VISIÓN ARTIFICIAL Y ROBÓTICA

Práctica 1. Mapeado 3D de Interiores y Sigue Líneas

Santiago Nogales Chiarenza 74011341D
Miguel Rodriguez Sanchez 48717878Z

1. Introducción

El propósito principal de esta primera práctica es la creación de un sistema que permita a un robot turtlebot3 con una cámara Creative, recolectar información de un entorno simulador y realizar un mapeo 3D. Para ello, seguiremos una pipeline de desarrollo que explicaremos más adelante, junto con el uso de herramientas como:

1. **ROS** (Robot Operating System), un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo. En este proyecto, ROS se utilizará para gestionar la comunicación entre los diferentes componentes del sistema, como el robot, la cámara Kinect y el software de procesamiento de nubes de puntos.

2. El simulador de robótica 2D/3D de código abierto **Gazebo** (que lo lanzaremos usando ROS). Utilizaremos Gazebo para simular el entorno en el que el robot con la cámara Kinect recogerá datos para el mapeo 3D, concretamente simularemos una habitación con diferentes objetos en ella y obtendremos los resultados de ahí.

3. **C++** y la librería **Point Cloud Library (PCL)**, una herramienta potente para procesar datos de nubes de puntos, que ofrece una variedad de algoritmos para el registro, segmentación y reconstrucción en 3D. Estos algoritmos serán fundamentales para generar un mapa tridimensional preciso del entorno simulado.

4. **Github** como gestor de versiones.

A lo largo de esta práctica explicaremos y experimentaremos con diferentes formas de realizar el mapeo 3D. Emplearemos diferentes tipos de algoritmos y técnicas y compararemos los resultados para obtener los mejores.

2. Propuesta

Antes de comenzar con la práctica, vamos a explicar qué pasos hemos seguido y unas ligeras definiciones de lo que es lo que estamos haciendo.

Para empezar, expliquemos que es el mapeo 3D. Se refiere al proceso de crear representaciones visuales de objetos tridimensionales en un espacio, es decir, proyectar las características tridimensionales de un objeto en un plano o superficie, con el objetivo de crear una representación visual realista y comprensible del objeto.

Para crear esta representación, debemos de emplear una serie de pasos o pipelines. Una pipeline es una secuencia de procesos interconectados que se utilizan para realizar tareas específicas de manera eficiente y automatizada. En nuestro caso, hemos seguido el pipeline tradicional:

En primer lugar, mediante la cámara Creative, obtendremos nubes de puntos, es decir, un conjunto de puntos de datos en el espacio. Suele referirse a un conjunto de vértices en un sistema de coordenadas tridimensionales. Estos puntos son, en general, una representación de la superficie externa de un objeto o una escena.

Siguiendo nuestro pipeline, el siguiente paso es obtener los keypoints, pero antes debemos de realizar un **subsampling.** Este proceso intermedio trata de eliminar todos los puntos innecesarios ya que de los muchísimos que hay, muchos son innecesarios. En nuestro caso, usamos un VoxelGrid que devuelve la media de los puntos que hay dentro de un voxel con un tamaño que nosotros introducimos y reduce la densidad de la nube de puntos.

Por otro lado debemos de calcular las **normales** de las superficies, que son una serie de vectores perpendiculares a dicha superficie, que dependiendo del parámetro de vecindad, tendremos unas normales más precisas o no. Con una vecindad muy grande, las normales se van a ver influenciadas por estructuras muy lejanas. Con una vecindad demasiado pequeña, puede que la normal no represente fielmente el plano debido al ruido de los datos.

Una vez calculamos estos pasos internos, procedemos a calcular los **keypoints**, o puntos clave, que son ubicaciones específicas en la imagen que son utilizadas para capturar características significativas. Estos puntos son elegidos por su habilidad para reflejar detalles importantes de la imagen, como esquinas, bordes o regiones texturizadas distintivas dependiendo del algoritmo que utilicemos. Posteriormente presentaremos qué algoritmo utilizamos y su experimentación.

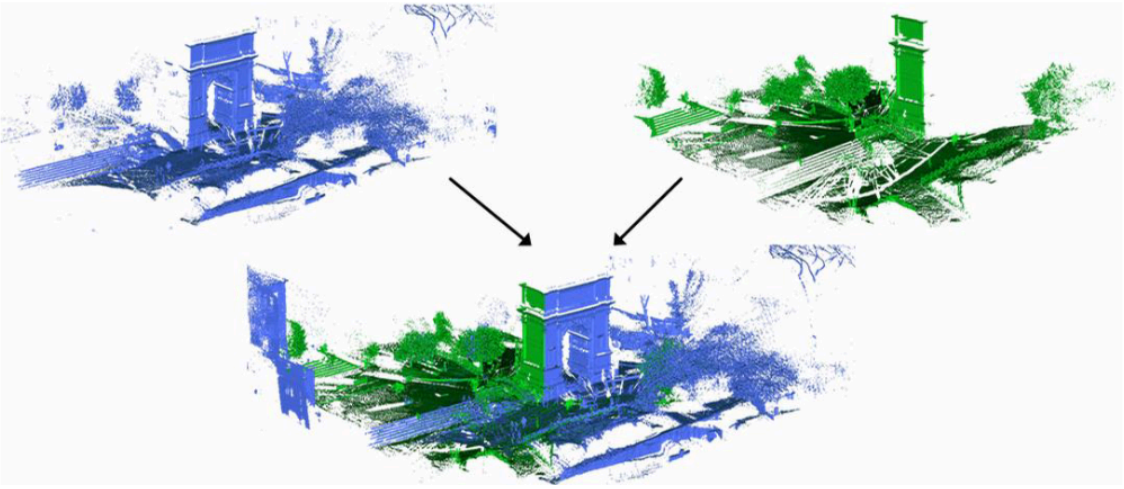
Seguidamente debemos calcular los **descriptores** de los keypoints. Un descriptor es una representación numérica compacta de una imagen (un vector) o de una parte significativa de ella. Estos descriptores resumen la información visual esencial de las imágenes de manera que se pueda utilizar para tareas como reconocimiento, clasificación o comparación de imágenes. Son herramientas cruciales para transformar datos visuales crudos (como los píxeles de una imagen) en un formato más manejable y útil para realizar diversas operaciones computacionales.

Una vez tenemos todo esto, debemos de realizar un emparejamiento o **matching**, es decir, calcular qué puntos del objeto se corresponde con qué puntos de la escena. Debemos buscar para cada descriptor del objeto, su descriptor más cercano de la escena. Los puntos que se corresponden deben estar en el mismo lugar del espacio de los descriptores.

El paso previo es la aplicación de **RANSAC** (Random Sample Consensus). RANSAC es un método que facilita el cálculo de los parámetros de un modelo a partir de muestras con ruido, y puede ser utilizado en una variedad de contextos. Selecciona aleatoriamente un subconjunto de puntos de la nube de puntos, ajusta un modelo a este subconjunto, y luego evaluar la bondad de ajuste de este modelo respecto a los demás puntos de la nube. Este

proceso se repite múltiples veces para obtener el mejor modelo posible, ignorando los puntos que no se ajustan bien al modelo.

Finalmente, para mejorar aún más la precisión, empleamos el algoritmo de **ICP** (Iterative Closest Point). Este método se utiliza para reducir la distancia entre dos conjuntos de puntos. Se selecciona uno de los conjuntos como referencia y, de manera iterativa, se aplican transformaciones al otro conjunto para gradualmente minimizar una métrica de error, comúnmente la suma de las diferencias al cuadrado entre los puntos correspondientes. Al concluir, proporciona la transformación refinada que optimiza la alineación entre los conjuntos de puntos.



3. Experimentación

3.1. VoxelGrid

En primer lugar, vamos a probar a aumentar y disminuir el tamaño de los voxeles de nuestro método de subsampling. Voxelgrid es un tipo de subsampling que, de forma clara, es hacer la media de los puntos que están dentro de cada voxel de un tamaño definido, de forma que reducimos la cantidad total de puntos.

El voxelgrid que he utilizado en primer lugar es de 0.04. Vamos a probar ahora a subirlo a 0.1 y disminuirlo a 0.01 a ver qué es lo que ocurre. Todas las pruebas se han realizado sobre una nube de 2073600 puntos.

VoxelGrid 0.04: 8173 puntos finales
VoxelGrid 0.01: 94273 puntos finales
VoxelGrid 0.001: Leaf size is too small for the input dataset. Integer indices would overflow.
VoxelGrid 0.1: 1571 puntos finales

Como vemos, a mayor tamaño de voxel, tenemos menos puntos. Pero cuidado, si ponemos un voxel muy pequeño para ese dataset, podemos causar un overflow.

3.2. Keypoints

Para este apartado vamos a explicar los diferentes tipos de algoritmos de obtención de keypoints que hemos empleado junto con los resultados que nos aporta cada uno de ellos.

En primer lugar vamos a evaluar el algoritmo de **ISS (Intrinsic Shape Signatures)**, el cual trabaja evaluando la distribución geométrica de los puntos vecinos alrededor de cada punto en la nube. Busca puntos que tengan una distribución única de vecinos, indicando características geométricas distintivas como esquinas, bordes, o cambios abruptos en la superficie. Estos puntos son considerados importantes para entender la estructura subyacente y la forma del objeto escaneado.

Para hacer esto primero se realiza un análisis en toda la nube de puntos para identificar aquellos puntos que tienen propiedades únicas en términos de su distribución espacial local. Esto se hace generalmente calculando matrices de covarianza basadas en los vecinos más cercanos de cada punto y luego analizando los valores y vectores propios de estas matrices. Luego entre los puntos candidatos, se aplican criterios adicionales para seleccionar aquellos que tienen características más significativas. Esto puede implicar comparar la escala de las características locales y asegurar que los keypoints seleccionados sean representativos de las características más prominentes. Una de las ventajas del ISS es que los keypoints detectados son invariantes a transformaciones como traslaciones, rotaciones y cambios de escala, lo que los hace útiles para el reconocimiento de objetos y la registración de nubes de puntos en diferentes posiciones y orientaciones.

Para las pruebas, hemos ido cambiando los parámetros con el objetivo de comprobar tanto el tiempo que tarda en encontrar los puntos clave como la cantidad que encuentra.

Los parámetros que vamos a variar son:

- **Salient Radius**: Es la escala o tamaño de las características en una imagen que son consideradas significativas o destacadas. Se utiliza para definir el tamaño de la vecindad alrededor de un punto de interés.
- **NonMaxRadius**:Se refiere al radio utilizado en la supresión de no máximos (Non-Maximum Suppression, NMS). Esto ayuda a reducir el número de características detectadas redundantes, dejando solo las más prominentes.
- **Threshold21 y Threshold32**:Estos umbrales son específicos del algoritmo ISS y se utilizan para filtrar los puntos clave basándose en ciertas propiedades geométricas calculadas a partir de los autovalores del tensor de covarianza local.
- **MinNeighbors**: Este parámetro especifica el número mínimo de vecinos que un punto debe tener dentro del "salient radius" para ser considerado un punto clave en el algoritmo ISS.

Las nube contiene 88975 puntos tras voxelgrid

Salient radius	NonMax Radius	Threshold21	Threshold32	MinNeighbors	Tiempo (s)	Puntos

6	4	0.975	0.975	5	3.500	2685
2.5	2.5	0.95	0.95	3	2.988	6129
2	2	0.95	0.95	2	2.712	10455
8	6	0.98	0.98	5	4.334	1258
2.5	2.5	0.95	0.95	5	2.777	6087
2.5	2.5	0.50	0.95	5	2.647	3120
2.5	2.5	0.95	0.50	5	2.887	6097
10	2	0.95	0.95	3	4.683	9258
10	10	0.95	0.95	3	5.005	383
2.5	2.5	0.50	0.50	3	2.661	3163
3	3	0.8	0.9	4	3.003	4446
2.5	2.5	1	0.01	4	3.276	6243
2.5	2.5	0.01	1	4	2.628	1829
2.5	2.5	0.95	0.95	10	2.683	4116

Con los resultados obtenidos de las pruebas, se puede proceder a observar la tabla y se puede sacar varias conclusiones. Aumentar el Salient Radius y el NonMaxRadius tiende a disminuir el número de puntos clave detectados además de aumentar el tiempo necesario. Por ejemplo, cuando el Salient Radius es de 8 y el NonMaxRadius es de 6, el número de puntos clave disminuye a 1258, en comparación con cuando estos valores son más bajos, donde se detectan más puntos clave. Configuraciones con umbrales más altos (cercanos a 1) parecen producir menos puntos clave, como se muestra cuando Threshold21 y Threshold32 están en 0.975 y 0.980, respectivamente. Sin embargo, los umbrales extremadamente bajos o muy dispares (como 0.01 y 1) también pueden disminuir la cantidad de puntos clave, probablemente debido a que muchos candidatos son filtrados por no cumplir con los criterios de relevancia geométrica. Un número mayor de MinNeighbors reduce la cantidad de puntos clave detectados, lo cual se alinea con la idea de que solo las características con suficientes vecinos cercanos son consideradas significativas y se tiende a incrementar el tiempo de procesamiento, ya que el algoritmo tiene que verificar un mayor número de vecinos alrededor de cada punto clave potencial.. Por ejemplo, cuando MinNeighbors es 10, solo se encuentran 4116 puntos clave. Buscar un equilibrio en los parámetros puede llevar a resultados óptimos. Por ejemplo, con un Salient Radius y NonMaxRadius de 2.5, y un Threshold21 y Threshold32 de 0.95, se encuentra un número relativamente alto de puntos clave en un tiempo razonable. Esto sugiere que una configuración moderada puede ser más eficiente para la detección de puntos clave sin comprometer el tiempo de procesamiento.

Por otro lado, hemos probado el algoritmo de **SIFT (Scale-Invariant Feature Transform).** Este algoritmo de obtención de puntos clave es un detector de características en imágenes

que proporciona una sólida robustez frente a escala y rotaciones de las capturas de nubes de puntos.

SIFT utiliza filtros gaussianos que se aplican a un parche con diferentes desviaciones estándar. Esto genera diferentes versiones de ese parche, cada cuál más difuminada que la anterior. Se detectan los máximos y mínimos comparando el punto con sus vecinos más cercanos de la misma escala y de las escalas anterior y posterior, y se devuelven como keypoints si es mucho mayor o mucho menor que estos vecinos.

Los parámetros que vamos a utilizar son:

- **MinScale:** Define la desviación estándar inicial para la escala más pequeña en el espacio de escala de SIFT. Un valor menor detecta características más finas.
- **NrOctaves:** Número de octavas que SIFT utiliza para la detección de puntos clave. Cada octava duplica la escala, permitiendo la captura de características a diferentes tamaños.
- **NrScalesPerOctave:** Especifica la cantidad de escalas diferentes que se analizan dentro de cada octava. Un número mayor permite una búsqueda más granular de puntos clave entre escalas.
- **MinimumContrast:** Umbral mínimo de contraste para que un punto sea considerado un punto clave. Valores más altos ignoran puntos menos distintos, mientras que valores más bajos permiten una mayor sensibilidad.

Las nube contiene 88975 puntos tras voxelgrid

MinScale	NrOctaves	NrScalePer Octave	MinimumContrast	Tiempo (s)	Puntos
0.01	3	4	0.001	10.119	1311
0.01	3	4	0.01	9.793	1299
0.01	3	4	0.1	9.934	1273
0.01	4	4	0.001	10.217	1353
0.001	3	4	0.001	10.461	2733
0.01	3	6	0.001	11.599	2528
0.01	1	4	0.001	6.465	855
0.01	1	1	0.001	8.624	284

De la anterior tabla de resultados, podemos sacar en claro que el cambio en el MinScale de 0.01 a 0.001 incrementa notablemente la cantidad de puntos clave detectados (de un promedio de alrededor de 1200-1300 a 2733), lo que indica una mayor sensibilidad a las características más finas de la nube de puntos. Esto, sin embargo, conlleva un incremento en el tiempo de procesamiento, aunque no proporcionalmente grande (10.461 segundos

para el MinScale de 0.001). Aumentar el número de NrOctaves (de 3 a 4) tiene un efecto menor tanto en el tiempo de procesamiento como en el número de puntos clave (de 1311 a 1353 puntos). Esto nos dice que la inclusión de una mayor gama de escalas no impacta dramáticamente la eficiencia del algoritmo, pero sí permite capturar características adicionales que no son detectadas con menos octavas. Incrementar NrScalesPerOctave de 4 a 6 aumenta el tiempo de procesamiento (de 10.119 a 11.599 segundos) y también el número de puntos clave detectados (de 1311 a 2528). Esto implica que explorar más escalas dentro de cada octava proporciona una detección más detallada a costa de un mayor tiempo de procesamiento. El MinimumContrast tiene una influencia clara en la cantidad de puntos clave: al incrementar este valor de 0.001 a 0.1, la cantidad de puntos clave disminuye significativamente (de 1311 a 1273). Este parámetro sirve como un umbral para la selección de puntos clave y su aumento filtra puntos menos contrastados, lo cual puede ser beneficioso para evitar falsos positivos pero también puede excluir características válidas. La selección de los parámetros de SIFT debe equilibrar la necesidad de detectar una cantidad significativa de puntos clave con la restricción del tiempo de procesamiento disponible. Por ejemplo, un MinScale bajo y un NrScalesPerOctave más alto parecen ser preferibles para una detección detallada de puntos clave, mientras que el NrOctaves puede ser modificado con menor impacto en el tiempo. El MinimumContrast debe ser cuidadosamente ajustado para mantener un equilibrio entre la sensibilidad y la especificidad de la detección de puntos clave.

Harris es un detector de esquinas. Una esquina es una zona donde intersecan dos bordes (un borde es una zona que sufre un cambio brusco de iluminación), por lo tanto sólo funciona en imágenes de tono gris. No es robusto ante diferentes escalas. Se toman ventanas de NxN píxeles alrededor de cada pixel y se computará una puntuación Harris value. Los píxeles cuyo Harris value excede un umbral se consideran un punto de esquina.

Harris 3D propone una manera para calcular las escalas (los diferentes radios) de forma automática observando los vecinos más cercanos de cada punto. También propone un método para eliminar exceso de puntos en áreas concretas y para mantener los puntos en áreas cuyos puntos no superaron el filtro del umbral.

Las nube contiene 88975 puntos tras voxelgrid

Threshold	Tiempo (s)	Puntos
1e-12	5.101	21172
1e-9	4.875	21172
1e-6	2.597	7087
1e-3	0.666	337

La elección del umbral tiene un impacto significativo tanto en el tiempo de procesamiento como en la cantidad de puntos de esquina detectados. Un umbral extremadamente bajo como 1e-12 no filtra casi ningún punto, resultando en un alto número de puntos detectados (21172 puntos), pero requiere más tiempo de procesamiento (5.101 segundos), A medida que el umbral aumenta, el tiempo de procesamiento disminuye notablemente. Esto es

evidente al comparar el umbral de 1e-9 con el de 1e-6, donde el tiempo de procesamiento se reduce casi a la mitad (de 4.875 a 2.597 segundos) y la cantidad de puntos de esquina detectados disminuye drásticamente (de 21172 a 7087 puntos). Se debe encontrar un equilibrio entre la cantidad de puntos de esquina detectados y el tiempo de procesamiento. Umbrales muy bajos pueden no ser prácticos para aplicaciones en tiempo real debido a su mayor tiempo de procesamiento, mientras que umbrales muy altos pueden resultar en perder información valiosa al detectar muy pocas esquinas.

3.3. Descriptores

En este segundo apartado de experimentación vamos a ver diferentes tipos de descriptores y qué conclusiones sacamos de cada uno.

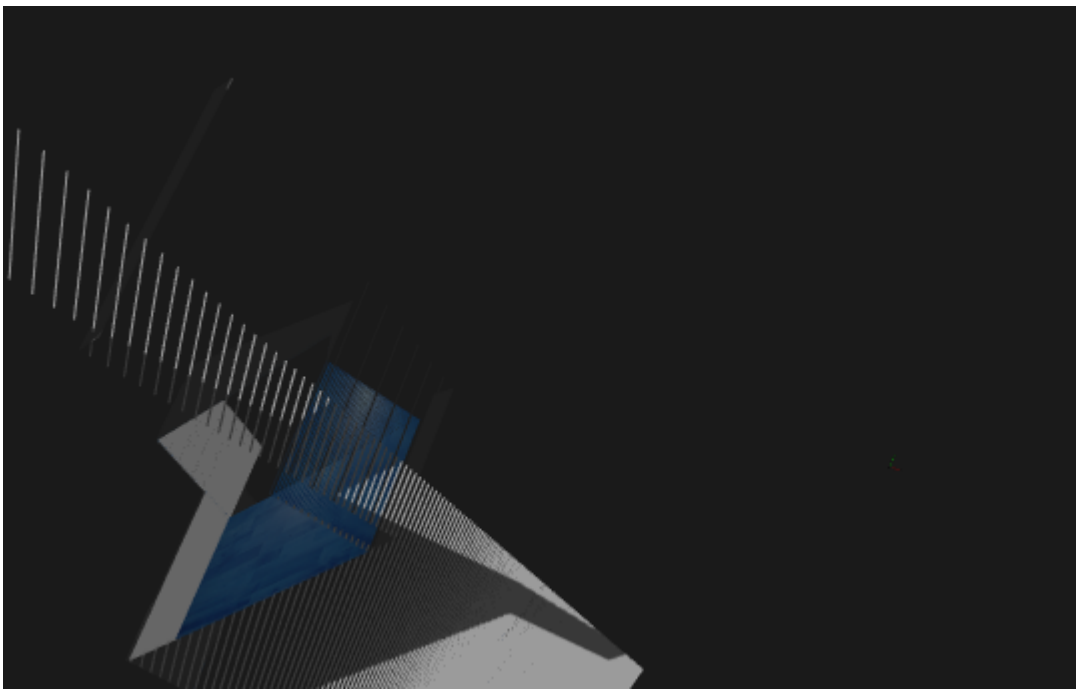
En un primer momento hemos probado el algoritmo **PFH (Point Feature Histograms),** pero debido a el excesivo tiempo que ha tardado en calcular los primeros descriptores (más de una hora de ejecución y no avanzaba), quedó totalmente descartado.

Vamos a empezar con **FPFH (Fast Point Feature Histograms)** surge como mejora del algoritmo PFH el cual es muy costoso a nivel de computo. Su funcionamiento es que emparejamos cada punto de la vecindad con el keypoint, y cada uno de esos puntos con sus vecinos. Las aportaciones de los vecinos de los vecinos están pesadas por su distancia al keypoint. Este algoritmo utiliza 3 histogramas para su uso, además de que necesita previamente las normales de los puntos capturados.

Las nube contiene 88975 puntos tras voxelgrid

RadiusSearch	Tiempo (s)	Puntos
0.5	0.636	7476
0.1	0.096	7476
0.01	0.087	7476

En cuanto a tiempo, vemos que a menos radio de búsqueda, es menor, pero con esto no podemos sacar conclusiones. Cuando vemos la captura en nuestro visualizador, notamos que contra más bajo es el radio, más problemas presenta el modelo, haciendo que RANSAC no pueda realizar bien los cálculos y falle la construcción de la transformación. En resumen, a menor radio de búsqueda, menor precisión en RANSAC, además de que el tiempo de cálculo del mismo se dispara.



Por otro lado, tenemos los descriptores CVFH y SHOT. No hemos podido realizar pruebas con ellos porque fue lo último que implementamos una vez conseguimos que la totalidad de la práctica funcionará (queríamos experimentar con ellos), pero debido a que no logramos que funcionaran de forma correcta, no hemos podido hacer las pruebas oportunas con otros descriptores.

SHOT (Signature of Histograms of Oriented Gradients) es un algoritmo de extracción de características que básicamente funciona dividiendo el espacio alrededor de un keypoint en esferas y calculando histogramas de orientaciones de gradientes locales para cada esfera.

Este algoritmo inicialmente os daba inicialmente un error en el cual el número de puntos de la nube de input no coincidía con las normales. Una vez solucionado ese problema, obtenimos correctamente los descriptores, pero tras 2 iteraciones, el matching nos daba un error donde no aceptaba NaN e Inf, que, a pesar de supuestamente solucionar ese problema eliminando esos puntos que daban conflicto a través de cálculos, no logramos sobrepasar ese error.

En cuanto a **CVFH (Clustered Viewpoint Feature Histogram)**, se trata de un algoritmo que básicamente divide el objeto en diferentes regiones o grupos basados en cómo se ven desde distintos ángulos. Luego, calcula un histograma para cada grupo que describe cómo se ven esas regiones desde esos ángulos específicos. Finalmente, combina todos estos histogramas en un solo vector de características, que es lo que se utiliza para representar el objeto y compararlo con otros objetos.

El problema que hemos tenido con CVFH es que, da igual que parámetros utilizaras, se quedaba calculando infinitamente. Probamos cambiando las entradas que recibía y diferentes tipos de modificaciones, pero finalmente lo único que conseguimos fue que devolviera o 0 o 1, algo que no nos aportaba nada.

3.4. RANSAC

Ahora voy a realizar una serie de pruebas con RANSAC, cambiando los parámetros para buscar una serie de soluciones óptimas y ver que parámetros son cada uno y qué comportamiento cambia. Para la experimentación de RANSAC, hemos guardado la rotación y posición de robot en dos posiciones diferentes y la hemos replicado para tomar dos capturas de nubes.

En primer lugar, debemos realizar el correcto matching de las nubes de puntos. Para ello, utilizamos los descriptores de las nubes de puntos anterior y la actual. Una vez lo tenemos, empezamos con las pruebas de parámetros.

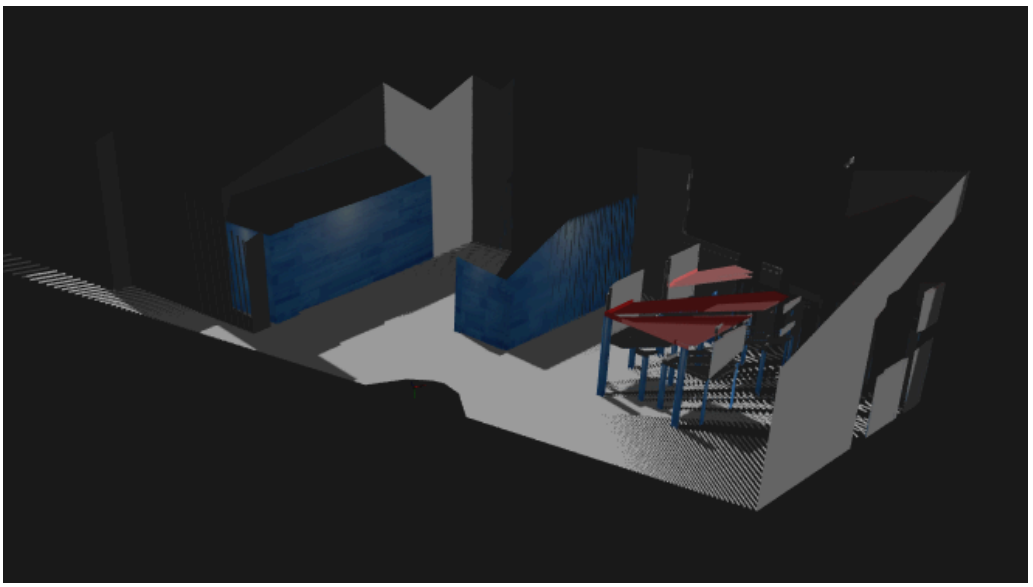
Los parámetros que vamos a variar de RANSAC son:

- **Threshold:** Este parámetro determina la distancia máxima entre un punto y el modelo para que pueda ser considerado como parte del modelo. Es decir, si la distancia entre un punto y el modelo es menor que el valor de threshold, se considera que ese punto es consistente con el modelo. Aumentar este valor puede permitir la inclusión de más puntos en el modelo, mientras que disminuirlo puede hacer que el modelo sea más restrictivo y acepte menos puntos.
- **MaximumIterations:** Establece el número máximo de iteraciones que el algoritmo realizará para encontrar el mejor modelo. Aumentar este valor puede aumentar la probabilidad de encontrar un mejor modelo, pero también aumentará el tiempo que tarda en calcular. En cambio, reducir este valor puede disminuir el tiempo de cálculo, pero también la probabilidad de encontrar el modelo acertado.

El otro valor que vamos a dejar fijo es RefineModel, que básicamente si lo dejamos en true, realizará optimizaciones extra para tener un modelo final más acertado.

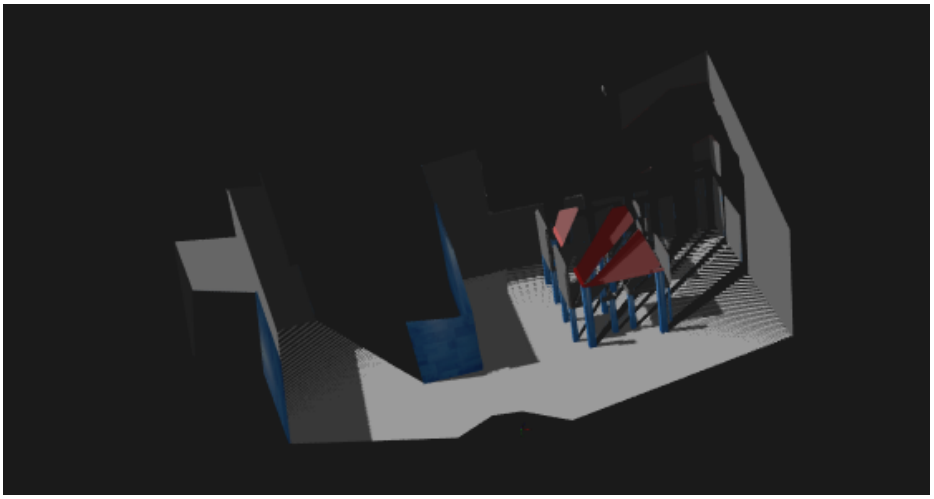
Estas pruebas vamos a ver como posición las nubes de puntos en la transformación. Para eso, mostraré el modelo final. Todas las pruebas están realizadas con el mejor ISS y el descriptor del algoritmo FPFH. La rotación del robot es manual debido al ruido del simulador, además de que aproximadamente las capturas se hacen cada 10 grados.

Mejor solución
Threshold: 0,5 - MaximumIterations: 10000
Tiempo medio: 0,120 s



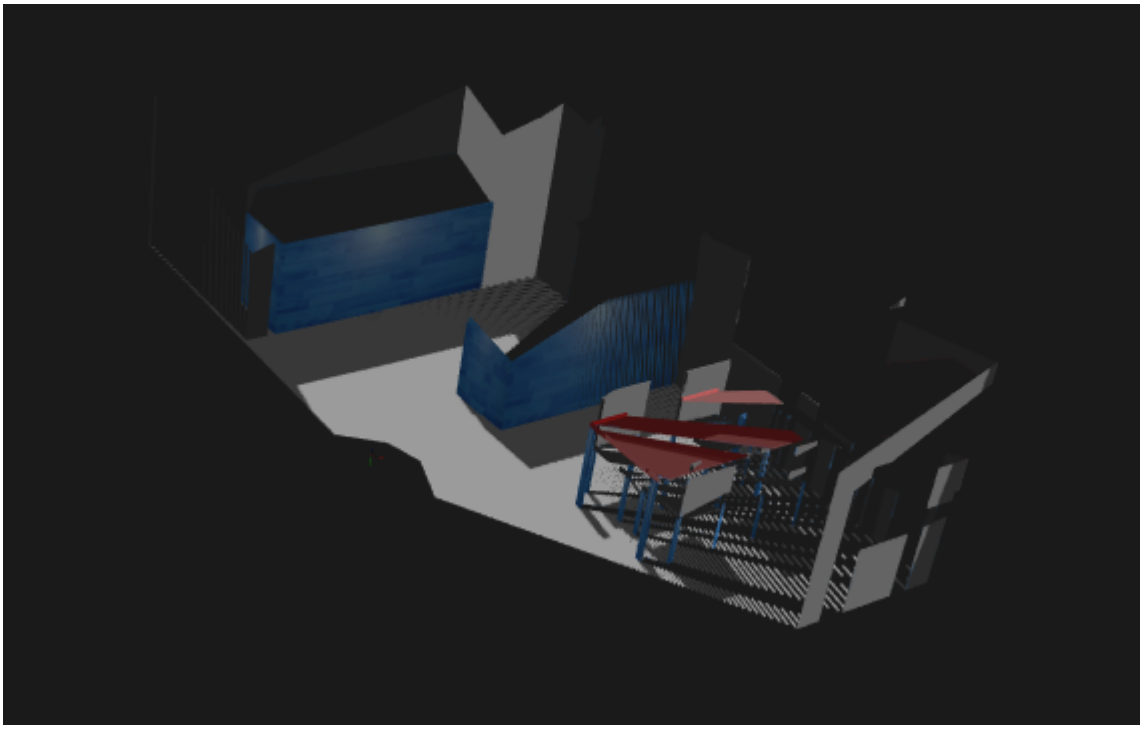
Aquí como vemos, tenemos un resultado bastante acertado, donde cada captura está en el sitio que debe y con la rotación correcta.

Threshold: 0,1 - MaximumIterations: 10000
Tiempo medio: 19,65 s



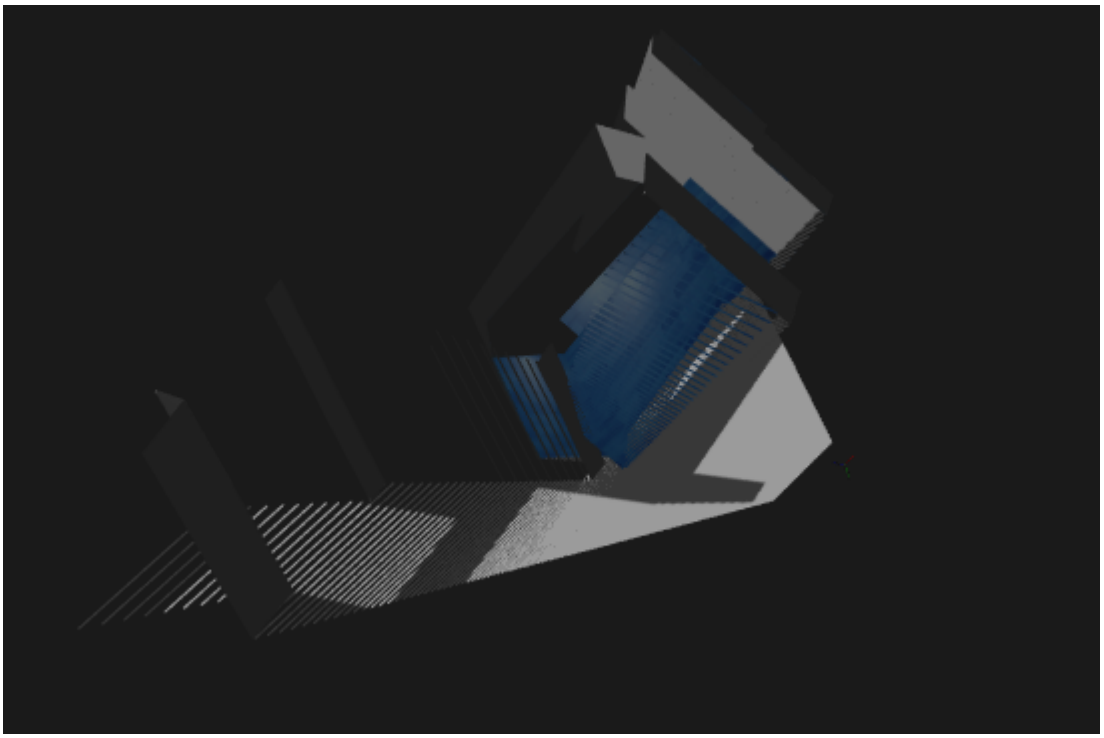
Esta segunda configuración nos vuelve a proporcionar un buen modelo pero, a diferencia del anterior, el tiempo de RANSAC se ha disparado, llegando a una media de tiempo bastante alta.

Threshold: 0,1 - MaximumIterations: 1000
Tiempo medio: 9,34 s



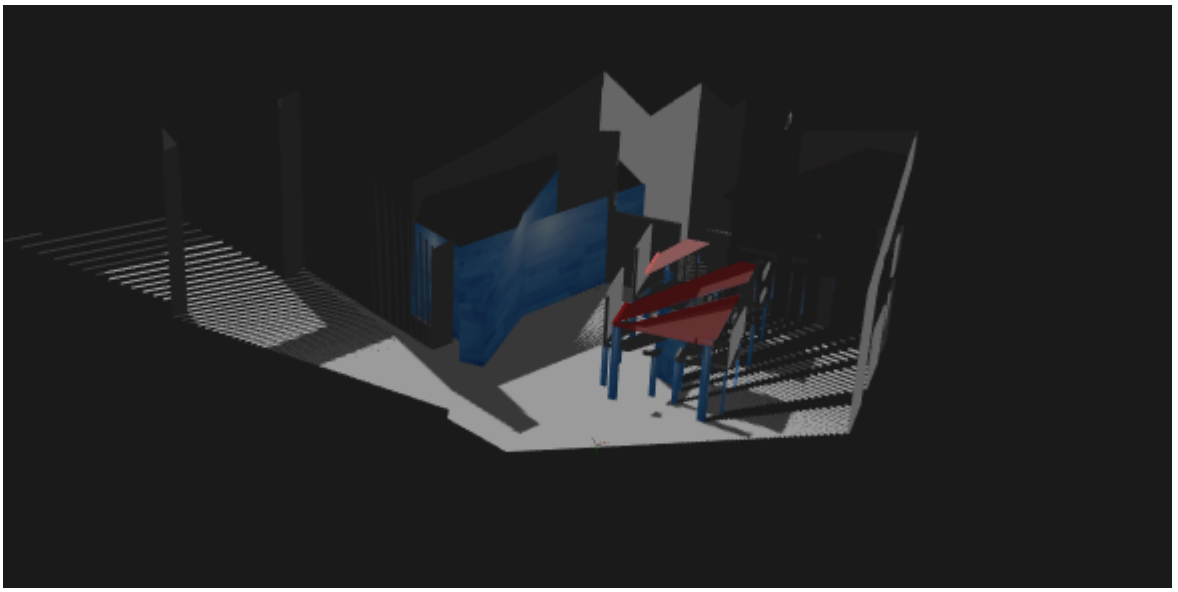
Aquí vemos que, reduciendo las iteraciones máximas que podemos hacer, conseguimos un tiempo más rápido con unos resultados similares, lo hemos reducido en torno a un 50%.

Threshold: 0,5 - MaximumIterations: 20000
Tiempo medio: 4,51 s



Esta configuración nos ha dado fallos. Las capturas no se montan correctamente, dando lugar a un mapeo erróneo.

En esta última prueba simplemente he querido rotar demasiado el robot entre captura y captura para ver cómo funcionaba RANSAC cuando estaba en esta situación. Este es el resultado:



Como vemos, la captura se ha colocado en la rotación correspondiente, pero obviamente no se acerca al modelo que buscamos.

Tras realizar las pruebas correspondientes podemos comprobar ciertas afirmaciones:

RANSAC es sumamente efectivo para alinear nubes de puntos con la configuración adecuada de parámetros. Al ajustar cuidadosamente el umbral de distancia y el número máximo de iteraciones, se puede lograr una alineación precisa que sitúa las capturas en la posición correcta y con la orientación adecuada. El parámetro de umbral es esencial en la determinación de la inclusión de puntos en el modelo. Un umbral mayor tiende a incluir más puntos, lo que puede ser bueno en casos de datos con ruido, pero también puede aumentar la probabilidad de incluir puntos atípicos. Por otro lado, un umbral menor resulta en un modelo más restrictivo y posiblemente más preciso, pero puede excluir datos válidos.

El número de iteraciones de RANSAC afecta directamente el tiempo de cómputo y la calidad del modelo. Un mayor número de iteraciones aumenta las posibilidades de encontrar un modelo óptimo pero con un costo en tiempo de procesamiento. Reducir las iteraciones puede disminuir el tiempo a expensas de la precisión del modelo. Tras una experimentación hemos podido comprobar la configuración óptima encontrada con un umbral de 0.5 y 10000 iteraciones ofreció un equilibrio entre precisión y tiempo de ejecución. Aunque configuraciones con un umbral más bajo también dieron buenos resultados, el tiempo de ejecución se incrementó dejando un balance bastante bueno de velocidad y efectividad.

Configuraciones con un alto número de iteraciones y umbrales inadecuados llevaron a errores en la alineación, indicando que un ajuste cuidadoso es crucial y que más iteraciones no siempre se traducen en mejores resultados. El ruido en el simulador y el movimiento manual del robot introducen variables que pueden afectar la alineación, ese es el motivo por el cual paramos el simulador y rotamos manualmente.

3.5. ICP

ICP se encarga de refinar el modelo, acercando las nubes de puntos lo más posible para obtener la mejor transformación. Para estas pruebas, vamos a variar 4 parámetros:

- **MaxCorrespondenceDistance:** Este parámetro define la distancia máxima a la que se buscarán los puntos correspondientes entre las dos nubes de puntos. Los puntos que están más allá de esta distancia no se considerarán como correspondencias potenciales.
- **MaximumIterations:** Marca el número máximo de iteraciones que puede hacer el algoritmo para converger.
- **TransformationEpsilon:** Controla cuándo se considera que el algoritmo ha convergido a una solución aceptable. Aumentar el valor de este parámetro puede permitir una convergencia más rápida si se está satisfecho con una solución aproximada, mientras que disminuirlo puede aumentar la precisión pero también el tiempo de cómputo.
- **EuclideanFitnessEpsilon**: Determina cuándo el algoritmo ICP ha alcanzado una solución que se considera lo suficientemente precisa. Aumentar este valor permitirá un ajuste menos estricto y, por lo tanto, puede hacer que el algoritmo ICP se detenga antes, mientras que disminuirlo hará que el algoritmo sea más exigente en términos de precisión.

Vamos a hacer las pruebas, que controlaremos el tiempo y el FitnessScore, una métrica que nos dice que, a mayor valor, mayor distancia existe entre las nubes de puntos y por tanto tenemos una mala alineación. Por el contrario, si tenemos un valor bajo, significa que las nubes se han alineado bien.

Tras unir dos nubes de puntos en la misma rotación y posición

MaxCorrespondenceDistance	Maximum Iterations	Transformation Epsilon	EuclideanFitness Epsilon	Tiempo (s)	Fitness Score
0.05	50	1e-8	1e-6	0.3598	0.1897
0.1	50	1e-8	1e-6	30.1887	0.5969
0.001	50	1e-8	1e-6	0.4557	0.4557
0.05	500	1e-8	1e-6	42.903	0.0504
0.05	10	1e-8	1e-6	5.868	0.0902
0.05	50	1e-12	1e-6	50.6498	0.6062
0.05	50	1e-4	1e-6	5.0786	0.5203

0.05	50	1e-8	1e-12	32.883	0.1221
0.05	50	1e-8	1e-4	ERROR	ERROR

Basándonos en los resultados presentados y considerando el impacto de los parámetros de ICP en el proceso de alineación de nubes de puntos, las conclusiones serían las siguientes:

Un valor bajo de MaxCorrespondenceDistance(0.05) generalmente resulta en una alineación más precisa, como se evidencia por los menores Fitness Scores. Valores altos (0.1) permiten que puntos más lejanos sean considerados como correspondencias, lo cual puede resultar en una alineación menos precisa y un Fitness Score más alto. Aumentar las iteraciones (hasta 500) mejora la precisión del alineamiento hasta cierto punto, lo cual se refleja en un Fitness Score bajo (0.0504), sin embargo, un número excesivo de iteraciones no garantiza siempre mejores resultados y puede aumentar significativamente el tiempo de cómputo (42.903 segundos). Un valor muy bajo de TransformarionEpsilon (1e-12) no necesariamente mejora la alineación y puede conducir a errores donde el modelo no puede ser refinado, posiblemente debido a la convergencia prematura del algoritmo en una solución local subóptima mientras que un valor más alto (1e-4) no mejora la precisión, como se observa en los Fitness Scores comparativamente altos y puede llevar a errores en casos donde el umbral de parada es demasiado tolerante. Un valor muy bajo de EuclideanFitnessEpsilon (1e-12) parece ser demasiado restrictivo, resultando en tiempos de procesamiento largos sin necesariamente mejorar la precisión, pero un valor demasiado alto llevó a un error, lo que sugiere que el algoritmo detuvo prematuramente la iteración antes de alcanzar una alineación adecuada.

El mensaje de error "[pcl::registration::CorrespondenceRejectorSampleConsensus::getRemainingCorrespondences] Could not refine the model! Returning an empty solution" indica que el proceso de RANSAC interno en ICP no pudo refinar el modelo, probablemente debido a un ajuste inadecuado de los parámetros, llevando a una falta de correspondencias válidas. Existe un equilibrio entre el tiempo de cómputo y la precisión de la alineación. Encontrar el conjunto óptimo de parámetros que ofrece un compromiso entre estos dos aspectos es crucial para la eficiencia del proceso de ICP. La mejor configuración de parámetros parece ser con MaxCorrespondenceDistance en 0.05, MaximumIterations alrededor de 50 a 100, TransformationEpsilon en 1e-8, y EuclideanFitnessEpsilon en 1e-6. Esta configuración proporciona un buen equilibrio entre tiempo de procesamiento y precisión de alineación, logrando scores de fitness bajos que indican una alineación precisa en tiempos de cómputo razonables.

4. Conclusiones

Como conclusión general, hemos logrado establecer un sistema que finalmente nos ha permitido realizar un programa para un robot equipado con una cámara Creative, recopilar datos en un entorno simulado y generar un mapeo 3D detallado.

El pipeline de desarrollo utilizado ha sido eficaz para procesar nubes de puntos y generar el mapeo 3D mayoritariamente preciso (en algunos casos puntuales puede cometer errores,pero por lo general los resultados son buenos a pesar de el tiempo de procesamiento).

Las pruebas con diferentes algoritmos de detección de keypoints y generación de descriptores, como son ISS o SIFT, ha dejado en claro la gran importancia de una correcta selección de estos elementos para mejorar la precisión del mapeado, además de la importancia de los ajustes de los parámetros de estos algoritmos, los cuales han sido esenciales para optimizar el programa, afectando tanto en la cantidad de keypoints detectados como en tiempo de procesamiento, buscando una media de precisión y eficiencia. En algunos de estos algoritmos, cuando realizamos algún cambio mínimo, suponía un gran cambio en los resultados finales.

Como errores cometidos/complicaciones a lo largo de la práctica, recalcar unos cuantos de ellos:

- Al inicio de la práctica, solo capturaba 1 o 0 keypoints. Esto se debía a que necesitaba, al menos en el algoritmo de ISS, la resolución de la nube que obtenemos a partir de una función.

- A la hora de calcular las correspondencias, en un principio utilizabamos las nubes de keypoints, recibiendo unos cuantos errores y quebraderos de cabeza ya que el modelo de RANSAC no funcionaba correctamente. Más tarde, nos dimos cuenta de que el error residía en que debíamos utilizar los descriptores para el matching y estabamos utilizando otros datos.

- RANSAC funcionó tras muchísimas horas de investigación. El problema no residía en los parámetros o en los algoritmos anteriores, si no que en lugar de multiplicar una transformación local por una global (para ir acumulandola), la sumamos, por lo cual las nubes de puntos se superponían en lugar de poner la rotación y posición correcta. Finalmente añadimos la transformación global y el problema se soluciono.

- ICP nos mostraba un error de Relax your parameters!. En primer lugar cometimos el trivial error de pensar que hacer el threshold más pequeño era relajar los parámetros. Pero el problema no residía ahí, sino que estábamos pasando una nube vacía al algoritmo, de forma que no era capaz de encontrar otra nube con la que trabajar.

A pesar del correcto funcionamiento del mapeado para los criterios de esta práctica, se podría implementar alguna mejora para poder mapear con el movimiento que hemos

implementado por terminal, eliminando el ruido del movimiento y haciéndolo en tiempo real. Es más, contamos con un nodo para el movimiento del robot, pero como el ruido del mismo simulador alteraba el movimiento, no hemos decidido usarlo. Lo ideal sería que fuera todo automático.

5. Referencias

1. ROS - Robot Operating System. Recuperado de https://www.ros.org/.
2. Documentación PCL. Recuperado de https://pointclouds.org/documentation
3. ROS Wiki. Recuperado de http://wiki.ros.org/.
4. The Iterative Closest Point (ICP) Algorithm. Recuperado de https://eng.libretexts.org/Bookshelves/Mechanical_Engineering/Introduction_to_Autonomous_Robots_(Correll)/12%3A__RGB-D_SLAM/12.02%3A_The_Iterative_Closest_Point_(ICP)_Algorithm.
5. RANSAC. Recuperado de https://docs.mrpt.org/reference/latest/tutorial-ransac.html.
6. Repositorio Github 1. Recuperado de https://github.com/mycodeself/VAR/tree/master/p2_ws
7. Repositorio Github 2. Recuperado de https://github.com/jgm139/var1718P2/tree/master/catkin_ws/src/get_pointclouds/src