



Escuela  
Politécnica  
Superior

# Segmentación de objetos 3D para tareas de interacción con robots



Grado en Ingeniería Robótica

## Trabajo Fin de Grado

Autor:

Gonzalo Ferrando Alonso

Tutor/es:

Miguel A. Cazorla Quevedo

Félix Escalona Moncholí

Junio 2021



Universitat d'Alacant  
Universidad de Alicante



# Segmentación de objetos 3D para tareas de interacción con robots

---

## Autor

Gonzalo Ferrando Alonso

## Tutor/es

Miguel A. Cazorla Quevedo

Departamento de Ciencia de la Computación e Inteligencia Artificial

Félix Escalona Moncholí

Departamento de Ciencia de la Computación e Inteligencia Artificial



Grado en Ingeniería Robótica



Escuela  
Politécnica  
Superior



Universitat d'Alacant  
Universidad de Alicante

ALICANTE, Junio 2021



# **Preámbulo**

“La razón por la que se ha llevado a cabo este proyecto es el interés hacia las nuevas tecnologías y las nuevas técnicas de programación. A lo largo del grado se nos ha introducido muchos conceptos que se han desarrollado hace pocos años y la causa de elección de este Trabajo de Fin de Grado es para poder combinar dichos elementos en un sistema que englobe varios de ellos como la detección de objetos, el tratamiento de datos 2D y 3D y la localización y mapeado simultáneos.”



# **Agradecimientos**

Este trabajo no se podría haber llevado a cabo si no fuera por los consejos y guías de mis tutores Miguel Cazorla y Félix Escalona, los cuales me prestaron su ayuda durante cada etapa de este proyecto.

También quiero acordarme de mis compañeros de la carrera, exactamente de Nicolás, Darío, Ángel, Carlos A., María, Óscar, Fernando, Iván, Carlos E. y Jorge, ya que sin ellos puede que no hubiera llegado al punto en el que estoy en la carrera y en la vida.

Por último y no menos importante, quiero agradecer a mi madre, mi padre y mi hermana por haber estado conmigo cada día y haber sido uno de mis grandes apoyos morales y emocionales a lo largo de toda la carrera, dado que no ha sido nada fácil estos cuatro años de carrera y menos este último año y medio.

Muchas gracias a todos, de verdad.



*El fracaso es una gran oportunidad para empezar otra vez con más inteligencia.*

Henry Ford.



# Índice general

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Marco Teórico</b>	<b>5</b>
2.1	Detección de objetos . . . . .	5
2.2	SLAM . . . . .	7
2.3	Segmentación de objetos 3D . . . . .	8
2.4	Eliminación de background . . . . .	8
2.4.1	Algoritmo Region Growing Segmentation . . . . .	8
2.4.2	Algoritmo Color-based Region Growing Segmentation . . . . .	9
2.4.3	Algoritmo Conditional Euclidean Clustering . . . . .	10
2.4.4	Eliminación de planos dominantes . . . . .	11
<b>3</b>	<b>Objetivos</b>	<b>13</b>
<b>4</b>	<b>Metodología</b>	<b>15</b>
4.1	Setup . . . . .	15
4.2	ROS . . . . .	15
4.2.1	¿Qué es ROS? . . . . .	15
4.2.2	Objetivos de ROS . . . . .	16
4.2.3	Principales conceptos . . . . .	16
4.2.3.1	Conceptos de sistemas de archivos . . . . .	16
4.2.3.2	Conceptos de computación gráfica . . . . .	16
4.2.4	Funcionamiento . . . . .	17
4.3	Point Cloud Library (PCL) . . . . .	18
4.4	OpenCV . . . . .	19
4.5	Datasets de prueba . . . . .	19
4.6	YOLO Darknet . . . . .	20
4.6.1	Paquete de ROS para YOLO-Darknet . . . . .	21
4.6.1.1	Parámetros . . . . .	21
4.6.1.2	Topics de entrada y salida . . . . .	22
4.6.1.3	Acciones . . . . .	22
4.7	Registro por parejas de nubes de puntos . . . . .	22
4.8	ORB SLAM 2 . . . . .	24
4.8.1	ORB-SLAM2 para ROS . . . . .	25
4.8.1.1	Parámetros . . . . .	25
4.8.1.2	Topics de entrada y salida . . . . .	26
4.8.1.3	Servicios . . . . .	26

<b>5 Desarrollo</b>	<b>27</b>
5.1 Desarrollo del pipeline . . . . .	27
5.1.1 Lectura de topics . . . . .	29
5.1.2 Detección de objetos en Darknet . . . . .	30
5.1.3 Procesamiento de las nubes de puntos . . . . .	30
5.1.4 Incorporación del ORB-SLAM2 . . . . .	32
5.1.4.1 Mapeo y localización con ORB-SLAM2 . . . . .	32
5.1.4.2 ORB-SLAM2 en el pipeline general . . . . .	35
5.1.5 Combinación de los mensajes de ORB-SLAM2 y las nubes de puntos recortadas . . . . .	36
5.1.5.1 Registro de pares de nubes de puntos . . . . .	38
5.1.5.2 Aplicación de las matrices del registro de pares de nubes . . . . .	40
5.1.5.3 Formación de la escena final . . . . .	40
5.2 Sincronización de topics . . . . .	41
5.2.1 Exact Time Policy . . . . .	41
5.2.2 Approximate Time Policy . . . . .	41
5.2.3 Mensajes personalizados para la sincronización de mensajes . . . . .	41
<b>6 Experimentación y resultados</b>	<b>43</b>
6.1 Experimentación con ORB-SLAM2 con diferentes datasets . . . . .	43
6.2 Eliminación de background en las nubes de puntos de objetos . . . . .	44
6.2.1 Region growing segmentation . . . . .	45
6.2.1.1 Resultados . . . . .	45
6.2.2 Conditional Euclidean Clustering . . . . .	46
6.2.2.1 Resultados . . . . .	46
6.2.3 Color-based region growing segmentation . . . . .	47
6.2.3.1 Resultados . . . . .	47
6.2.4 Eliminar planos dominantes . . . . .	50
6.2.4.1 Resultados . . . . .	50
6.3 Sincronización de mensajes de ORB-SLAM2 y las nubes de los objetos . . . . .	52
6.4 Uso del registro de pares de nubes . . . . .	53
6.5 Elección del mejor pipeline de registro de pares de nubes . . . . .	53
6.6 Eliminación del error acumulado al transportar las nubes de puntos al sistema de la nube origen . . . . .	55
6.7 Combinación de nubes puntos para formar la escena completa . . . . .	57
6.8 Resultados del desarrollo del pipeline . . . . .	57
<b>7 Conclusiones</b>	<b>63</b>
<b>Bibliografía</b>	<b>65</b>
<b>Lista de Acrónimos y Abreviaturas</b>	<b>67</b>

# Índice de figuras

1.1	Esquema de una RCNN. Fuente (Girshick y cols., 2014) . . . . .	1
1.2	Influencia del NMS en la selección de área de interés. Fuente: (Bagnat, 2020)	2
1.3	Funcionamiento de SLAM. Fuente: (Durrant-Whyte y Bailey, 2006) . . . . .	3
2.1	Comparación de Yolo-v4 en frente a otros detectores. Fuente: (Bochkovskiy y cols., 2020) . . . . .	6
4.1	Arquitectura básica del funcionamiento de ROS. . . . .	17
4.2	Visualización de una nube de puntos. . . . .	19
4.3	Ejemplo de imagen del archivo bag de ROS. . . . .	20
4.4	Ejemplo de predicción de imágenes de Darknet. . . . .	21
4.5	Estructura del pipeline. Fuente: ( <i>PCL pointcloud pairwise registration</i> , s.f.) .	24
4.6	Estructura de hilos de ORB-SLAM2. Fuente: (Mur-Artal y Tardós, 2017) . .	25
5.1	Terminal lanzando roscore. . . . .	27
5.2	Terminal lanzando Darknet-ROS. . . . .	28
5.3	Terminal lanzando ORB-SLAM2-ROS. . . . .	28
5.4	Terminal lanzando archivo bag. . . . .	28
5.5	Terminal cargando los parámetros de la cámara para empezar ORB-SLAM. .	33
5.6	Dispaly de RViz para ORB-SLAM2. . . . .	33
5.7	Keypoints detectados de la imagen. . . . .	34
5.8	Nube de puntos de los keypoints. . . . .	34
5.9	Entrada de mensajes del topic de imágenes de profundidad. . . . .	35
5.10	Entrada de mensajes del topic de imágenes de RGB. . . . .	35
5.11	Antes del registro completo entre dos nubes de puntos. . . . .	38
5.12	Emparejamientos entre dos nubes de puntos. . . . .	39
5.13	Registro completo entre dos nubes de puntos. . . . .	39
6.1	Escena de prueba. . . . .	44
6.2	Objetos recortados de la escena para hacer las pruebas de eliminación de background. . . . .	44
6.3	Objetos recortados de la escena, tras haberle aplicado el algoritmo de Region Growing Segmentation para eliminar el background. . . . .	46
6.4	Clusters de los objetos recortados de la escena usando Conditional Euclidean Clustering. . . . .	47
6.5	Nube de puntos tras haberse aplicado RGB region growing segmentation. .	48
6.6	Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del portátil. . . . .	49
6.7	Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del monitor. . . . .	49

6.8 Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del ratón de portátil. . . . .	49
6.9 Objetos recortados de la escena recortando los planos dominantes. . . . .	50
6.10 Ejemplo de varios objetos detectados en la misma zona. . . . .	51
6.11 Nube de puntos recortada del oso de peluche y la silla. . . . .	51
6.12 Dos perspectivas de la escena. . . . .	52
6.13 Ejemplo de separación entre dos escenas. . . . .	53
6.14 Mala alineación por acumulación de error en las matrices de transformación(i). . . . .	56
6.15 Mala alineación por acumulación de error en las matrices de transformación(ii). . . . .	56
6.16 Escena final bien alineada. . . . .	57
6.17 Gráfico final del pipeline. . . . .	58
6.18 Nube de puntos de los objetos bien recortadas. . . . .	59
6.19 Nube de puntos con objetos no recortados. . . . .	59
6.20 Nube de puntos con objetos mal recortados. . . . .	60
6.21 Escena final(i). . . . .	61
6.22 Escena final(ii). . . . .	61
6.23 Escena final(iii). . . . .	62
6.24 Escena final mal alineada. . . . .	62

# Índice de tablas

6.1	Mejores parámetros del algoritmo Region Growing Segmentation. . . . .	46
6.2	Mejores parámetros del algoritmo Conditional Euclidean Clustering. . . . .	47
6.3	Mejores parámetros del algoritmo Color-based Region Growing Segmentation.	48
6.4	Comparación del tiempo de ejecución(ms) de los extractores de características.	54
6.5	Comparación del tiempo de ejecución(ms) de los extractores de características más el computo de descriptores. . . . .	54
6.6	Parámetros del pipeline del registro de pares de nubes utilizado (ISS más PFH).	55



# Índice de Códigos

2.1	Código del algoritmo de Region Growing Segmentation. . . . .	9
2.2	Código del algoritmo de Color-based region growing segmentation. . . . .	9
2.3	Código del algoritmo de Conditional Euclidean Clustering. . . . .	10
2.4	Código del algoritmo de Plane Model Segmentation. . . . .	11



# 1 Introducción

Hoy en día se habla mucho de la inteligencia artificial, el deep learning y todas las herramientas punteras que van surgiendo poco a poco y que en un futuro no muy lejano se irán desarrollando y serán mucho mejores y más variadas. En este proyecto se hacen uso de estas nuevas tecnologías y técnicas para la detección de objetos en imágenes 2D, el tratamiento de nubes de puntos para obtener las proyecciones de los objetos en formato 3D y, al mismo tiempo, localizar a la cámara dentro del entorno para poder crear un mapa con la información extraída de los objetos detectados en tiempo real. Para empezar, se va a hablar de las dos principales vertientes en las que se basa este proyecto, primero es la detección de objetos. Este campo es una técnica relacionada con la visión artificial y el procesamiento de imágenes que trata de detectar una clase de objeto o varias en un vídeo o en una imagen. Su funcionamiento en general se basa en las redes Region Based Convolutional Neural Network (R-CNN), que surgen sobre 2014 y su esquema se muestra en la Figura 1.1. Esta red tiene dos partes. La primera tiene como propósito determinar qué regiones son de “interés” dentro de la imagen y la segunda parte que realiza una clasificación de imágenes sobre las regiones que se han nombrado como de “interés”, esa clasificación se hará mediante una preentrada. En la siguiente Figura 1.1 se observa cómo es la estructura real de una R-CNN.

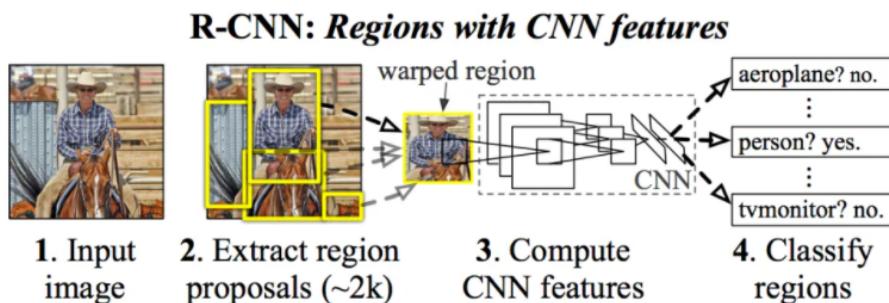


Figura 1.1: Esquema de una RCNN. Fuente (Girshick y cols., 2014)

La R-CNN recibe una entrada que en este caso serán las imágenes a detectar, para que luego detecte las áreas de “interés” como zonas contiguas del mismo color o detección de líneas que delimitan áreas, en muchos casos pueden llegar a tener miles de regiones y de diversos tamaños. El siguiente paso es pasar por la Convolutional Neural Network (CNN), que es una red neuronal cuyo propósito es analizar la entrada o imagen que llega, para ello hace uso de convoluciones, que son operaciones matemáticas. Luego, mediante el clasificador binario se valida si las regiones pertenecen a las clases correctas y se eliminan las de poca confianza. Por último, un regresor se encarga de ajustar la posición correcta. Hay casos en los que hay un solapamiento de varias zonas de interés y el método para poder solucionarlo es el de la Intersection over Union (IoU), que da un porcentaje de acierto del área de predicción

frente a la caja que se quería detectar (Ecuación 1.1).

$$IoU = \frac{\text{AreaOfOverlap}}{\text{AreaOfUnion}} \quad (1.1)$$

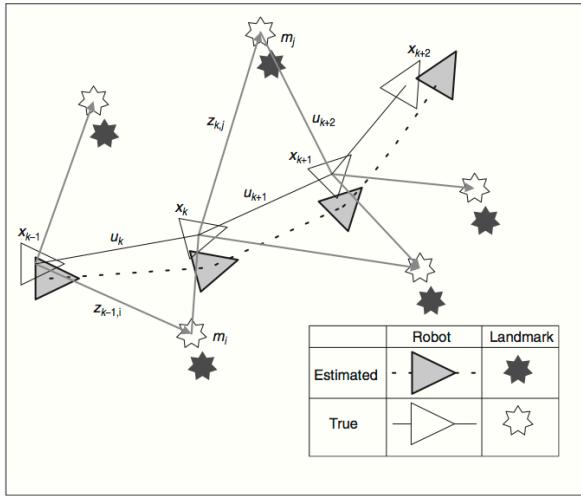
El IoU si se usa con Non Maximum Suppression (NMS) ayuda a seleccionar el área del objeto que se ha querido detectar, obteniendo el bounding box más ajustado y eliminando el resto de cajas detectadas de alrededor del objeto. Si se aplica el NMS influye como en la Figura 1.2.



**Figura 1.2:** Influencia del NMS en la selección de área de interés. Fuente: (Bagnat, 2020)

Finalmente, se obtiene un o varios bounding box del objeto u objetos que se han detectado en la imagen de entrada. Para este trabajo será muy útil saber los objetos que hay en cada escena, además, de las coordenadas donde estén en la imagen de entrada, ya que también se usará la nube de puntos que representa esa imagen, para poder obtener los objetos en formato 3D, luego se tendrá que tratar la nube de puntos recortada de la escena para obtener el objeto de la mejor manera.

La siguiente técnica que se hace uso en este trabajo es conocida como Simultaneous Localization And Mapping (SLAM) o localización y mapeado simultáneos (Durrant-Whyte y Bailey, 2006). Esta técnica se usa por robots y vehículos autónomos para la construcción de mapas de un entorno desconocido en el que el robot se encuentra. Al mismo tiempo, el robot estima en qué posición se encuentra, haciendo las tareas de localización y mapeado a la vez. Para ello es necesario conocer anteriormente unos puntos característicos que serán utilizados como referencias.



**Figura 1.3:** Funcionamiento de SLAM. Fuente: (Durrant-Whyte y Bailey, 2006)

En la Figura 1.3, se puede observar el comportamiento de un robot con cierta imprecisión de los sensores, lo que va generando incertidumbre en la posición donde está observando los puntos característicos. Aunque se tuvieran los mejores sensores, la incertidumbre seguirá surgiendo, esto es debido por el modelo de movimiento, que no es perfecto y no siempre se mueve exactamente con las órdenes de control y que no se conoce el punto de partida del robot, pero para SLAM es lo que le hace diferente a otros métodos, porque no necesita un protocolo de inicialización.

Otra ventaja que proporciona SLAM es que dicha incertidumbre se reduce cuando el robot vuelve a observar los puntos característicos que ya se han visitado anteriormente, por lo que lo hace ser como un sistema que va ganando en eficiencia y precisión.

Por último, pero no por eso menos importante, también se va a hacer uso de diferentes algoritmos proporcionados por la Point Cloud Library (PCL), con tal de realizar una segmentación en la nube de puntos de objetos que se hayan detectado en la escena correspondiente y de eliminar aquellos puntos no pertenecientes al objeto detectado. Este paso será uno intermedio entre la detección de objetos y la generación del mapa con la información que proporciona el SLAM que se use en el proyecto.

En este trabajo se van a aplicar tecnologías como ROS y su uso de paquetes para la detección de objetos mediante YOLO Darknet o SLAM. También se harán nodos de ROS propios para poder realizar diferentes funciones. En esos nodos se usará la librería PCL con la finalidad de tratar las nubes de puntos para recortar los objetos de la nube de puntos y eliminar su background mediante diferentes algoritmos, o algoritmos para realizar un registro de pares de nubes para alinear todas las escenas en un mismo sistema de coordenadas y poder unirlas para obtener un mapa donde puedan ser localizados dichos objetos detectados. Por lo tanto, este mapa contendrá mucha más información relevante que un mapa realizado a partir de un simple SLAM.

Así pues, el documento se resume en las diferentes secciones, en la Sección 2, se habla del marco teórico que hay en la detección de objetos y el SLAM, explicando diferentes proyectos que tienen algo de similitud con este. En la Sección 3 se proponen todos los objetivos pro-

puestos que se pretenden alcanzar durante la realización de este proyecto, tanto personales como del trabajo en sí. En la Sección 4 se habla de la metodología, en este apartado se explican todos los instrumentos y herramientas que se hacen uso en este proyecto para poder llevarlo a cabo. En la Sección 5 se explica que uso tienen cada uno de ellos y como se ha desarrollado y aplicado en el proyecto para poder formar el pipeline que consiga el objetivo final planteado. Después, se explica en la Sección 6 todas las pruebas y experimentaciones que se han llevado a cabo para poder encontrar los mejores resultados y llegar al objetivo final, además de mostrar los resultados finales obtenidos. Para terminar, en la Sección 7 se expondrán las conclusiones obtenidas durante el transcurso del proyecto.

## 2 Marco Teórico

A continuación, se van a explicar los antecedentes e investigaciones actuales de dos diferentes ramas de tecnologías como son la detección de objetos haciendo uso de redes neuronales y el mapeado y localización simultáneos de un robot en un entorno.

### 2.1 Detección de objetos

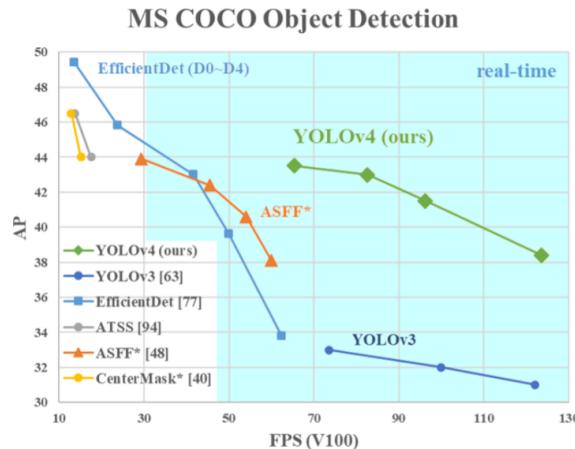
A día de hoy tanto las técnicas de detección de objetos mediante redes neuronales y el SLAM están muy avanzadas, pero siempre están en continuo desarrollo, ya que cada año hay avances tecnológicos o científicos que pueden ayudar a encontrar nuevos métodos que pueden hacer mejorar lo que ya había.

Una de las principales aplicaciones de la visión por computador es la de reconocer los objetos que nos rodean. Por ese motivo hay gran cantidad de estudios y técnicas en relación con el reconocimiento de objetos y que han ido mejorando cada año, dado que al principio el uso de redes neuronales para la detección de objetos tardaba bastante tiempo, hoy en día dicho tiempo se ha reducido a milésimas de segundo. A continuación, se van a mencionar algunos ejemplos de redes neuronales que tienen como objetivo la detección de objetos, pero cabe destacar que la que se va a utilizar en el proyecto es la primera que se menciona y se usará mediante un paquete de ROS.

Antes de mencionar diferentes ejemplos de redes neuronales que tienen como objetivo la clasificación de objetos, hay que mencionar que una red neuronal con este objetivo puede ser desarrollada por diferentes frameworks open-source como TensorFlow, PyTorch o Caffe entre ellas, pero en este trabajo se usará una red ya entrenada.

En 2016, se crea You Only Look Once (YOLO) (Bjelonic, 2016–2018), esta red hace una única pasada a la red convolucional y detecta todos los objetos para los que ha sido entrenada para clasificar. Al ser un “solo cálculo” y sin necesidad de iterar, logra velocidades nunca antes alcanzadas con ordenadores que no tienen que ser tan potentes. Esto también permite detección sobre vídeo en tiempo real de objetos en simultáneo.

Para su funcionamiento hace uso del framework de desarrollo Darknet (Redmon, 2013–2016), aunque también puede ser entrenada con cualquier otra red neuronal convolucional. Además, YOLO utiliza las neuronas de tipo convolucional al final de la cadena sin necesidad de hacer la transformación a una red “tradicional”. Con el paso del tiempo, YOLO ha ido mejorando y evolucionando las versiones YOLO V2, V3 y recientemente V4 que están enfocadas a mejorar esa precisión de las bounding boxes.



**Figura 2.1:** Comparación de Yolo-v4 en frente a otros detectores. Fuente: (Bochkovskiy y cols., 2020)

En la Figura 2.1, se compara la precisión (AP) frente a la velocidad (FPS). Como se puede ver YOLOv4 supera en velocidad a EfficientNet y en precisión a su antigua versión YOLOv3, así que se puede decir que la nueva versión de YOLO mejora tanto en la precisión de detectar los objetos como en la velocidad que lo hace.

También hay otras alternativas a YOLO y que hacen el mismo trabajo, pero con otras técnicas diferentes. A continuación, se mencionan algunos trabajos sobre redes neuronales para la detección de objetos.

La Single Shot Detector (SSD) (Liu y cols., 2016) es una red neuronal de detección de objetos desarrollada en 2016, destaca por su estructura piramidal en su CNN, lo que permite poder detectar tanto objetos grandes como pequeños. El modelo de SSD es simple frente a métodos que requieren propuestas de objetos, porque elimina por completa la generación de propuestas y la etapa de remuestreo de píxeles. Esto hace que la red sea fácil de entrenar y de integrar en un sistema que requiera la detección de objetos. Con los resultados experimentales con diferentes datasets, SSD confirma una precisión comparable con métodos que utilizan un paso adicional de propuestas de objetos. Además de ser mucho más rápido. Cabe destacar que SSD tiene muy buena precisión, incluso con imágenes de menor tamaño.

En 2018, se desarrolló RetinaNet (Lin y cols., 2018b), esta se basa en una estructura CNN piramidal mejorada para reconocer objetos de diversos tamaños en una sola pasada. Además, innova con una nueva función de pérdida llamada “Focal Loss” (Lin y cols., 2018a). RetinaNet es una única y unificada red neuronal compuesta por una red troncal y dos subredes de tareas específicas. La red troncal es responsable de calcular un mapa de características convolucionales sobre las imágenes de entrada. Además, es una red convolucional independiente. La primera subred ejecuta la clasificación de los objetos de la salida de la red troncal; la segunda subred realiza una regresión de cuadro delimitador convolucional.

Spinet (Xianzhi Du y Jaeyoun Kim, 2020) es una red neuronal de detección de objetos desarrollada por Google en 2019, destacando por su estructura, ya que rompe con la estructura piramidal y propone una llamada “scale-permuted”, en la que se alternan diversos tamaños en las convoluciones y habilita dos grandes mejoras en la arquitectura del diseño de la red troncal. La primera, la resolución espacial de los mapas de características debería poder

aumentar o disminuir en cualquier momento para que el modelo pueda retener información espacial a medida que se profundiza. La segunda, las conexiones entre mapas de características deberían poder atravesar escalas de características para facilitar la fusión de características de múltiples escalas.

Facebook (Carion y cols., 2020) presenta en junio de 2020 una “End to End detection with Transformers”, la red neuronal conocida como Detection Transformer (DETR). Esta red utiliza la más novedosa y efectiva técnica de redes neuronales. DETR aborda la detección de objetos como una predicción de conjuntos directos. Esta red no usa las técnicas tradicionales de la visión por computador, DETR aborda el problema de la detección de objetos como una predicción de conjuntos discretos. Consiste en una pérdida global basada en conjuntos, que fuerza predicciones únicas a través de un emparejamiento bipartito y una arquitectura de codificador-decodificador de transformador. También hay que recalcar que debido a su naturaleza paralela, DETR es muy rápido y eficiente, y su código es muy simple de implementar y fácil de experimentar.

## 2.2 SLAM

La localización y mapeo simultáneos (SLAM) ha sido un tema de investigación candente en las últimas dos décadas en las comunidades de visión artificial y robótica, y recientemente ha atraído la atención de empresas de alta tecnología. Los investigadores casi siempre centran sus esfuerzos en mejorar el algoritmo en algunos casos concretos. En los últimos años, los temas de mejora se centran en el reconocimiento de objetos u optimización de coste computacional.

Existen varios tipos de SLAM. Para clasificarlos se hace mediante los subsistemas que integran cada sistema, los tipos de sensores que se utilizan o la funcionalidad que aporta. Empezamos con los sistemas de SLAM 2D o 3D, los de 2D son útiles en entorno planos y basan las observaciones que realizan en sensores de profundidad o de distancia.

También se pueden diferenciar en basados en puntos característicos o directos, los primeros cogen referencias para poder ubicarse, estos puntos pueden ser datos introducidos manualmente o que son detectados por un sensor, en cambio, el método directo ignora los puntos característicos y se centra en la intensidad de los píxeles.

Otra clasificación es mediante el tipo de cámara que se esté usando, dado que se puede usar un input de imágenes Red, Green and Blue - Depth (RGB-D) como las que dan las cámaras monoculares, siendo un sistema simple y de bajo precio, también se hacen uso de cámaras estéreo que son capaces de calcular distancias de los objetos encontrados en la imagen o RGB-D que dan la entrada de imágenes Red, Green and Blue (RGB) e imágenes de profundidad de la escena.

Por último, se puede clasificar un método de SLAM dependiendo de cómo operan: pueden ser online que operan en tipo real, calculan las poses del punto de vista en cada momento o si se quiere profundizar en la secuencia y analizarla mucho mejor se puede hacer offline, que lo que se hace es grabar la secuencia en la cual se quiera hacer SLAM y más tarde analizarla, esto da la libertad y las ventajas de implementar un sistema más potente y mejor.

En este trabajo se trabajará con SLAM para cámaras RGB-D, por eso que el marco teórico de SLAM se centrará en proyectos que tengan en cuenta este tipo de cámaras. Uno de los trabajos más recientes es el RGB-D Tracking and Mapping (RGBDTAM) (Concha y Civera, 2017), es un sistema desarrollado en la Universidad de Zaragoza que fue desarrollado a

---

partir de otros trabajos previos, pero se le añadieron nuevas aportaciones por parte de los desarrolladores.

Otros de los métodos de SLAM para cámaras RGB-D es el ORB SLAM2 (Mur-Artal y Tardós, 2017), que es una biblioteca SLAM en tiempo real para cámaras monoculares, estéreo y RGB-D que calcula la trayectoria de la cámara y hace una reconstrucción 3D dispersa (en el caso estéreo y RGB-D con escala real).

Aunque SLAM es una técnica que mayoritariamente es usada en la robótica y en la automatización de procesos industriales, en los últimos años sus aplicaciones se han multiplicado. Por ejemplo, se tiene el mapeo 2D para el uso de robots domésticos como aspiradores, donde es suficiente para aplicaciones de zonas planas donde la información de distancias es lo único necesario. También se usa en la realidad aumentada, utilizando la información de profundidad que proporciona para colocar o ubicar los objetos de manera coherente, esto hace que su uso se haya extendido a empresas de tecnología aeroespacial, mecánica o médica, siendo de gran ayuda ya que puede simular entornos semi-virtuales para los trabajadores y dotarlos de un entorno semi-real para su instrucción. A fin de cuentas, el SLAM también se está ampliando en otros campos mucho más distantes de la robótica y la industria como son las aplicaciones en medicina o la industria del videojuego.

## 2.3 Segmentación de objetos 3D

En referencia al estado del arte de la segmentación de objetos 3D hay diferentes proyectos que hacen una segmentación perfecta de objetos 3D. Pero en este proyecto no se buscará ese objetivo por dos razones: la primera es que no se tendrá casi nunca una visión de 360º del objeto a recortar y la segunda es que el objetivo de segmentar objetos 3D es eliminar los puntos que no pertenezcan al objeto que se recorte de la escena. Por ejemplo, los puntos que están detrás del objeto o los que están por debajo.

## 2.4 Eliminación de background

Cuando se tengan los objetos segmentados de sus respectivas nubes, se usarán diferentes algoritmos y métodos como son Region Growing Segmentation, Color-based Region Growing Segmentation, Conditional Euclidean Clustering y la eliminación de planos dominantes que son proporcionados por la Point Cloud Library, para eliminar aquellos puntos que no pertenecen a dicho objeto. A continuación, se explica el marco teórico de los cuatro métodos.

### 2.4.1 Algoritmo Region Growing Segmentation

A continuación, se va a ver cómo trabaja el algoritmo Region Growing Segmentation (Library, s.f.-d). Primero, se observa el valor de curvatura de todos los puntos y se empieza con el punto con el valor de curvatura más pequeño o mínimo, la razón es porque dicho punto está localizado en un área plana. El proceso continúa en los siguientes pasos:

- El punto escogido es añadido al cluster.
  - Para cada punto se encuentra sus vecinos y para cada vecino se comprueba el ángulo entre la normal del vecino escogido y el punto actual. Sí el ángulo es menor que el valor
-

que es puesto, el punto se añade al cluster actual. Después de comprobar la curvatura del vecino, se compara la curvatura con el valor puesto en el algoritmo y si es menor se añade al cluster.

- Sí no cumple las restricciones anteriores no se añade al cluster.

Sí el cluster se vacía, significa que el algoritmo ha hecho crecer la región y el proceso se repite desde el principio. Finalmente, en el Código 2.1 se ve su implementación.

Código 2.1: Código del algoritmo de Region Growing Segmentation.

```

1   pcl::search::Search<pcl::PointXYZRGB>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZRGB>);
2   pcl::PointCloud <pcl::Normal>::Ptr normals (new pcl::PointCloud <pcl::Normal>);
3   pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> normal_estimator;
4   normal_estimator.setSearchMethod (tree);
5   normal_estimator.setInputCloud (cloud);
6   normal_estimator.setKSearch (50);
7   normal_estimator.compute (*normals);
8   pcl::RegionGrowing<pcl::PointXYZRGB, pcl::Normal> reg;
9   reg.setMinClusterSize (50);
10  reg.setMaxClusterSize (1000000);
11  reg.setSearchMethod (tree);
12  reg.setNumberOfNeighbours (30);
13  reg.setInputCloud (cloud);
14  reg.setInputNormals (normals);
15  reg.setSmoothnessThreshold (3.0 / 180.0 * M_PI);
16  reg.setCurvatureThreshold (1.0);
17  std::vector <pcl::PointIndices> clusters;
18  reg.extract (clusters);

```

## 2.4.2 Algoritmo Color-based Region Growing Segmentation

El marco teórico de Color-based Region Growing Segmentation (Library, s.f.-a) es muy similar al de la clase **pcl::RegionGrowing**, en el Código 2.2 se puede ver como se implementa y que parámetros usa, explicada en la Subsección 2.4.1, pero tiene dos diferencias notables. La primera, es que hace uso de colores en vez de las normales para clasificar los diferentes clusters y la segunda es que usa el algoritmo de fusión para el control de la segmentación.

Después de la segmentación, se intenta unir los clusters con colores similares, luego el segundo paso empieza. Durante el segundo paso cada cluster es verificado por el número de puntos comprobando si es menor que el número que ha definido el usuario, sí lo es dicho cluster es unido con su cluster vecino más cercano.

Código 2.2: Código del algoritmo de Color-based region growing segmentation.

```

1   pcl::search::Search <pcl::PointXYZRGB>::Ptr tree (new
2   pcl::search::KdTree<pcl::PointXYZRGB>);
3
4   pcl::RegionGrowingRGB<pcl::PointXYZRGB> reg;
5   reg.setInputCloud (cloud);
6   reg.setSearchMethod (tree);
7   reg.setDistanceThreshold (10);
8   reg.setPointColorThreshold (5);
9   reg.setRegionColorThreshold (6);
10  reg.setMinClusterSize (0);
11  std::vector <pcl::PointIndices> clusters;

```

```
12     reg.extract (clusters);
```

### 2.4.3 Algoritmo Conditional Euclidean Clustering

El algoritmo Conditional Euclidean Clustering (Library, s.f.-b) usa a la vez el “region growing segmentation” y “color-based region growing segmentation”, pero la ventaja de esta clase que los parámetros del clúster son definibles por el usuario. En cambio, también presenta desventajas como que no tiene un sistema inicial de semilla para empezar a agrupar puntos en clusters y también es que es menos eficiente frente al tiempo.

Cada vez que el cluster crece, se evaluarán las condiciones definidas por el usuario entre puntos que están dentro del cluster y cerca de los puntos candidatos. Los candidatos son encontrados mediante el radio de búsqueda Euclídeo de cada punto del cluster. Para cada punto añadido en el cluster, la condición debe mantenerse con al menos un vecino y no con todos.

La clase “Conditional Euclidean Clustering” también puede filtrar clusteres automáticamente según una restricción de tamaño. Los grupos clasificados como demasiado pequeños o demasiado grandes aún se pueden recuperar posteriormente. El tipo de nube que se va a utilizar es “**pcl::PointXYZI PointTypeIO**” y el tipo de normales es “**pcl::PointXYZINormal PointTypeFull**”.

Código 2.3: Código del algoritmo de Conditional Euclidean Clustering.

```
1      pcl::VoxelGrid<PointTypeIO> vg;
2      vg.setInputCloud (cloud);
3      vg.setLeafSize (0.05, 0.05, 0.05);
4      vg.setDownsampleAllData (true);
5      vg.filter (*cloud_out);
6
7      // Set up a Normal Estimation class and merge data in cloud_with_normals
8      pcl::copyPointCloud (*cloud_out, *cloud_with_normals);
9      pcl::NormalEstimation<PointTypeIO, PointTypeFull> ne;
10     ne.setInputCloud (cloud_out);
11     ne.setSearchMethod (search_tree);
12     ne.setRadiusSearch (1.0);
13     ne.compute (*cloud_with_normals);
14
15     // Set up a Conditional Euclidean Clustering class
16     pcl::ConditionalEuclideanClustering<PointTypeFull> cec (true);
17     cec.setInputCloud (cloud_with_normals);
18     cec.setConditionFunction (&customRegionGrowing);
19     cec.setClusterTolerance (500.0);
20     cec.setMinClusterSize (cloud_with_normals->size () / 1000);
21     cec.setMaxClusterSize (cloud_with_normals->size () / 5);
22     cec.segment (*clusters);
23     cec.getRemovedClusters (small_clusters, large_clusters);
24
25     // Using the intensity channel for lazy visualization of the output
26     for (int i = 0; i < small_clusters->size (); ++i)
27         for (int j = 0; j < (*small_clusters)[i].indices.size (); ++j)
28             (*cloud_out)[(*small_clusters)[i].indices[j]].intensity = -2.0;
29     for (int i = 0; i < large_clusters->size (); ++i)
30         for (int j = 0; j < (*large_clusters)[i].indices.size (); ++j)
31             (*cloud_out)[(*large_clusters)[i].indices[j]].intensity = +10.0;
32     for (int i = 0; i < clusters->size (); ++i)
33     {
```

```

34         int label = rand () % 8;
35         for (int j = 0; j < (*clusters)[i].indices.size (); ++j)
36             (*cloud_out)[(*clusters)[i].indices[j]].intensity = label;
37     }

```

En el Código 2.3, el primer paso es hacer uso de VoxelGrid mediante la clase **pcl::VoxelGrid** para reducir las muestras de la nube y obtener una densidad de puntos más igualada. Luego se calcularán las normales de cada uno de los puntos de la nube, que más adelante se usarán en la clase **pcl::ConditionalEuclideanClustering** y se aplica el algoritmo que reside en la misma clase.

#### 2.4.4 Eliminación de planos dominantes

En el algoritmo se hará una simple segmentación de plano de un grupo de puntos (Library, s.f.-c), en el Código 2.4 se ve como está implementado. El objetivo es encontrar todos esos puntos en la nube que están dentro del plano dominante, finalmente, se filtrará esa sección y se quedará con el resto de puntos. Este método se consigue haciendo lo siguiente. Primero, se crea un objeto **pcl:: SACSegmentation <pcl::PointXYZRGB>** y se inicializa el modelo y el tipo de método de segmentación, introduciendo como input la nube de puntos y obteniendo los inliers y los coeficientes. Los outputs obtenidos se utilizarán con la nube de puntos como input en el objeto **pcl:: ExtractIndices <pcl::PointXYZRGB>**, que finalmente hará el filtrado de la nube de puntos.

Código 2.4: Código del algoritmo de Plane Model Segmentation.

```

1 float umbral = 0.01;
2 pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);
3 pcl::PointIndices::Ptr inliers (new pcl::PointIndices);
4
5 pcl::SACSegmentation<pcl::PointXYZRGB> seg;
6 seg.setOptimizeCoefficients (true);
7 seg.setModelType (pcl::SACMODEL_PLANE);
8 seg.setMethodType (pcl::SAC_RANSAC);
9 seg.setDistanceThreshold (umbral);
10 seg.setInputCloud (cloud);
11 seg.segment (*inliers, *coefficients);
12
13 pcl::ExtractIndices<pcl::PointXYZRGB> eifilter (true);
14 // Initializing with true will allow us to extract the removed indices
15 eifilter.setInputCloud (cloud);
16 eifilter.setIndices (inliers);
17 eifilter.setNegative (false);
18 eifilter.filter (*cloud);

```

En resumen, en este proyecto se propone la combinación de las dos ramas principales que se han ido explicando en estas últimas secciones con la segmentación de objetos 3D, dado que se hará la detección de objetos mediante Darknet YOLO para luego mediante las coordenadas del bounding box obtener su nube de puntos de la escena donde hayan sido detectados, cuando se tenga la nube de puntos del objeto se aplicará un algoritmo de la Point Cloud Library(PCL) para eliminar los puntos no pertenecientes al objeto como el background o sus alrededores, al mismo tiempo, mediante el mapeado y localización simultáneos de una

escena a partir de una cámara RGB-D y usando el proyecto de ORB-SLAM2 se obtendrán las matrices de transformación entre escenas con la finalidad de aplicárselas a las nubes de puntos. Finalmente, combinando los datos de ambas ramas se obtendrá un mapa con todas los objetos alineados y con mucha más información, ya que incluimos la posición de los objetos que se han detectado. Todo este desarrollo se intentará realizar en ROS para crear un pipeline que se desarrolle en tiempo real.

### 3 Objetivos

La importancia de encontrar una solución a este proyecto se puede mencionar en dos ideas. La primera es la segmentación de objetos 3D, donde primero se detectarán objetos en imágenes 2D para luego segmentarlas de la nube de puntos con el objetivo de eliminar los puntos no pertenecientes al objeto. La segunda idea que se intenta buscar solución en este proyecto es la combinación de la localización y el mapeado simultáneos y la detección de objetos, para poder crear una escena que contenga más información para dar a los robots que use en el mapa.

La motivación personal de este proyecto se basa en el aprendizaje de nuevas técnicas de programación y desarrollar los conocimientos aprendidos de las asignaturas que están involucradas en la realización de este trabajo. De esta manera, la motivación personal ha sido:

- Mejorar el nivel del lenguaje de programación C++.
- Emplear técnicas o herramientas de tratamiento de imagen como OpenCV y nubes de puntos como la PCL.
- Profundizar en el uso de ROS y aprender nuevos conceptos para poder incorporarlos al proyecto.
- Conocer los principales modelos de detección de objetos.
- Conocer los modelos de RGB-D SLAM y usar el que mejor se adapte a este proyecto.

De esta forma, podemos reunir los objetivos de la siguiente lista:

- Usar los topics de ROS de un robot, como los de la cámara para detectar elementos interesantes de una escena, como objetos y personas, y segmentar su información 3D de las nubes de puntos generadas.
- Detección de objetos en imágenes 2D mediante redes neuronales, para obtener los parámetros de la bounding box detectada en cada objeto que serán utilizados en el tratamiento de nubes de puntos.
- Tratamiento de nubes de puntos para segmentar los objetos en formato 3D, además de eliminar los puntos de la nube que no pertenezcan al objeto.
- Mapeado y localización usando el paquete de ORB-SLAM2 en ROS.
- Desarrollar un sistema en tiempo real que integre todo lo mencionado anteriormente.



# 4 Metodología

En este apartado se va hablar de los elementos y herramientas que hacen funcionar nuestro trabajo.

## 4.1 Setup

En esta primera sección se hablará de la configuración y de las herramientas, como librerías y recursos de programación que se usan en este proyecto:

- Sistema operativo Ubuntu 16.04 en VMware Workstation.
- ROS Kinetic.
- Lenguaje de programación C ++.
- Point Cloud Library (PCL), versión 1.8.0.
- OpenCV 3.3.1-dev.
- CMake, una versión igual o superior a la 2.8.
- Paquetes de Robot Operating System (ROS) de ORB-SLAM2 y YOLO Darknet.

## 4.2 ROS

Para poder explicar gran parte del proyecto es necesario explicar que es ROS y su funcionamiento, para entender más a fondo las partes que engloba todo el trabajo.

### 4.2.1 ¿Qué es ROS?

ROS son las siglas de Robot Operating System que es un middleware robótico, el cual es usado para el desarrollo de software de robots y empezó a desarrollarse en 2007. Es un sistema de open-source y que actualmente funciona sobre plataforma de sistema UNIX (Ubuntu-Linux), pero se está adaptando a otros sistemas como Mac OS X, Microsoft Windows y muchos más.

A pesar de no ser un sistema operativo de tiempo real, ROS provee de varias herramientas y los servicios estándar de uno de estos. Por ejemplo, la abstracción del hardware, el paso entre mensajes entre procesos, el mantenimiento de paquetes, control de dispositivos de bajo nivel y la implementación de funcionalidad de uso común. También es posible integrarlo con código en tiempo real.

## 4.2.2 Objetivos de ROS

El objetivo principal de ROS es dar soporte y reutilizar código para el desarrollo e investigación de la robótica. También tiene como objetivo la independencia de lenguaje, tanto C++ y Python ya tienen librerías implementadas. Además, ROS permite un testeo más rápido para poder facilitar el aislamiento de los diferentes componentes del sistema. ROS también es un sistema escalable, siendo apropiado para tiempos de ejecución grandes y grandes procesos de desarrollo.

## 4.2.3 Principales conceptos

ROS tiene dos niveles de conceptos: el nivel de computación gráfica y el nivel del sistema de archivos.

### 4.2.3.1 Conceptos de sistemas de archivos

Los conceptos dentro del nivel del sistema de archivos hace referencia a los recursos de ROS, por ejemplo:

- **Paquetes:** el paquete es la unidad básica y principal para la organización de software en ROS. Un paquete puede contener los procesos de ejecución de ROS, conocidos como nodos, librerías, dataset, ...
- **Metapaquetes:** el metapaquete es un paquete que están especializados para representar un grupo de paquetes relacionados.
- **Manifiestos del paquete:** son archivos “.xml”, proporcionan metadatos sobre un paquete, como las dependencias u otra información exportada de otros paquetes.
- **Repositorios:** Una colección de paquetes que comparten un sistema VCS común, como la misma versión.
- **Tipos de servicios:** definen la estructura de los datos tanto de solicitud como de respuestas utilizados en los servicios de ROS.
- **Tipos de mensajes:** definen la estructura de los datos enviados en los mensajes de ROS.

### 4.2.3.2 Conceptos de computación gráfica

La Computación Gráfica es la red de procesos de ROS que se están procesando los datos en conjunto de peer-to-peer<sup>1</sup>, donde se puede actuar como cliente y servidor simultáneamente. Los conceptos de este nivel son:

- **Nodos:** un nodo es un ejecutable que usa ROS para comunicarse con otros nodos, los nodos se combinan entre ellos en un grafo y se comunican entre ellos haciendo uso de mensajes que se envían por topics o servicios.

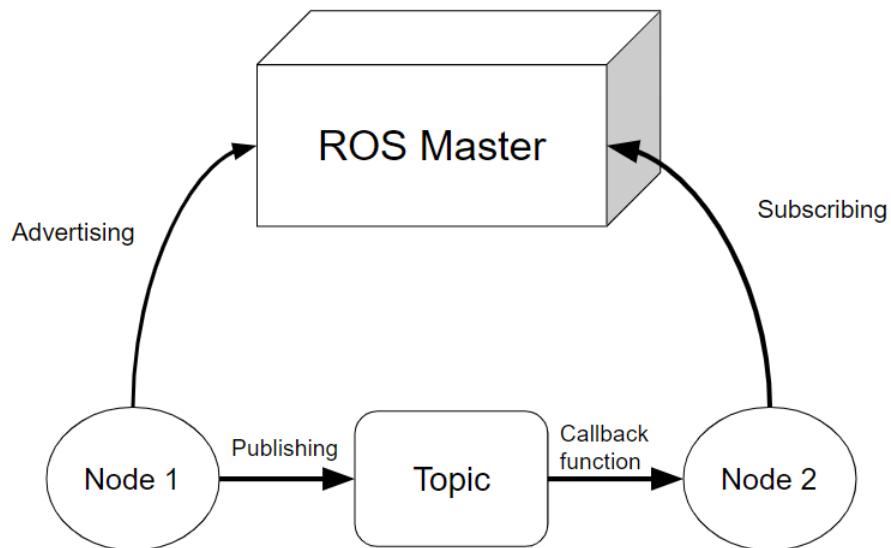
---

<sup>1</sup>Una red peer-to-peer, es una red de ordenadores en la que casi todos los aspectos funcionan sin clientes ni servidores, sino una serie de nodos que se comportan igual entre si.

- **Maestro:** ROS Master provee el nombre y el registro de los servicios al resto de nodos del sistema de ROS. Además, sigue las publicaciones y suscripciones de los topics y los servicios actuando de intermediario.
- **Mensajes:** un mensaje hace referencia a la información que se envía entre dos nodos que se comunican, su contenido puede ser desde un simple “string” hasta un mensaje personalizado que contenga una nube de puntos y una imagen.
- **Tópicos:** Es el canal que utilizan los nodos para comunicarse entre ellos.
- **Servicios:** es otra forma de comunicación que da ROS, pero en este caso lo que se hace es que el nodo envía una petición y se queda esperando la respuesta de dicho servicio.
- **Bags:** la herramienta bag o bolsas de ROS se utiliza para almacenar datos en forma de mensajes con el objetivo de reproducirlos en tiempo real posteriormente.

#### 4.2.4 Funcionamiento

A continuación, se explica cómo trabajan los conceptos para entender el funcionamiento de ROS y en la Figura 4.1 se muestra la arquitectura básica de ROS.



**Figura 4.1:** Arquitectura básica del funcionamiento de ROS.

El maestro es el que hace que haya comunicación entre ambos nodos, sin él los nodos no serían capaces de establecer una comunicación entre ellos, para poder tener el maestro activo en la terminal hay que escribir el comando **roscore**. ROS maestro actúa como un servicio de nombre en el nivel de Computación Gráfica, almacenando los servicios de información para los nodos. A medida que los nodos se comunican con el maestro, pueden recibir información o mensajes sobre otros nodos que se hayan inscrito al maestro y poder hacer una conexión entre

ambos. El maestro sólo proporciona información de búsqueda, haciendo uso de un protocolo de comunicación llamado TCPROS, que usa conexiones TCP e IP para el transporte de mensajes.

Los nodos envían datos en forma de mensajes haciendo uso de los topics como canal de comunicación. Los nodos pueden tener dos funciones, la primera es suscribirse a un topic para recibir mensajes de él y puede publicar mensajes en el topic, para que un suscriptor recoja esos mensajes. Un nodo no puede inscribirse ni publicar en el mismo topic, pero sí que pueden suscribirse a un topic y publicar en otro diferente.

Cuando un nodo se suscribe a un topic, su proceso queda parado hasta que otro nodo publica en ese topic, entonces se ejecuta la función callback que está relacionada al nodo que se ha suscrito a ese topic. Por ejemplo, un nodo se ha suscrito a un topic que recibe mensajes de nubes de puntos, cuando el mensaje llega a ese topic la función callback recoge el mensaje y se ejecuta el código de la función.

La comunicación entre topics en ROS para el intercambio de mensajes es asíncrona, que es una comunicación que es diferida en el tiempo, es decir, que no existe coincidencia temporal o no hay intervención de las dos partes. Los elementos importantes son el emisor que envía la información, el receptor que recibe los mensajes y el canal que es el medio físico por el que se transmite el mensaje. En cambio si se hacen uso de servicios la comunicación es síncrona, consiguiendo un intercambio de información en tiempo real. Por ejemplo, su uso podría darse cuando un robot necesita una respuesta o unos datos antes de continuar con el trabajo que estaba realizando.

### 4.3 Point Cloud Library (PCL)

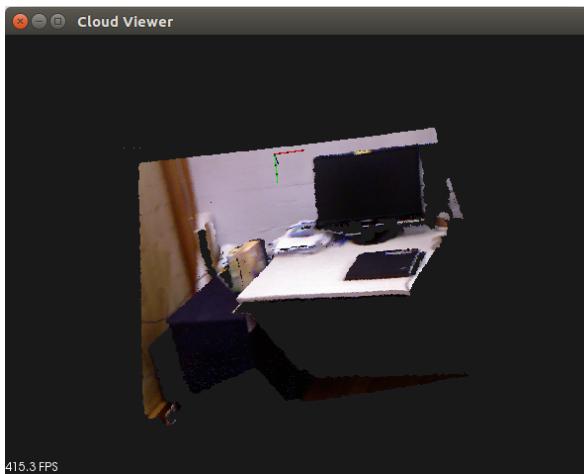
Point Cloud Library (PCL) (Rusu y Cousins, 2011) es una librería open-source de algoritmos para el tratamiento de nubes de puntos, geometría 3D e imágenes 2D o 3D, que está escrito en C++. Contiene algoritmos de filtrado, estimación de características, registro 3D, reconocimiento de objetos, segmentación o también la visualización como en la Figura 4.2.

Cada conjunto de algoritmos se define a través de la clase base, manteniendo así las implementaciones de los algoritmos compactas y limpias. Para simplificar la PCL se divide en módulos, que están implementados en una librerías más pequeñas, eso ayuda a que se puedan usar diferentes librerías y solo compilar esas y no todas. El PCL también contiene su propio formato para guardar nubes de puntos, pero también permite cargar y guardar datos en otros formatos. Las nubes de puntos se pueden adquirir a partir de sensores de hardware, como cámaras estéreo, escáneres 3D o cámaras de Time Of Flight, o se pueden generar a partir de un programa informático sintéticamente. Por ejemplo, la Figura 4.2 muestra una visualización de nube de puntos que proviene de uno de los datasets que se usan en el proyecto.

PCL es una librería que tiene un gran abanico de usos y es usada en diferentes campos de ciencia, algunos ejemplos es el reconocimiento de objetos 3D por su forma geométrica o filtrar el ruido de una nube de puntos. Para poder usar esta librería también se requiere la instalación de varias tercera librerías como Boost, Eigen, FLANN o VtK (Visualization ToolKit), con las versiones que se requieran.

En este trabajo se harán uso de algoritmos para la segmentación como el Region Growing Segmentation, el Color-Based Region Growing Segmentation, el Conditional Euclidean Clustering o la eliminación de planos dominantes para obtener los objetos de las nubes de

---



**Figura 4.2:** Visualización de una nube de puntos.

puntos, extractores de características de nube de puntos, computar descriptores, algoritmos de emparejamientos o el Iterative Closest Point para el pipeline del registro de par de nubes.

## 4.4 OpenCV

OpenCV o Open Computer Vision es una librería open source de análisis de imágenes, visión de computador y aprendizaje automático. Esta librería consta de múltiples algoritmos que permiten realizar un gran variedad de actividades como reconocer objetos o identificar rostros. OpenCV es una librería multiplataforma que puede ser utilizada en Windows, Mac, Linux y Android. Su API es de C++, pero también puede programarse tanto en Python, Java y Matlab.

OpenCV se estructura en cinco puntos principales:

- El componente CV que contiene los algoritmos de procesamiento de imágenes y de visión por ordenador.
- El componente ML que es la biblioteca de machine learning.
- HighGUI que contiene funciones de entrada y salida para el almacenamiento y carga de vídeo e imágenes.
- CXCore contiene las estructuras básicas, algoritmos y funciones básicas.
- CvAux contiene algunos algoritmos experimentales.

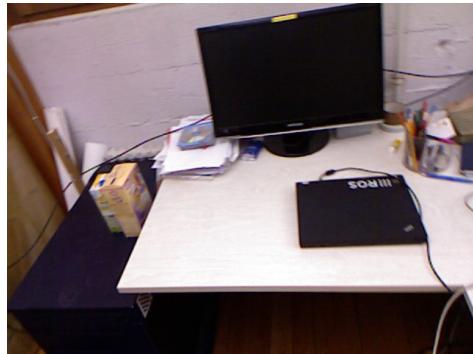
En este proyecto será uno de los requisitos que demanda el uso de YOLO Darknet para poder hacer la detección y clasificación de objetos.

## 4.5 Datasets de prueba

Para la realización de las diferentes pruebas se utilizan varios datasets. Los que más se han utilizado son los de los archivos bag que proporciona el Computer Vision Group of Technical

---

University of Munich (TUM) (Sturm y cols., 2012). Estos archivos proporcionan la secuencia de un vídeo grabado en una habitación u otro escenario como un escritorio, una gran sala o un pasillo. Los datos que incluye este archivo bag de ROS son topics que dan información de las imágenes captadas por la cámara tanto en RGB como en profundidad. Las nubes de puntos generadas por una cámara de profundidad. En algunos casos también se incluyen las



**Figura 4.3:** Ejemplo de imagen del archivo bag de ROS.

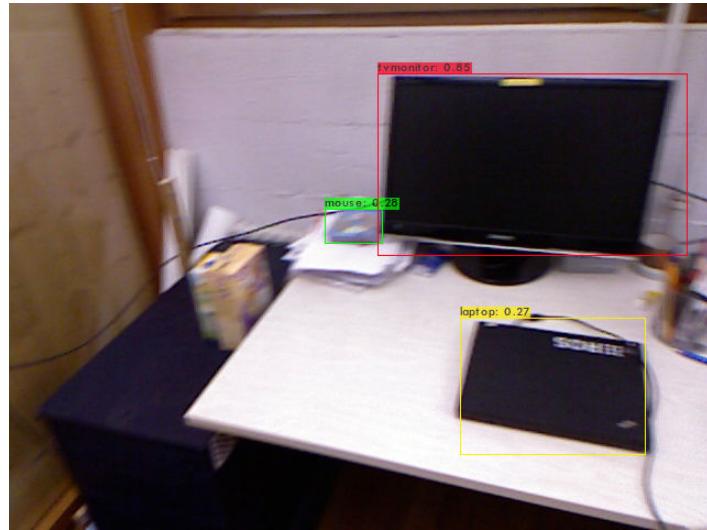
transformaciones de la cámara y si en estos archivos contiene el topic “`/camera/rgb/camera_info`” no será necesario hacer la calibración de la cámara, ya que este topic proporciona los parámetros intrínsecos de ella. En la Figura 4.3, se muestra una imagen que proporciona el archivo ROS bag.

También se hace uso del dataset que proporcionan los desarrolladores del ORB-SLAM2, dado que incluyen los mismos topics que se mencionan anteriormente y es otra manera de probar el sistema desarrollado.

## 4.6 YOLO Darknet

Para la detección de objetos, se hace uso de Darknet (Redmon, 2013–2016). Es una red neuronal open-source que está escrita en C, es rápida, fácil de instalar y soporta computación tanto en CPU y GPU. El framework trabaja con YOLO, un sistema de detección de objetos en tiempo real. Darknet es capaz de clasificar imágenes de unas 1000 clases. Además, muestra información a medida que carga el archivo de configuración y los pesos, luego clasifica la imagen como en la Figura 4.4 y muestra las clases de los objetos detectados.

---



**Figura 4.4:** Ejemplo de predicción de imágenes de Darknet.

#### 4.6.1 Paquete de ROS para YOLO-Darknet

Para este proyecto, se necesita un paquete de ROS que contenga dicha red neuronal. Por lo tanto, se hace uso del paquete de (Bjelonic, 2016–2018). Este paquete se ha desarrollado para la detección de objetos en cámaras y hace uso de Yolo(V3) en GPU y CPU. Además, que es una red neuronal open-source. El modelo pre-entrenado de la red neuronal es capaz de detectar las clases pre-entrenadas incluidas en el dataset de VOC y COCO, pero también se puede reentrenar para detectar las clases del usuario. Para poder usar este paquete se necesita OpenCV y la librería Boost de C ++, además de ROS.

Para poder hacer uso de este paquete hay que hacer los siguientes pasos:

- **catkin\_make**, para poder compilar el paquete
- **source devel/setup.bash**, para poder lanzar los archivos del paquete es necesario configurar el “setup.bash”, ya que sin eso ROS no los podría lanzar en ese directorio.
- **roslaunch darknet\_ros darknet\_ros.launch**, esto lanza la red neuronal y espera poder leer de un topic de imágenes para poder empezar a detectar objetos en ellas.

##### 4.6.1.1 Parámetros

Los parámetros que se necesitan para poder lanzar la red neuronal son los siguientes:

- Archivos de configuración y pesos de YOLO.
- Archivos de configuración de ROS y de la network.
- Los nodos y programas necesarios para poder hacer funcionar el paquete.

#### 4.6.1.2 Topics de entrada y salida

El paquete para poder funcionar necesita la entrada de imágenes, esto lo consigue leyendo del topic `/camera/rgb/image_raw`, que es donde se publicarán las imágenes para poder luego procesarlas y detectar los objetos que hay en ella.

Las salidas que tiene este paquete las divide en tres diferentes topics. El primero publica el número de objetos detectados en la imagen procesada, el segundo publica un mensaje personalizado de los bounding boxes que se encuentra en la carpeta de `darknet_ros_msgs`, este mensaje contiene el número de objetos detectados, la clase de cada uno, los porcentajes de probabilidad de que sean de esa clase y los puntos x e y de donde empieza y donde acaba el bounding box de cada objeto. El tercer topic publica la imagen con los bounding boxes dibujados sobre ella.

Todos los tipos de mensajes aparte de llevar su mensaje como una imagen, una nube de puntos o la información de los parámetros intrínsecos de una cámara, también llevan una sección reconocida como “Header”, que se usa para generalmente para comunicar datos con marca de tiempo en un marco de coordenadas particular. Estos datos nos proporcionan la siguiente información:

- **uint32 seq:** es el número de secuencia o ID que está asociado al mensaje.
- **time stamp:** es el tiempo asociado a ese mensaje.
- **string frame\_id:** es el marco con el que están asociados estos datos.

Cabe destacar que el time-stamp es un dato que se usará bastante en este proyecto, ya que se usará como dato para poder sincronizar diversos topics. El time-stamp es un tipo de dato “Unix Epoch Time”, es el tiempo que coge tu sistema y que ha transcurrido desde el 1 de enero de 1970. Por ejemplo, si tenemos un time-stamp de un valor de 1614423714, en milisegundos sería 1614423714000 y con este valor se podría sacar la fecha exacta de cuando fue tomado un mensaje.

#### 4.6.1.3 Acciones

Este paquete también provee de una acción, la cual se le envía una imagen y te devuelve un array o vector de los bounding boxes de los objetos detectados en ella.

### 4.7 Registro por parejas de nubes de puntos

Para lograr el registro por parejas de nubes de puntos hacemos uso de Random Sample Consensus (RANSAC), que es un algoritmo que de manera iterativa, estima los parámetros de manera robusta. En el caso que ocupa a este trabajo, se necesita estimar los parámetros de la matriz de transformación que mejor se ajusta a los emparejamientos de los descriptores de los puntos. Todos los elementos que se van a usar se dan gracias a la Point Cloud Library(PCL).

El primer paso es adquirir dos nubes de puntos consecutivas con las que se pueda trabajar, estas tendrán que ser del mismo tipo de datos para que no haya ningún error. Tras esto, se eliminarán los punto Not A Number(NaN).

Tras haber adquirido las nubes, se deberá calcular los puntos característicos o keypoints de cada a una, esto se hará mediante un extractor de características. Estos métodos de extracción de puntos se conocen como detectores y reciben una nube de puntos y devuelven otra nube de puntos con los keypoints. La PCL integra muchos métodos para detectar keypoints: Harris 3D, SIFT, ISS, Uniform Sampling, etc.

A continuación, se deberán calcular las normales de ambas nubes, que después se utilizarán en el momento de computar los descriptores.

Los descriptores son métodos que admiten una nube de puntos, en este caso será la nube de puntos de keypoints, y devuelven, para cada descriptor, el descriptor correspondiente. Un descriptor es una manera de codificar patrones geométricos del espacio en ese punto teniendo en cuenta cierta vecindad. A nivel de implementación es un vector de números. Estos vectores de números permiten comparar un descriptor con otro de forma fácil para saber si son iguales y así poder establecer correspondencias. La PCL integra varios métodos para computar descriptores: PFH/FPFH, RSD, SHOT, etc.

Los descriptores se pueden clasificar en:

- **Descriptores locales:** Son calculados para cada punto característico de manera individual. Describen cómo es la geometría alrededor de un punto y la vecindad es cercana al punto.
- **Descriptores globales:** Son calculados para un cluster completo que representa a un objeto. Codifican la geometría de un objeto.

El siguiente paso es calcular los emparejamientos entre ambas nubes de puntos. Normalmente esto se hace metiendo en un KdTree los descriptores de la nube más grande (la de la escena). Luego se iteran todos los descriptores de la nube pequeña (la del objeto) y se consulta al vecino más cercano usando el KdTree.

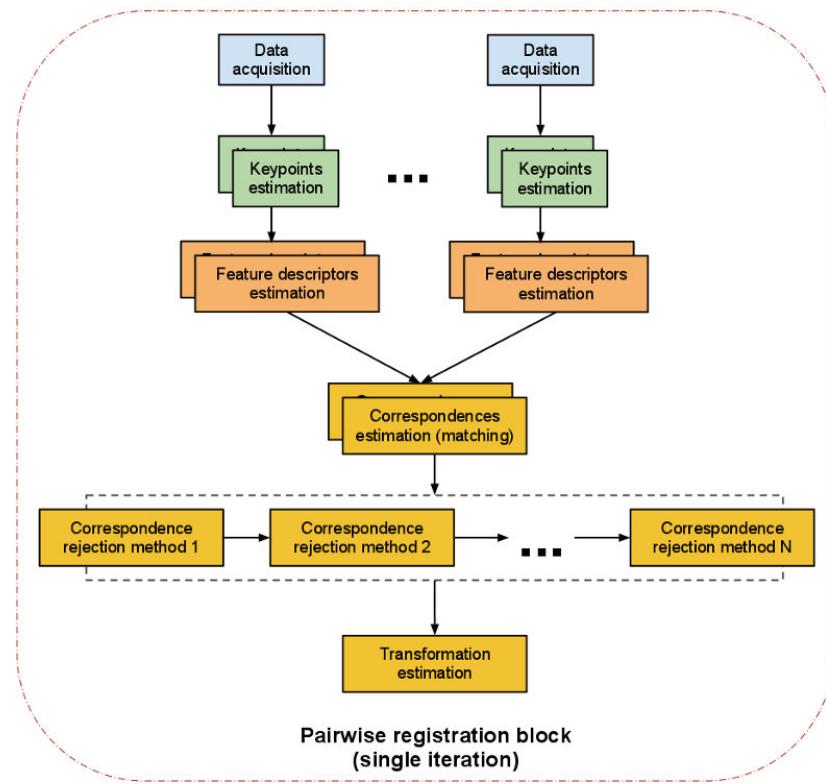
Un árbol KdTree (árbol k-dimensional) es una estructura de datos que divide el espacio y almacena un conjunto de puntos k-dimensionales en una estructura de árbol que permite búsquedas de rango eficientes y búsquedas de vecinos más cercanos.

Lo normal es que existan emparejamientos buenos y malos o falsos, y los falsos hay que filtrarlos. Para eliminar los falsos emparejamientos se usa RANSAC. Se toman tres pares de correspondencias, se calcula la transformación de la nube para alinearla con la otra y se evalúa el número de inliers en un umbral. Este proceso se repite n veces hasta obtener la mejor transformación, que es la que mayor número de inliers tiene.

Normalmente, la transformación que se obtiene de RANSAC suele ser buena, pero no la mejor. Por lo tanto, se aplica un refinamiento a las nubes de puntos, para obtener la mejor matriz de transformación. Esto se logra mediante Iterative Closest Point (ICP), pero para poder utilizar este método es necesario partir de una estimación inicial muy buena, por eso se le aplica la matriz de transformación obtenida de RANSAC a la nube puntos.

Unos problemas que surge con este método es que es muy costoso computacionalmente y tiene poca capacidad de generalización al tener muchos parámetros.

La estructura de pasos del registro de parejas de nubes de puntos queda tal como la Figura 4.5.

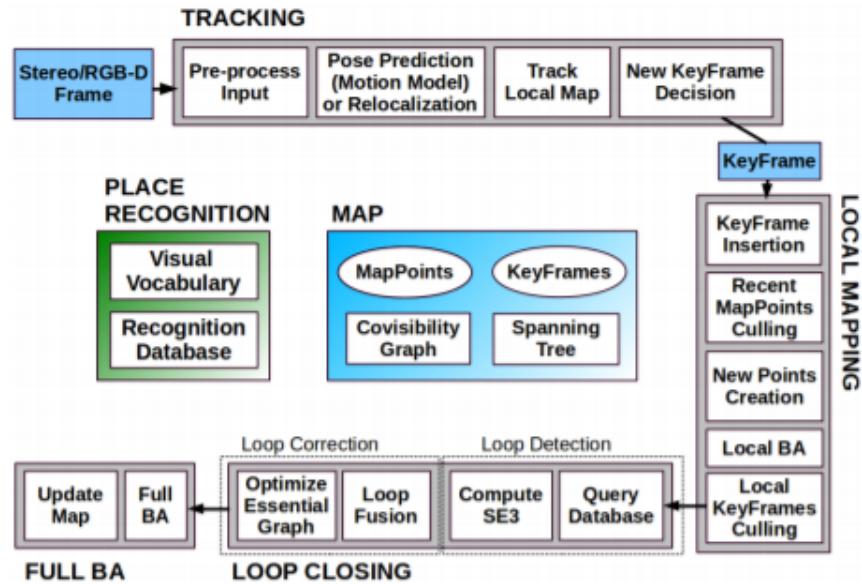


**Figura 4.5:** Estructura del pipeline. Fuente: (*PCL pointcloud pairwise registration*, s.f.)

## 4.8 ORB SLAM 2

Como se ha explicado anteriormente, ORB-SLAM2 (Mur-Artal y Tardós, 2017) es una librería de SLAM para cámaras RGB-D, Monocular o estéreo que calcula la trayectoria de la cámara, también es capaz de detectar bucles cerrados y relocalizar la cámara en tiempo real. Para poder hacer uso de esta librería es necesario instalar unos requisitos. Estos requisitos son librerías obligatorias como OpenCV para la manipulación de imágenes y características o Eigen3 u opcionales como ROS.

ORB-SLAM2 está basado en el antiguo trabajo de ORB-SLAM para cámaras monoculares. Este sistema tiene tres hilos principales, el primero es el rastreo de la localización de la cámara en cada uno de los frames encontrando emparejamientos en el mapa y reduciendo el error de reproyección, el segundo hilo es el mapeo local y el tercer y último hilo es de detectar que el bucle se cierra y corregir el error acumulado para optimizar el mapa. Este sistema hace uso de la información de las imágenes de profundidad para combinarla con las características extraídas de las imágenes, estas se consigue mediante un extractor de características, el programa hace uso de Oriented FAST and Rotated BRIEF (ORB). Estas características son robustas a la rotación y escala. Además, presentan una invariancia a la ganancia automática y exposición automática de la cámara, y cambios de iluminación. La Figura 4.6 muestra el esquema de los hilos de ORB-SLAM2 de una manera más gráfica.



**Figura 4.6:** Estructura de hilos de ORB-SLAM2. Fuente: (Mur-Artal y Tardós, 2017)

### 4.8.1 ORB-SLAM2 para ROS

En este proyecto se utilizará el paquete basado en el ORB-SLAM2 pero para ROS, el cual pueda proveer información mediante publicación de mensajes en topics. Este paquete es cogido de GitHub<sup>2</sup>.

Las características principales que presenta este paquete son que tiene compatibilidad total con ROS, soporta diferentes modelos y tipos de cámaras, los datos de entrada y salida se hacen vía topics, los parámetros se pueden configurar con rqt, que es un framework de ROS que proporciona herramientas de interfaz, durante el tiempo de ejecución, la carga de parámetros de la cámara se puede hacer vía un archivo externo que se coja en el archivo launch o usando el topic llamado “cam\_info”.

#### 4.8.1.1 Parámetros

En este paquete hay tres tipos de parámetros: los estáticos y dinámicos de ROS y los parámetros de la cámara. Los estáticos son parámetros que son enviados al servidor de ROS y no se pueden cambiar, estos parámetros se colocan en archivos launch que se encuentran en la carpeta launch del paquete. En estos parámetros se puede observar strings o cadenas de caracteres que hacen referencia a varios archivos de configuración que se necesitan cargar o algunos booleanos para poder activar ciertos topics o servicios.

Los parámetros dinámicos son aquellos que se pueden cambiar durante el tiempo de ejecución, estos se pueden actualizar vía comandos de terminal. También se configuran en los archivos launch mencionados anteriormente.

Finalmente, los parámetros de la cámara son los parámetros intrínsecos de calibración, estos se pueden encontrar en la carpeta “config” y se especifican en archivos “.yaml”.

<sup>2</sup>[https://github.com/appliedAI-Initiative/orb\\_slam\\_2\\_ros](https://github.com/appliedAI-Initiative/orb_slam_2_ros)

#### 4.8.1.2 Topics de entrada y salida

Hay dos tipos diferentes de topics, los de entrada que son los datos que recibe el algoritmo para poder trabajar y los de salida que son los datos que se obtienen durante la ejecución del paquete de ORB-SLAM2.

En este proyecto se utilizará el nodo para cámaras RGB-D. Por lo tanto, las entradas que habrán en este algoritmo son las siguientes:

- `/camera/rgb/image_raw` para la imagen RGB.
- `/camera/depth_registered/image_raw` para la información de profundidad.
- `/camera/rgb/camera_info` para la información de los parámetros de la calibración de la cámara, si `load_calibration_from_cam` es true/cierto. El `load_calibration_from_cam` es un booleano que se encuentra dentro del código que comprueba si hay un topic con el nombre de `/camera/rgb/camera_info` que si es true leerá los parámetros intrínsecos de la cámara de ese topic, en cambio si no está es topic los parámetros se cargarán a partir de un archivo que se adjunte.

Los topics que publicarán los datos de salida son los siguientes:

- Una nube de puntos (PointCloud2), que contiene los puntos característicos de la imagen procesada.
- La posición actual (PoseStamp) del frame de la cámara.
- La imagen (Image) con los puntos característicos encontrados.
- El tf, que es el vector de traslación y un quaternion del id frame de la nube de puntos al id frame de la cámara.

Tanto el topic que publica el PoseStamp como el tf hacen la misma publicación, pero en distinto formato. El topic tf publica si en el archivo bag que se está ejecutando no hay un topic que se llame igual que él.

#### 4.8.1.3 Servicios

Los servicios que presta este paquete son tres, que son para poder guardar el mapa dependiendo del nodo que se ejecute, dado que este paquete soporta cámaras monoculares, Stereo y RGB-D. Este servicios se puede ejecutar usando el siguiente comando:

- `rosservice call /orb_slam2_rgbd/save_map map.bin`
- `rosservice call /orb_slam2_stereo/save_map map.bin`
- `rosservice call /orb_slam2_mono/save_map map.bin`

El archivo se guardará en ROS\_HOME, que por defecto es `/.ros`.

# 5 Desarrollo

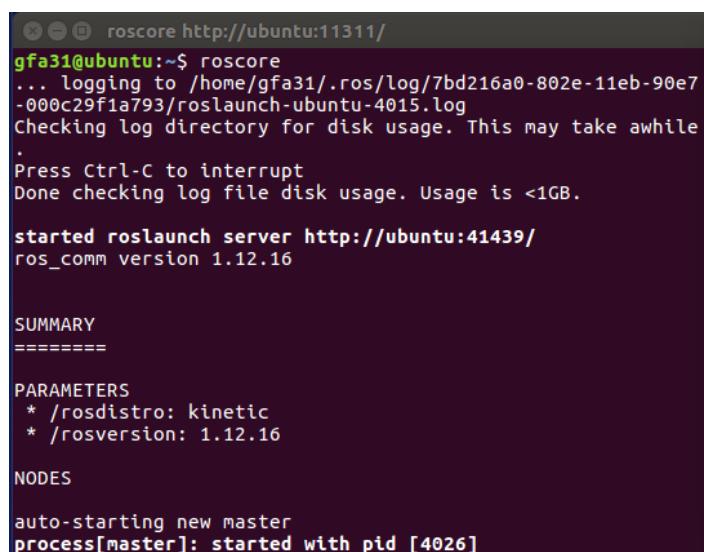
En este capítulo, primero se explicarán cómo funcionan los métodos que anteriormente se han presentado y posteriormente se va a explicar cómo se implementan los elementos y herramientas explicadas anteriormente y de qué manera se integran para lograr el funcionamiento del pipeline general del proyecto.

## 5.1 Desarrollo del pipeline

En estos párrafos se va a describir el pipeline general de los programas realizados para conseguir el objetivo del proyecto propuesto. Para llevar a cabo este objetivo es necesario tener un archivo bag que contenga tanto las imágenes como las nube de puntos de la escena, además de que la cámara esté calibrada. Esto se puede si existe el topic “/camara\_info” en el archivo bag. También es necesario tener los paquetes de Darknet y ORB-SLAM2 para ROS que se han descrito en secciones anteriores, para poder hacer la detección de objetos en escenas y la localización y mapeado simultáneos.

Para su mejor uso todos los nodos se han realizado en el mismo paquete de Darknet ROS, así que mediante el uso de un archivo launch se lanzan todos los nodos necesarios para llevar a cabo el trabajo, menos el nodo de ORB-SLAM2 que se tendrá que lanzar en una terminal diferente. El funcionamiento es el siguiente:

- Ejecutar el servicio maestro de ROS: **roscore** (Figura 5.1).



```
roscore http://ubuntu:11311/
gfa31@ubuntu:~$ roscore
... logging to /home/gfa31/.ros/log/7bd216a0-802e-11eb-90e7
-000c29f1a793/roslaunch-ubuntu-4015.log
Checking log directory for disk usage. This may take awhile
.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:41439/
ros_comm version 1.12.16

SUMMARY
=====

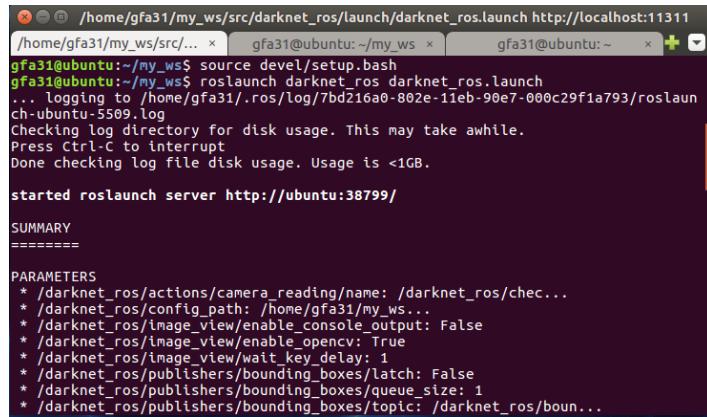
PARAMETERS
  * /rosdistro: kinetic
  * /rosversion: 1.12.16

NODES

auto-starting new master
process[master]: started with pid [4026]
```

Figura 5.1: Terminal lanzando roscore.

- Ejecutar el comando para lanzar los nodos necesarios para poder trabajar con el archivo launch: **roslaunch darknet\_ros darknet\_ros.launch** (Figura 5.2).

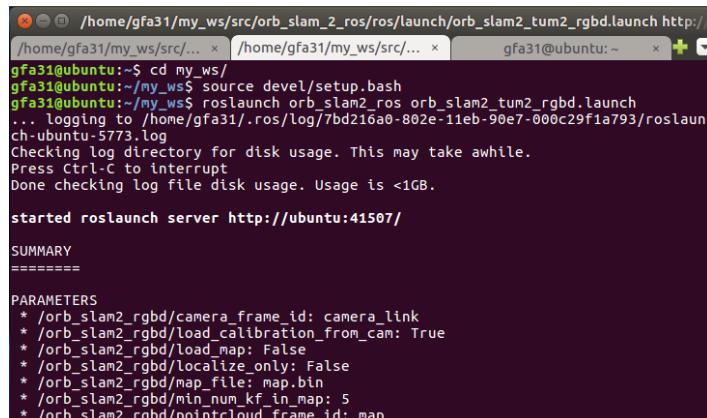


```
/home/gfa31/my_ws/src/darknet_ros/launch/darknet_ros.launch http://localhost:11311
[rosrun] [darknet_ros]: /home/gfa31/my_ws/src/darknet_ros/launch/darknet_ros.launch
gfa31@ubuntu:~/my_ws$ source devel/setup.bash
gfa31@ubuntu:~/my_ws$ rosrun darknet_ros darknet_ros.launch
... logging to /home/gfa31/.ros/log/7bd216a0-802e-11eb-90e7-000c29f1a793/roslaunch-ubuntu-5509.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:38799/
SUMMARY
=====
PARAMETERS
  * /darknet_ros/actions/camera_reading/name: /darknet_ros/check...
  * /darknet_ros/config_path: /home/gfa31/my_ws...
  * /darknet_ros/image_view/enable_console_output: False
  * /darknet_ros/image_view/enable_opencv: True
  * /darknet_ros/image_view/wait_key_delay: 1
  * /darknet_ros/publishers/bounding_boxes/latch: False
  * /darknet_ros/publishers/bounding_boxes/queue_size: 1
  * /darknet_ros/publishers/bounding_boxes/topic: /darknet_ros/boun...
```

**Figura 5.2:** Terminal lanzando Darknet-ROS.

- Ejecutar el paquete de ORB-SLAM2: **roslaunch orb\_slam\_2\_ros orb\_slam2\_tum2\_rgbd.launch** (Figura 5.3).

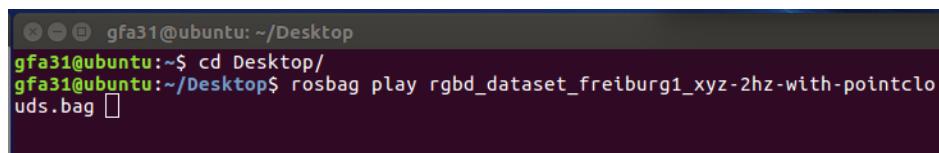


```
/home/gfa31/my_ws/src/orb_slam_2_ros/launch/orb_slam2_tum2_rgbd.launch http://
[rosrun] [orb_slam2_rgbd]: /home/gfa31/my_ws/src/orb_slam_2_ros/launch/orb_slam2_tum2_rgbd.launch
gfa31@ubuntu:~$ cd my_ws/
gfa31@ubuntu:~/my_ws$ source devel/setup.bash
gfa31@ubuntu:~/my_ws$ rosrun orb_slam2_rgbd orb_slam2_tum2_rgbd.launch
... logging to /home/gfa31/.ros/log/7bd216a0-802e-11eb-90e7-000c29f1a793/roslaunch-ubuntu-5773.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:41507/
SUMMARY
=====
PARAMETERS
  * /orb_slam2_rgbd/camera_frame_id: camera_link
  * /orb_slam2_rgbd/load_calibration_from_cam: True
  * /orb_slam2_rgbd/load_map: False
  * /orb_slam2_rgbd/localize_only: False
  * /orb_slam2_rgbd/map_file: map.bln
  * /orb_slam2_rgbd/min_num_kf_in_map: 5
  * /orb_slam2_rgbd/pointcloud_frame_id: map
```

**Figura 5.3:** Terminal lanzando ORB-SLAM2-ROS.

- Lanzar el archivo bag que contiene todos los datos de entrada: **rosbag play archivo.bag** (Figura 5.4).



```
gfa31@ubuntu:~/Desktop
gfa31@ubuntu:~$ cd Desktop/
gfa31@ubuntu:~/Desktop$ rosbag play rgbd_dataset_freiburg1_xyz-2hz-with-pointclouds.bag
```

**Figura 5.4:** Terminal lanzando archivo bag.

Con el comando “rosbag play archivo.bag”, se pueden usar diferentes argumentos para poder ejecutar el archivo de diferentes maneras. Por ejemplo, los más importantes para este proyecto serían el comando que puede modificar la frecuencia de publicación de mensajes con “-hz=HZ” o el de empezar en un tiempo diferente al 0 con el comando “-s SEC o –start=SEC”, siendo “SEC” el segundo donde empezar la ejecución del archivo.

Una vez explicado el funcionamiento general, se va a entrar en detalle en cada uno de los nodos que se han desarrollado para conseguir el objetivo propuesto en este proyecto.

### 5.1.1 Lectura de topics

El primer nodo que se ha desarrollado, el cual se le conocerá como “principal”, tiene dos funciones. La primera función es leer de manera sincronizada mensajes de dos topics, uno de imágenes (“**/camera/rgb/image\_color**”) y otro de nube de puntos (“**/camera/rgb/points**”), esto se consigue mediante el uso de “message\_filters” y la política de Approximate Time Policy, que hará que los mensajes que capten los suscriptores sean de un time-stamp similar, así coger la imagen y la nube de puntos de la misma secuencia. La segunda función es recoger el mensaje del bounding box del nodo de Darknet y enviar un mensaje con el bounding box de los objetos y de la nube de puntos a la que corresponde al siguiente nodo para procesar la escena.

Las funciones callbacks y variables que se vayan a usar están incluidos en una clase, los callbacks se encuentran en la parte pública para poder llamarlos en el main y las variables en la parte privada para poder utilizarlo entre diferentes callbacks que haya en el programa. Así que en el main se creará un objeto de dicha clase para poder acceder a las funciones callbacks y evitar tener problemas entre variables de distintas funciones.

Primero, en el main se hará uso de la política de sincronización de topics para suscribirse a los topics de la imagen y la nube de puntos y recibir los mensajes. Cuando estos lleguen, se llamará a la función callback asociada a esos suscriptores. Además, se inicializarán todos los “publishers” y “subscriber” necesarios para publicar mensajes. Seguidamente, se muestra ejemplos de “subscriber” y “publisher”.

- ros::Publisher sec\_pub = nh.advertise <darknet\_ros\_msgs ::Secuencia> (“/topic”, 20);
- ros::Subscriber cood\_sub = nh.subscribe (“/darknet\_ros/bounding\_boxes”, 20, &MyClass::callback\_bb, &myclass\_object);

Cuando entre a la función callback, que sincroniza los dos topics de imagen y nube de puntos, la imagen recogida se publicará en el topic correspondiente para enviarla al nodo de Darknet (“**/camera/rgb/image\_raw**”) y poder detectar los objetos que hay en ella. Además, se guardará la nube de puntos en un mensaje personalizado que consta de dos partes: la nube de puntos y el mensaje que se obtiene de los bounding boxes que proviene del nodo de Darknet (“**/darknet\_ros/bounding\_boxes**”). En el momento que se haya publicado el mensaje de la imagen para enviarlo al nodo de Darknet, una variable booleana pasará a “true” para evitar que la nube de puntos se sobrescriba y se envíe una nube de puntos diferente con un mensaje de bounding boxes de otra nube de puntos. Hay que recordar que si la imagen no tiene ningún objeto que pueda ser detectado por la red neuronal, todo este

pipeline no tendrá ninguna funcionalidad. Por lo tanto, hay que asegurarse que la imagen que se le pase al nodo contenga objetos dentro de la clasificación de la red neuronal.

La variable booleana, que se ha mencionado anteriormente, volverá a cambiar a “false” cuando se haya recibido el mensaje de Darknet con los objetos detectados y se entre a su función callback, ya que en este nodo también hay un suscriptor que recibe los mensajes de los bounding boxes. Por lo tanto, como la variable booleana es “true”, se ha recibido el mensaje de los bounding boxes y se tiene la nube de puntos, se puede publicar en un nuevo topic el mensaje personalizado. Cuando se haya enviado la variable booleana cambiará a “false” y el proceso se irá repitiendo por cada par de imágenes y nube de puntos que vaya llegando al nodo.

El uso de una variable booleana para controlar el flujo de envío de mensajes entre nodos se hace para que no se combinen mensajes de bounding box de objetos detectados en una escena errónea, dado que si esto pasa, primero, se combinarían un mensaje de bounding box y nube de puntos con diferente time-stamp y segundo, en el momento de procesar la nube de puntos, se recortarían puntos donde realmente no se encuentra ningún objeto. Por lo tanto, realizando esta pequeña máquina de estados se controla este nodo para evitar solapamiento de información.

### **5.1.2 Detección de objetos en Darknet**

Este siguiente nodo tiene la función de realizar la detección de objetos en las imágenes que este recibe. El topic que usa para leer los mensajes es “`/camera/rgb/image_raw`”. Este topic se usa en el nodo anterior y publica las imágenes que recoge del rosbag que se está ejecutando en ese mismo momento. Para que no haya ningún problema que haga perder información durante la ejecución, ninguna otra imagen se volverá a publicar hasta que no se obtenga el resultado de los objetos detectados en la imagen mediante un mensaje de ROS. Esto anterior se consigue con la máquina de estados explicada en la Sección 5.1.1.

Este mensaje contiene un “Header”, que se usa para comunicar el time-stamp del mensaje en un frame particular y un vector de otro mensaje personalizado, el cual contiene todos los parámetros necesarios como el nombre de la clase del objeto, el porcentaje de probabilidad que sea ese objeto y los puntos en los ejes x e y que forman el bounding box donde se encuentra el objeto detectado. El nombre del mensaje del vector es “`BoundingBox.msg`” y el mensaje que contiene el vector y el “header” se llama “`BoundingBoxes.msg`”, ambos se encuentran en el paquete de ROS “`darknet_ros_msgs`” y para poder utilizarlos en un programa hay que añadir las librerías `#include <darknet_ros_msgs/BoundingBoxes.h>` y `#include <darknet_ros_msgs/BoundingBox.h>`.

Así que, cuando en el nodo “principal” haya recibido el mensaje de los bounding boxes, este lo guardará en un mensaje personalizado que ya contiene la nube de puntos que está asociada a la imagen que se ha enviado al nodo de Darknet. Este mensaje será enviado al siguiente nodo para comenzar el procesamiento de la nube de puntos para obtener los objetos recortados de ella y eliminar los puntos que no pertenezcan al objeto.

### **5.1.3 Procesamiento de las nubes de puntos**

El segundo nodo que se desarrolla tiene la función de procesar las nubes de puntos que llega con el mensaje de los bounding box detectados, para recortar los objetos de la escena.

---

Al igual que el primer nodo, en este también se ha diseñado una clase que contiene los callbacks de los suscriptores en la parte pública para utilizarlos en el main y las variables en la parte privada para utilizarlas entre callbacks, así que en el main se ha creado un objeto de la clase para poder llamar a las funciones callback en los suscriptores. Cuando el suscriptor reciba el mensaje, este lo enviará a la función callback correspondiente, el mensaje que recibe este nodo como se ha explicado anteriormente se compone de una nube de puntos y del mensaje de los bounding boxes de los objetos detectados en la imagen a la que está asociado la nube.

El primer paso en la función es convertir el mensaje de la nube de puntos que es de tipo “**sensor\_msgs::PointCloud2**” a “**pcl::PointCloud<pcl::PointXYZRGB>::Ptr**”, de esta manera se podrá trabajar para poder recortar las nubes de puntos de los objetos detectados y eliminar el background que no pertenezca al objeto.

Luego, se hará un bucle que vaya de 0 al tamaño del vector de los bounding boxes para poder tratar cada uno de los objetos detectados y recortarlo de la nube de puntos original. En este bucle primero se obtendrán las variables de ancho y alto del bounding box y se comprobará que no supera ninguno de los límites de tamaño de la nube, ya que si fuera el caso se estaría entrando en puntos que no existen y saltaría un error durante el tiempo de ejecución, para poder corregir esos problemas se establecen que esos puntos sean 0 o el tamaño máximo del ancho o alto dependiendo de la situación.

Con las variables importantes ya definidas, es el momento de recortar la nube del objeto de la original, esto se conseguirá creando una nube de puntos del tamaño del bounding box y accediendo a cada punto de la nube original para ir copiando todas sus intensidades en la nube de puntos del objeto. Cuando se tenga la nube del objeto copiada y recortada, se hará un paso intermedio, el cual es eliminar los puntos NaN (Not a Number), ya que en los siguientes pasos esos puntos pueden dar problemas durante la ejecución del pipeline.

Con la nube ya recortada, se hace un filtrado del tipo de clase del objeto que se ha detectado, dado que hay situaciones en la que se puede captar un objeto no estático o que tenga mucha profundidad, y con el método que se está utilizando para eliminar ese background no es del todo óptimo hacerlo. Así que, dependiendo de la clase del objeto se le eliminará el background o no.

A continuación, se llega al penúltimo paso de este nodo, que es utilizar la eliminación de planos dominantes para obtener la nube de puntos del objeto sin el background, esto se consigue gracias al método “**pcl::SACSegmentation<pcl::PointXYZRGB>**” que hace la segmentación del objeto y se obtienen los inliers y coeficientes del plano. Por último, siempre que el tamaño de los inliers sea mayor que 0, se le aplica el filtro gracias a la clase “**pcl::ExtractIndices<pcl::PointXYZRGB>**”, obteniendo la nube del objeto sin los puntos que no pertenecen al objeto detectado. Este proceso se repite con cada uno de los objetos que sean detectados en la imagen y si en algún caso falla el método, la nube se quedará sin haber eliminado los puntos.

Al final, cada nube de puntos de los objetos se reconvierte otra vez a “**sensor\_msgs::PointCloud2**” para poder volverla a enviar al siguiente nodo, pero antes se crea un mensaje personalizado que guarda los parámetros importantes de cada una de las nubes como el tipo del objeto o el tamaño del bounding box, además, de la propia nube de puntos recortada. Al mismo tiempo, este mensaje se guarda en un nuevo mensaje personalizado que contendrá un vector con los mensajes de las nubes de puntos de los objetos recortados y la escena completa,

---

todo esto en variables de mensajes de ROS.

#### **5.1.4 Incorporación del ORB-SLAM2**

Acto seguido se describe de cómo se introducen los mensajes que se obtienen de ORB-SLAM2, para poder usarlos en el pipeline general, pero primero se va a hablar de como trabaja el nodo de ORB-SLAM2.

##### **5.1.4.1 Mapeo y localización con ORB-SLAM2**

Para poder hacer uso de ORB-SLAM2 en ROS será necesario descargar el paquete de ROS, que hay en GitHub que se menciona en el anterior capítulo. Tras esto, se deberá añadir al directorio de ROS con el que se está trabajando y compilarlo mediante **catkin\_make**. Cuando se haya compilado se deberá iniciar **roscore**, que es el servicio maestro que proporciona ROS y es necesario para poder ejecutar cualquier programa que haya sido hecho en ROS.

En este trabajo se hacen uso de archivos bag de la Universidad de Munich del departamento de Computer Vision Group TUM Department y de los desarrolladores de ORB-SLAM2 como datasets. Estos archivos bag proporcionan a sus usuario diferentes topics de imágenes RGB y de profundidad, además de la información de los parámetros intrínsecos de la cámara usada y nubes de puntos de la escena. Para poder hacer uso de este archivo bag se deberá usar el siguiente comando: **rosbag play nombre\_archivo.bag**. Este comando ejecutará los datos guardados en tiempo real para poder leerlos, para ello se harán uso de suscriptores a los topics en cada uno de los nodos donde se requiera la información.

El paquete de ORB-SLAM2 proporciona un archivo launch para poder tratar estos dataset, pero antes de lanzar el launch se debe lanzar el comando **source /devel/setup.bash** para poder indicar que en el terminal se pueden lanzar archivos creados en el catkin workspace que ha creado el usuario. El archivo launch que se deberá lanzar se encuentra en la carpeta “/ros/-launch” y ahí se encuentra el siguiente archivo: **orb\_slam2\_tum2\_rgbd.launch**. Cuando se ejecute haciendo uso del comando **roslaunch orb\_slam2 orb\_slam2\_tum2\_rgbd.launch**, el nodo esperará hasta que hayan ejecutado el archivo bag para poder cargar tanto la información de los parámetros intrínsecos como se ve en la Figura 5.5 y leer de los topics de las imágenes.

El contenido del archivo “.launch” especifica qué topics hay que leer para poder hacer efectivo el proceso de SLAM. En este caso, se leerán los topics “**/camera/depth/image**” y “**/camera/rgb/image\_color**”, también se especifican los parámetros estáticos y dinámicos y la ruta de algunos archivos, como el archivo de ajustes o “**settings\_file**” que contiene los parámetros intrínsecos de la cámara, los parámetros del ORB y de visualización. Este archivo se usaría en el caso que no hubiera el topic “**/camera/rgb/camera\_info**” dentro del archivo bag, en cambio si existiera ese topic los parámetros se cogería la información de él y otros parámetros ya estarían predefinidos en el programa como parámetros del detector de características ORB.

Si se quisiera ver el proceso de SLAM del entorno, se puede hacer uso de la aplicación RViz, ya que este paquete de ROS provee un archivo de configuración del entorno de la aplicación, este se encuentra en la carpeta “/ros/config”. Esta configuración dota de los siguientes displays:

```

ORB-SLAM2 Copyright (C) 2014-2016 Raul Mur-Artal, University of Zaragoza.
This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions. See LICENSE.txt.

OpenCV version : 3.3.1-dev
Major version : 3
Minor version : 3
Subminor version : 1
Input sensor was set to: RGB-D

Loading ORB Vocabulary.
Vocabulary loaded!

Camera Parameters:
- fx: 525
- fy: 525
- cx: 319.5
- cy: 239.5
- k1: 0
- k2: 0
- p1: 0
- p2: 0
- fps: 30
- bf: 0
- color order: RGB (ignored if grayscale)

ORB Extractor Parameters:
- Number of Features: 1200
- Scale Levels: 8
- Scale Factor: 1.2
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7

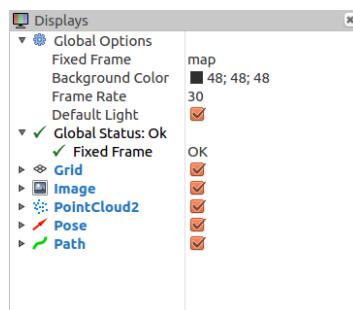
Depth Threshold (Close/Far Points): 0
Enable localization only: false

```

**Figura 5.5:** Terminal cargando los parámetros de la cámara para empezar ORB-SLAM.

- Una imagen en la cual se muestra las imágenes con los keypoints que se han detectado en ella.
- Una nube de puntos que contiene los keypoints encontrados en la imagen y los cuales se van visualizando en RViz.
- Un “Pose” que muestra la pose de la cámara.
- Un “Path” que muestra la trayectoria de la cámara.

Los displays se pueden observar en la Figura 5.6.



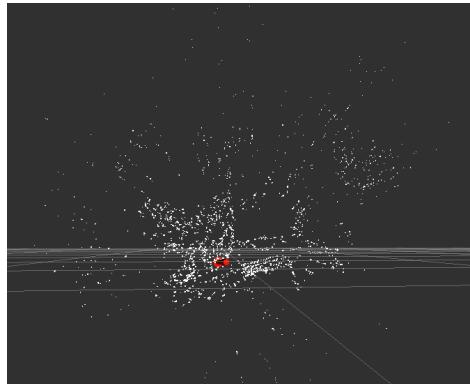
**Figura 5.6:** Dispaly de RViz para ORB-SLAM2.

Para poder ejecutar RViz se usa el comando “**rviz**” en la terminal y luego se carga el archivo de configuración de Rviz que se encuentre en “/orb\_slam\_2\_ros/ros/config” y permite que se utilicen los displays con los topics ya asociados que se han comentado anteriormente en la

aplicación, tras esto ya se puede ejecutar el launch del ORB-SLAM2 y el archivo bag. Los resultados que se obtienen se ven en la Figura 5.7 que muestra los keypoints en la imagen que recibe el nodo y la Figura 5.8 que es la generación de la nube de puntos a partir de los keypoints detectados en la imagen.



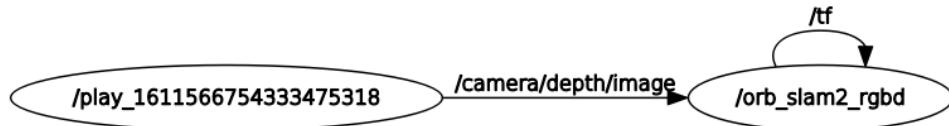
**Figura 5.7:** Keypoints detectados de la imagen.



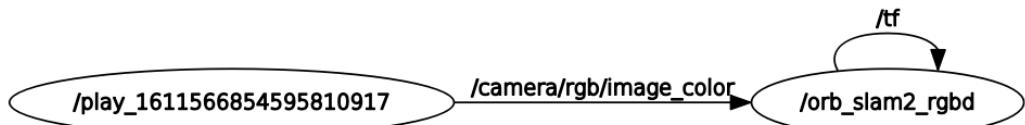
**Figura 5.8:** Nube de puntos de los keypoints.

Para poder ejecutar el programa se debe hacer uso de los comandos “**roslaunch orb\_slam\_2\_ros orb\_slam2 \_tum2\_rgbd.launch**”, para lanzar y ejecutar el nodo de ORB-SLAM2 y “**roscore**” para lanzar el servicio maestro. Esto hará que el programa lea los topics del archivo bag, que se ejecutará en una terminal a parte y que el nodo de ORB-SLAM2 vaya generando los resultados que se necesiten. En este trabajo se necesitarán los datos del topic que publique los posición actual de la cámara, que se publica en el topic “**/orb\_slam\_2\_ros/publish\_pose**”, dicho mensaje es el vector traslación y el quaternion de los ángulos para poder construir una matriz de transformación que se le aplicarán a las nubes de puntos para poder hacer la reconstrucción de 3D del entorno donde se este trabajando.

Para ver los nodos que trabajan se puede usar el comando “**rqt\_graph**”, este comando muestra un gráfico de los nodos que están en funcionamiento en ese momento y con que topics se relacionan.



**Figura 5.9:** Entrada de mensajes del topic de imágenes de profundidad.



**Figura 5.10:** Entrada de mensajes del topic de imágenes de RGB.

Como se ve en las Figuras 5.9 y 5.10, el nodo de “/orb\_slam2\_rgbd” toma información de los topics “/camera/depth/image” y “/camera/rgb/image\_color” del nodo que ejecuta el archivo bag, también toma información del topic “/camera/rgb/camera\_info” para los parámetros de la cámara.

#### 5.1.4.2 ORB-SLAM2 en el pipeline general

En el pipeline general lo que se busca es el mensaje que se obtiene del topic “/orb\_slam2\_ros /publish\_pose” y es de tipo “geometry\_msgs::PoseStamped”, el cual contiene los valores de traslación y rotación para poder crear una matriz de transformación y esta se pueda aplicar a la nube de puntos correspondiente para que tenga la posición y orientación del mundo real.

La matriz de transformación es una matriz cuatro por cuatro. Está formada por una matriz tres por tres o matriz de rotación, que hace referencia a las tres rotaciones que hay en los tres ejes. En la Ecuación 5.4 se muestra cómo es la combinación de las tres matrices de transformación en los ejes Z (Ecuación 5.1), Y (Ecuación 5.2) y X (Ecuación 5.3)<sup>1</sup>, qu, por último, se combina con una matriz de traslación para tres dimensiones (X, Y, Z) tal y como

---

<sup>1</sup>C es coseno y S es seno en las matrices de rotación.

se ve en la Ecuación 5.5.

$$\begin{bmatrix} C\theta & -S\theta & 0 & 0 \\ S\theta & C\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.1)$$

$$\begin{bmatrix} C\phi & 0 & S\phi & 0 \\ 0 & 1 & 0 & 0 \\ -S\phi & 0 & C\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.2)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha & -S\alpha & 0 \\ 0 & S\alpha & C\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.3)$$

$$\begin{bmatrix} C\phi \cdot C\theta & -C\alpha \cdot S\theta + S\phi \cdot C\theta \cdot S\alpha & S\alpha \cdot S\theta + S\phi \cdot C\theta \cdot C\alpha & 0 \\ C\phi \cdot S\theta & C\alpha \cdot C\theta + S\phi \cdot S\theta \cdot S\alpha & -S\alpha \cdot C\theta + S\phi \cdot S\theta \cdot C\alpha & 0 \\ -S\phi & C\phi \cdot S\alpha & C\alpha \cdot C\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.4)$$

$$\begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.5)$$

Finalmente, en el nodo de ROS que reciba los mensajes de ORB-SLAM2, el mensaje de “**geometry\_msgs:: PoseStamped**” será recibido por el suscriptor y se convertirá a “**tf:Transform**” con funciones que prestan diferentes librerías, obteniendo una matriz como la que se muestra en la Ecuación 5.6, siendo el resultado de la combinación de las matrices de las Ecuaciones 5.4 y 5.5.

$$T = \begin{bmatrix} Rotacion(3x3) & Traslacion(3x1) \\ 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} C\phi \cdot C\theta & -C\alpha \cdot S\theta + S\phi \cdot C\theta \cdot S\alpha & S\alpha \cdot S\theta + S\phi \cdot C\theta \cdot C\alpha & tx \\ C\phi \cdot S\theta & C\alpha \cdot C\theta + S\phi \cdot S\theta \cdot S\alpha & -S\alpha \cdot C\theta + S\phi \cdot S\theta \cdot C\alpha & ty \\ -S\phi & C\phi \cdot S\alpha & C\alpha \cdot C\phi & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.6)$$

### 5.1.5 Combinación de los mensajes de ORB-SLAM2 y las nubes de puntos recortadas

A continuación, se va a explicar la manera de combinar los mensajes de las nubes de puntos recortadas con los mensajes de ORB-SLAM2 para poder aplicar a las nubes las matrices de transformación y recrear la escena del archivo bag.

Para ello, se hace uso del último nodo de ROS que se desarrolla en este proyecto, con su propia clase como los anteriores nodos. Con las funciones callbacks en la parte pública de la clase para poderlas usar en la main del programa. En este caso, se hará uso de la política de sincronización Approximate Time Policy con tal de obtener ambos mensajes en un mismo callback y los cuales tengan el mismo time-stamp, dado que los mensajes vienen de diferentes nodos. Cuando un par de mensajes entran a la función, se decodificarán para poder usarlos, primero el mensaje de ORB-SLAM2 se convertirá a una variable “**tf:: Stamped <tf::Transform>**” y luego se convertirá a “**Eigen::Matrix4f**”. De esta manera, ya se tiene la matriz de transformación para poder aplicarla a las nubes de puntos.

Seguidamente, los mensajes que contengan las nubes de puntos con los objetos recortados se transformarán en “**pcl::PointCloud <pcl::PointXYZRGB> ::Ptr**” y se van guardando en un vector de nubes de puntos, al mismo tiempo, se les aplicará la matriz de transformación de ORB-SLAM2. Cuando se haya acabado de convertir todas las nubes de los objetos, estas se juntarán en una misma nube de puntos mediante el operador “+”. Por último, se debe convertir el mensaje con la escena final a una variable de Point Cloud Library y luego aplicar la matriz de transformación.

Tras estos pasos, ya se tienen las nubes de puntos del mensaje con la matriz de transformación de ORB-SLAM2 aplicada, pero antes se le debe aplicar el algoritmo de VoxelGrid, con tal de reducir el número de puntos de la nube y que los siguientes pasos no tengan mucha carga de trabajo durante el tiempo de ejecución. El filtro VoxelGrid se hace introduciendo primero la nube de puntos a la cual se quiere aplicar el filtro y después se crea una hoja con cierto tamaño para poder reducir el número de puntos de la nube.

El siguiente paso es comprobar si es la primera vez que se entra en la función callback. Si es cierto, se guardarán la nube de puntos de los objetos en un nuevo vector y la nube de la escena se guardará en un mensaje de ROS, reconvirtiéndolo a PointCloud2 para poder reutilizarlo la siguiente vez que se entre a la función callback. Además, también se mantendrá la matriz de ORB-SLAM2.

En cambio, si no es la primera vez que se entra en la función callback, primero se carga la escena anterior en una variable “**pcl::PointCloud <pcl::PointXYZRGB> ::Ptr**” y se comprueba si se tiene que hacer el registro de pares de nubes.

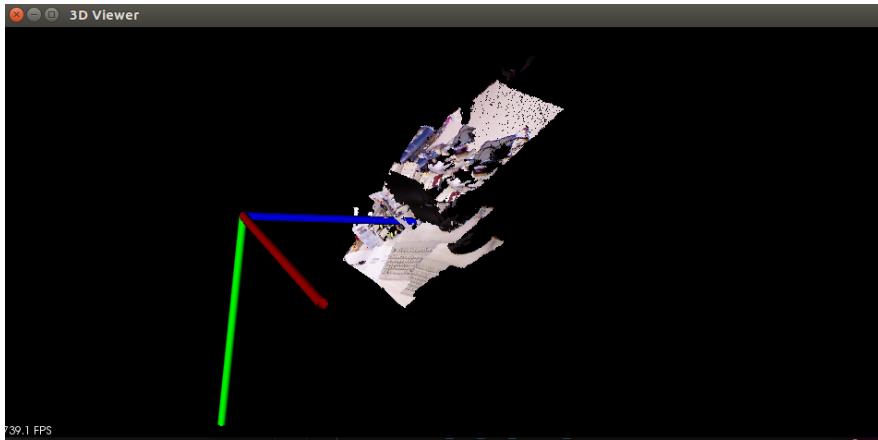
Para decidir cuándo hacer el registro de pares de nubes, se comprobará la diferencias que hay de la matriz de ORB-SLAM2 actual con la anterior. Por lo tanto, se comprueba la última columna de la matriz, que es donde se encuentra el vector de traslación, y si los resultados son mayores a un umbral propuesto se hará el registro de nubes de puntos, sino el proceso continuará y se guardará la escena con los objetos recortados en el vector donde se están guardando ese tipo de datos.

A continuación, se explicarán el registro de pares de nubes y la alineación de nubes con las matrices obtenidas. Estos pasos se harán siempre que la distancia obtenida del vector de traslación de la diferencia que hay entre la matriz de transformación actual y la anterior no supere el umbral propuesto. Finalmente, se hagan o no esos pasos, las variables de la matriz de transformación y la nube de puntos de la escena anteriores se actualizarán con los datos “actuales”, en cambio, si durante el registro de pares nubes hay un problema no se actualizan las variables.

---

### 5.1.5.1 Registro de pares de nubes de puntos

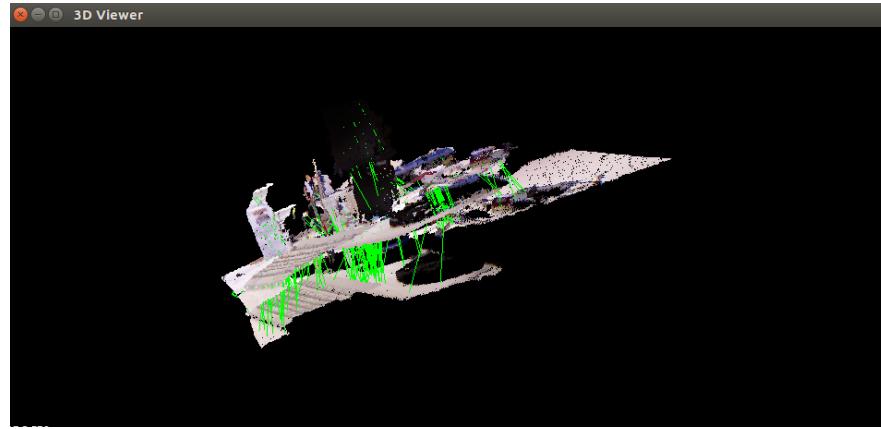
En este apartado se describe el registro de pares de nubes, en el caso que sea necesario hacerlo. El objetivo es conseguir adaptar mejor cada par de escenas, por eso se hace uso de este pipeline del registro de pares de nubes. Se usará como ejemplo la Figura 5.11.



**Figura 5.11:** Antes del registro completo entre dos nubes de puntos.

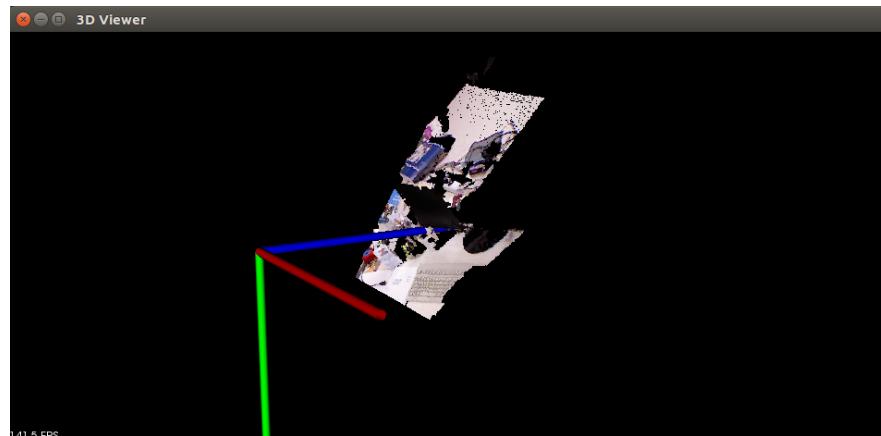
Para ello se debe seguir los siguientes pasos:

1. Crear las variables que se van a usar, al menos se deben crear dos “`pcl:: PointCloud <pcl::Normal> ::Ptr`” para obtener las normales de cada nube de puntos, dos variables de nubes de puntos para obtener los keypoints de cada nube de puntos y dos “`pcl:: PointCloud <Extractor-Keypoints> ::Ptr`” para guardar los descriptores de las nubes de puntos.
2. Tras esto, primero se calculan las normales con el método “`pcl:: NormalEstimationOMP <pcl::PointXYZRGB, pcl::Normal>`” de cada una de las nubes. Este método tendrá como input la nube de puntos y dos parámetros, el radio de búsqueda y el número de vecinos más cercanos.
3. El siguiente paso es calcular los keypoints de las nubes de puntos, para ello se utilizará un extractor de características proporcionado por la PCL.
4. Teniendo ya las normales y los keypoints, el siguiente paso es obtener los descriptores, para ello se utilizará un método proporcionado por la PCL.
5. El cuarto paso es encontrar las correspondencias entre ambas nubes, para ello la Point Cloud Library da el método “`pcl:: registration:: CorrespondenceEstimation <Extractor-Keypoints, Extractor-Keypoints>`”. Cuando se haya hecho este paso se tendrán que rechazar dichas correspondencias que no sean “buenas” con el método “`pcl::registration ::CorrespondenceRejectorSampleConsensus < pcl ::PointXYZRGB >`” y quedarán como los emparejamientos buenos como en la Figura 5.12.



**Figura 5.12:** Emparejamientos entre dos nubes de puntos.

6. Tras haber rechazado las “malas o incorrectas” correspondencias, se obtiene la matriz de transformación que se ha generado a partir de rechazar las correspondencias, para introducirla en el método ICP como transformación inicial.
7. Al final, como ya se tiene una transformación inicial, ya se puede usar ICP para acabar de adaptar la segunda nube a la primera. Cuando se haya acabado de realizar ICP, se habrá obtenido una nube de puntos ya alineada con la nube origen, pero antes se comprueba que se hayan alineado correctamente, y sí se da el caso y es verdadero la matriz obtenida por ICP se le aplicará a la nube de puntos con los objetos recortados correspondientes, sí es falso no se aplica la matriz. El resultado de la alineación entre nubes es el que se ve en la Figura 5.13.



**Figura 5.13:** Registro completo entre dos nubes de puntos.

El registro de pares de nubes se irá repitiendo con cada par de escenas, siempre que se hayan cumplido las condiciones entre las matrices de ORB-SLAM2. La matriz que se obtenga del ICP se irá aplicando a las nubes de los objetos recortados. En cambio, si el valor FitnessScore es superior al umbral que se propone la nube es descartada, tampoco se le aplica la matriz

de transformación y ni la nube de objetos se añade al vector de nubes.

### 5.1.5.2 Aplicación de las matrices del registro de pares de nubes

Cuando finalice el registro de un par de nubes de puntos y ambas nubes están alineadas y hayan convergido dando un resultado correcto y el valor de FitnessScore está por debajo del umbral propuesto, se le debe primero aplicar la matriz obtenida por ICP, que contiene la combinación de las matrices de ICP y RANSAC, a la nube de puntos con solo los objetos y luego se deberá adaptar la nube a la primera escena. La nube de puntos con la escena completa ya se le ha aplicado la matriz de ICP y no hará falta alinearla con las anteriores, ya que solo se quiere que la escena final este compuesta por las nubes de puntos con los objetos recortados.

El procedimiento a seguir es el siguiente, cuando acabe el primer registro de pares de nubes se deberá haber guardado la combinación de las matrices de RANSAC e ICP en una variable “**Eigen::Matrix4f**”, siendo la transformación entre la escena “n” y “n+1” para poder alinearlas y se le conocerá como  $T_1^2 = T^{Ransac} * T^{ICP}$ . Siempre la matriz más reciente irá a la derecha, en este caso la de ICP.

En el siguiente registro de pares de nubes entre la escena “n+1” y “n+2”, que generará una matriz  $T_2^3$ , que se le aplicará a la escena “n+2”, al resultado de esta transformación se le debe aplicar la matriz  $T_1^2$ , para poder alinear la escena “n+2” con la escena “n”.

Por último, para simplificar las operaciones las matrices  $T_1^2$  y  $T_2^3$  se combinarán en un producto para formar la matriz  $T_1^3$  y hacer los procesos de aplicación de matrices mucho más rápido. De manera resumida, la nueva nube de puntos estará dada por la siguiente ecuación

$$N^{1^n} = T_1^n * T_n^{(n+1)} * N^{(n+1)} \quad (5.7)$$

con el objetivo de alinearla con la primera escena, en esta ecuación la matriz más reciente es  $T_n^{(n+1)}$  así que es la que va más a la derecha. La manera de operar la Ecuación 5.7 será primero el producto de  $T_n^{(n+1)} * N^{(n+1)}$  y segundo será el producto de la matriz  $T_1^{(n)}$  y el resultado del producto anterior, el resultado de este último producto es la nube transformada  $N^{1^n}$ .

La leyenda utilizada con estas variables es la siguiente, “n” es el índice que se usa para recorrer el vector de nubes de puntos e indicar en qué nube de puntos se está, “ $T_x^y$ ” es la matriz de transformación y “N” es la nube de puntos. Todo este proceso se repetirá por tantas parejas de nubes de puntos consecutivas haya en el vector de escenas.

### 5.1.5.3 Formación de la escena final

Finalmente, para conseguir la escena con todas las subescenas con los objetos recortados se recorrerá el vector de las nubes de puntos con los objetos y mediante el operador suma las nubes se irán agrupando en una nueva variable, para tener en una variable todos los objetos detectados, recortados y alineados, gracias a las matrices de transformación, obteniendo la escena final. Para volver a usar la escena final en próximos programas, se guarda la nube final cada vez que se entra a la función.

## 5.2 Sincronización de topics

En esta sección se explica cómo se han sincronizado los topics de ROS para poder obtener los mensajes desde distintos topics en una misma función callback. Primero, se va explicar el uso de la librería de “message\_filters”, que su función es recolectar mensajes y filtrarlos en un mismo espacio. Un ejemplo de sincronización es que hay varios mensajes de diferentes fuentes, lo que se hace es que en una función callback se recogen los mensajes cuyo time-stamp sea similar o igual. Los métodos usados para la experimentación son **Exact Time Policy** y **Approximate Time Policy**, pero también se va a explicar el método más óptimo para la sincronización de topics y este trabajo que es el uso de mensajes personalizados, además de los tipos de mensajes que se han creado y que uso tienen.

### 5.2.1 Exact Time Policy

La primera política de message filters es la de Exact Time Policy, este método requiere que los mensajes tengan exactamente la misma marca de tiempo para que coincidan. Solo se llama a su devolución de llamada si se ha recibido un mensaje en todos los canales especificados con la misma marca de tiempo exacta. La marca de tiempo se lee en el campo de encabezado de todos los mensajes (que es obligatorio para esta política de sincronización de mensajes).

La causa por la cual este método no se utiliza es porque necesita que sea al mismo tiempo y en algunas situaciones durante la ejecución de la tarea hay mensajes que no tienen el mismo time-stamp, esto se soluciona gracias al uso del **Approximate Time Policy**.

### 5.2.2 Approximate Time Policy

La segunda política de message filters es **Approximate Time Policy** que hace uso de un algoritmo adaptativo para hacer coincidir los mensajes según su marca de tiempo.

Su uso es gracias a que da un margen de tiempo entre los topics para recoger mensajes, esto hace que se recojan bien los mensajes de los topics, en cambio tampoco era lo más óptimo. Finalmente la solución óptima fue crear mensajes personalizados, para que solo se lea de un único topic.

### 5.2.3 Mensajes personalizados para la sincronización de mensajes

Como solución final, se propuso crear los propios mensajes a medida, consiguiendo así un perfecto envío de los mensajes entre diferentes nodos. Para ello se creaba un archivo “.msg” en la carpeta de “darknet\_ros\_msgs” con los tipos de datos que se iban a enviar y luego se añadía al “**CMakeLists.txt**” para luego compilarlo y poder usarlos en los programas.

Los mensajes que se han creado son los que se van a explicar.

- El primer mensaje es uno que contiene dos nubes de puntos y dos imágenes. Esto se utiliza para recoger nubes e imágenes consecutivas, para luego enviarlas al nodo donde se haga la detección de objetos en las imágenes.
  - El segundo contiene una nube de puntos con la bounding box correspondiente para usar los datos de las bounding boxes para recortar los objetos detectados de la nube de puntos.
-

- El tercer mensaje contiene un vector con todas las nubes de puntos de los objetos recortados, además de la información necesaria como el tipo de clase que es cada objeto y las coordenadas de donde han sido recortados, luego también se añade la escena sin recortar y el time-stamp de cuando se tomo el frame.

Todos los tipos de datos mencionados anteriormente son tipos de mensajes para poder enviarlos mediante la publicación de mensajes vía topics de ROS. Con la creación de los mensajes a medida para el programa, el desarrollo del código se hace mucho más simple para el momento de enviar mensajes por topics, ya que solo será necesario crear un publisher para cada tipo de mensaje personalizado. En cambio, para la primera sincronización entre la imagen y la nube de puntos y la sincronización entre los mensajes de ORB-SLAM2 y las nubes recortadas, si que se utilizará la política de Approximate Time Policy, ya que en esos casos los mensajes personalizados no sirven.

# 6 Experimentación y resultados

En este capítulo se van a explicar las diferentes experimentaciones que se han llevado a cabo para resolver los problemas que se han planteado durante la realización de este proyecto, además, de los resultados finales obtenidos.

## 6.1 Experimentación con ORB-SLAM2 con diferentes datasets

A continuación, se explicará las primeras experimentaciones del proyecto. Estas se basan en diferentes pruebas realizadas con ORB-SLAM2 y con diferentes datasets, para comprobar los resultados que se obtienen.

Las pruebas que se han hecho con el paquete de ROS de ORB-SLAM2 para poder hacer SLAM, se han probado con diferentes datasets obtenidos de la Computer Vision Group TUM Department of Informatics Technical University of Munich y de los desarrolladores de ORB-SLAM2. El fin de estas pruebas es encontrar cuáles son las mejores situaciones para trabajar y encontrar los mejores resultados posibles.

Durante las pruebas realizadas se encontraron tres casos significativos a la hora de realizar SLAM de una escena. En el primer caso el archivo bag utilizado contenía una secuencia de imágenes de una habitación, pero dicha secuencia no era del todo fluida y había cortes entre diferentes frames. Eso se ha comprobado mirando los números de secuencia que había en el topic `/camera/rgb/camera_info`, dado que en su “header” contiene un apartado de número de secuencia y por terminal se puede ver como va saltando x número de secuencia entre frames. Esto hace que en el momento de aplicar el algoritmo de SLAM hubiera momentos que perdía la trayectoria de la cámara y se volvía a empezar el algoritmo. Otra manera que se puede comprobar que el SLAM se “reinicia” es obteniendo la matriz de transformación del topic, ya que si se imprime por terminal cada cierto tiempo aparece la matriz identidad. En este caso sería una cuatro por cuatro, porque la matriz de transformación está compuesta de matriz de rotación tres por tres y un vector traslación tres por uno.

El segundo caso encontrado era que a mitad de la secuencia se dejaba de encontrar keypoints en los frames y el SLAM dejaba de funcionar, siendo imposible la obtención de la pose de la cámara.

Finalmente, en el mejor de los casos es cuando el archivo bag tenía una secuencia completa y sin cortes de la escena. Esto significa que no dejaba ningún momento de detectar los keypoints en el frame y podía construir la nube de puntos de los keypoints detectados, pero no todos los datasets contenían tanto imágenes como nubes de puntos.

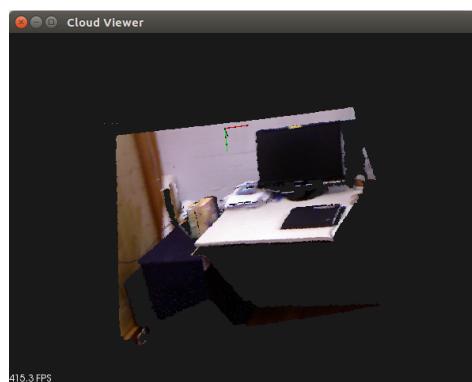
Como resumen de esta experimentación, hay que tener cuidado en la elección de un dataset para poder trabajar con él, ya que puede haber situaciones en la cual se elija uno que no funcione en el paquete de ORB-SLAM2 o que no hayan objetos que sean reconocibles por el paquete de Darknet ROS.

## 6.2 Eliminación de background en las nubes de puntos de objetos

En esta sección se hará una explicación sobre qué método es mejor para la eliminación de los puntos que no pertenecen al objeto en la nube de puntos recortada de la escena, a los cuales se harán referencia como background. Los métodos que se han utilizado son los siguientes:

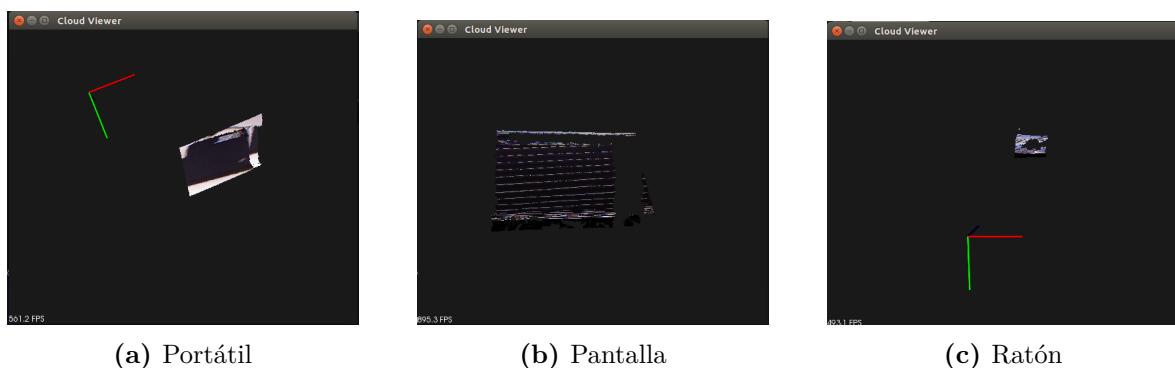
- Region growing segmentation
- Conditional Euclidean Clustering
- Color-based region growing segmentation
- Eliminar planos dominantes

La escena la cual se ha usado como ejemplo es la Figura 6.1.



**Figura 6.1:** Escena de prueba.

Los objetos recortados de la escena son los que se muestran en las Subfiguras 6.2a, 6.2b y 6.2c.



**Figura 6.2:** Objetos recortados de la escena para hacer las pruebas de eliminación de background.

## 6.2.1 Region growing segmentation

El algoritmo de crecimiento de regiones está implementado en la clase **pcl::RegionGrowing** < **pcl::PointXYZRGB, pcl::Normal**> (Library, s.f.-d), su marco teórico está explicado en la Sección 2.4.1. El propósito de dicho algoritmo es juntar puntos que están lo suficientemente cerca en términos de suavidad. Por lo tanto, la salida de dicho algoritmo es el conjunto de clusters que se consideran de la misma superficie. También trabaja con las comparaciones de las normales de cada punto.

### 6.2.1.1 Resultados

Para poder eliminar el background lo que se hace es coger el cluster con más puntos y recortarlo de la nube a la cual se le aplicado el **pcl::RegionGrowing**, así se quedaría la nube con el cluster con más puntos.

Los parámetros del algoritmo de **pcl::RegionGrowing** que se han modificado para obtener los mejores resultados son los siguientes:

- **reg.setMinClusterSize (Min\_Size\_Cluster)**, en esta función se indica el número mínimo de puntos que caben en el clúster.
- **reg.setMaxClusterSize (Max\_Size\_Cluster)**, en esta función se indica el número máximo de puntos que caben en el clúster.
- **reg.setNumberOfNeighbours (Number\_Neighbours)**
- **reg.setSmoothnessThreshold ( Smoothness / 180.0 \* M\_PI)**, este método pone el ángulo en radianes que será el umbral del ángulo de las normales.
- **reg.setCurvatureThreshold ( Curvature)**, este método pone el umbral de curvatura, dado que si entre dos puntos dicha curvatura es superior no se le añadirá al cluster, pero si es menor si que se le añadirá.

Primero, se ha probado a usar los resultados con los parámetros que vienen por defecto, pero el resultado cuando se ejecutaba es “**Segmentation Fault**”. Por lo tanto, se modificaron los parámetros, empezando primero por el valor de la función “**normal\_estimator.setKSearch (Normal\_Number\_Neighbours)**”. Primero, se ha ido aumentando el valor, pero seguía dando el mismo error y luego se hizo el camino contrario, reducir el valor, hasta que se llegó a el valor de 15.

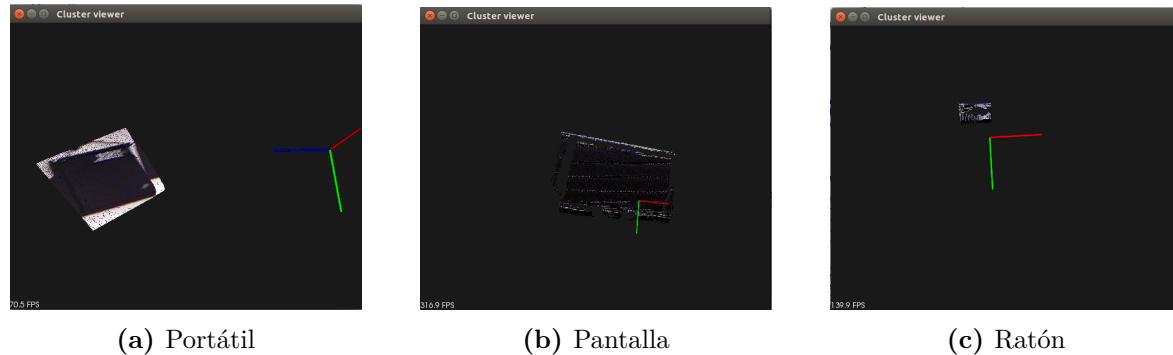
Los siguientes parámetros a modificar fueron los valores mínimos y máximos del tamaño del cluster, “**reg.setMinClusterSize (Min\_Size\_Cluster)**” y “**reg.setMaxClusterSize (Max\_Size\_Cluster)**”, los resultados obtenidos con los valores por defecto que son “**Min\_Size\_Cluster**” igual a 50 y “**Max\_Size\_Cluster**” igual a 1.000.000 son muy similares a las nubes de puntos originales de los objetos. Por lo tanto, se modificó el valor mínimo a 0 y el máximo se hice que fuera un porcentaje del producto de la anchura y altura de la nube de puntos.

Los últimos valores que se modifican son los umbrales de curvatura y y ángulo del algoritmo, primero se prueba con los valores por defecto, pero los resultados son que los clusters con más puntos no llegan a 500 y las nubes obtenidas son puntos agrupados pero el objeto no

se diferencia. Los mejores valores conseguidos con los cambios anteriormente realizados son para el umbral de la curvatura es igual a 15.0 y para el umbral del ángulo es igual a 30.0°. En resumen, los mejores valores para cada uno de los parámetros se recogen en la Tabla 6.1.

	Valores
<b>Normal Number Neighbours</b>	15
<b>Min Size Cluster</b>	0
<b>Max Size Cluster</b>	Producto de la altura y anchura de la nube de puntos
<b>Number Neighbours</b>	15
<b>Smoothness</b>	15,0
<b>Curvature</b>	30,0°

**Tabla 6.1:** Mejores parámetros del algoritmo Region Growing Segmentation.



**Figura 6.3:** Objetos recortados de la escena, tras haberle aplicado el algoritmo de Region Growing Segmentation para eliminar el background.

Como se puede ver en las Figuras 6.3a, 6.3b y 6.3c, los resultados obtenidos son muy similares a las nubes originales y es porque los resultados obtenidos anteriormente los clusters con más puntos no eran los suficientes para poder representar al objeto.

## 6.2.2 Conditional Euclidean Clustering

El algoritmo que se va explicar es el que está incluido en la clase de **pcl::Conditional Euclidean Clustering** (Library, s.f.-b), el marco teórico está explicado en la Sección 2.4.3, y se trata de un algoritmo de segmentación de clusters basado en la distancia euclídea y de las condiciones definidas por el usuario.

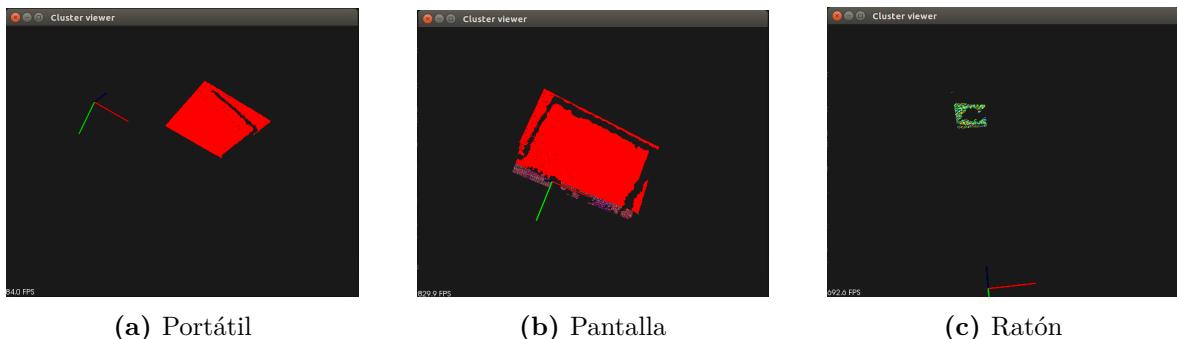
### 6.2.2.1 Resultados

Para obtener buenos resultados se ha tenido que cambiar primero el parámetro de **setLeafSize** del **pcl::Voxel Grid**, ya que eliminaba la mayoría de los puntos de la nube y hacía imposible poder trabajar con este algoritmo, las demás líneas de código se han dejado tal cual y se han modificado algunos parámetros, pero finalmente los mejores valores son los que se ven en la Tabla 6.2.

	Valores
<b>Cluster Tolerance</b>	300
<b>Min Size Cluster</b>	Tamaño de la nube de puntos entre 1000
<b>Max Size Cluster</b>	Tamaño de la nube de puntos entre 5

**Tabla 6.2:** Mejores parámetros del algoritmo Conditional Euclidean Clustering.

Más tarde, surgieron un par de problemas el primero fue que las nubes de puntos que se estaban usando de prueba no tenían el campo de intensidad, el cual era necesario para el desarrollo de este algoritmo, pero no daba más problemas que unos simples “warnings”. En cambio, el siguiente problema que surgió fue que los tiempos de ejecución con nubes con gran cantidad de puntos son muy altos, cosa que haría que fuese ineficiente si se añadiera en los nodos con ROS.



**Figura 6.4:** Clusters de los objetos recortados de la escena usando Conditional Euclidean Clustering.

Como se puede ver, los clusters pueden ser muchos como en la Figura 6.4c y que si se escogiera el más grande no se tendría la figura del objeto en la nube de puntos o que sea un único cluster como en la Figura 6.4b y que ahí si se escoge el cluster con más puntos también se escogería el background. Siendo algo ineficiente para este proyecto.

### 6.2.3 Color-based region growing segmentation

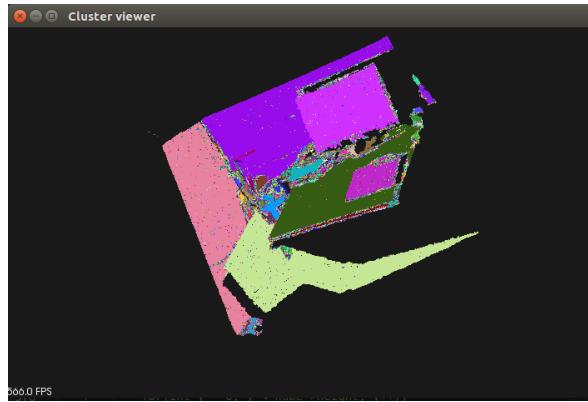
A continuación, se va a explicar cómo se usa el algoritmo de Region Growing Segmentation basado en colores, el cual esta implementado en la clase **pcl::RegionGrowingRGB**, (Library, s.f.-a) y su marco teórico está desarrollado en la Sección 2.4.2.

#### 6.2.3.1 Resultados

En este algoritmo se sigue el mismo método que en el algoritmo de Region Growing Segmentation. En cambio, aquí se tendrá en cuenta los colores en vez de las normales, pero se seguirá cogiendo el cluster con mayor puntos, que supuestamente será donde se encuentre el objeto sin los puntos que conforman el background.

Primero, se han modificado dos parámetros para evitar que siempre diera el error de “Segmentation Fault”, esos parámetros eran el de la función “**pass.setFilterLimits (0.0, 5.0)**”, que en un principio el límite máximo estaba a 1.0 y la función “**reg.setMinClusterSize**

(0)”, que en un principio estaba en 600, luego se ha probado con la escena que se está haciendo pruebas y ha dado el resultado que se ve en la Figura 6.5.



**Figura 6.5:** Nube de puntos tras haberse aplicado RGB region growing segmentation.

Los parámetros que se han modificado para conseguir los mejores resultados son los siguientes:

- **reg.setDistanceThreshold (10)**, este parámetro hace referencia a que si el punto escogido es vecino no, ya que si está localizado a una distancia menor que la puesta por el usuario se incluye en el cluster y se considera vecino, en cambio si es superior se descarta.
- **reg.setPointColorThreshold (5)**, este umbral es similar al umbral del ángulo que se usaba en el Region Growing Segmentation, pero este evalúa los puntos por el color.
- **reg.setRegionColorThreshold (6)**, este umbral es similar al anterior, pero este toma partida en el momento de unión de los puntos.

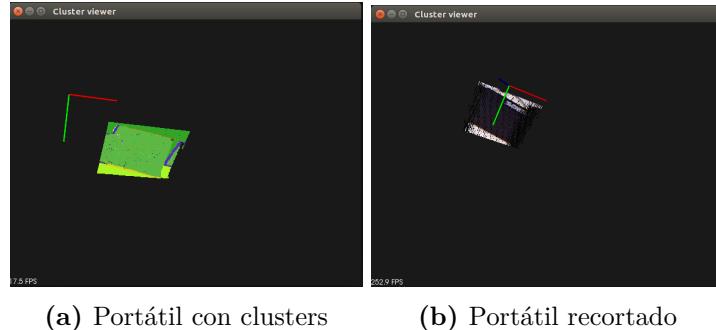
Para el primer parámetro se ha aumentado progresivamente hasta 100 y se ha disminuido hasta 1, dejando los dos otros parámetros sin tocar, lo mismo se ha hecho con los dos siguientes parámetros, se han modificado solamente ellos y los otros dos se han dejado con su valor predeterminado. En resumen, los mejores valores para los parámetros del algoritmo de Color-based Region Growing Segmentation se recogen en la Tabla 6.3.

	Valores
<b>Distance Threshold</b>	10
<b>Point Color Threshold</b>	5
<b>Region Color Threshold</b>	6
<b>Min Size Cluster</b>	0
<b>Max Size Cluster</b>	Producto de la altura y anchura de la nube de puntos

**Tabla 6.3:** Mejores parámetros del algoritmo Color-based Region Growing Segmentation.

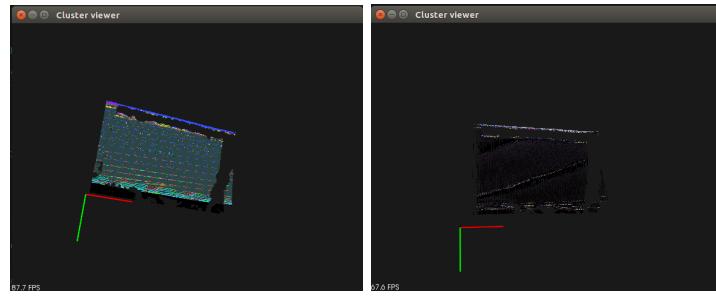
Finalmente, el resultado obtenido de la nube de puntos separada en clusters pintados en diferentes colores que se obtiene de la siguiente función “**reg.getColoredCloud ()**”, pero

para conseguir la nube de puntos que se quiere, se tendrá que coger el cluster más grande y separarlo de la nube de puntos del objeto. Como ejemplos se muestran las Figuras 6.6, 6.7 y 6.8.



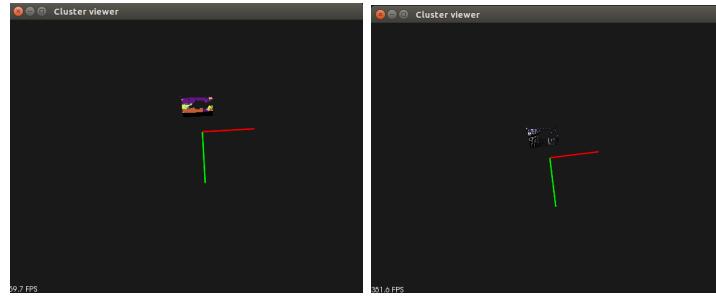
(a) Portátil con clusters      (b) Portátil recortado

**Figura 6.6:** Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del portátil.



(a) Monitor con clusters      (b) Monitor recortado

**Figura 6.7:** Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del monitor.



(a) Ratón de portátil con clus- (b) Ratón de portátil recorta-  
ters                                    do

**Figura 6.8:** Algoritmo Color-Based region growing segmentation aplicado a la nube puntos del ratón de portátil.

Como conclusión a este método utilizado es que el color puede afectar mucho a los objetos como un libro con portada con distintos colores o que tanto el objeto y su background coincidan en una misma tonalidad de color. Por lo tanto, en esos casos este algoritmo no eliminaría el background o eliminaría partes del objeto, así que no sería de gran utilidad.

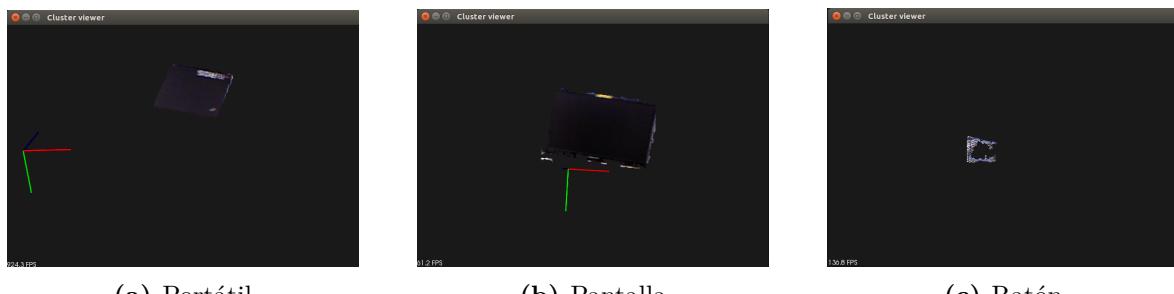
También hay situaciones que parece que los clusters dividen bien el objeto de lo que no es el objeto, pero en el momento de realizar dicha división el resultado no es el esperado, dado que siguen habiendo diversos puntos en el background y que no son pertenecientes al objeto.

#### 6.2.4 Eliminar planos dominantes

Finalmente, se explicará cómo eliminar planos dominantes en la nube de puntos del objeto, (Library, s.f.-c) y su marco teórico es desarrollado en la Sección 2.4.4.

##### 6.2.4.1 Resultados

En este método el único parámetro que se ha modificado ha sido el de función “**seg.setDistanceThreshold (umbral)**”, el mejor valor parámetro para esta función se ha conseguido mediante el método de prueba y error, hasta conseguir que en el valor 0.01 se hayan conseguido los mejores resultados posibles.



**Figura 6.9:** Objetos recortados de la escena recortando los planos dominantes.

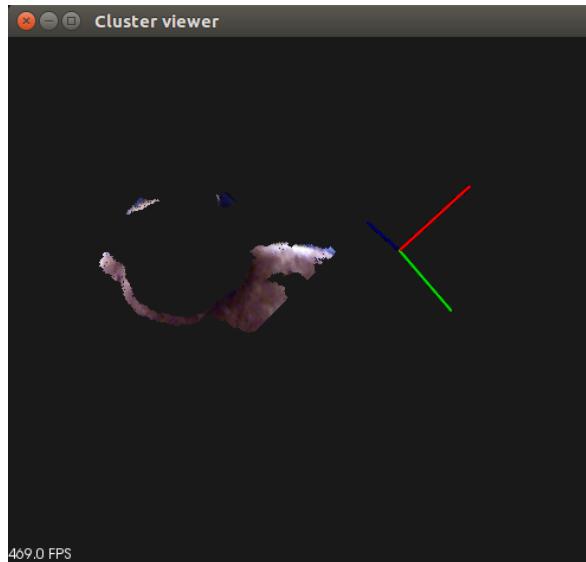
Como se puede ver en la Figura 6.9, el background de los objetos han sido eliminados con perfecta precisión. En cambio, hay situaciones en la que este modelo no podría funcionar, cuando el objeto se encuentra en el mismo plano que una superficie como un libro en una mesa o un cartel en una pared.

Otros problemas que pueden surgir en este método es que en la situación que haya dos o más objetos detectados en el mismo sitio. Por ejemplo, en la Figura 6.10. Por lo tanto, al utilizar este método solo se queda el plano donde se encuentra un objeto detectado y el otro lo recorta, también es difícil acertar con el umbral, ya que puede funcionar muy bien para algunos objetos, pero para otros puede funcionar bastante mal.



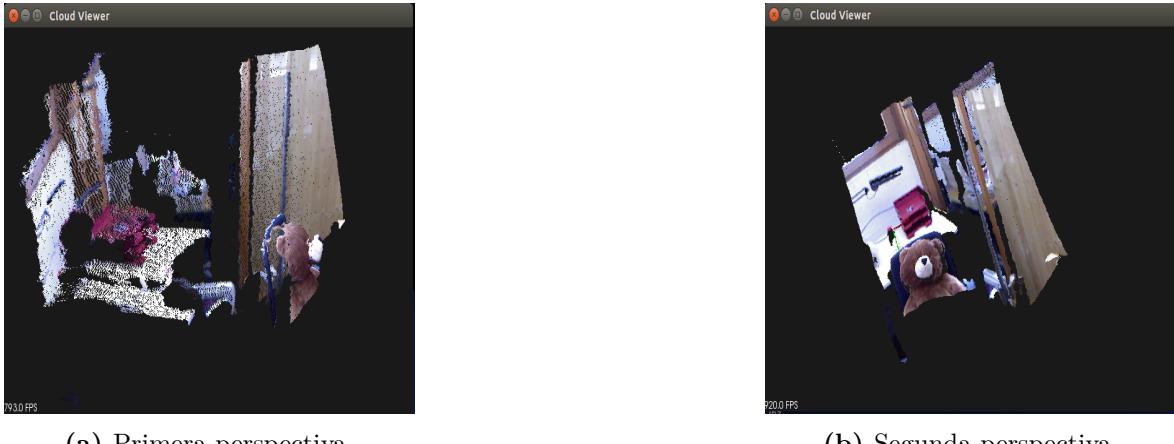
**Figura 6.10:** Ejemplo de varios objetos detectados en la misma zona.

Como resultado de recortar los objetos de la nube de puntos generada por la escena de la Figura 6.10, tanto el oso de peluche como la silla caen en le mismo plano y el resultado es el que se ve en la Figura 6.11.



**Figura 6.11:** Nube de puntos recortada del oso de peluche y la silla.

Como se puede ver en la Figura 6.11, la eliminación del background ha hecho que se recorte también parte del objeto del oso y que desaparezca la silla, pero la causa que como se ha comentado anteriormente es de la elección del umbral para eliminar planos dominantes, también puede afectar la profundidad de la nube de la escena como se ve en la Figuras 6.12a y 6.12b.



(a) Primera perspectiva.

(b) Segunda perspectiva.

**Figura 6.12:** Dos perspectivas de la escena.

Finalmente, el método que se utiliza es el de eliminación de planos dominantes, dado que ha sido el que mejores resultados se han obtenido. Para los problemas que han surgido de que se detecten varios objetos en una misma zona, la solución ha sido ver si habían suficiente puntos cuando se hayan eliminado los planos, porque sino había los suficientes, el proceso se revertía y se dejaba la nube de puntos tal cual estaba, así no se quita información importante de la nube. Luego, se plantea un filtrado de las clases de objetos que no sean estáticos o tengan diferentes profundidades como las persona o algún vehículo, ya que eliminando los planos dominantes puede que alguno contenga puntos del objeto detectado, con el filtrado lo que se hace es que no pasen por la parte de eliminación de los planos y así no eliminar puntos de la nube del objeto.

### 6.3 Sincronización de mensajes de ORB-SLAM2 y las nubes de los objetos

En este apartado se desarrollará cómo se ha conseguido sincronizar esos mensajes, para poder aplicar las matrices de transformación de los mensajes del nodo de ORB-SLAM2 a las nubes de puntos de los objetos detectados en las imágenes que se envían a Darknet y para poder formar una nube de puntos completa de la escena a partir de los objetos detectados en cada frame del vídeo que hay en el archivo rosbag.

El primer planteamiento que se propuso fue guardar los mensajes de ambos nodos en directorios cuyo nombre sería el time-stamp, ya que ambos mensajes contenían ese apartado y se podrá sincronizar en un programa aparte leyendo los directorios. Aunque la desventaja que salía es que se dejaba de trabajar en tiempo real y se hacía en un programa aparte, pero al hacerlo de esta manera se aseguraba que sí que se iban a sincronizar tanto las nubes de puntos como las matrices de transformación.

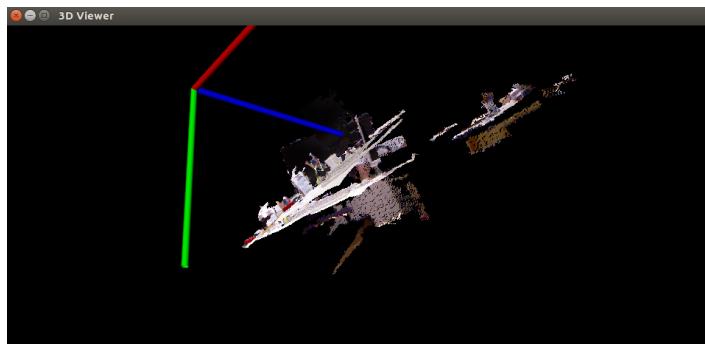
La segunda posibilidad que se propuso para la sincronización era el uso de message filters con tal de sincronizar ambos topics por su time-stamp. En cambio, el resultado no fue el esperado al principio, dado que los nodos que envían los mensajes trabajan a destiempo y su sincronización solo llega a ocurrir una o dos veces por ejecución, así que se descartó esa

posibilidad en primera instancia, pero si se modifica la resolución de las nubes de puntos con el algoritmo de VoxelGrid y la frecuencia con la que el archivo rosbag lanzaba los mensajes de esta manera sí que se llegan a sincronizar varios mensajes y se puede trabajar en tiempo real, cumpliendo uno de los objetivos propuestos para el Trabajo de Fin de Grado.

## 6.4 Uso del registro de pares de nubes

Durante la alineación de escenas se comprueba entre los datos que acaban de llegar y los anteriores si es necesario hacer el registro de pares de nubes con tal de reducir tiempo de ejecución del pipeline. Para ello, se cogen las matrices de ORB-SLAM2 que acaba de llegar y la anterior. Primero, se hace la diferencia entre ellas y luego se comprueba exactamente la última columna del resultado, que es el vector de traslación de la matriz, y si uno de los valores de ese vector es superior al umbral propuesto el registro se hace. En cambio, si no supera el umbral, el programa continua su curso sin hacer el registro de pares de nubes.

El umbral se ha escogido de forma experimental probando con diferentes datasets con tal de escoger el que mejor convenga para las nubes de puntos, ya que no se sabe a ciencia cierta qué distancia es lejos y cerca para las nubes de puntos.



**Figura 6.13:** Ejemplo de separación entre dos escenas.

Por ejemplo, en la Figura 6.13 si que se debería aplicar el registro de pares de nubes para poder alinear ambas escenas.

## 6.5 Elección del mejor pipeline de registro de pares de nubes

En esta sección se describirán los diferentes métodos de extracto de características y computar descriptores que se han usado para obtener el mejor resultado en el registro de pares de nubes.

Los extractores de características utilizados para la experimentación han sido ISS (Intrinsic Shape Signatures) y Uniform Sampling, la diferencia entre ambos extractores es que el método de ISS necesita bastantes parámetros que inicializar, que se muestran en el siguiente listado. Además, es necesario calcular la resolución de la nube de puntos que se vayan a usar, ya que lo demanda un par de parámetros.

- iss.setSearchMethod (tree)

- iss.setSalientRadius (iss\_salient\_radius\_)
- iss.setNonMaxRadius (iss\_non\_max\_radius\_)
- iss.setThreshold21 (iss\_gamma\_21\_)
- iss.setThreshold32 (iss\_gamma\_32\_)
- iss.setMinNeighbors (iss\_min\_neighbors\_)
- iss.setNumberOfThreads (iss\_threads\_)
- iss.setInputCloud (cloud1)
- iss.compute (\*escena\_keypoints)

En cambio, el método de Uniform Sampling no requiere tanto parámetros ni el cálculo de la resolución de la nube, esto lo hace mucho más sencillo y rápido que el método de ISS.

La Tabla 6.4 muestra la comparación de tiempo de CPU de ambos métodos.

Método	Tiempo de CPU (ms)
<b>ISS</b>	2384
<b>Uniform Sampling</b>	3

**Tabla 6.4:** Comparación del tiempo de ejecución(ms) de los extractores de características.

El siguiente paso es obtener los descriptores de las nubes de puntos, para ello se tendrá que elegir un método que proporciona la PCL y se combine bien con el método de extracción de características. Para el método de ISS se eligieron los métodos de PFH y FPFH, en cambio para el método de Uniform Sampling se eligieron los métodos SHOT y PFH.

Para la elección del método para el cómputo de los descriptores se ha tenido en cuenta los tiempos de CPU y los resultados, así que para ISS se combina con PFH y para Uniform Sampling se combina con SHOT. En la Tabla 6.5 se muestra el tiempo de CPU que combina ambos métodos.

Método	Tiempo de CPU (ms)
<b>ISS + PFH</b>	2384 + 25
<b>ISS + FPFH</b>	2384 + 37
<b>Uniform Sampling + PFH</b>	4 + 27
<b>Uniform Sampling +SHOT</b>	3 + 237

**Tabla 6.5:** Comparación del tiempo de ejecución(ms) de los extractores de características más el computo de descriptores.

Como conclusiones de esta experimentación se ha elegido el método de ISS más PFH sobre Uniform Sampling más SHOT, ya que el segundo método es mucho más rápido en tiempo de ejecución que el primero, como se ve en la Tabla 6.5, los resultados obtenidos son realmente decepcionantes comparado con los resultados obtenidos con el pipeline de ISS más PFH, siendo mucho más fiable y preciso en la alineación de escenas con los datasets probados.

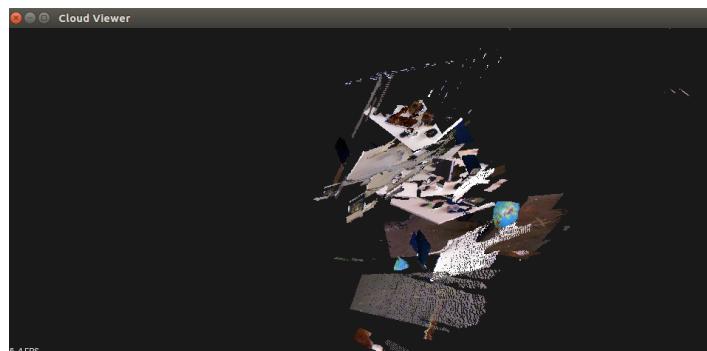
Parámetros	Valor
<code>iss_gamma_21_</code>	0.975
<code>iss_gamma_32_</code>	0.975
<code>iss_min_neighbors_</code>	25
<code>iss_threads_</code>	4
<code>iss_salient_radius_</code>	6 * model_resolution
<code>iss_non_max_radius_</code>	4 * model_resolution

**Tabla 6.6:** Parámetros del pipeline del registro de pares de nubes utilizado (ISS más PFH).

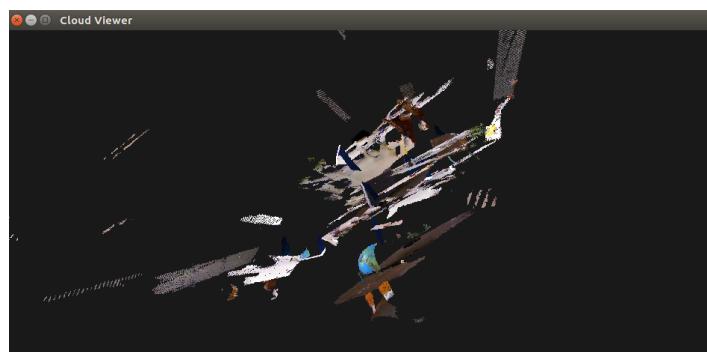
Estos parámetros que se muestran en la Tabla 6.6, son los que mejores resultados han dado con los diferentes datasets que se han utilizado, pero siempre pueden haber situaciones en los cuales puedan fallar y no hagan una buena alineación entre escenas después del registro. Durante las experimentaciones se han utilizado dos tipos de nubes de puntos: las que solo tienen los objetos recortados y las que son las escenas completas. El resultado obtenido es que con los objetos recortados las alineaciones entre nubes de puntos tienden más a fallos que con las nubes completas. Esto se debe a la falta de puntos característicos que puede haber entre los pares. Utilizando las escenas completas se puede conseguir una alineación mucho más fiable, pero más lenta, ya que son muchos más puntos que procesar, para poder solucionar este último problema lo que se hace es aplicar el algoritmo de VoxelGrid con tal de reducir el número de puntos de las escenas completas y que el registro entre pares de nubes sea mucho más rápido.

## 6.6 Eliminación del error acumulado al transportar las nubes de puntos al sistema de la nube origen

El transporte de una nube de puntos cualquiera a la nube original o nube cero se hace usando las matrices de transformación. Por lo tanto, si se quiere transportar la nube  $N^n$  al sistema de coordenadas de la nube  $N^1$  se hace uso de la Ecuación 5.7 donde  $T_1^n$  es la matriz para transportar del sistema  $n$  al 1 y  $T_n^{(n+1)}$  es la matriz que combina la matriz que se obtiene de RANSAC y ICP. Pero cuando las matrices  $T_1^n$  se van combinando con las  $T_n^{(n+1)}$  para optimizar el programa. Pero surge un problema y es que el error se va acumulando en las matrices, exactamente en la última columna que es el vector de traslación llegando a valores desorbitados, dando a una escena global donde las subescenas están separadas y sin coherencia ninguna como en las Subfiguras 6.14 y 6.15.



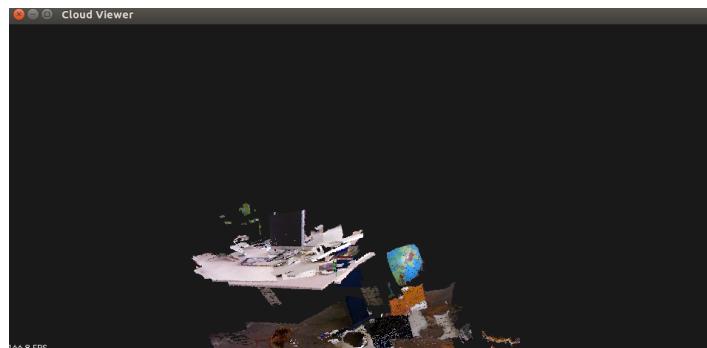
**Figura 6.14:** Mala alineación por acumulación de error en las matrices de transformación(i).



**Figura 6.15:** Mala alineación por acumulación de error en las matrices de transformación(ii).

La solución propuesta para eliminar el error acumulado en las matrices de transformación ha sido la siguiente: primero se evita aplicar las matrices obtenidas en RANSAC a las nubes de puntos, para luego pasarla como transformación inicial al algoritmo ICP. Cuando el ICP termina, se comprueba que las nubes han congeñado con la función **icp.hasConverged()** y si es “true” se coge la matriz obtenida con ICP, que es la combinación de la matriz inicial pasada al algoritmo y la que ha calculado el algoritmo, y se aplica a la nube “source” o fuente y a la nube de puntos de los objetos recortados respectiva. Con este método se evita que el error se vaya acumulando en el producto de las matrices de transformación y que no hayan valores desproporcionados en ellas: El resultado de este método se puede ver en la Figura 6.16, la cual es una escena donde las subescenas han congeñado y no están separadas como las anteriores figuras.

---



**Figura 6.16:** Escena final bien alineada.

## 6.7 Combinación de nubes puntos para formar la escena completa

En este apartado se describirá cómo se han combinado las diferentes escenas para formar el mapa completo. Para ello, primero se hace uso del registro de pares nubes utilizando uno de los métodos mencionados anteriormente, ya que obteniendo unas mejores transformaciones entre las escenas se conseguirán que todas ellas estén alineadas.

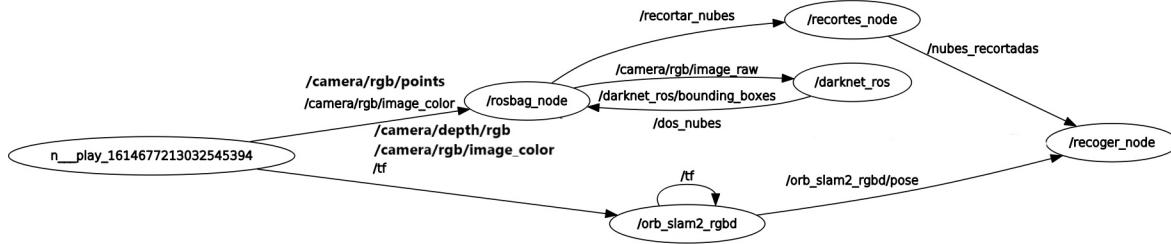
Cuando todas las escenas hayan pasado por el registro de pares de nubes y se les haya aplicado las dos matrices de transformación que surge durante el proceso, se irán añadiendo en vector que luego se usará para ir obteniendo cada escena. Al final, para obtener la escena completa se puede hacer de dos maneras:

1. Utilizando el operador suma (“+”), el cual va añadiendo en un `pcl :: PointCloud < pcl ::PointXYZRGB > :: Ptr` cada una de las nubes del vector, para finalmente mostrar la escena completa con todas las nubes de puntos unidas en una misma variable.
2. Hacer uso de un `pcl::visualization:: PCLVisualizer`, donde cada una de las escenas se van añadiendo para que finalmente se vean todas en un mismo visualizador.

Finalmente, el método elegido es usar el operador suma para ir añadiendo en una variable las nubes de puntos, de esta manera se podrá obtener la escena final de una manera mucho más sencilla que el otro método propuesto.

## 6.8 Resultados del desarrollo del pipeline

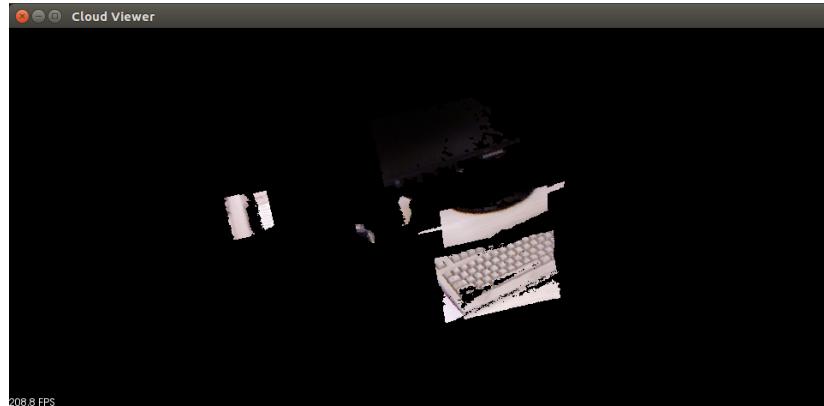
En este último apartado se va a mostrar el resultado final del pipeline que se ha desarrollado en este trabajo. Primero, se mostrará el gráfico de RQT de los nodos de ROS para ver como se relacionan cada uno de los nodos propuestos para el pipeline del proyecto y en última instancia se mostrará el resultado final, el cual consta de dos partes, la primera es la detección de objetos y su obtención de las nubes de puntos de las escenas y la segunda es la creación de la escena a partir de las matrices de ORB-SLAM2 y el pipeline de registro de pares de nubes, el cual ajustará las escenas entre ellas.



**Figura 6.17:** Gráfico final del pipeline.

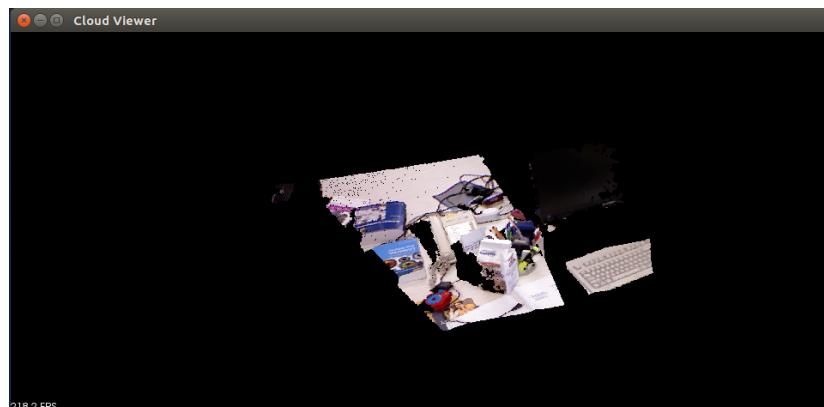
Como se ve en la Figura 6.17, están todos los nodos que intervienen en el pipeline de tiempo real con los topics desde donde reciben y envían los mensajes con los datos pertinentes, a continuación, se explicarán brevemente de izquierda a derecha. El primer nodo es el que contiene los topics del archivo rosbag que está publicando toda la información mediante mensajes de ROS, los cuales son leídos en los nodos “rosbag\_node” y “orb\_slam\_rgbd”, el primero se encarga de coger la imagen y la nube de puntos con el mismo time-stamp, la imagen se enviará al nodo de “darknet\_ros”, para que este devuelva al nodo “rosbag\_node” el bounding box con las imágenes detectadas. El mismo nodo “rosbag\_node” reenviará un mensaje personalizado con el mensaje de los bounding box de los objetos detectados y la nube de puntos correspondiente al nodo “recortes\_node”, para recortar los objetos de la nube de puntos. Por último, el nodo “recoger\_node” leerá los mensajes de las nubes recortadas y de las matrices de transformaciones de ORB-SLAM2, que vienen del nodo “orb\_slam\_rgbd”, de manera sincronizada mediante el time-stamp. El nodo “recoger\_node” irá haciendo el trabajo de unir cada una de las escenas que vayan llegando, en cambio, si entre escenas hay un cierta diferencia en las matrices de transformación de ORB-SLAM2 se les aplicará el registro de nubes de puntos para poder alinearlas.

La primera parte de los resultados es la detección de objetos y su posterior obtención de las nubes de puntos se realizan en varios nodos de ROS, exactamente en los nodos “darknet\_ros” y “rosbag\_node”. Los resultados obtenidos pueden ser bastante buenos como la Figura 6.18, dado que los objetos han sido recortados de la escena general y se les ha eliminado su background, que se realizan en el nodo “recortes\_node”.



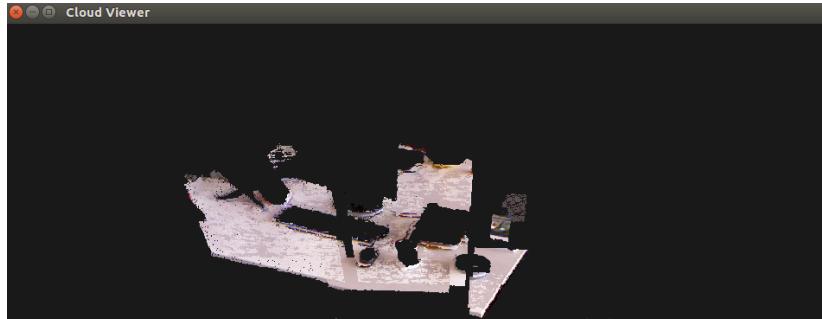
**Figura 6.18:** Nube de puntos de los objetos bien recortadas.

En cambio, pueden surgir situaciones en las que la eliminación de background no se haya hecho como en las Figuras 6.19 y 6.20, ya que en la Figura 6.19 se ve que el background de los libros en la mesa no se ha eliminado. Eso se debe a que en el algoritmo la nube resultante después de haberla recortado no había suficientes puntos, así que se ha vuelto a la nube original. En cambio, en la Figura 6.20 se han quitado los objetos en vez de los puntos situados en el background del objeto detectado. Estos problemas pueden surgir cuando se ha hecho un filtrado de clases de las nubes de puntos y no se ha recortado el objeto, que tras haber recortado la nube de puntos resultante no tenía suficiente puntos y se vuelve a la nube de puntos original o que se ha recortado mal el objeto a causa del umbral utilizado, dado que no siempre va a funcionar con todos los recortes de objetos.



**Figura 6.19:** Nube de puntos con objetos no recortados.

---

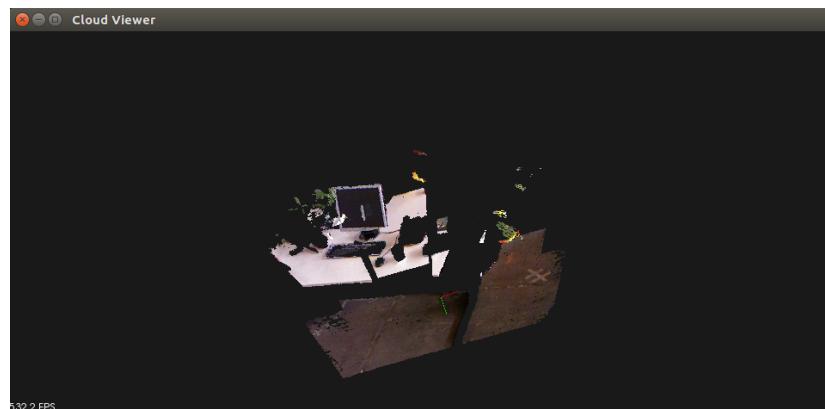


**Figura 6.20:** Nube de puntos con objetos mal recortados.

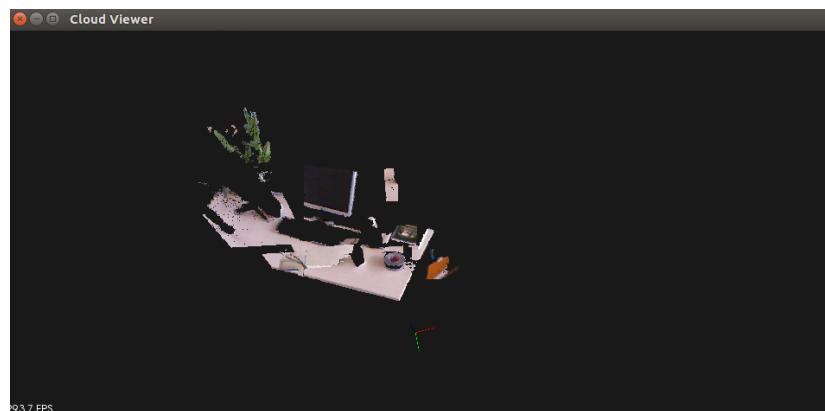
La segunda parte de los resultados es la generación de la escena final con los objetos de cada subescena, para ello se irán uniendo todas las escenas en una nube de puntos. Pero si entre un par de escenas están más separadas de lo normal, se les aplicará el registro de pares de nubes, para alinearlas correctamente, pero se puede dar que entre escenas no haya suficientes puntos para poder alinearlas, porque entre escenas puede haber una diferencia de time-stamp considerable. Por lo tanto, si el registro sale mal la nube será rechazada y se continuará con la siguiente. Todo este trabajo se hace en el nodo “recoger\_node”.

En las siguientes figuras se muestran diferentes resultados con diferentes datasets, exactamente de los datasets “**rgbd\_dataset \_freiburg1\_xyz-2hz- with-pointclouds.bag**” y “**rgbd\_dataset \_freiburg1\_xyz-2hz -with-pointclouds.bag**” de TUM, ya que debido al tiempo de publicación de los topics no siempre se van a conseguir los mismos resultados. Para obtener una escena final con varias subescenas, lo que se debe hacer es que el archivo rosbag publique los mensajes con una frecuencia mayor y para poder modificar eso hay que añadir “**-hz=1000**” en la ejecución del archivo rosbag, de la siguiente manera “**rosbag archivo.bag -hz=1000**”. La frecuencia predeterminada en la publicación de mensajes es de 100 hz, en cambio si se quiere que la publicación de mensajes sea más lenta, hay que aumentar la frecuencia y si se quiere que sea más rápida, se tiene que reducir el valor.

Los resultados para el dataset “**rgbd\_dataset \_freiburg1 \_xyz- 2hz -with- pointclouds.bag**” son los que se ven en las Figuras 6.21 y 6.22. Como se puede ver se ha realizado una alineación de subescenas que llevan a una escena final donde contiene los objetos recortados, pero en las otras figuras se puede ver que hay algunos objetos que se han recortado mal. Por ejemplo el teclado que hay sobre el escritorio, ya que se ha dejado la mesa y se quitado el teclado. Pero el resultado final puede concluir con una escena coherente donde la mayoría de objetos han sido detectados, alineados y ubicados.



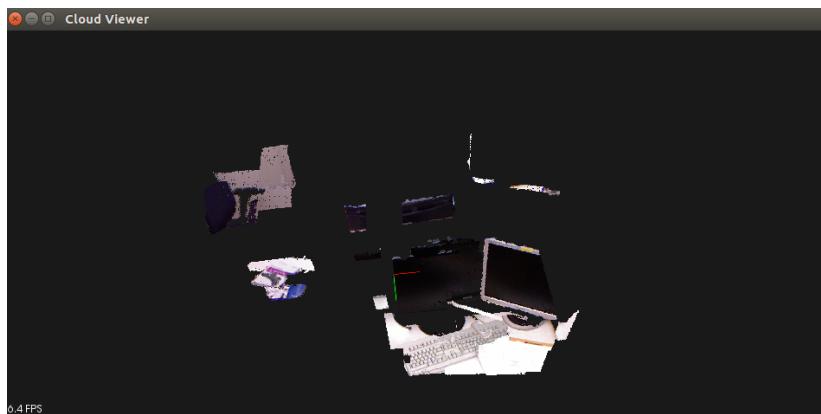
**Figura 6.21:** Escena final(i).



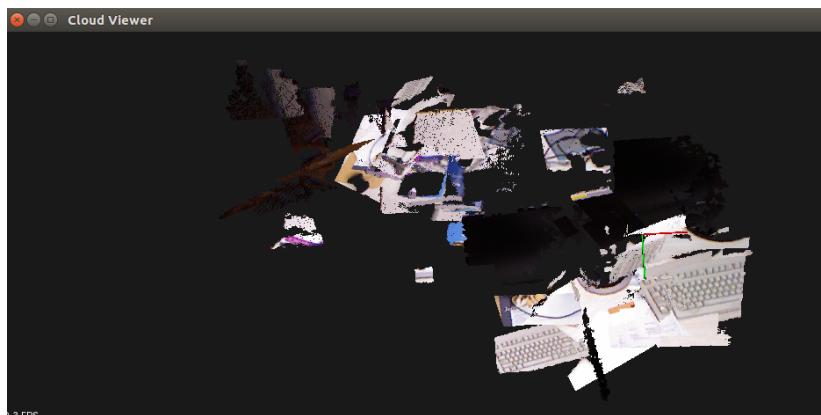
**Figura 6.22:** Escena final(ii).

En cambio, los resultados para el segundo dataset fueron diferentes, dado que han habido resultados buenos como en las Figuras 6.23, que se ejecutó el archivo rosbag con una frecuencia de 1000 hz para que fuese más lento la publicación de mensajes y se pudiese formar una escena más grande. Pero estos casos de resultados suelen ocurrir en poca frecuencia, ya que la mayoría de resultados obtenidos suelen ser como el de la Figura 6.24, donde las subescenas no se han alineado del todo bien y se pueden ver objetos duplicados y mal posicionados.

---



**Figura 6.23:** Escena final(iii).



**Figura 6.24:** Escena final mal alineada.

Por lo tanto, las conclusiones obtenidas de los resultados finales del pipeline desarrollado son que dependiendo del dataset de los resultados pueden ser muy variados y hay que elegirlo bien para poder obtener buenos resultados, dado que pueden salir bien como las Figuras 6.22 y 6.23, que son escenas finales que han quedado bien alineadas y se ha podido destacar los objetos detectados por la red neuronal YOLO Darknet. En cambio, se han encontrado fallos en la ejecución del pipeline dando forma a escenas mal alineadas como la Figura 6.24. Este mal resultado se puede deber a dos puntos, el primero y más posible es que se haya hecho un registro de pares de nubes entre escenas que no haya conseguido alinear las dos nubes perfectamente y el segundo punto es que la aplicación de matrices de transformación para poder traer una nube de puntos al sistema de coordenadas de la primera escena cogida en el pipeline falle provocando un efecto en cadena de mal posicionamiento de las subescenas que vayan llegando al pipeline.

## 7 Conclusiones

Este trabajo debe concluir resumiendo, en primer lugar, la idea principal por la cual se ha elegido este trabajo y el porqué se realiza, y en segundo lugar, analizar los resultados obtenidos y comprobar que se cumplen las metas planteadas para el proyecto.

Las ideas u objetivos propuestos para este proyecto son la detección de objetos en imágenes 2D, para su posterior segmentación y eliminación de background de sus respectivas escenas 3D. Esto se consigue haciendo uso de un paquete de ROS de Darknet para la detección y de la Point Cloud Library (PCL) para poder segmentarla de la escena y poder eliminar los puntos que no formen parte del objeto. Otro concepto que se usa es la inclusión de SLAM para el mapeado y localización simultáneos y obtener también un posicionamiento inicial de la escena, lo cual se consigue haciendo uso de un paquete de ROS de ORB-SLAM2. Por último, la idea principal es que ambas ideas anteriores se junten para poder desarrollar un sistema que vaya formando un escena donde solo estén los objetos detectados, creando una escena con mucha más información que las que crea un SLAM, todo esto se realizará con un pipeline en tiempo real gracias a ROS.

En la parte de los resultados, se concluye que la detección de objetos 2D y su posterior segmentación de los objetos de la escena se realizan de una manera bastante exitosa. Pero en la eliminación de puntos que no forman parte del objeto, los resultados obtenidos son variados, dado que hay que tener en cuenta qué tipo de clase es el objeto, como está situado y el algoritmo que se usa para procesar la nube de puntos. Por otro lado, para recrear la escena final se han tenido muchos más problemas. Uno de ellos es la sincronización de los datos de los topics para que cuadrasen en el tiempo de publicación y poder hacer un sistema en tiempo real, ya que al principio no se encontró una solución para poder hacerlo todo en tiempo real, pero al final rebajando la resolución de las nubes de puntos y cambiando la política de sincronización a Approximate Time Policy se consigue tener un pipeline que se ejecute en tiempo real.

Otro problema en la recreación de la escena final es que el SLAM obtenga todas las matrices de transformación de manera continua, dado que hay situaciones en las cuales se reinicia el proceso o deja de detectar las keypoints en las imágenes y sin la matriz de ORB-SLAM2 no se podría reconstruir la escena.

Por último, dependiendo del dataset puede haber problemas durante el registro de nubes de puntos hay un problema y es si un par de escenas no tienen los suficientes puntos característicos para poder alinearlos. Esto puede ser causado por una bastante diferencia de tiempo de cuando fueron tomadas las escenas o que los parámetros escogidos no son los correctos, el primer problema se intenta solucionar o minimizar el problema cuando se realiza ICP y se obtiene con el valor de FitnessScore del resultado de ICP, ya que si el valor de la función supera un umbral la nube queda descartada y se continua con el proceso. En cambio, con los parámetros se ha hecho una experimentación a fondo, por tal de encontrar el pipeline con los métodos de extractores de características, computador de descriptores y parámetros que

mejores resultados han dado, pero aunque se hayan encontrado los mejores resultados puede haber en ciertas ocasiones en las que no puedan servir para alinear totalmente las escenas, a causa de una mala matriz de transformación obtenida por RANSAC e ICP, lo cual puede repercutir en las próximas nubes de puntos cuando se tengan que alinear con la primera escena y rechazando el resultado obtenido.

Como conclusión final del proyecto, las motivaciones que han llevado a hacer este Trabajo de Fin de Grado han sido cubiertas. Realizando en este proyecto la combinación de varias ramas de las tecnologías emergentes y técnicas como son la detección de objetos en imágenes 2D, la segmentación de objetos 3D y el mapeado y la localización simultáneas, todo bajo el manto de ROS. Aunque el objetivo se cumple llegando a obtener un pipeline en tiempo real que detecta los objetos de un escena, los segmenta de ella y va reconstruyendo un mapa de nube de puntos donde se encuentran dichos objetos detectados, los resultados finales obtenidos no son los cien por cien esperados, dado que en ciertos datasets funciona correctamente obteniendo una buena escena final y en otros hay algunos fallos por los problemas mencionados en el registro de pares de nubes o en SLAM. Pero, al fin y al cabo, este proyecto podría ser en el futuro un pequeño paso para un gran desarrollo.

# Bibliografía

- Bagnat, J. I. (2020). *Modelos de detección de objetos.* <https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos>.
- Bjelonic, M. (2016–2018). *YOLO ROS: Real-time object detection for ROS.* [https://github.com/leggedrobotics/darknet\\_ros](https://github.com/leggedrobotics/darknet_ros).
- Bochkovskiy, A., Wang, C.-Y., y Liao, H.-Y. M. (2020). *Yolov4: Optimal speed and accuracy of object detection.*
- Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., y Zagoruyko, S. (2020). *De tr: End-to-end object detection with transformers.* <https://github.com/facebookresearch/detr>.
- Concha, A., y Civera, J. (2017). *Rgbdtam: A cost-effective and accurate rgb-d tracking and mapping system.*
- Durrant-Whyte, H., y Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2), 99-110. doi: 10.1109/MRA.2006.1638022
- Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation.*
- Library, P. C. (s.f.-a). *Color-based region growing segmentation.* Descargado de [http://pointclouds.org/documentation/tutorials/region\\_growing\\_rgb\\_segmentation.html#region-growing-rgb-segmentation](http://pointclouds.org/documentation/tutorials/region_growing_rgb_segmentation.html#region-growing-rgb-segmentation)
- Library, P. C. (s.f.-b). *Conditional euclidean clustering.* Descargado de [http://pointclouds.org/documentation/tutorials/conditional\\_euclidean\\_clustering.html#conditional-euclidean-clustering](http://pointclouds.org/documentation/tutorials/conditional_euclidean_clustering.html#conditional-euclidean-clustering)
- Library, P. C. (s.f.-c). *Plane model segmentation.* Descargado de [http://pointclouds.org/documentation/tutorials/planar\\_segmentation.html#planar-segmentation](http://pointclouds.org/documentation/tutorials/planar_segmentation.html#planar-segmentation)
- Library, P. C. (s.f.-d). *Region growing segmentation.* Descargado de [http://pointclouds.org/documentation/tutorials/region\\_growing\\_segmentation.html#region-growing-segmentation](http://pointclouds.org/documentation/tutorials/region_growing_segmentation.html#region-growing-segmentation)
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., y Dollár, P. (2018a). *Focal loss for dense object detection.*
- Lin, T.-Y., Goyal, P., Girshick, R., He, K., y Dollár, P. (2018b). *Retinanet.* <https://www.paperswithcode.com/method/retinanet>.

- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., y Berg, A. C. (2016). Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, 21–37. Descargado de [http://dx.doi.org/10.1007/978-3-319-46448-0\\_2](http://dx.doi.org/10.1007/978-3-319-46448-0_2) doi: 10.1007/978-3-319-46448-0\_2
- Mur-Artal, R., y Tardós, J. D. (2017). ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5), 1255–1262. doi: 10.1109/TRO.2017.2705103
- Pcl pointcloud pairwise registration. (s.f.). [http://pointclouds.org/documentation/tutorials/registration\\_api.php](http://pointclouds.org/documentation/tutorials/registration_api.php).
- Redmon, J. (2013–2016). Darknet: Open source neural networks in c. <http://pjreddie.com/darknet/>.
- Redmon, J., y Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.
- Rusu, R. B., y Cousins, S. (2011, May 9-13). 3D is here: Point Cloud Library (PCL). En *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China.
- Sturm, J., Engelhard, N., Endres, F., Burgard, W., y Cremers, D. (2012, Oct.). A benchmark for the evaluation of rgb-d slam systems. En *Proc. of the international conference on intelligent robot systems (iros)*.
- Xianzhi Du, S. E., y Jaeyoun Kim, G. R., Technical Program Manager. (2020). SpineNet: A novel architecture for object detection discovered with neural architecture search. *Google AI Blog*.

## **Lista de Acrónimos y Abreviaturas**

<b>CNN</b>	Convolutional Neural Network.
<b>DETR</b>	Detection Transformer.
<b>EFK</b>	Extended Filter of Kalman.
<b>ICP</b>	Iterative Closest Point.
<b>IEEE</b>	Institute of Electrical and Electronics Engineers.
<b>IoU</b>	Intersection over Union.
<b>NMS</b>	Non Maximum Suppression.
<b>ORB</b>	Oriented FAST and Rotated BRIEF.
<b>PCL</b>	Point Cloud Library.
<b>R-CNN</b>	Region Based Convolutional Neural Network.
<b>RANSAC</b>	Random Sample Consensus.
<b>RGB</b>	Red, Green and Blue.
<b>RGB-D</b>	Red, Green and Blue - Depth.
<b>RGBDTAM</b>	RGB-D Tracking and Mapping.
<b>ROS</b>	Robot Operating System.
<b>SLAM</b>	Simultaneous Localization And Mapping.
<b>SSD</b>	Single Shot Detector.
<b>TFG</b>	Trabajo Final de Grado.
<b>TUM</b>	Technical University of Munich.
<b>YOLO</b>	You Only Look Once.