

UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO



ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS  
COMPLEJOS MEDIANTE MÚLTIPLES SENSORES

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y  
AUTOMÁTICA

Autor: Alejandro Areta Díaz

Tutor: María José Gómez Silva



## Índice

<b>ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES.....</b>	<b>1</b>
<b>ÍNDICE.....</b>	<b>2</b>
LISTADO DE FIGURAS.....	5
LISTADO DE TABLAS.....	6
LISTADO DE ECUACIONES .....	7
AGRADECIMIENTOS.....	8
RESUMEN .....	9
ABSTRACT .....	10
<b>1. INTRODUCCIÓN .....</b>	<b>11</b>
1.1. OBJETIVOS .....	12
1.2. ESTRUCTURA DE LA TESIS .....	12
<b>2. ESTADO DEL ARTE .....</b>	<b>13</b>
2.1. MÉTODOS DE DETECCIÓN DE CAMBIOS .....	14
2.2. MODELADO 2D Y 3D DEL ENTORNO .....	15
2.3. TECNOLOGÍAS DE ADQUISICIÓN DE DATOS 3D .....	16
2.4. SISTEMAS DE MEDICIÓN DE DISTANCIAS MEDIANTE IMÁGENES .....	17
<b>3. DESCRIPCIÓN GENERAL .....</b>	<b>19</b>
3.1. HARDWARE.....	19
3.1.1. SENSOR KINECT .....	19
3.1.1.1. DATOS TÉCNICOS MICROSOFT KINECT .....	20
3.1.1.2. ESPECIFICACIONES SENSOR KINECT .....	21
3.1.1.3. MICROSOFT KINECT PARA WINDOWS COMO SENSOR DE VIGILANCIA .....	22
3.2. SOFTWARE.....	23
3.2.1. INTRODUCCIÓN .....	23
3.2.2. SDK KINECT.....	23
3.2.2.1. ARQUITECTURA SDK KINECT.....	24
3.2.2.2. NUI (NATURAL USER INTERFACE) .....	25
3.2.2.3. POSIBLES APLICACIONES DE LA SDK .....	25
3.2.3. PCL .....	26
3.2.3.1. TIPOS DE DATOS .....	27
3.2.4. LENGUAJE Y ENTORNO DE PROGRAMACIÓN .....	27
3.2.4.1. C++ .....	27
3.2.4.2. MICROSOFT VISUAL STUDIO.....	29
3.3. MÓDULOS DE LA APLICACIÓN .....	29
<b>4. CALIBRACIÓN.....</b>	<b>33</b>



4.1.	PRINCIPIOS TEÓRICOS.....	33
4.2.	ETAPAS.....	34
4.3.	ADQUISICIÓN .....	35
4.4.	FILTRADO DE LA NUBE .....	36
4.4.1.	IMPLEMENTACIÓN .....	37
4.4.2.	EXPLICACIÓN GRÁFICA DEL FILTRADO.....	37
4.5.	SEGMENTACIÓN DE LA ESFERA .....	39
4.5.1.	ETAPAS .....	39
4.5.2.	IMPLEMENTACIÓN .....	41
4.5.3.	EXPLICACIÓN GRÁFICA SEGMENTACIÓN.....	42
4.6.	CÁLCULO DE LA TRANSFORMADA .....	43
4.6.1.	EXPLICACIÓN .....	43
<b>5.</b>	<b>SISTEMA DE DETECCIÓN DE INTRUSOS.....</b>	<b>45</b>
5.1.	PRINCIPIOS TEÓRICOS.....	45
5.2.	ETAPAS.....	46
5.3.	MODELADO DE LA ESCENA.....	48
5.3.1.	IMPLEMENTACIÓN .....	50
5.3.2.	REPRESENTACIÓN GRÁFICA DEL PROCESO.....	50
5.4.	VIGILANCIA .....	52
5.4.1.	IMPLEMENTACIÓN .....	53
5.4.2.	EXPLICACIÓN GRAFICA DEL PROCESO .....	54
<b>6.</b>	<b>RESULTADOS EXPERIMENTALES .....</b>	<b>57</b>
6.1.	PROPIEDADES DEL SISTEMA Y OBJETIVOS CUMPLIDOS .....	57
6.2.	TIEMPOS DE PUESTA EN MARCHA Y COMPUTO.....	58
6.3.	PRECISIÓN DE LA SEGMENTACIÓN .....	58
6.4.	MEJORAS .....	59
6.4.1.	COLOR .....	59
6.4.2.	DISTANCIA DE FILTRO .....	59
6.4.3.	POSICIÓN DE LA ESFERA.....	59
6.4.4.	RESOLUCIÓN DE LA VIGILANCIA .....	60
6.5.	CONCLUSIONES .....	61
<b>7.</b>	<b>CONCLUSIONES Y TRABAJOS FUTUROS.....</b>	<b>63</b>
7.1.	CONCLUSIONES .....	63
7.2.	TRABAJOS FUTUROS .....	64
<b>8.</b>	<b>PRESUPUESTO .....</b>	<b>65</b>
<b>9.</b>	<b>PLANIFICACIÓN.....</b>	<b>67</b>
<b>10.</b>	<b>ANEXO I: FUNDAMENTOS EN ROBÓTICA.....</b>	<b>69</b>



A.	MATRIZ DE TRANSFORMACIÓN .....	69
11.	ANEXO II: ESTRUCTURACIÓN DE NUBES DE PUNTOS .....	71
A.	OCTREE .....	71
12.	REFERENCIAS.....	73



## Listado de figuras

Figura 1: Sensor Kinect de Microsoft.....	20
Figura 2: Componentes Microsoft Kinect.....	20
Figura 3: Rangos de Profundidad Sensor Microsoft Kinect.....	20
Figura 4: Representación del campo de movimiento del sensor Kinect.....	21
Figura 5: Arquitectura de la SDK de Kinect para Windows.....	24
Figura 6: Hardware y software de Kinect para la interacción con una aplicación.....	25
Figura 7: Módulos de PCL.....	26
Figura 8: Esquema de los diferentes módulos del proyecto.....	30
Figura 9: Esquema de calibración.....	34
Figura 10: Esquema adquisición de todas las nubes.....	35
Figura 11: Esquema filtrado de nube.....	36
Figura 12: Nube a filtrar.....	37
Figura 13: Nube filtrada de 3 a 5 metros.....	38
Figura 14: Nube filtrada de 1.2 a 2 metros.....	38
Figura 15: Segmentación de la esfera.....	40
Figura 16: Nube anteriormente filtrada.....	42
Figura 17: Nube con la esfera encontrada.....	42
Figura 18: Resultado de la segmentación.....	42
Figura 19: Esquema del cálculo de la matriz de transformación.....	44
Figura 20: Esquema etapas detección de intrusos.....	47
Figura 21: Esquema modelado de la escena.....	49
Figura 22: Nube de puntos posición frontal sin diana geométrica.....	50
Figura 23: Nube de puntos posición derecha sin diana geométrica.....	51
Figura 24 A: Mapa completo sin matriz aplicada.....	51
Figura 24 B: Mapa completo con matriz aplicada.....	51
Figura 25: Esquema Vigilancia.....	53
Figura 26: Mapa completo.....	54
Figura 27: Nube en tiempo real.....	55
Figura 28: Puntos resultantes.....	55
Figura 29: Vigilancia completa 1.....	56
Figura 30: Vigilancia completa 2.....	56
Figura 31: Errores en la vigilancia 1.....	60
Figura 32: Errores en la vigilancia 2.....	60
Figura 33: Vigilancia sin errores.....	61
Figura 34: Mismo punto desde dos sistemas de coordenadas diferentes.....	69
Figura 35: Estructura octree.....	71



## Listado de tablas

TABLA 1. Flujos de datos de Kinect.....	22
TABLA 2. Diferentes resultados de tiempos.....	58
TABLA 3. Precio de amortización.....	65
TABLA 4. Coste de desarrollador.....	65
TABLA 5. Planificación.....	67



## Listado de ecuaciones

(1) .....	33
(2) .....	43
(3) .....	43
(4) .....	43
(5) .....	43
(6) .....	43
(7) .....	43
(8) .....	65
(9) .....	69
(10) .....	69



## Agradecimientos

A mis padres, por enseñarme a ser persona y por apoyarme con todas las idioteces que se me han ocurrido.

A mi hermana, por escucharme siempre.

A Alex, por ser el mejor compañero que he podido tener en este viaje.

A Javi, por recordarme siempre quien soy.

A Athos, por sacarme siempre una sonrisa.

A Niki, por enseñarme qué es el amor.





## Resumen

Actualmente, vivimos en una sociedad en un constante avance y empeño por un desarrollo continuo de la seguridad. El progreso de los sistemas de seguridad está ligado al desarrollo de la raza humana como especie, para proporcionar una situación más cómoda y tranquila para las personas. Los sistemas tradicionales de seguridad, en los que un usuario se encarga de la monitorización de varias pantallas, presentan algunos errores o limitaciones debidos al factor humano, que con el desarrollo de la investigación se han conseguido erradicar, con nuevos sistemas de vigilancia inteligente.

Estos nuevos sistemas son más efectivos, ya que automatizan las tareas de vigilancia y no dependen del rendimiento del personal encargado de la monitorización. Estos nuevos sistemas, han ido incorporando nuevos tipos de sensores, además de las tradicionales cámaras a color, que presentan una fuerte dependencia de la iluminación. Para eliminar este problema se han empezado a utilizar sensores como los 2.5D que permiten capturar tanto datos de color como de profundidad, eso hace que la robustez de cualquier sistema de vigilancia basado en estos sensores sea más alta que cualquier sistema de vigilancia inteligente basado en cámaras. Una ventaja de estos sensores, es su bajo coste, ya que se tratan de sensores 2.5D y no sensores 3D que pueden dar una información más precisa, pero viendo los resultados de muchos sistemas basados en los primeros, podemos decir que la implementación de los mismos es un completo acierto. Un problema que comparten tanto sensores 2.5 D como las cámaras de vigilancia, es el campo de visión de los mismos, creando una necesidad de ampliación de la zona de vigilancia en algunas situaciones.

En la presente memoria se propone un sistema de vigilancia basado en datos tridimensionales capturados desde diferentes sensores Microsoft Kinect para Windows. El avance de este proyecto respecto a los sistemas tradicionales de seguridad, es la falta de dependencia a diferentes elementos, como pueden ser los datos de color, y en consecuencia la iluminación, a parte del antes mencionado operador humano. Nuestro proyecto consistirá en la adquisición datos tridimensionales desde los diferentes sensores, colocando previamente una diana geométrica que nos servirá como referencia a la hora de crear un mapeado completo con nuevos datos tridimensionales. Lo crearemos a partir del cálculo de una matriz de transformación la cual será hallada a partir del centro de la diana geométrica que hayamos colocado en la escena. Esta matriz multiplicará una de las nubes de puntos tomadas desde uno de los sensores, transformándola, para después sumarla a la otra nube, adquiriendo así el mapeado completo. Este mapeado nos servirá para comparar con nuevos datos tridimensionales tomados en tiempo real, de esta forma podremos ver las diferencias existentes entre cómo debería ser la situación en la escena y como es, detectando así los posibles intrusos.

En las diferentes pruebas a las que se ha sometido al prototipo los resultados han sido exitosos, incluso forzando el sistema a trabajar en condiciones adversas, probando así la robustez del sistema. Y pensando desde un principio en aumentar la sencillez para el usuario, creando una aplicación versátil.



## Abstract

Today, we live in a society in constant progress and commitment for continued development of security. The progress of the safety systems is linked to the development of the human race as a species, to provide a more comfortable and quiet for people situation. Traditional security systems, in which a user is responsible for monitoring multiple screens, have some limitations due to the human factor, but new intelligent surveillance systems have been created for eradicate this limitation.

The intelligent surveillance systems are more effective. These systems automate surveillance tasks and not dependent on the performance of personnel responsible for monitoring. These new systems have been incorporating new types of sensors, in addition to traditional color cameras, which have a strong dependence of lighting. To eliminate this problem, the new systems have started using sensors such as 2.5D that capture color data and depth, that makes these systems safer than any system based surveillance cameras. One advantage of these sensors is their low cost, and we can say that the implementation of these is a complete success. A problem shared by both 2.5 D sensors and surveillance cameras, is the field of view, because in specific situation you need more than one camera or sensor.

In this document we propose a surveillance system based on three-dimensional data captured from different sensors Microsoft Kinect for Windows. The progress of this project over traditional security systems is the lack of dependence on different elements, such as color data, and consequently lighting, besides the aforementioned human operator. Our project will consist of three-dimensional data acquisition from different sensors, previously placing a geometric target that will serve as a reference when creating a full three-dimensional mapping with new data. We will create from the calculation of a transformation matrix which will be calculated with the geometric centers of the target we have placed at the scene in different positions. The point clouds taken from one of the sensors will be multiply by the transformation matrix, transforming it, then add it to the other cloud, thereby acquiring the complete mapping. This mapping will help us to compare with new three-dimensional real-time data collected in this way we can see the differences between how the situation should be on the scene as it is, thereby detecting potential intruders.

In the various tests that has been subjected to prototype, the results have been successful, even forcing the system to work in adverse conditions, proving the robustness of the system. And thinking from the beginning to increase the simplicity for the user, creating a versatile application.



## 1. Introducción

Los sistemas de vigilancia son un elemento necesario en la sociedad actual, y han ido avanzando a lo largo de la historia con el objetivo de hacer más segura nuestra vida. Los sistemas de vigilancia sirven para proteger diferentes elementos que consideramos importantes, desde una propiedad hasta nuestra propia vida. Los sistemas tradicionales de vigilancia, consistían en unas cámaras que enviaban el vídeo capturado en directo a un gran número de pantallas, a través de las cuales un operador humano podía observar la escena. Este operador humano puede cometer errores, después de horas de trabajo su atención no sería tan elevada como podría ser al principio, o un simple despiste podría provocar que la escena no estuviese vigilada por un periodo de tiempo.

Por este motivo, comenzaron a desarrollarse diferentes sistemas de vigilancia inteligente. Estos sistemas de vigilancia inteligente, utilizan la visión por computador, para poder hacer un análisis de la escena en tiempo real, pudiendo detectar intrusos o situaciones no deseadas. A través de unas cámaras y un ordenador que se encargase del análisis en tiempo real de las escenas que iban captando, se podía llegar a detectar esas situaciones determinadas.

Un avance de estos procesos, es la incorporación de sensores de profundidad, que se encargasen de hacer esa misma labor, pero eliminando la dependencia de la luz o los colores. Estos sensores de profundidad pueden llegar a crear modelos tridimensionales de las escenas sin la necesidad de iluminación, además de realizar capturas de tridimensionales en tiempo real. Este proyecto se basa en la comparación del aspecto del modelo de la escena con el de la escena recogido en tiempo real, de esa forma pueden detectar si un determinado objeto ha sido sustraído, o la presencia de intrusos en la propia escena, además de otro tipo de situaciones.

Algunos sensores 2.5 D, que no presentan un coste elevado, pueden llegar a ser muy útiles en este tipo de procesos. Un ejemplo de estos sensores de 2.5D es el sensor Microsoft Kinect para Windows, el cual recoge datos de profundidad y color, haciendo posible su implementación en un sistema de vigilancia inteligente basado en la creación de entornos tridimensionales. Estos sensores no son tan precisos como otros de un coste mucho mayor, los cuales son llamados sensores 3D, pero realmente sirven para este tipo de aplicaciones, ya que el error de precisión es mínimo y nada o poco influyente en estos sistemas inteligentes de vigilancia.

Para llevar a cabo un sistema de vigilancia realmente efectivo, no hay que dejar puntos muertos en la propia escena, por ejemplo, en alguna situación en la que algún elemento no permita una vigilancia completa, dejando solo vigilada una parte de la propia habitación. Por esto mismo, existe la posibilidad de utilizar más de un sensor, pero para ello, tenemos que juntar los datos tridimensionales captado por uno de los sensores con los datos obtenidos por el otro sensor. Esto mismo, se podría llevar a cabo, calibrando ambos sensores, encontrando una referencia en la escena, la cual nos permita situar y alinear los datos obtenidos por cada uno de los sensores, creando de esta forma un mapa que represente la escena real.

El presente proyecto consiste en la calibración de dos sensores Microsoft Kinect para Windows, con el objetivo de crear un sistema de vigilancia basado en datos tridimensionales, con independencia a la luz y el color. En un primer momento se tomarán datos tridimensionales a partir de ambos sensores, colocando una diana geométrica en la propia escena a vigilar, después volviendo a colocar la diana en otra posición, para después volver a tomar datos de los mismos sensores. A partir del análisis de los datos obtenidos, se podrá encontrar la posición de esa diana geométrica, la cual dará la información necesaria para calcular la matriz de transformación. Se volverán a tomar datos desde ambos sensores, pero esta vez sin diana geométrica, de tal forma que la escena esté como se quiera mantener en el momento de la vigilancia. Se utilizará la matriz de transformación para crear un mapeado completo de la escena, con los datos obtenidos de la última captura. Este mapeado se comparará con nuevos datos en tiempo real, de esta forma el sistema detectará las diferencias de ambos mapas, creando así, un sistema de vigilancia desde diferentes posiciones con sensores Microsoft Kinect.

Una de las ventajas, es que la calibración en ningún momento depende ni de colores ni de luz, ya que tomaremos como referencia un objeto geométrico tridimensional, que puede tener el color que sea y puede estar no iluminado, de esta forma, a parte podríamos vigilar la escena sin ningún tipo de iluminación. Nuestro sistema de calibrado, no provoca que la escena tenga que cumplir unas determinadas



características, ya que hemos enfocado el proyecto desde un principio hacia un procedimiento más simple para el usuario encargado de llevarlo a cabo.

## 1.1. Objetivos

El objetivo del sistema es la vigilancia de una escena desde diferentes posiciones, a partir de los datos obtenidos por los sensores Microsoft Kinect.

El objetivo podemos dividirlo en diferentes procesos:

- Creación de diferentes nubes de puntos desde diferentes localizaciones con diana geométrica.
- Segmentación y obtención de datos de la diana geométrica.
- Calculo de la matriz de transformación.
- Aplicación de la matriz de transformación para adquirir el mapeado completo.
- Vigilancia de la escena.

## 1.2. Estructura de la tesis

La presente tesis se estructura de la siguiente manera:

En primer lugar, se analizarán las diferentes aplicaciones que se han podido estudiar de otros proyectos relacionados, en el estado del arte.

Posteriormente se describirá de forma general, el proyecto llevado a cabo, incluyendo un análisis de las herramientas utilizadas, en el capítulo de descripción general, para después comentar cada uno de los módulos los cuales forman la aplicación.

En capítulos posteriores, se analizará al detalle cada uno de esos módulos, antes mencionados, explicados de diferentes maneras para una mejor comprensión, en los capítulos de calibración y registro de intrusos.

Después se comentarán los diferentes resultados que se han obtenido, llegando a conclusiones y otras posibles aplicaciones, en los capítulos de resultados experimentales y conclusiones.

En los últimos capítulos se analizará el presupuesto y la planificación, así como dos anexos los cuales explicaran algunos elementos utilizados.



## 2. Estado del Arte

Cualquier sistema de vigilancia tiene como principio fundamental la detección. Es a partir de este principio en el que se lleva a diferentes procesos o aplicaciones.

En general los sistemas de vigilancia básicos, dependen en primera medida de la iluminación de la escena. Haciendo que estos sistemas, tengan una clara limitación en diferentes situaciones, en las que el escenario no tiene una gran iluminación u otros casos, en las que diferentes zonas de estudio, son inalcanzables por el mismo sensor.

Ha habido diferentes avances en los sistemas de vigilancia, en los que la base que sustenta estos mismos procesos, consta de una reconstrucción tridimensional del escenario, en el que la luz no juega un papel fundamental en la detección para su implementación en el sistema de vigilancia.

La introducción de nuevos sensores de bajo coste, en los que puedes llegar a captar, color y profundidad, hacen que se puedan llegar a crear sistemas de seguridad competentes sin necesidad de un gasto excesivo por parte del usuario.

Siendo tan accesible este tipo de sensores, hace que la comunidad de desarrolladores se vuelque buscando nuevas formas de detección o incluso otras aplicaciones posibles.

Antes de profundizar en la tesis que exponemos, vamos a ver las diferentes líneas de investigación que se han llevado a cabo en este terreno.

- Métodos de detección
- Modelado 2D y 3D del entorno
- Métodos de calibración
- Tecnologías de adquisición de datos 3D
- Sistemas de medición de distancias mediante imágenes



## 2.1. Métodos de detección de cambios

Para la identificación de los elementos de interés en la escena vigilada son utilizados múltiples métodos que pueden dividirse en las siguientes líneas de investigación generales:

- Identificación de elementos en movimiento.
- Sustracción del fondo.
- Detección de cambios con respecto a un modelo.

Todos estos métodos se basan en la idea de identificar a los elementos de interés por poseer una determinada propiedad, bien la de estar en movimiento, la de no pertenecer al fondo de la escena, o a un modelo ideal de la misma.

La mayor parte de la investigación existente se ha basado en el desarrollo de algoritmos para trabajar con imágenes 2D, a color, o en escala de grises, especialmente en el caso de los tres primeros grupos de métodos de detección. La incorporación de sensores que permiten conocer la distancia a la que se encuentran los objetos, facilitó las operaciones para sustraer el fondo de las escenas, e impulsó el desarrollo de nuevos métodos basados en la comparación de la escena con un modelo tridimensional de la misma. A continuación, es realizado un repaso por las investigaciones realizadas en cada una de estas cuatro ramas.

- Identificación de elementos en movimiento: Estos algoritmos se basan generalmente en la comparación de sucesivas capturas de la escena, ya sea mediante datos 2D o 3D. Los elementos en movimiento provocan cambios en el aspecto de las capturas de la escena realizadas en distintos instantes de tiempo.  
Uno de los algoritmos más conocidos para realizar este tipo de detección y que trabaja con datos 2D es el Optical Flow[1]. Se trata de un método propuesto como paso de procesamiento para algoritmos de visión de alto nivel. Una evaluación de los posibles algoritmos de Optical Flow aparece en [2].  
Este tipo de algoritmos han sido mejorados mediante la incorporación de la identificación y el modelado del fondo de la escena [3].
- Sustracción del fondo. En la mayoría de los sistemas una única cámara es usada para observar un área. La detección del fondo se realiza generalmente a partir de las diferencias de intensidad entre los objetos y el fondo. Debido a los posibles cambios en el fondo, la información temporal [4] es usada para actualizarlo dinámicamente. Este tipo de sustracción del fondo está restringida a los casos en los que se mueven son los objetos de interés y en los que la luz no cambia mucho. Aunque existen técnicas estadísticas para adaptar las condiciones de luz cambiante para calcular el fondo [5], el uso de solo información 2d reduce la efectividad de tales sistemas.
- Detección de cambios. La detección de cambios en múltiples imágenes 2D de una escena [6] es un campo muy estudiado y con un gran número de aplicaciones en diversas disciplinas, incluyendo la vigilancia, la diagnosticarían médica [7], evaluación de edificios [8], o incluso la supervisión del cambio climático [9].

Actualmente se están desarrollando algoritmos de detección de cambios que emplean un modelo 3D de la situación ideal de la escena y comparan el aspecto real de la escena con dicho modelo [10].



Entre los métodos de detección de cambios mediante datos 3D destacan los basados en el empleo de “voxels” o cajas tridimensionales [11] que permiten tener un conocimiento volumétrico de la escena y de su modelo, facilitando la comparación entre las regiones representadas en cada uno de ellos.

## 2.2. Modelado 2D y 3D del entorno

Existen diferentes tipos de modelos tridimensionales, en función de la información que contiene a cerca del objeto o de la escena modelada. A continuación, se presentan varios tipos de modelos tridimensionales de menor a mayor complejidad en la obtención de los mismos.

- Nubes de puntos. Una nube de puntos como su propio nombre indica representa con un punto cada muestra o medición realizada sobre escena, gracias a las coordenadas tridimensionales de dicho punto capturado. Los puntos pueden contener, además, información de color.
- Modelos poligonales o de mallas. En una representación poligonal de una forma, una superficie curva es modelada como muchas pequeñas superficies planas. El proceso de convertir una nube de puntos en un modelo poligonal 3D se llama reconstrucción, e implica encontrar y conectar los puntos adyacentes mediante líneas rectas con el fin de crear una superficie continua.
- Modelos de superficies. El siguiente nivel de sofisticación en la modelización implica el uso de un conjunto de pequeñas superficies curvas que unidas entre sí modelan una forma. Estas superficies pueden ser NURBS, T-Splines u otras representaciones curvas.
- Modelos sólidos CAD. Estos modelos no describen simplemente la forma del objeto, sino que también incorporan las características y parámetros fundamentales que la definen. Desde el punto de vista de la ingeniería y la fabricación, la representación fundamental de una forma digitalizada es el modelo CAD, totalmente editable.

Los algoritmos de modelado 3D permiten obtener una representación completa de una persona, un objeto o una escena. Los modelos de escenas o entornos amplios también son conocidos como mapas.

Para realizar un modelo completo ya sea de una escena, o de un objeto es necesario tomar diferentes capturas del mismo desde distintos puntos de vista, empleando sensores 2D o 3D. Posteriormente estas vistas o capturas deben ser integradas para representar al objeto o escena en un único modelo. Dependiendo del tipo de datos capturados, imágenes o datos tridimensionales, existen diferentes métodos o algoritmos de modelado o reconstrucción, que repasamos a continuación.

- Reconstrucción 3D a partir de imágenes. Estos métodos son empleados generalmente para modelar amplias zonas exteriores [12], donde la tecnología de los sensores 3D no siempre funciona correctamente. Se suelen emplear múltiples cámaras rodeando la escena [13]. También es posible la utilización de diferentes imágenes tomadas con vehículos aéreos que son integradas con técnicas SFM (Structure from motion). Existe una solución a medio camino entre la reconstrucción a partir de imágenes y la captura de puntos tridimensionales, que consiste en la reconstrucción a partir de las capturas realizadas con cámaras estéreo [14-15].



- Registro de nubes de puntos 3D. El registro consiste en la integración en un único modelo de datos los datos 3D obtenidos desde diferentes puntos de vista. Se trata de referenciar todos los puntos 3D respecto a un mismo sistema de referencia.

Para ello se han desarrollado métodos semiautomáticos [16], e incluso programas comerciales con interfaces gráficas para ayudar al registro de dos nubes de puntos, como Rapidform XOR. Este tipo de métodos necesita de la colaboración de un usuario para identificar algunos pares de puntos correspondientes (representativos de un mismo punto de la escena) de dos nubes de puntos, y posteriormente es calculada la transformación geométrica necesaria para hacerlos coincidir en el espacio.

La búsqueda automática de los pares de puntos correspondientes es una de las principales líneas de investigación, donde se intenta conseguir algoritmos robustos al ruido en las nubes de puntos [17], y cada vez más rápidos, como los basados en el método de “cubos binarios” (CBC, Coarse Binary Cubes) [18].

Con los algoritmos descritos se obtienen generalmente modelos de nubes de puntos, si se pretende conseguir otro tipo de modelos, más complejos (de mallas, superficie, o CAD), es necesario el tratamiento de las nubes de puntos obtenidas mediante algoritmos de integración de mallas, empleo de mapas de textura y la parametrización.

En el caso del modelado de personas, actualmente la investigación se dirige hacia el desarrollo de métodos que permitan obtener información no solo de la forma, sino también de los movimientos y comportamientos humanos [19].

Por otra parte, las investigaciones sobre el modelado de escenas interiores, avanzan dirigidas a obtener de algoritmos que permitan disponer de un modelo semántico de la escena, gracias a la identificación y reconocimiento de sus elementos, y al desarrollo de mapas dinámicos, que varían y se actualizan dependiendo de las condiciones. Estos algoritmos se basan en métodos de aprendizaje [20], y en la fusión de la información recibida por diferentes tipos de sensores [21].

El desarrollo de nuevos sensores comerciales, que incorporan sensores de color y de profundidad (como Microsoft Kinect), ha impulsado el desarrollo de métodos de modelado de interiores, basados en la combinación de los de datos de color y de profundidad [22]. Gracias a estos sensores es posible modelar tanto cuerpos humanos [23], como escenas interiores [24].

### 2.3. Tecnologías de adquisición de datos 3D

La geometría de una escena o un objeto puede ser adquirida mediante una gran variedad de técnicas y sensores con un amplio rango de coste y precisiones.

Existen dos tipos de técnicas en función de la necesidad o no de contacto con el objeto. Los sensores basados en el contacto con el objeto, como las máquinas de medición por coordenadas (CMM, Coordinate Measuring Machine), examinan la superficie del mismo apoyando en ella el elemento de medida. Las tecnologías de adquisición que no precisan de contacto pueden dividirse a su vez en dos grupos, sensores pasivos, o activos.

Los sensores pasivos no emiten ninguna clase de radiación por sí mismos, en su lugar detectan la radiación reflejada del ambiente. La mayoría de los sensores de este tipo detectan la luz visible porque es una





radiación ya disponible en el ambiente. Otros tipos de radiación, tal como el infrarrojo podrían ser utilizados también. Algunos ejemplos de sensores pasivos son los sistemas estereoscópicos o los escáneres de silueta.

Los sensores activos emiten alguna clase de señal y analizan su retorno para capturar la geometría de un objeto o una escena. Se utilizan radiaciones electromagnéticas (desde ondas de radio hasta rayos X) o ultrasonidos. Los más conocidos son escáneres láser, que pueden alcanzar precisiones del orden de los milímetros.

Algunas de las tecnologías empleadas para determinar la distancia al a que se encuentra el objeto en el que incide el rayo láser son la de Tiempo de vuelo (TOF, Time of Flight), triangulación, diferencia de fase, la Holografía Conoscópica o la luz estructurada.

Además, cada vez está más extendido el uso de sensores 2.5 D, que miden la profundidad o distancia a la que se encuentra un objeto del sensor. Este tipo de sensores, como los de Prime Sense, emplean la tecnología de luz codificada (Light Coding Technology). Esta técnica consiste en emitir una luz de infrarrojo (IR) cercano con un determinado patrón, la luz es distorsionada dependiendo de la profundidad de los objetos. Un sensor de imagen CMOS lee la luz codificada reflejada por la escena y usando algoritmos de triangulación extrae la información de profundidad.

Los sensores de IR actualmente son de gran ayuda en las aplicaciones de vigilancia, por lo que aumenta el número de nuevos sensores basados en este tipo de tecnología [25 - 26]

## 2.4. Sistemas de medición de distancias mediante imágenes

El empleo de cámaras como sensores de medición de distancia ha tenido diferentes aplicaciones. Por ejemplo, su utilización para permitir la navegación autónoma de robots terrestres; en arquitectura como un instrumento de medición en interiores para obtener las dimensiones de paredes y pisos, así como para la ubicación correcta de muebles con el propósito de diseño de interiores, en exteriores para medir el tamaño y la posición de ventanas y puertas, y muchas más.

Una imagen o secuencia de imágenes trae consigo una cantidad muy grande de información geométrica acerca de la escena representada, se han desarrollado diferentes técnicas para la construcción de escenarios 3D a partir de imágenes en 2D. Se han desarrollado técnicas y métodos de descomposición de imágenes para su representación en el espacio del mundo real.

Thomas Bucher, describe un método para mapear una imagen a coordenadas del mundo real y obtener así, una aproximación de la altura de objetos, longitudes y cambios de posición; basándose en un pequeño grupo de parámetros de fácil estimación a partir de características de los objetos o marcas en la escena, esto sin la necesidad de requerir alguno de los parámetros intrínsecos de la cámara.

Al área de investigación que corresponde el obtener mediciones de distancias a partir de imágenes se le llama Metrología Visual y puede ser clasificada en dos tipos: Metrología de simple vista y metrología de múltiples vistas.

Entre las ventajas de la metrología visual sobre otras técnicas es que sólo se requiere una vista del objeto (capturada en una imagen) para hacer una medición, por lo cual se considera un método no invasivo, fácil de utilizar, con mayor cantidad de información, posibilidad de determinar muchas distancias en el sistema



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES

en base a una secuencia de imágenes y registro histórico para análisis posterior. Aunque, en ciertas aplicaciones, el resultado de la medición se requiere lo más pronto posible con respecto al momento en el que ésta se realizó, y el procesamiento digital de la imagen siempre consumirá un tiempo que se debe tomar en cuenta en el sistema de medición.

El problema de medir las dimensiones de objetos de manera directa con instrumentos como por ejemplo, cinta métrica, flexometro, regla, calibrador (Vernier), micrómetro, etc., es que el objeto tiene que estar disponible físicamente para colocar el instrumento de medición sobre él. Existen otros métodos para medir distancias sin contacto llamados activos, que requieren la activación de un emisor para generar ya sea un ultrasonido, un rayo de luz infrarroja o un láser; éste emisor, se debe direccionar hacia un punto específico para medir mediante un receptor, las características de retorno de la señal emitida y así, poder determinar la distancia existente entre el instrumento y un punto sobre un objeto remoto.

En la actualidad, éste tipo de instrumentos de medición de distancias digitales pueden traer incorporadas las funciones de registro histórico de las mediciones en chips de memoria interna o externa al instrumento, así como algunos tienen la posibilidad de comunicación con una computadora mediante puertos de comunicación como RS232, USB o Ethernet.



### 3. Descripción general

Como se comentó anteriormente en el presente documento, este proyecto tiene el objetivo de crear un sistema de vigilancia a partir de dos sensores Microsoft Kinect.

En este capítulo se analizarán las diferentes herramientas que hemos utilizado, tanto a nivel de hardware como de software.

El sensor Microsoft Kinect, ha sido elegido por su relación calidad precio, ya que es capaz de adquirir datos realmente fiables, con un coste muy pequeño.

Mientras que herramientas de Software como pueden ser las librerías PCL, se ajustan a las necesidades que tenemos con este proyecto, para el análisis de todos los datos adquiridos por el sensor Microsoft Kinect, dándonos la posibilidad de eliminar la dependencia al color.

#### 3.1. Hardware

Este proyecto tiene por objetivo la creación de un sistema de vigilancia a partir de sensores Kinect.

En este apartado se describen las herramientas utilizadas a nivel de hardware. A continuación, se presenta el sensor Kinect para Windows, el cual es el idóneo por sus características para este proyecto, ya que tiene un coste reducido y permite la adquisición de datos 3D en entornos interiores.

##### 3.1.1. Sensor Kinect

Kinect, originalmente conocido con el nombre en clave “Project Natal”, es un controlador de juego y de entretenimiento creado por Alex Kipman. Este proyecto fue desarrollado por Microsoft, el cual se pensó para su implementación en la videoconsola Xbox 360, y que tiene su análoga en la Xbox One. Este sensor fue desarrollado desde junio del 2011 para PC a través de Windows 7 y Windows 8.

Este sensor permite a los usuarios controlar la consola, y algunos juegos creados específicamente para esto, sin la necesidad de un contacto físico con ningún controlador, reconociendo gestos, comandos de voz, objetos e imágenes.

El sensor Microsoft Kinect, una vez que salieron al mercado, no solo se utilizaron para el control de videojuegos o consolas, ya que tiene infinidad de aplicaciones posibles.

Se han llegado a utilizar en ciertos campos, en los que no estaban pensados en un principio, por ejemplo, la robótica. Ahora mismo es una buena forma de dotar a tu ordenador de un sistema de entrada visual y auditiva.

Los diferentes sensores que contiene la Kinect, de color, profundidad y audio, están colocados en una estructura alargada en una posición horizontal, la cual es sujeta por una base, estando unidos por medio de un pivote monitorizado que permite ajustar la orientación de la estructura horizontal.



Figura 1: Sensor Kinect de Microsoft

A continuación, son descritos cada uno de los componentes de esta herramienta, además de sus especificaciones, se analizará los datos diferentes que puede proporcionar la Kinect del entorno en el que esté, y finalmente se verá la aplicación de este mismo sensor en tareas de vigilancia, de forma que veamos la relación de las prestaciones de esta herramienta con este proyecto.

#### 3.1.1.1. Datos técnicos Microsoft Kinect

Un sensor Microsoft Kinect utiliza los siguientes componentes:

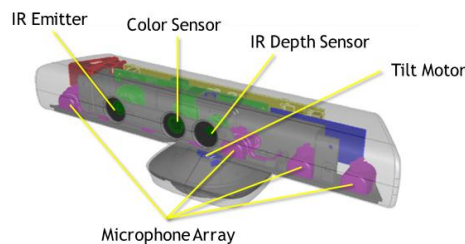


Figura 2: Componentes Microsoft Kinect

- Una cámara RGB, la cual tiene una resolución de 1280x960 a una velocidad de captura de 12 fps, o una resolución de 640x480 a 30fps.
- Un sensor de profundidad, el cual es un emisor laser de infrarrojos IR y sensor CMOS monocromo, el cual ha sido desarrollado por Prime Sense. Existen dos tipos de rango de profundidad, el primero es un rango por defecto, el cual tienen tanto la Kinect de Xbox 360 como la de Windows. Y otro rango que es un rango cercano, el cual está solo disponible para la Kinect de Windows.

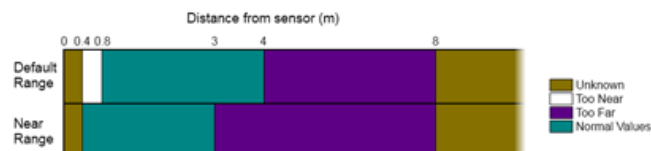


Figura 3: Rangos de Profundidad Sensor Microsoft Kinect

- Un micrófono multi-array formado por cuatro sensores diferentes de sonido. Utilizando cuatro sensores diferentes de audio, podemos conocer la localización de la fuente de sonido así como la dirección de ese mismo sonido.
- Un acelerómetro de tres ejes, el cual está configurado para un rango de 2G, donde G es la aceleración de la gravedad.
- Un motor de inclinación con un rango de giro de  $\pm 27$  grados. Este motor permite cambiar la orientación del sensor.

#### 3.1.1.2. Especificaciones sensor Kinect

Campo de visión:

- Campo de visión horizontal: 57 grados.
- Campo de visión vertical: 43 grados.
- rango de inclinación física:  $\pm 27$  grados.
- Rango de profundidad del sensor: 0,5 - 5 metros.

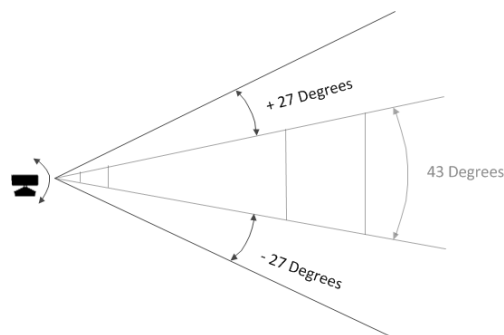


Figura 4: Representación del campo de movimiento del sensor Kinect

Flujos de datos:

El sensor Kinect puede capturar datos de tres tipos: audio, color y profundidad.

En la siguiente tabla presenta los diferentes flujos de datos de la Kinect.



		Resolución (píxeles)	Imágenes por segundo (fps)
Imágenes de color	RawBayer	1280x960	12
		640x480	30
	Raw YUV	640x480	15
		1280x960	12
		640x480	30
Imagen de infrarrojos	YUV	640x480	15
		640x480	30
Imágenes de profundidad		640x480	30
		320x240	30
		80x60	30

TABLA 1. Flujos de datos de Kinect

#### 3.1.1.3. Microsoft Kinect para Windows como sensor de vigilancia

Los componentes incorporados en el sensor Kinect para Windows, y los flujos de datos que permite obtener, lo convierten en un sensor muy potente y adaptable diversas aplicaciones, con una relación calidad-precio óptima.

Microsoft Kinect posee las capacidades necesarias y suficientes para ser utilizado en la vigilancia de interiores. Las ventajas que presenta frente a otros sensores, es la posibilidad de obtener información tridimensional de la escena por un bajo precio, permitiendo una gran velocidad de adquisición de datos mediante sensores ya calibrados. Permite la fusión de datos tridimensionales y de color para la reconstrucción de una escena en interiores y su posterior vigilancia sin necesidad de iluminación en la misma. Además, Kinect incorpora un motor con el que es posible cambiar la orientación del sensor y vigilar una región de amplias dimensiones.

Actualmente son varias las líneas de investigación que estudian el desarrollo de diferentes aplicaciones del sensor Kinect como sensor de vigilancia entre las que es posible destacar: reconocimiento de personas, su modelado, análisis de movimientos y comportamientos, modelado de entornos 3D o creación de mapas, sensor para el control de robots móviles.

Con respecto a Kinect para XBOX, el uso de Kinect para Windows ofrece varias características que no están disponibles en el sensor anterior:

- Modo cercano. Permite medir profundidades a partir de 40 cm.
- Modo sentado o de 10 articulaciones. Permite el seguimiento de cabeza, cuello y brazos para usuarios sentados o en pie.
- Cable USB. Asegura la fiabilidad en una amplia gama de ordenadores y mejora la convivencia con otros periféricos USB.
- Ajustes de la cámara ampliados. Proporciona ajustes adicionales, como el brillo, para mejorar la adquisición de datos.
- Kinect Fusión. Permite realizar reconstrucciones 3D.



En cuanto al desarrollo de nuevas aplicaciones, es importante destacar los siguientes aspectos:

1. **Precio.** Kinect para Windows con un precio de unos \$250 cuesta aproximadamente el doble que Kinect para Xbox 360. La SDK (Software Development Kit) de Kinect para Windows es de uso libre.
2. **Licencia.** En la nueva licencia que está disponible con el Kinect para Windows versión 1.0 y la versión de software 1.5 permite a los desarrolladores el desarrollo y distribución de aplicaciones comerciales con Kinect para Windows. Esto significa que para hacer pública una aplicación es necesario usar Kinect para Windows, ya que mediante Kinect para Xbox 360 no es legal. Eso explica el alto precio del sensor Kinect para Windows, ya que incluye las licencias.

## 3.2. Software

### 3.2.1. Introducción

En esta sección, se abordarán, las herramientas de software que se han utilizado. Siendo estas, la SDK de Kinect y las librerías PCL.

La elección de estas herramientas, está justificada por diferentes motivos:

- La SDK de Kinect, por ejemplo, es necesaria, ya que de esta forma podemos controlar el propio sensor. Esta misma SDK, además, permite conectar más de un sensor, algo necesario para nuestro proyecto.
- La librería PCL, es la que hemos elegido para llevar a cabo nuestro análisis de la escena, tanto como para la representación, como para la identificación de los elementos de la escena.

### 3.2.2. SDK Kinect

La SDK de Kinect para Windows proporciona una biblioteca de software sofisticada y herramientas para optimizar el empleo de la interfaz natural de usuario, NUI (Natural User Interface), en que se basa Kinect, que detecta y reacciona a los acontecimientos del mundo real.

La primera versión de la SDK fue lanzada por Microsoft en junio de 2011, desde entonces se han incorporado mejoras en las sucesivas versiones. En este proyecto es utilizada la SDK 1.7, lanzada en marzo de 2013.

A continuación, es descrita la arquitectura que relaciona a todos los componentes de la SDK, es analizada el funcionamiento de la NUI y las características que aporta esta SDK a nuevas aplicaciones.

### 3.2.2.1. Arquitectura SDK Kinect

La SDK de Kinect para Windows contiene drivers, herramientas, interfaces del dispositivo y códigos ejemplo para simplificar el desarrollo de nuevas aplicaciones, cuya arquitectura es mostrada en la figura 6.

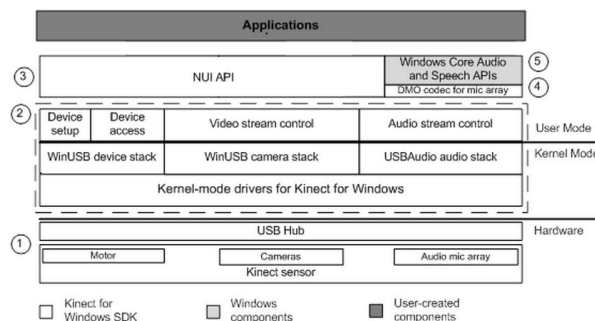


Figura 5: Arquitectura de la SDK de Kinect para Windows

La SDK incluye los siguientes componentes:

- Los componentes de hardware, incluyendo el sensor de Kinect y el concentrador USB a través del cual el sensor Kinect está conectado al computador.
- Los drivers de Kinect para Windows, que se instalan como parte del proceso de instalación de la SDK y que son compatibles con:

1. El conjunto de micrófonos Kinect como un dispositivo de audio, en modo de núcleo (kernel-mode) al que se puede acceder a través de las API de audio estándar de Windows.
2. Controles de transmisión de audio y video (color, profundidad, y el esqueleto).
3. Funciones de enumeración de dispositivos que permitan a una aplicación utilizar más de un sensor Kinect

- La Interfaz Natural de Usuario de Kinect, NUI API, para el seguimiento de esqueleto, de audio, y la captura de imágenes de color y profundidad.



- DirectX Media Object (DMO) para la formación de haces con el conjunto de micrófonos y la localización de la fuente de audio.
- Aplicaciones, APIs estándar para Windows 7 de audio, habla y APIs multimedia. Estas APIs también están disponibles para Windows 8.

#### 3.2.2.2. NUI (Natural User Interface)

La NUI es el núcleo de la SDK de Kinect para Windows. A través de ella se puede acceder a los siguientes datos de audio, color y profundidad del sensor en la aplicación.

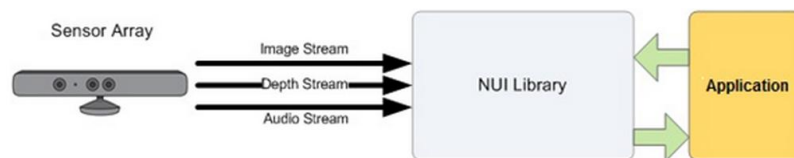


Figura 6: Hardware y software de Kinect para la interacción con una aplicación

La Kinect y la biblioteca de software interactúan con la aplicación, como se muestra en la Figura 1. El sensor proporciona los datos a la aplicación en forma de un flujo de datos y la NUI API permite mediante programación controlar y acceder a estos flujos.

Además de las capacidades de hardware, el software programado en el runtime de Kinect implementa una rutina que puede reconocer y realizar un seguimiento de un cuerpo humano, la integración con la API de Microsoft Speech para implementar un motor de reconocimiento de voz y una estrecha integración con la SDK de Face Tracking, lo que hace posible el seguimiento de rostros humanos.

#### 3.2.2.3. Posibles aplicaciones de la SDK

Gracias a esta biblioteca de software es posible desarrollar aplicaciones con las siguientes características:

- Entendimiento humano: La SDK de Kinect para Windows tiene un profundo entendimiento de las características humanas, incluyendo seguimiento facial y de esqueleto, y el reconocimiento de gestos y del habla.
- Ajuste de los datos para cada aplicación: La SDK proporciona herramientas efectivas para optimizar el uso de los datos, pudiendo controlar los ajustes de la cámara de color personalizándolos, para tener una aplicación refinada.
- Flexibilidad del hardware: La SDK de Kinect para Windows soporta hasta cuatro sensores en un computador, y puede ser usada en cualquier máquina virtual cuyo sistema operativo nativo pueda funcionar con Windows.

### 3.2.3. PCL

Para procesar los tipos de datos anteriormente descritos existen librerías que ofrecen los algoritmos apropiados. Para procesar las nubes de puntos empleamos algunos métodos de la librería PCL, Point Cloud Library.

La librería PCL es un proyecto abierto y de gran escala dedicado al procesamiento de nubes de puntos. Esta librería contiene una gran cantidad de algoritmos muy empleados en las aplicaciones que trabajan con nubes de puntos, como algoritmos de filtrado, de estimación de características, reconstrucción de superficies, registro, aproximación a un modelo o segmentación.

PCL es distribuido bajo los términos de la licencia BSD y es un software de código abierto. Es gratuito para el uso comercial y la investigación. Se trata de un proyecto desarrollado por un gran número de ingenieros y científicos de diferentes organizaciones, distribuidas geográficamente por todo el mundo. El proyecto es apoyado económicamente por un gran número de empresas y universidades. En 2011 el Ministerio de Conocimiento y Economía de Corea del Sur concedió a PCL el Primer Premio en el Open Source Software World Challenge.

PCL es multiplataforma, y ha sido compilado y desplegado en Linux, MacOS, Windows y Android con éxito.

Para simplificar el desarrollo, PCL se divide en una serie de bibliotecas de código más pequeñas, programadas en el lenguaje C++ y que pueden ser compiladas independientemente. Esta modularidad es importante para la distribución de PCL en plataformas con limitaciones computacionales o de tamaño reducido. A continuación, se muestra el conjunto más importante de módulos de PCL:

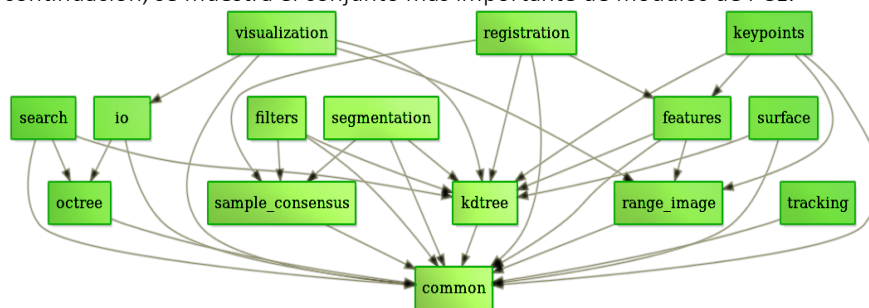


Figura 7: Módulos de PCL

PCL trabaja con nubes de puntos, utilizadas comúnmente para representar las coordenadas geométricas de los puntos muestreados en una superficie. Estas nubes de puntos pueden ser adquiridas con diferentes tipos de sensores tales como cámaras estéreo, escáneres 3D, cámaras de tiempo de vuelo, o generadas por un programa informático. PCL admite las interfaces OpenNI 3D, por lo que puede adquirir y procesar datos de dispositivos tales como las cámaras PrimeSensor 3D, el sensor Kinect Xbox 360 de Microsoft, o el Asus Xtion PRO, a través de su módulo para la entrada y salida de nubes de puntos, "io".

La versión de estas librerías empleada en esta aplicación es PCL 1.6.0.

El tipo de datos básico en PCL es una PointCloud. Una PointCloud es una clase de C++ que contiene los siguientes campos de datos:

- Width. Es un número entero que especifica el ancho del conjunto de datos de la nube de puntos.
- Height. Es un número entero que especifica la altura del conjunto de datos de la nube de puntos.
- Points. Contiene el array de datos donde son almacenados todos los puntos.
- Is\_dense. Es un parámetro booleano que especifica si todos los datos en points son finitos (true), o si los elementos de ciertos puntos podrían contener valores Inf/NaN (false).
- Sensor\_origin. Se trata de un vector que especifica la posición de adquisición del sensor. Este campo es normalmente opcional, y no es usado por la mayoría de los algoritmos de PCL.



- Sensor\_orientation. Se trata de un quaternion que indica la orientación del sensor. Este campo es normalmente opcional, y no es usado por la mayoría de los algoritmos de PCL.
- PointT es el tipo de dato de punto y describe los elementos que contenidos en cada punto o dato individual de la nube. PCL incorpora una gran variedad de diferentes tipos de puntos.

#### 3.2.3.1. Tipos de Datos

Utilizar estas librerías, nos da la ventaja de no depender de la iluminación, ya que nuestro análisis será con objetos y no con colores.

Los objetos de nuestro análisis serán las nubes de puntos. Una nube de puntos es una estructura de datos utilizada para representar una colección de puntos multidimensionales. Se trata de una sucesión o conjunto de vectores, cada uno de los cuales representa a un punto. Dicho vector puede contener diferentes elementos, desde las coordenadas tridimensionales del punto al que representa, hasta su color, o incluso el valor de otro tipo de propiedades que definen a dicho punto dentro de la escena.

En nuestro proyecto, se ha utilizado el color por la representación visual que hacemos del mismo, y ver de forma más clara que pasa en el propio desarrollo.

#### 3.2.4. Lenguaje y entorno de programación

El lenguaje de programación elegido para codificar los módulos de la aplicación es C++, empleado a demás por las librerías anteriormente descritas.

La aplicación es desarrollada en el sistema operativo Windows, para el cual está diseñado el sensor utilizado, Kinect para Windows. Concretamente el sistema operativo empleado es Windows 7 y el entorno de programación elegido es Microsoft Visual Studio 2010, que permite el trabajo en Windows con códigos programados en C++.

A continuación, son analizados tanto el lenguaje como el entorno de programación empleados.

##### 3.2.4.1. C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. Fue creado con la intención de ampliar el lenguaje de programación C con mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido. C++ es un lenguaje multiparadigma, para programación estructurada y programación orientada a objetos, muy útil en aplicaciones constituidas por varios módulos, o por varias clases.



#### 3.2.4.1.1. El concepto de clase

Los objetos en C++ son abstraídos mediante una clase. Según el paradigma de la programación orientada a objetos un objeto consta de:

- Identidad, que lo diferencia de otros objetos (Nombre que llevara la clase a la que pertenece dicho objeto).
- Métodos o funciones miembro
- Atributos o variables miembro

#### 3.2.4.1.2. Bibliotecas de C++

Los lenguajes de programación suelen tener una serie de bibliotecas de funciones integradas para la manipulación de datos a nivel más básico. En C++, además de poder usar las bibliotecas de C, puede usar la nativa STL (Standard Template Library), propia del lenguaje. Proporciona una serie de plantillas (templates) que permiten efectuar operaciones de almacén de datos y procesos de entrada/salida. Las clases *basic\_ostream* y *basic\_stream*, proporcionan la entrada y salida estándar de datos (teclado/pantalla), para la comunicación entre la aplicación y el usuario.

#### 3.2.4.1.3. Compiladores de C++

Uno de los compiladores libres de C++ es el de GNU, el compilador G++ (parte del proyecto GCC, que engloba varios compiladores para distintos lenguajes). Otros compiladores comunes son Intel C++ Compiler, el compilador de Xcode, el compilador de Borland C++, el compilador de CodeWarrior C++, el compilador g++ de Cygwin, el compilador g++ de MinGW, el compilador de Visual C++, Carbide.c++, entre otros.

#### 3.2.4.1.4. Entornos de desarrollo para C++

Es posible codificar aplicaciones en C++ en distintos entornos de desarrollo, en función del sistema operativo empleado:

- Para Microsoft Windows. Code::Blocks, Dev-C++, Visual C++, wxDev-C++
- Para MacOS. Xcode
- Para DOS. Turbo C, reemplazado por C++Builder
- Para GNU/Linux. Code::Blocks, NetBeans, Eclipse, Geany



#### 3.2.4.2. Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, Integrated Development Environment) de Microsoft para sistemas operativos Windows.

Visual Studio utiliza plataformas de desarrollo de software de Microsoft, como la API de Windows, Windows Forms, Windows Presentation Foundation, Windows Store y Microsoft Silverlight. Puede producir tanto código nativo como código administrado.

Visual Studio incluye un editor de código de soporte de IntelliSense, así como la refactorización de código. El depurador integrado funciona tanto como un depurador a nivel de fuente como un depurador a nivel de máquina. Otras herramientas integradas son, por ejemplo, un diseñador de formularios para la creación de aplicaciones con interfaces gráficas de usuario, y un diseñador de páginas web. Admite plug-ins que mejoran la funcionalidad en casi todos los niveles, incluyendo la adición de soportes para el control de código fuente (como Subversion y Visual SourceSafe) y la adición de nuevos conjuntos de herramientas como editores y diseñadores visuales para lenguajes específicos o conjuntos de herramientas para otros aspectos del desarrollo de software (como el Team Foundation Server cliente: Team Explorer).

Visual Studio es compatible con múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django, et., a lo cual hay que añadir sus capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos o consolas (xbox 360, xbox one).

También existen versiones de Visual Studio individuales específicos para cada lenguaje, como Microsoft Visual Basic, Visual J #, Visual C # y Visual C + +.

Microsoft ofrece ediciones "Express" de su Visual Studio sin costo alguno. Las versiones comerciales de Visual Studio, junto con ciertas versiones anteriores están disponibles de forma gratuita a los estudiantes a través del programa Microsoft DreamSpark.

### 3.3. Módulos de la aplicación

Una vez analizado los diferentes elementos que hemos utilizado, tanto de hardware como de software, vamos a proceder a ver una vista general de los diferentes puntos de la aplicación.

Los módulos en los que hemos dividido la aplicación son los siguientes:

-Calibración: Este módulo consiste en encontrar la relación entre las nubes de puntos tomadas desde los diferentes sensores. En esta parte, recibimos las diferentes nubes de puntos tomadas con la diana geométrica y calculamos la matriz de transformación para poder ser aplicada en la creación del mapa completo y en la vigilancia de la propia escena.

Consta de diferentes partes las cuales vamos a nombrar, pero que más tarde se analizaran de forma detenida.

- Adquisición de nubes de puntos con diana geométrica
- Filtrado
- Segmentación
- Calculo de la matriz de transformación

-Detección de intrusos: Consiste en la comparación de la situación ideal con una real, para ello, hay una parte offline de modelado del entorno, y una parte de detección de cambios en tiempo real. Para la parte offline se deberán adquirir dos nubes de puntos nuevas, las cuales no contarán con una diana geométrica en sus escenas, una desde un sensor y otra desde el otro. Como en el módulo anterior se calculó la matriz de transformación, se podrá aplicar esta misma a una de las nubes de puntos para la posteriormente sumar a la otra nube. De esta forma ya tendremos el mapa offline, el cual nos servirá para comparar con la nube de puntos que tomaremos en tiempo real.

Las diferentes partes de este módulo son las siguientes:

- Adquisición de nubes de puntos sin diana geométrica
- Mapeado completo
- Representación en tiempo real

A continuación, se expone un esquema de lo que se ha comentado en este apartado.

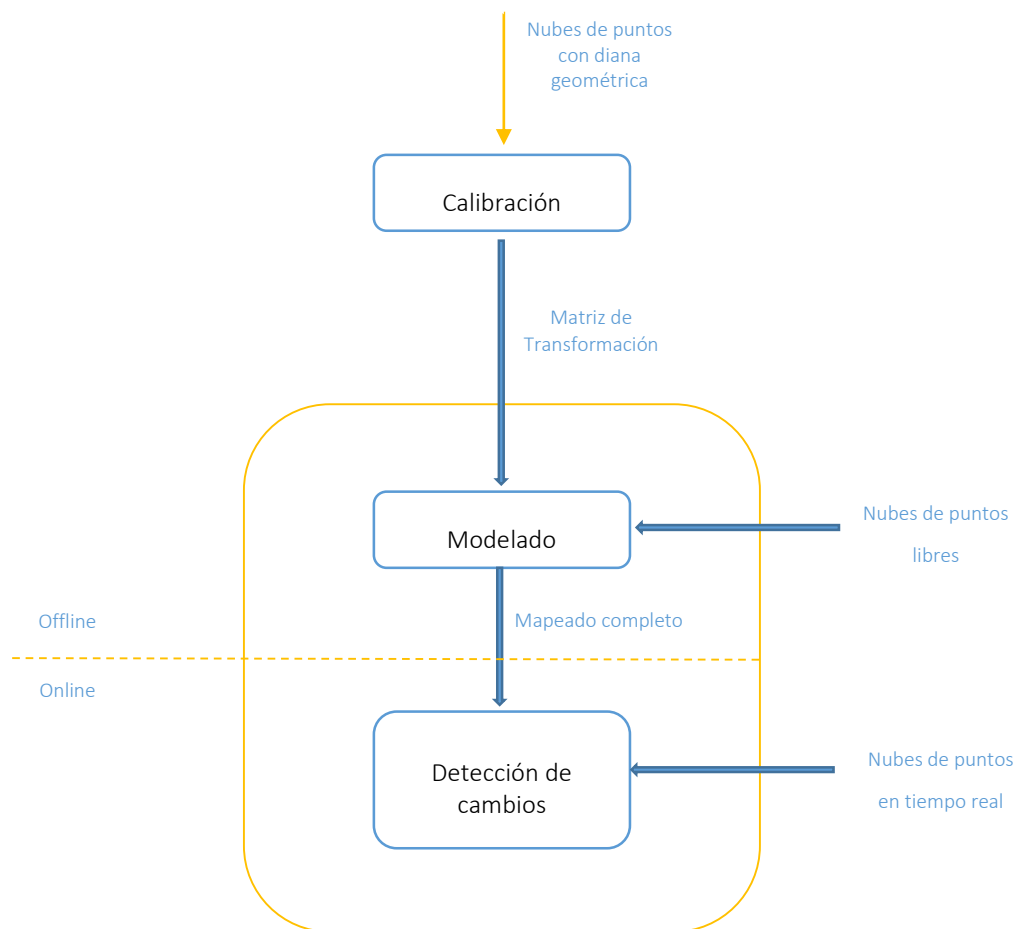


Figura 8: Esquema de los diferentes módulos del proyecto



Antes de entrar a explicar cada uno de los módulos del proyecto, vamos a explicar algunos conceptos y pasos anteriores que hemos dado para el desarrollo del mismo.

Para poder calibrar nuestros sensores, necesitamos en un principio, un conjunto de nubes de puntos que nos permitan encontrar la relación entre las dos posiciones.

Tendremos que preparar la escena, y adquirir las diferentes nubes de puntos de dichas escenas.

Tendremos diferentes tipos de nubes de puntos:

- Diana geométrica en la primera posición:

Esta nube de puntos contará con la diana geométrica en una primera posición. Dicha posición no será elegida a conciencia, si no que será cualquiera que esté dentro del campo de visión de ambos sensores.

- Diana geométrica en la segunda posición:

Será similar a la anterior nube de puntos, pero la diana geométrica se encontrará en una nueva posición, respecto de la anterior. Tienen que ser posiciones diferenciadas entre sí, es decir, que la nueva posición esté realmente diferenciada de la otra ya que, por medio de un análisis de error de precisión del sensor, se ha calculado que es de aproximadamente, unos 3 cm, como máximo. Para una mejor calibración del proyecto, se recomienda una distancia mínima de 30 cm.

- Libre:

Esta nube de puntos no contará con la diana geométrica dentro de sus datos tridimensionales, si no que será nuestra nube referencia de cómo tiene que estar la escena en el momento de la vigilancia, por eso mismo tendríamos que tomar los datos justo antes de la vigilancia, para asegurarnos que se parezcan lo máximo.

- Tiempo real:

Esta será la nube que se irá comparando con las nubes libres, ya en su forma de mapeado completo. La única diferencia entre la nube en tiempo real y la nube libre, tiene que ser el intruso.

La adquisición de los datos a partir de los sensores, lo hemos llevado a cabo gracias a un anterior proyecto de adquisición de nubes de puntos a partir del sensor Microsoft Kinect [2].

Se utiliza solo una parte del proyecto, la adquisición de nubes de puntos, la cual nos permite adquirir las diferentes nubes de puntos, tanto para el análisis, como la representación y la vigilancia.

En términos generales, este proyecto funcionaba de tal forma, que adquiría los datos de color y los datos de profundidad, juntándolos en una sola nube de puntos, creando de esa forma una representación de la realidad en forma tridimensional. En nuestro proyecto, se trabajará con estos datos, siendo el color innecesario, pero solo se utilizará con el objetivo de crear una representación más visual.



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES





## 4. Calibración

El objetivo de la calibración es llegar a calcular la matriz de transformación que representa la orientación y posición de un sensor con respecto al otro, la cual explicamos en el Anexo I, teniendo como entrada cuatro nubes de puntos, dos nubes tomadas desde cada uno de los sensores con la diana geométrica en la primera posición, y dos nubes tomadas desde cada uno de los sensores con la diana geométrica en la segunda posición.

### 4.1. Principios teóricos

Como se analiza en el Anexo I, la matriz de transformación, es un instrumento matemático que se utiliza para cambiar la posición relativa de un punto en el espacio con respecto a un origen de coordenadas, a otro origen de coordenadas.

Esta será la base de nuestra calibración, poder encontrar esa matriz para que podamos utilizarla, con el objetivo, de representar todos los puntos referenciados a un sistema de coordenadas, en otro sistema.

De esta forma podremos juntar los puntos de ambas nubes de puntos, en una sola.

La ecuación que se muestra a continuación, será la que nos ayude a comprender como funciona esta matriz en el presente proyecto.

$$A_1 = T \times A_2 \quad (1)$$

$A_2$  es un punto tomado desde el segundo sensor, y  $A_1$  es un punto tomado desde el primer sensor, siendo estos puntos, el mismo punto en el espacio real, es decir,  $A$ . De esta forma, sabremos la posición relativa del punto a uno de los sensores, si conocemos la matriz de transformación y la posición relativa de ese punto al otro sensor.

Pero para entender este proyecto, se debe utilizar la ecuación de forma diferente, ya que, si conocemos la posición relativa de ese punto a uno de los sensores, y conocemos la posición relativa de ese mismo punto al otro sensor, podremos conocer la matriz de transformación. Siendo este, el objetivo de este módulo.



## 4.2. Etapas

-Adquisición de nubes de puntos mediante interfaz de ayuda al usuario: Colocamos la diana geométrica en el campo de visión de la cámara. Para realizar este proceso se ha diseñado un algoritmo que se encarga de representar, en tiempo real, lo que está capturando el Microsoft Kinect. Una vez que se puede apreciar que la esfera está en este campo de visión, este módulo permite capturar la nube mostrada presionando cualquier tecla.

-Filtrado nubes de puntos:

Se filtran las cuatro nubes de puntos sobre un eje, para mejorar la efectividad de procesos posteriores y reducir el tiempo empleado.

-Segmentación diana:

Se busca la diana geométrica en las nubes de puntos filtradas, adquiriendo los puntos centrales de las dianas encontradas.

-Calculo matriz de transformación:

Una vez tengamos los cuatro puntos antes hallados, se pasará a calcular la matriz de transformación.

A continuación, se representa el módulo completo, en el que se verán las diferentes partes descritas, viendo el proceso global de forma esquemática.

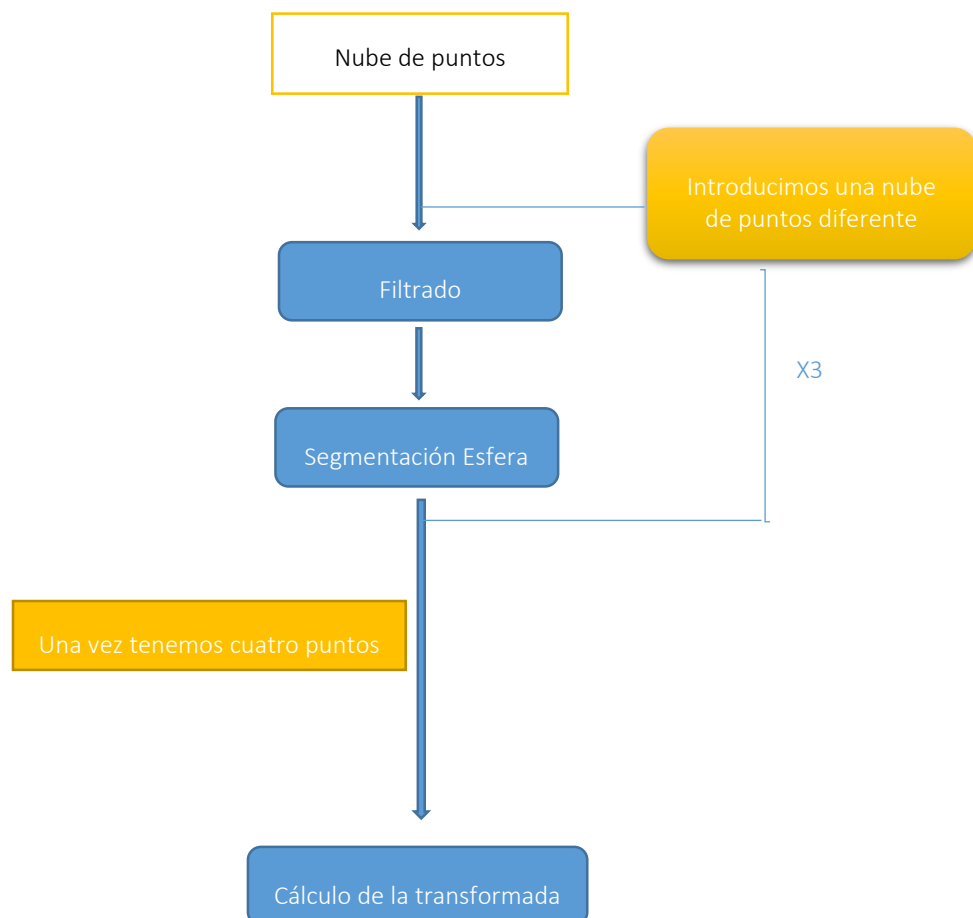


Figura 9: Esquema de calibración



### 4.3. Adquisición

Para calcular la matriz de transformación, se necesitan dos puntos en una situación real, tomados desde las dos posiciones, de esta forma, tendremos cuatro puntos de trabajo.

A continuación, se representa un esquema de cómo hemos llevado a cabo la adquisición de nubes de puntos, de esta forma tener una aclaración esquemática del proceso:

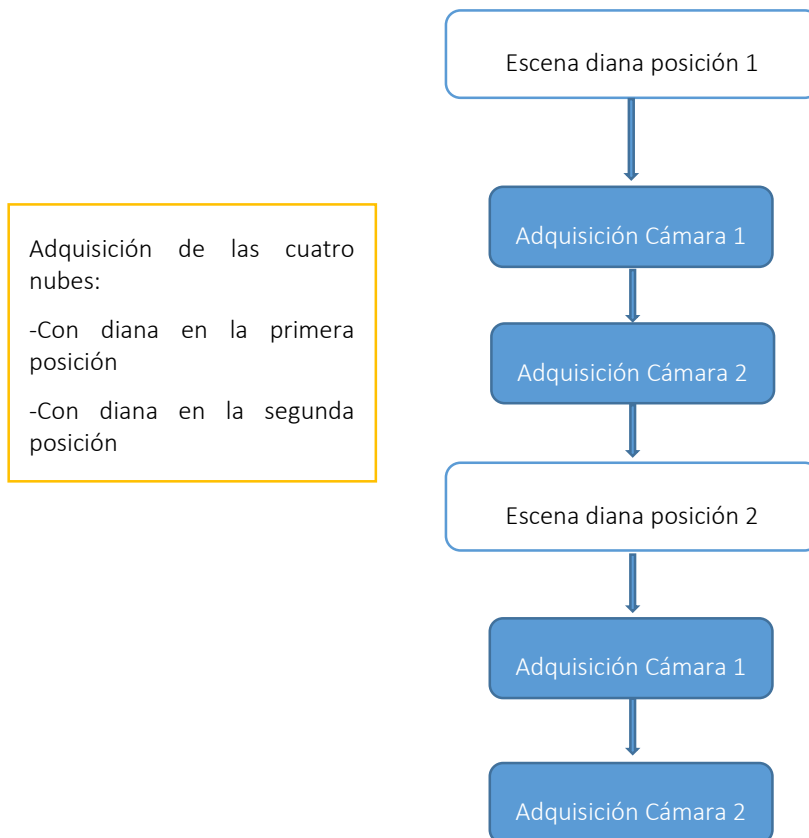


Figura 10: Esquema adquisición de todas las nubes

De esta forma, se habrán adquirido cuatro nubes de puntos, las cuales se avanzará a un análisis completo, el cual permitirá calcular la matriz de transformación.



#### 4.4. Filtrado de la nube

Con este proceso se facilitará la segmentación, para que no tenga que buscar entre tantos puntos, la propia diana geométrica.

Este proceso consiste en elegir el vector en el que filtra la nube de puntos, y colocando las dimensiones del filtrado.

De esta forma se recorta la propia nube de puntos, eliminando todos aquellos puntos que no interesen en el análisis.

Se ha programado para que la diana geométrica no tenga que estar en una determinada posición, si no que se pueda colocar en cualquier lugar, de esa forma una vez teniendo la nube de puntos con la diana dentro del campo de visión, podremos cortar con la distancia que se necesite, pudiendo repetir el proceso de tal forma que, si no encontramos esa diana en la nube filtrada, se pueda volver hacer el filtrado hasta que se encuentre en nuestra nube de puntos.

Una vez se haya filtrado la nube, con nuestra diana geométrica, se podrá pasar a la segmentación.

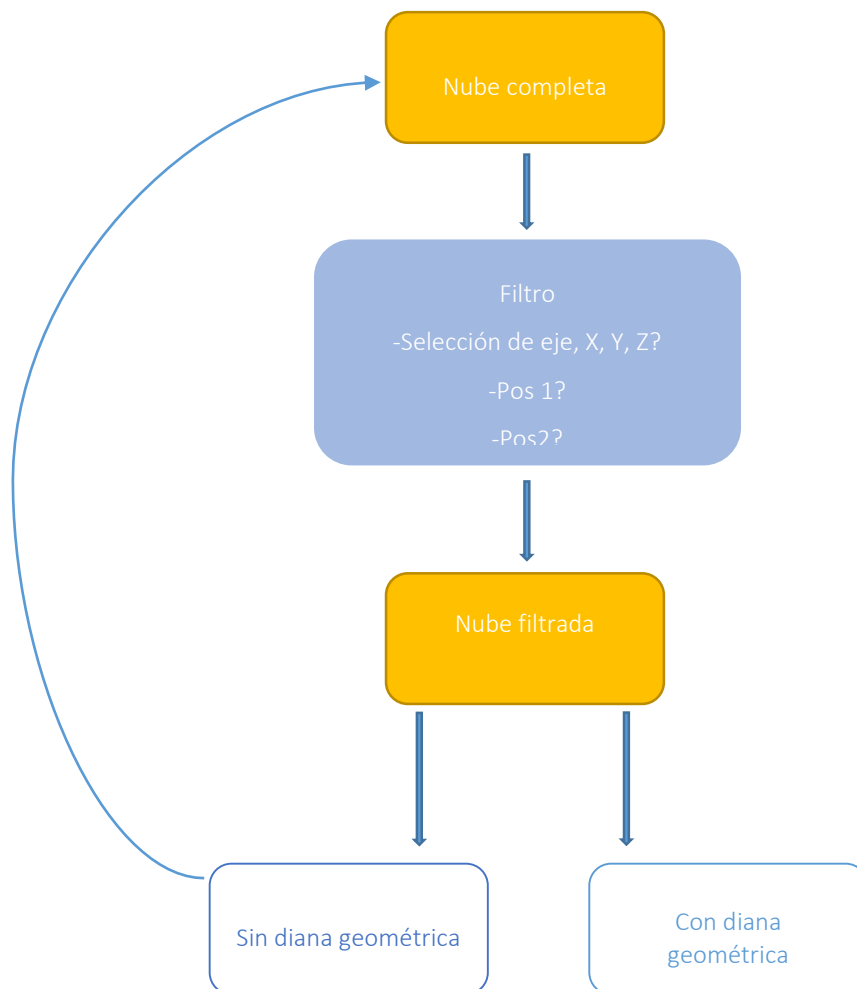


Figura 11: Esquema filtrado de nube

#### 4.4.1. Implementación

Las librerías PCL contienen una clase, PassThrough, la cual permite un proceso sencillo de filtrado con los sus diferentes métodos. Para llevar a cabo el filtrado de una nube de puntos, se debe seguir un proceso paso a paso, el cual se explica a continuación.

- Se introduce la nube de puntos la cual se procederá a filtrar.
- Se elige el eje en el que se recorta la nube.
- Se limita el filtrado, posición inicial y final del corte.
- Se introduce la nube de puntos en la cual, se guarda el resultado.

#### 4.4.2. Explicación gráfica del filtrado

En esta parte del módulo se explicará de forma más grafica todo el proceso.

En un principio, se tiene la nube de puntos la cual contiene la diana geométrica.

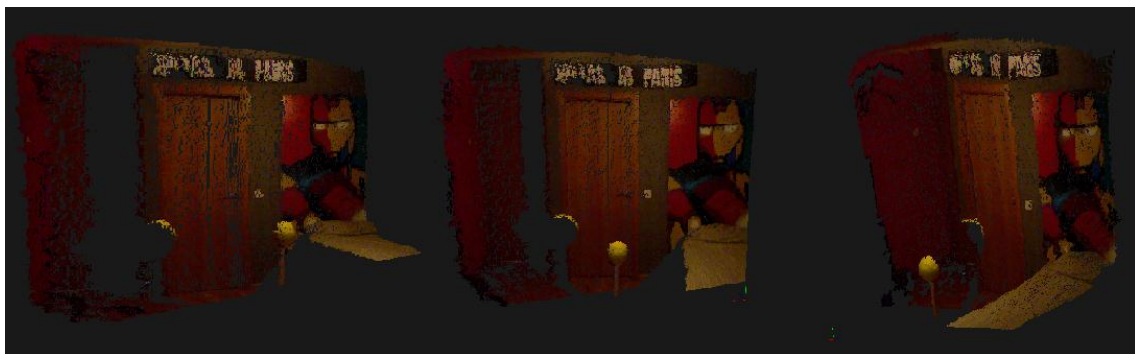


Figura 12: Nube a filtrar

Entonces se procede a filtrar, eligiendo primero en que vector se quiere recortar la nube, y después en que posiciones consideramos que se encuentra nuestra diana geométrica.

En la siguiente imagen se puede ver, como se puede no conseguir filtrar bien la nube, quedando la diana geométrica fuera.



Figura 13: Nube filtrada de 3 a 5 metros.

Como vemos la diana geométrica ha sido completamente eliminada. Lo que podemos ver detrás, una parte que pertenece a esta misma diana, es un error del propio sensor, que colorea partes pertenecientes de una superficie a otra a la cual no pertenece, debido a que el sensor de color y profundidad no están perfectamente calibrados.

La misma aplicación nos preguntará, si la diana se encuentra en la nube resultado del filtro, a esta misma la responderemos que no, para poder volver a filtrar con valores diferentes.

Nos vuelve a abrir la nube de puntos original, para que podamos tener una referencia más clara de la posición de la diana geométrica.

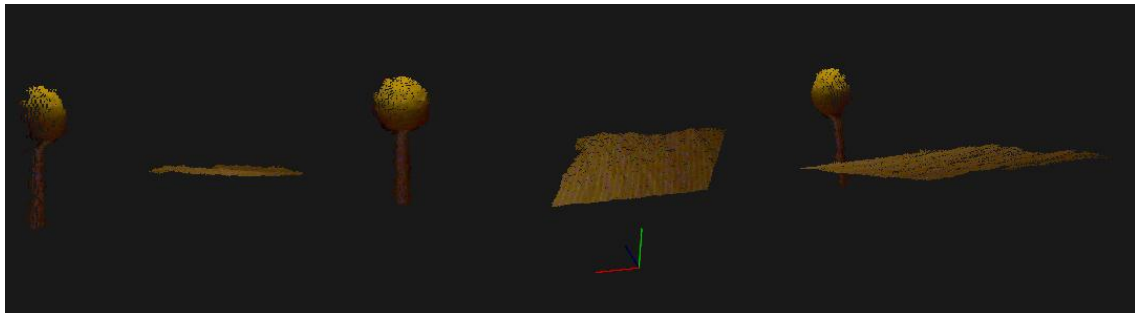


Figura 14: Nube filtrada de 1.2 a 2 metros.

Aquí se puede ver la nube ya filtrada, dentro de unos parámetros en la que la diana si se encuentra. Como se ha mencionado, se podría apurar más volviendo a repetir el proceso, probando valores diferentes que se ajusten más a la posición real de la diana. De esa forma, nuestra nube de puntos, tendría un número menor de posiciones que estudiar y le sería mucho más sencillo encontrar la diana.



## 4.5. Segmentación de la esfera

En este apartado se analizará cómo llevar a cabo la búsqueda de la diana geométrica. La elección de una esfera es sencillamente por la necesidad que tenemos en la continuación del proyecto.

Necesitamos tener una referencia para las diferentes posiciones de los sensores, en el mundo real.

La esfera como elemento geométrico, es la mejor opción, ya que tiene un punto central, así que por simplicidad ha sido la elección que hemos tomado. Otras opciones que permite el método de segmentación de las librerías PCL, como pueden ser los planos, podía llevarnos a problemas, ya que el plano es un objeto más común en el tipo de escenas que vamos a vigilar.

Otra opción como el cubo, tienen más puntos de interés como pueden ser sus vértices, pero hay que tener en cuenta, que cuando se hace una captura desde una posición, tendrías que orientar ese mismo cubo, de tal forma que los vértices estén en el campo de visión de ambos elementos, siendo estos, los mismos vértices.

El cilindro, por ejemplo, tampoco lo veíamos como una opción real, ya que te proporciona un eje pero no un punto determinado.

Por todas estas opciones descartadas, hemos optado por la esfera.

### 4.5.1. Etapas

Este proceso se divide en dos etapas, segmentación y extracción.

La segmentación es parte esencial del proyecto, mientras que la extracción la utilizaremos como una herramienta para la comprobación visual.

-Segmentación: devuelve cuatro datos de interés, los cuales tres primeros son las coordenadas del centro de la esfera y el cuarto el radio de la misma.

-Extracción: una vez conociendo ciertos valores de la segmentación, podemos extraer la esfera en una nueva nube de puntos.

Se ha creado un algoritmo que es capaz de que se pueda tener una comprobación para no arrastrar errores en procesos siguientes, creando un proceso muy similar al anterior, en el que se comprueba si es la diana geométrica, la cual ha sido adquirida, si es la que nos interesa, de esa forma poder repetir el proceso si es necesario. Aunque en todos los intentos, ha sido innecesario, ya que el filtrado provoca que haya muy pocos puntos, encontrando rápidamente el resultado.

Entonces, a la entrada de este módulo tendremos cuatro nubes filtradas y a la salida cuatro puntos, dos de un sistema de coordenadas y dos del otro sistema, siendo solo dos puntos en el espacio real.

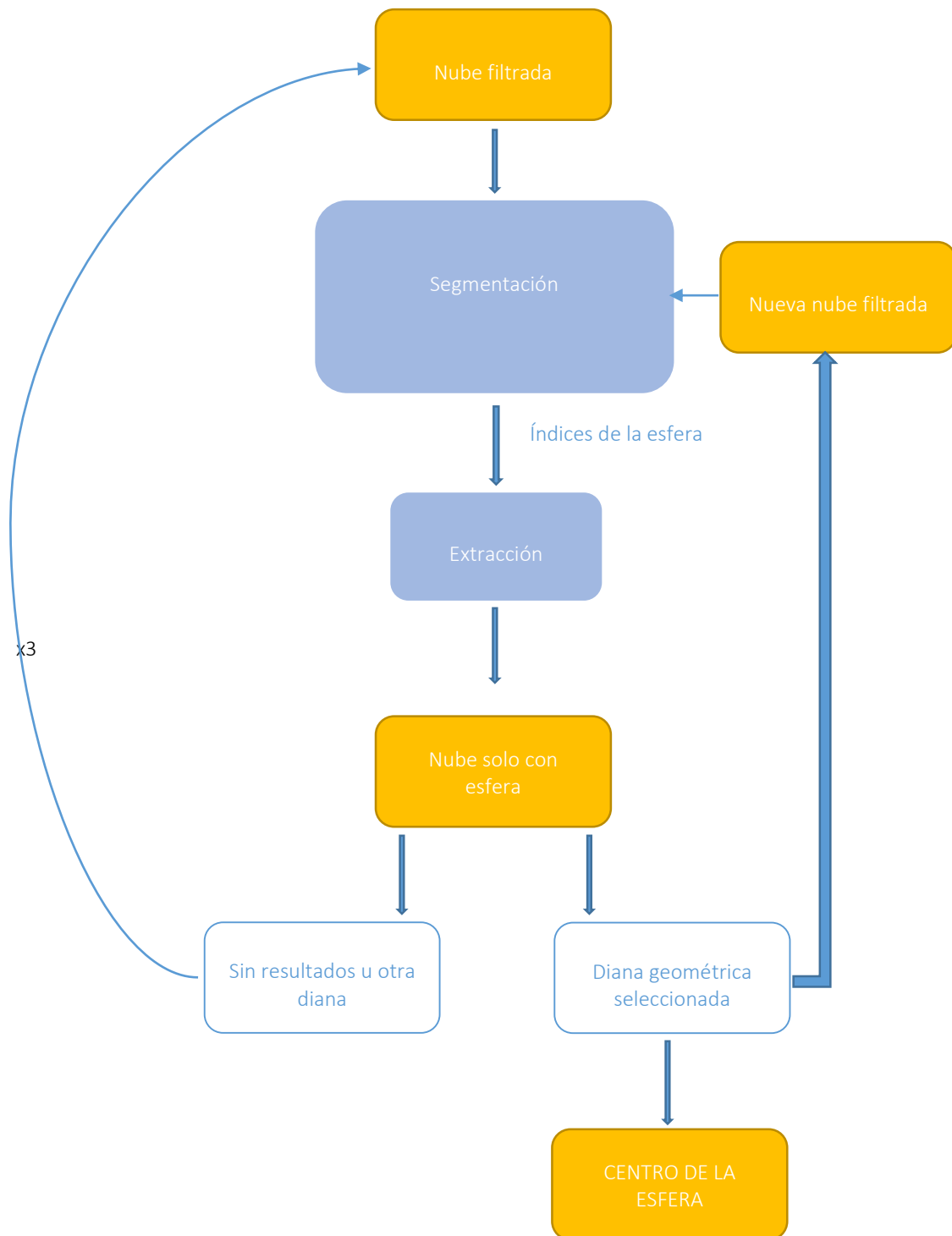


Figura 15: Segmentación de la esfera





#### 4.5.2. Implementación

Como se ha mencionado en el presente documento, solo con la parte de segmentación valdría, ya que es en esta parte en la que recogemos los datos de la esfera encontrada.

La segunda parte se implementa con el objetivo de tener una comprobación visual del resultado. Es decir, una vez se sabe dónde está la esfera, se pasa a extraer esa misma y guardarla en una nube diferente, de esa forma, al ver la nueva nube, se podrá observar que es la esfera que queremos, de la cual hemos extraído los datos correctos, que utilizaremos más adelante.

Las librerías Point Cloud Library, contienen diferentes clases, las cuales tienen diferentes métodos que permiten llevar a cabo el proceso.

**Segmentación.** Se utiliza la clase SACSegmentation. Creando un objeto de esta clase, se puede acceder a los diferentes métodos que permite hacer una segmentación, es decir, encontrar un determinado objeto.

Esta clase, permite elegir el modelo geométrico, el método que vamos a utilizar y el rango del radio dentro del cual va a estar la esfera. Y devolverá cuatro coeficientes del modelo elegido, que son las tres posiciones en x, y, z de su centro y el cuarto coeficiente que es el tamaño del radio, todos los datos los devuelve en metros.

**Extracción.** Se utiliza la clase ExtractIndices. Con los valores recibidos de los índices de la esfera segmentada en la anterior etapa, se puede extraer esa misma esfera, de la nube de puntos original. Se puede elegir si queremos extraer la esfera y guardarla en una nueva nube o tener la nube original sin la esfera.

Como hemos dicho anteriormente, la extracción para nuestra aplicación, solo nos sirve como una forma de comprobación visual, ya que se verá si es la esfera que hemos extraído, la que de verdad queríamos en un principio

Funciones empleadas:

```
setModelType (); // Se introduce el modelo que queremos encontrar
setMethodType (); // Se elige el método a utilizar, en nuestro caso eliminar la necesidad de utilizar normales
setDistanceThreshold (); // Se selecciona la distancia a partir de la cual se quiere buscar la esfera
setRadiusLimits(); // Se elige el rango del radio de la esfera que se quiere encontrar
setMaxIterations(); // Se decide el número de iteraciones máximas
setInputCloud (); // Se introduce la nube que se quiera analizar
segment (); // Se recoge los valores de los índices y los coeficientes
```

//Comenzamos la extracción

```
setInputCloud (); // Se selecciona la nube de la cual queremos extraer
setIndices (); // Se introduce los índices antes calculados
setNegative (); // Si se quiere eliminar la esfera o tener solo esfera
filter (); // Guardar el resultado en esta nube
```

#### 4.5.3. Explicación gráfica segmentación

En este apartado se representará gráficamente lo que hemos explicado tanto a nivel de concepto como a nivel técnico.

En un principio, en este módulo, se tendría ya la nube filtrada. Ya solo queda encontrar la esfera, de la cual su punto central, servirá de referencia para las diferentes posiciones de las cámaras.

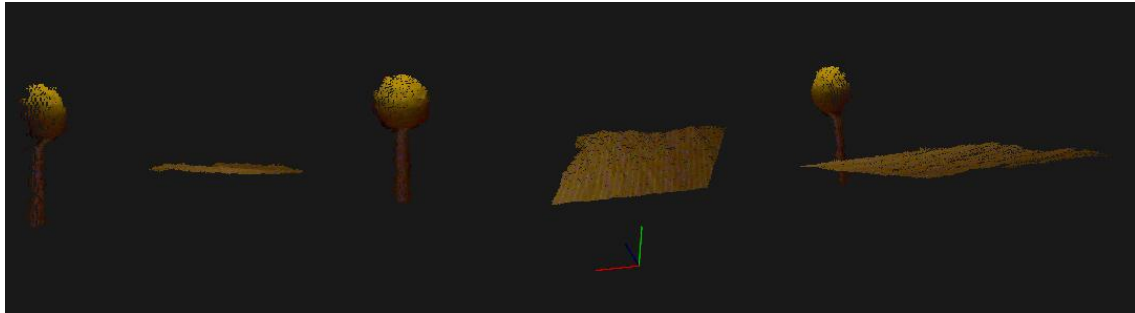


Figura 16: Nube anteriormente filtrada.

Como se puede apreciar en la imagen, esta es la nube anteriormente filtrada, a partir de la cual, se pasará a segmentar la esfera amarilla que se encuentra en esta misma nube.

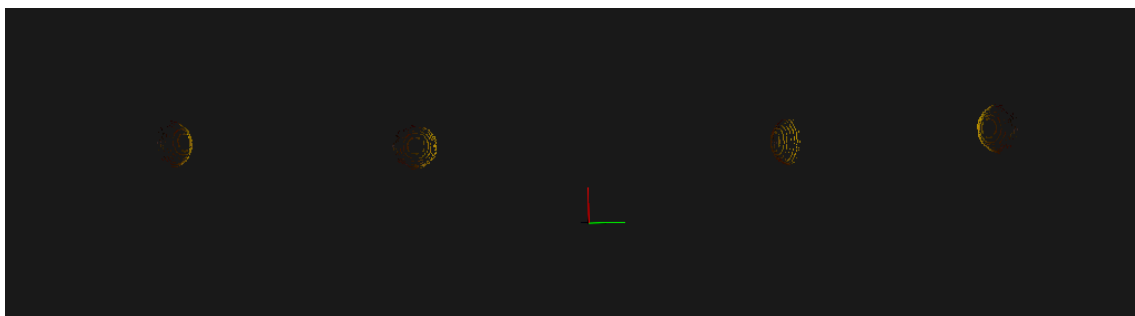


Figura 17: Nube con la esfera encontrada.

Como se explicó, esta parte sirve para hacer una comprobación visual y ver de verdad que esa es nuestra esfera.

A continuación, se pueden observar los resultados de los coeficientes calculados.

```
x es :0.338638  
y es :-0.380421  
z es :2.03907  
El radio es :0.115962
```

Figura 18: Resultado de la segmentación

Como se mencionó con anterioridad estos son los resultados que nos devuelve la segmentación, la posición del centro de la esfera y el tamaño del radio en metros.

Estos datos son necesarios para la siguiente parte de nuestro proyecto.



## 4.6. Cálculo de la transformada

Este módulo consiste en calcular la matriz de transformación (Anexo I) que hará que una de las nubes transforme su origen de coordenadas para poder ser incluida en la otra nube, tomada desde el otro sensor.

Los métodos de PCL te permiten sumar dos nubes y su forma de sumar estas dos nubes, es sumar los puntos poniendo el origen de coordenadas en la misma posición. Por ese mismo motivo, necesitamos cambiar toda la orientación de una de las nubes, para que se pueda sumar.

Solo se analizará cómo la hemos hallado, dejando la explicación de cómo la hemos implementado para el siguiente módulo, ya que forma parte del mapeado completo de la escena

### 4.6.1. Explicación

Para calcular esta matriz, primero se tuvo que ver cómo resolver el sistema de ecuaciones para después aplicarlo en forma de código, automatizando el proceso.

El concepto desde el que partimos es el siguiente:

$$A_1 = T \times A_2 \quad (2)$$

$$B_1 = T \times B_2 \quad (3)$$

Siendo  $T$  la matriz de transformación,  $A_1$  la posición del centro de la esfera en la primera posición desde el sensor 1,  $A_2$  la posición del centro de la esfera en la primera posición desde el sensor 2. Mientras que  $B_1$  es la posición del centro de la esfera en la segunda posición vista desde el sensor 1, y  $B_2$  es la posición desde ese mismo centro de la esfera desde el sensor 2.

Es decir, la matriz de transformación, multiplica el mismo punto desde otra referencia calculando así el mismo punto desde la otra referencia.

Lo mismo con la segunda ecuación, pero trabajando con la esfera en la segunda posición.

$$\begin{pmatrix} Ax_1 \\ Ay_1 \\ Az_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \alpha & 0 & \sin \alpha & x \\ 0 & 1 & 0 & y \\ -\sin \alpha & 0 & \cos \alpha & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} Ax_2 \\ Ay_2 \\ Az_2 \\ 1 \end{pmatrix} \quad (4)$$

Resolviendo de la siguiente forma:

$$Ax_1 = \cos \alpha x Ax_2 + \sin \alpha x Az_2 + x \quad (5)$$

$$Ay_1 = Ay_2 + y \quad (6)$$

$$Az_1 = -\sin \alpha x Ax_2 + \cos \alpha x Az_2 + z \quad (7)$$

Siendo  $Ax_1$  la componente x de  $A_1$ , lo mismo con y, y z. Con  $A_2$  se tiene la misma nomenclatura.

Como se puede comprobar, se obtiene tres ecuaciones, con cinco incógnitas, utilizando solo un punto, el  $A$ , con cada una de sus representaciones en cada uno de los sensores. Entonces se necesitan más ecuaciones para poder resolver las incógnitas. Como la matriz de transformación es igual de aplicable a un segundo punto, obteniendo el mismo, se tendrá lo necesario para resolver el sistema.



Nuestra función recibe cuatro puntos diferentes, es decir dos puntos en el espacio real, pero vistos desde dos perspectivas diferentes, el primer sensor y segundo sensor. Esta misma función utiliza números auxiliares que permiten hacer el proceso mucho más sencillo.

Esta misma función devuelve la matriz completa, lista para aplicar en el siguiente proceso.

A continuación, se expone un esquema del proceso del cálculo de la matriz de transformación

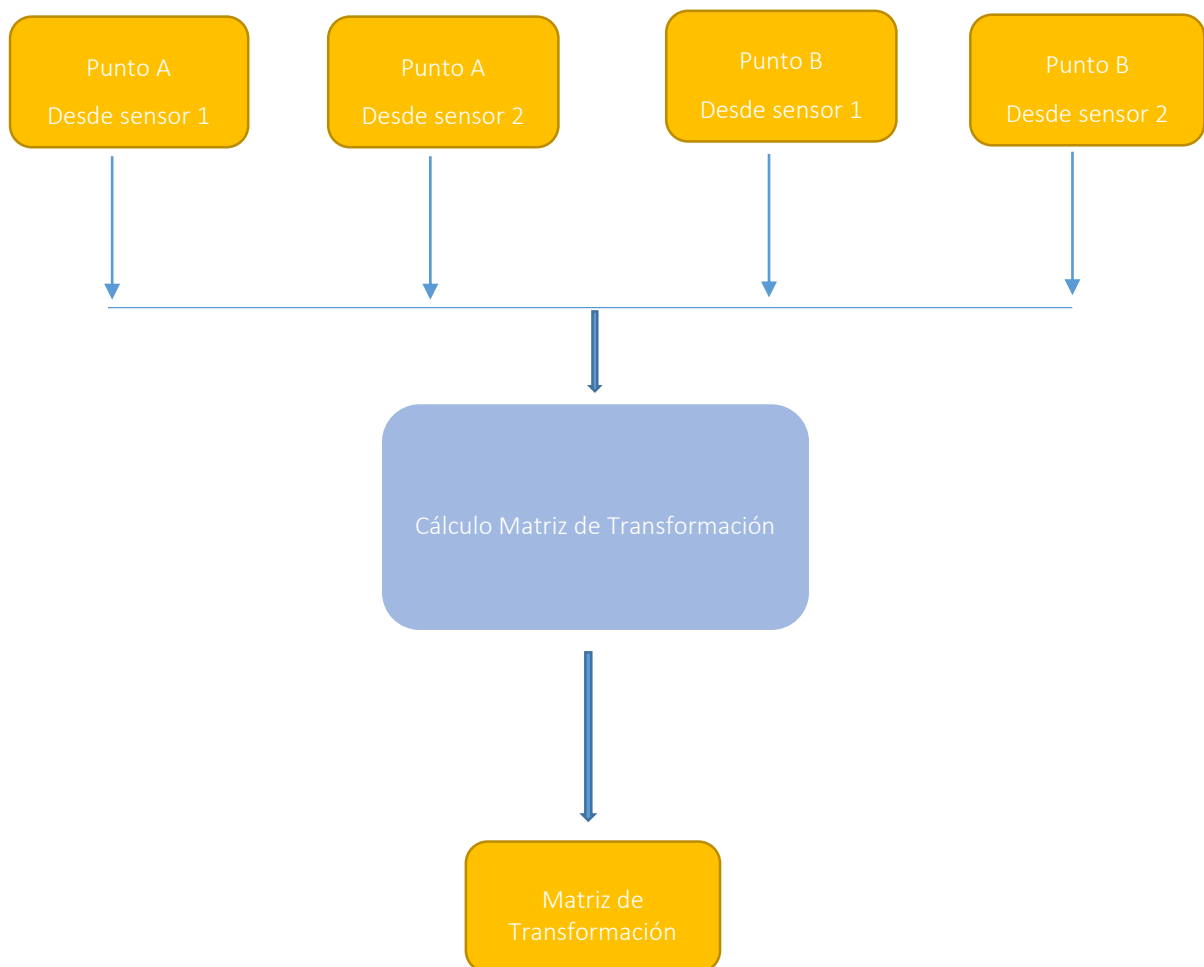


Figura 19: Esquema del cálculo de la matriz de transformación



## 5. Sistema de detección de intrusos

El sistema de detección de intrusos tiene como objetivo el cálculo del mapeado completo, para después poder comparar este mapeado creado, con un mapa tomado en tiempo real, de esta forma se detectarán los diferentes elementos que se considerarán intrusos.

En el módulo anterior se calculó la matriz de transformación, la cual en este módulo se utilizará en la creación del mapa completo y en la etapa de vigilancia, cuando utilicemos uno de los sensores.

Como resultado tendremos una escena vigilada, que detecta aquellos puntos que no coinciden en las dos escenas, la ideal y la real.

### 5.1. Principios teóricos

El motivo por el cual se necesita una matriz de transformación, es por la forma con la que las librerías PCL suma nubes de puntos. Coge los puntos de una nube, y los de otra, juntándolos en una misma, cogiendo uno de los orígenes de coordenadas de una de las nubes, siendo esta la referencia de la nube completa.

Por esto mismo, a la hora de vigilar la escena, si se vigila desde el sensor que se ha utilizado como origen absoluto, no hará falta utilizar la matriz de transformación en la propia vigilancia, mientras que, si se vigila desde el segundo sensor, se tendrá que multiplicar por la matriz de transformación la nube de puntos que vaya adquiriendo en tiempo real, de esta forma poder ser comparada con el mapeado completo, el cual ha sido creado a partir de esa matriz.



## 5.2. Etapas

Este proceso contiene tres etapas diferenciadas entre sí, las cuales se explicarán brevemente, para después pasar a un análisis más completo de las mismas.

### -Adquisición nuevas nubes de puntos:

Anteriormente se tomaron diferentes nubes de puntos, desde posiciones diferentes, con una diana geométrica que cambiaba de posición. Ahora se necesitan nuevas nubes de puntos las cuales permitan, crear una representación real de la esfera.

Estas nubes de puntos deben de ser ideales en nuestra situación, es decir, tendrá que ser una escena la cual servirá de referencia de cómo se tiene que mantener todos los elementos en el momento de la vigilancia.

El proceso es el mismo que se llevó a cabo con anterioridad, en la adquisición de nubes de puntos con diana geométrica colocada en la escena.

### -Modelado de la escena:

En este proceso se utilizará la matriz de transformación calculada en el módulo anterior, creando la nube de puntos la cual será nuestra referencia, y que se comparará con la nube de puntos tomada en tiempo real.

### -Vigilancia:

En esta parte se tomarán nuevas adquisiciones de datos tridimensionales, pero esta vez en tiempo real, lo cual es lo mismo que hicimos con anterioridad, cuando se tenía que tomar nubes de puntos con la diana geométrica, pero ahora en un bucle infinito, para ver qué sucede en la escena.

Estas nubes tomadas en tiempo real, se compararán con el mapa resultante del modelado de la escena, de esta forma se detectarán los diferentes elementos que no estaban presentes en el modelado, pero si en el momento en tiempo real.

El siguiente esquema representa las diferentes etapas del módulo del sistema de detección de intrusos.

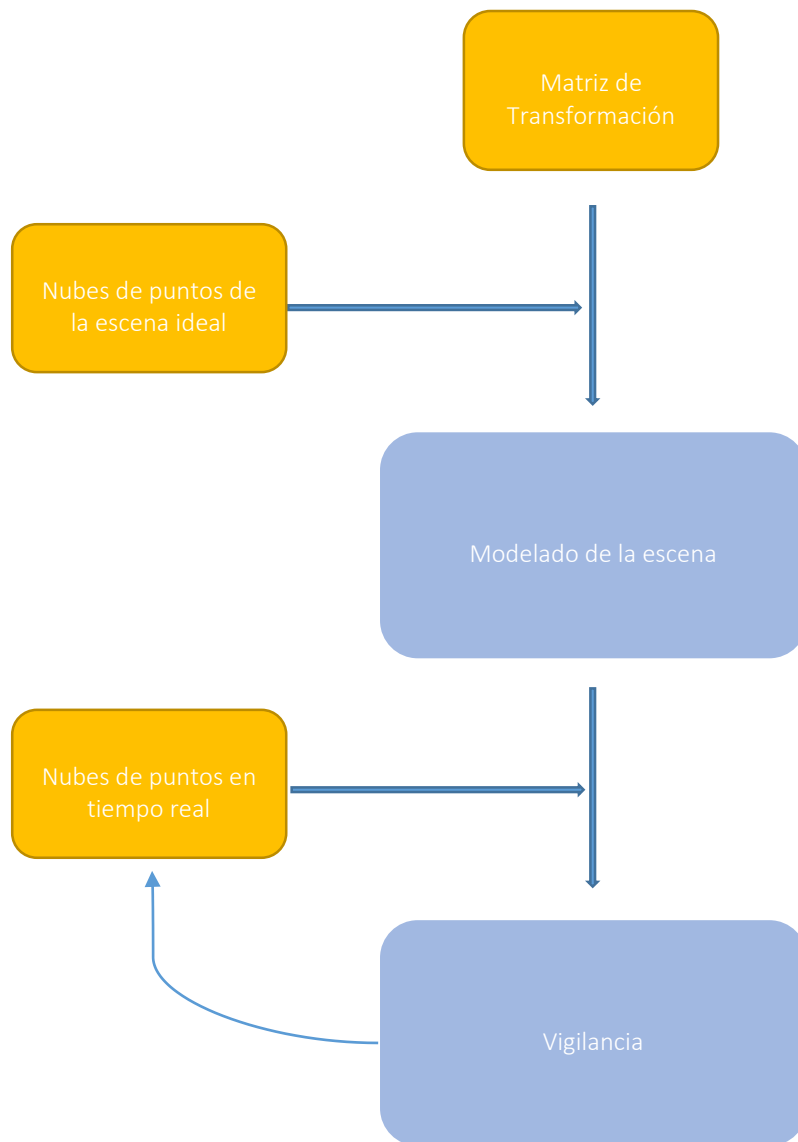


Figura 20: Esquema etapas detección de intrusos



### 5.3. Modelado de la escena

En este apartado se describe brevemente el proceso del mapeado completo, siguiendo la línea que se ha llevado hasta ahora en procesos anteriores.

Una vez calculada la matriz de transformación, ya solo queda aplicarla a nuestra nube de puntos. Esta nube transformada se sumará a la anterior, de forma que se construya una representación en forma de nube de puntos, la cual colocará cada campo de visión tomado por cada una de las cámaras, en su sitio correspondiente.

Por esto mismo, el proceso se divide en diferentes partes:

-Aplicación matriz de transformación:

Se multiplica la nube de puntos de la segunda posición por la matriz de transformación, haciendo que todos los puntos de esta, cambien de orientación, orientados de tal forma que permita el alineamiento correcto con la otra nube de puntos.

-Concatenación de nubes de puntos:

Una vez se tiene la nube de puntos desde la primera posición, y la nube de puntos desde la segunda ya transformada, se puede pasar a la concatenación de los puntos de estas mismas nubes en una sola. Realmente lo que hace este proceso, es juntar ambas nubes, colocando todo de tal forma, que coincidan los orígenes de coordenadas de ambas nubes de puntos, creando el mapa total de la escena, el cual será para nosotros el mapa ideal.



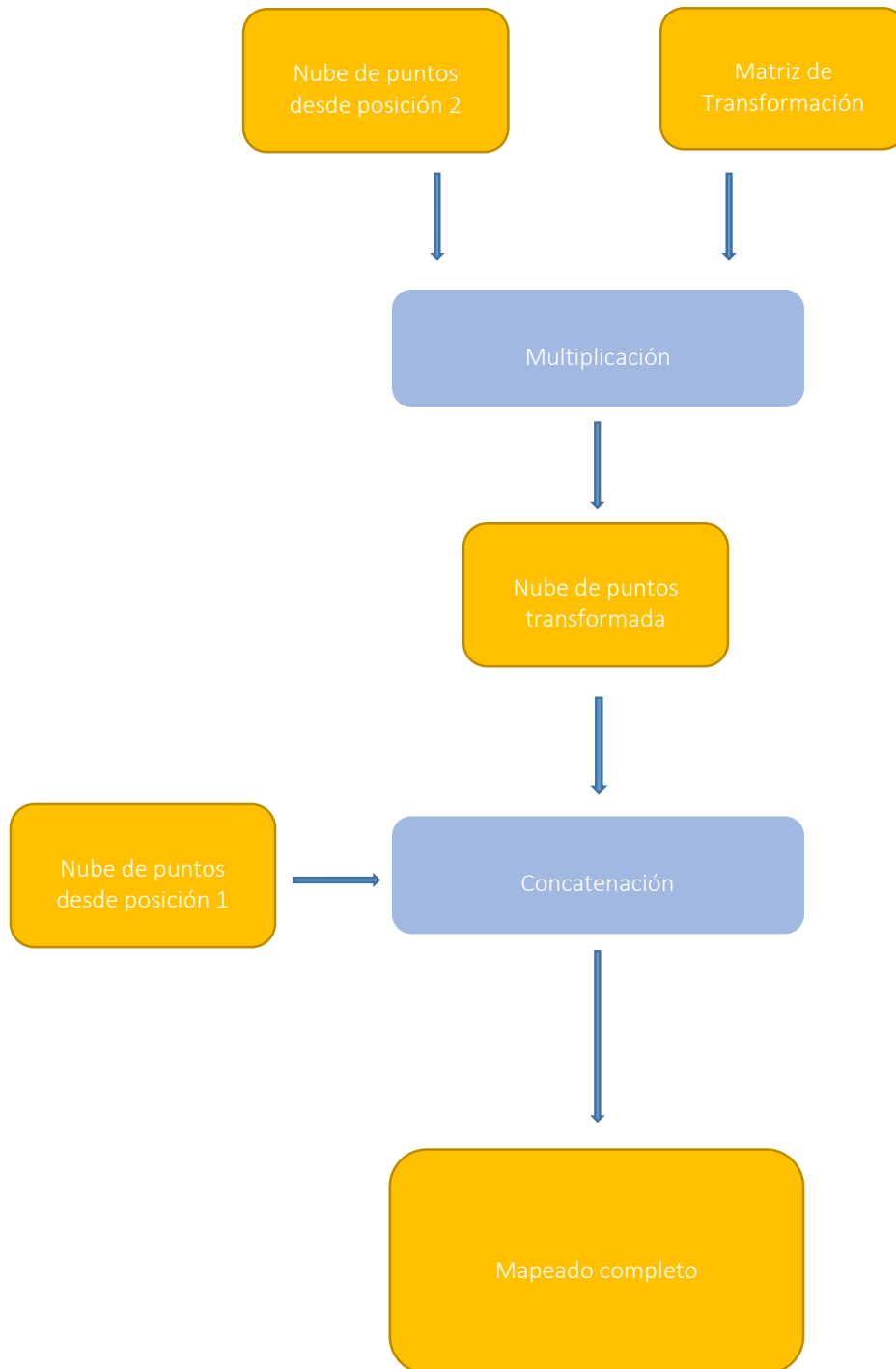


Figura 21: Esquema modelado de la escena

### 5.3.1. Implementación

Las librerías PCL, permiten la multiplicación de una nube de puntos por una matriz 4x4, la cual es nuestra matriz de transformación. El método que se utiliza para este proceso es `transformPointCloud()`, en el cual se tiene que introducir la nube que quieres transformar, la matriz que se quiere transformar y la nube de puntos resultante.

Una vez se tiene la nube de puntos transformada, se puede pasar a concatenar las nubes. Esta concatenación funciona de tal forma que se puede juntar los puntos de ambas nubes, en una sola, la cual se guardará y es nuestro mapeado completo.

PCL permite crear nubes a partir de los puntos de dos nubes diferentes, juntándolos en una sola, lo que lleva a crear el mapeado completo, ya que se han alineado las dos nubes, a partir de la multiplicación por la matriz de transformación.

### 5.3.2. Representación gráfica del proceso

En este punto, se tienen las dos nubes de puntos sin esfera, ya que hicimos una adquisición sin diana geométrica, desde las diferentes posiciones de los sensores.

Una vez calculada la matriz de transformación, se puede transformar la segunda nube de puntos, para conseguir un mapeado completo desde las dos posiciones.

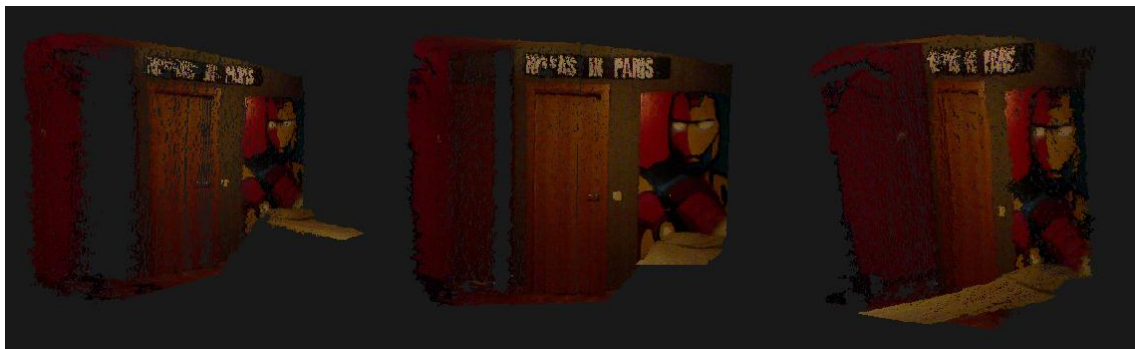


Figura 22: Nube de puntos posición frontal sin diana geométrica

Se puede observar la primera nube de puntos, la cual se sumará a la transformada de la segunda nube.

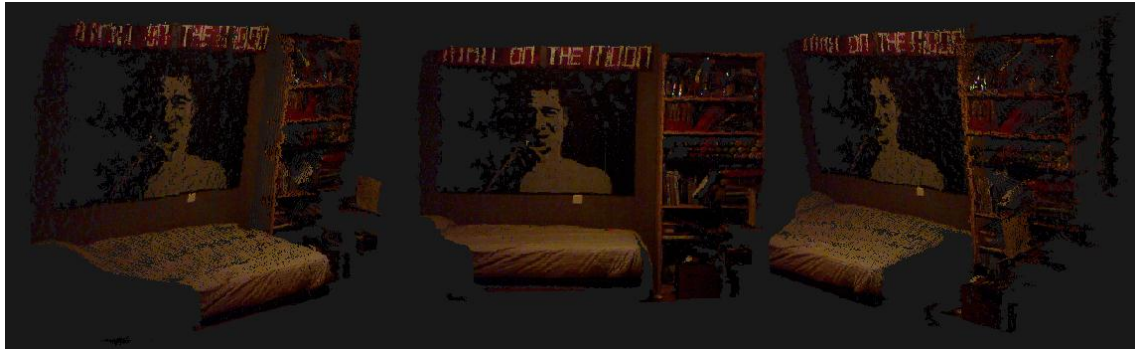


Figura 23: Nube de puntos posición derecha sin diana geométrica

Esta es la segunda nube de puntos, tomada desde la segunda posición. Será la que se transforme por medio de la matriz de transformación.

Una vez transformada, cambia su origen de coordenadas a la posición real del primer sensor, el cual es el origen de coordenadas de la primera nube de puntos, de esa forma, podrá ser añadida con una relación real a esa primera nube de puntos que no ha sido transformada, creando el mapeado completo.



Figura 24 A: Mapa completo sin matriz aplicada

Figura 24 B: Mapa completo con matriz aplicada

Como se puede apreciar en la nube resultante, tenemos ambas nubes unidas en una sola. El origen de coordenadas, que ha sido rodeado en verde, se puede apreciar que está en la posición de la primera nube. Por lo tanto, el proceso ha sido como hemos descrito, dejando el sistema de coordenadas en el



mismo lugar para la primera nube. Se transforma la segunda nube para hacer que el sistema de coordenadas esté en el mismo lugar “real” que el de la primera y sumamos ambas nubes.

A continuación, se pasará a la vigilancia de la escena, en la que el mapeado completo, juega un papel fundamental en el desarrollo del proyecto, ya que servirá para hacer una comparación con la nube tomada en tiempo real, lo que nos dará las diferencias encontradas entre las nubes.

## 5.4. Vigilancia

En esta parte se explica el proceso que se ha seguido para vigilar la escena.

Una vez se ha creado el mapa completo de la habitación podemos pasar a la vigilancia de la misma, siguiendo un proceso sencillo, que permita detectar los cambios vistos en la propia escena, de esta forma se podrá ver que elementos son nuevos dentro de la habitación, al igual que los elementos que ya no están.

-Adquisición de la nube en tiempo real:

Como se hizo antes, al capturar una nube para el posterior análisis, se tiene que tomar ahora los datos del sensor en tiempo real, haciendo lo mismo, pero sobrescribiendo la nube que se elija para tener una representación en el momento.

-Comparación con la escena:

Esta nube que se captura en tiempo real, se comparará por diferentes métodos que permite las librerías PCL, viendo de esta forma las diferencias existentes y representándolas con el mapa que se creó con anterioridad, y así se podrá detectar esos objetos no coincidentes.

A continuación, se representa un esquema que nos permita una comprensión esquemática del proceso.

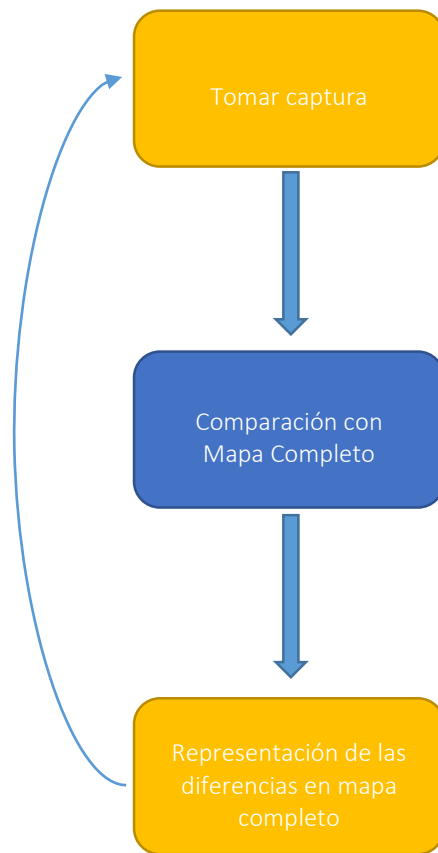


Figura 25: Esquema Vigilancia

#### 5.4.1. Implementación

Como se ha mencionado anteriormente, la adquisición en tiempo real de la escena, es simplemente la adquisición que se hizo con anterioridad, pero en un bucle infinito. Haciendo uso de una clase encargada de esta adquisición de un proyecto anterior.

Se tienen dos nubes dentro del bucle, la que se acaba de adquirir y la nube que se creó la cual contiene el mapeado completo, estas dos serán las nubes que se compararán, para ver qué puntos son diferentes y detectar esos cambios.

PCL contiene un método encargado de la comparación entre nubes, el cual está contenido en su módulo Octree, el cual se explica en el Anexo II. Nosotros utilizaremos la clase **OctreePointCloudChangeDetector**.

El proceso que se lleva a cabo es muy sencillo, ya que se trata de introducir las nubes de forma consecutiva, para sacar los valores que definen los puntos de diferencia entre ambas nubes introducidas.

Estos valores se pueden transformar en puntos para poder ser representados. En nuestro caso la representación se ha hecho encima de nuestro mapa completo, con el objetivo de que sea mucho más visual el proceso.

`setInputCloud ()` //Se introduce la primera nube



```
addPointsFromInputCloud () //Se añaden los puntos de esa misma nube  
switchBuffers () //Se cambian los buffers para introducir otra nube  
setInputCloud () //Se introduce otra nube para la comparación entre ambas
```

```
addPointsFromInputCloud ()//Se añaden todos los puntos  
getPointIndicesFromNewVoxels ()//Se recogen los indices de los puntos que no  
//coinciden
```

Este proceso se repetiría de forma infinita, de esa forma se podría hacer una vigilancia en tiempo real de la escena, comparando el mapeado completo con la imagen adquirida en el momento, siguiendo el proceso que acabamos de describir.

Octree es necesario, ya que divide el espacio en rejillas tridimensionales, siendo las divisiones cubos de un determinado tamaño, el cual se puede seleccionar, de manera que para ambas nubes sabe que cubos contienen puntos y cuales no y así es posible una comparación de acuerdo a la ocupación espacial de cada nube, ya que no se pueden comparar nubes de puntos simplemente restando esos mismos puntos, porque los puntos de ambas es imposible que ocupen exactamente el mismo espacio.

#### 5.4.2. Explicación grafica del proceso

Esta nube es la primera que se introduce en nuestro sistema de comparación y es exactamente la que se calculó a la hora de hacer el mapa completo de la habitación.



Figura 26: Mapa completo

Una vez que esta nube es cargada en nuestro sistema, hay que cargar la segunda nube la cual se toma en tiempo real.



Figura 27: Nube en tiempo real

El sistema antes planteado, devuelve unos puntos que detecta como no correspondientes a la primera nube, los cuales se pueden representar y son los siguientes:

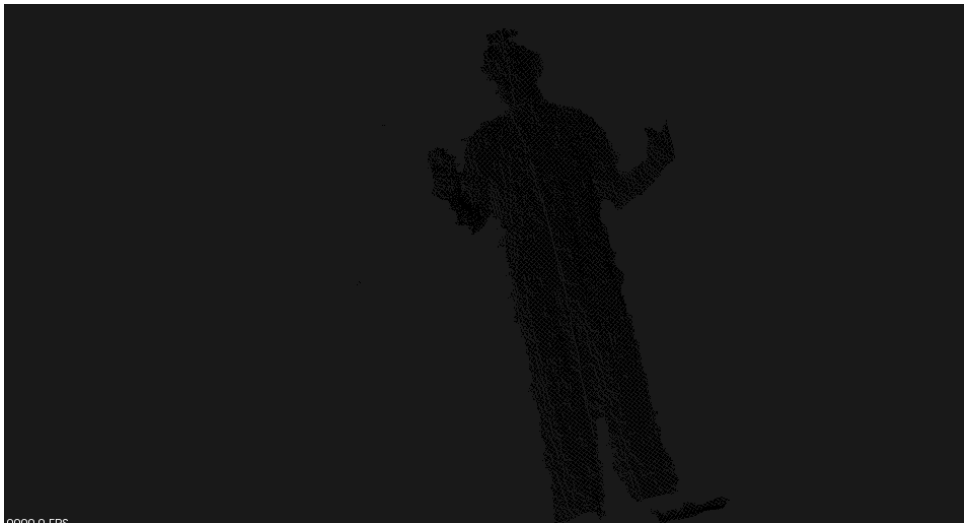


Figura 28: Puntos resultantes

Como se puede apreciar, estos puntos son negro sobre negro, a los cuales se tendrá que dar un color para que sea mucho más claro.

Estos mismos puntos, se han introducido en el mapa completo, para ver la ubicación exacta del intruso.





Figura 29: Vigilancia completa 1

Aquí se ven, ya una vez coloreados estos mismos puntos, el elemento que no estuvo en el momento de crear el mapa completo, el cual se considera nuestro mapa ideal, de cómo debe mantenerse la habitación en el momento de la vigilancia.



Figura 30: Vigilancia completa 2

En este caso, detecta que la puerta está abierta cuando debería estar cerrada, como en el momento del mapeado completo.





## 6. Resultados experimentales

### 6.1. Propiedades del sistema y objetivos cumplidos

Nuestro proyecto se ha llevado a cabo siguiendo los objetivos que se describieron en un principio, y como hemos visto, se han cumplido cada uno de ellos.

**Robustez.** Esta aplicación demuestra robustez, ya que utilizando un sensor 2.5D como es el sensor Microsoft Kinect, que presenta poca precisión en la captura, se han conseguido resultados satisfactorios. Este tipo de sensores arrojan unos datos no tan precisos como pueden ser los sensores 3D, con los cuales se tendría un resultado más fiable, pero gracias a las diferentes herramientas que se han utilizado en nuestro proyecto, se ha llegado a conseguir una alta fiabilidad en los resultados, ya que en los diferentes intentos que se han llevado a cabo, cambiando la posición de los sensores, los resultados siempre eran los esperados. Se tiene que tener en cuenta, que el coste de estos sensores es muy bajo, en comparación con otros sensores.

**Entorno amigable para el usuario.** Durante el desarrollo del proyecto, una de las ideas más importantes que se quería desarrollar era la facilidad de uso del propio sistema, y de esta forma, agregar agilidad al mismo. Así este proyecto podría llevarse a cabo en diferentes situaciones, siempre haciendo que se pudiese adaptar a cada una de las situaciones que se presentasen en cada momento. Esto se llevó a cabo, por ejemplo, con filtro, haciendo que el usuario pueda elegir las distancias a las cuales ha colocado la diana geométrica, de esta forma, provocando que el usuario tenga más margen de error en su trabajo.

**Versatilidad.** Este proyecto es aplicable al uso de más de dos sensores, pudiendo utilizar hasta cuatro, los cuales permite la SDK de Kinect. Se pensó durante el desarrollo, que podría utilizarse para hacer un mapeado completo de la realidad de la habitación, por eso mismo se incluye esa posibilidad.

**Fidelidad.** Como se ha podido apreciar en las imágenes, las nubes de puntos tomadas están alineadas perfectamente, correspondiendo a la colocación real de la habitación.

**Independia a la iluminación o a los colores.** Como se ha mencionado en algunas partes de este documento, esta aplicación tiene una independencia completa a la luz y el color, lo cual la hace aplicable a cualquier tipo de habitación o iluminación, sin necesidad de cambiar nada en la escena para que se pueda llevar a cabo. Se han utilizado los colores que recibimos del Sensor Microsoft Kinect simplemente por tener una representación visual de la escena que se acerca mucho más a la realidad. Pero este proyecto, de lo único que depende es del sensor de profundidad, y de las formas que este recoge, haciéndolo completamente útil en zonas poco o prácticamente nada iluminadas.



## 6.2. Tiempos de puesta en marcha y computo

En cuanto a tiempos de preparación y cómputo, se van a diferenciar en tres partes, la parte de adquisición, la parte de calibración y la parte de registro de la escena.

Adquisición	Calibración	Sistema de detección de intrusos
Adquisición inicial de las cuatro nubes con los cambios de posición de la diana geométrica	Calculo de la matriz de transformación con el análisis de las nubes de puntos	Mapeado completo de la escena y vigilancia
30 min	20 min	5 min

TABLA 2. Diferentes resultados de tiempos

Como se puede observar en la tabla, se puede tener el programa funcionando ya en situación de vigilancia en aproximadamente una hora. Sin contar con la instalación de los propios sensores en las diferentes situaciones que elijamos.

En cuanto al tiempo de refresco en la vigilancia, no se aprecian cambios elevados respecto de la adquisición de datos en tiempo real, de esta forma comprobando que el algoritmo introducido en el proceso no provoca una lentitud elevada, el resultado obtenido es satisfactorio.

## 6.3. Precisión de la segmentación

Durante el proceso de segmentación se fue comprobando los diferentes resultados que se fueron obteniendo en cuanto a las medidas de la esfera segmentada. Cada vez que se comprobaba el tamaño del radio de la esfera, se veía una pequeña variación en el mismo. El radio lo consideramos constante, ya que la esfera real no cambia de tamaño, puede ser una buena forma de comprobación de la precisión de la toma de distancias.

Después de la toma de medidas del radio de la esfera, de 16 nubes diferentes. Podemos ver que la media de la diferencia de valores obtenidos es de 0.0061 m. Llegando a alcanzar valores máximos de 0,0086 m, lo que equivale a aproximadamente 1 cm.

Viendo las medidas a las que puede llegar a captar valores la propia Microsoft Kinect, consideramos este tipo de variaciones no influyentes en el mismo proceso de calibración.



## 6.4. Mejoras

A lo largo del desarrollo del proyecto se han producido diferentes situaciones que han llevado a cambios en el proyecto. Estos cambios están justificados por diferentes motivos que explicaremos a continuación.

### 6.4.1. Color

Durante la fase de concepción del proyecto, a la hora de pensar una referencia para las diferentes cámaras, se pensó en utilizar dianas de color. Estas dianas hubiesen sido fácilmente reconocibles por algunos métodos de las librerías OpenCV.

El inconveniente de esta opción, es que, si por ejemplo se utilizaba en una habitación determinada, ese color de la diana podría haber estado en la propia escena, añadiendo complejidad a la situación. Otro inconveniente era que la Kinect, cuando colorea la nube de puntos, por error en la precisión de sus sensores, puede colorear algo de un color que pertenece a otro objeto, eso también llevaría a errores.

El programa que se ha creado, no depende de colores, ni si quiera depende de la luz. Ya que se busca una forma determinada, independientemente del color de esta o la situación lumínica de la escena.

Se eligió esta forma de hacer las cosas, ya que vimos que las librerías de PCL permiten detectar diferentes formas de objetos y segmentarlas, incluso pudiendo adquirir los valores que caracterizan cada uno de los objetos, por ejemplo, en el caso de la esfera su punto central y el radio de la misma.

### 6.4.2. Distancia de filtro

Una vez implementada la parte de segmentación de la esfera, se producían diferentes errores, ya que encontraba esferas que no eran si quiera esferas. Después de intentar encontrar solución al problema. Se vio que la nube que procesábamos tenía demasiados puntos para el método de segmentación, por eso mismo se recurrió al filtrado de nube.

Esto hacia que surgiese una nueva dependencia, ya que ahora había que colocar la esfera a una determinada distancia.

Por eso mismo, se creó una función que te mostrase la nube original y te preguntase por donde querías filtrar la nube. Si se veía que la esfera se encontraba dentro de la nueva nube filtrada, puedes continuar con tu proceso, en cambio si no se encuentra dentro de ese filtrado, tendrías que volver a filtrar la nube original.

De esta forma, puedes colocar la esfera donde quieras.

### 6.4.3. Posición de la esfera

Cuando se comenzó el proceso, a la hora de elegir la posición de la esfera, podemos ver por las nubes de puntos que adquirimos en tiempo que real, que la esfera se encuentra en el campo de visión de ambos sensores. De esta forma, se facilita el proceso al usuario. A parte que, si adquirimos desde un sensor y después desde otro, simplemente hay que colocarla en un sitio, sin necesidad de marcarlo, simplemente dejar la esfera en el sitio, tomar las nubes desde ambas posiciones y poner la esfera en una nueva posición haciendo exactamente lo mismo. Un proceso que no implica una complejidad para el usuario, ya que desde un principio se ha pensado en hacerlo todo mucho más sencillo.

#### 6.4.4. Resolución de la vigilancia

La resolución del método que se ha utilizado **OctreePointCloudChangeDetector**, es el tamaño que tienen las rejillas cúbicas en las que divide la nube de puntos. Si este valor es bajo, el sistema será mucho más sensible a cambios, que si utilizamos una resolución mayor, la cual nos permitirá aislar a nuestro objeto de estudio, el intruso.

La vigilancia ha sido adaptada a una determinada resolución, para que no detecte, pequeños cambios de la escena producidos por el error del sensor.

Como se puede apreciar en la siguiente imagen, la resolución en este caso, es muy pequeña. De esta forma detecta cambios es la escena, los cuales no son intrusos.



Figura 31: Errores en la vigilancia 1

En la siguiente imagen, se puede comprobar que hemos aumentado la resolución, de esta forma detecta menos cambios en las propias paredes de la habitación.



Figura 32: Errores en la vigilancia 2

Hasta llegar a una resolución, a la cual el sensor detecta los cambios que se han considerado intrusos.



Figura 33: Vigilancia sin errores

Como se puede apreciar, ya solo detecta al intruso.

## 6.5. Conclusiones

Las conclusiones de nuestro proyecto es que se ha conseguido lo planteado de una forma práctica para el usuario, eliminando las dependencias existentes las cuales existían en otros sistemas de vigilancia y creando una aplicación robusta y versátil.



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES



## 7. Conclusiones y trabajos futuros

### 7.1. Conclusiones

El objetivo de este proyecto, era la creación de un mapa conjunto, con la adquisición de dos sensores desde posiciones diferentes, evitando de esta forma, que cualquier elemento que impidiese la visión completa en cualquier situación se resolviese creando este mapa conjunto. Después, poder utilizar este mapa y tener una representación en tiempo real de esos mismos sensores en nuestro mapa creado, todo ello utilizando el sensor Kinect para Windows.

El sistema de calibración creado, tiene independencia a la luz y los colores, así como el emplazamiento de los sensores o de la diana geométrica. Es de uso sencillo, ya que solo requiere introducir las posibles distancias de la diana, pudiendo repetir el proceso en caso de error. Tenemos que tener en cuenta que este sistema, solo necesita dos puntos diferentes del espacio real, lo que le convierte en una aplicación realmente interesante por el poco trabajo que conlleva al usuario, y la fiabilidad de los resultados.

Finalmente se han logrado los objetivos, de forma que mediante el diseño de algoritmos que buscasen una referencia común en ambas nubes de puntos, pudiesen crear conjuntamente un mapeado y una representación en tiempo real de la escena, a partir de diferentes sensores.

Además, se han diseñado diferentes algoritmos pensando en la comodidad y la sencillez de los usuarios, haciendo que esta aplicación sea lo suficientemente versátil como para poder utilizarla en cualquier situación, incluso creando diferentes módulos que facilitasen el trabajo a módulos posteriores. Todas estas medidas han contribuido al cumplimiento de los objetivos marcados desde un principio.

Como conclusión general de este proyecto tenemos que destacar la calidad de los resultados obtenidos y, sobre todo, la sencillez que se ha podido alcanzar gracias al desarrollo de métodos de optimización durante el proceso.



## 7.2. Trabajos futuros

Pero otras aplicaciones de este mismo proyecto se pueden llevar a cabo:

-Creación de un Mapeado Completo a partir de un mayor número de sensores:

Utilizando la misma referencia que hemos utilizado a lo largo de los procesos anteriores, se puede llegar a crear un sistema de mapeado completo, a partir de más de dos sensores Microsoft Kinect. El modo en el que se llevaría a cabo, es el mismo que hemos utilizado nosotros anteriormente.

Se puede crear primero un mapeado completo como el que hemos creado desde dos posiciones, para después hacer lo mismo desde otras dos posiciones diferentes.

De esta forma, se tendría dos mapas completos, cada una creada a partir de dos sensores en posiciones diferentes.

Se podría calcular una nueva matriz de transformación, la cual estaría condicionada por los orígenes de coordenadas de los mapas completos, de esta forma se podría multiplicar uno de estos mapas por la transformada para una posterior concatenación, teniendo como resultado un mapa completo de la habitación, tomado desde cuatro sensores diferentes.

-Vigilancia desde el móvil:

PCL ha desarrollado librerías para la plataforma Android, existiendo algunas aplicaciones que hacen uso de las mismas. Estas librerías se podrían utilizar en estos sistemas de vigilancia, para ver en tiempo real desde cualquier dispositivo, la escena que quieres vigilar. Así se podría ver la situación de tu casa desde cualquier lugar, simplemente con una aplicación que reciba los datos de la nube de puntos que se actualiza periódicamente en tu teléfono móvil.

-Domótica:

Cuando se menciona el sensor Microsoft Kinect, se habla de los diferentes dispositivos que tiene integrado el propio sensor. Por ejemplo, los micrófonos con las cuales se puede conocer la dirección del sonido, y de donde proviene. A parte un nuevo sistema podría detectar a la persona que está en la habitación, de esta forma los sensores podrían escuchar a una persona y se podría conectar actuadores al propio ordenador, así recibir órdenes de la propia persona y utilizando la SDK de Kinect, detectar que quiere exactamente esa persona, mandando posteriormente información a los actuadores.





## 8. Presupuesto

Se necesita conocer la amortización que tienen los diferentes elementos que se ha utilizado en nuestro proyecto. Viendo las tablas que la Agencia Tributaria publica una vez al año, si nos fijamos en la de 2015 (la última publicada), podemos ver que los equipos electrónicos tienen una amortización del 20%. Al utilizar dos Kinect y un ordenador de sobremesa, podremos incluir ambos elementos en esa categoría.

A parte, el desarrollo del proyecto se ha llevado unas 3 horas diarias y unas 12 los fines de semana, de esta forma se puede conocer el tiempo que ha trabajado el desarrollador con el objetivo de calcular su sueldo y las desamortizaciones.

Para hacer el cálculo de la amortización aplicada a las diferentes herramientas usadas en el desarrollo del proyecto, se tiene en cuenta el porcentaje de uso que se le ha dado en el tiempo de duración del propio desarrollo, ya que existen diferencias en el tiempo de aplicación tanto de las Kinect como del PC.

Con el objetivo de calcular el precio de desamortización, se aplica la siguiente fórmula:

$$\text{Precio de desamortización} = \text{Precio} * \frac{\text{uso}}{100} * \text{coef. amortización} * \frac{4,5 \text{ meses}}{12 \text{ meses}} \quad (8)$$

Elemento	Precio €	Número	Uso	Coeficiente de amortización	Precio de desamortización
Sensor Microsoft Kinect	249	2	30%	0.2	11.20

TABLA 3. Precio de amortización

Para calcular el coste del salario del desarrollador hemos estipulado que unos 20.000 € es lo que cobra al año un Ingeniero recién salido de la universidad.

Entonces el coste del personal sería el siguiente:

Apellidos, Nombre	Categoría	Tiempo (meses)	Coste (€/mes)	Coste total (€)
Areta Díaz, Alejandro	Ingeniero	4,5	803.5	3214.28

TABLA 4. Coste de desarrollador

Una vez calculado el gasto de las herramientas que hemos utilizado en el proyecto y del gasto del salario del desarrollador el cálculo total, asciende a la suma de los dos:

3225.48€



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES



## 9. Planificación

Planificación del trabajo	Duración																	
Mes	Febrero				Marzo				Abril				Mayo					Junio
Semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	5	1	2
1.Planteamiento																		
-Planteamiento																		
-Análisis PCL																		
-Análisis adquisición datos																		
-Estudio de mejoras																		
2.Adquisición datos																		
-Cambios en anterior proyecto																		
-Elección de posicionamiento																		
-Comprobación de resultados																		
3.Segmentación																		
-Comprensión segmentación																		
-Filtrado																		
-Extracción de valores																		
-Comprobación de resultados																		
4.Cálculo matriz de transformación																		
-Cálculo hoja Excel																		
-Planteamiento código																		
-Comprobación de resultados																		
5.Mapeado																		
-Cálculo del mapeado																		
-Pruebas																		
6.Adquisición en tiempo real																		
-Extracción y representación																		
7.Vigilancia																		
-Estudio herramienta Octree																		
-Implementación y resultados																		
8.Tesis																		
-Planteamiento y escritura																		

TABLA 5. Planificación



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES

## 10. Anexo I: Fundamentos en Robótica

### A. Matriz de transformación

La matriz de transformación es una matriz 4x4 que, al ser multiplicada por un punto referenciado a un sistema de coordenadas determinado, podamos recoger los valores de ese mismo punto referenciado a otro sistema de coordenadas.



Figura 34: Mismo punto desde dos sistemas de coordenadas diferentes.

Como se ve en la imagen, se tienen dos sistemas de coordenadas, el O1 y el O2, y un solo punto A. Siendo A1 la representación del punto A desde O1 y A2 la representación desde O2.

En la siguiente fórmula vemos lo que antes explicábamos a la hora de multiplicar esa matriz de transformación T, que si multiplicamos A2 podremos calcular los valores de A1.

$$A_1 = T \times A_2 \quad (9)$$

La matriz de transformación que hemos utilizado es la siguiente:

$$\begin{pmatrix} \cos \alpha & 0 & \sin \alpha & x \\ 0 & 1 & 0 & y \\ -\sin \alpha & 0 & \cos \alpha & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (10)$$



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES

## 11. Anexo II: Estructuración de nubes de puntos

### A. Octree

Un octree es una estructura de datos basada en un árbol jerárquico para organizar los datos tridimensionales dispersos. Cada nodo interno tiene exactamente 8 "hijos". El nodo raíz describe una caja límite cúbica que encapsula todos los puntos. En cada nivel del árbol, este espacio queda subdividido por un factor de 2 en cada una de las tres dimensiones, es decir la caja inicial es dividida en ocho cajas iguales, que se traduce en un aumento de la resolución en la subdivisión espacial. El proceso de subdivisión finaliza cuando el tamaño de las celdas alcanza un valor mínimo determinado.

La siguiente figura ilustra los cuadros delimitadores de las celdas (voxels) de los nodos de un octree en el nivel más bajo del árbol. Los voxels del octree rodean cada punto 3D de la superficie del conejito de Stanford. Los puntos rojos representan los datos de puntos.

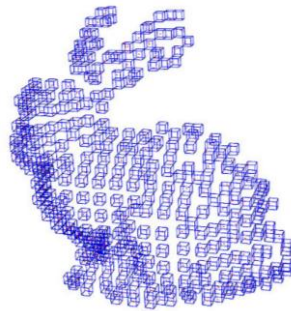


Figura 35: Estructura octree

Información aportada por la estructura octree y sus aplicaciones

- Con el ultimo nivel de subdivisión se consigue que cada punto o dato tridimensional, espacialmente este dentro de una de estas celdas de tamaño determinado. Por ello las estructuras octree se emplean principalmente para dividir un espacio tridimensional, dividiéndolo recursivamente en ocho octantes.
- Desde un punto de vista estructural cada punto ocupa su posición en el octree, en función de las subdivisiones anteriores del espacio de las que procede, de modo que los puntos que están próximos en el espacio van a ser hijos de un mismo nodo. Esto permite las operaciones de búsqueda en el conjunto de datos de puntos.



## ANÁLISIS Y RECONSTRUCCIÓN 3D DE ENTORNOS COMPLEJOS MEDIANTE MÚLTIPLES SENSORES





## 12. Referencias

- [1] F. Castanedo, J. García, M. A. Patricio y J. M. Molina, "Data Fusion to improve trajectory tracking in Cooperative Surveillance Multi-Agent Architecture", *Information Fusion*, Vol.3, p. 243-255, 2010.
- [2] B. Galvin, B. McCane, K. Novins, D. Mason y S. Mills, "Recovering Motion Fields: An Evaluation of Eight Optical Flow Algorithms", *British Machine Vision Conference*, 1998.
- [3] S.C.Huang, F.C. Cheng, "Motion detection with pyramid structure of background model for intelligent surveillance systems", *Engineering Applications of Artificial Intelligence*, Advanced issues in Artificial Intelligence and Pattern Recognition for Intelligent Surveillance System in Smart Home Environment, Vol.25, No.7, p.1338-1348, 2012.
- [4] S. Huwer and H. Niemann, "Adaptive change detection for real-time surveillance applications", *IEEE International Workshop on Visual Surveillance*, pp. 37–45, (Dublin, Ireland), 2000.
- [5] S. Huwer, H.Niemann, "Adaptive change detection for real-time surveillance applications", *Proceedings. Third IEEE International Workshop on Visual Surveillance*, p.37-46, 2000.
- [6] R. J. Radke, S.Andra, O. Al-Kofahi, B. Roysam, "Image change detection algorithms: a systematic survey", *Image processing, IEEE trasactions on*, Vol. 14, p.294-307, 2005.
- [7] S. Kuthirummal, M.Bansal, H.Sawhney, J.Eledath, "3D Alignment and Change Detection from Uncalibrated Eye Images", *First IEEE International Conference on Healthcare Informatics, Imaging and Systems Biology (HISB)*, p.299 - 306, 201
- [8] M.Irani, P.Anandan, "A unified approach to moving object detection in 2D and 3D scenes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.20, No.6, p.577-589, 1998.
- [9] D. Lu, P. Mausel, E. Brondízio y E. Moran, "Change detection techniques", *International Journal of Remote Sensing*, Vol.25, p. 2365-2401, 2004.
- [10] Y. Fischer, J. Beyerer, "A Top-Down-View on Intelligent Surveillance Systems" *The Seventh International Conference on Systems, ICONS*, p.43-48, 2012.
- [11] T. Gill, J. M. Keller, D. T. Anderson, R. H. Luke, "A system for change detection and human recognition in voxel space using the Microsoft Kinect sensor", *Applied Imagery Pattern Recognition Workshop (AIPR)*, IEEE, p.1-8, 2011.
- [12] K. M"uller, A. Smolic, M. Drose, P. Voigt, and T. Wiegand, "3-D reconstruction of a dynamic environment with a fully calibrated background for traffic scenes", *IEEE Transactions on Circuits and Systems for Video Technology* 15(4), pp. 538–549, 2005.
- [13] A. K. Mishra, B. Ni, S. Winkler, A. Kassim, "3D Surveillance System Using Multiple Cameras", *SPIE Proceedings*, Vol.6491, 3D Sensing: Videometrics, 2007.
- [14] M. Agrawal, K. Konolige, and L. locchi, "Real-time detection of independent motion using stereo", *IEEE Workshop on Motion and Video Computing*, pp. 207–214, (Breckenridge, CO, USA), 2005.
- [15] M. Harville, "Stereo person tracking with adaptive plan-view statistical templates," *tech. rep.*, Hewlett-Packard, 2004.
- [16] A.K.Krishnan, S.Saripalli, E.Nissen, R.Arrowsmith, "3D change detection using low cost aerial imagery", *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, p.1 – 6, 2012.
- [17] A.Myronenko, X.Song, "Point Set Registration: Coherent Point Drift", *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol.32, No.12, PP. 2262-2275, 2010.
- [18] J.L.Martinez, J.L., A.J.Reina, J.Morales, A.Mandow, "Using multicore processors to parallelize 3D point cloud registration with the Coarse Binary Cubes method", *IEEE International Conference on Mechatronics (ICM)*, pp.335-340, 2013.
- [19] J.Tang, J.Luo, T.Tjahjadi y Y. Gao, "2.5D Multi-View Gait Recognition Based on Point Cloud Registration", *Sensors*, No.14, p. 6124-6143, 2014.



- [20] D.Makris, T. Ellis, "Learning semantic scene models from observing activity in visual surveillance", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetic, Vol.35, No.3, p.397-408, 2005.
- [21] L. Wang, G. Sohn, "An Integrated Framework for Reconstructing Full 3D Building Models", Advances in 3D Geo-Information Sciences, Lecture Notes in Geoinformation and Cartography , p.261-274, 2011.
- [22] P.Henry, M. Krainin, E.Herbst, X.Ren, D.Fox, "RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments", Experimental Robotics, Springer Tracts in Advanced Robotics, Vol. 79, pp. 477-491, 2014.
- [23] J. Tang, J. Luo, T. Tjahjadi y Y. Gao, "2.5D Multi-View Gait Recognition Based on Point Cloud Registration", Sensors, Vol.14, No.4, pp.6124-6143, 2014.
- [24] P. Henry, M. Krainin, E.Herbst, X. Ren, D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments", International Journal of Robotics Research, Vol.31, No.5, pp. 647-663, 2012.
- [25] S. Lee, W.K. Chung, "Rotating IR Sensor System for 2.5D Sensing", 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.814-819, 2006.
- [26] H. Park, S. Lee y W.K. Chung, "Obstacle Detection and Feature Extraction using 2.5D Range Sensor", SICE-ICASE, 2006. International Joint Conference, p.2000-2004, 2006.
- [30] Rodríguez Zalapa, O., Hernández Zavala, A., & Huerta Ruelas, J. A. (2014). Sistema de medición de distancia mediante imágenes para determinar la posición de una esfera utilizando el sensor Kinect XBOX. Polibits, (49), 59