Ejercicio 6 – Visión 3D

Este ejercicio tiene como objetivo aplicar los conceptos aprendidos en el Tema 8: Visión 3D.

La defensa del ejercicio se hará en clase, y hay que entregar un archivo **pcl_node_p6.cpp** con el código generado que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

Utilizando el simulador con Tiago, se pide crear un programa que trabaje con el PointCloud generado por la cámara de profundidad. En esta sesión vamos, a trabajar con nubes de puntos 3D utilizando la librería PCL y RViz2.

Utilizando el código de ejemplo del archivo **pcl_node.cpp** proporcionado en el Docker, se pide:

1. **Configuración del entorno**

Añadir la siguiente línea en la función **topic_callback_3d**, antes de publicar los datos, esto copia la cabecera del mensaje de entrada en el de salida y evita errores en RViz2:

```
output.header = msg->header;
```

Abre RViz2 (ejecutar **rviz2** en la terminal) y carga la configuración (**config_cv.rviz**) que se proporciona dentro del workspace. Esto nos carga los topic necesarios para visualizar las imágenes (original y modificada en cv_node), las nubes de puntos PointCloud2 (original y modificada en pcl_node) y el modelo del robot, entre otros.

Compila el paquete (**computer_vision**) y ejecuta este nuevo programa (**pcl_node**) generado.

Comprueba que puedes visualizar la nube de puntos de salida, deberá ser igual que la de entrada. En algunas ocasiones, la calidad del servicio puede hacer que no se actualice correctamente los datos. Si esto ocurre, en RViz2 tenemos que seleccionar el mensaje deseado, en este caso **PCL 3D PointCloud2**, y dentro del apartado **Topic**, modificar la **Reliability Policy** de **Best Effort** a **Reliable**.

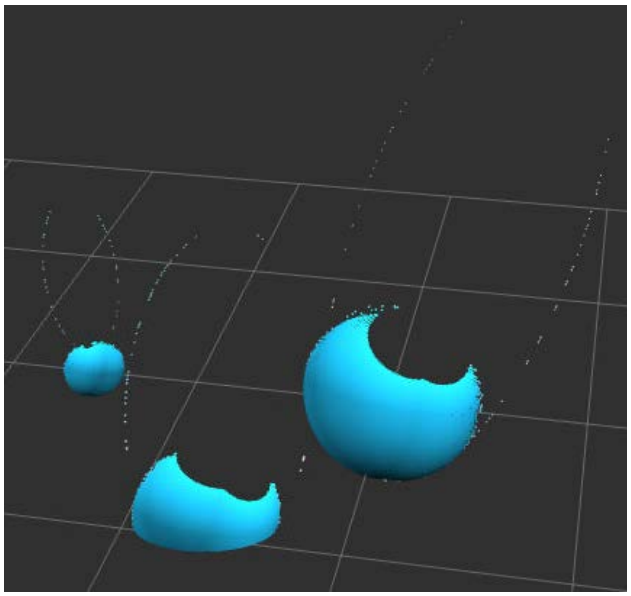
2. **Filtro de color**

Utilizando el PointCloud de entrada en formato **pcl::PointXYZRGB**, se pide crear un filtro de color que únicamente muestre el color azul de la pelota.

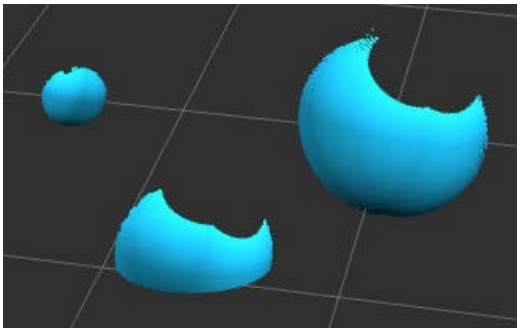
En caso de necesitar hacer un cambio del espacio de color, existen múltiples funciones implementadas dentro de PCL que nos facilitan el trabajo. Para más detalles, ver:

https://pointclouds.org/documentation/namespacepcl.html

Dependiendo de la precisión del filtrado, puede que haya valores anormales (outliers).



Filtro de color con outliers



Filtro de color sin outliers

3. Eliminación de valores outliers

Para eliminar los valores outliers, existen una serie de filtros, como el estadístico. El uso de este filtro puede verse aquí:

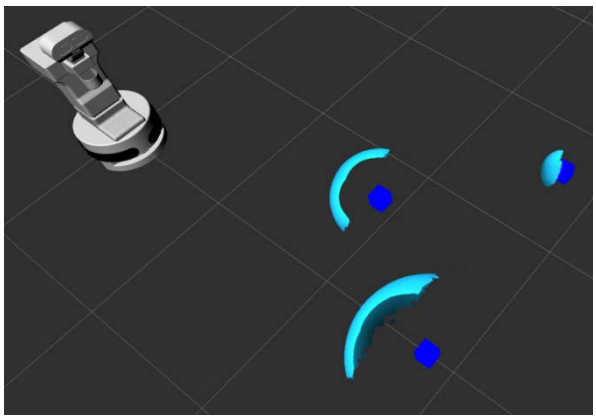
https://pcl.readthedocs.io/projects/tutorials/en/latest/statistical_outlier.html

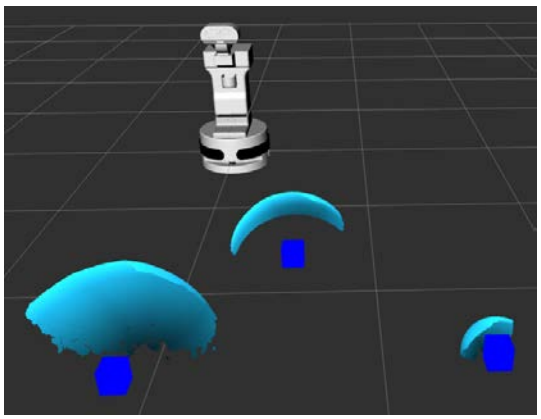
Este programa genera dos nubes de puntos con los inliers y outliers detectados por el filtro estadístico. Nosotros nos quedaremos únicamente con los inliers, cuyo resultado será la nube de puntos obtenida tras el filtro de color sin outliers.

4. Detección de esferas

En este apartado, se pide detectar tantas esferas como haya en la escena, y de cada una extraer los coeficientes que pertenecen a la posición [X,Y,Z] y al radio R.

Una vez detectadas, se dibujará un cubo de color azul en dicha posición. Para ello, puedes crear un método que genere un cubo insertando puntos en el PointCloud con cierto tamaño.





En la detección de las esferas, se quiere utilizar **RANSAC**. Hay varias formas de hacerlo, bien utilizando RandomSampleConsensus, o bien mediante extracción de índices de forma genérica SACSegmentation.

Un ejemplo de **RandomSampleConsesus**, donde se obtiene la nube de puntos que mejor se ajusta a un modelo tanto de plano, como de esfera, puede verse aquí:

https://pcl.readthedocs.io/projects/tutorials/en/latest/random_sample_consensus.html

En el caso de la extracción de índices con **SACSegmentation**, en el tutorial de PCL tienes un ejemplo donde **se extraen planos** hasta que la nube de puntos original es reducida a menos del 30% de la nube de puntos original:

https://pcl.readthedocs.io/projects/tutorials/en/latest/extract_indices.html

En dicho ejemplo, se utiliza un objeto **seg** de la clase **pcl::SACSegmentation<pcl::PointXYZ>** para la segmentación del plano. El resultado de la segmentación **seg.segment()** son los coeficientes de la ecuación del plano y una lista de índices a los puntos de la nube que pertenecen a dicho plano.

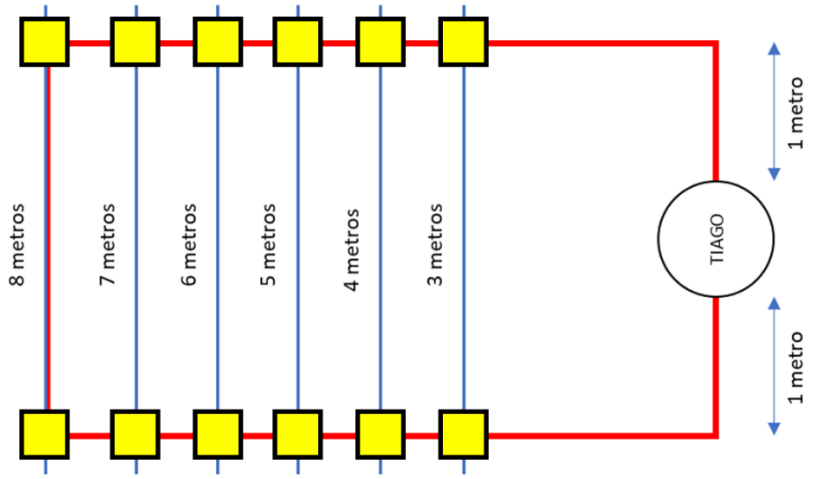
Observa que para recuperar dichos puntos a partir de la lista de índices, se utiliza un filtro **extract** del tipo **pcl::ExtractIndices<pcl::PointXYZ>**. Aunque en nuestro caso, todos los puntos serán **pcl::PointXYZRGB**.

En el caso de necesitar ajustar a otros modelos, los proporcionados por PCL pueden verse aquí:

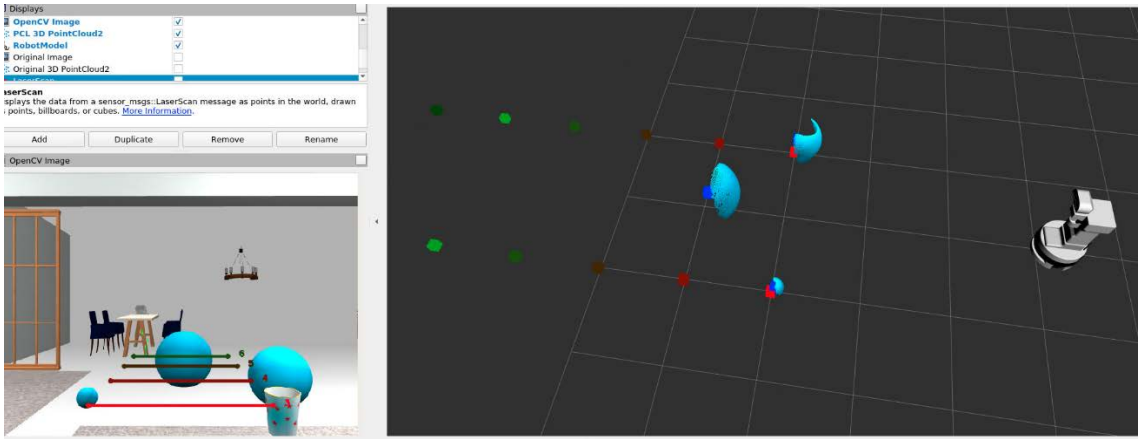
https://pointclouds.org/documentation/namespacepcl.html#a801a3c83fe807097c9aded5534df1394

5. Incluir cubos

Finalmente, se pide incluir cubos en las posiciones 3D que tenían de los puntos utilizados en la práctica anterior, los usados para generar proyecciones. En este caso, se dibujarán únicamente un cubo por cada punto, los cuales estarán distanciados 1 metro a cada lado del robot, e irán a una distancia frontal del robot desde los 3 metros hasta los 8 metros, separadas 1 metro entre sí. Además, los cubos tendrán colores según su distancia (misma línea, mismo color).



Una imagen del resultado final, junto con la de la práctica anterior puede verse a continuación:



Si se superpone el PointCloud original, puede verse lo siguiente: