Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

SISTEMAS ELECTRÓNICOS AVANZADOS

TRABAJO FIN DE MÁSTER

DETECCIÓN DE OBJETOS 3D MEDIANTE LIDAR EN SISTEMAS EMBEBIDOS

Estudiante *Díaz Prieto, Asier*
Director/Directora *Ibarra Basabe, Edorta*
Departamento *Tecnología Electrónica*
Curso académico *2021-2022*

Bilbao, 26 de junio del 2022

Resumen

Los avances en la Inteligencia Artificial y a la disponibilidad de tecnologías de sensorización más recientes, como el LiDAR, posibilitan que el desarrollo de los sistemas autónomos esté en boga. Este Trabajo Fin de Máster desarrollado en la empresa Ikerlan, está enmarcado dentro de una de las principales ramas de los sistemas autónomos, como es la percepción. El objetivo del trabajo es desarrollar un sistema de detección de objetos 3D, concretamente de personas, utilizando únicamente los datos proporcionados por el LiDAR Livox Mid-70, y el *middleware* ROS. Además, este sistema se ha integrado en una plataforma embebida para aproximarse al caso de uso de los vehículos o robots autónomos. En el desarrollo del trabajo, se han estudiado varios sistemas de percepción con el objetivo de descubrir el más adecuado para el Livox Mid-70, donde los sistemas basados en *Deep Learning* han mostrado un rendimiento superior frente a los sistemas clásicos.

Palabras clave: LiDAR, ROS, *Deep Learning*, detección 3D, percepción, Livox.

Laburpena

Adimen Artifizialean egindako aurrerapenei eta sentsorizazio teknologia berrien eskuragarritasunari esker (LiDARa, adibidez), sistema autonomoen garapena modan egotea ahalbidetzen dute. Ikerlan enpresan burututako Master Amaierako Lan hau, sistema autonomoen adar nagusietako batean kokatzen da, pertzepzioan, alegia. Lanaren helburua 3Dko objektuak, zehazki pertsonak, detektatzen dituen sistema bat garatzea da soilik, Livox Mid-70 LiDARetik eskuratutako datuak eta ROS *middleware*-a erabiliz. Gainera, sistema hau ibilgailu eta robot autonomoen erabilerara hubiltzeko plataforma txertatu batean integratu da. Lanaren garapenean, zenbait pertzepzio sistema aztertu dira Livox Mid-70erako egokiena zein den jakiteko, non *Deep Learning*-ean oinarritutako sistemek errendimendu handiagoa erakutsi duten sistema klakikoen aldean.

Hitz gakoak: LiDAR, ROS, *Deep Learning*, 3D detekzioa, pertzepzioa, Livox.

Abstract

Due to advances in Artificial Intelligence and the availability of more recent sensorisation technologies, such as LiDAR, the development of autonomous systems is in vogue. This Master's Thesis, developed in the company Ikerlan, is framed within one of the main branches of autonomous systems, such as perception. The aim of the present work is to develop a 3D object, specifically people, detection system using only the data provided by the LiDAR Livox Mid-70 and the ROS middleware. Furthermore, this system has been integrated into an embedded platform to approach the use case of autonomous vehicles or robots. In the development of this work, several perception systems have been studied with the aim of discovering the most suitable for the Livox Mid-70, where systems based on Deep Learning have shown superior performance compared to classical systems.

Key words: LiDAR, ROS, *Deep Learning*, 3D detection, perception, Livox.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

- **ADS** - Automated Driving System
- **API** - Application Programming Interfaces
- **BEV** - Bird Eye View
- **BRTA** - Basque Research and Technology Alliance
- **BSD** - Berkeley Software Distribution
- **CNN** - Convolutional Neural Network
- **CPU** - Central Processing Unit
- **DDS** - Data Distribution Service
- **FCN** - Fully Connected Network
- **FN** - False Negative
- **FOV** - Field of View
- **FP** - False Positive
- **FPS** - FurthestPoint-Sampling
- **FV** - Frontal View
- **GPU** - Graphics Processing Unit
- **IDL** - Interface Definition Language
- **IoU** - Intersection over Union
- **KITTI** - Karlsruhe Institute of Technology and Toyota Technological Institute
- **KNN** - K Nearest Neighbors
- **LiDAR** - Light Detection and Ranging

- **NMS** - Non Maximum Suppression
- **ODD** - Operational Design Domain
- **OGRE** - Object-Oriented Graphics Rendering Engine
- **ONXX** - Open Neural Network Exchange
- **P2P** - Peer to Peer
- **PBR** - Phisically Based Rendering
- **PCD** - Point Cloud Data
- **PCL** - Point Cloud Library
- **PKW** - Predicted Keypoint Weighting
- **RANSAC** - Random Sampling and Consensus
- **ReLU** - Rectified Linear Unit
- **RoI** - Region of Interest
- **ROS** - Robotic Operative System
- **RPN** - Region Proposal Network
- **SAE** - Society of Automotive Engineers
- **SDF** - Simulator Description Format
- **SECOND** - Sparsely Embedded CONvolutional Detector
- **SOTA** - State of the Art
- **SSD** - Single Shot Detetor
- **STF** - Sistema Txertatu Fidagarriak
- **SWF** - Software Fidagarria
- **TP** - True Positive
- **UDP** - User Protocol Diagram
- **URDF** - Universal Robotic Description Format

- **VFE** - Voxel Feature Encoding
- **VSA** - Voxel Set Abstraction
- **WS** - WorkStation

1 Introducción

El objetivo de los investigadores en los sistemas de Inteligencia Artificial desarrollados para el vehículo autónomo es la de seguir progresando en los sistemas autónomos. Watson y Scheidt definen un sistema autónomo como "... *sistemas que pueden cambiar su comportamiento en respuesta a eventos no anticipados mientras están operando*"[1]. Para estos sistemas es fundamental la incorporación de la inteligencia - la habilidad para percibir, procesar, recordar, aprender, y determinar los cursos de acción como resultado de la integración de estos procesos [2].

Un sistema autónomo requiere una percepción precisa del entorno que le rodea para funcionar de forma fiable. El sistema de percepción de estos sistemas autónomos transforma los datos sensoriales en información semántica que permite operar de forma autónoma. La detección de objetos es una función fundamental de este sistema de percepción, que normalmente emplea técnicas basadas en *Machine Learning* (por ejemplo, el *Deep Learning*).

1.1 Sistemas de conducción autónoma

En las últimas décadas, debido al conocimiento acumulado en la dinámica de los vehículos, los avances en la visión artificial provocados por la llegada de técnicas de *Deep Learning* y la disponibilidad de nuevas tecnologías de sensores, como el LiDAR, los sistemas de conducción autónoma (ADS) están ganando una gran popularidad [3].

Son muchas las ventajas que los ADS pueden originar. Basándose en las estadísticas de accidentes de tráfico, se concluye que aproximadamente el 94 % de los accidentes son ocasionados por fallos de los conductores, incluyendo maniobras inapropiadas y distracciones [4]. De esta forma, los sistemas de conducción autónoma tienen el potencial de salvar miles de vidas evitando los errores humanos en la conducción. Además, también pueden ocasionar que la conducción sea más eficiente, lo cual se traduce en una reducción en el consumo del combustible y, por ende, en un menor impacto adverso sobre el medio ambiente. En cuanto a las personas con problemas de movilidad que no pueden conducir por si solas, podrán disfrutar de la movilidad con el desarrollo de esta nueva tecnología.

Para analizar el avance de estos sistemas y para poder compararlos, la Sociedad de Ingenieros

de Automoción (*SAE-Society of Automotive Engineers*) ha definido seis niveles de conducción autónoma [5], tal y como se muestra en la Figura 1.1.

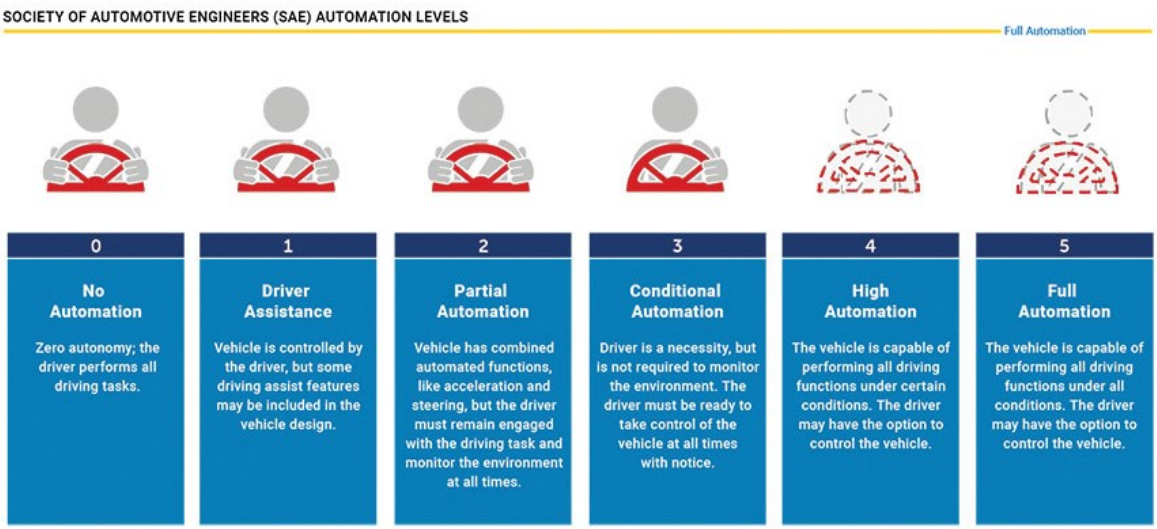


Figura 1.1: Niveles de conducción autónoma definidos por la SAE [4].

En esta taxonomía, el nivel cero indica que el vehículo no tiene ningún tipo de autonomía. Los sistemas primitivos de asistencia al conductor, como el control de crucero adaptativo, los sistemas de antibloqueo de frenos y el control de estabilidad comienzan en el nivel uno. El nivel dos corresponde a la automatización parcial, en la que se integran los sistemas de asistencia avanzados, como el frenado de emergencia o el sistema anticolisiones. Con la experiencia de la industria y el conocimiento acumulado, este nivel de automatización se ha convertido en una tecnología factible. El verdadero reto comienza a partir de este nivel [5].

El nivel tres corresponde a la automatización condicional, donde el conductor se puede centrar en otras tareas que no sean la conducción mientras no haya un estado de emergencia, porque, en este caso, tiene que responder rápidamente y estar listo para tomar el control. Además, los ADS de nivel tres únicamente funcionan en dominios de diseño operativo (ODD) limitados, como las autopistas. En los niveles cuatro y cinco no es necesaria la atención del conductor. Sin embargo, el nivel cuatro únicamente puede operar en ODDs limitados donde existen infraestructuras especiales o mapas detallados. En el caso de que se salga de estas zonas, el vehículo debe detener el viaje estacionándose automáticamente. Por último, el nivel cinco (sistema totalmente automatizado) puede operar en cualquier red de carreteras y bajo cualquier condición climatológica. Hoy en día ninguno de los vehículos que se están produciendo es apto para el cuarto o quinto nivel de conducción automatizada [5].

1.2 | Sensores y hardware

En los ADS modernos es necesaria una alta redundancia de sensores en la mayoría de las tareas para conseguir que estos sean sistemas robustos y fiables, por lo que se utilizan una amplia selección de sensores. Las unidades hardware se pueden clasificar en cinco categorías: sensores exteroceptivos para la percepción, sensores propioceptivos para las tareas de control del estado interno del vehículo, las matrices de comunicación, los actuadores y las unidades de cálculo [3].

Este proyecto se centra en los sensores exteroceptivos, los cuales se utilizan para percibir el entorno, que incluye objetos dinámicos y estáticos. A continuación, se exponen los sensores exteroceptivos más utilizados.

1.2.1 | Cámaras monoculares

Las cámaras monoculares son aquellas cámaras que únicamente tienen una sola lente, a diferencia de las cámaras estéreo o binoculares. Estas cámaras pueden percibir colores y son pasivas, es decir, no emiten ninguna señal para las mediciones. La percepción de los colores es muy importante para tareas como el reconocimiento de semáforos [3]. Además, al ser un sensor pasivo no interfiere con otros sistemas, ya que no emite ninguna señal. Sin embargo, una de las desventajas es su ausencia de información de la profundidad, la cual es necesaria para la estimación precisa del tamaño y la posición de los objetos [6]. Asimismo, las condiciones de iluminación afectan drásticamente a su rendimiento.

1.2.2 | Cámaras omnidireccionales

Las cámaras omnidireccionales se utilizan como alternativa al conjunto de cámaras para lograr una visión 2D en 360º [7]. Su uso se ha generalizado con un hardware cada vez más compacto y de mayor rendimiento. La visión panorámica es especialmente útil para aplicaciones como la navegación, la localización y la cartografía [3]. En la Figura 1.2 se muestra un modelo de este tipo de cámaras.

1.2.3 | Cámaras de eventos

Las cámaras de eventos se encuentran entre las modalidades de detección más recientes que se han utilizado en ADS. Estas cámaras registran datos de píxeles individuales de forma asíncrona respecto al estimulo visual. La salida es, por tanto, una secuencia irregular de puntos o eventos

Figura 1.2: Cámara omnidireccional Ricoh Tetha V.

desencadenados por los cambios de brillo. La principal limitación de estos sensores es el tamaño de los píxeles y la resolución de imagen [3]. En la Figura 1.3 se muestra una cámara de eventos.

Figura 1.3: Cámara de eventos DAVIS 240.

1.2.4 | Radar

El radar (Figura 1.4) es un sensor activo que emite ondas de radio que rebotan en los objetos y miden el tiempo transcurrido desde la emisión de la señal hasta que vuelve al sensor a causa de esos rebotes. Estos sensores, junto con los LiDAR, los cuales se mencionan a continuación, son muy útiles para cubrir las carencias de las cámaras. La información de profundidad, es decir, la distancia a los objetos, puede medirse eficazmente para obtener información 3D, además que no se ven afectados por las condiciones de iluminación. Los radares son una tecnología bien establecida que es ligera y rentable [7].

Figura 1.4: RADAR para vehículos autónomos.

1.2.5 | LiDAR

LiDAR, acrónimo del inglés *Light Detection and Ranging*, es un sistema de medición láser, con el que la distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada [6]. Funciona con un principio similar al del radar, pero emite ondas de luz infrarroja en lugar de ondas de radio. Las ondas de luz infrarroja rebotan en los objetos de alrededor y regresan al sensor. Mediante la repetición de este proceso millones de veces por segundo se crea un mapa 3D preciso y en tiempo real, también llamado nube de puntos. El LiDAR tiene una precisión mucho mayor que el radar por debajo de los 200 metros, aunque las condiciones meteorológicas desfavorables, como la niebla o la nieve, influyen negativamente en su rendimiento [7]. En la Figura 1.5 se puede observar un LiDAR de 360º de la marca Velodyne.

Figura 1.5: LiDAR Velodyne Puck.

2 | Contexto

Este Trabajo Fin de máster se ha desarrollado en el contexto de las prácticas en el centro tecnológico Ikerlan, que es líder en la transferencia de conocimiento y en la aportación de valor competitivo a las empresas. Ikerlan es una cooperativa miembro de la Corporación MONDRAGON y del consorcio empresarial *Basque Research and Technology Alliance* (BRTA).

El trabajo que se recoge en este documento es parte de los desarrollos tecnológicos realizados en el contexto del proyecto IKMobility, en el que se pretende desarrollar un entorno de *Smart Mobility* para vehículos autónomos. El proyecto IKMobility, es un proyecto subvencionado por el Gobierno Vasco de unos tres años de duración, en el que participan siete investigadores del equipo de Software Confiable del departamento de Sistemas Embebidos Confiables (STF). Principalmente, se trabajan dos aspectos o líneas de investigación importantes: La primera línea de investigación se enfoca en los sistemas autónomos, donde se trabaja con IA embebida y confiable y se estudian nuevas técnicas SW a aplicar, con el objetivo de aumentar el grado de automatización y autonomía de distintos sistemas. La segunda línea de investigación trata el paradigma del *Edge Computing*, cuya idea es analizar y profundizar en arquitecturas hardware/software en las que las aplicaciones se comunican en tiempo real, a la vez que se estudian metodologías y herramientas para mejorar el desarrollo.

Este Trabajo de Fin de Máster se sitúa dentro de la línea de investigación de los sistemas autónomos. Se procede a desarrollar una aplicación de detección de objetos 3D utilizando únicamente el LiDAR Livox Mid-70 como dispositivo de sensorización. Para ello y debido a la complejidad de realizar un sistema de detección de objetos 3D basado únicamente en los datos proporcionados por el LiDAR, se llevará a cabo una investigación del estado del arte de los diferentes sistemas de percepción existentes en la actualidad. De esta forma, se desea acercar este tipo de tecnologías a la empresa a la par que se desarrolla una aplicación que será embebida en una plataforma hardware.

3 Objetivos y alcance del trabajo

El objetivo principal de este proyecto es desarrollar e implementar una aplicación software de detección de objetos 3D utilizando el sensor LiDAR Livox Mid-70 en una plataforma de hardware embebida. En la actualidad, la mayoría de las aplicaciones de detección utilizan la fusión sensorial, es decir, la utilización de diferentes sensores conjuntamente como cámaras, radars, LiDARs, etc. En este caso, se pretende analizar la efectividad de realizar la detección de objetos 3D con un único LiDAR.

A continuación se definen los siguientes objetivos específicos para la consecución del objetivo principal:

1. Realizar el estado del arte relativo a la detección de objetos orientado al sistema de percepción de vehículos autónomos. Se analizarán las diferentes metodologías y herramientas utilizadas, llevando a cabo una comparación en la que se especifiquen las ventajas y desventajas. También se han incluido los diferentes sistemas de percepción de objetos 3D basados en la tecnología LiDAR en este análisis. Se han analizado los sistemas de detección clásicos y los sistemas basados en *Deep Learning*. Se examinarán las diferentes técnicas utilizadas para la elaboración de los algoritmos y se compararán en función de su eficacia para la detección de diferentes clases de objetos.

2. Aplicar varios de los sistemas del estado del arte existentes para verificar su utilidad y determinar la mejor opción para el LiDAR disponible. En el caso de que la detección no sea del todo efectiva, partiendo del sistema elegido se desarrollará una aplicación válida.

3. Desarrollar una aplicación software en la que se detecte diferentes clases de objetos o personas utilizando el sensor LiDAR Livox Mid-70.

4. Integrar la aplicación en una plataforma hardware embebida para que se pueda utilizar en diferentes aplicaciones, como por ejemplo, en los vehículos autónomos.

5. Llevar a cabo una comparativa de rendimiento utilizando diferentes plataformas hardware embebidas y estudiar la solución óptima.

6. Determinar las posibles líneas de investigación futuras relacionadas con la aplicación desarrollada y las tecnologías utilizadas.

El alcance principal del trabajo es aplicarlo en un sistema autónomo, como pueden ser los vehículos o robots. Sin embargo, al estar en una primera fase de desarrollo, se ha implementado y validado en el laboratorio, orientando el propósito del proyecto a la detección de personas ya que es un área de aplicación fundamental dentro de la detección de objetos. Este área es extensivamente utilizada en aplicaciones complejas como, vídeo vigilancia, conducción autónoma, etc.

4 | Estado de la tecnología

El desarrollo de vehículos autónomos ha experimentado un gran auge en la última década, impulsado principalmente por el aumento en las capacidades de computación derivadas de las mejoras introducidas en las unidades de procesamiento gráfico (GPU), las cuales permiten el análisis de una mayor cantidad de datos para obtener un resultado más fiable y preciso en un tiempo menor, incrementando la estabilidad y seguridad del sistema autónomo [8]. La Figura 4.1 muestra un ejemplo del auge de los vehículos autónomos.

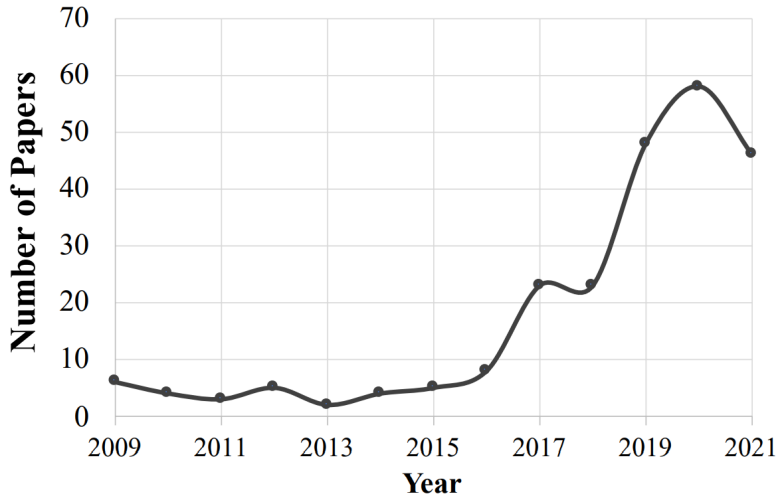


Figura 4.1: Artículos publicados en el campo de las carreras autónomas desde 2009 hasta 2021 [9].

Como se ha mencionado con anterioridad, en el mundo de la inteligencia artificial y los sistemas autónomos, una de los áreas donde más se ha investigado es en la detección de objetos. A continuación, se estudiará el estado del arte en el campo de la detección de objetos 3D con LiDAR.

4.1 | Sistemas de percepción tradicionales

En primer lugar, se han estudiado las técnicas clásicas o tradicionales utilizadas en los sistemas de percepción utilizando únicamente sensores LiDAR, las cuales se basan en la discretización y agrupamiento de los puntos. Estás técnicas no abundan, por lo que a continuación se presentan las técnicas más conocidas y utilizadas en la actualidad.

4.1.1 | Voxelización

Las nubes de puntos generadas por los LiDAR contienen una gran cantidad de puntos. Esta cantidad de puntos varía según el tipo del sensor, pero puede llegar a ser de más de 1,39 millones de puntos por segundo como es el caso del Velodyne HDL-32E [10]. El análisis de tal cantidad de datos puede llegar a ser inviable en un vehículo al tener una capacidad de computo limitada, por lo tanto, una posibilidad para superar esta limitación consiste en utilizar una técnica de voxelización.

Mediante la voxelización se transforma la nube de puntos sin procesar en una representación de cuadrícula estructurada, lo cual minimiza el computo al reducir la resolución de la escena [11], tal y como se muestra en la Figura 4.2.

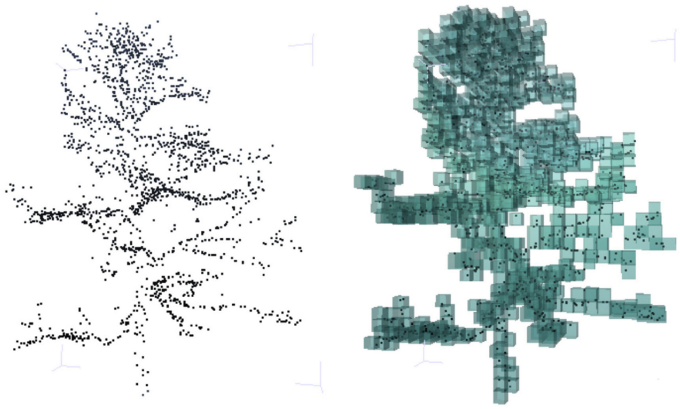


Figura 4.2: Ejemplo de voxelización de una nube de puntos [12].

Para voxelizar una nube de puntos se siguen los siguientes pasos [13]:

1. Se establece el tamaño del vóxel, donde se especifica la anchura, la largura y la altura, lo que viene a ser un vector tridimensional. También es posible definir el máximo de puntos por vóxel.

2. Tras establecer el tamaño, la escena se divide en un conjunto de ortoedros o vóxeles.

3. Si dentro del vóxel se encuentra al menos un punto de la nube de puntos, éste se activa.

Esta representación cuadriculada de la escena puede utilizarse directamente como entrada a las redes neuronales convolucionales (CNN) de detección de objetos, por lo que los recientes avances en este tipo de redes aumenta la utilización de la voxelización. También existe la voxelización 2D, que es muy similar a la 3D, con la diferencia de que únicamente se define el tamaño del vóxel a lo largo de los ejes x e y, y no del eje z [13].

4.1.2 | RANSAC

En los sistemas de detecciones de objetos 3D, la información perteneciente a los puntos que inciden en el suelo no es necesaria. Por esta razón, una de las técnicas más utilizadas para la selección del plano perteneciente al suelo es el algoritmo *Random Sampling and Consensus* (RANSAC) 3D [14].

RANSAC es un método iterativo para calcular los parámetros de un modelo matemático de un conjunto de datos observados que contiene valores atípicos [15]. Tiene una funcionalidad similar a la regresión linear donde, a partir de un conjunto de datos, ambos hayan la relación lineal entre dos características.

La idea principal del algoritmo es la de generar rectas a partir de dos o más puntos para determinar como la recta mejor ajustada aquella que contenga más puntos entre un límite seleccionado [15]. Los puntos dentro del límite son denominados como normales o *inliers* y los demás como anómalos o *outliers* tal y como se observa en la Figura 4.3.

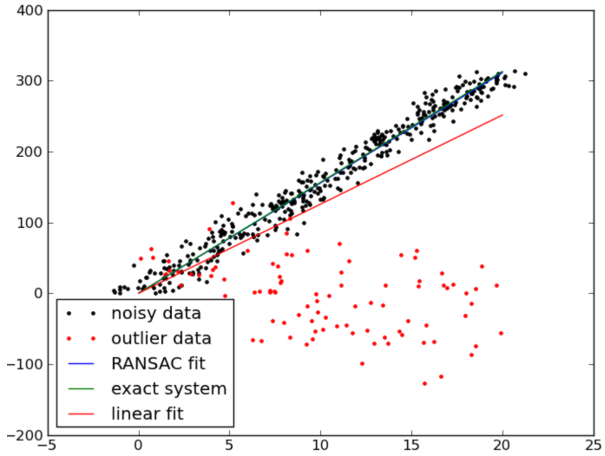


Figura 4.3: Aplicación de la técnica RANSAC para el ajuste de rectas.

En las nubes de puntos obtenidas con el LiDAR, al representar un espacio tridimensional es necesaria una variación del algoritmo para su correcto funcionamiento [14]. Dicha variación se conoce como RANSAC 3D, la cual trabaja con datos 3D en vez de 2D, por lo que pasa de determinar el mejor ajuste de una recta a ajustar un plano [16]. De esta forma y mediante la utilización del RANSAC 3D, se genera un plano ajustándose a la mayoría de puntos que hay en la escena, anotando los puntos pertenecientes al suelo como *inliers* y los demás puntos que colisionan con el entorno como *outliers*. Por lo tanto, se utiliza para clasificar el plano del suelo de la nube de puntos respecto a los demás puntos del entorno, puesto que el suelo y los obstáculos tienen fuertes dependencias de datos al estar situados en la superficie de las carreteras y las calles.

4.1.3 | KD-tree

Una vez que se ha extraído el plano del suelo de la nube de puntos, los distintos objetos del entorno se encuentran separados, puesto que el suelo era el elemento unificador de la mayoría de puntos en la escena. Teniendo ello en cuenta, es necesario realizar una técnica de *clustering*, es decir, un método para agrupar los puntos más cercanos por distancia, ya que hacerlo punto por punto resultaría en una complejidad desmedida.

Para la agrupación de los puntos se podría utilizar directamente el algoritmo *K Nearest Neighbors* (KNN) [17], pero esto produciría agrupaciones o clústeres al encontrarse objetos con pocos vóxeles o demasiados y no establecer una distancia máxima entre vóxeles. Además, la complejidad de computación sería muy elevada.

KD-tree es una estructura de datos de particionado del espacio que organiza los puntos en un espacio k-dimensional [18], lo cual lo convierte en una adecuada estructura para trabajar con datos en un entorno tridimensional, como es el caso de las nubes de puntos o vóxeles. El algoritmo KD-tree tiene una estructura muy similar a un árbol binario en el que cada nodo es un punto k-dimensional.

Para analizar correctamente este tipo de estructura, es necesario comprender los árboles binarios y su funcionamiento. Los árboles binarios son una estructura de datos en forma de árbol en la que cada nodo tiene como máximo dos hijos, los cuales se denominan hijo izquierdo e hijo derecho. La utilidad de la estructura radica en la forma en la que se pueden guardar los datos, puesto que, para buscar un valor, no es necesario iterar por todos ellos caso contrario al de las listas. En un árbol binario ordenado, los valores de la rama de la izquierda del nodo raíz (el nodo más alto del árbol) son iguales o menores. En cambio, los de la rama de la derecha son mayores [19], tal y como se puede observar en la Figura 4.4.

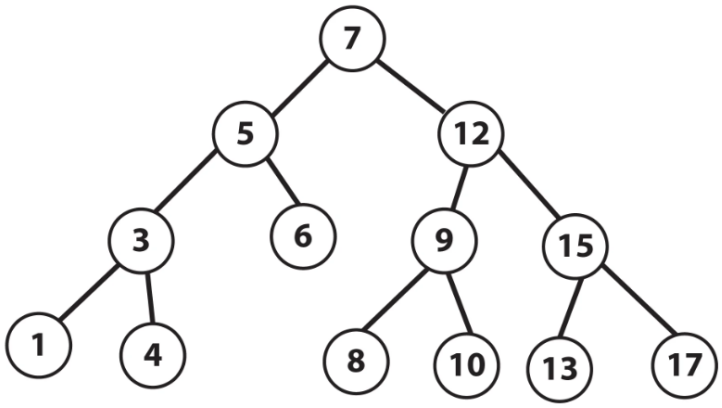


Figura 4.4: Estructura de un árbol binario ordenado.

El algoritmo KD-tree, como se ha mencionado anteriormente, sigue esta misma estructura. Sin embargo, este algoritmo es capaz de localizar un número de K dimensiones, puesto que la principal diferencia respecto a los arboles binarios es la rotación entre dimensiones sobre la que se ordenan los elementos en cada altura del árbol. Esta manera de guardar los datos produce que, a medida que se va aumentando la profundidad del árbol, la región de los nodos hijos es cada vez menor, lo que permite una agrupación y un estudio más sencillo de los datos [18]. En la Figura 4.5 se puede observar la estructura de un KD-tree de dos dimensiones, donde el espacio se divide por regiones. Esto se debe a que cada nodo divide en dos el espacio sobre el que se encuentran sus hijos, lo cual es una perfecta manera de agrupar los puntos en *clústers*.

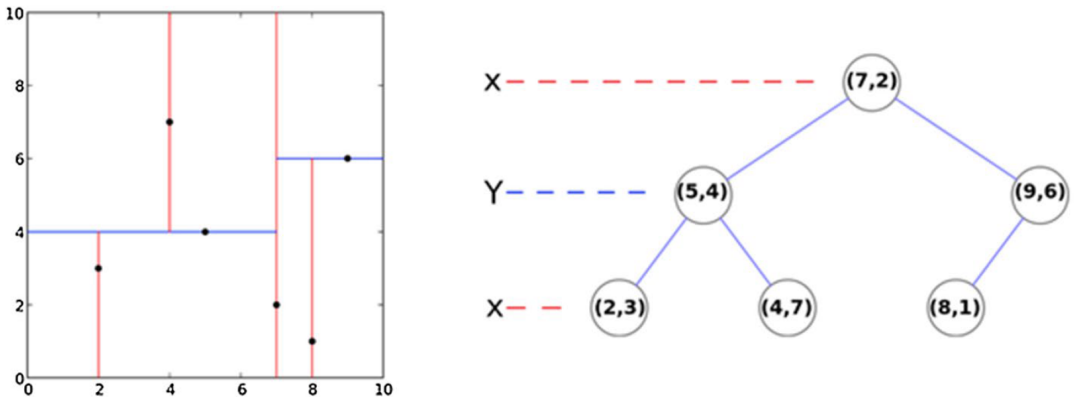


Figura 4.5: Estructura de un KD-tree de dos dimensiones [20].

Aunque sea una técnica que data del año 1975, la eficiencia de esta estructura ha provocado que sea una estructura que se siga estudiando y utilizando como es en el caso de robots aéreos [21] o incluso mejorando como en [22].

4.2 Sistemas de percepción 3D basados en algoritmos Deep Learning

Tras analizar los sistemas de percepción tradicionales, se realiza la revisión del estado del arte de los algoritmos de detección de objetos 3D más actuales y basado en Deep Learning utilizando únicamente datos proporcionados por sensores LiDAR. Al existir una gran variedad de algoritmos se, detallarán los más característicos o utilizados en el momento.

En primer lugar, cabe destacar que el aprendizaje profundo aplicado a las nubes de puntos se enfrenta a varios desafíos [23]:

1. **La naturaleza de la nube de puntos**, ya que contiene datos de alta dimensión y de una naturaleza dispersa y no estructurada.

2. **Requisitos muy exigentes en términos de rendimiento**. Se espera que los vehículos autónomos extraigan características de la nube de puntos y detecten y clasifiquen objetos en tiempo real. Como habitualmente el escenario se escanea a 10 Hz, significa que los modelos tienen un intervalo de 0.1 s para procesar cada escenario y dar salida a las predicciones. Además, deben ser fiables y robustos.

3. **Limitaciones de configuración**. Las unidades de procesamiento equipadas en los vehículos tienen recursos limitados, lo que indica que tendrán que determinar sus predicciones mediante modelos computacionales eficientes.

La representación de la nube de puntos determina el diseño de la red de detección 3D, por lo que todos los algoritmos se dividen según la metodología de representación en tres categorías: métodos basados en la proyección, métodos basados en vóxeles y métodos basados en puntos [24]. En la línea de tiempo de la Figura 4.6 se puede observar la clasificación de los distintos algoritmos a partir de 2015 hasta hoy en día, donde también se encuentran algunos algoritmos de fusión sensorial.

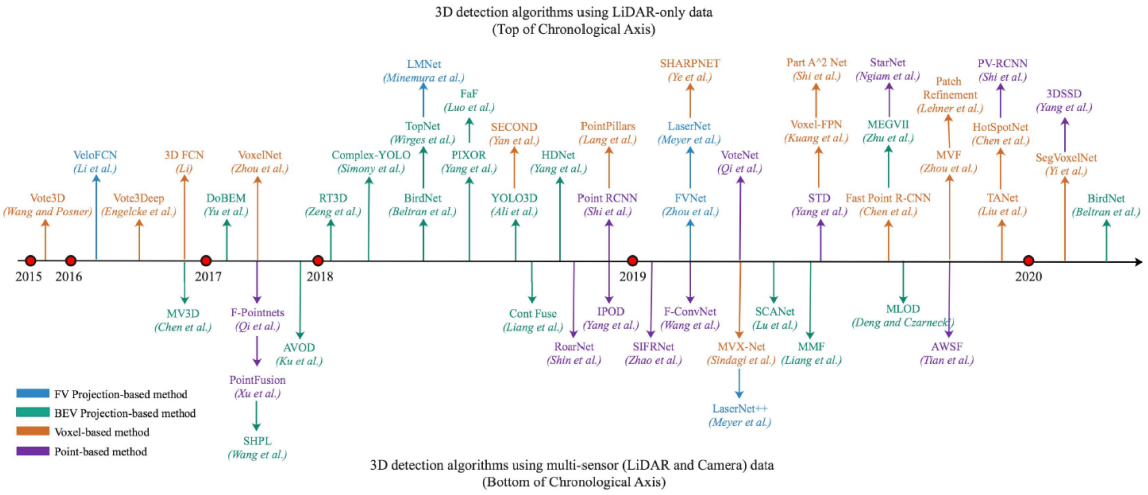


Figura 4.6: Resumen cronológico de los algoritmos de detección de objetos 3D mediante LiDAR [24].

4.2.1 | Métodos basados en proyección

Los métodos basados en proyección se centran en la visualización de la nube de puntos desde una perspectiva específica. En este caso la nube de puntos 3D, se proyecta, en primer lugar, en un mapa 2D compacto y ordenado bajo un punto de vista específico. Esta técnica resulta atractiva puesto que permite utilizar las Redes Neuronales Convolucionales (CNN de sus siglas en ingles) en 2D para tareas en 3D. Por este motivo, muchos de los primeros algoritmos de

detección 3D son métodos basados en proyección. A su vez, estos métodos se dividen en dos subcategorías: Métodos basados en la vista frontal (*Frontal View*) y métodos basados en la vista de pájaro (*Bird Eye View*) [24].

- **Métodos basados en Frontal View (FV):** El mapa FV 2D se obtiene mediante una proyección cilíndrica de la nube de puntos 3D desde la vista frontal del sensor LiDAR. Suele formar fácilmente un mapa en dos dimensiones para utilizar el detector 2D disponible. Este método facilita la fusión sensorial, puesto que la información codificada en el mapa FV permite capturar las dependencias entre distintas vistas, como la nube de puntos sin procesar y la imagen de la cámara. Sin embargo, la proyección provoca inevitablemente una pérdida de detalle. Además, el tamaño del objeto percibido varía en función de la distancia a la que se encuentra del sensor y los problemas de oclusión que se encuentran en el detector 2D también son complicados en estos métodos.
- **Métodos basados en Bird Eye View (BEV):** La proyección de vista de pájaro se obtiene mediante la proyección en el plano del suelo de toda la nube de puntos y la discretización en una determinada resolución. El mapa BEV conserva la información de longitud y anchura del objeto y facilita el calculo del *yaw* del objeto. Además, en las aplicaciones de escenas exteriores, la proyección del plano del suelo elimina los problemas de la escala del objeto y los problemas de oclusión, por lo que la localización del objeto es más factible. Sin embargo, las fluctuaciones del plano del suelo acumulan errores de predicción de altura y no se puede manejar bien la oclusión en objetos como postes, ya que tienen pocos puntos después de la proyección BEV.

Los métodos basados en proyección dependen, en gran medida, de la actualización de los algoritmos de detección 2D. Asimismo, la perdida irreversible de información introducida por la proyección hoy en día sigue limitando la precisión de la detección de objetos [24].

4.2.2 | Métodos basados en vóxeles

El propósito de los métodos basados en vóxeles consiste en convertir los datos irregulares de la nube de puntos en una matriz ordenada para aplicar un filtro de convolución. El entorno 3D se discretiza en cuadrículas de vóxel de tamaño fijo, los cuales alojan puntos no estructurados. A diferencia de la proyección BEV, la voxelización se realiza en tres dimensiones, por lo que preserva la estructura 3D de los datos de la nube de puntos sin procesar.

No obstante, la nube de puntos LiDAR dispersa da lugar a un gran número de vóxeles vacíos, mientras que el cálculo espacial 3D crece exponencialmente con una resolución de vóxeles más

fina. Analizar eficazmente los vóxeles dispersos supone un gran reto para estos métodos. Dentro de estos métodos se encuentran los algoritmos de SECOND y *PointPillars*:

- **SECOND:**

  Sparsely Embedded CONvolutional Detector (SECOND) [25] es un modelo basado en vóxeles publicado en 2018 para mejorar la detección de objetos 3D utilizando únicamente datos LiDAR. Consiste en tres elementos: extractor de características a nivel de vóxel, una CNN dispersa y una Region Proposal Network (RPN). A continuación se detalla todo el proceso desde las nubes de puntos sin procesar a la salidas del modelo tal y como se muestra en la Figura 4.7:

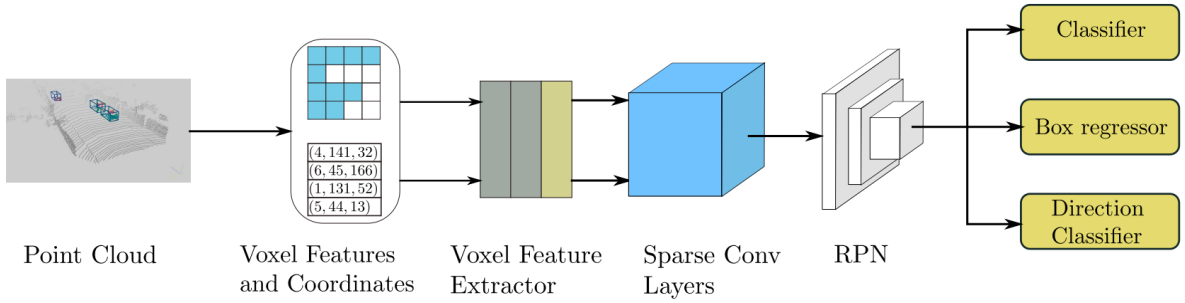


Figura 4.7: Arquitectura del detector SECOND [25].

1. **Procesamiento de la nube de puntos**: Se sigue el procedimiento descrito en [26] para obtener una representación voxelizada de la nube de puntos. En primer lugar, se preasignan los buffers en función del límite especificado en el número de vóxeles para, posteriormente, iterar sobre la nube de puntos, asignando los puntos a sus vóxeles asociados. Se guardan tanto el número de vóxeles como el número de puntos por vóxel. Se utiliza un tamaño de vóxel fijo de [0.4, 0.2, 0.2] m y el máximo número de puntos por vóxel se establece en 35 para la detección de coches, el cual se selecciona en función de la distribución del número de puntos por vóxel en el conjunto de datos KITTI (*Karlsruhe Institute of Technology and Toyota Technological Institute*) [25]. En cambio, para la detección de peatones o ciclistas, el número máximo de puntos se establece en 45 porque, al ser relativamente pequeños se necesitan más puntos para la extracción de características en los vóxeles.
2. **Extractor de características a nivel de vóxel:** Se utiliza una capa de *Voxel Feature Encoding* (VFE) tal y como se describe en [26] para extraer las características a nivel de vóxel. Una capa VFE toma todos los puntos del mismo vóxel como entrada y utiliza una red totalmente conectada (FCN) que consiste en una capa lineal, una capa que aplica *batch normalization* y una *Rectified Linear Unit* (ReLU) para extraer

características puntuales. A continuación, se aplica la operación matemática *max pooling* para obtener las características agregadas localmente para cada vóxel. Por último, se concatenan tanto las características obtenidas como las puntuales.

3. **Extractor convolucional disperso:** Utilizando las redes neuronales *Sparse Convolutional Networks* se consigue una mejora en el rendimiento, puesto que no se tienen en cuenta los vóxeles que no contienen ningún punto. Para poder aplicar este tipo de redes, es necesario un algoritmo que contenga las reglas que indican que partes del kernel o que índices van a ser utilizados [26].

   Se construye una tabla de matrices de reglas para guardar los índices utilizados y las reglas se generan mediante un algoritmo. *SparseConvNet* [27] es el modelo original que ofrece la implementación del Submanifold Convolution, una técnica dentro del campo de las redes *Sparse Convolutional Networks* que realiza la generación de reglas en CPU.

   En cambio, en el modelo SECOND se implementa un generador de reglas en GPU aprovechando la aceleración por hardware y el procesamiento paralelo para conseguir una generación de reglas en la mitad de tiempo que *SparseConvNet* [27].Este extractor se utiliza para convertir la información 3D en un formato similar a una imagen en BEV.

4. **Region Proposal Network:** Se utiliza una arquitectura RPN similar a la arquitectura de los detectores de disparo único (SSD) para que, a partir del mapa de características extraído en la fase anterior, se obtengan las predicciones del modelo [26].

5. **Obtención de las detecciones a la salida del modelo:** En la salida se utilizan unos tamaños fijos para las diferentes clases de objetos que han sido determinados en función de la media de los tamaños y las ubicaciones centrales del *groundtruth* del dataset KITTI.

En el momento de su publicación consiguió convertirse en el mejor modelo de detección del estado del arte (SOTA) según la evaluación con el conjunto de validación de KITTI. Además, esta arquitectura ha demostrado funcionar en tiempo real con un tiempo de computo de 0.05 s (20 Hz) en su modelo completo y 0.025 s en su modelo reducido utilizando la tarjeta gráfica GTX 1080 Ti[26].

- ***PointPillars*:**

  El algoritmo *PointPillars* [28] además de conseguir una mejora en la precisión de la detección de objetos 3D, lo hace con una velocidad de inferencia de 62 Hz, utilizando la tarjeta gráfica GTX 1080 Ti al igual que en [26]. Este aumento en la velocidad facilita la

integración en los sistemas embebidos, puesto que puede llegar a alcanzar los 105 Hz con la pérdida limitada de la precisión [28].

Este modelo, a diferencia de SECOND, consigue eliminar el uso de las capas convolucionales 3D para obtener esas velocidades de inferencia tan altas, convirtiendo las nubes de puntos en imágenes BEV [28]. A continuación, se detallan las tres partes fundamentales del modelo, las cuales se pueden observar en la Figura 4.8.

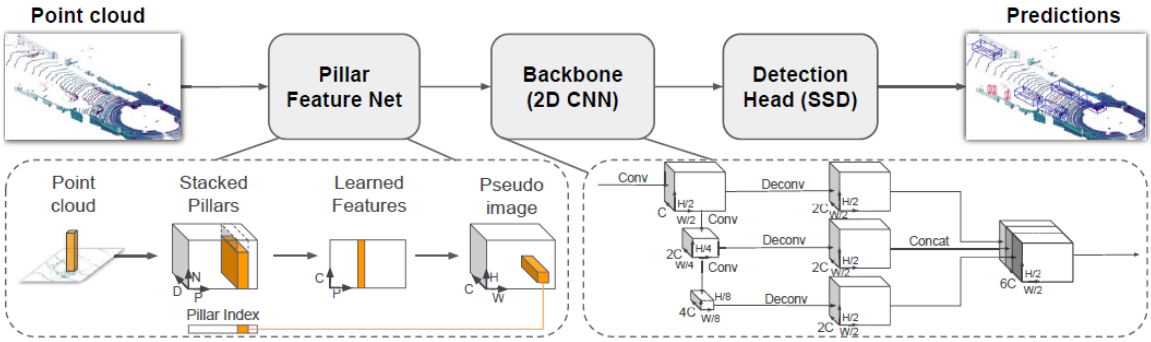


Figura 4.8: Arquitectura del detector *PointPillars* [28].

1. **De nube de puntos a pseudo-imagen:** Para aplicar una arquitectura convolucional 2D, en primer lugar la nube de puntos se convierte en una pseudo-imagen [28]. Se comienza discretizando la nube de puntos en una cuadrícula uniformemente espaciada en el plano x-y, creando un conjunto de pilares, que es lo mismo que un conjunto de vóxeles pero con extensión espacial ilimitada en la dirección z.

   A los puntos xyz de cada pilar se les añade la reflectividad, la distancia a la media aritmética de todos los puntos del pilar en las tres dimensiones y el desplazamiento desde el centro del pilar en x e y. De esta forma, se crea un espacio de nueve dimensiones por pilar. El conjunto de pilares estará mayoritariamente vacío debido a la escasez de puntos, por lo que se impone un límite tanto al número de pilares no vacíos como al número de puntos por pilar. En caso de que algún pilar contenga muy pocos datos se le aplica la técnica *zero padding*.

   A continuación, se utiliza una versión simplificada de PointNet [29], donde se aplica una capa linear, seguido de un *batch normalization* y una ReLU. Transfiriendo los pilares no vacíos a la altura y anchura de la imagen en función de la posición de dichos pilares se consigue una pseudo-imagen, por lo que, al utilizar pilares en vez de vóxeles, se ha conseguido eliminar las capas convolucionales 3D.

2. **Backbone:** Se emplea un backbone similar al de VoxelNet [26], el cual contiene dos subcapas. Una de ellas es una red descendente que produce características con una

resolución espacial cada vez más pequeña y la otra realiza una concatenación de las características de los pilares [28].

3. **Detection Head:** Para realizar la detección de objetos 3D se utiliza el *Single Shot Detector* (SSD) o Detector de Disparo Único.

Este modelo puede ser entrenado de principio a fin y, utilizando el dataset KITTI en el momento de la publicación, se demuestra que *PointPillars* es en general superior al resto de métodos existentes ofreciendo un mayor rendimiento de detección a una mayor velocidad [28].

4.2.3 | Métodos basados en puntos

Los dos métodos anteriores regularizan la nube de puntos en cuadrículas de imágenes o vóxeles, con el objetivo de realizar una convolución para la extracción y detección de características. Sin embargo, este proceso conlleva perder la geometría natural de los puntos, por lo que resulta atractivo modelar directamente la nube de puntos sin procesar para reducir la perdida de información [24].

En 2017, en el algoritmo PointNet [29] se introduce una arquitectura que consume directamente conjuntos de puntos desordenados para capturar las características. Rápidamente, esta idea ha sido trasladada a la detección de objetos 3D, siendo independiente al formato de la nube de puntos.

- **PointRCNN**

  Más o menos a la par que *PointPillars*, también se publicó el modelo PointRCNN [30] para la detección de objetos 3D mediante nubes de puntos irregulares. La estructura del modelo se muestra en la Figura 4.9 y consta de una primera fase de generación de las detecciones 3D y otra de refinamiento de las *bounding boxes* 3D.

  1. **Generación de propuestas 3D ascendentes mediante la segmentación de nubes de puntos:** Los métodos de detección de objetos 2D de dos etapas generan, en primer lugar, las propuestas y refinan las propuestas y las confidencias en una segunda etapa. Sin embargo, la extensión directa de los métodos de dos etapas 2D a 3D no es trivial debido al enorme espacio de búsqueda 3D y el formato irregular de las nubes de puntos [30].

     En el modelo PointRCNN se propone un algoritmo de generación de propuestas 3D basado en la segmentación de la nube de puntos. Se observa que los objetos en 3D

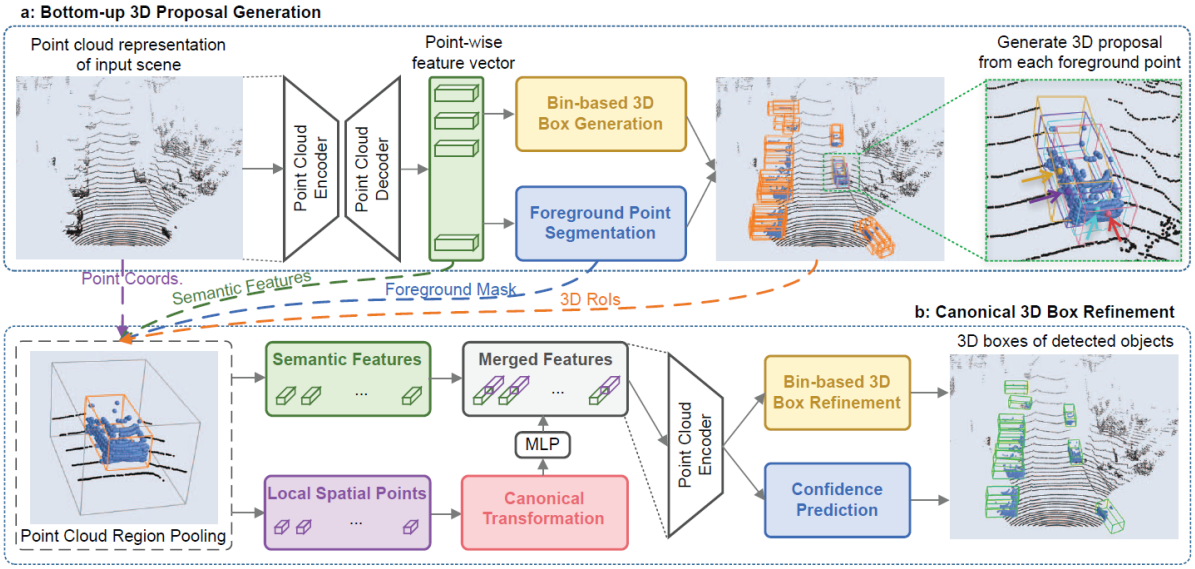


Figura 4.9: Arquitectura PointRCNN para la detección de objetos 3D [30].

se separan de forma natural sin superponerse entre sí, por lo que todas las máscaras de segmentación de los objetos se pueden obtener directamente por las anotaciones de sus cajas delimitadoras 3D.

Para extraer las características a nivel de punto de la nube de puntos se utiliza PointNet++ con agrupación multiescala como red principal. Con esta red y el método propio de generación de *bounding boxes* 3D (bin-based) se hayan las múltiples detecciones 3D, que se reducen por cada objeto aplicando el procedimiento *Non Maximum Suppression* (NMS), basándose en el criterio de evaluación *Intersection over Union* (IoU) a vista de pájaro.

2. **Refinamiento de las cajas delimitadoras 3D:** En cuanto al refinamiento, se trata de determinar de una manera más exacta el centro y la orientación de los *bounding boxes* generados. Se aumenta el tamaño de cada detección en un valor constante y se aplica una máscara para diferenciar los puntos de la detección original al espacio aumentado. Además, cada detección pasa a utilizar un sistema de coordenadas propio.

   Se combinan los puntos de cada agrupación y las características extraídas en la primera fase para un mayor refinamiento de la caja delimitadora. Además, como los objetos lejanos suelen tener menos puntos que los objetos cercanos, también se ha incluido la distancia al sensor $\sqrt{x^2+y^2+z^2}$ para compensar la pérdida de información de profundidad. Tras ello, se vuelve a utilizar el modelo *bin-based* para obtener las cajas delimitadoras 3D y aplicar nuevamente NMS sobre un IoU a vista de pájaro de 0.01 para eliminar solapamientos [30].

Los experimentos realizado en [30] muestran que PointRCNN supera a los métodos anteriores del estado del arte en la detección 3D. Sin embargo, al ser de dos etapas y tener como entrada la nube de puntos sin procesar, obtiene una velocidad de inferencia de 10 Hz [31], lo suficiente para aplicarse en tiempo real, ya que normalmente las nubes de puntos se obtienen a exactamente la misma frecuencia.

- **PV-RCNN**

  El algoritmo PV-RCNN [32] aumenta el rendimiento de la detección 3D incorporando las ventajas tanto de los métodos de aprendizaje de características basado en puntos como en vóxeles. El principio de operación de PV-RCNN reside en el hecho de que la operación basada en vóxeles codifica eficazmente representaciones de características multiescala y puede generar propuestas 3D de alta calidad, mientras que la operación de abstracción de conjuntos basada en puntos preserva la información precisa de localización [32]. Este modelo contiene tres etapas fundamentales (Figura 4.10):

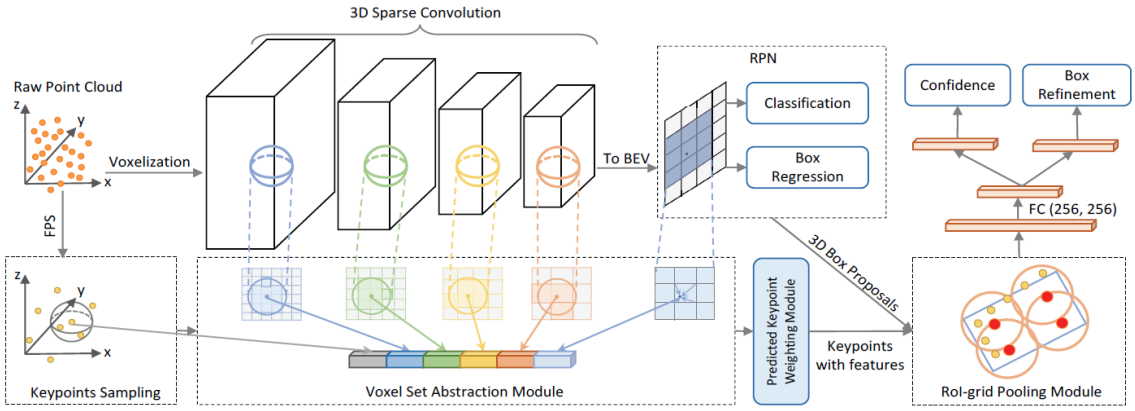


Figura 4.10: Arquitectura del modelo PV-RCNN [32].

1. **CNN de vóxeles 3D para la codificación eficiente de características y generación de propuestas:** Se emplea la CNN de vóxeles con convolución dispersa 3D, la cual es una opción muy popular entre los detectores 3D de última generación para convertir eficientemente las nubes de puntos en volúmenes de características 3D dispersas. Este método obtiene, de forma interna, características semánticas de los vóxeles, además de conseguir las detecciones de los objetos a partir de los tamaños prefijados.
2. **Codificación de escenas de vóxeles a puntos clave:** En primer lugar, se agregan las características multiescala de los vóxeles que representan toda la escena en unos puntos clave, los cuales sirven de enlace entre la CNN de la etapa anterior y el refinamiento de las detecciones [32]. Se adopta el algoritmo *FurthestPoint-Sampling*

(FPS) para muestrear un pequeño número de puntos clave que se distribuyen uniformemente alrededor de los vóxeles no vacíos y ser representativos de la escena. En el siguiente paso, se incluye el módulo *Voxel Set Abstraction* (VSA) para codificar las características de la primera etapa a los puntos clave.

Por último, como los puntos clave que pertenecen a los objetos en primer plano deberían contribuir más a la precisión de las propuestas, mientras que los de las regiones del fondo deberían contribuir menos, se introduce el módulo *Predicted Keypoint Weighting* (PKW). En este módulo se introducen los puntos clave codificados para dar un peso a cada uno en función de su importancia para las detecciones [32].

3. **Abstracción de características Region of Interest (RoI) de puntos clave a cuadricula para el refinamiento de propuestas:** Se utiliza el módulo *RoI-grid pooling* para agregar las características de los puntos clave a los puntos de la cuadrícula RoI. Se identifican los puntos clave vecinos del puntos de cuadrícula dentro de un radio, tal y como se observa en la Figura 4.11. Tras ello, se obtienen las características de la cuadrícula y, a partir de estas, se ajusta la detección con una pequeña red de dos capas.

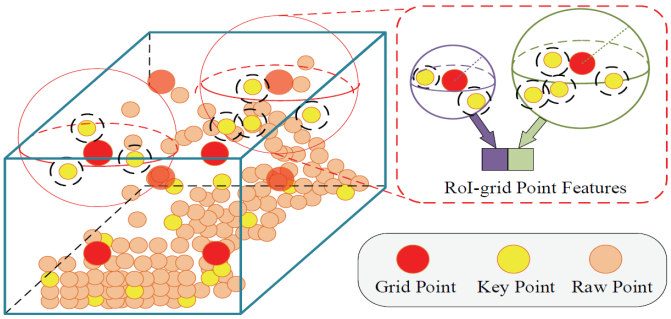


Figura 4.11: Módulo RoI-grid pooling de PV-RCNN [32].

En este caso, mediante los experimentos realizados por los autores se ha demostrado una mejora en el rendimiento de la detección de objetos 3D, pero al igual que el algoritmo PointRCNN tiene una velocidad de inferencia baja de alrededor de 12 Hz [24].

5 | Herramientas y Tecnologías

En este apartado se detallan las distintas herramientas, tanto software como hardware, que se han utilizado a lo largo del Trabajo Fin de Máster. Además, se describen las tecnologías en las que se basa el trabajo.

5.1 | Hardware

Durante la realización del proyecto se ha hecho uso de diferentes dispositivos o equipos hardware. A continuación, se detallan las características y funcionalidades de los equipos utilizados en las distintas tareas.

5.1.1 | LiDAR Livox Mid-70

El sensor LiDAR utilizado en este trabajo es el Livox Mid-70, tal y como se ha mencionado anteriormente. El Livox Mid-70 se encuentra en la categoría de LiDARs terrestres, en cuanto a la plataforma utilizada, y es un LiDAR de estado sólido en cuanto al mecanismo de escaneo, puesto que tiene un campo de visión fijo sin partes móviles, a diferencia de los giratorios que realizan un escaneo de 360º (Figura 5.1).

Figura 5.1: Sensor LiDAR Livox Mid-70.

La tecnología única de patrones de escaneo no repetitivos de los sensores LiDAR de Livox difieren significativamente del escaneo lineal repetitivo que ofrecen los sensores LiDAR tradicionales. Las áreas escaneadas dentro del campo de visión de un sensor Livox aumentan a medida que aumenta el tiempo de integración, por lo que garantiza una nube de puntos de alta densidad, tal y como se puede observar en la Figura 5.2.

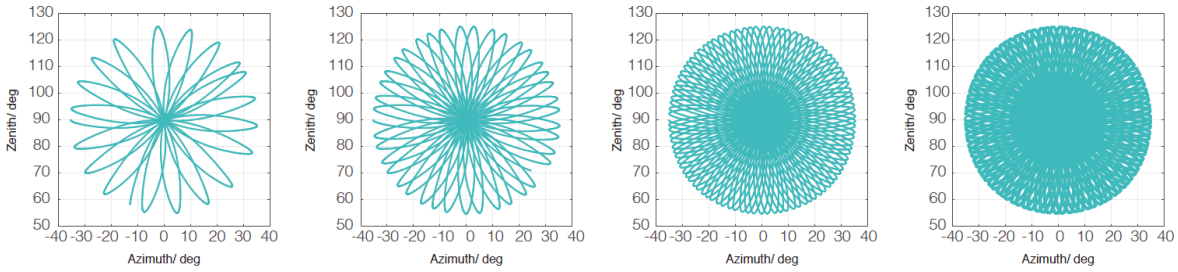


Figura 5.2: Patrón de escaneo según el tiempo de integración del Livox Mid-70 [33].

En la Figura 5.3 se muestra la cobertura del campo de visión (FOV de sus siglas en ingles) del Mid-70 comparada con los sensores LiDAR mecánicos tradicionales que utilizan métodos de escaneo comunes. En el gráfico se puede observar que, cuando el tiempo de integración es de 0.2 segundos, la cobertura del FOV es similar a la del sensor LiDAR de 32 líneas. A medida que sube el tiempo de integración, la cobertura del FOV del Mid-70 se incrementa significativamente. Después de 1.5 segundos, se aproxima al 86 %, por lo que casi todas las zonas estarían iluminadas por los rayos láser [33].

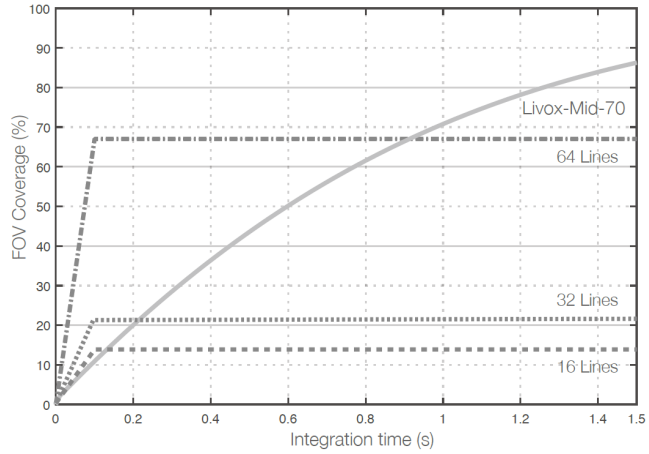


Figura 5.3: Comparación entre el escaneo del Livox Mid-70 y el escaneo lineal [33]..

Para finalizar, el Livox Mid-70 tiene un campo de visión circular de 70.4º y una zona ciega de 5 cm. Además, dispone de un rango de detección mayor a 260 m, que puede ser alcanzado cuando el objeto de destino refleja un 80 % o más de la luz, y la precisión de detección es de 3 cm de 0.2 a 1 m [33].

5.1.2 | Nvidia Jetson Nano

La NVIDIA Jetson Nano es un pequeño y potente ordenador que permite ejecutar varias redes neuronales en paralelo para aplicaciones como la clasificación de imágenes, la detección de objetos, la segmentación o el procesamiento del habla. Diseñada para su uso en entornos con limitaciones de energía, la Jetson Nano exprime muy buenas capacidades de cálculo. Estos son sus componentes principales [34]:

- SoC NVIDIA Tegra de la serie X1
  - GPU NVIDIA Maxwell
  - CPU ARM Cortex-A57 de cuatro núcleos
- 4GB de memoria LPDDR4
- 16 GB de almacenamiento eMMC 5.1
- Gigabit Ethernet (10/100/1000 Mbps)
- PMIC, reguladores, monitores de potencia y voltaje
- Conector con llave de 260 pines (expone E/S estándar de alta y baja velocidad)
- Sensores de temperatura en el chip

5.1.3 | WorkStation

La *WorkStation* (WS) es un equipo de Ikerlan, concretamente del grupo STF, que se utiliza cuando es necesario una mayor capacidad de computo o la utilización una tarjeta gráfica más potente, como por ejemplo para utilizar el simulador CARLA o entrenar redes neuronales. Los componentes principales de la WS son los siguientes:

- GPU NVIDIA RTX 2080 Ti 11 GB
- CPU Intel i9-9900k
- 64 GB de RAM
- Disco duro de 4 TB

5.2 | Software

En este apartado se describen las tecnologías y herramientas software utilizadas en Trabajo Fin de Máster. Entre ellas se incluyen *Robotic Operative System* y el simulador CARLA.

5.2.1 | Robotic Operative System

Robotic Operative System o ROS es un meta-sistema operativo o *middleware* de código abierto que contiene un amplio set de herramientas y librerías software para el desarrollo de aplicaciones de robótica. ROS provee al desarrollador los servicios estándar de un sistema operativo, tales como la abstracción del hardware, el control de dispositivos a bajo nivel, la implementación de funcionalidades de uso común, la transferencia de mensajes entre procesos y la gestión o administración de paquetes. También proporciona herramientas y librerías para obtener, construir, escribir y ejecutar código en múltiples dispositivos [35].

ROS fue diseñado para hacer frente a una serie de retos específicos a los que se debe enfrentar el desarrollador de robots a gran escala. Para conseguirlo, la filosofía de ROS se puede resumir en los conceptos que se detallan a continuación [36]:

- **Peer-to-Peer:** Un sistema construido usando ROS consiste en un número de procesos conectados en una topología *peer-to-peer* (P2P). Este tipo de topología no funciona con clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre si. De esta forma, las redes P2P permiten el intercambio directo de información entre nodos interconectados. P2P requiere de algún tipo de mecanismo de búsqueda que permita a los procesos encontrarse entre sí en tiempo de ejecución. Este mecanismo se llama servicio o maestro.

- **Multilenguaje:** Para solventar el problema de preferencia de lenguajes por parte de diferentes usuarios, ROS se ha diseñado para que sea neutral en cuanto al lenguaje. Por esta razón, actualmente es compatible con cuatros lenguajes muy diferentes: C++, Python, Octave y LISP. Para soportar el desarrollo entre lenguajes, ROS utiliza un lenguaje de definición de interfaz (IDL) simple y neutral para describir los mensajes enviados entre módulos. El resultado final es un esquema de procesamiento de mensajes neutro en el que se pueden mezclar y combinar diferentes lenguajes.

- **Basado en herramientas:** Con el objetivo de sobrellevar la complejidad de ROS, se ha optado por el diseño de un microkernel, donde se utilizan un gran número de herramientas para construir y ejecutar los diferentes componentes ROS, en lugar de construir un entorno monolítico de desarrollo y de ejecución.

- **Thin:** Por distintas razones, es difícil extraer la funcionalidad de gran parte del código y reutilizarlo fuera de su contexto original. Para oponer esta tendencia, todo el desarrollo de controladores y algoritmos se realiza en bibliotecas independientes que no dependen de ROS. En consecuencia, toda la complejidad recae en las bibliotecas y únicamente crea pequeños ejecutables que exponen la funcionalidad de la biblioteca a ROS, permitiendo una reutilización de código y una extracción más fácil.

- **Gratuito y de código abierto:** El código fuente completo de ROS está disponible públicamente. Esta característica es fundamental para la depuración de la pila de software en todos los niveles. Se distribuye bajo los términos de la licencia *Berkeley Software Distribution* (BSD), que permite el desarrollo de proyectos tanto comerciales como no comerciales.

ROS fue creado en 2007 y, desde entonces, se han producido muchos cambios en la robótica y en la comunidad de ROS [37]. Como consecuencia, se han detectado varias debilidades. ROS no satisface los requisitos de ejecución en tiempo real, no puede garantizar la tolerancia a fallos, los plazos o la sincronización de procesos y, además, requiere de importantes recursos [38]. Para adaptarse a estos cambios y combatir las debilidades se ha creado ROS2, el cual aprovecha las fortalezas de ROS e incorpora mejoras que permiten superar sus principales debilidades. A continuación, se detallan los conceptos básicos de ROS para comprender el funcionamiento del sistema:

- **Nodos**

  Un nodo equivale a un proceso que ejecuta un tipo de computación. Cada nodo debe ser responsable de un único módulo. Por ejemplo, un nodo para controlar las ruedas del motor, un nodo que controle el sensor de ultrasonidos, un nodo que lleve a cabo la localización, etc. Cada nodo puede mandar y recibir información mediante *topics*, servicios, acciones o parámetros.

  El uso de nodos aporta ventajas significativas al sistema.Estos reducen la complejidad del código y aumentan la tolerancia a fallos, puesto que si un nodo está defectuoso no hay que cambiar todo el sistema.

- **Topics**

  Los *topics* son un elemento vital dentro de ROS, puesto que actúan como un bus para que los nodos intercambien información. Un nodo puede publicar datos a cualquier número de *topics* y, simultáneamente, tener subscripción a cualquier *topic*. Están pensados para una comunicación unidireccional y en *streaming*.

Los nodos que publican información en un *topic* se llaman *publishers* y los que recogen información *subscribers*. Por lo tanto, un mismo nodo puede desempeñar las dos tareas. De la misma forma, un mismo *topic* puede tener más de un *publisher* y más de un *subscriber*. En el diagrama que se muestra a continuación (Figura 5.4) se describe la comunicación de distintos nodos a través de un *topic*.

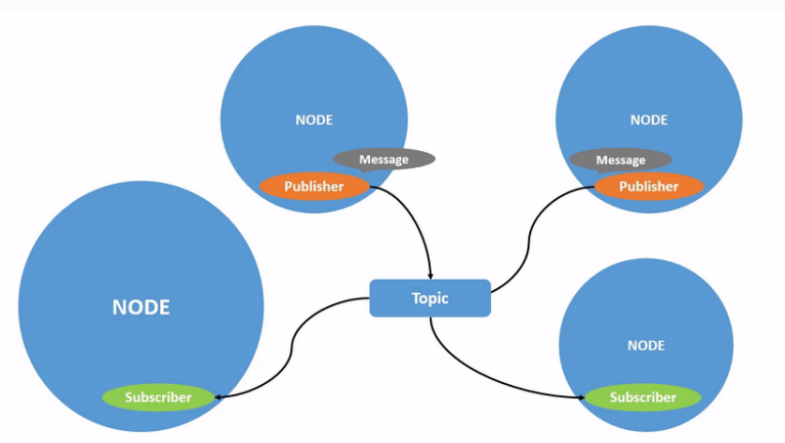


Figura 5.4: Diagrama que muestra la funcionalidad de los *topics* [37].

- **Servicios**

  Los servicios proveen otra manera de comunicación entre nodos que se basan en el sistema cliente-servidor. Los *topics* se basan en un modelo de *publisher-subscriber*; en cambio, los servicios se basan en el modelo llamada y respuesta. Mientras los *topics* habilitan a los nodos a suscribirse a la información y obtener actualizaciones continuas, los servicios únicamente proporcionan información cuando has sido llamados específicamente por el cliente. En la Figura 5.5 se muestra la comunicación de dos nodos mediante un servicio.

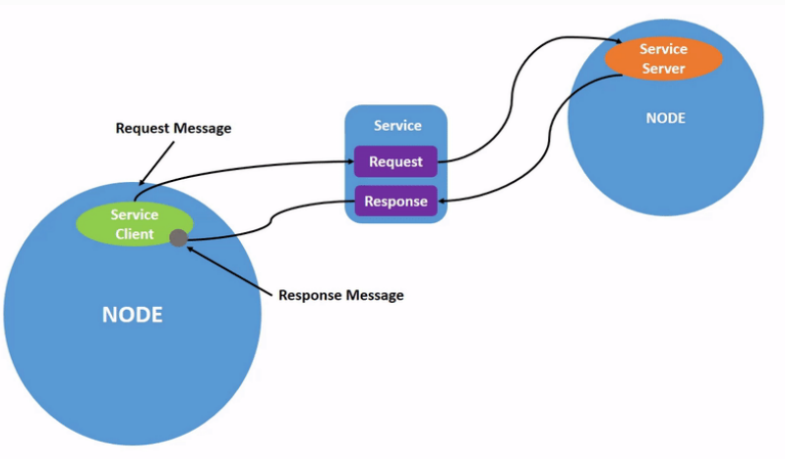


Figura 5.5: Diagrama que muestra la funcionalidad de los servicios [37].

- **Acciones**

  Las acciones son un método de comunicación exclusivo de ROS2 y están destinadas a tareas de larga duración. Consisten en tres partes: Un objetivo, una retroalimentación y un resultado. Su funcionalidad es similar a la de los servicios, puesto que también utiliza el modelo cliente-servidor. Sin embargo, las acciones se pueden cancelar mientras se ejecutan y también proporcionan una retroalimentación constante, a diferencia de los servicios que devuelven una única respuesta. En el diagrama que se muestra a continuación (Figura 5.6) se describe la comunicación mediante una acción.

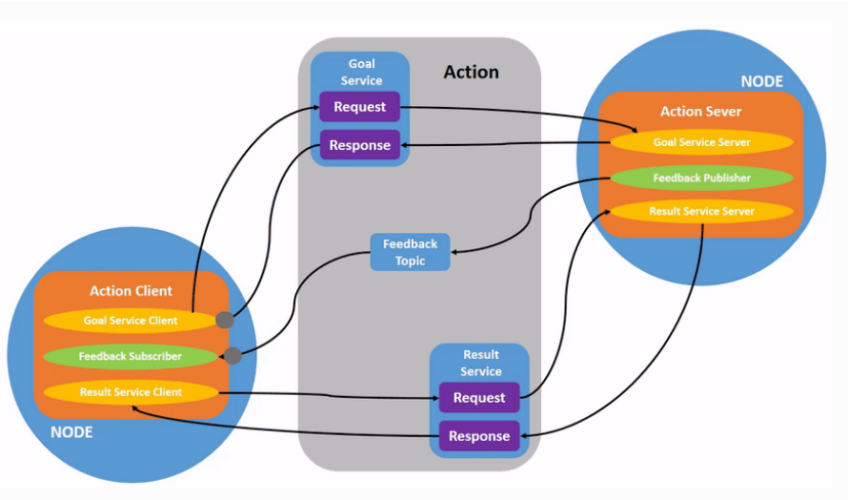


Figura 5.6: Diagrama que muestra la funcionalidad de las acciones [37].

- **Parámetros**

  Los parámetros se utilizan para configurar los nodos al inicio y durante el tiempo de ejecución, sin cambiar el código. En ROS, estos parámetros se almacenan en el llamado *parameter-server*, que es un diccionario compartido y multivariable que los nodos utilizan para almacenar y recuperar parámetros. En cambio, en ROS2 los parámetros están asociados con nodos individuales y su ciclo de vida está vinculado con el ciclo de vida del nodo. Cada parámetro consiste en una clave, un valor y un descriptor.

  En este trabajo, en la medida de lo posible se ha utilizado ROS2, ya que es la versión mejorada de ROS. Sin embargo, al estar en proceso de emigración, todavía existen algunas funcionalidades que no están implementadas, por lo que se ha terminado utilizando ROS en la mayoría de los casos.

5.2.2 | CARLA

CARLA es un simulador de código abierto para la investigación de la conducción autónoma [39]. Ha sido desarrollado para apoyar el entrenamiento, la creación de prototipos y la validación de modelos de conducción autónoma, incluyendo la percepción y el control [40]. Uno de los objetivos principales de CARLA es ayudar a democratizar la I+D de la conducción autónoma, sirviendo como herramienta de fácil acceso y modificación. Se basa en el motor de videojuegos *Unreal Engine* para ejecutar la simulación y utiliza el estándar OpenDRIVE para definir las carreteras y los entornos urbanos [39]. En la Figura 5.7 se muestra la apariencia del simulador CARLA.

Figura 5.7: Simulador CARLA [40].

Es una herramienta muy útil para analizar el funcionamiento de los distintos algoritmos utilizados en un entorno simulado, puesto que entre los sensores que integra se encuentra un LiDAR de 32 haces de luz láser.

5.2.3 | Docker

Docker [41] es un proyecto *Open-Source* que automatiza el despliegue de aplicaciones dentro de contenedores software, proporcionando una capa de abstracción y automatización de la virtualización de aplicaciones en múltiples sistemas operativos [42].

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado (auto contenidos), independientemente del sistema operativo que la máquina tenga por debajo, facilitando así los

despliegues. Además, se pueden compartir los contenedores de forma muy sencilla haciendo uso de Docker Hub con un funcionamiento basado en *push/pull*.

5.2.4 | Gazebo

Gazebo [43] es un simulador de robótica 3D de código abierto. Es capaz de simular una serie de robots, sensores y objetos en un mundo tridimensional. Tiene las siguientes características técnicas:

- **Simulación dinámica**, con acceso a múltiples motores de física de alto rendimiento.
- **Gráficos 3D avanzados**. Usando *Object-Oriented Graphics Rendering Engine* (OGRE) puede generar entornos y renderizar formas realistas, texturas, luces, sombras, etc.
- **Sensores y ruido**, donde es posible obtener una amplia variedad de datos.
- **Plugins** para que los desarrolladores puedan personalizar robots, sensores y entornos de control gracias a su API.
- **Múltiplrd modelos** de robot ya realizados o para construir uno propio usando *Simulator Description Format* (SDF) [44].

5.2.5 | Open3D

Open3D [45] es una biblioteca *open-source* que facilita el desarrollo software que trata con datos 3D. El *frontend* de Open3D contiene un conjunto de estructuras de datos y algoritmos tanto en C++ como en Python. El *backend* está altamente optimizado y está preparado para la paralelización. Las características principales de la librería son las siguientes:

- Estructura de datos 3D.
- Algoritmos para procesar datos 3D.
- Reconstrucción de escenas.
- Alineación de superficies.
- Visualización 3D.
- Renderizado basado en la física (PBR).

- Soporte de aprendizaje automático 3D con PyTorch y TensorFlor.
- Aceleración de la GPU para las operaciones 3D principales.

5.2.6 | NumPy

NumPy [46] es una librería *open-source* de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Incorpora una clase que permite representar colecciones de datos de un mismo tipo en varias dimensiones, además de funciones muy eficientes para su manipulación.

La ventaja es que es más rápido en el procesamiento de los arrays que las listas predefinidas en Python, por lo que es ideal para procesar vectores y matrices multidimensionales.

5.2.7 | Pytorch

Pytorch [47] es un *framework* de *machine learning open-source* basado en la librería Torch. Se utiliza para aplicaciones como la visión artificial y el procesamiento de lenguajes naturales.

Pytorch proporciona dos características principales:

- Computación tensorial (como NumPy) con una aceleración muy desarrollada a través de unidades de procesamientos gráficos (GPU).
- *Deep neural networks* construidas en un sistema de diferenciación automática de bases de datos.

5.2.8 | OpenPCDet

OpenPCDet [48] es un proyecto *open-source* desarrollado por OpenMMLab. Su principal aportación es integrar en un mismo espacio de trabajo distintas redes neuronales especializadas en la detección de objetos 3D para facilitar su ejecución y comparativa. Este ecosistema proporciona las arquitecturas de red del estado del arte, así como un conjunto de modelos entrenados previamente sobre el *dataset* KITTI, ahorrando el tiempo de procesamiento computacional requerido para su entrenamiento.

El patrón de diseño de OpenPCDet se basa en las siguientes tres funcionalidades:

1. Separación de datos y modelos con coordenadas de nubes de puntos unificadas para ampliar fácilmente a conjuntos de datos personalizados.

2. Definición de *bounding boxes* 3D unificadas: [*x, y, z, dx, dy, dz, heading*].

3. Estrucutura de modelo felxible para soportar varios modelos de detección 3D.

Esta herramienta incluye diversos modelos en diferentes *datasets*, tal y como se muestra en la Tabla 5.1. De un conjunto de datos a otro, el modelo con el mismo nombre puede ser una variante diferente.

Tabla 5.1: Modelos disponibles en los *datasets*.

Modelos	**KITTI**	**NuScenes**	**Waymo**
PoitnPillars	X	X	X
SECOND	X	X	X
PointRCNN	X		X
Part-A^2	X		X
PV-RCNN	X		X
Voxel R-CNN	X		X
CeneterPoint		X	X
CaDDN	X		
PV-RCNN++			X

OpenPCDet es una herramienta que permite entrenar y evaluar los diferentes modelos, incluso modificar los ya existentes o crear nuevos modelos a partir de objetos en Python que utilicen la flexibilidad que OpenPCDet ofrece para reutilizar módulos de otros modelos [48].

6 Metodología

Con la finalidad de efectuar todas las tareas de una manera ordenada y eficiente, se requiere utilizar una metodología que contemple todo el ciclo de vida de un proyecto software. En este trabajo en particular, se ha utilizado la metodología Scrum, la cual se detalla a continuación, para la verificación del cumplimiento de las tareas y para la agilización de su desarrollo. Además, en este capitulo se muestra el diagrama de Gantt y el desglose del presupuesto total del trabajo.

6.1 Scrum

Scrum es una metodología de trabajo ágil y de las más utilizadas para el desarrollo, no solo en la industria del software si no también en las áreas de las finanzas, la investigación, etc . Se basa en una iteración continua, donde se realizan pequeñas actividades para construir un producto o proyecto de forma incremental, entregando en cada iteración una propuesta de valor cada vez más desarrollada. Esta metodología fue diseñada para aumentar la velocidad de desarrollo, definir una cultura centrada en el rendimiento, tener una buena comunicación del rendimiento a todos los niveles y mejorar el desarrollo individual [49]. Scrum se cimienta en los siguientes pilares [50]:

- **Transparencia:** El proceso y el trabajo emergente deben ser visibles tanto para los que realizan el trabajo como para los que lo reciben. En este caso, al ser un proyecto interno, la misma empresa Ikerlan es quien lo recibe. La transparencia permite desarrollar el siguiente punto que es la inspección.
- **Inspección:** Se realiza una inspección continua del progreso para poder detectar posibles desviaciones indeseables en el objetivo.
- **Adaptación:** Si algún aspecto de un proceso se desvía fuera de los límites aceptables o si el producto resultante es inaceptable el proceso que se aplica debe ajustarse.

Con el objetivo de utilizar adecuadamente la metodología Scrum, en el equipo de SWF de Ikerlan se han definido los siguientes eventos con sus respectivas frecuencias (Figura 6.1:

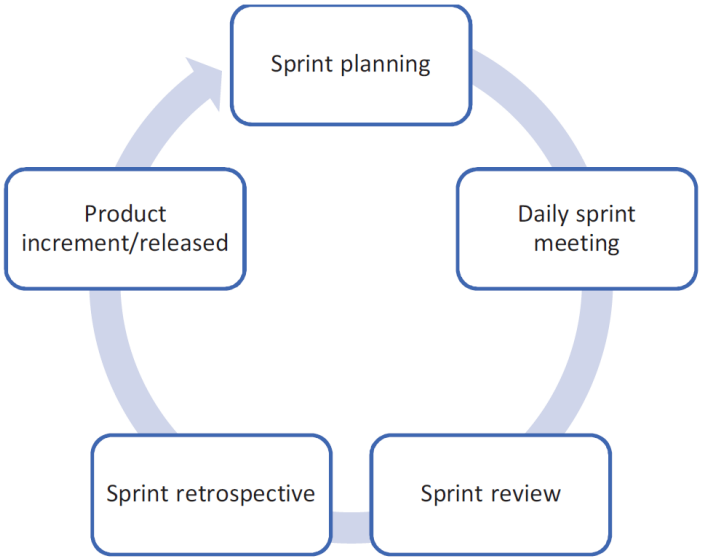


Figura 6.1: Flujo de eventos de Scrum [51].

- ***Sprint planning*:** El jefe de proyecto define cuál será el objetivo del *sprint* eligiendo tareas del proyecto y el modo de operación a seguir. De esta forma, se definen tareas concretas correspondientes al *sprint*. Esta actividad se ha realizado el primer lunes de cada mes.

- ***Daily meeting*:** Consiste en una reunión breve para que cada miembro indique las tareas en las que trabajó el día anterior, las tareas en las que trabajará ese mismo día y si existe algún problema o bloqueo que se debe solucionar.

- ***Sprint review*:** Se observan los resultados obtenidos, es decir, el estado de las tareas concretadas. Esta actividad se ha realizado el último viernes de cada mes junto con el *sprint retrospective*.

- ***sprint retrospective*:** Se realiza una evaluación de cómo se ha implementado la metodología Scrum en el último *sprint*. El resultado es una lista de mejoras que se debe aplicar en el siguiente *sprint*.

6.2 Diagrama de Gantt

La duración del Trabajo Fin de Máster ha sido de 6 meses, desde enero de 2022 hasta junio, ambos inclusive. La dedicación durante todo el periodo ha sido de 7 horas diarias, es decir, 35 horas semanales, sumando un total de 840 horas para la elaboración completa del trabajo. En la Figura 6.2 se muestra el diagrama de Gantt simplificado, donde se detallan las tareas más significativas.

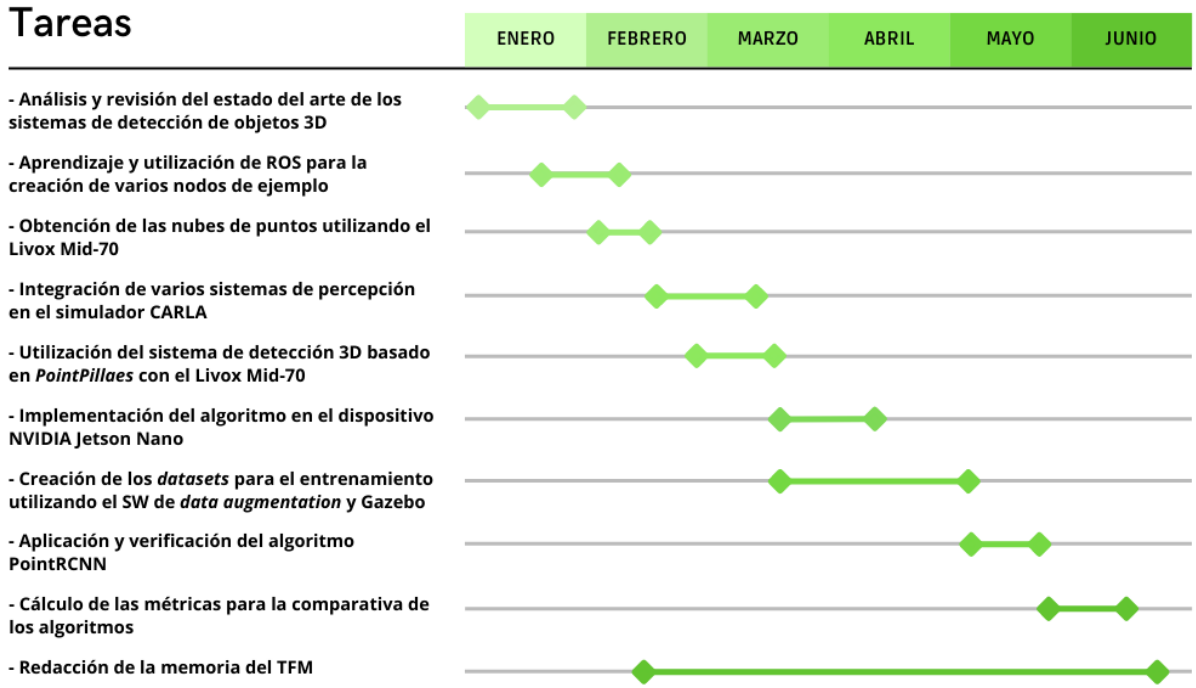


Figura 6.2: Diagrama de Gantt.

En primer lugar, se ha realizado la fase de investigación. Una vez concretado el objetivo, se ha realizado el análisis y la revisión del estado del arte de los sistemas de detección de objetos 3D, lo cual ha llevado todo el mes de enero.

A finales de enero, a la par que el estudio del estado del arte, se han ido adquiriendo conocimientos del *middleware* de ROS. Al disponer del LiDAR Livox Mid-70, se ha entendido el funcionamiento adquiriendo las primeras nubes de puntos.

Tras utilizar el LiDAR, se ha ido obteniendo manejo con el simulador CARLA para la integración de varios sistemas de percepción. Tras probar en el simulador los sistemas, se ha realizado la misma función pero, con el LiDAR Livox Mid-70.

Seguidamente, se ha implementado el algoritmo de *Deep Learning PointPillars* en el dispositivo NVIDIA Jetson Nano para realizar la inferencia utilizando su GPU. Para ello ha sido necesario convertir el modelo de Pytorch a *Open Neural Network Exchange (ONXX)* y así poder utilizarlo con TernsorRT.

La tarea de la creación de *datasets* y el entrenamiento del modelo *PointPillars* es la que más tiempo ha llevado. Para la creación de los *datasets* se han utilizado dos métodos. El primer método utiliza datos reales obtenidos del sensor, y el segundo datos sintéticos adquiridos utilizando la herramienta Gazebo. El entrenamiento se ha realizado de diferentes formas, tal y como se detalla en el Capítulo 7, sin la consecución de resultados adecuados.

Tras el entrenamiento, se ha empleado el algoritmo PointRCNN para la detección de objetos

3D, concretamente la de peatones. Además, se han calculado las métricas para la comparación de los dos algoritmos utilizados.

Por último, la redacción de la memoria aunque se prolongue mucho en el tiempo, es al final donde más horas se le han dedicado.

6.3 Presupuesto

En este apartado se especifica de forma detallada el presupuesto de la ejecución de este trabajo.

6.3.1 Coste de personal

El personal utilizado en este Trabajo Fin de Máster ha sido un ingeniero junior, el cual ha estado durante la elaboración de todo el trabajo, y dos ingenieros senior que han dado soporte en diferentes fases.

El Trabajo Fin de Máster ha tenido una duración de alrededor de seis meses. En el caso del ingeniero junior, la dedicación ha sido de 35 horas semanales, por lo que la dedicación total han sido 840 horas. Asumiendo un precio de 30 € la hora del ingeniero junior y de 50 € la hora de los ingenieros senior, el coste de la mano de obra se refleja en la siguiente Tabla 6.1:

Tabla 6.1: Coste de personal del trabajo.

Trabajador	Coste Horario (€/h)	Horas (h)	Coste (€)
Ingeniero junior	30	840	25.2000
Ingeniero/a senior A	50	80	4.000
Ingeniero/a senior B	50	30	1.500
		SUBTOTAL	**30.700 €**

Con el precio por hora anteriormente mencionado y el tiempo dedicado por cada una de las personas involucradas en el Trabajo Fin de Máster, el subtotal referente al coste de personal es de 30.700 €.

6.3.2 Amortizaciones

Las amortizaciones tienen en cuenta los equipos informáticos utilizados para el desarrollo del trabajo, así como los distintos dispositivos hardware empleados. Como en el desarrollo de este

trabajo todo el software empleado ha sido de código abierto no se ha incluido en el desglose del presupuesto (Tabla 6.2).

Tabla 6.2: Coste de las amortizaciones.

Inversión/Activo fijo	Cantidad (uds)	Coste de adquisición (€)	Vida útil (meses)	Tiempo de uso (meses)	Amortización (€)
Ordenador	1	1.400	48	6	175
LiDAR	1	1.099	48	5.5	125,93
WorkStation	1	8.000	60	2	$266,\widehat{6}$
NVIDIA Jetson Nano	1	100	42	2	4,77
				SUBTOTAL	**$527,3\widehat{6}$ €**

Por lo tanto, el coste de las amortizaciones es de $527,3\widehat{6}$ €.

6.3.3 | Coste total

En la Tabla 6.3 se presenta el presupuesto total, en el cual se han sumado los subtotales del coste de personal y el de las amortizaciones.

Tabla 6.3: Coste total del trabajo.

Concepto	Coste (€)
Coste de personal	30.700 €
Amortizaciones	$527,3\widehat{6}$ €
TOTAL	**$31.227,3\widehat{6}$ €**

Por tanto, se puede observar que el presupuesto total asciende a $31.227,3\widehat{6}$ €.

7 Desarrollo

Tras la revisión del estudio del estado del arte de los sistemas de percepción clásicos y los sistemas basados en *Deep Learning*, en este apartado se procede a explicar el desarrollo llevado a cabo para conseguir implementar, poner en marcha y validar el sistema de detección de personas.

Este capitulo está dividido en diferentes secciones. En primer lugar, se detalla el proceso necesario para obtener las nubes de puntos con el LiDAR Livox Mid-70. En el segundo apartado, se explica como se han integrado los sistemas de percepción en el simulador CARLA y se menciona como se ha realizado la transformación de ROS a ROS2. Posteriormente, se especifica como se han utilizado esos sistemas con el Livox Mid-70, y en la siguiente sección, se detalla el proceso de integración del sistema basado en *Deep Learning* en el dispositivo Jetson Nano. A continuación, se describe el procedimiento para el entrenamiento del modelo *PointPillars*, y por último, se expone el empleo del modelo PointRCNN con el LiDAR Livox Mid-70.

7.1 Obtención de los datos 3D

En primer lugar y para la obtención de la nube de puntos proporcionada por el LiDAR es necesaria la conexión al ordenador. El Livox Mid-70 utiliza un conector de aviación M12 para la alimentación, la señal de control y la transmisión de datos, el cual se conecta al *Livox Converter 2.0*. Este convertidor contiene un puerto para el LiDAR, un puerto de sincronización, un puerto de alimentación y un puerto Ethernet [33]. Además, se requiere una conexión Ethernet para la transmisión de los datos mediante UDP (*User Protocol Diagram*). La conexión al ordenador se ha realizado a través de la IP estática que proporciona el LiDAR por defecto. La Figura 7.1 muestra el esquema simplificado de la conexión del LiDAR con la computadora.

Una vez establecida la conexión, el fabricante proporciona una serie de herramientas para obtener las nubes de puntos. La primera de las herramientas es el Livox SDk, que es el kit de desarrollo software diseñado para todos los productos Livox. Está desarrollado en lenguaje C/C++ y proporciona una API de estilo C [52]. Esta herramienta consiste en el protocolo de comunicación, el núcleo de Livox SDK y la propia API.

La segunda y última herramienta proporcionada por el fabricante es el Livox ROS Driver, el cual

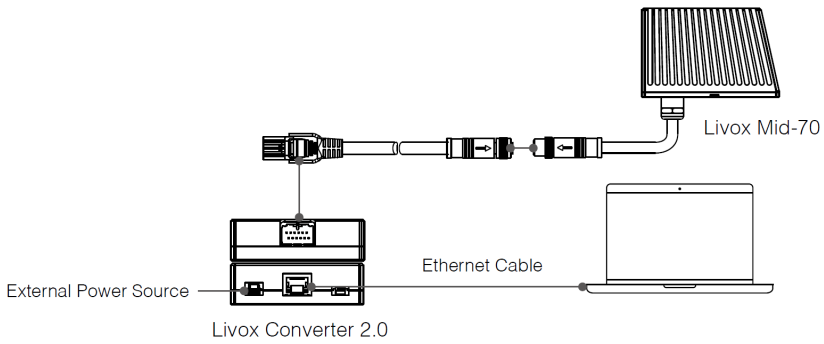


Figura 7.1: Conexión del LiDAR al PC [33].

es un paquete de ROS utilizado especialmente para la conexión de sensores LiDAR producidos por el fabricante Livox. Este driver dispone de varios archivos de lanzamiento que se adaptan a distintos escenarios. Dichos archivos contienen parámetros de configuración con los que se puede determinar, por ejemplo, la frecuencia de publicación del *topic* de la nube de puntos o el formato de salida de la nube [53]. También contiene archivos de configuración para establecer el sistema de coordenadas, el código *broadcast* del LiDAR al que se quiere conectar, etc.

El formato elegido para representar la nube de puntos es el correspondiente al de la librería *Point Cloud Library* (PCL) [54], donde se define como el estándar del mensaje el *PointCloud2* de ROS, el cual se emplea para guardar información de una colección de puntos en N dimensiones. En la Tabla 7.1 se detalla la estructura del mensaje.

Tabla 7.1: Estructura del mensaje PointCloud2 [55].

Tipo	Nombre
Header	header
uint32	heigth
uint32	width
PointField[]	fields
bool	is_bigendian
uint32	point_step
uint32	row_step
uint8[]	data
bool	is_dense

Mediante este tipo de mensaje, la nube de puntos viene definida como un array de bytes en formato *little endian*, en la que cada número es representado por 32 bits. Estos números representan las posiciones x,y,z y la intensidad (*intensity*) de cada punto. Por lo tanto, la nube de puntos es definida como un vector de 4 dimensiones en un sistema de coordenadas cartesiano.

Con el parámetro *intensity* anteriormente mencionado se obtiene la información de la intensidad del haz reflejado en ese punto, lo cual indica la reflectividad del objeto del entorno.

7.2 | Implementación de los sistemas en CARLA

Antes de utilizar la nube de puntos generada por el LiDAR Mid-70, se comienza por integrar un sistema de detección clásico en el simulador CARLA con el objetivo de adquirir manejo con el simulador y conocimientos del LiDAR que viene incorporado, para proseguir con la implementación de un modelo basado en *Deep Learning*.

7.2.1 | LiDAR de CARLA

Con la finalidad de integrar los dos sistemas mencionados anteriormente en el simulador CARLA, antes de nada hay que analizar y comprender el funcionamiento del LiDAR que incorpora. La comunicación con CARLA se realiza con el *CARLA ROS bridge*, cuya función es establecer una comunicación bidireccional entre ROS y CARLA. De esta forma, la información proveniente del simulador es trasladada a *topics* de ROS y los mensajes mandados entre nodos de ROS son trasladados a comandos que se aplican en CARLA [56]. En este caso, la información del sensor de CARLA proviene del *topic /carla/ego_vehicle/lidar*, que contiene el tipo de mensaje *sensor_msgs/PointCloud2* detallado anteriormente.

7.2.2 | Sistema de percepción clásico

Tras analizar varias técnicas clásicas correspondientes a los sistemas de percepción en el apartado 4.1, se ha integrado una aplicación de detección de objetos 3D en el simulador CARLA. Dicha aplicación se ha obtenido del repositorio en [57].

La aplicación ha sido implementada en lenguaje C++, por lo que ha sido necesaria la utilización de la librería de ROS *roscpp* para establecer la comunicación con el *CARLA ROS bridge* y obtener la nube de puntos del LiDAR. El programa es un nodo de ROS que se subscribe al *topic /carla/ego_vehicle/lidar* y publica en otro *topic* las detecciones de los objetos del entorno mediante *bounding boxes* 3D para su visualización mediante RViz. El programa sigue la siguiente secuencia para obtener finalmente las cajas delimitadoras de los objetos más trascendentes del entorno:

1. Obtención del mensaje de tipo *PointCloud2* del simulador CARLA y transformación a una estructura C++ de la librería PCL [54].

2. Voxelización de la nube de puntos obtenida.
3. Filtrado de la cuadrícula estructurada en función de la región de interés.
4. Aplicación del algoritmo RANSAC para la exclusión del plano del suelo.
5. Creación de un KD-tree tridimensional, donde se han introducido los puntos de la nube según sus coordenadas.
6. *Clusterización* de los vóxeles utilizando el KD-tree y fijando una distancia máxima entre vóxeles.
7. Filtrado de los *clusters* por número de vóxeles, tamaño y volumen.
8. Creación de las *bounding boxes* 3D correspondientes a los objetos detectados.

La mayoría de las funciones han sido obtenidas de la librería PCL y los parámetros, como el tamaño del vóxel o la región de interés, entre otros, se han adecuado para un correcto funcionamiento del LiDAR de 32 haces de CARLA. EL resultado de la detección se detalla en el capitulo 8.2.

7.2.3 | Sistema de percepción basado en *Deep Learning*

Tras comprobar los resultados de la aplicación del sistema clásico se ha implementado un sistema de percepción basado en un modelo de *Deep Learning*, que es la tendencia de los sistemas del estado del arte más reciente. El programa que se ha implementado ha sido creado por el mismo autor del anterior programa del sistema de percepción clásico y se ha obtenido en [58]. Dicho trabajo se ha basado en otro repositorio al que se le ha aplicado una refactorización del código para facilitar su mantenimiento y su reutilización.

Para la implementación de los modelos se ha utilizado la herramienta OpenPCDet explicada en el apartado 5.2.8. El modelo de *Deep Learning* escogido para la implementación de la aplicación de detección de objetos 3D ha sido *PointPillars*, sobre todo por su altísima velocidad de inferencia en comparación con otros modelos y su mayor facilidad para integrarlo en un sistema embebido, tal y como se puede observar en la sección 8.1.

Esta aplicación es otro nodo de ROS, que en este caso tiene más *subscribers* y *publishers* para obtener toda la información necesaria de CARLA. Además, también carga el modelo escogido y realiza las funciones convenientes para crear las cajas delimitadoras. Para ejecutar el programa se utiliza la herramienta *roslaunch* especificando el archivo *launch* y algún parámetro si se requiere su modificación a la hora de la ejecución. En paralelo, se pone en funcionamiento el

simulador CARLA junto con el *CARLA ROS bridge* y un vehículo a controlar. Al ser necesarias ciertas librerías que requieren de GPU para utilizar la herramienta OpenPCDet y, al mismo tiempo, utilizar CARLA, lo cual consume bastante memoria gráfica, el programa se ha lanzado en la WS explicada en el apartado 5.1.3. El funcionamiento del programa es el siguiente:

1. Carga de la red neuronal *PointPillars* en Pytorch junto con sus pesos obtenidos de OpenPCDet en la GPU.
2. Mediante un *subscriber* se guarda la información de la odometría del vehículo cada vez que se recibe un mensaje desde el *topic*.
3. Guardado de la nube de puntos del LiDAR junto con el *timestamp* mediante otro *subscriber*.
4. Ejecución del modelo y obtención de las predicciones de los objetos detectados.
5. Filtrado de las predicciones en función de cada clase (coche, peatón, ciclista).
6. Transformación de la velocidad relativa al coche que lleva el LiDAR a la velocidad absoluta utilizando la odometría almacenada.
7. Creación de las cajas delimitadoras 3D y las flechas de la velocidad para ser publicadas en diferentes *topics* mediante *publishers*.

En la sección 8.2 se muestran los resultados obtenidos con el modelo *PointPillars* sobre el simulador CARLA.

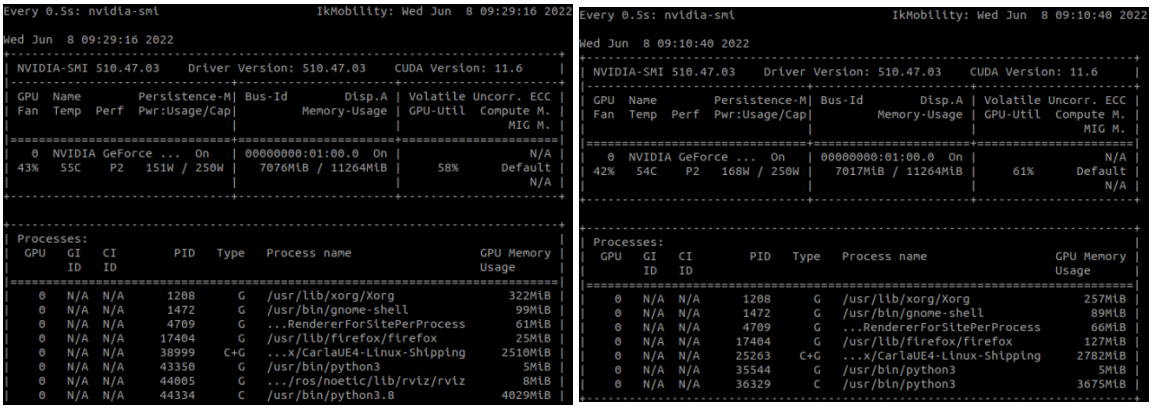
7.2.4 | Transformación de ROS a ROS2

Como el programa obtenido del repositorio para la aplicación de la detección de objetos basado en el modelo *PointPillars* se ha creado con la versión ROS1, se ha realizado una conversión para poder utilizarla en ROS2. Esto se debe a que al ser la versión actualizada de ROS adquiere ciertas funcionalidades como requisitos de tiempo real, agrupaciones de más de un robot, etc [59]. Además, el equipo de STF de Ikerlan apuesta por esta nueva versión.

Para la conversión, primero de todo se ha comprobado si los paquetes empleados de ROS1 están implementados en ROS2 y, si no es así, se ha analizado cómo poder sustituirlos. Por ejemplo, en lugar de utilizar la librería *rospy* de ROS para python en ROS2 se utiliza *rclpy*. Ello conlleva que hay que modificar todas las funciones de ROS a la nueva librería de ROS2 utilizando su API. Por otro lado, además de tener que cambiar los nombres de los mensajes utilizados, también se

ha modificado el archivo *launch* al no existir, por ejemplo, los parámetros globales, puesto que tienen que ir anidados en los nodos de ROS2 [60].

Con el objetivo de verificar que la transformación se ha realizado exitosamente, se ha comprobado que la aplicación funciona correctamente, tal y como lo hacía en ROS1. Además, a la hora de ejecutar el programa se ha detectado una peculiaridad en la ocupación de la memoria de la GPU, ya que la versión de ROS2 ocupa algo menos. Esta diferenciación se puede observar comparando la ocupación del programa python en las Figura comparativa 7.2 (en ROS1 ocupa 4 GB mientras que en ROS2 ocupa 3.6 GB).



(a) Ocupación de memoria GPU en ROS1.

(b) Ocupación de memoria GPU en ROS2.

Figura 7.2: Comparación de memoria GPU entre ROS1 y ROS2.

Aún obteniendo estas mejoras, se ha optado por continuar utilizando ROS1, puesto que varias funcionalidades todavía no han sido implementadas en ROS2. Una de ellas es la función que posibilita la trasformación de un *rosbag* (fichero de ROS para guardar mensajes de *topics*) o de un *topic* directamente a formato PCD (*Point Cloud Data*) de la librería PCL.

7.3 Utilización de los sistemas con el LiDAR

Tras comprobar el funcionamiento del sistema clásico y el basado en *Deep Learning* en el simulador, se ha analizado el comportamiento de los mismos utilizando el LiDAR Livox Mid-70 y su patrón de escaneo no repetitivo. Para ello, al utilizar el mismo tipo de mensaje que el LiDAR de CARLA, ha sido suficiente cambiar el nombre del *topic* al que se subscribe el nodo ha sido suficiente para poner los sistemas en funcionamiento. A su vez, se ha ido ajustando la frecuencia con la que publica cada mensaje en el *topic* para poder acumular más puntos y que la nube de puntos sea algo más nítida.

Se han ido ajustando los parámetros como el tamaño del vóxel o el número máximo de puntos por vóxel en función de las características del LiDAR para adecuar la detección. Además,

también se han modificado ciertos parámetros según el caso de uso. En el laboratorio el rango de la nube de puntos se ha limitado a sus dimensiones, es decir 25 m de largo y 8 m de ancho. En el apartado 8.3 se puede observar y comparar la disparidad de los distintos sistemas, incluso comparándolos con los resultados obtenidos con el simulador.

7.4 | Integración del sistema en la Jetson Nano

Al ser una aplicación totalmente enfocada a los sistemas autónomos, el poder integrarlo en un sistema embebido es una de las tareas primordiales. Por ello, en este apartado se explica como ha sido el proceso de integración del modelo en una Jetson Nano, la cual es ideal para aplicaciones de inteligencia artificial, puesto que dispone de una GPU de NVIDIA tal y como se detalla en el apartado 5.1.2.

7.4.1 | Adquisición de los datos a través del dispositivo Jetson Nano

Como primera aproximación, conectando el LiDAR directamente al dispositivo Jetson Nano se ha comprobado la posibilidad de obtener las nubes de puntos. Para ello, se han instalado los paquetes necesarios, tales como el *Livox ROS driver*, y se ha establecido una IP estática para la comunicación con el LiDAR. Al verificar que se obtienen los datos satisfactoriamente, se ha realizado la conexión que se muestra en la Figura 7.3 para realizar la inferencia en la estación de trabajo mientras se trasladan los datos del sensor desde el dispositivo Jetson Nano.

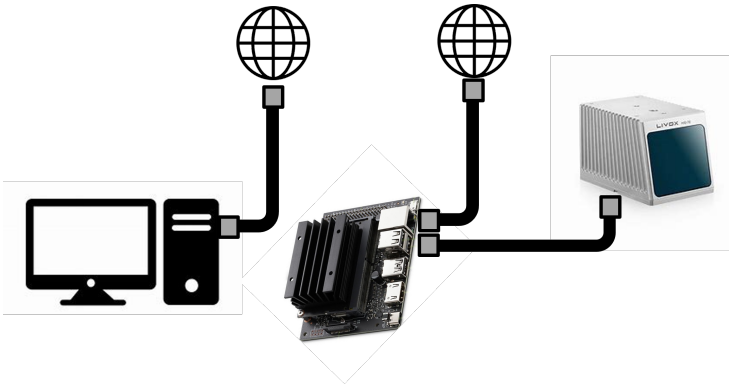


Figura 7.3: Esquema de conexión entre Jetson Nano, LiDAR y WS.

El LiDAR está conectado a la placa a través de un adaptador USB/Ethernet, puesto que solo dispone de un puerto Ethernet, el cual se utiliza para conectarse a la red. La WS también está conectada a la misma red y la trasferencia de la nube de puntos del dispositivo Jetson Nano a

la WS se realiza mediante ROS. Con el objetivo de compartir datos entre dispositivos que están conectados a la misma red utilizando ROS y funcione en varios ordenadores al mismo tiempo, es necesario realizar la configuración mediante ciertos comandos. Se establece cual va a ser el maestro, en este caso el dispositivo Jetson Nano, para ejecutar el *roscore* que es indispensable para la comunicación entre nodos ROS, y se exporta su IP para que sea visible para los demás dispositivos de la red. En la estación de trabajo se establece que el maestro es el otro dispositivo y se exporta su IP. Una vez seguidos estos pasos, la WS ya se puede subscribir al *topic* donde se están publicando los datos del LiDAR en la placa y así realizar la inferencia con el modelo *PointPillars*, la cual se puede observar en la Figura 7.4.

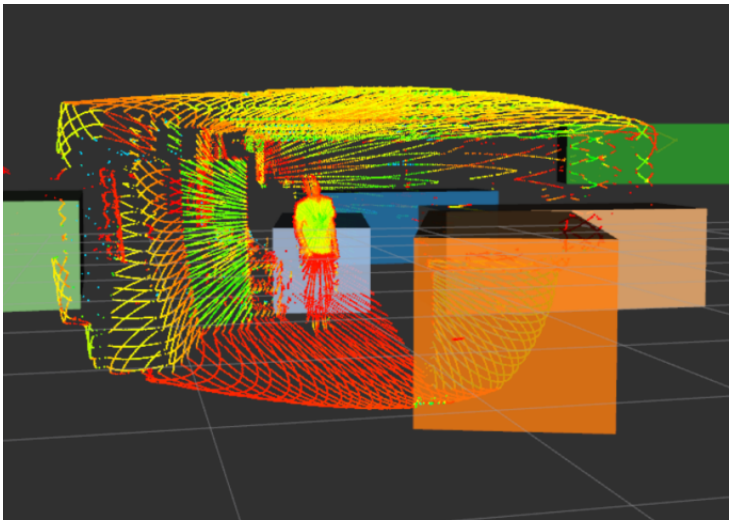


Figura 7.4: Inferencia con *PointPillars* a través de la Nano.

Uno de los potenciales de ROS es que se podrían subscribir muchos nodos al mismo *topic* y realizar diferentes operaciones en paralelo únicamente estando conectados a la misma red. Además, al utilizar *Data Distribution Service* (DDS) en ROS2 y no necesitar el *roscore* para la comunicación entre nodos, esta conexión se realizaría automáticamente sin utilizar ningún comando con por estar en la misma red.

7.4.2 | Inferencia en el dispositivo Jetson Nano

Una vez realizados los pasos anteriores, es momento de integrar el modelo *PointPillars* en el dispositivo hardware para realizar directamente la inferencia utilizando su GPU. Al ser un producto de NVIDIA se puede utilizar TensorRT, que es un SDK que incluye un optimizador de aprendizaje profundo y un tiempo de ejecución con baja latencia y alto rendimiento para aplicaciones de inferencia. Sin embargo, como el modelo *PointPillars* se encuentra bajo el *framework* PyTorch la conversión no es directa. Aquí es donde entra en juego Open Neural Network Exchange (ONXX). ONNX es una librería diseñada para orientar la interoperabilidad de

los *framework* y la accesibilidad de la optimización entre otras funcionalidades. De esta forma, ONXX se utiliza como un formato intermedio entre los dos *framework* para poder realizar la conversión.

Para trasladar el modelo de PyTorch a ONNX, se utiliza el comando *torch.onnx.export* que requiere los siguientes argumentos: el propio modelo preentrenado, el tensor con el mismo tamaño que los datos de entrada, el nombre del archivo ONNX y los nombres de las entradas y salidas. En este caso, hay tres tensores de entrada, uno que representa el máximo número de vóxeles, otro que es de dos dimensiones y especifica los valores xyz y reflectividad del vóxel y, por último, uno de tres dimensiones que contiene la información de los anteriores y el máximo número de puntos por vóxel. En cambio, las salidas son la clase del objeto predicho, sus coordenadas y la caja delimitadora predicha.

Una vez que se ha obtenido el modelo en formato ONNX, ya se puede realizar la conversión a TensorRT y, de esta forma, realizar la inferencia directamente en el dispositivo Jetson Nano. El resultado obtenido de los tiempos de inferencia se muestra en la siguiente Figura 7.5. Se ha realizado la inferencia sobre diferentes escenarios de nubes de puntos y se puede observar el tiempo transcurrido en cada etapa, con un tiempo total de entre 300 y 400 ms. Además del tiempo de inferencia, también muestra cuantas cajas delimitadoras han sido creadas.

```
GPU has cuda devices: 1
----device id: 0 info----
  GPU : NVIDIA Tegra X1
  Capbility: 5.3
  Global memory: 3964MB
  Const memory: 64KB
  SM in a block: 48KB
  warp size: 32
  threads in a block: 1024
  block dim: (1024,1024,64)
  grid dim: (2147483647,65535,65535)

load TRT cache.
<<<<<<<<<<<
load file: /home/livox/livox_data/000000.bin
Datu fitxategiaren luzeera: 1741248find points num: 108828
find pillar_num: 3027
TIME: generateVoxels: 2.21932 ms.
TIME: generateFeatures: 16.1099 ms.
TIME: doinfer: 358.476 ms.
TIME: doPostprocessCuda: 4.68188 ms.
TIME: pointpillar: 382.875 ms.
Bndbox objs: 10
>>>>>>>>>>>
<<<<<<<<<<<
load file: /home/livox/livox_data/000001.bin
Datu fitxategiaren luzeera: 1644736find points num: 102796
find pillar_num: 3123
TIME: generateVoxels: 0.832917 ms.
TIME: generateFeatures: 2.98266 ms.
TIME: doinfer: 308.95 ms.
TIME: doPostprocessCuda: 4.67307 ms.
TIME: pointpillar: 321.654 ms.
Bndbox objs: 10
>>>>>>>>>>>
```

Figura 7.5: Inferencia en la Jetson Nano

7.5 | Entrenamiento del modelo *PointPillars*

En este apartado se detallan los procedimientos que se han seguido para entrenar la red neuronal *PointPillars*, puesto que los resultados obtenidos no han sido los deseados para el LiDAR Mid-70 al no realizar una detección muy depurada como se ha mostrado en la sección anterior. Con el objetivo de entrenar el modelo, se ha optado por tomar dos direcciones a la hora de construir el *dataset*. Una de ellas ha consistido en la creación de una aplicación software de *data augmentation* utilizando las nubes de puntos obtenidas del LiDAR. La ha consistido en la utilización de datos sintéticos mediante el uso de la herramienta Gazebo.

7.5.1 | Software de data augmentation

El *data augmentation* o el aumento de datos es una parte esencial de los procesos de entrenamiento de cualquier modelo de clasificación y detección de objetos [61]. El objetivo del aumento de datos es mejorar la generalización y hacer que las redes se vuelvan invariables respecto a la rotación, la traslación y los cambios naturales en los valores de los píxeles en el caso de las imágenes y los puntos en el de las nubes de puntos.

Los métodos de *data augmentation* aplicados a las nubes de puntos se inspiran principalmente en los métodos diseñados para las imágenes. Estos métodos deben ser compatibles con las leyes físicas que rigen la propagación de los rayos láser, como por ejemplo que a medida que se va aumentando la distancia respecto al sensor la densidad de la nube de puntos vaya disminuyendo. Por lo tanto, si el aumento de datos no sigue estas leyes, no estaría imitando un escenario del mundo real.

Tal y como se puede observar en la Figura 7.6, algunos de los métodos más comunes para el aumento de datos son la rotación alrededor del eje z del marco de coordenadas del LiDAR, el volteo con respecto a los planos x-z e y-z, la dispersión de los puntos, el ruido Gaussiano aditivo aplicado a los puntos y el aumento de la escena con nubes de puntos de objetos. Para la creación de la aplicación se han escogido algunos de estos métodos.

7.5.1.1 Creación de los escenarios

En primer lugar, partiendo de la nube de puntos del laboratorio obtenida mediante el LiDAR Mid-70, se han ido aislando los objetos que se pueden encontrar en este lugar. Para ello, el objeto se posiciona en un lugar aislado o bastante espacioso donde no haya nada que pueda obstruir la nube de puntos del objeto. De esta forma, utilizando la función *crop* de la clase *PointCloud* de

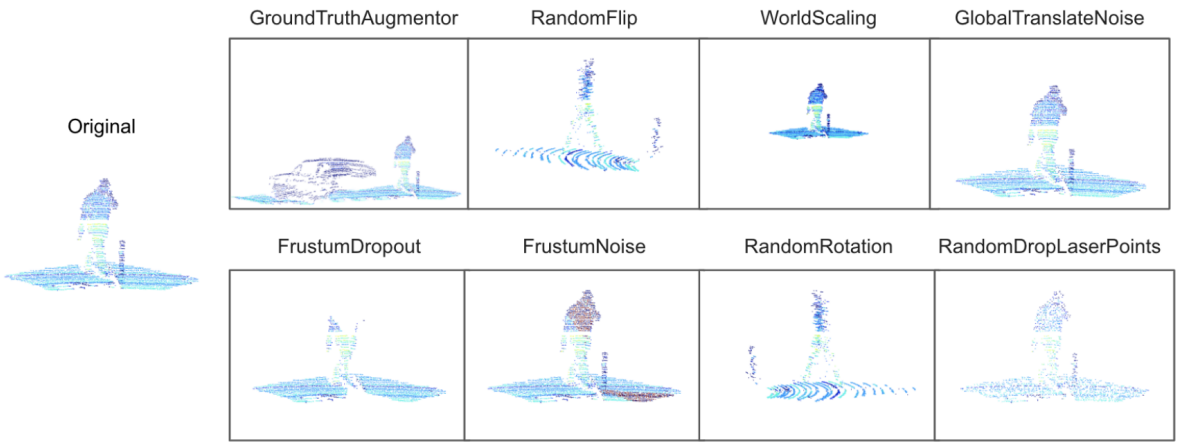


Figura 7.6: Métodos de *data augmentation* aplicados a nubes de puntos [61].

la librería Open3d se recorta la nube de puntos en el lugar de interés para quedarse únicamente con los puntos correspondientes al objeto. Por último, para que la localización de los objetos en la nube de puntos de los escenarios sea más sencilla, estos se han movido al origen de las coordenadas del sensor. Se ha realizado exactamente la misma operación con el suelo y con personas en diferentes posturas, tal y como se puede apreciar en la Figura 7.7.

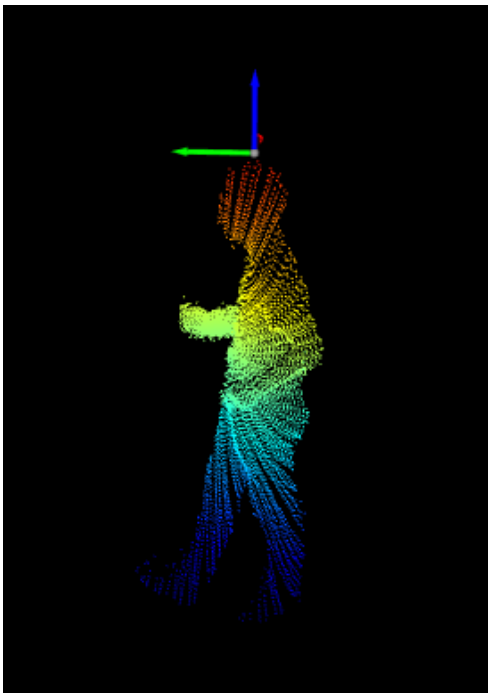


Figura 7.7: Nube de puntos de una persona aislada.

Una se han obtenido las nubes de puntos de todos los objetos y las personas, se ha procedido a crear varios escenarios con ellos. Para ello, se han ido colocando en el suelo, el cual también ha sido aislado, en posiciones aleatorias creando un valor de x e y para cada elemento. Con

el propósito de que no existan solapamientos entre objetos, esas posiciones aleatorias han sido analizadas priorizando la creación de un nuevo valor en el eje y para que así no sé encuentre un objeto justo detrás de otro. Este proceso se repite una y otra vez hasta que no existan solapamientos o hasta que tras ciertas iteraciones se cambia el valor del eje x, el cual tiene un rango más amplio por lo que, facilita la ausencia de solapamiento. Aunque se cambie el valor del eje x el proceso de comprobación debe comenzar de nuevo.

Tras el posicionamiento aleatorio, también se introduce una rotación aleatoria de 0 a 90º a cada elemento. No ha sido necesario un rango de rotación mayor, ya que a partir de los 90º la nube de puntos es prácticamente la misma, es decir, que en el caso de las nubes de puntos, una persona de frente o de espaldas se aprecia de la misma forma. Para la rotación, se ha multiplicado cada punto de cada objeto con la matriz de rotación del eje Z (7.1) como se muestra en (7.2)-(7.5)

$$R_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7.1}$$

$$\begin{pmatrix} x\prime \\ y\prime \\ z\prime \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{7.2}$$

Donde $x\prime$, $y\prime$ y $z\prime$ son:

$$x\prime = xcos\theta - ysin\theta \tag{7.3}$$

$$y\prime = xsin\theta + ycos\theta \tag{7.4}$$

$$z\prime = z \tag{7.5}$$

Por último, utilizando la función *random_down_sample*, también de la clase *PointCloud*, se ha disminuido la densidad de la nube de puntos en función de la distancia del sensor. Está función realiza un submuestreo de la nube de puntos eliminando puntos aleatorios. En este proceso no entra el suelo, puesto que ya tiene es disminución de la densidad respecto a la distancia por defecto. Tras este procesado de la nube de puntos ya se pueden generar miles de escenarios diferentes como el que se muestra en la Figura 7.8.

Ha sido necesario bajar la altura de la nube de puntos, o lo que es lo mismo subir el LiDAR, a la altura en la que se encuentra en el *dataset* KITTI (1.7 m aproximadamente). Esto se debe a que al estar el modelo previamente entrenado utilizando los datos de KITTI, realiza una detec-

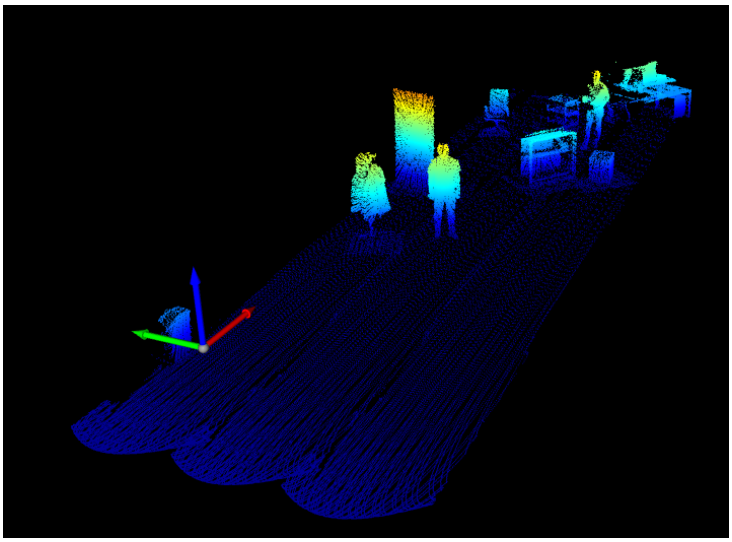


Figura 7.8: Escenario creado con el SW de aumento de datos.

ción mucho más precisa si se encuentra a esa altura, tal y como se ha comprobado pasándole escenarios del *dataset* creado con el aumento de datos a diferentes alturas. De esta forma, se ha creado una aplicación a la que se pueden introducir varios elementos aislados para generar distintos escenarios automáticamente, lo cual puede ser interesante para poder crear *datasets*.

7.5.1.2 Creación de las etiquetas

En el aprendizaje supervisado para poder entrenar el modelo, además de necesitar muchos escenarios, también es necesario que estos estén etiquetados. En este caso, como el espacio se limita a un laboratorio, de las clases mencionadas únicamente se encontrarán peatones. Al basarse en el *dataset* KITTI, se ha mantenido la estructura del *dataset* para crear las mismas (Tabla 7.2).

Como únicamente se dispone de las nubes de puntos de los escenarios y estas se encuentran representadas en tres dimensiones, únicamente se ha prestado atención a los parámetros correspondientes a las cajas delimitadoras 3D. Para obtener dicha información se ha utilizado la combinación de las clases *AxisAlignedBoundingBox* y *OrientedBoundingBox*. Para ello, en primer lugar se ha creado una caja delimitadora alineada a los ejes con *AxisAlignedBoundingBox* cuando todavía no se le ha dado una rotación aleatoria a la persona. De esta forma, la caja siempre tendrá las mismas dimensiones aunque se rote la persona. El único inconveniente es que este tipo de *bounding box* no se puede rotar. Por este motivo entra en juego la otra clase que define una caja delimitadora orientada y puede ser rotada. De esta manera, se crea una caja delimitadora orientada a partir de la anterior y se le da exactamente la misma rotación que a la persona. El resultado es el que se observa en la Figura 7.9.

Una vez creados los *bounding box*, se extrae la información de las dimensiones y de las localiza-

Tabla 7.2: Estructura de las etiquetas de KITTI.

Num elem.	Parameter name	Description	Type	Range	Example
1	Class names	The class to which the object belongs	String	N/A	pedestrian, car, cyclist
1	Truncation	How much of the object has left image boundaries	Float	0=yes 1=no	1
1	Oclussion	Occlusion state [0=fully visible, 1=partly visible, 2=largely occluded, 3=unknown]	Integer	[0,3]	2
1	Alpha	Observation angle of object	Float	[-pi, pi]	0.146
4	2D bounding box coordinates	Location of the object in the image [xmin, ymin, xmax, ymax]	Float(0 based index)	N/A	100, 120, 180, 160
3	3D dimension	Height, width, length of the object (in meters)	Float	N/A	1.65, 1.67, 3.64
3	Location	Object location x, y, z in camera coordinates (in meters)	Float	N/A	-0.65, 1.71, 46.7
1	Rotation	Rotation ry around the Y-axis in camera coordinates	Float	[-pi, pi]	-1.59

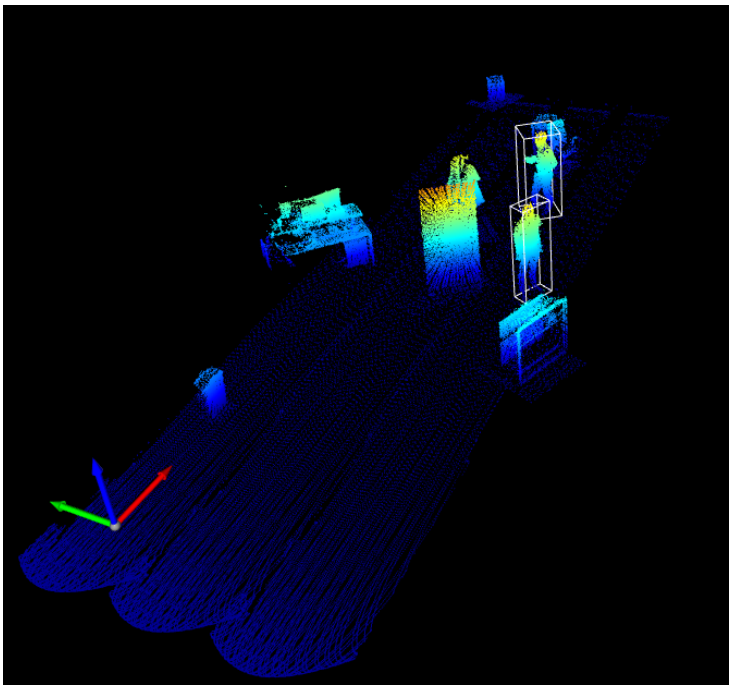


Figura 7.9: Escenario con las cajas delimitadoras.

ciones, donde la localización corresponde al centro de la caja. También se adquiere la rotación que se le ha sido asignada para completar los datos necesarios de la etiqueta. Evidentemente, en el primer elemento de la etiqueta se especifica la clase del objeto, que en este caso es *Pedestrian*.

7.5.2 | Creación de escenarios con Gazebo

Con el objetivo de obtener las nubes de puntos utilizando Gazebo, en primer lugar es necesario integrar el mismo tipo de LiDAR en el entorno. Para ello, se ha utilizado un *plug-in* con el que se ha introducido el sensor Livox Mid-70. Para poder dar la altura deseada a la nube de puntos, puesto que es determinante tal y como se ha explicado anteriormente, se ha creado una estructura con URDF (*Universal Robotic Description Format*) para, así, poderlo incorporar en Gazebo. Esta estructura, ha sido utilizada para colocar el sensor a la altura de aproximadamente 1.7 m. Como escenario se ha utilizado un modelo de cafetería, la cual ha sido vaciada para poder ubicar los objetos deseados de forma aleatoria. Se ha utilizado la herramienta Blender para la integración de distintos modelos de personas, el cual es un paquete de creación 3D gratuita y de código abierto que soporta todo el proceso de creación 3D. De esta forma, se obtiene la información de las dimensiones de los modelos, lo cual es indispensable para la creación de las etiquetas.

Mediante un archivo *launch* se especifica cual va a ser el punto de partida, es decir, que se carga el escenario de la cafetería con la posición exacta del LiDAR, la cual siempre será la misma al inicio de la ejecución. Tras iniciar el escaneo del LiDAR, se crea un número aleatorio de personas y de objetos que se van ubicando en el espacio aleatoriamente. Cada vez que se obtiene una nube de puntos del escenario, el LiDAR se cambia de posición en el eje horizontal y en el de orientación. De esta manera, al conocer la orientación y el ángulo de la cobertura de visión del sensor, se puede determinar qué elementos se encuentran fuera del campo de visión y obviarlos (Figura 7.10) .

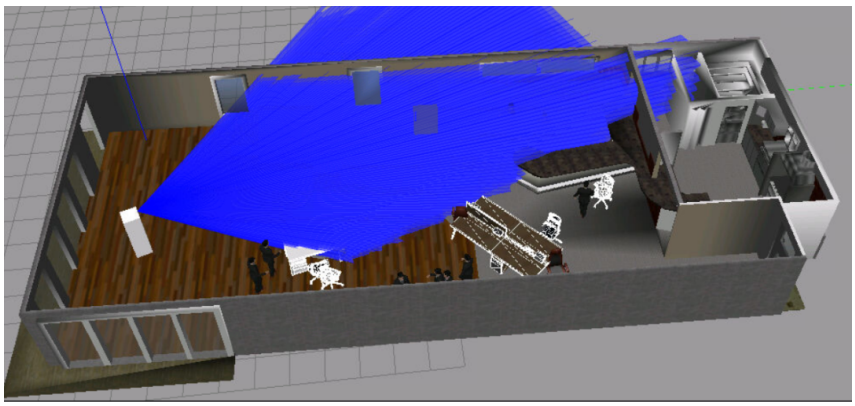


Figura 7.10: Escenario creado en Gazebo con elementos colocados aleatoriamente.

Para el entrenamiento supervisado se crean las etiquetas necesarias, al igual que con el software de aumento de datos, pero sin necesidad de crear las cajas delimitadoras. Al conocer la ubicación donde han sido colocados, la rotación que se les ha asignado y las dimensiones al utilizar Blender, se crean las etiquetas directamente. La nube de puntos que da como resultado este proceso es muy parecida a la obtenida con el LiDAR real, tal y como se puede observar en la Figura 7.11.

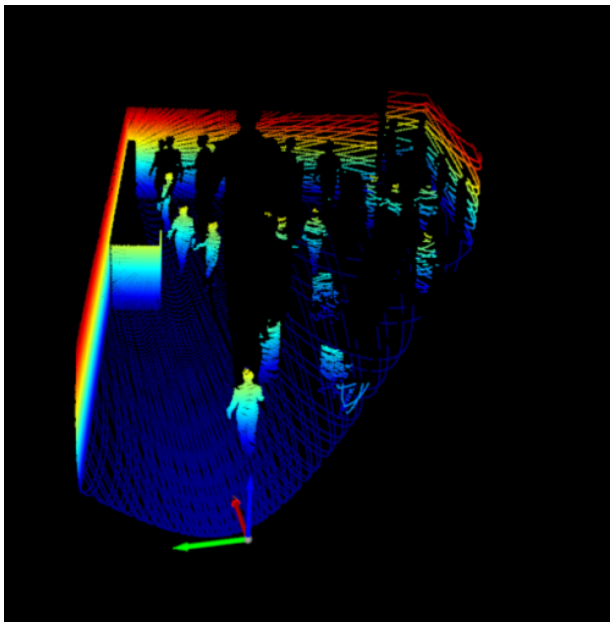


Figura 7.11: Nube de puntos obtenida utilizando Gazebo.

7.5.3 | Estructura y formato de los ficheros

A la hora de entrenar el modelo PointPillars utilizando la herramienta OpenPCDet sin realizar grandes modificaciones en los scripts para el entrenamiento, el *dataset* tiene que tener una estructura concreta y los ficheros un formato concreto. Al utilizar el *dataset* KITTI, la estructura empleada es la que se muestra en la Figura 7.12.

```
OpenPCDet
├── data
│   ├── kitti
│   │   ├── ImageSets
│   │   ├── training
│   │   │   ├──calib & velodyne & label_2 & image_2 & (optional: planes) & (optional: depth_2)
│   │   ├── testing
│   │   │   ├──calib & velodyne & image_2
├── pcdet
├── tools
```

Figura 7.12: Estructura del dataset.

El *dataset* consta de tres directorios. El de *training* contiene la carpeta velodyne, donde se almacenan las nubes de puntos en formato binario con la extensión .bin. La carpeta *label_2* contiene las etiquetas en formato de texto (.txt), el directorio de *image_2* con las imágenes de la cámara en formato png y la carpeta *calib* en formato de texto también con la información necesaria para la calibración de los sensores. El directorio de *testing* es prácticamente igual, con la diferencia de que no se dispone de las etiquetas. Por último, en *ImageSets* se encuentran los archivos de texto para indicar cuantos escenarios o muestras se van a utilizar a la hora de entrenar y testar. Este corresponde con el número de archivos que se encuentran dentro de los directorios de *testing* y *training*.

En el *dataset* propio únicamente se utilizan los datos adquiridos por el LiDAR, por lo que no se dispone de imágenes ni de archivos de calibración. Para seguir manteniendo la misma estructura, estos directorios han sido completados con imágenes y archivos de calibración genéricos de KITTI. Tanto la localización como la rotación de las etiquetas para las cajas delimitadoras en 3D se tienen que representar en coordenadas de la cámara de KITTI, tal y como se ha mostrado en la tabla 7.2. Con dicha finalidad, se ha realizado la debida transformación para los elementos mencionados de las etiquetas teniendo en cuenta el sistema de coordenadas del LiDAR, en este caso el Mid-70, y la cámara de KITTI. Los dos sistemas de coordenadas se muestran en la Figura 7.13.

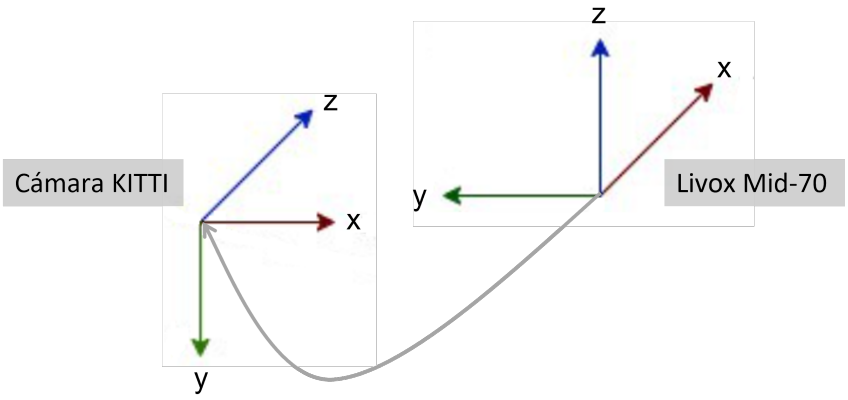


Figura 7.13: Transformación del sistema de coordenadas.

Con el objeto de realizar menos transformaciones, también se ha eliminado todo lo correspondiente a los archivos de calibración y a las imágenes en los *scripts* del entrenamiento. De esta forma, no ha sido necesario disponer de los directorios de imágenes ni de calibración, además de que se introducen los datos del LiDAR sin modificaciones, es decir, sin tener que realizar ningún tipo de transformación para la creación de las etiquetas.

7.6 | Sistema de percepción con PointRCNN

Tras comprobar que con el sistema de percepción del modelo *PointPillars* y el LiDAR Mid-70 no se ha conseguido una detección de las personas muy exacta, se ha optado por probar el modelo PointRCNN, el cual se explica en el apartado 4.2. Como entrada, este modelo recibe la nube de puntos en crudo, es decir, todos los puntos de la nube de puntos sin realizar ningún tipo de voxelización previa, como en el caso de *PointPillars*. En primer lugar, se han verificado los resultados de la inferencia ante escenarios del software de aumento de datos, o directamente de la nube de puntos obtenida del laboratorio utilizando la herramienta OpenPCDet.

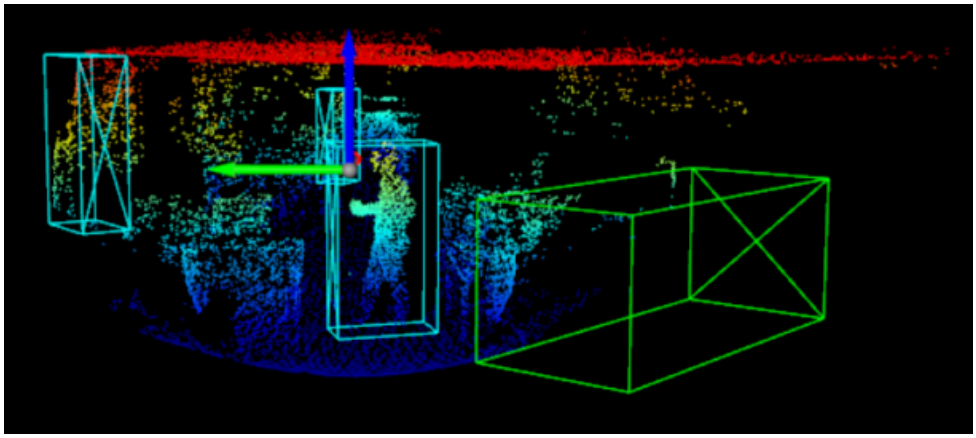


Figura 7.14: Inferencia del modelo PoinRCNN en el laboratorio.

En la Figura 7.14 se puede apreciar que con el modelo PointRCNN, se crea perfectamente la caja delimitadora alrededor de la persona, por lo que la detección es correcta. Sin embargo también se puede observar algún que otro falso positivo a la izquierda y a la derecha de la persona. A su derecha estaría detectando un coche por el tamaño y el color del *bounding box* y a su izquierda a otra persona.

Al igual que para el modelo *PointPillars*, con este modelo la inferencia también se ha realizado obteniendo los datos del LiDAR desde el dispositivo Jetson Nano mediante ROS. A diferencia de la inferencia anterior, donde la nube de puntos se ha introducido utilizando un *rosbag*, esta se realiza directamente de los datos que se están publicando en ese momento en el *topic*.

Como se puede observar en la Figura 7.15, en esta ocasión la detección de la persona también es correcta. A pesar de que, en ese instante, no se vea, también aparecen algunos falsos positivos esporádicamente y en función de hacia donde se han dirigido los rayos láser del LiDAR. Por otra parte, en ese mismo *topic* que genera las cajas delimitadoras que se visualizan por rviz, adicionalmente se publica la posición en metros del centro del *bounding box*, el ángulo de rotación que tiene y sus dimensiones.

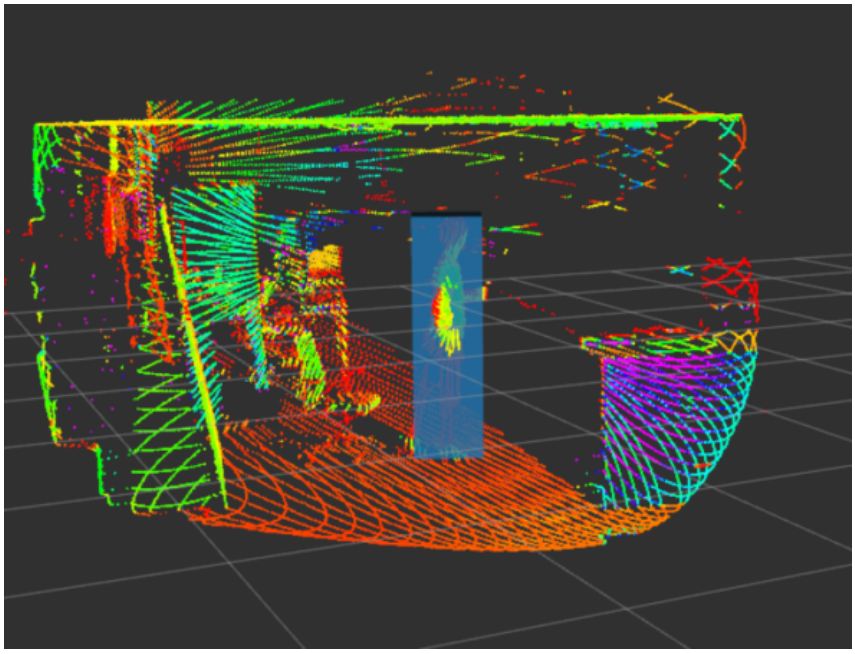


Figura 7.15: Inferencia con PointRCNN a través del dispositivo Jetson Nano.

8 | Resultados

En este capítulo se muestran los resultados experimentales obtenidos durante la realización del Trabajo Fin de Máster, donde se han utilizado varios modelos, algoritmos y configuraciones para la detección 3D, utilizando únicamente los datos del LiDAR. Dichos resultados han sido obtenidos tanto en el simulador CARLA como experimentalmente con el LiDAR Livox Mid-70.

8.1 | Análisis de algoritmos de detección 3D

En primer lugar, se han evaluado los algoritmos detallados en el apartado 4.2, los cuales han sido entrenados con el *dataset* KITTI. Como el Trabajo Fin de Máster se centra principalmente en la detección 3D, los modelos se han examinado sobre el *benchmark* de detección 3D de KITTI.

El *dataset* de KITTI contiene tres clases de objetos y utiliza un *threshold* de IoU para dar por correcta la detección. En el caso de la clase del coche, dicho *threshold* es de 0.7 o 70 %. En cambio, es de 0.5 para peatones o ciclistas, puesto que, al contener menos cantidad de puntos, es algo más complicada la detección. Además, hay que tener en cuenta que se definen tres dificultades diferentes. Los objetos que tienen un máximo de oclusión del 15 % son denominados fáciles, los que tienen un máximo de 30 % moderados y, en la evaluación difícil, un máximo de 50 %. Teniendo en cuenta estas indicaciones, se ha elaborado la Tabla 8.1 con los valores obtenidos de [24] y [31]. Además, se ha añadido el tiempo de entrenamiento para cada modelo, el cual ha sido determinado con 8 GPUs TITAN XP y PyTorch 1.5, que ha sido obtenido del repositorio de la herramienta OpenPCDet.

Tabla 8.1: Evaluación de los algoritmos de detección de objetos 3D en KITTI.

Método	**Data rep.**	**Train time**	**Coche**			**Peatón**			**Ciclista**			**Tiempo(s)**
			F	M	D	F	M	D	F	M	D	
SECOND	Vóxel	∼1.7h	84.7	75.9	68.7	48.9	38.8	34.9	71.3	52.1	45.8	0.04
PointPillars	Vóxel	∼1.2h	82.6	74.3	69.0	51.5	41.9	38.9	77.1	58.7	51.9	0.016
PointRCNN	Point	∼3h	86.9	75.6	70.7	48.0	39.4	36.0	74.9	58.8	52.5	0.1
PV-RCNN	V, P	∼5h	90.3	81.4	76.8	52.2	43.3	40.3	78.6	63.7	57.7	0.08

En la Tabla 8.1 se puede observar que las detecciones más precisas son llevadas a cabo por el algoritmo más actual, es decir PV-RCNN. Aún así, no existe una diferencia muy significativa entre los cuatro modelos. En cuanto a la clase de peatón, que a priori es la clase que nos interesa, puesto que el LiDAR Livox Mid-70 se utiliza dentro del laboratorio, una vez más PV-RCNN obtiene, una vez más, los mejores resultados, aunque le sigue muy de cerca *PointPillars*.

Prestando atención al tiempo de respuesta o de inferencia, como se había anticipado en el apartado 4.2, *PointPillars* es el que obtiene el mejor resultado con bastante notoriedad. Se puede deducir que los modelos que tienen que manejar las nubes de puntos en bruto tienen una menor velocidad de inferencia. Por último, se puede apreciar que el menor tiempo de entrenamiento también lo tiene *PointPillars*, con una diferencia extremadamente elevada frente a PV-RCNN. Este es un parámetro a tener en cuenta, ya que son los tiempos estimados utilizando 8 GPUs TITAN XP y la Woskstation únicamente dispone de una GTX 1080 Ti, por lo que los tiempos serán significativamente más elevados.

8.2 Análisis de los sistemas utilizados sobre el simulador CARLA

En este apartado se analizan los dos sistemas de detección utilizados con el simulador CARLA. De esta forma, es posible examinar su rendimiento utilizando los datos simulados de un LiDAR.

En primer lugar, tras estudiar las técnicas tradicionales más utilizada en el capítulo 4.1, se ha implementado una aplicación que ha sido detallada en el apartado7.2.2, la cual consta de un nodo ROS que se subscribe al *topic* del LiDAR del simulador y publica las detecciones, tal y como se observa en la Figura 8.1 .

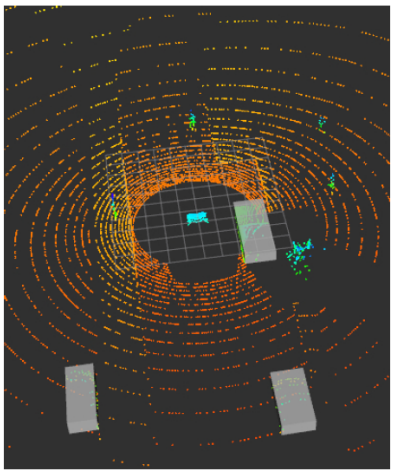


Figura 8.1: Sistema de detección clásico en CARLA.

Parece que la detección de este sistema es bastante precisa, puesto que no genera un número elevado de falsos positivos. Sin embargo, como se puede observar, únicamente detecta coches, ya que, si se intentara detectar peatones, se crearían muchos falsos positivos con farolas, semáforos, postes, etc. Por ese motivo, no es posible la diferenciación de clases de objetos, porque al diferenciarlos por tamaños, en función de la cantidad de puntos que incidan sobre el objeto se agruparían en la misma clase objetos como peatones, arbustos, farolas y ciclistas.

En cuanto al sistema de detección basado en el modelo *PointPillars* explicado en el apartado 7.2.3, este también consta de un nodo ROS que se subscribe al mismo *topic* donde se publican los datos del LiDAR de CARLA a través del *CARLA ROS bridge*. También se lanza un script de la API del *bridge* para generar peatones y coches en el simulador. El resultado se puede apreciar en la Figura 8.2.

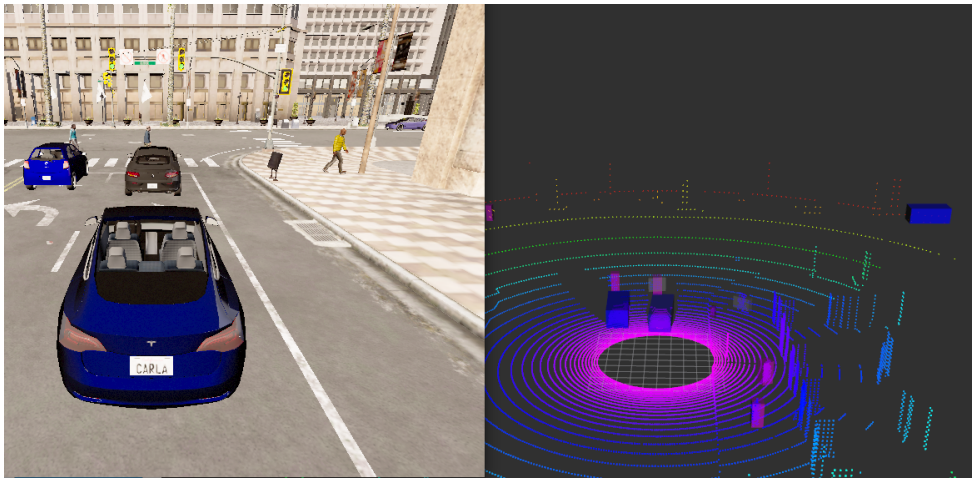


Figura 8.2: Sistema de detección basado en *PointPillars* sobre CARLA.

En este caso, se puede apreciar como detecta perfectamente los dos coches que tiene delante, e incluso el que está cruzando a la derecha de la imagen. Además, también detecta peatones, lo que son las cajas delimitadoras rosas, aunque con un poco menos de precisión (por ejemplo, el peatón que se encuentra en la parte derecha de la imagen, al lado del poste, no se detecta claramente). Incluso genera falsos positivos, como se puede observar al crear una caja para la papelera.

Como conclusión, se puede apreciar un funcionamiento mucho más preciso en el sistema basado en *PointPillars* sobre el simulador CARLA, donde además, si que existe la diferenciación entre clases de objetos. Sin embargo, también crea algún que otro falso positivo que se debe tener en cuenta.

8.3 | Análisis de los sistemas utilizados sobre el Livox Mid-70

Tras analizar los sistemas sobre el simulador CARLA, en este apartado se examina el funcionamiento experimental de los modelos con el Livox Mid-70 dentro del laboratorio.

Con el sistema de detección clásico no se espera un resultado muy exacto, ya que en el laboratorio no hay más que objetos como sillas, mesas, ordenadores, etc, además de personas. En la Figura 8.3 se puede observar el resultado obtenido.

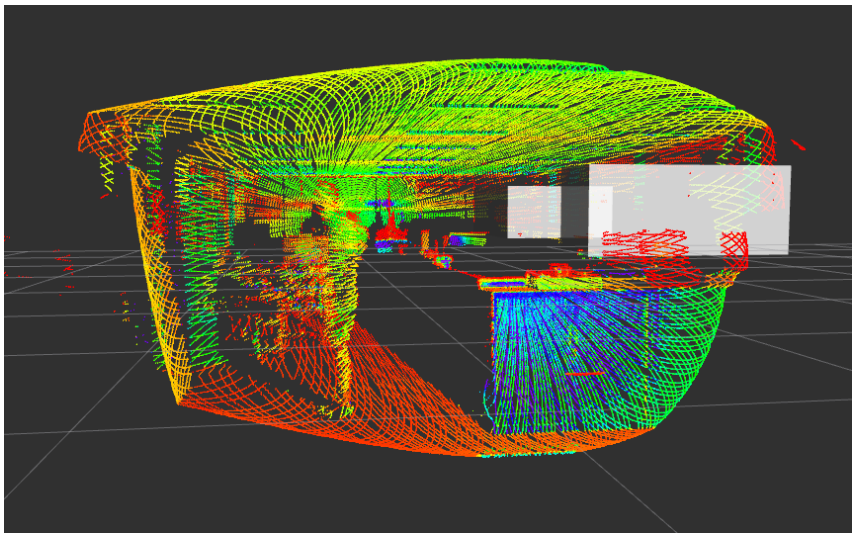


Figura 8.3: Resultados experimentales del sistema de detección clásico sobre Livox Mid-70.

Tal y como se esperaba, el sistema tradicional no detecta la persona que se encuentra en el centro de la nube de puntos cerca del sensor. Sin embargo, se puede apreciar que crea dos falsos positivos suponiendo que son coches, lo cual puede deberse a que el escenario se encuentra en un espacio interior, donde la cantidad de puntos es mayor.

A priori, se supone que, con el modelo *PointPillars*, la detección puede ser algo más precisa, ya que se ha logrado un mejor rendimiento sobre el simulador y, al diferenciar entre clases de objetos, podría detectar alguna persona. Como se puede observar en la Figura 8.4, en este caso tampoco se ha detectado a la persona y se han creado varios falsos positivos. A diferencia del sistema clásico, parece que está detectando a personas donde no debería, fundamentalmente debido al tamaño de las cajas delimitadoras.

En definitiva, la detección no ha sido exitosa con ninguno de los dos sistema. Esto se puede deber al patrón no repetitivo característico de la tecnología Livox, el cual se diferencia mucho al del simulador y al del *dataset* KITTI, que es con el que está entrenado el modelo. Por otro

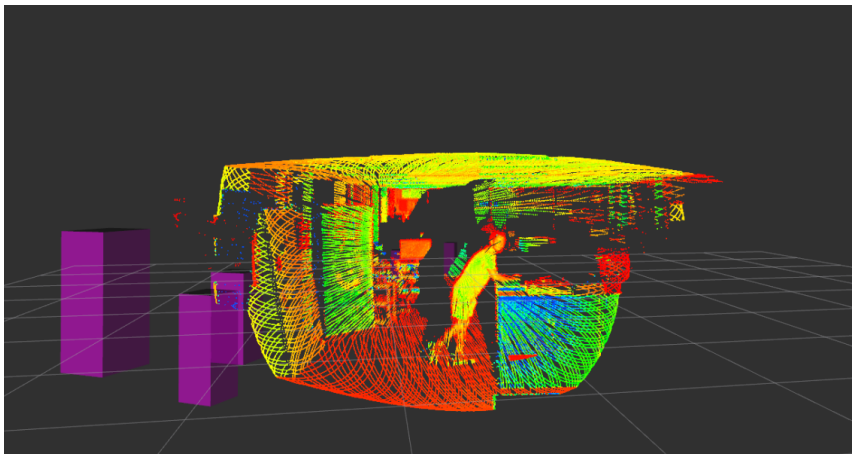


Figura 8.4: Resultados experimentales del modelo PointPillars utilizando el Livox Mid-70.

lado, al realizarlo en un espacio interior, la cantidad de puntos que inciden sobre los objetos es mucho mayor, por lo que este factor puede llegar a ser determinante.

8.4 | Entrenamiento del modelo PointPillars

Al observar que los resultados obtenidos utilizando el modelo *PointPillars* con el LiDAR Livox Mid-70 dentro del laboratorio no son nada precisos, se ha optado por reentrenar el modelo para mejorar la precisión de la detección, sobre todo la de las personas.

Para ello, es necesario crear el *dataset* con nubes de puntos del LiDAR. Para ello, tal y como se ha explicado en el apartado 7.5, se han tomado dos direcciones. Un *dataset* ha sido creado con el software de *data augmentation*, en el que se han utilizado elementos aislados obtenidos con el LiDAR para colocarlos en posiciones aleatorias, con una rotación aleatoria, y crear, así, miles de escenarios diferentes. Por el otro lado, con la herramienta Gazebo se han generado escenarios virtuales donde se va cambiando tanto la posición del sensor como la posición y la rotación de los objetos introducidos.

Tras crear escenarios con los dos métodos, se han comprobado los resultados utilizando la herramienta OpenPCDet y el script demo.py. Mediante este *script*, se puede probar un modelo preentrenado con datos de nubes de puntos personalizados (en este caso los escenarios creados) y visualizar los resultados predichos. En la Figura 8.5 se muestra el resultado utilizando un escenario de la aplicación de aumento de datos.

Se puede apreciar que, aunque no sean precisas al 100 % las detecciones correspondientes a las dos personas son correctas. Sin embargo, el algoritmo *PointPillars* crea muchos falsos positivos, e incluso crea una caja delimitadora (amarilla) correspondiente a un ciclista. Para obtener este resultado, ha sido necesario ajustar el archivo de configuración del modelo, puesto que, por

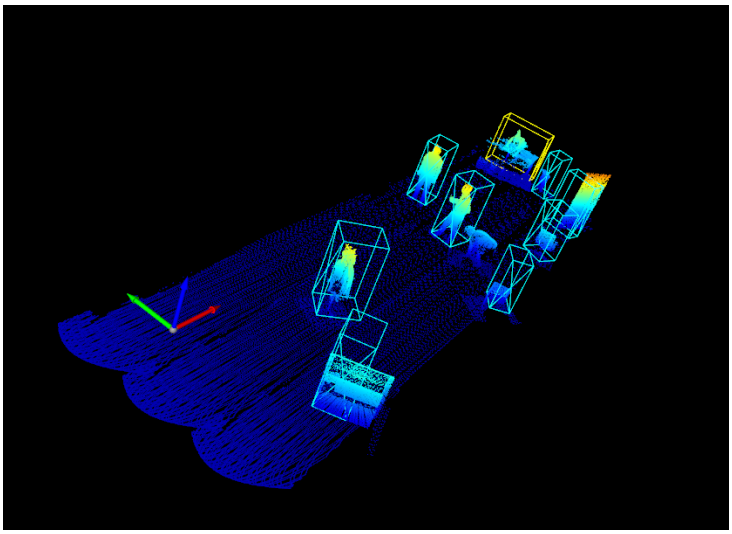


Figura 8.5: Resultados del escenario creado con el SW de aumento de datos y *PointPillars*.

defecto, se han creado más falsos positivos. De esta forma, aumentando el número máximo de puntos por vóxel, el tamaño del vóxel, etc. se ha conseguido mejorar la detección. El objetivo del entrenamiento ha sido eliminar dichos falsos positivos y mejorar la detección de las personas.

A la hora de entrenar *PointPillars* utilizando la herramienta OpenPCDet, puesto que el modelo se ha obtenido de la misma, también se han utilizado varios métodos. En primer lugar y sin realizar grandes modificaciones en el *script* train.py, los *datasets* se han adecuado a la estructura requerida (esta estructura es la seguida por el *dataset* KITTI). Para ello, tal y como se ha detallado anteriormente, ha sido necesario modificar las etiquetas cambiando el sistema de coordenadas del LiDAR al de la cámara de KITTI. Además, se han seguido utilizando tanto los directorios de las imágenes como los de calibración.

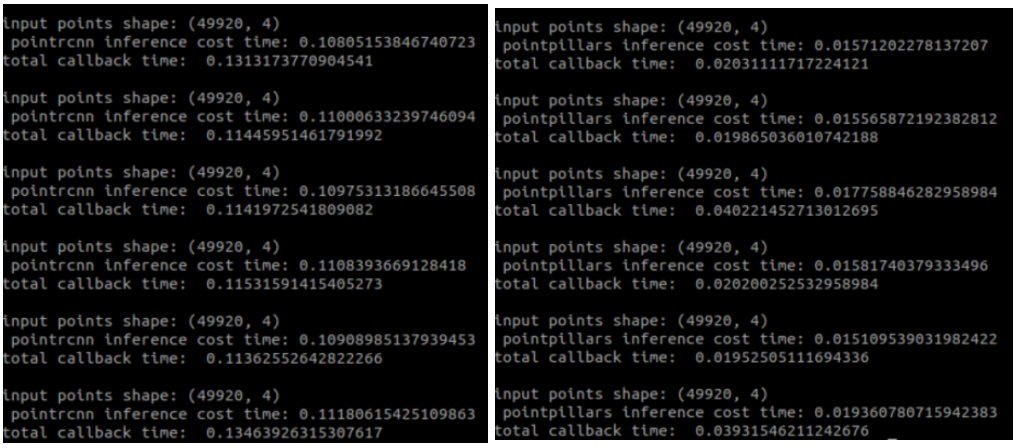
Por otro lado, al disponer únicamente de los datos proporcionados por el LiDAR, se ha configurado el *script* para que no sean necesarios las imágenes ni los archivos de calibración. Además, las etiquetas se han mantenido con las coordenadas del LiDAR. Se ha probado a seguir manteniendo las tres clases de objetos como también a configurarlo y únicamente disponer de una, en este caso la de peatón. Sin embargo, no se ha logrado mejorar el resultado de las detecciones.

8.5 | Comparativa entre *PointPillars* y PointRCNN

Al no obtener resultados óptimos para la detección 3D, se ha optado por probar otro modelo. En particular, se ha seleccionado el modelo PointRCNN, ya que, a la entrada necesita únicamente la nube de puntos en crudo. Como se ha avanzado en el capítulo 7.6, realizando la inferencia en la WS mientras se obtienen los datos del LiDAR sobre el dispositivo Jetson Nano el resultado es más exacto que en el caso de *PointPillars*. Con PointRCNN se consiguen detectar a las personas

y se crea un número muy inferior de falsos positivos.

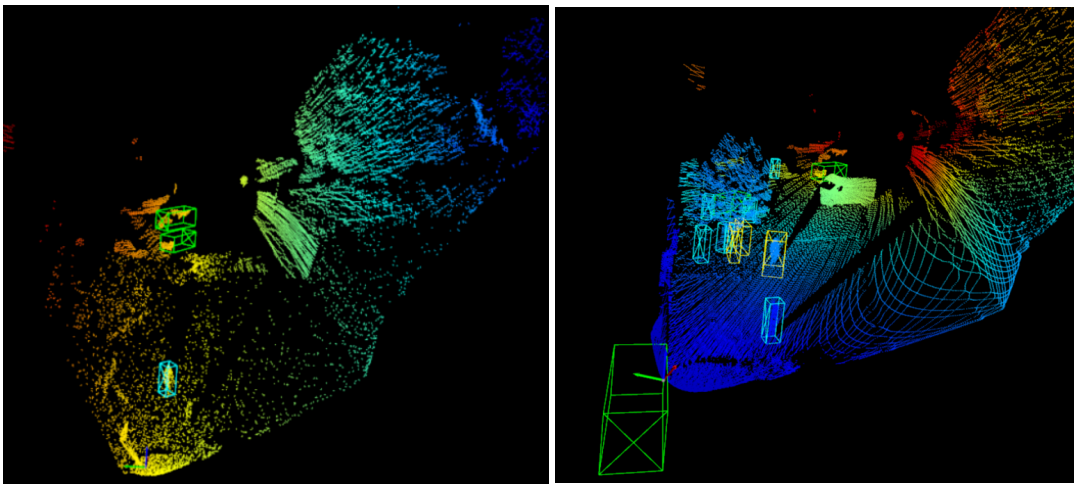
En cuanto al tiempo de inferencia, en la Figura 8.6 se muestran los tiempos de inferencia obtenidos con los dos modelos. Como se ha mencionado en el apartado 4.2, *PointPillars* tiene una velocidad mayor e incluso se asemeja a la frecuencia mencionada de 62 Hz. Con el modelo PointRCNN se obtiene una velocidad seis veces menor, pero una vez más, cerca de los 10 Hz tal y como se había anticipado.



(a) PointRCNN. **(b)** *PointPillars*.

Figura 8.6: Comparación de tiempos de inferencia entre PointRCNN y *PointPillars*.

También se ha comprobado el funcionamiento de los dos modelos en el exterior para analizar hasta qué punto es determinante realizar la detección en un espacio cerrado, donde la cantidad de puntos incidentes en los objetos es mucho mayor.



(a) PointRCNN en el exterior. **(b)** *PointPillars* en el exterior.

Figura 8.7: Comparación del funcionamiento de los modelos en el exterior.

Tal y como se puede observar en las Figura 8.7, el resultado de la detección 3D del modelo de PointRCNN es mucho más precisa, ya que, en este caso se han detectado perfectamente

la persona y los dos coches sin ningún falso positivo. Sin embargo, en el caso del modelo *PointPillars*, se detectan muchos falsos positivos y, además, la persona, que es la segunda silueta que se encuentra en el centro de la imagen, no es detectada con la clase de objetos de peatón, sino con la de ciclista. La caja delimitadora correspondiente a un peatón que se ha creado en la primera silueta que se encuentra en el centro de la imagen es, en este caso, la estación de carga para coches eléctricos.

Para terminar la comparativa, se han calculado las métricas de *Precisión*, *Recall* y *F1 Score*, que se detallan a continuación. Para poder calcular estas métricas, en primer lugar es necesario definir los parámetros TP, FP y FN, donde TP representa *True Positive*, FP *False Positive* y FN *False Negative*.

Para definir estos valores, es necesario determinar una relación entre las detecciones y el *ground-truth*, lo cual se realiza con el IoU 3D, puesto que mide el volumen de intersección entre dos cajas delimitadoras, tal y como se muestra en la Figura 8.8 y en (8.1).

$$IoU\ 3D = \frac{Volumen de la intersección}{Volumen de la unión} \tag{8.1}$$

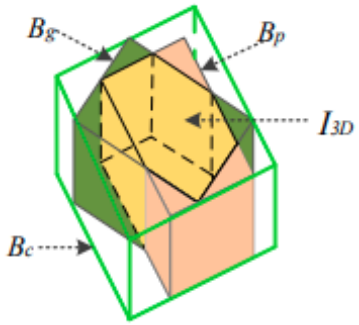


Figura 8.8: Representación de IoU 3D.

Para calcular el IoU ha sido necesario obtener las esquinas de las cajas delimitadoras, tanto del *groundtruth* como de las detecciones predichas y calcular la intersección entre ellas. Las detecciones se han ordenado de mayor *score*, es decir mayor confianza de detección, a menor y, en caso de que intersecte con alguna *bounding box* del *groundtruth*, ese es el que le corresponde. Por otro lado, si se intersecta con más de una caja, se asocia con la que ha conseguido un porcentaje de intersección mayor.

Tras calcular el IoU, se obtienen TP, FP y FN de la siguiente forma:

- ***True Positive*** **(TP)**: Si el cálculo de el IoU es mayor que un *threshold* estipulado, en este caso 0.5.

- ***False Positive* (FP)**: Si el IoU es menor que 0.5, si se obtienen multiples cajas delimitadoras para un mismo objeto o si no coinciden las clases de los objetos.
- ***False Negative* (FN)**: Cuando se queda algún objeto del *groundtruth* sin detectar.

Una vez que se han obtenido los valores de TP, FP y FN, se dispone a calcular las métricas mencionadas anteriormente:

- **Precisión:** Mide cómo de precisas son las predicciones de los modelos mediante un porcentaje de predicciones correctas (8.2).

$$Precision = \frac{TP}{TP + FP} \tag{8.2}$$

  Este parámetro también se conoce como valor predicho positivo, se da como la relación de verdadero positivo (TP) y el número total de positivos pronosticados, esto es, incluyendo los falsos positivos (FP).

- ***Recall* o Sensibilidad:** La cual también se conoce como tasa de verdaderos positivos (TRP). Mide la capacidad del modelo de detectar casos positivos. Se define como la relación de positivos verdaderos y el total de positivos del *groundtruth*, es decir, la suma de los TP y los falsos negativos (FN): (8.3).

$$Recall = \frac{TP}{TP + FN} \tag{8.3}$$

- ***F1 Score*:** Es el promedio ponderado de Precisión y *Recall*. Por lo tanto, esta puntuación tiene en cuenta tanto los falsos positivos como los falsos negativos (8.4).

$$F1\ Score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{8.4}$$

En la Tabla 8.2 se muestran las métricas calculadas utilizando los diez mismos escenarios generados con el software de *data augmentation* en los modelos *PointPillars* y PointRCNN.

Tabla 8.2: Métricas de los modelos PointPillars y PointRCNN.

Método	TP	FP	FN	*Precision*	*Recall*	*F1 Score*
PoitnPillars	7	80	0	0.08	1	0.149
PointRCNN	10	20	0	$0.\widehat{33}$	1	0.5

Como se ha mencionado anteriormente, se ha utiliza un *trehshold* de 0.5. Se puede observar que la cantidad de falsos positivos es muy elevada. Sin embargo, la cantidad de falsos positivos del modelo *PointPillars* es cuatro veces mayor, claro indicativo de que la detección es significativamente peor. Además, la detección de PointRCNN también es mucho más precisa teniendo un valor mínimo, muy cercano a cero, en el caso de *PointPillars*. El *Recall* es exactamente el mismo para los dos modelo, puesto que no ha quedado ninguna caja delimitadora del *groundtruth* sin asociar con alguna detección.

9 | Conclusiones

En primer lugar, se ha comprobado que los sistemas de percepción tradicionales cuentan con ciertas limitaciones. Estos sistemas no se centran en aprender las distintas características que tienen los objetos, si no que las diferencian en función del tamaño. De esta manera, es realmente complicado obtener una detección de varias clases de manera satisfactoria. Por ello, los estudios más actuales del estado del arte se enfocan en los sistemas de detección de objetos 3D basados en algoritmos de *Deep Learning*.

En cuanto a los algoritmos de detección de objetos 3D basados en *Deep Learning*, existe una gran variedad de ellos, y cada uno utiliza algún método diferente, por lo que se debe determinar cual puede ser el óptimo para cada tipo de LiDAR o el que mejor se adapta a todos los LiDAR. Además, el hecho de utilizar únicamente datos del sensor láser reduce significativamente la información que se puede obtener del entorno. En cuanto a cuestiones relativas a la profundidad, queda patente su utilidad, aunque las cámaras, tienen un papel relevante en cuanto a obtener las texturas del entorno o ayudar a clasificar objetos.

Se ha demostrado el modelo PointRCNN realiza una detección más exacta que el modelo *Point-Pillars* para el LiDAR Livox Mid-70. Con el modelo PointRCNN se ha conseguido realizar la detección de las personas, incluso dentro del laboratorio, aunque generando algún que otro falso positivo. Además, es importante remarcar que la posibilidad de que el mal funcionamiento del algoritmo *PointPillars* se deba a estar en un espacio cerrado se ha descartado, ya que, en el exterior, tampoco se ha conseguido un resultado bueno, aunque si algo mejor. Es posible que el método de escaneo del Mid-70 tenga una influencia importante, ya que es muy utiliza al utilizado para el conjunto de datos KITTI o en el simulador CARLA.

En relación a los tiempos de inferencia, *PointPillars* es muy superior. Esta es una característica a tener muy en cuenta, ya que todos estos sistemas están orientados a los sistemas autónomos con la limitación de computo que ello conlleva.

En relación a otra serie de aportaciones del trabajo, se ha creado una aplicación software de aumento de datos para poder crear miles de escenarios para el LiDAR Livox Mid-70, lo cual puede ser muy útil para entrenar distintos modelos. Además, con la herramienta Gazebo también se ha creado una aplicación que genera distintos escenarios virtuales para poder crear *datasets* con datos sintéticos.

Los datos sintéticos son una de las tecnologías revolucionarias que jugarán un papel crucial en la construcción de modelos de Inteligencia Artificial, puesto que son muy útiles para enriquecer los datos reales y, así, construir modelos de excelente valor. En este Trabajo Fin de Máster, se ha demostrado que se pueden utilizar nubes de puntos sintéticos al poder realizar las detecciones con distintos modelos.

Por último, cabe remarcar que se se ha intentado optimizar la detección 3D de las personas entrenando los modelos de distintas formas, aunque no ha sido posible obtener un resultado satisfactorio. Es necesario tener muy claros los principios de funcionamiento de los modelos utilizados y se deben tener buenos conocimientos sobre Inteligencia Artificial para poder entrenar un modelo de forma adecuada.

10 | Líneas de trabajo futuro

Durante la realización de este trabajo se ha constatado el gran potencial de la tecnología aplicada. Por un lado, aunque este Trabajo Fin de Máster se ha centrado en la detección de personas utilizando el láser Livox Mid-70, no se ha hecho uso de todo su potencial. Se cree interesante incluir a la visualización de las cajas delimitadoras la distancia a la que se encuentra el objeto detectado, en vez de mostrarlo únicamente a través de un *topic* como se hace en la actualidad.

Por otro lado, combinando en Livox con una cámara, se podrían añadir funcionalidades a la aplicación como, por ejemplo, la identificación de personas.

También se ve conveniente ahondar más en el entrenamiento de las redes y determinar como optimizar la detección en 3D. El punto de partida debería ser entrenar los modelos utilizando datos del Mid-70. Estos datos podrían ser tanto sintéticos como reales. Además, se considera posible bloquear algunas partes de la red neuronal (para que no olvide lo aprendido) y únicamente entrenar la sección que se crea interesante, así como la parte de las salidas donde se predicen las detecciones. De esta forma, no se modifican por completo todos los pesos de la red.

También se considera interesante analizar diferentes plataformas hardware embebidas. De esta manera, se podrían evaluar y comparar las prestaciones de cada una y como afectan a la solución de detección. A lo largo del Trabajo Fin de Máster se ha empleado el dispositivo Jetson Nano, aunque se podría probar con una Jetson Xavier o una IMX8. Estos dispositivos son más potentes que la Jetson Nano y su uso está todavía más enfocado a los sistemas autónomos.

Bibliografía

[1] D. P. Watson and D. H. Scheidt, "Autonomous systems," *Technical Digest, Johns Hopkins Applied Physics Laboratory*, vol. 26, 2005.

[2] L. Troyer, "The criticality of social and behavioral science in the development and execution of autonomous systems," *Human-Machine Shared Contexts*, vol. 1, pp. 161–167, 2020.

[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[4] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: Insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, 12 2021.

[5] SAE, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," *SAE Tech. Paper J3016_201806*, 2016.

[6] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3d object detection methods for autonomous driving applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 3782–3795, 2019.

[7] J. V. Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 384–406, 2018.

[8] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. D. Souza, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, 2021.

[9] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," 2022.

[10] Velodyne, "Hdl-32e high resolution real-time 3d lidar sensor unprecedented field of view and point density." , datasheet, Disponible: https://velodynelidar.com/products/hdl-32e/.

[11] Z. Cui and Z. Zhang, "Blpnet: An end-to-end model towards voxelization free 3d object detection," *2020 Joint 9th International Conference on Informatics, Electronics & Vision (ICIEV) and 2020 4th International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, pp. 1–8, 2020.

[12] K. Zięba-Kulawik, K. Skoczylas, P. Wężyk, J. Teller, A. Mustafa, and H. Omrani, "Monitoring of urban forests using 3d spatial indices based on lidar point clouds and voxel approach," *Urban Forestry and Urban Greening*, vol. 65, 2021.

[13] M. Sanatkar, "Lidar 3d object detection methods." https://towardsdatascience.com/lidar-3d-object-detection-methods-f34cf3227aea, 2020. Visitado: 2022.

[14] A. Raj, "3d ransac algorithm for lidar pcd segmentation." Disponible: https://medium.com/@ajithraj_gangadharan/3d-ransac-algorithm-for-lidar-pcd-segmentation-315d2a51351, 2020. Visitado: 2022.

[15] J. Foley, M. Fischler, and R. Bolles, "Graphics and image processing random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 1–15, 1981.

[16] I.-S. Weon, S.-G. Lee, and J.-K. Ryu, "Object recognition based interpolation with 3d lidar and vision for autonomous driving of an intelligent vehicle," *IEEE Access*, vol. 8, pp. 65599–65608, 2020.

[17] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[19] N. Parlante, "Binary trees." http://cslibrary.stanford.edu/110/BinaryTrees.html. Visitado: 2022.

[20] M. Ala'raj, M. Majdalawieh, and M. F. Abbod, "Improving binary classification using filtering based on k-nn proximity graphs," *Journal of Big Data*, vol. 7, 12 2020.

[21] F. Kong, W. Xu, Y. Cai, and F. Zhang, "Avoiding dynamic small obstacles with onboard sensing and computation on aerial robots," *IEEE Robotics and Automation Letters*, vol. 6, pp. 7869–7876, 10 2021.

[22] I. C. S. Society, D. da xue, C. A. of Automation. Technical Committee on Control, D. of Cyber Physical Systems, I. S. S. I. E. Chapter, C. da xue, I. of Electrical, and E. Engineers., *Proceedings of the 30th Chinese Control and Decision Conference (2018 CCDC) : 09-11 June 2018, Shenyang, China*.

[23] D. Fernandes, A. Silva, R. Névoa, C. Simões, D. Gonzalez, M. Guevara, P. Novais, J. Monteiro, and P. Melo-Pinto, "Point-cloud based 3d object detection and classification methods for self-driving applications: A survey and taxonomy," *Information Fusion*, vol. 68, pp. 161–191, 2021.

[24] Y. Wu, Y. Wang, S. Zhang, and H. Ogai, "Deep 3d object detection networks using lidar data: A review," *IEEE Sensors Journal*, vol. 21, pp. 1152–1171, 2021.

[25] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors (Switzerland)*, vol. 18, 2018.

[26] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4490–4499, 2018.

[27] "Sparseconvnet project." Disponible: `https://github.com/facebookresearch/SparseConvNet`. Visitado: 2022.

[28] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," vol. 2019-June, pp. 12689–12697, IEEE Computer Society, 2019.

[29] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," vol. 2017-January, pp. 77–85, Institute of Electrical and Electronics Engineers Inc., 11 2017.

[30] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 770–779, 2019.

[31] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, pp. 4338–4364, 2021.

[32] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 10526–10535, 2020.

[33] L. Tech, "Livox mid-70 user manual." Disponible: `https://terra-1-g.djicdn.com/65c028cd298f4669a7f0e40e50ba1131/Download/Mid-70/new/Livox%20Mid-70%20User%20Manual_EN_v1.2.pdf`, 2020.

[34] Nvidia, "Data sheet nvidia jetson nano system-on-module maxwell gpu + arm cortex-a57 + 4gb lpddr4 + 16gb emmc," 2014.

[35] F. Tully, "ROS wiki." Disponible: `http://wiki.ros.org/`, 2020. Visitado: 2022.

[36] Q. Morgan, G. Brian, C. Ken, F. Josh, F. Tully, L. Jeremy, B. Eric, W. Rob, and N. Andrew, "ROS: an open-source robot operating system," Computer Science Department, Stanford University, Standford University, Standford, CA, USA, Mayo 2009.

[37] "ROS 2 documentation: Foxy." Open Robotics. Disponible: `https://docs.ros.org/en/foxy/index.html`. Visitado: 2022.

[38] M. Yuya, K. Shinpei, and A. Takuya, "Exploring the performance of ros2," in *International Conference on Embedded Software (EMSOFT), 13th International Conference, Pittsburgh, PA, USA*, Octubre 2016.

[39] CARLA, "Carla simulator documentation." Disponible: `https://carla.readthedocs.io/en/latest/`. Visitado: 2022.

[40] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[41] "Docker." `https://www.docker.com/`. Visitado: 2022.

[42] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[43] O. Robotics, "Gazebo." `https://gazebosim.org/home`. Visitado: 2022.

[44] "Sdf model object." `https://classic.gazebosim.org/tutorials?tut=build_model`. Visitado: 2022.

[45] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[46] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[47] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[48] O. D. Team, "Openpcdet: An open-source toolbox for 3d object detection from point clouds." `https://github.com/open-mmlab/OpenPCDet`, 2020.

[49] F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab, and M. Abbas, "The influence of agile methodology (scrum) on software project management," in *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 145–149, 2019.

[50] L. A. Garcia, E. OliveiraJr, and M. Morandini, "Tailoring the scrum framework for software development: Literature mapping and feature-based support," *Information and Software Technology*, vol. 146, p. 106814, 2022.

[51] R. Kurnia, R. Ferdiana, and S. Wibirama, "Software metrics classification for agile scrum process: A literature review; software metrics classification for agile scrum process: A literature review," 2018.

[52] L. Tech, "Livox sdk." Disponible: `https://github.com/Livox-SDK/Livox-SDK`, 2019. Visitado: 2022.

[53] L. Tech, "Livox ros driver." Disponible: `https://github.com/Livox-SDK/livox_ros_driver`, 2019. Visitado: 2022.

[54] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), IEEE, May 9-13 2011.

[55] O. Robotics, "Estructura del mensaje pointcloud2." Disponible: `http://docs.ros.org/en/api/sensor_msgs/html/msg/PointCloud2.html`. Visitado: 2022.

[56] "ROS bridge documentation." Disponible: `https://carla.readthedocs.io/projects/ros-bridge/en/latest/`. Visitado: 2022.

[57] Javier-Dlap, "3d lidar based clustering." Disponible: https://github.com/Javier-DlaP/3D_lidar_based_clustering.git, 2021. Visitado: 2022.

[58] Javier-Dlap, "t4ac openpcdet ros." Disponible: https://github.com/RobeSafe-UAH/t4ac_openpcdet_ros.git, 2020. Visitado: 2022.

[59] ADLINK, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2020.

[60] O. Robotics, "Migrating launch files from ros 1 to ros 2." Disponible: https://docs.ros.org/en/crystal/Tutorials/Launch-files-migration-guide.html#migrating-tags-from-ros-1-to-ros-2. Visitado: 2022.

[61] S. Cheng, Z. Leng, E. D. Cubuk, B. Zoph, C. Bai, J. Ngiam, Y. Song, B. Caine, V. Vasudevan, C. Li, Q. V. Le, J. Shlens, and D. Anguelov, "Improving 3d object detection through progressive population based augmentation," *European Conference on Computer Vision*, pp. 279–294, 2020.