Universidad Zaragoza
1542

Trabajo Fin de Grado

Etiquetado semántico automático de una nube de puntos 3D de un entorno, para mejorar el reconocimiento semántico de una escena

Automatic semantic labelling of a 3D point cloud of an environment to improve semantic recognition of a scene

Autor

Juan Plo Andrés

Directores

Luis Riazuelo Latas

Luis Montano Gella

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2023

AGRADECIMIENTOS

Debo agradecer la ayuda otorgada para la realización de este proyecto a dos personas. Ellos son mis dos directores del trabajo fin de grado, Luis Riazuelo y Luis Montano. Agradezco toda la ayuda, tutoría y seguimiento que han realizado durante este largo y costoso proceso, por todas las dudas resueltas y aclaraciones realizadas. Además, gracias a la guía de realización del trabajo que me han otorgado y por todo el material que me han proporcionado. Sin duda, no habría podido completar este proyecto sin la ayuda de ambos.

También a mi familia, quienes me han apoyado y animado a seguir con el desarrollo de este increíble proyecto.

Y por último, quiero agradecer a la Universidad de Zaragoza, y a la Escuela de Ingeniería por otorgarme la oportunidad de obtener una formación de calidad.

RESUMEN

Actualmente, los algoritmos de reconocimiento de escenas o de segmentación semántica juegan un papel crucial en el desarrollo y aplicación de múltiples tecnologías que abarcan campos como la visión por computador, la automatización industrial o la conducción autónoma, entre otras. Sin embargo, dichos algoritmos requieren ser entrenados con grandes cantidades de datos especialmente tratados. Gracias al entrenamiento, los algoritmos obtienen la capacidad de realizar tareas como la clasificación de imágenes, detección de objetos y la identificación de patrones.

El desafió es obtener conjuntos de datos lo suficientemente amplios y representativos para lograr una precisión y fiabilidad aceptables en el reconocimiento de escenas.

En este proyecto se busca desarrollar un método que optimice y automatice el proceso de etiquetado de los datos y la obtención de los conjuntos de datos.

Para llevar a cabo el proyecto se cumplieron ciertos objetivos utilizando diversas herramientas de tratamiento de objetos 3D. Tales como la obtención de un modelo 3D de un entorno y la reducción de tamaño del modelo para posterior etiquetado y manipulación, seguido del etiquetado del modelo para posterior carga en un simulador. Una vez en el simulador, se realizarán simulaciones de trayectorias generadas automáticamente para la obtención de puntos 3D etiquetados.

La manipulación del modelo se llevó a cabo mediante scripts implementados en Python y C++. Además, se llevó a cabo una labor de estudio y análisis de código *Open Source* para su modificación y adaptación a los objetivos del proyecto.

Durante el desarrollo del proyecto se utilizaron múltiples herramientas *Open Source*. Estas herramientas son CloudCompare, Blender, Blensor [1], PointLabeler [2] y MeshLab.

Una vez cumplidos los objetivos y desarrollado el método de etiquetado y obtención de datos automáticamente se llevaron a cabo diversas pruebas y evaluaciones de resultados para comprobar el correcto funcionamiento del método y la eficiencia del mismo.

Gracias a este método se podrán entrenar múltiples algoritmos de segmentación semántica que resultarán útiles y beneficiosos en aplicaciones futuras, logrando así unos resultados sólidos y eficientes por parte de estos tan reclamados y útiles algoritmos.

Índice

Capítulo 1

Introducción y objetivos

1.1. Motivación y contexto

Los algoritmos de reconocimiento de escenas juegan un papel crucial en diversas áreas de la tecnología y sus múltiples aplicaciones benefician a la sociedad. Estos algoritmos desempeñan un papel crucial en el desarrollo y aplicación de múltiples tecnologías que abarcan desde la visión por computador hasta la automatización industrial y la conducción autónoma, entre otras. Existen diversos métodos y arquitecturas de redes neuronales que utilizan estos algoritmos para llevar a cabo diferentes funciones que requieren del reconocimiento de escenas. Uno de ellos es RangeNet++ [3], el cual se enfoca en el rastreo de objetos en movimiento en vídeos, lo que lo hace especialmente útil para aplicaciones como vigilancia, conducción autónoma, análisis de tráfico y sistemas de asistencia a la conducción, haciendo uso de técnicas avanzadas de aprendizaje profundo y procesamiento de imágenes para lograr una detección y seguimiento precisos y eficientes de objetos en movimiento. (Figura 1.1).

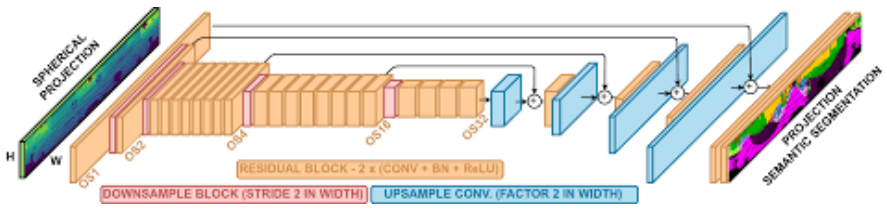


Figura 1.1: Arquitectura de RangeNet++

Existen otros muchos métodos; por ejemplo, los que usan Redes Neuronales Convolucionales o Convolutional Neural Networks (CNNs): AlexNet [4] o VGGNet [5] Sin embargo, dichos algoritmos requieren ser entrenados usando grandes cantidades de datos especialmente tratados. Este proceso de entrenamiento es esencial para dotar a los algoritmos de la capacidad de 'ver' y comprender el entorno, permitiéndoles llevar a cabo tareas como la clasificación de imágenes, la detección de objetos y la identificación

de patrones. El desafío de obtener conjuntos de datos lo suficientemente amplios y representativos para lograr una precisión y fiabilidad aceptables en el reconocimiento de escenas es el factor limitante.

Los conjuntos de datos para el entrenamiento son obtenidos de escenas reales. Para ello se utilizan nubes de puntos que representan dichas escenas (Figura 1.2). En este trabajo se utilizan nubes de puntos obtenidas mediante 3D LiDAR. LiDAR(Light Detection and Ranging) es un sistema de detección remota que usa pulsos de luz láser para medir la distancia y crear imágenes tridimensionales del entorno. No obstante, desde el punto de vista de la computación solo vemos coordenadas en un mundo tridimensional, es por ello que cobra importancia la segmentación semántica.

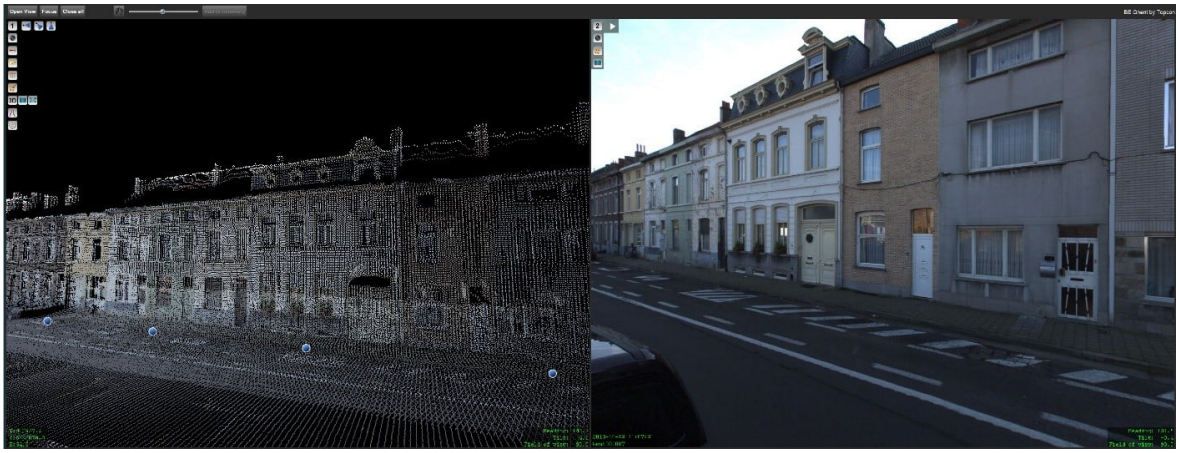


Figura 1.2: Nube de puntos 3D v.s. realidad [6]

La segmentación semántica es un proceso por el cual se identifica y se asigna una etiqueta a, bien, los píxeles de una imagen, o bien, los puntos de una nube de puntos, la cual les atribuye un significado, permitiendo así el reconocimiento de distintas regiones en las mismas. Un ejemplo (Figura 1.3) de una aplicación para la conducción autónoma donde se divide la imagen en dos clases: Carretera y Vehículos.

Figura 1.3: Imagen dividida en dos clases: Carretera y Vehículos [7]

Hasta ahora, el proceso de obtención de los conjuntos de datos era el siguiente:

1. Mediante el uso de un escáner 3D láser acoplado a la cámara de un robot o vehículo que captura una nube de puntos de la escena.

2. Las diferentes zonas en la nube de puntos, que representan diferentes partes de la escena (suelo, techo, paredes, vehículos, etc.), se etiquetan manualmente para su posterior uso en el entrenamiento de algoritmos y reconocimiento.

El coste de realizar el etiquetado manualmente es muy elevado dada la gran cantidad de datos de la nube y también debido a que la obtención de las nubes de puntos tridimensionales se hace mediante múltiples escaneos láser; para formar la nube de puntos etiquetada resultante del escaneo hace falta fusionar todos los escaneos. El etiquetar todos los escaneos y que los puntos de diferentes escaneos que corresponden a la misma zona de la realidad sean etiquetados con la misma etiqueta conlleva un coste extremadamente elevado. El proyecto tiene como objetivo principal el desarrollo de un método que permita generar el etiquetado automático de nubes de puntos 3D de manera más eficiente. En un determinado escenario, se simularán diferentes trayectorias de un robot que lleve embarcado el escáner láser, y se evaluará que el etiquetado de las diferentes zonas de la escena a partir de la nube de puntos 3D se realiza correctamente para todas ellas.

Mientras que hasta ahora se obtenían los datos de la ejecución de una simple trayectoria en una escena, lo cual resulta costoso en tiempo y en recursos, el método de etiquetado automático permitirá la obtención de elevadas cantidades de datos de las trayectorias que se desee mediante un etiquetado más eficiente.

1.2. Objetivos y tareas

El objetivo general del proyecto es desarrollar un método que optimice el proceso de etiquetado de los datos y la obtención de los conjuntos de datos. Para ello es necesario cumplir los siguientes objetivos específicos:

1. Obtener un modelo 3D del entorno

2. Reducir de la densidad de puntos de la nube 3D para reducir la carga computacional y permita la manipulación de la misma.

3. Etiquetar las diferentes zonas de la nube de puntos, para darle el significado semántico que creamos que representa mejor la realidad de la misma.

4. Cargar la nube de puntos en un simulador que nos permita realizar las trayectorias de un escáner láser que se consideren necesarias para la obtención de los datos con significado.

5. Obtener puntos 3D etiquetados de las simulaciones.

Las tareas que se han realizado para cumplir los objetivos son las siguientes:

1. Reducción de la densidad de la nube de puntos.
2. Creación de una malla a partir de la nube de puntos reducida.
3. Etiquetado de la nube de puntos para darle significado semántico.
4. Carga de la nube en un simulador.
5. Desarrollo de una herramienta para la creación de trayectorias.
6. Creación de un modelo de láser para la simulación que se asemeje al que se usará en la realidad.
7. Desarrollo de una herramienta que permita llevar a cabo la simulación completa de una trayectoria y la captura de datos.
8. Desarrollo de una herramienta que interprete la información obtenida de la simulación.

1.3. Contenido de la memoria

El Capítulo 2 describe la preparación de la malla tridimensional que se usará en futuras simulaciones, partiendo de la nube de puntos que representa una escena. Este capítulo también describe la creación de una malla tridimensional a partir de la nube de puntos diezmada para posterior uso en la simulación de trayectorias. En el Capítulo 3 se tratará el etiquetado de la nube de puntos, el cual desarrollara el proceso de preparación de los datos para otorgarles un significado que corresponde con la realidad, que servirá tanto para obtención de datos etiquetados de forma automática y como *Ground truth*. En el Capítulo 4 se abarcará el proceso de preparación del simulador, generación de trayectorias, preparación de la escena a simular, puesta en marcha de la simulación y el análisis de los resultados obtenidos de la misma. El Capítulo 5 muestra varias simulaciones y la evaluación de los resultados obtenidos de las mismas. Para terminar en el Capítulo 6 se concluirá con las conclusiones técnicas del proyecto y la puesta a futuro del proyecto.

Capítulo 2

Diezmado y creación de la malla tridimensional a partir de una nube de puntos

2.1. Objetivo

El objetivo es reducir la densidad de puntos de la nube 3D para facilitar su manipulación y procesamiento sin perder la información relevante de la escena. Una nube de puntos 3D que representa una escena contiene cantidades inmensas de puntos, los cuales están separados por una distancia milimétrica, es decir, hay información redundante sobre la escena que puede ser descartada manteniendo el significado de la misma.

2.2. Origen de la nube de puntos usada en el proyecto

Para el desarrollo del proyecto se ha usado una nube de puntos que representa una sección del túnel de Somport, túnel que conecta España y Francia (Figura 2.1). Este túnel consta de túnel principal de 7.7 km, y varias galerías laterales. Esta fue obtenida mediante LiDAR. LiDAR(Light Detection and Ranging) es un sistema de detección remota que utiliza pulsos de luz láser para medir la distancia y crear imágenes tridimensionales del entorno. En este trabajo se va a procesar solo una parte del túnel, la correspondiente al cruce entre una zona del túnel central y una galería lateral. Esta zona está representada por una nube de más de 20 millones de puntos, siendo solo 69 metros de los 8.600 metros que abarca el túnel.

(a) **(b)**

Figura 2.1: Túnel de Somport: (a) Realidad; (b) Nube 3D

2.3. Procedimiento de diezmado

Para llevar a cabo el diezmado, se estudiaron dos algoritmos [8] que permitieran la reducción de la densidad de la nube de puntos sin llegar a perder información relevante de la escena que representa y reduzca considerablemente el coste de computación.

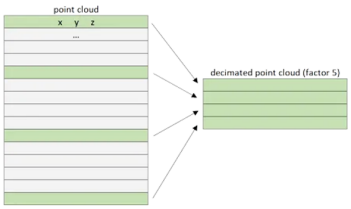


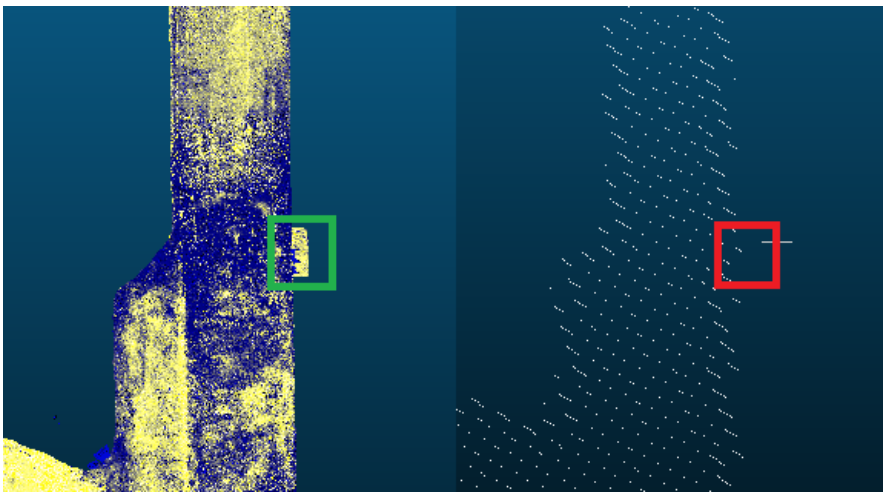
Figura 2.2: Reducción mediante factor 5

2.3.1. Subdivisión Aleatoria

Consiste en eliminar un porcentaje de puntos de la nube original de acuerdo a un factor de reducción. Dicho de otra manera, conservamos un punto por cada cantidad de puntos equivalente al factor (Figura 2.2).

El resultado de este método no es convincente del todo debido a que se basa en el orden en que aparecen los puntos en la información proporcionada y podría perderse información relevante de la escena. Un claro ejemplo de este caso es cuando queremos reducir la nube en varios órdenes, y como resultado perdemos la información de zonas de interés representadas por un menor número de puntos (por ejemplo los apeaderos del túnel, ver Figura 2.3).

Esta representación se obtuvo de la experimentación de un diezmado aleatorio con factor igual a 125 sobre una nube de más de 4 millones de puntos, lo que resulta en una nube reducida de aproximadamente 32 mil puntos.



(a) **(b)**

Figura 2.3: Perdida de información de un apeadero: Nube original (a); Nube diezmada mediante un factor de 125.

Al intentar crear una malla tridimensional con la nube reducida mediante este método se comprobó que, debido a la naturaleza de la separación, resultaba imposible computar las normales de los puntos. El plugin de creación de la malla necesita las normales de cada punto, es por ello y por los resultados que proporciona que este método fue descartado.

2.3.2. Subdivisión en vóxeles

Este método consiste en la división de la nube 3D en 'vóxeles' (Figura 2.4), término que en el ámbito de los gráficos 3D se usa para referirse a cubos que subdividen un espacio tridimensional (Figura 2.5), y obtener la media de las coordenadas de los puntos abarcados por el área del 'vóxel', creando así un nuevo punto tridimensional con la media de las coordenadas, formando así la nube diezmada con los puntos obtenidos de cada vóxel en el que se divide la nube de puntos. La cantidad de puntos de la nube resultante dependerá del tamaño de los vóxeles.

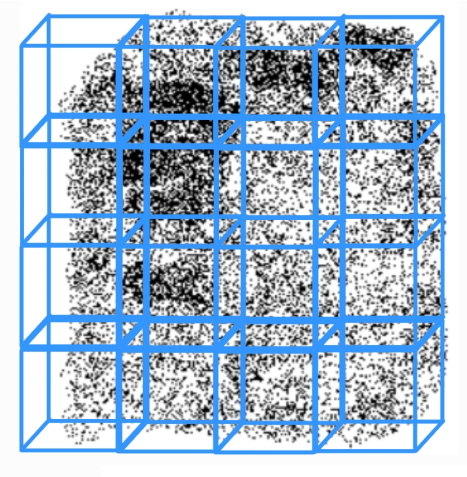


Figura 2.4: Vóxeles dividiendo una nube de puntos en 16 partes

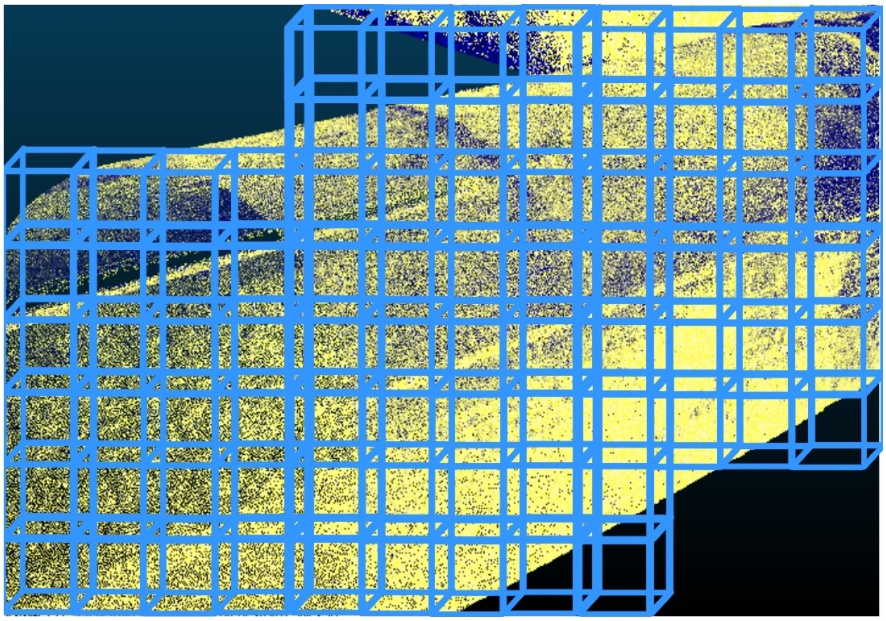


Figura 2.5: Representación de Vóxeles

Se experimentó este método utilizando distintos valores para el área de los "vóxeles", que dividirán la nube de puntos, intentando llegar a la máxima reducción posible que no haga perder la información de la escena que representa la nube. El procedimiento era sencillo, se empezó con un valor de área 0.01 y se fue aumentando hasta que fuera notable la pérdida de información para quedarse con el último valor de área que mantuviera la información esencial. Como resultado de la experimentación, se obtuvo un diezmado de aproximadamente el 99.375 %, partiendo de una nube de más de 4 millones de puntos y obteniendo una nube de menos de 25 mil puntos (Figura 2.6).

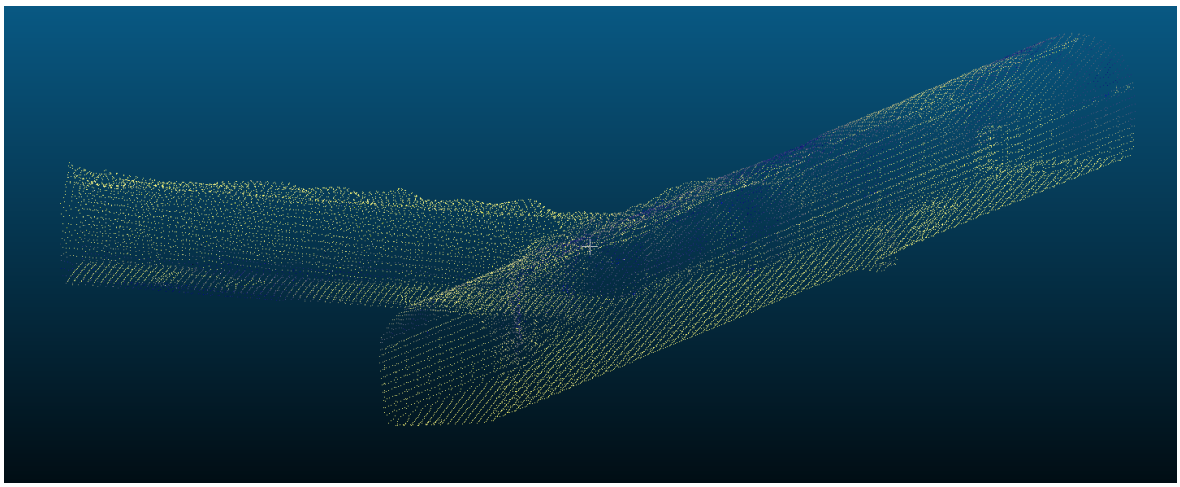


Figura 2.6: Nube diezmada mediante Vóxeles

El método escogido finalmente fue el diezmado mediante 'vóxeles", ya que tras la experimentación de ambos métodos se comprobó que la naturaleza de la división aleatoria no permitía la computación de las normales de los puntos de la nube. Además, debido a la naturaleza del método, se pierde la información de las zonas de interés representadas por una densidad baja de puntos. Por otro lado, el método de división mediante 'vóxeles' logra una reducción aceptable y permite la posterior creación de una malla tridimensional.

2.4. Creación de la malla tridimensional

La creación de gráficos 3D se basa, a alto nivel, en la unión de primitivas geométricas formando una figura, la más básica y a la vez más usada son los triángulos, ya que a nivel computacional solo se necesita de la información de sus vértices, es decir, las coordenadas de tres puntos tridimensionales.

La creación de la malla se realizó mediante el uso de un *plugin* de CloudCompare llamado *PoissonRecon* (Figura 2.7), el cual toma los puntos de la nube de puntos y en función de la distancia entre los mismos, crea triángulos para formar el modelo 3D (Figura 2.8).

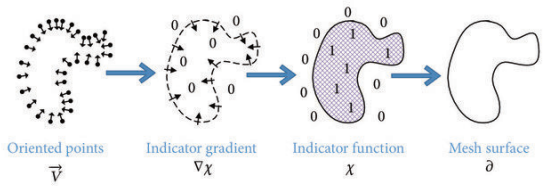


Figura 2.7: Generación de la superficie de una malla a partir de puntos orientados mediante PoissonRecon

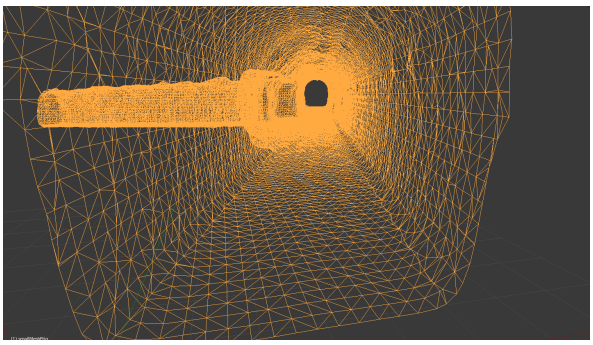


Figura 2.8: Vértices de la malla generada

Aquí el diezmado juega un papel importante, debido a que el modelo 3D de la figura 2.8 contiene más de 32 mil vértices partiendo de una nube de puntos de menos de 25 mil. Sin el diezmado, el modelo hubiese contenido cerca de 5 millones de vértices, una cantidad inasumible para poder ser computada. Una justificación adicional para el diezmado se basa en lo que representan más o menos vértices; por ejemplo, en la figura 2.9 se aprecian dos planos con la misma textura, pero la realidad es que el de la izquierda tiene 4.225 vértices y el de la derecha simplemente 4 como se aprecia en la figura 2.10.

Figura 2.9: Dos planos con la misma textura formada por distintas cantidades de vértices

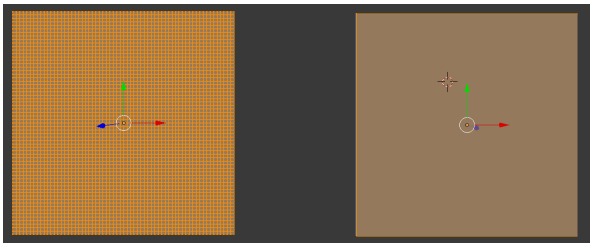


Figura 2.10: Vértices de los planos: Derecha: 4.225 vértices; Izquierda: 4 vértices

Ejecutando el plugin de PoissonRecon y comparando la malla resultada con la nube de puntos, se concluyó que la malla no perdía información relevante en el proceso. Sobre todo se comprobó la información relativa a los apeaderos debido a su poca densidad de puntos y se confirmó la integridad de los mismos, sin perder información relevante de

ellos (Figura 2.11). Como resultado se obtuvo la misma conclusión sobre la resolución en función del tamaño de vóxel de la experimentación con distintos tamaños.

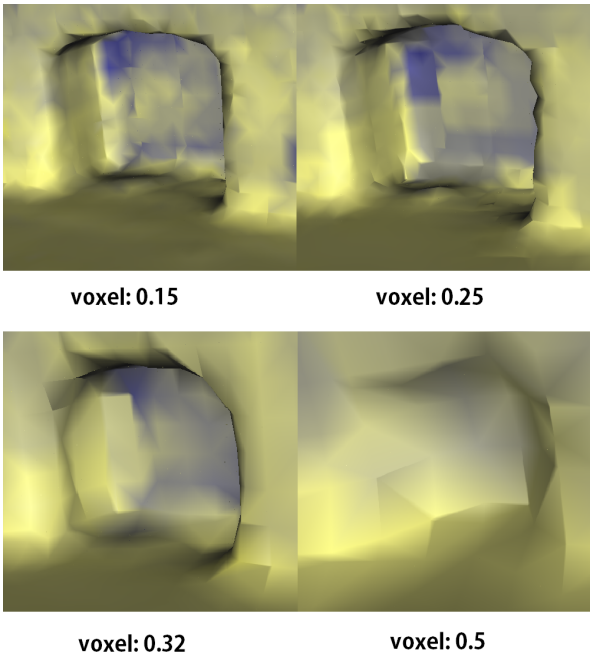


Figura 2.11: Apeadero formado con tamaños de vóxel diferentes

Capítulo 3

Etiquetado de la nube de puntos

3.1. Objetivo

El objetivo es generar de forma automática datos etiquetados para entrenar algoritmos de segmentación semántica. Este etiquetado servirá tanto para las futuras simulaciones que se hará sobre la malla etiquetada, como para generar lo que se conoce como *Ground truth*. Esta es la información que sabemos que es real o verdadera, obtenida de la observación o medición que servirá para comprobaciones de lo precisos y acertados que han sido los resultados de los algoritmos de segmentación semántica durante el entrenamiento de los mismos y para evaluar los resultados obtenidos de las simulaciones.

3.2. Procedimiento

Pese a la reducción de la densidad de puntos previa al etiquetado, la labor de asignar una etiqueta a cada uno de los puntos de una nube de puntos es una labor costosa. Una opción no asumible sería mediante observación en un visualizador 3D de la nube e ir apuntando las coordenadas de los puntos y anotar la etiqueta que queremos asignarle para realizar un cambio manual del color de cada uno de los puntos en función de las etiquetas anotadas. Tras una reducción de densidad de una nube de puntos que represente una escena completa puede contener cerca de un millón de puntos y etiquetarlos manualmente todos resulta una tarea imposible.

Existen herramientas que permiten el etiquetado de forma semi-automática de nubes de puntos. Una de ellas es PointLabeler (Figura 3.1) [2].

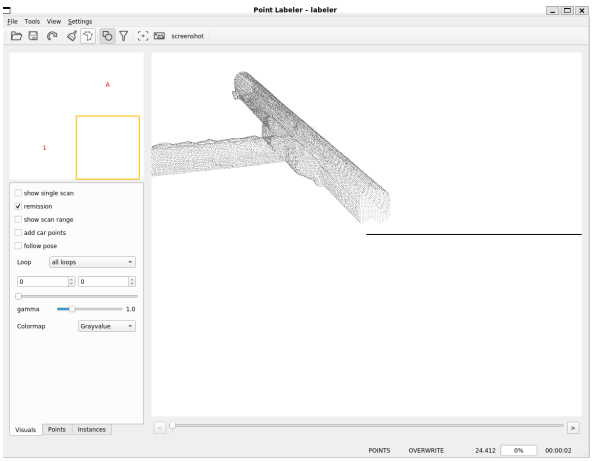


Figura 3.1: Herramienta PointLabeler

PointLabeler es una herramienta *Open Source* creada por una asociación del grupo de Fotogrametría y Robótica, el grupo de Visión por Computador y el grupo de Sistemas Autónomos Inteligentes de la Universidad de Bonn. La herramienta surge de la necesidad de etiquetar conjuntos de datos inmensos y permite el etiquetado de los mismos de forma semi-automática. Estos conjuntos de datos pertenecen a KITTI [9], un dataset de escaneos de escenarios relacionados con la conducción y la navegación (Figura 3.2).

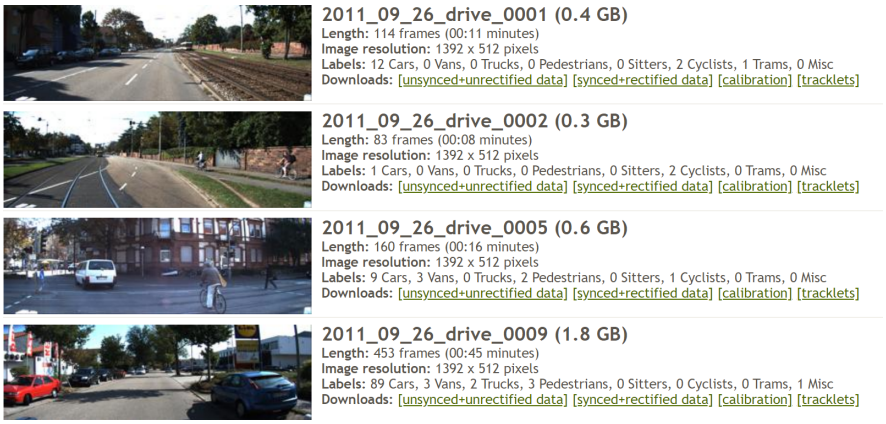


Figura 3.2: Escenarios escaneados del dataset KITTI

Los datos que pertenecen al conjunto de datos de KITTI siguen un formato estandarizado, es por ello que la herramienta PointLabeler fue creada para trabajar con conjuntos de datos con dicho formato. Para poder trabajar con la herramienta es necesario traducir los datos originales de la escena que queremos etiquetar a este formato; para ello se desarrolló un script en Python que llevase a cabo la traducción (Véase Anexo H).

La herramienta PointLabeler permite la visualización de la nube de puntos en un espacio tridimensional sobre el que se puede etiquetar usando dos tipos de herramientas.

La primera permite usar una "brocha"que permite colorear los puntos con el ratón. La otra permite la creación de un área que abarca los puntos que se desea etiquetar (Figura 3.3). Como estamos etiquetando un entorno tridimensional como si fuera un lienzo bidimensional, hace falta seleccionar una sub-área de la escena que queremos etiquetar para evitar el etiquetado de puntos a los que no les corresponde la etiqueta usada. PointLabeler permite crear un plano que divide la nube de puntos para poder etiquetar la nube de una forma más eficiente.

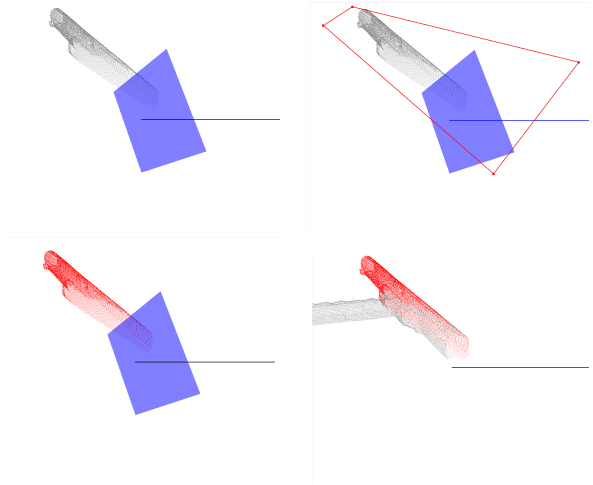


Figura 3.3: Etiquetado sencillo

Una vez etiquetada la nube, se genera un fichero con extensión .label. El fichero .label es un fichero binario que guarda la información de las etiquetas que han sido asignadas a cada punto. Con este fichero y la nube de puntos diezmada se debe crear una nueva nube de puntos que contenga la información del etiquetado usando el color en RGB de cada punto. Para realizar esta tarea se desarrolló un script en C++ que creará la nube de puntos etiquetada (Figura 3.4) usando el fichero .label y la nube diezmada (Véase Anexo I).

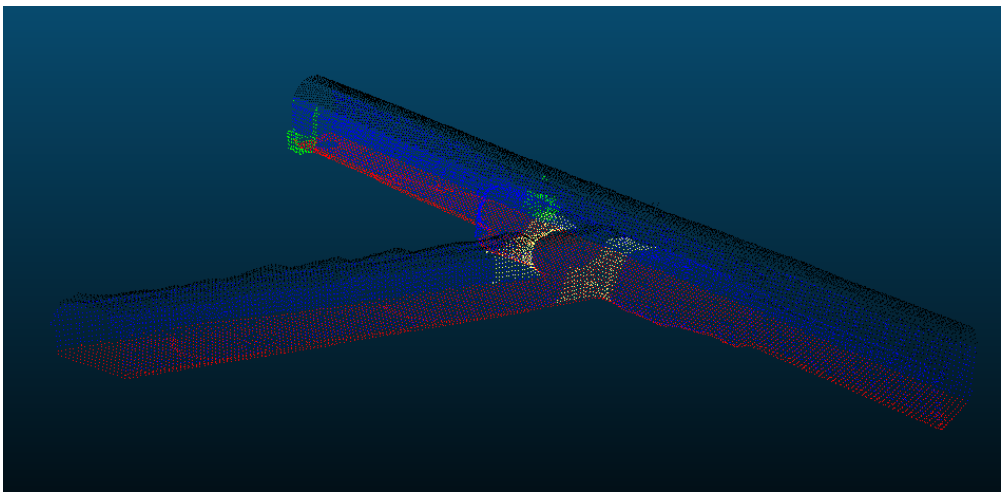


Figura 3.4: Nube con etiquetado

Para el etiquetado de la nube se han elegido las siguientes etiquetas:

- Pared: Color Azul RGB: (0, 0, 255)
- Suelo: Color Rojo RGB: (255, 0, 0)
- Cruce: Color Amarillo RGB: (255, 255, 128)
- Apeadero: Color Verde RGB: (0, 255, 0)
- Techo: Color Negro RGB: (0, 0, 0)

El motivo por el que se ha elegido este etiquetado es para que el algoritmo sea capaz de identificar y diferenciar los cruces de las galerías laterales con el corredor central, otorgándole información de un posible giro del robot para la navegación. Además, se asignan etiquetas diferentes a los cruces y a los apeaderos para que el algoritmo no se confunde y malinterprete un apeadero como un posible giro. Por otro lado, es conveniente asignar etiquetas diferentes al suelo y a las paredes para que el algoritmo obtenga información de la estructura y la geometría del entorno en el que se encuentra. Sobre la generación de la malla etiquetada (Figura 3.5) y volviendo a hablar sobre los gráficos 3D, al ser la malla una conexión de triángulos con 3 vértices cada uno, el color del triángulo resulta de la interpolación del color de los 3 vértices del mismo(Figura 3.7). Esta interpolación ocasiona una nueva etiqueta(Figura 3.6) no asignada a la cual no se le da importancia debido a que estas nuevas etiquetas se encuentran en un porcentaje ínfimo de la malla.

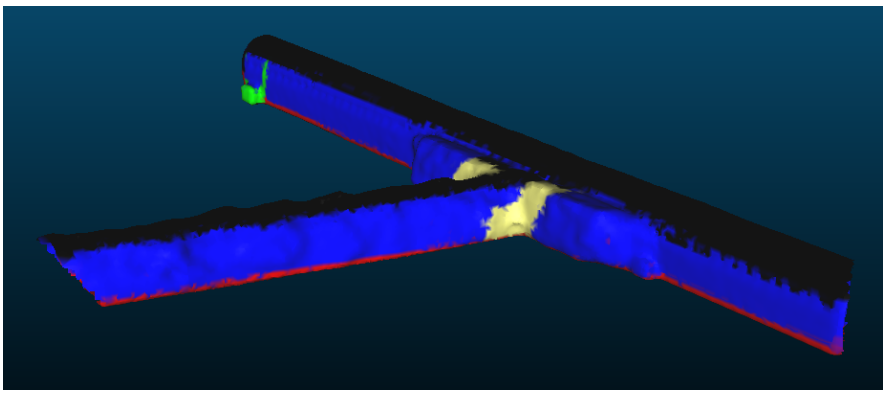


Figura 3.5: Malla tridimensional con etiquetado

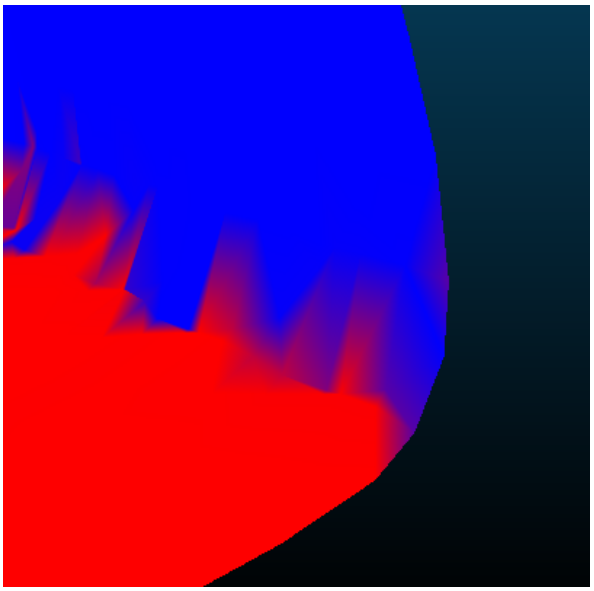


Figura 3.6: Etiqueta interpolada del color Rojo y Azul

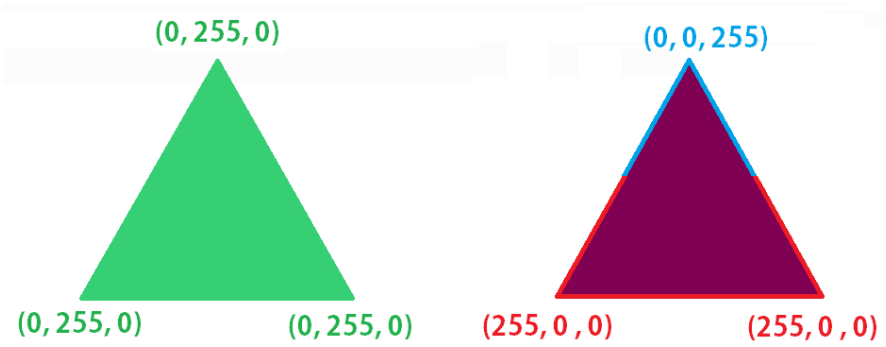


Figura 3.7: Interpolación del color de los triángulos 3D en función del color de los puntos de la nube

Capítulo 4

Generación de trayectorias y simulación

4.1. Objetivo

El objetivo es desarrollar un método que permita generar trayectorias de forma automática para la simulación de un escáner láser 3D que las seguirá. Además, el objetivo es introducir en un simulador la malla tridimensional creada a partir de la nube de puntos que representa nuestra escena. Una vez cumplidos estos objetivos, se busca automatizar el proceso de generación de trayectorias y la simulación para obtener datos etiquetados para el entrenamiento de algoritmos.

4.2. Simulador

El simulador elegido para llevar a cabo la simulación es Blensor [1], una versión de Blender 2.79 capaz de simular un escáner láser 3D. Blender es un software de código abierto para creación de gráficos por computador, animación, modelado 3D y renderizado. Las herramientas que ofrece para el modelado 3D y simulación cumplen con el objetivo de llevar a cabo simulaciones de trayectorias generadas de manera automática, seguidas por un escáner láser 3D que recogerá datos de la escena modelada en 3D.

El simulador permite la simulación de múltiples tipos de escáneres láser, tales como Generic LiDAR, Kinect, Deathmap, etc. El utilizado para llevar a cabo las simulaciones del proyecto es el escáner láser Velodyne HDL. Velodyne es una de las empresas líder en el desarrollo y fabricación de sensores LiDAR utilizados en aplicaciones de percepción y mapeo en 3D. Blensor ofrece dos modelos de escáner láser Velodyne:

1. Velodyne HDL-64E2: Este modelo emite pulsos láser en 64 planos horizontales diferentes y en hasta 360^{o} . Estas características le aportan una mayor capacidad

de obtención de datos y reconocimiento de la escena que rodea al escáner. Sin embargo, el emitir pulsos en tantos planos puede suponer una carga para la simulación y aumentar el tiempo de cómputo.

2. Velodyne HDL-32E: Este modelo emite pulsos láser en 32 planos horizontales diferentes y en hasta 360º. Posee menos capacidad de obtención de datos y de reconocimiento de la escena que rodea al escáner. Sin embargo, ofrece una carga computacional menor, siendo la mitad de pulsos a ser emitidos.

Estos modelos poseen además una serie de características modificables desde el propio simulador, tales como la resolución, la velocidad de rotación o la distancia máxima de emisión de los pulsos.

Para el desarrollo del proyecto se buscaba crear un modelo de escáner láser (Figura 4.1) Velodyne personalizado, que permitiera modificar todas las características del escáner. La capacidad de crear un escáner láser personalizado permite la creación de un escáner lo más parecido al escáner láser real que se usará en los dispositivos embarcados en el vehículo para la navegación autónoma.

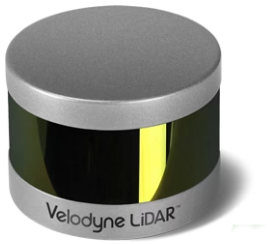


Figura 4.1: Velodyne PUCK LITE

El escáner creado para el desarrollo del proyecto es una versión del Velodyne, la principal diferencia es que este escáner emitiría pulsos láser en 16 planos horizontales. Siendo esta una opción perfecta para la experimentación con el simulador debido al bajo coste computacional.

Para la creación del modelo de escáner láser personalizado hubo que analizar e investigar el código fuente del software del simulador y realizar las modificaciones necesarias. (Véase Anexo C).

Blender ofrece la herramienta de creación de caminos o trayectorias mediante puntos tridimensionales. Dicha trayectoria se le puede asignar al escáner láser para que a la hora de ejecutar la simulación el escáner siga dicha trayectoria, la misma. Para el objetivo del proyecto se desarrolló un script en Python que lea un fichero con la información de puntos tridimensionales que forman una trayectoria y genere el camino equivalente en el simulador (Véase Anexo D).

A la hora de cargar la malla creada se observó que Blender ofrece dos formatos para importar la malla: *Stanford* (.ply) y *Wavefront* (.obj). Consultando foros del software se decidió que el formato .ply era el adecuado debido a que dicho formato, a diferencia de .obj, guarda la información del color de los vértices. Esta característica es fundamental en el desarrollo del proyecto, ya que posibilita la simulación sobre mallas con significado semántico obtenido mediante el etiquetado. Al cargar la malla se observó que estaba situada en una localización alejada del origen. Esto es debido a estareferenciada a las coordenadas de los puntos de la nube original. Tener la malla alejada del origen supone obtener resultados de las simulaciones con información de puntos que no concordarían con la realidad. Para solucionar este problema se realizó un añadido al script de traducción a formato KITTI para referenciar la nube entera al origen de coordenadas, eligiendo uno de los puntos como origen. Otro problema fue la orientación de la misma, ya que no se puede conocer la orientación de la nube de puntos. Sin embargo, en el simulador se puede cambiar la orientación de los objetos y se modificó la orientación de la malla para que apuntara hacia el eje X positivo de avance del robot.

Realizando simulaciones experimentales para comprobar el correcto funcionamiento de la trayectoria y del escáner, se observó que el simulador no guarda la información relativa a su localización global. La localización del escáner es información crucial dentro de los datos de entrenamiento de algoritmos de navegación autónoma, ya que informa sobre la posición en la escena en la que se realizó el escaneo de los datos de entrenamiento. Para obtener la información de la localización del escáner hubo que analizar el código fuente relacionado con la simulación del escáner y modificarlo para que creará un fichero con dicha información. (Véase Anexo E).

Para comprender los resultados generados de una simulación, se experimentó con un escáner que no seguía ninguna trayectoria y apuntaba hacia una de las paredes de la malla tridimensional. El escáner emitiría pulsos láser únicamente en el rango de 0^{o} a 1^{o}. Como resultado de esta simulación experimental se obtuvo una ristra de 320 puntos. Este conjunto reducido de datos facilitó la confirmación del correcto funcionamiento del escáner.

La documentación del simulador [1] explica cómo guarda los resultados del escáner. El simulador guarda un fichero en formato numpy por cada uno de los escaneos que se realizaron durante la simulación. Para facilitar la consulta de los resultados para posible depuración futura se desarrolló un *script* en Python que leyera los ficheros de los resultados y los tradujera a un formato legible (Véase Anexo F).

Se observó que las coordenadas de los puntos obtenidos del escáner eran coordenadas globales, mientras que los puntos tridimensionales necesarios para el entrenamiento de

algoritmos de navegación autónoma tienen que ser coordenadas locales al escáner. Con las coordenadas locales al escáner se extrae información de la posición del entorno relativa al escáner; por ejemplo, si los puntos detectados de una pared se encuentran a la izquierda o a la derecha del escáner. El propio simulador ofrece una opción para que los puntos calculados del escaneo sean coordenadas locales al escáner.

4.3. Experimentos

4.3.1. Trayectoria completa que recorre el túnel con un recorrido simple

Se realizó un experimento de una trayectoria que recorriese el túnel por completo (Figura 4.2) con el objetivo de comprobar el rendimiento del simulador y la relación entre el coste temporal y la complejidad de la simulación. La trayectoria que se simuló tiene una longitud de aproximadamente 126 metros y se realizaron 300 escaneos a lo largo de la simulación, obteniendo así un escaneo cada 42 centímetros. Como resultado se obtuvieron exactamente 17.337.600 puntos, durando la simulación y procesado de los resultados aproximadamente 23 minutos.

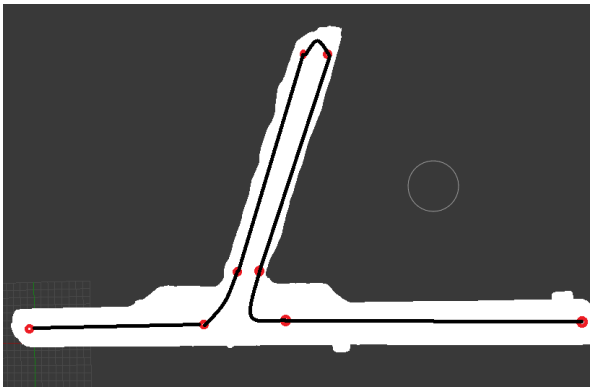


Figura 4.2: Trayectoria que recorre la sección del túnel por completo

4.3.2. Trayectoria recta sin pasar por el cruce

Se realizó un experimento de una trayectoria que fuera desde el principio del túnel hasta la salida para comprobar como afecta al costo temporal, la longitud y la cantidad de frames. La trayectoria simulada tiene una longitud aproximada de 65 metros y se realizaron 100 escaneos a lo largo de la simulación, obteniendo así un escaneo cada 65 centímetros. Como resultado se obtuvieron exactamente 5.817.600 puntos, durando la simulación y procesado de los resultados aproximadamente 6.5 minutos.

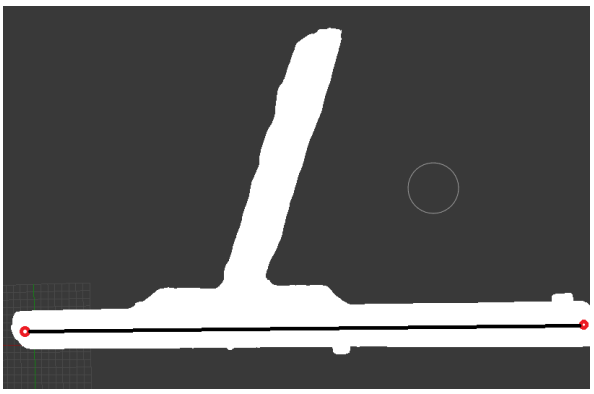


Figura 4.3: Trayectoria que recorre el túnel sin girar por el cruce

4.3.3. Trayectoria de ida y vuelta cubriendo la totalidad del túnel

Finalmente, se realizó un experimento que recorriera la totalidad del túnel por dos motivos (Figura 4.4). El primero fue para comprobar como afecta una carga mayor a la simulación y comprobar con los experimentos previos que es lo que influye en el coste temporal. El segundo fue para poder realizar evaluaciones de como se obtiene la información semántica mediante el escaneo de zonas críticas, como puede ser el cruce (sección 5.2.4), por eso la trayectoria recorre perpendicularmente el cruce donde se encuentra gran parte del etiquetado del cruce. Otra zona interesante son los apeaderos (sección 5.2.5), para ello se puso un punto de la trayectoria cerca de uno de ellos. La trayectoria simulada tiene una longitud aproximada de 221 metros y se realizaron 500 escaneos a lo largo de la simulación, obteniendo así un escaneo cada 0.41 centímetros. Como resultado se obtuvieron exactamente 28.857.600 puntos, durando la simulación y procesado de los resultados aproximadamente 31 minutos.

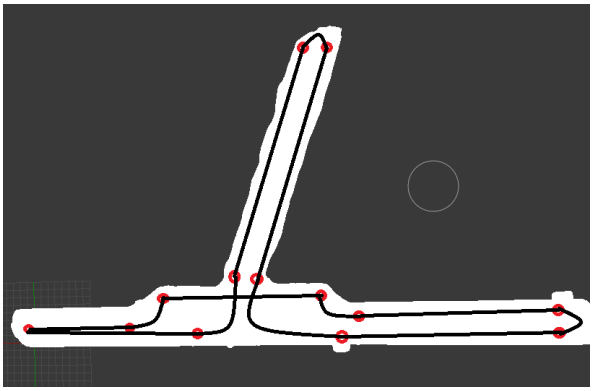


Figura 4.4: Trayectoria que recorre la totalidad del túnel y vuelve a la entrada

Como conclusión, usando este método se puede obtener grandes cantidades de datos de forma automática con el coste de creación y preparación de una malla tridimensional. Una vez preparada la malla se pueden obtener más datos de distintas trayectorias, con distinta longitud o con distinta cantidad de escaneos a lo largo de la simulación. Con relación al coste, la complejidad del recorrido, si hay más curvas o menos, no influye en el coste. Únicamente influye el número de frames que se quiere escanear. Esto es debido a que se configuró el escáner para que devolviera siempre el mismo número de puntos, tanto los de los laseres que colisionan con superficies como los que superan la distancia máxima. Esto fue necesario debido al orden que siguen los puntos escaneados, lo cual otorga la información de la localización del punto respecto al escáner.

Capítulo 5

Simulación y evaluación de resultados

5.1. Objetivo

El objetivo es realizar una serie de evaluaciones del funcionamiento de la simulación y para comprobar si se realizó correctamente la segmentación.

5.2. Evaluaciones

5.2.1. Evaluación visual

Con los escaneos obtenidos de la simulación de la trayectoria completa que recorría el túnel con un recorrido simple (sección 4.3.1) se cargaron los 300 escaneos en el visualizador 3D y se observó la nube de puntos resultante mantenía la forma de la nube de puntos original. ***Vídeo muestra de la reconstrucción***.

5.2.2. Evaluación cuantitativa

Para la evaluación cuantitativa se realizó un script en Python que leyera los escaneos obtenidos de una simulación y tradujera el color obtenido a la etiqueta más parecida. Con las etiquetas de los escaneos y la nube etiquetada de *Ground truth* se midió la cantidad de veces que el escáner reconoce una zona de la malla con la misma etiqueta que su correspondiente zona en la nube de puntos utilizada como *Ground truth* y la cantidad de veces en la que la etiqueta es distinta se calculan los aciertos, fallos y la precisión como los aciertos obtenidos entre el total de puntos escaneados. El escáner se diseñó para que obtuviera en cada escaneo la misma cantidad de puntos, esto genera los vacíos. Los vacíos son el resultado de aquellos pulsos emitidos por el escáner que superan la distancia máxima o no cumplían la distancia mínima.

5.2.3. Evaluación cuantitativa de un tramo simple

Para comprobar la precisión del escáner reconociendo paredes y el suelo se llevó a cabo una evaluación de un tramo que contuviera en su mayoría pared y suelo (Figura 5.1). El tramo evaluado consta de 10 frames, cada uno de 57.600 puntos.

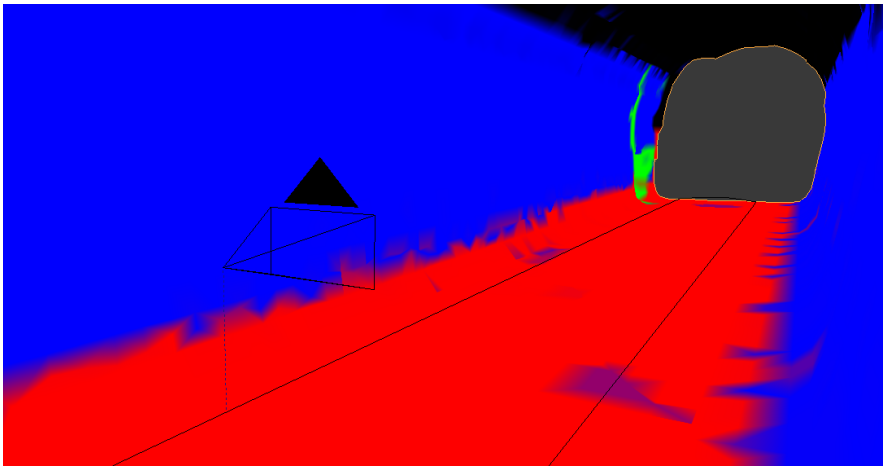


Figura 5.1: Tramo simple

	ACIERTOS	FALLOS	PRECISIÓN
PARED	406.750	23.660	94.5 %

Tabla 5.1: Resultados del escaneo de la pared

	ACIERTOS	FALLOS	PRECISIÓN
SUELO	54.902	4.618	92.2 %

Tabla 5.2: Resultados del escaneo del suelo

	ACIERTOS	FALLOS	VACÍOS	PRECISIÓN
TOTAL	509.498	58.598	7.904	93 %

Tabla 5.3: Resultados del escaneo del tramo simple

Los resultados muestran una precisión alta, siendo este un resultado bastante sólido. El escaneo ha realizado una tarea bastante efectiva. Ese pequeño error puede ser debido a zonas en las que el color se mezcla debido a que se forma esa parte de la malla con vértices de distintos colores. En la figura 5.1 se puede apreciar como en el límite entre el suelo y la pared se forma una mezcla del color rojo y azul resultando en un morado. Al calcular cuál sería la etiqueta más cercana a este color mezclado, dependiendo de la iluminación de la escena, puede ser tanto pared como suelo. Sin embargo, esto solo sucede en un 7 % del total de puntos. Sobre la etiqueta que resultaría de este color mezclado, la etiqueta dependerá del color que se acerque más en su codificación RGB. Una posible solución sería realizar un etiquetado más preciso y reducir al mínimo la cantidad de triángulos con vértices de distinto color.

5.2.4. Evaluación cuantitativa del cruce

Para comprobar la precisión del escáner reconociendo el cruce diferenciándolo de la pared se llevó a cabo una evaluación de un tramo de la trayectoria que recorría el túnel en su totalidad (sección4.3.3), la cual fue diseñada para obtener información de estos tramos de evaluación (Figura 5.2). La trayectoria recorre un tramo perpendicular al cruce donde el escáner puede detectar el cruce y diferenciarlo de una pared o apeadero. El tramo evaluado consta de 15 frames, cada uno de 57.600 puntos. Obteniendo como resultado:

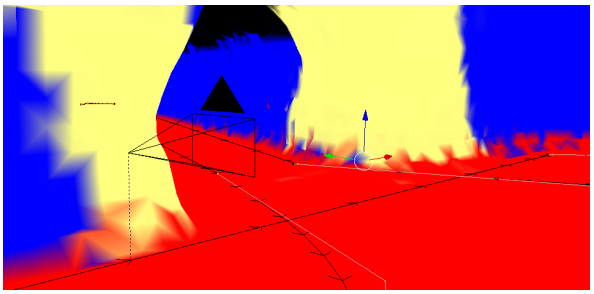


Figura 5.2: Tramo que recorre un tramo perpendicular al cruce

	ACIERTOS	FALLOS	PRECISIÓN
CRUCE	169.992	7.037	96 %

Tabla 5.4: Resultados del escaneo del cruce

	ACIERTOS	FALLOS	PRECISIÓN
PARED	468.231	40.231	92 %

Tabla 5.5: Resultados del escaneo de la pared

	ACIERTOS	FALLOS	VACÍOS	PRECISIÓN
TOTAL	847.658	63.326	10.616	93 %

Tabla 5.6: Resultados del escaneo del tramo perpendicular al cruce

En general, los resultados muestran una precisión similar al anterior experimento, mostrando un resultado sólido de 93 % de precisión. Sin embargo, el reconocimiento del cruce ha mostrado un resultado incluso mejor que el general. Este resultado es debido a la elección de la etiqueta, al ser un amarillo muy cercano al blanco (255, 255, 128) se aleja mucho del color de la pared (0, 255, 0) es por ello que no está afectado por efectos de la iluminación de la escena. Por otro lado, la precisión del reconocimiento de las paredes se mantiene con respecto al experimento anterior, esto debido a que en el tramo sigue presente el suelo por razones obvias y ocurre igual que en el experimento anterior.

5.2.5. Evaluación cuantitativa de un apeadero

Para comprobar la precisión del escáner reconociendo un apeadero diferenciándolo de la pared se llevó a cabo una evaluación de un tramo de la trayectoria que recorría el túnel en su totalidad, la cual fue diseñada para obtener información de estos tramos de evaluación (Figura 5.3). La trayectoria recorre un tramo cercano a un apeadero donde pueda detectarlo y diferenciarlo de una pared o cruce. El tramo evaluado consta de 10 frames, cada uno de 57.600 puntos. Obteniendo como resultado:

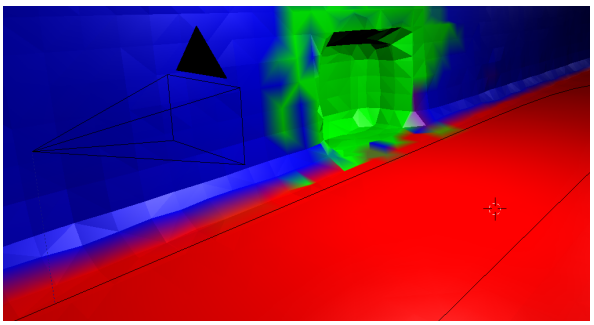


Figura 5.3: Tramo que recorre un tramo cercano a un apeadero

	ACIERTOS	FALLOS	PRECISIÓN
APEADERO	64.149	15.028	81 %

Tabla 5.7: Resultados del escaneo del apeadero

	ACIERTOS	FALLOS	PRECISIÓN
PARED	416.522	15.162	96.5 %

Tabla 5.8: Resultados del escaneo de la pared

	ACIERTOS	FALLOS	VACÍOS	PRECISIÓN
TOTAL	581.797	42.414	9.389	93 %

Tabla 5.9: Resultados del escaneo del tramo cercano a un apeadero

En general, los resultados muestran una precisión similar al anterior experimento, mostrando un resultado sólido de 93 % de precisión. Sin embargo, el reconocimiento del apeadero ha bajado considerablemente, comparándolo con anteriores resultados. Esto es debido a que un apeadero presenta cuatro zonas en las que el color se mezcla con la pared y el suelo, a diferencia del cruce, cuyo etiquetado fue pensado para que ocupara la pared en su totalidad. Este etiquetado no se podía repetir con el apeadero debido a su tamaño, ocupando este un tercio de la altura de la pared.

Capítulo 6

Conclusiones

6.1. Conclusiones técnicas

El proceso de obtención de datos etiquetados de una escena es un proceso que involucra muchos tratamientos y procedimientos costosos. Dichos tratamientos son variados y de complejidad variante. Los primeros procesos, diezmado y la creación de la malla, fueron de complejidad sencilla y más de familiarización con los entornos 3D. Por otro lado, la simulación llevo la complejidad a exponentes diferentes, familiarizarse con una herramienta Open Source de documentación escasa y comprender lo que necesitas que el simulador ofrezca y llegar a ponerlo en marcha fue una labor ardua, pero gratificante. El analizar el código de una herramienta Open Source, comprender su funcionamiento y modificarlo para que se adapte a las necesidades del proyecto fue retador, pero enriquecedor, además de ser la tarea que más horas de trabajo abarcó. La tarea restante, el etiquetado, requirió el uso de otra herramienta Open Source, pero en este caso no requirió tanto tiempo, ni tratamiento de código, sino que requirió el aprendizaje de un estándar de representación de datos obtenidos por escáner y el propio aprendizaje de la herramienta.

El proyecto ha abarcado puntos bastante variados del campo de la Computación de la Informática, sobre todo el tema de los gráficos y entornos 3D, y temas como el de la Robótica simulando trayectorias de robots con escáner.

Gracias al método implementado, se ha optimizado el proceso de etiquetado de datos y la obtención de grandes conjuntos de datos, los cuales son necesarios para entrenar algoritmos de reconocimiento de escenas, de una forma más eficiente.

En mi opinión, me parece un trabajo que tiene mucha utilidad en el ámbito de la Robótica, Navegación Automática, Realidad Aumentada, Arquitectura, etc. Ha sido un trabajo gratificante, ya que me ha exigido poner en práctica los 4 años de formación y muy enriquecedor, del cual he obtenido conocimiento de campos como el de la simulación y la geometría 3D.

6.2. Trabajo futuro

Teniendo este proyecto como base, se apunta a utilizar el proyecto para entrenar una red neuronal de un algoritmo de navegación automática y comprobar sus resultados realizando pruebas de navegación usando un robot con un escáner y que con el algoritmo de navegación entrenado realice trayectorias de forma autónoma.

Otra posibilidad sería el desarrollo de una aplicación que englobe el proceso entero del proyecto, ya que para la realización del proyecto han hecho falta 5 herramientas distintas y la ejecución de scripts de Python y C++, lo cual resulta carente de usabilidad.

Capítulo 7

Bibliografía

[1] Michael Gschwandtne and Roland Kwitt. Blensor Blender Sensor Simulator. URL `https://www.blensor.org/pages/documentation.html`.

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[3] Jens Behley Cyrill Stachniss Andres Milioto, Ignacio Vizzo. Rangenet++: Fast and accurate lidar semantic segmentation. 2019.

[4] Krizhevsky et al. Imagenet classification with deep convolutional neural networks. 2012.

[5] AndrewZisserman KarenSimonyan. Very deep convolutional networks for large-scale image recognition. 2015.

[6] Sławomir Halat. Point cloud vs 3d reality mesh. URL `https://blog.virtuosity.com/point-cloud-vs-3d-reality-mesh`.

[7] MathWorks. Segmentación semántica. URL `https://es.mathworks.com/solutions/image-video-processing/semantic-segmentation.html`.

[8] Ph.D. Florent Poux. How to automate LiDAR point cloud sub-sampling with Python, 2020. URL `https://towardsdatascience.com/how-to-automate-lidar-point-cloud-processing-with-python-a027454a536c`.

[9] Karlsruhe Institute of Technology. The kitti vision benchmark suite. URL `https://www.cvlibs.net/datasets/kitti/`.

[10] Feiyu Zhao Xiyan Yin Buyun Sheng, Chenglei Zhang. A lightweight surface reconstruction method for online 3d scanning point cloud data oriented toward 3d printing. 2018.

[11] VR Workout. Blensor tutorial velodyne, 16 ene 2018. URL https://www.youtube.com/watch?v=-CfLK72nCjU&t=17s&ab_channel=VRWorkout.

[12] siusiulala. Convert color point cloud to textured mesh using meshlab, 16 sept 2013. URL https://www.youtube.com/watch?v=6wP_e37t7PI&t=236s&ab_channel=siusiulala.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Gestión del trabajo

Sobre la distribución del trabajo realizado, este comenzó a principios de Abril y se acabó a finales de Julio. Desde Abril hasta el final de las clases se siguió un ritmo no muy constante. Sin embargo, desde el final de las clases en Mayo hasta finales de Julios se siguió un ritmo constante.

A.1. Horas invertidas

Tarea	Horas invertidas
Diezmado y creación de la malla	25
Etiquetado	30
Instalación y compresión de Blensor y Blender	15
Creación de trayectorias en Blender	12
Análisis de resultados del simulador	20
Creación del modelo Velodyne de 16 planos	10
Automatización de la simulación	30
Modificaciones de Blensor	25
Conseguir información etiquetado en Blender	20
Experimentación y pruebas	50
Memoria	70
Reuniones	10
Total	317

Tabla A.1: Horas invertidas

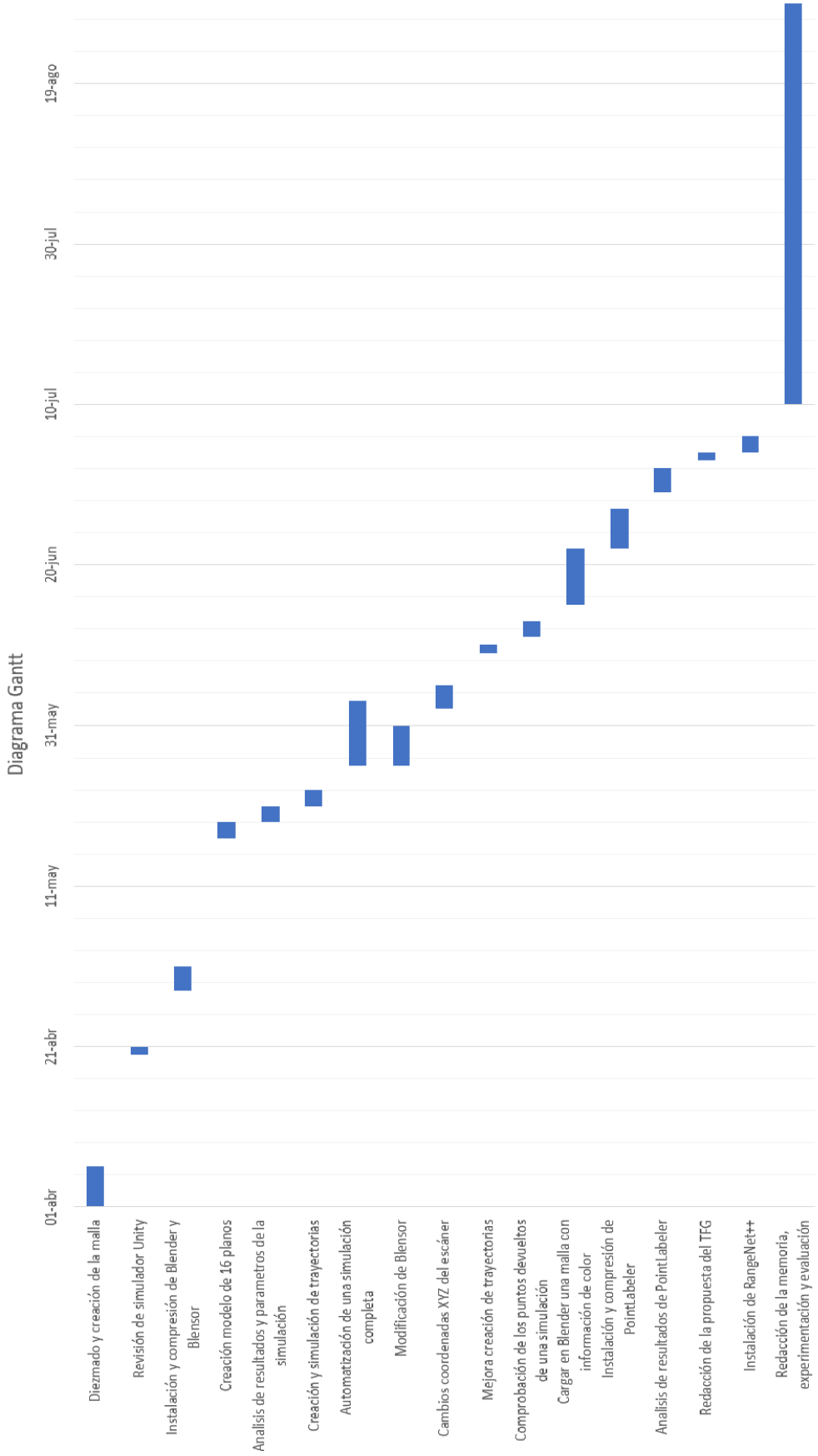


Figura A.1: Diagrama de Gantt

Anexos B

README: Proceso completo de ejecución del método

Partiendo de una nube de puntos que representa un cruce del Túnel de Somport que contiene más de 4 millones de puntos.(Figura B.1)

1. Utilizando el script de diezmado, se realiza un diezmado mediante vóxeles (Figura B.2).

   ```
   python .\voxelDiez.py 0.25 nube.las
   ```

   Donde el primer argumento es el tamaño de vóxel y el segundo es la nube de puntos en formato .las.

2. Para poder etiquetar la nube de puntos diezmada se traduce la nube de puntos al formato estandarizado KITTI (Véase Anexo H) para que se pueda etiquetar mediante la herramienta PointLabeler.

3. Una vez se tiene la nube diezmada y traducida se etiqueta mediante la herramienta PointLabeler para otorgar las siguientes etiquetas:

   - Pared: Color Azul
   - Suelo: Color Rojo
   - Cruce: Color Amarillo
   - Apeadero: Color Verde
   - Techo: Color Negro

4. Con el fichero .label que genera PointLabeler y la nube de puntos diezmada se crea la nube de puntos con la información semántica del etiquetado (Véase Anexo I) (Figura B.3).

5. Haciendo uso de CloudCompare se genera la malla tridimensional con la información del etiquetado (Figura B.4).

6. Con el objetivo de crear el material con la información del etiquetado de la malla en el simulador se genera dicha información usando la herramienta MeshLab.

7. Dado que la orientación y coordenadas de la nube de puntos original es indeterminada, se carga manualmente en el simulador para ajustarla al origen y a la orientación deseada (Véase Anexo G).

8. Una vez están preparados los preparativos para la simulación, se ejecuta el script que carga y preparar la escena para la simulación (Véase Anexo J).

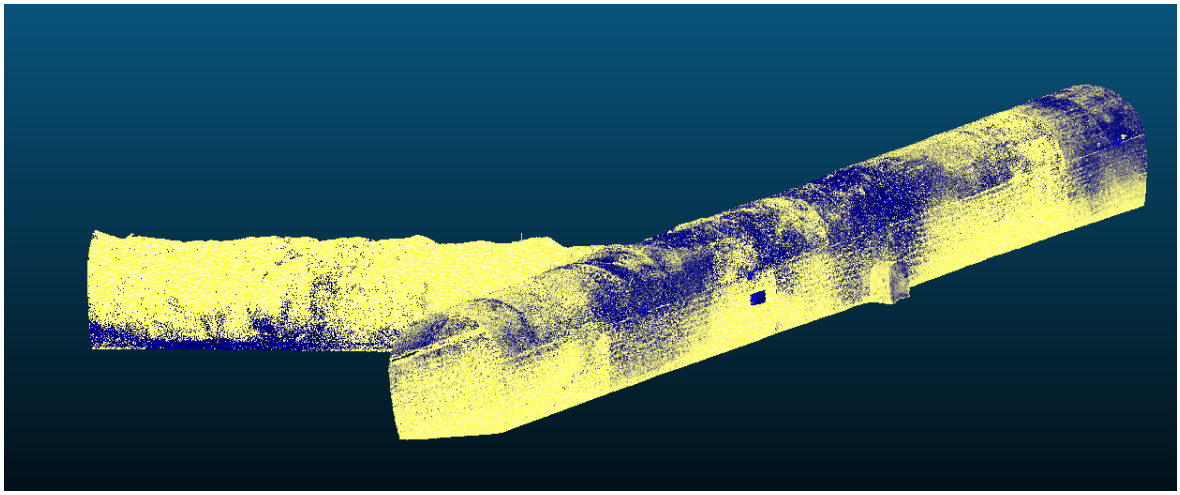


Figura B.1: Nube original del túnel

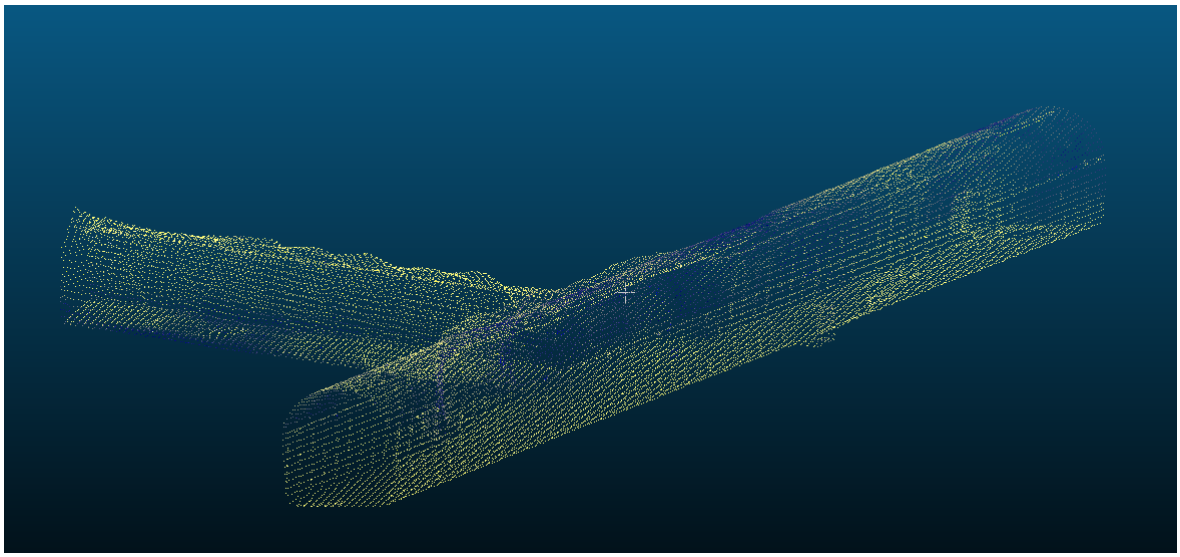


Figura B.2: Nube diezmada

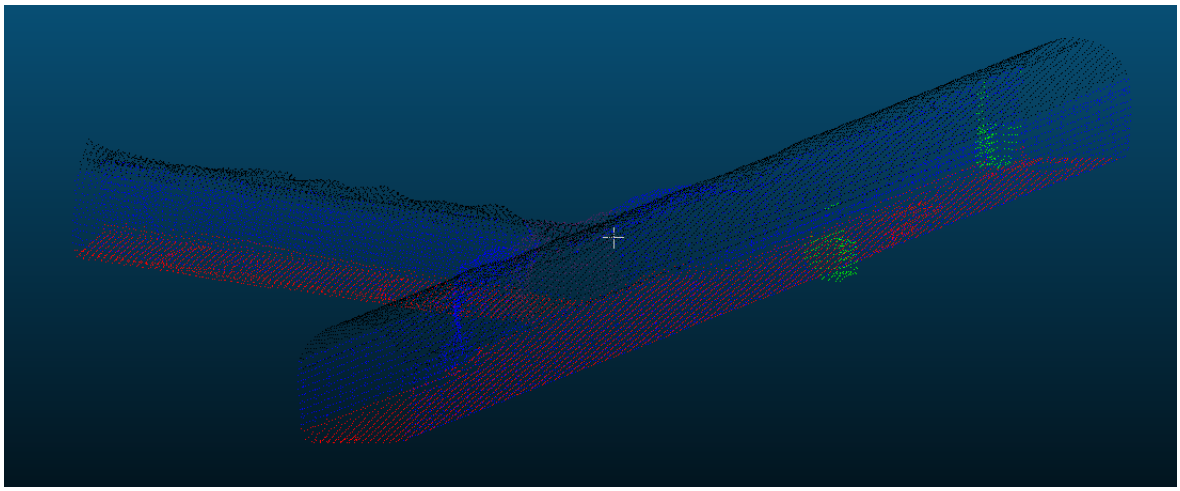


Figura B.3: Nube etiquetada

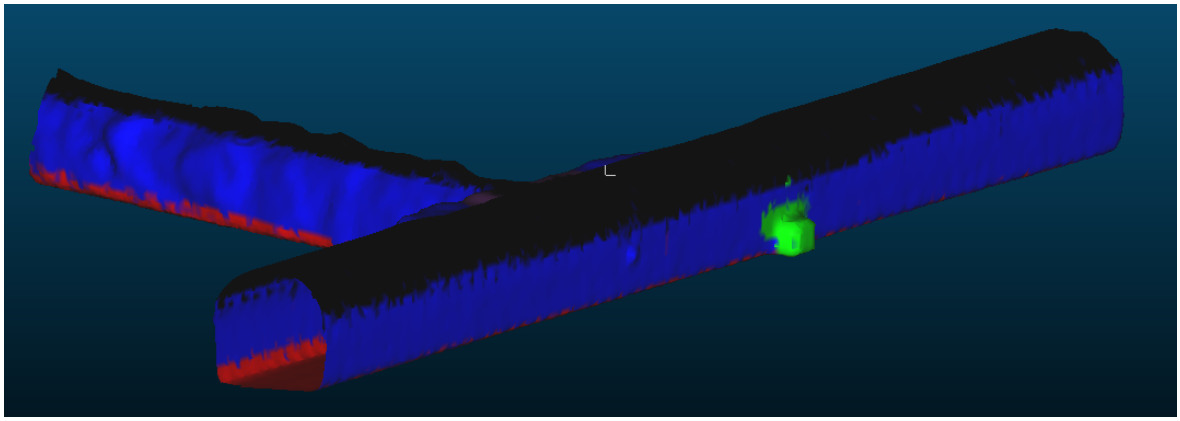


Figura B.4: Malla etiquetada

Anexos C

Creación del modelo de láser Velodyne de 16 planos

Investigando el código fuente, se encontró el fichero con el código relacionado a los láseres y los escaneos. Siendo ese Blensor-1.0.18-Blender-2.79/Winx64/2.79/scripts/addons/blensor/blendodyne.py.
En el fichero se encuentra las siguientes líneas de código:

```
laser_angles_32 = [-30.67, -9.33, -29.33, -8.00, -28.00, -6.66, -26.66,
                   -5.33, -25.33, -4.00, -24.00, -2.67, -22.67, -1.33,
                   -21.33, 0.00, -20.00, 1.33, -18.67, 2.67, -17.33,
                   4.00, -16.00, 5.33, -14.67, 6.67, -13.33, 8.00,
                   -12.00, 9.33, -10.67, 10.67]
```

Este código crea una lista que contiene los ángulos de los planos sobre los que el láser escaneará. Para la creación del modelo se añadió el siguiente código al fichero:

```
laser_angles_16 = [1, 3, 5, 7, 9, 11, 13, 15,
                   -1, -3, -5, -7, -9, -11, -13, -15]
```

Seguidamente, se cambió el código para que se pueda usar esté nuevo modelo en el simulador. Se encontraron las líneas de código que se usan para añadir los modelos al simulador:

```
parameters = {"angle_resolution":0.1728, "rotation_speed":10,"max_dist":120,
"noise_mu":0.0,"noise_sigma":0.01,"start_angle":0,"end_angle":360,
"distance_bias_noise_mu": 0, "distance_bias_noise_sigma": 0.078,
"reflectivity_distance":50,"reflectivity_limit":0.1,"reflectivity_slope":0.01,
"noise_types": [("gaussian", "Gaussian", "Gaussian distribution (mu/simga)"),
("laplace","Laplace","Laplace distribution (sigma=b)")],
"models": [(BLENSOR_VELODYNE_HDL64E2, "HDL-64E2", "HDL-64E2"),
(BLENSOR_VELODYNE_HDL32E, "HDL-32E", "HDL-32E")]}
```

En la componente "models"se ven reflejados los 2 modelos que tiene el simulador, así que se le añade la siguiente línea de código para que se pueda seleccionar el modelo creado:

```
parameters = {"angle_resolution":0.1728, "rotation_speed":10,"max_dist":120,
"noise_mu":0.0,"noise_sigma":0.01,"start_angle":0,"end_angle":360,
"distance_bias_noise_mu": 0, "distance_bias_noise_sigma": 0.078,
"reflectivity_distance":50,"reflectivity_limit":0.1,"reflectivity_slope":0.01,
"noise_types": [("gaussian", "Gaussian", "Gaussian distribution (mu/simga)"),
("laplace","Laplace","Laplace distribution (sigma=b)")],
"models": [(BLENSOR_VELODYNE_HDL64E2, "HDL-64E2", "HDL-64E2"),
(BLENSOR_VELODYNE_HDL32E, "HDL-32E", "HDL-32E"),
(BLENSOR_VELODYNE_HDL16E, "HDL-16E", "HDL-16E")]}
```

Así, el simulador permite la selección del modelo creado en el selector de modelos.(Figura C.1)
El último paso es cambiar el código para que cuando se escoja el modelo 'HDL-16E' el código del escáner use los ángulos definidos anteriormente. El código del escaneo reside en una función llamada:

```
def scan_advanced(scanner_object, rotation_speed = 10.0,x_off = 0,frame_start = -1,
    changeAxis = False, z_off = 0, y_off = 0, write = True, local = False,
    simulation_fps=24, angle_resolution = 0.1, max_distance = 120,
    evd_file=None,noise_mu=0.0, noise_sigma=0.03, start_angle = 0.0,
    end_angle = 360.0, evd_last_scan=True, add_blender_mesh = False,
    add_noisy_blender_mesh = False, frame_time = (1.0 / 24.0),
    simulation_time = 0.0, world_transformation=Matrix()):

    scanner_angles = laser_angles
    scanner_noise = laser_noise
    if scanner_object.velodyne_model == BLENSOR_VELODYNE_HDL32E:
      scanner_angles = laser_angles_32
```

Donde la variable scanner angles contiene los ángulos que definen el modelo. El código hace una comprobación del atributo scanner object.velodyne model que es modelo elegido en el simulador(Figura C.1). Para agregar el modelo creado se añaden las siguientes líneas de código.

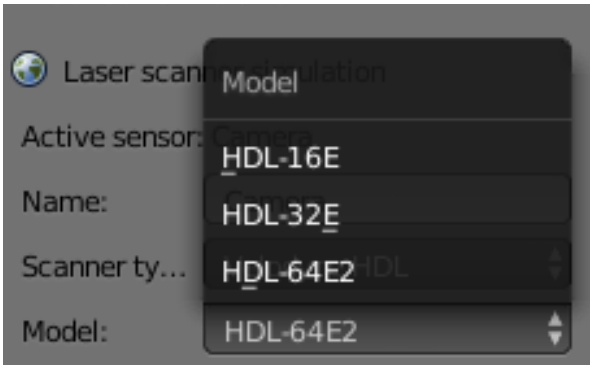


Figura C.1: Selector de modelos

```
def scan_advanced(scanner_object, rotation_speed = 10.0,x_off = 0,frame_start = -1,
    changeAxis = False, z_off = 0, y_off = 0, write = True, local = False,
    simulation_fps=24, angle_resolution = 0.1, max_distance = 120,
    evd_file=None,noise_mu=0.0, noise_sigma=0.03, start_angle = 0.0,
    end_angle = 360.0, evd_last_scan=True, add_blender_mesh = False,
    add_noisy_blender_mesh = False, frame_time = (1.0 / 24.0),
    simulation_time = 0.0, world_transformation=Matrix()):

    scanner_angles = laser_angles
    scanner_noise = laser_noise
    if scanner_object.velodyne_model == BLENSOR_VELODYNE_HDL32E:
      scanner_angles = laser_angles_32
    elif scanner_object.velodyne_model == BLENSOR_VELODYNE_HDL16E:
      scanner_angles = laser_angles_16
```

Con estas variaciones el simulador ya realiza las simulaciones con el modelo creado.

Anexos D

Formato y creación de trayectorias

Para automatizar la creación de trayectorias se creó un script que leyera un fichero JSON con la información de puntos guía tridimensionales que forman la trayectoria. El formato es el siguiente:

```
{
        "tipo": "recta",
        "puntos": [
            {
                "x": 1.0,
                "y": 2.0,
                "z": 0.5,
                "id": 1
            },
            {
                "x": 4.0,
                "y": 2.0,
                "z": 0.5,
                "id": 2
            }
        ]
    },
    {
        "tipo": "recta",
        "puntos": [
            {
                "x": 6.0,
```

```
            "y": 4.0,
            "z": 0.5,
            "id": 3,
            "r": 4,
            "tipo": 1
        },
        {
            "x": 10.0,
            "y": 8.0,
            "z": 0.5,
            "id": 4
        }
    ]
}
```

Es una lista en JSON, donde cada elemento es un par de puntos que definen una recta, el valor 'r' que tiene el primer punto del segundo par define el radio de la curva que forma la unión entre ese punto y la recta anterior en la lista. (Figura D.1)

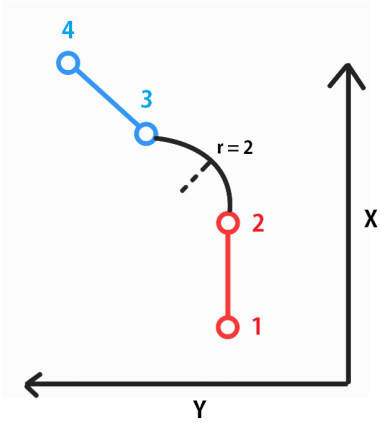


Figura D.1: Trayectoria generada con 4 puntos

Anexos E

Obtención de la odometría del láser

Para obtener la odometría del láser hace falta escribir en un fichero auxiliar la localización global del láser en cada frame. Para ello, hay que añadir código al fichero donde se encuentra el código del escaneo:

```
Blensor-1.0.18-Blender-
    2.79/Winx64/2.79/scripts/addons/blensor/blendodyne.py.
```

En la función 'scan advanced' se añade el siguiente código al final de la función:

```
fichero = open("camLoc/camera_pos.txt", "a+")
buff = ""
euler = wt.to_euler()
buff += str(frame_start)+"_: XYZ:(" + str(round(wt[0][3], 3))+ ", "
    + str(round(wt[1][3], 3)) + ", " + str(round(wt[2][3], 3)) + ")
    XYZrot(" + str(round(euler.x - x_off, 3)) + ", " +
    str(round(euler.y - y_off, 3)) + ", "
    + str(round(euler.z -  z_off, 3)) + ")\n"
fichero.write(buff)
fichero.close()
```

El cual obtiene la posición y la rotación del láser en función a la matriz global del láser que guarda el simulador. Un ejemplo de odometría sería la siguiente:

```
1_: XYZ: (1.0, 2.0, 1.5) XYZrot(0.0, 0.0, 0.0)
2_: XYZ: (2.142, 2.0, 1.5) XYZrot(0.0, 0.0, 0.0)
```

Donde XYZ son las coordenadas del escáner y XYZrot son los angulos de rotación de euler que indican la orientación del escáner.

Anexos F

Procesamiento de los ficheros resultantes de la simulación

La documentación de Blensor explica que para leer los ficheros mediante scripts, primero hace falta guardar los escaneos especificando el formato de los mismos, que en este caso sería numpy, formato que la librería del mismo nombre de Python lee fácilmente. Para ello, a la hora de realizar la simulación y elegir el directorio destino del resultado hay que especificar en el nombre del fichero la extensión .numpy.
Para el procesamiento de los ficheros se creó un script en Python que lee cada fichero numpy, obteniendo así una matriz de Npuntosx15. Es decir, una línea por cada punto conteniendo los siguientes atributos:

```
timestamp, yaw, pitch, distance, noise, X, Y, Z, Xnoise, Ynoise,
Znoise, id, R, G, B, idx
```

Dando como resultado un fichero scan.txt donde cada línea es el resultado del escaneo de cada frame con el siguiente formato:

```
timestamp;angle_min;angle_max;scanres;range_min;
range_max;x,y,z...|r,g,b...
```

Anexos G

Tratamiento de la malla para la simulación

Para guardar la información de color la malla hay que guardar la malla en formato .ply que contiene información del color de los vértices.
Con esa información de color de los vértices hay que generar la información de los colores de los puntos para que el simulador pueda utilizarla en la creación del material de la malla. Para ello se usa la herramienta MeshLab.
Las coordenadas de la nube de puntos original pueden no estar centradas en el origen, para ello en CloudCompare hay que seleccionar el punto que se desea que sea el origen de la nube y anotarlas (Figura G.1) y traducir la nube, restando a cada punto las coordenadas del origen.

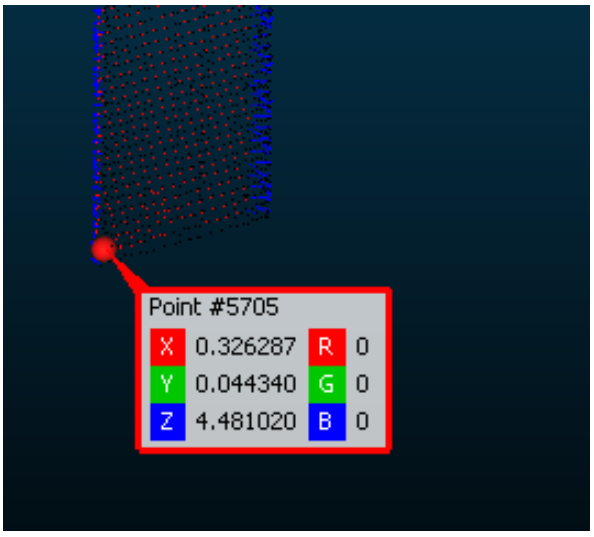
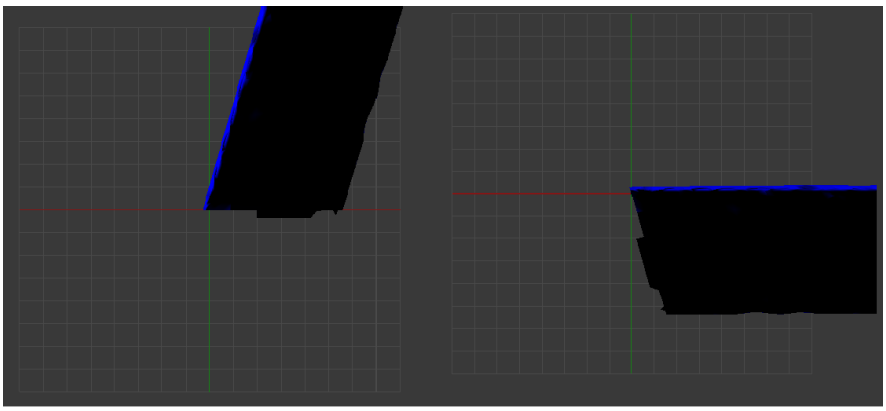


Figura G.1: Coordenadas del punto origen

```
xOrigen = 0.32
yOrigen = 0.04
zOrigen = 4.48

for punto in nube:
    punto.x = punto.x - xOrigen
    punto.y = punto.y - yOrigen
    punto.z = punto.z - zOrigen
```

Para el desarrollo del proyecto es necesario que la malla esté orientada en el eje X positivo para que el avance del escáner signifique un avance en el eje X, y que el desplazamiento a izquierda o derecha quede reflejado en el eje Y. Sin embargo, la malla al ser importada en el simulador se le da una orientación por defecto que no coincide con las necesidades. Para solucionar el problema se orienta la malla como se desea y se anota la rotación aplicada en los ejes rotativos (Figura G.2).



(a) **(b)**

Figura G.2: (a) Orientación por defecto de la malla al ser importada; (b) Orientación corregida de la malla)

Anexos H

Formato KITTI

La herramienta PointLabeler requiere de la siguiente estructura de los datos:

```
-----labeler
    |------velodyne/
    |------labels/
    |------ calib.txt
    |------ poses.txt
```

- La carpeta velodyne debe contener la nube de puntos a etiquetar en formato KITTI
- La carpeta labels contiene los resultados del etiquetado
- calib.txt y poses.txt, contienen como debe estar colocada la cámara del visualizador del programa, por lo que basta con poner datos por defecto.

Siguiendo la guía que se encuentra en el repositorio de PointLabeler se desarrolla un script que traduzca la nube de puntos a formato KITTI. El script simplemente lee la nube de puntos y los escribe en binario en un fichero binario. El código además aprovecha y realiza el tratamiento de mover la nube al origen deseado.

H.1. Uso del script

El comando para ejecutar el script es el siguiente:

```
python .\translatePC.py small.txt 0 0 0
```

Donde el primer argumento es la nube en formato txt con el siguiente formato:

```
X Y Z R G B
```

y los demás argumentos son el desplazamiento de los puntos al origen.

Anexos I

Creación de la nube etiquetada usando el fichero .label de PointLabeler y la nube diezmada

Para entender como era el formato del fichero .label hizo falta explorar los ficheros fuentes del programa, donde se encontró el siguiente código en C++:

```
std::vector<uint32_t> labels;
std::ifstream in("label/small.label", std::ios::binary);
if (!in.is_open()) {
    std::cerr << "Unable to open label file. " << std::endl;
    return 0;
}

labels.clear();

in.seekg(0, std::ios::end);
uint32_t num_points = in.tellg() / (sizeof(uint32_t));
in.seekg(0, std::ios::beg);

labels.resize(num_points);
in.read((char*)&labels[0], num_points * sizeof(uint32_t));
```

En el código se aprecia que lo único que hace es leer el fichero de manera binaria, leyendo enteros de 32 bits, los cuales enumeran las etiquetas del programa. Realizando pruebas se identificó las etiquetas usadas en el programa:

- 0: NEGRO. Techo
- 1: ROJO. Suelo
- 10: AZUL: Pared
- 15: MORADO: Giro
- 70: VERDE: Apeadero

El script simplemente lee la nube y recorriendo los puntos y simultáneamente las etiquetas del fichero .label se escribe cada punto con el color que indica la etiqueta.

I.1. Uso del script

El comando para ejecutar el script es el siguiente:

```
g++ labelParser.cpp -o labelParser para compilacion
./labelParser etiq.label nube.txt 0 0 0
```

Donde el primer argumento es el fichero de etiquetas que genera PointLabeler, el segundo es la nube de puntos con el siguiente formato:

```
X Y Z R G B
```

Los 3 restantes argumentos son el desplazamiento que se desee dar a todos los puntos de la nube para colocarlos en el origen deseado.

Anexos J

Simulación automática

El script simularTrayectoria.py encontrado en:

`código/sensor/Blensor-1.0.18-Blender-2.79-Winx64/py_scripts`

crea la escena de la simulación y genera la trayectoria dada en un fichero con el siguiente formato descrito en el Anexo D.

J.1. Uso del script

Para poder ejecutar es necesario seguir esta estructura:

```
-----Blensor-1.0.18-Blender-2.79-Winx64
    |
    |------camLoc           (Carpeta contenedora del fichero de odometría)
    |------mesh             (Carpeta contenedora de la malla de la escena a
                                escanear)
    |------scans            (Carpeta contenedora del resultado de la
                                simulación)
    |------ py scripts      (Carpeta contenedora del script)
    |------ trayectorias    (Carpeta contenedora de las trayectorias)
    |------ settings.json   (Fichero con los parametros de la simulación)
```

En el directorio Blensor-1.0.18-Blender-2.79-Winx64 se encuentra el ejecutable de blender, blender.exe, el cual puede ser ejecutado en línea de comandos de la siguiente manera:

```
./blender.exe -P ./py scripts/crearTrayectoria.py
```

para que ejecute el script python antes de abrir blender.
El fichero settings contiene los parámetros de la simulación, estos son:

- malla: Nombre del fichero que contiene la malla sobre la que se va a simular.
- mallaOffset: Desplazamiento para colocar la malla como se desee.
- mallaRotation: Orientación de la malla.
- escanerLocation: Posición del escáner respecto a la trayectoria.
- trayectoria: Nombre del fichero que contiene la trayectoria que se va a simular.
- length: Número de escaneos en los que se dividirá la simulación.
- start_frame: Escaneo por el que se quiere empezar la simulación, por si se desea realizar una simulación sobre una sub-división de la trayectoria.
- end_frame: Escaneo por el que se quiere terminar la simulación.
- angle_max y angle_min: Rango de los angulos sobre los que el escáner lanzara los rayos en cada escaneo.
- scanres: Resolución del escáner, es decir, cada cuantos ángulos emitirá el escáner los pulsos láser.
- range_min y range_max: Distancia mínima y máxima de emisión de pulsos láser.
- model: Modelo que se desea usar en la simulación.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Juan Plo Andrés,

con nº de DNI 26280142C en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Etiquetado semántico automático de una nube de puntos 3D de un entorno, para mejorar el reconocimiento semántico de una escena,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 31 de Agosto de 2023

Firmado por Juan Plo Andrés el día 31/08/2023 con un certificado emitido por AC FNMT Usuarios

Fdo: