eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo Fin de Máster

Máster Universitario en Ingeniería Computacional y Sistemas Inteligentes

Validación de la detección de objetos 3D basada en nubes de puntos para la conducción autónoma mediante motores de simulación

Aitor Iglesias Hernández

Dirección

Ignacio Arganda	UPV/EHU
Mikel García	VICOMTECH
Nerea Aranjuelo	VICOMTECH

13 de septiembre de 2023

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas e instituciones que han hecho posible la realización de este Trabajo de Fin de Máster.

En primer lugar, quiero agradecer al Centro de Investigación VICOMTECH por haberme brindado la oportunidad de realizar este proyecto en sus instalaciones, así como los recursos prestados. Agradezco especialmente a todos los miembros del equipo de VICOMTECH por su colaboración y apoyo.

Asimismo, quiero agradecer a mis directores de este Trabajo de Fin de Máster en VICOMTECH, Mikel y Nerea, por su excelente trabajo, dedicación y por haberme guiado en todo momento. Gracias a su experiencia y conocimientos he podido desarrollar este proyecto de la mejor manera posible.

Por otro lado, quiero agradecer al director dentro de la Universidad del País Vasco, Nacho, por su dirección y orientación.

Por último, agradezco el apoyo incondicional de mi familia y amigos, quienes me han brindado su paciencia, comprensión y motivación para seguir adelante en todo momento.

Resumen

La detección de objetos en nubes de puntos es un tema emergente en la conducción autónoma que ha cobrado importancia gracias a los avances en tecnologías de sensores LiDAR *(Laser Imaging Detection and Ranging)* y técnicas de procesamiento de nubes de puntos.

En este proyecto se ha llevado a cabo una investigación de los modelos de aprendizaje profundo de detección de objetos en nubes de puntos, analizando no solo la fiabilidad de estos sino también a la frecuencia a la que funcionan. Para ello se ha hecho uso del conjunto de datos nuScenes, que permite que los modelos puedan realizar inferencia con una nube de puntos o con varias, acumulando varias nubes en una sola, permitiendo así realizar inferencia en base a una entrada más rica que aporta más información a la red. Se han entrenado los modelos variando la cantidad de nubes de puntos de entrada para posteriormente medir y analizar la diferencia de precisión de los modelos y también la diferencia de frecuencia.

Si bien se ha podido hacer uso del conjunto de datos nuScenes, la recopilación de estos datos requiere una gran cantidad de tiempo y recursos. Además, muchas situaciones son difíciles de adquirir, ya sea por la peligrosidad que suponen o por ser situaciones inusuales. Es ahí donde entran en juego los simuladores de automoción que permiten generar datos sintéticos. Estos datos se utilizan muchas veces para entrenar modelos o validar funciones ADAS (*Advanced Driver Assistance Systems*), sin embargo, es innegable que existe un *domain gap* entre los datos reales y sintéticos.

En este proyecto se realiza un estudio de esta brecha de dominio o *domain gap*. Para ello se hace uso de CARLA, un simulador realista de automoción, y se implementa un código capaz de generar datos sintéticos, a partir de los cuales se crea un conjunto de datos. Este conjunto de datos se utiliza junto al conjunto de datos nuScenes se evalúan algunos modelos del estado del arte en detección de objetos en nubes de puntos para realizar un análisis del comportamiento de los modelos ante ambos tipos de datos.

Palabras clave: *aprendizaje automático, visión por computador, detección de objetos 3D, conducción autónoma*

Índice de contenidos

Índice de figuras

Índice de tablas

CAPÍTULO 1

Motivación y objetivos

1.1. Motivación

En el contexto actual de la industria automotriz, la seguridad vial y la conducción autónoma han emergido como áreas de investigación y desarrollo críticas. La detección precisa y confiable de objetos en el entorno cercano de un vehículo es esencial para garantizar la seguridad de los pasajeros, peatones y otros usuarios de la vía. A medida que la tecnología avanza, las nubes de puntos han demostrado ser una fuente rica y prometedora de datos tridimensionales que capturan con precisión el mundo que nos rodea. Estas representaciones digitales permiten una comprensión más completa y detallada del entorno, lo que a su vez brinda la oportunidad de mejorar significativamente los sistemas de detección de objetos en comparación con los métodos tradicionales basados en imágenes.

En este proyecto se exploran los modelos de detección de objetos en nubes de puntos aplicados al ámbito automotriz, con el objetivo de superar los desafíos actuales y contribuir al desarrollo de vehículos más seguros e inteligentes. Para lograr esto, se lleva a cabo un análisis exhaustivo de varios modelos de detección, evaluando no solo su precisión en la identificación de objetos, sino también su tiempo de inferencia, un factor crítico en aplicaciones en tiempo real como la conducción autónoma. Además, se realiza una variación en el número de nubes de puntos de entrada utilizadas en el proceso, con el fin de determinar si es viable y beneficioso sacrificar una cierta cantidad de precisión en aras de lograr un incremento significativo en la velocidad de detección. Este enfoque permite desarrollar soluciones más efectivas y eficientes que puedan mejorar significativamente la seguridad y la capacidad de toma de decisiones de los vehículos automatizados.

La adquisición de datos desempeña un papel fundamental en el desarrollo y validación de algoritmos de detección de objetos en el campo de la automoción. Sin embargo, obtener conjuntos de datos extensos y variados que representen de manera precisa y exhaustiva los escenarios de conducción del mundo real puede resultar costoso y limitante en términos de tiempo y recursos. Es en este contexto que la generación de datos sintéticos se alza como una herramienta práctica para superar estas limitaciones. El uso de simuladores permite crear entornos virtuales que replican de manera fiel las condiciones de manejo en diferentes situaciones, ofreciendo la capacidad de generar volúmenes significativos de datos de manera controlada y diversa. En este proyecto se estudia la viabilidad, efectividad

y aplicabilidad de la generación de datos sintéticos a través de simuladores en el ámbito de la detección de objetos en automoción. Al hacerlo, busca no solo abordar los desafíos prácticos y técnicos asociados con la generación de datos realistas, sino también demostrar cómo esta metodología puede fortalecer la validez y robustez de los algoritmos de detección, acelerando así el avance de la seguridad y la tecnología en la industria automotriz.

1.2. Estructura de la memoria

La memoria de esta tesis de máster sigue una estructura dividida en varios apartados. En el primero se realiza una introducción sobre los conceptos básicos de la conducción autónoma, como los diferentes niveles de autonomía de un vehículo y los sensores utilizados en los vehículos. También se introducen las nubes de puntos y los sistemas de coordenadas y transformaciones utilizadas sobre estas a lo largo del proyecto. A continuación, se presenta el estado del arte de la generación de datos sintéticos centrándose principalmente en los datos generados por medio de simuladores, se mencionan los diferentes conjuntos de datos de automoción, especialmente en aquellos que cuentan con nubes de puntos, también se recopilan algunos de los modelos de detección de objetos en nubes de puntos. En el siguiente capítulo, se detalla la metodología utilizada a lo largo del proyecto, indicando las herramientas utilizadas, el proceso que se ha seguido para la generación del conjunto de datos sintético utilizando el simulador CARLA, especificaciones del conjunto de datos generado, las métricas de evaluación utilizadas y la configuración de los modelos de detección de objetos estudiados. Tras esto, se analizan los resultados obtenidos por los modelos y la diferencia entre los datos reales y sintéticos y como afectan a los modelos. Finalmente, se presentan las conclusiones y posibles líneas de trabajo futuro para mejorar la detección de objetos en nubes de puntos en la conducción autónoma, así como la generación de datos sintéticos.

1.3. Objetivos

El proyecto fin de máster queda definido por los siguientes objetivos de alto nivel:

1. Análisis del estado del arte en modelos de detección de objetos para nubes de puntos.
2. Creación de un *pipeline* de entrenamiento y testeo para modelos de detección basados en nubes de puntos.
3. Entrenamiento de modelos usando conjuntos de datos públicos.
4. *Pipeline* de generación de datos anotados en entornos de simulación.
5. Testeo de modelos con los datos virtuales.
6. Evaluación de los resultados obtenidos para conjuntos de datos y datos generados en simulación.

CAPÍTULO 2

Introducción

En el ámbito de las tecnologías de asistencia a la conducción (ADAS, *Advanced Driver Assistance Systems*) y la conducción autónoma (AD, *Autonomous Driving*), la percepción del entorno exterior del vehículo se convierte en el componente primordial para el desarrollo de funciones ADAS que incluyen frenado de emergencia, control de velocidad, ayuda al estacionamiento y mantenimiento de carril.

La percepción se basa en la utilización de sensores de medición del entorno, para determinar su posición, velocidad, tamaño y clase de los objetos cercanos al vehículo. Esta tarea suele ser llevada a cabo usando información de dos tipos de sensores, cámaras o LiDAR (*Light Detection and Ranging*) [1]. Las cámaras proporcionan mayor información semántica de una escena a cambio de perder información de profundidad en el proceso proyectivo. Los sensores basados en láseres como el LiDAR proporcionan una precisión de muestreo físico del entorno mucho mayor gracias a producir información tridimensional en forma de nube de puntos de la escena local. En el mundo de la automoción, esta diferencia de precisión puede ser vital para la comprensión de lo que está sucediendo alrededor del vehículo y para la posterior toma de decisiones.

Desde la aparición de las técnicas modernas de aprendizaje profundo en 2012, junto con los grandes avances tecnológicos en hardware de cómputo de altas capacidades (GPUs), es posible disponer de modelos de detección y clasificación de objetos en nubes de puntos o en imágenes. A partir de estas detecciones se puede obtener información de los actores (vehículos, peatones, etc.) y obstáculos dentro de las imágenes y nubes de puntos. Uno de los principales problemas en el uso de técnicas basadas en aprendizaje profundo es la necesidad de una gran cantidad de datos anotados. Realizar grabaciones y anotarlas manualmente es una tarea que requiere una gran cantidad de tiempo y coste financiero. Estas grabaciones necesitan una calibración previa de los sensores del vehículo y una sincronización en la captura de datos, tareas que, aun a día de hoy, suponen un gran reto en la industria de la automoción [2, 3, 4, 5, 6]. Es por ello que la mayoría de trabajos científicos en el ámbito de la automoción usan conjuntos de datos anotados y verificados por anotadores humanos. Gracias a la gran cantidad de conjuntos de datos de automoción anotados que han sido publicados en los últimos años [7, 8, 9, 10, 11, 12, 13, 14, 15], entrenar y generar modelos de detección robustos para vehículos autónomos es mucho más accesible y está generando un gran interés científico e industrial.

Estos conjuntos de datos proporcionan varios *terabytes* de datos anotados obtenidos en situaciones de conducción reales en gran variedad de entornos y situaciones. Sin embargo, es imposible generar datos reales en situaciones peligrosas como atropellos o accidentes entre vehículos. Para ello, una posibilidad es el uso de entornos de simulación realistas basados en motores gráficos como Unreal Engine[1] o Unity[2]. Existen varios entornos de simulación de código abierto en el mundo de la automoción como, por ejemplo, CARLA [16] o LGSVL [17]. En estos entornos podemos simular y personalizar tanto la configuración o *setup* de sensores de un vehículo instrumentado, así como generar escenarios de riesgo y obtener datos etiquetados (*ground truth*) de la simulación. Uno de los mayores inconvenientes de usar datos simulados es que, pese a que en los últimos años se ha conseguido dar gran realismo a los entornos de simulación, todavía existe una brecha entre estos y los datos obtenidos de grabaciones reales [18, 19, 20, 21].

En la actualidad, existe una gran variedad de modelos de aprendizaje profundo que se utilizan para la detección de objetos en nubes de puntos [22, 23, 24, 25, 26, 27, 28], lo que ha permitido un gran avance en la tecnología de conducción autónoma. Para llevar a cabo esta investigación, se ha utilizado el software proporcionado por la compañía OpenMMLab, que ofrece una amplia gama de herramientas y modelos para la detección de objetos en nubes de puntos. Entre ellos, destacan MMDetection3D [29], que permite entrenar, evaluar y testear modelos de manera sencilla, y MMDeploy [30], que permite optimizar algunos de estos modelos. El uso de estas herramientas permite analizar modelos que funcionan en diferentes conjuntos de datos, consiguiendo una comparación detallada de su rendimiento y precisión en diferentes situaciones de conducción autónoma.

2.1. Conducción autónoma

La conducción autónoma es una de las áreas de investigación y desarrollo más relevantes en la industria del transporte y de la tecnología. En los últimos años, ha habido avances significativos en la tecnología y la regulación para permitir la conducción autónoma, la cual puede ser clasificada en diferentes niveles de automatización, que van desde la autonomía parcial hasta la autonomía completa (Figura 2.1).

Con respecto a la tecnología utilizada, las empresas líderes en el espacio de conducción autónoma, como Waymo, Tesla, Cruise, Mobileye y Mercedes, están utilizando una combinación de sensores, cámaras y algoritmos de aprendizaje automático para permitir que los vehículos autónomos reconozcan su entorno y tomen decisiones de conducción en tiempo real. Estos sistemas han mejorado significativamente en los últimos años y han demostrado una capacidad cada vez mayor para evitar obstáculos, responder a las señales de tráfico y adaptarse a diferentes condiciones climáticas y de iluminación.

En cuanto a la regulación, varios países y estados han comenzado a implementar marcos regulatorios para permitir la conducción autónoma en las carreteras públicas. En los Estados Unidos, por ejemplo, la Sociedad de Ingenieros de Automoción (SAE) ha desarrollado un marco de seis niveles para describir los diferentes grados de automatización de los vehículos [31], desde los vehículos sin asistencia al conductor hasta los vehículos totalmente autónomos (Figura 2.1). Los niveles de autonomía son los siguientes:

[1] https://www.unrealengine.com/en-US
[2] https://unity.com/es

- **Nivel 0: sin automatización.** El conductor es responsable de todas las funciones de conducción y control del vehículo.
- **Nivel 1: asistencia al conductor.** El vehículo tiene sistemas de asistencia al conductor que pueden controlar la dirección o la velocidad, pero no ambos simultáneamente.
- **Nivel 2: automatización parcial.** El vehículo tiene sistemas de asistencia al conductor que pueden controlar la dirección y la velocidad simultáneamente. Sin embargo, el conductor sigue siendo responsable de supervisar y tomar el control del vehículo en todo momento.
- **Nivel 3: automatización condicional.** El vehículo puede controlar todas las funciones de conducción bajo ciertas condiciones y situaciones, pero el conductor debe estar preparado para intervenir en caso de emergencia.
- **Nivel 4: automatización alta.** El vehículo puede controlar todas las funciones de conducción en la mayoría de las situaciones, pero aún puede requerir la intervención humana en situaciones extremas o poco comunes.
- **Nivel 5: automatización completa.** El vehículo puede controlar todas las funciones de conducción en todas las situaciones posibles, y no se requiere intervención humana en ningún momento.

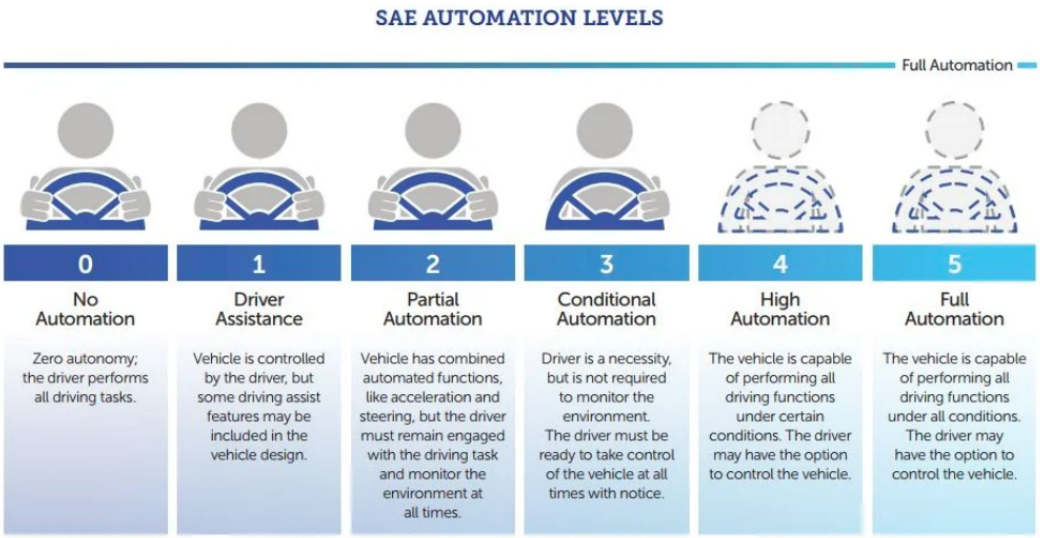


Figura 2.1: Distintos niveles de autonomía según la SAE [32].

2.1.1. Sensores

Un coche autónomo está equipado con una serie de componentes tecnológicos avanzados que le permiten "ver" el mundo que lo rodea y tomar decisiones en consecuencia. En la Figura 2.2 podemos ver algunos de estos componentes, aunque hay vehículos autónomos que constan de más sensores de los que se ven en la imagen, dependiendo de su autonomía y características. Los sensores más habituales en un vehículo autónomo son los siguientes:

- **GNSS** *(Global navigation satellite system)*: es un sistema de navegación por satélite que permite al coche autónomo conocer su posición exacta en todo momento. Esto es esencial para la planificación de rutas de conducción precisas y para asegurarse de que el vehículo llegue a su destino de manera segura.

- **IMU** *(Inertial Measurement Unit)*: es un dispositivo que utiliza giroscopios y acelerómetros para medir la velocidad y la dirección del vehículo. Esto permite al coche autónomo detectar cualquier cambio en su movimiento y ajustar su ruta de conducción, en consecuencia, este sensor permite mejorar la localización dada por el GNSS.

- **LiDAR** *(Laser Imaging Detection And Ranging)*: es un sistema de detección que utiliza láseres para medir la distancia entre el vehículo y los objetos que lo rodean. Esto permite que el coche autónomo "vea" el mundo que lo rodea y detecte obstáculos que puedan estar en su camino.

- **Radar** *(Radio Detection And Ranging)*: es un dispositivo que utiliza ondas de radio para detectar objetos a larga distancia y medir su velocidad y dirección de movimiento. Esto permite que el coche autónomo tenga una comprensión más completa de su entorno y pueda tomar decisiones informadas sobre cómo conducir de manera segura.

- **Sensores de ultrasonidos:** son dispositivos que envían ondas de sonido a través del aire para detectar objetos cercanos al vehículo. Esto es especialmente útil para detectar objetos que pueden estar ocultos a la vista de las cámaras y el LiDAR, como objetos bajos y pequeños. Se suelen usar para asistir al conductor en maniobras de aparcamiento.

- **Cámaras:** son componentes clave en los coches autónomos ya que permiten que el vehículo "vea" el mundo que lo rodea y reconozca objetos y patrones. Esto es esencial para identificar objetos o detectar cambios en el entorno que pueden ser difíciles de detectar con otros sensores.

- **Ordenador central:** es el "cerebro" del coche autónomo. Este dispositivo recopila y procesa la información de los diferentes sensores y utiliza esta información para tomar decisiones informadas sobre cómo conducir de manera segura. El ordenador central también puede ajustar la velocidad y la dirección del vehículo para evitar obstáculos y garantizar una conducción segura.

- **Sensores de odometría:** son dispositivos que miden el movimiento y la posición de un vehículo utilizando los datos de rotación de las ruedas. Estos sensores se utilizan para calcular la distancia recorrida por el vehículo y su posición actual.

Para que estos sensores puedan realizar su función es necesario calibrarlos adecuadamente. Para ello, hay que tener dos conceptos en cuenta: los extrínsecos e intrínsecos de los sensores.

Los extrínsecos de un sensor se refieren a las características o propiedades que están relacionadas con el entorno o las condiciones externas en las que el sensor está operando. Los extrínsecos son la pose de un sensor respecto a un punto en concreto del coche. Llegar a un nivel de precisión alto a la hora de calibrar un sensor es una tarea crucial. Pequeños

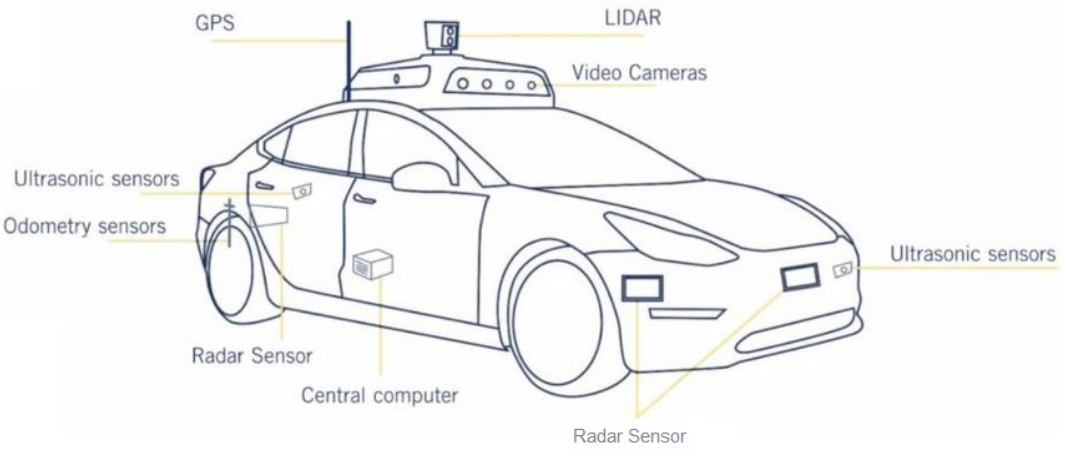


Figura 2.2: Componentes principales de un coche autónomo [33].

errores en la pose del sensor pueden suponer varios metros de error a la hora de ubicar objetos detectados a media-larga distancia del vehículo.

Los intrínsecos de un sensor se refieren a las características o propiedades inherentes y fundamentales del propio sensor, que están relacionadas con su diseño, funcionamiento interno y capacidad para medir o detectar una determinada variable. Por ejemplo, el rango de un LiDAR o el angulo de visión de una cámara.

La calibración de los sensores es fundamental a la hora de desarrollar un vehículo autónomo por varios motivos, debido a que estos aportan información acerca del entorno del mismo, como puede ser la distancia a una intersección u objeto. Además, cuando entran en juego varios sensores, la calibración coge mayor importancia, ya que es necesario relacionar la información que aportan los diferentes sensores.

Cuando se habla de conducción autónoma, uno de los conceptos a tener en cuenta es el sistema de coordenadas del vehículo. Definir un sistema de coordenadas estándar puede ser de utilidad para que investigadores y empresas utilicen un sistema común y, de esta manera, centralizar la manera en la que se trabaja con estos datos. En la actualidad hay una gran variedad de sistemas de coordenadas definidos. En cuanto a la automoción, uno de los más habituales es el sistema de coordenadas ISO-8855 [34]. Este es un sistema de coordenadas pensado para la automoción, este se puede apreciar en la Figura 2.3 y estas son sus características principales:

- El origen del sistema de coordenadas se encuentra en el punto central del eje trasero del vehículo proyectado en el suelo.
- El eje x va desde la parte trasera del vehículo a la parte delantera del mismo.
- El eje y es perpendicular al eje x y toma sus valores positivos a la izquierda del vehículo.
- El eje z es perpendicular a los ejes (x, y) y toma sus valores positivos encima del vehículo.

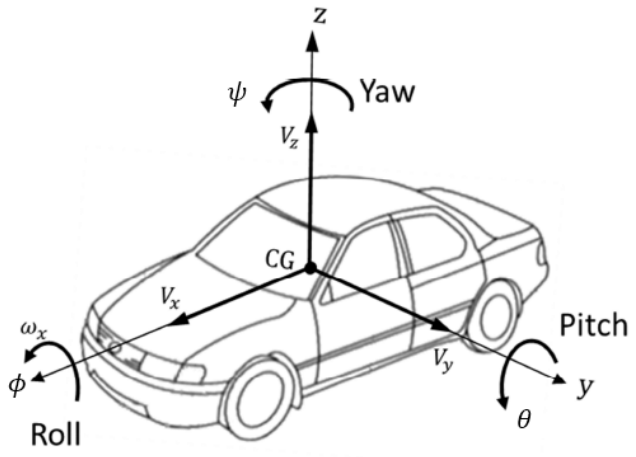


Figura 2.3: Sistema de coordenadas ISO-8855 de los vehículos. Se pueden observar la dirección de los ejes de coordenadas. $Roll$, $pitch$, yaw corresponden a la orientación del vehículo en los ejes x, y, z respectivamente. Fuente: [35]

2.2. Transformaciones de sistemas de coordenadas

Un tema recurrente a lo largo de la memoria son las transformaciones entre sistemas de coordenadas, las cuales son transformaciones en el espacio 3D. Estas transformaciones se realizan haciendo operaciones matriciales y para ello es necesario definir una matriz de transformación. Estas matrices pueden ser usadas para realizar distintas transformaciones a puntos y vectores, tales como traslación, rotación, escalado, efecto espejo, etc. En concreto, para este proyecto se han utilizado las matrices de traslación y rotación. Una propiedad fundamental de las matrices de transformación es que la inversa de la matriz representa la transformación opuesta. Por ejemplo, si una matriz rota $90°$ en *yaw*, su inversa rotará $-90°$ en *yaw*.

La matriz de traslación permite trasladar puntos, dadas las coordenadas iniciales (x, y, z) y el vector de translación (a, b, c). Esta translación se obtiene sumando el vector de coordenadas a las coordenadas iniciales.

$$\begin{aligned} x' &= x + a \\ y' &= y + b \\ z' &= z + c \end{aligned} \tag{2.1}$$

Este sistema de ecuaciones se puede definir mediante la siguiente matriz:

$$\begin{pmatrix} 1*x & 0*y & 0*z & a \\ 0*x & 1*y & 0*z & b \\ 0*x & 0*y & 1*z & c \\ 0*x & 0*y & 0*z & 1 \end{pmatrix} \tag{2.2}$$

Esta matriz puede ser descompuesta en matriz por vector para obtener la definición de la matriz de traslación, donde el vector es la posición original (x, y, z), por tanto la matriz resultante (en coordenadas homogéneas) es:

$$T(a,b,c) = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

La matriz de rotación permite rotar puntos y vectores respecto a los ejes del sistema de coordenadas, esta matriz se define divide en tres matrices, una por cada eje, las cuales son:

$$\begin{aligned} R_x(\alpha) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ R_y(\beta) &= \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ R_z(\gamma) &= \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{2.4}$$

donde $R_x(\alpha)$ rota α grados en el eje x, $R_y(\beta)$ rota β grados en el eje y y $R_z(\gamma)$ rota γ grados en el eje z.

Tanto las matrices de rotación como traslación se pueden concatenar para aplicar varias transformaciones con una sola operación, por ejemplo se puede definir una matriz de rotación

$$\begin{aligned} R(\alpha,\beta,\gamma) &= R_z(\alpha)R_y(\beta)R_z(\gamma) \\ R(\alpha,\beta,\gamma) &= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & 0 \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma - \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma + \cos\alpha\sin\gamma & 0 \\ -\sin\beta & \cos\beta\sin\gamma & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{2.5}$$

Uno de los procesos más habituales es transformar un punto P de un sistema de coordenadas A a un sistema de coordenadas B, para ello es necesario conocer los siguientes datos:

- (x_{PA}, y_{PA}, z_{PA}): posición del punto P en el sistema de coordenadas A.
- $(\alpha_{PA}, \beta_{PA}, \gamma_{PA})$: orientación del punto P en el sistema de coordenadas A.
- (x_{AB}, y_{AB}, z_{AB}): posición del origen del sistema de coordenadas A respecto al sistema de coordenadas B.
- $(\alpha_{AB}, \beta_{AB}, \gamma_{AB})$: orientación del sistema de coordenadas A respecto al sistema de coordenadas B.

Se pueden multiplicar todas las matrices una a una al punto P y a su rotación, pero este método es muy poco eficiente. El método que se utiliza es representar el punto como una matriz de transformación y después obtener la posición y rotación de la matriz resultante. Además, generalmente estas operaciones se les suele aplicar a varios puntos, por lo que en vez de multiplicar todas las matrices, debido a que la multiplicación matricial cumple la propiedad asociativa, primero se genera una matriz de transformación que aplique todas las operaciones a la vez.

El proceso es el siguiente:

1. Se genera una matriz M_{PA} que represente al punto P.

$$M_{PA} = T(x_{PA}, y_{PA}, z_{PA})R(\alpha_{PA}, \beta_{PA}, \gamma_{PA}) \tag{2.6}$$

2. Se genera una matriz de traslación del origen del sistema de coordenadas A al sistema de coordenadas B.

$$T_{AB} = T(x_{AB}, y_{AB}, z_{AB})^{-1} \tag{2.7}$$

3. Se genera una matriz de rotación del sistema de coordenadas A al sistema de coordenadas B.

$$R_{AB} = R(\alpha_{AB}, \beta_{AB}, \gamma_{AB})^{-1} \tag{2.8}$$

4. Se concatenan ambas matrices.

$$M_{AB} = T_{AB}R_{AB} \tag{2.9}$$

5. Se premultiplica la matriz obtenida en el paso anterior M_{AB} a la matriz que representa el punto M_{PA}.

$$M_{PB} = M_{AB}M_{PA} \tag{2.10}$$

6. Se extraen los datos de la matriz M_{PB}

En caso de no tener datos de traslación y rotación de un sistema de coordenadas a otro, si ambos sistemas de coordenadas pertenecen a uno en común C, se pueden concatenar las operaciones generando primero la matriz de transformación del sistema de coordenadas A al C y después otra matriz del sistema de coordenadas C al B.

2.3. Nubes de puntos

Anteriormente, se ha hablado sobre el sensor LiDAR, el cual es capaz de generar vectores con datos acerca de la información que ha obtenido de sus láseres. Hay varios tipos de sensores LiDAR y es posible que cada uno de ellos aporte información diferente. En la automoción se suelen usar LiDARes rotativos y, más recientemente, de estado sólido. Este último aporta información como la distancia a la que colisiona cada láser, la identificación del láser que ha obtenido cada punto, la intensidad con la que ha rebotado el láser o el instante de tiempo en el que ha rebotado el láser. A partir de esta información, se pueden

generar nubes de puntos. Hay muchas marcas de LiDAR (Hesai [3], Livox [4], Blickfeld [5], Cepton, [6] entre otras). El LiDAR con el que se han generado las nubes de puntos de este estudio es el LiDAR Velodyne [7].

Las nubes de puntos son un tipo de datos tridimensionales que contiene información sobre el entorno. Se suele trabajar con ellas como flujo de datos y luego se suelen almacenar en distintos formatos. Los más habituales son Point Cloud Data (PCD) y Polygon File Format (PLY). En este proyecto se ha trabajado principalmente con el formato .pcd por lo que en este apartado se explicará como son este tipo de archivos.

Los ficheros .pcd son similares a los ficheros de texto, estos constan de dos partes: la cabecera y los puntos. La cabecera contiene información sobre los puntos, las variables que tiene cada punto, el tipo de variable, el número de puntos, el formato del texto, entre otros. Los puntos son similares a tablas, con n líneas de m palabras, siendo n el número de puntos y m el número de variables. Además, pueden estar codificados en formato ASCII o binario.

Aunque en la actualidad hay muchas librerías que permiten trabajar con nubes de puntos como Open3D [8] o pypcd [9], muchas veces las funcionalidades de estas son limitadas y es necesario trabajar con ellas tratándolas como ficheros de texto, lo cual puede dificultar ciertas tareas. En cuanto a la visualización de estas, también hay varias herramientas disponibles como MeshLab [10] o CloudCompare [11].

Si se dispone de la odometría del vehículo, es posible reconstruir el entorno del vehículo a partir de nubes de puntos. La odometría también puede utilizarse para acumular varias nubes de puntos con el objetivo de generar una nube de puntos más densa. La acumulación de nubes de puntos es usada para aportar información adicional a los modelos de aprendizaje profundo. En este proyecto se realiza este proceso. Por ello, en este apartado se explicará como hacerlo. Los puntos de las nubes de puntos pertenecen al sistema de coordenadas del LiDAR. El vehículo está en constante movimiento, lo que significa que un punto en la posición (x, y, z) en una nube de puntos no estará en la misma posición en otra nube de puntos, es por ello que es necesario realizar un cambio de sistema de coordenadas a todos los puntos que se van a acumular.

Antes de explicar el proceso es necesario entender ciertos conceptos. Se van a realizar transformaciones de sistemas de coordenadas entre el sensor LiDAR en un fotograma o *frame* y en otro. En ningún conjunto de datos están almacenadas las matrices para cambiar el sistema de coordenadas del LiDAR de un *frame* a otro porque se necesitaría mucho espacio. En su lugar se define un sistema de coordenadas global y por cada *frame* se almacena la matriz de transformación del vehículo respecto su posición inicial. También se almacena la matriz de extrínsecos de los sensores respecto al vehículo. Pero debido a que los sensores son estáticos, se puede utilizar la matriz de transformación del vehículo y la posición inicial de los sensores para cambiar los sistemas de coordenadas de los sensores de un *frame* a

[3] https://www.hesaitech.com/
[4] https://www.livoxtech.com/
[5] https://www.blickfeld.com/
[6] https://www.cepton.com/products/overview
[7] https://velodynelidar.com/
[8] http://www.open3d.org/
[9] https://github.com/dimatura/pypcd
[10] https://www.meshlab.net/
[11] https://www.danielgm.net/cc/

otro.

Primero, es necesario decidir cuál va a ser el sistema de coordenadas principal. Este puede ser el sistema de coordenadas global o el sistema de coordenadas del LiDAR en un *frame* concreto. En este ejemplo se utilizará como sistema de coordenadas principal el sistema de coordenadas del LiDAR en el *frame* t_n. También hay que decidir el número de nubes de puntos que se van a acumular. Para este ejemplo se acumularán dos (la principal y la obtenida en el *frame* t_{n-1}), pero se pueden acumular tantas como se desee. Además, no es necesario acumular nubes de puntos de *scans* de LiDAR seguidos. Una vez decidido cómo va a ser la nube de puntos resultante, hay que transformar los puntos del sistema de coordenadas del LiDAR en el *frame* t_{n-1} al sistema de coordenadas global. Para ello, se multiplican estos por la inversa de la matriz de transformación del sistema de coordenadas global a la posición del vehículo en el *frame* t_{n-1}. Después, hay que transformar los puntos del sistema de coordenadas global al sistema de coordenadas del LiDAR en el *frame* t_n. Con esa finalidad, se multiplican los puntos por la matriz de transformación del sistema de coordenadas global a la posición del vehículo en el *frame* t_n.

La diferencia entre una nube de puntos normal y una nube de puntos resultado de una acumulación de nubes de puntos, es la cantidad de puntos de esta. Esto produce que la nube sea más densa y los objetos se puedan reconocer de manera más sencilla, como se puede ver en la Figura 2.4. Además, si se dispone de las diferencias de tiempo, se puede tratar de aproximar la dirección y velocidad de los objetos con mayor facilidad.

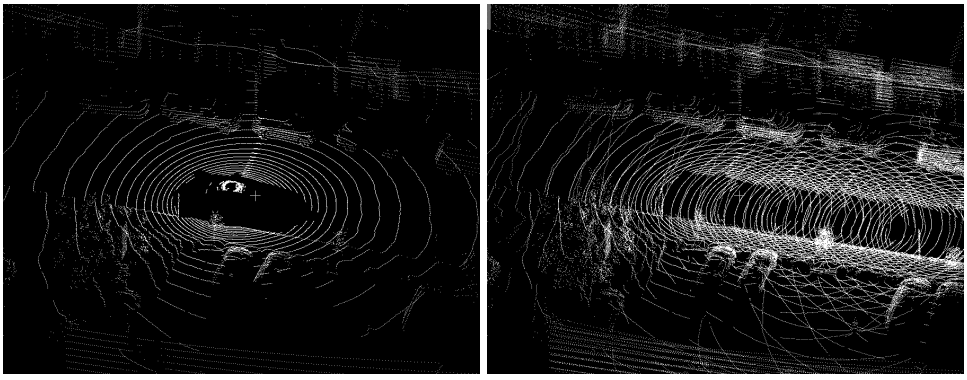


Figura 2.4: Comparación entre nubes de puntos de nuScenes [7] sin acumulación y con acumulación. A la izquierda una nube de puntos del conjunto de datos, a la derecha varias nubes de puntos consecutivas acumuladas, entre las que se encuentra la nube de puntos de la izquierda

CAPÍTULO 3

Estado del arte

El presente capítulo tiene como objetivo ofrecer un panorama completo y actualizado del estado del arte en el campo de la detección de objetos en nubes de puntos dentro de la conducción autónoma. Se presentarán los últimos avances en diferentes áreas, como la adquisición de datos y los modelos de aprendizaje profundo. En los últimos años, la investigación en conducción autónoma ha experimentado un gran avance gracias al desarrollo de nuevas tecnologías y a la colaboración entre diferentes disciplinas. Por tanto, el objetivo de este capítulo es ofrecer al lector una visión general de las tendencias más relevantes en el campo de la conducción autónoma y los desafíos que todavía existen en este campo en constante evolución.

3.1. Conjuntos de datos de conducción autónoma

El desarrollo de la conducción autónoma ha dependido en gran medida de la disponibilidad de conjuntos de datos de alta calidad para entrenar y validar los algoritmos de conducción autónoma. Entre los conjuntos de datos más populares se encuentran KITTI [8], Waymo [10] y nuScenes [7].

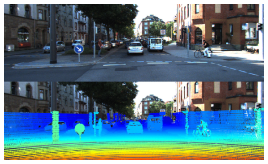
Los conjuntos de datos utilizados en la conducción autónoma suelen incluir tres tipos de información:

- Imágenes, que son capturadas por las cámaras del vehículo y pueden provenir de cuatro a seis cámaras para obtener una percepción completa del entorno del mismo. Además de las cámaras en el exterior del vehículo, algunos conjuntos de datos ofrecen imágenes del interior del vehículo para monitorizar al conductor [36, 37, 38].

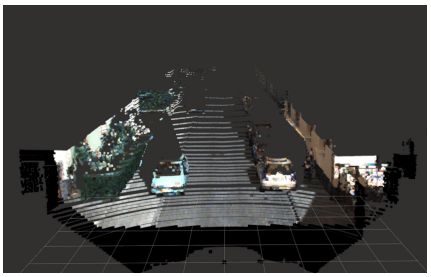
- Nubes de puntos, que se generan mediante el uso de radares o sensores LiDAR, lo que permite una percepción de profundidad de la escena y la simulación de un entorno tridimensional.

- Datos de odometría, que son obtenidos a través de GPS, IMU y sensores de odometría, y contienen información de posición y rotación del vehículo a lo largo de la grabación.

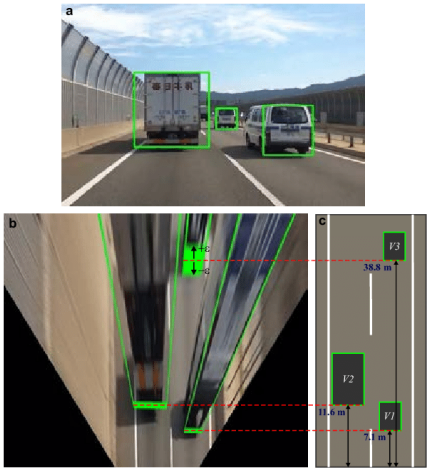
Estos datos permiten varias aplicaciones, tales como la creación de mapas de profundidad en las imágenes, la proyección de los colores de las imágenes en las nubes de puntos, la generación de una vista aérea a partir de las imágenes, y diferentes técnicas de visión por computador como la detección o el seguimiento de objetos como se puede ver en la Figura 3.1.



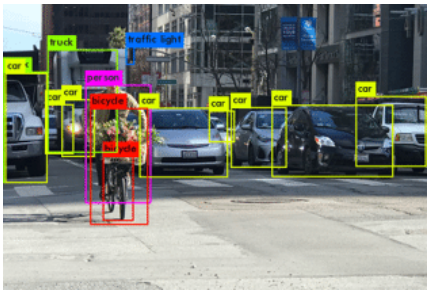
(a) Mapa de profundidad en imágenes. Fuente: [39].



(b) Reconstrucción de escenario mediante nubes de puntos y proyección de colores. Fuente: [40]



(c) Estimación de pose basada en vista aérea. Fuente: [41].



(d) Detección y clasificación de objetos. Fuente: [42]

Figura 3.1: Distintos ejemplos de aplicaciones de conjuntos de datos de automoción.

Sin embargo, los conjuntos de datos utilizados en la conducción autónoma presentan algunos desafíos, como los siguientes:

- **Limitaciones de los datos**. Dado que se obtienen en un entorno real, no se puede recolectar información de situaciones peligrosas, como colisiones de vehículos o atropellos de peatones.
- **Problemas de sincronización**. Los distintos sensores del vehículo suelen funcionar a distintas frecuencias, por tanto, la sincronización precisa de los datos es una tarea fundamental en el contexto de la conducción autónoma. Unos milisegundos de desincronización pueden suponer varios centímetros de diferencia a la hora de proyectar objetos en el entorno cercano del vehículo, suponiendo un grave problema en los algoritmos de decisión del vehículo.
- **Calibración de los sensores**. La calibración de sensores es otra de las fuentes de problemas en los datos obtenidos en un vehículo automatizado. La calibración

de parámetros extrínsecos de los sensores nos permite conocer la pose de estos respecto al vehículo, mientras que los parámetros intrínsecos nos permiten modelar matemáticamente las cámaras del vehículo para rectificar o corregir la distorsión de estos sensores. La calibración de sensores nos permite proyectar datos de distintos sensores (cámaras, LiDAR, etc.) en el sistema local del vehículo, por lo que una mala calibración puede resultar en un mapeo erróneo de los objetos detectados respecto al vehículo.

Es importante tener en cuenta estos desafíos al trabajar con conjuntos de datos de conducción autónoma, ya que pueden afectar la calidad y la precisión de los modelos de conducción autónoma desarrollados a partir de ellos. Además, para abordar estos desafíos y promover la investigación y el desarrollo en el campo de la conducción autónoma, en este trabajo se presentarán los conjuntos de datos más utilizados y reconocidos en la comunidad científica. Estos conjuntos de datos han sido recopilados en entornos reales y contienen una amplia variedad de escenarios de conducción, lo que permite a los investigadores evaluar y comparar el rendimiento de diferentes algoritmos y enfoques de conducción autónoma. Entre los conjuntos de datos destacados se encuentran KITTI, nuScenes y Waymo.

El conjunto de datos KITTI [8], es uno de los más antiguos y utilizados en la comunidad de conducción autónoma. KITTI fue el pionero entre los conjuntos de datos de conducción autónoma y ha desempeñado un papel fundamental en el avance de la investigación y el desarrollo de sistemas de visión por computadora para la percepción de vehículos y peatones en entornos urbanos.

El vehículo utilizado para la captura de datos en el conjunto de datos KITTI está equipado con una variedad de sensores que permiten la adquisición de información detallada sobre el entorno. En primer lugar, el vehículo está equipado con múltiples cámaras estéreo de alta resolución. Estas cámaras están ubicadas estratégicamente en diferentes puntos del vehículo para capturar vistas estéreo sincronizadas. Además de las cámaras estéreo, el vehículo también cuenta con un sensor LiDAR. La información del LiDAR es especialmente valiosa para la detección y reconstrucción tridimensional de objetos, así como para la generación de mapas de alta fidelidad.

El conjunto de datos de Waymo [10] es una colección de datos de percepción de vehículos autónomos recopilados por Waymo LLC, una compañía subsidiaria de Google. Este conjunto de datos se ha convertido en una referencia en la comunidad de investigación y desarrollo de vehículos autónomos debido a su tamaño masivo, su diversidad y su alta calidad.

Este contiene datos recopilados de una flota de vehículos autónomos equipados con diversos sensores, como cámaras, LiDAR y radar. Estos vehículos han recorrido diferentes áreas urbanas y suburbanas, lo que garantiza una variada gama de escenarios de conducción. Los datos están organizados en secuencias temporales, lo que permite el desarrollo de modelos que pueden aprender a comprender el contexto y la dinámica de conducción a lo largo del tiempo.

El conjunto de datos nuScenes [7] es un conjunto de datos públicos de gran escala para conducción autónoma. El objetivo de este conjunto de datos es apoyar la investigación pública en visión por computador y conducción autónoma. Este conjunto de datos cuenta con 1000 escenas, cada escena tiene una duración aproximada de 20 segundos y se seleccionaron

manualmente para mostrar una serie diversa e interesante de maniobras de conducción, situaciones de tráfico y comportamientos inesperados.

El conjunto de datos nuScenes ofrece una variedad de sensores utilizados en vehículos autónomos, incluyendo cámaras, LiDAR, radares y sensores de posicionamiento. Esta amplia gama de sensores permite a los investigadores abordar desafíos más complejos en la conducción autónoma, como la detección y seguimiento de objetos en entornos urbanos dinámicos y desafiantes.

Cada escena en el conjunto de datos nuScenes está anotada con información detallada, como la posición 3D y la categoría de los objetos, las trayectorias de los vehículos y peatones. Estas anotaciones precisas y completas proporcionan una base sólida para el desarrollo y evaluación de algoritmos de detección, seguimiento y predicción de objetos.

Además, nuScenes ofrece una amplia gama de desafíos en términos de condiciones de iluminación, clima y densidad de tráfico. Los escenarios incluyen situaciones urbanas complejas, como intersecciones concurridas, carriles estrechos y áreas de construcción, lo que permite a los investigadores evaluar y mejorar la robustez y el rendimiento de sus algoritmos en situaciones del mundo real.

Los datos recopilados en este conjunto de datos provienen de una flota de vehículos equipados con una amplia gama de sensores, incluyendo un GPS, seis cámaras, un sensor LiDAR y múltiples radares. Entre estos sensores, se destaca la presencia estratégica de seis cámaras, que permiten capturar información visual del entorno circundante del vehículo, tal como se ilustra en la Figura 3.2. Además, el sensor LiDAR desempeña un papel fundamental al generar nubes de puntos que representan con precisión el entorno, como se puede ver en la Figura 3.3, mientras que el GPS brinda información exacta sobre la localización del vehículo.

Figura 3.2: Imágenes del conjunto de datos nuScenes [7]. Seis imágenes pertenecientes al mismo *frame* captadas por cada una de las cámaras del vehículo con el que se han recopilado los datos.

3.2. Generación de datos sintéticos y sus usos

En el ámbito de la industria automotriz, la recolección y análisis de datos son elementos fundamentales tanto para el desarrollo de vehículos autónomos como para la implementa-

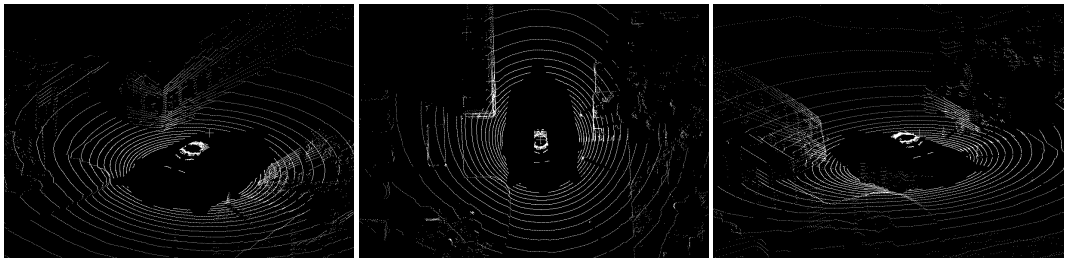


Figura 3.3: Nube de puntos del conjunto de datos nuScenes [7] vista desde diferentes perspectivas. Esta nube de puntos corresponde al mismo frame que las imágenes de la Figura 3.2.

ción de funciones de asistencia en la conducción, como el sistema de frenado de emergencia. Sin embargo, este proceso de recopilación de datos representa un reto considerable, ya que implica la instalación y calibración de sensores, así como la grabación continua de datos durante períodos extensos, que pueden llegar a ser de varias horas o incluso días. Esta tarea demanda tanto tiempo como recursos financieros significativos. Además de estas tareas, una vez recopilados los datos es necesario anotarlos de manera manual, lo que también requiere de bastante personal preparado, tiempo y recursos financieros.

No obstante, gracias a las herramientas más actuales como CARLA [16], es posible generar datos sintéticos realistas de automoción. CARLA es un simulador de automoción realista con una gran variedad de sensores, vehículos y escenarios. Estos datos sintéticos se crean artificialmente y simulan situaciones reales en el entorno de conducción. Utilizando esta tecnología, se puede ahorrar tiempo y dinero al generar una cantidad significativa de datos sin necesidad de realizar costosas y prolongadas grabaciones en carretera.

Gracias a esta herramienta, se han podido generar muchos conjuntos de datos [43, 44, 45, 46, 47, 48, 49] con diferentes sensores y fines. Además, se ha demostrado la utilidad de estos datos en diferentes ámbitos como la detección de objetos en imágenes y nubes de puntos [50, 51, 52, 53, 54, 55].

Actualmente, gracias a los últimos avances en la visión por computador, se pueden utilizar redes neuronales generativas [56, 57] para la creación de imágenes. Entre estas imágenes de automoción, incluso hay trabajos sobre el uso de este tipo de redes para la generación de nubes de puntos [58, 59]. A pesar de todas las ventajas de la generación de datos sintéticos, estos no son idénticos a los reales. Esto produce que haya una brecha de dominio. Este es un factor a tener en cuenta a la hora de utilizar estos datos, ya sea para entrenamiento de modelos o para análisis de resultados.

3.3. Detección de objetos en nubes de puntos

La detección de objetos en nubes de puntos es un área de investigación en constante evolución en el campo de la percepción artificial, la visión por computador y la robótica. La detección de objetos en nubes de puntos se refiere a la tarea de identificar y localizar objetos en entornos tridimensionales representados como nubes de puntos. Estos entornos están formados por miles de puntos. Además, adicionalmente a las coordenadas de los puntos, las nubes de puntos pueden tener información como la intensidad de cada punto, el instante de tiempo en el que se ha obtenido la información, etc.

La detección de objetos en nubes de puntos es una tarea fundamental en aplicaciones como la navegación de vehículos autónomos, la inspección de estructuras y la planificación de la fabricación. Sin embargo, esta tarea es muy difícil debido a la complejidad y la variabilidad de las nubes de puntos, que pueden contener ruido, oclusiones y objetos con formas y tamaños muy diferentes.

En los últimos años, se han propuesto diversas técnicas para la detección de objetos en nubes de puntos [26, 22, 23, 24, 27, 28, 25]. Algunos de los enfoques más recientes incluyen redes neuronales convolucionales en 3D, técnicas de *clustering* y segmentación, y enfoques basados en grafos.

La estructura general de las redes neuronales utilizadas para la detección de objetos en nubes de puntos (Figura 3.4) se divide típicamente en cinco partes:

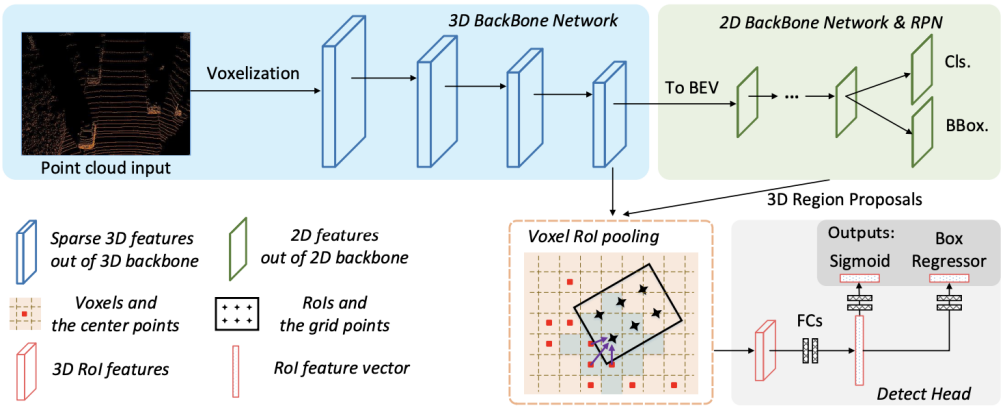


Figura 3.4: Estructura de las redes neuronales de detección de objetos en las nubes de puntos. Se pueden apreciar los distintos elementos de las arquitecturas típicamente usadas para la detección de objetos en nubes de puntos: codificador de voxels, estructura principal, cuello, cabeza de red de proposición de regiones y cabeza de región de interés. Fuente: [60].

- **Codificador de vóxeles (*Voxel Encoder*)**. Esta es la primera etapa de la red y se encarga de transformar la nube de puntos en una representación de *voxels* (cubos tridimensionales) que son procesados de manera más ágil por una red neuronal. En esta etapa, se agrupan los puntos de la nube en *voxels* y se calculan las características de cada *voxel*, como la densidad o la intensidad de los puntos. Otros modelos utilizan pilares en vez de *voxels*, los pilares son una forma de discretizar o subdividir el espacio tridimensional de la nube de puntos en celdas tridimensionales con forma de pilares.

- **Estructura principal (*Backbone*)**. La segunda etapa de la red es el *backbone*, que es una red neuronal convolucional 3D que se encarga de extraer características de los *voxels*. Esta etapa se basa en la arquitectura de una red neuronal pre-entrenada, como una red ResNet [61], que se adapta para su uso en la detección de objetos en nubes de puntos.

- **Cuello (*Neck*)**. La tercera etapa es el cuello o *neck*, que es un módulo que se utiliza para fusionar características de diferentes escalas en la red. Esta etapa se encarga de combinar características de alto nivel y bajo nivel de la nube de puntos para mejorar la detección de objetos.

- **Cabeza de red de proposición de regiones (*RPN Head*)**. La cuarta etapa es el *RPN Head*, que es un detector de regiones de interés (*Region Proposal Network*) que se encarga de proponer posibles regiones donde se puedan encontrar objetos en la nube de puntos. Este módulo utiliza la información de las características de la nube de puntos y propone una serie de regiones que pueden contener objetos.
- **Cabeza de región de interés (*ROI Head*)**. La última etapa es el *ROI Head*, que se encarga de clasificar y localizar objetos en las regiones propuestas por el *RPN Head*. Este módulo utiliza las características de las regiones propuestas para detectar y clasificar objetos en la nube de puntos.

A pesar de esto, cada modelo trata de innovar y obtener mejores resultados que los anteriores. Por ello, a pesar de que todos se basen en una misma estructura, cada uno de ellos es diferente. A continuación se explicarán las arquitecturas y cualidades de los modelos del estado del arte utilizados en este proyecto.

3.3.1. PointPillars

PointPillars [22] consta de tres partes como se puede ver en la Figura 3.5:

- **Codificador de Características (*Feature Encoder*)**: transforma la nube de puntos a una pseudo-imagen.
- **Estructura de Convolución 2D (*2D Convolutional Backbone*)**: transforma la pseudo-imagen en una representación intermedia.
- **Cabeza de Detección (*Detection Head*)**: detecta los cuboides a partir de la representación.

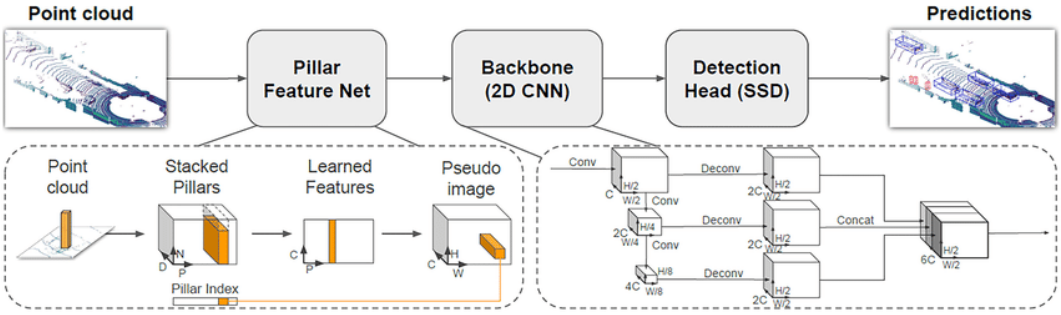


Figura 3.5: Estructura del modelo de detección de objetos en nubes de puntos PointPillars [22]. Los principales componentes de la red son: una red de características de los pilares, la red principal, y una cabeza de detección SSD [62]. La nube de puntos se convierte en un tensor de pilares apilados, después se extraen las características que pasan por la red principal para que finalmente la cabeza de detección prediga los bounding boxes.

3.3.1.1. *Feature Encoder*

Para aplicar una convolución 2D, primero es necesario generar una pseudo-imagen a partir de una nube de puntos. Inicialmente, la nube de puntos tiene 5 dimensiones: *x, y, z, intensity* y

time. Estos puntos se agrupan creando pilares en los ejes x e y. Una vez generados los pilares, se aumentan los puntos añadiéndoles 5 dimensiones extra, denominadas: x_c, y_c, z_c, x_p y y_p, donde el subíndice c denota la distancia a la media de los puntos, mientras que el subíndice p denota el desplazamiento desde el centro del pilar. De esta manera los puntos se quedan con $D = 10$ dimensiones.

Debido a la distribución de puntos, la mayoría de pilares están vacíos y aquellos que no lo estén, tendrán una densidad muy pequeña de puntos. Esta escasez es aprovechada imponiendo un límite tanto en el número de pilares no vacíos por muestra (P) como en el número de puntos por pilar (N) para crear un tensor de tamaño (D, P, N). Si un pilar contiene demasiados datos para caber en este tensor, los datos se seleccionan de forma aleatoria. Por otro lado, si una muestra o pilar tiene muy pocos datos para completar el tensor, se aplica relleno con ceros o *zero padding*.

Una vez obtenidos los pilares, a cada punto se le aplica una función lineal con *batch normalization* [63] y una función de activación ReLU [64] obteniendo un tensor de salida de dimensiones (C, P, N). Seguido por una operación *max* sobre los canales para crear una salida de dimensiones (C, P).

Tras obtener las características, estas son remapeadas a los pilares originales, generando finalmente la pseudo-imagen de tamaño (C, H, W).

3.3.1.2. *Backbone*

El *backbone* consta de dos subredes: una red descendente (*top-down*), que produce características con una resolución espacial cada vez más pequeña, y una segunda red que realiza el *upsampling* (aumento de la resolución) y concatenación de las características obtenidas por la red descendente.

La red descendente está compuesta por una serie de bloques, llamados $Block(S, L, F)$. Cada bloque opera con un *stride* S. Un bloque consta de L capas de convolución 2D de tamaño 3×3, con F canales de salida. Después de cada capa de convolución, se aplica una capa de *batch normalization* y una función de activación ReLU.

La primera convolución dentro de cada bloque tiene un *stride* $\frac{S}{S\_in}$ para asegurarse de que el bloque opere en la resolución espacial indicada por S después de recibir una entrada de resolución S_in. Todas las convoluciones subsiguientes dentro de un bloque tienen un *stride* de 1.

Las características finales de cada bloque de la red descendente se combinan mediante *upsampling* y concatenación de la siguiente manera:

Primero, las características se aumentan de tamaño mediante una convolución 2D transpuesta llamada $Up(S\_in, S\_out, F)$. Esta convolución aumenta la resolución espacial de las características de una resolución inicial S_in a una resolución final S_out. El resultado de esta convolución tiene F características finales.

A continuación, se aplica BatchNorm y ReLU a las características aumentadas de tamaño.

Las características finales de salida se obtienen concatenando todas las características que se originaron en diferentes resoluciones.

3.3.1.3. *Detection Head*

PointPillars utiliza la misma estrategia que Single Shot Detector (SSD) [62] para llevar a cabo la detección 3D. Al igual que en SSD, compara los *bounding boxes* (salida del *backbone*) con los etiquetados usando una medida de solapamiento en 2D (la unión sobre la intersección o IoU por sus siglas en inglés). La altura y la elevación del cuboide no se utilizaron para el emparejamiento de las detecciones con las anotaciones; en su lugar, dado un emparejamiento 2D, la altura y la orientación se convierten en objetivos de regresión adicionales. Los parámetros estimados por el modelo son los siguientes: posición en los ejes (x, y, z), profundidad, altura, anchura y orientación.

3.3.2. Shape Signature Network (SSN)

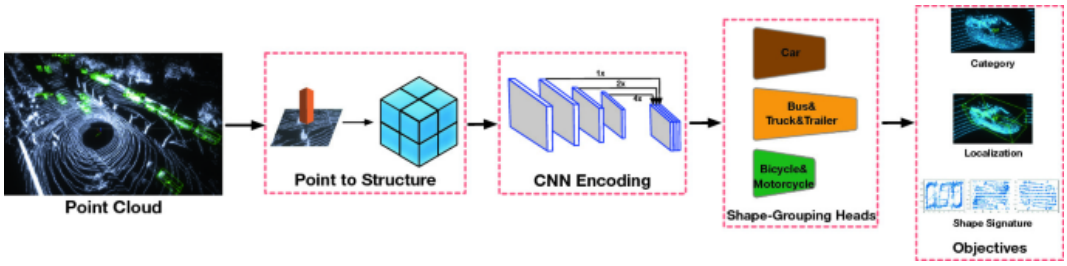


Figura 3.6: Estructura del modelo de detección de objetos en nubes de puntos Shape Signature Network (SSN) [23]. Esta red esta formada por cuatro componentes: El primer componente convierte la representación de puntos en una representación más simple como voxels o pilares. La red principal, que extrae las características de la representación de la nube de puntos. El tercer componente agrupa las detecciones de la red anterior en base al tamaño. Y la última parte que se encarga de localizar y clasificar los objetos.

SSN [23] es un modelo de aprendizaje profundo con el objetivo de detectar y clasificar objetos de diferentes clases en nubes de puntos. Para lograr esto, SSN consta de cuatro partes como se puede ver en la Figura 3.6 en las que se hace uso del tamaño de los objetos para clasificarlos. Estas cuatro partes son: *Point-to-Structure*, *Pyramid Feature Encoding*, *Shape-aware Grouping Heads* y *Multi-task objectives*.

Los componentes clave de la SSN son el *Shape Signature* y los *Shape-Aware Grouping Heads*. En particular, durante el entrenamiento, el *Shape Signature* puede guiar el aprendizaje de características discriminatorias mediante retropropagación o *backpropagation*, lo que favorece la discriminación multiclase. Tras el entrenamiento, el *Shape Signature* deja de ser necesario.

3.3.2.1. *Shape Signature*

Haciendo uso del *ground truth* de los puntos, se realiza un preprocesado de las nubes de puntos, utilizado únicamente en el entrenamiento. Este preprocesado trata de que las nubes de puntos tengan dos propiedades: que sean compactas y efectivas, y que sean robustas frente a la escasez y al ruido. Para obtener esto, se realizan varias operaciones como se puede ver en la Figura 3.7:

- Completado de Forma (*Shape Completion*): debido a que los *scans* del LiDAR únicamente muestran parte de los objetos (2 caras, máximo 3), el objetivo de esta parte

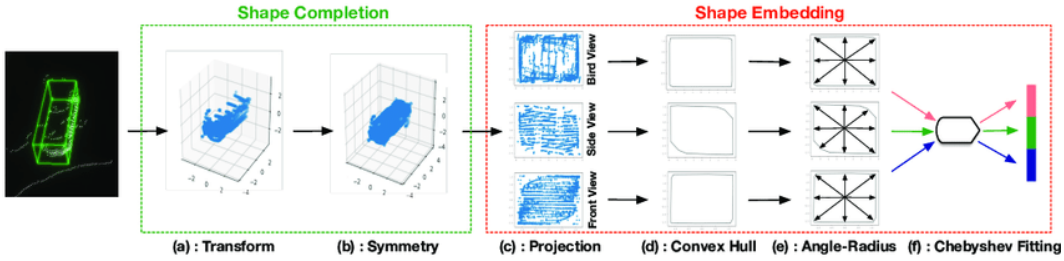


Figura 3.7: *Workflow* del modelo de detección de objetos en nubes de puntos *Shape Signature* [23]. *Shape completion* y *Shape Embedding* están ilustradas en dos en dos rectángulos. (a) transforma el centro del *bounding box* en el punto de origen. (b) es la operación espejo respecto al eje y del objeto. (c) es la proyección de los puntos 3D a tres vistas. (d) es la extracción del *convex hull*. (e) es el *angle radius*. (f) es el ajuste de Chebyshev.

es completar la forma de estos para solucionar este problema. Para ello, se aplica la simetría central, que se puede entender como una transformación espejo respecto al eje x del objeto a los puntos, como se puede ver en la Figura 3.7.

- Integración de Forma (*Shape Embedding*): se aplican las siguientes operaciones para cumplir este objetivo.

  ***Proyección*:** una vez se tienen los puntos completos se proyectan de 3D a vistas 2D: *bird-view, front-view* y *side-view*, esto permite describir las formas 3D y reduce los parámetros.

  ***Envoltura Convexa (Convex Hull)*:** la organización de los puntos 2D es limitada para representar eficazmente la forma y sigue existiendo dispersión. Esta parte se introduce para caracterizar estos puntos 2D y enfatizar los contornos.

  ***Angle-Radius*:** para describir el *Convex Hull* y marcar el contorno, se ha diseñado la siguiente función:

  $$f(\theta) = dist(\sigma \xrightarrow{\theta} \mathbb{C}), \tag{3.1}$$

  donde $\mathbb{C}$ es el *Convex Hull* y $dist$ es la distancia de entre el punto de origen y la intersección, es decir, el radio. Esta implementación obtiene un vector de 360 dimensiones con los diferentes radios del *Convex Hull*.

  ***Ajuste de Chebyshev (Chebyshev Fitting)*:** el objetivo es aplicar Interpolación Polinómica de Chebyshev para una mejor aproximación de de la función $f(\theta)$. En este caso se utilizan polinomios de Chebyshev de primer tipo $T_n(x)$, que pueden ser definidos por la siguiente relación de recurrencia:

  $$\begin{aligned} T_0 &= 1 \\ T_1 &= x \\ T_{n+1} &= 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{3.2}$$

  Se define la fórmula para la aproximación de Chebyshev de la siguiente manera

  $$f(x) \approx \sum_{n=0}^{N} \alpha_n T_n(x), \tag{3.3}$$

donde α son los coeficientes y estos se pueden obtener mediante las fórmulas:

$$\begin{aligned} \alpha_0 &= \frac{1}{N+1}\sum_{n=0}^{N} f(x_n)T_0(x_n) \\ \alpha_j &= \frac{2}{N+1}\sum_{n=0}^{N} f(x_n)T_j(x_n) \end{aligned} \tag{3.4}$$

Se trata de aproximar la función *angle radius* como una función de tercer grado, al haber tres vistas (*bird-view*, *front-view* y *side-view*) el tensor resultante es de dimensión 9.

3.3.2.2. Arquitectura

La arquitectura se construye basándose en el preprocesado *Shape Signature*, esta consta de cuatro partes como se puede ver en la Figura 3.6:

- *Point-to-Structure*: transforma la nube de puntos a una estructura más ordenada, para esto hay varias opciones: *voxels*, pilares, *bird-view*. Estas son tres representaciones intermedias diferentes. Por un lado, los *voxels* y los pilares agrupan los puntos en diferentes estructuras, los *voxels* en cubos y los pilares en pilares. Por otro lado, el *bird-view*, o BEV, proyecta los puntos al plano (x, y) formando de esta manera una imagen. En el caso particular del modelo utilizado en este estudio, la estructura utilizada ha sido pilares, donde cada pilar contiene un máximo de puntos de la nube con la información original de estos. Tras la obtención de la representación estructurada se pueden aplicar convoluciones 2D o 3D.

- Codificación de Características en Pirámide (*Pyramid Feature Encoding*): esta implementación se ha realizado siguiendo la idea de FPN [65]. Se aplica primero una red convolucional descendente para extraer las características de múltiples resoluciones espaciales. Luego, todas las características se fusionan mediante un proceso de *upsampling* y concatenación.

- *Shape-aware Grouping Heads*: en esta capa el modelo separa los diferentes objetos detectados por la red anterior en función del tamaño de estos. Haciendo que cada uno de estos pase por una *Detection Head* diferente, de esta manera cada red aprende pesos distintos. En cada *Detection Head* hay una red tipo SSD [62], cuanto mayores sean los objetos mayor es esta red.

- Objetivos Multi-tarea (*Multi-task objectives*): este apartado tiene tres objetivos: Clasificación Multiclase (*multi-class classification*), Regresión de Localización (**localization regression**) y Regresión de Vectores de Forma (*shape vector regression*).

  Multi-class classification: esta tarea está basada en Second [66] y su uso del *focal loss* [67]

  $$\mathcal{L}_{cls} = -\alpha_t(1-p_t)^{\gamma}\log(p_t), \tag{3.5}$$

  donde p_t es la probabilidad del *bounding box*, se usan los valores $\alpha = 0.25$ y $\gamma = 2$.

Localization regression: para el *localization loss*, se usa *smooth L1 loss* [68] para minimizar la distancia entre las predicciones y el *ground truth*.

$$\mathcal{L}_{loc} = SmoothL1(\Delta b), \tag{3.6}$$

donde Δb es el *ground truth* incluyendo la posición (x, y, z), escala (w, h, l) y rotación (θ)

Shape vector regression: a diferencia del Localization regression, el modelo está entrenado directamente para hacer regresión del *shape vector*, este vector contiene el tamaño de las *bounding box* detectadas. En este caso también se utiliza *smooth L1 loss*.

$$\mathcal{L}_{shape} = SmoothL1(\mathbb{S}), \tag{3.7}$$

donde $\mathbb{S}$ es el *shape vector*.

El objetivo de las tres tareas es:

$$\mathcal{L} = \beta_1\mathcal{L}_{cls} + \beta_2\mathcal{L}_{loc} + \beta_3\mathcal{L}_{shape}, \tag{3.8}$$

donde β son valores constantes para equilibrar los resultados de cada *loss*, ya que el *shape loss* es mucho más grande que los otros dos. Por ello se toman los siguientes valores: $\beta_1 = 1.0$, $\beta_2 = 1.0$ y $\beta_3 = 0.5$.

3.3.3. CenterPoint

CenterPoint [24] hace uso de los mismos *backbones* de otros modelos de detección de objetos, como pueden ser VoxelNet [69] o PointPillars [22]. El *backbone* es capaz de detectar ciertos objetos, su *workflow* se puede ver en la Figura 3.8. La novedad que presenta este modelo es la implementación del *Detection Head*, que está dividida en los tres siguientes apartados.

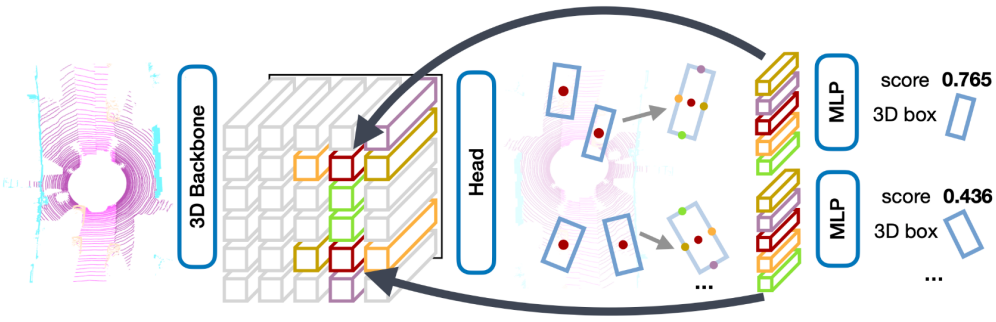


Figura 3.8: *Workflow* del modelo de detección de objetos en nubes de puntos CenterPoint [24]. El *backbone* extrae una representación de características de la nube de puntos. Después una arquitectura CNN [70] 2D encuentra el centro de los objetos y a partir de estos genera *bounding boxes*. Se extraen características de los *bounding boxes* y finalmente estas características y las características de la nube de puntos pasan por un perceptrón multicapa (MLP) [71] para obtener la puntuación de confianza.

3.3.3.1. Cabeza de Mapa de Calor del Centro (*Center Heatmap Head*)

El objetivo de esta parte de la arquitectura es producir un mapa de calor con picos en los objetos detectados por el *backbone*. Este *head* produce un mapa de calor por clase. Durante el entrenamiento, se utiliza un enfoque basado en una distribución gaussiana 2D generada a partir de la proyección de centros tridimensionales de los *bounding boxes* anotados en una vista de pájaro (BEV). Se emplea un *focal loss* [67] para el entrenamiento.

En la vista de mapa, los objetos son más dispersos en comparación con una imagen, además en la vista de mapa, las distancias son absolutas, mientras que en la vista de imagen se ven distorsionadas debido a la perspectiva. Se aumenta la supervisión positiva para el mapa de calor objetivo al ampliar el pico gaussiano generado en cada centro de los *ground truth*. Específicamente, se establece el radio gaussiano (θ) en $\theta = max(f(wl), \tau)$, donde $\tau = 2$ es el radio gaussiano mínimo permitido y f es una función de radio definida en Corner-Net [72]. De esta manera, CenterPoint mantiene la simplicidad de la asignación de objetivos basada en centros, y el modelo recibe una supervisión más densa de los píxeles cercanos.

3.3.3.2. Cabezas de Regresión (*Regression Heads*)

Se almacenan varias propiedades de objetos en las características centrales de los objetos. Estas propiedades incluyen: la ubicación del *bounding box*, altura sobre el suelo, tamaño en 3D y un ángulo de rotación yaw. El refinamiento de ubicación *bounding box* reduce el error de cuantificación de la voxelización. La altura sobre el suelo ayuda a localizar el objeto en 3D y agrega la información de elevación faltante eliminada por la proyección de vista en planta. La predicción de orientación utiliza el seno y coseno del ángulo de giro como objetivo de regresión continua. Combinado con el tamaño de la caja, estos *Regression Heads* proporcionan toda la información del *bounding box* 3D (localización, tamaño y rotación). A cada una de estas detecciones le corresponde un *Detection Head* distinto. Durante el entrenamiento, solo se supervisan el *ground truth* de la posición utilizando *L1 Loss* [68]. Se realiza una regresión en tamaño logarítmico para manejar mejor las cajas de varias formas. Durante la inferencia, se extraen todas las propiedades indexando en las salidas del encabezado de regresión densa en la ubicación pico de cada objeto.

3.3.3.3. Cabeza de Velocidad y Seguimiento (*Velocity Head and tracking*)

Además de la información de los objetos mencionada anteriormente, CenterPoint también es capaz de detectar la velocidad de los objetos en los ejes x e y. Para lograr esto, se utilizan dos bird-view de entrada: la del *frame* actual y la del *frame* anterior. El bird-view o BEV, proyecta los puntos de la nube al plano (x, y) formando de esta manera una imagen. Estas vistas se utilizan para calcular la estimación de velocidad, que es la predicción de la diferencia en la posición del objeto entre el *frame* actual y el *frame* anterior. La estimación de velocidad se supervisa utilizando *L1 Loss* en la ubicación del objeto en tiempo real.

El *tracking* se realiza en la etapa de inferencia, se asocian las detecciones del *frame* actual con las del *frame* anterior. Para ello se utiliza la velocidad detectada, a los centros de los objetos se les aplica la negativa de la velocidad para asociarlos con los objetos del *frame* anterior mediante la mínima distancia. Las detecciones que no han sido emparejadas, se mantienen guardadas durante 3 *frames* con su última velocidad conocida para emparejarlas en futuras detecciones.

3.3.3.4. CenterPoint de Dos Etapas (*Two-Stage CenterPoint*)

Se usa CenterPoint sin cambios como primera etapa. La segunda etapa extrae características adicionales de puntos a partir de la salida del componente principal. Se extrae una característica de punto del centro de cada cara del *bounding box* predicho. Para cada punto, obtenemos una característica mediante interpolación bilineal de la salida de la vista del mapa del componente principal M. A continuación, se concatenan las características de punto extraídas y se pasan a través de un MLP. La segunda etapa realiza la predicción de una puntuación de confianza sin relación con la clase y un refinamiento del *bounding box* sobre los resultados de predicción de CenterPoint de una sola etapa.

Para la predicción de la puntuación de confianza sin relación con la clase, utilizamos un objetivo de puntuación I guiado por el IoU del *bounding box* predicho con el *ground truth*:

$$I = min(1, max(0, 2 \times IoU_t - 0.5)) \tag{3.9}$$

donde IoU_t es el IoU entre el *bounding box* propuesto t y el *ground truth*. El entrenamiento se realiza con un *binary cross entropy loss*:

$$\mathcal{L}_{score} = -I_t log(\hat{I}_t) - (1 - I_t) log(1 - \hat{I}_t) \tag{3.10}$$

donde $\hat{I}_t$ es la puntuación de confianza predicha. Durante la inferencia, se usa directamente la predicción de clase de CenterPoint de una sola etapa y se calcula la puntuación de confianza final como el promedio geométrico de las dos puntuaciones:

$$\hat{Q}_t = \sqrt{\hat{Y}_t * \hat{I}_t} \tag{3.11}$$

donde $\hat{Q}_t$ es la confianza de predicción final del objeto t, $\hat{Y}_t = max_{0 \leq k \leq K} \hat{Y}_{p,k}$ y $\hat{I}_t$ son las confianzas de la primera y segunda etapa del objeto t, respectivamente.

Para la regresión del *bounding box*, el modelo predice un refinamiento sobre las propuestas de la primera etapa y se entrena con un *L1 loss* [68].

3.3.4. TransFusion

TransFusion [25] es un modelo que fusiona nubes de puntos e imágenes para predecir objetos 3D como se puede ver en la Figura 3.9. Hay varios modelos previos a este que realizan esta fusión [27, 73] y que también están implementados para funcionar con la librería MMDetection3D. Sin embargo, este modelo es el que mejores resultados obtiene. Adicional a esto, TransFusion tiene una versión en la que utiliza solo datos del LiDAR, esa es la versión que ha sido utilizada para los experimentos realizados en este proyecto, debido a que el análisis de modelos multimodales queda fuera del alcance de este proyecto.

3.3.4.1. Transformer para Detección en 2D

Los Transformers [74], una arquitectura relativamente nueva, han sido ampliamente utilizados en la detección de objetos 2D [75, 76, 77, 78]. Estos usan una CNN [70] *backbone* para extraer las características de las imágenes y un transformer para convertir un set de *embeddings* (llamados *object query*) aprendidos en un set de predicciones. Las predicciones

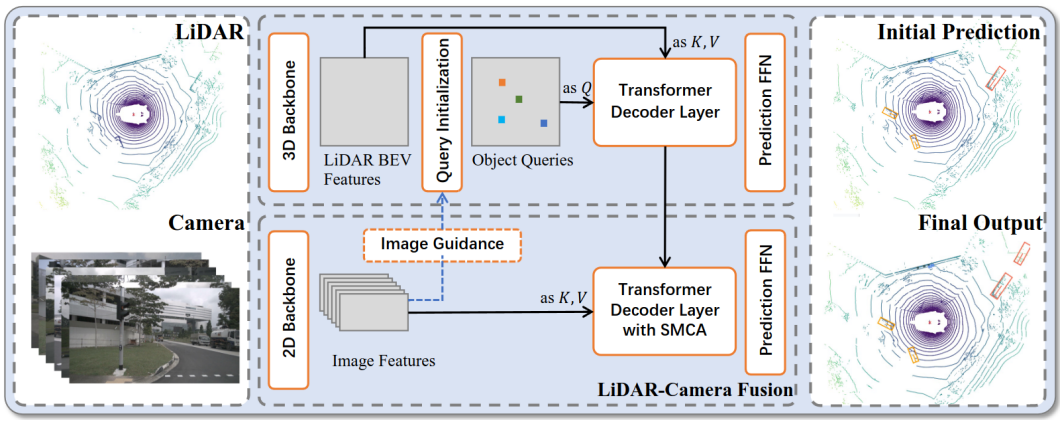


Figura 3.9: Estructura del modelo de detección de objetos en nubes de puntos TransFusion [25]. TransFusion utiliza redes 3D y 2D estándar para extraer mapas de características LiDAR BEV e imágenes. La etapa de detección consta de dos capas secuenciales de transformadores: (1) La primera capa genera *bounding boxes* iniciales utilizando características de objetos. (2) La segunda capa asocia y fusiona las características de objetos con las características de imagen, mejorando los resultados de detección con información de textura y color. Se introduce un mecanismo de atención cruzada para mejorar la atención a regiones de imagen relevantes.

finales son los offsets relativos. En el caso de TransFusion estos son la posición del objeto e información acerca del *bounding box*, como el tamaño, la orientación, etc.

3.3.4.2. Query Initialization

Input-dependent. TransFusion propone una estrategia de inicialización dependiente de la entrada basada en el mapa de calor para usar únicamente un capa de decodificación. Dado un mapa de características del LiDAR BEV $F_L \in \mathbb{R}^{X \times Y \times d}$ de d dimensiones, primero se predice el *heatmap* $\hat{S} \in \mathbb{R}^{X \times Y \times K}$ donde $X \times Y$ describe el tamaño del mapa de características del BEV y K son el número de categorías. El *heatmap* se crea de igual forma que el del modelo CenterPoint (véase sección 3.3.3). A continuación, se considera el *heatmap* como $X \times Y \times K$ candidatos y se seleccionan los N mejores como los *object queries* iniciales.

Category-aware. Los objetos en el plano BEV varían muy poco en escala dentro de una misma categoría. Para hacer uso de estas propiedades para una mejor detección multiclase, se hace que los *objects queries* sean sensibles a las categorías, equipando cada una con una *category embedding*. Esto proporciona beneficios en dos aspectos. Por un lado, añade información adicional a los módulos de auto atención y atención cruzada. Por otro lado, tiene conocimiento previo del objeto a la hora de la predicción.

3.3.4.3. Transformer Decoder y FFN

El *decoder layer* sigue el diseño de DETR [79]. La atención cruzada entre los *object queries* y los mapas de características (ya sea de nubes de puntos o de imágenes) agrega el contexto relevante a los objetos candidatos, mientras que la auto atención entre los *object queries* genera las relaciones por pares entre los diferentes objetos candidatos. Las *query positions* se agregan a las d-dimensional positional encodings con un Multilayer Perceptron [71], esto permite que la red sea capaz de asociar el contexto y la posición. Los *object queries* pasan por una Feed Forward Network (FFN) [80]. Al igual que CenterPoint la FFN predice

la posición (x, y, z), el tamaño (*length*, *width*, *height*), la orientación (yaw) y la velocidad en los ejes x, y (v_x, v_y), también se asigna una probabilidad por cada clase a cada predicción.

3.3.4.4. LiDAR-Camera Fusion

Extracción de Características de Imagen (*Image Feature Fetching*). A pesar de los últimos avances en la fusión a nivel de puntos entre imágenes y nubes de puntos, estas fusiones aún son un poco limitadas debido a la distribución de los puntos. Cuando un objeto sólo contiene un pequeño número de puntos LiDAR, sólo puede obtener el mismo número de características de la imagen, desperdiciando la información semántica de las imágenes de alta resolución. Para mitigar este problema, no se obtienen las características de la imagen multivista basándonos en la asociación rígida entre los puntos LiDAR y los píxeles de la imagen. En su lugar se retienen todas las características de la imagen como banco de memoria y se utiliza el mecanismo de atención cruzada para realizar la fusión de características de forma adaptativa y de escasa densidad, como se puede ver en la Figura 3.9. La atención cruzada es un método de fusión novedoso e intuitivo en el que las máscaras de atención de una modalidad se utilizan para resaltar las características extraídas en otra modalidad.

Atención Multi-Cabeza Espacial para Fusión de Características de Imagen (*SMCA for Image Feature Fusion*). *Multi Head Attention* es un mecanismo para construir asociaciones entre dos sets de *inputs*. Para mitigar la sensibilidad a la calibración del sensor y las características de la imagen que aporta la estrategia *hard-association*, se aprovecha el mecanismo de atención cruzada para construir el *soft association* entre LiDAR y las imágenes, lo que permite a la red determinar de forma adaptativa dónde y qué información debe tomarse de las imágenes. Primero se identifica la imagen específica en la que se encuentran los objetos y después se aplica atención cruzada entre los *object queries* y el *Feature Map* de la imagen. Sin embargo, como las características LiDAR y características de la imagen proceden de dominios completamente distintos, los *object queries* pueden referirse a regiones visuales no relacionadas con el *bounding box* que se desea predecir, lo que hace que la red necesite mucho tiempo de entrenamiento para identificar con precisión las regiones adecuadas en las imágenes. Se ha diseñado un módulo *Spatially Modulated Cross Attention* (SMCA) inspirados por [81] que pasa la atención cruzada por una máscara gaussiana 2D para cada centro de cada *object query*, proyectado en la imagen. La máscara gaussiana 2D M es generada de manera similar a la de CenterNet [82]

$$M_{i,j} = exp\left(-\frac{(i - c_x)^2 + (j - c_y)^2}{\sigma r^2}\right), \tag{3.12}$$

donde (i, j) son los índices espaciales de la máscara, (c_x, c_y) es el centro 2D de cada *object query* proyectado en la imagen, r es la distancia del centro del *bounding box* 3D a sus vértices y σ es un hiperparámetro para modular el *bandwidth* de la distribución gaussiana. La red suele centrarse principalmente en los píxeles en primer plano cerca de los objetos, ya que estos son los que más información aportan y esto agiliza el proceso. Tras el SMCA se utiliza otro FFN para producir el *output* final que contiene tanto información de *bounding boxes* 3D como 2D.

En el artículo de este modelo no se menciona, pero a pesar de que el punto fuerte de este modelo sea la fusión entre la cámara y el LiDAR, es posible utilizar solamente la parte del modelo enfocada en el LiDAR, que es la que se ha utilizado en este estudio.

CAPÍTULO 4

Metodología

En el campo de la automoción, los datos son cruciales, ya sea para el entrenamiento de modelos de aprendizaje profundo o bien para validar las funciones desarrolladas, como el frenado automático. La obtención de datos de automoción requiere una gran cantidad de tiempo y coste financiero. Es necesario disponer de un vehículo o una flota de vehículos a los que previamente se le han añadido y configurado los sensores. Tras eso, es necesario conducir el vehículo durante varias semanas o meses. Para finalizar, es necesario interpretar los datos y anotarlos de manera adecuada, para lo que hace falta mucho personal debido a la gran cantidad de datos que se generan. Además, hay situaciones que no se pueden generar debido a la peligrosidad que supondrían.

En la Tabla 4.1 obtenida del artículo *Driving to Safety* [83], se proporciona información sobre la cantidad de millas (años[1]) necesarias para validar vehículos autónomos o funciones ADAS. Dado que es costoso y a menudo impracticable obtener únicamente datos reales para lograr la cantidad, variabilidad y diversidad requeridas, los datos sintéticos son ampliamente utilizados como complemento. Estos datos sintéticos se generan para abarcar situaciones peligrosas o inusuales que pueden no ser fáciles de encontrar en datos reales, además evita el proceso y el coste que supone la obtención de datos reales. Para la generación de estos datos se recurre a entornos de simulación como CARLA[16] incluso se han publicado conjuntos de datos exclusivamente con datos sintéticos [43, 44, 45, 46, 47, 48, 49].

En este proyecto se ha generado un conjunto de datos sintético para en un futuro hacer uso del mismo con diferentes propósitos. Para ello se han replicado los sensores LiDAR, GNSS e IMU en un simulador de automoción. No solo eso, el conjunto de datos ha sido utilizado para evaluar modelos de aprendizaje profundo y estudiar como se comportan estos frente a un cambio de dominio de datos reales a sintéticos. Además, es importante destacar que a pesar de haber utilizado los sensores mencionados, se ha generado un *pipeline* fácil de utilizar que permite replicar la generación del conjunto de datos, brindando la oportunidad de añadir más sensores como cámaras o radares, e incluso sensores inexistentes en la vida real pero existentes en el simulador como cámaras de segmentación o LiDARes semánticos.

[1]Para calcular las millas que equivalen a un año se toma en cuenta una flota de 100 vehículos autónomos conduciendo las 24 horas del día, los 365 días del año, a una velocidad media de 25 millas por hora.

		Tasa de fallos de referencia		
Cuestión estadística	Durante cuantas millas (años) es necesario que se conduzcan vehículos autónomos...	(A) 1.09 víctimas mortales por cada 100 millones de millas?	(B) 77 lesiones reportadas por cada 100 millones de millas?	(C) 190 accidentes reportados por cada 100 millones de millas
	(1) para demostrar sin fallos con una tasa de confianza del 95 % que la tasa de error es como máximo...	275 millones de millas (12.5 años)	3.9 millones de millas (2 meses)	1.6 millones de millas (1 mes)
	(2) para demostrar con un 95 % de confianza su tasa de fallos dentro del 20 % de la tasa real de...	8.8 mil millones de millas (400 años)	125 millones de millas (5.7 años)	51 millones de millas (2.3 años)
	(3) para demostrar con un 95 % de confianza y una potencia estadística del 80 % que su tasa de fallos es un 20 % mejor que la tasa de fallos de los conductores humanos...	11 mil millones de millas (500 años)	161 millones de millas (7.3 años)	65 millones de millas (3 años)

Tabla 4.1: Ejemplos de millas y años necesarios para demostrar la fiabilidad de un vehículo autónomo. Fuente [83].

En este apartado se explicará el proceso que se ha seguido para la generación del conjunto de datos sintético, explicando detalladamente las herramientas utilizadas. Posteriormente, también se explicarán las características del conjunto de datos generado. Finalmente, se explicarán las métricas y configuración de los modelos que se utilizarán para posteriormente realizar un estudio comparando el comportamiento de los modelos frente al conjunto de datos generado y al conjunto de datos nuScenes [7].

4.1. Herramientas utilizadas a lo largo del proyecto

Como se ha mencionado anteriormente, la generación de un conjunto de datos de automoción necesita una gran cantidad de recursos y herramientas, vehículos, gasolina, anotadores humanos, etc. La ventaja de generar datos sintéticos permite evitar el uso de estas herramientas tan costosas. Pero eso no quiere decir que no sea necesario el uso de herramientas, así como conocimientos sobre estas herramientas y tiempo para generar los datos. En este apartado se hablará sobre las herramientas utilizadas para la generación del conjunto de datos sintético.

4.1.1. Simulador CARLA

CARLA [16] es un simulador de código abierto y gratuito basado en Unreal4 [2], diseñado para ayudar en la investigación y el desarrollo de sistemas de conducción autónoma. CARLA permite a los investigadores y desarrolladores simular entornos de conducción autónoma en un entorno virtual altamente realista. El simulador proporciona un conjunto de herramientas de alta fidelidad para simular sensores de vehículos como cámaras, radares y LiDAR, así como modelos físicos precisos para vehículos, peatones y objetos del entorno. CARLA también proporciona una gran variedad de mapas en distintos entornos de automoción (urbano, rural, autopista, etc).

CARLA ha sido utilizado en el desarrollo de varios artículos científicos [44, 45, 46, 51, 52, 53, 54, 55], además también está empujando los avances de conducción autónoma en su https://carlachallenge.org/, que se celebra anualmente, donde anima a desarrolladores de todo el mundo a participar en diferentes retos de Machine Learning y automoción.

4.1.2. Estandar de anotación: OpenLABEL

Uno de los principales problemas de los conjuntos de datos de automoción existentes es que cada uno usa su formato de anotación, base datos relacional (nuScenes, Lyft) datos anotados en ficheros planos de texto (KITTI). Para este proyecto se ha decidido utilizar un formato estándar de automoción con el fin de evitar estos problemas, este estándar es OpenLABEL[3].

OpenLABEL define el formato de anotación y los métodos de etiquetado para objetos y escenarios. El uso de un formato normalizado ayuda a reducir costes y ahorrar recursos en la creación, conversión y transferencia de datos anotados y etiquetados. OpenLABEL se representa en formato JSON y, por tanto, puede ser analizado fácilmente por herramientas y aplicaciones.

OpenLABEL especifica los diferentes métodos de etiquetado que pueden aplicarse a flujos de datos multisensor, por ejemplo, 2D *bounding boxes* para imagen. Con OpenLABEL, se proporcionan varios métodos de etiquetado que permiten a los usuarios etiquetar flujos de datos comunes, como imágenes o nubes de puntos.

La gran mayoría de vehículos cuentan con una gran variedad de sensores, OpenLABEL define la manera en la que se anotan las características de estos sensores, como por ejemplo la posición de estos respecto al vehículo principal o sus intrínsecos.

Además de sensores, OpenLABEL permite anotar una diversa cantidad de tipos de objetos, tales como, cuboides 2D y 3D en imágenes, segmentación de objetos en imágenes, cuboides 3D en nubes de puntos. No solo eso, también se puede etiquetar información adicional, como la velocidad, aceleración, atributos del objeto, etc.

Las escenas de automoción cuentan con varios *frames*, el vehículo y los diferentes elementos de la escena se mueven según la escena avanza. OpenLABEL define la manera en la que se debe anotar el cambio de posición de los elementos.

Todos estos datos se recolectan en un fichero JSON. El fichero JSON está formado por

[2] https://www.unrealengine.com/en-US
[3] https://www.asam.net/standards/detail/openlabel/

```
{
    "openlabel" : {
        "actions": {...},
        "contexts": {...},
        "coordinate_systems": {...},
        "events": {...},
        "frame_intervals": {...},
        "frames": {...},
        "metadata": {...},
        "objects": {...},
        "ontologies": {...},
        "relations": {...},
        "resources": {...},
        "streams": {...},
        "tags": {...}
    }
}
```

Listing 4.1: Estructura fichero OpenLABEL.

un objeto OpenLABEL que a su vez contiene distintos objetos con información sobre el escenario como se puede ver en la siguiente estructura 4.1.

A pesar de toda la información que se puede almacenar en estos ficheros, para este proyecto solo se han utilizado los siguientes campos de la estructura:

- **metadata:** este campo se ha utilizado únicamente para guardar la versión de OpenLABEL.
- **streams:** en este campo se guardan datos de los diferentes sensores: cámaras, LiDAR, etc. y sus propiedades, como la matriz de intrínsecos y la matriz de extrínsecos. Además para este proyecto se ha utilizado para guardar las características de los sensores de CARLA.
- **coordinate_systems:** en este campo se encuentran anotados los diferentes sistemas de coordenadas de la escena (mundo, vehículo, sensor, etc) y las relaciones entre estos. En este campo también se encuentran matrices de transformación, estas matrices se utilizan para indicar la posición relativa de un sistema de coordenadas respecto a otro. Todos los sistemas de coordenadas están basados en el ISO-8855 [34].
- **frames:** en este campo está la información de cada *frame* de la escena. Cada *frame* contiene datos sobre la transformación al vehículo desde el origen del mundo (si hubiese otro objeto en movimiento, también habría datos sobre este), la ruta a ficheros con información de ese *frame* (imágenes, nubes de puntos, etc) y las anotaciones de los objetos.

- **objects:** en este campo se encuentran todos los objetos pertenecientes a la escena e información sobre estos, como el sistema de coordenadas al que pertenecen, el tipo de objeto y en que *frames* aparece.
- **frame_intervals:** en este campo se encuentran los intervalos de *frames* de la escena

4.1.3. Video Content Description (VCD)

A pesar de que OpenLABEL es un estándar de automoción, la asociación creadora de este formato (ASAM) no ofrece ninguna herramienta para la creación e interpretación de estos ficheros. Es ahí donde entra VCD [84].

VCD es una librería desarrollada en Vicomtech para proporcionar un formato de anotación y un conjunto de herramientas que cubren la falta de un modelo de anotación estructurado para etiquetar conjuntos de datos multisensor complejos en diferentes dominios, como automoción, vigilancia, medicina y deportes. VCD permite la creación e interpretación de ficheros compatibles con el estándar OpenLABEL.

Esta herramienta se ha utilizado tanto en el proceso de generación de vehículos y sensores en CARLA como en la anotación de los objetos y actores de las escenas.

4.1.4. MMDetection3D

MMDetection3D [29] es una librería de python basada en PyTorch desarrollada por el grupo de investigación OpenMMLab[4], basada a su vez en otra de sus librerías, MMDetection [85]. MMdetection3D es una librería que se centra en la detección y el seguimiento de objetos en entornos 3D, lo que la hace especialmente valiosa en aplicaciones como la conducción autónoma, la robótica y la visión por computadora en general. La librería MMDetection3D se ha convertido en una herramienta fundamental para investigadores y profesionales que trabajan en problemas relacionados con la detección de objetos en escenarios 3D complejos.

Una característica notable de MMDetection3D es su arquitectura modular. Está diseñada de manera que los usuarios puedan combinar y personalizar diferentes componentes para crear soluciones a medida para sus problemas específicos. Esto facilita la experimentación con diferentes configuraciones y enfoques, lo que es esencial en la investigación y desarrollo de tecnologías de detección 3D.

4.2. Generación del conjunto de datos sintético

Un conjunto de datos bien construido y estructurado es esencial para obtener resultados precisos y confiables en diversas tareas. En este apartado, se explora el proceso realizado para la generación del conjunto de datos, así como la estructura y especificaciones del mismo.

4.2.1. Proceso de generación de los datos

La intención de crear el proceso de generación de datos sintéticos es que cualquiera con conocimientos básicos sobre las herramientas mencionadas pudiera utilizarlo para generar

[4] https://github.com/open-mmlab

sus propios datos. Es por ello que para la carga de los vehículos y sensores se ha utilizado OpenLABEL, donde se pueden definir los datos de varios sensores.

Posteriormente se ha escrito un código que a partir de estos ficheros carga los sensores y vehículos indicados, siempre y cuando existan en CARLA.

Se puede observar el *pipeline* de la generación del conjunto de datos en la Figura 4.1.

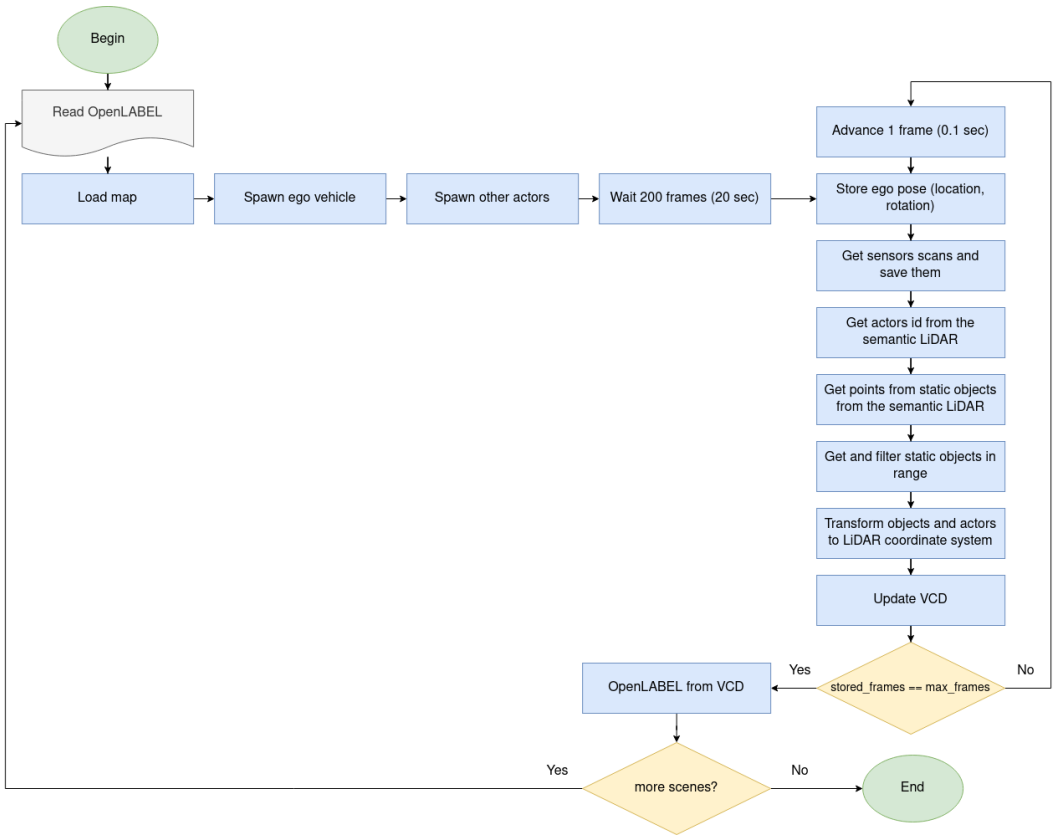


Figura 4.1: Diagrama de flujo de la generación de nuestro conjunto de datos sintético. El procedimiento explicado de manera resumida sería el siguiente: (1) cargar mapa de carla, vehículos de la escena y sensores, (2) cada 0.1 segundo obtener los datos del LiDAR sintético para generar una nube de puntos, (3) al mismo tiempo obtener los datos de los vehículos pertenecientes a la nube de puntos, (4) una vez han transcurrido los el número de *frames* indicados se almacenan los datos en un fichero OpenLABEL. Este algoritmo se ejecuta tantas veces como escenas se quieran obtener.

Primero se selecciona y carga el mapa de CARLA en el que va a suceder la escena, después, haciendo uso del código mencionado anteriormente, a partir de un fichero OpenLABEL se cargan el vehículo principal y sus sensores.

Una vez cargados el mapa, los sensores y el vehículo se generan varios actores en diferentes puntos de la escena, en este caso en particular por cada escena se cargaban: 30 coches, 10 camiones, 10 furgonetas, 10 buses, 20 bicicletas, 20 motos y 60 peatones. Sin embargo, este *pipeline* permite al usuario utilizar la distribución necesaria con el fin de generar el conjunto de datos más adecuado.

Tras la carga de diferentes actores, a todos ellos se les activa el modo autopiloto, que permite que se muevan de manera independiente respetando las normas de circulación. Antes de comenzar a generar y anotar datos hay un margen de 200 *frames* (20 segundos)

para que cada vehículo pueda alcanzar la velocidad adecuada para la posición en la que se encuentran.

Una vez comienza la generación de los datos, si el vehículo dispone de sensor LiDAR, se generarán nubes de puntos cada frame (0.1 segundos), cada punto cuenta con 5 campos: *(x, y, z, intensity, laser_id)*.

Para finalizar, se anotarán los datos en un fichero OpenLABEL, para ello se hace uso de un sensor único de CARLA, el LiDAR semántico. Este sensor aporta información adicional al LiDAR, como la id del actor con el que colisiona cada punto o el *tag* del objeto con el que colisiona. Gracias a este sensor se pueden reconocer los objetos que aparecen en escena. En caso de que haya un LiDAR en el fichero OpenLABEL original, este sensor se genera en la misma posición que el LiDAR original, con la misma rotación y propiedades, para poder hacer uso de este para la anotación de datos. El motivo por el que se utilizan dos sensores y no únicamente el sensor semántico es porque el sensor LiDAR obtiene la intensidad, mientras que el LiDAR semántico no.

Este *pipeline* permite generar tantas escenas como se desee y con tantos *frames* como se deseen. Sin embargo, a pesar de que el *pipeline* permita añadir todos los sensores disponibles en CARLA, en este trabajo solo nos hemos encargado de la automatización en la anotación de sensores de tipo LiDAR.

En la Figura 4.2, se pueden ver los resultados de este *pipeline*. Además, en la Figura 4.3, se pueden ver dos nubes de puntos en situaciones muy similares, siendo la imagen de la izquierda parte del conjunto de datos de nuScenes y la imagen de la derecha parte del conjunto de datos sintéticos.

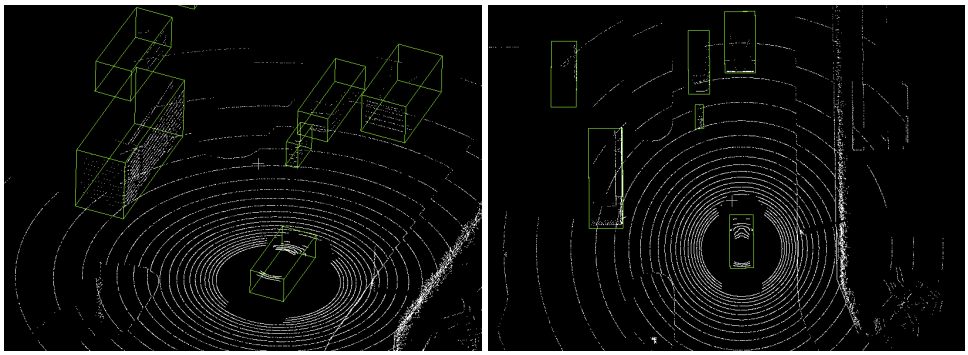
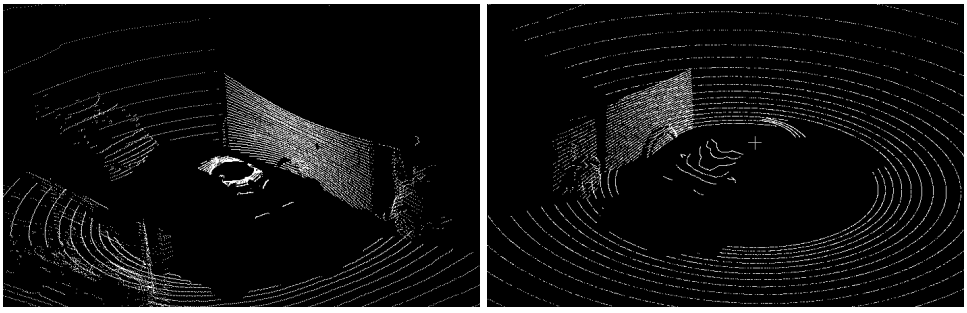


Figura 4.2: Nube de puntos sintética.

4.2.2. Estructura del conjunto de datos

El conjunto de datos consta de un total de 96 escenas con 100 *frames* anotados a una frecuencia de 10 Hz. Este conjunto de datos se puede dividir en dos partes, por un lado, las nubes de puntos y por otro las anotaciones de estas.

Las anotaciones están hechas en formato OpenLABEL. Cada escena está definida por un fichero OpenLABEL. Cada fichero OpenLABEL contiene anotaciones temporales para cada *frame* obtenido en nuestro entorno de simulación. Cada *frame* consta de los siguientes datos:



(a) Nube de puntos del conjunto de datos de nuScenes.

(b) Nube de puntos del conjunto de datos de sintético.

Figura 4.3: Comparación entre nubes de puntos del conjunto de datos de nuScenes y nube de puntos del conjunto de datos sintético.

- Ruta a la nube de puntos: a cada *frame* le corresponde una nube de puntos compuesta por 5 campos: *(x, y, z, intensity, laser_id)*, en esta sección del conjunto de datos se encuentra la ruta al fichero.

- Matriz de transformación: debido a que en cada *frame* de la escena el vehículo está en movimiento, es necesario tener una referencia de la posición del LiDAR respecto al sistema de coordenadas del mapa. Principalmente, porque las nubes de puntos y anotaciones de las mismas están en el sistema de coordenadas del LiDAR, y si se quieren acumular varias nubes de puntos, esta matriz es indispensable.

- Objetos: en cada *frame* están anotados los objetos que aparecen en el mismo, cada objeto cuenta con los siguientes datos:

  - *Bounding box* 3D: un vector que representa el cuboide que envuelve al objeto. En este caso la representación consta con 9 valores, tal y como se especifica en el estándar OpenLABEL. Los tres primeros parámetros corresponden a la posición del cuboide, cada parámetro corresponde a un eje de coordenadas en este orden: (x, y, z). La unidad de medida son metros. Los tres siguientes valores corresponden a la rotación del cuboide, en el siguiente orden: $(roll, pitch, yaw)$ y la unidad de medida son radianes. Los tres últimos parámetros corresponden al tamaño del cuboide, en el siguiente orden: *length* (eje x), *width* (eje y) y *height* (eje z). La unidad de medida son metros.
  - Vector de velocidad: un vector tridimensional que representa la velocidad del objeto en cada uno de los ejes (v_x, v_y, v_z). La unidad de cada uno de los parámetros es metros por segundo.
  - Vector de aceleración: un vector tridimensional que representa la aceleración del objeto en cada uno de los ejes (a_x, a_y, a_z). La unidad de cada uno de los parámetros es metros cuadrados por segundo.
  - Velocidad: float que representa la velocidad del objeto.
  - Aceleración: float que representa la aceleración del objeto.
  - Clase: la clase del objeto, en este conjunto de datos se han anotado 6 clases diferentes: *car, pedestrian, bicycle, motorbike, bus, truck.*

4.2.3. Especificaciones del conjunto de datos

El conjunto de datos generado comprende 96 escenas, cada una compuesta por 100 nubes de puntos a un *frame rate* de 0.1 segundos que han sido anotados de manera automática para generar un total de 9600 nubes de puntos. Cada nube de puntos ha sido etiquetada con información detallada sobre los objetos presentes en ella, habiendo un total de 6 tipos de objetos distintos: *car, pedestrian, bicycle, motorbike, bus, truck*. Todos los detalles cuantitativos del conjunto de datos se presentan en la Tabla 4.2, y se resumen gráficamente en la Figura 4.4. De ser necesaria otra distribución de datos, no hay una manera precisa de lograrla, la única manera que ofrece el pipeline para ello es cambiar el número de actores cargados al comenzar la simulación, si esto no es suficiente para obtener la distribución deseada, se puede realizar un postprocesado del conjunto de datos. Se debe destacar que el conjunto de datos completo tiene un tamaño de 11,3 GB.

Object Class	Number of annotations
car	48878
truck	26273
bus	13097
pedestrian	29373
motorcycle	20526
bicycle	18177
total	156324

Tabla 4.2: Número de clases del conjunto de datos.

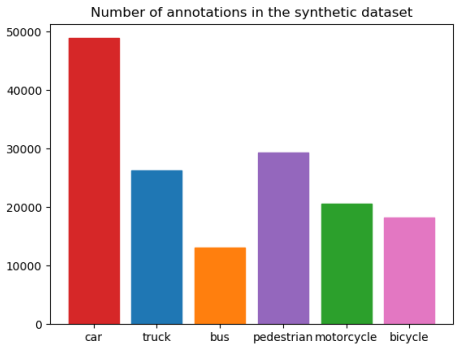


Figura 4.4: Número de anotaciones del conjunto de datos.

Para la generación de estos datos, se han utilizado cuatro mapas del entorno de simulación CARLA, esto para darle más variedad a las escenas del conjunto de datos. Los mapas elegidos para este conjunto de datos son Town3 4.5a, Town5 4.5b, Town6 4.5c y Town10 4.5d debido la variabilidad de las zonas, pues estos mapas tienen entornos rurales y urbanos. CARLA también permite variar el tiempo y el clima, sin embargo, para este estudio se han utilizado los valores por defecto, los cuales son tiempo medio día y clima soleado.

4.3. Métricas y configuración de los modelos

En el presente apartado se explorarán detalladamente las medidas de evaluación utilizadas para cuantificar el rendimiento de los modelos propuestos, así como la estrategia empleada para ajustar y optimizar los parámetros de dichos modelos. Este análisis profundo resulta esencial para garantizar la fiabilidad y eficacia de los resultados obtenidos en el contexto de nuestro estudio.

4.3.1. Métricas de evaluación

Para este estudio se han utilizado las métricas de evaluación habituales del estado del arte, también empleadas en el conjunto de datos de nuScenes, las cuales son:

- *Average Precision* (AP): la precisión mide la proporción de objetos clasificados correc-

(a) Mapa Town3 de CARLA.

(b) Mapa Town5 de CARLA.

(c) Mapa Town6 de CARLA.

(d) Mapa Town10 de CARLA.

Figura 4.5: Mapas del entorno de simulación CARLA.

tamente como positivos en relación a todos los objetos clasificados como positivos. Para calcular el AP, primero se ordenan los objetos detectados por su confianza de detección, de mayor a menor, y se calcula la precisión correspondiente a cada umbral de confianza. Se considera que un objeto ha sido bien clasificado si su distancia a la anotación *ground truth* es menor a una distancia específica. En este caso se realiza la media de la precisión calculada con cuatros valores para la distancia: $0.5, 1.0, 2.0, 4.0$ metros.

- *Average Transition* Error (ATE): esta métrica muestra el error al clasificar la posición de los objetos bien clasificados. Se calcula midiendo la distancia de la predicción con el *ground truth*. La unidad de medida son metros.

- *Average Scale Error* (ASE): esta métrica muestra el error al clasificar el tamaño de los objetos bien clasificados. Para ello se calcula el *Intersection over Union* (IoU), como si el cuboide que el modelo ha detectado estuviera perfectamente alineado con el *ground truth*. Se calcula dividiendo el volumen de la intersección del *bounding box* detectado con el *bounding box* del *ground truth* entre el volumen de la unión de los mismos. La unidad de medida son metros.

- *Average Orientation Error* (AOE): esta métrica muestra el error al clasificar la orientación de los objetos bien clasificados. Para ello se calcula la distancia más pequeña de una orientación a otra. Debido a que las predicciones de MMDetection3D de orientación varían desde $-\pi$ hasta π, esta métrica puede variar desde 0 hasta π. Además, solo se tiene en cuenta la rotación en yaw. La clase *traffic_cone* no tiene orientación, por lo que el resultado en esta métrica será nan (*Not a number*).

- *Average Velocity Error* (AVE): esta métrica muestra el error al clasificar la velocidad de los objetos bien clasificados. Se calcula como la diferencia de la velocidad *ground truth* del objeto y la velocidad de la predicción en los ejes x e y. La unidad es metros por segundo.

- *Average Attribute Error* (AAE): esta métrica muestra el error al clasificar los atributos de los objetos bien clasificados. Es el porcentaje de los atributos mal clasificados de los objetos. Clases como *traffic_cone o barrier* no tienen atributo, por lo que el resultado en estos casos es nan (*Not a number*).

4.3.2. Proceso de entrenamiento

Para el proceso de entrenamiento se han utilizado las funciones de la librería MMDetection3D [29]. Esta librería ofrece tanto ficheros de configuración de los modelos como pesos para estos ficheros. Estos ficheros de configuración varían en función del modelo y del conjunto de datos para el que esté preparado el modelo. Los pesos que ofrece MMDetection3D son los pesos de los modelos entrenados previamente. Para este proyecto se ha decidido entrenar los modelos con el conjunto de datos de nuScenes, ya que de los formatos de conjunto de datos disponibles en la librería solamente Waymo y nuScenes tenían las clases de interés para este proyecto, sin embargo, Waymo no permite el uso del conjunto de datos con fines comerciales y los resultados de investigación de este proyecto han servido como base de investigación en un proyecto industrial.

Debido a la magnitud del conjunto de datos y la lentitud de los modelos, el proceso de entrenamiento puede llegar a tardar hasta 20 días, es por ello que la posibilidad de variar los hiperparámetros de los modelos con el fin de obtener mejores resultados ha sido limitado. En su lugar, se han replicado los valores del estado del arte. Para el entrenamiento de todos los modelos se han usado dos GPUs Nvidia a40, por limitaciones de memoria todos los modelos se han tenido que entrenar con un *batch size* igual a 2.

El conjunto de datos de nuScenes está dividido en varios sets. Por un lado, la versión mini que dispone de 10 escenas, este set se ha utilizado para realizar pruebas y poner en marcha la librería MMDetection3D. Por otro lado, está la versión trainval con 850 escenas, que a su vez consta de dos sets, la versión de *train* con 700 escenas, utilizada para entrenar los modelos, y la versión de *validation* con 150 escenas, para evaluar los modelos. Por último, el conjunto de datos de nuScenes consta con una versión de test con 150 escenas, sin embargo, esta no ha sido utilizada, ya que no consta de anotaciones. Cada escena consta de aproximadamente 20 segundos de grabación, el LiDAR utilizado para la recolección de los datos funciona a 10 Hz, lo que significa que por cada escena hay un total de 200 nubes de puntos aproximadamente.

Los modelos son optimizados para el set de validación, es decir que se ha seleccionado como criterio de parada del entrenamiento los resultados de la evaluación del modelo con este set, el entrenamiento se detuvo después de dos evaluaciones seguidas sin mejora. Los pesos se inicializan con valores aleatorios. Los aumentos que se utilizan durante el entrenamiento son *random flip* que consiste en voltear la nube de puntos respecto el eje x o y y *global rotation* que consiste en rotar la nube de puntos respecto el eje z, también se hacen rotaciones de muy pocos grados en los otros dos ejes. Es necesario mencionar que las nubes de puntos son recortadas en base del rango del modelo, en el caso de los modelos utilizados PointPillars y SSN tienen un rango de 50 metros, mientras que CenterPoint y TransFusion tienen un rango de 54 metros. El optimizador utilizado para los entrenamientos de los modelos es *Adam Optimization* [86], el optimizador Adam es un algoritmo de optimización que combina las ventajas de los métodos de descenso de gradiente estocástico con momento y adaptación del *learning rate*. Los modelos se evaluaban cada 4 épocas, pero se guardaban

los pesos de cada época. A pesar de que en la evaluación se obtuvieran todas las métricas, la decisión de cuál de los modelos era el mejor se tomaba en base al mAP. Al finalizar los entrenamientos se evaluaban los modelos con los pesos de épocas cercanas al mejor resultado visto en el entrenamiento, tras eso se seleccionaban los pesos que aportaran mejor resultado. Debido a este criterio de parada, en algunos casos se ha entrenado el modelo durante más épocas que las indicadas en el estado del arte.

CAPÍTULO 5

Resultados

En la actualidad, la detección de objetos en nubes de puntos se ha convertido en un tema de investigación relevante y en constante crecimiento. La proliferación de modelos de aprendizaje profundo en los últimos años ha llevado a la necesidad de identificar y estudiar diversas opciones para determinar las carencias y ventajas de los distintos modelos entre sí.

En este capítulo se lleva a cabo un análisis exhaustivo de diferentes modelos disponibles en la librería MMDetection3D, centrándose principalmente en aquellos que han sido diseñados específicamente para trabajar con el conjunto de datos nuScenes. A pesar de que en el respectivo artículo de cada uno de los modelos ya se realiza un estudio sobre los mismos, estos estudios no ofrecen la información necesaria respecto a los modelos, como la diferencia de tiempo y precisión al variar el número de nubes acumuladas. Entre los modelos que se han considerado se incluyen PointPillars [22], Shape Signature Network [23], CenterPoint [24], y también se ha adaptado un modelo externo a la librería, conocido como TransFusion[25]. Se ha decidido usar el conjunto de datos nuScenes debido a que las nubes de puntos de este conjunto de datos son muy similares a las que se pueden generar con el vehículo autónomo de Vicomtech como se puede ver en la Figura 5.1. Esto se debe a que ambos LiDARes tienen un rango similar y el mismo número de haces, además estas nubes abarcan los 360 grados alrededor del vehículo a diferencia de las nubes de otros conjuntos de datos.

El objetivo de este estudio es analizar las capacidades y limitaciones de cada modelo: tiempo de procesamiento, precisión y generalización y, en última instancia, identificar cuál de ellos es el más adecuado para abordar los diferentes retos de detección de objetos en nubes de puntos. Con este fin, se presentan los resultados de diversos experimentos y pruebas llevadas a cabo con cada uno de los modelos estudiados.

5.1. Protocolo de experimentación y análisis de los resultados

Para llevar a cabo esta investigación, de las diez clases disponibles (*car*, *truck*, *bus*, *trailer*, *construction vehicle*, *pedestrian*, *motorcycle*, *bicycle*, *traffic cone*, *barrier*) en el conjunto de

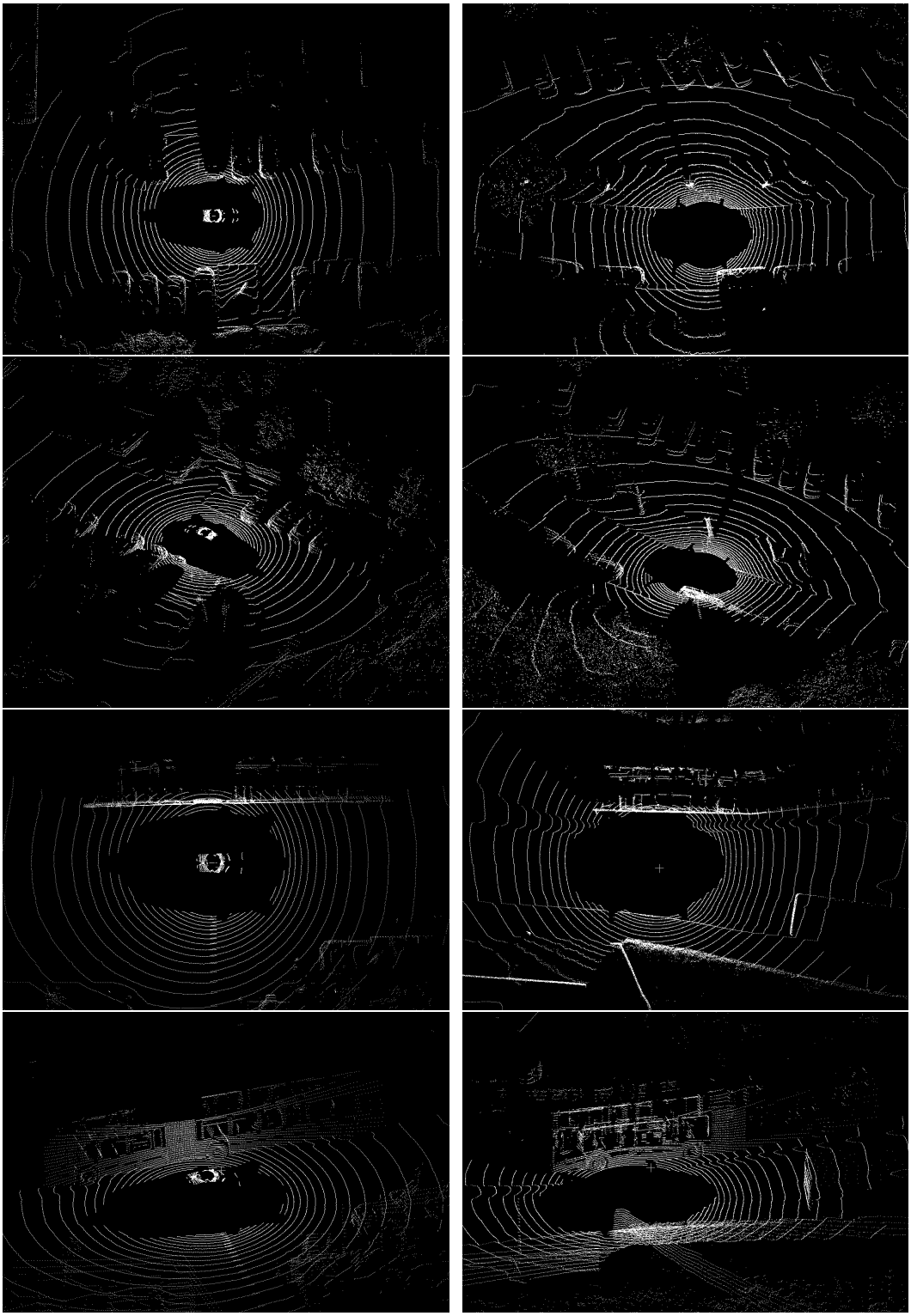


Figura 5.1: Comparación entre las nubes de puntos del conjunto de datos de nuScenes (columna izquierda), y nubes de puntos de un *recording* realizado con el vehículo autónomo de Vicomtech (columna derecha).

datos nuScenes se han elegido solamente las seis clases más relevantes, aprovechando esto se ha investigado si reducir el número de clases aumenta la precisión del modelo.

Todos los modelos permiten acumular nubes de puntos capturados en los instantes previos a la detección para incrementar el número de puntos usados como *input* y así mejorar los resultados de las detecciones. Pese a que este proceso mejore la precisión en las detecciones, supone un coste computacional adicional tanto en preprocesamiento como en inferencia. Por este motivo, se ha estudiado el tiempo de inferencia y precisión de cada modelo tanto con acumulación como sin acumulación de nubes de puntos. En la Figura **??** y la Tabla 5.1 se puede observar la latencia de cada modelo con diferentes acumulaciones de nubes de puntos. Estos datos se han obtenido utilizando el *High Performance Computing* (HPC) de Vicomtech, este ordenador tiene las siguientes especificaciones: seis procesadores Intel Xeon Gold 6238R[1] con 56 *threads* cada uno, 378 GiB de RAM, ocho GPUs Nvidia Tesla T4[2] de 16 GB cada una y ocho GPUs Nvidia Tesla A40[3] de 48 GB cada una, en el caso de estas pruebas se ha utilizado una GPU Nvidia Tesla A40. Se daba por hecho que el tiempo de inferencia de los modelos sería distinto en el ordenador del coche autónomo y en el ordenador en el que se hicieron las pruebas, sin embargo, los tiempos obtenidos con esta GPU son los que se utilizan como referencia para indicar si un modelo funciona en tiempo real o no, como referencia para entrenar modelos funcionales en tiempo real.

Number of sweeps	PointPillars	SSN	CenterPoint	TransFusion
0	14.3	**19.3**	10.3	12.2
1	11.6	**15.0**	8.9	10.1
2	10.0	**13.0**	7.9	8.7
3	8.2	**8.8**	7.2	7.4
4	7.4	**7.7**	6.6	6.6
5	6.5	**6.6**	6.1	5.9
6	5.7	**6.1**	5.8	5.4
7	5.2	5.3	**5.5**	4.9
8	4.7	4.9	**5.1**	4.4
9	4.4	4.5	**4.7**	4.0
10	**4.3**	4.1	**4.3**	3.6

Tabla 5.1: Frecuencia de inferencia (nubes de puntos/segundo) de los modelos con diferentes acumulaciones de nubes de puntos (GPU: Nvidia a40).

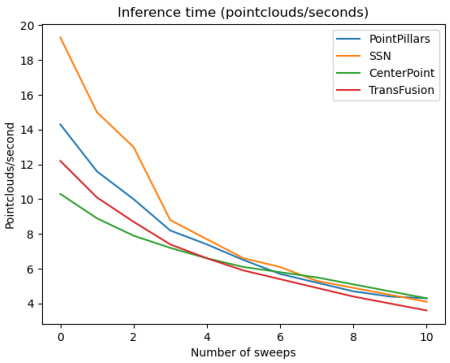


Figura 5.2: Frecuencia de inferencia (nubes de puntos/segundo) de los modelos con diferentes acumulaciones de nubes de puntos (GPU: Nvidia a40).

En la Tabla 5.1 se pueden ver distintos tiempos en *frames* por segundo, que varían depende del número de *sweeps*. Los *sweeps* es el término que utiliza nuScenes para el número de nubes de puntos adicionales se añaden a la nube principal. En la tabla se puede observar que SSN es el modelo más rápido, frente a CenterPoint el más lento. A medida que se van acumulando nubes de puntos, el tiempo que tardan los diferentes modelos se asemeja más. Esto podría deberse a que el tiempo del que supone la acumulación de nubes de puntos junto a otras operaciones de preprocesado que se realizan crezca más rápido que el tiempo de inferencia de los modelos, provocando que este se vuelva insignificante en comparación al tiempo de preprocesado. Se han medido el tiempo que se tarda en cargar y acumular las nubes de puntos, estos se pueden ver en la Figura 5.3 y en la Tabla 5.2. A pesar de que la

[1] https://www.intel.la/content/www/xl/es/products/sku/199345/intel-xeon-gold-6238r-processor-38-5m-cache-2-20-ghz/specifications.html
[2] https://www.nvidia.com/es-es/data-center/tesla-t4/
[3] https://www.nvidia.com/es-es/data-center/a40/

gráfica muestre que la frecuencia disminuya de manera exponencial, en la tabla se puede observar como la frecuencia con 10 sweeps es de 30 nubes de puntos por segundo, que es una frecuencia muy superior a lo que tarda el modelo en realizar inferencia con 0 *sweeps*. Es decir, el tiempo de carga y acumulación de nubes no es relevante frente al tiempo que tarda el modelo en hacer inferencia. Por ello se puede descartar que el modelo se vuelva tan lento debido al tiempo de carga, es muy probable que según aumentan los sweeps el modelo dedique más tiempo al preprocesado. Parte del preprocesado es la voxelización de los puntos, la voxelización es el proceso que transforma las nubes de puntos en una representación de tamaño fijo que se utiliza como input para el modelo, como el tamaño del input no varía, se puede afirmar que el modelo se vuelve más lento debido a este proceso.

Number of sweeps	Time (seconds)	Frecuency (Pointclouds / second)
0	0.00201065	497.35
1	0.00508516	196.65
2	0.00782336	127.82
3	0.01106336	90.39
4	0.01457531	68.61
5	0.01705160	58.65
6	0.02009967	49.75
7	0.02371208	42.17
8	0.02780213	35.97
9	0.02948592	33.91
10	0.03284957	30.44

Tabla 5.2: Frecuencia de carga en la acumulación de nubes de puntos.

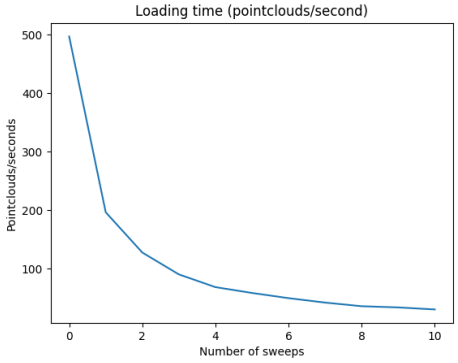


Figura 5.3: Frecuencia de carga en la acumulación de nubes de puntos.

Tras la medición del tiempo de inferencia, se ha entrenado los modelos con una acumulación reducida de nubes de puntos para ver su desempeño en tiempo real y ver cuanto se reduce la precisión de estos. Pero para ello primero es necesario definir la velocidad necesaria de un modelo para que este funcione en tiempo real. Puesto que el sensor LiDAR Velodyne funciona a 10 Hz, es decir, 10 ciclos por segundo, es necesario que el modelo de detección funcione como mínimo a la misma velocidad. Por lo que el modelo deberá ser capaz de procesar como mínimo 10 *frames* por segundo. Según los datos de la tabla, todos los modelos se pueden usar en tiempo real sin acumulación de nubes de puntos. PointPillars y SSN son capaces de funcionar a tiempo real acumulando hasta 2 nubes de puntos, TransFusion con 1 y CenterPoint no es capaz de llegar a 10Hz aplicando acumulación. Dado que los últimos dos modelos de la tabla son más potentes, pero no pueden acceder a tanta información como los dos primeros en tiempo real, será necesario realizar un estudio de la precisión de cada modelo con las restricciones de tiempo de inferencia para su uso en tiempo real.

5.1.1. Resultados con el conjunto de datos nuScenes

En primer lugar, en la Figura 5.4 se pueden apreciar detecciones del conjunto de datos de nuScenes [7] con el modelo TransFusion [25]. Estas imágenes muestran las capacidades de estos modelos.

A continuación se verán varias tablas 5.3, 5.4, 5.5, 5.6. Cada una de las tablas compara los resultados obtenidos usando un mismo modelo, pero variando el número de clases y/o el número de nubes de puntos acumuladas. El objetivo de estas tablas es analizar el desempeño de estas variaciones del modelo para diferentes clases. Debido a toda la información que

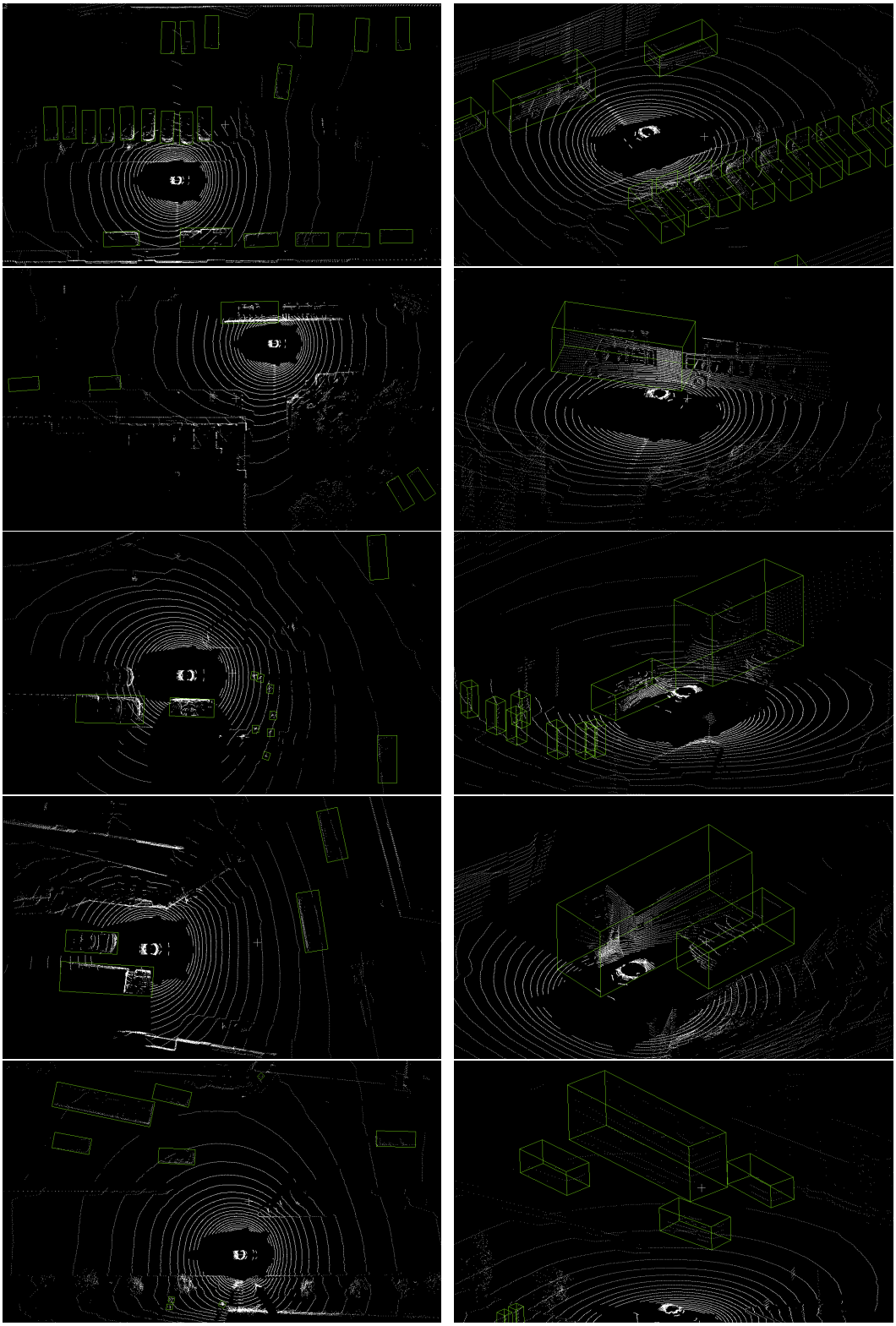


Figura 5.4: Detecciones del modelo TransFusion [25] en el conjunto de datos de nuScenes [7].

aportan estás tablas, se ha recopilado la métrica más importante para este estudio (*Average Precision*) en una sola tabla 5.7. También se han agrupado los resultados de los modelos que son capaces de funcionar en tiempo real (más de 10 *frames* por segundo) en la tabla 5.8.

PointPillars

Modelo Preentrenado						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.799	0.217	0.157	0.149	0.256	0.198
truck	0.357	**0.460**	0.219	0.268	**0.250**	**0.210**
bus	0.428	0.553	0.211	0.204	0.496	0.246
trailer	**0.261**	**0.789**	**0.231**	**0.527**	**0.191**	**0.159**
construction_vehicle	**0.055**	**0.835**	**0.469**	**1.365**	**0.133**	**0.346**
pedestrian	0.717	0.155	0.288	0.366	0.248	0.070
motorcycle	**0.394**	**0.260**	0.260	0.618	**0.506**	0.226
bicycle	**0.106**	0.294	0.295	0.860	0.273	**0.026**
traffic_cone	**0.334**	**0.247**	**0.368**	nan	nan	nan
barrier	**0.520**	**0.405**	**0.305**	**0.050**	nan	nan
Modelo Reentrenado 10 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	**0.803**	**0.209**	**0.153**	**0.134**	**0.245**	**0.197**
truck	**0.362**	**0.460**	**0.215**	**0.218**	**0.250**	0.223
bus	0.426	**0.486**	0.208	0.211	**0.488**	**0.204**
pedestrian	**0.742**	**0.154**	**0.279**	**0.349**	**0.238**	**0.066**
motorcycle	0.332	0.269	0.251	0.544	0.558	0.234
bicycle	0.094	**0.282**	**0.274**	**0.761**	**0.267**	0.032
Modelo Reentrenado 2 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.781	0.220	0.157	0.161	0.310	0.199
truck	0.332	0.477	0.216	0.246	0.315	0.211
bus	**0.440**	0.525	**0.202**	**0.191**	0.586	0.227
pedestrian	0.694	0.160	0.283	0.384	0.283	0.076
motorcycle	0.269	0.284	**0.249**	**0.532**	0.803	**0.224**
bicycle	0.088	0.301	0.285	0.845	0.341	0.035

Tabla 5.3: Análisis de los resultados del modelo PointPillars [22].

Shape Signature Network

Modelo Preentrenado						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	**0.873**	0.209	0.167	0.220	**0.127**	**0.089**
truck	**0.624**	**0.255**	**0.163**	0.187	**0.102**	**0.036**
bus	**0.943**	**0.212**	**0.086**	**0.055**	0.737	**0.100**
trailer	0.000	0.000	0.000	0.000	0.000	0.000
construction_vehicle	0.000	0.000	0.000	0.000	0.000	0.000
pedestrian	**0.779**	0.167	**0.257**	**0.342**	**0.248**	0.112
motorcycle	0.387	0.251	0.291	1.397	**0.057**	**0.000**
bicycle	**0.371**	**0.230**	**0.157**	**0.657**	0.248	**0.000**
traffic_cone	0.000	**0.471**	**0.514**	nan	nan	nan
barrier	0.000	0.000	0.000	0.000	nan	nan
Modelo Reentrenado 10 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.816	**0.203**	**0.153**	**0.143**	0.260	0.190
truck	0.464	0.390	0.187	**0.139**	0.230	0.234
bus	0.561	0.442	0.185	0.152	**0.486**	0.229
pedestrian	0.665	0.167	0.277	0.412	0.263	0.094
motorcycle	**0.427**	**0.237**	0.239	0.524	0.549	0.152
bicycle	0.158	0.240	0.248	0.672	**0.215**	0.021
Modelo Reentrenado 2 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.792	0.207	0.154	0.160	0.323	0.190
truck	0.440	0.398	0.195	0.160	0.281	0.232
bus	0.540	0.453	0.183	0.149	0.598	0.218
pedestrian	0.657	**0.164**	0.277	0.419	0.282	**0.088**
motorcycle	0.373	0.238	**0.234**	**0.483**	0.632	0.244
bicycle	0.104	0.243	0.249	0.945	0.268	0.012

Tabla 5.4: Análisis de los resultados del modelo Shape Signature Network [23].

CenterPoint

Modelo Preentrenado						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	**0.851**	**0.180**	**0.152**	**0.109**	**0.279**	**0.191**
truck	**0.539**	0.336	**0.182**	**0.083**	**0.254**	0.239
bus	**0.665**	0.329	0.188	**0.051**	**0.462**	**0.274**
trailer	**0.332**	**0.538**	**0.202**	**0.496**	**0.210**	**0.111**
construction_vehicle	**0.160**	**0.635**	**0.413**	**0.970**	**0.124**	**0.295**
pedestrian	**0.845**	**0.145**	**0.275**	**0.386**	**0.213**	0.092
motorcycle	**0.559**	**0.184**	0.241	**0.270**	**0.460**	**0.259**
bicycle	**0.373**	0.156	**0.270**	0.507	**0.174**	**0.013**
traffic_cone	**0.681**	**0.143**	0.329	nan	nan	nan
barrier	**0.681**	**0.201**	0.278	**0.066**	nan	nan
Modelo Reentrenado 10 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.821	0.186	0.159	0.169	0.395	0.196
truck	0.484	**0.320**	0.190	0.121	0.342	**0.232**
bus	0.627	**0.322**	0.185	0.136	0.652	0.292
pedestrian	0.748	0.146	0.286	0.397	0.235	**0.089**
motorcycle	0.433	**0.184**	0.239	0.275	0.555	0.266
bicycle	0.302	**0.153**	0.274	**0.442**	0.254	0.018
Modelo Reentrenado 0 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.741	0.198	0.160	0.252	1.267	0.248
truck	0.399	0.343	0.208	0.184	1.331	0.345
bus	0.570	0.280	**0.169**	0.184	2.202	0.404
pedestrian	0.654	0.150	0.285	1.051	0.811	0.361
motorcycle	0.317	0.200	**0.235**	0.531	2.784	0.115
bicycle	0.180	0.197	**0.266**	0.923	0.582	0.016

Tabla 5.5: Análisis de los resultados del modelo CenterPoint [24].

TransFusion

Modelo Reentrenado 10 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	**0.871**	**0.169**	**0.151**	**0.083**	**0.241**	**0.191**
truck	0.387	0.398	0.225	**0.101**	**0.397**	**0.308**
bus	**0.734**	0.312	**0.182**	**0.047**	**0.351**	**0.258**
pedestrian	**0.866**	**0.130**	**0.275**	**0.336**	**0.197**	**0.081**
motorcycle	**0.620**	**0.179**	**0.233**	**0.204**	**0.346**	**0.259**
bicycle	**0.294**	0.213	0.293	**0.550**	**0.175**	0.043
Modelo Reentrenado 0 *sweeps*						
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AAE \downarrow$
car	0.806	0.184	0.154	0.141	0.880	0.209
truck	**0.426**	**0.388**	**0.208**	0.131	1.032	0.339
bus	0.696	**0.310**	0.184	0.085	1.469	0.329
pedestrian	0.783	0.134	0.293	0.922	0.727	0.241
motorcycle	0.531	0.196	0.235	0.366	1.691	0.139
bicycle	0.285	**0.198**	**0.257**	0.605	0.486	**0.012**

Tabla 5.6: Análisis de los resultados del modelo TransFusion [25].

Average Precision de los modelos

Modelo	PointPillars [22]			SSN [23]			CenterPoint [24]			TransFusion [25]	
Nº *sweeps*	10 *sweeps*		2 *sweeps*	10 *sweeps*		2 *sweeps*	10 *sweeps*		0 *sweeps*	10 *sweeps*	0 *sweeps*
Nº clases	10 cls	6 cls	6 cls	10 cls	6 cls	6 cls	10 cls	6 cls	6 cls	6 cls	6 cls
car	0.799	0.803	0.781	**0.873**	0.816	0.792	0.851	0.826	0.741	0.871	0.806
truck	0.357	0.362	0.332	**0.624**	0.464	0.440	0.539	0.497	0.399	0.387	0.426
bus	0.428	0.426	0.440	**0.943**	0.561	0.540	0.665	0.621	0.570	0.734	0.696
trailer	0.261	-	-	0.000	-	-	**0.332**	-	-	-	-
construction_vehicle	0.055	-	-	0.000	-	-	**0.160**	-	-	-	-
pedestrian	0.717	0.742	0.694	0.779	0.665	0.657	0.845	0.741	0.654	**0.866**	0.783
motorcycle	0.394	0.332	0.269	0.387	0.427	0.373	0.559	0.398	0.317	**0.620**	0.531
bicycle	0.106	0.094	0.088	0.371	0.158	0.104	**0.373**	0.253	0.180	0.294	0.285
traffic_cone	0.334	-	-	0.000	-	-	**0.681**	-	-	-	-
barrier	0.520	-	-	0.000	-	-	**0.681**	-	-	-	-
Mean Average Precision	0.397	0.460	0.434	0.398	0.515	0.484	0.568	0.556	0.477	**0.629**	0.587

Tabla 5.7: Comparación del *Average Precision* de los diferentes modelos.

Average Precision de los modelos funcionales en tiempo real

Object Class	PointPillars [22]	SSN [23]	CenterPoint [24]	TransFusion [25]
car	0.781	0.792	0.741	**0.806**
truck	0.332	**0.440**	0.399	0.426
bus	0.440	0.540	0.570	**0.696**
pedestrian	0.694	0.657	0.654	**0.783**
motorcycle	0.269	0.373	0.317	**0.531**
bicycle	0.088	0.104	0.180	**0.285**
Mean Average Precision	0.434	0.484	0.477	**0.587**

Tabla 5.8: Comparación del *Average Precision* de los diferentes modelos funcionales en tiempo real.

Una de las primeras cosas que se pueden apreciar es que, a pesar de haber entrenado los modelos con el fin de que detecten menos clases, si bien la precisión general aumenta, la precisión individual de las clases se reduce en la mayoría de casos. Analizando las tablas, se

puede deducir que es probable que la precisión general aumente por la dificultad que supone la detección de las clases *trailer* y *construction_vehicle*, que son excluidas al reentrenar el modelo. Detectar estas clases puede ser complejo para los modelos por varios motivos como la similitud entre las clases *trailer* y *truck* o lo diferentes que son los objetos anotados con la clase *constuction_vehicle* entre sí. Otro factor que afecta negativamente a la precisión de estas clases es el número de anotaciones en el conjunto de datos, puesto que es pequeño en comparación a otras clases.

Otro aspecto que tienen en común todos los modelos es que en las clases de menores dimensiones como *bicycle* y *motorcycle* son las más difíciles de detectar, lo que resulta coherente al pensar la cantidad de puntos escasa que pueden representar estos elementos en un entorno real. Sin embargo, esto no pasa con la clase de *pedestrian* cuya dimensión es incluso menor que la de las clases mencionadas anteriormente. Esto probablemente se deba a dos motivos, por un lado, el número de anotaciones de los *pedestrians* en el conjunto de datos con el que los modelos han sido entrenados es mayor al número de anotaciones de *bicycles* o *motorcycles* como se puede ver en la Tabla 5.9 obtenida de la página oficial de nuScenes [4]. Por el otro lado, la similitud entre estas dos clases en nubes de puntos, puede ser un factor importante en la baja precisión del modelo en estas clases.

Object Class	Number of samples
car	493.322
truck	88.519
bus	16.321
trailer	24.860
construction_vehicle	14.671
pedestrian	222.164
motorcycle	12.617
bicycle	11.859
traffic_cone	97.959
barrier	152.087
Total	1,134,379

Tabla 5.9: Número de anotaciones en el conjunto de datos de nuScenes [7].

Si bien todos los modelos obtienen un buen desempeño, hay dos modelos que destacan sobre los demás, SSN y TransFusion. Es cierto que CenterPoint obtiene los mejores resultados en las clases *trailer*, *construction_vehicle*, *traffic_cone* y *barrier*, es posible que este modelo sea el más útil en alguna situación, sin embargo, para este estudio estás clases no son relevantes. TransFusion es superior a todos los demás, tanto con acumulación como sin acumulación, especialmente destaca en la precisión de la clase *motorcycle*, mientras que el resto de modelos tienen muchos problemas para detectar esta clase, este modelo obtiene una precisión bastante decente. También es necesario mencionar que TransFusion es el modelo que mejores resultados tiene al detectar *pedestrians*, es importante destacar esto debido a que esta es la clase más vulnerable en la carretera y, por tanto, hay especial interés en detectarla en el campo de la conducción autónoma, además es necesario mencionar que este modelo puede recibir imágenes como *input* adicional y no se está haciendo uso de esta ventaja. Si bien SSN no alcanza los resultados de TransFusion, obtiene resultados bastante

[4] https://www.nuscenes.org/nuscenes

buenos, más teniendo en cuenta que es el modelo que más destaca en velocidad, es por ello que podría ser una opción a tener en cuenta a la hora de implementar un modelo de detección de objetos en nubes de puntos en un vehículo autónomo. Posteriormente, estos modelos han sido probados con nubes de puntos de un *recording* del vehículo autónomo de Vicomtech, estos resultados no se pueden comparar con un *ground truth*, ya que no están anotados, en la Figura 5.5 se pueden apreciar las detecciones.

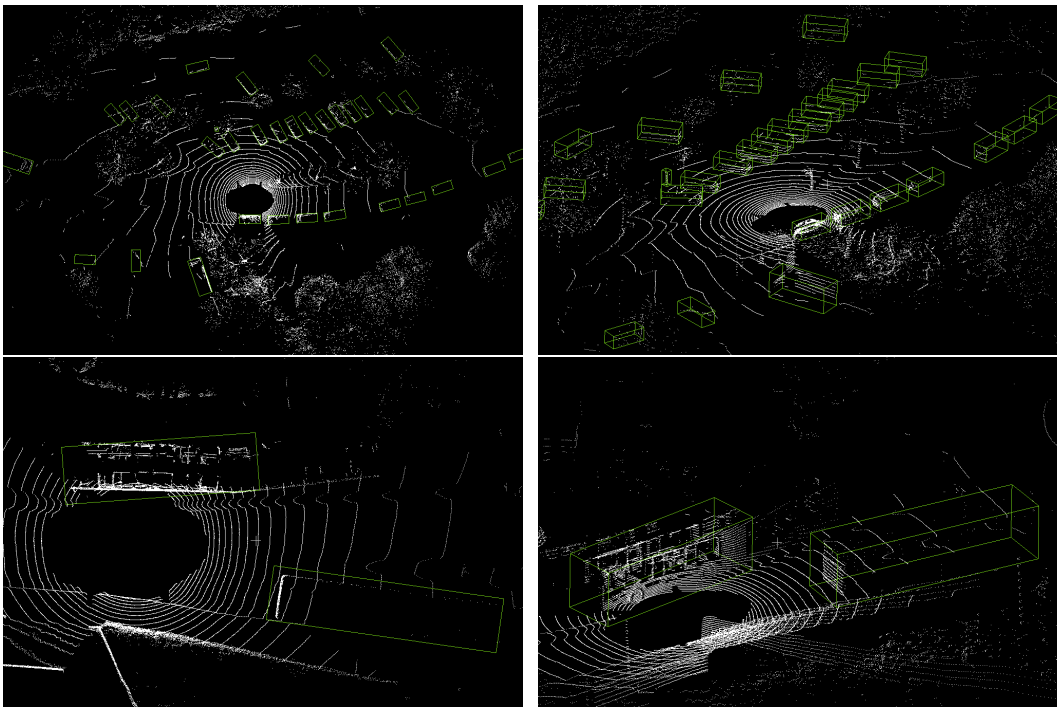


Figura 5.5: Detecciones del modelo TransFuison [25] en nubes de puntos obtenidas con el vehículo autónomo de Vicomtech. Cada fila contiene una nube de puntos vista desde distintos angulos.

5.1.2. Resultados con el conjunto de datos sintético

En esta sección se han evaluado algunos de los modelos utilizados en la sección anterior 5.1.1 con el conjunto de datos generado en el capítulo anterior 4.2. Al evaluar varios modelos, se puede observar cuál es más robusto frente al cambio de dominio, esto no se medirá viendo cuál obtiene mejor precisión, sino viendo la precisión de cuál varía menos. Además, es posible que los resultados de los modelos aporten información acerca de la diferencia entre ambos dominios. Cuando hay dos dominios, existe una brecha de dominio (*domain gap*) que potencialmente deteriora la precisión del modelo.

Las nubes de puntos sintéticas tienen diferencias que podrían perjudicar a los modelos, las diferencias detectadas son las siguientes:

- Todos los láseres que utiliza el LiDAR siempre obtienen información. Esto con un LiDAR real no sucede, ya que bien sea por la superficie o bien sea el material con el que colisiona el láser o bien sea por el clima, muchos de los puntos se pierden. Sin embargo, esto no sucede en un entorno simulado.

- Los puntos forman círculos perfectos, una vez más este fenómeno se debe a que la nube se ha obtenido en un entorno simulado, los mismos motivos que producen que se pierdan puntos también produce que los puntos obtenidos no estén perfectamente alineados.

- Por último, otro factor que produce que estos dominios sean diferentes es el valor de la intensidad de los puntos, debido a la manera en la que CARLA calcula la intensidad, los valores de esta no son realistas, CARLA calcula la intensidad basándose en la distancia de los puntos, lo que es muy alejado de la realidad en comparación con un LiDAR real, que varía el valor de la intensidad dependiendo de la reflectividad del material con el que colisiona y su ángulo de incidencia.

Estas diferencias se pueden ver en la Figura 5.6 en la que se comparan una nube de puntos del conjunto de datos de nuScenes (izquierda) con una nube de puntos del conjunto de datos sintético (derecha), el color de los puntos varía en base a la intensidad de los mismos.

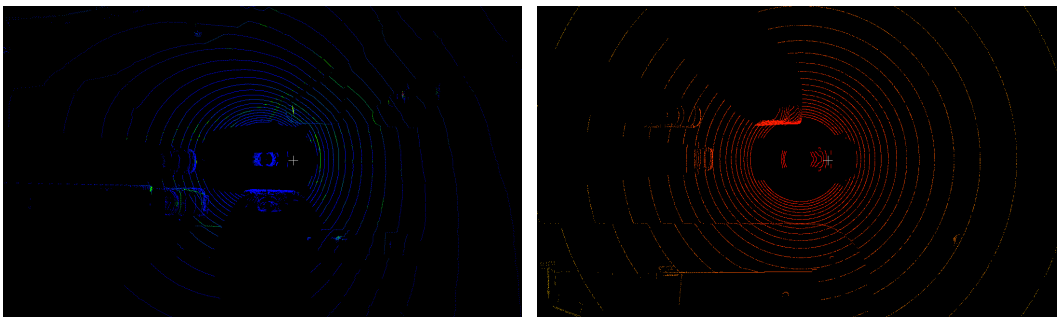


Figura 5.6: Comparación entre nube de puntos del conjunto de datos de nuScenes [7] (izquierda) y del conjunto de datos sintético (derecha).

En este proyecto se ha propuesto realizar un preprocesado a estas nubes para tratar de reducir las diferencias entre ambos dominios, se han evaluado los modelos con nubes con y sin preprocesado, para poder comparar si las diferencias realmente afectan negativamente a los modelos. Para solucionar el problema de la pérdida de puntos en el entorno real se eliminan puntos de manera aleatoria de la nube, en este caso se ha decidido eliminar el $10\,\%$ de los puntos. Para solucionar el problema de los puntos perfectos se suman a los puntos valores aleatorios dentro de un rango a cada punto para de esta manera trasladarlos, en este caso el rango decidido es $(-0.05, 0.05)$. Finalmente, para lidiar con el problema del valor de la intensidad se han cambiado los valores de la intensidad por un mismo valor, en este caso el valor elegido ha sido 1. Los valores para la reducción y traslación de los puntos han sido seleccionados tratando de obtener nubes lo más cercanas a la realidad. En cuanto al valor de la intensidad, se decidió que tomara el valor 1 al ser el más común en las nubes de puntos reales. En la Figura 5.7, se puede ver una comparación del cambio de la nube antes y después de aplicarle el preprocesado. Realizando este proyecto solamente se encontró un artículo que simula una intensidad realista [87], sin embargo, este artículo no ofrece el código y replicarlo no estaba en el *scope* del proyecto, además de que se requieren imágenes y el conjunto de datos generado no consta de imágenes, es por ello que se ha optado por asignar un valor constante.

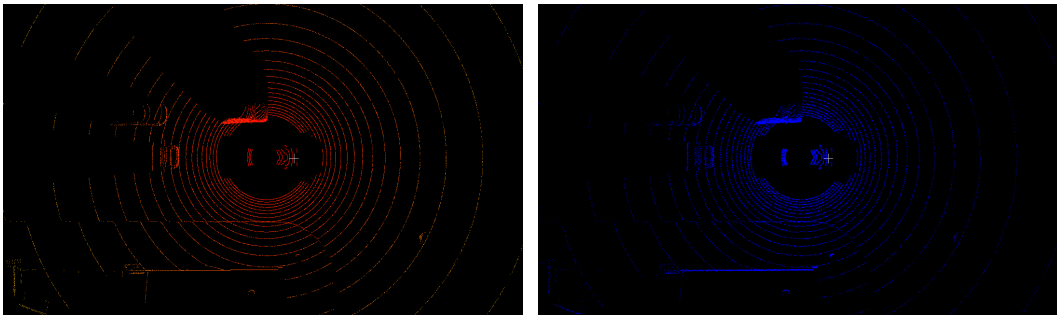


Figura 5.7: Comparación entre nube de puntos del conjunto de datos sintético (izquierda) y la misma nube de puntos tras aplicar el preprocesado en ella (derecha).

Los resultados de la evaluación de los modelos se encuentra en las Tablas 5.10, 5.12, 5.14, 5.16. En las Tablas 5.11, 5.13, 5.15, 5.17, se comparan la precisión obtenida de los modelos con ambos conjuntos de datos. Antes de empezar con el análisis es necesario mencionar que se han utilizado los modelos disponibles, pero en este caso en particular no tiene sentido hablar de funcionamiento en tiempo real, ya que el objetivo de esta parte del estudio es determinar la robustez de cada modelo frente al cambio de dominio, por lo que sería injusto comparar modelos sin acumulación de nubes y con acumulación de pocas nubes de puntos. Es por eso que los modelos funcionales en tiempo real no se compararán con otros modelos, únicamente se compararán con sus versiones con mayor acumulación de nubes de puntos. Para un análisis más extenso, son necesarios los modelos entrenados con la misma acumulación de nubes de puntos.

PointPillars

Modelo Reentrenado 10 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.662	0.222	**0.210**	0.288	3.312	**0.673**	0.225	0.215	**0.261**	3.301
truck	0.522	0.270	0.258	**0.133**	2.070	0.515	0.273	0.267	0.144	2.082
bus	0.194	0.341	0.399	0.300	1.888	0.180	0.345	0.400	0.194	1.814
pedestrian	**0.873**	0.147	0.776	**0.144**	0.964	0.841	0.150	0.778	0.153	0.923
motorcycle	0.540	**0.214**	0.263	0.316	2.922	0.533	0.220	0.273	0.327	2.887
bicycle	0.333	**0.183**	**0.191**	**0.087**	1.936	0.337	0.189	0.205	0.101	2.116
Mean	0.521	0.229	**0.349**	0.211	2.182	0.513	0.234	0.356	**0.197**	2.187
Modelo Reentrenado 2 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.602	**0.208**	0.216	0.313	2.076	0.620	0.212	0.220	0.300	**2.073**
truck	**0.553**	**0.254**	**0.257**	0.161	1.624	0.542	0.255	0.261	0.169	**1.573**
bus	**0.338**	**0.324**	0.387	0.103	1.442	0.312	**0.324**	**0.383**	**0.095**	**1.353**
pedestrian	0.687	**0.142**	0.763	0.239	0.856	0.660	0.145	**0.760**	0.244	**0.838**
motorcycle	**0.590**	0.237	0.251	0.258	**1.632**	0.583	0.241	**0.248**	**0.246**	1.687
bicycle	0.650	0.204	0.248	0.142	**1.261**	**0.661**	0.205	0.252	0.170	1.270
Mean	**0.570**	**0.228**	0.354	0.203	1.482	0.563	0.230	0.354	0.204	**1.466**

Tabla 5.10: Análisis de los resultados del modelo PointPilars [22] con datos sintéticos.

Nº *sweeps*	10 *sweeps*			2 *sweeps*		
Dataset	nuScenes	Sintético	Variación	nuScenes	Sintético	Variación
car	0.803	0.662	-0.141	0.781	0.602	-0.179
truck	0.362	0.522	+0.160	0.332	0.553	+0.221
bus	0.426	0.194	-0.232	0.440	0.338	-0.102
pedestrian	0.742	0.873	+0.131	0.694	0.687	-0.007
motorcycle	0.332	0.540	+0.208	0.269	0.590	+0.321
bicycle	0.094	0.333	+0.239	0.088	0.650	+0.562
Mean Average Precision	0.460	0.521	+0.061	0.434	0.570	+0.136

Tabla 5.11: Comparación entre los resultados (*Average Precision*) de la evaluación del modelo PointPillars [22] con el conjunto de datos nuScenes [7] y el conjunto de datos sintético.

Shape Signature Network

Modelo Reentrenado 10 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.615	0.205	**0.203**	0.329	2.334	0.636	0.211	0.206	0.330	2.323
truck	0.564	0.242	0.241	0.140	2.702	0.584	0.241	0.241	0.117	2.544
bus	0.316	0.332	0.391	0.141	2.675	0.309	0.334	**0.390**	0.122	2.567
pedestrian	0.769	0.146	0.743	0.339	0.806	0.732	0.149	0.744	0.342	0.792
motorcycle	**0.640**	0.213	**0.219**	0.445	1.972	0.626	0.222	0.223	0.369	1.922
bicycle	0.721	**0.145**	0.191	0.262	1.307	0.740	**0.145**	0.190	0.382	1.277
Mean	0.604	0.214	**0.331**	0.276	1.966	0.604	0.217	0.332	0.277	1.904
Modelo Reentrenado 2 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.638	**0.196**	0.205	0.296	2.079	**0.652**	0.202	0.210	**0.280**	**2.068**
truck	0.614	**0.233**	**0.237**	**0.109**	2.053	**0.616**	**0.233**	0.240	0.113	**1.959**
bus	**0.356**	**0.320**	0.402	0.157	1.332	0.316	0.327	0.399	**0.101**	**1.257**
pedestrian	**0.815**	**0.139**	0.741	**0.219**	0.794	0.794	0.143	**0.739**	0.225	**0.784**
motorcycle	0.610	0.202	0.242	0.326	1.761	0.604	**0.210**	0.252	**0.325**	**1.759**
bicycle	**0.753**	0.152	**0.187**	**0.116**	1.296	0.752	0.155	0.190	**0.116**	**1.229**
Mean	**0.631**	**0.207**	0.336	0.204	1.552	0.622	0.212	0.338	**0.193**	**1.509**

Tabla 5.12: Análisis de los resultados del modelo Shape Signature Network [23] con datos sintéticos.

Nº *sweeps*	10 *sweeps*			2 *sweeps*		
Dataset	nuScenes	Sintético	Variación	nuScenes	Sintético	Variación
car	0.816	0.615	-0.201	0.792	0.638	-0.154
truck	0.464	0.564	+0.100	0.440	0.614	+0.174
bus	0.561	0.316	-0.245	0.540	0.356	-0.184
pedestrian	0.665	0.769	+0.104	0.657	0.815	+0.158
motorcycle	0.427	0.640	+0.213	0.373	0.610	+0.237
bicycle	0.158	0.721	+0.563	0.104	0.753	+0.649
Mean Average Precision	0.515	0.604	+0.089	0.484	0.631	+0.147

Tabla 5.13: Comparación entre los resultados (*Average Precision*) de la evaluación del modelo Shape Signature Network [23] con el conjunto de datos nuScenes [7] y el conjunto de datos sintético.

CenterPoint

Modelo Reentrenado 10 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.331	0.338	0.277	0.259	5.203	0.563	0.221	0.207	0.374	3.529
truck	0.320	0.281	0.492	0.155	4.843	**0.590**	**0.242**	**0.251**	0.105	3.042
bus	0.037	**0.000**	**0.000**	**0.000**	**0.000**	**0.247**	0.303	0.421	0.083	3.309
pedestrian	0.455	0.147	**0.691**	0.455	0.951	**0.949**	**0.116**	0.774	**0.125**	**0.674**
motorcycle	0.000	0.000	0.000	0.000	0.000	**0.054**	**0.215**	0.399	**1.194**	**3.080**
bicycle	0.094	0.313	0.253	0.039	1.345	**0.204**	0.187	0.231	0.657	2.246
Mean	0.206	0.180	0.285	**0.151**	**2.057**	**0.434**	0.214	0.380	0.423	2.647
Modelo Reentrenado 0 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.125	**0.000**	**0.000**	**0.000**	**0.000**	**0.588**	0.209	0.197	0.694	4.119
truck	0.007	0.359	0.500	**0.008**	**0.252**	0.428	0.247	0.252	0.356	4.866
bus	0.000	0.000	0.000	0.000	0.000	0.216	0.370	0.400	0.438	3.710
pedestrian	0.053	0.172	0.708	1.778	1.315	0.312	0.122	0.733	1.514	1.577
motorcycle	0.000	0.000	0.000	0.000	0.000	0.024	0.292	**0.388**	1.885	4.360
bicycle	0.053	**0.000**	**0.000**	**0.000**	**0.000**	0.119	0.212	0.196	1.174	3.809
Mean	0.040	**0.088**	**0.201**	0.298	0.261	0.281	0.241	0.361	1.010	3.740

Tabla 5.14: Análisis de los resultados del modelo CenterPoint [24] con datos sintéticos.

Nº *sweeps*	10 *sweeps*			0 *sweeps*		
Dataset	nuScenes	Sintético	Variación	nuScenes	Sintético	Variación
car	0.821	0.563	-0.258	0.741	0.588	-0.153
truck	0.484	0.590	+0.106	0.399	0.428	+0.029
bus	0.627	0.247	-0.380	0.570	0.216	-0.354
pedestrian	0.748	0.949	+0.201	0.654	0.312	-0.342
motorcycle	0.433	0.054	-0.379	0.317	0.024	-0.293
bicycle	0.302	0.204	-0.098	0.180	0.119	-0.061
Mean Average Precision	0.569	0.434	-0.135	0.477	0.281	-0.196

Tabla 5.15: Comparación entre los resultados (*Average Precision*) de la evaluación del modelo CenterPoint [24] con el conjunto de datos nuScenes [7] y el conjunto de datos sintético.

TransFusion

Modelo Reentrenado 10 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.000	0.000	0.000	0.000	0.000	0.770	0.178	0.231	0.234	3.664
truck	0.000	0.000	0.000	0.000	0.000	0.693	0.233	0.296	0.023	1.468
bus	0.000	0.000	0.000	0.000	0.000	0.246	**0.321**	0.369	**0.082**	**1.078**
pedestrian	0.000	0.000	0.000	0.000	0.000	0.974	0.103	0.815	**0.080**	**0.461**
motorcycle	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
bicycle	0.081	**0.146**	0.275	0.744	6.571	0.311	0.158	0.297	**0.489**	**1.501**
Mean	0.013	**0.024**	**0.046**	**0.124**	1.095	0.499	0.165	0.335	0.151	1.362
Modelo Reentrenado 0 *sweeps*										
Preprocesado	No					Si				
Object Class	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$	$AP \uparrow$	$ATE \downarrow$	$ASE \downarrow$	$AOE \downarrow$	$AVE \downarrow$
car	0.500	**0.096**	**0.179**	**0.018**	**0.451**	**0.771**	0.182	0.226	0.651	3.739
truck	0.500	0.326	**0.217**	**0.012**	**0.000**	**0.831**	**0.282**	0.264	0.073	4.138
bus	0.000	0.000	0.000	0.000	0.000	**0.271**	0.333	**0.341**	0.219	3.515
pedestrian	0.000	0.000	0.000	0.000	0.000	**0.983**	**0.080**	**0.786**	1.081	1.348
motorcycle	0.000	0.000	0.000	0.000	0.000	**0.057**	**0.213**	**0.403**	**1.096**	**2.399**
bicycle	0.079	0.175	**0.206**	0.846	2.345	**0.351**	0.220	0.274	0.914	3.363
Mean	0.180	0.099	0.100	0.146	**0.466**	**0.544**	0.218	0.382	0.672	3.084

Tabla 5.16: Análisis de los resultados del modelo TransFusion [25] con datos sintéticos.

Nº *sweeps*	10 *sweeps*			0 *sweeps*		
Dataset	nuScenes	Sintético	Variación	nuScenes	Sintético	Variación
car	0.871	0.770	-0.101	0.806	0.771	-0.035
truck	0.387	0.693	+0.306	0.426	0.831	+0.405
bus	0.734	0.246	-0.488	0.696	0.271	-0.425
pedestrian	0.866	0.974	+0.108	0.783	0.983	+0.200
motorcycle	0.620	0.000	-0.620	0.531	0.057	-0.474
bicycle	0.294	0.311	+0.017	0.285	0.351	+0.066
Mean Average Precision	0.629	0.499	-0.130	0.588	0.544	-0.044

Tabla 5.17: Comparación entre los resultados (*Average Precision*) de la evaluación del modelo TransFusion [25] con el conjunto de datos nuScenes [7] y el conjunto de datos sintético.

Lo primero que se puede apreciar observando estas tablas es como los modelos PointPillars y SSN son capaces de obtener buenos resultados en las nubes sin preprocesado mientras que modelos más actuales como CenterPoint y TransFusion tienen problemas en este formato de nubes. Otro aspecto a destacar es que, si bien realizar el preprocesado a las nubes parece no afectar demasiado a PointPillars y SSN e incluso es perjudicial, se puede observar una gran mejora en la precisión de los otros dos modelos, especialmente en TransFusion. Esta mejora se puede apreciar mejor en la Tabla 5.18. De esta información se pueden sacar dos conclusiones:

- Los dos primeros modelos son más flexibles frente al cambio, haciéndolos más adecuados para tareas en las que exista una posible brecha de dominio entre los datos disponibles para entrenamiento y los datos en los que se va a usar el modelo. Esto se confirma al observar las tablas que indican la variación de la evaluación de los modelos para dos conjuntos de datos diferentes, donde los modelos PointPillars y

SSN muestran una diferencia muy pequeña en los resultados de ambas evaluaciones, mientras que CenterPoint y TransFusion muestran una diferencia muy alta.

- El preprocesado que se le realiza a las nubes de puntos, produce que estas en cierta manera sean más parecidas a nubes de puntos reales. Si se observaran solo los modelos PointPillars y SSN se podría pensar que no hay una brecha de dominio significativa y que el preprocesado no es efectivo, pero son los modelos que mejores resultados obtienen con datos reales los que más sufren esta brecha de dominio y los que más se benefician del preprocesado.

Modelo	PointPillars [22]		SSN [23]		CenterPoint [24]		TransFusion [25]	
Object Class	10 *sweeps*	2 *sweeps*	10 *sweeps*	2 *sweeps*	10 *sweeps*	0 *sweeps*	10 *sweeps*	0 *sweeps*
car	+0.011	+0.018	+0.021	+0.014	+0.232	+0.463	+0.770	+0.271
truck	-0.007	-0.011	+0.020	+0.002	+0.270	+0.421	+0.693	+0.331
bus	-0.014	-0.026	-0.007	-0.040	+0.210	+0.216	+0.246	+0.271
pedestrian	-0.032	-0.027	-0.037	-0.021	+0.494	+0.259	+0.974	+0.983
motorcycle	-0.007	-0.007	-0.014	-0.006	+0.054	+0.024	+0.000	+0.057
bicycle	+0.004	+0.011	+0.019	-0.001	+0.110	+0.066	+0.230	+0.272
Mean	-0.008	-0.007	0.000	-0.009	+0.228	+0.242	+0.486	+0.364

Tabla 5.18: Diferencia del *Average Precision* Error de los modelos al aplicarles el preprocesado.

Otro aspecto a destacar es el *Average Velocity Error*, debido a sus altos valores. Al comprobar las detecciones de los modelos sin acumulación de nubes de puntos, se detectó que, en casi todos los casos, asignaban valores muy cercanos a cero o cero a la velocidad de los objetos. Esto explica por qué el error es menor en objetos que se mueven a menor velocidad como los peatones. En una primera instancia, esto parece lógico, pues la manera más común de calcular la velocidad de un objeto es con la diferencia de la posición del mismo en dos instantes de tiempo diferentes y cercanos, al no tener acumulación de nubes de puntos el modelo no tiene información de posiciones anteriores del objeto. Sin embargo, esto no sucede con nubes de puntos reales, en las tablas de la sección anterior se puede apreciar un error en la velocidad mucho menor, tanto con acumulación como sin. La hipótesis de por qué el error es tan alto con las nubes de puntos sintéticas es debido a que estas nubes no poseen ningún tipo de distorsión de los objetos debido al movimiento de los mismos. Un LiDAR rotativo real, va obteniendo los puntos según rota y, por tanto, según el resto de objetos están en movimiento. Esto produce que los objetos que aparecen en la nube de puntos se vean distorsionados si estos o el ego-vehicle están en movimiento. La distribución de puntos resultante probablemente sea un factor clave para la aproximación de la velocidad. Sin embargo, debido a la manera en la que se obtienen las nubes de puntos sintéticas, los vehículos no muestran ninguna distorsión en base a su velocidad. Se cree que esta falta de distorsión afecta negativamente a los modelos que se aprovechan de la acumulación de nubes de puntos, ya que estas nubes presentan un mismo objeto en varias posiciones, pero sin distorsión, lo que, en ciertas situaciones, podría hacer pensar al modelo que hay varios objetos parados. Esto explicaría que la precisión de los modelos no aumente notablemente o incluso que se reduzca al acumular nubes, como se puede ver en las tablas.

Además de los resultados analizados, en la Figura 5.8 se pueden ver detecciones en nubes de puntos sintéticas, en la columna izquierda las anotaciones y en la columna derecha las detecciones.

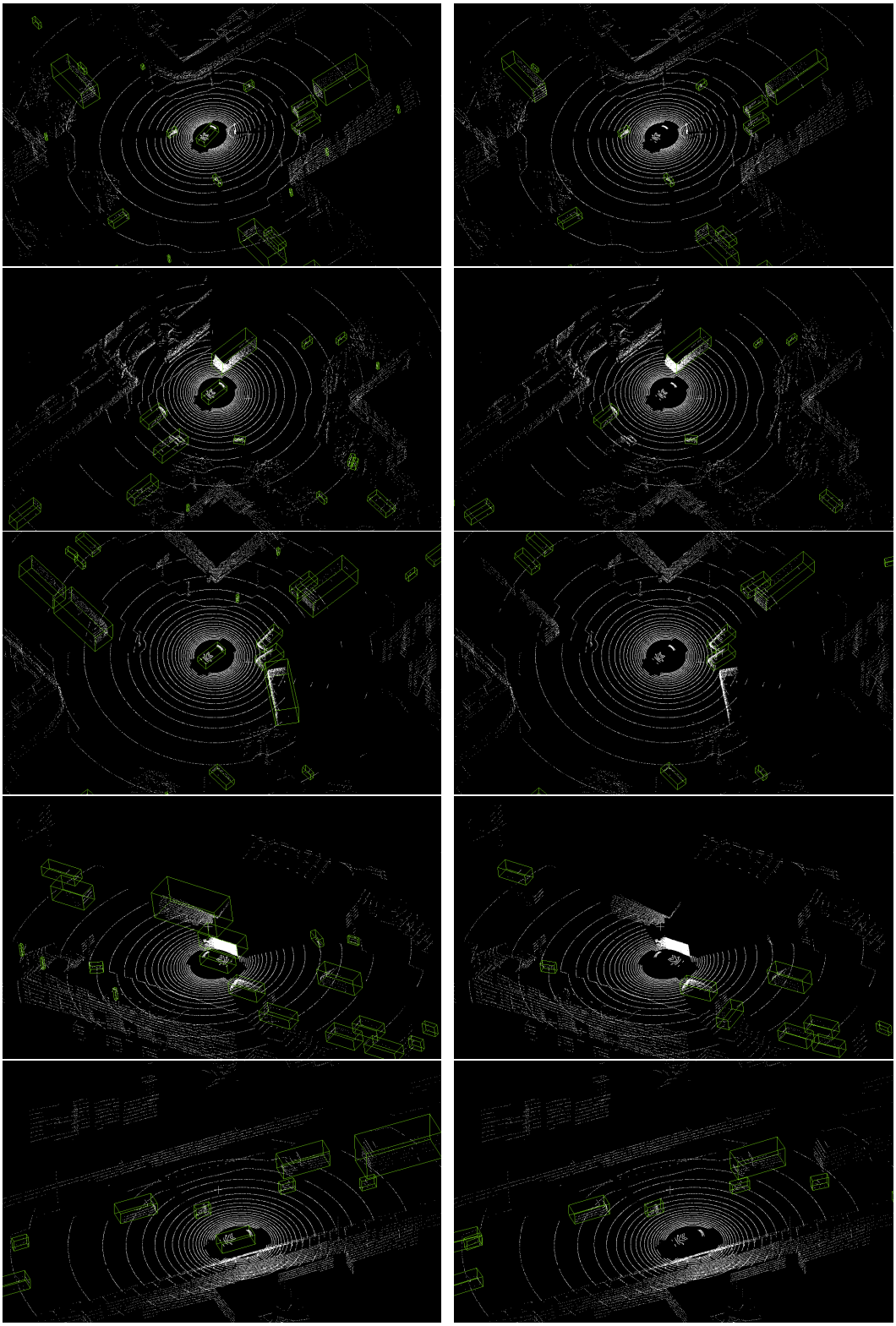


Figura 5.8: Comparación entre las anotaciones del conjunto de datos sintético (izquierda) y las detecciones del modelo TransFusion [25] (derecha).

5.1.3. Resultados en nubes de puntos de otros dominios

Una vez demostrada la fiabilidad de estos modelos en el dominio de los datos sintéticos, se han probado los modelos en otros dominios. A pesar de tener datos de otros dominios estos no están anotados, por lo que solamente se mostrarán resultados cualitativos.

El primer dominio en el que se probaron los modelos fue el interior de un parking interior. Las nubes de puntos utilizadas se han obtenido con un LiDAR de 32 haces, esto las hace muy similares a las nubes de puntos del conjunto de datos con el que los modelos han sido entrenados. Sin embargo, se consideran nubes de puntos de otro dominio debido a que estas nubes tienen puntos en el techo del parking, estos puntos podrían confundir a los modelos debido a que nuScenes no ofrece ninguna nube de puntos en el interior de un parking. En la Figura 5.9 se pueden ver los resultados, y se puede apreciar que los modelos obtienen resultados similares con estas nubes de puntos y las del vehículo autónomo de Vicomtech.

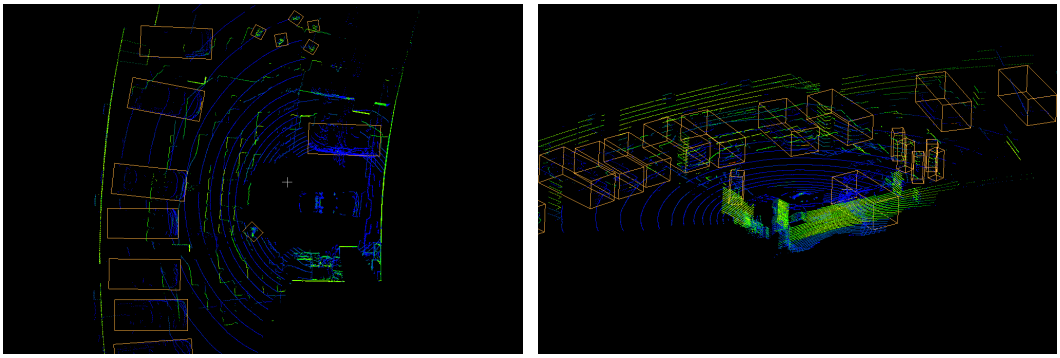


Figura 5.9: Detecciones (peatones y coches) del modelo TransFusion [25] de una nube de puntos obtenida en el interior de un parking.

El siguiente dominio en el que se utiliza es en unas nubes de puntos obtenidas con un LiDAR rotativo de la marca Livox[5]. Este tiene un funcionamiento bastante diferente al del conjunto de datos de nuScenes, generando nubes con una distribución de puntos diferente. En la Figura 5.10 se pueden ver los resultados, si bien los modelos son capaces de detectar algunos vehículos, solo funciona con aquellos que estén cerca del sensor, desaprovechando la cualidad principal de este LiDAR que es su rango.

Por último, se ha probado en un dominio externo a los coches autónomos. En este caso, se han utilizado nubes de puntos del conjunto de datos *Open Sensor Data for Rail 2023* [88]. Este conjunto de datos entre otras cosas consta de nubes de puntos en un entorno ferroviario, estas nubes de puntos están compuestas por 6 LiDARes diferentes. En este caso hay tres factores por los que la nube de puntos es diferente a las del conjunto de datos nuScenes, los cuales son el entorno, la cantidad de puntos y la distribución de los mismos. En la Figura 5.11 se pueden ver los resultados, sin embargo, para obtener estos resultados se ha trasladado la nube de puntos en el eje z, debido a que los peatones están encima del andén y los modelos solo detectan los objetos que están al ras del suelo, lo que implica que en este caso no se podrían detectar al mismo tiempo objetos en las vías y en los andenes. Además, los modelos solo son capaces de detectar algunos de los peatones.

[5] https://www.livoxtech.com/

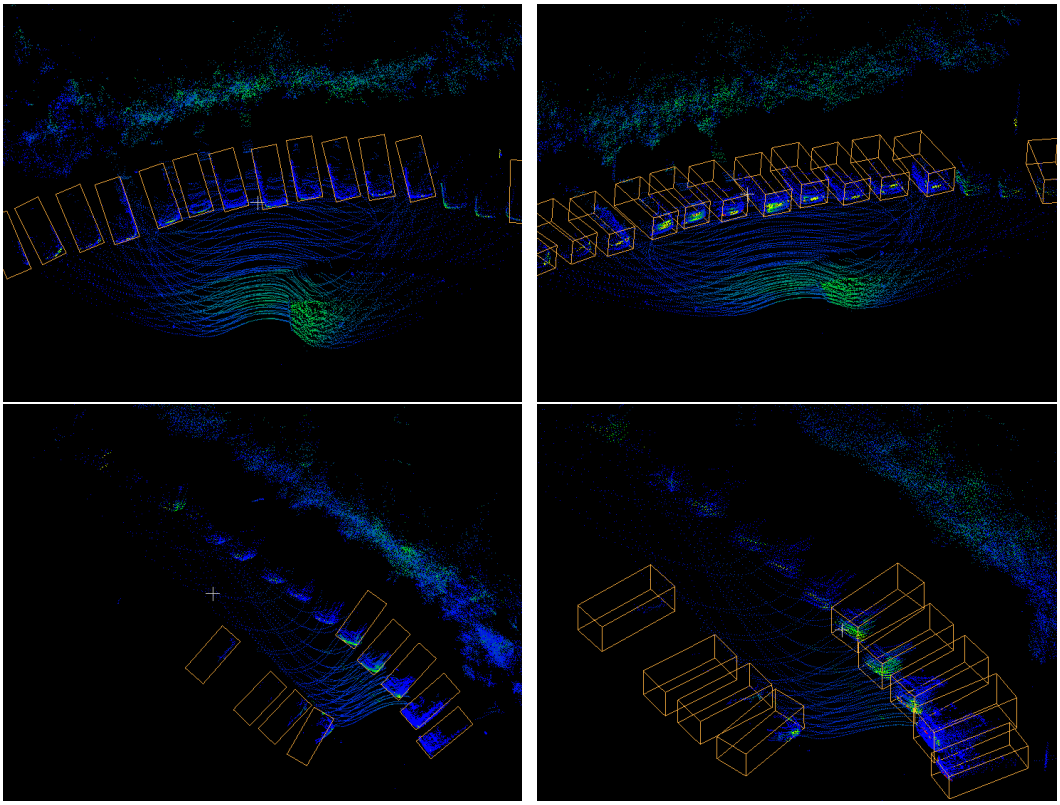


Figura 5.10: Detecciones del modelo TransFusion [25] de una nube de puntos obtenida con un sensor LiDAR de la marca Livox.

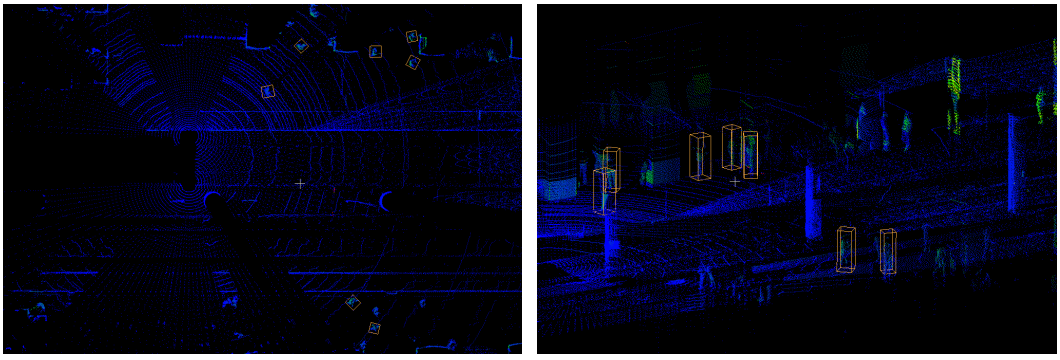


Figura 5.11: Detecciones del modelo TransFusion [25] de una nube de puntos del conjunto de datos Open Sensor Data for Rail 2023 [88].

Si bien en el primer dominio presentado se han podido detectar todos los objetos, en los otros dos dominios se presentan problemas que indican que los modelos no están listos para utilizarse en esos dominios, sin embargo, los modelos son capaces de detectar algunos objetos. Por ello es posible que estos puedan ser utilizados en estos dominios si se entrenan con un conjunto de datos diferente o si se utilizan técnicas de adaptación de dominio durante el entrenamiento.

5.1.4. Tiempo de inferencia en el vehículo autónomo

Al final del proyecto, se tuvo acceso al vehículo autónomo de Vicomtech. Oportunidad que se aprovechó para medir el tiempo de inferencia de cada uno de los modelos con diferentes acumulaciones de nubes de puntos, dando como resultado la Figura 5.12 y en la Tabla 5.19. Este vehículo posee un ordenador con dos Nvidia 2080ti, pero para este experimento solo se ha utilizado una.

Number of sweeps	PointPillars	SSN	CenterPoint	TransFusion
0	25.3	**29.4**	14.9	15.0
1	19.6	**22.8**	11.8	12.2
2	15.1	**17.6**	10.6	9.7
3	12.4	**14.2**	9.4	8.4
4	9.9	**11.5**	8.4	7.1
5	8.2	**9.5**	7.3	6.1
6	6.9	**7.9**	6.6	5.3
7	5.9	**6.6**	6.1	4.6
8	5.1	**5.7**	5.3	4.1
9	4.4	**4.9**	**4.9**	3.6
10	3.9	4.3	**4.4**	3.2

Tabla 5.19: Frecuencia de inferencia (nubes de puntos/segundo) de los modelos con diferentes acumulaciones de nubes de puntos (GPU: Nvidia 2080ti).

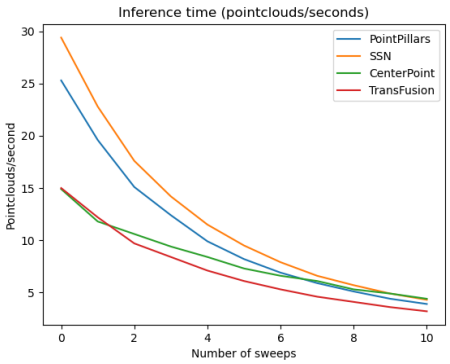


Figura 5.12: Frecuencia de inferencia (nubes de puntos/segundo) de los modelos con diferentes acumulaciones de nubes de puntos (GPU: Nvidia 2080ti).

Sorprendentemente, los modelos han resultado ser más rápidos en estas GPUs a pesar de tener menor capacidad de cómputo, esto puede deberse a la manera en la que el HPC reparte los recursos a los trabajos que recibe. Estos datos son más fiables debido a que han sido probados sin que ningún otro proceso se estuviera ejecutando en el mismo ordenador. Estos resultados conllevan implicaciones altamente favorables, dado que permiten la integración de modelos más potentes en el vehículo. Asimismo, se otorga un mayor margen de tiempo en caso de que otros programas necesiten realizar un preprocesado de los datos.

CAPÍTULO 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

Tras la creación de un *pipeline* de generación de datos sintéticos y el extenso estudio de los modelos de detección de objetos en nubes de puntos, se ha llegado a las siguientes conclusiones.

En primer lugar, ha sido necesario familiarizarse con temas de conducción autónoma, desde los componentes usados, el estado del arte en cuanto a detección de objetos en nubes de puntos, software de *Machine Learning* como MMdetection3D, software de simulación de automoción como CARLA y el propio vehículo de prueba de Vicomtech CarLOTA.

Se han cumplido los objetivos del proyecto, obteniendo modelos entrenados de detección de objetos en nubes de puntos, se ha completado el *pipeline* para la creación de un código capaz de generar datos sintéticos a partir del simulador CARLA con el que se han generado datos y posteriormente se han evaluado los modelos entrenados.

En cuanto a los modelos, se ha comprobado que cuantas más nubes de puntos sean acumuladas más robusto será el modelo, sin embargo, esta potencia reduce drásticamente la velocidad del modelo, no se ha podido demostrar que reduciendo la cantidad de clases que el modelo puede detectar este se vuelve más preciso. Los modelos entrenados pueden ser usados en tiempo real, siendo SSN el que mejores resultados es capaz de obtener si es necesario que estos funcionen en tiempo real. En cambio, si la velocidad del modelo no es una prioridad, por ejemplo, para generar preanotaciones, el modelo TransFusion es el que obtiene mejores resultados.

Se ha creado un código capaz de generar datos sintéticos en un entorno de automoción realista anotado automáticamente. Posteriormente, se ha utilizado este código para generar un conjunto de datos con el que se han evaluado los modelos. Mientras que PointPillars y SSN han sido capaces de adaptarse bien al cambio de dominio de real a sintético, CenterPoint y TransFusion apenas son capaces de detectar objetos en nubes de puntos sintéticas. Para intentar mejorar la precisión de los modelos, se ha propuesto realizar un preprocesamiento de las nubes. Tras aplicar este preprocesamiento, PointPillars y SSN no muestran una gran mejora; sin embargo, CenterPoint y TransFusion sí lo hacen, siendo TransFusion el modelo que obtiene la mejor puntuación. Otro problema que se ha detectado a lo largo del estudio

es que los modelos no son capaces de predecir de manera adecuada la velocidad de los objetos en las nubes de puntos sintéticas, esto posiblemente se deba a que el simulador detiene la simulación para obtener las nubes.

También se ha podido comprobar que los modelos son capaces de detectar objetos en diferentes dominios, tales como: datos reales y datos sintéticos, o incluso de manera cualitativa en el interior de un parking o usando otro tipo de LiDAR. Sin embargo, no se puede afirmar que se adapten bien a todos los dominios, ya que en la mayoría no son capaces de obtener resultados similares a los obtenidos con el conjunto de datos nuScenes.

Tras el estudio se puede afirmar que el cambio de dominio afecta desfavorablemente y es algo a tener en cuenta. En muchos trabajos suelen utilizarse datos sintéticos para entrenar modelos y crear escenas de conducción interesantes; sin embargo, en muchas ocasiones no se tiene en cuenta la brecha de dominio. En lo que respecta a las nubes de puntos sintéticas en automoción, no se han realizado muchas propuestas para tratar de reducir esta brecha de dominio.

Además del estudio realizado, se han integrado con éxito los modelos entrenados en este proyecto en el vehículo autónomo de Vicomtech de dos maneras diferentes: la primera mediante la herramienta RTMaps y la segunda mediante la herramienta ROS. Es necesario destacar que esta integración no la ha realizado el autor de la memoria, sino que ha colaborado con investigadores del centro.

6.2. Trabajo a futuro

A pesar de ser un proyecto bastante completo, aún hay mucho trabajo que se puede realizar y muy probablemente se realizará.

Antes se ha mencionado que uno de los problemas de las nubes sintéticas es su intensidad poco realista. Ya hay estudios para añadirle intensidad a nubes de puntos sin esta [87], en este artículo utilizan modelos de *Machine Learning*. Sin embargo, una solución más simple podría ser la creación de un sensor de CARLA en Unreal. Vistas las diferencias de dominio que hay, se debe seguir investigando la brecha de dominio que existe para minimizarlo tanto entre dominios real y sintético como diferentes dominios reales, donde los datos de entrenamiento suelen tener ciertas diferencias con los datos para los que se necesita el modelo.

Si bien varios modelos de la librería MMDetection3D han sido estudiados, esta librería posee muchos más modelos implementados para los conjuntos de datos KITTI, Waymo y Lift. Sabiendo esto, con intención de ampliar el estudio de los modelos se podría trabajar en adaptar uno de estos modelos. Tarea que se empezó a realizar con el modelo PartA2 [26], pero que se pospuso para priorizar el resto de tareas.

Además de los modelos propios de la librería, existen muchas implementaciones de modelos que, según sus artículos, obtienen mejores resultados que los modelos utilizados en este estudio [89, 90, 91]. Estos modelos podrían ser adaptados para la librería de MMdetection3D con el fin de ampliar el estudio. Asimismo este proyecto se ha centrado en modelos que usan únicamente LiDAR, pero hay varios modelos que hacen fusión de varios sensores [92, 93, 94], estos modelos también aportarían mucha información al estudio.

No solamente se puede trabajar con modelos que fusionan sensores; también se pueden

usar herramientas como EagerMOT [95], que permiten asociar las detecciones 2D con las detecciones 3D. La combinación de los modelos utilizados en este estudio junto con otros modelos de detección 2D mediante la herramienta EagerMOT está actualmente en estudio y desarrollo en Vicomtech.

Un objetivo adicional al proyecto es la optimización de los modelos vistos en este, ya sea realizando modificaciones en la arquitectura o aplicando técnicas de optimización como la cuantización de los pesos, o transformando los modelos a TensorRT. Para las dos últimas opciones mencionadas, existe una librería supuestamente compatible con los modelos de MMDetection3D, MMDeploy [30]. Si bien esta herramienta ha sido utilizada en algún momento del proyecto, no se han obtenido resultados significativos y debido a que la optimización no es un objetivo del proyecto no se ha mencionado en la memoria.

Para finalizar, debido a la labor de investigación que se ha realizado a lo largo del proyecto, se trabajará en la elaboración de un artículo científico sobre este proyecto. No solo eso, también se colaborará en un artículo científico sobre el *pipeline* de la generación de un conjunto de datos sintético usando el entorno de simulación CARLA. También hay interés en llevar a cabo un estudio sobre el uso de nubes de puntos sintéticas para el entrenamiento de modelos de detección 3D, principalmente con el objetivo de publicar un artículo en el futuro con las conclusiones del estudio.

Bibliografía

[1] Santiago Royo and Maria Ballesta-Garcia. An overview of lidar imaging systems for autonomous vehicles. *Applied Sciences*, 9:4093, 09 2019.

[2] Brahayam Ponton, Magda Ferri, Lars König, and Marcus Bartels. Efficient extrinsic calibration of multi-sensor 3d lidar systems for autonomous vehicles using static objects information. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6285–6292, 2022.

[3] Yufeng Zhu, Chenghui Li, and Yubo Zhang. Online camera-lidar calibration with sensor semantic information. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4970–4976, 2020.

[4] Guohang Yan, Zhuochun Liu, Chengjie Wang, Chunlei Shi, Pengjin Wei, Xinyu Cai, Tao Ma, Zhizheng Liu, Zebin Zhong, Yuqian Liu, Ming Zhao, Zheng Ma, and Yikang Li. Opencalib: A multi-sensor calibration toolbox for autonomous driving. *Software Impacts*, 14:100393, 2022.

[5] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6), 2021.

[6] Marcelo Pereira, David Silva, Vitor Santos, and Paulo Dias. Self calibration of multiple lidars and cameras on autonomous vehicles. *Robotics and Autonomous Systems*, 83:326–337, 2016.

[7] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, and et al Xu. nuscenes: A multimodal dataset for autonomous driving, 2019.

[8] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d, 2021.

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

[10] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, and et al Chouard. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[11] Sampurna Mandal, Swagatam Biswas, Valentina Balas, Rabindra Shaw, and Ankush Ghosh. *Lyft 3D object detection for autonomous vehicles*, pages 119–136. Elsevier, 01 2021.

[12] Siddharth Agarwal, Ankit Vora, Gaurav Pandey, Wayne Williams, Helen Kourous, and James R. McBride. Ford multi-av seasonal dataset. *CoRR*, abs/2003.07969, 2020.

[13] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[14] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. 2020.

[15] Matthew Pitropov, Danson Evan Garcia, Jason Rebello, Michael Smart, Carlos Wang, Krzysztof Czarnecki, and Steven Lake Waslander. Canadian adverse driving conditions dataset. *CoRR*, abs/2001.10117, 2020.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017.

[17] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. Lgsvl simulator: A high fidelity simulator for autonomous driving, 2020.

[18] Koustav Mullick, Harshil Jain, Sanchit Gupta, and Amit Arvind Kale. Domain adaptation of synthetic driving datasets for real-world autonomous driving, 2023.

[19] Fabio Reway, Abdul Hoffmann, Diogo Wachtel, Werner Huber, Alois Knoll, and Eduardo Ribeiro. Test method for measuring the simulation-to-reality gap of camera-based object detection algorithms for autonomous driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1249–1256, 2020.

[20] Chuqing Hu, Sinclair Hudson, Martin Ethier, Mohammad Al-Sharman, Derek Rayside, and William Melek. Sim-to-real domain adaptation for lane detection and classification in autonomous driving. In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 457–463, 2022.

[21] Xiangyu Bai, Yedi Luo, Le Jiang, Aniket Gupta, Pushyami Kaveti, Hanumant Singh, and Sarah Ostadabbas. Bridging the domain gap between synthetic and real-world data for autonomous driving, 2023.

[22] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2018.

[23] Xinge Zhu, Yuexin Ma, Tai Wang, Yan Xu, Jianping Shi, and Dahua Lin. Ssn: Shape signature networks for multi-class object detection from point clouds, 2020.

[24] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CVPR*, 2020.

[25] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers, 2022.

[26] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network, 2019.

[27] Yanwei Li, Yilun Chen, Xiaojuan Qi, Zeming Li, Jian Sun, and Jiaya Jia. Unifying voxel-based representation with transformer for 3d object detection, 2022.

[28] Junho Koh, Junhyung Lee, Youngwoo Lee, Jaekyum Kim, and Jun Won Choi. Mgtanet: Encoding sequential lidar points using long short-term motion-guided temporal attention for 3d object detection, 2022.

[29] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. https://github.com/open-mmlab/mmdetection3d, 2020.

[30] MMDeploy Contributors. Openmmlab's model deployment toolbox. https://github.com/open-mmlab/mmdeploy, 2021.

[31] A. D. NHTSA Systems. Automated driving systems 2.0: A vision for safety. 2013.

[32] Ryan Harrington, Carmine Senatore, John Scanlon, and Ryan Yee. The role of infrastructure in an automated vehicle future. *Bridge*, 40:48–55, 06 2018.

[33] H A J Sadiq Sha and Pradeep P Patil. Study on self-driving cars. *Seybold Report*, volume 15:789–801, 07 2020.

[34] Teh STANDARD PREVIEW. Road vehicles — vehicle dynamics and road-holding ability — vocabulary. *INTERNATIONAL STANDARD*, 2011.

[35] Moad Kissai, Bruno Monsuez, Xavier Mouton, Didier Martinez, and Adriana Tapus. Adaptive robust vehicle motion control for future over-actuated vehicles. *Machines*, 7:26, 04 2019.

[36] Juan Diego Ortega, Neslihan Kose, Paola Cañas, Min-An Chao, Alexander Unnervik, Marcos Nieto, Oihana Otaegui, and Luis Salgado. DMD: A large-scale multi-modal driver monitoring dataset for attention and alertness analysis. *CoRR*, abs/2008.12085, 2020.

[37] Walaa Othman, Alexey Kashevnik, Ammar Ali, and Nikolay Shilov. Drivermvt: In-cabin dataset for driver monitoring including video and vehicle telemetry information. *Data*, 7(5), 2022.

[38] Phillip Taylor, Nathan Griffiths, Abhir Bhalerao, Zhou Xu, Adam Gelencser, and Thomas Popham. Warwick-jlr driver monitoring dataset (dmd): Statistics and early findings. AutomotiveUI '15, page 89–92, New York, NY, USA, 2015. Association for Computing Machinery.

[39] Gledson Melotti, Cristiano Premebida, and Nuno Gonçalves. Multimodal deep-learning for object recognition combining camera and lidar data. 04 2020.

[40] Khaled Elmadawi, Moemen Abdelrazek, Mohamed Elsobky, Hesham Eraqi, and Mohamed Zahran. End-to-end sensor modeling for lidar point cloud. pages 1619–1624, 10 2019.

[41] Mahdi Rezaei, Mutsuhiro Terauchi, and Reinhard Klette. Robust vehicle detection and distance estimation under challenging lighting conditions. *IEEE Transactions on Intelligent Transportation Systems*, 01 2015.

[42] Manasvi Sagarkar, Shreyas More, Akash Singh, Prithwish Jana, and Bithika Pal. Perception and planning in autonomous car. 12 2020.

[43] Zhihang Song, Zimin He, Xingyu Li, Qiming Ma, Ruibo Ming, Zhiqi Mao, Huaxin Pei, Lihui Peng, Jianming Hu, Danya Yao, and Yi Zhang. Synthetic datasets for autonomous driving: A survey, 2023.

[44] Andreas Kloukiniotis, Andreas Papandreou, Christos Anagnostopoulos, Aris Lalos, Petros Kapsalas, Duong-Van Nguyen, and Konstantinos Moustakas. Carlascenes: A synthetic dataset for odometry in autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4520–4528, June 2022.

[45] Jean-Emmanuel Deschaud. KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator. *arXiv e-prints*, 2021.

[46] Jean-Emmanuel Deschaud, David Duque, Jean Pierre Richa, Santiago Velasco-Forero, Beatriz Marcotegui, and François Goulette. Paris-carla-3d: A real and synthetic outdoor point cloud dataset for challenging tasks in 3d mapping. *Remote Sensing*, 13(22), 2021.

[47] Ahmed Rida Sekkat, Yohan Dupuis, Varun Ravi Kumar, Hazem Rashed, Senthil Yogamani, Pascal Vasseur, and Paul Honeine. SynWoodScape: Synthetic surround-view fisheye camera dataset for autonomous driving. *IEEE Robotics and Automation Letters*, 7(3):8502–8509, jul 2022.

[48] Emanuele Alberti, Antonio Tavera, Carlo Masone, and Barbara Caputo. IDDA: a large-scale multi-domain dataset for autonomous driving. *CoRR*, abs/2004.08298, 2020.

[49] Runsheng Xu, Hao Xiang, Xin Xia, Xu Han, Jinlong Li, and Jiaqi Ma. Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication, 2022.

[50] Ye Wang, Weiwen Deng, Zhenyi Liu, and Jinsong Wang. Deep learning-based vehicle detection with synthetic image data. *IET Intelligent Transport Systems*, 13(7):1097–1105, 2019.

[51] DR Niranjan, BC VinayKarthik, and Mohana. Performance analysis of ssd and faster rcnn multi-class object detection model for autonomous driving vehicle research using carla simulator. In

2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT), pages 1–6, 2021.

[52] B Ravi Kiran, Luis Roldao, Benat Irastorza, Renzo Verastegui, Sebastian Suss, Senthil Yogamani, Victor Talpaert, Alexandre Lepoutre, and Guillaume Trehard. Real-time dynamic object detection for autonomous driving using prior 3d-maps. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[53] Tom Bu, Xinhe Zhang, Christoph Mertz, and John M. Dolan. Carla simulated data for rare road object detection. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2794–2801, 2021.

[54] Weihua Gao, Jiakai Tang, and Taotao Wang. An object detection research method based on carla simulation. *Journal of Physics: Conference Series*, 1948(1):012163, jun 2021.

[55] D.R. Niranjan, B C VinayKarthik, and Mohana. Deep learning based object detection model for autonomous driving research using carla simulator. In *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 1251–1258, 2021.

[56] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[57] Hao "Tang, Song Bai, Li Zhang, Philip H. S. Torr, and Nicu"Sebe. "xinggan for person image generation". In Andrea "Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael"Frahm, editors, *Çomputer Vision – ECCV 2020*
, pages "717–734", Çham", "2020". "Springer International Publishing".

[58] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. Learning to generate realistic lidar point clouds, 2022.

[59] Ahmad El Sallab, Ibrahim Sobh, Mohamed Zahran, and Mohamed Shawky. Unsupervised neural sensor models for synthetic lidar data augmentation, 2019.

[60] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. Voxel R-CNN: towards high performance voxel-based 3d object detection. *CoRR*, abs/2012.15712, 2020.

[61] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[62] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[63] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[64] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[65] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[66] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[67] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[68] Xiangyu He and Jian Cheng. Revisiting L1 loss in super-resolution: A probabilistic view and beyond. *CoRR*, abs/2201.10084, 2022.

[69] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

[70] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[71] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5):183–197, 1991.

[72] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. *CoRR*, abs/1808.01244, 2018.

[73] Xuanyao Chen, Tianyuan Zhang, Yue Wang, Yilun Wang, and Hang Zhao. Futr3d: A unified sensor fusion framework for 3d detection, 2023.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[75] Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of DETR with spatially modulated co-attention. *CoRR*, abs/2101.07448, 2021.

[76] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, and Ping Luo. Sparse R-CNN: end-to-end object detection with learnable proposals. *CoRR*, abs/2011.12450, 2020.

[77] Zhuyu Yao, Jiangbo Ai, Boxun Li, and Chi Zhang. Efficient DETR: improving end-to-end object detector with dense prior. *CoRR*, abs/2104.01318, 2021.

[78] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. *CoRR*, abs/2010.04159, 2020.

[79] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *CoRR*, abs/2005.12872, 2020.

[80] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

[81] Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of DETR with spatially modulated co-attention. *CoRR*, abs/2101.07448, 2021.

[82] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. *CoRR*, abs/1904.08189, 2019.

[83] Nidhi Kalra and Susan M. Paddock. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* RAND Corporation, Santa Monica, CA, 2016.

[84] O. Senderos M. Nieto and O. Otaegui. Boosting ai applications: Labeling format for complex datasets. *SoftwareX*, 2021.

[85] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[86] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[87] Benoit Guillard, Sai Vemprala, Jayesh K. Gupta, Ondrej Miksik, Vibhav Vineet, Pascal Fua, and Ashish Kapoor. Learning to simulate realistic lidars, 2022.

[88] Rustam Tagiew, Martin Köppel, Karsten Schwalbe, Patrick Denzler, Philipp Neumaier, Tobias Klockau, Martin Boekhoff, Pavel Klasek, and Roman Tilly. Osdar23: Open sensor data for rail 2023, 2023.

[89] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia. Largekernel3d: Scaling up kernels in 3d sparse cnns, 2023.

[90] Tao Lu, Xiang Ding, Haisong Liu, Gangshan Wu, and Limin Wang. Link: Linear kernel for lidar-based 3d perception, 2023.

[91] Jinglin Zhan, Tiejun Liu, Rengang Li, Jingwei Zhang, Zhaoxiang Zhang, and Yuntao Chen. Real-aug: Realistic scene synthesis for lidar augmentation in 3d object detection, 2023.

[92] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation, 2022.

[93] Zeyu Yang, Jiaqi Chen, Zhenwei Miao, Wei Li, Xiatian Zhu, and Li Zhang. Deepinteraction: 3d object detection via modality interaction, 2022.

[94] Junjie Yan, Yingfei Liu, Jianjian Sun, Fan Jia, Shuailin Li, Tiancai Wang, and Xiangyu Zhang. Cross modal transformer: Towards fast and robust 3d object detection, 2023.

[95] Aleksandr Kim, Aljosa Osep, and Laura Leal-Taixé. Eagermot: 3d multi-object tracking via sensor fusion. *CoRR*, abs/2104.14682, 2021.