Universidad Zaragoza

Trabajo Fin de Grado

Sistema de localización con cámara RGBd basado en ICP sobre un entorno conocido

English title:

ICP-based RGBd camera location system on a known environment

Autor:

David Turbica Mamblona

Director:

José Luis Villarroel Salcedo

Escuela de Ingeniería y Arquitectura

Zaragoza, Septiembre de 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. David Turbica Mamblona,

con nº de DNI 73412231A en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado en Ingeniería Electrónica y Automática, (Título del Trabajo) Sistema de localización con cámara RGBd basado en ICP sobre un entorno conocido,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 28 de septiembre de 2017

Fdo: DAVID TURBICA MAMBLONA

RESUMEN

Sistema de localización con cámara RGBd basado en ICP sobre un entorno conocido

El objetivo de este proyecto es desarrollar un programa capaz de definir cúal ha sido la trayectoria que ha seguido una cámara RGB-D dentro de un entorno conocido y partiendo de una localización inicial también conocida.

La localización se ha realizado aplicando un algoritmo ICP (Iterative Closest Point) sobre una nube de puntos previa del entorno (mapa previo) y la nube de puntos que proporciona la cámara RGB-D. Este algoritmo iterativo precisa de una estimación inicial para poder converger.

Para ello se dispone además de un IMU o Unidad de Medición Inercial con la que se obtendrá una estimación de la localización de la cámara.

Inicialmente se va a trabajar con una secuencia pregrabada que recoge la información de la cámara y del IMU, y se va a realizar un análisis de tiempos de computación para ver si es posible su migración a una aplicación que se ejecute en tiempo real.

Para poder llevar a cabo todo esto primero se procede a la construcción de un mapa y a la posterior grabación de una secuencia de movimiento sobre ese mapa.

Para la grabación de la secuencia se ha desarrollado, en este caso sí, una aplicación de tiempo real que recoge la información de la cámara y del imu, y genera una estructura de ficheros que puede ser interpretada posteriormente.

La aplicación principal leerá de memoria esa secuencia pregrabada y la procesará emulando una aplicación de tiempo real, analizando los datos en el orden temporal en que fueron recogidos.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS......3
ÍNDICE DE TABLAS......5

Capítulo 1: Introducción......7
1. Contexto y estado del arte......7
2. Objetivos y requisitos......8
2.1. Requisitos de partida......8
2.2. Análisis de objetivos......8
3. Organización de la memoria......10

Capítulo 2: Descripción del hardware y el software......11
1. Hardware......11
1.1. Cámara RGB-D: Xtion PRO LIVE......11
1.2. IMU (Inertial Measurement Unit)......13
2. Software......17
2.1. C++......17
2.2. ROS (Robot Operating System)......17
2.2.1. openni2_camera......18
2.2.2. phidgets_imu......19
2.2.3. imu_filter_madgwick......19
2.3. PCL (Point Cloud Library)......20
2.4. RGBDSlam......20
2.5. Rviz......21
2.6. MATLAB......22

Capítulo 3: Creación del mapa......23
1. Elección del entorno de experimentación......23
2. Construcción del mapa......24

Capítulo 4: Generación de la secuencia de movimiento 27

1. Estructura de datos 27
2. Descripción del software 30
3. Lectura del IMU 31
4. Lectura de la cámara RGB-D 32

Capítulo 5: Procesamiento de la secuencia de movimiento 33

1. Datos de entrada 33
2. Descripción del software 35
3. Procesamiento del mapa 38
   3.1. Voxel Grid Filter 38
   3.2. Eliminación de “outliers” 39
      3.2.1. Statistical Outlier Removal Filter 39
      3.2.2. Radius Outlier Removal Filter 40
4. Estimación de la posición 41
5. Localización del sistema - ICP 50
6. Ejecución 55

Capítulo 6: Pruebas de funcionamiento 57

1. Ratio de captura de datos para crear la secuencia 57
2. Calibración del IMU 59
3. Filtrado de las nubes de puntos 60
4. Análisis de tiempos de cómputo 64
   4.1. Tiempo de cómputo del IMU 65
   4.2. Tiempo de cómputo de la cámara 65

Capítulo 7: Conclusiones y recomendaciones 67

1. Conclusiones 67
2. Recomendaciones 68

BIBLIOGRAFÍA 69

ANEXOS 71

ÍNDICE DE FIGURAS

Figura 1.1. Ejemplo ICP 8
Figura 1.2. Ejemplo de la aplicación del proyecto 9
Figura 2.1. ASUS Xtion PRO LIVE. Cámara RGB-D. Pasiva. [1] 11
Figura 2.2. ASUS Xtion PRO LIVE. Frontal. Partes [1] 12
Figura 2.3. Imagen RGB-D. Imagen RGB. Imagen D. Imagen IR 13
Figura 2.4. Esquema simplificado de un IMU [3] 14
Figura 2.5. IMU Xsense MTw Awinda. [4] 14
Figura 2.6. IMU PhidgetSpatial 3/3/3 [5] 15
Figura 2.7. Ejemplo de aplicación sobre ROS. 18
Figura 2.8. Logo ROS Indigo [6] 18
Figura 2.9. Logo PCL [9] 20
Figura 2.10. Nube de puntos o PointCloud. PC+RGB (izq). PC (der) 20
Figura 2.11. RGBDSlam GUI [10] 21
Figura 2.12. Rviz GUI 21
Figura 3.1. Ejemplos 3D insatisfactorios con RGBDSlam 24
Figura 3.2. Comparación captura automática (izq) y captura manual (der) 25
Figura 3.3. Mapa creado con RGBDSlam utilizado para probar la aplicación 25
Figura 4.1. Ejemplo de secuencia 27
Figura 4.2. Estructura de datos de la secuencia 28
Figura 4.3. Ejemplo clouds_data.txt 28
Figura 4.4. Ejemplo imus_data.txt 29
Figura 4.5. Ejemplo parameters.txt 29
Figura 4.6. Diagrama de flujo de tfg_recorder 30
Figura 4.7. sensor_msgs/Imu [11] 31
Figura 4.8. Diagrama de flujo de procesamiento del IMU 31
Figura 4.9. Diagrama de flujo de procesamiento de la cámara 32
Figura 5.1. Ejemplo "origin.txt" 33
Figura 5.2. Ejemplo "imu_calibration_params.txt" 34
Figura 5.3. Estructura de datos de entrada de "tfg_processor" 34
Figura 5.4. Diagrama de flujo de tfg_processor (a) 35
Figura 5.4. Diagrama de flujo de tfg_processor (b) 36
Figura 5.4. Diagrama de flujo de tfg_processor (c) 37
Figura 5.5. Efecto de "ghosting" 38
Figura 5.6. Ejemplo Voxel Grid Filter [12] 38
Figura 5.7. Ejemplo Outliers en una nube de puntos [13] 39
Figura 5.8. Diagrama de flujo de procesamiento del IMU 41
Figura 5.9. Sistema de referencia solidario al IMU 42
Figura 5.10. Ejemplo de medida de aceleración del IMU 43
Figura 5.11. Análisis frecuencial de las aceleraciones del IMU 43
Figura 5.12. Filtro de media aplicado al IMU. Espectro frecuencial 44
Figura 5.13. Filtro de media aplicado al IMU. Secuencia temporal 44
Figura 5.14. Sistema de referencia del IMU 46

Figura 5.15. Referencias y transformaciones......46
Figura 5.16. Referencias a lo largo de la trayectoria 47
Figura 5.17. Aceleración. Velocidad. Posición......48
Figura 5.18. Integración numérica por trapecios......48
Figura 5.19. Diagrama de flujo de procesamiento de la nube de puntos 50
Figura 5.20. Transformación cámara – IMU......51
Figura 5.21. Resultado de la aplicación de la transformación ${}^{CAM}T_{IMU}$......52
Figura 5.21. Ejemplo de iteración de ICP [16] 53
Figura 5.22. Mensaje de ayuda de tfg_processor 55
Figura 6.1. Mapa a resolución original (izq) y filtrado con x = 0.05 (der)......60
Figura 6.2. Aplicación del filtro "Statistical Outlier Removal Filter" 62
Figura 6.3. Sobredimensionamiento de los parámetros del filtro "Statistical Outlier Removal Filter" 62
Figura 6.4. Eliminación de puntos aislados......63
Figura 6.5. Resultado de ejecución de tfg_processor (a)......64
Figura 6.5. Resultado de ejecución de tfg_processor (b)......65

ÍNDICE DE TABLAS

Tabla 2.1. Especificaciones IMU PhidgetSpatial [5] 15
Tabla 2.2. Topics de openni2_camera 19
Tabla 2.3. Topics de Phidgets_imu 19
Tabla 2.4. Topics de imu_filter_madgwick 19
Tabla 5.1. Cálculo de los offsets del IMU 45
Tabla 6.1. Resultados test de multiprocesos...57
Tabla 6.2. Medias de calibración de los acelerómetros del IMU 59
Tabla 6.3. Análisis de tiempos para diferentes resoluciones 61
Tabla 6.4. Tiempos de procesamiento de la nube de puntos 66

CAPÍTULO 1

Introducción

1. CONTEXTO Y ESTADO DEL ARTE

En los últimos años ha surgido un creciente interés por los UAVs (Unmanned Aerial Vehicle, en español, Vehículo Aéreo no Tripulado) y su navegación autónoma.

A diferencia de los dispositivos móviles terrestres, no disponen de ningún contacto físico con el entorno que permita la obtención de la odometría (como por ejemplo las ruedas). Por tanto se deben buscar maneras alternativas de conocer el movimiento del UAV.

En entornos abiertos, exteriores, se utiliza habitualmente el GPS para este propósito. Sin embargo, el GPS deja de funcionar en interiores debido a la pérdida.

Habitualmente la localización en interiores se realiza con sistemas de visión con cámaras estáticas situadas en las estancias en las que se mueve el UAV. Pero, obviamente, esto implica la instalación de dichas cámaras y dificulta la portabilidad de esta solución a otros entornos.

Por tanto, la solución óptima parece apuntar a que el sistema de localización debe ir embarcado en el propio UAV. Según foros especializados, los sistemas inerciales se comportan realmente mal a la hora de calcular trayectorias. La otra alternativa es utilizar algún sistema de visión por computador para reconocer el entorno.

A partir de las imágenes se puede localizar el UAV, de manera absoluta, si se tiene un conocimiento previo de la estancia en la que se encuentra, o de manera relativa, respecto de la ultima imagen que se tiene.

Habitualmente se utilizan representaciones 3D del entorno en el que se mueve el UAV y se está trabajando en la idea de mejorar los sistemas de localización de drones en espacios cerrados, basándose en un proceso conocido como ICP, al que además se incorpora la medida de un sensor inercial (IMU).

El presente TFG es una primera implementación de esta técnica en el grupo de Robótica de la Universidad de Zaragoza.

2. OBJETIVOS Y REQUISITOS

El objetivo principal es desarrollar un algoritmo basado en ICP (Iterative Closest Point) que, a partir de la información obtenida de una cámara RGB-D y un IMU (Inertial Measurement Unit), permita localizar un sistema móvil dentro de una nube de puntos correspondiente a un entorno conocido. Posteriormente, estudiar la posibilidad de implementar el algoritmo en un computador embarcado en un UAV (Unmanned Aerial Vehicle).

2.1. Requisitos de partida

El material necesario para llevar a cabo el proyecto ha sido proporcionado por el Grupo de Investigación en Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza. Este material consta de una cámara RGB-D modelo ASUS Xtion PRO LIVE y un IMU modelo PhidgetSpatial 3/3/3. El software deberá adaptarse a este hardware.

Además se darán por hecho ciertos supuestos. Como son: la localización inicial sistema, que es conocida, y la existencia de un mapa previo.

2.2. Análisis de objetivos

El proceso de localización comienza capturando un fotograma de la cámara RGB-D. El algoritmo debe ser capaz de localizar lo que está viendo en una escena que previamente conoce. Entendiendo por escena el entorno en el que se encuentra la cámara.

Al tratarse de una cámara RGB-D, se tiene la información de profundidad de la imagen y permite trabajar directamente con nubes de puntos en vez de imágenes planas. Esto implica que se puede aplicar un algoritmo ICP para localizar la imagen en la escena. El ICP es un algoritmo iterativo que trata de alinear una nube de puntos con otra para encontrar la mejor transformación que relaciona una con otra. La librería PCL proporciona varias implementaciones de este algoritmo.

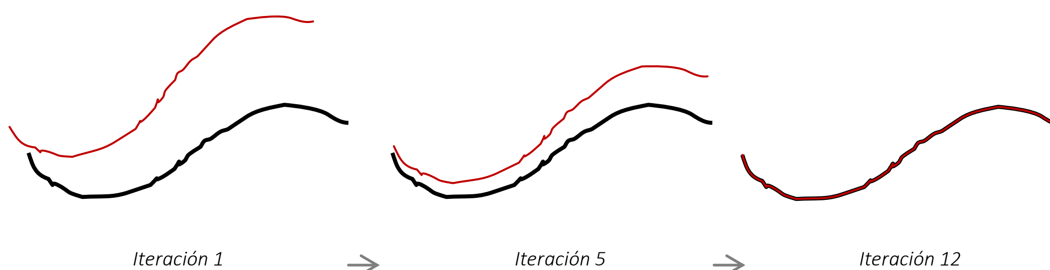

Figura 1.1. Ejemplo ICP

Pero el algoritmo ICP debe converger a una solución final, y para ello es necesario partir de una buena estimación inicial. En otras palabras, se le debe indicar dónde debe buscar para encontrar lo que busca.

La manera de obtener esa estimación inicial es a través del IMU. Conocida la última localización del sistema, el tiempo transcurrido y las aceleraciones que entrega el IMU, se puede integrar en el tiempo para conocer la velocidad del sistema en cada instante, y volver a integrar para conocer la posición del sistema. No es un proceso preciso, dada la doble integración, pero sirve como estimación.

De esta manera, el sistema va calculando una estimación de su localización a partir del IMU y cada cierto periodo de tiempo, utiliza la imagen capturada por la cámara para relocalizarse con precisión. En la figura 1.2. la línea negra sería la trayectoria real del sistema, la línea azul, la estimación del IMU y la línea roja, la corrección de la cámara. Los únicos puntos en los que se conoce la localización real son los puntos rojos.

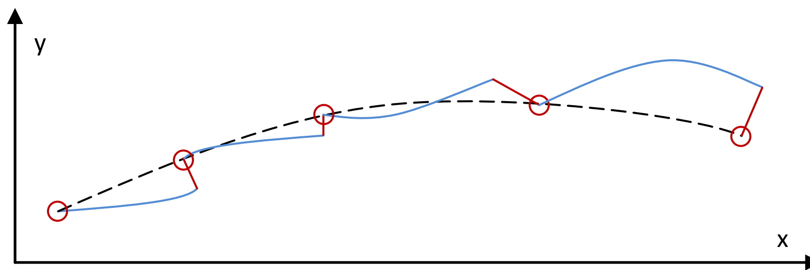

Figura 1.2. Ejemplo de la aplicación del proyecto

Ahora bien, todo esto se va a hacer en post-procesado, lo que implica una obtención previa de datos. Por tanto, es necesario elaborar un programa que grabe una secuencia de imágenes y aceleraciones que posteriormente puedan ser analizadas. Para grabar esta secuencia es necesario obtener los datos de los sensores. Los drivers necesarios para ello (openni2_camera y phidgets_imu) están diseñados como nodos de ROS. ROS o Robot Operating System es el entorno virtual que va a gestionar las comunicaciones entre los sensores y el programa. Por tanto, la aplicación se desarrollará como un nodo de ROS.

Como ya se ha comentado, se necesita de un mapa sobre el que localizarse: la escena. Por tanto se debe construir un mapa. La herramienta que se usa para ello es: RGBDSlam, que a partir de una cámara RGB-D puede construir una nube de puntos con los sucesivos fotogramas. Nuevamente, este programa se apoya también en ROS.

Una vez obtenido el mapa y la secuencia, se pasa al procesamiento de la misma y análisis de resultados, en cuanto a tiempo de cómputo y precisión de la aplicación.

3. ORGANIZACIÓN DE LA MEMORIA

Este documento se ha dividido en siete capítulos:

- Capítulo 1. Introducción.

  Se trata de este mismo capítulo, con el que se pretende situar al lector dentro del contexto del proyecto.

- Capítulo 2. Descripción del hardware y el software.

  En este capítulo se ofrece una introducción a los diferentes componentes hardware utilizados en el proyecto, así como una breve explicación de las herramientas software que se han empleado.

El proyecto se ha dividido en tres etapas bien diferenciadas: creación del mapa, generación de la secuencia y, finalmente, procesamiento de la secuencia. En los tres capítulos siguientes se describe en que consiste cada una de estas etapas y como se han llevado a cabo.

- Capítulo 3. Creación del mapa.

  En este capítulo se contesta al cómo hacer el mapa y sobre qué hacerlo.

- Capítulo 4. Generación de la secuencia.

  En este capítulo se describe el funcionamiento del programa tfg_recorder.

- Capítulo 5. Procesamiento de la secuencia.

  En este capítulo se describe el funcionamiento del programa tfg_processor.

En estos dos últimos capítulos se analizan los resultados obtenidos.

- Capítulo 6. Pruebas de funcionamiento.

  En este capítulo se presentan las distintas pruebas realizadas sobre el sistema y sus respectivos resultados.

- Capítulo 7. Conclusiones y recomendaciones.

  En este capítulo se presentan los resultados obtenidos y se proponen vías de mejora y de continuación.

Además se incorpora a modo de anexos el código completo de las aplicaciones.

CAPÍTULO 2

Descripción del hardware y el software

En este capítulo se ofrece una introducción a los diferentes componentes hardware utilizados en el proyecto, así como una breve explicación de las herramientas software que se han empleado.

1. HARDWARE

Para llevar a cabo el desarrollo de la aplicación se ha hecho uso de los siguientes componentes hardware: una cámara RGBd y una unidad de medición inercial o IMU.

1.1. Cámara RGB-D: Xtion PRO LIVE

Una cámara RGB-D es una cámara digital capaz de ofrecer información de color (RGB) y profundidad (D) para cada pixel de la imagen. La información de color se captura con una cámara digital convencional y suele codificarse en el espacio de color RGB por extensión de su nombre. La información de profundidad puede obtenerse con diversas tecnologías. Estas tecnologías pueden clasificarse en activas o pasivas [2].

Figura 2.1. ASUS Xtion PRO LIVE. Cámara RGB-D. Pasiva. [1]

Se entiende por tecnologías pasivas aquellas en las que la cámara únicamente recibe luz. Un ejemplo son las cámaras estereoscópicas. Estas constan de dos cámaras digitales con una transformación conocida entre ellas. Simultáneamente se captura la imagen con las dos cámaras y se computa la diferencia entre ellas para obtener la información de profundidad. Estas cámaras dependen de la iluminación externa para poder funcionar. [2]

Por el contrario, se entiende por tecnologías activas aquellas en las que la cámara no solo recibe luz sino que además la emite. La principal ventaja es que no se necesita de una fuente de luz externa para que funcionen. En este caso, se tiene una cámara digital para capturar el color y otro sistema para obtener la profundidad. Se utilizan dos métodos distintos: El primero, emitiendo una luz y midiendo el tiempo que tarda en reflejarse y volver. Y el segundo, emitiendo un patrón de luz y viendo al distorsión que aparece en el patrón tras ser reflejado por la superficie. En definitiva, este segundo método trata de emular una cámara estereoscópica. [2]

Habitualmente estos sensores activos emiten luz infrarroja dado que esta no es visible por el ser humano, sin embargo, esto limita el uso de estas cámaras a espacios cerrados o interiores donde no incide directamente la luz solar. Esto se debe a que el sol también emite en el espectro infrarrojo y puede cegar al sensor. [2]

Para el desarrollo de este proyecto se ha utilizado la cámara ASUS Xtion PRO LIVE. Un modelo de cámara RGB-D activa de patrón infrarrojo, producida por la compañía ASUSTeK Computer Inc. Se trata de una cámara diseñada para su uso en interiores y posee un rango de funcionamiento entre los 0.8 m y los 3.5 m. Tiene una resolución de 640x480 px. grabando a 30fps y puede reducirse a la mitad, 320x240 px. para alcanzar los 60 fps. No obstante, para el desarrollo del proyecto se ha usado la primera configuración. El campo de visión que ofrece la cámara es de 58° en horizontal, 45° en vertical y 70° en diagonal [1].

En la figura 2.2. se indican las distintas partes y sensores que componen la cámara. Como se puede ver, dispone además de dos micrófonos, uno en cada extremo, para grabar audio en estéreo pero no se ha hecho uso de ellos.

Figura 2.2. ASUS Xtion PRO LIVE. Frontal. Partes [1]

La cámara tiene conexión USB2.0 y ROS pone a nuestra disposición un nodo para comunicar la cámara con el ordenador, como ya detallaremos más adelante en este mismo capítulo.

Además esta cámara cuenta con un procesador integrado para procesar la imagen de profundidad en la propia cámara.

En la figura 2.3. se pueden ver los tres tipos de imágenes que entrega la cámara. La primera es la imagen RGB en color. La segunda es la imagen D que muestra la profundidad de la escena en escala de grises, de tal manera que los tonos más oscuros están más cerca y los más claros al fondo. Y la tercera es la imagen captada por sensor de infrarrojos. Se puede apreciar un punteado de la imagen que se corresponde con el patrón emitido por el emisor de infrarrojos.

Figura 2.3. Imagen RGB-D. Imagen RGB. Imagen D. Imagen IR

Nótese además, aprovechando que se puede apreciar bien en la segunda imagen de la figura 2.3, otro de los problemas que presentan este tipo de cámaras. Se ha comentado que en la imagen de profundidad los colores más oscuros indican cercanía, hasta llegar al negro puro, que se reserva para aquellos pixeles en los que es imposible determinar la distancia. Estos puntos negros aparecen con bastante frecuencia en zonas como bordes o esquinas donde el patrón de infrarrojos se refleja de una manera que le que le es imposible al receptor capturarlo.

Además el espectro infrarrojo no interacciona con la materia de la misma manera que el espectro visible. Es el caso, por ejemplo, de la botella de plástico. Debido al material y a su curvatura, distorsionan el patrón infrarrojo de una manera que la cámara no es capaz de procesar.

1.2. IMU (Inertial Measurement Unit)

Un IMU (Inertial Measurement Unit), en español, Unidad de Medición Inercial, es un dispositivo electrónico dotado de una serie de acelerómetros, giróscopos y, en ocasiones, magnetómetros, que ofrece información acerca de las aceleraciones a las que está sometido, las velocidades angulares y, en ocasiones, la orientación.

Las magnitudes básicas que mide un IMU son aceleración lineal y velocidad angular. A partir de ahí, cada fabricante y cada modelo ofrece otras magnitudes secundarias derivadas de las primeras que varían en función del precio y calidad del sensor, pudiéndose llegar a tener valores de posición espacial.

En la figura 2.4. se puede ver la estructura más simple que puede componer un IMU, en cuanto a la medición de magnitudes se refiere. A este esquema habría que añadirle toda la circuitería asociada a la adquisición y adecuación de la señal, así como el procesador o procesadores para tratar los datos obtenidos y la etapa de comunicaciones para extraerlos, ya sea RS-232, USB, bluetooth, etc.

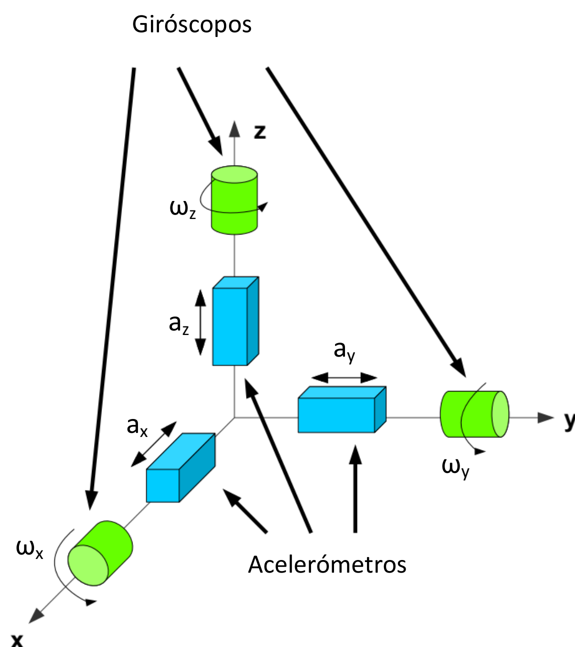

Figura 2.4. Esquema simplificado de un IMU [3]

Para este proyecto se disponía inicialmente del IMU MTw Awinda, un producto de la compañía Xsense. Se trata de un IMU con conexión wireless y una alta precisión, diseñado para su uso en aplicaciones biomédicas. En la figura 2.5. se ve, el IMU (naranja) y el receptor inalámbrico (negro). Sin embargo, no se ha podido usar finalmente dada la imposibilidad de conectarlo al ordenador sin utilizar el propio programa de la casa Xsense. Tampoco estaban a disposición drivers de terceros y Xsense no ofrecía al usuario el protocolo de comunicación como para poder desarrollar esos drivers por cuenta propia.

Figura 2.5. IMU Xsense MTw Awinda. [4]

Tras varios intentos fallidos de utilizar el IMU de Xsense, se optó por buscar una alternativa: el IMU PhidgetSpatial Precision 3/3/3 High Resolution. Algo menos preciso que el anterior pero mucho más económico. En este caso utiliza una conexión USB2.0.

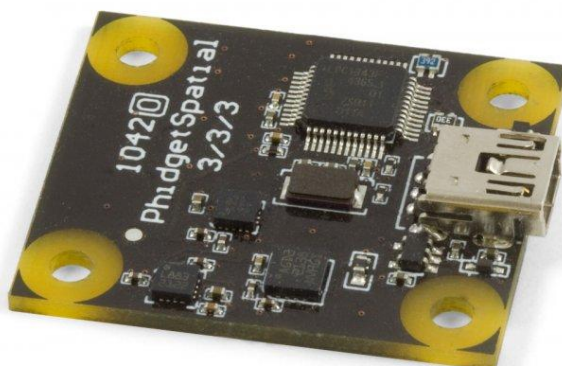

Figura 2.6. IMU PhidgetSpatial 3/3/3 [5]

Se trata de un IMU básico en cuanto a la información que proporciona. Se compone de un acelerómetro, un giróscopo y un magnetómetro por eje, entregando aceleración lineal, velocidad angular y campo magnético. En la tabla 2.1. se muestran algunas de sus especificaciones.

Acelerómetro	
Resolución	976.7 μg
Rango máximo	± 8 g
Acelerómetro de precisión	
Resolución	76.3 μg
Rango máximo	± 2 g
Giróscopo	
Resolución	0.07 °/s
Rango máximo	± 2000 °/s
Giróscopo de precisión	
Resolución (eje X, eje Y)	0.02 °/s
Resolución (eje Z)	0.013 °/s
Rango máximo (eje X, eje Y)	± 400 °/s
Rango máximo (eje Z)	± 300 °/s
Magnetómetro	
Resolución	3 mG
Rango máximo	5.5 G

Tabla 2.1. Especificaciones IMU PhidgetSpatial [5]

Como se puede ver en la tabla 2.1. tanto el acelerómetro como el giróscopo, se dividen en dos. Diferenciando el sensor por defecto y otro de precisión. Esto se debe a que internamente, el IMU dispone de dos modos de funcionamiento de tal forma que activa el modo de precisión en el acelerómetro cuando la aceleración es menor de 2 g y activa el modo de precisión en el giróscopo cuando la velocidad angular es menor de 100 °/s. El cambio de modo se realiza internamente y no es apreciable ni manipulable desde el exterior.

2. SOFTWARE

Para llevar a cabo el desarrollo de la aplicación se ha hecho uso de las siguientes herramientas software: C++ , ROS (Robot Operating System), PCL (Point Cloud Library), MATLAB.

2.1. C++

Todo el proyecto se ha programado utilizando C++ como lenguaje de programación. Un lenguaje orientado a objetos que permite utilizar estructuras más complejas de datos.

2.2. ROS (Robot Operating System)

Como se describe en la propia página oficial de ROS.org (Traducido al español): "Robot Operating System (ROS) es un entorno para el desarrollo de software para robots. Es un conjunto de herramientas, librerías y convenciones que tratan de simplificar la tarea de crear rutinas complejas y robustas sobre una gran variedad de robots".

> *"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms" [6]*

ROS es un pseudo sistema operativo de código abierto. No es un sistema operativo al uso porque no se ejecuta directamente sobre el hardware del computador, necesita de un sistema operativo sobre el que ejecutarse. Por ahora solo está disponible para plataformas basadas en Unix. En este caso se ha utilizado Ubuntu 14.04. para realizar las pruebas. ROS emula un sistema operativo en el sentido de que gestiona sus propios procesos e interrupciones.

ROS se fundamenta en tres pilares fundamentales: el "master" (maestro), los "nodes" (nodos) y los "topics" (tópicos). Se utilizarán ambas nomenclaturas, inglesa y española indistintamente..

El master es el proceso principal. Es el "sistema operativo" en sí. Se encarga de gestionar la ejecución de todos los nodos, las publicaciones y lecturas sobre los topics, y en definitiva todas las interacciones entre los anteriores. Sin el máster no habría comunicación entre los nodos.

Los nodos son cada uno de los procesos que se ejecutan sobre el master. Permiten modularizar la ejecución de una aplicación compleja dividiéndola en diferentes nodos que además pueden ejecutarse en diferentes computadores.

Los topics son los canales de comunicación entre los nodos. Estos manejan unas unidades de datos conocidas como "messages" (mensajes), de tal manera que un nodo puede publicar en un topic un mensaje o subscribirse a ese topic para recibir el mensaje. La lectura y escritura en estos topics la gestiona el master.

En la figura 2.7. se puede ver un ejemplo gráfico del funcionamiento de una aplicación ejecutándose sobre ROS. En círculos los nodos, en cuadrados los topics y las flechas indican el sentido de

la información. Por ejemplo, "Lectura encoders" sería un nodo que se publica en el topic "ODOMETRÍA" y "Planificación trayectoria" sería un nodo que se subscribe a "ODOMETRÍA" para recoger el mensaje.

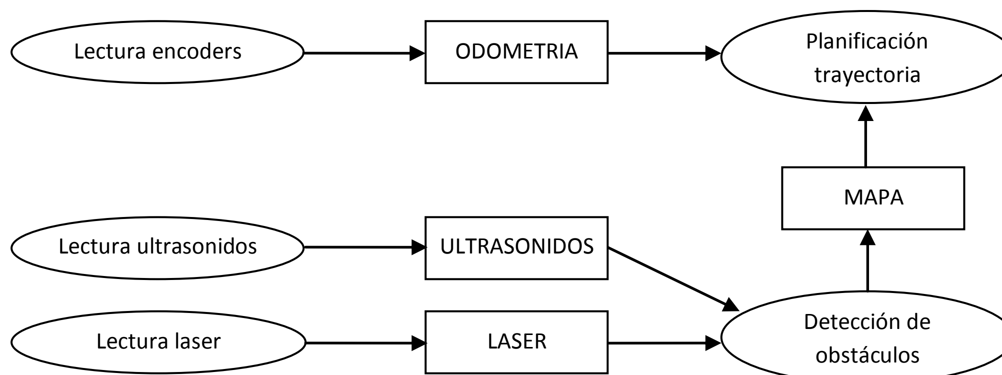

Figura 2.7. Ejemplo de aplicación sobre ROS.

Existen distintas distribuciones de ROS, es decir, diferentes versiones. Para el desarrollo del proyecto se ha usado la distribución ROS Indigo Igloo.

Figura 2.8. Logo ROS Indigo [6]

Una de las ventajas que tiene ROS es que su uso está bien extendido en el mundo de la robótica y existe multitud de nodos ya creados por la comunidad que permiten la comunicación con diversos hardwares, drivers en definitiva, que pueden ser implementados en nuestra aplicación. A continuación se explican cuales son los nodos ofrecidos por ROS que se han utilizado en este proyecto.

2.2.1. openni2_camera

Este paquete contiene los nodos necesarios para poder utilizar hardware compatible con OpenNI. En este caso es necesario para poder comunicarnos con la cámara RGB-D ASUS Xtion PRO LIVE. Contiene los drivers de bajo nivel para extraer las imágenes de la cámara y publica un gran numero de topics con los diversos tipos de datos que pueden obtenerse de la cámara.

Algunos de los topics que ofrece se pueden ver en la tabla 2.2.

/camera/depth/image	Imagen de profundidad
/camera/depth/points	Nube de puntos
/ camera/depth_registered/points	Nube de puntos + RGB
/ camera/ir/image	Imagen de infrarrojos
/ camera/rgb/image_raw	Imagen RGB

Tabla 2.2. Topics de openni2_camera

Una de las principales utilidades que tiene es que proporciona la escena directamente en formato nube de puntos, en este caso utilizando el tipo de ROS "sensor_msg/PointCloud2". Para el propósito de este proyecto interesa la información de nube de puntos pero sin necesidad de conocer el color de cada punto. Por ello, el topic que se ha usado es: "/camera/depth/points". Al subscribirse a este topic, la estructura de datos es menos pesada, pues no precisa de canales RGB, y el procesamiento, algo más rápido.

2.2.2. phidgets_imu

Este paquete contiene los drivers para el IMU PhidgetSpatial 3/3/3. Su funcionamiento es simple: lee y publica los valores que capturan los sensores. En la tabla 2.3. se muestran los topics en los que publica.

/imu/data_raw	Aceleración lineal + Velocidad angular
/imu/mag	Campo magnético
/ imu/is_calibrated	Flag para indicar el fin de calibración

Tabla 2.3. Topics de Phidgets_imu

2.2.3. imu_filter_madgwick

Este paquete se fundamenta en el paquete anterior (phidgets_imu). Filtra y combina la información de aceleración angular, velocidad lineal y campo magnético para entregar la orientación del IMU. Lo hace en forma de cuaternio.

En la tabla 2.4. se puede ver el único topic en el que publica. Utiliza el la estructura de datos de ROS: "sensor_msg/Imu" donde se unifican: orientación en forma de cuaternio, velocidad angular y aceleración lineal.

/imu/data	Acel. lineal + Vel. angular + Orientación

Tabla 2.4. Topics de imu_filter_madgwick

Una de las ventajas que tiene utilizar la información de campo magnético para determinar la orientación es que esta siempre estará referenciada respecto del mismo sistema de coordenadas, determinado por el norte y la gravedad. Incluso tras hacer reset al IMU.

2.3. PCL (Point Cloud Library)

La PCL o Point Cloud Library es un conjunto de librerías bajo la licencia BSN dedicadas al procesamiento de imágenes en 2D, 3D y nubes de puntos. Contiene los últimos algoritmos (state-of-art) para el filtrado, la estimación de features, el mapeo, la identificación y la segmentación de imágenes y nubes de puntos [9].

Figura 2.9. Logo PCL [9]

En el desarrollo de este proyecto se han utilizado estas librerías para tener una estructura base con la que trabajar: las "PointClouds", y para el filtrado, redimensionado, seccionado e identificación de esas nubes de puntos.

Cabe definir pues, qué es una nube de puntos o PoinCloud. Se trata de una estructura de datos multidimensional que generalmente toma tres dimensiones para representar puntos en el espacio tridimensional (x,y,z) o cuatro dimensiones si se incluye el color. En la figura 2.10. se puede ver un ejemplo para cada caso. En la nube de puntos de la derecha se representa con una escala de color la profundidad de la escena.

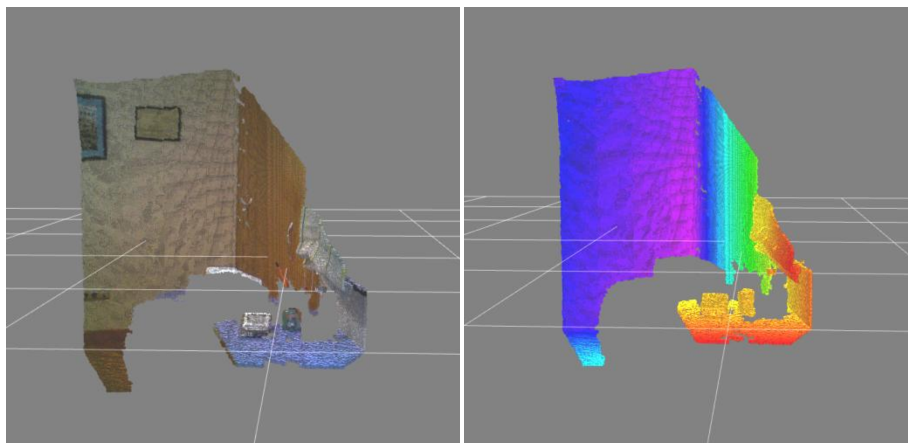

Figura 2.10. Nube de puntos o PointCloud. PC+RGB (izq). PC (der)

2.4. RGBDSlam

Se trata de un paquete creado para funcionar sobre ROS (Robot Operating System) sin embargo, dado que se trata de una aplicación con GUI propia creo conveniente tratarlo como un software a parte, y no incluirlo con los otros paquetes mencionados en el apartado 2.2. de este capítulo.

Esta aplicación permite crear un mapa a partir de una cámara RGB-D por composición de las sucesivas nubes de puntos que va capturando. Utiliza puntos de interés y descriptores SIFT para establecer relaciones entre los fotogramas.

Para el desarrollo del proyecto se precisaba de un mapa previamente creado. En este caso se ha utilizado RGBDSlam para ello pero podría tratarse de un mapa generado por cualquier otra técnica de mapeo. En la figura 2.11. se ofrece una imagen de la GUI del RGBDSlam.

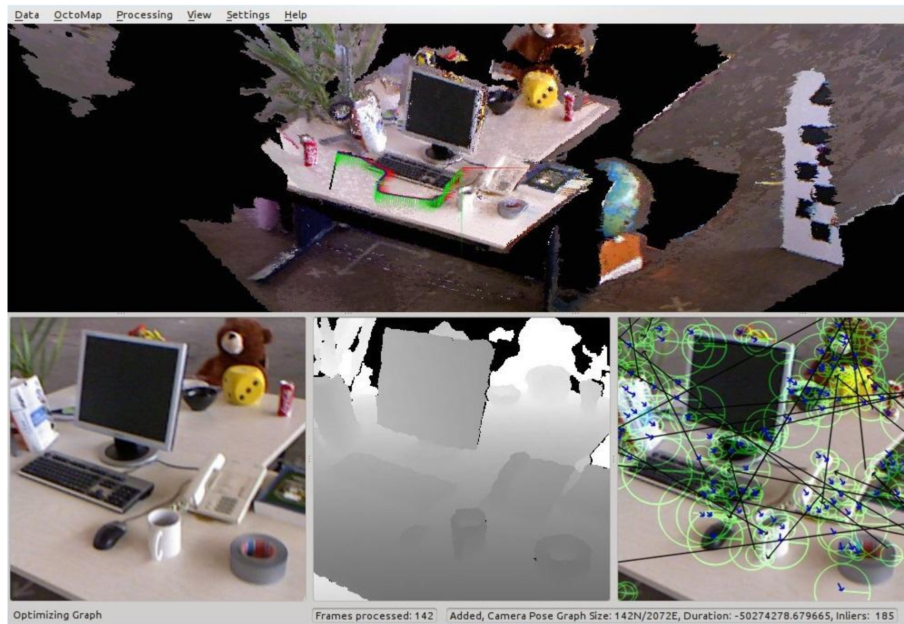

Figura 2.11. RGBDSlam GUI [10]

2.5. Rviz

Se trata de un paquete creado para funcionar sobre ROS (Robot Operating System) sin embargo, dado que se trata de una aplicación con GUI propia creo conveniente tratarlo como un software a parte, y no incluirlo con los otros paquetes mencionados en el apartado 2.2. de este capítulo.

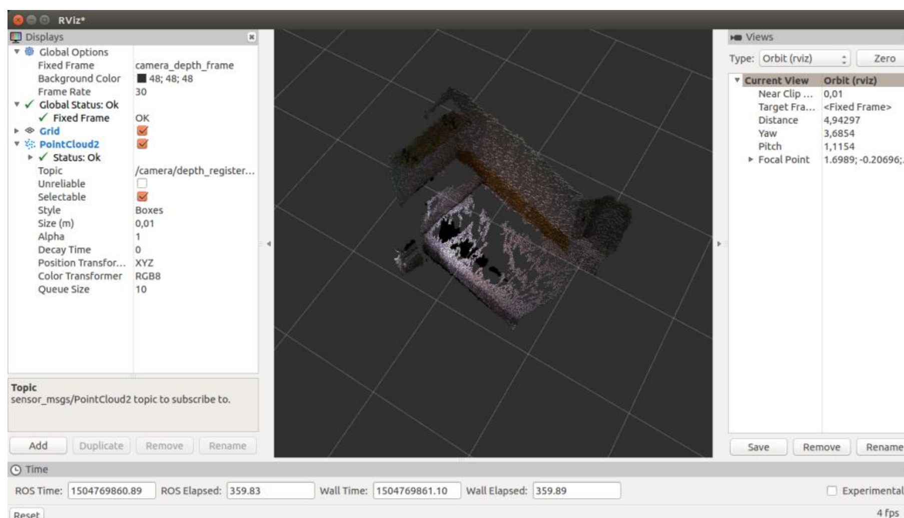

Figura 2.12. Rviz GUI.

Esta aplicación permite visualizar transformaciones espaciales, imágenes y PointClouds, entre otros mensajes de ROS. Se subscribe directamente al topic en el que se esta emitiendo la información a visualizar. Ha sido una herramienta utilizada principalmente para depurar ("Debug") los algoritmos.

En la figura 2.12. se ofrece una imagen de la GUI de Rviz, en este caso, visualizando una nube de puntos.

2.6. MATLAB

Se ha utilizado el entorno de cálculo y programación MATLAB para realizar análisis de los datos obtenidos por el IMU.

CAPÍTULO 3

Creación del mapa

El objetivo es ubicar el sistema cámara-IMU en un entorno conocido, un mapa. Para ello, el primer paso es crear dicho mapa. Esto plantea dos preguntas: ¿sobre qué entorno construir el mapa? y ¿cómo construir el mapa? A estas preguntas se les da respuesta en los dos apartados siguientes respectivamente.

1. ELECCIÓN DEL ENTORNO DE EXPERIMENTACIÓN

Como ya se ha indicado previamente en el capítulo 2, se dispone de una cámara RGB-D. Esto limita el alcance del proyecto a espacios cerrados en los que no afecte la radiación solar. Además el rango teórico de visión de la cámara, en cuanto a profundidad se refiere, se encuentra entre los 0.8 y los 3.5 metros. Esto reduce aún más el alcance, limitándose a espacios cerrados y de pequeño tamaño.

Por tanto, se podría elegir cualquier interior de un edificio, siempre y cuando tenga suficiente detalle. En otras palabras, evitando espacios diáfanos donde un pequeño campo de visión, como es el de la cámara, dé lugar a la captura de imágenes que puedan asociarse a muchas zonas diferentes del mapa. Esto se puede ilustrar con el siguiente ejemplo: cuando una cámara RGB-D se enfrenta a un movimiento paralelo a una pared plana, no es capaz de diferenciar en que zona está, y se trata de otro problema diferente que este proyecto no aspira a resolver. Por ello, se tratará de simplificar la escena, en este aspecto.

Otro entorno que a priori podría resultar adecuado sería el interior de cuevas o grutas, con tamaño dentro del rango de visión de la cámara. Incluso tratándose de ambientes con baja o nula iluminación, puesto que al ser un sensor activo, como se explica en el capítulo 2, no precisa de una fuente de iluminación externa.

Por facilidad y cercanía a la hora de testear la aplicación, se ha optado por realizar el mapa del interior de un edificio, más concretamente, de una vivienda.

2. CONSTRUCCIÓN DEL MAPA

Una vez resuelto el qué, toca responder al cómo. Existen diversas técnicas y tecnologías para construir un mapa 3D. A grandes rasgos, se pueden utilizar tecnologías basadas en el sonido o en la luz y se puede componer la escena extrayendo puntos característicos o conociendo de manera precisa la localización del sensor en cada imagen registrada. Sensores láser darían una mejor resolución al mapa, pero no se dispone del sistema necesario para llevar esto a cabo. Y casi por eliminación y por ser la única manera disponible de construir un mapa, se ha utilizado la propia cámara RGB-D.

Como ya se adelantaba en el capítulo 2, se ha utilizado el paquete RGBDSlam de ROS para construir el mapa a partir de la cámara RGB-D. Se trata de un programa algo impreciso y se obtienen unas nubes de puntos en ocasiones algo distorsionadas si no se manipula siguiendo unas determinadas pautas.

Este programa de mapeo se basa en la extracción de descriptores SIFT y trata después de buscar relaciones entre fotogramas para solaparlos. El algoritmo que utiliza responde relativamente bien a desplazamientos lineales de la cámara pero no ocurre lo mismo cuando se trazan trayectorias curvas tratando de capturar una misma escena desde todos sus ángulos. Esto, nuevamente, limita aún más la calidad del mapa.

En la figura 3.1. se pueden ver dos ejemplos de intento de mapeo de una escena 3D desde 360°. En la primera imagen, capturando un objeto haciendo una trayectoria circular a su alrededor y en la segunda imagen, capturando una habitación girando la cámara sobre si misma. Como se puede apreciar el resultado no es muy preciso. Habitualmente con forma aumenta el número de fotogramas capturados empieza a confundir zonas en la escena y a establecer conexiones erróneas. El cerrado de bucles es otro de sus puntos débiles.

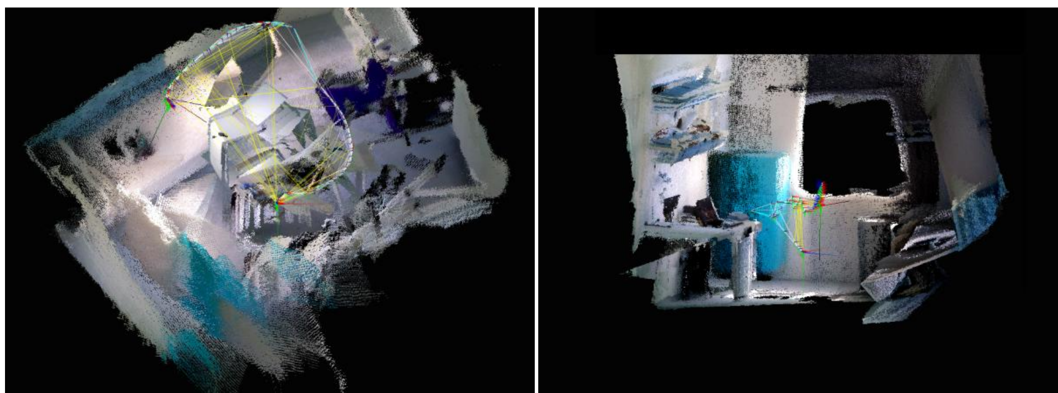

Figura 3.1. Ejemplos 3D insatisfactorios con RGBDSlam

RGBDSlam permite la captura de video para la generación de mapas con tan solo desplazar la cámara, pero la experiencia demuestra que se obtienen mejores resultados cuando se captura la escena a partir de fotogramas individuales en posiciones concretas lanzados de forma manual. Véase figura 3.2.

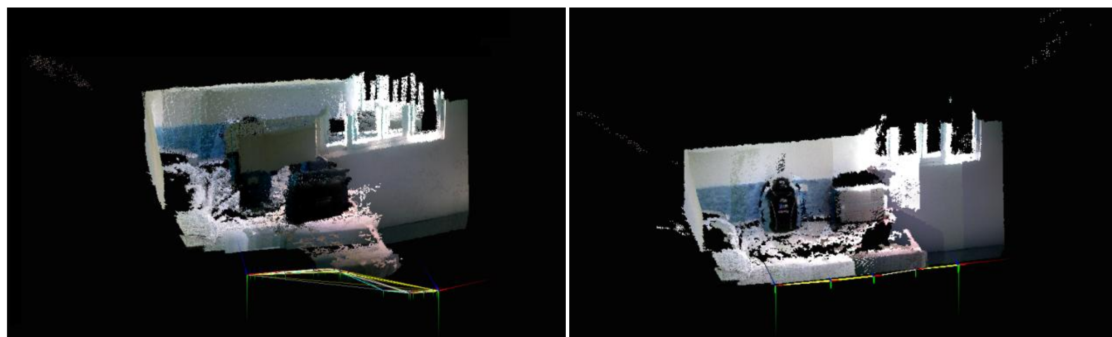

Figura 3.2. Comparación captura automática (izq) y captura manual (der)

En la figura 3.3. se puede ver el resultado final. Se trata de un conjunto de muebles con algunos elementos encima para obtener mas volumen. En amarillo se puede ver la trayectoria seguida por la cámara y los puntos rojos muestran las posiciones en las que se tomaron los distintos fotogramas que componen la nube de puntos. Como se puede apreciar se trata de trayectorias rectilíneas intentando obtener puntos de vista diferentes pero sin variar la orientación de la cámara. Nótese que la trayectoria se ha dibujado sobre la imagen a modo de esquema y no representa los puntos exactos. No se puede ver el color real porque solo se ha guardado la información de profundidad, que es la necesaria para la aplicación.

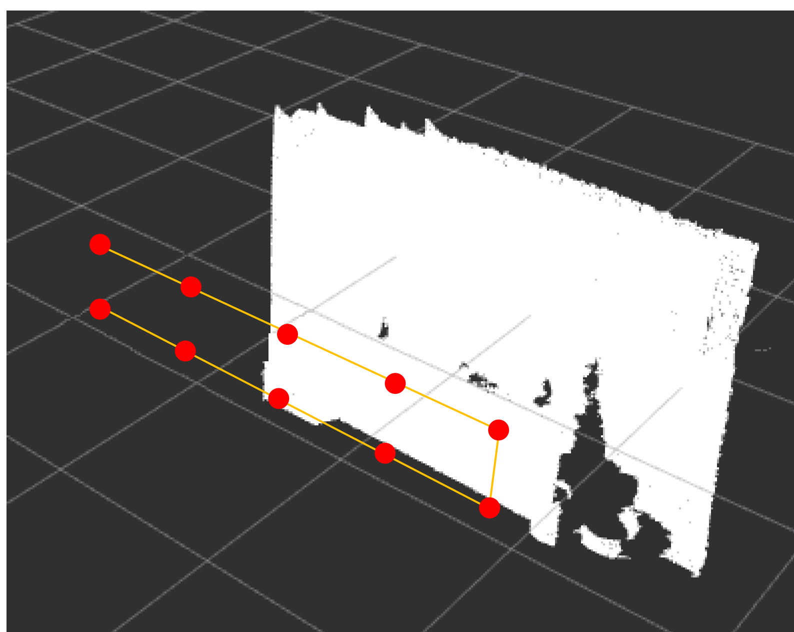

Figura 3.3. Mapa creado con RGBDSlam utilizado para probar la aplicación

CAPÍTULO 4

Generación de la secuencia de movimiento

Una vez se tiene el mapa se procede a la captura de la secuencia. El análisis de los datos provenientes de los sensores se va a realizar en post-procesado, es decir, se independiza la captura de datos, del análisis de los mismos. Por tanto, este capítulo, se centra únicamente en como capturar la información y en como guardarla para que sea accesible en el futuro.

1. ESTRUCTURA DE DATOS

Se entiende por secuencia, en el contexto de este proyecto, una sucesión de nubes de puntos y mediciones de IMU, localizadas en el tiempo.

Por tanto se debe capturar, no solo la información del sensor, sino también la estampa temporal que permita situar esa información en el tiempo. No interesa una escala de tiempo absoluta, con fecha y hora reales, basta con una escala de tiempo relativa al inicio de la secuencia. Pues el único propósito del tiempo es poder ordenar los eventos y conocer su periodo (medida diferencial).

En la figura 4.1. se puede ver una posible secuencia a modo de ejemplo.

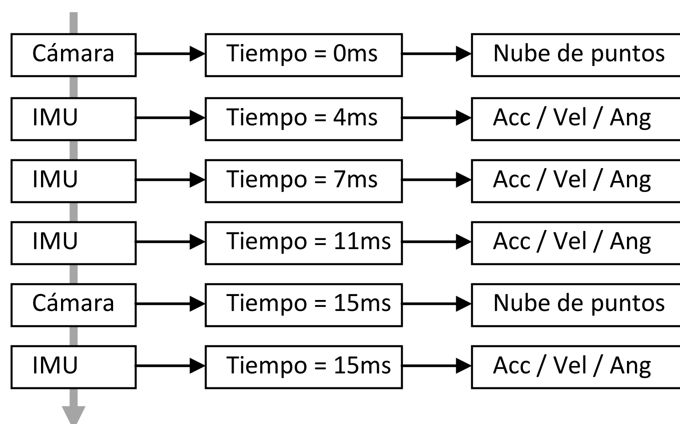

Figura 4.1. Ejemplo de secuencia

Esta secuencia se almacena en ficheros. PCL tiene su propio formato de fichero de nube de puntos (.pcd). Para todo lo demás, se utilizan archivos de texto (.txt). En la figura 4.2. se puede ver el esquema de archivos.

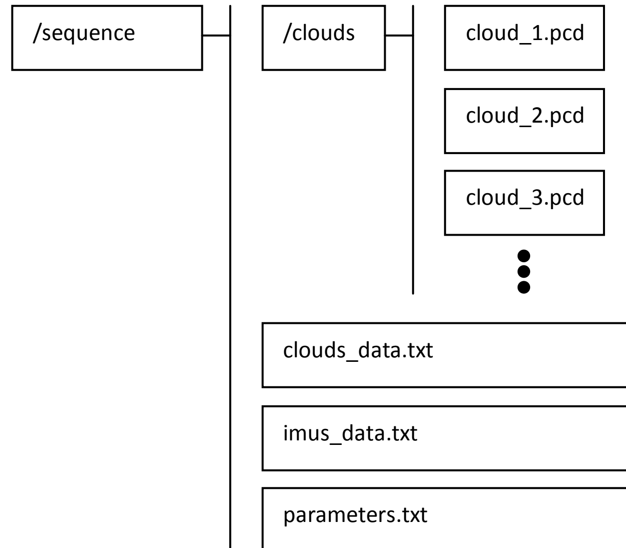

Figura 4.2. Estructura de datos de la secuencia

/clouds

Partiendo del directorio raíz (/sequence) se tiene un segundo directorio (/couds) en que se encuentran cada uno de los fotogramas tomados por la cámara en formato .pcd.

clouds_data.txt

Contiene una lista de las direcciones de las nubes de puntos contenidas en /clouds y el instante temporal relativo en milisegundos. Véase figura 4.3.

```
cloud_1.pcd 386

cloud_2.pcd 940

cloud_3.pcd 1440

cloud_4.pcd 2007

⋮
```

Figura 4.3. Ejemplo clouds_data.txt

imus_data.txt

Contiene una lista de los registros del IMU y el instante temporal relativo en milisegundos. Véase figura 4.3. en cada fila se tiene en el siguiente orden:

1. Id
2. Instante temporal
3. Orientación en x (Cuaternio)
4. Orientación en y (Cuaternio)
5. Orientación en z (Cuaternio)
6. Orientación en w (Cuaternio)
7. Velocidad angular en x
8. Velocidad angular en y
9. Velocidad angular en z
10. Aceleración lineal en x
11. Aceleración lineal en y
12. Aceleración lineal en z

```
imu_1 0 -0.00707902 0.00783342 -0.948014 0.318055 -0.00492148 0.00492148 0 0.0478728 -0.180406 9.71857

imu_2 4 -0.00668146 0.00792755 -0.948017 0.318052 -0.00386695 0.00456997 -0.00351352 0.0373761 -0.182662 9.7275

imu_3 8 -0.00662037 0.00805606 -0.948016 0.31805 -0.00281225 0.00386695 -0.00257663 0.0171675 -0.169909 9.73348

imu_4 12 -0.00620709 0.00805725 -0.948018 0.318054 0.00210923 0.00632769 0 0.0150093 -0.169909 9.7223
⋮
```

Figura 4.4. Ejemplo imus_data.txt

parameters.txt

Contiene el número total de entradas de la secuencia, separando entradas de la cámara y entradas del IMU. Véase figura 4.5.

```
Cld_instances: 15

Imu_instances: 2336
```

Figura 4.5. Ejemplo parameters.txt

Además, siempre se descarta la ultima entrada tanto del IMU como de la cámara para evitar que la interrupción del programa pueda dar lugar a una entrada incompleta.

2. DESCRIPCIÓN DEL SOFTWARE

El programa que realiza la tarea de crear la secuencia es “tfg_recorder”. Está implementado como un nodo ROS. Esto implica que necesita de la ejecución de “roscore”, el master, para poder ser ejecutado.

El programa comienza con la inicialización del nodo ROS, registrándose como “tfg_recorder”, y subscribiéndose a los topics:

1. “/camera/depth/points”, que recibe las nubes de puntos.
2. “/imu/data”, que recibe la información del IMU.

Se ha modificado el comportamiento de interrupción del programa para poder cerrar ficheros y asegurar la integridad de los datos recogidos hasta el momento.

Finalmente, al tratarse de un nodo ROS, es necesario llamar al “spinner”, que gestiona las diferentes interrupciones que debe procesar el nodo.

En la figura 4.6. se puede ver el diagrama de flujo general del programa. Las subrutinas de interrupción se detallan en los apartados siguientes.

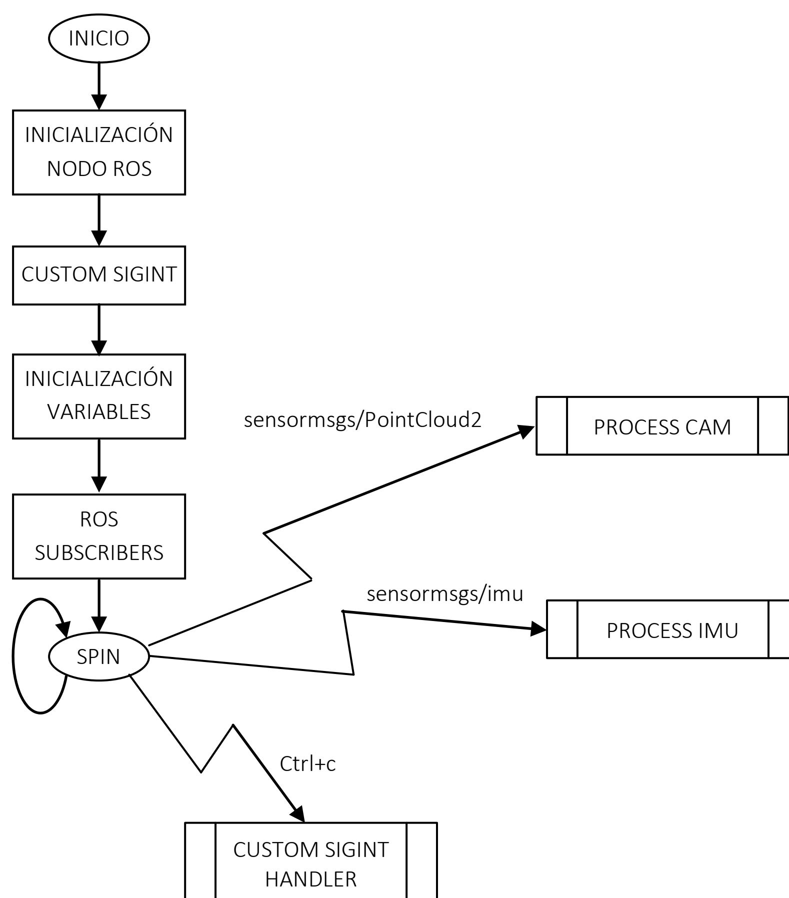

Figura 4.6. Diagrama de flujo de tfg_recorder

3. LECTURA DEL IMU

El IMU, tras ser interpretado por el nodo “phidgets_imu” y preprocesado por el nodo “imu_filter_madgwick”, publica en el topic “/imu/data” la información de orientación, velocidad angular y aceleración lineal utilizando el tipo de dato de ROS “sensor_msgs::Imu”. En la figura 4.7. se pueden ver los campos que se utilizan de este tipo de dato. La definición completa de “sensor_msgs/Imu” se puede encontrar en el anexo 1.

std_msgs/Header	header
geometry_msgs/Quaternion	orientation
geometry_msgs/Vector3	angular_velocity
geometry_msgs/Vector3	linear_acceleration

Figura 4.7. sensor_msgs/Imu [11]

En la figura 4.8. se muestra el flujograma correspondiente a la interrupción que se ejecuta cada vez que se publica un nuevo mensaje en el topic “/imu/data”. El acceso a las variables comunes a ambas interrupciones está protegido por mutex para evitar accesos simultáneos. Para conseguir procesar a tiempo real todos los mensajes de entrada es necesario forzar al nodo a ejecutarse en varios núcleos. De cómo hacerlo se encarga ROS y es invisible desde este nivel. Por seguridad se protegen dichas variables.

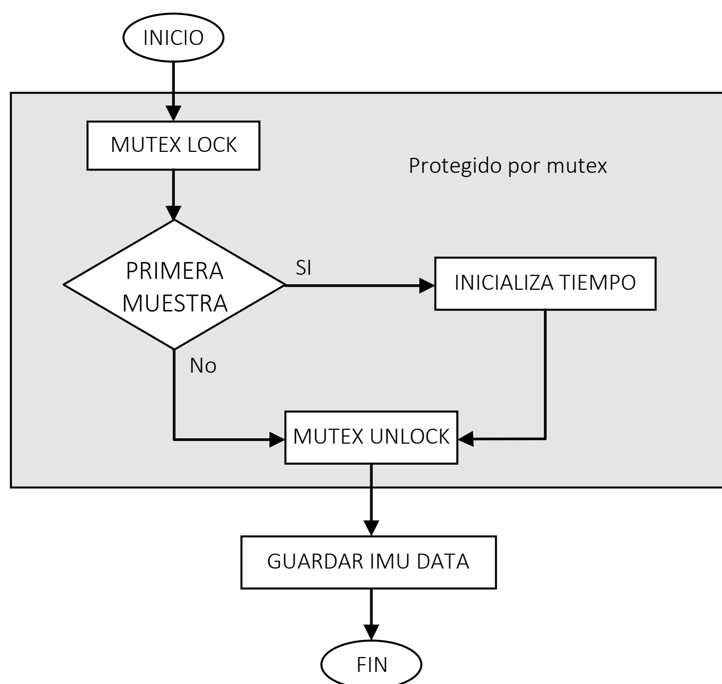

Figura 4.8. Diagrama de flujo de procesamiento del IMU

4. Lectura de la cámara RGB-D

La cámara RGB-D se comunica por USB con el computador. El nodo "openi2_camera" contiene los drivers para recoger las imágenes y publica en el topic "/camera/depth/points" la nube de puntos correspondiente a la escena que está enfocando. El formato que utiliza es el tipo de ROS "sensor_msgs/PointCloud2". (Anexo 2).

En la figura 4.9. se muestra el flujograma correspondiente a la interrupción que se ejecuta cada vez que se publica un nuevo mensaje en el topic "/camera/depth/points". Como ya se explica en el apartado anterior, se utiliza nuevamente el mutex para evitar el acceso simultaneo a las variables compartidas.

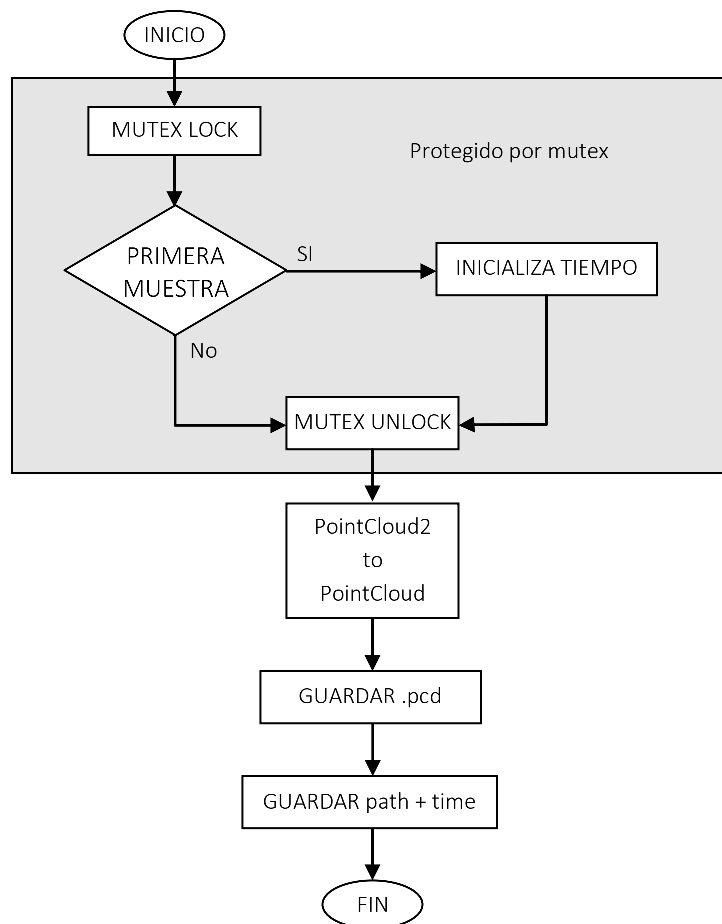

Figura 4.9. Diagrama de flujo de procesamiento de la cámara

La nube de puntos se guarda en formato .pcd, que es el formato propio de la PCL (Point Cloud Library), con la que luego se manipulará la nube de puntos.

CAPÍTULO 5

Procesamiento de la secuencia de movimiento

En este capítulo se explica el funcionamiento del programa “tfg_processor”. Este programa es el encargado de interpretar la secuencia previamente grabada y procesar la información que contiene para calcular la trayectoria que ha seguido el sistema.

1. DATOS DE ENTRADA

El programa “tfg_processor” toma como datos de entrada la secuencia previamente creada con “tfg_recorder”, el mapa previamente creado con “RGBDSlam”, y unos datos de calibración del IMU, también obtenibles con “tfg_recorder”. Por tanto, para que el programa pueda interpretar la información, se le ha de presentar siguiendo la estructura que se muestra en la figura 5.3.

El programa “tfg_processor” recibe como argumento una dirección al directorio “/map” que puede tomar el nombre que se quiera. (Véase Capítulo 5.5. Ejecución). A partir de ahí, la nomenclatura se ha de seguir estrictamente.

“map.pcd” contiene el mapa creado con RGBDSlam en el formato estándar de la PCL (.pcd).

“/map” contiene tantas secuencias como se quiera con el nombre “sequence_<n>”, siendo <n> el número de la secuencia. El programa puede recibir como argumento que secuencia procesar. (Véase Capítulo 5.5. Ejecución).

Cada secuencia “sequence<n>” se corresponde con la secuencia obtenida con el programa “tfg_recorder” a la que se deben añadir algunos fichero más. Estos ficheros son los siguientes.

“origin.txt” contiene la matriz de transformación que se corresponde con la localización inicial del sistema respecto de la referencia del mapa. Ejemplo en la figura 5.1.

```
1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0
```

Figura 5.1. Ejemplo “origin.txt”

“imu_calibration.txt” contiene una serie de capturas de IMU similar a la de la figura 4.4. Se utiliza para calibrar las medidas obtenidas con el IMU y eliminar el efecto de la gravedad. Se puede obtener grabando una secuencia con el sistema en reposo. Basta extraer y renombrar el archivo “imus_data.txt” obtenido en esa secuencia.

“imu_calibration_params.txt” contiene el número de entradas del archivo “imu_calibration.txt”. De la misma manera que el anterior, este se corresponde con el archivo “parameters.txt” del que se elimina la primera línea. Véase la figura 5.2.

```
Cld_instances: 15

Imu_instances: 2336
```

Figura 5.2. Ejemplo “imu_calibration_params.txt”

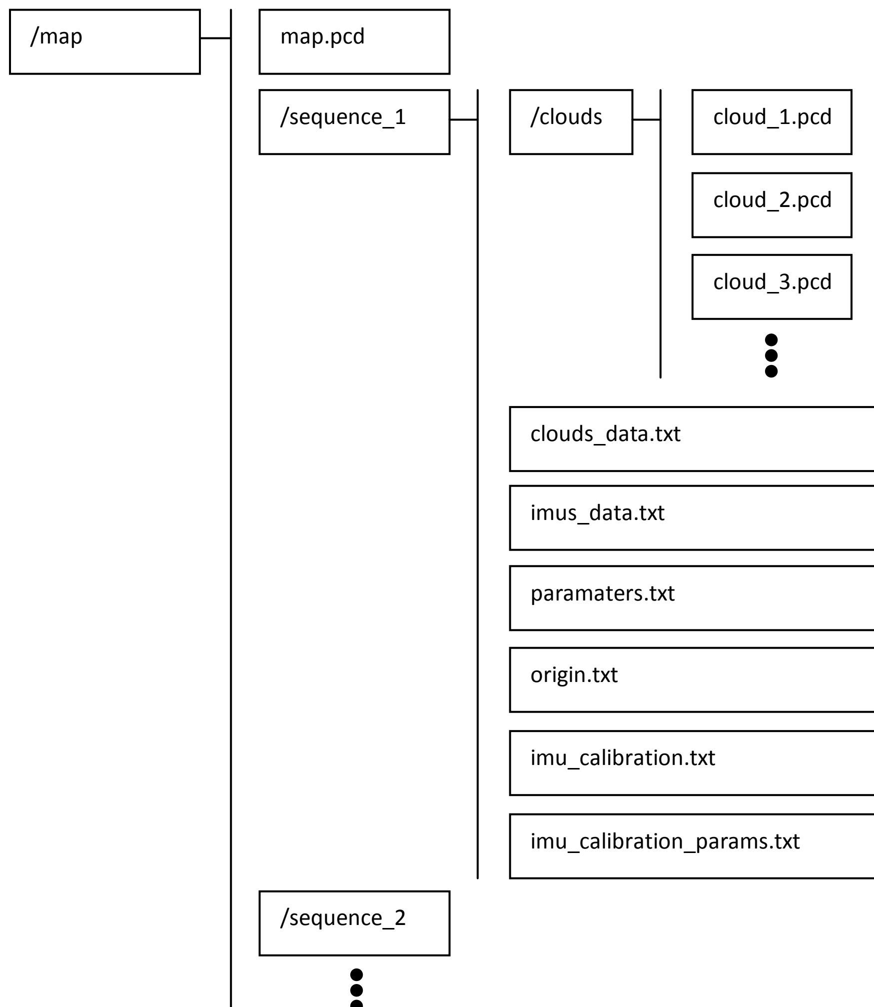

Figura 5.3. Estructura de datos de entrada de “tfg_processor”

2. DESCRIPCIÓN DEL SOFTWARE

El programa que realiza la tarea de procesar la secuencia es “tfg_processor”. Está implementado como un nodo ROS. Esto implica que necesita de la ejecución de “roscore”, el master, para poder ser ejecutado.

El programa comienza con la inicialización del nodo ROS, registrándose como “tfg_processor”. No necesita subscribirse ningún topic y por tanto no es necesario llamar al “spinner”

Se ha modificado el comportamiento de interrupción del programa para poder cerrar ficheros y asegurar la integridad de los ficheros.

En la figura 5.4. se puede ver el diagrama de flujo general del programa. Las subrutinas se detallan en los apartados siguientes.

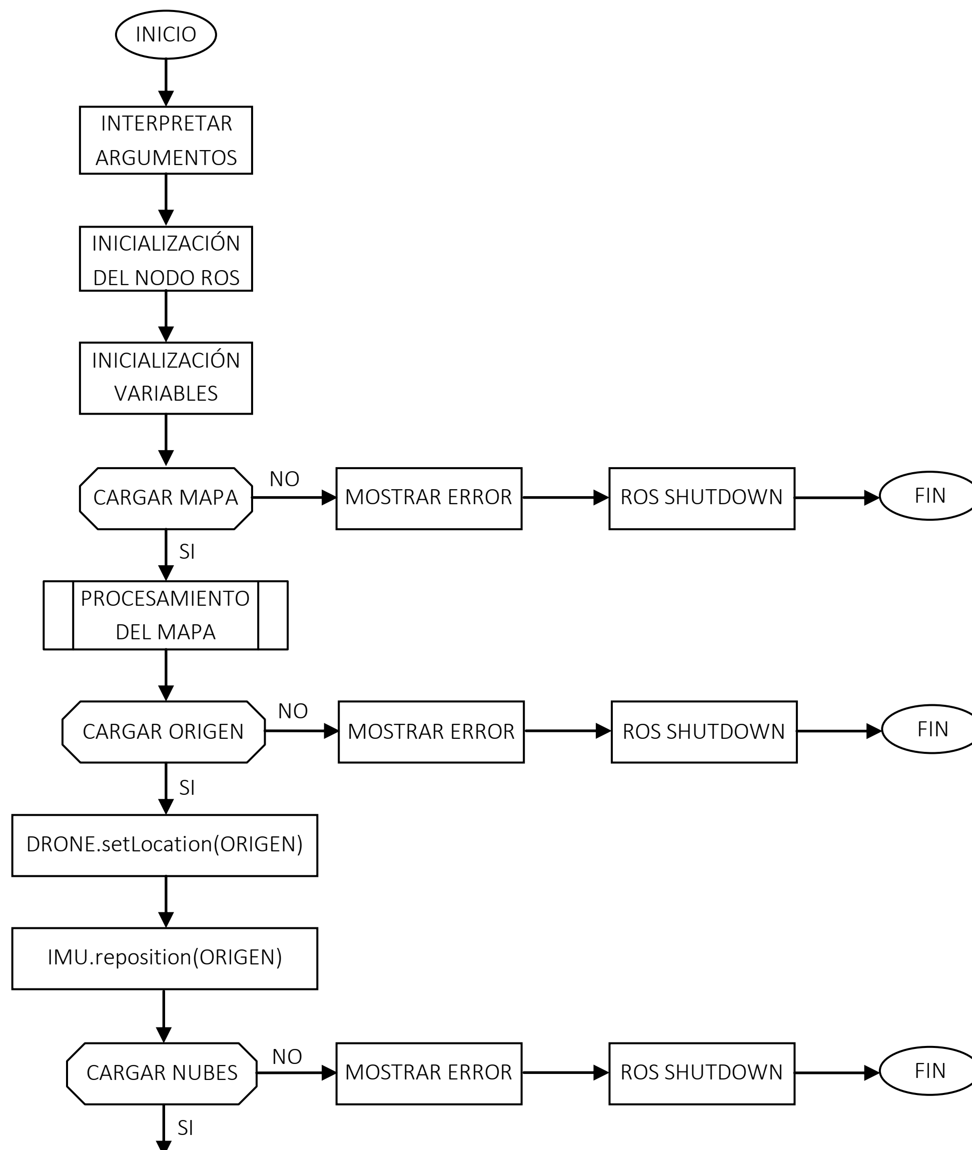

Figura 5.4. Diagrama de flujo de tfg_processor (a)

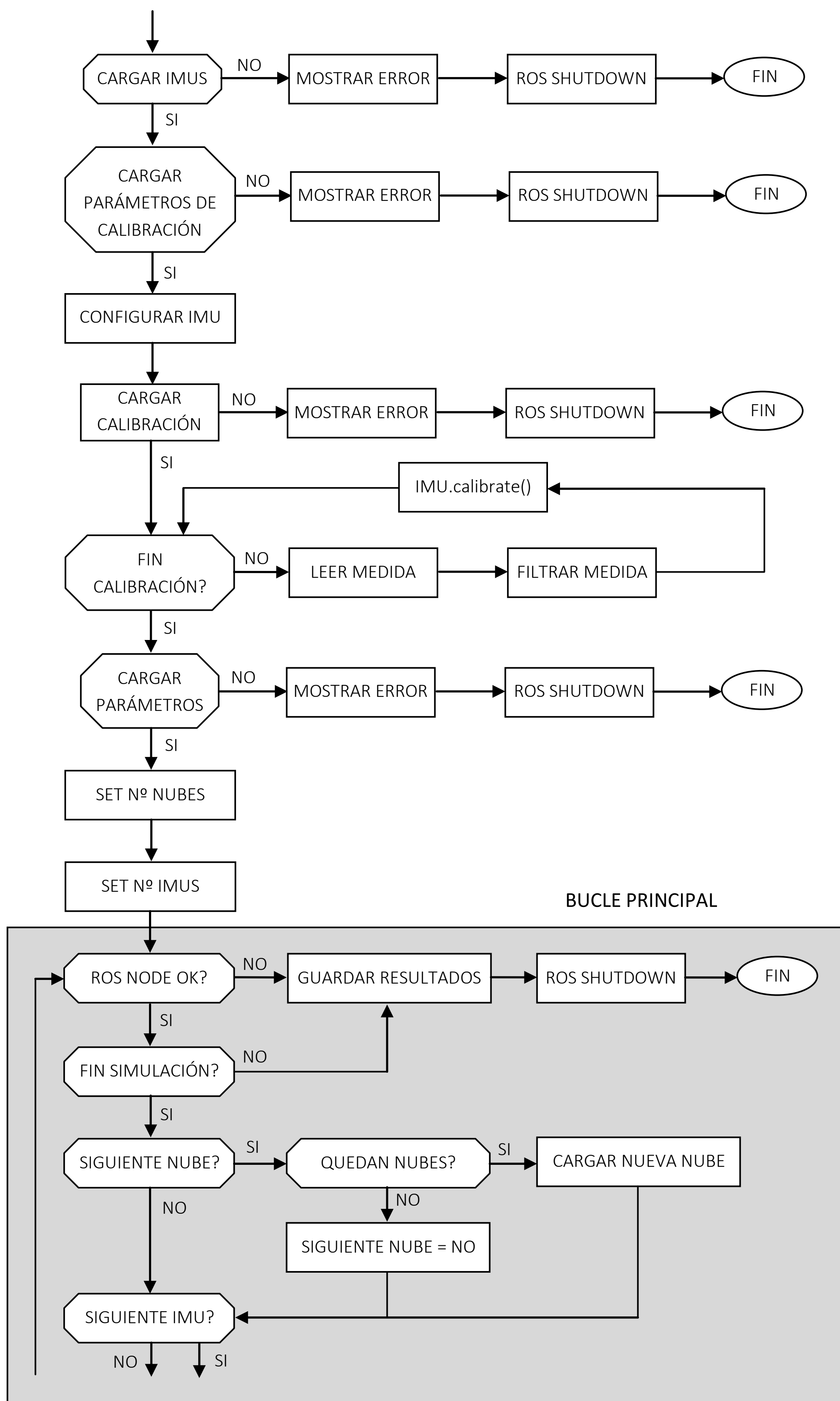

Figura 5.4. Diagrama de flujo de tfg_processor (b)

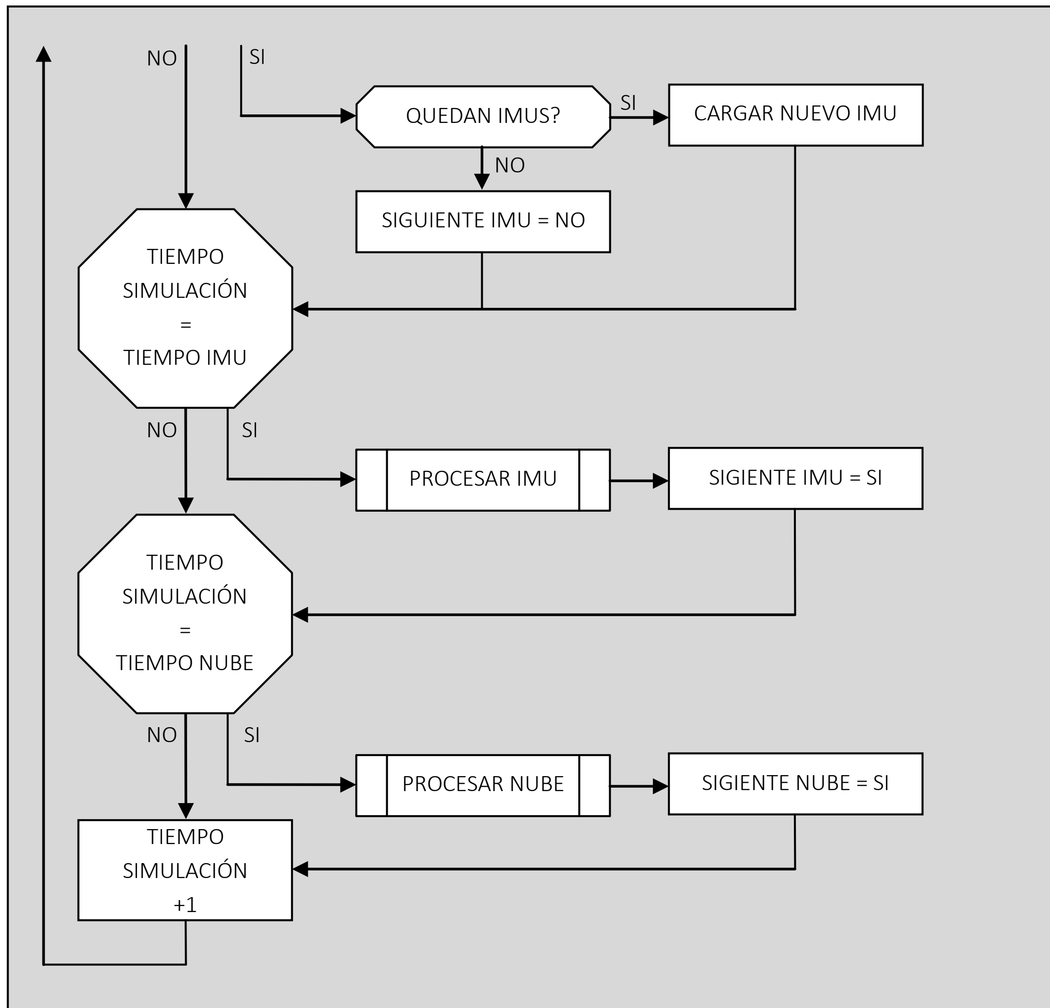

Figura 5.4. Diagrama de flujo de tfg_processor (c)

El programa trata de emular una aplicación de tiempo real y trabaja como un simulador. Comienza con la carga de los ficheros. En el momento en que se detecta alguna anomalía en alguno de los ficheros se cancela la ejecución y se informa al usuario.

Una vez cargados todos los datos e inicializado el sistema, comienza el procesado de la secuencia. Se tiene un reloj interno de simulación con precisión de un milisegundo. Este va incrementándose cada iteración y cuando coincide con la estampa temporal de una de las entradas, ya sea del IMU o de la cámara, esta es procesada. Se consigue de esta manera procesar los mensajes procedentes de los sensores en el mismo orden en que fueron capturados. El procesado de los mensajes del IMU y la cámara se detallan en los apartados 4 y 5 de este capítulo, respectivamente.

La simulación dura hasta que se agotan las muestras. En ese momento, se muestran y se guardan los resultados.

3. PROCESAMIENTO DEL MAPA

El mapa consiste en definitiva en una nube de puntos de la escena. Como se trata de una nube de puntos formada por la composición de muchas otras nubes de putos (creada con RGBDSlam) contiene una gran cantidad de puntos. Más correctamente se diría que tiene una alta densidad de puntos.

A mayor es el número de puntos, mayor es el tiempo de cómputo, en general, para cualquier algoritmo que se aplique sobre la nube de puntos. Además, al tratarse de una composición de nubes de puntos aparece un efecto de "ghosting" debido a la superposición no perfecta de las nubes. Figura 5.5.

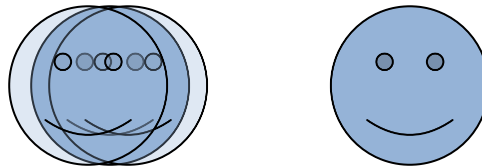

Figura 5.5. Efecto de "ghosting"

Para eliminar este y otros efectos indeseables se aplican una serie de filtros que se detallan a continuación.

3.1. Voxel Grid Filter

De acuerdo con la documentación ofrecida en la PCL [12], este filtro divide el espacio en cubos 3D de un determinado tamaño. Este tamaño se corresponde con la resolución de la nube de putos que se desea obtener. A continuación calcula el centroide de todos los puntos contenidos en cada cubo, esto es, la media de las coordenadas de estos puntos. El resultado no es exactamente una nube con los puntos separados una distancia fija (posiciones de los puntos discretas), sino una nube discretizada en cubos en los que se encuentra un único punto por cubo.

El efecto que produce es una reducción de la resolución y un filtrado paso bajo de la nube de puntos. Se elimina algo de ruido.

En la figura 5.6. se muestra un ejemplo extraído de la documentación de la librería de la PCL.

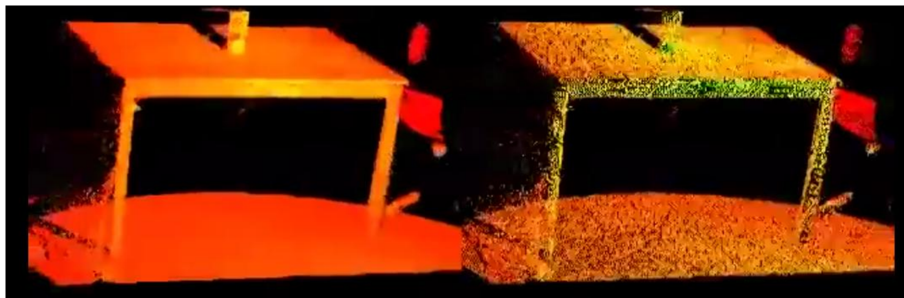

Figura 5.6. Ejemplo Voxel Grid Filter [12]

3.2. Eliminación de "outliers"

Los "outliers" son puntos que aparecen en la nube pero que no se corresponden con nada existente en la escena real. Su origen puede ser debido a una mala captura del sensor o a la existencia de bordes o perfiles en la escena con un gradiente de profundidad muy grande que confunden al sensor. Se aprecian formando como una estela en los bordes que puede ser confundida con una superficie en la propia dirección de proyección de la cámara.

En la figura 5.7. se puede ver, a la derecha, los "outliers" y, a la izquierda, la nube limpia de "outliers".

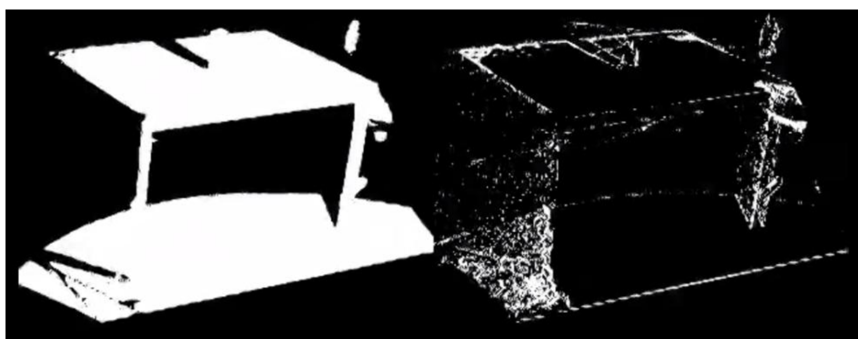

Figura 5.7. Ejemplo Outliers en una nube de puntos [13]

Para eliminar estos "outliers" se aplican dos técnicas distintas: "Statistical Outlier Removal Filter" y "Radius Outlier Removal Filter".

3.2.1. Statistical Outlier Removal Filter

De acuerdo con la documentación ofrecida en la PCL [13], este filtro utiliza una estadística del entorno de cada punto para determinar si se trata de un outlier.

Este algoritmo de filtrado itera sobre la nube de puntos dos veces. En la primera iteración calcula la media de las distancias de cada punto a sus k-vecinos. El valor de k es configurable y determina el tamaño del filtro. A continuación, calcula la media y la desviación estándar de todas estas distancias para determinar un umbral. El umbral se define como:

$$umbral = media + constante * desviación \qquad \text{[ec. 5.1]}$$

El valor de la constante también es configurable.

En la segunda iteración clasifica los puntos como inliers o outliers en función de si la media de las distancias de ese punto a sus k-vecinos está por debajo o por encima del umbral, respectivamente.

Con este filtro se consigue eliminar de manera efectiva el ruido producido por los bordes en la escena. No obstante, se debe configurar con cuidado porque puede eliminar superficies con poca densidad de puntos que interesaría conservar. Por ello, tras aplicar este filtro, pueden seguir quedando puntos sueltos. Para eliminarlos se aplica el siguiente filtro: "Radius Outlier Removal Filter".

3.2.2. Radius Outlier Removal Filter

De acuerdo con la documentación ofrecida en la PCL [14], este filtro elimina puntos de una nube en función del número de vecinos (puntos a su alrededor).

Este algoritmo de filtrado itera una única vez por todos los puntos de la nube. Para cada punto determina el número de vecinos dentro de un radio dado. Si tiene menos vecinos de los configurados como umbral, el punto es considerado como outlier.

Configurando un número de vecinos bajo, se consigue eliminar los puntos que han quedado sueltos tras aplicar el filtro: "Statistical Outlier Removal Filter".

4. Estimación de la posición

En este apartado se explica el procesamiento de la información proporcionada por el IMU.

Como ya se ha comentado antes, esta información que proporciona el IMU se utiliza para obtener una estimación de la localización del sistema, entendiendo por localización, posición más rotación.

En la figura 5.8. se puede ver el diagrama de flujo de procesamiento del IMU. Para mantener una explicación ordenada se irá comentando una a una las distintas etapas del diagrama.

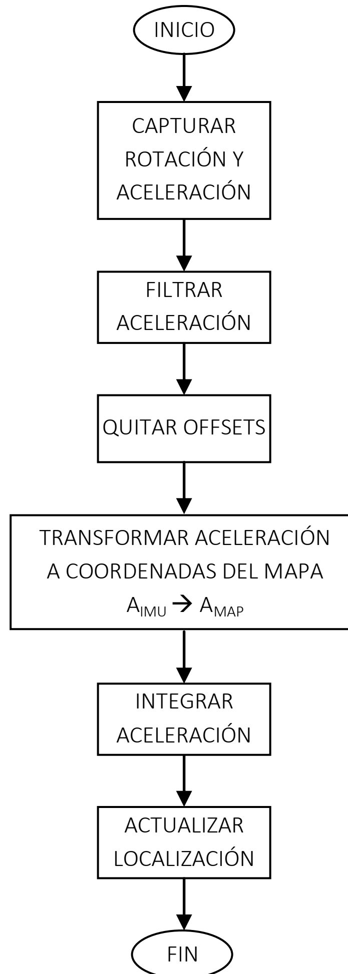

Figura 5.8. Diagrama de flujo de procesamiento del IMU

Capturar rotación y aceleración

En primer lugar se captura la información del IMU. En una supuesta aplicación de tiempo real se cogería directamente del topic de ROS correspondiente. En este caso se lee de un mensaje reconstruido a partir de los datos guardados en memoria. A efectos de tiempo de cómputo no se tiene en cuenta el tiempo empleado en la carga del fichero, dado que en una supuesta aplicación de tiempo real, este paso no sería necesario.

De la lectura del IMU se obtiene

1. un vector de dimensión tres, correspondiente a la aceleración lineal en los ejes ‘x’, ‘y’, y ‘z’ de una referencia solidaria al propio IMU. De ahora en adelante se le hará referencia como A_{IMU}. Véase la figura 5.9.

Figura 5.9. Sistema de referencia solidario al IMU

2. y un cuaternión, o vector de dimensión 4, correspondiente al ángulo de rotación del IMU respecto de un sistema de referencia interno y fijo del IMU. Este sistema de referencia fijo, al que de ahora en adelante se le hará alusión con el subíndice “FIX”, permanece constante en el tiempo y entre ejecuciones dado que el IMU lo localiza internamente a partir del norte polar (obtenido del magnetómetro que contiene) y la fuerza de gravedad.

   Por comodidad se utilizan matrices de rotación en vez de cuaterniones para representar las rotaciones. Estas matrices de rotación se van a representar de ahora en adelante usando la siguiente nomenclatura: ${}^{OGN}R_{FIN}$. Donde ‘R’ indica matriz de rotación, ‘OGN’ el sistema de referencia de origen y ‘FIN’ el sistema de referencia al que lleva la transformación.

   De esta manera, se puede definir ya la matriz de rotación que indica la rotación del IMU respecto de su sistema de referencia fijo como: ${}^{FIX}R_{IMU}$.

Filtrar aceleración

Como prácticamente todo sensor, el IMU entrega una medida con ruido. Este ruido, puede pasar inadvertido cuando se miden magnitudes grandes, sin embargo, cuando se miden magnitudes pequeñas el ruido cobra una gran importancia.

En este caso, se están midiendo aceleraciones con picos en torno a los 3 m/s^2. Teniendo en cuenta que el sensor tiene unos límites de ±19.6 m/s^2 en su rango de mayor precisión (como se explica en el apartado 1.2. del capítulo 2), significa que estamos ante una magnitud pequeña. En la gráfica 5.10. se puede ver un ejemplo de las aceleraciones en el eje ‘y’ del IMU ante un desplazamiento rectilíneo en la dirección negativa de ese eje. Se pueden distinguir las etapas de aceleración y frenada (picos negativo y

positivo respectivamente) pero se aprecia como el ruido tiene una magnitud similar a la de la señal en prácticamente todo el recorrido.

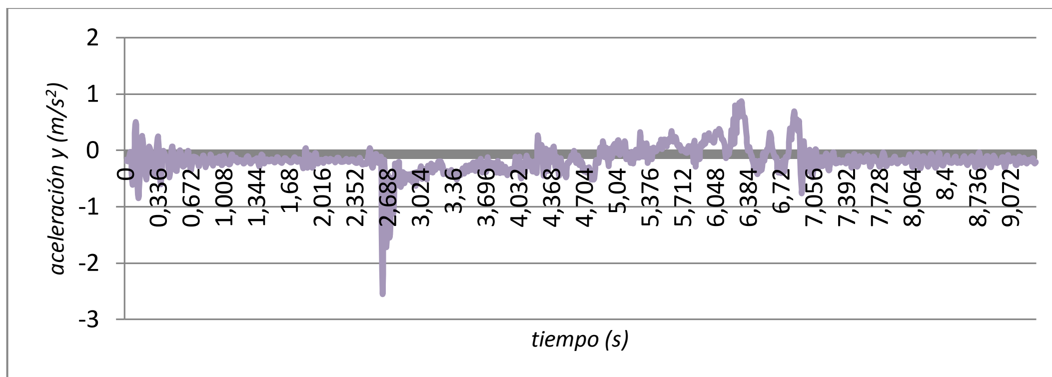

Figura 5.10. Ejemplo de medida de aceleración del IMU

Para conocer ante que tipo de ruido se encuentra el sistema se ha llevado a cabo un análisis frecuencial de las mediciones del IMU. El ensayo se ha realizado utilizado 100.000 muestras del IMU en parado. En la figura 5.11. se muestra el resultado del análisis frecuencial.

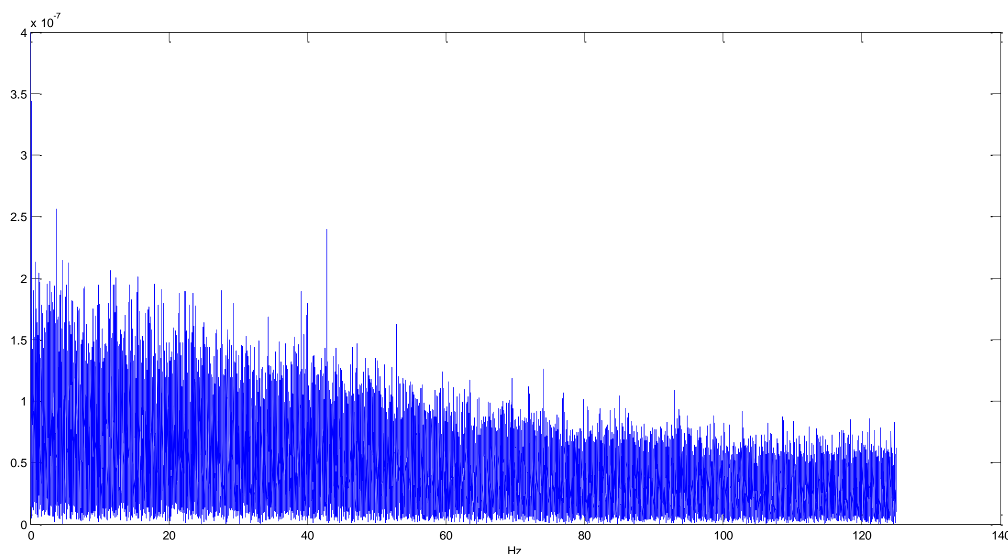

Figura 5.11. Análisis frecuencial de las aceleraciones del IMU

Como se puede ver, existe un pico en torno a cero que se corresponde con un offset en la medición del sensor. Además aparecen frecuencias en todo el rango de aproximadamente la misma magnitud, es decir, se trata de un ruido blanco.

Para reducir este ruido se aplica un filtro de media móvil. Este filtro entrega como salida la media de las últimas 'n' mediciones. En la figura 5.12. se puede ver una comparación de los espectros de frecuencia antes y después de aplicar el filtro. Para el ejemplo de la gráfica se ha utilizado una media móvil de veinte medidas.

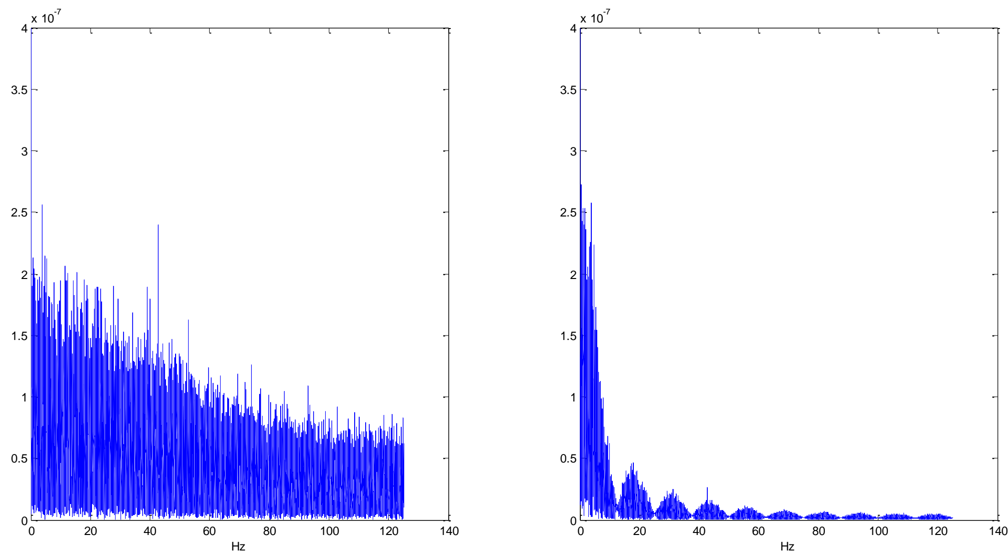

Figura 5.12. Filtro de media aplicado al IMU. Espectro frecuencial

Se debe tener en cuenta que al aplicar este filtro se consigue reducir el ruido, pero se introduce un retraso temporal en la medida. Este retraso se hace más acusado cuanto mayor es el numero de muestras que utiliza el filtro.

En la figura 5.13. se puede ver el resultado de aplicar el filtro de media móvil de 20 muestras sobre la señal. Se ha reducido el ruido. Este es el filtro que se usa en la aplicación.

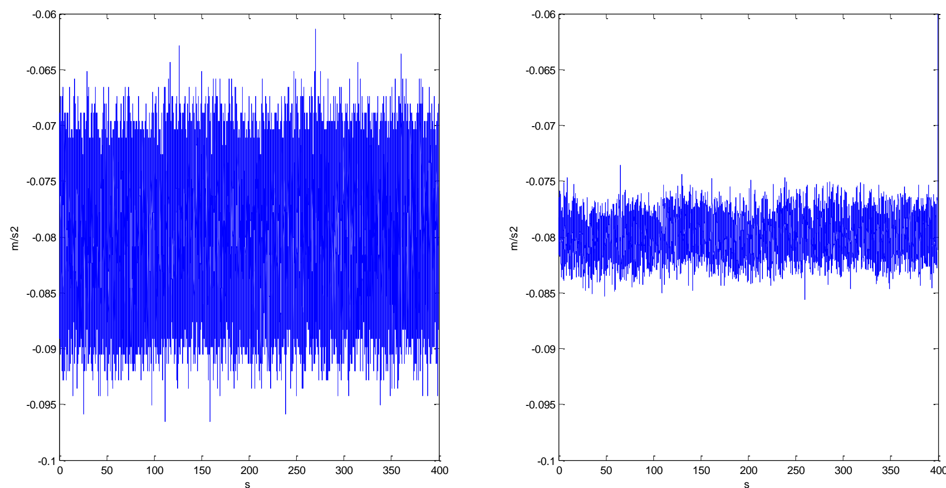

Figura 5.13. Filtro de media aplicado al IMU. Secuencia temporal

Quitar *offsets*

Como todos los sensores, el IMU necesita de una calibración previa antes de usarse. La medida del IMU se ve afectada por dos offsets distintos de cara al resultado que se espera obtener del IMU. Por un lado, está el offset inherente al sensor, debido a un descentramiento del valor nulo y que es diferente e independiente para cada eje. Y por otro lado, está el offset debido al efecto de la gravedad, que está presente siempre en la misma dirección en el sistema de referencia de la Tierra pero que afecta a los 3 ejes del IMU en función de su orientación.

Para conocer el primer offset del sensor se debe realizar el siguiente experimento. En realidad se trata de tres offsets distintos, uno para cada eje. Para cada uno de los ejes y con el IMU en posición estática, se sitúa el eje positivo en dirección vertical de tal forma que toda la fuerza de la gravedad sea sensada en ese eje. Se toma una serie de medidas y se calcula la media. Se realiza el mismo experimento hasta un total de seis veces, para cada uno de los ejes y en sus dos sentidos. Al final se tiene una tabla como la siguiente (Tabla 5.1.).

	Sentido Positivo	Sentido Negativo
Eje x	xp	xn
Eje y	yp	yn
Eje z	zp	zn

Tabla 5.1. Cálculo de los offsets del IMU

Los offsets se calculan de la siguiente forma.

$$offset_x = xp + xn \qquad \text{[ec. 5.2]}$$

$$offset_y = yp + yn \qquad \text{[ec. 5.3]}$$

$$offset_z = zp + zn \qquad \text{[ec. 5.4]}$$

Una vez se tienen calculados los offsets, es decir, se ha calibrado el IMU, la eliminación de los offsets en tiempo de ejecución es tan simple como restar estos valores a los obtenidos por el IMU directamente en el sistema de referencia del IMU. En el Anexo 2 se presenta el cálculo de los offsets para el IMU utilizado en la aplicación.

Por otro lado, está el efecto de la gravedad. En este caso se trata de un offset que no pertenece a ninguno de los ejes del IMU sino que se presenta como una aceleración vertical y hacia abajo, y que aparecerá de diferente manera en el sistema de referencia del imu en función de su orientación. Sin embargo, como ya se comentaba antes, se dispone de una matriz de rotación ${}^{FIX}R_{IMU}$ (Figura 1.14) que relaciona el sistema de coordenadas del IMU con un sistema de coordenadas fijo que tiene la propiedad de tener el eje z, vertical respecto de la Tierra. De esta forma se sabe que la gravedad aparecerá siempre en el eje z de este sistema de referencia fijo (FIX).

Para poder eliminar el efecto de la gravedad, se transforma el vector aceleración A_{IMU} al sistema de coordenadas FIX, obteniéndose A_{FIX}.

$$A_{FIX} = {}^{FIX}R_{IMU} * A_{IMU} \qquad \text{[ec. 5.5]}$$

Una vez se tiene la aceleración en la referencia FIX, eliminar el efecto de la gravedad es tan sencillo como sustraer su valor, 9.8 m/s^2.

$$A_{FIX} = A_{FIX} - 9.8 \qquad \text{[ec. 5.6]}$$

Finalmente se deshace el cambio de referencia para volver a ejes del IMU.

$$A_{IMU} = {}^{IMU}R_{FIX} * A_{FIX} = {}^{FIX}R_{IMU}{}^{-1} * A_{FIX} \quad \text{[ec. 5.7]}$$

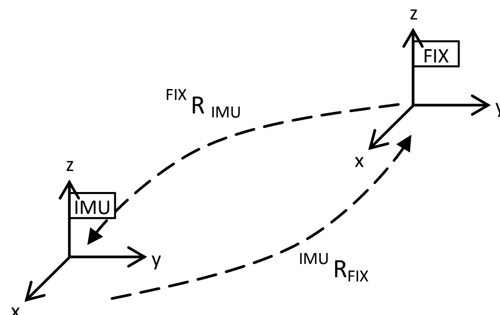

Figura 5.14. Sistema de referencia del IMU

Transformar aceleración a coordenadas del mapa

En este momento se tiene una aceleración libre de ruido y de offsets, es decir, se tienen las aceleraciones puras a las que está sometido el IMU debido a su movimiento. Pero estas aceleraciones están referenciadas respecto de un sistema de referencia móvil como es el IMU. Es necesario llevar esas aceleraciones a un sistema de referencia fijo. Como ya se ha comentado, se dispone del sistema de referencia FIX, pero como el objetivo final es localizar el sistema dentro de un mapa, se utilizará el sistema de referencia propio de ese mapa como sistema fijo. De ahora en adelante se hará alusión a este sistema de referencia como MAP.

En la figura 5.15. se muestran todas las transformaciones que se aplican en el algoritmo de localización.

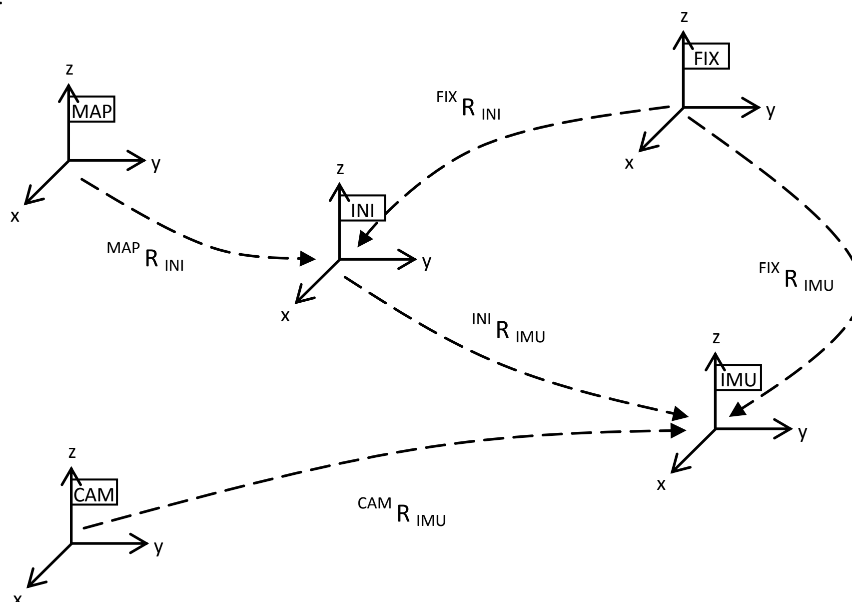

Figura 5.15. Referencias y transformaciones

El sistema de referencia CAM se usará más adelante en el procesado de la nube de puntos.

El sistema de referencia INI se corresponde con la ultima referencia IMU que ha sido localizada en el mapa. Con un periodo más largo que para los mensajes del IMU, se reciben los mensajes de la cámara. Es en ese momento en el que, como se explicará más adelante, se aplica el algoritmo ICP que determina con precisión la localización del sistema. Ese es el único punto en el que se tiene una transformación precisa entre el sistema y el mapa. Por tanto, para cualquier sucesiva transformación que proporcione el IMU se usará esta referencia como vínculo de unión a la referencia MAP. La referencia INI se actualiza cada vez que llega una nube de puntos de la cámara asignando el valor de ${}^{FIX}R_{IMU}$ a ${}^{FIX}R_{INI}$ en ese momento.

Utilizando la misma simplificación en 2D de la figura 1.2, en la figura 5.16. se indican algunas de las referencias sobre la trayectoria. En azul las referencias IMU y en rojo las referencias INI. En negro la referencia MAP.

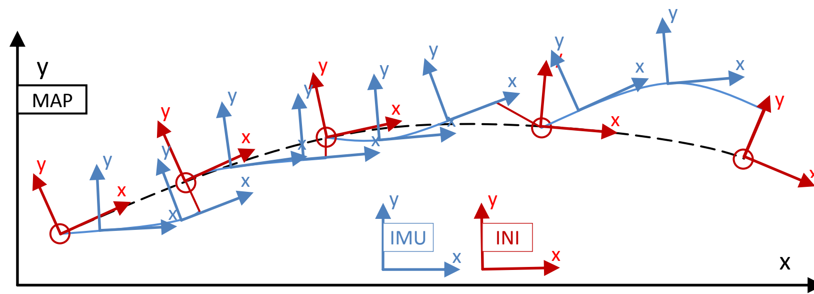

Figura 5.16. Referencias a lo largo de la trayectoria

Para llevar el vector aceleración A_{IMU} al sistema de referencia MAP se aplican las siguientes transformaciones.

$$ {}^{INI}R_{IMU} = {}^{FIX}R_{INI}{}^{-1} * {}^{FIX}R_{IMU} \quad \text{[ec. 5.8]} $$

$$ {}^{MAP}R_{IMU} = {}^{MAP}R_{INI} * {}^{INI}R_{IMU} = {}^{MAP}R_{INI} * {}^{FIX}R_{INI}{}^{-1} * {}^{FIX}R_{IMU} \quad \text{[ec. 5.9]} $$

$$ A_{MAP} = {}^{MAP}R_{IMU} * A_{IMU} \quad \text{[ec. 5.10]} $$

Integración de la aceleración

Una vez se tienen las aceleraciones en el sistema de referencia MAP, ya se conoce cual es la aceleración que toma el sistema en cada uno de los ejes x,y,z que definen el mapa. Es en este momento en el que se puede calcular el desplazamiento del sistema.

Teóricamente, para un móvil que describe una trayectoria $x(t)$, se define su velocidad, $v(t)$, como la derivada de su posición ($v(t) = dx(t)/dt$). Y de la misma manera, se define su aceleración, $a(t)$, como la derivada de su velocidad ($a(t) = dv(t)/dt$).

Realizándose la operación inversa se tiene:

$$ v(t) = \int a(t)\, dt \quad \text{[ec. 5.11]} $$

$$x(t) = \int v(t)\,dt \qquad \text{[ec. 5.12]}$$

Se dispone de la posición del sistema en cada instante, por tanto. integrando dos veces se obtiene su posición. El inconveniente es que las ecuaciones anteriores son continuas, están definidas para todo t, y las aceleraciones que entrega el IMU son discretas en el tiempo. Por tanto, se aplicará un método de integración numérica por trapecios.

La integral de una función se puede entender como el área que delimita bajo ella (Figura 1.17). La integración por trapecios estima como buena aproximación de la integral entre dos puntos, el área bajo la recta que une dos puntos consecutivos de la función (Figura 1.18).

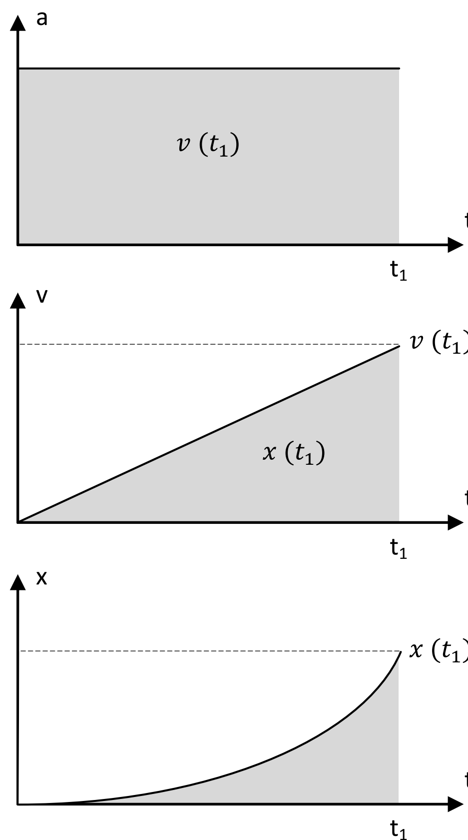

Figura 5.17. Aceleración. Velocidad. Posición

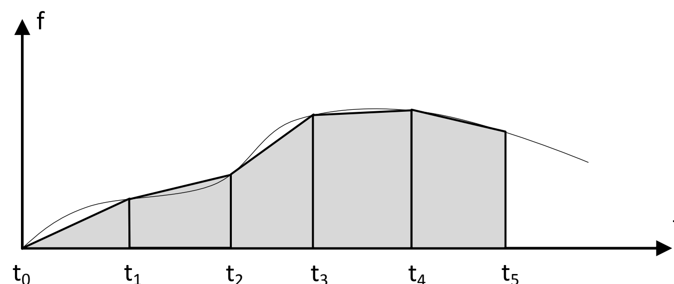

Figura 5.18. Integración numérica por trapecios

Para cada muestra se calcula la velocidad y a continuación la posición de la siguiente forma.

$$v(k) = v(k-1) + 0.5 * \big(a(k) + a(k-1)\big) * \big(t(k) - t(k-1)\big) \quad \text{[ec. 5.13]}$$

$$x(k) = x(k-1) + 0.5 * \big(v(k) + v(k-1)\big) * \big(t(k) - t(k-1)\big) \quad \text{[ec. 5.14]}$$

Este proceso de doble integración sobre una señal como es la medida de aceleración tiene sus inconvenientes. En primer lugar, al tratarse de una señal discreta y aplicar una integración numérica, se está asumiendo un error per se. Se trata de una aproximación. En segundo lugar, al tratarse de una señal con ruido, esto supone un error de la media con respecto al valor real. Al integrar se está sumando y acumulando el valor de ese error para cada muestra sucesiva. Y finalmente, en tercer lugar, el hecho de integrar dos veces, que amplifica por dos los errores anteriores.

Como el objetivo del cálculo de la posición es obtener una estimación de la posición del sistema en el mapa, para que posteriormente pueda utilizarse como punto de partida del algoritmo ICP, la existencia de un error en la integración de la aceleración puede ser asumible dentro de un rango. Se trata de una estimación.

Actualizar localización

Llegado este punto se tiene la variación de posición en coordenadas del mapa desde el punto que se consideró como origen en la inicialización del programa y el punto en el que el IMU considera que se encuentra el sistema en este momento. Como ya se ha comentado, existe un error en esa posición, y conforme más se prolongue la integración de la aceleración en el tiempo mayor será el error. La manera de corregir este error es redefiniendo el origen y la velocidad cada vez que la cámara entregue una localización más precisa. Esto corresponde al algoritmo de procesamiento de la cámara y se explicara en el apartado 4 de este mismo capítulo.

Por ahora, basta saber que el sistema conoce su posición (estimada) respecto de la ultima localización conocida y además conoce su orientación: ${}^{MAP}R_{IMU}$. Con estas dos informaciones se puede componer una matriz de transformación a la que se hará referencia como ${}^{MAP}T_{IMU}$. La matriz de transformación es una matriz 4x4 que contiene la información de traslación y rotación, así como de perspectiva y escala, que permite aplicar transformaciones de elementos expresados en un sistema de referencia a otro.

$${}^{MAP}T_{IMU} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & R_{13} & x \\ R_{21} & R_{22} & R_{23} & y \\ R_{31} & R_{32} & R_{33} & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 5.15]}$$

5. Localización del sistema - ICP

En este apartado se explica el procesamiento de la información proporcionada por la cámara.

Como ya se ha comentado antes, se utiliza la nube de puntos capturada por la cámara para, aplicando un algoritmo ICP, obtener la transformación que relaciona el mapa con la nube de puntos.

En la figura 5.19. se puede ver el diagrama de flujo de procesamiento de la nube de puntos. Para mantener una explicación ordenada se irá comentando una a una las distintas etapas del diagrama.

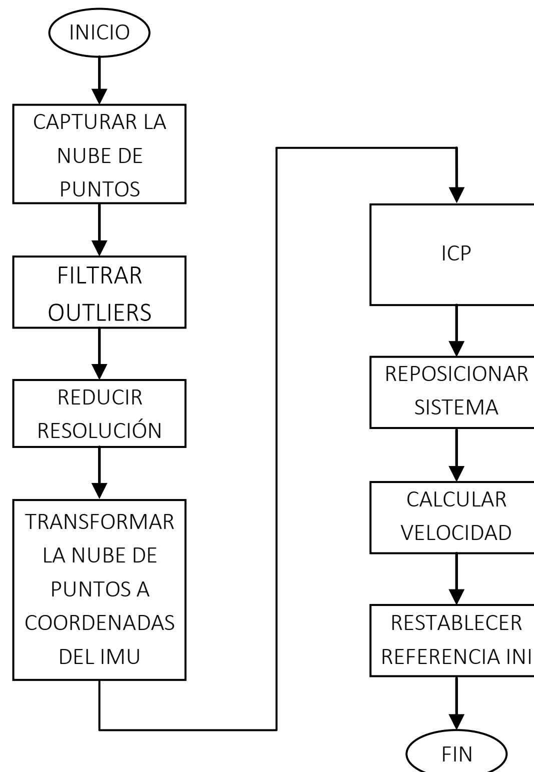

Figura 5.19. Diagrama de flujo de procesamiento de la nube de puntos

Capturar la nube de puntos

En primer lugar se captura la información de la nube de puntos. En este caso, la nube de puntos procede de un fichero .pcd del que se extrae directamente la nube de puntos en formato "pcl::PointCloud". Si se tratase de una aplicación de tiempo real la nube de puntos se recibiría en formato "sensor_msgs::PointCloud2" y habría que convertirla a formato "pcl::PointCloud" para poder manejarla. El tiempo de carga no se ha tenido en cuenta a efectos de tiempos de cómputo dado que en una supuesta aplicación de tiempo real no sería necesario realizar este paso. Por el contrario, si que se ha tenido en cuenta para el tiempo de procesamiento de cada nube, el tiempo medio de transformación de la nube de puntos de un formato a otro.

Filtrar outliers

De la misma manera que se hace en el procesamiento del mapa, se aplica una serie de filtros para eliminar los outliers.

En primer lugar, se lleva a cabo la eliminación de los puntos NaN (Not a Number). La cámara en ocasiones no es capaz de obtener el valor de profundidad para un determinado pixel, bien porque el objeto se encuentra demasiado cerca de la cámara o bien porque el sensor no recibe el infrarrojo reflejado en la escena. Estos puntos quedan grabados en la nube de puntos con el valor NaN. Para esto se aplica el filtro “pcl::removeNaNFromPointCloud”.

En segundo lugar, se lleva a cabo la eliminación de los outliers debidos a la existencia de bordes o perfiles en la escena. Estos bordes crean una estela de puntos tras ellos. Para ello se aplica el filtro “pcl::StatistcialOutlierRemoval”. Véase el apartado 3.2.1. de este mismo capítulo.

Finalmente, se aplica un tercer filtro para eliminar los puntos solitarios que quedan tras realizar todos los pasos anteriores. Se utiliza un filtro “pcl::RadiusOutlierRemoval”. Véase el apartado 3.2.2. de este mismo capítulo.

Reducir resolución

De la misma manera que se hace en el procesamiento del mapa, se aplica un filtro “Voxel Grid Filter” para reducir la resolución de la nube de puntos e igualarla a la resolución del mapa. Se reduce así el número de puntos y se normaliza la densidad de puntos para facilitar la posterior tarea de ICP. Véase el apartado 3.1. de este mismo capítulo.

Transformar la nube de puntos a coordenadas del IMU

Una vez filtrada la nube de puntos, de la misma manera que se hace con las aceleraciones, se aplican unas transformaciones para llevar la nube capturada por la cámara a coordenadas del mapa. Véase la figura 5.15 en la que se muestran todas las transformaciones.

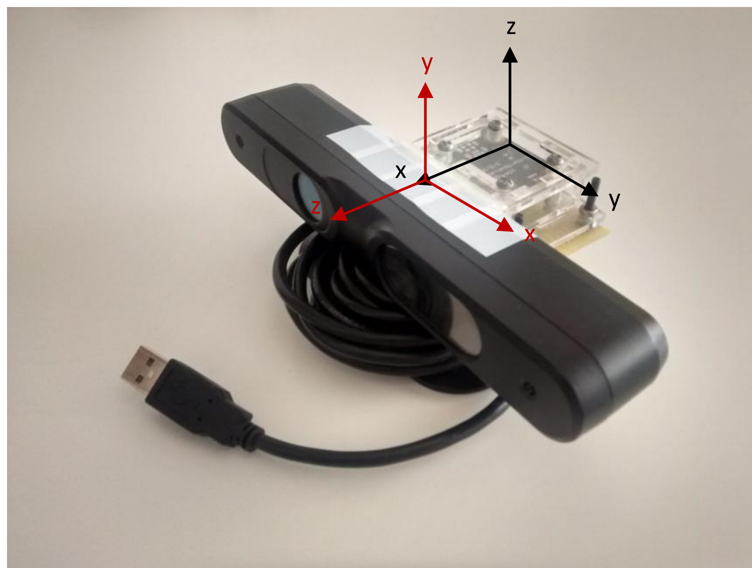

Figura 5.20. Transformación cámara – IMU

En la figura 5.20. se puede ver las referencias IMU (en negro) y CAM (en rojo). La transformación que las relaciona ${}^{CAM}T_{IMU}$ es constante y viene definida por las características físicas del conjunto cámara-IMU. Queda definida por tanto como:

$$ {}^{CAM}T_{IMU} = \begin{pmatrix} 0 & 0 & 1 & -0.06 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{[ec. 5.15]} $$

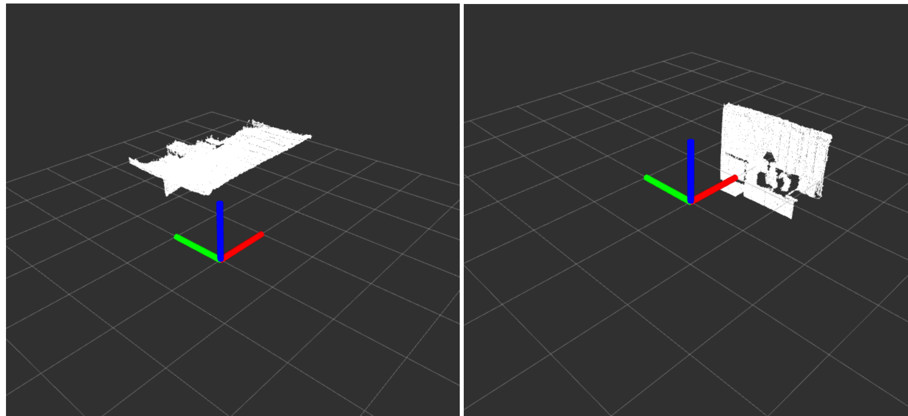

Figura 5.21. Resultado de la aplicación de la transformación ${}^{CAM}T_{IMU}$

Iterative Closest Point (ICP)

Llegado este momento, se tiene una nube de puntos en el sistema de referencia IMU, y además, se conoce la estimación de la localización a partir de las aceleraciones que proporciona el IMU. Es el momento de aplicar el, ya mencionado anteriormente, ICP.

El algoritmo ICP o Iterative Closest Point, es un algoritmo iterativo que trata de encontrar la mejor trasformación entre dos nubes de puntos. El algoritmo toma como entradas:

1. una nube de puntos, que recibe el nombre de "source" y que es la nube que se quiere alinear,
2. otra nube de puntos, que recibe el nombre de "target" y que es la nube contra la que se quiere alinear el "source",
3. una estimación de la transformación entre las dos nubes para tomarla como valor inicial al comenzar las iteraciones,
4. y, finalmente, unos criterios de parada, que pueden ser:
    1. un número máximo de iteraciones,
    2. una diferencia mínima (épsilon) entre las transformaciones obtenidas en dos iteraciones sucesivas,
    3. o, cuando el error cuadrático medio de las distancias euclídeas entre puntos correspondientes es menor que un umbral.

La manera en que funciona el algoritmo ICP es, manteniendo el "target" fijo, se van aplicando diferentes transformaciones (rotación + traslación) al "source" y en cada iteración se calcula el error entre ambas nubes y se propone una nueva transformación hasta que el algoritmo converge o se llega a alguno de los, antes mencionados, criterios de parada.

Existen diferentes maneras de calcular el error entre dos nubes de puntos. La más común y la que utiliza el algoritmo ICP propuesto en la PCL, el que se usa en este proyecto, es el error cuadrático medio de las distancias euclideas entre un punto del "target" y el punto más próximo en el "source". Existen también diversas formas de encontrar ese punto más próximo. En este caso se utiliza un algoritmo de búsqueda conocido como "k-d tree".

La estimación de la transformación a aplicar en la siguiente iteración se basa en la Descomposición en Valores Singulares (SVD – Singular Value Decomposition).

En la figura 5.21. se pueden ver tres iteraciones de ICP para alinear la nube de puntos roja ("source") con la nube de puntos negra ("target").

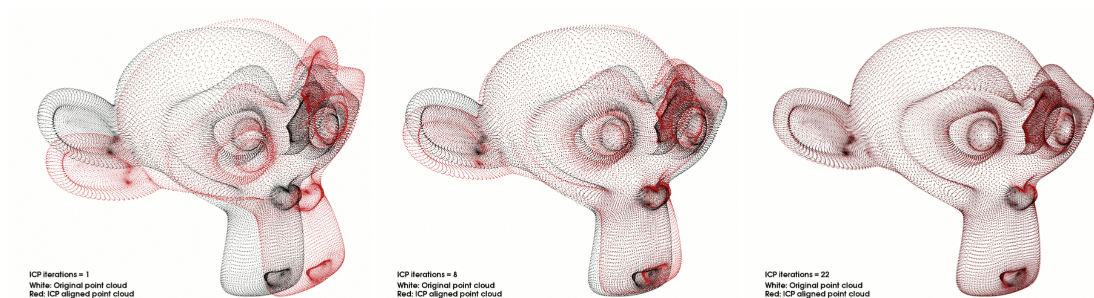

Figura 5.21. Ejemplo de iteración de ICP [16]

Para tener una explicación más detallada del algoritmo ICP véase la publicación original del algoritmo [15].

Reposicionar sistema

Si el ICP concluye con éxito, en este momento se tiene la transformación exacta entre el mapa y el sistema cámara-IMU. Como se comentaba en el apartado 3 de este capítulo, para no acumular indefinidamente el error de posición procedente de la integración de la aceleración, cada vez que se localiza la nube de puntos en el mapa, se actualizan el vector posición y el vector velocidad.

La actualización de la posición es trivial, pues se tiene directamente de la matriz de transformación obtenida con el ICP.

Calcular velocidad

Para la actualización de la velocidad si que es necesario realizar alguna suposición dado que no tenemos una medida directa de esta. La manera en que se obtiene el vector velocidad es partir de las dos últimas posiciones del sistema confirmadas por la cámara.

$$\vec{v}(k) = {\vec{x}(k) - \vec{x}(k-1)} \Big/ {t(k) - t(k-1)} \qquad \text{[ec. 5.16]}$$

Se trata, por supuesto, de una aproximación, pero a la larga resulta más preciso que mantener la velocidad procedente de la integración de la aceleración, donde el error puede crecer sin límite.

Restablecer referencia INI

Como se comentaba en el apartado 4 de este mismo capítulo, la referencia INI, que conserva la última localización confirmada por la cámara, debe ser actualizada cada vez que se procesa una nube de puntos. De esta manera se redefine su valor como:

$${}^{FIX}R_{INI} = {}^{FIX}R_{IMU} \qquad \text{[ec. 5.16]}$$

6. EJECUCIÓN

La ejecución de "tfg_processor" es configurable desde el terminal. Entre otras opciones se puede elegir el mapa y la secuencia a procesar, o el número de muestras del filtro del IMU. En la figura 5. se muestra el mensaje de ayuda de "tfg_processor".

```
Usage:   tfg_processor  <option(s)>

Options:

        -h,--help                          Show help
        -d,--debug                         Set Verbose level to DEBUG
        -i,--info                          Set Verbose level to INFO
        -e,--error                         Set Verbose level to ERROR
        -di,--debugimu                     DEBUG + step-by-step IMU
        -dc,--debugcloud                   DEBUG + step-by-step CLOUD
        -db,--debugboth                    DEBUG + step-by-step IMU + CLOUD
        -n,--nolog                         Set Verbose level to NOLOG
        -m <path>,--map <path>             Set the path to a custom map
        -s <number>, --seq <number>        Sets the sequence to be processed
         -f <number>, --filter <number>    Sets the size of the imu filter

Default settings:

        -Verbose level:                    DEBUG
        -Step-by-step:                     No IMU, no CLOUD
        -Map:                              src/tfg_v2/map/
        -Sequence:                         1 (sequence_1)
        -Filter size:                      20 (max 50)
```

Figura 5.22. Mensaje de ayuda de tfg_processor

CAPÍTULO 6

Pruebas de funcionamiento

En este capítulo se presentan las distintas pruebas realizadas sobre el sistema y sus respectivos resultados.

1. RATIO DE CAPTURA DE DATOS PARA CREAR LA SECUENCIA

Inicialmente se plantea el programa "tfg_recorder" para funcionar sobre un único proceso, pero se comprueba que es imposible recoger la totalidad de los mensajes que se publican en ambos topics, "/imu/data" y "/camera/depth/points", y se produce una reducción en la frecuencia de muestreo.

ROS ofrece la posibilidad de gestionar las interrupciones en múltiples procesos en paralelo. Se ha analizado el comportamiento del programa incrementando el número de procesos máximo que se le permite utilizar a ROS. Se resumen los resultados en la tabla 6.1.

1 Proceso	Cámara	T = 490 ms	f = 2.04 Hz
	IMU	T = 132 ms	f = 7.76 Hz
2 Procesos	Cámara	T = 552 ms	f = 1.81Hz
	IMU	T = 2 ms	f = 500 Hz
3 Procesos	Cámara	T = 564 ms	f = 1.77 Hz
	IMU	T = 2 ms	f = 500 Hz
4 Procesos	Cámara	T = 561 ms	f = 1.78 Hz
	IMU	T = 2 ms	f = 500 Hz
5 Procesos	Cámara	T = 601 ms	f = 1.66 Hz
	IMU	T = 2 ms	f = 500 Hz
6 Procesos	Cámara	T = 559 ms	f = 1.78 Hz
	IMU	T = 2 ms	f = 500 Hz

Tabla 6.1. Resultados test de multiprocesos

Teóricamente la cámara ASUS Xtion PRO LIVE graba a 30 fps (f = 30 Hz) [5] y el IMU emite a entre 250 Hz y 1000 Hz.

En la tabla 6.1. se puede ver como para un único proceso no se cumplen dichas especificaciones. Al aumentar a dos procesos, el procesamiento del IMU ya entra dentro del rango teórico, sin embargo no pasa lo mismo con la cámara. Al seguir aumentando el número de procesos no se aprecia mejora alguna. Esto tiene sentido teniendo en cuenta que el nodo únicamente está subscrito a 2 topics, y se están procesando dichos topics en procesos separados. Al aumentar por encima de 2 el número de procesos, no se les está dando realmente uso a éstos.

Los ratios obtenidos con dos procesos ya son unas frecuencias de muestreo aceptables para el propósito del proyecto. Interesa tener una alta frecuencia de muestreo del IMU, para tener mayor precisión en la estimación de la posición. Por el contrario, el algoritmo ICP puede resultar pesado como para intentar procesar las nubes de puntos a 30 Hz. Dos nubes de putos puede ser un ratio aceptable, incluso algo menor.

2. Calibración del IMU

Como se explica en el capítulo 5, las medidas de aceleración del IMU tienen unos "offsets", en otras palabras, que su rango de medida no está centrado en cero, y, por tanto, cuando cabría esperar una aceleración nula (estando el IMU en reposo) se tienen unos valores de aceleración no nulos.

Para comprobar la correcta calibración del IMU, y en caso negativo, calibrarlo, se reliza el siguiente experimento. Para cada uno de los ejes del IMU y en sus dos sentidos, dejando el IMU en reposo, se alinea el eje en cuestión con la vertical de la Tierra de tal manera que la gravedad sea solo sensada por el acelerómetro de ese eje. Se realiza una serie de 10.000 medidas para cada una de las 6 combinaciones y se calcula la media. En la tabla 6.2. se muestran los resultados obtenidos.

	Sentido Positivo	Sentido Negativo
Eje x	xp	xn
Eje y	yp	yn
Eje z	zp	zn

Tabla 6.2. Medias de calibración de los acelerómetros del IMU

A partir de los resultados obtenidos se calculan los offsets de cada acelerómetro sustituyendo en las ecuaciones 5.2, 5.3, y 5.4.

$$offset_x = xp + xn = 9.75 - 9.87 = -0.12 \quad \text{[ec. 6.1]}$$

$$offset_y = yp + yn = 9.71 - 9.89 = -0.18 \quad \text{[ec. 6.2]}$$

$$offset_z = zp + zn = 9.73 - 9.88 = -0.15 \quad \text{[ec. 6.3]}$$

3. FILTRADO DE LAS NUBES DE PUNTOS

Tanto al mapa, como a las sucesivas nubes de puntos que captura la cámara, se les aplica una serie de filtros explicados en el apartado 3 del capítulo 5. Estos filtros toman como entrada unos parámetros. A continuación se indican cuales son los valores elegidos para estos parámetros, para cada uno de los filtros.

Filtro: "Voxel Grid Filter"

Este filtro reduce la densidad de puntos de la nube y al mismo tiempo reduce la resolución. Divide el espacio en cajas de dimensiones x, y, z, dentro de las cuales solo puede haber un punto. Toma como entrada el tamaño de esas cajas. Por simplicidad se opta por utilizar cajas cúbicas, es decir con x = y = z. El número de puntos que tiene una nube está directamente relacionado con el tiempo de computo que conlleva aplicar cualquier algoritmo sobre esa nube. Se debe encontrar un valor de 'x' que permita tener tiempos de computo menores sin sacrificar la resolución de la nube en exceso.

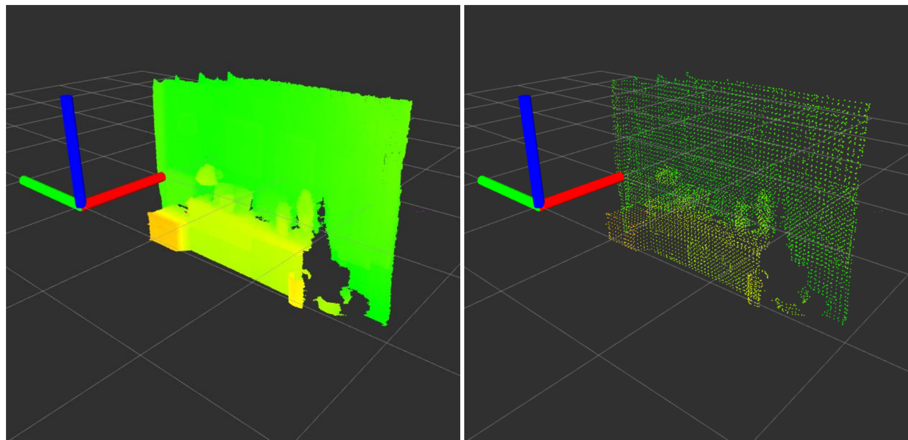

Figura 6.1. Mapa a resolución original (izq) y filtrado con x = 0.05 (der)

En la tabla 6.3. se pueden ver los tiempos de cómputo (en segundos) al aplicar el "Voxel Grid Filter" sobre una nube de puntos capturada por la cámara y al aplicar el ICP contra esa misma nube pero habiéndole aplicado una pequeña transformación T:

$$T = \begin{pmatrix} 0.975 & -0.198 & 0.099 & 0.1 \\ 0.198 & 0.980 & 0.099 & 0.1 \\ -0.099 & 0.010 & 0.995 & 0.1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{[ec. 6.4]}$$

Se han medido los tiempos para cuatro tamaños distintos de cubo. Se ve como el tiempo de "downsampling" (reduccciónm de la resolución) es prácticamente constante e igual a 0.011 segundos. Sin embargo, el tiempo de ejecuciónm del ICP incrementa rápidamente cuanto mayor es la resolución de la nube. A continuación de la tabla se muestra en forma de matriz de transformación el error entre la transformación obtenida y la realpara cada una de las resluciones. Este error se define como la trasformación que lleva de la estimada por el icp a la real.

.

Lado del cubo (x) en metros	Tiempo "downsampling"	Tiempo ICP	Iteraciones	Puntos
0.2	0.01111	0.00728	50	138
0.1	0.01146	0.02503	50	514
0.05	0.01338	0.10452	50	2140
0.025	0.01458	0.27665	50	7919
Original	0	24.68752	50	296547

Tabla 6.3. Análisis de tiempos para diferentes resoluciones

$$x = 0.2 \rightarrow Error = \begin{pmatrix} 0.999 & -0.019 & 0.005 & 0.134 \\ 0.019 & 0.999 & 0.002 & -0.009 \\ -0.005 & -0.002 & 0.999 & 0.011 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 6.5]}$$

$$x = 0.1 \rightarrow Error = \begin{pmatrix} 1 & -0.004 & 0.002 & 0.068 \\ 0.004 & 0.999 & 0.019 & 0.033 \\ -0.002 & -0.019 & 0.999 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 6.6]}$$

$$x = 0.05 \rightarrow Error = \begin{pmatrix} 1 & -0.002 & 0 & 0.038 \\ 0.002 & 1 & 0.001 & 0.003 \\ 0 & -0.001 & 1 & 0.002 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 6.7]}$$

$$x = 0.025 \rightarrow Error = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 6.8]}$$

$$Original \rightarrow Error = \begin{pmatrix} 1 & -0.004 & 0 & 0.003 \\ 0.004 & 1 & 0 & 0.002 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{[ec. 6.9]}$$

El mejor resultado se obtiene con x = 0.025, que para el caso del ejemplo consigue calcular la transformación sin error apreciable.

Filtro: "Statistical Outlier Removal Filter"

Este filtro realiza un estudio estadístico del entorno del punto y determina si ese punto es un outlier o no. Toma como parámetros de entrada el número de vecinos del punto a analizar y una constante de ponderación de la desviación típica para determinar el umbral de clasificación.

La calibración de este filtro se realiza de manera intuitiva, mirando los resultados que produce en las nubes de puntos. Si infra ajusta, no se consigue eliminar del todo los "outliers" y si se sobre ajusta, se puede eliminar puntos de la escena que no son "outliers"

Los valores elegidos para los parámetros son:

1. Número de vecinos: 50
2. Constante de la desviación: 3

En la figura 6.2. se puede ver el resultado conseguido sobre el mapa. A la izquierda el mapa sin filtrar. A la derecha, el mapa filtrado. Aunque pueda ser difícil de apreciar en la imagen, se obtiene un gran resultado eliminando los puntos que aparecen sobresaliendo de la pared, el ruido que aparece a lo largo de la superficiere (en torno a la línea naranja).

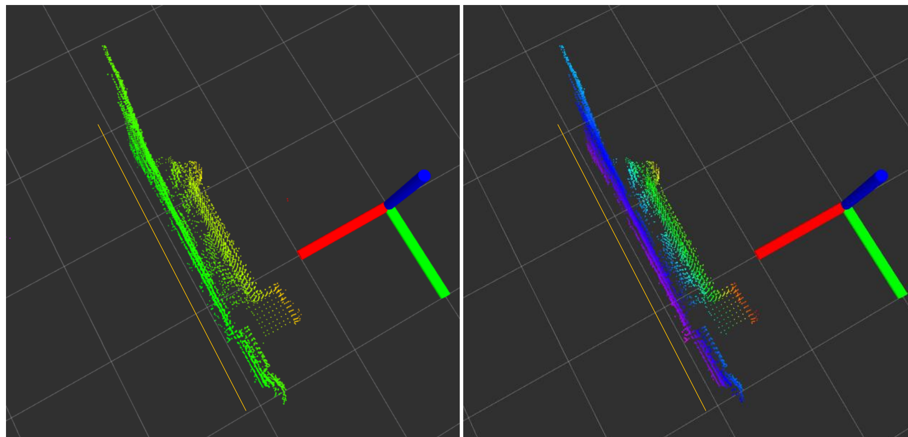

Figura 6.2. Aplicación del filtro "Statistical Outlier Removal Filter"

En la figura 6.3. Se puede ver el efecto que produce un sobre dimensionamiento del filtro. Se ha usado un número de vecinos de 100 y una constante de 1. El efecto queda visible sobre todo en la zona marcada de naranja donde el filtro ha eliminado parte de la superficie horizontal de la escena.

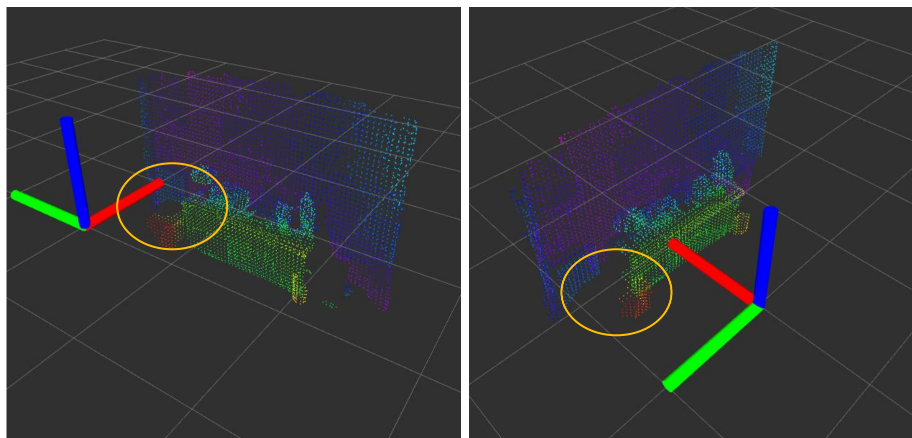

Figura 6.3. Sobredimensionamiento de los parámetros del filtro "Statistical Outlier Removal Filter"

Filtro: "Radius Outlier Removal Filter"

Este filtro elimina puntos solitarios contando si el número de puntos que lo rodean dentro de una esfera de radio 'R' es menor que un umbral. De la misma manera que el anterior, la calibración de este filtro se realiza de manera intuitiva viendo los resultados.

Si se sobre ajusta este filtro se empieza a perder el detalle en zonas de la escena con menor cantidad de puntos generalmente porque se trata de objetos pequeños. Los valores elegidos para los parámetros son:

1. Radio: 0.06
2. Vecinos: 4

En la figura 6.4. se puede ver, a la izquierda, el mapa antes de filtrar y, a la derecha, después de aplicar el filtro. Notese que se está utlizando una escala de color sobre el eje x (rojo) donde colores más calidos indican cercanía al origen y colores más fríos, lejanía. En la nube de la izquierda no aparece la gama tonal entera y esto se debe a la existencia de algún punto con 'x' pequeña o muy grande que está ensanchando el rango de valores que toma la 'x' de los puntos. La imagen e la izquierda comprende la gama tonal entera desde el rojo al azul porque se han eliminado los puntos ajenos a la escena.

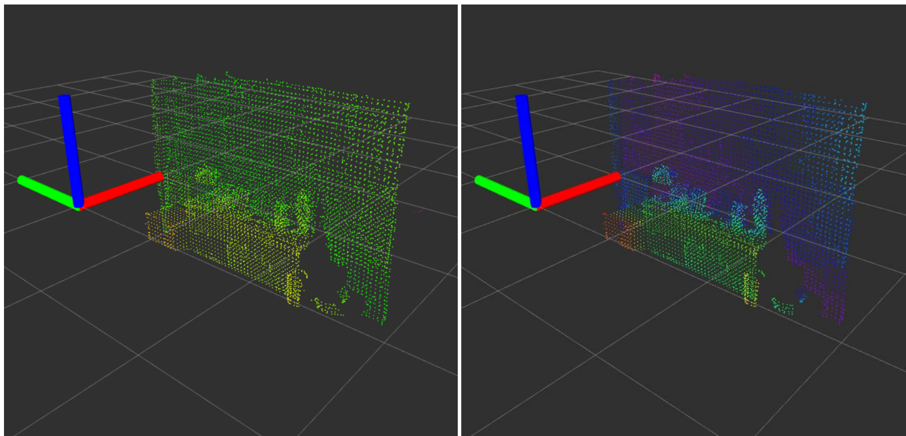

Figura 6.4. Eliminación de puntos aislados

4. ANÁLISIS DE TIEMPOS DE CÓMPUTO

En esta prueba se va a ejecutar el algoritmo completo sobre una secuencia pregrabada de un movimiento rectilíneo horizontal. Se analizará el tiempo de computo empleado en las dos etapas principales del algoritmo: la estimación de la posición con el IMU y la localización del sistema con la cámara.

En la figura 6.5. se muestra la información de salida que entrega el programa tfg_processor al procesar una secuencia de movimiento.

```
1.  ======================================================================
2.  |                                                                    |
3.  |                          TFG_PROCESSOR                             |
4.  |                                                                    |
5.  |====================================================================|
6.  |                                                                    |
7.  | Author:           David Turbica Mamblona                           |
8.  |                                                                    |
9.  | Description:      ICP-based RGBd camera location system on a       |
10. |                   known environment                                |
11. |                                                                    |
12. ======================================================================
13.
14. ----------------------------------------------------------------------
15. Loading data from the following sources:
16.     map.pcd                      src/tfg_v2/map/map.pcd
17.     origin.txt                   src/tfg_v2/map/sequence_1/origin.txt
18.     clouds/                      src/tfg_v2/map/sequence_1/clouds/
19.     clouds_data.txt              src/tfg_v2/map/sequence_1/clouds_data.txt
20.     imus_data.txt                src/tfg_v2/map/sequence_1/imus_data.txt
21.     imu_calibration_params.txt   src/tfg_v2/map/sequence_1/imu_calibration_params.txt
22.     imu_calibration.txt          src/tfg_v2/map/sequence_1/imu_calibration.txt
23.     parameters.txt               src/tfg_v2/map/sequence_1/parameters.txt
24. ----------------------------------------------------------------------
25. Checking files:
26.     map.pcd                      LOADED
27.     origin.txt                   LOADED
28.     clouds_data.txt              LOADED
29.     imus_data.txt                LOADED
30.     imu_calibration_params.txt   LOADED
31.     imu_calibration.txt          LOADED
32.         Calibrating IMU 0%
33.         Calibrating IMU 25%
34.         Calibrating IMU 50%
35.         Calibrating IMU 75%
36.         Calibrating IMU 100%
37.     parameters.txt               LOADED
38. ----------------------------------------------------------------------
39. Processing sequence:
40.       Progress    Sim.Time    Pro.Time    Time.Remaining    ICP Itera.
41.         [  4%]     0.386 s     1.756 s          40.373 s            50
42.         [ 10%]     0.940 s     3.522 s          31.266 s            50
43.         [ 15%]     1.440 s     5.288 s          28.867 s            50
44.         [ 21%]     2.007 s     7.049 s          25.702 s            50
45.         [ 27%]     2.575 s     8.831 s          23.158 s            50
46.         [ 33%]     3.142 s    10.632 s          20.927 s            50
47.         [ 39%]     3.642 s    12.421 s          19.389 s            50
48.         [ 44%]     4.176 s    14.296 s          17.621 s            50
49.         [ 50%]     4.710 s    16.199 s          15.885 s            50
50.         [ 56%]     5.311 s    18.106 s          13.707 s            50
51.         [ 62%]     5.878 s    19.988 s          11.742 s            50
52.         [ 68%]     6.412 s    21.868 s           9.946 s            50
53.         [ 74%]     6.979 s    23.787 s           8.024 s            50
54.         [ 80%]     7.513 s    25.691 s           6.216 s            50
55.         [ 86%]     8.047 s    27.606 s           4.412 s            50
56.         [100%]     9.340 s    27.607 s           0.000 s            50
57. ----------------------------------------------------------------------
```

Figura 6.5. Resultado de ejecución de tfg_processor (a)

```
58.
59. Average IMU processing time: 0.000042 s
60. Max     IMU processing time: 0.000047 s
61. Min     IMU processing time: 0.000039 s
62.
63. Average CLD processing time: 0.543277 s
64. Max     CLD processing time: 0.631403 s
65. Min     CLD processing time: 0.463613 s
66.
67. ======================================================================
68. |                                                                    |
69. |                          TFG_PROCESSOR                             |
70. |                                                                    |
71. |====================================================================|
72. |                                                                    |
73. | Author:           David Turbica Mamblona                           |
74. |                                                                    |
75. | Description:      ICP-based RGBd camera location system on a       |
76. |                   known environment                                |
77. |                                                                    |
78. |====================================================================|
79. |                                                                    |
80. | Node terminated:  The processing has successfully ended.           |
81. |                                                                    |
82. ======================================================================
83.
```

Figura 6.5. Resultado de ejecución de tfg_processor (b)

4.1. Tiempo de cómputo del IMU

El tiempo de cómputo del IMU comprende todo el proceso representado en el diagrama de flujo de la figura 5.8.

Como se puede ver en las líneas 59 – 61 de la figura 6.5, el tiempo de procesamiento de un mensaje del IMU es muy rápido. La media es de 42 ns, habiéndose registrado un tiempo mínimo de 39 ns y un tiempo máximo de 47 ns.

Como se ha visto en el apartado 1 de este capítulo, en la tabla 6.1, el periodo medio con el que se recibe un mensaje del IMU es de 2 ms. Se puede calcular un ratio de ocupación de la siguiente manera.

$$ratio_{IMU} = {}^{tiempo_cómputo}/_{periodo} = {}^{0.047ms}/_{2ms} = 0.0235 = 2.35\% \qquad \text{[ec. 6.10]}$$

Se trataría de un supuesto 2.35% de ocupación de la CPU si solo tuviese un nucleo.

4.2. Tiempo de cómputo de la cámara

El tiempo de cómputo de la cámara comprende todo el proceso representado en el diagrama de flujo de la figura 5.19, a excepción de la primera etapa de lectura de ficheros que se sutituye por una etapa de transformación de la nube de puntos de un formato "sensor_message::PointCloud2" a otro formato "pcl::PointCloud<pcl::PointXYZ>".

El tiempo de cambio de formato se ha medido a parte durante el proceso de creación de la secuencia. En media, esta etapa tarda 0.003152 s en ejecutarse. Se estima que como todas la nubes de puntos capturadas por la cámara tienen en mismo número de puntos, el tiempo de cambio de formato no variará prácticamente de una a otra. Por tanto, sumando esta medida a las que se tienen en las líneas 63 – 65 de la figura 6.5, quedan los siguientes tiempos de procesamiento de las nubes de puntos.

Tiempo medio	0.546429 s
Tiempo máximo	0.634555 s
Tiempo mínimo	0.466765 s

Tabla 6.4. Tiempos de procesamiento de la nube de puntos

En este caso si que se trata de unos tiempos bastante grandes, sobre todo comparado con el periodo con el que se están recibiendo las nubes de puntos desde la cámara. La cámara graba a 30fps, es decir, que se recibe una nube de puntos cada 33ms.

Si se calcula el ratio de ocupación de CPU de la misma manera que se hizo para el IMU, resulta lo siguiente.

$$ratio_{NUBE} = {}^{tiempo_cómputo}/_{periodo} = {}^{546.429ms}/_{33ms} = 16.5585 = 1655.85\% \qquad \text{[ec. 6.11]}$$

Esto es inviable. La única opción que cabe es inframuestrear la cámara, es decir, utilizar solo una de cada tantas nubes que captura la cámara. Supóngase que se se muestrea con un periodo de superior al tiempo de cómputo de 546.429ms. Debe ser multiplo de 33ms, que es el periodo con que se tienen imágenes de la cámara. El múltiplo de 33 más cercano a 546.429 por encima es 561 ($33*17$). El ratio de uso de CPU queda:

$$ratio_{NUBE} = {}^{tiempo_cómputo}/_{periodo} = {}^{546.429ms}/_{561ms} = 0.9740 = 97.40\% \qquad \text{[ec. 6.12]}$$

Junto al procesamiento del IMU hace un total de 99.75%, que es todavía inviable dado que junto a esto se está ejecutando ROS y todos los procesos del sistema operativo.

Una posibilidad para solventar este problema sería lanzar el procesamiento de la nube de puntos en un proceso aparte, de tal manera, que en una CPU multiprocesador, se pudiese estar ejecutando paralelamente el procesamiento de los mensajes del IMU mientras se procesa la nube de puntos. Eso sí, nunca se podría superar el periodo de 561ms entre nubes de puntos pues no tiene sentido empezar a procesar la siguiente cuando aún no ha terminado la anterior.

CAPÍTULO 7

Conclusiones y recomendaciones

1. CONCLUSIONES

A pesar del esfuerzo puesto en este proyecto, no se ha conseguido alcanzar el objetivo final de obtener de manera precisa la trayectoria del sistema. Se planteaba al comienzo la posibilidad de utilizar la aceleración que proporciona el IMU para, integrando, obtener la posición del sistema. Este proceso de doble integración (aceleración-velociad-posición) resulta ser realmente sensible al ruido, y precisamente la aceleración que proporciona el Imu es una señal muy ruidosa.

En cuanto a la localización de la nube de puntos, que proporciona la cámara, en el mapa, ésta se realiza con éxito en tanto que la estimación de la posición del sistema sea buena. Obteniéndose la posición correcta para las primeras capturas de la cámara pero perdiéndose con en movimiento del sistema. Por tanto, todo apunta a que con un mejor filtrado de la aceleración o con una medición más precisa podrían obtenerse buenos resultados.

En cuanto a los objetivos intermedios. Se planteba la necesidad de generar un mapa de una estancia para tomarlo como entorno de pruebas. El mapa se ha generado con éxito y se ha logrado manipular para obtener una versión limpia del mismo.

Se planteba la necesidad de grabar una secuencia de movimiento del sistema, de generar el algoritmo que comunicase con los sensores para obtener los datos de aceleración, orientación y visión en tiempo real. Se ha implementado como un nodo de ROS.

Se palnteaba diseñar un entorno de simulación que recogiese la secuencia de movimiento previamente grabada y fuese capaz de emular el procesamiento que se haría en una aplicación de tiempo real. Se ha diseñado y se ha podido analizar el tiempo de procesamiento necesario.

Y finalmente se planteaba la pregunta de si el algoritmo sería portable a una aplicación en tiempo real. Los tiempos de ejecución obtenidos indican que esto sería posible pero con ciertas limitaciones, como son una frecuencia baja de muestreo de la cámara y la necesidad de implementar el algoritmo en una versión multiproceso que pudiese procesar en paralelo la cámara y el IMU.

2. RECOMENDACIONES

De cara a un futuro se podría mejorar el funcionamiento del algotimo en los siguientes aspectos:

- Tratar de implementar el algoritmo utilizando otro IMU que pueda ofrecer algo más de precisión. No necesariamente que tenga un mayor resolución, sino que sea más insensible al ruido.

- Diseñar un mejor filtro para eliminar el ruido que aparece en la medida de aceleración. Posiblemente implementar un filtro de Kalman dado que se trata de un ruido blanco.

- Utilizar descriptores de más alto nivel para encontrar puntos característicos en la nube de puntos que peudan ser utilizados como una etapa de localización intermedia entre la estimación de la posición del IMU y la aplicación del ICP. Permitiendo de estea manera asumir mayores errores en la estimación de la posición.

BIBLIOGRAFÍA

[1] ASUSTeK Computers Inc. Página web oficial. Consultada a fecha 5 de septiembre de 2017.

"https://www.asus.com/es/3D-Sensor/Xtion_PRO_LIVE/"

[2] Christian Kerl. "Odometry from RGB-D Cameras for Autonomous Quadrocopters". Master's Thesis in Robotics, Cognition, Intelligence. Fakultät Für Informatik, Der Technischen Universität München. 12 de noviembre de 2012.

[3] David Tedaldi. "IMU calibration without mechanical equipment". Master's Thesis. Dipartimento di Ingegneria dell'Informazion. Universitá degli Studi di Padova. 23 de septiembre de 2013.

[4] Xsense. Página web oficial. Consultada a fecha 5 de septiembre de 2017.

"https://www.xsens.com/products/mtw-awinda/"

[5] Phidgets Inc. Página web oficial. Producto: PhidgetSpatial Precision 3/3/3 High Resolution. Consultada a fecha 5 de septiembre de 2017.

"https://www.phidgets.com/?tier=3&catid=10&pcid=8&prodid=32"

[6] Robot Operating System. ROS.org. Página web oficial. Consultada a fecha 6 de septiembre de 2017.

"http://www.ros.org/about-ros/"

[7] ROS Indigo openni2_camera. Pagina web. Visitada a fecha 7 de septiembre de 2017.

"http://wiki.ros.org/openni2_camera"

[8] ROS Indigo phidgets_imu. Página web. Visitada a fecha 7 de septiembre de 2017.

"http://wiki.ros.org/phidgets_imu"

[9] PCL. Point Cloud Library. Página web oficial. Visitada a fecha 7 de septiembre de 2017.

"http://pointclouds.org/about"

[10] ROS Indigo RGBDSlam. Página web. Visitada a fecha de 7 de septiembre de 2017.

"http://wiki.ros.org/rgbdslam"

[11] ROS Indigo Documentation sensor_msgs/Imu. Página web oficial. Visitada a fecha 10 de septiembre de 2017.

"http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html"

[12] PCL. Point Cloud Library. Voxel Grid Filter. Página web oficial. Visitada a fecha 18 de septiembre de 2017.

"http://docs.pointclouds.org/1.7.0/classpcl_1_1_voxel_grid.html"

[13] PCL. Point Cloud Library. Statistical Outlier Removal Filter. Página web oficial. Visitada a fecha 18 de septiembre de 2017.

"http://docs.pointclouds.org/1.7.0/classpcl_1_1_statistical_outlier_removal.html"

[14] PCL. Point Cloud Library. Radius Outlier Removal Filter. Página web oficial. Visitada a fecha 18 de septiembre de 2017.

"http://docs.pointclouds.org/1.7.0/classpcl_1_1_radius_outlier_removal.html"

[15] Chen, Yang; Gerard Medioni (1991). "Object modelling by registration of multiple range images". Image Vision Comput. Newton, MA, USA: Butterworth-Heinemann: 145–155.

[16] PCL. Point Cloud Library. Iterative Closest Point. ICP Documentation. Pagina web oficial. Visitada a fecha 20 de septiembre de 2017.

ANEXOS

Anexo 1. sensor_msgs/Imu 73
Anexo 2. sensor_msgs/PointCloud2 75
Anexo 3. tfg_v2_recorder.h 77
Anexo 4. tfg_v2_recorder.cpp 79
Anexo 5. tfg_v2_Cloud.h 83
Anexo 6. tfg_v2_Cloud.cpp 85
Anexo 7. tfg_v2_CloudMsg.h 87
Anexo 8. tfg_v2_CloudMsg.cpp 89
Anexo 9. tfg_v2_Drone.h 91
Anexo 10. tfg_v2_Drone.cpp 95
Anexo 11. tfg_v2_Filter.h 97
Anexo 12. tfg_v2_Filter.cpp 99
Anexo 13. tfg_v2_FilterVector3.h 101
Anexo 14. tfg_v2_FilterVector3.cpp 103
Anexo 15. tfg_v2_Imu.h 105
Anexo 16. tfg_v2_Imu.cpp 107
Anexo 17. tfg_v2_ImuMsg.h 111
Anexo 18. tfg_v2_ImuMsg.cpp 113
Anexo 19. tfg_v2_Map.h 117
Anexo 20. tfg_v2_Map.cpp 119
Anexo 21. tfg_v2_Process.h 121
Anexo 22. tfg_v2_Process.cpp 123
Anexo 23. tfg_v2_processor.h 125
Anexo 24. tfg_v2_processor.cpp 127
Anexo 25. tfg_v2_Timer.h 141
Anexo 26. tfg_v2_Timer.cpp 143
Anexo 27. tfg_v2_utilities.h 145
Anexo 28. tfg_v2_utilities.cpp 147

Anexo 1. sensor_msgs/Imu

sensor_msgs/Imu	
std_msgs/Header	header
geometry_msgs/Quaternion	orientation
float64[9]	orientation_covariance
geometry_msgs/Vector3	angular_velocity
float64[9]	angular_velocity_covariance
geometry_msgs/Vector3	linear_acceleration
float64[9]	linear_acceleration_covariance

std_msgs/Header	
uint32	seq
time	stamp
string	frame_id

geometry_msgs/Quaternion	
float64	x
float64	y
float64	z
float64	w

geometry_msgs/Vector3	
float64	x
float64	y
float64	z

Anexo 2. sensor_msgs/PointCloud2

sensor_msgs/PointCloud2	
std_msgs/Header	header
uint32	height
uint32	width
sensor_msgs/PointField[]	fields
bool	is_bigendian
uint32	point_step
uint32	row_step
uint8[]	data
bool	is_dense

std_msgs/Header	
uint32	seq
time	stamp
string	frame_id

sensor_msgs/PointField		
uint8	INT8	=1
uint8	UINT8	=2
uint8	INT16	=3
uint8	UINT16	=4
uint8	INT32	=5
uint8	UINT32	=6
uint8	FLOAT32	=7
uint8	FLOAT64	=8
string	name	
uint32	offset	
uint8	datatype	

Anexo 3. tfg_v2_recorder.h

```
//=================================================================
//
//  File:       tfg_v2_recorder.h
//  Author:     David Turbica
//
//=================================================================

#ifndef __TFG_V2_RECORDER_H_INCLUDED__
#define __TFG_V2_RECORDER_H_INCLUDED__

#include <ros/ros.h>
#include <signal.h>

//C++ includes
#include <string>
#include <boost/thread/mutex.hpp>

//ROS includes
#include <sensor_msgs/PointCloud2.h>
#include <sensor_msgs/Imu.h>

//PCL includes
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl_conversions/pcl_conversions.h>

//TFG includes
#include "tfg_v2_Logger.h"


//Global Variables
int instant; //in miliseconds
long initial_instant; //in miliseconds
bool initial_instant_set; //in miliseconds
const std::string path_clds = "src/tfg_v2/sequence/clouds/";
const std::string path_clds_data = "src/tfg_v2/sequence/clouds_data.txt";
const std::string path_imus_data = "src/tfg_v2/sequence/imus_data.txt";
const std::string path_parameter = "src/tfg_v2/sequence/parameters.txt";
boost::mutex mutex_initial_instant;

//Variables CAM
long cld_t;
int cld_ratio;
int cld_counter;
std::ofstream cld_file;

//Variables IMU
long imu_t;
int imu_ratio;
int imu_counter;
std::ofstream imu_file;


//Functions
int main (int argc, char** argv);
```

```
void processCam (const sensor_msgs::PointCloud2::ConstPtr &input);
void processImu (const sensor_msgs::Imu::ConstPtr &input);
void customSigintHandler(int sig);


#endif //__TFG_V2_RECORDER_H_INCLUDED
```

Anexo 4. tfg_v2_recorder.cpp

```
//===================================================================
//
//  File:       tfg_v2_recorder.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_recorder.h"


int main (int argc, char** argv) {
    //------------------------------------------------------------------------------
    //  Main function. It initializes the whole program and sets the timers and
    //  and topics' subscribers and publishers.
    //------------------------------------------------------------------------------

    PRINT_INFO ("==================================================================");
    PRINT_INFO ("|                                                                |");
    PRINT_INFO ("|                          TFG_RECORDER                          |");
    PRINT_INFO ("|                                                                |");
    PRINT_INFO ("|================================================================|");
    PRINT_INFO ("|                                                                |");
    PRINT_INFO ("| Author:           David Turbica Mamblona                       |");
    PRINT_INFO ("|                                                                |");
    PRINT_INFO ("| Description:      It records a sequence of PointCloud2 and IMU |");
    PRINT_INFO ("|                   ROS-messages to a file so it can be used as  |");
    PRINT_INFO ("|                   input data later on.                         |");
    PRINT_INFO ("|                                                                |");
    PRINT_INFO ("==================================================================");

    //Initialize ROS
    ros::MultiThreadedSpinner spinner(6);
    ros::init (argc, argv, "tfg_recorder");
    ros::NodeHandle nh;
    ros::NodeHandle nh_imu;
    signal(SIGINT, customSigintHandler);
    LOG_LEVEL = 0;

    //Initialize global variables
    instant = 0;
    initial_instant = 0;
    initial_instant_set = false;

    //Initialize CAM variables
    cld_ratio = 0;
    cld_counter = 1;
    cld_file.open (path_clds_data.c_str());

    //Initialize IMU variables
    imu_ratio = 0;
    imu_counter = 1;
    imu_file.open (path_imus_data.c_str());

    //Subscribers
```

```
    ros::Subscriber sub_cam = nh.subscribe ("/camera/depth/points", 1, processCam);
    ros::Subscriber sub_imu = nh_imu.subscribe ("/imu/data", 1, processImu);

    PRINT_INFO ("Node tfg_core is set up and running");

    //Spin
    spinner.spin();
}

void processCam (const sensor_msgs::PointCloud2::ConstPtr &input) {
    //-----------------------------------------------------------------------------
    //  Function called when a new message from '/camera/depth/points' topic is
    //  received.
    //-----------------------------------------------------------------------------
    PRINT_DEBUG ("Processing cloud " << cld_counter);
    //Check for the initial time
    mutex_initial_instant.lock();
    if (!initial_instant_set){
        long t = (long)input->header.stamp.sec * 1000 + (input->header.stamp.nsec / 1000000) -
initial_instant;
        initial_instant = t;
        initial_instant_set = true;
    }
    mutex_initial_instant.unlock();
    //Save the cloud
    pcl::PointCloud<pcl::PointXYZ> cloud;
    pcl::fromROSMsg (*input, cloud);
    pcl::io::savePCDFile (path_clds + "cloud_" + boost::to_string(cld_counter) + ".pcd", cloud);
    //Save the time stamp
    long t = (long)input->header.stamp.sec * 1000 + (input->header.stamp.nsec / 1000000) -
initial_instant;
    cld_file << "cloud_" + boost::to_string(cld_counter) + ".pcd " + boost::to_string(t) + "\n";
    //Calc the average rate
    cld_ratio = cld_ratio + (((int)(t - cld_t) - cld_ratio) / cld_counter);
    cld_t = t;
    cld_counter ++;
}

void processImu (const sensor_msgs::Imu::ConstPtr &input) {
    //-----------------------------------------------------------------------------
    //  Function called when a new message from '/imu/data' topic is received.
    //-----------------------------------------------------------------------------
    PRINT_DEBUG ("Processing imu " << imu_counter);
    //Check for the initial time
    mutex_initial_instant.lock();
    if (!initial_instant_set){
        long t = (long)input->header.stamp.sec * 1000 + (input->header.stamp.nsec / 1000000) -
initial_instant;
        initial_instant = t;
        initial_instant_set = true;
    }
    mutex_initial_instant.unlock();
    //Save the index
    imu_file << "imu_" + boost::to_string(imu_counter) + " ";
    //Save the time stamp
    long t = (long)input->header.stamp.sec * 1000 + (input->header.stamp.nsec / 1000000) -
initial_instant;
    imu_file << boost::to_string(t) + " ";
    //Save the imu data
    imu_file << input->orientation.x << " ";
    imu_file << input->orientation.y << " ";
    imu_file << input->orientation.z << " ";
    imu_file << input->orientation.w << " ";
    imu_file << input->angular_velocity.x << " ";
    imu_file << input->angular_velocity.y << " ";
```

```
    imu_file << input->angular_velocity.z << " ";
    imu_file << input->linear_acceleration.x << " ";
    imu_file << input->linear_acceleration.y << " ";
    imu_file << input->linear_acceleration.z << "\n";
    //Calc the average rate
    imu_ratio = imu_ratio + (((int)(t - imu_t) - imu_ratio) / imu_counter);
    imu_t = t;
    imu_counter ++;
}

void customSigintHandler(int sig){
    //----------------------------------------------------------------------------
    //  Function that overrides default ROS sigint handler. It allows to print
    //  some information when the node has been shuted down and end child
    //  processes if needed.
    //----------------------------------------------------------------------------
    //Closing files oppened
    if (cld_file.is_open()) cld_file.close();
    if (imu_file.is_open()) imu_file.close();
    //Writing parameters file
    std::ofstream param_file;
    param_file.open (path_parameter.c_str());
    if (param_file.is_open()){
        //We subtract 2 in order to prevent 'imu_counter++' from having been executed
        //and also to prevent the latest data from not being written entirely to file.
        param_file << "Cld_instances: " << cld_counter - 2 << "\n";
        param_file << "Imu_instances: " << imu_counter - 2 << "\n";
        //Closing the file
        param_file.close();
    }
    PRINT_INFO ("");
    PRINT_INFO ("Cam   period    " << cld_ratio << " ms");
    PRINT_INFO ("      ratio     " << 1000/cld_ratio << " Hz");
    PRINT_INFO ("Imu   period    " << imu_ratio << " ms");
    PRINT_INFO ("      ratio     " << 1000/imu_ratio << " Hz");
    PRINT_INFO ("");
    PRINT_INFO ("====================================================================");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("|                          TFG_RECORDER                            |");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("|==================================================================|");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("| Author:           David Turbica Mamblona                         |");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("| Description:      It records a sequence of PointCloud2 and IMU   |");
    PRINT_INFO ("|                   ROS-messages to a file so it can be used as    |");
    PRINT_INFO ("|                   input data later on.                           |");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("|==================================================================|");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("| Node terminated:  No reason given (Ctrl+c)                       |");
    PRINT_INFO ("|                                                                  |");
    PRINT_INFO ("====================================================================");
    //Overrides ROS default Sigint Handler
    ros::shutdown();
}
```

Anexo 5. tfg_v2_Cloud.h

```
//=====================================================================
//
//  File:       tfg_v2_Cloud.h
//  Author:     David Turbica
//
//=====================================================================

#ifndef __TFG_V2_CLOUD_H_INCLUDED__
#define __TFG_V2_CLOUD_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <vector>
#include <Eigen/Dense>

//ROS includes


//PCL includes
#include <pcl/io/pcd_io.h>
#include <pcl/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/statistical_outlier_removal.h>
#include <pcl/filters/radius_outlier_removal.h>
#include <pcl/common/transforms.h>

//TFG includes
#include "tfg_v2_Logger.h"


class Cloud{
    //Variables
    public:
        pcl::PointCloud<pcl::PointXYZ>::Ptr point_cloud;

    //Functions
    public:
        Cloud (void);
        Cloud (pcl::PointCloud<pcl::PointXYZ>::Ptr new_point_cloud);
        bool loadCloud (std::string path);
        void saveCloud (std::string path);
        void downsample (double resolution);
        void removeOutliersStatisticaly (int neighbours = 50, double dev_const = 1.0);
        void removeOutliersRadius (double radius = 0.1, int points = 10);
        void applyTransform (Eigen::Matrix4f transformation_matrix);
        void drawPoint (float x, float y, float z);
        friend std::ostream & operator << (std::ostream &os, const Cloud &c){
            os << "Cloud:  points[]: " << c.point_cloud->points.size () << std::endl
               << "        width: " << c.point_cloud->width << std::endl
               << "        height: " << c.point_cloud->height << std::endl
               << "        is_dense: " << c.point_cloud->is_dense << std::endl
```

```
            << "        sensor origin (xyz): ["
                                    << c.point_cloud->sensor_origin_.x () << ", "
                                    << c.point_cloud->sensor_origin_.y () << ", "
                                    << c.point_cloud->sensor_origin_.z () << "] / orientation
(xyzw): ["
                                    << c.point_cloud->sensor_orientation_.x () << ", "
                                    << c.point_cloud->sensor_orientation_.y () << ", "
                                    << c.point_cloud->sensor_orientation_.z () << ", "
                                    << c.point_cloud->sensor_orientation_.w () << "]";
            return os;
        }
};

#endif //__TFG_V2_CLOUD_H_INCLUDED
```

Anexo 6. tfg_v2_Cloud.cpp

```
//=================================================================
//
//  File:       tfg_v2_Cloud.cpp
//  Author:     David Turbica
//
//=================================================================

#include "tfg_v2_Cloud.h"

Cloud::Cloud (void){
    //------------------------------------------------------------------------------
    //  Default constructor.
    //------------------------------------------------------------------------------
    this->point_cloud = pcl::PointCloud<pcl::PointXYZ>::Ptr (new
pcl::PointCloud<pcl::PointXYZ>);
}

Cloud::Cloud (pcl::PointCloud<pcl::PointXYZ>::Ptr new_point_cloud){
    //------------------------------------------------------------------------------
    //  Constructor providing a PointCloud Pointer.
    //------------------------------------------------------------------------------
    this->point_cloud = new_point_cloud;
}

bool Cloud::loadCloud (std::string path){
    //------------------------------------------------------------------------------
    //  It loads a PointCloud from a PCD file. It removes NaN points. Returns true
    //  when PointCloud was loaded successfully, false when it was not.
    //------------------------------------------------------------------------------
    bool output = true;
    if (pcl::io::loadPCDFile<pcl::PointXYZ> (path, *this->point_cloud) == -1){
        output = false;
    }
    else{
        std::vector<int> index;
        pcl::removeNaNFromPointCloud (*this->point_cloud, *this->point_cloud, index);
    }
    return output;
}

void Cloud::saveCloud (std::string path){
    //------------------------------------------------------------------------------
    //  Saves a PointCloud to a .pcd file.
    //------------------------------------------------------------------------------
    pcl::io::savePCDFile (path, *this->point_cloud);
}

void Cloud::downsample (double resolution){
    //------------------------------------------------------------------------------
    //  Downsamples its PointCloud.
    //------------------------------------------------------------------------------
    pcl::PointCloud<pcl::PointXYZ>::Ptr point_cloud_filtered (new
pcl::PointCloud<pcl::PointXYZ>);
    pcl::VoxelGrid<pcl::PointXYZ> filter;
```

```
    filter.setInputCloud (this->point_cloud);
    filter.setLeafSize (resolution, resolution, resolution);
    filter.filter (*point_cloud_filtered);
    this->point_cloud = point_cloud_filtered;
}

void Cloud::removeOutliersStatisticaly (int neighbours, double dev_const){
    //----------------------------------------------------------------------------
    //  Statistical filter to remove outliers
    //----------------------------------------------------------------------------
    pcl::PointCloud<pcl::PointXYZ>::Ptr point_cloud_filtered (new
pcl::PointCloud<pcl::PointXYZ>);
    pcl::StatisticalOutlierRemoval<pcl::PointXYZ> filter;
    filter.setInputCloud (this->point_cloud);
    filter.setMeanK (neighbours);
    filter.setStddevMulThresh (dev_const);
    filter.filter (*point_cloud_filtered);
    this->point_cloud = point_cloud_filtered;
}

void Cloud::removeOutliersRadius (double radius, int points){
    //----------------------------------------------------------------------------
    //  Remove lonely outliers
    //----------------------------------------------------------------------------
    pcl::PointCloud<pcl::PointXYZ>::Ptr point_cloud_filtered (new
pcl::PointCloud<pcl::PointXYZ>);
    pcl::RadiusOutlierRemoval<pcl::PointXYZ> filter;
    filter.setInputCloud (this->point_cloud);
    filter.setRadiusSearch (radius);
    filter.setMinNeighborsInRadius (points);
    filter.filter (*point_cloud_filtered);
    this->point_cloud = point_cloud_filtered;
}

void Cloud::applyTransform (Eigen::Matrix4f transformation_matrix){
    //----------------------------------------------------------------------------
    //  Applies a Transformation to its point_cloud.
    //----------------------------------------------------------------------------
    pcl::transformPointCloud (*this->point_cloud, *this->point_cloud, transformation_matrix);
}

void Cloud::drawPoint (float x, float y, float z){
    this->point_cloud->push_back(pcl::PointXYZ (x, y, z));
}
```

Anexo 7. tfg_v2_CloudMsg.h

```
//===================================================================
//
//  File:       tfg_v2_CloudMsg.h
//  Author:     David Turbica
//
//===================================================================

#ifndef __TFG_V2_CLOUDMSG_H_INCLUDED__
#define __TFG_V2_CLOUDMSG_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <ostream>

//ROS includes


//PCL includes


//TFG includes



class CloudMsg{
    //Variables
    private:
        int time;
        std::string path;

    //Functions
    public:
        CloudMsg (void);
        void setTime (int new_time);
        void setPath (std::string new_path);
        int getTime (void);
        std::string getPath (void);
        friend std::ostream & operator << (std::ostream &os, const CloudMsg &cm){
            os << "Cloud:  Time:  " << cm.time << "\n"
               << "        Path:  " << cm.path;
            return os;
        }
};

#endif //__TFG_V2_CLOUDMSG_H_INCLUDED
```

Anexo 8. tfg_v2_CloudMsg.cpp

```
//==================================================================
//
//  File:       tfg_v2_CloudMsg.cpp
//  Author:     David Turbica
//
//==================================================================

#include "tfg_v2_CloudMsg.h"

CloudMsg::CloudMsg (void){
    //---------------------------------------------------------------------------
    //  Default constructor.
    //---------------------------------------------------------------------------
    this->time = 0;
    this->path = "";
}

void CloudMsg::setTime (int new_time){
    //---------------------------------------------------------------------------
    //  Sets 'time' value
    //---------------------------------------------------------------------------
    this->time = new_time;
}

void CloudMsg::setPath (std::string new_path){
    //---------------------------------------------------------------------------
    //  Sest 'path' value
    //---------------------------------------------------------------------------
    this->path = new_path;
}

int CloudMsg::getTime (void){
    //---------------------------------------------------------------------------
    //  Returns its 'time' value
    //---------------------------------------------------------------------------
    return this->time;
}

std::string CloudMsg::getPath (void){
    //---------------------------------------------------------------------------
    //  Returns its 'path' value
    //---------------------------------------------------------------------------
    return this->path;
}
```

Anexo 9. tfg_v2_Drone.h

```
//===================================================================
//
//  File:       tfg_v2_Drone.h
//  Author:     David Turbica
//
//===================================================================

#ifndef __TFG_V2_DRONE_H_INCLUDED__
#define __TFG_V2_DRONE_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <ostream>
#include <vector>
#include <tf2/LinearMath/Matrix3x3.h>
#include <tf2/LinearMath/Transform.h>
#include <tf2/LinearMath/Vector3.h>

//ROS includes


//PCL includes


//TFG includes
#include "tfg_v2_Map.h"


class Drone{
    //Variables
    private:
        std::vector<tf2::Transform> locations;
        std::vector<bool> checked;
        tf2::Transform location;
        tf2::Transform location_before;
        tf2::Transform location_estimation;
    //Functions
    public:
        Drone (void);
        void setLocation (tf2::Transform loc);
        void setLocationEstimation (tf2::Transform loc);
        tf2::Transform getLocation (void);
        tf2::Transform getLocationEstimation (void);
        tf2::Vector3 getVelocityEstimation (float period);
        void drawTrayectory (Map &map);
        friend std::ostream & operator << (std::ostream &os, const Drone &d){
            os  << "Drone:  First Location:"
            << std::setw(20) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[0][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[0][1]
```

```
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[0][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getOrigin()[0]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[1][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[1][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[1][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getOrigin()[1]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[2][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[2][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getBasis()[2][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.locations.front().getOrigin()[2]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 1.0
            << std::endl
            << std::endl
                << "        Last Checked Location:"
            << std::setw(13) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[0][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[0][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[0][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getOrigin()[0]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[1][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[1][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[1][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getOrigin()[1]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[2][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[2][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getBasis()[2][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location.getOrigin()[2]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 1.0
            << std::endl
            << std::endl
                << "        Location Estimation:"
```

```
            << std::setw(15) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[0][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[0][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[0][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getOrigin()[0]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[1][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[1][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[1][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getOrigin()[1]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[2][0]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[2][1]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getBasis()[2][2]
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) <<
d.location_estimation.getOrigin()[2]
            << std::endl
            << std::setw(43) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 0.0
            << std::setw(14) << std::right << std::fixed << std::setprecision(5) << 1.0;
            return os;
        }
};

#endif //__TFG_V2_DRONE_H_INCLUDED
```

Anexo 10. tfg_v2_Drone.cpp

```
//=====================================================================
//
//  File:       tfg_v2_Drone.cpp
//  Author:     David Turbica
//
//=====================================================================

#include "tfg_v2_Drone.h"

Drone::Drone (void){
    //----------------------------------------------------------------------------
    //  Default constructor.
    //----------------------------------------------------------------------------
}

void Drone::setLocation (tf2::Transform loc){
    //----------------------------------------------------------------------------
    //  Sets a new location. Adds that new location to the trayectory.
    //----------------------------------------------------------------------------
    this->location_before = this->location;
    this->location = loc;
    this->locations.push_back (loc);
    this->checked.push_back (true);
}

void Drone::setLocationEstimation (tf2::Transform loc){
    //----------------------------------------------------------------------------
    //  Sets a new location estimation. Adds that new location to the trayectory.
    //----------------------------------------------------------------------------
    this->location_estimation = loc;
    this->locations.push_back (loc);
    this->checked.push_back (false);
}

tf2::Transform Drone::getLocation (void){
    //----------------------------------------------------------------------------
    //  Returns its 'location' value.
    //----------------------------------------------------------------------------
    return this->location;
}

tf2::Transform Drone::getLocationEstimation (void){
    //----------------------------------------------------------------------------
    //  Returns its 'location_estimation' value.
    //----------------------------------------------------------------------------
    return this->location_estimation;
}

tf2::Vector3 Drone::getVelocityEstimation (float period){
    //----------------------------------------------------------------------------
    //  It computes the to latest locations checked to estimate velocity.
    //----------------------------------------------------------------------------
    return (this->location.getOrigin() - this->location_before.getOrigin()) / period;
}
```

```
void Drone::drawTrayectory (Map &map){
    //---------------------------------------------------------------------------
    //  It draws drone trayectory points in its cloud.
    //---------------------------------------------------------------------------
    std::vector<tf2::Transform>::iterator it_loc = this->locations.begin();
    std::vector<bool>::iterator it_che = this->checked.begin();
    while (it_loc != this->locations.end() && it_che != this->checked.end()){
        map.drawLocation (*it_loc, *it_che);
        it_loc ++;
        it_che ++;
    }
}
```

Anexo 11. tfg_v2_Filter.h

```
//=====================================================================
//
//  File:       tfg_v2_Filter.h
//  Author:     David Turbica
//
//=====================================================================

#ifndef __TFG_V2_FILTER_H_INCLUDED__
#define __TFG_V2_FILTER_H_INCLUDED__

class Filter{
    //Variables
    public:
        double measures [50];
        int number_of_measures;
        int iterator;
        bool full;
    //Functions
    public:
        Filter (void);
        Filter (int new_number_of_measures);
        void avoidStartAttenuance (void);
        void addMeasure (double new_measure);
        double calcMean (void);
        double calcMedian (void);
};

#endif //__TFG_V2_FILTER_H_INCLUDED
```

Anexo 12. tfg_v2_Filter.cpp

```
//===================================================================
//
//  File:       tfg_v2_Filter.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_Filter.h"

Filter::Filter (void){
    //-----------------------------------------------------------------------------
    //  Default constructor
    //-----------------------------------------------------------------------------
    this->number_of_measures = 0;
    this->iterator = 0;
    this->full = true;
}

Filter::Filter (int new_number_of_measures){
    //-----------------------------------------------------------------------------
    //  Constructor given the lenght of the filter, the number of measures to be
    //  processed.
    //-----------------------------------------------------------------------------
    this->number_of_measures = new_number_of_measures;
    this->iterator = 0;
    this->full = true;
    for(int i=0; i<this->number_of_measures; i++){
        this->measures[i] = 0.0;
    }
}

void Filter::avoidStartAttenuance (void){
    //-----------------------------------------------------------------------------
    //  It will make the filter only compute measures that had already been added,
    //  preventing the filter to compute as zero all the measures not already set.
    //-----------------------------------------------------------------------------
    this->full = false;
}

void Filter::addMeasure (double new_measure){
    //-----------------------------------------------------------------------------
    //  It adds cyclicly a new measure to the filter. FIFO stack.
    //-----------------------------------------------------------------------------
    this->measures[iterator] = new_measure;
    this->iterator = (this->iterator + 1) % this->number_of_measures;
    if (!this->full) if (iterator == 0) this->full = true;
}

double Filter::calcMean (void){
    //-----------------------------------------------------------------------------
    //  It returns the mean of all the measures contained in the filter, even
    //  not already set measures which will be computed as cero. Unless
    //  'avoidStartAttenuance ()' had been called, where only added measures
    //  will be computed.
```

```
    //------------------------------------------------------------------------
    double result = 0.0;
    int last_measure = this->number_of_measures;
    if (!this->full) last_measure = this->iterator;
    for(int i=0; i<last_measure; i++){
        result += this->measures[i];
    }
    result = result / last_measure;
    return result;
}

double Filter::calcMedian (void){
    //------------------------------------------------------------------------
    //  It returns the median of all the measures contained in the filter, even
    //  not already set measures which will be computed as cero. Unless
    //  'avoidStartAttenuance ()' had been called, where only added measures
    //  will be computed.
    //------------------------------------------------------------------------
    double result = 0.0;
    int last_measure = this->number_of_measures;
    if (!this->full) last_measure = this->iterator;
    if(last_measure > 0){
        double sorted_measures[50];
        int number_of_sorted_measures = 1;
        sorted_measures[0] = this->measures[0];
        for(int i=1; i<last_measure; i++){
            bool found = false;
            for(int e=number_of_sorted_measures-1; e>=0 && !found; e--){
                if(this->measures[i]>sorted_measures[e]){
                    sorted_measures[e+1] = sorted_measures[e];
                }
                else{
                    sorted_measures[e+1] = this->measures[i];
                    found = true;
                }
            }
            if(!found){
                sorted_measures[0] = this->measures[i];
            }
            number_of_sorted_measures ++;
        }
        result = sorted_measures[number_of_sorted_measures/2];
    }
    return result;
}
```

Anexo 13. tfg_v2_FilterVector3.h

```
//=================================================================
//
//  File:       tfg_v2_FilterVector3.h
//  Author:     David Turbica
//
//=================================================================

#ifndef __TFG_V2_FILTERVECTOR3_H_INCLUDED__
#define __TFG_V2_FILTERVECTOR3_H_INCLUDED__

//C++ includes
#include <tf2/LinearMath/Vector3.h>

//ROS includes


//PCL includes


//TFG includes
#include "tfg_v2_Filter.h"


class FilterVector3{
    //Variables
    public:
        Filter filter_x;
        Filter filter_y;
        Filter filter_z;
    //Functions
    public:
        FilterVector3 (void);
        FilterVector3 (int new_number_of_measures);
        void avoidStartAttenuance (void);
        void addMeasure (tf2::Vector3 new_measure);
        tf2::Vector3 calcMean (void);
        tf2::Vector3 calcMedian (void);
};

#endif //__TFG_V2_FILTERVECTOR3_H_INCLUDED
```

Anexo 14. tfg_v2_FilterVector3.cpp

```
//=================================================================
//
//  File:       tfg_v2_FilterVector3.cpp
//  Author:     David Turbica
//
//=================================================================

#include "tfg_v2_FilterVector3.h"

FilterVector3::FilterVector3 (void){
    //---------------------------------------------------------------------------
    //  Default constructor.
    //---------------------------------------------------------------------------
    this->filter_x = Filter();
    this->filter_y = Filter();
    this->filter_z = Filter();
}

FilterVector3::FilterVector3 (int new_number_of_measures){
    //---------------------------------------------------------------------------
    //  Constructor given the lenght of the filter, the number of meassures to be
    //  processed.
    //---------------------------------------------------------------------------
    this->filter_x = Filter (new_number_of_measures);
    this->filter_y = Filter (new_number_of_measures);
    this->filter_z = Filter (new_number_of_measures);
}

void FilterVector3::avoidStartAttenuance (void){
    //---------------------------------------------------------------------------
    //  It will make the filter only compute measures that had already been added,
    //  preventing the filter to compute as zero all the measures not already set.
    //---------------------------------------------------------------------------
    this->filter_x.avoidStartAttenuance();
    this->filter_y.avoidStartAttenuance();
    this->filter_z.avoidStartAttenuance();
}

void FilterVector3::addMeasure (tf2::Vector3 new_measure){
    //---------------------------------------------------------------------------
    //  It adds cyclicly a new meassure to the filter. FIFO stack.
    //---------------------------------------------------------------------------
    this->filter_x.addMeasure (new_measure.getX());
    this->filter_y.addMeasure (new_measure.getY());
    this->filter_z.addMeasure (new_measure.getZ());
}

tf2::Vector3 FilterVector3::calcMean (void){
    //---------------------------------------------------------------------------
    //  It returns the mean of all the meassures contained in the filter, even not
    //  already set meassures which will be computed as vectors (0,0,0).
    //---------------------------------------------------------------------------
    tf2::Vector3 output;
    output.setX(this->filter_x.calcMean());
```

```
    output.setY(this->filter_y.calcMean());
    output.setZ(this->filter_z.calcMean());
    return output;
}

tf2::Vector3 FilterVector3::calcMedian (void){
    //---------------------------------------------------------------------------
    //  It returns the median of all the meassures contained in the filter, even
    //  not already set meassures which will be computed as vectors (0,0,0).
    //---------------------------------------------------------------------------
    tf2::Vector3 output;
    output.setX(this->filter_x.calcMedian());
    output.setY(this->filter_y.calcMedian());
    output.setZ(this->filter_z.calcMedian());
    return output;
}
```

Anexo 15. tfg_v2_Imu.h

```
//==================================================================
//
//  File:       tfg_v2_Imu.h
//  Author:     David Turbica
//
//==================================================================

#ifndef __TFG_V2_IMU_H_INCLUDED__
#define __TFG_V2_IMU_H_INCLUDED__

//C++ includes
#include <string>
#include <ostream>
#include <tf2/LinearMath/Vector3.h>
#include <tf2/LinearMath/Matrix3x3.h>

//ROS includes


//PCL includes


//TFG includes
#include "tfg_v2_utilities.h"
#include "tfg_v2_Logger.h"


class Imu{
    //Variables
    private:
        //Calibration variables
        bool calibrated;
        int counter;
        int number_calibration_samples;
        tf2::Vector3 center_offset;
        tf2::Vector3 calibration_offset;
        //Integration variables
        int time;
        tf2::Vector3 acceleration;
        tf2::Vector3 velocity;
        tf2::Vector3 position;
        //Angle offset
        tf2::Matrix3x3 angle_offset;
    //Functions
    public:
        Imu ();
        //Calibration functions
        void setNumberCalibrationSamples (int value);
        tf2::Vector3 removeOffsets (tf2::Matrix3x3 rotation, tf2::Vector3 acceleration);
        void center (tf2::Vector3 acceleration);
        int calibrate (tf2::Vector3 acceleration);
        //Integration functions
        void update (tf2::Vector3 current_acceleration, int current_time_stamp);
        tf2::Vector3 calcVelocityVariation (tf2::Vector3 previus_acceleration,
```

```
                                         tf2::Vector3 acceleration,
                                         int period_ms);
        tf2::Vector3 calcPositionVariation (tf2::Vector3 previus_velocity,
                                         tf2::Vector3 velocity,
                                         int period_ms);
        tf2::Vector3 getPosition (void);
        void reposition(tf2::Vector3 pos, tf2::Vector3 vel = tf2::Vector3 (0.0, 0.0, 0.0));
        //Angle offset
        tf2::Matrix3x3 getAngleOffset (void);
        void setAngleOffset (tf2::Matrix3x3 new_offset);
        //Others
        friend std::ostream & operator << (std::ostream &os, const Imu &i){
            os << "Imu:    acceleration:    " << PRINT_VECTOR3(i.acceleration) << std::endl
               << "        velocity:        " << PRINT_VECTOR3(i.velocity) << std::endl
               << "        position:        " << PRINT_VECTOR3(i.position) << std::endl
               << "        calibration:     " << PRINT_VECTOR3(i.center_offset) << std::endl
               << "        is calibrated:   " << std::boolalpha << i.calibrated
                                              << std::noboolalpha << " ["
                                              << i.counter << "/"
                                              << i.number_calibration_samples
                                              << "]" << std::endl
               << "        calibration:     " << PRINT_VECTOR3(i.calibration_offset);
            return os;
        }
};

#endif //__TFG_V2_IMU_H_INCLUDED
```

Anexo 16. tfg_v2_Imu.cpp

```
//===================================================================
//
//  File:       tfg_v2_Imu.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_Imu.h"

Imu::Imu (){
    //-----------------------------------------------------------------------------
    //  Default constructor.
    //-----------------------------------------------------------------------------
    //Calibration variables
    this->calibrated = false;
    this->counter = 0;
    this->number_calibration_samples = 0;
    this->center_offset = tf2::Vector3(0,0,0);
    this->calibration_offset = tf2::Vector3(0,0,0);
    //Integration variables
    this->time = 0;
    this->acceleration = tf2::Vector3(0,0,0);
    this->velocity = tf2::Vector3(0,0,0);
    this->position = tf2::Vector3(0,0,0);
    //Angle offset
    this->angle_offset.setIdentity();
}

//============================================================================
//  Calibration functions
//============================================================================

void Imu::setNumberCalibrationSamples (int value){
    //-----------------------------------------------------------------------------
    //  Sets 'number_calibration_samples' value
    //-----------------------------------------------------------------------------
    this->number_calibration_samples = value;
}

tf2::Vector3 Imu::removeOffsets (tf2::Matrix3x3 rotation, tf2::Vector3 acceleration){
    //-----------------------------------------------------------------------------
    //  Returns acceleration after removing both offsets. It receives the rotation
    //  matrix from some fixed axes where 'calibrate()' was called, and the
    //  acceleration in those relative axes 'rotation' is refering.
    //-----------------------------------------------------------------------------
    //Remove relative offset
    acceleration = acceleration - center_offset;
    //To prevent a worng output while calibrating is still in process
    if (this->calibrated){
        //Get acceleration in IMU fixed axes
        acceleration = rotation * acceleration;
        //Remove static calibration
        acceleration = acceleration - calibration_offset;
        //Get acceleration back in IMU relative axes
```

```
        acceleration = rotation.inverse() * acceleration;
    }
    return acceleration;
}

void Imu::center (tf2::Vector3 acceleration){
    //----------------------------------------------------------------------------
    //  Sets 'center_offset' value.
    //----------------------------------------------------------------------------
    this->center_offset = acceleration;
}

int Imu::calibrate (tf2::Vector3 acceleration){
    //----------------------------------------------------------------------------
    //  It calculate the mean of 'number_calibration_samples' meassures. It adds a
    //  a meassure every time this function is called. In order to calibrate the
    //  imu this functions has to be called 'number_calibration_samples' times.
    //  It returns the progress (%) of the calibration process.
    //----------------------------------------------------------------------------
    if (!this->calibrated){
        this->calibration_offset = this->calibration_offset + acceleration;
        this->counter ++;
        if(this->counter == this->number_calibration_samples){
            this->calibration_offset = this->calibration_offset / this-
>number_calibration_samples;
            this->calibrated = true;
        }
    }
    return (int)(100 * this->counter / this->number_calibration_samples);
}

//=============================================================================
//  Integration functions
//=============================================================================
void Imu::update (tf2::Vector3 current_acceleration, int current_time){
    //----------------------------------------------------------------------------
    //  It integrates the current acceleration in order to get velocity and
    //  position values.
    //----------------------------------------------------------------------------
    int period_ms;
    int previus_time;
    tf2::Vector3 current_velocity (0,0,0);
    tf2::Vector3 current_position (0,0,0);
    tf2::Vector3 velocity_variation (0,0,0);
    tf2::Vector3 position_variation (0,0,0);
    if(current_time < this->time){
        previus_time = this->time - 1000;
    }else{
        previus_time = this->time;
    }
    period_ms = current_time - previus_time;
    velocity_variation = this->calcVelocityVariation (this->acceleration, current_acceleration,
period_ms);
    current_velocity = this->velocity + velocity_variation;
    position_variation = this->calcPositionVariation (this->velocity, current_velocity,
period_ms);
    current_position = this->position + position_variation;
    this->time = current_time;
    this->acceleration = current_acceleration;
    this->velocity = current_velocity;
    this->position = current_position;
}

tf2::Vector3 Imu::calcVelocityVariation (tf2::Vector3 previus_acceleration, tf2::Vector3
acceleration, int period_ms){
```

```
    //---------------------------------------------------------------------------
    //  It returns a vector containing the result of integrating acceleration. It
    //  uses discrete trapezoidal integratiobn.
    //---------------------------------------------------------------------------
    tf2::Vector3 velocity;
    velocity.setX(0.5 * (acceleration.getX() + previus_acceleration.getX()) * (double)period_ms
/ 1000.0);
    velocity.setY(0.5 * (acceleration.getY() + previus_acceleration.getY()) * (double)period_ms
/ 1000.0);
    velocity.setZ(0.5 * (acceleration.getZ() + previus_acceleration.getZ()) * (double)period_ms
/ 1000.0);
    //velocity = roundVector3 (velocity, 3);
    return velocity;
}

tf2::Vector3 Imu::calcPositionVariation (tf2::Vector3 previus_velocity, tf2::Vector3 velocity,
int period_ms){
    //---------------------------------------------------------------------------
    //  It returns a vector containing the result of integrating velocity. It
    //  uses discrete trapezoidal integratiobn.
    //---------------------------------------------------------------------------
    tf2::Vector3 position;
    position.setX(0.5 * (velocity.getX() + previus_velocity.getX()) * (double)period_ms /
1000.0);
    position.setY(0.5 * (velocity.getY() + previus_velocity.getY()) * (double)period_ms /
1000.0);
    position.setZ(0.5 * (velocity.getZ() + previus_velocity.getZ()) * (double)period_ms /
1000.0);
    //position = roundVector3 (position, 3);
    return position;
}

tf2::Vector3 Imu::getPosition (void){
    //---------------------------------------------------------------------------
    //  Returns its 'position' value
    //---------------------------------------------------------------------------
    return this->position;
}

void Imu::reposition (tf2::Vector3 pos, tf2::Vector3 vel){
    //---------------------------------------------------------------------------
    //  Sets 'velocity' and 'position' values
    //---------------------------------------------------------------------------
    this->velocity = vel;
    this->position = pos;
}

//============================================================================
//  Angle offset
//============================================================================

tf2::Matrix3x3 Imu::getAngleOffset (void){
    //---------------------------------------------------------------------------
    //  Returns its 'angle_offset' value.
    //---------------------------------------------------------------------------
    return this->angle_offset;
}

void Imu::setAngleOffset (tf2::Matrix3x3 new_offset){
    //---------------------------------------------------------------------------
    //  Sets 'angle_offset' value.
    //---------------------------------------------------------------------------
    this->angle_offset = new_offset;
}
```

Anexo 17. tfg_v2_ImuMsg.h

```
//===================================================================
//
//  File:       tfg_v2_ImuMsg.h
//  Author:     David Turbica
//
//===================================================================

#ifndef __TFG_V2_IMUMSG_H_INCLUDED__
#define __TFG_V2_IMUMSG_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <ostream>
#include <tf2/LinearMath/Vector3.h>
#include <tf2/LinearMath/Quaternion.h>

//ROS includes


//PCL includes


//TFG includes



class ImuMsg{
    //Variables
    private:
        int time;
        float quaternion_x;
        float quaternion_y;
        float quaternion_z;
        float quaternion_w;
        float angular_velocity_x;
        float angular_velocity_y;
        float angular_velocity_z;
        float linear_acceleration_x;
        float linear_acceleration_y;
        float linear_acceleration_z;

    //Functions
    public:
        ImuMsg (void);
        void setTime (int new_time);
        void setQuaternionX (float value);
        void setQuaternionY (float value);
        void setQuaternionZ (float value);
        void setQuaternionW (float value);
        void setAngularVelocityX (float value);
        void setAngularVelocityY (float value);
        void setAngularVelocityZ (float value);
```

```
        void setLinearAccelerationX (float value);
        void setLinearAccelerationY (float value);
        void setLinearAccelerationZ (float value);
        int getTime (void);
        tf2::Quaternion getOrientation (void);
        tf2::Vector3 getLinearAcceleration (void);
        friend std::ostream & operator << (std::ostream &os, const ImuMsg &im){
            os << std::setw(30) << std::left << "ImuMsg:  Time:"                << im.time <<
std::endl;
            os << std::setw(30) << std::left << "         Quaternion:"          << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.quaternion_x
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.quaternion_y
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.quaternion_z
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.quaternion_w << std::endl;
            os << std::setw(30) << std::left << "         Angular velocity:"    << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.angular_velocity_x
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.angular_velocity_y
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.angular_velocity_z << std::endl;
            os << std::setw(30) << std::left << "         Linear acceleration:" << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.linear_acceleration_x
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.linear_acceleration_y
                                                                                << std::setw(14)
<< std::right << std::fixed << std::setprecision(5) << im.linear_acceleration_z;
            return os;
        }
};

#endif //__TFG_V2_IMUMSG_H_INCLUDED
```

Anexo 18. tfg_v2_ImuMsg.cpp

```
//===================================================================
//
//  File:       tfg_v2_ImuMsg.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_ImuMsg.h"

ImuMsg::ImuMsg (void){
    //-----------------------------------------------------------------------------
    //  Default constructor.
    //-----------------------------------------------------------------------------
    this->time = 0;
    this->quaternion_x = 0.0;
    this->quaternion_y = 0.0;
    this->quaternion_z = 0.0;
    this->quaternion_w = 0.0;
    this->angular_velocity_x = 0.0;
    this->angular_velocity_y = 0.0;
    this->angular_velocity_z = 0.0;
    this->linear_acceleration_x = 0.0;
    this->linear_acceleration_y = 0.0;
    this->linear_acceleration_z = 0.0;
}

void ImuMsg::setTime (int new_time){
    //-----------------------------------------------------------------------------
    //  Sets 'time' value
    //-----------------------------------------------------------------------------
    this->time = new_time;
}

void ImuMsg::setQuaternionX (float value){
    //-----------------------------------------------------------------------------
    //  Sets 'quaternion_x' value
    //-----------------------------------------------------------------------------
    this->quaternion_x = value;
}

void ImuMsg::setQuaternionY (float value){
    //-----------------------------------------------------------------------------
    //  Sets 'quaternion_y' value
    //-----------------------------------------------------------------------------
    this->quaternion_y = value;
}

void ImuMsg::setQuaternionZ (float value){
    //-----------------------------------------------------------------------------
    //  Sets 'quaternion_z' value
    //-----------------------------------------------------------------------------
    this->quaternion_z = value;
}
```

```
void ImuMsg::setQuaternionW (float value){
    //------------------------------------------------------------------------------
    //  Sets 'quaternion_w' value
    //------------------------------------------------------------------------------
    this->quaternion_w = value;
}

void ImuMsg::setAngularVelocityX (float value){
    //------------------------------------------------------------------------------
    //  Sets 'angular_velocity_x' value
    //------------------------------------------------------------------------------
    this->angular_velocity_x = value;
}

void ImuMsg::setAngularVelocityY (float value){
    //------------------------------------------------------------------------------
    //  Sets 'angular_velocity_y' value
    //------------------------------------------------------------------------------
    this->angular_velocity_y = value;
}

void ImuMsg::setAngularVelocityZ (float value){
    //------------------------------------------------------------------------------
    //  Sets 'angular_velocity_z' value
    //------------------------------------------------------------------------------
    this->angular_velocity_z = value;
}

void ImuMsg::setLinearAccelerationX (float value){
    //------------------------------------------------------------------------------
    //  Sets 'linear_acceleration_x' value
    //------------------------------------------------------------------------------
    this->linear_acceleration_x = value;
}

void ImuMsg::setLinearAccelerationY (float value){
    //------------------------------------------------------------------------------
    //  Sets 'linear_acceleration_y' value
    //------------------------------------------------------------------------------
    this->linear_acceleration_y = value;
}

void ImuMsg::setLinearAccelerationZ (float value){
    //------------------------------------------------------------------------------
    //  Sets 'linear_acceleration_z' value
    //------------------------------------------------------------------------------
    this->linear_acceleration_z = value;
}

int ImuMsg::getTime (void){
    //------------------------------------------------------------------------------
    //  Returns its 'time' value
    //------------------------------------------------------------------------------
    return this->time;
}

tf2::Quaternion ImuMsg::getOrientation (void){
    //------------------------------------------------------------------------------
    //  Returns its 'quaternion' as a 'tf::Quaternion'
    //------------------------------------------------------------------------------
    tf2::Quaternion orientation (this->quaternion_x,
                                 this->quaternion_y,
                                 this->quaternion_z,
                                 this->quaternion_w);
    return orientation;
```

```
}

tf2::Vector3 ImuMsg::getLinearAcceleration (void){
    //-----------------------------------------------------------------------------
    //  Returns its 'linear_acceleration' as a 'tf::Vector3'
    //-----------------------------------------------------------------------------
    tf2::Vector3 acceleration (this->linear_acceleration_x,
                              this->linear_acceleration_y,
                              this->linear_acceleration_z);
    return acceleration;
}
```

Anexo 19. tfg_v2_Map.h

```
//==================================================================
//
//  File:       tfg_v2_Map.h
//  Author:     David Turbica
//
//==================================================================

#ifndef __TFG_V2_MAP_H_INCLUDED__
#define __TFG_V2_MAP_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <Eigen/Dense>
#include <tf2/LinearMath/Transform.h>

//ROS includes


//PCL includes
#include <pcl/registration/icp.h>
#include <pcl/common/transforms.h>

//TFG includes
#include "tfg_v2_Cloud.h"
#include "tfg_v2_Logger.h"
#include "tfg_v2_utilities.h"

class Map{
    //Variables
    public:
        Cloud cloud;

    //Functions
    public:
        Map (void);
        bool loadMap (std::string path);
        void saveMap (std::string path);
        void downsample (double resolution);
        void removeOutliersStatisticaly (int neighbours = 50, double dev_const = 1.0);
        void removeOutliersRadius (double radius = 0.1, int points = 10);
        void applyTransform (Eigen::Matrix4f transformation_matrix);
        tf2::Transform locate (Cloud frame, tf2::Transform estimation);
        void drawLocation (tf2::Transform loc, bool type);
        friend std::ostream & operator << (std::ostream &os, const Map &m){
            os << "Map:    Cloud:  points[]: " << m.cloud.point_cloud->points.size () <<
std::endl
               << "                width: " << m.cloud.point_cloud->width << std::endl
               << "                height: " << m.cloud.point_cloud->height << std::endl
               << "                is_dense: " << m.cloud.point_cloud->is_dense << std::endl
               << "                sensor origin (xyz): ["
                                    << m.cloud.point_cloud->sensor_origin_.x () << ", "
                                    << m.cloud.point_cloud->sensor_origin_.y () << ", "
```

```
                                    << m.cloud.point_cloud->sensor_origin_.z () << "] /
orientation (xyzw): ["
                                    << m.cloud.point_cloud->sensor_orientation_.x () << ", "
                                    << m.cloud.point_cloud->sensor_orientation_.y () << ", "
                                    << m.cloud.point_cloud->sensor_orientation_.z () << ", "
                                    << m.cloud.point_cloud->sensor_orientation_.w () << "]";
            return os;
        }
};

#endif //__TFG_V2_MAP_H_INCLUDED
```

Anexo 20. tfg_v2_Map.cpp

```
//==================================================================
//
//  File:       tfg_v2_Map.cpp
//  Author:     David Turbica
//
//==================================================================

#include "tfg_v2_Map.h"

Map::Map (void){
    //-------------------------------------------------------------------------------
    //  Default constructor.
    //-------------------------------------------------------------------------------
}

bool Map::loadMap (std::string path){
    //-------------------------------------------------------------------------------
    //  Loads a PointCloud from a .pcd file. Returns true when PointCloud was
    //  loaded successfully, false when it was not.
    //-------------------------------------------------------------------------------
    return this->cloud.loadCloud (path);
}

void Map::saveMap (std::string path){
    //-------------------------------------------------------------------------------
    //  Saves a PointCloud to a .pcd file.
    //-------------------------------------------------------------------------------
    this->cloud.saveCloud (path);
}

void Map::downsample (double resolution) {
    //-------------------------------------------------------------------------------
    //  Downsamples its Cloud to the resolution provided.
    //-------------------------------------------------------------------------------
    this->cloud.downsample (resolution);
}

void Map::removeOutliersStatisticaly (int neighbours, double dev_const){
    //-------------------------------------------------------------------------------
    //  Statistical filter to remove outliers
    //-------------------------------------------------------------------------------
    this->cloud.removeOutliersStatisticaly (neighbours, dev_const);
}

void Map::removeOutliersRadius (double radius, int points){
    //-------------------------------------------------------------------------------
    //  Remove lonely outliers
    //-------------------------------------------------------------------------------
    this->cloud.removeOutliersRadius (radius, points);
}

void Map::applyTransform (Eigen::Matrix4f transformation_matrix){
    //-------------------------------------------------------------------------------
    //  Applies a Transformation to its point_cloud.
```

```
    //--------------------------------------------------------------------------
    this->cloud.applyTransform (transformation_matrix);
}

tf2::Transform Map::locate (Cloud frame, tf2::Transform estimation){
    //--------------------------------------------------------------------------
    //  Locates a Cloud in map. Location estimation is used if 'estimation' is
    //  provided. It returns the location of the cloud in map.
    //--------------------------------------------------------------------------
    //Convert estimation form tf2::Transform to Eigen:Matrix4d
    Eigen::Matrix4f eigen_estimation = TransformToEigenMatrix (estimation);
    //IPC
    int iterations = 50;
    Eigen::Matrix4f transformation_matrix = Eigen::Matrix4f::Identity ();
    pcl::PointCloud<pcl::PointXYZ> output;
    pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
    icp.setMaximumIterations (iterations);
    icp.setTransformationEpsilon (1e-8);
    icp.setEuclideanFitnessEpsilon (1);
    icp.setInputSource (frame.point_cloud);                 //CLOUD
    icp.setInputTarget (this->cloud.point_cloud);           //MAP
    icp.align (output, eigen_estimation);
    if (icp.hasConverged ()){
        std::cout << "\nICP has converged, score is " << icp.getFitnessScore () << std::endl;
        std::cout << "\nICP transformation " << iterations << " : cloud_icp -> cloud_in" <<
std::endl;
        transformation_matrix = icp.getFinalTransformation();
        printEigenMatrix4f (transformation_matrix);
    }else{
        PCL_ERROR ("\nICP has not converged.\n");
    }
    //Convert result transformation from Eigen::Matrix4d into tf2::Transform
    return EigenMatrixToTransform (transformation_matrix);
}

void Map::drawLocation (tf2::Transform loc, bool type){
    this->cloud.drawPoint (loc.getOrigin()[0], loc.getOrigin()[1], loc.getOrigin()[2]);
}
```

Anexo 21. tfg_v2_Process.h

```
//====================================================================
//
//  File:       tfg_v2_Process.h
//  Author:     David Turbica
//
//====================================================================

#ifndef __TFG_V2_PROCESS_H_INCLUDED__
#define __TFG_V2_PROCESS_H_INCLUDED__


//C++ includes


//ROS includes


//PCL includes


//TFG includes
#include "tfg_v2_Timer.h"


namespace tfg {

    class Process{
        //Variables
        public:
            int steps;
            int current_step;
            float average_time;
            double last_time;
            Timer timer;

        //Functions
        public:
            Process (void);
            void create (int n_steps);
            void start (void);
            double stop (void);
            double step (void);
            double lastTime (void);
            int progress (void);
            float timeRemainingEstimation (void);
    };

} //end of namespace tfg

#endif //__TFG_V2_PROCESS_H_INCLUDED
```

Anexo 22. tfg_v2_Process.cpp

```
//======================================================================
//
//  File:       tfg_v2_Process.cpp
//  Author:     David Turbica
//
//======================================================================

#include "tfg_v2_Process.h"

namespace tfg {

    Process::Process (void){
        //------------------------------------------------------------------------------
        //  Default constructor.
        //------------------------------------------------------------------------------
        this->steps = 0;
        this->current_step = 0;
        this->average_time = 0.0;
        this->last_time = 0.0;
    }

    void Process::create (int n_steps){
        //------------------------------------------------------------------------------
        //  Sets 'steps' value.
        //------------------------------------------------------------------------------
        this->steps = n_steps;
        this->current_step = 0;
        this->average_time = 0.0;
        this->last_time = 0.0;
    }

    void Process::start (void){
        //------------------------------------------------------------------------------
        //  Starts meassuring time.
        //------------------------------------------------------------------------------
        this->timer.start();
    }

    double Process::stop (void){
        //------------------------------------------------------------------------------
        //  Returns the time elapsed since 'start()' was called.
        //------------------------------------------------------------------------------
        this->last_time = this->timer.stop();
        return this->last_time;
    }

    double Process::step (void){
        //------------------------------------------------------------------------------
        //  Advances one step. Returns the time elapsed sice last 'start()' was call.
        //  Also recalculates the period time average.
        //------------------------------------------------------------------------------
        double period = this->timer.period ();
        this->last_time = this->timer.step();
        this->current_step ++;
```

```
        this->average_time += ((period - this->average_time) / this->current_step);
        return this->last_time;
    }
    double Process::lastTime (void){
        //------------------------------------------------------------------------------
        //  Returns last_time
        //------------------------------------------------------------------------------
        return this->last_time;
    }

    int Process::progress (void){
        //------------------------------------------------------------------------------
        //  Returns the progress (%) of the process.
        //------------------------------------------------------------------------------
        return (int)((100 * (long)this->current_step) / (long)this->steps);
    }

    float Process::timeRemainingEstimation (void){
        //------------------------------------------------------------------------------
        //  Returns the estimated reminding time for the process to be completed.
        //------------------------------------------------------------------------------
        return (this->steps - this->current_step) * this->average_time;
    }

} //end of namespace tfg
```

Anexo 23. tfg_v2_processor.h

```
//===================================================================
//
//  File:       tfg_v2_processor.h
//  Author:     David Turbica
//
//===================================================================

#ifndef __TFG_V2_PROCESSOR_H_INCLUDED__
#define __TFG_V2_PROCESSOR_H_INCLUDED__

#include <ros/ros.h>
#include <signal.h>

//C++ includes
#include <string>
#include <iostream>
#include <fstream>
#include <limits>
#include <stdlib.h>
#include <Eigen/Dense>
#include <tf2/LinearMath/Vector3.h>
#include <tf2/LinearMath/Matrix3x3.h>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2/LinearMath/Transform.h>

//ROS includes
#include <sensor_msgs/PointCloud2.h>
#include <sensor_msgs/Imu.h>

//PCL includes


//TFG includes
#include "tfg_v2_CloudMsg.h"
#include "tfg_v2_ImuMsg.h"
#include "tfg_v2_Map.h"
#include "tfg_v2_Cloud.h"
#include "tfg_v2_Drone.h"
#include "tfg_v2_Imu.h"
#include "tfg_v2_FilterVector3.h"
#include "tfg_v2_Process.h"
#include "tfg_v2_Timer.h"
#include "tfg_v2_Logger.h"


//Command parser variables
std::string main_path;
std::string sequence;
int filter_size;

//Global Variables
bool sigint_called = false;
bool more_data;
long simulation_time; //in miliseconds
```

```
std::string path_clds;
std::string path_clds_data;
std::string path_imus_data;
std::string path_parameter;
std::string path_map;
std::string path_origin;
std::string path_imu_calibration;
std::string path_imu_calibration_params;
Map map;
Drone drone;
Imu imu;

//Variables CAM
int cld_counter;
bool next_cld;
CloudMsg cld_msg;
int time_last_cloud;

//Variables IMU
int imu_counter;
int imu_counter_calibration;
bool next_imu;
ImuMsg imu_msg;
int filter_counter;
FilterVector3 filter;

//Debugging step-by-step
bool stop_cld = false;
bool stop_imu = true;

//Files
std::ifstream origin_file;
std::ifstream cld_file;
std::ifstream imu_file;
std::ifstream param_file;
std::ifstream calib_file;
std::ifstream calib_params_file;

//Functions
int main (int argc, char** argv);
void customSigintHandler(int sig);
void printHelp (char* program_name);
void pauseSimulation (void);

#endif //__TFG_V2_PROCESSOR_H_INCLUDED
```

Anexo 24. tfg_v2_processor.cpp

```
//===================================================================
//
//  File:       tfg_v2_processor.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_processor.h"


int main (int argc, char** argv) {
    //-------------------------------------------------------------------------------
    //  Main function. It initializes the whole program and sets the timers and
    //  and topics' subscribers and publishers.
    //-------------------------------------------------------------------------------

    //==============================================================================
    //  Default arguments
    //==============================================================================
    main_path = "src/tfg_v2/map";
    sequence = "1";
    filter_size = 20;
    LOG_LEVEL = 1; //Info level
    stop_cld = false;
    stop_imu = false;

    //==============================================================================
    //  Parse arguments
    //==============================================================================
    for (int i=1; i<argc; i++){
        std::string arg (argv[i]);
        if (arg == "-h" || arg == "--help"){
            printHelp(argv[0]);
            return 0;
        }

        else if (arg == "-d" || arg == "--debug"){
            LOG_LEVEL = 0;
        }

        else if (arg == "-i" || arg == "--info"){
            LOG_LEVEL = 1;
        }

        else if (arg == "-e" || arg == "--error"){
            LOG_LEVEL = 2;
        }

        else if (arg == "-n" || arg == "--nolog"){
            LOG_LEVEL = 3;
        }

        else if (arg == "-di" || arg == "--debugimu"){
            LOG_LEVEL = 0;
```

```
        stop_cld = false;
        stop_imu = true;
    }

    else if (arg == "-dc" || arg == "--debugcloud"){
        LOG_LEVEL = 0;
        stop_cld = true;
        stop_imu = false;
    }

    else if (arg == "-db" || arg == "--debugboth"){
        LOG_LEVEL = 0;
        stop_cld = true;
        stop_imu = true;
    }

    else if (arg == "-m" || arg == "--map"){
        i++;
        if (i < argc){
            main_path = std::string (argv[i]);
        }
        else{
            printHelp(argv[0]);
            return 0;
        }
    }

    else if (arg == "-s" || arg == "--seq"){
        i++;
        if (i < argc){
            sequence = std::string (argv[i]);
        }
        else{
            printHelp(argv[0]);
            return 0;
        }
    }

    else if (arg == "-f" || arg == "--filter"){
        i++;
        if (i < argc){
            filter_size = std::atoi (argv[i]);
        }
        else{
            printHelp(argv[0]);
            return 0;
        }
    }

    else{
        printHelp(argv[0]);
        return 0;
    }
}

//=============================================================================
//  Print header
//=============================================================================
PRINT_INFO ("======================================================================");
PRINT_INFO ("|                                                                    |");
PRINT_INFO ("|                          TFG_PROCESSOR                             |");
PRINT_INFO ("|                                                                    |");
PRINT_INFO ("|====================================================================|");
PRINT_INFO ("|                                                                    |");
PRINT_INFO ("| Author:           David Turbica Mamblona                           |");
```

```
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Description:      ICP-based RGBd camera location system on a       |");
    PRINT_INFO ("|                   known environment                                |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("======================================================================");

    //============================================================================
    //  Initialization
    //============================================================================

    //Initialize ROS
    ros::init (argc, argv, "tfg_v2_processor");
    signal(SIGINT, customSigintHandler);

    //Suppresses any PCL's output to the console.
    //pcl::console::setVerbosityLevel(pcl::console::L_ALWAYS);

    //Initialize global variables
    more_data = true;
    simulation_time = 0;
    path_clds = main_path + "/sequence_" + sequence + "/clouds/";
    path_clds_data = main_path + "/sequence_" + sequence + "/clouds_data.txt";
    path_imus_data = main_path + "/sequence_" + sequence + "/imus_data.txt";
    path_parameter = main_path + "/sequence_" + sequence + "/parameters.txt";
    path_map = main_path + "/map.pcd";
    path_origin = main_path + "/sequence_" + sequence + "/origin.txt";
    path_imu_calibration = main_path + "/sequence_" + sequence + "/imu_calibration.txt";
    path_imu_calibration_params = main_path + "/sequence_" + sequence +
"/imu_calibration_params.txt";

    PRINT_INFO ("");
    PRINT_INFO ("----------------------------------------------------------------------");
    PRINT_INFO ("Loading data from the following sources:");
    PRINT_INFO ("    map.pcd                        " << path_map);
    PRINT_INFO ("    origin.txt                     " << path_origin);
    PRINT_INFO ("    clouds/                        " << path_clds);
    PRINT_INFO ("    clouds_data.txt                " << path_clds_data);
    PRINT_INFO ("    imus_data.txt                  " << path_imus_data);
    PRINT_INFO ("    imu_calibration_params.txt     " << path_imu_calibration_params);
    PRINT_INFO ("    imu_calibration.txt            " << path_imu_calibration);
    PRINT_INFO ("    parameters.txt                 " << path_parameter);
    PRINT_INFO ("----------------------------------------------------------------------");
    PRINT_INFO ("Checking files:");

    //============================================================================
    //  Load map from 'map.pcd' file
    //============================================================================
    if (map.loadMap (path_map)){
        PRINT_INFO ("    map.pcd                        LOADED");
        //Downsample
        map.downsample(0.1);
        //Remove outliers
        map.removeOutliersStatisticaly (50, 3);
        map.removeOutliersRadius (0.06, 4);
    }else{
        PRINT_ERROR ("    map.pcd                        ERROR");
        PRINT_INFO ("----------------------------------------------------------------------");
        PRINT_INFO ("");
        PRINT_ERROR ("Unable to load map from '" << path_map << "'");
        PRINT_INFO ("");
        PRINT_INFO ("======================================================================");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|                          TFG_PROCESSOR                             |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|====================================================================|");
```

```
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|  Author:           David Turbica Mamblona                          |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|  Description:      ICP-based RGBd camera location system on a      |");
        PRINT_INFO ("|                    known environment                               |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|====================================================================|");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|  Node terminated:  No map to work on.                              |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("======================================================================");
        ros::shutdown();
        return 0;
    }

    //==============================================================================
    //  Parse 'origin.txt' file
    //==============================================================================
    origin_file.open (path_origin.c_str());
    if (origin_file.is_open()){
        PRINT_INFO ("    origin.txt                      LOADED");
        tf2::Matrix3x3 rotation;
        tf2::Vector3 position;
        //Parse the file
        float cell;
        origin_file >> cell;
        rotation[0][0] = cell;
        origin_file >> cell;
        rotation[0][1] = cell;
        origin_file >> cell;
        rotation[0][2] = cell;
        origin_file >> cell;
        position[0] = cell;
        origin_file >> cell;
        rotation[1][0] = cell;
        origin_file >> cell;
        rotation[1][1] = cell;
        origin_file >> cell;
        rotation[1][2] = cell;
        origin_file >> cell;
        position[1] = cell;
        origin_file >> cell;
        rotation[2][0] = cell;
        origin_file >> cell;
        rotation[2][1] = cell;
        origin_file >> cell;
        rotation[2][2] = cell;
        origin_file >> cell;
        position[2] = cell;
        //Close the file
        origin_file.close();
        //Initialize DRONE location
        tf2::Transform origin_location (rotation, position);
        drone.setLocation (origin_location);
        //Initialize IMU position
        imu.reposition (position);
    }else{
        PRINT_ERROR ("    origin.txt                      ERROR");
        PRINT_INFO ("----------------------------------------------------------------------");
        PRINT_INFO ("");
        PRINT_ERROR ("Unable to load origin location from '" << path_origin << "'");
        PRINT_INFO ("");
        PRINT_INFO ("======================================================================");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|                          TFG_PROCESSOR                             |");
```

```
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Author:            David Turbica Mamblona                          |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Description:       ICP-based RGBd camera location system on a      |");
    PRINT_INFO ("|                    known environment                               |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Node terminated:   No origin location to start with                |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("======================================================================");
    ros::shutdown();
    return 0;
}

//============================================================================
//  Initialize CAM variables
//============================================================================
cld_file.open (path_clds_data.c_str());
cld_counter = 0;
next_cld = true;
time_last_cloud = 0;

//============================================================================
//  Initialize IMU variables
//============================================================================
imu_file.open (path_imus_data.c_str());
imu_counter = 0;
next_imu = true;
filter_counter = 0;
filter = FilterVector3 (filter_size);
filter.avoidStartAttenuance ();

//============================================================================
//  Checking files 'cld_file' and 'imu_file'
//============================================================================
if (cld_file.is_open()){
    PRINT_INFO ("    clouds_data.txt              LOADED");
}else{
    PRINT_ERROR ("    clouds_data.txt              ERROR");
    PRINT_INFO ("----------------------------------------------------------------------");
    PRINT_INFO ("");
    PRINT_ERROR ("Unable to load cloud info from '" << path_clds_data << "'");
    PRINT_INFO ("");
    PRINT_INFO ("======================================================================");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|                           TFG_PROCESSOR                            |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Author:            David Turbica Mamblona                          |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Description:       ICP-based RGBd camera location system on a      |");
    PRINT_INFO ("|                    known environment                               |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Node terminated:   No cloud data to work with.                     |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("======================================================================");
    ros::shutdown();
    return 0;
}
```

```
    if (imu_file.is_open()){
        PRINT_INFO ("    imus_data.txt                LOADED");
    }else{
        PRINT_ERROR ("    imus_data.txt                ERROR");
        PRINT_INFO ("---------------------------------------------------------------------");
        PRINT_INFO ("");
        PRINT_ERROR ("Unable to load imus info from '" << path_imus_data << "'");
        PRINT_INFO ("");
        PRINT_INFO ("=====================================================================");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|                         TFG_PROCESSOR                             |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|===================================================================|");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("| Author:           David Turbica Mamblona                          |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("| Description:      ICP-based RGBd camera location system on a      |");
        PRINT_INFO ("|                   known environment                               |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|===================================================================|");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("| Node terminated:  No imu data to work with.                       |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("=====================================================================");
        ros::shutdown();
        return 0;
    }

    //============================================================================
    //  Parse 'calib_params.txt' file
    //============================================================================
    calib_params_file.open (path_imu_calibration_params.c_str());
    if (calib_params_file.is_open()){
        PRINT_INFO ("    imu_calibration_params.txt   LOADED");
        std::string tokem;
        //Getting number of imus to process
        std::getline (calib_params_file, tokem, ' ');
        std::getline (calib_params_file, tokem, '\n');
        imu_counter = std::atoi (tokem.c_str());
        imu.setNumberCalibrationSamples (imu_counter);
        //Set this variable so we can keep its value for debugging
        imu_counter_calibration = imu_counter;
        //Closing file
        calib_params_file.close();
    }else{
        PRINT_ERROR ("    imu_calibration_params.txt   ERROR");
        PRINT_INFO ("---------------------------------------------------------------------");
        PRINT_INFO ("");
        PRINT_ERROR ("Unable to load imu calibration parameters from '" <<
path_imu_calibration_params << "'");
        PRINT_INFO ("");
        PRINT_INFO ("=====================================================================");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|                         TFG_PROCESSOR                             |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|===================================================================|");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("| Author:           David Turbica Mamblona                          |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("| Description:      ICP-based RGBd camera location system on a      |");
        PRINT_INFO ("|                   known environment                               |");
        PRINT_INFO ("|                                                                   |");
        PRINT_INFO ("|===================================================================|");
        PRINT_INFO ("|                                                                   |");
```

```
        PRINT_INFO ("| Node terminated:  No calibration parameters for the IMU.           |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("=====================================================================");
        ros::shutdown();
        return 0;
    }

    //============================================================================
    //  Parse 'calibration.txt' file
    //============================================================================
    calib_file.open (path_imu_calibration.c_str());
    if (calib_file.is_open()){
        PRINT_INFO ("    imu_calibration.txt          LOADED");
        FilterVector3 filter_calib (filter_size);
        //filter_calib.avoidStartAttenuance();
        //Progress updater variables
        bool p01 = false;
        bool p25 = false;
        bool p50 = false;
        bool p75 = false;
        while (imu_counter > 0){
            std::string tokem;
            //Getting name
            std::getline (calib_file, tokem, ' ');
            //Getting time
            std::getline (calib_file, tokem, ' ');
            imu_msg.setTime                  (atof (tokem.c_str()));
            //Getting quaternion x
            std::getline (calib_file, tokem, ' ');
            imu_msg.setQuaternionX           (atof (tokem.c_str()));
            //Getting quaternion y
            std::getline (calib_file, tokem, ' ');
            imu_msg.setQuaternionY           (atof (tokem.c_str()));
            //Getting quaternion z
            std::getline (calib_file, tokem, ' ');
            imu_msg.setQuaternionZ           (atof (tokem.c_str()));
            //Getting quaternion w
            std::getline (calib_file, tokem, ' ');
            imu_msg.setQuaternionW           (atof (tokem.c_str()));
            //Getting angular velocity x
            std::getline (calib_file, tokem, ' ');
            imu_msg.setAngularVelocityX      (atof (tokem.c_str()));
            //Getting angular velocity y
            std::getline (calib_file, tokem, ' ');
            imu_msg.setAngularVelocityY      (atof (tokem.c_str()));
            //Getting angular velocity z
            std::getline (calib_file, tokem, ' ');
            imu_msg.setAngularVelocityZ      (atof (tokem.c_str()));
            //Getting linear acceleration x
            std::getline (calib_file, tokem, ' ');
            imu_msg.setLinearAccelerationX   (atof (tokem.c_str()));
            //Getting linear acceleration y
            std::getline (calib_file, tokem, ' ');
            imu_msg.setLinearAccelerationY   (atof (tokem.c_str()));
            //Getting linear acceleration z
            std::getline (calib_file, tokem, '\n');
            imu_msg.setLinearAccelerationZ   (atof (tokem.c_str()));
            //Filtering imu data
            tf2::Quaternion orientation_quaternion = imu_msg.getOrientation ();
            tf2::Transform transform_matrix (orientation_quaternion);
            tf2::Vector3 relative_acceleration = imu_msg.getLinearAcceleration ();
            tf2::Vector3 fixed_acceleration = transform_matrix * relative_acceleration;
            filter_calib.addMeasure(fixed_acceleration);
            tf2::Vector3 fixed_filtered_acceleration = filter_calib.calcMean();
            //Adding calibration data
```

```
        int progress = imu.calibrate (fixed_filtered_acceleration);
        //Print progress
        if (!p01){
            if (progress > 1){
                PRINT_INFO ("        Calibrating IMU 0%");
                p01 = true;
            }
        }
        else if (!p25){
            if (progress > 25){
                PRINT_INFO ("        Calibrating IMU 25%");
                p25 = true;
            }
        }
        else if (!p50){
            if (progress > 50){
                PRINT_INFO ("        Calibrating IMU 50%");
                p50 = true;
            }
        }
        else if (!p75){
            if (progress > 75){
                PRINT_INFO ("        Calibrating IMU 75%");
                p75 = true;
            }
        }
        else{
            if (progress == 100){
                PRINT_INFO ("        Calibrating IMU 100%");
            }
        }
        //Read next line
        imu_counter --;
    }
    //Closing file
    calib_file.close();
}else{
    PRINT_ERROR ("    imu_calibration.txt          ERROR");
    PRINT_INFO ("----------------------------------------------------------------------");
    PRINT_INFO ("");
    PRINT_ERROR ("Unable to load imu calibration from '" << path_imu_calibration << "'");
    PRINT_INFO ("");
    PRINT_INFO ("======================================================================");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|                          TFG_PROCESSOR                             |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Author:           David Turbica Mamblona                           |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Description:      ICP-based RGBd camera location system on a       |");
    PRINT_INFO ("|                   known environment                                |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Node terminated:  No calibration data for the IMU.                 |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("======================================================================");
    ros::shutdown();
    return 0;
}

//============================================================================
//  Parse 'parameters.txt' file
//============================================================================
```

```
param_file.open (path_parameter.c_str());
if (param_file.is_open()){
    PRINT_INFO ("    parameters.txt                LOADED");
    PRINT_INFO ("----------------------------------------------------------------------");
    std::string counter_str;
    //Getting number of clouds to process
    std::getline (param_file, counter_str, ' ');
    std::getline (param_file, counter_str, '\n');
    cld_counter = std::atoi (counter_str.c_str());
    //Getting number of imus to process
    std::getline (param_file, counter_str, ' ');
    std::getline (param_file, counter_str, '\n');
    imu_counter = std::atoi (counter_str.c_str());
    //Closing file
    param_file.close();
}else{
    PRINT_ERROR ("    parameters.txt               ERROR");
    PRINT_INFO ("----------------------------------------------------------------------");
    PRINT_INFO ("");
    PRINT_ERROR ("File 'parameters.txt' not found");
    PRINT_INFO ("");
    PRINT_INFO ("======================================================================");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|                           TFG_PROCESSOR                            |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Author:           David Turbica Mamblona                           |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Description:      ICP-based RGBd camera location system on a       |");
    PRINT_INFO ("|                   known environment                                |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("|====================================================================|");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("| Node terminated:  No parameters for PointCloud and Imu to work     |");
    PRINT_INFO ("|                   with.                                            |");
    PRINT_INFO ("|                                                                    |");
    PRINT_INFO ("======================================================================");
    ros::shutdown();
    return 0;
}

//=============================================================================
//  Print some debug info
//=============================================================================
PRINT_DEBUG (map);
PRINT_DEBUG ("----------------------------------------------------------------------");
PRINT_DEBUG (drone);
PRINT_DEBUG ("----------------------------------------------------------------------");
PRINT_DEBUG ("Measures used in IMU calibration:  " << imu_counter_calibration);
PRINT_DEBUG ("----------------------------------------------------------------------");
PRINT_DEBUG (imu);
PRINT_DEBUG ("----------------------------------------------------------------------");
PRINT_DEBUG ("Number of clouds to process:  " << cld_counter);
PRINT_DEBUG ("----------------------------------------------------------------------");
PRINT_DEBUG ("Number of imus to process:    " << imu_counter);
PRINT_DEBUG ("----------------------------------------------------------------------");

//=============================================================================
//  Simulation progress control
//=============================================================================
PRINT_INFO ("Processing sequence:");
PRINT_INFO ("      Progress    Sim.Time    Pro.Time    Time.Remaining");
tfg::Timer cld_timer;
tfg::Timer imu_timer;
```

```
tfg::Process process;
process.create (cld_counter + imu_counter);
process.start();

//==========================================================================
//  Main bucle
//==========================================================================
while (ros::ok() && more_data){

    //--------------------------------------------------------------------------
    //  Parse one line from 'clouds_data.txt' file
    //--------------------------------------------------------------------------
    if (next_cld){
        if (cld_counter > 0){
            if (cld_file.is_open()){
                std::string tokem;
                //Getting path
                std::getline (cld_file, tokem, ' ');
                cld_msg.setPath (tokem);
                //Getting time
                std::getline (cld_file, tokem, '\n');
                cld_msg.setTime (std::atoi (tokem.c_str()));
                //Prevent from loading next data
                next_cld = false;
            }else{
                PRINT_ERROR ("File 'clouds_data.txt' lost");
                more_data = false;
            }
        }
        else{
            next_cld = false;
        }
    }

    //--------------------------------------------------------------------------
    //  Parse one line from 'imus_data.txt' file
    //--------------------------------------------------------------------------
    if (next_imu){
        if (imu_counter > 0){
            if (imu_file.is_open()){
                std::string tokem;
                //Getting name
                std::getline (imu_file, tokem, ' ');
                //Getting time
                std::getline (imu_file, tokem, ' ');
                imu_msg.setTime                   (atof (tokem.c_str()));
                //Getting quaternion x
                std::getline (imu_file, tokem, ' ');
                imu_msg.setQuaternionX            (atof (tokem.c_str()));
                //Getting quaternion y
                std::getline (imu_file, tokem, ' ');
                imu_msg.setQuaternionY            (atof (tokem.c_str()));
                //Getting quaternion z
                std::getline (imu_file, tokem, ' ');
                imu_msg.setQuaternionZ            (atof (tokem.c_str()));
                //Getting quaternion w
                std::getline (imu_file, tokem, ' ');
                imu_msg.setQuaternionW            (atof (tokem.c_str()));
                //Getting angular velocity x
                std::getline (imu_file, tokem, ' ');
                imu_msg.setAngularVelocityX       (atof (tokem.c_str()));
                //Getting angular velocity y
                std::getline (imu_file, tokem, ' ');
                imu_msg.setAngularVelocityY       (atof (tokem.c_str()));
                //Getting angular velocity z
```

```
            std::getline (imu_file, tokem, ' ');
            imu_msg.setAngularVelocityZ     (atof (tokem.c_str()));
            //Getting linear acceleration x
            std::getline (imu_file, tokem, ' ');
            imu_msg.setLinearAccelerationX  (atof (tokem.c_str()));
            //Getting linear acceleration y
            std::getline (imu_file, tokem, ' ');
            imu_msg.setLinearAccelerationY  (atof (tokem.c_str()));
            //Getting linear acceleration z
            std::getline (imu_file, tokem, '\n');
            imu_msg.setLinearAccelerationZ  (atof (tokem.c_str()));
            //Prevent from loading next data
            next_imu = false;
        }else{
            PRINT_ERROR ("File 'imus_data.txt' lost");
            more_data = false;
        }
    }
    else{
        next_imu = false;
    }
}

//------------------------------------------------------------------------------
//  Imu processing
//------------------------------------------------------------------------------
if (imu_msg.getTime() == simulation_time){
    //Start time measurement
    imu_timer.start();
    //Read IMU data
    tf2::Matrix3x3 fix_R_imu (imu_msg.getOrientation());
    PRINT_DEBUG (PRINT_MATRIX(fix_R_imu));
    tf2::Vector3 acceleration = imu_msg.getLinearAcceleration();
    PRINT_DEBUG ("Original: " << PRINT_VECTOR3(acceleration));
    //Filter acceleration in IMU relative axes
    filter.addMeasure (acceleration);
    acceleration = filter.calcMean();
    PRINT_DEBUG ("Filtered: " << PRINT_VECTOR3(acceleration));
    //Remove gravity effect in IMU fixed axes
    tf2::Vector3 A_imu = imu.removeOffsets (fix_R_imu, acceleration);
    PRINT_DEBUG ("Offseted: " << PRINT_VECTOR3(A_imu));
    //Get acceleration in MAP reference
    tf2::Matrix3x3 map_R_ini = drone.getLocation().getBasis();
    tf2::Matrix3x3 fix_R_ini = imu.getAngleOffset();
    tf2::Matrix3x3 ini_R_fix = fix_R_ini.inverse();
    tf2::Matrix3x3 map_R_imu = map_R_ini * ini_R_fix * fix_R_imu;
    tf2::Vector3 A_map = map_R_imu * A_imu;
    PRINT_DEBUG ("Map_ref:  " << PRINT_VECTOR3(A_imu));
    //Integrate acceleration in MAP reference
    imu.update (A_map, imu_msg.getTime());
    PRINT_DEBUG ("Imu_pos:  " << PRINT_VECTOR3(imu.getPosition()));
    //Update DRONE location
    tf2::Transform location (map_R_imu, imu.getPosition());
    drone.setLocationEstimation (location);
    //End time measurement
    imu_timer.stop();
    //Ask for a new imu message
    next_imu = true;
    imu_counter --;
    process.step();
    //Pause simulation
    if (LOG_LEVEL == 0 && stop_imu){
        //Debug info
        PRINT_DEBUG (imu_msg);
        PRINT_DEBUG ("----------------------------------------------------------------");
```

```
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');
            }
        }

        //------------------------------------------------------------------------------
        //  Cloud processing
        //------------------------------------------------------------------------------
        if (cld_msg.getTime() == simulation_time){
            //Processing it as if it was a real time PointCloud message
            Cloud frame;
            if (frame.loadCloud (path_clds + cld_msg.getPath())){
                PRINT_DEBUG ("Cloud '" << cld_msg.getPath() << "' loaded succesfully");
                //Start time measurement
                cld_timer.start();
                //Downsample cloud
                frame.downsample(0.1);
                //Remove outliers
                frame.removeOutliersStatisticaly (50, 3);
                frame.removeOutliersRadius (0.06, 4);
                //Transform
                frame.applyTransform (eulerTransform (0.0, M_PI/2.0, -M_PI/2.0, 0.0, 0.0, 0.0));
                //Locate cloud in map
                tf2::Transform new_location = map.locate (frame, drone.getLocationEstimation());
                drone.setLocation (new_location);
                //Calculate velocity from MAP movement
                tf2::Vector3 velocity = drone.getVelocityEstimation(cld_msg.getTime() -
time_last_cloud);
                time_last_cloud = cld_msg.getTime();
                //Reposition imu for the next section
                imu.reposition (new_location.getOrigin(), velocity);
                imu.setAngleOffset (new_location.getBasis());
                //End time measurement
                cld_timer.stop();
            }
            else{
                PRINT_ERROR ("Unable to load '" << cld_msg.getPath() << "'");
            }
            //Ask for a new PointCloud message
            next_cld = true;
            cld_counter --;
            process.step();
            //Pause simulation
            if (LOG_LEVEL == 0 && stop_cld){
                //Debug info
                PRINT_DEBUG (cld_msg);
                PRINT_DEBUG (PRINT_TRANSFORM (drone.getLocationEstimation()));
                PRINT_DEBUG ("----------------------------------------------------------------");
                //Wait user
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');
            }

        }

        //------------------------------------------------------------------------------
        //  Print simulation progress
        //------------------------------------------------------------------------------
        if (next_cld || cld_counter == 0 && imu_counter == 0){
            //Simulation progress is only displayed every time a PointCloud is processed
            //and also when all data has been processed (100%)
            PRINT_INFO ("        [" << std::setfill(' ') << std::setw(3) << std::right <<
process.progress()
                        << "%]" << std::setw(10) << std::right << std::fixed <<
std::setprecision(3)
                        << (float)simulation_time/1000 << " s" << std::setw(10) << std::right <<
std::fixed
```

```
                        << std::setprecision(3) << process.lastTime() << " s"  << std::setw(16)
<< std::right
                        << std::fixed << std::setprecision(3) <<
process.timeRemainingEstimation() << " s" );
        }

        //----------------------------------------------------------------------------
        //  Stops the simulation when data ends
        //----------------------------------------------------------------------------

        if (cld_counter == 0 && imu_counter ==0){
            more_data = false;
        }

        simulation_time ++;
    }

    //============================================================================
    //  Saving results
    //============================================================================
    map.saveMap (main_path + "/sequence_" + sequence + "result.pcd");

    //============================================================================
    //  The processing has successfully ended. Shutting down the node
    //============================================================================
    if (!sigint_called){
        PRINT_INFO ("");
        PRINT_INFO ("======================================================================");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|                          TFG_PROCESSOR                             |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|====================================================================|");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("| Author:            David Turbica Mamblona                          |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("| Description:       ICP-based RGBd camera location system on a      |");
        PRINT_INFO ("|                    known environment                               |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("|====================================================================|");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("| Node terminated:   The processing has successfully ended.          |");
        PRINT_INFO ("|                                                                    |");
        PRINT_INFO ("======================================================================");
    }
    //Shutting down the node
    ros::shutdown();
    return 0;
}

void customSigintHandler(int sig){
    //--------------------------------------------------------------------------
    //  Function that overrides default ROS sigint handler. It allows to print
    //  some information when the node has been shuted down and end child
    //  processes if needed.
    //--------------------------------------------------------------------------
    //Close files
    if (origin_file.is_open())  origin_file.close();
    if (cld_file.is_open())     cld_file.close();
    if (imu_file.is_open())     imu_file.close();
    //Stop debugging stpe-by-step
    stop_cld = false;
    stop_imu = false;
    //Advice a sigint has been called
    sigint_called = true;
    //Print end message
```

```
    PRINT_INFO ("");
    PRINT_INFO ("=======================================================================");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("|                           TFG_PROCESSOR                             |");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("|=====================================================================|");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("| Author:          David Turbica Mamblona                             |");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("| Description:     ICP-based RGBd camera location system on a         |");
    PRINT_INFO ("|                  known environment                                  |");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("|=====================================================================|");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("| Node terminated:  No reason given (Ctrl+c)                          |");
    PRINT_INFO ("|                                                                     |");
    PRINT_INFO ("=======================================================================");
    //Overrides ROS default Sigint Handler
    ros::shutdown();
}

void printHelp (char* program_name){
    std::cout << std::endl;
    std::cout << "Usage: " << program_name << " <option(s)>\n"
              << "Options:\n"
              << "        -h,--help                        Show help\n"
              << "        -d,--debug                       Set Verbose level to DEBUG\n"
              << "        -i,--info                        Set Verbose level to INFO\n"
              << "        -e,--error                       Set Verbose level to ERROR\n"
              << "        -di,--debugimu                   DEBUG + step-by-step IMU\n"
              << "        -dc,--debugcloud                 DEBUG + step-by-step CLOUD\n"
              << "        -db,--debugboth                  DEBUG + step-by-step IMU + CLOUD\n"
              << "        -n,--nolog                       Set Verbose level to NOLOG\n"
              << "        -m <path>,--map <path>           Set the path to a custom map\n"
              << "        -s <number>, --seq <number>      Sets the sequence to be processed\n"
              << "        -f <number>, --filter <number>   Sets the size of the imu filter\n"
              << "Default settings:\n"
              << "        -Verbose level:                  DEBUG\n"
              << "        -Step-by-step:                   No IMU, no CLOUD\n"
              << "        -Map:                            src/tfg_v2/map/\n"
              << "        -Sequence:                       1 (sequence_1)\n"
              << "        -Filter size:                    20 (max 50)\n" << std::endl;
}
```

Anexo 25. tfg_v2_Timer.h

```
//=================================================================
//
//  File:       tfg_v2_Timer.h
//  Author:     David Turbica
//
//=================================================================

#ifndef __TFG_V2_TIMER_H_INCLUDED__
#define __TFG_V2_TIMER_H_INCLUDED__


//C++ includes
#include <ctime>

//ROS includes


//PCL includes


//TFG includes


namespace tfg {

    class Timer{
        //Variables
        private:
            double time;
            clock_t ticks;
            clock_t ticks_period;
            bool started;

        //Functions
        public:
            Timer (void);
            double totalTime (void);
            void start (void);
            double step (void);
            double period (void);
            double stop (void);
    };

} //end of namespace tfg

#endif //__TFG_V2_TIMER_H_INCLUDED
```

Anexo 26. tfg_v2_Timer.cpp

```
//===================================================================
//
//  File:       tfg_v2_Timer.cpp
//  Author:     David Turbica
//
//===================================================================

#include "tfg_v2_Timer.h"

namespace tfg {

    Timer::Timer (void){
        //------------------------------------------------------------------------
        //  Default constructor.
        //------------------------------------------------------------------------
        this->time = 0.0;
        this->started = false;
    }

    double Timer::totalTime (void){
        //------------------------------------------------------------------------
        //  Returns its 'time' value.
        //------------------------------------------------------------------------
        return this->time;
    }

    void Timer::start (void){
        //------------------------------------------------------------------------
        //  Starts meassuring time.
        //------------------------------------------------------------------------
        this->started = true;
        this->ticks = clock();
        this->ticks_period = this->ticks;
    }

    double Timer::step (void){
        //------------------------------------------------------------------------
        //  Returns the time elapsed since last 'start()' was called. Time is not
        //  added to the total time.
        //------------------------------------------------------------------------
        double time_elapsed = 0;
        if (this->started){
            time_elapsed = (double)(clock()-this->ticks) / CLOCKS_PER_SEC;
            this->ticks_period = clock();
        }
        return time_elapsed;
    }

    double Timer::period (void){
        //------------------------------------------------------------------------
        //  Returns the time elapsed since last 'start()' or 'step()' or 'period()'
        //  was called. No time is added to total time.
        //------------------------------------------------------------------------
        double time_elapsed = 0;
```

```
        if (this->started){
            time_elapsed = (double)(clock()-this->ticks_period) / CLOCKS_PER_SEC;
            this->ticks_period = clock();
        }
        return time_elapsed;
    }

    double Timer::stop (void){
        //-----------------------------------------------------------------------------
        //  Returns the time elapsed since last 'start()' was called and adds it to
        //  the total time.
        //-----------------------------------------------------------------------------
        double time_elapsed = 0;
        if (this->started){
            time_elapsed = (double)(clock()-this->ticks) / CLOCKS_PER_SEC;
            this->time += time_elapsed;
            this->started = false;
        }
        return time_elapsed;
    }

} //end of namespace tfg
```

Anexo 27. tfg_v2_utilities.h

```
//===================================================================
//
//  File:       tfg_v2_utilities.h
//  Author:     David Turbica
//
//===================================================================

#ifndef __TFG_V2_UTILITIES_H_INCLUDED__
#define __TFG_V2_UTILITIES_H_INCLUDED__

#include <ros/ros.h>

//C++ includes
#include <string>
#include <math.h>
#include <Eigen/Dense>
#include <tf2/LinearMath/Vector3.h>
#include <tf2/LinearMath/Matrix3x3.h>
#include <tf2/LinearMath/Transform.h>

//ROS includes


//PCL includes
#include <pcl/common/transforms.h>

//TFG includes
#include "tfg_v2_Logger.h"

//Macros
#define PRINT_VECTOR3(vector3) vector3.getX() << "   " << vector3.getY() << "   " <<
vector3.getZ()

#define PRINT_TRANSFORM(transform) std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[0][0]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[0][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[0][2]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getOrigin()[0]\
                                << std::endl\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[1][0]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[1][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[1][2]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getOrigin()[1]\
                                << std::endl\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[2][0]\
```

```
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[2][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getBasis()[2][2]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << transform.getOrigin()[2]\
                                << std::endl\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << 0.0\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << 0.0\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << 0.0\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << 1.0

#define PRINT_MATRIX(matrix)       std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[0][0]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[0][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[0][2]\
                                << std::endl\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[1][0]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[1][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[1][2]\
                                << std::endl\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[2][0]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[2][1]\
                                << std::setw(14) << std::right << std::fixed <<
std::setprecision(5) << matrix[2][2]


double round (double number, int decimals);
tf2::Vector3 roundVector3 (tf2::Vector3 vector, int decimals);
void printEigenMatrix4f (const Eigen::Matrix4f & matrix);
Eigen::Matrix4f TransformToEigenMatrix (tf2::Transform input);
tf2::Transform EigenMatrixToTransform (Eigen::Matrix4f input);
Eigen::Matrix4f eulerTransform(float a, float b, float c, float x, float y, float z);


#endif //__TFG_V2_UTILITIES_H_INCLUDED
```

Anexo 28. tfg_v2_utilities.cpp

```
//=================================================================
//
//  File:       tfg_v2_utilities.cpp
//  Author:     David Turbica
//
//=================================================================

#include "tfg_v2_utilities.h"

double round (double number, int decimals){
    double factor = pow(10.0,(double)decimals);
    return (double)((int)(number*factor))/factor;
}

tf2::Vector3 roundVector3 (tf2::Vector3 vector, int decimals){
    return tf2::Vector3 (round(vector.getX(),decimals), round(vector.getY(),decimals),
round(vector.getZ(),decimals));
}

void printEigenMatrix4f (const Eigen::Matrix4f & matrix){
    printf ("Rotation matrix :\n");
    printf ("    | %6.3f %6.3f %6.3f | \n", matrix (0, 0), matrix (0, 1), matrix (0, 2));
    printf ("R = | %6.3f %6.3f %6.3f | \n", matrix (1, 0), matrix (1, 1), matrix (1, 2));
    printf ("    | %6.3f %6.3f %6.3f | \n", matrix (2, 0), matrix (2, 1), matrix (2, 2));
    printf ("Translation vector :\n");
    printf ("t = < %6.3f, %6.3f, %6.3f >\n\n", matrix (0, 3), matrix (1, 3), matrix (2, 3));
}

Eigen::Matrix4f TransformToEigenMatrix (tf2::Transform input){
    Eigen::Matrix4f output;
    output(0,0) = input.getBasis()[0][0];
    output(0,1) = input.getBasis()[0][1];
    output(0,2) = input.getBasis()[0][2];
    output(0,3) = input.getOrigin()[0];
    output(1,0) = input.getBasis()[1][0];
    output(1,1) = input.getBasis()[1][1];
    output(1,2) = input.getBasis()[1][2];
    output(1,3) = input.getOrigin()[1];
    output(2,0) = input.getBasis()[2][0];
    output(2,1) = input.getBasis()[2][1];
    output(2,2) = input.getBasis()[2][2];
    output(2,3) = input.getOrigin()[2];
    output(3,0) = 0.0;
    output(3,1) = 0.0;
    output(3,2) = 0.0;
    output(3,3) = 1.1;
    return output;
}

tf2::Transform EigenMatrixToTransform (Eigen::Matrix4f input){
    tf2::Matrix3x3 rotation (input(0,0), input(0,1), input(0,2),
                             input(1,0), input(1,1), input(1,2),
                             input(2,0), input(2,1), input(2,2));
    tf2::Vector3 position (input(0,3), input(1,3), input(2,3));
```

```
    return tf2::Transform (rotation, position);
}

Eigen::Matrix4f eulerTransform(float a, float b, float c, float x, float y, float z){
    Eigen::Matrix4f transformation_matrix = Eigen::Matrix4f::Identity ();
    // A rotation matrix
    transformation_matrix (0, 0) = cos(a)*cos(b)*cos(c)-sin(a)*sin(c);
    transformation_matrix (0, 1) = -cos(a)*cos(b)*sin(c)-sin(a)*cos(c);
    transformation_matrix (0, 2) = cos(a)*sin(b);
    transformation_matrix (1, 0) = sin(a)*cos(b)*cos(c)+cos(a)*sin(c);
    transformation_matrix (1, 1) = -sin(a)*cos(b)*sin(c)+cos(a)*cos(c);
    transformation_matrix (1, 2) = sin(a)*cos(b);
    transformation_matrix (2, 0) = -sin(b)*cos(c);
    transformation_matrix (2, 1) = sin(b)*sin(c);
    transformation_matrix (2, 2) = cos(b);
    // Translation on X,Y,Z axis (in meters)
    transformation_matrix (0, 3) = 0.0;
    transformation_matrix (1, 3) = 0.0;
    transformation_matrix (2, 3) = 0.0;
    return transformation_matrix;
}
```