Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis y Evaluación de Algoritmos de Procesamiento de Datos LIDAR en un Robot Móvil

Autor: Roberto Alaminos Beneitez
Tutor(a): Santiago Tapia Fernández

Madrid, Julio 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Análisis y Evaluación de Algoritmos de Procesamiento de Datos LIDAR en un Robot Móvil

Julio 2025

Autor: Roberto Alaminos Beneitez
Tutor: Santiago Tapia Fernández
Dep. Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este proyecto se enmarca dentro del grupo de investigación CVAR (Computer Vision and Aerial Robotics) de la ETSIINF-UPM, y tiene como objetivo principal el desarrollo de un sistema capaz de detectar objetos con las dimensiones de una caja en específico en entornos tridimensionales mediante el uso de un sensor LiDAR con la idea de montarlo en un futuro sobre un dron terrestre. La finalidad es explorar soluciones eficientes y ligeras para tareas de percepción robótica, sin recurrir necesariamente a modelos de inteligencia artificial.

En las próximas páginas se va a dar información de distintas cosas que envuelven como se hizo el proyecto, el aprendizaje etc..

- Primero se introduce en el contexto del proyecto, cuales son los objetivos y que tecnologías o lenguajes se utilizarán para conseguir el mismo
- En el capítulo de desarrollo primero se identifica el alcance de la solución es decir los requisitos funcionales y no funcionales. Además se exponen las distintas fases para la ejecución del proyecto:
  - Diseño de la solución
  - Aprendizaje
  - Distintas fases del desarrollo
- En el siguiente capítulo se exponen los resultados obtenidos con sus ejemplos, pruebas, explicaciones, problemas y futuras mejoras
- Para terminar se hace un resumen completo de los logros, problemas, aplicaciones y lineas de mejora.

Durante el desarrollo se definen los objetivos funcionales y no funcionales para poder hacer un diseño de la solución. Aparte de eso se hace una evolución continua e incremental en el código avanzando de problema en problema resolviéndolos secuencialmente. Previo a esto hubo que familiarizarse con las herramientas que se iban a necesitar en la evolución y las distintas tecnologías necesarias para el proyecto. Sin olvidarse del proceso de tomar grabaciones que permitieron poder afrontar el problema en una secuencia de prueba y error en la fase de etapas, y para la futura validación del código.

En los resultados se confirmó la robustez y validez del código implementado, además de descubrir los fallos en los datos que han de ser solucionados en un futuro.

Abstract

<<Abstract of the Final Degree Project. Maximum length: 2 pages.>>

This project is part of the CVAR (Computer Vision and Aerial Robotics) research group at ETSIINF-UPM, and its main objective is the development of a system capable of detecting objects with the dimensions of a specific box in three-dimensional environments using a LiDAR sensor, with the idea of eventually mounting it on a ground drone. The aim is to explore efficient and lightweight solutions for robotic perception tasks, without necessarily resorting to artificial intelligence models.

In the following pages, information will be provided on various aspects such as how the project was carried out, the learning process, etc.

• First, the project context is introduced, along with its objectives and the technologies or programming languages that will be used to achieve it.

• In the development chapter, the scope of the solution is first identified, that is, the functional and non-functional requirements. In addition, the different phases for the execution of the project are presented:

– Solution design

– Learning process

– Different development stages

• The next chapter presents the results obtained, including examples, tests, explanations, problems, and future improvements.

• Finally, a complete summary is provided covering achievements, issues, applications, and improvement lines.

During development, the functional and non-functional objectives were defined in order to design the solution. In addition, the code was developed through a continuous and incremental evolution, progressing from one problem to another and solving them sequentially. Prior to this, it was necessary to become familiar with the tools and technologies required throughout the project's evolution. It is also important to mention the process of recording data, which made it possible to approach the problem through a trial-and-error sequence during the development stages, and which will serve for future code validation.

The results confirmed the robustness and validity of the implemented code, while also revealing data issues that will need to be addressed in the future.

End

Agradecimientos

En primer lugar, mi más sincero agradecimiento al lector por dedicar su tiempo leyendo este documento muestra de mi trabajo. Sin tu lectura todo este esfuerzo invertido en esta investigación sería un sin sentido, y espero que el lo que lea a a continuación sea de su agrado y al menos le resulte una pizca de interesante.

También doy las gracias a mi tutor, Santiago Tapia Fernández, por no ignorarme cuando sin previo aviso mandaba un mensaje sin ningún tipo de contexto y por su paciencia intentando ayudarme y resolver esas dudas. Gracias a él y su guía este proyecto ha sido poco a poco redirigido y no me he perdido en las pequeñas cosas.

No puedo dejar de mencionar a mi familia y a mis amigos, quienes han estado a mi lado en los momentos más tensos del trabajo de fin de grado. Gracias por soportarme a mí en todos estos momentos cómo cuando creí que había perdido todo el TFG o cuando simplemente no sabía como avanzar. Pero aunque fuese el peor día ellos estaban ahí. Esto se ha completado y enviado en parte gracias a ellos ya que si no en algún momento la rendición hubiese podido conmigo.

Gracias.

Índice general

1 Introducción

1.1. Contexto

Este trabajo se ha llevado a cabo en colaboración con el grupo de investigación CVAR (Computer Vision and Aerial Robotics), vinculado a la Escuela Técnica Superior de Ingenieros Informáticos de la Universidad Politécnica de Madrid. El grupo se dedica principalmente a explorar cómo aplicar técnicas de visión por computador en entornos aéreos, utilizando drones equipados con sensores avanzados.

En este caso, se ha decidido trabajar con un sensor LiDAR, el cual nos permite obtener información 3D de lo que rodea al sensor mediante la medición del tiempo que tardan los distintos pulsos laser en reflejarse en los distintos objetos. El objetivo final del proyecto es acercarnos un poco más a un dron que pueda hacer tareas automatizadas pudiendo detectar distintos objetos a sus alrededores.

Dado que dos estudiantes han trabajado en el mismo ámbito, se decidió dividir el proyecto en dos partes complementarias:

- El primer estudiante se encargó del desarrollo de un sistema capaz de reconocer y diferenciar huecos en paredes.
- El segundo (autor de este proyecto) se centró en el diseño de un sistema que fuese capaz de detectar un objeto rectangular(caja) dentro de las mediciones de puntos del LiDAR, diferenciándolo de otros cúmulos que aparecen en las mediciones.

1.2. Objetivos

El propósito principal de este proyecto es desarrollar un sistema capaz de procesar las lecturas del sensor LiDAR para identificar y localizar un objeto poliédrico en un entorno tridimensional. Para iniciar el trabajo se ha optado por usar una caja porque es una figura sencilla y se debería poder obtener buenos resultados. Para ello, se plantean los siguientes objetivos específicos:

- Procesar las nubes de puntos generadas por el sensor para reconstruir el entorno en 3D.
- Identificar y separar los distintos cúmulos de puntos para analizarlos de forma individual.
- Aplicar una serie de restriciones de altura o longitud para determinar que cumulo de puntos las cumple por lo que puede ser el objeto buscado.

- Disponer de un proceso eficiente y ligero, ya que los drones o robots cuentan con potencia computacional limitada.

- Evaluar el uso de lenguajes de programación más eficientes en determinadas fases del procesamiento, evitando el uso de Python cuando no sea necesario, especialmente si se requiere un rendimiento más alto.

En resumen, el sistema debe ser preciso, eficiente y adaptable a las limitaciones hardware que se puedan montar en un dron.

1.3. Tecnologías

Debido a las diversas necesidades del proyecto como procesar puntos en coordenadas tridimensionales, agruparlos en clusters y encontrar la forma (es este caso rectas por la naturaleza del objeto a buscar), así como enviar y recibir datos, se usaron las siguientes herramientas principales:

1.3.1. Sensor LiDAR

Lidar (Light Detection and Ranging): Un sensor que mediante la emisión de pulsos láser en distintas dimensiones lo que proporciona información del entorno 3D a su alrededor. Su funcionamiento es sencillo ya que se basa en:

- Mandar el pulso láser

  - Medir el tiempo que tarda este mismo laser en regresar.

  El láser enviado cuando choca con la superficie de algún objeto rebota siendo precisamente este el motivo de que vuelva. Conociendo la velocidad de la luz y el tiempo medido (ida y vuelta) se calcula la distancia al objeto.

  - Estructura y escaneo:

La Figura 1.1 ilustra el funcionamiento básico de un sensor LiDAR:

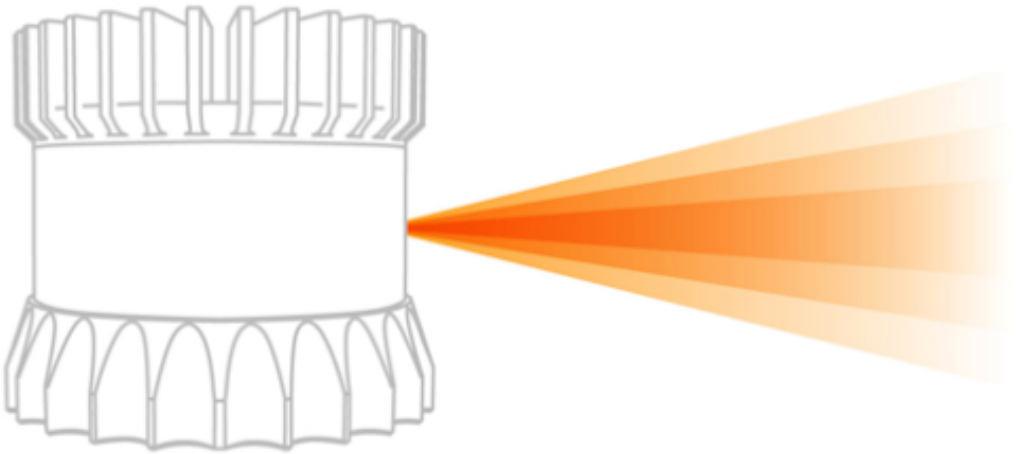

Figura 1.1: Funcionamiento LiDAR

En esta imagen se observa cómo el sensor, de forma cilíndrica, emite haces de luz en un patrón radial. Este patrón se genera gracias a dos movimientos combinados:

- Rotación horizontal (azimutal): el sensor gira 360° sobre su eje vertical (en 1024 planos), cubriendo todo el entorno en el plano horizontal.
  - Barrido vertical (elevación): el sensor dispone de múltiples emisores o planos de disparo (en este caso, 32), distribuidos uniformemente entre −11.25° y +11.25°, lo que permite capturar información en altura.

    El resultado es una matriz de datos de 1024 × 32:

    - Donde en cada una de las 1024 columnas están todos los sensores que comparten un ángulo horizontal o azimutal entre sí.
    - Mientras que dentro de esa columna hay 32 mediciones correspondiente a cada sensor en un ángulo vertical distinto pero igual que en otra columna.
  - Conversión a nube de puntos 3D
    - Para todos los puntos de la matriz hay que adquirir sus coordenadas x,y,z para ello es necesario usar estas funciones que calculan

cada una de ellas:

$$x = r\cos(\theta)\cos(\phi)$$
$$y = r\cos(\theta)\sin(\phi)$$
$$z = r\sin(\theta)$$

donde:

- ◇ r es la distancia medida por el sensor (en metros),
- ◇ θ es el ángulo de elevación (entre −11.25° y +11.25°),
- ◇ ϕ es el ángulo azimutal (entre 0° y 360°).

Esta transformación permite reconstruir el entorno en forma de una nube de puntos tridimensional, que puede ser procesada posteriormente para tareas como segmentación, detección de objetos, navegación autónoma, entre otras.

- Ventajas del LiDAR
  - ○ Gran distancia de medición en todos los ángulos.
  - ○ Independencia de las condiciones de iluminación (funciona en completa oscuridad).
  - ○ Capacidad para generar modelos 3D detallados del entorno.

1.3.2. ROS2

ROS 2 (Robot Operating System 2): Es el middleware utilizado para la comunicación entre los distintos componentes del sistema. ROS 2 permite la suscripción y publicación de mensajes a través de topics, que son canales de comunicación asíncronos. En este proyecto se han utilizado:

- Publicadores (publisher):
  - sensor_scan_pointCloud: publica la nube de puntos generada a partir de una matriz de distancias e intensidades del sensor.
  - matriz: topic adicional para enviar la misma nube de puntos a otros nodos o herramientas como RViz(interfaz grafica de ROS) para la visualización.
- Suscriptor (subscriber):
  - Se suscribe a un topic que recibe mensajes del tipo sensor_msgs/PointCloud2, procesando la nube de puntos para filtrar, agrupar y detectar estructuras geométricas.

Esta arquitectura basada en topics facilita la modularidad, escalabilidad y reutilización del sistema en otros proyectos robóticos.

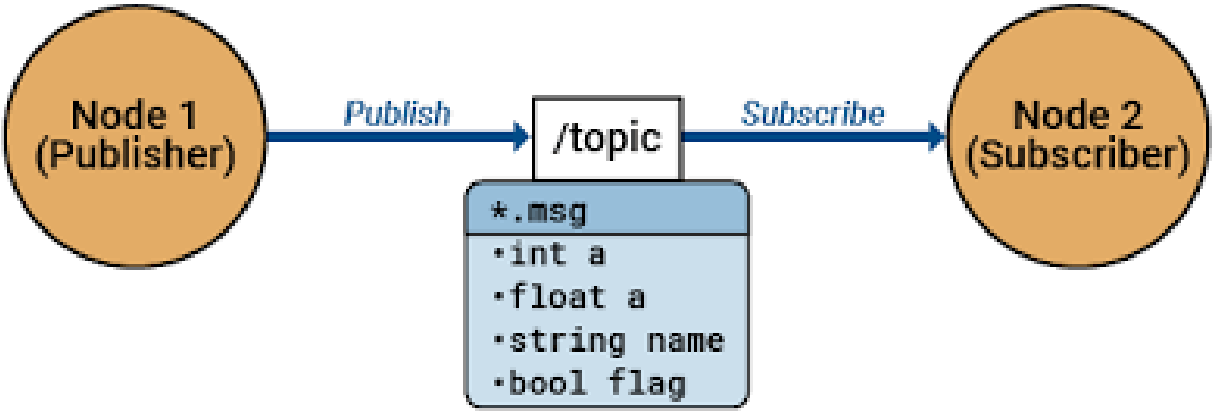

Figura 1.2: Esquema mensajería ROS 2

1.3.3. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):

- Herramienta que usando un algoritmo puede encontrar cúmulos de puntos o clusters dentro de todos los puntos que tenemos para la imagen a generar.
- Se utiliza para separar regiones significativas del entorno (como objetos o estructuras) del ruido(información no deseada o erronea) o puntos aislados.

1.3.4. RANSAC

RANSAC (Random Sample Consensus):

- Algoritmo robusto para el ajuste de modelos geométricos en presencia de ruido.
- En este caso, se utiliza para detectar líneas dentro de los cúmulos de puntos identificados por DBSCAN, permitiendo así reconocer los bordes del objeto rectangular buscado.

1.3.5. OpenCV

OpenCV:

- Biblioteca utilizada para la generación de imágenes a partir de los datos procesados.
- Permite representar visualmente los puntos filtrados y las líneas detectadas, facilitando la interpretación de los resultados.

1.3.6. Wireshark

Wireshark es una herramienta de análisis de red la cual sirve para:

- Visualizar paquetes de datos que circulan por una red y capturar creando grabaciones para un uso posterior.
- Diagnosticar problemas de comunicación entre nodos ROS o dispositivos conectados.
- Verificar si los datos del LiDAR o de otros sensores están llegando correctamente.

1.3.7. Ubuntu 22.04

Ubuntu 22.04: Es el sistema operativo sobre el que hemos estado trabajando. Es una distribución de Linux muy estable y necesaria por su compatibilidad con las herramientas:

- ROS 2 Humble la versión utilizada.
- Herramientas de desarrollo como RViz.

1.3.8. Python y bibliotecas científicas

Python y bibliotecas científicas: El procesamiento se ha implementado en Python, utilizando bibliotecas como:

- numpy para operaciones numéricas.
- scikit-learn para la implementación de DBSCAN.
- scipy para el cálculo de distancias entre puntos.
- cv2 para la visualización.

Se ha priorizado el uso de código eficiente y ligero, considerando que el sistema podría ejecutarse en hardware embarcado con recursos limitados.

2 Desarrollo

2.1. Análisis de Requisitos y Diseño

2.1.1. Requisitos Funcionales

Los requisitos funcionales son todas aquellas funciones, acciones, procesos y comportamientos que un sistema o producto debe cumplir para satisfacer las necesidades del usuario. Estos requisitos funcionales deberán ser desarrollados de forma técnica en el sistema o producto.

En este caso, como se requiere desarrollar un sistema que procese datos LiDAR para poder detectar objetos con la estructura de una caja, los requisitos funcionales son:

- **RF1:** Recepción de datos LiDAR
  - El sistema debe suscribirse a un topic de ROS 2 para recibir datos en formato PointCloud2.
- **RF2:** Filtrado de puntos por posición y altura
  - El sistema debe aplicar un filtro espacial para eliminar puntos fuera de una zona de interés (por ejemplo, por debajo de cierto umbral de altura o fuera de un rango de coordenadas).
- **RF3:** Agrupamiento de puntos
  - El sistema debe identificar cúmulos de puntos densos utilizando un algoritmo. En este caso DBSCAN.
- **RF4:** Detección de líneas
  - El sistema debe aplicar un algoritmo sobre los cúmulos detectados para encontrar líneas que representen bordes de objetos. En este caso RANSAC.
- **RF5:** Clasificación de la validez de la altura de los puntos de la recta
  - El sistema debe enviar los puntos que se ajustan a la recta a un módulo externo que evalúe si la recta detectada corresponde a puntos por encima de la altura posible de la caja o inferior a la altura necesaria de la caja. En este caso C++.
- **RF6:** Visualización de resultados

- El sistema debe generar una imagen con los puntos y líneas detectadas, y publicarla en un topic para su visualización en RViz o herramientas similares.

- **RF7:** Publicación de la nube de puntos procesada
  - El sistema debe publicar la nube de puntos original en un topic para su análisis y comparación de un usuario con la realidad.

2.1.2. Requisitos No Funcionales

Estos requisitos definen cómo debe comportarse el sistema, sin describir funciones específicas:

- **RNF1:** Eficiencia
  - El sistema debe procesar los datos en tiempo real o con baja latencia, adecuado para el procesado en vivo.
- **RNF2:** Compatibilidad con ROS 2
  - El sistema debe integrarse completamente en el ecosistema ROS 2, utilizando sus estándares de comunicación.
- **RNF3:** Modularidad
  - El sistema debe estar diseñado de forma modular, permitiendo la sustitución o mejora de componentes individuales sin afectar al resto del sistema.

2.1.3. Diseño

El diseño del sistema se basa en una arquitectura modular utilizando ROS 2, donde los distintos componentes se comunican a través de topics. Esta estructura permite una separación clara de responsabilidades, facilita la escalabilidad y mejora el mantenimiento del sistema.

La siguiente figura, generada con la herramienta rqt_graph, muestra la estructura de nodos y topics activos durante la ejecución del sistema:

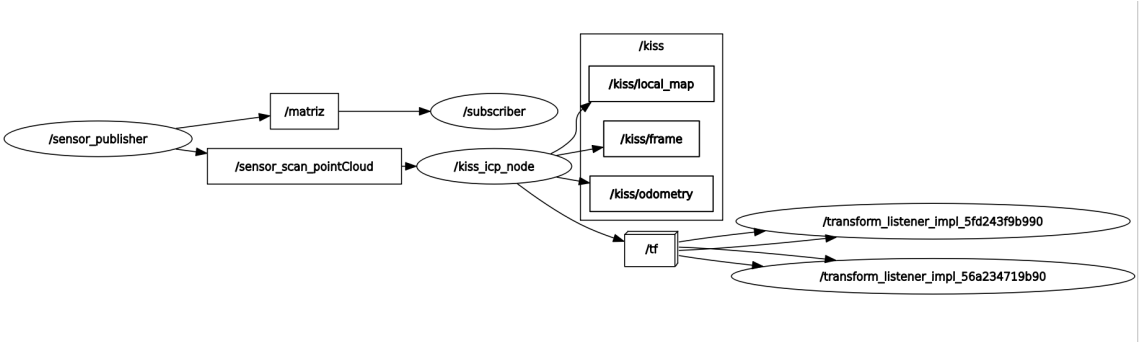

Figura 2.1: Grafo de nodos y topics de rqt_graph

Descripción del grafo

1. /sensor_publisher: Nodo que publica los datos del sensor LiDAR en dos topics:
   - /matriz: topic adicional que puede ser utilizado para depuración o visualización.
   - /sensor_scan_pointCloud: contiene la nube de puntos generada a partir de la matriz de distancias.
2. /subscriber: Nodo que recibe los datos de los topics anteriores y realiza el procesamiento:
   - Filtrado de puntos.
   - Agrupamiento con DBSCAN.
   - Detección de líneas con RANSAC.
   - Clasificación mediante un módulo en C++.
3. /kiss_icp_node: Nodo adicional para la visualización humana de los resultados en formato nube de puntos que se conecta al topic /sensor_scan_pointCloud y publica en:
   - /kiss/local_map
   - /kiss/frame
   - /kiss/odometry
4. /tf: para la gestión de transformaciones espaciales entre marcos de referencia.

Este diseño permite una separación clara de responsabilidades entre nodos, facilita la escalabilidad del sistema y permite la integración con herramientas de visualización como RViz.

Aparte aquí añado un pequeño esquema que puede ayudar al entendimiento de la situación del código final y como funciona. Dado que se explicará más adelante como fue evolucionando este.

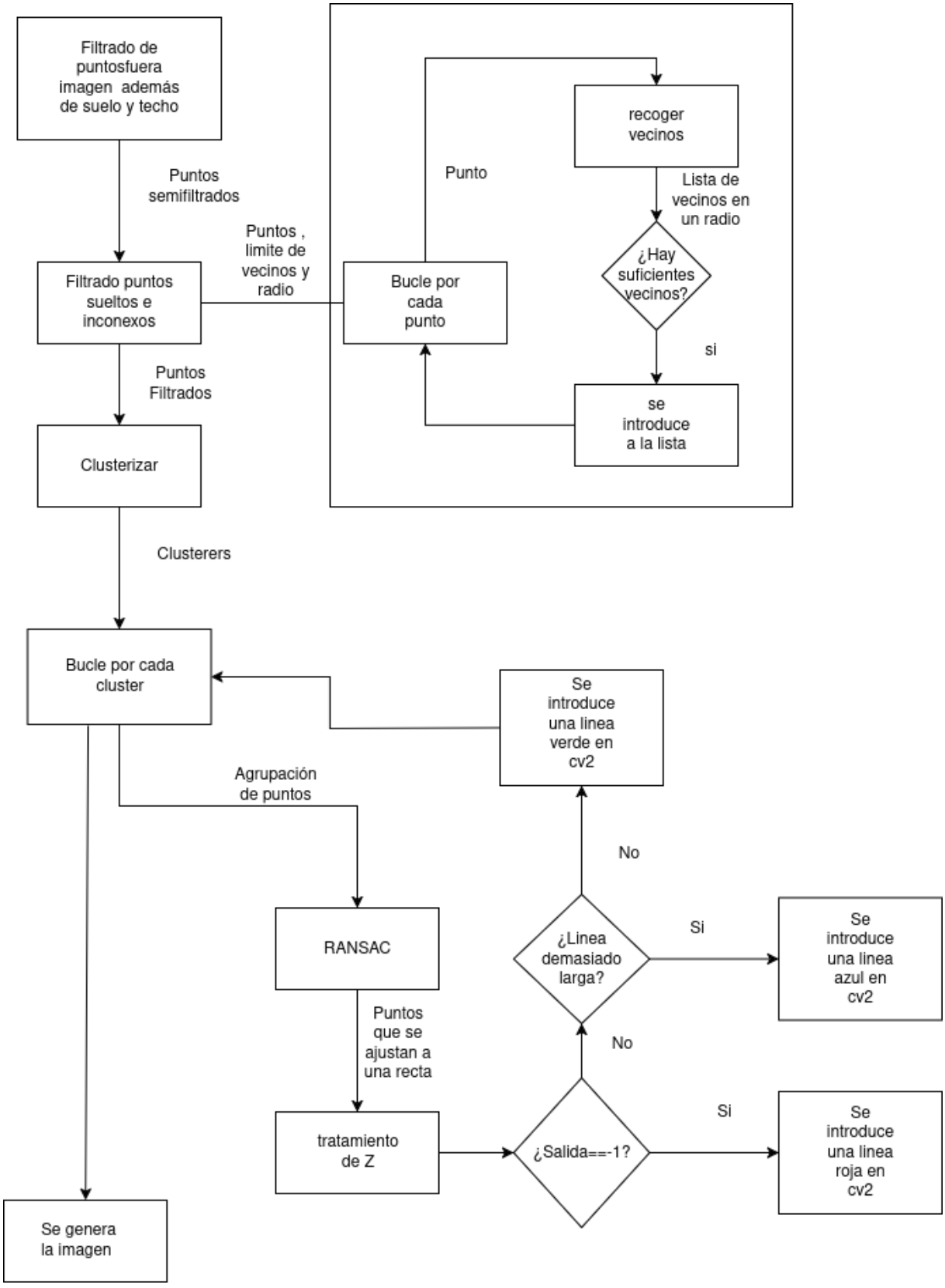

Figura 2.2: Diagrama explicatorio del código

2.2. Evolución

El desarrollo del TFG se ha realizado en diferentes etapas evolucionándolo de forma incremental. Primero se realizó un aprendizaje de las diferentes herramientas y algoritmos para su futuro uso en este proyecto. Posteriormente se fue desarrollando por etapas encontrando distintos problemas que tuvieron que ser solucionados con la implementación de desarrollos adicionales. (filtrado, agrupación, ...)

2.2.1. Aprendizaje

Debido a que muchas de las tecnologías utilizadas eran nuevas, primero se realizaron una serie de tareas prácticas para familiarizarse con ellas de manera progresiva y manejable. Estas actividades iniciales permitieron adquirir una base sólida antes de abordar el desarrollo principal.

Herramientas:

1. Wireshark para crear grabaciones :
    - La herramienta permitió la capturay guardado de los datos proporcionados por el sensor LiDAR.
2. Ubuntu 22.04: Es el sistema operativo sobre el que hemos estado trabajando. Es una distribución de Linux muy estable y necesaria por su compatibilidad con las distintas herramientas utilizadas.
3. ROS 2 Humble es un framework para el desarrollo de software robótico. En este caso, sirve para:
    - Gestionar la comunicación entre nodos (por ejemplo, entre el LiDAR y el procesador).
    - Publicar y suscribirse a datos como nubes de puntos, imágenes, transformaciones, etc.
    - Modularizar el sistema de forma escalable y reutilizable.
4. LiDAR Ouster OS2 es el sensor que se ha utilizado. Sirve para:
    - Capturar el entorno en 3D mediante pulsos láser.
    - Generar una nube de puntos que representa objetos, paredes, huecos, etc.
    - Proporcionar datos de alta precisión para tareas como navegación o detección de obstáculos.
5. Cloud Point (Nube de puntos) es el conjunto de puntos 3D que genera el LiDAR. Sirve para:
    - Representar el entorno físico en coordenadas espaciales.
    - Ser procesada por algoritmos como DBSCAN, RANSAC, etc.

- Visualizarse en herramientas como RViz o en imágenes generadas por el sistema.

6. Ouster LiDAR SDK es un Software Development Kit que permite:

   - Conectarse al sensor Ouster a través de Ethernet.
   - Recibir datos en tiempo real: tanto la nube de puntos como datos de intensidad, reflectividad, etc.
   - Configurar el sensor: ajustar parámetros como la frecuencia de escaneo, resolución, modo de operación, etc.
   - Convertir los datos brutos en formatos útiles como PointCloud2 para ROS o matrices NumPy para Python.
   - Visualizar los datos con herramientas incluidas o integrarlas en RViz.

Fases del aprendizaje:

- Instalación y configuración del entorno mediante descarga e instalación de todas las herramientas necesarias. En este proceso se encontraron algunas dificultades, ya que Ubuntu requiere configuraciones específicas y no siempre ofrece una experiencia intuitiva y fácil para el usuario. Fue necesario ajustar dependencias y permisos para que todo funcionase de una manera correcta.

- Una vez configurado el entorno la siguiente fase se desarrolló entorno a la grabación de datos con Wireshark y el SDK de Ouster, en la cual aprendimos a utilizar el SDK de Ouster para visualizar los datos del sensor y a usar Wireshark para capturar grabaciones de las transmisiones. Estas grabaciones serían fundamentales para pruebas posteriores sin necesidad de tener el sensor conectado en todo momento. En esta etapa hemos contado con el apoyo de estudiantes de máster del grupo CVAR, quienes nos guiaron en el inicio por la rareza de estas herramientas.

- Como ejercicio práctico, esta fase se centró en la visualización inicial de datos, se generó una imagen de 1024×1024 píxeles a partir de una grabación sin filtrar, con el objetivo de observar cómo se representan los puntos en un plano 2D. Esto nos permitió identificar patrones como el suelo y el techo, que aparecen como círculos concéntricos.

- La fase final se aprendió y entendió el funcionamiento de la comunicación entre nodos ROS 2, aprendimos a utilizar los topics de ROS 2 para establecer comunicación entre dos scripts: un publisher, que enviaba los datos, y un subscriber, que los recibía y procesaba. También se configuró la visualización en RViz para comprobar que los datos se estaban enviando y recibiendo de manera correcta.

2.2.2. Implementación

Una vez terminada la fase de aprendizaje se inició el desarrollo del código para identificar el objeto(caja). Antes de la construcción se identificaron una serie de etapas a desarrollar incrementalmente:

- Identificación de líneas
- Filtrado de datos
- Limpieza de datos no deseada
- Clusterización
- Identificar el objeto(caja)

Estos puntos se desarrollaron de forma secuencial una vez se había validado los resultados del punto anterior.

Primera etapa

El primer paso fue comprobar el funcionamiento de RANSAC (algoritmo de detección de líneas) para ello primero se filtró todo dato posible que no perteneciese al objeto(caja) para la comprobación del uso de esta herramienta y la creación de una línea en la imagen. También se comprobó como filtrar datos ya que se debían eliminar puntos como el suelo y restringir el espacio para que se filtrasen puntos como los que pertenecen a las paredes. Dado que para esta grabación, a diferencia de la anterior en la fase de aprendizaje, no debía aparecer personas o individuos nos situamos en una parte del espacio para poder ser eliminados posteriormente.

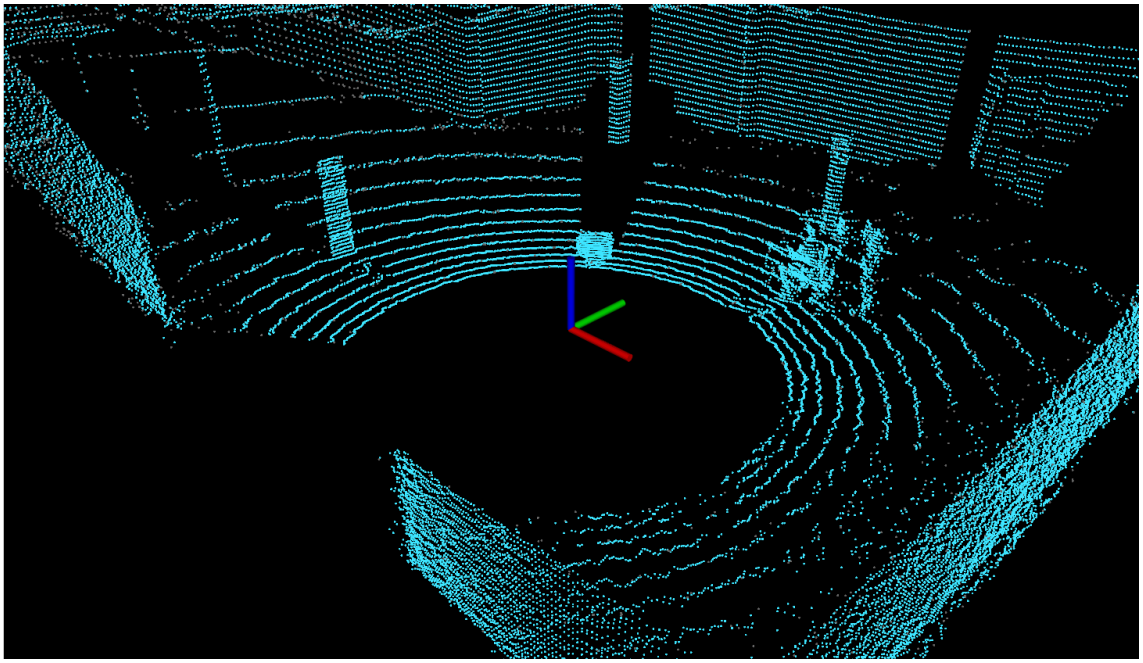

Figura 2.3: Nube de puntos para la visualización humana y entendimiento de la situación

La caja se situó en el espacio contrario a nuestra posición. La caja ocupa el espacio situado al lado negativo del eje x (linea roja). Dado esta situación fue necesario el reajuste de los puntos de la imagen para que el borde de la imagen se ajustase con los límites impuestos y se moviese de tal forma que uno de los bordes de la imagen coincidiese con uno de los límites impuestos.

Segunda etapa

Una vez realizado el cambio en la imagen lo siguiente fue utilizar RANSAC para crear una linea y pintarla en la imagen. Para realizar esto fue desarrollada una función auxiliar llamada detectar_lineas_ransac que lo que recibe es una lista de puntos para generar la recta que mejor se ajuste. Los puntos principales de la función son:

- Preparación de puntos
- Creación del objeto RANSAC
- Llamada fit (función de RANSAC) para ajuste de una recta a los puntos

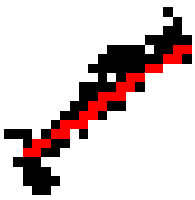

Figura 2.4: Recta creada

El código está desarrollado en Python y como resultado obtenemos:

- m: coeficiente de la recta
- b: punto por donde corta con el eje de las ordenadas en el plano
- inliers: puntos a los que se ajusta la recta

```
1 def detectar_lineas_ransac(puntos_filtrados):
```

```
        if len(puntos_filtrados) < 2:
            return None, None, np.array([])

        puntos_np = np.array(puntos_filtrados)
        X = puntos_np[:, 0].reshape(-1, 1)
        y = puntos_np[:, 1]

        ransac = RANSACRegressor(residual_threshold=0.2, min_samples=0.7)
        ransac.fit(X, y)

        m = ransac.estimator_.coef_[0]
        b = ransac.estimator_.intercept_
        inliers = puntos_np[ransac.inlier_mask_]

        return m, b, inliers
```

Explicación detallada del código:

- Primero se reciben los puntos a los que hay que ajustar una recta.
- Se comprueba si hay al menos dos puntos ya que para una recta es necesario al menos dos puntos.
- Se editan con numpy para que pasen de una lista de puntos a una matriz en la que cada fila aparecen las coordenadas del punto.
- Posteriormente se crea una lista de las x aparte de una lista de las y.
- Se crea el objeto ransac y se llama a fit para que ajuste una recta.
- Para terminar se recogen la pendiente, el punto donde corta con el eje de ordenadas y los puntos a los que se ha ajustado la recta.

De la lista de puntos inliers se recogen los dos puntos más alejados y se unen luego en una línea insertandolo más tarde en cv2(herramienta para insertar las lineas en la imagen).

Tercera etapa

Se detectaron puntos alejados que el programa unía en una misma recta este problema se solucionó mediante la identificación y eliminación de puntos lejanos y con pocos vecinos ya que estos no pertenecían al objeto que se quería escanear (caja). Esta identificación y eliminación se ha realizado en la función auxiliar filtrar_puntos_densos.

Figura 2.5: Punto alejado representación del problema

El código está desarrollado en Python y como resultado obtenemos:

- La lista de puntos válidos

```
def filtrar_puntos_densos(puntos, distancia_umbral, vecinos_minimos):
    """
    Elimina puntos que no tienen suficientes vecinos cercanos.
    """
    puntos_np = np.array([list(p) for p in puntos])
    if len(puntos_np) == 0:
        return np.array([])
    tree = KDTree(puntos_np[:, :2])  # usamos solo X e Y

    indices_validos = []
    for i, punto in enumerate(puntos_np):
        vecinos = tree.query_ball_point(punto[:2], r=distancia_umbral)
        if len(vecinos) >= vecinos_minimos:
            indices_validos.append(i)

    return puntos_np[indices_validos]
```

Explicación detallada del código:

- Se recibe la lista de puntos en los cuales hay que filtrar, la distancia umbral a la que hay que comprobar el número de vecinos, y los vecinos mínimos que ha de tener alrededor en esa distancia.
- Se editan con numpy para que pasen de una lista de puntos a una matriz en la que cada fila aparecen las coordenadas del punto.

- Se comprueba que hay más de 0 puntos la lista.

- Después se crea el KDTree con solo la x e y de cada punto.

- En un bucle se comprueba que puntos cumplen con esas restricciones y se añade en una lista su indice si el punto es válido.

- Se hace mediante el uso de query_ball_point dándole las coordenadas del punto a estudiar y un radio, devuelve la lista de sus vecinos dentro de ese radio teniendo que comprobar la longitud de la lista de vecinos devuelta siendo solo válido si supera o equipara el número de vecinos mínimo.

- Se devuelve la lista de los puntos válidos.

Esta función se llama antes de la identificación de las rectas con unos valores parametrizados siendo estos:

- distancia_umbral 0.3

- vecinos_minimos 5

Cuarta etapa

Durante las validaciones y pruebas de los resultados se observó que las columnas de la sala donde se realizó el escaneo se unían al objeto observado (caja). Tras el estudio de varias alternativas, se decidió usar una herramienta de minería de datos para la generación de clusteres llamada DBSCAN.

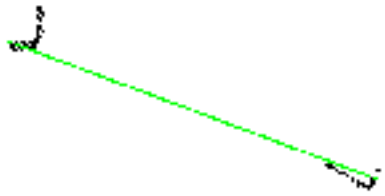

Figura 2.6: Imagen con recta de caja y columna

El código está desarrollado en Python.

```
1          dbscan = DBSCAN(eps=0.3, min_samples=5)
2          labels = dbscan.fit_predict(puntos_np[:, :2])
3          self.get_logger().info(str(len(puntos_np)))
4          self.get_logger().info(str(len(labels)))
5          for label in set(labels):
6              if label == -1:
7                  continue  # ruido
8              puntoscluster=puntos_filtrados_densos[labels==label]
```

Explicación detallada del código:

- Se crea un objeto DBSCAN proporcionando el valor eps, distancia máxima (en este caso 0.3) y el mínimo de puntos, mínimo de vecinos para seguir buscando (en este caso 5).
- Se llama a fit para que agrupe en distintas etiquetas los puntos que se le pasan a esta función entregando las coordenadas x e y de los puntos (no incluida la coordenada z). A los puntos que se identifican como ruido se le asigna la etiqueta -1.
- En base a estas etiquetas se hace un bucle de cada etiqueta donde más tarde se llamará para crear las distintas rectas.

Explicación de las etiquetas: La herramienta le da el mismo código numérico a todos los puntos de una misma agrupación. Por ejemplo para la imagen anterior crearía dos etiquetas, todos los puntos de la columna tendrán la misma etiqueta entre si y lo mismo con los puntos de la caja.

Quinta etapa

Una vez encontrada la lista de los puntos que para RANSAC están agrupados alrededor de la recta que no son los exactos pero son los que han ayudado a definirla, se debe incluir la altura o coordenada Z. Para realizar esto optimizando su rendimiento debido a la necesidad de incluir un bucle dentro de otro, se decidió desarrollar en lenguaje C++ por ser un lenguaje compilado, evitando el uso de Python.

```
int procesar_matrices(
    const std::vector<std::vector<double>>& A,
    const std::vector<std::vector<double>>& B,
    double C
) {

    size_t i=0;
    size_t j=0;
    double salida=0;
    double x=0;
    double y=0;
    double max=C;
    while(i<A.size()&&salida==0){
        x=A[i][0];
        y=A[i][1];
        while(j<B.size()&&salida==0){

            if(is_equal(B[j][0], x) && is_equal(B[j][1], y)){

                if(B[j][2]>0.1){
                    return -1;

                }
                if(B[j][2]>max){
                    max=B[j][2];
                }
            }
            j++;
        }
        j=0;
        i++;

    }
    if(max<-0.2){
        return -1;
    }
    return salida;
}
```

Explicación detallada del código:

- Recibe dos matrices A y B además de un número C, siendo A la lista de puntos sin Z, B la lista de puntos con Z y C la altura mínima obtenida.
- Se recorre en un bucle la matriz A recogiendo de cada fila el valor x e y

- Se recorre en esa iteración del primer bucle toda la matriz B recogiendo de cada fila el valor x e y, cuando las coordenadas x e y de los puntos de las matrices A y B coincidan, se comprueba si la coordenada Z es superior a la posición máxima estimada de la caja. Si esto ocurre se devuelve el valor -1. En este caso se ha tomado 0.1 como posición máxima de la caja (se ha tomado este valor por problemas de identificación del suelo, esto se explicará en detalle en el capítulo “Resultados”).
- Si no supera esa posición máxima, se actualiza la variable max si esta es menor que el valor de la coordenada Z.
- Una vez completados los bucles, se comprueba si la variable max supera el umbral que debería superar la caja (en este caso -0.2) para descartar posible ruido o un objeto que no puede ser la caja. En el caso de que no lo supere se devolverá -1.
- En caso de que todo sea correcto devuelve 0.

Para la representación de las rectas se ha usado un código de colores:

- Rojo para las rectas que han devuelto -1.
- Azul para las que han devuelto 0 pero se ha comprobado que tienen una longitud excesiva ya que no puede se el objeto (caja) (en este caso 0.7).
- Verde en caso de que todo sea correcto (caja)

-

3 Resultados

En este capítulo se presentan los resultados obtenidos con el código final para la validación del sistema. El objetivo principal es mostrar cómo se ha comportado el sistema en distintos escenarios (en 10 grabaciones) utilizando datos reales capturados con el sensor LiDAR Ouster OS2.

La sección se divide en dos partes:

- Pruebas y explicaciones: se muestran ejemplos visuales del funcionamiento del sistema. Cada figura va acompañada de una explicación técnica que detalla el caso especial sucedido y sus resultados obtenidos.
- Mejoras necesarias: se analizan las limitaciones encontradas durante las pruebas, así como los ajustes realizados para sortear ciertos problemas. También se proponen posibles líneas de mejora para futuros desarrollos, especialmente en lo relativo a la robustez del sistema frente a errores de medición o condiciones no controladas.

3.1. Pruebas y análisis

Ahora voy a proceder a realizar un análisis de los resultados de las pruebas seguiré una estructura de:

- Explicación de la situación y cual fue el motivo de elegir esa prueba
- La imagen devuelta por el programa

Para el visionado de las siguientes imágenes es importante recalcar cual es el motivo del color de cada línea:

- **Rojo** se refiere a cuando el objeto o agrupación de puntos analizados tiene una altura incorrecta.
- **Azul** si cumple con las restricciones de altura pero es demasiado largo.
- **Verde** cuando cumpla todos los criterios identificando esa agrupación como el objeto buscado.

La primera situación fue realizada de manera que fuese muy sencillo para el LiDAR, el objeto (caja) se situó en el centro de las columnas separado de todas de manera que fuese más fácil de diferenciar y que no se confundiese. Aparte de eso como no se sabía como iba a funcionar se posicionó la caja de manera que la parte más ancha estuvise de cara al LiDAR de manera que también facilitase la detección. Todo lo que no es el objeto se pintó con una línea roja como las

columnas que se pueden observar como cúmulos sueltos o las paredes. Y como se observa la imagen hay una gran densidad de puntos donde aparece la línea verde representando el objeto (caja). Por lo cual la identificación de los distintos cúmulos es correcta.

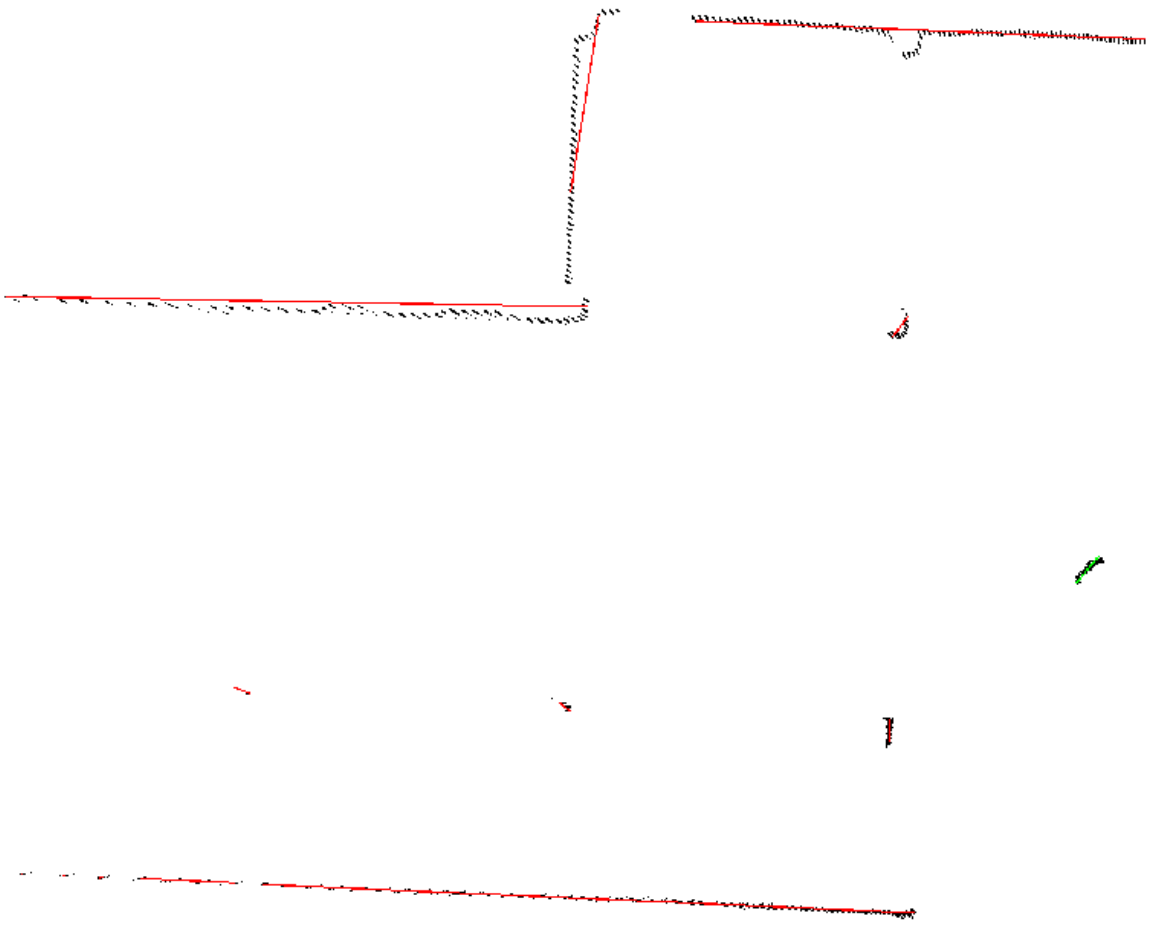

Figura 3.1: Primera situación

En las siguientes grabaciones se incrementó la dificultad de manera continuada con el fin de comprobar la robustez del código. Por lo cual en la segunda situación se alejó el objeto (caja) del sensor pero todavía lejos de cualquier otra columna y un poco angulado para provocar una dificultad a la hora de la detección. Aunque no representó una gran diferencia en la dificultad a la hora de detectar la caja.

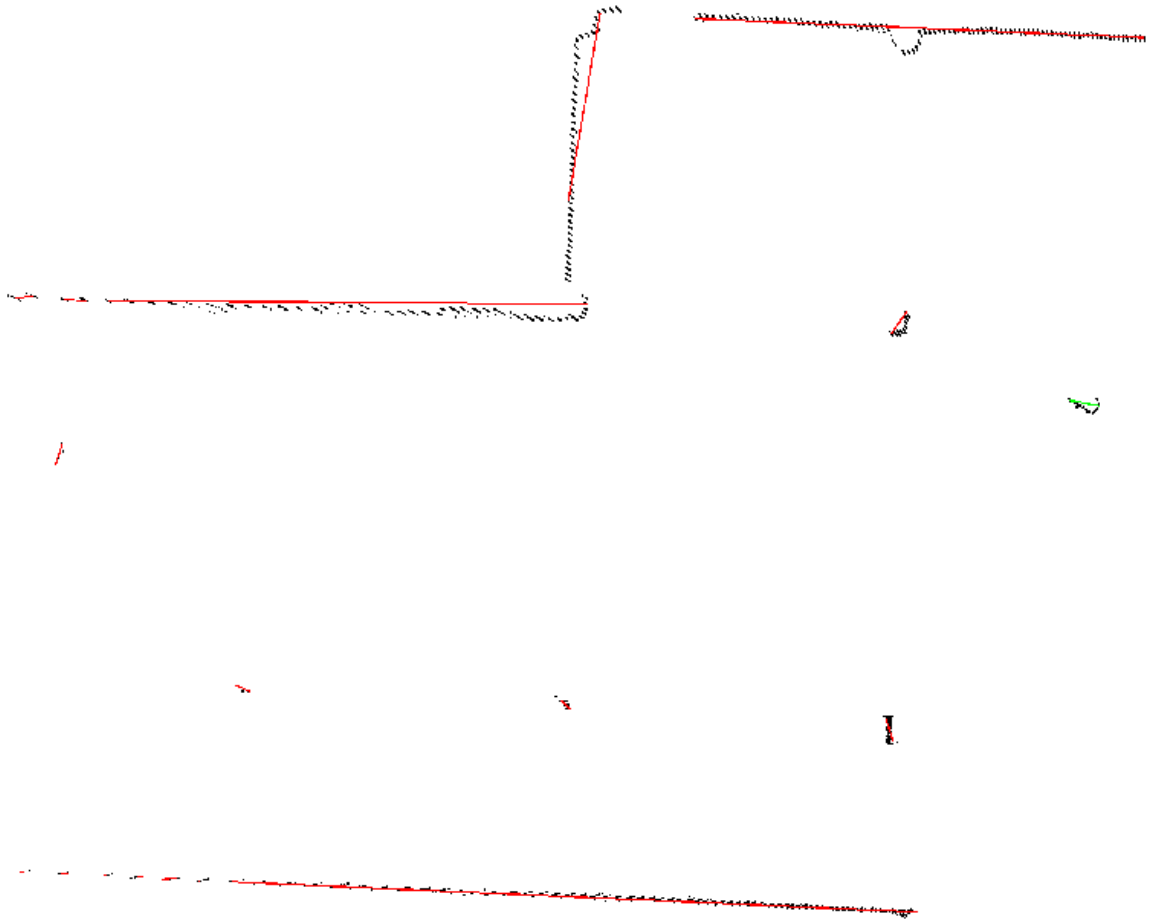

Figura 3.2: Segunda situación

En la tercera posición se decidió seguir alejando la caja para ver como iba a evolucionar con la distancia, manteniendo un poco angulado el objeto (caja) para intentar que la dificultad subiese de cualquier forma. Pero como se puede apreciar sigue detectandolo perfectamente aunque que se observa menos densidad de puntos.

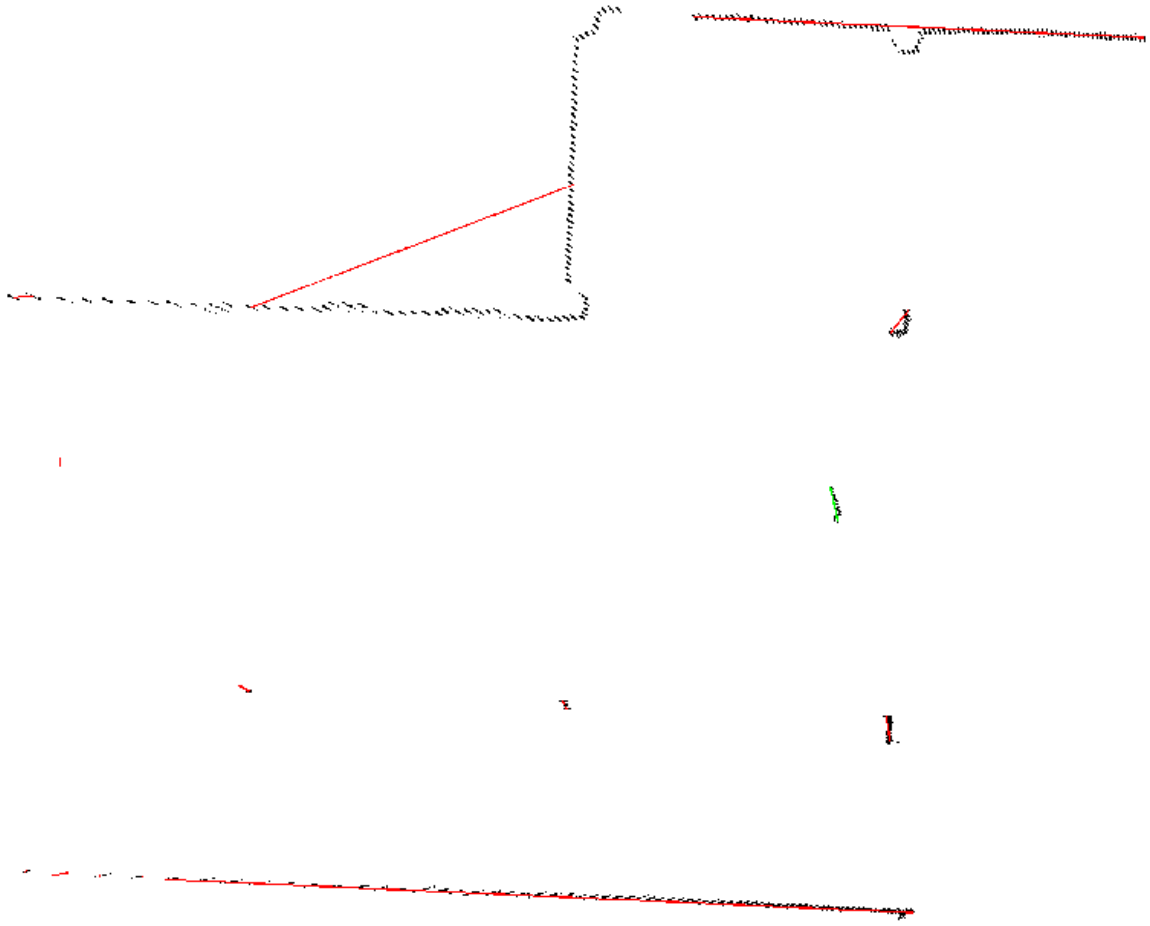

Figura 3.3: Tercera situación

En la cuarta situación se decidió acercar la caja a una de las columnas para incrementar considerablemente la dificultad por si provocaba que se uniesen en una sola línea el objeto (caja) y la columna. El algoritmo de clusterización agrupó correctamente separando ambos cúmulos de puntos en diferentes etiquetas. Mostrando que en efecto el código está preparado para esta situación de alta complejidad.

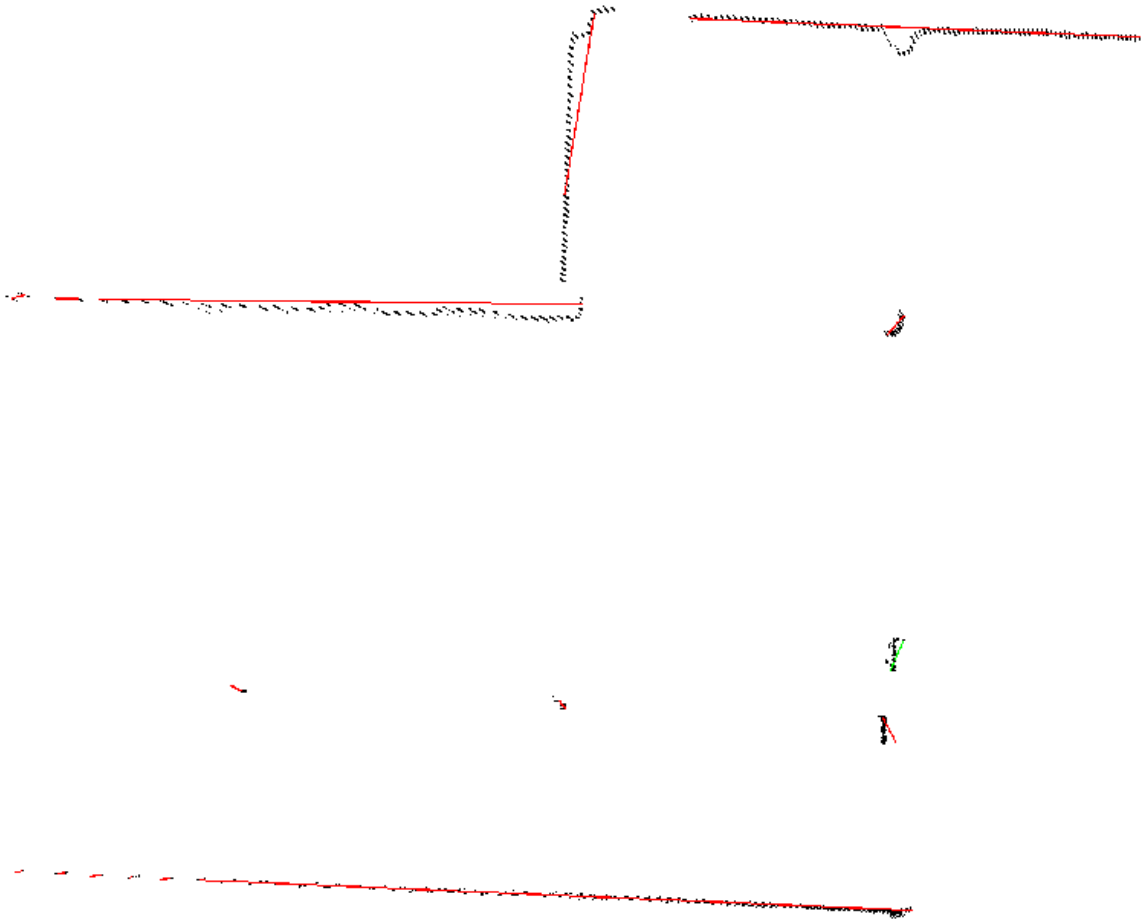

Figura 3.4: Cuarta situación

El resto de situaciones en las que se posicionó el objeto son bastante similares a las descritas con anterioridad. Esto nos ayudó a garantizar la calidad del código. Se muestran a continuación otras dos situaciones similares por el posible interés del lector.

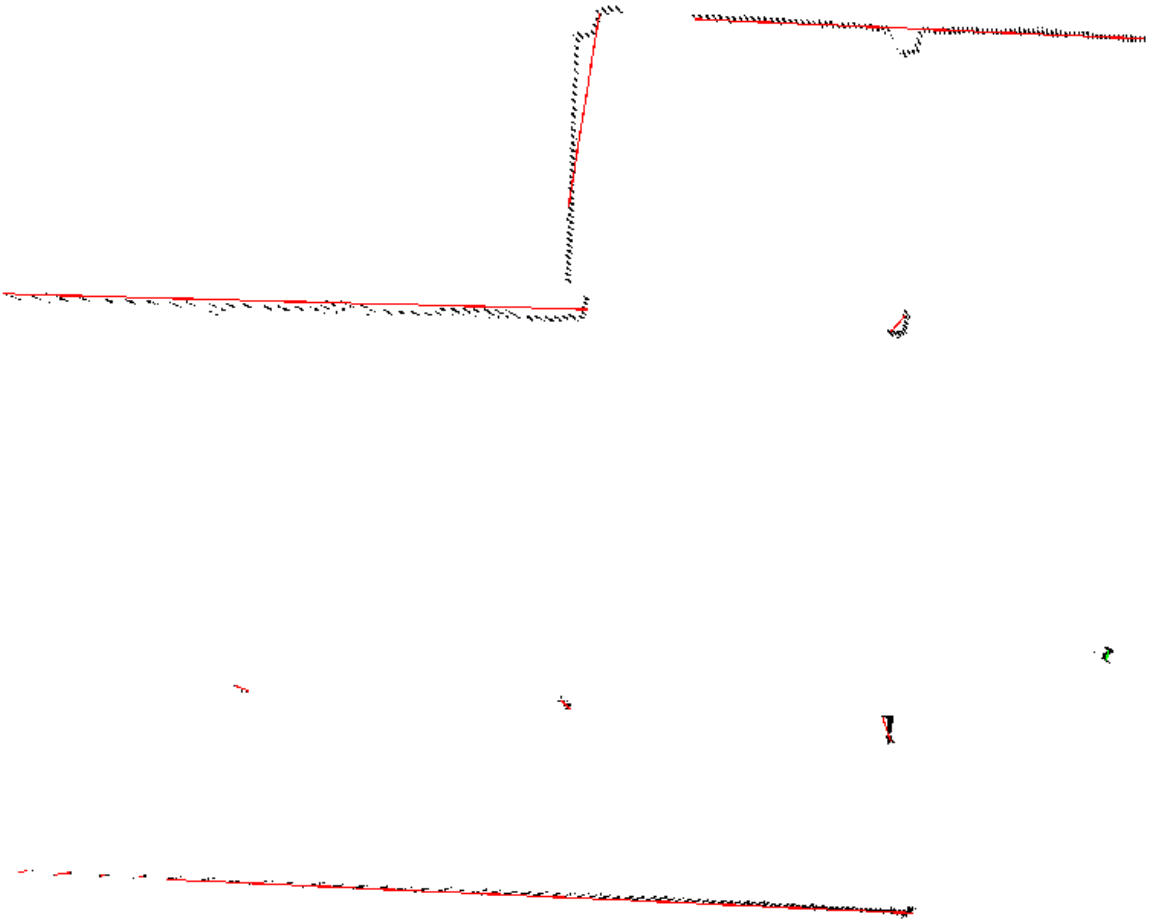

Figura 3.5: Quinta situación

Situación similar a la primera pero poniendo el objeto (caja) con su lado menos ancho.

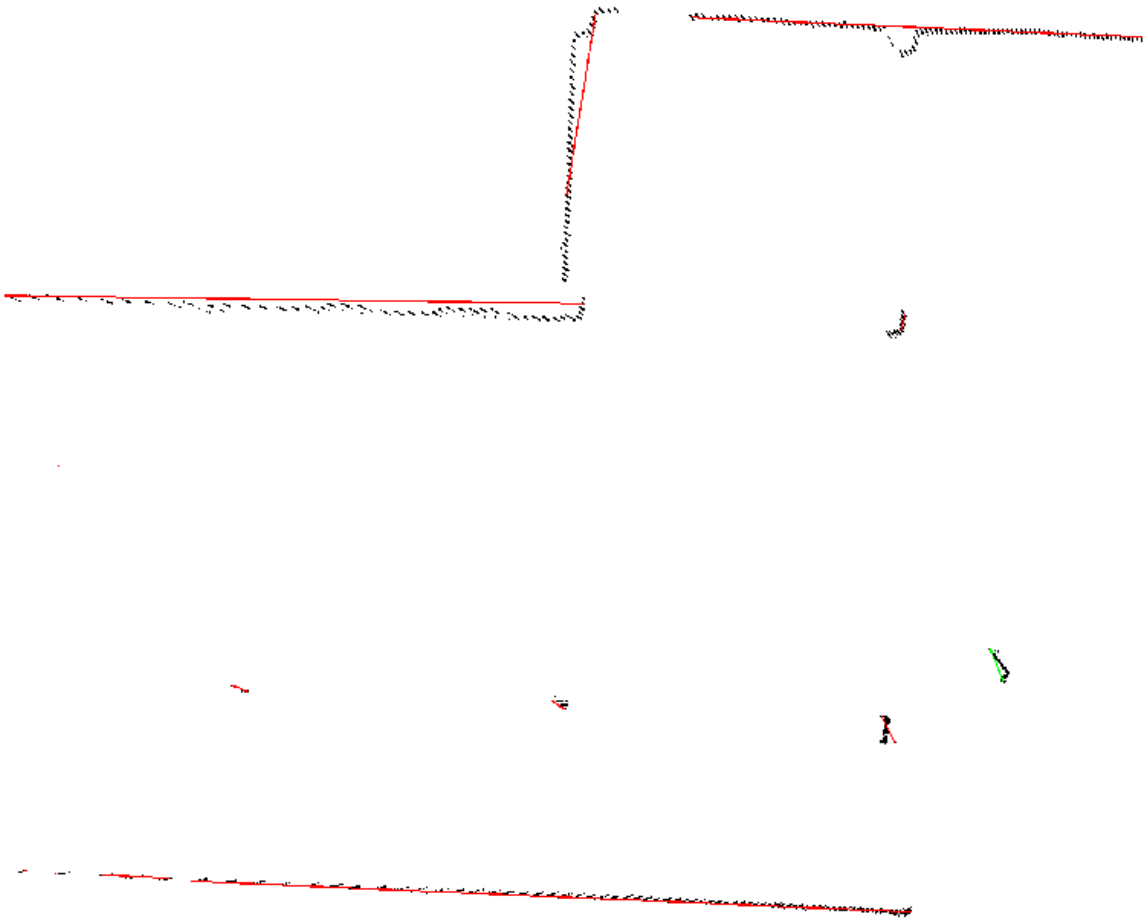

Figura 3.6: Sexta situación

Situación algo similar a la cuarta en el que el objeto (caja) se encuentra relativamente cercano a la columna.

3.2. Mejoras necesarias

Se identificó un problema al ser detectados puntos “ruido” por debajo del suelo, esto se cree que es debido a los reflejos del suelo. Este problema se sorteó en el código utilizando parámetros constantes como el 0.1 umbral máximo o el -0.2 umbral altura mínima requerida, umbrales explicados con anterioridad en la

fase de desarrollo en la quinta etapa. Otra de las soluciones implementadas por este problema fue sumar 0.6 a la variable min que es el valor de la z más bajo para eliminar el suelo.

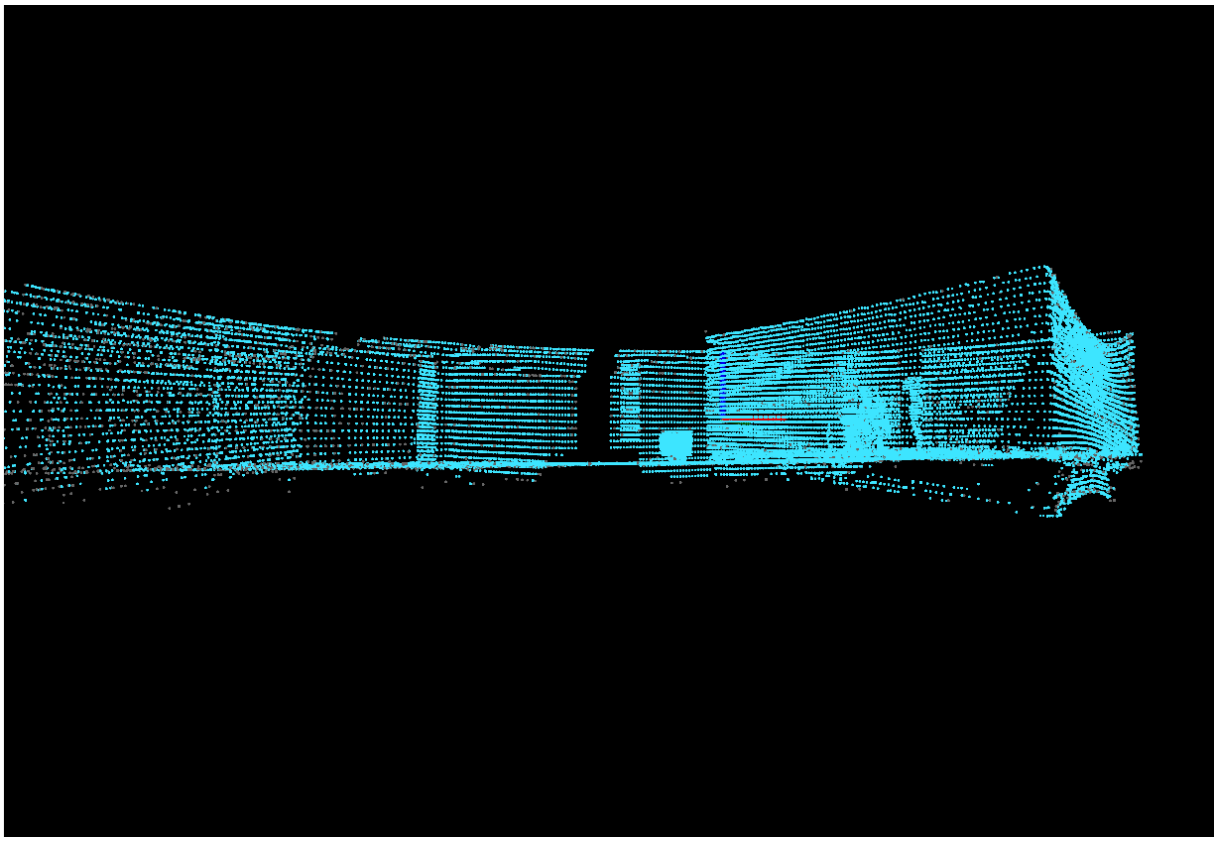

Figura 3.7: Muestra de puntos bajo el suelo

Como se puede apreciar visualmente en la imagen hay puntos "ruido" que se quedan por debajo del suelo. La idea inicial para identificar el suelo era coger el punto más bajo y filtrar un poco más arriba para quitar el suelo (+0.2) con el error de medición posible del sensor. Dado el problema descrito será necesario continuar los estudios para identificar correctamente el suelo de manera automática.

En la mayoría de las mediciones el punto "ruido" más bajo que está por debajo del suelo se encuentra entorno al -1.1 y sumándole + 0.6 se filtra el suelo. Pero hay veces que el punto más bajo lo detecta a -1.5 haciendo que la manera de sortear el problema manteniendo la idea inicial no funcione y se deje de filtrar los puntos del suelo. Como se explicó con anterioridad esto no puede ser así ya que si el robot se encuentra en otra situación no va a ser una solución satisfactoria. El caso de detectar en el -1.5 se puede apreciar en las siguientes imágenes en la que no hay filtrado del suelo

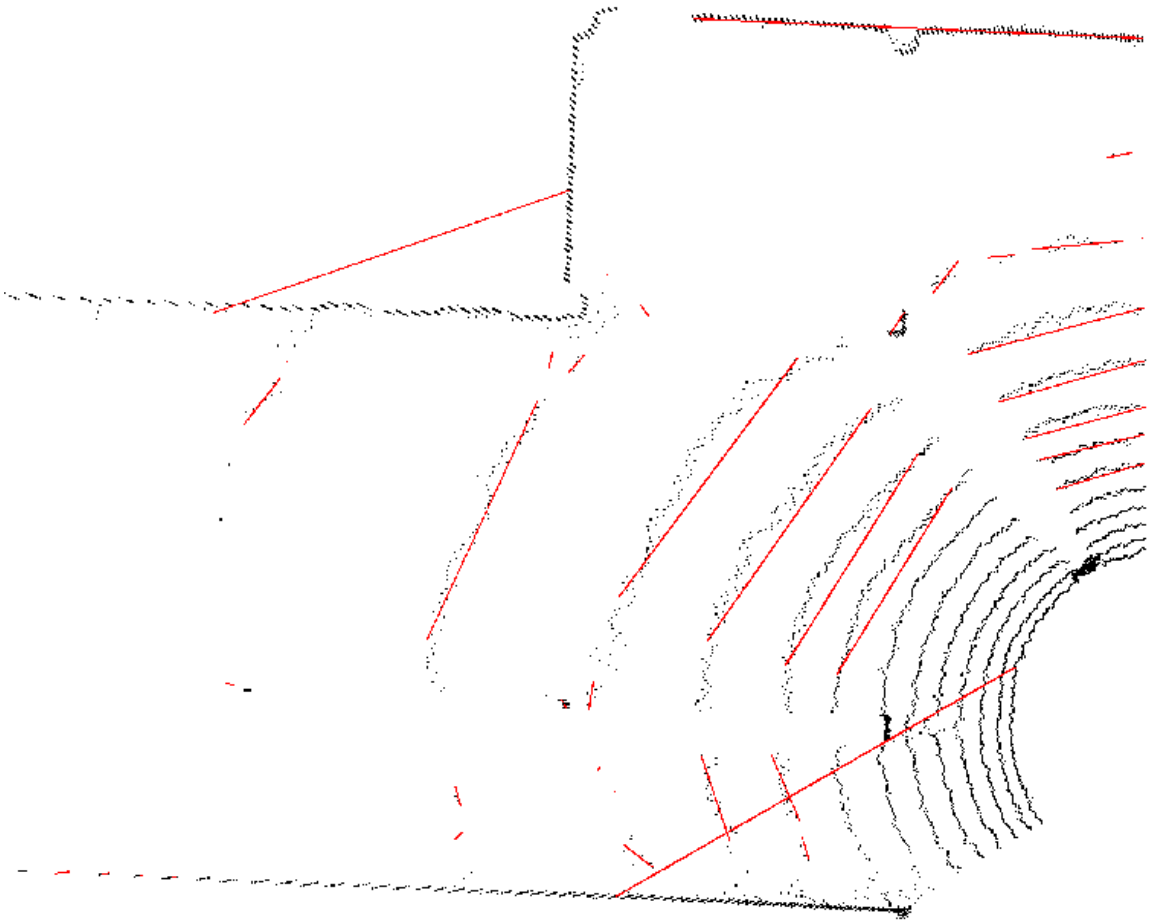

Figura 3.8: Sin filtrado del suelo por error “ruido”

Todas las lineas salen en rojo porque o superan la altura o se quedan debajo de la altura mínima para ser un posible objeto (caja) que se identificaría con la línea verde. Hay otros casos en los que se agrupa la caja y el suelo en una sola línea que al ser muy larga se pinta en azul como en la siguiente imagen:

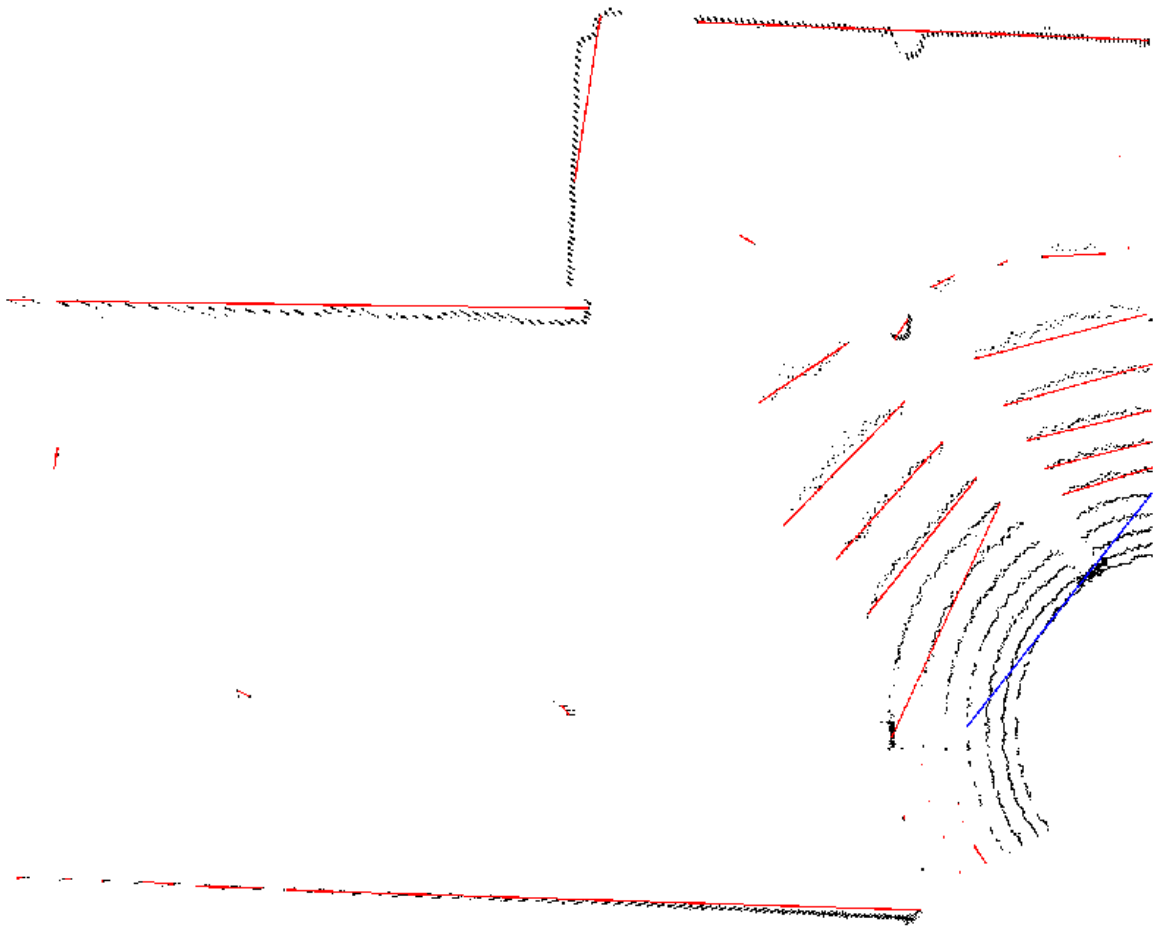

Figura 3.9: Sin filtrado de suelo por error linea con la caja y suelo

Esto demuestra que la configuración de la línea azul funciona ya que no teníamos ningún ejemplo con un objeto largo con una altura adecuada para ser el objeto (caja). Lo cual refuerza la idea de un trabajo bien hecho en su mayoría.

4 Impacto del desarrollo y cierre del proyecto

Una vez terminado completamente el desarrollo del proyecto, se puede mirar atrás para evaluar tanto los resultados obtenidos, como lo aprendido al afrontar diversos problemas. En esta sección se presenta el impacto como un resumen de los logros y las aplicaciones potenciales.

Este resumen no es solo para hacer una evaluación de si se han cumplido o no los objetivos previamente expuestos, sino también dar una base para la futura investigación de soluciones en ámbitos de la robótica.

4.1. Analisis impacto

El desarrollo de este proyecto ha tenido un impacto significativo en distintos aspectos tanto a nivel técnico como formativo.

- A nivel tecnológico se han conseguido distintos objetivos: se ha podido diseñar una solución que ha sido capaz de procesar datos LiDAR con rapidez, reconocer una estructura en un entorno 3D predefinida sin uso de un modelo IA y usar un módulo externo en un lenguaje diferente siendo este C++ para validar si la geometría esta dentro de unos parámetros. Y aunque esté todavía en una fase embriónica muestra que se puede sin usar IA combinar técnicas de visión por computador, algoritmos para crear agrupamientos y modelos que identifiquen una geometría en aplicaciones de percepción robótica.
- Uno de los puntos más destacados en este proyecto viene por el intentar demostrar que no es simpre necesario el uso de la inteligencia artificial para resolver problemas complejo. Aunque las inteligencias artificiales son herramientas muy potentes y que tienen mucho potencial suelen ser sistemas muy pesados además de que en muchas soluciones es desproporcionado a lo necesario ya que se puederesolver de otra manera. Donde otras personas hubiesen decidio usar redes neuronales o modelos entrenados para reconocimiento geométrico, el proyecto enseña como algoritmos más tradicionales como DBSCAN o RANSAC, que son mucho más ligeros sin necesidad de depender de grandes volúmenes de datos entrenados te permiten interpretar los datos. Y este planteamiento puede arrojar luz en situaciones en las que se encuentren recursos limitados o la velocidad sea clave.
- Además, el uso de herramientas como ROS 2, DBSCAN, RANSAC y el SDK de Ouster ha permitido construir una arquitectura modular, escalable y

adaptable a distintos escenarios. El sistema puede integrarse fácilmente en plataformas robóticas más complejas, como drones autónomos, y servir como base para futuros desarrollos en tareas de inspección, mapeo o navegación.

- Desde el punto de vista académico y personal, el proyecto ha supuesto una experiencia de aprendizaje intensiva. Se ha trabajado con tecnologías reales utilizadas en investigación avanzada, enfrentando problemas técnicos reales como el ruido en los datos, la necesidad de optimización en entornos embebidos y la integración de múltiples lenguajes de programación. Realizando todo esto he adquirido habilidades como mejora en resolución de problemas, poder diseñar incrementalmente, generar documentación técnica y saber pedir ayuda como cuando tuve que aprender del grupo de investigación del CVAR.

En resumen, el impacto del proyecto se refleja tanto en los resultados obtenidos como en el conocimiento adquirido, sentando una base sólida para futuras investigaciones o aplicaciones prácticas en el ámbito de la robótica y la percepción tridimensional.

4.1.1. Impacto y los Objetivos de Desarrollo Sostenible

El proyecto ha de seguir y contribuir a varios Objetivos de Desarrolo Sostenible (ODS) establecidos por las Naciones Unidos y recogidos en la Agenda 2030. En particular, se destacan los siguientes impactos:

- **ODS 4: Educación de calidad**

  El desarrollo del sistema ha implicado un proceso de aprendizaje intensivo en tecnologías emergentes como ROS 2, sensores LiDAR y algoritmos de visión por computador. Este conocimiento puede ser transferido a otros estudiantes e investigadores, fomentando una educación técnica avanzada y accesible.

- **ODS 9: Industria, innovación e infraestructura**

  El proyecto promueve la innovación tecnológica mediante el diseño de un sistema modular y escalable para el procesamiento de datos 3D, lo cual puede ser aplicado en sectores como la robótica, la logística o la automatización industrial.

- **ODS 11: Ciudades y comunidades sostenibles**

  La capacidad de detectar objetos mediante sensores LiDAR puede integrarse en sistemas de movilidad inteligente, vehículos autónomos o soluciones de accesibilidad urbana, contribuyendo a entornos urbanos más seguros y eficientes.

- **ODS 13: Acción por el clima**

  Al facilitar el desarrollo de tecnologías más eficientes y autónomas, el proyecto puede contribuir a reducir el consumo energético y las emisiones

asociadas a procesos manuales o ineficientes.

5 Conclusiones

Tras el desarrollo completo del proyecto, es posible realizar una valoración global de los resultados obtenidos, los retos enfrentados y las oportunidades de mejora identificadas. En esta sección se presentan las conclusiones principales, organizadas entorno a los logros alcanzados, los problemas técnicos detectados durante la implementación y una reflexión final que apunta a futuras líneas de trabajo. Esta visión de conjunto permite no solo evaluar el grado de cumplimiento de los objetivos iniciales, sino también establecer una base sólida para el desarrollo de soluciones más avanzadas en el ámbito de la robótica autónoma.

5.1. Resumen de logros y aplicaciones potenciales

Como se ha podido comprobar a lo largo del desarrollo del proyecto, se han alcanzado la mayoría de los objetivos planteados inicialmente. Este resultado, aunque pueda parecer sencillo en apariencia, representa un avance significativo en el diseño y validación de sistemas autónomos capaces de operar en entornos complejos o de difícil acceso. La implementación realizada demuestra que es posible aplicar soluciones de bajo coste y relativamente simples para tareas que requieren autonomía, percepción del entorno y toma de decisiones en tiempo real.

Este tipo de sistemas puede servir como base para aplicaciones más avanzadas, especialmente en el ámbito de la robótica móvil. Por ejemplo, un robot que deba identificar o recoger objetos en zonas donde el acceso humano es limitado (como espacios confinados, zonas con riesgo estructural, o entornos industriales peligrosos) podría beneficiarse directamente de los principios y técnicas desarrolladas en este trabajo. La capacidad de navegación autónoma, junto con la identificación de elementos clave del entorno, son pilares fundamentales para este tipo de aplicaciones.

5.2. Análisis de los problemas encontrados y sus implicaciones

A pesar de los resultados positivos, durante el desarrollo del sistema se identificaron varios desafíos técnicos que deben ser tenidos en cuenta para futuras implementaciones. Uno de los principales problemas fue la identificación precisa del suelo y del techo. En entornos reales, estas superficies no siempre presentan características uniformes o fácilmente distinguibles, lo que complica su detección mediante sensores o algoritmos de visión artificial.

Asimismo, la presencia de vigas, estructuras irregulares o elementos colgantes en el techo introduce ruido en los datos captados por los sensores, dificultando el filtrado y la segmentación de las zonas relevantes. Este problema es especialmente crítico en espacios industriales o construcciones antiguas, donde la geometría del entorno no sigue patrones predecibles.

Estos obstáculos ponen de manifiesto la necesidad de incorporar técnicas más robustas de percepción, como el uso combinado de sensores (fusión sensorial) o algoritmos que puedan identificar el suelo y techo.

5.3. Reflexión final y líneas de mejora

En conjunto, este proyecto no solo ha demostrado la viabilidad técnica de un sistema autónomo para tareas de identificación, sino que también ha puesto de relieve la importancia de una planificación cuidadosa, una integración eficiente de sensores y una programación adaptativa. La experiencia adquirida durante el desarrollo ha sido clave para entender las limitaciones actuales de este tipo de sistemas y, al mismo tiempo, ha abierto nuevas posibilidades para su evolución.

De cara al futuro, existen varias líneas de mejora que podrían potenciar significativamente el rendimiento del sistema:

- Mejora en la percepción del entorno: Incorporar sensores adicionales como cámaras estéreo o sensores de profundidad que no sean afectados por reflejos permitiría una reconstrucción más precisa del entorno tridimensional, facilitando la identificación de obstáculos y superficies.
- Simulación y pruebas en entornos reales: Ampliar las pruebas a escenarios más variados y realistas permitiría validar la robustez del sistema y detectar nuevas áreas de mejora.
- Implementación real en robot: Montar el sensor con un sistema de procesamiento encima del robot para comprobar en efecto que es lo necesariamente ligero, eficiente y veloz como se necesita.

En definitiva, este trabajo representa un paso sólido hacia la creación de soluciones robóticas más inteligentes, versátiles y aplicables a una amplia gama de contextos. Aunque aún existen retos por superar, los resultados obtenidos sientan una base firme sobre la que seguir construyendo.

Bibliografía

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6): 381–395, 1981.

[H+20] Charles R. Harris et al. Array programming with numpy. *Nature*, 585: 357–362, 2020.

[JRM23] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python. `https://github.com/pybind/pybind11`, 2023.

[Ope22] Open Robotics. Ros 2 humble hawksbill. `https://docs.ros.org/en/humble/`, 2022. Robot Operating System 2 distribution.

[Ope23a] Open Robotics. Understanding ros 2 nodes - ros 2 documentation. `https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html`, 2023. Tutorial oficial de ROS 2 Humble sobre nodos y herramientas de línea de comandos.

[Ope23b] OpenCV Team. Opencv: Open source computer vision library. `https://opencv.org/`, 2023. Version 4.x.

[Ous23] Ouster Inc. Ouster sdk. `https://github.com/ouster-lidar/ouster_example`, 2023.

[V+20] Pauli Virtanen et al. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17: 261–272, 2020.

[VGM+23] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2): 1029–1036, 2023, https://github.com/PRBonn/kiss-icp.

[Wir23] Wireshark Foundation. Wireshark: The world's foremost network protocol analyzer. `https://www.wireshark.org/`, 2023.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jul 02 20:21:01 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)