siguiente combinación de transformaciones, Escala, Traslación, Rotación y Offset. Como se puede observar en los resultados de las estimaciones, 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.



(a)

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

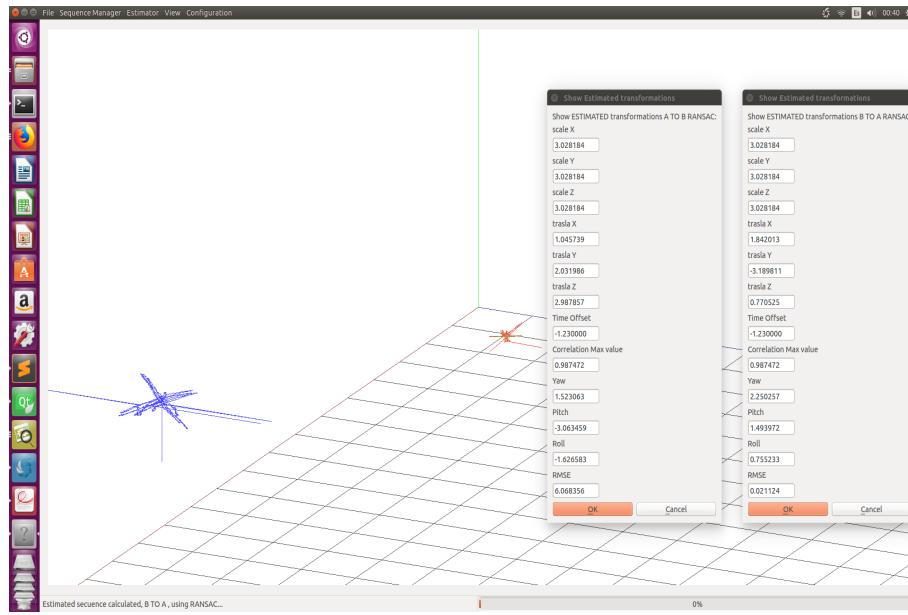
En la imagen superior, podemos ver el dataset original en verde, el dataset transformado en azul y el dataset estimado en rojo. Las transformaciones realizadas en este caso han sido, Escala, Traslación, Rotación, Offset y Ruido Gaussiano. En este caso, al incluir ruido Gaussiano en la transformación 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.



(a)

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 Cómico y además se ha utilizado el método RANSAC para calcular el error.

Capítulo 6

Conclusiones

La robótica móvil es ya una realidad gracias a los algoritmos Visual SLAM que permiten estimar con mínimo error la localización y generación de mapas en entornos desconocidos. En este documento se han descrito algunos de estos algoritmos que ya están funcionando, pero se sigue investigando en la generación de nuevos métodos de navegación autónoma para conseguir mayor fiabilidad, robustez y exactitud de los cálculos.

Dependiendo de las características del entorno o de los requisitos del problema que estemos tratando será más conveniente utilizar un algoritmo u otro. Por ejemplo si necesitásemos generar un mapa de gran exactitud, lo más conveniente sería utilizar DTAM, si por el contrario el mapa no fuese muy importante y la potencia del hardware fuese muy limitada podríamos utilizar SVO.

Por ahora las limitaciones hardware hacen que en robótica móvil se opte por utilizar aquellas técnicas que requieren poca capacidad de cómputo (PTAM,SVO) ya que son fácilmente procesables por los microprocesadores de los actuales robots móviles. En el futuro y a medida que los robots tengan más capacidad de proceso, probablemente se impongan los métodos más robustos que realicen una localización más exacta y cuyos mapas sean muy fiables como podría ser el método ORB-SLAM o DSO.

No obstante todavía queda un camino largo que avanzar en Visual SLAM, ya que algunos algoritmos no son del todo robustos en grandes espacios o entornos donde exista excesivo movimiento alrededor de la cámara, por ejemplo si nuestro robot se encontrase en un jardín frondoso, donde soprase una cierta brisa, le sería difícil al robot mapear el entorno ya que el movimiento de hojas y ramas podría generar inestabilidad en la estimación de la posición 3D de la cámara.

Aunque la gran revolución se producirá cuando la mayoría de smartphones y cámaras

estén equipadas con dispositivos que puedan medir la profundidad de las imágenes, como el proyecto Tango. Sin duda los cálculos de mapeo y posición se acelerarán y mejorará notablemente la exactitud de las estimaciones de posición.

No sería de extrañar que próximamente apareciesen nuevos dispositivos periféricos que pudiesen ser controlados por el Smartphone, por ejemplo un nuevo tipo de aspiradora que no tuviese capacidad para realizar Visual SLAM por si sola, solo un par de motores que le permitan avanzar y girar. Si quisiésemos que esta aspiradora comenzase a aspirar de forma autónoma sólo tendríamos que colocar nuestro Smartphone en posición vertical sobre ella. El smartphone comenzaría a mapear la habitación y a dirigir la navegación de la aspiradora hasta que todo el suelo de la habitación quedase limpio. De esta forma todo el proceso de Visual SLAM de la aspiradora quedaría relegada al Smartphone. Y quien sabe, quizá el futuro de la conducción autónoma dependa de la capacidad con la que estén equipados para realizar Visual SLAM los cada vez más potentes Smartphones.

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.
- [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.
- [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.
- [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.
- [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.