Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Estudio de sistemas basados en cámara para la detección tridimensional de objetos en conducción autónoma

Autor: Miguel Antunes García

Tutor: Luis Miguel Bergasa Pascual

2023

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Estudio de sistemas basados en cámara para la detección tridimensional de objetos en conducción autónoma

Autor: Miguel Antunes García

Tutor: Luis Miguel Bergasa Pascual

Tribunal:

Presidente: Carlos Andrés Luna Vázquez

Vocal 1º: Manuel Ocaña Miguel

Vocal 2º: Luis Miguel Bergasa Pascual

Fecha de depósito: 15 de septiembre de 2023

A mi madre, familia y compañeros ...

"El conocimiento no es una vasija que se llena, sino un fuego que se enciende."
Plutarco

Agradecimientos

Antes de comenzar, quiero aprovechar esta oportunidad para expresar mi más sincero agradecimiento a todas las personas que han contribuido a que este Trabajo de Fin de Máster haya llegado a buen puerto.

A mi familia y especialmente a mi madre María del Carmen, por su apoyo y sacrificio durante estos años, lo que me ha impulsado haca el futuro y permitido llegar hasta este punto de mi camino.

A mis amigos, quienes han estado a mi lado brindándome su apoyo, compañía y risas. Su amistad ha hecho que este viaje sea mucho más llevadero.

A mis compañeros del grupo RobeSafe: Santi, Pablo, Fabio, Navil, Carlos, Rodrigo y Felipe; por compartir ideas, conocimientos y experiencias que han enriquecido mi trabajo y me han ayudado a sobrellevar los peores momentos. No podría olvidarme de Javier Araluce y Javier de la Peña, que aunque ya no se encuentren en grupo, también me han influenciado y apoyado en los últimos años para llegar hasta aquí.

Agradecer a mi tutor Luis Miguel Bergasa Pascual, por depositar su confianza en mi al acogerme en este gran grupo, guiándome tanto en lo académico como en lo personal. Espero que mi viaje por la conducción autónoma continúe siendo tan fructífero bajo su dirección en los próximos años.

Por último, agradecer a la Universidad de Alcalá, por proporcionarme tantos conocimientos y experiencias durante los últimos seis años.

Este trabajo no habría sido posible sin vuestro apoyo. A todos, muchas gracias.

Resumen

La percepción del entorno es un componente esencial en el funcionamiento de sistemas autónomos. Este rol se vuelve especialmente crítico en el caso de vehículos autónomos, donde el entendimiento del entorno se enfrenta a escenarios altamente complejos y velocidades variables y elevadas. En este contexto, la cámara se ha consolidado como uno de los sensores más ampliamente utilizados, gracias a su costo asequible y facilidad de implementación. Adicionalmente, los avances significativos en técnicas de reconocimiento y detección en los últimos años han elevado su potencial de manera notable.

El objetivo principal de este trabajo de fin de máster es el análisis del estado del arte de técnicas que permitan obtener detecciones de objetos en el entorno tridimensional de un vehículo, utilizando la información procedente únicamente de las cámaras. Concretamente se cuantifican las diferencias entre dos tipos de sistemas de percepción: por un lado, los sistemas end-to-end monoculares, que obtienen las detecciones de objetos directamente en una única etapa a partir de una imagen de la cámara; y por otro lado, los sistemas modulares estéreo, que requieren múltiples etapas para obtener las detecciones. En este último enfoque, se utiliza una primera etapa para estimar la profundidad en la imagen antes de realizar las detecciones de los objetos, lo cual es común en el SOTA. Tras una identificación inicial de las principales técnicas, se procede a seleccionar varios sistemas, que destacan por sus resultados sobresalientes o por su eficiencia en términos de velocidad de procesamiento, y a someterlos a varios procesos de entrenamiento y evaluación.

Los datos utilizados para los experimentos proceden del dataset KITTI [1], sin embargo, debido a que este trabajo se enmarca en un contexto de una arquitectura completa de conducción autónoma, en la que habitualmente se llevan a cabo pruebas en entornos simulados, se utilizan también datos del simulador CARLA [2] recopilados en el dataset SHIFT [3] para cuantificar la capacidad de generalización de los sistemas en estos dos entornos.

El objetivo final es estudiar las capacidades y limitaciones de estas técnicas, realizando una evaluación justa que permita elegir la mejor implementación para ser utilizada sobre un vehículo autónomo real y comprobar la capacidad de generalizar el conocimiento tanto en entornos reales como simulados.

Palabras clave: Deep Learning, Transfer Learning, Detección de objetos 3D, KITTI, CARLA.

Abstract

Perceiving the environment is an essential component in the operation of autonomous systems. This role becomes especially critical in the case of autonomous vehicles, where understanding the environment is challenged by highly complex scenarios and varying high speeds. In this context, the camera has emerged as one of the most widely used sensors, thanks to its affordability and ease of implementation. Additionally, significant advancements in recognition and detection techniques in recent years have notably enhanced its potential.

The main objective of this master's thesis is to analyze the state of the art in techniques for obtaining object detections in the three-dimensional environment of a vehicle using information only from cameras. Specifically, the differences between two types of perception systems are quantified: on one hand, monocular end-to-end systems, which directly obtain object detections in a single stage from a camera image; and on the other hand, stereo modular systems, which require multiple stages to obtain the detections. In this approach, a preliminary stage is used to estimate depth in the image before performing object detections, a common practice in the state of the art. Starting from this point, several systems are selected, either for their outstanding results or for their processing speed efficiency, and subjected to various training and evaluation processes.

The KITTI [1] dataset provides the data used for the experiments. However, since this work is part of a context involving a complete autonomous driving architecture, where testing is typically conducted in simulated environments, data from the CARLA simulator [2], collected in the SHIFT dataset [3], is used to assess the systems generalization capabilities in both real-world and simulated environments.

The ultimate goal is to study the capabilities and limitations of these techniques, conducting a fair evaluation that allows for the selection of the best implementation for use on a real autonomous vehicle and assessing the knowledge generalization capacity in both real and simulated environments.

Keywords: Deep Learning, Transfer Learning, 3D object detection, KITTI, CARLA.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

AOS	Average Orientation Similarity.
AP	Average Precision.
BEV	Bird Eye View.
CNN	Convolutional Neural Networks.
DL	Deep Learning.
FN	False Negative.
FOV	Field of view.
FP	False Positive.
GPU	Graphics Processing Unit.
HOG	Histogram of Oriented Gradients.
IA	Inteligencia Artificial.
IoU	Intersection Over Union.
ML	Machine Learning.
MLP	Multi Layer Perceptron.
NLP	Natural Language Processing.
RCNN	Region Convolutional Neural Network.
SAE	Society of Automotive Engineers.
SOTA	State Of The Art.
SSD	Single Shot MultiBox Detector.
SVM	Support Vector Machine.

TP True Positive.

YOLO You Only Look Once.

Capítulo 1

Introducción

En algún lugar, algo increíble está esperando ser conocido.

Carl Sagan

1.1 Conducción autónoma y el grupo RobeSafe

Gracias al avance tecnológico en los últimos años, han aparecido sistemas de conducción con niveles de autonomía mas avanzados, ofreciendo cada vez un mayor grado de asistencia a la conducción. Para conocer hasta que punto la asistencia afecta a la acción del conductor, la Society of Automotive Engineers (SAE) americana propuso una escala con cinco niveles de autonomía [4] [5], siendo el nivel 0 un vehículo sin ningún tipo de automatización:

- Nivel 1: el control del vehículo recae totalmente en el conductor. El sistema inteligente proporciona asistencia básica como la detección de señales de tráfico o el freno automático ante situaciones de emergencia.
- Nivel 2: el conductor sigue siendo la pieza clave en el funcionamiento del vehículo, sin embargo, los sistemas inteligentes pueden llegar a tomar parte del control del vehículo. En esta categoría encontramos herramientas como el ajuste al carril o de velocidad, que actúan sobre la dirección y los pedales.
- Nivel 3: el vehículo puede tomar el control por completo en ciertas situaciones y durante largos periodos de tiempo. El conductor debe mantener la atención suficiente para tomar el control en unos pocos segundos cuando sea necesario.
- Nivel 4: este nivel implica que el vehículo puede mantener la autonomía en todas las situaciones posibles sin intervención humana. En caso de emergencias se requiere acción humana, luego el conductor debe poder tomar el control aunque se le permita realizar acciones como leer o dormir.
- Nivel 5: el vehículo realiza todas las acciones necesarias a la hora de conducir, eliminando la necesidad de la presencia de conductor.

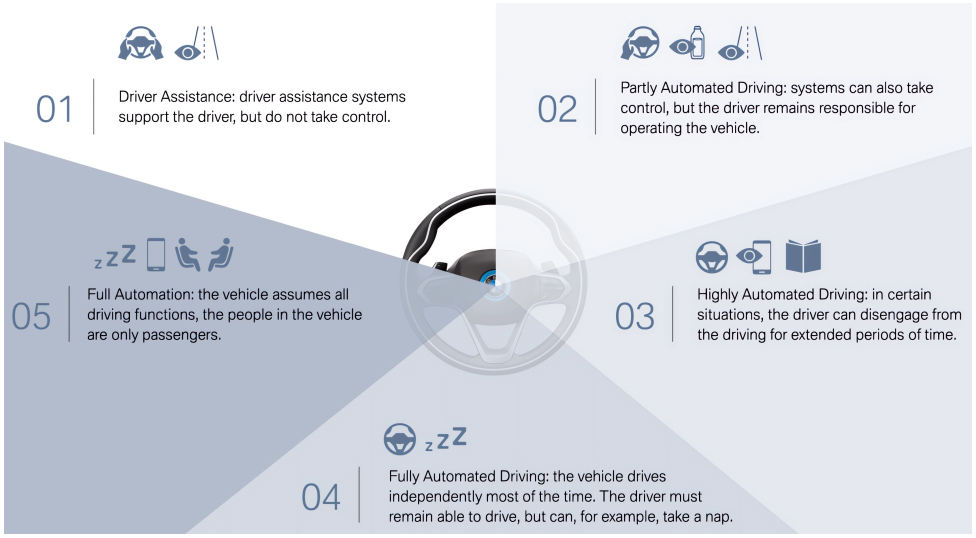

Figura 1.1: Niveles de autonomía en un vehículo autónomo [5].

El trabajo presentado en este documento se lleva a cabo en el marco del grupo de investigación RobeSafe, cuyo enfoque principal radica en el desarrollo e implementación de los distintos componentes de la arquitectura de vehículos autónomos. En la actualidad, contamos con un vehículo equipado con diversos sensores como cámaras estéreo Zed y LiDAR Velodyne (Figura 1.2), que permiten operar a un nivel de autonomía 3. Específicamente, este estudio se centra en la capa de percepción, la cual tiene como objetivo caracterizar el entorno en el que el vehículo se desplaza con el fin de tomar decisiones adecuadas a partir de la información de los diferentes sensores.

Figura 1.2: Vehículo del grupo RobeSafe.

1.2 Sistemas de percepción

Los sistemas de percepción desempeñan un papel esencial en diversos ámbitos de los sistemas inteligentes, especialmente debido a su importancia en la adquisición y comprensión de la información del entorno. A través de diferentes sensores se adquiere la información en bruto, la cual va a ser procesada por la capa de inteligencia del sistema autónomo, normalmente basada en técnicas de Inteligencia Artificial (IA). Existen multitud de sensores que pueden capturar información del exterior, de los cuales los mas utilizados son:

- Cámara: recoge la información visual de una región del entorno, perdiendo información tridimensional pero consiguiendo información 2D y semántica muy rica en su campo de visión (FOV). Las oclusiones de objetos y cambios de iluminación como los que se dan en el ciclo día-noche pueden suponer problemas a la hora de emplear sistemas de IA para tareas de detección, segmentación, etc.
- LiDAR: genera nubes de puntos con información geométrica del entorno muy precisa, reduciendo posibles errores en objetos con cierta oclusión, transparencias o cambios de iluminación. Como principales inconvenientes se pueden destacar el precio del sensor, considerablemente mas elevado que las cámaras o los radares estándar, la sensibilidad a condiciones atmosféricas adversas como lluvia o niebla, o la necesidad de mas potencia de cómputo para procesar nubes que pueden llegar a ser muy pesadas.
- Radar: al igual que el LiDAR, este sensor genera una nube de puntos del entorno, sin embargo, la densidad de información es mucho menor, pudiendo suponer problemas para identificar objetos de poco tamaño. Sus puntos a favor son la capacidad de proporcionar información de las velocidades de los objetos a través del Doppler y su robustez a condiciones ambientales adversas.

En la Figura 1.3 se puede ver una comparativa entre la información que obtiene cada sensor, observando las diferencias comentadas anteriormente. Las nubes de puntos de LiDAR y Radar se han proyectado sobre la imagen.

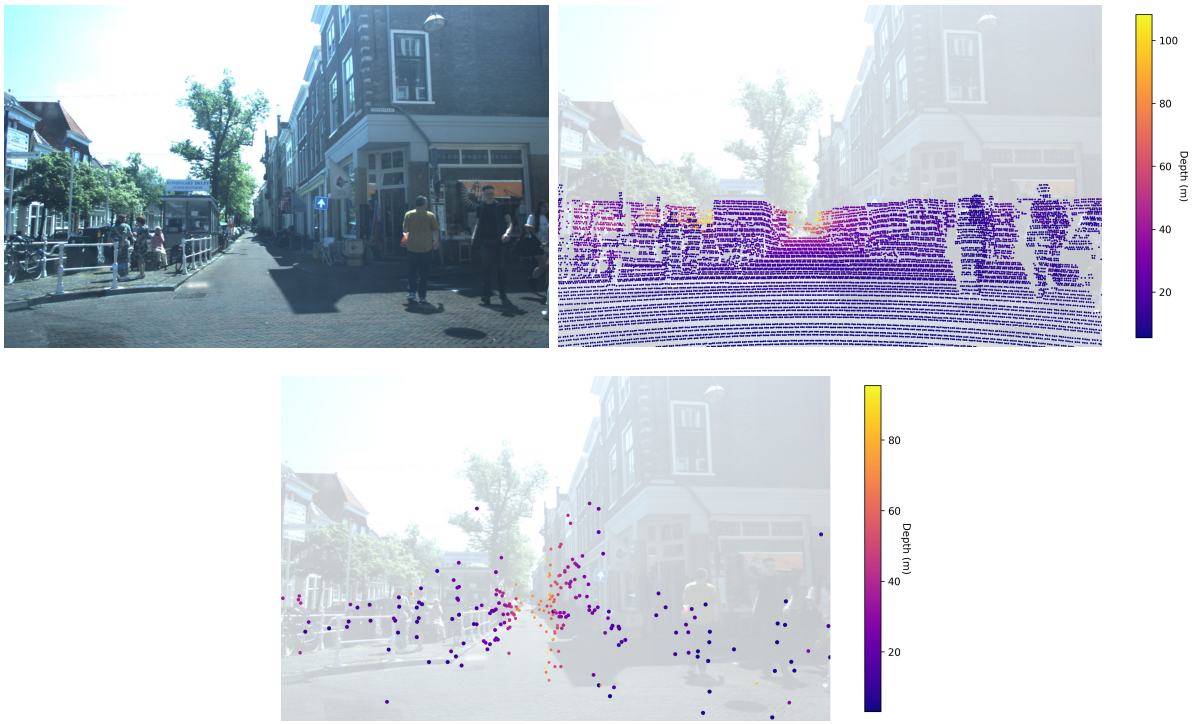

Figura 1.3: Ejemplo de datos de cada sensor proyectados a la imagen.
a) Cámara b) LiDAR c) Radar

1.3 Metodologías clásicas y Deep Learning para detección de objetos

La detección de objetos es un campo ampliamente explorado en visión artificial debido a su gran utilidad en sistemas robóticos. Como resultado, se han desarrollado numerosas técnicas y algoritmos basados en Machine Learning (ML) para abordar este desafío. Entre los primeros sistemas de detección de objetos en imágenes se encuentra el detector de características Histogram of Oriented Gradients (HOG) [6], que utiliza un enfoque clásico con un extractor de características HOG y un clasificador basado en Support Vector Machine (SVM). Sin embargo, estos sistemas clásicos presentan limitaciones en la detección de objetos de diferentes escalas y en la generalización de las detecciones. En este contexto, también han surgido nuevas arquitecturas neuronales, como el Perceptrón o el Multi Layer Perceptron (MLP), para abordar estos desafíos. Aprovechando el surgimiento de técnicas basadas en redes neuronales convolucionales (CNN), estos modelos clásicos se han visto sustituidos por detectores de DL como RCNN de dos etapas [7] (Figura 1.4) o las primeras versiones del detector YOLO [8] con una sola etapa.

Actualmente las CNN conviven con las redes basadas en mecanismos de atención como Transformers [9], las cuales comienzan a ganar relevancia en detección con arquitecturas como ViT [10] aunque originalmente se utilizaron en otras tareas como Natural Language Processing.

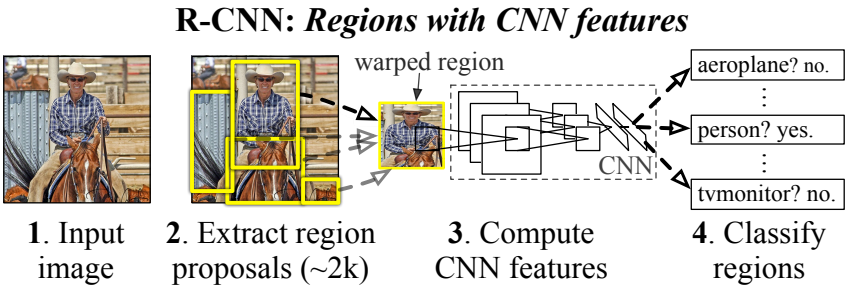

Figura 1.4: Esquema de las etapas del detector RCNN [7].

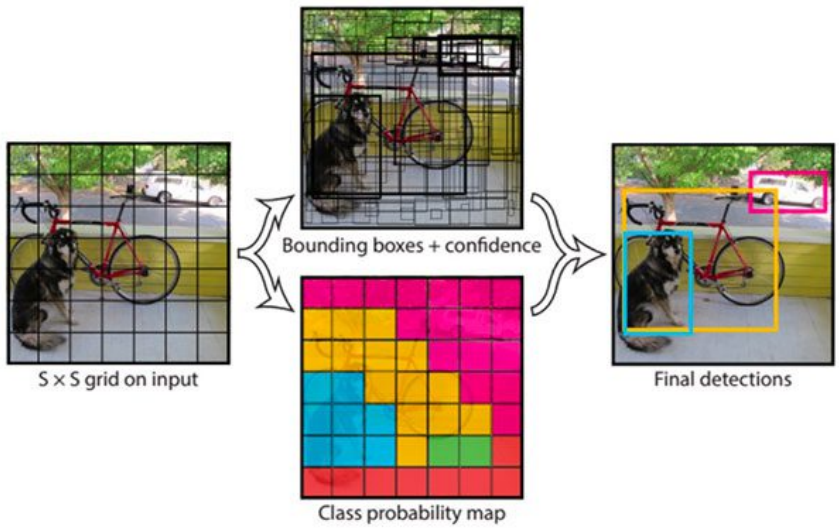

Figura 1.5: Esquema de detección de la YOLO original [8].

1.4 Objetivo del trabajo

El trabajo presentado tiene como objetivo principal llevar a cabo un estudio del estado del arte de varios sistemas de detección de objetos tridimensionales basados en cámara. Concretamente se pretende comparar el rendimiento y resultados de sistemas monoculares de una única etapa o end-to-end, y sistemas estéreo modulares con varias etapas. Para lograr este objetivo, se inicia con una revisión teórica de una gama de sistemas de detección y estimación disponibles en la literatura. Durante esta revisión, se destacan las ventajas y los inconvenientes asociados a cada uno de ellos, así como los diferentes niveles de rendimiento que se han obtenido en diversas aplicaciones y escenarios.

En el caso de los sistemas end-to-end, las detecciones de los objetos se obtienen directamente de las imágenes. Primeramente se realiza una revisión exhaustiva del SOTA para identificar las mejores opciones disponibles. A continuación se priorizan las técnicas SMOKE [11] y PGD [12] debido a sus buenos resultados y su compatibilidad con un framework común ampliamente utilizado, procediendo a su entrenamiento y evaluación.

En el caso de los sistemas modulares, el proceso es más complejo debido a la necesidad de extraer en primer lugar la información tridimensional de las imágenes. En el SOTA esto se logra mediante una primera estimación de la profundidad sobre las imágenes. Por lo tanto, en primer

lugar es necesario llevar a cabo una revisión del SOTA centrada en estimación de profundidad para seleccionar las opciones más adecuadas. En este contexto, se opta por entrenar y evaluar SDN [13], debido a sus resultados destacados, y CoEx [14], debido a su velocidad. Respecto a la arquitectura modular, se emplea como punto de partida la propuesta de los autores de SDN en su publicación PseudoLidar++ [13]. Esta arquitectura incluye la estimación de profundidad, rectificación y detección 3D utilizando PointPillars [15]. Sobre este esquema se realizan varias modificaciones para acelerar el sistema y mejorar su rendimiento. Finalmente, se entrena y evalúa la arquitectura modular completa para obtener una comprensión completa de su capacidad y eficacia en nuestro contexto.

Adicionalmente, es importante destacar que el trabajo se sitúa en la capa de percepción de una arquitectura de conducción autónoma más compleja, lo que implica que, habitualmente, las pruebas y evaluaciones de dicha arquitectura suelen llevarse a cabo en un entorno simulado como paso previo a su implementación en un vehículo real. Para evaluar la capacidad de generalización de los sistemas entrenados, los datos utilizados en todas los entrenamientos y evaluaciones provienen de dos fuentes diferentes: el dataset KITTI [1], el cual recopila datos de un vehículo real; y el dataset SHIFT [3], el cual recopila datos sintéticos procedentes del simulador de conducción autónoma CARLA [2].

Los resultados obtenidos proporcionarán información valiosa para la selección y desarrollo de sistemas de detección eficientes y confiables que puedan ser empleados en el vehículo real del grupo RobeSafe.

Capítulo 2

Estudio teórico y del estado del arte

Cada día sabemos más y entendemos menos.

Albert Einstein

En este capítulo se exponen algunas de las arquitecturas y sensores mas utilizados en los últimos años en el estado del arte para la detección de objetos en entornos de conducción autónoma. En primer lugar se exponen las características de los dos sensores de interés en este trabajo, así como la forma de trabajar con los datos que proporcionan y como se relacionan entre ellos. En segundo lugar se aborda la detección de objetos 2D con cámara ya que será necesaria para implementar la detección 3D con un sistema modular. Por último, se mostrarán detalles de los sistemas de detección de objetos 3D end-to-end. Cabe destacar que los resultados mostrados en este apartado son los proporcionados por los autores de cada modelo. En el apartado de Experimentos se realizarán varios entrenamientos y evaluaciones propias de los modelos seleccionados en las mismas condiciones, de tal forma que la comparativa sea justa y en el entorno que se requiere.

2.1 Sensores en conducción autónoma

Como se ha comentado en el apartado introductorio a este documento, los vehículos pueden contar con multitud de sensores independientemente de su grado de autonomía, permitiendo desde proporcionar información para asistir al conductor, hasta la toma de acciones del propio vehículo de forma automática. En el trabajo presentado en este documento, los datos con los que se trabajan presentan formato tanto de cámara como de LiDAR, es por ello que en este apartado se realiza una introducción a estos dos sensores. Concretamente se detallan las características de cada uno de ellos, así como los diferentes sistemas de coordenadas con los que se van a trabajar y transformaciones que se deben realizar entre ambos para poder fusionar la información, lo cual es crítico en las arquitecturas propuestas.

2.1.1 Cámara

En el campo de la percepción en la conducción autónoma y la visión artificial, las cámaras desempeñan un papel esencial en la captura y procesado de la información visual del entorno del vehículo. Su capacidad para capturar imágenes detalladas y ricas en información brinda una visión similar a la humana, lo que las convierte en una herramienta fundamental en el desarrollo de sistemas autónomos y de asistencia al conductor. La gran cantidad de modelos y bajo coste las hacen una opción atractiva en la integración de sistemas de percepción en vehículos, especialmente si se quiere explotar la gran información visual que aporta. Sin embargo, como cualquier tecnología, las cámaras también presentan ciertos inconvenientes. Su rendimiento se ve afectado por condiciones de clima adversas, como lluvia o niebla, que pueden reducir la visibilidad. Además, por el propio funcionamiento de la cámara, la información capturada se proyecta sobre el plano imagen, luego la información 3D como posiciones o dimensión de objetos se pierde prácticamente por completo si trabajamos con cámaras monoculares. Esto no sucede así en caso de cámaras estéreo, ya que al contar con dos focos es posible recuperar parte de la información de profundidad con cierto error.

Las cámara cuentan con una serie de parámetros que son necesarios para caracterizar la proyección entre el mundo tridimensional y la imagen, así como aplicar rectificación si es necesario:

- Distancia focal en cada eje f_x y f_y.
- Factor de corrección s.
- Centro óptico de la imagen c_x y c_y.

Como se refleja en la Ecuación 2.1, estos parámetros se agrupan en la matriz de parámetros intrínsecos $K \in \mathbb{R}^{3\times3}$, la cual si se multiplica por la matriz de parámetros extrínsecos $E \in \mathbb{R}^{3\times4}$ que refleja la posición y rotación del sensor, se obtiene la matriz de proyección $P \in \mathbb{R}^{3\times4}$

$$K \times E = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} = P \tag{2.1}$$

Esta matriz de proyección P la suelen proporcionar los datasets directamente, y nos permite calcular las proyecciones de un punto entre el plano imagen (u, v) y el 3D (X, Y, Z) siguiendo la Ecuación 2.2.

$$\begin{bmatrix} wu \\ wv \\ w \end{bmatrix} = K \times E \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = P \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.2}$$

Por último, es importante destacar que el sistema de coordenadas 3D de la cámara que se va a utilizar es el definido por KITTI, el cual se encuentra muy extendido. Como se observa en la Figura 2.1, el eje X apunta hacia la derecha del sensor, el Y hacia abajo y el Z hacia adelante.

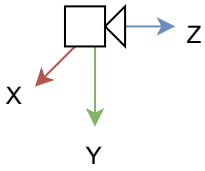

Figura 2.1: Eje de coordenadas 3D de cámara.

2.1.2 LiDAR

Dentro de la conducción autónoma y la percepción del entorno, los sensores LiDAR o Light Detection and Ranging son una tecnología esencial en la adquisición de datos tridimensionales del entorno circundante. Estos sistemas utilizan pulsos láser para medir distancias y crear mapas de puntos en 3D, permitiendo una detección y comprensión detallada de los objetos y estructuras presentes. Entre las ventajas más destacadas del LiDAR se encuentra su capacidad para generar nubes de puntos precisas, ofreciendo información tridimensional detallada e información sobre la . Esto habilita la percepción de obstáculos, vehículos, peatones y otros elementos con una resolución y confiabilidad, incluso en escenarios complejos. Además, el LIDAR es menos susceptible a las variaciones de luminosidad en comparación con otros sensores ópticos, lo que lo hace especialmente valioso en entornos nocturnos o de baja visibilidad. No obstante, su alto costo y tamaño pueden plantear desafíos en términos de integración en vehículos. Además, puede haber dificultades para detectar ciertos materiales y superficies reflectantes, lo que puede afectar su capacidad de percepción en situaciones específicas.

Respecto al sistema de coordenadas, en el caso del LiDAR los ejes varían respecto a los de la cámara, como se ve en la Figura 2.2 el eje X para a apuntar hacia el frente del sensor, el Y hacia la izquierda y el Z hacia arriba. Conociendo la posición y rotación de los sensores es posible construir matrices para transformar coordenadas entre los dos sistemas.

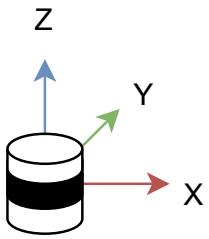

Figura 2.2: Eje de coordenadas 3D del LiDAR.

2.2 Detección de objetos 2D

La detección 2D de objetos en imágenes es un paso fundamental en el camino hacia la detección tridimensional en muchas arquitecturas. Los sistemas de detección de objetos tienen la capacidad de localizar y clasificar objetos de interés en las imágenes capturadas por la cámara del vehículo, aportando resultados muy buenos gracias a la densidad e información semántica proporcionadas por la cámara. En este trabajo concretamente, la detección 2D sobre la imágenes constituye una de las primeras etapas en el sistema de detección 3D modular, permitiendo filtrar regiones indeseadas de la escena (se proporcionan mas detalles en el Capítulo 4). Debido a que las detecciones dependen únicamente de la información proporcionada por un sensor, sus limitaciones y debilidades se transfieren directamente al resto de etapas, lo que puede tener un impacto significativo en su rendimiento. Por ejemplo, cambios en las condiciones de iluminación o en el clima pueden afectar negativamente la precisión y confiabilidad de las detecciones. Con esto en mente, es importante estudiar el rendimiento de estos detectores para garantizar unas detecciones fiables y rápidas.

A lo largo de los años se han ido desarrollando y mejorando diversas arquitecturas enfocadas en detección. Se pueden destacar detectores de una etapa como Single Shot MultiBox Detector (SSD) [16] o las primeras versiones de YOLO [8], y detectores de varias etapas como RCNN [7] y sus mejoras Fast-RCNN [17] y Faster-RCNN [18]. Actualmente su uso ha quedado bastante relegado a sistemas antiguos, debido especialmente a la aparición de nuevas redes mas potentes y eficientes, de las que se van a ofrecer detalles en este apartado del documento.

2.2.1 YOLO

La familia de detectores You Only Look Once (YOLO) se considera una de las mas importantes en tareas de detección sobre imágenes. La idea fundamental en la que se basan es en la realización de las detecciones teniendo el contexto de la imagen completo y con una única inferencia. Al procesar toda la imagen a la vez en la red en lugar de únicamente regiones separadas, se consigue mejorar tanto la velocidad como los resultados respecto a otros sistemas.

La versión v3 [19] del detector fue ampliamente utilizada durante varios años, sin embargo, se ha ido sustituyendo por versiones mas actualizadas. Actualmente las versiones v5 [20] o v8 [21] de Ultralytics se encuentran ampliamente extendidas, gracias sobre todo a su gran rendimiento y la facilidad que proporcionan los creadores para desplegarlas en todo tipo de entornos y dispositivos.

En mayo de 2023 vio la luz la versión YOLO-NAS [22] de Deci AI, la cual ha alcanzado rendimientos que superan a sus competidoras hasta la fecha y que además ha sido adaptada a la librería de Ultralytics.

Como se observa en la Figura 2.3 , cada uno de los diferentes modelos actuales cuenta con varios tamaños, los cuales permiten adaptar la escala de la red al entorno donde se vaya a trabajar y el rendimiento que se necesita en la tarea a realizar. Por ejemplo, en nuestro caso del vehículo autónomo, hay que realizar un balance para que el tiempo de inferencia sea suficientemente bajo para conseguir resultados en tiempo real, a la vez que las detecciones sean suficientemente

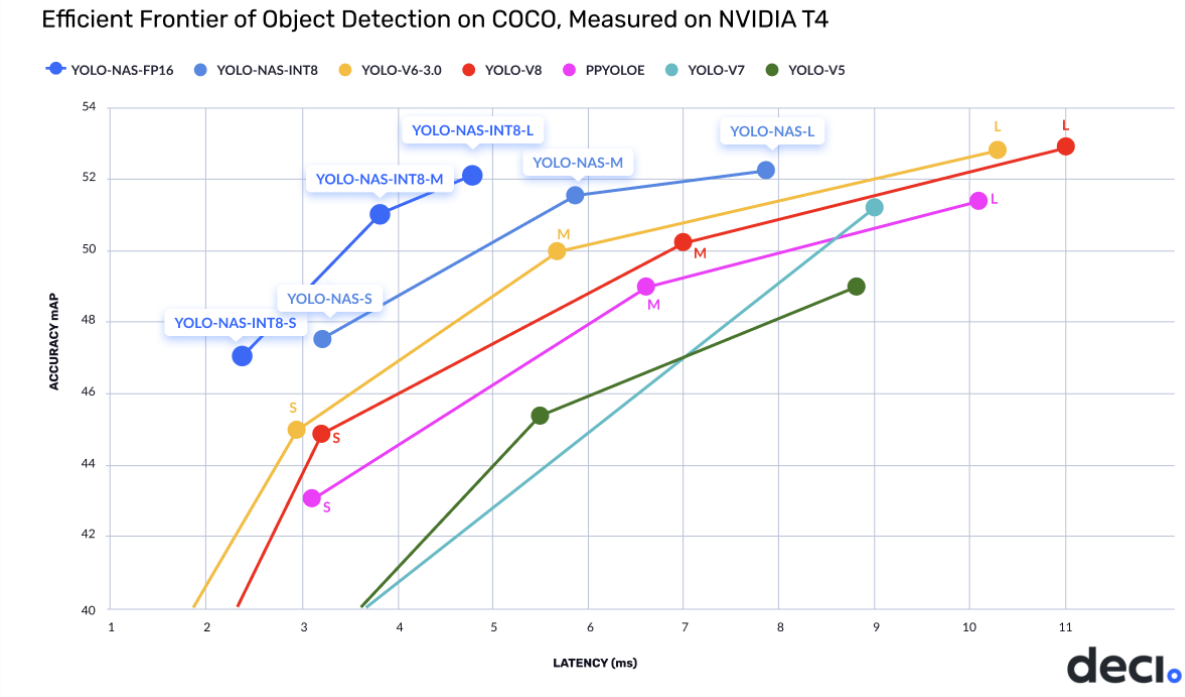

Figura 2.3: Comparativa entre el rendimiento entre las últimas versiones de YOLO [22].

buenas como para que el vehículo tome las decisiones correctas. Esta flexibilidad en el tamaño del modelo y en la librería de Python de Ultralytics convierte a estos modelos en los candidatos ideales para se utilizados en nuestro vehículo autónomo. A fecha de escritura de este trabajo, YOLOv5 [20] (Figura 2.4) presenta el código mas adecuado para un uso en tiempo real y sistemas embebidos respecto a sus competidoras.

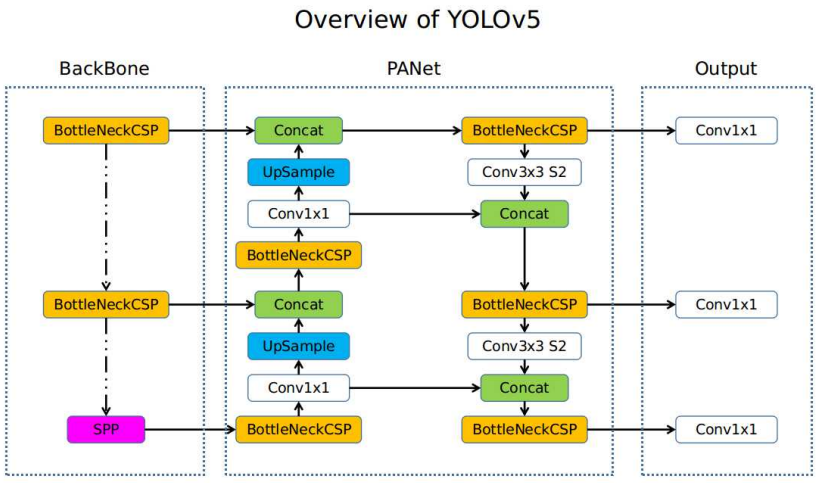

Figura 2.4: Arquitectura del detector YOLOv5.

2.2.2 Detection Transformers (DETR)

La aparición de los mecanismos de atención y Transformers en 2017 [9] ha supuesto la aparición de nuevos sistemas de DL con nuevas propiedades y rendimiento. Si bien destacan en tareas como NLP, también han surgido arquitecturas de detección de objetos.

En 2020 se propone DETR (Detection Transformer) [23], uno de los primeros modelos de detección de objetos con una estructura basada en Transformers. Con la arquitectura de la Figura 2.1, los autores consiguen resultados del estado del arte como se puede ver en la Tabla 2.1. Sin embargo, varias semanas después de su publicación salió a la luz YOLOV5, que presentaba resultados superiores y tiempos de inferencia mucho menores.

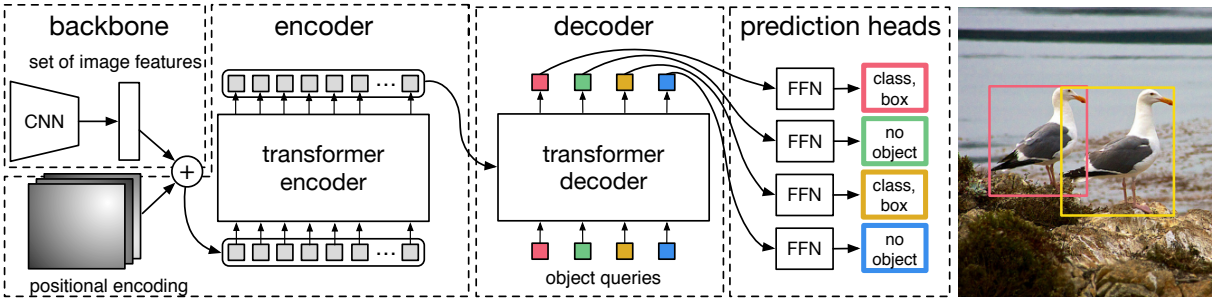

Figura 2.5: Arquitectura de DETR [23].

Modelo	FPS ↑	AP ↑
DETR	28	42.0
DETR-DC5	12	43.3
DETR-R101	20	43.5
DETR-DC5-R101	10	44.9
RT-DETR-R50	108	53.1
RT-DETR-R101	74	54.3
RT-DETR-L	114	53.0
RT-DETR-X	74	54.8

Tabla 2.1: Tabla de rendimiento de DETR y RT-DETR sobre el split de validación de COCO.

Han ido apareciendo nuevas versiones que buscan mejorar diversos aspectos. Por ejemplo RT-DETR [24] consigue superar en tiempo y rendimiento incluso a las últimas versiones de YOLO. No ha sido posible utilizarlo en los experimentos ya que ha sido publicada en julio del 2023, momento en el que los experimentos que requieren de un detector 2D ya habían finalizado.

2.3 Detección de objetos 3D monocular end-to-end

De la misma manera que existen detectores de objetos sobre imágenes como los detallados en el apartado anterior, en el estado del arte se pueden encontrar redes que buscan detectar objetos directamente en el entorno tridimensional que influye sobre el vehículo. Para conseguir la caracterización completa de los objetos de interés, además de detectar su posición y clase, también es necesario estimar sus dimensiones y las diferentes rotaciones.

En la Tabla 2.2 se puede observar una comparativa entre varios métodos del SOTA. MonoDETR [25] es el modelo más actual del estudio, presentando los mejores resultados. Respecto a los modelos que son compatibles con MMDetection3D se pueden destacar SMOKE [11] y PGD [12], el cual es una evolución de FCOS3D. En este apartado se detallan las arquitecturas y el rendimiento de los modelos mencionados.

Cabe destacar que los valores de las métricas de detectores 3D de cámara suelen ser inferiores a los de otros sensores como LiDAR, especialmente debido a que conseguir valores de IoU por encima del umbral es mas complicado en un entorno tridimensional para la cámara.

Método	**3D ↑**			**Tiempo(s) ↓**	**Framework**
	Easy	**Med**	**Hard**		
MonoDETR [25]	28.84	20.61	16.38	0.038	
SMOKE [11]	14.76	12.85	11.50	0.03	✓
MonoDLE [26]	17.45	13.66	11.68	0.04	
FCOS3D [27]	13.90	11.61	10.98	-	✓
PGD [12]	24.35	18.34	16.90	0.028	✓
ImVoxelNet (R50) [28]	24.54	17.8	15.67	0.7	✓

Tabla 2.2: Comparativa de detectores 3D end-to-end sobre el split de validación de KITTI (AP40 con IoU 0.7)

2.3.1 MonoDETR y SSD-MonoDETR

Al igual que sucede con las arquitecturas de detección sobre imágenes, podemos encontrar sistemas como MonoDETR [25] (Figura 2.6), basados en Transformers, para detectar objetos en un entorno 3D.

El primer paso del sistema es la generación de los mapas de características. Esto implica obtener los mapas visuales mediante un backbone, como ResNet-50 [29], y los mapas de profundidad a partir de las características visuales mediante la aplicación de dos capas convolucionales. Una vez obtenidas las características, se les aplica un encoder con self-attention para obtener los embeddings correspondientes. Es importante destacar que se utilizan dos ramas separadas para la información visual y de profundidad, lo que facilita el aprendizaje de las características específicas de cada tipo de dato. A continuación, se aplican bloques decoder a partir de un número N de queries, que son aprendidas durante el entrenamiento. Finalmente, las cabezas de detección entran en acción para predecir diferentes atributos de los detectados: la categoría del objeto, su detección en la imagen, la posición 3D, la profundidad, el tamaño del objeto y su orientación. En resumen, este proceso combina diversas etapas, como la generación de mapas

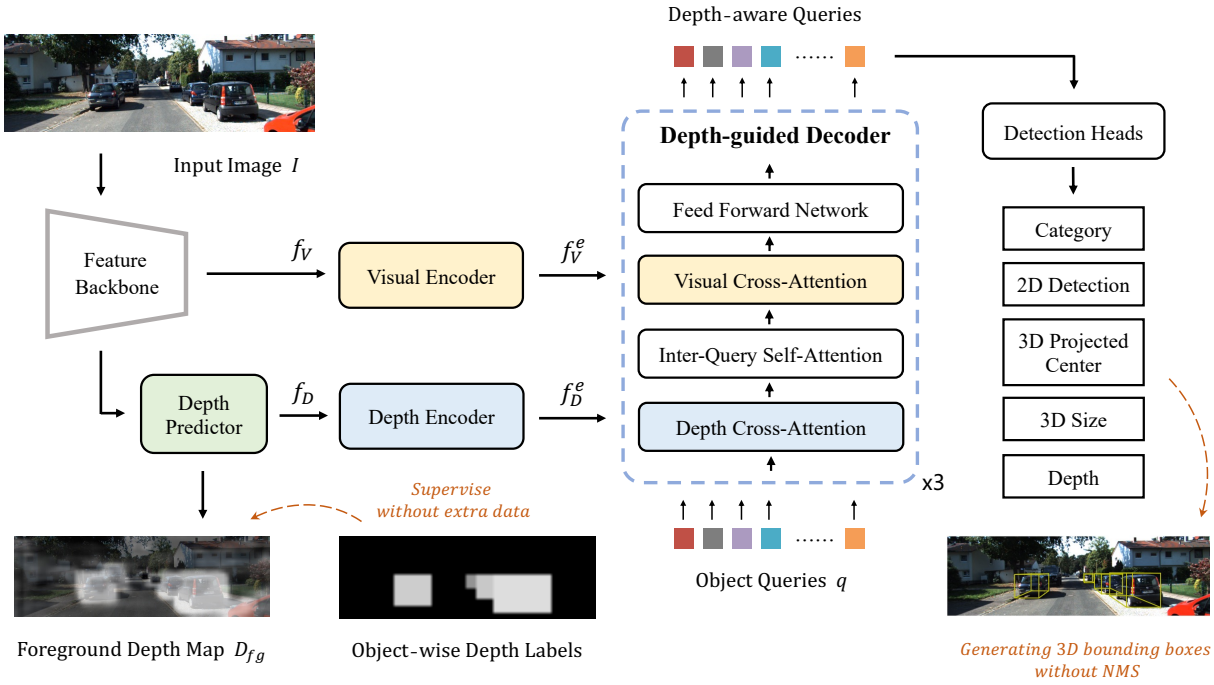

Figura 2.6: Arquitectura de MonoDETR [25].

de características, el uso de self-attention, el encoder-decoder y las cabezas de detección, para lograr un sistema de detección de objetos completo. La versión mas actual de esta arquitectura es SSD-MonoDETR [30], publicada en julio del año 2023, proponiendo un nuevo módulo SSDA (Scale-aware Deformable Attention) utilizado en el decoder. En la Tabla 2.3 se presentan los resultados proporcionados por los autores, concretamente evaluando los objetos de la clase *Car* sobre el split de validación de KITTI. Normalmente la dimensión de altura no aporta demasiada información para caracterizar los elementos del entorno en conducción autónoma, por este motivo se suelen proporcionar métricas como el AP BEV que solo tienen en cuenta las otras dos dimensiones en vista de pájaro en lugar de las tres como sucede en AP 3D.

Como era de esperar, los resultados caen considerablemente respecto a sistemas que utilizan LiDAR o información estéreo, aunque se encuentran en el estado del arte de detección monocular.

	BEV ↑			**3D ↑**		
	Easy	Med	Hard	Easy	Med	Hard
MonoDETR	37.86	26.95	22.80	28.84	20.61	16.38
SSD-MonoDETR	38.00	29.44	26.94	29.53	21.96	18.20

Tabla 2.3: Tabla de rendimiento de MonoDETR y SSD-MonoDETR sobre el split de validación de KITTI (AP40 con IoU de 0.7) para la clase *Car* [30].

2.3.2 SMOKE

SMOKE o Single-Stage Monocular 3D Object Detection via Keypoint Estimation [11] tiene como objetivo estimar las bounding boxes 3D utilizando una única etapa. Eliminando etapas adicionales como la generación de propuestas sobre la imagen se consigue una red que puede

lograr tiempos de inferencia bastante reducidos, del orden de 30-50 ms, lo cual la convierte en una buena candidata para sistemas en tiempo real como es el caso de un vehículo autónomo.

	BEV ↑			**3D ↑**		
	Easy	Med	Hard	Easy	Med	Hard
SMOKE	19.99	15.61	15.28	14.76	12.85	11.50

Tabla 2.4: Tabla de rendimiento de SMOKE (AP11 con un IoU de 0.7) sobre el split de validación de KITTI para la clase *Car*.

Como se refleja en la Figura 2.7, el backbone para extracción de características de la red es una DLA-34 [31], compuesta por 34 capas que hacen uso de Deep Layer Aggregation para combinar los mapas de características de forma eficiente, obteniendo un mapa 4 veces mas pequeño que la imagen de entrada. Este mapa pasa a ser procesado por dos ramas diferentes, las cuales se centran en conseguir la localización y las características de los objetos. A partir de la información generada por las ramas se puede calcular cada detección en el 3D.

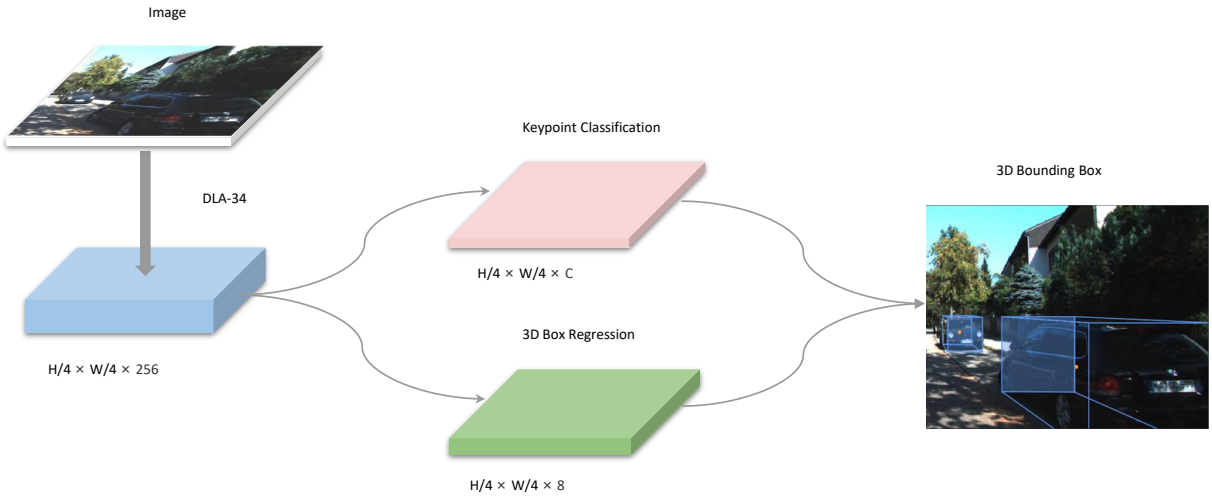

Figura 2.7: Arquitectura de SMOKE [11].

Los autores además realizan un estudio del error en distancia entre la detección y la etiqueta original del objeto (Figura 2.8). Se puede observar como a partir de 30 metros, los errores empiezan a superar los 2 metros, que en el caso de objetos de menor tamaño como peatones, puede suponer un error muy elevado en relación al tamaño del propio objeto detectado. Esto se ve reflejado en los resultados cuantitativos sobre KITTI de la Tabla 2.4, teniendo en cuenta que se han calculado con un IoU de 0,7.

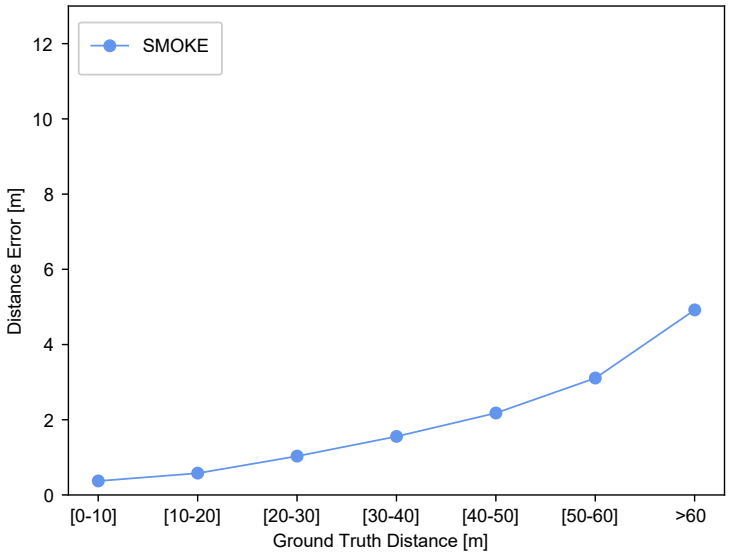

Figura 2.8: Error de detección de SMOKE [11].

2.3.3 FCOS3D

FCOS3D o Fully Convolutional One-Stage Monocular 3D Object Detection [27] presenta la arquitectura de la Figura 2.9 para realizar una detección 3D en una única etapa. Los autores buscan abordar varios de los problemas de los detectores monoculares planteando una reformulación de la forma de predecir los objetos centrada en las posiciones 3D de los centros.

El detector 3D propuesto utiliza una ResNet101 como backbone extractor de características y un neck basado en Feature Pyramid Network [32]. El método se basa en un enfoque sin anchors, siguiendo el principio de CenterNet [33], lo que significa que no se utilizan bounding boxes predefinidas como referencia para proponer detecciones, sino que se basa directamente en la distancia a los centros de los puntos en primer plano en la imagen. Esto se refleja en el valor de salida *Centerness (c)*, que sirve como un score para identificar las mejores detecciones.

El detector utiliza varias shared heads, donde hay 4 bloques convolucionales compartidos y pequeñas cabezas para cada objetivo. La red es capaz de generar el offset del centro del objeto al punto en primer plano $(\delta x, \delta y)$, la profundidad d, las dimensiones del objeto, la rotación, la clase y el score de centerness.

Los autores proporcionan métricas sobre el split de validación de nuScenes [34], las cuales se reflejan en la Tabla 2.5 junto a los valores obtenidos en el conjunto de test, tomados del benchmark oficial.

	mAP↑	**NDS↑**
CenterNet	30.6	32.8
FCOS3D (val)	34.3	41.5
FCOS3D (test)	35.8	42.8

Tabla 2.5: Tabla de rendimiento de FCOS3D sobre nuScenes.

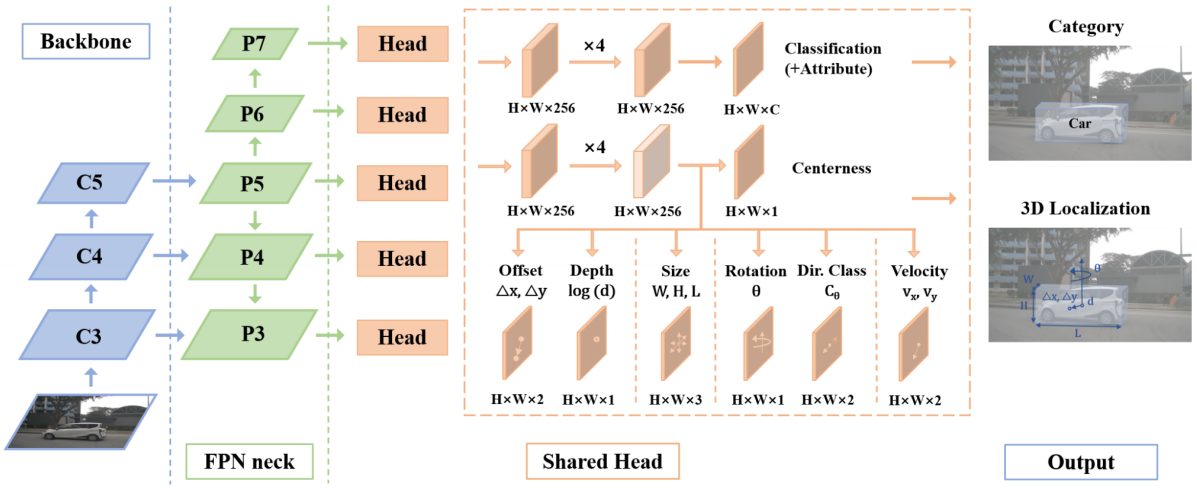

Figura 2.9: Arquitectura de FCOS3D [27].

2.3.4 PGD

La arquitectura de FCOS3D es mejorada en el detector PGD o Probabilistic and Geometric Depth [12]. Como se observa en la Figura 2.10, los autores proponen la introducción de un nuevo módulo para la estimación del valor de profundidad d de los objetos, argumentando que es uno de los factores mas limitantes a la hora de mejorar las detecciones.

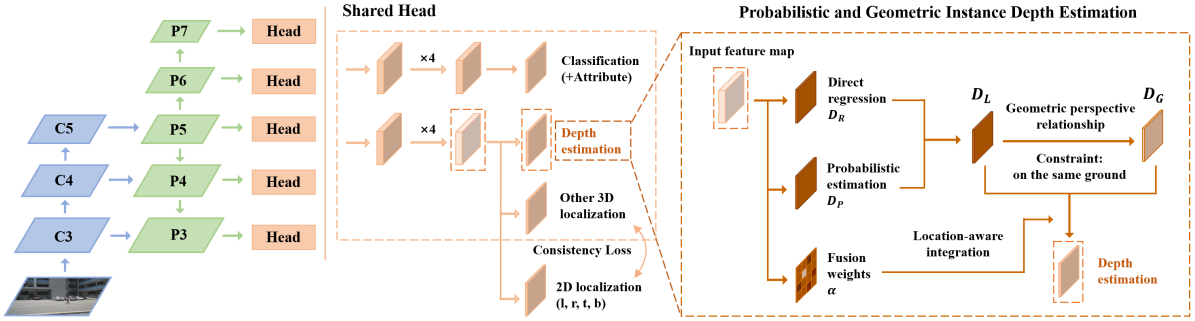

Figura 2.10: Arquitectura de PGD [12].

Respecto al modelo anterior, se mantiene la estructura de extracción de características y el resto de cabezas de la red, incorporando nuevas mejoras en el proceso de regresión del valor de profundidad. En lugar de realizar una regresión directa del valor d, se lleva a cabo una representación probabilística de la profundidad generada en cada objeto para reflejar la incertidumbre asociada. Posteriormente se construye un grafo entre los objetos para proporcionar contexto a la escena. Finalmente, utilizando estos nuevos datos y la información del grafo, se genera una nueva estimación del valor d. Ambas calculos se pueden observar de forma visual en la Figura 2.11 .

Las métricas del detector reflejadas en la Tabla 2.6 corroboran que efectivamente los cambios introducidos respecto a FCOS3D consiguen mejorar las detecciones. Por ejemplo, en nuScenes se logran mejoras de mAP de +2,6 en el split de validación y +3,1 en el split de test.

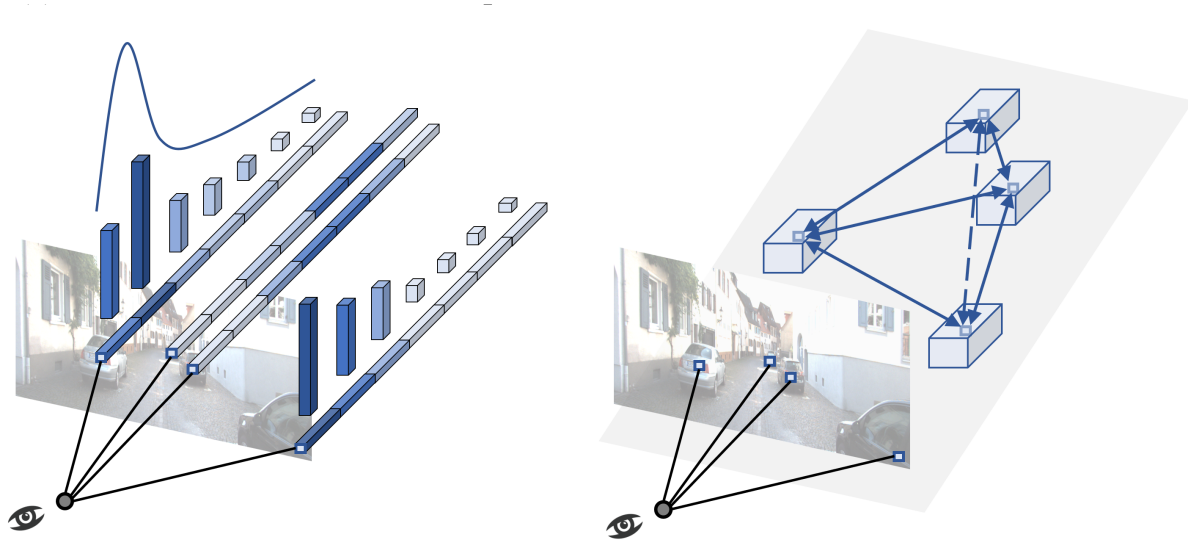

Figura 2.11: Esquema de los datos extra generados para el calculo de la profundidad [12]. a) Representación probabilística de d para cada objeto. b) Grafo de contexto de la escena.

	mAP ↑	**NDS ↑**
PGD(val)	36.9	32.8
PGD (test)	38.6	44.8

Tabla 2.6: Tabla de rendimiento de PGD sobre nuScenes.

Los autores proporcionan además métricas sobre el dataset KITTI (Tabla 2.7), consiguendo mejores resultados que SMOKE.

	BEV ↑			**3D ↑**		
	Easy	Med	Hard	Easy	Med	Hard
PGD	30.56	23.67	20.84	24.35	18.34	16.90

Tabla 2.7: Tabla de rendimiento de PGD (AP40 con IoU de 0.7) sobre el split de validación de KITTI para la clase *Car*.

2.4 Detección estéreo modular: estimación de profundidad

Uno de los principales problemas de la cámara es la pérdida de información tridimensional que se produce al capturar las imágenes. Por esta razón, las arquitecturas modulares que utilizan únicamente datos procedentes de cámara normalmente necesitan una etapa que se encargue de recuperar parte de la información perdida. Una práctica muy extendida en el SOTA es el uso de sistemas de estimación de profundidad para obtener parte de la información tridimensional. Este enfoque permite generar un mapa de profundidad del mismo tamaño que la imagen de entrada, proporcionando valores de distancia a cada píxel de la imagen, pudiendo utilizar esta información, por ejemplo, para obtener una nube de puntos para detectar sobre ella.

Método	EPE px ↓	Tiempo de procesado (s) ↓
HITNet XL	0.36	0.114
HITNet L	0.43	0.054
EdgeStereo [35]	0.74	0.32
LEAStereo [36]	0.78	0.3
GA-Net [37]	0.84	1.6
PSMNet [38]	1.09	0.41
StereoNet [39]	1.1	0.015
CoEx [14]	0.69	0.027
SDN[13]	-	0.2

Tabla 2.8: Tabla comparativa de HITNet con otros métodos de estimación de disparidad en Scene Flow [14] [40].

Si bien este proceso se puede realizar tanto en cámaras monoculares, con modelos como Monodepth [41] o SQLDepth [42], como con cámaras estéreo con modelos como los indicados en la Tabla 2.8. En este trabajo se pone el foco sobre estas últimas por su mejor resultado y por la disponibilidad de cámaras de este tipo en el vehículo del grupo de investigación. En el caso de la arquitectura propuesta en este trabajo (explicada en el Capítulo 4), se hace uso de una etapa de estimación de profundidad estéreo de la zona visible, luego en este apartado se busca hacer un análisis de arquitecturas del SOTA para comprobar el rendimiento que presentan.

2.4.1 Estimación por geometría

El método mas inmediato para conseguir información de profundidad con una cámara estéreo es el uso de la geometría y parámetros de las cámaras directamente.

A partir de las dos imágenes se calcula el mapa de disparidad, el cual refleja la diferencia entre un píxel en uno de los focos y su equivalente en el otro foco. A cada valor de este mapa se le aplica la Formula 2.3, en la que es necesario conocer la distancia focal en el eje horizontal de la cámara y la distancia entre focos o línea base.

$$Z(u,v) = \frac{f_x \cdot B}{D(u,v)} \tag{2.3}$$

Tomando los valores del dataset KITTI, se obtiene una relación entre disparidad y profundidad como la de la Figura 2.12. Se puede observar como para valores de disparidad pequeños (distancias grandes) se cuenta con una menor resolución en distancia, agrupando valores de entre 20 y 80 metros entre disparidad de 5 y 20. Esto puede suponer un incremento muy significativo del error de la estimación a distancias medias (30-40 metros), lo cual puede ser inasequible en conducción autónoma. Por esta razón se recurren a sistemas de DL como los explicados posteriormente en este apartado.

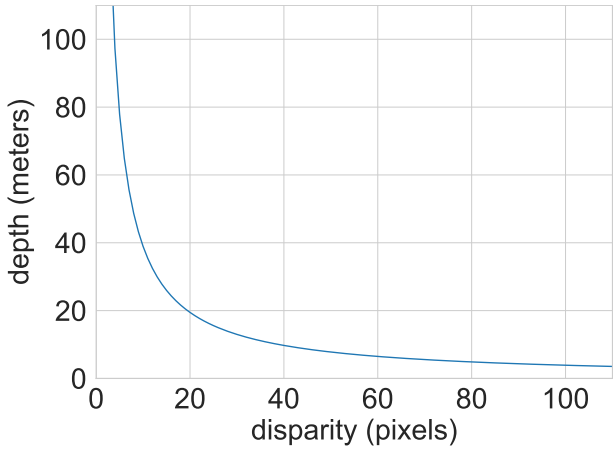

Figura 2.12: Relación entre la disparidad de píxeles y la profundidad asociada para la configuración de cámara de KITTI [13].

2.4.2 Stereo Depth Netowrk: SDN

El modelo SDN o Stereo Depth Network [13], representado en la Figura 2.13, fue propuesto para abordar y mejorar la estimación de profundidad en conducción autónoma. En primer lugar se extraen las características de ambas imágenes con capas con los mismos pesos para cada una. Muchos de los modelos de estimación de profundidad generan a su salida un mapa de disparidad, que posteriormente hay que transformar. En esta red, utilizando ambos mapas de características se construye el volumen de disparidad y se transforma a profundidad. Los autores de SDN afirman que hacer que la red genere directamente este valor lleva a reducir el error.

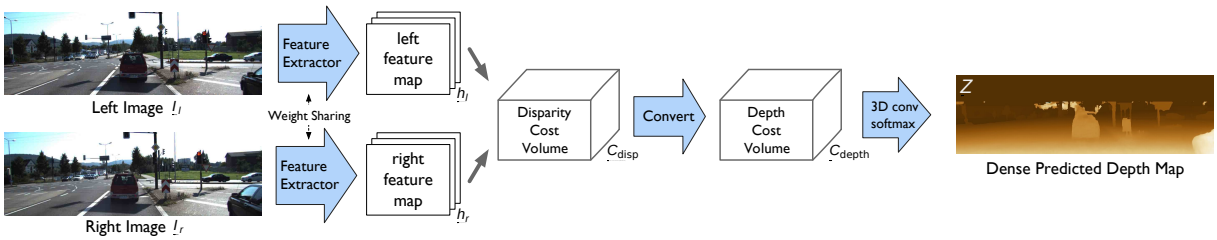

Figura 2.13: Esquema de SDN [13].

Además del propio modelo de estimación, los autores proponen una arquitectura que permite rectificar los valores de profundidad con datos de otros sensores, y además consigue detectar objetos a partir de esta información utilizando un detector de nube de puntos. Parte de esa arquitectura se utilizará mas adelante, por lo que se explica mas detalladamente junto a los cambios propuestos en el Capítulo 4. Por este motivo, los autores se centran en proporcionar métricas de detección de objetos como las de la Tabla 2.9, en las que se ve una clara mejora respecto a los sistemas end-to-end monoculares, a costa de incrementar el tiempo de procesado.

En las Tablas 2.10 y 2.11 se representa el rendimiento en función de la distancia, observando como el error en la profundidad aumenta considerablemente con la misma, especialmente a partir de 30 metros. Esto se traduce en una reducción de la precisión para detectar objetos considerable.

	BEV ↑			3D ↑			Tiempo(s) ↓
	Easy	Med	Hard	Easy	Med	Hard	
SDN	82.0	64.0	57.3	67.9	50.1	45.3	0.2
SDN + rectificación	88.2	76.9	73.4	75.1	63.8	57.4	0.3

Tabla 2.9: Resultados de la clase *Car* sobre el split de validación de KITTI [13]. El detector 3D de nube de puntos usado es P-RCNN [43].

	0-30 m		30-50 m	
Entrada	**BEV ↑**	**3D ↑**	**BEV ↑**	**3D ↑**
SDN	68.6	56.7	27.4	11.3
SDN + rectificación	84.7	67.8	49.9	31.5
LiDAR	88.5	84.0	69.9	51.5

Tabla 2.10: Rendimiento de SDN en la clase *Car* sobre el split de validación de KITTI en función de la distancia [13]. El detector 3D es P-RCNN [43].

	Rango (m)					
	0-10	**10-20**	**20-30**	**30-40**	**40-50**	**50-60**
SDN	0.21±0.89	0.35±1.16	0.87±2.31	1.80±4.22	2.67±6.00	4.27±8.78
SDN + rect.	0.21±0.90	0.35±1.17	0.84±2.34	1.74±4.27	2.59±6.06	4.14±8.85

Tabla 2.11: Error medio y desviación estándar en función de la distancia estimada por la SDN.

2.4.3 Correlate and Excite: CoEx

Al igual que el modelo presentado en el apartado anterior, COEX o Correlate and Excite [14], busca lograr la estimación de profundidad a partir de la información de una cámara estéreo y en una única etapa. En este caso, los autores buscan mejorar métodos que utilicen volúmenes de costes (como la SDN) utilizando mapas de características extraídos de imágenes mediante la técnica propuesta Guided Cost Volume Excitation (GCE), método el cual plasman en la arquitectura de la Figura 2.14 de su nuevo estimador. Para la extracción de características se utiliza una arquitectura MobileNetV2 [44] debido a su rapidez, compartiendo pesos para la imagen izquierda y la derecha. A continuación se aplica un módulo basado en U-Net [45] para obtener estos mapas a diferentes escalas y obtener el volumen de coste utilizando una capa de correlación. Los mapas de características de las imágenes pasan a utilizarse para resaltar las propiedades geométricas importantes en la segunda parte donde se aplican las convoluciones 3D.

En la Tabla 2.12 se representan los resultados proporcionados por los autores, calculando la media de error absoluto de la disparidad para todos los píxeles en el dataset Scene-Flow [46].

En capítulos posteriores se realizarán mas pruebas para calcular métricas de profundidad directamente, las cuales proporcionan resultados mas orientativos para el caso de estudio.

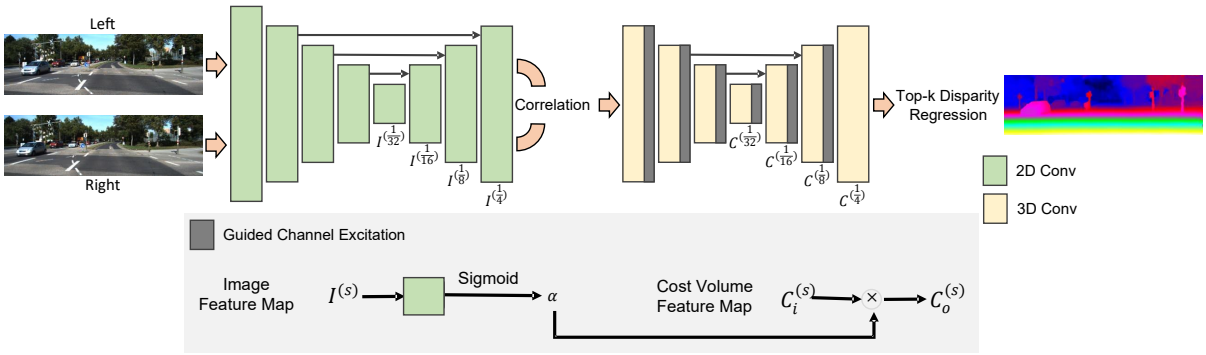

Figura 2.14: Esquema de COEX [14].

	Scene-Flow EPE (px) ↓	Tiempo (s) ↓
StereoNet	1.101	0.015
DispNetCorr	1.68	0.060
CoEx	0.69	0.027

Tabla 2.12: Comparación sobre Scene-Flow de varios modelos de estimación de disparidad [14].

2.4.4 HITNet

HITNet o Hierarchical Iterative Tile Refinement Network [40] fue publicada a principios de 2023 por Google, proponiendo una arquitectura enfocada en obtener unos bajos tiempos de procesado. En lugar de construir volúmenes y convoluciones 3D como aplicaban SDN y CoEx, en este modelo utiliza los mapas de características a múltiples escalas para construir el mapa de disparidad. Para extraer las características se adopta una arquitectura basada en U-Net [45], de la cual se extraen las características a diferentes escalas en la parte de upsampling del modelo o decoder. En cada escala se genera una hipótesis de disparidad en el paso que denominan inicialización, y posteriormente se elige de entre todas la que menos coste presente, la cual se refina en la denominada etapa de propagación.

En la Tabla 2.8 se reflejan los buenos resultados en Scene Flow, superando a otros modelos como CoEx, explicado en el apartado anterior.

Si bien el modelo es prometedor, a fecha de la escritura del documento Google no ha puesto a disposición de los usuarios código para manejarlo ni configuración del propio modelo, lo que dificulta los intentos de reproducción y personalización. Durante el desarrollo del trabajo se ha utilizado el código proporcionado para hacer inferencia en los datos utilizados en los experimentos, obteniendo tiempos y resultados acordes a los presentados por los autores. Además, se han repetido pruebas con réplicas de código abierto de este modelo, sin embargo, los resultados y tiempos de procesado son mucho peores a los especificados en la publicación original.

Debido a que no es posible realizar los entrenamientos que se plantean en este trabajo sobre este modelo, tras realizar las pruebas que se han comentado anteriormente, se decide no utilizar este modelo.

Capítulo 3

Herramientas

El aspecto más triste de la vida es que la ciencia reúne el conocimiento más rápidamente que la sociedad la sabiduría.

Isaac Asimov

3.1 NVIDIA CUDA y PyTorch

Uno de los motivos del gran avance en IA en la última década ha sido la aparición y evolución de las tarjetas gráficas GPU de NVIDIA, las cuales han permitido multiplicar la capacidad de cálculo en paralelo. Para facilitar el acceso y uso de las capacidades de las GPU, NVIDIA pone a disposición de los desarrolladores la plataforma CUDA, sobre la que se ejecutan una gran cantidad de librerías de procesamiento paralelizado para diversas áreas y lenguajes de programación.

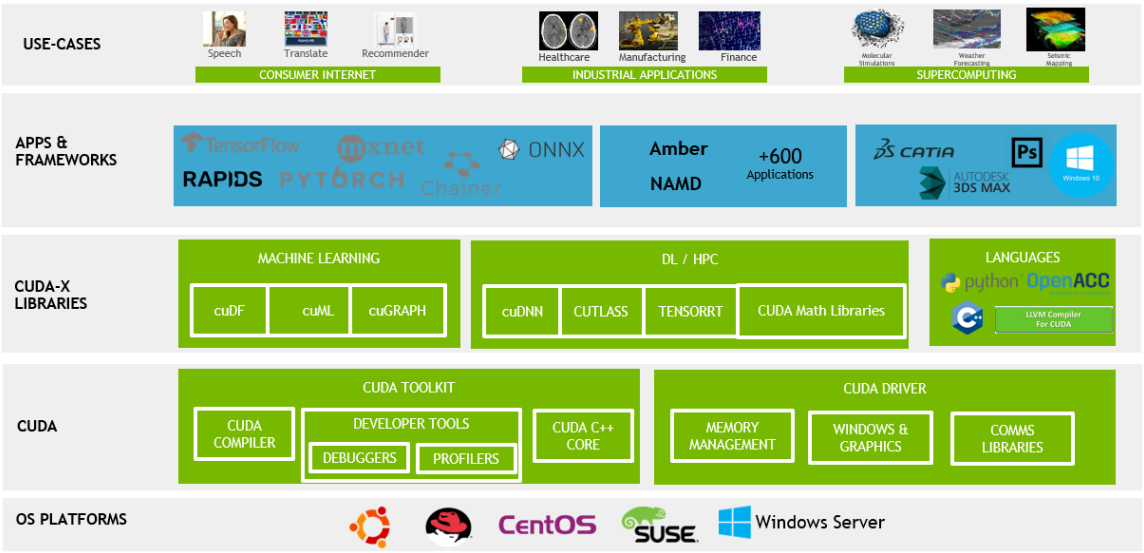

Figura 3.1: Esquema de las diferentes capas que interactúan con NVIDIA CUDA [47].

En este trabajo concretamente son de interés los frameworks de DL que sean compatibles con nuestro código Python y además utilicen CUDA. Las dos grandes librerías utilizadas actualmente son PyTorch [48] y TensorFlow [49], de las que se decide utilizar la primera por su gran aumento de popularidad en los últimos años y al hecho de que muchos de los principales frameworks de DL se basan en ella.

3.2 Docker

La herramienta Docker [50] permite el encapsulamiento de software en unidades denominadas *contenedores*. Estos contenedores se caracterizan por incluir todo lo necesario para la ejecución de la aplicación desplegada, consiguiendo independizar prácticamente por completo el entorno de la aplicación y del sistema operativo del host donde se ejecuta. Para lograr una gran escalabilidad y facilidad de despliegue, los contenedores se generan a partir de *imágenes*, las cuales representan un estado fijo del entorno que se quiere ejecutar. Estas imágenes se pueden almacenar en *registros* centralizados para tener acceso desde cualquier dispositivo, y pueden dar lugar a la cantidad de contenedores que se desee.

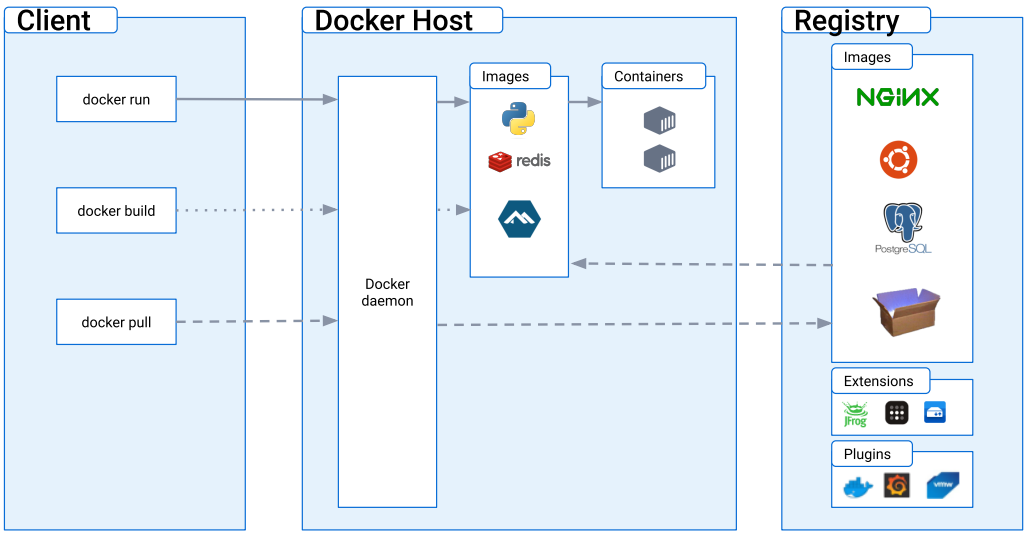

Figura 3.2: Arquitectura de la herramienta Docker [50].

En el caso de este trabajo, Docker se utiliza fundamentalmente para construir el entorno de ejecución del código Python desarrollado y las librerías y frameworks específicas para DL. La imagen utilizada como base es la proporcionada por NVIDIA en su registro propio, contando con una instalación optimizada de elementos tan importantes como PyTorch y CUDA. Gracias a esto, el entorno sería directamente trasladable a la infraestructura del vehículo real del grupo una vez realizadas las pruebas correspondientes al sistema de detección.

3.3 MMDetection3D

MMDetection3D [51] es un framework sobre PyTorch de código abierto creado por OpenMMLab para el entrenamiento, evaluación y despliegue de modelos de DL centrado en modelos de detección tridimensional con cámara y LiDAR. En la Figura 3.3 se representa la estructura simplificada de la librería con otros componentes de OpenMMLab:

- **MMEngine y MMCV**: son las librerías requeridas para el uso de los frameworks de OpenMMLab de visión artificial.
- **Archivos de configuración**: permiten definir detalles del detector a utilizar como el modelo, configuración de entrenamiento o preprocesado de los datos.
- **API**: permite utilizar las diferentes características de la librería a través de un archivo de configuración. Los creadores proporcionan unos archivos que permiten entrenar y hacer inferencia con modelos estándar de la librería de una forma ágil, sin embargo, se centran en realizar detecciones sobre archivos almacenados en disco previamente, no en arquitecturas en tiempo real. Otra forma de utilizar MMDetection3D es acceder directamente a la API a través de nuestro código, lo cual es lo recomendable en el caso de buscar ejecuciones en tiempo real como es el caso de nuestra arquitectura.

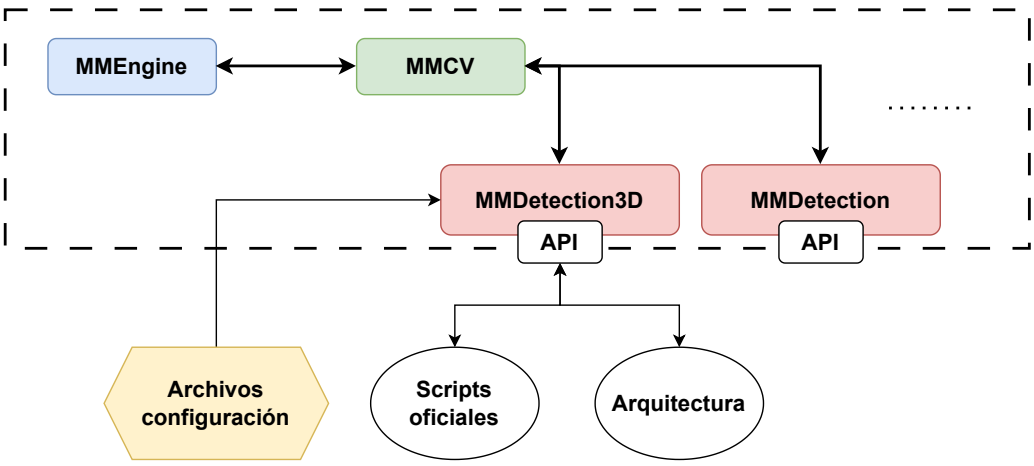

Figura 3.3: Estructura de funcionamiento de MMDetection3D.

La elección de este framework ha sido influenciada significativamente tanto por la amplia variedad de modelos disponibles para los usuarios, como por la agilidad que ofrece para entrenar y evaluar los modelos que son objeto de estudio en este trabajo.

3.4 KITTI dataset

Si bien existen multitud de bases de datos centradas en detección de objetos sobre imágenes o nubes de puntos, para este estudio se necesitan datos procedentes de entornos de conducción. Con esto en mente se ha seleccionado el dataset KITTI [1] para realizar los entrenamientos y evaluaciones, la cual ha sido un referente desde su publicación en 2012.

De todas las diferentes categorías disponibles, se utilizan los datos pertenecientes a la categoría de *Detección 3D*. Concretamente, se necesitan los datos de la cámara estéreo y las nubes del LiDAR, de las que se genera el ground-truth de profundidad. Los detalles de los sensores son los siguientes, sabiendo que su configuración es la de la Figura 3.4:

- **Cámara estéreo**: conformada por dos cámaras Point Grey Flea 2 (FL2-14S3C-C) de 1.4 Megapíxeles distanciadas 0.54 metros. Las imágenes capturadas tienen una resolución de 1382×512 píxeles, que al aplicar rectificación dan lugar a imágenes de 1242×375 píxeles.
- **LiDAR**: Velodyne HDL-64E que proporciona nubes con una resolución vertical de 64 a una tasa de 10 Hz.

Al trabajar con información en diferentes sistemas de coordenadas, es importante conocer las posiciones de los sensores y transformaciones entre sus diferentes ejes de referencia (Figura 3.4). La cámara concretamente también cuenta con parámetros intrínsecos, los cuales permiten transformar valores entre la imagen y el sistema 3D. Estos parámetros son las distancias focales de cada eje (f_x, f_y), el centro óptico (c_x, c_y) y un factor de corrección (s). Normalmente se agrupan en una matriz intrínseca, de tal forma que las transformaciones pasan a ser multiplicación de matrices.

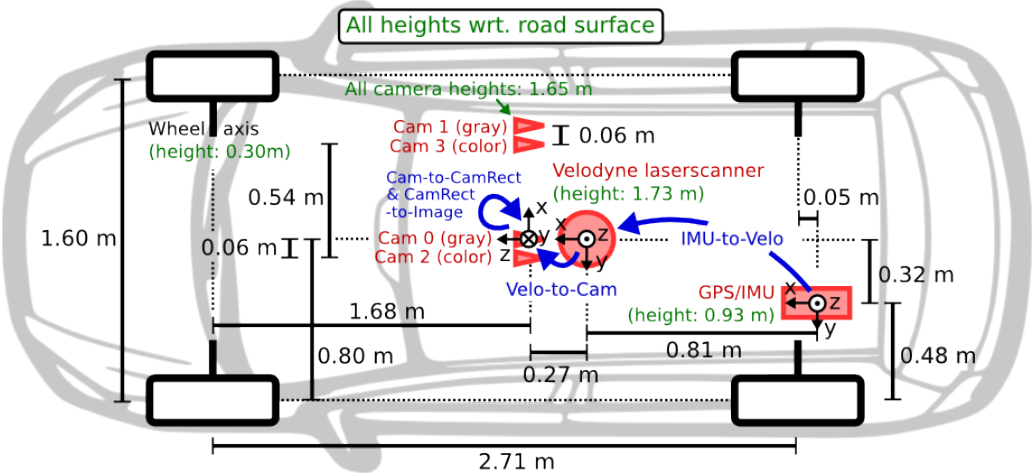

Figura 3.4: Esquema de configuración de los sensores de KITTI [1].

Además de los datos de los diferentes sensores, KITTI proporciona información sobre los objetos de la escena para poder entrenar los diferentes sistemas de detección. Cada elemento lleva asociada la siguiente información detallada:

1. **Type**: un string que especifica a qué tipo de objeto se refiere. Los tipos más comunes son 'Car', 'Pedestrian' y 'Cyclist'.

2. **Truncated**: un número float de 0 a 1 que especifica si el objeto en cuestión se sale del límite de la imagen.

3. **Occluded**: indica si el objeto está oculto.

4. **Alpha**: ángulo de observación del objeto, que se corresponde con el ángulo que forma la línea que une la cámara con el centroide del objeto con el eje X en coordenadas cámara.

5. **Bounding Box**: cuatro valores de píxeles que definen left, top, right, bottom de la bounding box en 2D de cada objeto.

6. **Dimensions**: tres valores que especifican las dimensiones tridimensionales del objeto (altura, anchura y longitud).

7. **Location**: tres valores que definen la posición xyz del centroide del objeto.

8. **Rotation_y**: valor del ángulo que forma la orientación del objeto con el eje X en coordenadas de la cámara.

En total se han utilizado 7481 imágenes estéreo con sus correspondientes nubes de puntos y el ground-truth de profundidad generado, haciendo una división de 50/50 para conjunto de entrenamiento y validación. En este caso la información de profundidad procede directamente del sensor LiDAR, el cual proporciona nubes de puntos precisas, las cuales se proyectan sobre la cámara para obtener un ground-truth como el de la Figura 3.5.

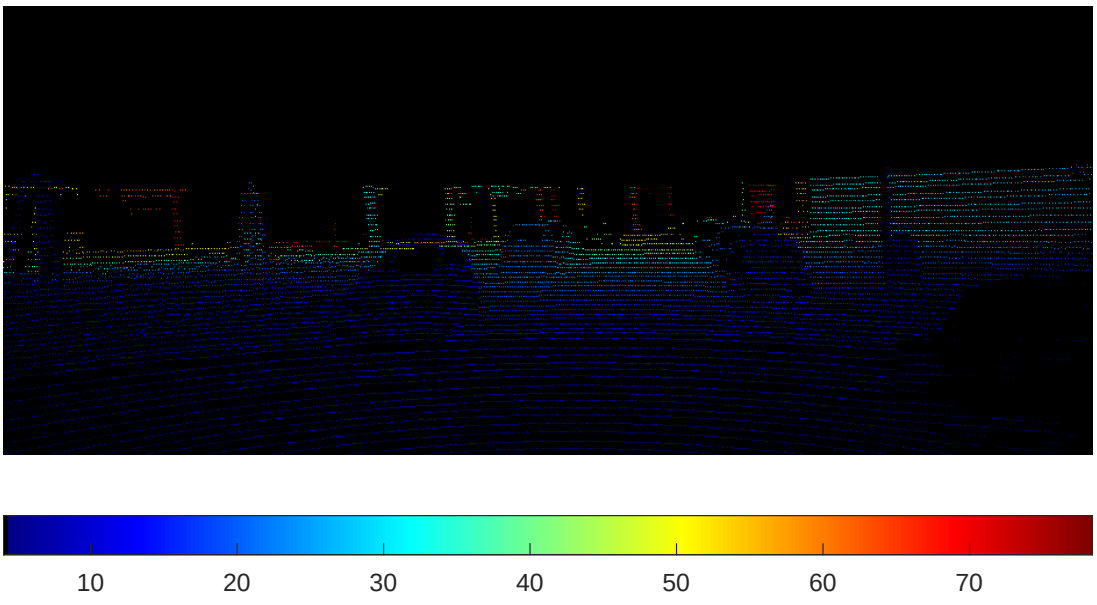

Figura 3.5: Datos de profundidad generados en KITTI.

Toda la información mencionada anteriormente se proporciona siguiendo la estructura de la Figura 3.6, en la que las carpetas *image_2* e *image_3* se corresponden con los datos de la cámara estéreo a color, *label_2* son las etiquetas de los objetos en el sistema de coordenadas de la cámara, *velodyne* son las nubes de puntos asociadas a cada imagen y *calib* contiene los ficheros con las matrices de parámetros de los sensores.

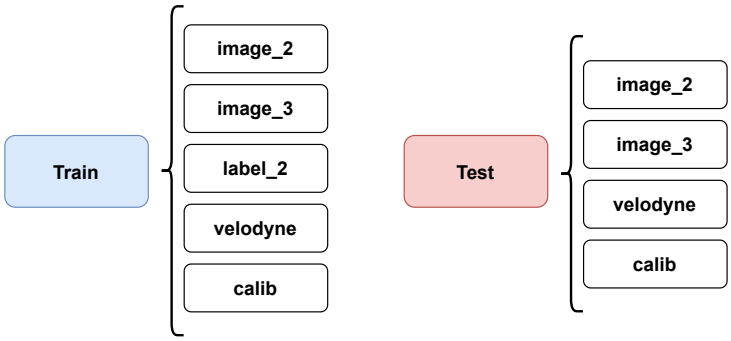

Figura 3.6: Estructura de archivos de KITTI [1].

En la Figura 3.7 se ha representado un ejemplo de la información proporcionada para un frame del dataset. En la parte superior se representa la imagen obtenida por el foco izquierdo de la cámara estéreo, mientras que en la parte inferior aparece la nube de puntos y las etiquetas de los objetos observadas en vista de pájaro.

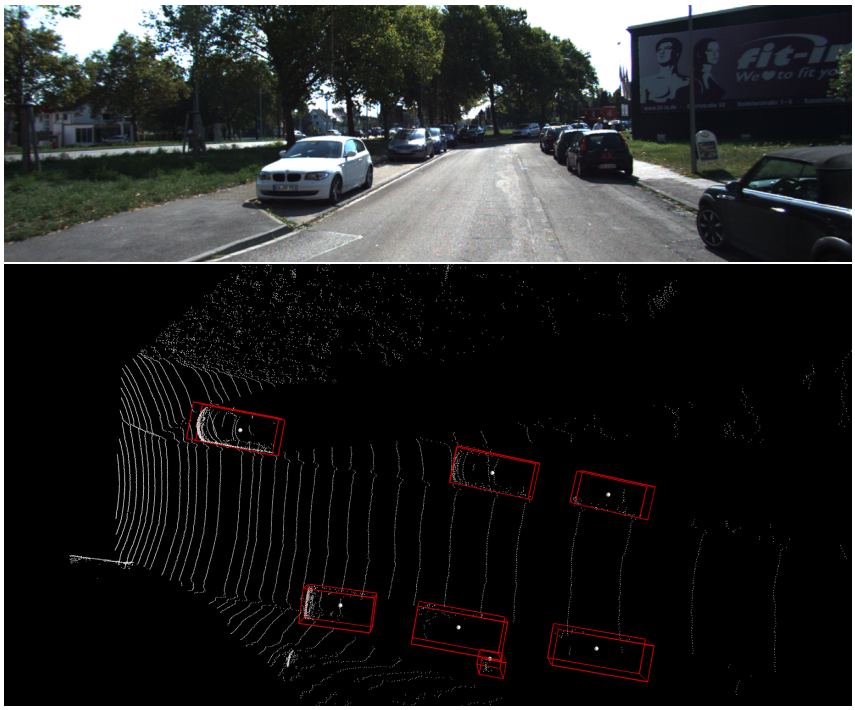

Figura 3.7: Ejemplo de datos proporcionados por KITTI: a) Imagen RGB b) Nube de puntos de LiDAR y etiquetas.

3.5 Simulador CARLA y SHIFT dataset

El uso de entornos de simulación es una etapa clave en el desarrollo de sistemas autónomos, permitiendo probar y evaluar arquitecturas de conducción en un entorno controlado, pudiendo variar elementos del tráfico y ambientales, forzando condiciones donde el sistema podría fallar. Sobre esta idea surgió el simulador de código abierto CARLA [2], desarrollado sobre el motor Unreal Engine, ofrece una plataforma de pruebas inmersiva que combina detalles visuales que buscan el mayor parecido con la realidad, físicas realistas, una amplia gama de sensores simulados y la capacidad de generar y configurar *actores* que interactúan entre ellos y con la simulación. Como rasgos generales se pueden destacar las siguientes características, reflejadas en la Figura 3.8:

- **Arquitectura servidor-clientes**: permite la ejecución simultánea de nodos que controlen diferentes actores u otros elementos de la simulación.
- **ROS-Bridge**: el simulador puede conectarse a arquitecturas basadas en ROS, como es nuestro caso, para intercambiar información.

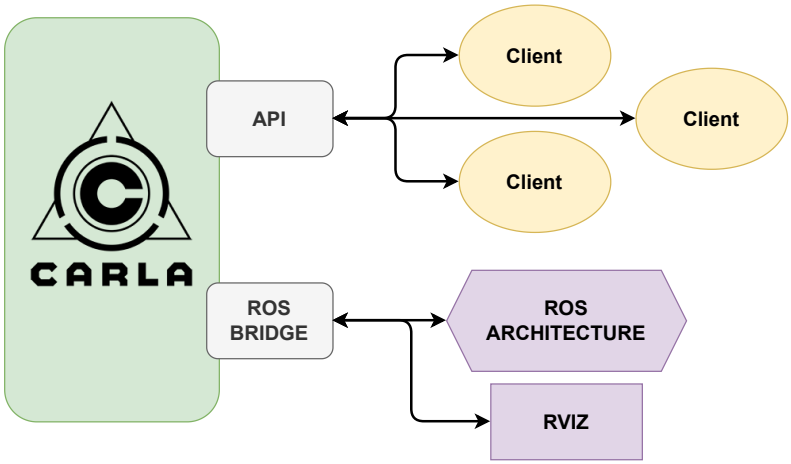

Figura 3.8: Esquema de conexión de CARLA.

- **API**: permite a los usuarios controlar todos los aspectos relacionados con la simulación en curso, desde el clima hasta la generación de tráfico.
- **Simulación de sensores**: el simulador permite introducir multitud de diferentes sensores habituales como cámara o LiDAR. Presentan bastante flexibilidad de configuración, sin embargo información que se base en variables físicas, como la intensidad del LiDAR o el doppler del radar, no se simulan correctamente. En contraparte CARLA ofrece datos que no se pueden extraer directamente de sensores reales como imágenes de profundidad completas o máscaras de segmentación semántica de la escena.
- **Generación de mapas**: a través de la herramienta RoadRunner de Mathworks [52] es posible construir mapas que se ajusten a los requerimientos o que simulen una zona real de forma precisa.

- **Generación de escenarios de simulación**: para realizar pruebas complejas, el usuario puede definir rutas y comportamientos de otros actores para simular situaciones reales.

Figura 3.9: Entorno dentro del mapa *Town10* de CARLA.

Debido a la importancia de la simulación en el desarrollo de una arquitectura, se ha decidido incorporar datos procedentes de CARLA en los experimentos para comprobar como se desenvuelven los diferentes sistemas tanto en entornos reales como en simulación. En lugar de obtener los datos en bruto del simulador, se ha decidido utilizar la base de datos SHIFT [3], la cual aprovecha la flexibilidad de CARLA para ofrecer casos que no suelen aparecen en datasets reales como condiciones climatológicas desfavorables (lluvia o niebla) o en horario nocturno.

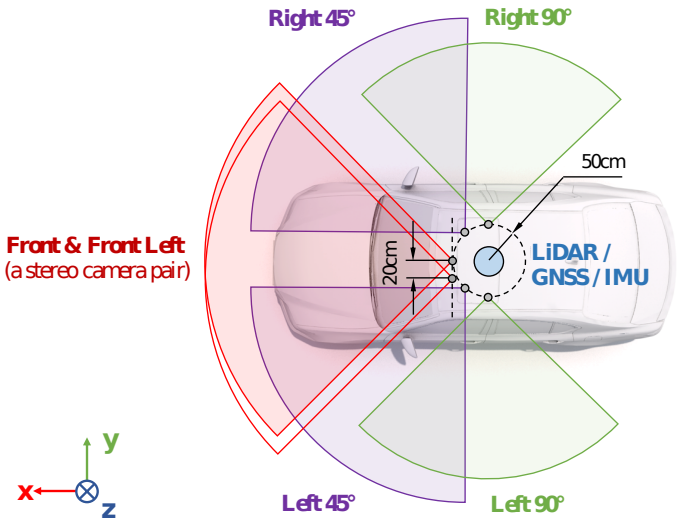

Figura 3.10: Esquema de configuración de los sensores de SHIFT [3].

En la figura 3.10 se puede ver la configuración de sensores del dataset en CARLA, contando con:

- Información visual procedente de un sistema multicámara que proporciona imágenes de 1280×800 píxeles y un FOV de 90^o. En este trabajo solo se utiliza la información de las cámaras frontales, las cuales forman una cámara estéreo con distancia entre focos de 20 centímetros.

- Sensor LiDAR con una resolución de 128 haces. Es importante tener en cuenta que la información del cuarto canal de la nube, que en un LiDAR real aporta información sobre la intensidad de la reflexión, no aporta la misma información, luego se debe evitar en caso de querer hacer adaptación de dominio de detectores 3D sobre nubes de puntos.

Adicionalmente, es posible obtener directamente una imagen de profundidad completa como la de la Figura 3.11 sin necesidad de la intervención de ningún sensor como sucede en KITTI. Estos datos densos se utilizan para entrenar los estimadores de profundidad de este trabajo y comprobar su funcionamiento en un entorno simulado y si su uso consigue ayudar al rendimiento en casos reales.

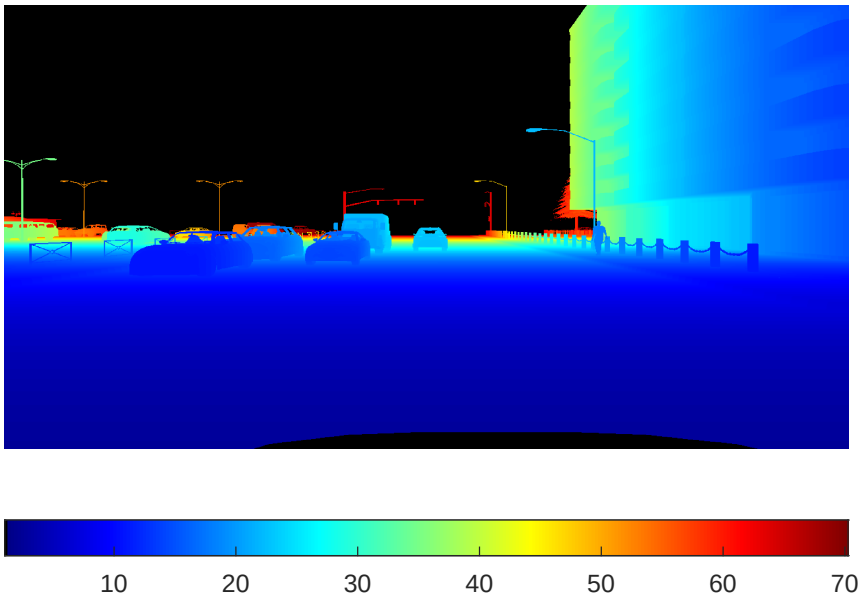

Figura 3.11: Datos de profundidad generados en SHIFT.

Es importante destacar que al contar con un número muy elevado de datos, los creadores de SHIFT utilizan un formato comprimido para proporcionarlos, los cuales siguen la estructura de la Figura 3.12. Los datos se encuentran divididos en tres splits, conteniendo datos para entrenamiento, validación y test. Para cada uno de los splits se proporcionan datos organizados por sensor y vista. Los datos de cada sensor se dividen en secuencias con frames, para los cuales se proporcionan los datos de imagen, información semántica, etc. De la misma forma las etiquetas de bounding boxes se almacenan en archivos dedicados a ello. Para poder trabajar con los datos de ambos datasets al mismo tiempo, se ha adaptado la estructura de archivos y la información de las etiquetas de SHIFT a un formato similar al de KITTI, el cual se ha detallado en el apartado anterior.

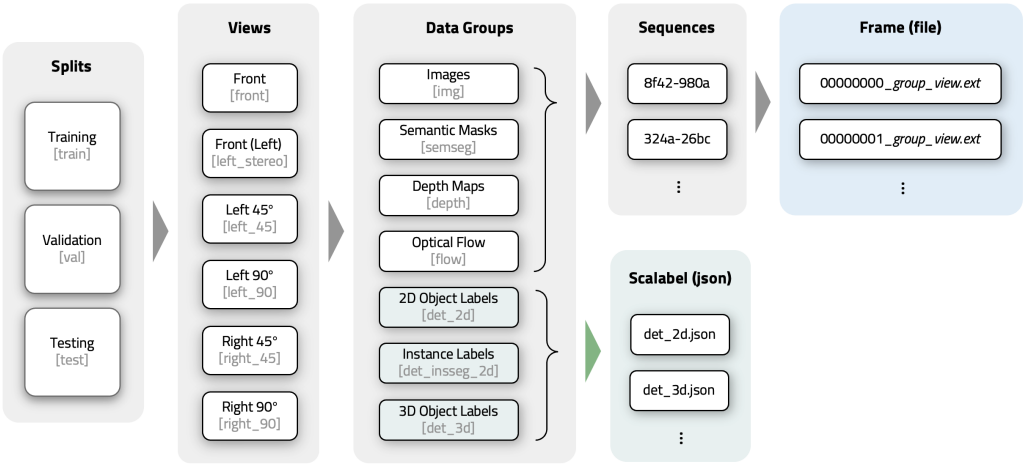

Figura 3.12: Esquema de datos del dataset SHIFT [3].

De todos los datos proporcionados, se utilizan 50 secuencias para obtener una cantidad de datos similar a los proporcionados por KITTI, 7500 imágenes, volviendo a realizar una división 50/50 para los conjuntos de entrenamiento y validación.

Figura 3.13: Ejemplo de datos proporcionados por SHIFT: a) Imagen RGB b) Nube de puntos de LiDAR y etiquetas.

Capítulo 4

Implementación

La vida es como andar en bicicleta. Para mantener el equilibrio, debe mantenerse en movimiento.

Albert Einstein

Tras realizar el estudio de técnicas presentes en el estado del arte en el capítulo 2, es necesario decidir cuales de ellas van a ser utilizadas para realizar los diferentes experimentos. En este capítulo se detallan las dos posibles arquitecturas que se plantean para conseguir las detecciones tridimensionales en conducción autónoma, así como los modelos de DL que se han seleccionado para utilizarse dentro de ellas.

4.1 Detección de objetos end-to-end

Como su propio nombre indica, trabajar con sistemas de detección end-to-end implica simplificar considerablemente la arquitectura de detección, consiguiendo los resultados en una única etapa. Esta etapa se refleja en la Figura 4.1, en la que a partir de una única imagen de entrada $I \in \mathbb{R}^{H \times W \times 3}$, se obtienen directamente las bounding boxes 3D de los objetos.

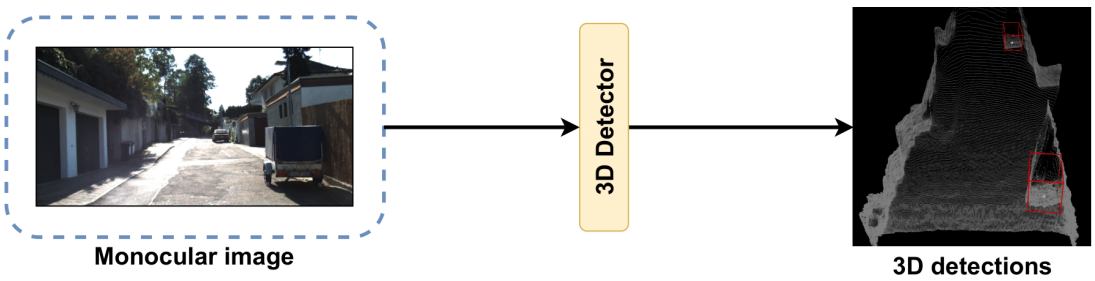

Figura 4.1: Esquema del sistema de detección end-to-end.

En el proceso de selección de modelos para el entrenamiento y evaluación, se ha seguido una doble consideración. En primer lugar, se ha dado prioridad a aquellos modelos que se

encuentran en MMDetection3D, lo que permite una integración y experimentos más sencillos y fluidos. Además, se ha puesto un énfasis particular en la velocidad de los modelos, un aspecto crucial en este contexto. Esta elección se alinea con la búsqueda de un rendimiento óptimo en términos de eficiencia, permitiendo una implementación eficaz y resultados en tiempo real.

Como se plasma en la Tabla 2.2 y en el desarrollo del Capítulo 2, SMOKE [11] y PGD [12] son las dos redes mas rápidas que además son compatibles con el framework de trabajo, luego son las elegidas para realizar los estudios end-to-end monoculares.

4.2 Sistema modular basado en profundidad para detección de objetos 3D

Si se utiliza un sistema de estimación de profundidad para conseguir la información 3D del entorno, es necesario construir un sistema mas complejo para lograr la detección de objetos. En este trabajo concretamente, se ha propuesto el sistema de la Figura 4.2, el cual es una modificación de las ideas planteadas por los creadores de la SDN en su publicación PseudoLidar++ [13]. A partir de un par de imágenes de una cámara estéreo $I \in \mathbb{R}^{2\times H\times W\times 3}$, se realiza una estimación de profundidad con una de las redes seleccionadas y una detección de objetos 2D con la YOLOv5m. Con esta información se genera una nube 3D similar a la que se obtendría con sensores como el LiDAR, filtrando las zonas de interés únicamente en el campo de visión de la cámara, sobre la cual se realiza una detección en la última etapa.

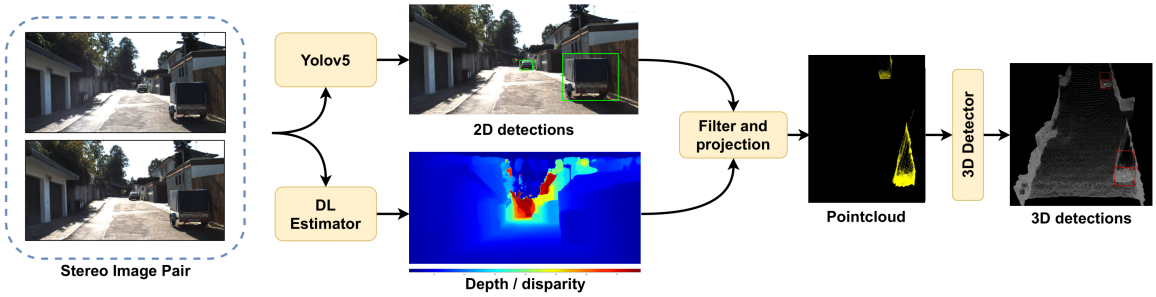

Figura 4.2: Esquema del sistema multietapa de detección basado en estimación de profundidad.

4.2.1 Detección 2D

Para detectar objetos en 2D, se ha optado por utilizar YOLOv5 debido a su versatilidad con sus múltiples versiones, buenos resultados y rapidez. Se utiliza la imagen del foco izquierdo, $I \in \mathbb{R}^{H\times W\times 3}$, como punto de partida y el modelo identifica sobre ella los objetos importantes para la conducción, como vehículos y peatones, sirviendo estas detecciones en la siguiente etapa del sistema (Figura 4.3).

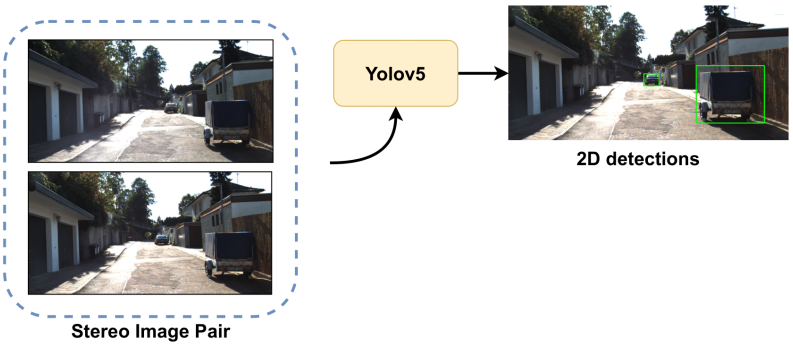

Figura 4.3: Esquema de detección 2D del sistema multietapa.

4.2.2 Estimación de profundidad

Además de la información semántica y localización de los objetos, se ha introducido la capa de estimación de profundidad necesaria para generar la información tridimensional de la escena. A partir del par de imágenes procedentes de la cámara estéreo $I \in \mathbb{R}^{2\times H\times W\times 3}$, la red genera un mapa del mismo tamaño $D \in \mathbb{R}^{H\times W}$, proporcionando un valor por píxel.

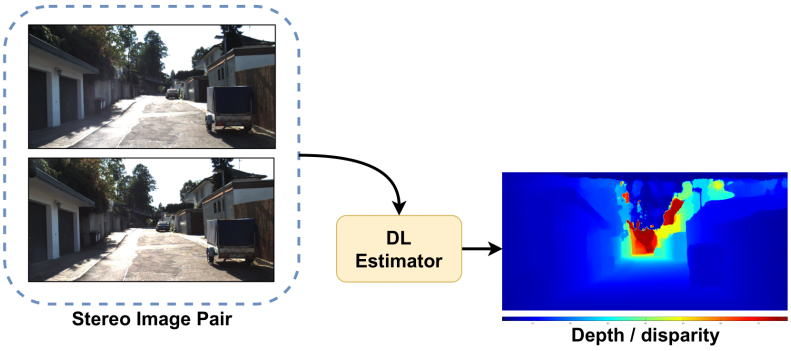

Figura 4.4: Esquema de estimación de profundidad del sistema multietapa.

La selección de los modelos a utilizar en esta etapa se basa en el análisis realizado en el Capítulo 2. HITNet [40] no se ha utilizado debido a que los autores no han publicado el código y detalles necesarios para replicar sus resultados y rendimiento. StereoNet [39] es la más rápida de la comparativa (Tabla 2.8), sin embargo, es la que mas error presenta, luego se descarta. Se decide finalmente utilizar CoEx [14] por su rapdez y buenos resultados respecto al resto de modelos, y SDN [13] por ser prácticamente igual de ligera y ser la propuesta original de los autores de PseudoLidar++, arquitectura de la que se parte en este trabajo, lo cual permite conocer la influencia de las modificaciones en la arquitectura en los resultados finales.

En caso específico de CoEx, la red genera valores de disparidad, luego es necesario hacer una transformación a profundidad siguiendo la Ecuación 4.1, en la que se requiere conocer la distancia focal f_x y la distancia entre los focos de la cámara estéreo B. Todos estos valores son parámetros fijos que dependen del hardware del vehículo, y que en el caso de los datasets como KITTI y SHIFT son dados directamente.

$$Z(u,v) = \frac{f_x \cdot B}{D(u,v)} \tag{4.1}$$

4.2.3 Filtrado y proyección

La información obtenida en el apartado anterior debe ser adaptada para poder utilizar el detector 3D de la siguiente etapa.

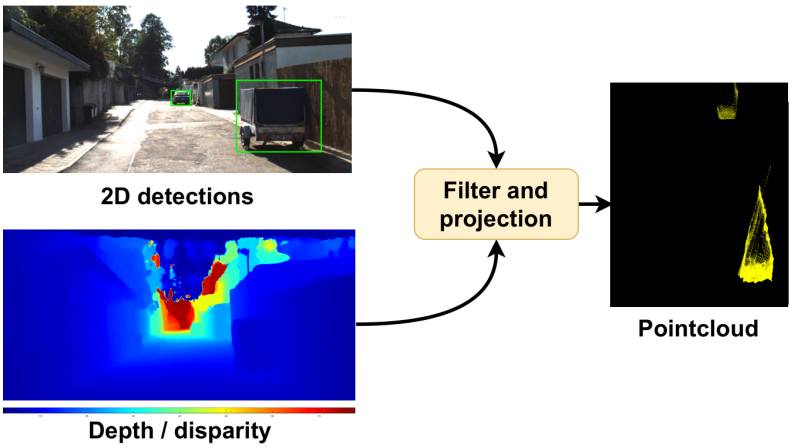

Figura 4.5: Esquema de filtrado y proyección del sistema multietapa.

En primer lugar, para reducir la probabilidad de obtener detecciones el posiciones erróneas, se eliminan todos los valores del mapa de profundidad que se encuentren fuera de las bounding boxes obtenidas de la YOLOv5 (Figura 4.6a). Con este filtrado se logra eliminar prácticamente por completo ruido introducido por el cielo u objetos que no son de interés. Con los valores restantes se calcula la posición 3D de cada píxel siguiendo los cálculos de la Ecuación 4.2, dando lugar a una nube de puntos con formato similar a la de un LiDAR o radar, $PCL \in \mathbb{R}^{n\times 4}$, en el que el canal adicional a la posición no aporta información, sin embargo se mantiene para dar coherencia a las operaciones matriciales.

$$x = \frac{(u - c_x) \cdot z}{f_x}, \quad y = \frac{(v - c_y) \cdot z}{f_y}, \quad z = Z(u, v) \tag{4.2}$$

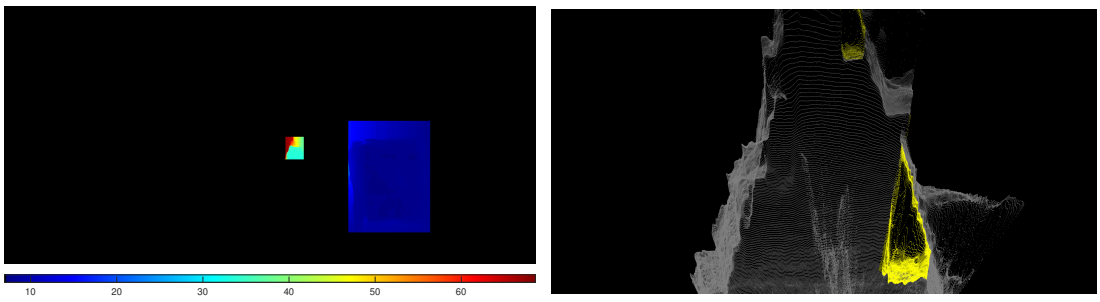

Figura 4.6: a) Imagen de profundidad filtrada. b) Nube de puntos generada a partir de la profundidad. En gris todos los puntos y en amarillo los que corresponden a la zona filtrada.

Debido a que normalmente los datos 3D se utilizan estando referenciados al sistema de coordenadas del LiDAR, tras generar la nube se utiliza la matriz de transformación correspondiente para cambiar la nube de sistema de referencia de la cámara al LiDAR (Ecuación 4.3).

$$PCL_{lidar} = Tr\_cam\_2\_velo^{-1} \times PCL_{camera} \tag{4.3}$$

En el Capítulo 5 se realiza una comparativa de tiempo y resultados entre la arquitectura propuesta en [13] y la modificada en este trabajo

4.2.4 Detección 3D

La fase final de nuestro sistema se dedica a llevar a cabo la detección de objetos sobre la nube generada como resultado de la interacción entre las distintas etapas de la arquitectura (como se ilustra en la Figura 4.7). Siguiendo la tendencia de eficiencia que hemos mantenido a lo largo de todas las fases, se requiere un sistema ágil para esta tarea. En este sentido, se ha optado por el uso del detector PointPillars [15], cuya elección se basa en su capacidad para brindar una detección precisa y eficaz, cumpliendo con nuestra prioridad de velocidad y eficiencia.

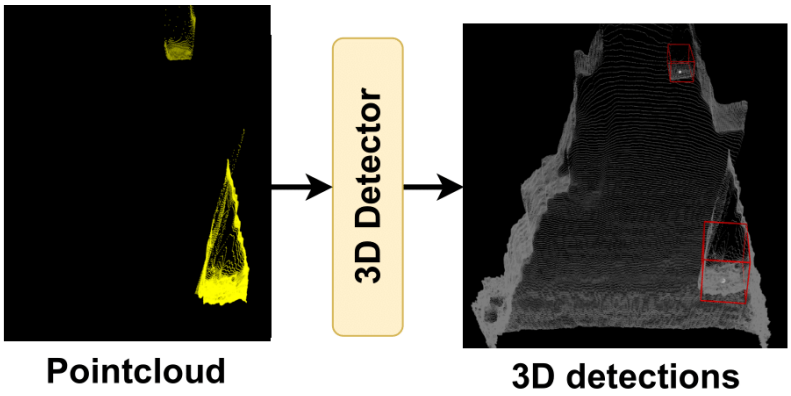

Figura 4.7: Esquema de detección 3D del sistema multietapa.

4.3 Entrenamiento

Dado que los dos aspectos fundamentales de este estudio son la identificación de un sistema de detección adecuado para un vehículo autónomo y la evaluación de su capacidad de generalización entre un entorno real y una simulación, los procesos de entrenamiento se llevan a cabo de la siguiente manera:

En una primera etapa, se procede a entrenar cada una de las propuestas utilizando el conjunto de datos real de KITTI. Esto permite comparar y contrastar los resultados obtenidos en nuestro entorno con las métricas previamente proporcionadas por los diversos autores. Esta fase de entrenamiento es esencial para establecer una base sólida de referencia y evaluar cómo se comporta el sistema en condiciones del mundo real.

A continuación, se realiza una segunda fase de entrenamiento utilizando datos de SHIFT, siguiendo un esquema de entrenamiento similar al utilizado en la fase anterior. Esto incluye la configuración del optimizador, la selección de hiperparámetros y demás elementos relevantes. Este proceso busca verificar la capacidad del sistema para adaptarse y generalizar su rendimiento desde un entorno simulado hacia situaciones del mundo real.

Todos los entrenamientos utilizan la técnica de transfer learning, la cual aprovecha el conocimiento de las redes sobre otras bases de datos para acelerar el aprendizaje. Para concretar, algunos de los detalles de los entrenamientos son:

- Detectores 3D monoculares end-to-end: se entrenan durante 100 epochs.
- Estimadores de profundidad (etapa del detector 3D modular): se entrenan durante 50 epochs.
- Detector 3D de nube de puntos (etapa final del detector 3D modular): se entrena 15 epochs sobre nubes generadas con los estimadores de profundidad.

Capítulo 5

Experimentos y resultados

Los grandes resultados requieren grandes ambiciones.

Heráclito

En este capítulo se recopilan los experimentos llevados a cabo en este trabajo, así como los resultados que han sido obtenidos. Se comienza con una explicación detallada sobre las métricas que se utilizan para evaluar tanto la detección de objetos como la estimación de profundidad. A continuación se presenta en detalle el proceso de entrenamiento de todos los sistemas considerados, incluyendo tanto los sistemas end-to-end como nuestra propuesta modular. Este entrenamiento es seguido por una exhaustiva evaluación de cada sistema, donde medimos su rendimiento y analizamos su eficacia en distintos contextos y escenarios. Además, este capítulo incluye una sección en la que se analizan resultados cualitativos obtenidos. Aquí, a través de representaciones visuales y ejemplos concretos, se muestra cómo los sistemas han logrado detectar y estimar la profundidad en diversos entornos y situaciones de prueba. Esta visualización de los resultados cualitativos proporciona una perspectiva clara y ayuda a comprender mejor el desempeño de los sistemas en situaciones reales.

5.1 Métricas

Con el propósito de obtener información sobre el rendimiento de los diversos sistemas de DL, es imprescindible generar un conjunto de métricas estándar que permita la comparativa con otros modelos SOTA. En el caso del trabajo realizado, son necesarias las métricas que permitan evaluar tareas de detección de objetos, tanto en una imagen como en 3D, y tareas de estimación de profundidad. Los experimentos se han realizado sobre una GPU NVIDIA GTX 1080 Ti.

5.1.1 Detección de objetos

Para evaluar el rendimiento de las diferentes etapas de detección planteadas se utilizan las mismas métricas que el benchmark de *3D Object Detection* de KITTI y que son ampliamente utilizadas en el estado del arte de detectores de objetos.

En primer lugar es importante entender el concepto de Intersection Over Union (IoU), el cual es la relación entre el área/volumen común de la detección y el etiquetado y el área/volumen total entre ambas bounding boxes (Ecuación 5.1).

$$IoU = \frac{Intersection}{Union} \tag{5.1}$$

Este valor se utiliza para asociar las detecciones generadas con las etiquetas de la base de datos, categorizándolas en función de si supera un valor de IoU umbral:

- True Positive (TP): se considera que una detección es un TP cuando la bounding box supera el IoU umbral y es asociada correctamente a una etiqueta.
- False Positive (FP): una detección es un FP cuando no se puede asociar a ninguna etiqueta, normalmente porque la red predice elementos que no existen realmente.
- False Negative (FN): este caso sucede cuando existe una etiqueta pero la red no ha detectado el elemento.

A partir de esta clasificación de las detecciones, se generan los valores de *precision* y *recall* fijando múltiples umbrales de los *scores* de las detecciones:

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN} \tag{5.2}$$

Con los diferentes valores obtenidos, se genera una curva de precision-recall como la de la Figura 5.1 para cada clase. La información se puede resumir en la métrica Average Precision (AP), la cual se calcula midiendo el área bajo la curva.

Es importante destacar que se proporcionan los valores de AP40, que hace referencia al número de puntos utilizados para construir la curva en cada caso. Si el sistema detecta correctamente, los valores de precisión tienden a aumentar para valores de recall altos. Las métricas de rendimiento de los detectores de objetos se dan para valores de IoU de 0.7 en el caso de la clase *Car* y de 0.5 para el resto.

Además de métricas para medir las detecciones, KITTI establece la métrica Average Orientation Similarity (AOS) para medir la calidad de las orientaciones generadas por los detectores, siguiendo las ecuaciones reflejadas en 5.3, donde $\mathcal{D}(r)$ es el conjunto de detecciones para un cierto valor de recall r, y $\Delta_\theta^{(i)}$ es la diferencia en la estimación del ángulo θ y el valor real para una detección asociada i. δ_i actúa como un mecanismo de penalización: $\delta_i = 1$ si $\text{IoU}_i \geq 0{,}5$, o $\delta_i = 0$ en caso contrario.

$$AOS = \frac{1}{11} \sum_{k=0}^{k=10} \max s\,(0{,}1 \cdot r) \qquad s(r) = \frac{1}{|\mathcal{D}(r)|} \sum_i \frac{1 + \cos \Delta_\theta^{(i)}}{2} \delta_i \tag{5.3}$$

5.1.2 Estimación de profundidad

La evaluación de la profundidad requiere un cambio drástico en las métricas utilizadas, ya que no buscamos clasificar y evaluar detecciones, sino calcular como de correctos son los valores de

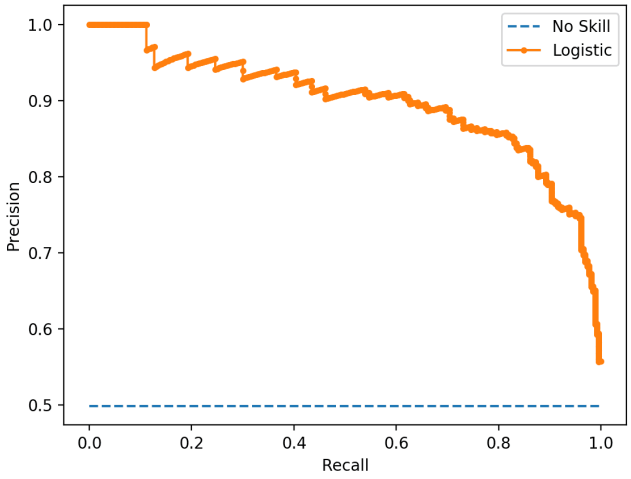

Figura 5.1: Ejemplo de curva precision-recall. Fuente

profundidad estimados en cada píxel. Se busca calcular los errores entre el valor de distancia etiquetado y el valor estimado píxel a píxel, como se puede observar en la Figura 5.2.

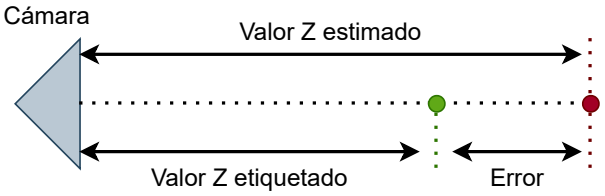

Figura 5.2: Esquema de valor estimado, etiquetado y error en un píxel de profundidad.

Concretamente se utilizan métricas estandarizadas en benchmarks como el del dataset KITTI [1], en las que todos los valores de Z se dan en metros (a excepción de iRMSE que es en Km) y N es el número de píxeles de cada imagen:

- Root Mean Squared Error (**RMSE**): diferencia al cuadrado entre el valor de profundidad estimado y el valor etiquetado.

$$\textbf{RMSE} = \sqrt{\frac{\sum_{i,j}(\widehat{Z}^{i,j} - Z^{i,j})^2}{N}} \tag{5.4}$$

- Inverse Root Mean Squared Error (**iRMSE**): diferencia al cuadrado entre los valores inversos de profundidad estimados y valores etiquetados, ambos en Km.

$$\textbf{iRMSE} = \sqrt{\frac{\sum_{i,j}(\frac{1}{\widehat{Z}^{i,j}} - \frac{1}{Z^{i,j}})^2}{N}} \tag{5.5}$$

- Relative Absolute Error (**absErrorRel**): porcentaje de error relativo al valor etiquetado en el dataset.

$$\textbf{absErrorRel} = \frac{1}{N}\sum_{i,j}\frac{|\widehat{Z}^{i,j} - Z^{i,j}|}{Z^{i,j}} \tag{5.6}$$

- Logarithmic Root Mean Squared Error (**RMSE log**): diferencia al cuadrado entre el logaritmo natural del valor estimado y el logaritmo natural del valor etiquetado.

$$\textbf{RMSElog} = \sqrt{\frac{\sum_{i,j}(log(\widehat{Z}^{i,j}) - log(Z^{i,j}))^2}{N}} \tag{5.7}$$

- Threshold n (δ_n): representa el porcentaje de píxeles que difieren menos de un porcentaje (25 % para n = 1).

$$max(\frac{\widehat{Z}^{i,j}}{Z^{i,j}}, \frac{Z^{i,j}}{\widehat{Z}^{i,j}}) < 1{,}25^n \tag{5.8}$$

5.2 Evaluación de PointPillars

Actualmente una gran variedad de detectores 3D se basan generalmente en información procedente de LiDAR, en lugar de utilizar a la cámara como la fuente principal de los datos del entorno. Para plantear una base sobre la que comparar el resto de resultados, se ha evaluado PointPillars [15], uno de los detectores de LiDAR mas utilizados, con la misma distribución de datos y condiciones que el resto.

	BEV			**3D**			**AOS**		
	Easy	**Med**	**Hard**	**Easy**	**Med**	**Hard**	**Easy**	**Med**	**Hard**
Car (IoU 0.7)									
Por dificultad	92.58	88.50	85.75	88.52	79.27	76.31	95.67	92.09	89.14
Media		88.94			81.37			92.30	
Pedestrian (IoU 0.5)									
Por dificultad	61.45	55.58	51.16	57.28	51.01	46.44	44.12	41.73	39.16
Media		56.06			51.58			41.67	
Cyclist (IoU 0.5)									
Por dificultad	87.75	66.59	62.69	83.90	62.79	59.50	88.73	69.48	66.49
Media		72.34			68.73			74.90	

Tabla 5.1: Resultados de PointPillars sobre el conjunto de validación de KITTI, con unos pesos entrenados sobre el LiDAR.

Los valores reflejados en la Tabla 5.1 mantienen la misma tendencia que los valores proporcionados sobre el conjunto de test por los autores en la publicación original [15]. Concretamente se presentan las métricas de detección sobre una vista de pájaro (BEV), directamente en el entorno 3D y un valor AOS para evaluar la orientación de los objetos detectados. Tomando por ejemplo la métrica en vista de pájaro BEV, el rendimiento en nuestro conjunto de los coches para dificultad media es de 88.5 mientras que el de la publicación es 86.10, diferencia que se atribuye a que el conjunto de test son datos de secuencias completamente diferentes a las de entrenamiento y validación. Este efecto se traslada a las otras dos clases evaluadas, las cuales presentan peores métricas especialmente por ser de menor tamaño que los coches, dificultando su detección sobre nubes de puntos. En el caso de peatones se obtiene un 55.58 de BEV para

dificultad media, reportando los autores un 50.23, mientras que los ciclistas en nuestro caso obtienen un 66.59 y en la publicación un 62.25.

5.3 Entrenamiento y evaluación de detección de objetos 3D end-to-end

Las primeras pruebas con datos procedentes únicamente de cámara han consistido en entrenar la dos redes end-to-end seleccionadas tras el estudio realizado en el Capítulo 2: SMOKE [11] y PGD [12].

En ambos casos se emplean las siguientes técnicas para acelerar el aprendizaje:

1. Transfer learning: los entrenamientos parten de unos pesos preentrenados en otros dataset, de tal forma que se aprovecha ese conocimiento para obtener mejores resultados.

2. Data augmentation: para ayudar al modelo a generalizar se aplican ciertas transformaciones a los datos de entrenamiento para aumentar la cantidad de información que recibe la red. Concretamente se utiliza inversión aleatoria, la cual invierte horizontalmente las imágenes y variaciones de escala para detectar objetos a diferentes tamaños y distancias.

En el caso concreto de SMOKE, se han empleado un total de 100 epochs para el proceso de entrenamiento utilizando el optimizador Adam y partiendo de los pesos preentrenados proporcionados de MMDetection3D.

	BEV			**3D**			**AOS**		
	Easy	**Med**	**Hard**	**Easy**	**Med**	**Hard**	**Easy**	**Med**	**Hard**
Car (IoU 0.7)									
Por dificultad	23.60	16.13	12.93	13.19	10.47	9.85	89.02	86.37	77.85
Media		17.55			11.17			84.41	
Pedestrian (IoU 0.5)									
Por dificultad	8.76	6.92	6.59	8.34	6.63	6.10	48.60	43.00	36.93
Media		7.42			7.02			42.84	
Cyclist (IoU 0.5)									
Por dificultad	5.35	4.56	3.81	5.24	3.91	3.82	38.25	29.13	28.78
Media		4.57			4.32			32.05	

Tabla 5.2: Resultados de SMOKE sobre el split de validación de KITTI.

Los resultados de evaluar los pesos entrenados sobre nuestro split de validación de KITTI son los presentados en la Tabla 5.2. La primera idea que se puede resaltar es la clara caída de AP respecto a PointPillars, especialmente debido a que se trabaja con una imagen monocular, por lo que inferir la información 3D no es tarea sencilla. Otro punto a destacar es que la clase *Car* sigue siendo la que mejor se detecta, ya que al igual que sucede con el LiDAR, los vehículos al ser por norma general los elementos mas voluminosos en la escena son mas sencillos de detectar.

En el caso de *Pedestrian* y *Cyclist* los resultados empeoran bastante, lo cual se puede achacar al error en al estimación del modelo y al menor tamaño de las cajas de los objetos.

Los autores de SMOKE únicamente ofrecen datos para la clase *Car*, concretamente reportan un AP BEV de 19.99, 15.61, 15.28 y un AP 3D de 14.76, 12.85, 11.50 para cada dificultad, lo cual supone un rendimiento similar al de los experimentos de este trabajo. Los resultados por cada dificultad del resto de clases se comparan con los proporcionados por MMDetection3D para los mismos IoU:

- Pedestrian: AP BEV de 12.61, 11.32, 11.14 y AP 3D de 11.13, 11.10, 10.67, lo cual supone varios puntos mas elevado que nuestro experimento.
- Cyclist: tanto AP BEV como AP 3D son valores alrededor de 1, lo cual es inferior a nuestros resultados.

La tendencia general de los resultados obtenidos en los experimentos se encuentra en línea con los reportados por las otra fuentes.

Este proceso de entrenamiento se repite manteniendo la configuración y utilizando únicamente los datos del dataset SHIFT, de tal forma que se pueda evaluar el modelo en simulación para comparar con el rendimiento en real. Es importante destacar que no se pueden obtener resultados por dificultad ya que el dataset no proporciona esta información, y que la clase *Cyclist* no está etiquetada de la misma forma en los dos datasets, por lo que los valores de esta clase para SHIFT son simplemente orientativos, no sirviendo para la comparativa.

	BEV	**3D**	**AOS**
Car (IoU 0.7)	31.67	23.87	22.66
Pedestrian (IoU 0.5)	4.25	4.16	34.58
Cyclist (IoU 0.5)	1.68	1.14	9.09

Tabla 5.3: Resultados de SMOKE sobre el split de validación de SHIFT.

En la Tabla 5.3 se plasman los resultados finales sobre SHIFT del entrenamiento realizado. Ya que no hay dificultad en las etiquetas, estos valores se deben comparar con los valores medios proporcionados para KITTI en la Tabla 5.2. La clase *Car* consigue mejorar sus resultados, por ejemplo en BEV se mejora de 17.55 a 31.67 (+14.12 %). Sin embargo, en el caso de *Pedestrian* el rendimiento cae en BEV de 7.42 a 4.25 (-3.17 %). Esta caída se puede asociar al entorno simulado, en el que los vehículos son mas parecidos a los reales que en el caso de los peatones, lo que se traduce en un mayor ruido en las imágenes y a la entrada de la red. Además, SHIFT contiene una cantidad muy elevada de peatones por escena en comparación con datasets reales como KITTI, lo cual puede perjudicar aún mas la métrica.

El proceso de entrenamiento se repite para el detector PGD, realizando 100 epochs con un optimizador SGD y manteniendo las mismas técnicas de data augmentation que en el entrenamiento de SMOKE. De la misma manera se realiza el entrenamiento sobre los datos sintéticos de SHIFT para realizar la comparativa. Ambos resultados están recogidos en la Tabla 5.4 y 5.5,

observando que sobre los datos reales el rendimiento en vehículos mejora respecto a SMOKE, con un AP BEV que se eleva desde 16.13 a 21.88 (+5.75 %) para dificultad media, mientras que en peatones y ciclistas se mantiene prácticamente idéntico. Respecto a los resultados de la publicación original, los autores proporcionan métricas sobre el conjunto de validación de la clase *Car*, las cuales se pueden comparar a las de la Tabla 5.4. Concretamente reportan un AP BEV de 30.56, 23.67, 20.84 y un AP 3D de 24.35, 18.34, 16.90; lo cual marca un tendencia similar al experimento con resultados ligeramente mejores para esta clase.

	BEV			3D			AOS		
	Easy	**Med**	**Hard**	**Easy**	**Med**	**Hard**	**Easy**	**Med**	**Hard**
Car (IoU 0.7)									
Por dificultad	27.20	20.72	17.73	20.74	16.11	12.62	93.43	81.74	72.39
Media		21.88			16.49			82.52	
Pedestrian (IoU 0.5)									
Por dificultad	8.17	7.36	6.78	7.24	6.67	6.11	44.17	34.87	29.04
Media		7.43			6.67			36.03	
Cyclist (IoU 0.5)									
Por dificultad	5.86	2.64	2.36	4.89	2.15	1.97	27.65	15.17	14.56
Media		3.62			3.01			19.13	

Tabla 5.4: Resultados de PGD sobre el split de validación de KITTI.

En el caso de los datos de simulación (Tabla 5.5) se mantiene la tendencia de SMOKE, mejorando resultados en la clase *Car*, destacando en este caso que es una mejora menor, de un AP BEV medio de 21.88 a 30.22 (+8.34 %). Los resultados para peatones siguen siendo pobres.

	BEV	**3D**	**AOS**
Car (IoU 0.7)	30.22	27.25	24.91
Pedestrian (IoU 0.5)	6.54	7.67	31.31
Cyclist (IoU 0.5)	3.14	3.25	13.19

Tabla 5.5: Resultados de PGD sobre el split de validación de SHIFT.

Además de métricas de rendimiento, se ha medido el tiempo de inferencia de los dos modelos sobre el hardware donde se realizan las pruebas, obteniendo 55 ms para SMOKE 90 ms para PGD de tiempo medio de inferencia sobre las imágenes de KITTI.

Cabe destacar que se ha realizado la evaluación sobre KITTI utilizando pesos con entrenamientos realizados únicamente en los datos sintéticos de SHIFT. El rendimiento de ambos detectores en este contexto es bastante pobre, debido probablemente a que al ser dos entornos con grandes diferencias a nivel visual, es necesario recurrir a téncicas de transfer learning de datos reales para conseguir mejores resultados.

5.4 Entrenamiento y evaluación de sistemas de estimación de profundidad

Tras realizar las evaluaciones de los detectores end-to-end, se procede a abordar las evaluaciones y entrenamientos de las diferentes etapas del sistema modular de detección 3D. En primer lugar, como se ha explicado en el Capítulo 4, es necesario generar información 3D a partir de las imágenes estéreo de entrada. Para ello es necesario entrenar y evaluar los dos modelos de estimación de profundidad que mas se ajustan a los requerimientos: SDN y CoEx. En ambos casos, los procesos de entrenamiento para ambas redes han seguido una estructura similar. Se ha utilizado transfer learning, aprovechando pesos preentrenados previamente, cada entrenamiento ha abarcado un total de 50 epochs y se ha utilizado el algoritmo Adam como optimizador.

Method	Train Set	Eval. Set	RMSE ↓	iRMSE ↓	RMSElog ↓	Rel. abs. error (%) ↓
SDN	KITTI	KITTI	3.3366	13.6359	0.1498	9.6186
	SHIFT	SHIFT	4.6104	16.3160	0.1855	8.2246
	SHIFT	KITTI	3.9165	21.2901	0.1914	11.9913
COEX	KITTI	KITTI	3.5426	15.6604	0.1607	10.3993
	SHIFT	SHIFT	4.3414	17.4171	0.1769	7.2447
	SHIFT	KITTI	6.2974	18.8357	0.2073	15.0839

Tabla 5.6: Métricas de profundidad de los dos modelos (SDN y CoEx) entrenando y evaluando sobre diferentes subsets de KITTI y SHIFT.

Method	Train Set	Eval. Set	δ_1 ↑	δ_2 ↑	δ_3 ↑
SDN	KITTI	KITTI	0.9581	0.9753	0.9856
	SHIFT	SHIFT	0.9396	0.9667	0.9785
	SHIFT	KITTI	0.9294	0.9624	0.9766
COEX	KITTI	KITTI	0.9567	0.9741	0.9843
	SHIFT	SHIFT	0.9383	0.9690	0.9805
	SHIFT	KITTI	0.9275	0.9612	0.9743

Tabla 5.7: Valores de delta thresholds en diferentes subsets de KITTI y SHIFT.

Dado que los datos etiquetados de profundidad de la simulación son mas ricos que los reales, adicionalmente se ha evaluado el comportamiento sobre datos reales si se utiliza el modelo entrenado únicamente sobre simulación. Los resultados de las evaluaciones se presentan en las Tablas 5.6 y 5.7, de las cuales se pueden inferir varias conclusiones clave. Primero, ambos modelos logran obtienen su mejor rendimiento en datos reales, siendo SDN el más destacado al ser un sistema más pesado, logrando reducir el RMSE en 20 cm y aproximadamente un 0,7 % de error absoluto relativo. En contraste, al entrenar y evaluar en entornos simulados, se observa una disminución en el rendimiento, posiblemente debido a los drásticos cambios visuales en las características de las imágenes sintéticas. Adicionalmente, estos sistemas de estimación muestran una mejor generalización en comparación con los detectores entrenados en el apartado anterior,lo cual se refleja en un menor salto en las métricas entre el entorno real y simulado en comparación con SMOKE o PGD.

En la Figura 5.3 se ha plasmado el resultado de ambos detectores en una escena de KITTI. Se observa como en zonas cercanas al vehículo hasta 20-30 metros, ambos sistemas captan prácticamente todos los objetos correctamente, añadiendo cierto ruido en los bordes. Sin embargo, al alejarse la estimación comienza a empeorar debido a la falta de resolución de la cámara, lo cual se nota aún mas en el caso de CoEx al ser una red mas ligera. Por ejemplo, CoEx no logra caracterizar los bordes de los semáforos, señales y postes del lado izquierdo de la escena. Además los vehículos del fondo se difuminan, mientras que la SDN logra visualizar el contorno.

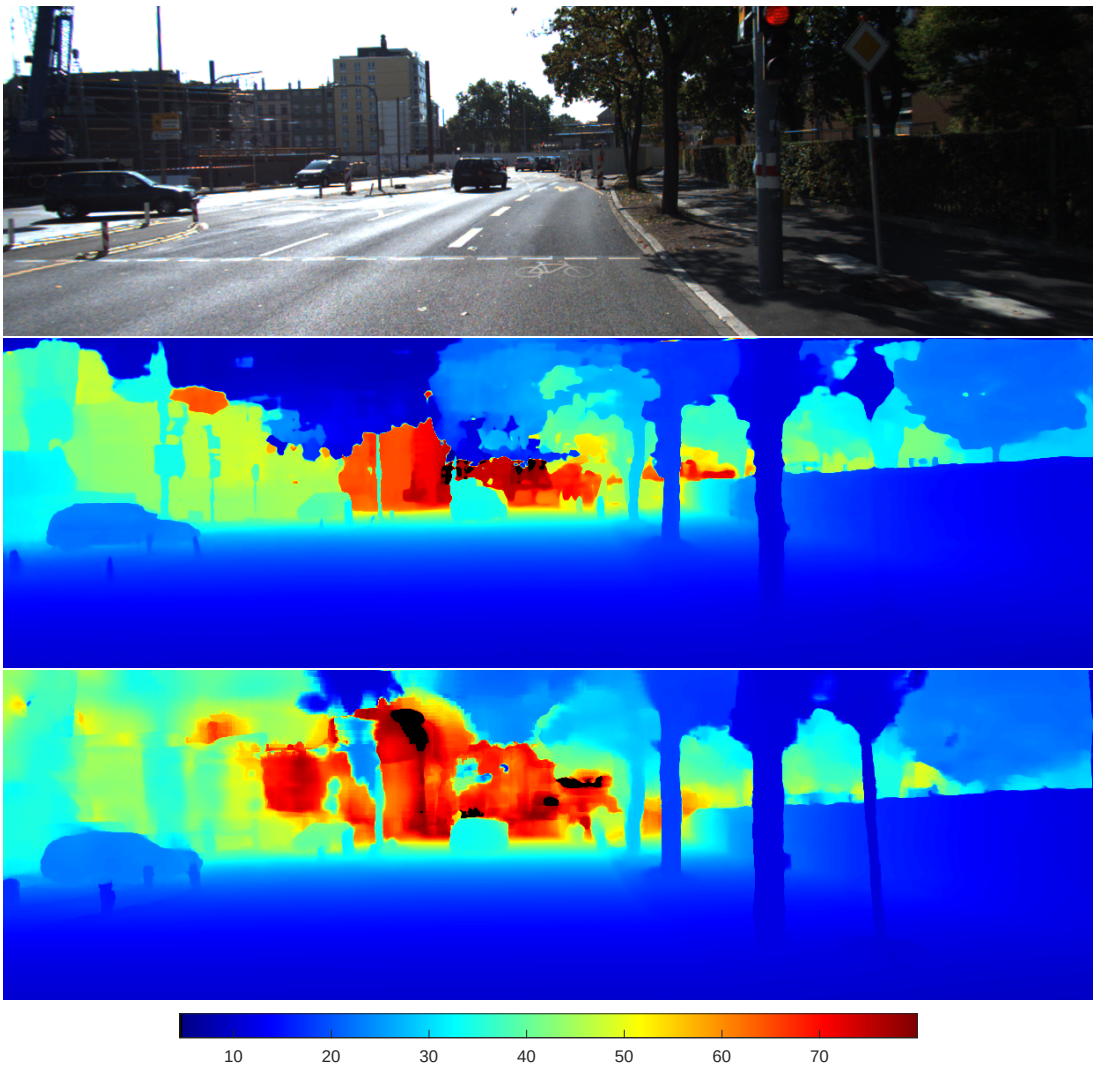

Figura 5.3: Resultado cualitativo de las redes de estimación de profundidad sobre KITTI. a) Imagen original. b) Resultado de SDN. c) Resultado de CoEx

Otro ejemplo donde se pueden observar estos mismos efectos en la estimación es el de la Figura 5.4. De nuevo la SDN consigue capturar los bordes de una manera mas precisa, especialmente en las señales u objetos verticales con cierta altura. Sin embargo, los objetos importantes de la escena se encuentran a una menor distancia que en el ejemplo anterior, luego ambas redes identifican los bordes de una manera bastante similar, consiguiendo la SDN aportar ciertos detalles extra, como se puede ver en la furgoneta del centro de la escena.

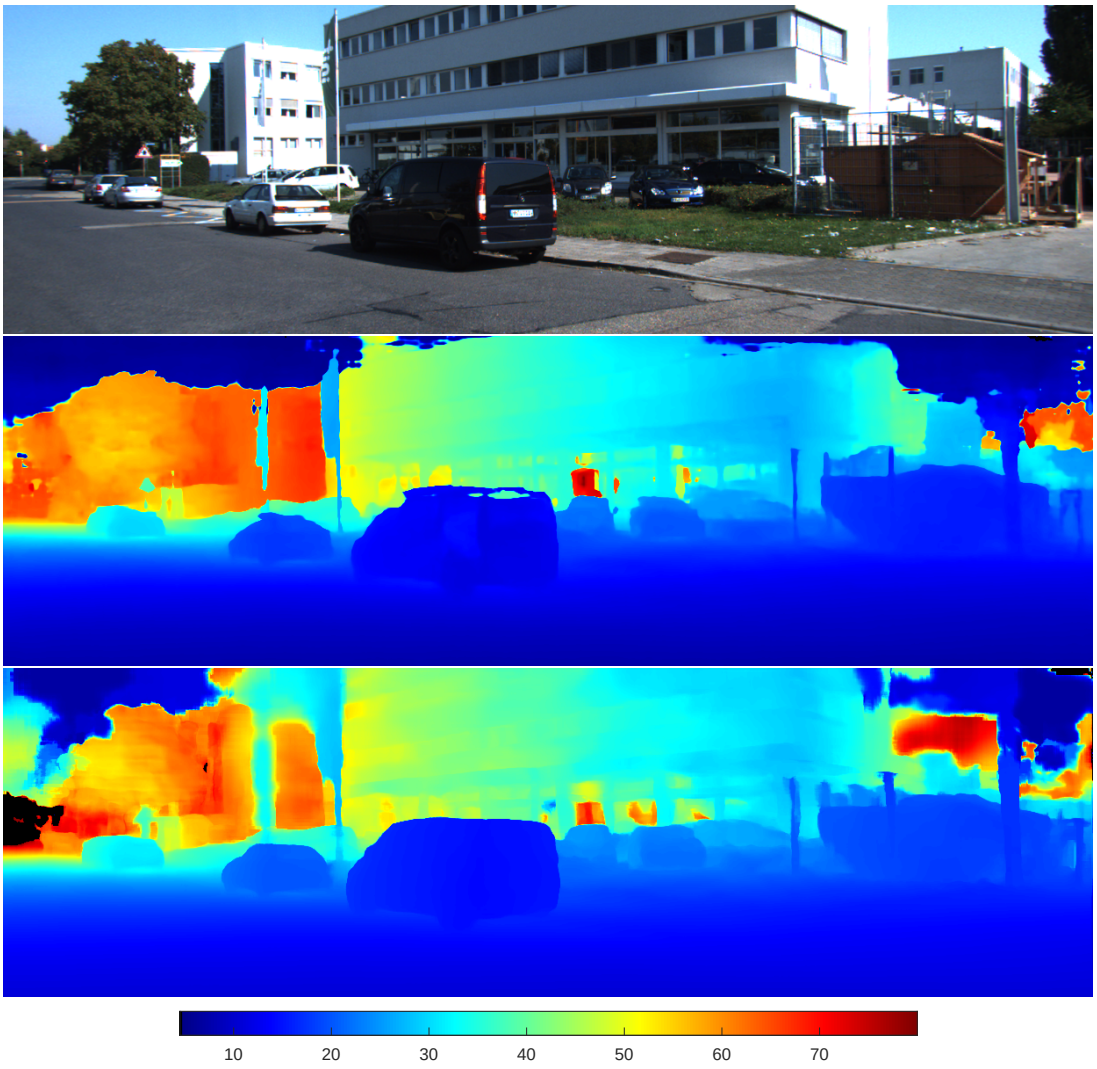

Figura 5.4: Segundo resultado cualitativo de las redes de estimación de profundidad sobre KITTI. a) Imagen original. b) Resultado de SDN. c) Resultado de CoEx

5.5 Evaluación de la arquitectura modular completa

Si bien la evaluación de los estimadores de profundidad nos permite conocer como de bien funcionan las etapas intermedias de la arquitectura modular, el objetivo final es la detección de los objetos en 3D. Por ello, para poder comparar esta arquitectura con las redes end-to-end evaluadas anteriormente, es necesario entrenar la última etapa y evaluar la arquitectura completa. En este caso, el entrenamiento no es tan inmediato como si se entrenara PointPillars directamente sobre KITTI, ya que las nubes de LiDAR y las generadas por la estimación de profundidad tienen características muy diferentes. En consecuencia, el primer paso para entrenar esta última etapa es generar los mapas de profundidad de todos los datos de entrenamiento y validación utilizando SDN y CoEx. Estos mapas de profundidad se utilizan para generar las nubes de puntos con las que se entrena el detector durante 15 epochs, las cuales son suficientes para saturar los resultados. Al igual que en en la comparativa de SMOKE y PGD, es necesario calcular la media de los resultados de las tres dificultades de KITTI para comparar con SHIFT.

La clase *Cyclist* no se incluye ya que, como se ha comentado anteriormente, el etiquetado en ambos datasets es diferente, en uno se etiqueta la bicicleta y ciclista como un único objeto, mientras que en el otro se etiqueta bicicleta y persona por separado, luego no son comparables.

Dataset	**Método**	**Car (IoU 0.7)**		
		BEV	**3D**	**AOS**
KITTI	**PP**	88.94	81.37	92.30
	Propuesta (SDN + PP)	66.97	48.11	86.22
	Propuesta (CoEx + PP)	57.97	41.55	77.55
SHIFT	**PP**	94.13	92.53	30.69
	Propuesta (SDN + PP)	66.79	59.68	27.87
	Propuesta (CoEx + PP)	66.74	57.62	31.98

Tabla 5.8: Resultados de las propuestas de detección modular sobre ambos datasets para la clase Car.

Dataset	**Método**	**Pedestrian (IoU 0.5)**		
		BEV	**3D**	**AOS**
KITTI	**PP**	56.06	51.58	41.67
	Propuesta (SDN + PP)	13.54	10.04	21.07
	Propuesta (CoEx + PP)	16.47	12.17	15.27
SHIFT	**PP**	50.19	31.22	20.53
	Propuesta (SDN + PP)	9.89	9.49	7.40
	Propuesta (CoEx + PP)	6.75	6.51	3.21

Tabla 5.9: Resultados de las propuestas de detección modular sobre ambos datasets para la clase Pedestrian.

Todos los resultados de la evaluación cuantitativa se recogen en las Tablas 5.8 y 5.9. Para ambos datasets los resultados obtenidos con la arquitectura modular mejoran considerablemente a los sistemas end-to-end, alcanzando valores de 66.97 y 57.97 de AP BEV, superando a SMOKE

y PGD con 17.55 (+49.42 % y +40.42 %) y 21.88 (+45.09 % y +36.09 %) respectivamente en KITTI. Esto se puede achacar a que la información 3D se estima en una etapa aparte en lugar de en la misma red, como sucedía en las end-to-end. Esta mayor robustez permiten que la información del entorno sea mas descriptiva y por lo tanto se mejoren las detecciones finales.

Si bien el sistema modular genera mejores detecciones que los detectores end-to-end, especialmente en coches, presenta una desventaja considerable respecto al tiempo total de procesado. En la tabla 5.10 se realiza un estudio de ablación en el que se representan varias métricas de detecciones sobre KITTI y el tiempo total de cada arquitectura, comprobando el efecto de incluir nuestro filtro 2D y como se comporta respecto al rectificado original con la técnica de los autores: GDC o Graph Based Depth Correction. El mejor resultado se consigue con la versión completa de la arquitectura PseudoLidar++, sin embargo, el tiempo de procesado asciende hasta casi 500 ms. Si se elimina la rectificación con GDC se consigue bajar el tiempo hasta unos 360 ms, reduciendo la calidad de resultados considerablemente hasta valores de AP BEV de 65. Tanto en esa arquitectura como en la propuesta el cuello de botella es la red de estimación de profundidad, por lo que cambiar la SDN por CoEx reduce considerablemente el tiempo, con la consiguiente reducción de métricas. Destacar además como la introducción del filtro 2D en la arquitectura propuesta permite mejorar resultados y acelerar PointPillars gracias a la reducción del número de puntos que procesa.

Método	2D filter	Car BEV	Car 3D	Tiempo (ms)
PL ++	-	79.50	65.43	300 + 90 + 80
PL ++ (sin GDC)	-	65.56	54.43	300 + 80
SDN + PP (Propuesta)	✓	66.97	48.11	300 + 60
SDN + PP (Propuesta)		61.54	42.87	300 + 100
CoEx + PP (Propuesta)	✓	57.97	41.55	120 + 60
CoEx + PP (Propuesta)		55.81	39.74	120 + 100

Tabla 5.10: Estudio de ablación de nuestra propuesta modular y la arquitectura PseudoLidar++ (PL++) en función del uso de rectificación o filtrado 2D.

5.6 Comparativa y resultados cualitativos de detección 3D

Aunque las evaluaciones de los modelos nos proporcionan métricas cuantitativas para comprender el rendimiento de los diversos sistemas, es igualmente importante representar estos resultados en un formato visual, donde se aprecie el contexto de la escena y la información recibida de los sensores. Esta representación visual permite un análisis más profundo y facilita la identificación de posibles fuentes de fallos o áreas de mejora.

Previo a mostrar los resultados finales de las detecciones, se pretende mostrar con cierto detalle las salidas obtenidas en las diferentes etapas del sistema modular propuesto en un mismo entorno. En la Figura 5.5 se observan tres escenas (una por columna) con la información que discurre por cada módulo. En la imagen superior se representan los datos de entrada a la primera etapa. Los mapas de colores representan la salida de esta etapa, obteniendo las profundidades por píxel que se requieren para formar la nube de puntos. En las siguientes fotos se representan tanto la nube de puntos sobre la que se detecta, como las detecciones en verde y el ground-truth en rojo, de las que se extrae la información en vista de pájaro y se representa en la última fila de imágenes, en las que las detecciones son las cajas de color oscuro y el ground-truth son las cajas claras.

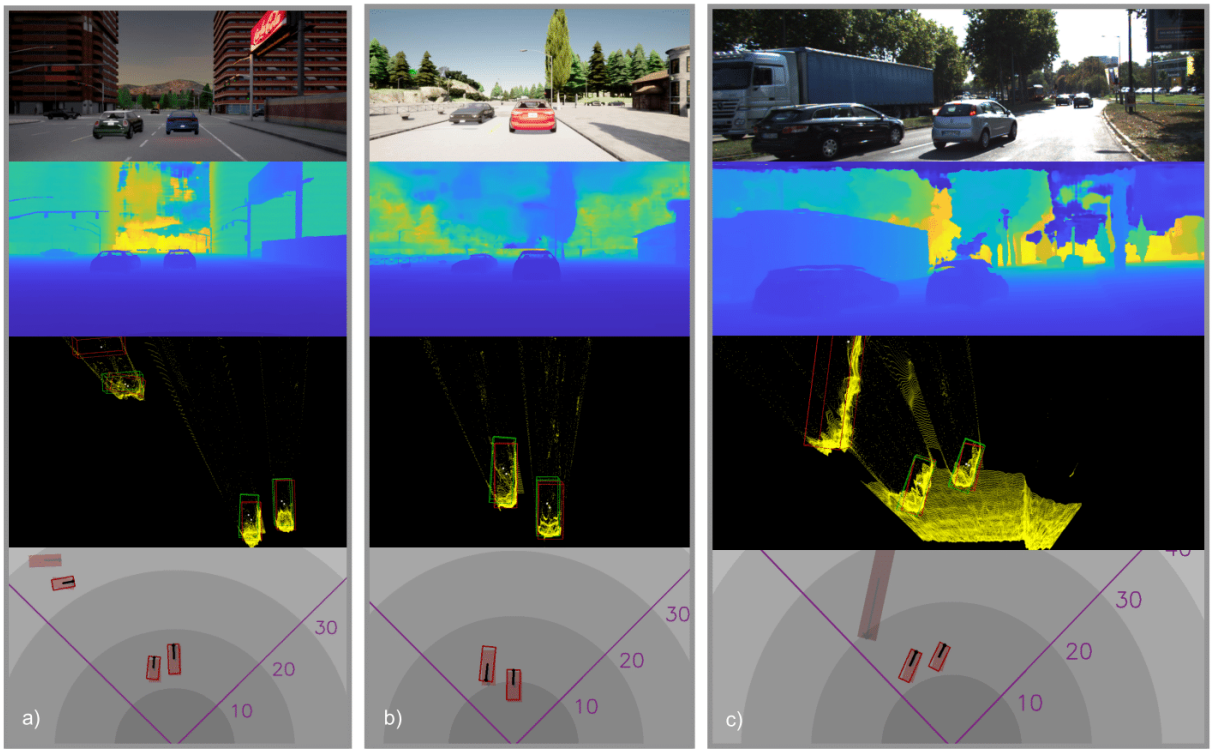

Figura 5.5: Resultado de las diferentes etapas del sistema modular. En las nibes de puntos, en rojo las bounding boxes etiquetadas y en verde las detecciones.
a) SHIFT con SDN y PointPillars. b) SHIFT con CoEx y PointPillars. c) KITTI con SDN y PointPillars.

En las Figuras 5.6, 5.7 y 5.8 se muestran los resultados sobre tres escenas de los cuatro modelos objeto de estudio: SMOKE, PGD y nuestra propuesta modular variando entre el estimador SDN y CoEx.

1. Figura 5.6 representa una escena en una autovía con tráfico denso, en la que es crítico conocer las posiciones del resto de vehículos por las velocidades que se pueden alcanzar. A excepción del sistema modular con SDN, todas las arquitecturas consiguen detectar los vehículos cercanos tanto en el carril actual como en el contiguo. En el caso de nuestra propuesta con SDN el falso negativo se puede deber a un fallo en el filtro 2D, detectándolo como un objeto de otra clase. Se destaca además que los sistemas end-to-end, al realizar la detección directamente sin mapa de profundidad, consiguen detectar vehículos mas ocluidos en carriles mas lejanos aunque disminuye la precisión.

2. Figura 5.7 representa una escena de una zona residencial con vehículos estacionados a ambos lados del carril. Los dos vehículos mas cercanos se detectan correctamente con todos los sistemas, destacando la gran precisión de las detecciones de nuestra propuesta con SDN. Los vehículos en zona lejana (mas de 30-40 metros) solamente se detectan en las redes end-to-end gracias a las características visuales de la imagen, sin embargo son detecciones con gran error. La estimación de profundidad a tanta distancia es dispersa y con error alto, luego no se llegan a detectar.

3. Figura 5.8 representa una calle residencial con pendiente ascendente, pudiendo comprobar la robustez de los sistemas frente a cambios de elevación. En todos los casos los dos vehículos entre 10 y 30 metros se detectan con gran precisión. Como sucedía en los anteriores casos, al aumentar la distancia la estimación de profundidad se vuelve menos precisa, por lo tanto en objetos cercanos a estructuras como muros, edificios y árboles, como es este caso, es muy probable que estos objetos no se detecten correctamente.

De la misma manera, se representan varias escenas de SHIFT, las cuales contienen mayor dificultad por el mayor número de elementos en la escena y porque se trata de imágenes del simulador CARLA.

1. En la Figura 5.9 se representan las detecciones sobre una intersección con múltiples vehículos. Se observa como en rangos cercanos de 10 a 20 metros, todos los sistemas logran detectar correctamente los dos vehículos que se encuentran justo delante. Respecto a los coches que están en las otras salidas de la intersección, el mas visible a la izquierda se detecta en todos los casos, obteniendo mas precisión si se utiliza la arquitectura modular. El resto al estar a 40 metros o mas, con obstáculos como plantas en la línea de visión de la cámara, no se detectan.

2. La Figura 5.10 sucede en una escena con una iluminación reducida. En los cuatro casos se detectan correctamente los dos vehículos mas cercanos. El coche mas lejano se detecta con bastante error por los sistemas end-to-end, mientras que los modulares no generan ninguna estimación, probablemente debido a que al ser un todoterreno, el detector 2D que alimenta al filtrado no haya generado la detección, por lo que se elimina de la entrada en la siguiente capa.

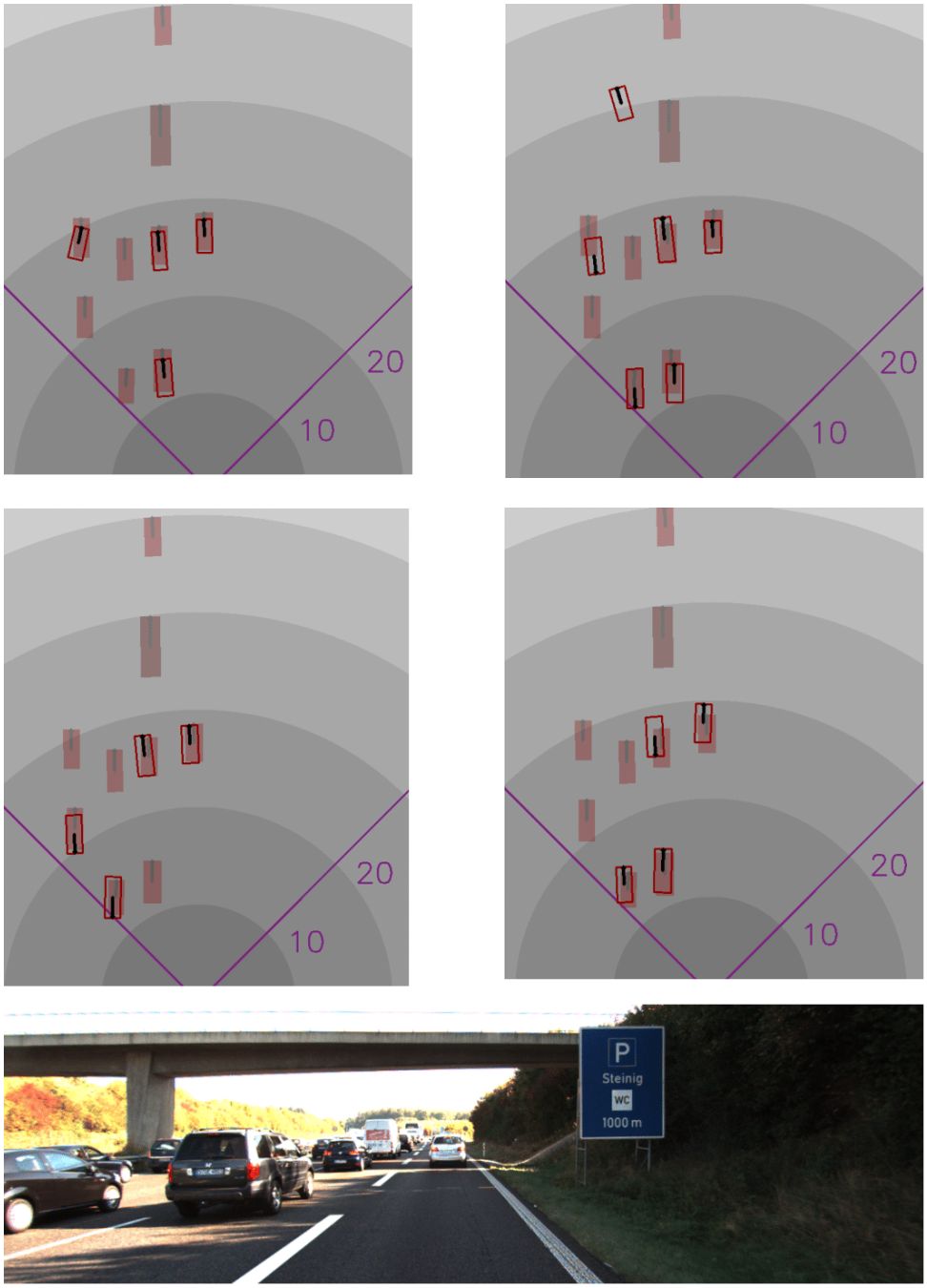

Figura 5.6: Resultado cualitativo de SMOKE (arriba izquierda), PGD (arriba derecha), la propuesta modular con SDN (abajo izquierda) y la propuesta modular con CoEx (abajo derecha) en KITTI.

Figura 5.7: Resultado cualitativo de SMOKE (arriba izquierda), PGD (arriba derecha), la propuesta modular con SDN (abajo izquierda) y la propuesta modular con CoEx (abajo derecha) en KITTI. (2)

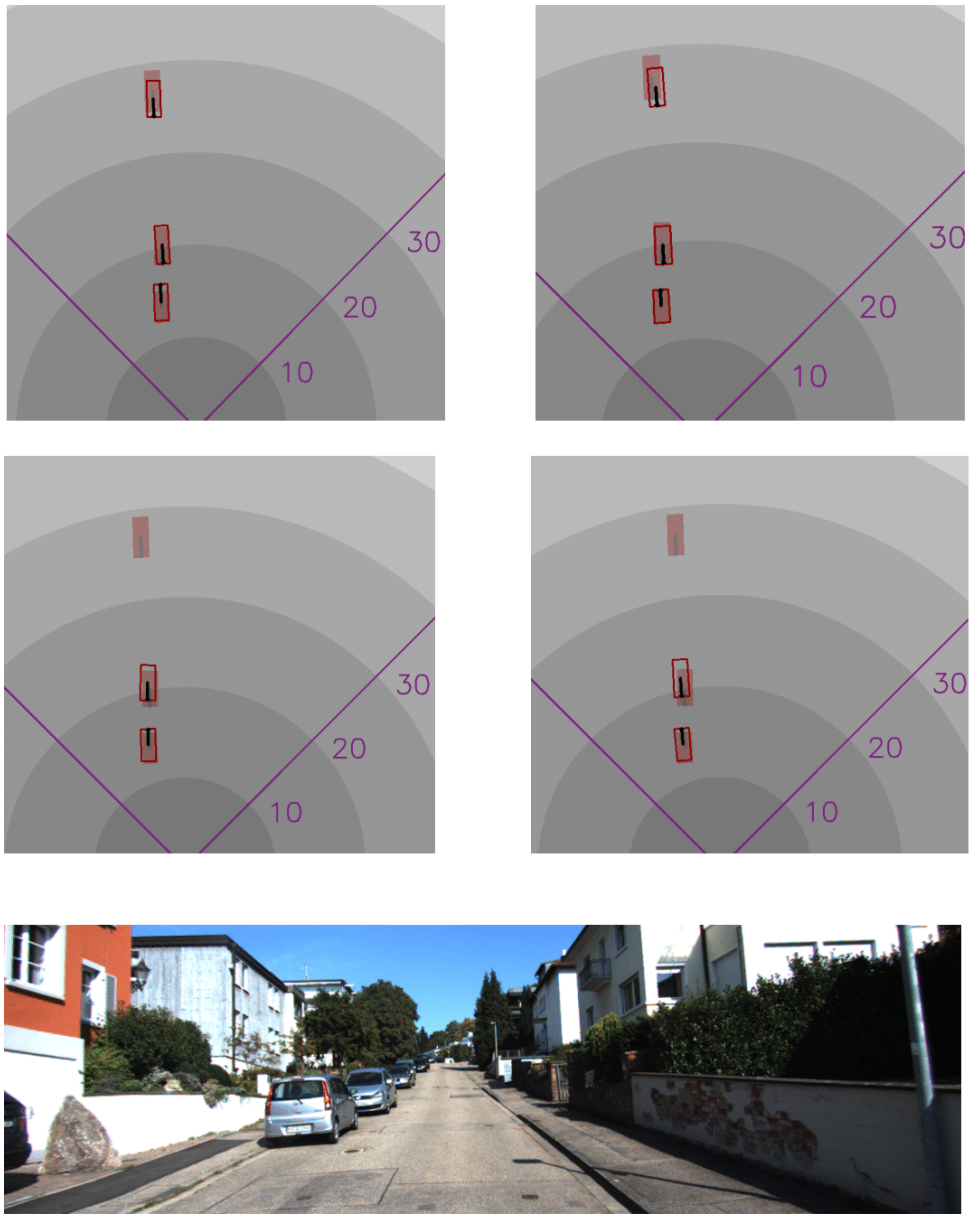

Figura 5.8: Resultado cualitativo de SMOKE (arriba izquierda), PGD (arriba derecha), la propuesta modular con SDN (abajo izquierda) y la propuesta modular con CoEx (abajo derecha) en KITTI. (3)

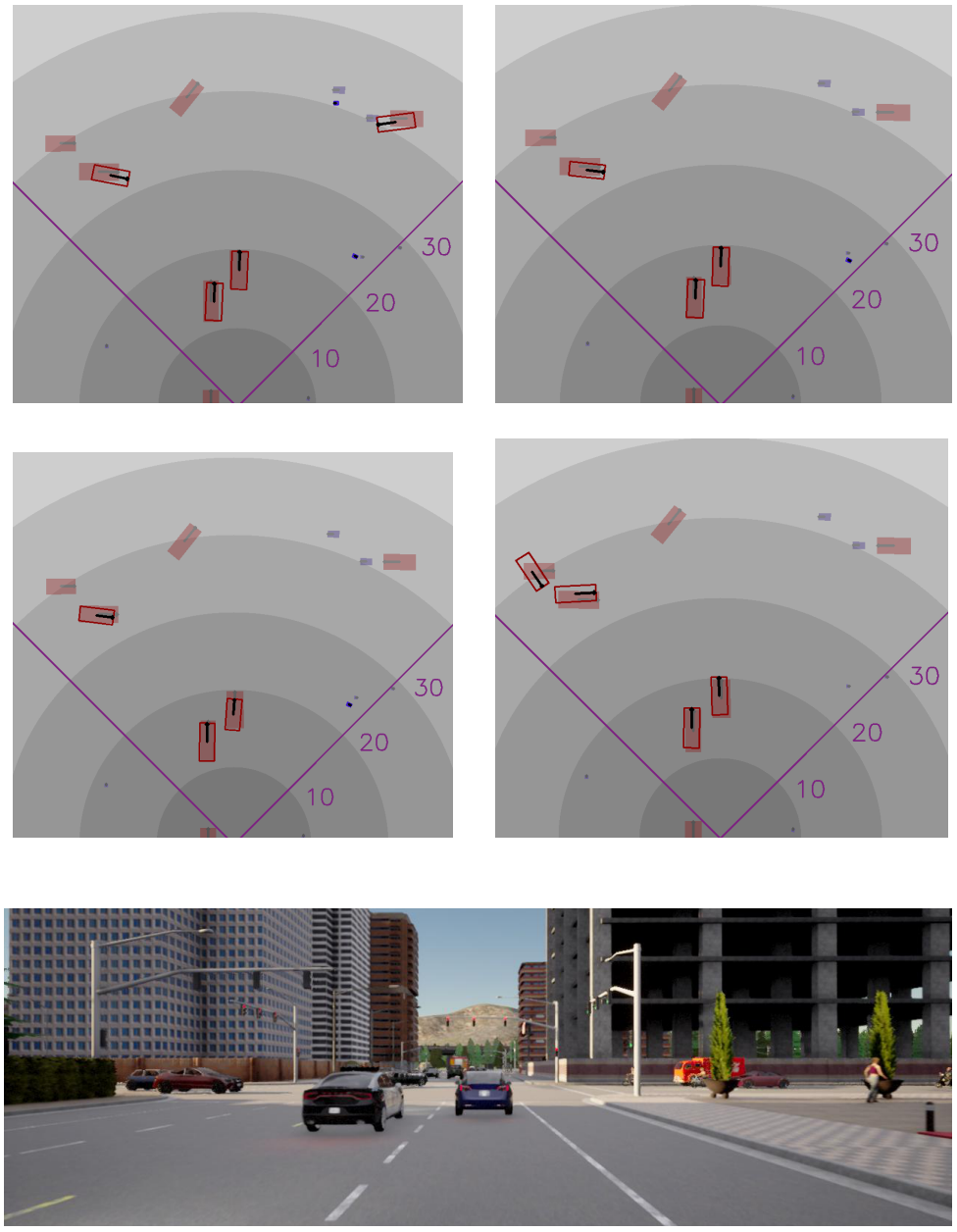

Figura 5.9: Resultado cualitativo de SMOKE (arriba izquierda), PGD (arriba derecha), la propuesta modular con SDN (abajo izquierda) y la propuesta modular con CoEx (abajo derecha) en SHIFT.

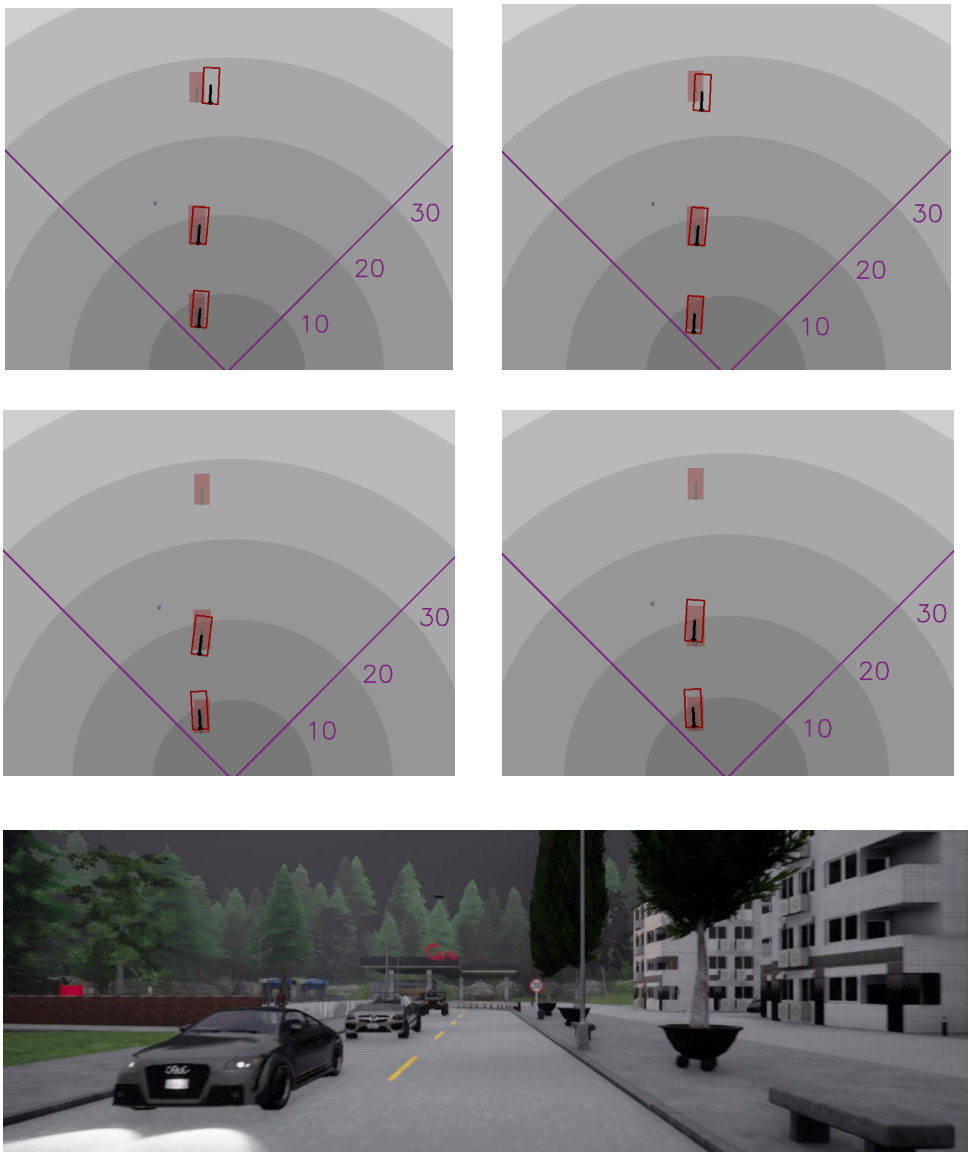

Figura 5.10: Resultado cualitativo de SMOKE (arriba izquierda), PGD (arriba derecha), la propuesta modular con SDN (abajo izquierda) y la propuesta modular con CoEx (abajo derecha) en SHIFT. (2)

Capítulo 6

Conclusiones y líneas futuras

No hay final real. Solamente es el lugar donde detienes la historia.

Frank Herbert

En este capítulo se pretenden recolectar las ideas sobre las que se han estado trabajando en este Trabajo de Fin de Máster, así como las conclusiones que se han obtenido a través de los experimentos y los posibles trabajos que continúen con lo desarrollado en este documento.

6.1 Conclusiones

Este trabajo busca estudiar las ventajas e inconvenientes del uso de diferentes tipos de detectores 3D basados únicamente en cámara, aplicados a un vehículo autónomo. En primer lugar, se ha realizado un estudio del estado del arte de la detección 3D sobre imágenes, detallando varias arquitecturas diferentes y resaltando su rendimiento. Este estudio se ha llevado a cabo teniendo en mente dos tipos diferentes de sistemas: arquitecturas end-to-end, las cuales se encargan de detectar directamente sobre la imagen de entrada en una única etapa, y un sistema modular de detección, que parte de una estimación de profundidad inicial. Con los datos recabados se eligen varios sistemas de cada tipo, priorizando eficiencia y rendimiento. Además, para el caso del sistema modular, se ha partido de una arquitectura del SOTA y se ha modificado, buscando una mayor velocidad en conjunto, tanto en modelos como en el código, así como en unos resultados que permitan al vehículo autónomo conocer el entorno lo suficientemente bien como para operar correctamente.

Los cuatro sistemas objeto de estudio (SMOKE, PGD y nuestra propuesta con SDN y CoEx), se someten a varios entrenamientos sobre datos del dataset real KITTI y del dataset SHIFT, procedente del simulador CARLA. Con las métricas obtenidas se ha conseguido cuantificar el rendimiento de todos ellos sobre un entorno de pruebas común, así como comprobar la capacidad de adaptarse a entornos simulados, los cuales son esenciales en el desarrollo de arquitecturas de vehículos autónomos. Las principales conclusiones son:

- El uso de sistemas con una arquitectura modular lleva a mejores resultados de detección, especialmente en vehículos y objetos de mayor tamaño, a costa de un incremento del tiempo de detección.

- Las redes end-to-end son mucho mas veloces que los sistemas modulares basados en profundidad, lo que las hace mas recomendadas para sistemas en tiempo real y con menos capacidad de cómputo.

- En objetos a corta distancia (hasta 20-25 metros), las calidad de las detecciones es bastante similar entre los dos tipos de sistemas.

- La capacidad de generalización es mayor en el caso de las arquitecturas end-to-end que en las modulares. Concretamente se consiguen buenos resultados en el caso de vehículos, mientras que en objetos mas pequeños como peatones sufren mas por el cambio de entorno.

- La cámara por su naturaleza sufre una gran desventaja respecto a otros sensores como LiDAR o radar a partir de media distancia (35-40 metros).

La Tabla 6.1 ofrece un resumen del rendimiento sobre KITTI de los sistemas estudiados: las arquitecturas end-to-end y nuestra propuesta de arquitectura modular. Con todos los datos recabados se decide que SMOKE es un candidato para la implementación en el vehículo real. Este sistema destaca por tener el tiempo de inferencia mas bajo de todos, lo cual es crucial en tiempo real. En cuanto a los resultados, si bien se encuentra por debajo de sus competidores, aunque a relativamente poca distancia de PGD, se ha comprobado que hasta un rango de 30 metros presenta un rendimiento sólido y confiable.

Este sistema destaca por su rápido tiempo de inferencia, superando a todos los demás en esta métrica crucial. Esto significa que es capaz de tomar decisiones y realizar acciones de manera extremadamente eficiente, lo que puede ser fundamental en aplicaciones de tiempo crítico, como la conducción autónoma. En condiciones normales de operación, este sistema es efectivo para detectar y rastrear elementos en la escena dentro de un rango de distancias razonable, evitando aumentar considerablemente el tiempo de detección para mejorar las detecciones mas lejanas.

	BEV Car	**BEV Pedestrian**	**Inferencia (ms)**
		End-to-end	
SMOKE	17.55	7.42	**55**
PGD	21.88	7.43	90
		Modular	
SDN + PointPillars	**66.97**	13.54	300+ 60
CoEx + PointPillars	57.97	**16.47**	120 + 60

Tabla 6.1: Comparativa final de los modelos.

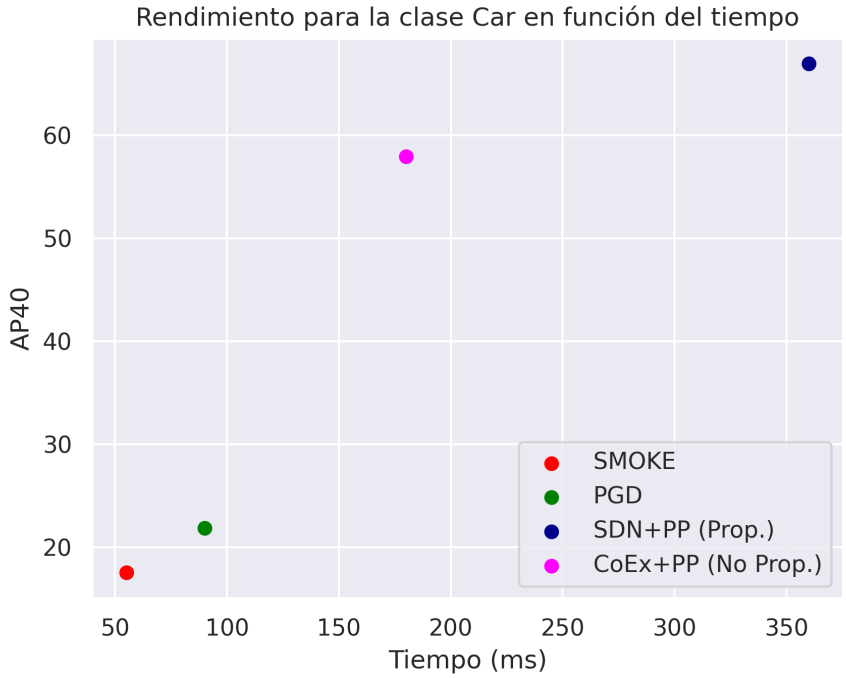

Figura 6.1: Gráfica de rendimiento-tiempo para los sistemas estudiados sobre objetos de la clase *Car*.

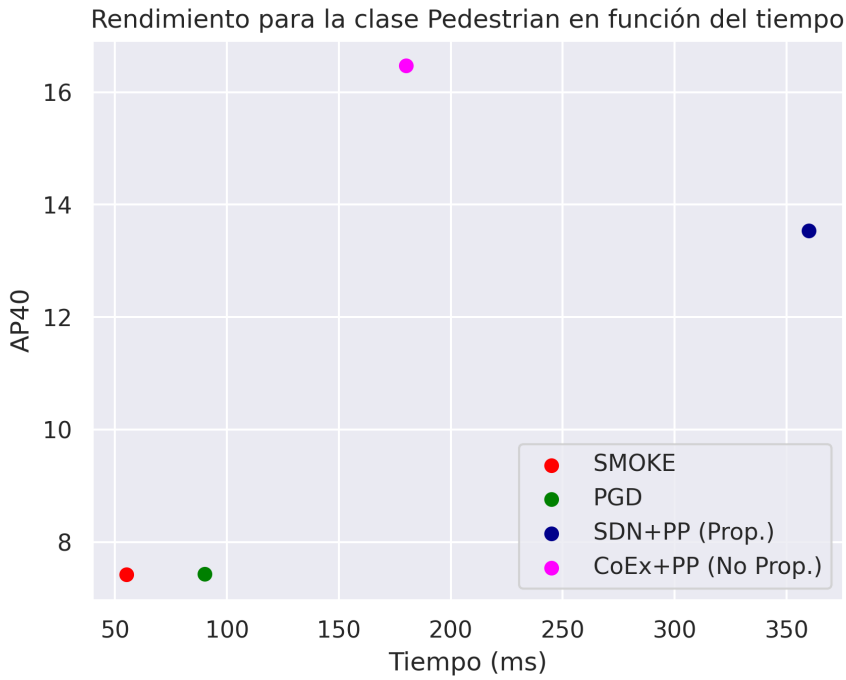

Figura 6.2: Gráfica de rendimiento-tiempo para los sistemas estudiados sobre objetos de la clase *Pedestrian*.

6.2 Líneas futuras

Los datos obtenidos permiten tener una mayor cantidad de información a la hora de tomar una decisión sobre que sistema de detección es recomendable en cada caso. Sin embargo, existen varias líneas de trabajo que pueden ayudar a la implementación en un vehículo autónomo real.

La primera de ellas es la adaptación de los modelos a la arquitectura de conducción autónoma del grupo. Concretamente es necesario implementar uno o varios nodos de ROS2 [53] para poder adquirir los datos de entrada de los sensores y publicar las detecciones, las cuales son utilizadas por otras capas de la arquitectura para la toma de decisiones del vehículo u otras tareas.

Para poder desplegar todo tipo de modelos de DL en sistemas con menos recursos, como puede ser un sistema embebido en un vehículo, han surgido herramientas como TensorRT de NVIDIA, las cuales buscan acelerar arquitecturas simplificando ciertas capas o reduciendo la precisión numérica en la que se almacenan y procesan los datos. Sería interesante intentar adaptar alguno de los modelos mas ligeros a una arquitectura embebida como las NVIDIA Jetson.

Al utilizar datos provenientes de la cámara, los sistemas estudiados se encuentran limitados a detectar únicamente en su campo de visión. Una posible solución a esto sería ampliar hacia un sistema multicámara, el cual aborde mas regiones del entorno del vehículo.

Bibliografía

[1] A. Geiger, P. Lenz y R. Urtasun, "Are we sready for Autonomous Driving? The KITTI Vision Benchmark Suite", en *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez y V. Koltun, "CARLA: An Open Urban Driving Simulator", en *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, págs. 1-16.

[3] T. Sun, M. Segu, J. Postels et al., "SHIFT: A Synthetic Driving Dataset for Continuous Multi-Task Domain Adaptation", en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2022, págs. 21 371-21 382.

[4] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016_202104*. https://www.sae.org/standards/content/j3016_202104/ [Último acceso 14/junio/2023].

[5] *Autonomous driving. 5 steps to the self-driving car*. https://www.bmw.com/en/automotive-life/autonomous-driving.html [Último acceso 14/junio/2023].

[6] N. Dalal y B. Triggs, "Histograms of oriented gradients for human detection", en *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886-893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[7] R. Girshick, J. Donahue, T. Darrell y J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[8] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, 2016. arXiv: 1506.02640 [cs.CV].

[9] A. Vaswani, N. Shazeer, N. Parmar et al., *Attention Is All You Need*, 2017. arXiv: 1706.03762 [cs.CL].

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2021. arXiv: 2010.11929 [cs.CV].

[11] Z. Liu, Z. Wu y R. Tóth, *SMOKE: Single-Stage Monocular 3D Object Detection via Keypoint Estimation*, 2020. arXiv: 2002.10111 [cs.CV].

[12] T. Wang, X. Zhu, J. Pang y D. Lin, *Probabilistic and Geometric Depth: Detecting Objects in Perspective*, 2021. arXiv: 2107.14160 [cs.CV].

[13] Y. You, Y. Wang, W.-L. Chao et al., *Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving*, 2020. arXiv: 1906.06310 [cs.CV].

[14] A. Bangunharcana, J. W. Cho, S. Lee, I. S. Kweon, K.-S. Kim y S. Kim, *Correlate-and-Excite: Real-Time Stereo Matching via Guided Cost Volume Excitation*, 2021. arXiv: 2108.05773 [cs.CV].

[15] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang y O. Beijbom, *PointPillars: Fast Encoders for Object Detection from Point Clouds*, 2019. arXiv: 1812.05784 [cs.LG].

[16] W. Liu, D. Anguelov, D. Erhan et al., "SSD: Single Shot MultiBox Detector", en *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, págs. 21-37. DOI: 10.1007/978-3-319-46448-0_2. dirección: https://doi.org/10.1007%2F978-3-319-46448-0_2.

[17] R. Girshick, *Fast R-CNN*, 2015. arXiv: 1504.08083 [cs.CV].

[18] S. Ren, K. He, R. Girshick y J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2016. arXiv: 1506.01497 [cs.CV].

[19] J. Redmon y A. Farhadi, "YOLOv3: An Incremental Improvement", *arXiv*, 2018.

[20] G. Jocher, *YOLOv5 by Ultralytics*, ver. 7.0, mayo de 2020. DOI: 10.5281/zenodo.3908559. dirección: https://github.com/ultralytics/yolov5.

[21] G. Jocher, A. Chaurasia y J. Qiu, *YOLO by Ultralytics*, ver. 8.0.0, ene. de 2023. dirección: https://github.com/ultralytics/ultralytics.

[22] S. Aharon, Louis-Dupont, Ofri Masad et al., *Super-Gradients*, 2021. DOI: 10.5281/ZENODO.7789328. dirección: https://zenodo.org/record/7789328.

[23] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov y S. Zagoruyko, *End-to-End Object Detection with Transformers*, 2020. arXiv: 2005.12872 [cs.CV].

[24] W. Lv, Y. Zhao, S. Xu et al., *DETRs Beat YOLOs on Real-time Object Detection*, 2023. arXiv: 2304.08069 [cs.CV].

[25] R. Zhang, H. Qiu, T. Wang et al., *MonoDETR: Depth-guided Transformer for Monocular 3D Object Detection*, 2023. arXiv: 2203.13310 [cs.CV].

[26] X. Ma, Y. Zhang, D. Xu et al., *Delving into Localization Errors for Monocular 3D Object Detection*, 2021. arXiv: 2103.16237 [cs.CV].

[27] T. Wang, X. Zhu, J. Pang y D. Lin, *FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection*, 2021. arXiv: 2104.10956 [cs.CV].

[28] D. Rukhovich, A. Vorontsova y A. Konushin, *ImVoxelNet: Image to Voxels Projection for Monocular and Multi-View General-Purpose 3D Object Detection*, 2021. arXiv: 2106.01178 [cs.CV].

[29] K. He, X. Zhang, S. Ren y J. Sun, *Deep Residual Learning for Image Recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[30] X. He, F. Yang, K. Yang et al., *SSD-MonoDETR: Supervised Scale-aware Deformable Transformer for Monocular 3D Object Detection*, 2023. arXiv: 2305.07270 [cs.CV].

[31] F. Yu, D. Wang, E. Shelhamer y T. Darrell, *Deep Layer Aggregation*, 2019. arXiv: 1707.06484 [cs.CV].

[32] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan y S. Belongie, *Feature Pyramid Networks for Object Detection*, 2017. arXiv: 1612.03144 [cs.CV].

[33] X. Zhou, D. Wang y P. Krähenbühl, *Objects as Points*, 2019. arXiv: 1904.07850 [cs.CV].

[34] H. Caesar, V. Bankiti, A. H. Lang et al., "nuScenes: A multimodal dataset for autonomous driving", en *CVPR*, 2020.

[35] X. Song, X. Zhao, L. Fang y H. Hu, *EdgeStereo: An Effective Multi-Task Learning Network for Stereo Matching and Edge Detection*, 2019. arXiv: 1903.01700 [cs.CV].

[36] X. Cheng, Y. Zhong, M. Harandi et al., *Hierarchical Neural Architecture Search for Deep Stereo Matching*, 2020. arXiv: 2010.13501 [cs.CV].

[37] F. Zhang, V. Prisacariu, R. Yang y P. H. S. Torr, *GA-Net: Guided Aggregation Net for End-to-end Stereo Matching*, 2019. arXiv: 1904.06587 [cs.CV].

[38] J.-R. Chang e Y.-S. Chen, *Pyramid Stereo Matching Network*, 2018. arXiv: 1803.08669 [cs.CV].

[39] S. Khamis, S. Fanello, C. Rhemann, A. Kowdle, J. Valentin y S. Izadi, *StereoNet: Guided Hierarchical Refinement for Real-Time Edge-Aware Depth Prediction*, 2018. arXiv: 1807.08865 [cs.CV].

[40] V. Tankovich, C. Häne, Y. Zhang, A. Kowdle, S. Fanello y S. Bouaziz, *HITNet: Hierarchical Iterative Tile Refinement Network for Real-time Stereo Matching*, 2023. arXiv: 2007.12140 [cs.CV].

[41] C. Godard, O. M. Aodha y G. J. Brostow, *Unsupervised Monocular Depth Estimation with Left-Right Consistency*, 2016. arXiv: 1609.03677 [cs.CV].

[42] Y. Wang, Y. Liang, H. Xu, S. Jiao y H. Yu, *SQLdepth: Generalizable Self-Supervised Fine-Structured Monocular Depth Estimation*, 2023. arXiv: 2309.00526 [cs.CV].

[43] S. Shi, X. Wang y H. Li, *PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*, 2019. arXiv: 1812.04244 [cs.CV].

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov y L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, 2019. arXiv: 1801.04381 [cs.CV].

[45] O. Ronneberger, P. Fischer y T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. arXiv: 1505.04597 [cs.CV].

[46] N. Mayer, E. Ilg, P. Häusser et al., "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation", en *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, arXiv:1512.02134, 2016. dirección: http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16.

[47] *What Is CUDA | NVIDIA Official Blog*. dirección: https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/.

[48] A. Paszke, S. Gross, F. Massa et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library", en *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, págs. 8024-8035. dirección: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[49] Martín Abadi, Ashish Agarwal, Paul Barham et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. dirección: https://www.tensorflow.org/.

[50] *Docker: Accelerated, Containerized Application Development*. dirección: https://www.docker.com/.

[51] MMDetection3D Contributors, *OpenMMLab's Next-generation Platform for General 3D Object Detection*, jul. de 2020. dirección: https://github.com/open-mmlab/mmdetection3d.

[52] Mathworks, *RoadRunner - MATLAB*. dirección: https://es.mathworks.com/products/roadrunner.html.

[53] S. Macenski, T. Foote, B. Gerkey, C. Lalancette y W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild", *Science Robotics*, vol. 7, n.º 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. dirección: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

Apéndice A

Configuración MMDetection3D

Se adjuntan dos ejemplos de configuración de los modelos usados de MMDetection3D.

A.1 SMOKE

```
_base_ = [
    '../_base_/datasets/kitti-mono3d.py', '../_base_/models/smoke.py',
    '../_base_/default_runtime.py'
]

backend_args = None

train_pipeline = [
    dict(type='LoadImageFromFileMono3D', backend_args=backend_args),
    dict(
        type='LoadAnnotations3D',
        with_bbox=True,
        with_label=True,
        with_attr_label=False,
        with_bbox_3d=True,
        with_label_3d=True,
        with_bbox_depth=True),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(type='RandomShiftScale', shift_scale=(0.2, 0.4), aug_prob=0.3),
    dict(type='AffineResize', img_scale=(1280, 384), down_ratio=4),
    dict(
        type='Pack3DDetInputs',
        keys=[
            'img', 'gt_bboxes', 'gt_bboxes_labels', 'gt_bboxes_3d',
            'gt_labels_3d', 'centers_2d', 'depths'
        ]),
]
test_pipeline = [
    dict(type='LoadImageFromFileMono3D', backend_args=backend_args),
```

```
        dict(type='AffineResize', img_scale=(1280, 384), down_ratio=4),
        dict(type='Pack3DDetInputs', keys=['img'])
    ]

    train_dataloader = dict(
        batch_size=8, num_workers=4, dataset=dict(pipeline=train_pipeline))
    test_dataloader = dict(dataset=dict(pipeline=test_pipeline))
    val_dataloader = dict(dataset=dict(pipeline=test_pipeline))

    # training schedule for 6x
    max_epochs = 100
    train_cfg = dict(
        type='EpochBasedTrainLoop', max_epochs=max_epochs, val_interval=5)
    val_cfg = dict(type='ValLoop')
    test_cfg = dict(type='TestLoop')

    # learning rate
    param_scheduler = [
        dict(
            type='MultiStepLR',
            begin=0,
            end=max_epochs,
            by_epoch=True,
            milestones=[50],
            gamma=0.1)
    ]

    # optimizer
    optim_wrapper = dict(
        type='OptimWrapper',
        optimizer=dict(type='Adam', lr=2.5e-4),
        clip_grad=None)

    find_unused_parameters = True
```

A.2 PGD

```

_base_ = [
    '../_base_/datasets/kitti-mono3d.py', '../_base_/models/pgd.py',
    '../_base_/schedules/mmdet-schedule-1x.py', '../_base_/default_runtime.py'
]
# model settings
model = dict(
    data_preprocessor=dict(
        type='Det3DDataPreprocessor',
        mean=[103.530, 116.280, 123.675],
        std=[1.0, 1.0, 1.0],
        bgr_to_rgb=False,
        pad_size_divisor=32),
    backbone=dict(frozen_stages=0),
    neck=dict(start_level=0, num_outs=4),
    bbox_head=dict(
        num_classes=3,
        bbox_code_size=7,
        pred_attrs=False,
        pred_velo=False,
        pred_bbox2d=True,
        use_onlyreg_proj=True,
        strides=(4, 8, 16, 32),
        regress_ranges=((-1, 64), (64, 128), (128, 256), (256, 1e8)),
        group_reg_dims=(2, 1, 3, 1, 16,
                        4),  # offset, depth, size, rot, kpts, bbox2d
        reg_branch=(
            (256, ),  # offset
            (256, ),  # depth
            (256, ),  # size
            (256, ),  # rot
            (256, ),  # kpts
            (256, )  # bbox2d
        ),
        centerness_branch=(256, ),
        loss_cls=dict(
            type='mmdet.FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=1.0),
        loss_bbox=dict(
            type='mmdet.SmoothL1Loss', beta=1.0 / 9.0, loss_weight=1.0),
        loss_dir=dict(
            type='mmdet.CrossEntropyLoss', use_sigmoid=False, loss_weight=1.0),
        loss_centerness=dict(
```

```
            type='mmdet.CrossEntropyLoss', use_sigmoid=True, loss_weight=1.0),
        use_depth_classifier=True,
        depth_branch=(256, ),
        depth_range=(0, 70),
        depth_unit=10,
        division='uniform',
        depth_bins=8,
        pred_keypoints=True,
        weight_dim=1,
        loss_depth=dict(
            type='UncertainSmoothL1Loss', alpha=1.0, beta=3.0,
            loss_weight=1.0),
        bbox_coder=dict(
            type='PGDBBoxCoder',
            base_depths=((28.01, 16.32), ),
            base_dims=((0.8, 1.73, 0.6), (1.76, 1.73, 0.6), (3.9, 1.56, 1.6)),
            code_size=7)),

    train_cfg=dict(code_weight=[
        1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2,
        0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 1.0, 1.0, 1.0, 1.0
    ]),
    test_cfg=dict(nms_pre=100, nms_thr=0.05, score_thr=0.001, max_per_img=20))

backend_args = None

train_pipeline = [
    dict(type='LoadImageFromFileMono3D', backend_args=backend_args),
    dict(
        type='LoadAnnotations3D',
        with_bbox=True,
        with_label=True,
        with_attr_label=False,
        with_bbox_3d=True,
        with_label_3d=True,
        with_bbox_depth=True),
    dict(type='mmdet.Resize', scale=(1242, 375), keep_ratio=True),
    dict(type='RandomFlip3D', flip_ratio_bev_horizontal=0.5),
    dict(
        type='Pack3DDetInputs',
        keys=[
            'img', 'gt_bboxes', 'gt_bboxes_labels', 'gt_bboxes_3d',
            'gt_labels_3d', 'centers_2d', 'depths'
        ]),
]
test_pipeline = [
    dict(type='LoadImageFromFileMono3D', backend_args=backend_args),
    dict(type='mmdet.Resize', scale_factor=1.0),
```

```
95      dict(type='Pack3DDetInputs', keys=['img'])
96  ]
97
98  train_dataloader = dict(
99      batch_size=3, num_workers=3, dataset=dict(pipeline=train_pipeline))
100 test_dataloader = dict(dataset=dict(pipeline=test_pipeline))
101 val_dataloader = dict(dataset=dict(pipeline=test_pipeline))
102
103 # optimizer
104 optim_wrapper = dict(
105     optimizer=dict(lr=0.001),
106     paramwise_cfg=dict(bias_lr_mult=2., bias_decay_mult=0.),
107     clip_grad=dict(max_norm=35, norm_type=2))
108
109 # learning rate
110 param_scheduler = [
111     dict(
112         type='LinearLR',
113         start_factor=1.0 / 3,
114         by_epoch=False,
115         begin=0,
116         end=500),
117     dict(
118         type='MultiStepLR',
119         begin=0,
120         end=48,
121         by_epoch=True,
122         milestones=[32, 44],
123         gamma=0.1)
124 ]
125
126 train_cfg = dict(max_epochs=100, val_interval=2)
127 auto_scale_lr = dict(base_batch_size=12)
```

Apéndice B

Herramientas y recursos

Las el software y hardware necesarios para la elaboración del proyecto han sido:

- PC para desarrollo y entrenamiento con:
  - GPU NVIDIA GTX 1080 Ti.
  - Intel Core i7-8700.
  - 32 GB de RAM DDR4
- Sistema operativo Ubuntu 20.
- Entorno de desarrollo Visual Studio Code.
- Python 3, destacando librerías como PyTorch y MMDetection3D para manejo de modelos y matplotlib, opencv o mayavi para procesado y representación de escenas.
- Docker.
- GitHib.
- KITTI dataset [1].
- SHIFT dataset [3].
- LaTeXy Overleaf.

Apéndice C

Presupuesto

Los costes del proyecto se pueden dividir en dos apartados: coste de equipo y coste de personal.

Respecto al coste de equipo, el proyecto se ha desarrollado por completo en un ordenador de sobremesa del grupo de investigación RobeSafe, cuyas especificaciones y costes se reflejan en la Tabla C.1.

El software necesario no aporta coste al proyecto ya que todos los elementos son de libre acceso o código abierto.

Concepto	**Coste (€)**
NVIDIA GTX 1080 Ti	450
Intel Core i7-8700	300
32 GB RAM	150
Otros componentes	200
Total	**1100**

Tabla C.1: Precio actualizado a 2023 de componentes utilizados.

El coste de personal se calcula en función de las horas dedicadas a las diferentes tareas, cuyo desglose es el de la Tabla C.2. Se aplica un salario de 15 € por hora, con una media de 6 horas por día.

Tareas	Días	Coste (€)
Estudio del estado del arte	60	5400
Creación y configuración del entorno de trabajo en Docker	10	900
Adaptación del dataset SHIFT	15	1350
Configuración y adaptación de MMDetection3D	20	1800
Entrenamiento y evaluación de redes end-to-end	30	2700
Entrenamiento y evaluación de redes de estimación de profundidad	30	2700
Implementación del sistema modular de detección	15	1350
Entrenamiento y evaluación del detector de nubes de puntos	15	1350
Evaluación del sistema modular de detección	15	1350
Recopilación y comparativa de resultados	15	1350
Escritura de la memoria	30	2700
Total	255	**22950**

Tabla C.2: Listado de tareas del trabajo.

En total, los costes ascienden a 24050 €, a lo que hay que aplicar el IVA fijado al 21 %, lo que eleva la cifra a **29100,5 €**.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá